May 2016

Raft in details

Ivan Glushkov
ivan.glushkov@gmail.com
@gliush

Raft is a **consensus algorithm** for managing a **replicated log**

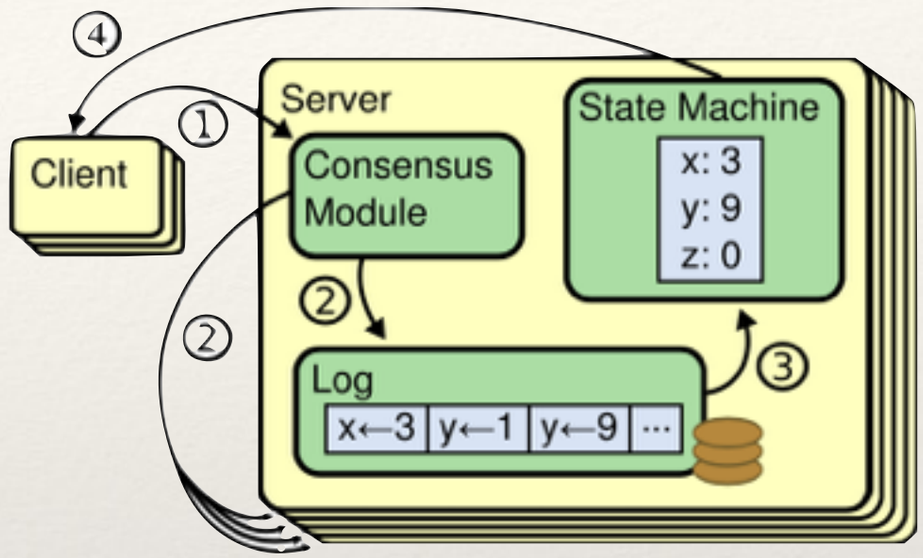
Why

- Very important topic
- [Multi-]Paxos, the key player, is very difficult to understand
- Paxos -> Multi-Paxos -> different approaches
- Different optimization, closed-source implementation

Raft goals

- Main goal - understandability:
- Decomposition: separated leader election, log replication, safety, membership changes
- State space reduction

Replicated state machines



- The same state on each server
- Compute the state from replicated log
- Consensus algorithm: keep the replicated log consistent

Properties of consensus algorithm

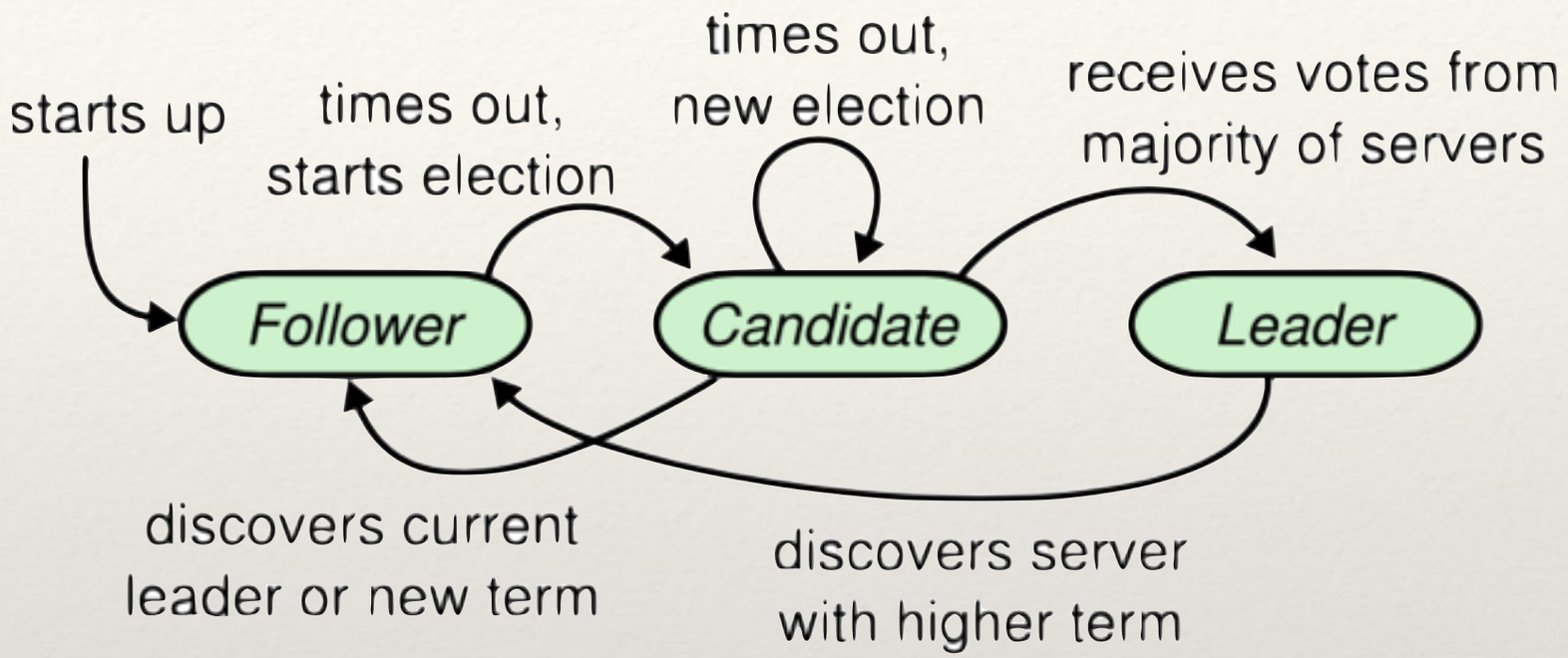
- Never return an incorrect result: Safety
- Functional as long as any majority of the severs are operational: Availability
- Do not depend on timing to ensure consistency (at worst - unavailable)

