BYZANTINE AGREEMENT AND PAXOS

Consensus or Byzantine Agreement

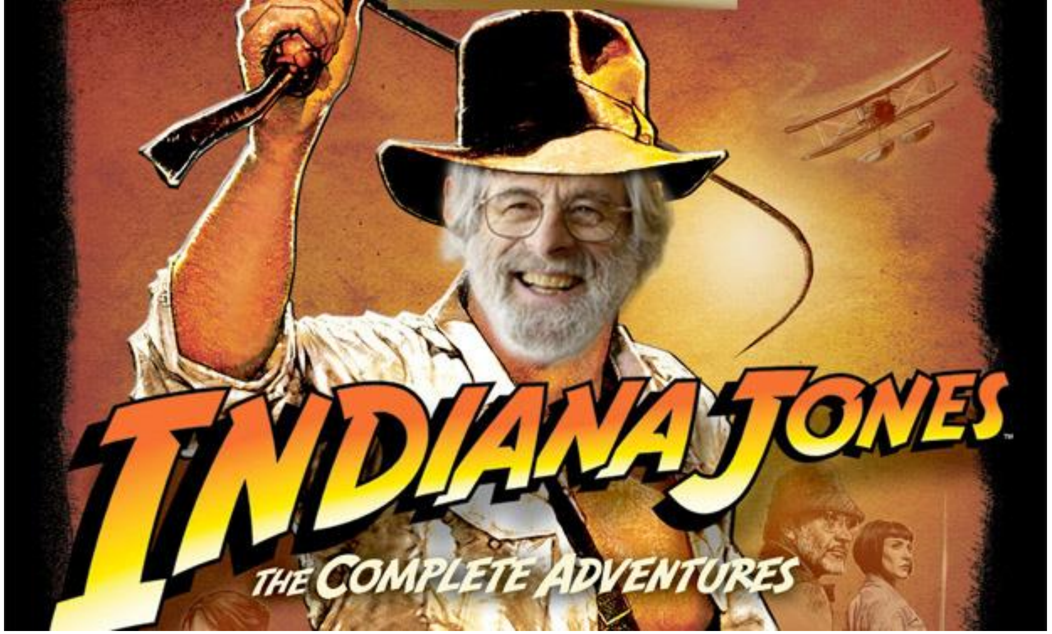
- Malicious Failures (byzantine failures)
- General sends a binary value to n-1 participants such that:

1. Agreement: All correct participants agree on same value
2. Validity: If general is correct, every participant agrees on the value general sends

PAXOS

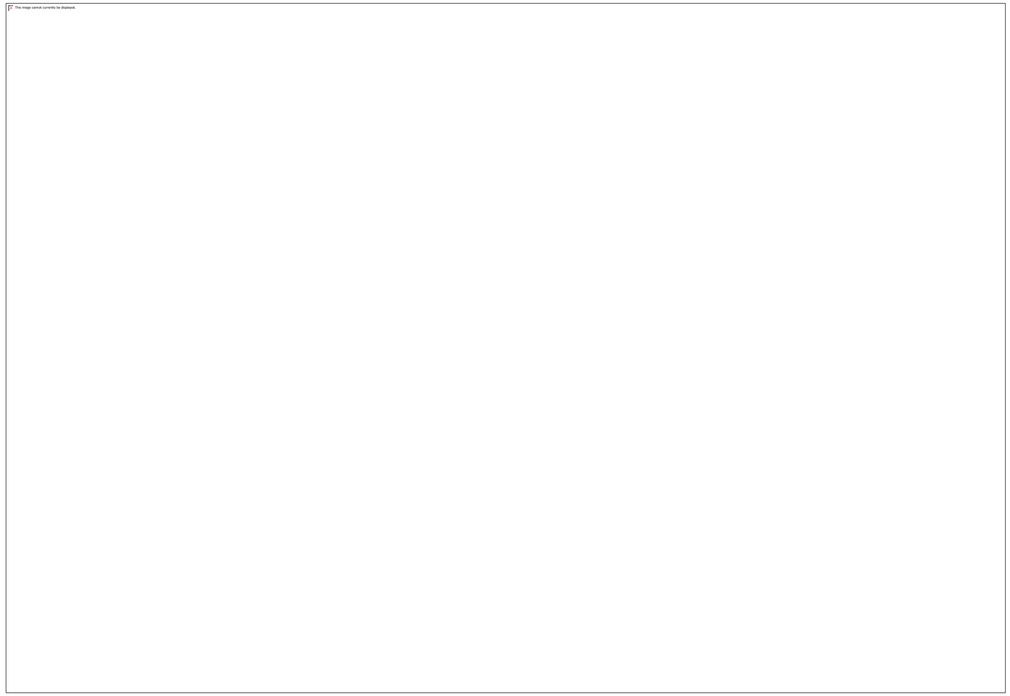
Thanks for slides: Idit Kaider, John Ousterhout and Diego Angaro

Paxos



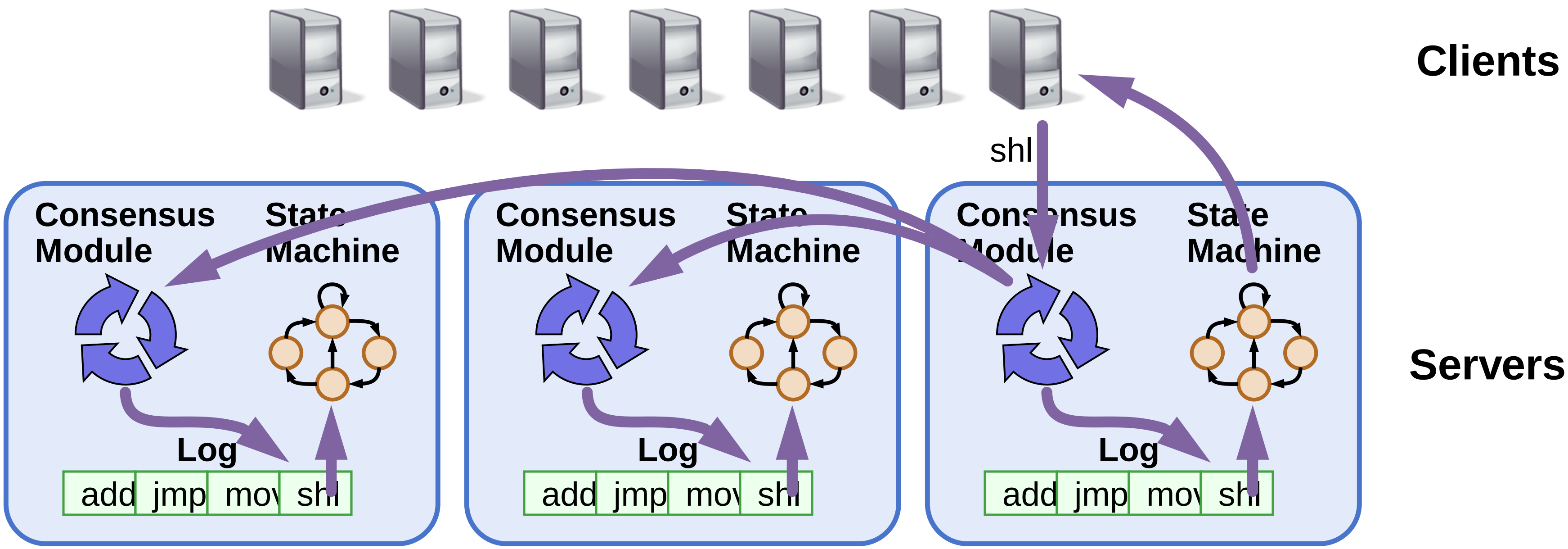
- Lamport the archeologist and the "Part-time Parliament" of Paxos:
  - The Part-time Parliament, TOCS 1998
  - Paxos Made Simple, *ACM SIGACT News 2001.*
  - Paxos Made Live, PODC 2007
  - ....

Distributed State Machine



- Fault-tolerance through replication.
  - Need to ensure that replicas remain consistent.
  - Replicas must process requests in the same order.

