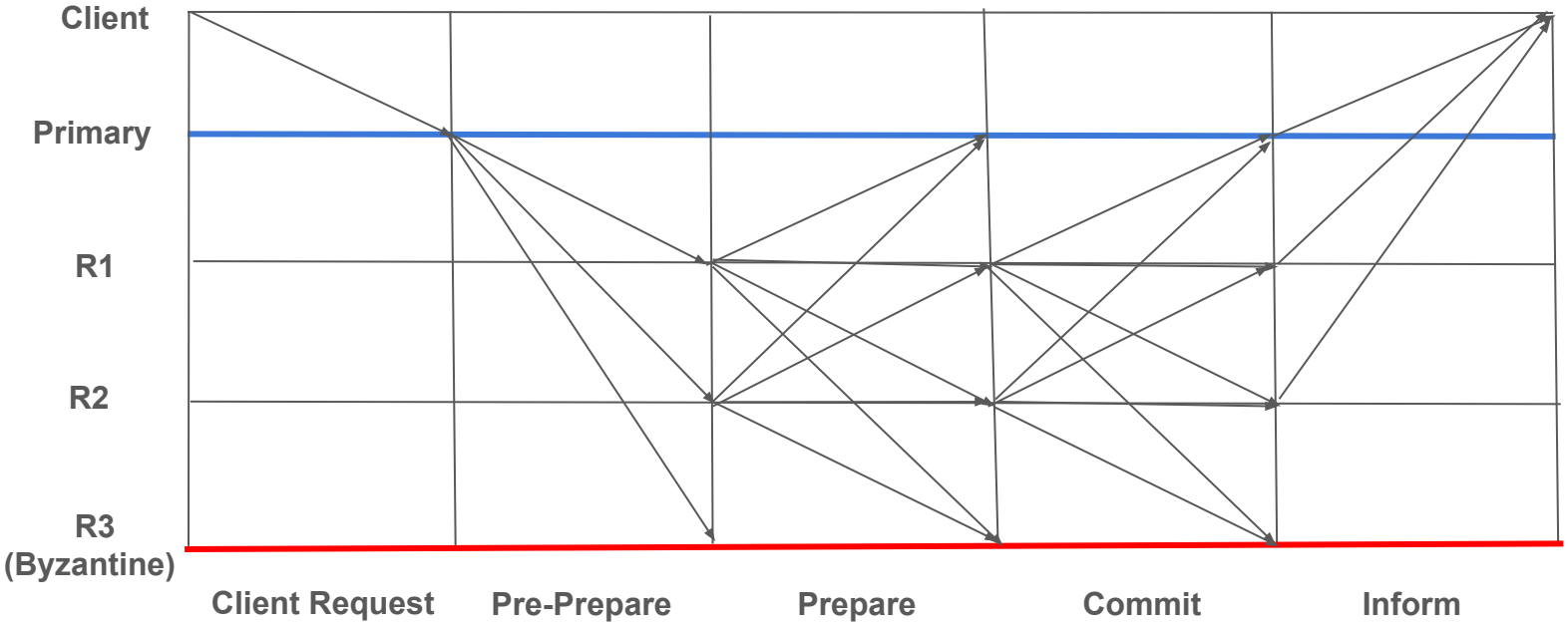
SpotLess: Concurrent Rotational Consensus Made Practical through Rapid View Synchronization

Dakai Kang, Sajjad Rahnama, Jelle Hellings, Mohammad Sadoghi

Presenters: Shray, Shreyas, Sayali, Himanshu & Amritanand

UCDAVIS

Practical Byzantine Fault-Tolerant Consensus

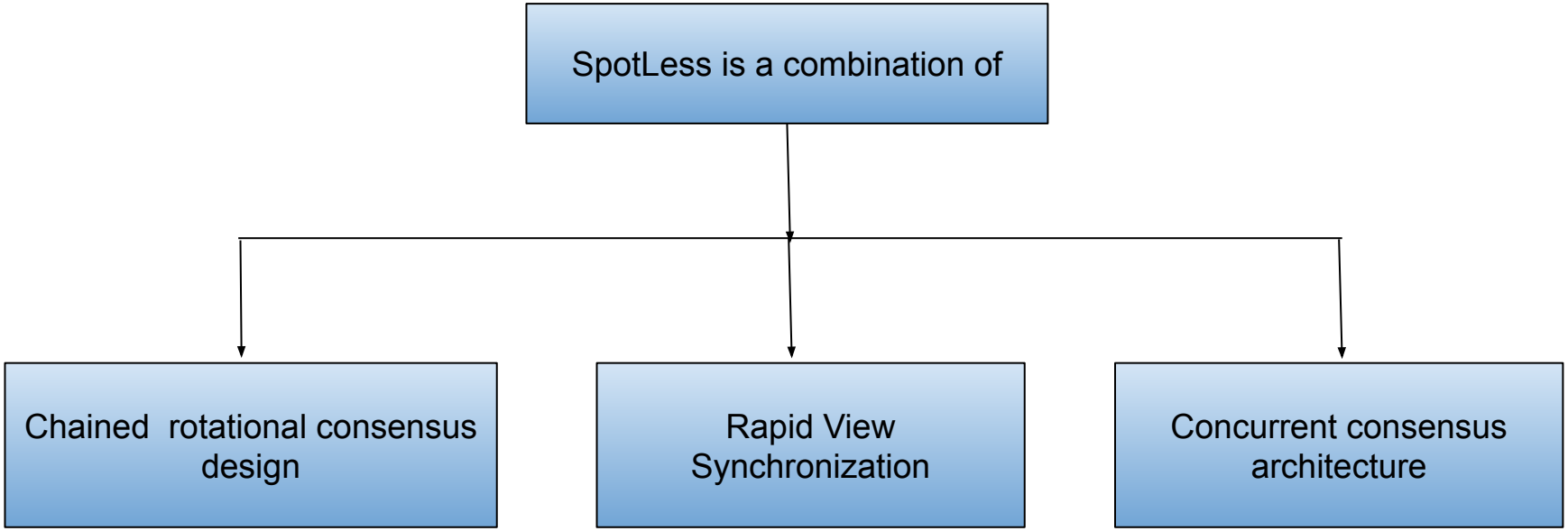


Limitations：High Message Complexity, Complex view-change protocols, Centralized primary bottleneck

Limitations with existing protocols

Problems with consensus protocols	Our Solution: Spotless
High Message Complexity	Chained rotational consensus design minimizes message exchanges
Complex & Error-prone view change protocols	Rapid View Synchronization allows nodes to synchronize independently without complex protocols
Sequential Processing Bottleneck	Concurrent Consensus Architecture enables parallel processing of multiple consensus instances

Our Solution: SpotLess



SpotLess : Key Features

Chained rotational consensus design

- Leadership rotates among the replicas → each replica gets a turn to act as the leader
- Avoids the need for to exchange messages for view-change protocol → Reduces communication costs and time
- If one leader fails → next replica in line takes over
- System doesn't need to go through a complex failure detection and recovery process

