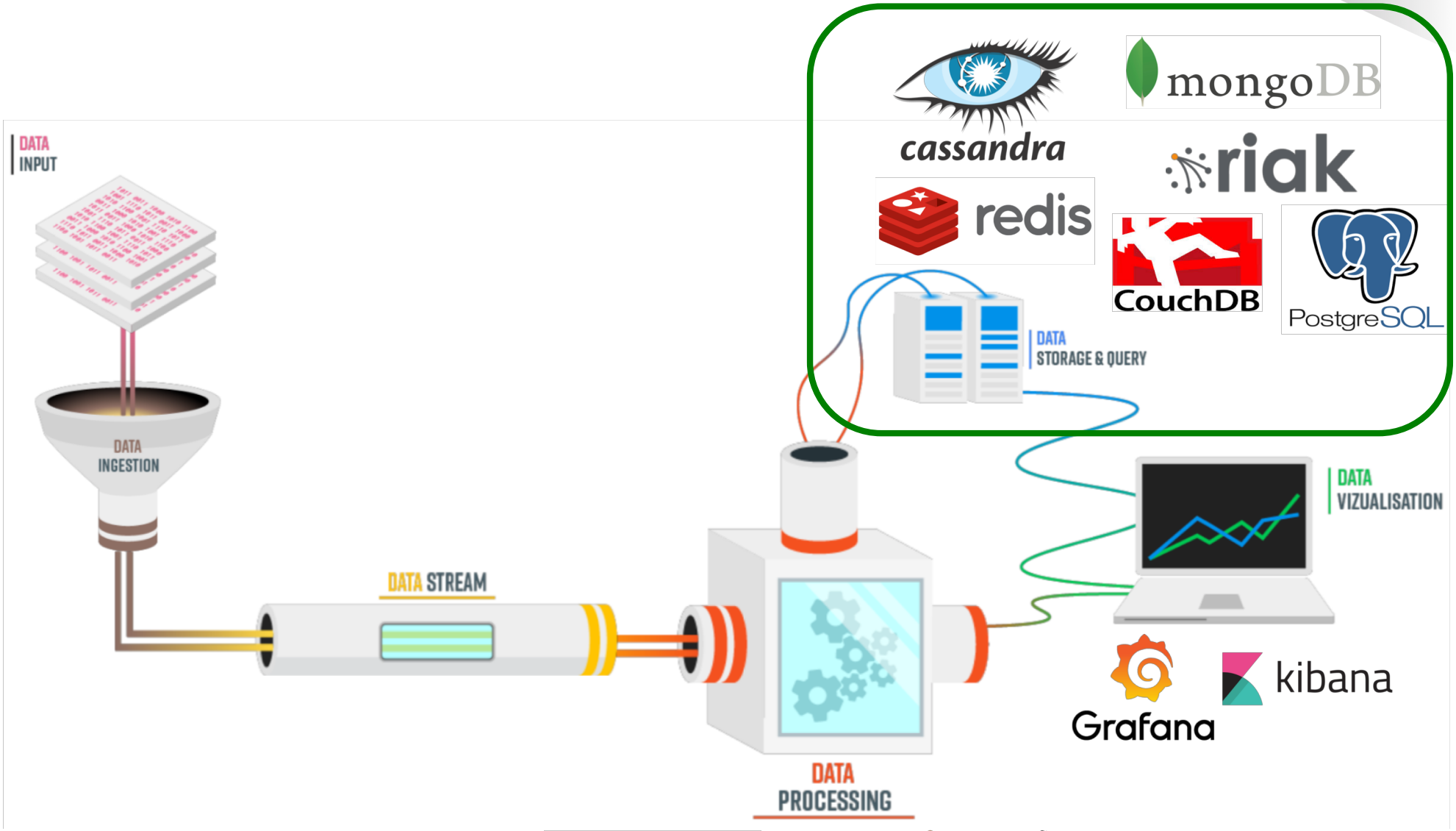
INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

# Data Models for Big Data

Alexandru Costan

# The Big Data pipeline

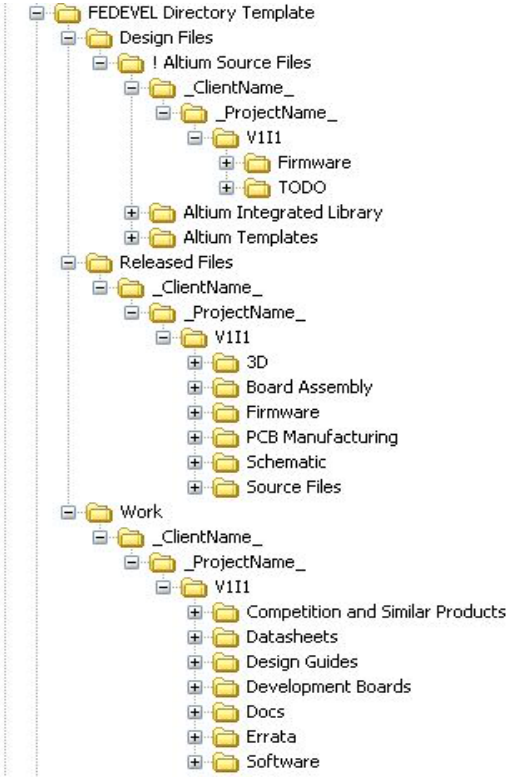


# How do we use data today ?

- A single question requires the **integration** of many data elements, that are
  - in different locations
  - in different formats (JSON, Text, HDF, etc.)
  - described in different ways
- Best **grouping** can vary during investigation
  - Longitudinal, vertical, cross-cutting
- But always needs to be operated on as a **unit**
  - Share, annotate, process, copy, archive

# How do we manage data today?

- Often, a curious mix of ad hoc methods
  - Organize in directories using file and directory naming conventions
  - Capture status in README files, spreadsheets, notebooks
- Time-consuming, complex, error prone




**Why can't we manage our data like we manage our pictures and music?**

# Browse

OVERVIEW CHARTS GENRES & MOODS NEW RELEASES DISCOVER CONCERTS

## Finish strong!


**Rock Solid**
New classics and essential cuts from all your favorite rock bands. / Cover: Soundgarden / Rest in...
804,426 FOLLOWERS

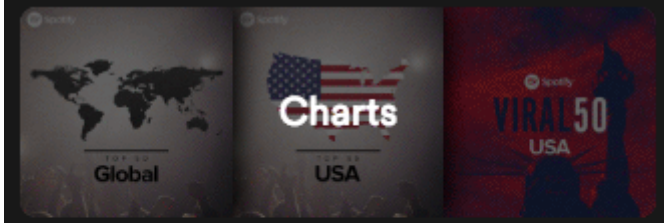

**Hit Rewind**
The biggest hits from the last few years! Listen to all this hits you've been missing in Hit Rewind!...
1,473,648 FOLLOWERS


**Soul Lounge**
Whether you're lounging at home or out with your friends, let this playlist provide your soundtrack....
336,151 FOLLOWERS


**The Realest Down South**
It doesn't get no realer than hip hop from the dirty states.
298,156 FOLLOWERS

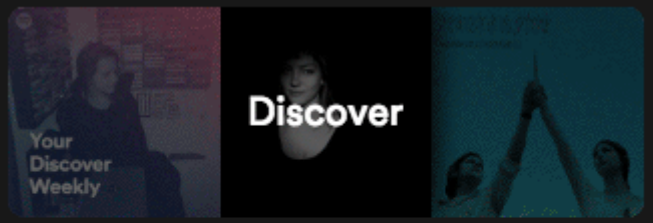

**Country Gold**
Familiar Country Music you know and love! Updated Friday afternoons. Artist - Taylor Swift
815,014 FOLLOWERS


**Teen Party**
Keep your friends dancing with all your favorite party songs courtesy of Post Malone! Congratulations...
2,743,748 FOLLOWERS



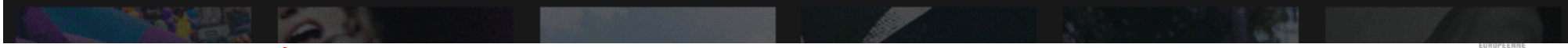

Global and regional top charts


Katy Perry, David Guetta, Phoenix




Recommended for you

## Genres & Moods



# Introducing the dataset

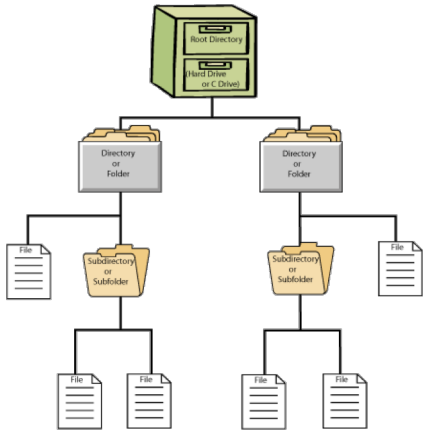


- **A collection of data**
- Group data based on use, not location
  - Logical grouping to organize, search and describe usage
- **Tag** with characteristics that reflect content or the current status in investigation
  - Stage of processing, provenance, validation
- **Share** data sets for collaboration
  - Control access to data and metadata
- Operate on datasets as **units**
  - Copy, export, analyze, tag, archive

# Introducing the dataset

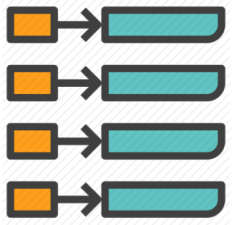


- Can still be stored in files…

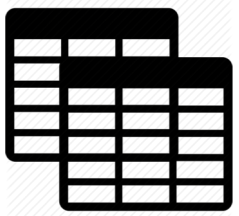



- … but also in
  - Binary Large Objects


  - Key-Value stores

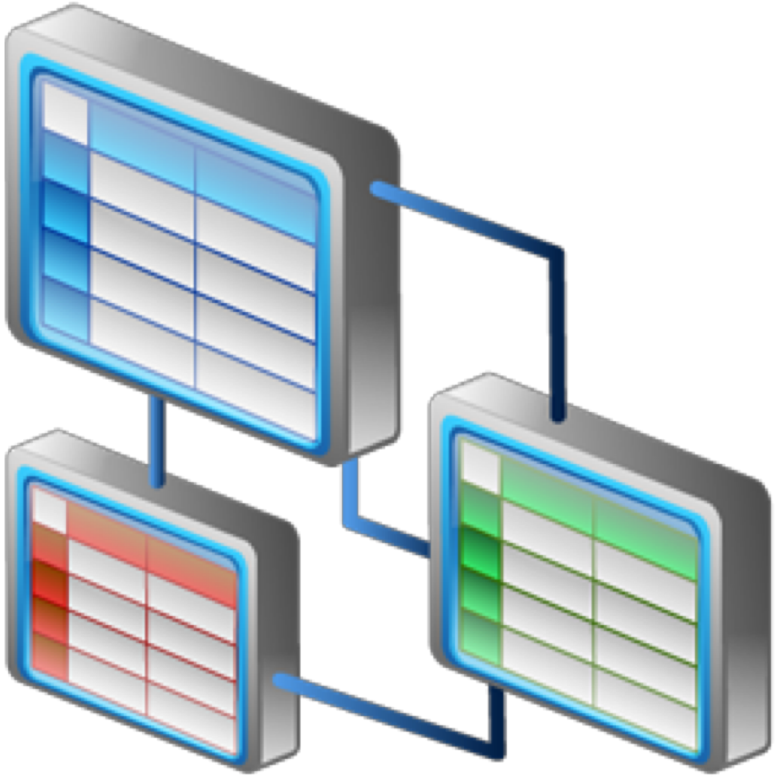


  - Tables



# 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 |

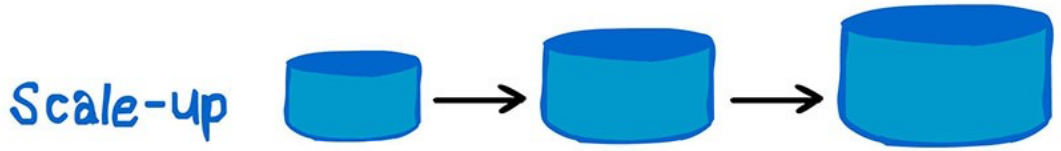
most interesting queries require computing **joins**

