Fast Paxos

Trevor Chan

Outline

1. Paxos Protocol
2. Fast Paxos Protocol

Consensus Correctness Criteria

- **Safety**
  - If value is chosen, then value must be chosen by any other process that has chosen a value
  - Value chosen must have been proposed by one of processes in system
  - Only value chosen by process can be learned by a process
- **Liveness**
  - Eventually, some value is chosen and a process in the system can learn that value

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
- 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

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!

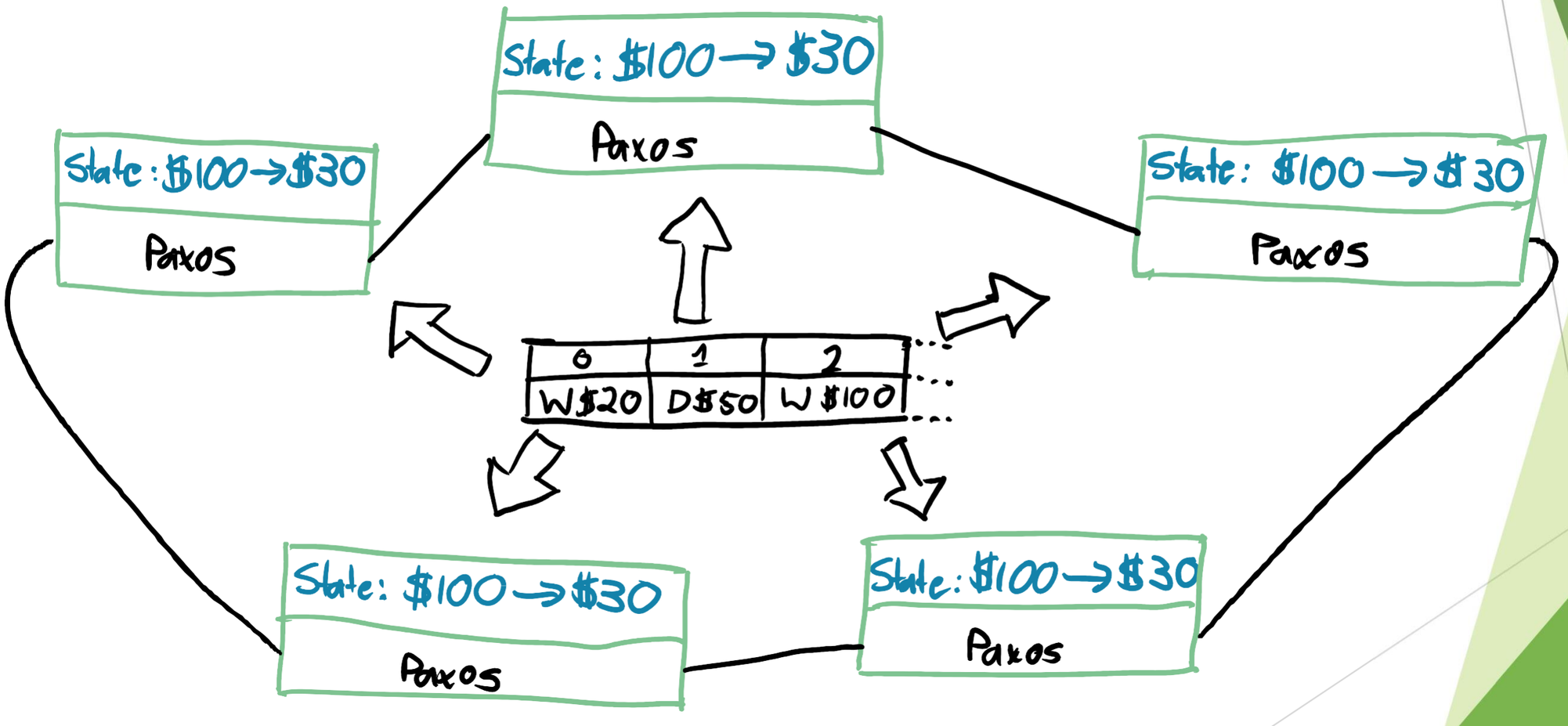
How can the bank maintain consistency among the replicas?

Bank Account Problem

What should the bank achieve through replication?

- Confirmed transactions - deposit & withdrawal - don’t disappear (Safety)
- Customers able to deposit &withdrawn when server crashes are not too many (Liveness)

