THE BEGINNERS GUIDE TO noSQL

THE

WHY

WE ARE STORING MORE DATA NOW THAN WE EVER HAVE BEFORE

THE WHY

WE ARE STORING MORE DATA NOW THAN WE EVER HAVE BEFORE

CONNECTIONS BETWEEN OUR DATA ARE GROWING ALL THE TIME

THE
WHY

WE ARE STORING MORE DATA NOW THAN WE EVER HAVE BEFORE

CONNECTIONS BETWEEN OUR DATA ARE GROWING ALL THE TIME

WE DON'T MAKE THINGS KNOWING THE STRUCTURE FROM DAY 1

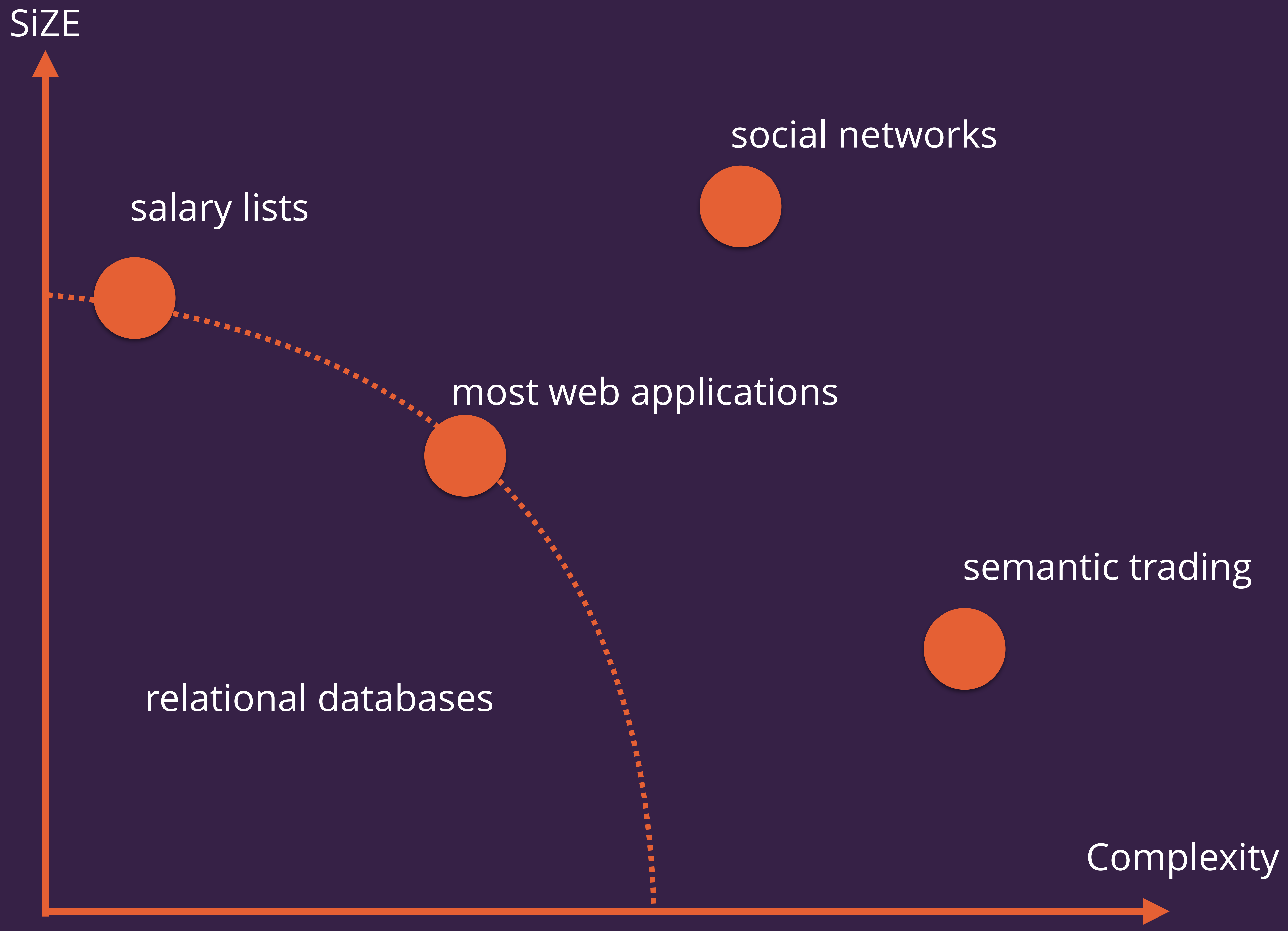
THE WHY

WE ARE STORING MORE DATA NOW THAN WE EVER HAVE BEFORE

CONNECTIONS BETWEEN OUR DATA ARE GROWING ALL THE TIME

WE DON'T MAKE THINGS KNOWING THE STRUCTURE FROM DAY 1

SERVER ARCHITECTURE IS NOW AT A STAGE WHERE WE CAN TAKE ADVANTAGE OF IT


SiZE
social networks
salary lists
most web applications
semantic trading
relational databases
Complexity