Raft in a nutshell

- Elect a leader
- Give full responsibility for replicated log to leader:
  - Leader accepts log entries from client
  - Leader replicates log entries on other servers
  - Tell servers when it is safe to apply changes on their state

Raft basics: States

- Follower
- Candidate
- Leader

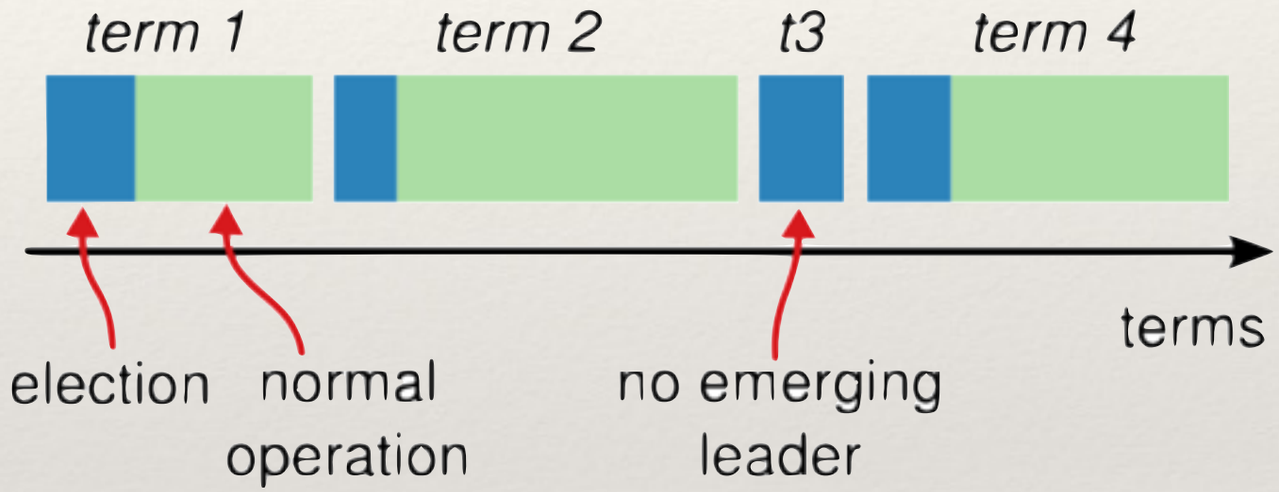


RequestVote RPC

- Become a candidate, if no communication from leader over a timeout (random *election timeout*: e.g. 150-300ms)
- Spec: (term, candidateId, lastLogIndex, lastLogTerm)
  -> (term, voteGranted)
- Receiver:
  - false if term < currentTerm
  - true if not voted yet and term and log are Up-To-Date
  - false

