Risk Architecture

Part 2 — Archetypes, Patterns and the Severity Layer

Version 49.25 · 11 November 2026

Reconstructed from the code, gate by gate.

Part 1 documents Layer 1 — the individual measures, each scoring risk points against its own maximum. This documents **Layer 2**: the archetypes, patterns and severity conditions that score combinations no single ratio captures, and the machinery that assembles them.

Every entry was rebuilt by reading the gate that implements it. Firing rates are counted from the validated 2026 baseline (1,401 scored institutions), so each entry states not only what the gate tests but how often it actually fires.

STALE FIRING RATES — read before quoting any number in this document (revised 28 October 2026, v49.25). Every firing rate and baseline count below was counted on the validated 2026 baseline (1,401 scored institutions), before 3 September 2026. The inputs the gates read have changed many times since: the 3 September data repairs (DPP FTE for 1,009 institutions, Debt-to-Revenue rebuilt on the level of leverage, the Primary Reserve fallback and floor); the v49.8 pension-adjusted basis for public balance-sheet measures; the v49.12 private margin series; the v49.14 private admissions and salaries scoring; and v49.20, which makes Instructional Support informational. Several gates were also changed or repaired between v49.10 and v49.19 (see "Changes from v49.10 to v49.20" in §3). **No rate in this document may be quoted.** Re-baselining them needs a national run at v49.25 (both sectors, Carnegie 15–22). Until then, quote a rate only from a named run and build. Where an entry carries a dated v49.1x or v49.20 note, that note states the current gate and governs any older text below it. Line references are approximate: most predate v49.10, and the file has grown since.

The rule is the same as Part 1: **this document describes what the tool does.** Where a gate behaves in a way that looks unintended, the behavior is documented and the concern is recorded rather than quietly corrected in the prose. Layer 2 needs that discipline more than Layer 1 did — this reading found four elements that cannot fire at all, and the reasons are in their entries. Two of the four, A8 and P9, were repaired in October 2026 and now fire; their entries keep the causes as history.

Contents

1. What Layer 2 does
2. How the pieces combine
3. The catalog
   - A1 — High Vulnerability — Small Scale, High Dependency
   - A2 — Leverage-Led Growth Bet
   - A3 — Academic Sprawl Trap
   - A4 — Monoculture Dependency
   - A5 — High-Cost Architecture, Stalling Demand
   - A6 — Brand-Weakening Channel
   - A7 — High-Pell "Mission Pressure Cooker"
   - A8 — Brand-Strong but Margin-Thin
   - A9 — The Squeezed Middle
   - A10 — Endowment-Masked Operating Deficit
   - A12 — Broad Balance-Sheet Failure
   - A14 — Demand Collapse with Intact Reserves
   - R10 — Resilient by Scale and Diversification
   - R11 — Research-Strong Anchors
   - R12 — Brand-Strong Anchors
   - P1 — Siena Pattern
   - P2 — High Discount, Flat Enrollment
   - P3 — Appropriations Dependence with Enrollment Decline
   - P4 — Net Tuition Risk
   - P5 — Unrestricted Net Assets Risk
   - P6 — Research Coverage Erosion
   - P7 — Brand Positioning
   - P8 — Market Position
   - P9 — Endowment Drain Risk
   - U38 and U39 — unscored flags (v49.17)
4. The severity layer
5. Shared machinery and known defects

1. What Layer 2 does

Layer 1 asks 34 separate questions and adds up the answers. That misses anything whose meaning lies in the *combination* — an institution with a thin margin and falling enrollment and rising discount is in a different position from one with any of those alone, and no single ratio says so.

Layer 2 is the set of gates that read across Layer 1's output and score those combinations. It contributes up to 50 points.

The three kinds of element

Archetypes (A-series) name a recognizable institutional situation — the leveraged growth bet, the squeezed middle, the endowment-masked deficit. Each fires or does not, and adds its declared points.

Patterns (P-series) name a mechanism rather than a situation, usually a relationship between an institution and its peers: discounting faster than its peer group, sitting below peer net tuition, losing market position. Several escalate — a base award, plus more if a second condition also holds.

Resilience elements (R-series) subtract. They are the only credits in the framework, and there are three of them against twenty-one charges.

Alongside these sit three **severity conditions** — financial distress, operational inefficiency and the enrollment cliff — which are not archetypes but behave like them, and additionally feed the severity override described in §4.

The asymmetry worth knowing about

Layer 2 enters the numerator but never the denominator. Up to 50 points are added to an institution's risk total, while every denominator loop in the orchestrator skips the risk-adjustment analyzer. On a framework maximum of around 200 points, a full 50-point Layer 2 contribution moves the composite by roughly 25 percentage points — more than the distance between two bands. This is deliberate in the sense that it is consistent, but it means Layer 2's calibration is doing more work than its point values suggest.

There is a cap but no floor. `calculateTotalAdjustmentPoints()` caps the total at 50. Nothing stops it going negative: R10 (−8) and R12 (−6) stack to −14 with no double-count guard, and in the 2026 baseline 264 institutions carry a negative Layer 2 contribution, with the observed minimum at exactly −14. No composite went negative because Layer 1 points always exceeded the credit, but nothing in the code guarantees that.

2. How the pieces combine

The assembly is short and it matters, because it is not a sum.

1. Every gate is evaluated. Archetypes, patterns and resilience elements each return `{triggered, points, reason}`. 2. `calculateTotalAdjustmentPoints()` sums the points of every element that triggered — including the negative ones. 3. `checkSeverityCondition()` counts how many severity triggers fired. If it clears its bar, a **severity floor** is computed from the pattern of triggers. 4. The two are combined as `Math.min(50, Math.max(severityPoints, archetypePatternTotal))`.

Step 4 is a maximum, not a sum, and this is the single most consequential line in Layer 2. Where the override applies, the summed archetype and pattern total is *discarded* and replaced by the severity floor. An institution can gain an archetype and see its composite not move at all — which is exactly what happened when A6 was rebuilt in v49.1: A6's firing rate rose 15.6 points and the override rate did not move, because the additional points were absorbed.

It also means **resilience credits are discarded whole** whenever the override applies. An institution that earns −14 in credits and then trips the override keeps none of it.

What the override is

The severity override exists to stop a broad set of moderate problems reading as less serious than a single acute one. It is reported in exports as "A11 Severity Pattern", though no A11 gate exists — the name is a label in a reason string. §4 documents it in full.

3. The catalog

Twenty-four elements. Firing rates were counted over the 1,401 scored institutions in the validated 2026 baseline, which was superseded on 3 September 2026 — see the warning at the head of this document. Where an element fires for no institution, that is stated. Since then A8 and P9 have been repaired and fire (v49.15, v49.11), and A7 no longer fires at community colleges (v49.19). Their rows are marked.

Element	Points	Fires for	Of scored population
A1 High Vulnerability — Small Scale, High Dependency	4	9	0.6%
A2 Leverage-Led Growth Bet	4	306	21.8%
A3 Academic Sprawl Trap	1	83	5.9%
A4 Monoculture Dependency	1	94	6.7%
A5 High-Cost Architecture, Stalling Demand	3	205	14.6%
A6 Brand-Weakening Channel	2	431	30.8%
A7 High-Pell "Mission Pressure Cooker"	3 (0 at community colleges, v49.19)	395	28.2%
A8 Brand-Strong but Margin-Thin	2	never before v49.15; fires since (not re-baselined)	—
A9 The Squeezed Middle	2	224	16.0%
A10 Endowment-Masked Operating Deficit	6	87	6.2%
A12 Broad Balance-Sheet Failure	6	121	8.6%
A14 Demand Collapse with Intact Reserves	8	82	5.9%
R10 Resilient by Scale and Diversification	-8	285	20.3%
R11 Research-Strong Anchors	-4	never	—
R12 Brand-Strong Anchors	-6	289	20.6%
P1 Siena Pattern	5	127	9.1%
P2 High Discount, Flat Enrollment	4 → 6	30 (old test; 68 of 1,000 privates at v49.25)	2.1%
P3 Appropriations Dependence with Enrollment Decline	5	122	8.7%
P4 Net Tuition Risk	5 → 7	485	34.6%
P5 Unrestricted Net Assets Risk	6	278	19.8%
P6 Research Coverage Erosion	—	never	—
P7 Brand Positioning	6 → 8	76	5.4%
P8 Market Position	5 → 7	434	31.0%
P9 Endowment Drain Risk	4	never before v49.11; fires since (not re-baselined)	—

Four elements never fired in the baseline, for four different reasons, and two of them have since been repaired. A8 and P9 were evaluated for every institution and could not pass. P9 was repaired in v49.11 (15 October 2026) and A8 in v49.15 (19 October 2026); both now fire, and their entries keep the old causes as history. R11 is switched off at source by a single hard-coded constant, deliberately, pending a replacement measure. P6 does not exist: it was removed on 2026-08-28 and only its display label survives.

Before the repairs, A8 and P9 accounted for 6 points of risk the framework documented and never charged (A8's 2 and P9's 4). This table previously gave P9 6 points; the gate has always awarded +4. R11's 4 points of credit are still never given. Each element is documented in its own entry rather than omitted, because a reference that quietly drops the elements that do not work tells its reader the framework is smaller and tidier than it is.

A13 Critical Mass Escalation is not in the table. It was repriced to four points on 2026-08-27 and retained as a diagnostic; it is documented in §4 with the severity machinery.

Changes from v49.10 to v49.20 (19 October 2026)

Layer 2 changed in seven releases between 15 and 19 October 2026. Each was predicted on named anchors before it was applied and validated on tool output after; the README (§3.12–§3.23) holds the record. In summary:

- **v49.10 (15 October) — gate status prints its own reason.** Non-scoring. Every untriggered gate now prints why it did not fire. Gates that read outside the metric ledger are stamped `unledgered` (evaluated, did not trigger) or `evaluated_unledgered` (triggered), where they used to read `not_reached — no detail`. Neither status means the gate was skipped. A10's untriggered reason prints its four readings. R10 and financial distress read appropriations dependency only for publics.
- **v49.11 (15 October) — P9 can fire.** Scoring. Gate 3 now reads endowment per FTE and its peer median from `analysisResults`. All 20 predicted privates fire at +4. When A10 fires, P9 defers to it.
- **v49.12 and v49.13 (18–19 October) — private margin gates.** Scoring, privates only. Six private margin tests now read the three-year average of the v2 operating margin, with cutoffs set to keep the share of privates each selects (`PRIVATE_MARGIN_GATES`): financial distress weak margin and the severity override margin trigger at or below −2.7%; the distress research clause and P5's weak margin at or below −1.5%; A5's weak margin at or below −5.3%; A9's still-coping test above −3.6%. v49.12 had pointed them briefly at the structural margin with endowment gifts removed (`PRIVATE_STRUCT_GATES_V4912`, no longer read). A10, P9 and the unscored flags still read the structural margin, which is the right input for reserve dependence. Publics and community colleges are unchanged.
- **v49.14 (19 October) — R12 margin gate.** Scoring, privates only. R12 also requires the v2 operating margin's three-year average above −2.7%.
- **v49.15 (19 October) — A8 reads fixed.** Scoring. Exact-key reads, the net tuition `percentChange` read, and an admissions guard. Conditions and the +2 are as originally specified.
- **v49.16 (19 October) — the deferred P9 verdict.** Non-scoring. When A10 fires, P9 still scores 0 but evaluates its gates and prints whether it would have fired. Archetype reasons for A1, A2, A3, A5, A9 and R10 print percentages to one decimal and dollars per FTE to the dollar.
- **v49.17 (19 October) — unscored flags U38 and U39.** Non-scoring. See their entry after P9.
- **v49.19 (19 October) — A7 retired for community colleges.** Scoring, sector 4 only.
- **Layer 1 changes that move Layer 2 inputs** (documented in Part 1): the v49.12 private margin series; the v49.14 private admissions and salaries scoring, which changes what A8's and R12's admissions tests select; v49.18, after which the restriction profile, endowment draw and endowment dependency honor the peer frame (no score change); and v49.20, which makes Instructional Support informational in every sector. Operational inefficiency reads the instructional share, not the measure's points, so it is unaffected.
- **Dependence on the v2 file.** If `annualized adjusted operating margin pr v2.xlsx` is not loaded, the private adjusted margin falls back to the legacy total margin and its finding prints `Basis: LEGACY`. The private cutoffs above are then not valid for that run.

Changes from v49.21 to v49.25 (19–28 October 2026)

One Layer 2 change, P2, in v49.25. The README (§3.25–§3.29) holds the record.

- v49.21 to v49.23 (19–25 October) — no Layer 2 logic changed. v49.21 fixed the institution roster; v49.22 added header lines to the individual export; v49.23 repaired double-encoded characters (≥ ≤ ↓ –) in the A1 and A7 labels, the financial-distress reason, the P8 signals and the severity lines. No score moved.
- v49.24 (27 October) — first-year discount analyzer. Non-scoring. A new unscored measure (Part 1) reports the IPEDS first-year discount, falls 2010–2023. It is the input P2 now reads.
- v49.25 (28 October) — P2 rebuilt on the measured discount. Scoring, privates only. See the P2 entry. In the private Carnegie 15–22 run P2 fires at 68 (was 35); 87 of 1,000 privates changed score and 9 changed band. Validated on the 27 anchors.

A1 — High Vulnerability — Small Scale, High Dependency

Points 4 · fires for 9 of 1,401 (0.6%)

- **Method**: `checkA1_SmallScaleHighDependency()` (lines 1469–1561)
- **Points**: `A1: 4` — line 180: `A1: 4, A2: 4, A3: 1, A4: 1, A5: 3, A6: 2, A7: 3, A8: 2, A9: 2,A10: 6,` Applied as `const points = triggered ? (ARCHETYPE_BASE_POINTS?.A1 ?? 4) : 0;` (line 1554).
- **What it tests**: a small institution (under 1,000 students) that depends on a single revenue source above a sector-specific bar **and** has a thin cushion — measured either as unrestricted net assets per FTE, or as a high risk score on primary reserve or viability. The in-code comments describe the thresholds only as "your original A1 cutoff" (line 1543) and "your original dep rule for A1" (line 1544); no rationale is recorded.
- **Inputs**:

variable	analyzer (key)	field(s) attempted, in order	route
`enrollNow`	enrollment (`enrollment`, resolved from canonical `enrollment_trends`)	`currentTotal`, `total`, `current`, `components.currentTotal`, `components.currentValue`, `analysisResults.currentTotal`, `analysisResults.total`, `analysisResults.current`, `analysisResults.components.currentTotal`, `analysisResults.components.currentValue`; then regex `/(current(?!.*year)|total)/i`	`makeGetter` over a locally built `analysesMap`
`depPct`	tuition dependency (`tuitiondependency`)	`currentValue`, `value`, `dependency`, `dependencyPercent`, `components.currentValue`, `analysisResults.currentValue`, `analysisResults.value`, `analysisResults.dependency`, `analysisResults.dependencyPercent`, `analysisResults.tuition_percentage`, `analysisResults.tuition_dependency`, `analysisResults.components.currentValue`; then regex `/(dependency|tuition.*percent|current)/i`	`makeGetter`, wrapped in `toPct()`
`apprDepPct`	appropriations (`appropriations`)	`currentValue`, `value`, `dependency`, `dependencyPercent`, `components.currentValue`, `analysisResults.currentValue`, `analysisResults.value`, `analysisResults.dependency`, `analysisResults.dependencyPercent`, `analysisResults.components.currentValue`; then regex `/(dependency|current)/i`	`makeGetter`, wrapped in `toPct()`
`prRisk`	primary reserve (`primaryreserve`)	`analysisResults.riskPoints`, `riskPoints`	`makeGetter`
`viRisk`	viability (`viability`)	`analysisResults.riskPoints`, `riskPoints`	`makeGetter`
`unaFTE`	unrestricted net assets (`unrestrictednetassets`)	`resolveUnaPerFTE()` first; then `total`, `currentTotal`, `una_total`, `unrestricted_net_assets_total`, `components.total`, `components.currentTotal`, `analysisResults.total`, `analysisResults.currentTotal`, `analysisResults.una_total` (÷ `enrollNow`); then `scanNumericDeep(record, /(una.*fte|per.*fte|una)/i)`	helper method, then `makeGetter`, then deep numeric scan

Publisher check: tuition dependency publishes `currentValue` as a **ratio 0–1** (`unified_tuition_dependency_analyzer.js` line 132, `valueScale: 1` at line 56); appropriations publishes `currentValue` as a **percent 0–100** (`unified_appropriations_analyzer.js` line 127). `toPct()` reconciles the two scales. `analysisResults.tuition_percentage`, `analysisResults.tuition_dependency`, `una_total` and `unrestricted_net_assets_total` **are not published by their named producers** — inert links, since earlier links resolve.

- **Gate logic**:

```
1543|  const smallCut = 1000;                        // your original A1 cutoff
1544|  const depCut   = (segKey === 'public') ? 70  : 85;  // your original dep rule for A1
```

```
1545|  const thinOK =
1546|    (isNum(unaFTE) && unaFTE < 20000) ||
1547|    (isNum(prRisk) && prRisk >= 6)    ||
1548|    (isNum(viRisk) && viRisk >= 6);
1549|
1550|  const depOK = isNum(depAnyPct) && depAnyPct >= depCut;
1551|  const sizeOK = isNum(enrollNow) && enrollNow < smallCut;
1552|
1553|  const triggered = !!(sizeOK && depOK && thinOK);
```

Fires when `sizeOK AND depOK AND thinOK`.

condition	quantity	test	kind
`sizeOK`	current total enrollment	`< 1000`	**ceiling** (strict)
`depOK` (public)	appropriations dependency, %	`>= 70`	**floor**
`depOK` (all other segKeys)	tuition dependency, %	`>= 85`	**floor**
`thinOK` leg 1	UNA per FTE, dollars	`< 20000`	**ceiling** (strict)
`thinOK` leg 2	primary reserve risk points (max 10)	`>= 6`	**floor**
`thinOK` leg 3	viability risk points (max 8)	`>= 6`	**floor**

- **Sector differences**: `segKey` is derived at lines 1471–1474:

```
1471|  const segKey =
1472|    (this.sector === 2 || this.sector === '2') ? 'private' :
1473|    (this.sector === 1 || this.sector === '1') ? 'public'  :
1474|    String(this.sector ?? '');
```

Only `'1'` yields `'public'` and only `'2'` yields `'private'`. The dependency source is chosen at lines 1513–1515:

```
1513|  const depAnyPct = (segKey === 'public')
1514|    ? (isNum(apprDepPct) ? apprDepPct : depPct)
1515|    : (isNum(depPct)     ? depPct     : apprDepPct);
```

So publics prefer appropriations dependency and fall back to tuition dependency; everyone else prefers tuition dependency and falls back to appropriations. `depCut` is 70 for `segKey === 'public'`, 85 otherwise — **the bar switches on segKey, not on which dependency series was actually used**, so a public whose appropriations dependency is missing is judged on *tuition* dependency against the 70 floor, and a sector-3 or sector-4 institution using appropriations dependency is judged against the 85 floor.

- **Data-gap behavior**: **fails closed on all three legs.** `isNum()` (line 125) requires a finite number, so a `null`/`undefined`/string input makes `sizeOK`, `depOK` or the corresponding `thinOK` leg false. `thinOK` is a three-way OR, so it survives the loss of any two of its inputs; `sizeOK` and `depOK` each have a single input and a miss on either suppresses the gate outright. The exception is the regex fallback inside `makeGetter`, which can substitute a wrong-quantity number rather than returning `undefined` — see **Notes**.
- **2026-08 changes**: none dated inside this method. The only dated design-intent comments reachable from A1 are on its helpers: `resolveUnaPerFTE()`'s header (lines 699–701, quoted above) and the `_getMetricValueRaw` step-5 fail-closed note (quoted under A5).
- **Notes**:
  - **Fail-open risk in the enrollment lookup.** If every explicit path for `enrollNow` returns `null`/`undefined` — which is exactly the data-gap case, since the enrollment analyzer publishes `currentTotal: null` when the institution is not found (`unified_enrollment_analyzer.js` line 324) — `makeGetter` falls to the regex `/(current(?!.*year)|total)/i` and scans key *names* one level deep. On the orchestrator record the first matching key with a numeric value is `peerMedianTotal` (the peer group's median headcount), and failing that `peerMedianTotalCount` (a **count of peers**), and failing that `totalChangePercent` (a **percentage**). Any of the three would be tested as `< 1000` and would pass. The same shape applies to `depPct`/`apprDepPct`, whose regexes (`/(dependency|tuition.*percent|current)/i`, `/(dependency|current)/i`) will match any key containing `current`. This is the class of defect the step-5 fail-closed change was made to eliminate in `getMetricValue`, but `makeGetter` is a separate resolver and was not changed.
  - `smallCut = 1000` is an absolute headcount with no sector or Carnegie adjustment, and `unaFTE < 20000` is an absolute dollar figure with no inflation base year recorded. **UNCLEAR**: the vintage of the $20,000 figure. Nothing in the file records when it was set or against what distribution.

- Community colleges (`sector === '4'`) and private for-profits (`sector === '3'`) fall into the `else` branch of `segKey` and are therefore treated with the **private** dependency rule (tuition first, 85% floor) — for CCs, despite their being appropriations-funded and despite the appropriations analyzer being loaded for them (`analysis_orchestrator.js` lines 123–128). Behavior, not a stated intent.
- `toPct()` multiplies any value `<= 1` by 100. Appropriations dependency is published in percent, so a genuine appropriations dependency of 0.8% is read as 80%. Rare, but it moves an institution from far below the floor to just below it.
- The method emits an unconditional `console.log('[A1 DEBUG]', …)` (line 1559) on every institution. In a ranking run this is per-institution console traffic that the `__ratQuiet()` guard used elsewhere in the pipeline does not suppress.

A2 — Leverage-Led Growth Bet

Points 4 · fires for 306 of 1,401 (21.8%)

- **Method**: `checkA2_LeverageLedGrowth()` (lines 1565–1583)
- **Points**: `A2: 4` — line 180. Applied as `(ARCHETYPE_BASE_POINTS?.A2 ?? 4)` (line 1580).
- **What it tests**: an institution carrying heavy debt relative to revenue whose enrollment is not growing — i.e. leverage taken on for growth that did not arrive. Enrollment weakness may be shown by either the long-run or the five-year series.
- **Inputs**:

variable	analyzer (key)	field	route
`debtRiskRP`	debt-to-revenue (`debtrevenue`)	`analysisResults.riskPoints`	direct `completedAnalyses.find(a => a.analyzerKey === 'debtrevenue')`, then `this.num()`
`e10`	enrollment (`enrollment`)	`analysisResults.totalChangePercent`	direct `completedAnalyses.find(a => a.analyzerKey === 'enrollment')`, then `this.num()`
`e5`	enrollment (`enrollment`)	`shortTerm.totalChangePercent`	`this.getEnroll5()` → `getMetricValue('enrollment','5_year_change')`

All three fields are published by their producers (`simple_debt_revenue_analyzer.js` line 93; `unified_enrollment_analyzer.js` lines 123–124).

- **Gate logic**:

```
1578|  const enrollWeak = (e10 != null && e10 <= 0) || (e5 != null && e5 <= 0);
1579|  const triggered = (debtRiskRP != null && debtRiskRP >= 6) && enrollWeak;
```

condition	quantity	test	kind
leverage	debt-to-revenue risk points	`>= 6`	**floor**
`enrollWeak` leg 1	`e10`, 2010→2024 total enrollment change, %	`<= 0`	**ceiling**
`enrollWeak` leg 2	`e5`, five-year total enrollment change, %	`<= 0`	**ceiling**

Note `debtRiskRP >= 6` is 60% of the private/FP/CC maximum (10) but **75% of the public maximum (8)** — an absolute bar against different maxima, the same asymmetry A6 was rebuilt to remove.

- **Sector differences**: none in the code. The differing effect of the `>= 6` bar across sectors arises entirely from the framework maxima, not from any branch here.
- **Data-gap behavior**: **fails closed.** `debtRiskRP == null` suppresses the gate. For enrollment, `enrollWeak` is an OR over two independently-nullable inputs, so it survives the loss of one but not both; the `!= null` guards mean a missing series cannot be read as non-growth. No fallback substitution occurs anywhere in this gate.
- **2026-08 changes**:

> ***2026-08-29 (v49.1)*** *(lines 1571–1575): "was `ugChangePercent` — the percentage-POINT change in the UNDERGRADUATE SHARE of enrollment, 2010->2024. Not a five-year figure, not a growth rate, and not a volume at all. Because ug and gd shares are complements, a falling UG share fires for roughly half the sector on mix alone, regardless of enrollment. Now the real five-year change, via the repaired `5_year_change` alias."*

- **Notes**:
  - The gate's own comment at line 1573 says `ugChangePercent` was "2010->2024"; that is the long-run window, consistent with the enrollment analyzer. No inconsistency, recorded for completeness.
  - `_annotateGateStatus()` (lines 802–810) states that "A gate that fired without recording any getMetricValue lookup did not short-circuit — it read completedAnalyses directly (checkA6, checkA2)". **That comment is now stale for A2**: since the 2026-08-29 change, A2 calls `getEnroll5()`, which calls `getMetricValue()`, which attributes the lookup to `checkA2_LeverageLedGrowth` via the `_callingGate()` stack regex (lines 774–778). A2 is ledgered; A6 still is not.

A3 — Academic Sprawl Trap

Points 1 · fires for 83 of 1,401 (5.9%)

- **Method**: `checkA3_AcademicSprawl()` (lines 1585–1611)
- **Points**: `A3: 1` — line 180: `A1: 4, A2: 4, A3: 1, A4: 1, A5: 3, …`. Applied as `const points = triggered ? (ARCHETYPE_BASE_POINTS?.A3 ?? 3) : 0;` (line 1608) — **the inline default is 3, which disagrees with the declared constant of 1**. Unreachable while `ARCHETYPE_BASE_POINTS.A3` exists, so the awarded value is 1.
- **What it tests**: a program portfolio that is diffuse rather than concentrated (`worstDirection === 'sprawl'`), at the top severity the program-distribution measure can award, in an institution whose enrollment is not growing. Per the inline comment: *"Trigger: sprawl direction at max severity (6 = outer decile) plus soft demand"* (line 1606).
- **Inputs**:

variable	analyzer (key)	field	route
`sprawlRisk`	program distribution (`programdistribution`)	`analysisResults.riskPoints`	direct `completedAnalyses.find(a => a.analyzerKey === 'programdistribution')`, then `this.num()`
`hasSprawl`	program distribution	`analysisResults.worstDirection`	direct read, `=== 'sprawl'`
`ar.worstExtremity`	program distribution	`analysisResults.worstExtremity`	direct read, **reason string only** — not part of the gate
`e5`	enrollment (`enrollment`)	`shortTerm.totalChangePercent`	`this.getEnroll5()`
`e5` fallback	enrollment (`enrollment`)	`totalChangePercent`	`getMetricValue('enrollment','totalChangePercent')`, only when `getEnroll5()` returned `null`

All published: `worstDirection`, `worstExtremity`, `worstLevel` were added to the producer's `analysisResults` on 2026-08-26 (`unified_program_distribution_analyzer.js` lines 885–903).

- **Gate logic**:

```
1600|  let e5 = this.getEnroll5();
1601|  const e5IsLongTerm = (e5 === null);
1602|  if (e5IsLongTerm) e5 = this.num(this.getMetricValue('enrollment', 'totalChangePercent'));
1603|
1604|  const nonGrowth = (e5 != null && e5 <= 0);
1605|
1606|  // Trigger: sprawl direction at max severity (6 = outer decile) plus soft demand
1607|  const triggered = hasSprawl && (sprawlRisk != null && sprawlRisk >= 6 && nonGrowth);
```

condition	quantity	test	kind
`hasSprawl`	`worstDirection`	`=== 'sprawl'`	exact string
severity	program-distribution risk points (max 6)	`>= 6`	**floor** — and, since the producer awards only `{0, 3, 6}` (`unified_program_distribution_analyzer.js` lines 764–770), this means *exactly 6*
`nonGrowth`	five-year enrollment change, % — or the 2010→2024 change if the five-year figure is unavailable	`<= 0`	**ceiling**

The producer sets `worstDirection` to `'balanced'` unless `basePoints > 0` (lines 774–777), so the direction test and the `>= 6` test are correlated but not redundant: an institution scoring 3 has a direction and still fails the severity floor.

- **Sector differences**: none. Program-distribution max is 6 in every sector.
- **Data-gap behavior**: **fails closed.** With no program-distribution record, `ar = {}`, `worstDirection` is `undefined`, and `undefined === 'sprawl'` is false. The producer's no-data and error results (`createNoDataResult`, `createErrorResult`, lines 1043–1080) publish `riskPoints: 0` and omit `worstDirection` entirely, so both legs fail. Missing enrollment leaves `e5 === null` after both attempts and `nonGrowth` false. A3 does **not** inspect `isDataUnavailable`; it does not need to, because the no-data path already zeroes both inputs.
- **2026-08 changes**:

> ***2026-08-29 (v49.1)*** *(lines 1594–1599): "was `ugChangePercent` — the percentage-POINT change in the UNDERGRADUATE SHARE of enrollment, 2010->2024. Not a five-year figure, not a growth rate, and not a volume at all. Because ug and gd*

> *shares are complements, a falling UG share fires for roughly half the sector on mix alone, regardless of enrollment. Now the real five-year change, via the repaired `5_year_change` alias. The long-term change remains an explicit LAST resort, and the reason string says so."*

The producer-side change that made A3 reachable at all is dated 2026-08-26 and lives in the program-distribution analyzer:

> ***PUBLISHED 2026-08-26*** *(`unified_program_distribution_analyzer.js` lines 885–901): "These three were computed and then dropped at this boundary, and A3 (Academic Sprawl) and A4 (Monoculture) both gate on worstDirection: … `undefined === 'sprawl'` is false, so BOTH archetypes were structurally incapable of firing - not mistuned, unreachable. Confirmed against the v48.4 private C21 run: A3 and A4 fired 0 times in 192 institutions while 27 of them scored a maxed 6/6 on this very analyzer. The signal was there; the classification of it never left this object."*

- **Notes**:
  - The inline point default `?? 3` contradicts `ARCHETYPE_BASE_POINTS.A3 = 1`.
  - `e5IsLongTerm` is set from `e5 === null` **before** the fallback runs, so it is a flag for "the five-year figure was unavailable", not "the value now in `e5` is a long-term figure". If the fallback also returns `null`, the reason string is never built (the gate cannot trigger), so the mislabeling is not observable in output. Behaviorally inert; noted because the variable name asserts something the assignment does not establish.
  - When the fallback is taken, `nonGrowth` is evaluated against a **fourteen-year** change on the same `<= 0` bar as the five-year change. The reason string annotates this (`' (LONG-TERM FALLBACK)'`) but the threshold is not adjusted.
  - A3 awards 1 point. Combined with the requirement that the producer be at its 6/6 ceiling *and* that direction be sprawl *and* that enrollment be flat-or-down, this is among the narrowest gates in the layer for the smallest award.

A4 — Monoculture Dependency

Points 1 · fires for 94 of 1,401 (6.7%)

- **Method**: `checkA4_Monoculture()` (lines 1613–1646)
- **Points**: `A4: 1` — line 180. Applied as `(ARCHETYPE_BASE_POINTS?.A4 ?? 1)`… **no** — line 1641 reads `const points = triggered ? (ARCHETYPE_BASE_POINTS?.A4 ?? 3) : 0;`. **The inline default is 3, disagreeing with the declared constant of 1.** Unreachable while the constant exists; awarded value is 1.
- **What it tests**: a program portfolio concentrated rather than diffuse. Two independent paths are written: (1) the distribution measure at its severity ceiling with direction `monoculture`; (2) a raw concentration share of 50% or more.
- **Inputs**:

variable	analyzer (key)	field	route
`hasMonoculture`	program distribution (`programdistribution`)	`analysisResults.worstDirection`	direct read, `=== 'monoculture'`
`pdRisk`	program distribution	`analysisResults.riskPoints`	direct read, `this.num()`
`share`	program distribution	`topProgramShare`, `largestProgramShare`, `maxProgramShare`, `concentration`; then any `analysisResults` key matching `/top.*share|largest.*share|max.*share|concentration/i`	direct read, `this.pct()`
`pdReport`	program distribution	`detailedReport`	direct read — **assigned at line 1616 and never used**

- **Gate logic**:

```
1619|   const hasMonoculture = (ar.worstDirection === 'monoculture');
...
1623|   const pdRisk = this.num(ar?.riskPoints);
1624|   const riskPath = hasMonoculture && (pdRisk != null && pdRisk >= 6);
1625|
1627|  let share = this.pct(ar.topProgramShare) ?? this.pct(ar.largestProgramShare)
1628|           ?? this.pct(ar.maxProgramShare) ?? this.pct(ar.concentration);
1629|
1630|  if (share == null) {
1631|    for (const [k,v] of Object.entries(ar)) {
1632|      if (/top.*share|largest.*share|max.*share|concentration/i.test(k)) {
1633|        const num = this.pct(v);
1634|        if (num != null) { share = num; break; }
1635|      }
1636|    }
1637|  }
1638|  const sharePath = (share != null && share >= 50);
1639|
1640|  const triggered = riskPath || sharePath;
```

condition	quantity	test	kind
`riskPath` leg 1	`worstDirection`	`=== 'monoculture'`	exact string
`riskPath` leg 2	program-distribution risk points (max 6)	`>= 6`	**floor** — effectively *exactly 6*, the producer awarding only `{0, 3, 6}`
`sharePath`	largest-program share, %	`>= 50`	**floor** — see Notes, **unreachable**

- **Sector differences**: none.
- **Data-gap behavior**: **fails closed.** Missing record → `ar = {}` → `worstDirection` undefined → `riskPath` false; `share` stays `null` → `sharePath` false. The producer's no-data path publishes `riskPoints: 0, riskLevel: 'Unknown', isDataUnavailable: true` without `worstDirection`, so both paths fail. A4 does not read `isDataUnavailable`.
- **2026-08 changes**: none dated inside this method. The enabling producer change is the 2026-08-26 publication of `worstDirection`, quoted under A3 — it names A4 explicitly.

- **Notes**:
  - **sharePath is unreachable.** `unified_program_distribution_analyzer.js` publishes, on `analysisResults`, exactly: `riskPoints`, `maxRiskPoints`, `riskLevel`, `institutionType`, `dataQuality`, `confidence`, `worstDirection`, `worstExtremity`, `worstLevel` (lines 876–903). None of `topProgramShare`, `largestProgramShare`, `maxProgramShare` or `concentration` is among them, and no published key matches the salvage regex (`maxRiskPoints` contains `max` but not `share`). `share` is therefore always `null` and `sharePath` always `false`. **A4 reduces to `riskPath` alone** — i.e. to the same single-measure-at-ceiling test as A3, with the direction inverted. The `sharePath` variable is still interpolated into the reason string (line 1643), where it always prints `false`.
  - This is a gate reading four fields its producer does not publish, plus a regex sweep intended to catch a fifth name that also does not exist.
  - `pdReport` (line 1616) is a declared constant that is never read.
  - The inline point default `?? 3` contradicts `ARCHETYPE_BASE_POINTS.A4 = 1`.
  - `this.pct()` would multiply any share expressed as a ratio `<= 1` by 100 before the `>= 50` comparison — correct handling, but moot given the path is dead.
  - **UNCLEAR**: whether `sharePath` was intended to be retired or whether the producer was intended to publish a top-program share. Nothing in either file records a decision.

A5 — High-Cost Architecture, Stalling Demand

Points 3 · fires for 205 of 1,401 (14.6%)

Update (v49.13, 19 October 2026). For privates the test is `enrollWeak` AND the v2 operating margin's three-year average at or below −5.3% (`PRIVATE_MARGIN_GATES.a5MarginWeak`); the reason prints `opMargin3=`. The private `aomRP >= 6` test shown below is the pre-v49.12 gate. Publics are unchanged: `aomRP` or `omRP >= 7`, with net tuition per FTE not rising. Current method: from line 1747.

- **Method**: `checkA5_HighCostStalling()` (lines 1648–1701)
- **Points**: `A5: 3` — line 180. Applied as `(ARCHETYPE_BASE_POINTS?.A5 ?? 3)` (line 1696); inline default agrees with the constant.
- **What it tests**: an institution whose margin is weak — a cost structure it is not covering — while demand has stopped growing. For publics the gate additionally requires that net tuition per FTE not be rising.
- **Inputs**:

variable	analyzer (key)	field	route
`e10`	enrollment (`enrollment`)	`totalChangePercent` (2010→2024 total enrollment change, %)	`getMetricValue('enrollment','totalChangePercent')` → step 3 exact match
`e5`	enrollment (`enrollment`)	`shortTerm.totalChangePercent` (five-year, %)	`this.getEnroll5()`
`aomRP`	adjusted operating margin (`adjustedoperatingmargin`)	`analysisResults.riskPoints`	direct `completedAnalyses.find(...)`, `this.num()`
`omRP`	operating margin (`operatingmargin`)	`analysisResults.riskPoints`	direct `completedAnalyses.find(...)`, `this.num()`
`ntrTrend`	net tuition per FTE (`nettuitionperfte`)	`percentChange`	`getMetricValue('nettuitionperfte','percentChange')` → step 3 exact match

`percentChange` **is** published (`net_tuition_per_fte_analyzer.js` line 154) and is computed at line 271 as `((current2024 - first2010) / first2010) * 100` — a **2010→2024** change, not a recent trend, despite the variable name `ntrTrend` and the comment header "NTR trend".

- **Gate logic**:

```
1657|  const enrollWeak = ((e10 != null && e10 < 5) || (e5 != null && e5 <= 0));
...
1688|  const isPublic = this.sector !== '2';
1689|  const marginWeakPrivate = ((aomRP != null && aomRP >= 6) || (omRP != null && omRP >= 6));
1690|  const marginWeakPublic  = ((aomRP != null && aomRP >= 7) || (omRP != null && omRP >= 7));
1691|
1692|  const triggered = isPublic
1693|    ? (enrollWeak && marginWeakPublic && (ntrTrend != null && ntrTrend <= 0))
1694|    : (enrollWeak && marginWeakPrivate);
```

condition	quantity	test	kind
`enrollWeak` leg 1	`e10`, 2010→2024 enrollment change, %	`< 5`	**ceiling** (strict)
`enrollWeak` leg 2	`e5`, five-year enrollment change, %	`<= 0`	**ceiling**
`marginWeakPrivate` leg 1	adjusted operating margin risk points (max 8)	`>= 6`	**floor**
`marginWeakPrivate` leg 2	operating margin risk points (max **0**)	`>= 6`	**floor** — see Notes, dead
`marginWeakPublic` leg 1	adjusted operating margin risk points (max 8)	`>= 7`	**floor**
`marginWeakPublic` leg 2	operating margin risk points (max **0**)	`>= 7`	**floor** — dead
public-only third gate	`ntrTrend`, 2010→2024 net tuition per FTE change, %	`<= 0`	**ceiling**

- **Sector differences**: the branch is `isPublic = this.sector !== '2'`. Because `this.sector` is normalized to `'1'|'2'|'3'|'4'`, **sector 3 (private for-profit) and sector 4 (community college) both take the "public" branch**: the higher margin floor (7 rather than 6) *and* the mandatory net-tuition condition. Only sector 2 (private non-profit) takes the private branch. The framework's adjusted-operating-margin maximum is 8 in every sector, so the 6-vs-7 split is a genuine 75%-vs-87.5% difference in bar height, not an artifact of different maxima.

- **Data-gap behavior**: **fails closed on every leg.** All comparisons are guarded by `!= null`, and since the 2026-08-30 change the resolver returns `null` rather than substituting a generic field. Consequences:
    - `enrollWeak` is an OR over two inputs and survives the loss of one.
    - `marginWeak*` is an OR over two inputs, but one of them (`omRP`) is structurally 0, so in practice it depends on `aomRP` alone; a missing adjusted-operating-margin record suppresses the gate.
    - For the public branch a missing `percentChange` suppresses the gate outright. This is the explicit intent of the 2026-08-30 change: *"If it is null the input is genuinely unavailable, and the gate should not fire on a substitute."*
- **2026-08 changes**:

> ***2026-08-29 (v49.1)*** *(lines 1651–1655): "was `ugChangePercent` — the percentage-POINT change in the UNDERGRADUATE SHARE of enrollment, 2010->2024. Not a five-year figure, not a growth rate, and not a volume at all. Because ug and gd shares are complements, a falling UG share fires for roughly half the sector on mix alone, regardless of enrollment. Now the real five-year change, via the repaired `5_year_change` alias."*
>
> ***2026-08-30 (v49.2)*** *(lines 1665–1685): "the two fallback links are removed. Neither named a field that exists: `nettuitionperfte.growthRate` — the net tuition analyzer has never published a field of that name. Under the old resolver it fell through to `currentValue`: net tuition per FTE in DOLLARS. `tuitiondependency.trendDirection` — published only inside `trends`, as `trends.direction`, and as a STRING. Under the old resolver it fell through to `currentValue`: the dependency ratio. Both were then tested by `ntrTrend <= 0` as though they were a percentage change. A dollar figure and a dependency ratio are both normally positive, so in practice the links suppressed the gate rather than opening it — but for an institution with negative net tuition per FTE (they exist; see the FY2022 negative-revenue work) the second link would have fired A5 on a number that is not a trend at all. `percentChange` is the field that answers the question, and the net tuition analyzer does publish it. If it is null the input is genuinely unavailable, and the gate should not fire on a substitute."*

The resolver change that this depends on, in `_getMetricValueRaw` step 5:

> ***2026-08-30 (v49.2)*** *(lines 979–1017): "This step used to be a generic substitution … When a gate asked for a field its producer does not publish, the resolver returned whatever generic field happened to exist on that analyzer. Not null — a number. Of a different quantity, in a different unit, which the gate then compared against its threshold as though it were the thing requested. This is not a hypothetical. It is how P2 came to compute discount shortfalls of −20,000 percentage points: it asked net tuition for `peerPercentChange`, which was not published, and got `peerAverage` — dollars — instead. And it is still live in `checkA5_HighCostStalling`, whose fallback chain asks net tuition for `growthRate` (never published; resolves to `currentValue`, i.e. net tuition per FTE in DOLLARS) and tuition dependency for `trendDirection` (published only as `trends.direction`; resolves to `currentValue`, i.e. the dependency ratio). Both are then tested against `<= 0` as if they were a trend. `__hasResilientScale()` already refused to use this resolver for exactly this reason, in a comment that says a scale bar built from a change percent instead of a headcount "would be catastrophic and invisible". That workaround was the right diagnosis applied in one place. The resolver is the fault. A field that cannot be resolved by an EXPLICIT route — the keyMappings table, the exact published name, a declared alias, or a declared per-key variation — now returns null. A gate that cannot read its input must know it cannot read its input. Nulls are visible: `getMetricValue()` records them in the gate ledger and `_annotateGateStatus()` marks the gate 'unavailable'. A substituted number is not visible at all."*

The `A5` entry in that comment is written in the present tense ("it is still live in checkA5_HighCostStalling") because the resolver comment was written alongside, not after, the A5 edit. As the code now stands the chain is gone; the comment describes the state the change removed.

