Dr. TLA+ Series – Fast Paxos

Cheng Huang

August 29th, 2016

template from Andrew Helwer

Fast Paxos

Leslie Lamport

14 July 2005
Revised 18 January 2006
Minor revision 14 April 2006

MSR-TR-2005-112

To appear in *Distributed Computing*.

Fast Paxos Made Easy:
Theory and Implementation

Wenbing Zhao, Department of Electrical and Computer Engineering, Cleveland State University, Cleveland, OH, USA

ABSTRACT

Distributed consensus is one of the most important building blocks for distributed systems. Fast Paxos is one of the latest variants of the Paxos algorithm for distributed consensus. Fast Paxos allows an acceptor to cast a vote for a value of its choice unilaterally in a fast round, thereby eliminating a communication step for reaching consensus. As a tradeoff, the coordinator must build a quorum that is bigger than the simple majority used in Classic Paxos. This article presents the theory, implementation, and a comprehensive performance evaluation of the Fast Paxos algorithm. The theory is described in an easier-to-understand way compared with the original article by Lamport. In particular, an easy-to-implement value selection rule for the coordinator is derived. In the implementation of Fast Paxos for state-machine replication, a number of additional mechanisms are developed to cope with practical scenarios. Furthermore, the experiments reveal that Fast Paxos is most appropriate for use in a single-client configuration. The presence of two or more concurrent clients even in a local area network would incur frequent collisions, which would reduce the system throughput and increase the mean response time as experienced by clients. Due to frequent collisions, Fast Paxos actually performs worse than Classic Paxos in the presence of moderate to large number of concurrent clients.

Keywords: *Distributed Consensus, End-to-End Latency, Fault Tolerance, Probability Density Function, Quorum Requirements, State-Machine Replication, System Throughput*

INTRODUCTION

Distributed consensus is one of the most important building blocks for distributed systems (Zhao, 2014). For example, it is impossible to build a highly available cloud service without using some distributed consensus algorithm to ensure that all replicas remain consistent (Camargos, Madeira, & Pedone 2006; Camargos, Schmidt, & Pedone, 2008; Zhao, Melliar-Smith, & Moser, 2010; Zhao, 2010). Fast Paxos (Lamport, 2006) is one of the latest variants of the original Paxos algorithm (Lamport, 2001) (referred to as Classic Paxos) for distributed consensus. Classic Paxos is a good fit for state-machine replication and it has been used in a number of practical fault tolerant systems (Bolosky et al., 2011; Burrows, 2006; Hunt et al., 2010; Mao et al., 2008; Rao, Shekita, & Tata, 2011). Fast Paxos aims to further reduce the latency for reaching consensus by using a larger quorum size. Similar to Classic Paxos, Fast Paxos operates in rounds and there are two phases in each round. If a consensus is not reached within a round, a new round will be launched for liveness. In Fast Paxos, there can

