NoSQL and the idea of scaling up and out Databases

What lies beyond Relational

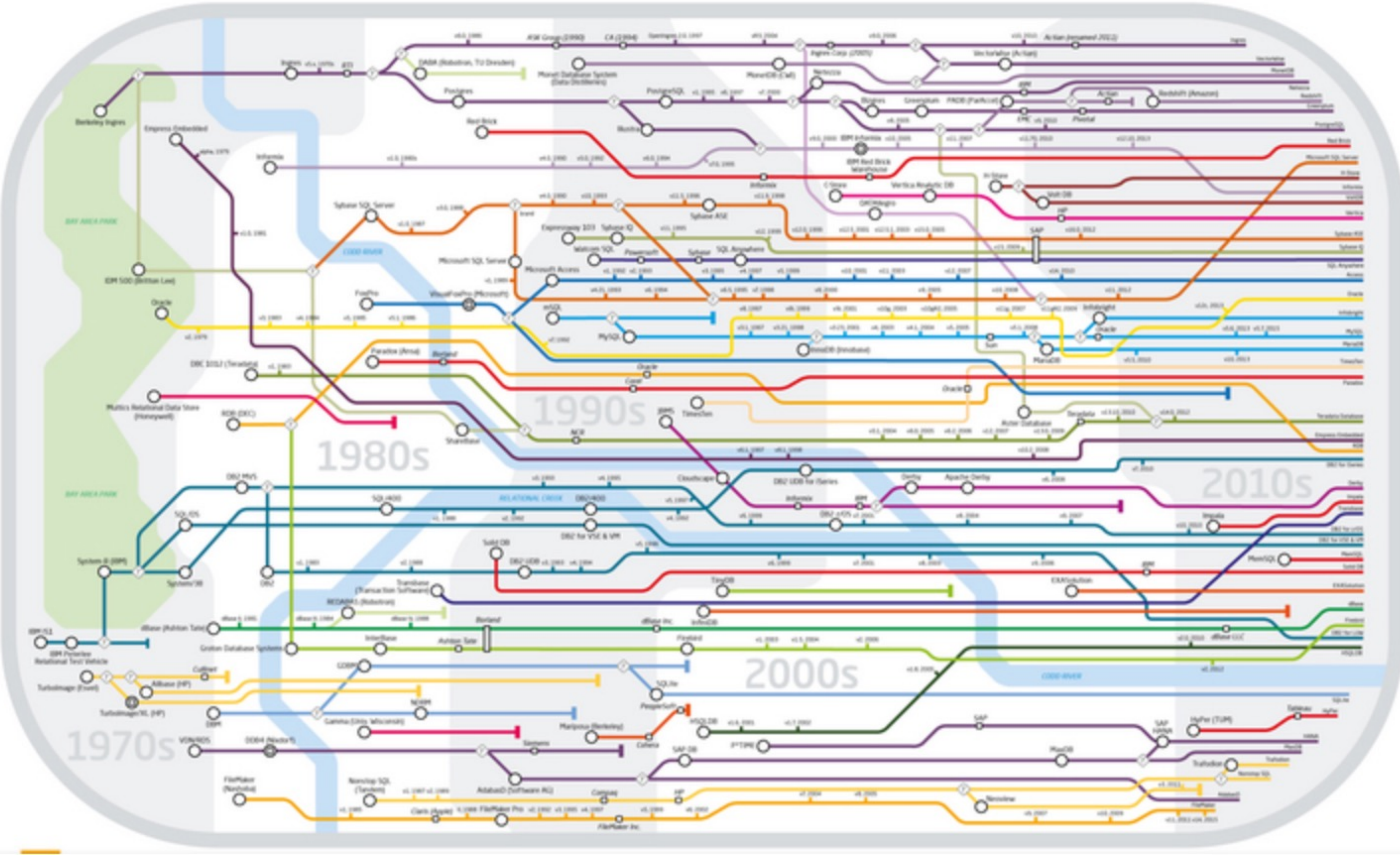
Prof. Yannis Velegrakis
Utrecht University

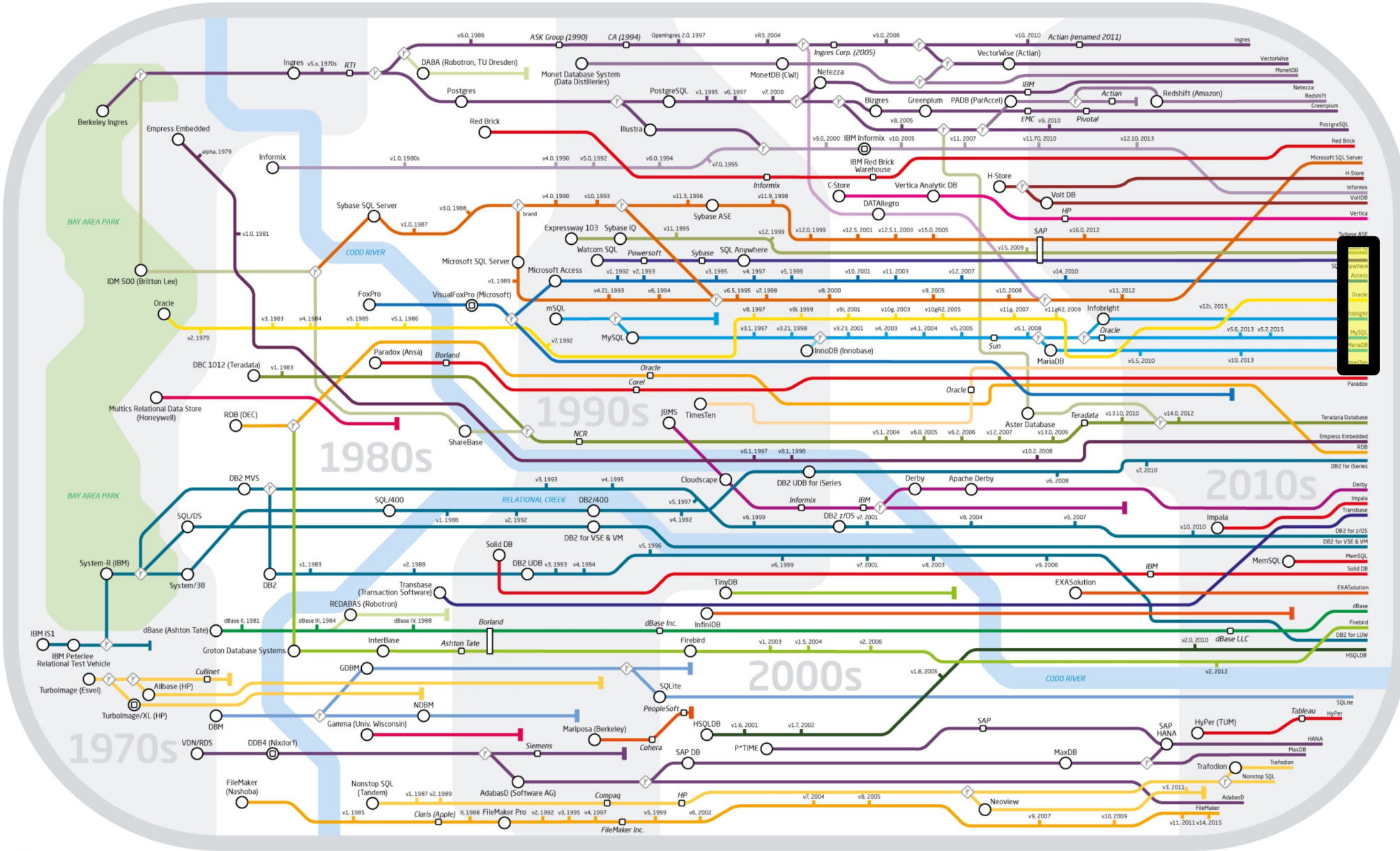
i.velegrakis@uu.nl
https://velgias.github.io

Disclaimer:

Slides courtesy of

- Martin Fowler
- Felix Naumann
- Michalis Petropoulos
- Bill Howie
- Yannis Velegrakis


1970s
1980s
1990s
2000s
2010s


1970s
1980s
1990s
2000s
2010s
BAY AREA PARK
CODD RIVER
RELATIONAL CREEK
Ingres
Berkeley Ingres
Empress Embedded
Informix
Oracle
IDM 500 (Britton Lee)
Sybase SQL Server
Microsoft SQL Server
Microsoft Access
FoxPro
VisualFoxPro (Microsoft)
mSQL
MySQL
InnoDB (Innobase)
MariaDB
Infobright
Paradox (Ansa)
Borland
DBC 1012 (Teradata)
Multics Relational Data Store (Honeywell)
RDB (DEC)
ShareBase
Oracle
Corel
NCR
JBMS
TimesTen
Aster Database
Teradata
DB2 MVS
SQL/400
SQL/DS
System-R (IBM)
System/38
DB2
DB2/400
DB2 for VSE & VM
DB2 z/OS
DB2 UDB for iSeries
Cloudscape
Derby
Apache Derby
Impala
MemSQL
Solid DB
DB2 UDB
Transbase (Transaction Software)
REDABAS (Robotron)
TinyDB
EXASolution
dBase
InfiniDB
dBase Inc.
dBase LLC
Firebird
InterBase
Ashton Tate
Groton Database Systems
dBase (Ashton Tate)
IBM IS1
IBM Peterlee Relational Test Vehicle
Cullinet
Allbase (HP)
Turbolmage (Esvel)
Turbolmage/XL (HP)
GDBM
NDBM
DBM
Gamma (Univ. Wisconsin)
Mariposa (Berkeley)
PeopleSoft
Cohera
SQLite
HSQLDB
VDN/RDS
DDB4 (Nixdorf)
Siemens
SAP DB
P*TIME
SAP HANA
HyPer (TUM)
Tableau
MaxDB
Trafodion
Nonstop SQL
AdabasD (Software AG)
Compaq
HP
Neoview
FileMaker (Nashoba)
Claris (Apple)
FileMaker Pro
FileMaker Inc.
Illustra
Red Brick
PostgreSQL
Postgres
Monet Database System (Data Distilleries)
MonetDB (CWI)
DABA (Robotron, TU Dresden)
Netezza
Bizgres
Greenplum
PADB (ParAccel)
Redshift (Amazon)
Actian
Ingres Corp. (2005)
VectorWise (Actian)
IBM Informix
IBM Red Brick Warehouse
C-Store
Vertica Analytic DB
DATAllegro
H-Store
Volt DB
HP
SAP
Sybase ASE
Sybase IQ
Expressway 103
Watcom SQL
Powersoft
Sybase
SQL Anywhere
Solid DB
Derby
Impala
Transbase
DB2 for z/OS
DB2 for VSE & VM
MemSQL
Solid DB
EXASolution
dBase
Firebird
DB2 for LUW
HSQLDB
Teradata Database
Empress Embedded
RDB
DB2 for iSeries
Paradox
Sybase ASE
SQL Anywhere
Access
Oracle
Infobright
MySQL
MariaDB
InnoDB
Red Brick
Microsoft SQL Server
H-Store
Informix
VoltDB
Vertica
Ingres
VectorWise
MonetDB
Netezza
Redshift
Greenplum
PostgreSQL


“DATA IS THE NEW GOLD”

The Need for Data Storage

How do we store data?

Why would I want a database?

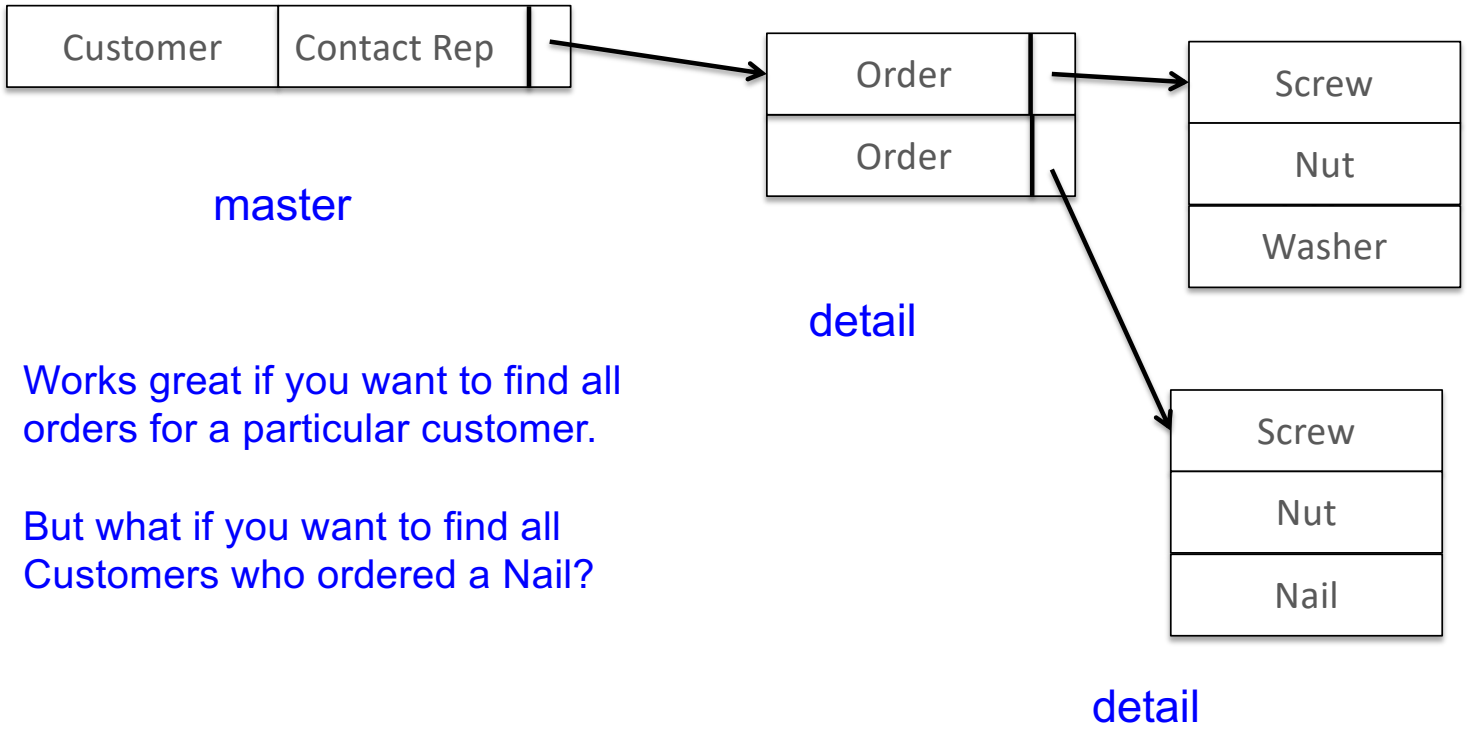
What problem do they solve?

1. Sharing
   Support concurrent access by multiple readers and writers
2. Data Model Enforcement
   Make sure all applications see clean, organized data
3. Scale
   Work with datasets too large to fit in memory
4. Flexibility
   Use the data in new, unanticipated ways

What is a Data Model?

1. Structures
   - rows and columns?
   - nodes and edges?
   - key-value pairs?
   - a sequence of bytes?
2. Constraints
   - all rows must have the same number of columns
   - all values in one column must have the same type
   - a child cannot have two parents
3. Operations
   - find the value of key x
   - find the rows where column “lastname” is “Jordan”
   - get the next N bytes

Historical Example: Hierarchical Databases



Works great if you want to find all orders for a particular customer.

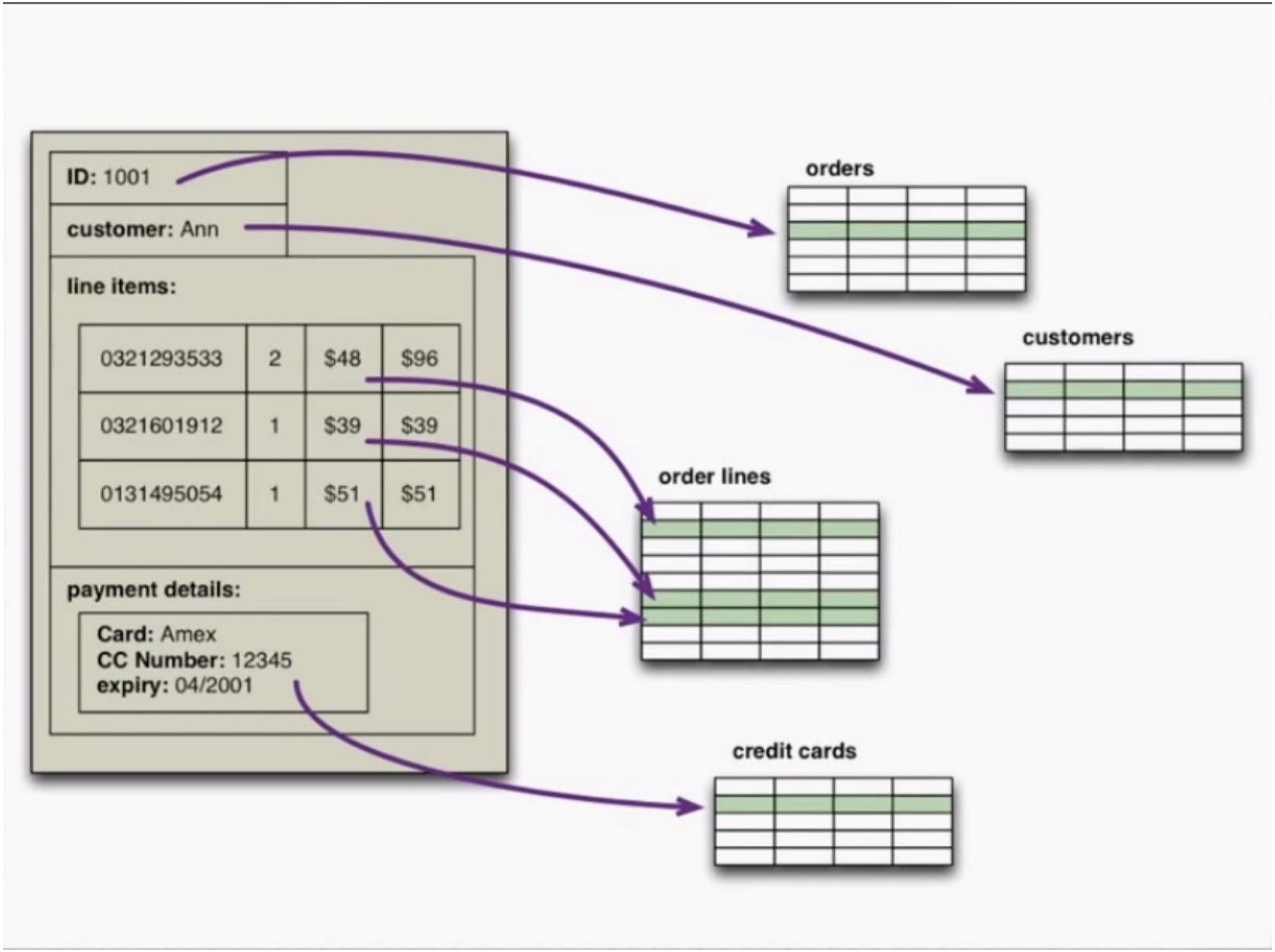
But what if you want to find all Customers who ordered a Nail?

