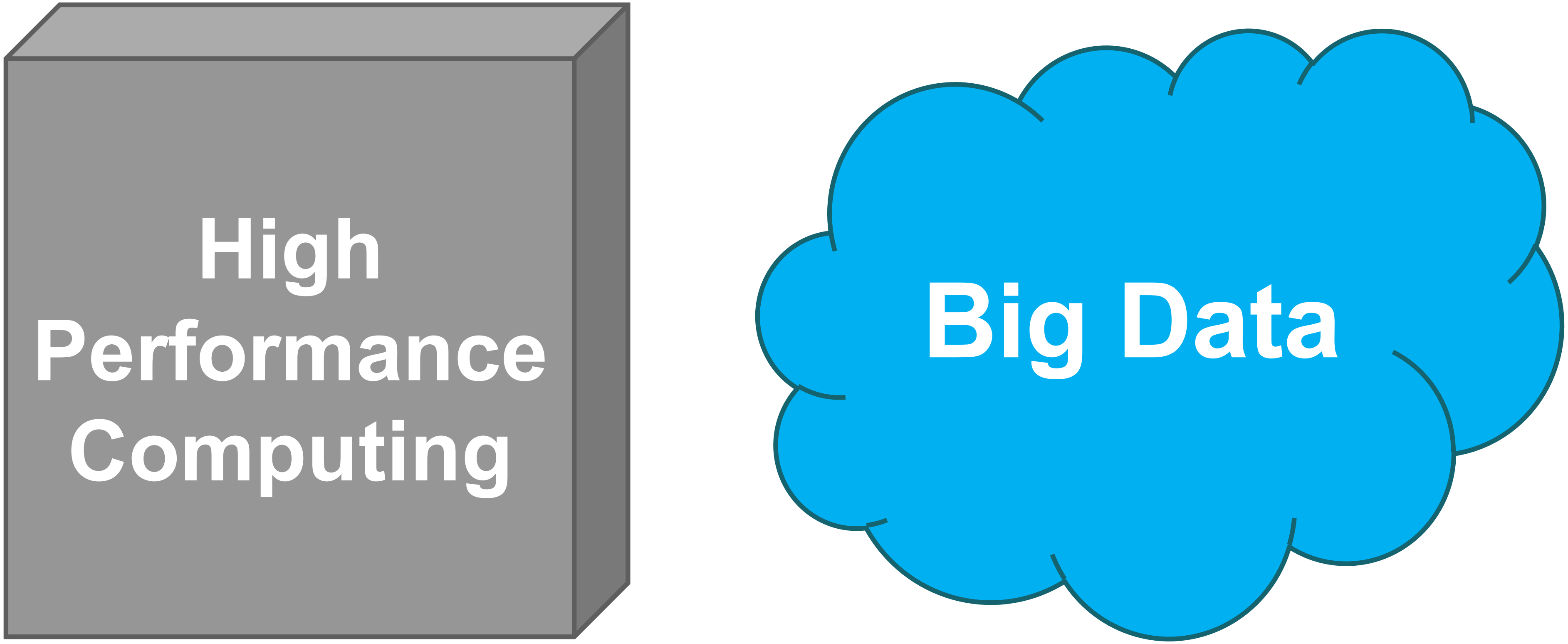
INSTITUT NATIONAL
DES SCIENCES
APPLIQUÉES
RENNES

Data and Processing Models for Big Data

Alexandru Costan

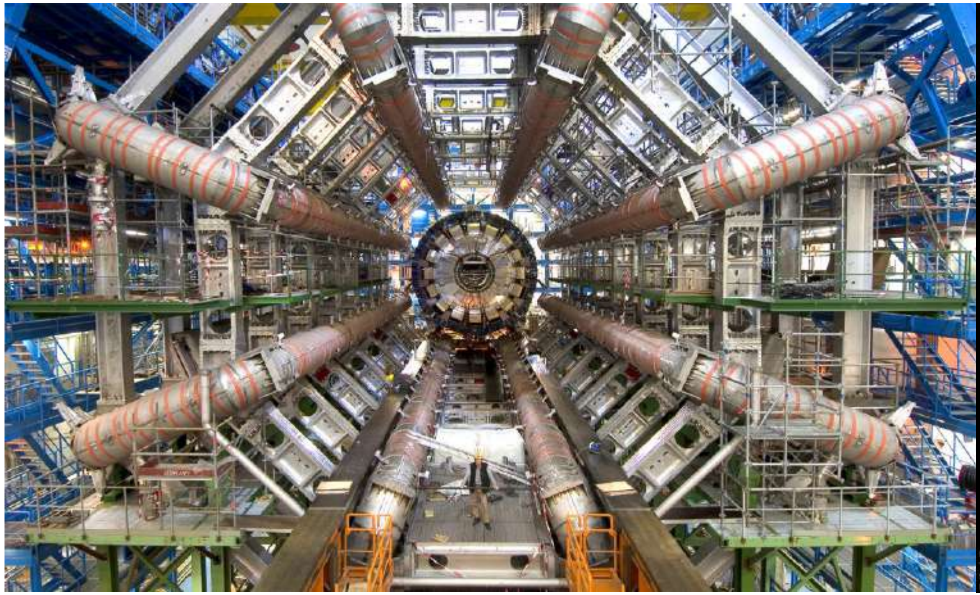
Two worlds

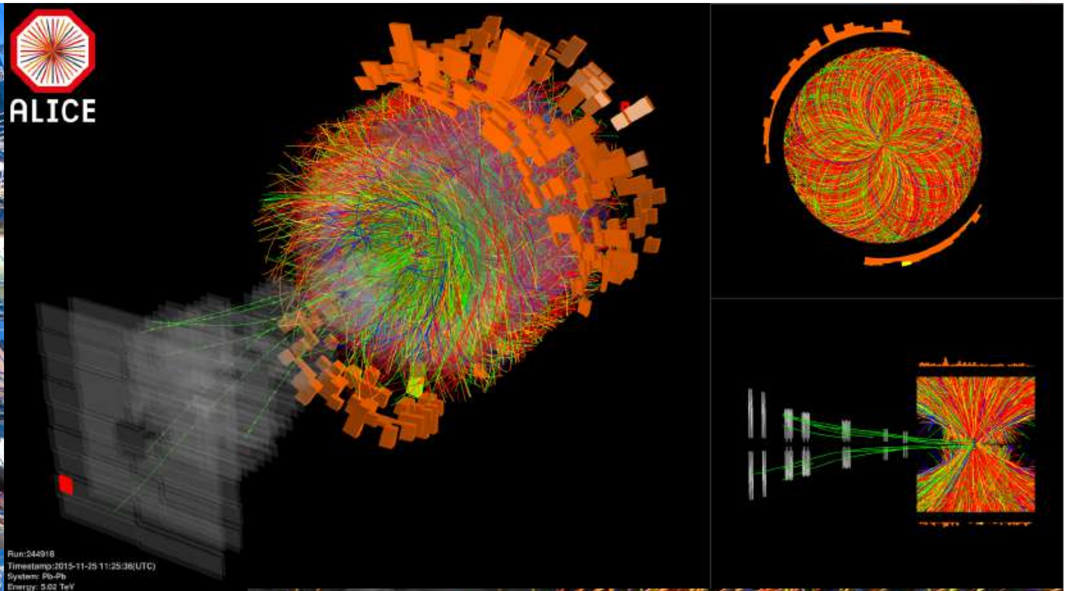


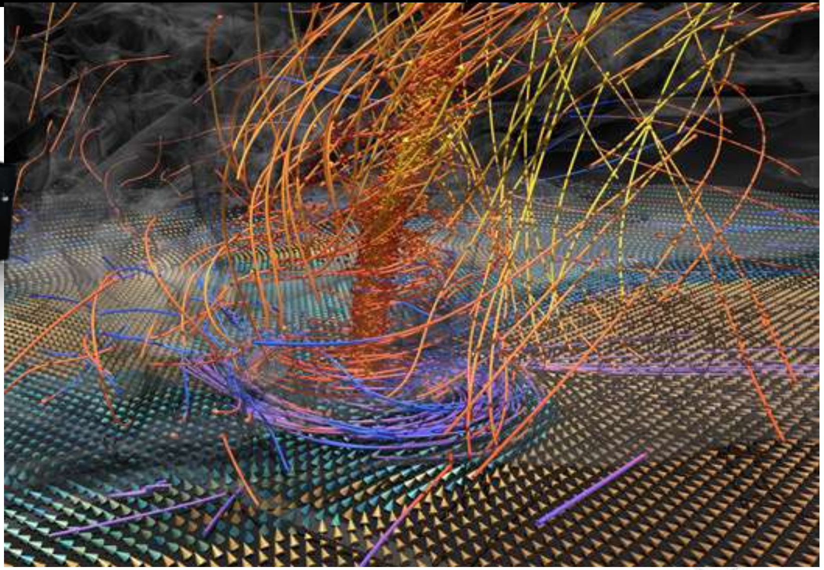
High Performance Computing

Big Data

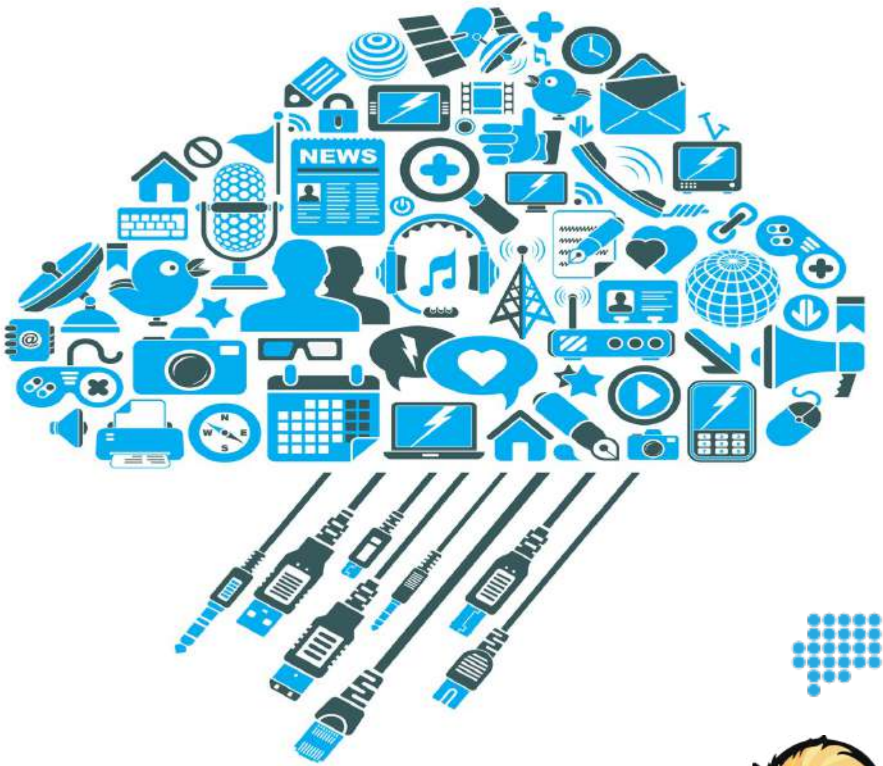
HPC: Simulations and experiments on supercomputers



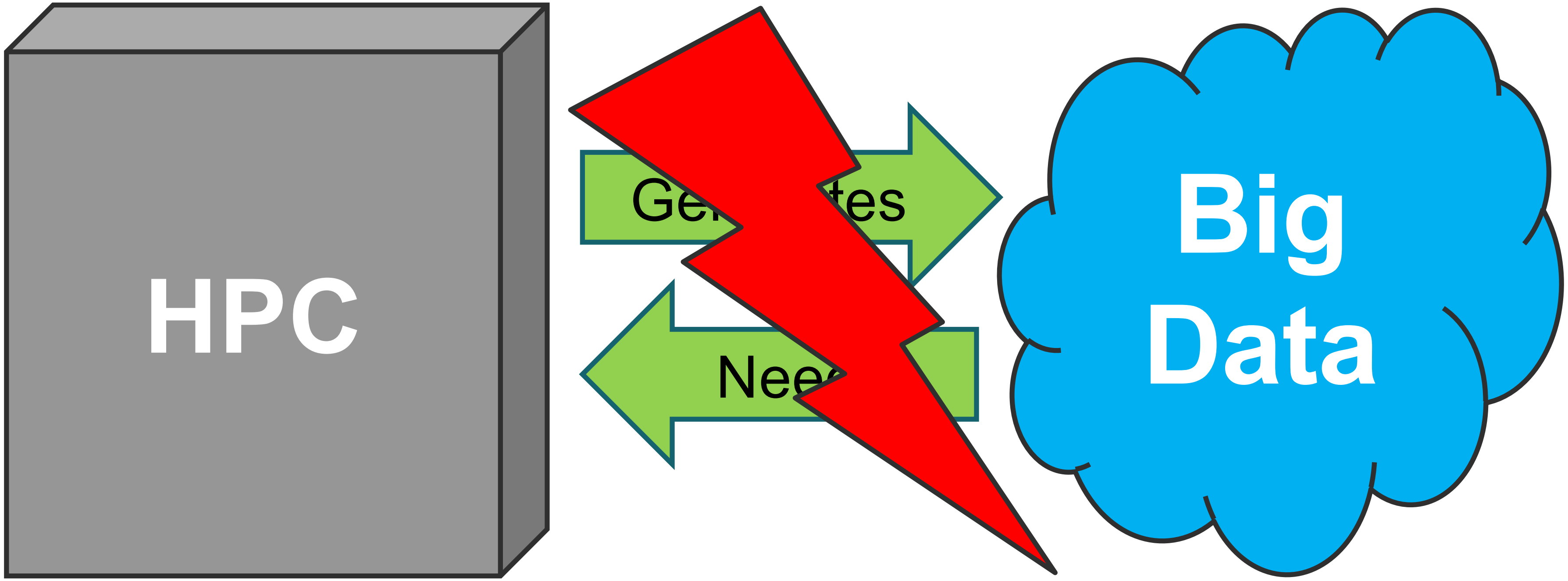




Big Data: Commercial and scientific analytics on clouds

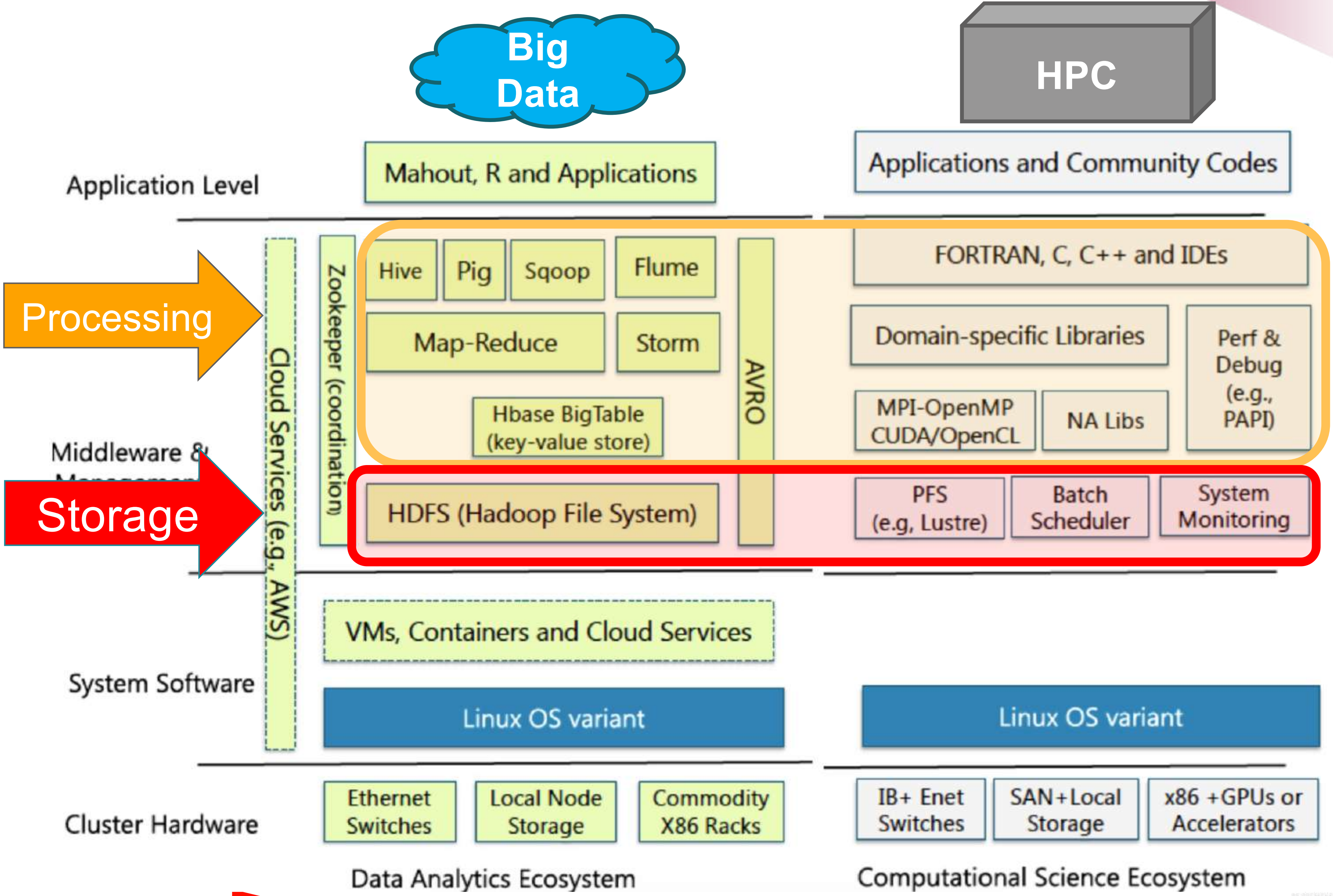


Big Data and HPC



Yet, their tools and cultures diverged…
... to the detriment of both!