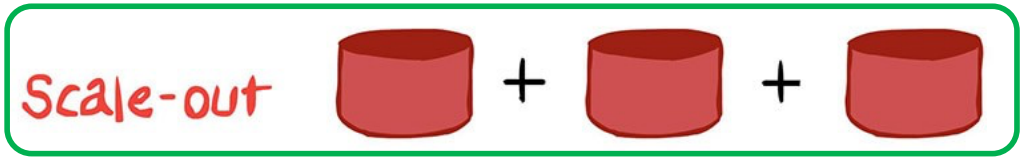
TABLE lectures

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

# Scaling out

- Issues when the dataset is just too big
- Began to look at multi-node database solutions
  - RDBMS were not designed to be distributed
- Known as 'scaling out' or 'horizontal scaling'

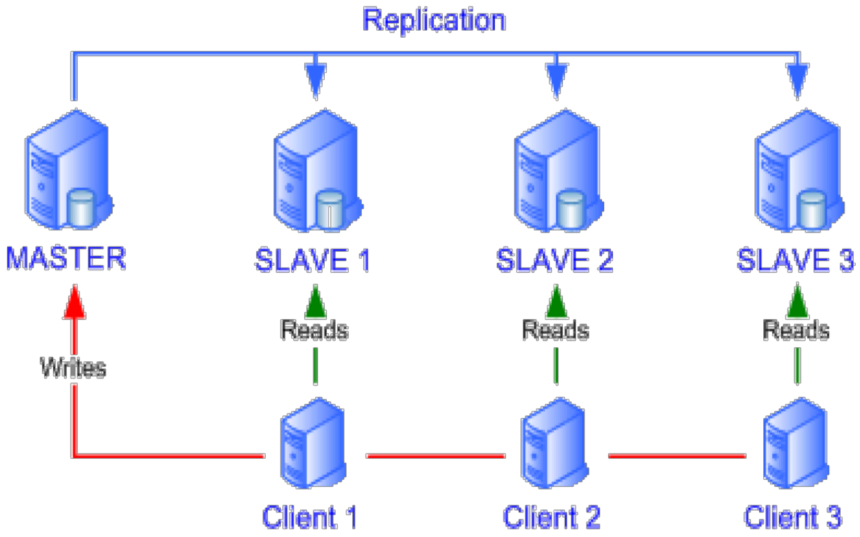



- Different approaches include:
  - **Master-slave**
  - **Sharding**

# Scaling RDBMS: Master/Slave

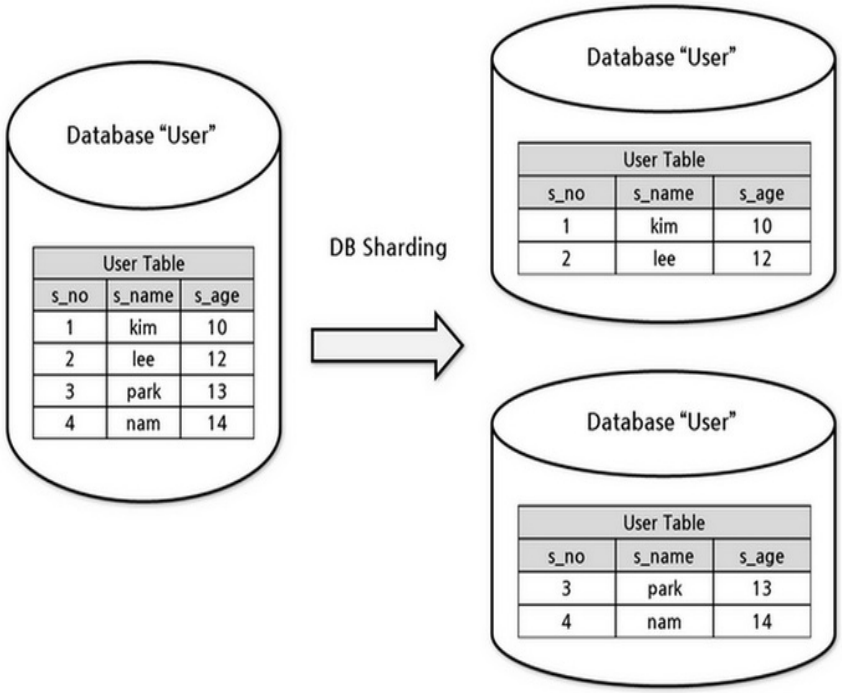
- Data **replicated** on slaves
  - All *writes* are written to the *master*
  - All *reads* from the replicated *slaves*




- 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

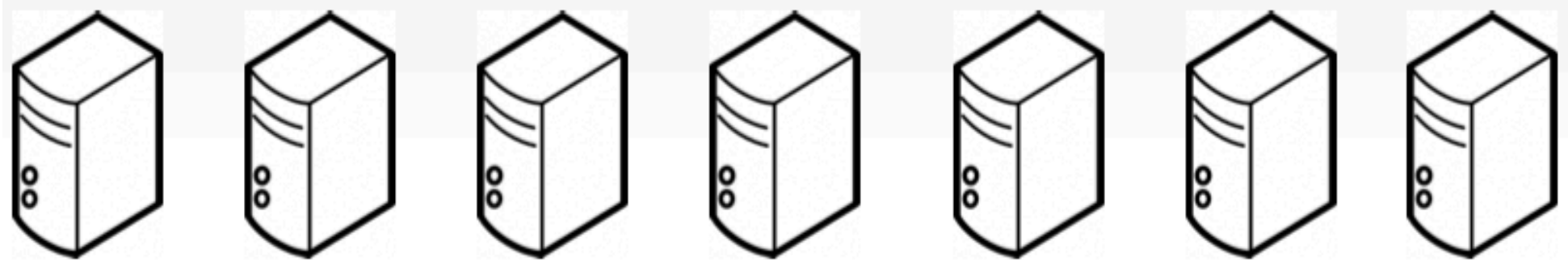
# Scaling RDBMS: Sharding

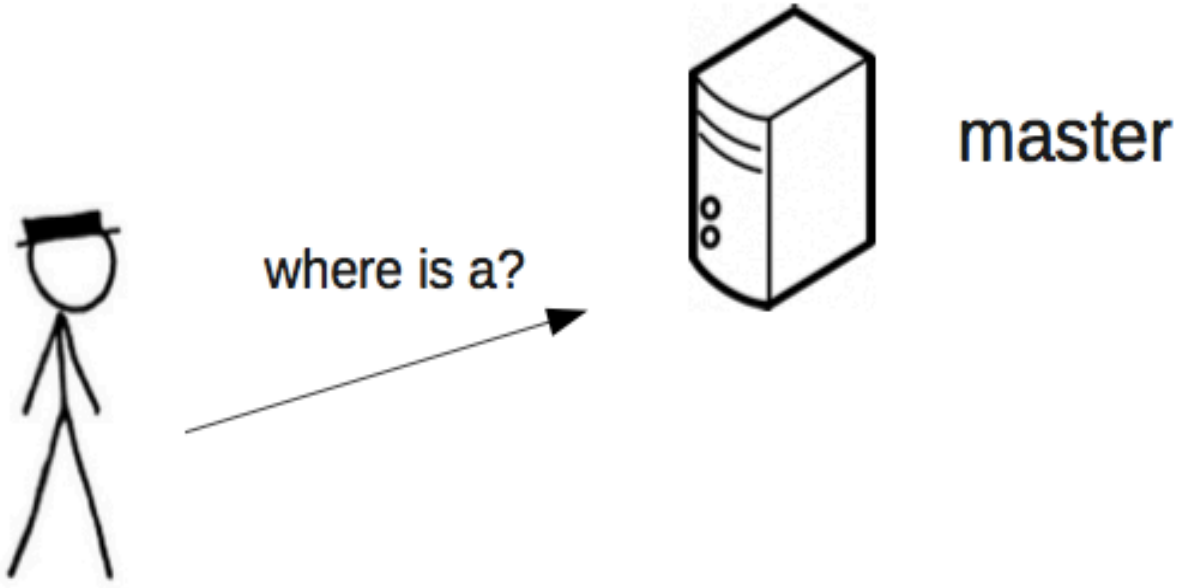
- Data **partitioned** to slaves
- Advantage
  - Scales well for both reads and writes
- Problems
  - Not transparent, application needs to be partition-aware
  - Can no longer have relationships/joins across partitions
  - Loss of referential integrity across shards



