UNIVERSITY OF WATERLOO

Data-Intensive Distributed Computing

CS 431/631 451/651 (Fall 2019)

Part 7: Mutable State (2/2)

November 14, 2019

Ali Abedi

Motivating Scenarios

Money shouldn’t be created or destroyed:

Alice transfers $100 to Bob and $50 to Carol

The total amount of money after the transfer should be the same

Phantom shopping cart:

Bob removes an item from his shopping cart…

Item still remains in the shopping cart

Bob refreshes the page a couple of times… item finally gone

Motivating Scenarios

People you don’t want seeing your pictures:

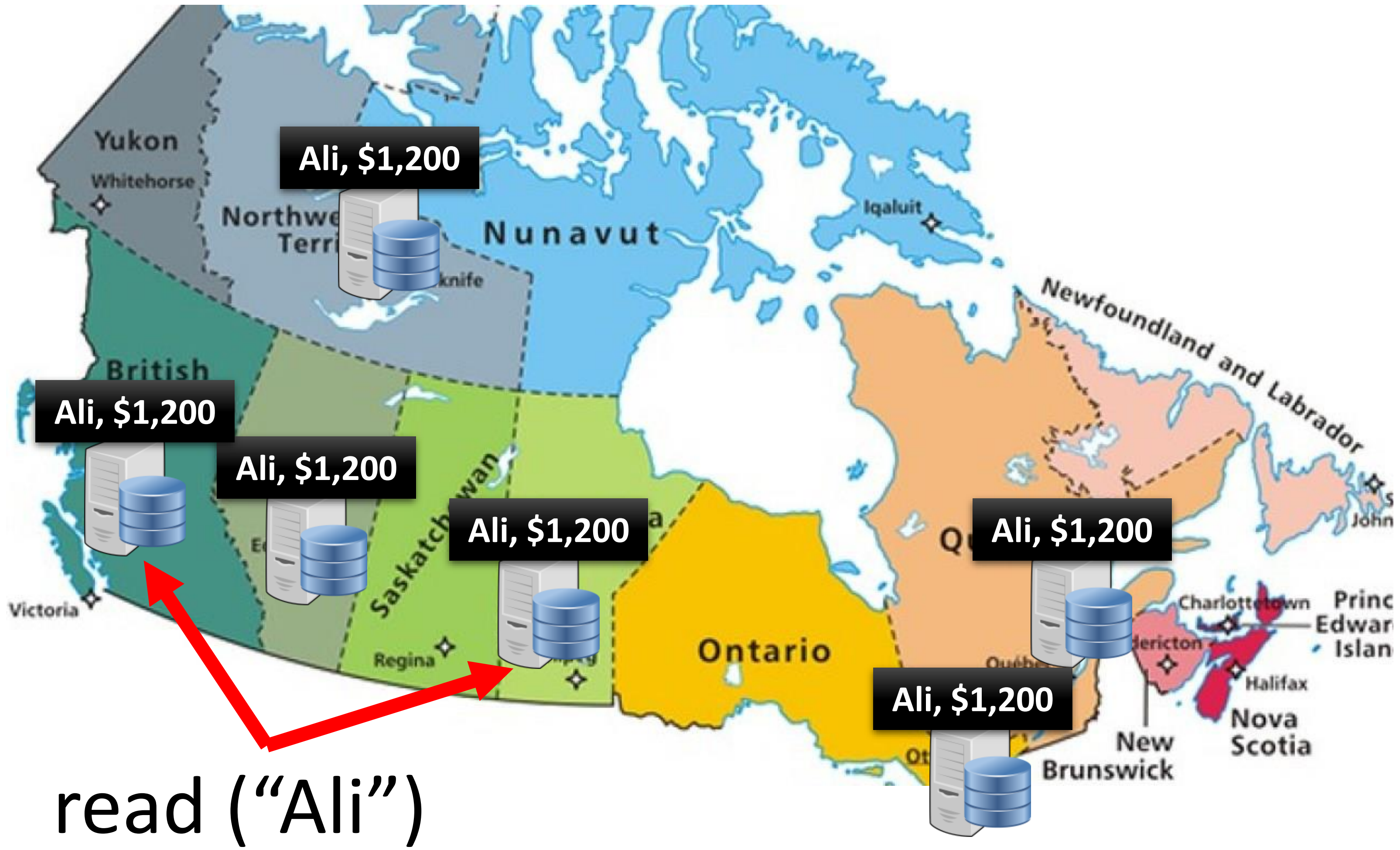
Alice removes mom from list of people who can view photos
Alice posts embarrassing pictures from Spring Break
Can mom see Alice’s photo?

Why am I still getting messages?

Bob unsubscribes from mailing list and receives confirmation
Message sent to mailing list right after unsubscribe
Does Bob receive the message?

Consistency

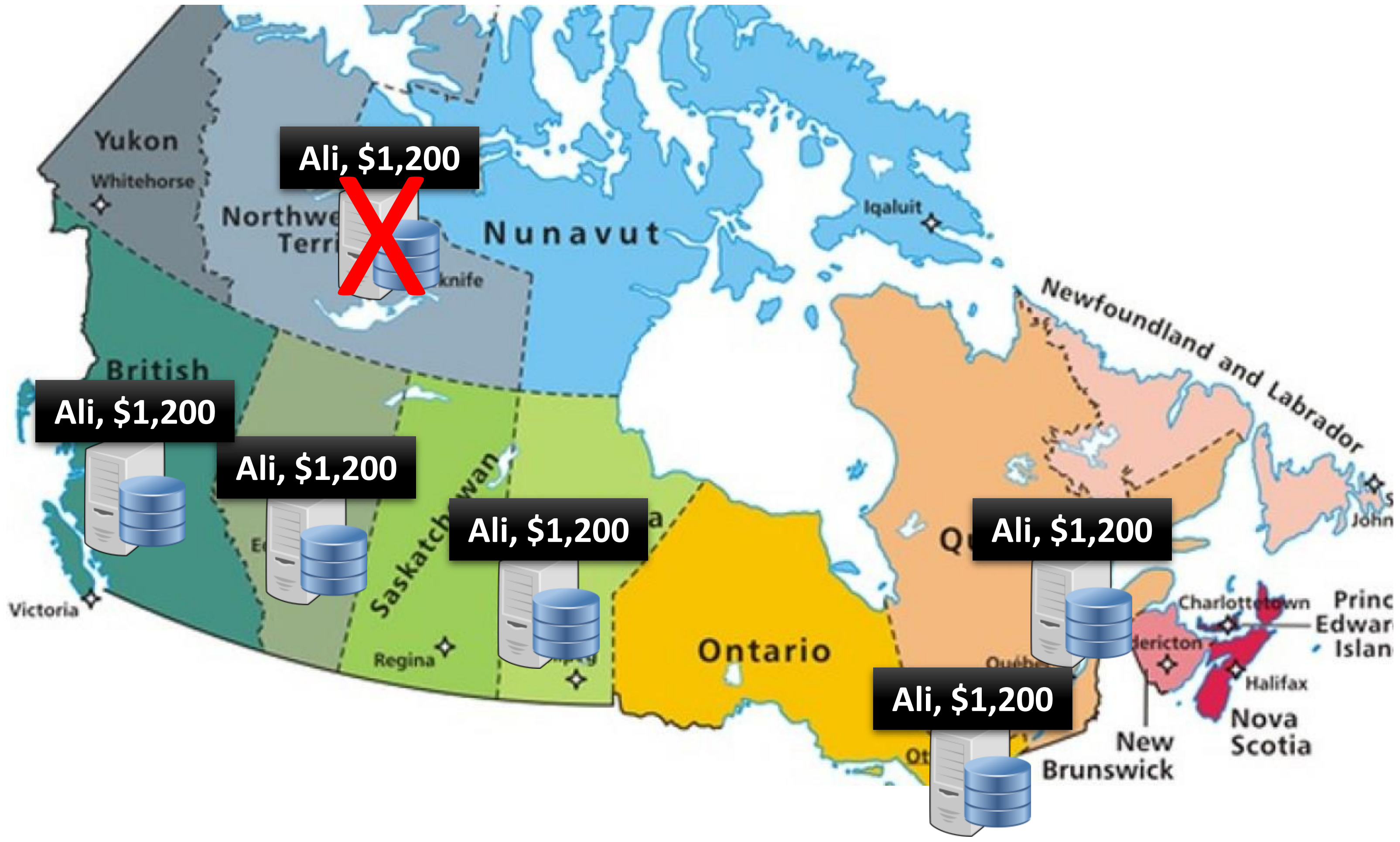
All nodes should see the same data at the same time



