

Paxos

Jinyang Li

Some slides are adapted from Ousterhout and Ongaro

What we've learnt last time

- Strong consistency model: linearizability
 - execution is equivalent to a serial history that preserves global completion-to-issue order
- How to implement linearizability
 - no replication
 - viewstamp replication

Consensus \leftrightarrow consistent replication

- Consensus allows a set of nodes to agree on something
- Consensus \leftrightarrow Replica servers agree on the same sequence of operations for execution
 \leftrightarrow Linearizable system

Paxos solves the consensus problem

The screenshot shows the ACM Turing Award website. At the top left is the ACM logo with the text "MORE ACM AWARDS". To its right is a large graphic for the "A.M. TURING AWARD" featuring a portrait of Alan Turing. Further right is a search bar with a home icon and the text "Search TYPE HERE". Below these are three rows of small portrait icons of past award winners. A navigation bar below the portraits has three tabs: "ALPHABETICAL LISTING", "YEAR OF THE AWARD", and "RESEARCH SUBJECT". The "ALPHABETICAL LISTING" tab is selected. Below the navigation bar, a large portrait of Leslie Lamport is shown on the left. To the right of the portrait, the name "LESLIE LAMPORT" is displayed in large blue letters, followed by a green "DL" icon. Below the name, it says "United States – 2013". Underneath that, the word "CITATION" is written in orange, followed by a paragraph of text: "For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency." At the bottom of the page, there are four small blue icons representing social media links.

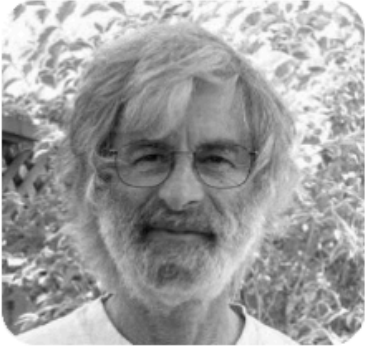
acm
MORE ACM AWARDS

A.M.
TURING
AWARD

Search TYPE HERE

A.M. TURING AWARD WINNERS BY...

ALPHABETICAL LISTING YEAR OF THE AWARD RESEARCH SUBJECT



LESLIE LAMPORT DL

United States – 2013

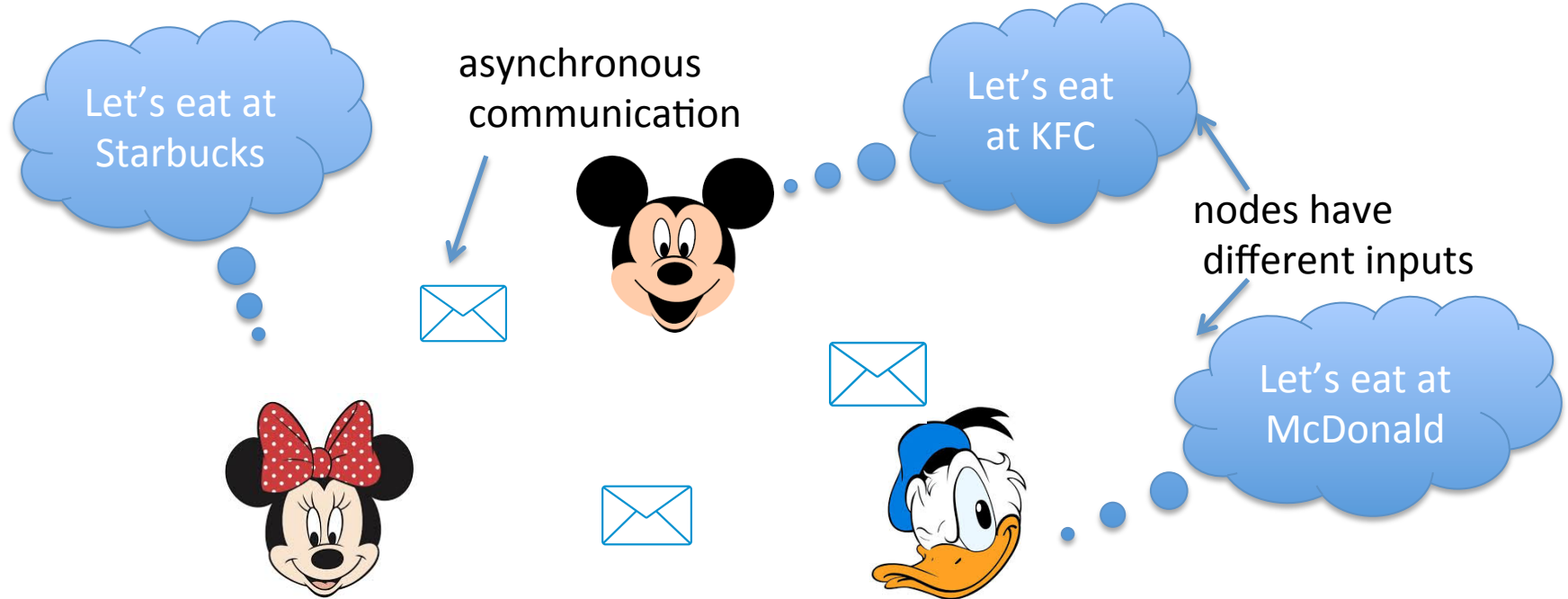
CITATION

For fundamental contributions to the theory and practice of distributed and concurrent systems, notably the invention of concepts such as causality and logical clocks, safety and liveness, replicated state machines, and sequential consistency.

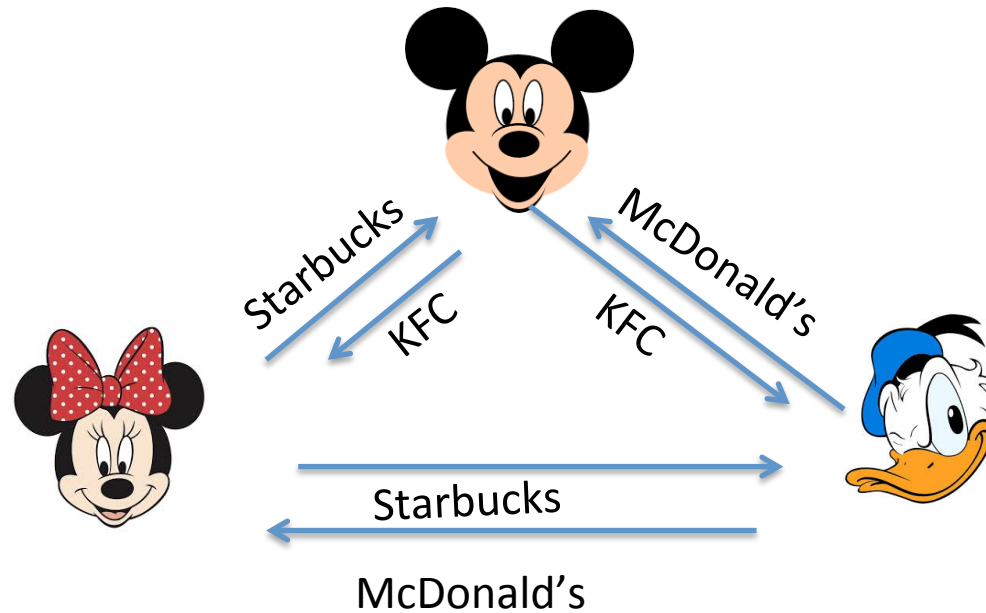
Single-decree consensus

- Problem Setting
 - n nodes, each with a (potentially different) input
 - one or more nodes may fail
 - network is asynchronous (cannot tell node crash from slow communication)
- Goal
 - Safety (chosen values by different nodes are the same)
 - Liveness (all live nodes eventually choose some value)