NOSQL

USE CASES

LARGE DATA VOLUMES

MASSIVELY DISTRIBUTED ARCHITECTURE
REQUIRED TO STORE THE DATA
GOOGLE, AMAZON, FACEBOOK, 100K SERVERS

NOSQL

USE CASES

LARGE DATA VOLUMES

MASSIVELY DISTRIBUTED ARCHITECTURE
REQUIRED TO STORE THE DATA
GOOGLE, AMAZON, FACEBOOK, 100K SERVERS

EXTREME QUERY WORKLOAD

IMPOSSIBLE TO EFFICIENTLY DO JOINS AT THAT
SCALE WITH AN RDBMS

NOSQL

USE CASES

LARGE DATA VOLUMES

MASSIVELY DISTRIBUTED ARCHITECTURE
REQUIRED TO STORE THE DATA
GOOGLE, AMAZON, FACEBOOK, 100K SERVERS

EXTREME QUERY WORKLOAD

IMPOSSIBLE TO EFFICIENTLY DO JOINS AT THAT
SCALE WITH AN RDBMS

SCHEMA EVOLUTION

SCEMA FLEXIBILITY IS NOT TRIVIAL AT A LARGE
SCALE BUT IT CAN BE WITH NO SQL

NOSQL

PROS AND CONS

PROS

MASSIVE SCALABILITY
HIGH AVAILABILITY
LOWER COST
SCHEMA FLEXIBILITY
SPARCE AND SEMI STRUCTURED DATA

NOSQL

PROS AND CONS

PROS

MASSIVE SCALABILITY
HIGH AVAILABILITY
LOWER COST
SCHEMA FLEXIBILITY
SPARCE AND SEMI STRUCTURED DATA

CONS

LIMITED QUERY CAPABILITIES
NOT STANDARDISED (PORTABILITY MAY BE AN ISSUE)
STILL A DEVELOPING TECHNOLOGY

FOUR

EMERGING TRENDS IN NOSQL DATABASES

BIGTABLE

KEY VALUE

GRAPHDB

DOCUMENT

BUT FIRST…

IMAGINE A LIBRARY

LOTS OF DIFFERENT FLOORS

DIFFERENT SECTIONS ON EACH FLOOR

DIFFERENT BOOKSHELVES IN EACH SECTION

LOTS OF BOOKS ON EACH SHELF

LOTS OF PAGES IN EACH BOOK

LOTS OF WORDS ON EACH PAGE

EVERYTHING IS WELL ORGANISED AND EVERYTHING HAS A SPACE

BUT FIRST...

IMAGINE A LIBRARY

WHAT HAPPENS IF WE BUY TOO MANY BOOKS!?

(THE WORLD EXPLODES AND THE KITTENS WIN)

BUT FIRST...

IMAGINE A LIBRARY

WHAT HAPPENS IF WE WANT TO STORE CDS ALL OF A SUDDEN!?

(THE WORLD EXPLODES AND THE KITTENS WIN)

BUT FIRST…

IMAGINE A LIBRARY

WHAT HAPPENS IF WE WANT TO GET RID OF ALL BOOKS THAT MENTION KITTENS

(KITTENS STILL WIN)

BIG TABLE

BEHAVES LIKE A STANDARD RELATIONAL DATABASE BUT WITH A SLIGHT CHANGE

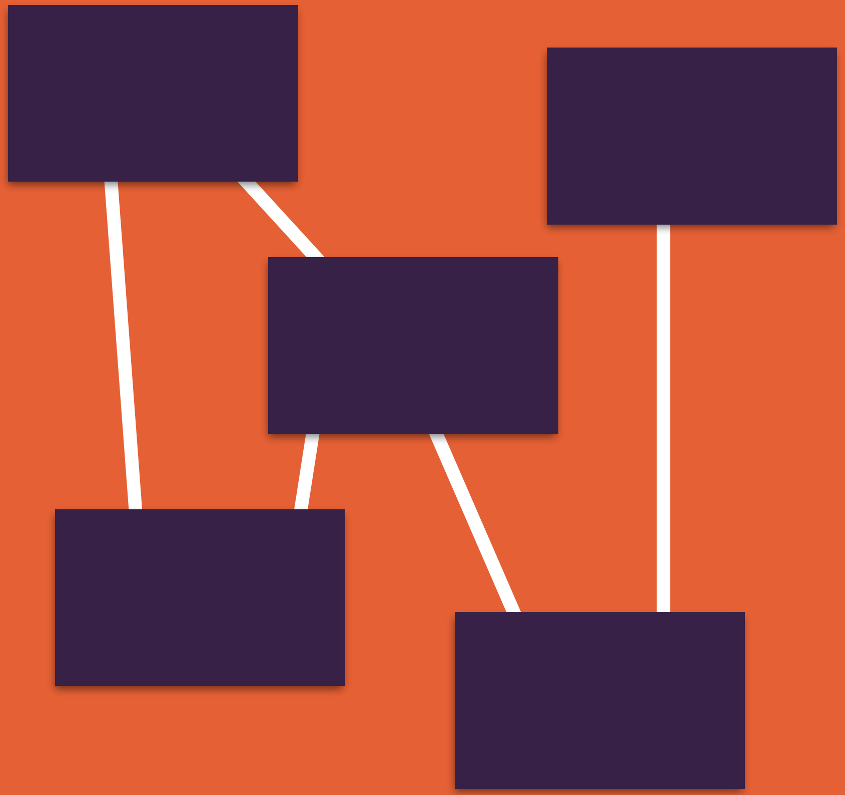
DESIGNED TO WORK WITH A LOT OF DATA...A REALLY BIG CRAP TON

CREATED BY GOOGLE AND NOW USED BY LOTS OF OTHERS

http://research.google.com/archive/spanner.html

http://research.google.com/archive/bigtable.html

BIG TABLE



THIS IS A STANDARD RELATIONAL DATABASE

THIS IS A BIG TABLE DATABASE
(AND NOW THE NAME MAKES SENCE!)

BIG TABLE

"A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes."

BIG TABLE

"A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes."

BIG TABLE

"A Bigtable is a sparse, distributed, persistent multidimensional sorted map. The map is indexed by a row key, column key, and a timestamp; each value in the map is an uninterpreted array of bytes."

