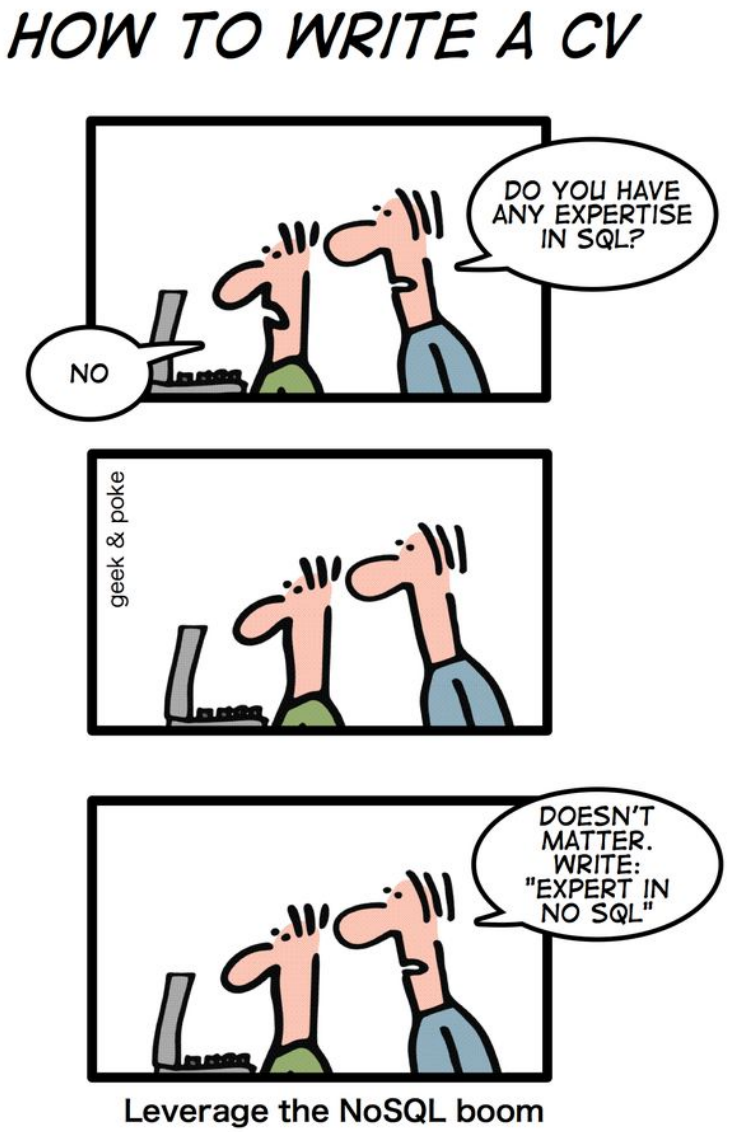
Introduction to NoSQL

Dr Annalina Caputo
Research Fellow @ ADAPT centre
annalina.caputo@adaptcentre.ie

Why?



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

Remember

Who Are These Guys?

The relational model : 1970

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. Act
at terminals and most application programs s
unaffected when the internal representation of da
and even when some aspects of the external
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.

Volume 13 / Number 6 / June, 1970

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.

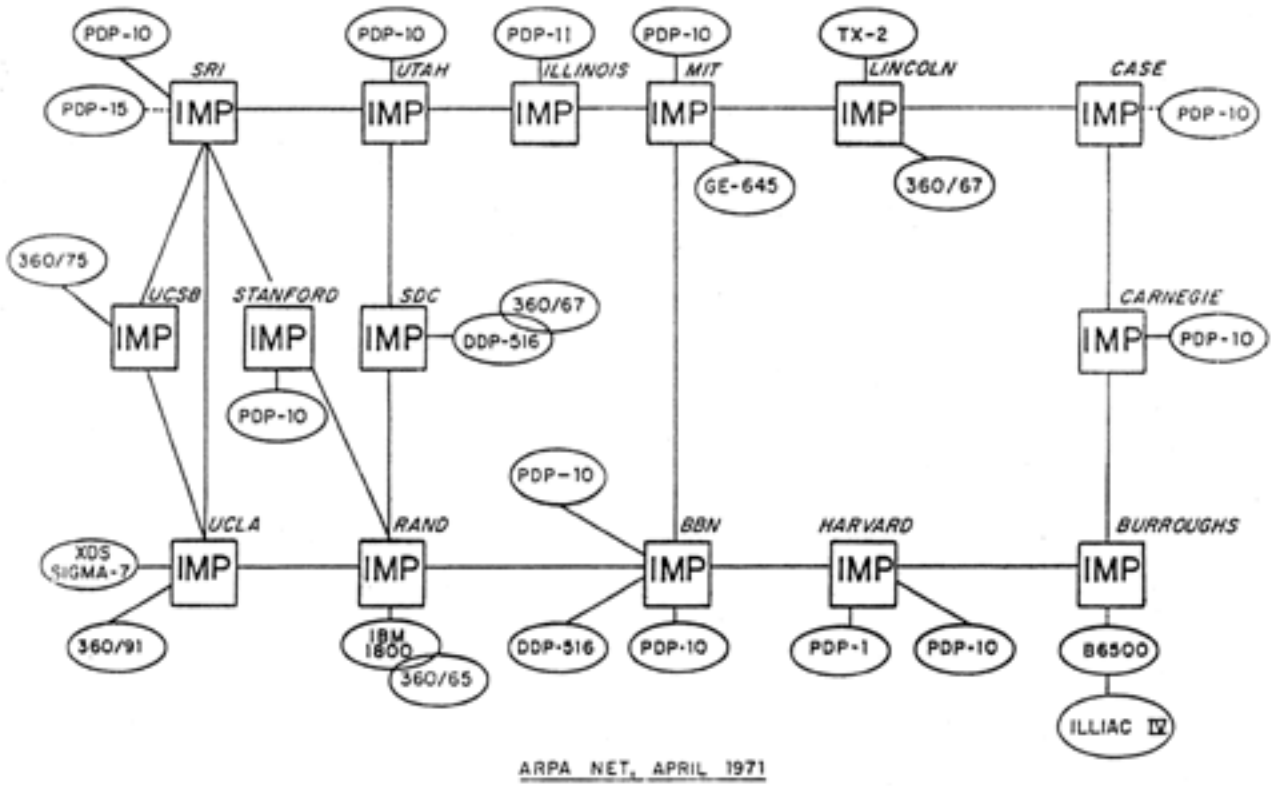
A further advantage of the relational view is that it forms a sound basis for treating derivability, redundancy,
ussed in Section
, has spawned a
ich is mistaking
the derivation of connections for the derivation of relations (see remarks in Section 2 on the "connection trap").

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

and terminal activities from growth in data types and changes in data representation—and certain kinds of *data inconsistency* which are expected to become troublesome even in nondeductive systems.

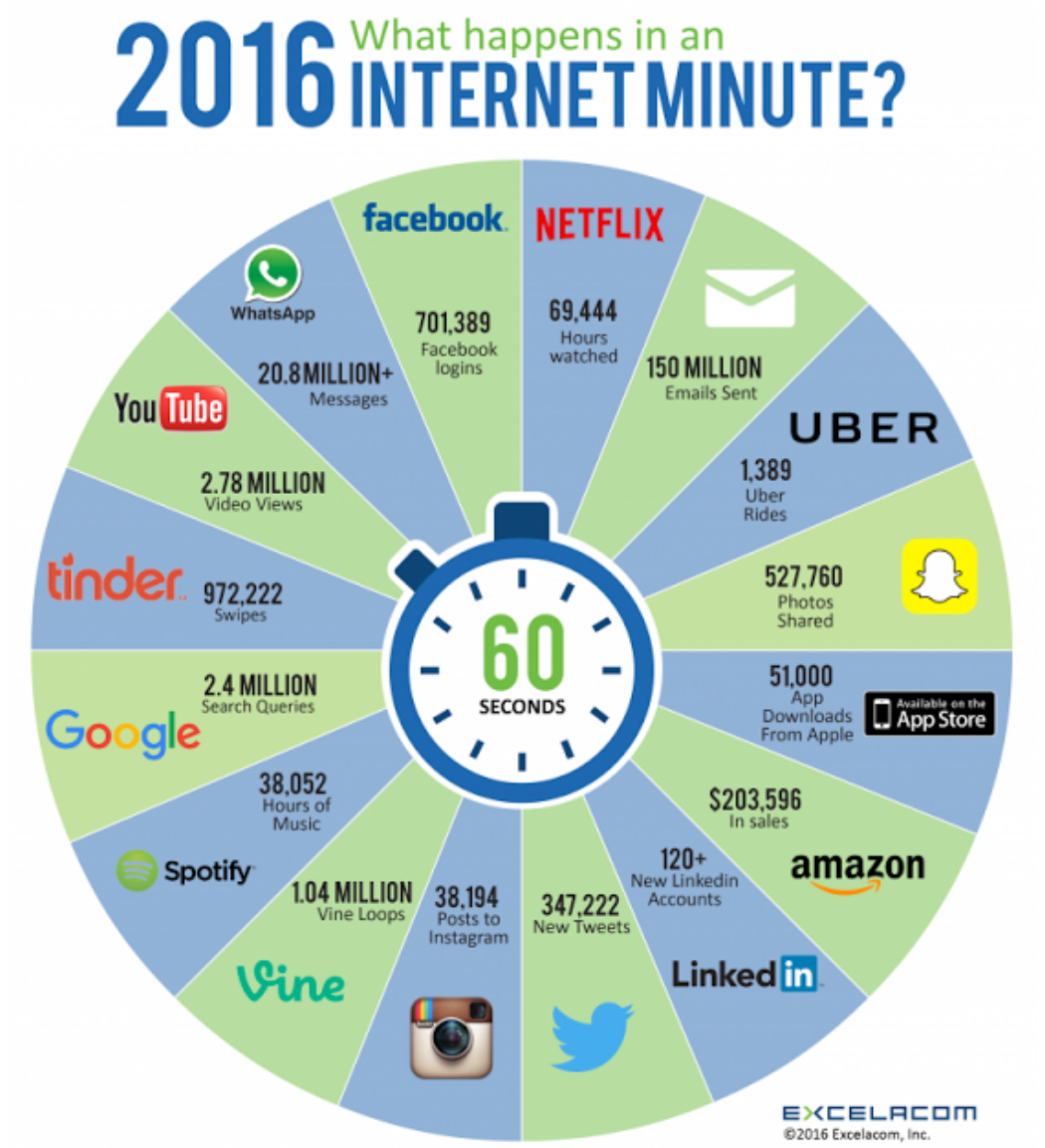
parts might be stored in ascending order by part serial number. Such systems normally permit application programs to assume that the order of presentation of records from such a file is identical to (or is a subordering of) the

The Internet - 1971



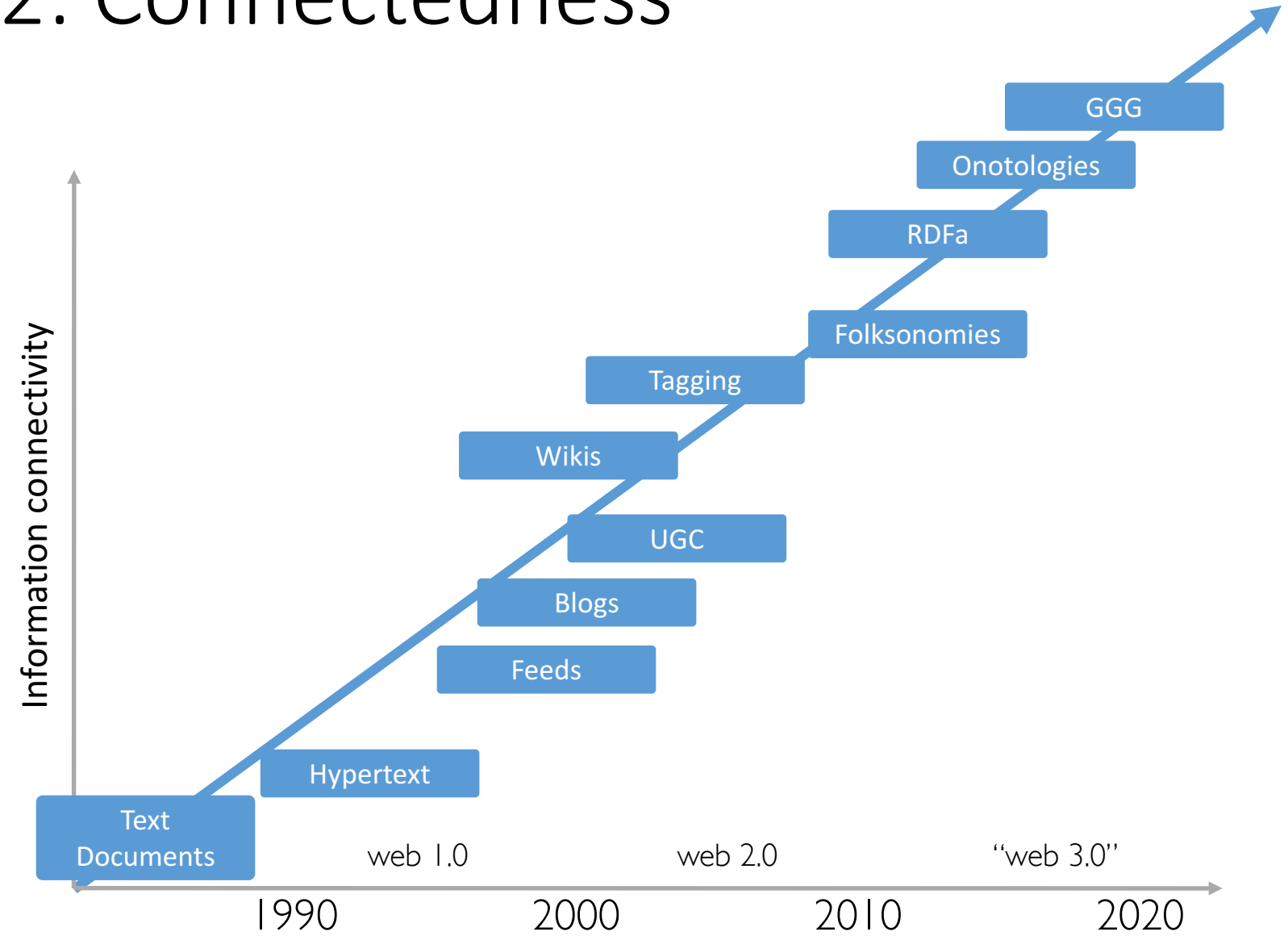
The Internet 2014

Trend 1: Data Size



http://twulibraries.blogspot.ie/2016/06/every-60-seconds-this-happens-online.html

Trend 2: Connectedness

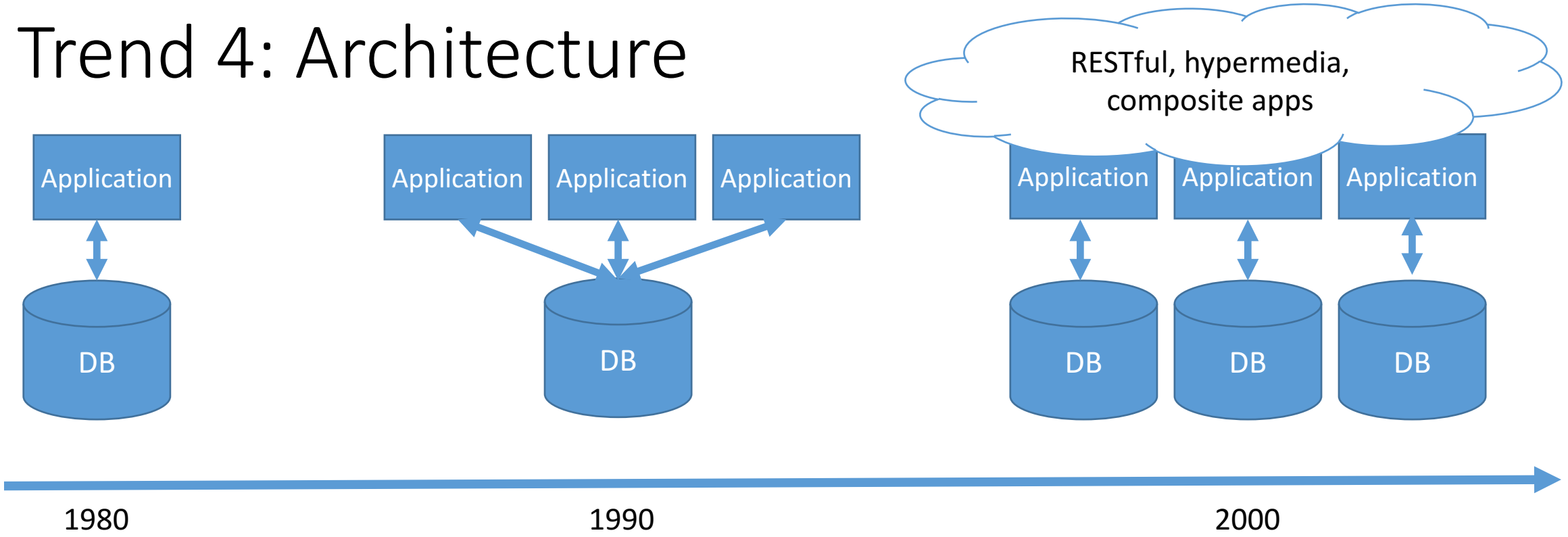


Trend 3: Semi-Structure

- 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



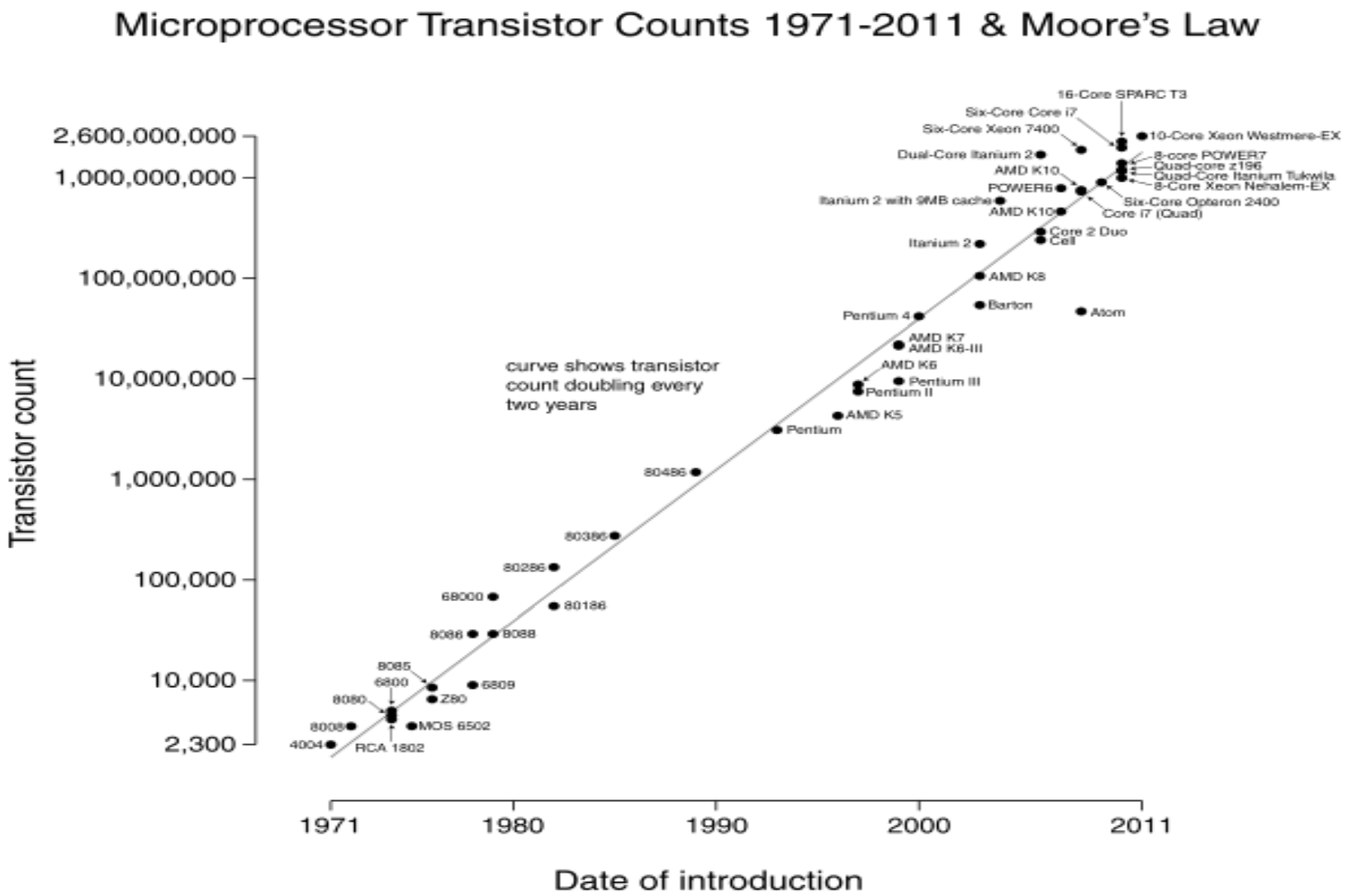
Mainframe application

Database as Integration hub

Decoupled services

Multicore
Parallelization/distributed
Cloud
Schema less

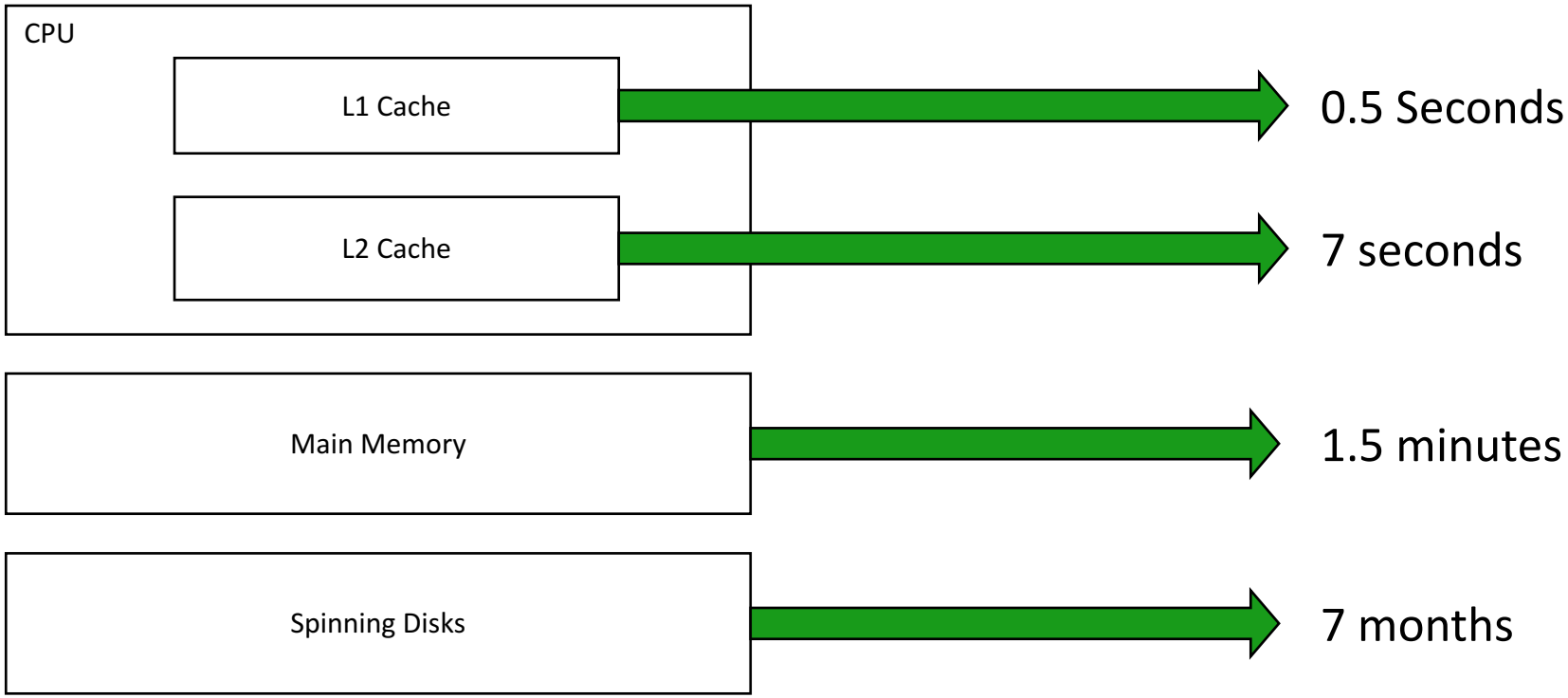
Moore’s Law


Microprocessor Transistor Counts 1971-2011 & Moore's Law
Transistor count
2,600,000,000
1,000,000,000
100,000,000
10,000,000
1,000,000
100,000
10,000
2,300
curve shows transistor count doubling every two years
16-Core SPARC T3
Six-Core Core i7
Six-Core Xeon 7400
10-Core Xeon Westmere-EX
Dual-Core Itanium 2
8-core POWER7
Quad-core z196
Quad-Core Itanium Tukwila
AMD K10
8-Core Xeon Nehalem-EX
POWER6
Itanium 2 with 9MB cache
Six-Core Opteron 2400
AMD K10
Core i7 (Quad)
Core 2 Duo
Itanium 2
Cell
AMD K8
Barton
Atom
Pentium 4
AMD K7
AMD K8-III
AMD K6
Pentium III
Pentium II
AMD K5
Pentium
80486
80386
80286
68000
80186
8086
8088
8085
6800
6809
8080
Z80
8008
MOS 6502
4004
RCA 1802
1971
1980
1990
2000
2011
Date of introduction

Limits to Moore's Law

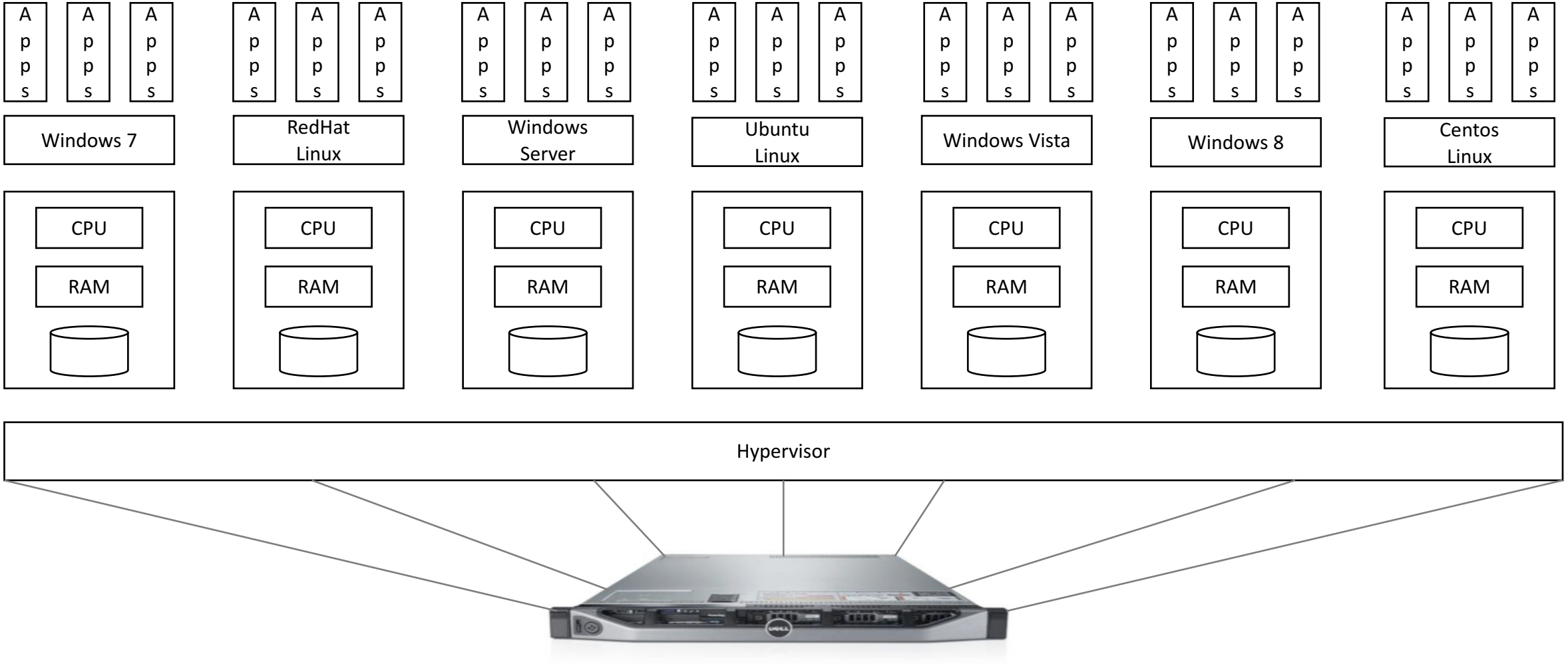
- Great for chip/transistor based technologies
- But has had little impact on Disks
- Disks are still dog slow
  … Like really, really, really slow
- Slower than molasses on a cold day in Alaska?
- Even slower than that
- How slow….

How Slow is a Disk*

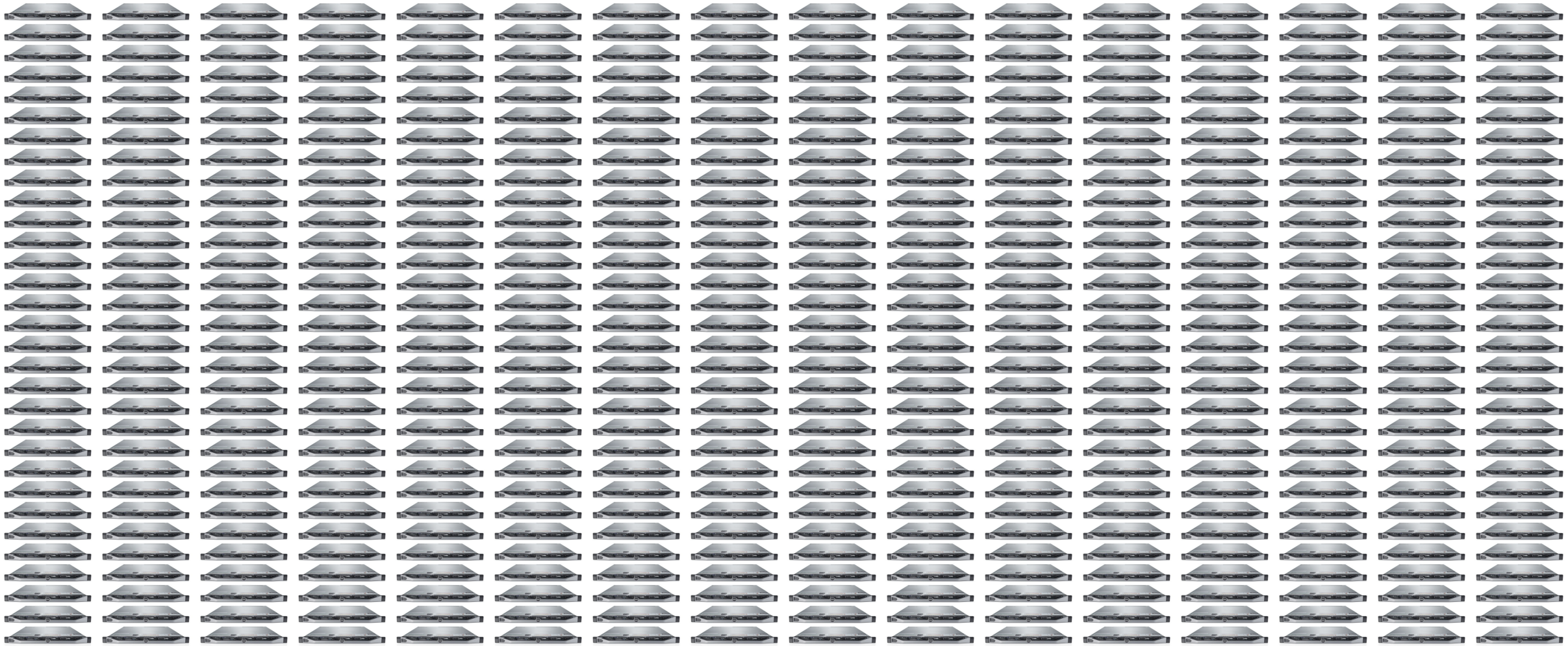


* http://norvig.com/21-days.html#answers

Virtualization



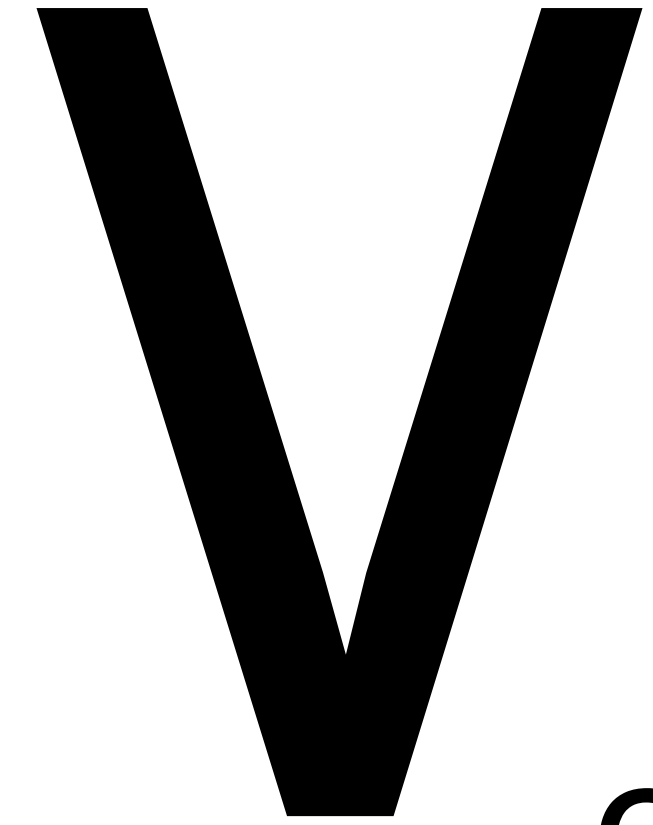
Now Imagine Lots



Cloud Computing

- Takes virtualization to the extreme
- Amazon has over 500,000 physical servers
- Microsoft has over 1 million physical servers
- Google has over 2.3 million physical servers
- Anyone with a credit card can start hundreds of servers in a matter of minutes

Why NoSQL



elocity

ariety

olume

Unlock Your Big Data

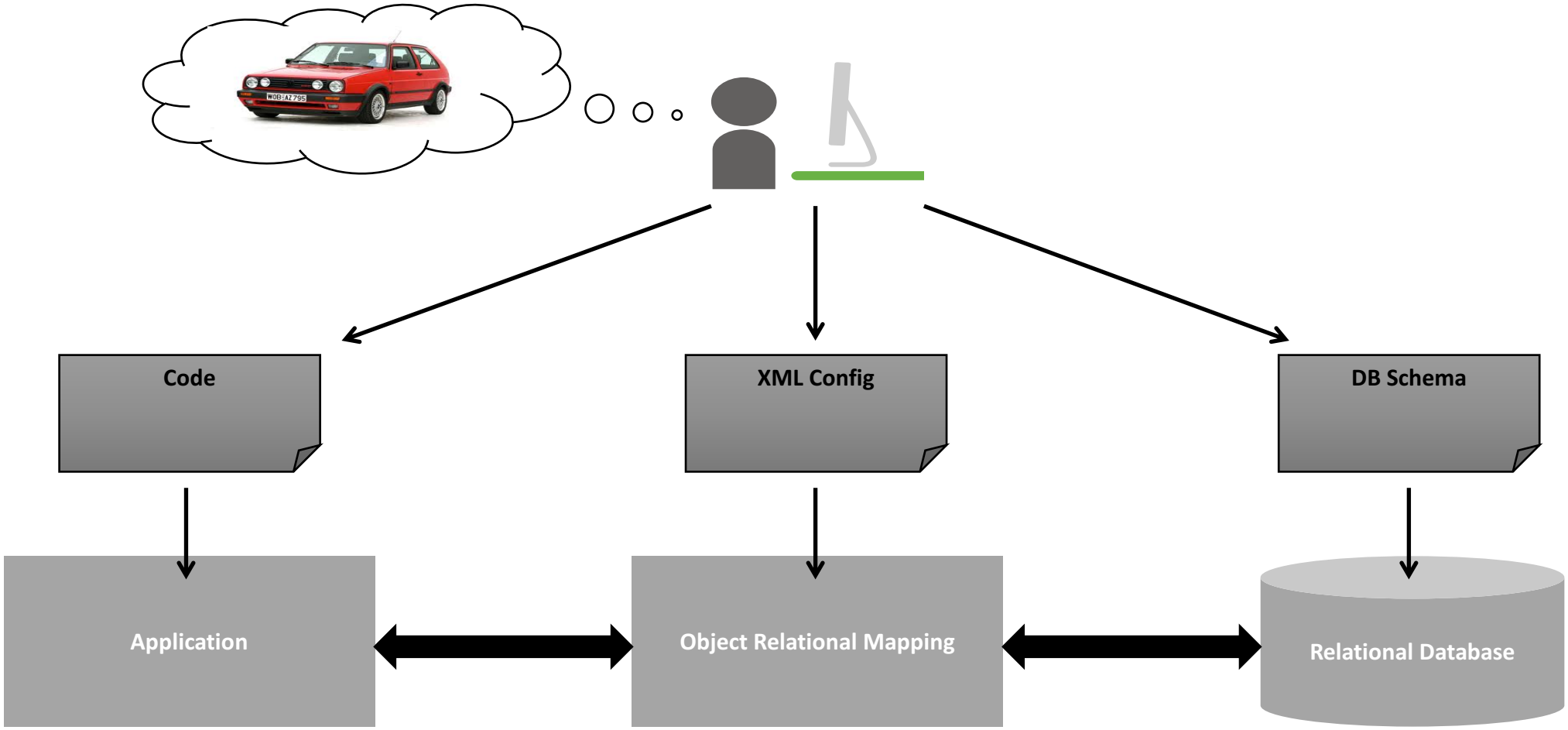
Traditional Data Approaches

- **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 monolithic memory
  - Assume single disk farm
  - Hard to partition
  - Based on 1970’s storage assumptions

It makes development hard



Limitations of Relational Databases

- Impedance Mismatch
- Application and integration
- Scale up vs. scale out

The AWS Disruption - 2006

- Cost per GB/Month for Stable Storage
  - ~$5GB down to .15 cent per GB
- Unlimited Storage
- Purchased in GB chunks
- Pay only for what you use

What did this mean?

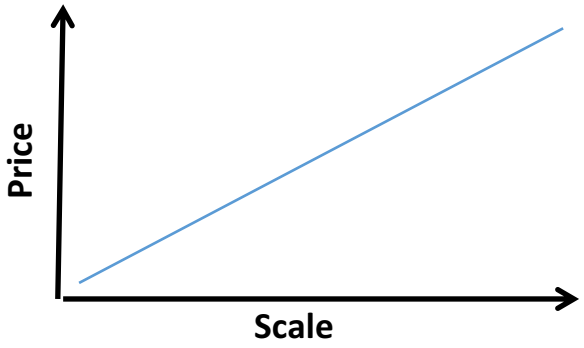
- No CAPEX
- No Data Centre
- Availability of Google Scale
- Utility Pricing

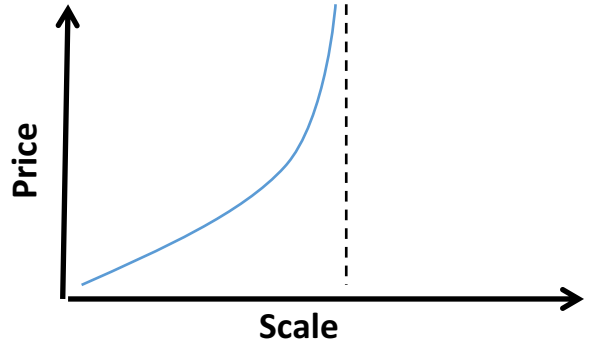
Filter-Store-Distribute

Store-Filter-Distribute

NoSQL Scales Better

Vs.





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

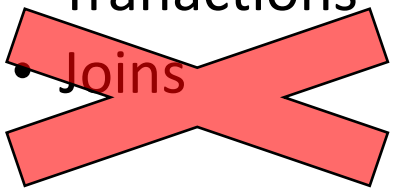
What?

NoSQL – No Definition

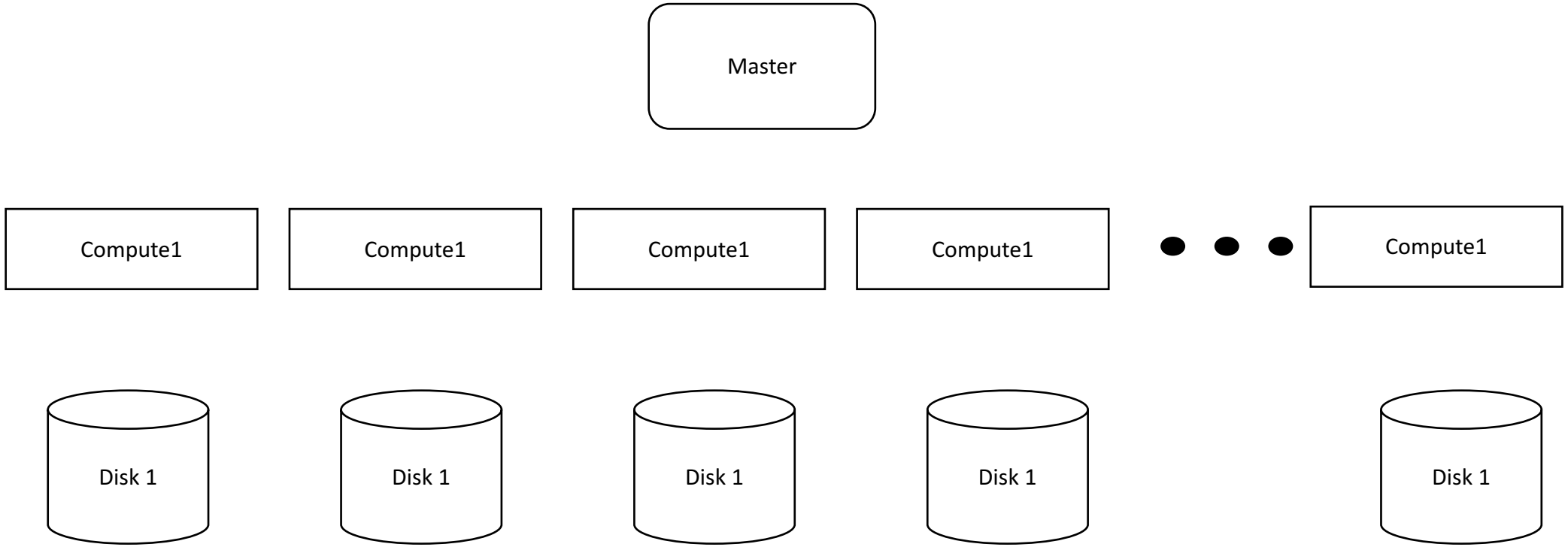
- Non Relational
- No SQL as query language
- Schema less
- Usually –not always – Open-Source project
- They are distributed: they are usually driven by the requirements of running on clusters (with the exception of graph DBs)
- RDBMS use ACID transactions, NoSQL don’t

What is NoSQL?

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



Distributed Computing



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 unavailablity

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

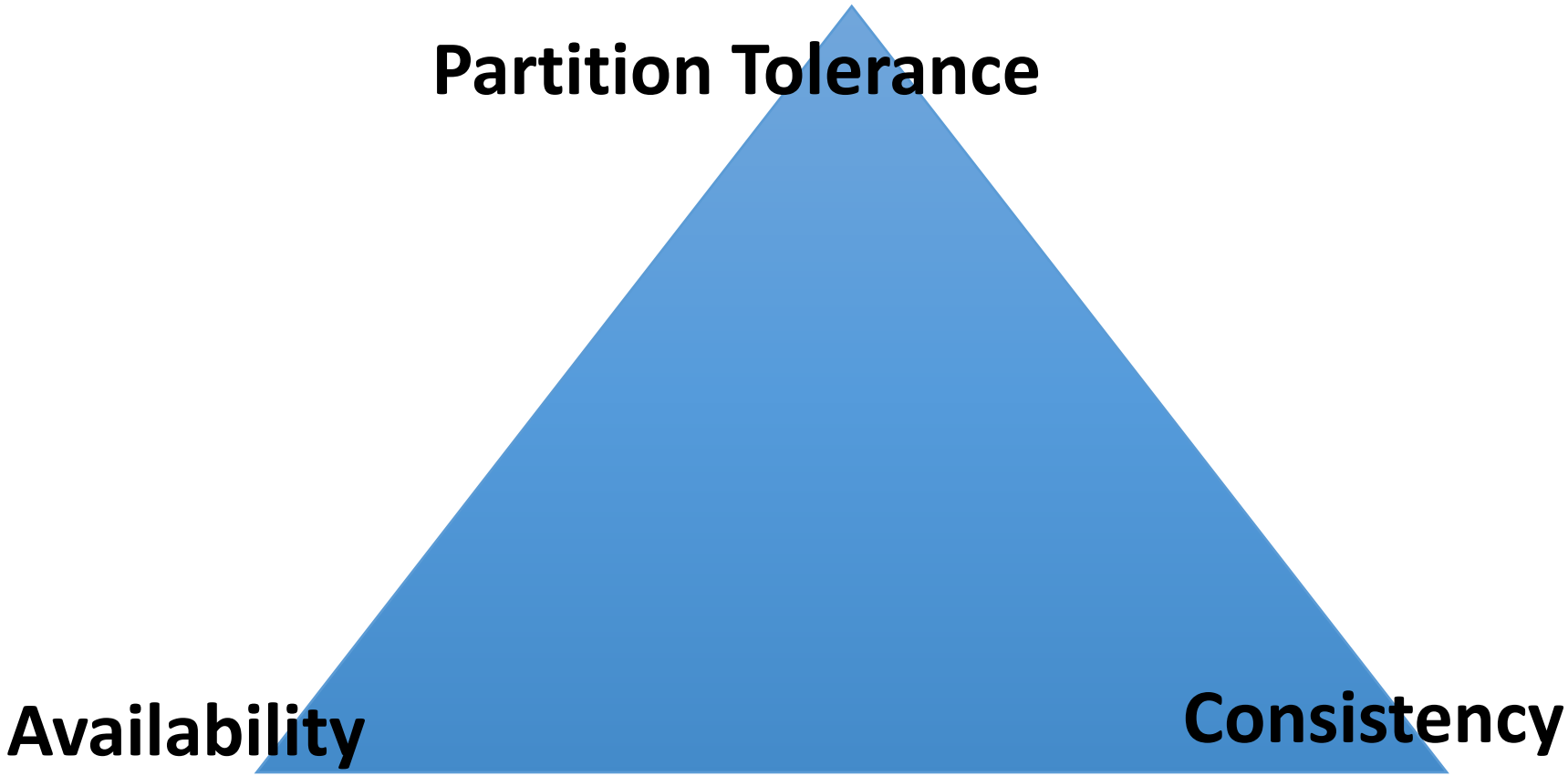
- The same data is replicated and copied over multiple nodes
- Peer-to-peer: all replicas have equal weight and can accept writing
  - Scaling by adding nodes
  - Node failure without losing write capability
  - Inconsistency problem (write)

Combined Approaches

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

CAP Theorem

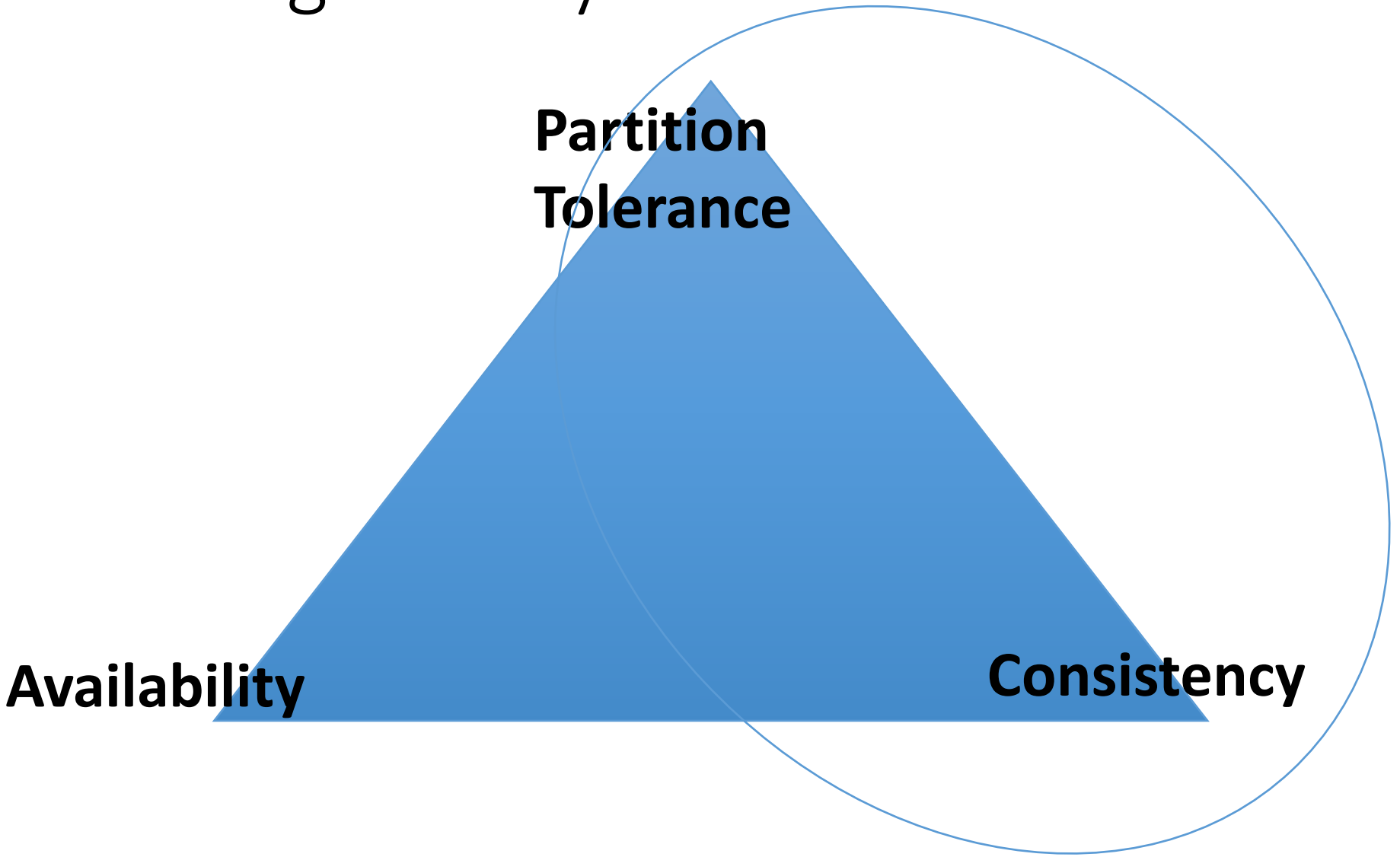
CAP Theorem



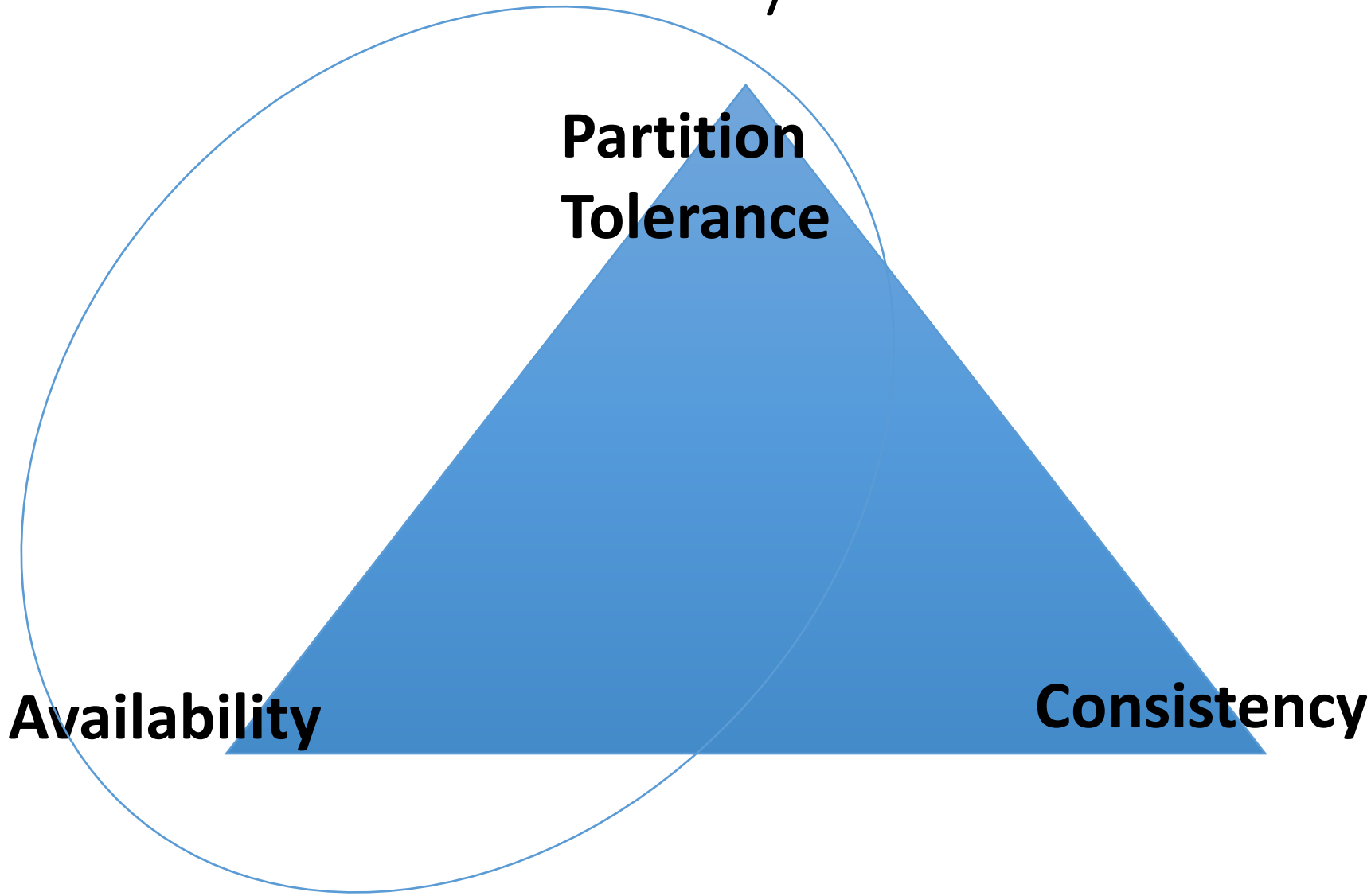
CAP Theorem

- **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 – MongoDB Style



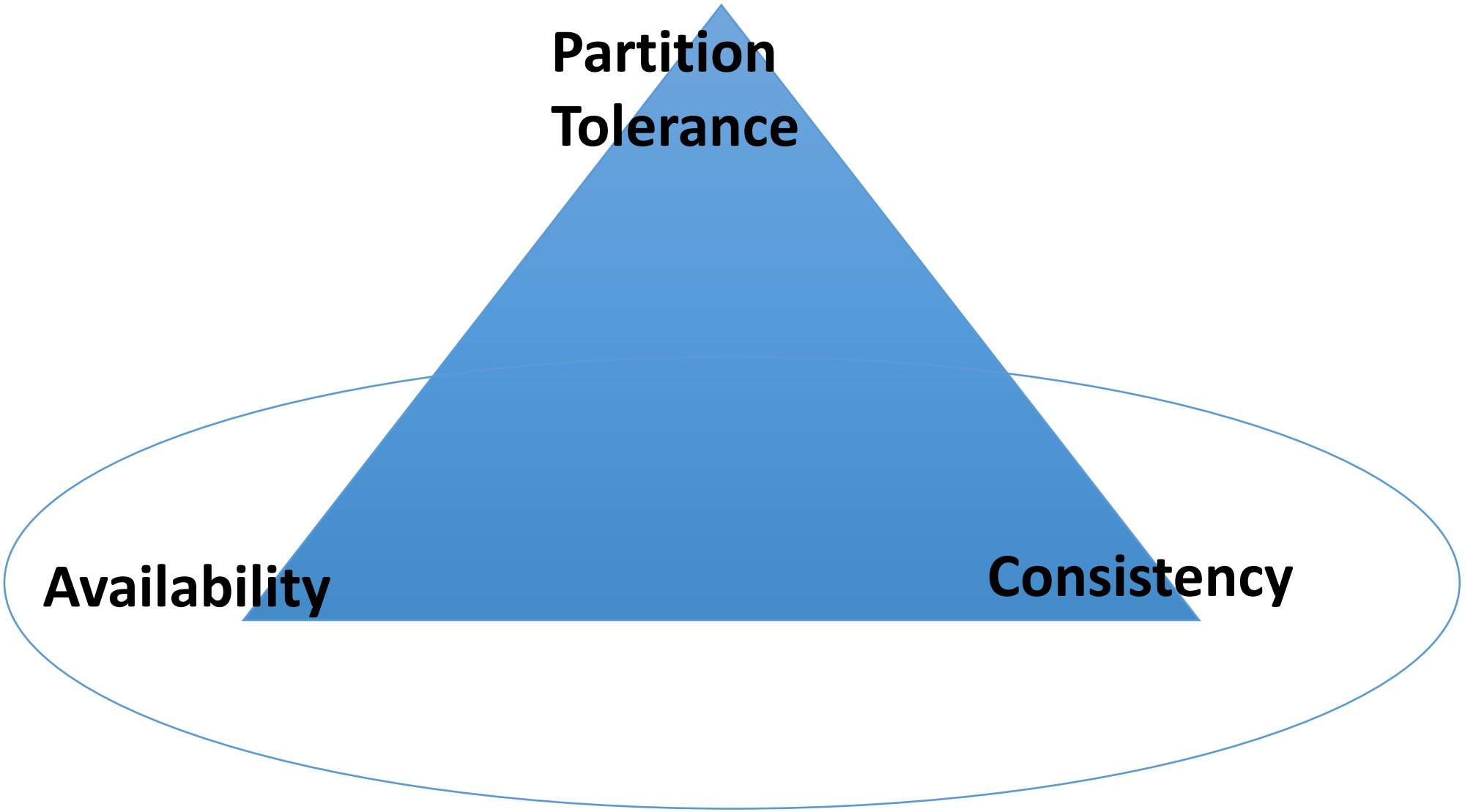
CAP – I Want Availability As Well



Be Careful What You Wish For

- Even Google thinks Eventual Consistency Sucks
- Reconciliation is a problem

But What About CA?



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.

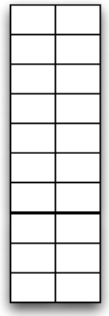
BASE

- **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.

Data Model

Key-Value

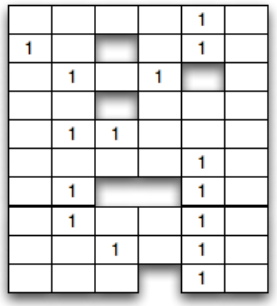


Voldemort (LinkedIn), Amazon SimpleDB, Memcache,
BerkeleyDB, Oracle NoSQL

Graph DB

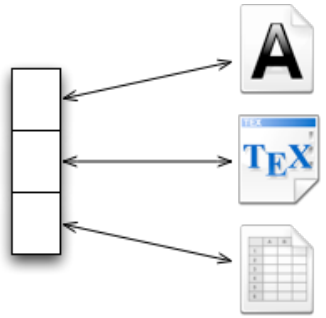
Neo4j, InfiniteGraph, OrientDB, Titan GraphDB

BigTable



Cassandra, Hbase, Amazon Redshift,
HP Vertica, Teradata

Document



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

Data Model

- A data model is a representation 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

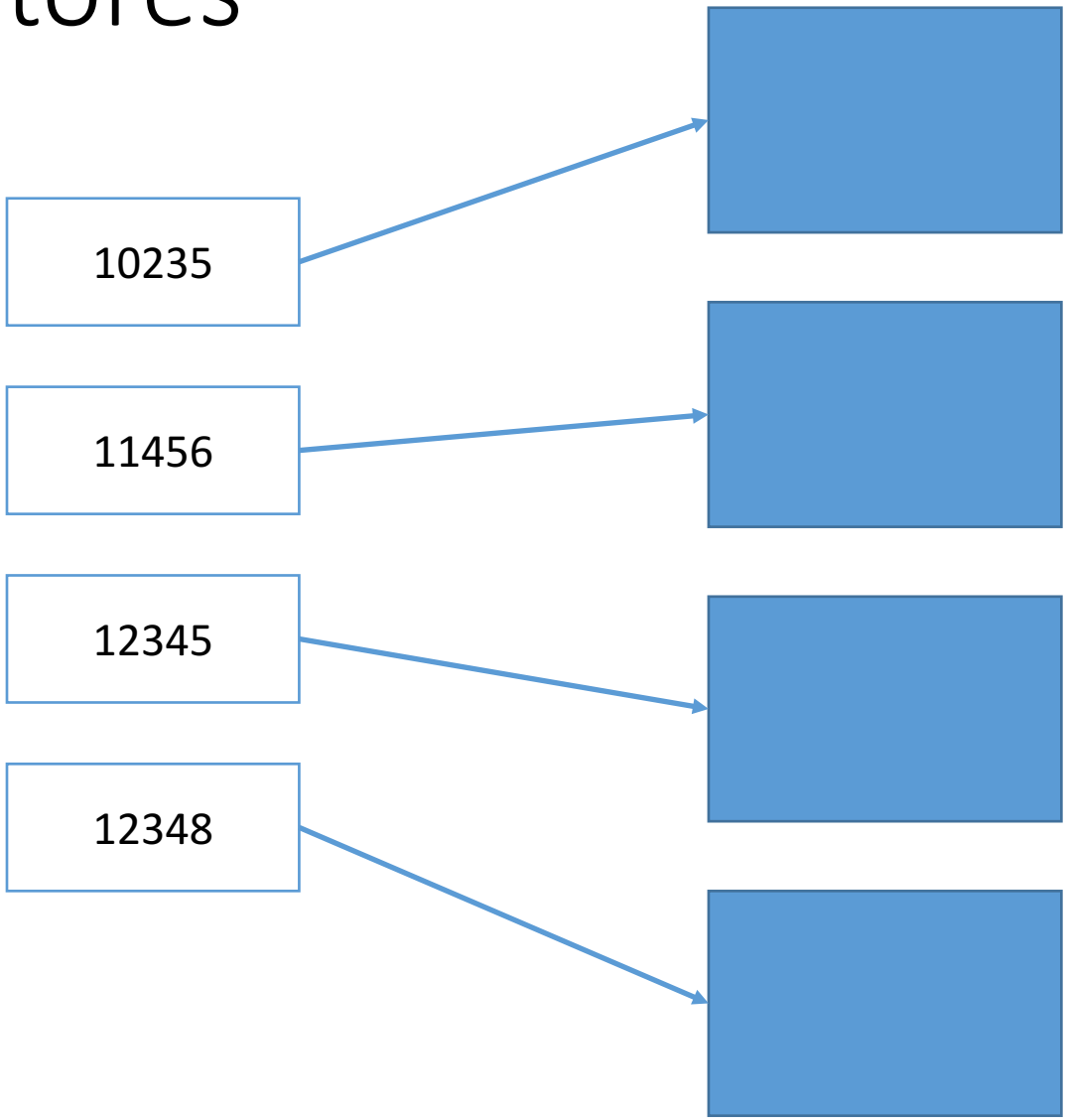
- Key Value Stores
- Document Stores
- Column Stores

Aggregate[1]

- Graph Stores
- Note
  - Lots of Hybrids
  - Lots of NewSQL vendors
  - Some niche Graph Stores

[1]Pramod J. Sadalage and Martin Fowler. NoSQL Distilled, A Brief Guide to the Emerging World of Polyglot Persistence. Addison-Wesley.

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

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, … }
  ]
}
```

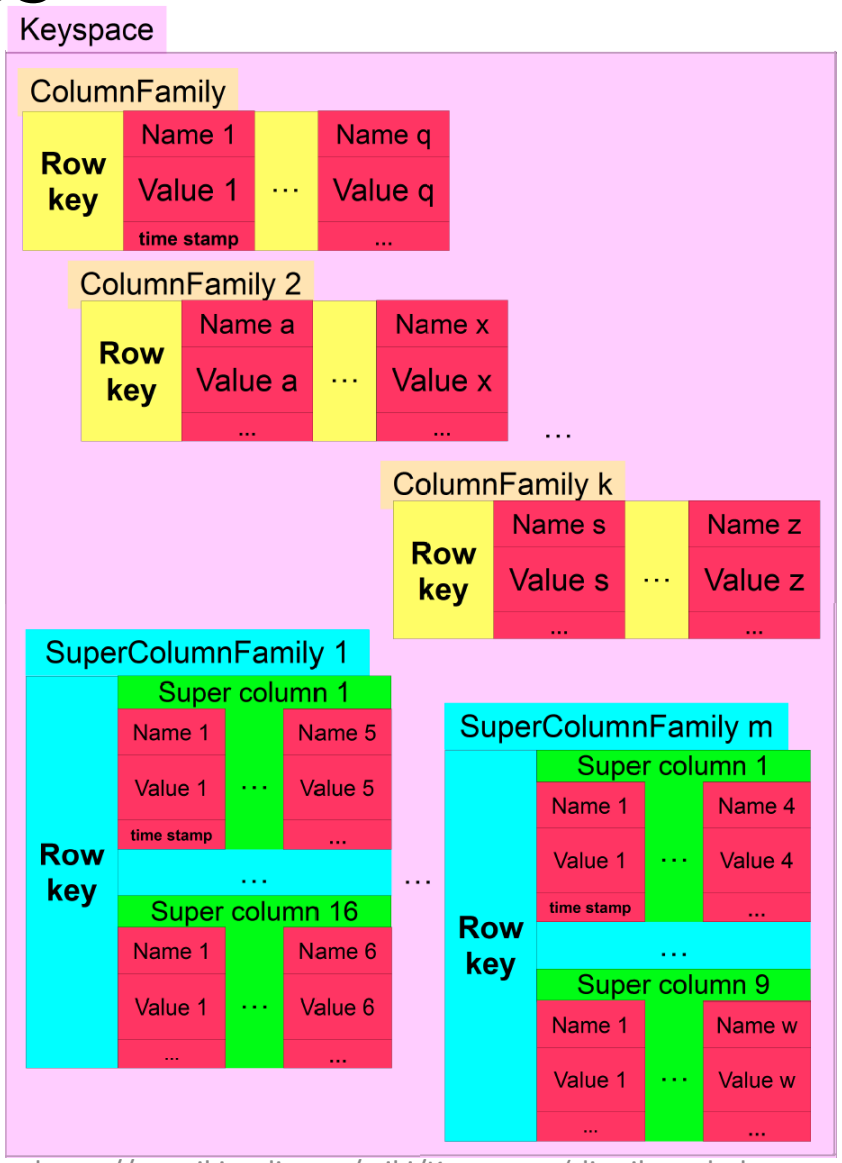
Document DBs are full featured

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 in the world?

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



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

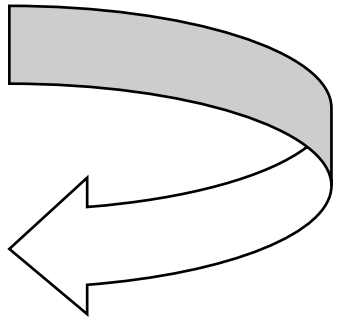
Column Stores

- Store data as columns rather 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

- Materialise storage as rows

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

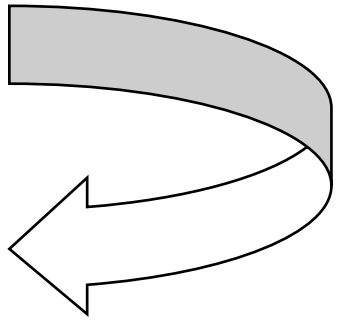


1	Joe D	2400	1/Jun/1970	1	Joe D	2400	1/Jun/1970	1	Joe D	2400	1/Jun/1970

Column Databases

- Materialise data as columns

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



1	Joe D	24000	1/Jun/1970
2	Peter J	28000	1/Feb/1972
3	Phil G	23000	1/Jan/1973

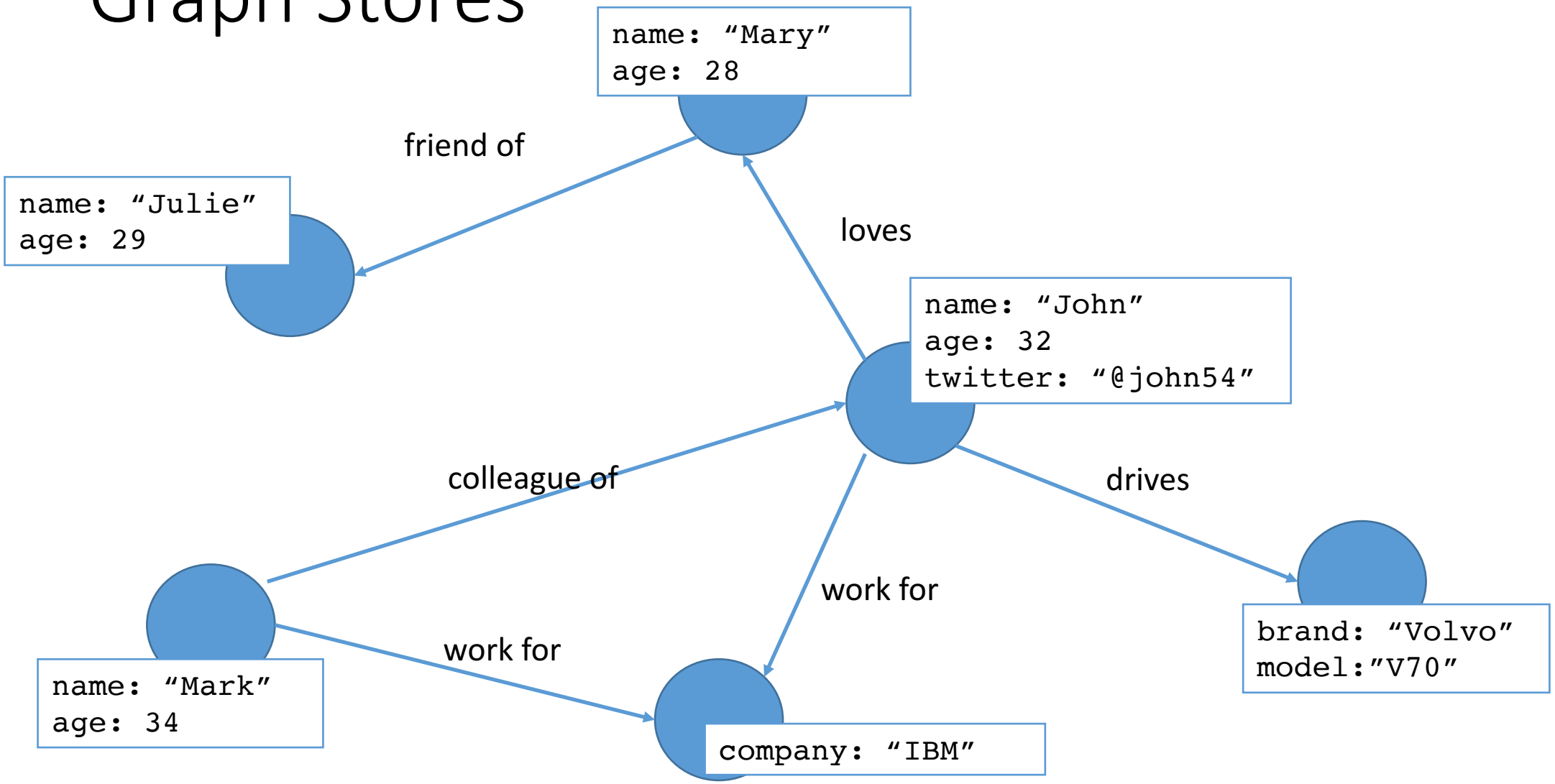
Pros and Cons

- 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*

Where Does Hadoop Fit?

- Hadoop is a Map/Reduce Framework
- Used to partition computation on large datasets
- Used where you need to analyse most 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 yesterdays purchases

Graph Stores



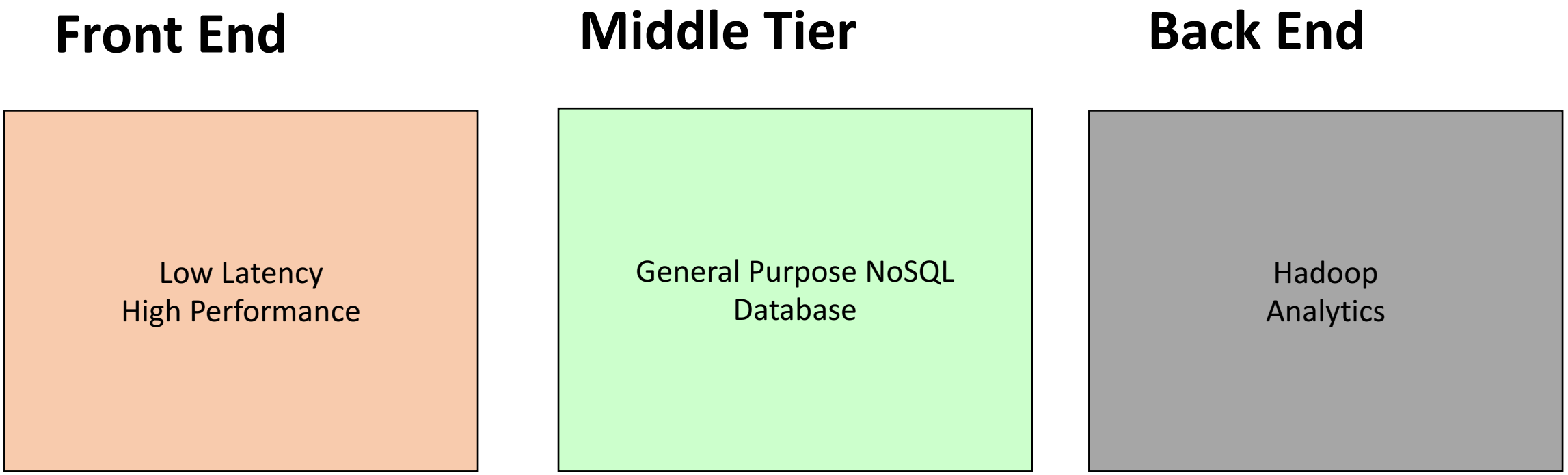
Graph Stores

- 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 inter-connected
  - 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.

A Mature NoSQL Model



NoSQL vs. Relational Databases

Pros

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

Cons

- Not ACID compliant
- No standards
- Eventually 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
- In the future it will replace Relational
- It’s all Open Source
- Ask me about it after ☺