# How sharding works

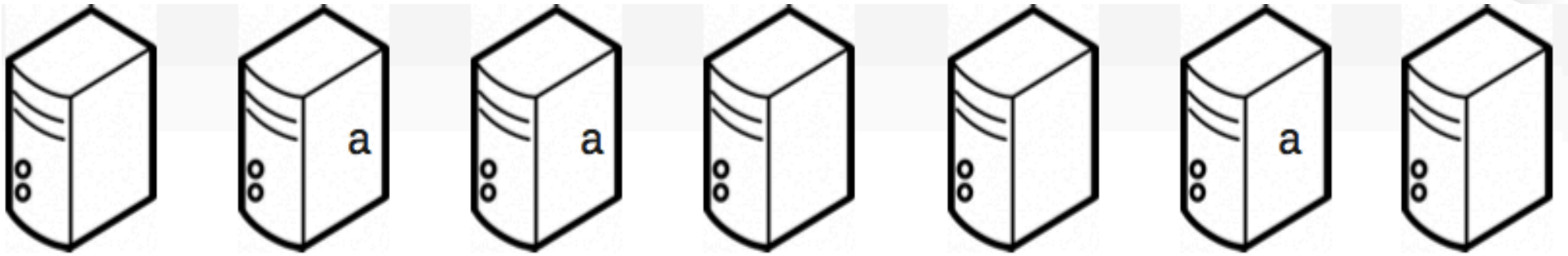


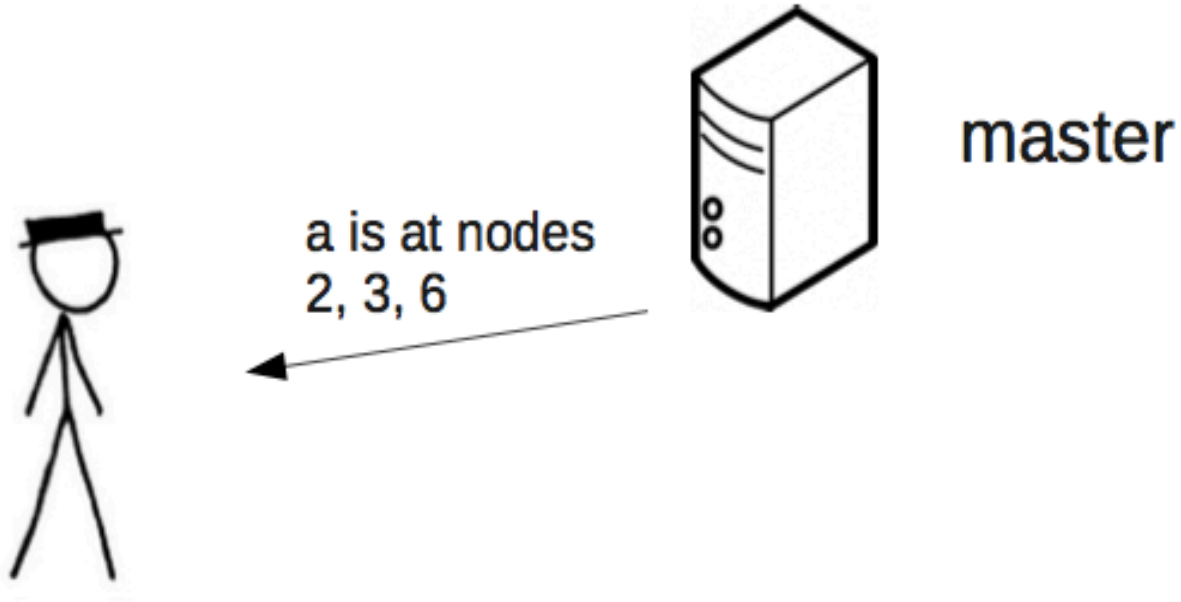




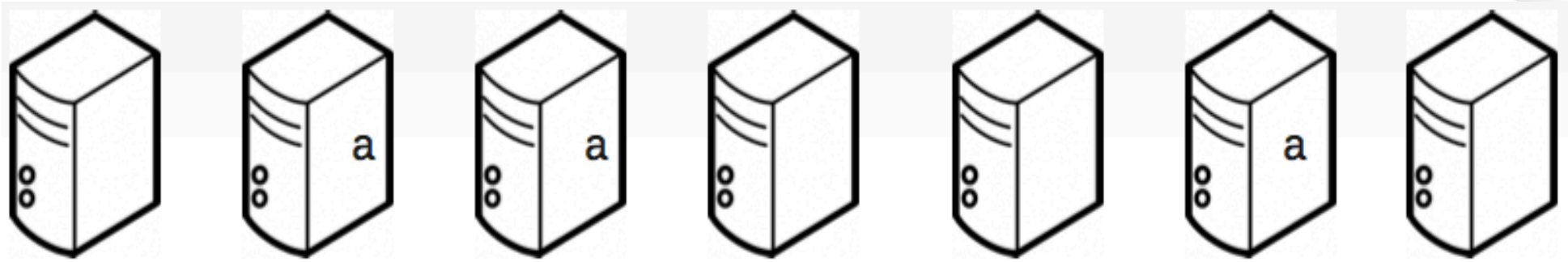
# How sharding works

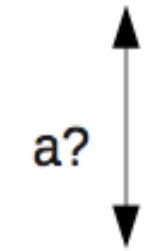




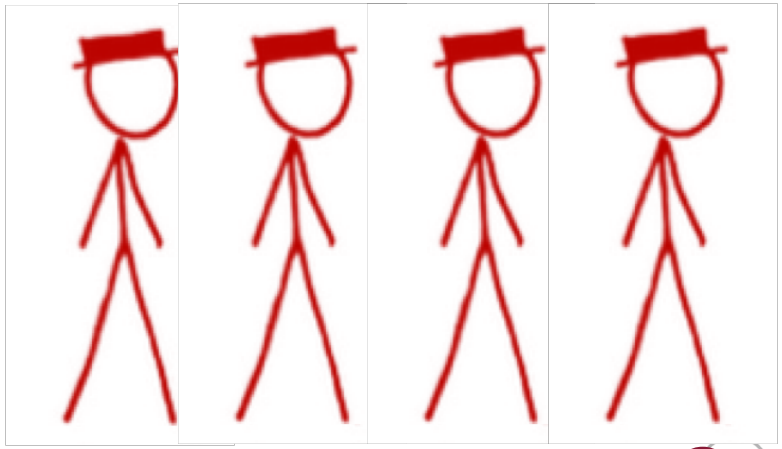



# How sharding works




master



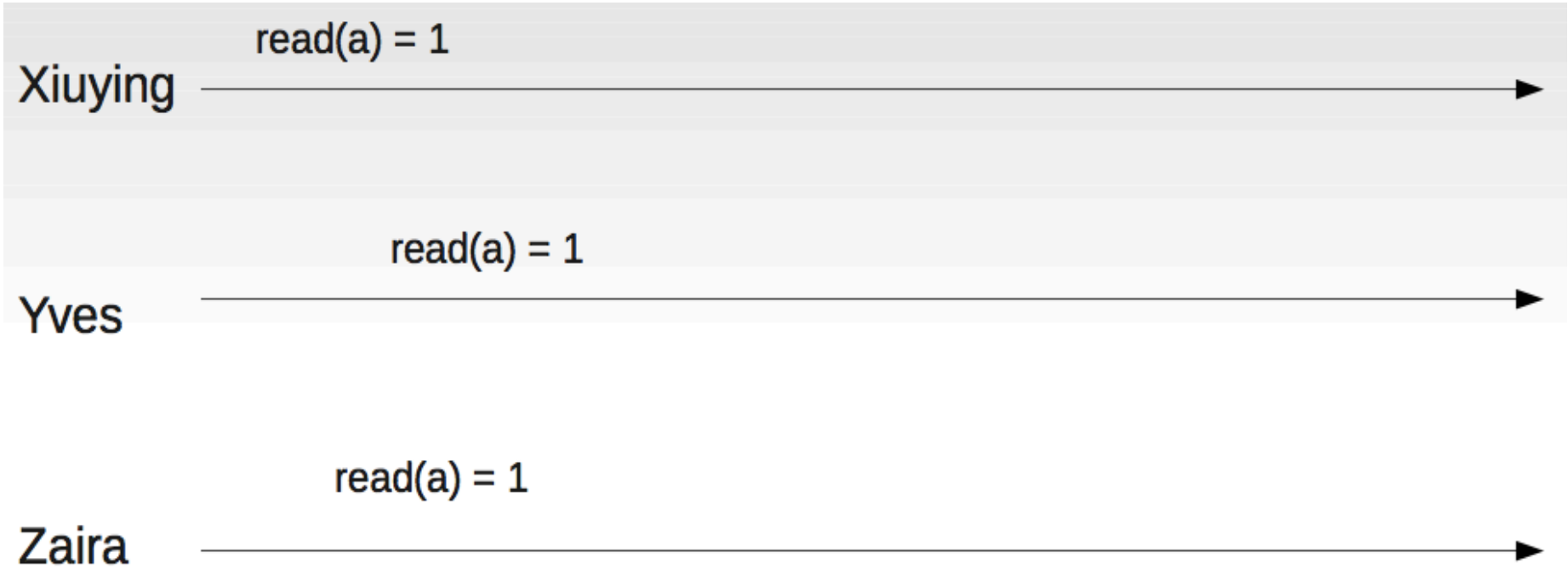
What about concurrent accesses ?

