CS 582: Distributed Systems

Paxos

Dr. Zafar Ayyub Qazi

Fall 2024

Acknowledgments: Slides use some material from John Ousterhout (Stanford University

A note before we start

- This is the most important module of the course and the most tricky
  - We will cover several consensus protocols, e.g., Paxos, Raft, PBFT, ….
- An important goal: Build strong intuition for the key ideas
  - What are the key ideas? Why do they work? When they might not work?
- Requirement from you:
  - Attend classes, pay attention, and ask questions
  - Do recommended readings, homework questions, and assignment
  - Review these concepts after class to consolidate your understanding

Today’s Agenda

- Basic Paxos

Specific learning outcomes

By the end of today’s lecture, you should be able to:

❑ Explain how Basic Paxos works

❑ Analyze the design choices made in Basic Paxos

❑ Analyze and evaluate Paxos in terms of safety, liveness, and fault tolerance

Recap: Consistency Protocols

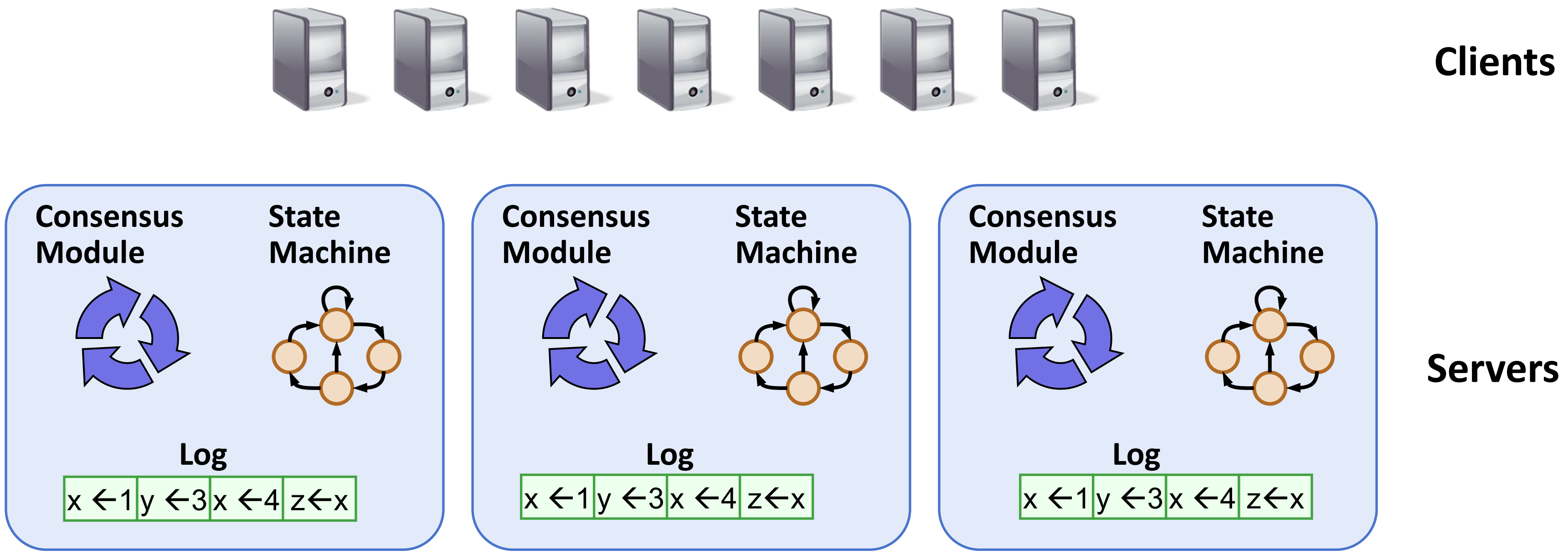
- Single phase primary-backup schemes
  - Safety: Can’t roll back updates, so safety can get violated
  - Liveness: doen’t provide liveness with failures
- Two Phase Commit (2-PC)
  - Safety: Allows for rolling back updates and making nodes consistent
  - Liveness: blocks in case of failures

Paxos

Paxos

- Paxos
  - Consensus algorithm
  - Proposed by Leslie Lamport in 1989
- Paxos and its variants widely used in industry
  - Zookeeper (Yahoo), Google Chubby, Google Spanner and many others

Raft Goal → Replicated Log



x ←1	y ←3	x ←4	z←x

x ←1	y ←3	x ←4	z←x

x ←1	y ←3	x ←4	z←x

- Replicated log => replicated state machine
  - All servers execute same commands in same order
- Consensus module ensures proper log replication
- System should make 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:
  - Uses multiple instances of basic Paxos (one for each log entry)

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
  - Only a value that has been proposed may be chosen

