Raft

Yicheng Jin

Some materials are from Prof. Danyang Zhuo

Fault-Tolerant Distributed System

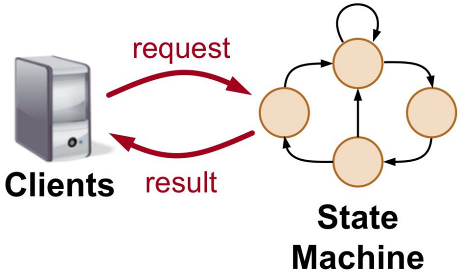
Fundamental Goals

- Safety
    - Nothing bad happens. System runs correctly
    - **Consistency**: All nodes see the same data at the same time
- Liveness
    - Something good eventually happens. System makes progress
    - **Availability**: Systems remain responsive during failures

Fault-Tolerant Distributed System

Generic Technique: Replicated State Machine (RSM)

- Replicated state machine
  - Model service as a *deterministic state machine*
    - Same commands → same output
  - Run multiple replicas that fail *independently*
    - Redundancy and independency
    - No single point of failure
  - Replicas *agree on* the seq of ops they run



Model Services as State Machines

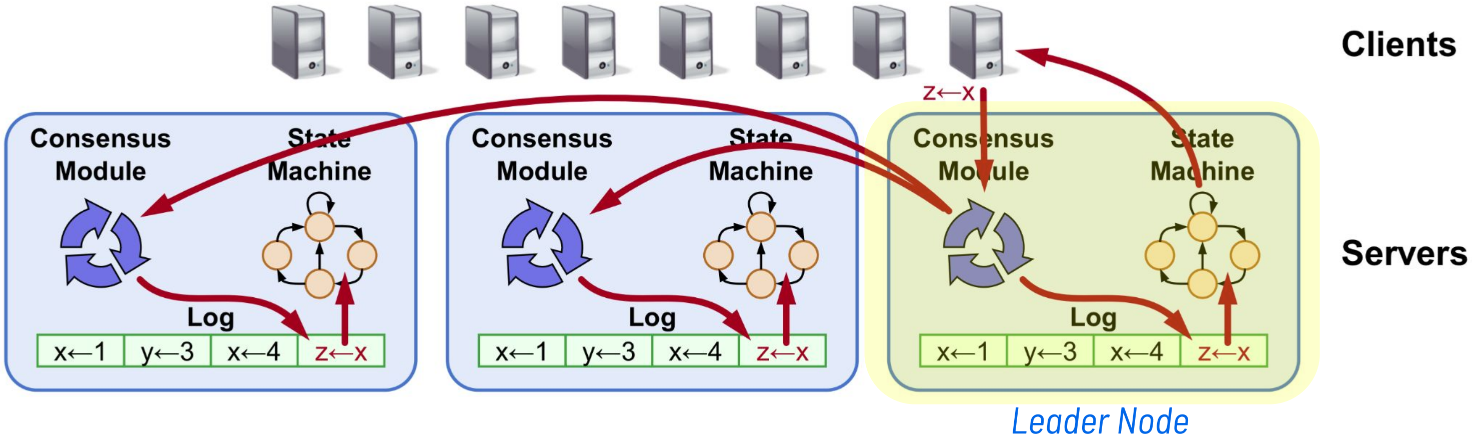
Fault-Tolerant Distributed System

Consensus Algorithm

- Consensus algorithm
  - Nodes agree on something in the presence of failures
  - Type 1. Byzantine fault tolerance
    - Proof-of-work (BitCoin), proof-of-stake (Ethereum)
  - Type 2. Crash fault tolerance
    - Paxos, Raft (this lecture)

Fault-Tolerant Distributed System

RSM with Consensus

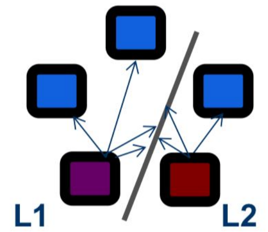


- **Leader-based**: A leader node is designated to coordinate consensus process
- **Consensus**: Ensure consistency across nodes via log replication

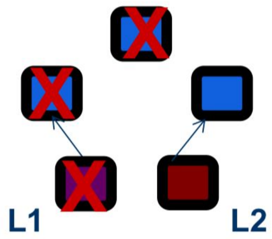
Fault-Tolerant Distributed System

Fundamental Trade-Off Between Safety and Liveness

- FLP Impossibility
  - **Theorem**: In an *asynchronous* (unbounded network delay) distributed system with even one *faulty* node, it is impossible to guarantee both safety and liveness in reaching consensus.
  - **Proof**: Because messages can be delayed indefinitely, a protocol cannot distinguish between a slow node (or network partition) and a failed node.
  - **Implication**: Consensus algorithms must sacrifice liveness under certain failures to maintain safety.



