The Paxos Protocol

Dr. TLA+ Series

Learning Objectives

1. What is the Paxos Protocol, and what problem does it solve?
2. How does the Paxos Protocol work, and why does it work that way?
3. What can the Paxos TLA+ spec teach us about writing specifications?

A problem!

- Your bank has your account balance stored on a computer
- Don’t want to lose account balance if computer crashes/is hit by meteorite
- Solution: bank replicates the account balance to multiple computers!

How can the bank maintain consistency among the replicas?

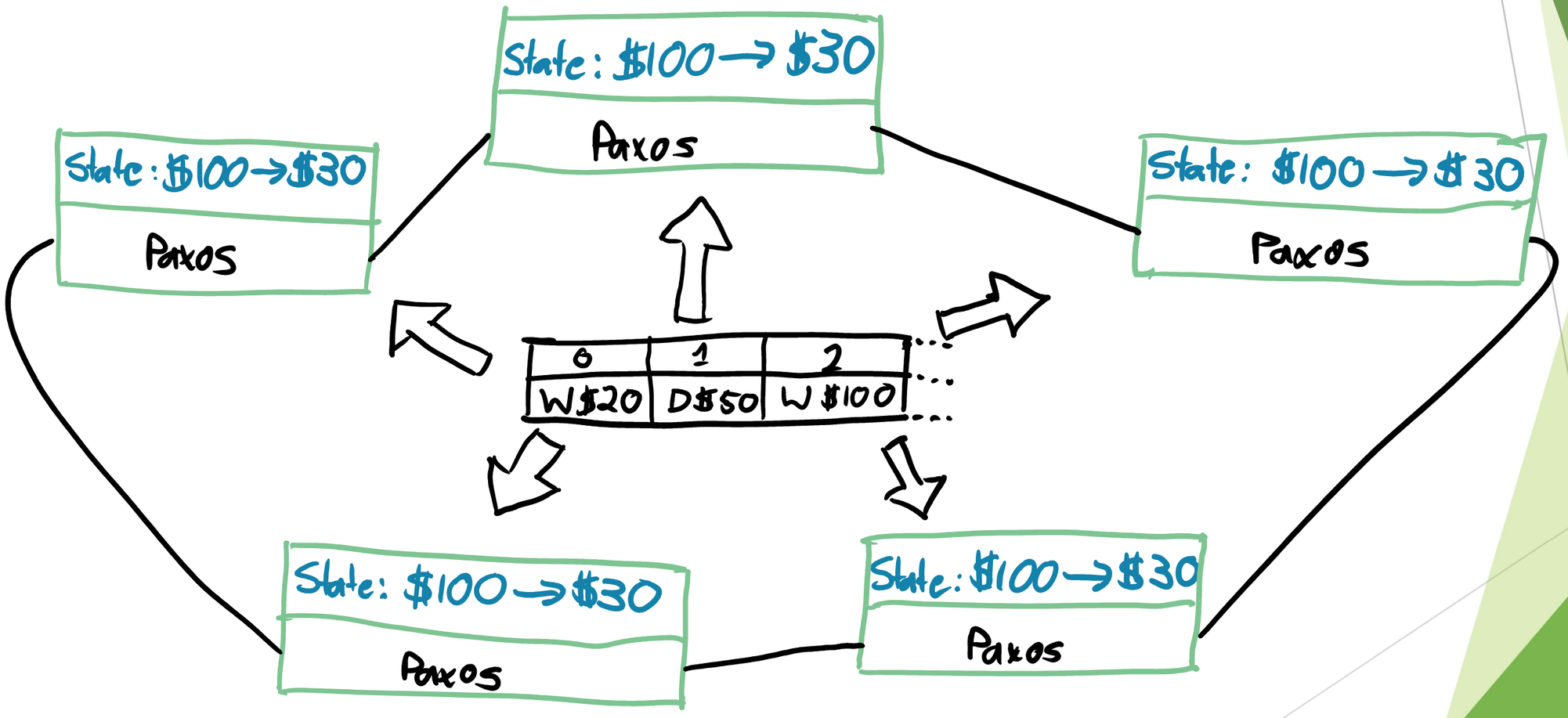
Fault-Tolerant Consensus

- How can we get a network of processes to agree to a single data value?
- Very difficult in the presence of faults; ad-hoc approaches always fail
  - Messages sent but not delivered
  - Messages delivered multiple times
  - Processes dying, missing messages, then later recovering
- What does it mean for processes to “agree” anyway?
  - Usually if majority (quorum) choose single value, that value is agreed upon
- No deterministic fault-tolerant consensus protocol can guarantee progress
  - All we can do is design protocols such that problems are unlikely to occur

What is the Paxos Protocol?

- The Paxos Protocol solves fault-tolerant consensus!
- Introduced by Leslie Lamport in 1998
- Reputed to be difficult to understand; this is a myth, as we shall see!
- High-level overview:
  - A single elected leader (proposer) handles all client requests
  - The protocol has two phases, prepare and accept
  - Can withstand complete loss of a minority of nodes
  - Protocol can become livelocked, but this state is unlikely and unstable