read ("Ali")

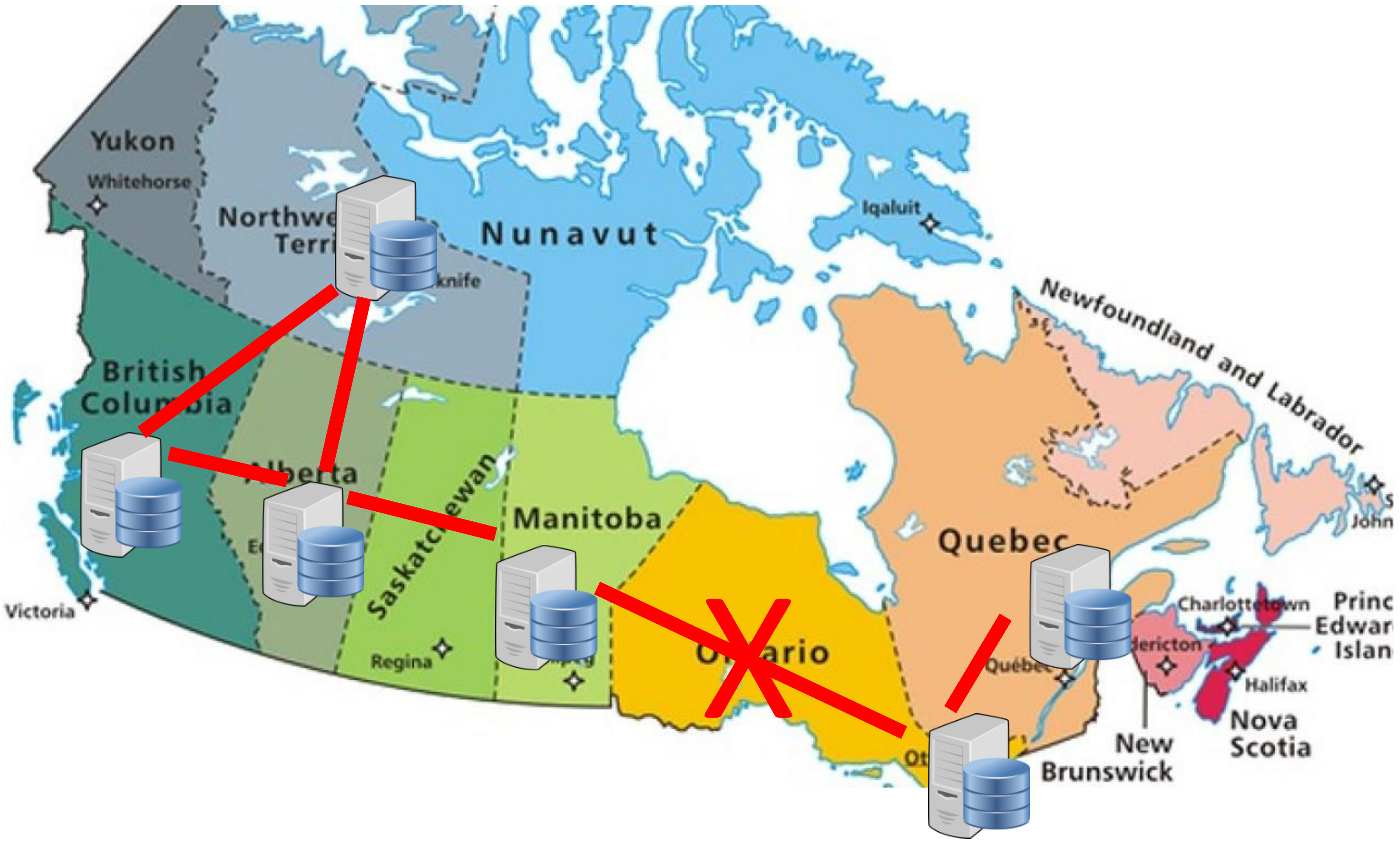
Availability

Node failures do not prevent survivors from continuing to operate



Partitioning-tolerance

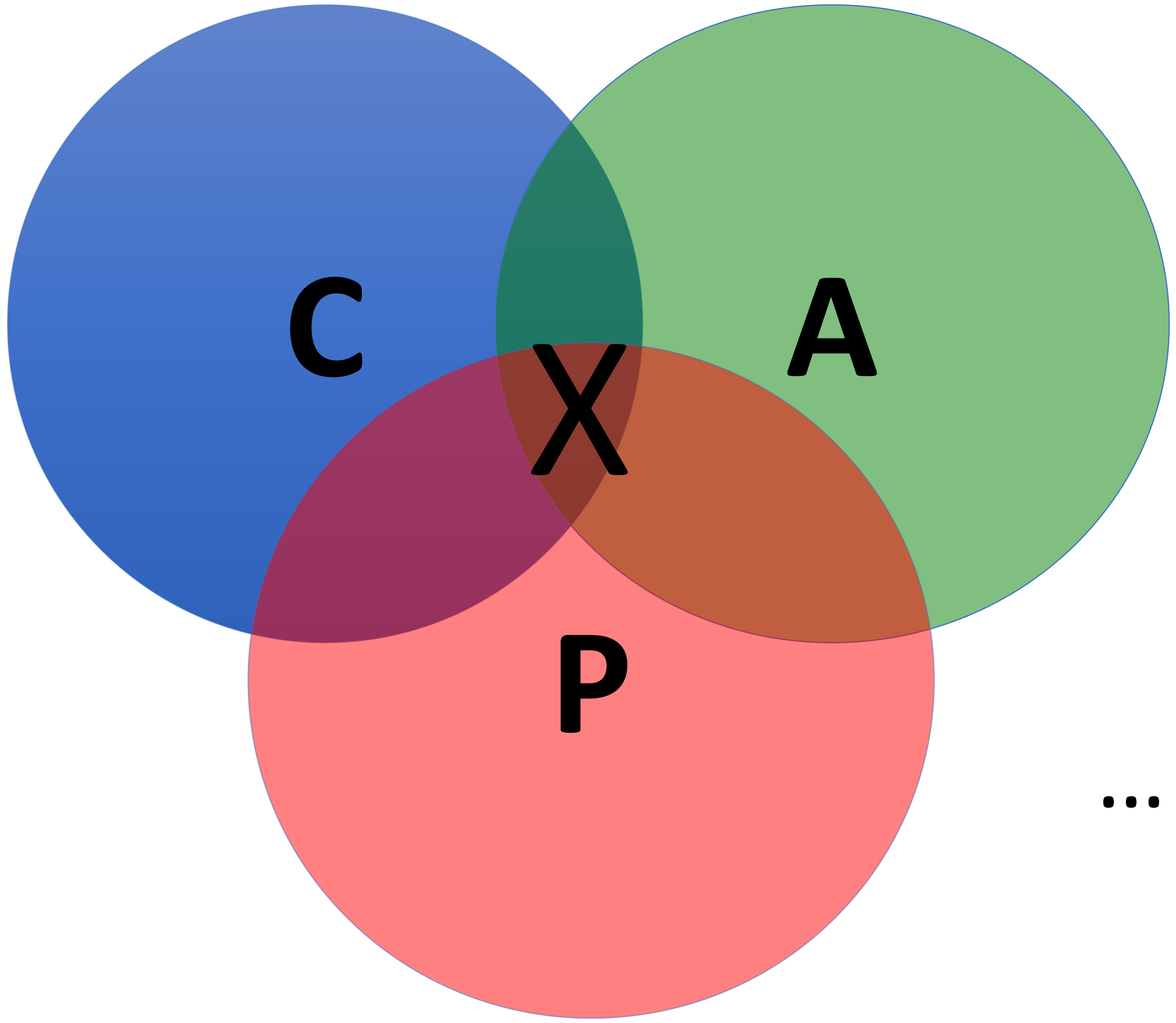
The system continues to operate despite network partitions



CAP Theorem

- **<u>C</u>**onsistency:
  - All nodes should see the same data at the same time
- **<u>A</u>**vailability:
  - Node failures do not prevent survivors from continuing to operate
- **<u>P</u>**artition-tolerance:
  - The system continues to operate despite network partitions