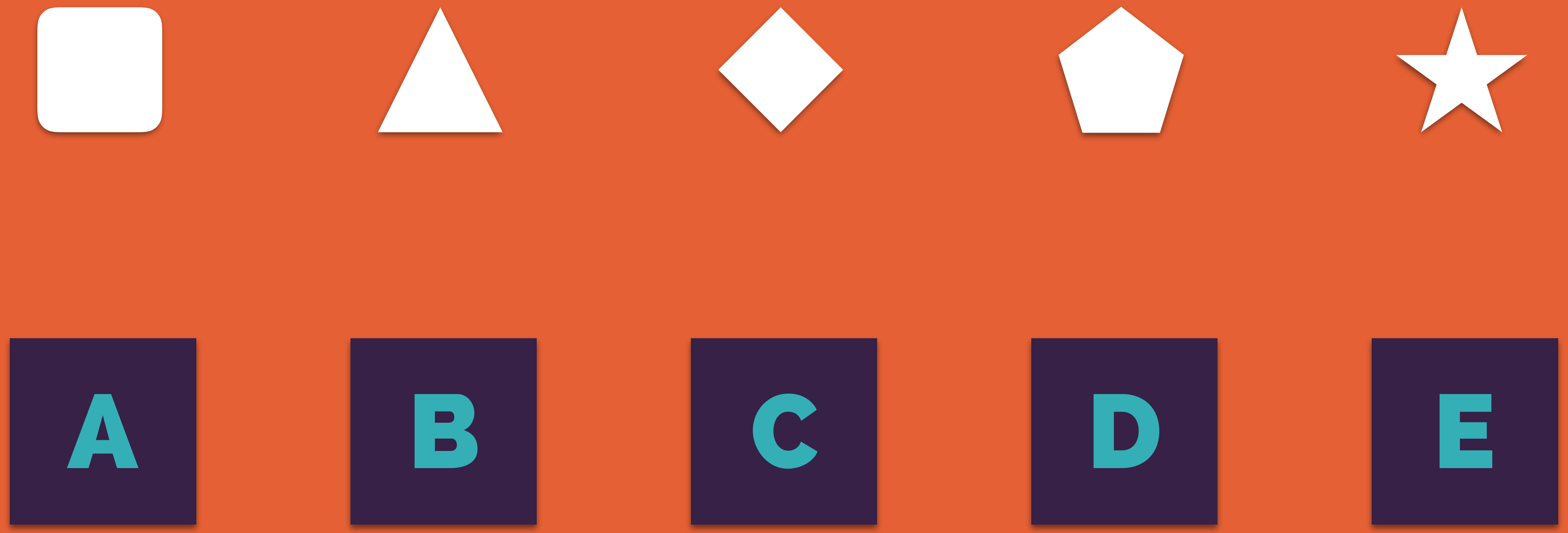
KEY VALUE

AGAIN, DESIGNED TO WORK WITH A LOT OF DATA

EACH BIT OF DATA IS STORED IN A SINGLE COLLECTION

EACH COLLECTION CAN HAVE DIFFERENT TYPES OF DATA

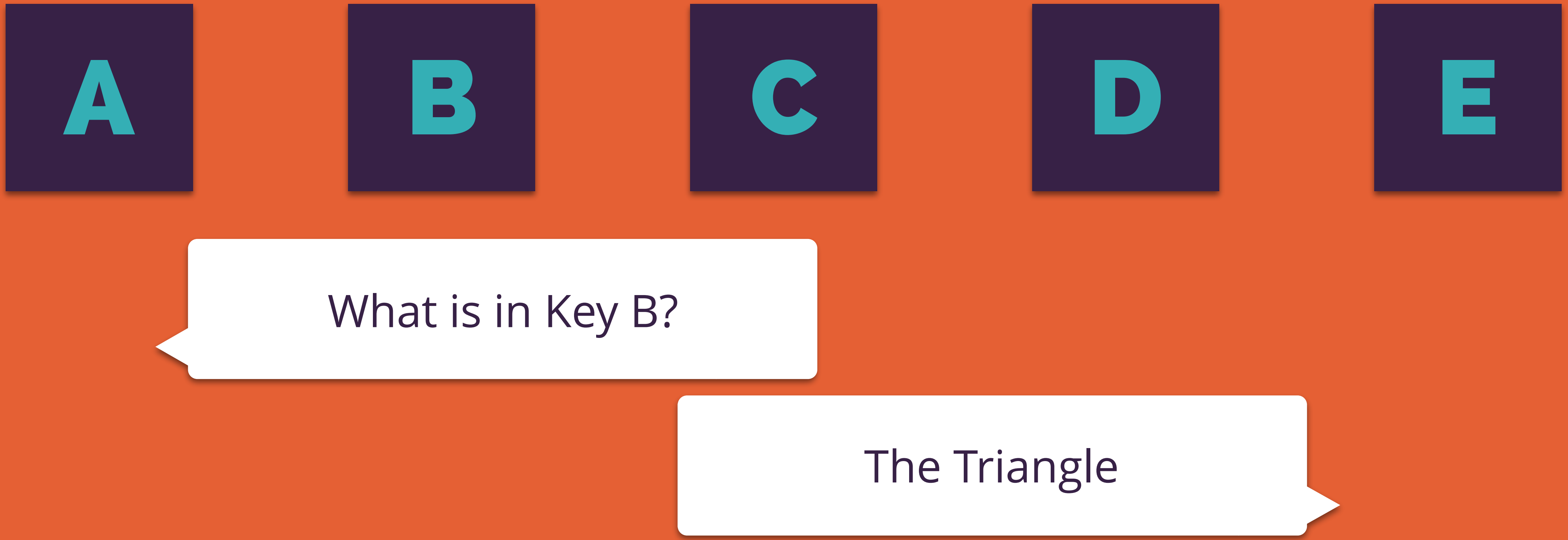
KEY
VALUE



KEY VALUE

OUR VALUES ARE HIDDEN INSIDE THE KEYS

TO FIND OUT WHAT THEY ARE WE NEED TO QUERY THEM



KEY VALUE

DYNOMITE

redis

(VOLDERMORT)

