NoSQL concepts

Worldline

M. Fanilo Andrianasolo

UNIVERSITÉ LUMIÈRE LYON 2
UNIVERSITÉ DE LYON

Data Analytics Tech Lead & Product Manager

Worldline

2013 - 2016	2016 - 2017	2017-2019	2019-current
Big Data Engineer	Data Analytics Evangelist	Data Science Tech Lead	Data Analytics Product Manager

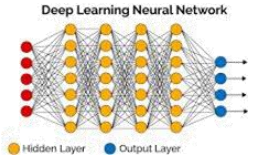
hadoop Spark docker +ableau SOFTWARE

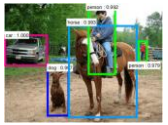
Data is at the center of all IT activities

Data

Hardware

Innovations

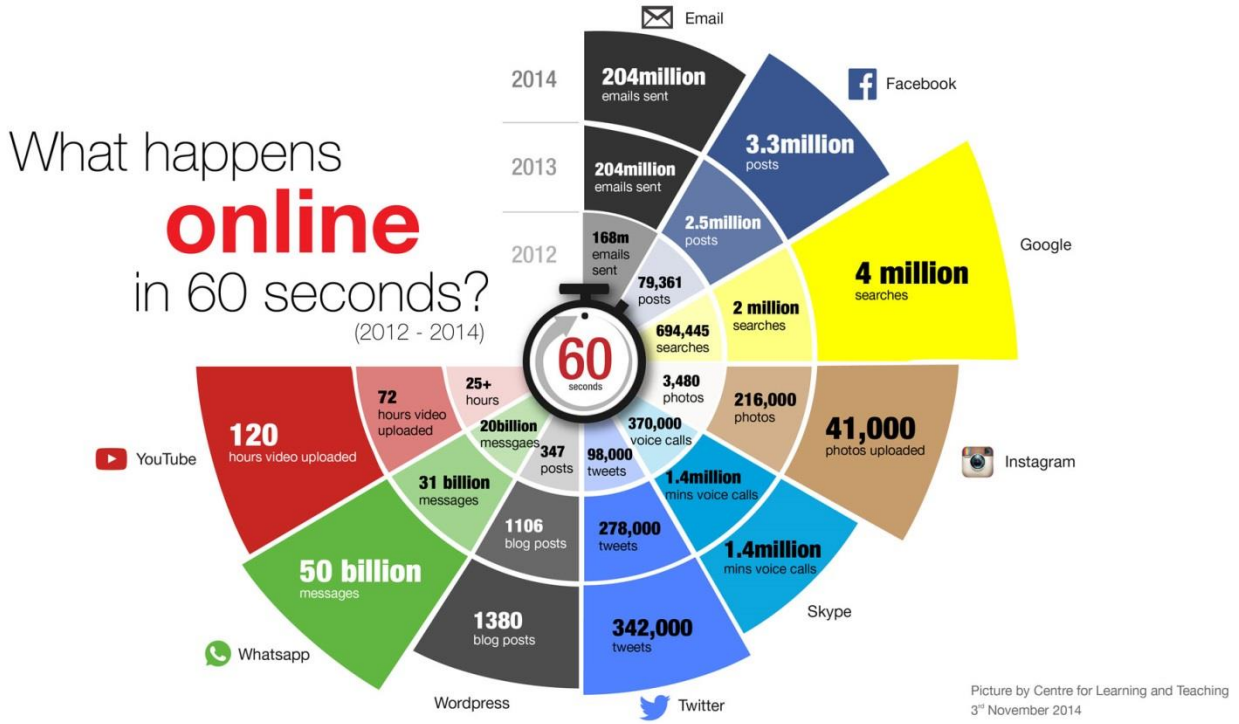




Data Scientist:

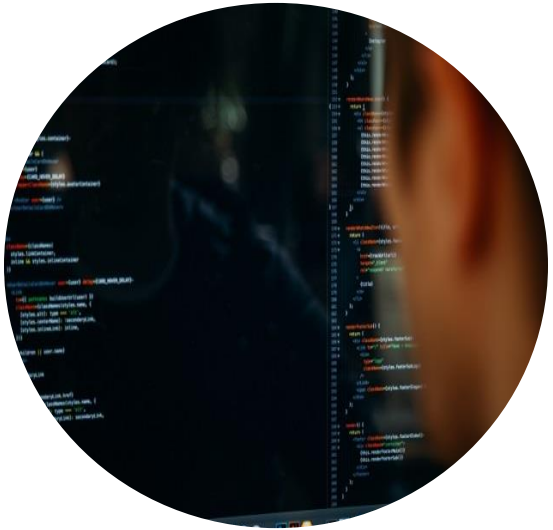
The Sexiest Job of the 21st Century

Problem



How should we store and query such data ?