The bank replicas as state machines



Zooming in on a single transaction: Paxos in its entirety

- Phase 1a: Prepare
  - Proposer (leader) receives a client request, so creates a proposal tagged with ordered GUID **N**
  - *Prepare* message sent to all Acceptors, containing **N**
- Phase 1b: Promise
  - If **N** is greater than any proposal ID previously seen by the Acceptor, Acceptor returns a *Promise* message
  - The *Promise* message indicates it will reject any future proposals with value less than **N**
  - If the Acceptor previously accepted a proposal, it must include its ID and value in the message
- Phase 2a: Acceptance
  - If the Proposer received promises from the majority of Acceptors (a quorum), this phase is entered
  - If any Acceptors returned a previously accepted proposal, its value overwrites the client request
  - The Proposer sends an *Accept* request to all acceptors with **N** and the associated value
- Phase 2b: Chosen
  - Acceptor accepts *Accept* request IFF it has not returned a *Promise* message for ID greater than **N**
  - If the majority of Acceptors accept the request, the value is chosen and cannot be overwritten

Leader Election

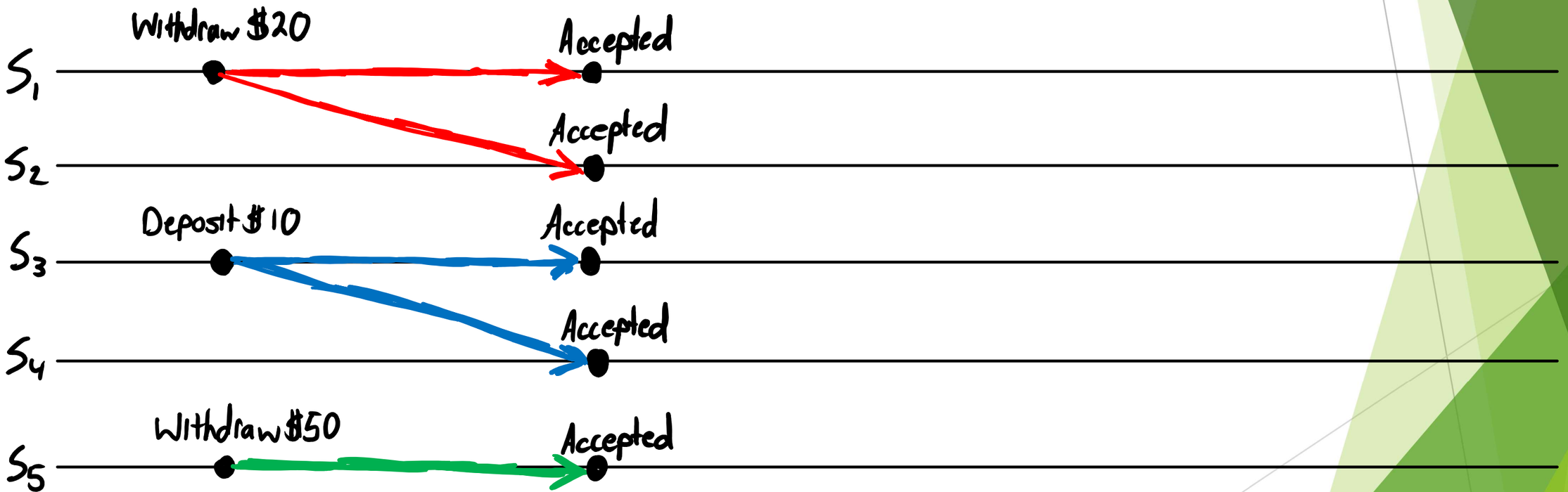
- Leader election can be as complicated or simple as you want it to be!
- Simple leader election algorithm:
  - Number each process from 1 to *N*
  - Processes send heartbeat messages to every other process every *T* seconds
  - If process *X* does not receive a heartbeat from any process *Y* > *X* in 2*T* seconds, it elects itself leader and begins to service client requests
- Paxos is designed to gracefully handle multiple concurrent leaders