DOI: 10.4018/ijdst.2015010102

Fast Paxos

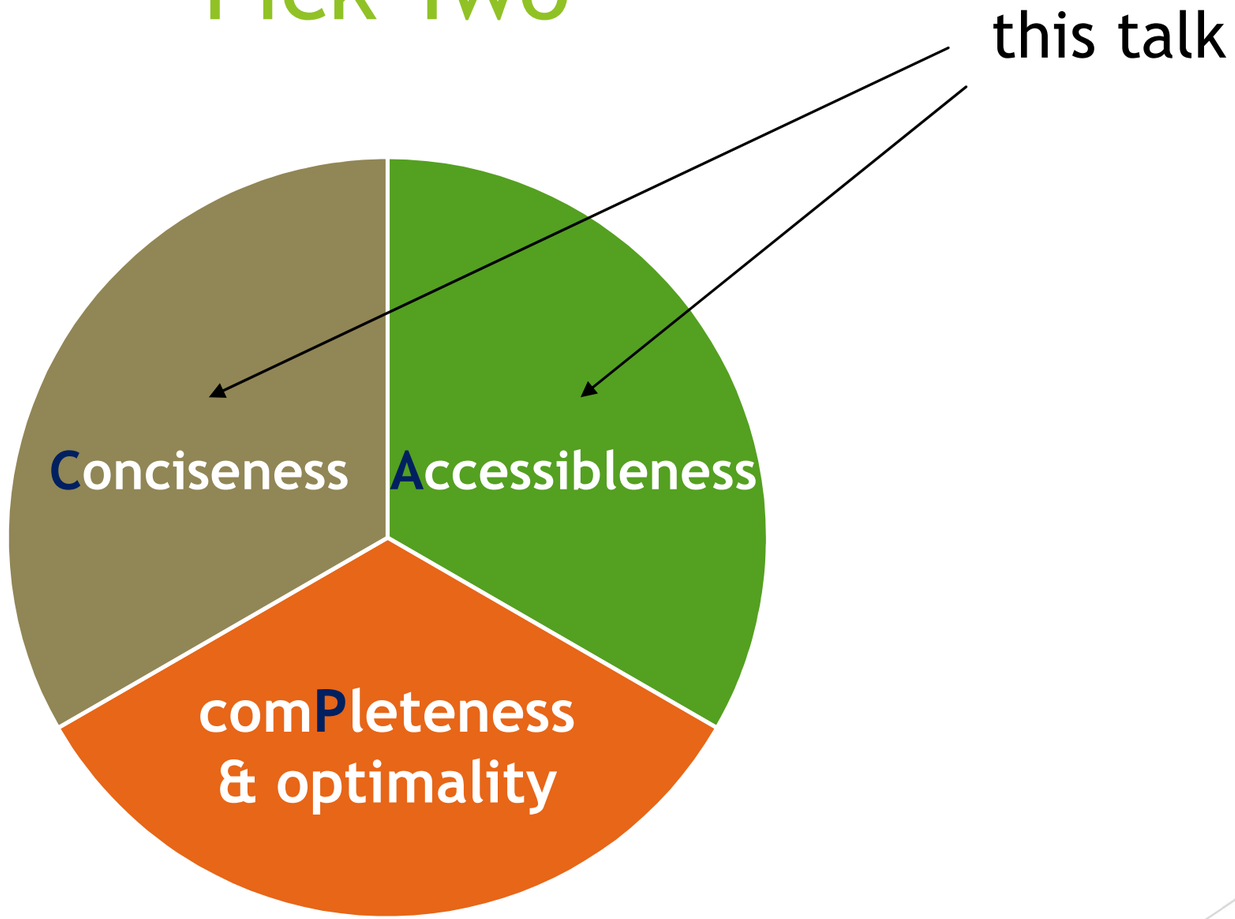
Leslie Lamport

14 July 2005
Revised 18 January 2006
Minor revision 14 April 2006

MSR-TR-2005-112

To appear in *Distributed Computing*.

Explanation of Complex Algorithm – Pick Two



Learning Objectives

1. What is the Fast Paxos protocol, and what problem does it solve?

2. How does Fast Paxos work, and why does it work that way?

3. What can the Fast Paxos TLA+ spec teach us about writing specifications?

Bank Account Problem

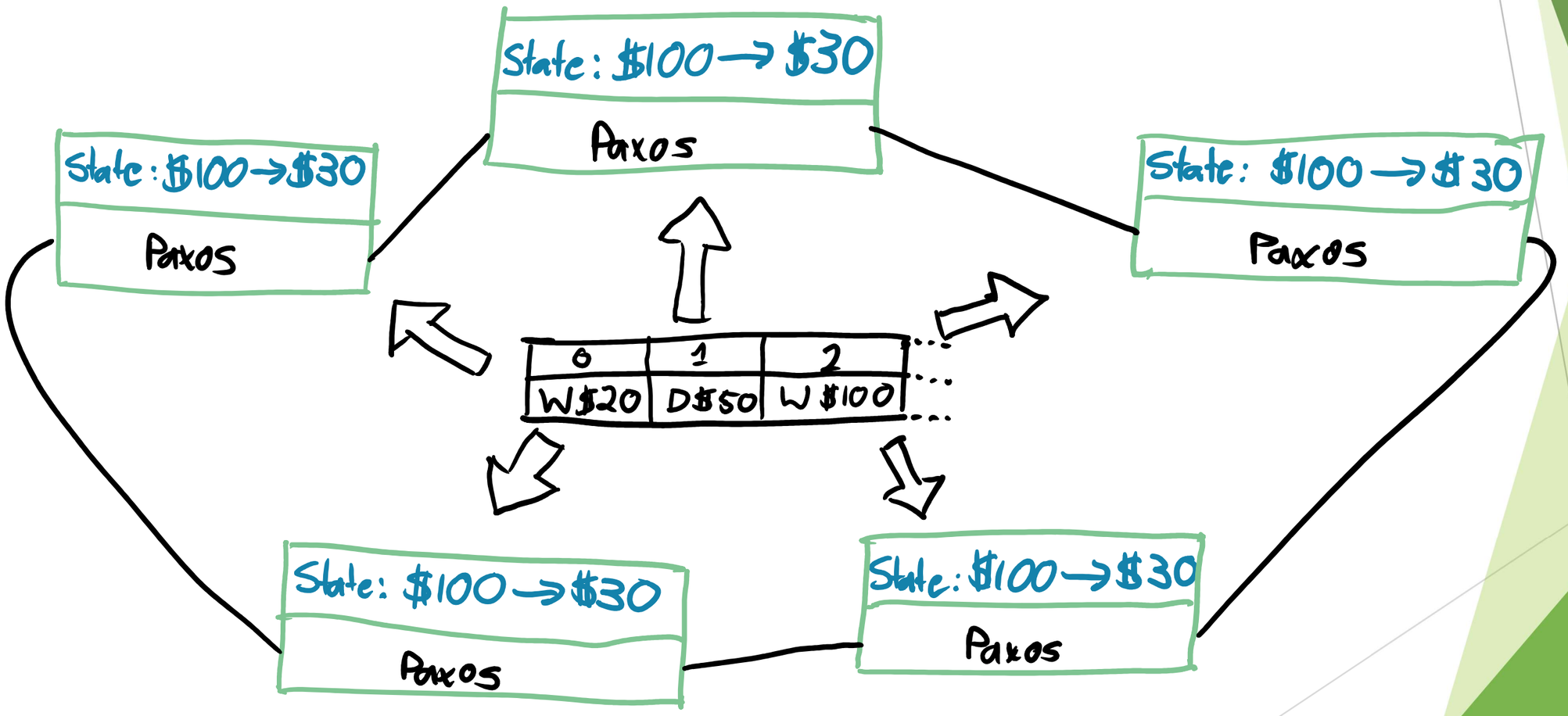
- Your bank has your account balance stored on a server
- Don’t want to lose account balance if the server crashes
- Solution: bank replicates the account balance to multiple servers!