Single-decree consensus

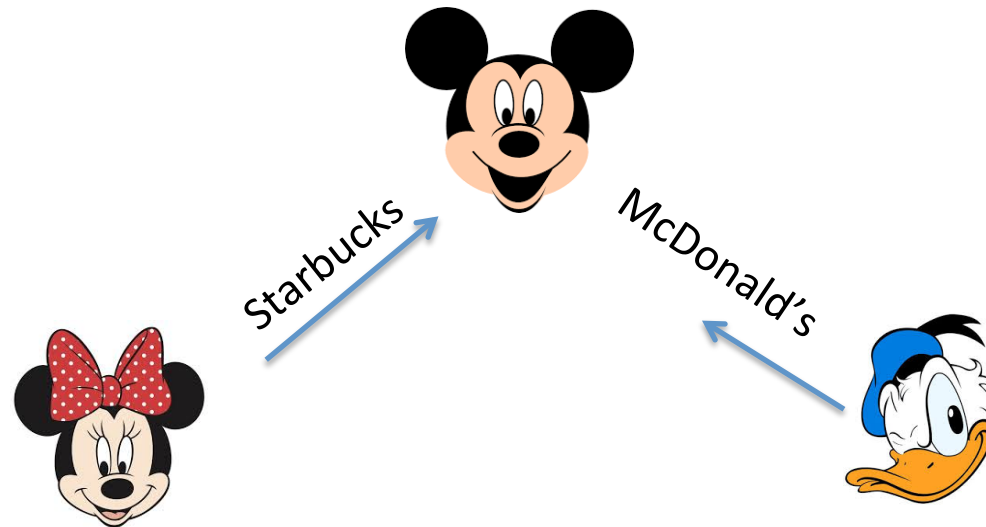


Strawman #1



- Every node sends its value to a designated node
- Every node chooses the first value seen

Strawman #2



- Every node sends its value to a designated node (acceptor)
- Acceptor chooses the first value

The FLP [1985] impossibility result

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of California, Berkeley

ABSTRACT
Under
unreli-
the
pro-
pro-

Ca-
pro-
ap-

ity, Availability, and Serviceability; F.1.2 [Computation by Abstract Devices]: Modes of Computation-parallelism; H.2.4 [Database Management]: Systems-distributed systems, transaction processing

General Terms: Algorithms, Reliability, Theory

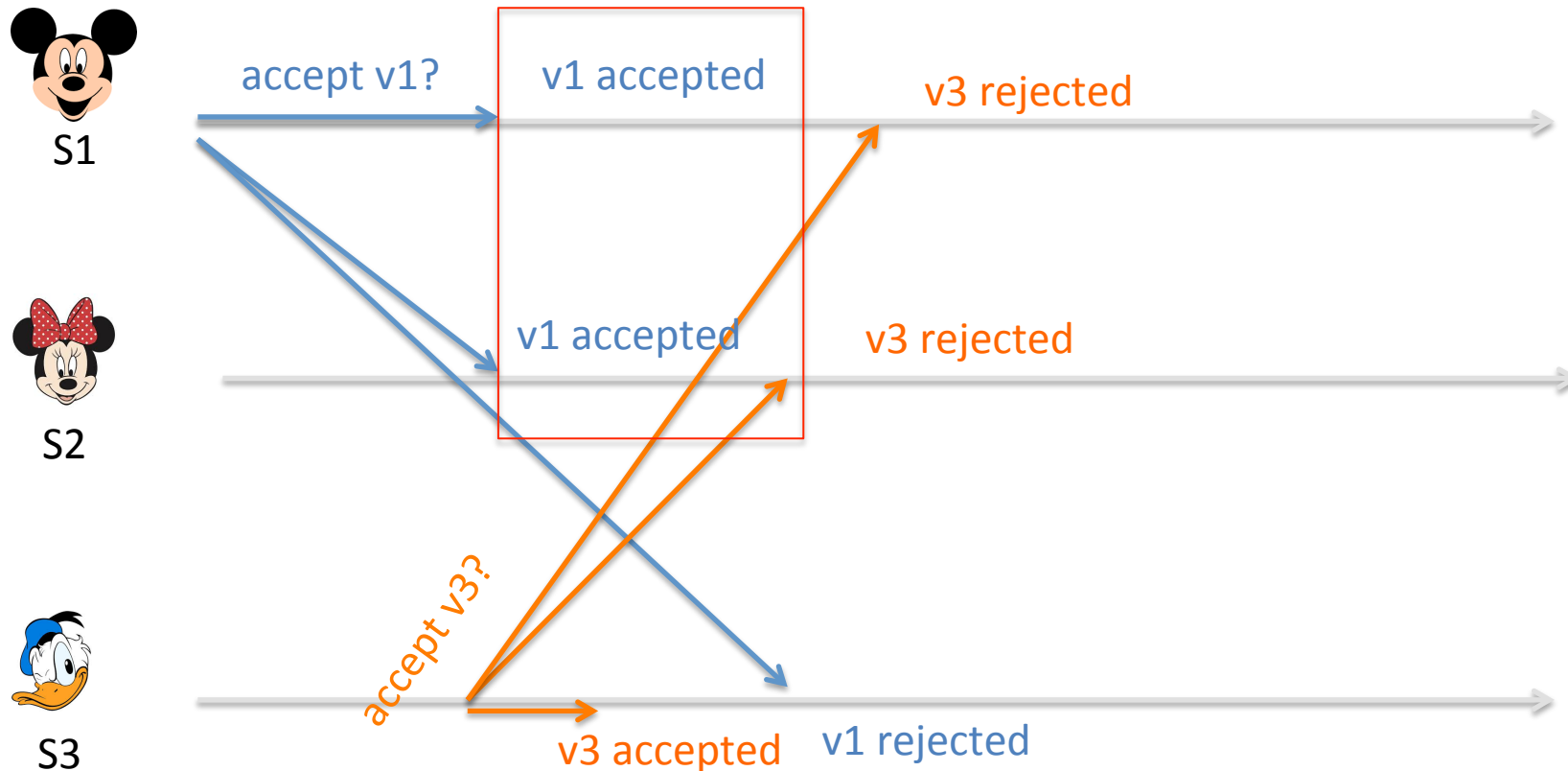
Additional Key Words and Phrases: Agreement problem, asynchronous system, Byzantine Generals problem, commit problem, consensus problem, distributed computing, fault tolerance, impossibility proof, reliability

No *deterministic* server logic
can solve consensus
in the face of a single server failure!