Goal: Replicated Log



- Replicated log => replicated state machine
  - All servers execute same commands in same order
- Consensus module ensures proper log replication

Correctness

- Safety
  - Only a value that has been proposed may be chosen.
  - Only a single value is chosen.
  - A node never learns that a value has been chosen unless it actually has been.
- Liveness
  - Some proposed value is eventually chosen.
  - If a value has been chosen, a node can eventually learn the value

Paxos System Assumptions

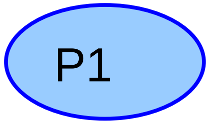
- Paxos is an asynchronous consensus algorithm
  - Asynchronous networks
- Set of processes is known a-priori
- Failure model: fail-stop (not Byzantine), delayed/lost messages

Paxos Properties

- Paxos guarantees safety.
  - Consensus is a stable property: once reached it is never violated; the agreed value is not changed.
- Paxos does not guaranteed liveness.
  - Consensus is reached if “a large enough subnetwork...is non-faulty for a long enough time.”
  - Otherwise Paxos might never terminate.

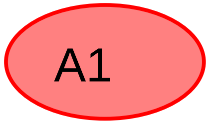
Paxos Participants

- ***Proposer***



  - Suggests values for consideration by Acceptors.
  - Advocates for a client.

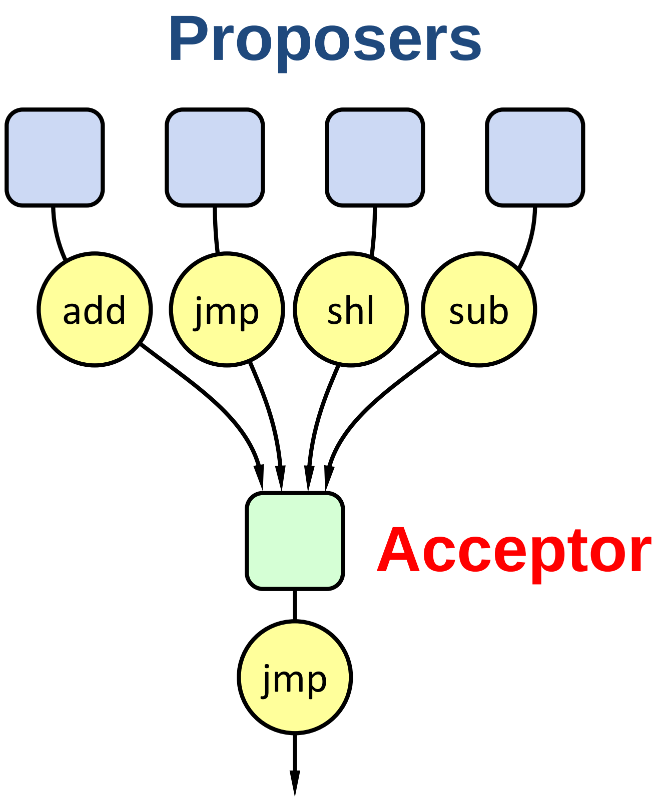
- ***Acceptor***



  - Considers the values proposed by proposers.
  - Renders an accept/reject decision.
- ***Learner*** ***(will ignore most of the time)***
  - Learns the chosen value.
- In practice, each node will usually play all three roles.

Strawman: Single Acceptor

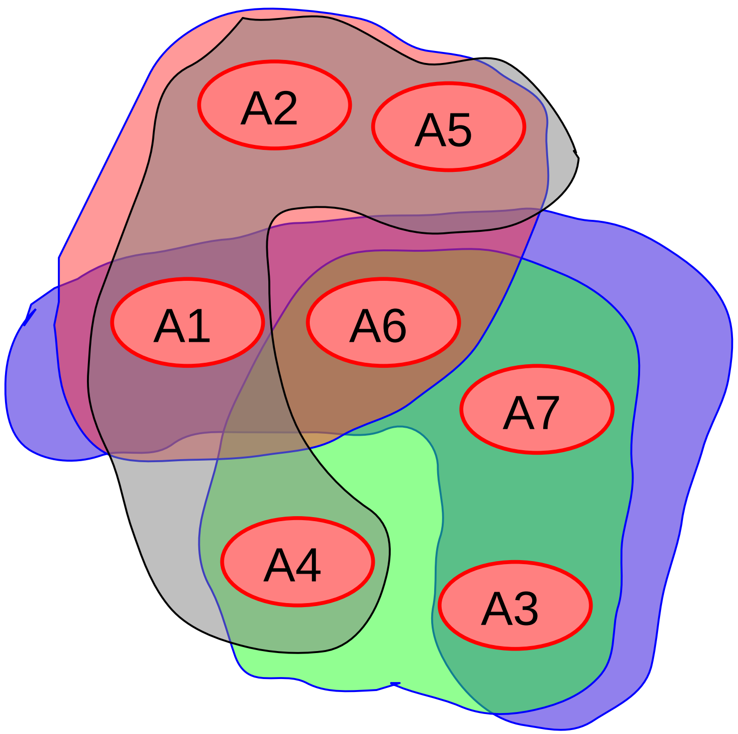
- Simple (incorrect) approach: ONE acceptor chooses value
- What if acceptor crashes after choosing?
- Solution: quorum
  - Multiple acceptors (3, 5, ...)
  - Value v is chosen if accepted by majority of acceptors



If one acceptor crashes, chosen value still available

Majority Quorums

- Majority / “Quorum”
  - A set of acceptors consisting of more than half of all acceptors.
- Any two quorums have a nonempty intersection.
- In a system with 2F+1 acceptors, F acceptors can fail and we'll be OK.

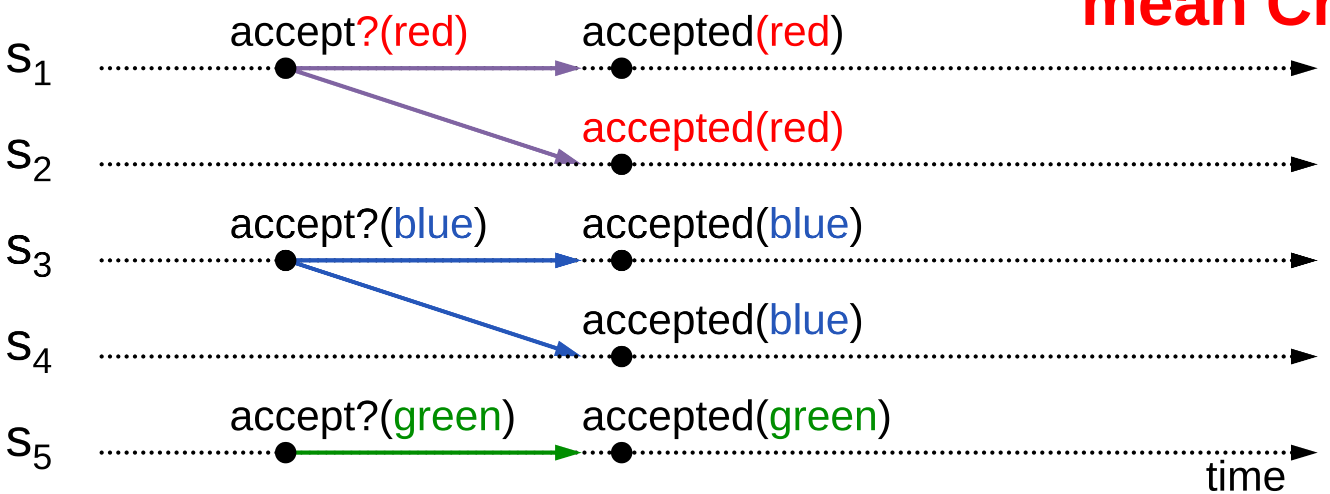


Quorums in a system with seven acceptors.

