UCDAVIS

Proof-of-Execution: Reaching Consensus through Fault-Tolerant Speculation

EDBT ‘21 Suyash Gupta, Jelle Hellings, Sajjad Rahnama, Mohammad Sadoghi

Presenters

Harish Krishnakumar | Bismanpal Singh Anand | Krishna Karthik | Georgy Zaets

Fall Quarter 2024
ECS 265 | Distributed Database Systems
University of California, Davis

INTRODUCTION

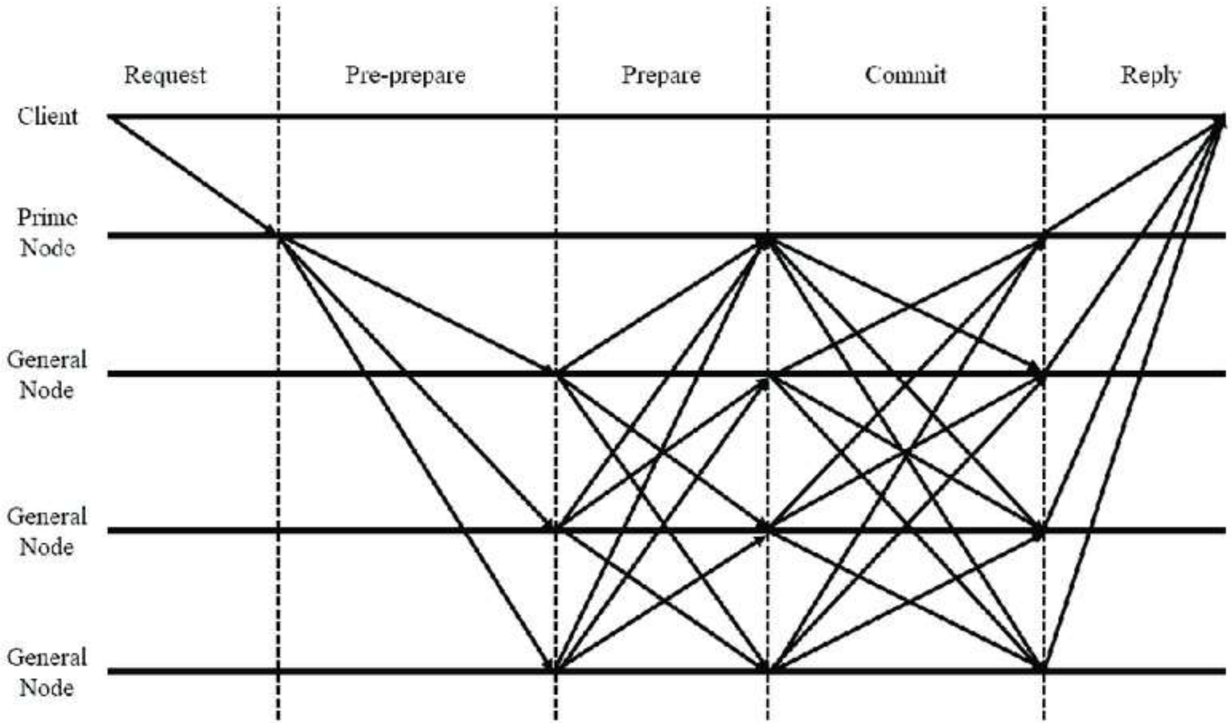
Byzantine Fault Tolerance (BFT):

- Distributed systems often face **Byzantine failures**, where some nodes can fail or act maliciously.
- BFT consensus protocols allow **non-faulty replicas** to agree on a consistent state, even with faulty nodes.

Practical Byzantine Fault Tolerance (PBFT):

- **PBFT** is a popular BFT protocol that operates in **three phases** (pre-prepare, prepare, and commit).
- Pbft ensures **strong safety guarantees**, meaning all non-faulty replicas agree on the order of operations.
- **Communication Complexity**: Pbft has a **quadratic communication overhead ($O(n^2)$)**, which makes it inefficient as the system scales.

PBFT Phases


Request
Pre-prepare
Prepare
Commit
Reply
Client
Prime Node
General Node
General Node
General Node

Issues in PBFT

Communication Overhead:

- PBFT requires **all-to-all communication** between replicas in both the prepare and commit phases, which leads to high **latency** and **low throughput** in large systems.

Three Communication Phases:

- PBFT uses **three full phases** to ensure safety and correctness. Each phase involves significant message exchanges, which increases delay, especially in large networks.

Scalability Problem:

- As the number of replicas increases, PBFT's quadratic communication costs become impractical for large-scale, high-throughput systems.

Introduction to Proof-of-Execution (PoE)

Proof of Execution (PoE)

- A novel BFT protocol that achieves resilient agreement in just three linear phases. The paper portrays PoE as a scalable and reliable agreement protocol that shields against malicious attacks. PoE’s scalable and resilient design emerges by adding four design elements to PBFT.

Challenges with Existing Protocols:

- As already mentioned, PBFT operates in three communication phases, two of which necessitate quadratic communication complexity.
- PBFT is considered unrealistic in large scale data management systems due to higher complexity.
- PBFT involves high computational power.

UC DAVIS

Key Design Innovations of PoE

- **Speculative Execution :**
  - Replicas execute requests **immediately after the prepare phase**, bypassing the commit phase.
  - This **reduces communication overhead**, leading to faster processing and lower latency in response times.

- **Proof-of-Execution :**
  - Clients must receive **2f + 1 identical responses** from replicas to confirm the execution of their request.
  - This approach provides **stronger safety guarantees** compared to Pbft's requirement of f + 1 responses, ensuring a higher level of agreement among non-faulty replicas.

Key Design Innovations of PoE

- **Safe Rollbacks :**
  - PoE allows replicas to **rollback speculative transactions** if the required consensus is not achieved.
  - This mechanism ensures that the system can **recover from execution errors** while maintaining data integrity and consistency.

- **Agnostic Signatures :**
  - PoE is designed to use **Message Authentication Codes (MACs)** for smaller setups and **Threshold Signatures (TSs)** for larger systems.
  - This flexibility in cryptography allows PoE to adapt to various network sizes, ensuring **efficient communication** and scalability.

Comparison with Other BFT Protocols

Zyzzyva:

- Utilizes a **twin-path model** with an optimistic fast path, but relies on **client-dependent slow path recovery**, adding complexity and error-prone recovery.

SBFT:

- Similar to Zyzzyva with a fast path, but requires **external entities (collectors)** to aggregate responses, increasing overhead.

HotStuff:

- Involves **8 communication phases** for primary rotation and threshold signatures, but enforces strict **sequential processing**.

PoE's Advantage:

- PoE operates efficiently without the need for external entities or client involvement, and with **fewer communication phases** than most other protocols.

Comparison Table of BFT Protocols

Protocol	Phases	Messages	Resilience	Requirements
Zyzzyva	1	$\mathcal{O}(\mathbf{n})$	0	Reliable clients and unsafe
PoE (our paper)	3	$\mathcal{O}(3\mathbf{n})$	f	Sign. agnostic
Pbft	3	$\mathcal{O}(\mathbf{n} + 2\mathbf{n}^2)$	f	
HotStuff	8	$\mathcal{O}(8\mathbf{n})$	f	Sequential Consensus
SBFT	5	$\mathcal{O}(5\mathbf{n})$	0	Optimistic path

Out-of-Order Processing

Normal Systems:

- Traditional BFT systems process requests in the order they arrive, which can create delays, especially if some requests take longer to complete.

How PoE Works:

- PoE allows replicas to handle multiple requests at the same time, even if they come in at different times.
- It uses an **active window** and **watermarks** to keep track of what's being processed.

Benefit:

- This approach improves throughput and efficiency, making PoE better at managing requests, even when there are delays.

Out-of-Order Execution

Normal Systems:

- In many BFT systems, replicas wait for a complete order before executing requests.
- This can slow things down because they have to wait for everyone to agree.

How PoE Works:

- In PoE, replicas can start executing requests right after they get partial agreement.
- This means they don’t have to wait for everyone to agree fully, making processing **faster**.