The bank replicas as state machines

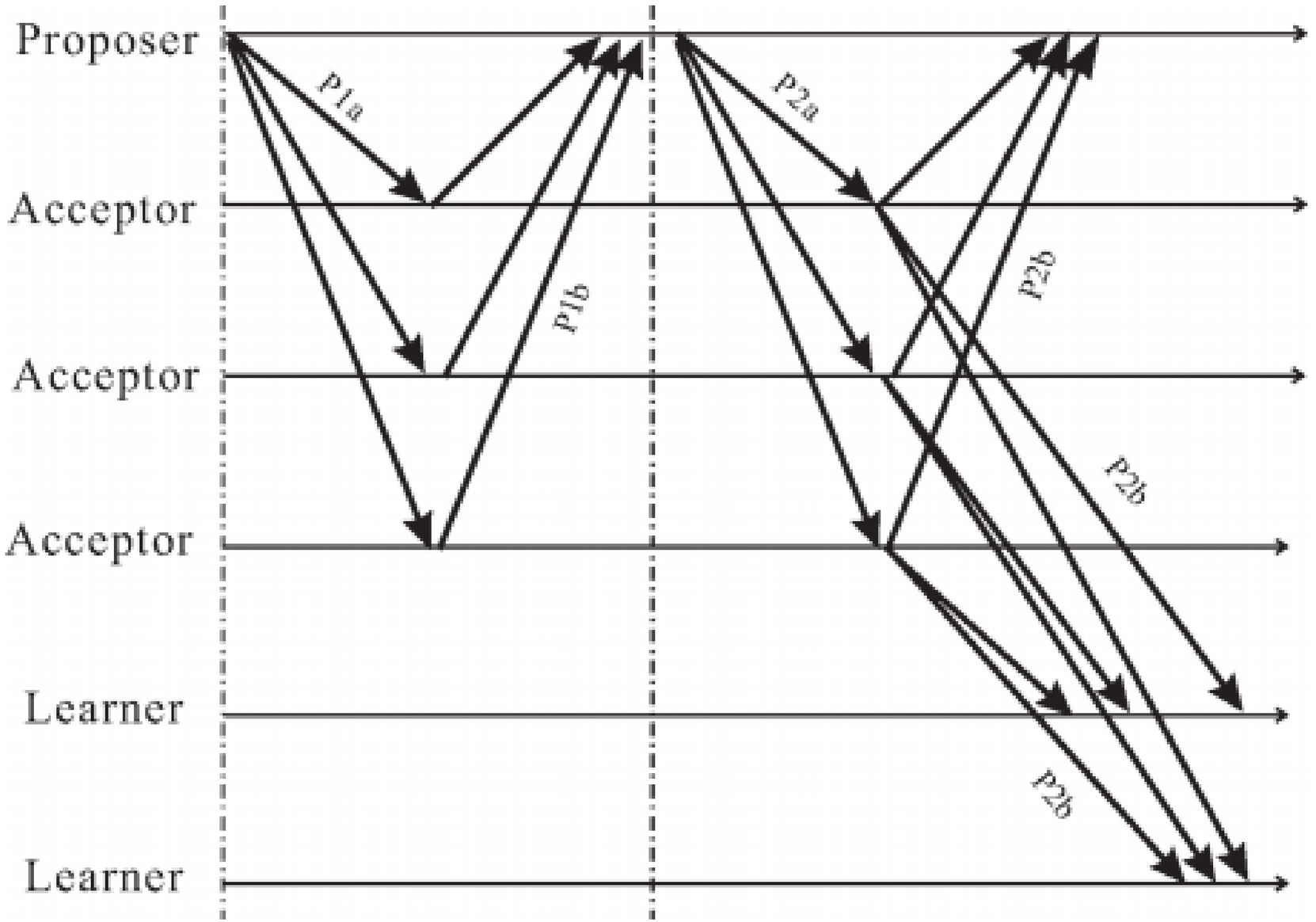


Paxos Roles

- **Proposer/Coordinator**
  - Proposes values to be chosen (by acceptors) and learned (by learners)
- **Acceptor**
  - Participates in agreement negotiation on the values proposed
- **Learner**
  - Learns the values that are chosen

Paxos: Phases in a single transaction

- **Phase 1a (P1a): Prepare**
  - Proposer (Coordinator) receives a client request, so creates a proposal tagged with ordered ID N
  - Prepare message sent to all Acceptors, containing N
- **Phase 1b (P1b): 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 ID value less than N
  - If the Acceptor previously accepted a proposal, it must include its ID and value in the message
- **Phase 2a (P2a): Propose**
  - 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 (P2b): Accept**
  - 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