Bank Account Problem

What should the bank achieve through replication?

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

State Machine Approach



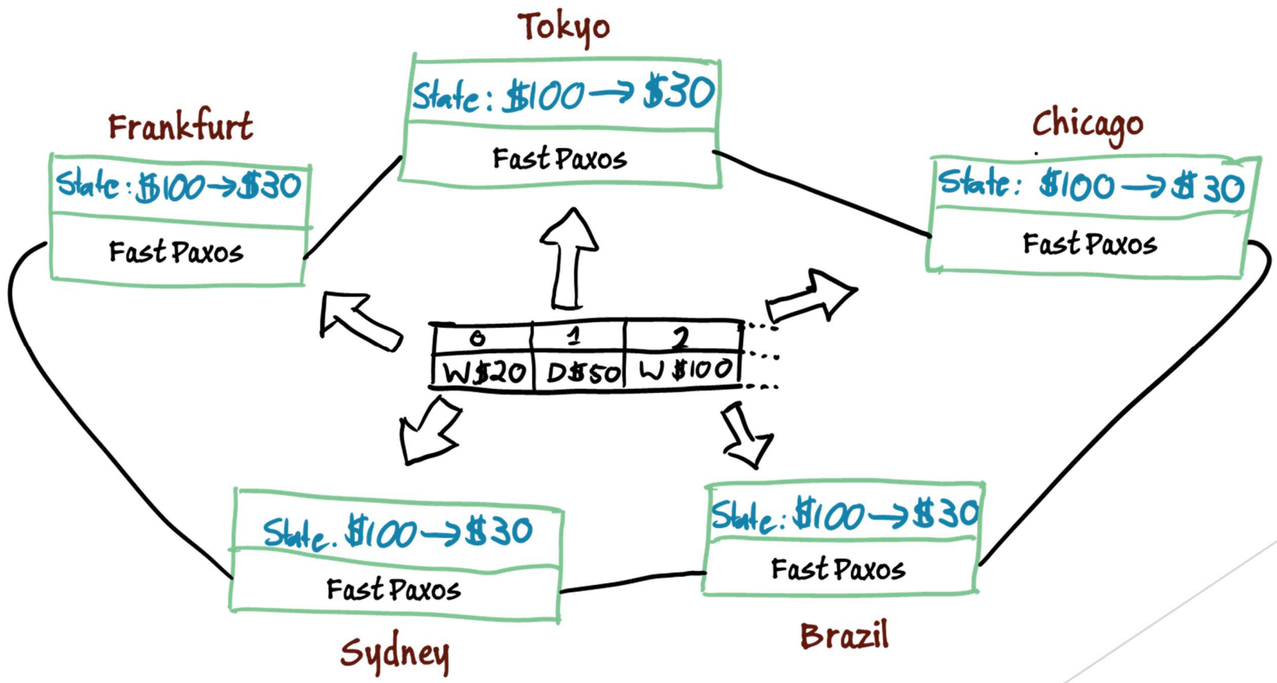
Classic Paxos

- 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 DC

Fast Paxos

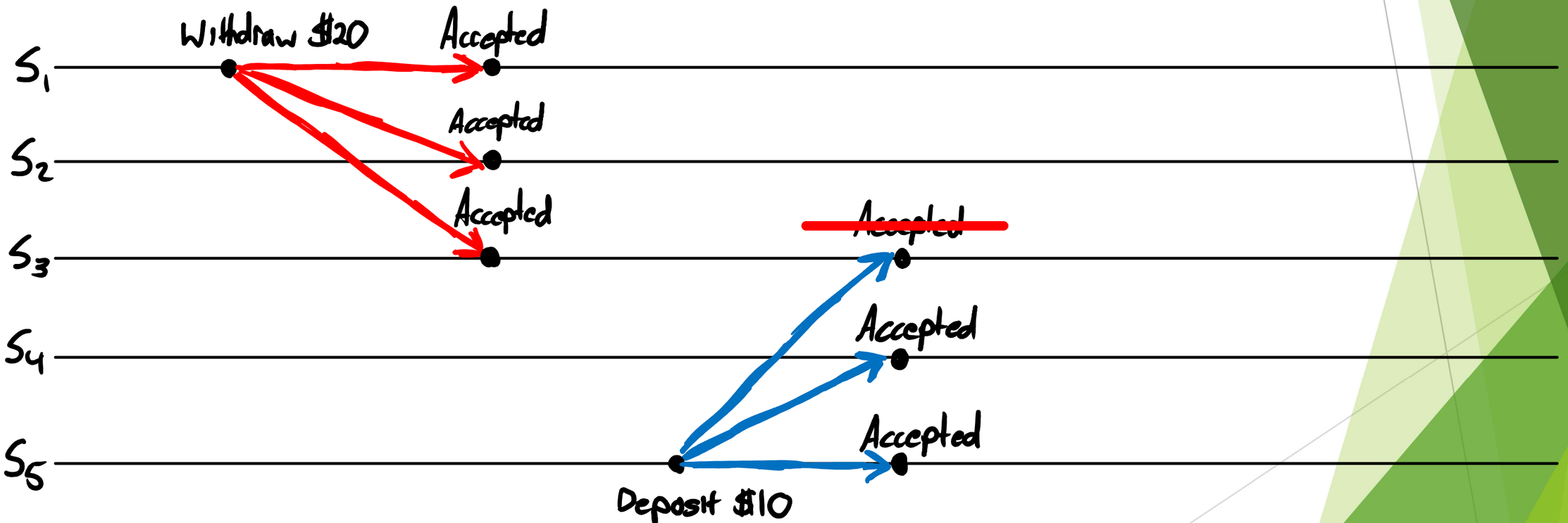