Leader Election

- Heartbeats from leader
- Election timeout
- Candidate results:
  - It wins
  - Another server wins
  - No winner for a period of time -> new election

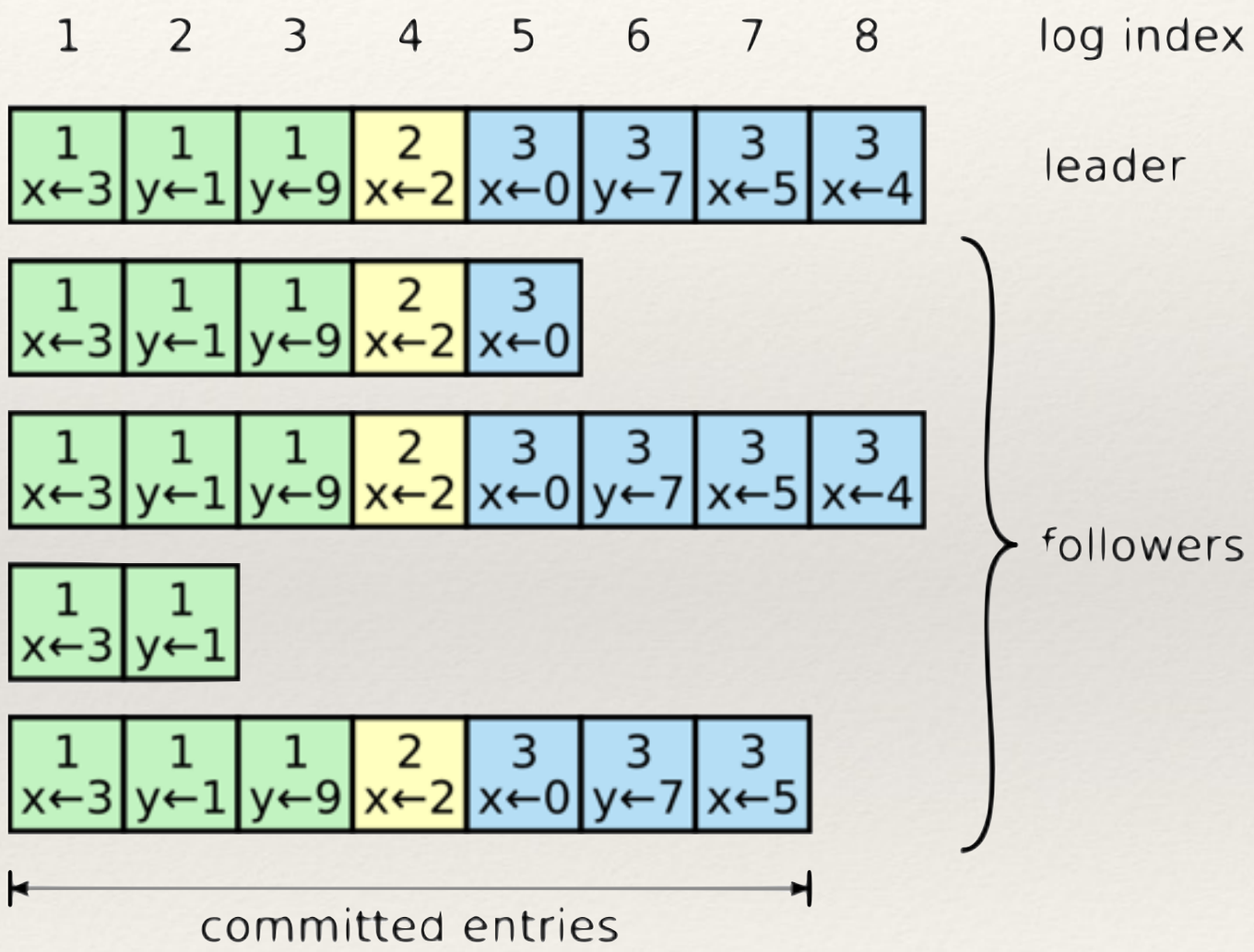


Leader Election Property

- **Election Safety**: at most one leader can be elected in a given term

Log Replication

- log index
- log term
- log command



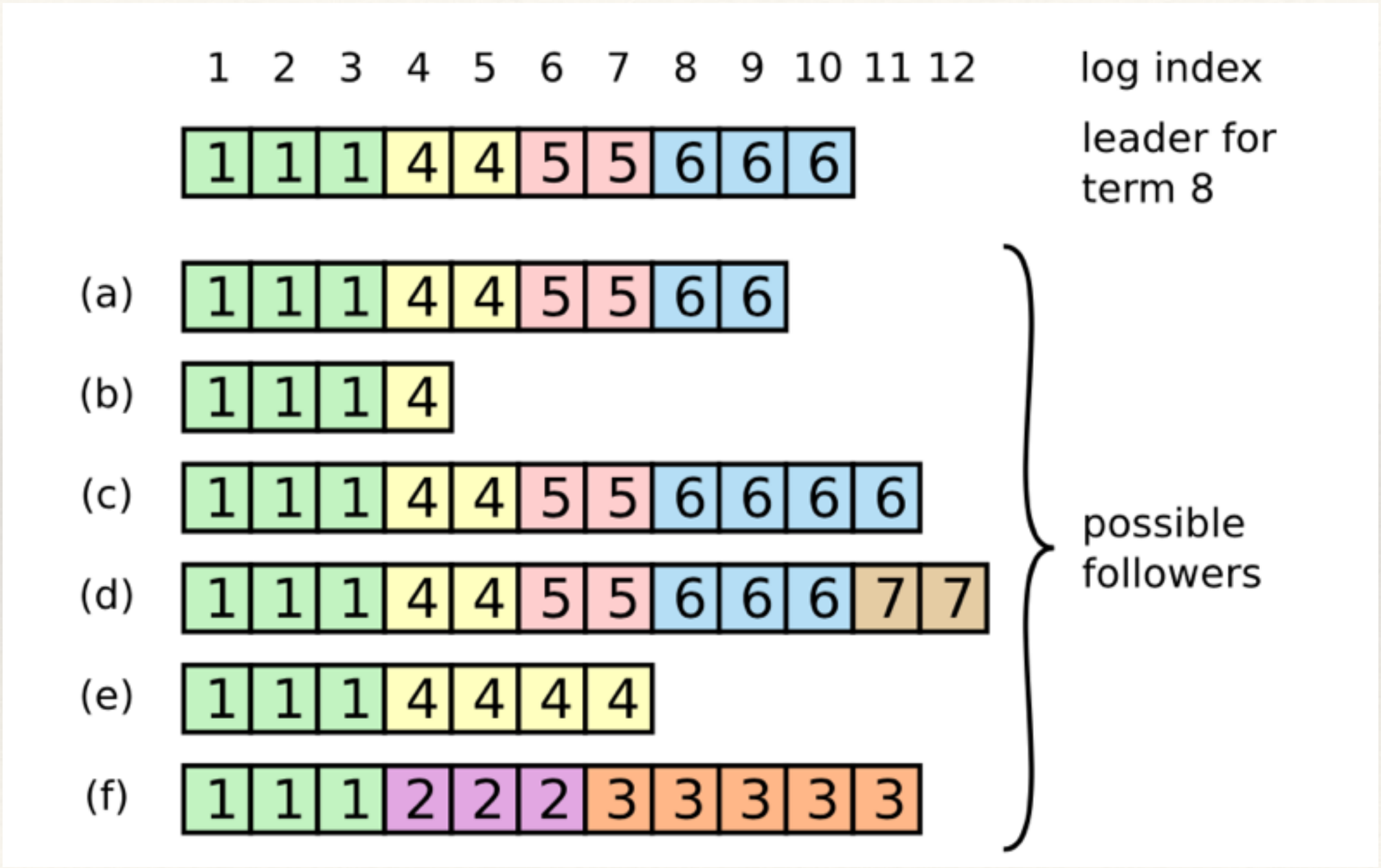
Log Replication

- Leader **appends** command from client to its log
- Leader issues AppendEntries RPC to replicate the entry
- Leader replicates entry to majority -> apply entry to state -> “committed entry”
- Follower checks if entry is committed by server -> commit it locally

Log Replication Properties

- **Leader Append-Only**: a leader never overwrites or deletes entries in its log; it only appends new entries
- **Log Matching:** If two entries in different logs have the same index and term, then they store the same command.
- **Log Matching:** If two entries in different logs have the same index and term, then the logs are identical in all preceding entries.