SpotLess : Key Features

Rapid View Synchronization Protocol

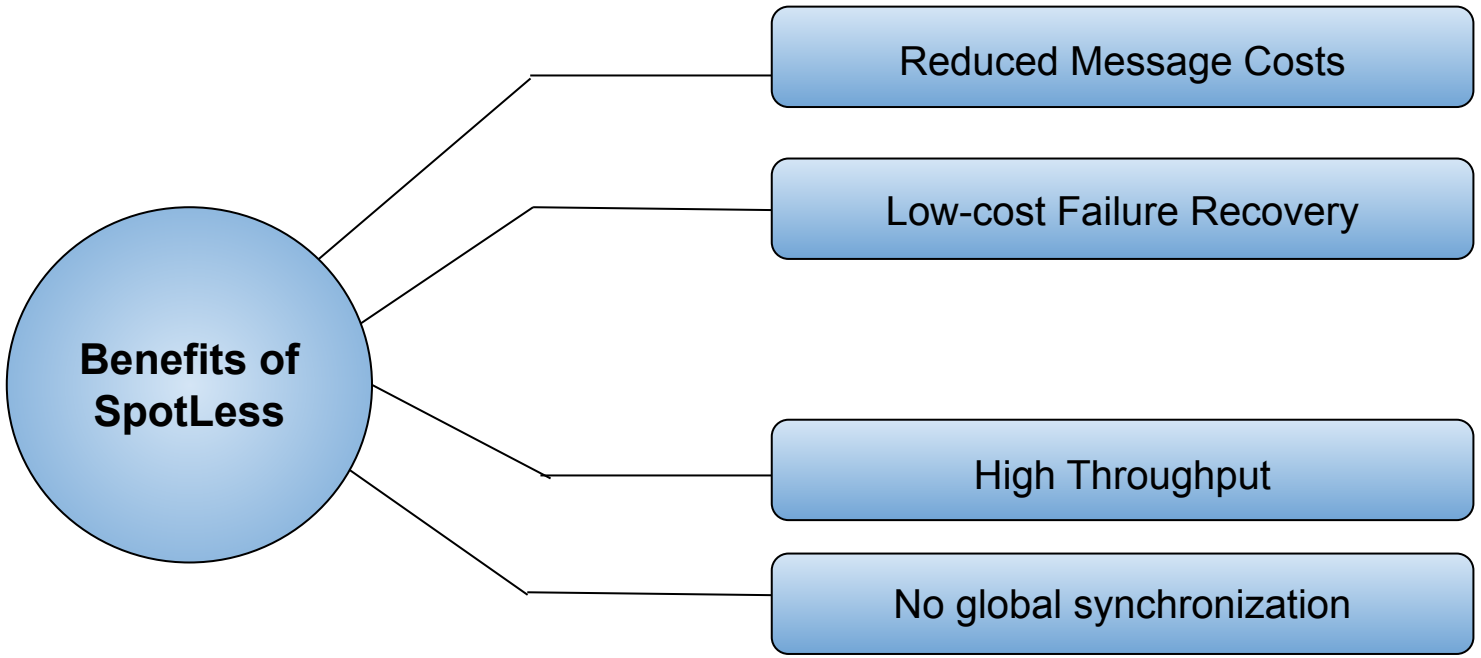
- Eliminates the need for replicas to be synchronized in time
- Replicas can independently decide the state of the system and handle failures
- Allows replicas to stay in sync regarding the system's state without relying on a clock

SpotLess : Key Features

Concurrent consensus architecture

- Allows different replicas to participate in multiple independent consensus instances concurrently
- Each leader can handle its own set of requests independently
- Avoids bottleneck and enables the system to process requests in parallel and improves throughput

SpotLess : Benefits



Preliminaries : Consensus

Three consensus guarantees:

01 **Termination**

If any non-faulty replica makes a decision on the ϱ-th client request, all non-faulty replicas will make the same decision on that request

02 **Non-Divergence**

All non-faulty replicas will agree on the same result for the ϱ-th client request

03 **Service**

When a non-faulty client requests execution, all non-faulty replicas will eventually fulfill that request

Preliminaries : Communication

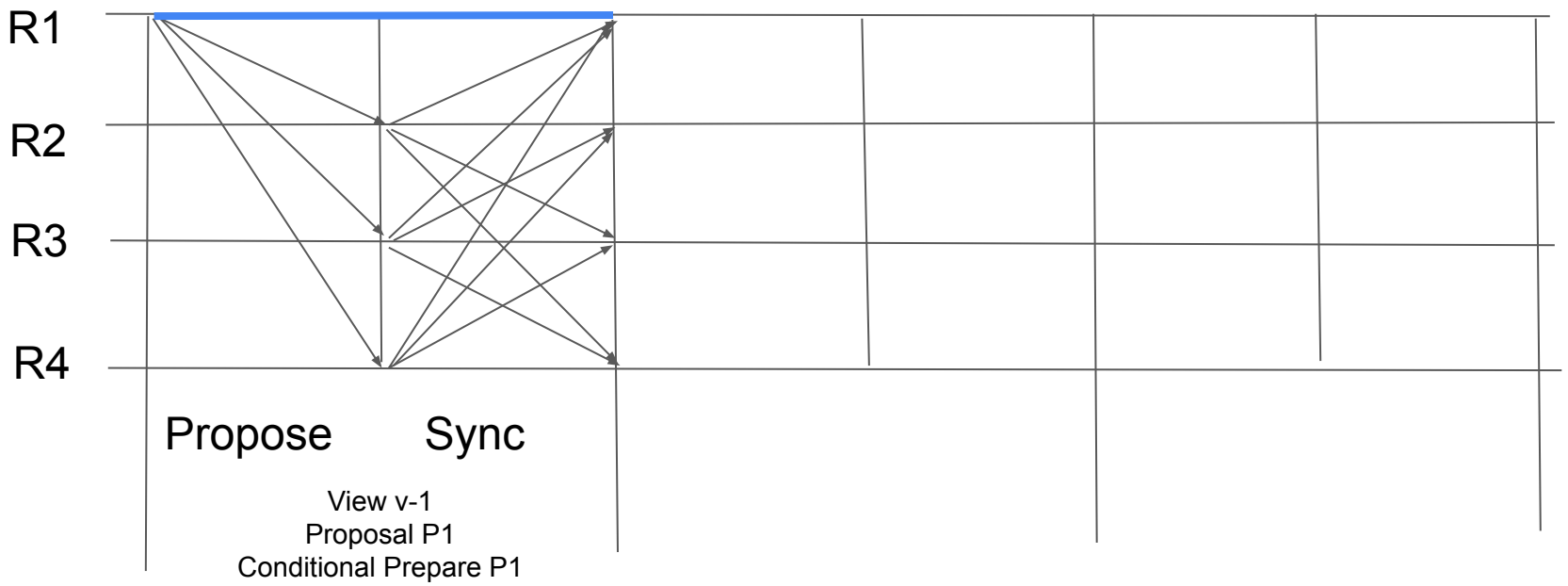
- Asynchronous communication is assumed → Messages can get lost or arbitrarily delayed
- Partially Synchronous Model of PBFT
- Safety (non-divergence) is always guaranteed
- Liveness (termination and service) is only guaranteed during periods of reliable communication
- Periods of unreliable communication are always followed by sufficiently-long periods of synchronous communication

