Practical Lab

Cloud Systems Engineering

(cloud-lab)

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

Task #3:

Replicated distributed KVS

Task #3

Implement the Raft replication protocol and creating replicated KVS

- Implement normal operations
- Implement leader election

Background

Learning goals

In this task you will learn about:

- Motivations of replication
- Leader-based replication
    - Raft

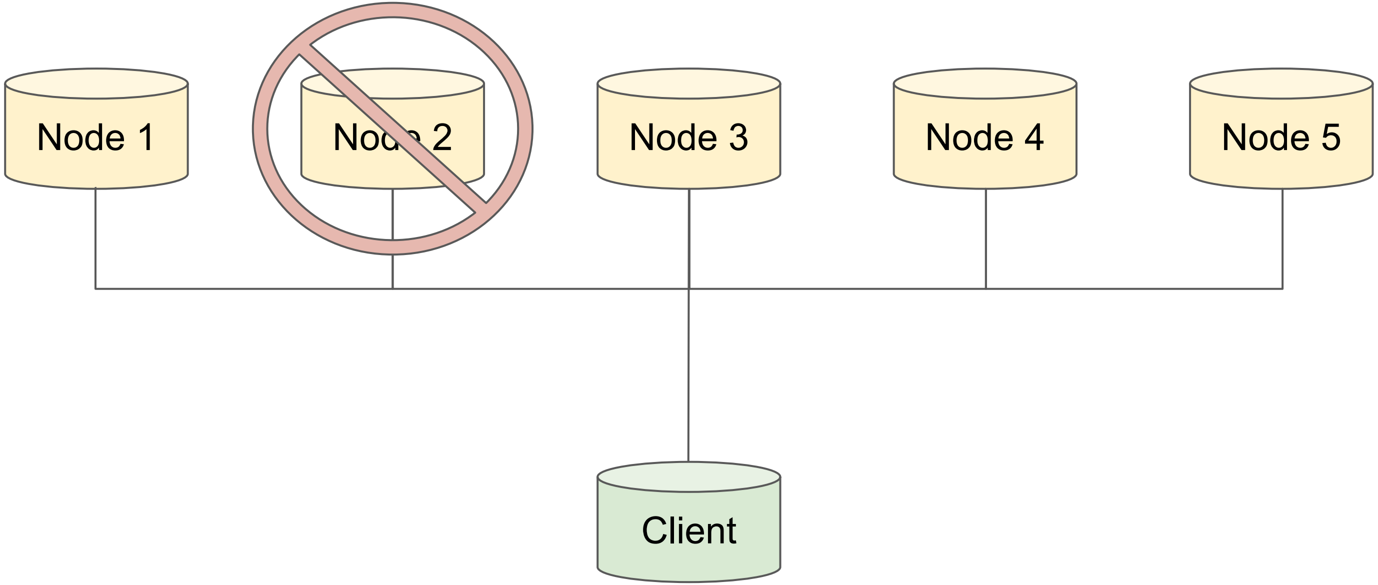
Why distribute a single-node KVS?

- Fault-tolerance
  - A single node is a single point of failure
  - If this node fails, the system become unavailable
  - **Solution:** replication (this task !)
- 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!)