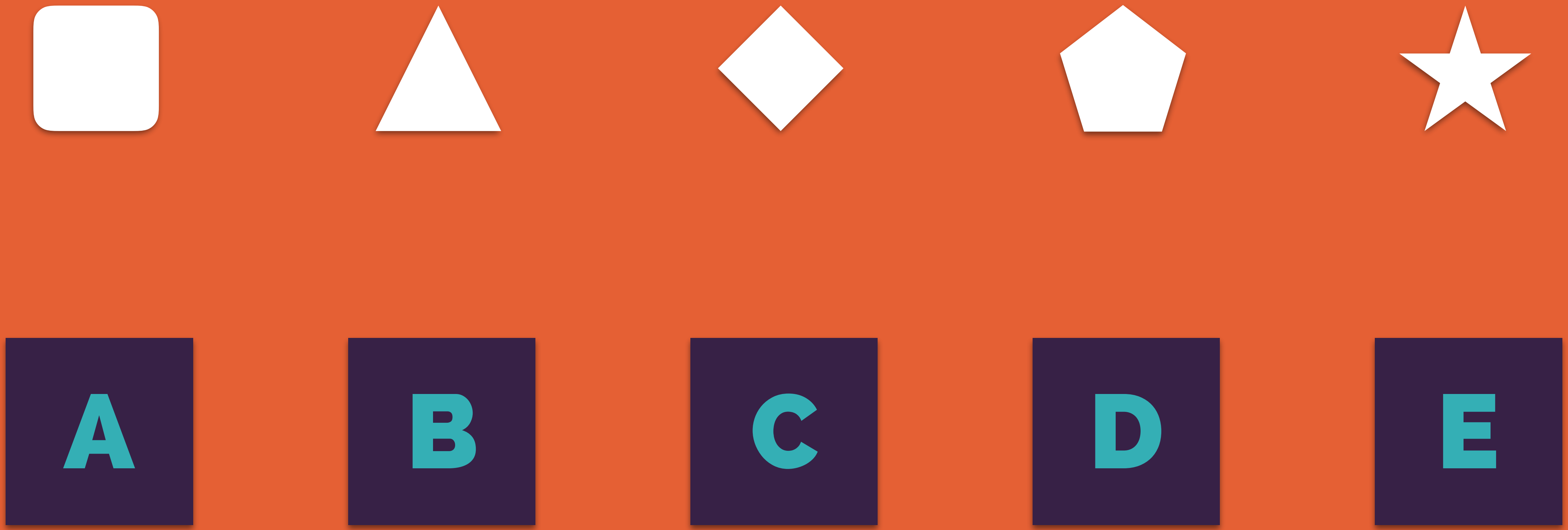
DOCUMENT STORE

DESIGNED TO WORK WITH A LOT OF DATA (BEGINNING TO NOTICE A THEME?)

VERY SIMILAR TO A KEY VALUE DATABASE

MAIN DIFFERENCE IS THAT YOU CAN ACTUALLY SEE THE VALUES

DOCUMENT STORE



DOCUMENT STORE

A
B
C
D
E
Bring me the triangles
Yes m'lord.

SIDE NOTE

REMEMBER HOW SQL DATABASES ARE LIBRARIES?

NO SQL IS MORE LIKE A BAG OF CATS!

SIDE NOTE

WE CAN ADD IN FIELDS AS AND WHEN WE NEED THEM

colour: tabby
name: Gunther

colour: ginger
name: Mylo

colour: grey
name: Ruffus
age: kitten

colour: ginger(ish)
name: Fred
age: kitten

colour: ginger(ish)
name: Quentin
legs: 3

DOCUMENT STORE

Bring me the KITTENS!

Of course m'lord.