- Replicating transactions across geographically distributed DCs
  - surviving earthquake, etc.



- Fast Paxos – single WAN RTT
  - Classic Paxos – 2 WAN RTTs

Is Simple Majority Sufficient?

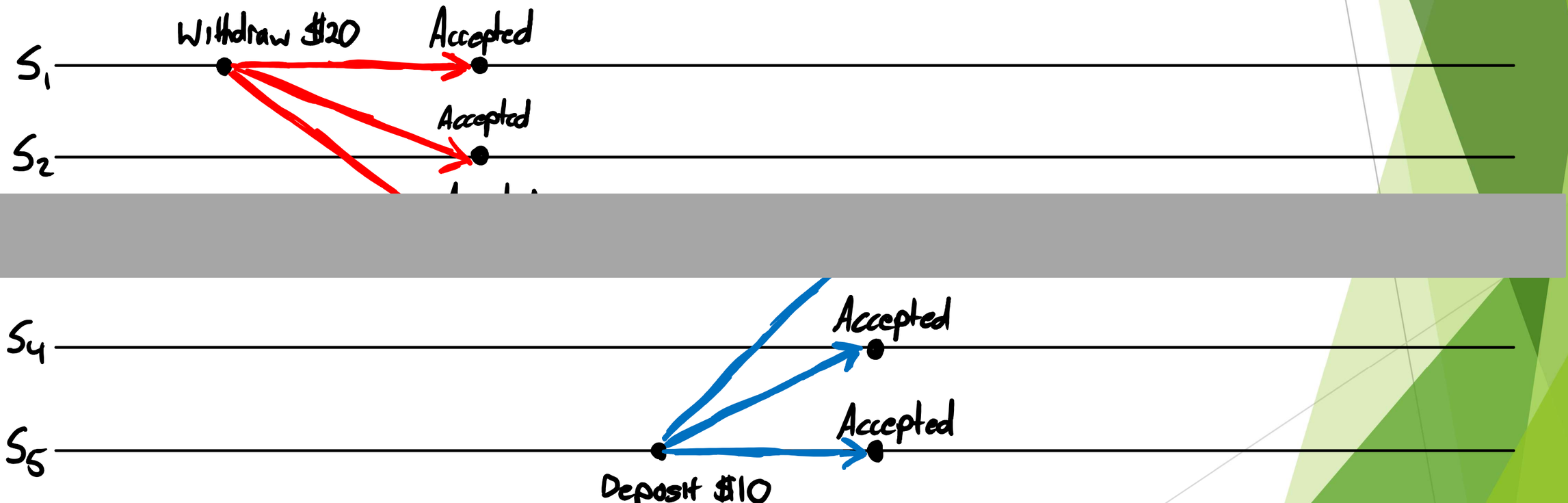
- Accept only the first value + declare success with simple majority



Any problem?

