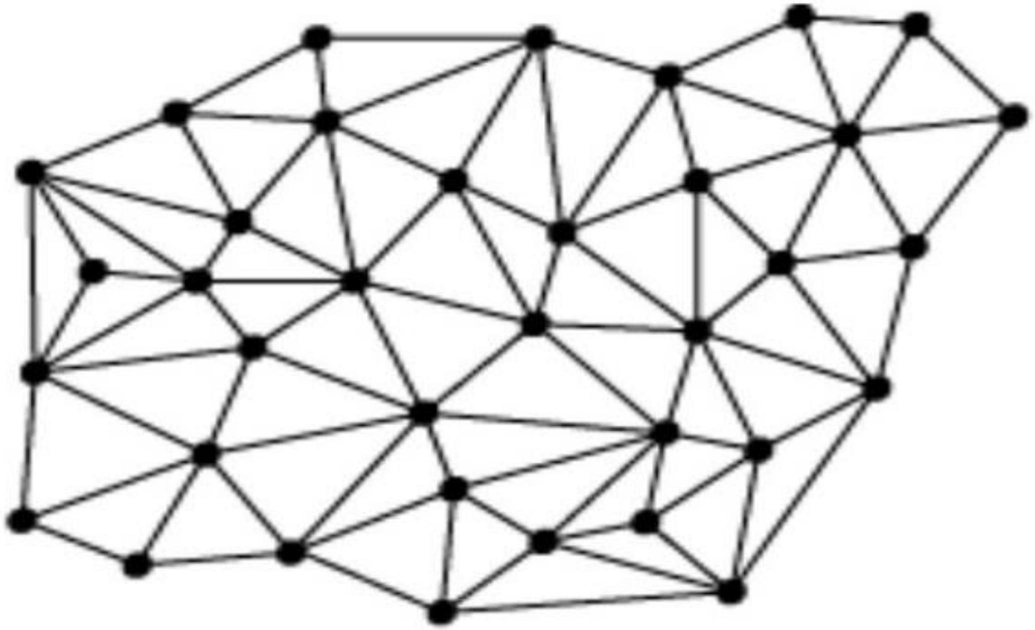
Distributed System Course

2021-22-GICI41SSD-Distributed System



Academic Year: 2021-2022 Lecturer: SOK Kimheng

Information

Course	**Distributed System**	**48h, 12 Weeks, 4h/week (3 Groups = 96h)**
	Week 1	Information, Self-Study Skill, Introduction
General Distributed System	Week 2	Distributed Communication (TCP/IP, Socket, RPC, REST, gRPC, OMQ)
	Week 3	Clock, Timestamp
	Week 4	Fault Tolerance (Two general problem, Byzantine General Problem)
	Week 5	Consensus Algorithm (Paxos, ZooKeeper, Raft)
	Week 6	**Quiz**
Blockchain	Week 7	Basic Cryptography
	Week 8	Blockchain and Bitcoin (Proof of Work)
	Week 9	Ethereum and Smart Contract (Proof of Stake)
	Week 10	Hyperledger and Self-Sovereign Identity
	Week 11	Security
	Week 12	**Final Exam**

Distributed System Course

2021-22-GICI41SSD-Distributed System

Week5:

Consensus Algorithm

Paxos, Zookeepr, Raft

Academic Year: 2021-2022 Lecturer: SOK Kimheng

Agenda

1. Definition
2. Case study
3. Paxos
4. ZooKeeper
5. Raft

Consensus Algorithm

Definition

- A general agreement
- A collectively agreement on the same output by all nodes | peers within the distributed system

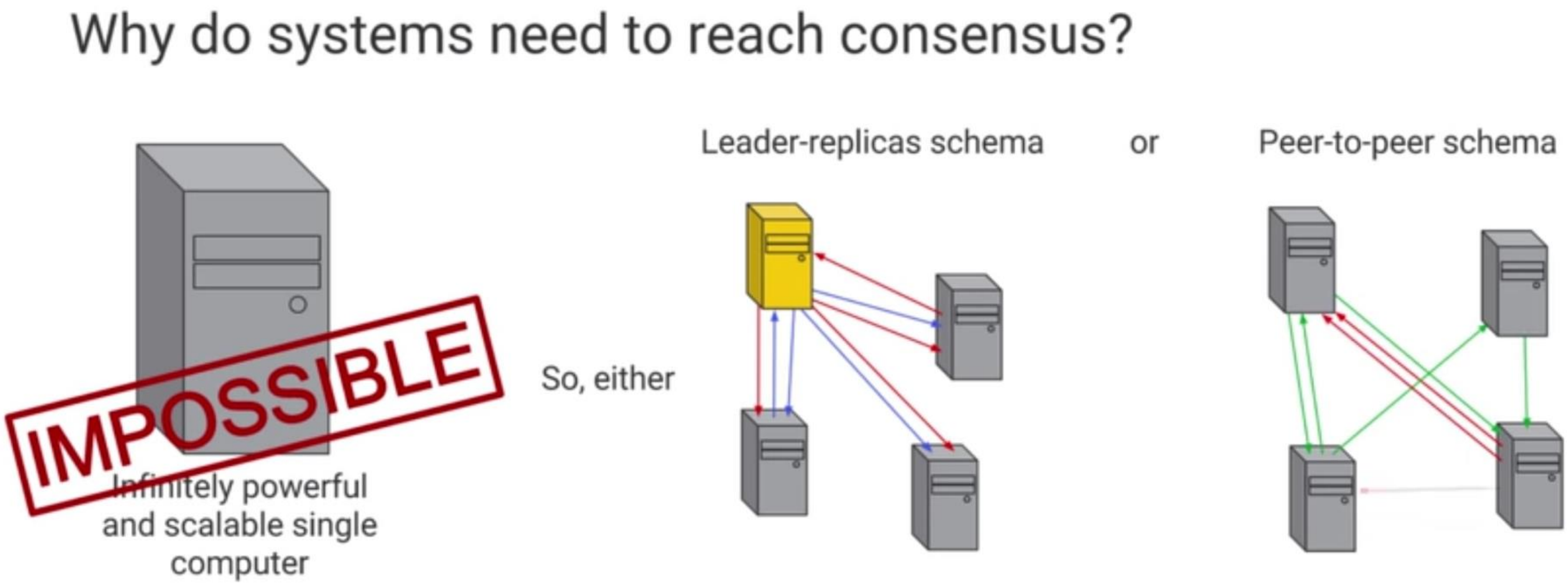
Consensus Algorithm

Case study

1. A group of students decide to make a group T-shirt
2. Choosing T-shirt color.
   - ❖Let's see who is initiator, proposer, leader
3. Finding members, Creating rule
   - ❖Split the group (divide, fork) , or unite by majority
4. Asking confirmation
   - ❖Number of confirmation received
5. Commit the decision
   - ❖What is the final color?
6. Leader term (Communism or Democracy?)

Consensus Algorithm

Case study



Why do systems need to reach consensus?