# 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

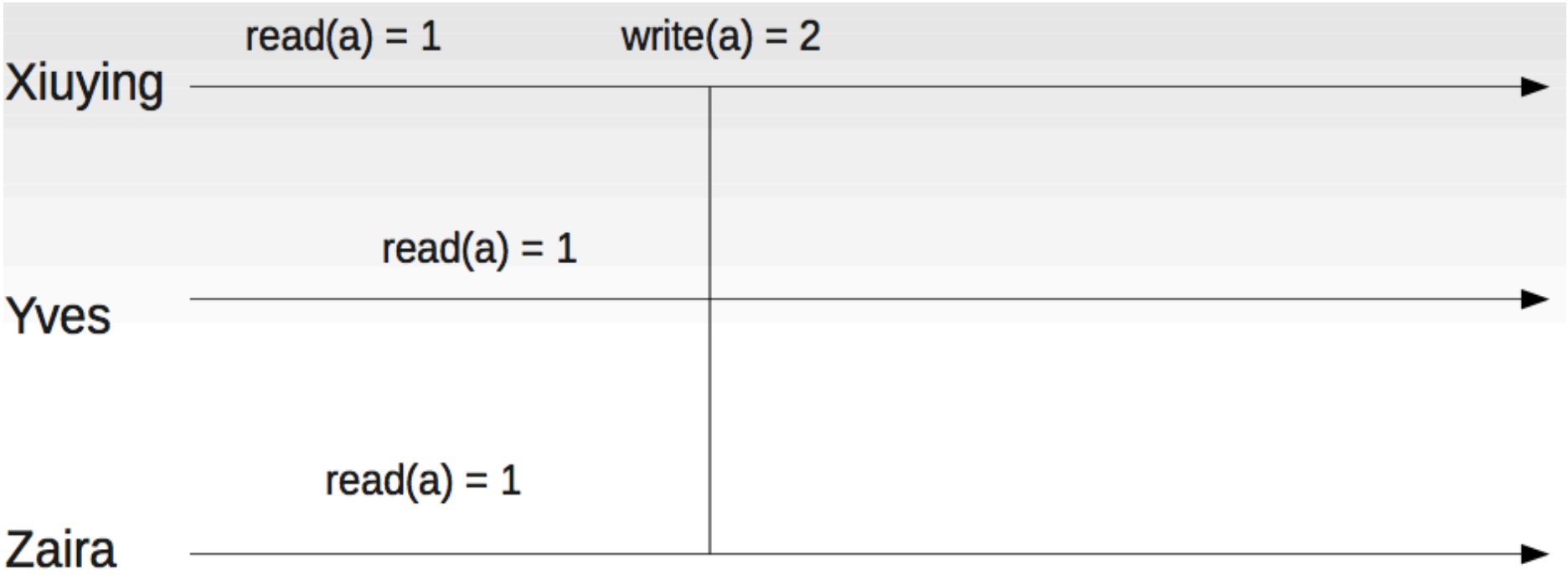
# Strong Consistency





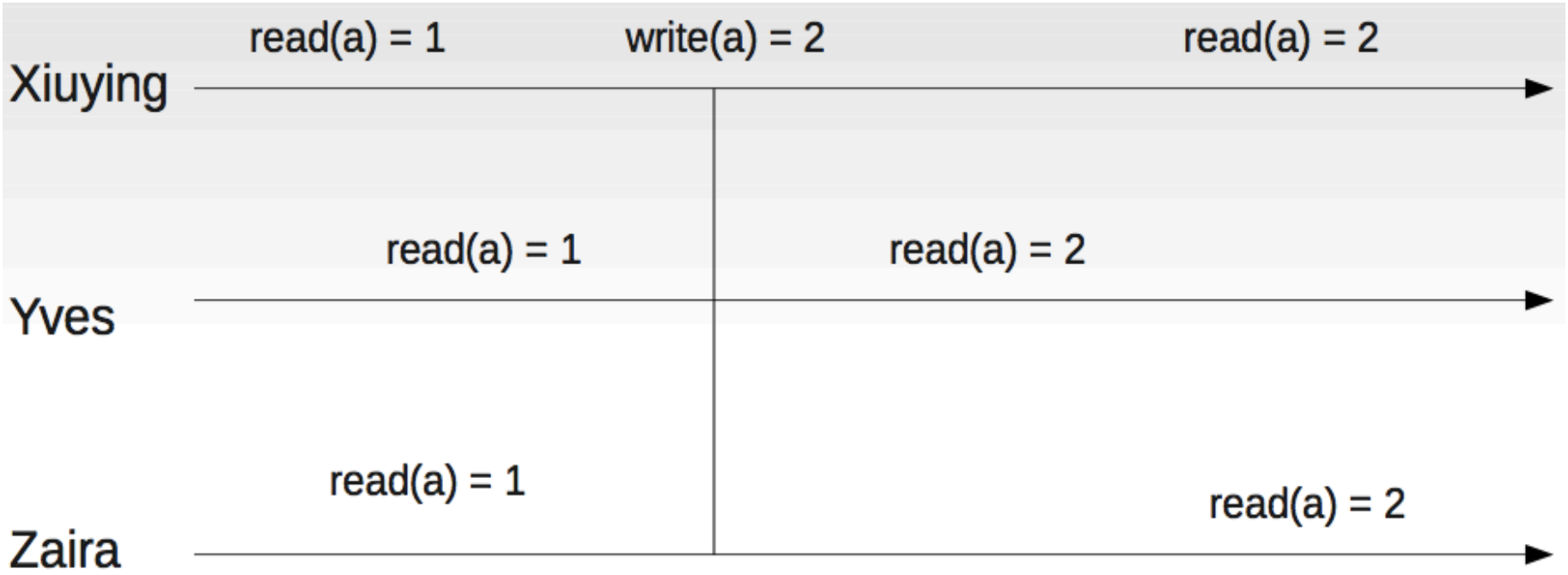
# Strong Consistency





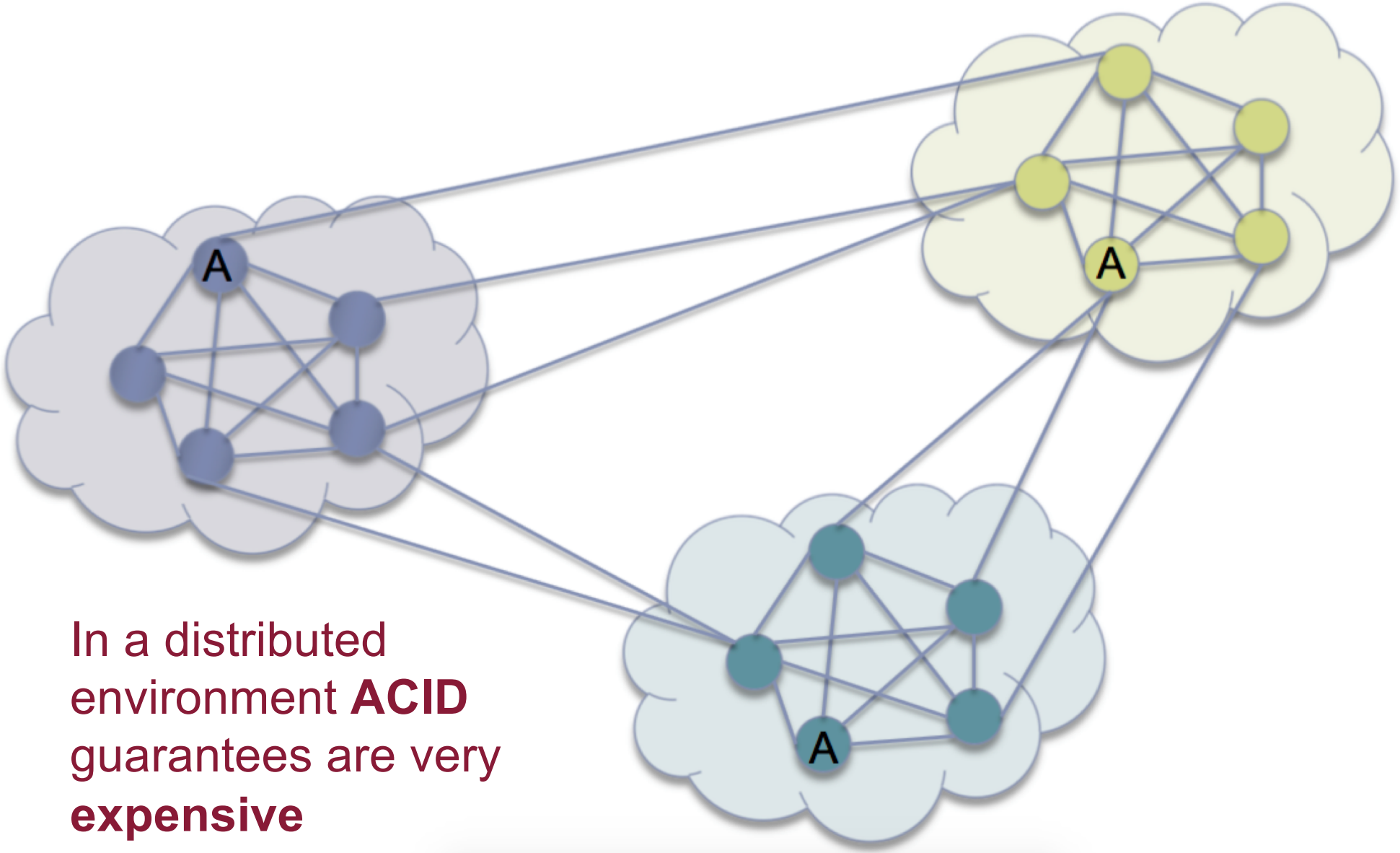
# Strong Consistency





# Today: network shared data systems

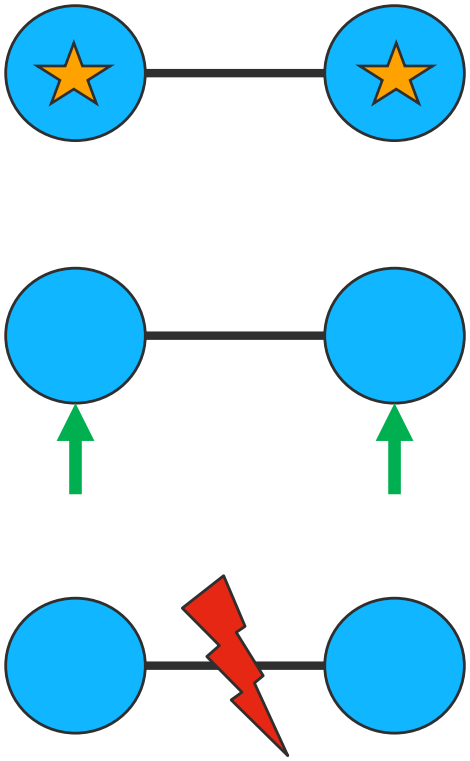





In a distributed environment **ACID** guarantees are very **expensive**

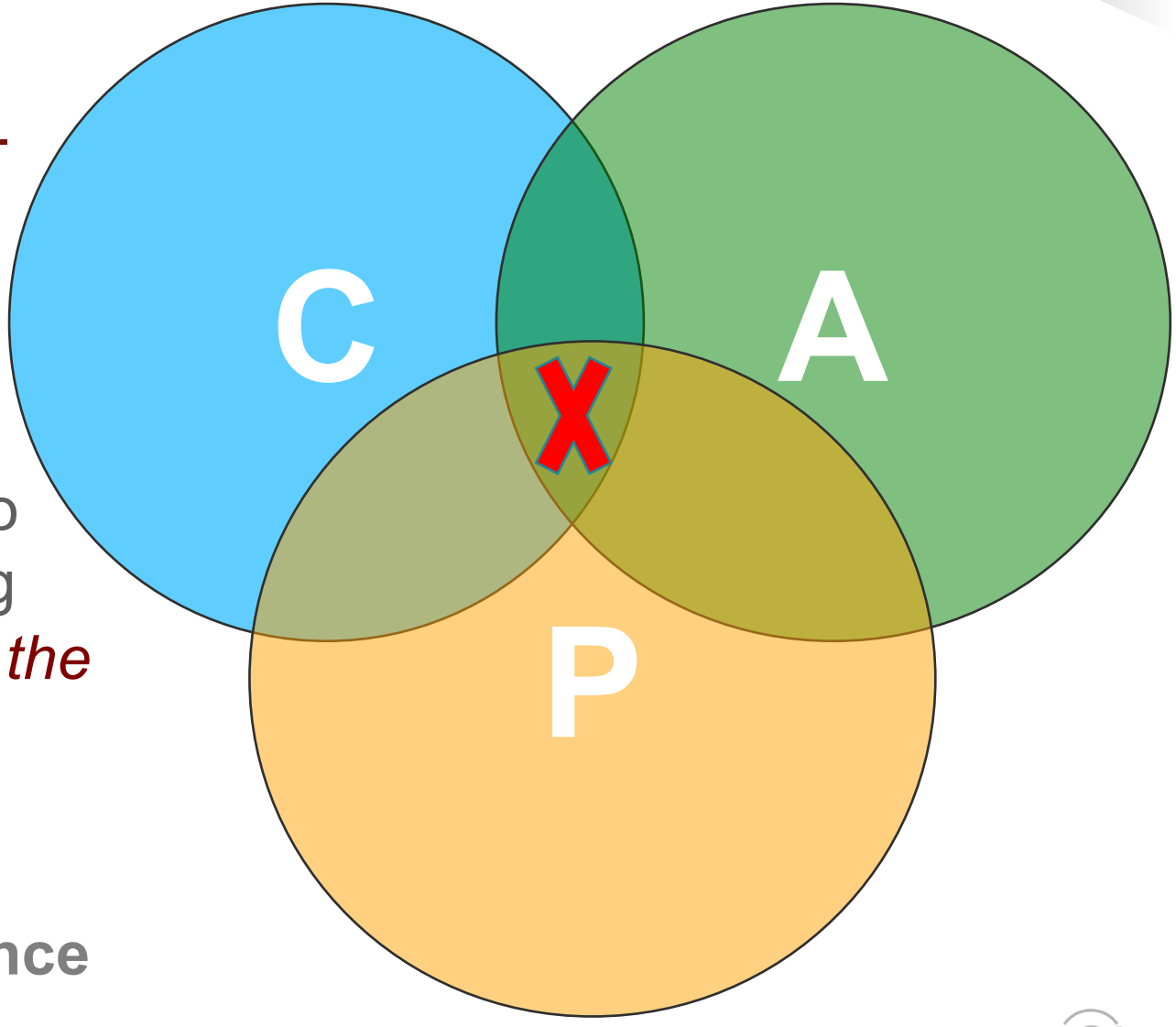
# Fundamental properties of distributed data management systems

