

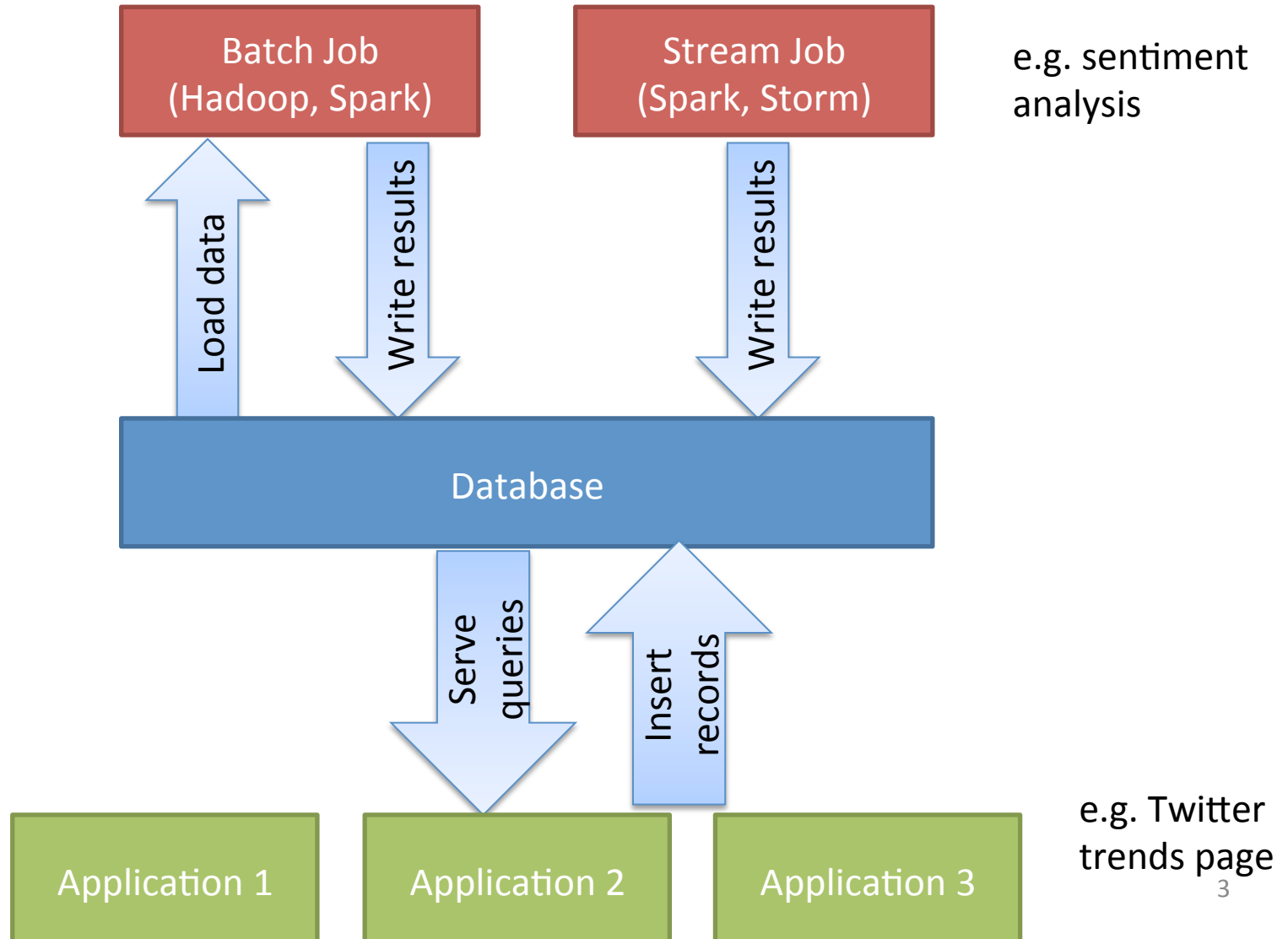
NoSQL Databases

Vincent Leroy

Database

- Large-scale data processing
 - First 2 classes: Hadoop, Spark
 - Perform some computation/transformation over a full dataset
 - Process **all** data
- Selective query
 - Access a **specific** part of the dataset
 - Manipulate only data needed (1 record among millions)
 - Database system

Processing / Database Link



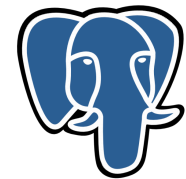
Different types of databases

- So far we used HDFS



- A file system can be seen as a very basic database
- Directories / files to organize data
- Very simple queries (file system path)
- Very good scalability, fault tolerance ...