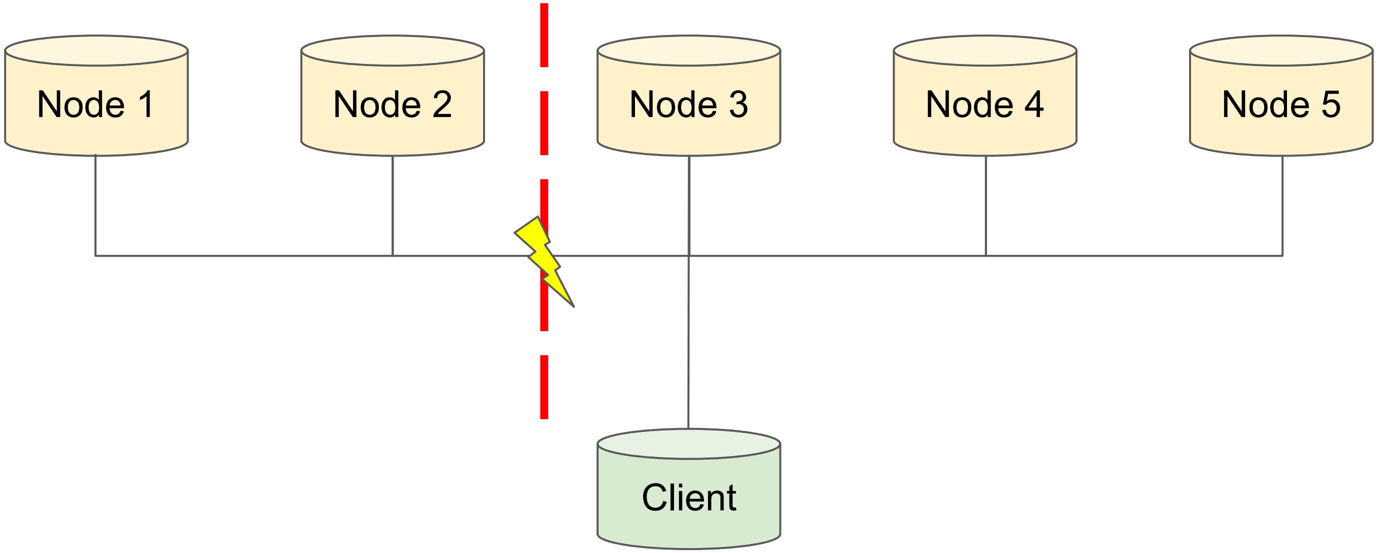
Faults are inevitable

- Programming error
- Configuration error
- Soft-errors
  - Gamma ray
  - Voltage fluctuations
- Hard-errors
  - Burned silicone
  - Failed hard-drive
- Can happen at any time