- **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



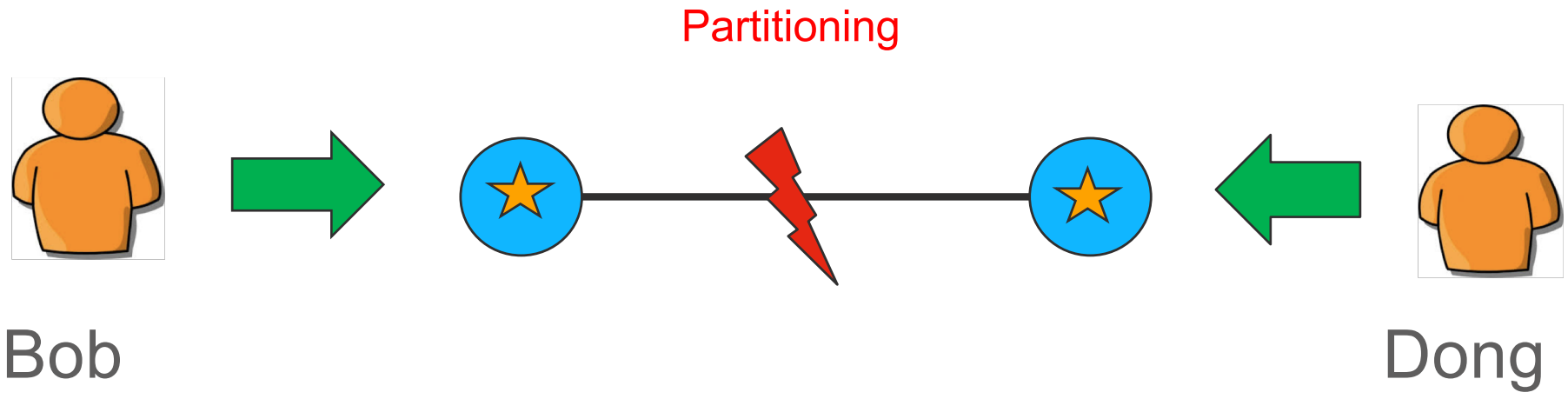

**Pick two !**

# CAP Theorem



A simple example (and an informal proof):

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



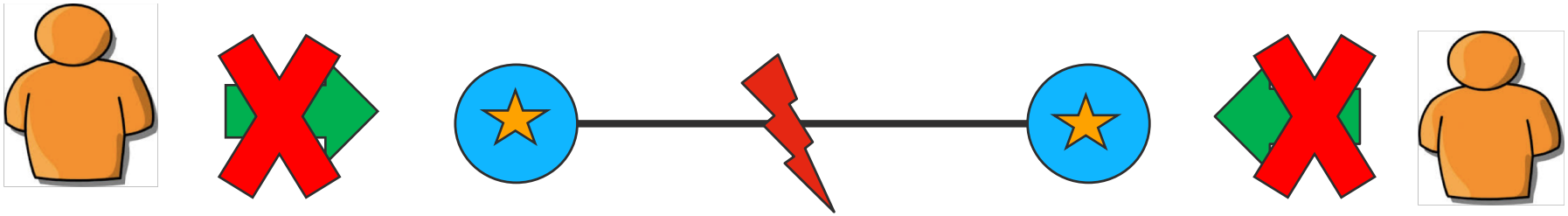
# CAP Theorem



A simple example (and an informal proof):

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

Partitioning



Bob Dong

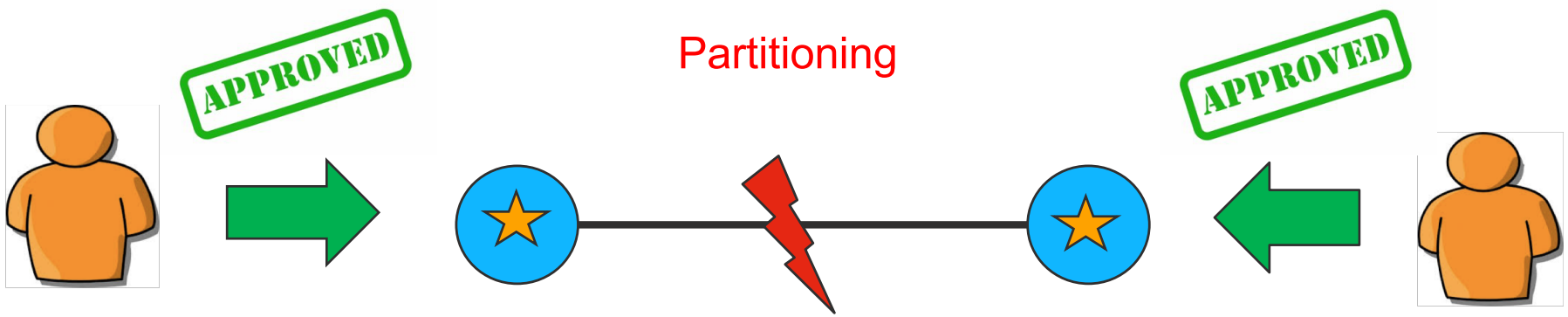
No booking: give up availability

# 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



Consistency

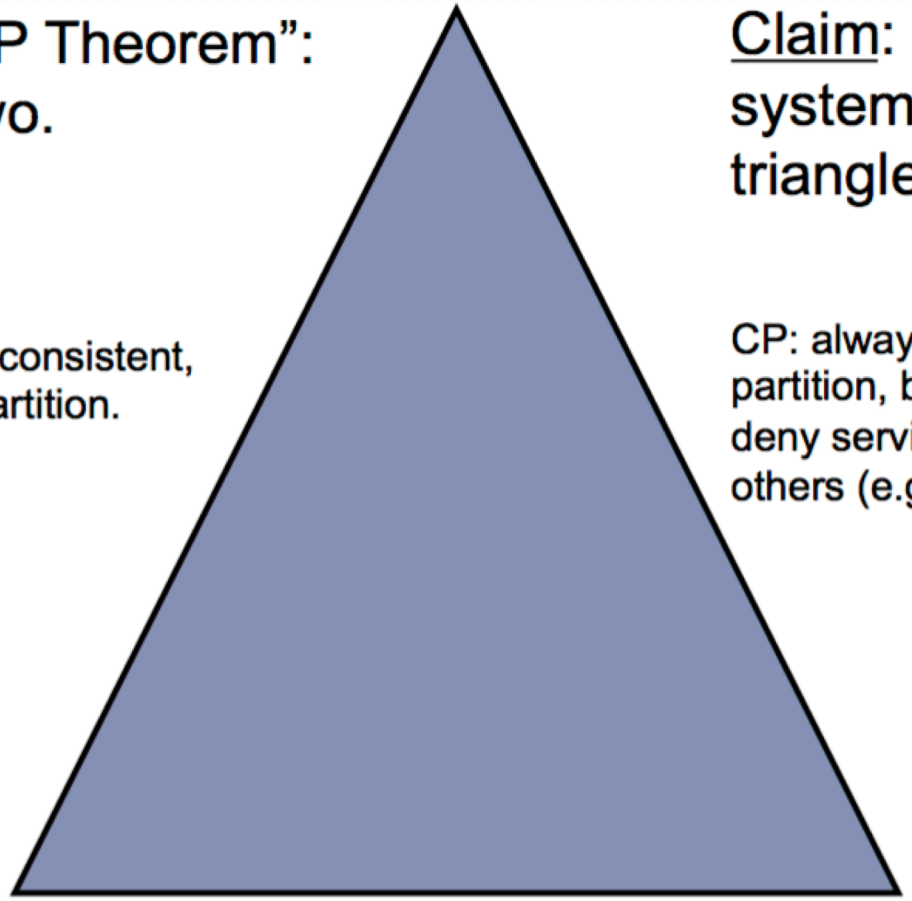
C

Fox&Brewer "CAP Theorem":
C-A-P: choose two.

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).