- Other end of the spectrum: Relational Databases

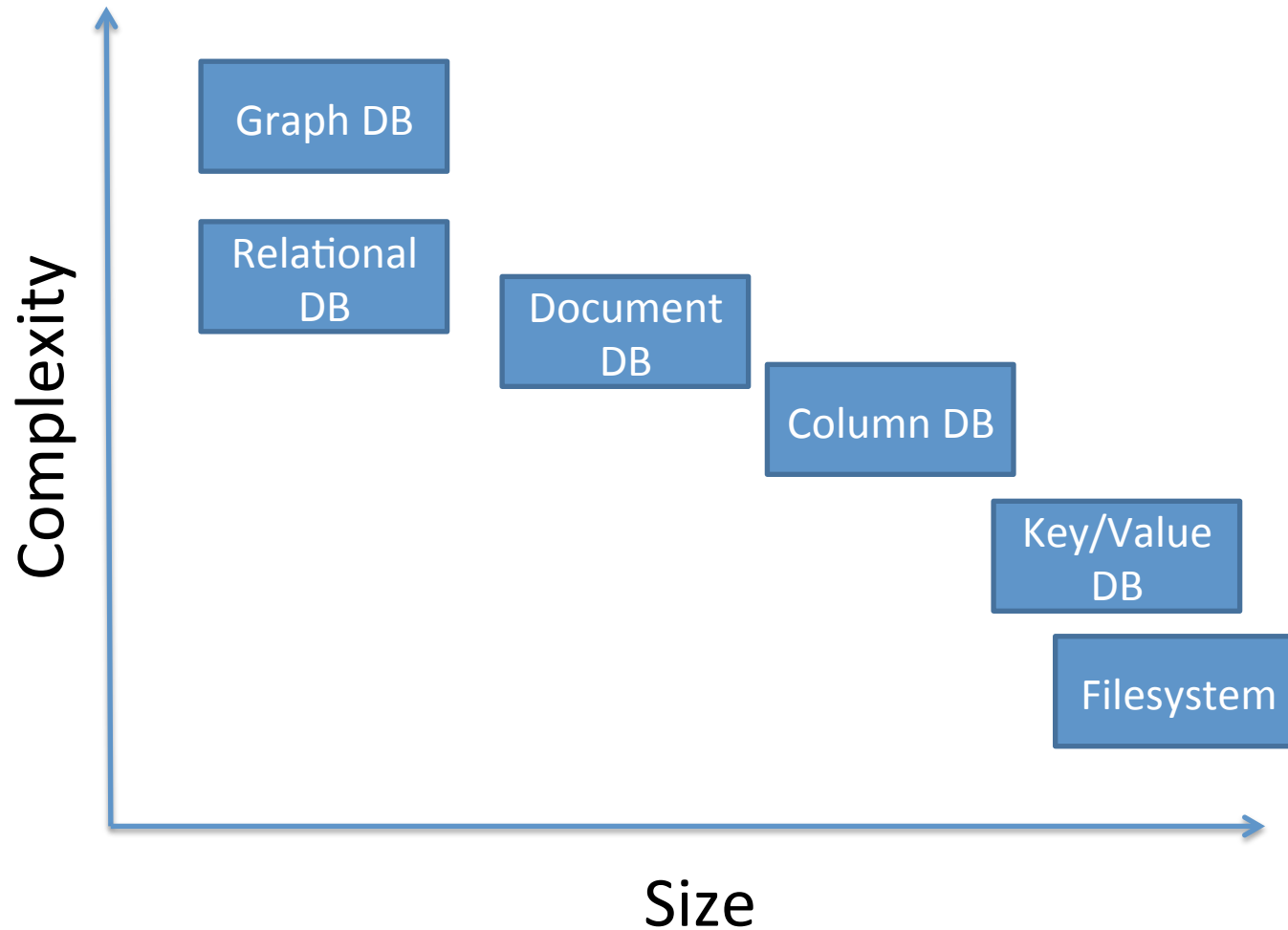


PostgreSQL
















- SQL query language, very expressive
- Limited scalability (generally 1 server)



Size / Complexity



The NoSQL Jungle

Document Database	Graph Databases
   	 
Wide Column Stores	Key-Value Databases
   	    

@cloudtxt <http://www.aryannava.com>

Goal of these slides

- Present an overview of the NoSQL landscape
 - Trade-off in choosing a solution
 - Theorems and principles
- Not a manual to learn specific DBs
 - Too many of them
 - Not that complicated (especially K/V stores)
 - Focus on Neo4j graph DB in lab work

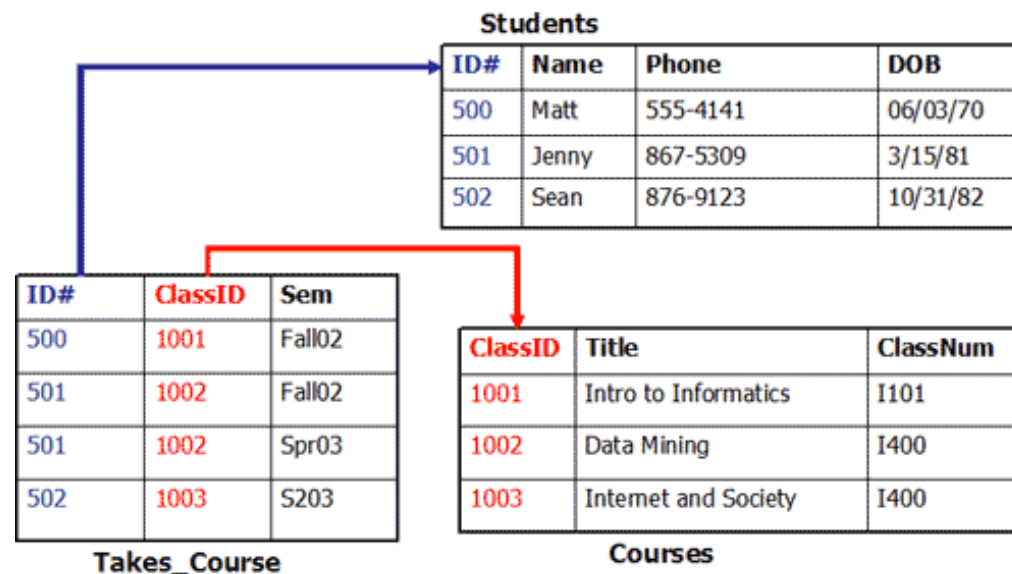
Relational Databases: SQL

- SQL language born 1974
 - Still used by most data processing systems (including Spark)
 - Learn it! Don't be a victim of the NoSQL hype!

Relational Databases model

- 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



ACID properties

- **A**tomicity
 - Transaction are all or nothing (e.g. when adding a bi-directional friendship relation, it's added both ways or not at all)
- **C**onsistency
 - Only valid data written (e.g. cannot say a student takes a course that is not in the courses table)
- **I**solation
 - When multiple transactions execute simultaneously, they appear as if they were executed sequentially (aka serializability)
- **D**urability
 - 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

Why NoSQL then?

- What does NoSQL mean?
 - No SQL
 - New SQL
 - Not only SQL ...
- SQL strong properties limit its ability to scale to very large datasets
 - Relax some properties to deal with larger datasets (~~ACID~~)
 - *But at what cost?*
- SQL is very structured (each record has the same columns ...), Web data often is not
 - Semi-structured data
 - Unstructured data
 - Graph data

CAP

- Consistency
 - When multiple operations execute simultaneously, it appears as if they were executed one after the other (A of ACID)
- Availability
 - Every request received by a non failed node must be answered
- Partition tolerance
 - System must respond correctly even if network fails

