CSU34041

Introduction to NoSQL

Yvette Graham
ygraham@tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Student Online Teaching Advice Notice

The materials and content presented within this session are intended solely for use in a context of teaching and learning at Trinity.

Any session recorded for subsequent review is made available solely for the purpose of enhancing student learning.

Students should not edit or modify the recording in any way, nor disseminate it for use outside of a context of teaching and learning at Trinity.

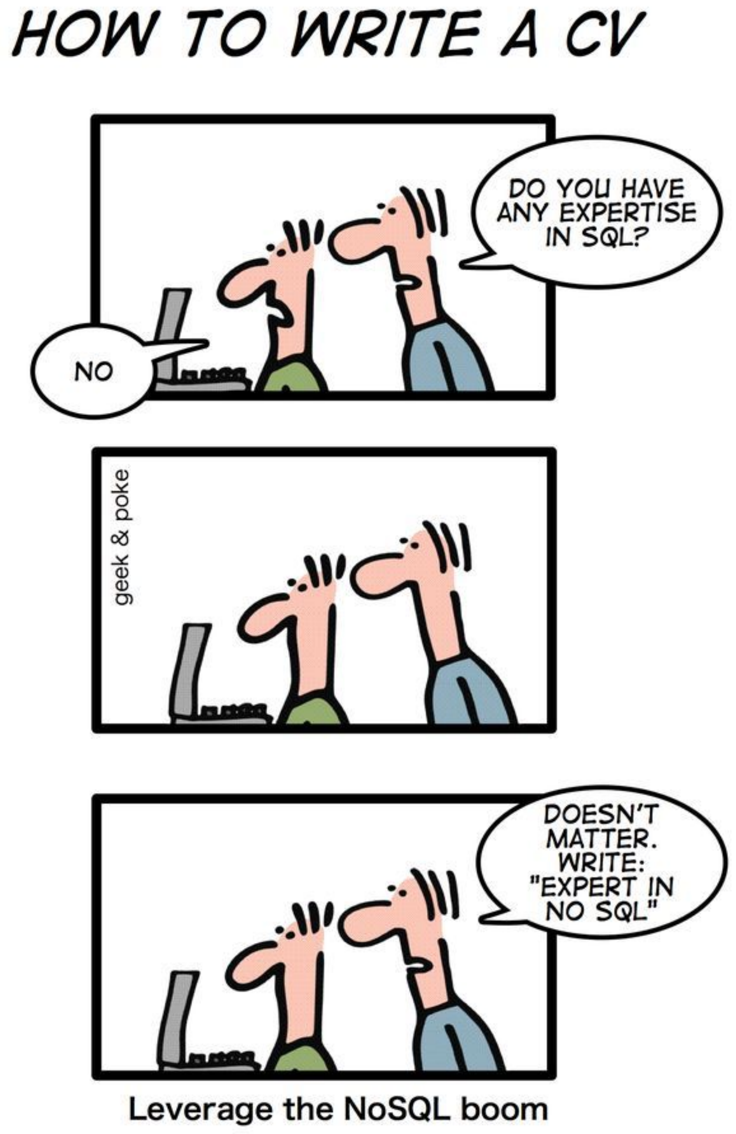
Please be mindful of your physical environment and conscious of what may be captured by the device camera and microphone during videoconferencing calls.

Recorded materials will be handled in compliance with Trinity's statutory duties under the Universities Act, 1997 and in accordance with the University's policies and procedures.

Further information on data protection and best practice when using videoconferencing software is available at https://www.tcd.ie/info_compliance/data-protection/.

© Trinity College Dublin 2020

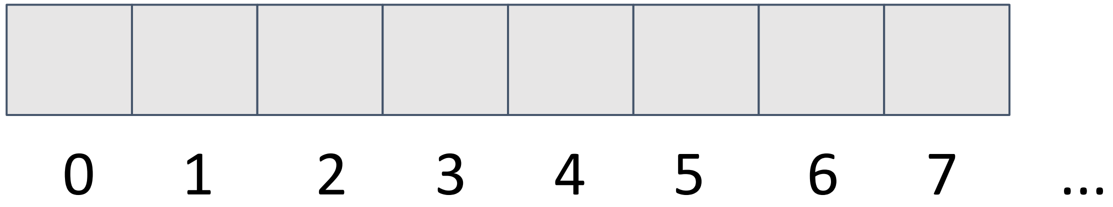
Why?


HOW TO WRITE A CV
DO YOU HAVE ANY EXPERTISE IN SQL?
NO
geek & poke
DOESN'T MATTER. WRITE: "EXPERT IN NO SQL"
Leverage the NoSQL boom

http://geek-and-poke.com/geekandpoke/2011/1/27/nosql.html

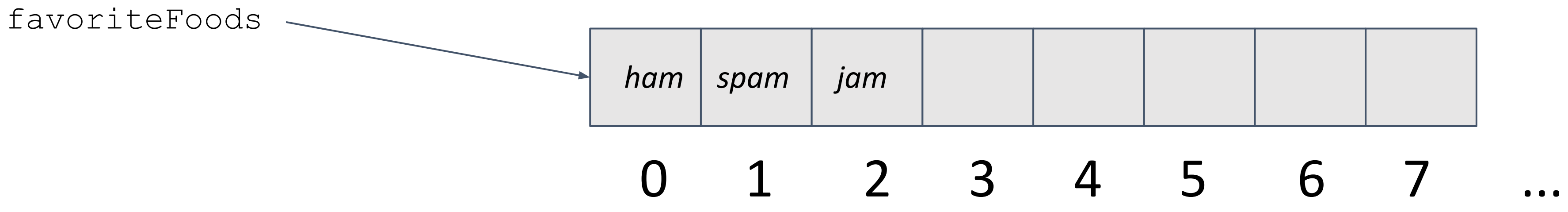
Data Structures

- A **data structure** is a particular way of **organizing data in memory** so that it can be used effectively by software/ computer programs



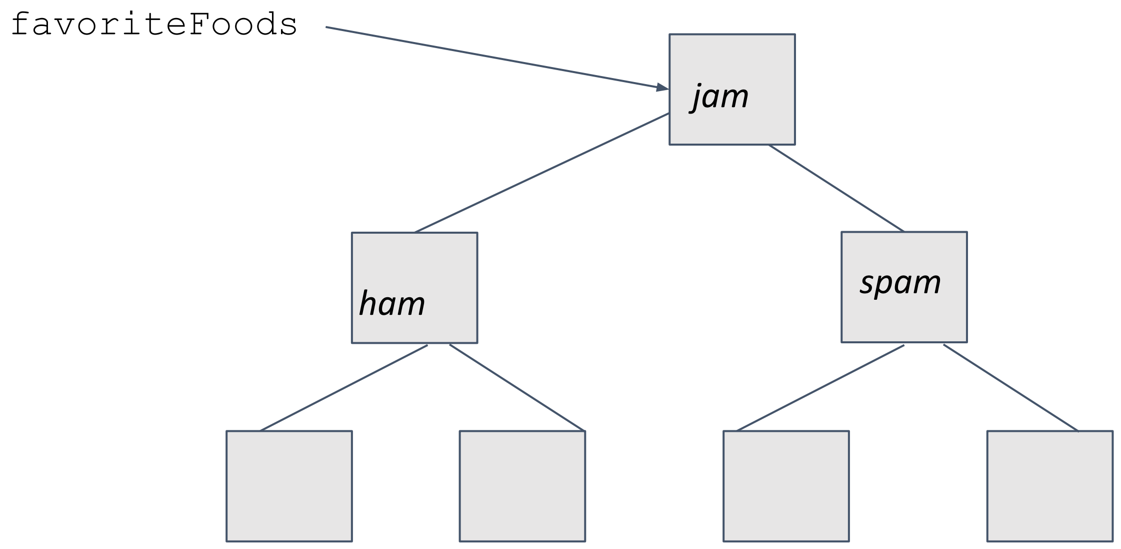
Data Structures

- A **data structure** is a particular way of **organizing data in memory** so that it can be used effectively by software/ computer programs

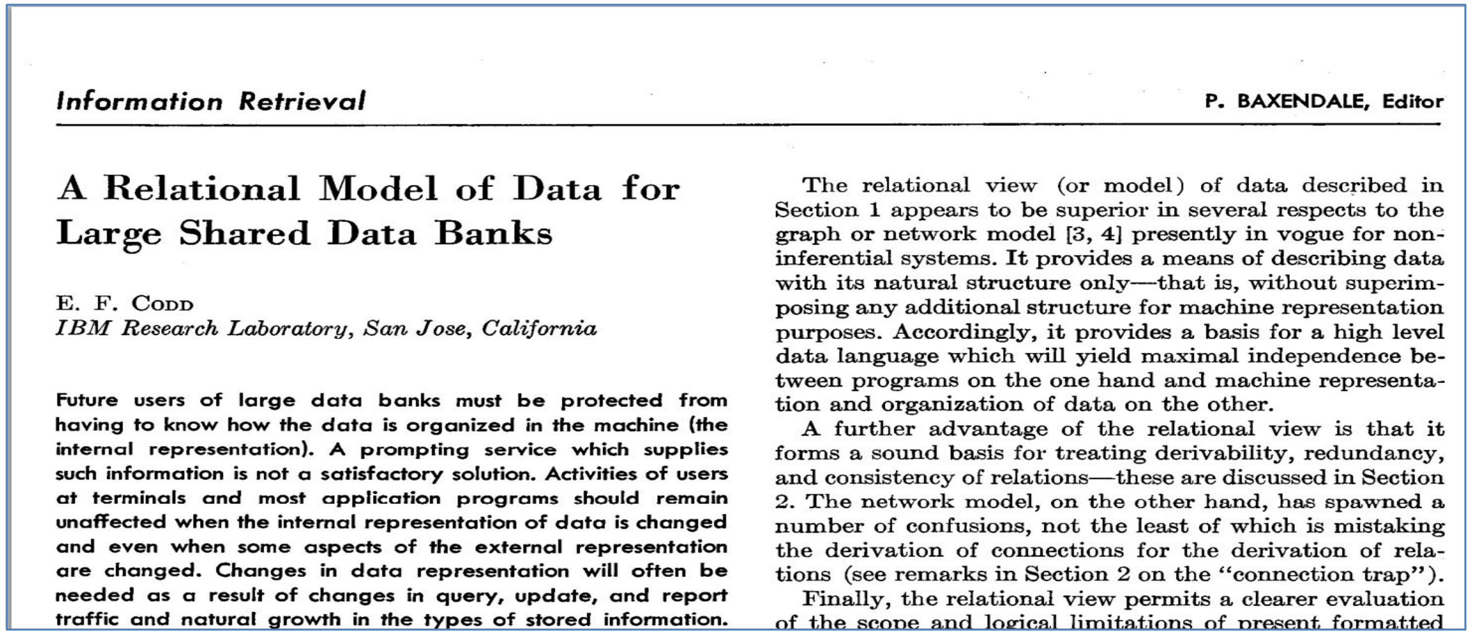


Data Structures

- A **data structure** is a particular way of **organizing data in memory** so that it can be used effectively by software/ computer programs



The relational model : 1970



Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

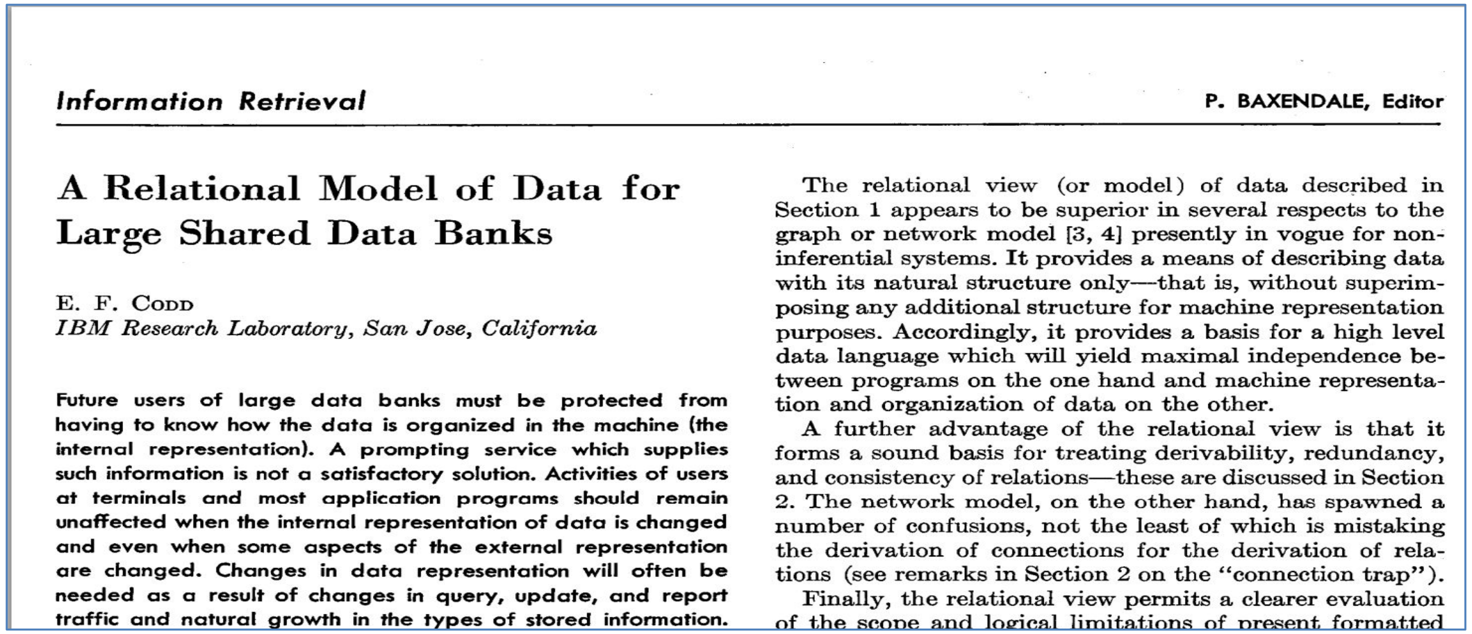
The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted

- Since they first appearance, relational databases have been a default choice in many different context and especially in enterprise applications - Persistence + Concurrency + Integration + Almost standard model

So why do we need anything else?