Benefit:

- Out-of-order execution allows for quicker responses to clients, improving overall system **performance**.

Twin-Path Consensus

Normal Systems:

- In protocols like Zyzzyva, there are two paths: a fast path for quick execution and a slow path for recovery if something goes wrong.
- If the fast path fails, the system has to switch to a more complicated and slower method, which can cause errors.

How PoE Works:

- PoE avoids using a slow recovery path. Instead, it allows replicas to execute requests based on early agreement.
- If a request fails, PoE can handle it without needing to switch paths, keeping things simple.

Benefit:

- This makes PoE more straightforward and efficient, as it reduces the chances of running into problems during execution.

Primary Rotation

Normal Systems:

- In some BFT systems, the primary (leader) replica changes after every decision to prevent one node from controlling the process.
- This rotation adds more communication steps, making it slower.

How PoE Works:

- PoE can operate without strict primary rotation. It still manages leadership but allows for more flexibility.
- This means it can handle requests efficiently without needing to constantly change the primary.

Benefit:

- By not relying on frequent primary changes, PoE improves overall speed and adaptability in decision-making.

PROOF OF EXECUTION

System Model And Notations

- **Replicas Set ($\Re$) -> Set of replicas processing client requests.**
- **Identification: Each replica $r \in R$ is assigned a unique identifier id(r) with $0 \leq id(r) < |R|$.**
- **$F \subseteq R$: Byzantine (faulty) replicas.**
- **$R \setminus F$: Non-faulty replicas.**
- ***n (|R\ )*: Total number of replicas.**
- ***f (|F|)*: Number of faulty replicas.**
- **$nf = n - f$: Number of non-faulty replicas.**
- **We assume that $n > 3f$ ($nf > 2f$).**

Authenticated Communication

Authenticated Communication is necessary when there are Byzantine Failures.

So what do we use?

- ➔ **Message Authentication Codes (MACs):**
- ➔ Based on symmetric cryptography.
- ➔ Each pair of replicas shares a secret key.
- ➔ Non-faulty replicas must keep their secret keys hidden.

- ➔ **Threshold Signatures (TSs):**
- ➔ Based on asymmetric cryptography.
- ➔ Each replica holds a distinct private key to create a signature share.
- ➔ A valid threshold signature requires at least nf distinct signature shares.

Consensus Guarantees in PoE

Requirements of PoE

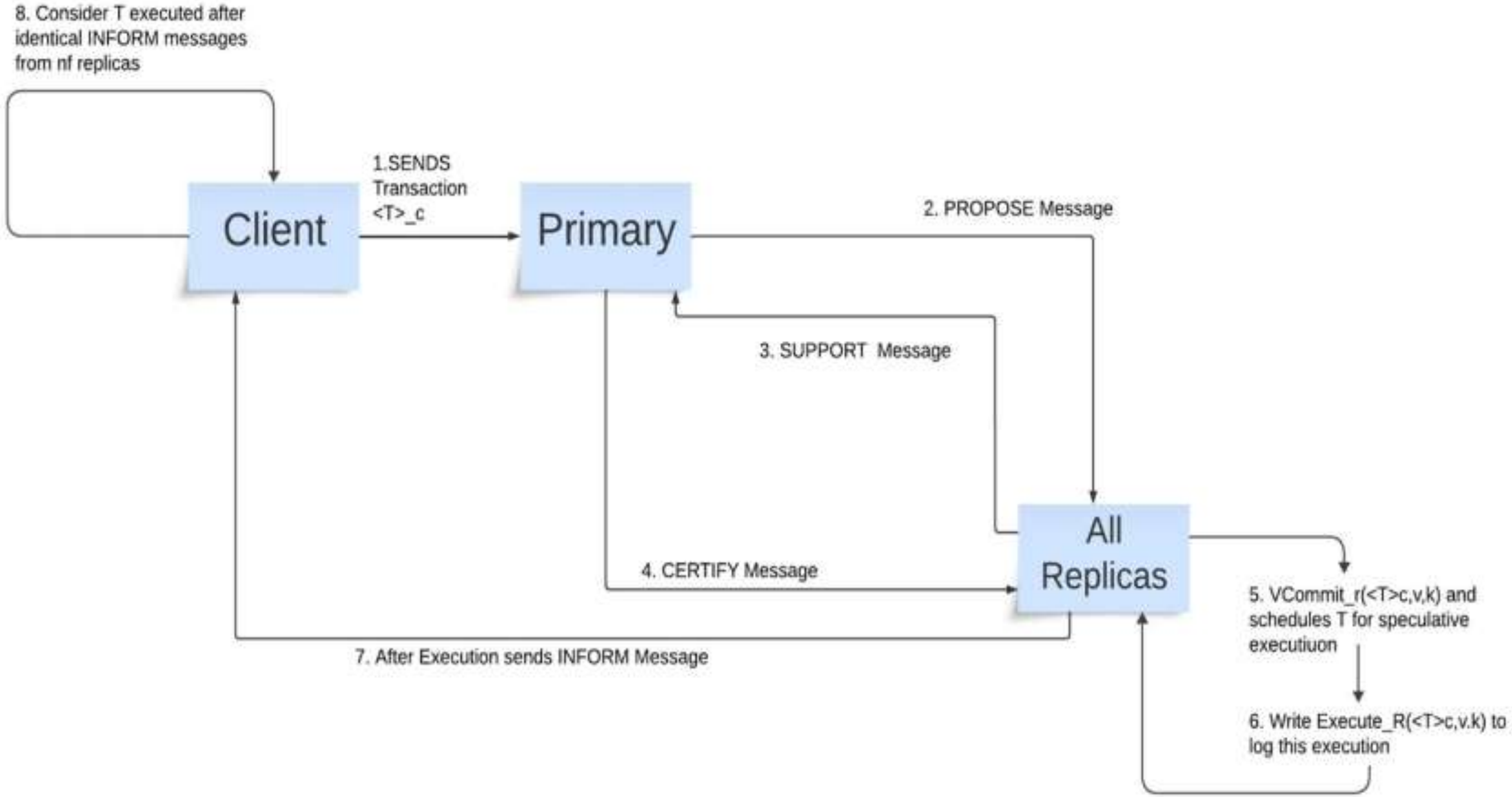
- **Termination (Liveness):**
  Every non-faulty replica executes a transaction.

- **Non-Divergence (Safety):**
  All non-faulty replicas execute the same transaction.

- **Speculative Non-Divergence:**
  If $nf - f \geq f + 1$ non-faulty replicas accept and execute the same transaction T, then all non-faulty replicas will eventually accept and execute T

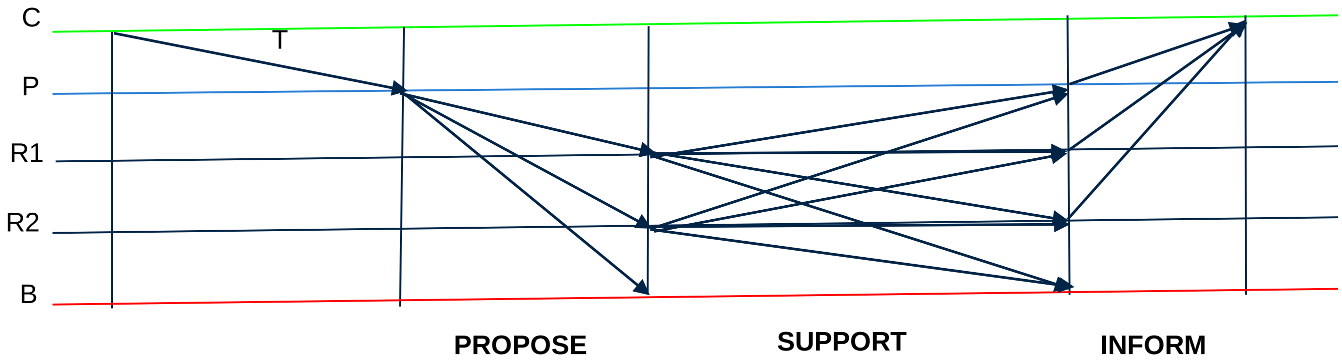
UC DAVIS

Normal Case Algorithm Flowchart (Using TS)