Divergent ecosystems



Big Data

HPC

Processing

Storage

Application Level

Middleware &

System Software

Cluster Hardware

Mahout, R and Applications

Cloud Services (e.g., AWS)

Zookeeper (coordination)

Hive

Pig

Sqoop

Flume

Map-Reduce

Storm

Hbase BigTable (key-value store)

AVRO

HDFS (Hadoop File System)

VMs, Containers and Cloud Services

Linux OS variant

Ethernet Switches

Local Node Storage

Commodity X86 Racks

Data Analytics Ecosystem

Applications and Community Codes

FORTRAN, C, C++ and IDEs

Domain-specific Libraries

Perf & Debug (e.g., PAPI)

MPI-OpenMP CUDA/OpenCL

NA Libs

PFS (e.g, Lustre)

Batch Scheduler

System Monitoring

Linux OS variant

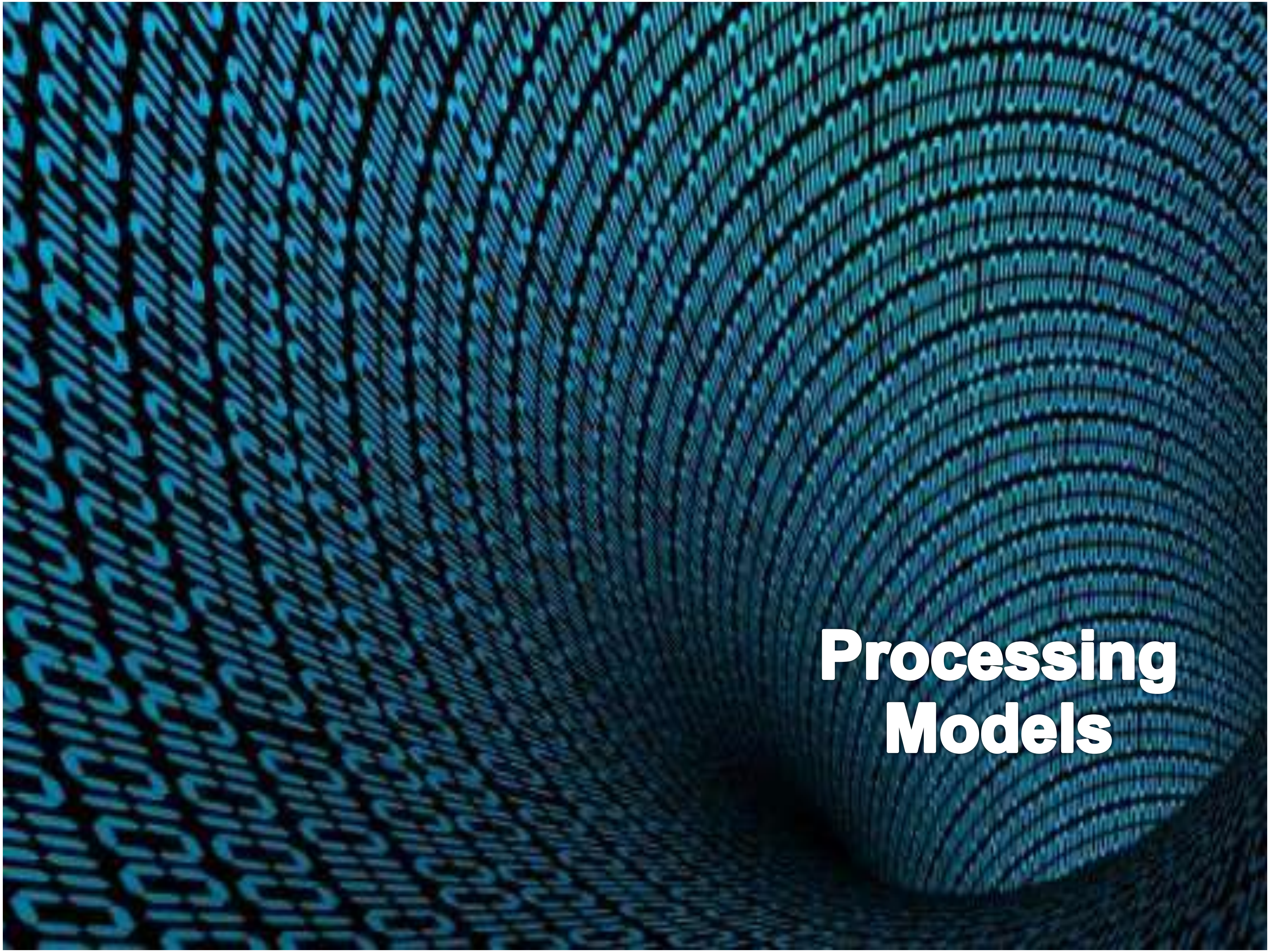
IB+ Enet Switches

SAN+Local Storage

x86 +GPUs or Accelerators

Computational Science Ecosystem

Credits: Dan Reed

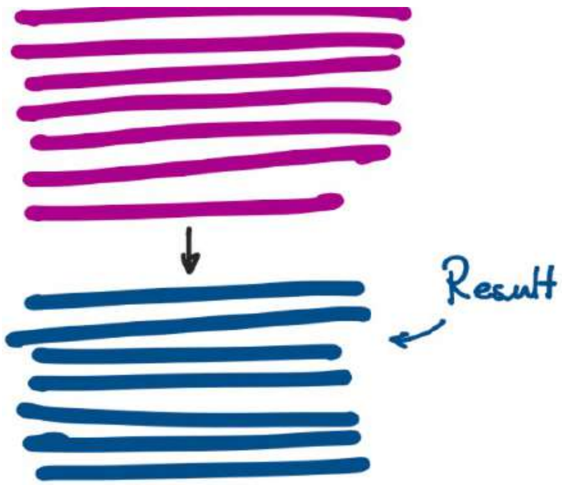


Processing Models

Two ways of processing Big Data

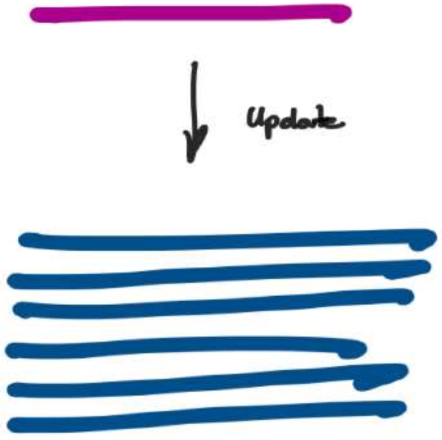
Batch processing

- collecting a series of data
- storing it until a given quantity of data has been collected
- then processing all of that data as a group – in other words, as a batch

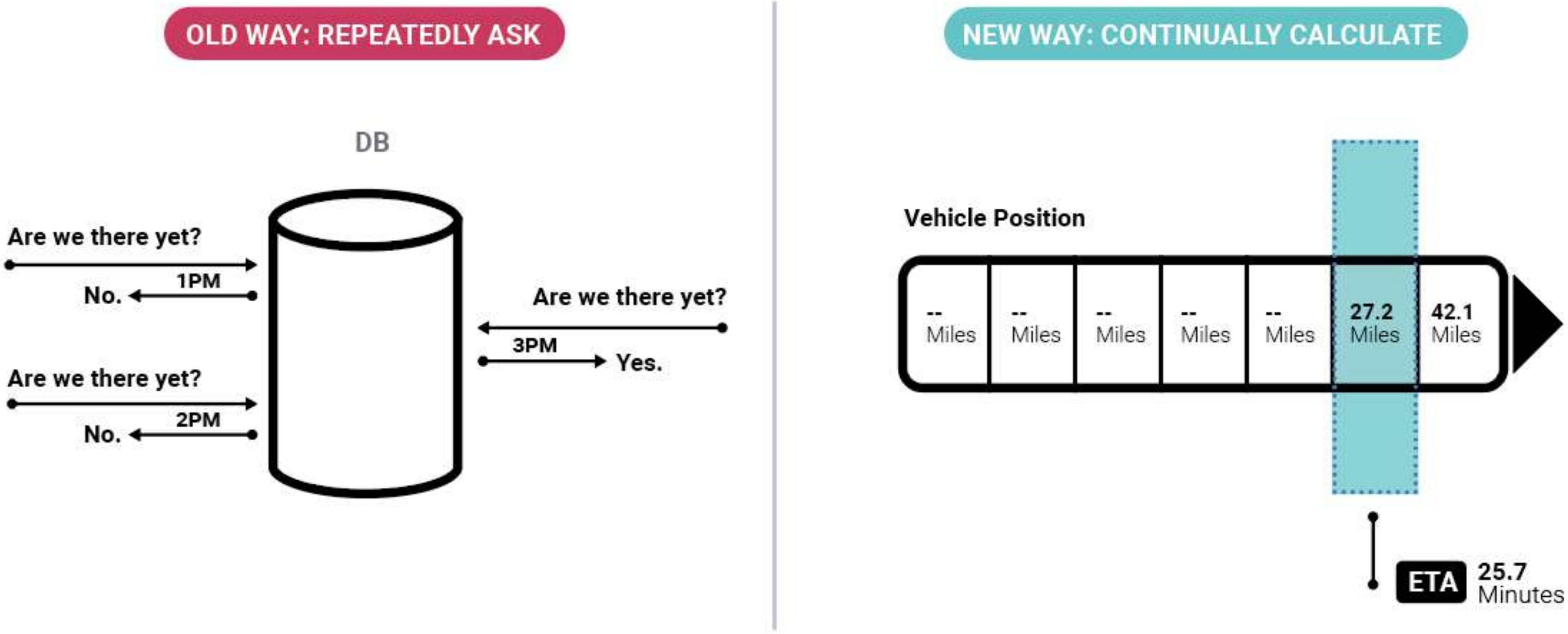


Real-time (stream) processing

- each piece of data is processed as soon as it is collected
- results available virtually instantaneously



Batch vs. real-time

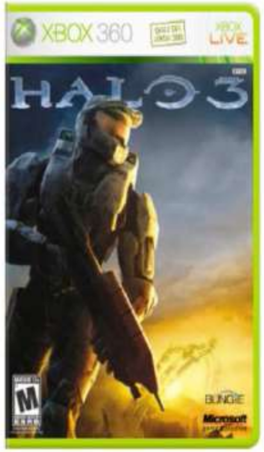


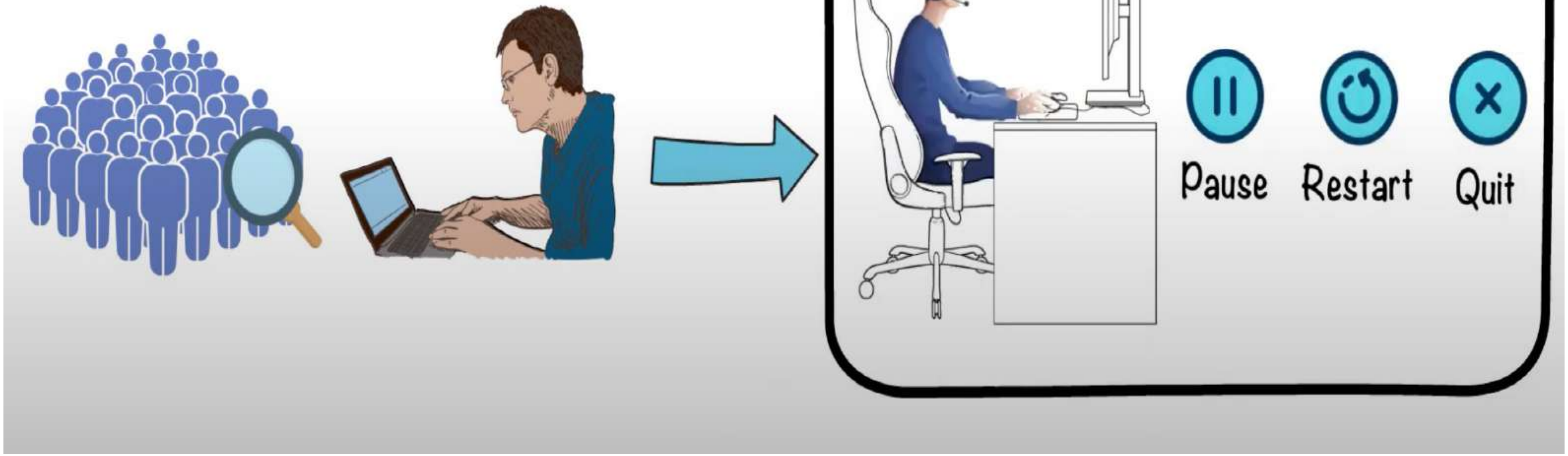
Credits: Jay Kreps

Batch vs. real-time

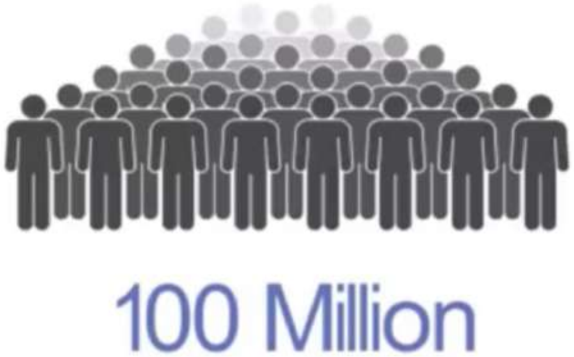
Which is better for which use-cases?

