RCC: Resilient Concurrent Consensus

for High-Throughput Secure Transaction Processing

Author: *Suyash Gupta, Jelle Hellings, Mohammad Sadoghi*

Angela Yoon
Fuhsin Liao
Yin-Kai Huang
Jinho Lee

Contents

01 Introduction

- Consensus-Based System
- Limitations of Traditional Consensus System
- Solution: Concurrent Consensus

02 Resilient Concurrent Consensus

- Introduction of RCC
- RCC Paradigm
- Detectable failures
- Undetectable failures
- Client Interactions

03 Improvement

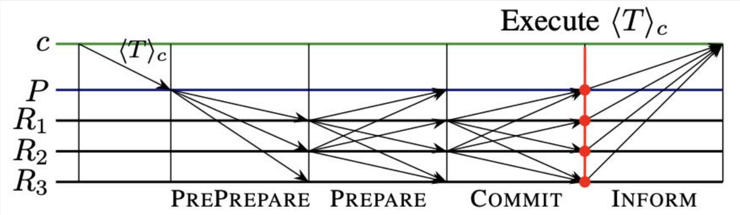
- Security Challenges & Mitigations

04 Evaluation and Conclusion

Consensus-Based System

- More resilience during failures
- Strong support for data provenance
- Enables federated data processing

- Prevent disruption of service
- Help improve data quality



Limitations of Traditional Consensus

- **Bottleneck of the primary's outbound bandwidth**
- **Non-primary replicas underutilize their resources**

$$T_{max} = \frac{B}{(n-1)st}$$

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

$$T_{max} \approx T_{PBFT}$$

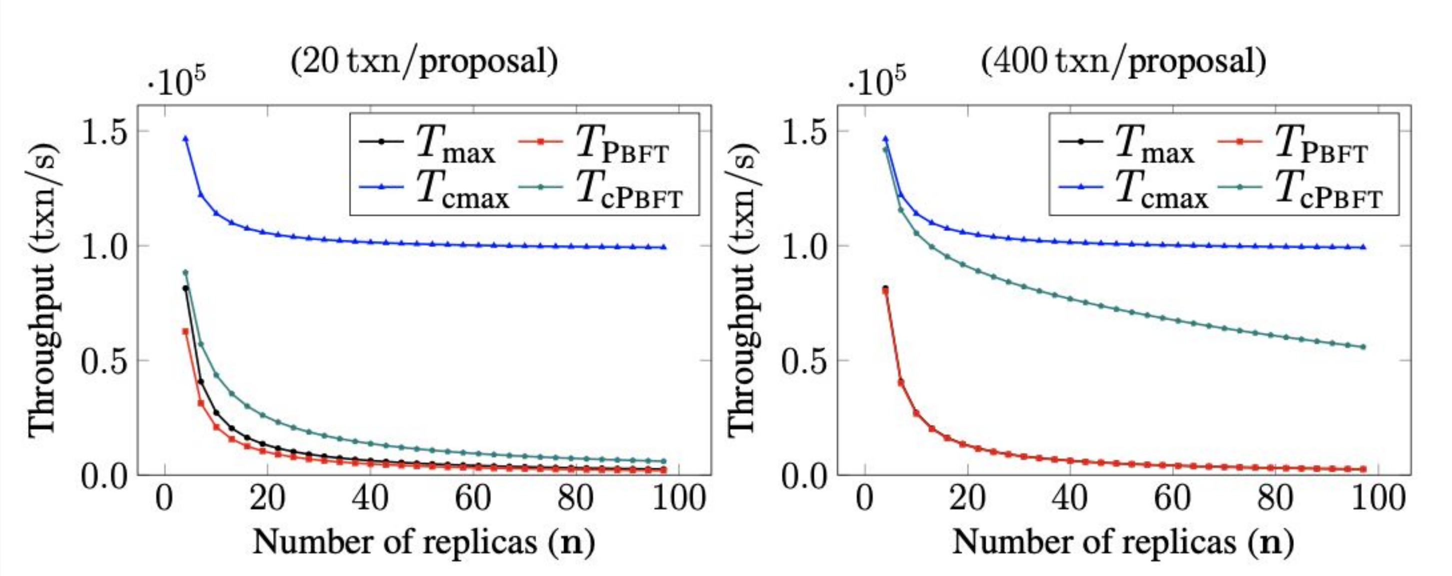
when size of transaction(st) >> size of message(sm)

Solution?