- A distributed system can satisfy any two of these guarantees at the same time **but not all three**

CAP Theorem



... pick two

CAP Theorem

- This suggests there are three kinds of distributed systems:
- CP: Big Table and Hbase
- AP: DNS
- CA → Impossible in distributed systems

CAP Theorem: Impact

Divides the database community (even today)

SQL
Correctness above all

Microsoft SQL Server

PostgreSQL

MySQL

NoSQL
Availability above all

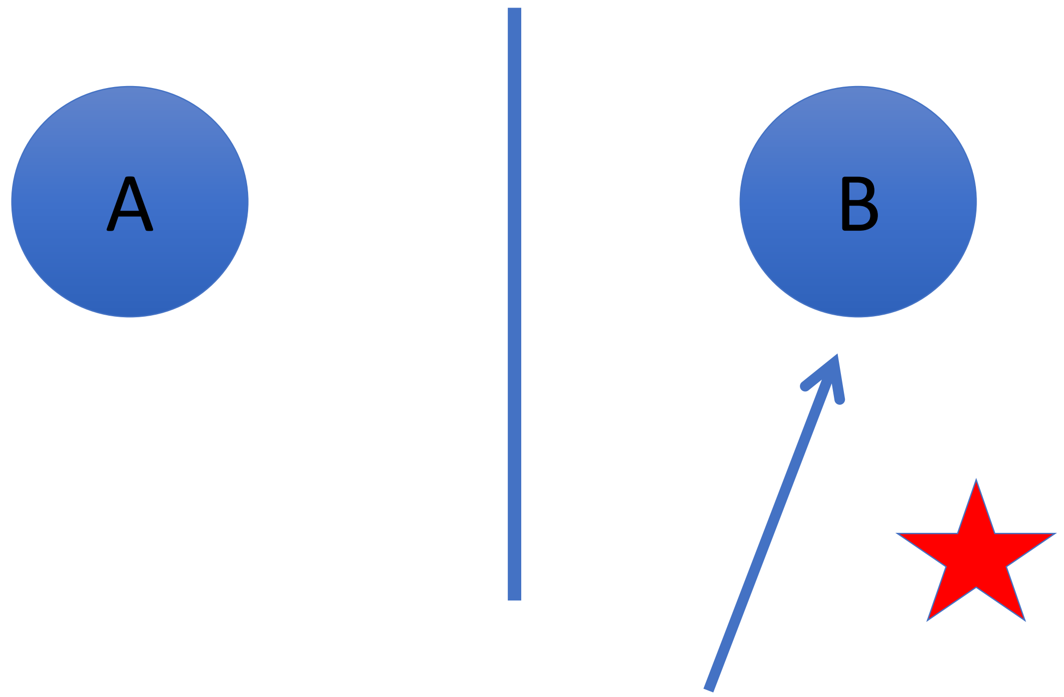
riak

redis

amazon DynamoDB

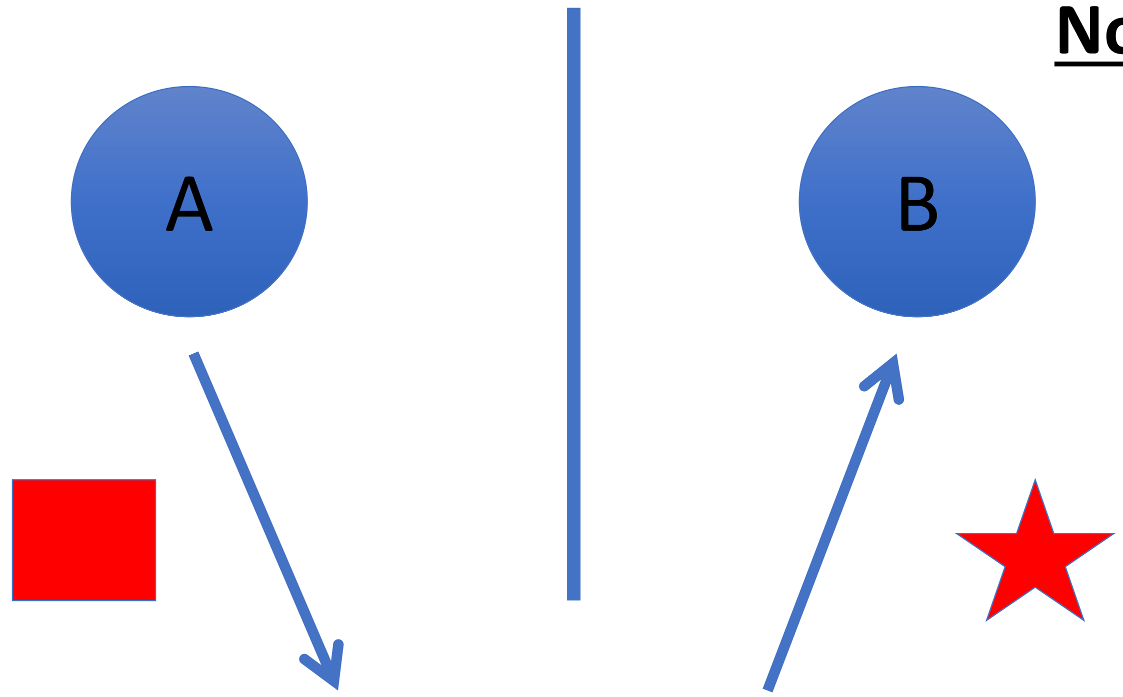
CAP Theorem: Proof

- A simple proof using two nodes:



