Beyond 348 (Optional)

CS348 Spring 2024

Instructor: Sujaya Maiyya

Sections: **002 & 003 only**

Announcements

- Assignment 3 due today on the 19th
- Send your choice of project demo (online or video) to your TA by **July 22nd**
- Next class: July 30th – review for finals Q&A style at 2:30 PM

YouTube

Spotify

Gmail

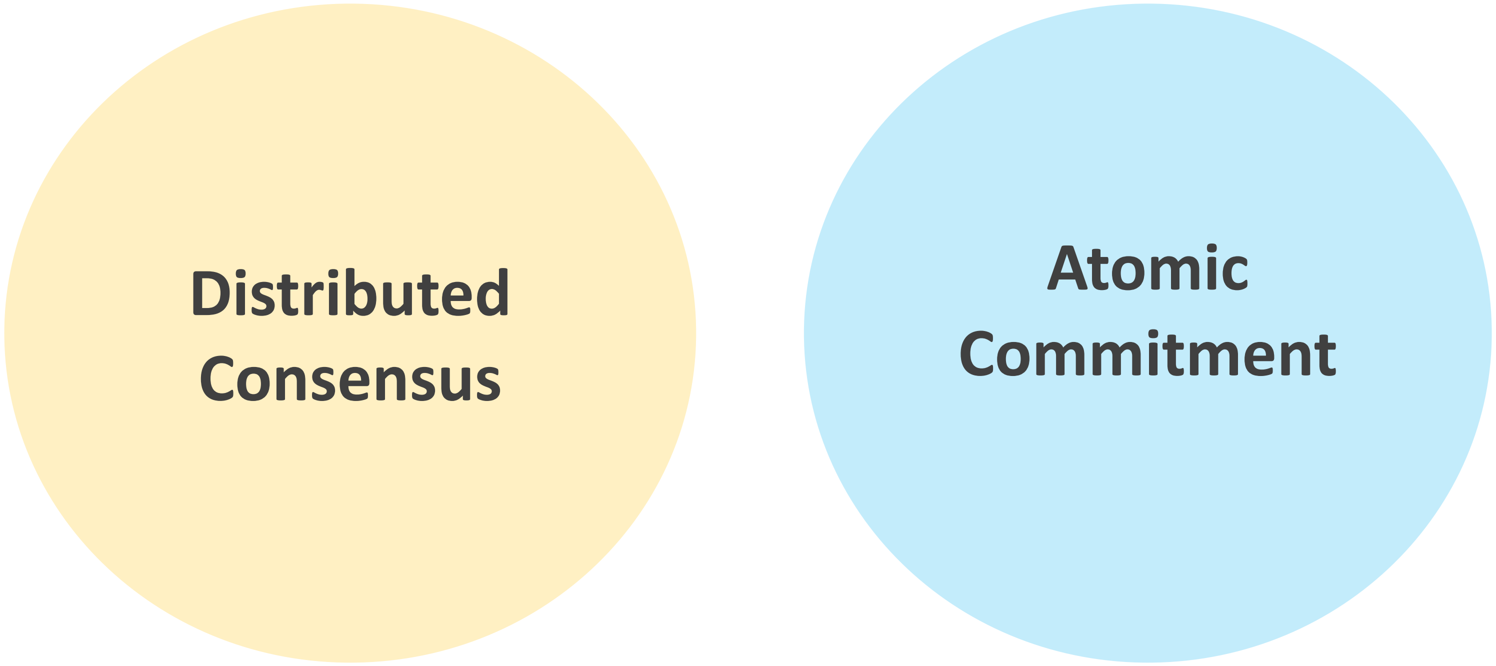
facebook

Instagram

NETFLIX

ebay

All these products (directly or indirectly) use



Many also store their data in the cloud

Properties Of A Data Management System

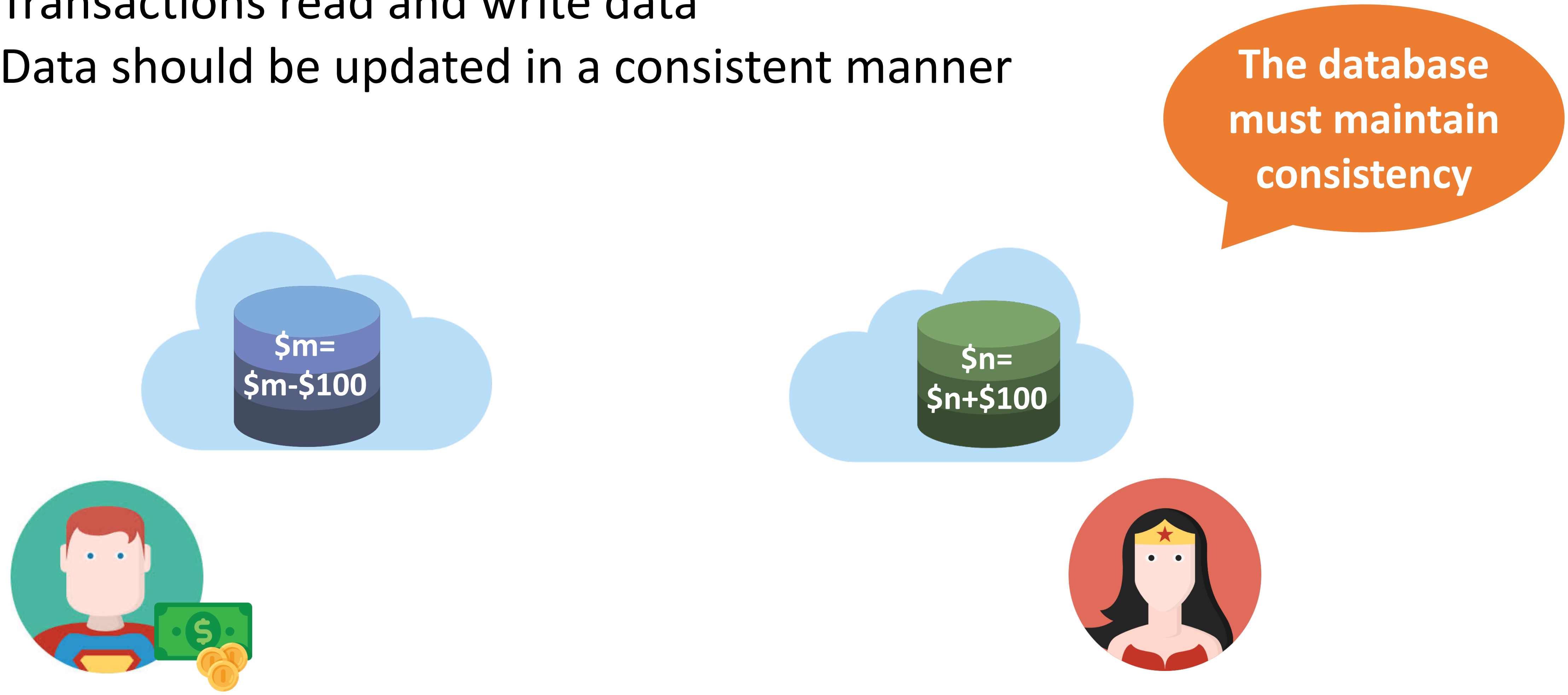
Scalability

- Data can be too large to be stored in a single server
- Shard or Partition the data
- Store smaller chunks in each server

Partition data e.g. based on category

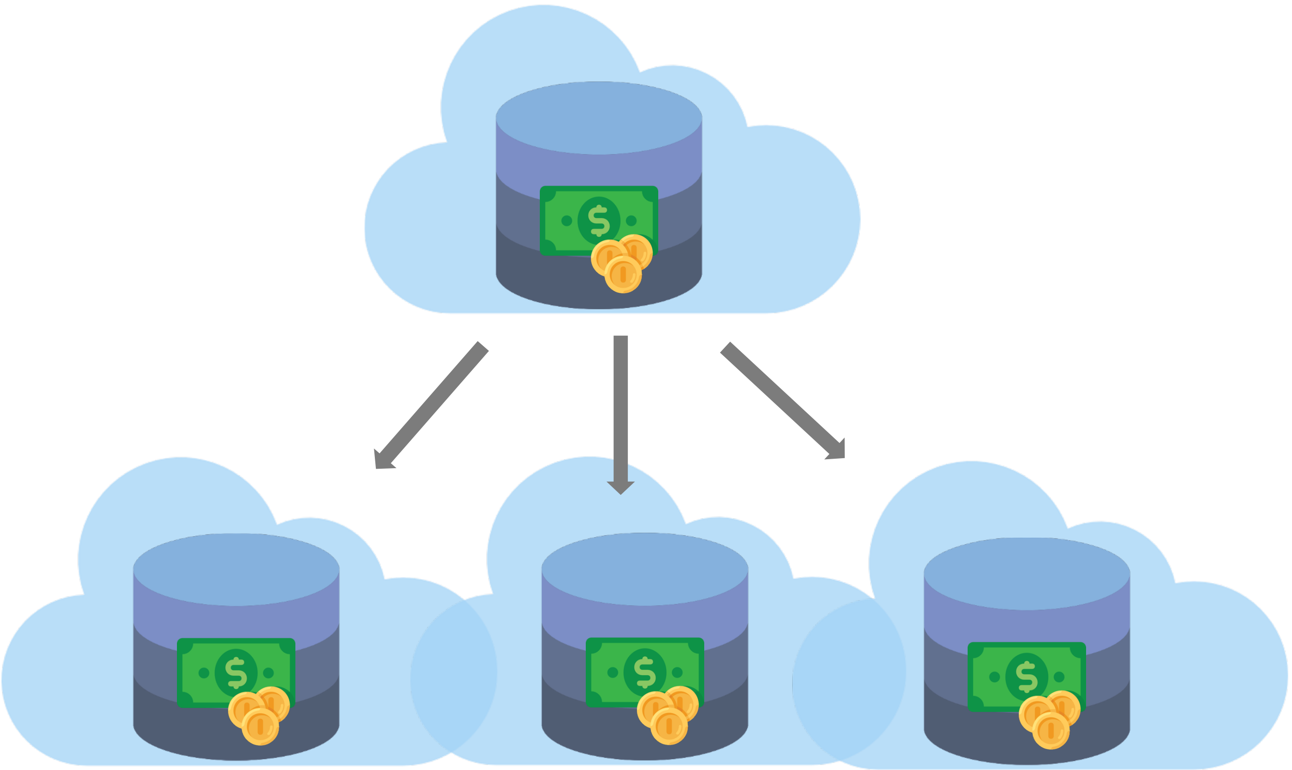
Consistency

- Transactions read and write data
- Data should be updated in a consistent manner