Information Retrieval P. BAXENDALE, Editor

A Relational Model of Data for Large Shared Data Banks

E. F. Codd
IBM Research Laboratory, San Jose, California

Future users of large data banks must be protected from having to know how the data is organized in the machine (the internal representation). A prompting service which supplies such information is not a satisfactory solution. Activities of users at terminals and most application programs should remain unaffected when the internal representation of data is changed and even when some aspects of the external representation are changed. Changes in data representation will often be needed as a result of changes in query, update, and report traffic and natural growth in the types of stored information.

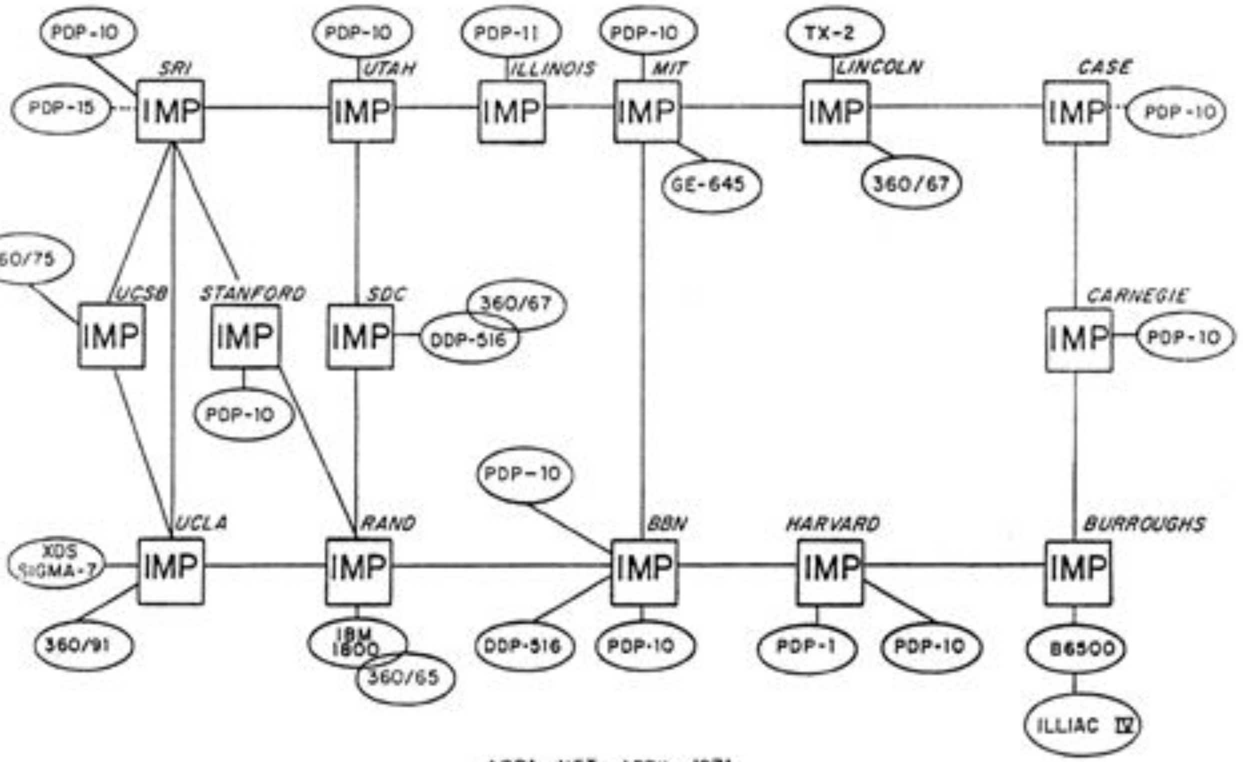
The relational view (or model) of data described in Section 1 appears to be superior in several respects to the graph or network model [3, 4] presently in vogue for non-inferential systems. It provides a means of describing data with its natural structure only—that is, without superimposing any additional structure for machine representation purposes. Accordingly, it provides a basis for a high level data language which will yield maximal independence between programs on the one hand and machine representation and organization of data on the other.

A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy, and consistency of relations—these are discussed in Section 2. The network model, on the other hand, has spawned a number of confusions, not the least of which is mistaking the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

Finally, the relational view permits a clearer evaluation of the scope and logical limitations of present formatted

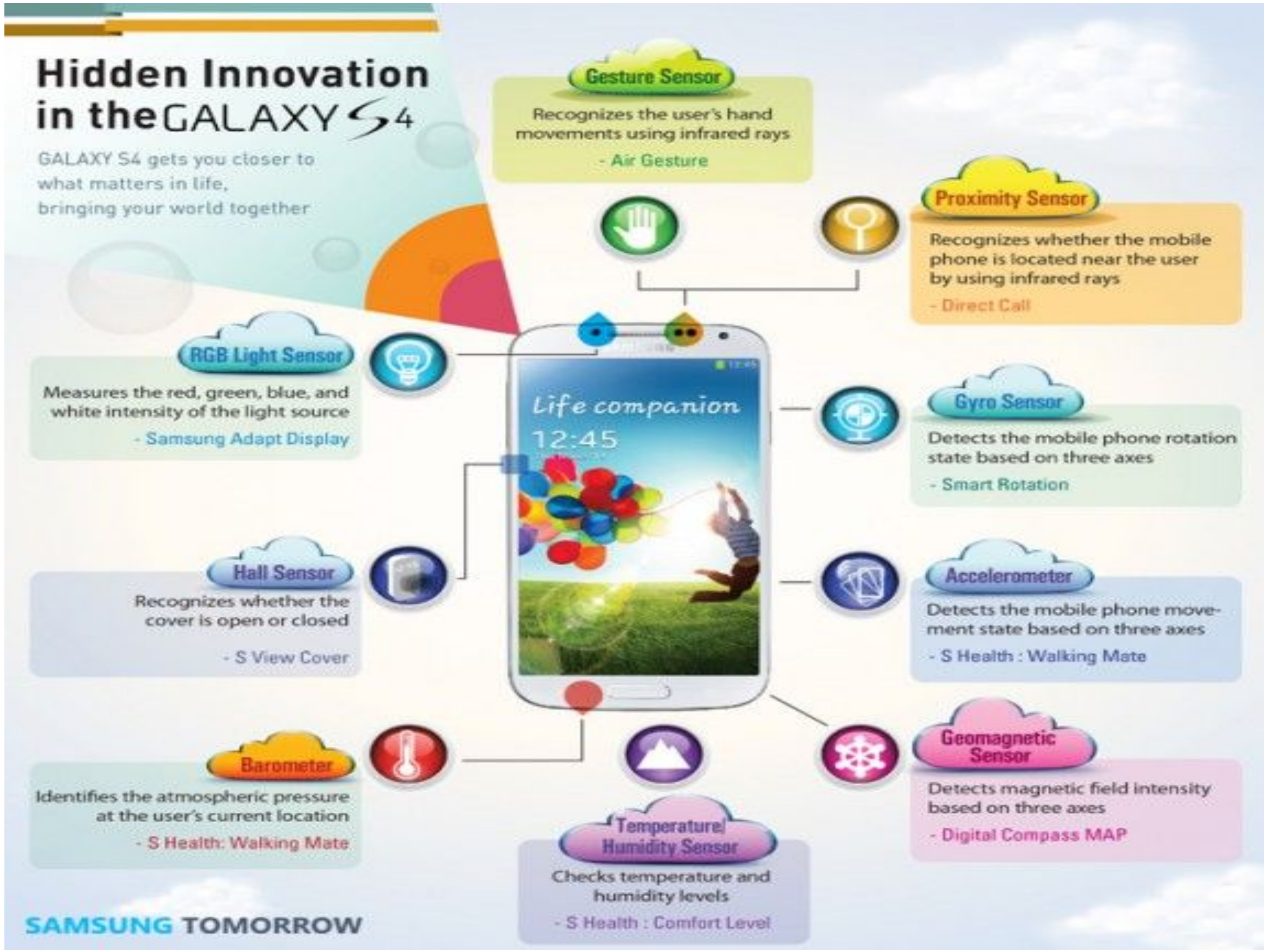
- For application developers, the biggest frustration has been what's commonly called the **impedance mismatch**: the difference between the relational model and the in-memory data structure
- No non-simplistic data structures, such as nested records or lists
- if you want to use a richer in-memory data structure, you have to translate it to a relational representation to store it on disk

The Internet - 1971



ARPA NET, APRIL 1971

The Internet 2014



Hidden Innovation in the GALAXY S4

GALAXY S4 gets you closer to what matters in life, bringing your world together

Gesture Sensor
Recognizes the user's hand movements using infrared rays
- Air Gesture

Proximity Sensor
Recognizes whether the mobile phone is located near the user by using infrared rays
- Direct Call

RGB Light Sensor
Measures the red, green, blue, and white intensity of the light source
- Samsung Adapt Display

Gyro Sensor
Detects the mobile phone rotation state based on three axes
- Smart Rotation

Hall Sensor
Recognizes whether the cover is open or closed
- S View Cover

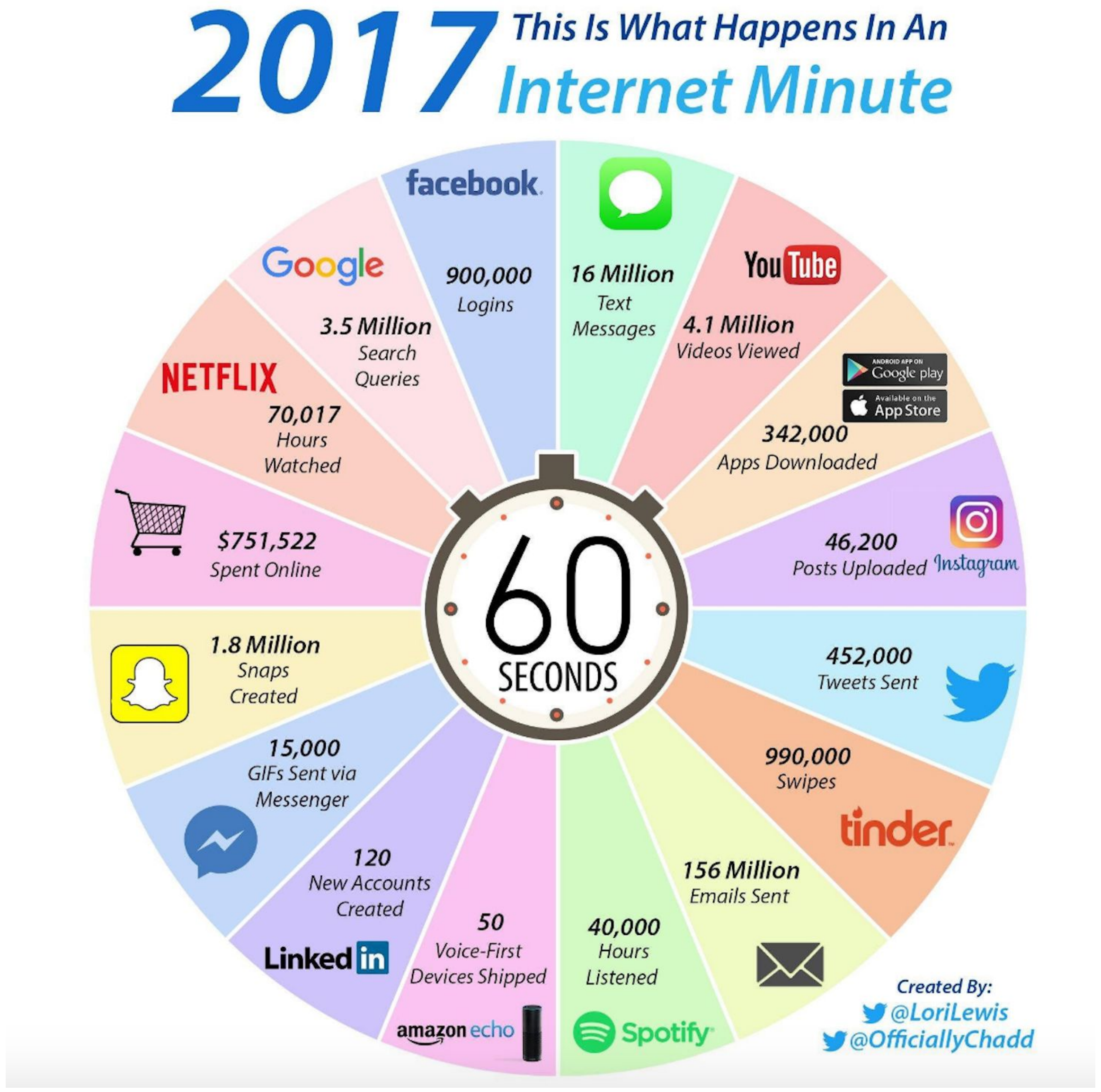
Accelerometer
Detects the mobile phone movement state based on three axes
- S Health : Walking Mate

Barometer
Identifies the atmospheric pressure at the user's current location
- S Health: Walking Mate

Temperature/ Humidity Sensor
Checks temperature and humidity levels
- S Health : Comfort Level