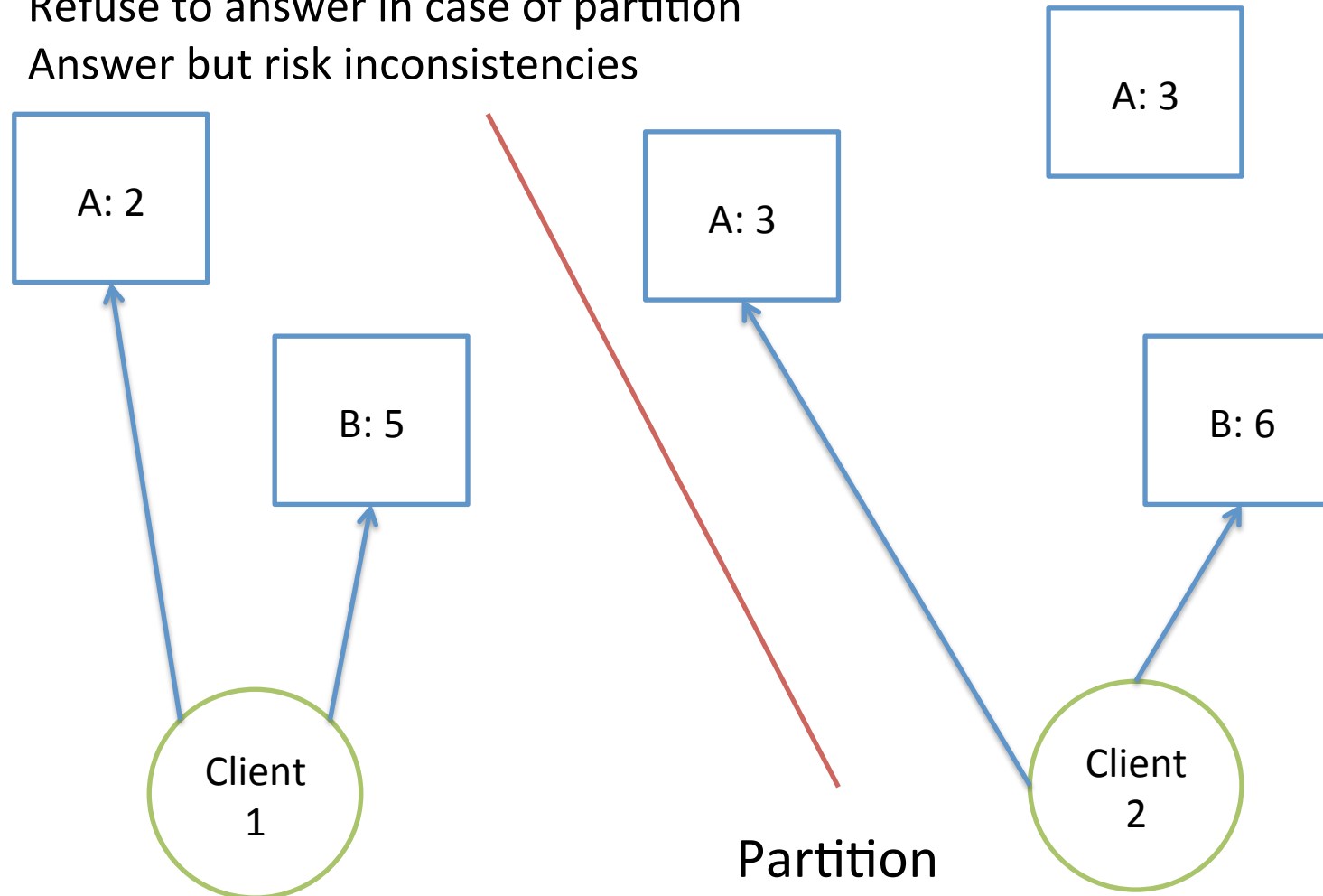
CAP theorem

- Impossible to have 3 simultaneously
 - Choose CA, CP, or AP
 - In a centralized system, no need for P
 - Relational databases have CA
 - In a distributed system, you cannot ignore P
 - Distributed databases choose CP or AP