- **Notes**:
    - **The `omRP` legs are dead.** `framework.json` gives `UniversalOperatingMarginAnalyzer` a maximum of **0 in all four sectors**, and `analysis_orchestrator.js` lines 240–250 force `riskPoints = 0` and `riskLevel = 'Informational'` for any framework-excluded measure. So `omRP` is `0` whenever the analyzer ran and `null` when it did not; `omRP >= 6` and `omRP >= 7` can never be true in the live pipeline. `marginWeakPrivate` and `marginWeakPublic` are effectively single-input tests on `aomRP`. The reason string still reports `omRP=0`.
    - **`ntrTrend` is a fourteen-year change, not a trend.** The variable name, the section header comment ("NTR trend (for publics gate)", line 1663) and the phrase "percentChange is the field that answers the question" all read as a recent-direction test, but the producer computes `percentChange` from 2010 to 2024. An institution whose net tuition per FTE rose through 2018 and has fallen sharply since 2020 will typically still show a positive `percentChange` and be excluded from A5. Behavior, not a defect claim — but the naming does not match the quantity.

- **The `e10 < 5` threshold is asymmetric with every other enrollment test in these six gates.** A2, A3 and A5's own `e5` leg all test `<= 0`; A5's `e10` leg admits growth of up to 5%. Nothing records why. **UNCLEAR**: the basis for 5.
- **Sector 3 and sector 4 are routed as publics.** `this.sector !== '2'` was almost certainly written when the framework had two sectors. Community colleges in particular are then required to clear a net-tuition condition on a measure whose framework maximum is 6 for them and which is not a natural community-college signal.
- A5 reads `aomRP`/`omRP` by direct `completedAnalyses.find()` but `e10` and `ntrTrend` through `getMetricValue()`. Only the latter two appear in the gate ledger, so `_annotateGateStatus()` can report A5 'unavailable' for a missing enrollment or net-tuition input but not for a missing margin input.

A6 — Brand-Weakening Channel

Points 2 · fires for 431 of 1,401 (30.8%)

- **Method**: `checkA6_BrandWeakening()` (lines 1438–1467); design-intent comment 1404–1437.
- **Points**: `A6: 2` — line 180. Applied as `(ARCHETYPE_BASE_POINTS?.A6 ?? 2)` (line 1460); inline default agrees with the constant.
- **What it tests**: market position eroding in two places at once — the admissions funnel and the enrollment that funnel is supposed to produce. Both tests are expressed as a **share of each measure's own maximum**, so the sectors face the same bar despite different maxima.
- **Inputs**: A6 does **not** use `getMetricValue()`. It reads the orchestrator records directly through the local `share()` closure (lines 1439–1448), which resolves the analyzer with `pickAnalysis()` and then reads four fields off `analysisResults`:

variable	analyzer names tried	fields read	route
`brand`	`'acceptanceyield'`, `'brand_pipeline'`	`riskPoints`, `maxRiskPoints`, `riskLevel`, `isDataUnavailable`	`pickAnalysis(this.completedAnalyses, keys)`, then `a?.analysisResults ?? a ?? {}`
`enrol`	`'enrollment'`	`riskPoints`, `maxRiskPoints`, `riskLevel`, `isDataUnavailable`	same

All four fields are published by both producers (`unified_acceptance_yield_analyzer.js` lines 106–109 and its error path at 881–886; `unified_enrollment_analyzer.js` lines 128–133). `maxRiskPoints` is subsequently overwritten by the orchestrator with the framework value.

- **Gate logic**:

```
1439|  const share = (keys) => {
1440|    const a = pickAnalysis(this.completedAnalyses, keys);
1441|    const ar = a?.analysisResults ?? a ?? {};
1442|    const pts = Number(ar.riskPoints), max = Number(ar.maxRiskPoints);
1443|    const level = String(ar.riskLevel || '').toLowerCase();
1444|    if (!isFinite(pts) || !isFinite(max) || max <= 0) return null;
1445|    if (ar.isDataUnavailable === true) return null;
1446|    if (['not_applicable', 'unknown', 'insufficient_data', 'informational'].includes(level)) return null;
1447|    return { share: pts / max, pts, max };
1448|  };
1449|
1450|  const BRAND_SHARE  = 0.50;   // half of the acceptance-and-yield maximum
1451|  const ENROL_SHARE  = 0.33;   // a third of the enrolment maximum
...
1456|  const brandWeak = brand !== null && brand.share >= BRAND_SHARE;
1457|  const enrolWeak = enrol !== null && enrol.share >= ENROL_SHARE;
1458|
1459|  const triggered = brandWeak && enrolWeak;
```

condition	quantity	test	kind
`brandWeak`	acceptance/yield `riskPoints / maxRiskPoints`	`>= 0.50`	**floor**
`enrolWeak`	enrollment `riskPoints / maxRiskPoints`	`>= 0.33`	**floor**
`share()` rejection 1	`maxRiskPoints`	`<= 0` → `null`	**ceiling** (rejection)
`share()` rejection 2	`isDataUnavailable`	`=== true` → `null`	exact
`share()` rejection 3	lowercased `riskLevel`	∈ {`not_applicable`, `unknown`, `insufficient_data`, `informational`} → `null`	set membership

Translated into points, using the framework maxima:

sector	brand max	brand points needed	enrollment max	enrollment points needed
public (1)	10	≥ 5.0	15	≥ 4.95

sector	brand max	brand points needed	enrollment max	enrollment points needed
private non-profit (2)	8	≥ 4.0	18	≥ 5.94
private for-profit (3)	8	≥ 4.0	10	≥ 3.30
community college (4)	**0** — `share()` returns `null`, gate cannot fire	—	15	≥ 4.95

- **Sector differences**: none written into the gate — that is the point of the rebuild. The differences in the table above follow entirely from `framework.json` maxima. Community colleges are structurally excluded: the acceptance/yield analyzer is not even loaded for sector 4 (`analysis_orchestrator.js` lines 45–47) and its framework maximum is 0, so `share()` rejects at line 1444 in either case.
- **Data-gap behavior**: **fails closed, deliberately and explicitly.** `share()` returns `null` on a missing analyzer (`ar = {}` → `Number(undefined)` is `NaN` → `!isFinite`), on a zero or absent maximum, on an explicit `isDataUnavailable`, and on any of four non-scoring risk levels. `brandWeak`/`enrolWeak` require `!== null`, so a `null` on either side suppresses the gate. The reason string distinguishes this case: `'insufficient signal (brand pipeline or enrolment not measured)'` (line 1464).
- **2026-08 changes**: the whole gate. Comment block, lines 1404–1436:

> ***2026-08-30 REBUILD.*** *"The gate was `acceptance_yield riskPoints >= 4`, and nothing else. Two problems, one of them structural. It was not an archetype. An archetype is defined in the risk architecture as a pattern "not fully captured by a single ratio, but rather by combinations of factors". A6 was a restatement of one measure at a threshold, which means it billed the brand measure a second time rather than identifying anything the measure could not say itself. And the threshold was absolute against different maxima: 4 of 10 for publics (40%) against 4 of 8 for privates (50%), so the two sectors faced different bars for no stated reason. The result was a flag that fired for 60.4% of all scored institutions at a discrimination of +8.1 - too common to distinguish anything. Raising the threshold does not fix it: at 70% of maximum it still fires for 34.5% at +8.1. The measure on its own simply does not separate institutions. A second condition does. Brand weakening as a STRUCTURAL pattern means the market position is eroding, and that shows in two places at once: the admissions funnel, and the enrollment that funnel is supposed to produce. A weak funnel alone is ordinary. A weak funnel with enrollment following it down is the archetype. before: 846 institutions (60.4%), gap +8.1 after: 431 institutions (30.8%), gap +15.2 The group it now selects has median brand pipeline at 75% of its maximum and median enrollment risk at 73% of its maximum; 253 of the 431 sit in High or Critical and only 6 in Low. Overlap with P8_MarketPosition falls from 85% of that pattern to 63%. Both tests are ratios of each measure's OWN maximum, so the sectors face the same bar."*

The `pickAnalysis()` fix that A6 depends on is dated by content rather than by a date stamp (lines 20–27, quoted under **Shared helpers**): it names A6 as the gate that was reading `enrollment` risk points and reporting them as acceptance/yield.

- **Notes**:
  - **A6 is unledgered.** It makes no `getMetricValue()` call, so `_annotateGateStatus()` (lines 802–810) will label it `'evaluated_unledgered'` when it triggers and `'not_reached'` when it does not — the latter being inaccurate, since A6 always evaluates both inputs. The comment at lines 803–806 anticipates this and names `checkA6`.
  - The design-intent comment says the rebuild puts *both* tests on each measure's own maximum "so the sectors face the same bar". That holds for the *share*, but the two constants differ from one another (0.50 vs 0.33) and neither is derived in the comment. **UNCLEAR**: why 0.33 for enrollment specifically. The stated post-change medians (75% and 73% of maximum) sit well above both floors, so the constants are not simply the observed medians.
  - `'brand_pipeline'` is offered as an alternative analyzer name at line 1453. No producer in this tree emits that key; the live key is `acceptanceyield`. Harmless — it is the second name tried and the first succeeds — but it is a name the pipeline does not publish.
  - `share()` accepts a fractional `riskPoints`. Program-style analyzers round to two decimals; acceptance/yield and enrollment award whole and half points. No rounding is applied before the division, so a value such as `4.0/8` sits exactly on the floor and passes (`>=`).
  - `share()` reads `riskLevel` case-insensitively but the reject list uses snake_case (`not_applicable`, `insufficient_data`). Producers in this tree publish display-cased levels (`'Unknown'`, `'Informational'`, `'Critical'`), so only `'unknown'` and `'informational'` can actually match after lowercasing; `not_applicable` and `insufficient_data` are defensive entries for exporters not present here.

A7 — High-Pell "Mission Pressure Cooker"

Points 3 (0 at community colleges since v49.19) · fires for 395 of 1,401 (28.2%, pre-3-September baseline)

Update (v49.19, 19 October 2026): retired for community colleges. For sector 4 the method returns 0 points before any test, with the reason `retired for community colleges (v49.19): Pell is scored as its own measure`. At community colleges A7 had become a Pell flag: it fired at 66% of colleges, at 86–96% of those with Pell at or above 45% and at 0–10% below, because retention under 78–80% and UNA per FTE under $25,000 are near-universal in the sector. Pell is already scored as its own measure, so A7 charged it twice. Publics and privates are unchanged, including sector-1 associate's colleges, which run on the public framework. Current method: from line 1810.

(display name transcribed from line 144: `A7: 'High-Pell â€œMission Pressure Cookerâ€'` — the file carries mojibake where curly quotes should be; the intended string is `High-Pell "Mission Pressure Cooker".`)

- **Method**: `checkA7_PellPressureCooker()` (lines 1706–1771). Aliased to `this.checkA7_HighPellPressure` at lines 2874–2876 before the map is built; the map calls the alias (line 2892).
- **Points**: `A7: 3` — line 180, `ARCHETYPE_BASE_POINTS`. Applied at line 1762 as `ARCHETYPE_BASE_POINTS?.A7 ?? 3`.
- **What it tests**: a high-Pell enrollment combined with weak retention and a thin financial cushion. Header comment, lines 1704–1705: `// A7 - High-Pell Mission Pressure Cooker` / `// A7 â€“ Pell Pressure Cooker (tiered + robust)`. Three alternative tiers, each looser on Pell as it becomes stricter elsewhere.

Inputs

Field read	Producer (analyzer key)	Route	Published?
`pell.currentValue`	`pell` (`simple_pell_analyzer.js` line 210, `currentValue: pellAnalysis.overallPellPercentage`)	`getMetricValue` → exact field	Yes
`pellAR.currentValue / pellPercent / pell_percentage / percent / share`	`pell` record found by `a.analyzerKey === 'pell'` (line 1722, strict equality)	direct `pick()`	`currentValue` yes; the other four **no**
`grantAR.pgrnt_p / pell_percent`	looked up as `a.analyzerKey === 'grant_distribution'` (line 1723)	direct `pick()`	**No such analyzer key exists** — see Notes
`retention.currentRetentionRate`	`retention` (`unified_retention_analyzer.js` line 133)	`getMetricValue` → exact field	Yes
`retention.first_year_retention`, `retention.currentValue`	`retention`	`getMetricValue` → declared alias (lines 544–549)	Not published under those names; resolve via alias to `currentRetentionRate`
`retentionAR.retention_rate / rate`	`retention`, strict `analyzerKey === 'retention'` (line 1735)	direct `pick()`	**No**
`una_per_fte` (via `resolveUnaPerFTE()`, lines 701–713)	`unrestrictednetassets` (`unrestricted_net_assets_analyzer.js` line 180)	direct field, then `total/currentNetAssets/currentValue ÷ enrolment`	Yes (`una_per_fte` published directly)
`primaryreserve.riskPoints`	`primaryreserve`	`getMetricValue` → exact field	Yes
`viability.riskPoints`	`viability`	`getMetricValue` → exact field	Yes

Gate logic

Normalizers: `pct()` (line 1709) delegates to `this.pct` (lines 658–662), which returns `v <= 1 ? v * 100 : v`.

Sub-condition	Line	Test	Direction
`ret78`	1746	`retention < 78`	**ceiling**, strict
`ret80`	1747	`retention < 80`	**ceiling**, strict
`cushionStrong`	1749	`unaPerFTE < 25000` **OR** `prRP >= 4`	ceiling (strict) / **floor**
`cushionMedium`	1750	`unaPerFTE < 35000` **OR** `prRP >= 3` **OR** `viaRP >= 2`	ceiling (strict) / **floor** / **floor**

Sub-condition	Line	Test	Direction
Tier 1 `t1`	1753	`pellPct >= 60` **AND** `ret78`	**floor**
Tier 2 `t2`	1756	`pellPct >= 50` **AND** `ret78` **AND** `cushionMedium`	**floor**
Tier 3 `t3`	1759	`pellPct >= 45` **AND** `ret80` **AND** `cushionStrong`	**floor**

`triggered = !!(t1 || t2 || t3)` (line 1761).

Tier 1 is an override: it requires **no** cushion signal at all.

- **Sector differences**: none in the gate. Indirectly, `pell`, `retention`, `primaryreserve`, `viability` and `unrestrictednetassets` run in all four sectors, so A7 is evaluable everywhere. Baseline: 395 firings — 206 private, 189 public. (Superseded for sector 4 at v49.19: see the update at the head of this entry.)
- **Data-gap behavior**: **fails closed on the Pell and retention legs.** Every comparison is guarded by an explicit `!= null` / `!== null` test, so a null Pell percentage or null retention makes all three tiers false. A missing cushion makes `cushionStrong`/`cushionMedium` false, closing t2 and t3 but leaving t1 unaffected — so a null balance sheet does **not** block a Tier 1 firing.
- **2026-08 changes**: no dated comment inside the method. One dated comment in `metricAliases` names it directly (lines 537–543):

> *`// 2026-08-30 (v49.2): the second and third links of the retention chains in // checkOperationalEfficiency and checkA7_PellPressureCooker name fields the // retention analyzer has never published (it publishes currentRetentionRate). // Under the old resolver they fell through to generic substitution; under the new // one they would simply return null. Declared as aliases instead, so the chains // mean what they appear to mean. No behavioral change while the first link // resolves — which it does whenever the retention analyzer ran.`*

Notes

- **Dead fallback.** Line 1723 searches for `a.analyzerKey === 'grant_distribution'`. No analyzer produces that key — the grant analyzer is `UniversalGrantFundingAnalyzer` → `grantfunding`. `grantAR` is therefore always `{}` and the `pgrnt_p` / `pell_percent` fallback at line 1728 can never resolve. Harmless while the `pell` analyzer runs, but it is not the safety net the comment at line 1727 describes: *"fallback if Pell analyzer missing: some feeds expose pgrnt_p"*.
- **Naming is inverted.** `cushionStrong` (line 1749) is true when the cushion is *weak* (UNA/FTE below $25,000, or primary-reserve **risk** at or above 4). The names describe the strength of the *distress signal*, not of the cushion. The logic is self-consistent; the identifiers read backwards.
- **`pct()` scaling hazard.** `this.pct` multiplies any value `<= 1` by 100. An institution with a genuine Pell share of 0.9% would be read as 90%. Not observed in the baseline, but the transform is unguarded.
- Three of the four `pick()` fallback paths (lines 1726, 1735) name fields no producer publishes. They are inert rather than harmful, because `pick` returns `null` on a miss.

A8 — Brand-Strong but Margin-Thin

Points 2 · never fired in the baseline; fires since v49.15 (rate not re-baselined)

- **Method**: `checkA8_BrandStrongMarginThin()` (lines 1773–1786).
- **Points**: `A8: 2` — line 180. Applied at line 1783 as `ARCHETYPE_BASE_POINTS?.A8 ?? 2`.
- **What it tests**: an institution whose admissions brand is intact while its operating margin or net-tuition trajectory is not. There is no descriptive comment beyond the header at line 1772.

Current gate (v49.15, 19 October 2026)

The Inputs, Gate logic and Notes that follow this block describe the gate before v49.15. They are kept as the record of why A8 never fired. The current gate is `checkA8_BrandStrongMarginThin()`, from line 1886:

- **Reads:** exact analyzer keys only, matched on the normalized key: `acceptanceyield`, `adjustedoperatingmargin`, `operatingmargin` and `nettuitionperfte`. `pickAnalysis` is no longer used.
- **brandStrong:** admissions risk at or below 1 point, and admissions must be scored. An absent record, or one whose level is `not_applicable`, `unknown`, `insufficient_data` or `informational`, is not a strong brand, and the reason reads `insufficient signal (admissions not scored)`.
- **marginThin:** adjusted operating margin risk at or above 5, OR operating margin risk at or above 5, OR net tuition per FTE `percentChange` at or below 0 (2010 to 2024).
- **Trigger and points:** brandStrong AND marginThin, +2, as originally specified (D. Greenstein, 19 October 2026). For privates the adjusted margin record is the v2 operating basis (v49.12). The operating margin leg is kept as specified; it scores 0 at every four-year institution, so it cannot carry the gate there.
- **Sector:** A8 cannot fire at community colleges, which have no admissions score.
- **Validation:** on the 27-institution anchor run (19 October 2026) A8 fired at exactly the four predicted: Edgewood, Thomas Aquinas, Hilbert and Averett.
- **Evidence recorded for calibration:** on the private closure set A8 fires for 79 of 811 privates with admissions data, of which 1 closed or merged in 2024–26 (1.3%, against 2.1% for the sector). On the 4 October public run it fires for 19 publics, about half of them large flagships in the Low or Moderate band. Colleges with strong demand and thin margins close less often than the sector, and the margin is also scored on its own. The price is open for review.

History — the gate before v49.15

Inputs

Variable	Line	Field read	Producer	Route	Published?
`ayRisk`	1774	`analysisResults.riskPoints` from `pickAnalysis(['acceptanceyield'])`	`acceptanceyield` (Brand Pipeline)	`pickAnalysis`	**Yes** — resolves correctly
`aomRisk`	1775	`analysisResults.riskPoints` from `pickAnalysis(['adjustedoperatingmargin'])`	intended: `adjustedoperatingmargin`; **actual: `operatingmargin`**	`pickAnalysis`	Field yes, **wrong analyzer**
`omRisk`	1776	`analysisResults.riskPoints` from `pickAnalysis(['operatingmargin'])`	`operatingmargin`	`pickAnalysis`	Yes, but always `0`
`ntrChg`	1777	`analysisResults.fiveYearChange`	`nettuitionperfte`	`pickAnalysis`	**NEVER PUBLISHED**
`ntrChg` (fallback)	1778	`analysisResults.recentChange`	`nettuitionperfte`	`pickAnalysis`	**NEVER PUBLISHED**

The net-tuition analyzer publishes the quantity as `percentChange` (`net_tuition_per_fte_analyzer.js` line 154).

Gate logic

Sub-condition	Line	Test	Direction
`brandStrong`	1780	`ayRisk <= 1`	**ceiling**
`marginThin` (a)	1781	`aomRisk >= 5`	**floor** — always false, see below

Sub-condition	Line	Test	Direction
`marginThin` (b)	1781	`omRisk >= 5`	**floor** — always false
`marginThin` (c)	1781	`ntrChg <= 0`	**ceiling** — always false (input unreadable)

`triggered = brandStrong && marginThin` (line 1782).

Note that `ayRisk <= 1` is an absolute point count, not a ratio. Brand Pipeline's `maxRiskPoints` is 8 in the private/CC framework and 10 for sector 1 (`unified_acceptance_yield_analyzer.js` line 34), so the same bar means ≤12.5% of max in one sector and ≤10% in another.

- **Sector differences**: none written into the gate. The Brand Pipeline analyzer is **excluded for community colleges** (`analysis_orchestrator.js` line 46, `CC_EXCLUDED_ANALYZERS`), so for sector 4 `ayRisk` is `null` and `brandStrong` is false as well — A8 fails on both halves there rather than one.
- **Data-gap behavior**: **fails closed**, on both halves. Every comparison is guarded by `!= null`. There is no fail-open path. The problem is not that the gate opens on missing data; it is that its inputs are missing or misrouted **by construction**, so it never opens at all.
- **2026-08 changes**: no dated comment in or above the method. The defect is recorded externally, in `rat_selfcheck.js` lines 9–19:

> *`* The August 2026 audit found the same defect seven times: a layer-2 gate reads * a field at a path its producer does not publish on the record the gate actually * receives. Every instance was silent. The gate returned "criteria not met", the * composite came out looking plausible, and nothing anywhere complained. * * P9_EndowmentDrainRisk read rawData.perFTEEndowment.currentValue * simple_pell_analyzer read peerAnalysis.peerValues (published one level deeper) * checkA8 read analysisResults.fiveYearChange (never published)`*

Notes

- **FIXED IN v49.15. Before then: DEFECTIVE — the gate cannot fire.** See the priority finding above. Three independent faults, each sufficient on its own to kill the `marginThin` disjunction: 1. *Misrouted lookup* (line 1775): `pickAnalysis(['adjustedoperatingmargin'])` returns the `operatingmargin` record because `pickAnalysis` has no exact-match pre-pass and `'adjustedoperatingmargin'.includes('operatingmargin')` is true. `aomRisk` and `omRisk` are the same number. 2. *Framework exclusion* (`risk_scoring_engine.js` lines 22, 63, 105, 155): Operating Margin is 0 points in every sector; the orchestrator forces `riskPoints` to `0`. Confirmed 1,401/1,401 in the baseline. `0 >= 5` is false. 3. *Unpublished fields* (lines 1777–1778): `fiveYearChange` and `recentChange` exist in no producer.
- **Consequence for the framework.** 2 points of documented risk are never charged. This is the same class of error as the LA1/LA2 removal recorded at lines 154–160 — *"the framework documented 4 points of risk it never charged"* — except that A8 is registered and evaluated, so it is invisible to a registration audit.
- **The status annotation is misleading.** A8 makes no `getMetricValue` calls, so `_annotateGateStatus` (lines 802–810) stamps it `'not_reached'` every time. That status means "a front gate short-circuited". A8 short-circuits nothing. (Since v49.10 an untriggered A8 is stamped unledgered and prints its reason.)
- **The repair was applied in v49.15** (README item 37): exact-key reads, the `percentChange` read, and the admissions guard. The Operating Margin leg was kept, as specified. Validated 19 October 2026.
- **UNCLEAR**: whether the `ntrChg <= 0` leg was intended as a five-year change or a since-2010 change. `percentChange` in the net-tuition analyzer is `((current2024 - first2010) / first2010) × 100` (that file, line 271) — a fourteen-year figure — while the variable name in A8 says `fiveYear`. The code does not resolve which window was intended. Resolved in v49.15: the gate reads percentChange, 2010 to 2024.

A9 — The Squeezed Middle

Points 2 · fires for 224 of 1,401 (16.0%)

Update (v49.13, 19 October 2026). For privates, stillCoping is the v2 operating margin's three-year average above −3.6% (`PRIVATE_MARGIN_GATES.a9StillCoping`), and the reason prints `opMargin3=`. The `marginRP <= 4` test below, and the note that an always-zero `omRP` masks a missing margin, now apply only to publics and community colleges. For privates a missing margin closes the gate. Current method: from line 1950.

- **Method**: `checkA9_SqueezedMiddle()` (lines 1814–1858). Design comment lines 1788–1813.
- **Points**: `A9: 2` — line 180. Applied at line 1852 as `ARCHETYPE_BASE_POINTS?.A9 ?? 2`.
- **What it tests**: demand pressure that is real but not terminal, on a balance sheet under real pressure, at an institution whose operating margin is still holding. Stated intent, lines 1790–1793:

> *// INTENT, as stated by D. Greenstein 29 August 2026: institutions experiencing real // structural pressure that are managing as best they can against it. Three things at // once - a genuine risk that earns points, a category layer 1 systematically scores as // Moderate because the institution is still coping, and a useful descriptive tag.*

Inputs

Variable	Line	Field read	Producer	Route	Published?
`chg10`	1819	`enrollment.totalChangePercent`	`enrollment` (`unified_enrollment_analyzer.js` line 123)	`getMetricValue` → exact field	Yes — 2010→2024 change
`recent`	1820	`this.getEnroll5()` → `enrollment.5_year_change` → keyMapping `shortTerm.totalChangePercent` (line 898)	`enrollment` (that file, line 401)	`getMetricValue` deep path	Yes — 2019→2024 change
`debtRP`	1834	`debtrevenue.riskPoints`	`debtrevenue`	`getMetricValue` → exact field	Yes
`prRP`	1835	`primaryreserve.riskPoints`	`primaryreserve`	`getMetricValue` → exact field	Yes
`viaRP`	1836	`viability.riskPoints`	`viability`	`getMetricValue` → exact field	Yes
`aomRP`	1844	`adjustedoperatingmargin.riskPoints`	`adjustedoperatingmargin`	`getMetricValue` → **exact key pass wins**, resolves correctly	Yes
`omRP`	1845	`operatingmargin.riskPoints`	`operatingmargin`	`getMetricValue` → exact field	Yes, but always `0`

A9 does **not** suffer A8's misrouting, because `_getMetricValueRaw` runs a complete exact `normKey` equality pass (lines 861–862) before any substring fallback.

Gate logic — four conditions, all required

Constant / condition	Line	Test	Direction
`DEMAND_LONG = -10`	1821	`chg10 <= -10`	**ceiling** (a 10%+ fall over fourteen years)
`DEMAND_SHORT = -5`	1822	`recent <= -5`	**ceiling** (a 5%+ fall over five years)
`demandPressure`	1823–1824	`(chg10 !== null && chg10 <= DEMAND_LONG)` **OR** `(recent !== null && recent <= DEMAND_SHORT)`	disjunction
`NOT_COLLAPSING = -35`	1828	—	—
`stillStanding`	1829	`(chg10 === null)` **OR** `(chg10 >= -35)`	**floor**
`balanceSheetPressure`	1837–1839	`debtRP >= 4` **OR** `prRP >= 4` **OR** `viaRP >= 3`	three **floors**
`marginRP`	1846–1848	`max(aomRP, omRP)`, `null` only if both are null	—

Constant / condition	Line	Test	Direction
stillCoping	1849	marginRP !== null && marginRP <= 4	**ceiling**

`triggered = !!(demandPressure && stillStanding && balanceSheetPressure && stillCoping)` (line 1851).

The balance-sheet floors are documented at lines 1832–1833:

> *// Was: debt 1-6, primary reserve 2-6, viability 1-4. One point of debt risk out of // eight is not pressure. These floors are half of each measure's maximum.*

- **Sector differences**: none in the gate. All five source analyzers run in all sectors. Baseline: 224 firings — 128 private, 96 public.
- **Data-gap behavior**: **mixed.**
  - `demandPressure` fails **closed** — both legs require a non-null value.
  - `balanceSheetPressure` fails **closed** — all three legs require non-null.
  - `stillCoping` fails **closed** — requires `marginRP !== null`. Because `omRP` is always `0` (never null), `marginRP` is in practice never null, so `stillCoping` degenerates to `max(aomRP, 0) <= 4` and is satisfied whenever AOM risk is 0–4 *or* AOM is unreadable. See Notes.
  - `stillStanding` fails **OPEN**: line 1829 explicitly treats `chg10 === null` as "not collapsing". This is the only fail-open leg in the gate, and it is deliberate — a missing fourteen-year series should not disqualify an institution whose five-year series shows the pressure.
- **2026-08 changes**: the whole gate was rebuilt on 2026-08-29. Verbatim, lines 1795–1813:

> *// 2026-08-29 REBUILD. The gate did not implement that. Every band opened at or near // zero - long-term enrolment change in [-20, +2], margin risk points in [0, 4], debt // risk points from 1 - so it tested "not bad on any axis", which is a description of a // comfortable institution, not a squeezed one. Nothing required that pressure be // PRESENT. Across the 1,401 scored institutions it selected 243 of whom 33% had GROWING // enrolment over fourteen years, 43% carried zero margin risk, and half sat at one point // or less of primary-reserve and viability risk. Their margins ran 29 points better than // the rest of the population. They averaged 7.7 points BELOW everyone else, because the // gate was selecting institutions that were simply fine. // // The rebuild puts a floor under each band. Pressure must be present on demand AND on // the balance sheet; the margin condition stays as the test of whether the institution // is still coping, which is the part the original had right. // // The group it now selects: median long-term enrolment -14.3%, median recent -10.8%, // median debt risk 7 of 8, median viability 4 of 8 - and median margin risk 1 of 8. // Enrolment falling, leverage heavy, margin still holding. It averages +3.5 points above // the rest of the population, and 121 of its 224 members sit in the Moderate band, which // is the point: layer 1 reads them as unremarkable and this says they are not.*

The claimed post-rebuild count of 224 members **matches the validated baseline exactly** (224 firings across 1,401 scored).

Notes

- **`omRP` contributes nothing but does mask a null.** Operating Margin is 0 points in every sector, so `omRP` is always the number `0`, never `null`. `marginRP` at lines 1846–1848 is therefore `max(aomRP ?? -Infinity, 0)` and is never `null`. The consequence: when the AOM analyzer fails to produce a value, `stillCoping` evaluates `0 <= 4` → true, i.e. an institution with **no readable margin at all** is treated as "still coping". The `marginRP !== null` guard at line 1849 was written to prevent exactly this and is neutralized by the always-zero `omRP`. Concern recorded; no change made.
- **`NOT_COLLAPSING` only tests the long-term series.** Line 1829 checks `chg10` alone. An institution with a null fourteen-year change and a catastrophic five-year change (say −60%) passes `stillStanding` on the null branch and can still trigger A9, even though the comment at lines 1826–1827 says the gate should stand aside for collapse cases:

> *// Not collapsing. Beyond this the institution is not "managing" and the enrolment // cliff, A12 and the severity override already carry it; A9 would only double-count.*

- **Constants.** `DEMAND_LONG`, `DEMAND_SHORT`, `NOT_COLLAPSING` are all read. `balanceSheetPressure` and `stillCoping` are locals, both read. No dead constant in this method.
- **`probeA9Once()` (lines 451–468) is orphaned.** It is a debug probe reading the same inputs; grep finds no caller. Its `chg10` fallback calls `this.getEnrollmentDeclineWorst()`, which the live A9 does not use.

A10 — Endowment-Masked Operating Deficit

Points 6 · fires for 87 of 1,401 (6.2%)

Update (v49.10 and v49.11, 15 October 2026). The gate is unchanged. Its untriggered reason now prints the four readings, for example `not met (needs structMargin>=4, endowDraw>=4, endowDep>=2, restrict>=3): structMargin=0, endowDraw=0, endowDep=0, restrict=0`, and an untriggered A10 is stamped `unledgered` rather than `not_reached`. When A10 fires, P9 defers to it (see P9). Current method: from line 2004.

- **Method**: `checkA10_EndowmentMaskedDeficit()` (lines 1861–1884).
- **Points**: `A10: 6` — line 180. Applied at line 1879 as `ARCHETYPE_BASE_POINTS?.A10 ?? 6`.
- **What it tests**: an operating deficit concealed by endowment support — structural margin failing, the draw unsustainable, dependency on that draw material, and the net assets behind it restricted. No descriptive comment beyond the header at line 1860.

Inputs

All four are read through a local `get()` (lines 1863–1867) that normalizes analyzer keys (`.toLowerCase().replace(/[^a-z0-9]/g,'')`), matches against an exact `Set`, and returns `a?.analysisResults?.riskPoints ?? a?.riskPoints`. **No fuzzy matching** — A10 is not exposed to the `pickAnalysis` collision.

Variable	Line	Keys accepted	Producer	Published?
`structRP`	1868	`structuralmargin`, `structural_margin`, `structural_operating_margin`	`structuralmargin`	Yes
`drawRP`	1869	`endowmentdrawsustainability`, `endowment_draw_sustainability`	`endowmentdrawsustainability`	Yes
`depRP`	1870	`endowmentdependency`, `endowment_dependency`	`endowmentdependency`	Yes
`restrictRP`	1871	`restrictionprofile`, `restriction_profile`	`restrictionprofile`	Yes

The record shape carries `riskPoints` at both the top level and inside `analysisResults` (`analysis_orchestrator.js` lines 362–375), so the `??` fallback at line 1866 is belt-and-braces rather than load-bearing.

Gate logic — four conditions, all required (lines 1873–1877)

Condition	Line	Test	Direction
Structural margin	1874	`Number.isFinite(structRP) && structRP >= 4`	**floor**
Endowment draw	1875	`Number.isFinite(drawRP) && drawRP >= 4`	**floor**
Endowment dependency	1876	`Number.isFinite(depRP) && depRP >= 2`	**floor**
Restriction profile	1877	`Number.isFinite(restrictRP) && restrictRP >= 3`	**floor**

All four are absolute point counts, not ratios. Declared maxima (sector 2): structural margin 8, endowment draw 10, endowment dependency 6, restriction profile 6 (`risk_scoring_engine.js` lines 58, 65, 67, 68). So the bars are 50%, 40%, 33% and 50% of max respectively.

- **Sector differences**: **A10 is private-only in effect, though nothing in the gate says so.** In `risk_scoring_engine.js`, three of the four inputs are declared at 0 points outside sector 2:

Analyzer	Sector 1 (public)	Sector 2 (private)	Sector 4 (CC)
`UniversalRestrictionProfileAnalyzer`	**0** (line 18)	6 (line 58)	**0** (line 136)
`UniversalEndowmentDrawSustainabilityAnalyzer`	**0** (line 34)	10 (line 67)	**0** (line 181)
`UniversalEndowmentDependencyAnalyzer`	**0** (line 33)	6 (line 68)	**0** (line 180)

A framework value of 0 forces `riskPoints` to `0` and `riskLevel` to `'Informational'` (`analysis_orchestrator.js` lines 228–251). `0 >= 4` and `0 >= 3` are false, so `drawRP`, `depRP` and `restrictRP` all fail for publics and community colleges. Baseline confirms: **87 of 87 A10 firings are private, 0 public.**

- **Data-gap behavior**: **fails closed.** `Number.isFinite(undefined)` and `Number.isFinite(null)` are both `false`, so any unresolved input closes the gate. The `reason` string for the untriggered case is `'insufficient signal'` (line

1882), which does not distinguish "measured and passed" from "could not read". (Superseded at v49.10: the untriggered reason now prints the four readings.)

- **2026-08 changes**: none. No dated comment in or above the method.

Notes

- The sector asymmetry above is **structural, not a defect**, but it is nowhere documented in the gate. A reader of `checkA10_*` alone would expect it to be sector-neutral. The contrast with A14, whose header comment explicitly discusses its sector selectivity (lines 1956–1957), is worth noting.
- `reason: 'insufficient signal'` (line 1882) is the same message whether the institution was measured and cleared, or could not be measured at all. Every other gate in this range emits `'criteria not met'` or a diagnostic count. (Superseded at v49.10.)
- A10 makes no `getMetricValue` calls, so like A8 it is stamped `'not_reached'` by `_annotateGateStatus` whenever it does not trigger (lines 802–810). (Since v49.10: stamped unledgered, with its reason.)
- **The gap between A10 and P9** (README item 38). 61 privates had a Critical structural margin and a three-year draw above 5.5% and fired neither gate. A10 needs restriction at or above 3 points. P9 needs endowment per FTE below 0.48× the peer median, which excludes colleges whose peers are also poorly endowed: Viterbo (restriction 0; endowment per FTE 1.39× its peers) and Madonna (draw 12.5%) are examples. Since v49.17 the unscored flag U38 marks these colleges; see the entry after P9.

A12 — Broad Balance-Sheet Failure

Points 6 · fires for 121 of 1,401 (8.6%)

- **Method**: `checkA12_BalanceSheetConcentration()` (lines 1893–1948). Header comment lines 1886–1892. Note the method name says "Concentration" while the display name says "Failure".
- **Points**: `A12: 6` — line 195, `ARCHETYPE_BASE_POINTS`. Applied at line 1942 as `ARCHETYPE_BASE_POINTS?.A12 ?? 6`.
- **What it tests**: balance-sheet distress that spans **independent** constructs, not one quantity measured several ways. Header, lines 1887–1892:

> *// Fires when balance-sheet distress spans INDEPENDENT constructs, not when one // underlying quantity is measured three ways. Primary reserve, viability and UNA // are all functions of unrestricted/expendable net assets: a negative UNA turns // all three Critical simultaneously. Counting that as three signals double-counts // one fact — and empirically it fires on GASB 68/75 pension/OPEB allocations // across entire state systems rather than on institutional distress.*

Inputs

A12 iterates `this.completedAnalyses` directly (line 1916). For each record it reads `analysisResults.riskLevel`, `analysisResults.riskPoints`, `analysisResults.maxRiskPoints` and `analysisResults.scoringBasis`, and matches `norm(a.analyzerKey)` — falling back to `norm(a.measure)` — against a nine-entry family map.

FAMILY map (lines 1898–1908):

Analyzer key	Family	Runs?
`primaryreserve`	`net_asset_position`	Yes, all sectors
`viability`	`net_asset_position`	Yes, all sectors
`unrestrictednetassets`	`net_asset_position`	Yes, all sectors
`cfi`	`net_asset_position`	**NEVER — see Notes**
`restrictionprofile`	`net_asset_position`	Yes, but 0-point outside sector 2
`capitalizationratio`	`leverage`	Yes, all sectors
`debtrevenue`	`leverage`	Yes, all sectors
`returnonnetassets`	`flow`	Yes, all sectors
`endowmentdrawsustainability`	`endowment`	Yes, but 0-point outside sector 2

`scoringBasis` is published by `unified_capitalization_ratio_analyzer.js` (line 172, value `'restriction-adjusted (negative net position)'` at line 864) and by `unified_viability_analyzer.js` (line 796, value `'restriction-adjusted'` only — it never contains the substring A12 tests for).

Gate logic

Constant / step	Line	Rule	Direction
`MIN_POINTS = 4`	1910	`if (mx !== null && mx < 4) continue` — skip low-weight analyzers	**floor** on the analyzer's declared max
`CRITICAL_RATIO = 0.80`	1911	ratio bar for "Critical"	**floor**
Family lookup	1917	`FAMILY[norm(analyzerKey)] || FAMILY[norm(measure)]`; no match → skip	—
Capitalization re-family	1932–1934	if `String(ar.scoringBasis).includes('negative net position')` → `effFam = 'net_asset_position'`	—
`isCritical`	1935–1936	`riskLevel === 'critical'` **OR** (`rp/mx >= 0.80` with `mx > 0`)	**floor**
Trigger	1941	`hits.length >= 3` **AND** `families.size >= 2`	two **floors**

The re-family rule is documented at lines 1927–1930:

> *// Capitalization scored via the negative-net-position guard is a net-asset- // position fact, not an independent leverage fact: it fires because UNA + debt // is negative, i.e. off the same number*

> *as primary reserve, viability and UNA. // Counting it as 'leverage' would let one fact clear the two-construct test.*

Return value carries two extra keys beyond the standard shape: `criticalKeys: hits` and `families: Array.from(families)` (line 1947).

- **Sector differences**: none written into the gate, but two of the nine families are effectively private-only. `restrictionprofile` (max 0 outside sector 2) and `endowmentdrawsustainability` (max 0 outside sector 2) are both skipped by the `mx < MIN_POINTS` guard at line 1924 for publics and CCs — `0 < 4`. Public institutions therefore have only `primaryreserve`, `viability`, `unrestrictednetassets` in `net_asset_position`, plus `leverage` (2) and `flow` (1); the `endowment` family is unreachable for them. Baseline: 121 firings — **102 private, 19 public**.
- **Data-gap behavior**: **mostly fails closed, with one fail-open branch.**
  - A record with `rp` or `mx` null cannot satisfy the ratio test; it can still satisfy `isCritical` via `riskLevel === 'critical'` alone (line 1935), with no points at all.
  - Line 1924 reads `if (mx !== null && mx < MIN_POINTS) continue;`. **A null `mx` does not skip the record.** An analyzer that published a `riskLevel` of "Critical" but no `maxRiskPoints` would be counted as a hit regardless of its weight. This is a fail-open branch in an otherwise closed gate.
  - A missing analyzer simply contributes no hit, closing the gate.
- **2026-08 changes**: no dated comment. The header (lines 1886–1892) is undated design rationale.