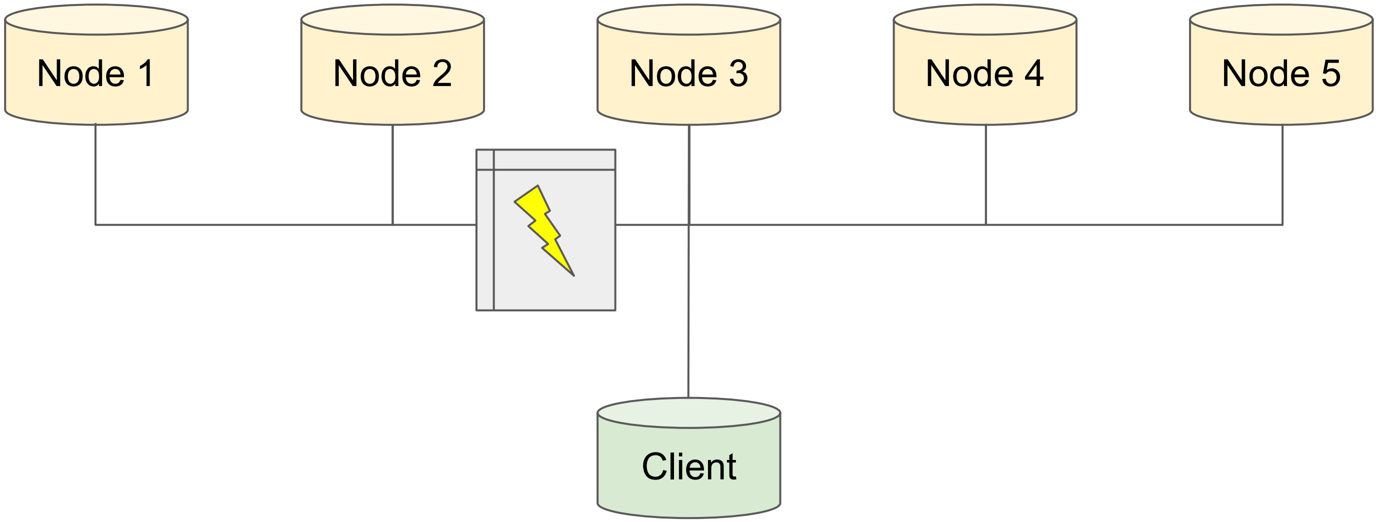
Faults Example (⅓): Node-failure



Faults Example (2/3): Network partition

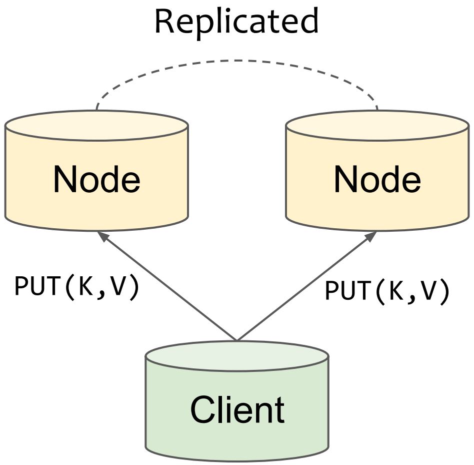


Faults Example (3/3): Packet loss



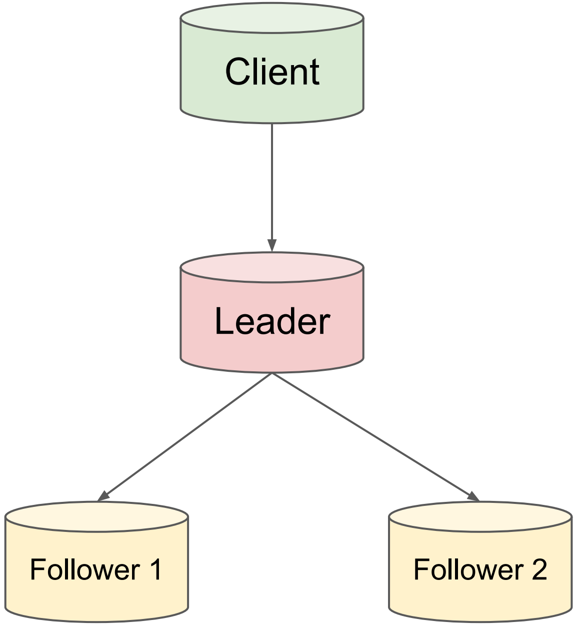
Naive solution to failures

- **Replicates operation** on multiple nodes
  - Clients can access any available node
- Problem
  - Replicas see operations at different times
    - Stale data
    - Conflicts
    - Diverging state (network partition)
- More elaborate replication strategy is needed
  - **Leader-based replication** (this task)
  - Leader-less replication
  - BFT