Is Simple Majority Sufficient?

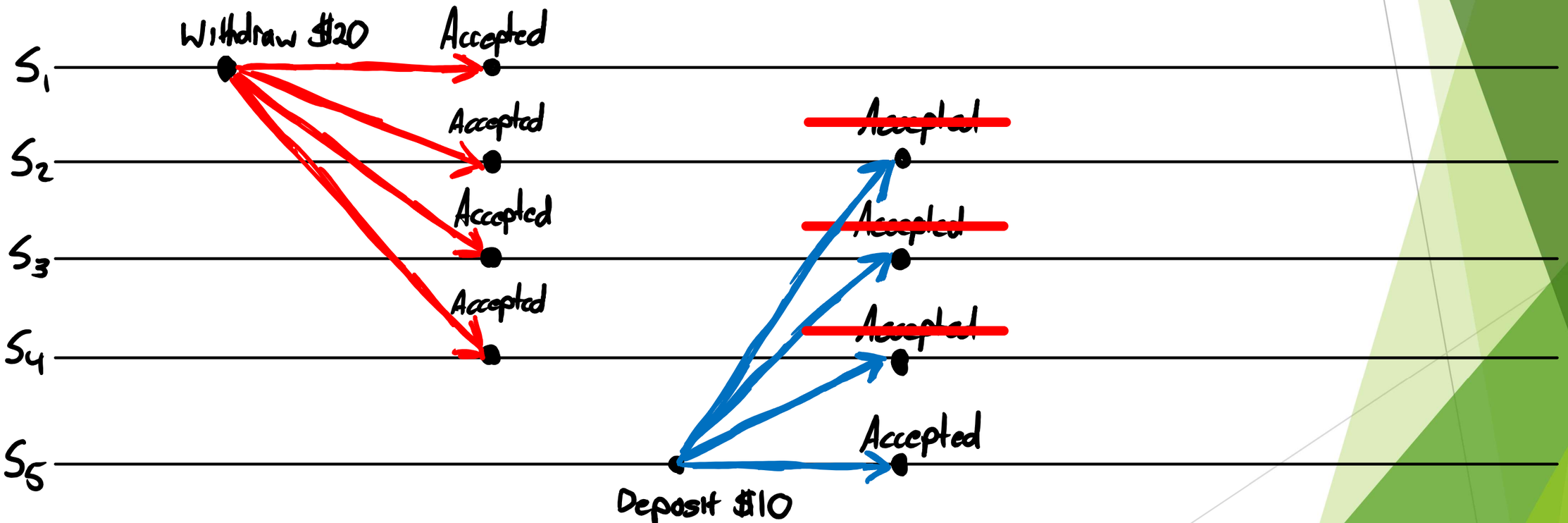
- What if S_3 is gone forever? Was it red, blue or neither?



How can we avoid ambiguity and fix this?

Avoiding Ambiguity with Larger Quorum

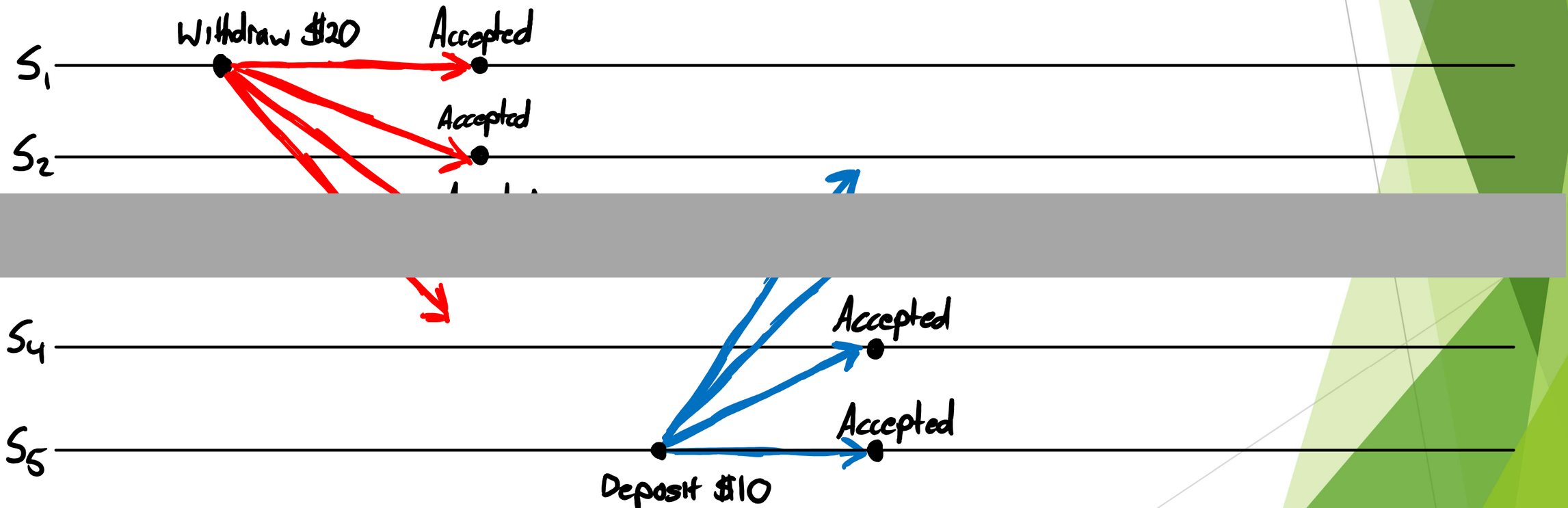
- Choose larger quorum (4 out of 5) + declare success with quorum