Notes

- **`cfi` is a constant never read.** Line 1902 maps `cfi: 'net_asset_position'`. There is a `UniversalCFIAnalyzer` class in `modular_tool_v48 no data cap.html` (line 1226), but it is **not** in the `allAnalyzers` array (`analysis_orchestrator.js` lines 65–107) and is not instantiated by any sector-specific branch (lines 120–168). It therefore never appears in `completedAnalyses` and the `cfi` entry can never match. (Were it registered, its key would be `cfi`, so the entry is at least correctly spelled — but `getAnalyzerDisplayName` names the measure `compositefinancialindex`, line 3457 of the HTML, so the two spellings are already inconsistent elsewhere.)
- **The `mx === null` pass-through** at line 1924 is described above under Data-gap. Recorded as a concern; not corrected.
- **Only capitalization can be re-familied.** Line 1932 tests `scoringBasis` on every record, but only `unified_capitalization_ratio_analyzer.js` ever publishes a string containing `'negative net position'`. The viability analyzer publishes `'restriction-adjusted'` without that substring, so a viability record scored on the same negative-net-position basis is *not* re-familied — it is already `net_asset_position`, so the outcome is unchanged, but the rule is narrower than its comment ("Analyzers sharing a family share an input and count once", line 1896–1897) implies.
- **Name mismatch.** `ARCHETYPES.A12` is `'Broad Balance-Sheet Failure'` (line 148); the method is `checkA12_BalanceSheetConcentration`. The method name is the older concept (concentration of criticals); the display name is the current one. Both are live.

A14 — Demand Collapse with Intact Reserves

Points 8 · fires for 82 of 1,401 (5.9%)

- **Method**: `checkA14_DemandCollapseIntactReserves()` (lines 1958–1996). Header comment lines 1950–1957.
- **Points**: `A14: 8` — line 195. Applied at line 1987 as `ARCHETYPE_BASE_POINTS?.A14 ?? 8`. **This is the largest positive award in the archetype layer.**
- **What it tests**: the pre-distress signature — demand-side measures failing while the balance sheet still reads adequate. Header, lines 1951–1957:

> *// The pre-distress signature: enrollment, pricing and program metrics failing while // primary reserve and viability still read adequate. The framework carries ~74 balance // sheet points against ~52 demand points, so an institution whose enrollment model is // broken but whose endowment is intact scores Moderate — the thing that is failing is // under-weighted relative to the thing that is holding. Sweet Briar is the canonical case. // Naturally sector-selective: fires 19% in C21 private, 10% in C22, 1% in C18, because // public reserves are GASB-depressed regardless of demand condition.*

Inputs

A14 iterates `completedAnalyses` directly (line 1972), keying on `norm(a.analyzerKey) || norm(a.measure)` (line 1973) and reading `analysisResults.riskPoints` and `analysisResults.maxRiskPoints` only. Matching is by `String.startsWith`, not equality (lines 1980–1981).

Set	Line	Members	Producer keys as registered
`DEMAND`	1961–1962	`enrollment`, `nettuitionperfte`, `acceptanceyield`, `retention`, `programssustainability`	all five exist and match exactly
`RESERVES`	1963	`primaryreserve`, `viability`	both exist and match exactly

All seven names are spelled to match the keys `analysis_orchestrator.js` line 219 actually produces (`UniversalProgramsSustainabilityAnalyzer` → `programssustainability`, with the double-s). No unpublished field is read.

Gate logic

Constant / step	Line	Rule	Direction
Guard	1977	`if (rp === null || mx === null || mx <= 0) continue`	record skipped entirely
`ratio`	1978	`rp / mx`	—
`DEMAND_HIT = 0.60`	1965	demand analyzer counts as failing at `ratio >= 0.60`	**floor**
`RESERVE_OK = 0.40`	1966	every reserve ratio must be `<= 0.40`	**ceiling**
`MIN_DEMAND = 3`	1967	`hits.length >= 3`	**floor**
`reservesIntact`	1984–1985	`reserveRatios.length > 0 && reserveRatios.every(v => v <= 0.40)`	**floor** on count, **ceiling** on each ratio
Trigger	1986	`hits.length >= MIN_DEMAND && reservesIntact`	—

Unlike A10 and A12, A14 works entirely in **ratios of max**, which makes it robust to the sector-varying maxima (Brand Pipeline 8 vs 10, and so on).

Return value carries one extra key: `demandKeys: hits` (line 1994).

- **Sector differences**: the header comment declares the gate "naturally sector-selective" and attributes it to GASB-depressed public reserves. Two structural facts reinforce it: 1. `UniversalAcceptanceYieldAnalyzer` is **excluded for community colleges** (`analysis_orchestrator.js` line 46), so sector 4 has only four candidate demand measures against `MIN_DEMAND = 3`. 2. Nothing else is sector-gated — `primaryreserve` and `viability` are scored in all four sectors, so `reservesIntact` is evaluable everywhere.

Baseline: 82 firings — 61 private, 21 public.

- **Data-gap behavior**: **fails closed on both halves, and unusually firmly.**
  - A demand analyzer with a null `rp`/`mx` is skipped (line 1977) and simply never becomes a hit.

    - `reservesIntact` requires `reserveRatios.length > 0` (line 1984). If **neither** primary reserve nor viability produced a usable rp/mx, the array is empty and `reservesIntact` is `false` — the gate closes rather than assuming reserves are fine. This is the correct direction for an 8-point award and is stricter than A12's equivalent handling.
    - `Array.prototype.every` on a non-empty array of real ratios has no vacuous-truth hazard here because of that length guard.
- **2026-08 changes**: no dated comment. The header (lines 1950–1957) is undated.

Notes

- **The header's sector figures do not match the validated 2026 baseline.** The comment at lines 1956–1957 claims *"fires 19% in C21 private, 10% in C22, 1% in C18"*. Recomputed from `baseline_recs.json` over the 1,401 scored institutions:

Cell	Comment	Baseline
C21 private	19%	**10%** (19/187)
C22 private	10%	**5%** (8/161)
C18 private	1%	**8%** (11/146)
C18 public	—	8% (11/135)

The comment is stale relative to the current baseline. Reported, not corrected; the direction of the claim (private-selective, C21-heaviest among privates) still holds, but the magnitudes and the C18 claim do not.

- **`startsWith` is a looser match than it looks.** Line 1980 uses `DEMAND.some(d => k.startsWith(d))`. Against the current registry no key is a prefix of another within these sets, so behavior is exact today. Adding an analyzer whose key begins with `enrollment`, `retention` or `acceptanceyield` would silently add a demand hit.
- **8 points against `hits.length >= 3` of a maximum 5 demand measures** makes A14 the most leveraged single test in the layer. No cap or taper is applied for institutions that clear the bar with 4 or 5 hits.
- Like A8, A10 and A12, A14 makes no `getMetricValue` calls and is therefore stamped `'not_reached'` by `_annotateGateStatus` on every non-firing institution.

R10 — Resilient by Scale and Diversification

Points -8 · fires for 285 of 1,401 (20.3%)

Update (v49.10, 15 October 2026). R10 reads appropriations dependency only for publics. Privates are evaluated on tuition dependency, as before, and the line `not_applicable — analyzer absent: stateappropriations.dependency` no longer prints for them. No private score changed. **Update (v49.17, 19 October 2026).** The unscored flag U39 marks R10 held while the structural margin's three-year average is −10% or worse, or while A10 or P9 fires. R10 keeps its −8; see the entry after P9.

Display name from ARCHETYPES (line 151):

```
R10: 'Resilient by Scale and Diversification',
```

- **Method**: `checkR10_ResilientScale()` (lines 2056–2075); shared helper `__hasResilientScale()` (lines 2021–2054)
- **Points**: `-8` — from ARCHETYPE_BASE_POINTS (line 195):

```
A12: 6, A13: 4, A14: 8, R10: -8, R11: -4, R12: -6  // resilience reduces risk; LA1/LA2 removed 2026-08-28
```

Applied as `const points = triggered ? (ARCHETYPE_BASE_POINTS?.R10 ?? -2) : 0;` (line 2072). The inline default `-2` is dead while the constant is defined; it does not match the constant.

- **What it tests**: an institution that is (a) large for its own type, (b) not dependent on a single revenue source beyond half of revenue, and (c) holding either a large per-student unrestricted cushion or low risk scores on both balance-sheet reserve measures. All three legs are required — the conjunction is `largeScale && diversified && cushionOK` (line 2071).
- **Inputs**:

Variable	Call	Producer analyzer	Route
enrollNow (inside helper)	`getMetricValue('enrollmenttrends','currentTotal') ?? getMetricValue('enrollmenttrends','total')`	`unified_enrollment_analyzer.js`, published as `analysisResults.currentTotal` (line 113)	`analyzerAliases` has no `enrollmenttrends` key, so the resolver's exact-key pass misses and the **fuzzy pass** matches (`'enrollmenttrends'.includes('enrollment')`, `_getMetricValueRaw` lines 863–867). `currentTotal` then resolves at step 4 (exact published name). The `'total'` fallback resolves through the declared `metricAliases.total = ['total','currentTotal','currentValue']` (lines 570–572, added 2026-08-30 v49.2)
peerMedian (inside helper)	**direct read**, not `getMetricValue`: `enrAnalysis.analysisResults.peerMedianTotal`	`unified_enrollment_analyzer.js` line 121, `peerAnalysis?.averages?.medianCurrentTotal ?? null`	analyzer located by an inline strict normalized match on `analyzerKey ∈ {enrollment, enrollmenttrends, enrollmentanalysis}` (lines 2031–2034)
peerN (inside helper)	direct read: `analysisResults.peerMedianTotalCount`	`unified_enrollment_analyzer.js` line 122, `peerAnalysis?.averages?.peerCurrentTotalCount ?? 0`	as above
tuitionDep	`pct(getMetricValue('tuitiondependency','tuition_percentage'))`	`unified_tuition_dependency_analyzer.js`, `analysisResults.currentValue` (line 132), a **ratio on 0–1**	`keyMappings['tuition_percentage'] = 'currentValue'` (line 912)
appropsDep	`pct(getMetricValue('stateappropriations','dependency'))`	`unified_appropriations_analyzer.js`, `analysisResults.currentValue = appropriationsAnalysis.currentDependency` (line 127), read as a raw **percentage** from year columns 2010–2024	`analyzerAliases.stateappropriations` includes `'appropriations'`, which is the orchestrator's derived key for `UniversalAppropriationsAnalyzer`; `keyMappings['dependency'] = 'currentValue'` (line 915)
unaFTE	`this.resolveUnaPerFTE()` (lines 701–713)	`unrestricted_net_assets_analyzer.js`,	direct read of the analyzer whose normalized key is

Variable	Call	Producer analyzer	Route
		analysisResults.una_per_fte (line 180)	unrestrictednetassets; una_per_fte is published, so the derived total / enrollment fallback in resolveUnaPerFTE is not normally reached
primRisk	num(getMetricValue('primaryreserve','riskPoints'))	unified_primary_reserve_analyzer.js, scale 0–10 (maxRiskPoints: 10, line 52)	exact key match; riskPoints resolves at resolver step 4
viabRisk	num(getMetricValue('viability','riskPoints'))	unified_viability_analyzer.js, scale 0–8 (this.maxRiskPoints = 8, line 42)	exact key match; step 4

this.pct() (lines 658–662) normalizes to 0–100 via return (v <= 1 ? v * 100 : v). This reconciles the tuition analyzer's 0–1 ratio with the appropriations analyzer's 0–100 percentage.

- **Gate logic**:

Sector selector (line 2065):

```
const dep = (this.sector === '1') ? appropsDep : tuitionDep;
```

Leg	Condition	Direction	Source line
largeScale	scale.ok — see the scale table below	—	2066
diversified	dep !== null && dep <= 50	**ceiling** ≤ 50 %	2067
cushionOK (leg A)	unaFTE !== null && unaFTE >= 40000	**floor** ≥ $40,000 per student	2068
cushionOK (leg B)	primRisk !== null && viabRisk !== null && primRisk <= 2 && viabRisk <= 2	**ceiling** ≤ 2 on each	2069
Trigger	largeScale && diversified && cushionOK	—	2071

cushionOK is leg A **OR** leg B. Leg B's ≤ 2 is against different denominators for the two measures — 2 of 10 for primary reserve, 2 of 8 for viability — so it is a stricter test of primary reserve (top 20 %) than of viability (top 25 %).

The scale bar — __hasResilientScale() (lines 2021–2054):

Step	Condition	Result
1	enrollNow === null	{ ok: false, basis: 'no enrolment value' } — hard stop
2	peerMedian === null || peerMedian <= 0 || peerN < 10	{ ok: enrollNow >= 10000 } — **floor** 10,000, absolute fallback. peerN is a **floor** of 10 usable peers
3	otherwise	ok = (enrollNow >= peerMedian) || (enrollNow >= 10000) — **floor** at the peer median, OR **floor** at 10,000

What the peer median is measured over, traced into the producer: medianCurrentTotal is the median 2024 twelve-month FTE enrollment of the institution's peer group (unified_enrollment_analyzer.js lines 520–525), where the peer group is built by findAndAnalyzePeerGroup() (lines 411–464) as:

- this.institutions, which extractAllInstitutions() filters to **the selected sector only** (if (unitId && sector === this.sector), line 73); then
- self excluded (inst.unitId !== institution.unitid, line 422); then
- filtered to peerCriteria.carnegieBasic when supplied as an array (lines 424–430); then
- optional HBCU and custom-peer-group filters (lines 431–445).

So the comparison is **sector × Carnegie**, i.e. one of the sixteen cells the change comment names. Peers with no currentTotal are excluded from the median and from peerCurrentTotalCount (lines 490–494), so peerN counts peers with usable enrollment, not peers found.

This is not only a wording correction: R10 and R12 both test institutional scale against this value, so the resilience gates have always been comparing full-time-equivalent enrollment while the documentation described headcount. The gates are correct; the description was not.

The median is deliberately a median, per the producer comment (lines 517–519):

> *// Median, not mean: peer enrollment distributions are right-skewed in every // segment, so a mean bar would sit above most of the group.*

- **Sector differences**:
  - The dependency leg switches producer on `this.sector === '1'` only: **public** institutions are tested on appropriations dependency, **every other sector value** (`'2'`, `'3'`, `'4'`, and the `String(sector || '1')` catch-all set in the constructor at lines 391–395) is tested on tuition dependency. The ≤ 50 % ceiling is identical for both.
  - The scale bar is implicitly sector-differentiated because the peer group is sector-filtered upstream in the enrollment analyzer, not because `__hasResilientScale()` branches — it does not.
  - The cushion thresholds ($40,000; ≤ 2 / ≤ 2) do not vary by sector.
- **Data-gap behavior**: **fails closed on every input.** Each leg uses an explicit `!== null` guard before its comparison, so a null makes that leg false and the credit is withheld.
  - `enrollNow` null → `ok:false` → no credit.
  - `dep` null → `diversified` false → no credit.
  - `unaFTE` null **and** either reserve risk null → `cushionOK` false → no credit.
  - A missing or thin peer median does **not** fail open: it reverts to the absolute 10,000 floor (step 2), which is the *stricter* bar for every institution type whose peer median is below 10,000 — i.e. the fallback removes the credit from small-type institutions rather than granting it.
  - Confirmed at the producer end: `unified_primary_reserve_analyzer.js`, `unified_viability_analyzer.js` and `unrestricted_net_assets_analyzer.js` route institution-not-found through `createErrorResult(...)`, which returns `success: false` with `isDataUnavailable: true` (`unified_primary_reserve_analyzer.js` lines 944–973; `unified_viability_analyzer.js` lines 1112–1123), and `analysis_orchestrator.js` line 331 only pushes records where `data.result.success` is true. A not-found analyzer therefore never appears in `completedAnalyses`, so `getMetricValue` returns `null` rather than a real `0`. This is the mechanism that stops R10's `riskPoints <= 2` legs from reading absence as strength — see Notes.
- **2026-08 changes**:

The design comment above `__hasResilientScale()` (lines 1998–2020), verbatim:

> *// R10 - Resilient by Scale and Diversification // ---------- // 2026-08-27: TYPE-RELATIVE SCALE for the resilience gates. // // R10 and R12 both tested enrollment >= 10,000. Measured across the 2026 baseline // (1,456 scored institutions, all sixteen Carnegie x sector cells) that bar was // cleared by 97% of public doctoral very-high-research institutions and by 0% of // public C21, 0% of private C19/C20/C21 and 1% of private C22. The resilience layer // was not testing resilience; it was testing enrollment. // // Replacement: scale is sufficient if the institution is at or above the MEDIAN // enrollment of its own peer group, OR clears the old 10,000 bar outright. The second // clause makes the change monotone — verified against the baseline, zero institutions // holding a credit today lose it — while the first makes "scale" mean large FOR YOUR // TYPE, which is what the concept was always meant to mean. A 2,000-student liberal // arts college is at scale for a liberal arts college. // // The peer median is published by unified_enrollment_analyzer as // analysisResults.peerMedianTotal, computed over the same peer group the rest of the // analysis uses, so it recalibrates on every data refresh with no hard-coded number // to go stale. If it is missing or the peer group is too small, the gate falls back // to the original absolute rule rather than guessing. // ----------*

The repricing comment in `ARCHETYPE_BASE_POINTS` (lines 181–195), verbatim:

> *// 2026-08-27 CHANGES, all measured against the v48.7 baseline (1,456 scored institutions): // // A13 8 -> 4. A13 fires exactly as often as the critical-mass computation is TRIGGERED — // 28/28, 16/16, 51/51 and so on in every one of the sixteen segments. It is a restatement // of that flag under another name, and it carried the highest base-score correlation in // the framework (r = +0.60), i.e. it charged 8 points for risk the measures had already // priced. Critical mass continues to drive escalation through checkCriticalMassEscalation; // A13 stays in the map at four points so the diagnostic remains visible in the export and // nothing downstream that reads archetypeFlags.A13 breaks. // // R10 -3 -> -8, R12 -2 -> -6. Resilience was priced against a layer-2 that averages +22 // points. 201 institutions held a credit and*

> *NINE had a band improved by it. Widening // eligibility (see __hasResilientScale) without repricing would have made a decorative // feature available to more people.*

The strict-read comment inside the helper (lines 2026–2030), verbatim:

> *// Read the two new fields STRICTLY, not through getMetricValue(). That resolver has a // generic-variations fallback which, when a requested field is absent, returns whatever // 'current'/'value'/'ratio' happens to exist on the analyzer — silently substituting an // unrelated number. A scale bar built from an enrollment CHANGE PERCENT instead of an // enrollment COUNT would be catastrophic and invisible. An exact read cannot fail open.*

The matching publication comment in the producer, `unified_enrollment_analyzer.js` lines 114–120, verbatim:

> *// 2026-08-27: PUBLISHED FOR THE RESILIENCE GATES. R10 and R12 previously // hard-coded enrollment >= 10,000 as their scale test, which made resilience // available to 97% of public doctoral institutions and 0% of baccalaureate // colleges. They now read this field. Publishing it here — rather than having // the adjustment layer recompute a peer median — keeps this analyzer the sole // producer of the value, and avoids the recurring defect in this codebase of a // gate reading a field its producer never published.*

And the peer-collection comment, `unified_enrollment_analyzer.js` lines 474–477, verbatim:

> *// 2026-08-27: peerCurrentTotals feeds medianCurrentTotal below, which the // resilience gates R10/R12 read as their type-relative scale bar. Collected // here rather than recomputed downstream so there is exactly one definition // of "the enrollment of a typical peer".*

LA1/LA2 removal, `ARCHETYPES` lines 154–160, verbatim:

> *// 2026-08-28 (v49.1): LA1 'Aid Arms-Race' and LA2 'Small-Scale Fragility' // REMOVED. They were named here and priced at 2 points each in // ARCHETYPE_BASE_POINTS, but were absent from detectArchetypeFlags() and so // never evaluated — the framework documented 4 points of risk it never // charged. Their only implementation lived in detectArchetypes(), deleted // 2026-07-27. Restore them by writing checkLA1_/checkLA2_ methods and // registering them, not by re-adding the names.*

and the trailing marker on the points line (line 195), verbatim:

> *// resilience reduces risk; LA1/LA2 removed 2026-08-28*

LA1 and LA2 are absent from `ARCHETYPES`, from `ARCHETYPE_BASE_POINTS` and from `__gateFnByKey`. They survive only as display strings in `modular_tool_v48 no data cap.html` line 3899 (`'LA1': 'LA1 Aid Arms-Race', 'LA2': 'LA2 Small-Scale Fragility'`), where they can never be populated.

- **Notes**:

1. **What produces the observed distribution.** Two of R10's three legs are effectively type-neutral and one is not, and the scale leg is the *weakest* discriminator of the three:
    - **The scale bar is close to a coin-flip within any cell.** Because the bar is the peer group's own median (step 3), roughly half of any sector × Carnegie cell clears it by construction, whatever that cell's absolute enrollment. The `|| enrollNow >= 10000` clause can only push a cell **above** 50 % — it admits institutions below their own median that still exceed 10,000, which is possible only in cells whose median is above 10,000, i.e. the large public doctoral cells. Ties count (>=), and self-exclusion from the median adds a small upward bias for any institution in a small cell. A cell firing at 62 % on scale alone is consistent with the median bar plus the absolute clause; a cell firing at 2 % is **not** explicable by the scale bar at all.
    - **The separating legs are `diversified` and `cushionOK`.** `dep <= 50` is read against tuition dependency for privates. Larger master's privates are, structurally, the tuition-dependent segment; a tuition dependency above 50 % fails the leg outright regardless of scale. Private R1s (C15) carry large non-tuition revenue and large per-student unrestricted net assets, so they pass both `dep <= 50` and `unaFTE >= 40000`.
    - **Publics are tested on a different quantity.** For sector `'1'` the dependency leg is appropriations dependency, which is typically well under 50 %, so publics generally pass that leg; their binding constraint is `cushionOK`, where `unaFTE >= 40000` is rare and leg B requires ≤ 2 of 10 on primary reserve *and* ≤ 2 of 8 on viability — a demanding conjunction for public balance sheets under GASB 68/75 (see `spec_A.md` on the viability analyzer's sector branch).

UNCLEAR: the code establishes the *mechanism* of the distribution but not the arithmetic split. Attributing the 62 % / 44 % / 2 % figures to specific legs requires the per-leg firing counts from the baseline run; those counts are not in the source and are not derivable from it. What the code does establish unambiguously is that the scale bar cannot be the cause of the 2 % master's-private figure, because a median bar cannot exclude 98 % of the cell it is computed from.

2. **unaFTE is not per FTE.** unrestricted_net_assets_analyzer.js computes unaPerFte(unitId, totalUna) as totalUna / this.getEnrollment(unitId) (lines 537–541), where getEnrollment reads a headcount map (this.enrollmentByUnitId, lines 531–535). The $40,000 threshold is therefore dollars of unrestricted net assets **per enrolled student**, not per FTE, despite the field name una_per_fte and the local variable unaFTE.

3. **pct() boundary hazard on the dependency leg.** pct() treats any value <= 1 as a ratio and multiplies by 100. The appropriations analyzer publishes a percentage, so a genuine appropriations dependency of, say, 0.8 % would be converted to 80 % and would fail diversified. This is a latent mis-read at the low end of the appropriations distribution only; it is directionally conservative for a credit (it withholds points), so it cannot give points away.

4. **No riskLevel guard on the cushion leg.** R11 carries an explicit data guard rejecting not_applicable / not_found risk scores (lines 2094–2104) because a real 0 from an analyzer that could not find the institution was being read as strength. R10's cushionOK leg B reads primaryreserve.riskPoints <= 2 and viability.riskPoints <= 2 with **no equivalent guard**. In the current pipeline this does not fail open, because both analyzers return success: false on not-found and the orchestrator drops non-success records (line 331), so the read returns null. The protection is therefore incidental to the orchestrator, not intrinsic to the gate: any future producer change that returns success: true with riskPoints: 0 and riskLevel: 'not_applicable' would make R10 hand out -8 on absent data.

5. **The peer-median read is invisible to the gate ledger.** peerMedianTotal and peerMedianTotalCount are read directly off completedAnalyses (lines 2031–2039), bypassing getMetricValue(). _annotateGateStatus() (lines 792–835) classifies a gate from the ledger of getMetricValue calls only, so an R10 that silently reverted to the absolute-10,000 fallback is still reported as evaluated. The strict read was a deliberate defense against the resolver's old fail-open substitution (see the quoted comment), but its side effect is that the scale basis is only recoverable from reason's scale.basis string on a *triggered* gate; a gate that failed the scale test reports only 'criteria not met' (line 2073).

6. **The credit can be discarded whole by the severity override.** applySeverityOverride() line 2248 sets this.totalAdjustmentPoints = Math.min(50, Math.max(severityPoints, archetypePatternTotal)), where severityPoints is 12, 16 or 20. For any institution that trips the severity gate, the archetype total — R10's -8 included — is replaced wholesale if it is the smaller number. A credited institution that is also in severe distress keeps none of the credit.

7. **The ?? -2 inline defaults are stale.** Lines 2072, 2126 and 2146 fall back to -2 if ARCHETYPE_BASE_POINTS is undefined. Those defaults are the pre-2026-08-27 prices for R12 and R11 and were never R10's price. They are unreachable while the constant is defined at module scope, but they would silently under-credit if it were not.

R11 — Research-Strong Anchors

Points -4 · fires for no institution in the baseline

Display name from ARCHETYPES (line 152):

```
R11: 'Research-Strong Anchors',
```

- **Method**: checkR11_ResearchStrong() (lines 2078–2135)
- **Points**: -4 in ARCHETYPE_BASE_POINTS (line 195). **Never applied.** The points expression (line 2126) is `const points = triggered ? (ARCHETYPE_BASE_POINTS?.R11 ?? -2) : 0;` and triggered is a literal false.
- **What it tests**: nominally, a research-oriented institution whose research-funding risk score is low — i.e. federal research support holding up against peers. In the shipped code it tests nothing and returns a constant.
- **Inputs**: every field the method reads, all of them evaluated and then discarded:

Variable	Call	Producer	Route / status
carnegie	this.getCarnegieCode() (lines 626–636)	this.carnegieBasic ?? this.institution?.carnegieBasic ?? this.peerCriteria?.carnegieBasic?.[0], set in the constructor at line 409	returns null and logs a warning when non-finite
isResearchOriented	this.isResearchOriented() (lines 638–640) → isResearchCarnegie(code) (lines 322–330)	—	return n === 15 || n === 16 || n === 17 || n === 18;
rgRisk	num(getMetricValue('research_grants','riskPoints'))	research_grants_analyzer.js (UniversalResearchGrantsAnalyzer → orchestrator key researchgrants), 0–8 scale	normKey('research_grants') === 'researchgrants', exact analyzer match; riskPoints resolves at resolver step 4
reRisk	num(getMetricValue('research_grants_expenses','riskPoints'))	**no analyzer registers this key**	resolves by the resolver's fuzzy pass: 'researchgrantsexpenses'.includes('researchgrants') is true (lines 863–867), so this reads **the same analyzer as rgRisk**
gfRisk	num(getMetricValue('grantfunding','riskPoints'))	unified_grant_funding_analyzer.js (UniversalGrantFundingAnalyzer → key grantfunding), 0–6 scale, student financial-aid coverage	resolves; **deliberately excluded** from researchRisk — diagnostic only
rgLevel, reLevel, gfLevel	getMetricValue(<same keys>,'riskLevel')	as above	used only by the naRe data guard
researchRisk	rgOK ? rgRisk : (reOK ? reRisk : null) (line 2111)	—	derived
hasResearchData	(rgOK || reOK) (line 2112)	—	derived

- **Gate logic**, transcribed exactly:

Data guard (lines 2098–2104):

```
const naRe = /not[_ ]?applicable|not[_ ]?found|unavailable|error|unknown/;
const rgOK = (rgRisk !== null) && !naRe.test(rgLevel);
const reOK = (reRisk !== null) && !naRe.test(reLevel);
const gfOK = (gfRisk !== null) && !naRe.test(gfLevel);
```

Evaluated-but-unused condition (lines 2119–2123) and the shipped result (line 2124):

```
const wouldTrigger = (
  isResearchOriented &&
  hasResearchData &&
  researchRisk !== null && researchRisk <= 2
);
const triggered = false;
```

Leg	Condition	Direction	Line
Carnegie precondition	`isResearchCarnegie(code)` → code ∈ {15, 16, 17, 18}	membership	329
Data present	`rgOK || reOK` — a non-null risk score whose `riskLevel` does not match `naRe`	—	2112
Research strength	`researchRisk <= 2` (of a maximum of 8)	**ceiling** ≤ 2	2122
Applied result	**`false`, unconditionally**	—	**2124**

- **Sector differences**: none inside the method. Indirectly, the research grants analyzer is instantiated by the orchestrator **only for sectors `'1'` and `'2'`** (`analysis_orchestrator.js` lines 159–165), so for community colleges and for-profits the input analyzer is not present at all and `rgOK`/`reOK` would be false.
- **Data-gap behavior**: irrelevant to scoring — the credit is withheld in all cases. Had it been live, it would fail **closed**: `researchRisk !== null` plus the `naRe` guard require an affirmative, applicable research score before the credit is considered. The guard exists precisely because the gate previously failed **open**; see the quoted comment below.
- **2026-08 changes**: R11 was disabled on **2026-07-27**, before the August window. No 2026-08 change touches it. The August-dated comments that bear on it are in its input chain, not the gate.

Header comment on the method (lines 2079–2083), verbatim:

> *// DISABLED 2026-07-27 pending redesign of the research-strength input. // The gate `researchRisk <= 2` reads a federal-grant-dependency measure, // so it fires for institutions with NO research activity and blocks // genuine R1s (Purdue: 8/8 research risk, R11 correctly-but-wrongly false). // Flip R11_ENABLED to true only after the input is replaced.*

The data guard's rationale (lines 2094–2097), verbatim:

> *// An analyzer that reports "Institution Not Found" publishes 0/8 with // riskLevel 'not_applicable' — a real 0, not null. Reading that 0 as a // passing score made R11 fire for 100% of C18 (145/145 not_applicable). // Absence of data is not evidence of research strength.*

The `grantfunding` exclusion (lines 2106–2110), verbatim:

> *// grantfunding EXCLUDED 2026-07-27: it is the Grant Funding Distribution // analyzer (student financial aid coverage, 0-6 scale), not a research // measure. It was backfilling researchRisk for 139/145 C18 institutions // and would have fired R11 for 66 of them on student-aid distribution. // gfRisk/gfOK retained below only for the diagnostic log line.*

The restatement immediately above the disable (lines 2115–2118), verbatim:

> *// R11 DISABLED pending input redesign. Its gate (researchRisk <= 2) reads a // federal grant dependency measure, so it fires for institutions with no // research activity and blocks genuine R1s (Purdue: 8/8 → blocked). // Retained below for diagnostics; forced off until a strength measure replaces it.*

The Carnegie-set comment inside `isResearchCarnegie()` (lines 324–328), verbatim:

> *// 18 (Master's: Larger Programs) RETAINED. C18 institutions do receive // research funding (e.g. UNT Dallas: $5.7M government grants). The 136/136 // R11 firing was NOT caused by this set — it was caused by R11 reading // "Institution Not Found" (0/8, not_applicable) as a passing research score. // Fixed in checkR11_ResearchStrong's data guard.*

The diagnostic export (lines 3401–3410), verbatim:

> *// R11 is disabled; publish what it WOULD have done so the decision to // re-enable rests on a population count, not a console buffer.*

Dated **2026-08-28**, in the input producer `research_grants_analyzer.js` line 82:

> *// 2026-08-28: the scored population. Carnegie 15 and 16 only.*

and line 539:

> *// 2026-08-28: narrowed from [15,16,17] to [15,16]. See the header note.*

- **Notes — why R11 fired zero times across all 1,401 institutions, including all 134 R1s**:

1. **The cause is a hard-coded constant, not a threshold, a missing field or an unsatisfiable precondition.** Line 2124 reads, in full:

```
    const triggered = false;
```

`triggered` is assigned a literal. It is never reassigned, and it is the only thing that gates the points expression on line 2126. Every input the method reads — `carnegie`, `isResearchOriented`, `rgRisk`, `reRisk`, `gfRisk`, the three `riskLevel` strings, the derived `researchRisk` and `hasResearchData` — feeds only `wouldTrigger` (lines 2119–2123), which is computed, logged (line 2131), returned as a diagnostic field (line 2134) and published into `r11Diagnostic` (lines 3401–3410). **`wouldTrigger` is read by nothing that scores.** The firing rate is therefore exactly zero for every institution in every sector and every Carnegie class, by construction, and would remain zero on any conceivable input data. The zero is not a symptom; it is the specified behavior.

2. **The threshold is reachable and the fields do exist.** For completeness, because the question "is a threshold unreachable / is a field never published?" has to be answered on its own terms:

- `research_grants.riskPoints` **is** published (`research_grants_analyzer.js` line 222, `riskPoints: analysisResults.totalScore`; `maxRiskPoints: 8` at line 98) and `researchRisk <= 2` is attainable — the analyzer's own narrative states that "institutions keeping pace with or ahead of their peers score zero" (line 677).
- `research_grants.riskLevel` is published and is not in `naRe` for a scored institution.
- The Carnegie precondition {15, 16, 17, 18} is satisfied by every R1 (C15). So had `triggered` been `wouldTrigger`, R11 would have fired — the comment's own evidence says it fired 136/136 and 145/145 in earlier states of the code. Nothing in the input chain independently forces zero.

3. **A second, independent blocker sits behind the constant** and would bite for C17/C18 even if the constant were removed. `isResearchCarnegie()` admits Carnegie **15, 16, 17 and 18** (line 329), but the producer scores **15 and 16 only** — `this.SCORED_CARNEGIE = [15, 16]` (`research_grants_analyzer.js` line 83), with everything outside it returned by `createNotApplicableResult()` at `riskLevel: 'not_applicable'` (lines 698–712). That level matches `naRe`, so `rgOK` is false, `hasResearchData` is false, and `wouldTrigger` is false for every C17 and C18 institution. The gate's population and its input's population disagree by two Carnegie classes, and the disagreement was *widened* on 2026-08-28 when the producer narrowed from [15,16,17] to [15,16] without a corresponding change to `isResearchCarnegie()`.

4. **`reRisk` is not a second opinion.** `getMetricValue('research_grants_expenses','riskPoints')` has no analyzer of its own; the resolver's substring fallback routes it to `researchgrants`. `rgRisk` and `reRisk` are therefore the same number from the same analyzer, and the `rgOK ? rgRisk : (reOK ? reRisk : null)` chain on line 2111 has no second branch in practice.

5. **The named toggle does not exist.** The header comment instructs: "Flip `R11_ENABLED` to true only after the input is replaced." There is no `R11_ENABLED` identifier anywhere in `risk_adjustment_analyzer.js` or in any other file in the working directory. Re-enabling requires editing line 2124, not setting a flag.

6. **The disable is invisible to the gate-status instrumentation.** `_annotateGateStatus()` sees a ledger record for `checkR11_ResearchStrong` (the method does make `getMetricValue` calls) and an untriggered result, so R11 is classified as `unavailable` or `not_applicable` depending on whether the research analyzer ran — never as "disabled". A reader of the status field cannot distinguish a suppressed element from a data gap.

7. **The framework still advertises the credit.** `ARCHETYPES.R11` and `ARCHETYPE_BASE_POINTS.R11 = -4` are both live, so R11 appears in the archetype catalog and in the published point table as a `-4` credit that cannot be earned. This is the mirror image of the LA1/LA2 defect the 2026-08-28 comment describes — there, 4 points of risk documented but never charged; here, 4 points of credit documented but never granted.

R12 — Brand-Strong Anchors

Points -6 · fires for 289 of 1,401 (20.6%)

Update (v49.14, 19 October 2026). For privates R12 also requires the v2 operating margin's three-year average above −2.7% (`PRIVATE_MARGIN_GATES.distress`). Without it, colleges whose entering classes grew while they lost money (Missouri Valley, Hilbert, Webster) earned the −6 credit. Under the new rule 193 privates hold R12, and none of them closed in 2024–26. When brand and scale pass but the margin fails, the reason reads `not met: operating margin X% at or below -2.7% (v49.14)`. Publics and community colleges are unchanged. The v49.14 private admissions scoring (Part 1) also changes what `accRisk <= 2` selects. Current method: from line 2298.

Display name from `ARCHETYPES` (line 153):

```
R12: 'Brand-Strong Anchors',
```

- **Method**: `checkR12_BrandStrong()` (lines 2137–2149); shared helper `__hasResilientScale()` (lines 2021–2054)
- **Points**: `-6` — from `ARCHETYPE_BASE_POINTS` (line 195). Applied at line 2146, verbatim:

```
  const points = triggered ? (ARCHETYPE_BASE_POINTS?.R12 ?? -2) : 0;
```

The inline `-2` default is the stale pre-repricing price (see Notes).

- **What it tests**: an institution with a strong admissions position (low risk on the acceptance/yield measure) that is also at scale for its type. Two legs, both required.
- **Inputs**:

Variable	Call	Producer	Route
`accRisk`	`num(getMetricValue('acceptanceyield','riskPoints'))`	`unified_acceptance_yield_analyzer.js` (`UniversalAcceptanceRateYieldAnalyzer` → orchestrator key `acceptanceyield`). `maxRiskPoints: selectedSector === '1' ? 10 : 8` (line 34)	`analyzerAliases.acceptanceyield` includes `'acceptanceyield'`; exact match, `riskPoints` at resolver step 4
`scale`	`this.__hasResilientScale()`	see R10 above — enrollment `currentTotal`, `peerMedianTotal`, `peerMedianTotalCount` from `unified_enrollment_analyzer.js`	identical to R10

- **Gate logic**:

```
  const brandStrong = (accRisk !== null && accRisk <= 2);
  const largeScale = scale.ok;
  const triggered = brandStrong && largeScale;
```

Leg	Condition	Direction	Line
`brandStrong`	`accRisk !== null && accRisk <= 2`	**ceiling** ≤ 2	2142
`largeScale`	`scale.ok` — peer-median **floor**, or absolute **floor** 10,000; peer-count **floor** of 10	—	2143, 2021–2054
Trigger	`brandStrong && largeScale`	—	2145

- **Sector differences**:
  - The gate itself does not branch on sector, but its threshold is not sector-neutral in effect. The acceptance/yield analyzer's maximum is **10 for sector `'1'` and 8 for all other sectors** (line 34), so the fixed `<= 2` ceiling is the top 20 % of the scale for publics and the top 25 % for everyone else. R12's brand bar is therefore **stricter for public institutions**, with no comment acknowledging the asymmetry.
  - The scale leg inherits the sector-filtered peer group described under R10.
- **Data-gap behavior**: **fails closed on both legs.** `accRisk === null` → `brandStrong` false. No enrollment value → `scale.ok` false. The acceptance/yield analyzer routes not-found through `createErrorResult('Institution not found in acceptance rate dataset', ...)` (line 66), which returns `success: false` with `isDataUnavailable: true` (lines 875–886), and the orchestrator drops non-success records, so an absent institution yields `null` rather than a real `0` — the same incidental protection described under R10, and, like R10, **without** an explicit `riskLevel`

guard of the kind R11 carries. If the producer ever returned `success: true` with `riskPoints: 0` and `riskLevel: 'not_applicable'`, R12 would award `-6` on absent data.

- **2026-08 changes**:

In-method pointer comment (line 2139), verbatim:

> *// 2026-08-27: same type-relative scale test as R10 — see __hasResilientScale().*

The repricing (`R12 -2 -> -6`) and the type-relative scale rationale are the same two 2026-08-27 comment blocks quoted in full under R10, at `ARCHETYPE_BASE_POINTS` lines 181–195 and above `__hasResilientScale()` at lines 1998–2020. The LA1/LA2 removal note of 2026-08-28 (lines 154–160 and line 195) is likewise quoted in full under R10.

- **Notes**:

1. **R12 is the widest of the three credits.** It requires only two legs where R10 requires three, and it shares R10's scale leg exactly. Any institution triggering R10 that also holds `accRisk <= 2` collects both, for a combined `-14`. Nothing in the code prevents double counting of the shared scale test; the two credits are independent entries in `archetypeFlags` and both are summed by `calculateTotalAdjustmentPoints()` (lines 2952–2955). 2. **The stale `?? -2` default** (line 2146) is the pre-2026-08-27 price. Unreachable while the constant is defined, but it would silently restore the old price if it were not. 3. **Scale basis is only observable on a hit.** As with R10, `reason` reports `scale.basis` only when `triggered` is true; an untriggered R12 returns the bare string `'criteria not met'` (line 2147), so a run cannot distinguish "failed on brand" from "failed on scale" from the published record. 4. **UNCLEAR — the 2026-08-27 monotonicity claim.** The helper's comment asserts the widened scale test is monotone: "verified against the baseline, zero institutions holding a credit today lose it". That is true for the scale leg in isolation (the new test is `relative || absolute`, a strict superset of `absolute`). It cannot be verified from source for R12 as a whole, and the verification itself is not in the repository. The claim is recorded here as the code states it, not endorsed.

P1 — Siena Pattern

Points 5 · fires for 127 of 1,401 (9.1%)

- **Method**: `calculateP1_SienaPattern()` (lines 2426–2451)
- **Points**: **4** base, escalating to **6**. Line 2442 `let pts = 4;`, line 2443 `if ((apps14 !== null && apps14 <= -20) || (enroll5 !== null && enroll5 <= -15)) pts += 2;` There is no third tier.
- **What it tests**: the header comment at line 2427 reads `// Price pressure: very high tuition dependence + demand weakness (privates only)`. A private institution that depends on tuition for at least 70% of revenue **and** shows a demand signal turning down — either applications falling since 2014, or enrollment falling on the worst-case reading. The escalation asks whether that demand weakness is severe rather than mild.