DOCUMENT
STORE

redis

mongoDB

GRAPH DATABASE

FOCUS HERE IS ON MODELLING THE STRUCTURE OF THE DATA

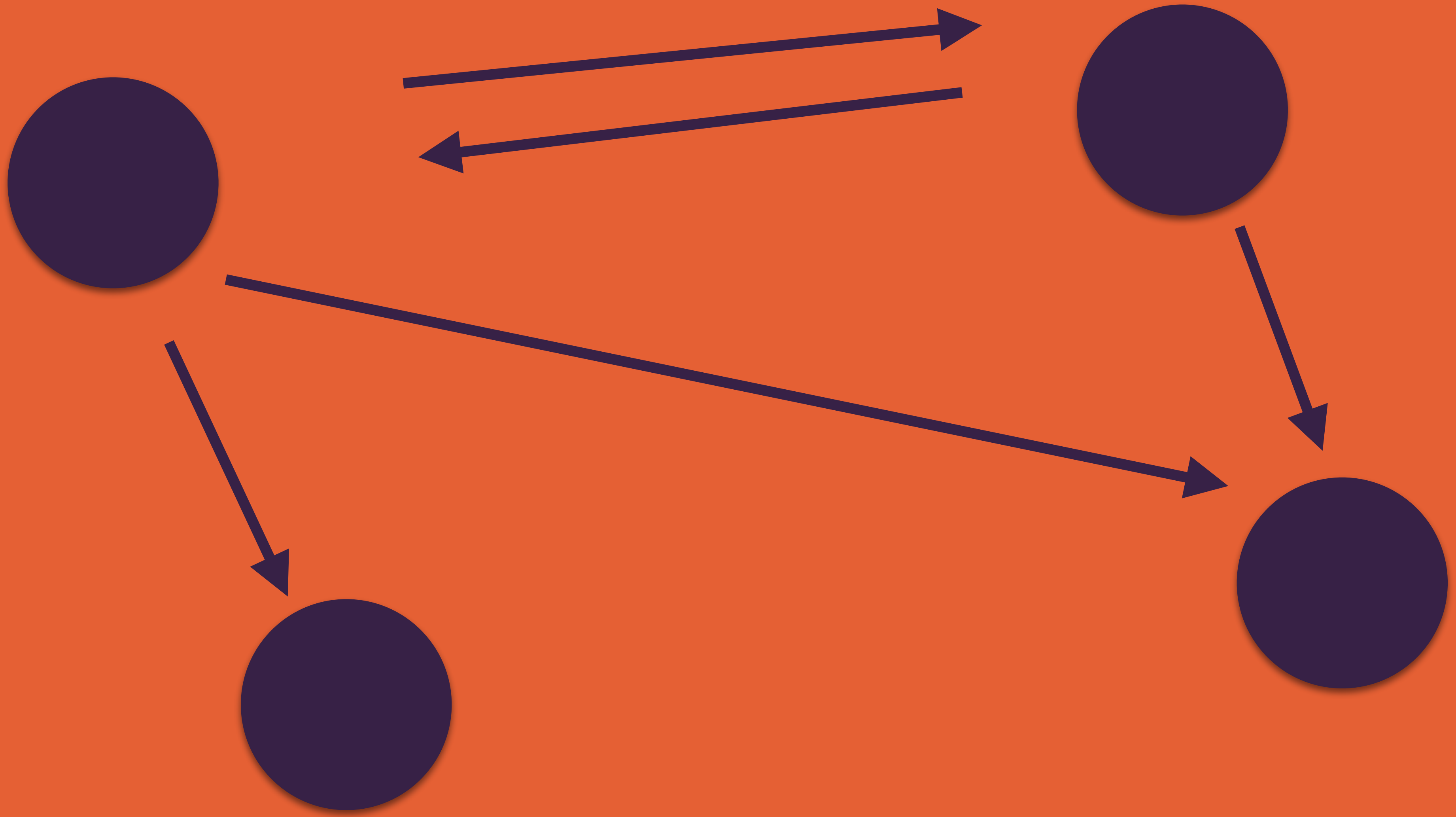
INSPIRED BY GRAPH THEORY (GO MATHS!)