CAP Theorem: Proof

- A simple proof using two nodes:

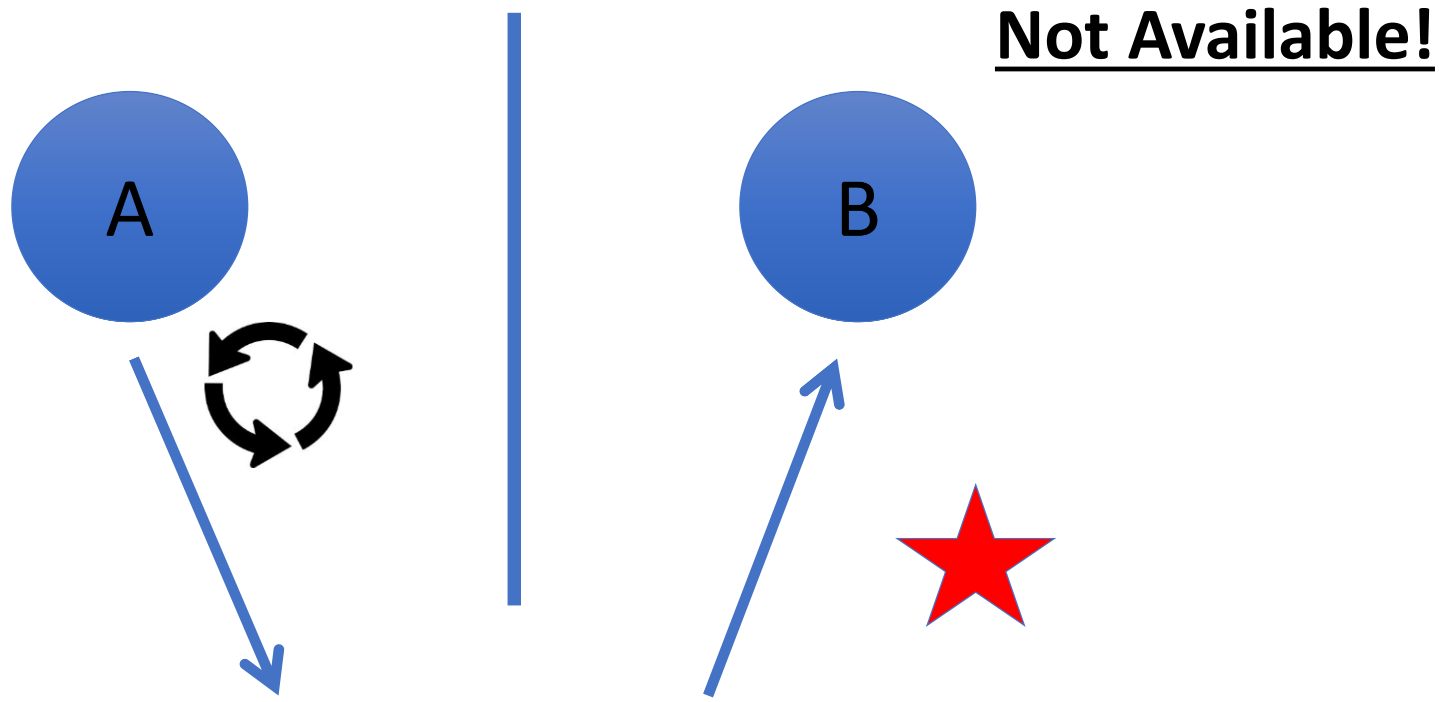


<u>Not Consistent!</u>

Respond to client

CAP Theorem: Proof

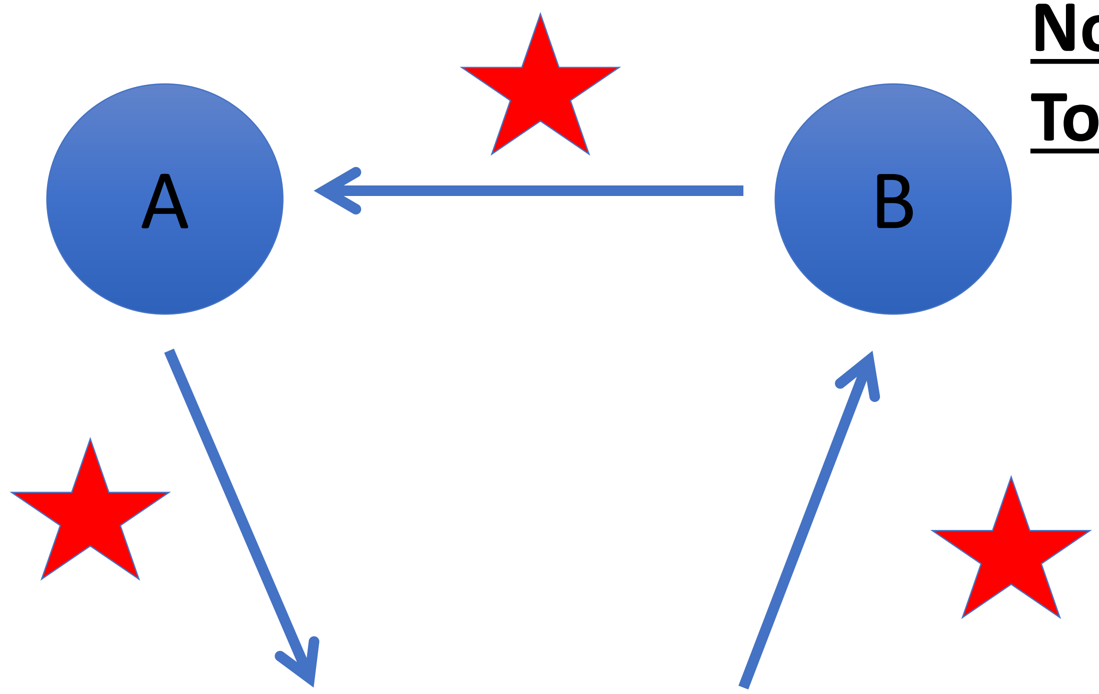
- A simple proof using two nodes:



Wait to be updated

CAP Theorem: Proof

- A simple proof using two nodes:



