

Implementing Replicated Logs with Paxos

John Ousterhout and Diego Ongaro
Stanford University

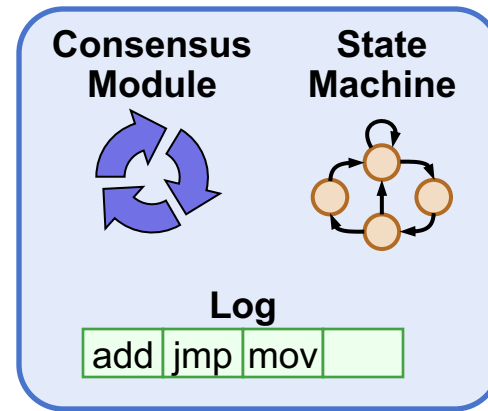
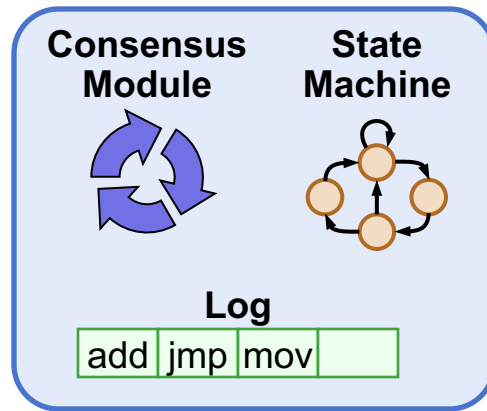
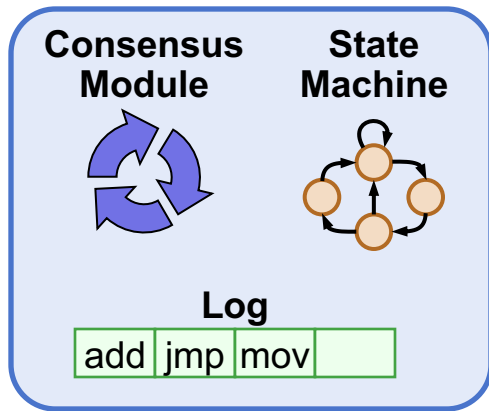


Note: this material borrows heavily from slides by Lorenzo Alvisi, Ali Ghodsi, and David Mazières

Goal: Replicated Log



Clients



Servers

- **Replicated log => replicated state machine**
 - All servers execute same commands in same order
- **Consensus module ensures proper log replication**
- **System makes progress as long as any majority of servers are up**
- **Failure model: fail-stop (not Byzantine), delayed/lost messages**

The Paxos Approach

Decompose the problem:

- **Basic Paxos (“single decree”):**
 - One or more servers propose values
 - System must agree on a **single value** as **chosen**
 - Only one value is ever chosen
- **Multi-Paxos:**
 - Combine several instances of Basic Paxos to agree on a series of values forming the log

Requirements for Basic Paxos

- **Safety:**
 - Only a single value may be chosen
 - A server never learns that a value has been chosen unless it really has been
- **Liveness (as long as majority of servers up and communicating with reasonable timeliness):**
 - Some proposed value is eventually chosen
 - If a value is chosen, servers eventually learn about it

The term “consensus problem” typically refers to this single-value formulation

Paxos Components

- **Proposers:**

- Active: put forth particular values to be chosen
- Handle client requests

- **Acceptors:**

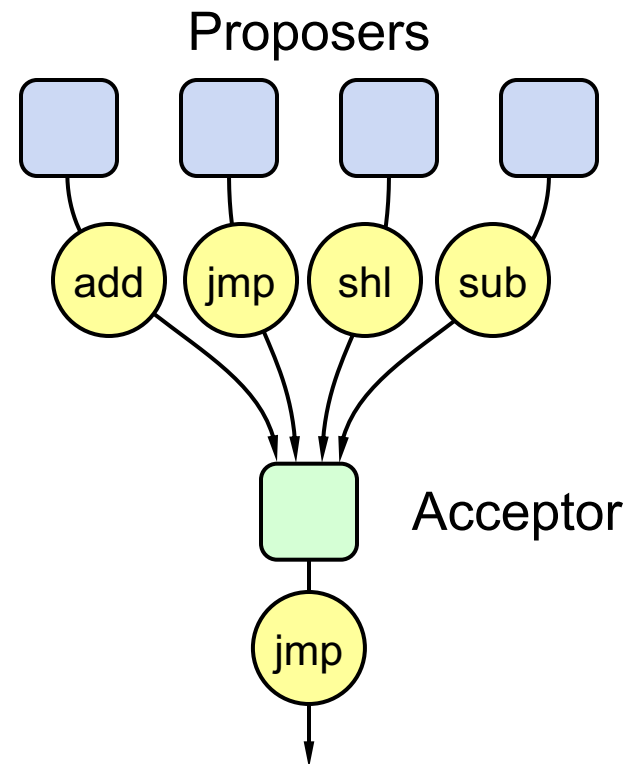
- Passive: respond to messages from proposers
- Responses represent votes that form consensus
- Store chosen value, state of the decision process
- Want to know which value was chosen