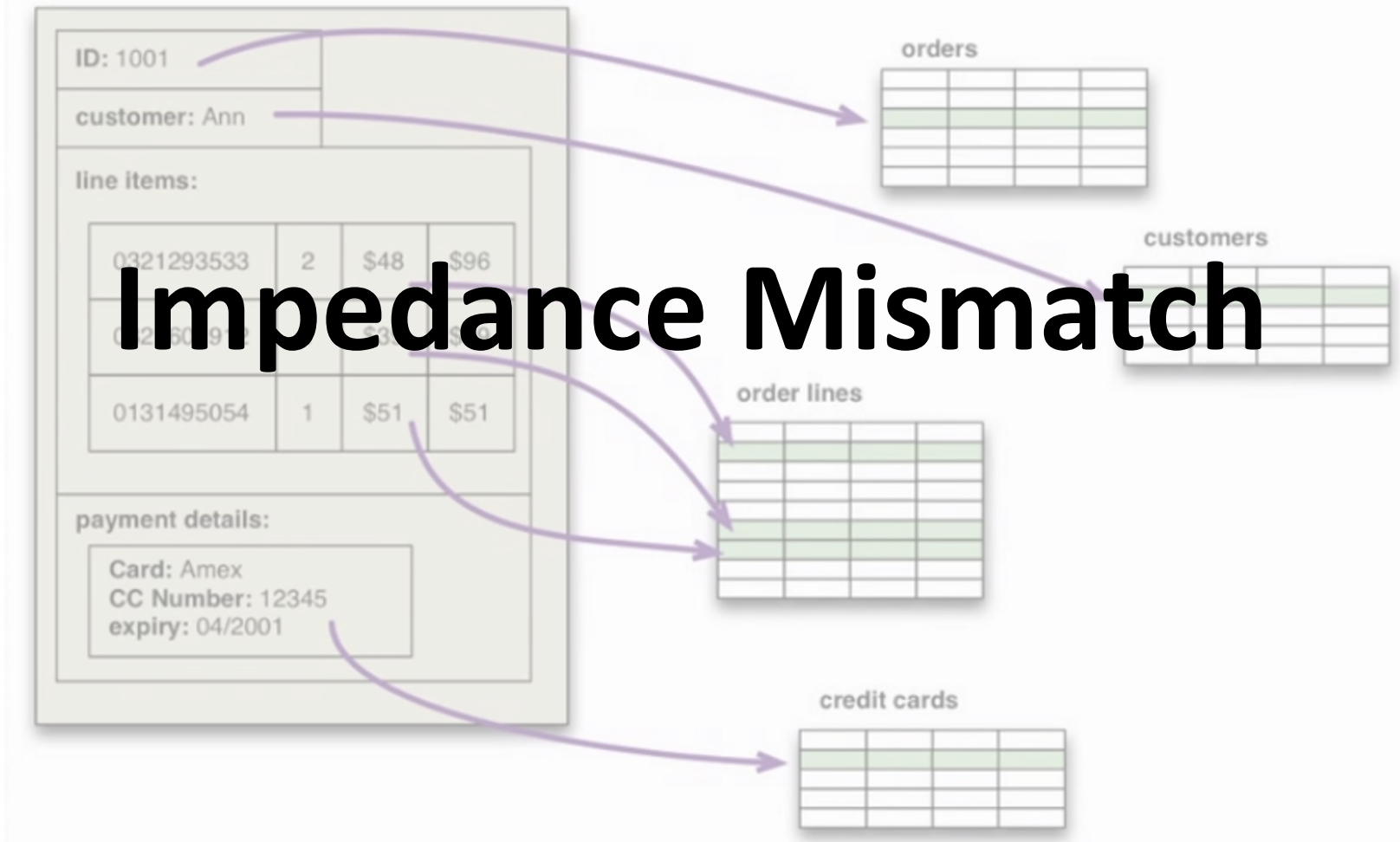
Relational Databases (Codd 1970)

- Everything is a table
- Every row in a table has the same columns
- Relationships are implicit: no pointers

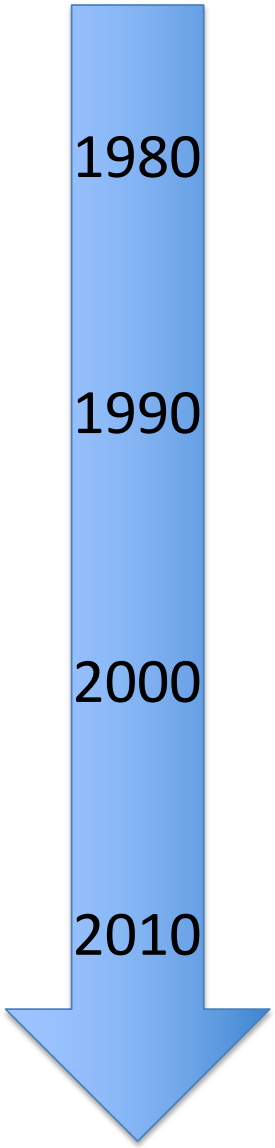
Course	Student Id
CSE 344	223…
CSE 344	244…
CSE 514	255..
CSE 514	244…

Student Id	Student Name
223…	Jane
244…	Joe
255..	Susan

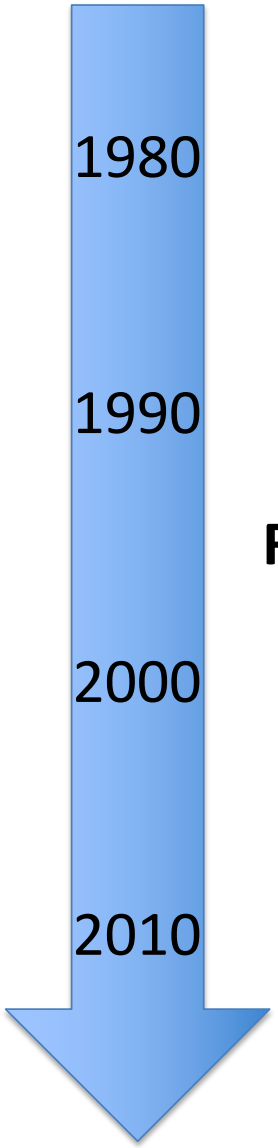

ID: 1001
customer: Ann
line items:
0321293533
2
$48
$96
0321601912
1
$39
$39
0131495054
1
$51
$51
payment details:
Card: Amex
CC Number: 12345
expiry: 04/2001
orders
customers
order lines
credit cards



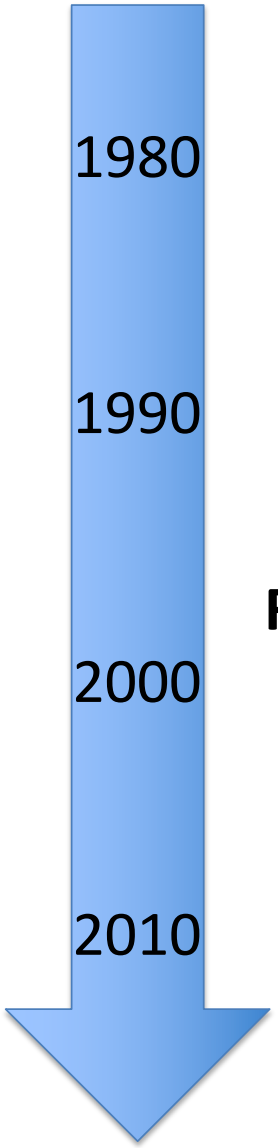
Impedance Mismatch



Rise of the Relational Databases

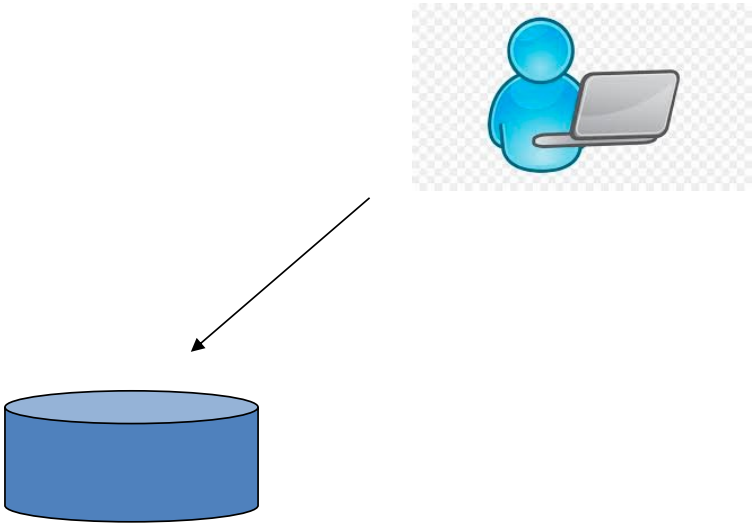


Rise of the Object Databases



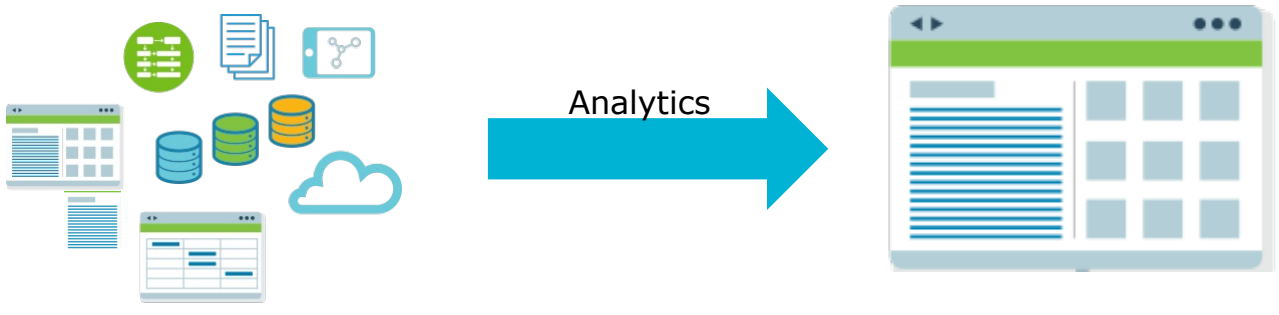
Relational Dominance

Data Management: The Old Model



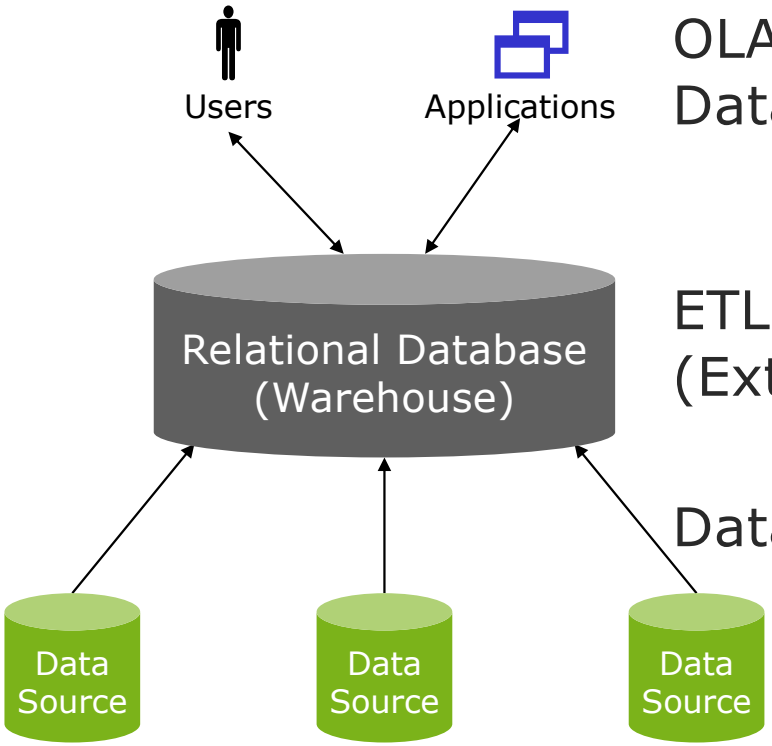
Data Analytics: The New Model

Extracting Value from Data



The Need for Integration

Data Warehouse Architecture



OLAP / Decision Support
Data Cubes / Data Mining

ETL Tools
(Extract-Transform-Load)

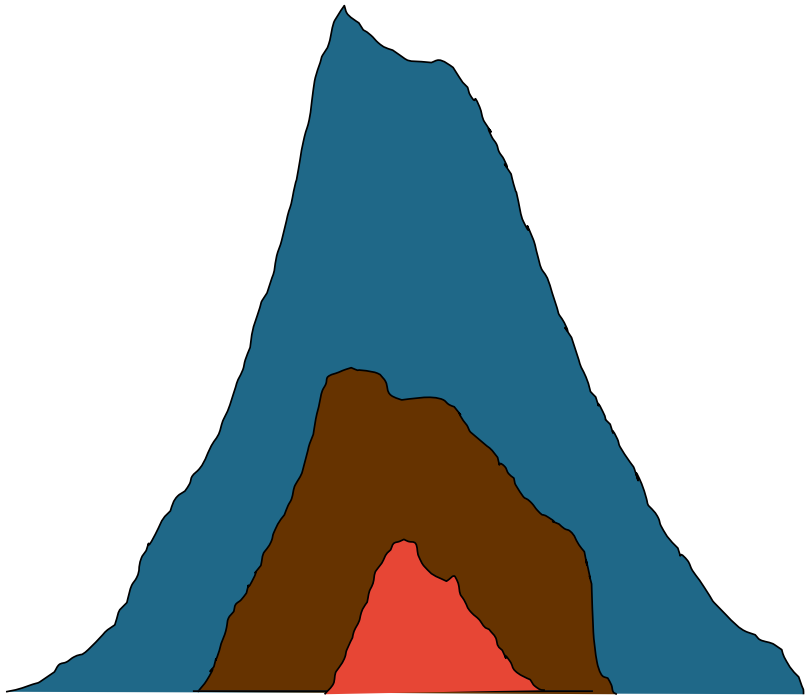
Data Cleaning

The Need for Scaling Up

Machine Generated Data

Internet Users

Employees / Data Engineers



2020 This Is What Happens In An Internet Minute

2021 This Is What Happens In An Internet Minute

A DAY IN DATA

The exponential growth of data is undisputed, but the numbers behind this explosion – fuelled by internet of things and the use of connected devices – are hard to comprehend, particularly when looked at in the context of one day

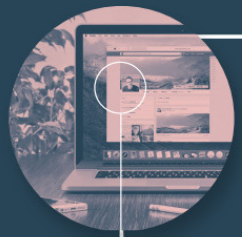
DEMYSTIFIYING DATA UNITS

From the more familiar 'bit' or 'megabyte', larger units of measurement are more frequently being used to explain the masses of data

Unit		Value	Size
b	bit	0 or 1	1/8 of a byte
B	byte	8 bits	1 byte
KB	kilobyte	1,000 bytes	1,000 bytes
MB	megabyte	$1{,}000^2$ bytes	1,000,000 bytes
GB	gigabyte	$1{,}000^3$ bytes	1,000,000,000 bytes
TB	terabyte	$1{,}000^4$ bytes	1,000,000,000,000 bytes
PB	petabyte	$1{,}000^5$ bytes	1,000,000,000,000,000 bytes
EB	exabyte	$1{,}000^6$ bytes	1,000,000,000,000,000,000 bytes
ZB	zettabyte	$1{,}000^7$ bytes	1,000,000,000,000,000,000,000 bytes
YB	yottabyte	$1{,}000^8$ bytes	1,000,000,000,000,000,000,000,000 bytes

*A lowercase "b" is used as an abbreviation for bits, while an uppercase "B" represents bytes.

500m
tweets are sent every day
Twitter



4PB
of data created by Facebook, including
350m photos
100m hours of video watch time
Facebook Research

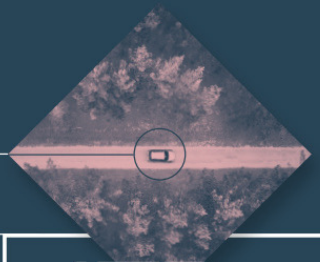
463EB
of data will be created every day by 2025
IDC

95m
photos and videos are shared on Instagram
Instagram Business

294bn
billion emails are sent
Radicati Group

320bn
emails to be sent each day by 2021

306bn
emails to be sent each day by 2020



65bn
messages sent over WhatsApp and two billion minutes of voice and video calls made
Facebook

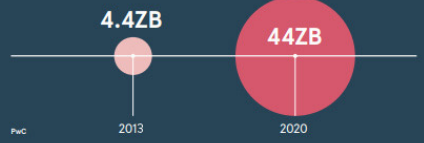
3.9bn
people use emails

4TB
of data produced by a connected car
Intel

Smart insights

28PB
to be generated from wearable devices by 2020
Statista

ACCUMULATED DIGITAL UNIVERSE OF DATA

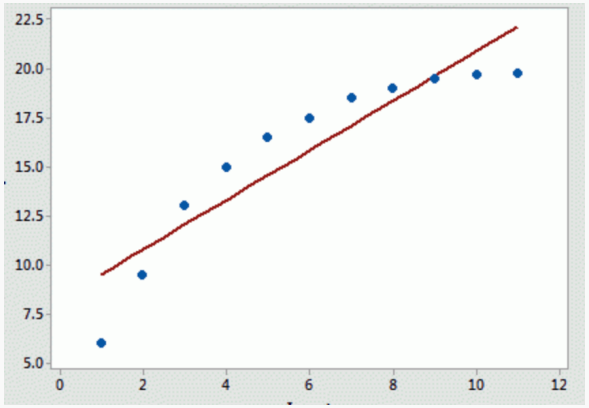


PwC

RACONTEUR

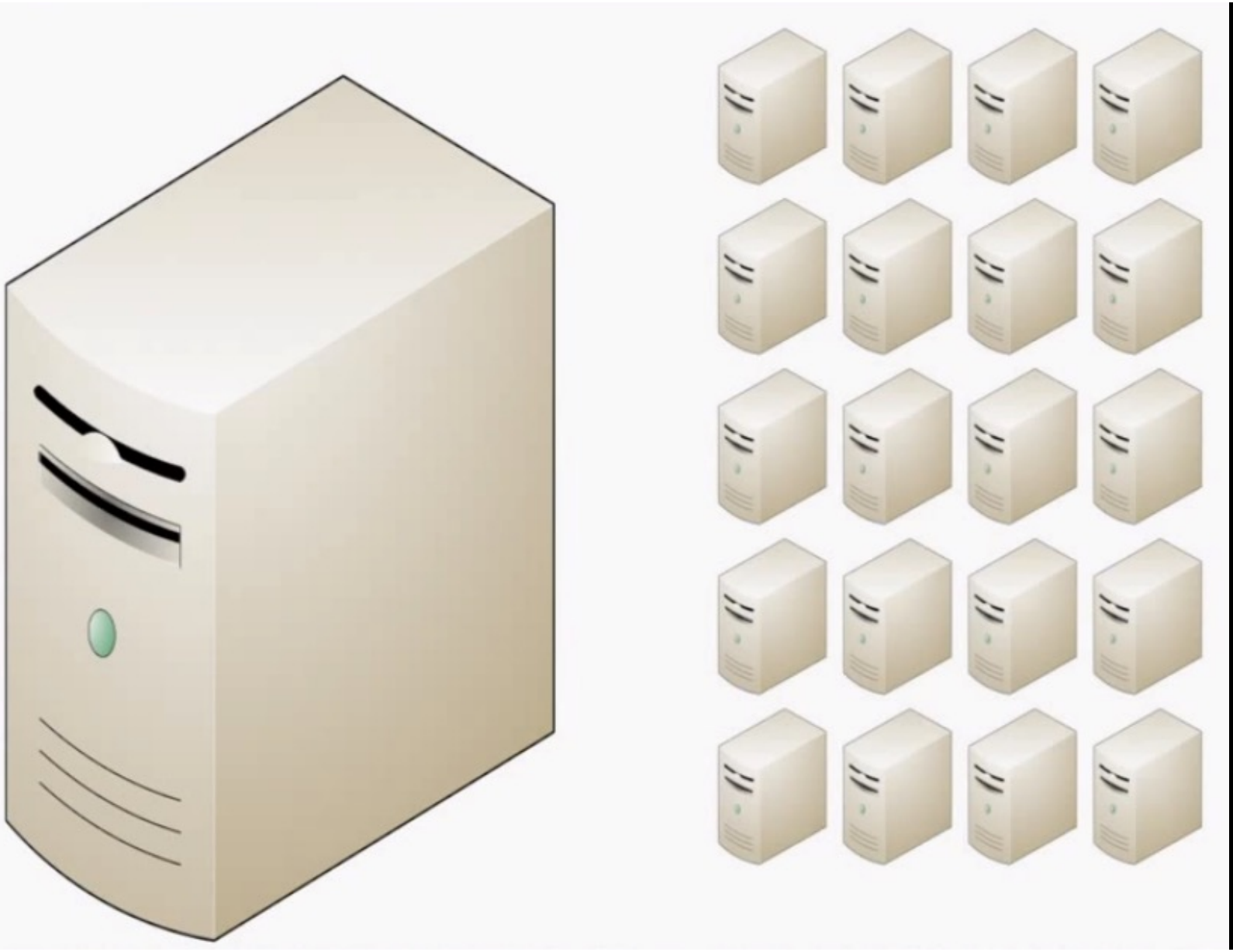

BIG DATA

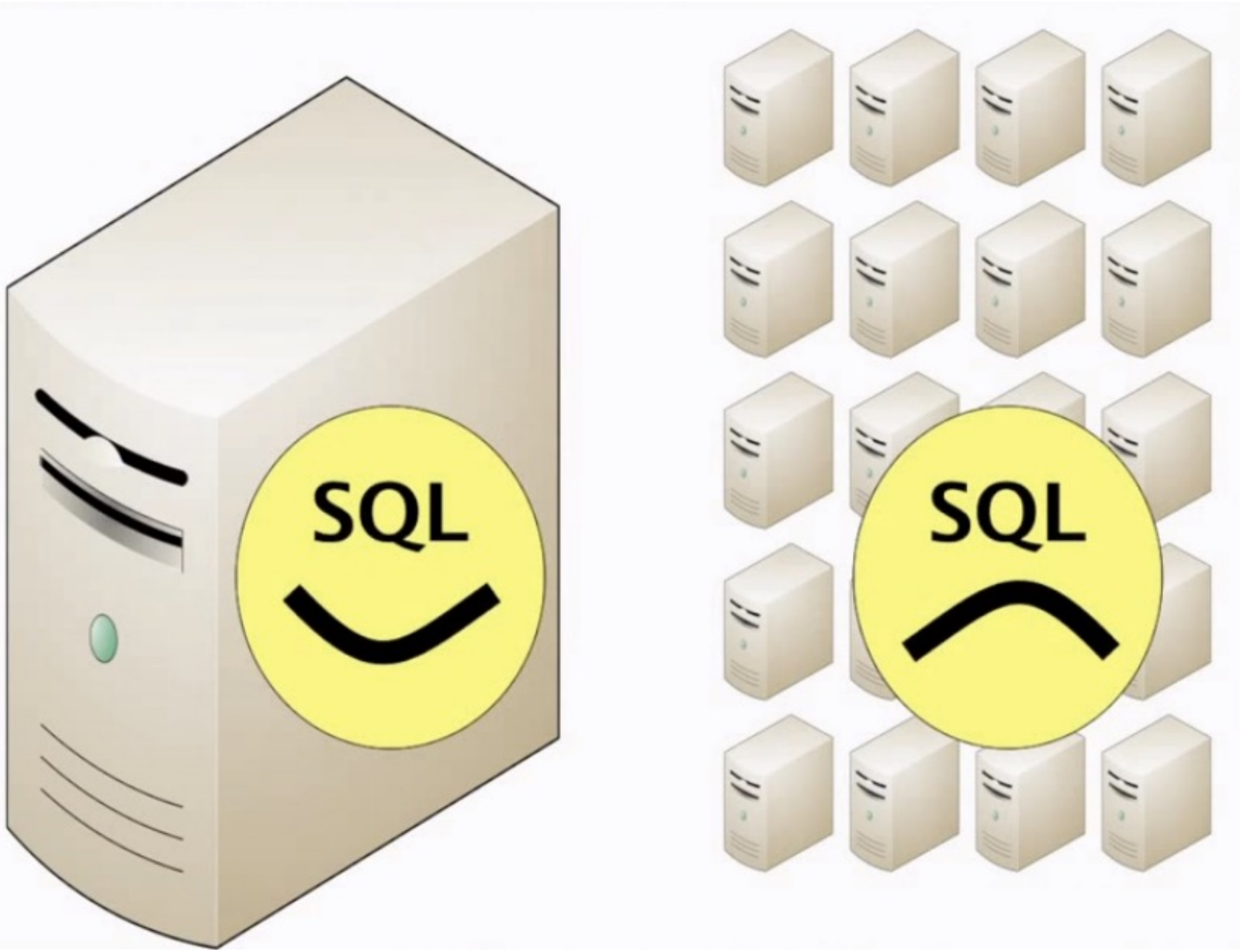

Lots of
Traffic


22.5
20.0
17.5
15.0
12.5
10.0
7.5
5.0
0
2
4
6
8
10
12

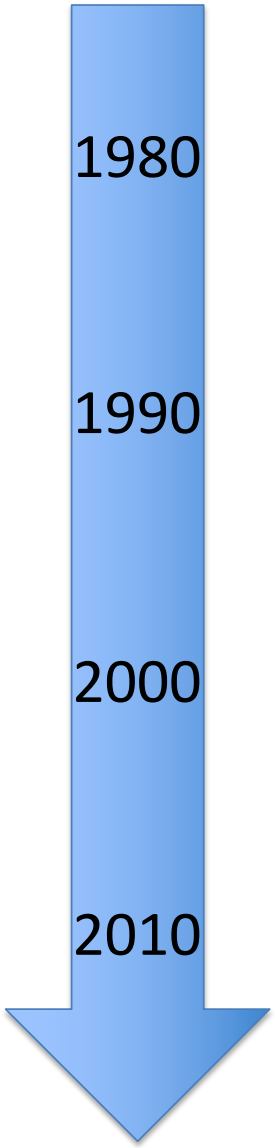

Google


amazon.com




SQL
SQL


Google
Bigtable
amazon.com
Dynamo



NoSQL

The Birth of NoSQL

Eventbrite Contact Last.fm for event and ticket information. Email Share Tweet Like

This event has ended! View current events hosted by Last.fm

NOSQL meetup

Thursday, June 11, 2009 from 10:00 AM to 5:00 PM (PT)
San Francisco, CA

Ticket Information

TYPE	REMAINING	END		QUANTITY
Free ticket	**Sold Out**	Ended	Free	**Sold Out**

Share this! Share Tweet Like Be the first of your friends to like this.