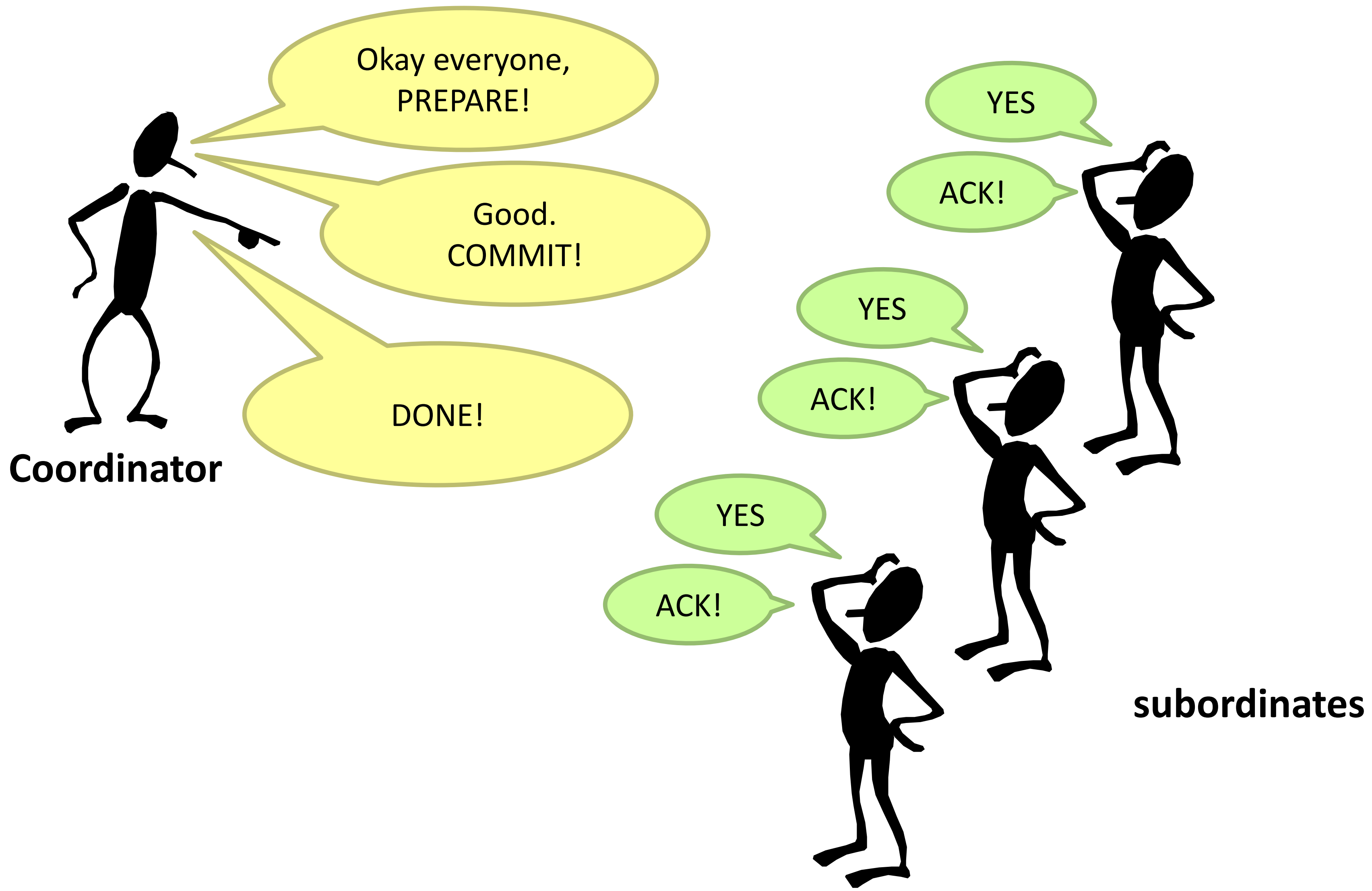
Not Partition Tolerant!

A gets updated from B

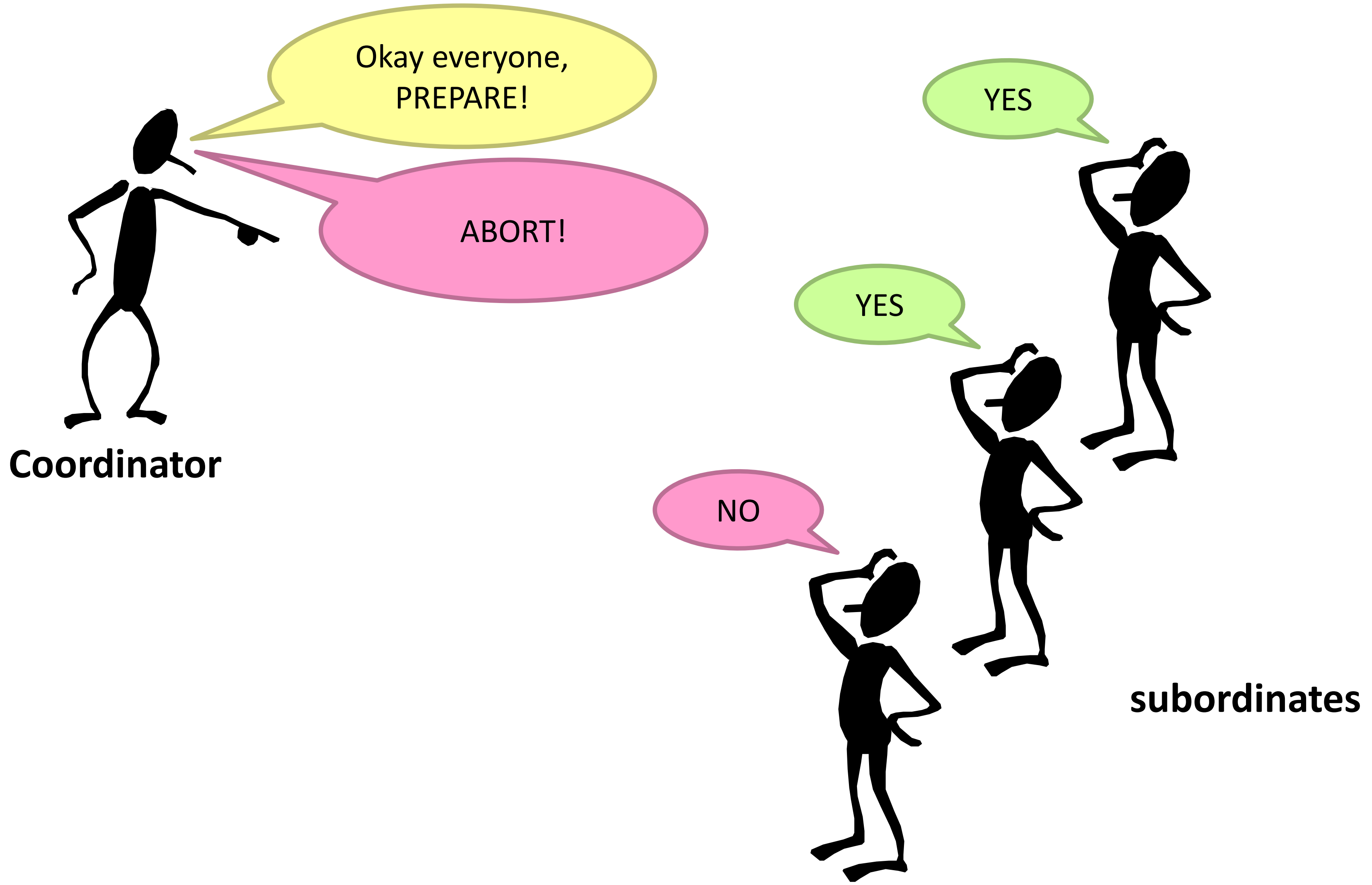
Types of Consistency

- Strong Consistency
  - After the update completes, **any subsequent access** will return the **same** updated value.