Designing our own consensus protocol!

- We'll take a few shots to see how simpler approaches fail
- Good for seeing *why* Paxos works the way it does
- What are our requirements?
  1. Consistency: only one value can be chosen
  2. Immutability: once a value is chosen, we cannot choose a different value
  3. Durability: progress is possible as long as the majority of nodes are reachable
- We'll define chosen to mean accepted by the majority of acceptors

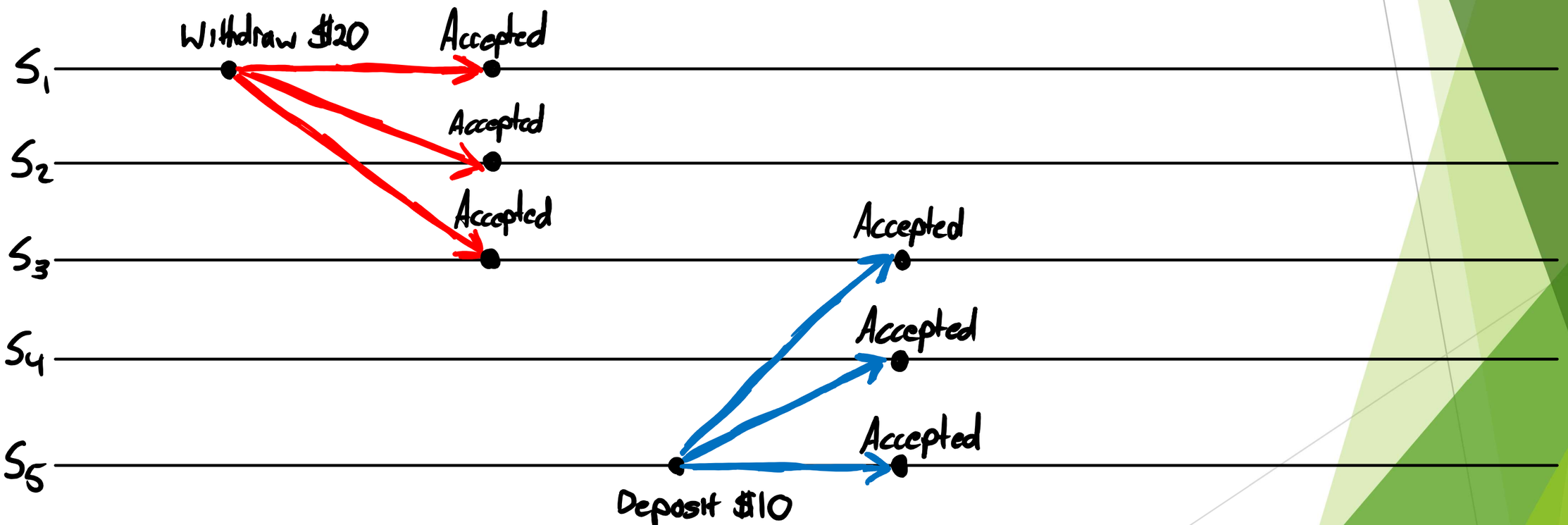
Designing our own consensus protocol!