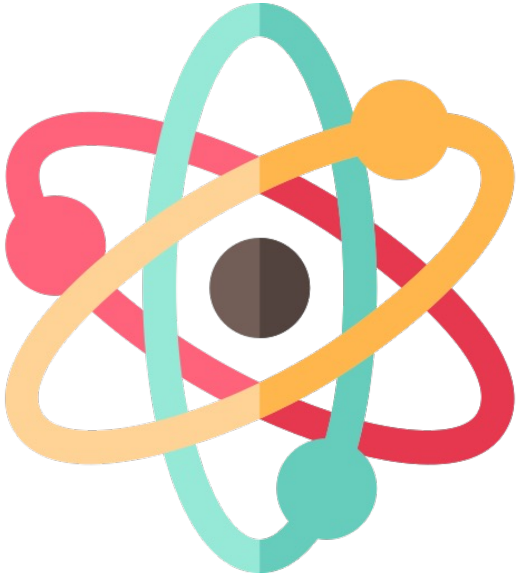
Fault-tolerance and Availability

- Commodity servers crash frequently
- Data should be replicated for fault-tolerance and high availability



Protocols Supporting the Cloud

- Scalability and Consistency
  - Atomic Commit Protocols
  - E.g., **Two Phase Commit**
  - Google Spanner, Apache Flink, VoltDB, Apache Kafka, and MS Azure SQL DB



- Fault-tolerance and Availability
  - Consensus and Replication Protocols
  - E.g., **Paxos**
  - MS CosmosDB, Google Spanner, Apache Cassandra, Neo4j, Amazon, IBM

PAXOS

PAXOS

- A ***consensus*** protocol: agreement on a single value

Retreat?

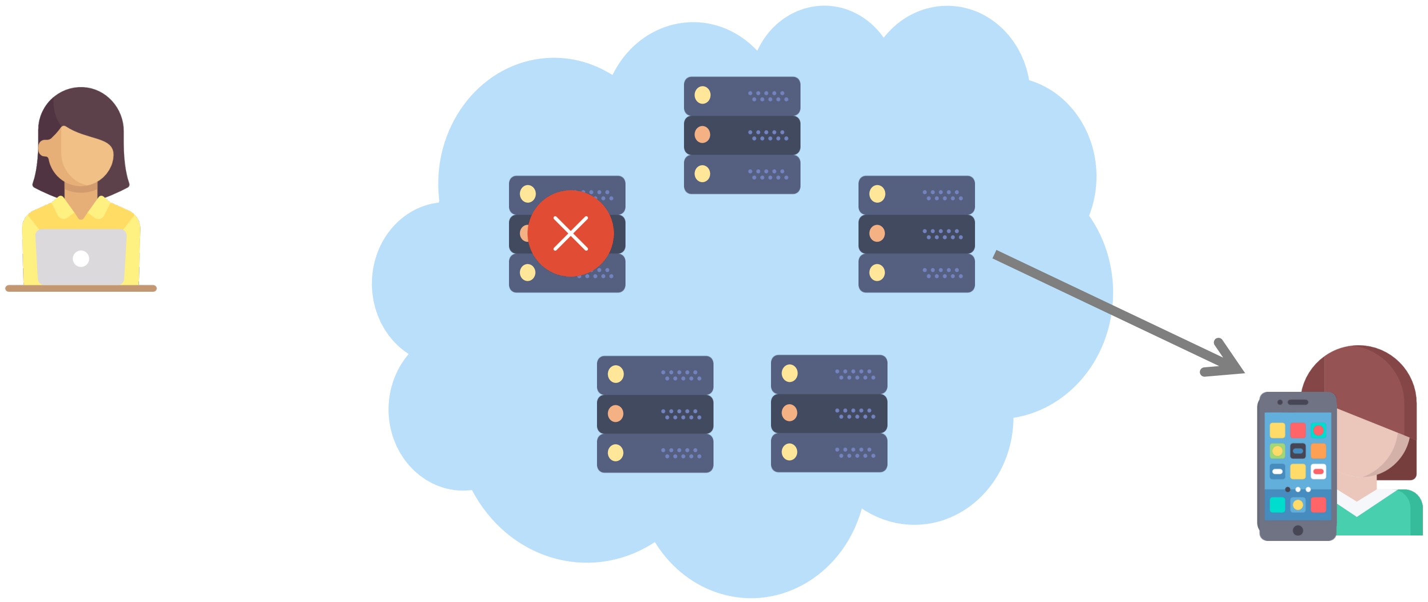
Attack?

PAXOS

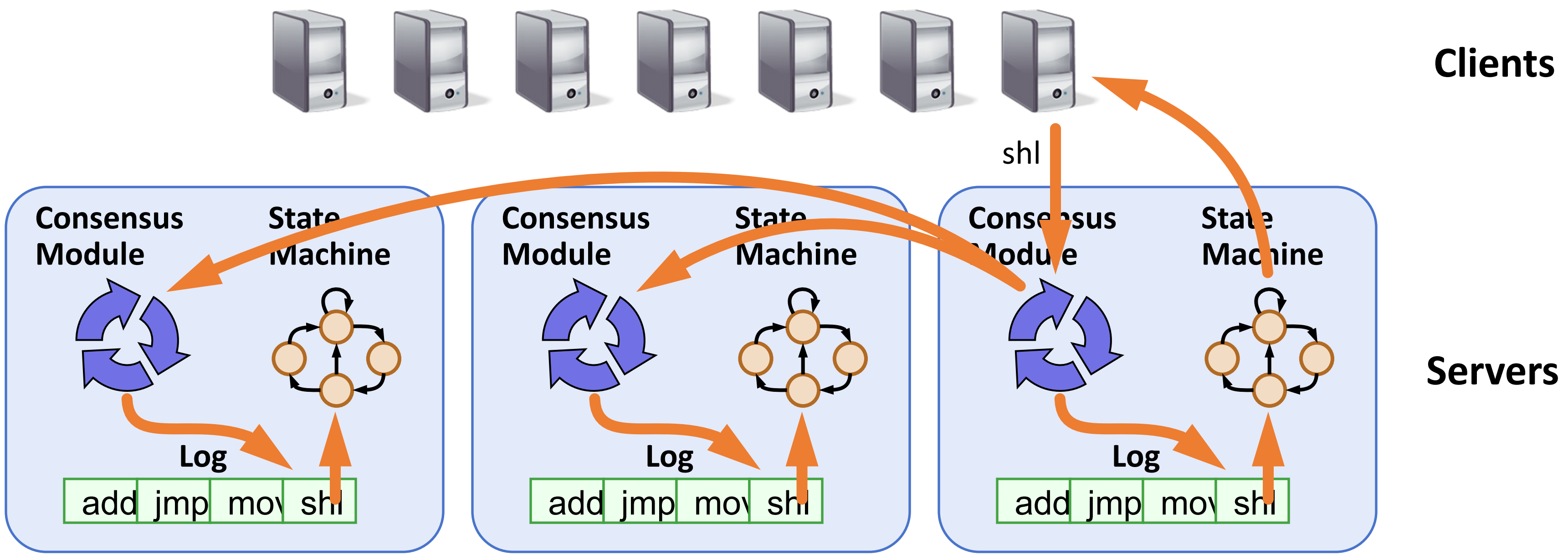
- A ***consensus*** protocol: agreement on a single value

Distributed State Machine

- Fault-tolerance through **replication**
  - Need to ensure that replicas remain consistent
  - Replicas must process requests in the same order



Goal: Replicated Log



- Replicated log → replicated state machine
  - All servers execute same commands in same order
  - Commands are deterministic
- Consensus module ensures proper log replication

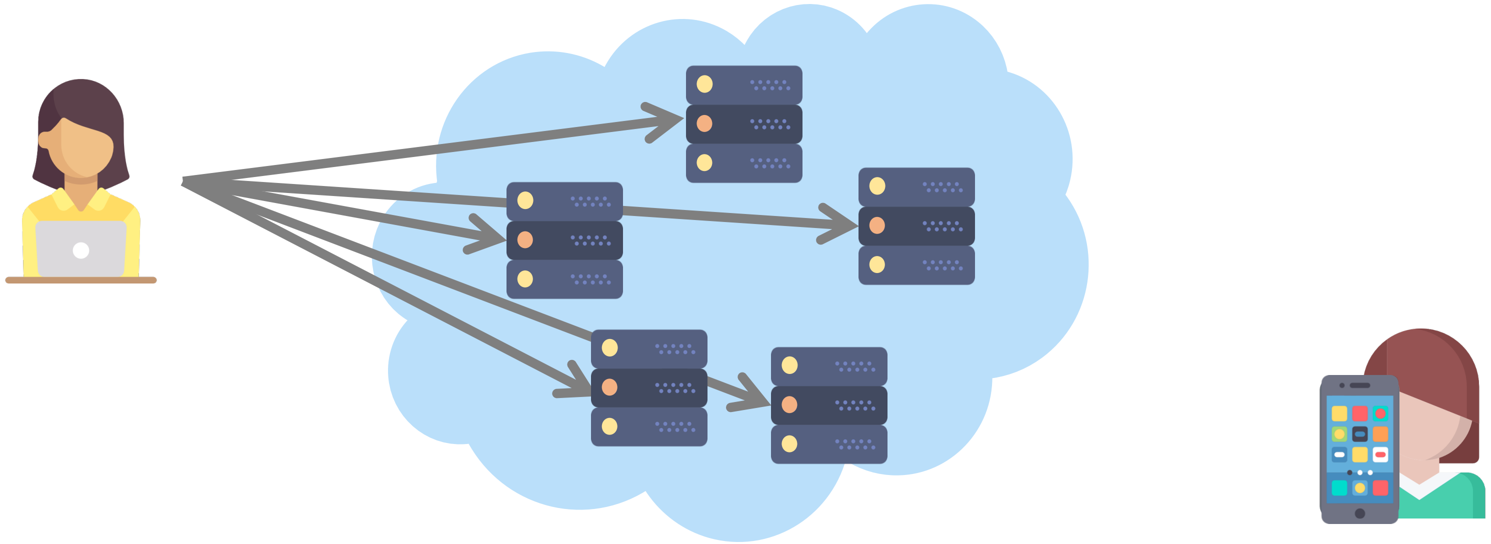
Paxos System Assumptions

- Paxos is an asynchronous consensus algorithm
  - Asynchronous networks
- Set of processes is known a-priori
- Failure model: fail-stop (not Byzantine), delayed/lost messages

- How many phases should Paxos have?