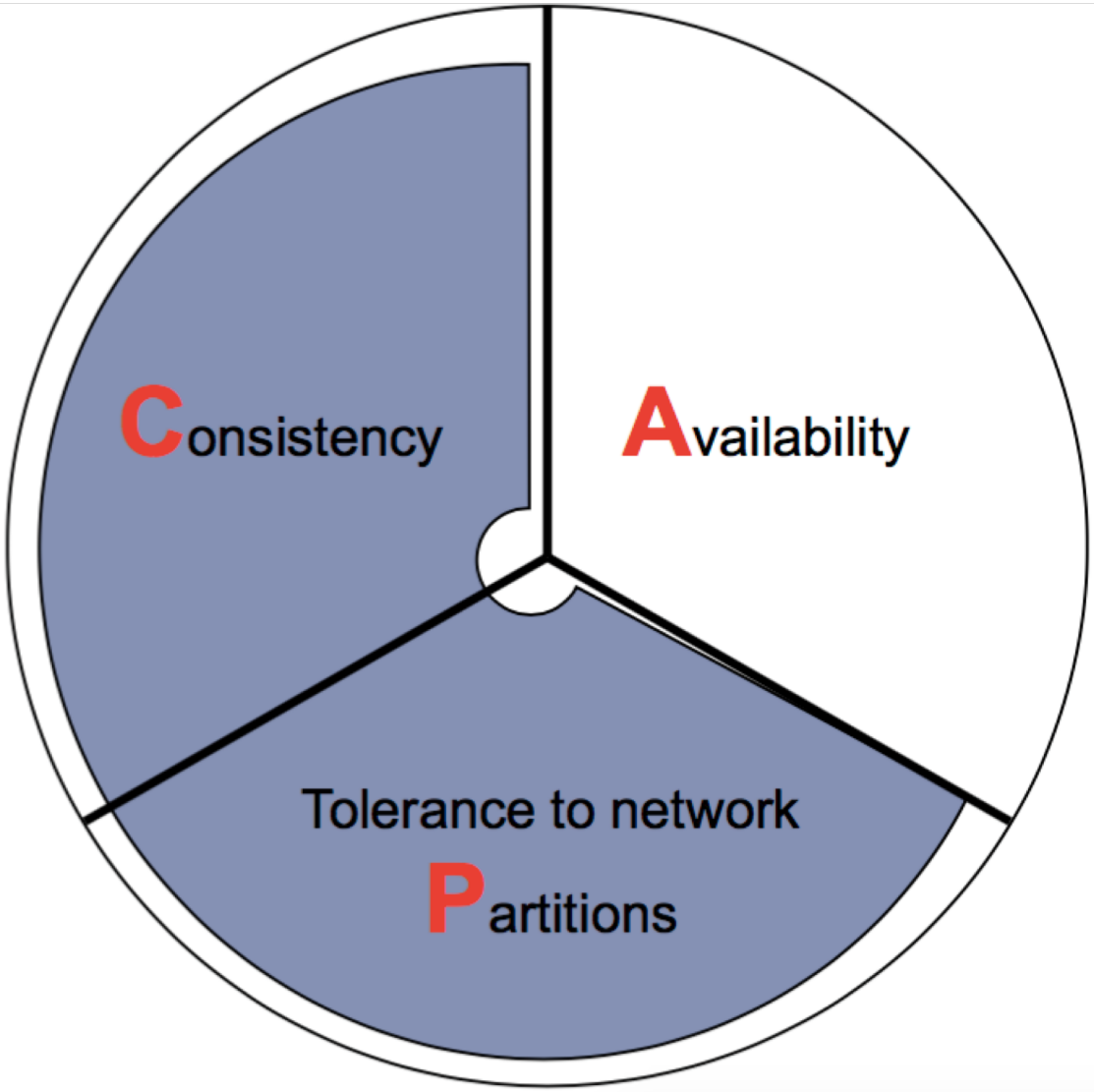
A
Availability

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

P
Partition-resilience

# Forfeit availability






mongoDB redis

## Examples

- Distributed databases
- Distributed locking
- Majority protocols

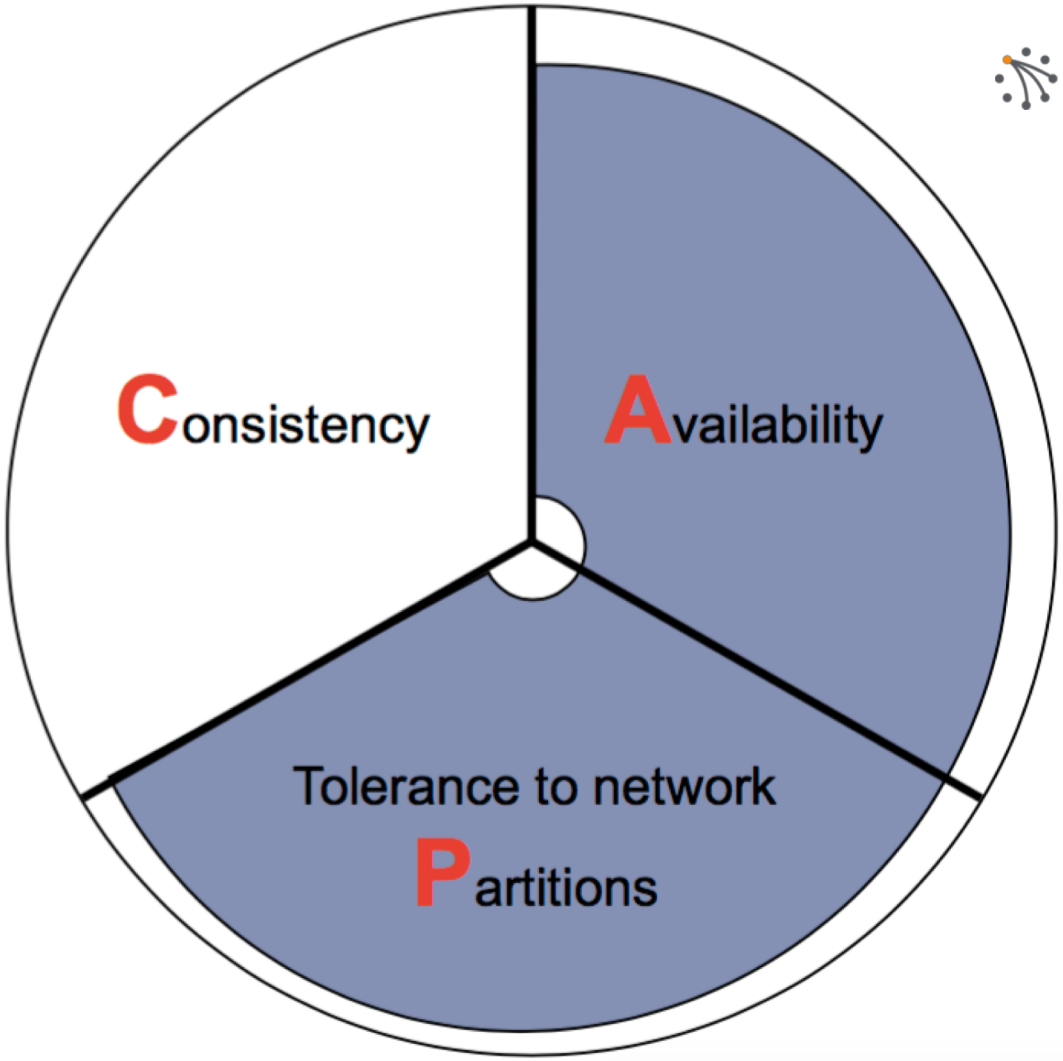
## Traits

- Pessimistic locking
- Make minority partitions unavailable

**Best effort availability**

# Forfeit consistency






riak

cassandra

CouchDB

**Examples**

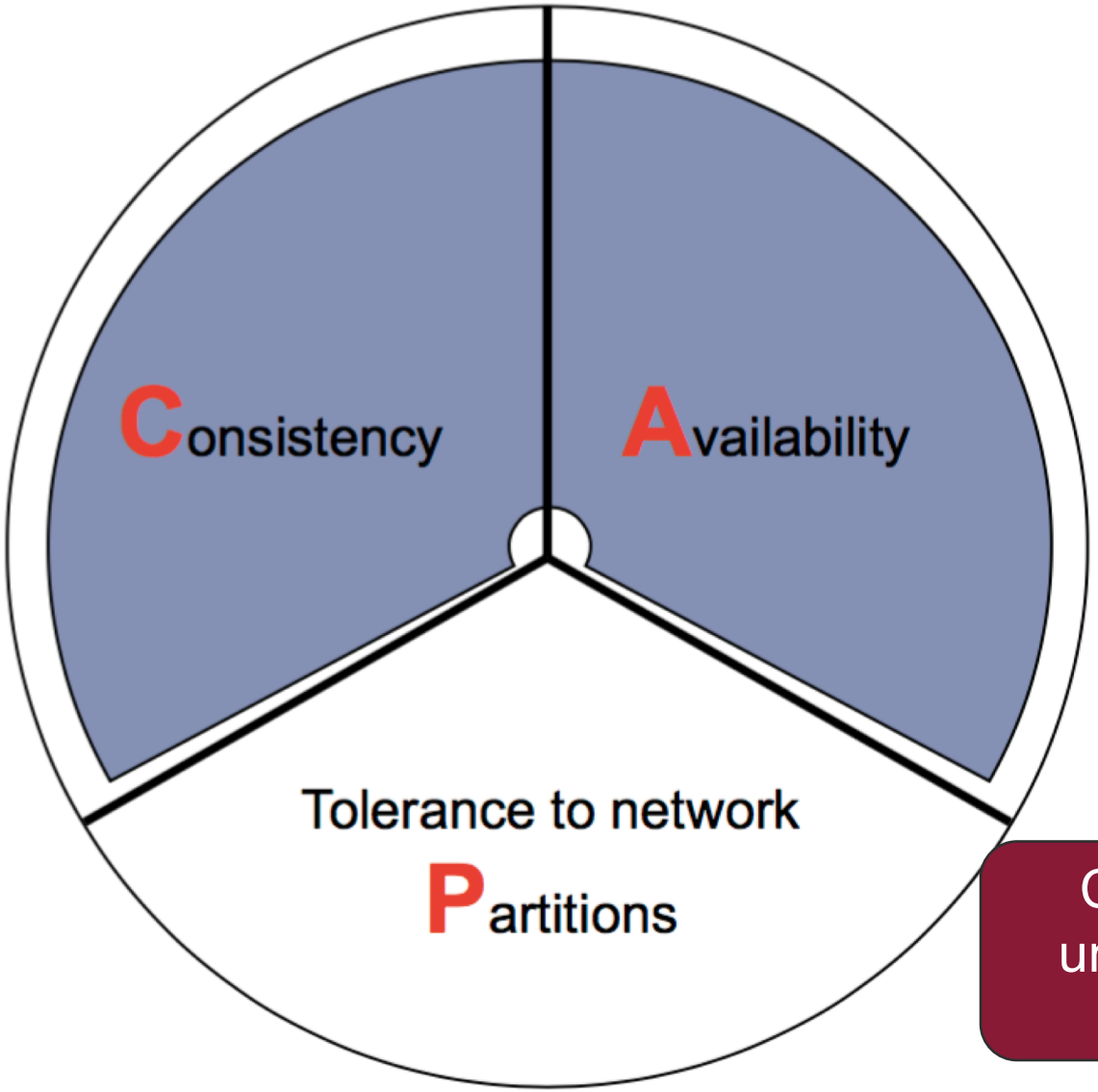
- Social networks
- Web caching
- DNS

**Traits**

- expirations/leases