Attempt #1: single round, Acceptors accept first value they receive



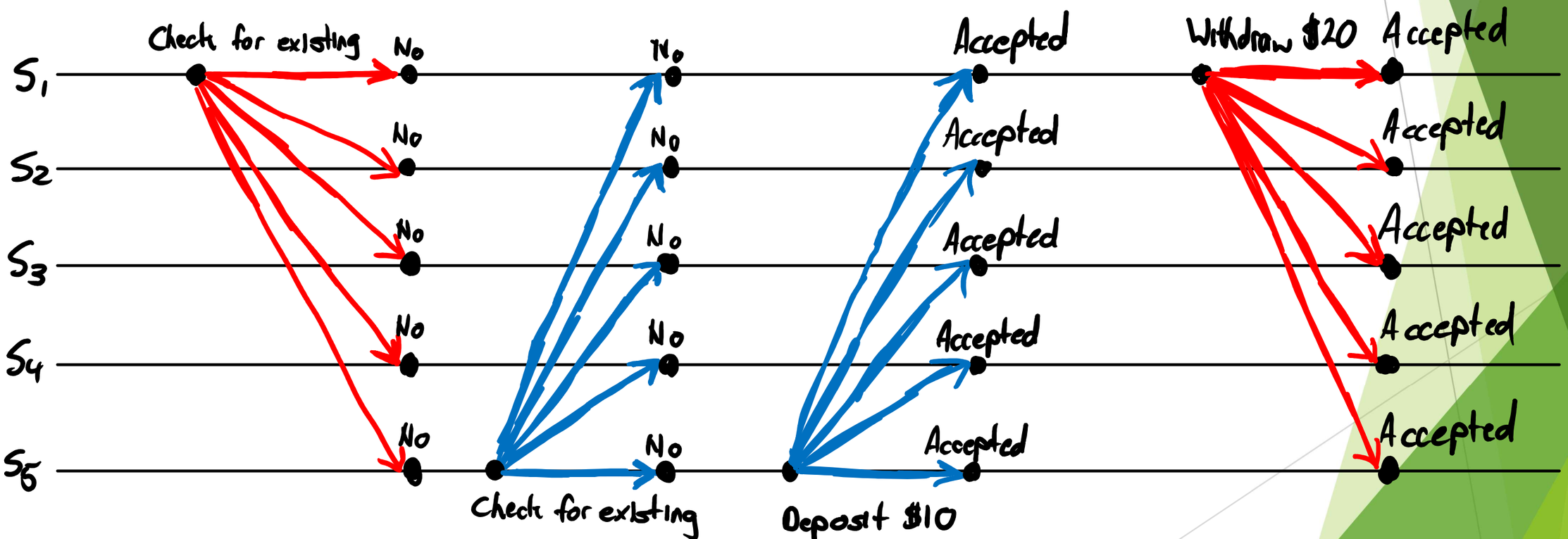
Designing our own consensus protocol!