Case 1. Network Partition

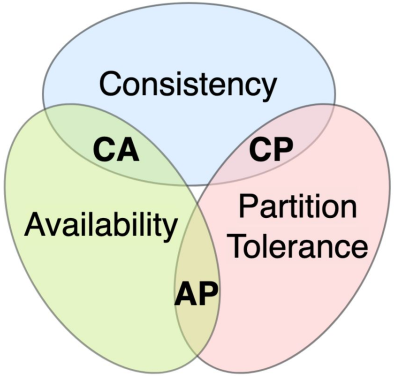


Case 2. Node Failure

Fault-Tolerant Distributed System

Fundamental Trade-Off Between Safety and Liveness

- CAP Theorem
  - **Theorem**: In a distributed system that can experience *network partitions*, it is impossible to simultaneously achieve consistency, availability, and partition tolerance.
  - **Proof**: Serving clients in one partition during network partition: (1) reject ops: sacrifice availability; (2) allow ops: sacrifice consistency.
  - **Implication**: Under network failures, you must choose between consistency (safety) and availability (liveness).



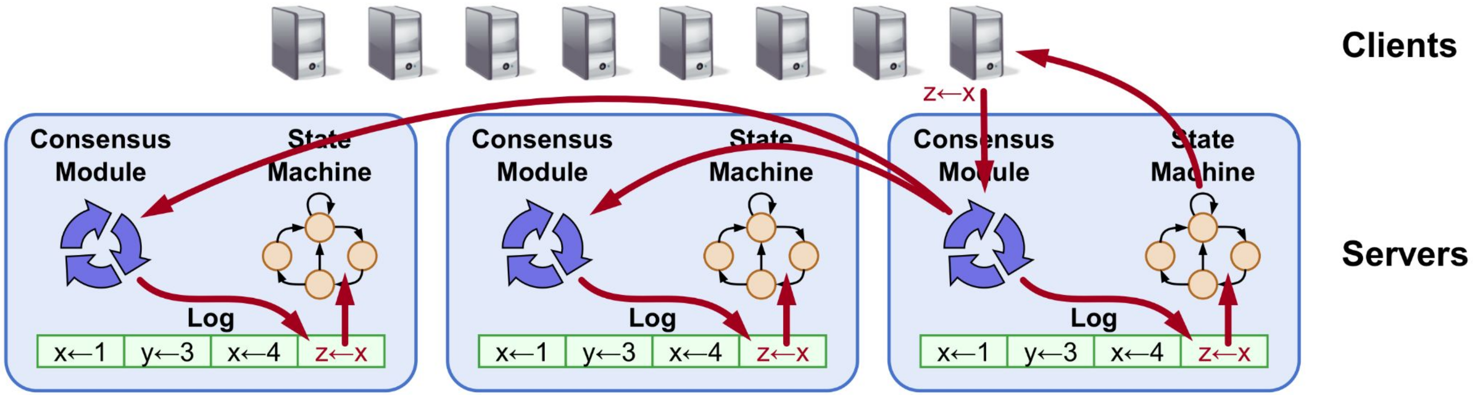
Categorizing Dist Sys By Trade-Offs They Make

Fault-Tolerant Distributed System

Quorum-Based Consensus: Trade Off Availability for Consistency

- Leader must be supported by a majority of the group (a *quorum*)
  - **Consistency**: At most one connected subgroup can serve requests
  - **Availability**: Once a majority of replicas fail, the remaining replicas should not serve requests due to the indistinguishbility issue.

