Practical Lab
Cloud Systems Engineering
(cloud-lab)

Chair of Decentralized Systems Engineering
https://dse.in.tum.de/

Task #3:
Replicated Distributed KVS

Why distribute a single-node KVS?

- Performance
  - A single node serves all PUTS/GETS for all keys
  - The system's throughput = the node's throughput
  - A single node is a bottleneck by definition
  - **Solution:** sharding (the previous task!)
- Fault-tolerance
  - A single node is a single point of failure
  - If this node fails, the system become unavailable
  - **Solution:** replication (this task !)

→ **Combine both!** A sharded KVS, where each shard is replicated!

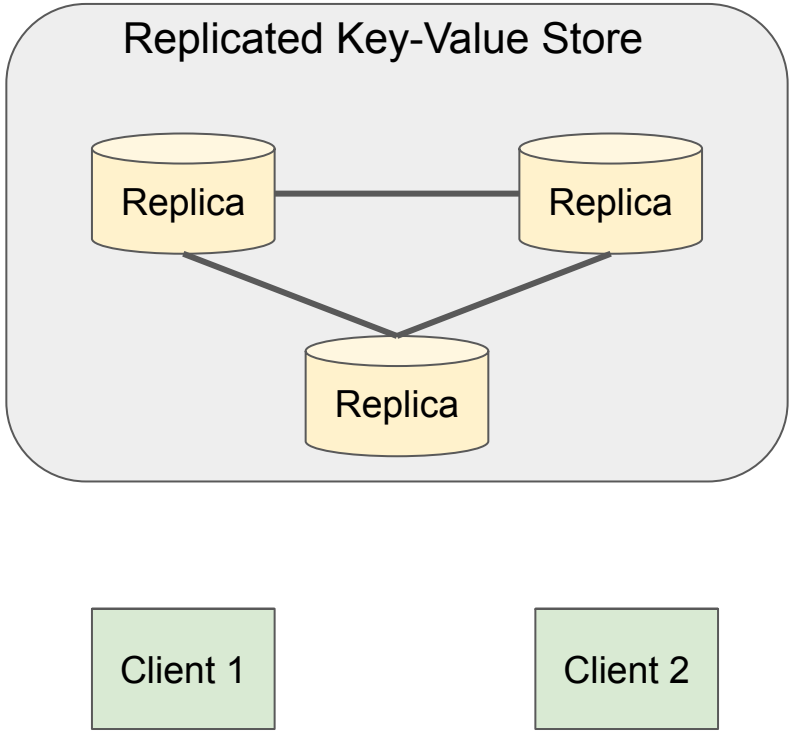
Learning Goals

In this week's task you will:

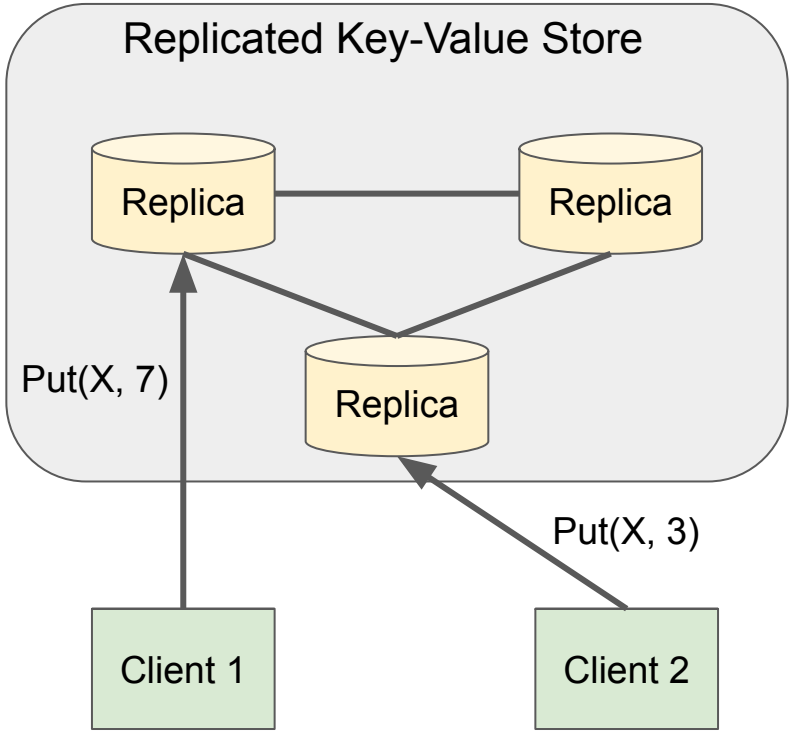
- Learn how to make a distributed KVS fault-tolerant through replication
- Learn the basics of the Raft consensus algorithm
- Build your own distributed and replicated key-value store with Raft