Problem: Split Votes

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

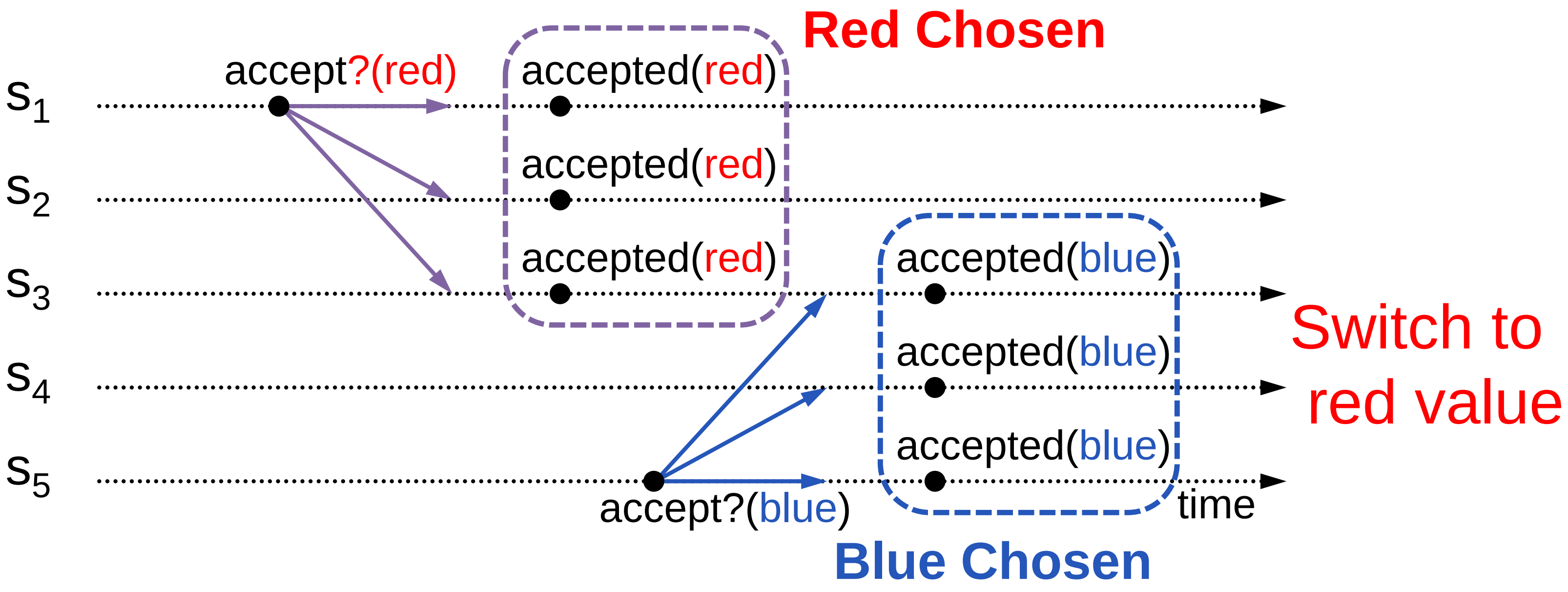
Accepted does not mean Chosen!



Acceptors must sometimes accept multiple (different) values (change its mind)-> multiple phases

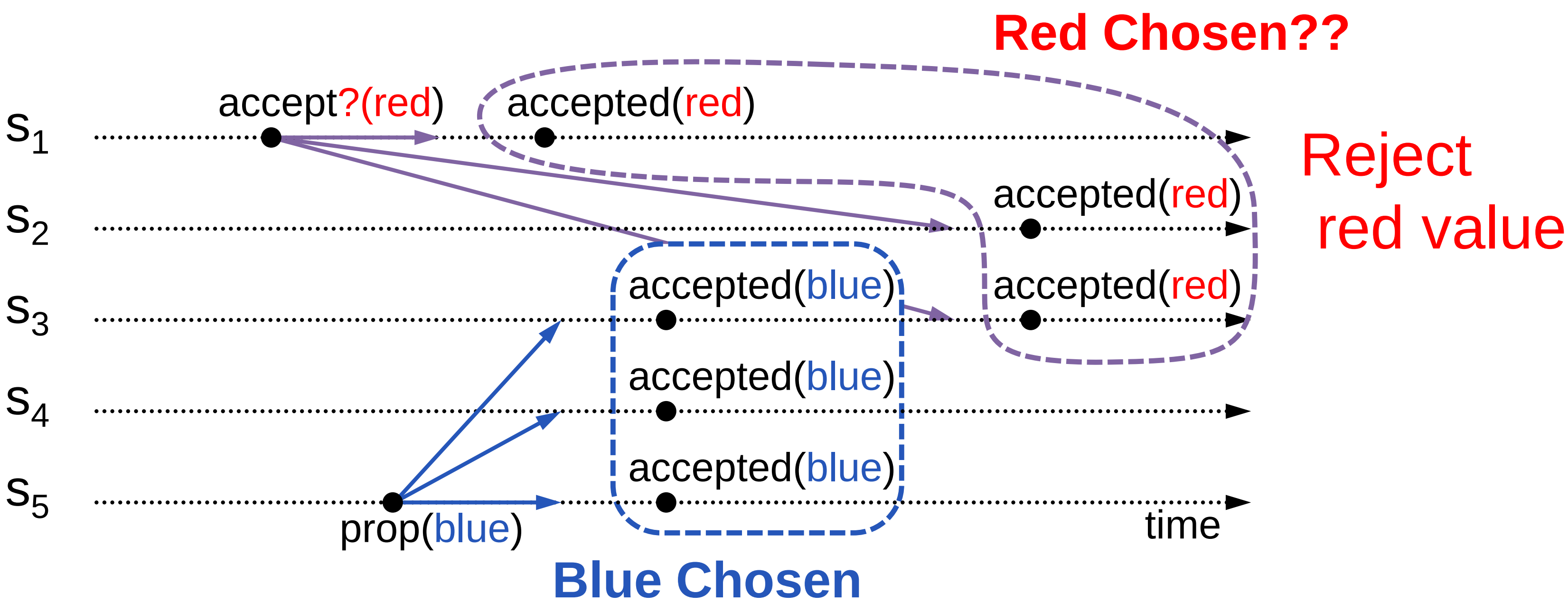
Problem: Conflicting Choices

- What if Acceptor accepts every value it receives?



Once a value is chosen, future proposals must propose/choose the **same** value (2-phase protocol)

Conflicting Choices, cont'd



- s_5 needn't propose red (it hasn't been chosen yet)
- s_1's proposal must be aborted (**s_3 must reject it**)

Must **order** proposals, reject old ones

Overview of the Paxos Algorithm

- Leader based: each client has an estimate of who is the current leader
- To order an operation, a client sends it to current leader
- The leader sequences the operation and launches an algorithm to ensure agreement

The Consensus Algorithm Structure

- Two phases
  - Phase 1: Prepare request
  - Phase 2: Accept request
- Leader contacts a majority in each phase
- There may be multiple concurrent leaders
- A proposal consists of a unique ballot ***number*** and a proposed ***value.***

Ballot Numbers

- *Ballots* distinguish among values proposed by different leaders
  - Unique, locally monotonically increasing
  - Processes respond only to leader with highest ballot
- Pairs ⟨**num, process id**⟩
- $\langle \mathbf{n_1, p_1} \rangle > \langle \mathbf{n_2, p_2} \rangle$
  - If $\mathbf{n_1 > n_2}$
  - Or $\mathbf{n_1 = n_2}$ and $\mathbf{p_1 > p_2}$
- If latest known ballot is ⟨**n, q**⟩ then
  - p chooses ⟨**n+1, p**⟩

The Two Phases of Paxos

- Phase 1: prepare or Leader Election
  - If you *want to be leader*
    - Choose new unique ballot number
    - Learn outcome of all smaller ballots from majority
- Phase 2: accept
  - Leader *proposes* a value with its ballot number
  - Leader gets majority to *accept* its proposal
  - A value accepted by a majority can be decided
- Phase 3: inform
  - Leader send final value to all

Basic Paxos Skeleton

Phase 1a: **“Prepare”**
Select proposal number *N* and send a ***prepare(N)*** request to a all acceptors.

Proposer

Phase 1b: **“Promise”**
If N > *number of any previous promises or acceptances*,
- send a ***promise(N)*** response

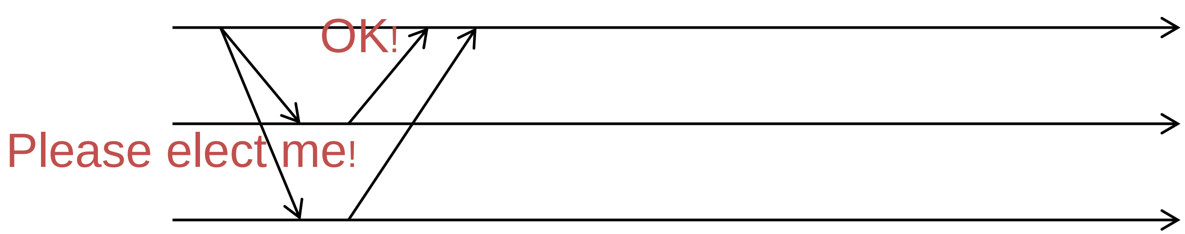
Phase 2a: **“Accept!”**
If proposer received promise responses from a majority,
- send an ***accept(N)*** request to all acceptors

Acceptor

Phase 2b: **“Accepted”**
If N >= *number of any previous promise*,
accept the proposal
- send an ***accepted*** notification to the learner