Attempt #2: single round, Acceptors accept every value they receive



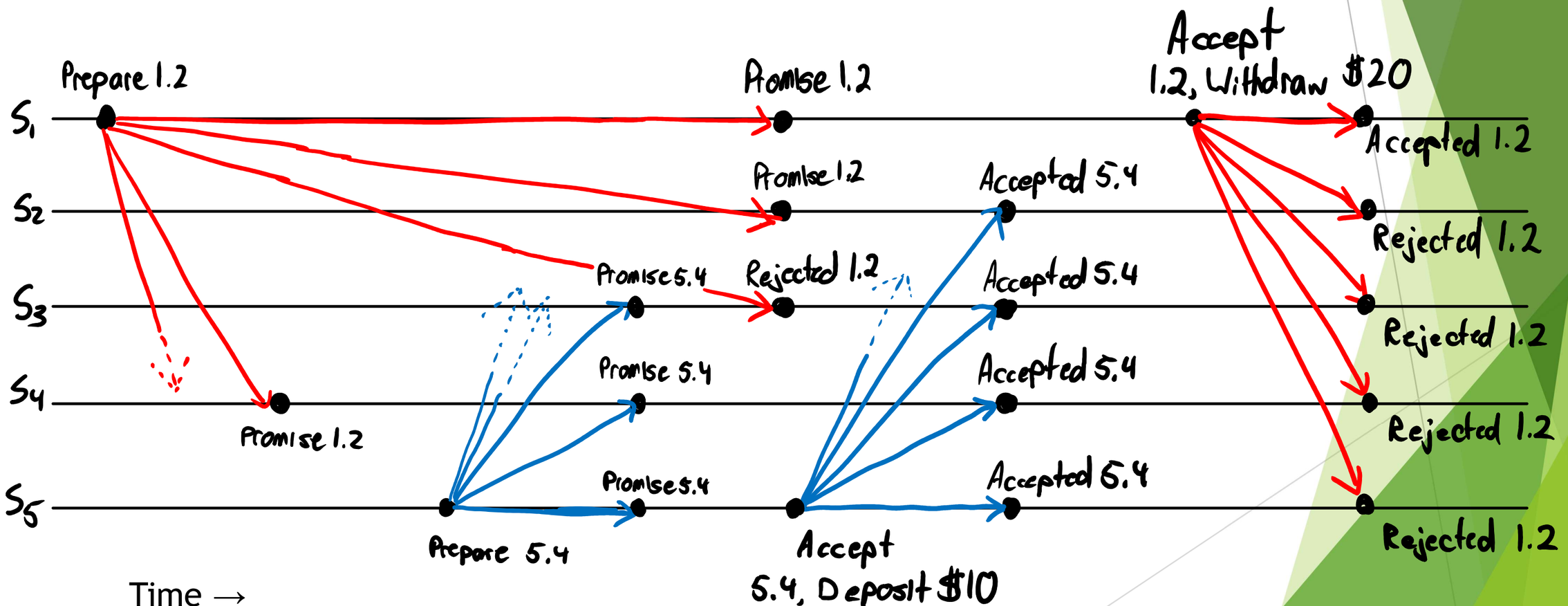
Designing our own consensus protocol!