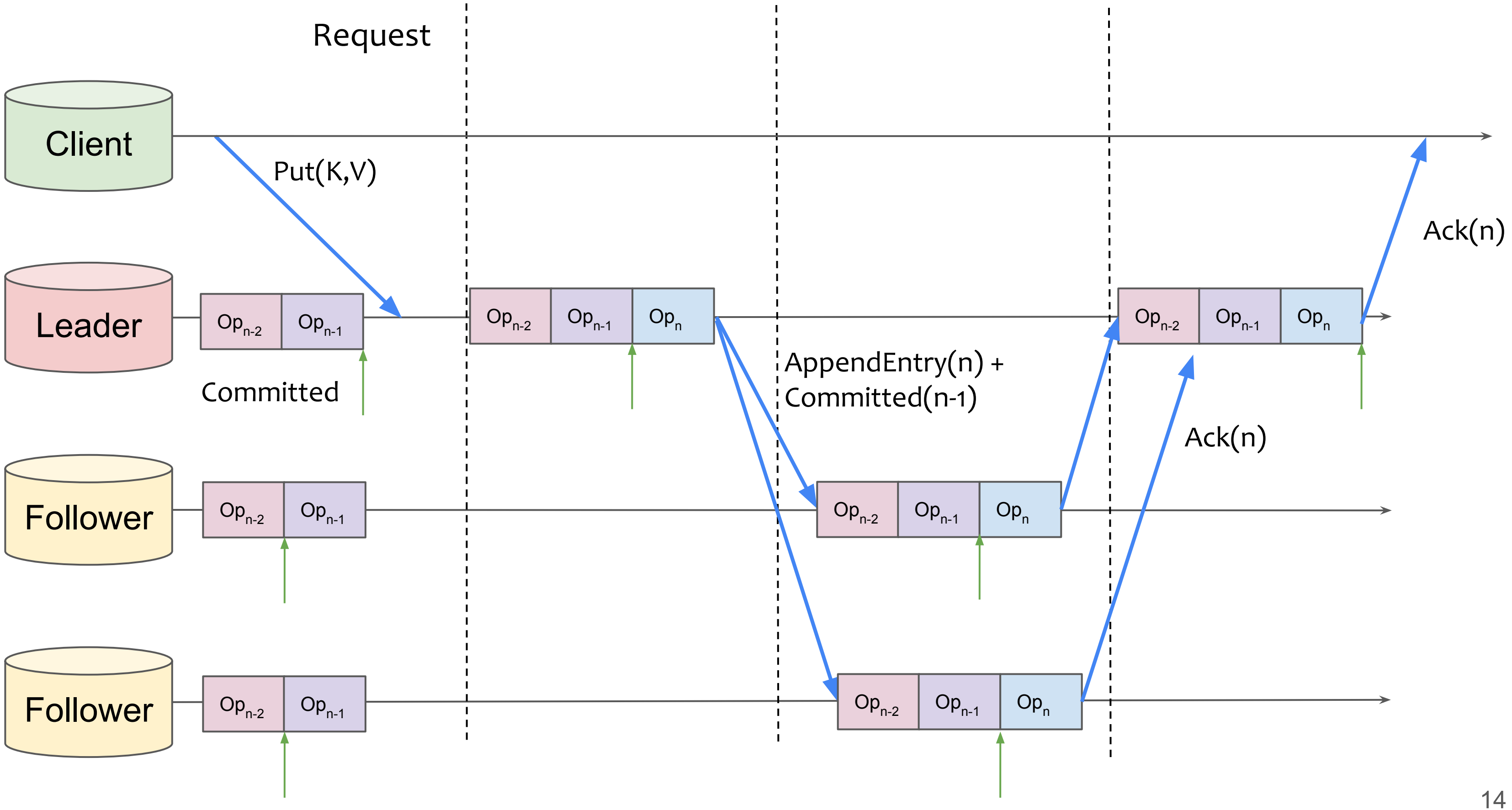


Leader-based Replication

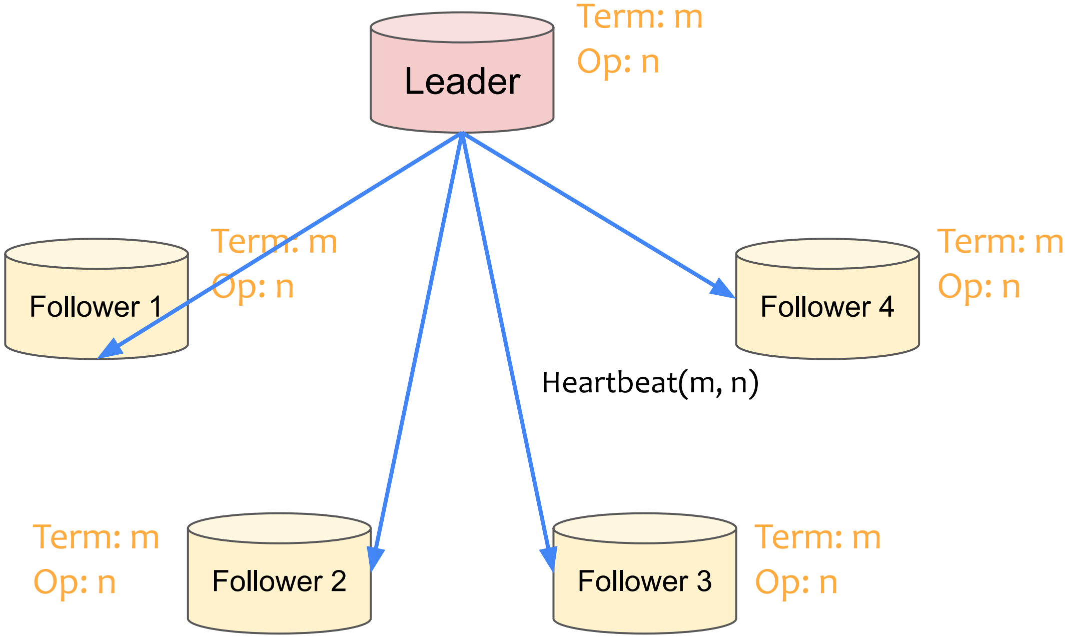
- One Leader
  - Communicates with the client
  - Determines order of operation
  - Orchestrate replication in all nodes
- Multiple followers
  - Do not communicate with client
  - Follow the commands of the leader
  - Involved in leader election
- Examples
  - **Raft**
  - Zab
  - Chain replication