IMPOSSIBLE

Infinitely powerful and scalable single computer

So, either

Leader-replicas schema

or

Peer-to-peer schema

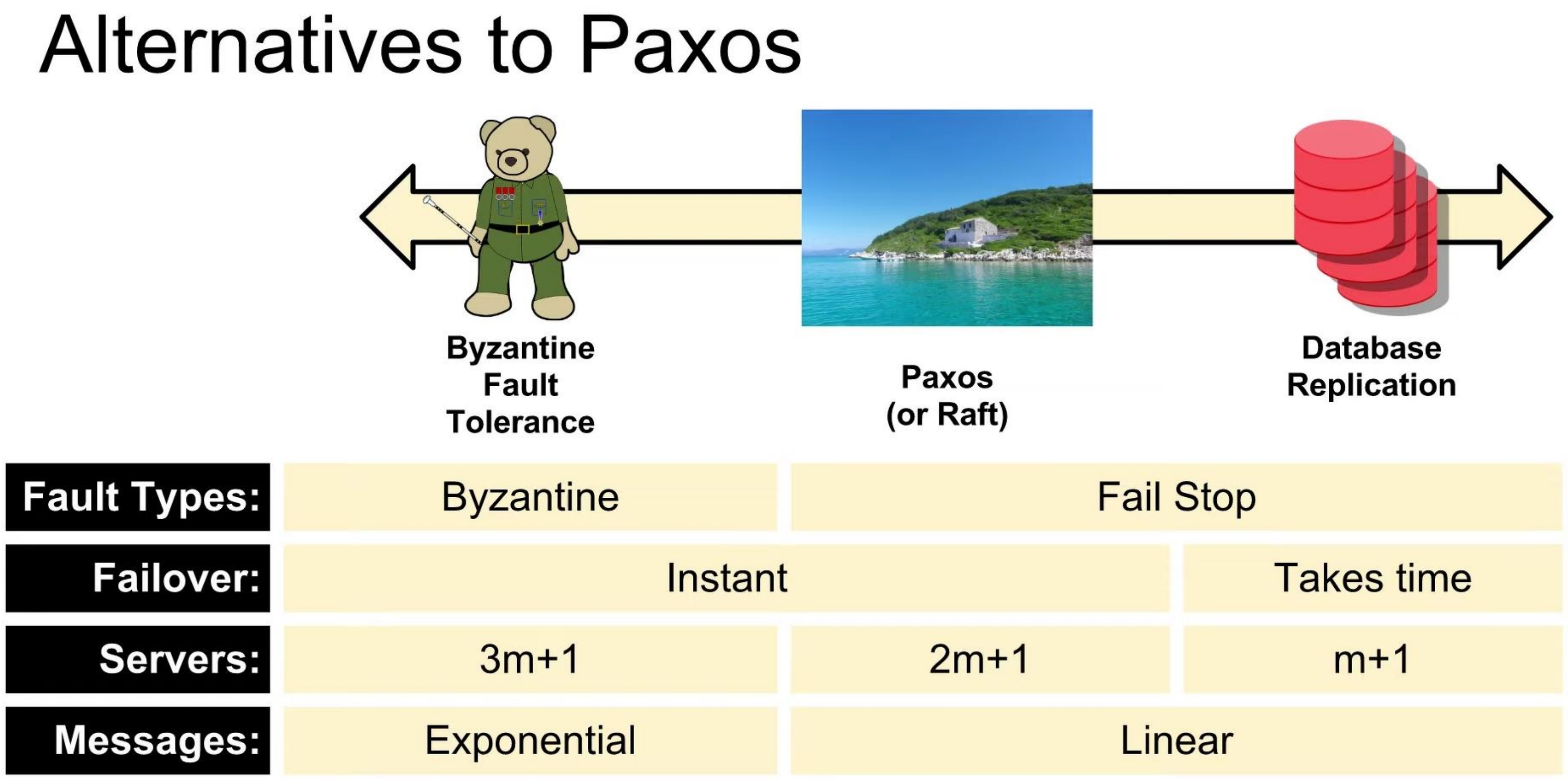
Consensus Algorithm

PAXOS (1989, 2001)

- Consensus is agreement on one result
- Once a majority agrees on a proposal, that is the consensus
- The reached consensus can be eventually known by everyone
- Paxos defines 3 roles: Proposers, Acceptors, and Learners
- Paxos nodes can take multiple roles, even all of them
- Paxos nodes must know how many acceptors a majority is
- Paxos nodes must be persistent: they can't forget what they accepted

Consensus Algorithm

PAXOS (1989, 2001)



Alternatives to Paxos

Byzantine Fault Tolerance

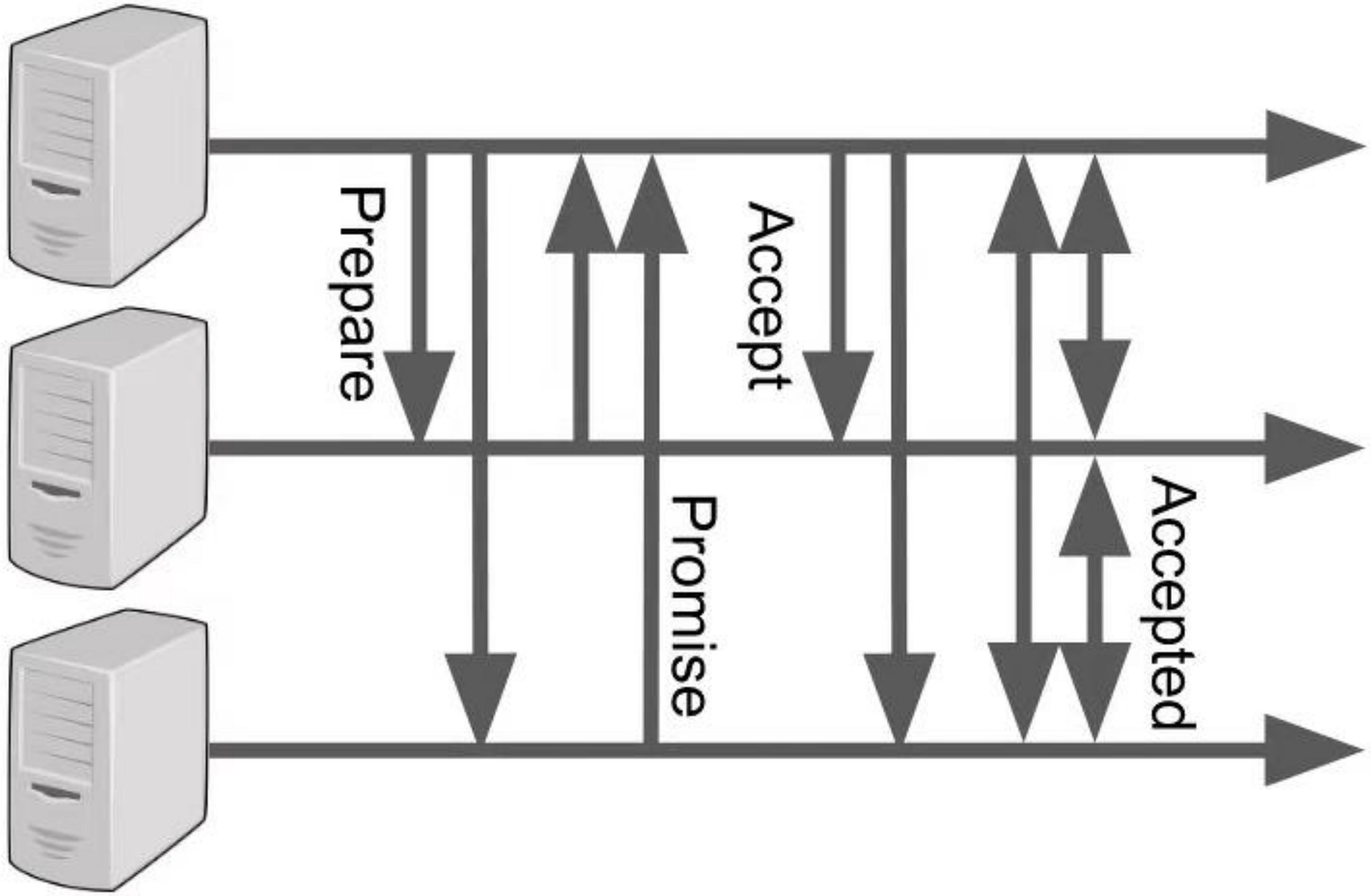
Paxos (or Raft)

Database Replication

Fault Types:	Byzantine	Fail Stop	
Failover:	Instant		Takes time
Servers:	3m+1	2m+1	m+1
Messages:	Exponential	Linear	

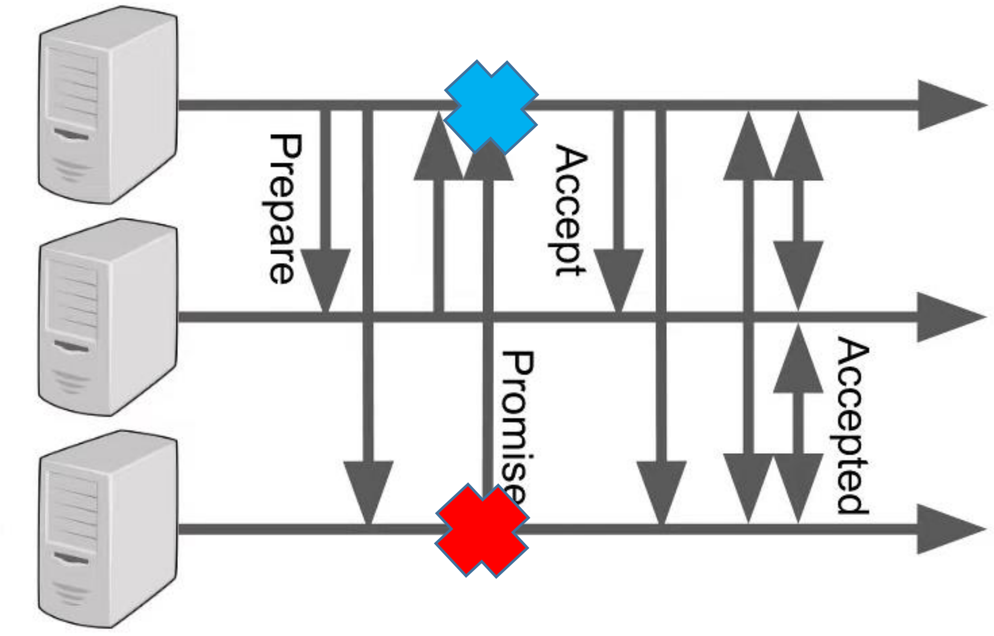
Consensus Algorithm

PAXOS (1989, 2001)