Does larger quorum indeed avoid ambiguity?

Avoiding Ambiguity with Larger Quorum

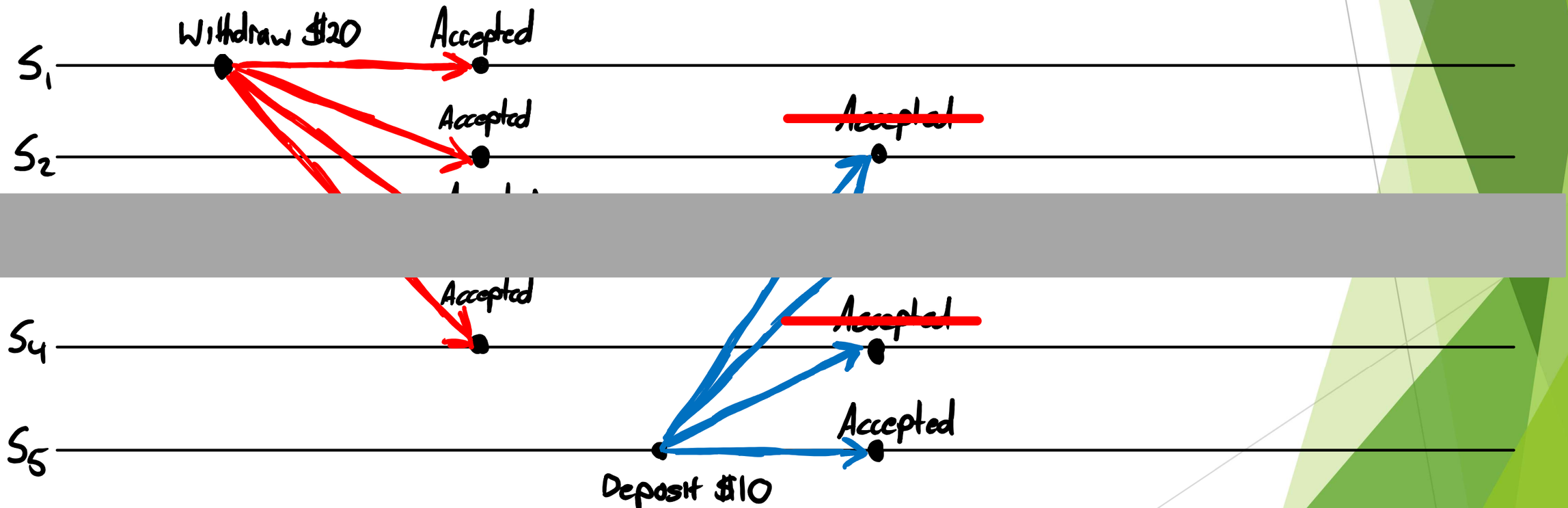
- Observing 2 red and 2 blue → neither red nor blue made it