The name is historical. A 2026-08-28 comment above (lines 2274–2280) records that a **second, earlier** definition of this method existed in the same class body, implementing the documented "Siena logic" with a 10+2 award, and that it never executed because a later definition in a class body silently overwrites an earlier one. The surviving 4+2 definition is the one specified here; it does not read retention, and the retained comment "high tuition dependency + weak retention/brand" at line 2274 describes the removed version.

- **Inputs**:

Local	Route	Analyzer	Published field	Scale
`tuitionDep`	`this.pct(this.getMetricValue('tuitiondependency','tuition_percentage'))`	`tuitiondependency`	`keyMappings['tuition_percentage'] → 'currentValue'`	**fraction 0–1** at source (`unified_tuition_dependency_analyzer.js` line 661 prints `currentValue * 100`); `pct()` rescales to 0–100
`apps14`	`this.num(this.getMetricValue('acceptanceyield','applications_change_since_2014'))`	`acceptanceyield`	`keyMappings['applications_change_since_2014'] → 'applicationChangePercent'`	percent change, already ×100 at source (`unified_acceptance_yield_analyzer.js` line 97)
`enroll5`	`this.getEnrollmentDeclineWorst()` (helper, direct `completedAnalyses` read)	`enrollment`	min of `totalChangePercent` (2010→2024) and `shortTerm.totalChangePercent` (2019→2024)	percent

Note the local is named `enroll5` but holds the **worst of a fourteen-year and a five-year** reading, not a five-year reading. See §0.7.

- **Gate logic**, transcribed from lines 2432–2449:

```
if (this.sector !== '2') {
  return { triggered: false, points: 0, reasoning: 'Private pattern only' };
}
if (tuitionDep === null) {
  return { triggered: false, points: 0, reasoning: 'No tuition dependency metric' };
}

const demandWeak = (apps14 !== null && apps14 <= -10) || (enroll5 !== null && enroll5 <= -8);

if (tuitionDep >= 70 && demandWeak) {
  let pts = 4;
  if ((apps14 !== null && apps14 <= -20) || (enroll5 !== null && enroll5 <= -15)) pts += 2;
```

Test	Threshold	Direction	Role
`this.sector !== '2'`	sector `'2'`	equality	front gate
`tuitionDep === null`	—	null check	front gate
`tuitionDep >= 70`	70 (%)	**floor**	required
`apps14 <= -10`	−10 (%)	**ceiling**	one of two, OR
`enroll5 <= -8`	−8 (%)	**ceiling**	one of two, OR
`apps14 <= -20`	−20 (%)	**ceiling**	escalation, OR

Test	Threshold	Direction	Role
`enroll5 <= -15`	−15 (%)	**ceiling**	escalation, OR

Fires when: `sector === '2'` AND `tuitionDep >= 70` AND (`apps14 <= -10` OR `enroll5 <= -8`).

- **Sector restriction**: **Private only.** Line 2432, `if (this.sector !== '2')`. `this.sector` is the normalized string set in the constructor (§0.4). Sector `'1'`, `'3'` and `'4'` all return `triggered: false, points: 0, reasoning: 'Private pattern only'`. Confirmed in the validated baseline: 127 of 902 privates (14.1%), 0 of 499 publics.
- **Data-gap behavior**: **fails closed**, with one qualification.
  - `tuitionDep` null → explicit early return, line 2435. No points.
  - `apps14` null and `enroll5` null → both legs of `demandWeak` are explicitly null-guarded, so `demandWeak` is `false` and the pattern cannot fire. No points.
  - One of `apps14`/`enroll5` null → the surviving leg decides. This is a designed OR, not a gap behavior.
  - The escalation legs are separately null-guarded, so a missing input costs the +2 rather than granting it.
- **2026-08 changes**: the dated comment sits above the *deleted* first definition, at lines 2274–2280:

> *`// P1 - Siena Pattern (Privates only: high tuition dependency + weak retention/brand) // 2026-08-28 (v49.1): the FIRST of two identical-signature definitions of // calculateP1_SienaPattern was removed here. A later definition in a class body // silently overwrites an earlier one, so this block had never executed — the // documented Siena logic and its 10+2 point award were dead code, while the // definition further down (the 4+2 version) was the one actually scoring P1. // The surviving definition is unchanged.`*

The P2 calibration table (line 221) records P1's award and measured discrimination:

> *`//  P1_SienaPattern  4-6  +13.3`*

- **Notes**: 1. **The three lookups are made before the sector gate.** Lines 2428–2430 call `getMetricValue` twice and `getEnrollmentDeclineWorst()` once; the `sector !== '2'` return is at line 2432. For a public institution the ledger therefore records two resolved lookups, and `_annotateGateStatus()` cannot classify P1 as `not_reached` — it will report `evaluated` (or `unavailable`, if `tuitiondependency/acceptanceyield` did not resolve) on an institution where the true reason for the zero is "wrong sector". P3 does the opposite and gates first. This is a reporting defect only; no points are affected. 2. **`pct()` misreads tuition dependency above 100%.** `pct()` rescales only values `<= 1` (§0.5). Tuition dependency is stored as a fraction, so a ratio of `1.05` (tuition revenue exceeding total revenue — possible where another revenue line is negative) passes through as `1.05`, fails `>= 70`, and P1 silently does not fire for the *most* tuition-dependent institutions in the file. A ratio of exactly `1.00` is handled correctly (`1 <= 1` → 100). 3. **`enroll5` is misnamed.** It holds `getEnrollmentDeclineWorst()`. An institution level for five years but down 20% since 2010 satisfies `enroll5 <= -8` on the fourteen-year leg. This is the exact statistic the P2 rebuild rejected as "the wrong statistic for a test of flatness"; P1's test is a decline test rather than a flatness test, so the choice is defensible, but it is not the five-year figure the variable name and the reason string imply. The reason string at line 2447 mentions only tuition dependence and does not state which demand leg fired. 4. The 28% segment concentration cited in the 2026 baseline is reproduced by the code as written: private 17s 37/133 (28%), private 18s 41/146 (28%), against private 21s 3/187 (2%).

P2 — High Discount, Flat Enrollment

Points 4 → 6 · fires for 30 of 1,401 (2.1%)

Current state (v49.25, 28 October 2026)

The rule. P2 now reads the first-year discount itself, from the unscored first-year discount analyzer (data release D3), instead of inferring it from net tuition per FTE. It fires at a private when all three hold: the discount rose 8 points or more from fall 2019 to fall 2023; the entering class over the same falls did not grow (applications × admit rate × yield from the acceptance and yield analyzer, which equals IPEDS ENRLT); and net tuition per FTE is at or below the peer average (getNtrNorm() ≤ 1.00). The award is unchanged: 4, +1 where the rise is 12 points or more, +1 where net tuition per FTE is at or below 80% of the peer average. Maximum 6.

Why the third condition. Without it the rule fired at 110 privates, among them Williams, Pomona, Grinnell and Bryn Mawr: colleges widening need-based aid with net tuition per FTE far above their peers, which is policy, not distress. Back-tested on privates in Carnegie 15–22, with the outcome the bottom quarter of total net tuition growth in the following years, flagged colleges were 1.7, 1.5 and 1.6 times as likely as others to land there (falls 2014–18, 2015–19, 2016–20); without the condition the ratio fell from 1.9 to 1.3. A weak-trend gate was tested and added nothing.

Fallback. Where an institution has no discount for both falls (87 of 1,000 privates in the sector run, Bastyr among the anchors), P2 runs the old test documented below, unchanged, and its reason begins LEGACY basis. A missing data file therefore never removes P2 silently.

Effect. Private Carnegie 15–22 run (headless, 27 October): P2 fires at 68 (35 under the old test; 5 under both). 87 privates changed score, 56 up and 31 down; 8 more had P2 change inside the 50-point adjustment cap. 9 changed band. The composite's separation of the 2024–26 closures is unchanged (AUC 0.854 → 0.852); one closure, Limestone, now fires P2. Anchors: Lawrence loses P2 (75.25 → 71.25/217, Moderate); Oakwood gains +5 inside the cap (147/215, Critical).

Reading it. P2 is still the one pattern whose subjects are not currently distressed. It says present stability is being bought with aid. Its award stays at the bottom of the pattern range for that reason. The text below describes the pre-v49.25 test, which now runs only as the fallback.

- **Method**: `calculateP2_HighDiscount_FlatEnroll()` (lines 2283–2352)
- **Points**: **4** base (`const P2_BASE_POINTS = 4;`, line 225), escalating to **6** by two independent +1 steps. Lines 2342–2344:

```
let pts = P2_BASE_POINTS;
if (trendGap <= -35) pts += 1;
if (ntrNorm !== null && ntrNorm <= 0.80) pts += 1;
```

The award was 8 rising to 10 before 2026-08-29. The constant is isolated at module scope so the award is one line to change.

- **What it tests**: an institution holding its enrollment roughly flat over the recent window while its net tuition per FTE grows materially more slowly than its peer group's over the long series — i.e. a discount that is *deepening year on year in real terms*, not merely a low price level. The price level survives only as a corroborator worth one extra point.
- **Inputs**:

Local	Route	Analyzer	Field	Window / scale
`recent`	`num(this.getEnroll5?.())` — helper → `getMetricValue('enrollment','5_year_change')` → keyMappings deep path	`enrollment`	`shortTerm.totalChangePercent`	2019→2024 (2018 fallback), percent
`own`	**direct `completedAnalyses` read**, line 2318	`nettuitionperfte`	`analysisResults.percentChange`	2010→2024, percent (`net_tuition_per_fte_analyzer.js` line 271)
`peer`	**direct `completedAnalyses` read**, line 2318	`nettuitionperfte`	`analysisResults.peerPercentChange`	peer **median** of the same 2010→2024 percent change (producer lines 160–161)
`ntrNorm`	`num(this.getNtrNorm())` — helper, direct read	`nettuitionperfte`	`net_tuition_per_fte_normalized`, else `currentValue / peerAverage`	ratio, ~1.0 at peer average

The direct read, lines 2318–2321:

```
const ntrRes = this.completedAnalyses?.find(a =>
    this.normKey(a.analyzerKey || '') === 'nettuitionperfte')?.analysisResults || {};
```

```
  const own  = num(ntrRes.percentChange);
  const peer = num(ntrRes.peerPercentChange);
```

num here is the local shadow declared at line 2306, `const num = v => (typeof v === 'number' && isFinite(v)) ? v : null;` — **stricter** than the class `this.num`: it rejects numeric strings rather than parsing them.

Only recent passes through getMetricValue, so P2 registers exactly **one** ledger entry.

- **Gate logic**, lines 2326–2344:

```
  if (recent === null || own === null || peer === null) {
    return { triggered: false, points: 0,
             reasoning: 'Insufficient data (needs recent enrolment change and the net-tuition trend against
peers)' };
  }

  const FLAT_BAND = 5;        // +/- 5% over five years counts as holding steady
  const TREND_SHORTFALL = -20; // percentage points behind peer net-tuition growth
  const flatEnroll = Math.abs(recent) <= FLAT_BAND;
  const trendGap = own - peer;
  const deepeningDiscount = trendGap <= TREND_SHORTFALL;

  if (!(flatEnroll && deepeningDiscount)) {
    return { triggered: false, points: 0, reasoning: 'No progressive-discount pattern' };
  }
```

Test	Threshold	Direction	Role
this.sector !== '2'	sector '2'	equality	front gate, line 2302
recent/own/peer null	—	null check	front gate
Math.abs(recent) <= 5	±5 (%)	**two-sided band** (ceiling on absolute value)	required
trendGap = own − peer <= -20	−20 (percentage points)	**ceiling**	required
trendGap <= -35	−35 (pp)	**ceiling**	+1
ntrNorm <= 0.80	0.80 (ratio)	**ceiling**	+1

Fires when: sector === '2' AND all three primary inputs non-null AND |recent| <= 5 AND own − peer <= -20.

Note flatEnroll is the only **two-sided** test among P1–P5: growth of +4% qualifies as readily as a decline of −4%. Growth of +6% disqualifies.

- **Sector restriction**: **Private only.** Lines 2302–2304, and this gate is the **first** statement in the method body, before any lookup:

```
  if (this.sector !== '2') {
    return { triggered: false, points: 0, reasoning: 'Private institutions only' };
  }
```

Non-privates therefore record no ledger entry and are correctly annotated not_reached. Confirmed in the validated baseline: 30 of 902 privates (3.3%), 0 of 499 publics; 2.1% of the 1,401 pool.

- **Data-gap behavior**: **fails closed on all three primary inputs.** Any one of recent, own or peer null → explicit return at line 2326 with a reason string naming the missing quantity. In particular:
  - NTR analyzer absent, or present but failed (so never pushed onto the record) → ?.analysisResults || {} yields {} → own/peer null → no fire.
  - peerPercentChange null (zero peers, or no peer with a computable 2010 base) → no fire.
  - ntrNorm null → the **corroborator is simply not applied**: the ntrNorm !== null guard on line 2344 means a missing level costs the +1 rather than granting it, and the reason string omits the level clause (line 2350). This is the conservative direction.
- **2026-08 changes**: three dated records. First, the calibration comment attached to P2_BASE_POINTS, lines 198–224, quoted **in full** as the design-intent record:

> *// 2026-08-29: P2's award, isolated so it is one line to change. It was 8 rising to 10; // it is now 4 rising to 6, which places it at the bottom of the pattern range alongside // P1. The placement is deliberate and the reasoning is on the record. /// // P2 is the only pattern in the layer whose subjects are not currently distressed. Flat // enrolment is a powerful present-tense protective signal: across 902 private // institutions, those holding enrolment within +/-5% average 32.2%*

> *composite against // 44.5% for the rest. Requiring a deepening discount pulls that group back toward the // average but cannot push it above - every variant tested still lands 3 to 5 points // BELOW the population. So this identifies institutions whose present stability is // purchased, and the risk it names has not yet arrived. // // That is a legitimate thing for a risk framework to score - a tool that waits until the // margin has gone is not forecasting anything. But it earned 8 to 10 points, the second // most generous award in the layer, on the weakest present-tense evidence in it: // // pattern award discrimination (mean gap) // P3_AppropsDep_EnrollDecl 10-12 +17.2 // P2 (old) 8-10 -2.2 // P5_UNA_Risk 6-8 +21.3 // P7_BrandPositioning 6-8 +14.8 // P4_NetTuitionRisk 5-7 +11.1 // P8_MarketPosition 5-7 +15.6 // P1_SienaPattern 4-6 +13.3 // // 4 keeps the signal without letting a forecast outweigh measured condition. Confirm the // new group's discrimination on the next full re-run before treating this as settled.*

Second, the rebuild rationale in the method body, lines 2284–2301:

> *// INTENT, as stated by D. Greenstein 29 August 2026: institutions maintaining roughly // flat enrolment, but doing so by PROGRESSIVELY discounting. They are at risk. // // 2026-08-29 REBUILD. The old gate tested the discount LEVEL - net tuition per FTE at // or below 85% of the peer average - and paired it with flat enrolment. A level is not // a progression: an institution that has always been inexpensive relative to its peers // and holds its enrolment steady is not deepening a discount, and it was being charged // up to 10 points for being cheap. The word the code was missing is "progressively". // // It also read enrolment through getEnrollmentDeclineWorst(), the most negative figure // across two different windows, which is the wrong statistic for a test of flatness - // an institution down 30% over fourteen years but level for the last five is exactly // the case this pattern is about, and the worst-case reading disqualified it. // // Now: flat over the recent window, AND net tuition per FTE growing materially slower // than peers over the long series. Median peer growth across 902 private institutions // is +33.0%; the institutions this selects have a median of +0.4%, a shortfall of // 31.7 percentage points. That is a discount deepening year on year in real terms.*

Third, the reason the record is read directly rather than through the resolver, lines 2311–2317:

> *// 2026-08-29: read the net-tuition record DIRECTLY rather than through // getMetricValue. The resolver does fuzzy name matching, and asked for // 'peerPercentChange' it returned peerAverage - the peer average net tuition per // FTE, in DOLLARS. P2 was therefore comparing a percentage against a dollar amount // and computing shortfalls of twenty thousand "percentage points", which fired the // pattern for 63 of 185 private baccalaureate institutions on the first run. // The fields wanted here are unambiguous and published on the record; take them.*

The same incident is the worked example in the resolver's own fail-closed comment, lines 1000–1007:

> *// This is not a hypothetical. It is how P2 came to compute discount shortfalls of // −20,000 percentage points: it asked net tuition for peerPercentChange, which // was not published, and got peerAverage — dollars — instead. And it is still // live in checkA5_HighCostStalling, whose fallback chain asks net tuition for // growthRate (never published; resolves to currentValue, i.e. net tuition per // FTE in DOLLARS) and tuition dependency for trendDirection (published only as // trends.direction; resolves to currentValue, i.e. the dependency ratio). Both // are then tested against <= 0 as if they were a trend.*

And the producer-side change that made the direct read possible, `net_tuition_per_fte_analyzer.js` lines 155–161:

> *// 2026-08-29 (v49.1): the peer trend was computed and shown in the report but never // published on the record layer 2 reads, so no gate could compare an institution's // net-tuition trajectory with its peers'. P2 needs exactly that: "progressively // discounting" is a statement about the TREND relative to peers, not the level. // 2026-08-29 (v49.1): the MEDIAN, not the mean - see percentChangeMedian above.*

- **Notes**: 1. **The two windows are different lengths and this is deliberate.** `recent` is a five-year enrollment change; `own` and `peer` are fourteen-year (2010→2024) net-tuition changes. The rebuild comment states the design as "flat over the recent window, AND net tuition per FTE growing materially slower than peers over the long series", so this is intended — but a `trendGap` of −20 accumulated over fourteen years is roughly −1.4pp per year, and the reason string (line 2349) reports the enrollment figure as "over five years" while reporting the tuition figures without a window at all. A reader of the output cannot tell the two spans apart. 2. **The analyzer-key match is strict equality where every sibling helper is loose.** Line 2319 tests `normKey(a.analyzerKey) === 'nettuitionperfte'`; `getNtrNorm()` (line 1144) accepts that plus `includes('nettuition')` plus `includes('ntr')`. It resolves today because the wrapper class is `UniversalNetTuitionPerFTEAnalyzer` and the orchestrator's `.replace('universal','').replace('analyzer','')` yields exactly `nettuitionperfte`. Note that the *unwrapped* class in `net_tuition_per_fte_analyzer.js` is named `UnifiedNetTuitionPerFTEAnalyzer` — "Unified", not "Universal" — and the orchestrator strips only "universal", so if that class were ever registered

directly the key would be `unifiednettuitionperfte`, `getNtrNorm()` would still resolve it and P2 would **silently return "Insufficient data" for every institution in the run**. There is no assertion or diagnostic that would surface that. 3. *`ntrNorm <= 0.80` is a discount* level *threshold reintroduced as an escalator. The rebuild comment says the level is "retained as a corroborator rather than as the test" (line 2323), and that is what the code does — but the retained cut-point is 0.80, not the 0.85 the old gate used, and nothing in the record explains the change from 0.85 to 0.80. 4. Local `num` shadows `this.num`.* Line 2306 declares a stricter parser used for all four inputs in this method only. A net-tuition field published as the string `"46.2"` would resolve under `this.num` but is rejected here. No producer currently publishes these as strings. 5. `P2_BASE_POINTS` is referenced exactly once (line 2342). The escalators are hard-coded integers, so "one line to change" applies to the base only; the ceiling of 6 is set by two literal `+= 1` statements. 6. The closing instruction of the calibration comment — "Confirm the new group's discrimination on the next full re-run before treating this as settled" — is an open action item as of this extraction.

P3 — Appropriations Dependence with Enrollment Decline

Points 5 · fires for 122 of 1,401 (8.7%)

- **Method**: `calculateP3_AppropsDepEnrollDecl(severity)` (lines 2354–2379)
- **Points**: **10** base, escalating to **12**. Lines 2370–2371:

```
let pts = 10;
if ((approps !== null && approps >= 50) || (enroll5 !== null && enroll5 <= -25)) pts += 2;
```

This is the largest award in the pattern layer, and the calibration table at line 215 records it as the best-discriminating: `// P3_AppropsDep_EnrollDecl 10-12 +17.2`.

- **What it tests**: a public institution structurally dependent on state appropriations (or, equivalently, one that raises very little of its revenue from tuition) that is also losing enrollment. The exposure is to a single funder whose allocation formulas are usually enrollment-driven, so the two conditions compound.
- **Inputs**:

Local	Route	Analyzer	Field	Scale
approps	this.pct(this.getFirstMetricValue(['stateappropriationsdependency','stateappropriations'], ['dependency','state_appropriations_dependency','appropriations_dependency']))	appropriations	keyMappings['dependency'] → 'currentValue' = appropriationsAnalysis.currentDependency	**0–100 percent** at source (unified_appropriations_analyzer.js line 73 bands are [[70,9],[50,7],[30,5],[20,3]])
tuitionDep	this.pct(this.getFirstMetricValue(['tuitiondependency'], ['tuition_percentage','dependency']))	tuitiondependency	'currentValue'	**fraction 0–1** at source; pct() rescales
enroll5	this.getEnrollmentDeclineWorst()	enrollment	min of 14-year and 5-year total change	percent
severity.hasSeverity	argument from calculatePatternAddons	—	see §0.2	boolean

getFirstMetricValue() (lines 2835–2843) iterates **analyzer keys in the outer loop, metric keys in the inner loop**, returning the first non-null. Neither 'stateappropriationsdependency' nor 'stateappropriations' is the actual analyzer key (appropriations); both resolve through _getMetricValueRaw's substring matching at lines 863–867 ('stateappropriationsdependency' contains 'appropriations'), and 'stateappropriations' additionally has an explicit alias list at line 506 that ends in 'appropriations'.

- **Gate logic**, lines 2355–2377, transcribed in full:

```
if (this.sector !== '1') return { triggered: false, points: 0, reasoning: 'Publics only' };
...
const depOK = (approps !== null && approps >= (severity?.hasSeverity ? 35 : 40))
           || (tuitionDep !== null && tuitionDep <= 20);

if (depOK && enroll5 !== null && enroll5 <= -15) {
    let pts = 10;
    if ((approps !== null && approps >= 50) || (enroll5 !== null && enroll5 <= -25)) pts += 2;
```

Test	Threshold (no severity)	Threshold (severity)	Direction	Role
this.sector !== '1'	sector '1'	same	equality	front gate
approps >= …	**40** (%)	**35** (%)	**floor**	depOK leg A
tuitionDep <= 20	20 (%)	same	**ceiling**	depOK leg B
enroll5 <= -15	−15 (%)	same	**ceiling**	required
approps >= 50	50 (%)	same	**floor**	escalation, OR
enroll5 <= -25	−25 (%)	same	**ceiling**	escalation, OR

Fires when: sector === '1' AND depOK AND enroll5 non-null AND enroll5 <= -15.

- **How severity changes behavior**: exactly one threshold moves. severity.hasSeverity === true lowers the appropriations-dependency **floor from 40% to 35%** — a five-point widening of leg A of depOK. It does not touch leg

B, the enrollment requirement, the escalation, or the award. An institution whose appropriations dependency is in [35, 40) qualifies only under severity; one at or above 40 qualifies either way.

The same 35/40 pair appears in `applySeverityOverride()` at line 2191 (`approps >= (sev.hasSeverity ? 35 : 40)`), so the two gates are calibrated together.

Redundancy worth flagging: `hasSeverity` is true when `enroll5_short <= -25` OR `viability <= 0.30`. P3 separately requires the *worst* enrollment reading `<= -15`. Any institution whose five-year decline is 25% or worse necessarily has a worst-case reading of −25% or worse, so it already satisfies both the base gate and the escalation. The 35% relaxation therefore only reaches institutions that qualified for severity through the **viability** leg, or through a five-year decline in [−25, −15] combined with weak viability. It is not "widen the gate for institutions in trouble" so much as "widen it for institutions with a weak viability ratio".

- **Sector restriction**: **Public only.** Line 2355 is the **first** statement in the method, before any lookup:

```
if (this.sector !== '1') return { triggered: false, points: 0, reasoning: 'Publics only' };
```

Non-publics record no ledger entry and are correctly annotated `not_reached`. Confirmed in the validated baseline: 122 of 499 publics (24.4%), 0 of 902 privates; 8.7% of the 1,401 pool. Segment concentration reproduces the reported 60%: public 21s 9/15 (60%), public 20s 22/37 (59%), against public 15s 2/95 (2%).

A second, softer sector effect: the appropriations analyzer returns `riskLevel: 'not_applicable'` with `currentValue: 0` for private institutions (`unified_appropriations_analyzer.js` lines 669–675). Were the sector gate ever removed, every private would read `approps = 0` — failing leg A but not erroring.

- **Data-gap behavior**: **fails closed**, with a structural caveat.
  - `approps` null AND `tuitionDep` null → `depOK` is false (both legs null-guarded) → no fire.
  - `enroll5` null → the `enroll5 !== null` conjunct fails → no fire.
  - `approps` null but `tuitionDep <= 20` → **the pattern still fires at 10 points**, and the escalation's `approps >= 50` leg can never be reached, so the ceiling is effectively 10 unless the enrollment leg escalates. Leg B is not a data-gap fallback by construction — it is a deliberate second definition of appropriations dependence — but it does mean the pattern's name can be satisfied without ever reading an appropriations figure.
  - The reason string, line 2375, is the fixed literal `'Appropriations dependence + enrollment decline'`. It reports no values and does not say which leg of `depOK` fired, so an institution that qualified purely on low tuition dependency is reported as appropriations-dependent.
- **2026-08 changes**: **none.** There is no dated comment inside `calculateP3_AppropsDepEnrollDecl`. Its inputs changed underneath it on 2026-08-28: `keyMappings['5_year_change']` was repointed from the fourteen-year series to `shortTerm.totalChangePercent` (lines 880–900), and `getEnrollmentDeclineWorst()` stopped mixing in the undergraduate/graduate share metrics on 2026-08-29 (lines 2811–2816). Both changes alter what P3 reads without altering P3. The relevant record, lines 880–897:

> *// 2026-08-28 (v49.1): THESE THREE POINTED AT THE FOURTEEN-YEAR CHANGE. // totalChangePercent is ((total[2024] - total[2010]) / total[2010]) * 100 — // the 2010->2024 change. Every gate requesting a "5 year" figure therefore // tested a fourteen-year decline while its comments, its published reason // strings and the architecture documentation all said five years.*

P3 is not one of the three gates named there, because it calls `getEnrollmentDeclineWorst()` directly rather than asking for a "5 year" figure — but its `enroll5` local carries the same misleading name.

- **Notes**: 1. **The escalation's `enroll5 !== null` re-check is dead.** Line 2371 re-tests `enroll5 !== null` inside a block that line 2369 already guarded with `enroll5 !== null`. Always true. Harmless. 2. **`pct()` on appropriations dependency inverts below 1%.** The appropriations analyzer publishes on a 0–100 scale, so `pct()` normally passes the value through — but a genuine dependency of 0.4% arrives as `0.4`, is rescaled to `40`, and clears the 40% floor. A public with essentially no state support would be scored as heavily state-dependent. The band at risk is `0 < approps < 1`; `approps === 0` maps to `0` correctly. 3. **The same `pct()` rule protects `tuitionDep` here rather than harming it**, because tuition dependency is stored as a fraction and leg B is a ceiling test: any fraction in (0,1) rescales correctly to (0,100). 4. `enroll5` is the worst-case reading (§0.7), so a public level for five years but down 18% since 2010 satisfies `<= -15`. Combined with leg B this is the mechanism behind the 59–60% firing rate in the small-public segments: those institutions have both long-run enrollment contraction and low tuition shares.

P4 — Net Tuition Risk

Points 5 → 7 · fires for 485 of 1,401 (34.6%)

- **Method**: `calculateP4_NetTuitionRisk(severity)` (lines 2380–2420)
- **Points**: **5** base, **7** on the severity route, **4** on the data-gap fallback. Line 2411 `const points = severityHit ? 7 : 5;`; line 2396 `return { triggered: true, points: 4, reasoning: 'Net tuition analyzer flagged risk (fallback)' };`. Calibration table, line 219: `// P4_NetTuitionRisk 5-7 +11.1`.
- **What it tests**: net tuition revenue per FTE materially below the peer-group average — a discounting or pricing-power problem. The base route tests the level alone. The severity route accepts a shallower shortfall when a demand-side stressor is present alongside it.
- **Inputs**:

Local	Route	Analyzer	Field	Scale
`ntrNorm`	`this.getNtrNorm()` — helper, direct read (**not** ledgered)	`nettuitionperfte`	`net_tuition_per_fte_normalized`, else `currentValue / peerAverage`	ratio, 1.0 = peer average
`enroll5`	`this.getEnrollmentDeclineWorst()`	`enrollment`	min of 14-year and 5-year total change	percent
`apps14`	`this.num(this.getMetricValue('acceptanceyield','applications_change_since_2014'))`	`acceptanceyield`	→ `applicationChangePercent`	percent
`accept`	`this.pct(this.getMetricValue('acceptanceyield','acceptance_rate'))`	`acceptanceyield`	→ `currentAcceptanceRate`	**UNCLEAR** — see Notes
`yieldR`	`this.pct(this.getMetricValue('acceptanceyield','yield_rate'))`	`acceptanceyield`	→ `currentYieldRate`	**UNCLEAR** — see Notes
`ntrRP` (fallback only)	`this.num(this.getMetricValue('nettuition','riskPoints'))`	resolves via `analyzerAliases.nettuition` (line 474) to `nettuitionperfte`	`riskPoints`	0–6
`severity.hasSeverity`	argument	—	§0.2	boolean

- **Gate logic**, lines 2382–2419:

```
    if (this.sector === '4' || this.sector === 4 ||
        (this.sectorName && this.sectorName.toLowerCase().includes('community'))) {
        return { triggered: false, points: 0, reason: 'P4 not applicable to community colleges (low NTR is
normal)' };
    }
  ...
  if (ntrNorm === null) {
    const ntrRP = this.num(this.getMetricValue('nettuition','riskPoints'));
    if (Number.isFinite(ntrRP) && ntrRP >= 4) {
      return { triggered: true, points: 4, reasoning: 'Net tuition analyzer flagged risk (fallback)' };
    }
    return { triggered: false, points: 0, reasoning: 'No NTR/FTE norm' };
  }

  const paired =
    (enroll5 !== null && enroll5 <= -15) ||
    (apps14  !== null && apps14  <= -10) ||
    (accept  !== null && accept  >= 90)  ||
    (yieldR  !== null && yieldR  <= 10);

  const baseHit     = Number.isFinite(ntrNorm) && ntrNorm <= 0.80;
  const severityHit = !!(severity?.hasSeverity && Number.isFinite(ntrNorm) && ntrNorm <= 0.90 && paired);
```

```
if (baseHit || severityHit) {
  const points = severityHit ? 7 : 5;
```

Test	Threshold	Direction	Role
CC gate	sector `'4'`	equality	front gate
`ntrNorm <= 0.80`	0.80 (ratio)	**ceiling**	`baseHit`
`severity.hasSeverity`	—	boolean	`severityHit`, required
`ntrNorm <= 0.90`	0.90 (ratio)	**ceiling**	`severityHit`, required
`enroll5 <= -15`	−15 (%)	**ceiling**	`paired`, one of four
`apps14 <= -10`	−10 (%)	**ceiling**	`paired`, one of four
`accept >= 90`	90	**floor**	`paired`, one of four
`yieldR <= 10`	10	**ceiling**	`paired`, one of four
`ntrRP >= 4`	4 (of 6)	**floor**	fallback only

Fires when: not sector `'4'` AND (`ntrNorm <= 0.80`, **or** severity with `ntrNorm <= 0.90` and any one paired stressor), **or** the fallback route.

- **How `severity` changes behavior**: it opens a **second, independent route** rather than moving a threshold. Without severity the only route is `ntrNorm <= 0.80`. With severity, an institution in the band `0.80 < ntrNorm <= 0.90` also fires, provided at least one of four demand stressors is present. Severity also **raises the award to 7** whenever `severityHit` is true — including for an institution that already cleared `baseHit`, since `severityHit` is evaluated independently and `ntrNorm <= 0.80` implies `ntrNorm <= 0.90`. So:

ntrNorm	severity	paired	Result
≤ 0.80	no	—	5 points
≤ 0.80	yes	no	5 points
≤ 0.80	yes	yes	**7 points**
(0.80, 0.90]	yes	yes	**7 points**
(0.80, 0.90]	yes	no	0
(0.80, 0.90]	no	—	0
> 0.90	any	any	0

- **Sector restriction**: **none in the 2026 baseline.** The only sector test is the community-college gate at lines 2382–2385, and it reduces to `this.sector === '4'`:
  - `this.sector === 4` (numeric) is **unreachable** — the constructor coerces `sector` to a string (§0.4), so a strict `===` against a number can never be true.
  - `this.sectorName.toLowerCase().includes('community')` is **redundant** — `getSectorName()` returns `'Community College'` only for sector `'4'`, which the first test already caught.

The validated 2026 baseline contains only sectors `public` and `private` (1,401 institutions, 499 public and 902 private; no sector `'3'` or `'4'` rows), so this gate never fires there and P4 is effectively **unrestricted**. Observed: 34.6% overall — 173/499 publics (34.7%) and 312/902 privates (34.6%), i.e. the same rate in both sectors, which is what an unrestricted level test against sector-matched peers should produce.

- **Data-gap behavior**: **fails open on the primary input.** This is the only fail-open path among P1–P5.
  - `ntrNorm` null → the method does **not** return zero. It reads the net-tuition analyzer's own `riskPoints` and awards **4 points** if that is 4 or more out of 6 (lines 2393–2398). Those risk points are computed by the producer from its own bands and are not a peer-normalized level, so the fallback awards pattern points on a quantity the pattern does not otherwise use, with no peer comparison at all. The comment marking it is `// 🎯 Fallback when normalized NTR isn't available` (line 2392).
  - `ntrNorm` null and `ntrRP < 4` or unresolvable → `'No NTR/FTE norm'`, no points.
  - Each of the four `paired` legs is individually null-guarded, so missing stressor data makes `paired` false and costs the severity route. `baseHit` is unaffected by any missing stressor.

    - Because `getNtrNorm()` bypasses `getMetricValue()`, a null `ntrNorm` leaves **no trace in the gate ledger**. On the fallback path the only ledger entry is `nettuition.riskPoints`; on the `'No NTR/FTE norm'` path there may be none at all, in which case `_annotateGateStatus()` reports `not_reached` rather than `unavailable` — the status says the gate short-circuited when in fact its primary input was missing.
- **2026-08 changes**: **none.** There is no dated comment in `calculateP4_NetTuitionRisk`. Its inputs changed underneath it: `getEnrollmentDeclineWorst()` was narrowed on 2026-08-29 (§0.7), and the resolver stopped substituting generic fields on 2026-08-30 (lines 986–1023), which affects the `acceptanceyield` lookups if any of those three keys ever stops resolving.
- **Notes**: 1. **The fallback is a genuine fail-open and is undocumented as such.** It awards 4 of the layer's points without ever comparing the institution to a peer group, on a `riskPoints` figure whose composition P4 does not inspect. It is also the only route in P1–P5 that can award points when the pattern's defining measurement is unavailable. 2. **`accept` and `yield_rate` are on an UNCLEAR scale.** The acceptance/yield analyzer passes the source column through `parseNumericValue` unchanged (lines 799–803) — no `× 100`, no normalization — so whether `currentAcceptanceRate` is `0.75` or `75` depends entirely on the input spreadsheet, which is not in this repository. `this.pct()` reconciles the two encodings for every value **except** those in `(0, 1)` under a percent encoding: a genuine 0.8% yield would be rescaled to 80 and would fail `yieldR <= 10`, suppressing the stressor for precisely the institutions it is meant to catch. **UNCLEAR**: the source encoding cannot be determined from the code. Unresolved. 3. **`accept >= 90` as a distress signal is unreachable for community colleges by design and near-unreachable elsewhere.** Open-admission institutions would satisfy it, but they are the ones the CC gate excludes; among selective institutions a 90%+ acceptance rate is rare. In practice `paired` is carried by the enrollment and applications legs. 4. **`Number.isFinite(ntrNorm)` is checked twice after a null test.** Lines 2407–2408 re-test finiteness on a value that line 2393 already established is not null. `getNtrNorm()` can return a non-finite number only if `net_tuition_per_fte_normalized` were `Infinity`, which its own `Number.isFinite` guard (line 1164) prevents, or from `current / peer` with `peer !== 0` — also guarded. The checks are therefore redundant but not wrong. 5. **Return-key inconsistency.** The CC gate returns `reason:` (line 2384) where every other return in the method uses `reasoning:`. The consumer at line 3201 reads `pattern.reasoning || pattern.reason`, so nothing is lost in the report — but any downstream code reading `.reasoning` alone would see `undefined` for a community college. P5 has the same inconsistency. 6. P4 is the highest-firing pattern in the layer (34.6%) and the second-lowest-discriminating by the table at line 219 (+11.1). The two facts together are worth a calibration review; the code records no such review.

P5 — Unrestricted Net Assets Risk

Points 6 · fires for 278 of 1,401 (19.8%)

Update (v49.13, 19 October 2026). For privates weakAom is the v2 operating margin's three-year average at or below −1.5% (`PRIVATE_MARGIN_GATES.p5`), and the reason says "weak operating margin". The −3.0 cutoff in the "all other sectors" column below now applies only to for-profits; community colleges are excluded from P5. Publics are unchanged. Current method: from line 2627.

- **Method**: `calculateP5_UNARisk()` (lines 2453–2498). **No severity argument.**
- **Points**: **6** base, escalating to **8**; **4** on the data-gap fallback. Lines 2489–2490:

```
  let pts = 6;
  if (unaPct <= 10) pts += 2;   // bottom decile, was `unaNorm <= 0.40`
```

Line 2478: `return { triggered: true, points: 4, reasoning: 'Sustained UNA weakness with currently negative UNA' };`. Calibration table, line 217: `//  P5_UNA_Risk  6-8 +21.3` — the **best-discriminating** pattern in the layer.

- **What it tests**: an institution in the bottom quartile of its peer group for unrestricted net assets per FTE, where that thin cushion is corroborated by either a persistently weak operating margin or low primary reserves. The percentile framing is deliberate: the raw ratio inverted for publics (see the 2026-08-28 record below).
- **Inputs**:

Local	Route	Analyzer	Field	Scale
unaPct	`this.getUnaPercentile()` — helper (lines 1208–1222), uses `pickAnalysis` + `getFirst`, **not** ledgered	unrestrictednetassets	una_per_fte_percentile	0 = weakest peer, 100 = strongest
aom3	`this.num(this.getMetricValue('structuralmargin','three_year_average'))` if sector `'1'`, else `this.num(this.getMetricValue('adjustedoperatingmargin','three_year_average'))`	structuralmargin / adjustedoperatingmargin	three_year_average = rollingAverages.recent3Year	percent
prPct	`this.pct(this.getMetricValue('primaryreserve','primary_reserve_ratio'))`	primaryreserve	keyMappings['primary_reserve_ratio'] → 'currentValue'	**percent**, already ×100 at source (unified_primary_reserve_analyzer.js line 941, `return num * 100;`)
unaRP (fallback)	`this.num(this.getMetricValue('unrestrictednetassets','riskPoints') ?? this.getMetricValue('una','riskPoints'))`	unrestrictednetassets	riskPoints	points
unaNow (fallback)	`this.num(this.getMetricValue('unrestrictednetassets','total') ?? this.getMetricValue('unrestrictednetassets','currentNetAssets'))`	unrestrictednetassets	'total' → keyMappings → 'currentValue' = current2024	dollars

`getUnaPercentile()` matches the analyzer through `pickAnalysis()` (lines 8–36) against the names `'UniversalUnrestrictedNetAssetsAnalyzer'`, `'UnifiedUnrestrictedNetAssetsAnalyzer'`, `'UnrestrictedNetAssetsAnalyzer'`. Normalized, `'unrestrictednetassets'` is a substring of each, so the `needle.includes(ak)` branch at line 29 matches.

- **Gate logic**, lines 2455–2496:

```
    if (this.sector === '4' || this.sector === 4 ||
        (this.sectorName && this.sectorName.toLowerCase().includes('community'))) {
        return { triggered: false, points: 0, reason: 'P5 UNA thresholds not applicable to community colleges'
};
    }
  ...
   if (unaPct === null) {
    const unaRP = this.num(this.getMetricValue('unrestrictednetassets','riskPoints') ??
this.getMetricValue('una','riskPoints'));
    const unaNow = this.num(this.getMetricValue('unrestrictednetassets','total') ??
this.getMetricValue('unrestrictednetassets','currentNetAssets'));
    if (Number.isFinite(unaRP) && unaRP >= 3 && unaNow !== null && unaNow < 0) {
      return { triggered: true, points: 4, reasoning: 'Sustained UNA weakness with currently negative UNA' };
    }
```

```
    return { triggered: false, points: 0, reasoning: (unaNow !== null && unaNow >= 0) ? 'Historical UNA
weakness only — currently positive' : 'No UNA/FTE norm' };
  }

  // 2026-08-28: bottom quartile, replacing `unaNorm <= 0.60` on the inverting ratio.
  const lowUna  = unaPct <= 25;
  const weakAom = (aom3 !== null) && (this.sector === '1' ? aom3 <= -2.0 : aom3 <= -3.0);
  const lowPR   = (prPct !== null) && (this.sector === '1' ? prPct < 20 : prPct < 25);

  if (lowUna && (weakAom || lowPR)) {
    let pts = 6;
    if (unaPct <= 10) pts += 2;   // bottom decile, was `unaNorm <= 0.40`
```

Test	Public (`'1'`)	Private / other	Direction	Role
CC gate	sector `'4'`	sector `'4'`	equality	front gate
`lowUna`	`unaPct <= 25`	`unaPct <= 25`	**ceiling**	required
`weakAom`	`aom3 <= -2.0` on **structural margin**	`aom3 <= -3.0` on **adjusted operating margin**	**ceiling**	one of two, OR
`lowPR`	`prPct < 20`	`prPct < 25`	**ceiling, strict <**	one of two, OR
escalation	`unaPct <= 10`	`unaPct <= 10`	**ceiling**	+2
fallback	`unaRP >= 3` (**floor**) AND `unaNow < 0` (**ceiling, strict**)	same	—	4 points

Fires when: not sector `'4'` AND `unaPct <= 25` AND (`weakAom` OR `lowPR`), or via the fallback.