Paxos components

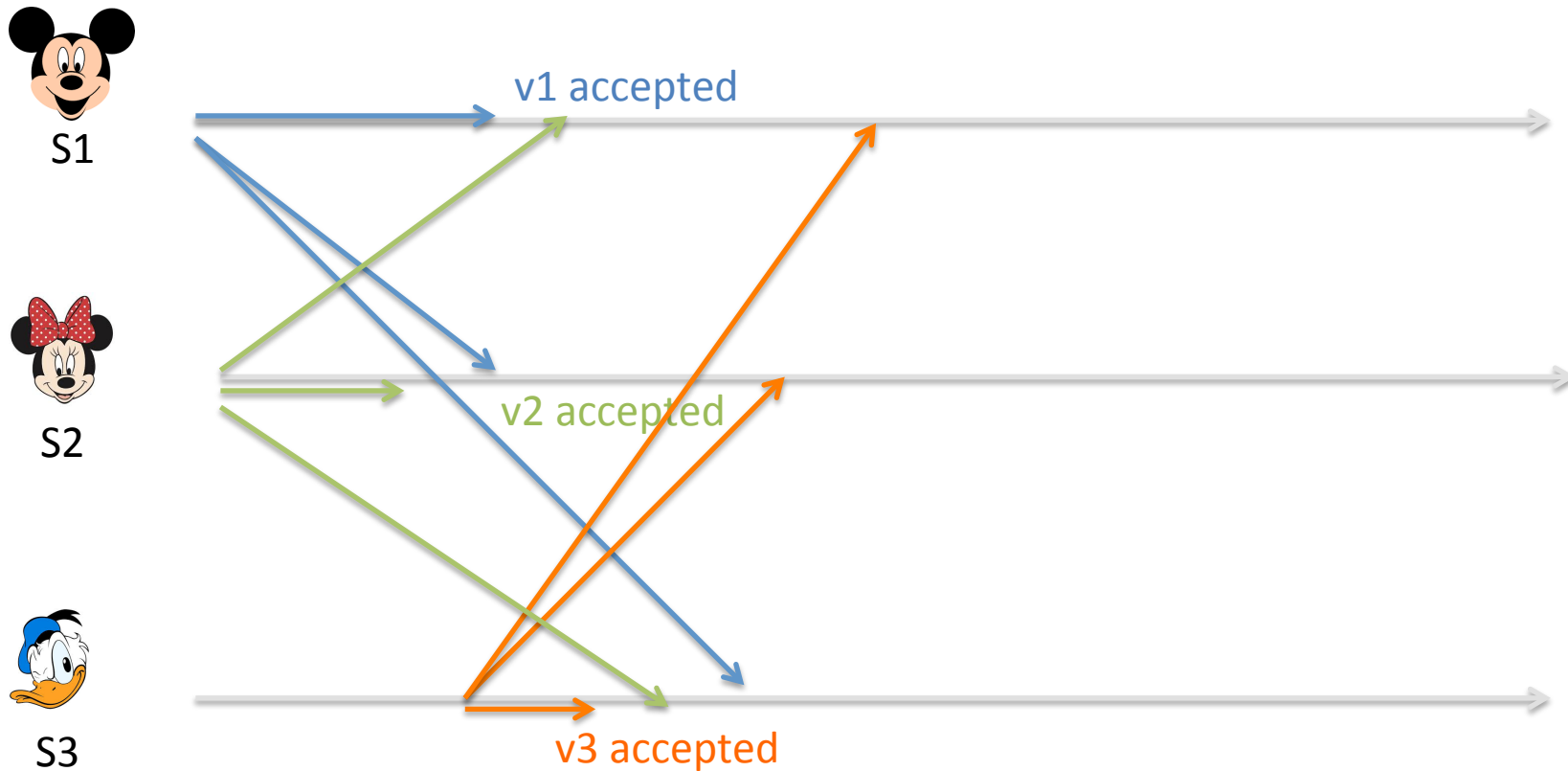
- Proposers
 - active role (send RPCs)
- Acceptors
 - passive role (responding to RPCs)
- Learners
 - passive role (learn about outcome of consensus)
- To simplify, we assume each server takes both the active and passive role

Idea #1: choose a value if a quorum of nodes accept



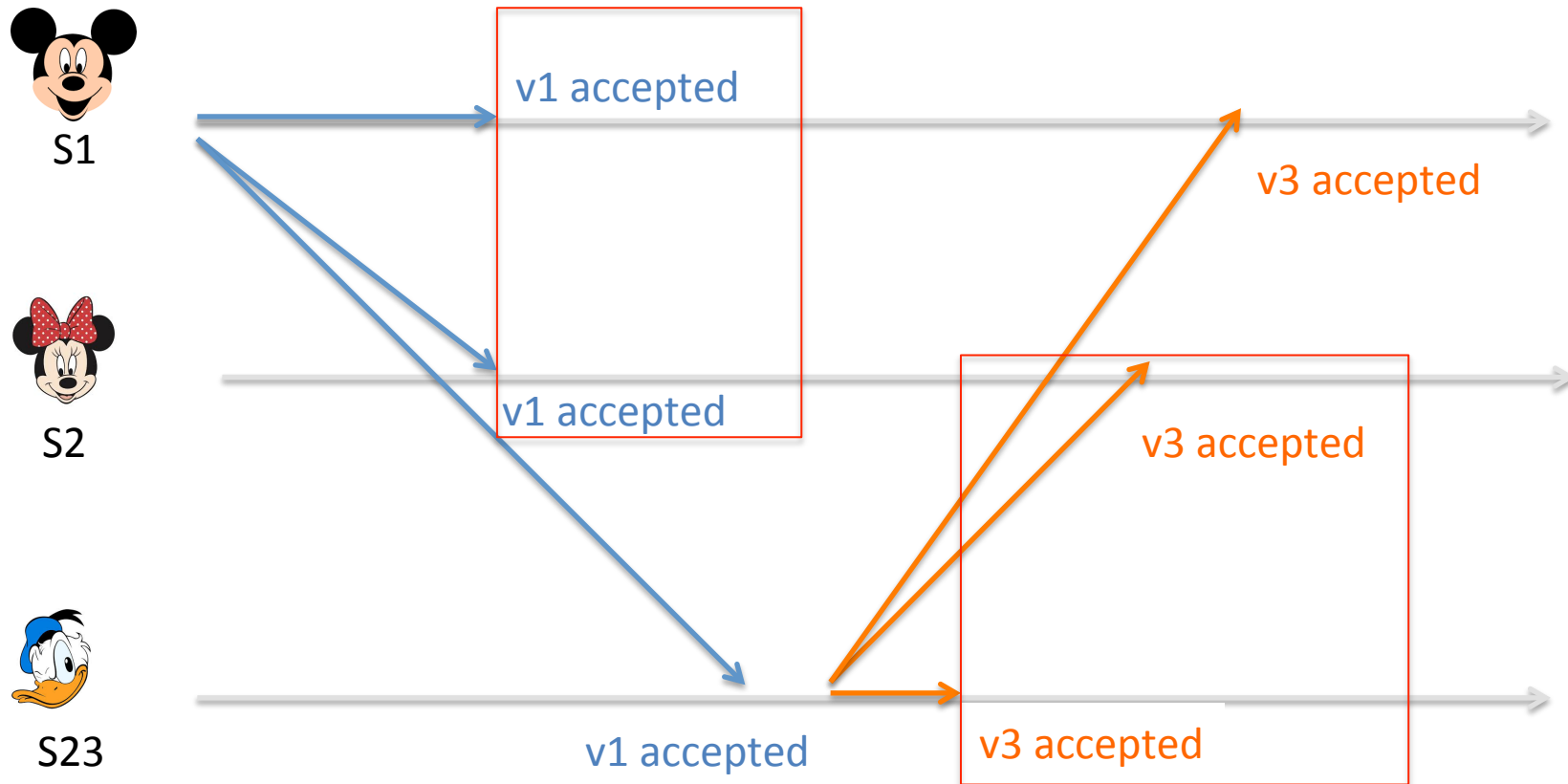
- A node sends out a proposal p to every node
- Each node accepts the first proposal seen
- If a proposal is accepted by majority of nodes, its value is chosen

Problem: split votes



Solution? Try again! → Each node has to accept more than one proposal!

Danger of accepting more than once

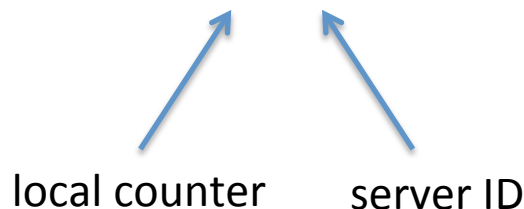


- To ensure correctness:
 - If a proposal p is chosen, then all subsequent accepted proposals must have the same value as p

Idea #2: Totally ordered proposals

- Each proposal is identified by a globally unique proposal number
- Proposals can be totally ordered
- Paper's suggestion:
 - proposal number = local counter + server-id

1.S1 < 1.S2 < 1.S3 < 2.S1 < 2.S2 < 2.S3 < ...