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



run Classic Paxos with red

Recap

- Choose larger quorum (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

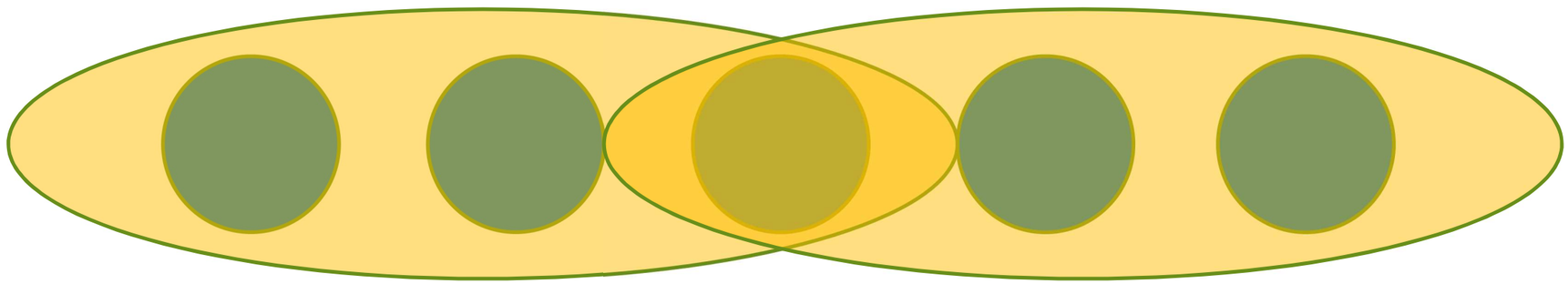
- Previous algorithm isn't exactly Fast Paxos, but covers the core idea
- Additional details of Fast Paxos
  - How to choose quorum size?
  - Collision recovery completes in single WAN RTT
    - Classic Paxos would have taken 2 WAN RTTs

Quorum Size

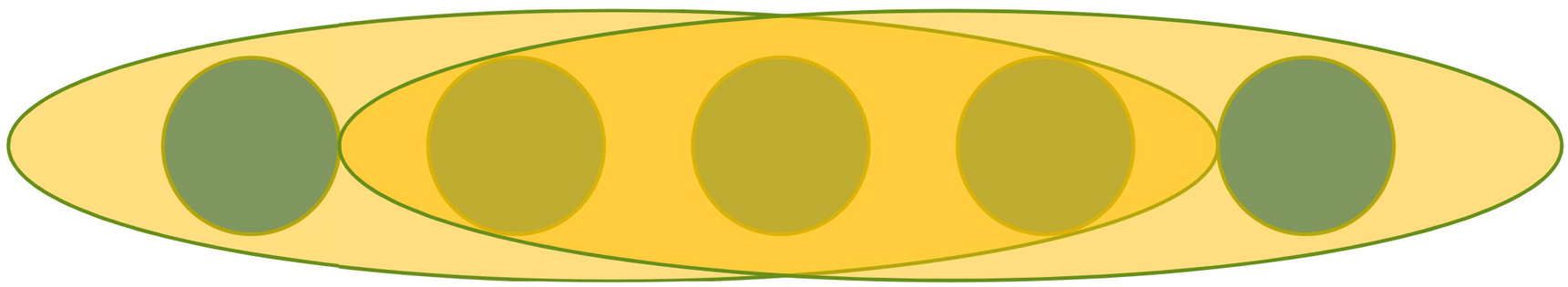
- Two types of rounds
  - FAST round – special
  - CLASSIC round – most identical to Classic Paxos
  - Quorum size may differ in FAST and CLASSIC rounds
- Quorum rule of Fast Paxos

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

Quorum Size



|FAST quorum|=3 => |CLASSIC quorum|=5



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

FAST quorum replica

Quorum Size

# of Replicas	|FAST Quorum|	|CLASSIC Quorum|
3	3	2
5	4	3
7	5	5
9	7	5

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)

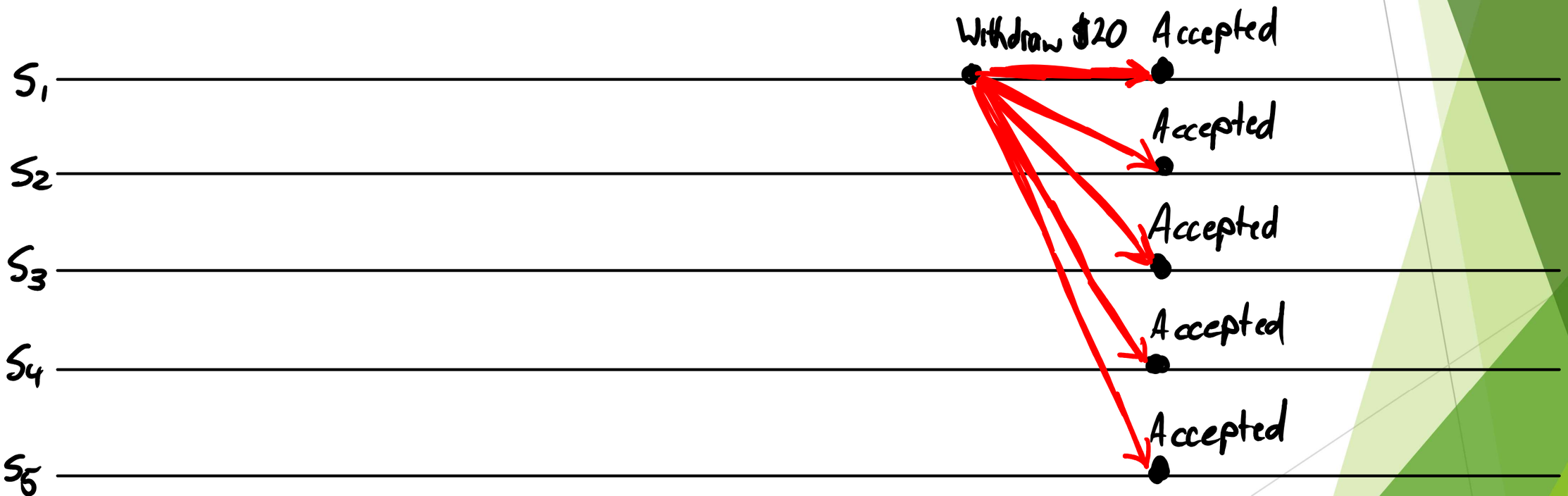
- Key idea behind single RTT completion
  - Phase 1 can be omitted, when it is implied by
    - initial state
    - messages in previous round