For this presentation:

- Each Paxos server contains both components

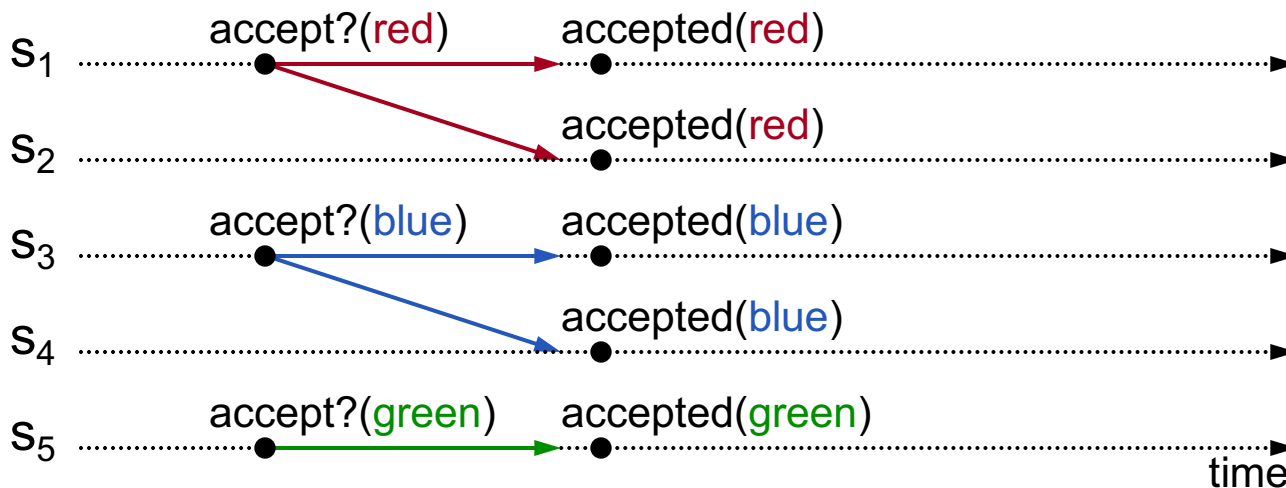
Strawman: Single Acceptor

- **Simple (incorrect) approach:** a single acceptor chooses value
- **What if acceptor crashes after choosing?**
- **Solution: quorum**
 - Multiple acceptors (3, 5, ...)
 - Value v is **chosen** if accepted by **majority** of acceptors
 - If one acceptor crashes, chosen value still available



Problem: Split Votes

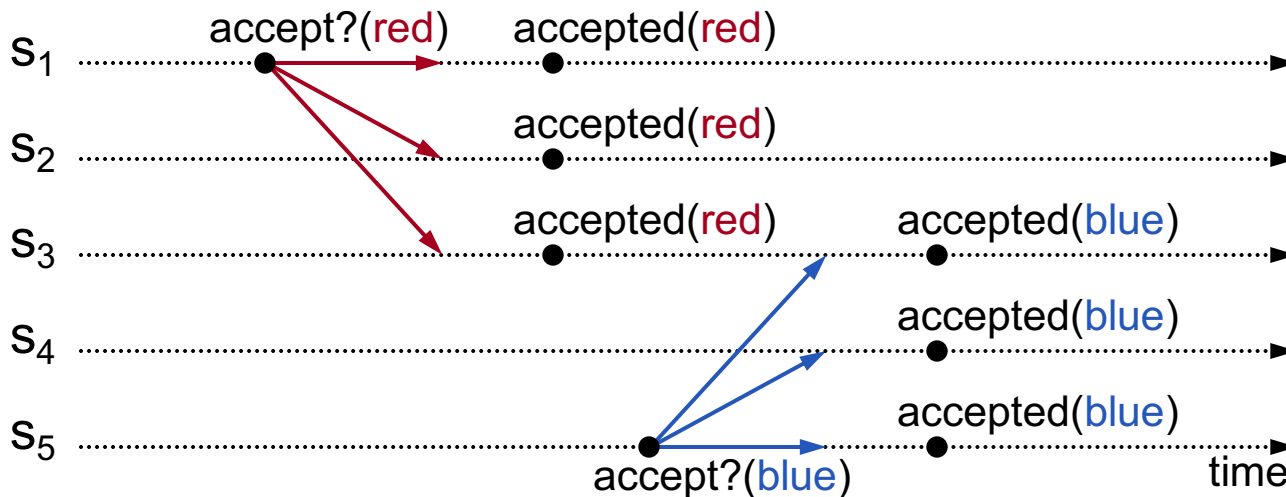
- **Acceptor accepts only first value it receives?**
- **If simultaneous proposals, no value might be chosen**