Raft: Overview



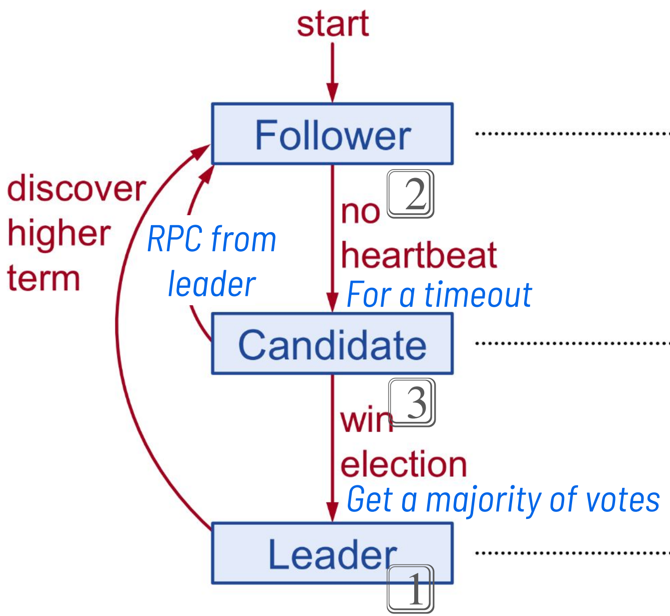
- Quorum-based, leader-based consensus for RSM
- Failure model: message delayed/lost messages, fail-stop

Raft: Problem Decomposition

- Leader election
  - Detect leader crashes and select a new leader
- Log replication
  - Leader appends commands from clients to its log
  - Leader replicate its log to other servers
- Safety properties

Raft: Leader Election

Server Roles and RPCs



Passive (but expects regular heartbeats)

Issues **RequestVote** RPCs to get elected as leader

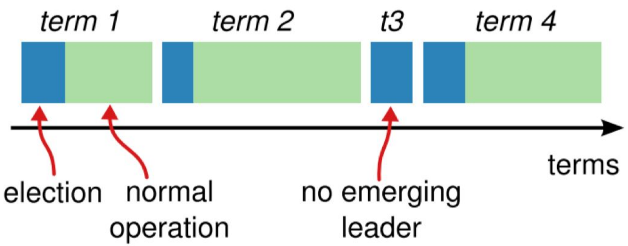
Nodes vote for the first request received

Issues **AppendEntries** RPCs:

- Replicate its log
- Heartbeats to maintain leadership

Raft: Leader Election

Terms

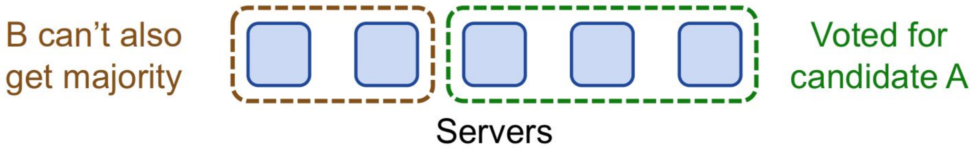


- Logical time individually maintained by each node
  - Incremented when a node initiates a new election
  - Synced when receives RPCs with newer terms
- Included in both types of RPCs to sync clock
  - Older term → reject requests and reply with an error
  - Newer term → advance node's own clock
- Each term has at most one leader
  - Agreement on term is equivalent to agreement on leader
  - Election achieves consensus on leader as well as term

Raft: Leader Election

Safety and Liveness of Election

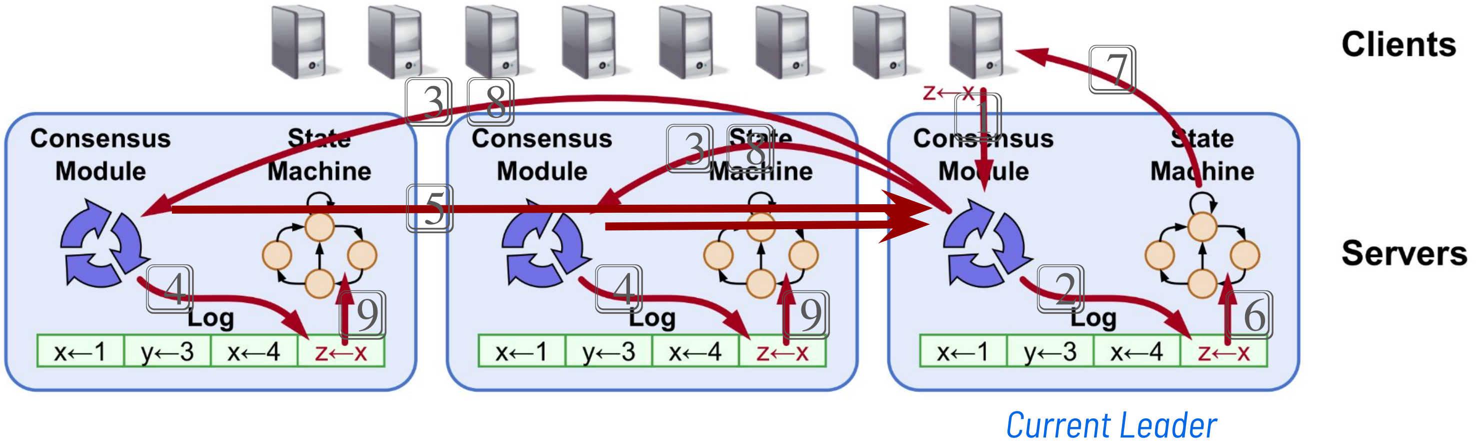
- Safety: at most one leader per term
    - **Proof**: one vote per node. Two majorities must overlap.