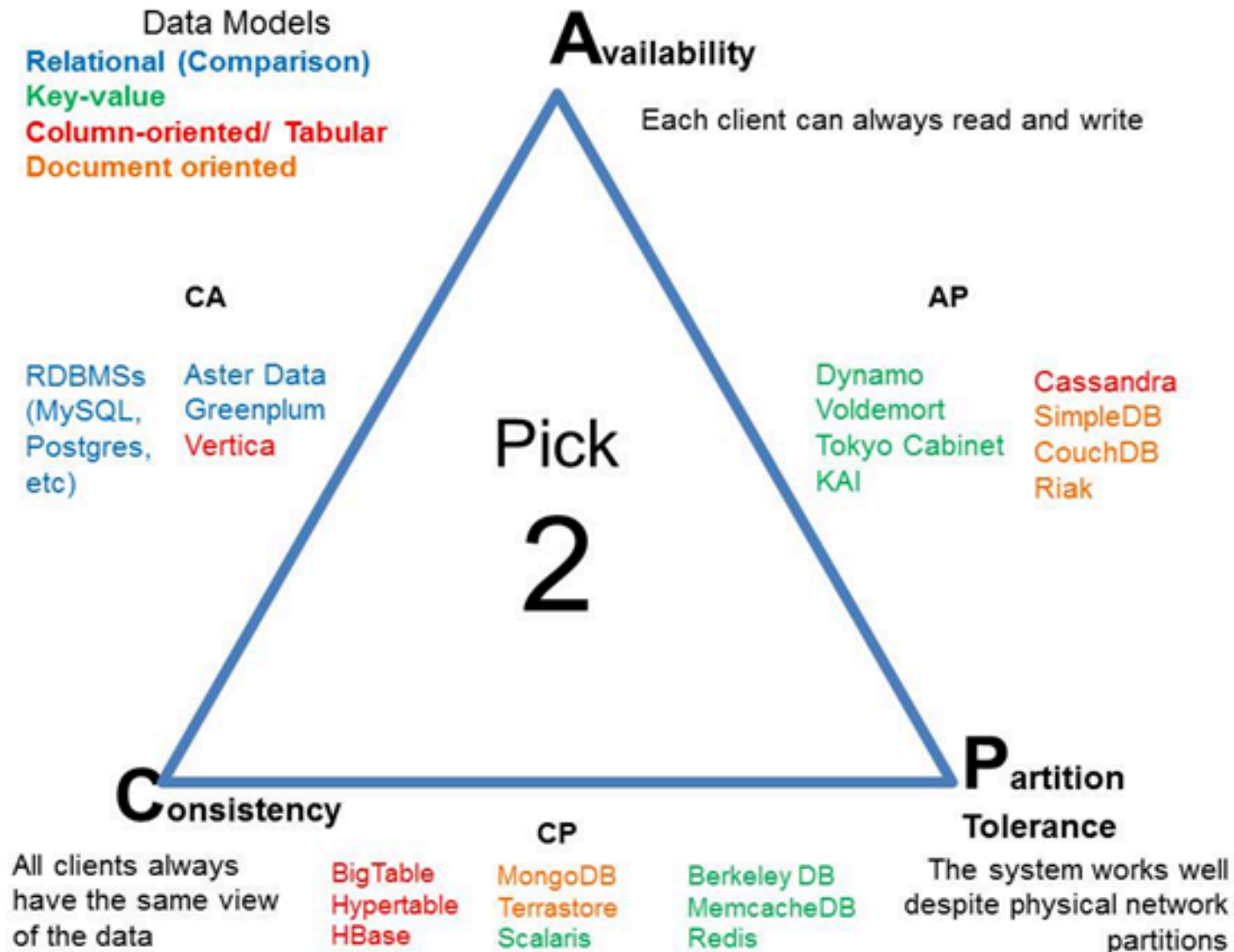
CAP intuition

2 solutions:

- Refuse to answer in case of partition
- Answer but risk inconsistencies



NoSQL and CAP



Weaker consistency models

- **Eventual** consistency
 - When there is no partition, DB is consistent
 - In case of partition, DB can return stale data
 - Once partition is gone, there is a time limit on how long it takes for consistency to return
- Different levels of consistency (consistency / cost trade-off)
 - Causal consistency
 - Read-your-writes consistency
 - Session consistency
 - Monotonic read consistency
 - Monotonic write consistency
 - Again, many choices, so many different systems

Vector clocks & conflict detection

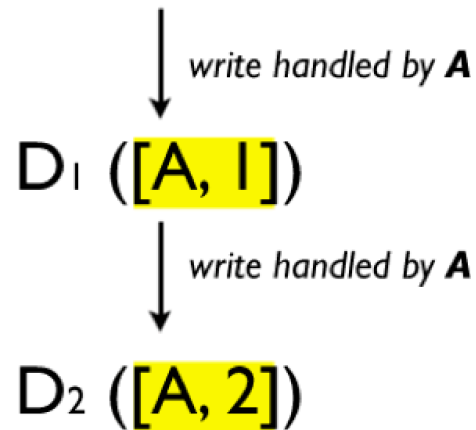


Causality-based partial order over events that happen in the system.

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 .

Vector clocks & conflict detection

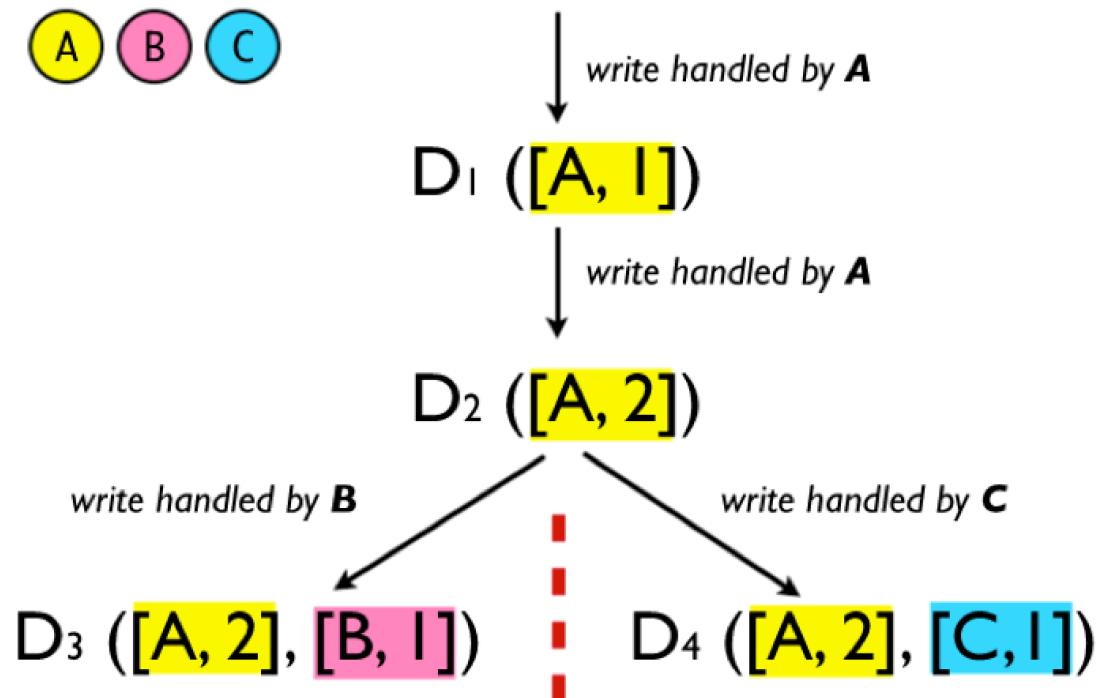


Causality-based partial order over events that happen in the system.

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 .