- 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

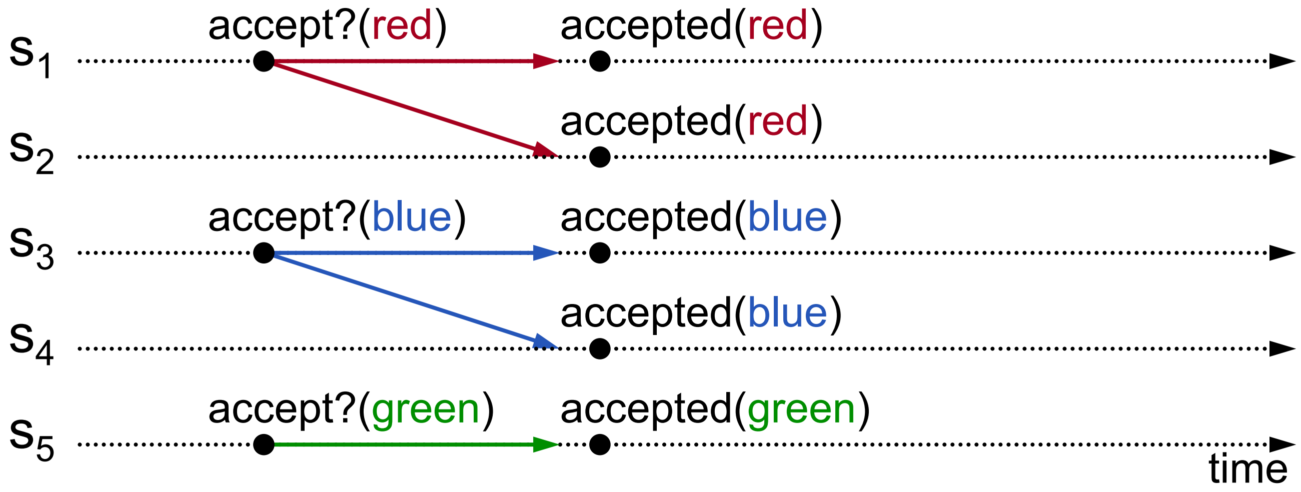
Paxos Components

- Proposers
  - Active: put forth particular values to be chosen
  - Handle client requests
- Acceptors
  - Passive: respond to messages from proposers
  - Responses represent votes that form a consensus
  - Store chosen value, state of the decision process
  - Want to know which value was chosen
- We will assume each Paxos server contains both components

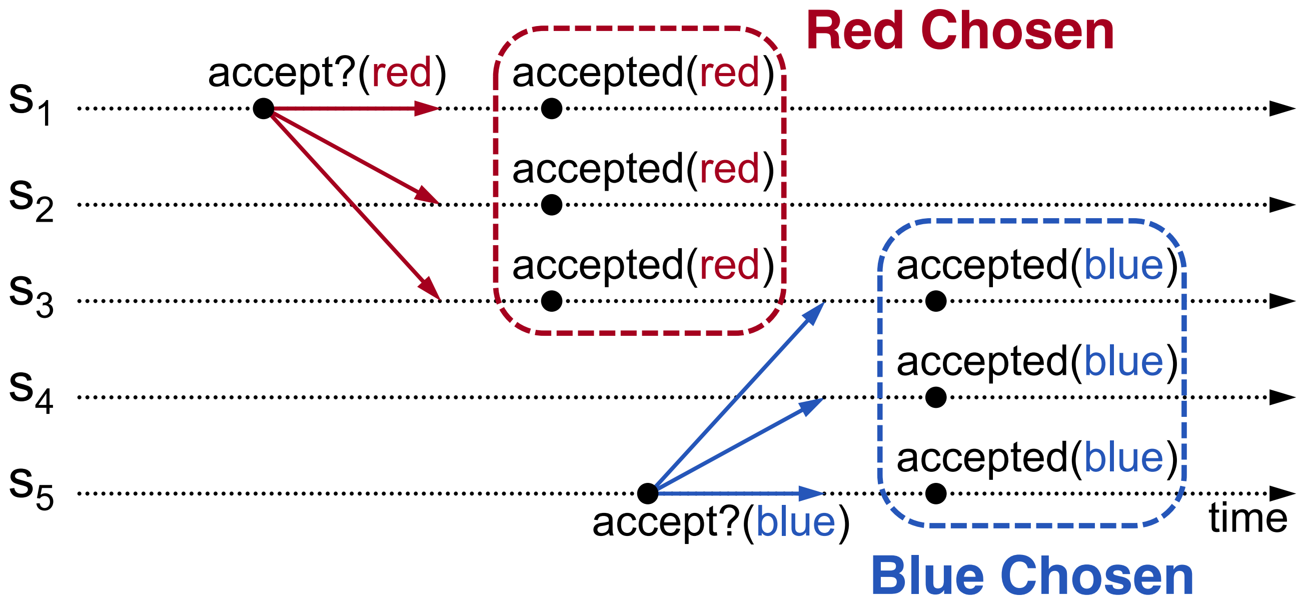
Design Decision

How does an Acceptor accept values (i.e., vote for a Proposer)?