Understanding user behavior

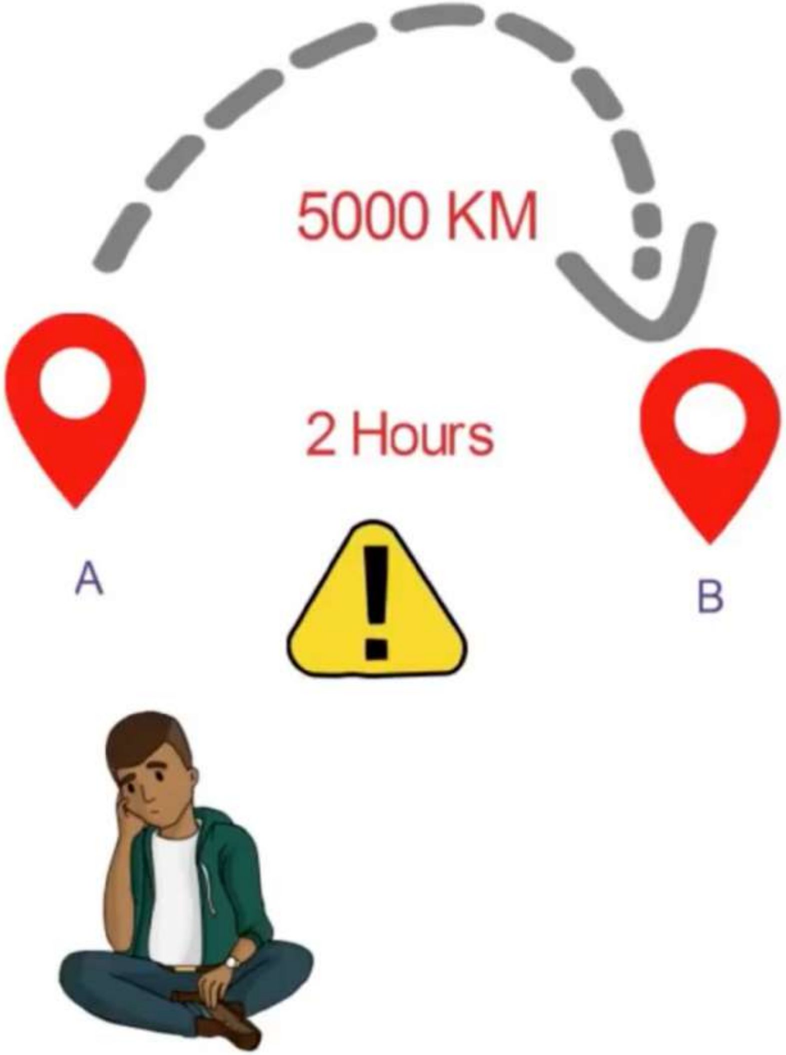




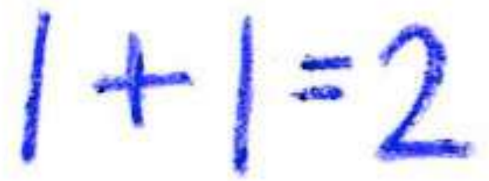
Recommendations



Fraud detection



Batch vs. real-time



Correctness

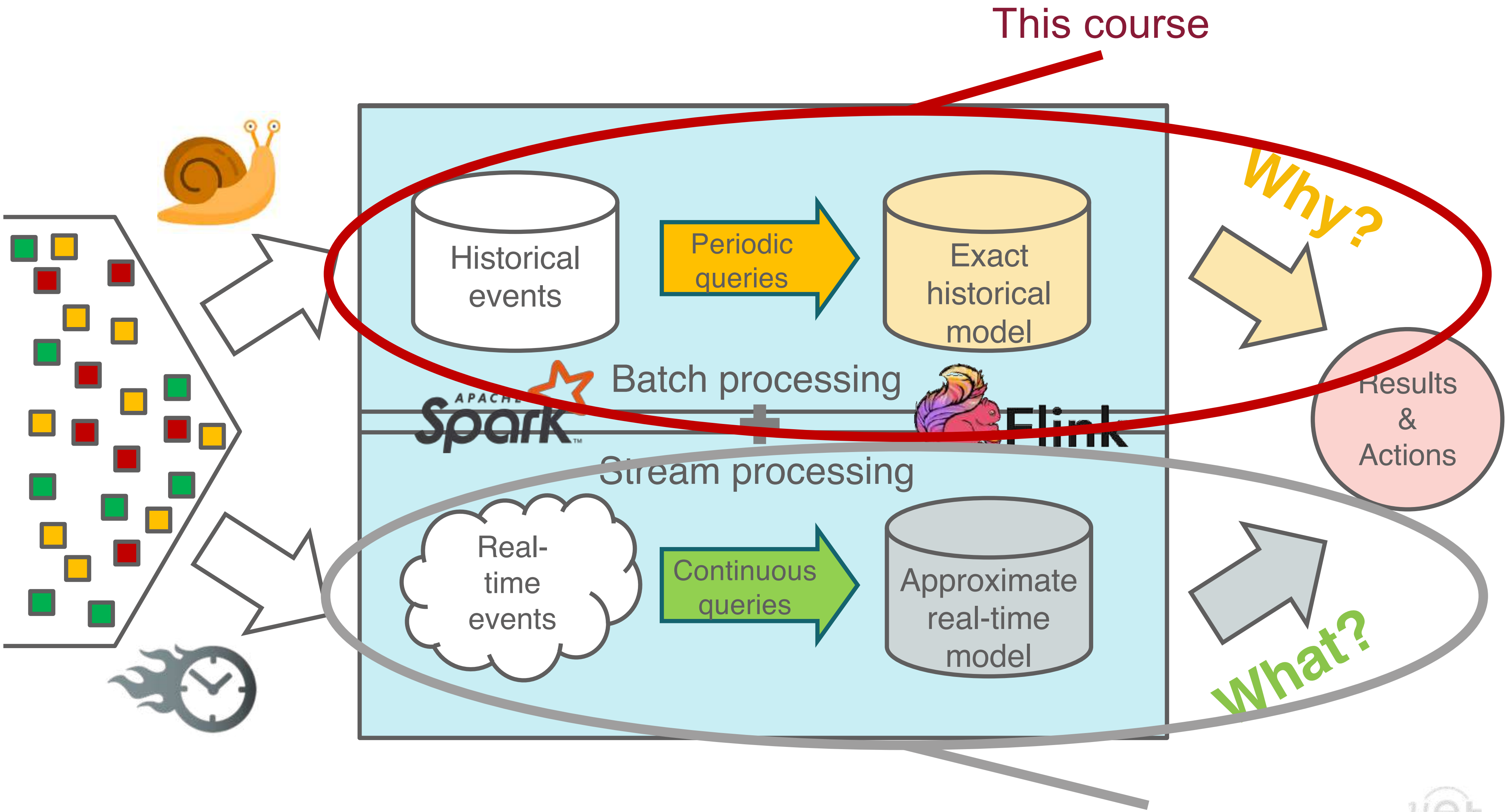
Latency

Cost

Batch vs. real-time

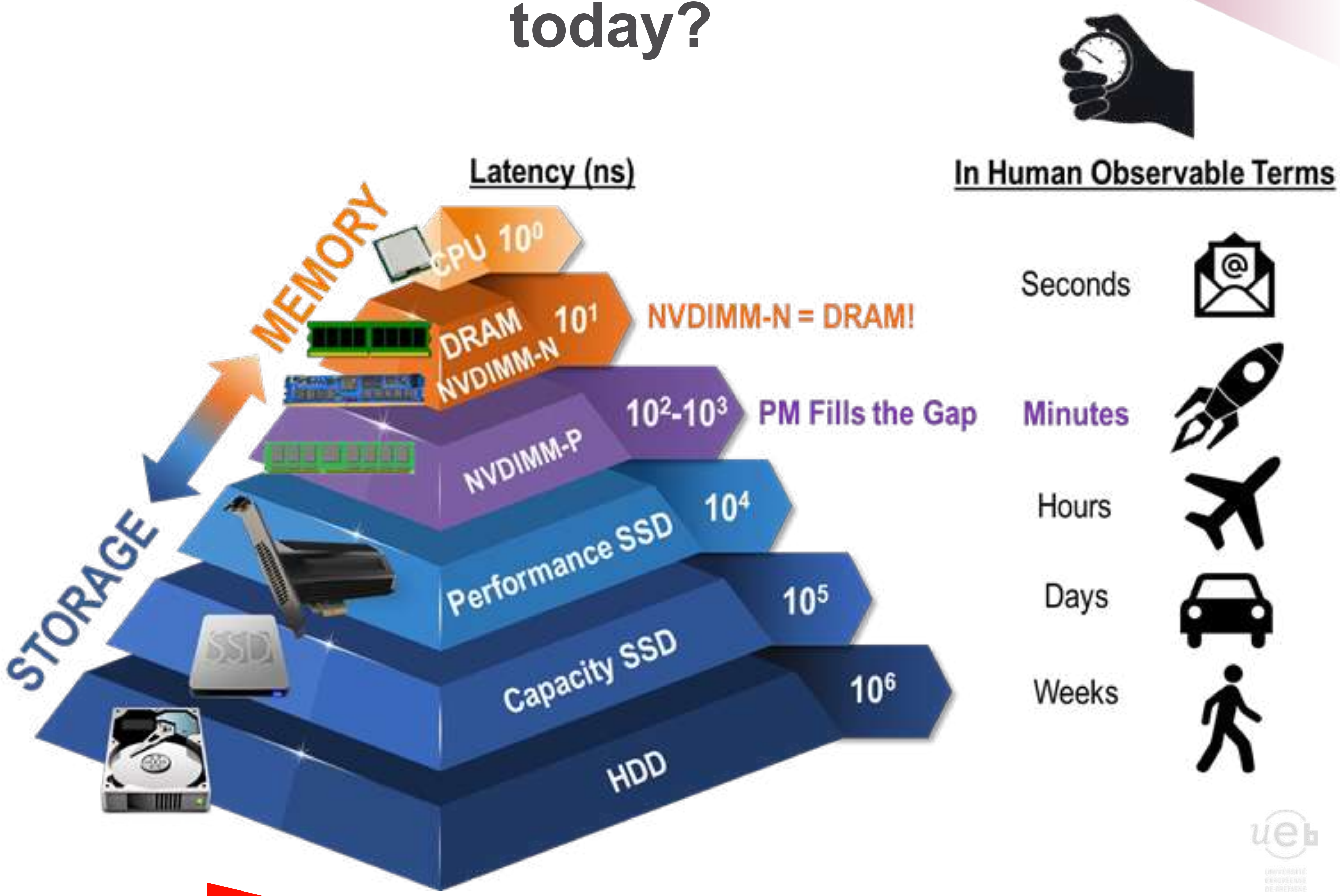
Correctness	Exact results	Approximate results
Latency	High-latency	Low-latency
Cost	Stateless	Stateful

State of the art: Lambda Architectures



Data Models

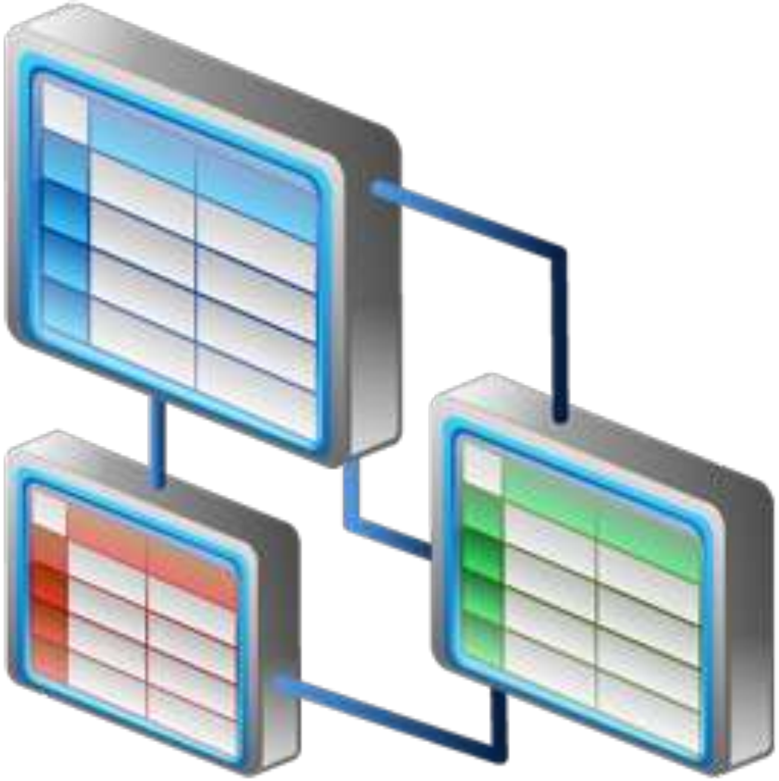
How do we **store** data today?



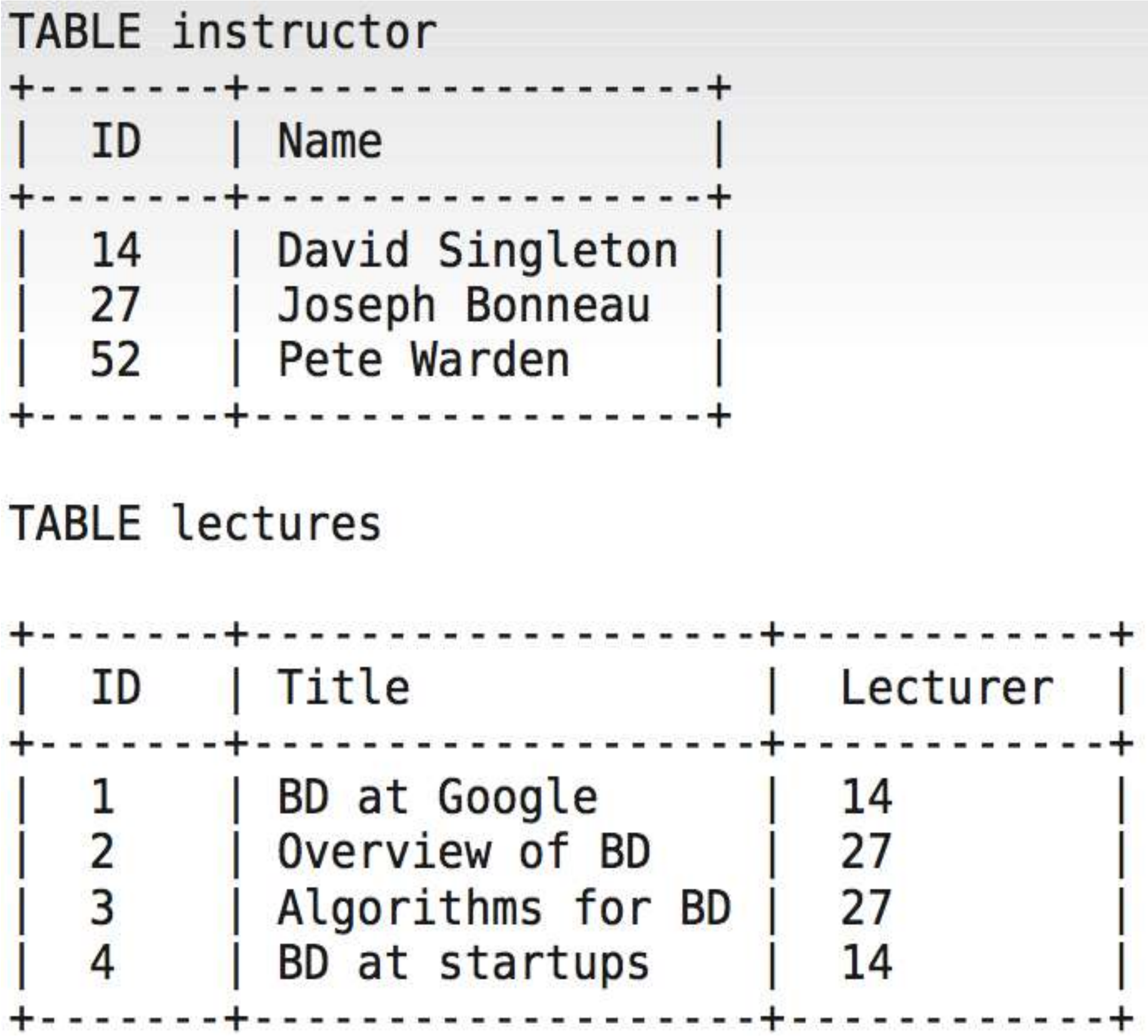
How do we **store** data today?

Relational Databases (RDBMS)

- Historically, the *de-facto* standard
- Optionally equipped with some caches
- Good for small and medium size data

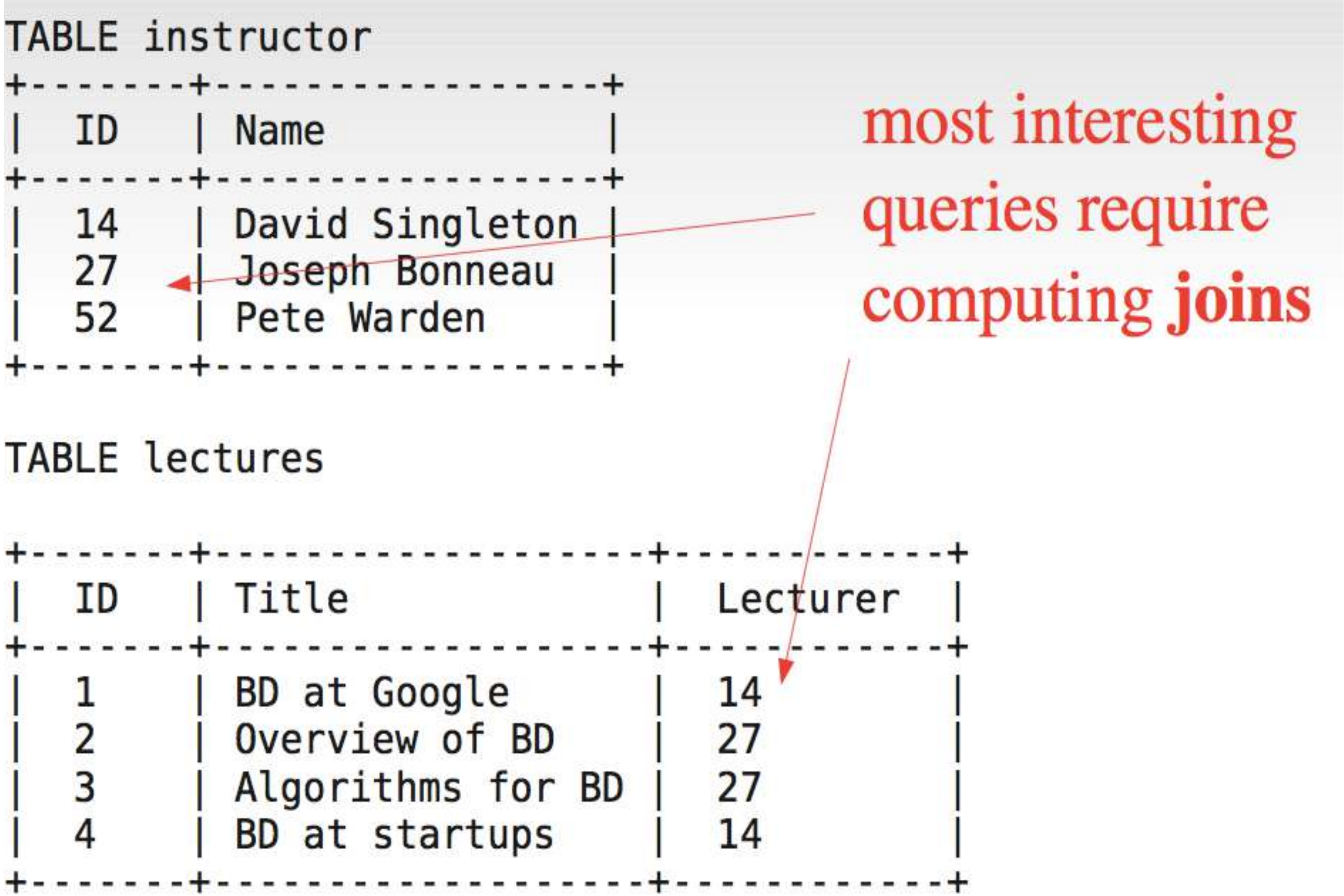


Traditional SQL databases



```
TABLE instructor
+-------+----------------+
|  ID   | Name           |
+-------+----------------+
|  14   | David Singleton |
|  27   | Joseph Bonneau  |
|  52   | Pete Warden     |
+-------+----------------+

TABLE lectures

+-------+-------------------+-----------+
|  ID   | Title             |  Lecturer |
+-------+-------------------+-----------+
|  1    | BD at Google      |  14       |
|  2    | Overview of BD    |  27       |
|  3    | Algorithms for BD |  27       |
|  4    | BD at startups    |  14       |
+-------+-------------------+-----------+
```