Agenda



- ❑ Establishing the role of NoSQL
- ❑ NoSQL data stores
- ❑ CAP theorem
- ❑ Denormalization
- ❑ Hands-on

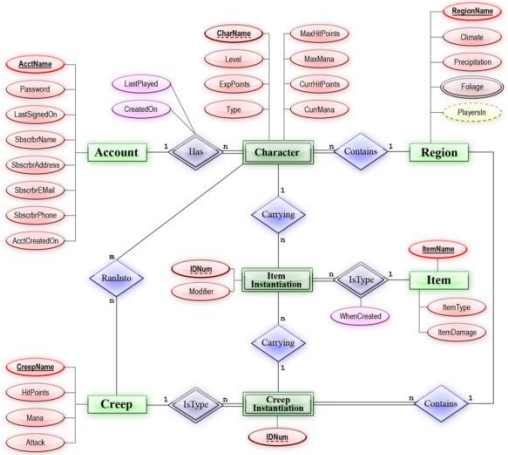
Establishing the role of NoSQL

RDBMS

Monopoly of RDBMS since 1970

- 1970: Edgar F. Codd theorizes the relational model
- 1974: Creation of the SQL language
- 1978: Multics, 1st commercial solution
- 1979: 1st Oracle solution
- 1986: SQL is standardized
- 2000: MySQL is open-sourced

RDBMS



Relational paradigm

- Database = set of tables and relations
- Table = set of tuples (rows) structured under columns
- Column = schema constraint on the data

Perfect for business informatics

The emergence of Not Only SQL

Big Data and the Web

- Velocity, Variety, Volume
- Simple storage problems

… limited by their tools

- Vertical scalability
- Relational paradigm is rigid
- Expensive !

Best quality of service

- Aim for no service interruption
- Aim for best user experience
- Service/data in constant evolution

The emergence of Not Only SQL

Summer 2009: NoSQL Meetup in California

- Talk about distributed storage systems
- Reuse the NoSQL term from Carlos Strozzi in 1998
- Voldemort (LinkedIn), Cassandra (Facebook), Dynomite, Hbase, Hypertable, CouchDB

A definition

> Non-relational, distributed, horizontally scalable and open-source database

http://nosql-database.org

Beyond the relational model

- Consistency makes it hard to handle scalability and millions of users connecting
- Relational model rigidity makes it hard to modify a model on the fly without slowing the service

Complements (and doesn’t replace) SQL

- Responds to new requirements : scalability, specialization in data storage
- Different constrains & paradigms

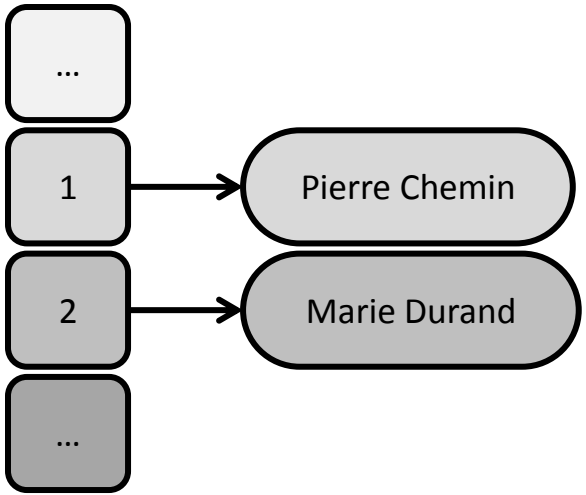
NoSQL Data stores

Data structures

So what data structures do you know in Computer Science ?

Key value stores

- N keys of type String
- For each key, an opaque value
- CRUD operations



⊕ Simplicity, performance

⊖ Cannot perform query on values

⊖ Not suited for complex modelisations

Ex : Redis, Riak, Memcached, Voldemort

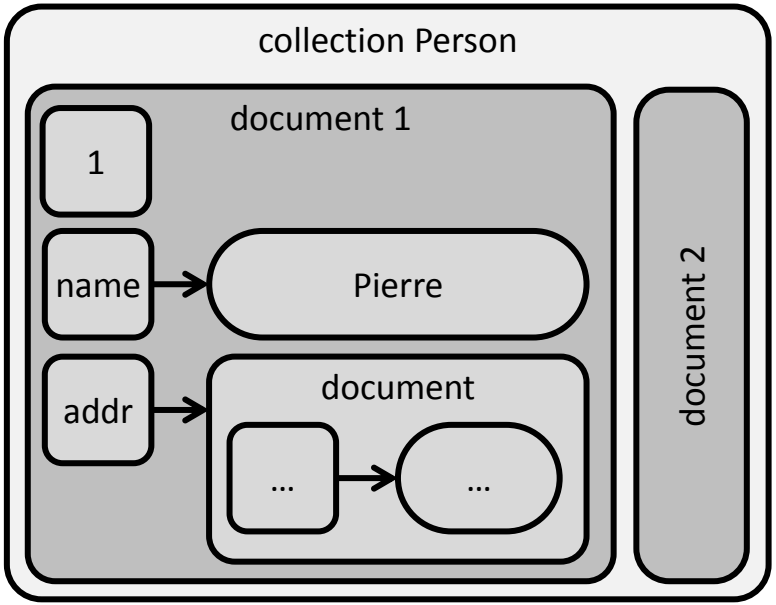
Document

- N collections of documents
- A document = id + k fields
- A field = name + value
- CRUD + query language