Vector clocks & conflict detection

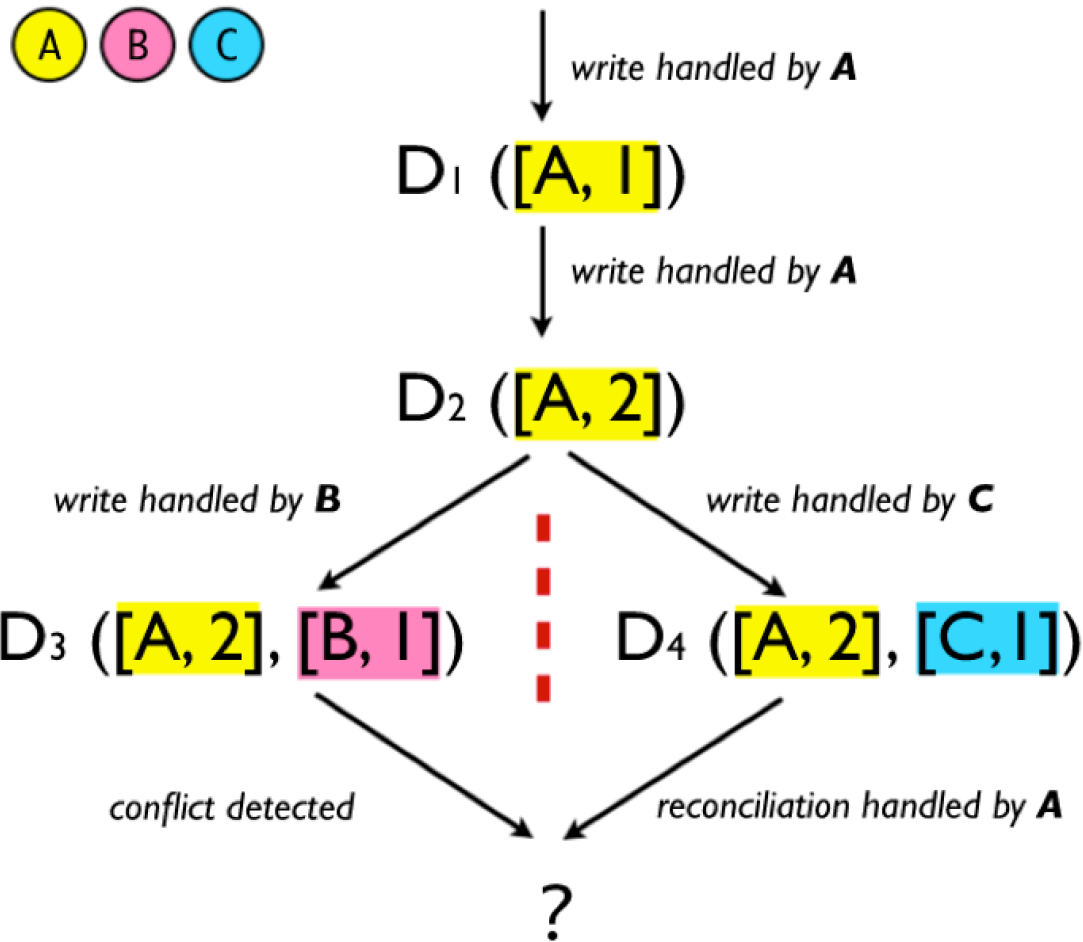


Causality-based partial order over events that happen in the system.

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 .

Vector clocks & conflict detection

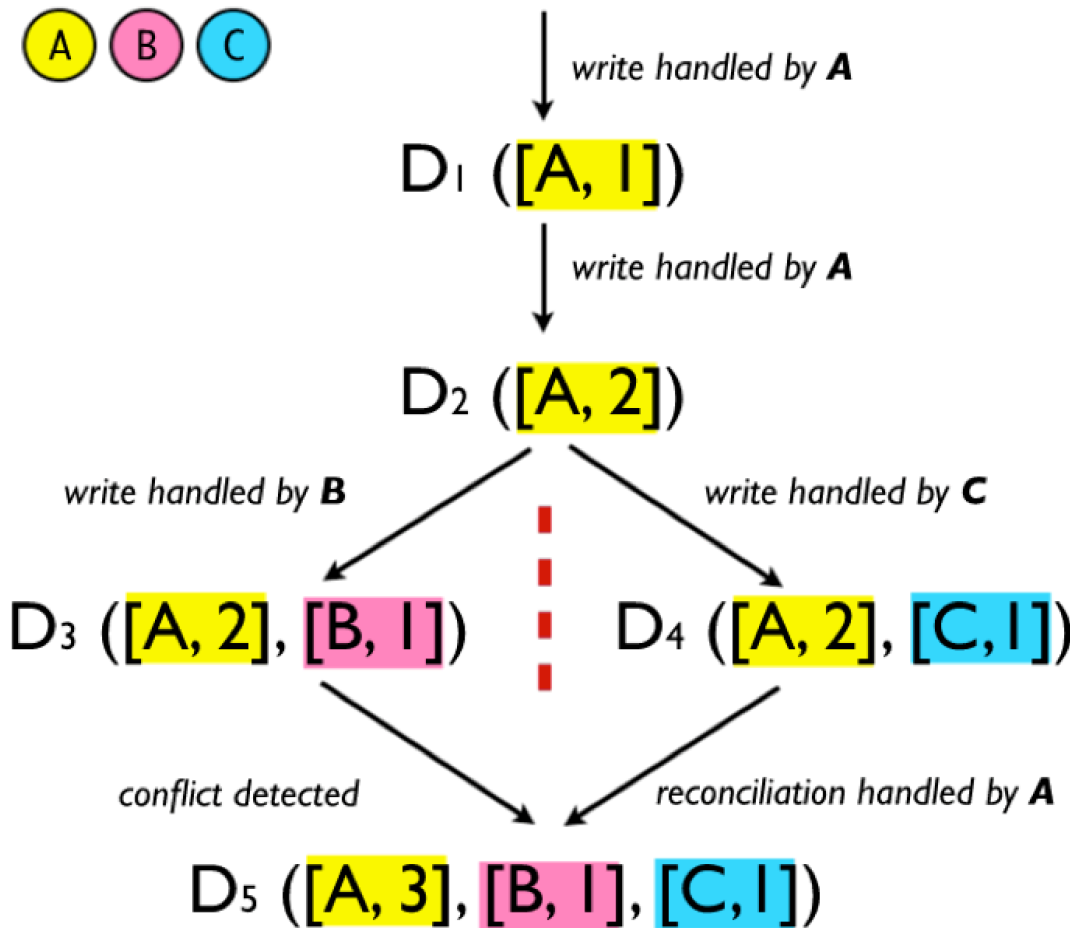


Causality-based partial order over events that happen in the system.

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 .

Vector clocks & conflict detection

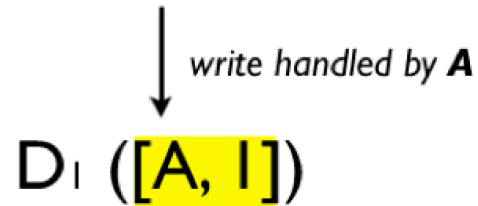
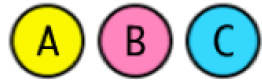


Causality-based partial order over events that happen in the system.

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 .

Vector clocks & conflict detection

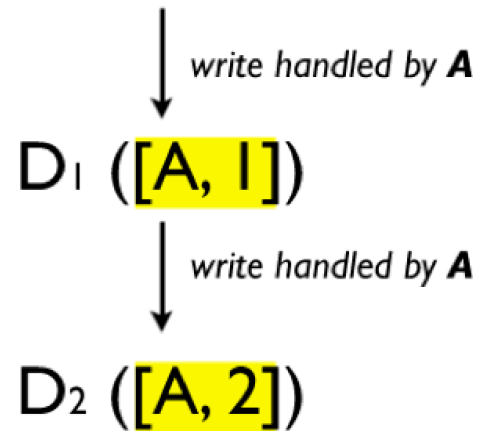


Vector Clocks can *detect* a conflict. The conflict *resolution* is left to the application or the user.