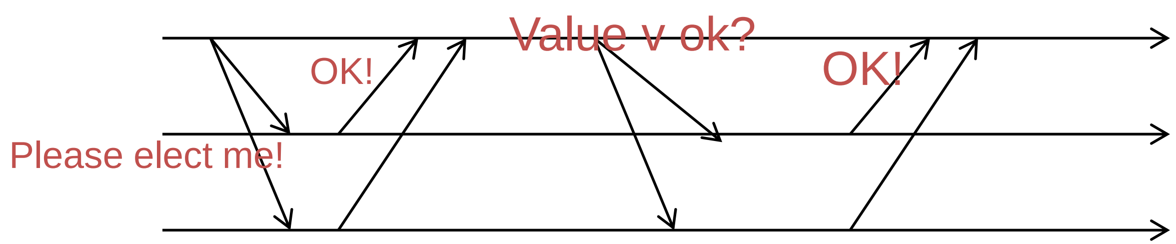
Phase 1 – Election

- *Potential leader chooses a unique ballot id, higher than seen anything*
- *Sends to all processes*
- *Processes respond to highest ballot id*
- *If a process has in a previous round decided on a value v', it includes value v' in its response*
- *If majority* respond OK then you are the leader



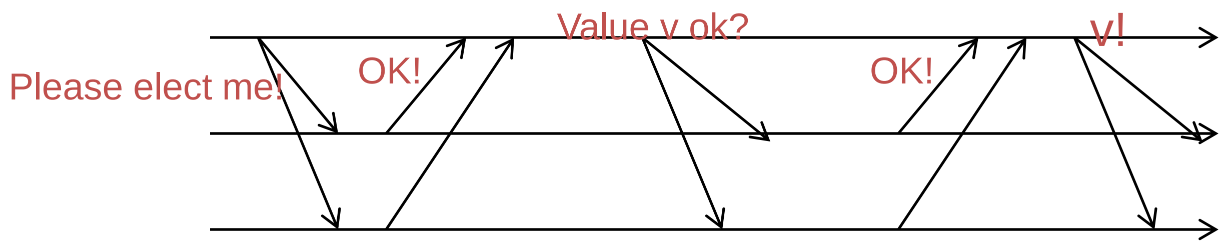
Phase 2 – Accept

- Leader sends proposed value v to all
  - use v=v' if some process already decided and sent you its decided value v'



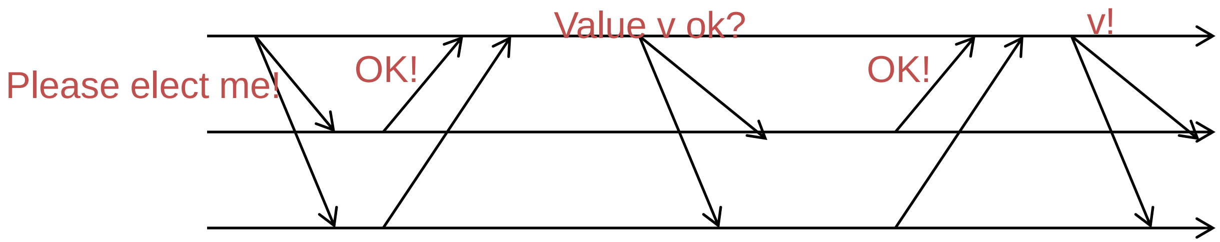
Phase 3 – Decision and Inform

- If leader hears a majority of OKs, it lets everyone know of the decision
- Recipients receive decision



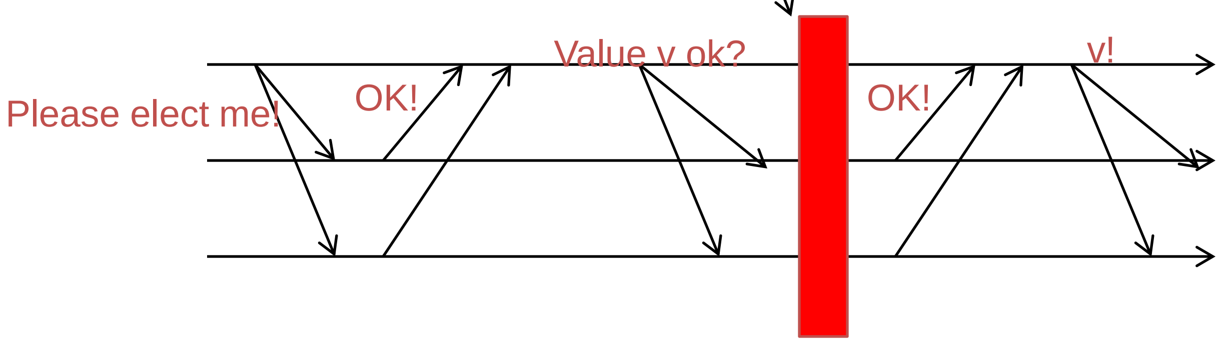
Which is the point of no-return?

- That is, when is consensus reached in the system?



Which is the point of no-return?

- If/when a majority of processes hear proposed value and accept it (i.e., have respond(ed) OK!)
- Processes *may not know it yet*, but a decision has been made for the group
  - Even leader does not know it yet
- What if leader fails after that?
  - Keep having rounds until some round completes



Paxos - Variables

$BallotNum_i$, initially $\langle 0,0 \rangle$

Latest ballot p_i took part in (phase 1)

$AcceptNum_i$, initially $\langle 0,0 \rangle$

Latest ballot p_i accepted a value in (phase 2)

$AcceptVal_i$, initially $\perp$

Latest accepted value (phase 2)

Phase I: Prepare - Leader

If I want to be leader **then**

BallotNum ← ⟨BallotNum.num+1, myId⟩

send ("prepare", BallotNum) to all

- Goal: contact other processes, ask them to join this ballot, and get information about possible past decisions

Phase I: Prepare - Cohort

- Upon receive ("prepare", bal) from *i*

  if bal $\geq$ BallotNum **then**

  BallotNum $\leftarrow$ bal

  send ("ack", bal, AcceptNum, AcceptVal) to *i*

This is a higher ballot than my current, I better join it

This is a promise not to accept ballots smaller than bal in the future

Tell the leader about my latest accepted value and what ballot it was accepted in

Phase II: Accept - Leader

Upon receive ("ack", BallotNum, b, val) from *majority*

if all vals = ⊥ **then** myVal = initial value

else myVal = received val with highest b

send ("accept", BallotNum, myVal) to all

The value accepted in the highest ballot might have been decided, I better propose this value

Phase II: Accept - Cohort

Upon receive ("accept", b, v)