PoE using MAC's

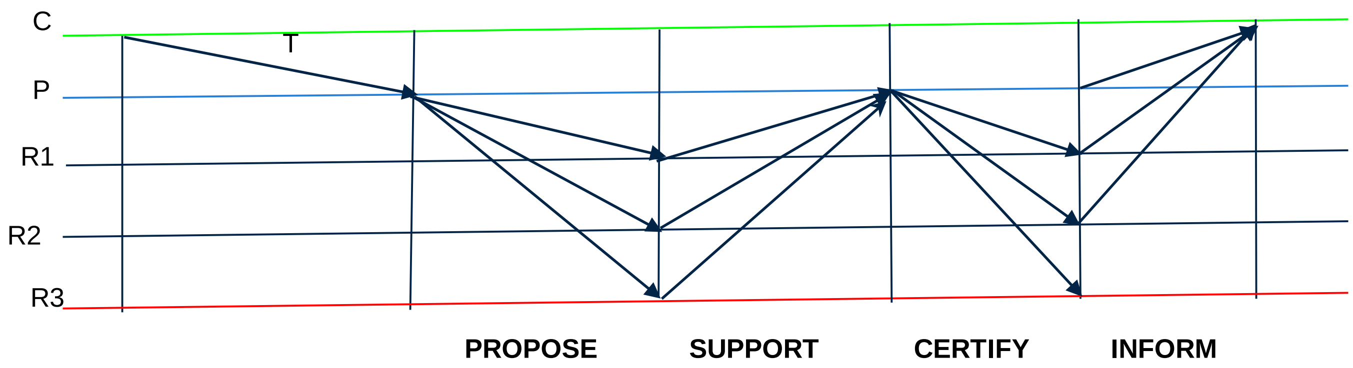
- MAC's reduce computational complexity of PoE
- Overall communication cost increases
- **SUPPORT** and **CERTIFY** phases are replaced by a single all-to-all **SUPPORT** phase



PoE using TS

PoE uses Threshold Signatures with a 3-phase linear communication process

- ➔ **Propose**: The primary proposes a transaction to all replicas.
- ➔ **Support**: Replicas send support messages with signature shares back to the primary.
- ➔ **Certify**: The primary aggregates the signature shares and sends a certify message to all replicas.



View Change

View Change Algorithm - Motivation

A process that elects a new primary when the current one fails, ensuring the system can recover.

View – The state that identifies the current primary process.

When does View Change happen?

- Primary failure (malicious or unresponsive).
- Timeout (no response from the primary).

Why is View Change needed?

To maintain system correctness and progress.

View Change Algorithm - Terminology

View Number:

- Ensures the system knows which primary is in charge and that all replicas are aligned with the correct primary.

Last Stable Checkpoint:

- Ensures that after a view change, no prepared requests are lost and guarantees that they are finalized correctly.

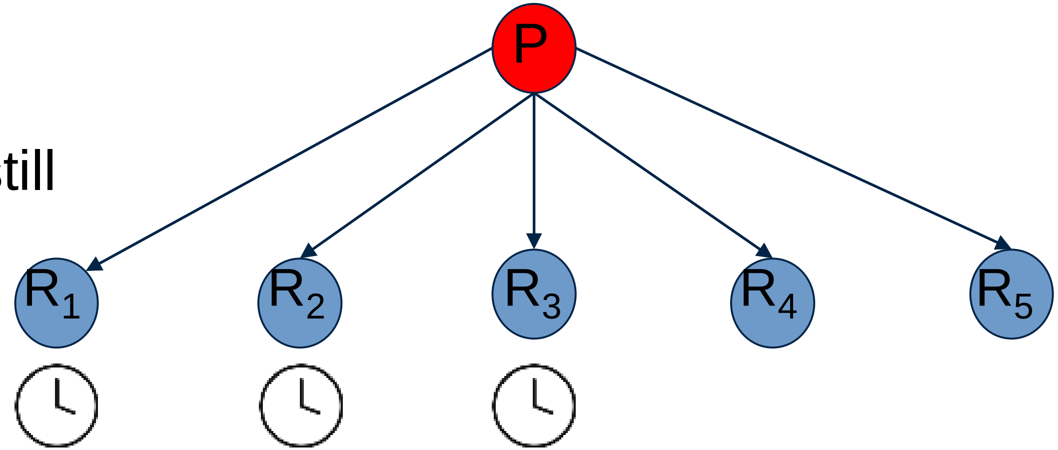
Log of Prepared Requests:

- Ensures that any requests that were in progress are finalized, maintaining consistency across all replicas and ensuring no request is lost.

View Change in PBFT, Detection using Timer

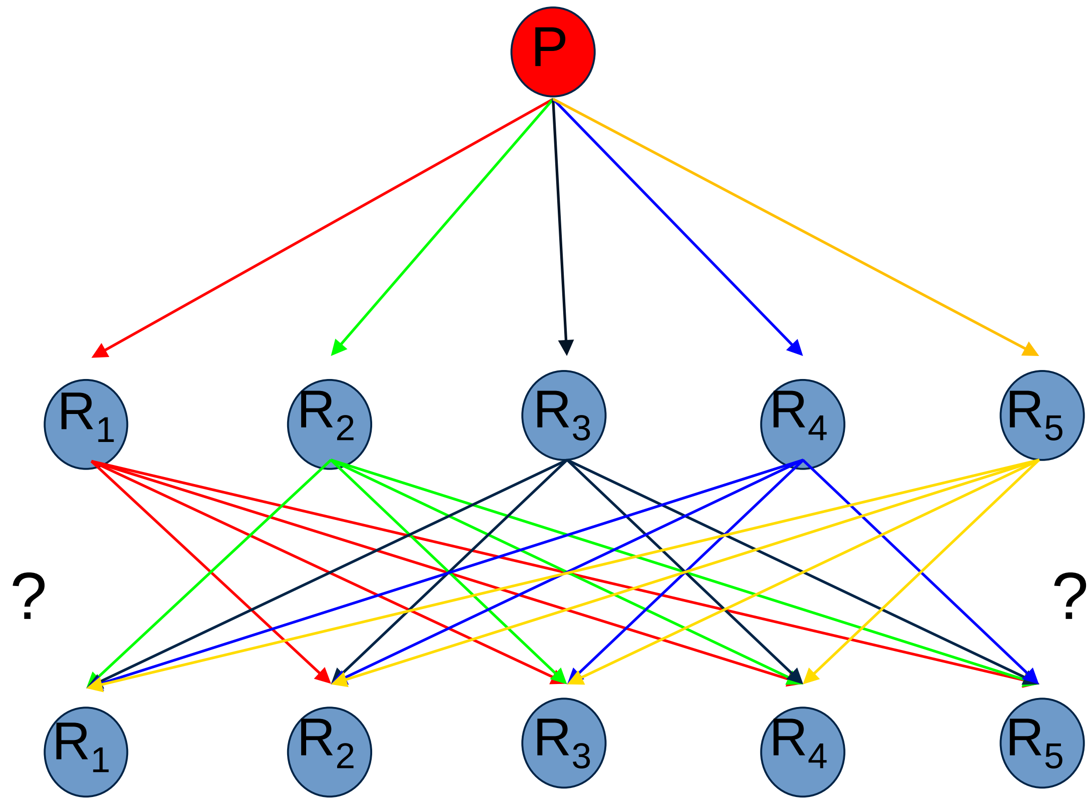
Each replica has a timeout mechanism.