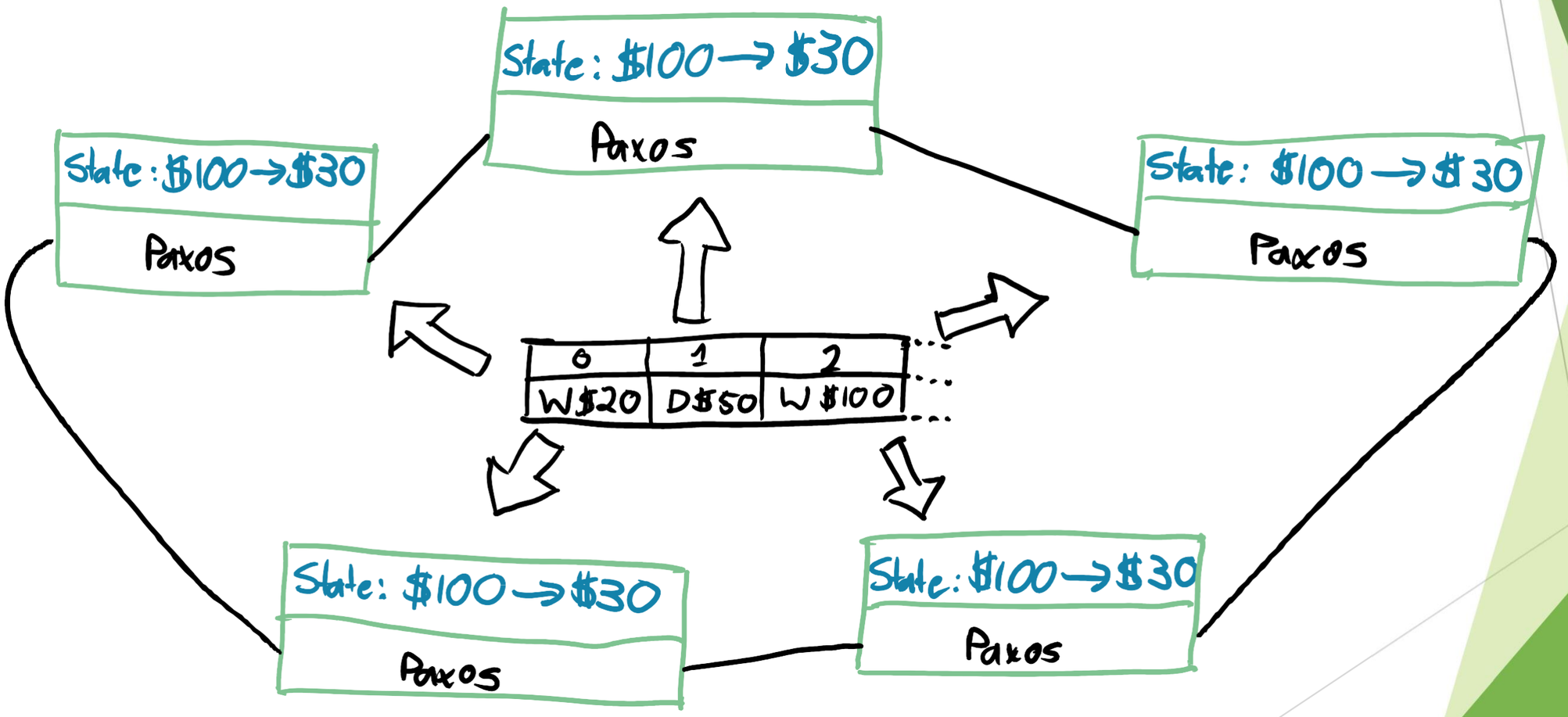

Proposer
Acceptor
Acceptor
Acceptor
Learner
Learner
P1a
P1b
P2a
P2b
P2b
P2b
Time →

Fast Paxos

1. Reduces end-to-end latency of reaching a consensus in scenarios when clients are responsible to propose values to be chosen by acceptors

   - Reduces cost of reaching consensus by enabling running of one P2a message for all instances of Fast Paxos in state-machine replication

State Machine Approach



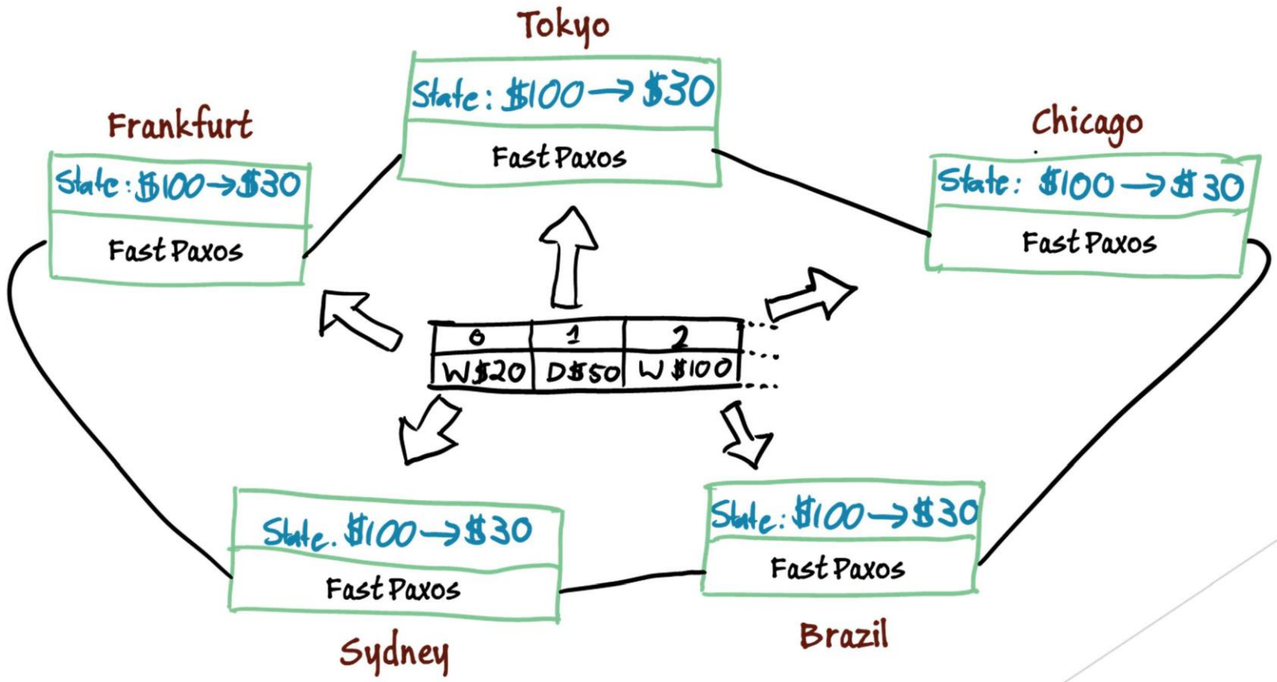
Classic Paxos

- Replicating single transaction
  - 1st RTT – Phase 1 (prepare request & response)
  - 2nd RTT – Phase 2 (accept request & response)

- Building block in cloud services (AWS, Azure, Google, …)
  - Replication across multiple servers in every datacenter