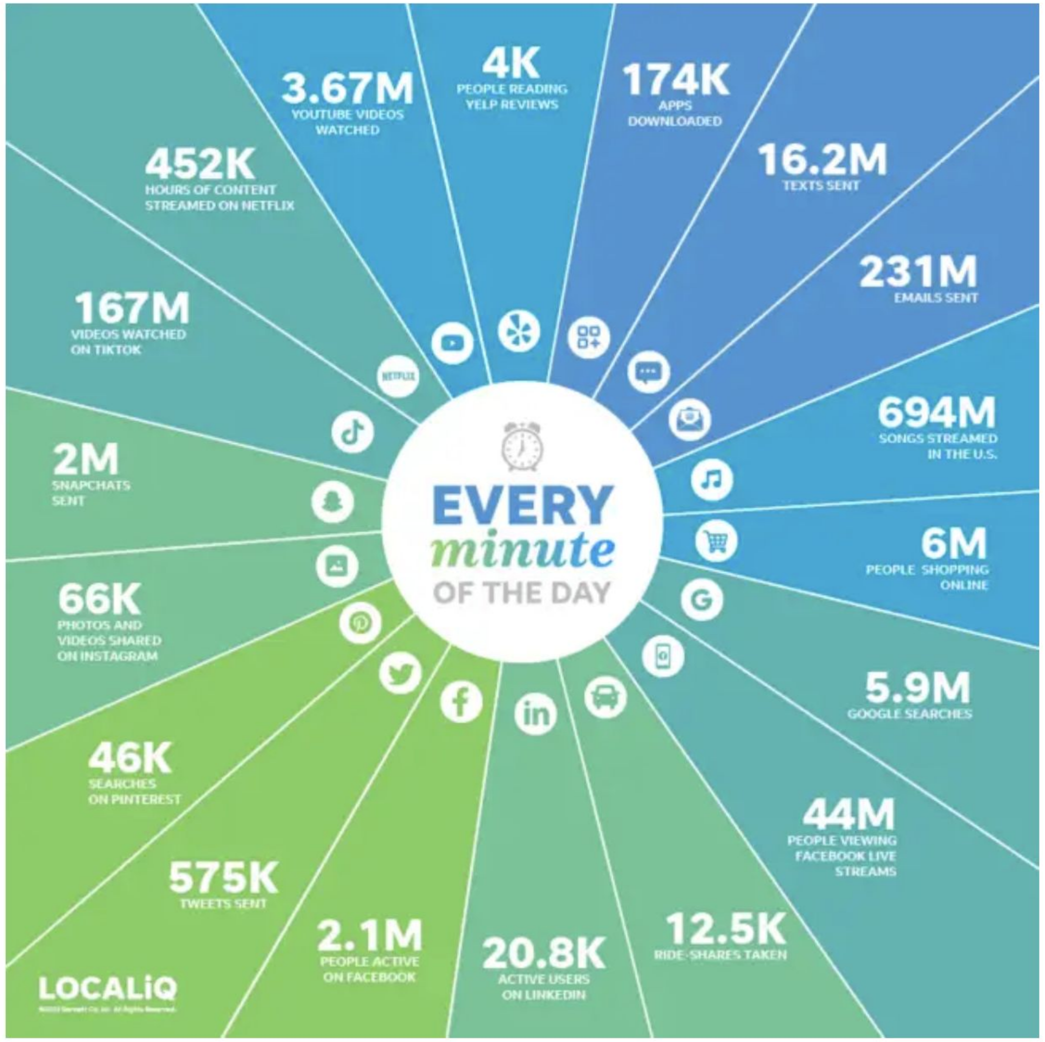
Geomagnetic Sensor
Detects magnetic field intensity based on three axes
- Digital Compass MAP

Trend 1: Data Size

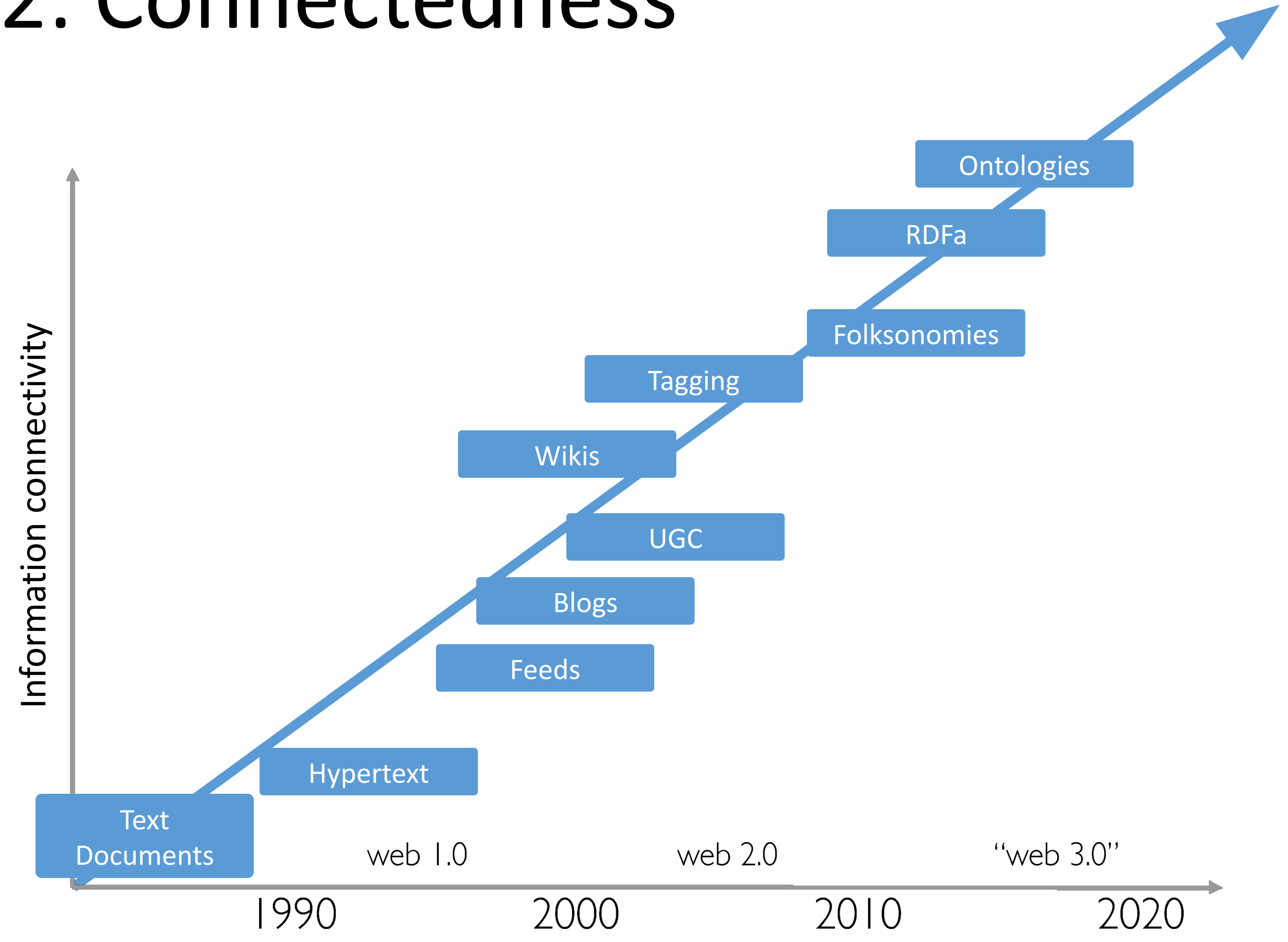


http://www.visualcapitalist.com/happens-internet-minute-2017/

Trend 1: Data Size



Trend 2: Connectedness

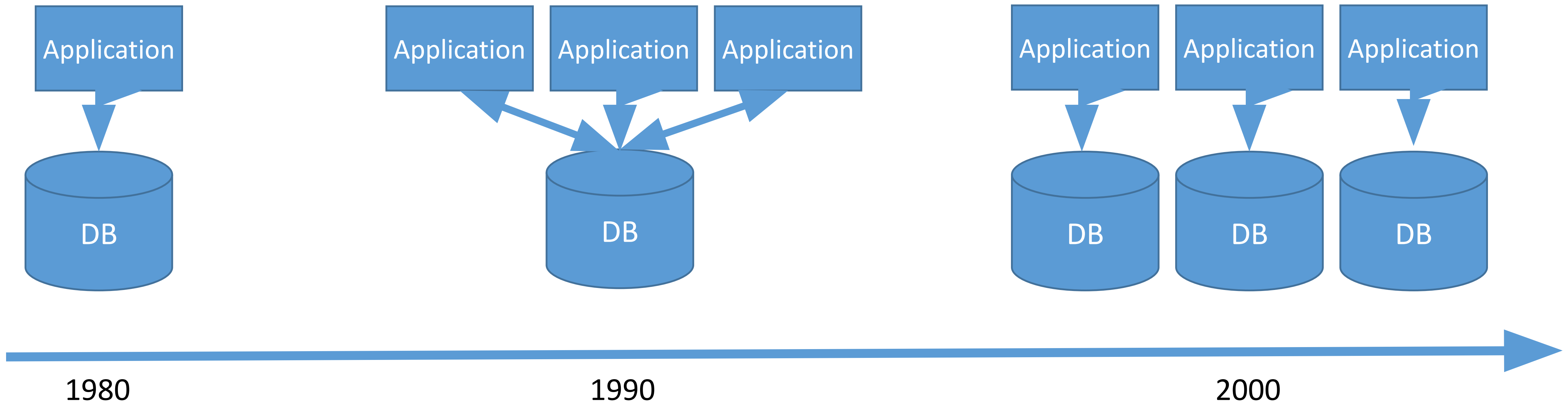


Trend 3: Semi-Structure

- Individualisation of content
  - In the salary lists of the 1970s, all elements had exactly one job
  - In the salary lists of the 2000s, we need 5 job columns! Or 8? Or 15?

- All encompassing “entire world views”
  - Store more data about each entity
- Trend accelerated by the *decentralization* of *content generation* that is the hallmark of the age of participation (“web 2.0”)