Option#1: Acceptor accepts first value it receives

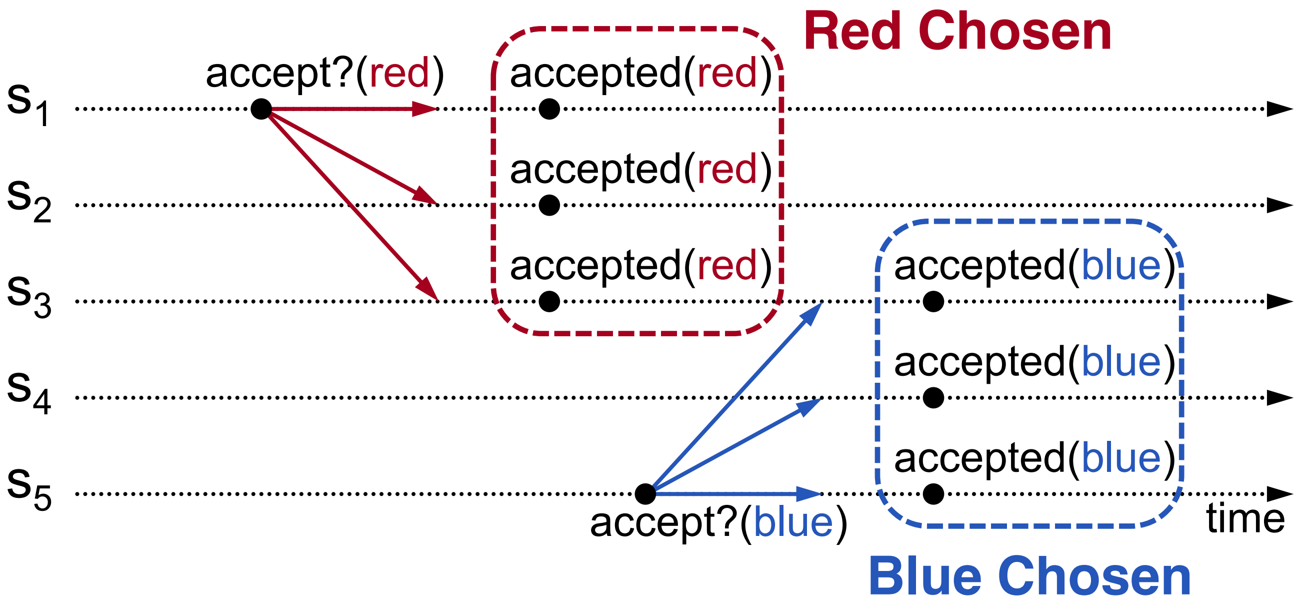


Option#2: Acceptor accepts every value it receives



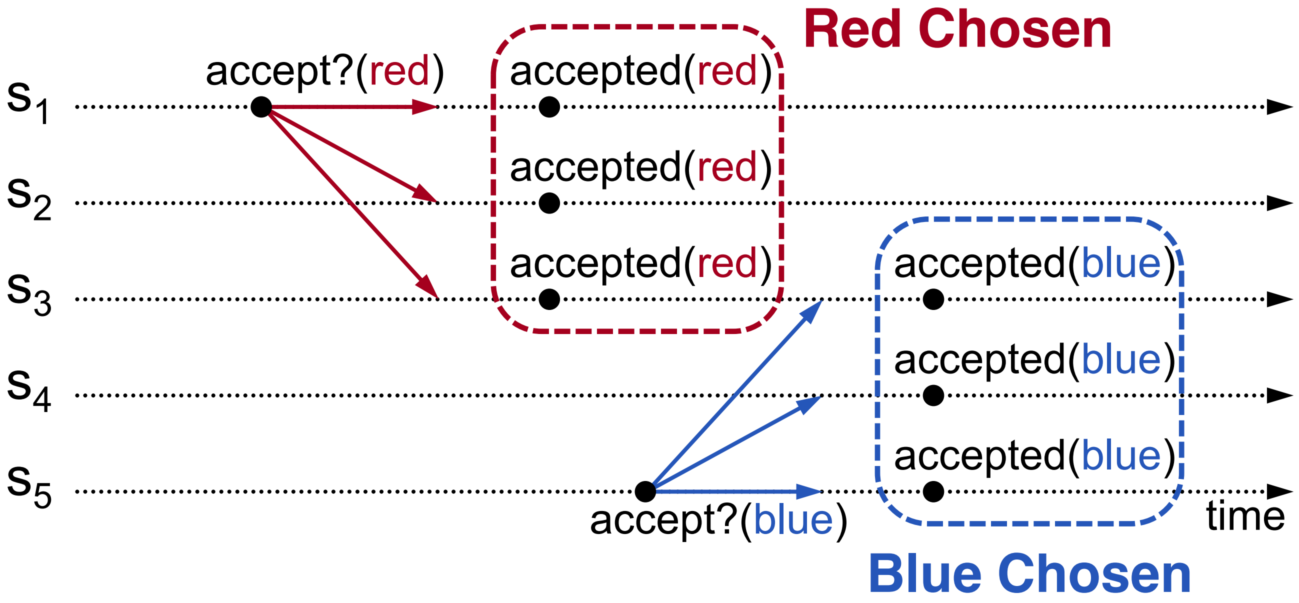
Challenge:
How to check if a value has already been chosen?