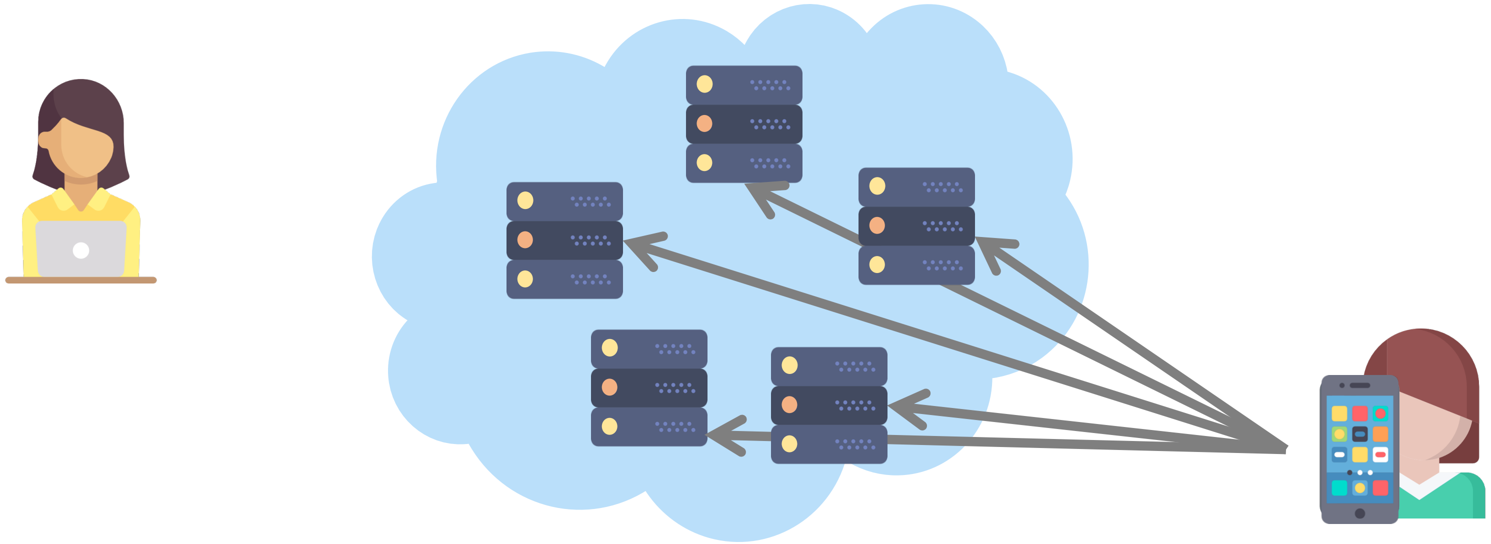
Attempt 1: Decentralized Protocol

- The clients 'know' all the replicas
- Clients send updates to all replicas



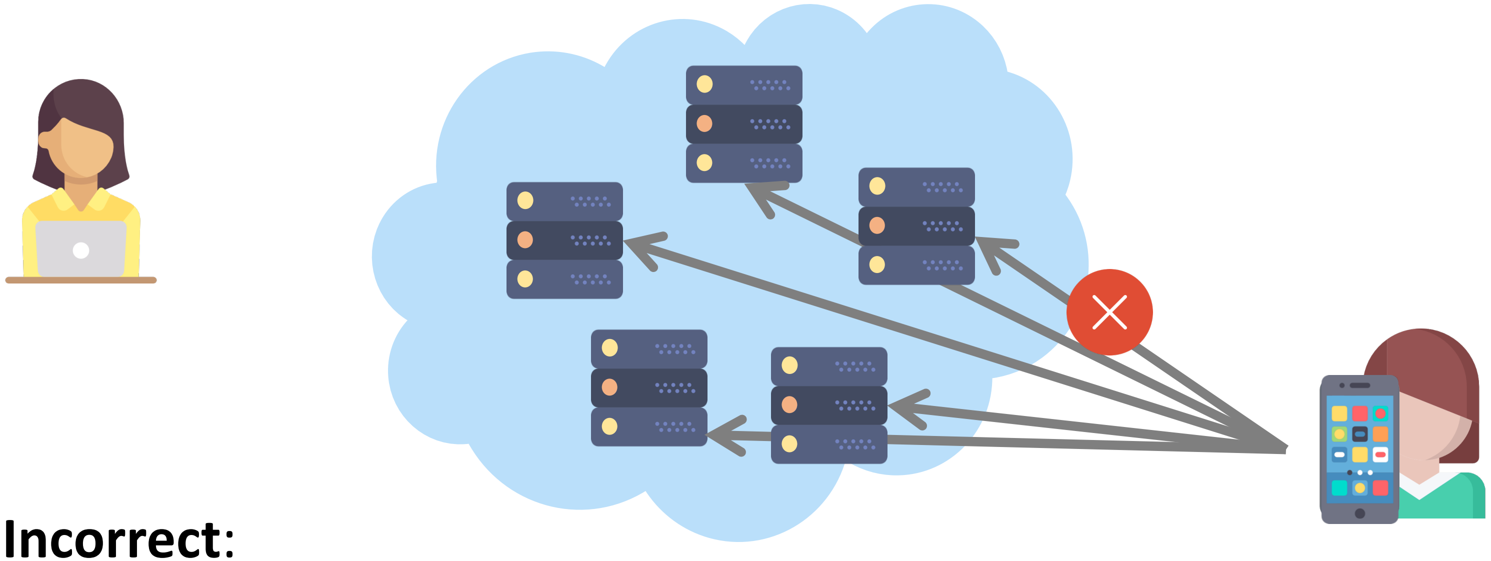
Attempt 1: Decentralized Protocol

- The clients ‘know’ all the replicas
- Clients send updates to all replicas



Attempt 1: Decentralized Protocol

- The clients 'know' all the replicas
- Clients send updates to all replicas



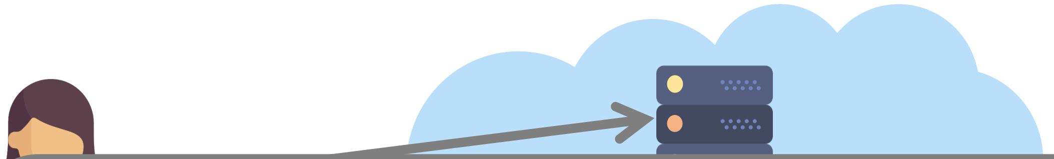
Incorrect:

- Message losses can lead to missed updates
- Reordered messages can cause unordered updates

→ Replicas in inconsistent state

Attempt 1: Decentralized Protocol

- The clients ‘know’ all the replicas
- Clients send updates to all replicas



Need a centralized solution
A leader to coordinate updates

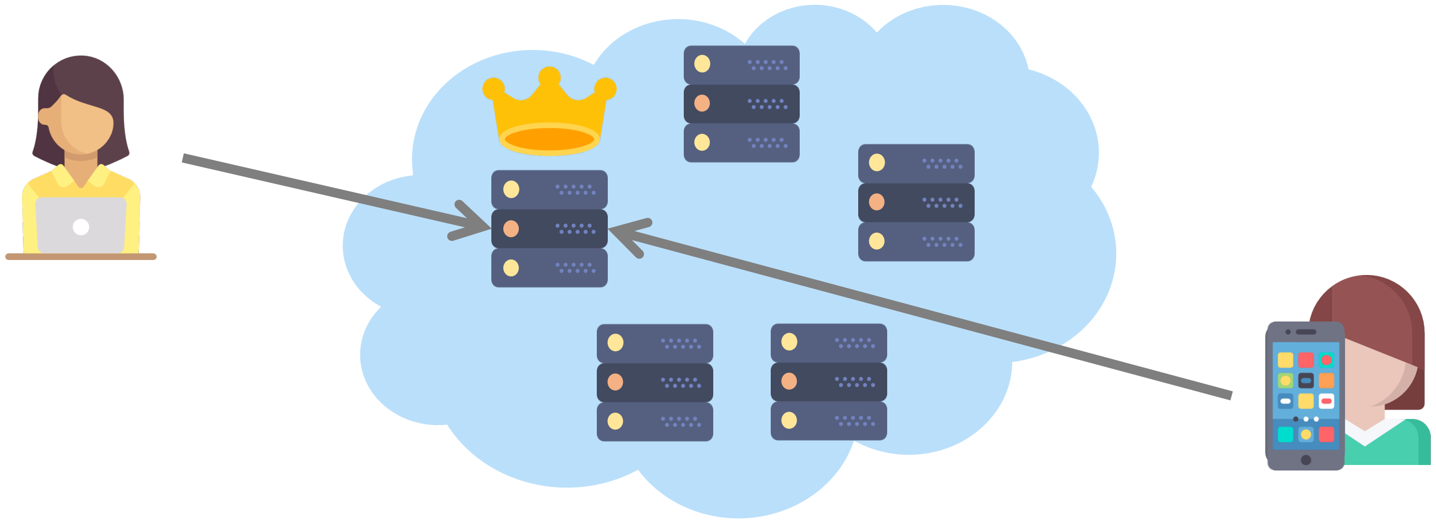
First phase: **LEADER ELECTION (Prepare)**