Background

Challenges in Replication



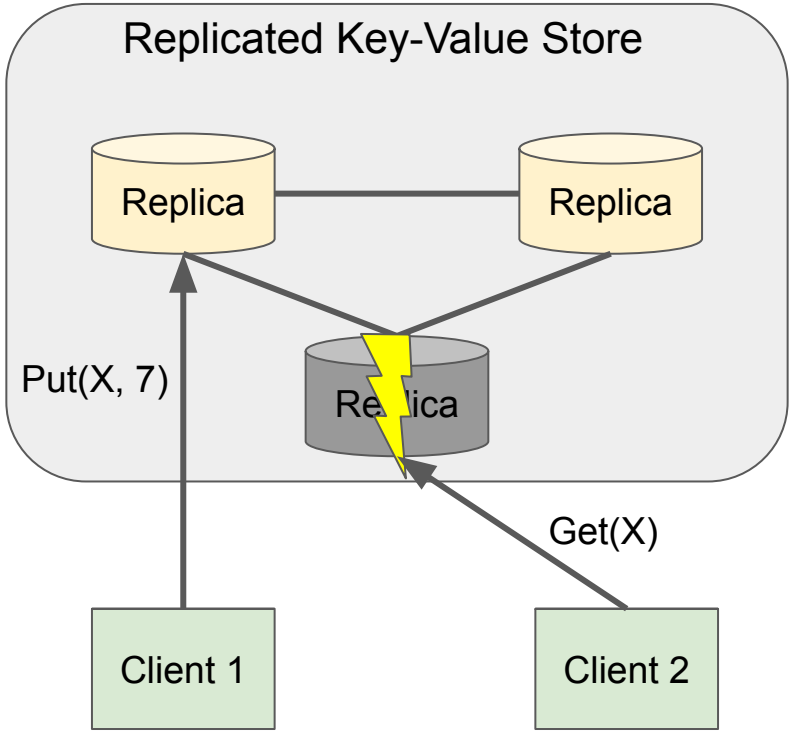
Challenges in Replication



State Conflicts

- How to decide one common value between all replicas?

Challenges in Replication



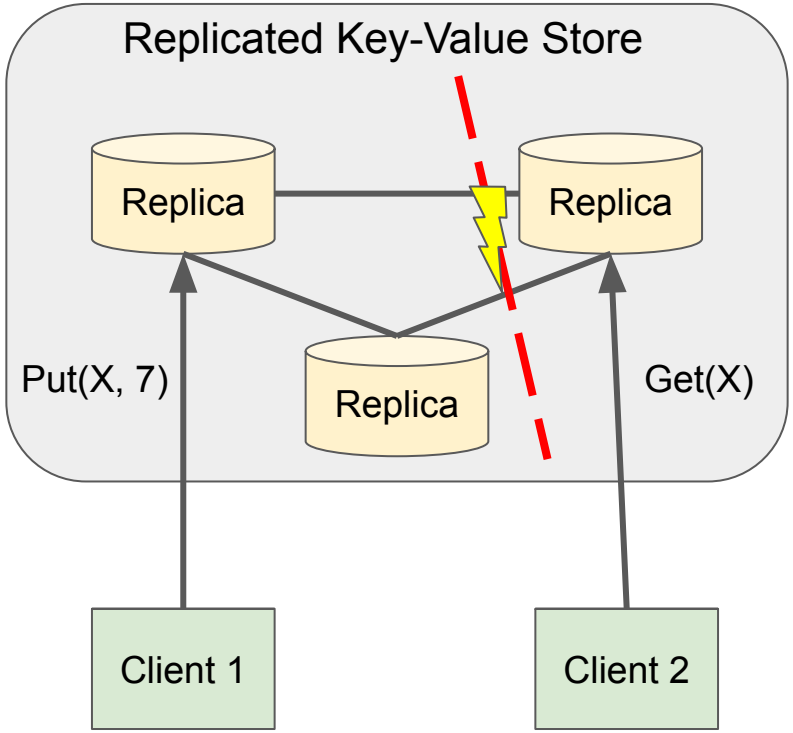
State Conflicts

- How to decide one common value between all replicas?

Node Failures

- What if a disk or server shuts down unexpectedly?
- What if the server starts up again?

Challenges in Replication



State Conflicts

- How to decide one common value between all replicas?

Node Failures

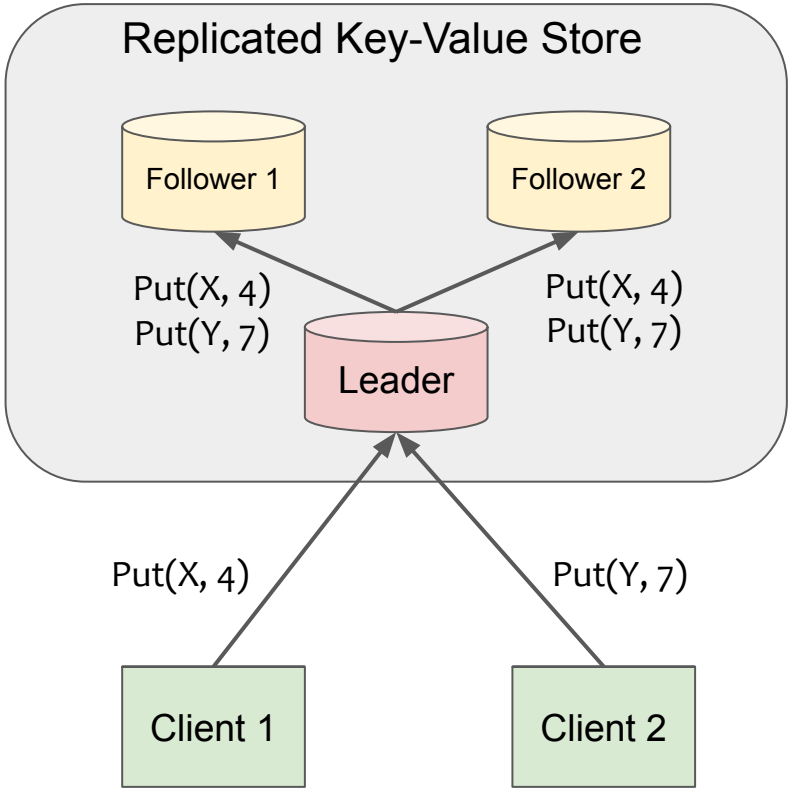
- What if a disk or server shuts down unexpectedly?
- What if the server starts up again?