Option#1: Wait to hear from some majority of Acceptors who have accepted a value

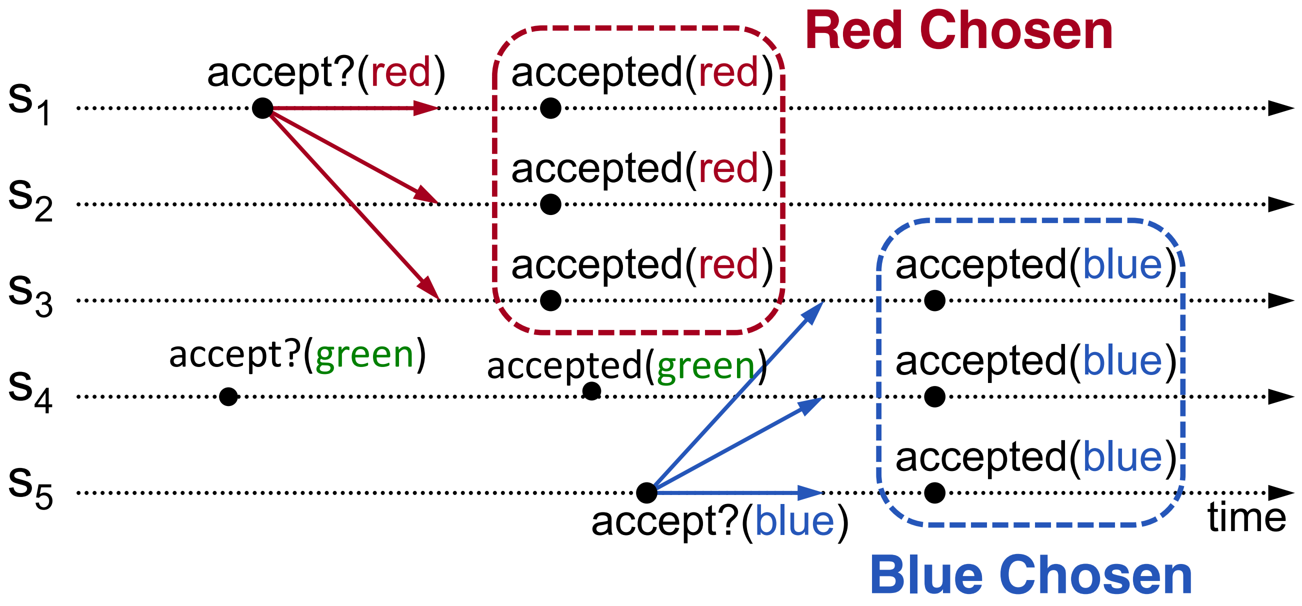


Option#2: Wait to hear from any majority of Acceptors

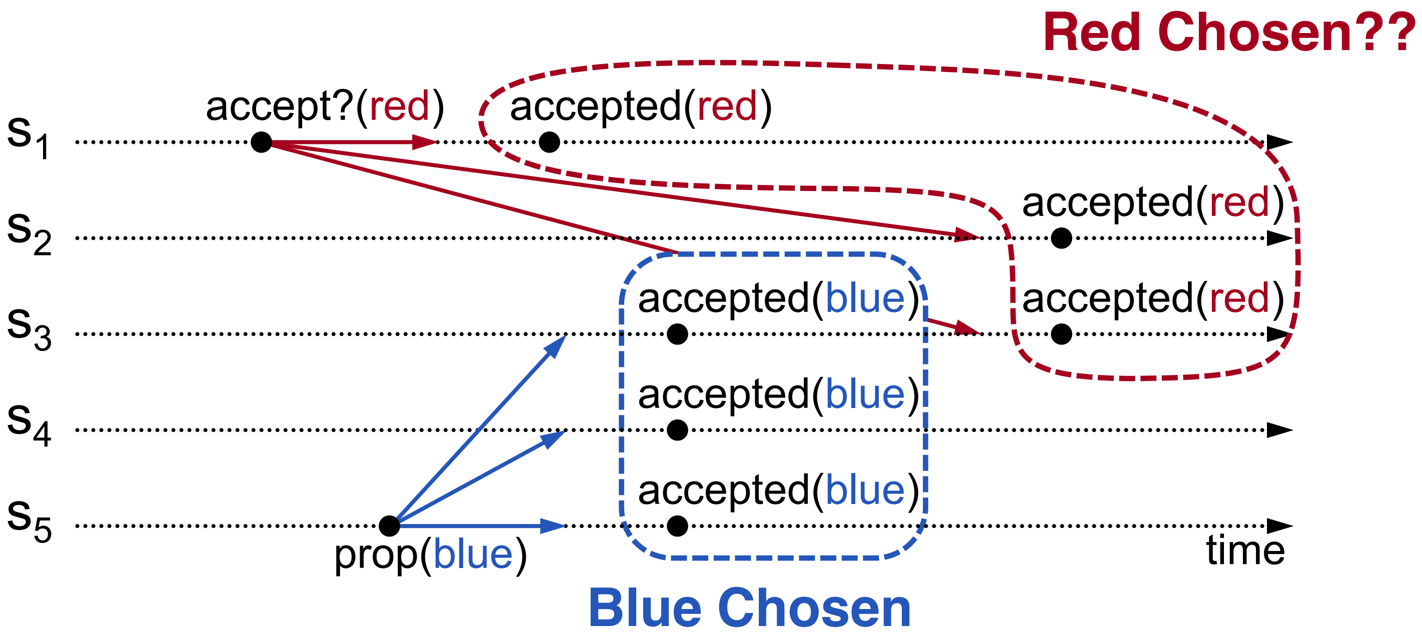
- Sub-option#1: If a majority of acceptors return an accepted value, then do not propose your value, otherwise go ahead with your proposed value
- Sub-option#2: If any of the acceptors returns an accepted value, do not propose your value, otherwise go ahead with your proposed value



Problem: What if the Proposer hears two different accepted values?



Another Problem: What if the second Proposer doesn't hear any accepts about a previous proposal?



Summary: Paxos Key Ideas

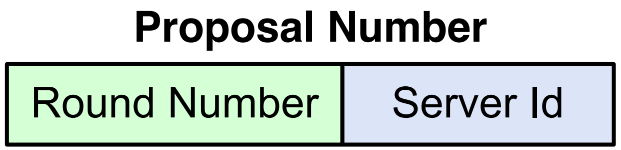
- A value is chosen only if a majority of acceptors accept it

- Use two phases (like in 2-PC)
  - First, check if there is a proposed value that has been chosen
    - Wait for some majority, if any acceptor has accepted a value, assume the value is chosen, and do not propose a different value
  - But two phases may not be enough

- Order proposals and block older proposals
  - To avoid choosing two different values and violating safety