Event Details

Introduction

This meetup is about "open source, distributed, non relational databases".

Have you run into limitations with traditional relational databases? Don't mind trading a query language for scalability? Or perhaps you just like shiny new things to try out? Either way this meetup is for you.

Join us in figuring out why these newfangled Dynamo clones and BigTables have become so popular lately. We have gathered presenters from the most interesting projects around to give us all an introduction to the field.

Preliminary schedule

When & Where

CBS interactive, Magma room
235 Second Street
San Francisco, CA 94105

Thursday, June 11, 2009 from 10:00 AM to 5:00 PM (PT)

Characteristics of NoSQL

non-relational

open-source

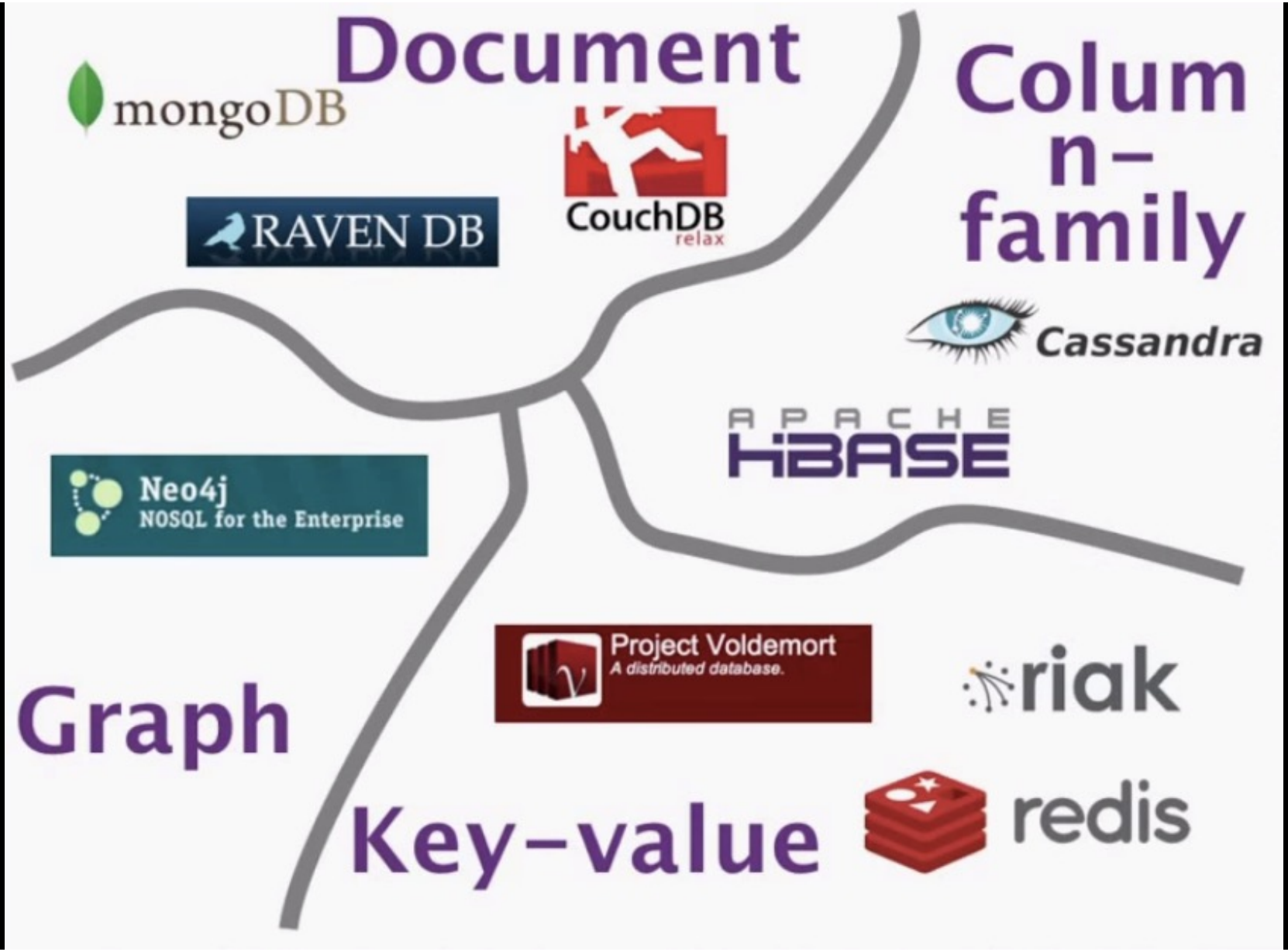
cluster-friendly

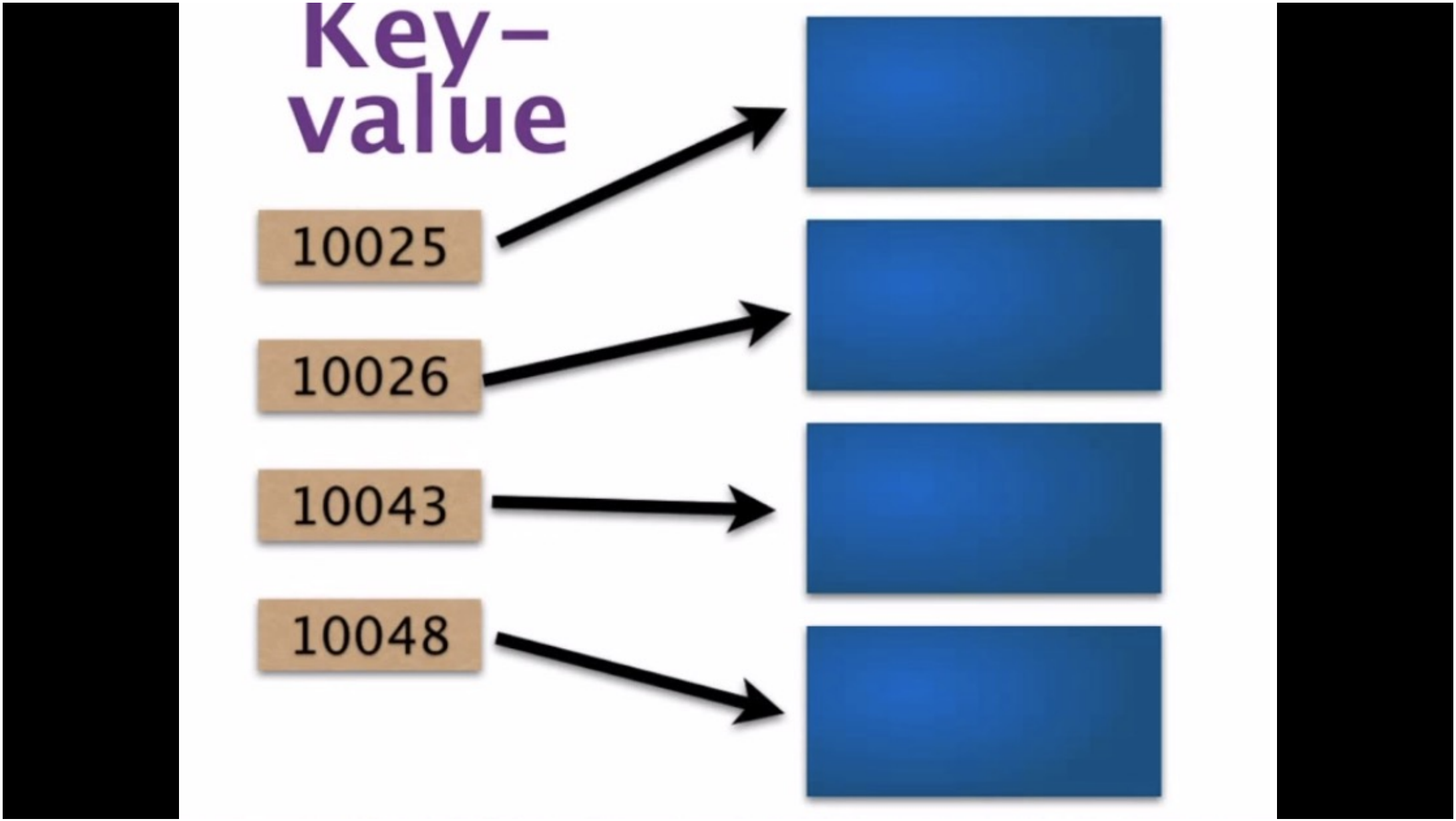
21st Century Web

schema-less

Data Model


mongoDB
RAVEN DB
CouchDB
relax
Cassandra
APACHE
HBASE
Neo4j
NOSQL for the Enterprise
Project Voldemort
A distributed database.
riak
redis

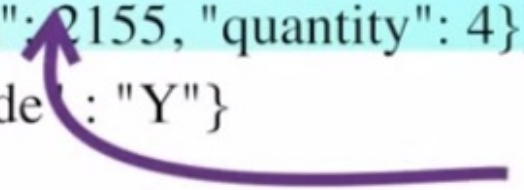

Document
mongoDB
RAVEN DB
CouchDB
relax
Colum
n-
family
Cassandra
APACHE
HBASE
Neo4j
NOSQL for the Enterprise
Project Voldemort
A distributed database.
riak
Graph
Key-value
redis


Key–
value
10025
10026
10043
10048

Document

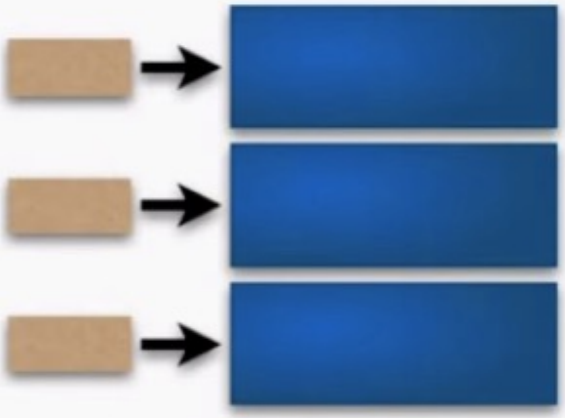
```
{"id": 1001,
"customer_id": 7231,
"line-itmes": [
{"product_id": 4555, "quantity": 8},
{"product_id": 7655, "quantity": 4}, {"product_id": 8755,
```

```
{"id": 1002,
"customer_id": 9831,
"line-itmes": [
{"product_id": 4555, "quantity": 3},
{"product_id": 2155, "quantity": 4}],
"discount-code": "Y"}
```



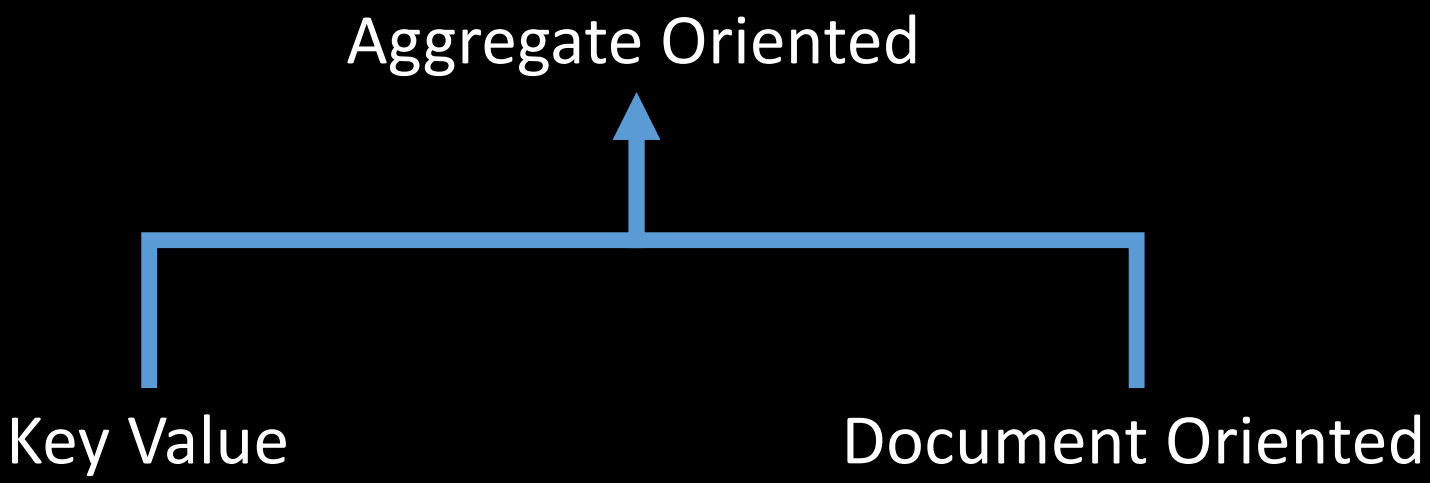
no schema

Key-Value

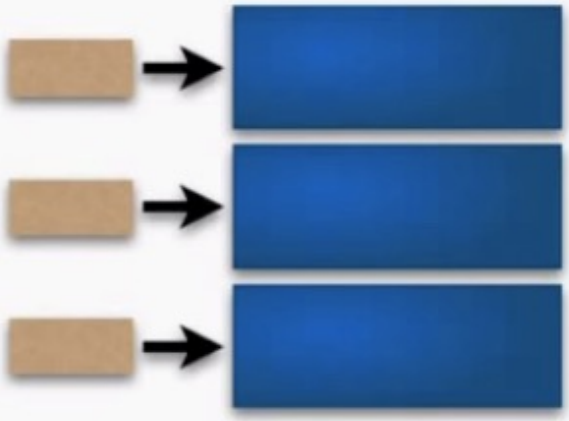


Document

```
{"id": 1001,
  {"id": 1002,
     "customer_id": 7231,
     "line-itmes": [
        {"product_id": 4555, "quantity": 8},
        {"product_id": 7655, "quantity": 4},
}       {"product_id": 8755, "quantity": 3}]
  }
```


Aggregate Oriented
Key Value
Document Oriented

Key-Value

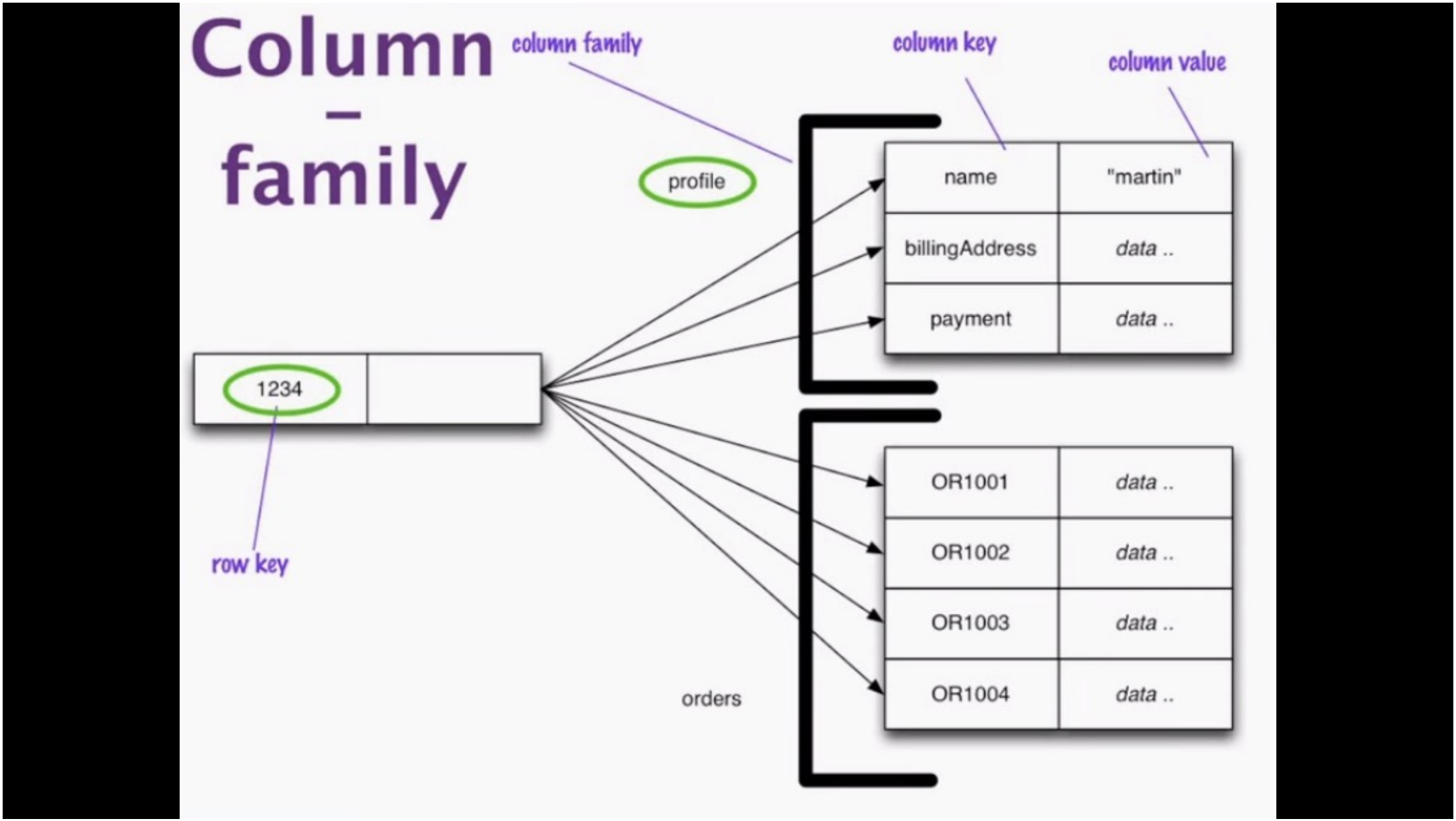


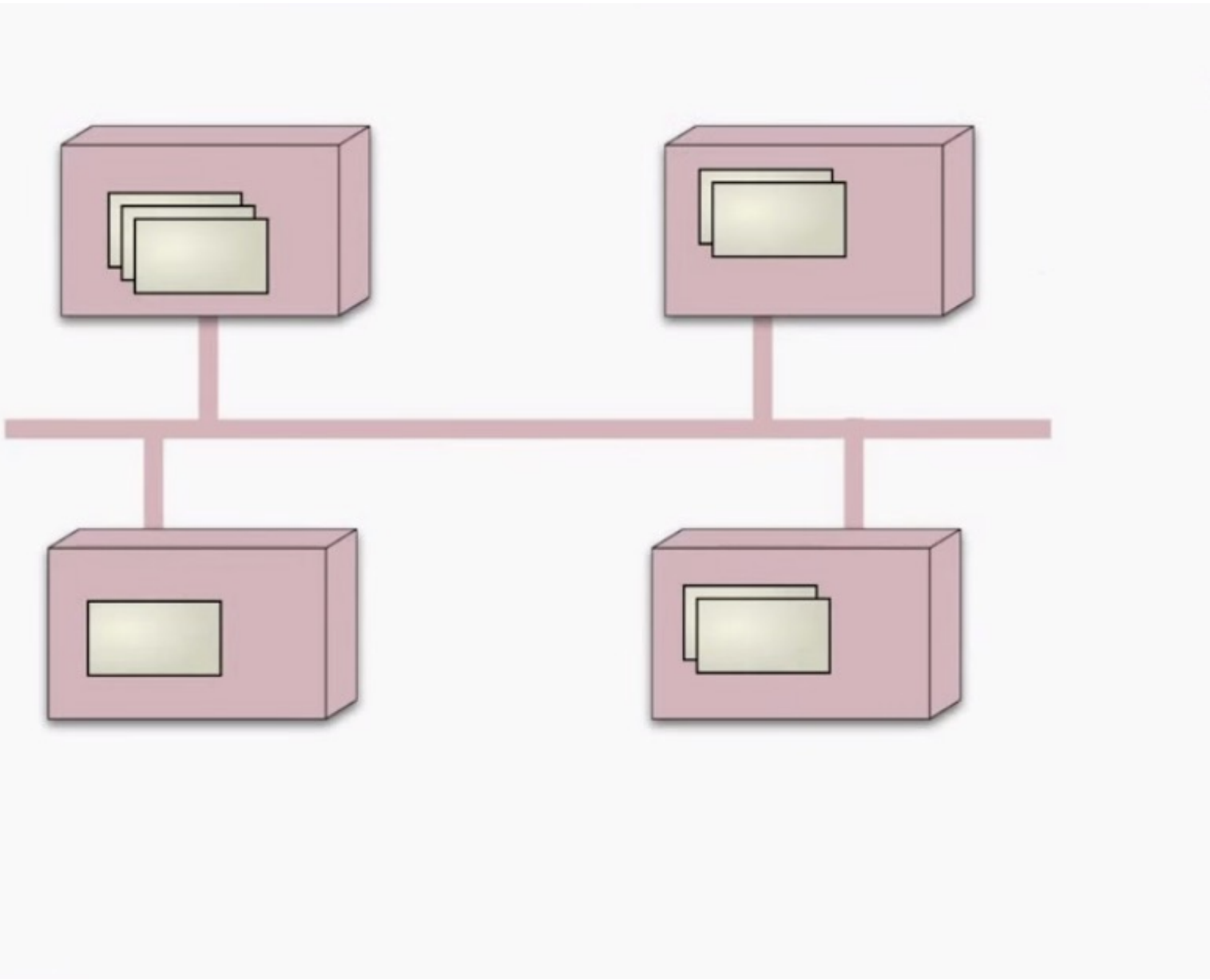
Document

```
{"id": 1001,
  {"id": 1001,
     "customer_id": 7231,
     "line-itmes": [
        {"product_id": 4555, "quantity": 8},
        {"product_id": 7655, "quantity": 4},
        {"product_id": 8755, "quantity": 3}]}
```