Inconsistency



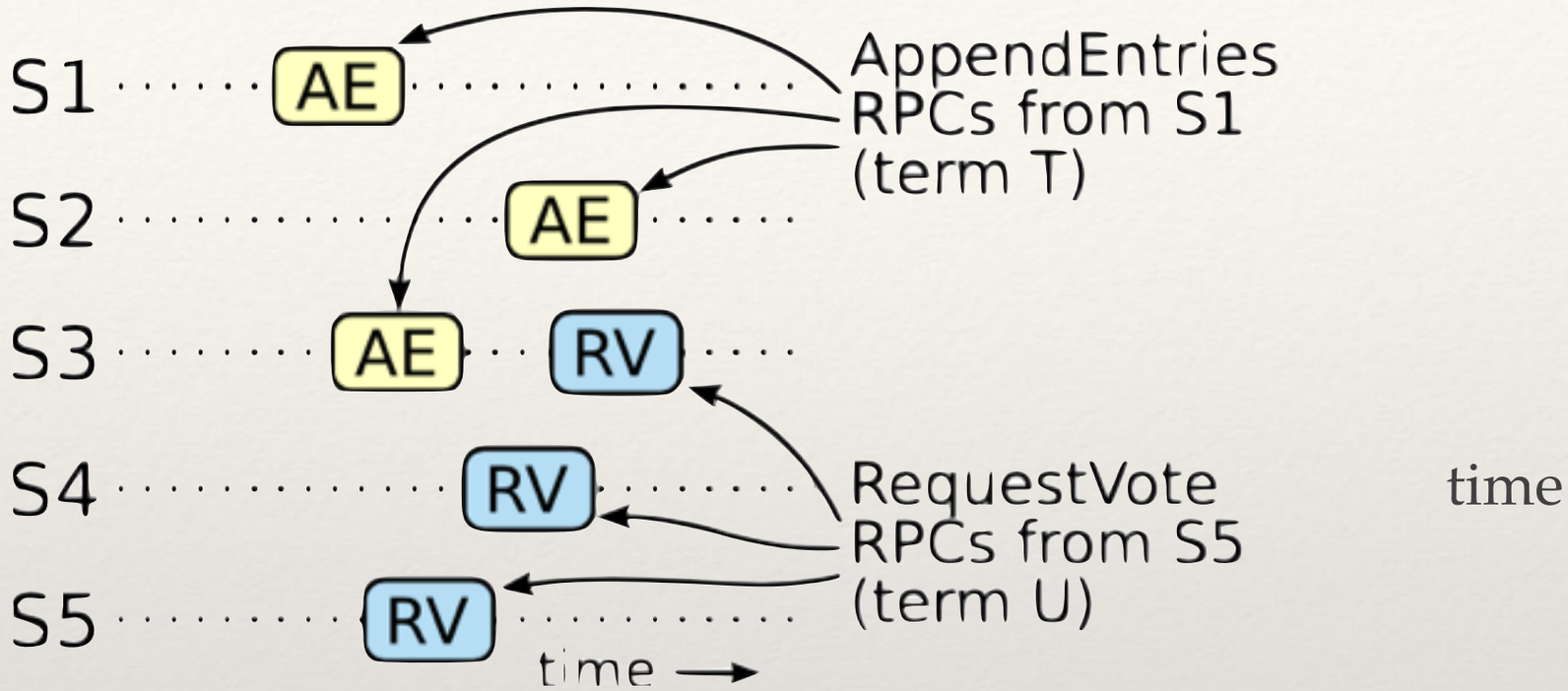
Solving Inconsistency

- Leader forces the followers' logs to duplicate its own
- Conflicting entries in the follower logs will be overwritten with entries from the Leader's log:
  - Find latest log where two servers agree
  - Remove everything after this point
  - Write new logs

Safety Property

- **Leader Completeness**: if a log entry is committed in a given term, then that entry will be present in the logs of the leaders for all higher-numbered terms
- **State Machine Safety**: if a server has applied a log entry at a given index to its state machine, no other server will ever apply a different log entry for the same index.

Safety Restriction



- **Up-to-date**: if the logs have last entries with different terms, then the log with the later term is more up-to-date. If the logs end with the same term, then whichever log is longer is more up-to-date.