Network Failures

- Network partitions or packet loss

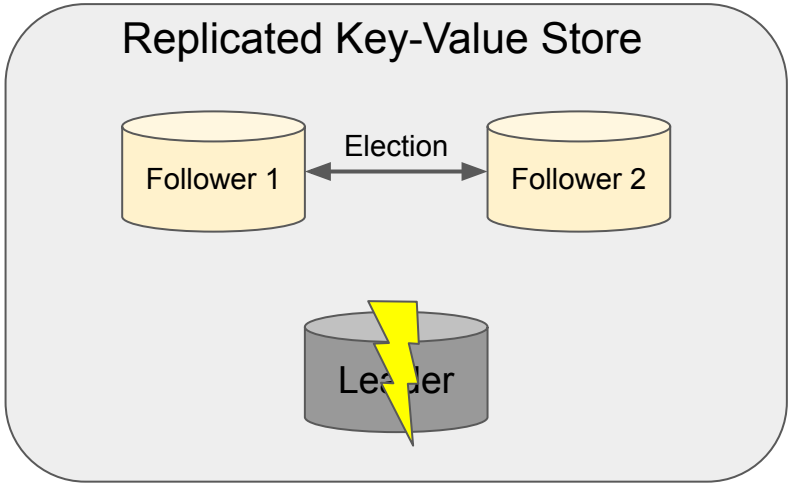
Leader-based Replication

- One Leader node
  - Communicates with the client
  - Determines order of operation
  - Orchestrate replication in all nodes
- Multiple follower nodes
  - Do not communicate with client
  - Follow the commands of the leader
  - Involved in leader election
- Examples
  - Paxos
  - ZAB
  - **Raft**



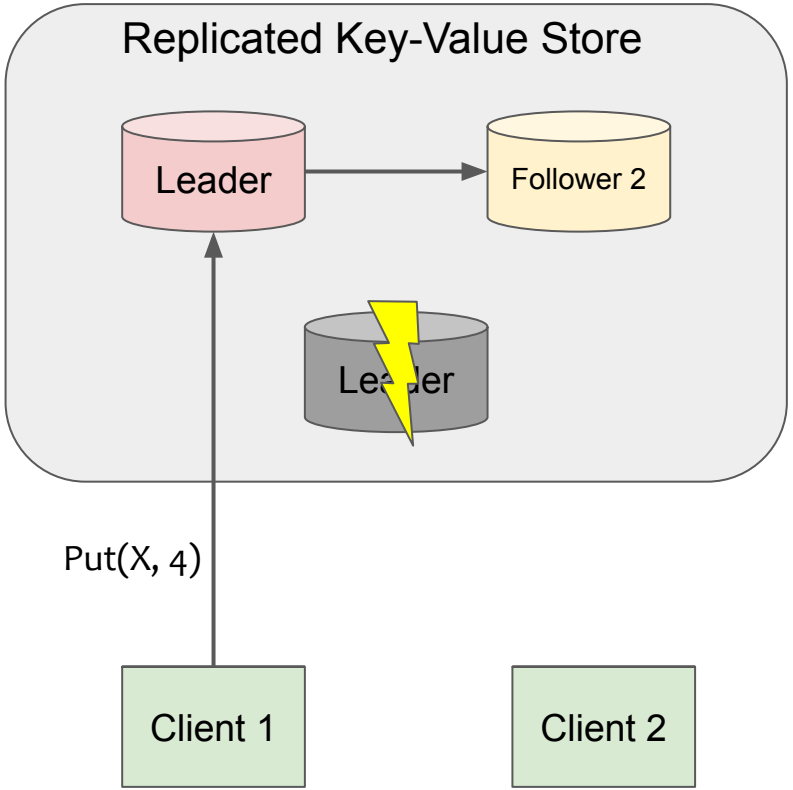
Leader-based Replication

- One Leader node
  - Communicates with the client
  - Determines order of operation
  - Orchestrate replication in all nodes
- Multiple follower nodes
  - Do not communicate with client
  - Follow the commands of the leader
  - Involved in leader election
- Examples
  - Paxos
  - ZAB
  - **Raft**



Leader-based Replication

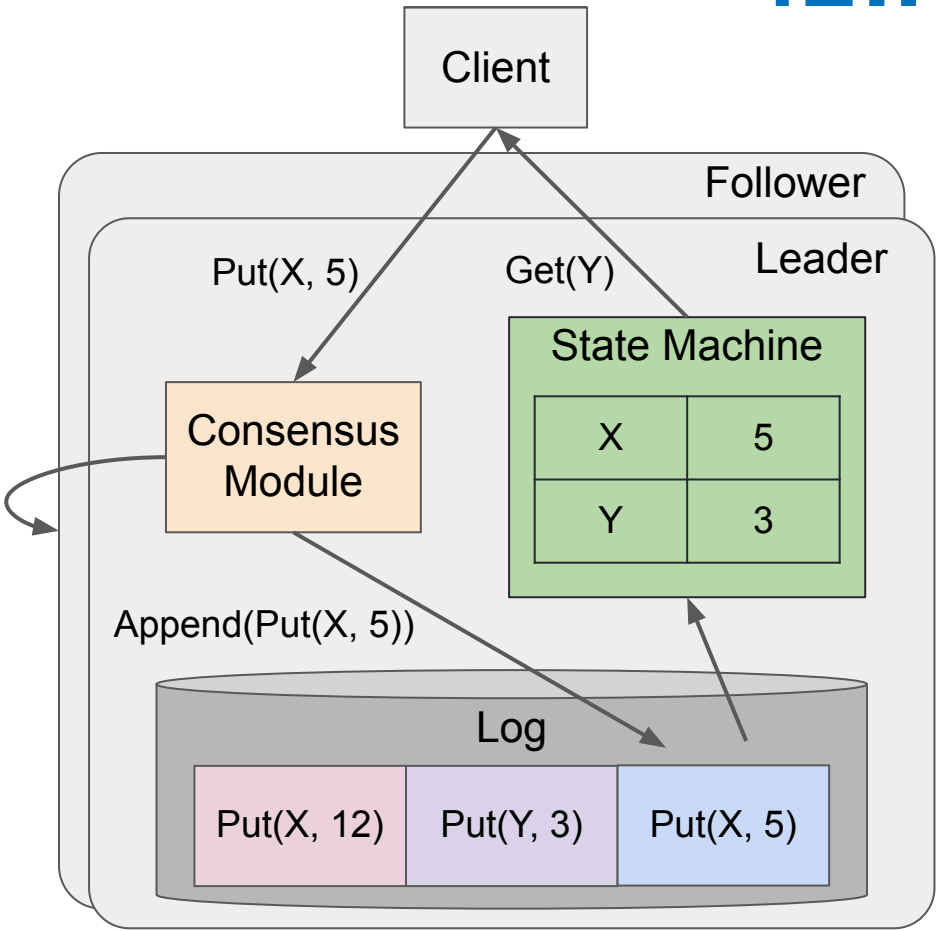
- One Leader node
  - Communicates with the client
  - Determines order of operation
  - Orchestrate replication in all nodes
- Multiple follower nodes
  - Do not communicate with client
  - Follow the commands of the leader
  - Involved in leader election
- Examples
  - Paxos
  - ZAB
  - **Raft**