Consensus Algorithm

PAXOS (1989, 2001)

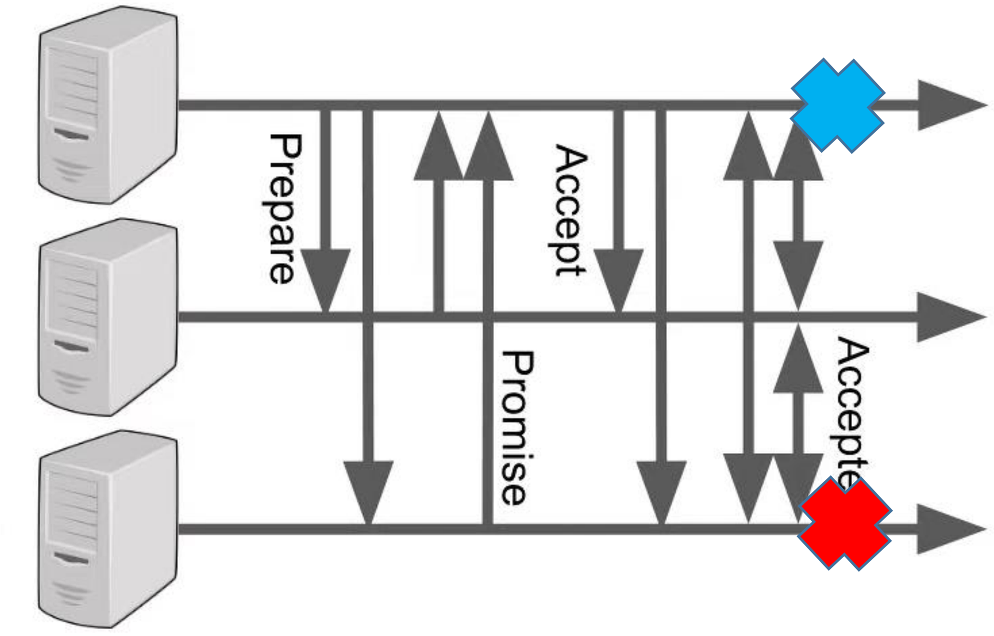


fault

- ➢One node fail to send back promise
- ➢Count the majority

Consensus Algorithm

PAXOS (1989, 2001)

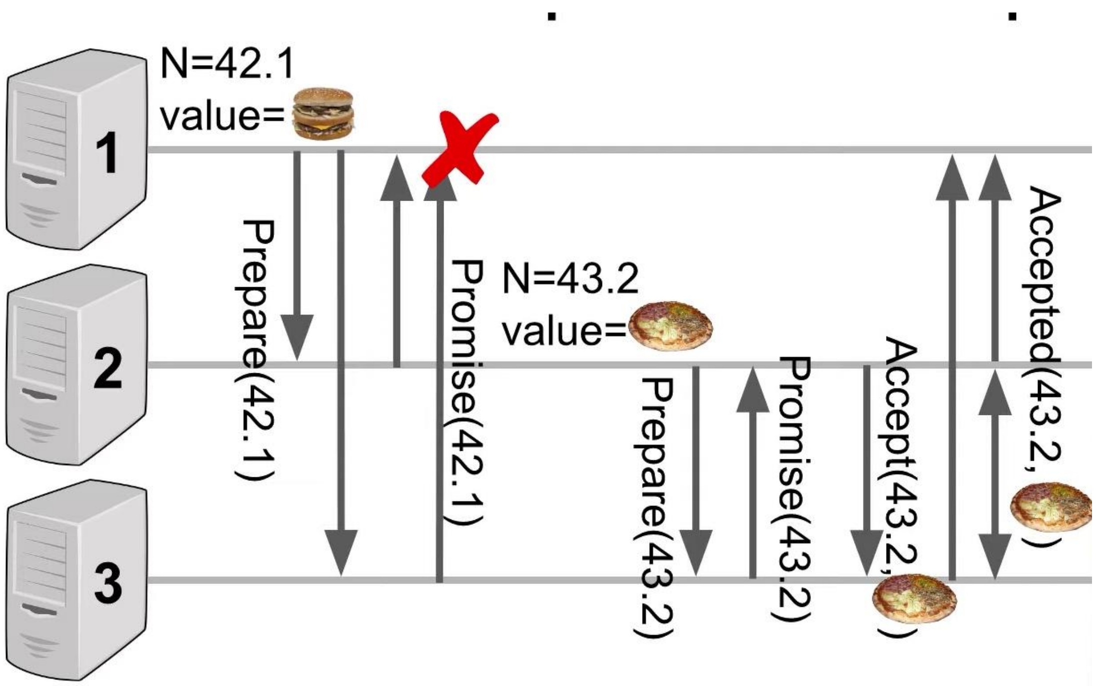


fault

- One node fail to send back Accepted
- Count the majority

Consensus Algorithm

PAXOS (1989, 2001)

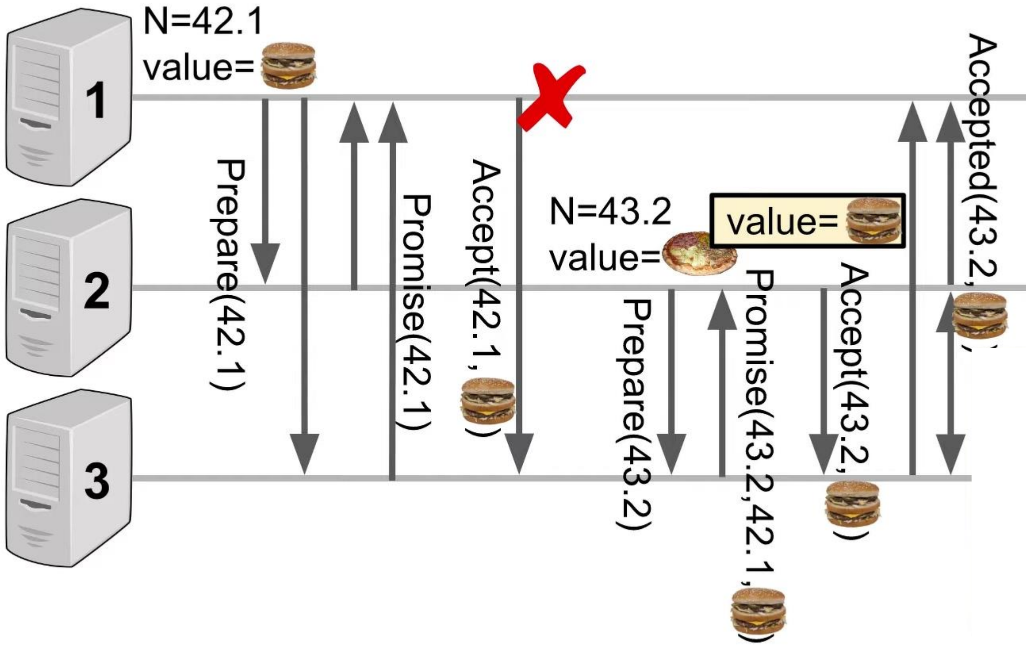


fault

- ➢ First node fail after Prepare
- ➢ Second node make new prepare
- ➢ New decision is accepted

Consensus Algorithm

PAXOS (1989, 2001)