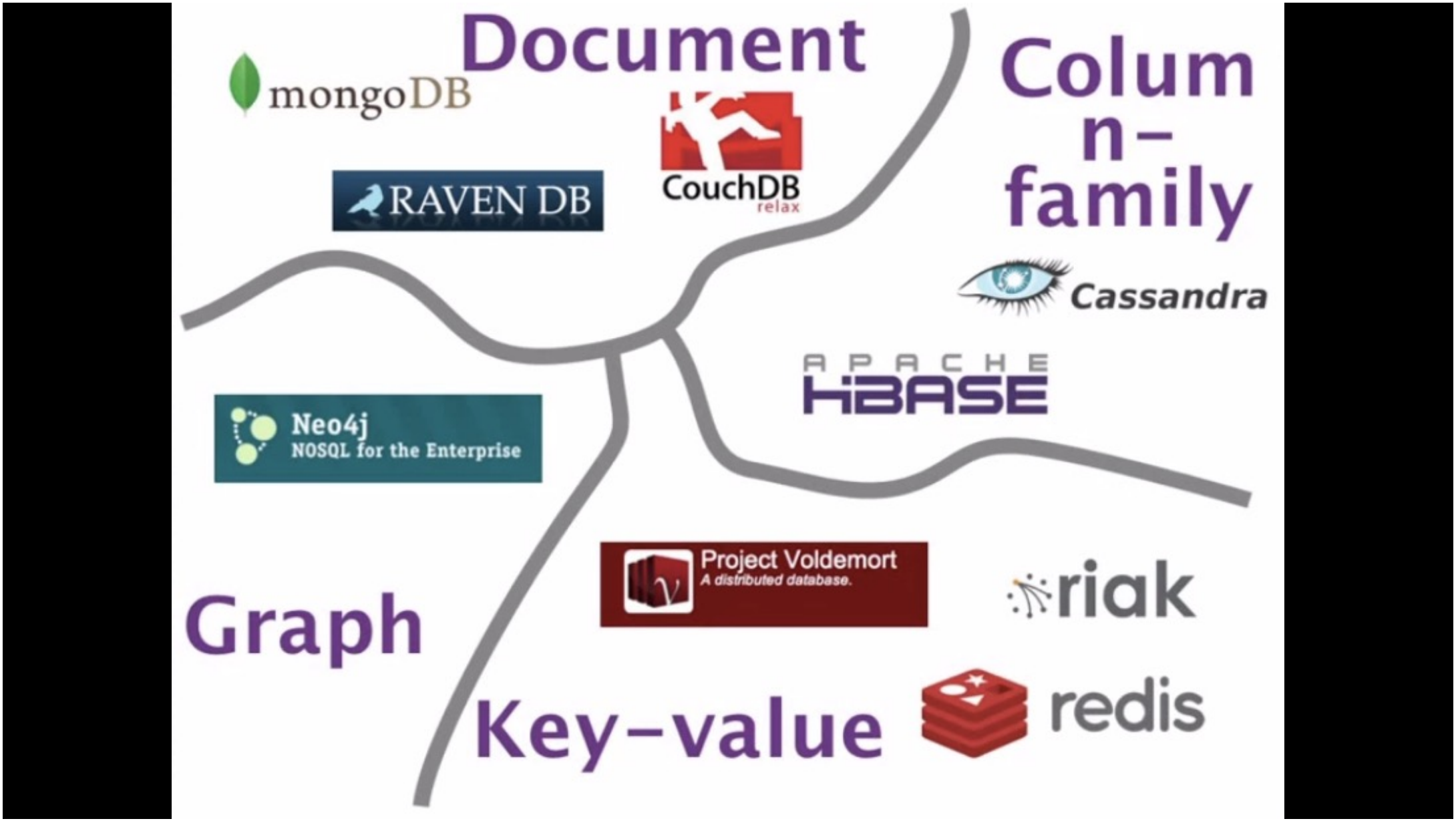
Value ==
Aggregate

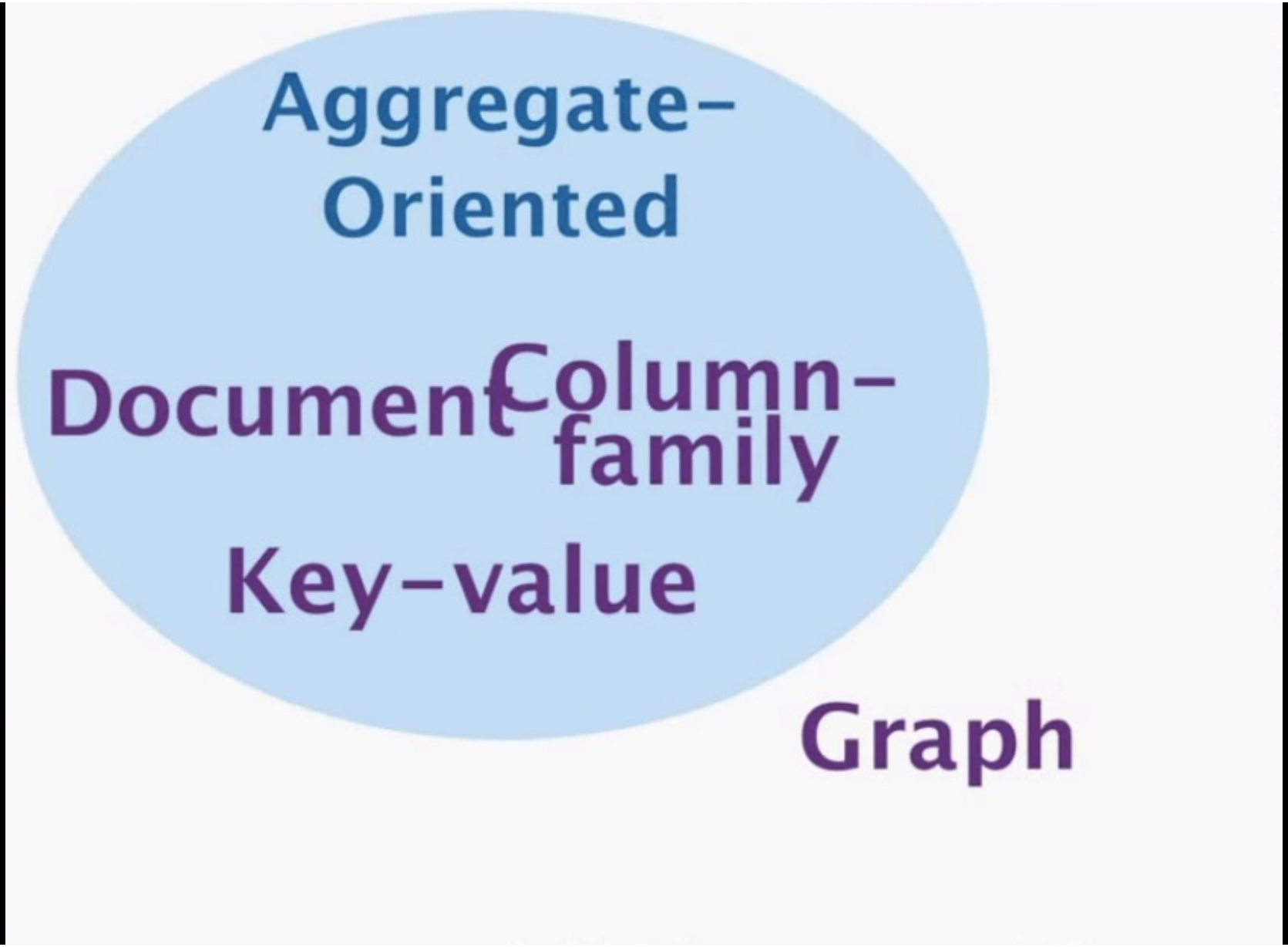
Document ==
Aggregate


Column
–
family
column family
column key
column value
profile
name
"martin"
billingAddress
data ..
payment
data ..
1234
row key
OR1001
data ..
OR1002
data ..
OR1003
data ..
OR1004
data ..
orders



Product	revenue	prior revenue
321293533	3083	7043
321601912	5032	4782
131495054	2198	3187
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...
...	...	...


Document
mongoDB
RAVEN DB
CouchDB
relax
Colum
n-
family
Cassandra
APACHE
HBASE
Neo4j
NOSQL for the Enterprise
Project Voldemort
A distributed database.
riak
Graph
Key-value
redis

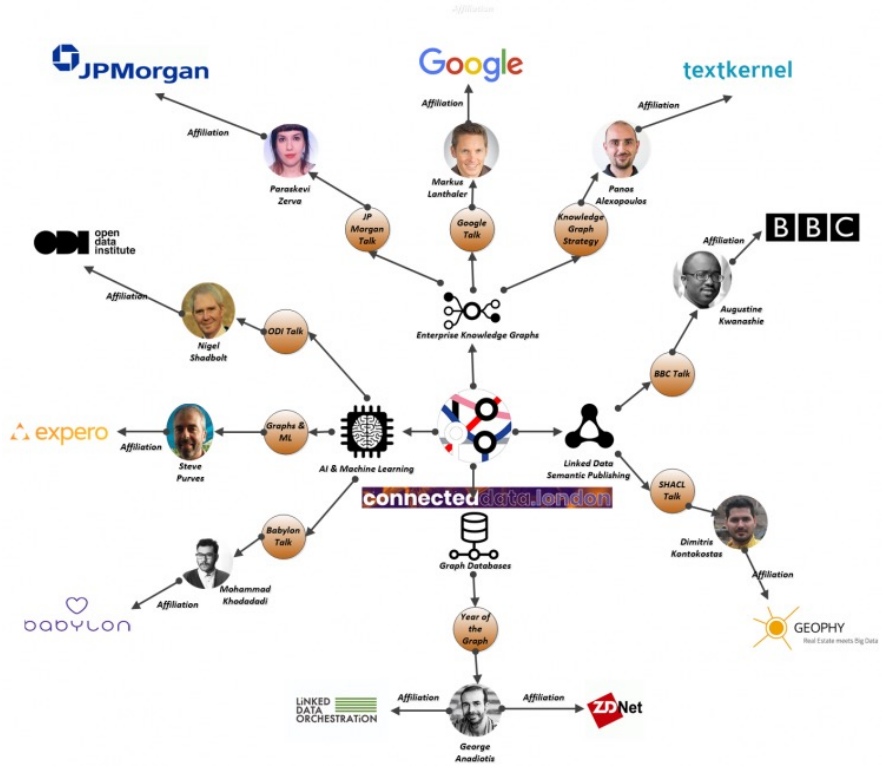

Aggregate-
Oriented
Document
Column-
family
Key-value
Graph

Humans think in graphs

- We understand better things by matching them to things that we know.
- And the things we know … are in the form of graphs

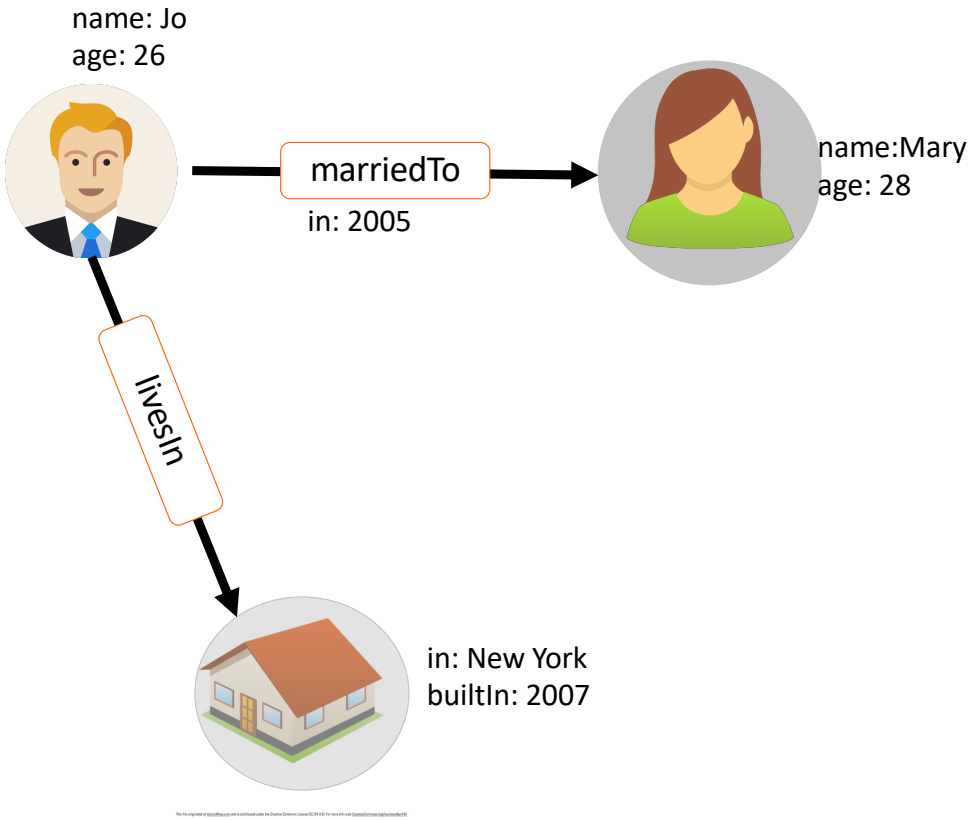
Graphs capture naturally heterogeneous data

- No specific schema
- Shows how different parts connect to each other.
- Hard to model with relational joins
- Very important for more informative insights



The property Graph Model

- Nodes
  - Represent Oblects
  - Can be labeled
- Relationships
  - Relate Nodes
  - Have a label (used as a type)
  - Have direction
- Properties
  - <Name: value> pairs
  - Describe characteristics
  - Can be on nodes or on edges

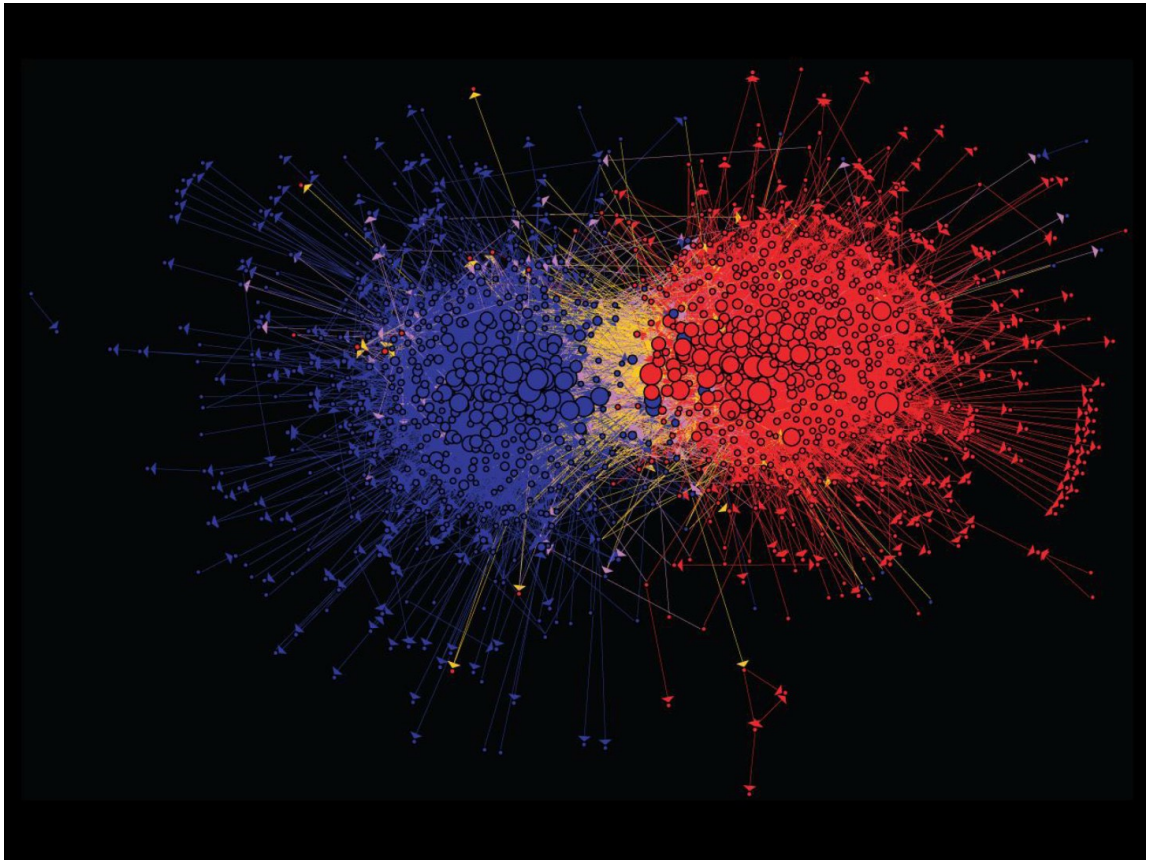


Graph Data: Social Networks

Facebook social graph

4-degrees of separation [Backstrom-Boldi-Rosa-Ugander-Vigna, 2011]

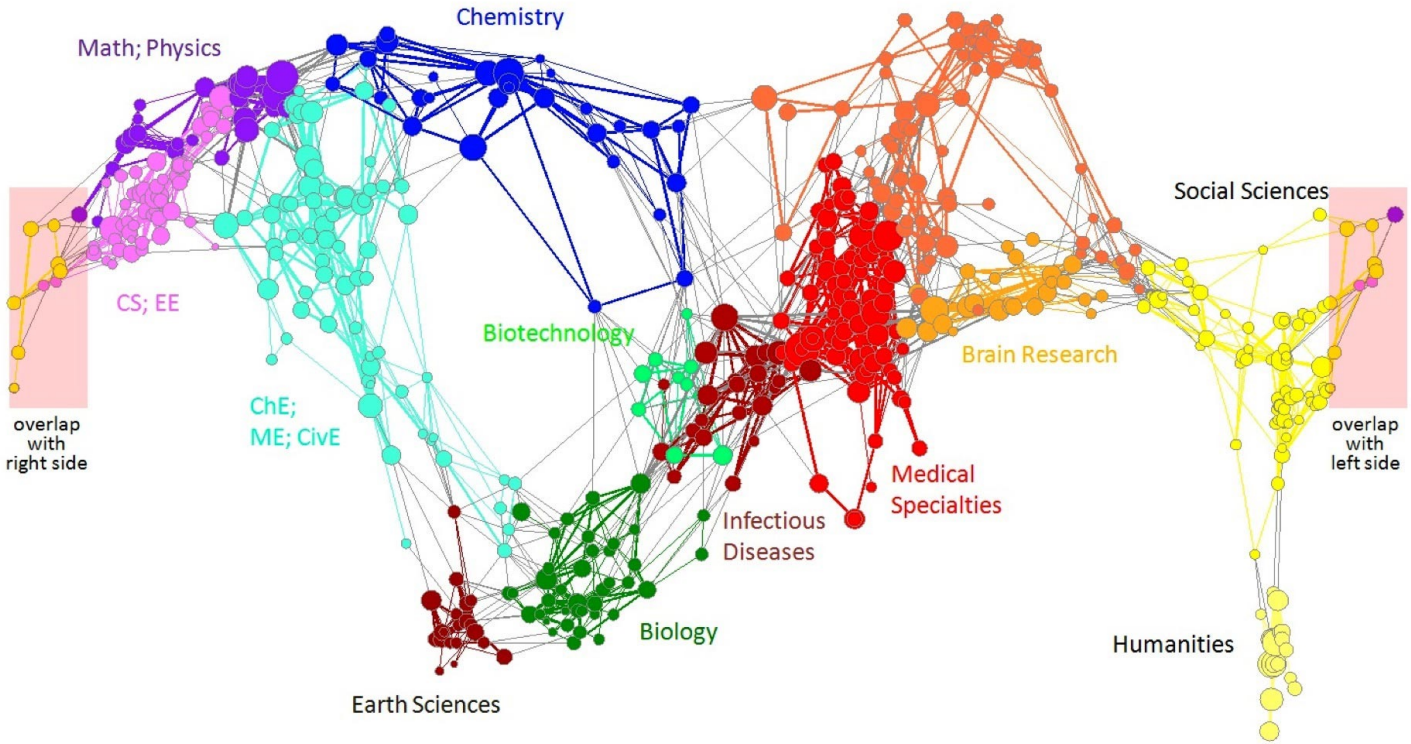
Graph Data: Media Networks



Connections between political blogs

Polarization of the network [Adamic-Glance, 2005]