Idea #2: Totally ordered proposals

- Paxos Invariant (P2b in paper)
 - If proposal p is chosen, then for all $p' > p$, $p'.value = p.value$
- → a proposal cannot blindly use the server's input value, it needs to use a **safe** value.
- How to discover a safe value?
 - Use an extra round of communication to find out safe value

Basic Paxos

- Two phase protocol
- Phase-1 (Prepare)
 - A server picks a proposal number and tries to discover a safe value for it
- Phase-2 (Accept)
 - Send the proposal to all for acceptance

Server state

- highestNum
- acceptedNum
- acceptedVal

Prepare-phase

- Choose new proposal number n
 - $n = \{\text{highestNum.counter} + 1, \text{server-id}\}$
- Send **Prepare(n)** to all servers
- If receiving majority OK replies
 - val = safe value found in majority OK replies
- Else
 - retry from beginning

RPC handler for Prepare(n)

- If $n > \text{highestNum}$
 - $\text{highestNum} = n$
 - return {OK, acceptedNum, acceptedVal}
- Else
 - return {NotOK}

Accept-phase

- Send **Accept(n, val)** to all servers
- If receiving majority AcceptOK replies
 - val is chosen
- Else
 - Retry from beginning

RPC handler for Accept(n, val)

- If $n \geq \text{highestNum}$
 - $\text{highestNum} = n$
 - $\text{acceptedNum} = n, \text{acceptedVal} = \text{val}$
 - return {OK}
- Else
 - return {NotOK}

Why majority quorum?

- Why requiring a majority AcceptOK?
 - No proposals can be chosen without intersecting at a common node
- Why requiring a majority PrepareOK?
 - If a proposal has been chosen (accepted by a majority), it'll be among a majority of PrepareOK relies.

How to find a safe value v

- Each prepareOK returns the highest proposal less than n that has been (or will be) accepted
- Q_{prepOK} = set of PrepareOKs from a majority
- If none in Q_{prepOK} has an accepted proposal
 - $v =$ proposer's own input
- Else
 - $v =$ value of the highest proposal in Q_{prepOK}

How to find a safe value v

Proof of Paxos invariant: if proposal m with v is chosen, then any proposal $n > m$ has value v .

Proof: By induction. Let's suppose invariant holds for $n-1$.

Let $p =$ highest proposal in Q_{prepOK} , then $m \leq p < n$.
Since all proposals $[m, \dots, n-1]$ have value v (by induction), the highest proposal in Q_{prepOK} have v .

Because p 's prepare-quorum intersects with m 's accept-quorum

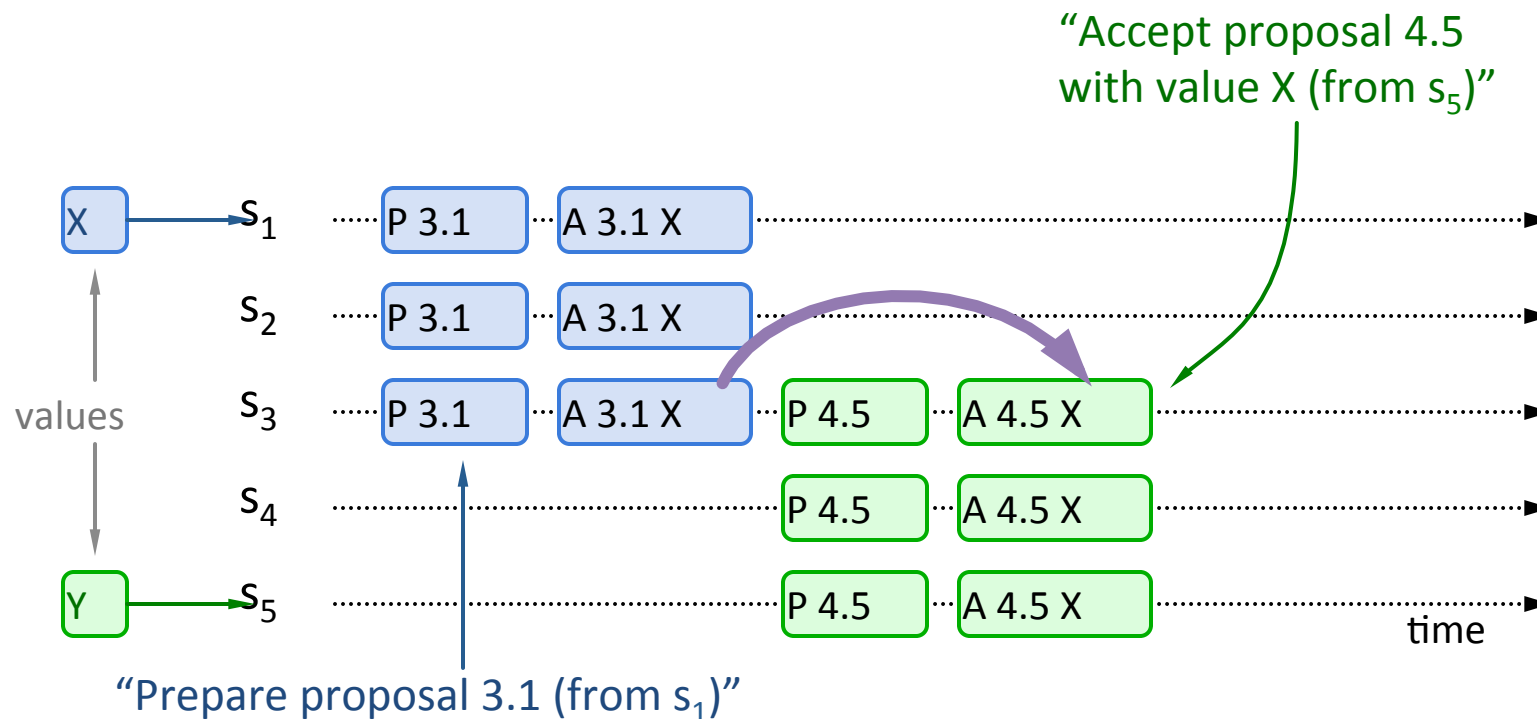
Because acceptor rejects n if it has seen a higher proposal

Basic Paxos Examples

Three possibilities when later proposal prepares:

1. Previous value already chosen:

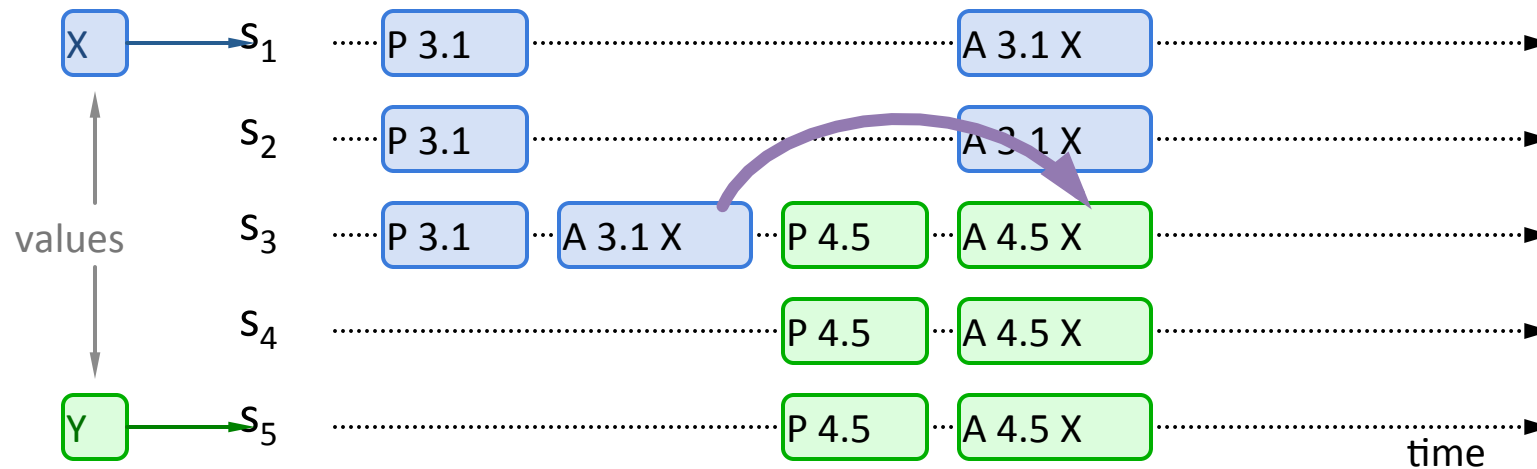
- New proposer will find it and use it



Basic Paxos Examples, cont'd

Three possibilities when later proposal prepares:

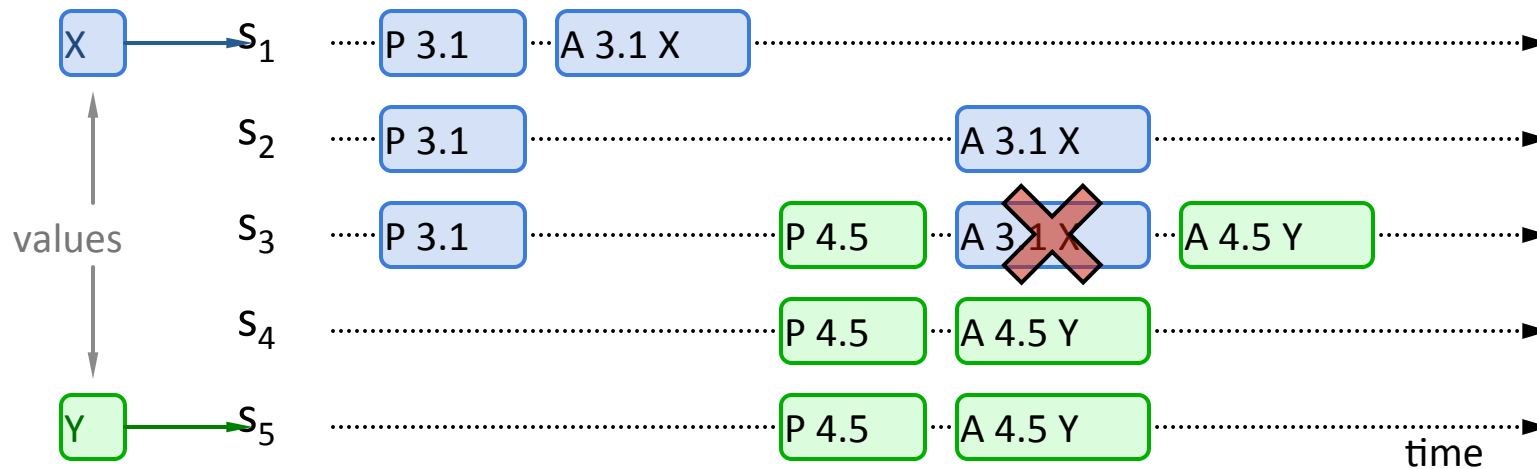
2. Previous value not chosen, but new proposer sees it:
 - New proposer will use existing value
 - Both proposers can succeed



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

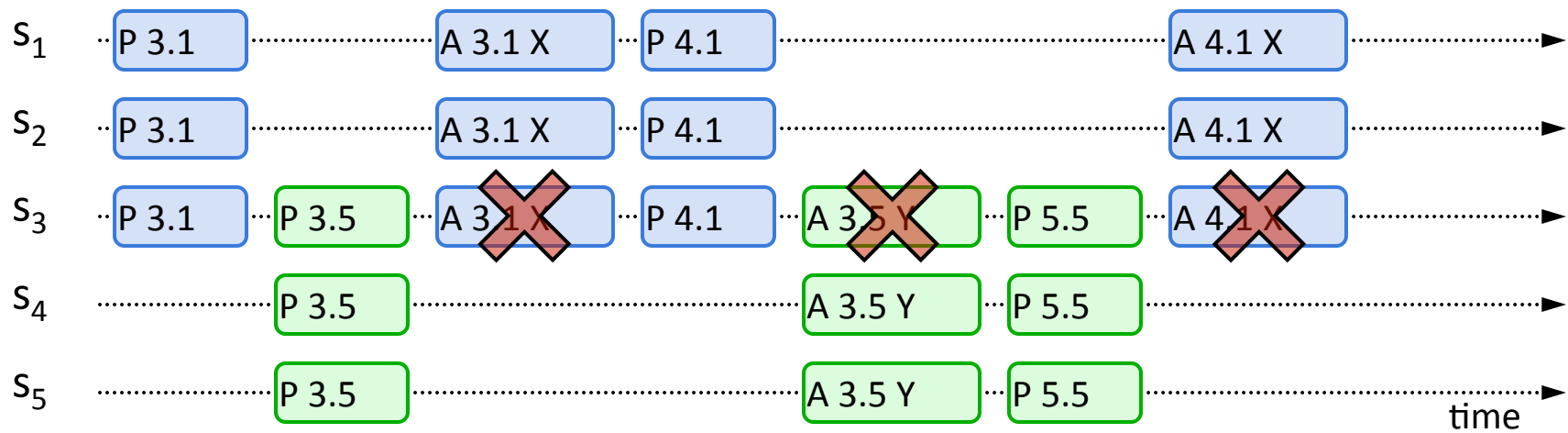


How does Paxos “get around” FLP impossibility?

- Paxos is not a deterministic algorithm
- Proposer retries with a randomized delay
- No guarantee that consensus is reached in a fixed amount of time (with high probability)

Liveness

- Competing proposers can livelock:



How can a crashed node rejoin safely?

- All Paxos state must be persisted durably
 - highest proposal number seen
 - highest accepted proposal (number and value)

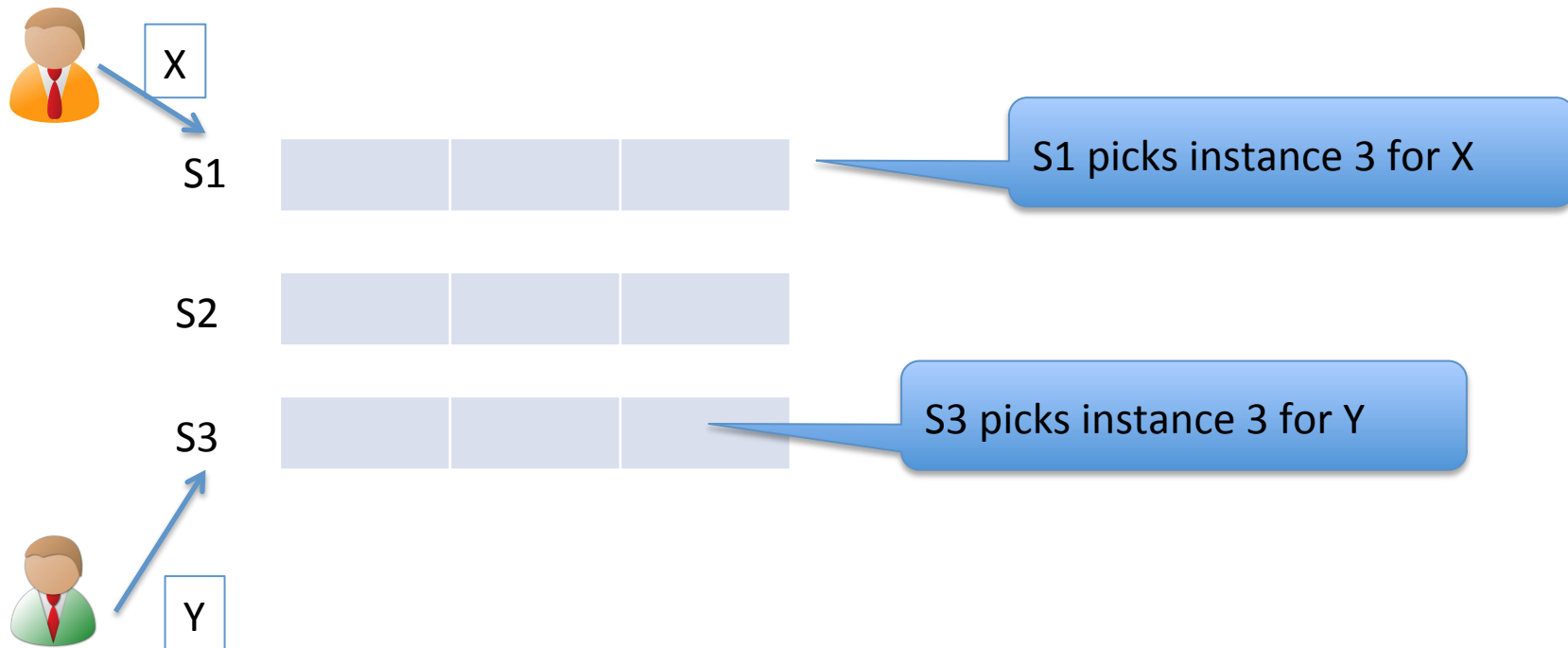
Multi-Paxos: agreeing on a
sequence of values