Concurrent Consensus

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

$$T_{cPBFT} = nf \frac{B}{(n-1)(st + 3sm) + (nf-1)(st + 4(n-1)sm)}$$



How do we design a satisfying concurrent consensus?

RCC

What is RCC?

- short for **R**esilient **C**oncurrent **C**onsensus
- RCC is a paradigm that can turn any primary-backup consensus into a concurrent consensus

Basic Ideas of RCC

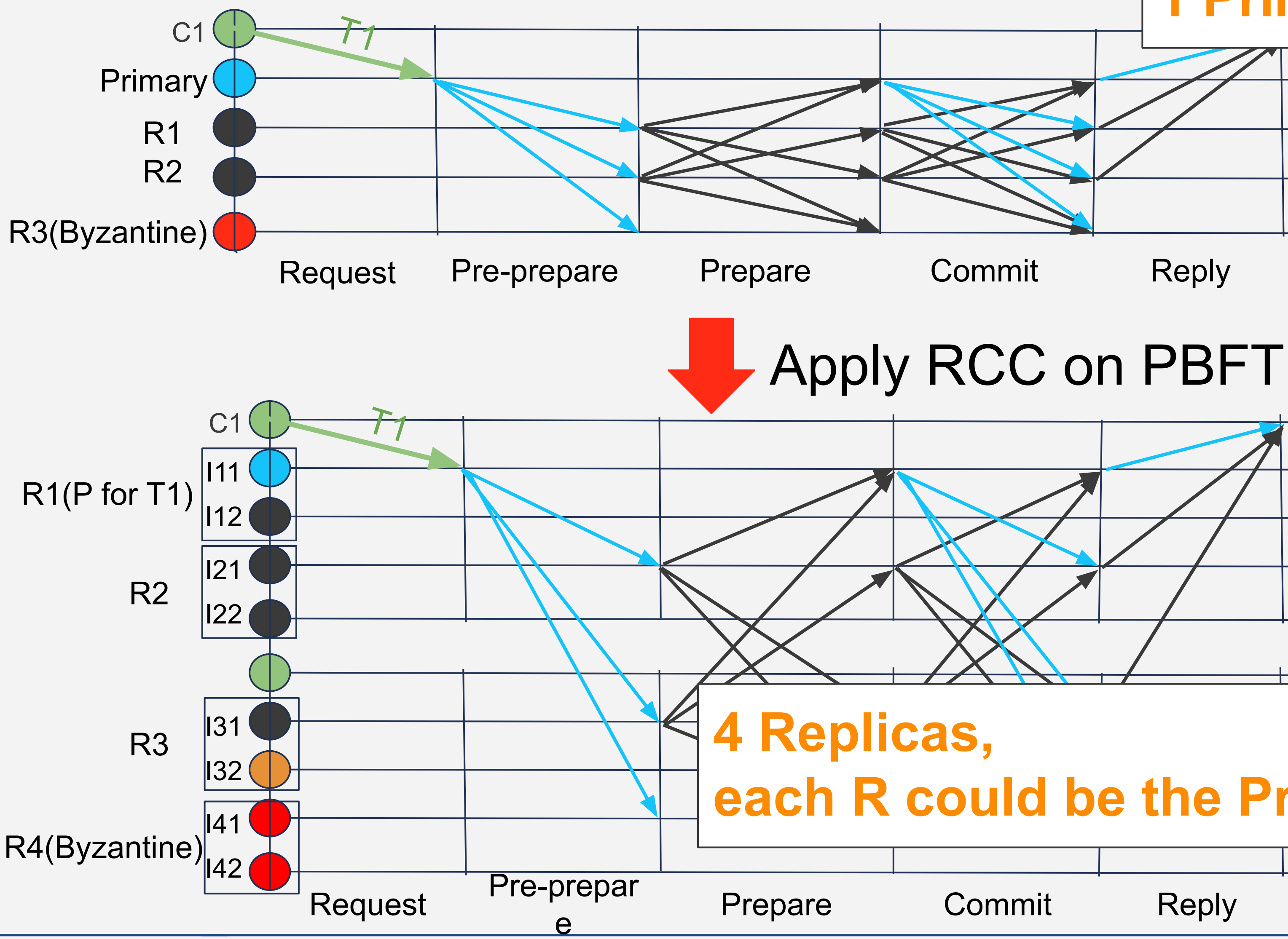
- **Idea:**
  1. **Making every replica a primary node**
  2. **Primary nodes can propose transactions simultaneously**
  3. **Every replica have multiple instances to handle different transactions**