Incorrect:
- Message losses can lead to missed updates
- Reordered messages can cause unordered updates

→ Replicas in inconsistent state

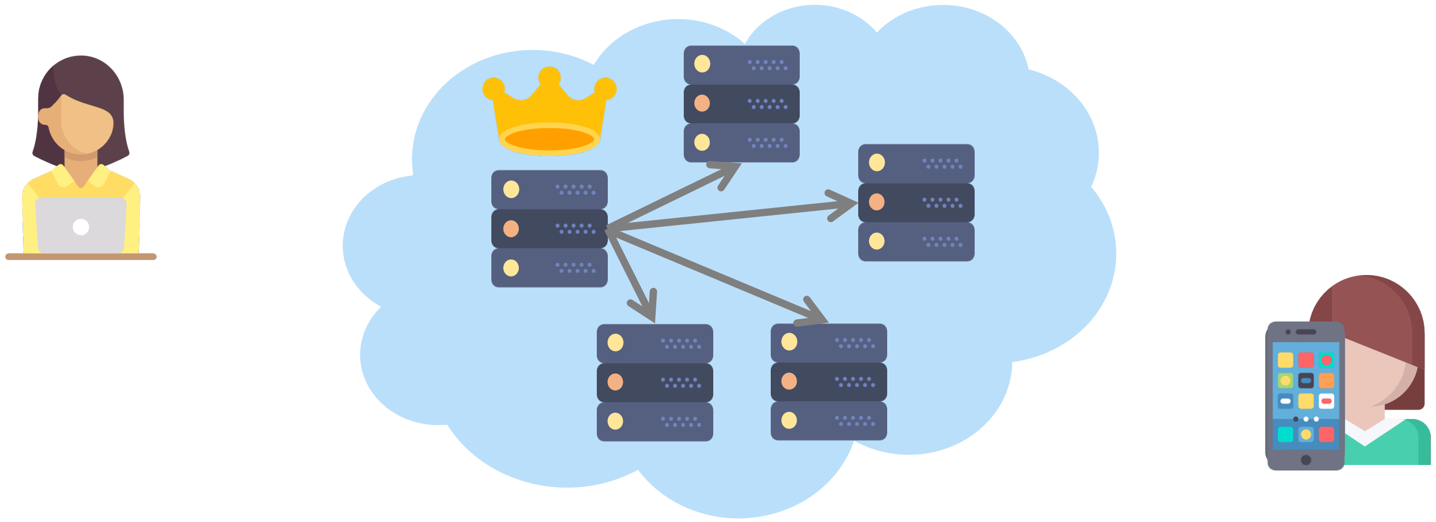
Attempt 2: Single Phase Solution

- Servers run Leader Election and elect a leader
- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas



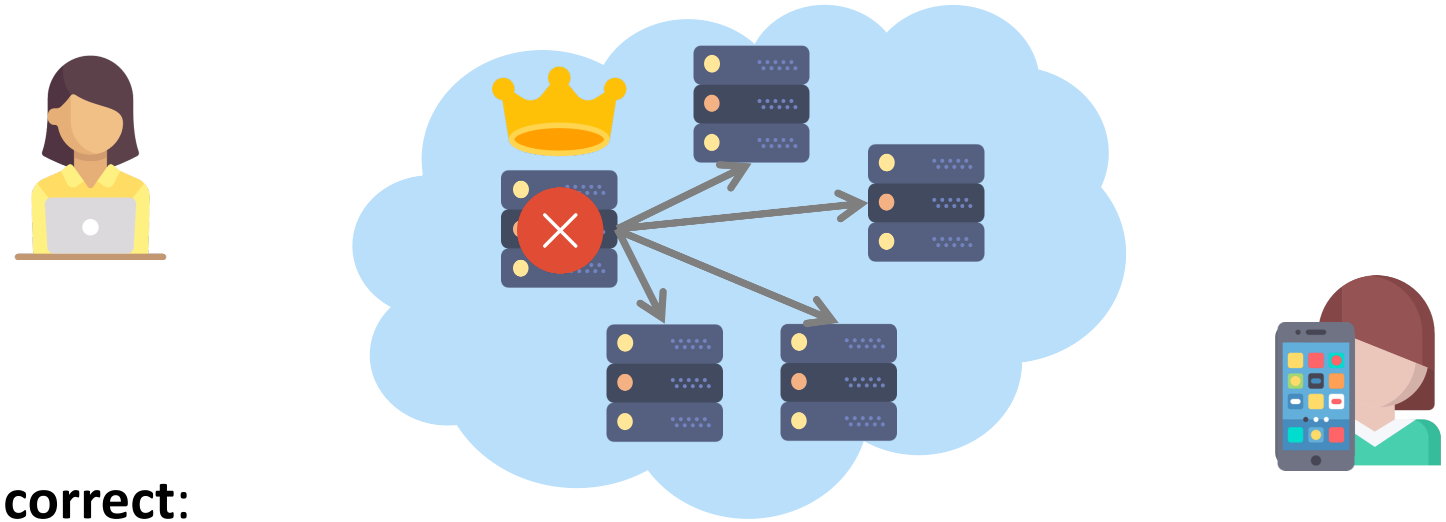
Attempt 2: Single Phase Solution

- Servers run Leader Election and elect a leader
- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas



Attempt 2: Single Phase Solution

- Servers run Leader Election and elect a leader
- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas

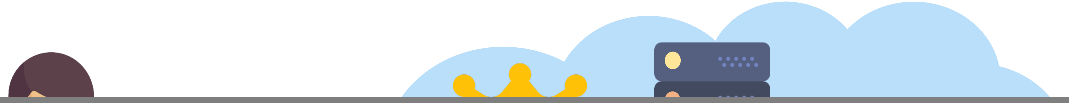


Incorrect:
- No confirmation that replicas got the updates sent by leader
- If leader crashes, no info about who got the updates

→ Replicas blocked or in inconsistent state

Attempt 2: Single Phase Solution

- Servers run Leader Election and elect a leader
- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas



Need a confirmation phase
For the replicas to agree on the update

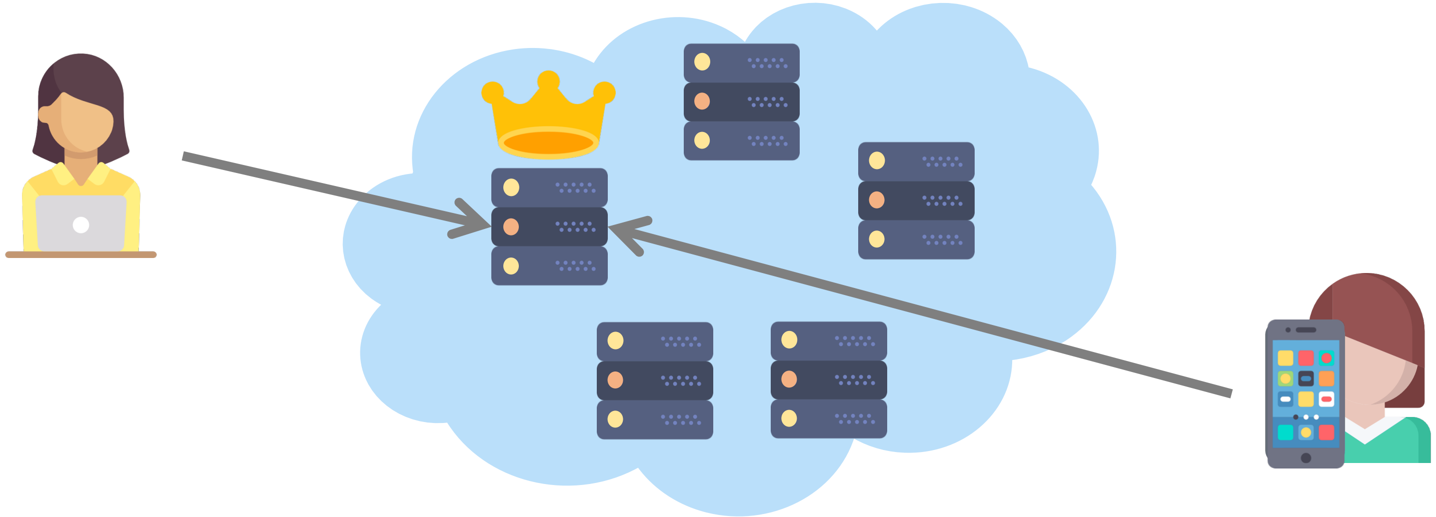
Second phase: **FAULT TOLERANT AGREEMENT (Accept)**

Incorrect:
- No confirmation that replicas got the updates sent by leader
- If leader crashes, no info about who got the updates