⊕ Native integration with object language

⊕ Querying on any field

⊖ Need to denormalize

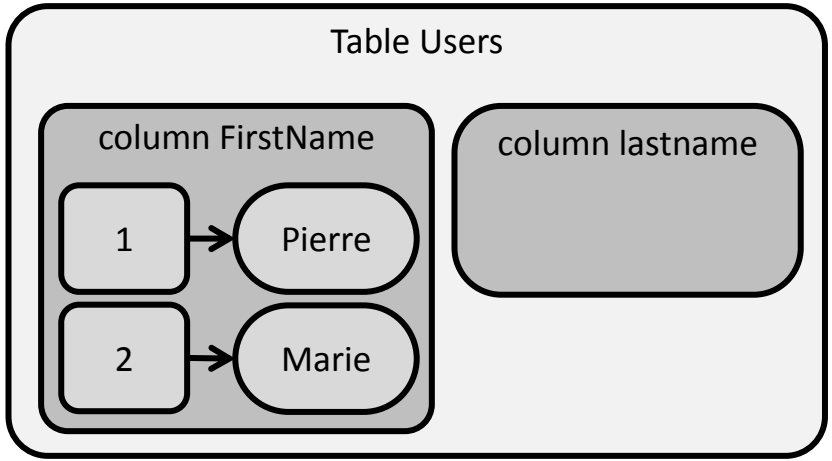


Ex : MongoDB, CouchDB, RethinkDB

RethinkDB

Column-oriented

- A table = N columns
- A tuple = set of columns associated to the same key
- CRUD + query per column



⊕ Tuple is a well known entity

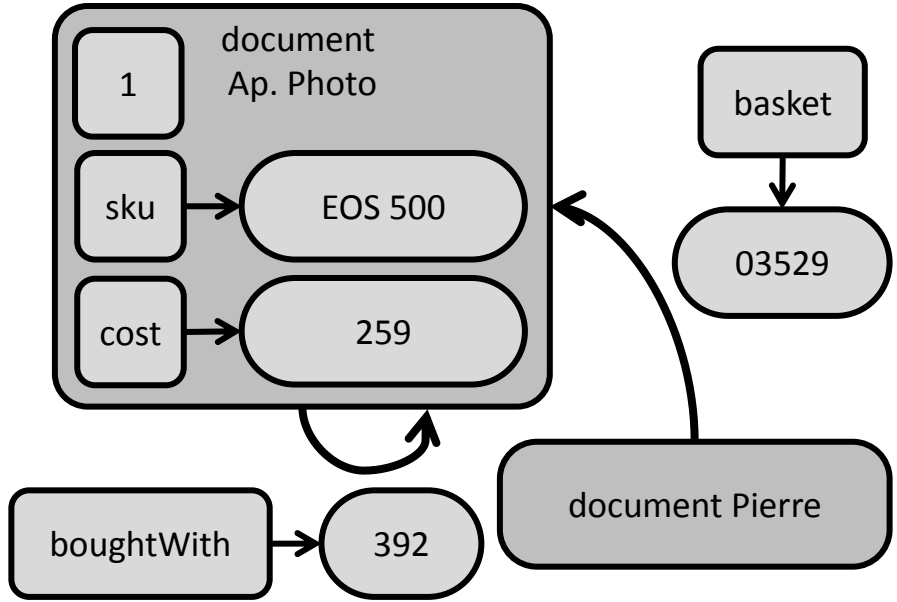
⊕ Easy to partition and rebalance

⊖ Need to denormalize

Ex: Hbase, Cassandra, HyperTable

Graph

- Vertices linked by edges
- Vertex = a document
- Edge = a document with origin and destination
- CRUD + query langague



⊕ Modelisation close to object oriented

⊖ More complex usage

⊖ Not as scalable as other

Ex: Neo4j, OrientDB, Titan (Cassandra / Hbase)



Practice ?

- ❑ File caching (Web static resources)
- ❑ Volatile data (counters, chats, basket…)

- ❑ Analysis of transactions (eCommerce, finance…)
- ❑ Content storage (mail, messages…)

- ❑ Content Managing System (Wiki, Blog, Forum…)
- ❑ Internet of Things (Machine2Machine, sensors)

- ❑ Recommendation/suggestion (retail, social…)
- ❑ Social networks

Fanilo → 36

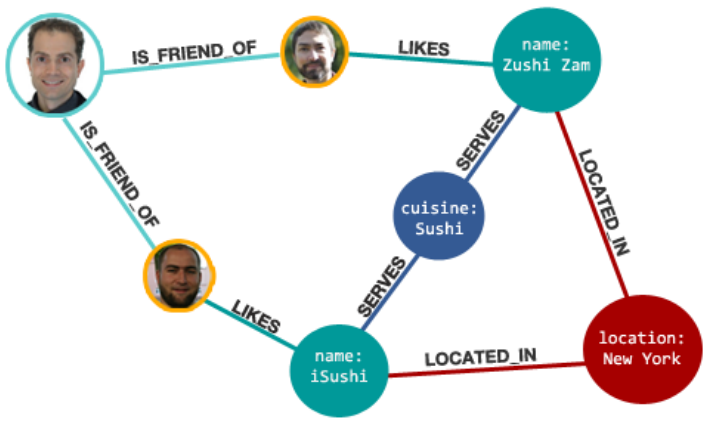
Adam → 54

Weather → [cold, hot]

Last_date → 09/08/2016

0	pomme		2,99	
7	poire		5,99	
3		red	12,00	music
		blue	89	

```
{ "menu":
  { "id": "file",
  "value": "File",
  "popup": {
    "menuitem": [
      { « v": "New", "onclick": "new()" },
      { « v": "Open", "onclick": "open()" },
      { « v": "Close", "onclick": "close()" }
    ]
    }
  }
}
```




RELAX

CAP theorem

Physical constraints on distributed systems

Distributed systems has two natural constraints
so they don’t behave like a single system !