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

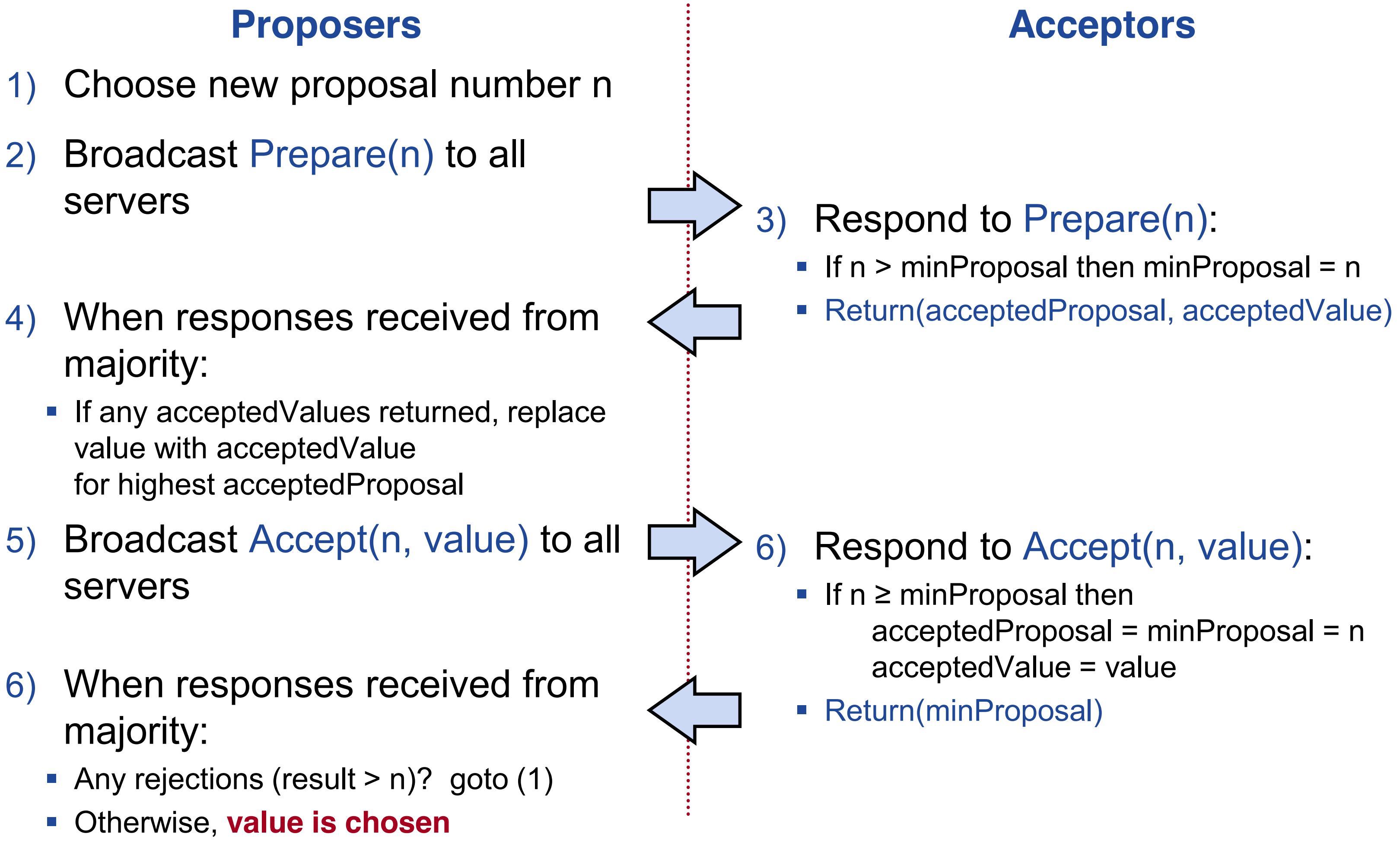
Round Number | Server Id

    - 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; so a proposer does not reuse proposal numbers after crash/restart

Two-Phase Approach

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

Paxos



Proposers

Acceptors

1) Choose new proposal number n

2) Broadcast Prepare(n) to all servers

3) Respond to Prepare(n):
   - If n > minProposal then minProposal = n
   - Return(acceptedProposal, acceptedValue)

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) Respond to Accept(n, value):
   - If n ≥ minProposal then
     acceptedProposal = minProposal = n
     acceptedValue = value
   - Return(minProposal)

6) When responses received from majority:
   - Any rejections (result > n)? goto (1)
   - Otherwise, **value is chosen**