- conflict resolution
- Optimistic

**Best effort consistency**

# Forfeit partitions




PostgreSQL

**Examples**

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

**Traits**

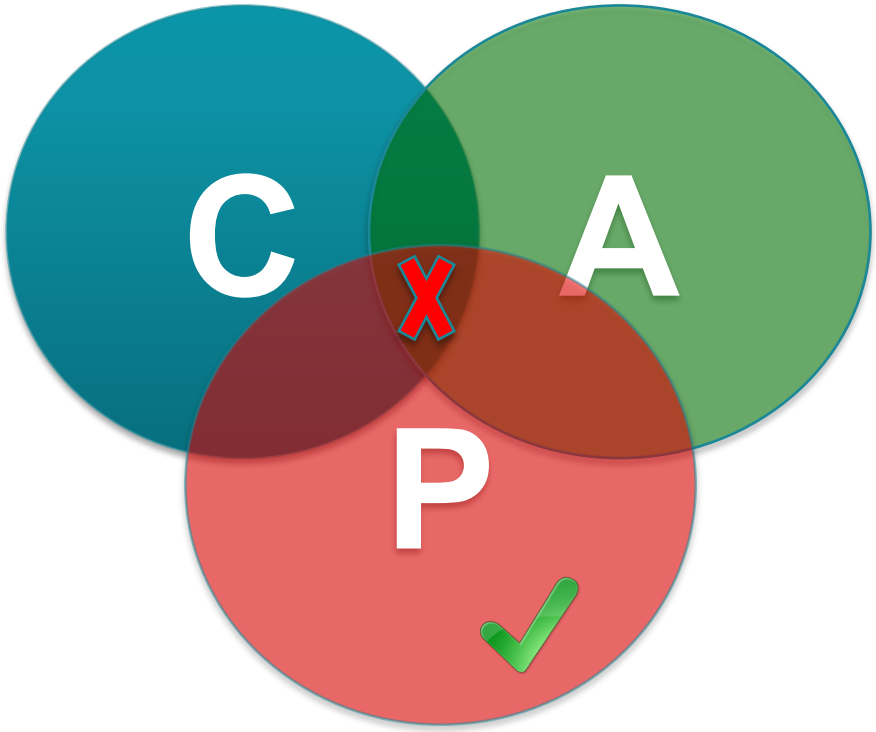
- 2-phase commit
- cache validation protocols

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

## Weak Consistency

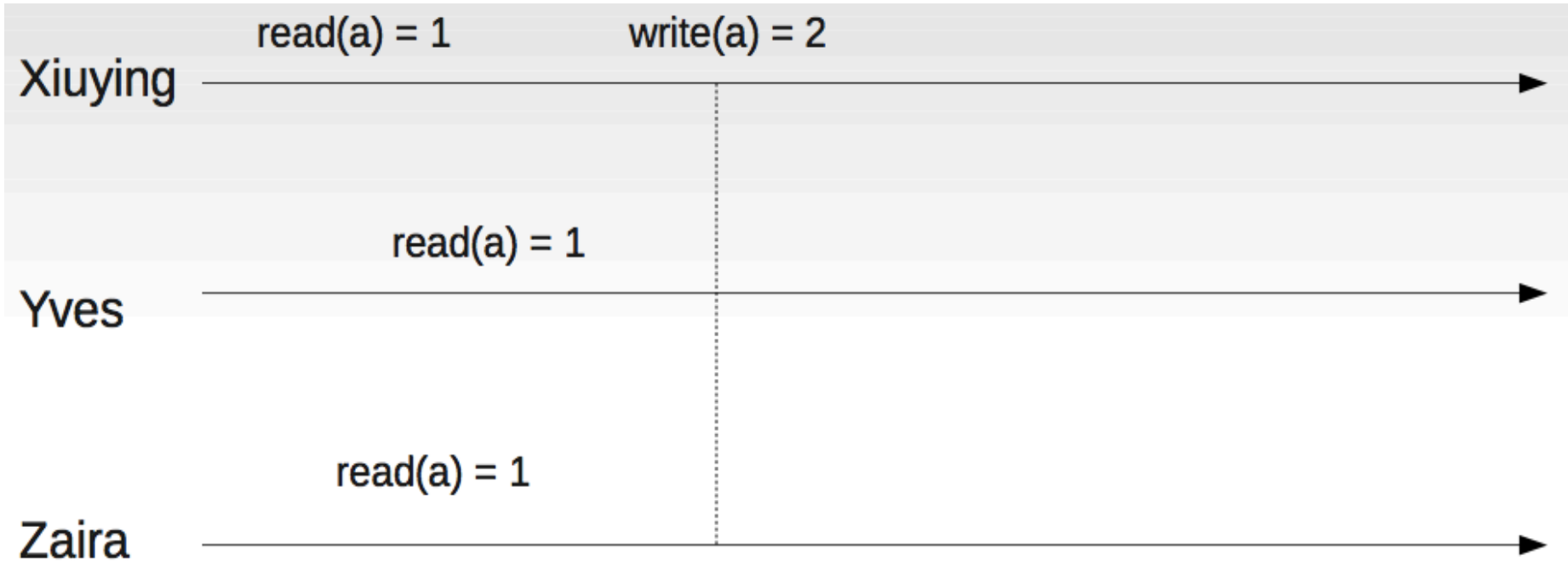
- It is not guaranteed that subsequent accesses will return the updated value

## Eventual Consistency

- Specific form of weak 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)