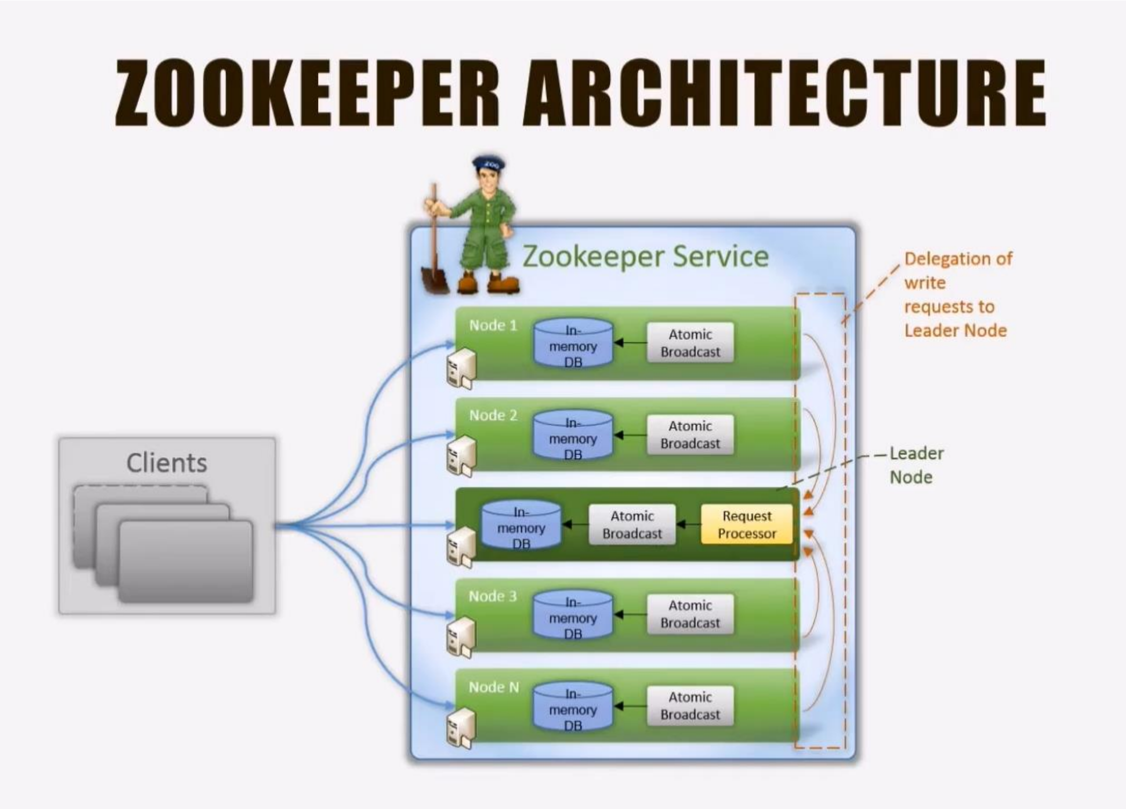
fault

- First node fail during Accept
- Second node make new prepare
- Third node inform second node about previous value
- Second and Third nodes agree for the previous value

Consensus Algorithm

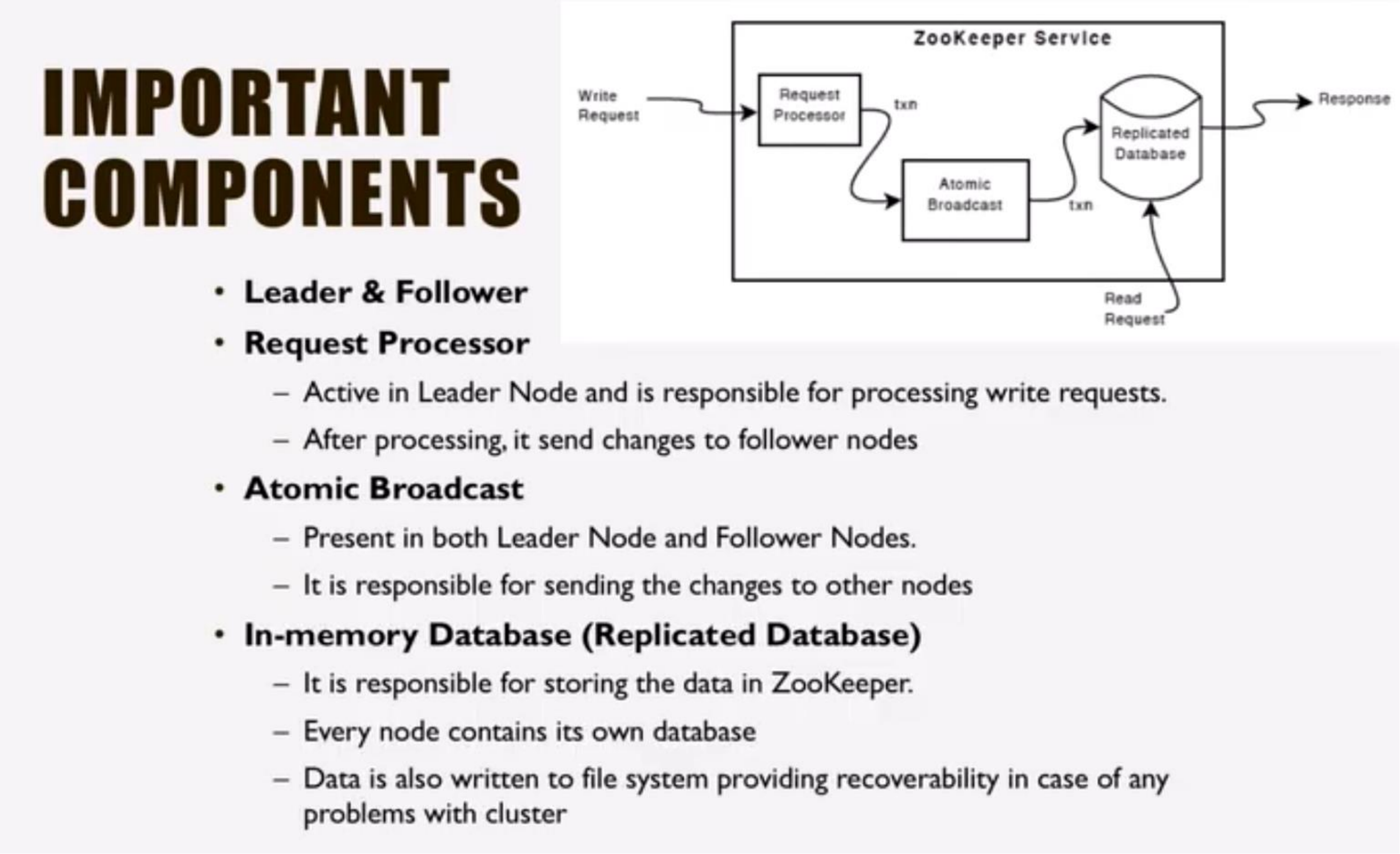
ZooKeeper

- Distributed and Open-source coordination service for decentralize applications



Consensus Algorithm

ZooKeeper

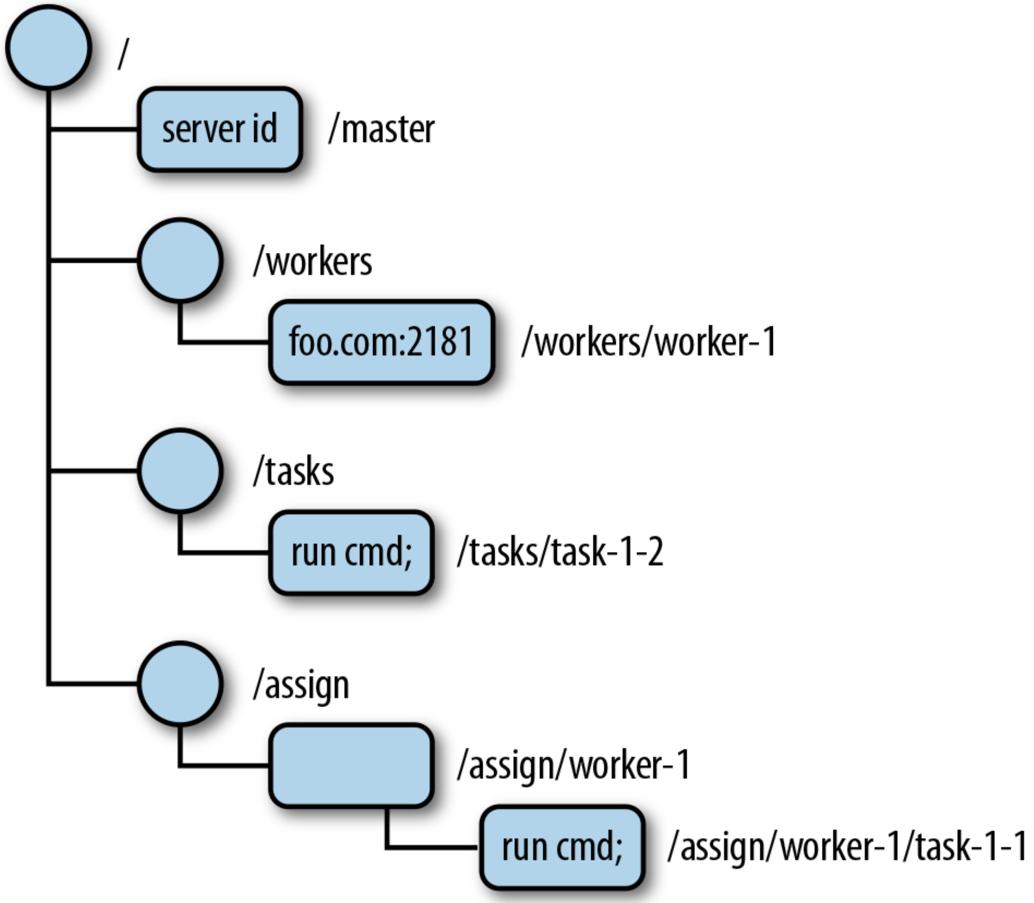


IMPORTANT COMPONENTS

- **Leader & Follower**
- **Request Processor**
  - Active in Leader Node and is responsible for processing write requests.
  - After processing, it send changes to follower nodes
- **Atomic Broadcast**
  - Present in both Leader Node and Follower Nodes.
  - It is responsible for sending the changes to other nodes
- **In-memory Database (Replicated Database)**
  - It is responsible for storing the data in ZooKeeper.
  - Every node contains its own database
  - Data is also written to file system providing recoverability in case of any problems with cluster