→ Replicas blocked or in inconsistent state

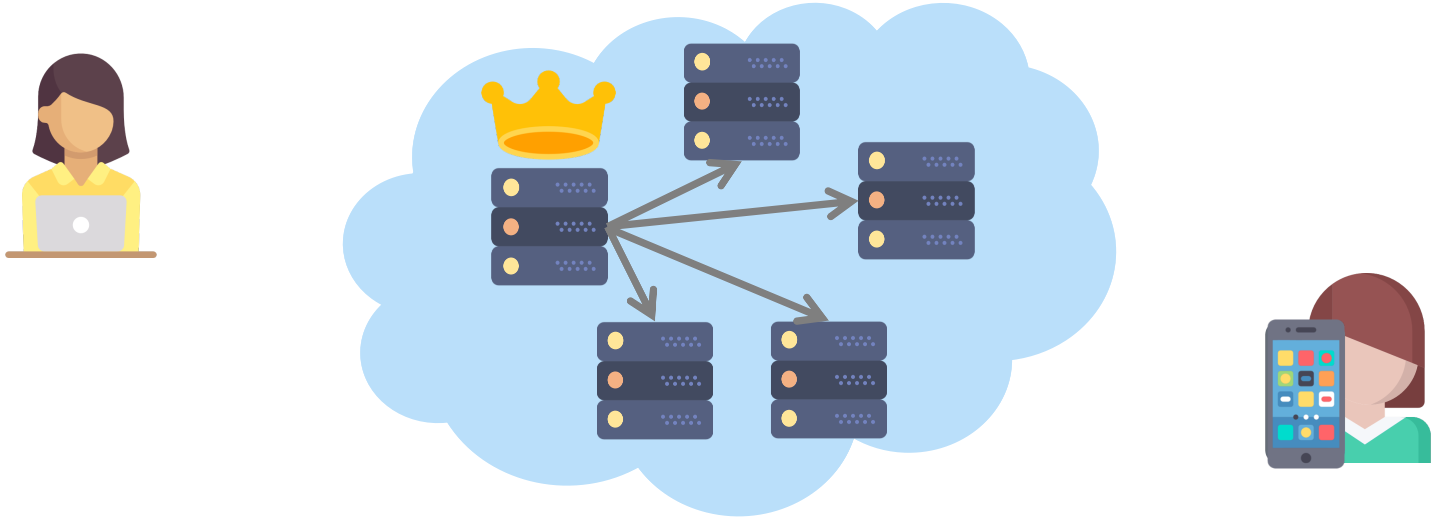
Attempt 3: Two Phase Solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates



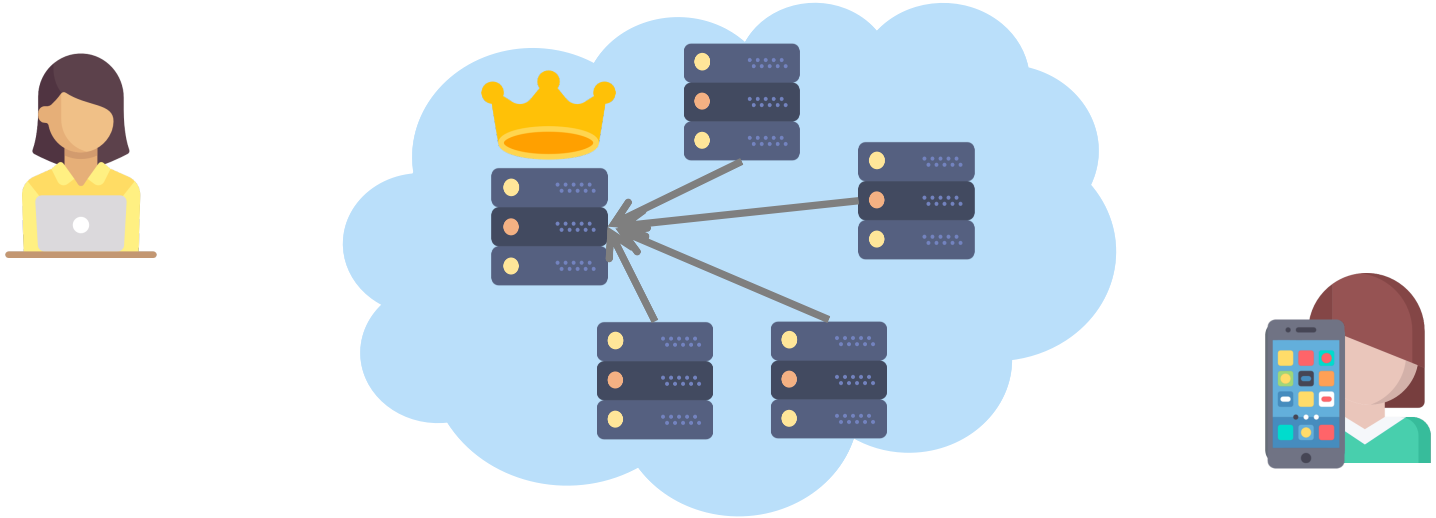
Attempt 3: Two Phase Solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates



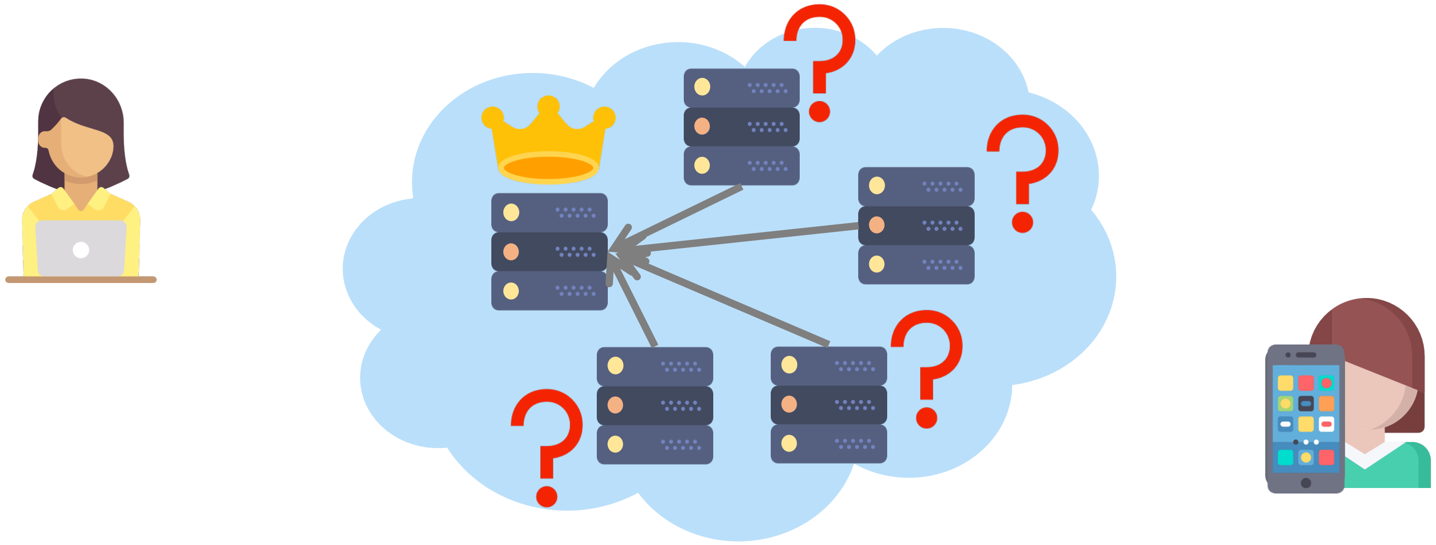
Attempt 3: Two Phase Solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates



Attempt 3: Two Phase Solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates

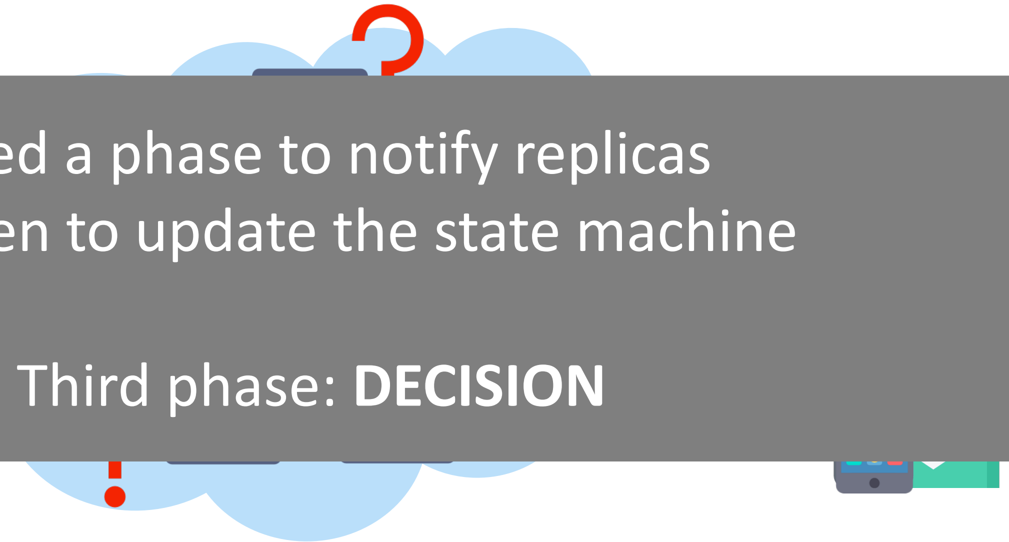


Not Enough:

- A replica needs to know when to update the state machine
- Unsure if leader got enough confirmation

Attempt 3: Two Phase Solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates



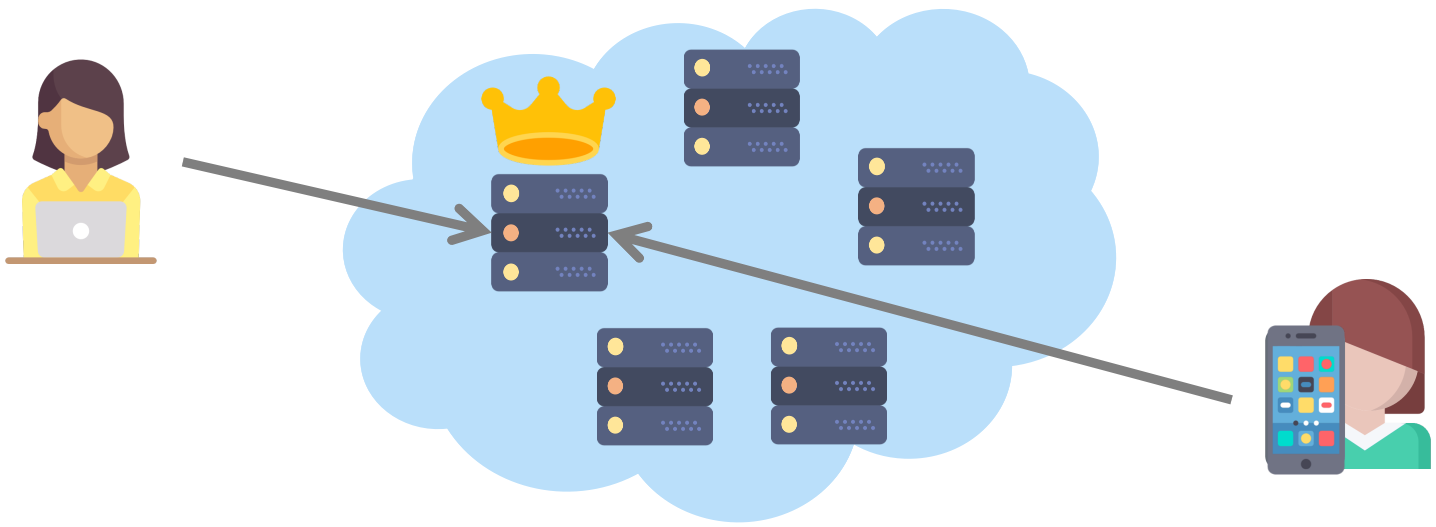
Need a phase to notify replicas
on when to update the state machine

Third phase: **DECISION**

Not Enough:
- A replica needs to know when to update the state machine
- Unsure if leader got enough confirmation

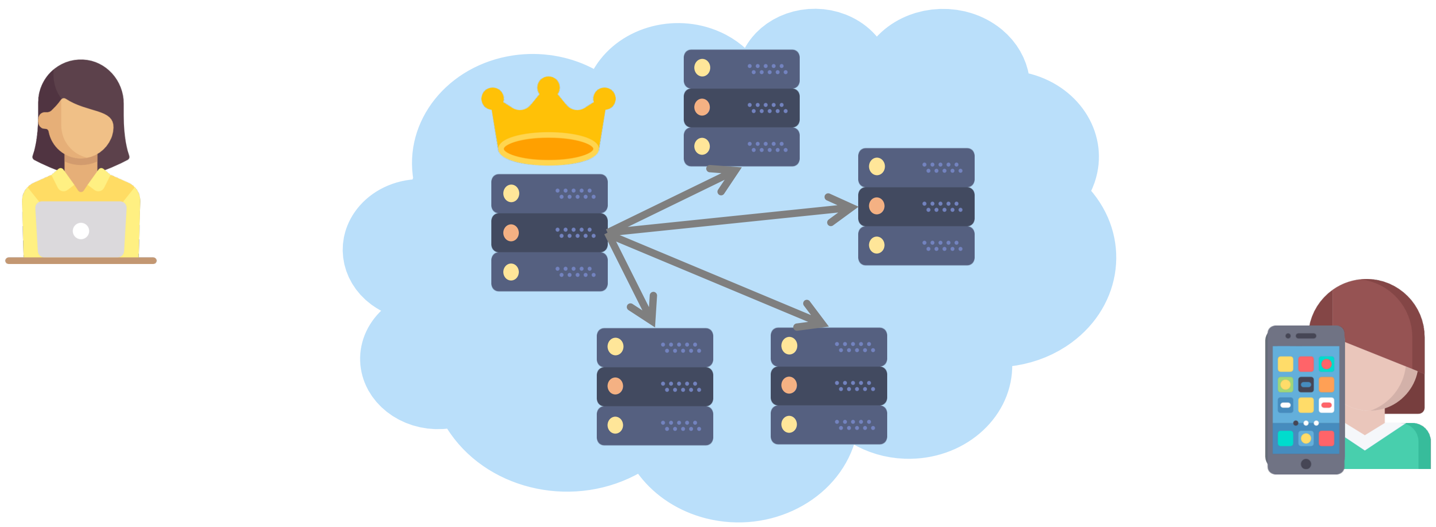
Final solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously



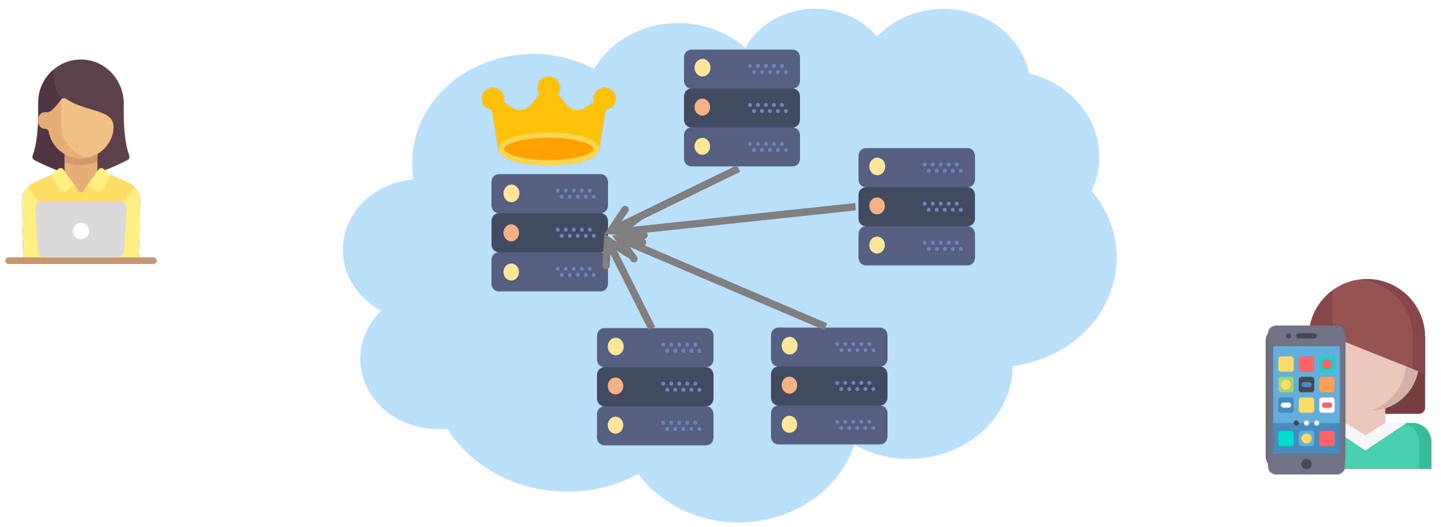
Final solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously



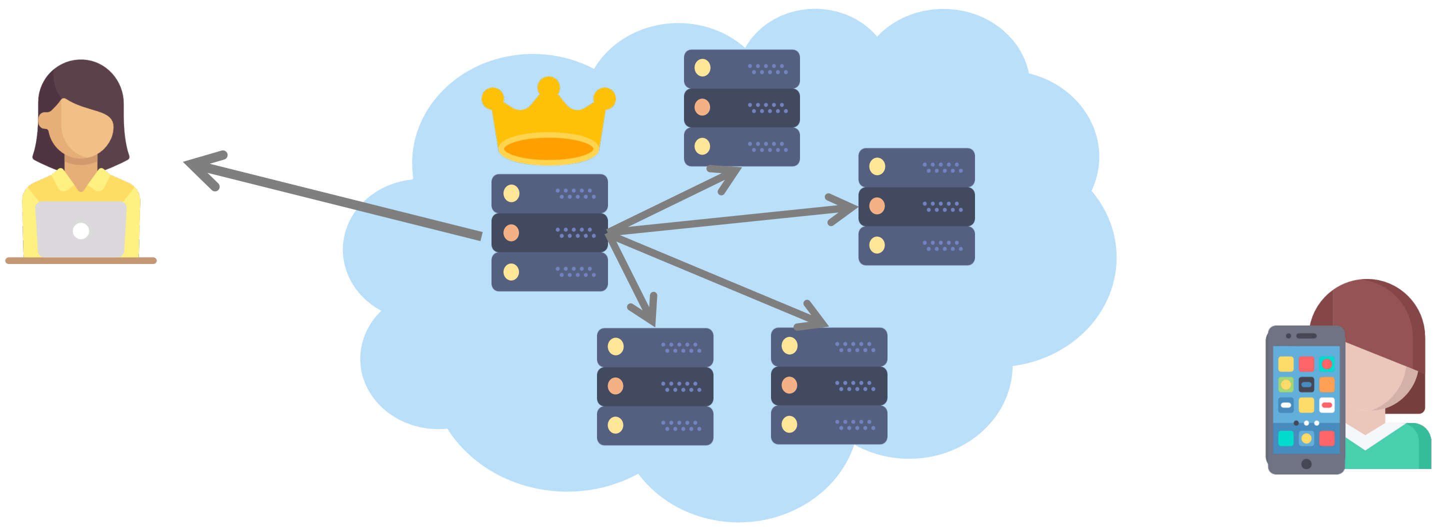
Final solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously



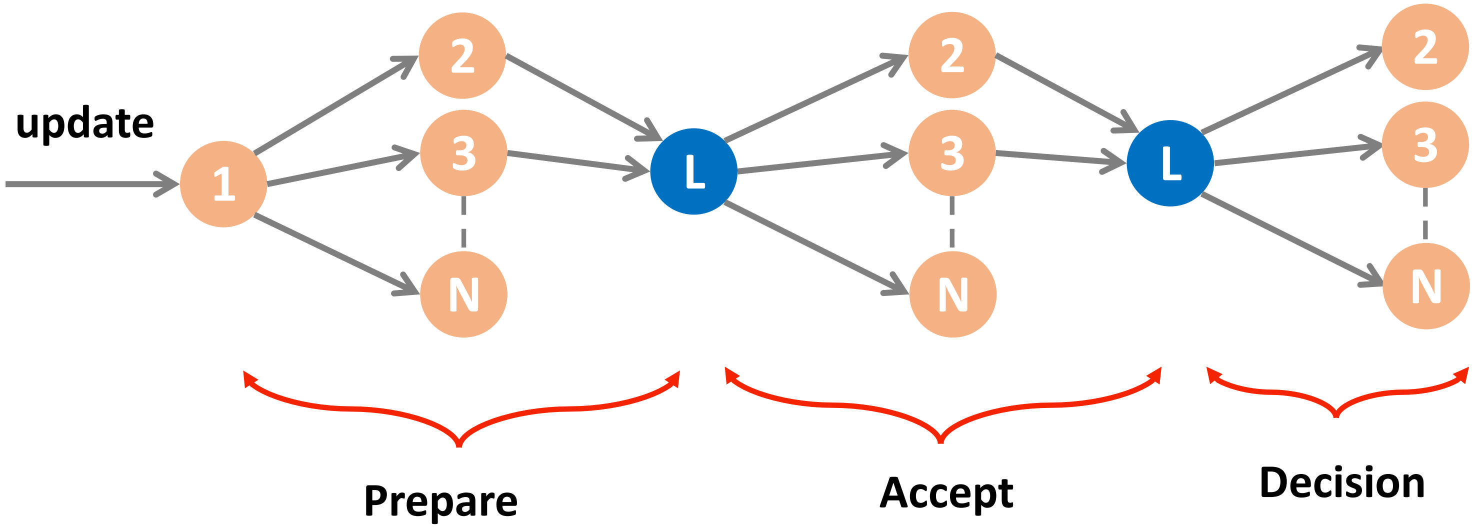
Final solution

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously



Final solution – Alternate rep.

- **Leader Election:** Initially, a leader is elected by a majority servers
- **Replication:** Leader replicates new updates on a majority servers
- **Decision:** Propagates decision to all **asynchronously**



Atomic Commitment

Two Phase Commit (2PC)

Paxos: Identical data

2PC: Partitioned data

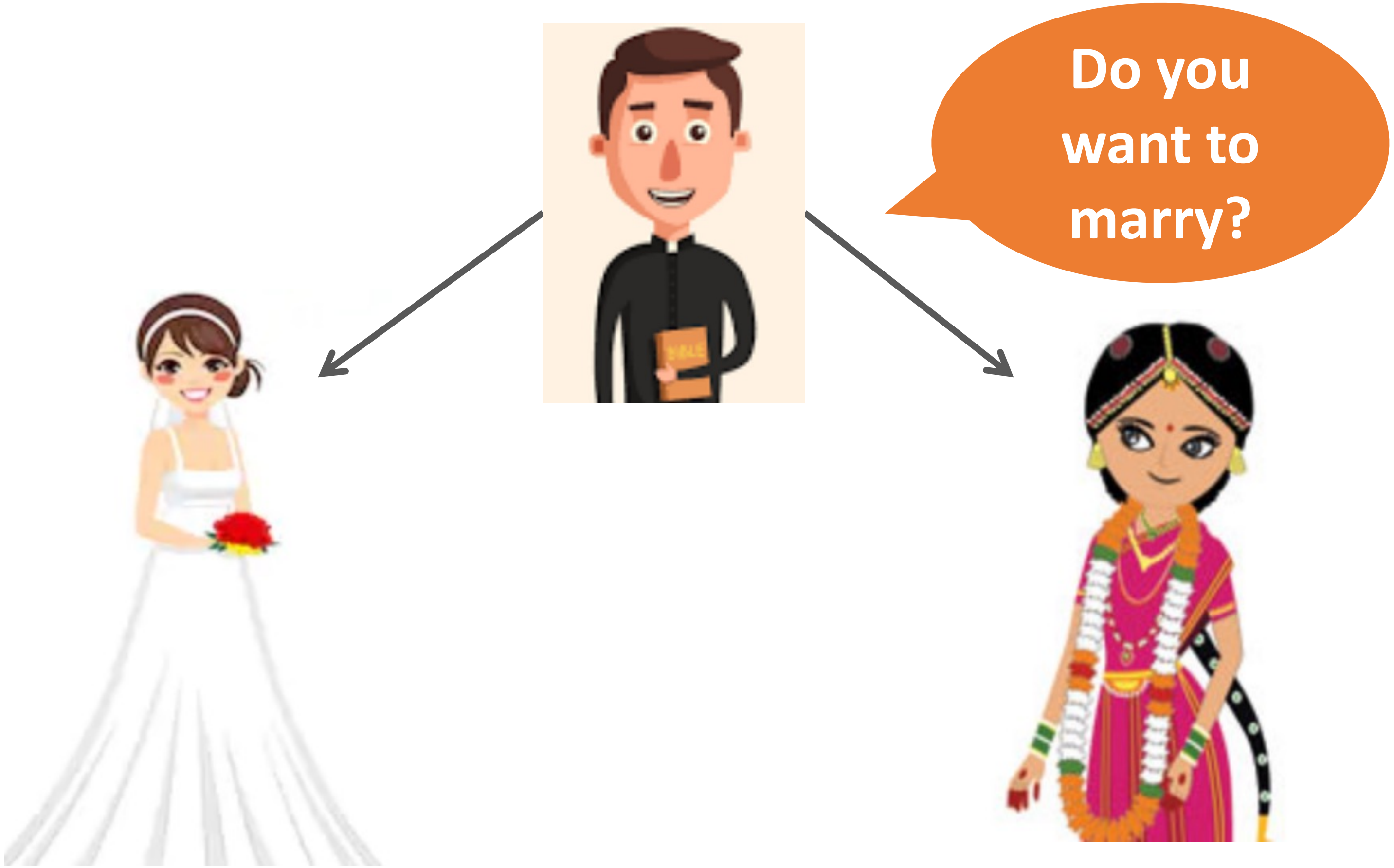
- A distributed transaction accesses data stored across multiple servers

- 2PC [1,2] is ***atomic commitment*** protocol: either all servers commit or no server commits

[1] J. N. Gray. "Notes on data base operating systems." *Operating Systems*. Springer, Berlin, Heidelberg, 1978. 393-481.
[2] B. Lampson and H. Sturgis. Crash recovery in a distributed system. Technical report, Xerox PARC Research Report, 1976.

Two Phase Commit

- Input from *all* parties necessary (unlike majority in Paxos)



Two Phase Commit

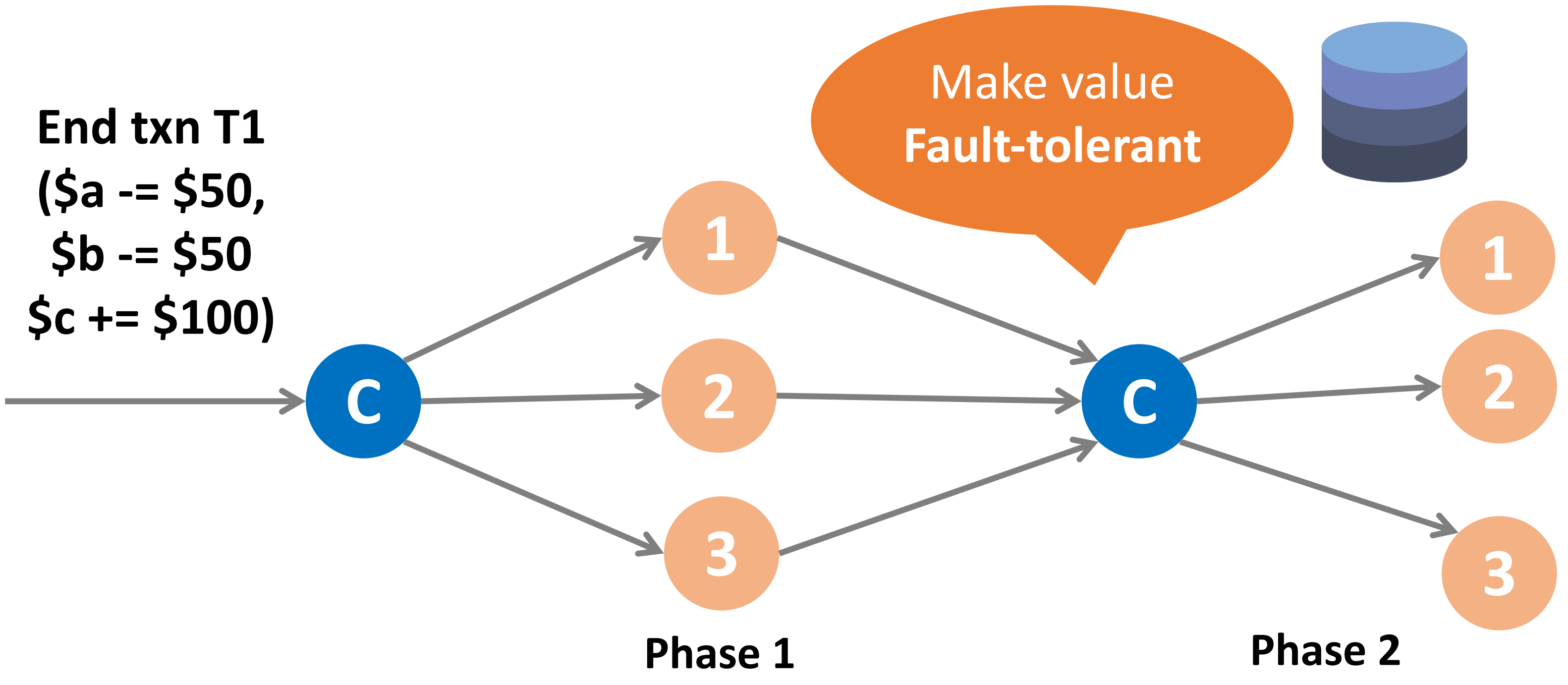
- Input from *all* parties necessary (unlike majority in Paxos)