Note `lowPR` is the only **strict** inequality among the P1–P5 primary tests; every other threshold is inclusive.

- **Sector restriction**: **none in the 2026 baseline**, but the thresholds and the margin **source** both branch on sector.
  - The CC gate (lines 2455–2458) is identical to P4's and has the same two dead sub-conditions (numeric `===` 4 unreachable; the `sectorName` test redundant). No sector-`'4'` institutions exist in the 1,401-institution baseline, so it never fires there.
  - Every other sector test is `this.sector === '1' ? … : …`. Sector `'3'` (For-Profit) therefore takes the **private** branch of all three ternaries by default. Nothing in the code marks that as a decision.
  - Observed: 19.8% overall — 127/499 publics (25.5%) and 151/902 privates (16.7%). The gap is consistent with publics being scored on structural margin at a −2.0 threshold against privates on adjusted operating margin at −3.0.
- **Data-gap behavior**: mixed. **Fails closed on the corroborators, fails open on a narrow fallback.**
  - `unaPct` null → the fallback path. It requires **both** `unaRP >= 3` **and** `unaNow < 0` — a currently negative unrestricted balance, not merely a weak one. That is a much tighter fallback than P4's, and it awards 4 rather than 6.
  - `unaPct` null with `unaNow >= 0` → the distinct reason `'Historical UNA weakness only — currently positive'`, which is diagnostic rather than a mere miss.
  - `aom3` null and `prPct` null → both legs are explicitly null-guarded, so `(weakAom || lowPR)` is false and the pattern **cannot fire even for an institution in the bottom percentile of UNA**. This is a genuine fail-closed: the corroborating requirement means a data gap in either margin or reserves is survivable, but a gap in both suppresses the pattern entirely.
  - Like `getNtrNorm()`, `getUnaPercentile()` bypasses `getMetricValue()`, so a null `unaPct` is not recorded in the gate ledger.
- **2026-08 changes**: two inline dated comments in the method, and the helper's replacement record.

In the method body, lines 2483 and 2490:

> *`// 2026-08-28: bottom quartile, replacing unaNorm <= 0.60 on the inverting ratio.`*
>
> *`if (unaPct <= 10) pts += 2;   // bottom decile, was unaNorm <= 0.40`*

On the public/private margin split, lines 2462–2464:

> *`// Publics: use structural margin (GASB pension/OPEB noise makes raw margin // unusable as a distress gate; peer avg on raw basis is ~-40%). // Privates: unchanged, FASB-basis margin.`*

And the helper record that both threshold changes derive from, lines 1187–1207:

```
// 2026-08-28 (v49.1): getUnaNorm is REPLACED by getUnaPercentile. // // It returned a ratio, unaPerFte / peerMedianPerFte, on which 1.0 meant "at the // peer median" and consumers tested < 0.60. That inverts whenever the peer // median is negative — which is the public case in Carnegie 18, 20, 21 and 22, // where GASB 68/75 pension and OPEB recognition drove roughly half of each group // below zero from FY2015. For those institutions the weakest cushions produced // the HIGHEST index values and the gates below read the signal backwards, for // about 267 publics. // // The old ratio fallback (current / peer) is deliberately gone as well: it had // exactly the same defect, so leaving it in place would have silently restored // the bug whenever the preferred field was missing. // // The analyzer now publishes una_per_fte_percentile: 0 = weakest in the peer // group, 100 = strongest, well-defined whatever the sign of any member. There is // no ratio and no denominator, so it cannot invert. // // THRESHOLD TRANSLATION. The old < 0.60 was intended to mean "meaningfully // below the peer median". On a percentile that is < 25 — the bottom quartile. // The old <= 0.40, the P5 escalation, becomes <= 10 — the bottom decile.
```

Note the translation comment says `< 25` while the implementation at line 2484 is `<= 25`. The one-institution difference at exactly the 25th percentile is immaterial in practice but the comment and the code do not match.

- **Notes**: 1. **`pct()` can invert `lowPR` for the weakest institutions.** The primary reserve analyzer already multiplies by 100 (line 941), so `currentValue` is a percent. `pct()` then rescales anything `<= 1`: a genuine primary reserve ratio of 0.8% arrives as `0.8`, becomes `80`, and **fails** `prPct < 20`. An institution with almost no reserves is read as having 80% and its corroborator is silently switched off. The at-risk band is `0 < prPct <= 1`; `prPct === 0` maps to `0` and is handled correctly, and negative values in `(-1, 0]` are also rescaled but stay below the threshold so the gate outcome does not flip. This is the single most consequential defect found in P1–P5, because it targets exactly the institutions the pattern exists to catch. Its population impact is **UNCLEAR** — the count of institutions with a primary reserve ratio strictly between 0% and 1% cannot be determined from source. 2. **The private AOM threshold disagrees with every other gate in the file.** P5 uses `aom3 <= -3.0` for privates (line 2485). `checkFinancialDistress()` at line 1256 uses `aom3 <= -5.0`; `applySeverityOverride()` at line 2194 uses `aom3 <= -5`. The public threshold (−2.0) is the same in all three. No comment explains the −3.0. Treat as deliberate-but-unrecorded, not a typo, since it appears once and is internally consistent within the method. 3. **The public −2.0 threshold is applied to a different quantity in different gates.** P5 reads it from `structuralmargin` (line 2466); `checkFinancialDistress()` (line 1236) and `applySeverityOverride()` (line 2178) read `adjustedoperatingmargin` for **all** sectors and apply −2.0 to publics. The threshold value is shared; the measure it is applied to is not. The comment at lines 2462–2463 explains why P5 switched but does not note that the other two gates did not. 4. **Return-key inconsistency**, as in P4: the CC gate returns `reason:` (line 2457) where every other return uses `reasoning:`. 5. **`weakAom` and `lowPR` are an OR, so the reason string can be misleading.** Line 2494 reads `` `Thin UNA/FTE (${unaPct}th percentile of peers) with ${weakAom ? 'weak AOM' : 'low reserves'}` ``. When both are true it reports only "weak AOM"; when the public branch fired it reports "weak AOM" for what is actually a structural margin. The label is wrong for publics in every case. 6. The fallback's `getMetricValue('una','riskPoints')` second link (line 2475) is reachable only if `'unrestrictednetassets'` failed to resolve — but both keys share the same alias list (lines 485–492), so if the first misses the second misses too. Dead as written.

P6 — Research Coverage Erosion

Points — · fires for no institution in the baseline

- **Method**: **NOT IMPLEMENTED** and **NOT REGISTERED**. No `calculateP6_*` function exists anywhere in the repository. A repo-wide search for `calculateP6`, `ResearchCoverageErosion` and `P6_` returns exactly three hits, none of them a definition or a call:
  - `risk_adjustment_analyzer.js:817` — a comment inside `_annotateGateStatus()` that still cites the removed method as its worked example
  - `risk_adjustment_analyzer.js:2500–2523` — the retirement/removal comment block, which occupies the space where the method used to be
  - `modular_tool_v48 no data cap.html:3904` — a display-name entry, `'P6_ResearchCoverageErosion': 'P6 Research Coverage Erosion'`, in a label map keyed on strings that `patternAddons` never contains
- **Points**: none. No value can ever be awarded.
- **What it tests**: nothing, in the current build. Per the retained comment it was intended to detect erosion of research-cost coverage — the level of coverage now (`covNow`), its trend (`covTrend`) and the research-expense trend (`expenseTrend`) — with a `signalCount` and a severity-2 escalation. Twenty measured fields per the removal note.
- **Inputs**: none are read, because no code reads anything. Historically the three coverage inputs read fields no analyzer publishes. `research_grants_analyzer.js` computes `ggcCAGR`, `rexpCAGR` and `ratioCAGR` (lines 376–380) and publishes only `ggcCAGR` and `ratioCAGR` onto `analysisResults` (lines 228–229); no coverage level and no 5-year point change is published anywhere.
- **Gate logic**: not present in source. Two thresholds survive only as names in the retirement comment, `COV_TREND_TRIP` and `SPREAD_TRIP`, with no values and no code. **UNCLEAR**: the numeric values of those two constants are unrecoverable from this repository — the implementation was deleted, not commented out.
- **Why it does not fire**: it does not exist. There is no method, so there is nothing to register; there is no entry in `calculatePatternAddons()` (lines 2851–2860) or in `__gateFnByKey` (lines 435–442), so nothing would call it even if it existed. Firing zero times across 1,401 institutions is the only possible outcome.
- **Data-gap behavior**: not applicable. Because `P6_*` is never a key in `this.patternAddons`, `_annotateGateStatus()` never visits it and the session tally `window.__RA_GATE_TALLY__` has no P6 row at all — P6 is not reported as `not_reached`, it is simply absent from every diagnostic surface.
- **2026-08 changes**: the whole block, verbatim (lines 2500–2523):

> *// P6 — Research Coverage Erosion (research-data only; no Carnegie dependency) // RETIRED 2026-08-09. Not registered in __gateFnByKey and not evaluated in the // pattern registry; retained unreferenced for the record. Do not re-register // without addressing all three reasons it was retired: // 1. The coverage fields it reads were never published. research_grants_analyzer // computes rexpCAGR, ggcCAGR and ratioCAGR but drops trends when building // analysisResults, so covNow, covTrend and expenseTrend were always null and // signalCount was always 0. Severity 2 never fired at any institution. // 2. Units mismatch. COV_TREND_TRIP and SPREAD_TRIP are 5-year point changes; // the only available research trends are annualized CAGRs. Publishing CAGR // under the *_5y key names would repeat the net revenue ratio scaling defect. // 3. covNow has no source at all. Only the ratio trend is computed, not the level, // so two of the three signals cannot be revived by publishing alone. // Validation would also need a research-intensive cohort: the research analyzer is // not_applicable at all 66 Carnegie-22 publics, so the standing control cannot test it. // 2026-08-28 (v49.1): calculateP6_ResearchCoverageErosion REMOVED. // A complete implementation reading twenty measured fields that was absent from // calculatePatternAddons() and from __gateFnByKey, so it had no call site anywhere // and fired 0 times. All three of its coverage inputs also read fields no analyzer // publishes, so wiring it as written would not have made it fire either. Both the // architecture documentation and the analyzer catalog described it as live. // Rebuild it against real published fields if research coverage erosion is wanted.*

- **Notes**:
  - The published architecture documentation and the analyzer catalog describe P6 as live. They are wrong, and were wrong before the 2026-08-28 removal as well — P6 was already unreachable from 2026-08-09.
  - The display-name map at `modular_tool_v48 no data cap.html:3904` still advertises P6. It is inert (the key never appears) but it is a residue that will make P6 look live to anyone reading the label map.
  - The comment at `risk_adjustment_analyzer.js:817` explains a live branch of `_annotateGateStatus()` by reference to a method that no longer exists. The branch itself is still reachable by other gates; only its worked example is stale.
  - P6's numbering is now a permanent gap: patterns run P1–P5, P7–P9.

P7 — Brand Positioning

Points 6 → 8 · fires for 76 of 1,401 (5.4%)

- **Method**: `calculateP7_BrandPositioning()`, lines **2526–2555**. Registered at line 2857 as `'P7_BrandPositioning'`. Fires 83 times in the baseline.
- **Points**: **6 base, escalating to 8.** Line 2546 `let pts = 6;`; line 2547 `if ((accept !== null && accept >= 90) || (yieldR !== null && yieldR <= 10)) pts += 2;`. Two discrete values only: 6 or 8.
- **What it tests**: from the code comment at line 2540, `// Brand pressure: big app declines paired with high admit or low yield`. A large multi-year fall in applications that coincides with either an unselective admit rate or a yield so low that admitted students are choosing elsewhere.
- **Inputs**:

Variable	Call	Analyzer	Route to a published field
`apps14`	`this.num(this.getMetricValue('acceptanceyield','applications_change_since_2014'))`	`acceptanceyield` (UniversalAcceptanceYieldAnalyzer)	keyMappings line 925: `'applications_change_since_2014': 'applicationChangePercent'` → `analysisResults.applicationChangePercent`, published at `unified_acceptance_yield_analyzer.js:93–97` as `(totalChange / baseline) * 100`
`accept`	`this.pct(this.getMetricValue('acceptanceyield','acceptance_rate'))`	same	keyMappings line 926: `'acceptance_rate': 'currentAcceptanceRate'` → published at :99 as `brandAnalysis.acceptanceRate?.current || null`
`yieldR`	`this.pct(this.getMetricValue('acceptanceyield','yield_rate'))`	same	keyMappings line 927: `'yield_rate': 'currentYieldRate'` → published at :101 as `brandAnalysis.yieldRate?.current || null`

Also read directly off the analyzer instance, not through the resolver: `this.sector` and `this.sectorName` (set in the constructor, lines 390–398).

All three metric routes are explicit `keyMappings` entries, so the 2026-08-30 fail-closed change does not affect P7. Each also has a declared `metricAliases` entry (lines 586–595) as a second route.

- **Gate logic**:

Front gate (lines 2528–2531) — returns before any lookup:

```
if (this.sector === '4' || this.sector === 4 ||
    (this.sectorName && this.sectorName.toLowerCase().includes('community'))) {
    return { triggered: false, points: 0, reason: 'P7 not applicable to open-access community colleges' };
}
```

All-null gate (lines 2536–2538): if `apps14`, `accept` and `yieldR` are all `null`, return `{triggered:false, points:0, reasoning:'No brand pipeline metrics'}`.

Signal	Line	Condition	Direction
`strongDecline`	2541	`apps14 !== null && apps14 <= -20`	**ceiling** (≤ −20 %)
`admitHigh`	2542	`accept !== null && accept >= 85`	**floor** (≥ 85 %)
`yieldLow`	2543	`yieldR !== null && yieldR <= 12`	**ceiling** (≤ 12 %)

Trigger (line 2545): `if (strongDecline && (admitHigh || yieldLow))`. The application decline is mandatory; either corroborating signal suffices.

Escalation	Line	Condition	Direction	Effect
+2 pts	2547	`accept >= 90`	**floor**	`pts` 6 → 8
+2 pts	2547	`yieldR <= 10`	**ceiling**	`pts` 6 → 8

The two escalators are OR-ed; satisfying both still adds only 2.

- **Why it does not fire**: not applicable — P7 fires. 83 institutions in the baseline.
- **Data-gap behavior**: **fails closed, per signal.** Every condition is guarded by an explicit `!== null` test, so a null input contributes no signal and can only make the gate harder to open. With `apps14` null the pattern is unreachable regardless of the other two. All three null short-circuits at line 2536 with an explicit reason. No null is ever coerced to 0 and compared against a threshold.
- **2026-08 changes**: **none.** `calculateP7_BrandPositioning()` carries no dated comment. Its inputs are affected only indirectly, by the 2026-08-30 resolver change, and all three have explicit routes so their behavior is unchanged.
- **Notes**:
  - The returned object uses **two different key names for the same field**: the community-college front gate returns `reason` (line 2530) while every other exit returns `reasoning` (2537, 2551, 2554). Any consumer reading only one of the two sees `undefined` for the other exits. `calculatePatternAddons()` does not read either, so scoring is unaffected; report surfaces may be.
  - `pct()` is a units shim, not a conversion. A yield rate legitimately published as `1` (meaning 1 %) satisfies `v <= 1` and becomes `100`, which then fails both `yieldR <= 12` and `yieldR <= 10`. The one value most deserving of the escalation is the one value the helper inverts. **UNCLEAR**: the source workbook columns (`<year> accrate`, `<year> yield`, read at `unified_acceptance_yield_analyzer.js:641–647` via `parseNumericValue`) are passed through unscaled, so the unit is whatever the source file uses and is not determinable from the code. The 83 baseline firings imply a 0–100 scale in practice.
  - `brandAnalysis.acceptanceRate?.current || null` (line 99 of the acceptance analyzer) converts a genuine `0` to `null` via the falsy `||`. A 0 % acceptance rate is not realistic, but a 0 % *yield* at a tiny program is; it would be published as `null` and P7 would lose the `yieldLow` signal rather than score it as the strongest possible case.
  - The gate is sector-guarded against sector 4 only. Sector 3 (`String(sector)` fallback, constructor lines 390–395) is not excluded and would be evaluated on brand metrics.

P8 — Market Position

Points 5 → 7 · fires for 434 of 1,401 (31.0%)

- **Method**: `calculateP8_MarketPosition()`, lines **2558–2588**. Registered at line 2858 as `'P8_MarketPosition'`. Fires 456 times in the baseline — the second most frequent pattern.
- **Points**: **5 or 7; escalates once, on signal count.** Line 2581: `const points = (signals.length >= 3) ? 7 : 5;` — 5 for exactly two signals, 7 for three or four. Line 2580 states the intent:

> *// Keep weights modest; P8 is a pattern add-on, not a wholesale override*

- **What it tests**: per the comment at line 2564, `// Market-position signals derived from acceptance/yield/app trends and (if available) share change`. A count-based deterioration test: any two of four independent adverse market signals.
- **Inputs**:

Variable	Call	Analyzer	Route to a published field
apps14	`this.num(this.getMetricValue('acceptanceyield','applications_change_since_2014'))`	acceptanceyield	keyMappings:925 → applicationChangePercent
accept	`this.pct(this.getMetricValue('acceptanceyield','acceptance_rate'))`	acceptanceyield	keyMappings:926 → currentAcceptanceRate
yieldR	`this.pct(this.getMetricValue('acceptanceyield','yield_rate'))`	acceptanceyield	keyMappings:927 → currentYieldRate
shareCh	`this.num(this.getMetricValue('enrollment','market_share_change'))`	enrollment (UniversalEnrollmentAnalyzer)	**no keyMappings entry**; resolves at step 4 by exact published name — unified_enrollment_analyzer.js:125 publishes market_share_change onto analysisResults. Declared metricAliases entry at lines 617–619 ('market_share_change','share_change','marketShareChange') is a redundant second route.

Plus `this.sector` / `this.sectorName` read directly.

All four fields therefore have an explicit route and survive the 2026-08-30 fail-closed change. `shareCh` is the only one that depends on the exact-name route rather than the mapping table; had the enrollment analyzer not published under that precise name, the fail-closed resolver would now return `null` where it previously substituted a generic field.

- **Gate logic**:

Front gate (lines 2560–2563), identical in shape to P7's:

```
if (this.sector === '4' || this.sector === 4 ||
    (this.sectorName && this.sectorName.toLowerCase().includes('community'))) {
    return { triggered: false, points: 0, reason: 'P8 not applicable to open-access community colleges' };
}
```

Signals, each pushed to `signals[]` (lines 2571–2574):

Signal	Line	Condition	Direction
High acceptance	2571	`accept !== null && accept >= 85`	**floor** (≥ 85 %)
Low yield	2572	`yieldR !== null && yieldR <= 12`	**ceiling** (≤ 12 %)
Applications decline	2573	`apps14 !== null && apps14 <= -10`	**ceiling** (≤ −10 %)
Market share loss	2574	`shareCh !== null && shareCh <= -5`	**ceiling** (≤ −5 %)

Trigger (lines 2576–2578): `if (signals.length < 2) return {triggered:false, ...}` — i.e. the gate opens on a **floor of 2 signals**.

Signal count	Line	Points
0–1	2576	0, not triggered
2	2581	5

Signal count	Line	Points
≥ 3 (**floor**)	2581	7

- **Why it does not fire**: not applicable — P8 fires, 456 times.
- **Data-gap behavior**: **fails closed.** Each of the four signals is `!== null` guarded, so a missing input simply cannot contribute a signal, and fewer available inputs makes the two-signal floor strictly harder to clear. There is no separate all-null exit; all-null falls through to `signals.length < 2` and returns `'Market position stable (insufficient adverse signals)'` — see Notes, the reason string misdescribes that case.
- **2026-08 changes**: **none inside the method.** No dated comment appears in lines 2558–2588. The `market_share_change` alias at lines 617–619 is marked only `// NEW`, undated. `calculateMarketShareChange()` in the enrollment analyzer carries its own fail-closed guard (lines 555–560): `if (n < 5) return null;` and a requirement that all four totals be finite and `> 0`, commented `// Fail closed: a share change built on a thin or lopsided group is noise.`
- **Notes**:
  - Same `reason` / `reasoning` key inconsistency as P7: the community-college gate returns `reason` (2562), the other two exits return `reasoning` (2577, 2586).
  - `'Market position stable (insufficient adverse signals)'` is returned both when the institution genuinely has fewer than two adverse signals **and** when the inputs could not be read at all. The two are indistinguishable from the reason string. The gate ledger does separate them — `_annotateGateStatus()` will mark the gate `unavailable` or `not_applicable` — but the published reason will still read "stable".
  - P8's thresholds are a strict subset relaxation of P7's on two of three shared signals (`apps14 <= -10` vs `-20`) and identical on the other two (`accept >= 85`, `yieldR <= 12`). An institution that fires P7 will normally also fire P8, and take both add-ons: **6–8 points from P7 plus 5–7 from P8, 11–15 points from one underlying brand story.** There is no double-count guard between them, unlike the explicit A10/P9 guard. In the baseline this is not a tendency but an identity: **all 83 P7 firings are also P8 firings** (83 of the 456). P7 has no institution of its own.
  - `shareCh` is the only signal not derived from the acceptance/yield analyzer, so at an institution where that analyzer is absent, P8 can have at most one signal available and is structurally unable to trigger.
  - Sector 3 is not excluded by the front gate.

P9 — Endowment Drain Risk

Points 4 · never fired before v49.11; fires since (rate not re-baselined)

- **Method**: `calculateP9_EndowmentDrainRisk()`, lines **2621–2721** (design comment 2590–2620). Registered at line 2859 as `'P9_EndowmentDrainRisk'`. **Fires 0 times** in the validated 2026 baseline. (Line numbers in this entry predate v49.10. Current: the method from line 2797; the three gates in __evalP9Gates(), from about line 2865.)
- **Points**: **+4, flat. Does not escalate.** Line 2712, `points: 4`, the single trigger return. Confirmed by the design comment at line 2617, `// Points: +4`. There is no severity tier and no second point value anywhere in the method.
- **What it tests**: quoted from the design block, lines 2593–2597:

> *// What it is: Institutions running structural operating deficits funded by // sustained endowment draws against an endowment base that is thin relative // to Carnegie peers. Distinct from A10: does NOT require high asset restriction. // Captures the broader population of institutions depleting modest endowments // to cover operating shortfalls they cannot otherwise fund.*

Current state (v49.11 and v49.16)

The repair (v49.11, 15 October 2026). Gate 3 could not read its inputs, as traced under "Why it did not fire" below. `unified_endowment_analyzer.js` now publishes `perFTECurrent` and `perFTEPeerMedian` on `analysisResults`. The median is `peerAnalysis.statistics.perFTECurrentValue.median`, the same figure the report prints as "peer median", so the gate and the report read one number. The gate reads these first and falls back to `rawData`. The two draw-rate reason strings now multiply by 100. Thresholds are unchanged. **Validated 18 October 2026:** all 20 predicted privates fire at +4.

The deferral to A10. It was always in the code; it was invisible only because P9 never reached it. It is one-way: P9 checks A10 before its own gates, and A10 never reads P9. `detectArchetypeFlags()` runs before `calculatePatternAddons()`, so A10 is always settled first. Since v49.16 (README item 52), when A10 fires P9 still scores 0, but it evaluates its gates and prints `deferred to A10 (endowment-masked deficit archetype); on its own P9 would have fired (+4)` or `... would not have fired: <reason>`. On the anchor run, Brandeis and Lawrence fire A10 and print `would not have fired: structural margin not critical (high)`.

What P9 still misses. Colleges with a Critical structural margin and a heavy draw whose endowment per FTE is at or above 0.48× the peer median, and which do not meet A10's restriction test, fire neither gate. U38 marks them (next entry).

- **Inputs**: P9 makes **no `getMetricValue()` calls**. It reads `this.completedAnalyses` directly, with its own local `norm`, `getRisk` and `getLevel` helpers defined inline at lines 2637–2647.

Value	Read at	Expression	Producing analyzer	Route
`this.sector`	2625	`this.sector !== '2' && this.sector !== 2`	—	constructor lines 390–395
`a10`	2630	`this.archetypeFlags?.A10`	Layer 2 itself	populated at Step 2 of `analyze()`, before Step 3
`structLevel`	2649	`getLevel(['structuralmargin','structural_margin','structural_operating_margin']) → a?.analysisResults?.riskLevel`	`UniversalStructuralMarginAnalyzer` → key `structuralmargin`	`analysisResults.riskLevel`, which survives the wrapper
`draw3yr`	2664	`drawAnalysis?.analysisResults?.three_year_avg ?? null`	`UniversalEndowmentDrawSustainabilityAnalyzer` → key `endowmentdrawsustainability`	`unified_endowment_draw_sustainability_analyzer.js:155`, `three_year_avg: drawData.threeYearAvg`, inside `analysisResults` — survives the wrapper
`eftVal`	2687	`endowAnalysis?.rawData?.perFTEEndowment?.currentValue ?? null`	`UniversalEndowmentAnalyzer` → key `endowment`	`unified_endowment_analyzer.js:135`, inside **`rawData` — does not survive the wrapper**
`eftMedian`	2688	`endowAnalysis?.rawData?.peerContext?.peerMedian ?? null`	same	`unified_endowment_analyzer.js:139`, inside **`rawData` — does not survive the wrapper**

Analyzer selection for the last two (lines 2680–2685):

```
  const endowAnalysis = (this.completedAnalyses || []).find(x =>
    norm(x.analyzerKey).includes('endowment') &&
```

```
    !norm(x.analyzerKey).includes('draw') &&
    !norm(x.analyzerKey).includes('depend') &&
    !norm(x.analyzerKey).includes('restrict')
  );
```

This selection is **correct**: `endowmentdrawsustainability` is excluded on `draw`, `endowmentdependency` on `depend`, and only `endowment` survives. The record is found. It is the field read off it that fails.

- **Gate logic**:

#	Line	Condition	Direction	Exit reason when it fails
Sector	2625	`this.sector === '2'` required	equality	`'not applicable (public sector)'`
Guard	2631	`a10?.triggered` → yield	—	`'deferred to A10 (endowment-masked deficit archetype)'`
1	2652	`structLevel === 'critical'`	equality, case-normalized	`` `structural margin not critical (${structLevel || 'unknown'})` ``
2a	2666	`draw3yr !== null`	presence	`'no draw sustainability data available'`
2b	2671	`draw3yr <= 0.50`	**ceiling** (data-artifact cap)	`` `draw rate ... exceeds 50% cap — likely data artifact` ``
2c	2674	`draw3yr > 0.055`	**floor**, strict	`` `3yr draw rate ... does not exceed 5.5% threshold` ``
3a	2690	`eftVal !== null && eftMedian !== null && eftMedian > 0`	presence + **floor**	`'endowment/FTE or peer median data not available'`
3b	2701	`eftRatio < 0.480` where `eftRatio = eftVal / eftMedian`	**ceiling**, strict	`` `EFT/median ratio ... not in bottom quartile (threshold: <0.48)` ``

Threshold constants, quoted from source:

- line 2671 `if (draw3yr > 0.50) {`
- line 2674 `if (draw3yr <= 0.055) {`
- line 2699 `const BOTTOM_QUARTILE_RATIO = 0.480;`
- line 2701 `if (eftRatio >= BOTTOM_QUARTILE_RATIO) {`

Units: `draw3yr` is a **fraction**, not a percentage. `unified_endowment_draw_sustainability_analyzer.js:261` computes `result.threeYearAvg = recent.reduce((s,r) => s + r.drawRate, 0) / recent.length` and line 377 renders it as `(drawData.threeYearAvg * 100).toFixed(2)`. So `0.055` = 5.5 % and `0.50` = 50 %. The gate-2 thresholds are internally consistent with the producer.

Rationale for gates 2 and 3, quoted from the design block (lines 2599–2607):

> *// 2. 3-year average endowment draw rate > 5.5% (NACUBO sustainable ceiling // is 4.5–5.0%; 5.5% adds a 0.5pp buffer to exclude borderline cases and // focus on institutions with a multi-year sustained pattern) // 3. Endowment/FTE in bottom quartile vs Carnegie peers (EFT value < 0.48x // peer median — empirically derived from sector distribution; excludes // institutions where the draw is a policy choice from a position of strength)*

and the double-count guard (lines 2609–2612):

> *// Double-count guard: if A10 has already triggered, P9 returns 0 points. // A10 is the more specific, higher-severity archetype (+6 pts, requires // restriction gate); P9 is the broader pattern (+4 pts, no restriction gate). // When both conditions are present, A10 takes precedence.*

- **Why it did not fire before v49.11 (history):**

Gate 3a. `eftVal` and `eftMedian` are always `null`, because `rawData` never reaches the Layer 2 record. Every P9 evaluation that survives the sector filter, the A10 guard and gates 1 and 2 exits at line 2691 with `'endowment/FTE or peer median data not available'`.

The trace, in four steps:

1. `unified_endowment_analyzer.js` lines 134–142 genuinely publishes both fields, nested under `rawData`:

```
    rawData: {
        perFTEEndowment: perFTETrends,
        totalEndowment: totalTrends,
        peerContext: {
            peerCount: ...,
```

```
            peerMedian: peerAnalysis.success ? peerAnalysis.statistics?.perFTECurrentValue?.median : null,
            ...
        }
    },
```

2. That object is the return value of `UnifiedEndowmentAnalyzer.analyze()`, which `UniversalEndowmentAnalyzer.performSectorAnalysis()` returns unmodified (`modular_tool_v48 no data cap.html:2078–2082`, `return result;`).

3. The orchestrator does **not** call `performSectorAnalysis()`. It calls `analyzer.analyze()` (`analysis_orchestrator.js:218`), which is the base class method `SectorAwareAnalyzer.analyze()`. That method reconstructs the result from three keys only (`modular_tool_v48 no data cap.html:1045–1049`):

```
    return {
        success: true,
        analysisResults: analysisResult.analysisResults,
        detailedReport: analysisResult.detailedReport || {}
    };
```

`rawData` is dropped here. Every analyzer in the tool is a `Universal*` subclass of this class (`analysis_orchestrator.js:65–168`), so there is no analyzer for which `rawData` survives. `rawData` appears nowhere else in the HTML runtime; a repo search finds it only as an unrelated local variable name at line 1139.

4. The 2026-08-28 repair added `rawData` to the **record push** (`analysis_orchestrator.js:385`, `rawData: data.result.rawData ?? null`). But `data.result` is the three-key object from step 3, so `data.result.rawData` is `undefined`, `?? null` yields `null`, and the record carries `rawData: null`. At line 2687, `endowAnalysis?.rawData?.perFTEEndowment` is then `undefined` (optional chaining on `null`), `?? null` makes it `null`, and gate 3a rejects.

The 2026-08-28 fix was applied one layer too low. It repaired the record builder while the field was being stripped one layer above it, at the wrapper boundary. Its own comment (`analysis_orchestrator.js:376–384`) states the diagnosis correctly and the remedy incompletely:

> *// 2026-08-28 (v49.1): rawData is now carried onto the record. // Its absence was the single most frequent defect in this codebase: a // layer-2 gate reading `someAnalysis.rawData.x` could never succeed, because // this push built the record from analysisResults alone. It is what disabled // calculateP9_EndowmentDrainRisk entirely — its third gate reads // rawData.perFTEEndowment.currentValue and rawData.peerContext.peerMedian, // both of which the endowment analyzer genuinely publishes, just not here. // 158 privates clear P9's first two gates and were then rejected by a test // that could not read its own input; P9 fired 0 times in 1,552 institutions.*

The phrase *"just not here"* is the error: the fields are not merely absent from the push, they are absent from `data.result` itself.

Gates 1 and 2 are sound and do pass. Three independent confirmations:

- Gate 1's exact-match lookup on `structuralmargin` is byte-identical in shape to `checkA10_EndowmentMaskedDeficit()` lines 1863–1868, and A10 fires 87 times in the baseline. The lookup resolves.
- `three_year_avg` sits inside `analysisResults` (`unified_endowment_draw_sustainability_analyzer.js:155`), which is one of the three keys that survives the wrapper. Gate 2 can read its input.
- The orchestrator comment above records that **158 privates cleared gates 1 and 2** in the pre-fix run.

Nothing else blocks P9. The sector filter admits sector 2, and the baseline contains many privates (998 of the 1,551 baseline records). The A10 guard removes 87 of those — all 87 A10 firings are private — leaving 911 privates that reach gate 1. Gate 3b is never reached. Fixing gate 3a — by carrying `rawData` through `SectorAwareAnalyzer.analyze()`, or by having the endowment analyzer publish `perFTEEndowment.currentValue` and `peerContext.peerMedian` onto `analysisResults` — is the only change needed to make P9 evaluable. It would not by itself make P9 *fire*: gate 3b is untested by any run to date.

- **Data-gap behavior**: **fails closed at every gate**, which is why the defect is silent rather than wrong. Each of gates 1, 2a and 3a returns `{triggered:false, points:0}` with an explicit reason when its input is missing. No null is coerced. P9 has never awarded a point it should not have; it has failed to award points it should have. The failure is invisible in scores and visible only in the reason string, which reads `'endowment/FTE or peer median data not available'` — accurate, and identical to what a genuinely peer-less institution would produce.
- **2026-08 changes**: no dated comment appears inside `calculateP9_EndowmentDrainRisk()` itself. The two dated comments that govern its behavior are elsewhere:

`analysis_orchestrator.js` lines 376–385, quoted in full above.

`modular_tool_v48 no data cap.html` line 1037 (inside `SectorAwareAnalyzer.analyze()`, the method that drops `rawData`):

> *// 2026-08-30 (v49.2): honor an explicit data-gap declaration. Only // `success === false` triggers this — a guard that omits the flag // entirely is still treated as a successful analysis, as before.*

and `modular_tool_v48 no data cap.html` line 2024, in the endowment wrapper's own missing-institution guard:

> *// 2026-08-30 (v49.2): declared as a data gap so the base class drops it from the framework.*

The 2026-08-30 fail-closed resolver change (`risk_adjustment_analyzer.js:986–1023`) is **not** implicated in P9: P9 never calls `getMetricValue()`.

- **Notes**:
  - **`NOT_TRIGGERED` is spread, not returned.** Line 2622 defines `const NOT_TRIGGERED = { triggered: false, points: 0, reason: 'criteria not met' };` and every later rejection spreads it and overwrites `reason`. That is safe here, but the object is a shared const — a future exit that returned `NOT_TRIGGERED` directly and mutated it would corrupt every other exit.
  - **Two reason strings misreport the draw rate.** `draw3yr` is a fraction, but line 2672 prints `` `draw rate ${draw3yr.toFixed(1)}% exceeds 50% cap` `` and line 2675 prints `` `3yr draw rate ${draw3yr.toFixed(2)}% does not exceed 5.5% threshold` `` — both render the fraction as though it were already a percentage, so a 6.2 % draw reads as "0.06 %". The success path at line 2715 does it correctly: `` `3yr draw rate ${(draw3yr * 100).toFixed(2)}% exceeds 5.5% NACUBO buffer` ``. The comparisons themselves are correct; only the display is wrong. Since P9 never triggers, **the only reason strings a reader has ever seen from gate 2 are the two wrong ones.** (Fixed in v49.11: both now multiply by 100.)
  - **P9 is permanently reported as `not_reached`.** `_annotateGateStatus()` (lines 802–810) classifies a gate by its `__gateLedger` record, which is populated only by `getMetricValue()` calls. P9 makes none, so `rec` is always null, and the untriggered branch stamps `res.status = 'not_reached'`. The gate tally `window.__RA_GATE_TALLY__.P9_EndowmentDrainRisk` will therefore show 100 % `not_reached` for every institution — including the 158 that reached and failed gate 3. The instrumentation built to surface exactly this class of defect cannot see it, because P9 bypasses the instrumented resolver. The same blind spot applies to A10, A2 and A6, and the comment at lines 803–806 acknowledges it for the triggered case only. (Since v49.10: an untriggered P9 is stamped unledgered and prints its reason.)
  - The design comment at line 2605 says *"Endowment/FTE in bottom quartile vs Carnegie peers"* and line 2697 says *"P25 of EFT/median ratio across all private institutions = 0.480"*. The peer group actually supplying `peerMedian` is whatever `peerAnalysis.statistics.perFTECurrentValue.median` was built from in `unified_endowment_analyzer.js`. **UNCLEAR**: whether that peer group is Carnegie-matched in every case is not determinable from P9's own code; it depends on the `peerCriteria` passed to the endowment analyzer by the orchestrator.
  - The `BOTTOM_QUARTILE_RATIO = 0.480` calibration is documented as empirically derived from the sector distribution. Because gate 3 has never executed with real values, **that threshold has never been exercised against live data.** Any expectation about P9's firing rate once gate 3a is repaired rests on an uncalibrated constant. (Exercised since v49.11. On the 18 October anchor run, P9's peer medians ranged from $19,623 (Carnegie 19) to $158,058 (Carnegie 21); Madonna and Viterbo fail gate 3 at ratios of 1.155 and 1.386.)
  - P9 returns `reason` at every exit, and never `reasoning` — the opposite convention to P7 and P8. Across the three patterns in this spec the key is inconsistent both between methods and, in P7 and P8, within a single method.
  - Unrelated, but in the same shared resolver that P7 and P8 depend on: `metricAliases` (lines 516–620) is an object literal with **duplicate keys**. `instructional_percentage` is defined at 554 and again at 608, `administrative_percentage` at 557 and 611, `student_services_percentage` at 560 and 614. In a JS object literal the later definition wins, so the 2026-08-30 entries that add `'currentRate'` (lines 554–562, added precisely so those three gates could resolve under the fail-closed resolver) are silently overwritten by the earlier-numbered, later-positioned entries that omit it. P7 and P8 do not read those three keys, so this spec's patterns are unaffected; the expenditure gates are.

U38 and U39 — unscored flags (v49.17)

Points 0 (unscored) · counted on the 19 October 2026 private run (1,000 institutions, v49.17)

- **Method:** `__computeUnscoredFlags()`, from line 2832. Published on the risk adjustment record as `unscoredFlags` (U38 and U39, each with `triggered`, `reason` and `runwayYears`).
- **What they are for:** neither adds a point. They are published so the gaps in README items 38 and 39 can be counted on full runs before anything is priced. The order set for item 38 is: flag first, then the case-library test, then pricing.
- **Where they print:** a header line, `- Unscored Flags (no points):`, under `R11 Would Trigger`. It reads none or lists each flag with its reason.
- **U38 — endowment carrying a structural deficit** (item 38). Private only. The structural margin is Critical; the endowment draw is above 5.5%, taking the larger of the current year and the three-year average and ignoring any draw above 50%, as P9 does; and neither A10 nor P9 fired. It marks the gap between A10 and P9 described under A10.
- **U39 — R10 credited on a structural deficit** (item 39). R10 fired, and either the structural margin's three-year average is −10% or worse, or A10 or P9 fired. R10 keeps its −8.
- **Runway:** both print the endowment runway in years: beginning-of-year endowment ÷ (−structural margin three-year average × total operating revenue), where the margin is negative. Revenue is IPEDS total operating revenue, which includes investment return, so the deficit is overstated in dollars and the runway is conservative. Moving the denominator to revenue excluding investment return is deferred to the 2025 data build.
- **Counts (19 October private run, v49.17):** U38 fires at 108, with 5 of the 2024–26 closures (4.6%, against 2.9% for the run); all 5 have a runway under 10 years. Its holders include Viterbo, Madonna, Valparaiso, Lasell, Lindenwood, Suffolk, Columbia College Chicago, Clarkson, Beloit, Monmouth and Doane. U39 fires at 64, with no closures, and 45 of them are in the Low band. They are Harvard, Stanford, Yale, MIT, Amherst, Williams, Swarthmore and similar colleges, where a structural margin of −10% or worse is the endowment model by design, not a deficit.
- **Decisions (19 October 2026):** U38 stays unscored. Its lift is real but small, and a runway cut (under 5, 7.5 or 10 years, with revenue excluding investment return) does not separate the closures. The proposed pricing for U39, withdrawing R10, was rejected. If R10 is ever conditioned, it will be on runway measured without investment return, not on the structural margin.

4. The severity layer

The three severity conditions, the gate that counts them, the override they feed, and the critical mass computation that sits alongside. These are reported as found.

Financial Distress (`financial_distress` / `checkFinancialDistress`)

- **Method**: `checkFinancialDistress()` (lines 1229–1288)

Update (v49.10 and v49.13). Appropriations dependency is read only for publics (v49.10). For privates, marginWeak is the v2 operating margin's three-year average at or below −2.7%, and the research clause's margin leg is at or below −1.5% (`PRIVATE_MARGIN_GATES`, v49.13). Publics keep −2.0 and −3; community colleges keep −5.0 and −3. The −5.0 shown for "all other sectors" in the table below is the pre-v49.12 private cutoff. Against 21 private closures, mergers and exigencies announced 2024–26, the v2 three-year margin separates distressed privates far better than the structural margin with endowment gifts removed (AUC 0.735 against 0.622). Current method: from line 1320.