Preliminaries : Authentication

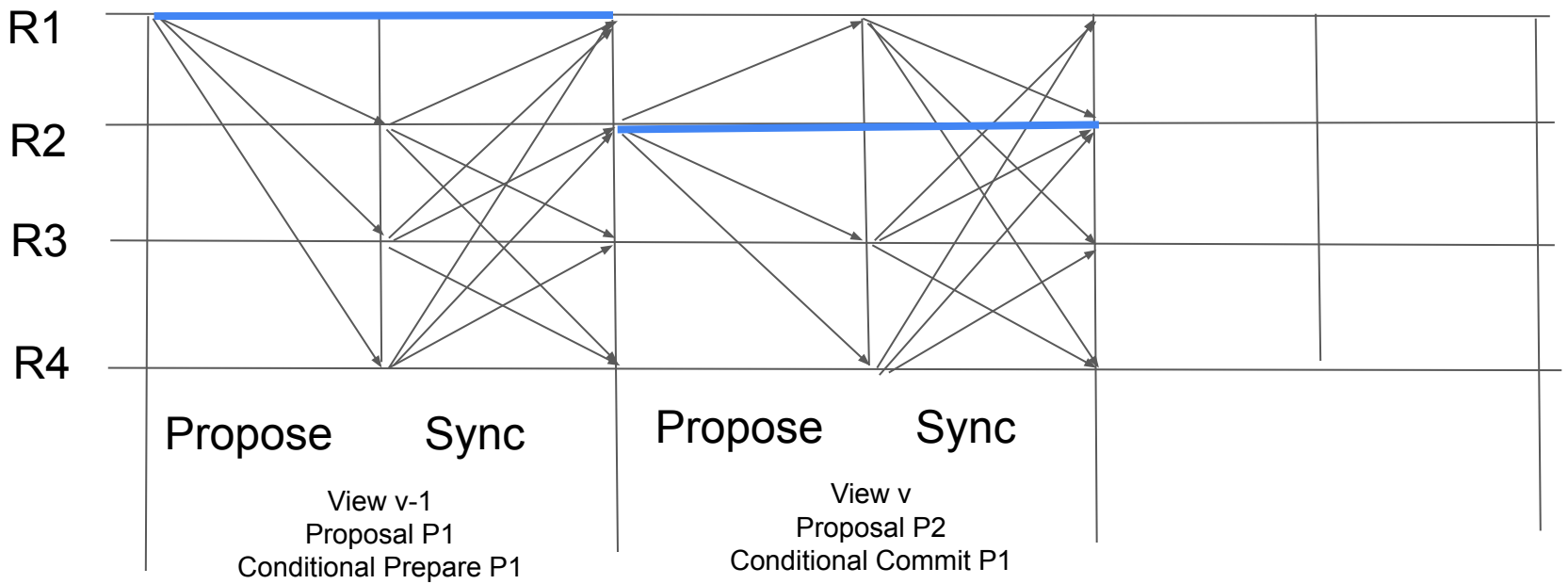
- Communication is authenticated → Faulty replicas can impersonate each other but they cannot impersonate a non-faulty replica
- Mechanisms for Enforcing Authentication

1. Message Authentication Codes (MACs) → Used for messages that are not forwarded.
2. Digital Signatures (DSs) → Used for messages that need to be forwarded.