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

Vector clocks can grow quite large (!)

Vector clocks & conflict detection

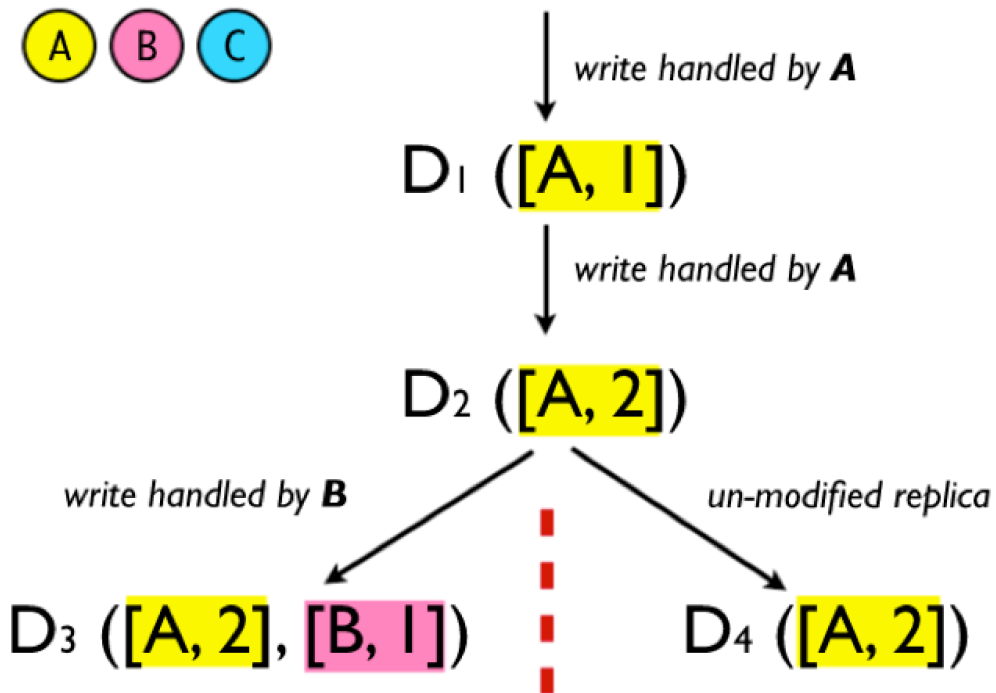


Vector Clocks can *detect* a conflict. The conflict *resolution* is left to the application or the user.

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

Vector clocks can grow quite large (!)

Vector clocks & conflict detection

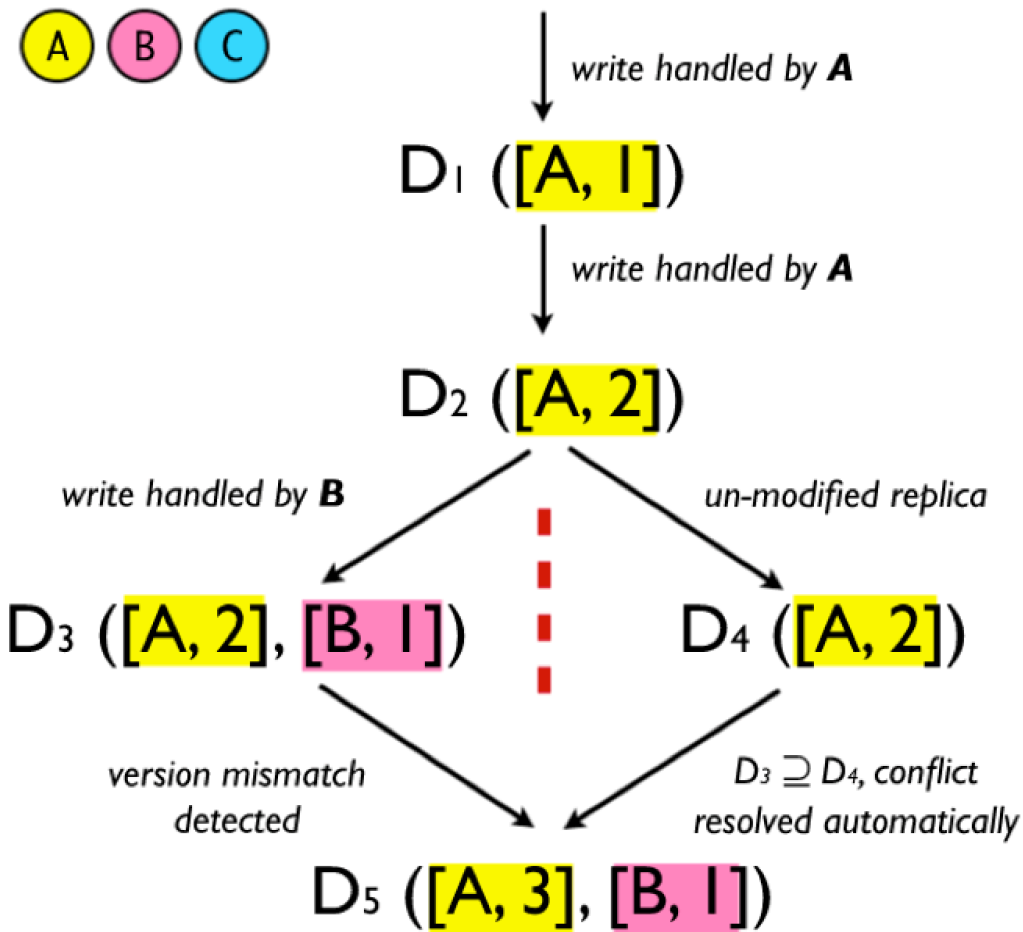


Vector Clocks can *detect* a conflict. The conflict *resolution* is left to the application or the user.

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

Vector clocks can grow quite large (!)

Vector clocks & conflict detection



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 (!)