Two Phase Commit

- Input from *all* parties necessary (unlike majority in Paxos)

Two Phase Commit

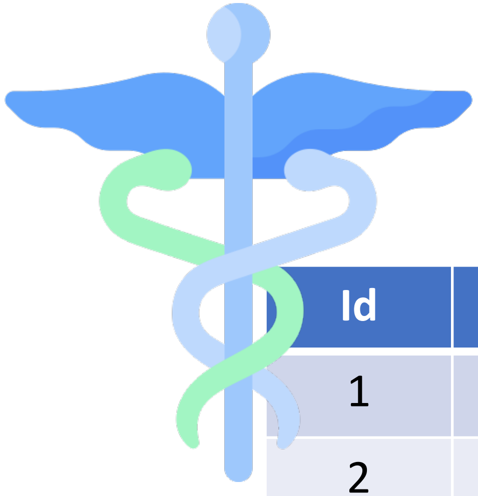
- **Phase 1:** Coordinator collects votes from **ALL** shards involved in the txn
- **Phase 2 (Decision)**: Send Decision to all cohorts



Data privacy

Data encryption to achieve privacy?

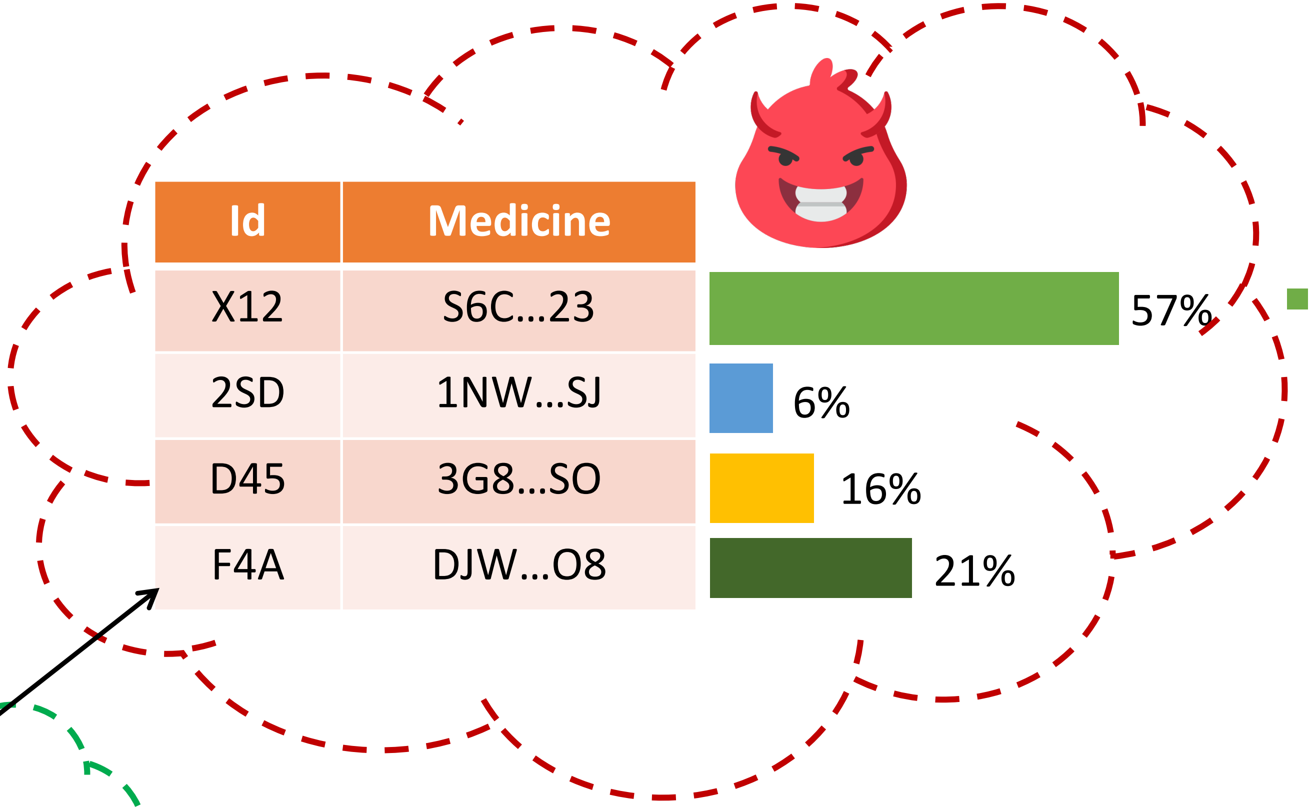
Potentially
non-trustworthy



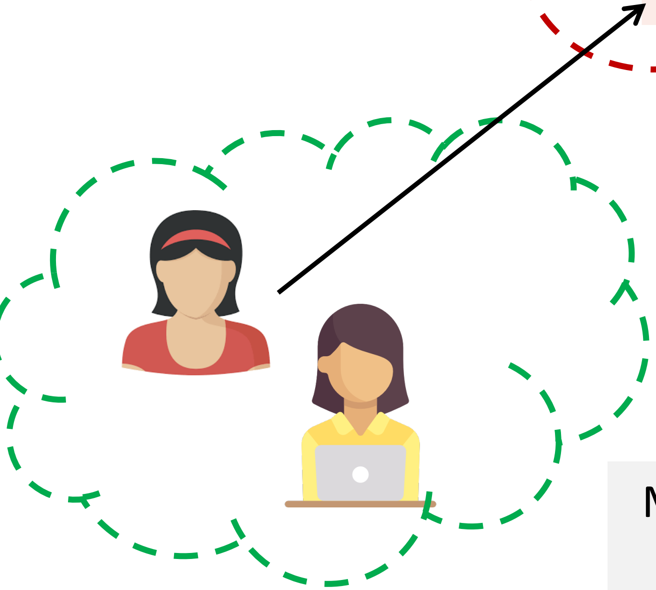
Id	Medicine
1	Humira
2	Januvia
3	Tivicay
4	Herceptin

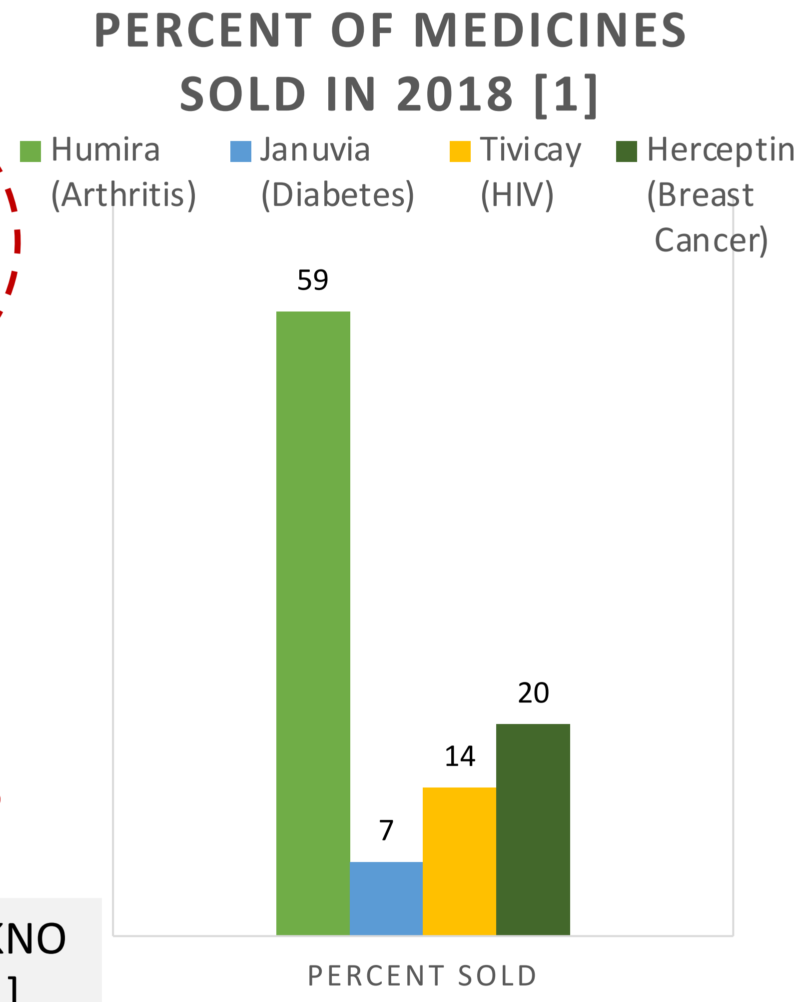
Id	Medicine
X12	S6C…23
2SD	1NW…SJ
D45	3G8…SO
F4A	DJW…O8

Encryption is **not** sufficient for data privacy



Id	Medicine
X12	S6C…23
2SD	1NW…SJ
D45	3G8…SO
F4A	DJW…O8





Access Pattern Attacks

Many practical attacks: [IKK NDSS'12], [CGPR CCS'15], [KKNO CCS'16], [GLMP S&P'19], [KPT S&P'19], [OK Security'21]

[1] https://truecostofhealthcare.org/pharmas-50-best-sellers/

We build

- Data systems that mitigate these attacks – called Oblivious databases
- Privacy-preserving systems that are scalable and fault tolerant
- Data systems that allow tuning security vs. performance trade-off

Your feedback matters

- Please fill out: https://perceptions.uwaterloo.ca by July 30th

Source: the clip-art library