Acceptors must sometimes accept multiple (different) values

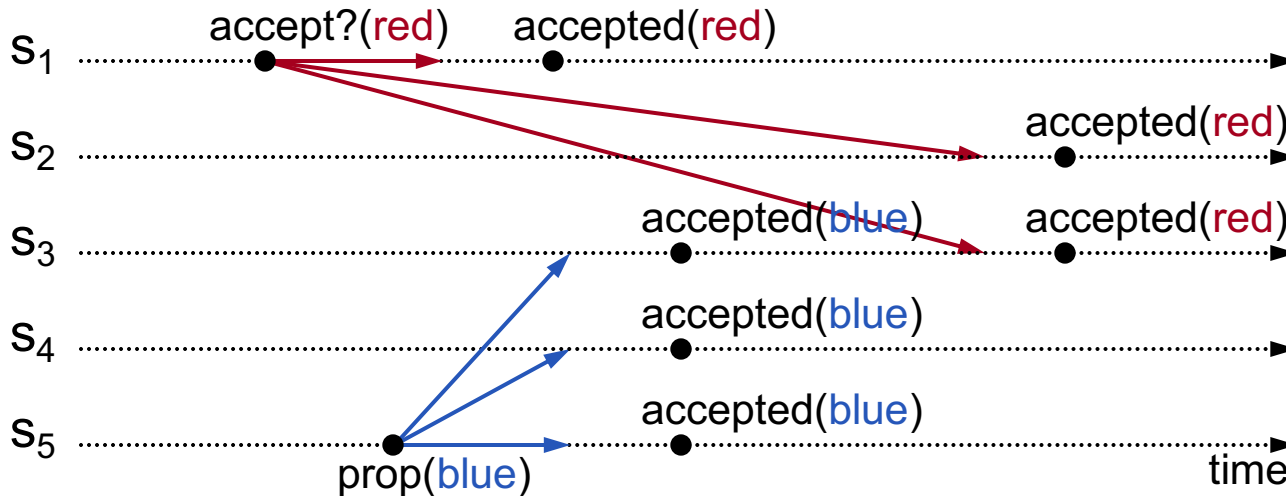
Problem: Conflicting Choices

- Acceptor accepts **every** value it receives?
- Could choose multiple values



Once a value has been chosen, future proposals must propose/choose that same value (**2-phase protocol**)

Conflicting Choices, cont'd



- s₅ needn't propose **red** (it hasn't been chosen yet)
- s₁'s proposal must be aborted (s₃ must reject it)

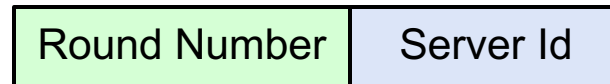
Must order proposals, reject old ones

Proposal Numbers

- **Each proposal has a unique number**
 - Higher numbers take priority over lower numbers
 - It must be possible for a proposer to choose a new proposal number higher than anything it has seen/used before

- **One simple approach:**

Proposal Number



- Each server stores maxRound: the largest Round Number it has seen so far
- To generate a new proposal number:
 - Increment maxRound
 - Concatenate with Server Id
- Proposers must persist maxRound on disk: must not reuse proposal numbers after crash/restart

Basic Paxos

Two-phase approach:

- **Phase 1: broadcast **Prepare** RPCs**
 - Find out about any chosen values
 - Block older proposals that have not yet completed
- **Phase 2: broadcast **Accept** RPCs**
 - Ask acceptors to accept a specific value

Basic Paxos

Proposers

- 1) Choose new proposal number n
- 2) Broadcast `Prepare(n)` to all servers
- 4) When responses received from majority:
 - If any `acceptedValues` returned, replace value with `acceptedValue` for highest `acceptedProposal`
- 5) Broadcast `Accept(n , value)` to all servers
- 6) When responses received from majority:
 - Any rejections (`result > n`)? goto (1)
 - Otherwise, **value is chosen**

Acceptors

- 3) Respond to `Prepare(n)`:
 - If $n > \text{minProposal}$ then $\text{minProposal} = n$
 - `Return(acceptedProposal, acceptedValue)`
- 6) Respond to `Accept(n , value)`:
 - If $n \geq \text{minProposal}$ then
`acceptedProposal = minProposal = n`
`acceptedValue = value`
 - `Return(minProposal)`