Trend 4: Architecture



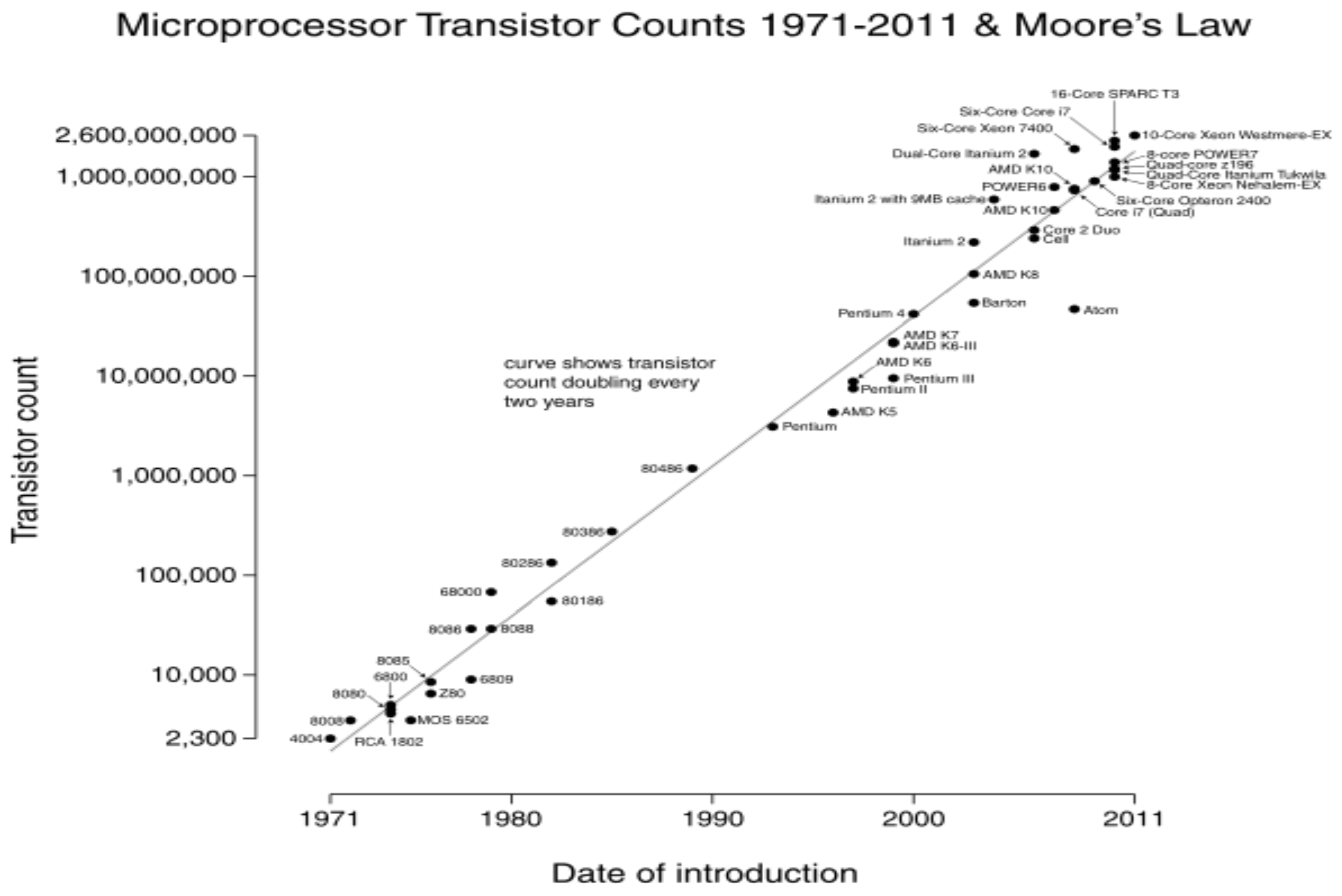
Mainframe application

Database as Integration hub

Decoupled services

Multicore
Parallelization/distributed
Cloud
Schema less

Moore’s Law



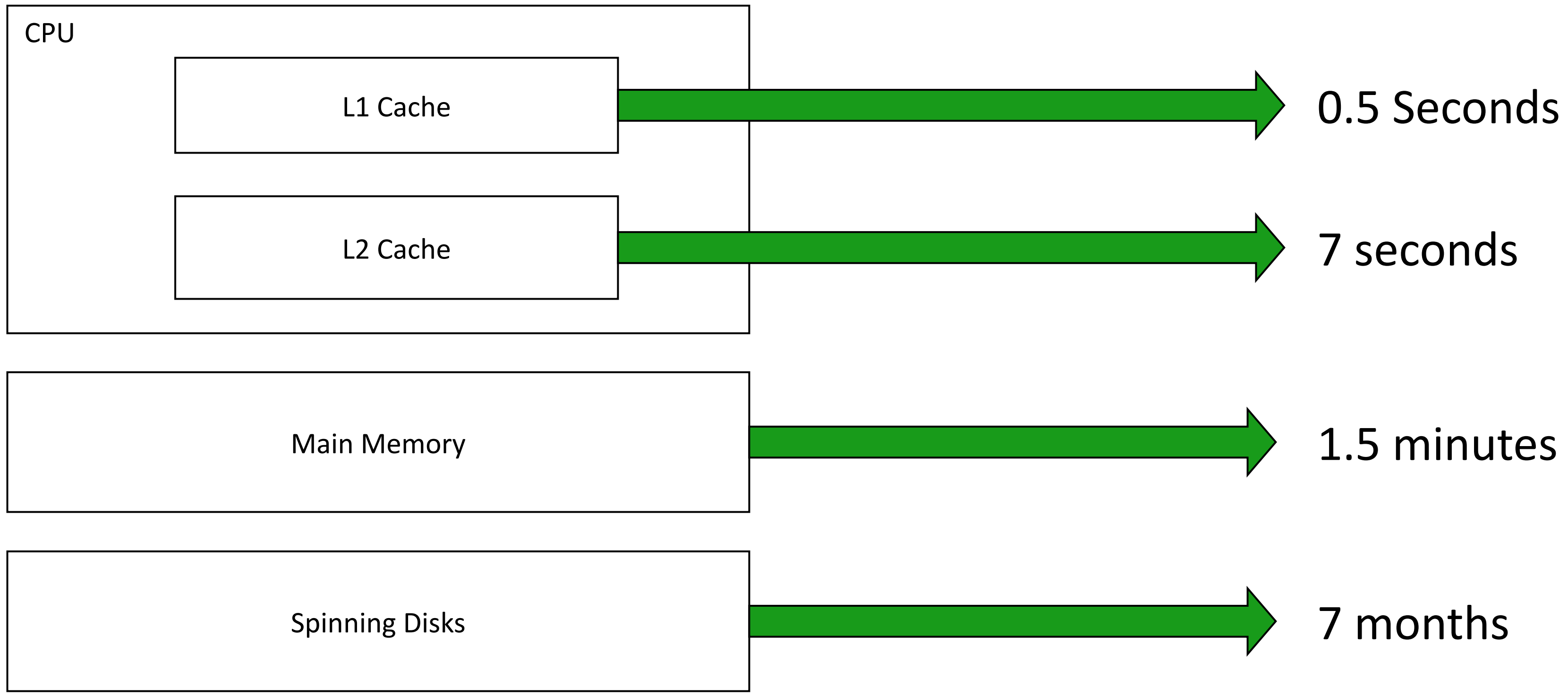
- **Moore's law** is the observation that the number of <u>transistors</u> in a dense <u>integrated circuit</u> doubles about every two years

Limits to Moore’s Law

- Great for chip/transistor based technologies
- But has had little impact on Disks
- Disks are still dog slow
  … Like really, really, really slow
- Slower than molasses on a cold day in Alaska?
- Even slower than that
- How slow….

How Slow is a Disk*

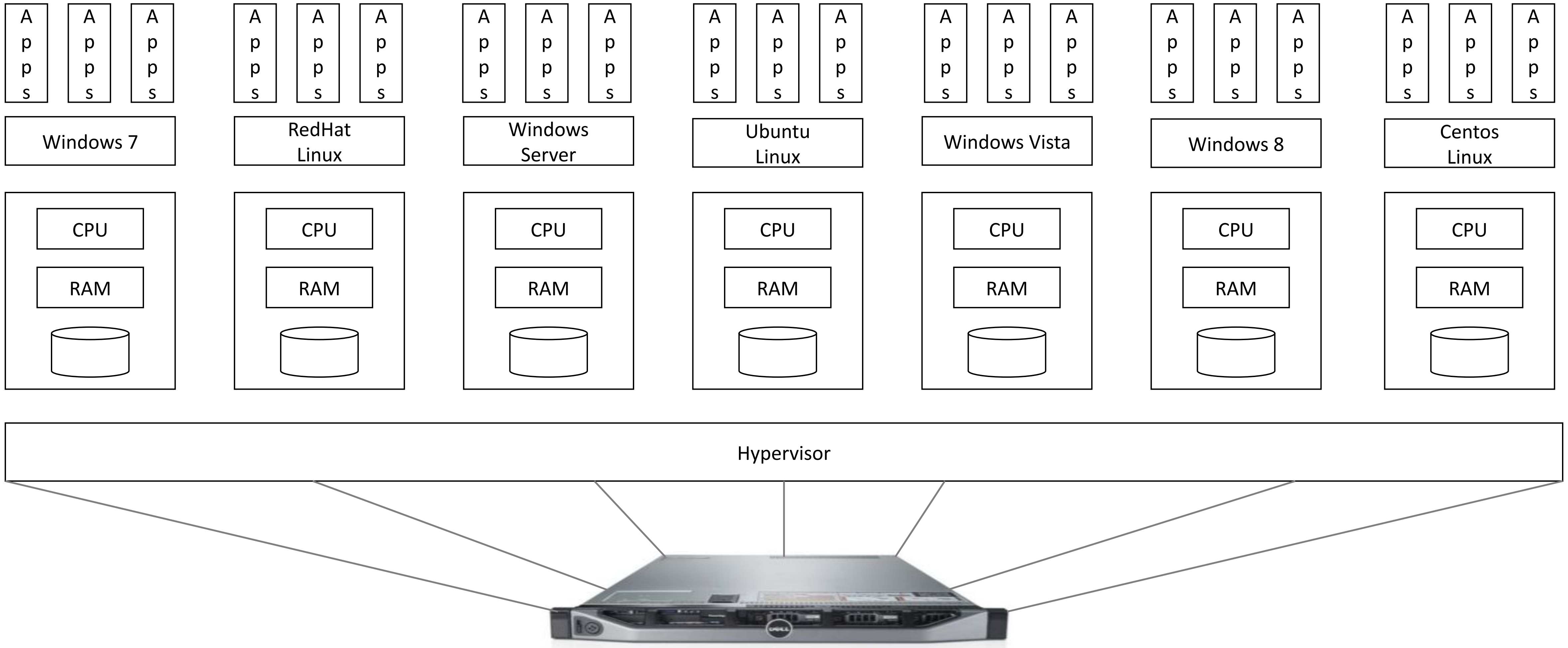
- A cache is a smaller, faster memory, located closer to processor core



- cache usually split into levels for different purposes

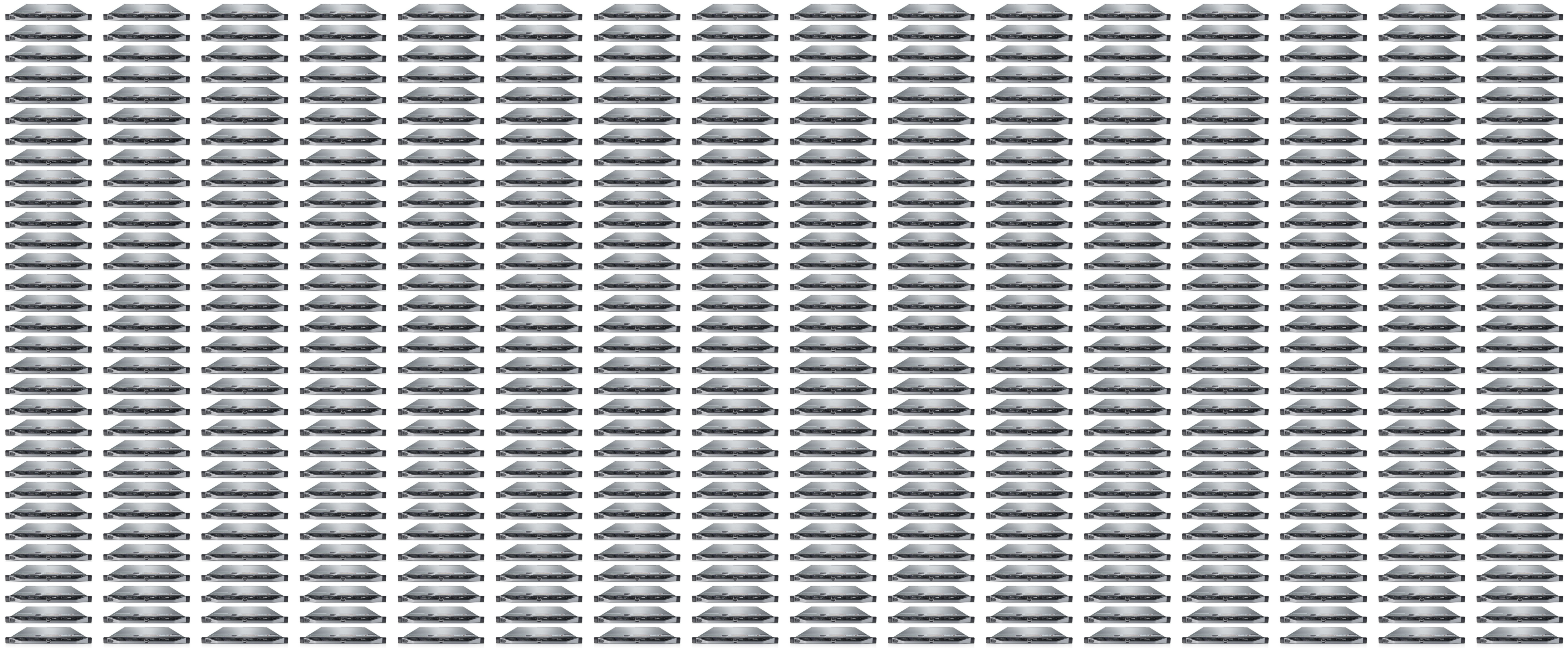
* http://norvig.com/21-days.html#answers

Virtualization



- Make lots of copies of a O/S. share the hardware
- Each virtual machine runs its own operating system and functions separately from the other VMs, even when they are all running on the same host

Now Imagine Lots



Cloud Computing

- Takes virtualization to the extreme
- Companies like Google, Amazon and Microsoft have over 500,000 physical servers
- Anyone with a credit card can start hundreds of servers in a matter of minutes
- Especially used for data storage and computing power, without direct active management by the user

Why NoSQL

V
elocity
ariety
olume

- **Explosion** of (unstructured) data
- Big data is data that **exceeds the processing capacity** of conventional database systems
- The data is too big, moves too fast, or **doesn’t fit** the structures of your database architectures
- To gain value from this data, you need **alternative way to process it**

Why NoSQL

- **Big data**, became viable as cost-effective approaches have emerged to tame the volume, velocity and variability of massive data
- Within this data lie valuable **patterns** and **information**, previously hidden because of the amount of work required to extract them
- Here we are storing **huge amount of data** - need a processing system to handle and analyse the data in efficient manner
- Data is growing
- We need to handle the huge **Volume** and **Variety** of data with **Velocity - 3 Vs**

Unlock Your Big Data

Traditional Data Approaches

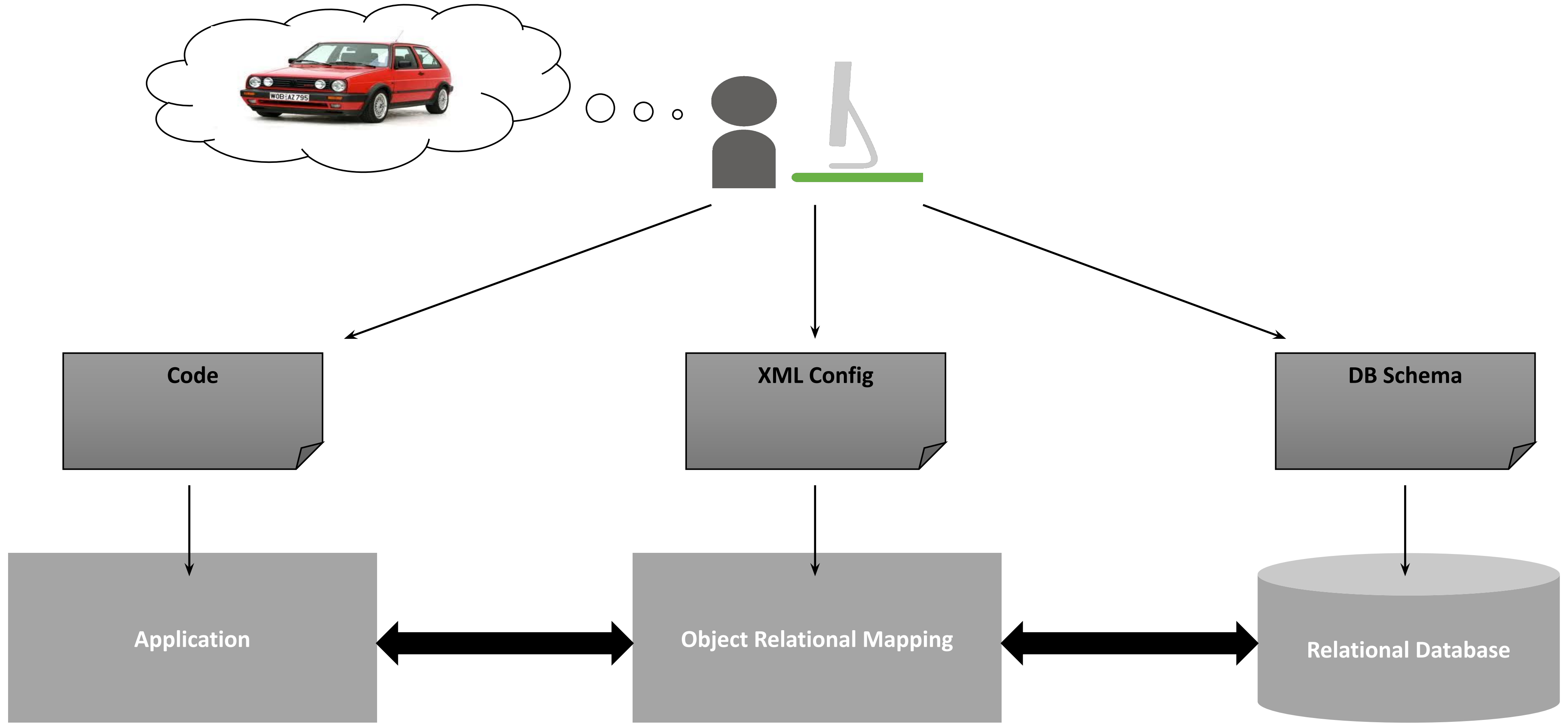
- **Filter – Store – Distribute**
    - Encyclopedias
    - Newspapers
    - Libraries
    - Banking
- Why?
    - Storage caps
    - Bandwidth caps