SCALES REALLY WELL TO THE STRUCTURE OF THE DATA

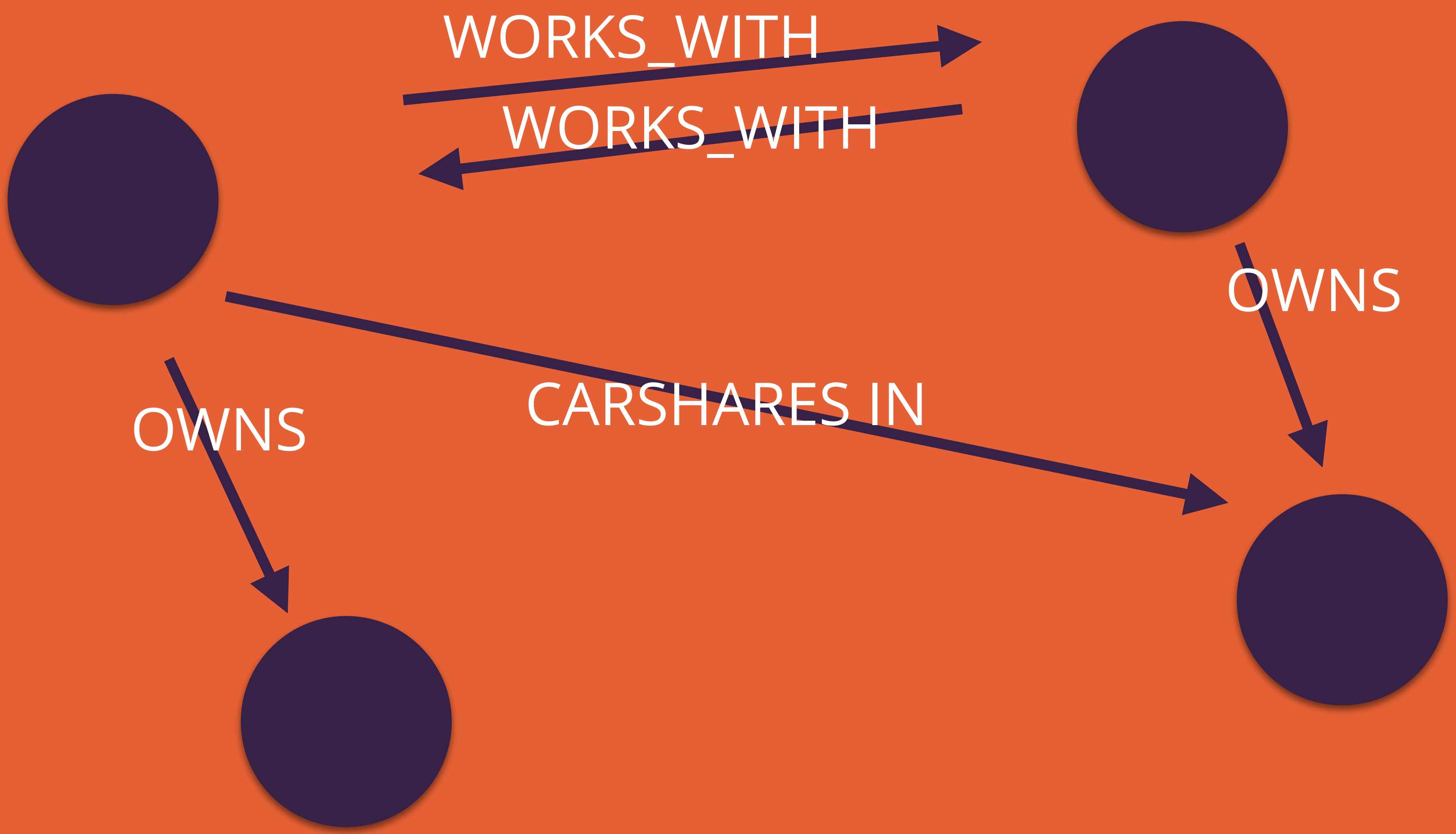
GRAPH

DATABASE

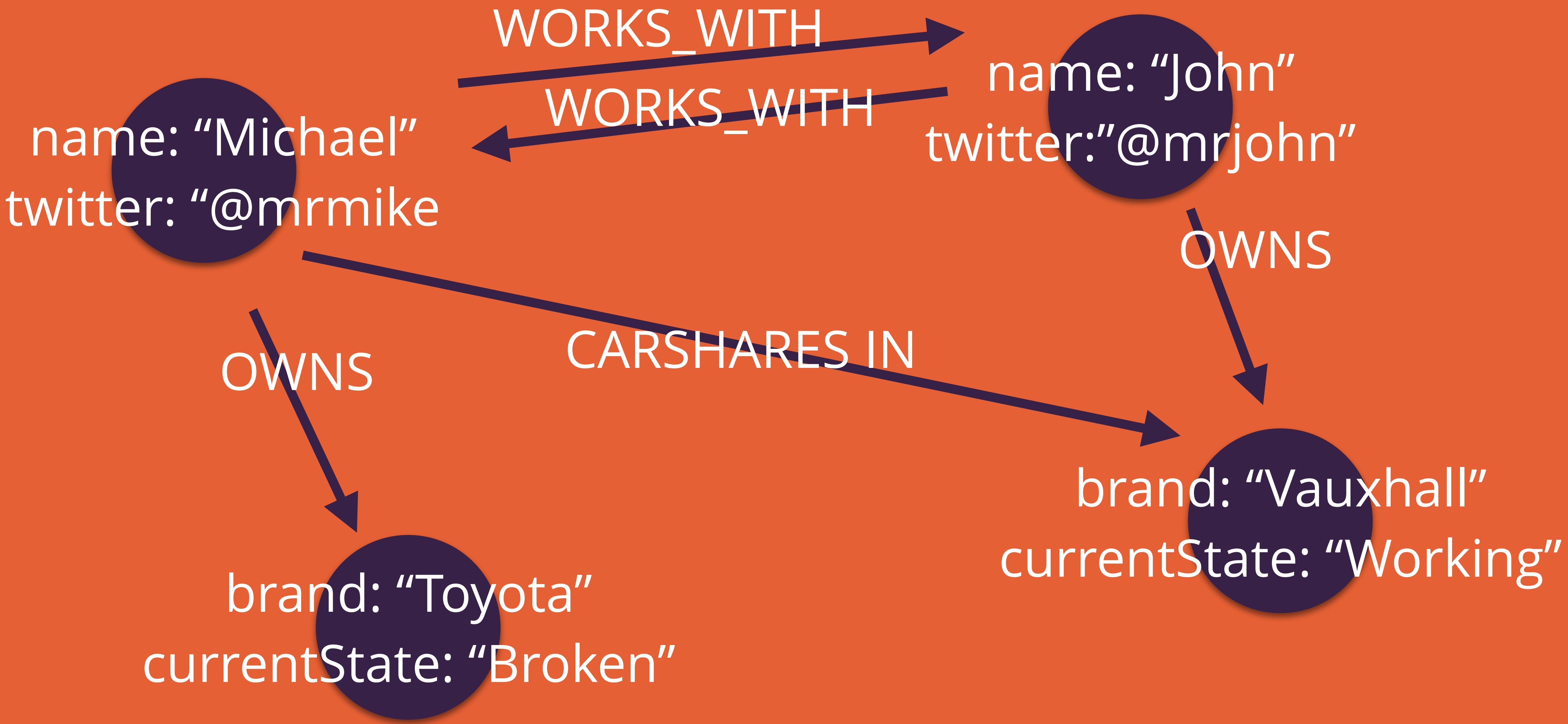
GRAPH
DATABASE



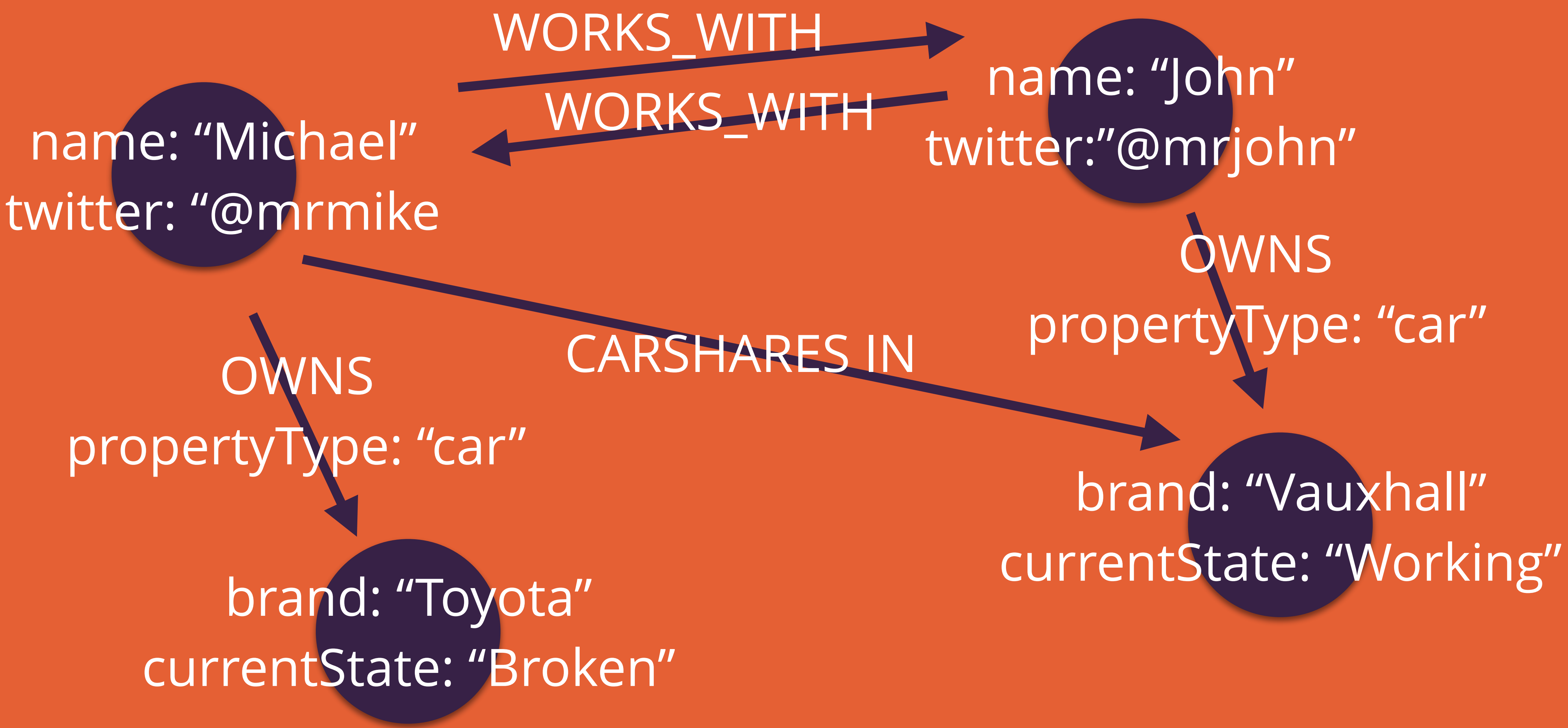
GRAPH
DATABASE



GRAPH
DATABASE

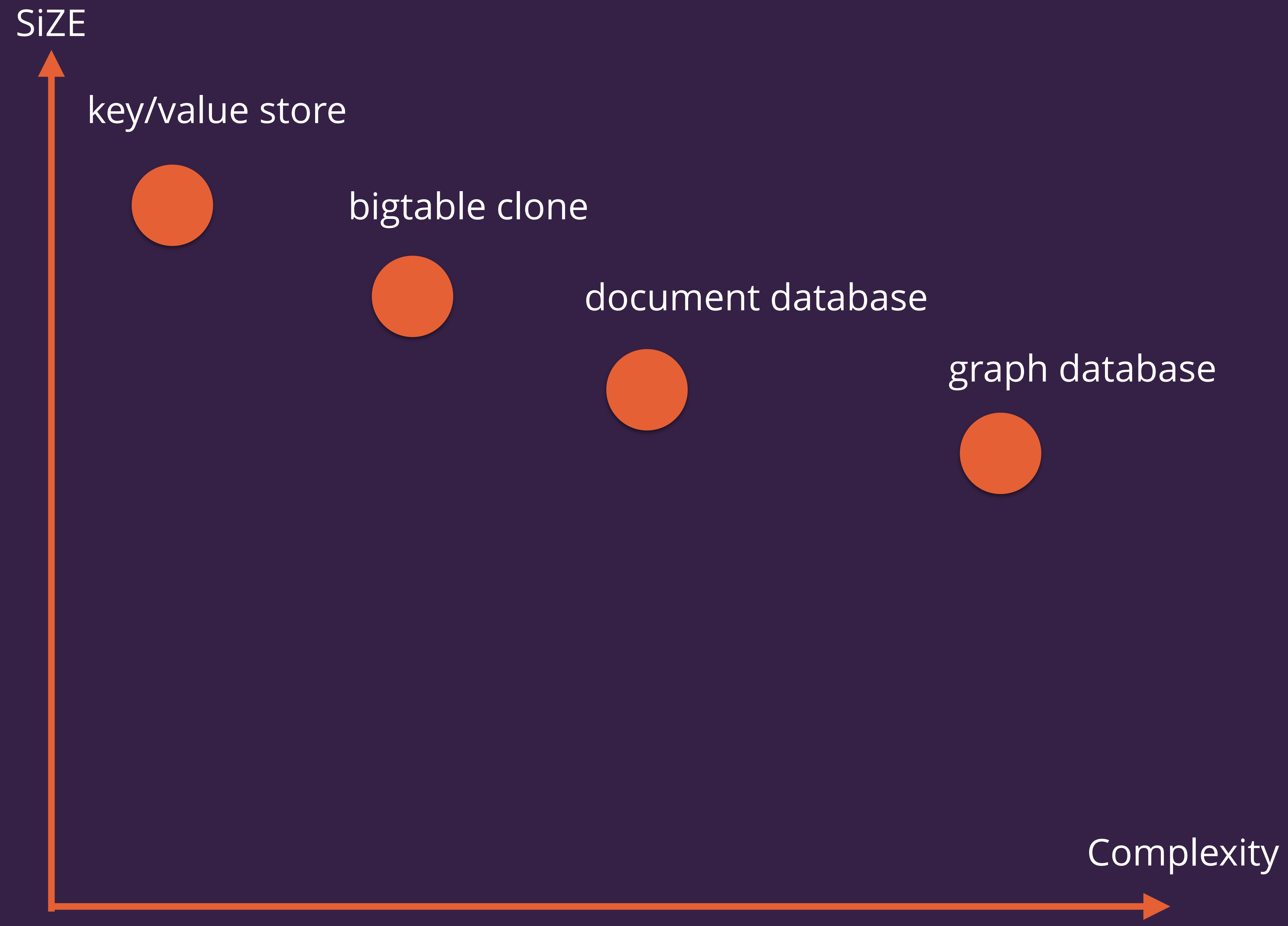


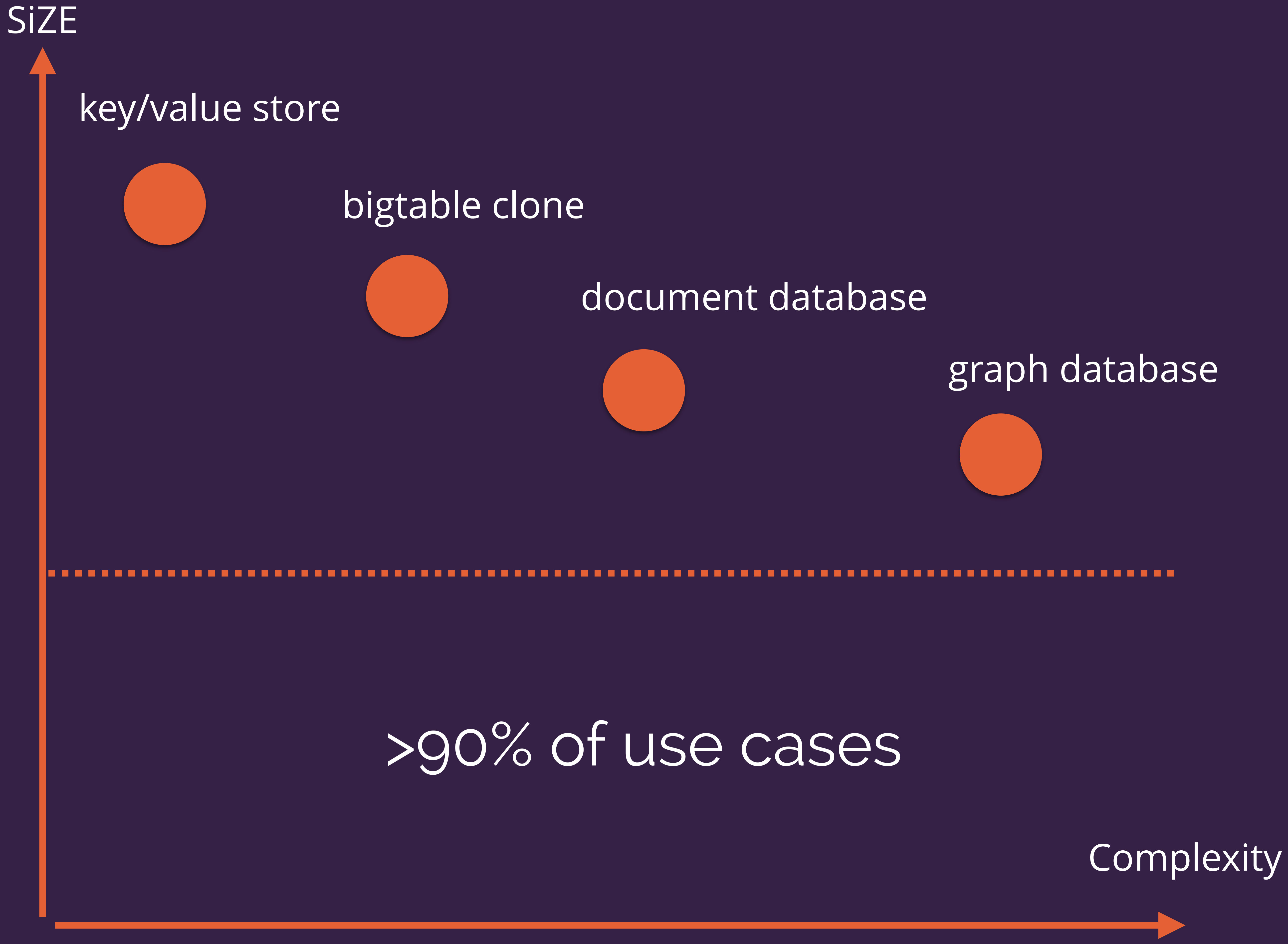
GRAPH DATABASE



GRAPH
DATABASE

neo4j


SiZE
key/value store
bigtable clone
document database
graph database
Complexity


SiZE
key/value store
bigtable clone
document database
graph database
>90% of use cases
Complexity

WHEN TO USE NOSQL AND WHEN TO USE SQL

THE BASICS

High availability and disaster recovery are a must

Understand the pros and cons of each design model

Don’t pick something just because it is new

Do you remember the zune?

Don’t pick something based JUST on performance