Traditional SQL databases

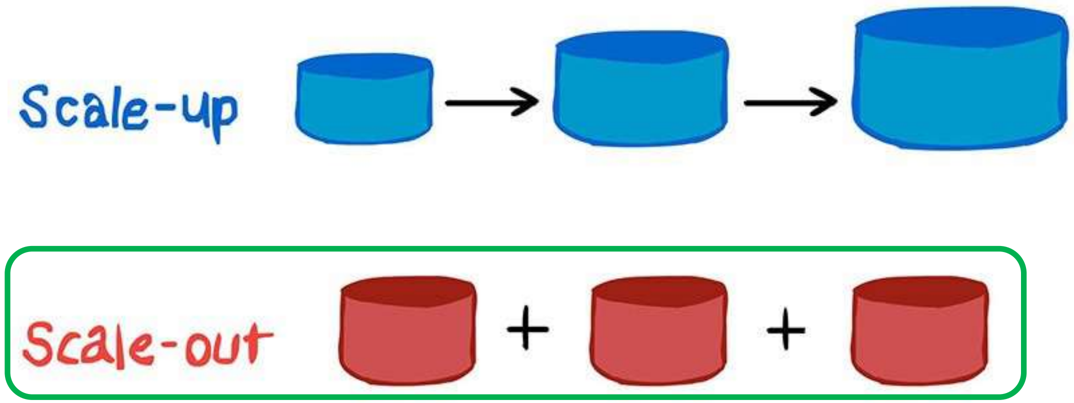


```
TABLE instructor
+-------+-----------------+
|  ID   | Name            |
+-------+-----------------+
|  14   | David Singleton |
|  27   | Joseph Bonneau  |
|  52   | Pete Warden     |
+-------+-----------------+

TABLE lectures

+-------+-------------------+------------+
|  ID   | Title             |  Lecturer  |
+-------+-------------------+------------+
|  1    | BD at Google      |  14        |
|  2    | Overview of BD    |  27        |
|  3    | Algorithms for BD |  27        |
|  4    | BD at startups    |  14        |
+-------+-------------------+------------+
```

most interesting queries require computing **joins**

Issues when the dataset is just too big

Scaling out

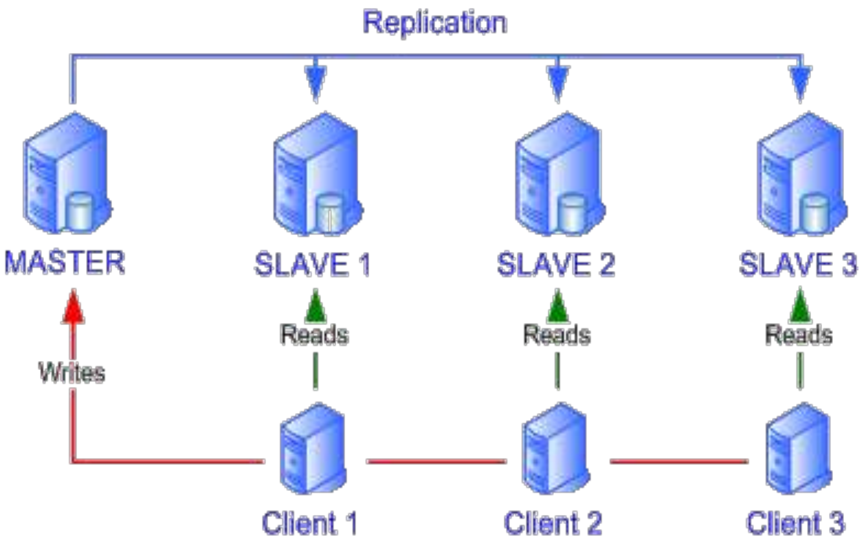


- Began to look at multi-node database solutions
  - Known as ‘scaling out’ or ‘horizontal scaling’
  - RDBMS were not designed to be distributed
- Different approaches include
  - **Master-slave**
  - **Sharding**

Scaling RDBMS: Master/Slave

Data **replicated** on slaves

- All *writes* are written to the *master*
- All *reads* from the replicated *slaves*



Advantage

- Good load balance for reads

Problems

- Critical reads may be incorrect as writes may not have been propagated down
- Large datasets are duplicated: huge storage

Scaling RDBMS: Sharding

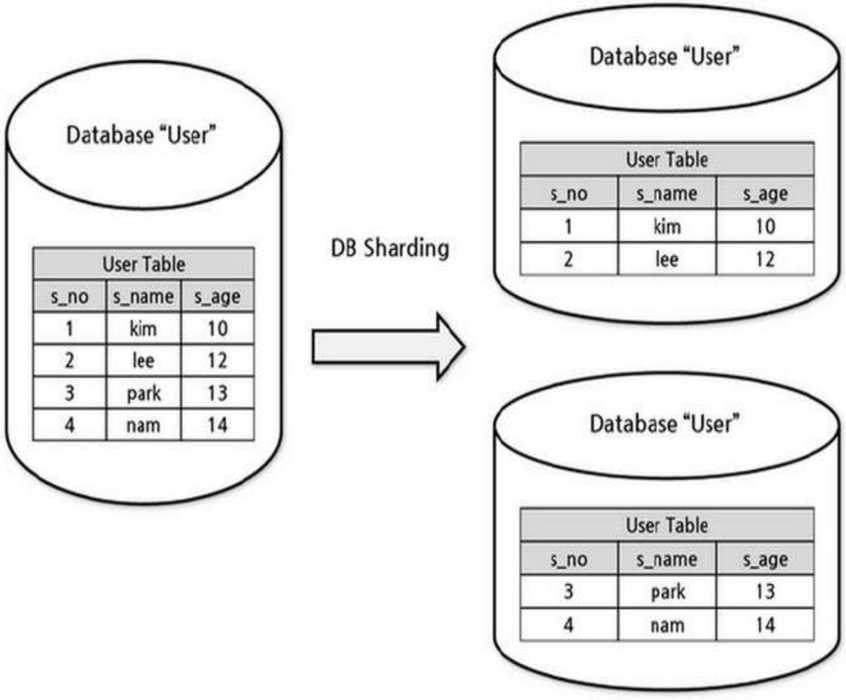
Data **partitioned** to slaves

Advantage

- Scales well for both reads and writes

Problems

- Not transparent, application needs to be partition-aware
- Can no longer have relationships/joins across partitions
- Loss of referential integrity across shards