FAST Round 0

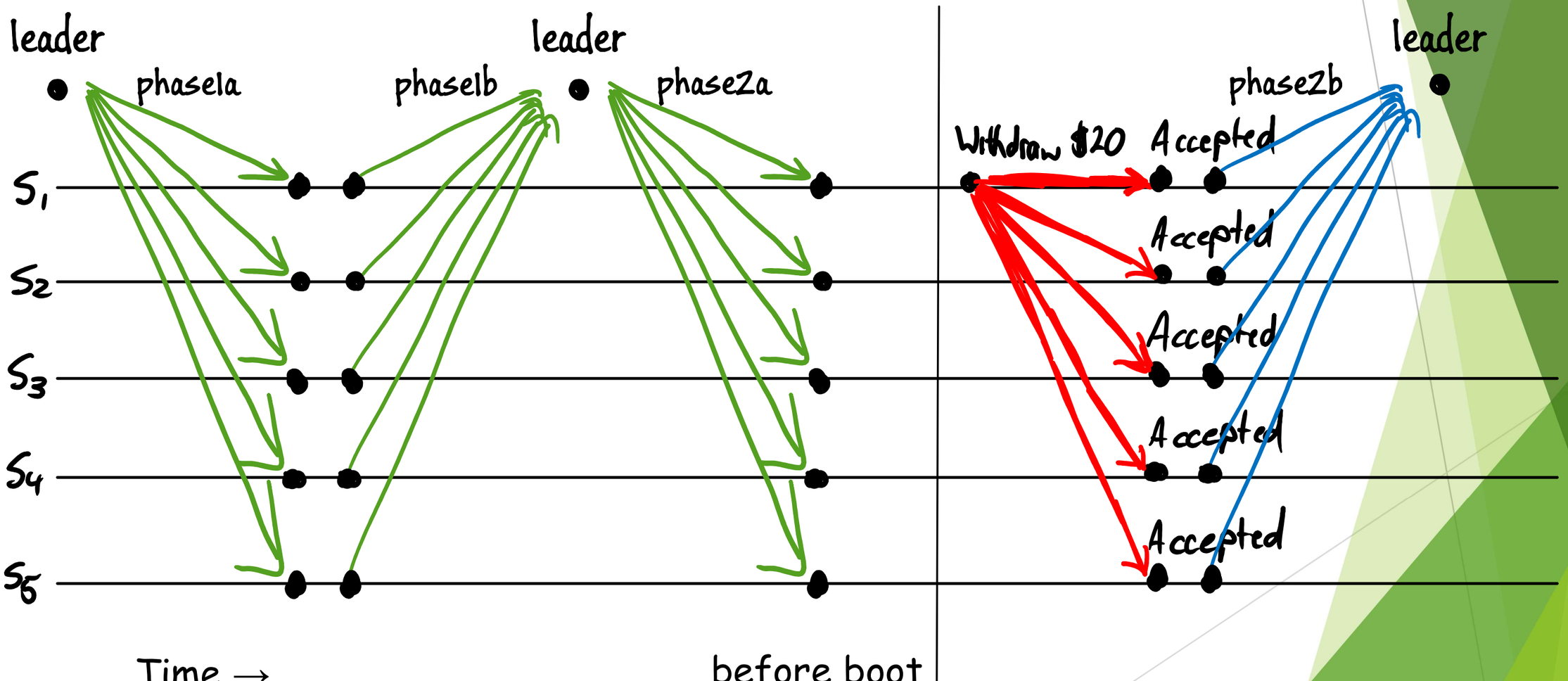
- Phase 1a: leader → all acceptors
  - Prepare request: [phase1a, round = 0]
- Phase 1b: acceptors → leader
  - Prepare response: [phase1b, round = 0, $acceptor_j$]
  - Acceptors promise to not accept anything with round < 0
- Phase 2a: leader → all acceptors
  - Accept request: [phase2a, round = 0, value = any]
- Phase 2b: acceptors → leader
  - 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



red messages are not Paxos; green messages are omitted

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?