Raft normal operation

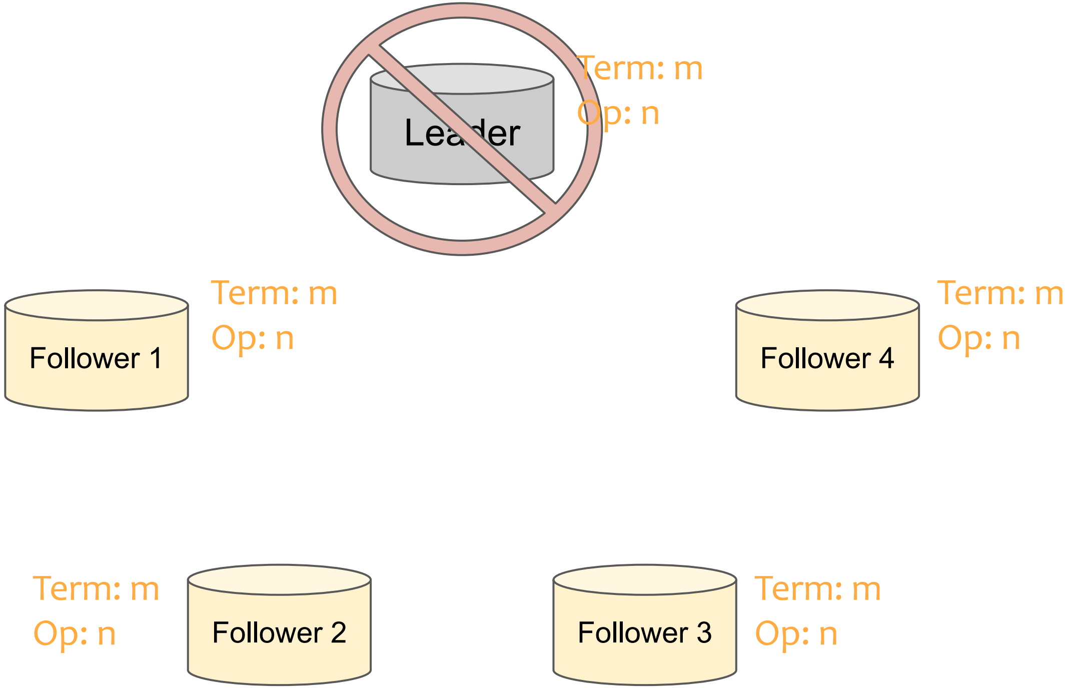


Raft leader election



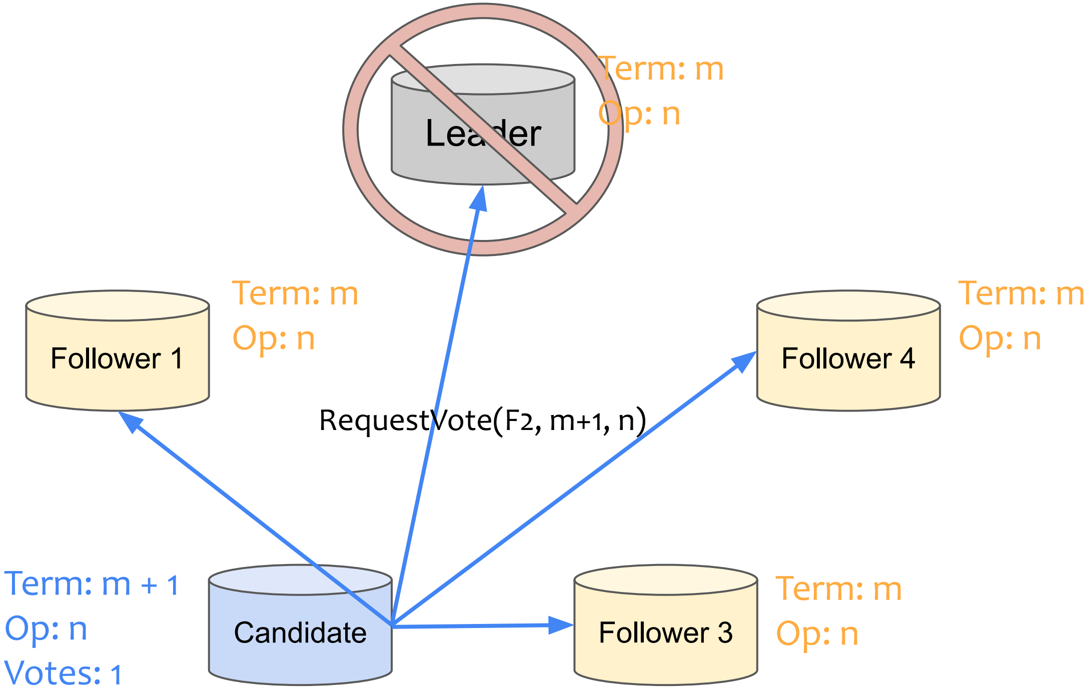
Raft leader election

- Leader goes down
  - Heartbeat timeout
  - Empty AppendEntry



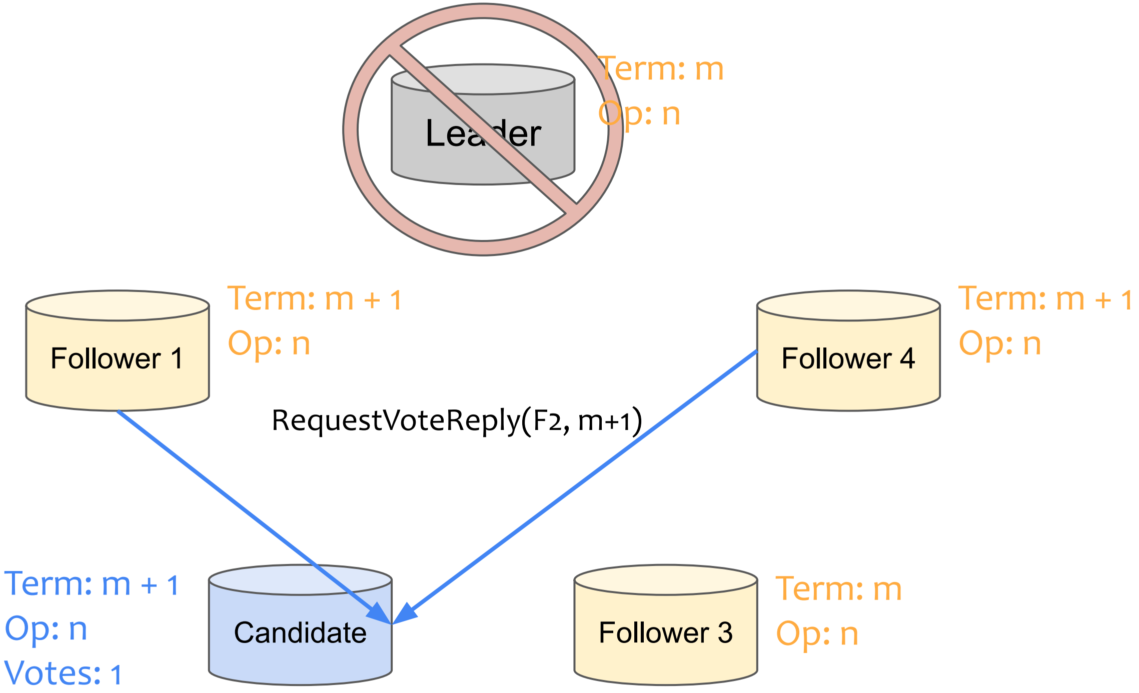
Raft leader election