- Liveness: some candidate eventually wins
    - **Issue**: fixed election timeout leads to *split votes*, where multiple nodes timeout and request votes simultaneously, and none receives a majority
    - **Solution**: nodes use *randomized timeout*, hoping one candidate wins before the next node timeout

Raft: Log Replication

Normal Operation



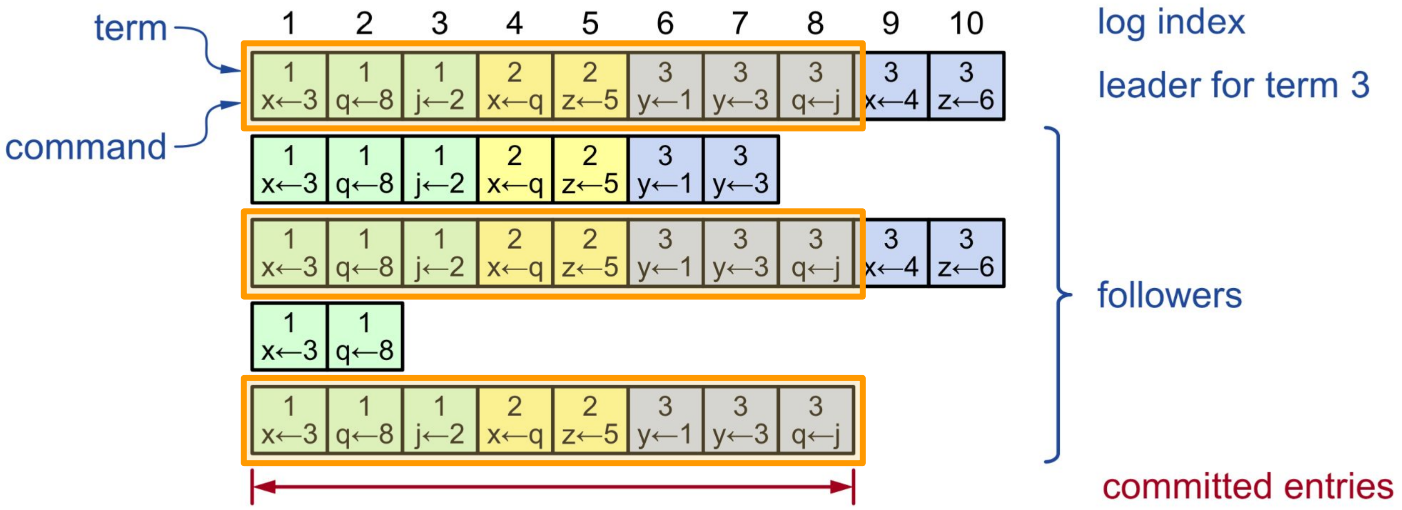
5 Followers reply with a positive acknowledgement