In this case:
$R_1 – R_3$ know P is faulty
R_4 & R_5 are waiting for P still



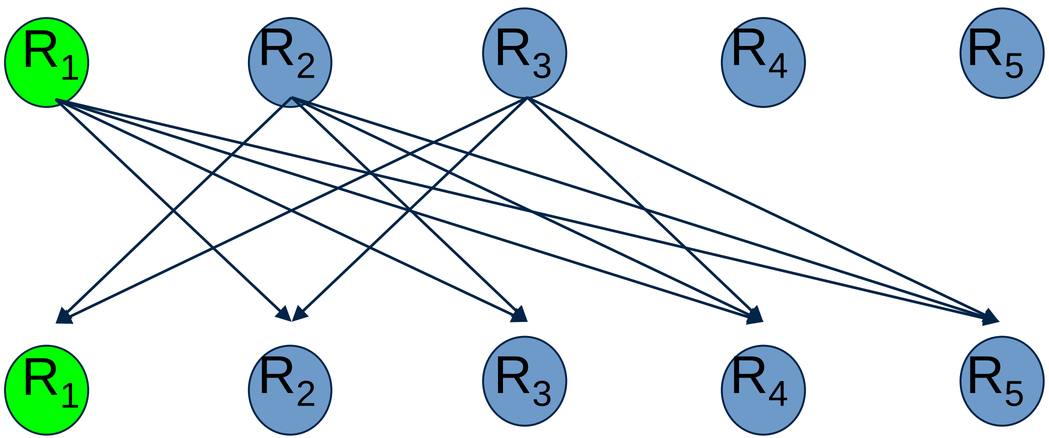
Replicas detect the primary is faulty because they don’t receive Pre-Prepare messages within a set timeout period.

View Change in PBFT, Detection in Prepare



Initiate View Change in PBFT

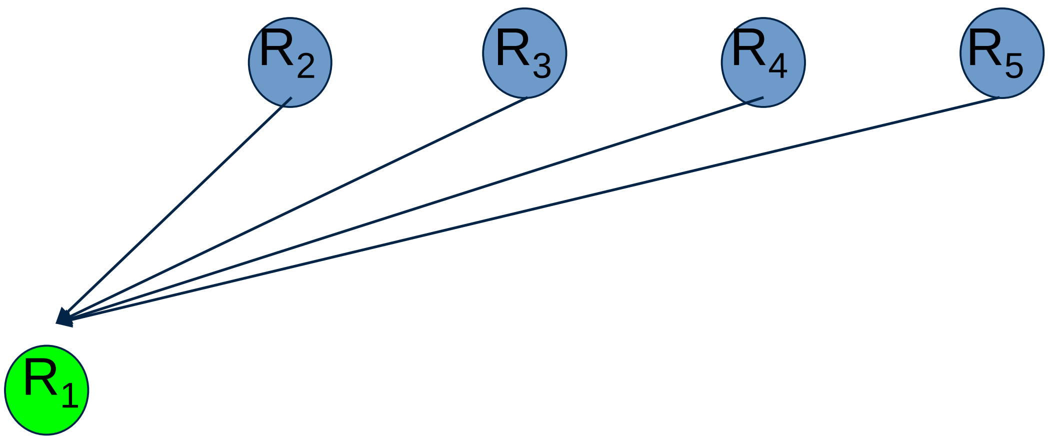
When replicas detect a faulty primary, View Change starts.
They send View-Change messages to other replicas.
R4 and R5 now know R1 is becoming the new primary.



Message Contents: View number, last stable checkpoint, log of prepared requests.

New Primary Collects View-Change Messages

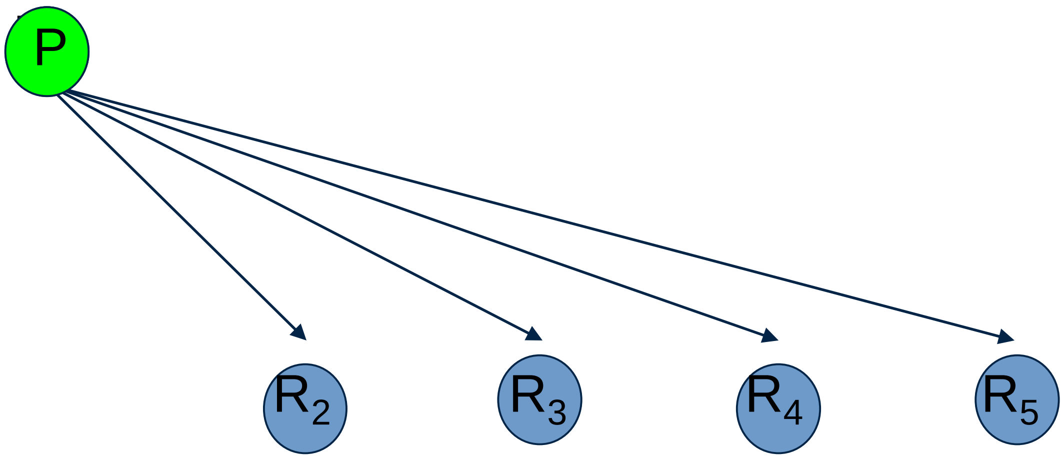
The new primary, R1, collects ***2f+1*** valid View-Change messages, confirming the prepared requests and the new view.



R2-R5 know they have sent their view-change messages to R1.

New-View Announcement by New Primary

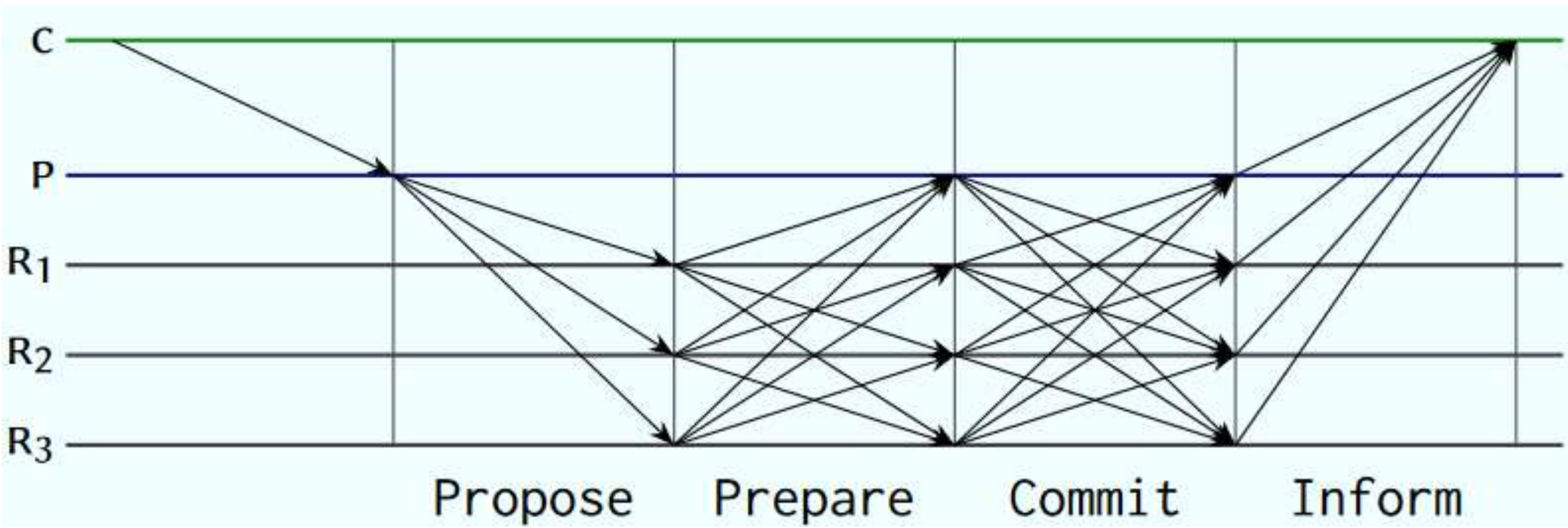
R1, the new primary, broadcasts the New-View message to confirm new view is established and synchronizes all replicas.



Are We Finished With View Change?