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



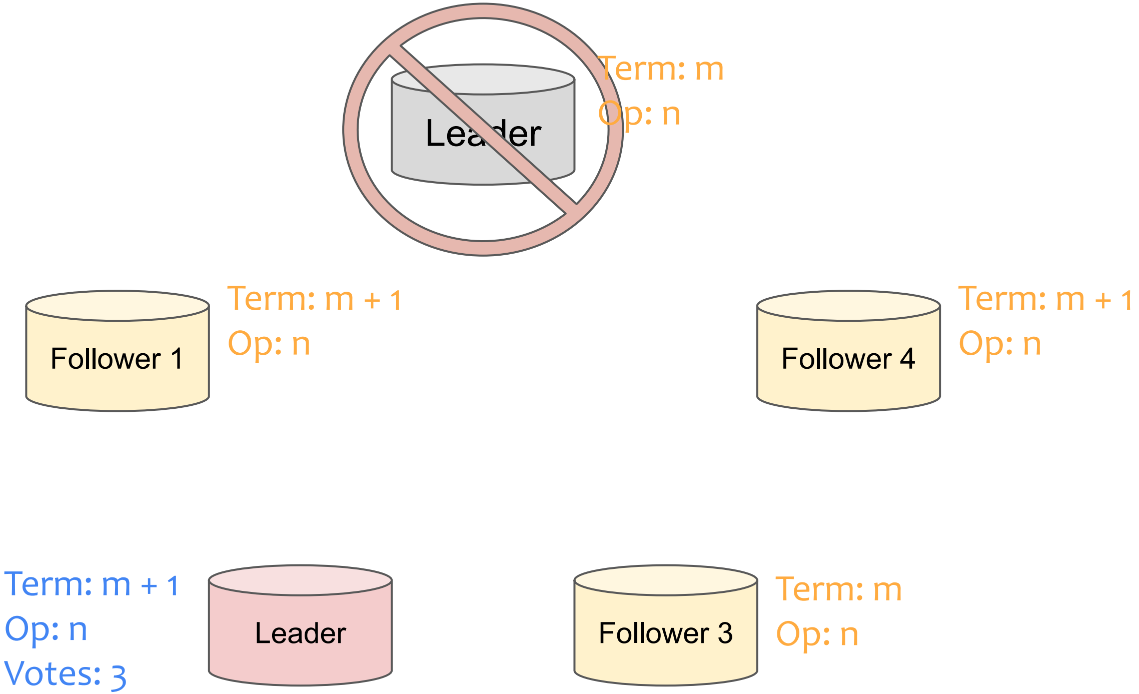
Raft leader election

- 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



Task #3

1. Implement Raft normal operation
    - AppendEntries RPC
2. Implement Raft leader election
    - RequestVote RPC

References

- The Raft Consensus Algorithm
  - https://raft.github.io/
- In Search of an Understandable Consensus Algorithm (Extended Version)
  - https://raft.github.io/raft.pdf
  - See Figure 2. for the summary of the Raft algorithm