8 Leader notifies followers of *committed entries* in subsequent `AppendEntries` RPCs

Replicated to any majority

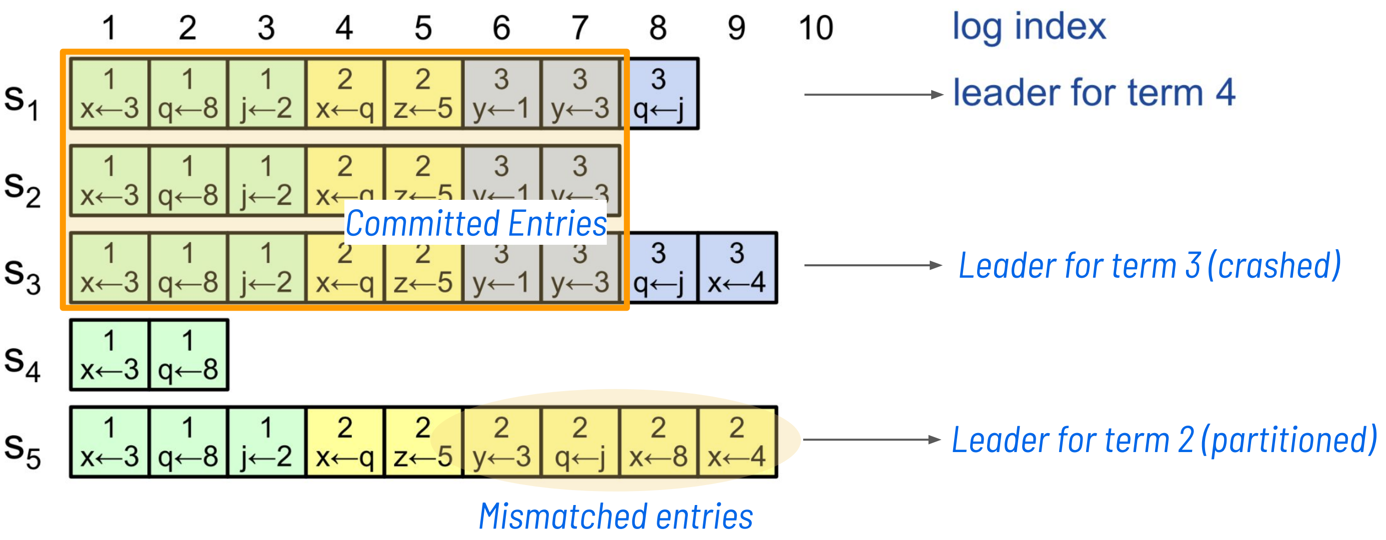
Raft: Log Replication

Log Structure (Normal Operation)



Raft: Log Replication

Issue: Log Inconsistency Caused by Crashes

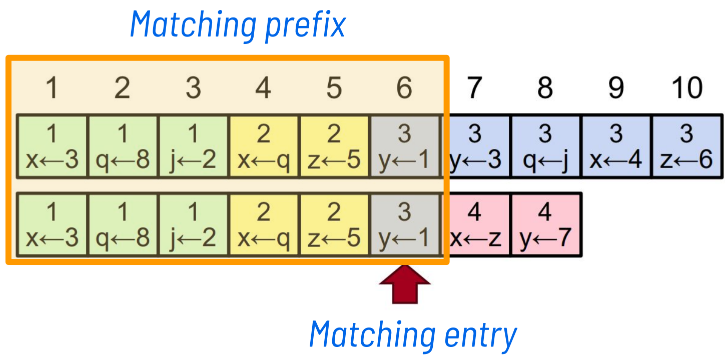


Raft: Log Replication

Fix: Log Matching Property via Consistency Check

- **Log matching property**: If log entries on different nodes have same index and term, then they store the same command, and the logs are identical in all preceding entries.

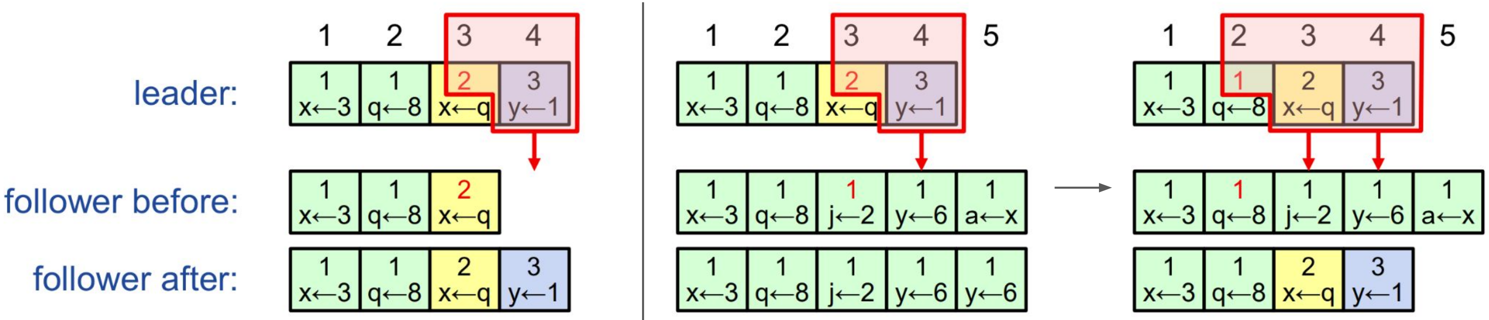
  → If a given entry is committed, all preceding entries are also committed.