Number of nodes

↗ Probability of the failure of the system

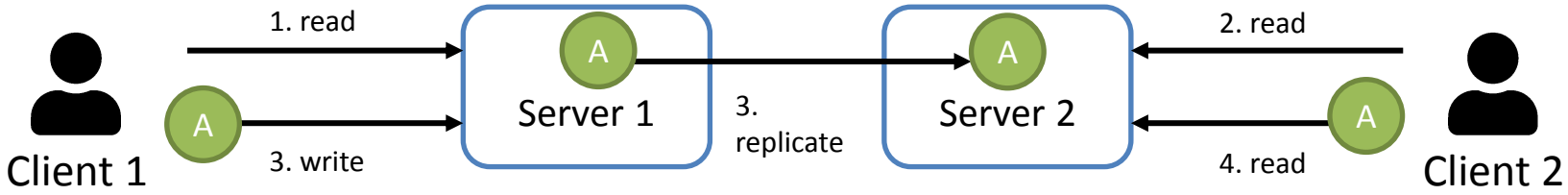
Need of communication between nodes

Distance between nodes

↗ Minimum latency for communication

Strong consistency

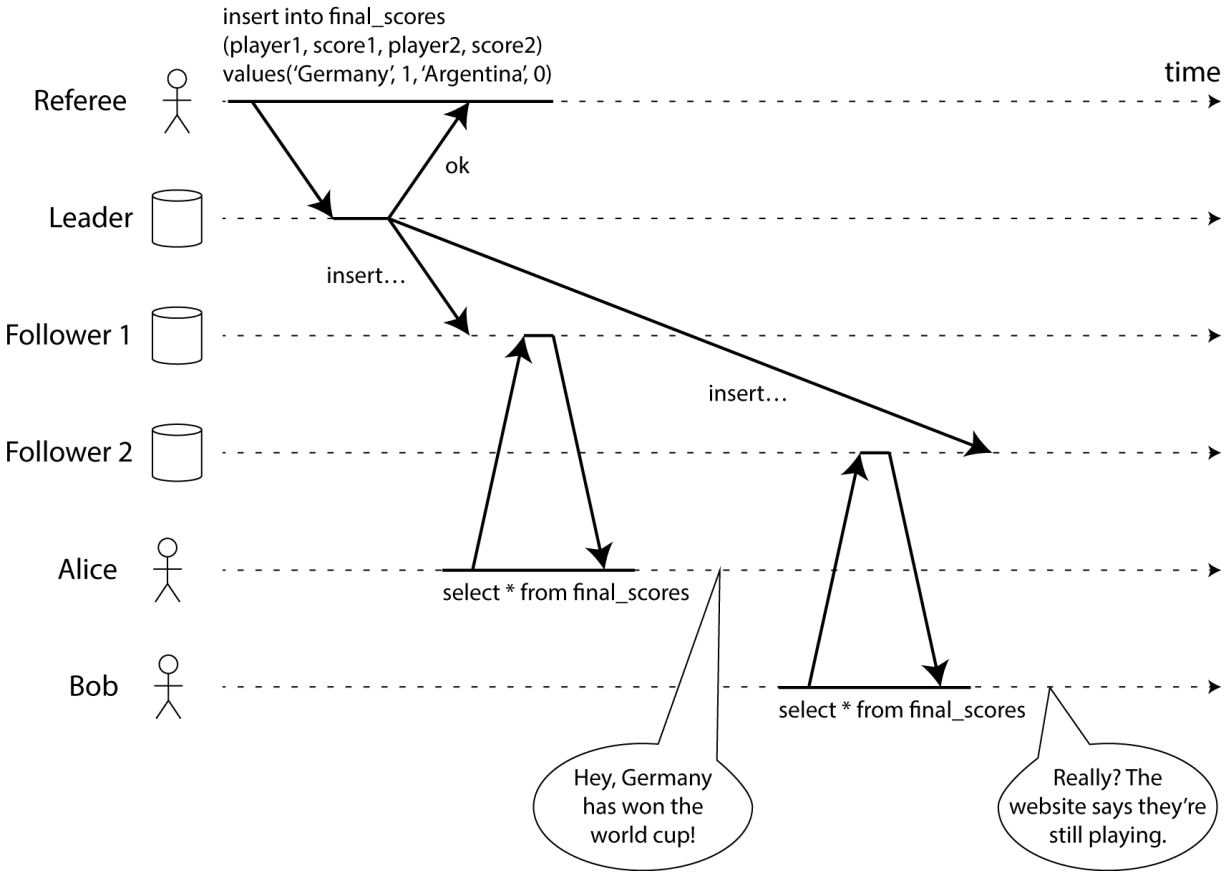
All nodes see the same data at the same time



- Linearizable consistency: all operations appear to have executed atomically in an order that is consistent with the global real-time ordering of operations
- Sequential consistency: all operations appear to have executed atomically in some order that is consistent with the order seen at individual nodes and that is equal at all nodes