2 Phase Commit: Sketch



2 Phase Commit: Sketch



2PC: Assumptions and Limitations

Assumptions:

Persistent storage and write-ahead log at every node

WAL is never permanently lost

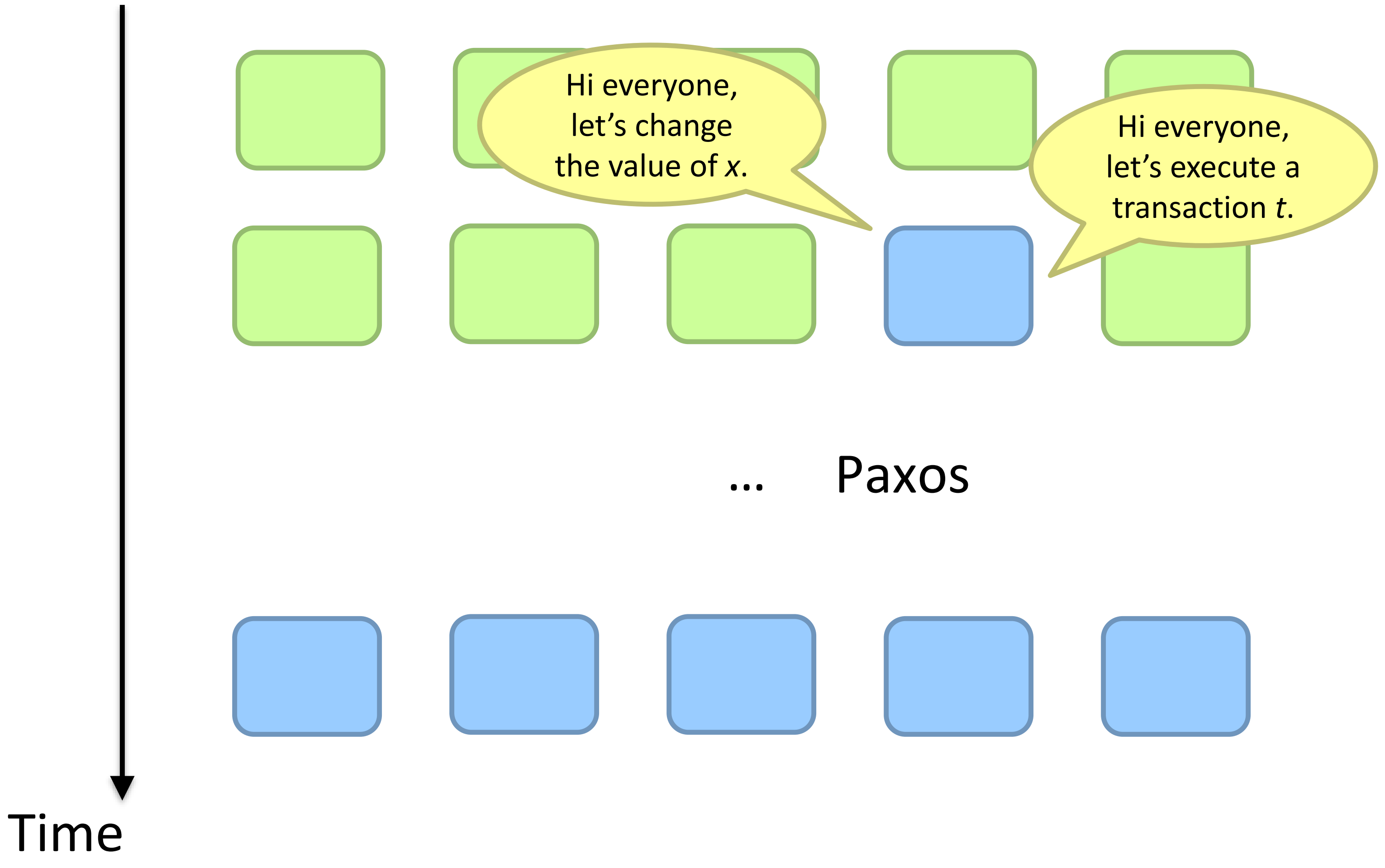
Limitations:

It’s blocking and slow

What if the coordinator dies?

Distributed Consensus

More general problem: addresses replication and partitioning


Hi everyone, let's change the value of *x*.
Hi everyone, let's execute a transaction *t*.
…
Paxos
Time

Types 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*)

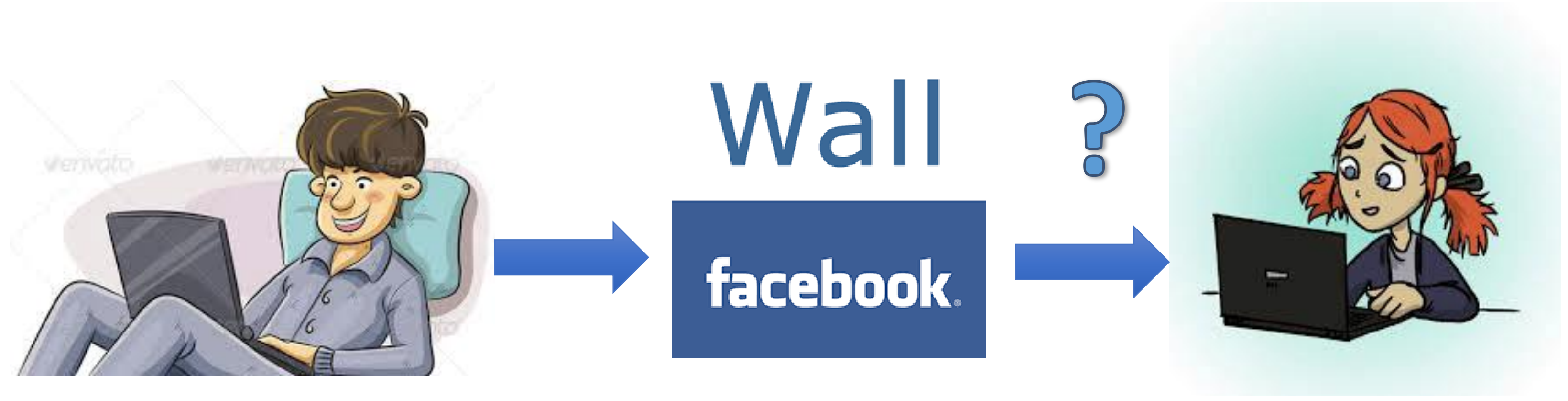
Eventual Consistency - An ATM Example

- In design of automated teller machine (ATM):
  - Strong consistency appear to be a nature choice
  - However, in practice, **A beats C**
  - Higher availability means **higher revenue**
  - ATM will allow you to withdraw money *even if the machine is partitioned from the network*
  - However, it puts **a limit** on the amount of withdraw (e.g., $200)
  - The bank might also charge you a fee when a overdraft happens