User Table

s_no	s_name	s_age
1	kim	10
2	lee	12
3	park	13
4	nam	14

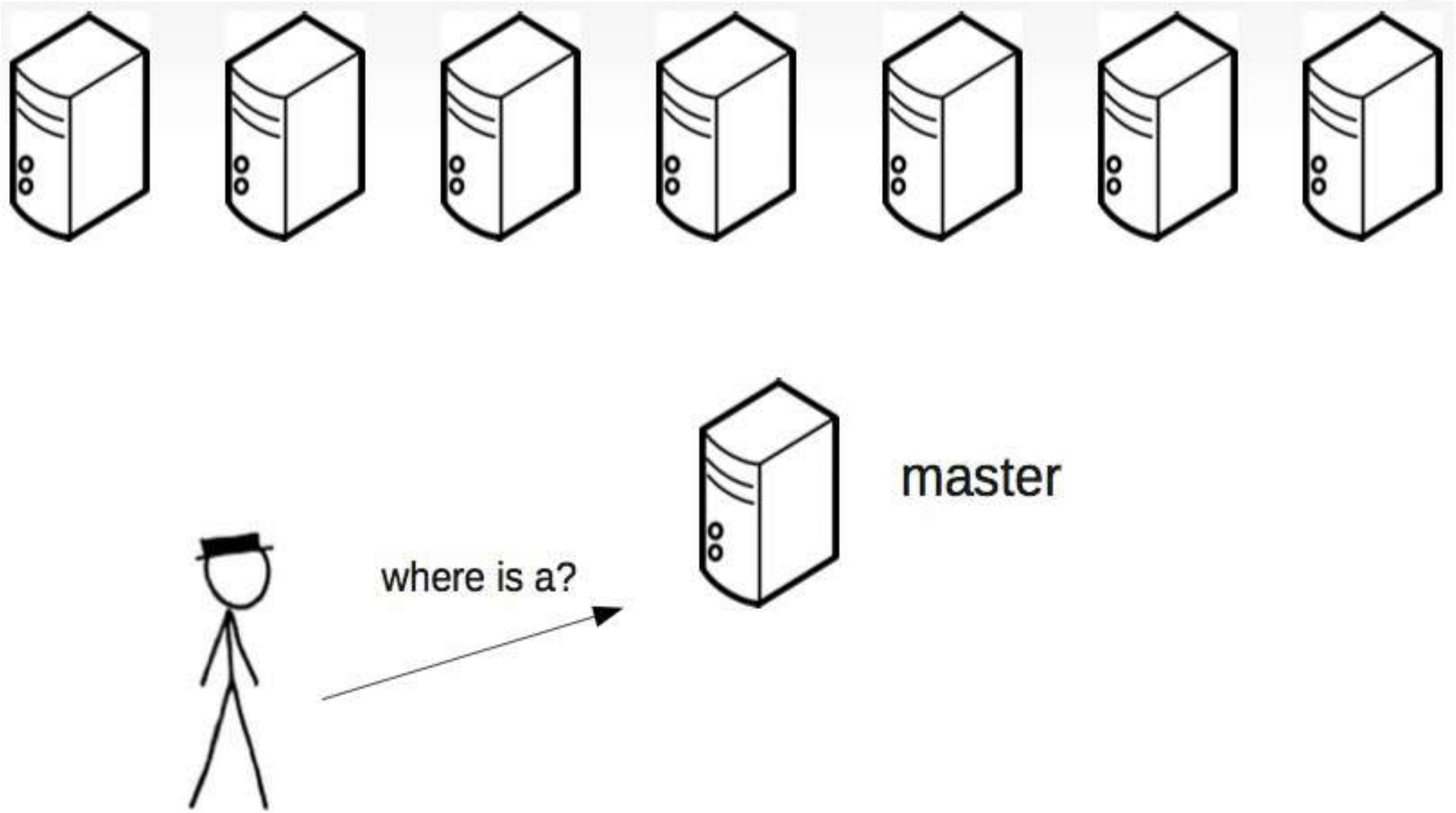
User Table

s_no	s_name	s_age
1	kim	10
2	lee	12

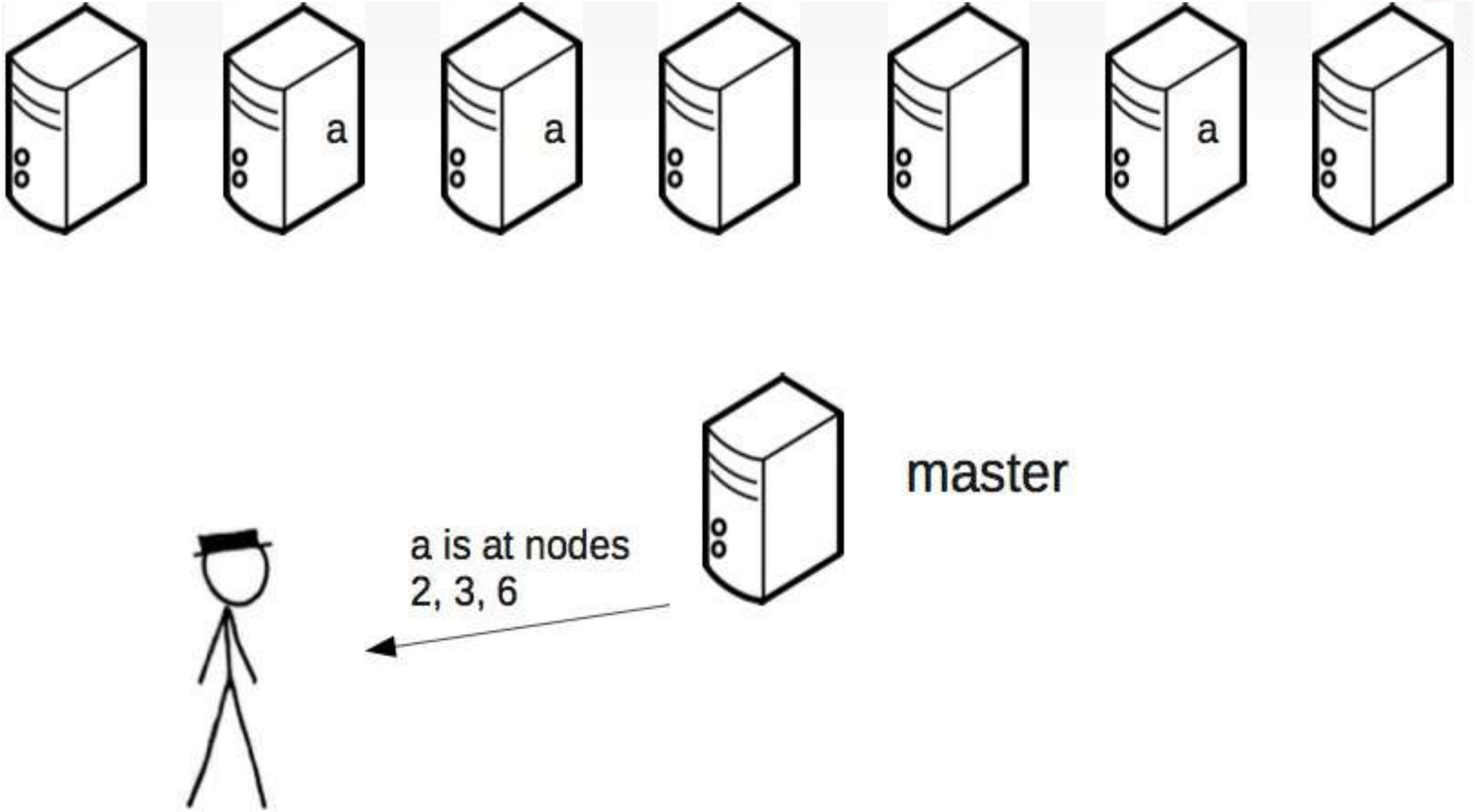
User Table

s_no	s_name	s_age
3	park	13
4	nam	14

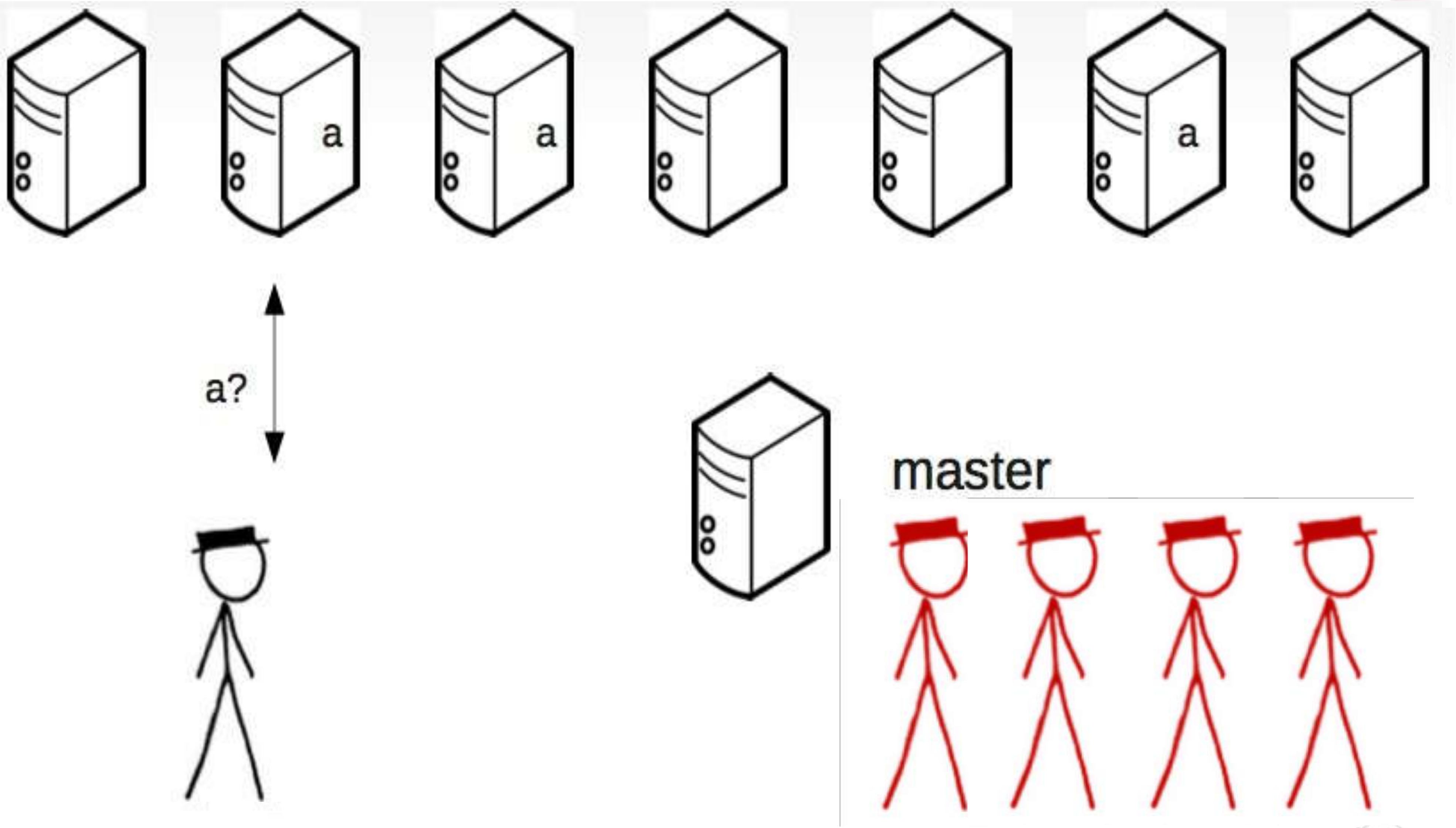
How sharding works



How sharding works



How sharding works

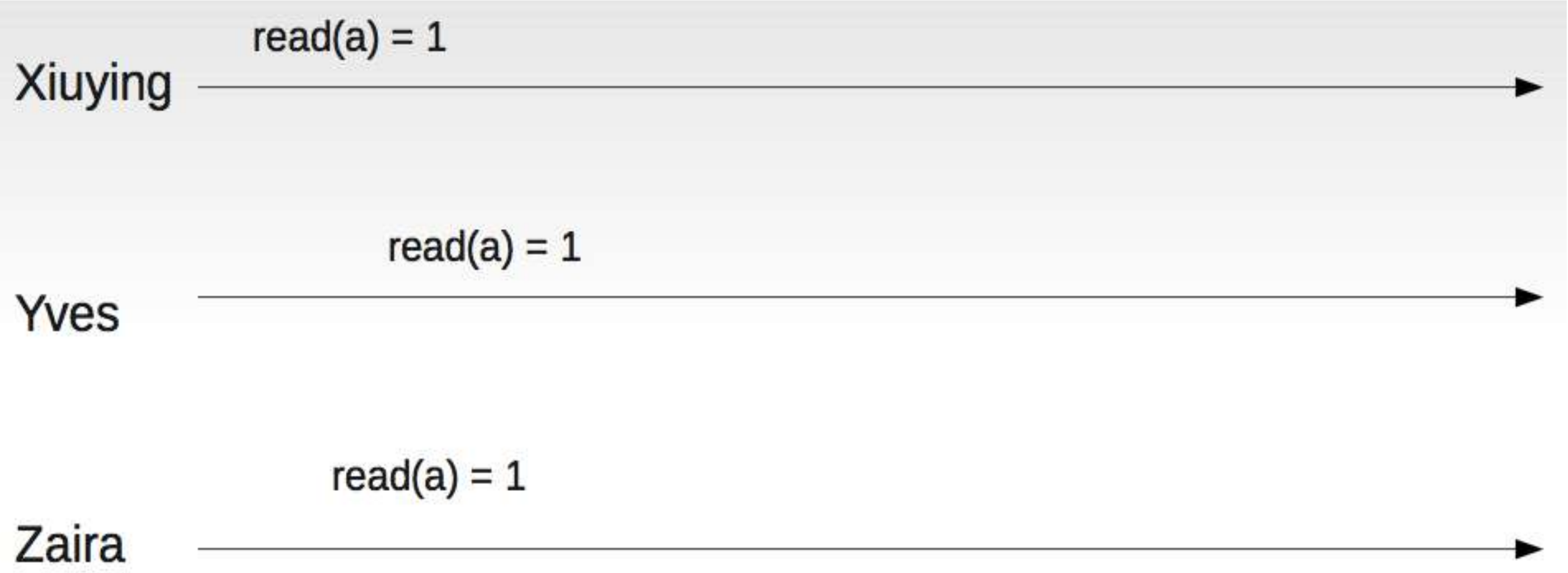


What about concurrent accesses ?

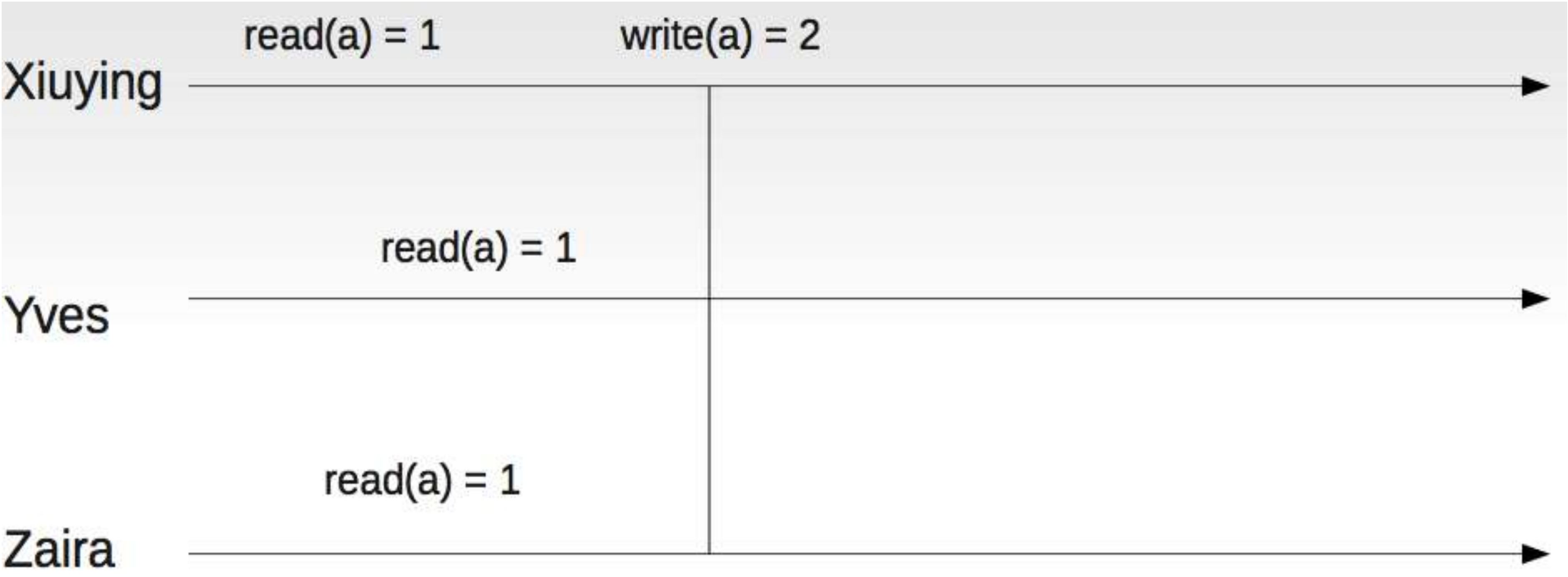
Fundamental properties of RDBMS transactions

- **Atomicity**
  - every operation is executed in "all-or-nothing" fashion
- **Consistency**
  - every transaction preserves the consistency constraints on data: strong consistency
- **Isolation**
  - transactions do not interfere
- **Durability**
  - after a commit, the updates made are permanent regardless possible failures

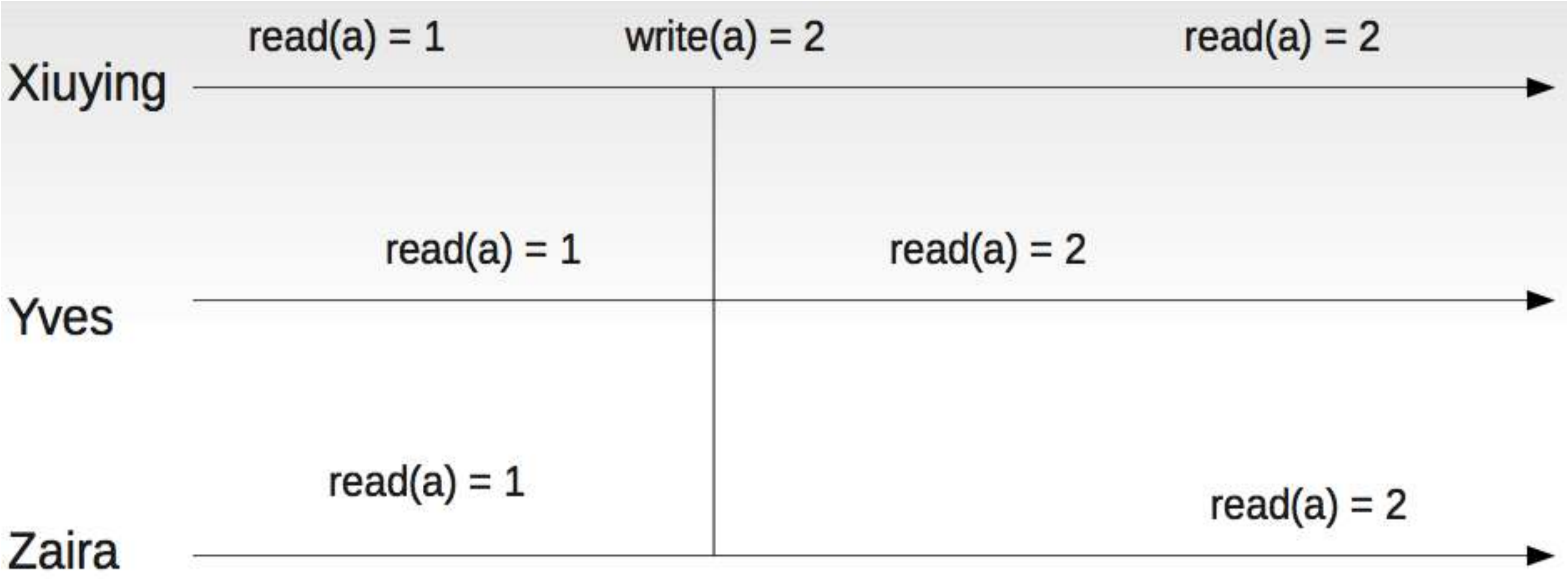
Strong Consistency



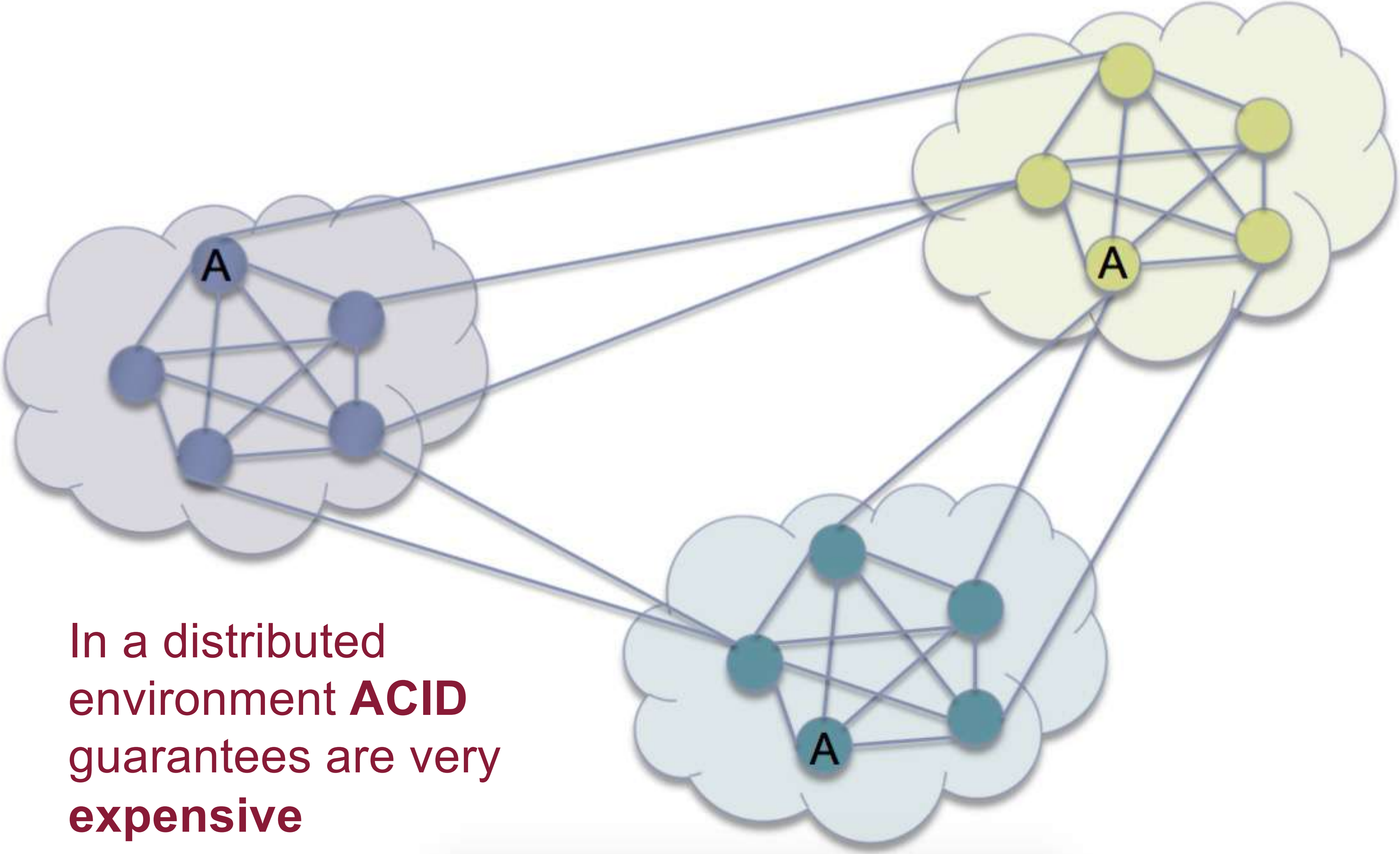
Strong Consistency



Strong Consistency



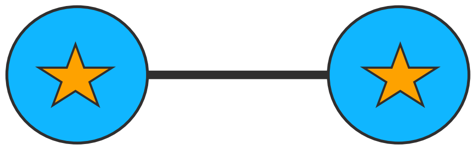
Big Data: network shared data systems



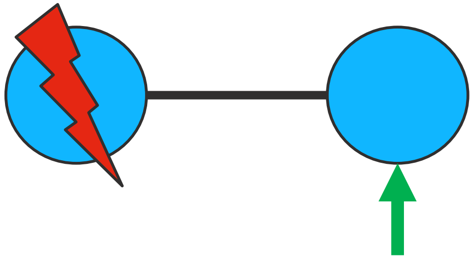
In a distributed environment **ACID** guarantees are very **expensive**

Fundamental properties of distributed data management systems

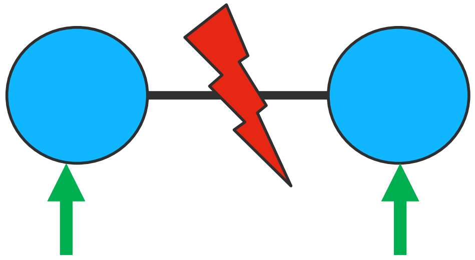
- **Consistency**
  - All nodes see the same data at the same time



- **Availability**
  - Every request receives a response
  - Node failures do not prevent survivors from continuing to operate