Attempt #3: two rounds, first check whether acceptors have accepted a value



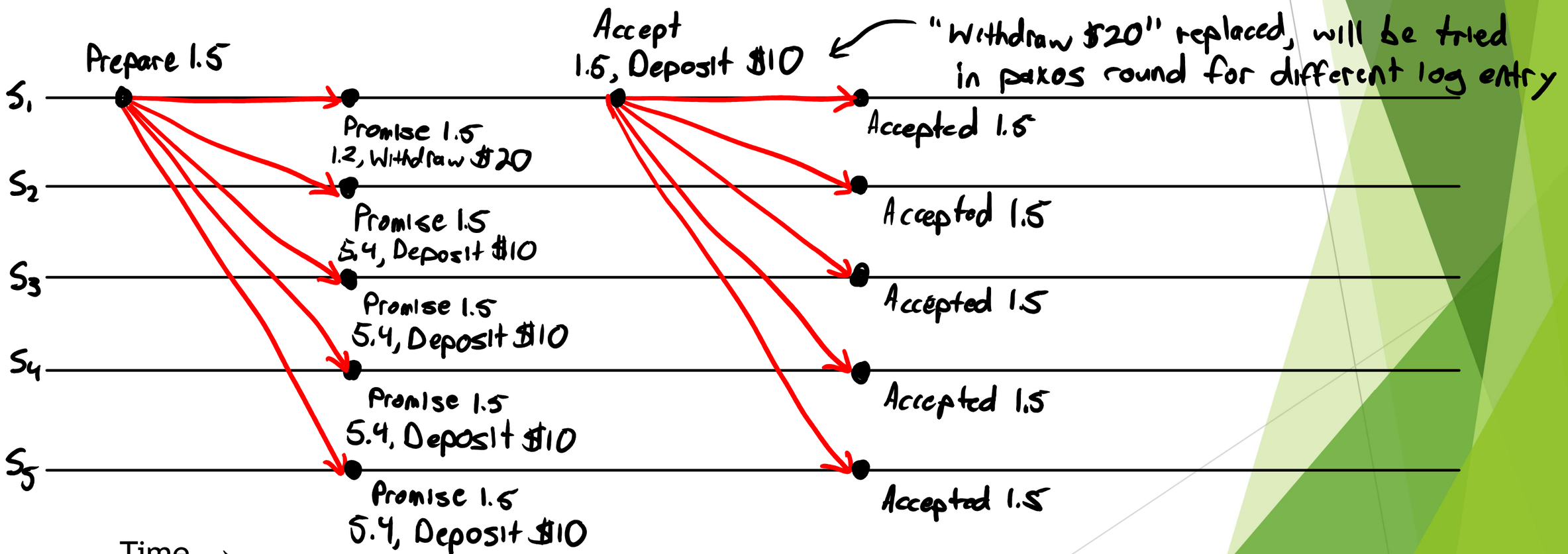
Designing our own consensus protocol!