Consensus Algorithm

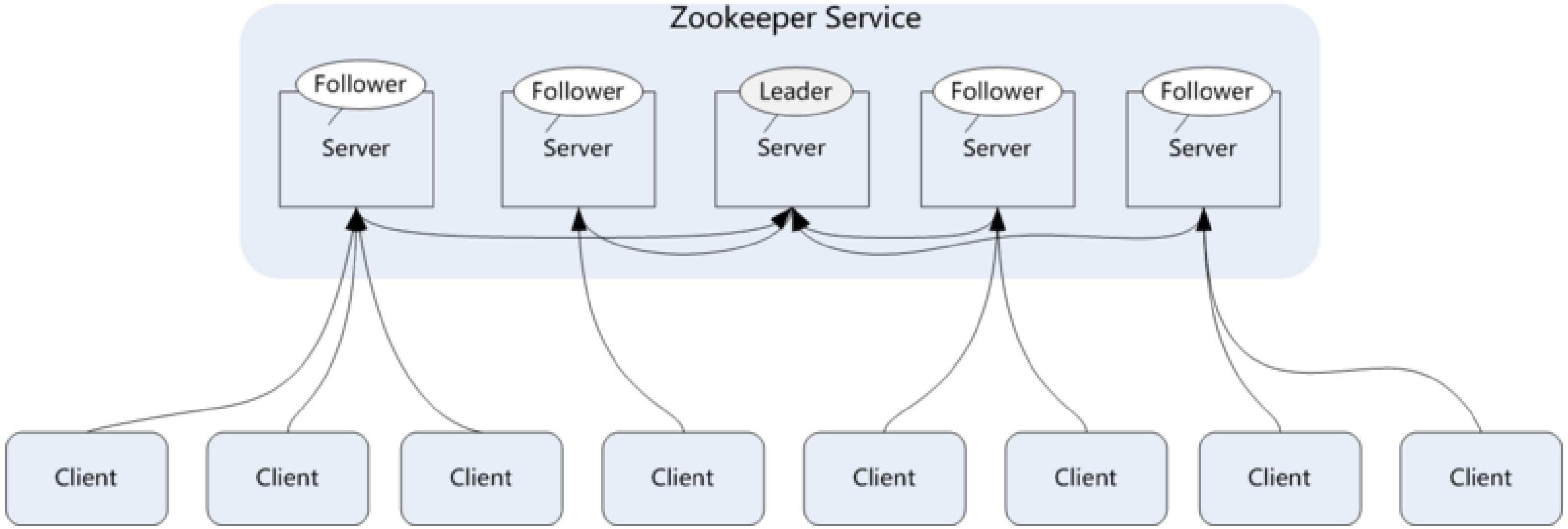
ZooKeeper file structure



Consensus Algorithm

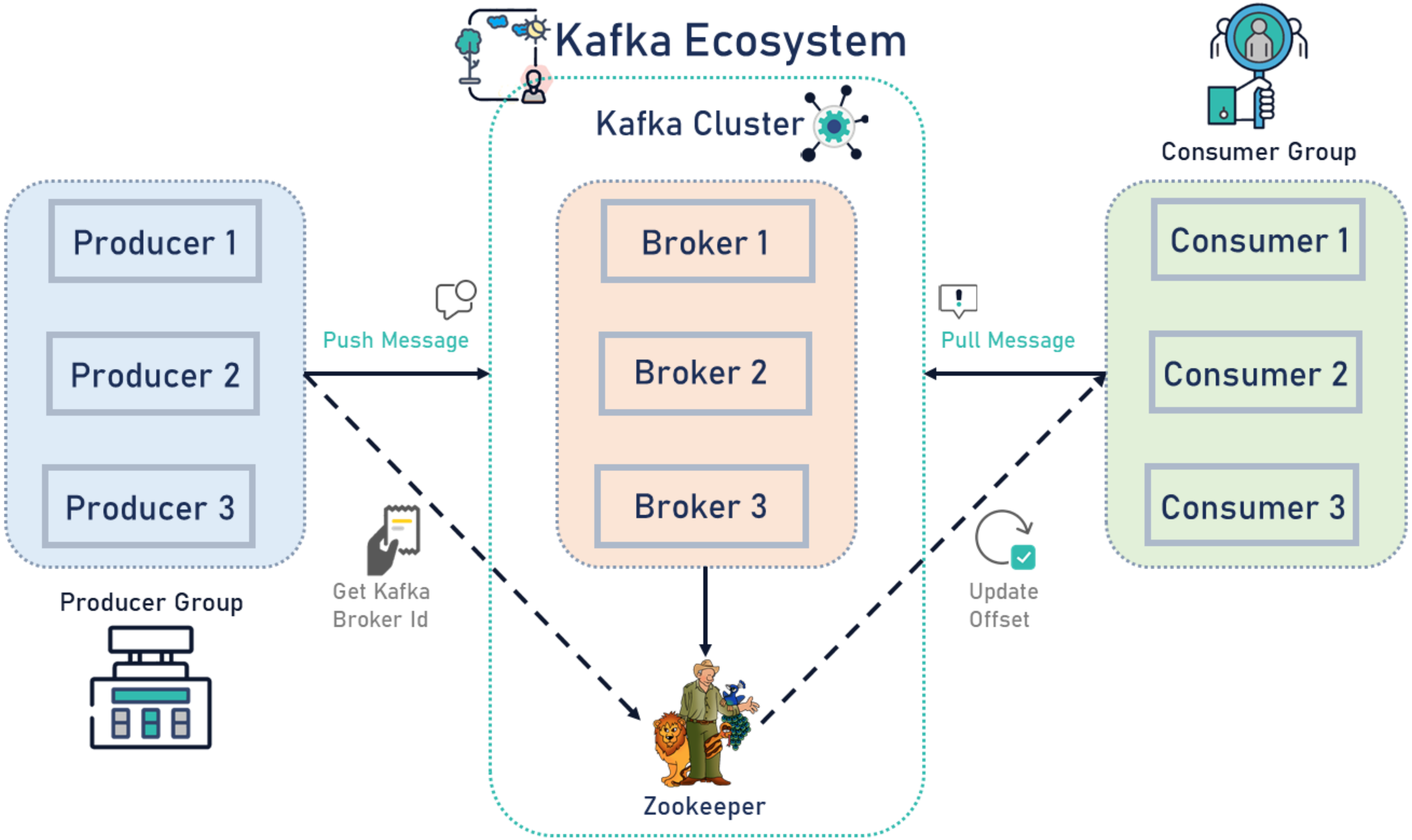
ZooKeeper

- ZooKeeper Maintains
    - Version number
    - ACL
    - Time Stamp
    - Data Length



Consensus Algorithm

ZooKeeper Problem



Consensus Algorithm

Raft (Reliable | Replicate | Redundant and Fault Tolerant)

- Leader Election
  - Select one server to act as leader
  - Detect crashes, choose new leader
- Log Replication
  - Leader accepts commands from clients, appends to its log
  - Leader replicates its log to other servers (overwrite if inconsistency)
- Safety
  - Keep logs consistent
  - Only servers with up-to-date logs can become leader

Consensus Algorithm

Raft (Reliable | Replicate | Redundant and Fault Tolerant)

- Roles / States
  - Follower, Candidate, Leader
- Election Term
  - At most 1 leader per term
  - Some term has no leader (Failed election)
  - Each server maintains current term value (No global view)
  - Exchange communication through RPC
  - If Leader encounters latest term, it updates its term and revert to follower
  - If incoming RPC has obsolete term, reply with error