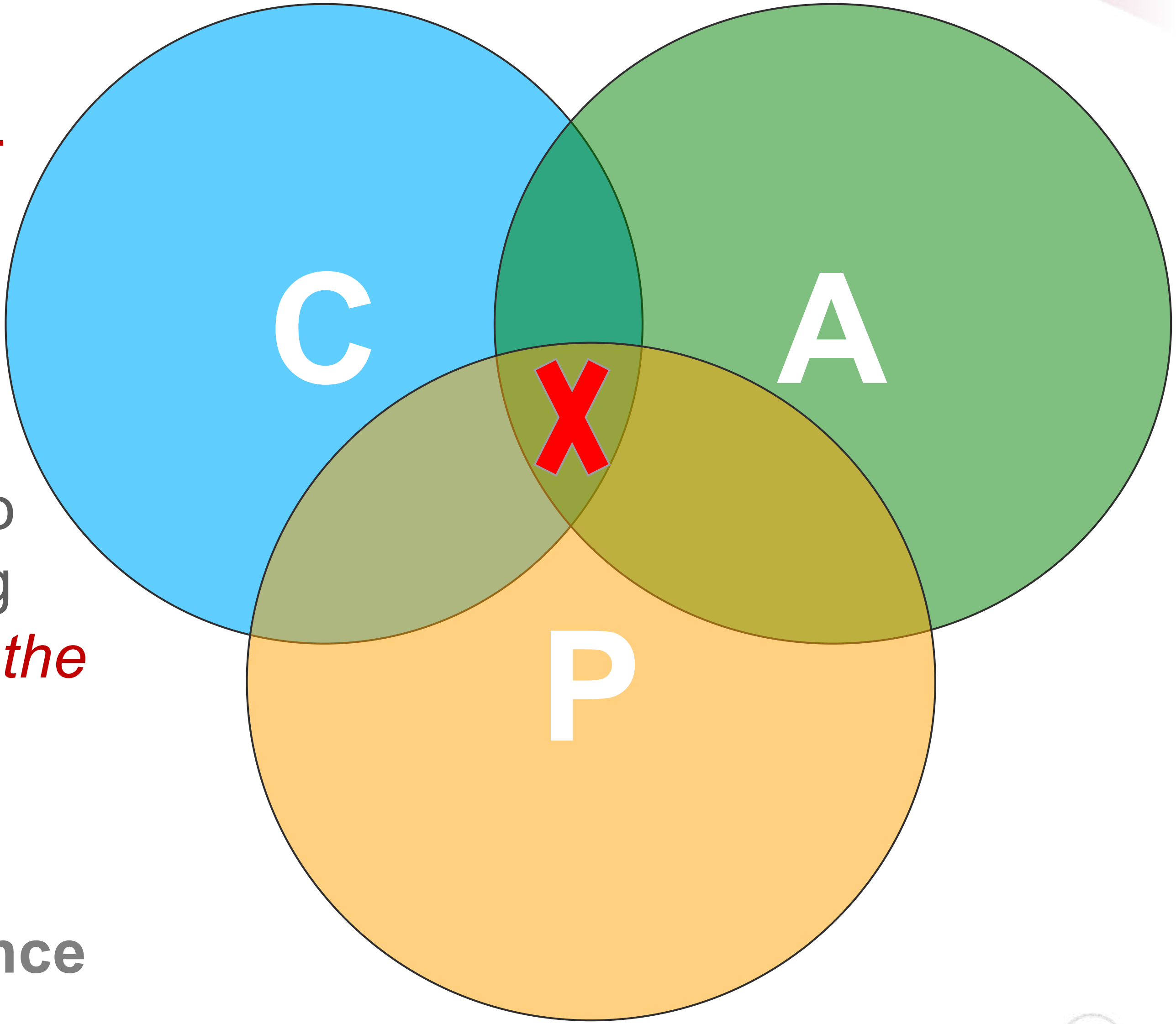


- **Partitioning**
  - Surviving failures of parts of the system
  - The system continues to operate despite arbitrary message loss



CAP Theorem

- Describes the trade-offs involved in distributed systems
- It is impossible for a distributed service to provide the following *three guarantees at the same time*
  - **C**onsistency
  - **A**vailability
  - **P**artition-tolerance

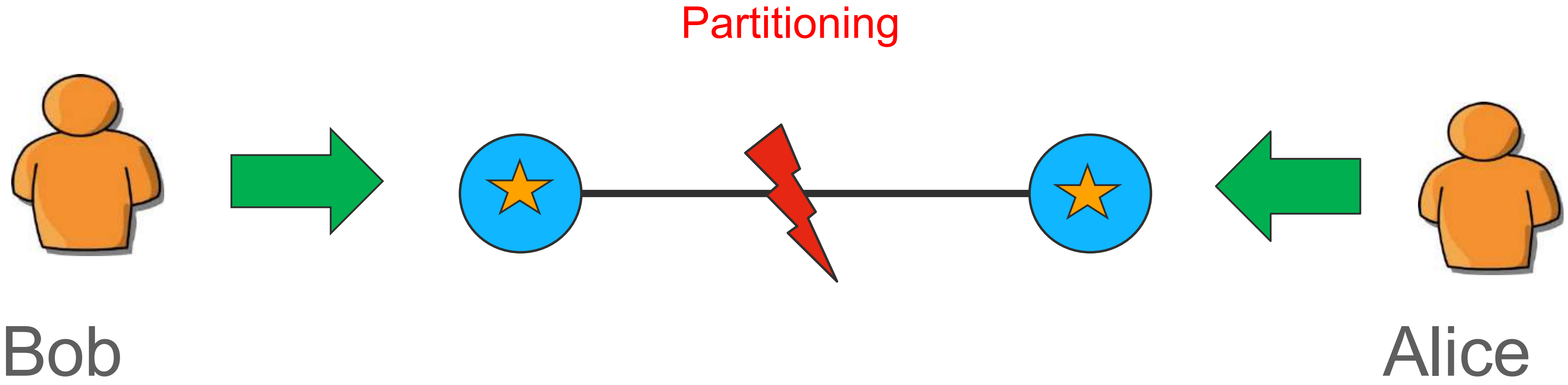


Pick two !

CAP Theorem

A simple example (and an informal proof)

Hotel booking: are we double-booking the same room?

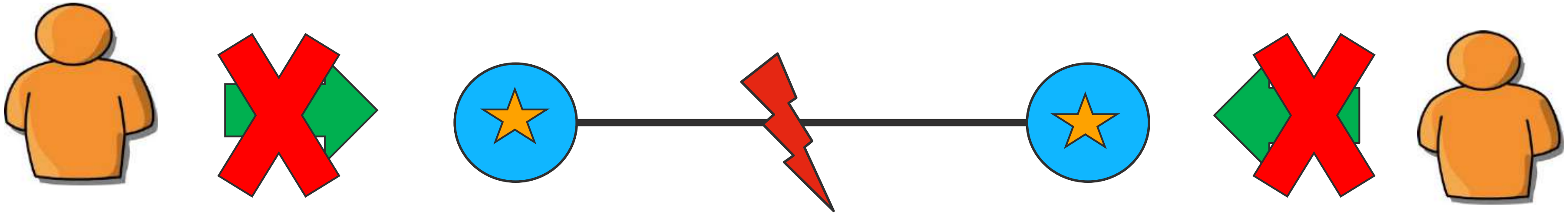


CAP Theorem

A simple example (and an informal proof)

Hotel booking: are we double-booking the same room?

Partitioning

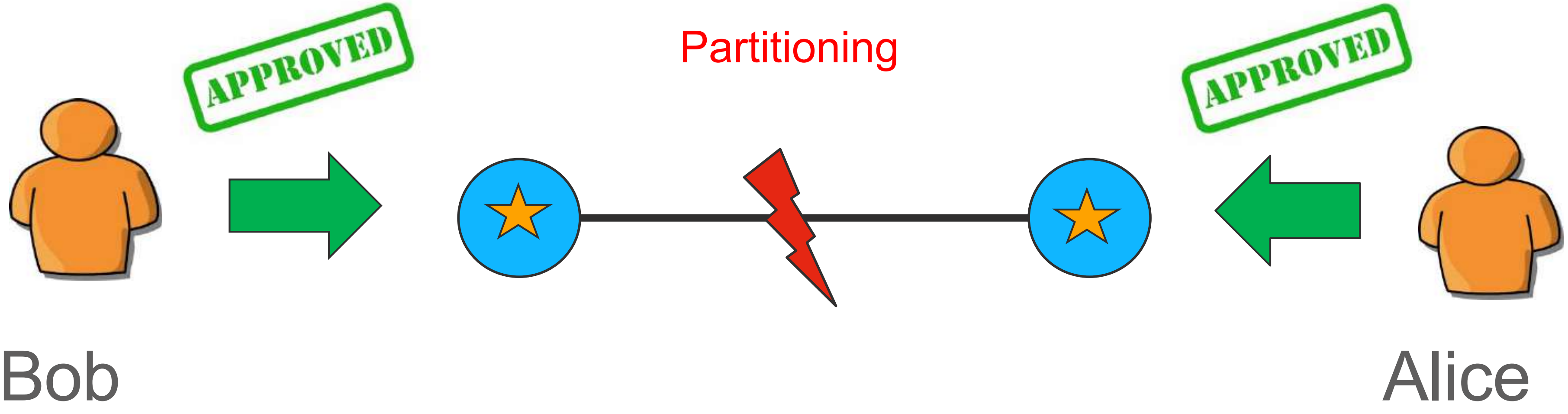


No booking: give up availability

CAP Theorem

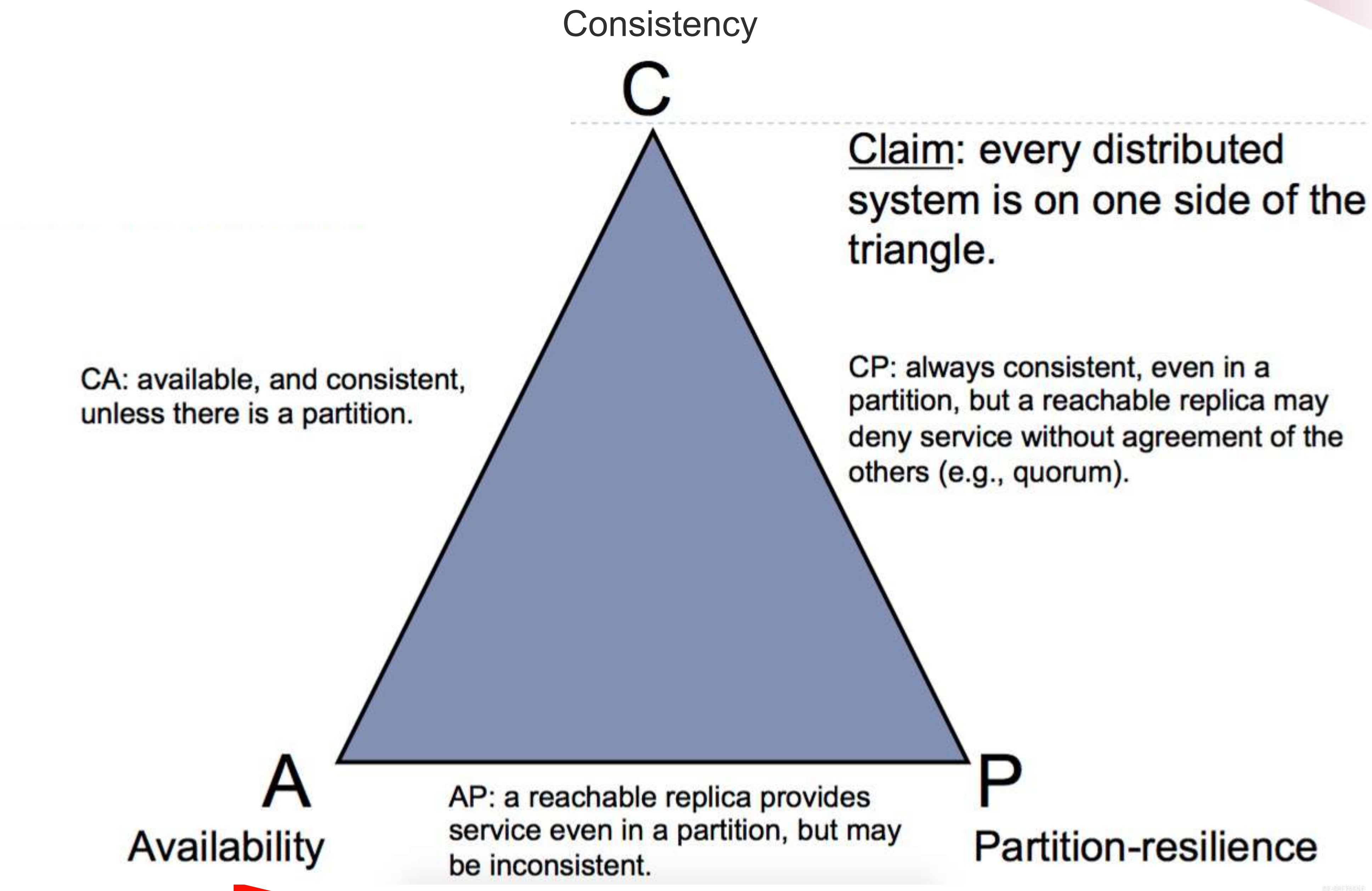
A simple example (and an informal proof)

Hotel booking: are we double-booking the same room?



Booking done: give up consistency

CAP Theorem



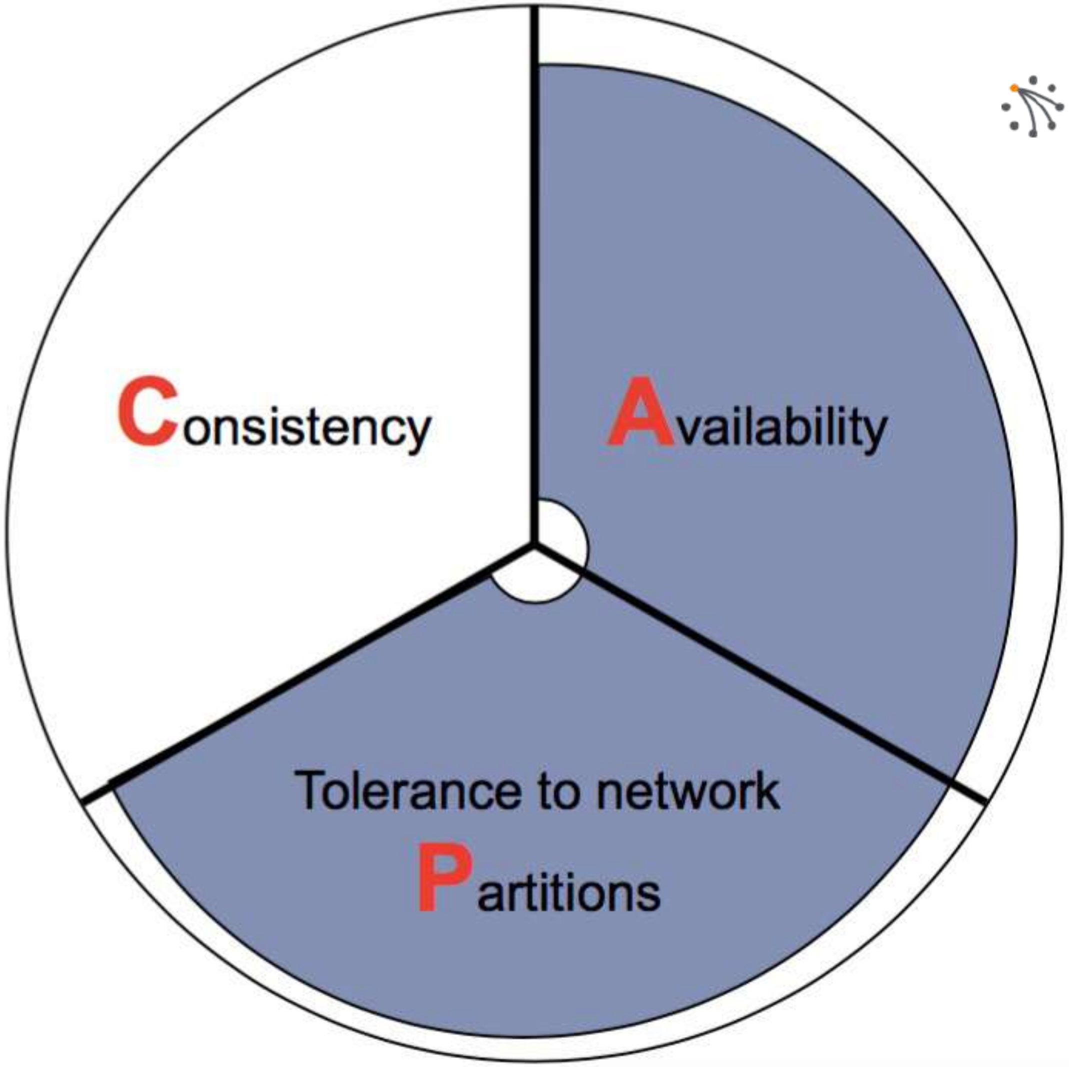
Claim: every distributed system is on one side of the triangle.