- **Points**: `points: triggered ? 12 : 0` (line 1285). Literal in the method; not from `ARCHETYPE_BASE_POINTS`.
- **What it tests**: six independent distress signals across reserves, viability, unrestricted net assets, three-year margin, enrollment and revenue dependency; requires a count of signals plus a count of "hard" (solvency/cash-flow) signals. The comment at 1246 reads `// ===== Thresholds tuned to reduce over-triggering in privates/R1s =====` and at 1264 `// "hard" = solvency/cash-flow`.
- **Inputs**:

Local	Source analyzer	Field requested	Route	Notes
`prPct`	`primaryreserve`	`primary_reserve_ratio`	`getMetricValue` → `keyMappings` → `currentValue`, wrapped in `this.pct()`	percent 0–100
`viRatio`	`viability`	`viability_ratio`	`getMetricValue` → `keyMappings` → `currentViabilityRatio`, wrapped in `this.num()`	raw ratio
`unaPct`	Unrestricted Net Assets	`una_per_fte_percentile`	`getUnaPercentile()` helper (1208–1222) via `pickAnalysis` + `getFirst` — **not** `getMetricValue`	0 = weakest peer, 100 = strongest
`aom3`	`adjustedoperatingmargin`	`three_year_average`	`getMetricValue` → `keyMappings` (identity)	percent
`enroll5`	`enrollment`	`5_year_change`	`this.getEnroll5?.() ?? this.getEnrollmentDeclineWorst?.()` (line 1238)	see Data-gap
`tuDep`	`tuitiondependency`	`tuition_percentage`	`getMetricValue` → `keyMappings` → `currentValue`, `this.pct()`	percent
`apDep`	`stateappropriations`, then `stateappropriationsdependency`	`dependency`	two `getMetricValue` calls joined by `??`, each wrapped in `this.pct()`	→ `currentValue`
`isResearch`	institution/peer criteria	`carnegieBasic`	`isResearchOriented()` → `isResearchCarnegie()` (322–330)	Carnegie 15, 16, 17 or 18

- **Gate logic**:

Signal definitions (lines 1248–1260):

Signal	Public (`sector === '1'`)	All other sectors	Direction
`reserveLow`	`prPct < 20`	`prPct < 20`	**ceiling**, strict (value < 20)
`viabilityWeak`	`viRatio < 0.35`	`viRatio < 0.80`	**ceiling**, strict
`unaThin`	`unaPct < 25`	`unaPct < 25`	**ceiling**, strict (bottom quartile)
`marginWeak`	`aom3 <= -2.0`	`aom3 <= -5.0`	**ceiling** (value <= X)
`enrollDecline`	`enroll5 <= -10`	`enroll5 <= -10`	**ceiling**
`depHigh`	`apDep >= 50`	`tuDep >= 70`	**floor** (value >= X)

dep is selected at line 1242: `const dep = (this.sector === '1') ? apDep : tuDep;`

Counts (1263–1264):

- `numericSignals` = count of the six above that are `true`.
- `hardCount` = count of `[reserveLow, viabilityWeak, marginWeak]` that are `true`.

Decision rule (1268–1270), transcribed:

```
const triggered = isResearch
  ? (numericSignals >= 4) && (hardCount >= 2) && (enrollDecline || (aom3 != null && aom3 <= -3))
  : (numericSignals >= 3) && (hardCount >= 1);
```

Cohort	Signals	Hard signals	Additional
Research (Carnegie 15–18)	**floor** ≥ 4	**floor** ≥ 2	`enrollDecline` OR `aom3 <= -3` (**ceiling**)
All others	**floor** ≥ 3	**floor** ≥ 1	—

- **Sector differences**: `viabilityWeak`, `marginWeak` and `depHigh` branch on `this.sector === '1'` only. Sectors `'2'` (Private), `'3'` (For-Profit) and `'4'` (Community College) all take the private branch. `reserveLow` and `unaThin` and `enrollDecline` are sector-invariant. Research status is orthogonal to sector and only affects the decision rule.
- **Data-gap behavior**: every signal is guarded by an explicit `!= null` / `!== null` test, so a missing input makes that signal `false` — it cannot make the gate fire, and it cannot block it either. A gate with four missing inputs can still trigger on the remaining two if they happen to be two hard signals plus... it cannot: the non-research rule needs three signals. With three or more inputs missing the gate can never fire for a non-research institution, and with three missing it can never fire for a research one. Nothing distinguishes "signal absent" from "signal present and benign" in the returned object; only the non-scoring `status` annotation from `_annotateGateStatus()` records it.
- **2026-08 changes**: the UNA leg, at lines 1251–1253:

> *// UNA cushion thin if in the bottom quartile of the peer group. // 2026-08-28: was `unaIdx < 0.60` on a ratio that inverted where the peer // median was negative. See getUnaPercentile.*

The governing rationale is in the `getUnaPercentile()` header (1187–1207):

> *// 2026-08-28 (v49.1): getUnaNorm is REPLACED by getUnaPercentile. // // It returned a ratio, unaPerFte / peerMedianPerFte, on which 1.0 meant "at the // peer median" and consumers tested `< 0.60`. That inverts whenever the peer // median is negative — which is the public case in Carnegie 18, 20, 21 and 22, // where GASB 68/75 pension and OPEB recognition drove roughly half of each group // below zero from FY2015. For those institutions the weakest cushions produced // the HIGHEST index values and the gates below read the signal backwards, for // about 267 publics. // // The old ratio fallback (current / peer) is deliberately gone as well: it had // exactly the same defect, so leaving it in place would have silently restored // the bug whenever the preferred field was missing. // // The analyzer now publishes una_per_fte_percentile: 0 = weakest in the peer // group, 100 = strongest, well-defined whatever the sign of any member. There is // no ratio and no denominator, so it cannot invert. // // THRESHOLD TRANSLATION. The old `< 0.60` was intended to mean "meaningfully // below the peer median". On a percentile that is `< 25` — the bottom quartile. // The old `<= 0.40`, the P5 escalation, becomes `<= 10` — the bottom decile.*

Also relevant: the `keyMappings` block quoted in the preamble names this gate as one of the three that had been reading a fourteen-year enrollment change (`checkFinancialDistress 12 pts, fired 345 times`).

- **Notes**:
  - **The enrollment fallback re-introduces the fourteen-year window this gate was repaired to stop using.** Line 1238 is `this.getEnroll5?.() ?? this.getEnrollmentDeclineWorst?.()`. `getEnroll5()` returns the five-year figure; when it is `null`, `getEnrollmentDeclineWorst()` (2792–2833) returns `Math.min` of the five-year and the **fourteen-year** change. So for any institution whose `shortTerm` block is absent, the `enroll5 <= -10` test is applied to a 2010→2024 decline — precisely the behavior the 2026-08-28 change record says was removed. The comment string published in `reasons` (line 1279) still reads `enrollment ↓5yr ≥10%` in that case.
  - **`getEnroll5()`'s own internal fallback is inert.** Lines 1128–1135 try `'5_year_change'`, then `'recent_change'`, then `'2018_2023_change'`. All three are mapped by `keyMappings` (898–900) to the identical path `shortTerm.totalChangePercent`. If the first returns `null`, so must the other two. The three-way chain does nothing.
  - **Community colleges (sector `'4'`) are scored on the private branch** for viability (`< 0.80` rather than `< 0.35`), margin (`<= -5.0` rather than `<= -2.0`) and dependency (tuition `>= 70` rather than appropriations `>=`

50), despite being public institutions subject to the same GASB 68/75 distortion the public thresholds exist to accommodate. No comment addresses this. applySeverityOverride treats sector '4' as public-like for its UNA leg (isPublicLike = (this.sector === '1' || this.sector === '4'), line 2210), so the two methods classify the same institution differently.

- **The research escalator's aom3 <= -3 is not sector-aware** while marginWeak is. For a private research institution the ordinary margin signal fires at <= -5 but the escalator is satisfied at <= -3, so the escalator can be satisfied by a margin that did not itself count as a signal. (Superseded for privates at v49.13: the research clause uses −1.5% and marginWeak −2.7%, both on the v2 operating margin.)
- apDep uses ?? between two this.pct(...) calls. pct() returns null (not undefined) on a missing value, and null ?? x evaluates x, so the second lookup does run. Correct, but note that both lookups are always executed when the first misses, each recording a ledger entry.
- unaPct bypasses getMetricValue entirely, so a missing UNA percentile leaves **no** entry in __gateLedger for this gate. The Stage 0 status annotation cannot see it.

Operational Inefficiency (operational_inefficiency / checkOperationalEfficiency)

- **Method**: checkOperationalEfficiency() (lines 1295–1400)

Update (v49.20, 19 October 2026). Instructional Support is informational in every sector (0 points, outside the maximum). This gate reads the instructional share first and falls back to the measure's points only when the share is missing; on the v49.20 validation runs the share was present wherever the measure ran, so no firing changed. Where the share is missing, the instructional fallback can no longer fire, since the measure's points are now 0.

- **Points**: points: triggered ? 3 : 0 (line 1389). Literal; not from ARCHETYPE_BASE_POINTS.
- **What it tests**: three cost-structure signals (instructional share low, administrative share high, student-services share high) plus two delivery signals (student/faculty ratio, retention). Each cost signal is tested peer-relatively where a peer mean is available and against an absolute floor where it is not. Header comment: // Operational Inefficiency (robust names + RP fallbacks).
- **Inputs**:

Local	Source analyzer	Field	Route
instrPct	instructionalexpenditure → instructional → instructional	currentRate, currentRate, instructional_percentage	three getMetricValue calls joined by ??
adminPct	institutionalexpenditure → institutional → institutional	currentRate, currentRate, administrative_percentage	as above
studPct	studentservicesexpenditure → studentservices → studentservices	currentRate, currentRate, student_services_percentage	as above
instrRP	instructional	riskPoints	getMetricValue (exact, step 4)
adminRP	institutional	riskPoints	getMetricValue
studentRP	studentservices	riskPoints	getMetricValue
sfrRisk	sfr → student_faculty_ratio	riskPoints	two getMetricValue calls joined by ??
retention	retention → retention → retention	currentRetentionRate, first_year_retention, currentValue	three getMetricValue calls joined by ??
instrPeer	instructionalexpenditure	analysisResults.peerAnalysis.statistics.currentRate.mean	**direct completedAnalyses read** via the local peerMean() helper (1335–1338), exact analyzerKey === match
adminPeer	institutionalexpenditure	as above	as above
studPeer	studentservicesexpenditure	as above	as above

Local	Source analyzer	Field	Route
`isResearch`	institution/peer criteria	`carnegieBasic`	`isResearchOriented()`

`currentRate` is written onto `analysisResults` by the orchestrator's pass-through block (`analysis_orchestrator.js` lines 341–366) for exactly these three analyzer keys, falling back to `expenditureAnalysis.expenditureRate.current`, `expenditureAreaAnalysis.expenditureRate.current`, and finally a regex scrape of the report's key findings (`_extractPercentFromReport`).

- **Gate logic**:

Cut-points and pads (lines 1326–1347):

Constant	Private (`sector === '2'`)	All other sectors	Research override
`adminCut`	26	27	—
`studCut`	12	13	—
`instrFloor`	35	40	—
`retCut`	—	—	76 if research, else 78
`sfrCut`	—	—	`isResearch ? 2 : 2`
`strongRP`	—	—	6 if research, else 5
`ADMIN_PAD`	—	—	6 if research, else 4
`STUD_PAD`	—	—	6 if research, else 4
`INSTR_PAD`	—	—	7 if research, else 4

Cost signals (1350–1372):

Signal	Peer available	Peer absent	Direction
`instrLowPct`	`instrPct < (instrPeer - INSTR_PAD)`	`instrPct < instrFloor`	**ceiling**, strict
`adminHighPct`	`adminPct > (adminPeer + ADMIN_PAD)`	`adminPct > adminCut`	**floor**, strict
`studHighPct`	`studPct > (studPeer + STUD_PAD)`	`studPct > studCut`	**floor**, strict

Risk-point fallbacks, taken **only** when the corresponding percentage is `null` (lines 1366–1368):

Signal	Condition	Direction
`instrLowFB`	`instrPct == null && instrRP >= strongRP`	**floor** (≥ 5, or ≥ 6 research)
`adminHighFB`	`adminPct == null && adminRP >= strongRP`	**floor**
`studHighFB`	`studPct == null && studentRP >= strongRP`	**floor**

Delivery signals (1377–1379):

Signal	Condition	Direction
`sfrHigh`	`sfrRisk >= sfrCut` (i.e. `>= 2`)	**floor**
`retLow`	`retention < retCut` (76 research / 78 other)	**ceiling**, strict

Decision rule (line 1385), transcribed:

```
  const triggered = (costStrongCount >= 2) || (costStrongCount === 1 && deliveryWeakCount === 2);
```

where `costStrongCount = [instrLow, adminHigh, studHigh].filter(Boolean).length` and `deliveryWeakCount = [sfrHigh, retLow].filter(Boolean).length`. The accompanying comment reads:

> *// ----- decision rule (tightened) ----- // To avoid 90%+ prevalence in R1s: // Require TWO cost signals; OR // Require ONE cost signal PLUS BOTH delivery weaknesses.*

- **Sector differences**: `adminCut`, `studCut` and `instrFloor` branch on `sector === '2'` only, so publics, for-profits and community colleges share one set. The peer-relative path, when it resolves, replaces all three absolute cuts, making the sector split moot for any institution with a peer group.
- **Data-gap behavior**: a missing percentage routes to the analyzer's own risk points; a missing peer mean routes to the absolute floor. A signal with neither a percentage nor risk points is `false`. `retention === null` makes `retLow` false; `sfrRisk` null makes `sfrHigh` false. The gate degrades toward not-firing.
- **2026-08 changes**: the retention chain, at `metricAliases` lines 537–543:

> *// 2026-08-30 (v49.2): the second and third links of the retention chains in // checkOperationalEfficiency and checkA7_PellPressureCooker name fields the // retention analyzer has never published (it publishes currentRetentionRate). // Under the old resolver they fell through to generic substitution; under the new // one they would simply return null. Declared as aliases instead, so the chains // mean what they appear to mean. No behavioral change while the first link // resolves — which it does whenever the retention analyzer ran.*

And on the percent-of-spend names (lines 550–553):

> *// Likewise the percent-of-spend names: the expenditure analyzers publish the // figure as `currentRate` (and the orchestrator copies it onto the record for // exactly this reason). perKeyVariations sends these to `currentPercentage`, // which none of the three analyzers publishes.*

- **Notes**:
  - **`sfrCut = isResearch ? 2 : 2` (line 1330) is a ternary with identical branches.** The trailing comment `// require 2/2 RP for SFR to count everywhere` explains the intent, and `UnifiedStudentFacultyRatioAnalyzer` publishes `maxRiskPoints: 2`, so the test is "full marks on a 2-point measure". The branch is dead but the value is correct.
  - **The percent-of-spend alias repair (2026-08-30) does not take effect.** As documented in the preamble, the `instructional_percentage` / `administrative_percentage` / `student_services_percentage` alias entries declaring `currentRate` at lines 554–562 are overwritten by duplicate declarations at lines 608–616. The third link of each cost chain — `get('instructional','instructional_percentage')` and its two siblings — therefore still returns `null`: `keyMappings` sends it to `currentPercentage` (unpublished), step 4b now resolves only `instructional_spend_pct` (unpublished), `perKeyVariations` offers only `currentPercentage/instructionalPercentage/currentPercent/percent/share/ percentOfTotal` (none published), and the container sweep does not reach a top-level field. The chain's **first** link (`instructionalexpenditure.currentRate`) resolves whenever the orchestrator pass-through ran, so no scoring difference is currently observable — but the comment claims a repair that is not in force.
  - **The second link of each cost chain is a duplicate of the first.** `analyzerAliases.instructional = ['instructional','instructionalsupport','instructionalexpenditure']` (line 507), so `get('instructional','currentRate')` resolves to the same analyzer record as `get('instructionalexpenditure','currentRate')`. If the first returns `null`, so does the second. Same for `institutional` (508) and `studentservices` (509).
  - **`peerMean()` uses exact equality on `analyzerKey`** (`analyses.find(x => x?.analyzerKey === key)`), unlike every other lookup in the class, which normalizes. The keys it hard-codes match what the orchestrator produces (`analyzer.constructor.name.toLowerCase().replace('universal','').replace('analyzer','')` over `UniversalInstructionalExpenditureAnalyzer` → `instructionalexpenditure`), so it works — but it is brittle to any rename of the analyzer class, and it will fail silently (peer mean `null` → fall back to the absolute floor) rather than error.
  - **`retention` published as `currentRetentionRate: retentionAnalysis.retentionRate?.current || null`** (`unified_retention_analyzer.js` line 133) — a `||` on a numeric field, so a genuine 0% retention becomes `null` and `retLow` is `false`. Out of scope for this document but it is this gate's input.
  - **The peer-relative path can invert the intended direction of the absolute cut.** For an institution whose peer mean administrative share is, say, 34%, `adminHighPct` requires `adminPct > 38` (non-research), i.e. a materially *higher* bar than `adminCut = 27`. Where the peer mean is low the bar is *lower* than the absolute cut. This is deliberate per the comment at 1343–1344 (`// Pads from peer mean: we call it "material" only if above mean by these many pts.`), but it means two institutions with identical spending shares can be scored differently on peer composition alone, with no floor or ceiling on the resulting bar.

- The absolute-floor branch for `instrLowPct` uses a **lower** floor for privates (35) than for publics (40), meaning a private is judged to have a low instructional share only at a more extreme value. No comment explains the direction.

Enrollment Cliff (`enrollment_cliff` / `checkEnrollmentCliff`)

- **Method**: `checkEnrollmentCliff()` (lines 2929–2946)
- **Points**: `points: shouldTrigger ? 10 : 0` (line 2941). Literal; not from `ARCHETYPE_BASE_POINTS`.
- **What it tests**: an enrollment decline that is either corroborated by a loss of market share against the peer group, or is severe enough on its own. The published reason string is `Enrollment cliff: <n>% decline indicating market position risk`.
- **Inputs**:

Local	Source analyzer	Field	Route
`enrollmentChange`	`enrollment`	`5_year_change`	`getMetricValue` → `keyMappings` → deep path `shortTerm.totalChangePercent`, wrapped in `this.num()`
`marketPosition`	`enrollment`	`market_share_change`	`getMetricValue` → step 4 exact match on `analysisResults.market_share_change`, wrapped in `this.num()`

`market_share_change` is published by `unified_enrollment_analyzer.js` line 125 from `calculateMarketShareChange()` (lines 548–566): the institution's share of (institution + peer group) enrollment at the endpoint versus the baseline, expressed as a percent change of that share. It returns `null` when the peer group has fewer than five members (`if (n < 5) return null; — // Fail closed: a share change built on a thin or lopsided group is noise.`) or when any of the four totals is non-finite or non-positive.

- **Gate logic** (lines 2934–2937), transcribed:

```
  const severeDecline = (enrollmentChange !== null) && enrollmentChange <= -10;
  const marketLoss = (marketPosition !== null) && marketPosition < -10;
  const shouldTrigger = severeDecline && (marketLoss || (enrollmentChange !== null && enrollmentChange <= -
30));
```

Test	Threshold	Direction
`severeDecline`	`enrollmentChange <= -10`	**ceiling** (value <= −10)
`marketLoss`	`marketPosition < -10`	**ceiling**, strict (value < −10)
Solo escalation	`enrollmentChange <= -30`	**ceiling**

Both a five-year enrollment decline of at least 10% **and** one of {market-share loss greater than 10%, five-year enrollment decline of at least 30%} are required.

- **Sector differences**: none. No `this.sector` reference anywhere in the method.
- **Data-gap behavior**: `enrollmentChange === null` → `severeDecline` false → gate cannot fire. `marketPosition === null` → `marketLoss` false → the gate can still fire, but only via the `<= -30` escalation. Because `market_share_change` fails closed for peer groups under five, any institution in a thin peer group faces a de facto 30% bar rather than a 10% one. Both null cases are recorded in `__gateLedger` (both reads go through `getMetricValue`), so the gate is annotated `unavailable` when untriggered with an unresolved input.
- **2026-08 changes**: no dated comment inside the method. The governing change is in `keyMappings`, quoted in full in the preamble; the sentence naming this gate is:

> *// checkEnrollmentCliff 10 pts, fired 593 times (38.2% of the pool)*

and

> *// A 25% decline over five years is an emergency; over fourteen it is a slow // contraction a great many institutions have had. The gates were far easier // to open than anyone believed, which is why enrollment cliff was the second // most frequent element in the whole layer.*

The method also carries an undated repair note at line 2933: `// FIXED: Use explicit null checks instead of truthiness`.

- **Notes**:
  - **The two thresholds are not on commensurable quantities**, though the method reads as if they were. `enrollmentChange` is a percent change in headcount; `marketPosition` is a percent change in a *share*. Both are compared against `-10`. The comparison is defensible but the identical cut-point is not derived anywhere.
  - **The redundant re-test at line 2937.** `shouldTrigger` re-checks `enrollmentChange !== null` inside a conjunction already guarded by `severeDecline`, which cannot be true when `enrollmentChange` is null. Harmless.
  - **The reason string names market position even when market position was never read.** When the gate fires on the `<= -30` branch with `marketPosition === null`, the published reasoning still reads `...decline indicating market position risk`. There is no market-position evidence in that case.
  - This gate reads `'5_year_change'` directly rather than through `getEnroll5()`, so — unlike `checkFinancialDistress` — it has **no** fallback to `getEnrollmentDeclineWorst()` and cannot silently revert to the fourteen-year window. This is the correct behavior and differs from the sibling gate.

Severity Condition (gate) (checkSeverityCondition)

- **Method**: `checkSeverityCondition()` (lines 2154–2167)
- **Points**: none directly. It returns a structured flag consumed by `applySeverityOverride()`, `calculateP3_AppropsDepEnrollDecl()`, `calculateP4_NetTuitionRisk()` and the debug snapshot in `buildDetailedReport()`.
- **What it tests**: whether either of two extreme conditions holds — a severe enrollment decline, or a viability ratio at or below 0.30. It is the front gate for the severity override; nothing else in Layer 2 can price distress at the override's level without it.
- **Inputs**:

Local	Source analyzer	Field	Route
`enroll5`	`enrollment`	`5_year_change` (→ `recent_change` → `2018_2023_change`)	`this.getEnroll5()` (1128–1135), all three mapped to `shortTerm.totalChangePercent`
`viability`	`viability`	`viability_ratio`	`getMetricValue` → `keyMappings` → `currentViabilityRatio`, wrapped in `this.num()`

- **Gate logic** (lines 2157–2161), transcribed:

```
const severeEnrollment = (enroll5 !== null && enroll5 <= -25);
const weakViability = (viability !== null && viability <= 0.30);
...
hasSeverity: (!!severeEnrollment || !!weakViability),
```

Flag	Threshold	Direction
`severeEnrollment`	`enroll5 <= -25`	**ceiling** (value <= −25)
`weakViability`	`viability <= 0.30`	**ceiling** (value <= 0.30)
`hasSeverity`	OR of the two	—

Returned shape: `{ hasSeverity, severeEnrollment, weakViability, enroll5, viability }`.

- **Sector differences**: **none.** `viability <= 0.30` is applied to publics, privates, for-profits and community colleges alike, even though `checkFinancialDistress` uses `< 0.35` for publics and `< 0.80` for everyone else, and even though `unified_viability_analyzer.js` scores publics on a percentile-relative branch precisely because "GASB 68/75 drives public expendable net assets negative sector-wide, so the 0.10/0.20/0.30/0.40 bands degenerate into a sign test".
- **Data-gap behavior**: both flags are explicitly null-guarded, so a missing input yields `false` and, if both are missing, `hasSeverity: false` — `applySeverityOverride()` then returns `{applied:false, reason:'No severity'}` at line 2173 without reading any corroborator, and `this.overrideTriggerList` is never assigned, publishing as `[]`. Neither read is distinguished from a resolved-and-benign read in the returned object.

- **2026-08 changes**: no dated comment inside the method. Governing change is the enrollment window remap in keyMappings, whose relevant line is:

> // *checkSeverityCondition arms the override for 325 institutions*

- **Notes**:
  - **getEnroll5()'s three-link chain is inert** for the reason given under Financial Distress: '5_year_change', 'recent_change' and '2018_2023_change' all resolve to shortTerm.totalChangePercent.
  - **This gate has no fallback to getEnrollmentDeclineWorst()**, unlike checkFinancialDistress (line 1238). Consequently an institution with no shortTerm block cannot arm the severity override on enrollment at all, while the same missing data would push Financial Distress onto the fourteen-year series. The two methods handle the same gap in opposite directions.
  - A viability ratio of exactly 0.30 arms the override (<=); a primary reserve of exactly 20% does not fire reserveLow in Financial Distress (<). Boundary conventions are not uniform across the layer.

Severity Override / A11 Severity Pattern (applySeverityOverride)

This is where the layer's total is decided when the severity gate is armed. It is the "assembly" half of the requested scope.

- **Method**: applySeverityOverride(sev) (lines 2172–2269), called from analyze() Step 5 (line 3317).

Update (v49.13, 19 October 2026). For privates the margin trigger aomBad is the v2 operating margin's three-year average at or below −2.7% (PRIVATE_MARGIN_GATES.severity). The aom3 <= -5 shown for "other sectors" in the table below is the pre-v49.12 private cutoff, which community colleges keep. Publics keep −2. Current method: from line 2344.

- **Points**: 12, 16 or 20, from the ternary at line 2242 — const severityPoints = corroborators <= 1 ? 12 : (corroborators === 2 ? 16 : 20); These values are **not** in ARCHETYPE_BASE_POINTS, and there is no A11 key in either ARCHETYPES or ARCHETYPE_BASE_POINTS.
- **What it tests**: whether the armed severity gate is corroborated by at least one further distress signal from a list of five, at a combined trigger count of three or more. If so it reprices the whole layer at a floor, rather than adding to it. The design principle is stated at lines 2244–2245:

> *// max-not-sum preserved: severity is an ALTERNATIVE pricing of distress already // counted in the analyzers and in A1/A6/A7/A9/P5, never an addition to it.*

- **Inputs**:

Local	Source analyzer	Field	Route
approps	stateappropriations → stateappropriationsdependency	dependency	getMetricValue ×2 joined by ??, then this.pct()
tuitionDep	tuitiondependency	tuition_percentage	getMetricValue, this.pct()
aom3	adjustedoperatingmargin	three_year_average	getMetricValue, this.num()
ntrNorm	Net Tuition per FTE	net_tuition_per_fte_normalized, else currentValue / peerAverage	getNtrNorm() helper (1139–1182) — **direct completedAnalyses read**, not getMetricValue
una	unrestrictednetassets	total → currentNetAssets	getMetricValue ×2 joined by ??, this.num()
unaYrs	unrestrictednetassets	years_negative	getMetricValue → keyMappings identity
unaCritical	unrestrictednetassets	riskLevel	getMetricValue, compared === 'Critical' (case-sensitive)
apps14	acceptanceyield	applications_change_since_2014	getMetricValue → keyMappings → applicationChangePercent
accept	acceptanceyield	acceptance_rate	→ currentAcceptanceRate, this.pct()
yieldR	acceptanceyield	yield_rate	→ currentYieldRate, this.pct()
sev.*	—	—	passed in from checkSeverityCondition()

- **Gate logic**:

Front gate (line 2173): if (!sev?.hasSeverity) return { applied: false, reason: 'No severity' };

Corroborators (lines 2190–2216):

Trigger	Public (`sector === '1'`)	Other sectors	Direction
`depOK`	`approps >= 35`	`tuitionDep >= 70`	**floor**
`aomBad`	`aom3 <= -2`	`aom3 <= -5`	**ceiling**
`ntrWeak`	`ntrNorm <= 0.70`, OR `ntrNorm <= 0.80` with a paired stressor	same	**ceiling**
`unaBad`	`unaCritical` (public-like: sector `'1'` or `'4'`)	`una < 0` AND (`unaYrs >= 3` OR `unaCritical`)	ceiling on una, **floor** on unaYrs
`brandCollapse`	`apps14 <= -20` AND (`accept >= 90` OR `yieldR <= 10`)	same	**ceiling** / **floor** / **ceiling**

The `ntrWeak` "paired stressor" set (lines 2196–2197): `sev.enroll5 <= -15` OR `apps14 <= -10` OR `accept >= 90` OR `yieldR <= 10`.

The two gate conditions from `checkSeverityCondition` are also pushed into the same list (lines 2227–2228): `sev.weakViability` → `'weak viability (≤0.30)'`, `sev.severeEnrollment` → `'severe enrollment decline (≤-25%)'`.

Decision (line 2234): `if (triggers >= 3)` where `triggers = firedTriggers.length`. Comment at 2232–2233:

> *// Option D: both gate conditions are pushed into firedTriggers, so >= 2 // collapses to gate-plus-nothing. Require the gate plus two real corroborators.*

Pricing (lines 2240–2242):

```
const gateCount = (sev.weakViability ? 1 : 0) + (sev.severeEnrollment ? 1 : 0);
const corroborators = Math.max(0, triggers - gateCount);
const severityPoints = corroborators <= 1 ? 12 : (corroborators === 2 ? 16 : 20);
```

Corroborators	Points
0 (unreachable — see Notes)	12
1	12
2	16
3, 4 or 5	20

Assembly (lines 2246–2249), transcribed exactly:

```
const archetypePatternTotal = this.calculateTotalAdjustmentPoints();

this.totalAdjustmentPoints = Math.min(50, Math.max(severityPoints, archetypePatternTotal));
this.overrideLiftPoints = Math.max(0, this.totalAdjustmentPoints - archetypePatternTotal);
```

On the non-firing path (2262–2267) `this.overrideTriggerList = firedTriggers;` is still assigned and `applied: false` is returned with the near-miss list in the reason string.

- **Sector differences**: `depOK` and `aomBad` branch on `sector === '1'` only. `unaBad` branches on `isPublicLike = (this.sector === '1' || this.sector === '4')`. `ntrWeak` and `brandCollapse` are sector-invariant.
- **Data-gap behavior**: every corroborator is explicitly null-guarded, so missing inputs make the corroborator `false`. A missing `sev` (null/undefined) short-circuits to `'No severity'`. Note that `unaCritical` is computed as a strict `=== 'Critical'` comparison against a value that may be `null`, so a missing risk level makes it `false` rather than throwing.
- **2026-08 changes**: the UNA leg carries a full design-intent block (lines 2199–2209):

> *// 2026-08-28: SECTOR-AWARE. Repairing the years_negative mapping above made // unaYrs resolve for the first time — and for publics a long negative-UNA // streak is a GASB 68/75 artifact, not a finding: the share of publics with // negative unrestricted net assets went 12% -> 50% in FY2015 and 50% -> 66% // in FY2018 with no operational change, and most have been continuously // negative since. Left unqualified, the repair would have handed this // severity corroborator to roughly half the public sector for an accounting // change. Publics are therefore corroborated on the peer-relative signal // (the UNA analyzer's own risk level, which is now percentile-scored); // privates, where negative UNA is 3% of the sector and means what it // appears to, keep both the streak and the level test.*

The mapping repair it refers to (lines 939–943):

> *// 2026-08-28: was mapped to 'yearsNegative', which the UNA analyzer has never // published — it publishes 'years_negative'. The severity override's unaBad leg // (line ~1897) therefore always read undefined and fell back to its riskLevel // test. Seventh instance of a gate reading a field its producer does not publish.*

And the near-miss publication (2259–2261):

> *// Publish the fired triggers even when the gate is not met: near-misses need to // stay visible for calibration, and a populated gate value next to "Triggers: none" // reads as a defect. triggerCount/triggers feed the report header at ~3611.*

The A11 header (2237–2239):

> *// A11 — Severity Pattern. Concentrated distress priced in POINTS like every // other archetype. No reference to any band boundary. // Gate conditions counted via structured flags, not trigger strings.*

- **Notes**:
  - **The stated decision rule and the implemented one differ when both gate conditions fire.** The comment says "Require the gate plus two real corroborators." The code requires `triggers >= 3` where `triggers` *includes* the gate conditions. An institution with both `weakViability` and `severeEnrollment` needs only **one** corroborator to reach 3 and is priced at 12 points. Only an institution with a single gate condition actually needs two corroborators. The comment describes the single-gate case as if it were general.
  - **`corroborators === 0` is unreachable, so the `<= 1` in the pricing ternary can only ever mean `=== 1`.** `triggers >= 3` and `gateCount <= 2` force `corroborators >= 1`.
  - **The `40` branch of `depOK` is unreachable.** Line 2191 reads `approps >= (sev.hasSeverity ? 35 : 40)`, but line 2173 has already returned when `hasSeverity` is falsy. The public appropriations bar is always 35.
  - **The `sev.hasSeverity &&` conjunct in `ntrWeak` (line 2195) is likewise always true** and therefore redundant, for the same reason.
  - **`unaBad` for publics reduces to `unaCritical` alone**, meaning the `una` and `unaYrs` lookups are performed and discarded for sectors `'1'` and `'4'`. Two ledger entries are recorded for reads that cannot affect the result.
  - **`unaCritical` is a case-sensitive `=== 'Critical'` string comparison** on a raw `getMetricValue` return, unlike the `.toLowerCase()` normalization used in `checkCriticalMassEscalation` (line 3081) and `calculateP9_EndowmentDrainRisk` (line 2646). A producer publishing `'critical'` would not match.
  - **The override can only raise the total, never lower it** (`Math.max`). Since `severityPoints` maxes at 20 and `calculateTotalAdjustmentPoints()` is already capped at 50, an institution whose archetypes and patterns already sum above 20 sees `applied: true`, `liftPoints: 0`, and no change to its score. The `applied` flag in the published `severityOverride` object therefore does **not** mean the score was affected; `liftPoints` is the field that says so.
  - **`floorTarget` is always `null`** (line 2256) on the applied path and absent on the non-applied path, resolving to `null` in `analyze()` (line 3372). The field name is a survival from a band-floor implementation the A11 comment explicitly disclaims ("No reference to any band boundary"). It is still published to the orchestrator, which copies it into `results.summary.severity_override`.
  - `getNtrNorm()` bypasses `getMetricValue`, so `ntrWeak`'s primary input leaves no ledger entry and cannot be seen by the Stage 0 status annotation.
  - The `reason` string names this "Severity pattern (A11)", but `A11` is registered in `__gateFnByKey` as `'__computedOutsideLedger'` (line 432) while **no `A11` key is ever created in `archetypeFlags`**. That ledger entry is dead: `_annotateGateStatus()` iterates the map's own keys, and `A11` is never one of them.

Adjustment Risk Level (`calculateAdjustmentRiskLevel`)

- **Method**: `calculateAdjustmentRiskLevel()` (lines 2762–2767), called once, from the `analyze()` return at line 3362.
- **Points**: none. Produces the `riskLevel` string published on the risk-adjustment analyzer's own `analysisResults`.
- **What it tests**: the Layer 2 total as a fraction of the 50-point cap.
- **Inputs**: `this.totalAdjustmentPoints` only. No analyzer reads.
- **Gate logic** (transcribed in full):

```
const pct = this.totalAdjustmentPoints / 50; // using your 50 cap
if (pct >= 0.45) return 'High';
```

```
if (pct >= 0.30) return 'Moderate';
return 'Low';
```

Band	Fraction	Points equivalent	Direction
High	`pct >= 0.45`	≥ 22.5 (i.e. ≥ 23 in integer points)	**floor**
Moderate	`0.30 <= pct < 0.45`	15 to 22	**floor**
Low	`pct < 0.30`	≤ 14, including negative	—

- **Sector differences**: none.
- **Data-gap behavior**: none possible — the input is an internal accumulator initialized to `0` in the constructor (line 445). On the `analyze()` catch path (3416–3423) the published level is the literal `'Error'`, not a value from this method.
- **2026-08 changes**: none. No dated comment.
- **Notes**:
  - **The bands are hard-coded here and are not the framework's own bands.** `riskArchitecture.determineRiskLevel()` in the orchestrator (line 619) is what bands the composite; this method's High/Moderate/Low is a separate, undocumented scale that applies only to the Layer 2 sub-score.
  - **A negative total returns `'Low'`.** An institution with R10 and R12 both firing (−8 and −6) and nothing else reaches −14, `pct = -0.28`, and is labeled `'Low'` — the same label as an institution at 0 or at +14. The scale has no band for a net credit.
  - The threshold `0.45` on a 50-point cap lands on 22.5 points, which no integer total can equal; the effective floor is 23.
  - The published `maxRiskPoints: 50` (line 3361) accompanies this level, but the orchestrator excludes the risk-adjustment record from the denominator (see the assembly section), so the 50 is never used as a divisor.

Pattern Add-on registry (`calculatePatternAddons`)

- **Method**: `calculatePatternAddons(severityCondition)` (lines 2848–2866), called from `analyze()` Step 3 (line 3308).
- **Points**: none of its own. It builds `this.patternAddons`, whose members' `points` are summed by `calculateTotalAdjustmentPoints()`.
- **What it tests**: nothing. It is the registration point that decides which pattern methods run.
- **Inputs**: `severityCondition` (from `checkSeverityCondition()`), forwarded to P3 and P4 only.
- **Gate logic**: the registry, transcribed (lines 2851–2860):

Key	Method	Severity passed	Max points	Sector restriction (from the method)
`P1_SienaPattern`	`calculateP1_SienaPattern()`	no	4 + 2 = 6	`sector !== '2'` → not triggered (line 2432)
`P2_HighDiscount_FlatEnroll`	`calculateP2_HighDiscount_FlatEnroll()`	no	`P2_BASE_POINTS` 4, +1, +1 = 6	`sector !== '2'` → not triggered (2302)
`P3_AppropsDep_EnrollDecl`	`calculateP3_AppropsDepEnrollDecl(severityCondition)`	**yes**	10 + 2 = 12	`sector !== '1'` → not triggered (2355)
`P4_NetTuitionRisk`	`calculateP4_NetTuitionRisk(severityCondition)`	**yes**	7 (or 4 on the fallback path)	sector `'4'` / name contains

Key	Method	Severity passed	Max points	Sector restriction (from the method)
				"community" excluded (2382)
P5_UNA_Risk	calculateP5_UNARisk()	no	6 + 2 = 8 (or 4 on fallback)	sector '4' / "community" excluded (2455)
P7_BrandPositioning	calculateP7_BrandPositioning()	no	6 + 2 = 8	sector '4' / "community" excluded (2528)
P8_MarketPosition	calculateP8_MarketPosition()	no	5 or 7	sector '4' / "community" excluded (2560)
P9_EndowmentDrainRisk	calculateP9_EndowmentDrainRisk()	no	4	sector !== '2' → not applicable (2625)

There is **no P6**. After the map is built, `this._annotateGateStatus(this.patternAddons)` runs (line 2862) — non-scoring.

- **Sector differences**: enforced inside each pattern method, not here. All eight run for every institution; five of the eight return `{triggered:false, points:0}` immediately for a given sector.
- **Data-gap behavior**: the registry has none of its own. Each method returns a `{triggered:false, points:0}` object with a `reasoning` or `reason` string on missing data; `calculateTotalAdjustmentPoints()` reads `pattern.points` only when `pattern.triggered`.
- **2026-08 changes**: the P6 removal, at lines 2500–2522, quoted verbatim:

> *// P6 — Research Coverage Erosion (research-data only; no Carnegie dependency) // RETIRED 2026-08-09. Not registered in __gateFnByKey and not evaluated in the // pattern registry; retained unreferenced for the record. Do not re-register // without addressing all three reasons it was retired: // 1. The coverage fields it reads were never published. research_grants_analyzer // computes rexpCAGR, ggcCAGR and ratioCAGR but drops `trends` when building // analysisResults, so covNow, covTrend and expenseTrend were always null and // signalCount was always 0. Severity 2 never fired at any institution. // 2. Units mismatch. COV_TREND_TRIP and SPREAD_TRIP are 5-year point changes; // the only available research trends are annualized CAGRs. Publishing CAGR // under the *_5y key names would repeat the net revenue ratio scaling defect. // 3. covNow has no source at all. Only the ratio trend is computed, not the level, // so two of the three signals cannot be revived by publishing alone. // Validation would also need a research-intensive cohort: the research analyzer is // not_applicable at all 66 Carnegie-22 publics, so the standing control cannot test it. // 2026-08-28 (v49.1): calculateP6_ResearchCoverageErosion REMOVED. // A complete implementation reading twenty measured fields that was absent from // calculatePatternAddons() and from __gateFnByKey, so it had no call site anywhere // and fired 0 times. All three of its coverage inputs also read fields no analyzer // publishes, so wiring it as written would not have made it fire either. Both the // architecture documentation and the analyzer catalog described it as live. // Rebuild it against real published fields if research coverage erosion is wanted.*

And the P1 duplicate-definition removal, at lines 2274–2280:

> *// P1 - Siena Pattern (Privates only: high tuition dependency + weak retention/brand) // 2026-08-28 (v49.1): the FIRST of two identical-signature definitions of // calculateP1_SienaPattern was removed here. A later definition in a class body // silently overwrites an earlier one, so this block had never executed — the // documented Siena logic and its 10+2 point award were dead code, while the // definition further down (the 4+2 version) was the one actually scoring P1. // The surviving definition is unchanged.*

- **Notes**:
  - `__gateFnByKey` (lines 435–442) registers exactly the eight keys this method builds, plus nothing for P6. Consistent.

- **Only P3 and P4 receive `severityCondition`.** P1, P2, P5, P7, P8 and P9 cannot vary with the severity gate. This is a design choice, not obviously a defect, but it means the "severity-aware" escalation exists in two patterns out of eight.
- **Three community-college exclusions test the sector three different ways in one expression:** `this.sector === '4' || this.sector === 4 || (this.sectorName && this.sectorName.toLowerCase().includes('community'))` (P4 line 2382, P5 2455, P7 2528, P8 2560). Since the constructor normalizes `this.sector` to a string (lines 391–395), the `=== 4` numeric comparison can never be true, and `sectorName` is derived from `this.sector` by `getSectorName()`, so the third clause is equivalent to the first. Two of the three tests are unreachable.
- **P4, P5, P7 and P8 return `reason` on their sector-exclusion path and `reasoning` on every other path** (e.g. lines 2384 vs 2396). `buildDetailedReport()` reads `pattern.reasoning || pattern.reason` (line 3201), so both surface, but the shape is inconsistent. P9 uses `reason` throughout.
- The eight patterns' maxima sum to 58 (6 + 6 + 12 + 7 + 8 + 8 + 7 + 4) before the archetypes are added; combined with the archetype maxima the 50-point cap in `calculateTotalAdjustmentPoints()` is a binding constraint, not a safety rail. See the assembly section.