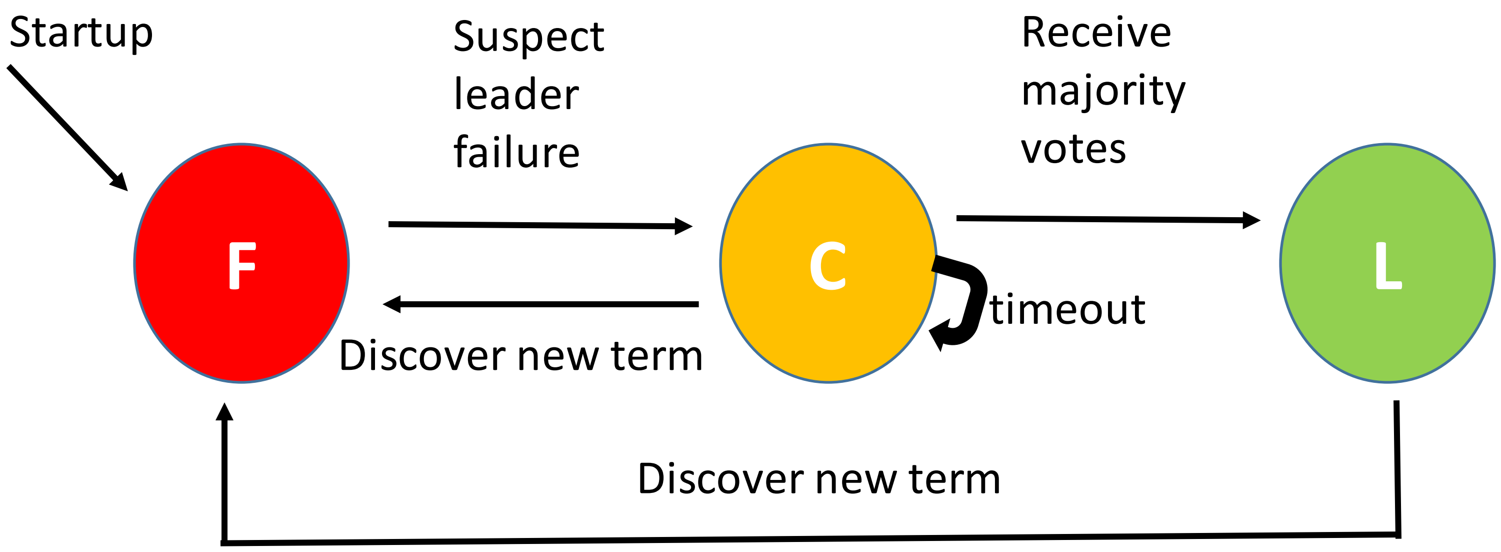
Consensus Algorithm

Raft (Reliable | Replicate | Redundant and Fault Tolerant)

- ➢Election Process
  - ➢Every node start with follower state
  - ➢Each nodes initiate a random election timeout
  - ➢If timeout, node changes from follower to candidate state, and start asking for vote
  - ➢If candidate received majority votes, it changes from candidate to leader state and start sending heartbeat to other nodes
  - ➢Leader will update the latest log to all the followers

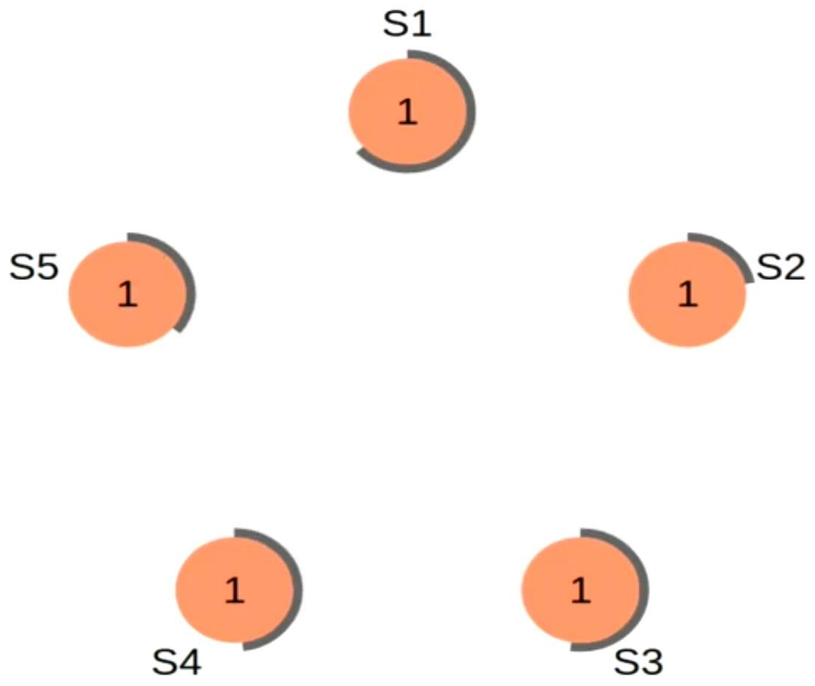
Consensus Algorithm

Raft (Reliable | Replicate | Redundant and Fault Tolerant)



Consensus Algorithm

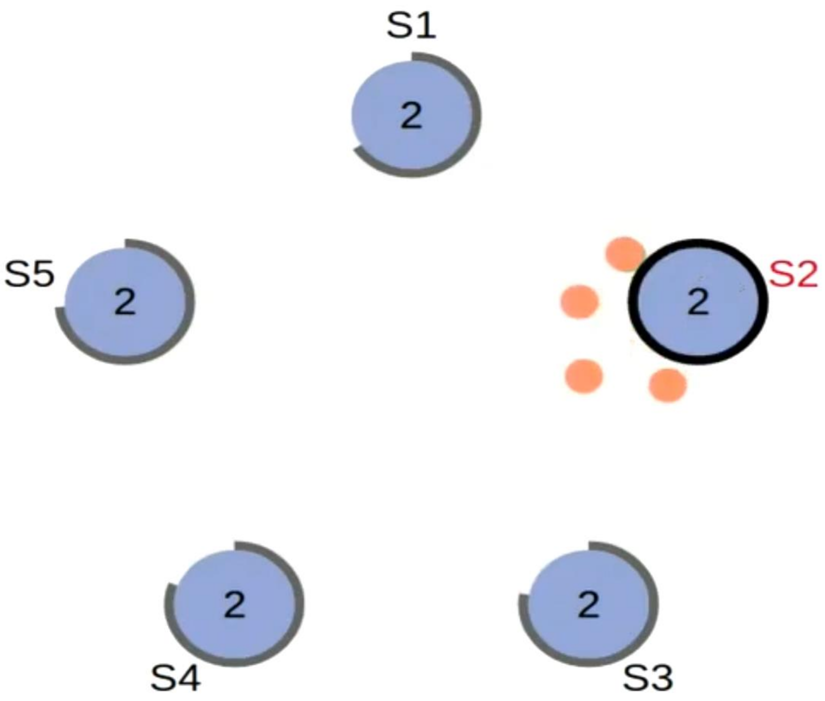
Raft (Reliable | Replicate | Redundant and Fault Tolerant)



Suppose there are 5 nodes S1 to S5 in term 1 with random election time out.

Consensus Algorithm

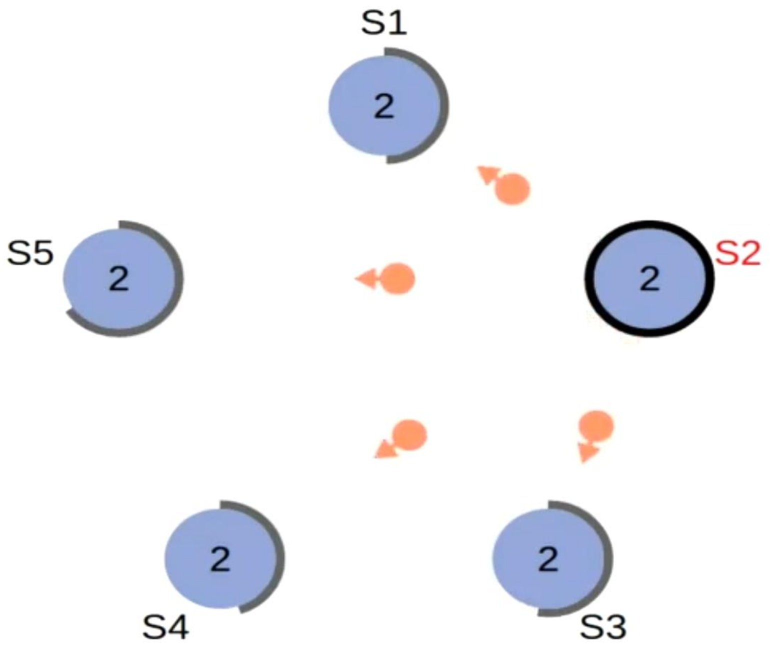
Raft (Reliable | Replicate | Redundant and Fault Tolerant)



S2 timeout and start sending vote request.