Graph Data: Information Nets

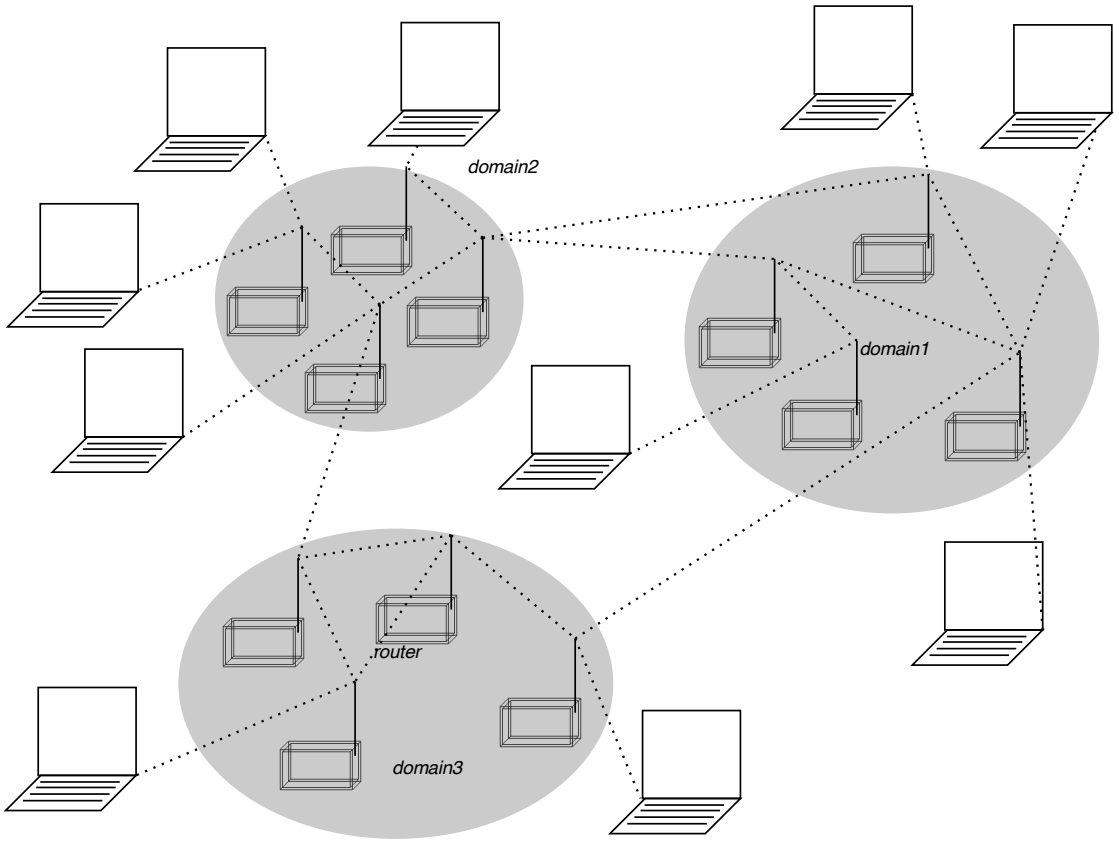


Citation networks and Maps of science

[Börner et al., 2012]

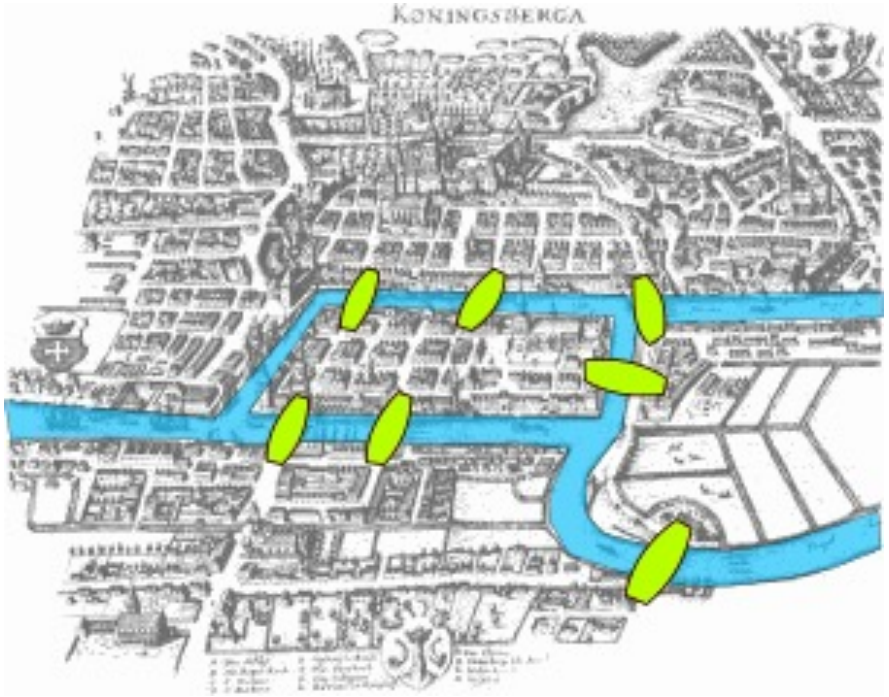
Graph Data: Transportation Networks

Graph Data: Communication Nets



Internet

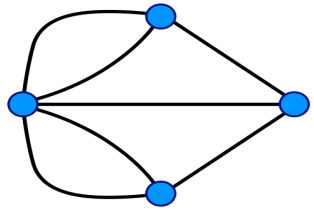
The first ever graph



Seven Bridges of Königsberg

[Euler, 1735]

Return to the starting point by traveling each link of the graph once and only once.

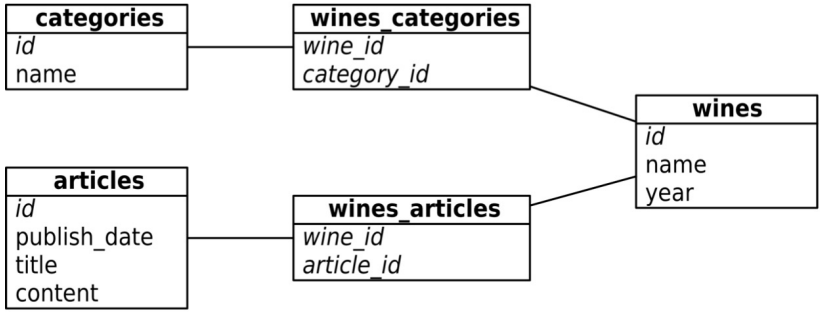


Neo4J

- Whiteboard friendly
- focuses more on the relationships between values than on the commonalities among sets of values
- can run in large clusters of servers using master-slave replication and store tens of billions of nodes and as many relationships

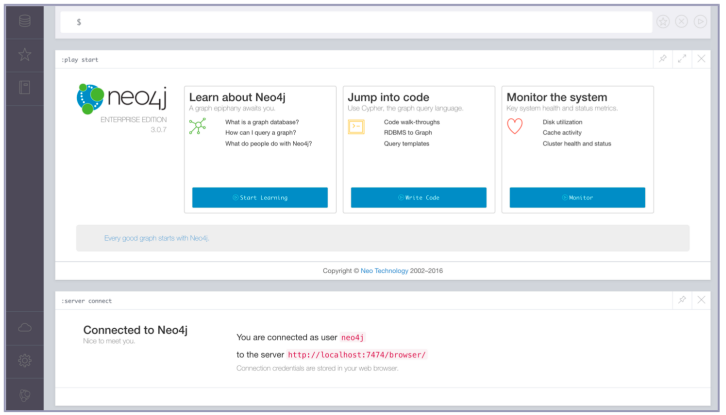
Whiteboard friendly

- Relational and Graph modelling differences
- Values and structure only when it is necessary

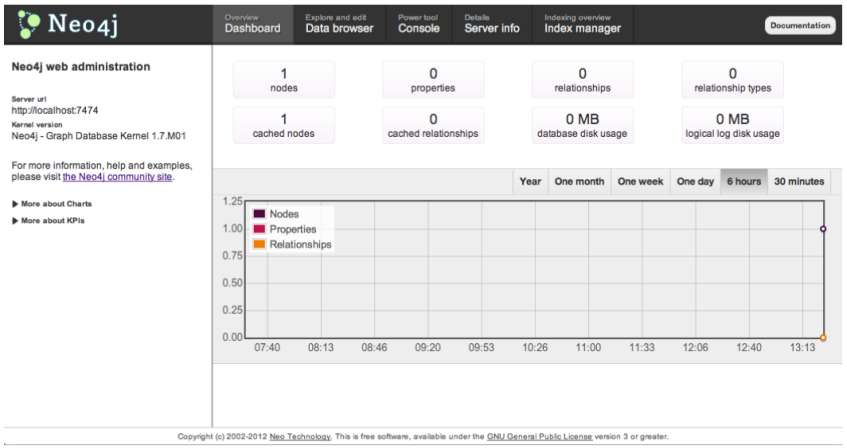


Interacting with Neo4J

- $ bin/neo4j start
- $ curl http://localhost:7474/db/data/
- http://localhost:7474/browser/
- Node = vertex
- Property: [name : "Wine Expert Monthly"])
  - Both nodes and edges



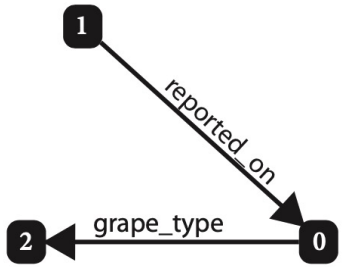
- Everything is an object
  - http://localhost:7474/db/data/relationship/0

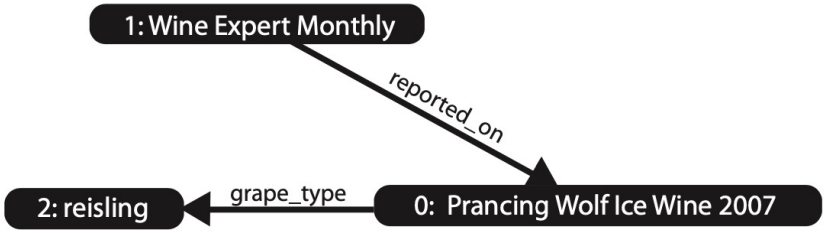


Gremlin

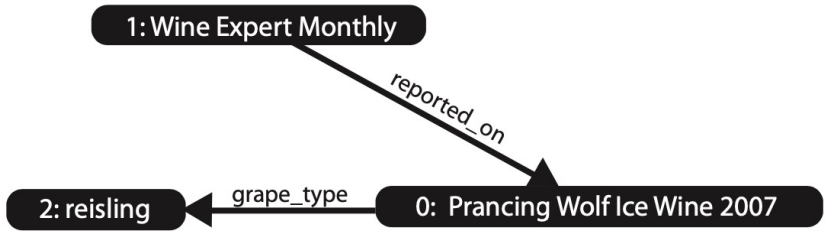
- Java, Rest, Cypher, Gremlin
- Gremplin: Graph traversal language
- g: variable that represents a graph
- Profiles

```
gremlin> g.V
==>v[0]
==>v[1]
==>v[2]

gremlin> g.E
==> e[0][1-reported_on->0]
==> e[1][0-grape_type->2]

gremlin> g.v(0)
==> v[0]

List properties:
gremlin> g.v(0).map()
==> name=Prancing Wolf Ice Wine 2007
```





Gremlin



gremlin> g.V.filter{**it**.name=='riesling’}
==> v[2]

Getting the outgoing edges:
gremlin> g.V.filter{it.name=='Wine Expert Monthly'}.outE()
==> e[0][1-reported_on->0]

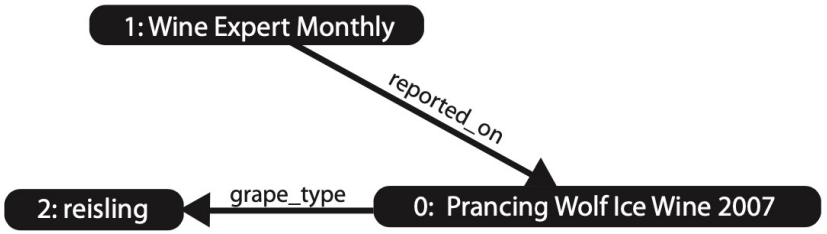
Or simply:
gremlin> g.V.filter{it.name=='Wine Expert Monthly'}.outE
==> e[0][1-reported_on->0]

Incoming edges:
gremlin> g.V.filter{it.name=='Wine Expert Monthly'}.outE.inV.name
==> Prancing Wolf Ice Wine 2007

Out is short for outE inV:
gremlin> g.V.filter{it.name=='Wine Expert Monthly'}.out.name
==> Prancing Wolf Ice Wine 2007

Gremlin

```
gremlin> pwolf = g.addVertex([name : 'Prancing Wolf Winery’]
==> v[3]
gremlin> g.addEdge(pwolf, g.v(0), 'produced’)
==> e[2][3-produced->0]
```



Gremlin = Pipes

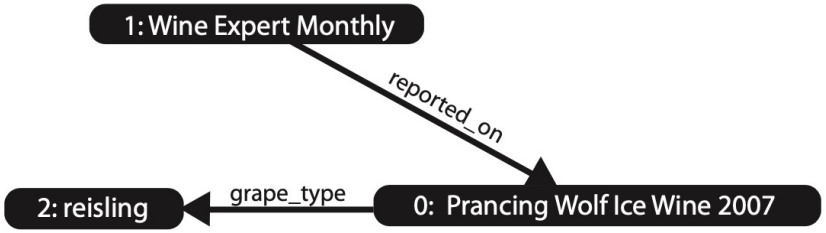
Built on top of a Java project called pipes

Example: follow an ice wine that also shares a grape_type edge with other out nodes (ignoring the initial wine node)

```
ice_wine = g.v(0)
ice_wine.out('grape_type').in('grape_type').filter{ !it.equals(ice_wine) }
```

```
enum WineRelationshipType implements RelationshipType {
  grape_type
}

import static WineRelationshipType.grape_type;

public static List<Node> same_variety( Node wine ) {
  List<Node> wine_list = new ArrayList<Node>();
  // walk into all out edges from this vertex
  for( Relationship outE : wine.getRelationships( grape_type ) ) {
    // walk into all in edges from this edge's out vertex
    for( Edge inE : outE.getEndNode().getRelationships( grape_type ) ) {
      // only add vertices that are not the given vertex
      if( !inE.getStartNode().equals( wine ) ) {
        wine_list.add( inE.getStartNode() );
      }
    }
  }

  return wine_list;
}
```

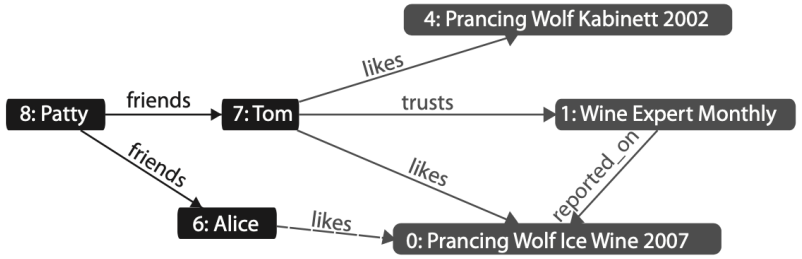
Classes & accessing the elements




```
gremlin> g.V.filter{it.name=='Prancing Wolf Winery'}.class
==>class com.tinkerpop.gremlin.pipes.GremlinPipeline
```

The next() command for iteration:

```
alice = g.addVertex([name:'Alice’])
ice_wine = g.V.filter{it.name=='Prancing Wolf Ice Wine 2007'}.next()
g.addEdge(alice, ice_wine, 'likes’)
```

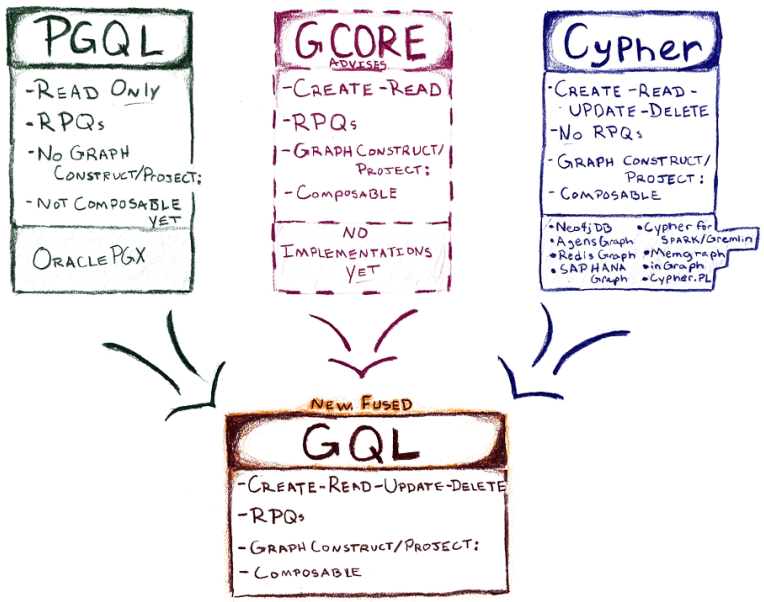


inE/outE/inV/outV: bothE & bothV

```
alice.bothE('friends').bothV.name
==> Alice
==> Patty
```

The Property Graph Query Language

- Why not SQL
  - Cannot express graph patterns
  - Not easy to do recursions
  - Go easier with heterogeneous structures
  - Conceptually easier to understand




```
...
MATCH (a)
CALL {
  MATCH (a)-[:FRIEND_OF]-(x)-[:FRIEND_OF]-(b)
  WHERE a <> x AND x <> b AND a <> b
  RETURN count(DISTINCT b) AS num_foafs
}
RETURN a.name, num_foafs
 ORDER BY num_foafs DESC
 LIMIT 100
```

Cypher

What Breweries Near Me That My Friends Also Like Have The Highest Rating?


```
1 MATCH (c:Category)<-[:IN_CATEGORY]-(brew:Business)
2 MATCH (brew)<-[:REVIEWS]-(r:Review)<-[:WROTE]-(u:User)-[:FRIENDS]-(me:User {user_id: 'qZB8BZ2ZzMF0VfIGhttu7w'})
3 WHERE c.name CONTAINS 'Breweries' AND brew.city = "Phoenix" AND r.stars > 4
4 MATCH (brew)<-[:REVIEWS]-(r2:Review)
5 RETURN brew.name, brew.address, avg(r2.stars) AS average_reviews ORDER BY average_reviews DESC
```

- `MATCH ...`
- `(node)-[:RELATIONSHIP]->(node)`
- Predicates
- Aggregations
- Ordering
- Graph vs tabular data

Neo4J strengths & weakneses

Strengths:

- Open source graph database
- Scales well
- Typelesss and schemaless (although there is a plugin for simulation of types)
- Scaling example: 34.4 billion nodes and 34.4 billion relationships

Weakness: Replicates graph … not subgraphs

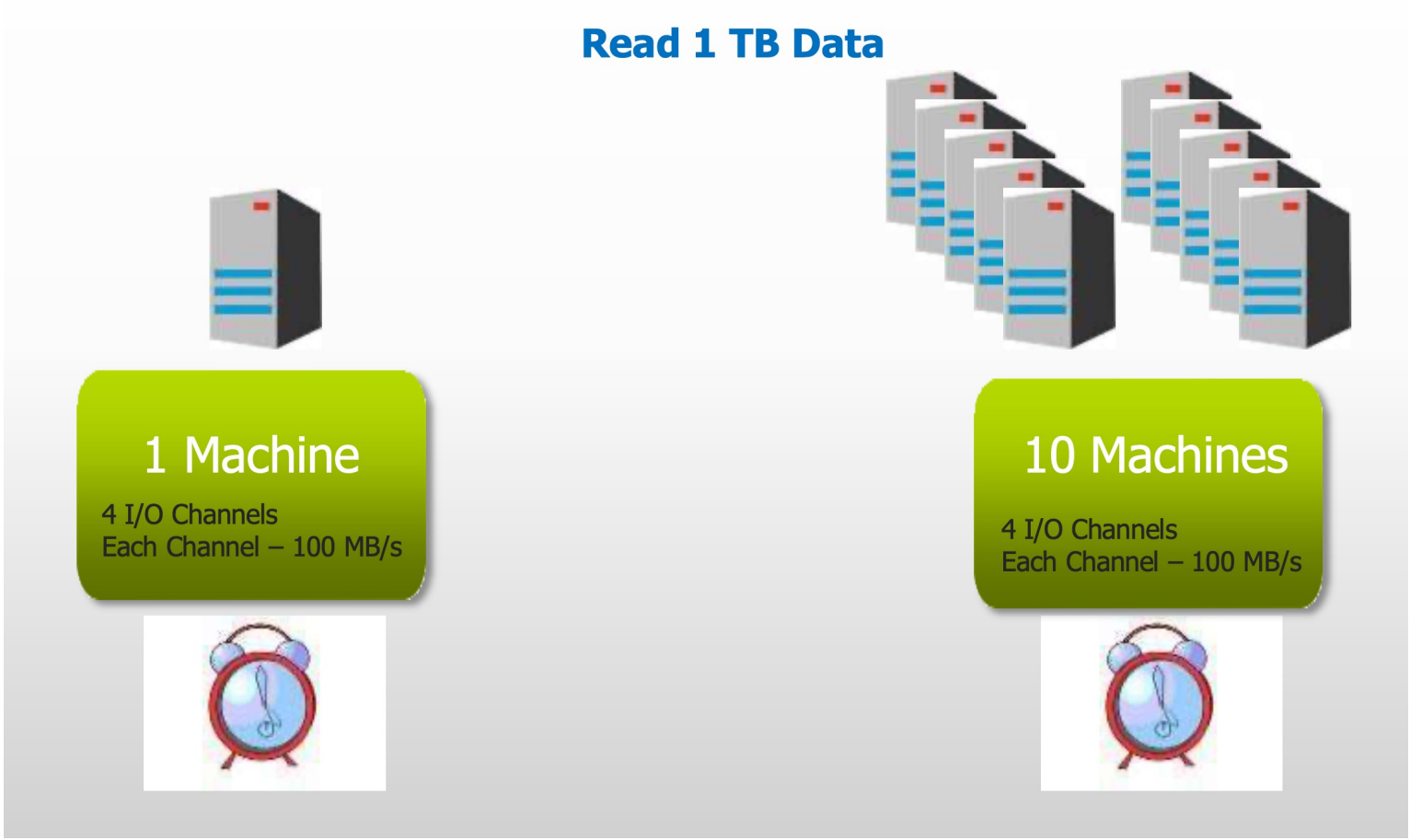
- Limits graph size although it is in the level of millions