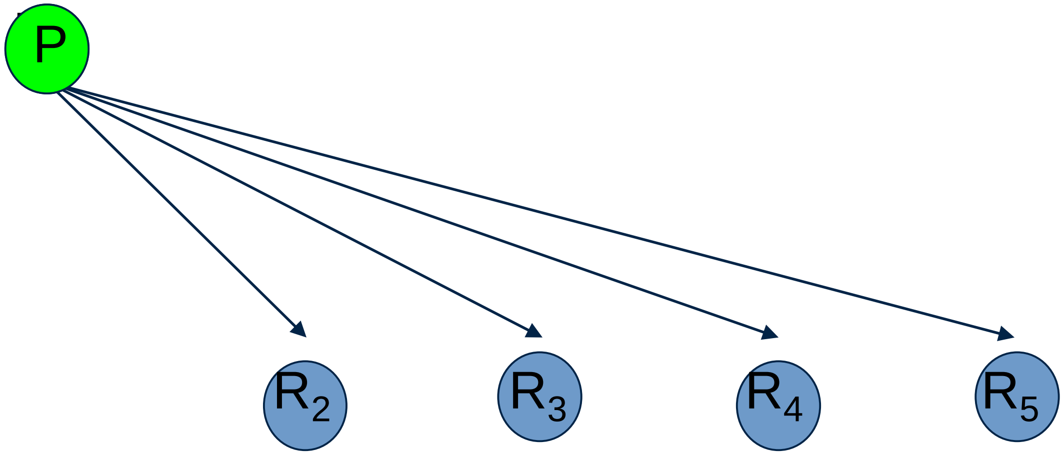
NO!
But Why?

Remembering PBFT



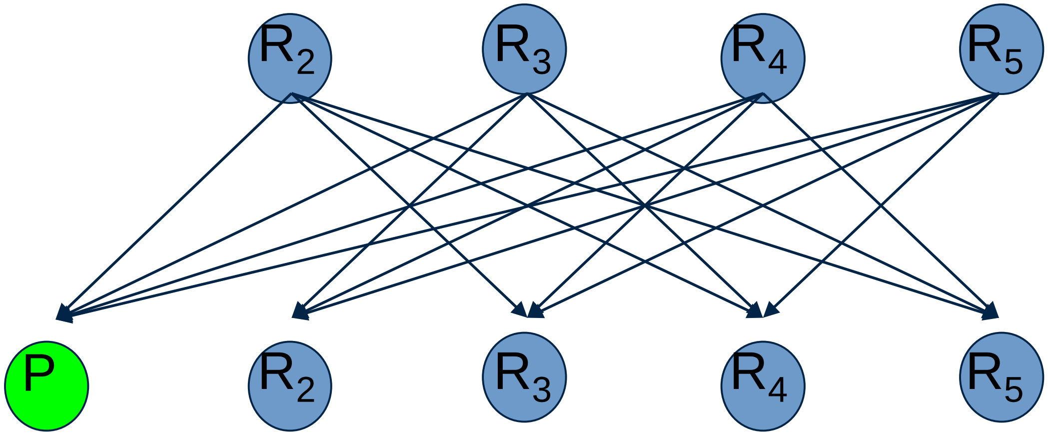
Pre-Prepare / Propose

R1 retransmits any uncommitted requests from the previous view to ensure the system continues processing without losing data.



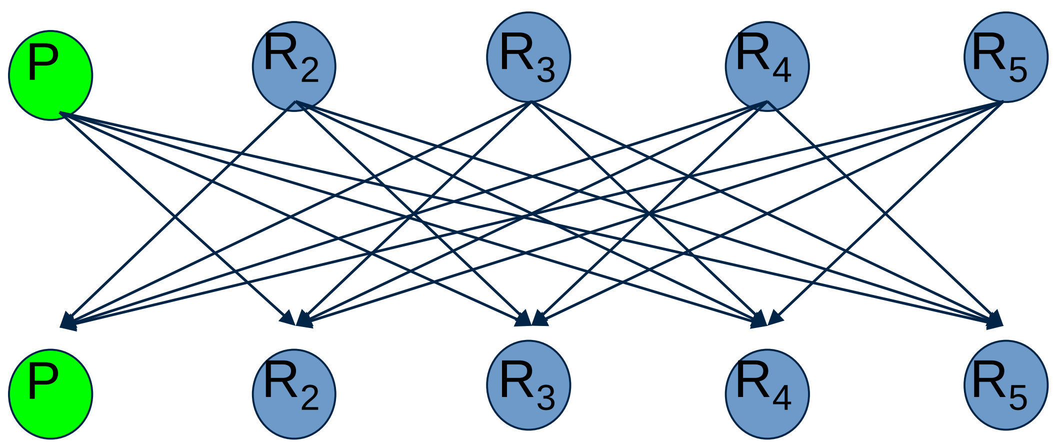
Prepare Phase

The Prepare phase ensures that ***2f+1*** replicas agree on the request order in New View, guaranteeing consistency.



Commit Phase

After receiving ***2f+1*** Commit messages, replicas execute the request, preserving consistency even after a primary failure.



Once the request is committed, the client receives replies from at least ***f+1*** replicas, guarantees that at least one reply is non-faulty.

Handling a Malicious Primary in PoE

What happens when the primary behaves maliciously?

- **By sending proposals for different transactions to different non-faulty replicas.**

  In this case, Proposition 3.2 guarantees that at most a single such proposed transaction will get view-committed by any non-faulty replica.

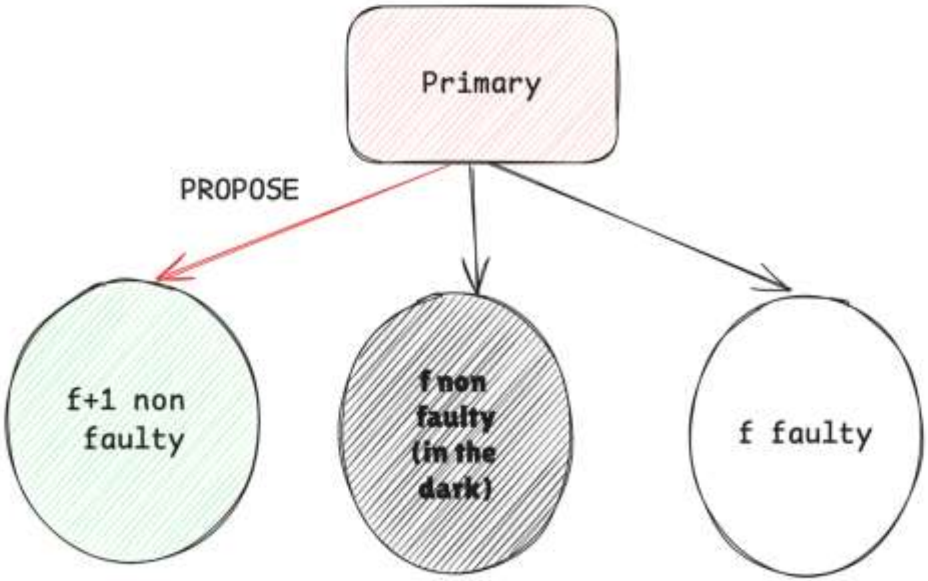
- **By keeping some non-faulty replicas in the dark by not sending proposals to them.**

  In this case, the remaining non-faulty replicas can still end up view-committing the transactions as long as at least $\mathbf{nf - f}$ non-faulty replicas receive proposals.