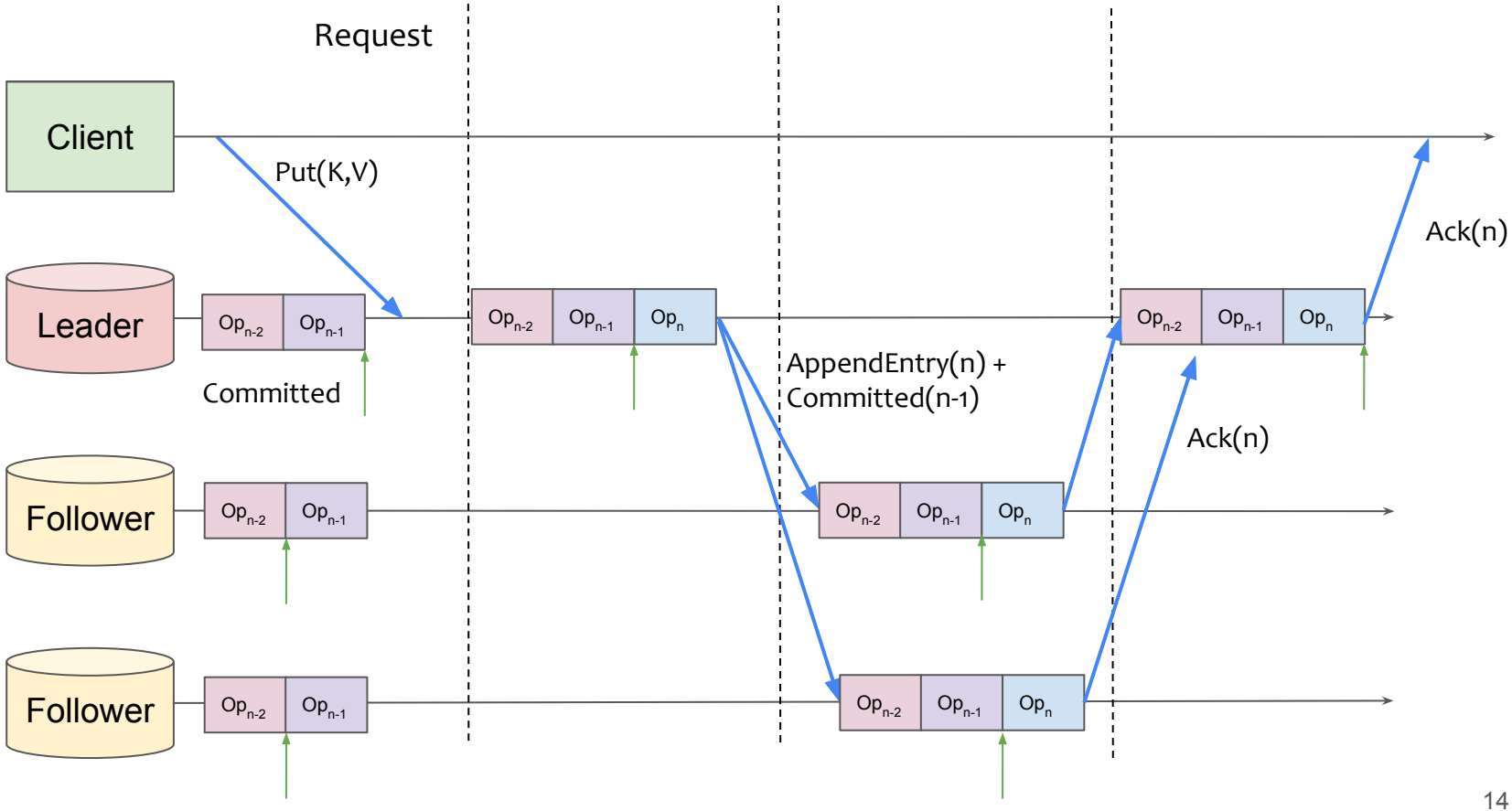
Raft overview

Raft is divided into 3 parts:

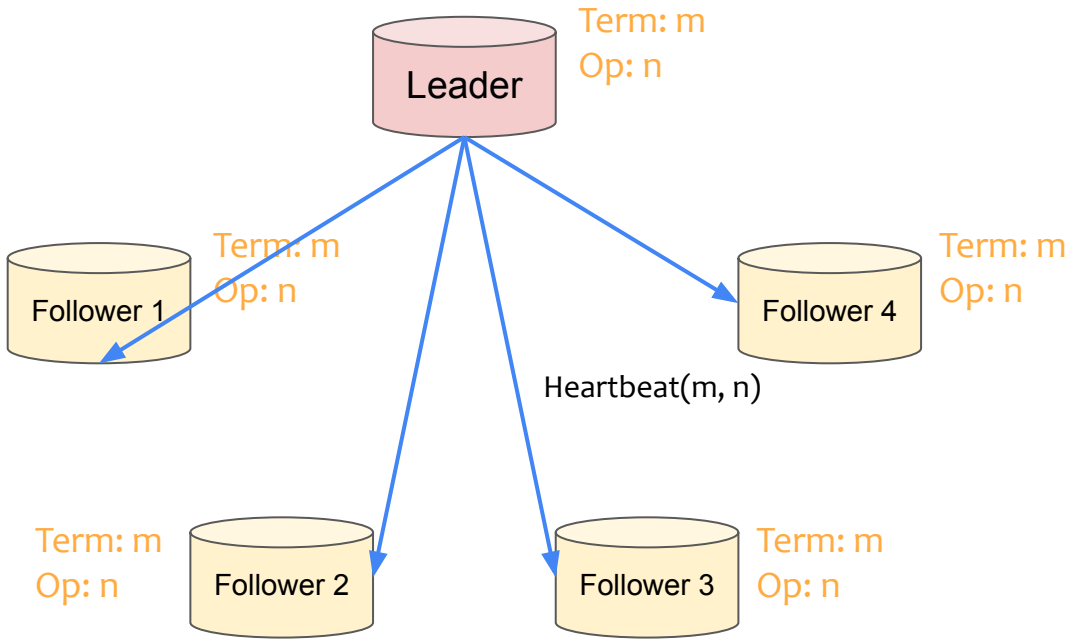
- **Consensus Module**
  - Implementation of the algorithm
  - Log-replication and leader-election
- **Log**
  - Persistent log to serialize all operations
- **State Machine**
  - The state resulting of applying all operations
  - In our case: a key-value store