Layer 2 total (`calculateTotalAdjustmentPoints`)

- **Method**: `calculateTotalAdjustmentPoints()` (lines 2949–2969). Called from two places: `applySeverityOverride()` line 2246, and `analyze()` Step 6 line 3321 (only when the override did not apply).
- **Points**: this is the function that produces the layer's point total.
- **What it tests**: nothing; it sums.
- **Inputs**: `this.archetypeFlags` (built by `detectArchetypeFlags()`, Step 2), `this.patternAddons` (Step 3), `this.accelerators` (Step 4). All three are initialized to `{}` in the constructor (lines 412–413, 444).
- **Gate logic**, transcribed in full:

```
  let total = 0;
  Object.values(this.archetypeFlags).forEach(flag => { if (flag.triggered) total += flag.points; });
  Object.values(this.patternAddons).forEach(pattern => { if (pattern.triggered) total += pattern.points; });
  Object.values(this.accelerators).forEach(accelerator => { if (accelerator.triggered) total +=
accelerator.points; });
  return Math.min(total, 50);
```

Term	Cap	Floor
Sum of triggered archetype flags	—	—
Sum of triggered pattern add-ons	—	—
Sum of triggered accelerators	—	—
Return	`Math.min(total, 50)` — **ceiling** at 50	**none**

Component maxima, for reference (all read from source):

Archetype flag	Points
`financial_distress`	12 (line 1285)
`enrollment_cliff`	10 (2941)
`operational_inefficiency`	3 (1389)
A1 4, A2 4, A3 1, A4 1, A5 3, A6 2, A7 3, A8 2, A9 2, A10 6, A12 6, A13 4, A14 8	42 total
R10 −8, R11 −4 (forced off), R12 −6	credits

Gross positive archetype maximum is 71 (12 + 10 + 3 + 46); the eight patterns add up to 58 more (P1 6, P2 6, P3 12, P4 7, P5 8, P7 8, P8 7, P9 4); accelerators add 0.

- **Sector differences**: none in this method. Sector restrictions live in the individual gates.
- **Data-gap behavior**: none. `flag.points` is read only when `flag.triggered` is truthy, and every check method returns both fields. If any of the three maps were still `{}` (i.e. the method were called out of sequence) the sum would silently omit that category — this is why the ordering comment at Step 1b (line 3299) exists.

- **2026-08 changes**: no dated comment inside the method. The relevant dated record is the A13 repricing quoted under the constants section, which set the only zero-point entry.
- **Notes**:
    - **There is no lower bound.** `Math.min(total, 50)` caps the top but not the bottom, so the method can return a negative number — up to −14 with R10 (−8) and R12 (−6) both firing and nothing positive. That negative propagates into `totalRiskPoints` in the orchestrator (see assembly), reducing the Layer 1 composite. R11 is forced off (line 2124, `const triggered = false;`) so its −4 cannot be reached.
    - **The 50-point cap is reached well before the framework's stated maximum exposure.** Positive archetype maxima total 67 and pattern maxima 58; even accounting for the mutually exclusive sector gates, a distressed private can co-fire `financial_distress` (12), `enrollment_cliff` (10), A1/A5/A6/A7/A8/A9/A10/A12/A14 and P1/P2/P4/P5/P7/P8/P9 far past 50. Above the cap the layer stops discriminating, and two institutions at 52 and at 95 gross points score identically.
    - **13 is counted as a triggered flag contributing 4 points (v49.7, 23 September 2026).**.
    - **The accelerator loop is permanently inert.** `detectAccelerators()` (2726–2750) sets three stubs, all `triggered: false, points: 0`:

> *// 2026-08-27: X1, X2 and X3 HAVE NEVER BEEN IMPLEMENTED. // // Confirmed empirically: the strings X1, X2 and X3 appear nowhere in the // 1,554 institutional reports of the 2026 baseline, across all sixteen // Carnegie x sector segments. These have been permanent stubs returning // triggered:false, points:0 since the accelerator layer was written. // // They are documented in "risk architecture part 2. archetypes etc" as // though they were live. THE DOCUMENTATION IS WRONG, not the code. Until // an accelerator is actually specified and built, nothing in the scoring // depends on this method and no report should describe accelerators as // part of the framework.*

    - **Double capping.** When the severity override applies, this method's already-capped return is passed through a second `Math.min(50, ...)` at line 2248. Harmless but redundant.
    - The method is called **twice** on the override path — once inside `applySeverityOverride()` and, if the override did not apply, once at Step 6. It is never called twice in a single run.

Critical Mass Escalation (`criticalMass` / `checkCriticalMassEscalation`)

- **Method**: `checkCriticalMassEscalation()` (lines 3005–3155), called from `analyze()` Step 1b (line 3302), before `detectArchetypeFlags()`.
- **Points**: **zero, directly.** It returns `levelBump: 1` when triggered, which nothing applies (see Notes). It is registered as archetype A13 by the IIFE at lines 2897–2912, whose award is `ARCHETYPE_BASE_POINTS.A13` = **4**.
- **What it tests**: whether the institution has four or more "Critical" analyzer ratings spread across three or more risk domains, with at least one in a financial domain. Header comment (2993–3004):

> *// CRITICAL MASS ESCALATION RULE // Strategy §2: Model-native level escalation // Detects when ≥3 analyzers are "Critical" across ≥2 domains. // Guardrail: requires at least one Financial/Revenue or Balance Sheet critical. // The Market/Demand requirement was removed — it blocked institutions in pure // financial extremis (Coe: 7 criticals across 4 domains, no demand critical). // Excludes: purely outcome metrics (grad rate, transfer out), // mission-linked metrics (Pell share). // Returns a levelBump signal for the orchestrator to apply.*

- **Inputs**: **direct `completedAnalyses` iteration** (line 3071) — no `getMetricValue` calls at all. Per record it reads:

Field	Use
`analysis.analyzerKey` (lowercased, whitespace → _)	domain lookup, primary
`analysis.measure`	domain lookup, fallback
`analysis.analysisResults.riskLevel` (lowercased)	Method 1 critical test
`analysis.analysisResults.riskPoints` (via `this.num()`)	Method 2 numerator
`analysis.analysisResults.maxRiskPoints` (via `this.num()`)	Method 2 denominator and weight filter

`measure` is the canonical name assigned by the orchestrator's `RA_CANONICAL` table (`analysis_orchestrator.js` lines 266–316); `maxRiskPoints` is normalized onto the record by the orchestrator before the push (line 231).

- **Gate logic**:

Eligible analyzers and their domains (lines 3010–3056), transcribed:

Domain	Keys
demand	enrollment, enrollment_trends, acceptanceyield, acceptance_yield, retention
revenue	tuitiondependency, tuition_dependency, nettuitionperfte, net_tuition_per_fte, appropriations, appropriationsdependency, appropriations_dependency, grantfunding, endowmentdependency, endowment_dependency
margin	operatingmargin, operating_margin, adjustedoperatingmargin, adjusted_operating_margin, structuralmargin, netincomeratio, net_income_ratio
balance_sheet	primaryreserve, primary_reserve, unrestrictednetassets, unrestricted_net_assets, cfi, capitalizationratio, capitalization_ratio, debtrevenue, debt_revenue, returnonnetassets, return_on_net_assets, restrictionprofile, restriction_profile, endowmentdrawsustainability, endowment_draw_sustainability, viability

Super-groups (3059–3060): `DEMAND_DOMAINS = {demand}` (declared, never read); `FINANCIAL_DOMAINS = {revenue, margin, balance_sheet}`.

Per-analyzer tests:

Test	Threshold	Direction	Line
Weight filter	skip if `maxRiskPoints !== null && maxRiskPoints < 4`	**floor** ≥ 4 to be considered	3087–3088
Method 1	`riskLevel === 'critical'` (lowercased)	exact	3098
Method 2	`riskPoints / maxRiskPoints >= 0.80`	**floor**	3103–3105

`CRITICAL_POINTS_RATIO = 0.80` (line 3066); `MIN_POINTS_FOR_ESCALATION = 4` (3087).

Decision rule (3124–3135), transcribed:

```
const CM_MIN_CRITICALS = 4;
const CM_MIN_DOMAINS   = 3;
...
const triggered = (
    criticalCount >= CM_MIN_CRITICALS &&
    domainCount >= CM_MIN_DOMAINS &&
    hasFinancialCritical
);
```

Condition	Threshold	Direction
Critical count	≥ 4	**floor**
Distinct domains	≥ 3	**floor**
At least one critical in `revenue`, `margin` or `balance_sheet`	boolean	—

Returned object (3136–3152): `{triggered, levelBump, criticalCount, domainCount, domainsHit, hasFinancialCritical, criticalKeys, details, reason}`.

- **Sector differences**: none in this method. Indirectly sector-dependent, because the orchestrator registers a different analyzer set for community colleges (`isExcludedForCC`, `analysis_orchestrator.js` line 110) and because framework-excluded measures are relabeled `'Informational'` with `maxRiskPoints: 0` (orchestrator lines 240–243), which the weight filter then skips.
- **Data-gap behavior**: an analyzer absent from `completedAnalyses` simply contributes nothing — it is indistinguishable from an analyzer that ran and was not critical. An analyzer with `riskLevel: 'unknown'` and `riskPoints: 0` is not critical by either method. An analyzer with `maxRiskPoints` null (which the orchestrator prevents) would skip the weight filter but be ineligible for Method 2, leaving only the explicit `riskLevel` test. Because the method makes no `getMetricValue` calls, none of this reaches `__gateLedger`, and `_annotateGateStatus()` cannot classify A13's evaluability — see the comment at lines 429–432:

> *// A11 and A13 are not check* methods — A13 is an IIFE reading this.criticalMass, // computed at Step 1b. They make no getMetricValue calls, so the ledger has no // record for them and _annotateGateStatus stamped them not_reached unconditionally.*

- **2026-08 changes**: the threshold raise, at lines 3117–3123:

> *// --- Decision rule --- // >=4 criticals across >=3 domains, with at least one financial-domain critical. // The demand-domain requirement was removed: it blocked institutions in pure // financial extremis (Coe College — 7 criticals across revenue, margin and // balance sheet, primary reserve 3.2% vs peer median 156.7% — failed only for // want of an enrollment critical). Breadth thresholds raised from 3/2 to 4/3 // to preserve selectivity: 3/2 without the guardrail fires for ~70% of a cohort.*

And the A13 registration, at lines 2898–2903:

> *// Critical Mass. Requires this.criticalMass to be populated before this map is // built — see Step 1b in the analyze() sequence. // 2026-08-27: RETIRED TO ZERO POINTS. A13 fires if and only if critical mass is // TRIGGERED, in every segment of the baseline, so as a scored archetype it was // charging 8 points for a flag the framework already acts on through // checkCriticalMassEscalation. It remains here as a reported diagnostic.*

- **Notes**:
  - **The header comment states the old thresholds.** Lines 2997–2998 say "Detects when ≥3 analyzers are 'Critical' across ≥2 domains"; the implementation (3124–3125) uses 4 and 3. The dated comment 120 lines below records the change; the header was not updated.
  - **The critical mass signal is not applied anywhere.** The A13 award is 4, and `levelBump` is read by exactly one consumer in the repository — `report_generator.js` line 410, which prints it as `TRIGGERED (bump +1, NOT APPLIED)`. The analyzer's own report says the same (line 3216). So the comment at line 3003, "Returns a levelBump signal for the orchestrator to apply", describes behavior that does not exist: the orchestrator does not read `criticalMass` at all. **Critical mass currently changes nothing about any institution's score or band.** The A13 repricing comment's claim that "Critical mass continues to drive escalation through checkCriticalMassEscalation" is therefore not borne out by the code.
  - **`DEMAND_DOMAINS` is declared and never read** (line 3059) — a remnant of the removed market/demand requirement.
  - **Two eligible-analyzer entries can never match.** `cfi` and `netincomeratio` / `net_income_ratio` name analyzers that are not registered in the orchestrator's `allAnalyzers` list (`analysis_orchestrator.js` 65–107), even though `unified_cfi_analyzer.js` and `unified_net_income_ratio_analyzer.js` exist on disk. No institution can produce those keys.
  - **The underscore variants in `ELIGIBLE_ANALYZERS` are unreachable in practice.** The lookup tries `ELIGIBLE_ANALYZERS[key]` first, where `key` is the orchestrator's compact `analyzerKey` (`enrollment`, `tuitiondependency`, `appropriations`, …), and every compact form is present in the map. The `measure`-based fallback (`enrollment_trends`, `tuition_dependency`, `appropriations_dependency`, …) is therefore never reached. Harmless duplication, but it doubles the apparent size of the eligible set.
  - **The `margin` domain is labeled "Operating performance & cost structure" but contains no cost-structure analyzer.** `instructionalexpenditure`, `institutionalexpenditure`, `studentservicesexpenditure`, `salariesbenefits`, `studentfacingexpenditure` and `sfr` are all absent from the eligible set. The header comment lists only grad rate, transfer out and Pell as intended exclusions; the cost-structure omission is undocumented. `netrevenueratio`, `endowment`, `programdistribution`, `programsustainability`, `pell`, `gradrate`, `transferout` and `federalpolicy` are likewise absent.
  - **Method 2 makes a `>= 0.80` points ratio equivalent to an explicit Critical rating**, without reference to whether the producing analyzer's own bands put 80% in its Critical range. An analyzer whose Critical band starts at 90% of maximum will be counted critical here at 80%.
  - `viability` appears in the `balance_sheet` block at line 3042, physically adjacent to `primary_reserve`; listed above in the same domain for completeness.
  - The `reason` string's final fallback, `'No financial-domain critical (guardrail)'` (line 3151), is reached only when the count and domain thresholds are both met, so it is correct — but it is presented as an `else` on a chain that has already excluded the other two causes, and a reader cannot tell from the string alone which of the three tests was closest.

Layer 1 / Layer 2 assembly (`analyze()` + `analysis_orchestrator.js`)

- **Method**: `RiskAdjustmentAnalyzer.analyze()` (lines 3279–3424); the join itself is in `analysis_orchestrator.js` lines 212–630.
- **Points**: Layer 2 contributes `this.totalAdjustmentPoints` (range −14 to +50) to the composite numerator.

- **What it tests**: n/a — this is the assembly.
- **Inputs**: the constructor takes (`completedAnalyses, sector, baseTotalPoints, frameworkMaxPoints = 177, debug = false, institution = null, peerCriteria = null`) (line 371). The orchestrator supplies (lines 467–475): `window.lastCompletedAnalyses`, `this.sector`, `baseRiskScore` (the plain sum of every analyzer's `riskPoints`, line 430), `frameworkMaxPoints`, `true`, `institution`, `peerCriteria`.
- **Gate logic** — the execution sequence inside `analyze()`, transcribed in order:

Step	Line	Action
1	3296	`const severityCondition = this.checkSeverityCondition();`
1b	3302	`this.criticalMass = this.checkCriticalMassEscalation();` — **must** precede Step 2
2	3305	`this.detectArchetypeFlags();` (builds 19 keys including A13 from `this.criticalMass`)
3	3308	`this.calculatePatternAddons(severityCondition);`
4	3311	`this.detectAccelerators();` (three permanent stubs)
5	3317	`const overrideResult = this.applySeverityOverride(severityCondition);`
6	3320–3322	`if (!overrideResult.applied) { this.totalAdjustmentPoints = this.calculateTotalAdjustmentPoints(); }`
7	3325	`this.generateDebugOutput();`
8	3328	`buildDetailedReport(severityCondition, overrideResult)`

The override arithmetic, quoted from lines 2246–2249 — this is the expression named in the task, and it is the only place the two paths meet:

```
const archetypePatternTotal = this.calculateTotalAdjustmentPoints();

this.totalAdjustmentPoints = Math.min(50, Math.max(severityPoints, archetypePatternTotal));
this.overrideLiftPoints = Math.max(0, this.totalAdjustmentPoints - archetypePatternTotal);
```

So the layer total is:

Case	Total
Severity gate unarmed, or armed with < 3 triggers	`min(50, Σ triggered archetypes + Σ triggered patterns + 0)`
Severity gate armed with ≥ 3 triggers	`min(50, max(severityPoints, Σ as above))` where `severityPoints ∈ {12, 16, 20}`

Published on `analysisResults` (3357–3412): `riskPoints: this.totalAdjustmentPoints`, `maxRiskPoints: 50`, `riskLevel: this.calculateAdjustmentRiskLevel()`, plus `criticalMass`, `severityOverride`, `allArchetypes`, `allPatterns`, `triggeredArchetypes`, `triggeredPatterns`, `pointsByArchetype`, `pointsByPattern`, `r11Diagnostic`.

The orchestrator's join (lines 477–485):

```
const riskAdjustmentResult = await riskAdjustmentAnalyzer.analyze();
results.individual_results['riskadjustment'] = { result: riskAdjustmentResult };
if (riskAdjustmentResult && riskAdjustmentResult.success && riskAdjustmentResult.analysisResults) {
    const adjustmentPoints = riskAdjustmentResult.analysisResults.riskPoints || 0;
    totalRiskPoints += adjustmentPoints;
}
```

`totalRiskPoints` at that point is the sum of every Layer 1 analyzer's clamped `riskPoints` (accumulated at line 252). The denominator is built **without** the risk-adjustment record: every denominator loop begins `if (analyzerKey === 'riskadjustment') continue;` (lines 496, 592) and the `raFullMax` / `frameworkFullMax` reductions filter it out explicitly (lines 435, 521). So:

Quantity	Definition	Lines
Numerator	`Σ Layer 1 riskPoints + Layer 2 total`	252, 484
Denominator	`Σ maxRiskPoints of Layer 1 analyzers that succeeded, are not isDataUnavailable, and are not not_applicable``	490–512
Denominator floor	`Math.floor(frameworkFullMax * 0.60)`, applied if the denominator is below it; sets `denominatorFloored = true`	537–548
`riskPercentage`	`(totalRiskPoints / adjustedMaxPoints) * 100`	614

Quantity	Definition	Lines
Band	`riskArchitecture.determineRiskLevel(riskPercentage)`, unless `insufficientData` (< 50% of analyzers with data) forces `'Insufficient Data'`	617–623

- **Sector differences**: the orchestrator excludes a set of analyzers for community colleges (`isExcludedForCC`, line 110) and registers sector-specific analyzers (appropriations for publics; endowment, restriction profile, draw sustainability, endowment dependency, research grants for others) at lines 120–166. Layer 2's own sector logic is per-gate, documented above.
- **Data-gap behavior**: on any thrown exception `analyze()` returns `{success: false, analysisResults: {riskPoints: 0, maxRiskPoints: 50, riskLevel: 'Error'}}` (3416–3423). The orchestrator's `if (... .success ...)` guard then adds nothing, so a Layer 2 crash silently yields a Layer-1-only composite with no marker in `results.summary` other than `severity_override` falling back to `{applied:false, triggerCount:0, unavailable:true}` (line 630).
- **2026-08 changes**: the Step 1b ordering constraint (lines 3299–3301):

> *// Step 1b: Critical Mass Escalation — MUST run before detectArchetypeFlags(), // which registers it as archetype A13. Reads analyzer results only; no // dependency on archetypes, patterns or accelerators.*

And the `rawData` carry-through in the orchestrator (lines 375–384), which is what made P9 evaluable at all:

> *// 2026-08-28 (v49.1): rawData is now carried onto the record. // Its absence was the single most frequent defect in this codebase: a // layer-2 gate reading `someAnalysis.rawData.x` could never succeed, because // this push built the record from analysisResults alone. It is what disabled // calculateP9_EndowmentDrainRisk entirely — its third gate reads // rawData.perFTEEndowment.currentValue and rawData.peerContext.peerMedian, // both of which the endowment analyzer genuinely publishes, just not here. // 158 privates clear P9's first two gates and were then rejected by a test // that could not read its own input; P9 fired 0 times in 1,552 institutions.*

- **Notes**:
  - **Layer 2 enters the numerator but not the denominator.** Up to 50 points are added to `totalRiskPoints` while `adjustedMaxPoints` counts only Layer 1 maxima. For an institution with a denominator around 130, a Layer 2 total of 50 is worth ~38 percentage points of composite on its own, and the composite is not bounded by 100%. This is what the code does; no comment addresses it.
  - **A negative Layer 2 total reduces the composite and can make it negative.** `|| 0` at line 483 does not intercept a negative (it is truthy), so R10 + R12 firing at a low-scoring institution yields `totalRiskPoints` below zero and a negative `riskPercentage`. There is no clamp on `riskPercentage` before it reaches `determineRiskLevel()`.
  - **`baseTotalPoints` is passed in and never used for scoring.** `getOriginalRiskPoints()` (2774–2776) is read only by `buildDetailedReport()` for the three summary lines at 3185–3188. The layer's own arithmetic never references the Layer 1 total, so no gate and no cap is proportional to it.
  - **`frameworkMaxPoints` is likewise unused for scoring.** `getFrameworkMaxPoints()` (2770–2772) has no callers inside the class; the orchestrator computes and passes it (lines 434–459) at some expense, including a 60% floor deliberately kept in sync with the denominator floor. The comment justifying that synchronization — `// MUST use the same availability rule and denominator floor as adjustedMaxPoints below, // or the severity floor lifts to 49% of one base and the score is reported against another.` — describes a "severity floor" expressed as a percentage of the base, which the current `applySeverityOverride()` does not implement (it is points-based, and `floorTarget` is always `null`). The parameter and its careful computation are vestigial.
  - **The constructor overrides the `debug` parameter.** Line 404 is `this.debug = true;`, discarding the argument. Every run therefore emits the full console trace, including a `console.table` of eight metric lookups and two `console.log` array dumps per institution at lines 3333–3352, plus the `=== AVAILABLE ANALYZER KEYS ===` loop at 3282–3290. The orchestrator has a `__ratQuiet()` mechanism for ranking runs; this class does not honor it.
  - **`this.getInstitutionName()`** (2778–2784) reads `this.completedAnalyses[0]?.institution?.name`, but the orchestrator's snapshot push (lines 369–386) does not include an `institution` key. Debug output therefore always reports `'Unknown Institution'`.
  - **`buildDetailedReport()`'s headline claims critical mass is included in the points** (line 3258): `Risk adjustment: N points (includes Critical Mass Escalation as A13)`. A13 is worth 4, so the parenthetical is false whenever it appears.

- `probeA90nce()` (451–468) is a diagnostic that is never called.

5. Shared machinery and known defects

Behavior shared across gates, and defects found during this reading and not yet fixed. Every item is reported as found; none has been reconciled in the prose. Items already recorded in an entry above are not repeated.

Shared input resolution (`getMetricValue` / `_getMetricValueRaw`)

Source: assembly

Not a scoring element, but every gate below reads through it and several of the findings in this document are properties of the resolver rather than of the gate.

- **Method**: `getMetricValue(analyzerKey, metricKey)` (lines 724–761), delegating to `_getMetricValueRaw()` (lines 856–1126).
- **Points**: none. Instrumentation only.
- **What it tests**: resolves a named field on a named analyzer's `analysisResults`, through an ordered chain of explicit routes.
- **Gate logic**: resolution order, in the order the source tries them:

Step	Lines	Route
1	858–867	Find analyzer: exact normalized `analyzerKey` match against `[analyzerKey, ...analyzerAliases[analyzerKey]]`; failing that, substring match in either direction
2/3	878–960	`keyMappings` table; dotted mapped keys (e.g. `shortTerm.totalChangePercent`) resolved as a deep path first, then as a flat key
4	962–965	Exact `results[metricKey]`
4b	967–984	Declared `metricAliases[metricKey]`, flat keys only, at top level
5	986–1038	**Fail closed.** `genericVariations` is `[]`. A refusal is counted into `window.__RA_FALLBACK_KEYS__` but no value is returned
6	1041–1058	`perKeyVariations`, defined only for `instructional_percentage`, `administrative_percentage`, `student_services_percentage`
7	1060–1121	Container sweep over `results.data/.metrics/.values/.meta/.shortTerm/.summary`; peer containers (`analysis.peerGroup.averages`, `analysis.peerGroup`, `analysis.averages`) are searched **only** when `metricKey` or `mappedKey` begins with `peer`
8	1123–1125	Return `null`

- **Data-gap behavior**: returns `null` at every unresolved path. The `null` is recorded against the calling gate in `this.__gateLedger` (attributed by regex on the call stack, line 783) and surfaces as gate `status` `unavailable` / `not_applicable` / `degraded` via `_annotateGateStatus()` (lines 792–854). Status is non-scoring.
- **2026-08 changes**: the step 5 removal is the single largest behavioral change in the layer.

> *// 5) FAIL CLOSED. (2026-08-30, v49.2) // // This step used to be a generic substitution: // // const genericVariations = ['current','currentValue','value','percentage', // 'ratio','change','trend','currentPercentage', // 'currentPercent','percent']; // for (const k of genericVariations) if (results[k] !== undefined) return results[k]; // // When a gate asked for a field its producer does not publish, the resolver // returned whatever generic field happened to exist on that analyzer. Not null — // a number. Of a different quantity, in a different unit, which the gate then // compared against its threshold as though it were the thing requested.*

And on the enrollment window, which governs three of the elements in this document:

> *// 2026-08-28 (v49.1): THESE THREE POINTED AT THE FOURTEEN-YEAR CHANGE. // totalChangePercent is ((total[2024] - total[2010]) / total[2010]) * 100 — // the 2010->2024 change. Every gate requesting a "5 year" figure therefore // tested a fourteen-year decline while its comments, its published reason // strings and the architecture documentation all said five years. Three of // the heaviest elements in layer 2 read it: // checkEnrollmentCliff 10 pts, fired 593 times (38.2% of the pool) // checkSeverityCondition arms the override for 325 institutions // checkFinancialDistress 12 pts, fired 345 times // A 25% decline over five years is an emergency; over fourteen it is a slow // contraction a great many institutions have had. The gates were far easier // to open than anyone believed, which is why enrollment cliff was the second // most frequent element in the whole layer. // // They now resolve to the genuine short-term series, which the enrollment // analyzer publishes as shortTerm.totalChangePercent and which is itself now // a five-year window (2019-2024, see unified_enrollment_analyzer.js v49.1). // Expect all three gates to fire substantially less often.*

So `'5_year_change'`, `'recent_change'` and `'2018_2023_change'` all map to `shortTerm.totalChangePercent` (lines 898–900), and `'long_term_change'` maps to `totalChangePercent` (line 902).

- **Notes**:
  - **`metricAliases` contains three duplicate keys, and the duplicates silently discard the 2026-08-30 repair.** `instructional_percentage` is declared at line 554 as `['instructional_percentage', 'currentRate', 'currentPercentage']` and again at line 608 as `['instructional_percentage','instructional_spend_pct']`. In a JavaScript object literal the later declaration wins, so the `currentRate` alias is discarded. The same happens for `administrative_percentage` (557 vs 611) and `student_services_percentage` (560 vs 614). The comment introducing the discarded declarations reads:

> *// Likewise the percent-of-spend names: the expenditure analyzers publish the // figure as `currentRate` (and the orchestrator copies it onto the record for // exactly this reason). perKeyVariations sends these to `currentPercentage`, // which none of the three analyzers publishes.*

The repair therefore did not take effect. Consequence is documented under `checkOperationalEfficiency` — the affected lookups are third-choice fallbacks whose first link resolves in practice, so no live scoring change is visible.

  - `analyzerAliases` also declares `grantfunding` twice (lines 477 and 495) with identical contents. Harmless.
  - `_callingGate()` attributes a lookup by regex-matching the JavaScript stack trace for `/(check[A-Z][A-Za-z0-9_]*|calculateP\d+_[A-Za-z0-9_]*)/`. This is engine-dependent and is wrapped in try/catch; the comment at line 758 states the constraint — `/* instrumentation must never affect scoring */`.

Module scoring constants (`ARCHETYPES`, `ARCHETYPE_BASE_POINTS`, `P2_BASE_POINTS`, `APPLY_ARCHETYPE_POINTS`, `SEGMENT_WEIGHTS`)

Source: assembly

- **Method**: module scope, lines 137–225; consumer helper `scoreArchetypes()` at lines 358–368.
- **Points**: `ARCHETYPE_BASE_POINTS` is the point value of every archetype flag except the three legacy flags (`financial_distress`, `enrollment_cliff`, `operational_inefficiency`), which hard-code their own awards. `P2_BASE_POINTS` is the P2 pattern's base award.
- **What it tests**: nothing; these are lookup tables read by the individual `checkAn_*` methods in the form `ARCHETYPE_BASE_POINTS?.Xn ?? <literal>`.
- **Inputs**: none.
- **Gate logic**: the values, transcribed from lines 179–196 and 225:

Code	Name (ARCHETYPES, line 137–162)	Points	Fallback literal at call site
A1	High Vulnerability – Small Scale, High Dependency	4	`?? 4` (line 1554)
A2	Leverage-Led Growth Bet	4	`?? 4` (1580)
A3	Academic Sprawl Trap	1	`?? 3` (1608)
A4	Monoculture Dependency	1	`?? 3` (1641)
A5	High-Cost Architecture, Stalling Demand	3	`?? 3` (1696)
A6	Brand-Weakening Channel	2	`?? 2` (1460)
A7	High-Pell "Mission Pressure Cooker"	3	`?? 3` (1762)
A8	Brand-Strong but Margin-Thin	2	`?? 2` (1783)
A9	The Squeezed Middle	2	`?? 2` (1852)
A10	Endowment-Masked Operating Deficit	6	`?? 6` (1879)
A12	Broad Balance-Sheet Failure	6	`?? 6` (1942)
A13	Critical Mass Escalation	4	`?? 0` (2907); `?? 8` at report line 3216
A14	Demand Collapse with Intact Reserves	8	`?? 8` (1987)
R10	Resilient by Scale and Diversification	**–8**	`?? -2` (2072)
R11	Research-Strong Anchors	**–4**	`?? -2` (2126)

Code	Name (ARCHETYPES, line 137–162)	Points	Fallback literal at call site
R12	Brand-Strong Anchors	**−6**	`?? -2` (2146)
P2	(pattern, not an archetype)	`P2_BASE_POINTS = 4`	— (line 2342)

There is **no A11 entry** in either table. The severity override labels itself "A11 — Severity Pattern" (line 2237) and prices itself from a hard-coded ternary at line 2242.

`scoreArchetypes(archetypes, segmentKey)` (358–368) is the only reader of `SEGMENT_WEIGHTS`:

```
  const w = (weights[f.code] ?? 1);
  pts += (base[f.code] || 0) * w;
```

- **Sector differences**: none. `SEGMENT_WEIGHTS` is the declared mechanism for segment-aware weighting and is `Object.create(null)` — empty (line 175).
- **Data-gap behavior**: n/a.
- **2026-08 changes**: three dated blocks, quoted verbatim.

On the removed lightweight archetypes (lines 154–160):

> *// 2026-08-28 (v49.1): LA1 'Aid Arms-Race' and LA2 'Small-Scale Fragility' // REMOVED. They were named here and priced at 2 points each in // ARCHETYPE_BASE_POINTS, but were absent from detectArchetypeFlags() and so // never evaluated — the framework documented 4 points of risk it never // charged. Their only implementation lived in detectArchetypes(), deleted // 2026-07-27. Restore them by writing checkLA1_/checkLA2_ methods and // registering them, not by re-adding the names.*

On the A13 and resilience repricing (lines 181–195):

> *// 2026-08-27 CHANGES, all measured against the v48.7 baseline (1,456 scored institutions): // // A13 8 -> 4. A13 fires exactly as often as the critical-mass computation is TRIGGERED — // 28/28, 16/16, 51/51 and so on in every one of the sixteen segments. It is a restatement // of that flag under another name, and it carried the highest base-score correlation in // the framework (r = +0.60), i.e. it charged 8 points for risk the measures had already // priced. Critical mass continues to drive escalation through checkCriticalMassEscalation; // A13 stays in the map at four points so the diagnostic remains visible in the export and // nothing downstream that reads archetypeFlags.A13 breaks. // // R10 -3 -> -8, R12 -2 -> -6. Resilience was priced against a layer-2 that averages +22 // points. 201 institutions held a credit and NINE had a band improved by it. Widening // eligibility (see __hasResilientScale) without repricing would have made a decorative // feature available to more people.*

On `P2_BASE_POINTS` (lines 198–224):

> *// 2026-08-29: P2's award, isolated so it is one line to change. It was 8 rising to 10; // it is now 4 rising to 6, which places it at the bottom of the pattern range alongside // P1. The placement is deliberate and the reasoning is on the record. // // P2 is the only pattern in the layer whose subjects are not currently distressed. Flat // enrollment is a powerful present-tense protective signal: across 902 private // institutions, those holding enrollment within +/-5% average 32.2% composite against // 44.5% for the rest. Requiring a deepening discount pulls that group back toward the // average but cannot push it above - every variant tested still lands 3 to 5 points // BELOW the population. So this identifies institutions whose present stability is // purchased, and the risk it names has not yet arrived. // // That is a legitimate thing for a risk framework to score - a tool that waits until the // margin has gone is not forecasting anything. But it earned 8 to 10 points, the second // most generous award in the layer, on the weakest present-tense evidence in it: // // pattern award discrimination (mean gap) // P3_AppropsDep_EnrollDecl 10-12 +17.2 // P2 (old) 8-10 -2.2 // P5_UNA_Risk 6-8 +21.3 // P7_BrandPositioning 6-8 +14.8 // P4_NetTuitionRisk 5-7 +11.1 // P8_MarketPosition 5-7 +15.6 // P1_SienaPattern 4-6 +13.3 // // 4 keeps the signal without letting a forecast outweigh measured condition. Confirm the // new group's discrimination on the next full re-run before treating this as settled.*

- **Notes**:
  - **`ARCHETYPES` is declared and never read.** A repository-wide grep for the identifier returns only its own declaration at line 137. Nothing in the analyzer, the orchestrator, the report generator or the HTML tool consumes the display names; every `checkAn_*` method builds its own reason string. The names in the table above are therefore documentation-only.
  - **`APPLY_ARCHETYPE_POINTS` is declared, assigned and never read.** Declared `let APPLY_ARCHETYPE_POINTS = true;` at line 166 with the comment `// Default OFF: points are preview-only unless explicitly enabled.` — which contradicts the initializer on the next line. `setApplyArchetypePoints()`

(169–171) is exported to no one and called by no one. Archetype points are unconditionally applied by `calculateTotalAdjustmentPoints()`. The "preview only until APPLY_ARCHETYPE_POINTS=true" comment at line 178 describes behavior that does not exist.

- **`scoreArchetypes()` has no callers**, so `SEGMENT_WEIGHTS` is unreachable. Its signature expects `{flags, resilience}` arrays, a shape no live code produces — `detectArchetypeFlags()` builds a flat keyed map. The segment-weighting mechanism described in the comment at line 173–174 (`e.g., { "private:21": { A4: 1.0, A3: 0.7 } }`) is not wired.
- **Six `??` fallback literals disagree with the table they fall back from**: A3 (`?? 3` vs 1), A4 (`?? 3` vs 1), R10 (`?? -2` vs −8), R11 (`?? -2` vs −4), R12 (`?? -2` vs −6), and the report line 3216 (`?? 8` vs 0). All are unreachable — `ARCHETYPE_BASE_POINTS` defines every key — but they are stale records of pre-2026-08-27 prices.
- A13 is registered as a flag with `triggered: true` and `points: 4`. It therefore appears in `triggeredArchetypes` and in `pointsByArchetype` as `A13: 4`, and it inflates the "Archetype Flags Triggered: N" count in the report without contributing points.
- The `ARCHETYPES` string values contain mojibake from a prior encoding round-trip (`'High Vulnerability â€“ Small Scale, High Dependency'`, `'High-Pell â€œMission Pressure Cookerâ€'`). Never displayed, because the constant is never read.

Shared helpers

Source: archetypes_A1_A6

These are read by the gates below; their actual return contracts matter to every threshold.

`num(v)` — class field, lines 650–656

```
650|    num = v => {
651|        if (v === null || v === undefined) return null;
652|        if (typeof v === 'number' && Number.isFinite(v)) return v;
653|        const s = String(v).replace(/,/g, '').replace(/[^0-9.\-eE]/g, ''); // drop % and symbols
654|        const n = parseFloat(s);
655|        return Number.isFinite(n) ? n : null;
656|    };
```

Returns a finite `Number` or `null`. Strings are stripped of `,`, `%` and other symbols and parsed. Never throws. **`null` is the only miss signal** — `0` is a legitimate value.

`pct(v)` — class field, lines 658–662

```
658|    pct = v => {                    // returns % on [0..100] or null
659|        v = this.num(v);
660|        if (v === null) return null;
661|        return (v <= 1 ? v * 100 : v); // 0.29 -> 29; 29 -> 29
662|    };
```

Returns a percent or `null`. **Any input `<= 1` is multiplied by 100**, including a genuine 0.5 percent and including negatives (`-0.4` → `-40`). Not range-clamped.

`toPct(v)` — module function, lines 109–122

Same normalization as `pct()` but returns `undefined` (not `null`) on a miss, and for strings uses `(/%/.test(s) || n >= 1) ? n : n * 100`. Used only by A1.

`getMetricValue(analyzerKey, metricKey)` — lines 724–760, raw resolver at 856–1126

Instrumented wrapper over `_getMetricValueRaw`. Resolution order: 1. analyzer lookup by `analyzerAliases` + normalized exact, then normalized substring match; 2. `keyMappings` table (supports one dotted path, e.g. `'5_year_change'` → `'shortTerm.totalChangePercent'`); 3. exact `metricKey` on `analysisResults`; 4. **(2026-08-30)** declared `metricAliases` at top level; 5. **FAIL CLOSED** — the generic-substitution list is now empty; 6. per-key variations (percent-of-spend metrics only); 7. deep container sweep over `results.data|metrics|values|meta|shortTerm|summary` (peer containers only when the key or its mapping starts with `peer`); 8. `return null`.

> ***2026-08-30 (v49.2)*** *(lines 992–1017, abridged here; quoted in full under A5) "A field that cannot be resolved by an EXPLICIT route — the keyMappings table, the exact published name, a declared alias, or a declared per-key variation — now returns null. A gate that cannot read its input must know it cannot read its input."*

`getEnroll5()` — lines 1128–1136

```
1128|    getEnroll5() {
1129|        let v = this.num(this.getMetricValue('enrollment','5_year_change'));
1130|        if (v !== null) return v;
1131|        // fallback to any "recent change" field the pipeline provides
1132|        v = this.num(this.getMetricValue('enrollment','recent_change')
1133|                  ?? this.getMetricValue('enrollment','2018_2023_change'));
1134|        return v;
1135|    }
```

Returns: the five-year total-enrollment change in percent, or `null`. `'5_year_change'`, `'recent_change'` and `'2018_2023_change'` are **all three mapped to the same field** by `keyMappings` (lines 897–899):

```
897|    '5_year_change': 'shortTerm.totalChangePercent',
898|    'recent_change': 'shortTerm.totalChangePercent',
899|    '2018_2023_change': 'shortTerm.totalChangePercent',
```

The producer (`unified_enrollment_analyzer.js` line 401) computes `((total[2024] - total[y]) / total[y]) * 100` for the earliest available y in the recent window. **The fallback chain on lines 1132–1133 is therefore a no-op**: if link 1 returns `null`, links 2 and 3 resolve through the identical mapping and also return `null`.

`getEnrollmentDeclineWorst()` — lines 2792–2833

Returns: the more negative of `totalChangePercent` (2010→2024) and `shortTerm.totalChangePercent` (five-year), in percent, or `null` when neither is finite.

> ***2026-08-29 (v49.1)*** *(lines 2818–2823): "the two share metrics are no longer candidates here. This function returns "the worst enrollment decline", and a minimum is only meaningful across commensurable quantities. Mixing percent-of-volume with points-of-mix meant an institution whose graduate share fell 30 points while total enrollment GREW reported a worst decline of -30%. The shares are complements, so one of them is negative for almost every institution."*

Not called by any of A1–A6. It is listed in the task scope but is used only by A9, P1, P7, P8 and the debug export. Documented here because it is easily confused with `getEnroll5()`: `getEnroll5()` is the five-year figure alone; `getEnrollmentDeclineWorst()` is a minimum over a five-year and a fourteen-year figure.

`getNtrNorm()` — lines 1139–1187

Returns: `net_tuition_per_fte_normalized` if the net-tuition analyzer publishes it finite; otherwise `currentValue / peerAverage` (a ratio ≈ 1.0 at peer average); otherwise `null`. **Not called by any of A1–A6.** A5 reads `percentChange` directly instead (see A5).

`resolveUnaPerFTE()` — lines 701–714

```
699| // Resolve UNA per student. The UNA analyzer publishes only a total, so a direct
700| // 'una_per_fte' lookup falls through getMetricValue step 5 and silently returns
701| // currentValue (the total in dollars). Always derive from total / enrollment.
```

Reads `analysisResults.una_per_fte` on the record whose normalized `analyzerKey` is exactly `unrestrictednetassets`; if absent, `total ?? currentNetAssets ?? currentValue` divided by `getMetricValue('enrollmenttrends','currentTotal') ?? …'total'`. Returns `null` if either side is missing or enrollment `<= 0`. The producer does publish `una_per_fte` (`unrestricted_net_assets_analyzer.js` line 180), so the division branch is a rarely-taken fallback. **The header comment is stale**: it says the analyzer "publishes only a total".

`makeGetter(sources)` — lines 270–297, with `resolveAnalyzerKey` (236–248) and `scanNumericDeep` (250–266)

Used only by A1. Searches `[record, record.components, record.analysisResults]` for each listed dot-path in order; if all miss and `opts.regex` is supplied, it scans shallow keys — **and one level of nested object keys** — of each bag and returns the first `number`/`string` value whose *key name* matches the regex. This is a fail-open path; see A1 **Notes**.

`pickAnalysis(analyses, names)` — lines 8–36

Used only by A6. Normalizes names to `[a-z0-9]` and matches on `analyzerKey|key|id|name` or `title|headline`, using bidirectional substring containment.

> *(lines 20–27) "Empty candidates must never match. Snapshot entries built by analysis_orchestrator.js carry no top-level `title` or `headline` (the headline is nested in detailedReport), so `title` is '' for every entry — and needle.includes('') is true. That made list.find() return element [0] for ANY unresolved key: A6 was reading `enrollment` riskPoints and reporting them as acceptance_yield."*

Analyzer keys and framework maxima

`analyzerKey` is derived in `analysis_orchestrator.js` line 219: `analyzer.constructor.name.toLowerCase().replace('universal','').replace('analyzer','')`. So `UniversalDebtRevenueAnalyzer → 'debtrevenue'`, `UniversalProgramDistributionAnalyzer → 'programdistribution'`, `UniversalEnrollmentAnalyzer → 'enrollment'`, etc. All strict-equality lookups in A2–A5 match live keys.