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

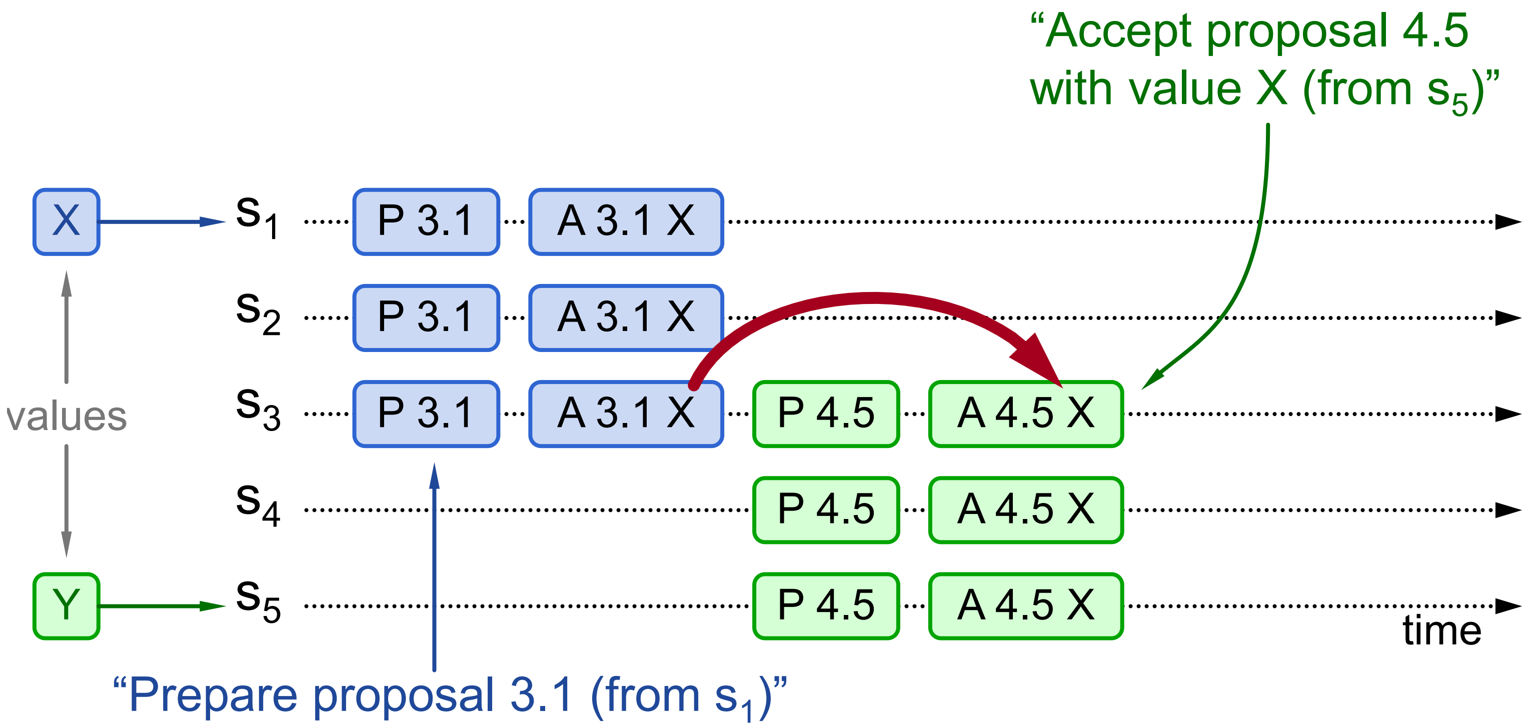
Paxos Examples

- To illustrate how Paxos works
- In particular, we will consider scenarios with two competing proposals
- There are three possible situations in such scenarios

Paxos Examples

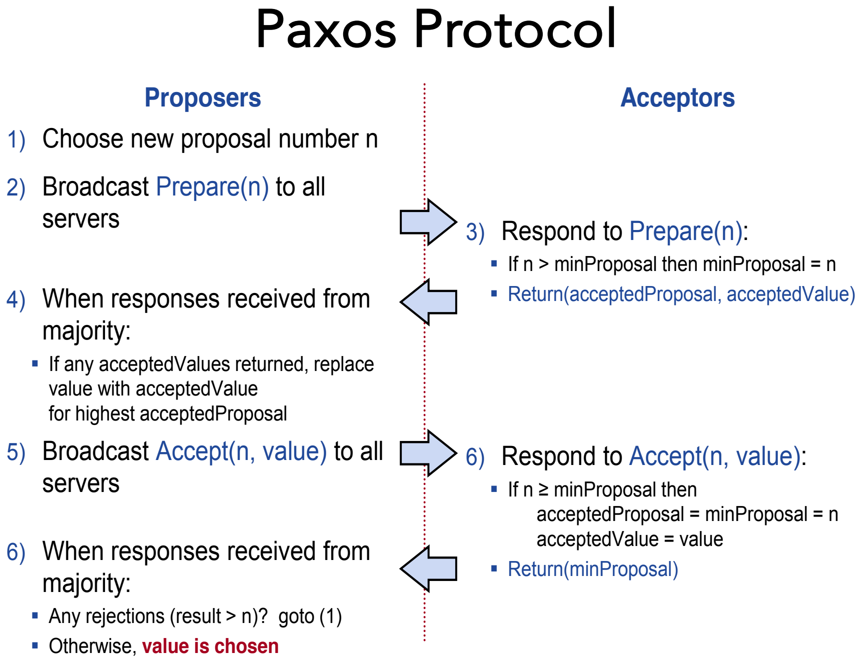
1. Previous value already chosen

- New proposer will find it and use it



"Accept proposal 4.5 with value X (from s_5)"

"Prepare proposal 3.1 (from s_1)"



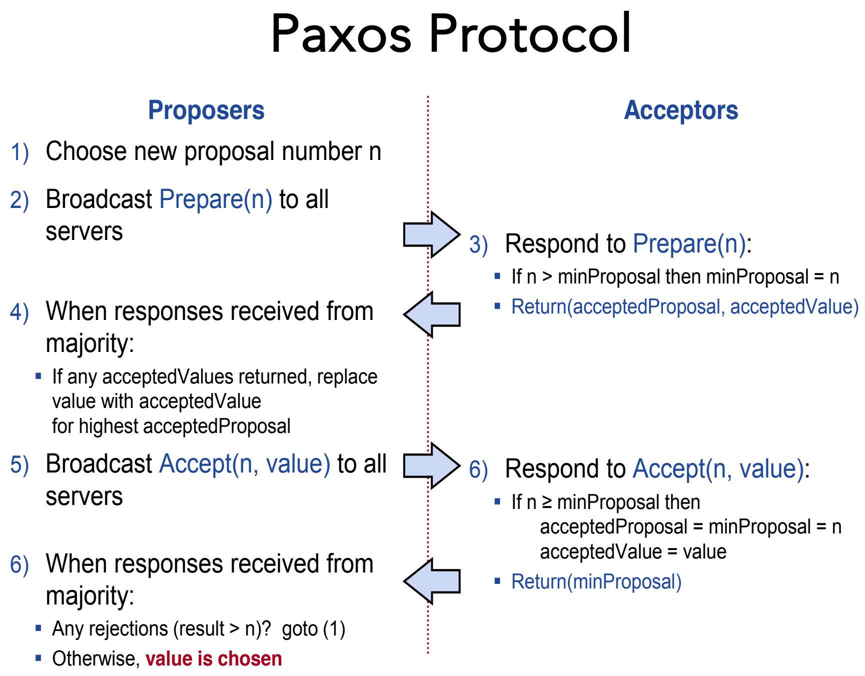
Paxos Protocol

Proposers | **Acceptors**

1) Choose new proposal number n
2) Broadcast Prepare(n) to all servers
3) Respond to Prepare(n):
   - If n > minProposal then minProposal = n
   - Return(acceptedProposal, acceptedValue)
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) Respond to Accept(n, value):
   - If n ≥ minProposal then acceptedProposal = minProposal = n acceptedValue = value
   - Return(minProposal)
6) When responses received from majority:
   - Any rejections (result > n)? goto (1)
   - Otherwise, **value is chosen**