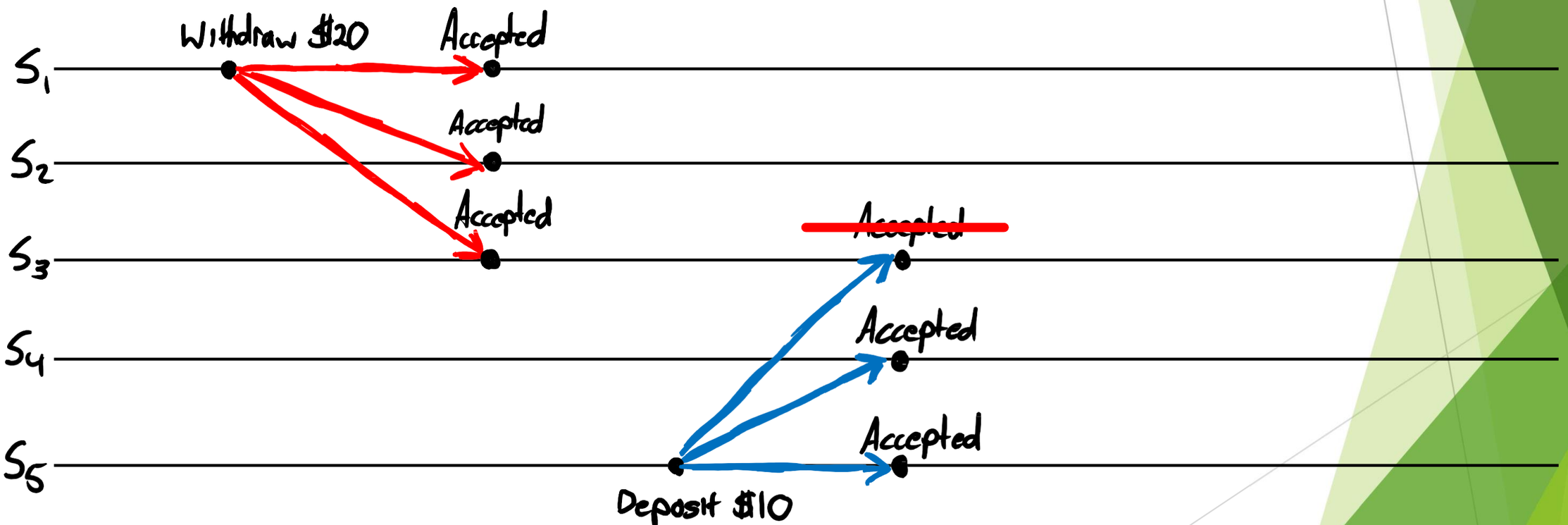
Fast Paxos

- Replicating transactions across geographically distributed datacenters
  - surviving earthquakes, etc.
- Fast Paxos – single RTT
  - Classic Paxos – 2 RTTs



Is Simple Majority Sufficient?

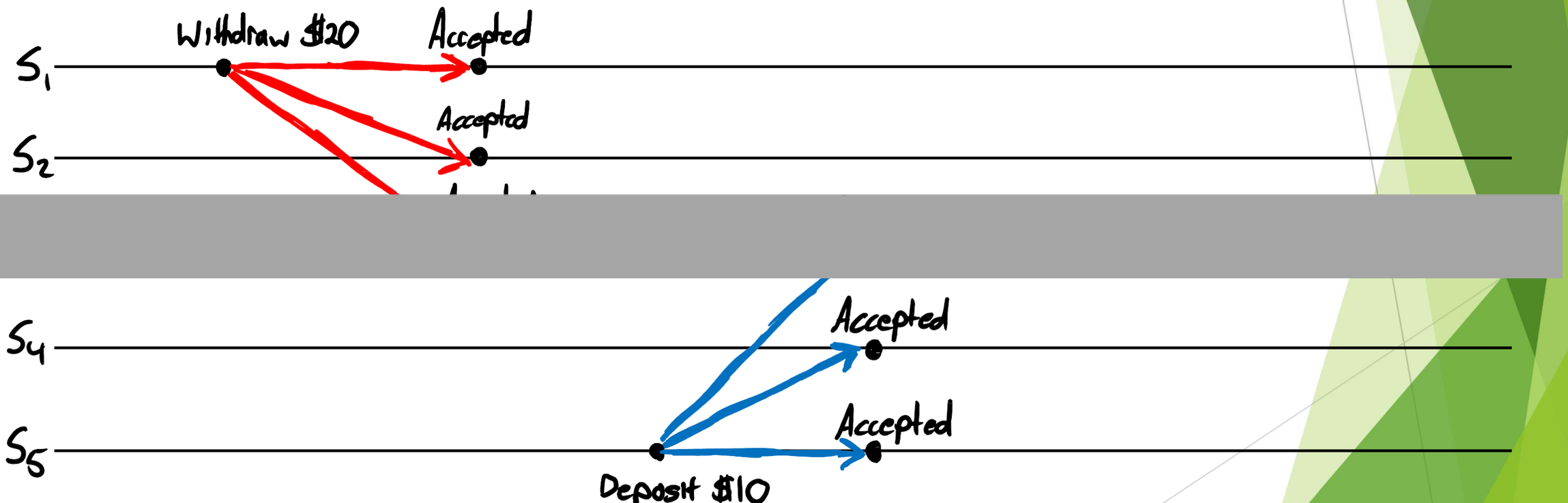
- Accept only the first value + declare success with simple majority



Any problem?

Is Simple Majority Sufficient?