Raft normal operation

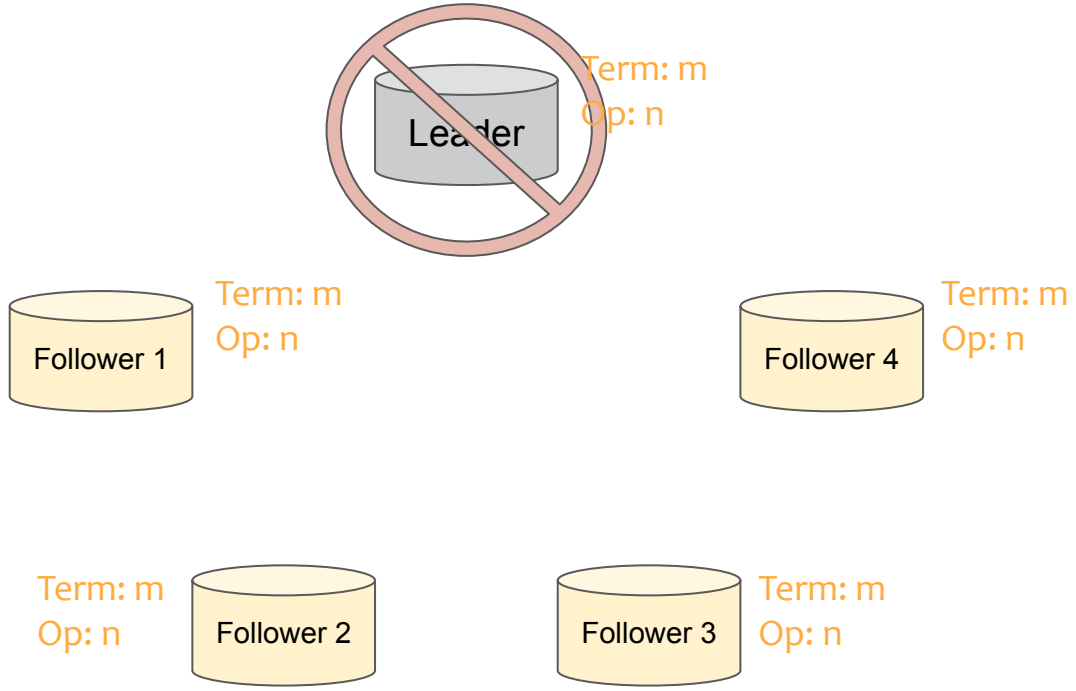


Raft leader election



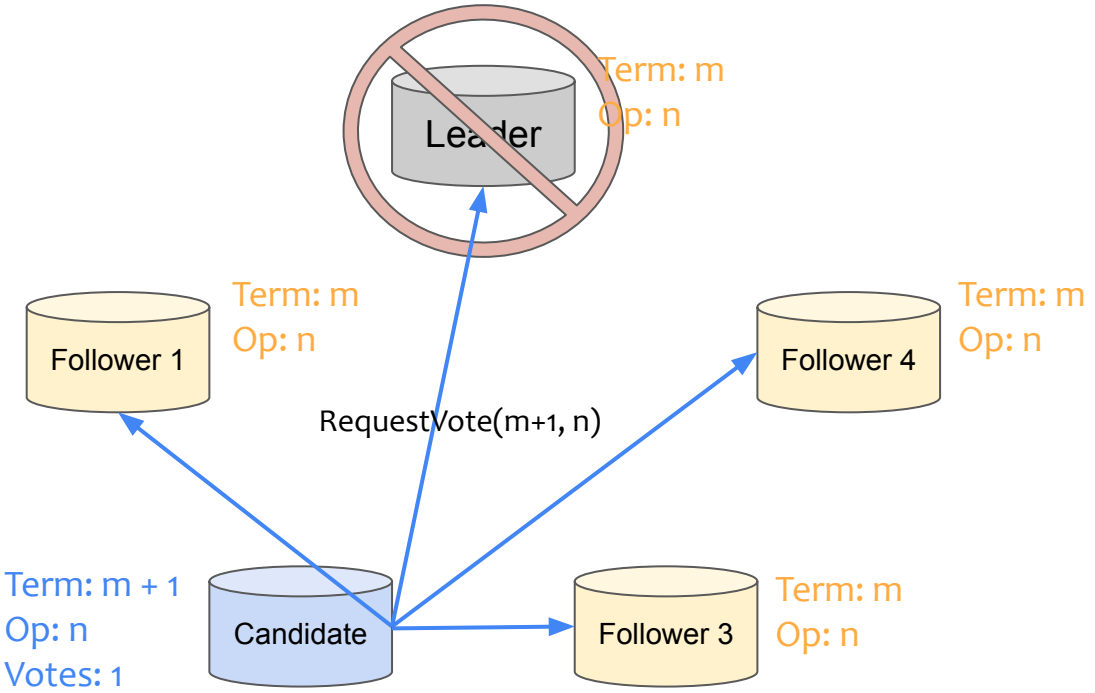
Raft leader election

- Leader goes down
  - Heartbeat timeout
  - Empty AppendEntry



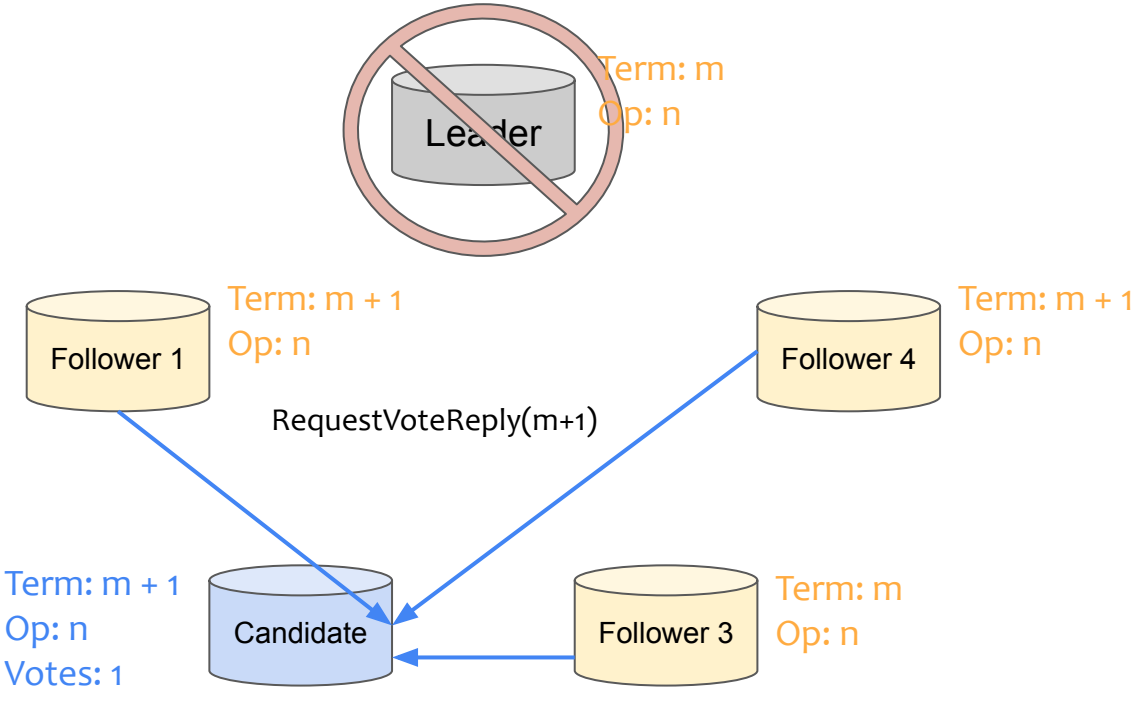
Raft leader election

- Leader goes down
  - Heartbeat timeout
  - Empty AppendEntry
- Follower -> Candidate
  - Increase term
  - Broadcast RequestVote
  - Votes for itself