Paxos Examples

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



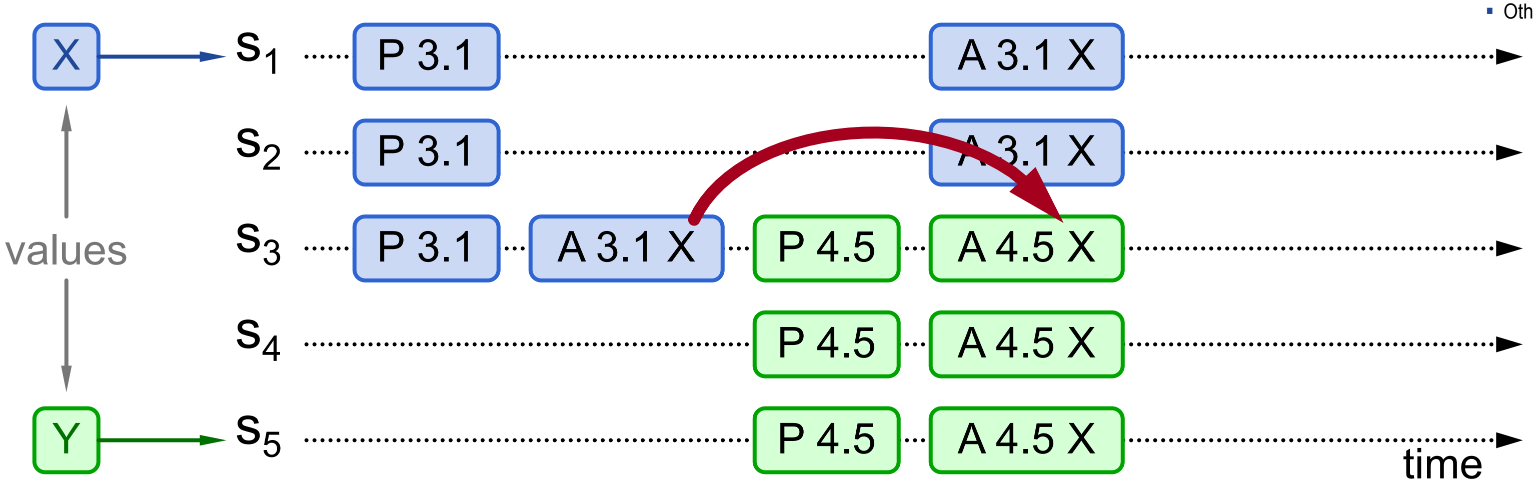
Paxos Protocol

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

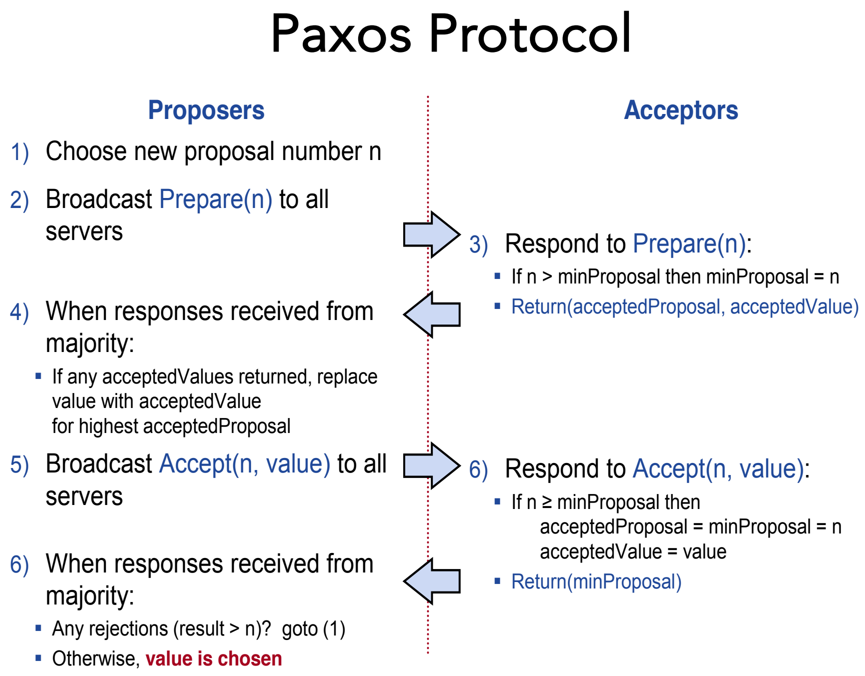
3) Respond to Prepare(n):
   - If n > minProposal then minProposal = n
   - Return(acceptedProposal, acceptedValue)
6) Respond to Accept(n, value):
   - If n ≥ minProposal then acceptedProposal = minProposal = n acceptedValue = value
   - Return(minProposal)



Paxos Examples

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

- New proposer chooses its own value
- Older proposal blocked



Paxos Protocol

Proposers

1) Choose new proposal number n
2) Broadcast Prepare(n) to all servers

Acceptors

3) Respond to Prepare(n):
   - If n > minProposal then minProposal = n
   - Return(acceptedProposal, acceptedValue)