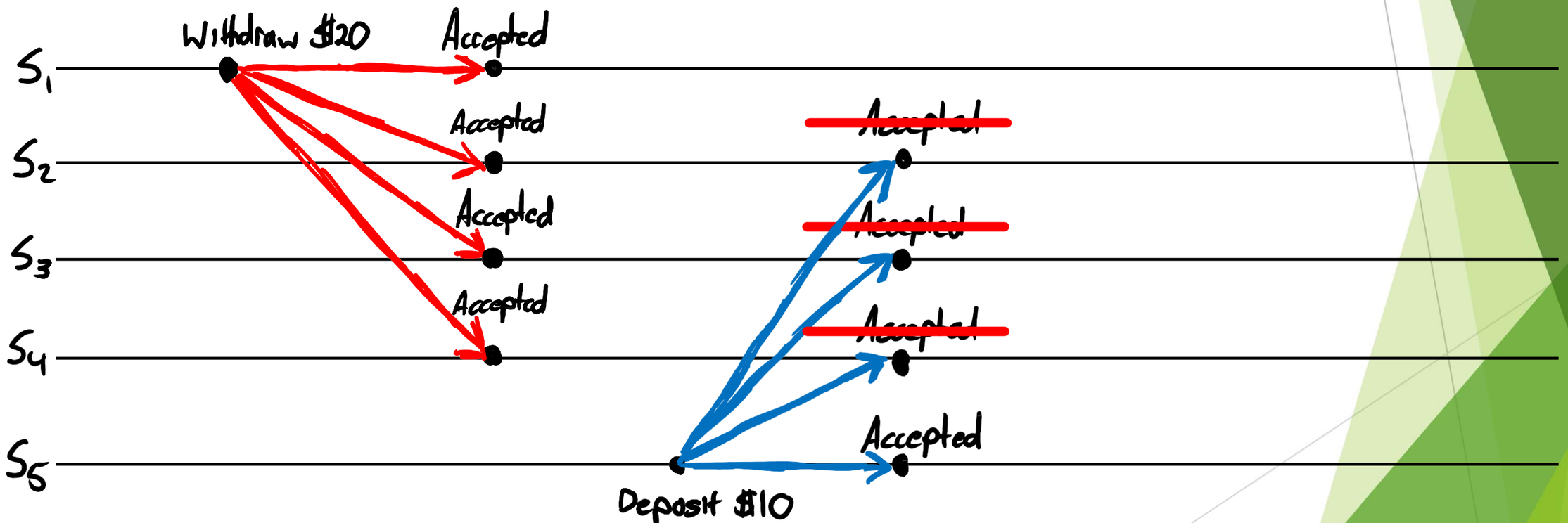
- What if S_3 is gone forever? Was it red, blue or neither?



How can we avoid ambiguity and fix this?

Avoiding Ambiguity with Larger Quorum

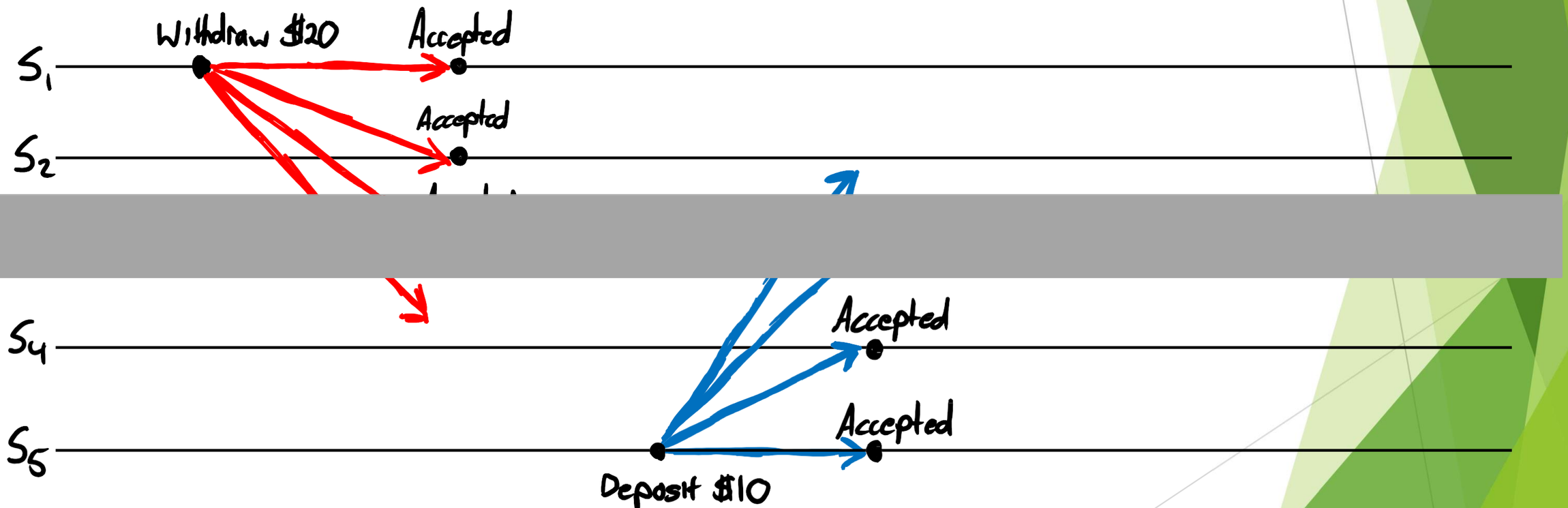
- Choose larger quorum (4 out of 5) + declare success with quorum



Does larger quorum indeed avoid ambiguity?