CA: available, and consistent, unless there is a partition.

CP: always consistent, even in a partition, but a reachable replica may deny service without agreement of the others (e.g., quorum).

AP: a reachable replica provides service even in a partition, but may be inconsistent.

Forfeit consistency



riak cassandra CouchDB

Examples

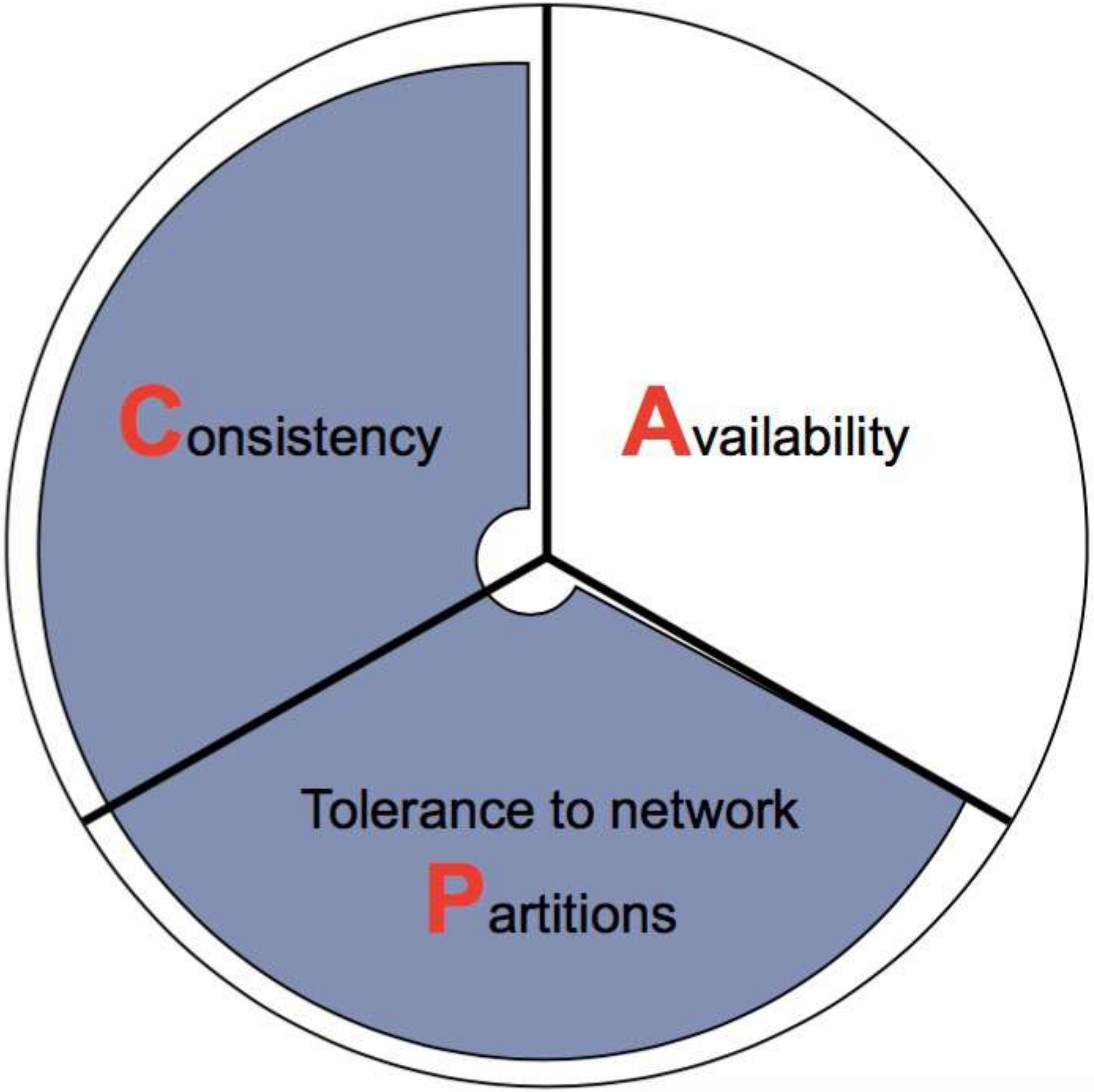
- Social networks
- Web caching
- DNS

Traits

- expirations/leases
- conflict resolution
- Optimistic

Best effort consistency

Forfeit availability



mongoDB redis

Examples

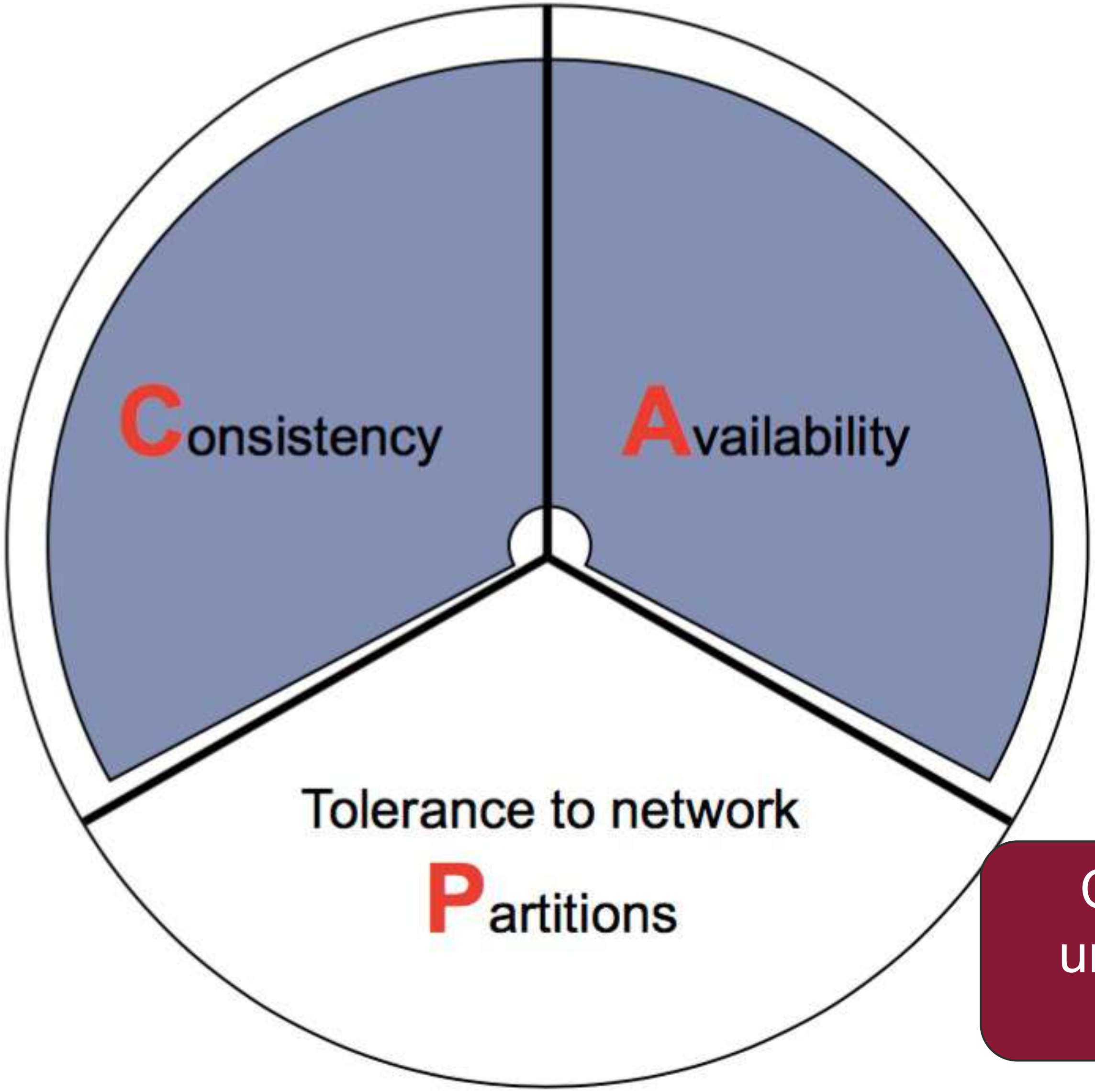
- Distributed databases
- Distributed locking
- Majority protocols

Traits

- Pessimistic locking
- Make minority partitions unavailable

Best effort availability

Forfeit partitions



MySQL PostgreSQL

Examples

- Single-site databases
- Cluster databases
- LDAP
- Fiefdoms

Traits

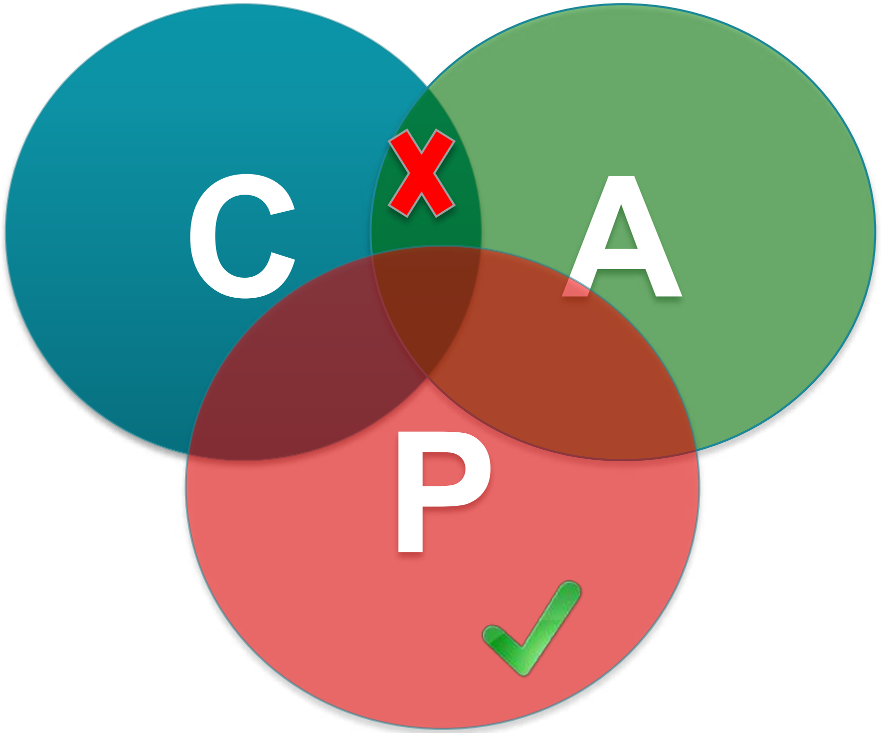
- 2-phase commit
- cache validation protocols

Can a distributed system (with unreliable network) really be not tolerant of partitions?

- To scale out, you have to distribute resources
- Partition is not really an option but rather a need
- The real selection is among consistency or availability

Consistency or Availability

- Consistency or Availability is not a “binary” decision
- AP systems relax consistency in favor of availability – but are not inconsistent
- CP systems sacrifice availability for consistency- but are not unavailable
- So, both AP and CP systems can offer a degree of consistency and availability, as well as partition tolerance



“Degrees” of consistency

Strong Consistency

- After the update completes, any subsequent access will return the same updated value

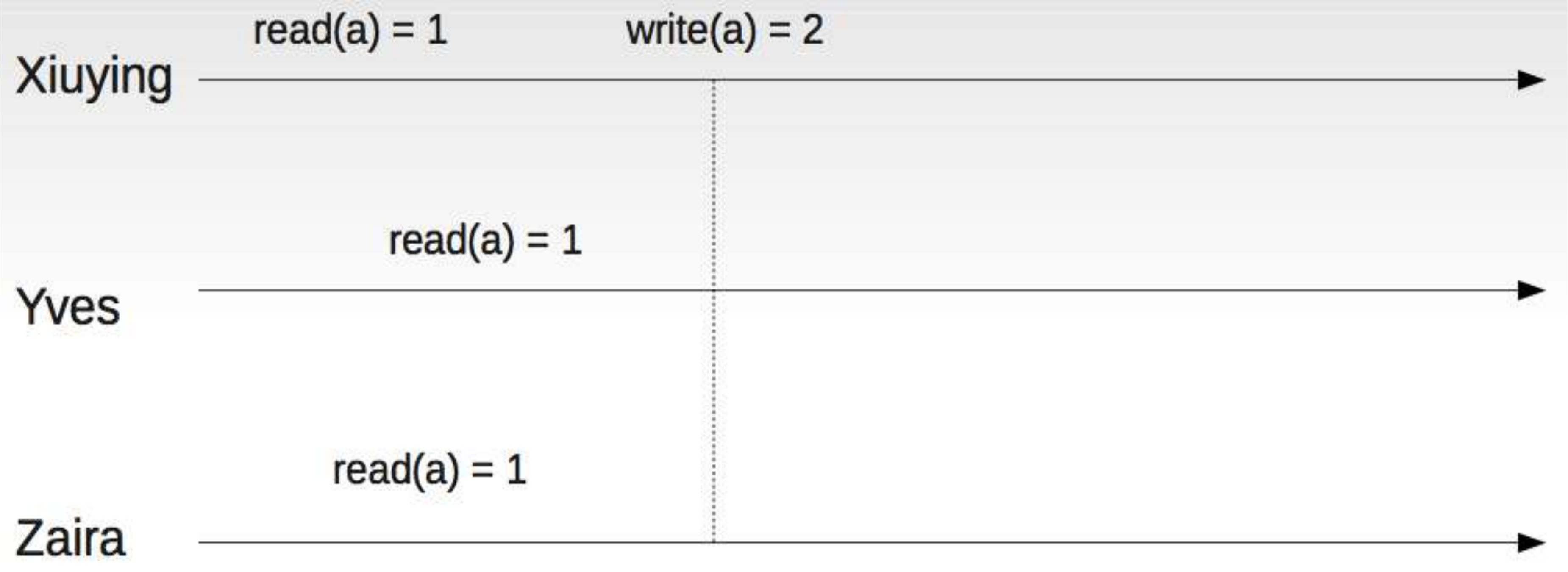
Eventual Consistency

- It is guaranteed that if no new updates are made to object, eventually all accesses will return the last updated value (e.g., propagate updates to replicas in a lazy fashion)