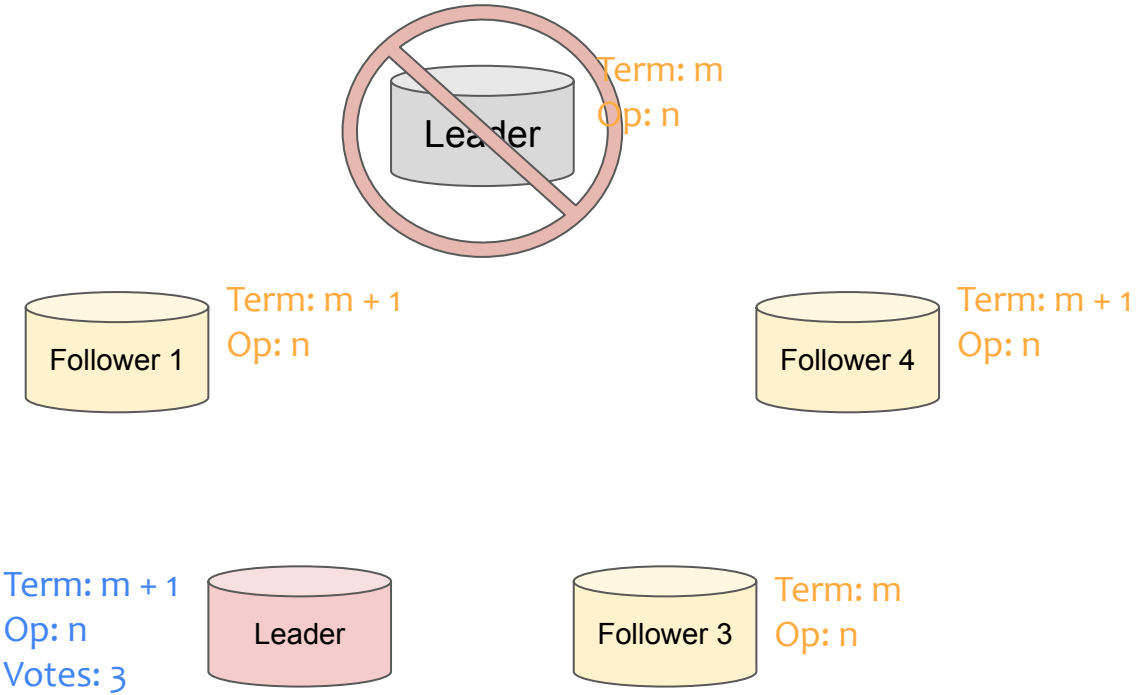
Raft leader election

- Leader goes down
  - Heartbeat timeout
  - Empty AppendEntry
- Follower -> Candidate
  - Increase term
  - Broadcast RequestVote
  - Votes for itself
- Follower vote for 1 candidate
  - First come first serve
  - Iff candidate log is at least as “current” as own log

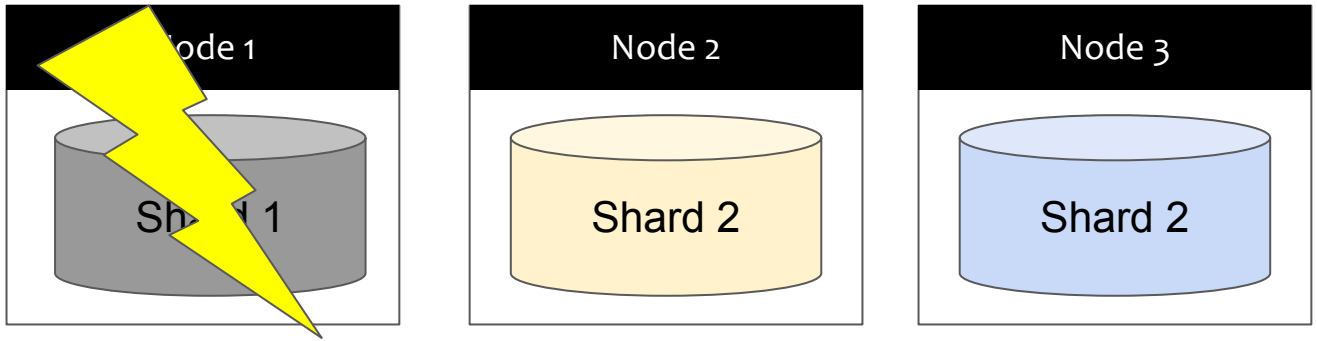


Raft leader election

- Leader goes down
  - Heartbeat timeout
  - Empty AppendEntry
- Follower -> Candidate
  - Increase term
  - Broadcast RequestVote
  - Votes for itself
- Follower vote for 1 candidate
  - First come first serve
  - Iff candidate log is at least as “current” as own log
- Candidate with an absolute majority of votes -> Leader