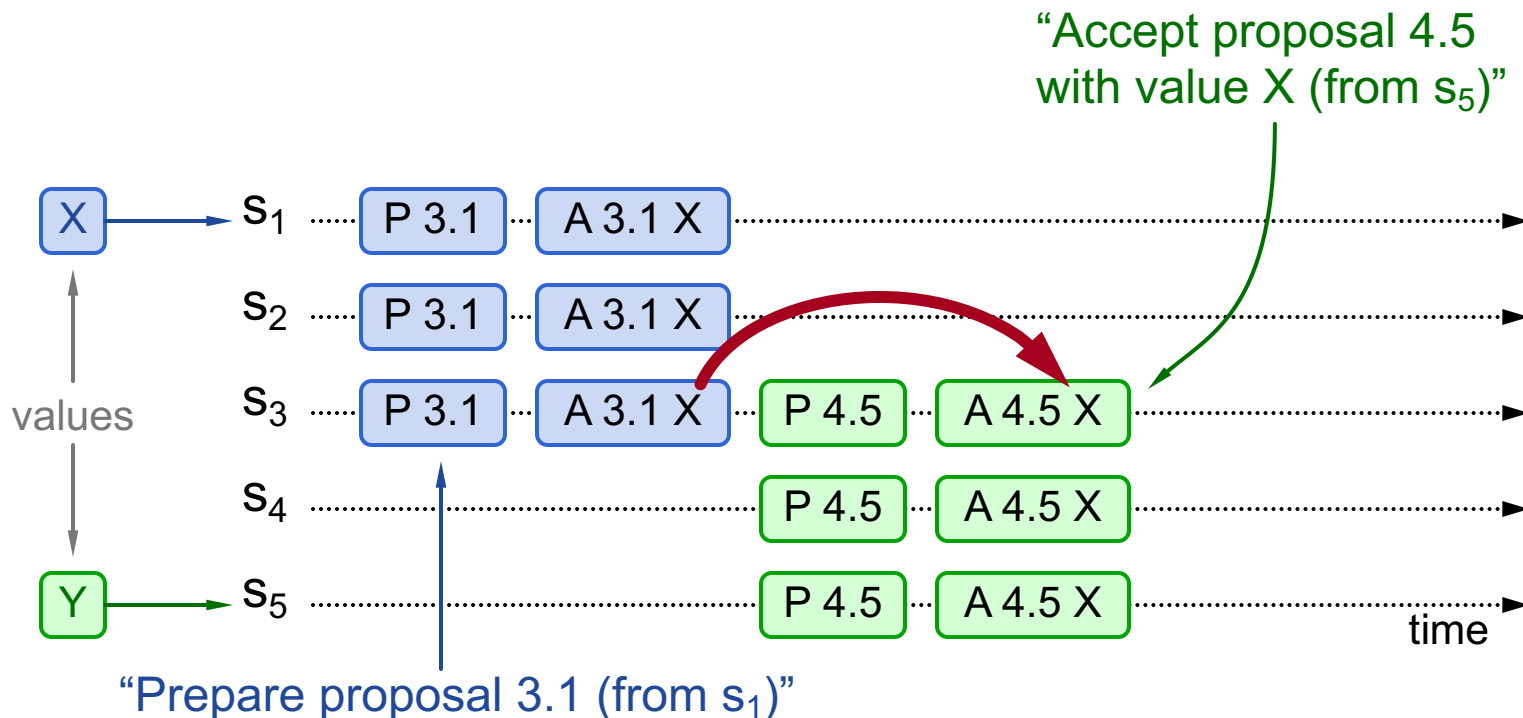
Acceptors must record `minProposal`, `acceptedProposal`, and `acceptedValue` on stable storage (disk)

Basic Paxos Examples

Three possibilities when later proposal prepares:

1. Previous value already chosen:

- New proposer will find it and use it

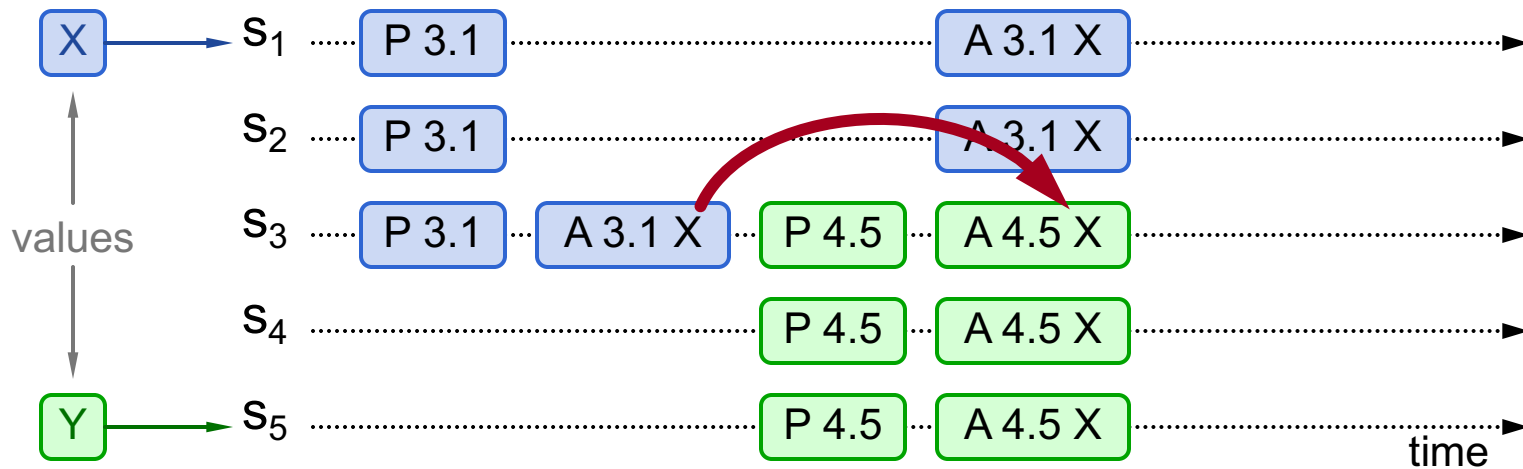


Basic Paxos Examples, cont'd

Three possibilities when later proposal prepares:

2. Previous value not chosen, but new proposer sees it:

- New proposer will use existing value
- Both proposers can succeed

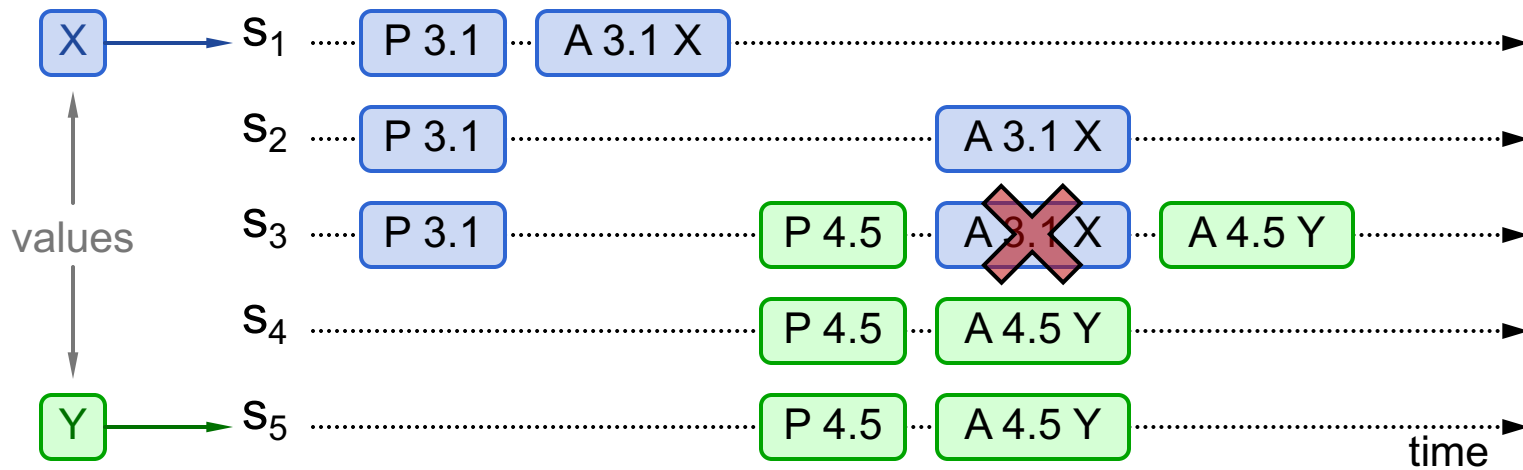


Basic Paxos Examples, cont'd

Three possibilities when later proposal prepares:

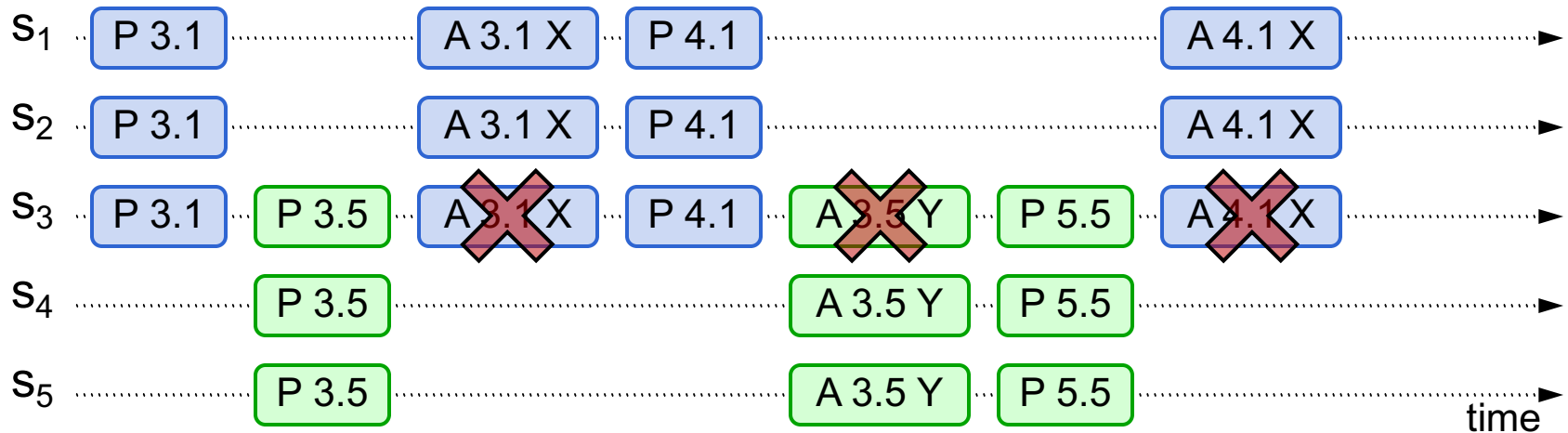
3. Previous value not chosen, new proposer doesn't see it:

- New proposer chooses its own value
- Older proposal blocked



Liveness

- **Competing proposers can livelock:**



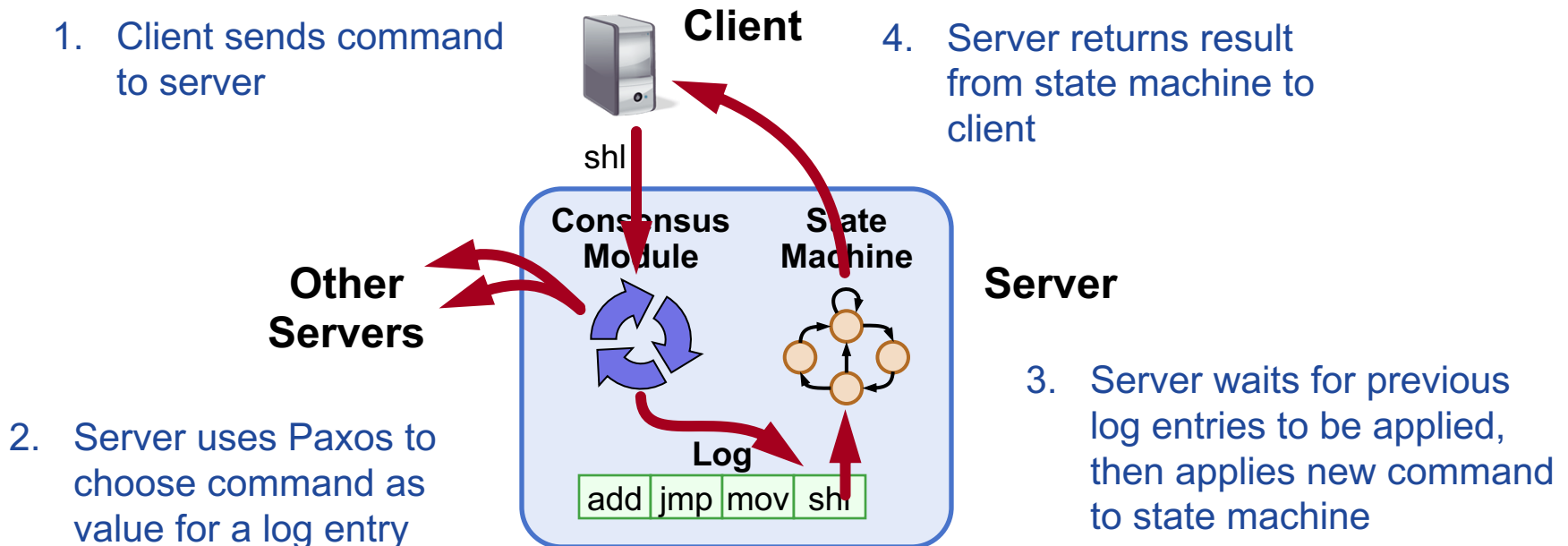
- **One solution: randomized delay before restarting**
 - Give other proposers a chance to finish choosing
- **Multi-Paxos will use leader election instead**

Other Notes

- **Only proposer knows which value has been chosen**
- **If other servers want to know, must execute Paxos with their own proposal**

Multi-Paxos

- **Separate instance of Basic Paxos for each entry in the log:**
 - Add **index** argument to Prepare and Accept (selects entry in log)



Multi-Paxos Issues

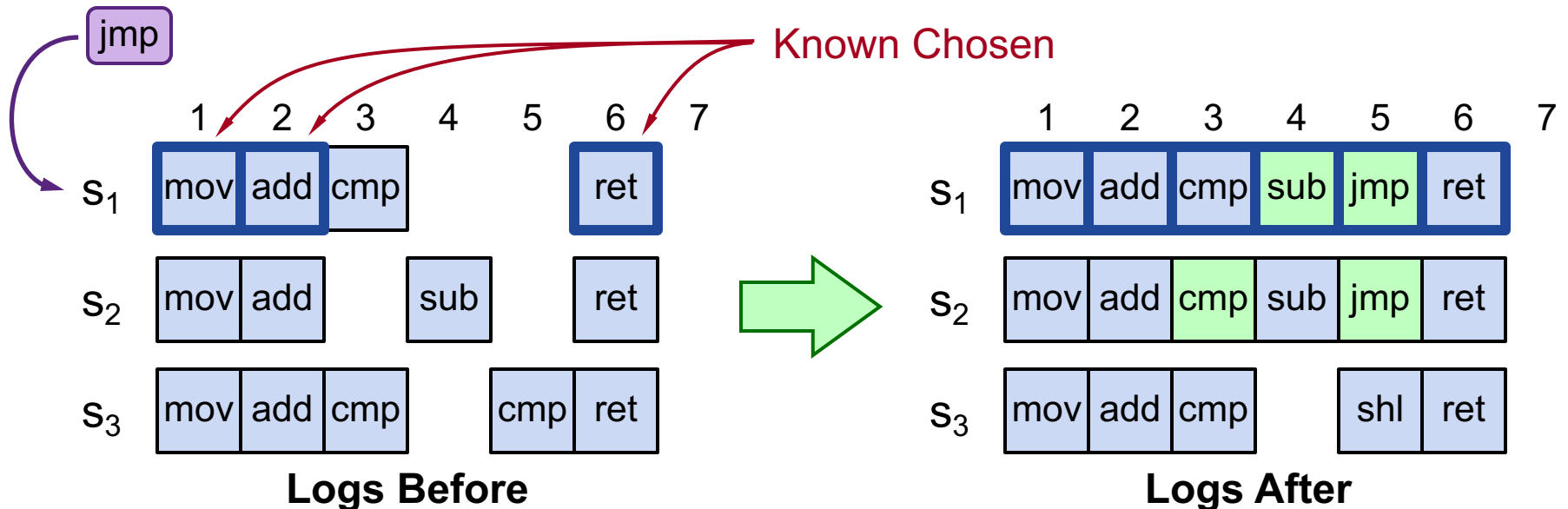
- **Which log entry to use for a given client request?**
- **Performance optimizations:**
 - Use leader to reduce proposer conflicts
 - Eliminate most Prepare requests
- **Ensuring full replication**
- **Client protocol**
- **Configuration changes**