Consensus Algorithm

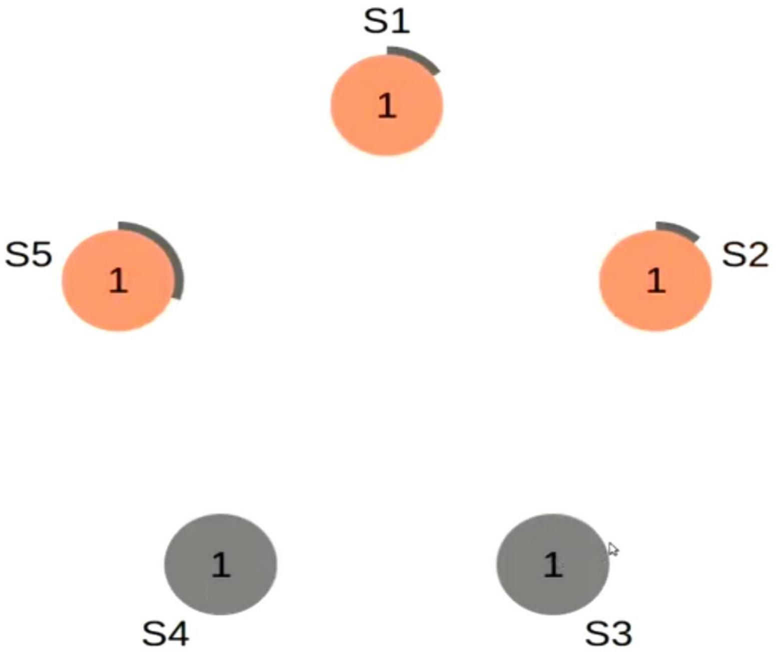
Raft (Reliable | Replicate | Redundant and Fault Tolerant)



S2 received majority vote and becomes leader, then start sending heartbeat message to all nodes.

Consensus Algorithm

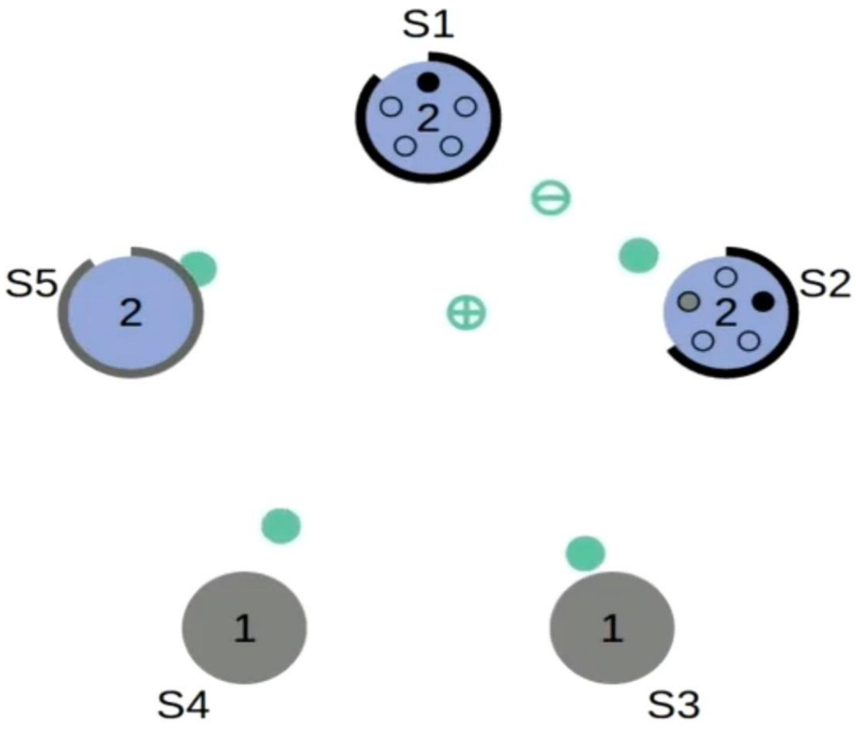
Raft (Reliable | Replicate | Redundant and Fault Tolerant)



In case S3 and S4 are unresponsive, what will happened?

Consensus Algorithm

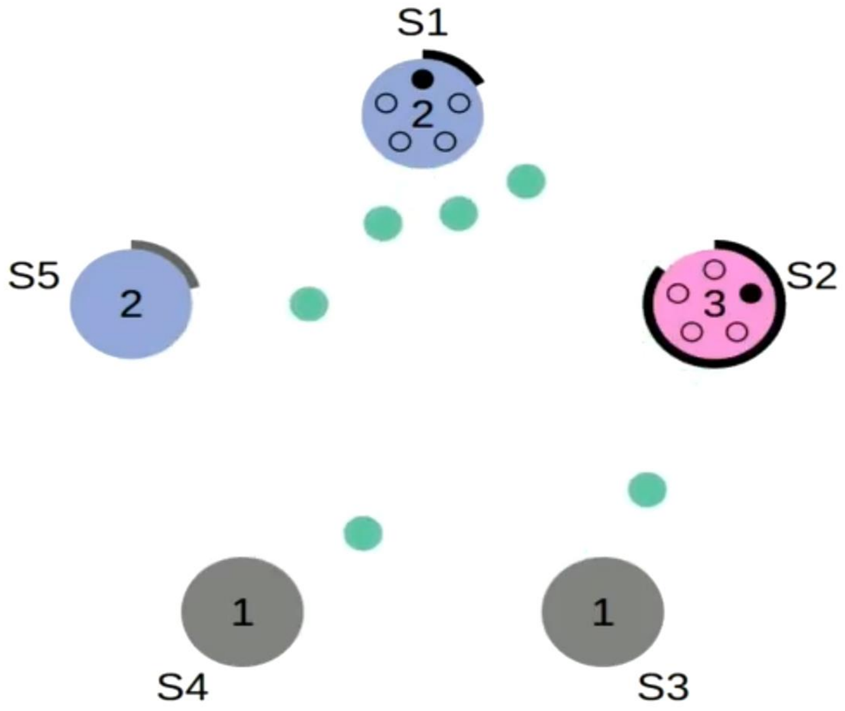
Raft (Reliable | Replicate | Redundant and Fault Tolerant)



S1 and S2 got only one vote from S5 and itself, so both of them can't become leader. So term 2 election is failed.

Consensus Algorithm

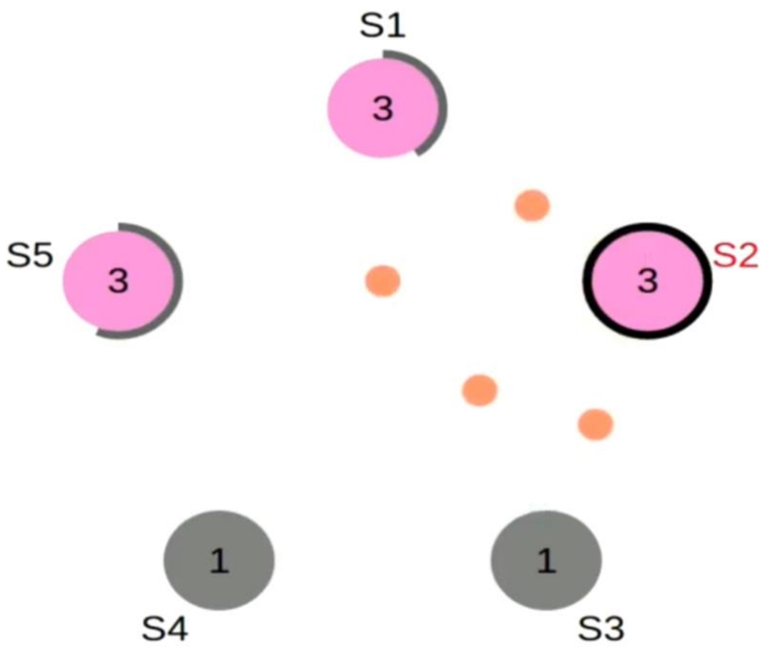
Raft (Reliable | Replicate | Redundant and Fault Tolerant)



When term 2 election timeout, S2 start term 3 election.