SQL
THE GOOD

High performance for transactions. Think ACID

Highly structured, very portable

Small amounts of data
SMALL IS LESS THAN 500GB

Supports many tables with different types of data

Can fetch ordered data

Compatible with lots of tools

SQL

ATOMICITY

CONSISTENCY

ISOLATION

DURABILITY

SQL

THE GOOD

High performance for transactions. Think ACID

Highly structured, very portable

Small amounts of data

SMALL IS LESS THAN 500GB

Supports many tables with different types of data

Can fetch ordered data

Compatible with lots of tools

SQL
THE BAD

Complex queries take a long time

The relational model takes a long time to learn

Not really scalable

Not suited for rapid development

noSQL
THE GOOD

Fits well for volatile data

High read and write throughput

In general it's faster than SQL

Scales really well

Rapid development is possible

noSQL

BASICALLY

AVAILABLE

SOFT STATE

EVENTUALLY CONSISTENT

noSQL
THE GOOD

Fits well for volatile data

High read and write throughput

In general it’s faster than SQL

Scales really well

Rapid development is possible

noSQL
THE GOOD

Key/Value pairs need to be packed/unpacked all the time

Still working on getting security for these working as well as SQL

Lack of relations from one key to another

tl;dr

SQL

works great, can’t scale for large data

noSQL

works great, doesn't fit all situations

so use both, but think about when you want to use them!

Finally

A lot of this content is loving ripped from lots of other (more impressive) presentations that are already on SlideShare - you should check them out!