Note: Multi-Paxos not specified precisely in literature

Selecting Log Entries

- **When request arrives from client:**

- Find first log entry not known to be chosen
- Run Basic Paxos to propose client's command for this index
- Prepare returns acceptedValue?
 - Yes: finish choosing acceptedValue, start again
 - No: choose client's command



Selecting Log Entries, cont'd

- **Servers can handle multiple client requests concurrently:**
 - Select different log entries for each
- **Must apply commands to state machine in log order**

Improving Efficiency

- **Using Basic Paxos is inefficient:**

- With multiple concurrent proposers, **conflicts** and restarts are likely (higher load → more conflicts)
- **2 rounds** of RPCs for each value chosen (Prepare, Accept)

Solution:

1. Pick a leader

- At any given time, only one server acts as Proposer

2. Eliminate most Prepare RPCs

- Prepare once for the entire log (not once per entry)
- Most log entries can be chosen in a single round of RPCs

Leader Election

One simple approach from Lamport:

- **Let the server with highest ID act as leader**
- **Each server sends a heartbeat message to every other server every T ms**
- **If a server hasn't received heartbeat from server with higher ID in last $2T$ ms, it acts as leader:**
 - Accepts requests from clients
 - Acts as proposer and acceptor
- **If server not leader:**
 - Rejects client requests (redirect to leader)
 - Acts only as acceptor

Eliminating Prepares

- **Why is Prepare needed?**
 - Block old proposals
 - Make proposal numbers refer to the **entire log**, not just one entry
 - Find out about (possibly) chosen values
 - Return highest proposal accepted for current entry
 - Also return **noMoreAccepted**: no proposals accepted for any log entry beyond current one
- **If acceptor responds to Prepare with noMoreAccepted, skip future Prepares with that acceptor (until Accept rejected)**
- **Once leader receives noMoreAccepted from majority of acceptors, no need for Prepare RPCs**
 - **Only 1 round of RPCs needed per log entry (Accepts)**

Full Disclosure

- **So far, information flow is incomplete:**
 - Log entries not fully replicated (majority only)
Goal: full replication
 - Only proposer knows when entry is chosen
Goal: all servers know about chosen entries
- **Solution part 1/4: keep retrying Accept RPCs until all acceptors respond (in background)**
 - Fully replicates most entries
- **Solution part 2/4: track chosen entries**
 - **Mark entries** that are known to be chosen:
acceptedProposal[i] = ∞
 - Each server maintains **firstUnchosenIndex**: index of earliest log entry not marked as chosen

Full Disclosure, cont'd

- **Solution part 3/4: proposer tells acceptors about chosen entries**
 - Proposer includes its firstUnchosenIndex in Accept RPCs.
 - Acceptor marks all entries i chosen if:
 - $i < \text{request.firstUnchosenIndex}$
 - $\text{acceptedProposal}[i] == \text{request.proposal}$
 - Result: acceptors know about *most* chosen entries

| log index | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|------------------|----------|----------|----------|-----|----------|-----|---|---|---|
| acceptedProposal | ∞ | ∞ | ∞ | 2.5 | ∞ | 3.4 | | | |

before Accept

... Accept(proposal = 3.4, index=8, value = v, firstUnchosenIndex = 7) ...

| | | | | | | | | |
|----------|----------|----------|-----|----------|----------|--|-----|--|
| ∞ | ∞ | ∞ | 2.5 | ∞ | ∞ | | 3.4 | |
|----------|----------|----------|-----|----------|----------|--|-----|--|

after Accept

Still don't have complete information

Full Disclosure, cont'd

- **Solution part 4/4: entries from old leaders**
 - Acceptor returns its `firstUnchosenIndex` in `Accept` replies
 - If proposer's `firstUnchosenIndex` > `firstUnchosenIndex` from response, then proposer sends **Success** RPC (in background)
- **Success(index, v): notifies acceptor of chosen entry:**
 - `acceptedValue[index] = v`
 - `acceptedProposal[index] = ∞`
 - return `firstUnchosenIndex`
 - Proposer sends additional `Success` RPCs, if needed