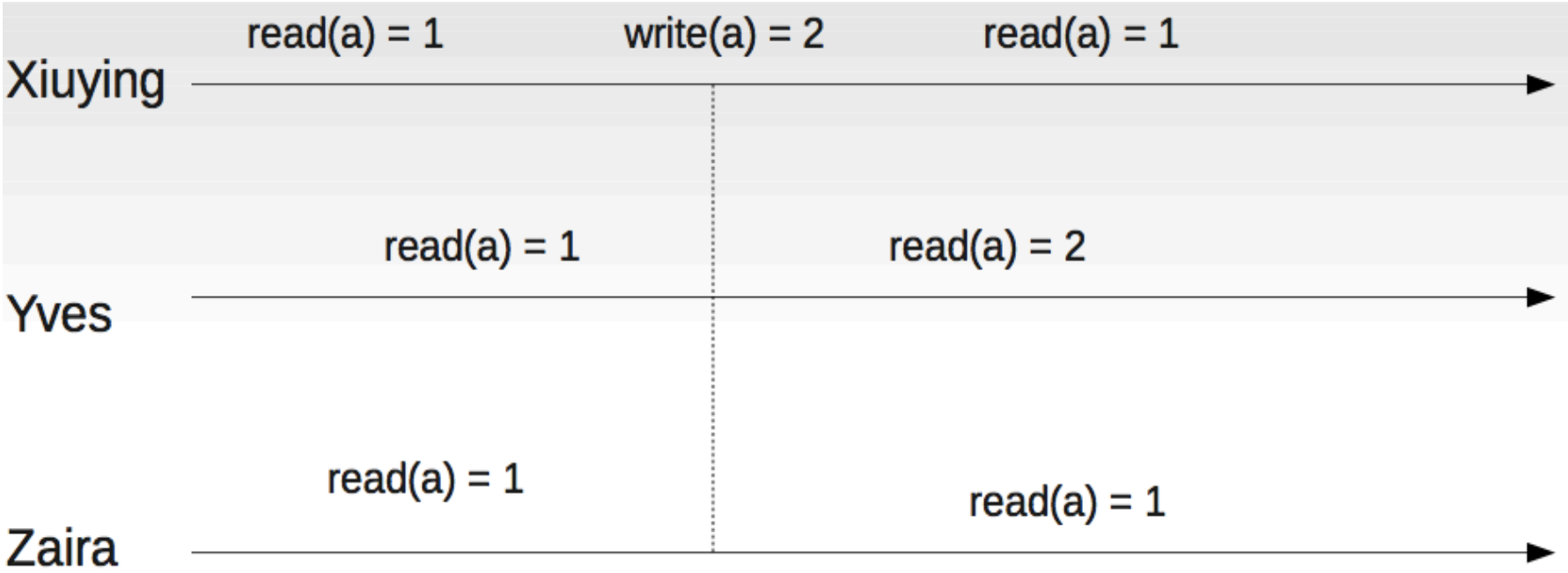
# Eventual Consistency





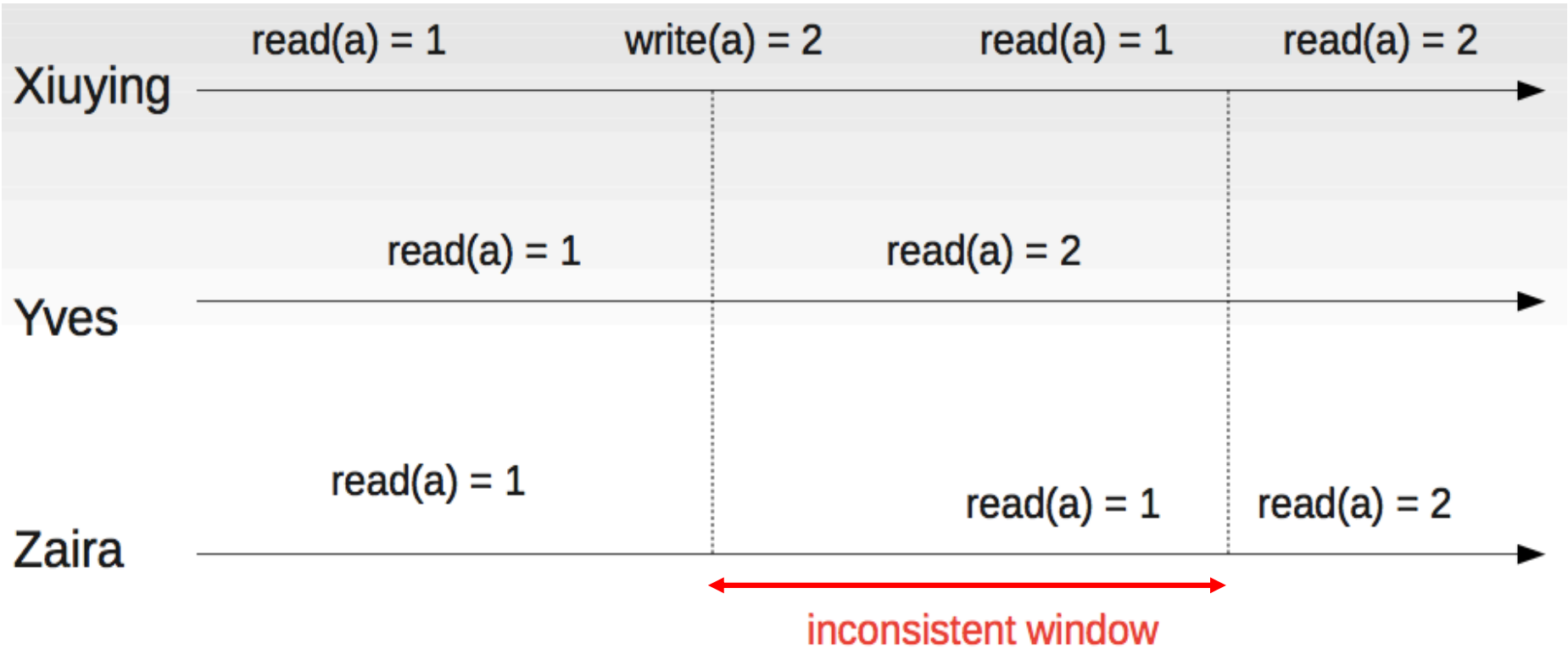
# Eventual Consistency





# Eventual Consistency





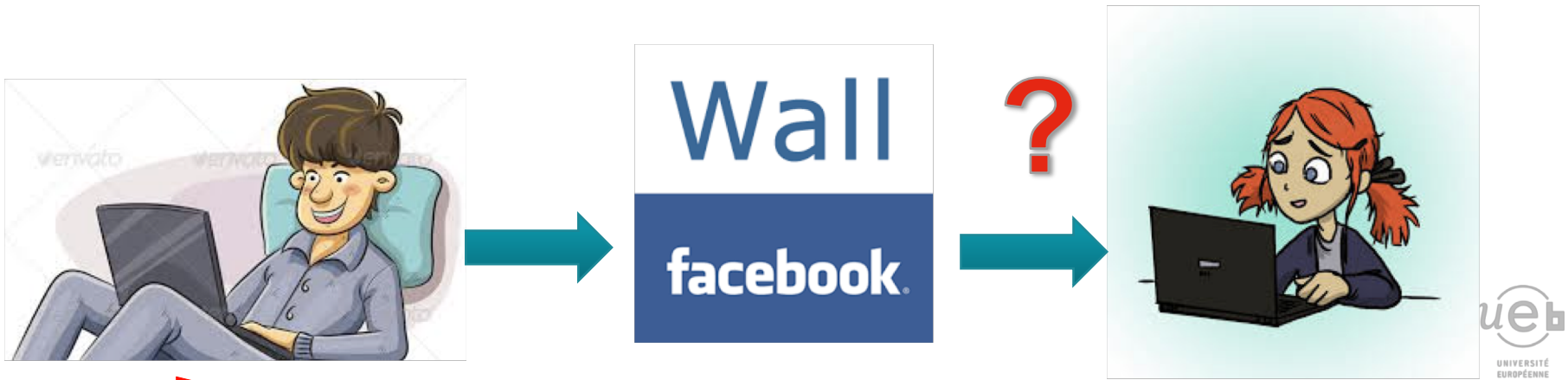
# Eventual Consistency



## 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!



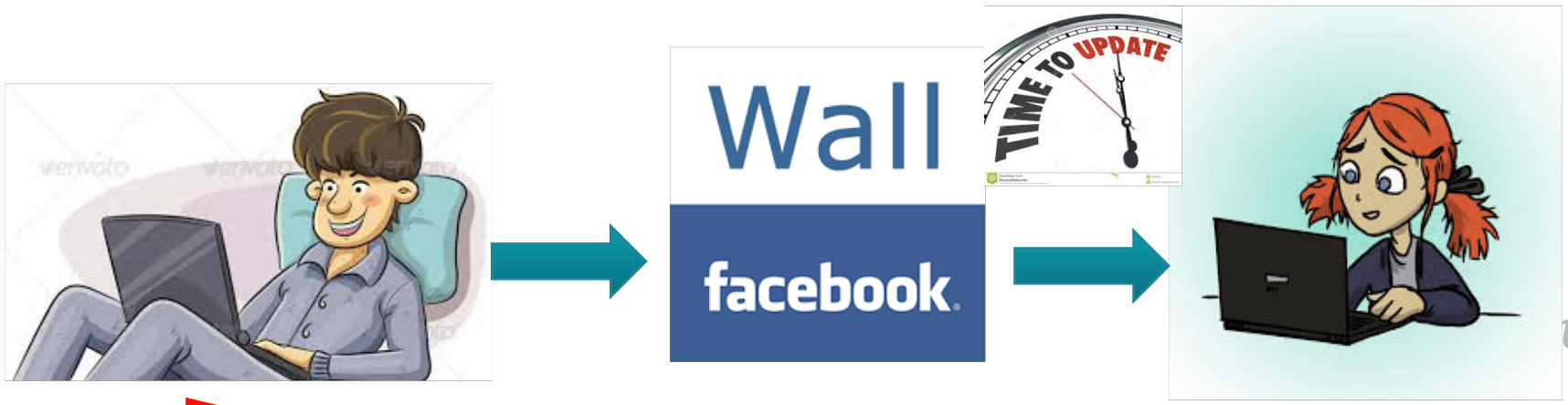
# Eventual Consistency



## Facebook example:

- Alice waits for a minute or so and checks back

She finds the story Bob shared with her!

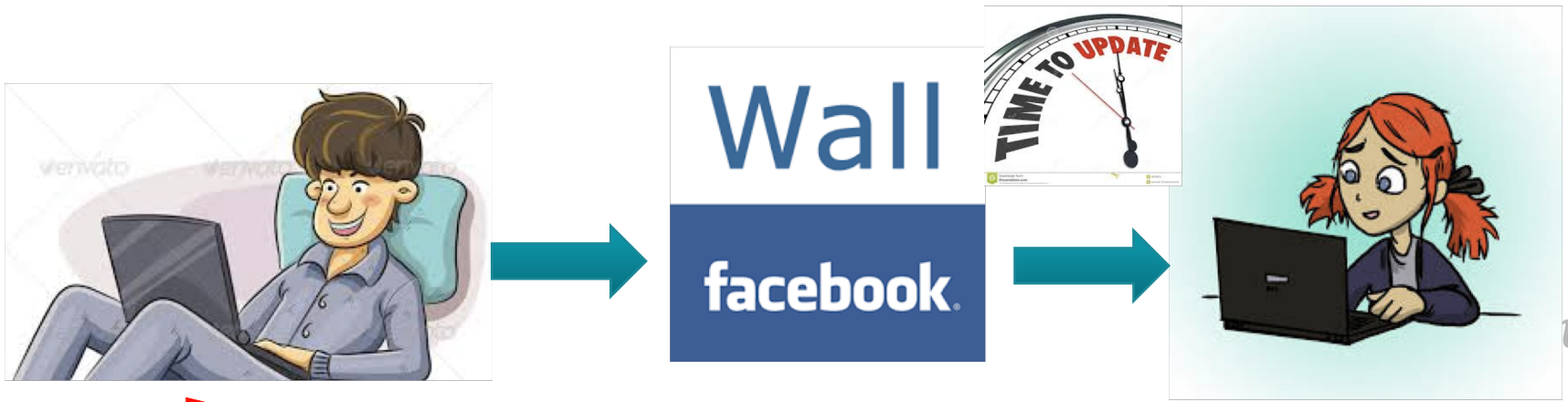


# Eventual Consistency



## 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

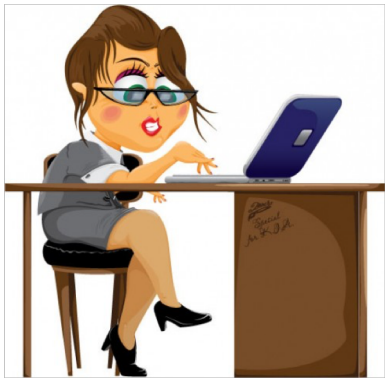


# Eventual Consistency



## Dropbox example:

- Dropbox enables strong consistency via synchronization in many cases
- However, what happens in case of a network partition?
- Dropbox embraces eventual consistency:
  - Strong consistency is impossible in case of a network partition
  - Users will feel bad if documents freeze due to the large latency to update all devices across WAN
  - Dropbox is oriented to personal synching, not on collaboration, so it is not a real limitation.



# Dynamic tradeoff between Consistency and Availability

## Airline reservation system:

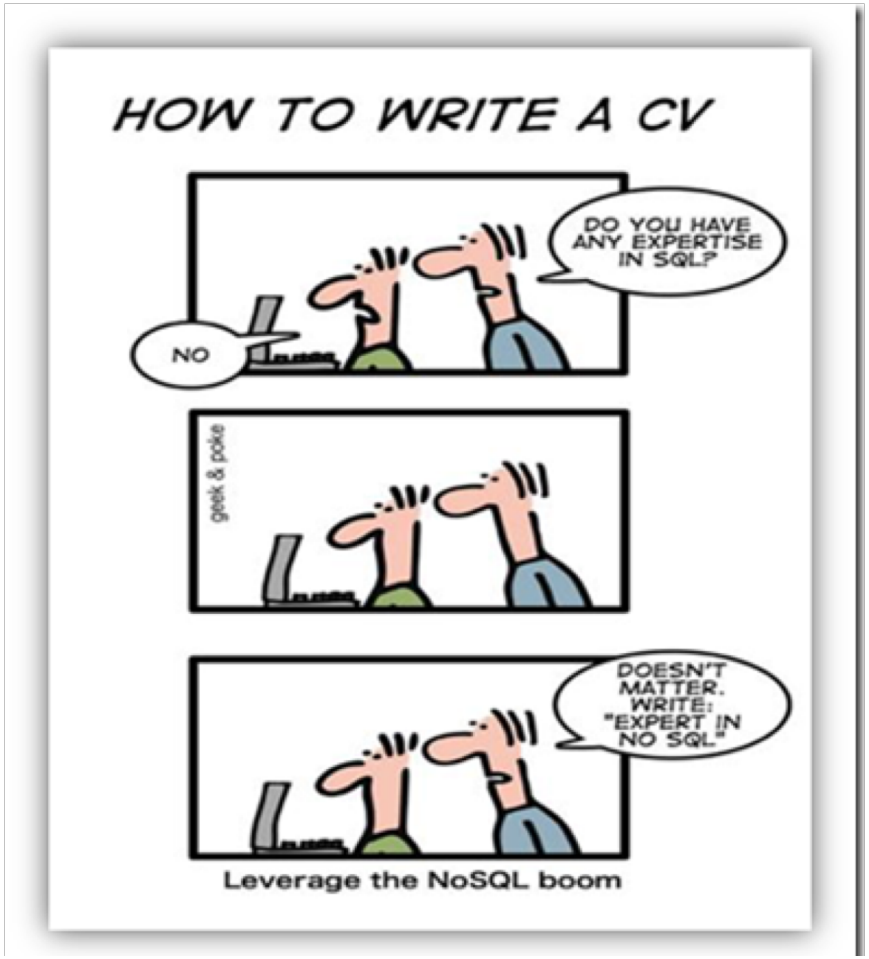
- When most of seats are available: it is ok to rely on somewhat out-of-date data, availability is more critical
- When the plane is close to be filled: it needs more accurate data to ensure the plane is not overbooked, consistency is more critical

# CAP Theorem and Big Data



BigData systems usually give up on (strong) consistency

Example:
**NoSQL**
(Not Only SQL)
databases



# 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

# Consistency vs. Latency tradeoff

**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

# Discussion

- In an e-commerce system (e.g., Amazon, e-Bay), what are the trade-offs between consistency and availability you can think of?

- Hint: things you might want to consider
  - Different types of data (e.g., shopping cart, billing, product)
  - Different types of operations (e.g., query, purchase)
  - Different groups of users (e.g., users in different geographic areas)