Proposers

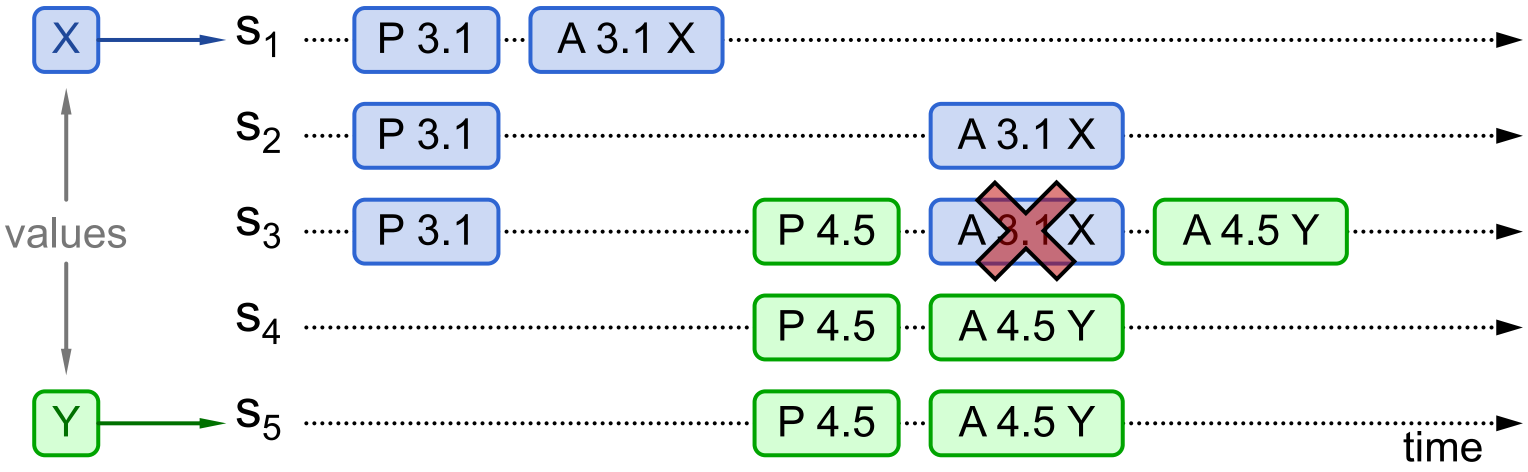
4) When responses received from majority:
   - If any acceptedValues returned, replace value with acceptedValue for highest acceptedProposal
5) Broadcast Accept(n, value) to all servers

Acceptors

6) Respond to Accept(n, value):
   - If n ≥ minProposal then acceptedProposal = minProposal = n acceptedValue = value
   - Return(minProposal)

Proposers

6) When responses received from majority:
   - Any rejections (result > n)? goto (1)
   - Otherwise, **value is chosen**



Other points to note

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

Paxos

- Safety?

- Liveness?

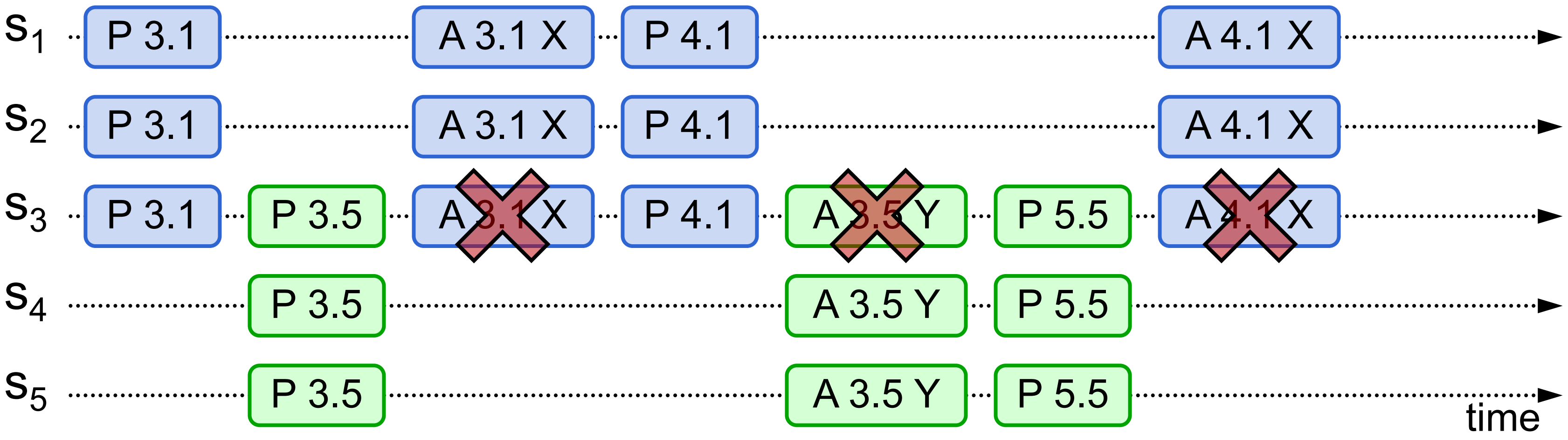
- Performance?

Safety

- Intuition: if a proposal with value v is chosen, then every higher-numbered proposal issued by any proposer has value v

Liveness

- Competing proposers can livelock:



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

Performance?

Paxos Fault tolerance

- If there can be f fail-stop failures in a system
- What are the minimum number of nodes Paxos needs to ensure consensus is reached?

Paxos: Summary

- Safety: Never violated
- On Liveness
  - If things go well sometime in the future (messages and failures, etc.), there is a good chance consensus will be reached.
- FLP result still applies:
  - Paxos is not guaranteed to reach a consensus (ever or within a bounded time)

Paxos Problems

- Basic Paxos solves the problem for a single value
    - However, non-trivial to extend to Multi-Paxos. No agreement on the details of Multi-Paxos
    - We will discuss Multi-Paxos after we complete our discussion of Raft
- Doesn’t fully address liveness
- Does not discuss cluster membership management

Led John Ousterhout and his PhD student Diego Ongara, to design a new consensus algorithm with **understandability as a primary design goal**

Next Lecture

- Raft: In Search of an Understandable Consensus Algorithm