`maxRiskPoints` is overwritten by the orchestrator (lines 224–232) from `framework.json`; a framework max of 0 forces `riskPoints = 0` and `riskLevel = 'Informational'` (lines 240–250). Relevant maxima:

measure	public (1)	private NP (2)	private FP (3)	CC (4)
AcceptanceYield	10	8	8	**0** (analyzer also excluded for CC)
Enrollment	15	18	10	15
DebtRevenue	8	10	10	10
ProgramDistribution	6	6	6	6
AdjustedOperatingMargin	8	8	8	8
OperatingMargin	**0**	**0**	**0**	**0**
PrimaryReserve	10	10	10	10
Viability	8	8	8	8
TuitionDependency	8	18	18	8
Appropriations	15	0	—	15

`this.sector` is normalized in the constructor (lines 391–395) to the strings `'1'`–`'4'`, falling back to `String(sector || '1')`.

Cross-cutting notes

Source: archetypes_A1_A6

1. **`ARCHETYPE_BASE_POINTS` inline defaults disagree with the table for A3 and A4.** Both use `?? 3` against declared values of 1. The `??` branch is unreachable while the constant object is intact, so the awarded value is 1 in both cases — but the two numbers are a standing contradiction in the file, and a reader checking the gate rather than the table would read A3 and A4 as 3-point archetypes.

2. **`APPLY_ARCHETYPE_POINTS`, `setApplyArchetypePoints()`, `SEGMENT_WEIGHTS` and `scoreArchetypes()` are declared and never read.** Lines 164–175 and 359–368. The comment at line 178 says the base points are *"preview only until APPLY_ARCHETYPE_POINTS=true"*, and the one at line 165 says *"Default OFF: points are preview-only unless explicitly enabled"* — but the flag is initialized to `true` (line 166) and, more to the point, nothing consults it. `calculateTotalAdjustmentPoints()` (lines 2949–2969) sums `flag.points` unconditionally. `SEGMENT_WEIGHTS` is an empty null-prototype object read only by the dead `scoreArchetypes()`, so **no segment weighting is applied anywhere on the live path**, and the comment at line 174 ("You can override per segment later") describes a capability that is not wired in.

3. **Two resolvers with opposite fail behavior.** `getMetricValue()` was made fail-closed on 2026-08-30. `makeGetter()` — used only by A1 — still has a regex key-name scan that can substitute a different quantity when the explicit paths miss. A1 is consequently the one gate among these six that can fire on a substituted number.

4. **Absolute risk-point bars against sector-varying maxima** remain in A1 (`prRisk >= 6` of 10; `viRisk >= 6` of 8), A2 (`debtRiskRP >= 6` of 8 public / 10 private) and A5 (`aomRP >= 6|7` of 8). A6 is the only one of the six expressed as a share of

maximum, and its rebuild comment identifies exactly this asymmetry as the defect it was fixing. The other gates were not changed to match.

5. **Fields read that producers do not publish**, by gate:

- A1: `analysisResults.tuition_percentage`, `analysisResults.tuition_dependency`, `una_total`, `unrestricted_net_assets_total`, `components.*` (no producer emits a `components` container) — all inert, earlier links resolve.
- A4: `topProgramShare`, `largestProgramShare`, `maxProgramShare`, `concentration` — **not inert**; they are the whole of `sharePath`, which is therefore dead.
- A6: analyzer name `brand_pipeline` — inert.
- A5: the two links removed on 2026-08-30 were of this kind; none remain.

6. **Unreachable branches** in scope: A4's `sharePath` (no producer field); A5's two `omRP` legs (framework maximum 0 in every sector forces `riskPoints` to 0); the `?? 3` defaults in A3 and A4.

Scope note: A11 and A13 — confirmed from the code

Source: archetypes_A7_A14

A11 does not exist as an archetype gate. Confirmed by exhaustive grep: there is no `checkA11_*` method anywhere in `risk_adjustment_analyzer.js`; `A11` does not appear in `ARCHETYPES` (lines 137–162) and does not appear in `ARCHETYPE_BASE_POINTS` (lines 179–196); it is not a key in the `archetypeFlags` map built by `detectArchetypeFlags()` (lines 2879–2917). The string "A11" occurs in exactly three places:

1. Line 432, in the diagnostic map `__gateFnByKey`, as `A11: '__computedOutsideLedger'` — a key for a gate that is never built. 2. Line 805, in a comment inside `_annotateGateStatus`. 3. Line 2253, inside the **severity override** (not the archetype layer), where the override's reason string is formatted as `` `Severity pattern (A11): ...` ``. The override prices concentrated distress at 12/16/20 points (line 2242) on a `max`-not-`sum` basis against the archetype+pattern total (lines 2246–2248). "A11" is a label used in that message, not a gate.

`A11: '__computedOutsideLedger'` at line 432 is therefore **a constant never read** — `_annotateGateStatus` only looks up `__gateFnByKey[key]` for keys present in the map it is given, and `A11` is never such a key.

A13 is not a `check` method in this range.* ARCHETYPES.A13 exists (line 149, 'Critical Mass Escalation') and ARCHETYPE_BASE_POINTS.A13 = 4 (line 195), but the flag is produced by an inline IIFE at lines 2897–2912 that reads this.criticalMass, populated by checkCriticalMassEscalation() (line 3005) — covered elsewhere. There is no checkA13_*` method.

Confirmed against the baseline. Across the 1,401 scored institutions, the archetype firing counts are: A1 9, A2 306, A3 83, A4 94, A5 205, A6 431, A7 395, **A8 0**, A9 224, A10 87, A12 121, A13 236, A14 82, R10 285, R11 0, R12 289. No `A11` key appears in any record.

PRIORITY FINDING (historical; fixed in v49.15) — why A8 fired zero times

Source: archetypes_A7_A14

Resolved 19 October 2026 (v49.15, README item 37). The three faults below were fixed: A8 now reads by exact key, reads net tuition from `percentChange`, and requires a scored admissions record. It fires; see its entry in §3. This section is kept as the record of the defect.

A8 **is** registered (line 425 in `__gateFnByKey`, line 2893 in the `archetypeFlags` map) and **is** evaluated for every institution. It is not dead code and it is not unreachable by registration. It fires zero times because **all three legs of its `marginThin` disjunction are permanently false**, for reasons that are individually verifiable in the source:

Leg 1 — `aomRisk >= 5` reads the wrong analyzer, and that analyzer is scored 0 in every sector. Line 1775 resolves its input with `pickAnalysis(this.completedAnalyses, ['adjustedoperatingmargin'])`. `pickAnalysis` (lines 8–36) matches on `ak === needle || ak.includes(needle) || needle.includes(ak)` and returns the **first** list element that satisfies any of the three, with no exact-match pre-pass. The analyzer whose key is `operatingmargin` satisfies `needle.includes(ak)` — `'adjustedoperatingmargin'.includes('operatingmargin')` is `true` — and `operatingmargin` is registered *before* `adjustedoperatingmargin` in `analysis_orchestrator.js` (lines 80 and 81 respectively), so it occupies the earlier position in `individual_results` and therefore in `completedAnalyses`. Simulated against the real 34-key list, `pickAnalysis(list,`

['adjustedoperatingmargin']) returns the operatingmargin record. aomRisk is therefore the Operating Margin analyzer's riskPoints, identical to omRisk on the next line.

Leg 2 — omRisk >= 5 is structurally impossible. UniversalOperatingMarginAnalyzer is declared at **0 points in every sector** in risk_scoring_engine.js (line 22 sector 1, line 63 sector 2, line 105 sector 3, line 155 sector 4). The orchestrator treats a framework value of exactly 0 as exclusion (analysis_orchestrator.js lines 228–251): maxRiskPoints is set to 0, riskLevel is rewritten to 'Informational', scoringDisabled = true, and riskPoints is forced to 0. In the validated baseline, Operating Margin reads 0.0 / 0.0 Informational for **1,401 of 1,401** scored institutions — no exceptions. omRisk is always 0, and by Leg 1 so is aomRisk. 0 >= 5 is false.

Leg 3 — ntrChg reads two fields no producer publishes. Lines 1777–1778 read analysisResults.fiveYearChange and then analysisResults.recentChange from the nettuitionperfte record. Grep across every .js file and the tool HTML finds **zero** producers of either name. net_tuition_per_fte_analyzer.js publishes the quantity as percentChange (line 154 of that file). this.num(undefined) returns null (line 651), and null ?? null is null, so the guard ntrChg != null is always false. This is not a new discovery — rat_selfcheck.js line 16 already lists it:

> * *checkA8 read analysisResults.fiveYearChange (never published)*

Net effect. marginThin (line 1781) evaluates to false || false || false for every institution in every sector. brandStrong is satisfiable and frequently satisfied — 245 of 1,401 institutions carry Brand Pipeline risk points of 0 or 1 — but triggered = brandStrong && marginThin (line 1782) can never be true. **A8 is unreachable in practice, and would remain unreachable even on perfect data**, because two of the three legs are wiring defects and the third is a framework exclusion.

Why the instrumentation did not catch it. A8 makes no getMetricValue() calls — it reads completedAnalyses directly through pickAnalysis — so the gate ledger (__gateLedger) holds no record for checkA8_BrandStrongMarginThin. In _annotateGateStatus (lines 802–810), a gate with no ledger record and triggered === false is stamped status: 'not_reached', which the surrounding comment glosses as "a front gate short-circuited". A8 is not short-circuiting; it is evaluating fully and failing on unreadable inputs. The diagnostic layer reports the one status that makes the defect look intentional.

Shared mechanics (apply to every gate below; not repeated in each entry)

Source: archetypes_A7_A14

Analyzer keys. analysis_orchestrator.js line 219 derives each key as analyzer.constructor.name.toLowerCase().replace('universal','').replace('analyzer',''). UniversalPrimaryReserveAnalyzer → primaryreserve, and so on.

The record a gate receives. analysis_orchestrator.js lines 369–388 push { analyzerKey, analyzerName, measure, riskPoints, maxRiskPoints, analysisResults, rawData, detailedReport, keyFindings }. There is **no** top-level title or headline — the fact pickAnalysis's comment at lines 22–28 was written to address.

Two different resolvers are in use, and they behave differently.

Route	Used by	Behavior
pickAnalysis(analyses, names) (lines 8–36)	A8	Single find pass; ak===needle || ak.includes(needle) || needle.includes(ak). **No exact-match pre-pass**, so a shorter key earlier in the list wins over the exact key later.
this.getMetricValue(k, f) → _getMetricValueRaw (lines 856–1126)	A7, A9	**Exact normKey equality across the whole list first** (lines 861–862), fuzzy substring only as a fallback (lines 863–867). Then keyMappings → exact field → declared metricAliases → **fail closed** (step 5, lines 986–1023).
Local norm+Set / direct iteration	A10, A12, A14	Exact normalized key match only; no fuzzy fallback.

The pickAnalysis route is the only one exposed to the substring-collision fault. Under _getMetricValueRaw, getMetricValue('adjustedoperatingmargin', 'riskPoints') resolves correctly because the exact pass runs to completion first.

Fail-closed resolver. As of 2026-08-30 (v49.2), _getMetricValueRaw step 5 returns null rather than substituting a generic field:

> *// 5) FAIL CLOSED. (2026-08-30, v49.2) // // A field that cannot be resolved by an EXPLICIT route — the keyMappings table, // the exact published name, a declared alias, or a declared per-key variation — // now returns null. A gate that cannot read its input must know it cannot read its // input. Nulls are visible: getMetricValue() records them in the gate ledger and // _annotateGateStatus() marks the gate 'unavailable'. A substituted number is not // visible at all.*

This affects A7 and A9 (which use `getMetricValue`). It does **not** protect A8, A10, A12 or A14, which bypass the resolver.

Points application. `calculateTotalAdjustmentPoints()` (lines 2949–2969) sums `flag.points` for triggered flags, adds pattern add-ons and accelerators, and caps the total at 50. The `points` value each gate returns is `ARCHETYPE_BASE_POINTS[code]` with a literal `??` default, so the map is the operative source.

Two module-level constants are never read. `APPLY_ARCHETYPE_POINTS` (line 166) is assigned and mutated by `setApplyArchetypePoints` (lines 169–171) but is read nowhere — points are always applied. `SEGMENT_WEIGHTS` (line 175) is read only by `scoreArchetypes()` (lines 359–368), which has **zero callers**. Both are flagged under Notes on the gates they nominally govern.

`__hasResilientScale()` — helper (lines 2021–2054)

Source: archetypes_A7_A14

Dependency check: no gate in this document calls it. Grep confirms exactly two callers, both outside scope: `checkR10_ResilientScale()` (line 2057) and `checkR12_BrandStrong()` (line 2140). A7, A8, A9, A10, A12 and A14 do not reference it. Documented here for completeness, as requested.

- **Returns**: `{ ok: boolean, enrollNow: number|null, basis: string }`.
- **What it tests**: whether an institution is "at scale" — reinterpreted in August 2026 as *large for its own type*, rather than large in absolute terms.

Inputs

Field	Line	Producer	Route
`enrollmenttrends.currentTotal`	2022	`enrollment`	`getMetricValue`; alias-resolved (`total` → `currentTotal`, lines 570–572)
`enrollmenttrends.total`	2023	`enrollment`	`getMetricValue`, second link
`analysisResults.peerMedianTotal`	2036–2037	`enrollment` (`unified_enrollment_analyzer.js` line 121)	**direct, strict typed read**
`analysisResults.peerMedianTotalCount`	2038–2039	`enrollment` (that file, line 122)	**direct, strict typed read**

The two peer fields are read outside `getMetricValue` on purpose, lines 2026–2030:

> *// Read the two new fields STRICTLY, not through getMetricValue(). That resolver has a // generic-variations fallback which, when a requested field is absent, returns whatever // 'current'/'value'/'ratio' happens to exist on the analyzer — silently substituting an // unrelated number. A scale bar built from an enrolment CHANGE PERCENT instead of an // enrolment COUNT would be catastrophic and invisible. An exact read cannot fail open.*

The analyzer lookup at lines 2031–2034 accepts `enrollment`, `enrollmenttrends` or `enrollmentanalysis` by exact normalized equality.

Gate logic

Step	Line	Rule	Direction
No enrollment	2024	`enrollNow === null` → `{ ok: false }`	**fails closed**
Peer median usable	2041	`peerMedian === null || peerMedian <= 0 || peerN < 10` → fall back	**floor** of 10 peers
Absolute fallback	2042	`ok = enrollNow >= 10000`	**floor**
Relative	2045	`enrollNow >= peerMedian`	**floor**

Step	Line	Rule	Direction
Absolute	2046	`enrollNow >= 10000`	**floor**
Result	2048	`ok = relative || absolute`	—

- **Sector differences**: none explicit; the peer median is computed over whatever peer group the rest of the analysis uses, so the bar is implicitly sector- and Carnegie-relative.
- **Data-gap behavior**: **fails closed on enrollment** (line 2024), and **degrades deliberately** on the peer median — a missing or thin peer group (n < 10) reverts to the original absolute 10,000 bar rather than guessing, with `basis` recording which rule applied.
- **2026-08 changes**: the whole helper was introduced on 2026-08-27. Verbatim, lines 2000–2019:

> *// 2026-08-27: TYPE-RELATIVE SCALE for the resilience gates. // // R10 and R12 both tested \enrollment >= 10,000\. Measured across the 2026 baseline // (1,456 scored institutions, all sixteen Carnegie x sector cells) that bar was // cleared by 97% of public doctoral very-high-research institutions and by 0% of // public C21, 0% of private C19/C20/C21 and 1% of private C22. The resilience layer // was not testing resilience; it was testing enrolment. // // Replacement: scale is sufficient if the institution is at or above the MEDIAN // enrolment of its own peer group, OR clears the old 10,000 bar outright. The second // clause makes the change monotone — verified against the baseline, zero institutions // holding a credit today lose it — while the first makes "scale" mean large FOR YOUR // TYPE, which is what the concept was always meant to mean. A 2,000-student liberal // arts college is at scale for a liberal arts college. // // The peer median is published by unified_enrollment_analyzer as // analysisResults.peerMedianTotal, computed over the same peer group the rest of the // analysis uses, so it recalibrates on every data refresh with no hard-coded number // to go stale. If it is missing or the peer group is too small, the gate falls back // to the original absolute rule rather than guessing.*

The associated repricing is recorded in `ARCHETYPE_BASE_POINTS`, lines 191–194:

> *// R10 -3 -> -8, R12 -2 -> -6. Resilience was priced against a layer-2 that averages +22 // points. 201 institutions held a credit and NINE had a band improved by it. Widening // eligibility (see __hasResilientScale) without repricing would have made a decorative // feature available to more people.*

Notes

- The comment at lines 2026–2030 describes a `getMetricValue` fail-open that **no longer exists** — step 5 was made fail-closed on 2026-08-30 (lines 986–1023), and the code there explicitly credits this helper for the diagnosis: *"__hasResilientScale() already refused to use this resolver for exactly this reason … That workaround was the right diagnosis applied in one place. The resolver is the fault."* The strict read is now belt-and-braces rather than necessary. Both comments are current and consistent; the helper's rationale is simply historical.
- The comment cites a 1,456-institution baseline; the current validated baseline is 1,401 scored (1,551 records). The figures quoted in it have not been recomputed against the current run.

Cross-cutting findings

Source: archetypes_A7_A14

Collected for the punch list. None of these has been changed in the source.

[Item 1 fixed in v49.15; P9's gate 3 fixed in v49.11.] 1. **A8 cannot fire.** Three independent faults; see the priority finding. 2 points of documented risk are never charged, across all sectors and all institutions. 2. **`pickAnalysis` has no exact-match pre-pass** (lines 8–36). Any caller passing a key that *contains* a shorter registered key as a substring gets the shorter one, if it is earlier in the list. `adjustedoperatingmargin` → `operatingmargin` is the live instance. A second latent instance: `endowmentdrawsustainability` → `endowment`. A12 and A10 both name `endowmentdrawsustainability` but neither uses `pickAnalysis`, so the collision is currently unexploited. 3. **Gates that bypass `getMetricValue` are invisible to the instrumentation.** A8, A10, A12 and A14 make no ledger entries, so `_annotateGateStatus` (lines 802–810) reports `'not_reached'` for every non-firing institution — indistinguishable from a legitimate short-circuit. This is why A8's zero rate survived to the validated baseline. 4. **Operating Margin is 0-point in every sector** yet is still read as a live risk input by A8 (lines 1775–1776) and A9 (line 1845). In A9 it silently defeats the `marginRP !== null` guard. 5. **`FAMILY.cfi`** (line 1902) is a constant never read — no CFI analyzer is registered. 6. **`__gateFnByKey.A11`** (line 432) is a constant never read — no A11 gate is built. 7. **`APPLY_ARCHETYPE_POINTS`** (line 166) and **`setApplyArchetypePoints`** (lines 169–171) are write-only; nothing reads

the flag. The "preview only" language in the comments at lines 165 and 178 no longer describes the behavior. 8. **`SEGMENT_WEIGHTS`** (line 175) and **`scoreArchetypes()`** (lines 359–368) have no callers. Segment weighting is documented but not implemented. 9. **A12 fail-open branch**: line 1924 skips a record only when `mx !== null && mx < 4`. A null `maxRiskPoints` with `riskLevel === 'critical'` counts as a hit at any weight. 10. **A9's collapse guard tests only the long-term series** (line 1829), so a null fourteen-year change plus a catastrophic five-year change still passes `stillStanding`. 11. **A14's header statistics are stale** relative to the validated 2026 baseline (see that section's Notes). 12. **A7's `grant_distribution` fallback** (line 1723) targets an analyzer key that does not exist.

Scope note — the three gates and their shared machinery

Source: resilience

Element	Method	Lines	Points
R10	`checkR10_ResilientScale()`	2056–2075	`-8`
R11	`checkR11_ResearchStrong()`	2078–2135	`-4` (never applied)
R12	`checkR12_BrandStrong()`	2137–2149	`-6`
—	`__hasResilientScale()` (shared helper, R10 + R12)	2021–2054	n/a

Registration in the live archetype map, `detectArchetypeFlags()` (lines 2914–2916):

```
R10: this.checkR10_ResilientScale(),
R11: this.checkR11_ResearchStrong?.() ?? { triggered:false, points:0, reason:'not implemented' },
R12: this.checkR12_BrandStrong()
```

Points are summed in `calculateTotalAdjustmentPoints()` (lines 2949–2969):

```
Object.values(this.archetypeFlags).forEach(flag => {
    if (flag.triggered) total += flag.points;
});
...
return Math.min(total, 50);
```

There is a **ceiling** at `+50` and **no floor**: a credit can drive the Layer 2 total negative without clamping at this step. See Notes under R10 for the severity-override interaction, which is where credits are in practice discarded.

Cross-cutting notes on the resilience layer

Source: resilience

1. **All three credits are conjunctive and all fail closed.** Not one of the eight leg-level comparisons across R10, R11 and R12 can be satisfied by a null. Every one is guarded by an explicit `!== null` test before the comparison. The layer cannot give points away on missing data through its own logic.

2. **The one place a credit could still fail open is a producer contract, not a gate.** R10's `primRisk <= 2 && viabRisk <= 2` and R12's `accRisk <= 2` read raw `riskPoints` with no `riskLevel` guard. They are safe today only because their producers return `success: false` on not-found and `analysis_orchestrator.js` line 331 drops non-success records. R11 is the only R-series gate that guards `riskLevel` explicitly (lines 2098–2104) — and it is the one that is switched off.

3. **Two of the three credits are live; one is a documented liability.** Of `-8`, `-4` and `-6` in the published point table, `-4` (R11) cannot be earned by any institution.

4. **Credits do not survive a severity override.** `applySeverityOverride()` line 2248 takes `Math.max(severityPoints, archetypePatternTotal)`, discarding the archetype total — credits included — whenever the severity pricing is larger.

5. **Naming.** The helper `__hasResilientScale()` is the sole definition of "at scale" in Layer 2. Its double-underscore prefix keeps it out of `_callingGate()`'s regex (`/(check[A-Z][A-Za-z0-9_]*|calculateP\d+_[A-Za-z0-9_]*)/`, line 783), so lookups made inside it are correctly attributed to `checkR10_ResilientScale` / `checkR12_BrandStrong` in the gate ledger rather than to the helper.

0. Shared machinery

Source: patterns_P1_P5

Everything in this section is common to P1–P5 and is not repeated in each section.

0.1 How the patterns are invoked

`calculatePatternAddons(severityCondition)`, lines 2848–2866, is the sole call site for all five methods. There is no other caller anywhere in the repository (`calculateP1…calculateP5` appear nowhere in `report_generator.js`, `ui_integration_layer.js`, `rat_selfcheck.js`, `simplified_unified_integration.js` or the tool HTML).

```
  this.patternAddons = {
 'P1_SienaPattern': this.calculateP1_SienaPattern(),
'P2_HighDiscount_FlatEnroll': this.calculateP2_HighDiscount_FlatEnroll(),
    'P3_AppropsDep_EnrollDecl':       this.calculateP3_AppropsDepEnrollDecl(severityCondition),
    'P4_NetTuitionRisk':              this.calculateP4_NetTuitionRisk(severityCondition),
    'P5_UNA_Risk':                    this.calculateP5_UNARisk(),
```

`severityCondition` is computed once at line 3296 (`const severityCondition = this.checkSeverityCondition();`) and passed to P3 and P4 only. Because that is the only call site, the `severity` argument is **never** `undefined` in production; the `severity?.` optional chaining in both methods is defensive only.

0.2 The severity object

`checkSeverityCondition()`, lines 2154–2167:

```
        const enroll5 = this.getEnroll5();
        const viability = this.num(this.getMetricValue('viability', 'viability_ratio'));
        const severeEnrollment = (enroll5 !== null && enroll5 <= -25);
        const weakViability = (viability !== null && viability <= 0.30);

        return {
            hasSeverity: (!!severeEnrollment || !!weakViability),
```

Field	Rule	Direction
`severeEnrollment`	`enroll5 <= -25` (five-year total enrollment change, %)	**ceiling**
`weakViability`	`viability <= 0.30` (viability ratio)	**ceiling**
`hasSeverity`	`severeEnrollment` OR `weakViability`	—

Only `hasSeverity` is read by P3 and P4.

0.3 Points accounting

`calculateTotalAdjustmentPoints()`, lines 2949–2969, sums triggered archetype flags, triggered pattern add-ons and triggered accelerators, then applies `return Math.min(total, 50);` (line 2968). Pattern points are therefore additive with archetype points and share a single 50-point layer cap. There is no per-pattern cap and no cap on the pattern block alone.

0.4 Sector normalization

Constructor, lines 391–395:

```
this.sector = (sector === 1 || sector === '1') ? '1' :
              (sector === 2 || sector === '2') ? '2' :
              (sector === 3 || sector === '3') ? '3' :
              (sector === 4 || sector === '4') ? '4' :
              String(sector || '1');
```

`this.sector` is **always a string** after this. `getSectorName()` (lines 2755–2760) maps `'1'→'Public'`, `'2'→'Private'`, `'3'→'For-Profit'`, `'4'→'Community College'`, anything else `'Unknown'`.

0.5 `num()` and `pct()`

Lines 650–662:

```
    num = v => {
        if (v === null || v === undefined) return null;
        if (typeof v === 'number' && Number.isFinite(v)) return v;
        const s = String(v).replace(/,/g, '').replace(/[^0-9.\-eE]/g, ''); // drop % and symbols
        const n = parseFloat(s);
        return Number.isFinite(n) ? n : null;
    };

    pct = v => {                    // returns % on [0..100] or null
```

```
        v = this.num(v);
        if (v === null) return null;
        return (v <= 1 ? v * 100 : v); // 0.29 -> 29; 29 -> 29
    };
```

`pct()` is a **guess about scale, not a conversion**. Any value at or below 1 is multiplied by 100, whatever it means. This matters at three P1–P5 inputs and is reported under the relevant Notes.

0.6 `getMetricValue()` — the resolver, and what "fail closed" now means

`getMetricValue(analyzerKey, metricKey)` (lines 724–761) is instrumentation around `_getMetricValueRaw()` (lines 856–1126). The raw resolver tries, in order:

1. Find the analyzer: exact normalized match on `analyzerKey` against `analyzerAliases`, then a substring match in either direction (lines 858–867). 2. `keyMappings` table (lines 878–944), including deep paths such as `shortTerm.totalChangePercent`. 3. The exact `metricKey` as published. 4. Declared `metricAliases` at the top level (lines 980–984, added 2026-08-30). 5. **Fail closed** — the generic-substitution list is now empty. 6. Per-key variations for the three percent-of-spend metrics only. 7. Deep container sweep of `results.data/metrics/values/meta/shortTerm/summary`; peer containers are searched **only** when the key or its mapping starts with `peer`. 8. `return null`.

The step-5 change is the direct cause of the P2 rebuild and is quoted in full under P2.

`_annotateGateStatus()` (lines 792–835) stamps each pattern result with a `status`. Relevant to P1–P5:

status	Condition
`unledgered (not_reached before v49.10)`	no getMetricValue lookup recorded and triggered === false; since v49.10 statusDetail prints the gate's own reason
`evaluated_unledgered`	no lookup recorded but `triggered === true` (inputs read outside the ledger)
`unavailable`	at least one unresolved input and `triggered === false`
`evaluated`	resolved, or resolved via fallback with some optional inputs unresolved
`not_applicable`	the analyzer itself is absent for this institution
`degraded`	a fail-open input (now only counts refusals)

Update (v49.10, 15 October 2026). A gate with no ledger record that does not trigger is now stamped `unledgered`: it was evaluated, did not trigger, and read its inputs outside the ledger. Its status detail prints the gate's own reason. `not_reached` is no longer assigned to such gates, and every `not_reached` remark in the entries above describes the behavior before v49.10. Current method: `_annotateGateStatus()`, from line 883.

0.7 Enrollment helpers — two different statistics

`getEnroll5()`, lines 1128–1135. The **recent** window.

```
    getEnroll5() {
        let v = this.num(this.getMetricValue('enrollment','5_year_change'));
        if (v !== null) return v;
        // fallback to any "recent change" field the pipeline provides
        v = this.num(this.getMetricValue('enrollment','recent_change')
                  ?? this.getMetricValue('enrollment','2018_2023_change'));
        return v;
    }
```

All three metric keys map to `shortTerm.totalChangePercent` (lines 898–900), so the fallback chain resolves to the same field. That field is 2019→2024 total enrollment change in percent, falling back to a 2018 base year if 2019 is missing (`unified_enrollment_analyzer.js` lines 371–381).

`getEnrollmentDeclineWorst()`, lines 2792–2833. The **worst-case** figure. Reads `completedAnalyses` directly, not through `getMetricValue`, matching `normKey(analyzerKey) === 'enrollment'`:

```
    const values = [
        ar.totalChangePercent,            // 14-year total enrolment change, percent
        ar.shortTerm?.totalChangePercent  // 5-year total enrolment change, percent
    ]
```

and returns the minimum. `totalChangePercent` is 2010→2024 (fourteen years). A 2026-08-29 comment at lines 2811–2816 records that the undergraduate/graduate *share* metrics were removed from this list because they are percentage points of mix, not percent of volume.

Because these are different statistics, gates disagree about what "enrollment decline" means. P2 uses `getEnroll5()`. P1, P3 and P4 use `getEnrollmentDeclineWorst()`. The P2 rebuild comment (lines 2293–2296) explicitly names the worst-case reading as the wrong statistic; P1, P3 and P4 were not changed.

0.8 `getNtrNorm()`

Lines 1139–1182. The net-tuition-per-FTE level index: ~`1.0` at the peer average, <1 below peers.

```
getNtrNorm() {
    // Find NET TUITION PER FTE analyzer (NOT tuition dependency!)
    const ntrAnalyzer = this.completedAnalyses.find(a => {
        const key = this.normKey(a.analyzerKey || '');
        // Be specific: look for "ntr" or "nettuition" but NOT just "tuition"
        return key === 'nettuitionperfte' ||
               key.includes('nettuition') ||
               key.includes('ntr');
    });
```

Resolution order:

1. No matching analyzer, or no `analysisResults` → `return null` (lines 1149–1155). 2. `ar.net_tuition_per_fte_normalized`, if non-null, non-undefined **and** `Number.isFinite` → return it (lines 1162–1167). 3. Otherwise compute `ar.currentValue / ar.peerAverage`, guarded on both being finite and `peer !== 0` (lines 1170–1178). 4. Otherwise `return null` (line 1181).

The producer (`net_tuition_per_fte_analyzer.js` lines 143–146) publishes `net_tuition_per_fte_normalized` as `current2024 / peerAverages.current2024`, or `null` when the peer count is zero — i.e. steps 2 and 3 compute the same quantity from the same numbers, and step 3 is reachable only when the producer's own guard failed but `currentValue` and `peerAverage` are both finite and `peerAverage !== 0`.

`getNtrNorm()` bypasses `getMetricValue()` entirely, so its lookups are **not** recorded in the gate ledger. It is called by P2 (as a corroborator), P4 (as its primary input), `applySeverityOverride()` (line 2180) and the export block at line 3335.

Note: `getNtrNorm()` is the only unguarded `this.completedAnalyses.find(...)` among these helpers — line 1141 has no `|| []`. If `completedAnalyses` were ever undefined it would throw rather than return null, but the constructor (lines 383–388) guarantees an array.

0.9 Analyzer keys

`analysis_orchestrator.js` line 219 builds the key from the class name:

```
const analyzerKey = analyzer.constructor.name.toLowerCase().replace('universal', '').replace('analyzer', '');
```

The instantiated classes are the `Universal*Analyzer` wrappers declared in the tool HTML (`analysis_orchestrator.js` lines 68–76 register `UniversalEnrollmentAnalyzer`, `UniversalNetTuitionPerFTEAnalyzer`, and so on). The keys relevant here are therefore:

Wrapper class	analyzerKey
`UniversalEnrollmentAnalyzer`	`enrollment`
`UniversalNetTuitionPerFTEAnalyzer`	`nettuitionperfte`
`UniversalTuitionDependencyAnalyzer`	`tuitiondependency`
`UniversalAppropriationsAnalyzer`	`appropriations`
`UniversalAcceptanceYieldAnalyzer`	`acceptanceyield`
`UniversalUnrestrictedNetAssetsAnalyzer`	`unrestrictednetassets`
`UniversalPrimaryReserveAnalyzer`	`primaryreserve`
`UniversalStructuralMarginAnalyzer`	`structuralmargin`
`UniversalAdjustedOperatingMarginAnalyzer`	`adjustedoperatingmargin`

`analysis_orchestrator.js` line 331 pushes an entry onto the record **only** when `data.result.success` is true, so an analyzer that returned a failure result is absent from `completedAnalyses` altogether rather than present with empty fields.

Appendix A — Cross-pattern comparison

Source: patterns_P1_P5

	P1	P2	P3	P4	P5
Lines	2426–2451	2283–2352	2354–2379	2380–2420	2453–2498
Base points	4	4 (P2_BASE_POINTS)	10	5	6
Max points	6	6	12	7	8
Fallback award	—	—	—	4	4
Takes severity	no	no	**yes**	**yes**	no
Sector confined to	'2' private	'2' private	'1' public	none (CC gate inert)	none (CC gate inert)
Sector gate position	after 3 lookups	first statement	first statement	first statement	first statement
Enrollment statistic	worst-case	five-year	worst-case	worst-case	not used
Fails open on a gap	no	no	no	**yes** (riskPoints >= 4)	narrowly (riskPoints >= 3 **and** UNA < 0)
Ledgered lookups	2	1	2–6	3 (+1 on fallback)	1 (+2 on fallback)
2026 baseline firing	9.1% (14.1% of privates)	2.1% (3.3% of privates)	8.7% (24.4% of publics)	34.6%	19.8%
Discrimination (line 215–221)	+13.3	−2.2 (pre-rebuild)	+17.2	+11.1	+21.3

Firing rates computed from baseline_recs.json, restricted to the 1,401 rows with excluded === false (499 public, 902 private).

Appendix B — Defects and open items, ranked

Source: patterns_P1_P5

1. **P5 / pct()**: a primary reserve ratio in (0, 1) percent is rescaled ×100 and the lowPR corroborator inverts, suppressing the pattern for the weakest institutions. Population impact **UNCLEAR**. 2. **P4 fallback**: awards 4 points with no peer comparison when ntrNorm is unavailable, and leaves no ledger trace, so the gate reports not_reached rather than unavailable. 3. **P1 lookup order**: three lookups precede the sector gate, so P1's gate status is misreported for every public institution. 4. **P3 / pct()**: an appropriations dependency in (0, 1) percent is rescaled ×100 and clears the 40% floor. 5. **P1 / pct()**: a tuition dependency ratio above 1.0 is not rescaled and fails the 70% floor. 6. **P4 acceptance/yield scale**: **UNCLEAR** from source; pct() reconciles the two plausible encodings except in (0, 1). 7. **P2 analyzer-key match**: strict equality on nettuitionperfte where every sibling helper is loose; a rename would silently zero the pattern with no diagnostic. 8. **P5 private AOM threshold** (−3.0) disagrees with checkFinancialDistress and applySeverityOverride (−5.0), unexplained. 9. **P5 public margin source**: the −2.0 threshold is applied to structural margin here and to adjusted operating margin in two other gates. 10. **P5 reason string** labels the public structural-margin leg "weak AOM". 11. **P4 / P5 CC gates**: this.sector === 4 (numeric) unreachable; the sectorName test redundant; reason vs reasoning key inconsistency. 12. **P3 reason string** is a fixed literal that reports appropriations dependence even when only the low-tuition-dependency leg fired. 13. **P5 helper comment** says < 25; the code is <= 25. 14. **P5 fallback** second link getMetricValue('una','riskPoints') is dead (shared alias list). 15. **P3 escalation** re-tests enroll5 !== null inside a block already guarded on it. 16. **Open action item**, from the P2 calibration comment: "Confirm the new group's discrimination on the next full re-run before treating this as settled."

Shared machinery (applies to every entry below; stated once)

Source: patterns_P6_P9

How a record reaches Layer 2

Every analyzer in the tool is a Universal*Analyzer extends SectorAwareAnalyzer (analysis_orchestrator.js lines 65–168 registers only Universal* classes). The orchestrator calls analyzer.analyze(institution, peerCriteria) at line 218, which always resolves through the **base class**, SectorAwareAnalyzer.analyze(). On the non-throwing, non-data-gap path that method returns (modular_tool_v48 no data cap.html lines 1045–1049):

```
return {
    success: true,
    analysisResults: analysisResult.analysisResults,
    detailedReport: analysisResult.detailedReport || {}
};
```

Three keys. `rawData`, `measure`, `institution`, `peerGroup` and `metadata` returned by the inner `Unified*` analyzer are **discarded at this boundary**. The data-gap path, `toDataGapResult()` (lines 1006–1021), likewise emits only `{success, error, analysisResults, detailedReport}`.

The orchestrator then builds the Layer 2 record (`analysis_orchestrator.js` 369–388) from `data.result`, i.e. from the three-key object above. This is load-bearing for P9 and is traced in full under that entry.

Analyzer keys

`analysis_orchestrator.js` line 219:

```
const analyzerKey = analyzer.constructor.name.toLowerCase().replace('universal', '').replace('analyzer', '');
```

So `UniversalEndowmentAnalyzer` → `endowment`; `UniversalEndowmentDrawSustainabilityAnalyzer` → `endowmentdrawsustainability`; `UniversalEndowmentDependencyAnalyzer` → `endowmentdependency`; `UniversalStructuralMarginAnalyzer` → `structuralmargin`; `UniversalAcceptanceYieldAnalyzer` → `acceptanceyield`; `UniversalEnrollmentAnalyzer` → `enrollment`.

The resolver, after the 2026-08-30 fail-closed change

`_getMetricValueRaw()` (lines 856–1126) resolves in this order: (1) locate the analyzer by `analyzerAliases` + normalized fuzzy match; (2) `keyMappings` table, dotted paths supported; (3) exact `metricKey` on `analysisResults`; (4b) declared `metricAliases` at top level; (5) **fail closed** — `genericVariations` is now `[]`; (6) `perKeyVariations`; (7) deep container sweep over `results.data/.metrics/.values/.meta/.shortTerm/.summary` (peer containers only when the key or its mapping begins with `peer`); (8) return `null`.

The step-5 comment states the change verbatim:

> *// 5) FAIL CLOSED. (2026-08-30, v49.2) // // This step used to be a generic substitution: // // const genericVariations = ['current','currentValue','value','percentage', // 'ratio','change','trend','currentPercentage', // 'currentPercent','percent']; // for (const k of genericVariations) if (results[k] !== undefined) return results[k]; // // When a gate asked for a field its producer does not publish, the resolver // returned whatever generic field happened to exist on that analyzer. Not null — // a number. Of a different quantity, in a different unit, which the gate then // compared against its threshold as though it were the thing requested.*

and

> *// A field that cannot be resolved by an EXPLICIT route — the keyMappings table, // the exact published name, a declared alias, or a declared per-key variation — // now returns null. A gate that cannot read its input must know it cannot read its // input.*

P9 is unaffected by this change. P9 makes no `getMetricValue()` calls at all; it reads `this.completedAnalyses` directly. P7 and P8 do use the resolver, and all four of their fields have explicit routes (traced in their entries).

Numeric helpers

```
num = v => { if (v === null || v === undefined) return null;
             if (typeof v === 'number' && Number.isFinite(v)) return v;
             const s = String(v).replace(/,/g,'').replace(/[^0-9.\-eE]/g,'');
             const n = parseFloat(s); return Number.isFinite(n) ? n : null; };

pct = v => { v = this.num(v); if (v === null) return null;
             return (v <= 1 ? v * 100 : v); };   // 0.29 -> 29; 29 -> 29
```

`pct()` is a units-guessing shim, not a conversion: it multiplies by 100 only when the value is `<= 1`. See Notes under P7 and P8.

Registration

`calculatePatternAddons()` (lines 2848–2866) is the sole registry. **Exactly eight patterns are registered**, transcribed verbatim from lines 2851–2860:

Key in this.patternAddons	Method called
P1_SienaPattern	this.calculateP1_SienaPattern()
P2_HighDiscount_FlatEnroll	this.calculateP2_HighDiscount_FlatEnroll()
P3_AppropsDep_EnrollDecl	this.calculateP3_AppropsDepEnrollDecl(severityCondition)
P4_NetTuitionRisk	this.calculateP4_NetTuitionRisk(severityCondition)
P5_UNA_Risk	this.calculateP5_UNARisk()
P7_BrandPositioning	this.calculateP7_BrandPositioning()
P8_MarketPosition	this.calculateP8_MarketPosition()
P9_EndowmentDrainRisk	this.calculateP9_EndowmentDrainRisk()

There is no P6 entry. __gateFnByKey (lines 435–442) carries the same eight keys and no P6. A pattern absent from this map is never constructed, never annotated, and can never contribute points.

Points are taken straight from each method's returned points; there is no separate points table for patterns. calculateTotalAdjustmentPoints() (2949–2969):

```
Object.values(this.patternAddons).forEach(pattern => {
    if (pattern.triggered) total += pattern.points;
});
...
return Math.min(total, 50);
```

Ordering

analyze() (3289–3330) runs Step 1b checkCriticalMassEscalation(), Step 2 detectArchetypeFlags(), then **Step 3 calculatePatternAddons(severityCondition)**. So this.archetypeFlags.A10 is populated before P9's double-count guard reads it. That ordering is correct.

Baseline evidence

baseline_recs.json, 1,551 records (the 1,401-institution validated cohort is a subset). Pattern frequencies:

Pattern	Times fired
P4_NetTuitionRisk	507
P8_MarketPosition	456
P5_UNA_Risk	284
P3_AppropsDep_EnrollDecl	136
P1_SienaPattern	130
P7_BrandPositioning	83
P2_HighDiscount_FlatEnroll	30
P6_*	**0 — never present as a key**
P9_EndowmentDrainRisk	**0**

A10 fired 87 times, which independently confirms that the exact-match analyzer lookup shared by A10 and P9 gate 1 resolves structuralmargin correctly.