if b $\geq$ BallotNum **then**

This is not from an old ballot

AcceptNum $\leftarrow$ b; AcceptVal $\leftarrow$ v   */* accept proposal */*

send ("accept", b, v) to leader

Phase III: Decide and Inform

Upon receive ("accept", b, v) from *majority*

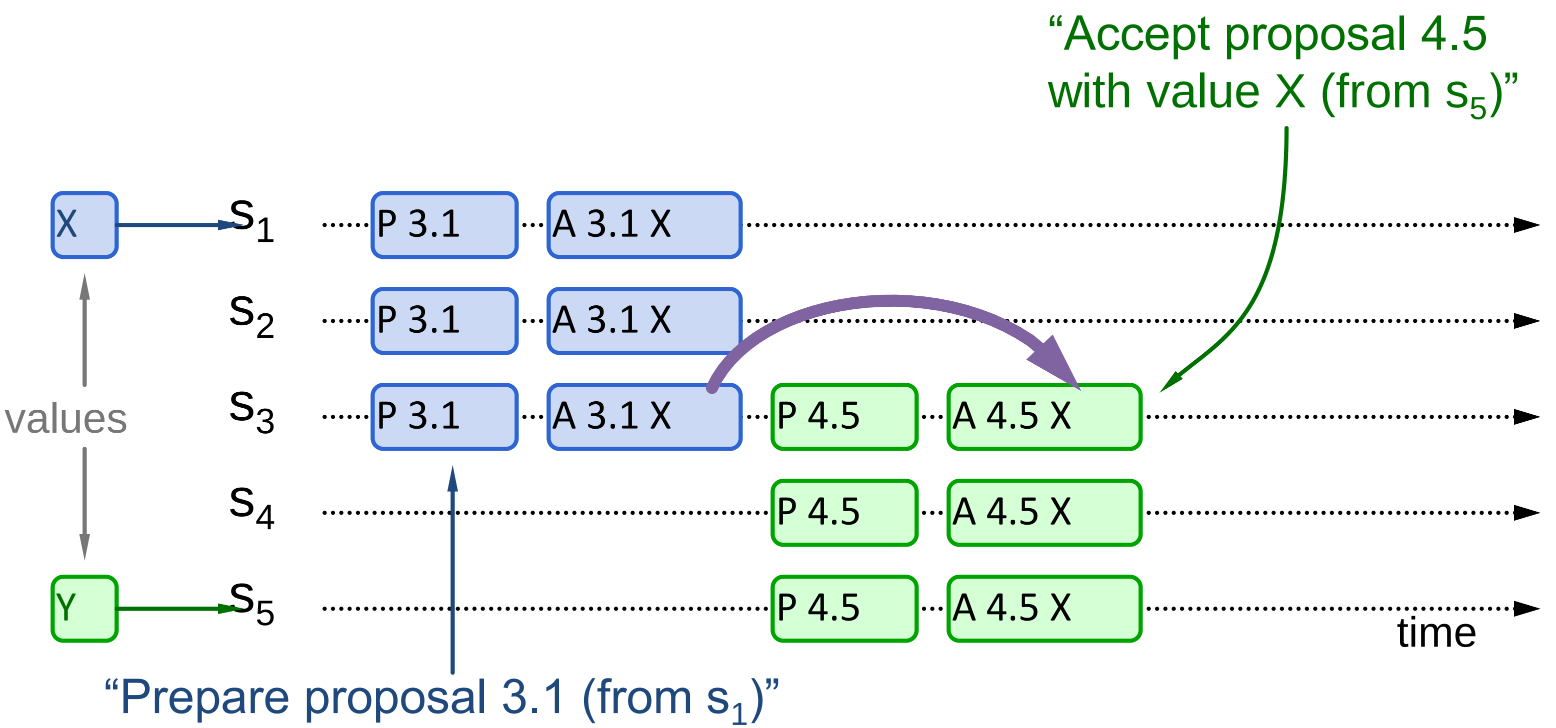
  decide v

send ("accept", b, v) **to all**)

Basic Paxos Examples

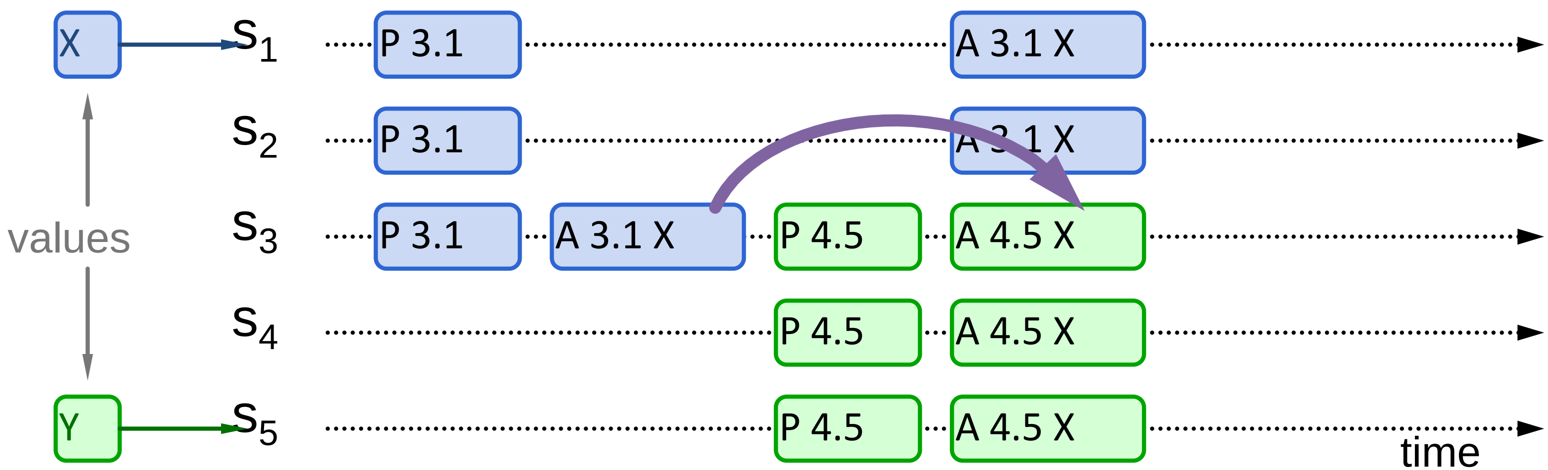
Three possibilities when later proposal prepares:

1. Previous value already chosen:
   – New proposer will find it and use it



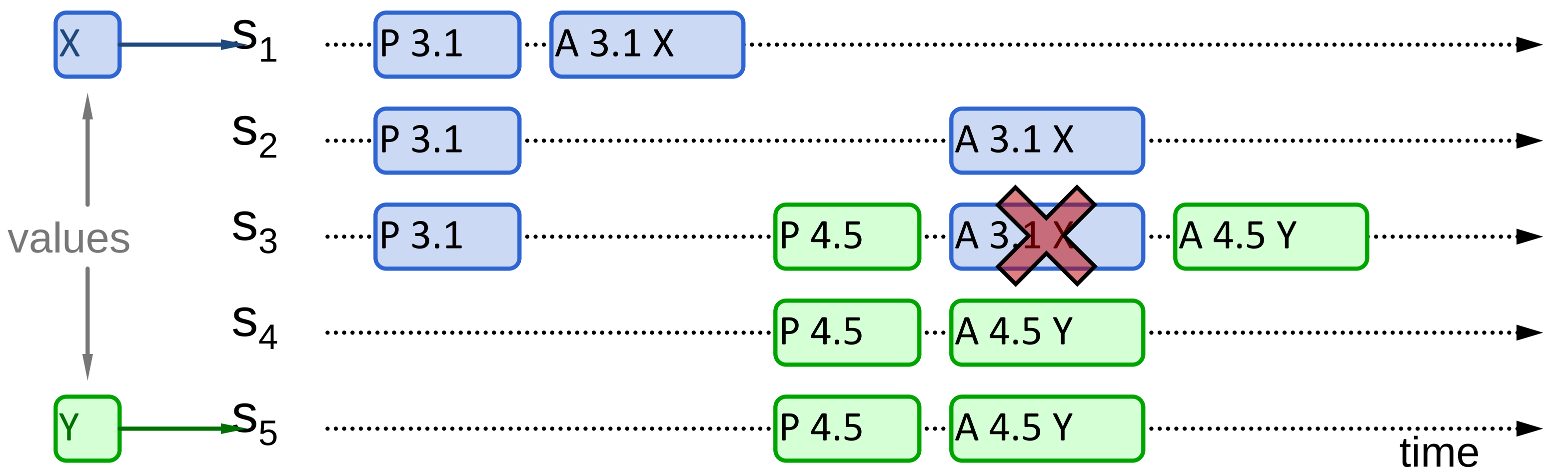
Basic Paxos Examples, cont'd

2. Previous value not chosen, but new proposer sees it:
   – New proposer will use existing value
   – Both proposers can succeed



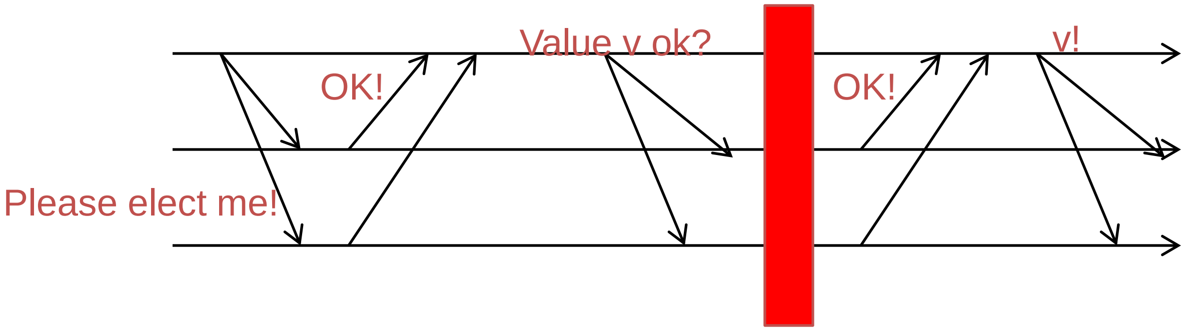
Basic Paxos Examples, cont'd

3. Previous value not chosen, new proposer doesn't see it:
   - New proposer chooses its own value
   - Older proposal blocked



Safety

- If some round has a majority hearing proposed value v' and accepting it (middle of Phase 2), then subsequently each round either:
  - 1) the round chooses v' as decision or
  - 2) the round fails
- Proof:
  - Potential leader waits for majority of OKs in Phase 1
  - At least one will contain v' (because two majorities intersect)
  - It will choose to send out v' in Phase 2
- Success requires majority, and any two majorities intersect