Raft: Log Replication

Fix: Log Matching Property via Consistency Check

- `AppendEntries` includes <term, index> of the preceding log entry
- Followers first check if they have the preceding log entry
    - Match: append log entry and send positive ack
    - Mismatch: send negative ack, asking leader to retry replicating the preceding entry
- Ensures log matching property via *induction*



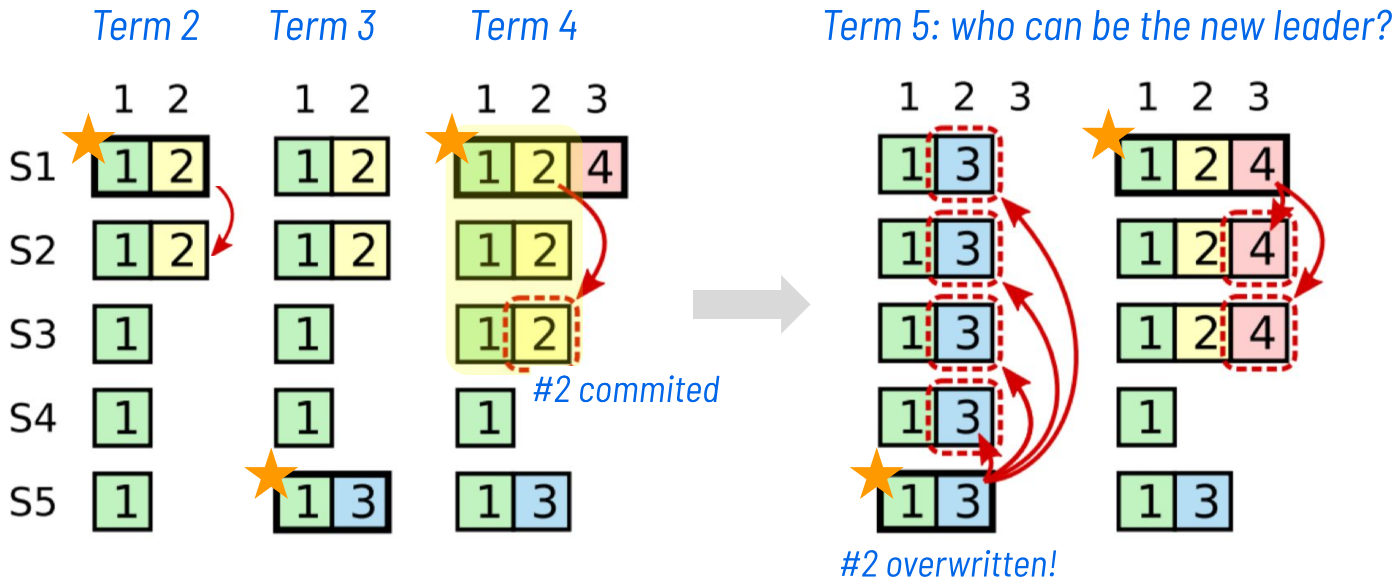
Example #1: success

Example #2: mismatch

Backtrack & retry append #3: success
Retry append #4: success

Raft: Log Replication

Issue: Leader Overwriting Committed Entries of Previous Terms



Raft: Log Replication

Fix: Leader Completeness Property During Election

- Ensure leaders contain all committed logs
- **How**: `RequestVote` includes <term, index> of the last log entry, and voters reject candidates whose log is less up-to-date than them.
- **Proof**: The new leader's log is more up-to-date than a majority of the nodes. The majority that committed the entries in the previous term must *overlap* with the majority that elects the new leader.

Raft: Log Replication

Committed Entries

- Why are log entries replicated to a majority of nodes considered stable and safe to apply to state machines?
    - Election rules make sure that new leaders always contain all committed entries, thus committed entries cannot be overwritten during log replication
    - Leaders can fix inconsistency/lagging in followers’ logs, making sure a majority of nodes contains all committed entries when replicating a new entry