Graph Management Systems

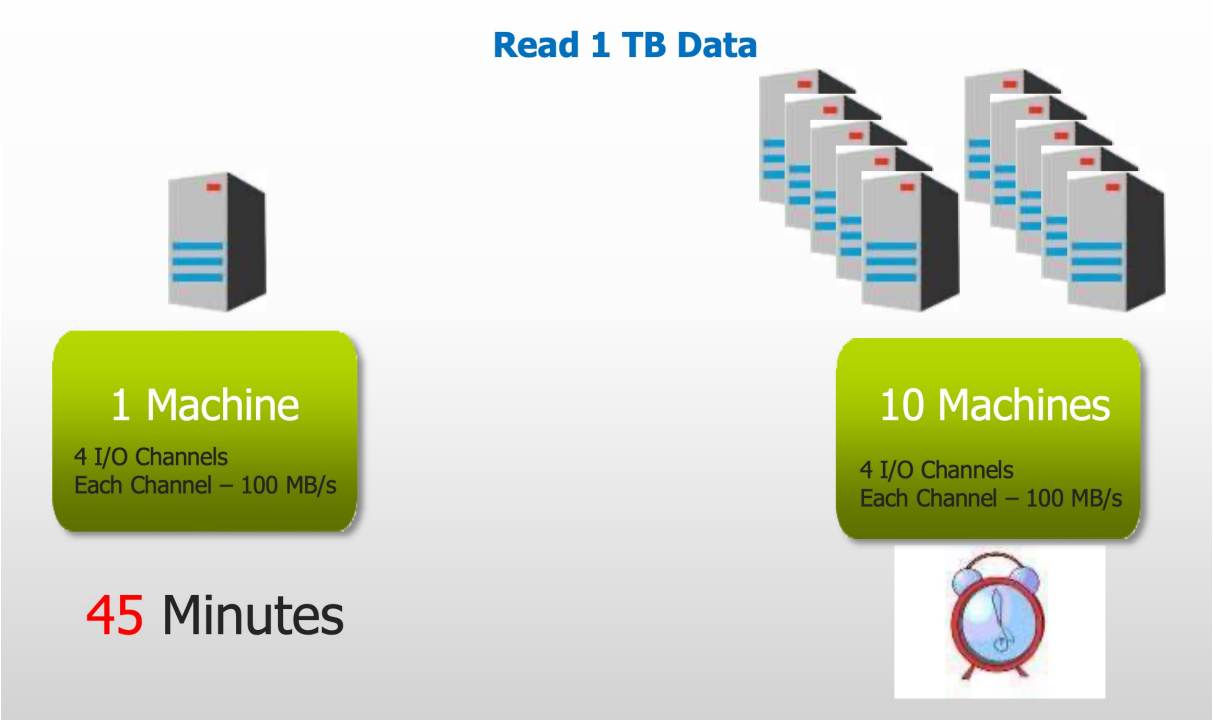
Visit https://people.cs.aau.dk/~matteo/gdb.html for a comparison

A file system for Clusters

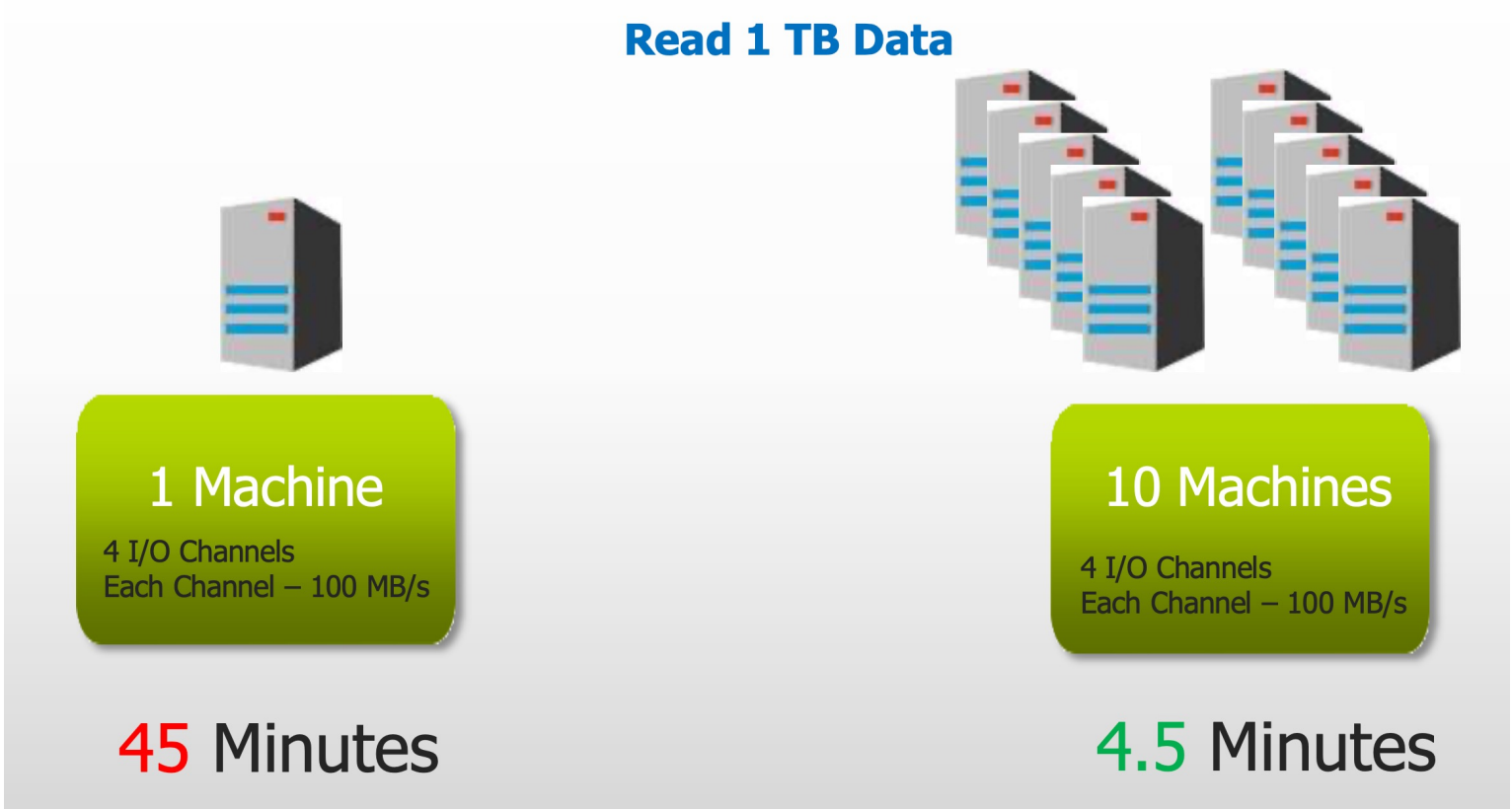
Distributed FS



Distributed FS

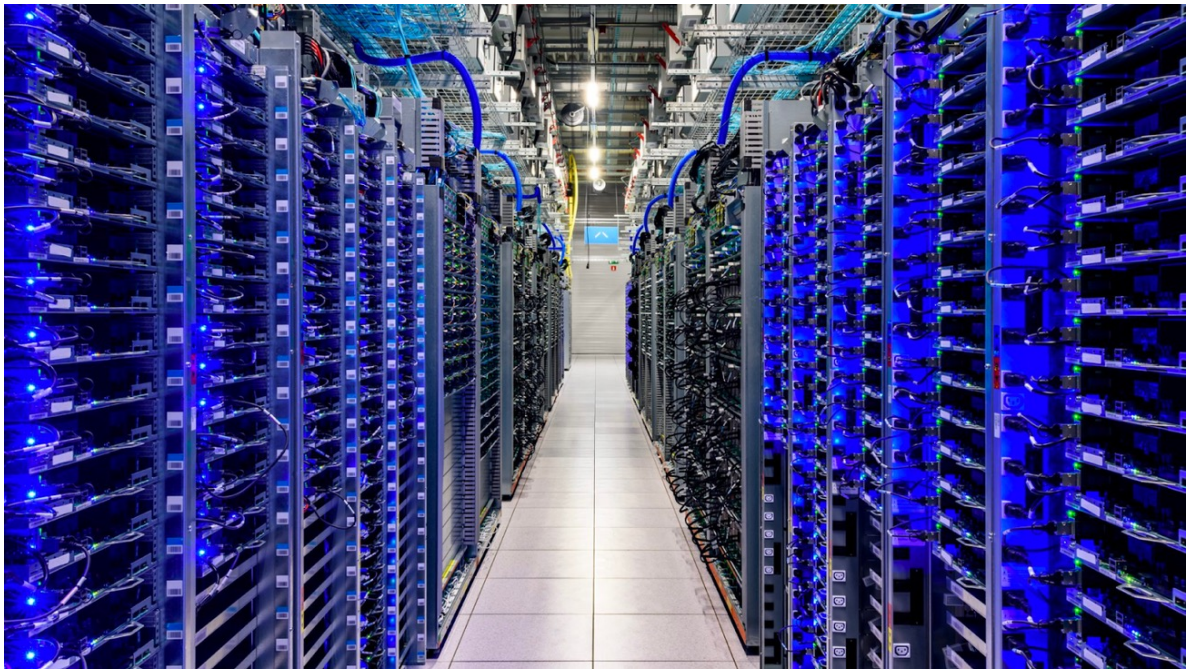


Distributed FS



- Replication of course

A Data Center



A Container style Data Center

A google data center may contain
45 containers which is ~60,000 servers

The Need for Concistency … or Not

ACID

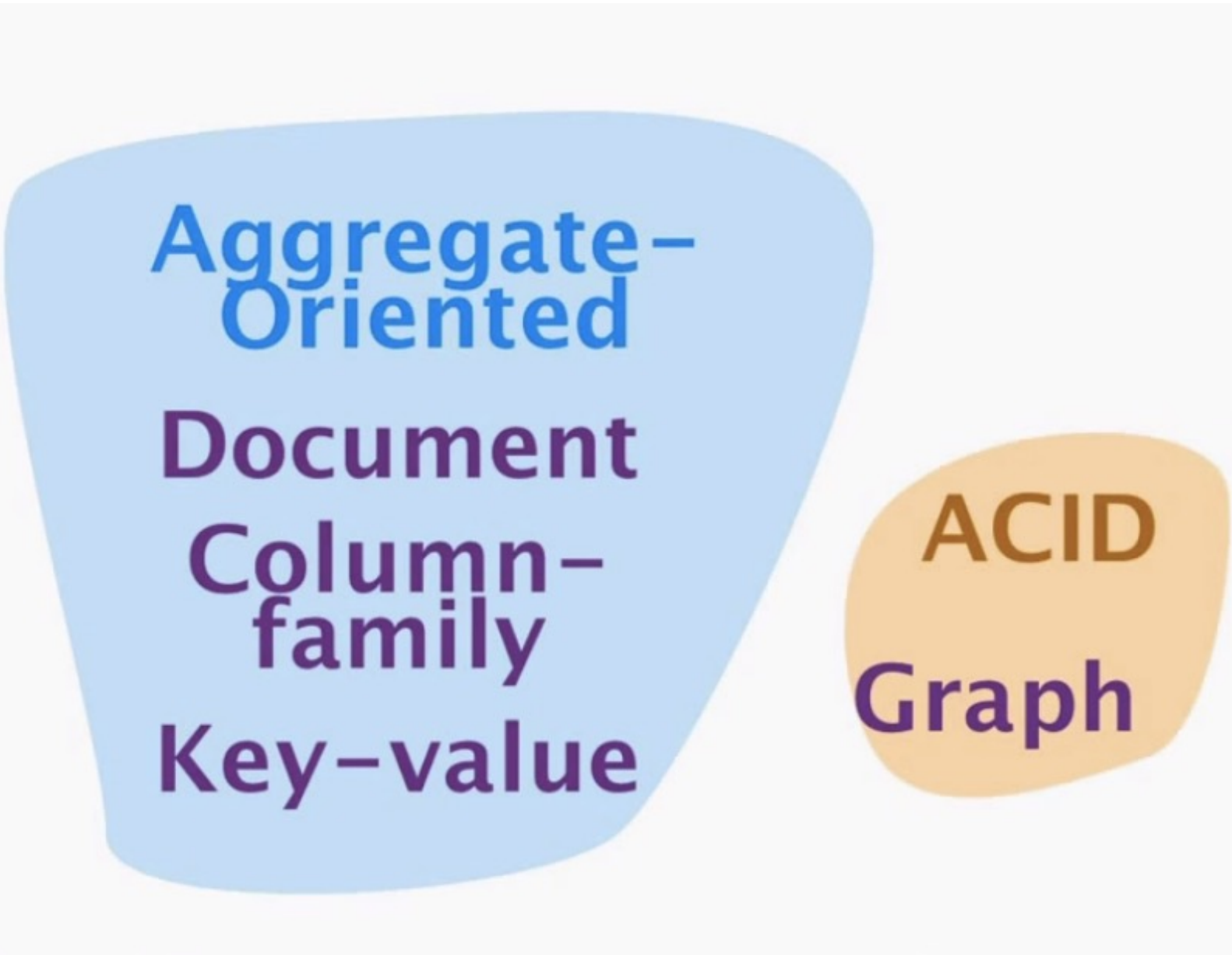
- ACID
  - **Atomic** – Each transaction is either properly carried out or the process halts and the database reverts back to the state before the transaction started. This ensures that all data in the database is valid.
  - **Consistent** – A processed transaction will never endanger the structural integrity of the database.
  - **Isolated** – Transactions cannot compromise the integrity of other transactions by interacting with them while they are still in progress.
  - **Durable** – The data related to the completed transaction will persist even in the cases of network or power outages. If a transaction fails, it will not impact the manipulated data.

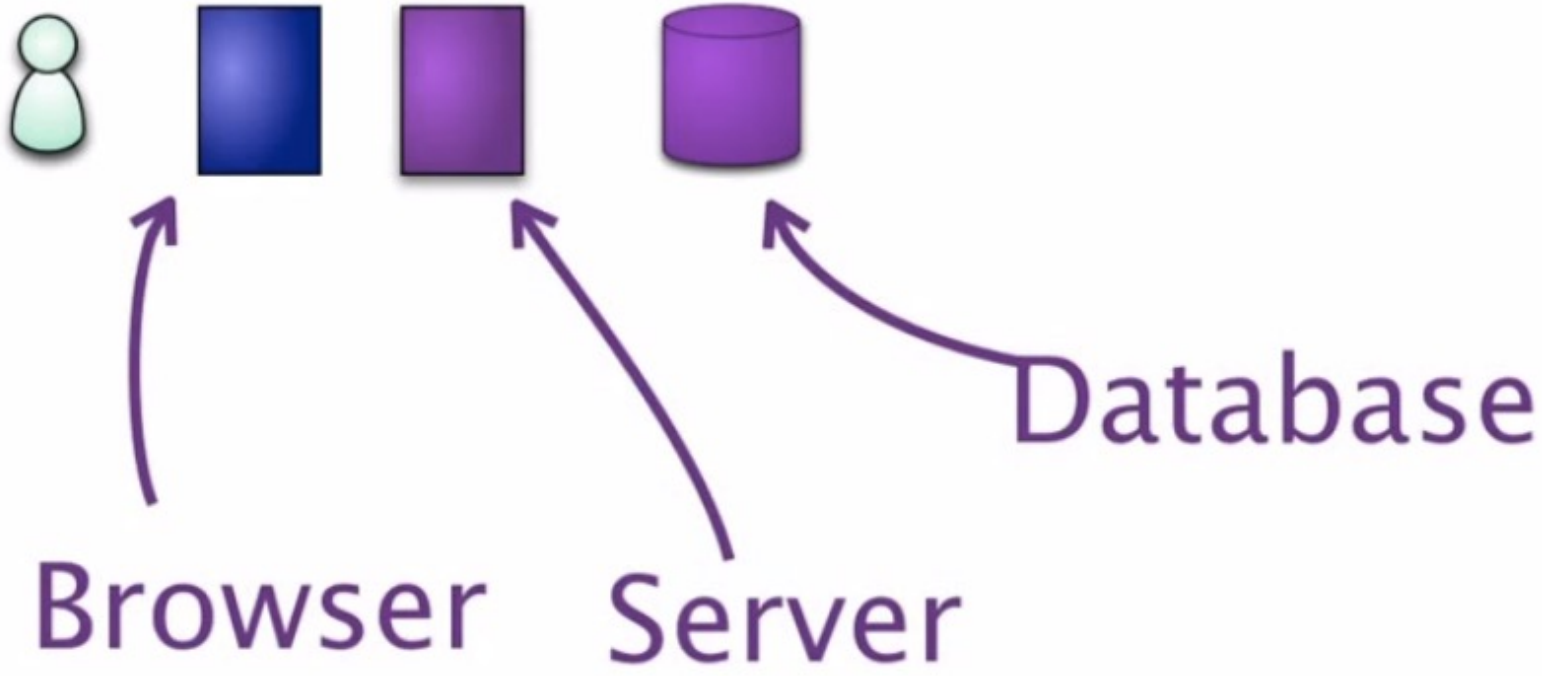
BASE

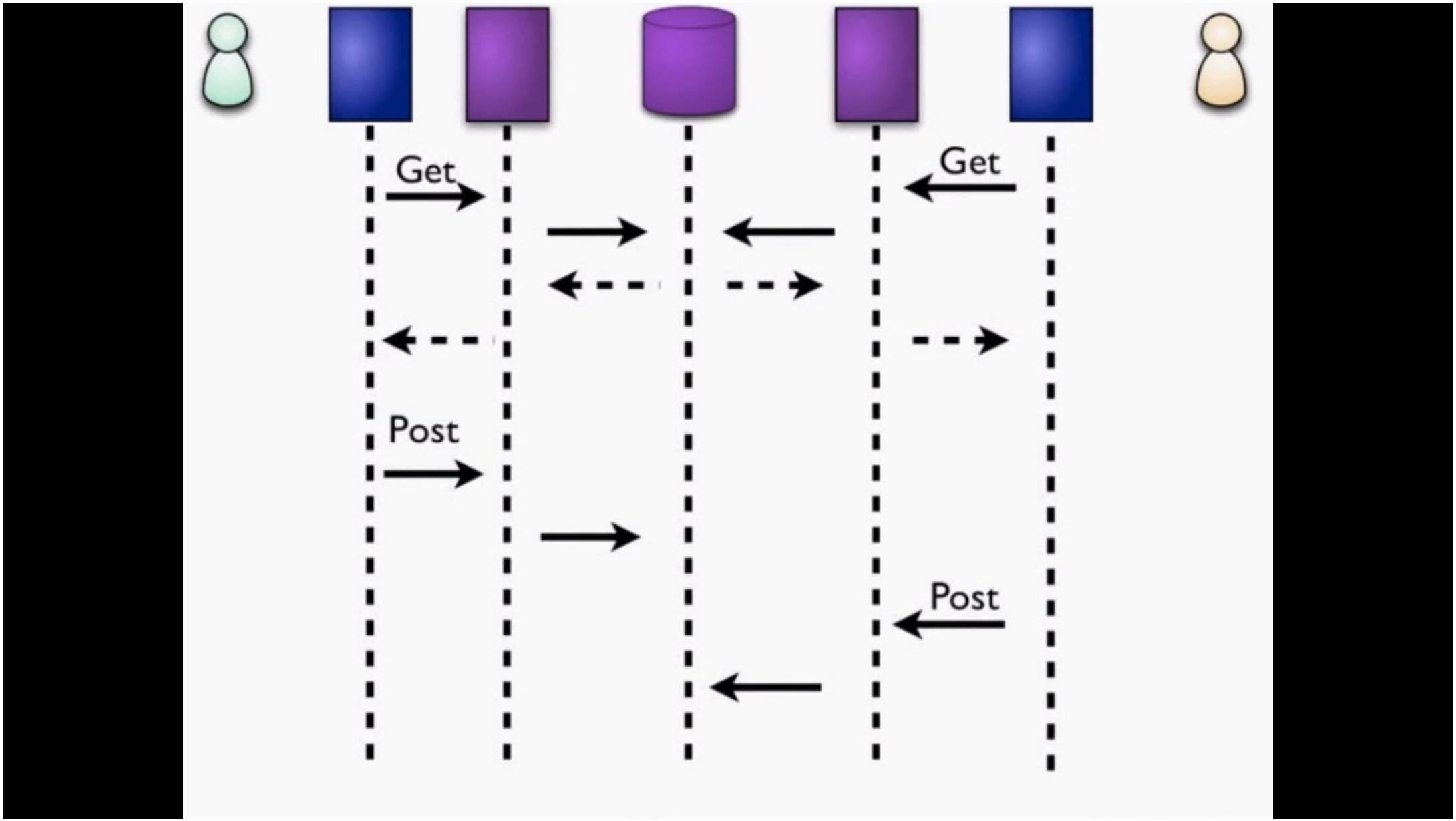
- Base
  - **Basically Available** – Rather than enforcing immediate consistency, BASE-modelled NoSQL databases will ensure availability of data by spreading and replicating it across the nodes of the database cluster.
  - **Soft State** – Due to the lack of immediate consistency, data values may change over time. The BASE model breaks off with the concept of a database which enforces its own consistency, delegating that responsibility to developers.
  - **Eventually Consistent** – The fact that BASE does not enforce immediate consistency does not mean that it never achieves it. However, until it does, data reads are still possible (even though they might not reflect the reality).