Store Everything Is A Challenge

SQL Databases

- Depend on a pre-filter
- Assume single disk farm
- Hard to partition
- Based on 1970's storage assumptions

This makes software development difficult - **impedance mismatch**: the difference between the relational model and the in-memory data structures



Limitations of Relational Databases

- Impedance Mismatch - complex objects are not suited to being represented in a relational way
- Application and integration - the database works as a integration database. However, in this scenario the structure of the database tend to be more complex
- Scale up vs. scale out (parallel)

The AWS Disruption - 2006

- Cost per GB/Month for Stable Storage
  - ~$5 per GB down to 2 cents per GB
- Unlimited Storage
- Purchased in GB chunks
- Pay only for what you use

What did possibility of scaling out result in?

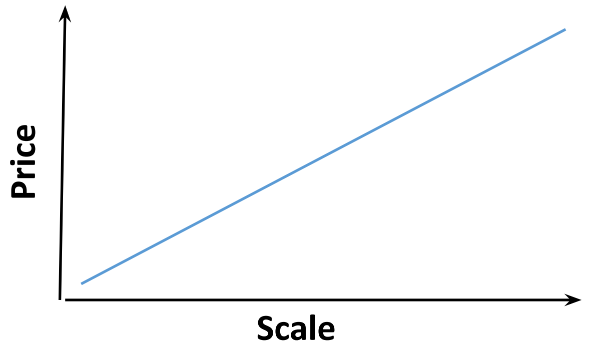
- No CAPEX - capital expenditure - funds used by a company to acquire or upgrade physical assets such as property, industrial building or equipment
- No Data Centre
- Availability of Scale
- Utility Pricing (pay per use)

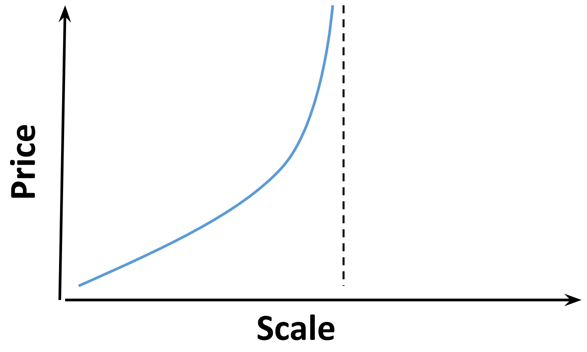
Filter-Store-Distribute

Store-Filter-Distribute

NoSQL Scales Better

Vs.





Scaling Better

- When you face “big data” you have two alternatives, create bigger computer or put together a lot of them
- There were two choices: scaling up or scaling out
- Scaling up implies bigger machine, more processors, disk storage and memory ($$$)
- Scaling out –horizontal scaling- based on data partitioning (divide the database across multiple machine) was the alternative, use a lot of machine in a cluster, it is cheaper and also more resilient (tolerance toward individual failures) => Problems: relational database not designed to run on cluster

- NoSQL It is born out of a need to handle large data volumes - It forces a fundamental shift to building large hardware platform through clusters of commodity servers.
- This need raises from the difficulties of making application code play well with relational databases

Scaling Better

- Clustered databases (Oracle RAC or Microsoft SQL Server) work on a concept of shared disk subsystem
- RDBM can also run on separate server with different sets of data (Sharding) However it needs an **application to control the shared database**
- At the same time we lose querying power, referential integrity, transaction or consistency cross shard. All made worse by licensing cost.
- “bend Relational Databases in many direction that - does not make any sense anymore to have them

When?

Emergence of NoSQL

- “NoSQL” first appeared in 90s as the name of an open-source relational database introduced by Carlo Strozzi
  - This is another story…
- Two companies played a key role
  - 2004: Google started the project BigTable. Paper published in 2006
  - 2007: Amazon publishes the research paper on Amazon Dynamo
- Johan Oskarsson decides to find out more about this new tendency to alternative data storage
  - 2009: Meetup in San Francisco
  - #NoSQL is used as simple hashtag to refer to the event

What?

Not
Only SQL

NoSQL – No Definition

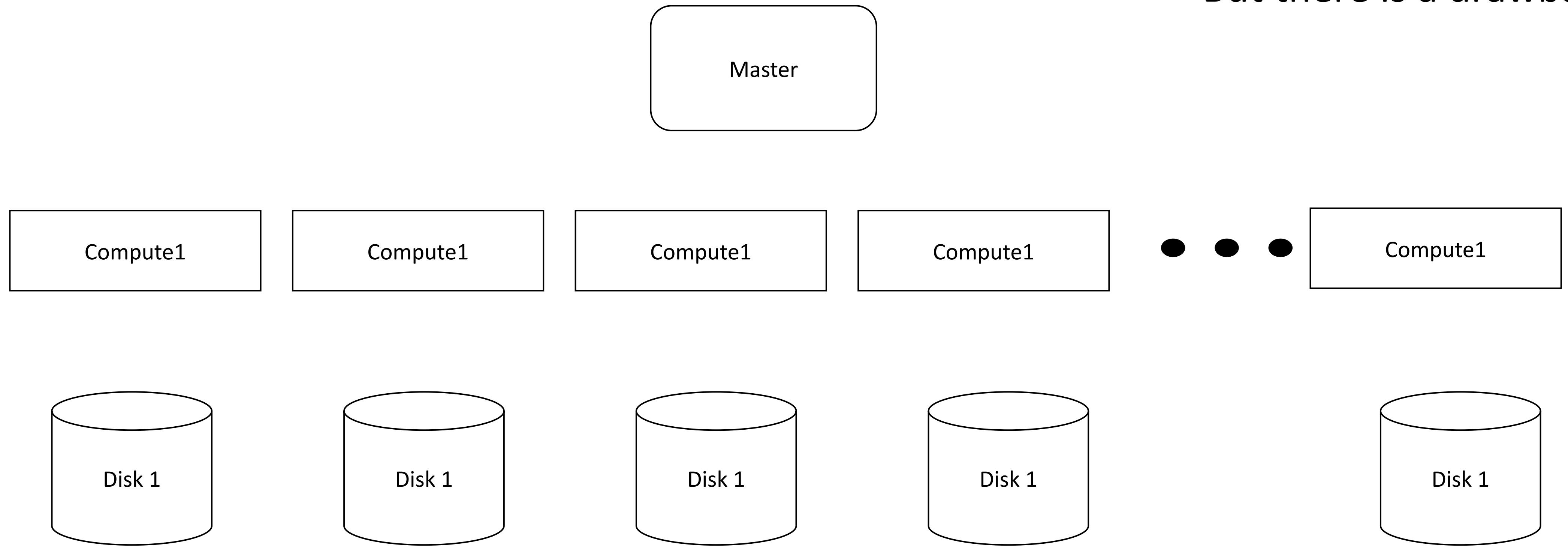
- Non Relational
- No SQL as query language - They don’t use SQL as query language, although some of them may have some form of query language that resembles SQL
- Schema less - structure can change
- Usually –not always – Open-Source project
- They are distributed: they are usually driven by the requirements of running on clusters (with the exception of graph DBs)
- RDBMS use ACID transactions, NoSQL don’t

What is NoSQL?

- Goal of a Database
    - Data Durability
    - Consistent performance
    - Graceful degradation under load
- Big Data Workloads require distributed computing
    - ~~Transactions~~
    - ~~Joins~~

Distributed Computing

- Much more power
- Cheaper
- More resilience
- But there is a drawback -



In distributed system we have a network of autonomous computers that communicate with each other in order to achieve a goal- The computers in a distributed system are independent and do not physically share memory or processor

Distribution Models

Data Distribution

- **Replication** takes the same data and copies it over multiple nodes
  - Master-slave: one node is the primary responsible for processing the update to data while the other are secondaries
  - Peer-to-peer: all replicas have equal weight and can accept writing
- **Sharding** puts different data on different nodes
  - Each server acts as a single source for the subset of data it is responsible for
- Replication and Sharding can be used in combination or alone

Sharding

- Different data on different nodes (horizontal scalability)
- Each server acts as a single source for the subset of data it is responsible for
- Ideal setting: one user talks with one server
  - Data accessed together are stored together
  - Example: access based on physical location, place data to the nearest server
- Many NoSQL databases offer auto-sharding
- Scales read and write on the different nodes of the same cluster
- No resilience if used alone: node failure => data unavailability

Replication

- The same data is replicated and copied over multiple nodes
- Master-slave: one node is the primary responsible for processing the update to data while the other are secondaries used for read operations
  - Scaling by adding slaves
  - Processing incoming data limited by master
  - Read resilience
  - Inconsistency problem (read)

Replication

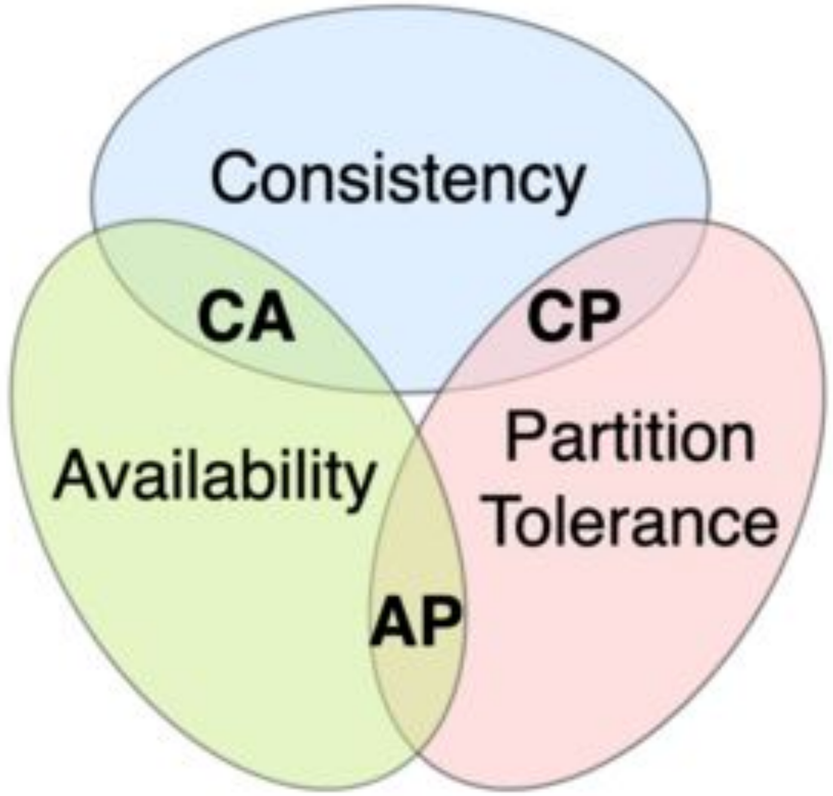
- The same data is replicated and copied over multiple nodes
- Peer-to-peer: all replicas have equal weight and can accept writing
  - Scaling by adding nodes
  - Node failure without losing write capability
  - Inconsistency problem (write)

Combined Approaches

- Master-slave & sharding
  - Multiple master, each data has a single master
- P2P & sharding (common for column-family databases)
  - Form of replication of the shard

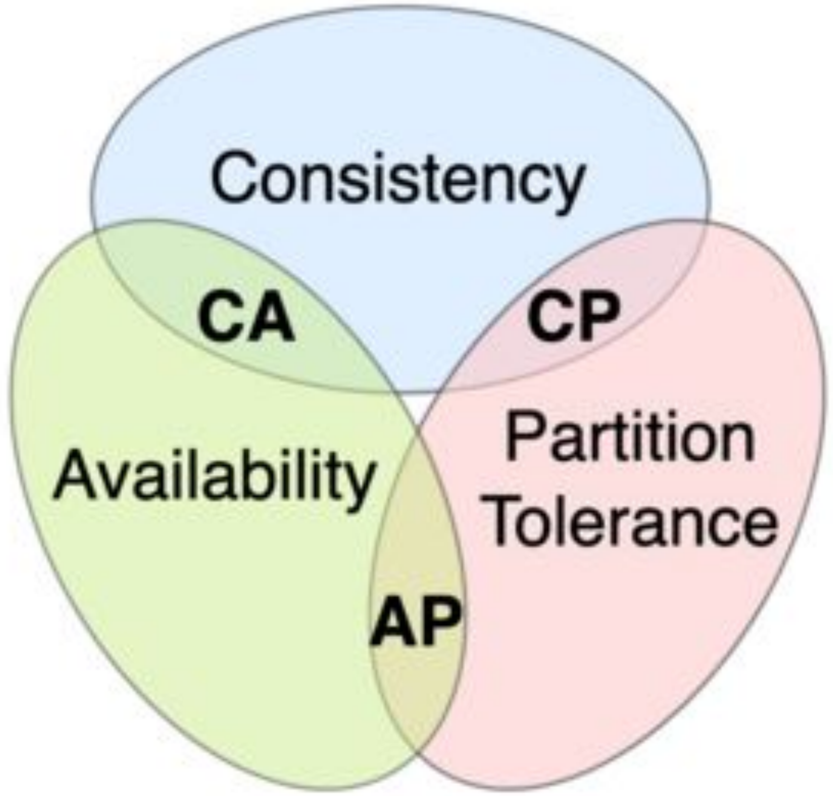
CAP Theorem

CAP Theorem - only 2 of can be guaranteed



- **Consistency**
  - All nodes see the same data at the same time
- **Availability**
  - A guarantee that every request receives a response about whether it succeeded or failed
- **Partition tolerance**
  - The system continues to operate despite arbitrary partitioning due to network failures