Client Protocol

- **Send commands to leader**
 - If leader unknown, contact any server
 - If contacted server not leader, it will redirect to leader
- **Leader does not respond until command has been chosen for log entry and executed by leader's state machine**
- **If request times out (e.g., leader crash):**
 - Client reissues command to some other server
 - Eventually redirected to new leader
 - Retry request with new leader

Client Protocol, cont'd

- **What if leader crashes after executing command but before responding?**
 - Must not execute command twice
- **Solution: client embeds a unique id in each command**
 - Server includes id in log entry
 - State machine records most recent command executed for each client
 - Before executing command, state machine checks to see if command already executed, if so:
 - Ignore new command
 - Return response from old command
- **Result: **exactly-once semantics** as long as client doesn't crash**

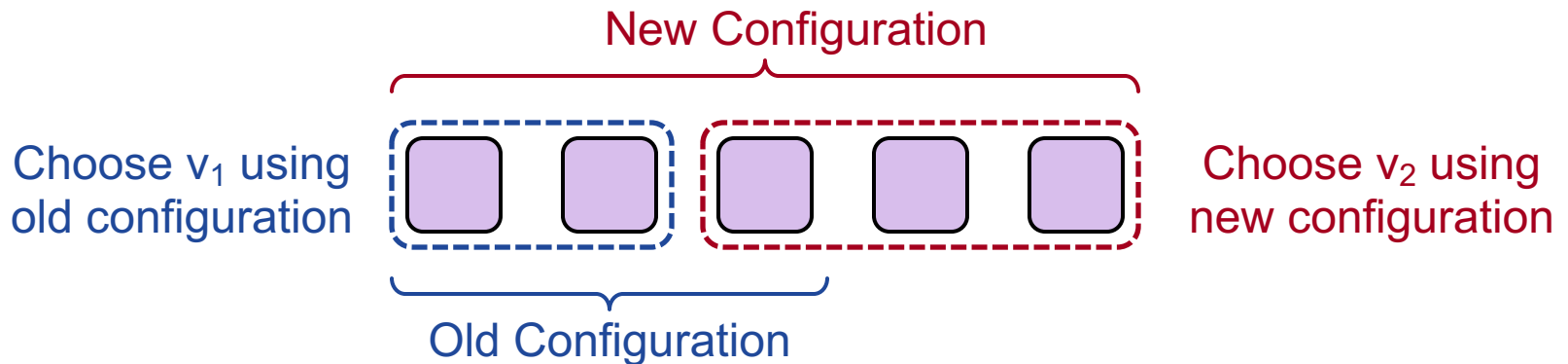
Configuration Changes

- **System configuration:**
 - ID, address for each server
 - Determines what constitutes a majority
- **Consensus mechanism must support changes in the configuration:**
 - Replace failed machine
 - Change degree of replication

Configuration Changes, cont'd

- **Safety requirement:**

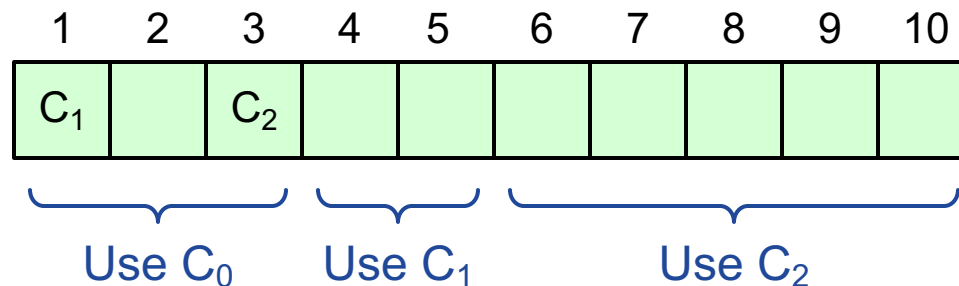
- During configuration changes, it must not be possible for different majorities to choose different values for the same log entry:



Configuration Changes, cont'd

- **Paxos solution: use the log to manage configuration changes:**

- Configuration is stored as a log entry
- Replicated just like any other log entry
- Configuration for choosing entry i determined by entry $i-\alpha$.
Suppose $\alpha = 3$:



- **Notes:**

- α limits concurrency: can't choose entry $i+\alpha$ until entry i chosen
- Issue no-op commands if needed to complete change quickly

Paxos Summary

- **Basic Paxos:**
 - Prepare phase
 - Accept phase
- **Multi-Paxos:**
 - Choosing log entries
 - Leader election
 - Eliminating most Prepare requests
 - Full information propagation
- **Client protocol**
- **Configuration changes**