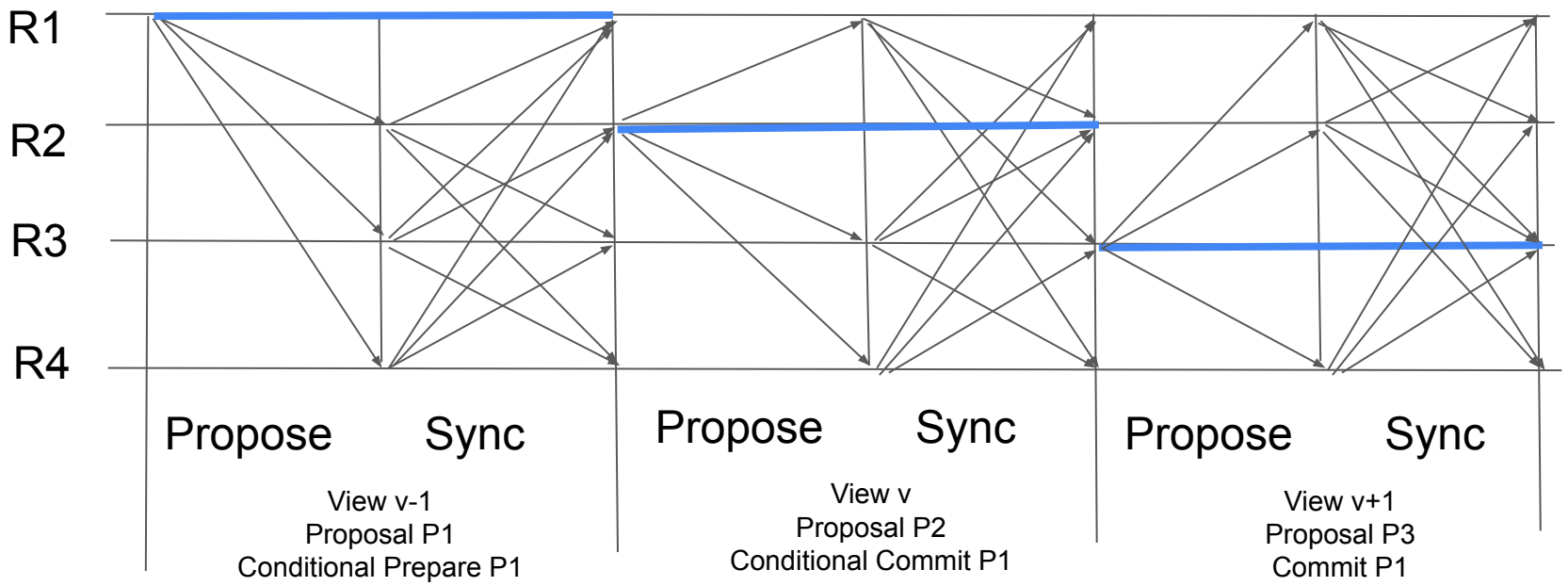
Design: Steps In Each View



Design: Steps In Each View

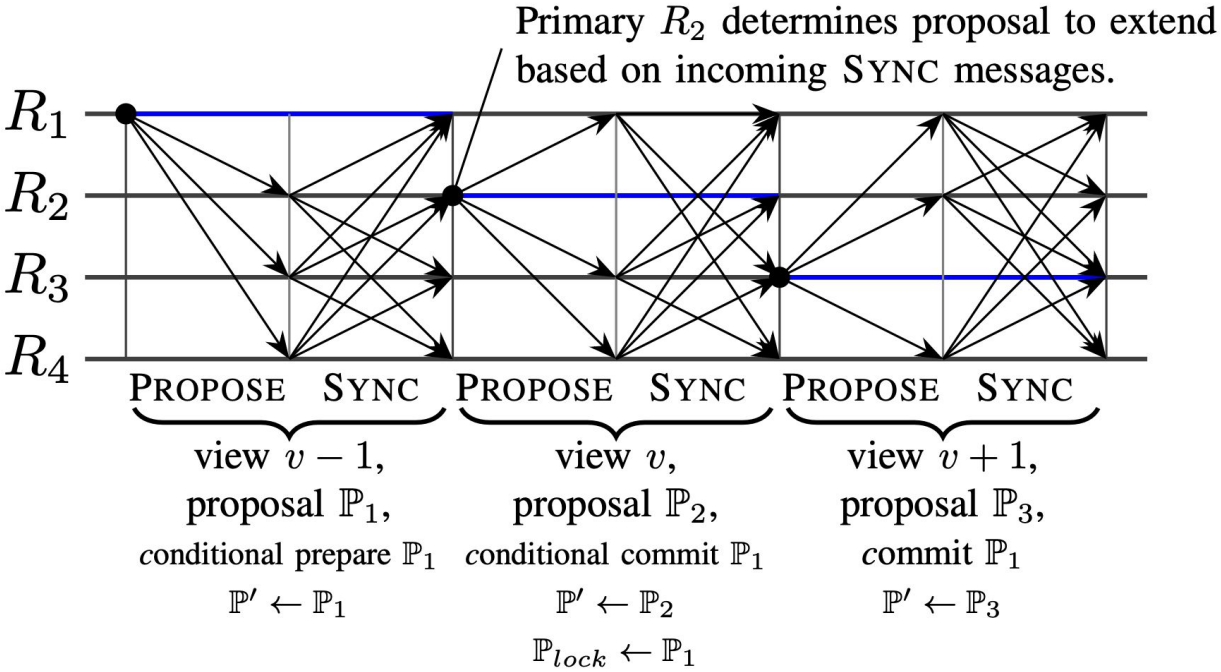


Design: Steps In Each View



Design: Steps In Each View

Each chained consensus instance of Spotless operates in views v ← 0,1,2,3,....



Step - 1: Propose

- Primary P finds the highest prior proposal, P′, with expected majority support from n-f replicas.
- P′ is the highest extendable proposal
- P selects a new client request, τ, and broadcasts a signed Propose message to all backups.
- Propose message format: PROPOSE(v, τ, cert(P′))
- Certificate for P′ + Primary's digital signatures ensures proposal authenticity and prevent forgery

How Do Replicas Verify The Proposal?

- R verifies Primary's digital signature
- R checks if client request, τ is valid
- R checks whether view v is the current view
- R checks whether certificate for P' is valid

Step - 2: Sync

- If R considers proposal P to be well-formed, it will broadcast the message msR := SYNC(v, claim(P), CP).
- CP is a set of pairs in the form of (view, digest) for the proposals that R has conditionally prepared.
- If R did not receive any valid proposals in view v, it will broadcast the message msR := SYNC(v, claim(∅), CP)
- Sync messages include MACs + Digital Signatures. MACs are always verified, Digital Signatures only verified during recovery.

Three-Phase Commit Algorithm

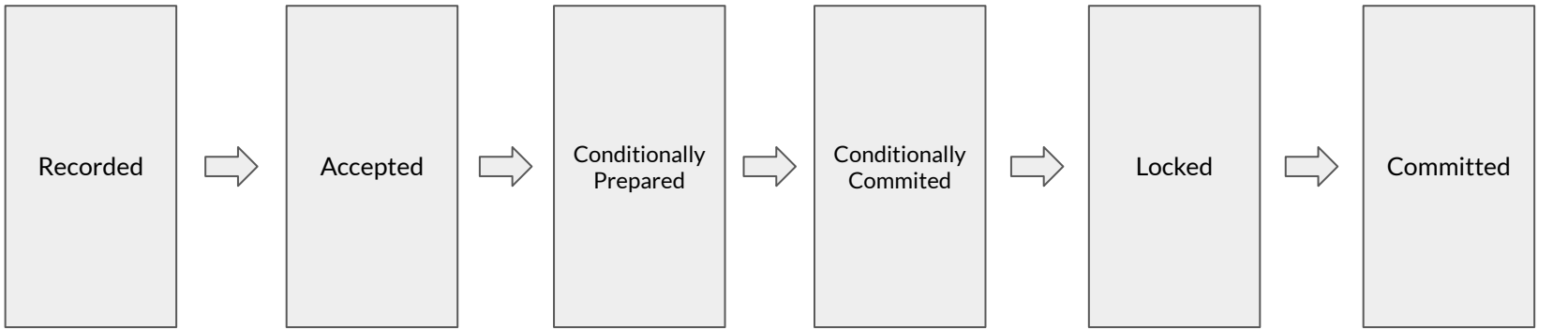
Each phase in the commit algorithm takes up one view:

- A successful first phase establishes a conditional prepare
- A successful second phase establishes a conditional commit
- A successful third phase achieves a commit that ensures the preservation of proposals across views

Key Definitions for Understanding Proposals

- Proposals are suggested changes that the system tries to reach consensus on.
- Each proposal is chained to its predecessor, ensuring continuity.
- $P = \text{PROPOSE}(v, \tau, \text{cert}(P'))$

Proposal States in SpotLess



Proposal P is verified and well formed

SYNC(v, claim(P), CP) is broadcast

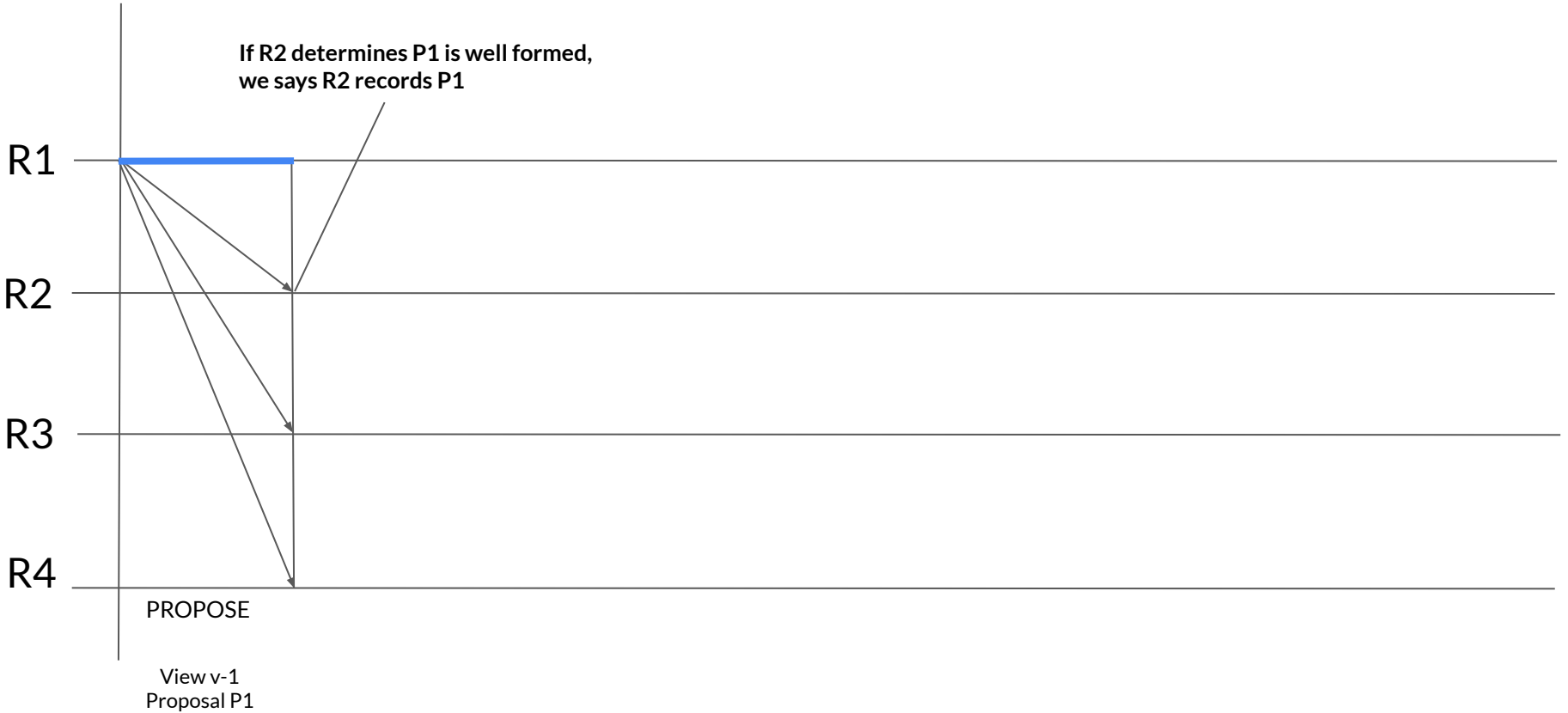
Replica receives n - f Sync messages SYNC(v, claim(P), CP) confirming support