Eventual Consistency
- A Facebook Example

- Bob finds an interesting story and shares 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:

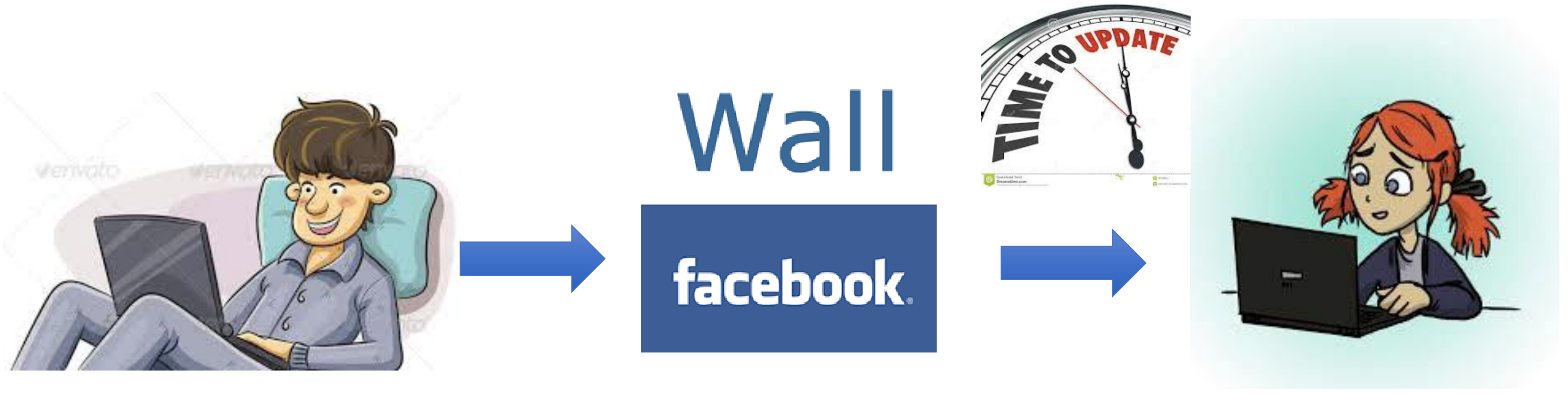
    - **Nothing is there!**



Eventual Consistency
- A Facebook Example

- Bob tells Alice to wait a bit and check out later
- Alice waits for a minute or so and checks back:

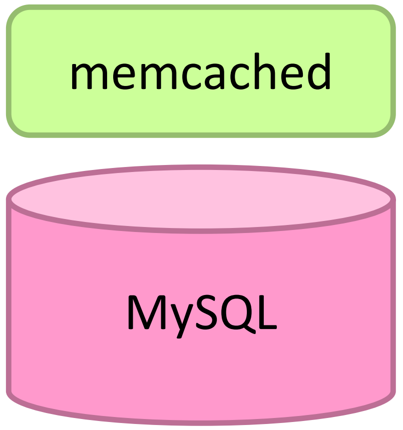
    - **She finds the story Bob shared with her!**



Eventual Consistency
- A Facebook Example

- Reason: it is possible because Facebook uses an **eventual consistent model**
- Why Facebook chooses eventual consistent model over the strong consistent one?
  - 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**

Facebook Architecture



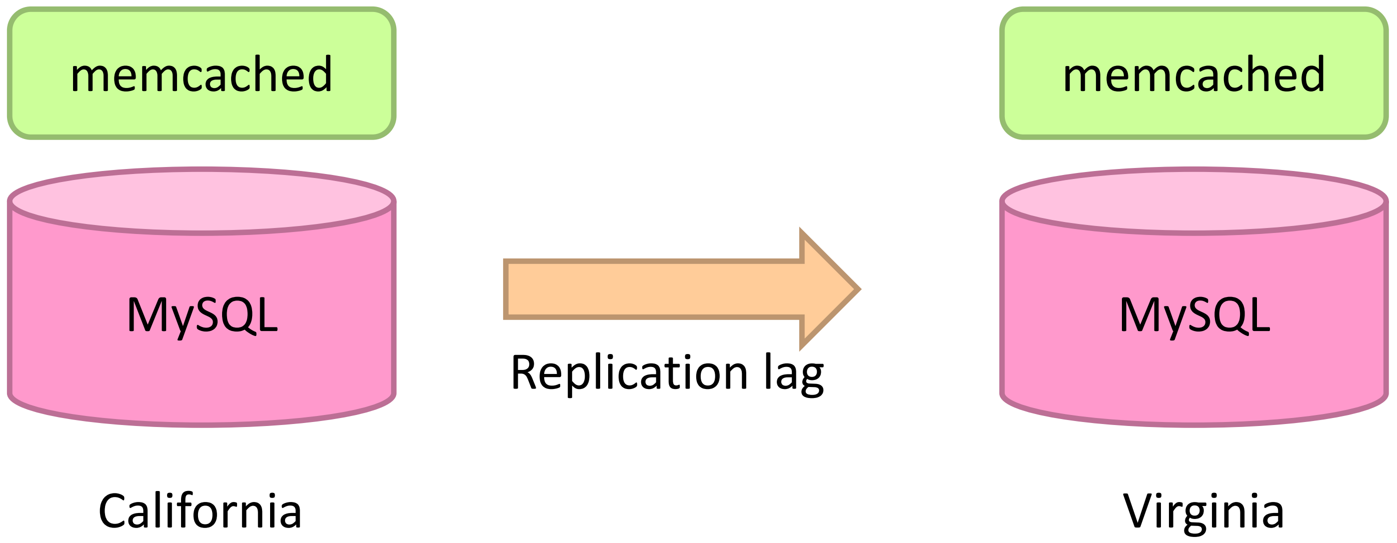
Read path:
Look in memcached
Look in MySQL
Populate in memcached