Multi-Paxos builds on top of basic Paxos

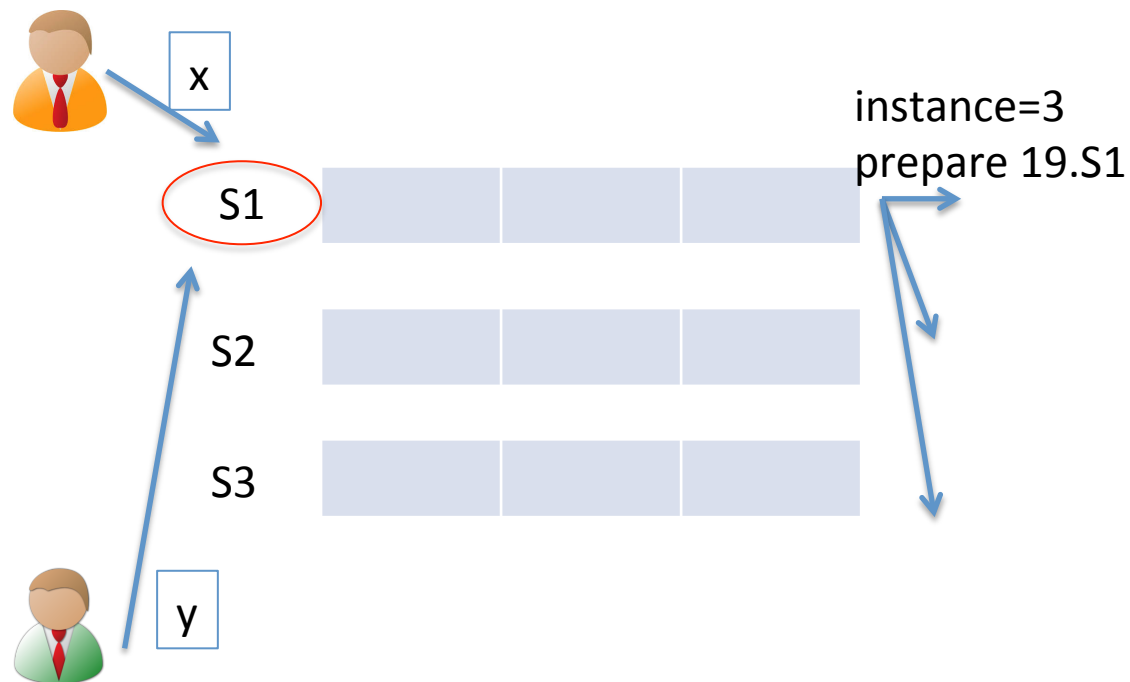
- How to implement Replicated State Machine
 - replicating a log of operations consistently
- The naive approach:
 - Run a separate instance of Paxos to agree on the value for each log index
 - Each instance of Paxos has its own copy of state
 - highest proposal seen
 - accepted proposal number
 - accepted proposal value

Naive MultiPaxos

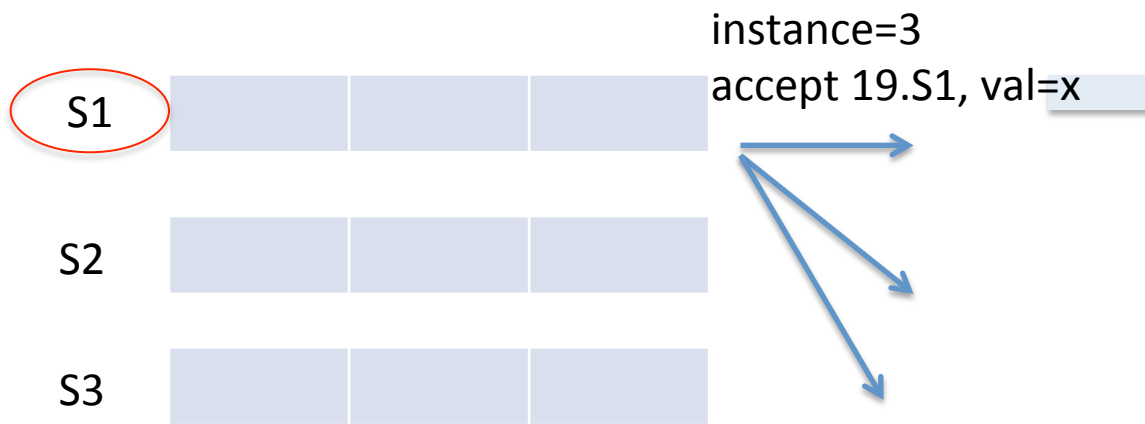
- Server can only execute i -th op if:
 - i -th Paxos instance has chosen a value
 - $i-1$ -th op has been executed



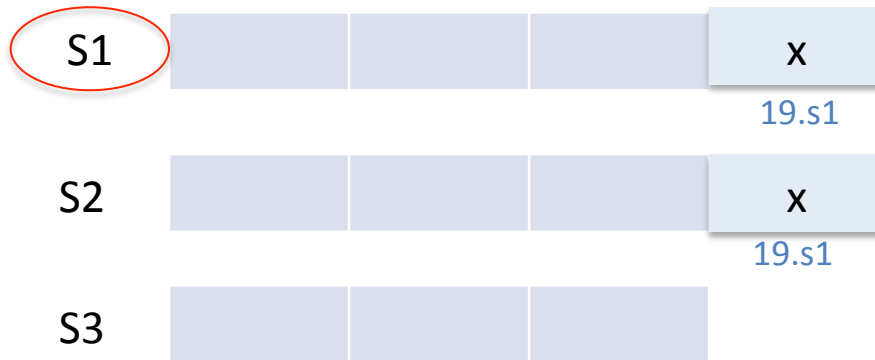
MultiPaxos uses a distinguished proposer (aka leader)



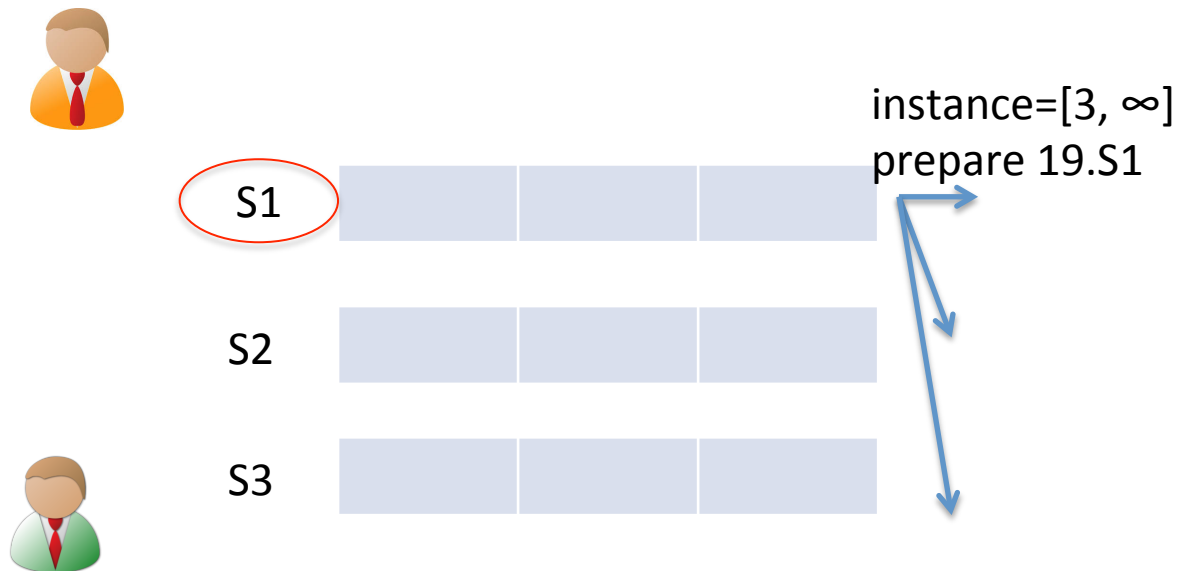
MultiPaxos uses a distinguished proposer (aka leader)