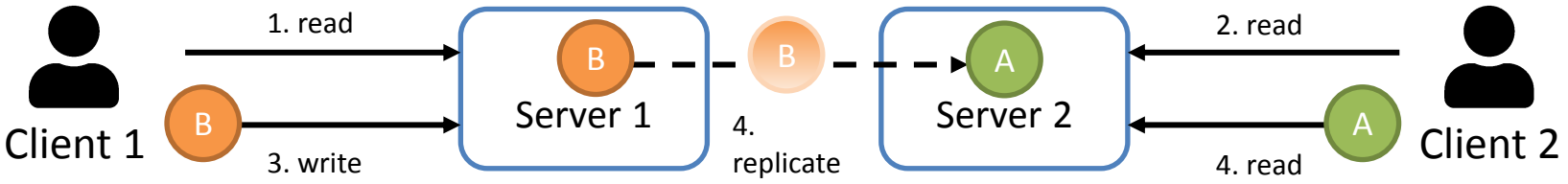
- ❑ Needs to block writing during the update
- ❑ Blocking time is unpredictable given network issues

Non-linearizable system



Weak consistency

If you stop changing values, then after some undefined amount of time all replicas will agree on the same value



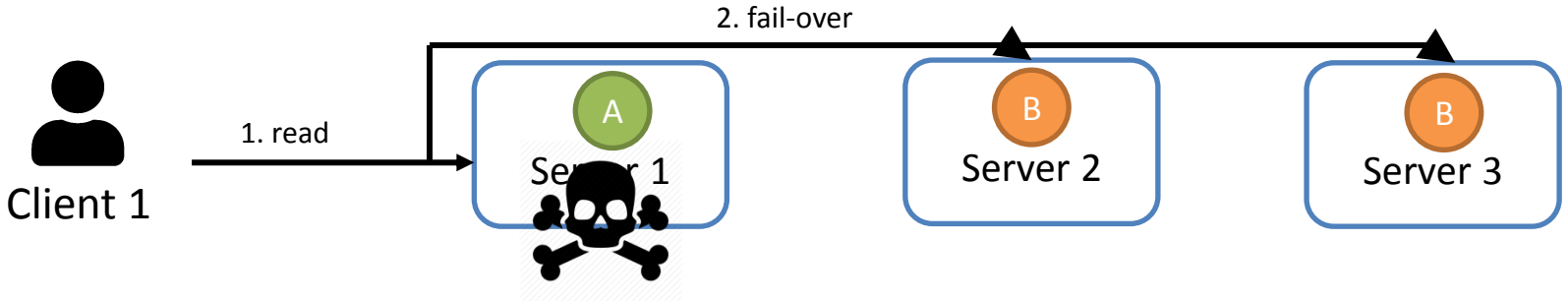
Saying something is eventually consistent is like saying “people are eventually dead” :

- how long is « eventually » ? (ex: Get a strict lower bound)
- how do the replicas agree on a value ? (ex: value with the largest timestamp always win)

Non-blocking writing : better performance, at the risk of reading stale values

Availability

Node failures do not prevent survivors from continuing to operate



Use replicas:

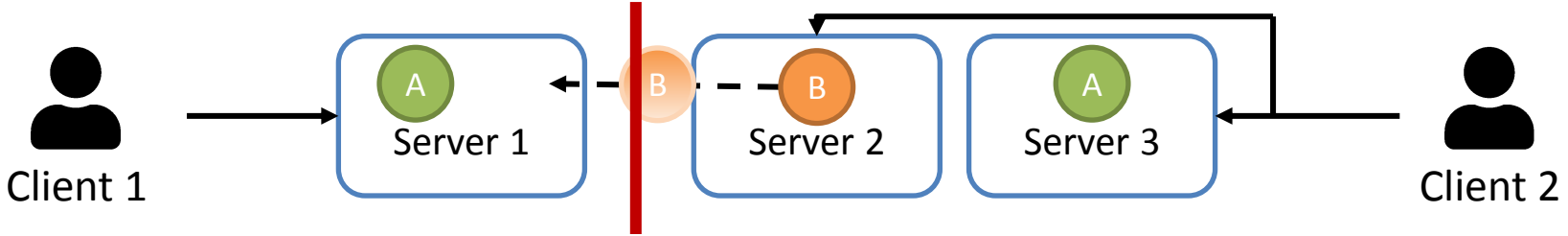
- For reading for better performance (nearest copy)
- For writing, if machine where we should write is dead (conflict resolution)

Failure tolerance

- Heartbeating / Gossip protocols to identify dead nodes
- Fail-over : when an original is dead, a replica is promoted to original, update all machines/clients to point to new original

Partition tolerance

The system continues to operate despite message loss due to network and/or node failure