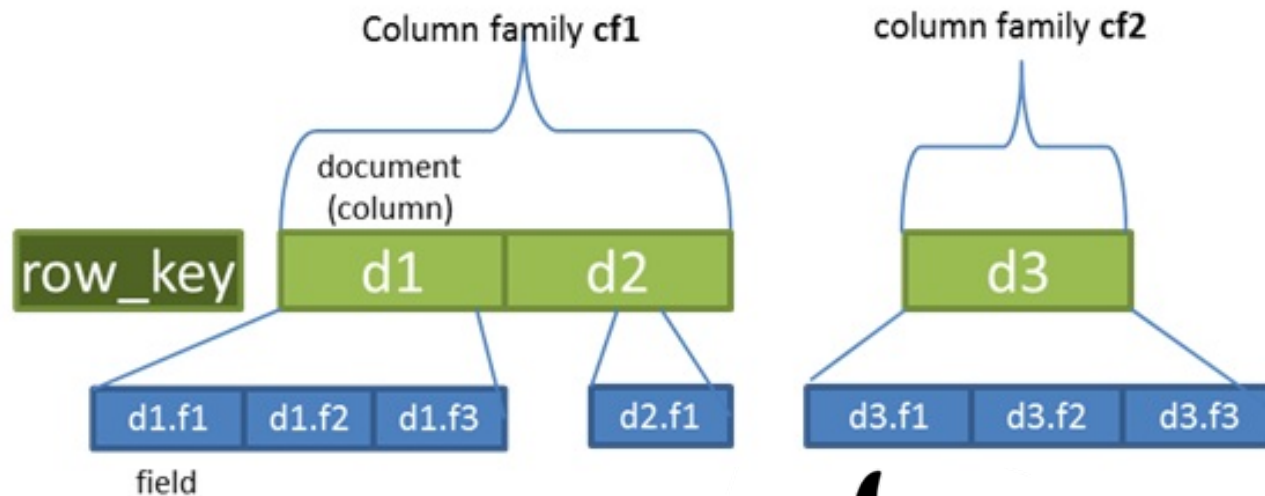
Key/Value store

- 2 basic operations, similar to the HashMap data structure
 - Put(K,V)
 - Get(K)
- Often used for caching information in memory
 - Facebook uses them a lot



Column/Tabular DB

- Data organized as rows with a primary key
 - Flexible format, each row can have different fields in a column family
 - Relies on HDFS for fault tolerance



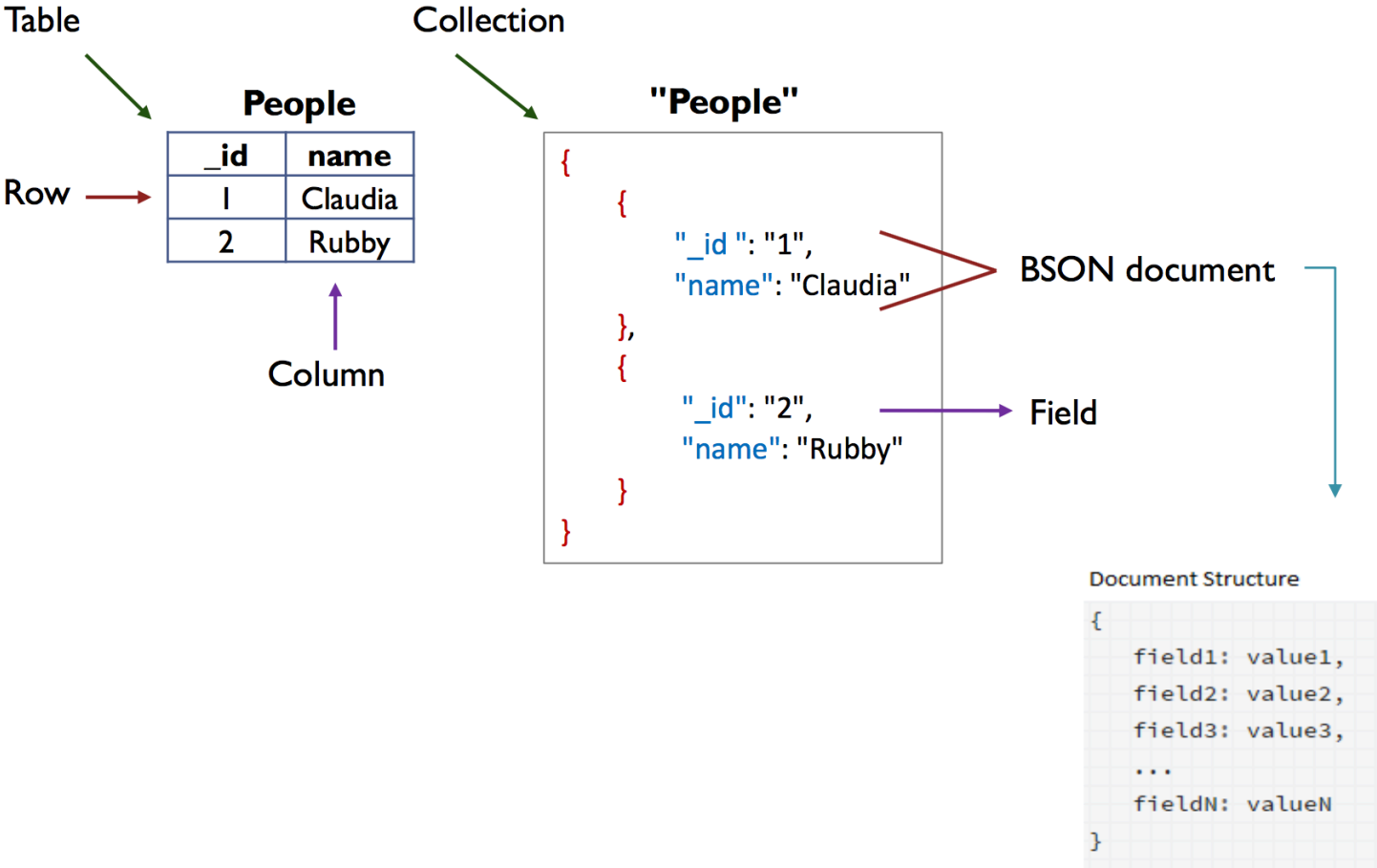
Document DB

- Data stored as documents (often JSON)
- Richer than K/V stores
 - Insert
 - Delete
 - Update
 - Find
 - Aggregation functions (Map, Reduce ...)
 - Indexes