Avoiding Ambiguity with Larger Quorum

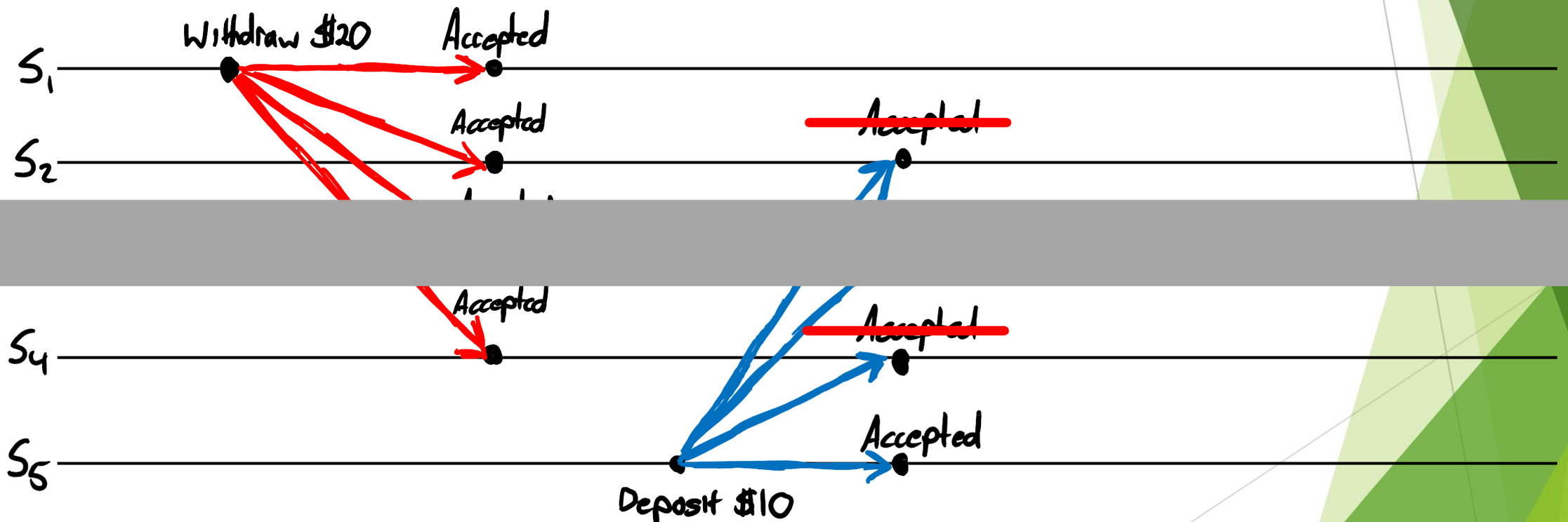
- Observing 2 red and 2 blue ⇒ neither red nor blue made it



Forget both red and blue ⇒ treat as clean slate

Avoiding Ambiguity with Larger Quorum

- Observing 3 red and 1 blue ⇒ be conservative and retry red



Time →

run Classic Paxos with red

Recap

- Choose larger quorum (Ex: 4 out of 5 servers)
- Perform single RTT request & response
  - send transaction to all 5 servers and solicit responses
- Inspect any quorum of responses
  - No collision: quorum containing single accepted value
    - transaction succeeded
- Collision recovery case I: multiple accepted values w/o majority
  - treat as clean slate
- Collision recovery case II: multiple accepted values w/ majority
  - run Classic Paxos with the majority value

Additional Details

- Previous algorithm isn't exactly Fast Paxos, but covers core idea
- Additional details of Fast Paxos
  - How to choose quorum size?
  - Collision recovery completes in single RTT
    - Classic Paxos would have taken 2 RTTs

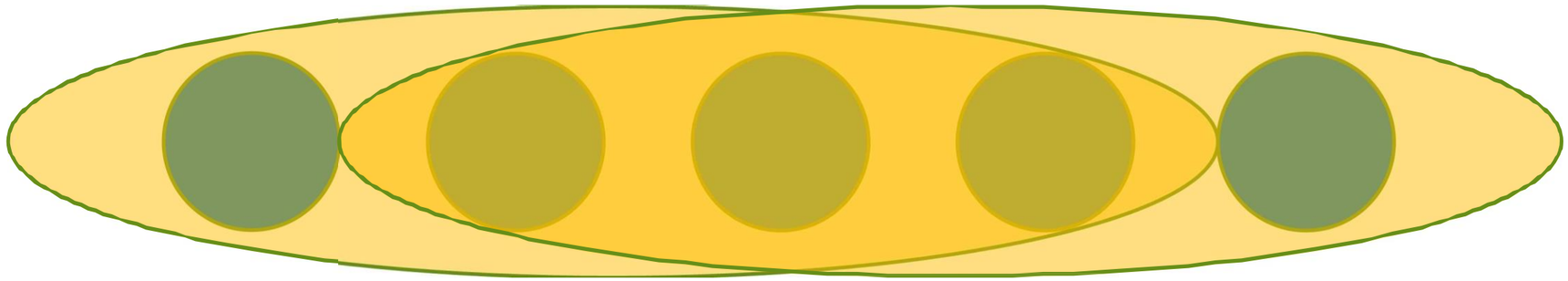
Quorum Size

- Two types of rounds
  - Fast round
  - Classic round –most identical to Classic Paxos
  - Quorum size may differ in fast and classic rounds