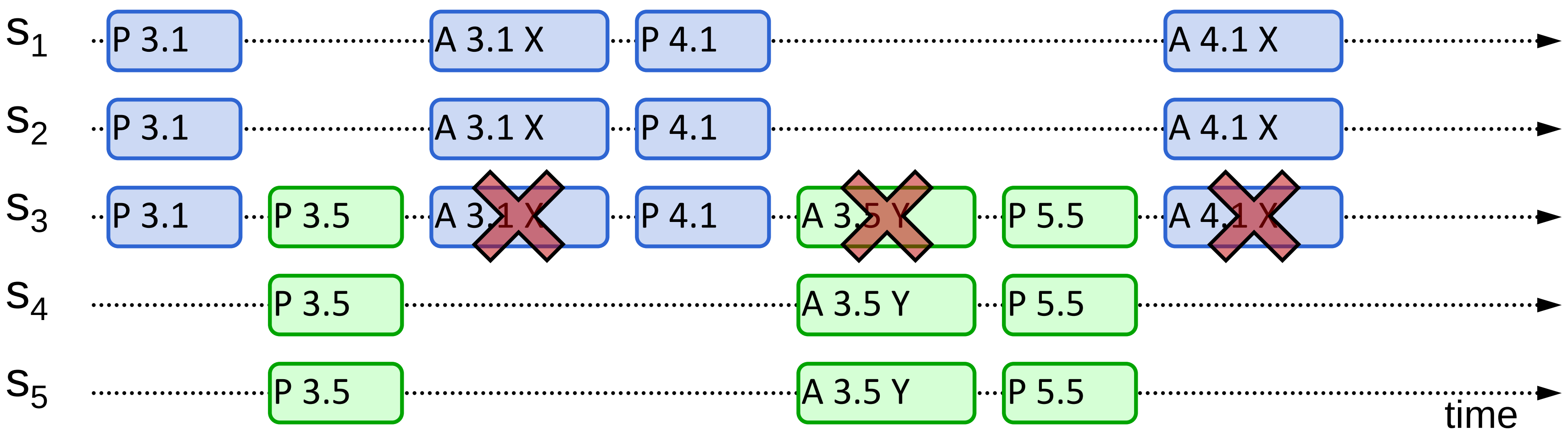


What could go wrong?

- Process fails
  - Majority does not include it
  - When process restarts, it uses log to retrieve past decisions and past-seen ballot ids.
- Leader fails
  - Start another round
- Note that anyone can start a round any time
- Protocol may never end!
  - Impossibility result not violated
  - If things go well sometime in the future, consensus reached

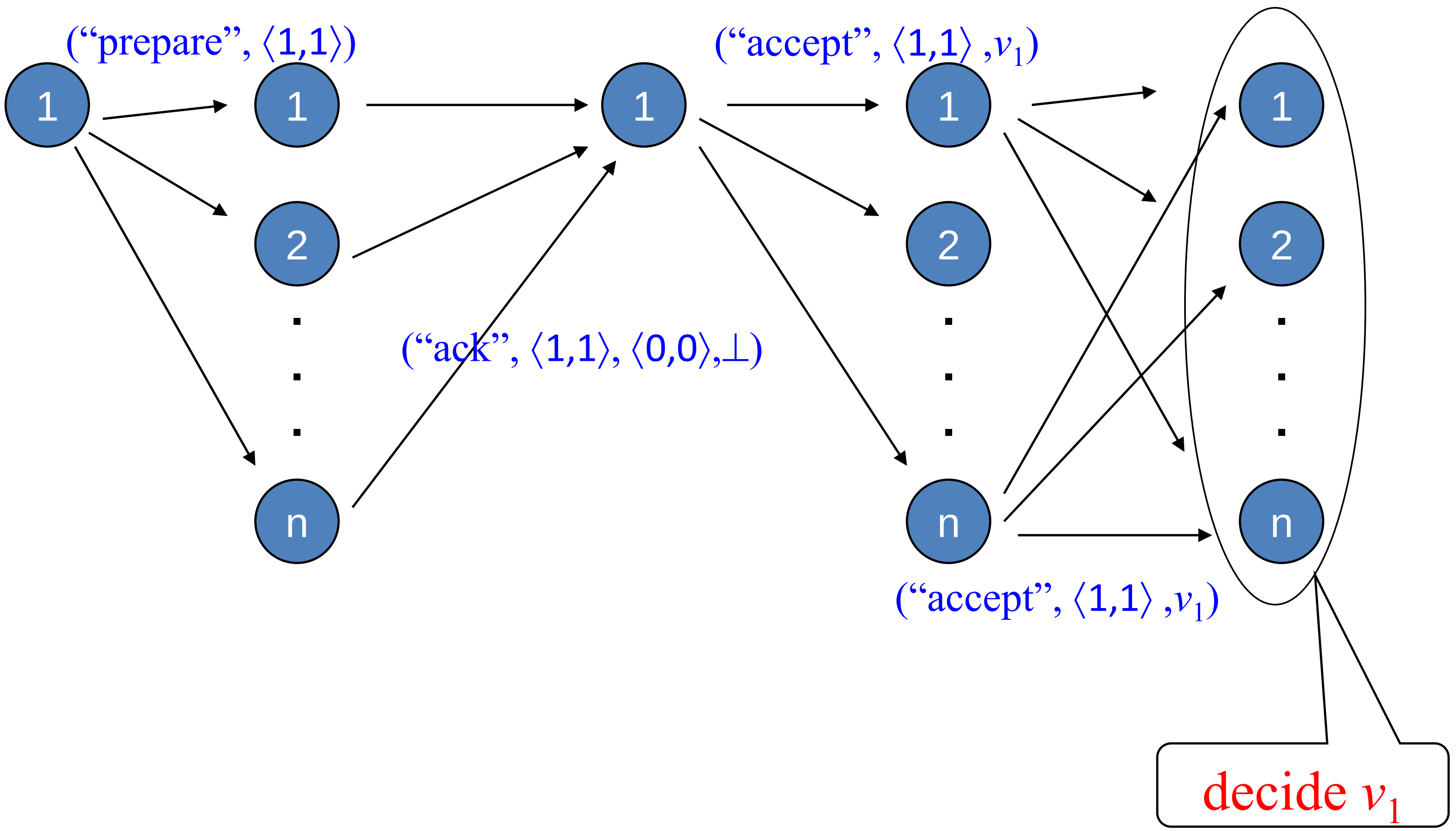
Liveness

- Competing proposers can livelock:

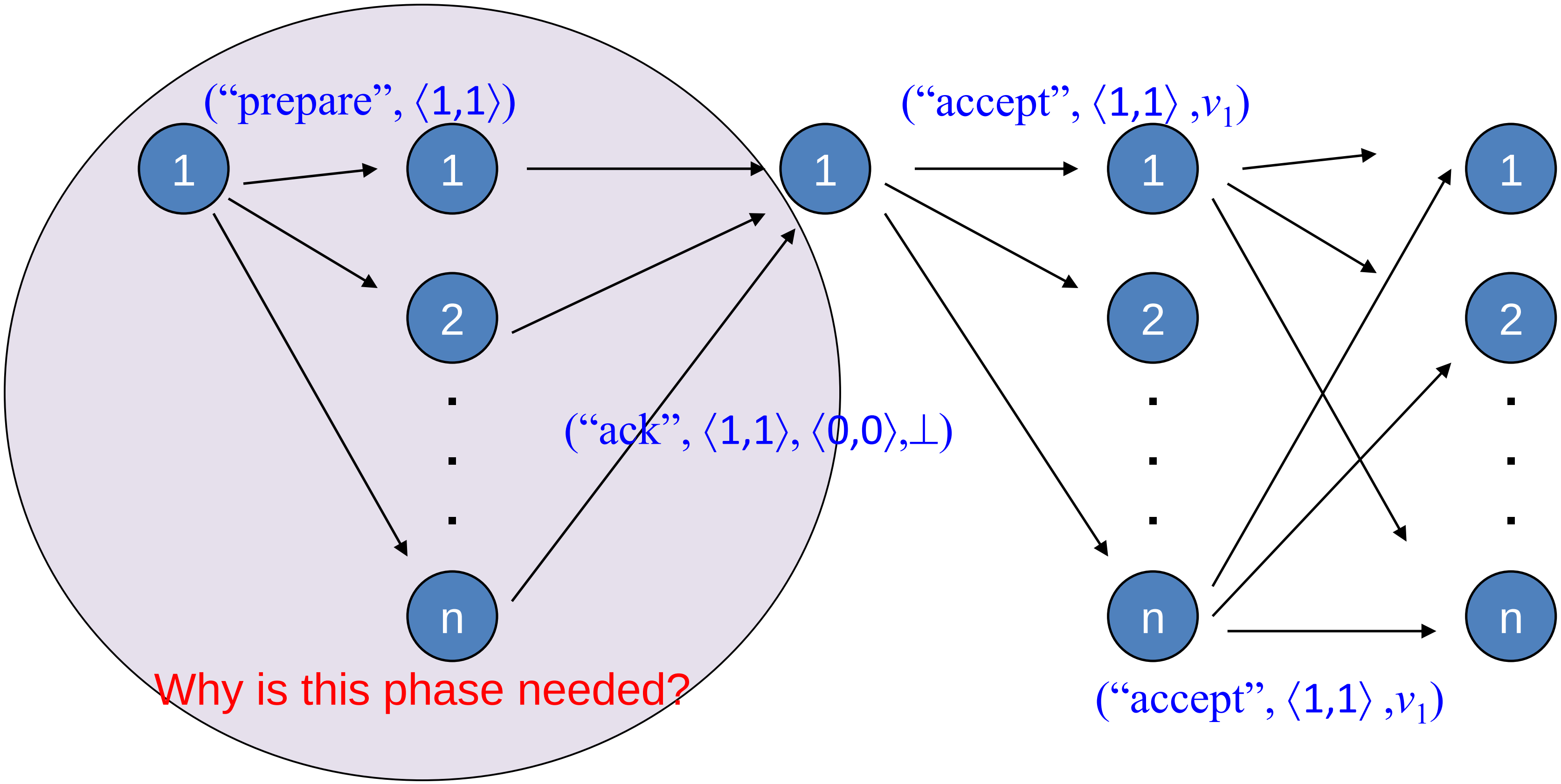


- One solution: randomized delay before restarting
  - Give other proposers a chance to finish choosing

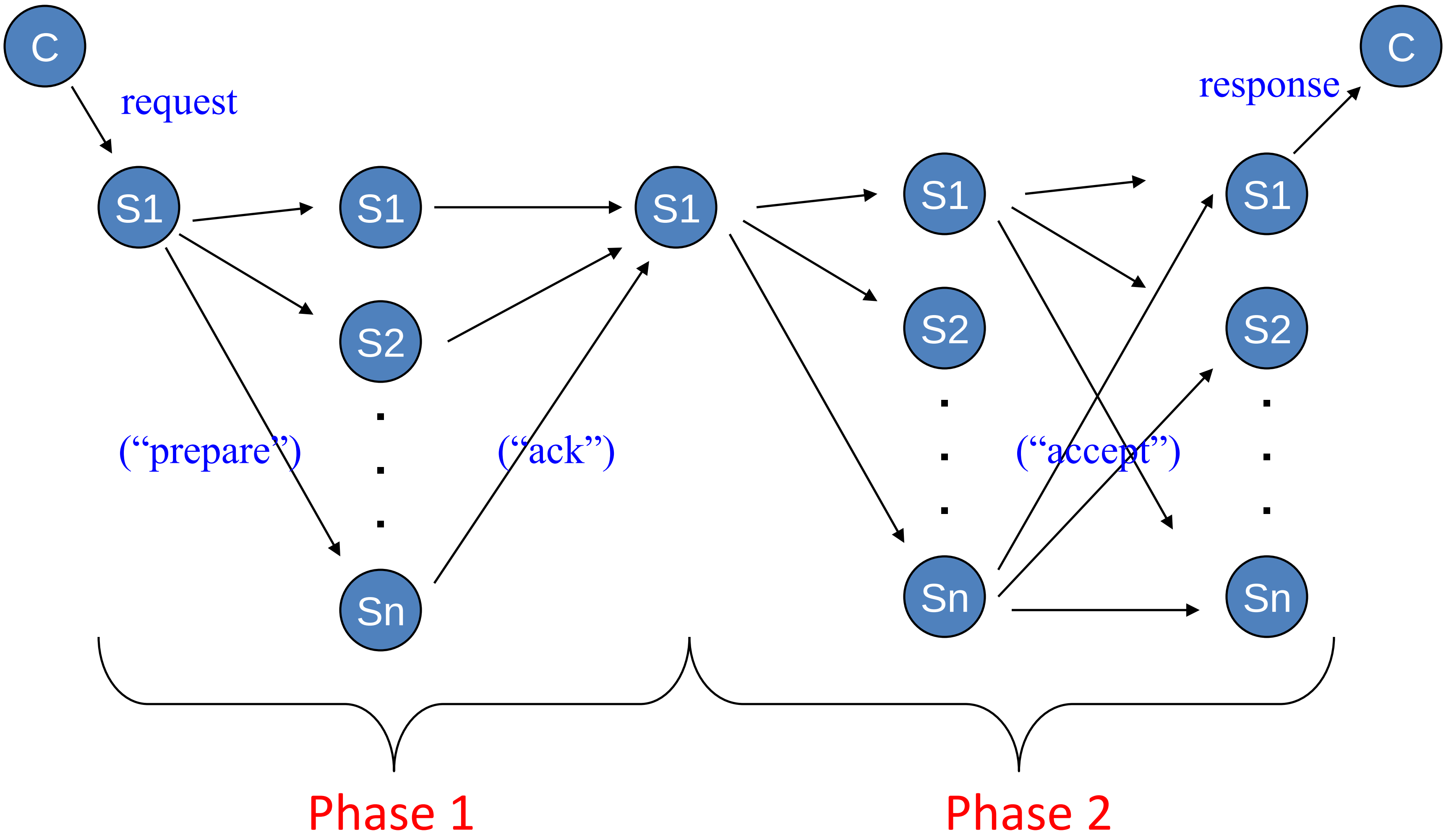
In Failure-Free Execution



Performance?