In future view w, replica prepares proposal P' = PROPOSE(w, τ , cert(P)).

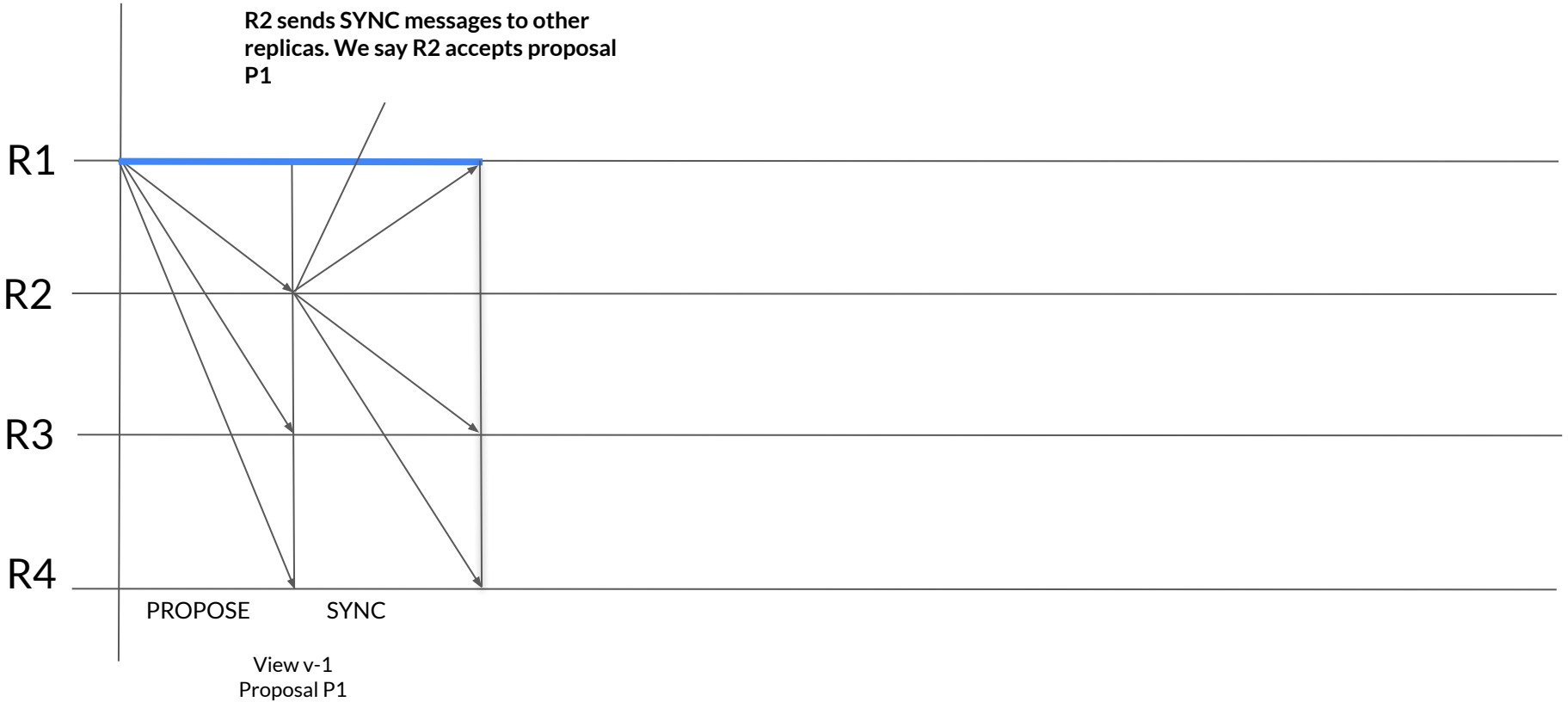
If p is highest proposal conditionally committed, the proposal is denoted by Plock.

In future view u, replica prepares proposal P'' = PROPOSE(u, τ, cert(P'))

