RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing

Authors: Suyash Gupta, Jelle Hellings, and Mohammad Sadoghi

Presenters:

Naitik Jain, Tanvi Mehta, Shivani Suryawanshi, Krithika Naidu

Contents

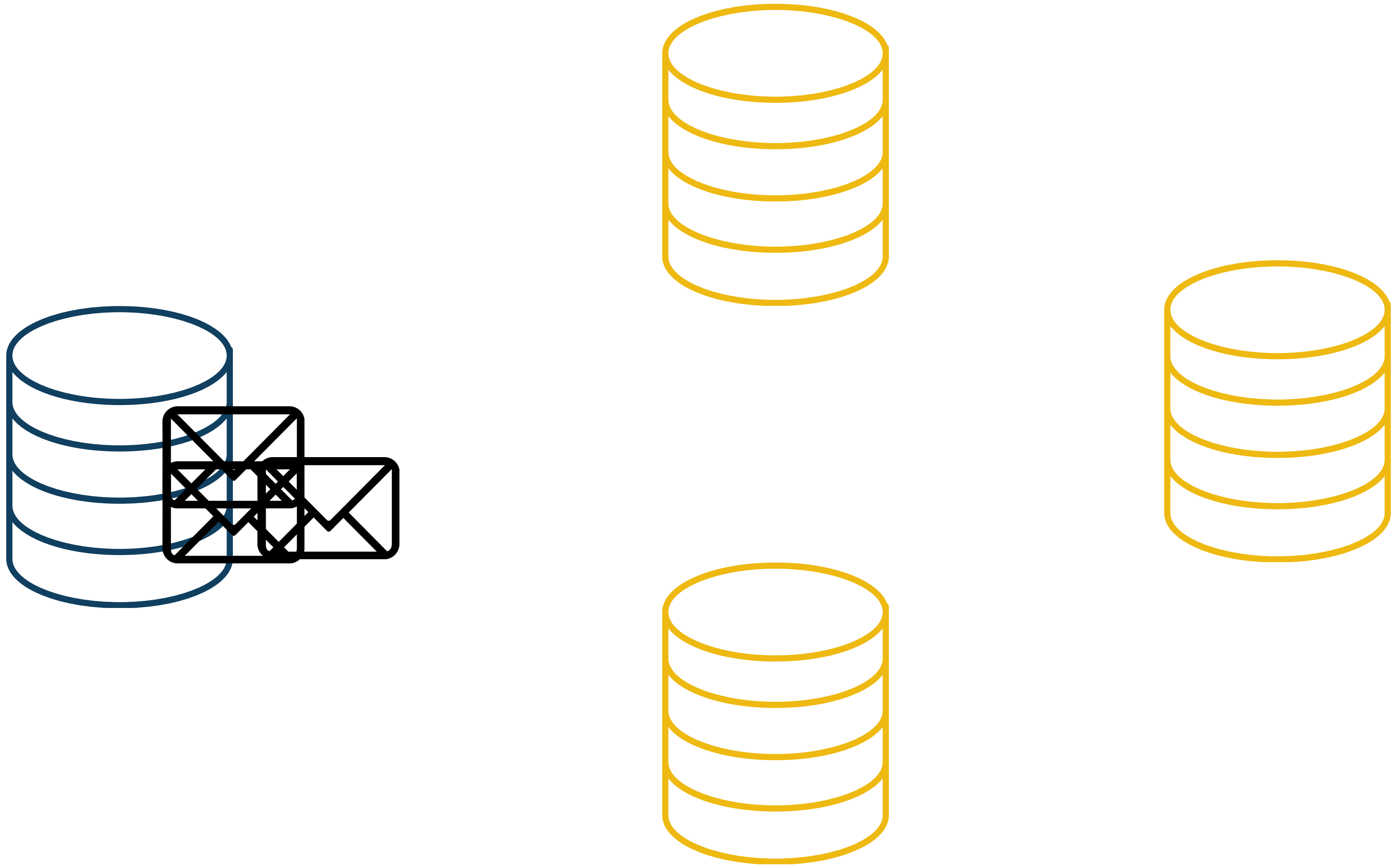
- Overview
- Traditional Consensus: PBFT
- Limitations of Traditional Consensus
- Promise of Concurrent Consensus
- Solution
- Resilient Concurrent Consensus (RCC)
- Design of RCC with Example
- Dealing with Detectable Failures
- Undetectable Failures
- Client Interactions with RCC
- RCC Improving Resilience of Consensus
- Evaluation of the Performance of RCC
- Conclusion
- References

Overview

- Consensus-based database systems are emerged, offering resilience, strong data provenance, and federated data management.
- Rely on Primary-backup protocols.
- RCC enhances the throughput, resilience, and security by enabling concurrent consensus.

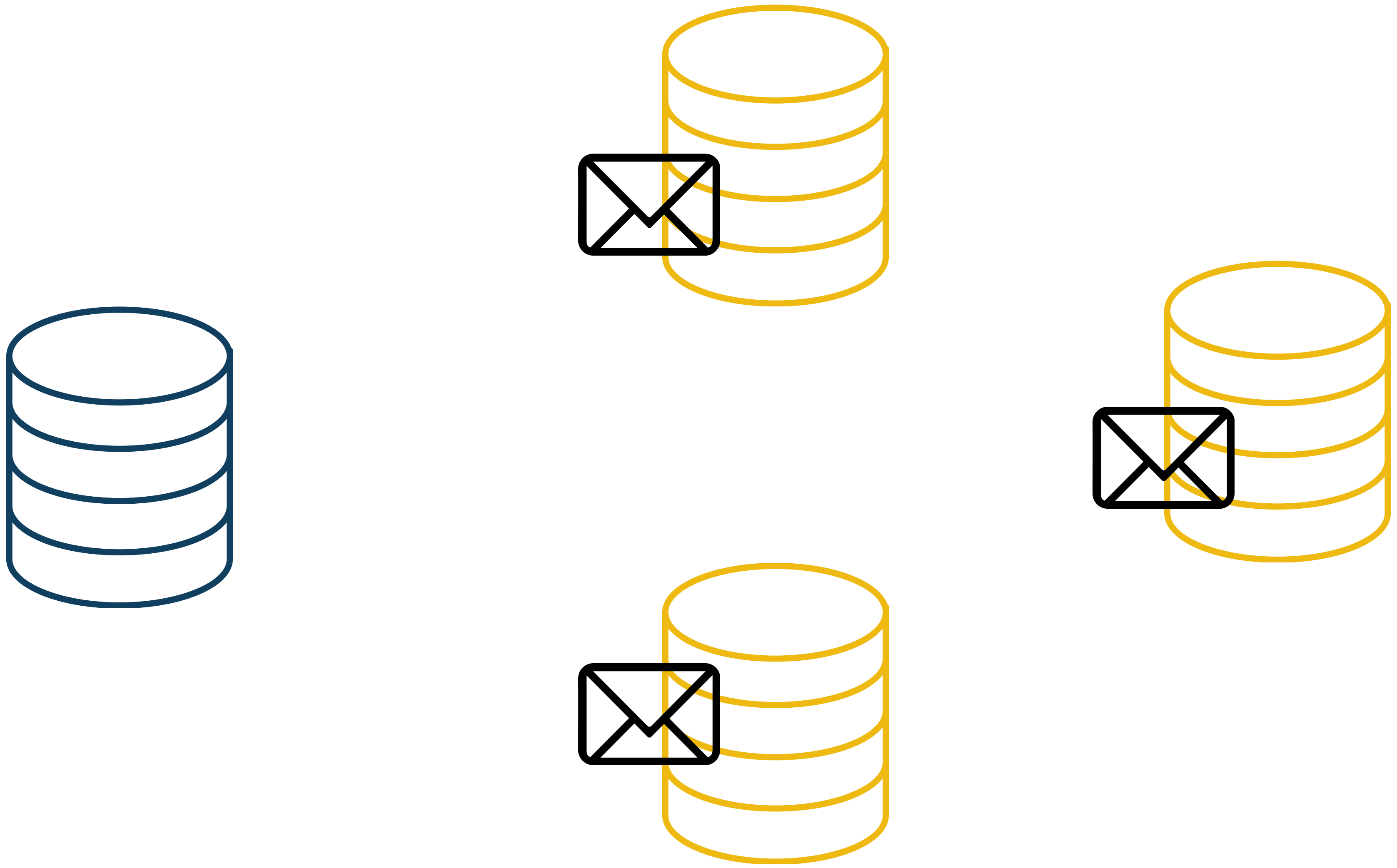
Overview

At the core of any blockchain application or distributed database system, there is Byzantine Fault-Tolerant (BFT) consensus protocol.

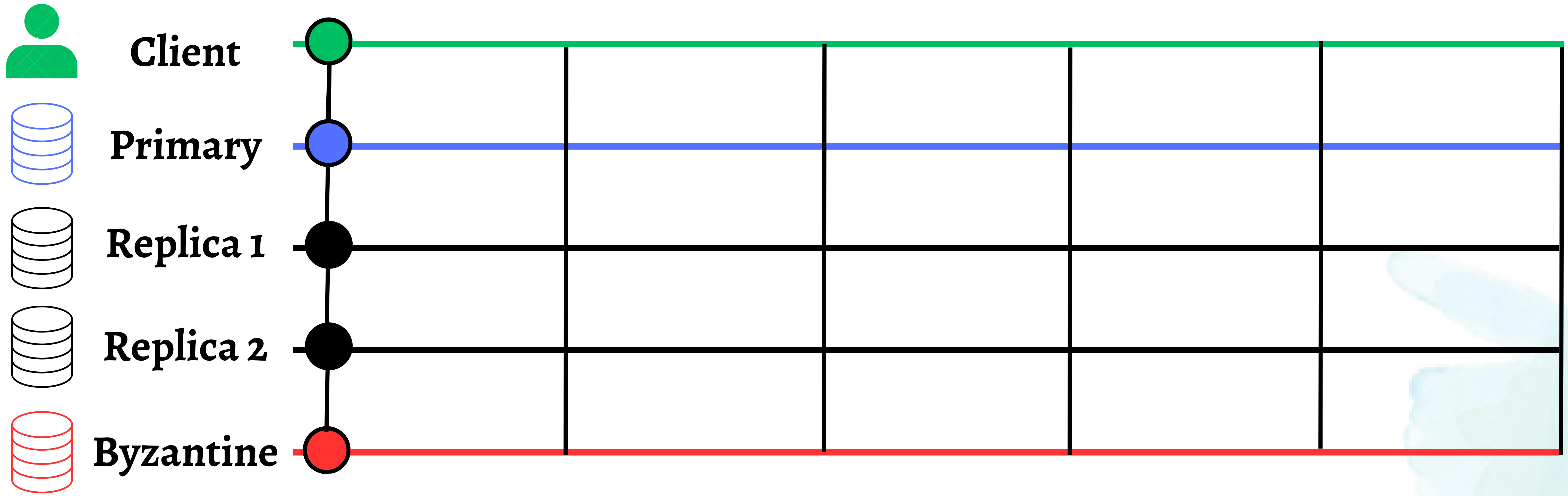


Overview

At the core of any blockchain application or distributed database system, there is Byzantine Fault-Tolerant (BFT) consensus protocol.

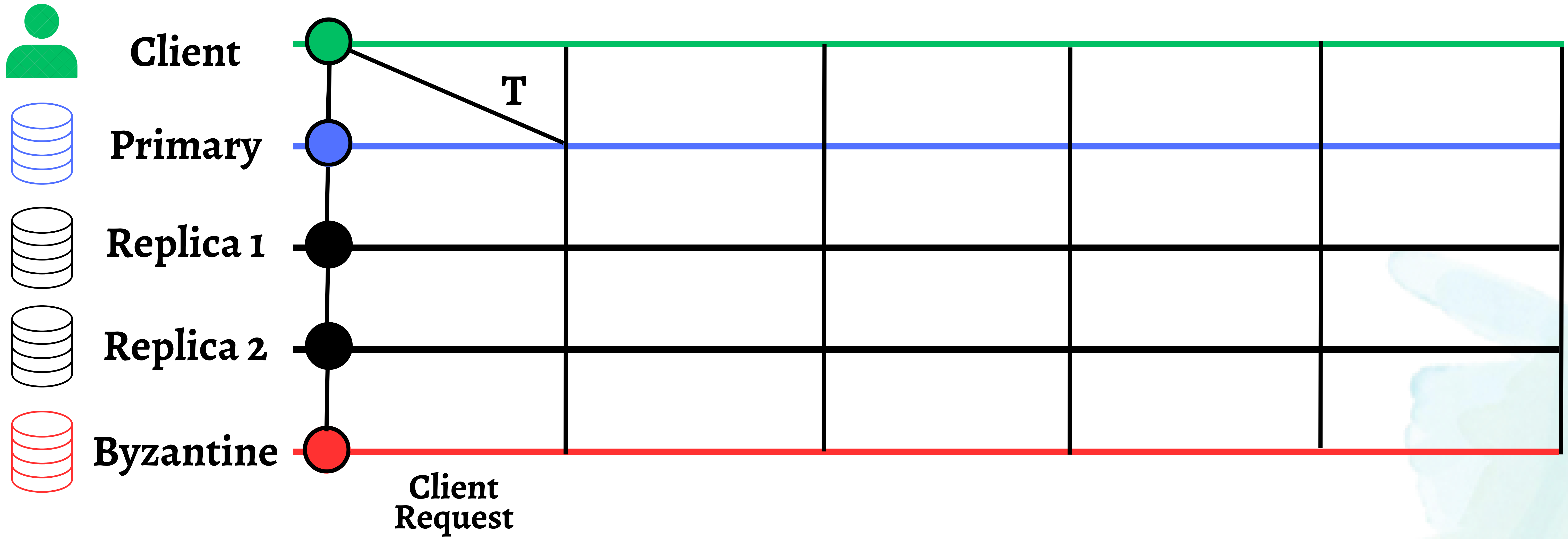


Traditional Consensus: PBFT



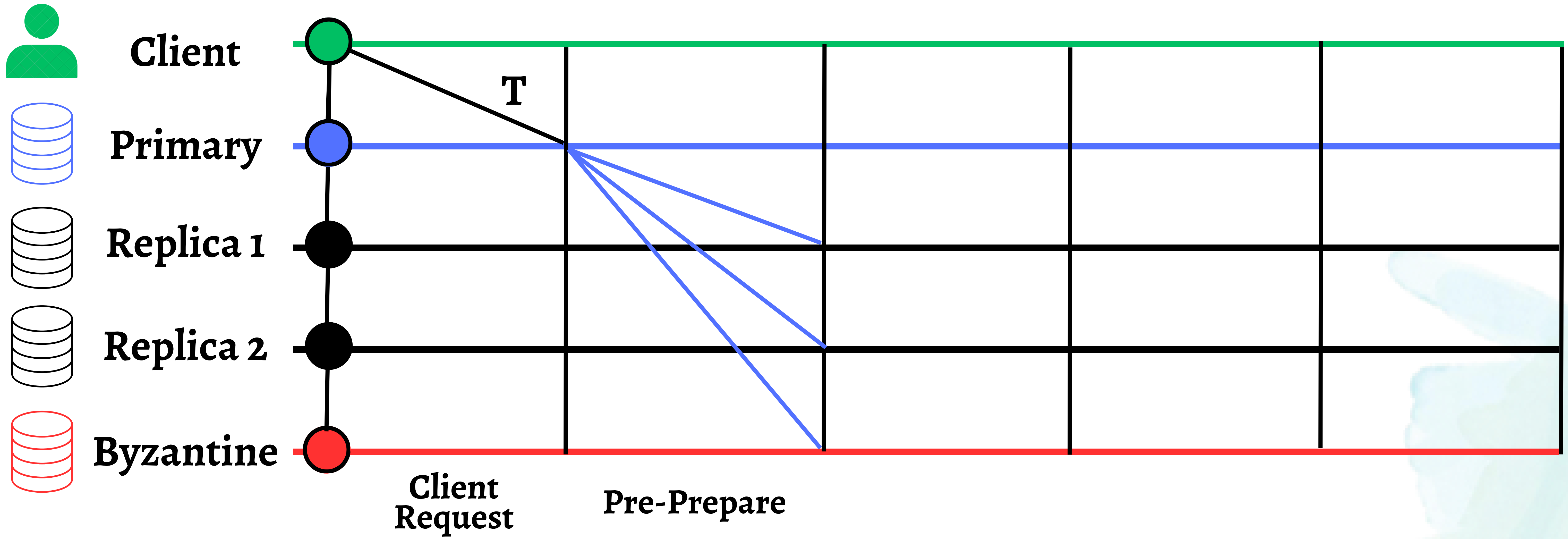
Consensus among n replicas and atmost f byzantine ⟶ $n \geq 3f+1$

Traditional Consensus: PBFT



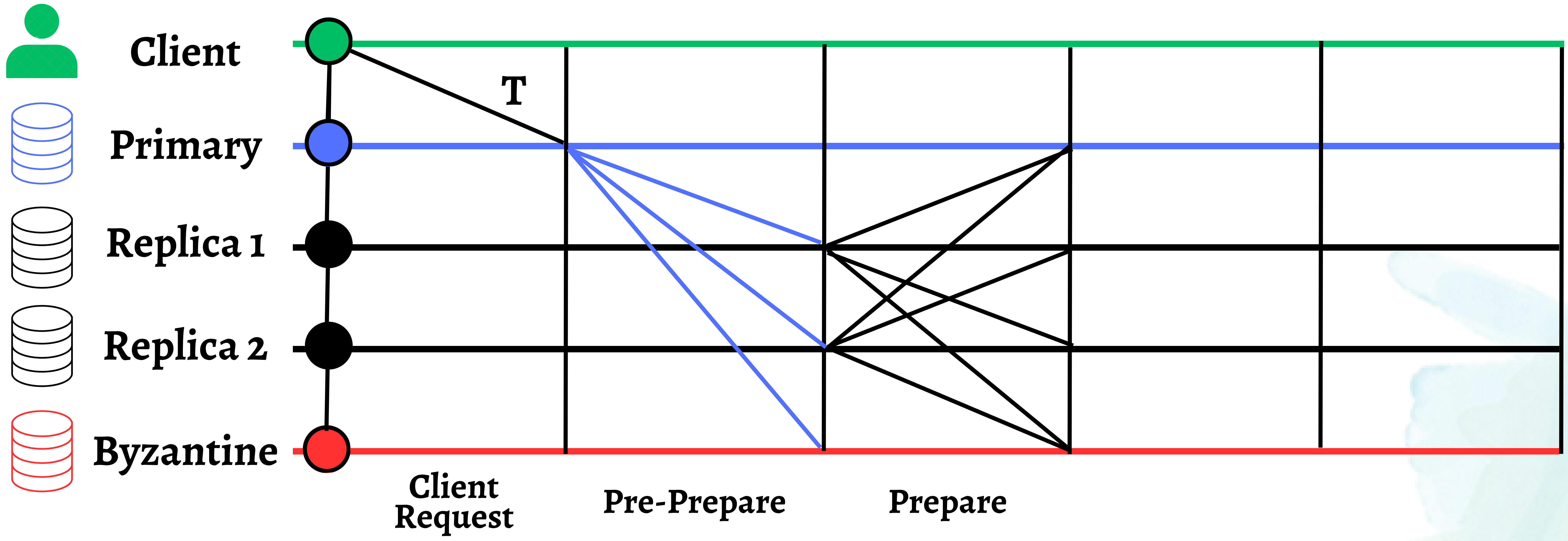
n = 4 replicas and f <= 1

Traditional Consensus: PBFT



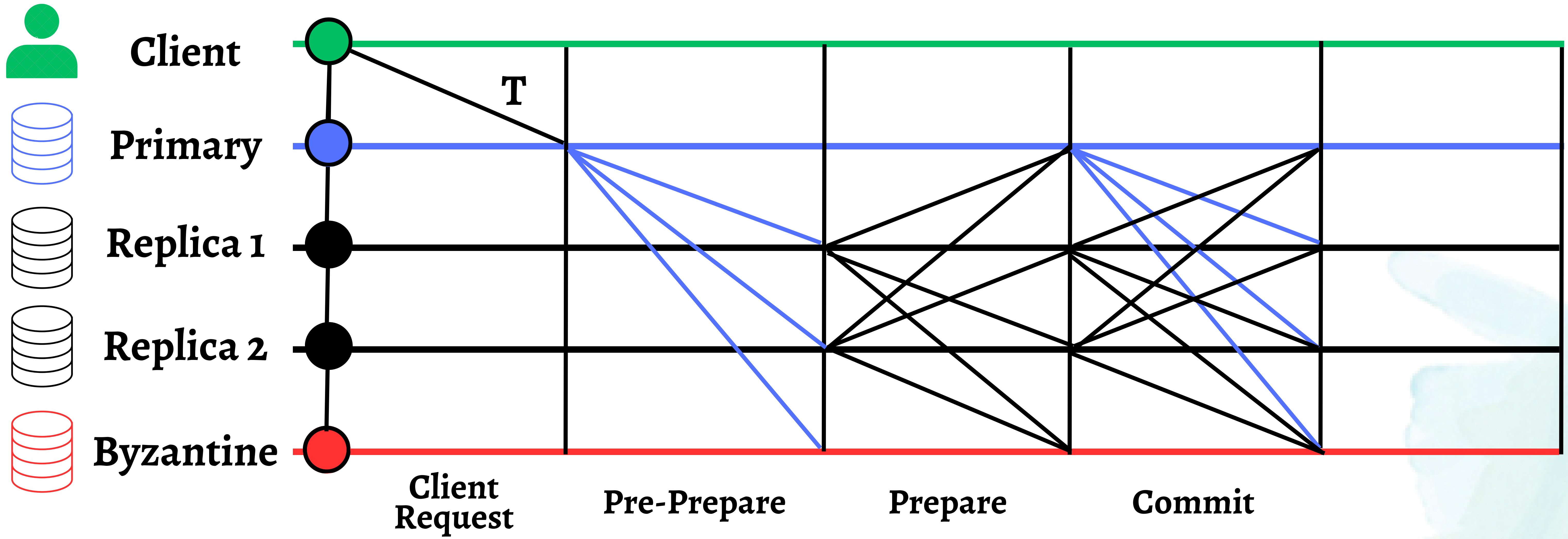
n = 4 replicas and f <= 1

Traditional Consensus: PBFT



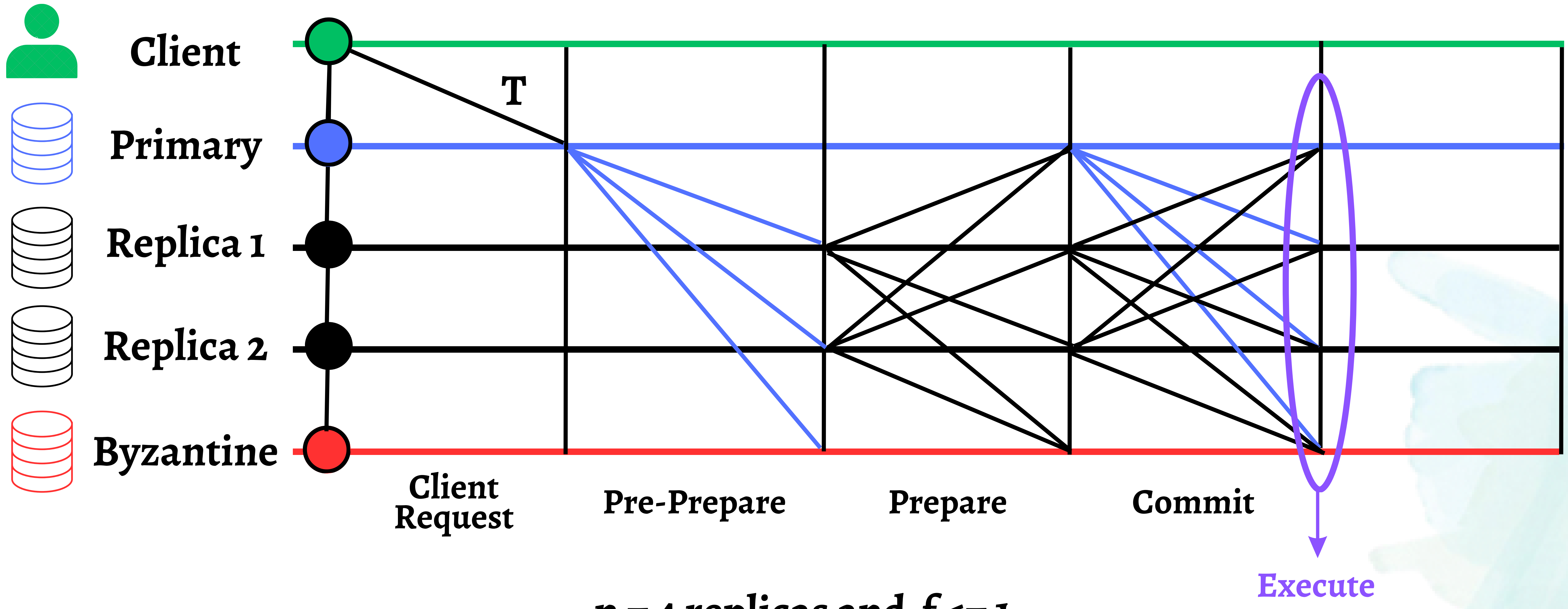
n = 4 replicas and f <= 1

Traditional Consensus: PBFT



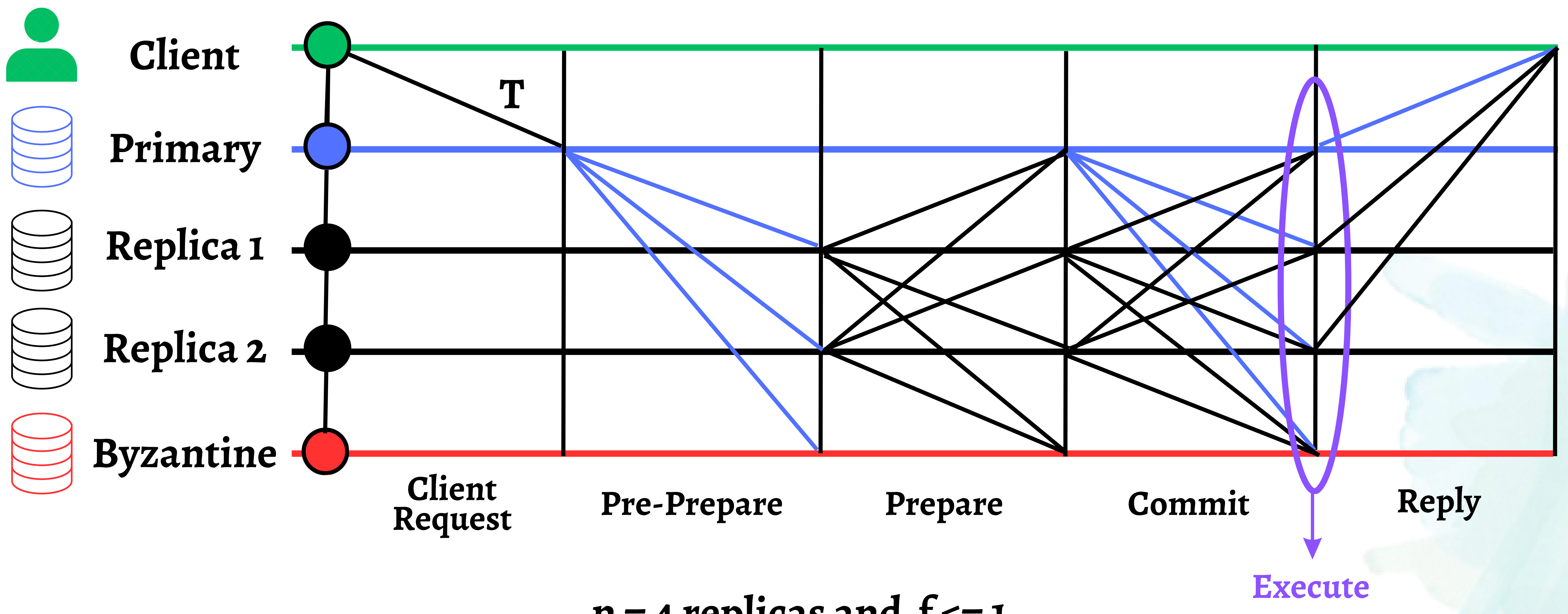
n = 4 replicas and f <= 1

Traditional Consensus: PBFT



n = 4 replicas and f <= 1

Traditional Consensus: PBFT



n = 4 replicas and f <= 1

Limitations

- Traditional primary-backup consensus protocols underutilize network resources and thus prevents maximization of transaction throughput, which is determined by the outgoing bandwidth of the primary.
- Considering a system,

$$T_{max} = \frac{B}{(n-1)st} \qquad T_{PBFT} = \frac{B}{((n-1)(\mathbf{st} + 3\mathbf{sm}))}$$

st is the size of each transaction sm is the size of each message

Limitations

$$T_{PBFT} \approx T_{max} \text{ when } \mathbf{st} >> \mathbf{sm}$$

- Replicas are underutilized: primaries must send (n-1)st while replicas only have to send and receive st bytes roughly, given that st >> sm.
- Underutilization of non-primary replicas in comparison to primaries!

Promise of Concurrent Consensus

- **Democracy** - Give all the replicas the power to be the primary.
- **Parallelism** - Run multiple parallel instances of a BFT protocol.
- **Decentralization** - Always there will be a set of ordered client requests.
- HotStuff balances load by consistently switching primaries, but does not address core issue of underutilization of resources.
- Concurrent consensus involves proposing at least **nf transactions** at a time.

$$T_{cmax} = \mathbf{nf}\frac{B}{((\mathbf{n}-1)\mathbf{st} + (\mathbf{nf}-1)\mathbf{st})}$$

Solution

- Concurrent consensus can achieve higher levels of throughput by efficiently utilizing all available replicas.

- Resilient Concurrent Consensus (RCC) is a paradigm for transforming any primary-backup consensus protocol into a concurrent consensus protocol with increased throughput.

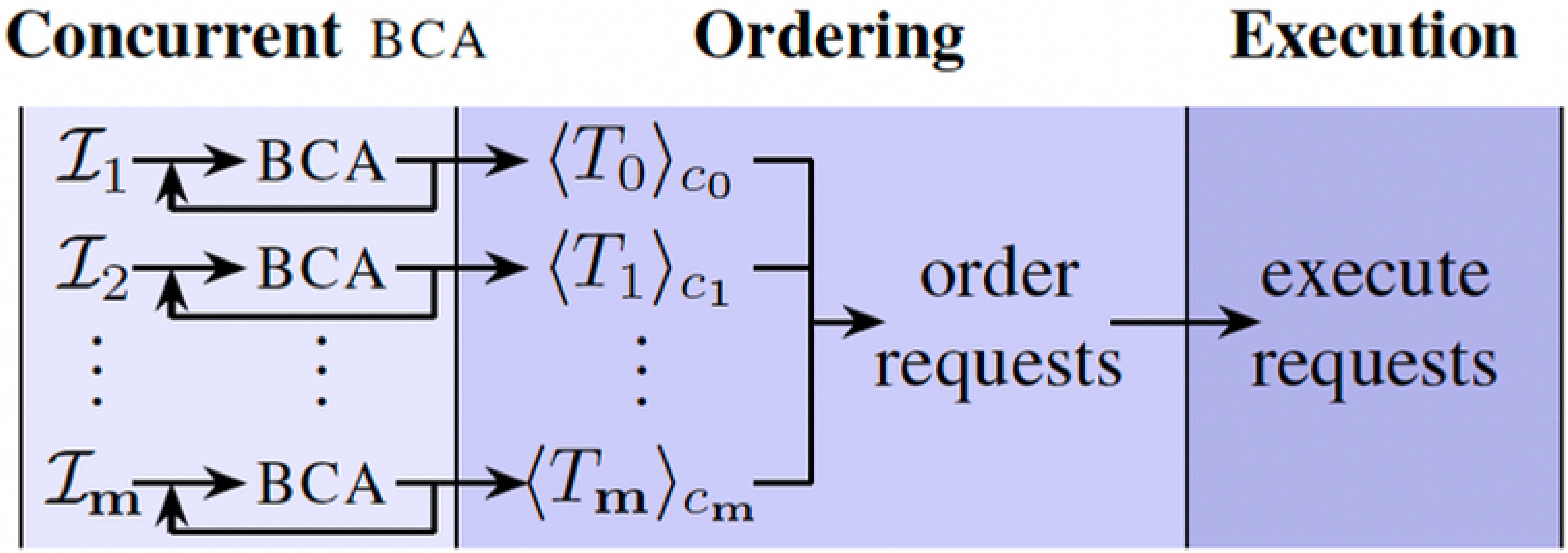
Resilient Concurrent Consensus (RCC)

- To deal with underutilisation of resources and low throughput
- Turn any primary backup consensus protocol into a concurrent consensus protocol

Design Goals:

- Provide consensus among replicas on client transactions that are to be executed
- Applied to any other protocols
- Dealing with faulty primaries does not interfere with the operations of other consensus-instances

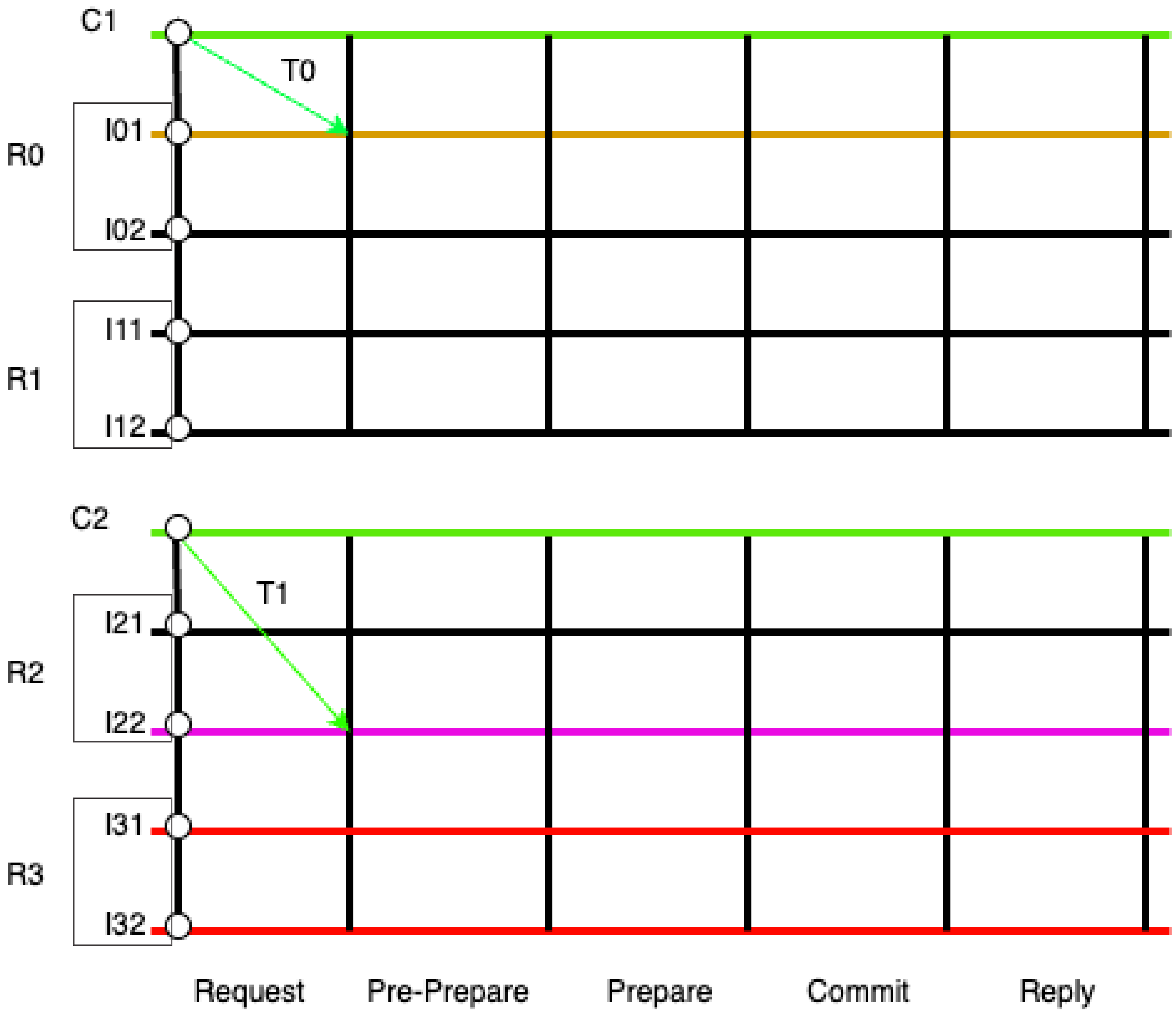
Design of RCC



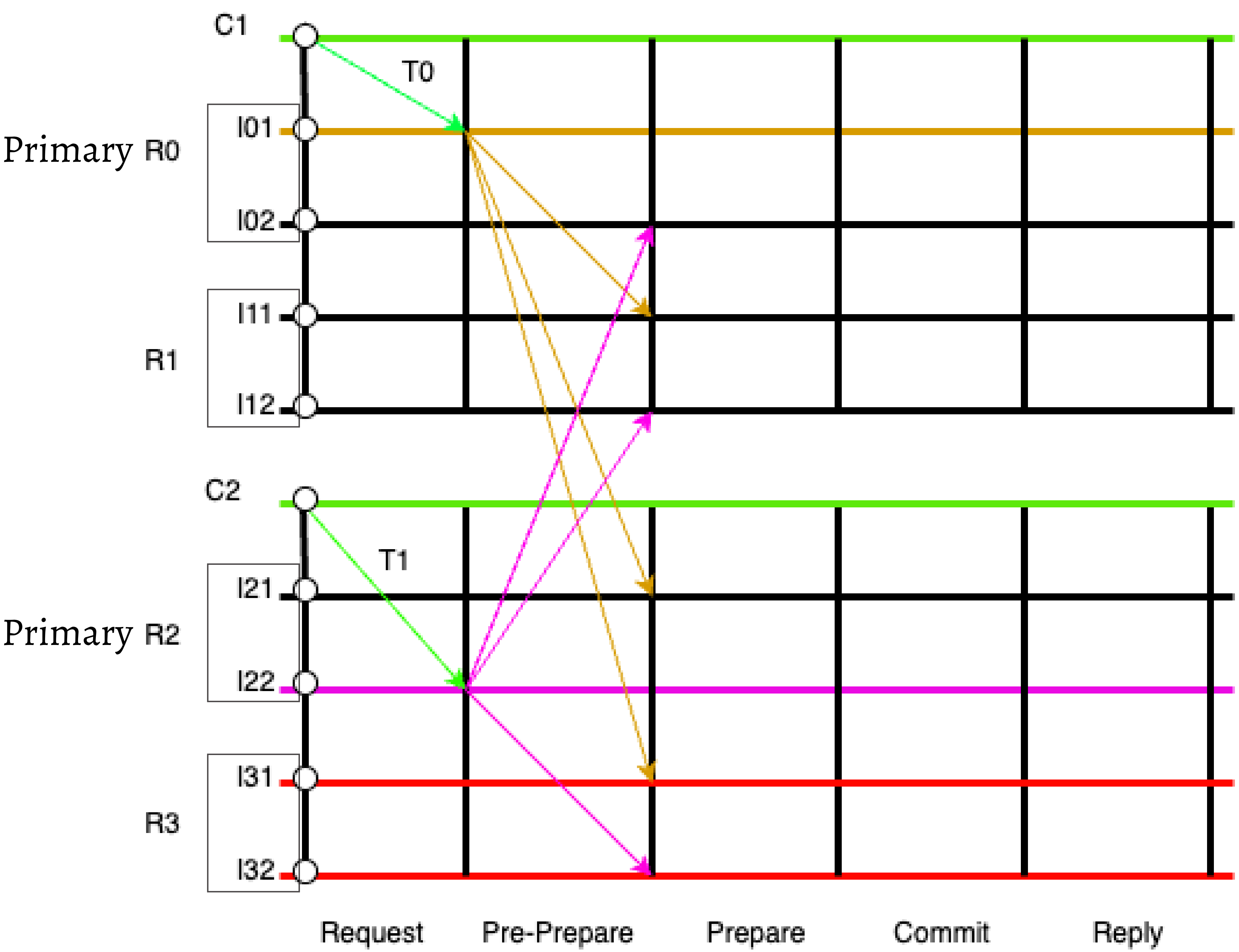
Example

Primary Replica

Primary Replica

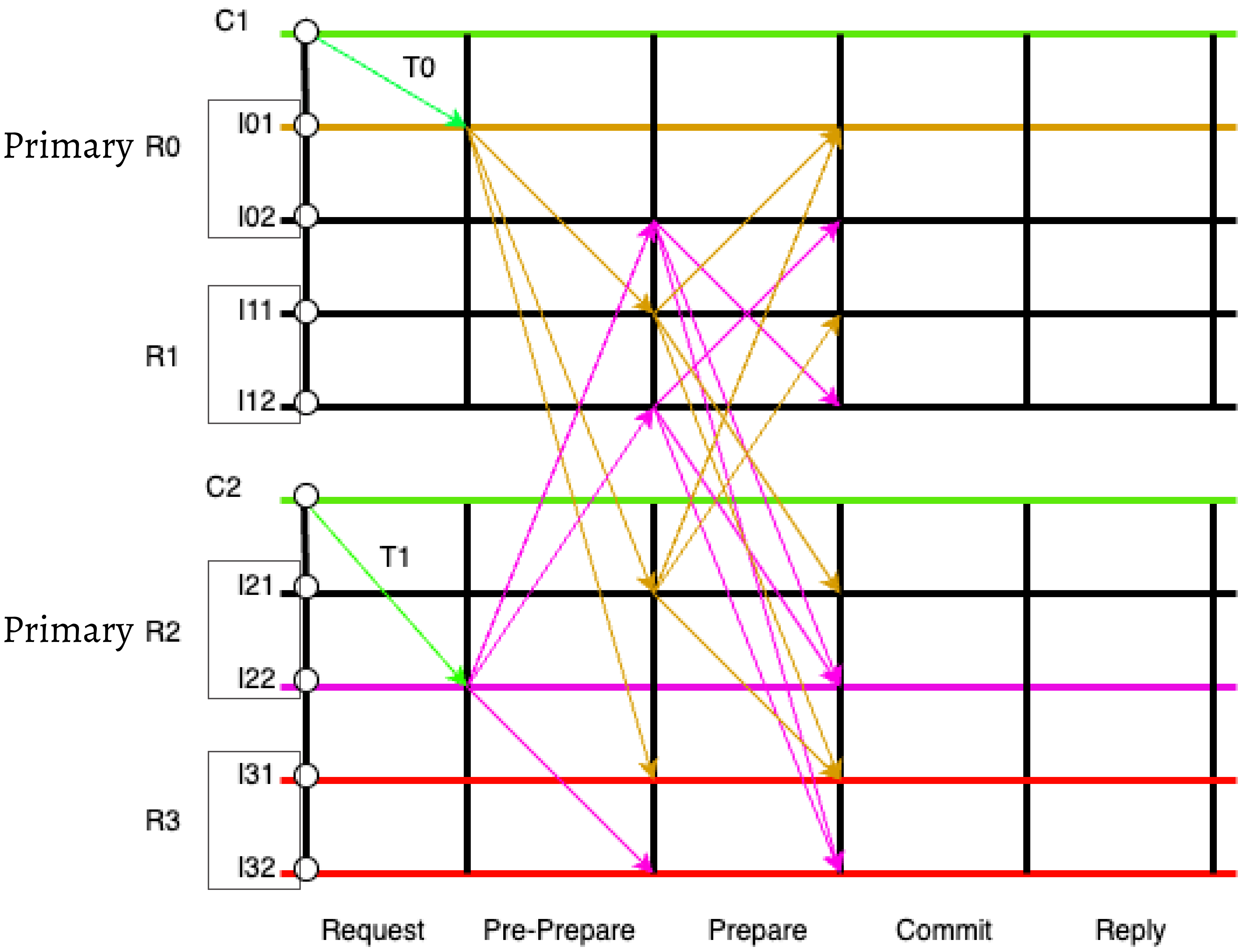


Example



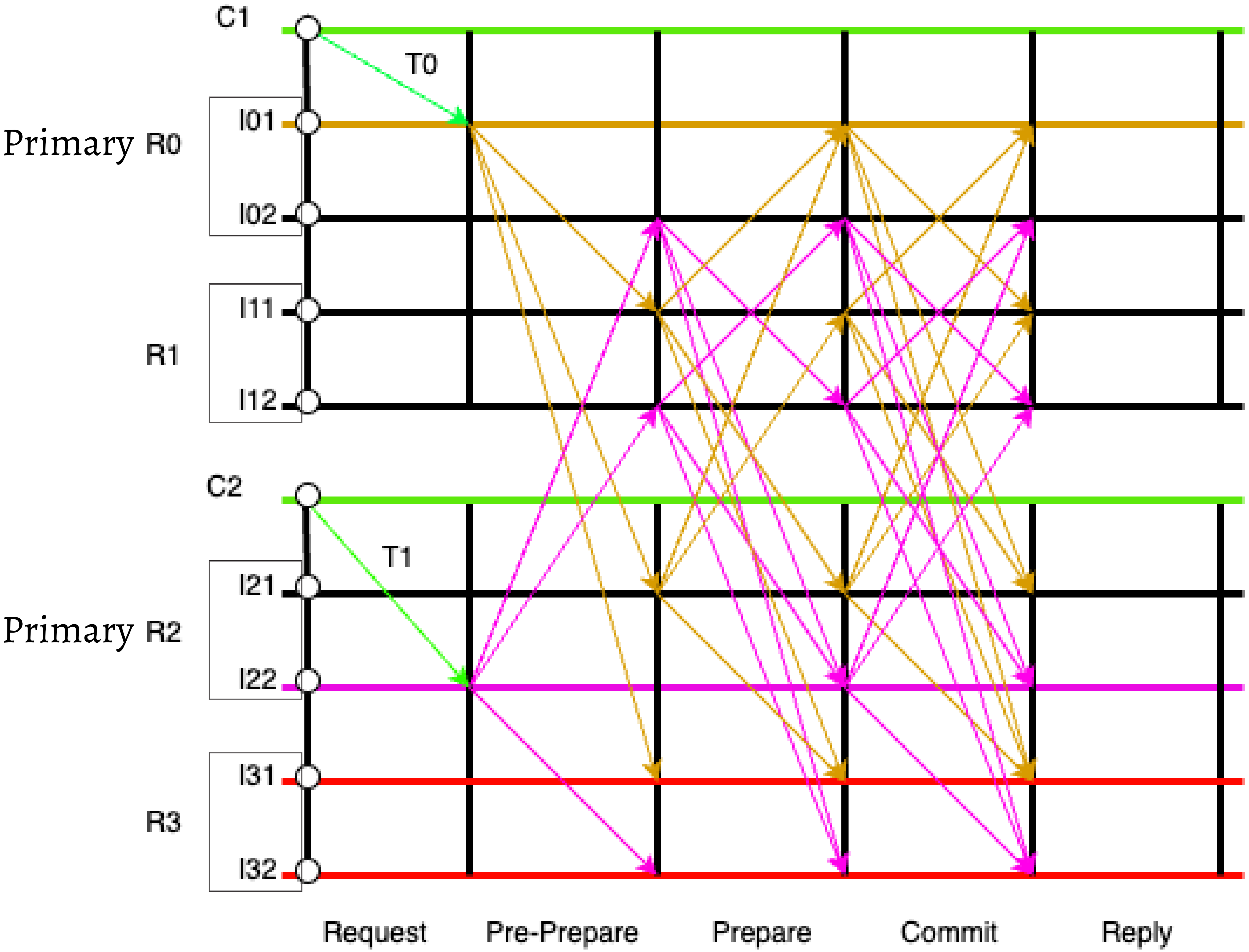
C1
T0
Primary R0
I01
I02
R1
I11
I12
C2
T1
Primary R2
I21
I22
R3
I31
I32
Request
Pre-Prepare
Prepare
Commit
Reply

Example



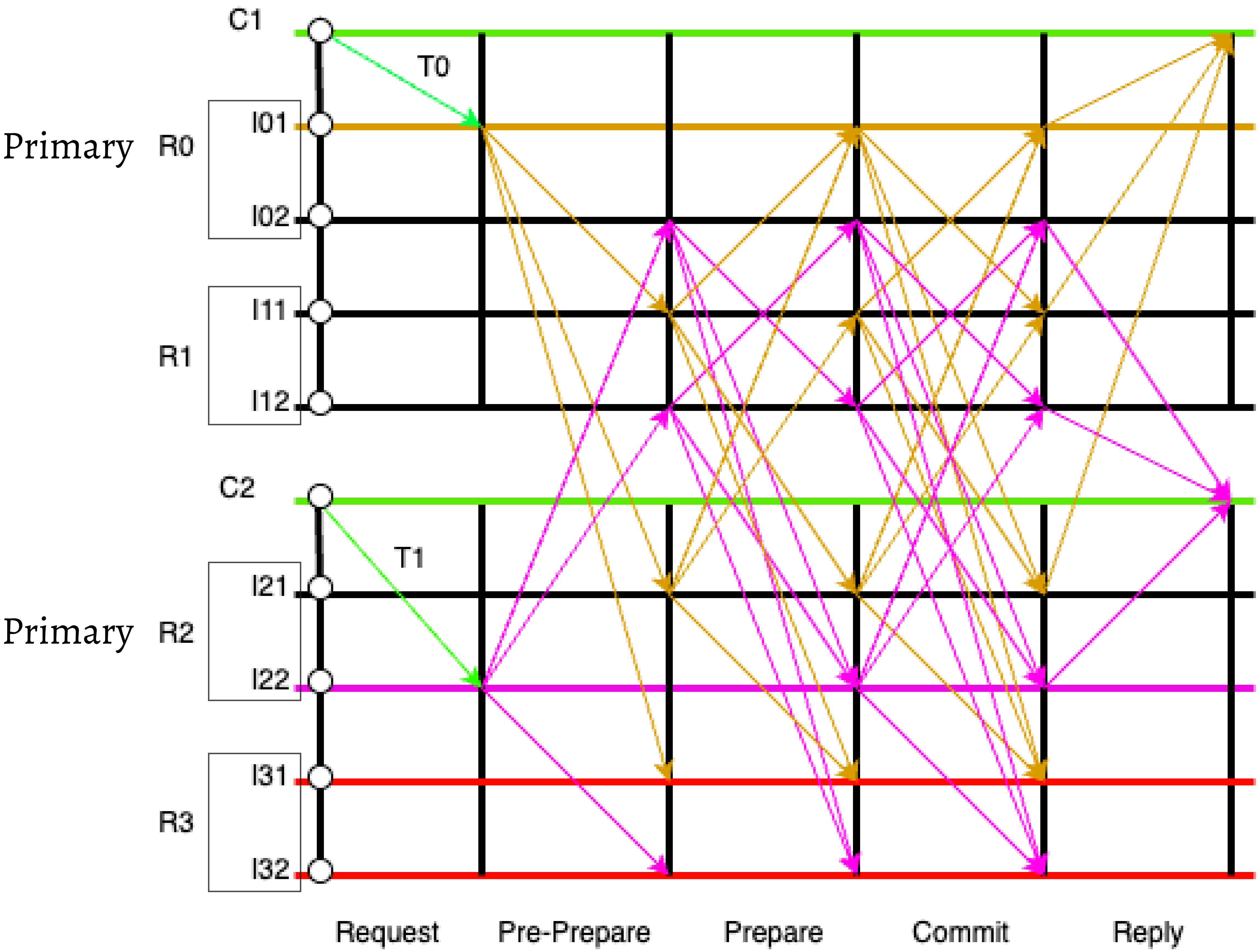
C1
T0
Primary R0
I01
I02
I11
R1
I12
C2
T1
I21
Primary R2
I22
I31
R3
I32
Request
Pre-Prepare
Prepare
Commit
Reply

Example



C1
T0
Primary R0
I01
I02
I11
R1
I12
C2
T1
I21
Primary R2
I22
I31
R3
I32
Request
Pre-Prepare
Prepare
Commit
Reply

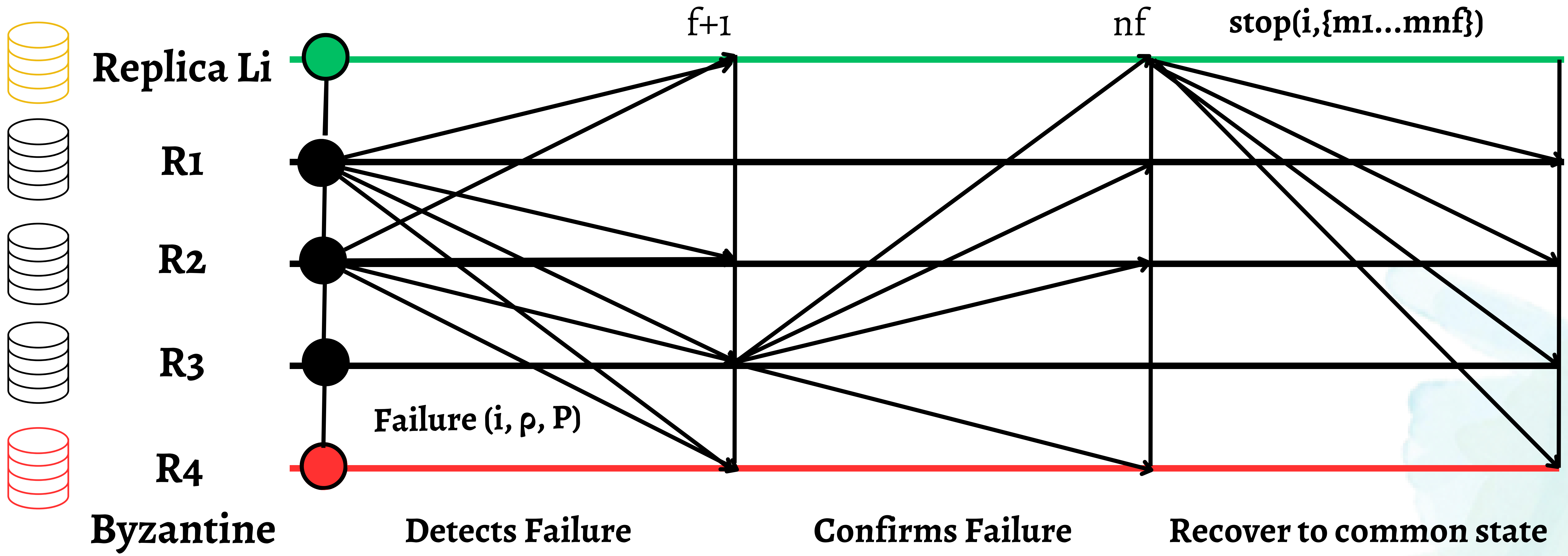
Example


C1
T0
Primary
R0
I01
I02
I11
R1
I12
C2
T1
I21
Primary
R2
I22
I31
R3
I32
Request
Pre-Prepare
Prepare
Commit
Reply

Dealing with failures

- RCC follows asynchronous communication - Impossible to distinguish between malicious primary and primary whose messages get lost in the network.
- Reliable bounded delay communication - All messages sent by nf replicas will arrive within some maximum delay.

Dealing with Detectable Failures



i: primary number
ρ: round number
P: state of R(that detects failure)
E: set of nf failure messages

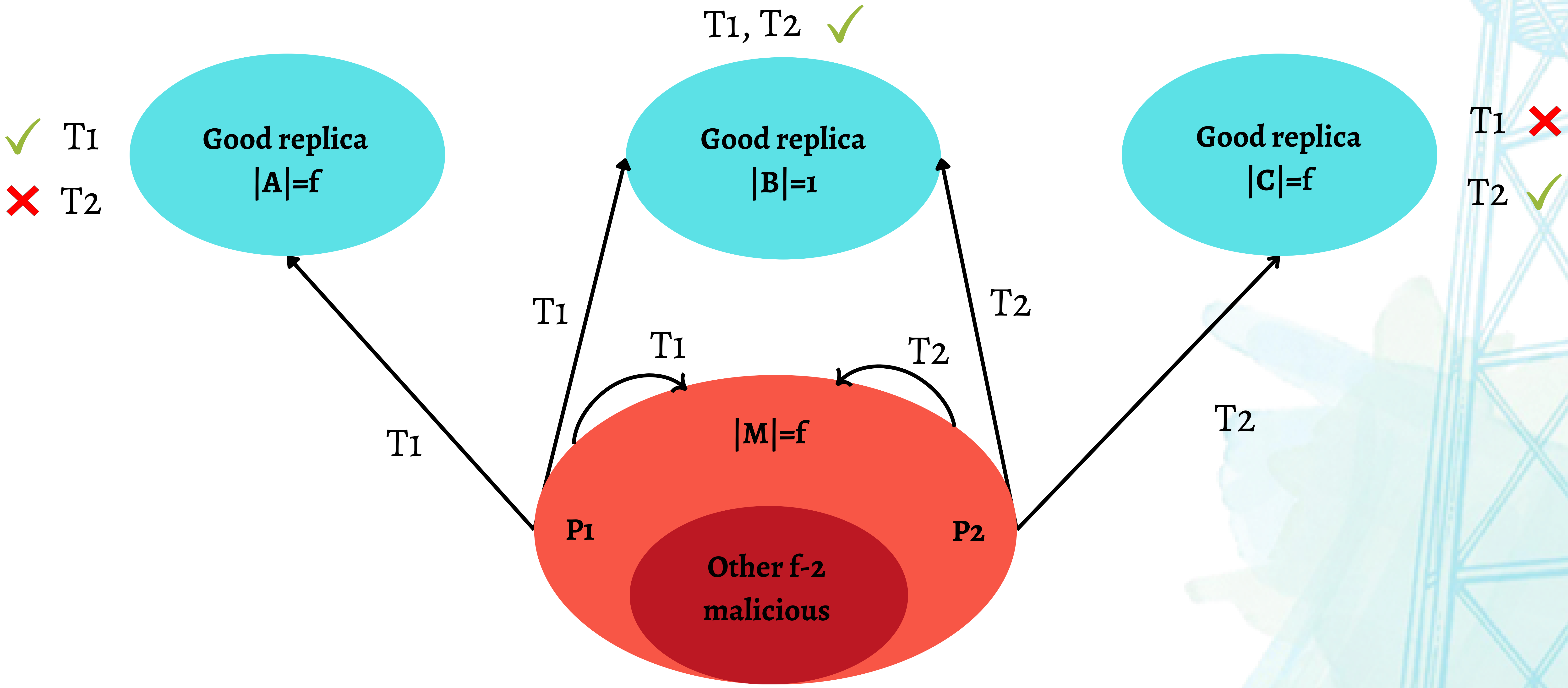
Dealing with Detectable Failures

Recovery process

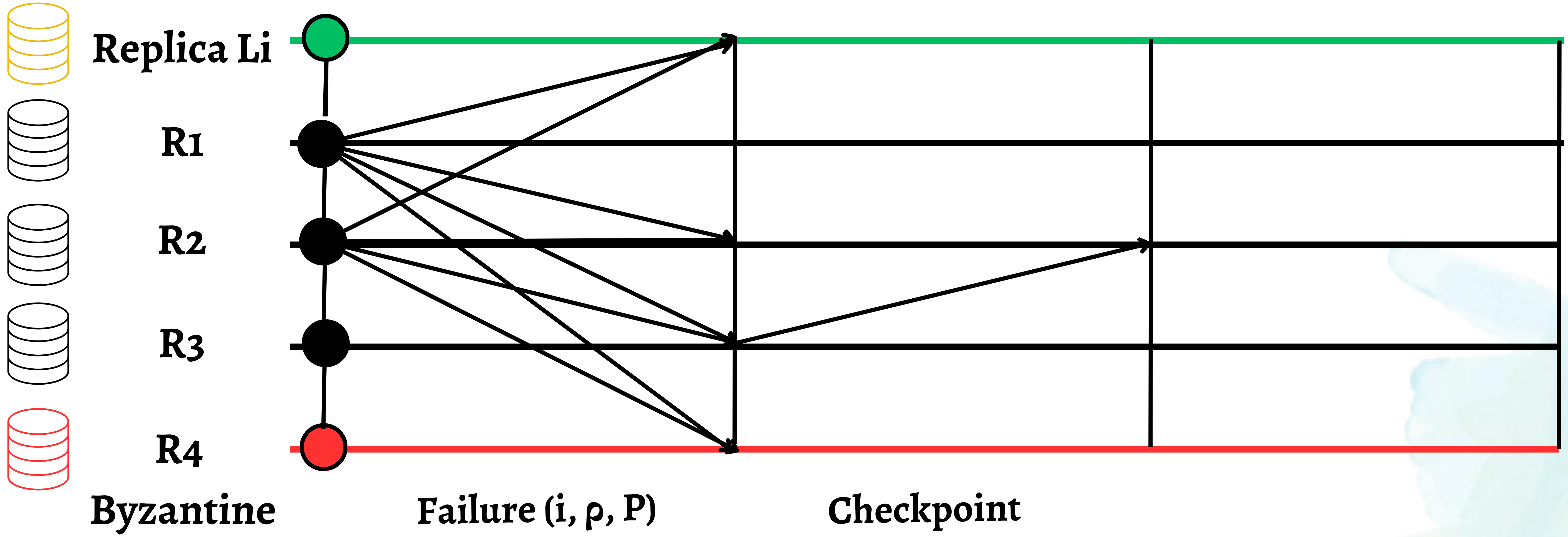
All non faulty replicas need to:

- Detect failure of primary
- Reach agreement on state I (faulty instance)
- Determine the round in which primary can resume operation

Dealing with Undetectable Failures



Dealing with Undetectable Failures



Client Interactions with RCC

- Importance of **distinct client transactions** for optimal throughput.
- It is designed to handle faulty clients without needing their cooperation.

Design Optimization

- RCC optimized for scenarios with many more clients than replicas.
- Each client is assigned to a single primary (Pi).
- Only (Pi) can propose transactions for that client (c)

Client Interactions with RCC

Two Key Challenges

Primaries not receiving client requests

- **Problem**: Less concurrent clients than replicas.
- **Solution**: Propose small "no-op" requests in such situations.

Faulty Primaries

- **Problem**: Faulty primaries refusing to propose some clients' requests.
- **Solution**: Incentivize Malicious primary to not refuse service and detect primary failure.

Client Interactions with RCC

- **Handling Unwilling or Incapable Primaries**

  Situation: Primary (Pi) is unwilling or crashes.

  Solution: Client (c) requests reassignment to another instance (Ij) by broadcasting a request (m := SWITCHINSTANCE(c, j)) to all replicas.

Improving Resilience of Consensus

What are the challenges in Traditional Consensus Protocols?

- Heavy Reliance on the primary
- Designed to handle primaries that completely fail to propose client transactions.
- Not effectively designed to handle various forms of malicious behavior..

Improving Resilience of Consensus

Ordering Attack

T1: Transfer (Alice, Bob, 3)
T2: Transfer (Bob, Eve, 2)

Alice	Bob	Eve
8	0	0

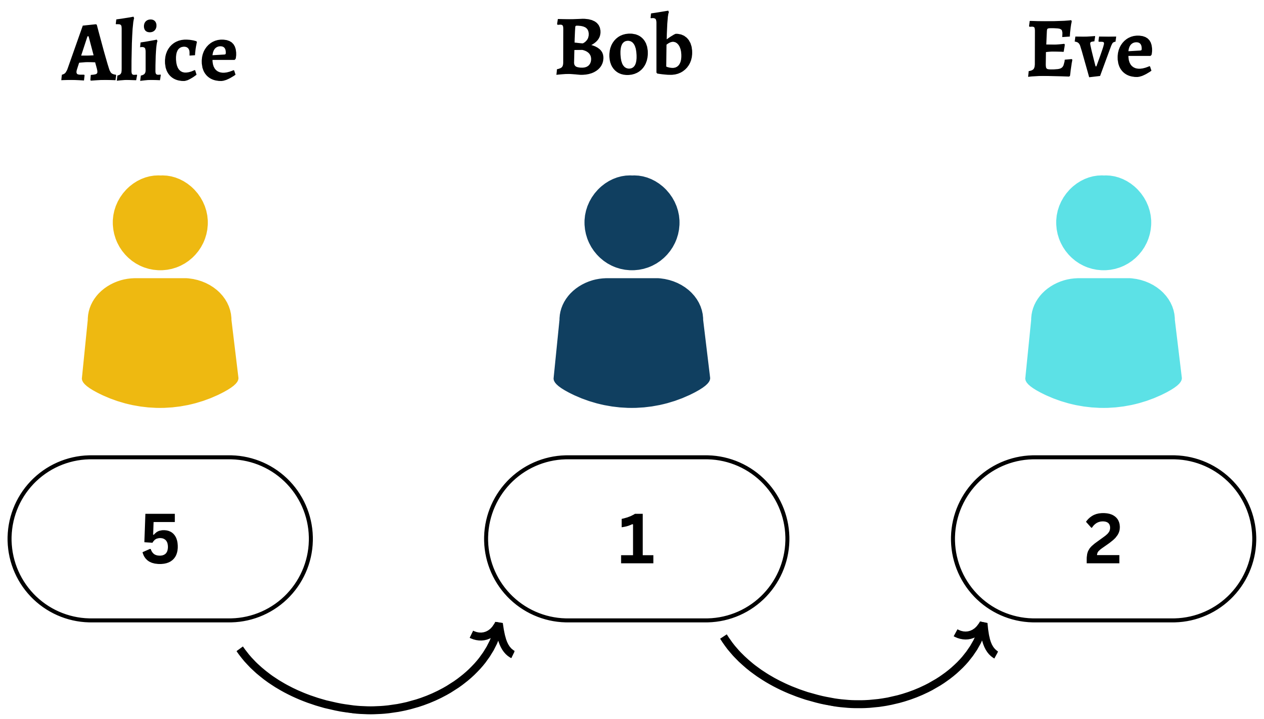
Transfer (A,B,m) -> **If** bal(A)> m
then
bal(A) = bal(A) - m
bal(B) = bal(B) + m

Case 1: T1 then T2

Improving Resilience of Consensus

Ordering Attack

T1: Transfer (Alice, Bob, 3)
T2: Transfer (Bob, Eve, 2)



Transfer (A,B,m) -> **If** bal(A)> m

then

bal(A) = bal(A) - m

bal(B) = bal(B) + m

Case 1: T1 then T2

Improving Resilience of Consensus

Ordering Attack

T1: Transfer (Alice, Bob, 3)
T2: Transfer (Bob, Eve, 2)

Transfer (A,B,m) -> **If** bal(A)> m
then
bal(A) = bal(A) - m
bal(B) = bal(B) + m

Alice	Bob	Eve
8	0	0

Case 1: T2 then T1

Improving Resilience of Consensus

Ordering Attack

T1: Transfer (Alice, Bob, 3)
T2: Transfer (Bob, Eve, 2)

Alice **Bob** **Eve**

Alice	Bob	Eve
5	3	0

Transfer (A,B,m) -> **If** bal(A)> m

then

bal(A) = bal(A) - m

bal(B) = bal(B) + m

Case 1: T2 then T1

Improving Resilience of Consensus

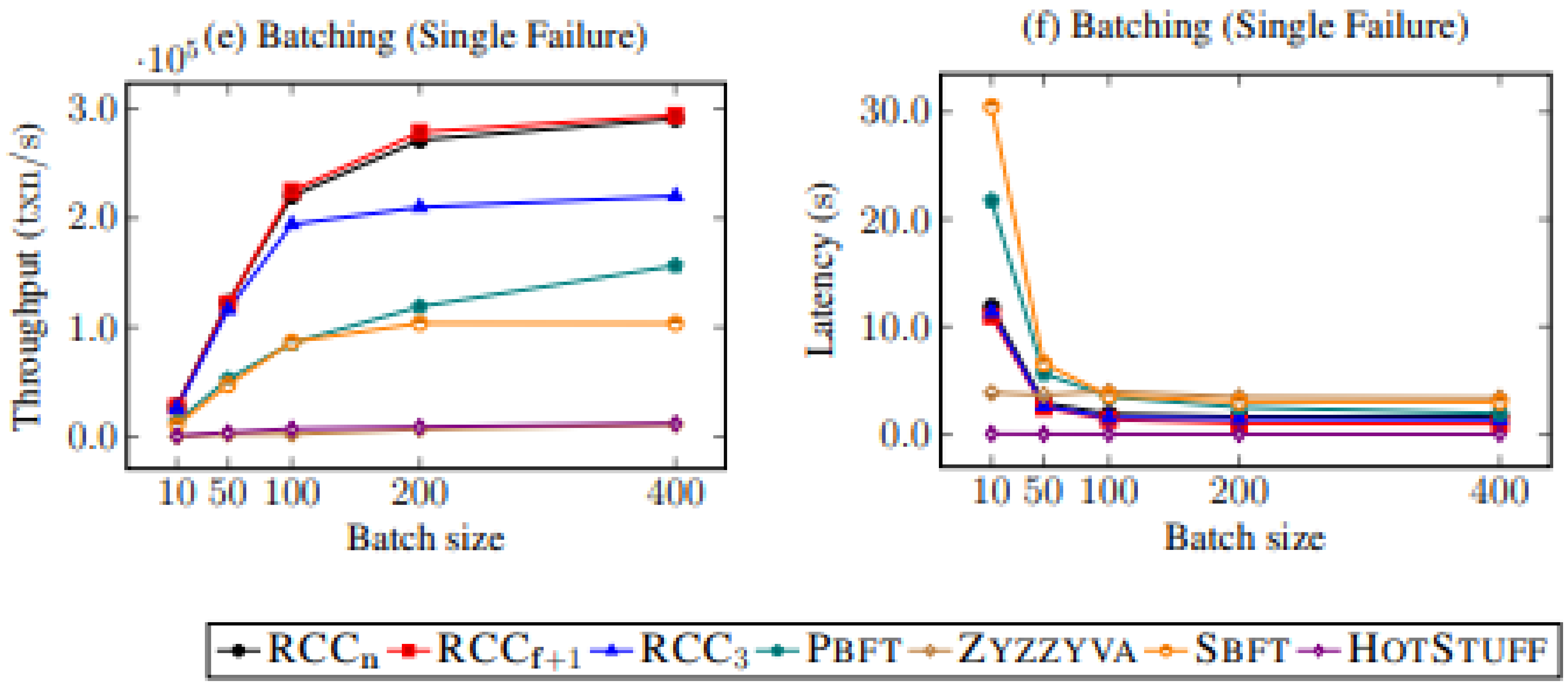
How does RCC mitigate the ordering attack

- A method to deterministically select different permutation of the order of execution in every round.
- The order is practically impossible to predict or influence by faulty replicas.

Evaluation

Varying batch size

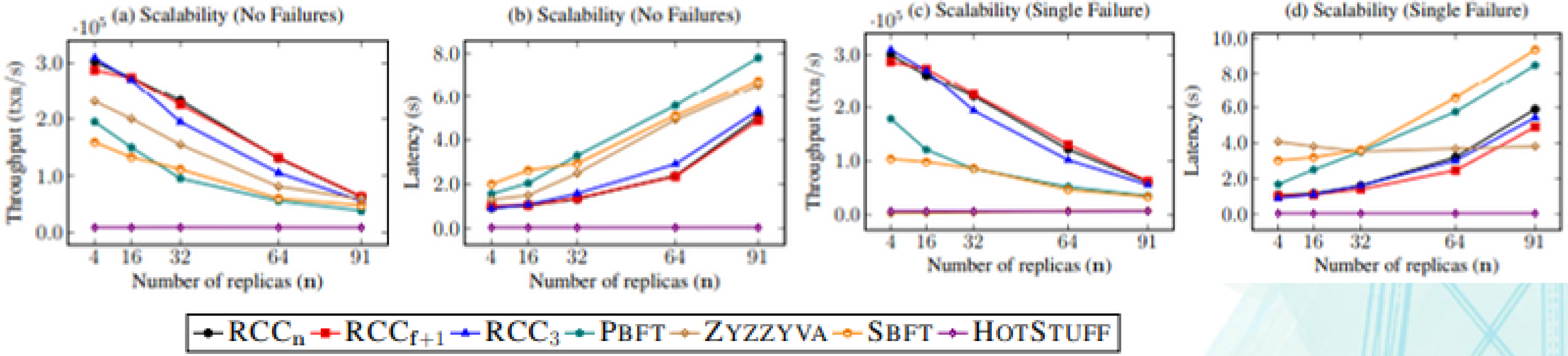
- Performance increases when batch size increases.



Evaluation

Varying Replicas

- Even in the best-case situation of no failures, RCC easily outperforms ZYZZYVA.
- RCC performs better than all other protocols in three versions.
- Performance is enhanced through improving concurrency by adding more instances.



Conclusion

In RCC,

- Every replica is allowed to be a primary (parallel consensus).
- Enhanced security features are provided.
- A set of ordered client requests are guaranteed.
- More throughput than primary-backup consensus is provided.

References

- S. Gupta, J. Hellings and M. Sadoghi; "RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing"; IEEE 37th International Conference on Data Engineering (ICDE); Chania; Greece; 2021

Thank You !