Sharded KVS

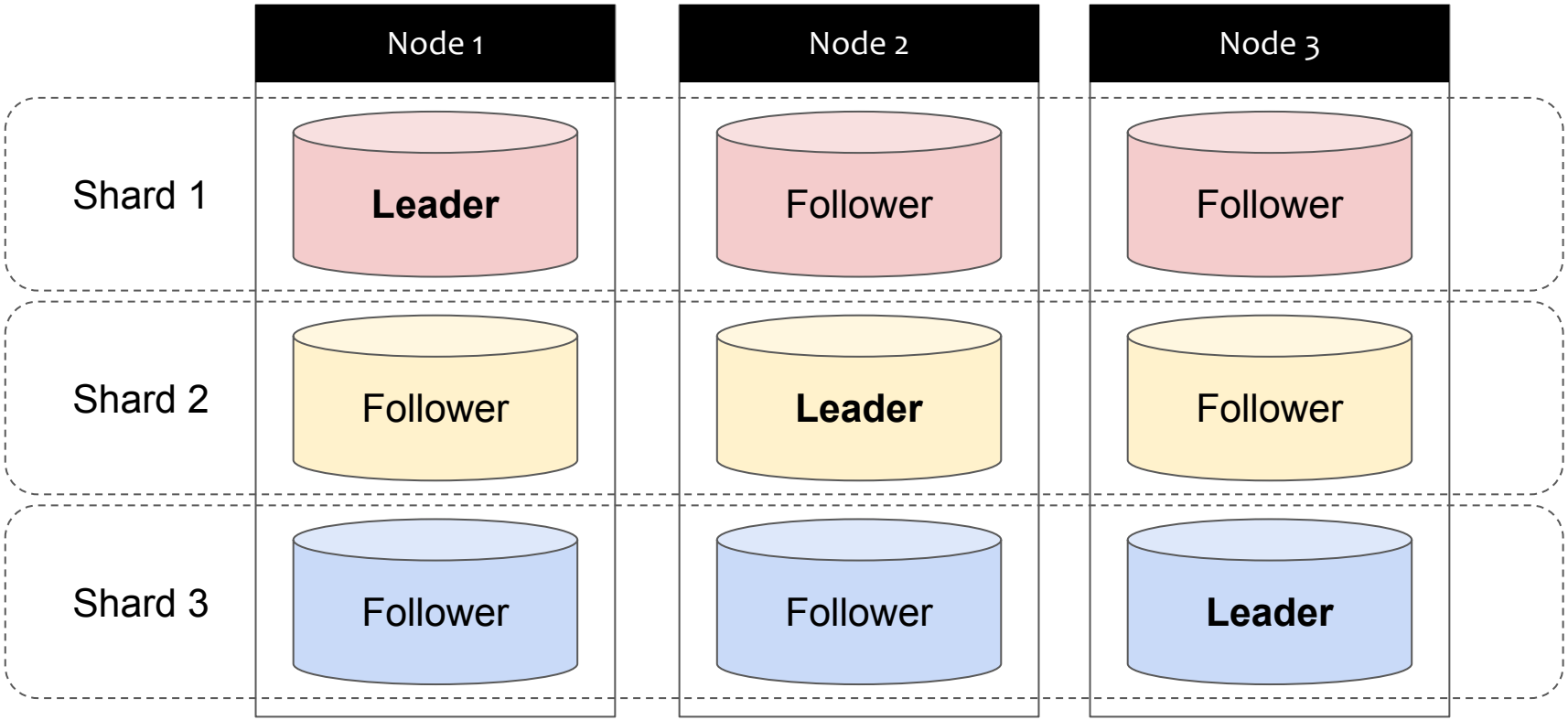


Key value pairs are organized in partitions, distributed across server-nodes

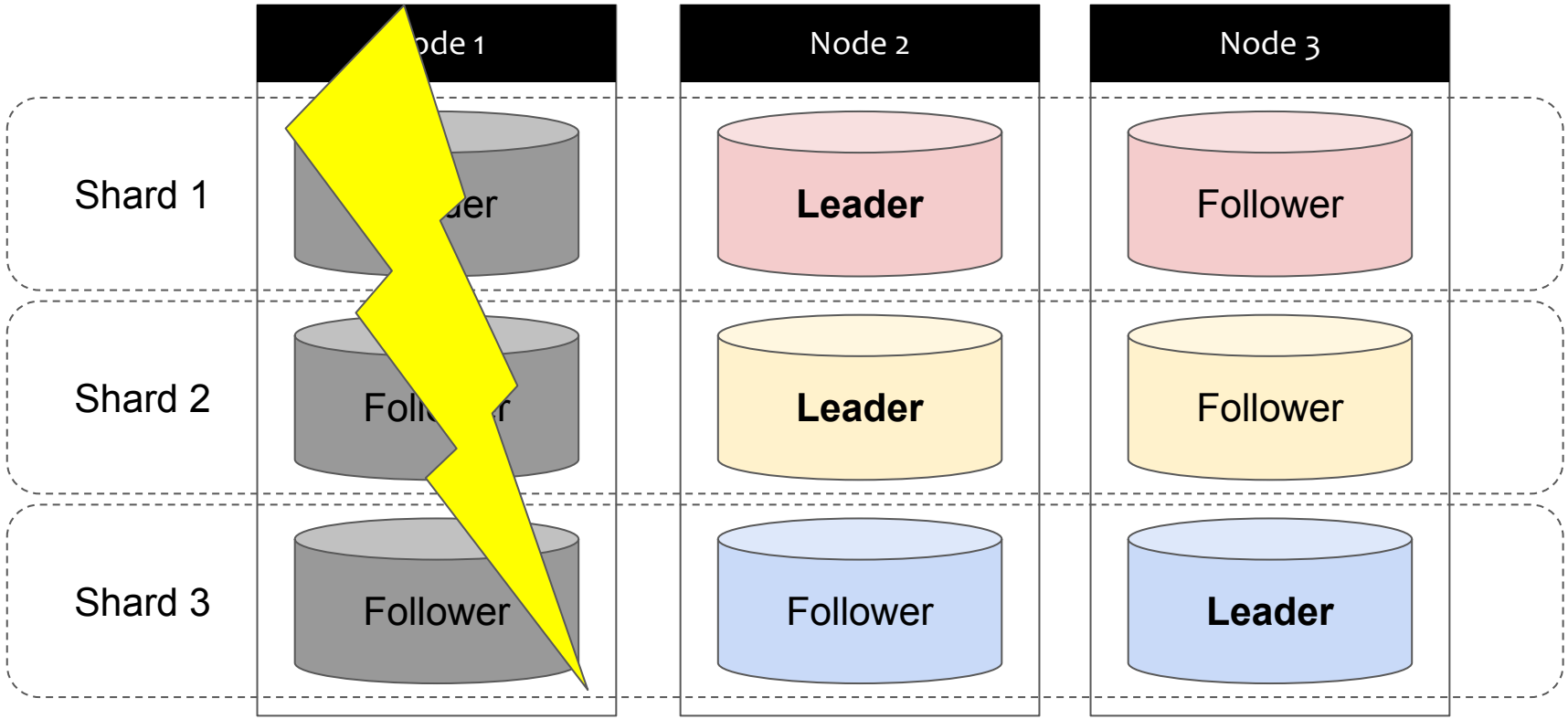
Problem:

If one node fails, all key-value pairs of the respective shard are lost

Replicating shards across nodes



Replicating shards across nodes



Further Reading

- Martin Kleppmann's lectures on Consensus (Lecture 6.1 and 6.2)
  - https://www.youtube.com/watch?v=rN6ma561tak
- The Raft Consensus Algorithm
  - https://raft.github.io/
- In Search of an Understandable Consensus Algorithm (Extended Version)
  - https://raft.github.io/raft.pdf

Task #3

Make your current distributed key-value store fault-tolerant:

1. Implement a failure-detection system using heartbeat
   a. Pings all servers in the cluster to see if they are still alive
2. Implement leader election
   a. A new leader must be chosen when an existing leader fails
3. Log replication
   a. The leader must accept log entries from clients and replicate them across the cluster