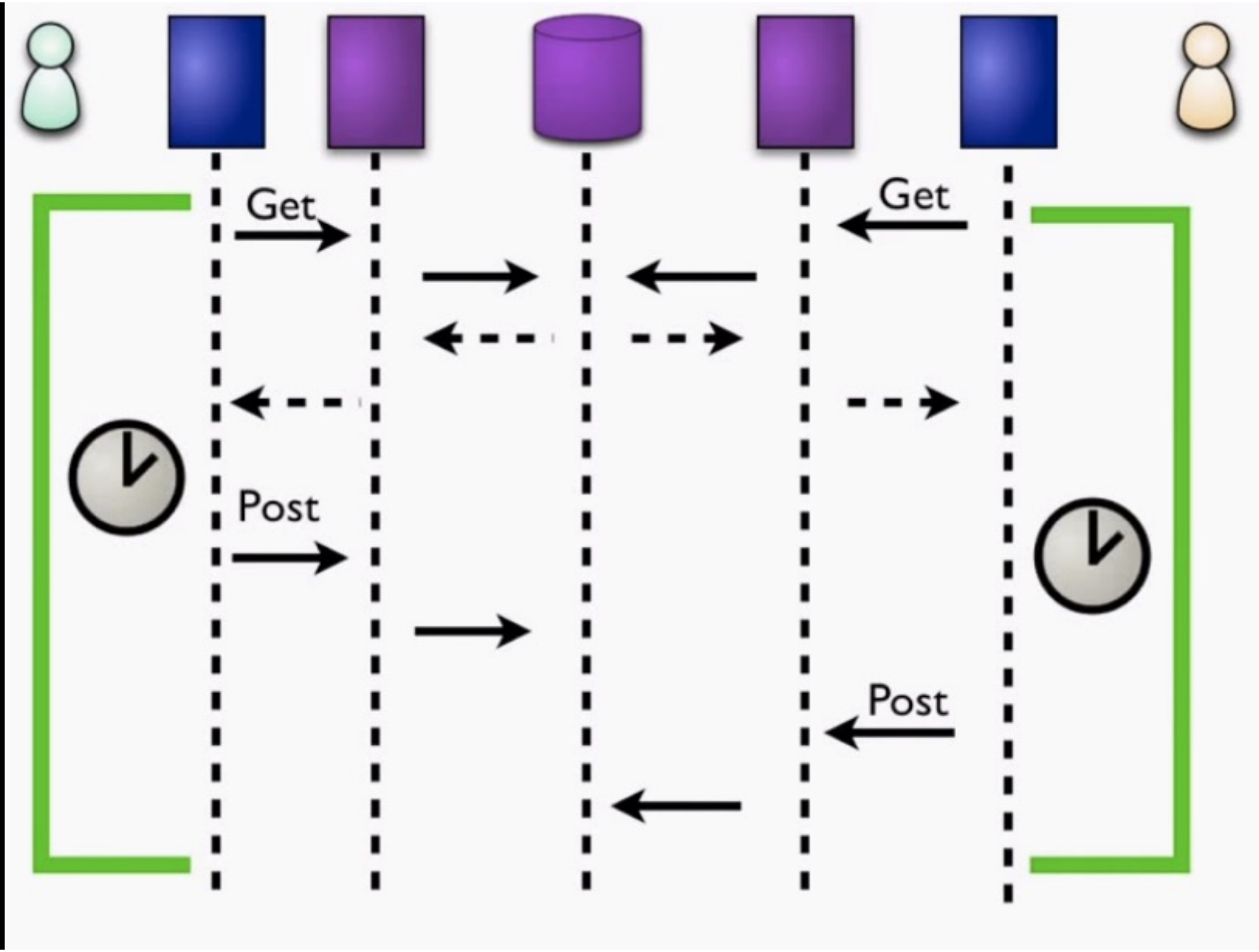
BASE

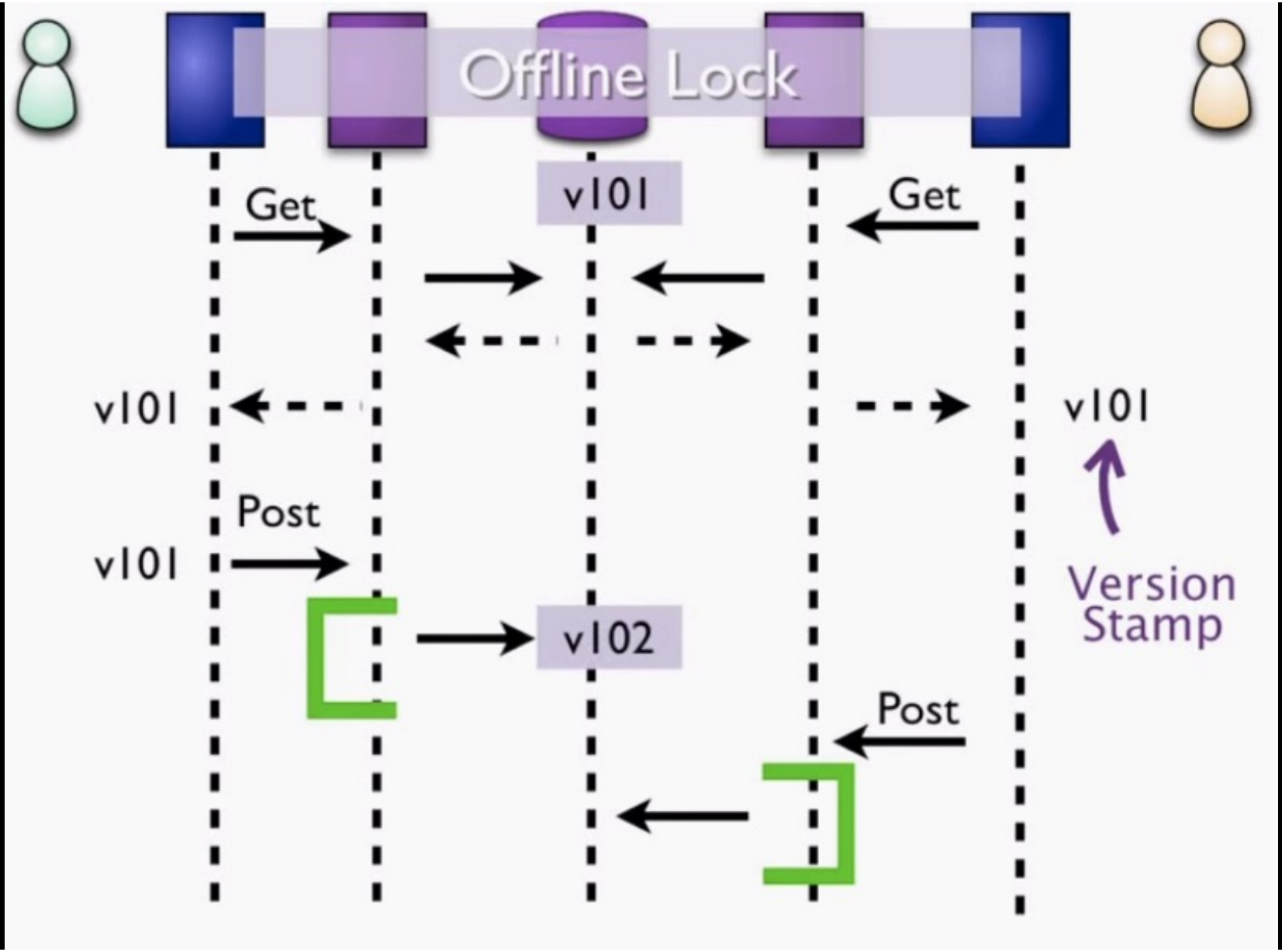
- RDBMS = ACID
- NoSQL = BASE (Well …. almost)


Aggregate-Oriented
Document
Column-family
Key-value
ACID
Graph


Database
Browser
Server

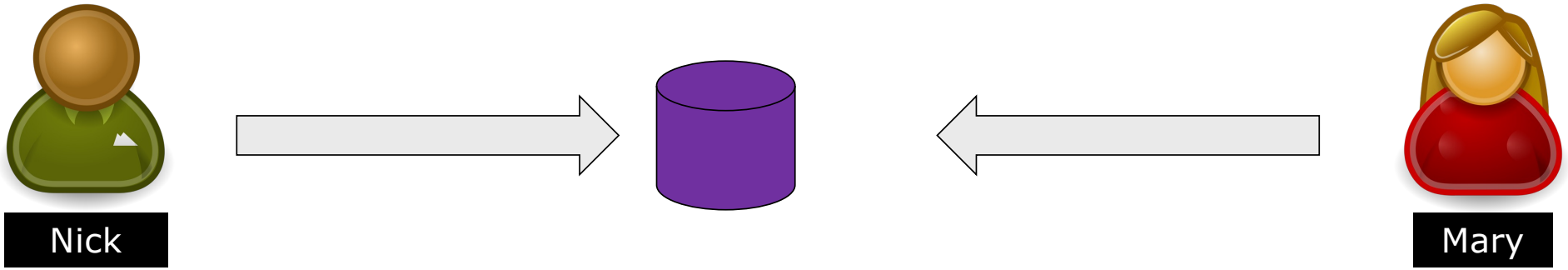

Get
Get
Post
Post


Get
Get
Post
Post

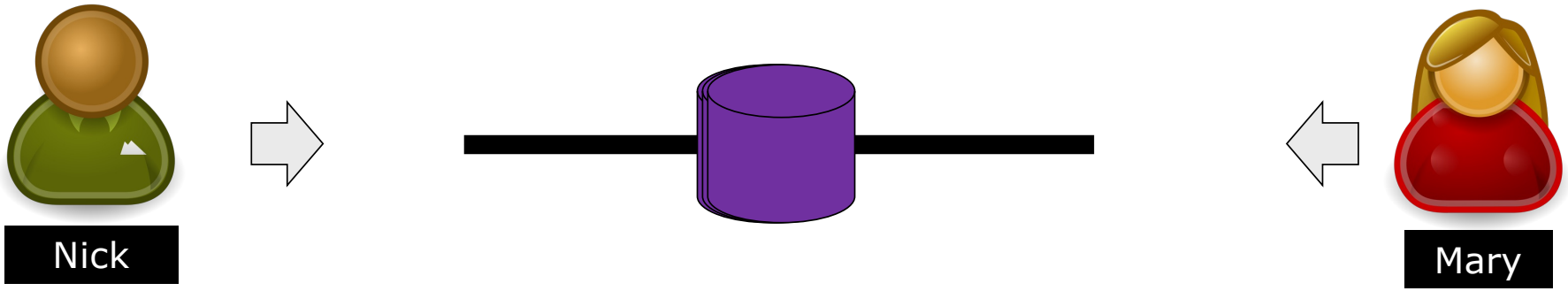

Offline Lock
v101
Get
Get
v101
v101
Post
v101
Version
Stamp
v102
Post

The Effect of Replication

The Effect of Replication



The Effect of Replication



The Effect of Replication

The Effect of Replication

Consistency Guaranteed

The Effect of Replication

Availability Guaranteed

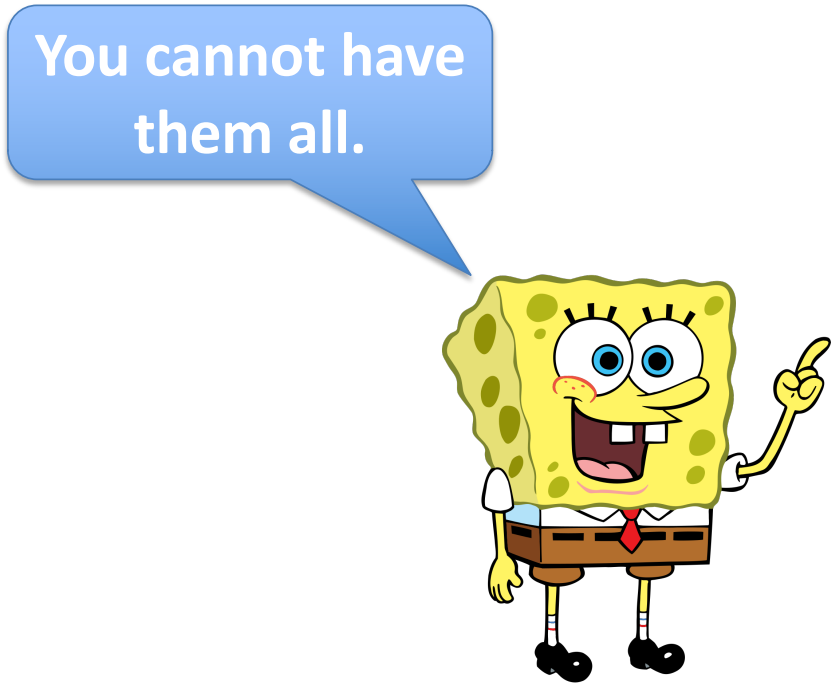
- **Consistency**
  - Do all applications see all the same data?
- **Availability**
  - If some nodes fail, does everything still work?
- **Partitioning**
  - If your nodes can’t talk to each other, does everything still work?

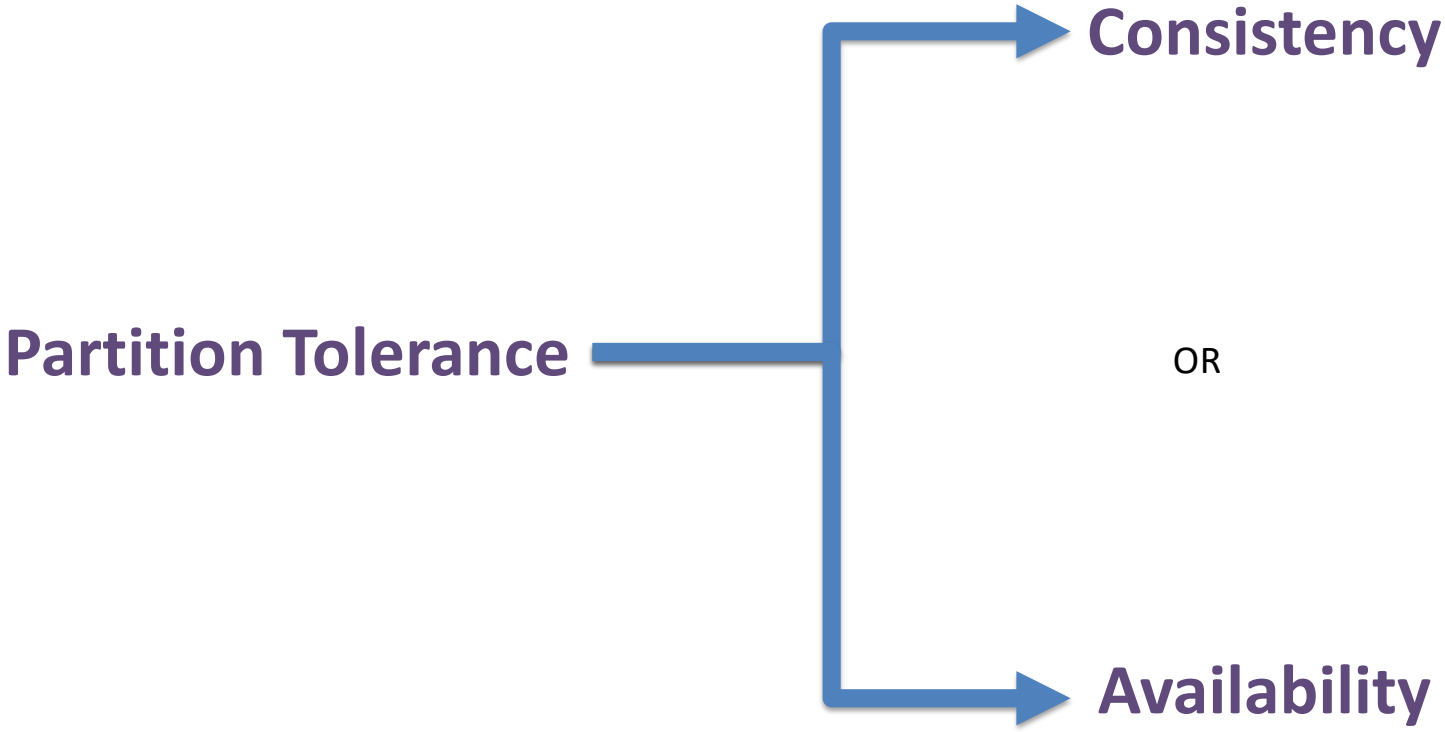
Partition Tolerance

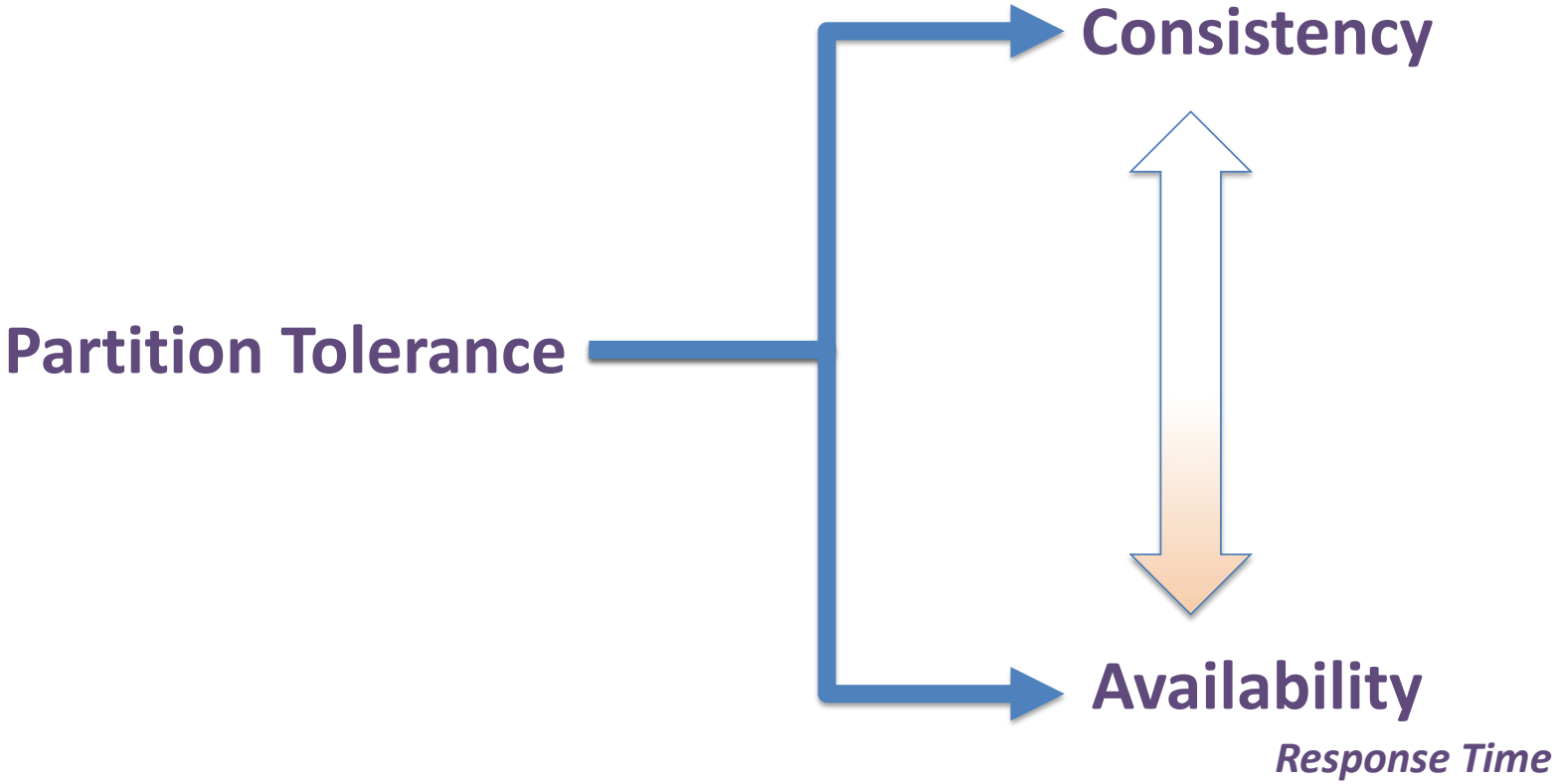
Consistency

CAP Theorem [Brewer 2000, Lynch 2002]