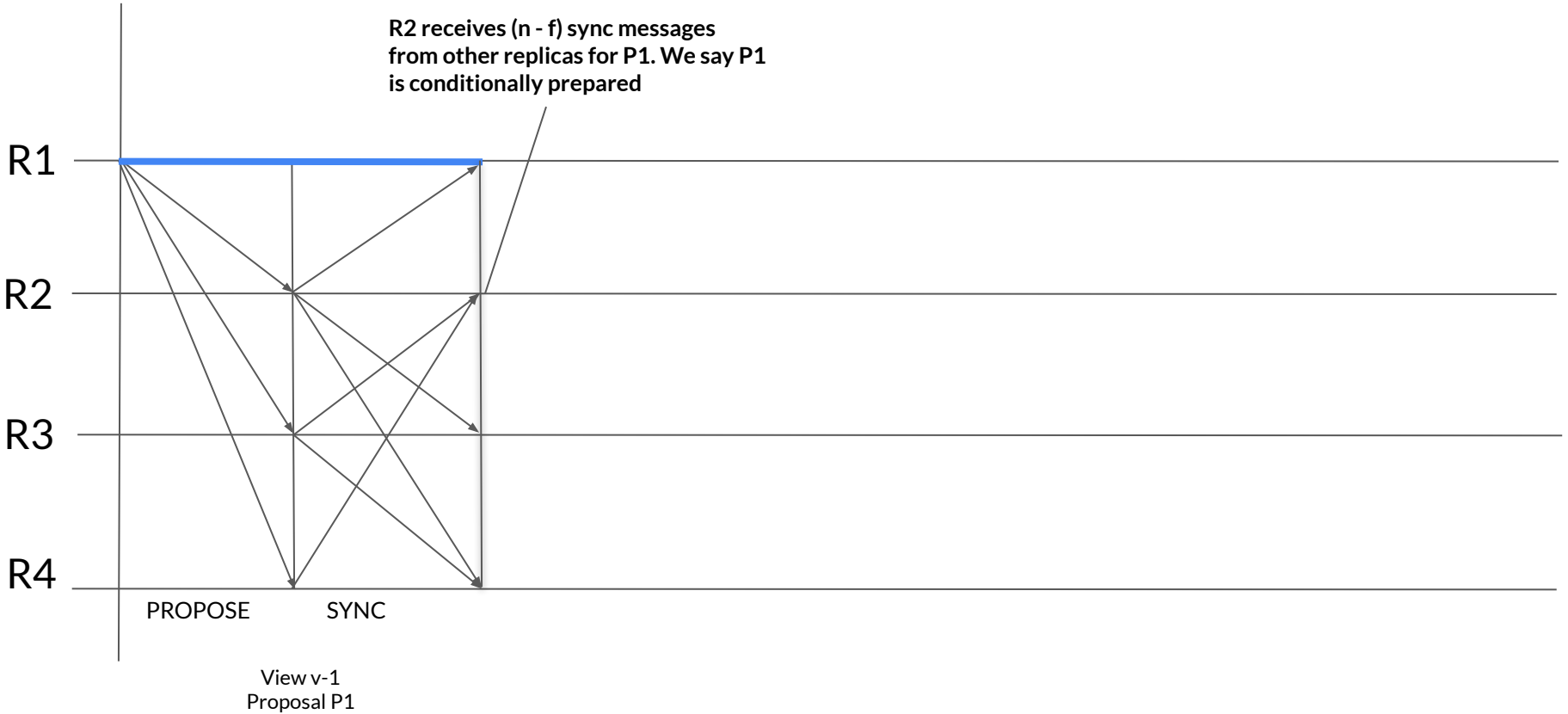
Proposal States in SpotLess



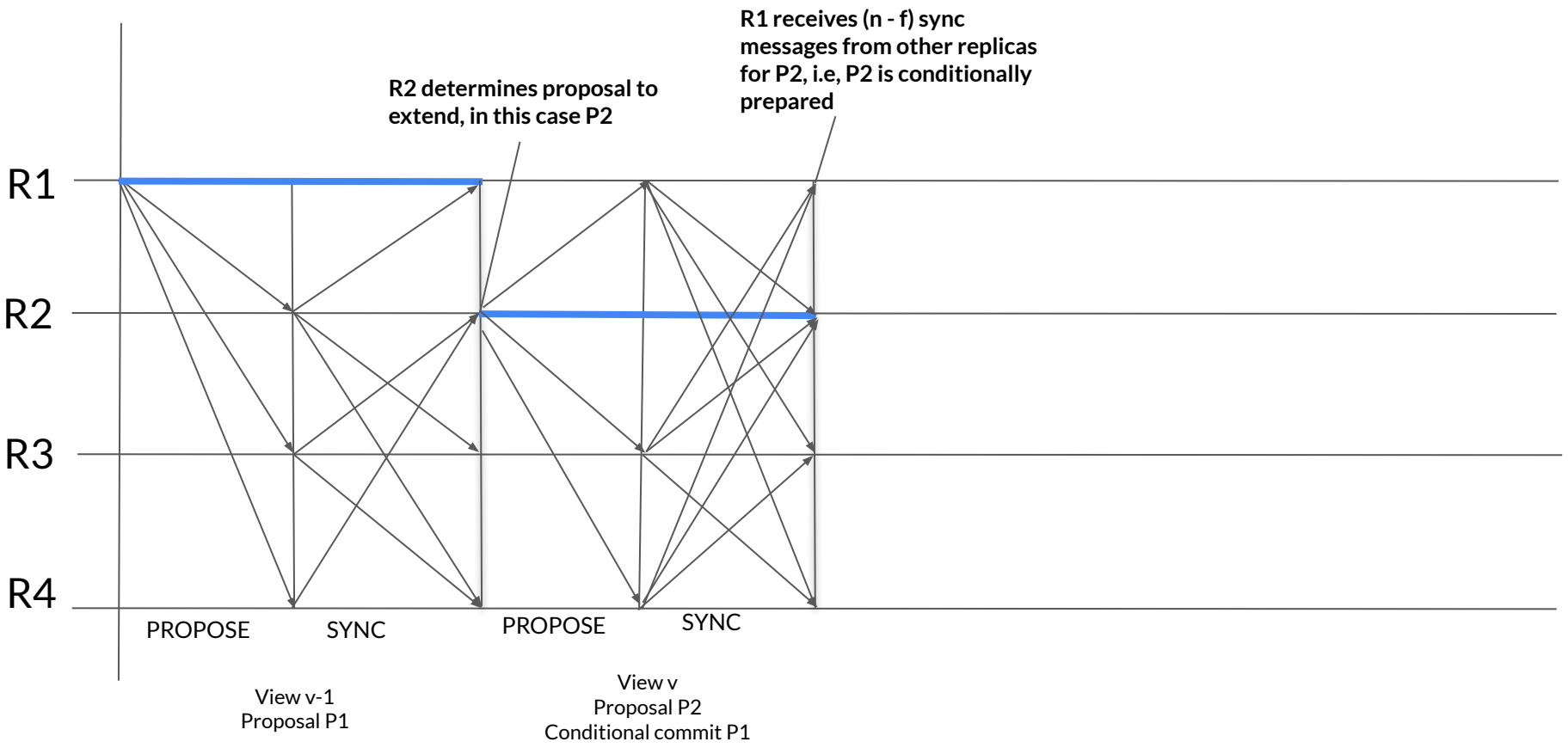
Proposal States in SpotLess



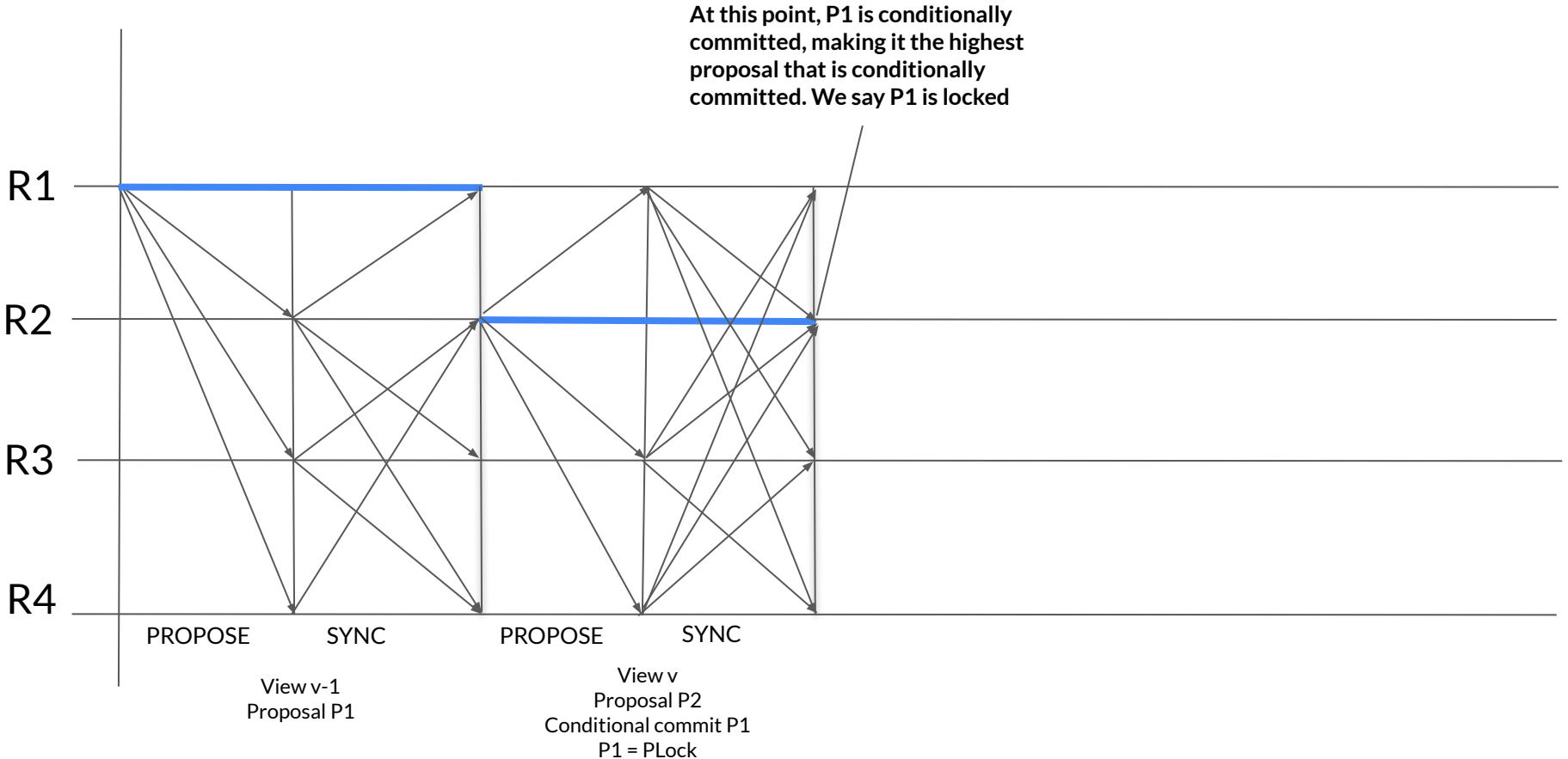
Proposal States in SpotLess



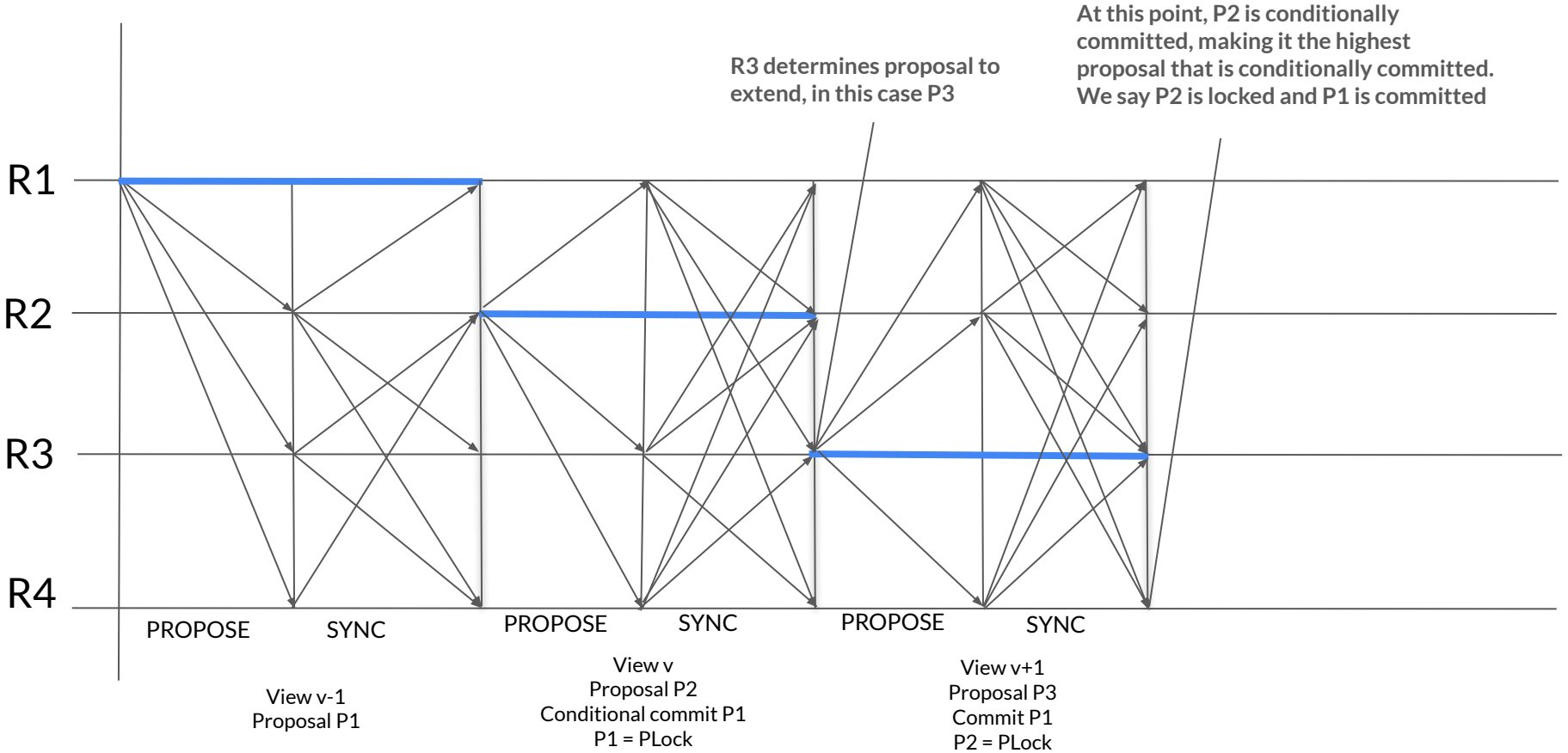
Proposal States in SpotLess



Proposal States in SpotLess



Proposal States in SpotLess



Conditions for Proposals to be Extended

A non-faulty primary can extend a proposal only if one of the following conditions is met:

- **E1**: The primary has a valid certificate for the preceding proposal
- **E2**: The primary has received SYNC messages from at least n - f replicas that claim to have conditionally prepared the proposal in a previous view.

Safety in SpotLess

- SpotLess uses safety rules to ensure no conflicting proposals.
- These rules help replicas make decisions based on valid, previously agreed proposals
- By following these rules, SpotLess prevents system divergence, ensuring that all non-faulty replicas reach the same conclusion.

The Three Rules Ensuring Safety of SpotLess

When a replica receives a new proposal $P := \text{PROPOSE}(v, \tau, \text{cert}(P'))$, it must decide whether to accept it based on the following rules:

- A1 (Validity Rule): Proposals must be well-formed and backed by $n-f$ votes.
- A2 (Safety Rule): Proposals must extend from the most recent locked proposal.
- A3 (Liveness Rule): Proposals must be in a view higher than the current locked proposal.
- Together, these rules ensure both safety (no conflicting proposals).

Dealing with Failures in View v

- If a replica doesn't receive any acceptable proposals from the primary in view v, it can still make progress by acting on **SYNC** messages from other replicas
- The replica can catch up by sending an **ASK** message to the replicas that sent the SYNC messages

Bootstrapping Liveness with Rapid View Synchronization

Liveness

The system can keep making progress, processing requests, and reaching agreements, even when there are delays or issues.

=

RVS

Liveness in SPOTLESS by making it easier for all the computers (called "replicas" or nodes) to stay in sync, even when there's unpredictable communication (called *asynchronous communication*).

Bootstrapping Liveness with Rapid View Synchronization

Rapid View Synchronization (RVS)

- RVS is central to SpotLess, enhancing resilience in asynchronous environments by synchronizing replica views to maintain system progress.
- **Goal**: Achieve consensus without requiring a global synchronization time.