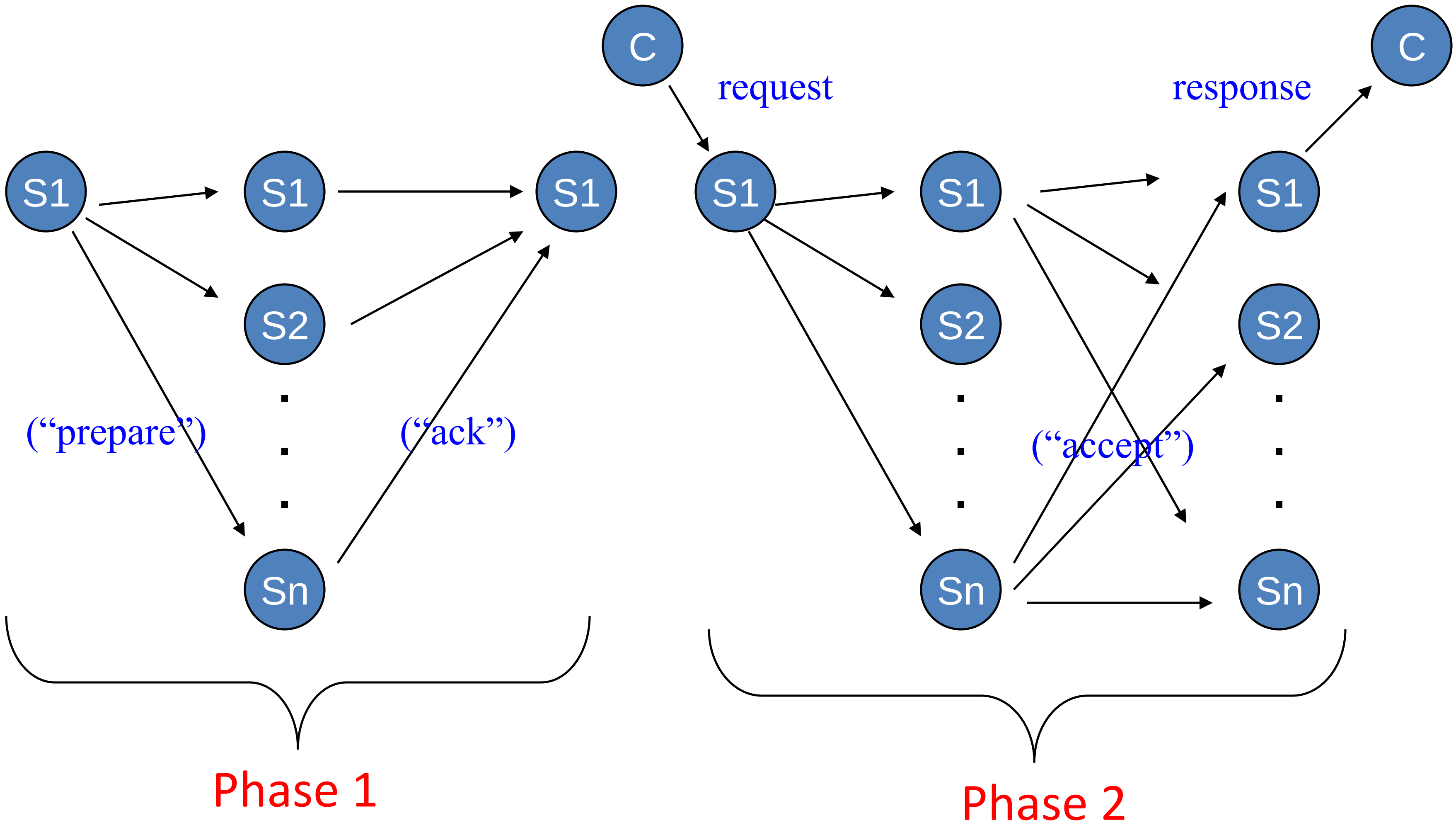
Failure-Free Execution



Observation

- In Phase 1, no consensus values are sent:
  - Leader chooses largest unique ballot number
  - Gets a majority to “vote” for this ballot number
  - Learns the outcome of all smaller ballots
- In Phase 2, leader proposes its own initial value or latest value it learned in Phase 1

Failure free execution



Optimization: Multi-Paxos

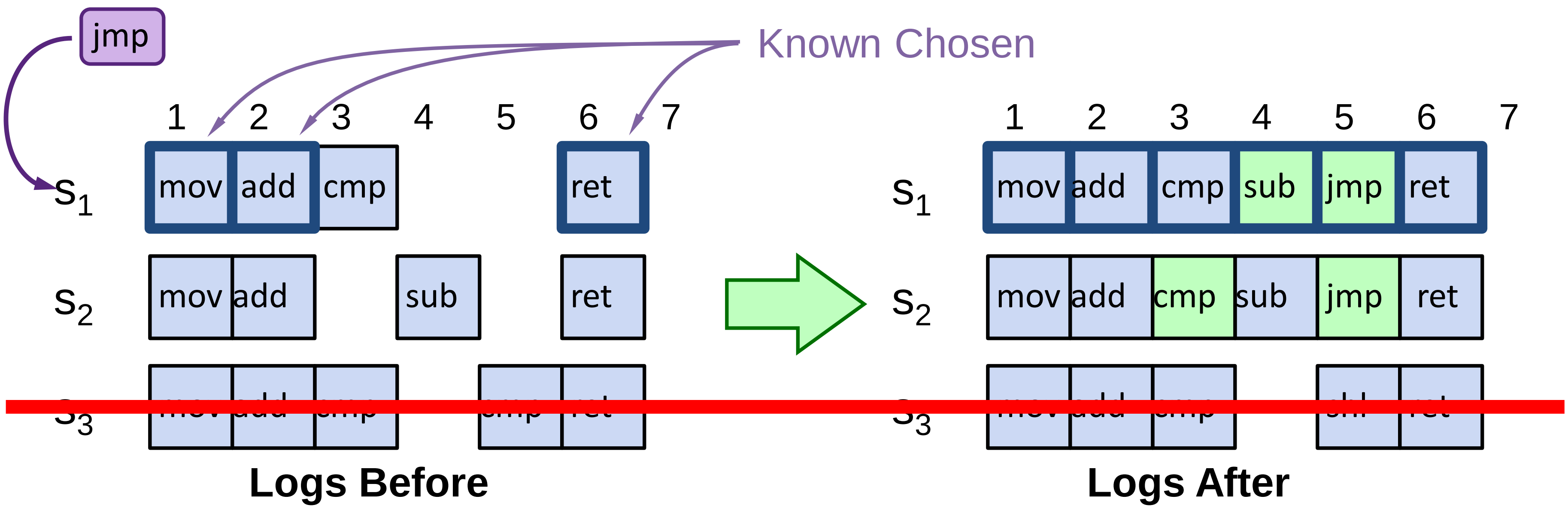
- Run Phase 1 only when the leader changes
  - Phase 1 is called "view change" or "leader election"
  - Phase 2 is the "normal mode"
- A leader is for The Log.
- Each message includes BallotNum (from the last Phase 1) and ReqNum
- Respond only to messages with the "right" BallotNum

Multi-Paxos

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

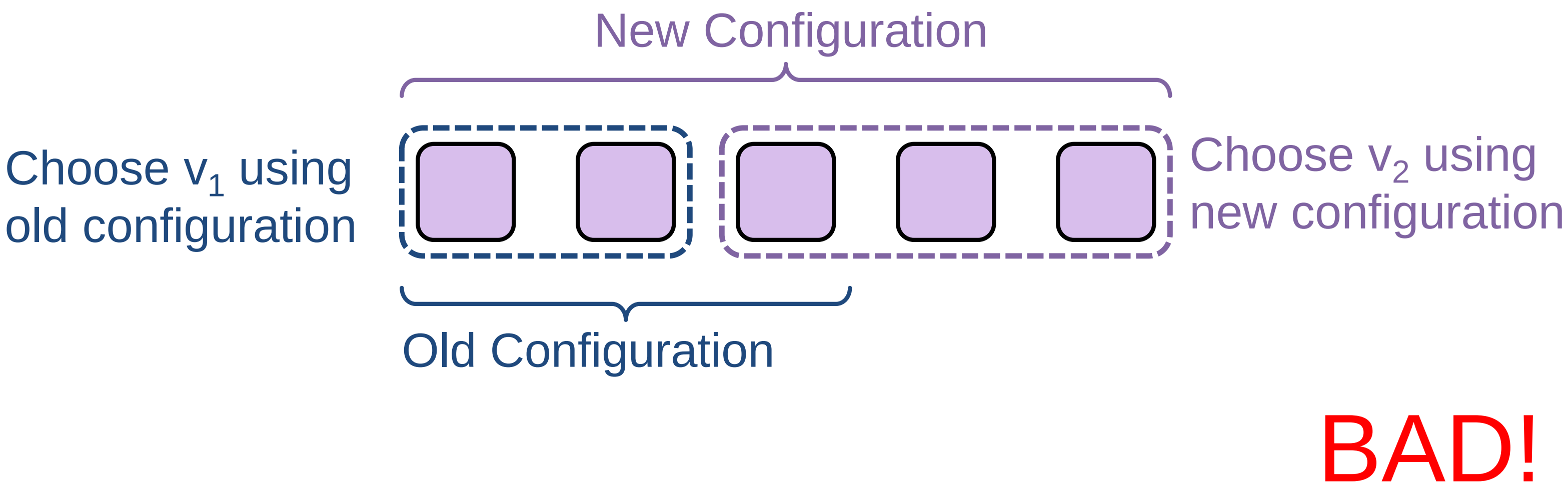
Selecting Log Entries

- When leader receives a request arrives from a client:
  - Find first log entry not known to be chosen
  - Run Paxos to propose client’s command for this index
  - Prepare returns acceptedValue?
    - Yes: finish choosing acceptedValue, start again
    - No: choose client’s command



Configuration Changes

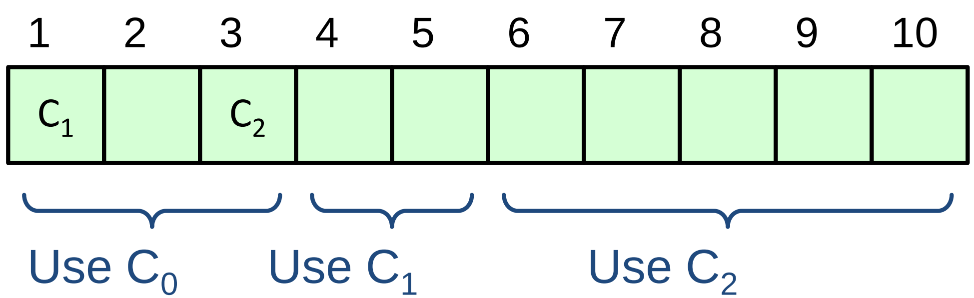
- Safety requirement:
  - During configuration changes, it must not be possible for different majorities to choose different values for the same log entry:



Configuration Changes, cont'd

- 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-α.

Suppose α = 3:



- Notes:
  - α determines concurrency:
    - α concurrent log entries.
    - But can't choose entry i+α until entry i chosen
  - Issue no-op commands if needed to complete change quickly


That's all Folks!