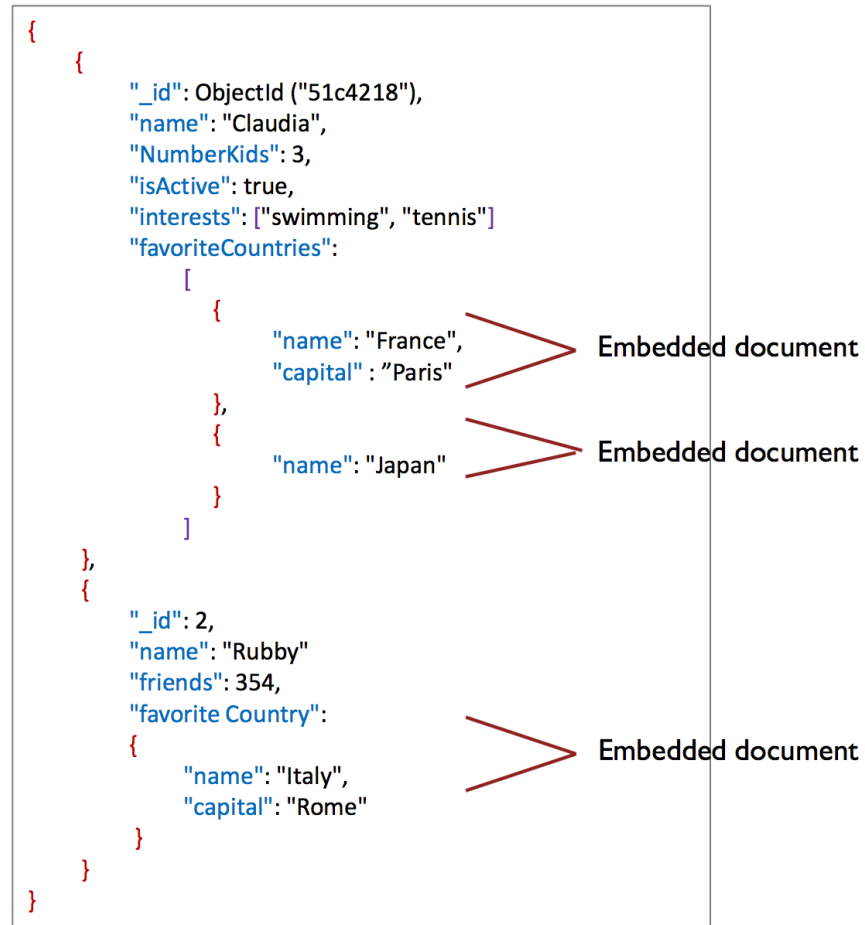
mongoDB

Document DB



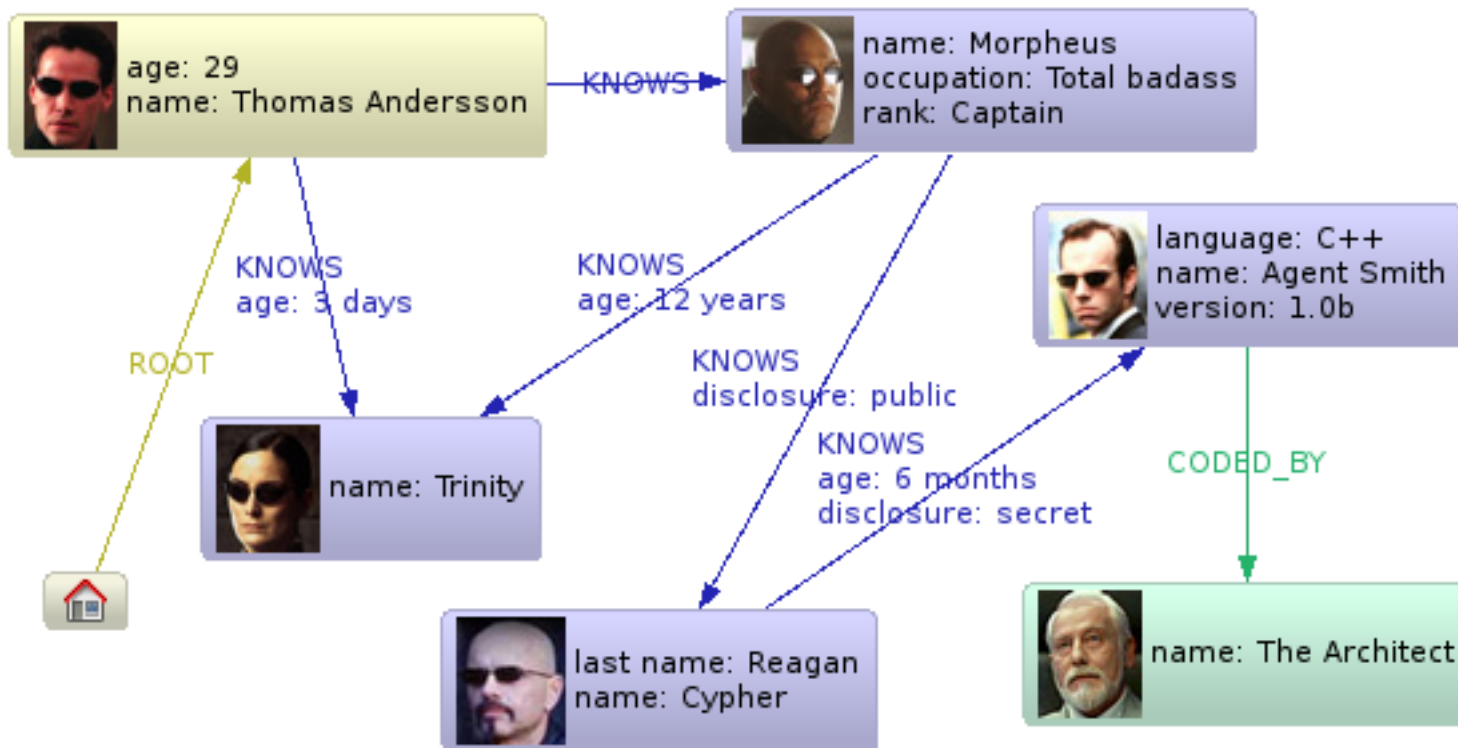
Document DB

A document can have one or more documents inside.



Graph DB

- Represent data as graphs
 - Nodes / relationships with properties as K/V pairs



Graph DB: Neo4j

- Rich data format
 - Query language as pattern matching
 - Limited scalability
 - Replication to scale reads, writes need to be done to every replica

Cypher Query Language