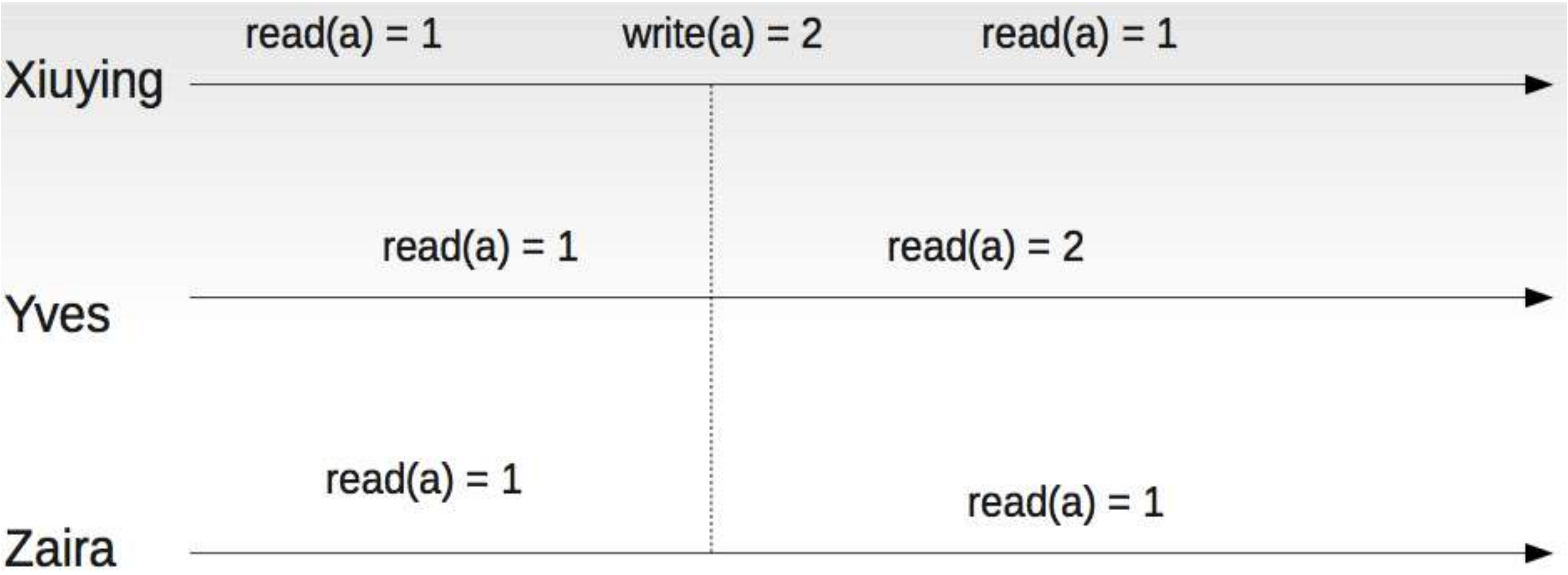
Weak Consistency

- It is not guaranteed that subsequent accesses will return the updated value

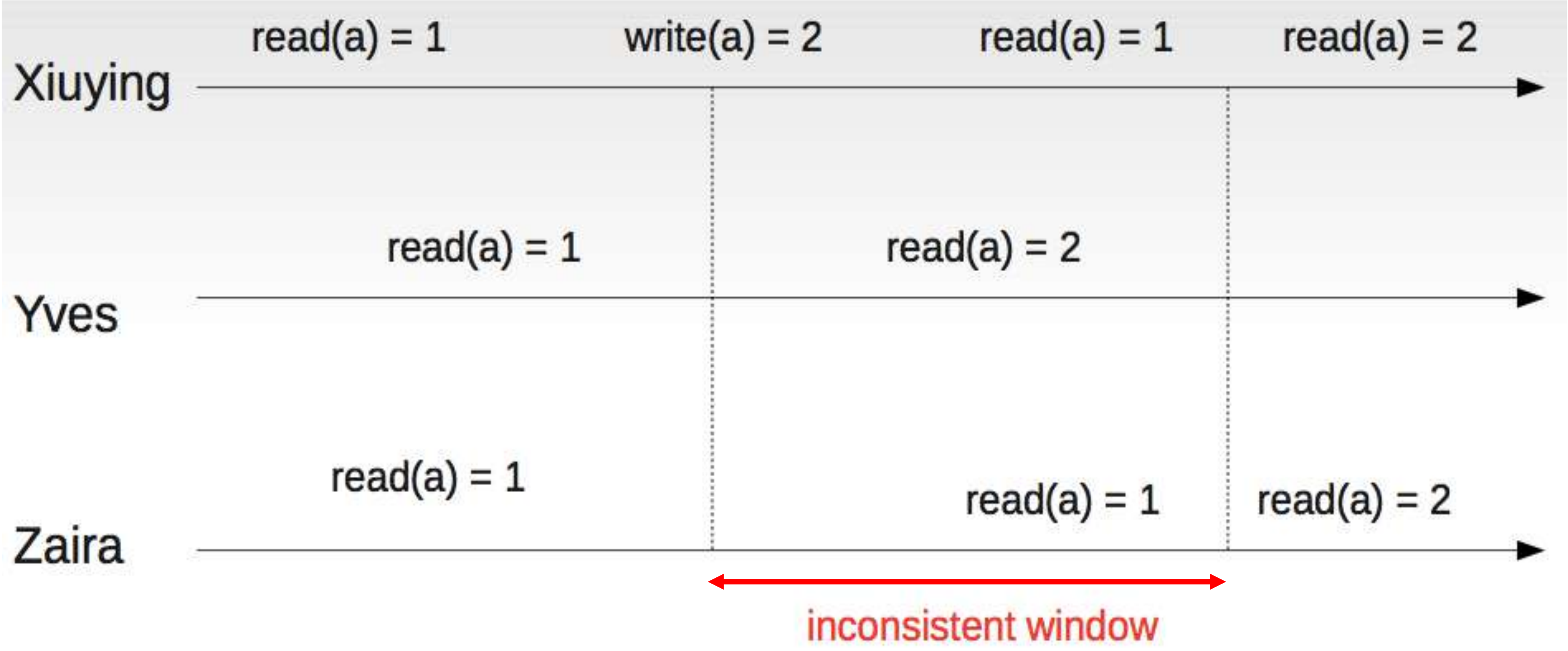
Eventual consistency



Eventual consistency



Eventual consistency

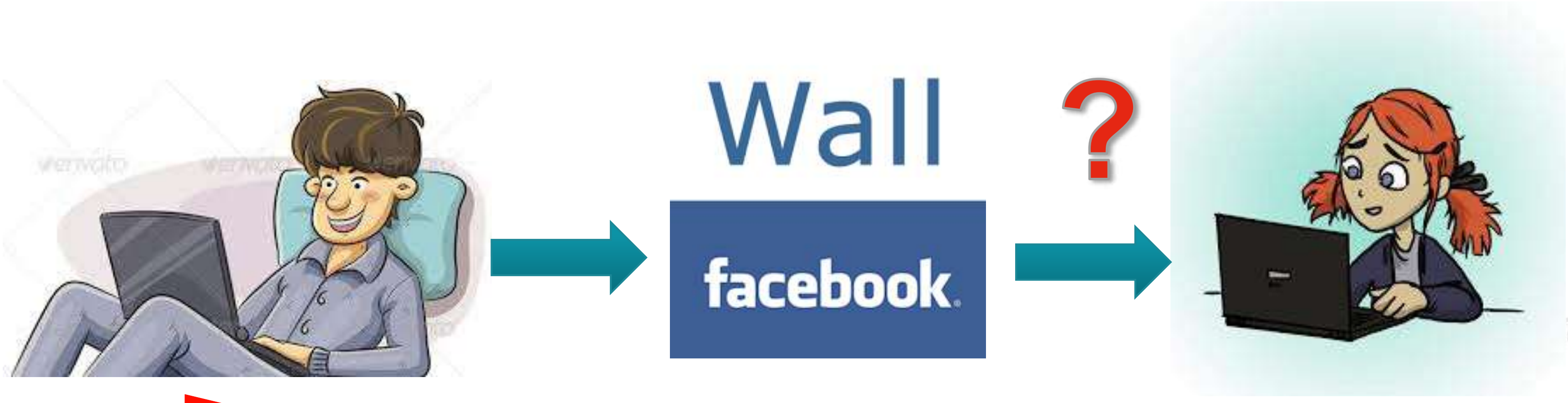


Eventual consistency

Facebook example

- Bob finds an interesting story and shares it with Alice by posting on her Facebook wall
- Bob asks Alice to check it out
- Alice logs in her account, checks her Facebook wall but finds:

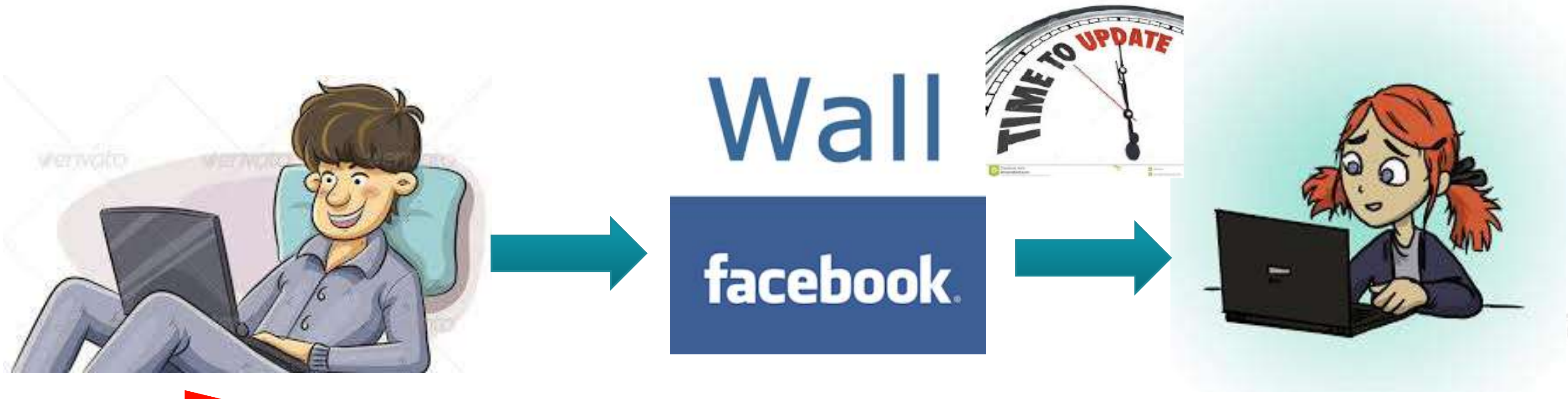
Nothing is there!



Eventual consistency

Facebook example

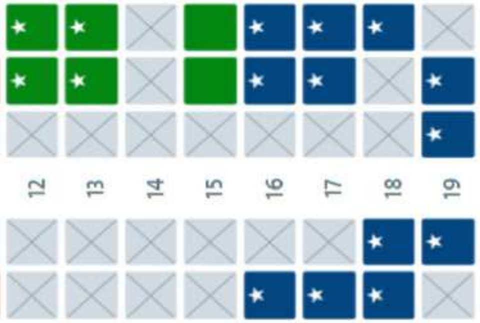
- Reason: Facebook uses an eventual consistent model
- Why this instead of strong consistency?
- Facebook has more than 1 billion active users
- It is non-trivial to efficiently and reliably store the huge amount of data generated at any given time
- Eventual consistent model offers the option to reduce the load and improve availability



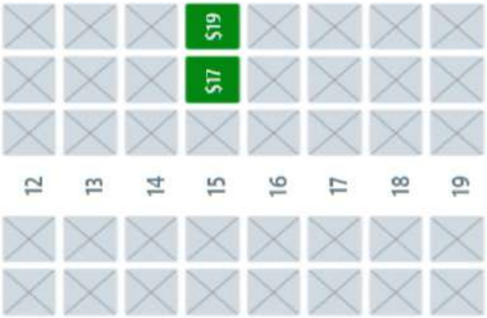
Dynamic tradeoff between Consistency and Availability

Airline reservation system

- When most of seats are available: it is ok to rely on somewhat out-of-date data, availability is more critical



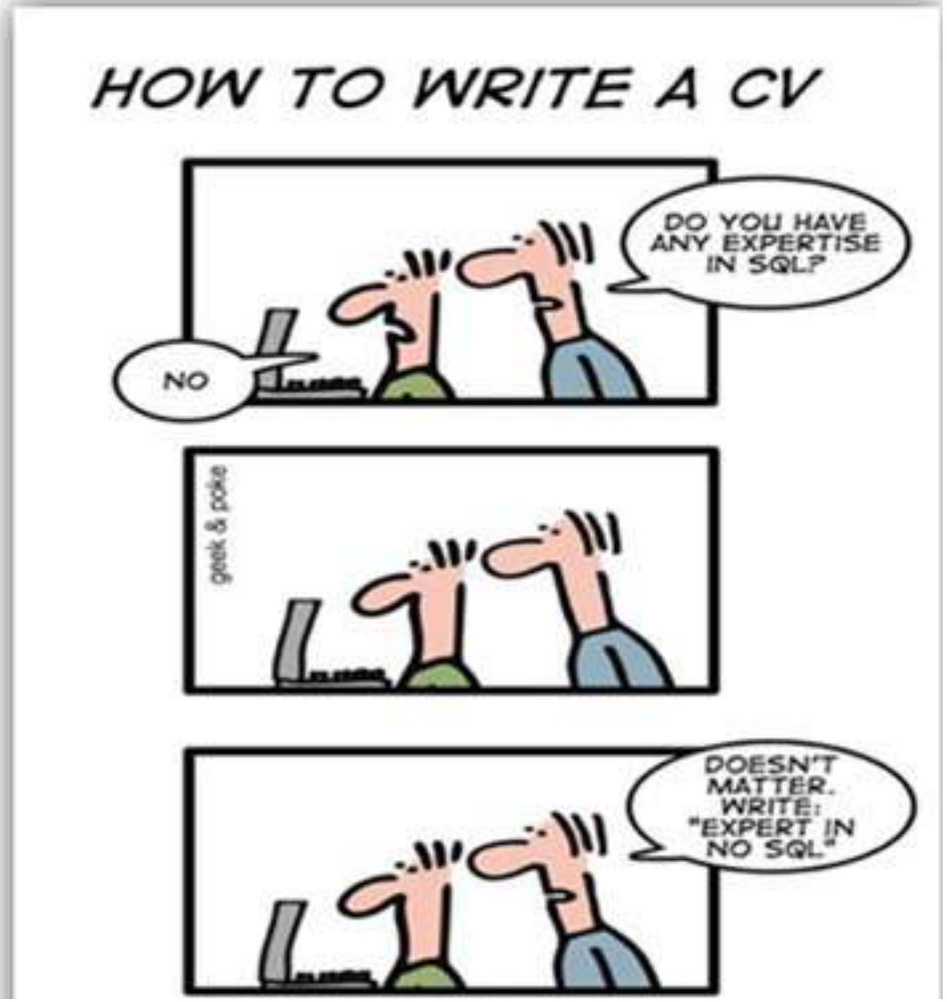
- When the plane is close to be filled: it needs more accurate data to ensure the plane is not overbooked, consistency is more critical



CAP Theorem and Big Data

BigData systems usually give up on (strong) consistency

Example:
NoSQL
(Not Only SQL)
databases


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

Consistency vs. Latency tradeoff

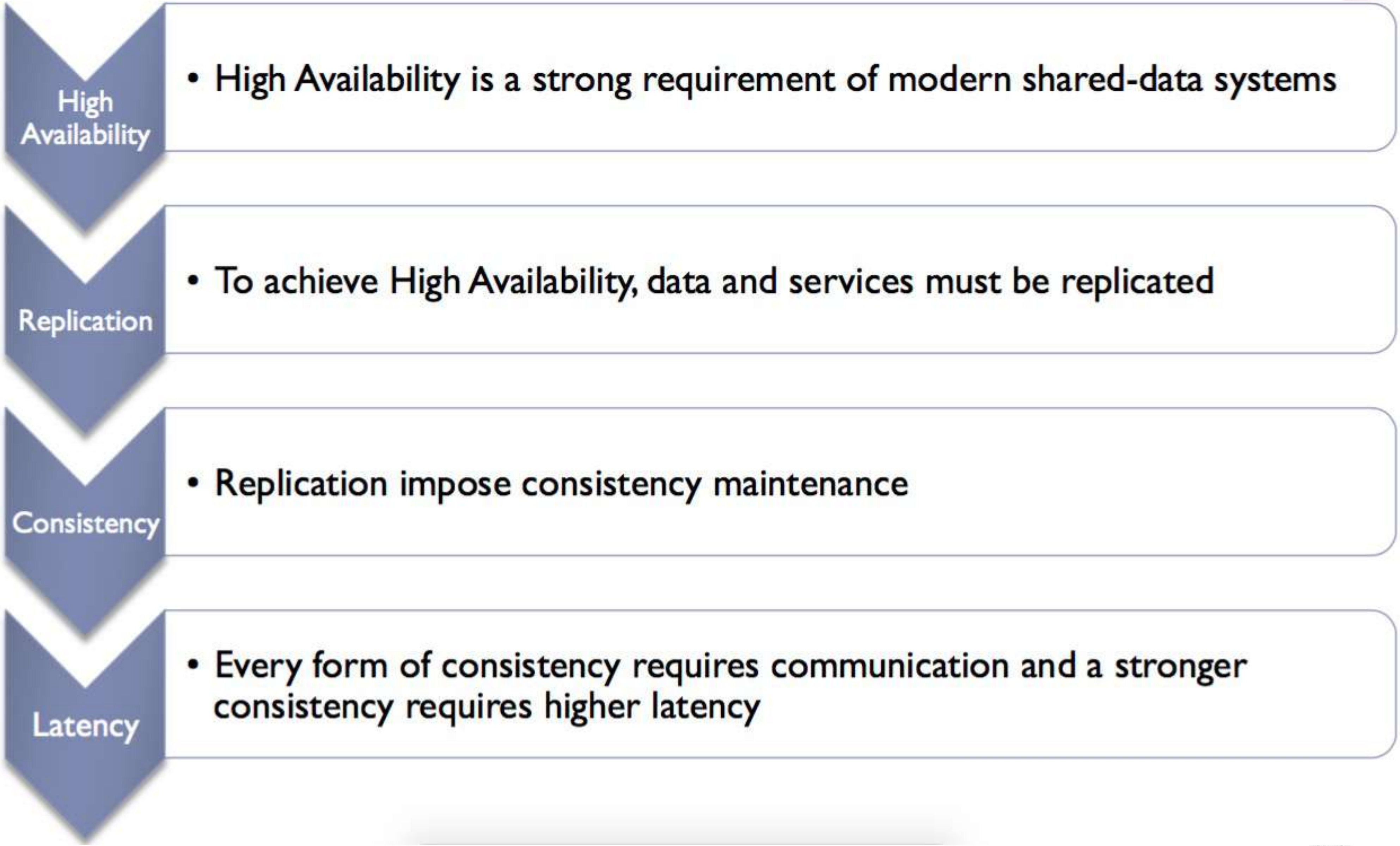
- CAP does not force designers to give up A or C but why there exists a lot of systems trading C?

- **Latency!**

- CAP does not explicitly talk about latency...

- ... however latency is crucial to get the essence of CAP

Availability and Latency are (almost) the same thing



High Availability

- High Availability is a strong requirement of modern shared-data systems

Replication

- To achieve High Availability, data and services must be replicated

Consistency

- Replication impose consistency maintenance

Latency

- Every form of consistency requires communication and a stronger consistency requires higher latency