- Quorum rule of Fast Paxos

$$Quorum_{FAST_i} \cap Quorum_{FAST_j} \cap Quorum_{CLASSIC}$$

Quorum Size



|FAST quorum|=4 => |CLASSIC quorum|=3

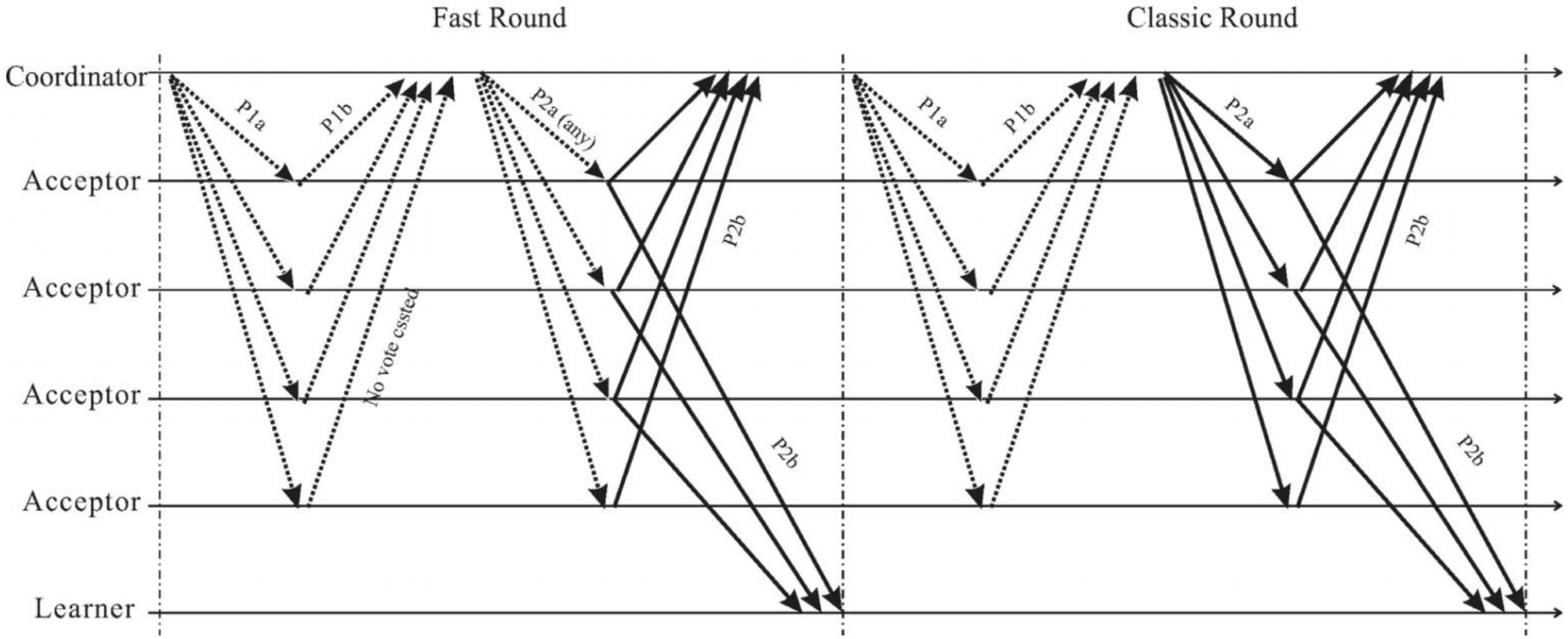
Single RTT Completion in Fast Paxos

- Both fast round and classic round take two RTTs
  - 1st RTT – Phase 1 (prepare request & response)
  - 2nd RTT – Phase 2 (accept request & response)

- Key idea behind single RTT completion
  - Phase 1 can be omitted, when it is implied by
    - initial state
    - messages in previous round

Quorum Size

# of Replicas	|Fast Quorum|	|Classic Quorum|
3	3	2
5	4	3
7	5	5
9	7	5

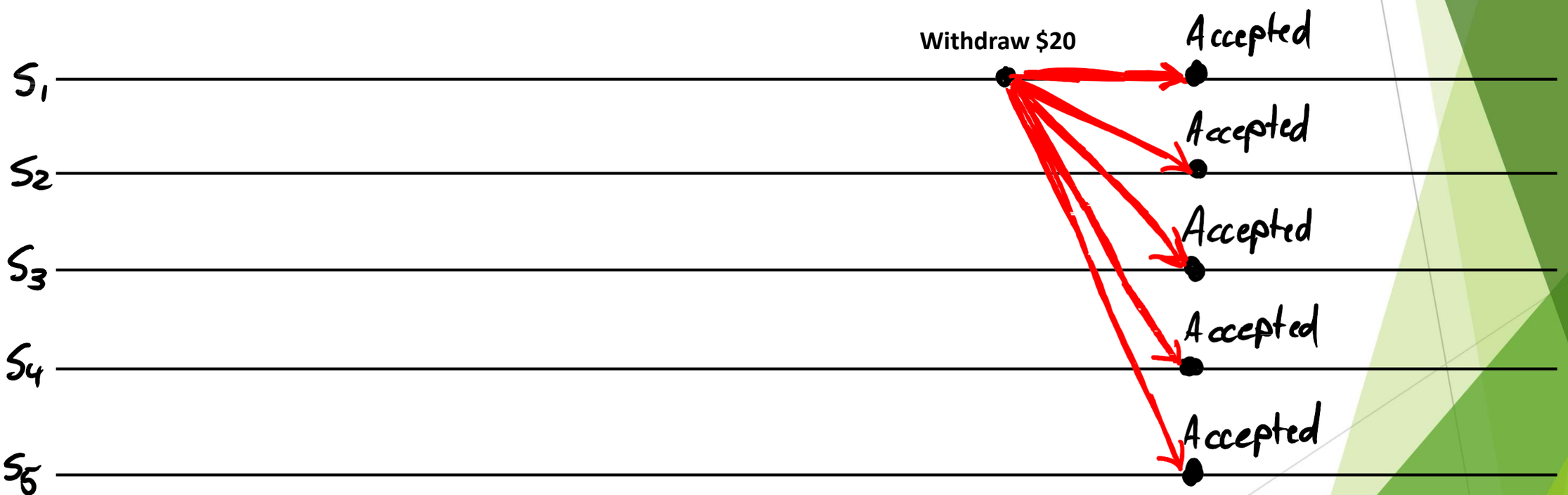

Fast Round
Classic Round
Coordinator
Acceptor
Acceptor
Acceptor
Acceptor
Learner
P1a
P1b
P2a (any)
P2b
P2a
No vote cssted
Time →