MultiPaxos uses a distinguished proposer (aka leader)

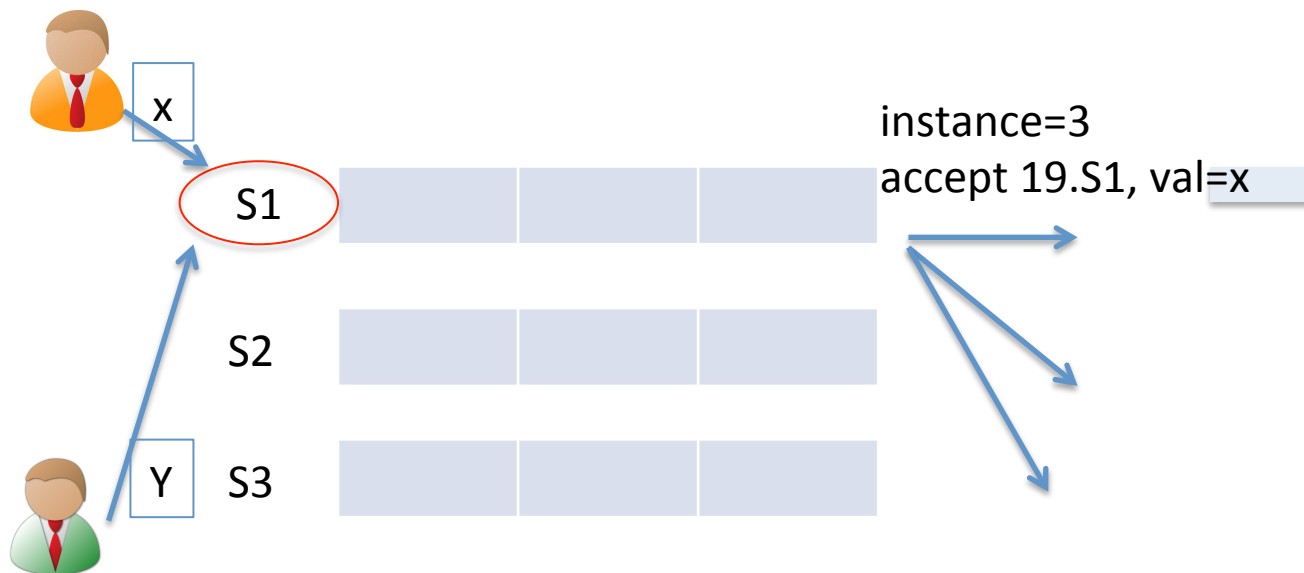


Leader batches Prepare-phase for many instances



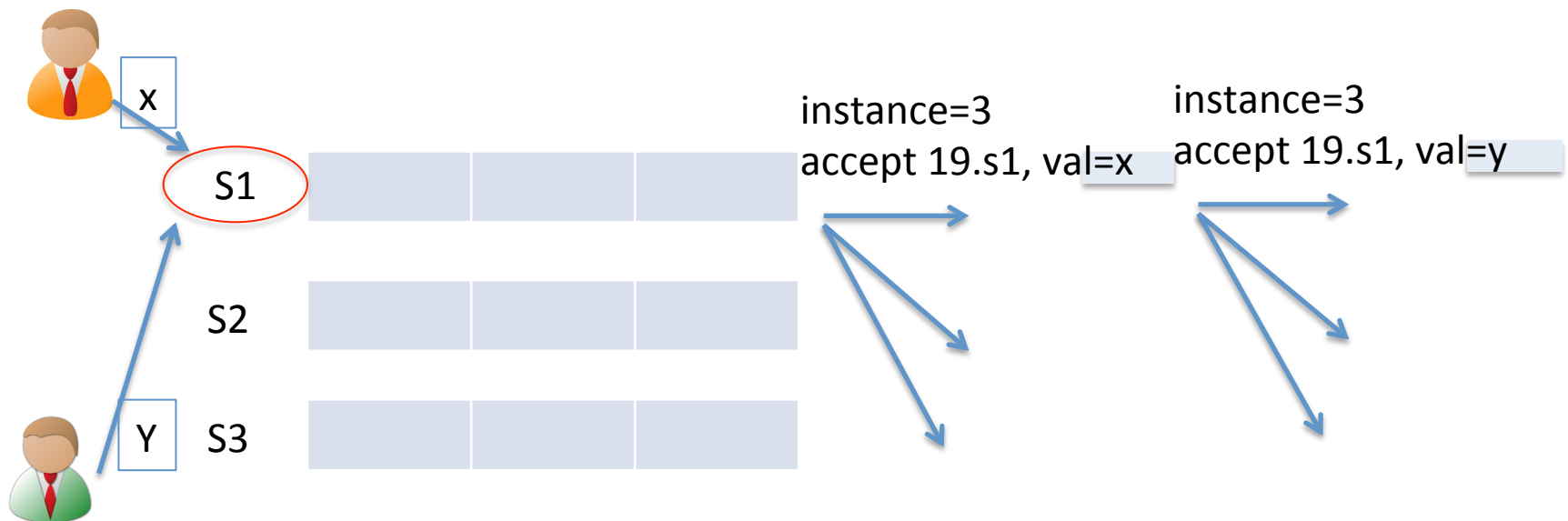
- All instances can share the state: highest proposal number seen
- → Batch prepare needs only one proposal number