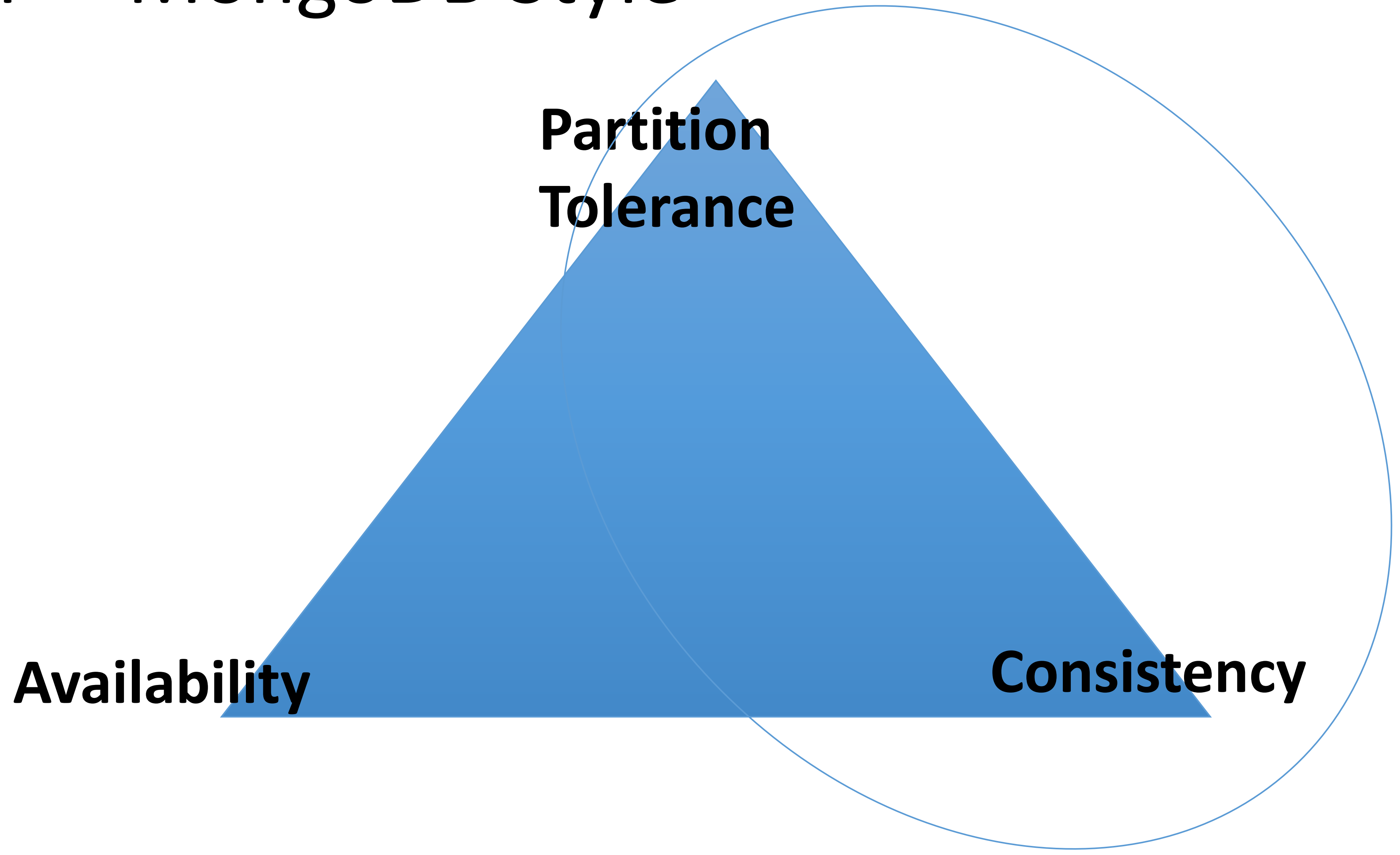
CAP Theorem - only 2 of can be guaranteed



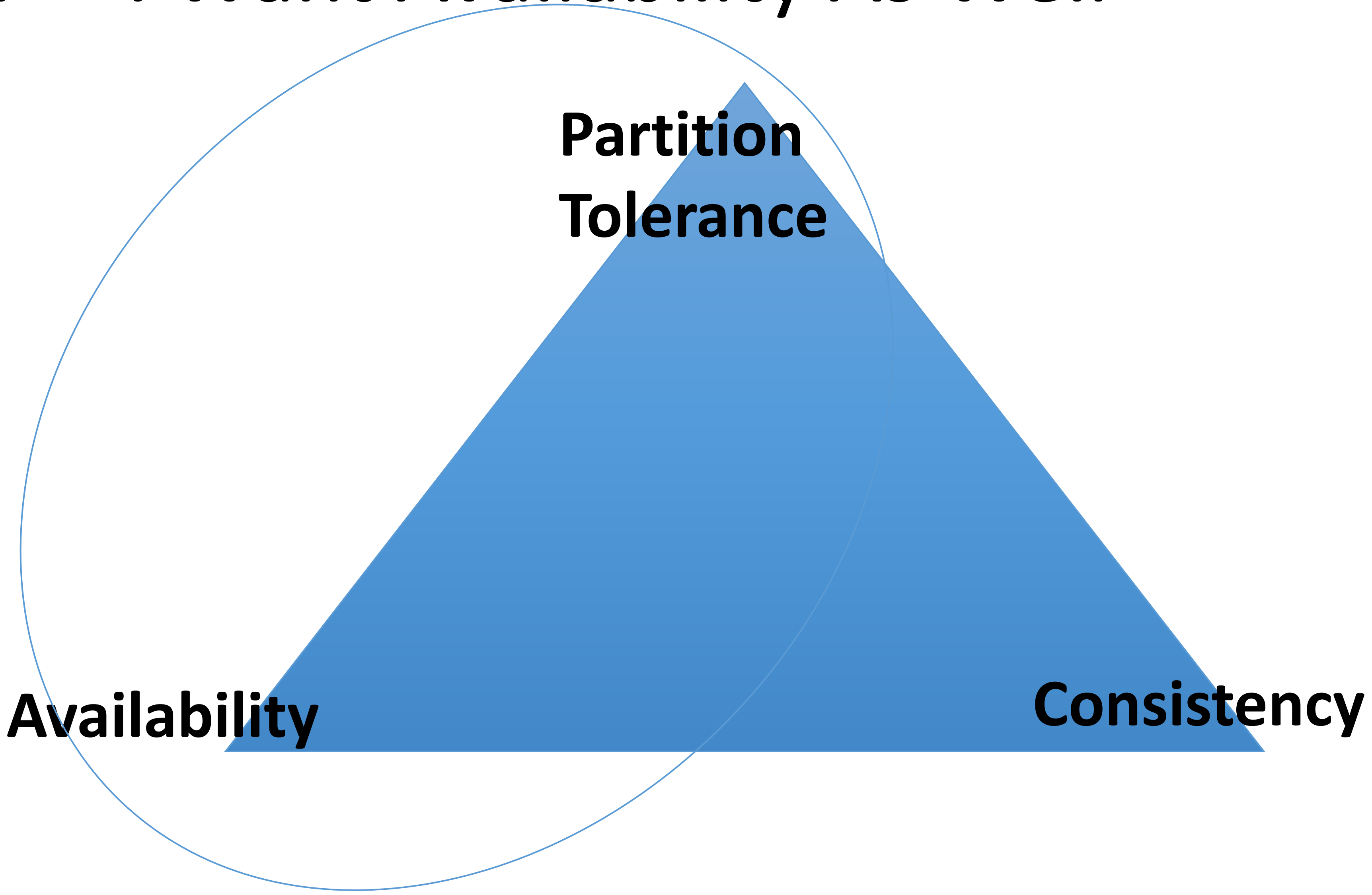
When a network partition failure happens, it must be decided whether to do one of the following:

- cancel the operation and subsequently decrease the **availability** but ensure **consistency**
- proceed with the operation and thus provide **availability** but risk **inconsistency**.

CAP – MongoDB Style



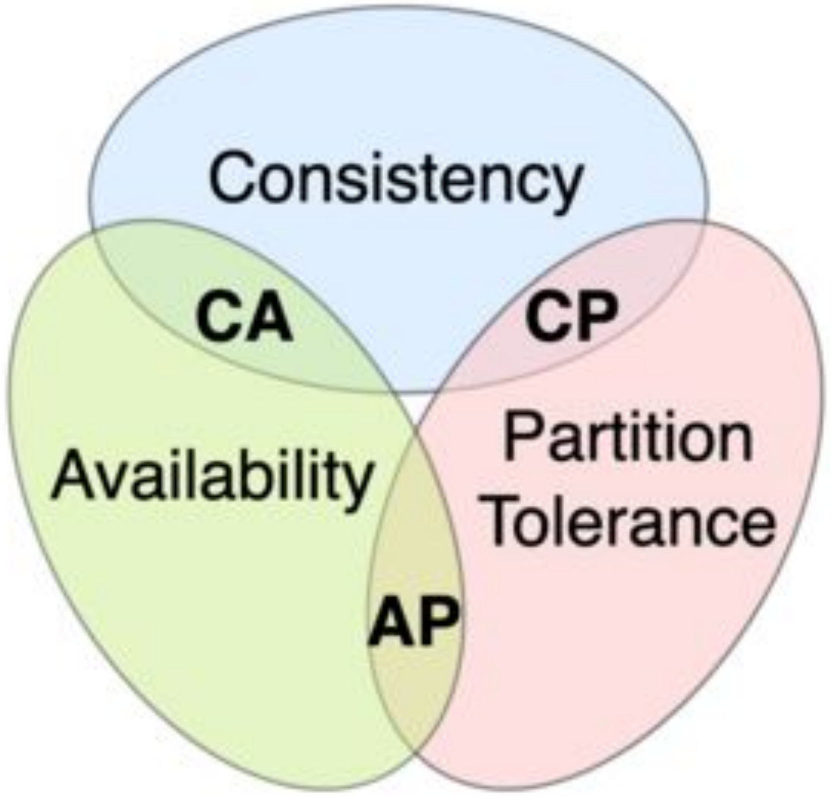
CAP – I Want Availability As Well



Eventual Consistency

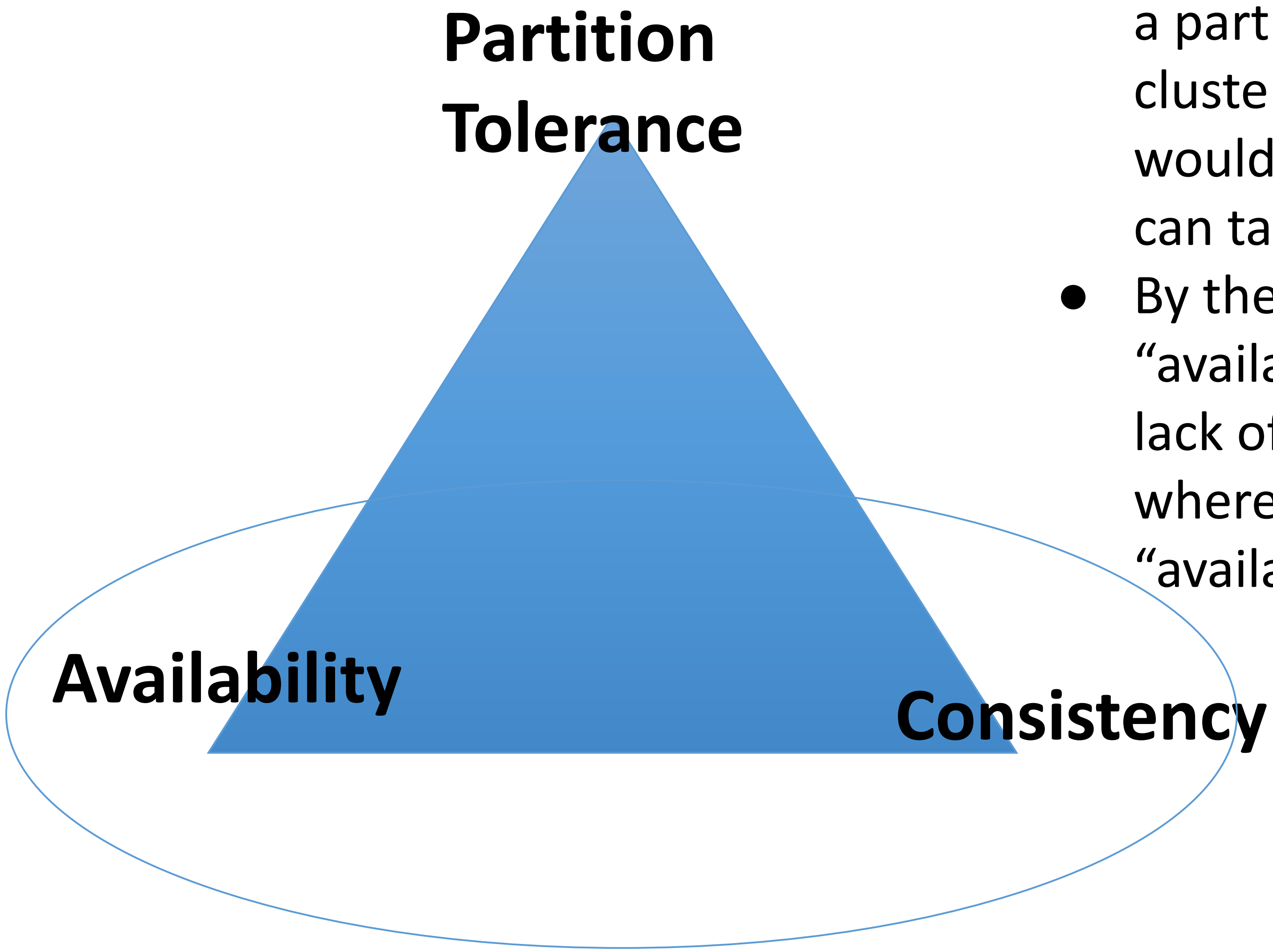
- **Eventual consistency** is a consistency model used in distributed computing to achieve high availability that informally guarantees that, if no new updates are made to a given data item, eventually all accesses to that item will return the last updated value
- Reconciliation is a problem - choosing an appropriate final state when concurrent updates have occurred, called **reconciliation**

But What About CA?