Key Features of RVS

Quick Synchronization:

- RVS rapidly aligns views among replicas during periods of reliable communication, reducing latency.

State Recovery and Primary Rotation:

- Ensures low-cost recovery of system state and efficient rotation of primary roles when a primary fails.

Three-Phase View Process in RVS

- Each replica (node) waits to receive a proposal, labeled "P," that meets certain requirements.
- Timer: A timer (called tR) counts down. If the node does not receive a valid proposal within this time, it may move forward or retry.

- The node waits to receive a specific set of "SYNC" messages to ensure other nodes have accepted the current view (or version of the document).
- No timer, unlike ST1

- The node confirms that all replicas are on the same page with the proposal from ST1.
- The node has another timer here, which helps it determine when to proceed if some messages are missing.

Handling Delays and Catch-Up Mechanism

1. **Direct View Advancement**:

   Replicas can jump directly to higher views upon detecting a sufficient number of SYNC messages from replicas already in higher views.

2. **Retransmission for Synchronization**:

   Replicas in lower views request retransmission to catch up, ensuring alignment across views.

Lemma III.5 and Theorem III.6 (Proofs for Reliability)

- **Lemma III.5**: If a node (R) receives enough SYNC messages to confirm a proposal "P," then R will eventually learn about all previous proposals leading up to "P." This ensures that nodes won't miss important past agreements.

- **Theorem III.6**: All non-faulty nodes (or replicas) will eventually have the same record of the sequence of proposals once the network is stable. This guarantees that the system remains consistent.

Mechanism Guaranteeing Liveness in SPOTLESS Protocol

- SPOTLESS ensures **liveness**, meaning all non-faulty replicas continue to make progress even in unreliable networks having delays or failures.

- The system needs to ensure that all replicas (nodes) can continue operating, eventually agreeing on proposal.

Backup Role and State Transitions

Backup Role (Replica R) - Key Steps: RVS allows replicas to figure out in which view they should operate based on the messages they receive.

- ST1 (Recording): Wait for a proposal or timeout, then broadcast SYNC message.
- ST2 (Syncing): Wait for n − f SYNC messages to move to Certifying.
- ST3 (Certifying): Set a timer and wait for SYNC messages to confirm the proposal or move to the next view. If there are delays, RVS ensures replicas can request retransmissions using the flag Y and ASK messages, allowing them to catch up on missed views.

Key Liveness Challenges

Liveness Challenge 1: Transition from Syncing (ST2) to Certifying (ST3)

- Replica R needs n − f SYNC messages to move from Syncing (ST2) to Certifying (ST3).
- Without these messages, progress halts.

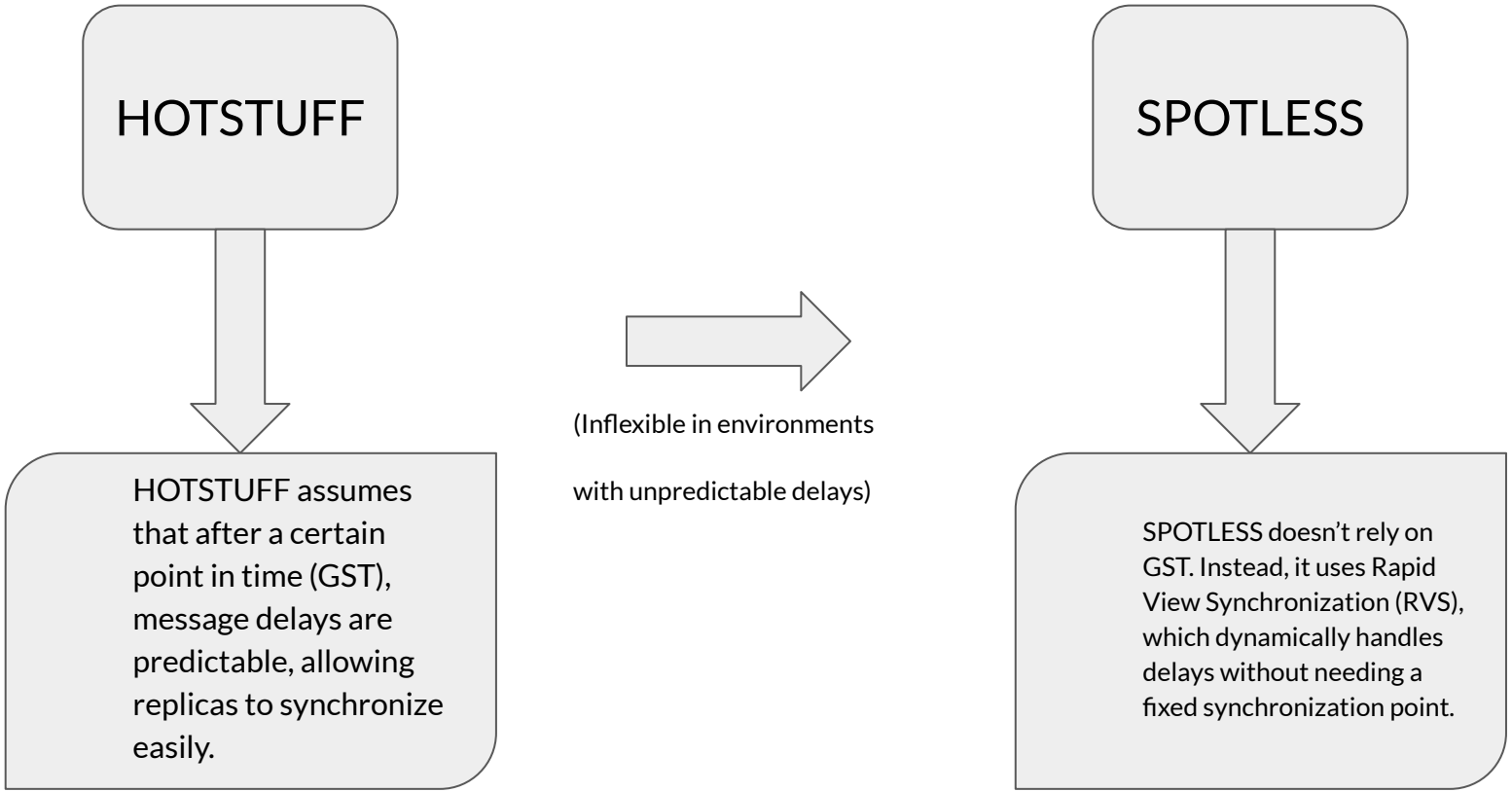
Liveness Challenge 2: Catching Up on Proposals

- Replica R must get responses from f + 1 other replicas to learn about missing proposals.
- Responses include flag Y, asking other replicas to retransmit their SYNC messages.

Liveness Challenge 3: Receiving Proposals from the Primary

- If R doesn't receive a proposal from the primary, it must rely on ASK messages to request the missing proposal from other replicas.

Comparison with HOTSTUFF (GST)



Dealing with Communication Delays

Timeout Adjustments:

- Traditional protocols use **exponential backoff**, where waiting time doubles after each missed message.
- SPOTLESS increases timeouts **moderately** with a small constant ε, avoiding long delays due to exponential backoff.
- **Timeout halving** if expected messages arrive earlier than anticipated.
- No network overload and less wait times.

THANK YOU

UCDAVIS