Attempt #4: two rounds, proposal IDs, acceptors reject lower proposal IDs

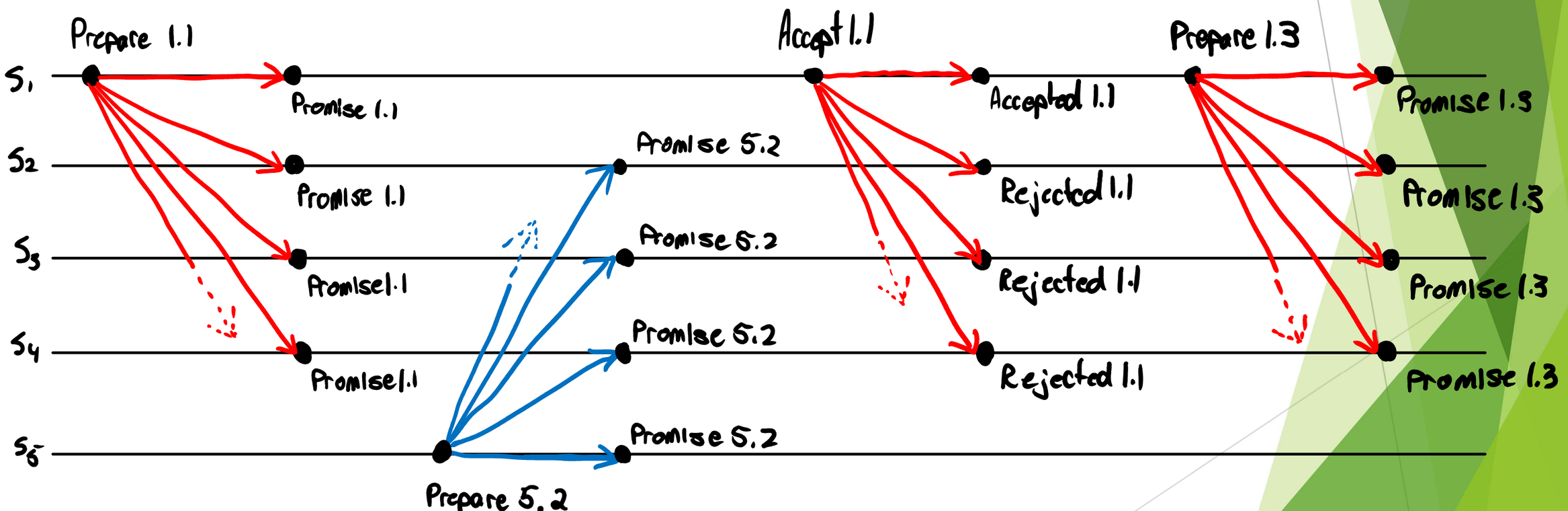


Designing our own consensus protocol!

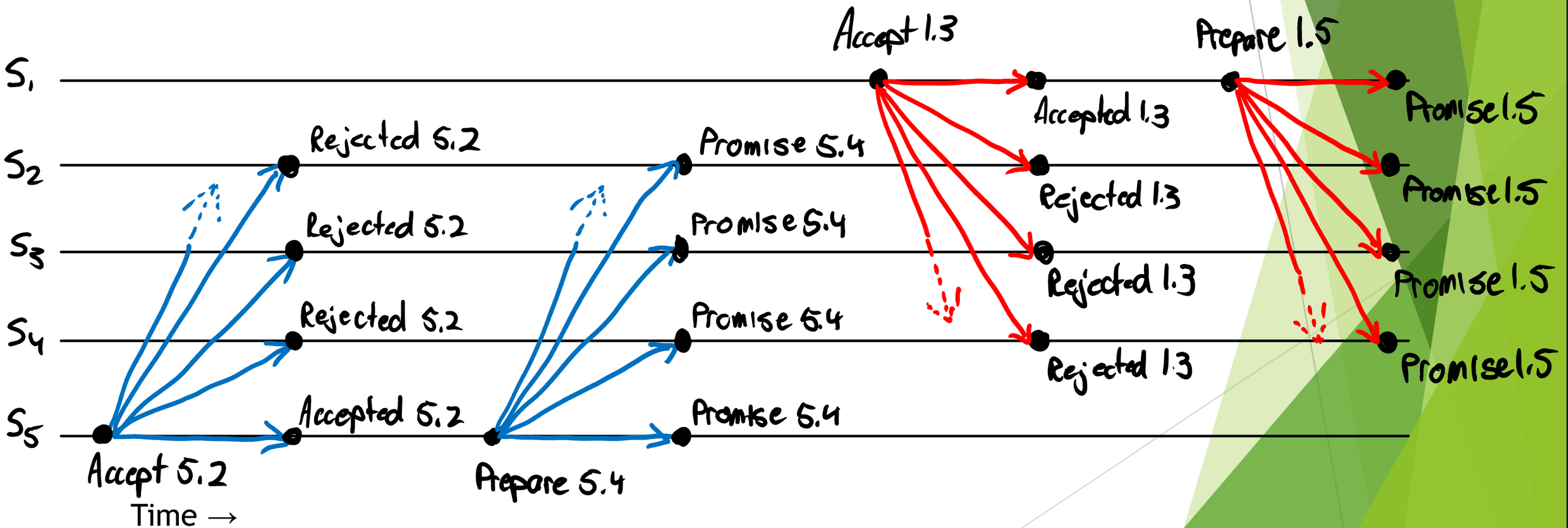
Attempt #4 continued: proposer converges on chosen value & terminates



We cannot guarantee progress :(



We cannot guarantee progress :(



TLA+ Spec

- Highlights:
  - Marvel at the Phase2a definition
  - The Consistency definition is an excellent example of an invariant
  - Fun to play around with temporal assertions!