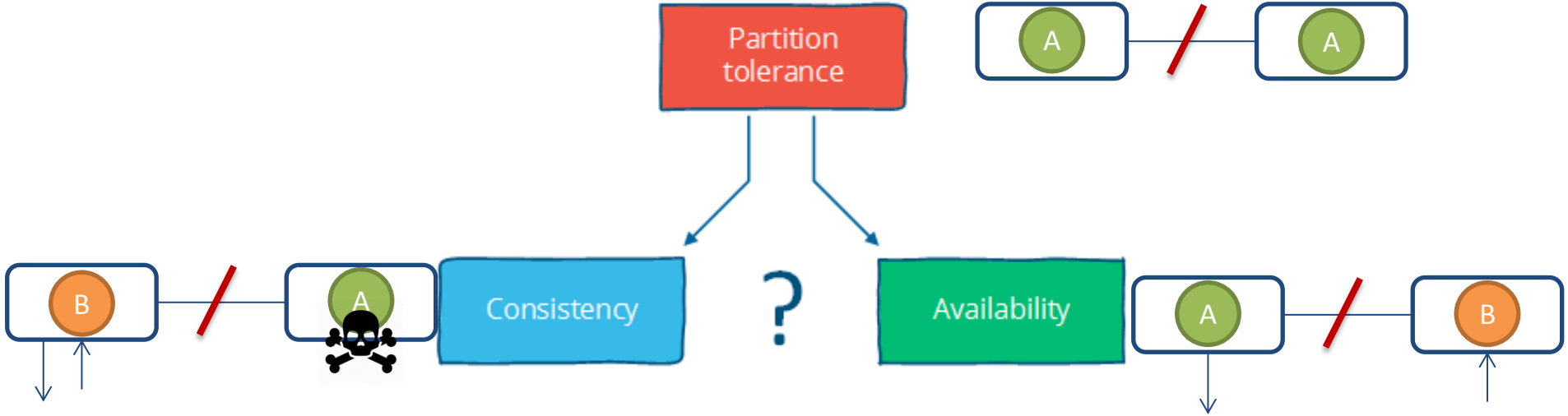


Divided machines must work independently

- Can still read and write data
- When network is back, should resolve data conflicts between different machines

- Many system designs used in early distributed relational database systems do not take into account partition tolerance

CAP theorem

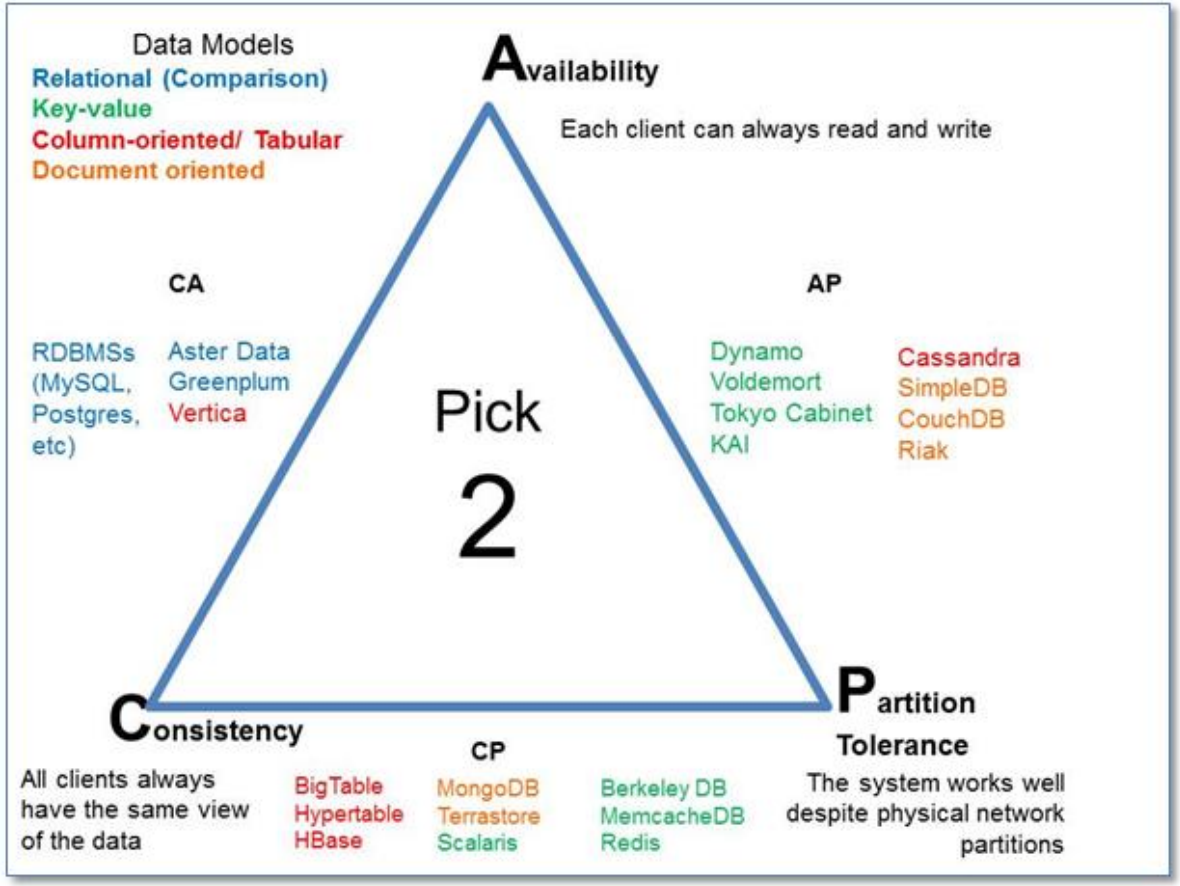


On a network failure

- Choose strong consistency → Refuse the write, knowing that the client might not be able to contact A or B until the partition heals.
- Choose availability → Accept the write, knowing that neither A nor B will know about this new data until the partition heals.