Example Walkthrough: Fast Round 0

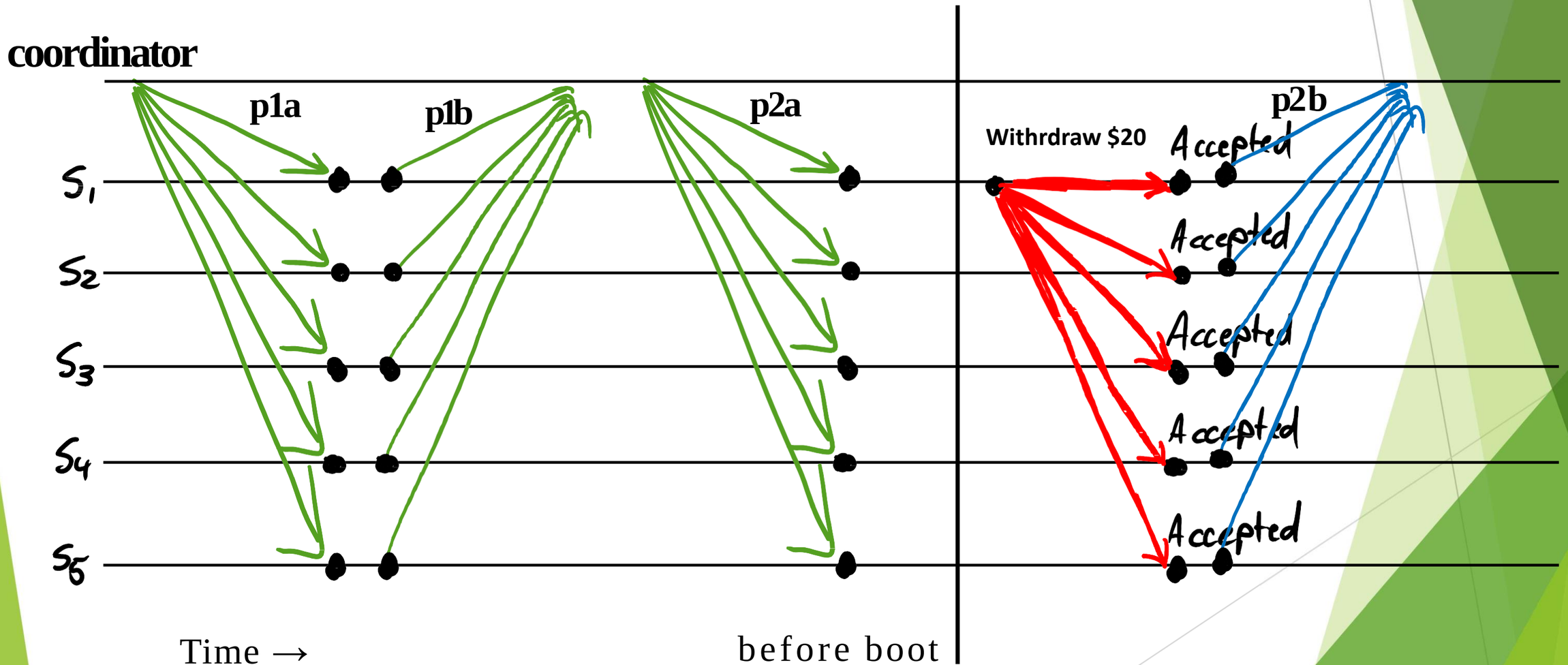
- **Phase 1a (p1a) :** coordinator ➔ all acceptors
  - Prepare request: [phase1a, round = 0]
- **Phase 1b (p1b) :** acceptors ➔ coordinator
  - Prepare response: [phase1b, round = 0, $acceptor_j$]
- **Phase 2a (p2a) :** coordinator ➔ all acceptors
  - Accept request: [phase2a, round = 0, value = any]
- **Phase 2b (p2b) :** acceptors ➔ coordinator
  - Accept response: [phase2b, round = 0, $acceptor_j$, value = v_j]
  - v_j: arbitrary value chosen independently by each acceptor

pre-executed
before boot
=>
safe to omit

FAST Round 0


Withdraw $20
Accepted
Accepted
Accepted
Accepted
Accepted
s_1
s_2
s_3
s_4
s_5
Time →

FAST Round 0


coordinator
p1a
p1b
p2a
p2b
Withdraw $20
Accepted
Accepted
Accepted
Accepted
s_1
s_2
s_3
s_4
s_5
Time →
before boot

Single RTT Collison Recovery

- $round_i$ accept response
  - [phase2b, round = i, $acceptor_j$, value = v_j]

- $round_{i+1}$ prepare response
  - [phase1b, round = i+1, $acceptor_j$, voted_round = i, voted_value = v_j]

- $round_i$ accept response => $round_{i+1}$ prepare response
  - safe to omit $round_{i+1}$ Phase 1

Summary

- Simplified Fast Paxos
  - Larger quorum
  - Single RTT request & response
  - Quorum of responses: unique value, w/ or w/o majority
- How to choose quorum size?
- How omitting Phase 1 makes Paxos fast?