Consensus Algorithm

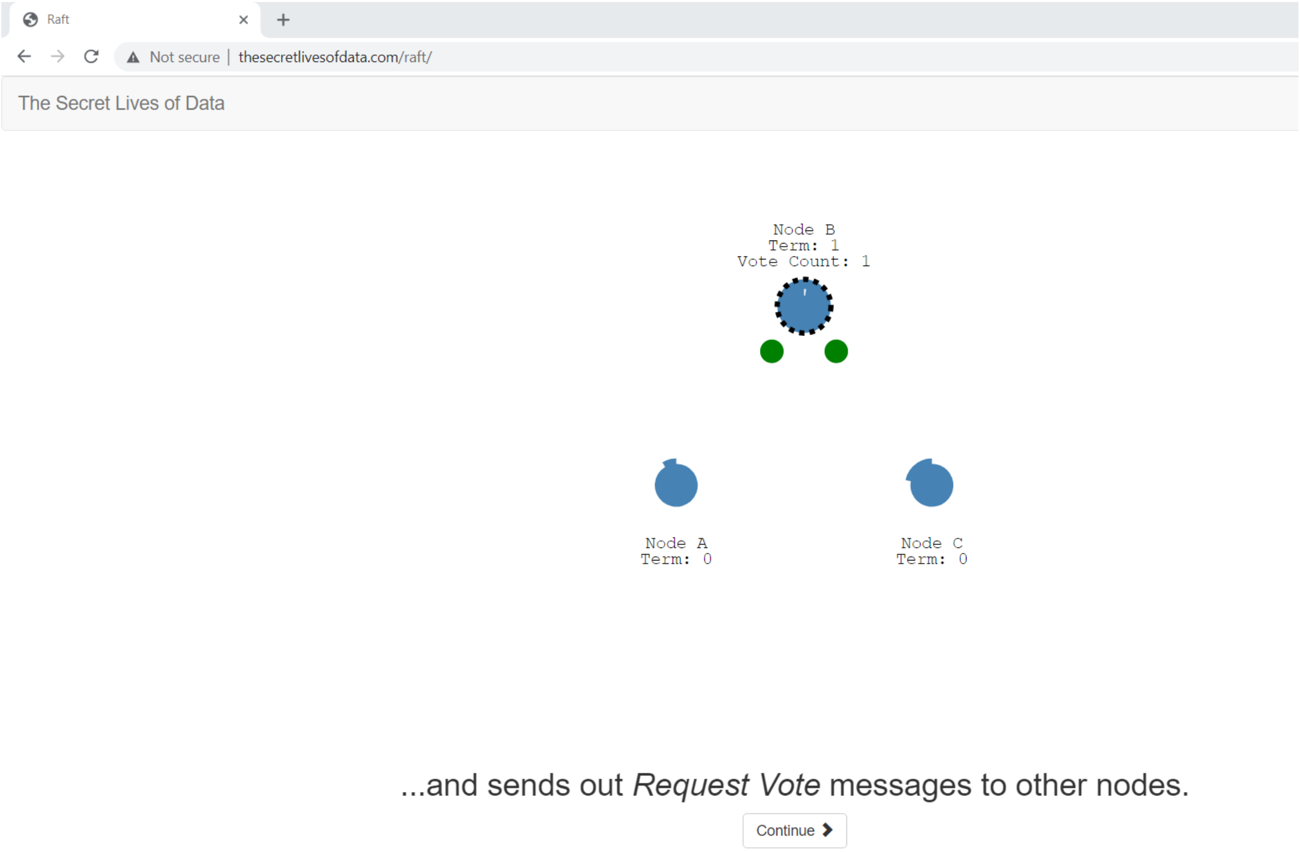
Raft (Reliable | Replicate | Redundant and Fault Tolerant)



When S2 received majority vote, it becomes leader of term 3 and start sending heartbeat message to all nodes.

Consensus Algorithm

Raft with animation

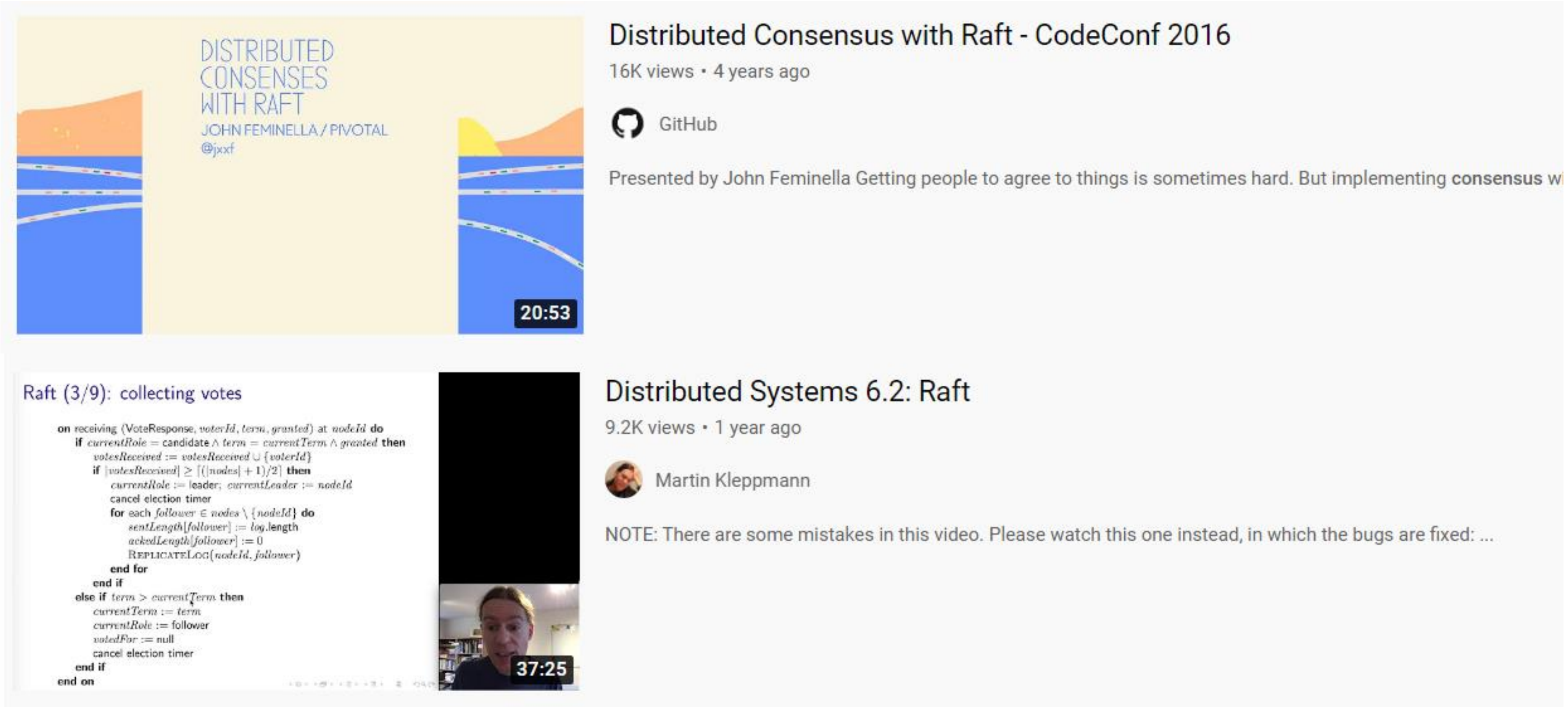


...and sends out *Request Vote* messages to other nodes.

http://thesecretlivesofdata.com/raft/

Consensus Algorithm

Reference Videos



Distributed Consensus with Raft - CodeConf 2016

16K views • 4 years ago

GitHub

Presented by John Feminella Getting people to agree to things is sometimes hard. But implementing **consensus** wi

Raft (3/9): collecting votes

```
on receiving (VoteResponse, voterId, term, granted) at nodeId do
    if currentRole = candidate ∧ term = currentTerm ∧ granted then
        votesReceived := votesReceived ∪ {voterId}
        if |votesReceived| ≥ ⌈(|nodes| + 1)/2⌉ then
            currentRole := leader; currentLeader := nodeId
            cancel election timer
            for each follower ∈ nodes \ {nodeId} do
                sentLength[follower] := log.length
                ackedLength[follower] := 0
                ReplicateLog(nodeId, follower)
            end for
        end if
    else if term > currentTerm then
        currentTerm := term
        currentRole := follower
        votedFor := null
        cancel election timer
    end if
end on
```

Distributed Systems 6.2: Raft

9.2K views • 1 year ago

Martin Kleppmann

NOTE: There are some mistakes in this video. Please watch this one instead, in which the bugs are fixed: ...