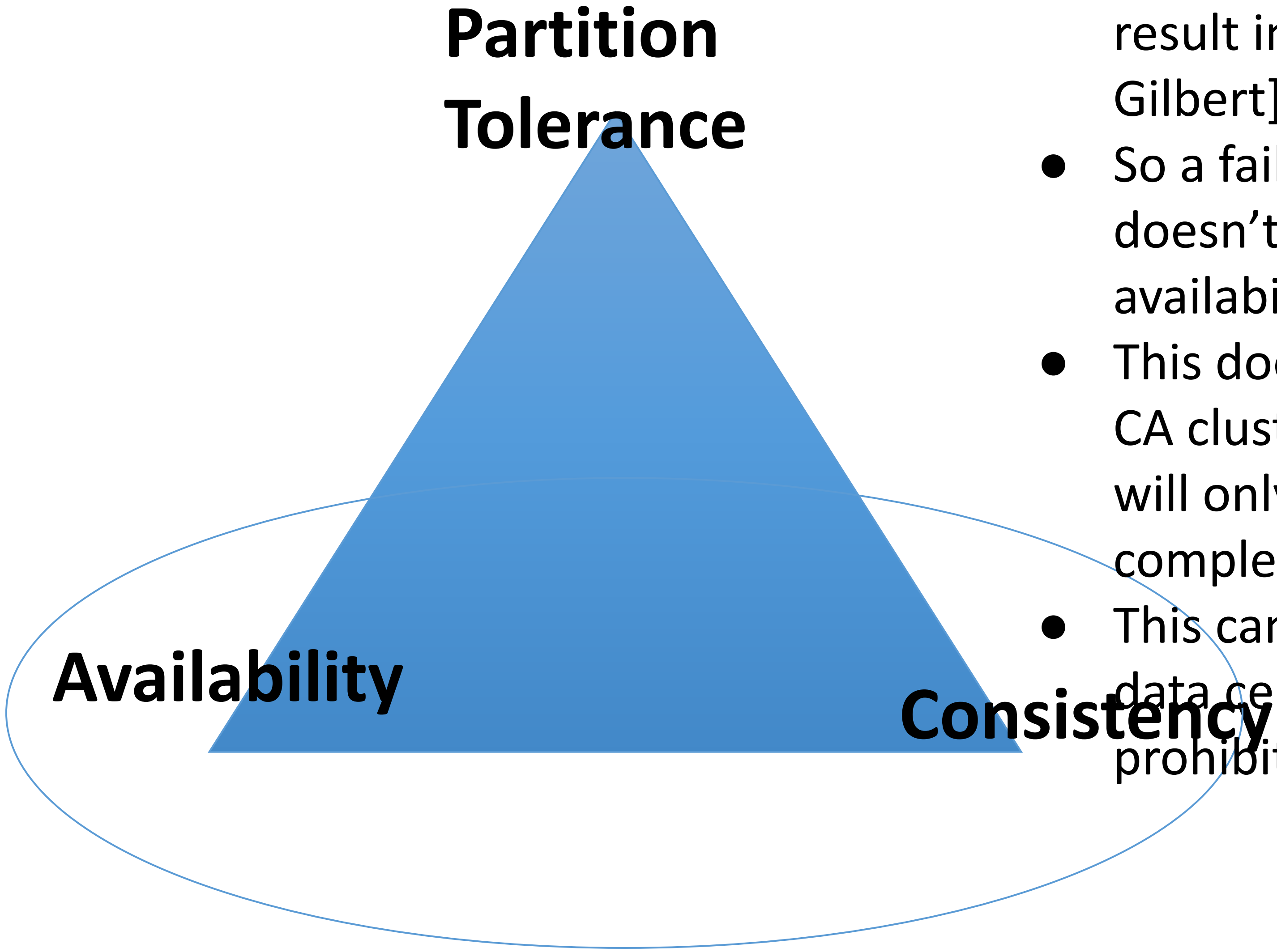
- A single-server system is the obvious example of a CA system—a system that has Consistency and Availability but not
- Partition tolerance: A single machine can't partition, so it does not have to worry about partition tolerance.
- There's only one node—so if it's up, it's available.
- Being up and keeping consistency is reasonable.
- This is the world that most relational database systems live in.

But What About CA?



- It is theoretically possible to have a CA cluster.
- However, this would mean that if a partition ever occurs in the cluster, all the nodes in the cluster would go down so that no client can talk to a node.
- By the usual definition of “available,” this would mean a lack of availability, but this is where CAP’s special usage of “availability” gets confusing.

But What About CA?



- CAP defines “availability” to mean “every request received by a non failing node in the system must result in a response” [Lynch and Gilbert].
- So a failed, unresponsive node doesn’t infer a lack of CAP availability.
- This does imply that you can build a CA cluster, but you have to ensure it will only partition rarely and completely.
- This can be done, at least within a data center, but it’s usually prohibitively expensive.

SQL vs. NoSQL

ACID

- **A**tomic: Everything in a transaction succeeds or the entire transaction is rolled back
- **C**onsistent: A transaction cannot leave the database in an inconsistent state
- **I**solated: Transactions cannot interfere with each other
- **D**urable: Completed transactions persist, even when servers restart etc.

BASE

- **B**asic **A**vailability: An application works basically all the time
- **S**oft-state: It does not have to be consistent all the time
- **E**ventual consistency: It will be in some known-state state eventually

Each node is always available to serve requests. As a trade-off, data modifications are propagated in the background to other nodes. The system may be inconsistent, but the data is still largely accurate.

Voldemort (LinkedIn), Amazon SimpleDB,
Memcache,
BerkeleyDB, Oracle NoSQL

Neo4j,
InfiniteGraph,
OrientDB, Titan
GraphDB

Data Model

Cassandra, Hbase, Amazon Redshift,
HP Vertica, Teradata

Couchbase, MongoDB, RavenDB,
ArangoDB, MarkLogic,-bash:
Couchbase, OrientDB, RavenDB,
Redis, RethinkDB

Data Model

- A data model is a representation of how we perceive and manipulate our data
- The data model describes how we interact with the data
  - Represents the data elements under analysis
  - How these elements interact with each others

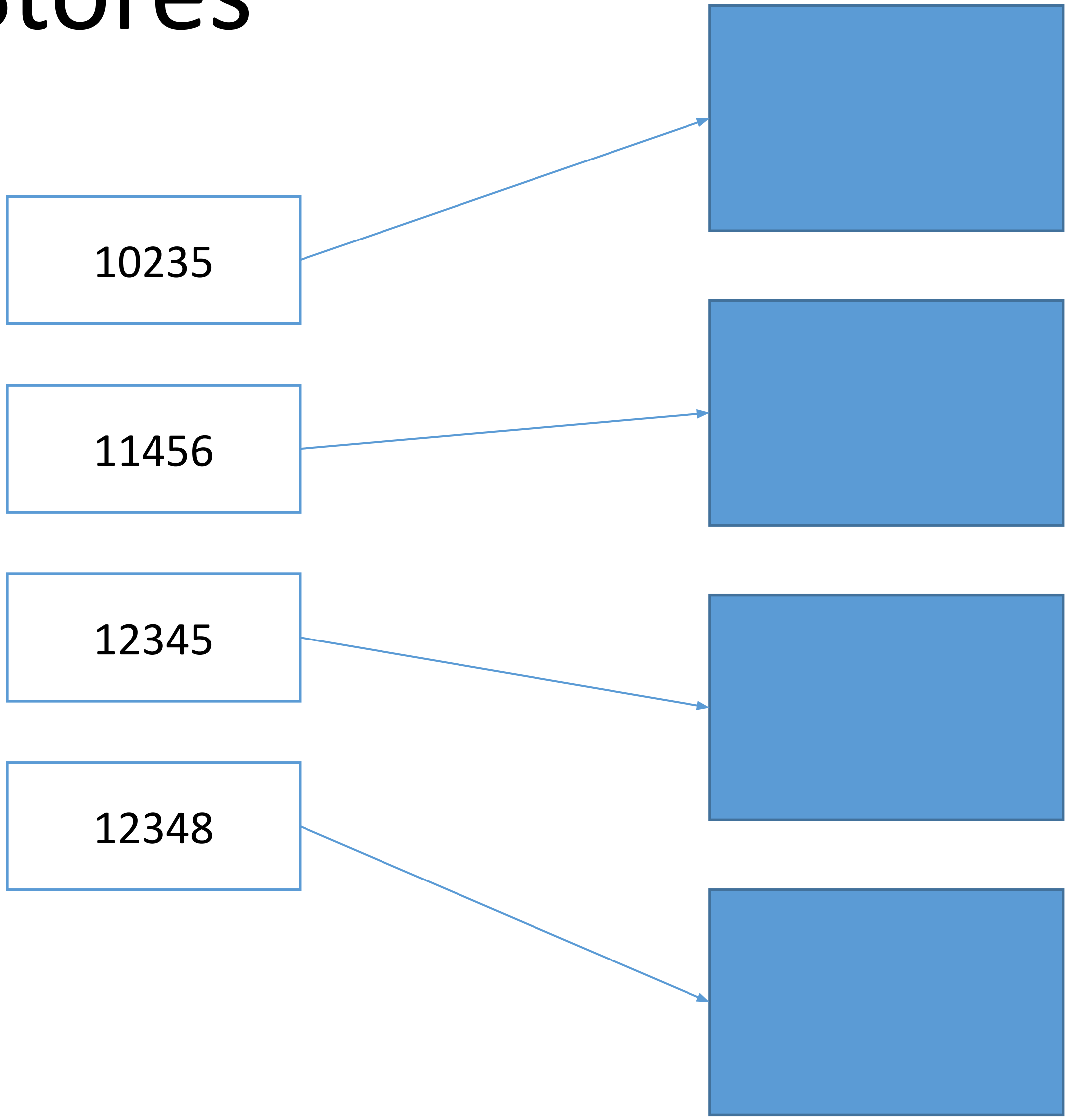
- The storage model describes how the database stores and manipulates the data internally

Four Common Types of NoSQL

- Key Value Stores
- Document Stores
- Column Stores
- Graph Stores

- Note
    - Lots of Hybrids

Key-Value Stores



Key-Value Stores

- Maps keys to values
- Values treated as a blob
  - They can be complex compound objects (list, maps, or other structures)
- Single Index
- Consistency applicable for operations on a single key
- Very fast and scalable
- Inefficient to do aggregate queries, "all the carts worth $100 or more" or to represent relationships between data
- Great for shopping carts, user profiles and preferences, storing session information

Document Databases

```
{"id": 10203,
 "name":"Sara",
 "surname": "Parker",
 "items": [
 {"product_id": 23, "quantity": 2},
 {"product_id": 45, "quantity": 1}]}
```

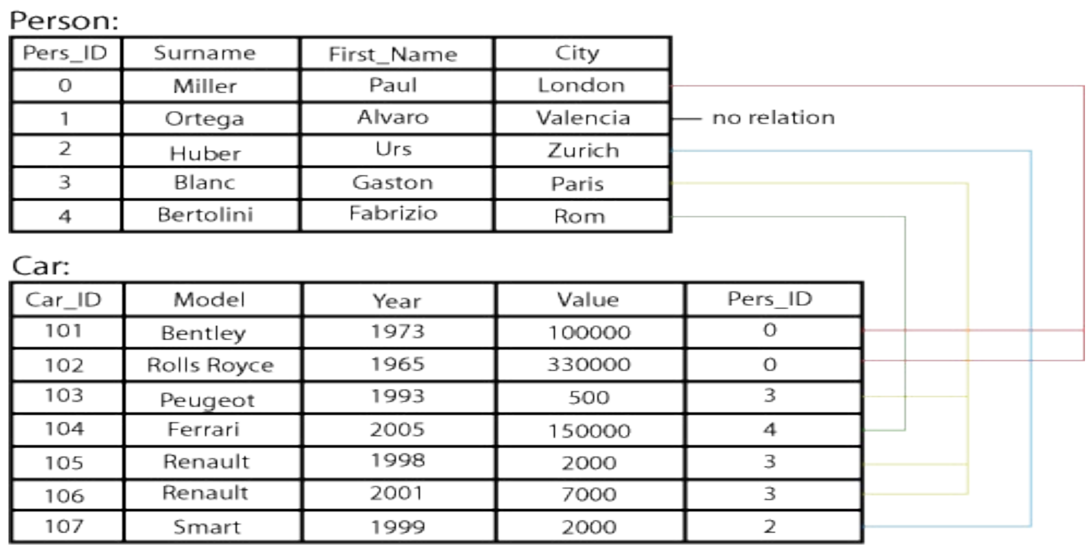
```
{"id": 10456,
 "fullName":"John Smith",
 "items": [
 {"product_id": 24, "quantity": 4},
 {"product_id": 45, "quantity": 1},
 {"product_id": 67, "quantity": 34}],
 "discount-code": "Yes"}
```