The CAP theorem is an example of distributed system ≠ single server

CAP theorem

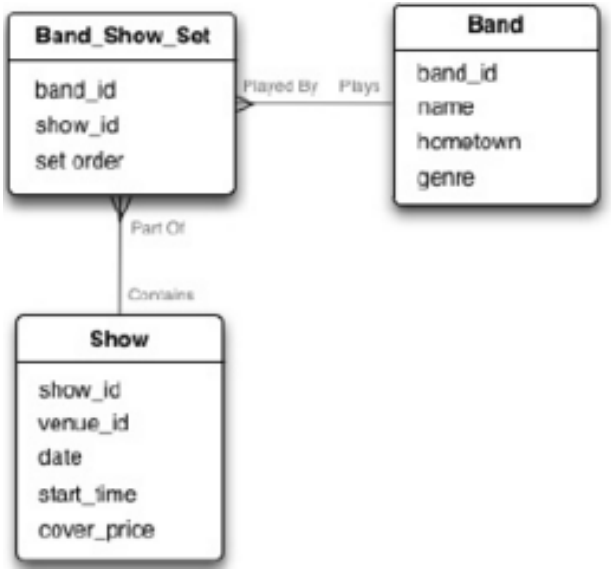

Data Models
Relational (Comparison)
Key-value
Column-oriented/ Tabular
Document oriented
Availability
Each client can always read and write
CA
RDBMSs (MySQL, Postgres, etc)
Aster Data
Greenplum
Vertica
Pick 2
AP
Dynamo
Voldemort
Tokyo Cabinet
KAI
Cassandra
SimpleDB
CouchDB
Riak
Consistency
All clients always have the same view of the data
CP
BigTable
Hypertable
HBase
MongoDB
Terrastore
Scalaris
Berkeley DB
MemcacheDB
Redis
Partition Tolerance
The system works well despite physical network partitions


RELAX

Denormalization

Denormalization

- We are used to representing relationships between entities in the relational model in a normalized way
  - In the DB we then perform joins on tables as optimally as we can

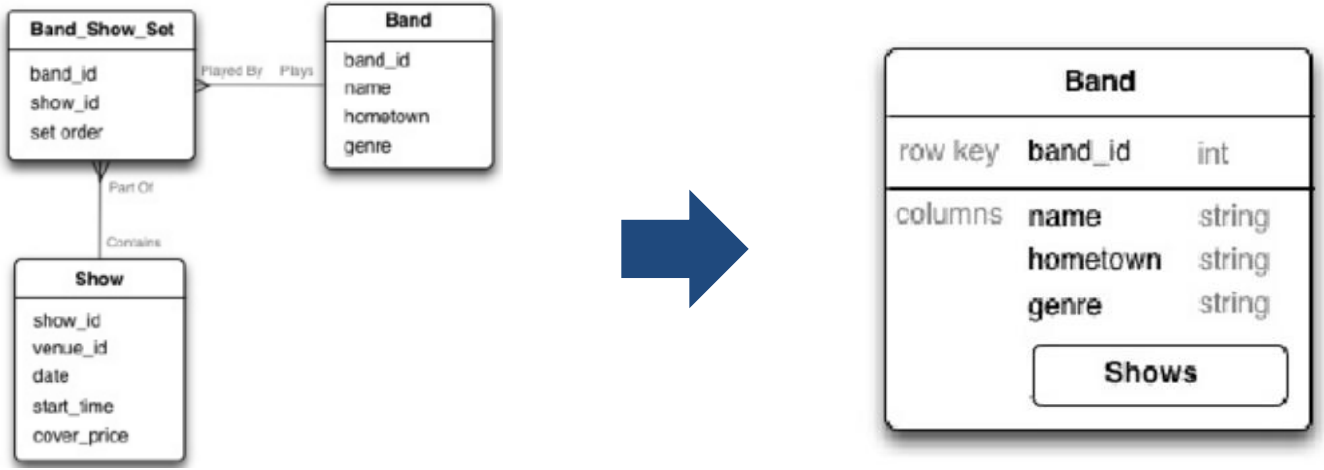


- But…the relational model doesn't acknowledge scale and we need to cheat
  - Add indexes, add hints, write complex SQL, trickle with the optimizer, cache…

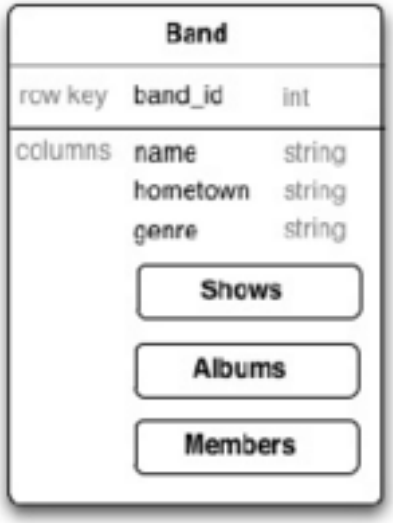
Denormalization

- NoSQL is a different way of physically representing your logical data model
  - \+ Scalable, efficient
  - \- “No” foreign keys, “nor” joins, “nor” cross-table transactions

- Denormalization : *logical* entities share one *physical* representation
  - – You can have one of your fields representing another entity → nested entity !