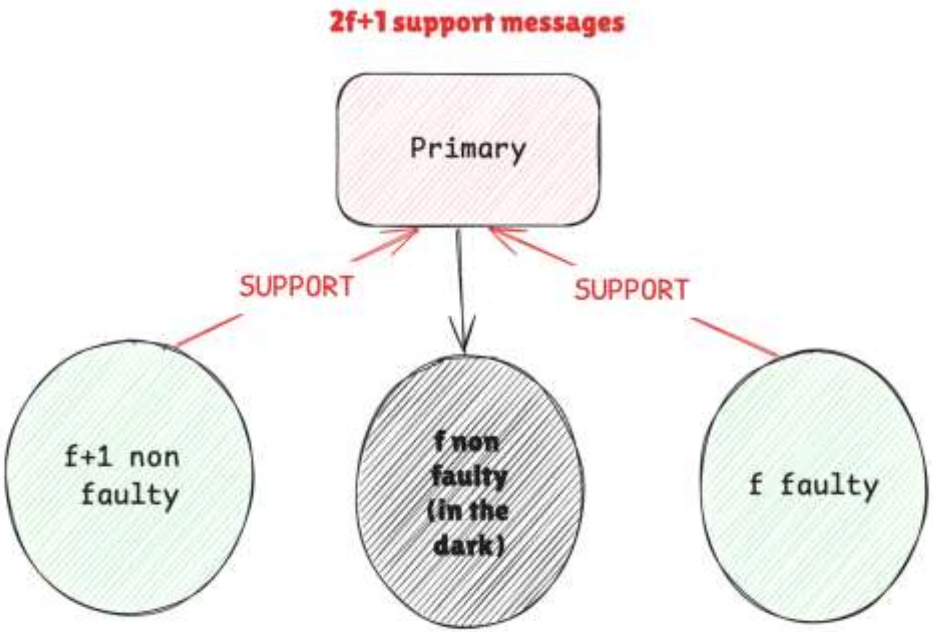
- **By preventing execution by not proposing a k-th transaction, even though transactions following the k-th transaction are being proposed.**

  In this case, the PoE protocol incorporates a view-change mechanism.

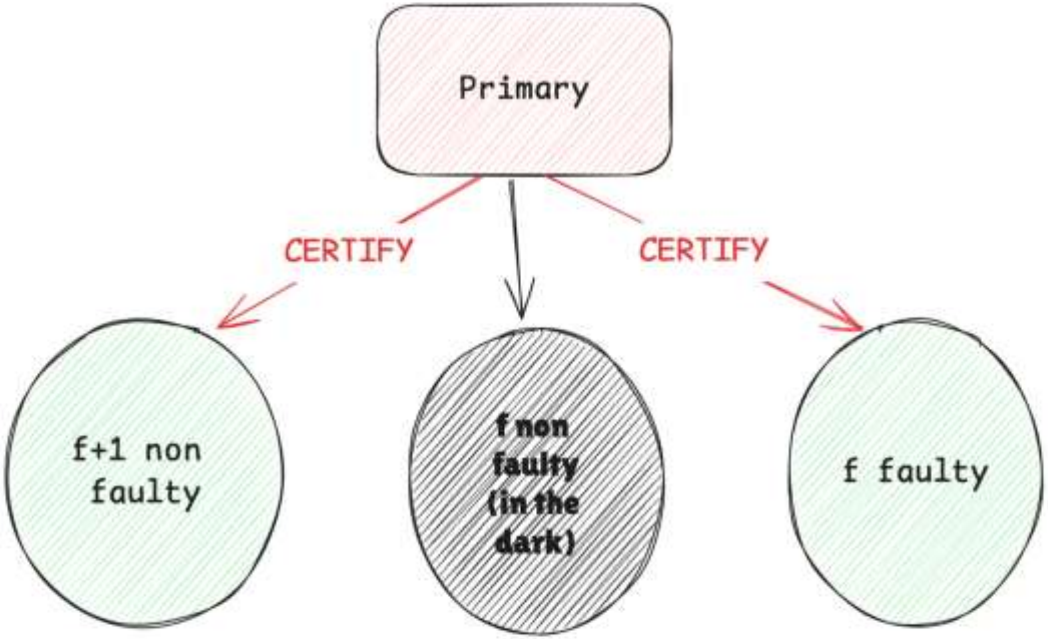
Handling a Malicious Primary in PoE



Handling a Malicious Primary in PoE



Handling a Malicious Primary in PoE

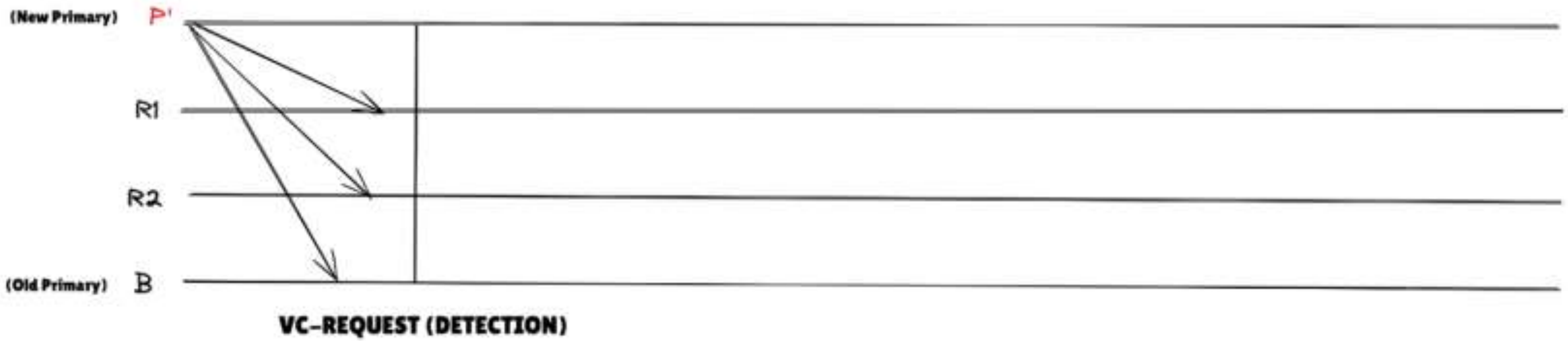


View Change Algorithm - Steps

View Change Algorithm consists of 3 steps:

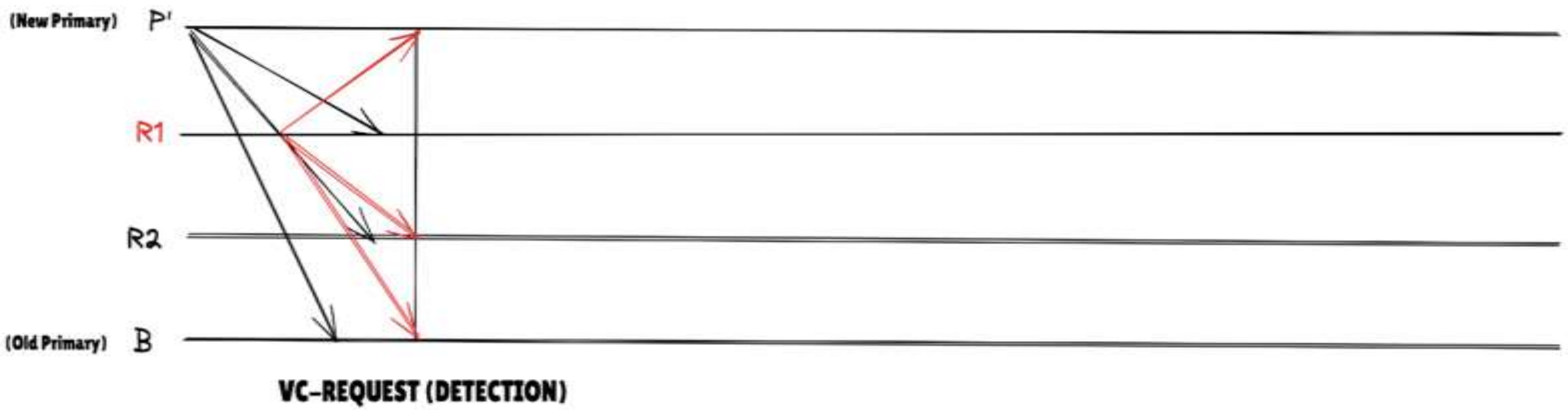
1. **Failure Detection and View-Change Requests**
   - Replica signals a primary failure by broadcasting a VC-REQUEST message to other replicas.
   - This VC-REQUEST contains a summary of all transactions executed by that replica.
   - A Replica detects a failure if
     - It *timeouts* waiting for a normal case operation.This initiates a VC-REQUEST (**detection**).
     - It receives a VC-REQUEST messages from f+1 **distinct** replicas. This indicates **joining** the view change request quorum.

View Change - Visualization



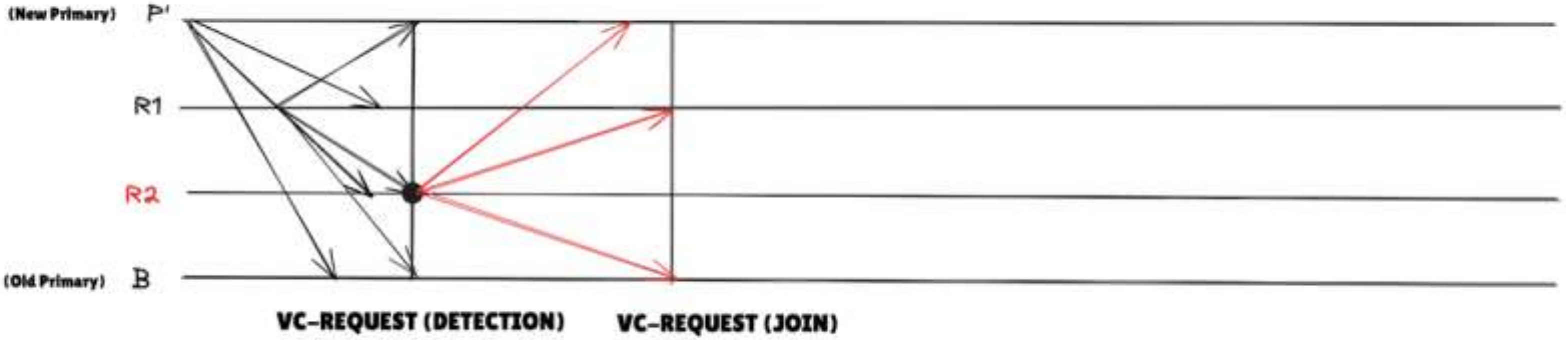
P' detects B is faulty and starts broadcasting VC-REQUEST messages to all other replicas

View Change - Visualization



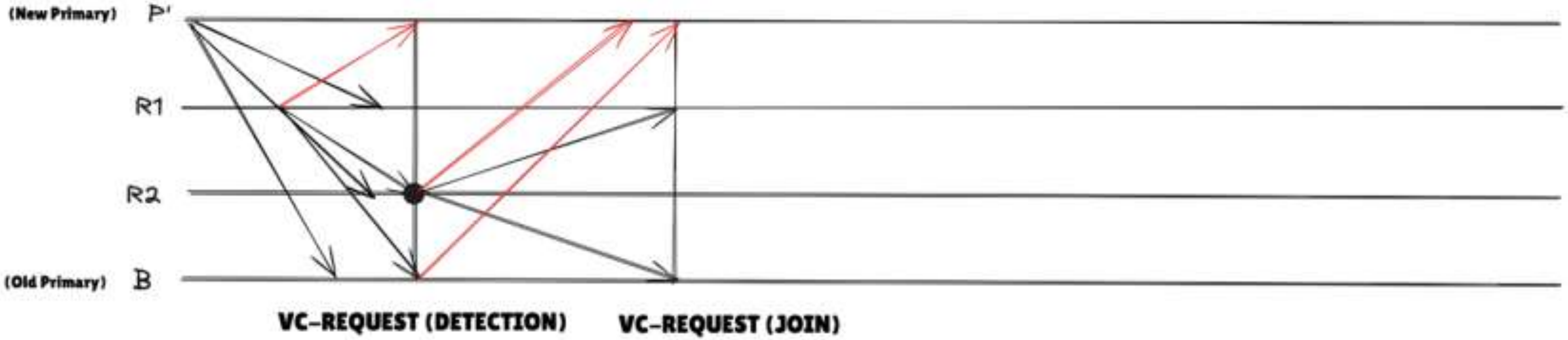
Now R1 detects B is faulty and starts broadcasting VC-REQUEST messages to all other replicas

View Change - Visualization



Now that R2 has recieved at least f+1 VC-REQUEST messages, it joins the View Change Quorum and starts broadcasting VC-REQUEST messages

View Change - Visualization



Now that the new primary P' has received VC-REQUESTS from 2f+1 distinct replicas (indicated by the red signals), it now progresses to broadcast the NV-PROPOSE message.

View Change - Steps

2. Proposing a new view

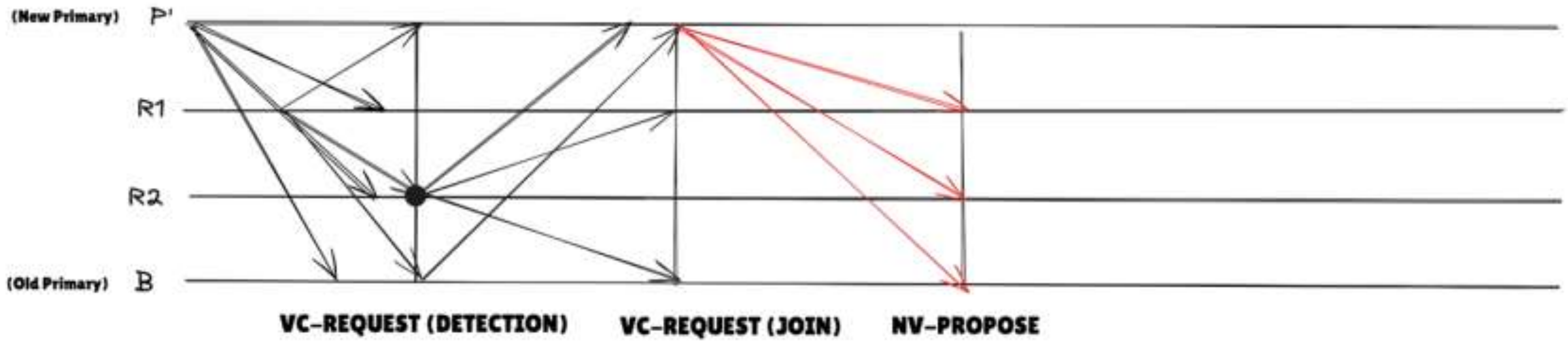
- In scenario of a view change, all replicas send and receive VC-REQUEST messages.

- Now that the old primary is faulty, a new primary p’ representing the new view (v+1) waits for a quorum of **nf** VC-REQUESTS. This should be from at least (2f+1) replicas.

- Once the new primary gets **nf** number of VC-REQUESTS, it knows that majority of the quorum can participate in the new view.

- It then broadcasts the **NV-PROPOSE** message.

View Change - Visualization



All replicas recieve NV-PROPOSE from the new primary P'

View Change - Steps

3. Moving to a new view
   - A replica R receives a NV-PROPOSE request.
   - From the VC-REQUESTS messages in the new proposal , replica R collects k_{max} transactions which happened in the last view.
   - R compares its snapshot with the proposed snapshot and starts **view committing** these.
   - Already executed ones are **skipped**, the ones which are in the old snapshot and not in the new one gets **rolled back**.
   - Once all k transactions are executed, switch seamlessly to the new view.