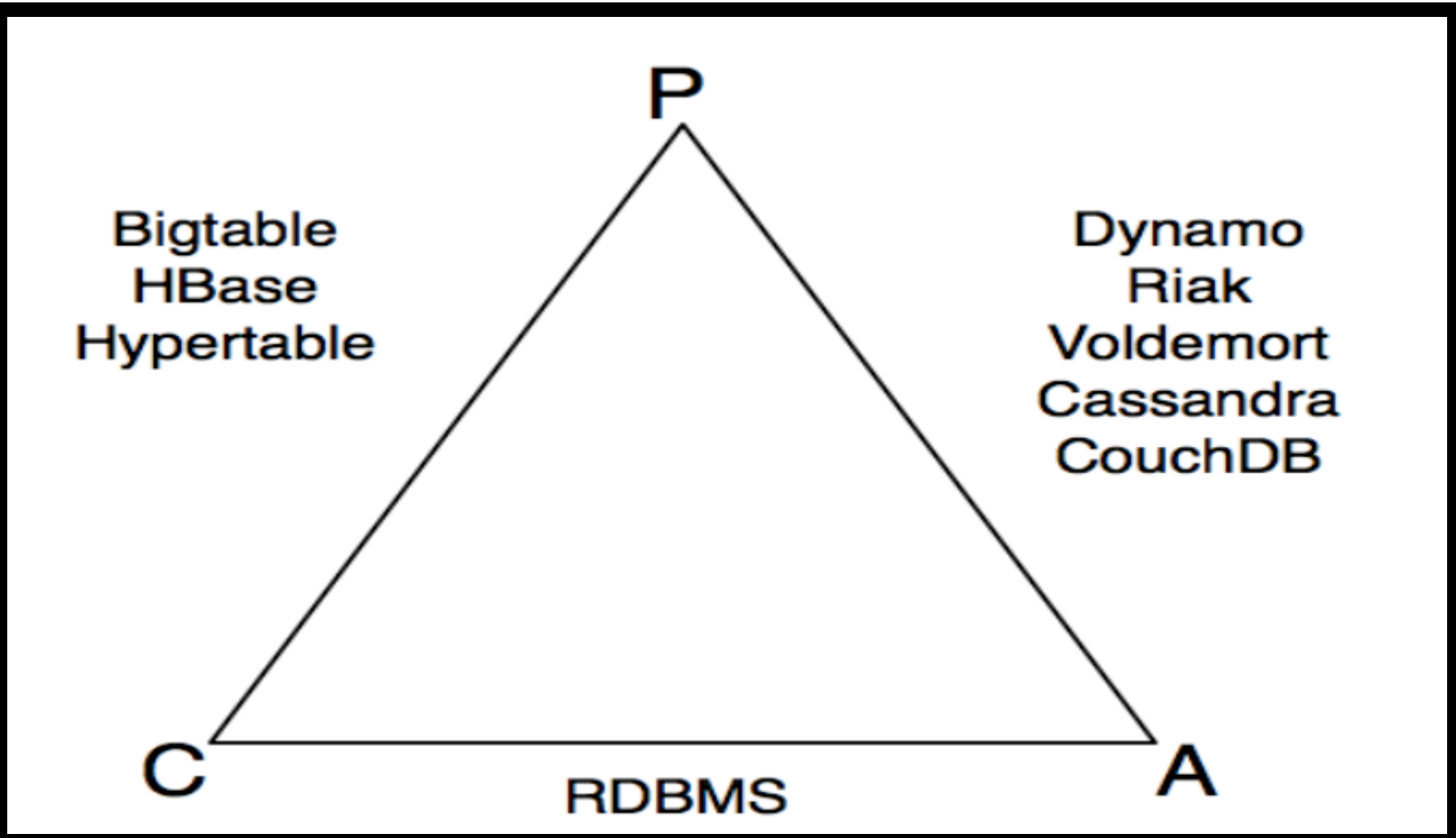
Availability




Consistency
Partition Tolerance
OR
Availability


Consistency
Partition Tolerance
Availability
Response Time

95



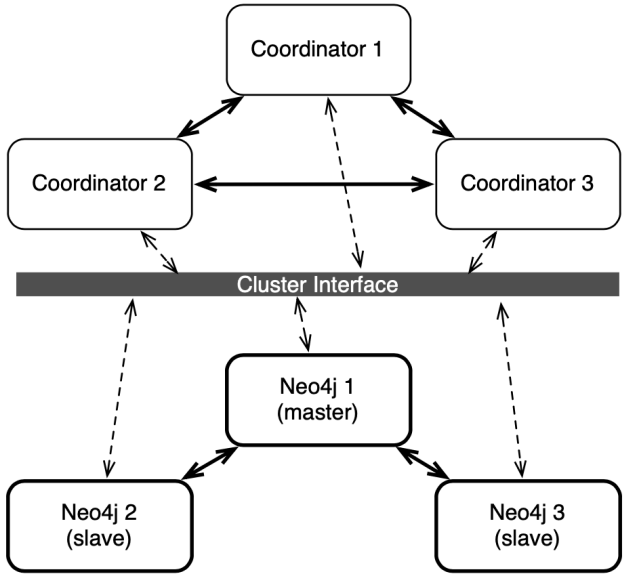
src: Shashank Tiwari

High Availability in Neo4J

Many nodes that are sychronized
A write in one slave is not immediatelly sychronize

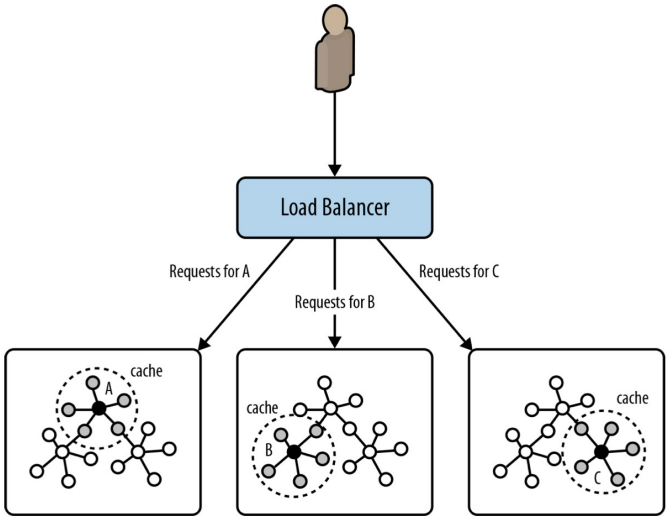
Synchronization is based on Zookeeper (Hadoop)

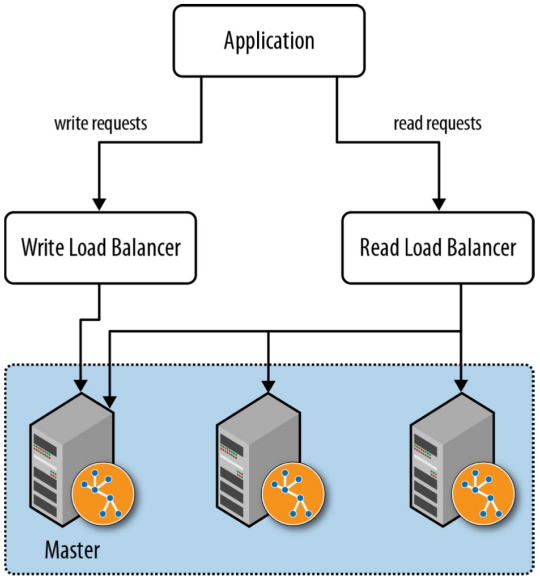
1. Set unique IDs for each coordinator server.
2. Configure each coordinator server to communicate with the other servers and its hosted Neo4j server.
3. Start up all three coordinator servers.
4. Configure each Neo4j server to run in HA (high availability) mode, give them unique ports, and make them aware of the coordinator cluster.

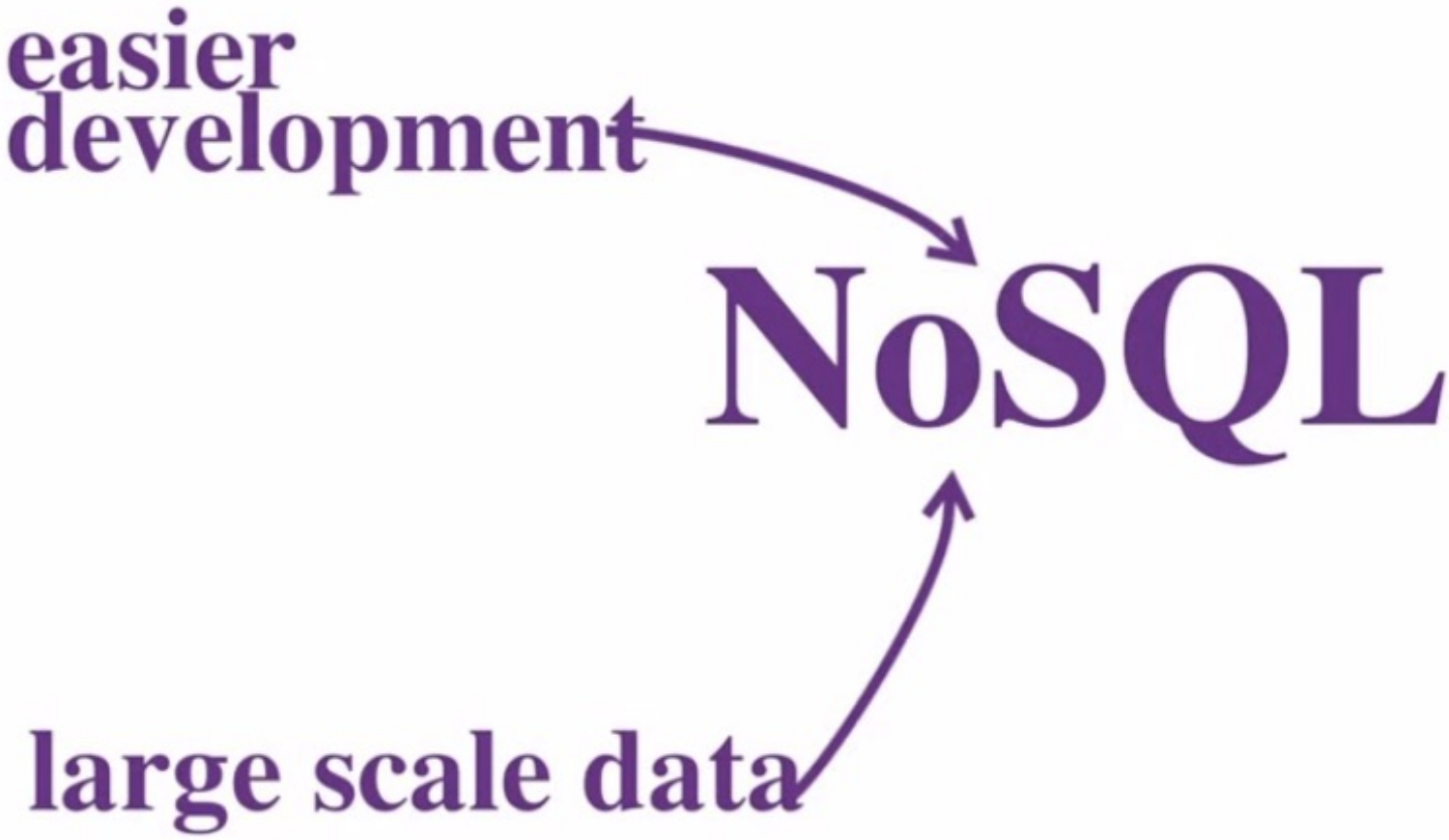


Load Balancing

1. Separate Reads and Writes
2. Cache sharding






easier
development
NoSQL
large scale data

NoSQL Criticism: flexibility argument

- Who are the customers of NoSQL?
  - Lots of startups
- Very few enterprises. Why? most applications are traditional OLTP on structured data; a few other applications around the “edges”, but considered less important

Year	System/ Paper	Scale to 1000s	Primary Index	Secondary Indexes	Transactions	Joins/ Analytics	Integrity Constraints	Views	Language/ Algebra	Data model	my label
1971	RDBMS	o	✓	✓	✓	✓	✓	✓	✓	tables	sql-like
2003	memcached	✓	✓	o	o	o	o	o	o	key-val	nosql
2004	MapReduce	✓	o	o	o	✓	o	o	o	key-val	batch
2005	CouchDB	✓	✓	✓	record	MR	o	✓	o	document	nosql
2006	BigTable (Hbase)	✓	✓	✓	record	compat. w/MR	/	o	o	ext. record	nosql
2007	MongoDB	✓	✓	✓	EC, record	o	o	o	o	document	nosql
2007	Dynamo	✓	✓	o	o	o	o	o	o	key-val	nosql
2008	Pig	✓	o	o	o	✓	/	o	✓	tables	sql-like
2008	HIVE	✓	o	o	o	✓	✓	o	✓	tables	sql-like
2008	Cassandra	✓	✓	✓	EC, record	o	✓	✓	o	key-val	nosql
2009	Voldemort	✓	✓	o	EC, record	o	o	o	o	key-val	nosql
2009	Riak	✓	✓	✓	EC, record	MR	o			key-val	nosql
2010	Dremel	✓	o	o	o	/	✓	o	✓	tables	sql-like
2011	Megastore	✓	✓	✓	entity groups	o	/	o	/	tables	nosql
2011	Tenzing	✓	o	o	o	o	✓	✓	✓	tables	sql-like
2011	Spark/Shark	✓	o	o	o	✓	✓	o	✓	tables	sql-like
2012	Spanner	✓	✓	✓	✓	?	✓	✓	✓	tables	sql-like
2012	Accumulo	✓	✓	✓	record	compat. w/MR	/	o	o	ext. record	nosql
2013	Impala	✓	o	o	o	✓	✓	o	✓	tables	sql-like

Rick Cattel's clustering from
"Scalable SQL and NoSQL Data Stores"
SIGMOD Record, 2010

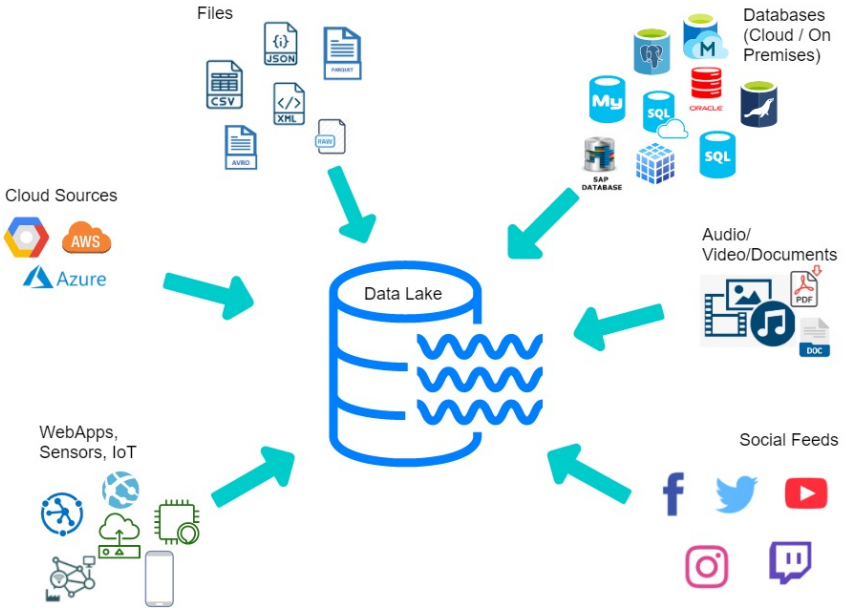
extensible record stores

document stores

key-value stores

Data Lake

- a centralized repository that allows you to store all your structured and unstructured data at any scale.



DATA LAKE

Data

unstructured

Users

Data Scientists,
Data Analysts

Use cases

Stream Processing,
Machine Learning,
Real time analysis

Raw

Data Lakes contain unstructured, semi structured and structured data with minimal processing. It can be used to contain unconventional data such as log and sensor data

Large

Data Lakes contain vast amounts of data in the order of petabytes. Since the data can be in any form or size, large amounts of unstructured data can be stored indefinitely and can be transformed when in use only

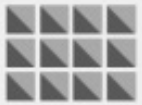
Undefined

Data in data lakes can be used for a wide variety of applications, such as Machine Learning, Streaming analytics, and AI

vs

DATA WAREHOUSE

Data



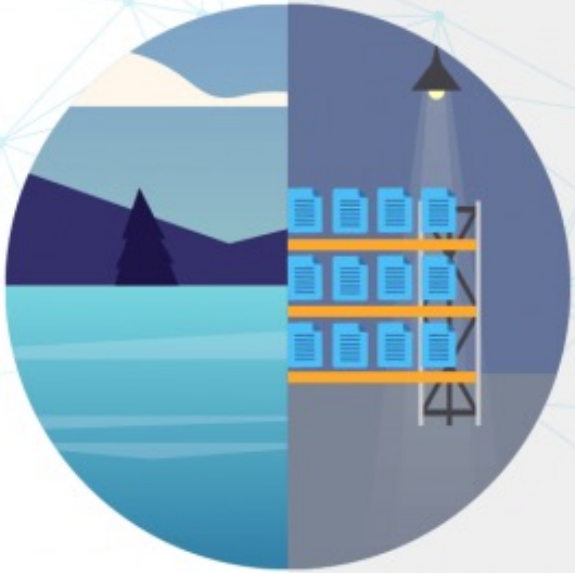
Structured

Users

Business Analysts

Use cases

Batch Processing,
BI, Reporting



Refined

Data Warehouses contain highly structured data that is cleaned, pre-processed and refined. This data is stored for very specific use cases such as BI.

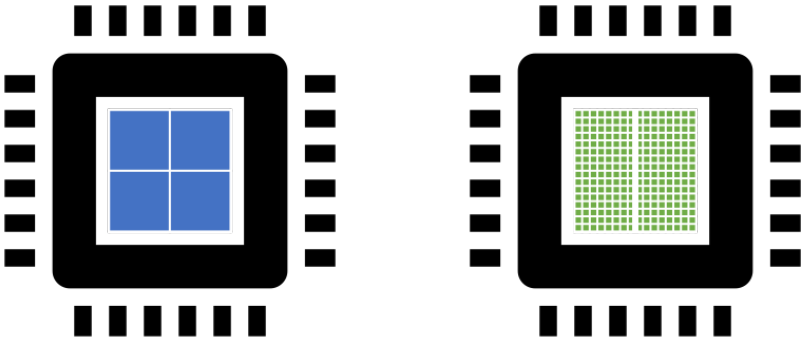
Smaller

Data Warehouses contain less data in the order of terabytes. In order to maintain data cleanliness and health of the warehouse, Data must be processed before ingestion and periodic purging of data is necessary

Relational

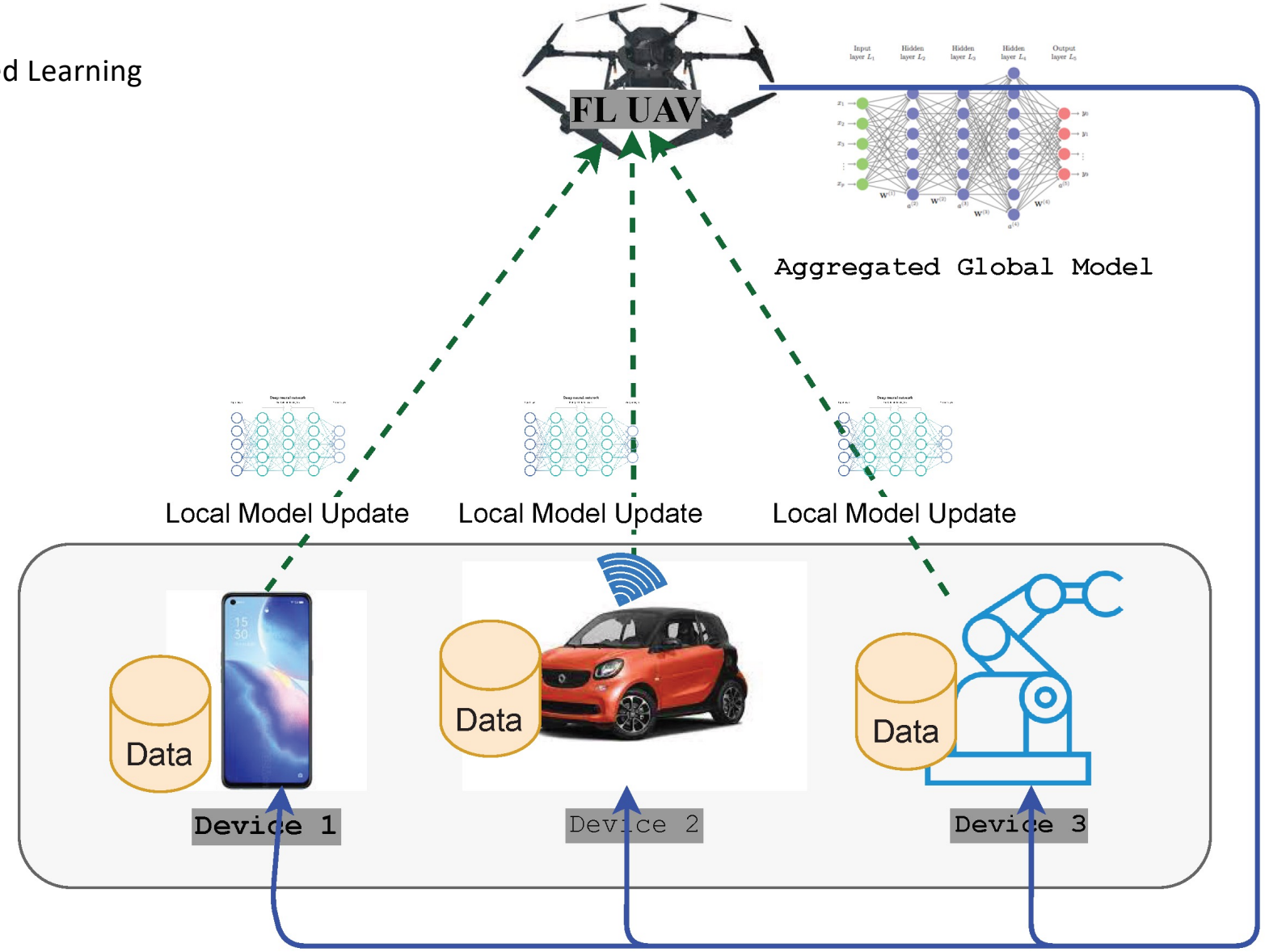
Data Warehouses contain historic and relational data, such as transaction systems, operations etc

CPUs are ideal for performing sequential tasks quickly, while GPUs use parallel processing to compute tasks simultaneously with greater speed and efficiency.



CPU	GPU
Central Processing Unit	Graphics Processing Unit
4-8 Cores	100s or 1000s of Cores
Low Latency	High Throughput
Good for Serial Processing	Good for Parallel Processing
Quickly Process Tasks That Require Interactivity	Breaks Jobs Into Separate Tasks To Process Simultaneously
Traditional Programming Are Written For CPU Sequential Execution	Requires Additional Software To Convert CPU Functions to GPU Functions for Parallel Execution

Federated Learning



Thank you.