Follower and Candidate Crashes

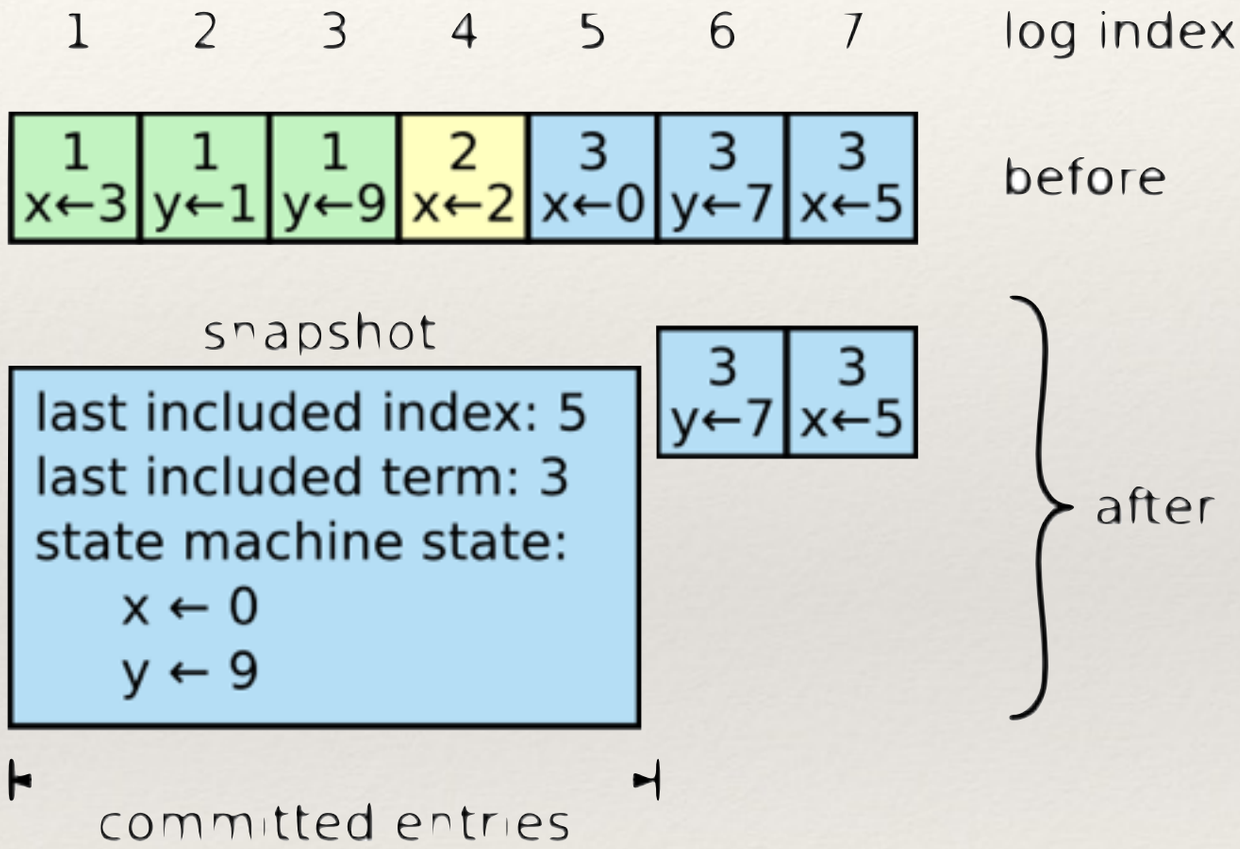
- Retry indefinitely
- Raft RPC are idempotent

Raft Joint Consensus

- Two phase to change membership configuration
- In Joint Consensus state:
  - replicate log entries to both configuration
  - any server from either configuration may be a leader
  - agreement (for election and commitment) requires separate majorities from both configuration

Log compaction

- Independent snapshotting
- Snapshot metadata
- InstallSnapshot RPC



Client Interaction

- Works with leader
- Leader respond to a request when it commits an entry
- Assign uniqueID to every command, leader stores latest ID with response.
- Read-only - could be without writing to a log, possible stale data. Or write "*no-op*" entry at start

Correctness

- TLA+ formal specification for consensus alg (400 LOC)
- Proof of linearizability in Raft using Coq (community)

Questions?