View Change - Move to new view

Scenario -1

R (Initial State)

A
B
C
D

Proposed State

A
B
C
D

New View

E

Requests which need to reconciled before moving to the new view

R1 (Final State)

A - Skipped
B - Skipped
C - Skipped
D - Skipped

E - Committed

View Change - Move to new view

Scenario -2

R (Initial State)

A
B
B’
C
D

Proposed State

A
B
C
D
— New View —
E

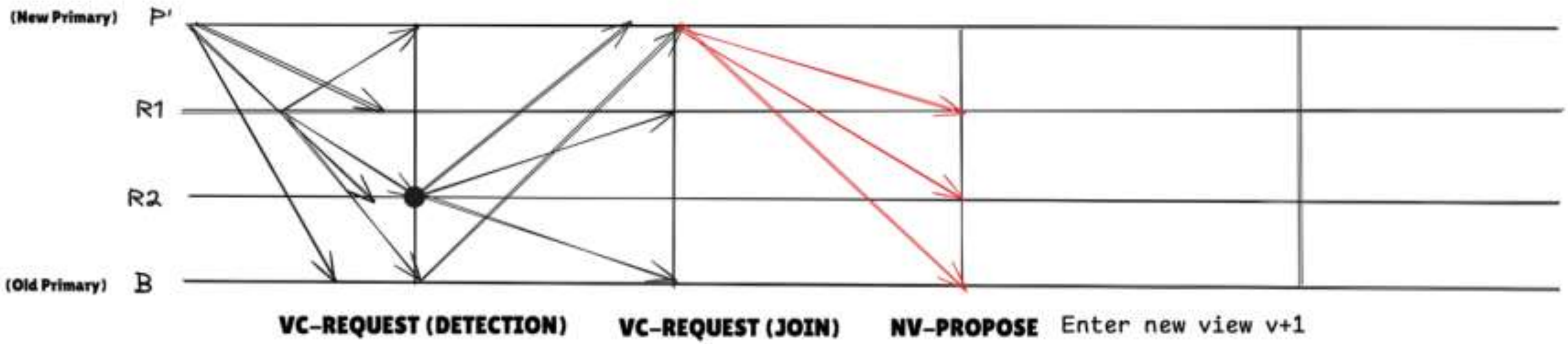
R1 (Final State)

A - Skipped
B - Skipped
B’ - **Rolled Back**
C - Skipped
D - Skipped

E - Committed

R has view-committed a transaction which has not been committed by other 2f+1 distinct replicas

View Change - Visualization



Once a replica R reconciles transactions (rollbacks/skips) transactions, then it enters the new view

Correctness Of Poe

- **Theorem:** Consider a system in view v, in which the first $k - 1$ transactions have been executed by all non-faulty replicas, in which the primary is non-faulty, and communication is reliable. If the primary received $\langle T \rangle c$, then the primary can use the normal-case algorithm to ensure that there is non-divergent execution of T.

- **Proposition:** Let $\langle T \rangle c$ be a request for which client c already received a proof-of-execution showing that T was executed as the k-th transaction of view v. If n > 3f, then every non-faulty replica that switches to a view v' > v will preserve T as the k-th transaction of view v.

- **Safety of PoE**: PoE provides speculative non-divergence if n > 3f.

- **Liveness of PoE**: PoE provides termination in periods of **reliable bounded-delay communication** if n > 3f.

UCDAVIS

Optimizations

Reduction of Signature Shares:

- Primary can generate one signature share itself.
- Only requires nf−1 shares from other replicas to reach **nf.**

Message Forwarding Optimization:

- Propose, support, inform, and nv-propose messages are not forwarded.
- Messages only need Message Authentication Codes (MACs) for security.

Certify Message Handling:

- Certify messages do not require signatures.
- Tampering with certify messages invalidates the threshold signature.

VC-Request Messages:

- VC-Request messages must be signed.
- Signing ensures integrity when messages are forwarded without tampering

ResilientDB Fabric

- Resilient DB provides state-of-the-art **replicated transactional engine** and offers a **high-throughput permissioned blockchain fabric**.

- Goals of Resilient DB:
    1. Implementing and testing different consensus protocols.

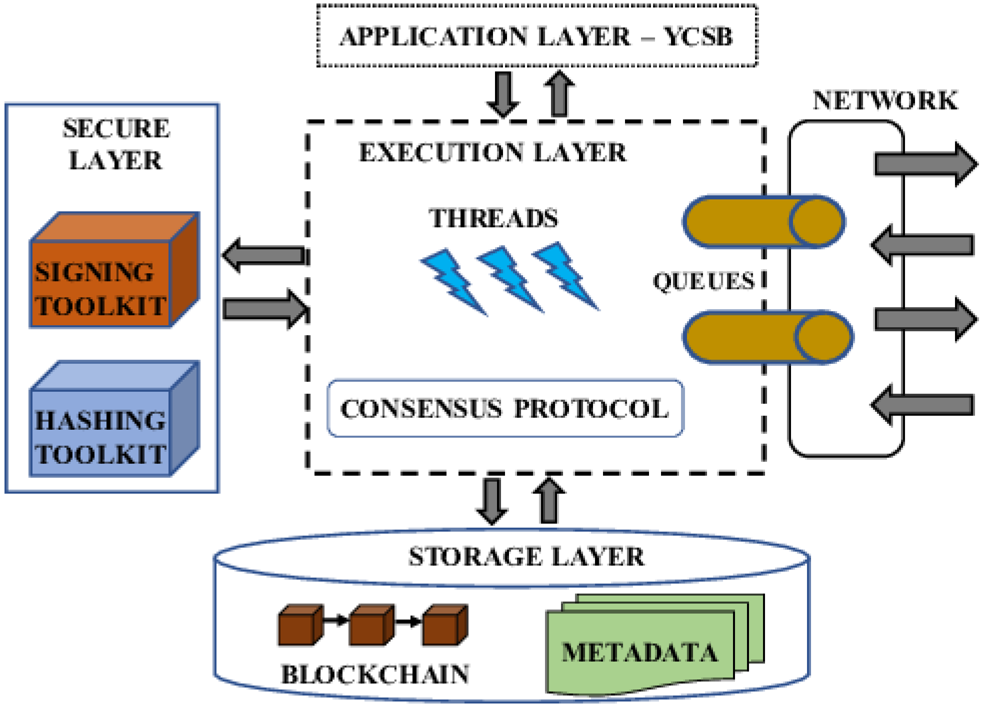
    1. Balance tasks using a **Parallel pipelined architecture** (distributing the workload evenly across different stages).

    1. Minimize communication cost **batching** client transactions.

    1. Enable the use of a secure and efficient ledger.

UCDAVIS

ResilientDB Architecture



Batching - A Key Design Decision

- Client-Server Architecture - client sends a request and waits for a response from the server.

- In a high throughput system, **batching** or aggregating requests reduces cost (latency, compute etc).

- In Resilient DB, client transactions are batched to reduce **cost of consensus**.

BATCHING AT THE PRIMARY

- The **input-threads** at the primary receive the client requests, assign them sequence numbers, and enqueue them into a shared **lock-free queue** called the **batch queue**.

- Each batch-thread continues adding requests to a batch until it reaches a predefined size. Once the batch is full, the thread hashes the requests to create a unique digest for the batch.

UCDAVIS

Batching At The Replicas

- All incoming messages at a replica are enqueued by the **input-thread** into the **work-queue**.
- The **single worker-thread** processes the messages from the work-queue.
- Upon receiving a **certify message** from the primary, the replica forwards the request to the **execute-thread**.
- The **execute-thread** handles the execution of the request.
- After execution is complete, the **execution-thread** creates an **inform message**.
- The **inform message** is transmitted to the client.

Ledger Management

- **Blockchain Ledger**: Maintained across replicas as an immutable ledger where blocks are linked as a chain.

- **Block Structure**: Each block(B) is a combination of k,d,v,H(B_{prev}).
  - k- sequence number of the client request
  - d- transaction digest
  - v- view number
  - H(B_{prev}) - hash of the previous block

- **Genesis Block**: The first primary replica creates a **genesis block** before any consensus, which acts as the first block in the blockchain, using the **hash of the initial primary's identity**.

- **Execution and Block Creation**: The **execute-thread** creates a block by hashing the previous block and adding the new batch of transactions.

- **Proof of Validity**: Each block contains a proof-of-acceptance for the corresponding request, validated by a **threshold signature** sent by the primary as part of the certify message.

Conclusion

- POE Protocol – A fast and reliable Byzantine Fault-Tolerant consensus through speculative execution, reducing delays and improving efficiency.
- POE provides a scalable, efficient, and fault-tolerant consensus mechanism for modern blockchain applications, outperforming existing BFT protocols.
- Achieves up to 80% more throughput than traditional BFT protocols, handling failures without relying on slow twin-path models.
- Efficient across various system sizes, with support for both symmetric and asymmetric cryptographic signatures, adaptable to different environments.
- Implemented and tested in ResilientDB, practical value for high-throughput blockchains.

UC DAVIS

THANK YOU