Leader batches Prepare-phase for many instances



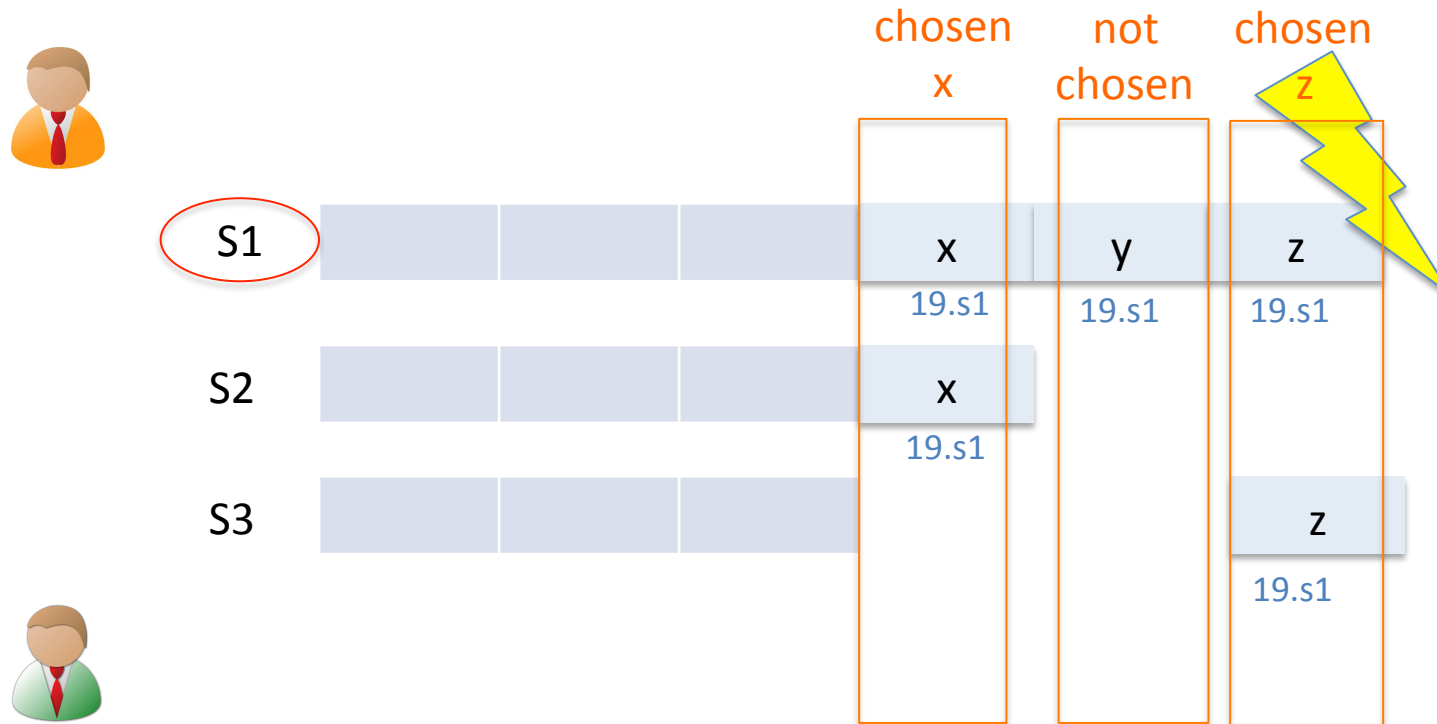
- All instances can share the state: highest proposal number seen
- → Batch prepare needs only one proposal number

Leader batches Prepare-phase for many instances



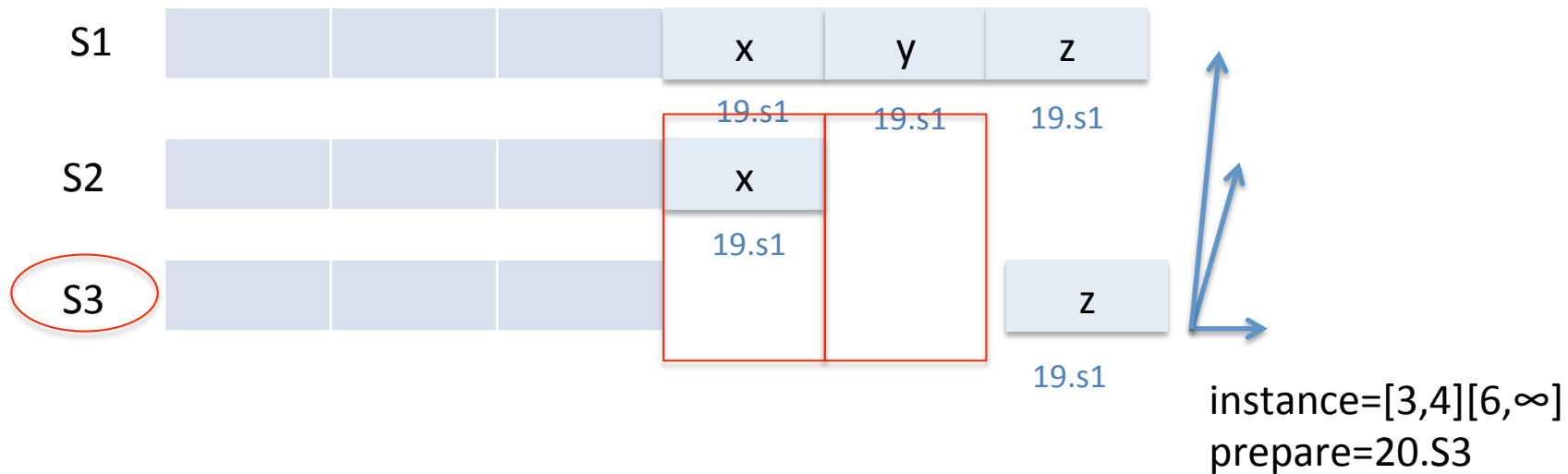
- Efficient: leader only needs to run the accept-phase to replicate an operation during normal time

MultiPaxos runs multiple instances concurrently



- Can run accept-phase of many instances concurrently
- → the prefix of chosen values are not contiguous

MultiPaxos leader switch-over



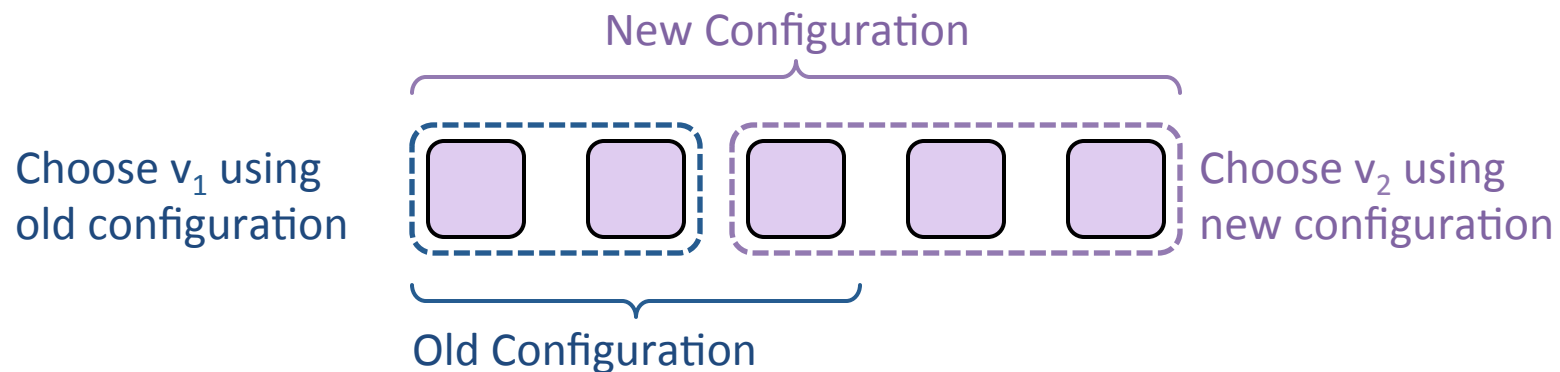
- New leader will try to commit all started (but non-chosen) instance
- Pick value = no-op if there are no client commands to fill a particular started-but-non-committed instance

Configuration change

- So far, we have assumed a static configuration
 - Set of nodes participating is fixed
- Configuration change is needed:
 - Remove failed machine
 - Add a fresh new machine

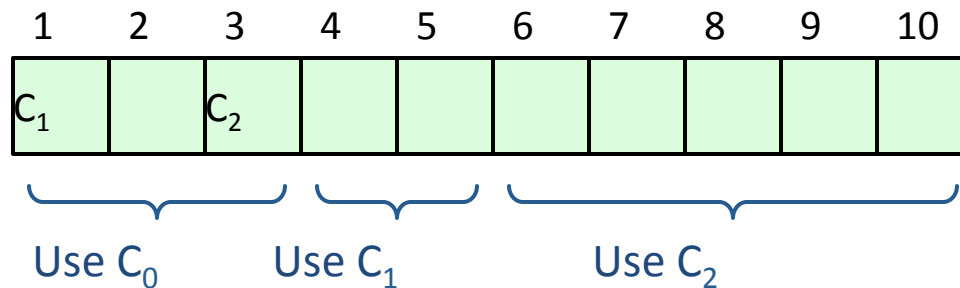
Challenge in configuration change

- Danger: majority quorum of old configuration does not intersect with majority quorum of new configuration



Configuration Changes, cont'd

- MultiPaxos solution:
 - Use the RSM log to manage configuration changes:
 - Configuration is stored as a 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