Write path:
Write in MySQL
Remove in memcached

Subsequent read:
Look in MySQL
Populate in memcached

Source: www.facebook.com/note.php?note_id=23844338919

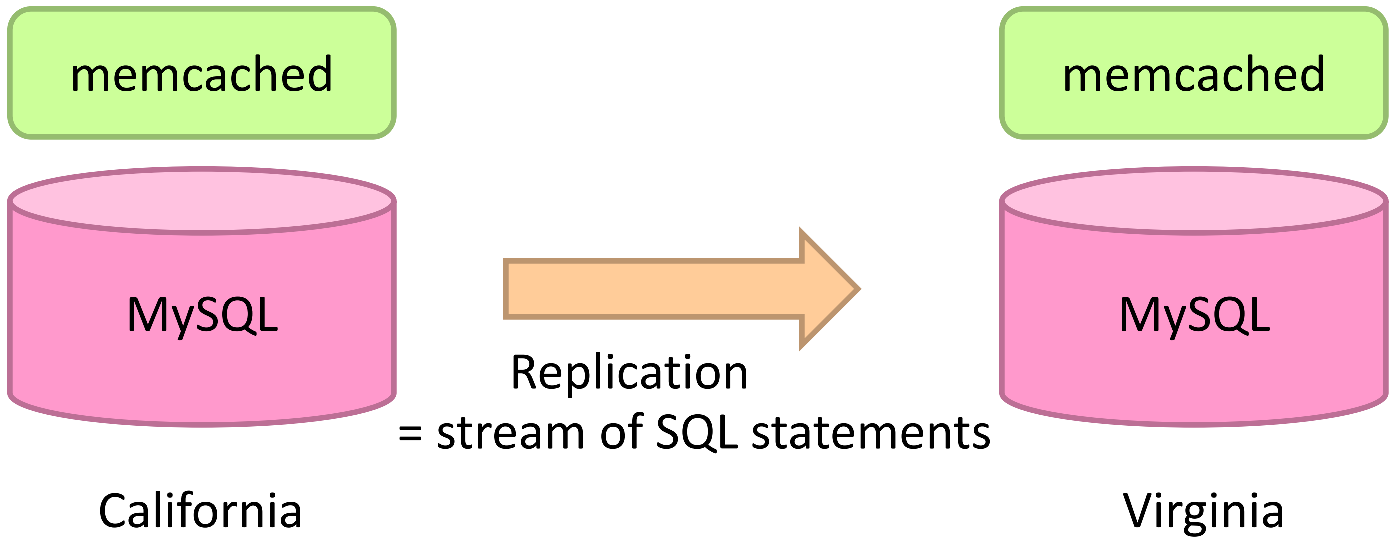
Facebook Architecture: Multi-DC



1. User updates first name from “Jason” to “Monkey”.
2. Write “Monkey” in master DB in CA, delete memcached entry in CA and VA.
3. Someone goes to profile in Virginia, read VA replica DB, get “Jason”.
4. Update VA memcache with first name as “Jason”.
5. Replication catches up. “Jason” stuck in memcached until another write!

Source: www.facebook.com/note.php?note_id=23844338919

Facebook Architecture: Multi-DC



Solution: Piggyback on replication stream, tweak SQL

```
REPLACE INTO profile (`first_name`) VALUES ('Monkey’)
WHERE `user_id`='jsobel' MEMCACHE_DIRTY 'jsobel:first_name'
```

Source: www.facebook.com/note.php?note_id=23844338919

What if there are no partitions?

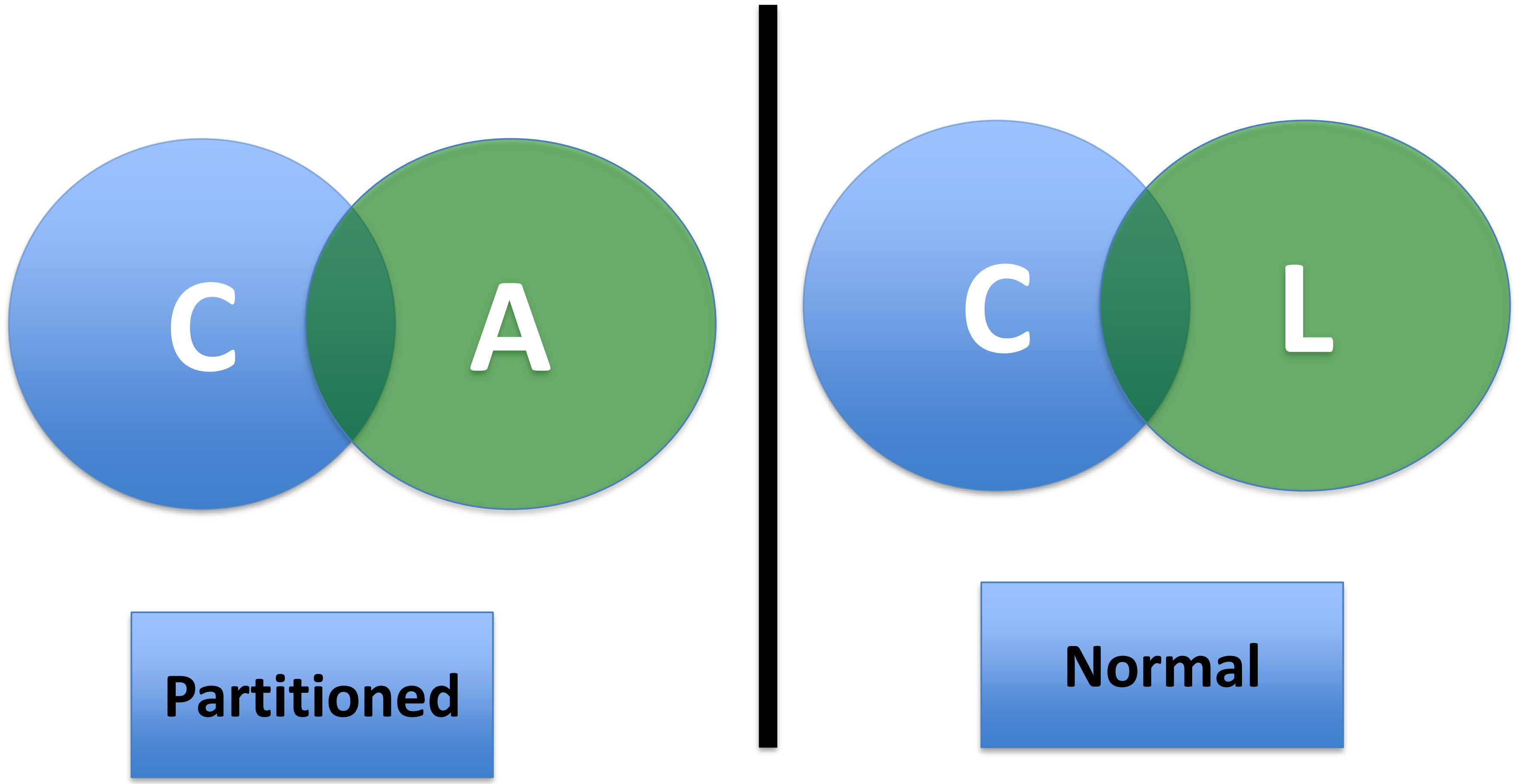
- Tradeoff between **Consistency** and **Latency**:
- Caused by the **possibility of failure** in distributed systems
  - High availability -> replicate data -> consistency problem
- Basic idea:
  - Availability and latency are arguably **the same thing**: unavailable -> extreme high latency
  - Achieving different levels of consistency/availability takes different amount of time

CAP -> PACELC

- A more complete description of the space of potential tradeoffs for distributed system:
  - If there is a **partition (P)**, how does the system trade off **availability and consistency (A and C)**; **else (E)**, when the system is running normally in the absence of partitions, how does the system trade off **latency (L) and consistency (C)**?

Abadi, Daniel J. "Consistency tradeoffs in modern distributed database system design." Computer-IEEE Computer Magazine 45.2 (2012): 37.

PACELC



Examples

- **PA/EL Systems:** Give up both Cs for availability and lower latency
  - Dynamo, Cassandra, Riak
- **PC/EC Systems:** Refuse to give up consistency and pay the cost of availability and latency
  - BigTable, Hbase, VoltDB/H-Store
- **PA/EC Systems:** Give up consistency when a partition happens and keep consistency in normal operations
  - MongoDB
- **PC/EL System:** Keep consistency if a partition occurs but gives up consistency for latency in normal operations
  - Yahoo! PNUTS