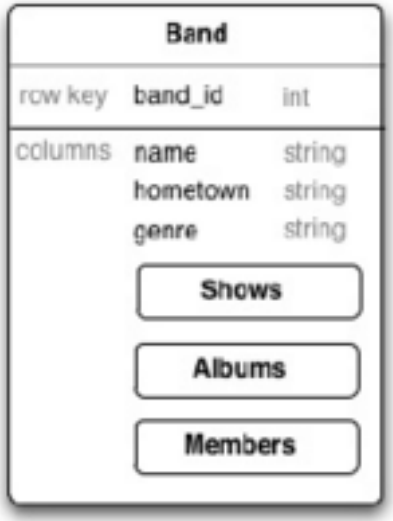


Denormalization



What do you think about querying this structure ?

Denormalization



- You then get all shows per band for free and fast
- You can put more nested entities
- You can nest more entities into the nested entity
- In the end, you create one denormalized table per business query you need

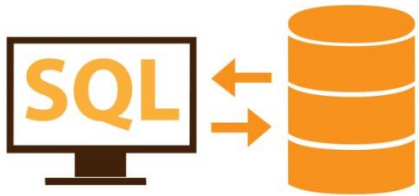
---

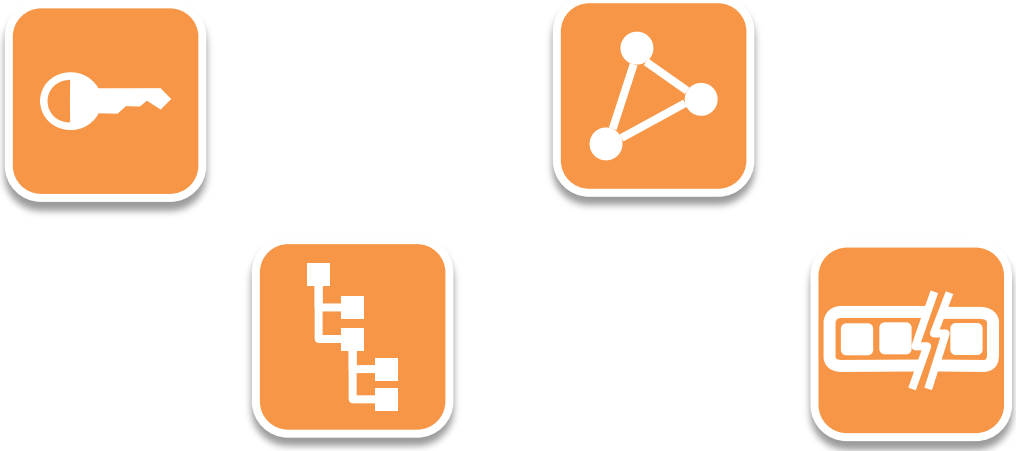
- Nested entites aren't independent anymore
  - "SELECT avg(price) FROM albums WHERE name = "bla"
- Decomposition is unilateral
- Makes you think more about physical access than relational model

CONCLUSION

Conclusion

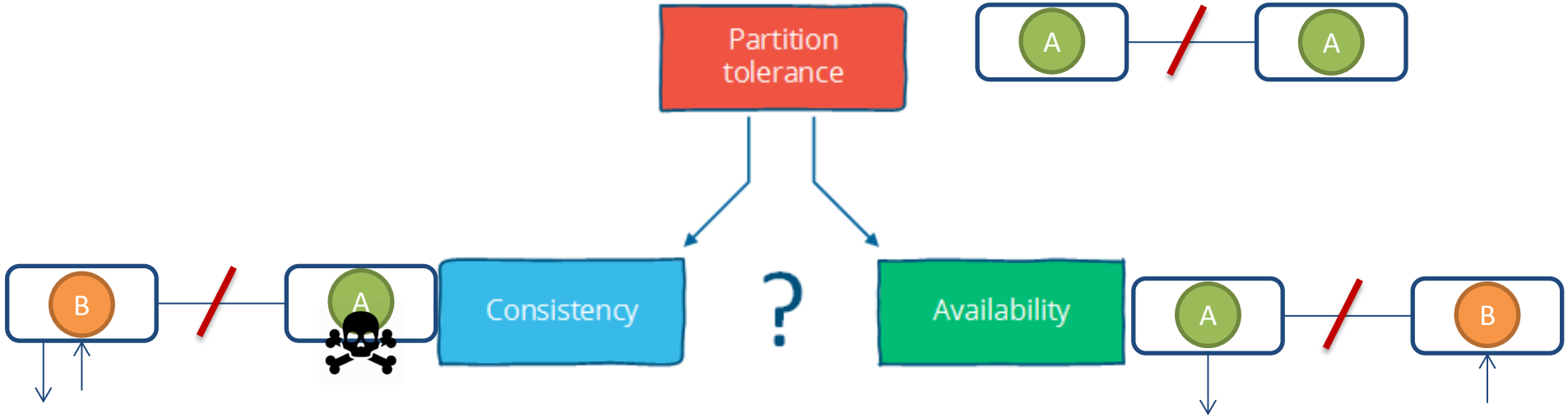
- A **NoSQL** (originally referring to "non SQL" or "non relational") database provides a mechanism for storage and retrieval of data which is modeled in means other than the tabular relations used in relational databases
- Simplicity of design, simpler “horizontal” scaling, finer control over availability, more flexible data structures





Conclusion

- Offers compromise consistency in favor of availability, partition tolerance and speed
- Concept of eventual consistency where databases changes are propagated to all nodes “eventually”, queries for data might not return updated data immediately or result in reading data that is not accurate



THANKS

@andfanilo

@andfanilo

andfanilo@gmail.com