Document Databases

- A document is like an hash, with one id and many values
- Store Javascript Documents
    - JSON = JavaScript Object Notation
    - An associative array
    - Key value pairs
    - Values can be documents or arrays
    - Arrays can contain documents
- Data is implicitly denormalised - closer to a single table than lots of tables with relations connecting them
- Document databases allow indexing of documents on the basis of not only its primary identifier but also its properties

Documents are easier

Relational



Person:

Pers_ID	Surname	First_Name	City
0	Miller	Paul	London
1	Ortega	Alvaro	Valencia
2	Huber	Urs	Zurich
3	Blanc	Gaston	Paris
4	Bertolini	Fabrizio	Rom

no relation

Car:

Car_ID	Model	Year	Value	Pers_ID
101	Bentley	1973	100000	0
102	Rolls Royce	1965	330000	0
103	Peugeot	1993	500	3
104	Ferrari	2005	150000	4
105	Renault	1998	2000	3
106	Renault	2001	7000	3
107	Smart	1999	2000	2

Document DB

```
{
  first_name: 'Paul',
  surname: 'Miller'
  city: 'London',
  location: [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Document DB Features

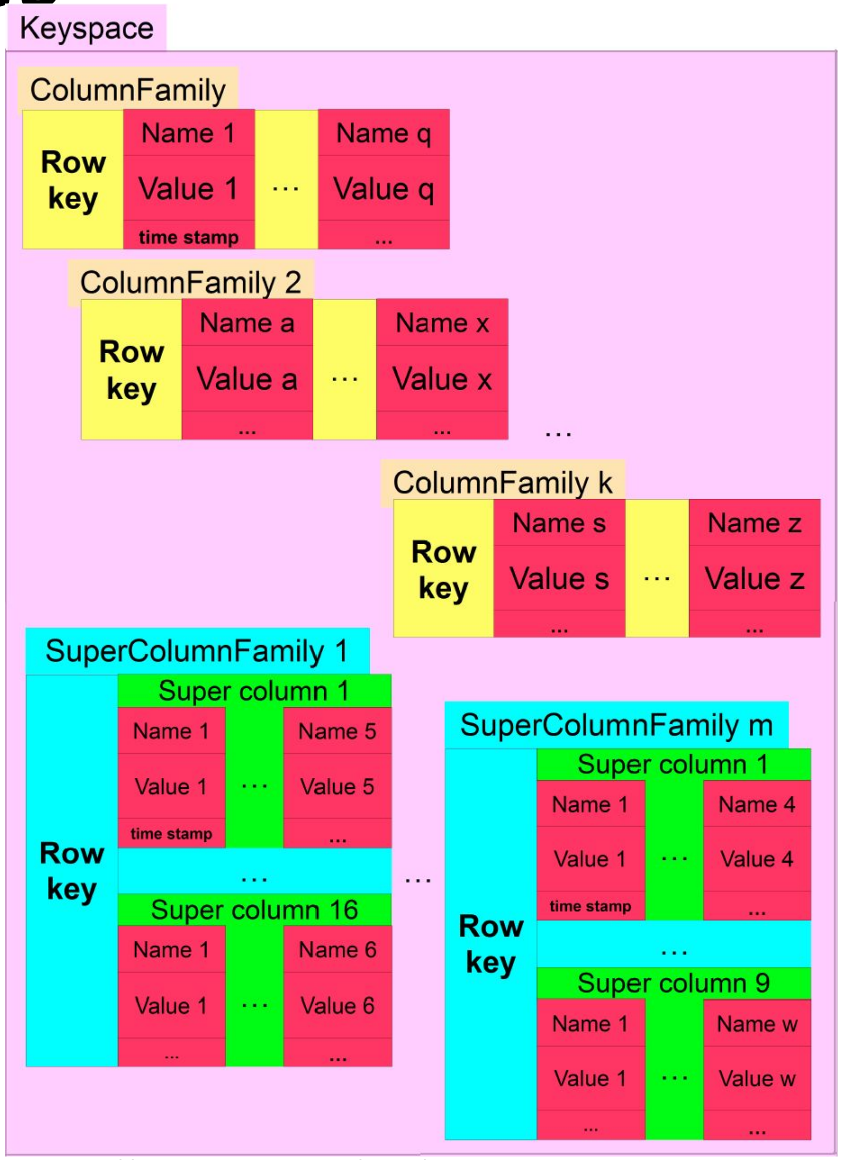
Rich Queries	• Find Paul's cars • Find everybody who owns a car built between 1970 and 1980
Geospatial	• Find all of the car owners in London
Text Search	• Find all the cars described as having leather seats
Aggregation	• What's the average value of Paul's car collection
Map Reduce	• For each make and model of car, how many exist?

MongoDB

```
{
  first_name: 'Paul',
  surname: 'Miller',
  city: 'London',
  location: [45.123,47.232],
  cars: [
    { model: 'Bentley',
      year: 1973,
      value: 100000, … },
    { model: 'Rolls Royce',
      year: 1965,
      value: 330000, … }
  ]
}
```

Column Stores

- the column can vary in width
- sparse wide column sets



https://en.wikipedia.org/wiki/Keyspace_(distributed_data_store)

Column Stores

- Store data as columns rather than rows
    - Columns organized in column family
    - Each column belongs to a single column family
    - Column acts as a unit for access
    - Particular column family will be accessed together

- Efficient to do column ordered operations
- Not so great at row based queries
- Adding columns is quite inexpensive and is done on a row-by-row basis
- Each row can have a different set of columns, or none at all, allowing tables to remain sparse without incurring a storage cost for null values

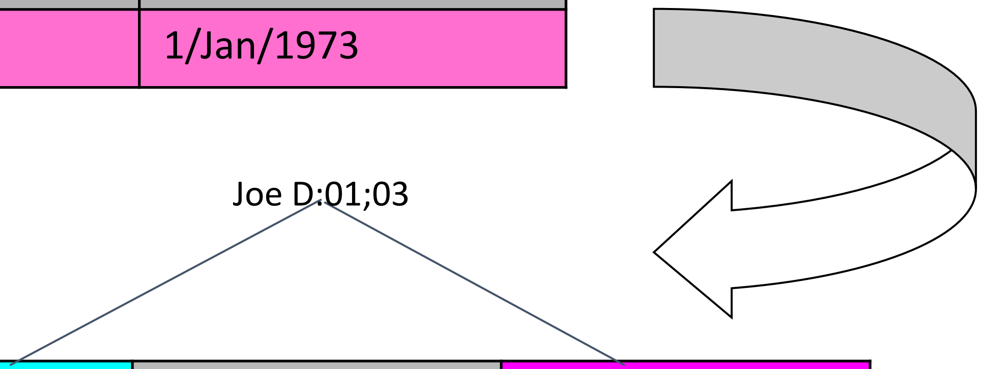
Relational/Row Order Databases

ID	Name	Salary	Start Date
1	Joe D	$24000	1/Jun/1970
2	Peter J	$28000	1/Feb/1972
3	Joe D	$23000	1/Jan/1973

Column Databases

- Materialise storage as columns, primary key is the data, which is mapped from rowids

ID	Name	Salary	Start Date
1	Joe D	$24000	1/Jun/1970
2	Peter J	$28000	1/Feb/1972
3	Joe D	$23000	1/Jan/1973



Joe D:01	Peter J:02	Joe D:03
24000:01	28000:02	23000:03
1/Jun/1970:01	1/Feb/1972:02	1/Jan/1973:03

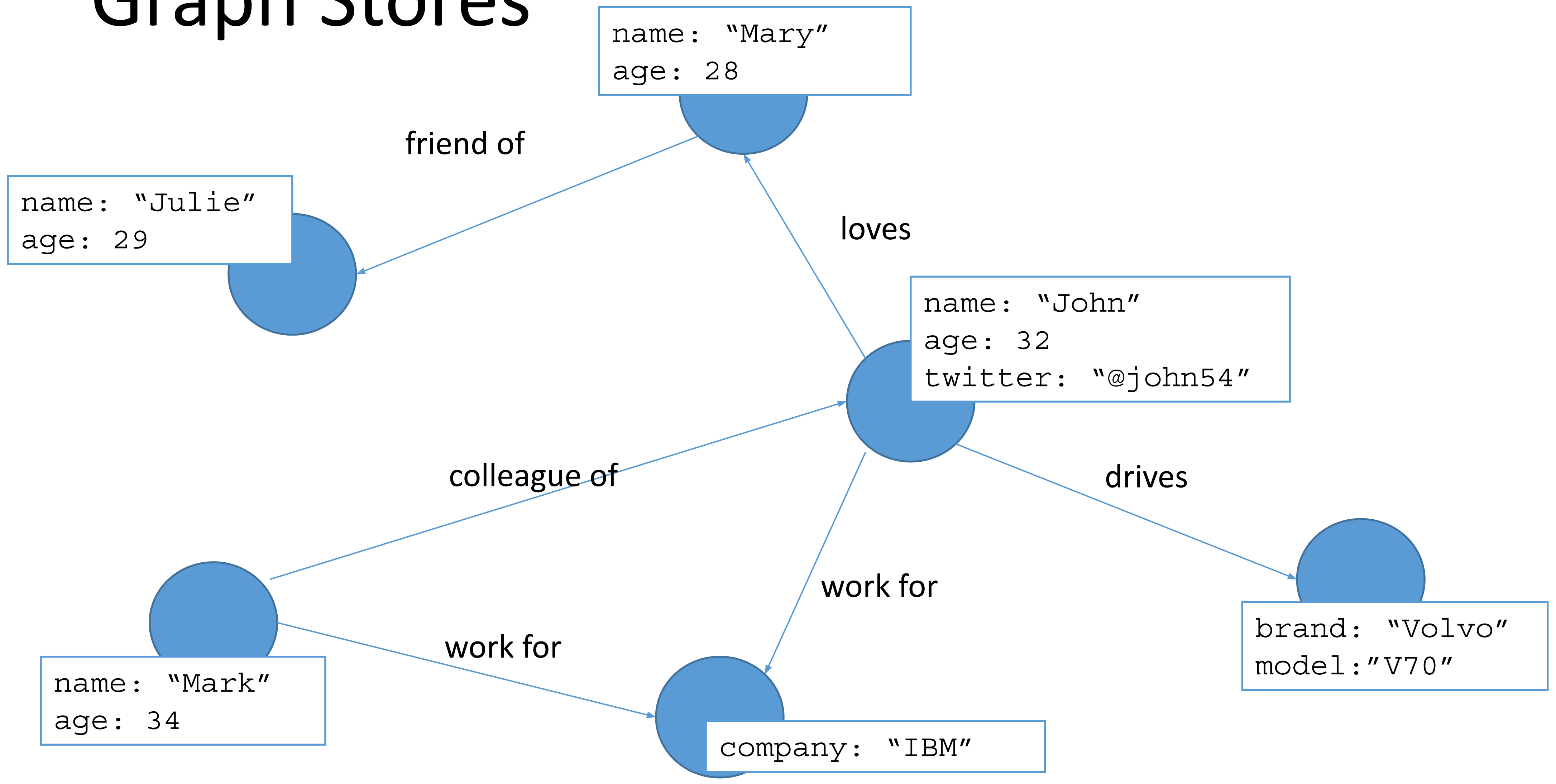
Joe D:01	Peter J:02	Joe D:03
24000:01	28000:02	23000:03
1/Jun/1970:01	1/Feb/1972:02	1/Jan/1973:03

Pros and Cons

- Relational: Good For
  - Queries that return small subsets of rows
  - Queries that use a large subset of row data
  - e.g. *find all employee data for employees with salary > 12000*

- Column: Good For
  - Queries that require just a column of data
  - Queries that require a small subset of row data
  - E.g. *Give me the total salary outlay for all staff*

Graph Stores



Graph Stores

- Data model composed by nodes connected by edges
  - Nodes represent entities
  - Edges represent the relationships between entities
  - Nodes and edges can have properties

- Querying a graph database means *traversing* the graph by following the relationships

- Pros:
  - Representing objects of the real world that are highly interconnected
  - Traversing the relationships in these data models is cheap

Graph Stores vs Relational Databases

- Relational databases are not ideally suited to representing relationships

- Relationship implemented through foreign keys
    - Expensive joins required to navigate relationships
    - Poor performance for highly connected data models

Hadoop

- Hadoop is a Map/Reduce Framework
- Used to partition computation on large datasets
- Used where you need to analyse very large volumes of the data
- E.g.
  - Count all the links on all the web pages in Ireland
  - Calculate the overnight interest on every account
  - Analyse the recommendations based on yesterday’s purchases

Example of A Mature NoSQL Model

Front End

Low Latency
High Performance

Middle Tier

General Purpose NoSQL
Database

Back End

Hadoop
Analytics

Low latency - optimized to process a very high volume of data with minimal delay

NoSQL vs. Relational Databases

Pros

- Flexible schema
- Simple API
- Scalable
- Distributed and Replicated storage
- Cheap

Cons

- Not ACID compliant
- No standards
- Eventual consistency
- Some products are at early stage: poor documentation/support

Conclusions

- Great technical transition of our generation
- Everyone will have a NoSQL deployment
- Right now it sits alongside Relational
- It’s Open Source

CSU44D01

Introduction to NoSQL

Yvette Graham
ygraham@tcd.ie

Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin