# NoSQL databases

Francieli ZANON BOITO

# Goal of this class

- To understand the motivations behind NoSQL ("Not only SQL") systems
- An overview of different solutions
- NOT a manual to learn specific NoSQL databases
  - Too many of them
  - For a comprehensive list: http://nosql-database.org/
  - Next class and the lab activity: Neo4j

# "Traditional" applications

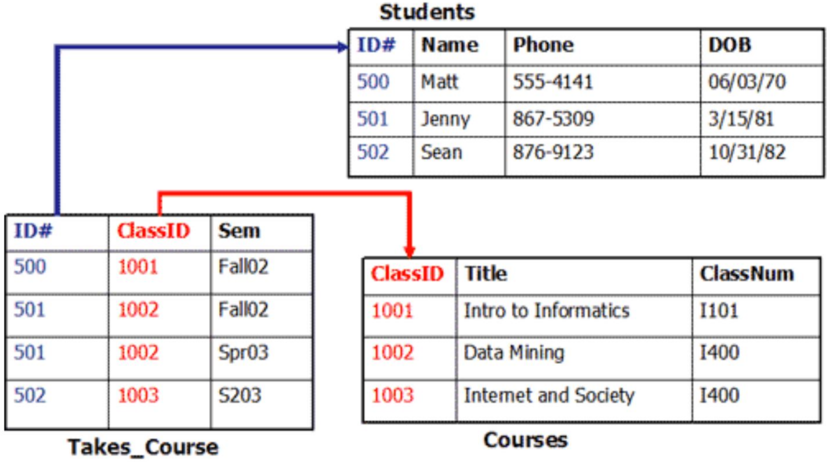
- Months of planning and development
    - Including the schema for the relational database (MySQL, Oracle, PostgreSQL, …)
- Structured data
- Its scale is known in advance
- Configuration for the servers is chosen accordingly
- Scale-up

Source: slides by Vincent Leroy

# Relational databases

- Data organized as tables
  - Row = record, Column = attribute
- Relations between tables
  - Integrity constraints

Select title from courses natural join takes_courses group by ClassID having count(*) > 10



Students

| ID# | Name | Phone | DOB |
|---|---|---|---|
| 500 | Matt | 555-4141 | 06/03/70 |
| 501 | Jenny | 867-5309 | 3/15/81 |
| 502 | Sean | 876-9123 | 10/31/82 |

Takes_Course

| ID# | ClassID | Sem |
|---|---|---|
| 500 | 1001 | Fall02 |
| 501 | 1002 | Fall02 |
| 501 | 1002 | Spr03 |
| 502 | 1003 | S203 |

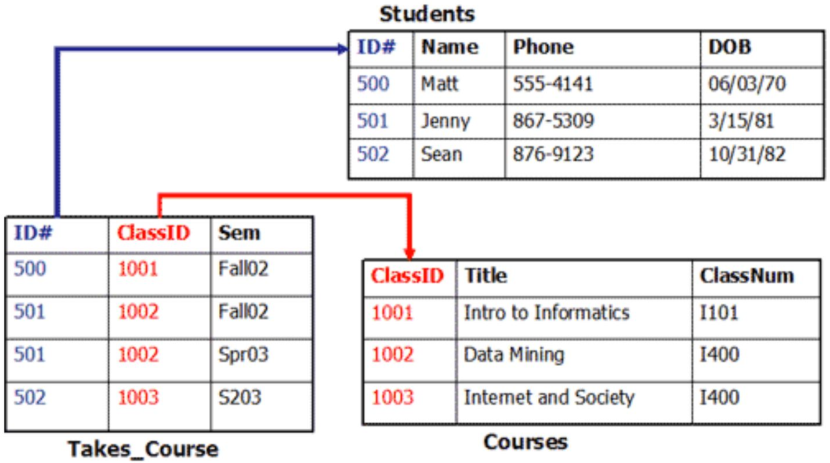
Courses

| ClassID | Title | ClassNum |
|---|---|---|
| 1001 | Intro to Informatics | I101 |
| 1002 | Data Mining | I400 |
| 1003 | Internet and Society | I400 |

# The big data era

- Agile development
  - Frequent release of new features, possibly changing the data model
- Data structure can be unknown or variable
- Large amounts of data, thousands to millions of users
- Need to scale-out
- Cloud-based

```
Select title from courses natural join takes_courses group by ClassID having count(*) > 10
```



Students

| ID# | Name | Phone | DOB |
|---|---|---|---|
| 500 | Matt | 555-4141 | 06/03/70 |
| 501 | Jenny | 867-5309 | 3/15/81 |
| 502 | Sean | 876-9123 | 10/31/82 |

| ID# | ClassID | Sem |
|---|---|---|
| 500 | 1001 | Fall02 |
| 501 | 1002 | Fall02 |
| 501 | 1002 | Spr03 |
| 502 | 1003 | S203 |

Takes_Course

| ClassID | Title | ClassNum |
|---|---|---|
| 1001 | Intro to Informatics | I101 |
| 1002 | Data Mining | I400 |
| 1003 | Internet and Society | I400 |

Courses

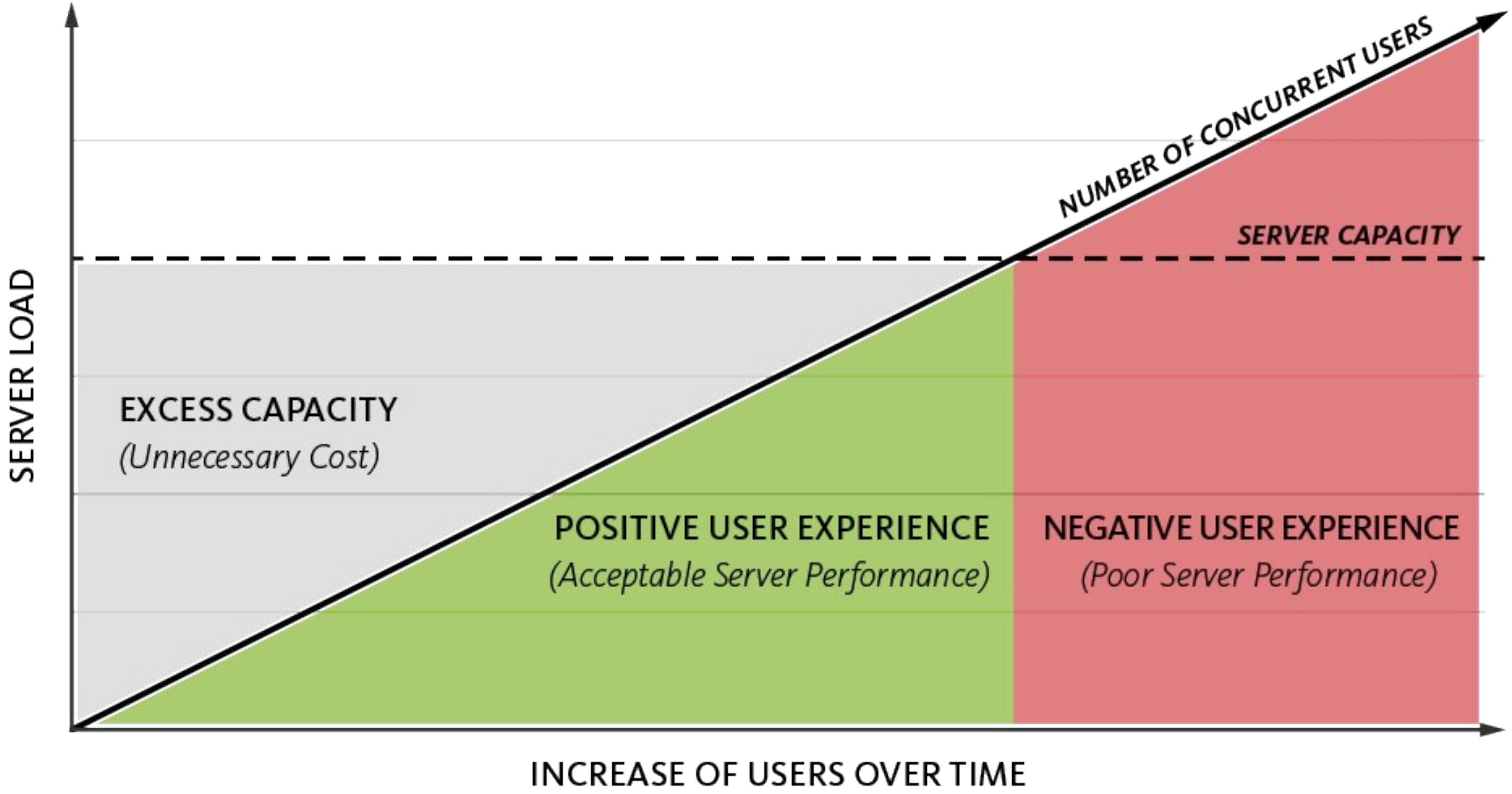

SERVER LOAD
NUMBER OF CONCURRENT USERS
SERVER CAPACITY
EXCESS CAPACITY
(Unnecessary Cost)
POSITIVE USER EXPERIENCE
(Acceptable Server Performance)
NEGATIVE USER EXPERIENCE
(Poor Server Performance)
INCREASE OF USERS OVER TIME

Figure from https://www.couchbase.com/resources/why-nosql

| SQL relational databases | NoSQL databases |
|---|---|
| Data is organized in tables | Data is organized in key-value pairs, sparse columns, documents, or graphs |
| Pre-defined schema | Less rigid formats, documents can have different fields, add as you go |
| ACID | |
| | |
| | |
| | |

Source: slides by Vincent Leroy

# ACID properties

- Atomicity
  - Transaction are all or nothing (e.g. when adding a bi-directional friendship relation, it’s added both ways or not at all)
- Consistency
  - Only valid data written (e.g. cannot say a student takes a course that is not in the courses table)
- Isolation
  - When multiple transactions execute simultaneously, they appear as if they were executed sequentially (aka serializability)
- Durability
  - When data has been written and validated, it is permanent (i.e. no data loss, even in the case of some failures)

→ Easy life for the developer

| SQL relational databases | NoSQL databases |
|---|---|
| Data is organized in tables | Data is organized in key-value pairs, sparse columns, documents, or graphs |
| Pre-defined schema | Less rigid formats, documents can have different fields, add as you go |
| ACID | Looser consistency models |
| | |
| | |
| | |

# CAP theorem (Brewer's theorem)

- **Consistency:** every node returns the same, most recent, successful write (sequential consistency)
- **Availability:** every non-failed node answer all requests it receives
- **Partition tolerance:** the system continues to work when network fails

- In a centralized system, no need for P, we have CA
- In a distributed data store, P is essential
  - When the network fails, we need to choose between C and A

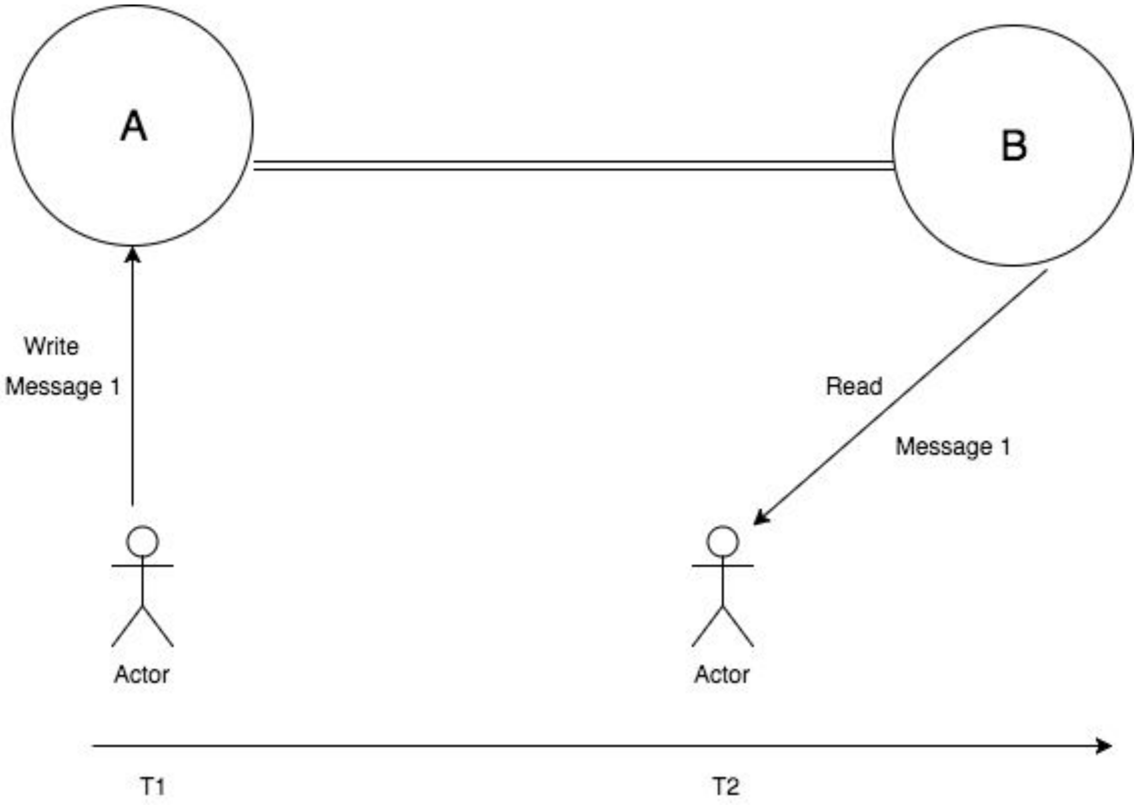


Figure from https://shekhargulati.com/2018/08/08/week-2-cap-theorem-for-application-developers/

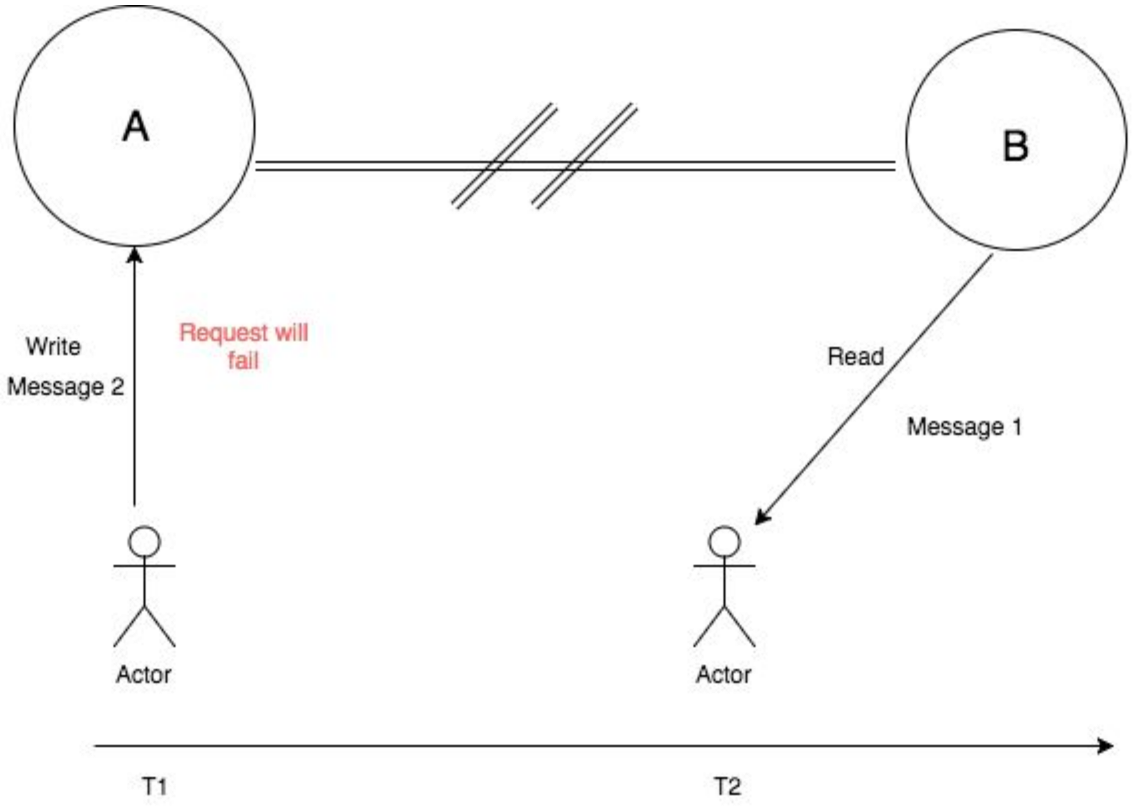


Figure from https://shekhargulati.com/2018/08/08/week-2-cap-theorem-for-application-developers/

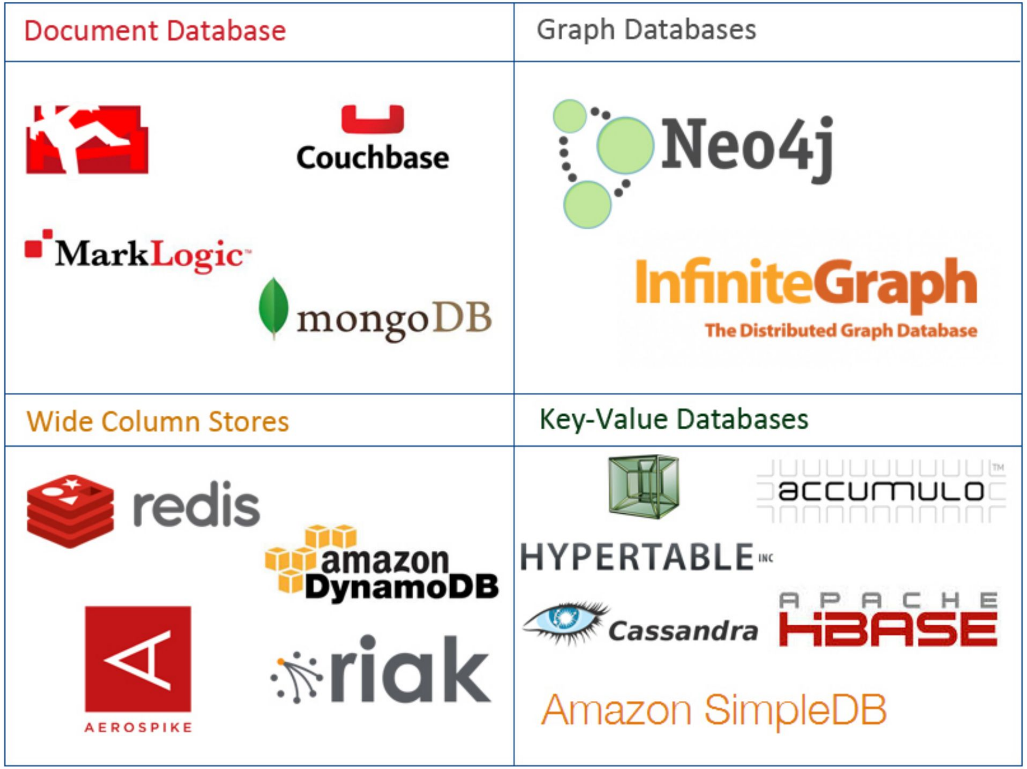
# Weak consistency

- Eventual consistency
  - It will be consistent after some time, when there is no network partition
  - Sometimes we could be writing data that is going to be read only later
- Different levels of consistency
  - Causal consistency
  - Read-your-writes consistency
  - Etc
- What to choose? It depends on the application!
- Some databases are not updated very often

| SQL relational databases | NoSQL databases |
|---|---|
| Data is organized in tables | Data is organized in key-value pairs, sparse columns, documents, or graphs |
| Pre-defined schema | Less rigid formats, documents can have different fields, add as you go |
| ACID | Looser consistency models |
| 40-year-old standard (from the 70s) | First papers in 2006 and 2007 |
| SQL query language | Diverse query APIs, it can be difficult to migrate between solutions |
| Query to access small subsets of the data | We often want to process ALL data |

# SQL or NoSQL?

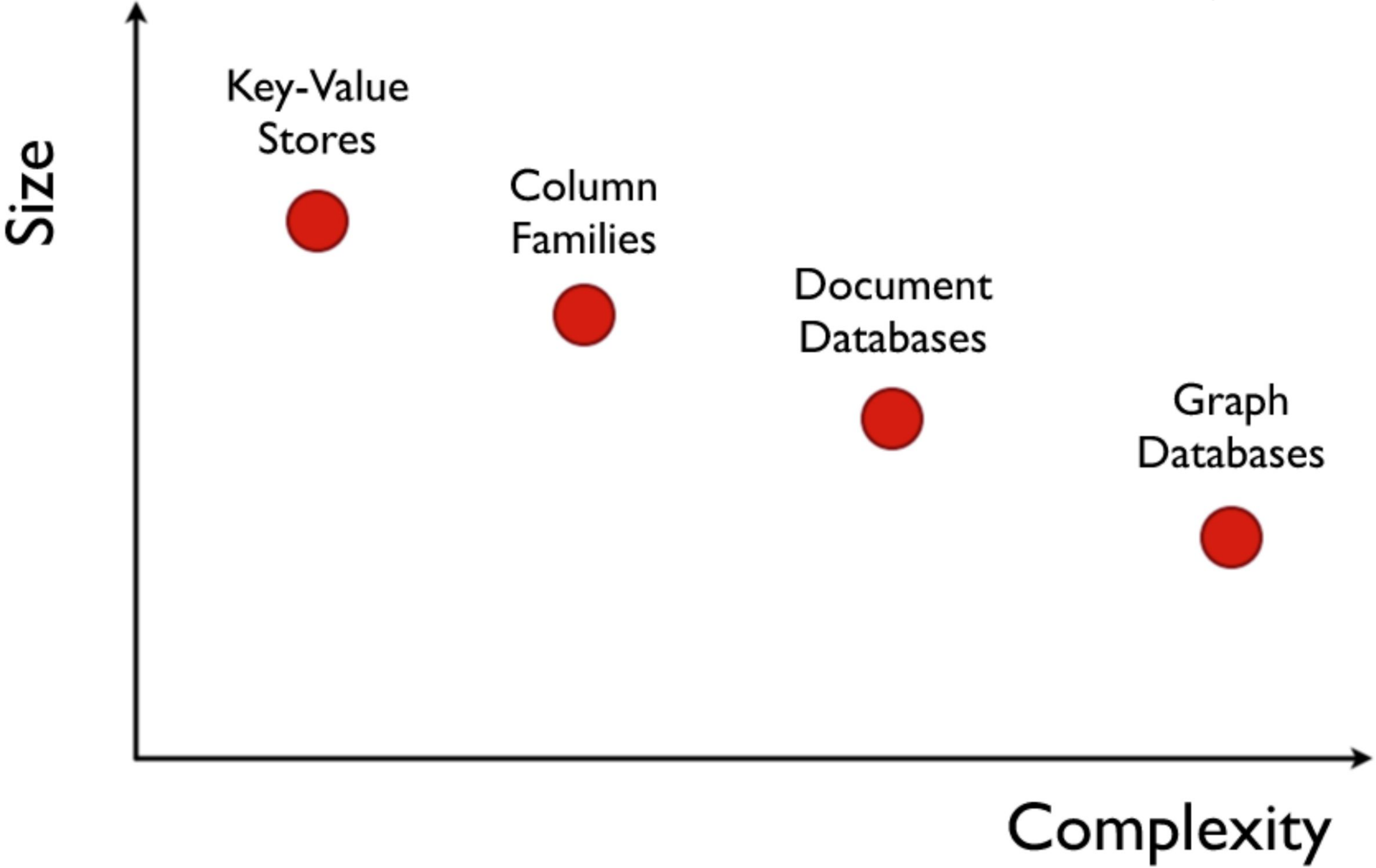
- It depends on the application!
- Snapshot stories use Amazon DynamoDB *
- Facebook and Netflix use/used Apache Cassandra
- Ryanair uses Couchbase for their mobile app (over 3 million users) **



@cloudtxt http://www.aryannava.com

* https://www.youtube.com/watch?v=WUleQzu9I_8
** https://www.couchbase.com/customers/ryanair

Source: slides by Lorenzo Alberton



# Key-value store

- Data in < key, value > pairs
- Two basic operations (similar to data structures like hashMap and dictionaries)
    - Put(K,V)
    - Get(K)
- Can be used to cache information in memory
- Recent research: accelerate it with hardware

redis

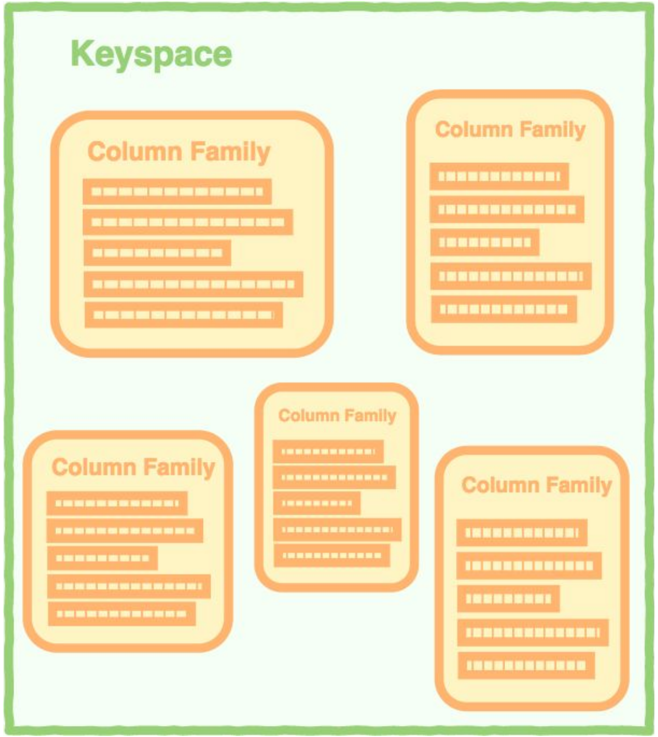
amazon
DynamoDB

# Wide Column/Tabular DB

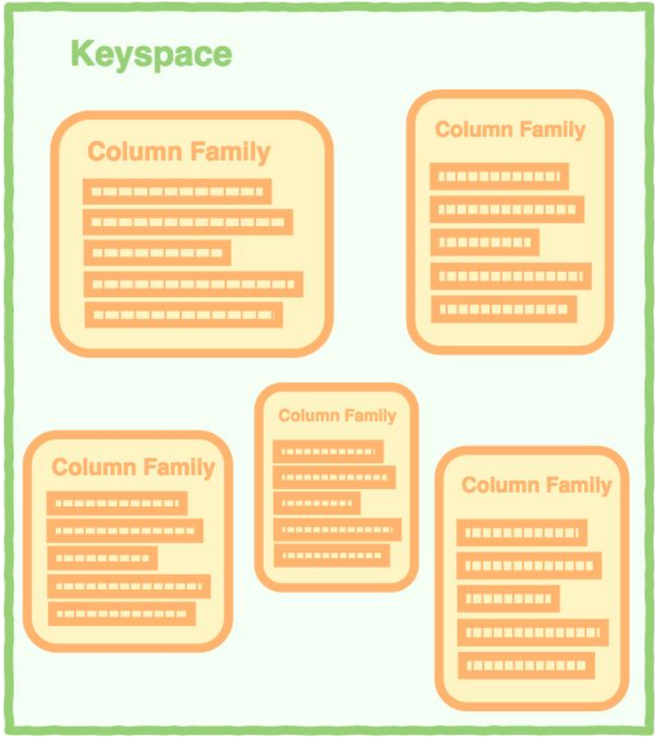
- Data is organized in rows with a primary key
- Stored in a distributed sparse multidimensional sorted map
- Data is retrieved by key per column family

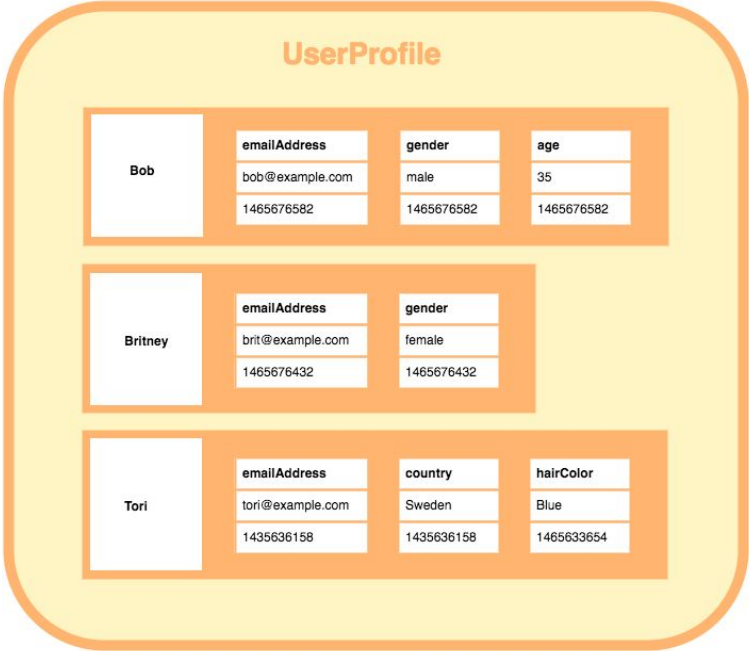
APACHE HBASE

cassandra



Figures from https://database.guide/what-is-a-column-store-database/





Bob

| emailAddress | gender | age |
|---|---|---|
| bob@example.com | male | 35 |
| 1465676582 | 1465676582 | 1465676582 |

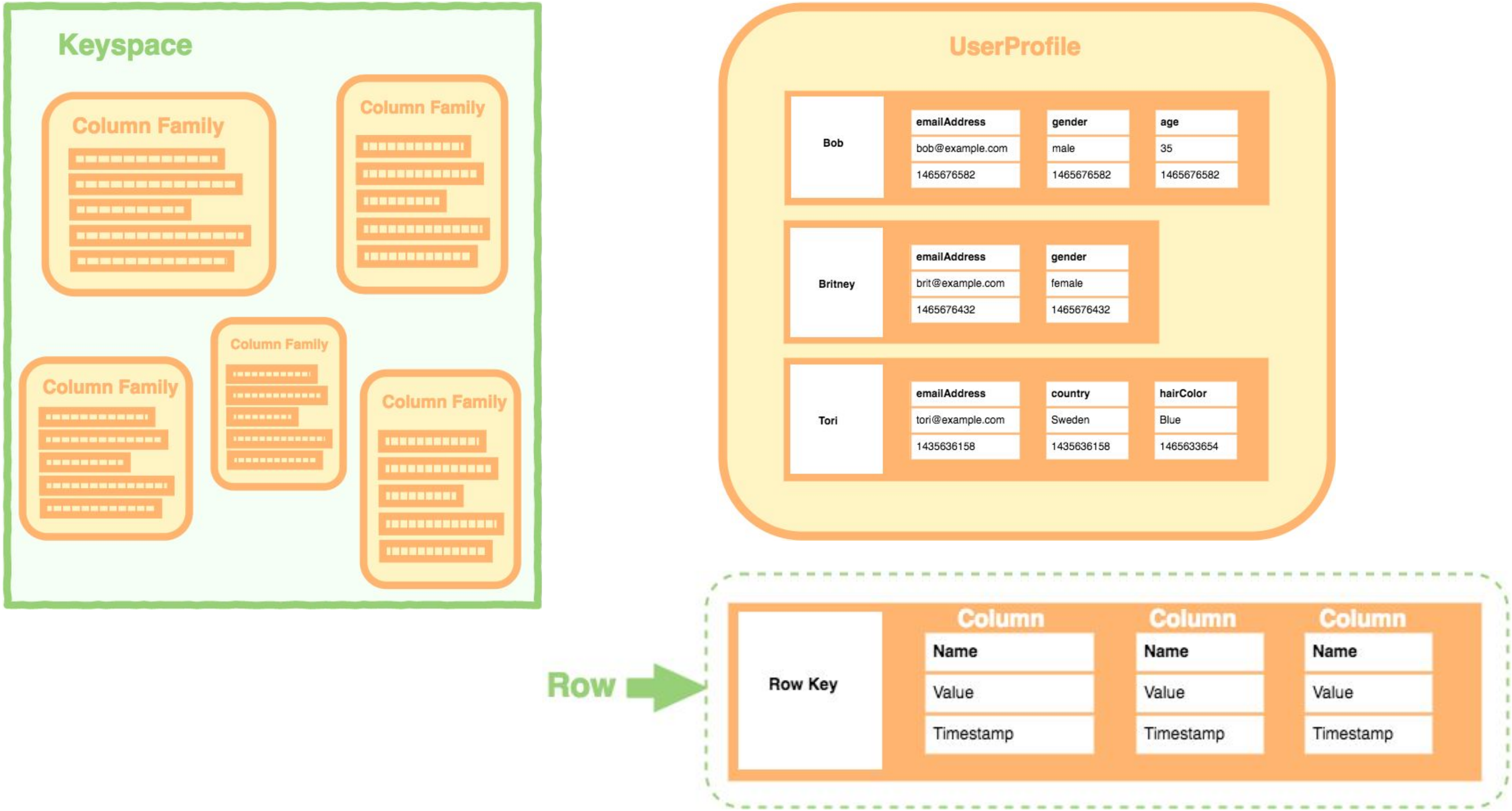
Britney

| emailAddress | gender |
|---|---|
| brit@example.com | female |
| 1465676432 | 1465676432 |

Tori

| emailAddress | country | hairColor |
|---|---|---|
| tori@example.com | Sweden | Blue |
| 1435636158 | 1435636158 | 1465633654 |

Figures from https://database.guide/what-is-a-column-store-database/



Keyspace

Column Family

Column Family

Column Family

Column Family

Column Family

UserProfile

Bob

| emailAddress | gender | age |
|---|---|---|
| bob@example.com | male | 35 |
| 1465676582 | 1465676582 | 1465676582 |

Britney

| emailAddress | gender |
|---|---|
| brit@example.com | female |
| 1465676432 | 1465676432 |

Tori

| emailAddress | country | hairColor |
|---|---|---|
| tori@example.com | Sweden | Blue |
| 1435636158 | 1435636158 | 1465633654 |

Row

Row Key

| Column | Column | Column |
|---|---|---|
| Name | Name | Name |
| Value | Value | Value |
| Timestamp | Timestamp | Timestamp |

Figures from https://database.guide/what-is-a-column-store-database/

# When to use them?

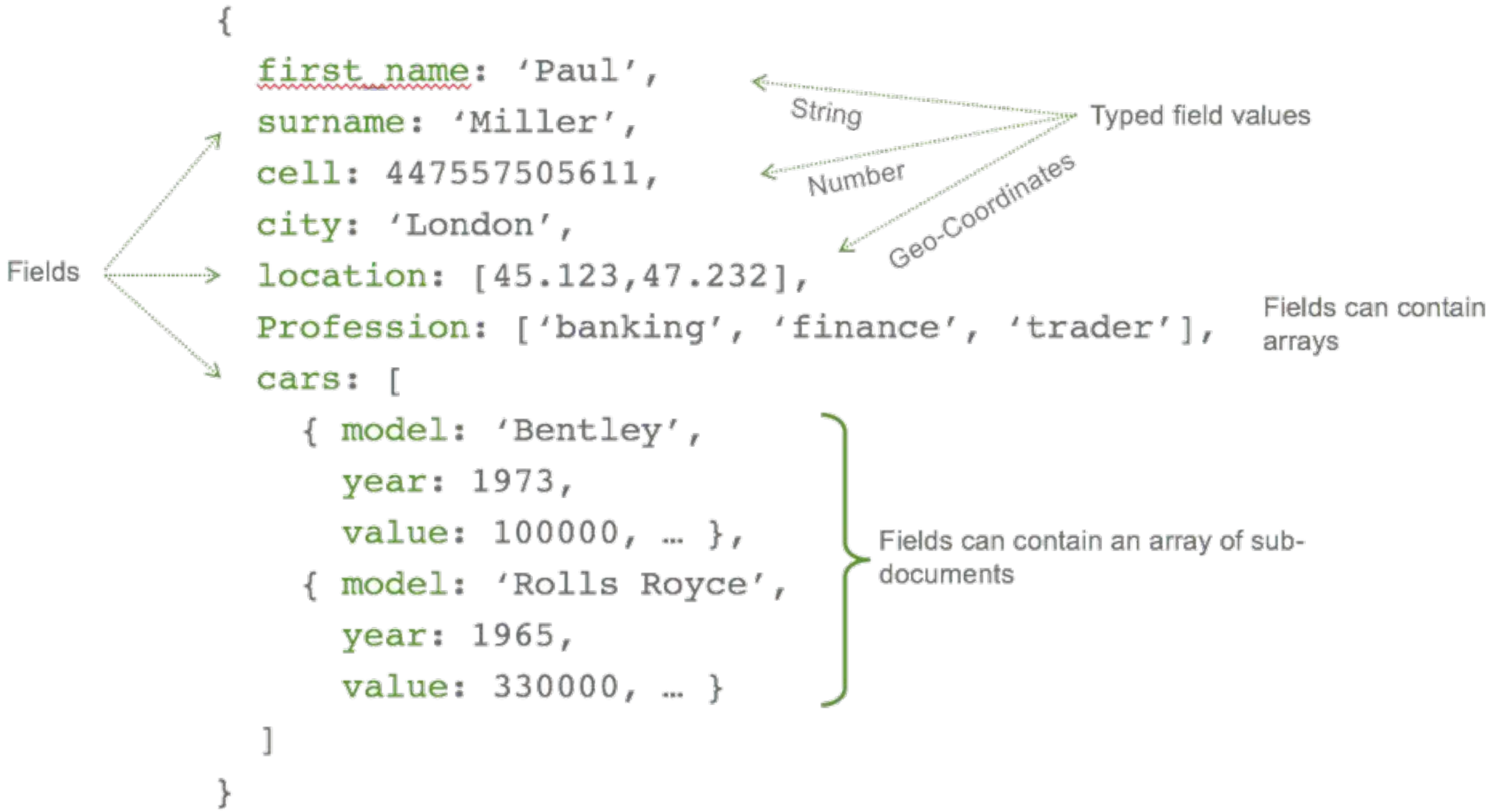
- Key-value and column DB achieve good performance performance
  - Access pattern is simple and the format is opaque -> lots of optimization opportunities
  - Column family DB is good for aggregation queries (average, sum, etc)
- Applications that only query data by a single or a limited range of key

# Document DB

- Data stored as documents (often JSON)
  - A document has many fields and their values
  - Documents can be nested
  - They can have different fields
- Queries can be done over any field
- Documents are closely aligned with object-oriented programming
- Performance advantage: instead of having to combine data from multiple tables, everything about an object is in the same document

CouchDB

mongoDB®



```
{
  first_name: ‘Paul’,
  surname: ‘Miller’,
  cell: 447557505611,
  city: ‘London’,
  location: [45.123,47.232],
  Profession: [‘banking’, ‘finance’, ‘trader’],
  cars: [
    { model: ‘Bentley’,
      year: 1973,
      value: 100000, … },
    { model: ‘Rolls Royce’,
      year: 1965,
      value: 330000, … }
  ]
}
```

Typed field values

Fields can contain arrays

Fields can contain an array of sub-documents

Figure from https://studio3t.com/

# Graph DB

- Data is represented by a graph
  - Nodes and relationships have properties as < key, value >
- Useful when traversing relationships is important
  - For instance: social networks, supply chains, etc
- Can be inefficient for other operations
  - Often coupled with another db to store properties

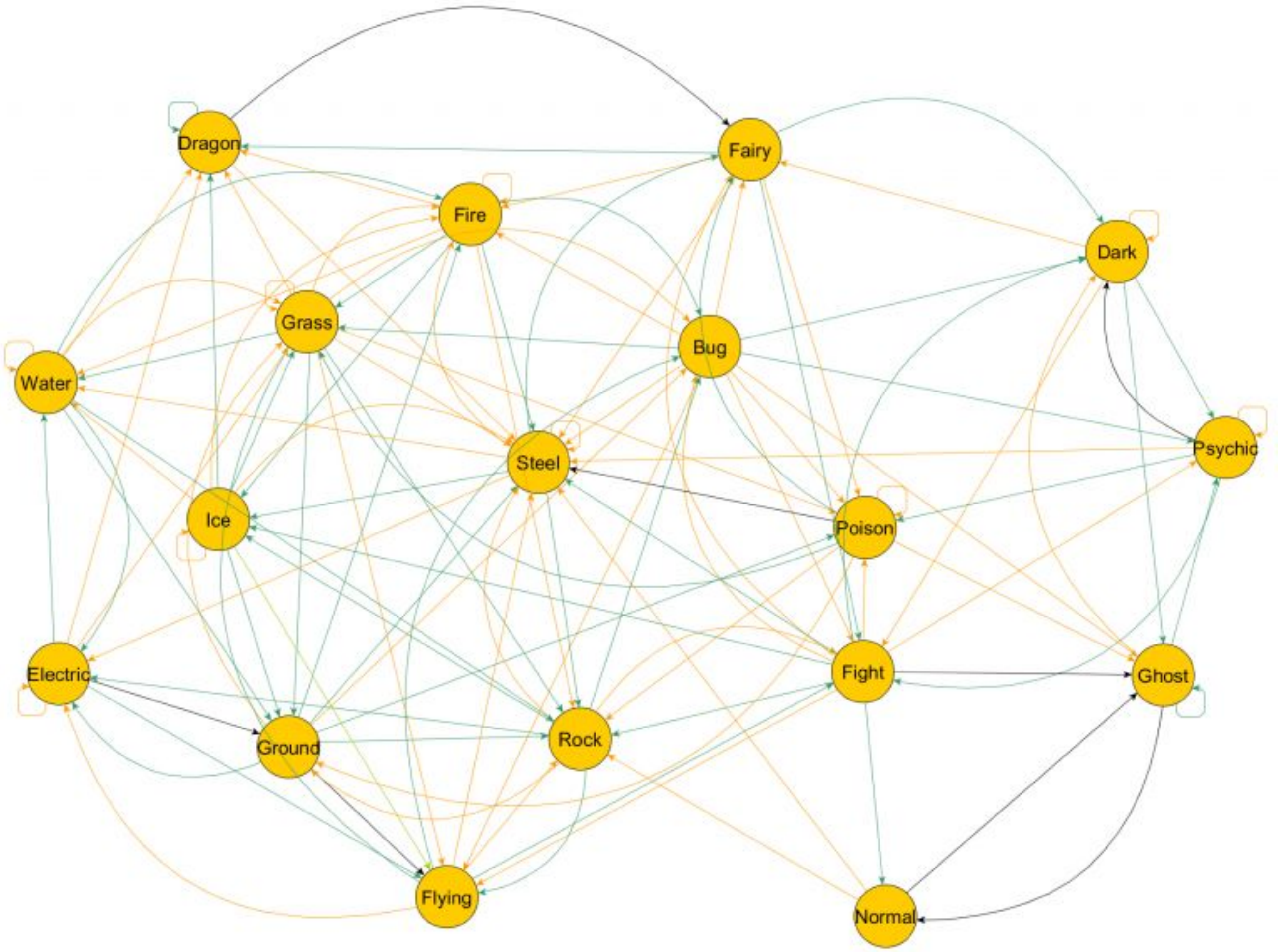
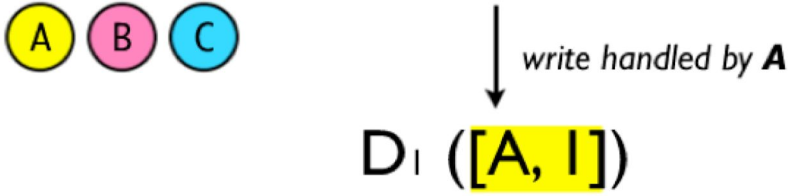


Figure from http://sparsity-technologies.com/blog/gotta-graphem-pokemon-graph-databases/

# Vector Clocks

- Classic algorithm for partial ordering of events in distributed systems (from 1988)
- Each process has a vector with clocks for all processes
  - Every internal event, it increases its own clock
  - Every message sent, it increases its own clock and sends the whole vector
  - Every message received, it increases its own clock and merges the vectors (by taking the maximum)



*write handled by* ***A***

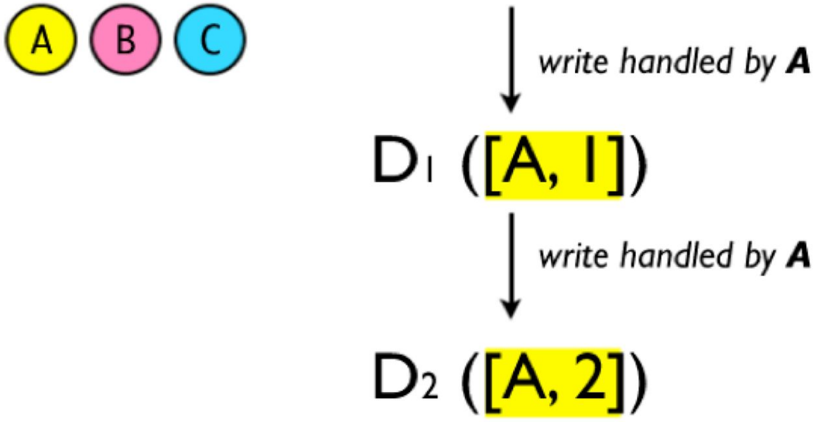
$D_1$ ([A, 1])

Causality-based partial order over events that happen in the system.

Document version history: a counter for each node that updated the document.

If all update counters in $V_1$ are smaller or equal to all update counters in $V_2$, then $V_1$ precedes $V_2$.

Source: slides by Lorenzo Alberton



*write handled by* ***A***

$D_1$ ([A, 1])

*write handled by* ***A***

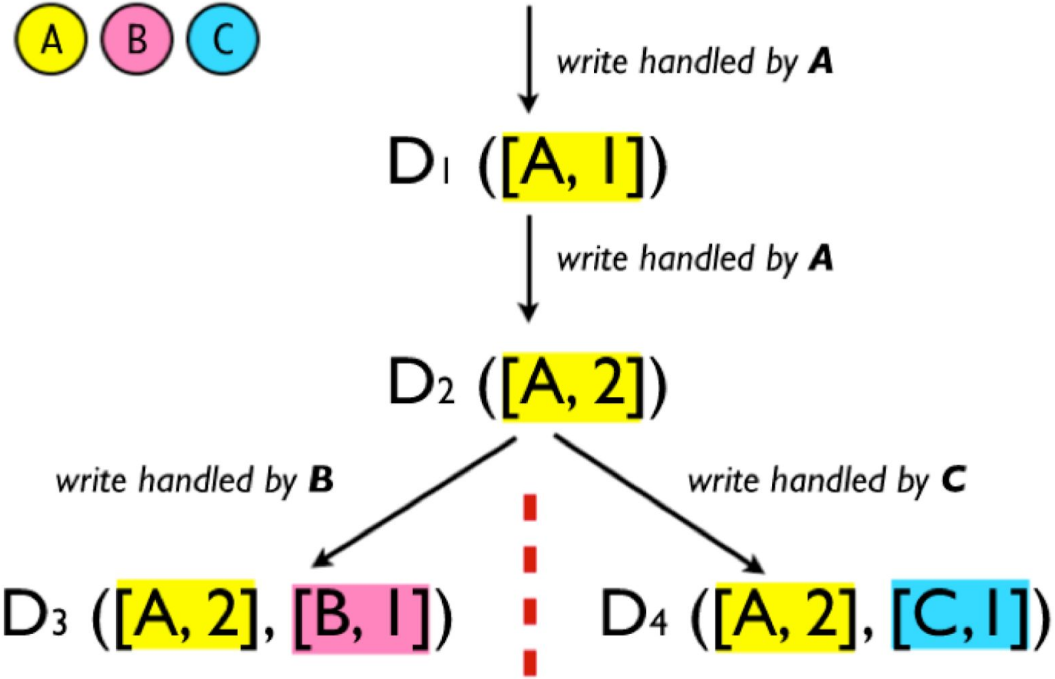
$D_2$ ([A, 2])

Causality-based partial order over events that happen in the system.

Document version history: a counter for each node that updated the document.

If all update counters in $V_1$ are smaller or equal to all update counters in $V_2$, then $V_1$ precedes $V_2$.

Source: slides by Lorenzo Alberton



A B C

write handled by ***A***

$D_1$ ([A, 1])

write handled by ***A***

$D_2$ ([A, 2])

write handled by ***B***

write handled by ***C***

$D_3$ ([A, 2], [B, 1])

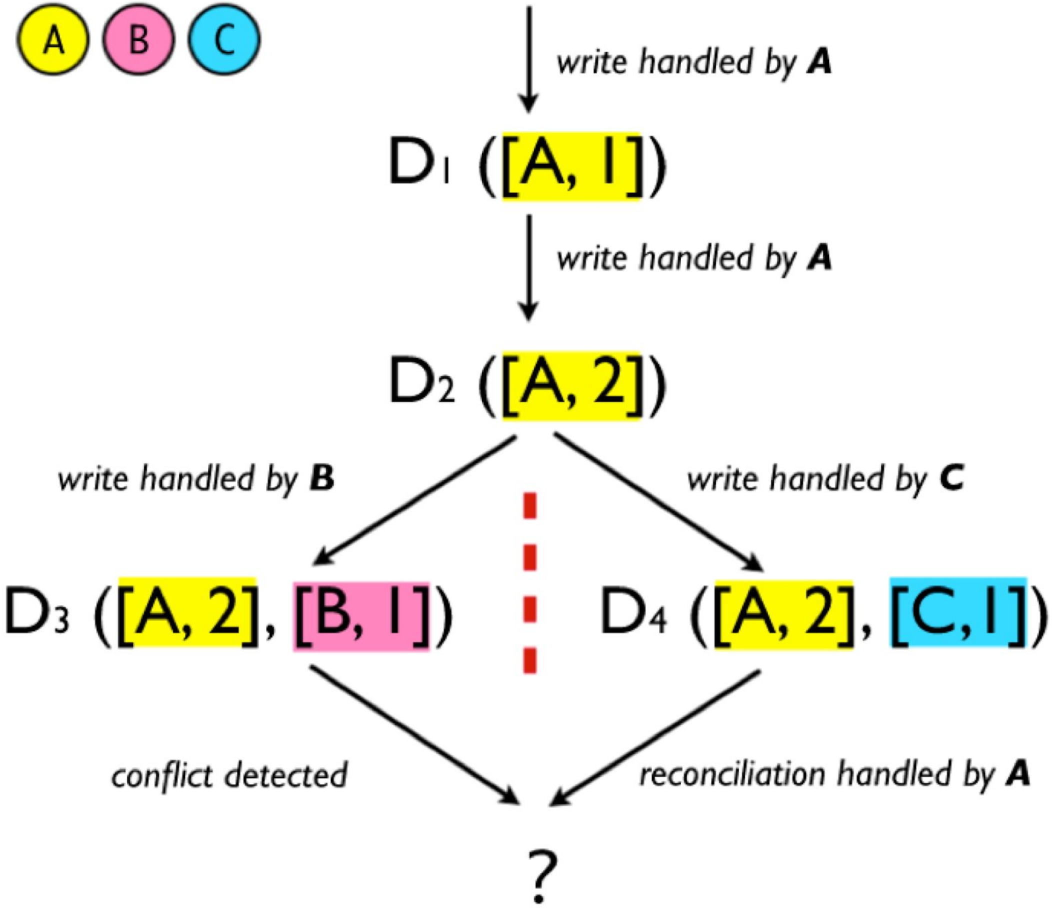
$D_4$ ([A, 2], [C, 1])

Causality-based partial order over events that happen in the system.

Document version history: a counter for each node that updated the document.

If all update counters in $V_1$ are smaller or equal to all update counters in $V_2$, then $V_1$ precedes $V_2$.

Source: slides by Lorenzo Alberton

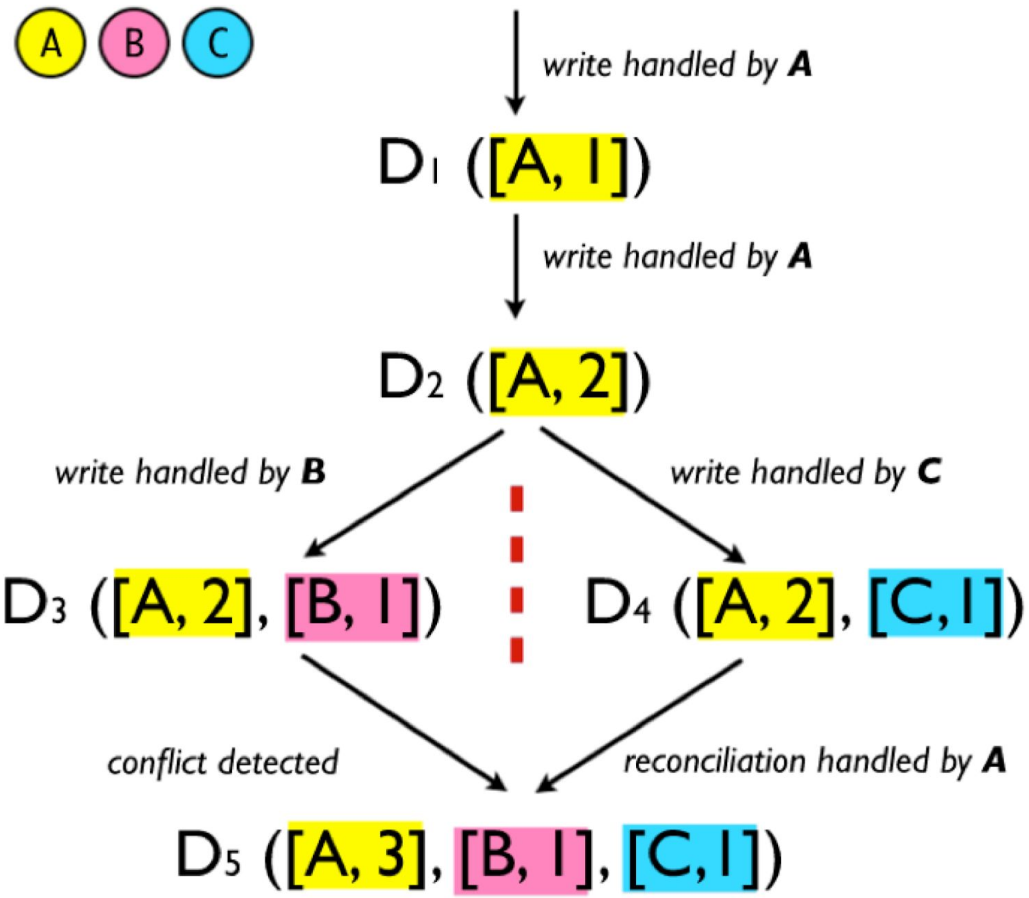


Causality-based partial order over events that happen in the system.

Document version history: a counter for each node that updated the document.

If all update counters in $V_1$ are smaller or equal to all update counters in $V_2$, then $V_1$ precedes $V_2$.

Source: slides by Lorenzo Alberton

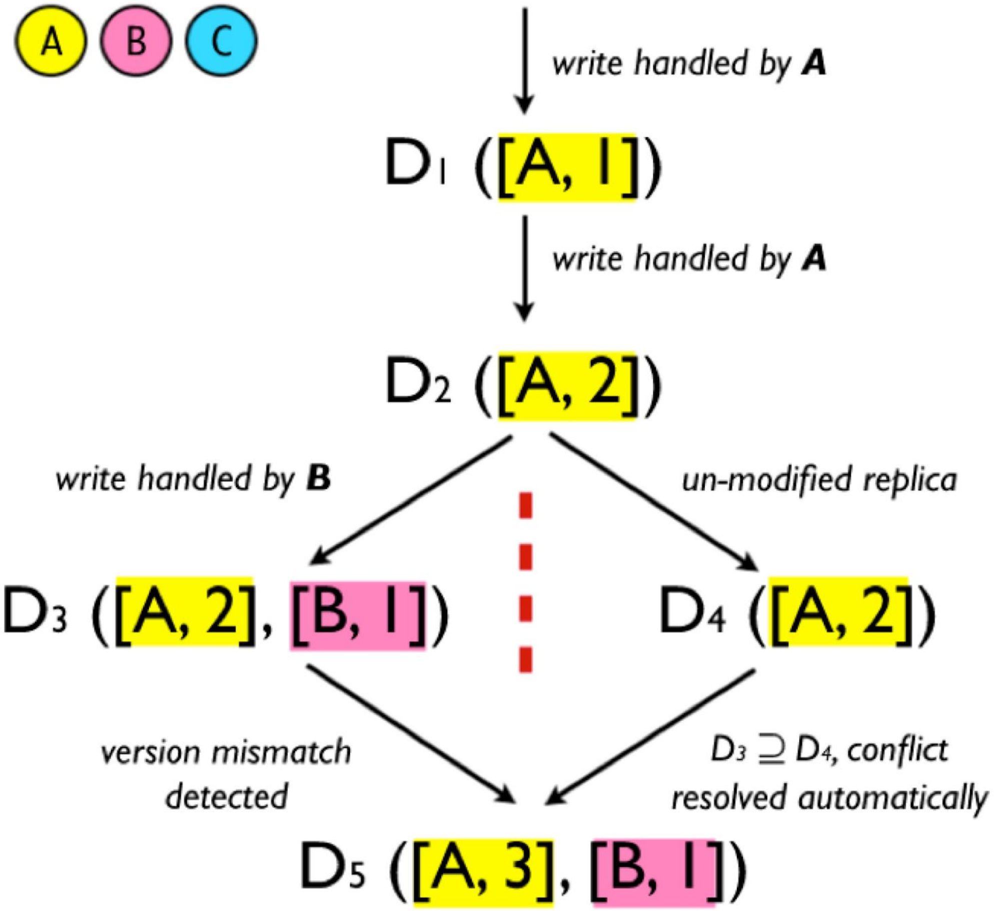


Causality-based partial order over events that happen in the system.

Document version history: a counter for each node that updated the document.

If all update counters in $V_1$ are smaller or equal to all update counters in $V_2$, then $V_1$ precedes $V_2$.

Source: slides by Lorenzo Alberton



Vector Clocks can *detect* a conflict. The conflict *resolution* is left to the application or the user.

The application *might* resolve conflicts by checking relative timestamps, or with other strategies (like merging the changes).

Vector clocks can grow quite large (!)

**Source: slides by Lorenzo Alberton**

# Reading

- For next class:
  - G. DeCandia et al. "Dynamo: amazon's highly available key-value store"
  - F. Chang et al. "BigTable: A distributed storage system for structured data"
- Illustrated proof of the CAP theorem:

  https://mwhittaker.github.io/blog/an_illustrated_proof_of_the_cap_theorem/
- Extra:
  - https://www.mongodb.com/nosql-explained
  - https://www.couchbase.com/resources/why-nosql
  - http://nosql-database.org/