RCC Design Goals

1. RCC guarantee the order of the client request
2. Clients can interact with RCC to force execution of their transactions
3. RCC can be applied to any primary-backup consensus protocol
4. **Non-faulty primaries are always able to propose transactions at** maximum throughput and won't be affected by faulty replica
5. Dealing with faulty primaries does not interfere with operations of other consensus-instances

Take PBFT as an example

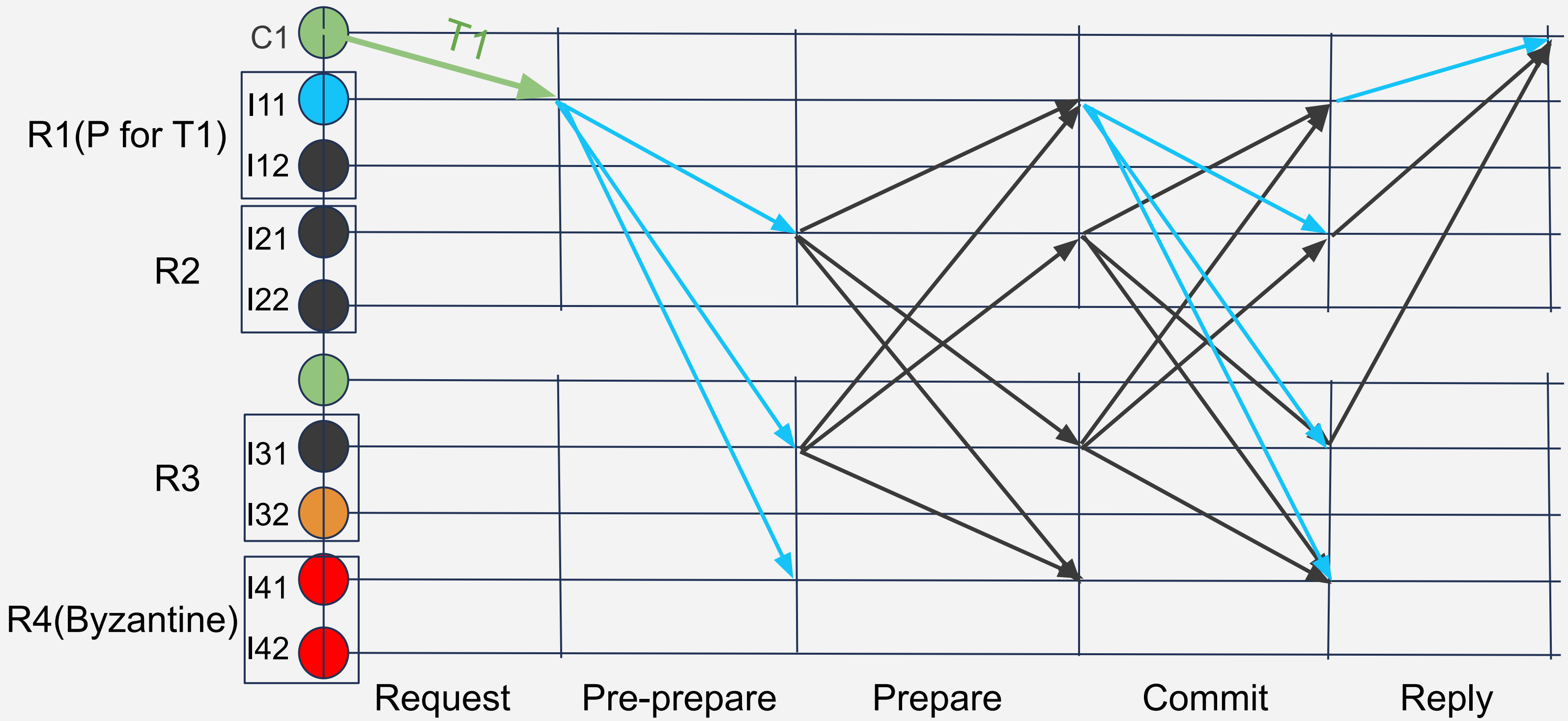
3 Replicas + 1 Primary!



4 Replicas, each R could be the Primary!

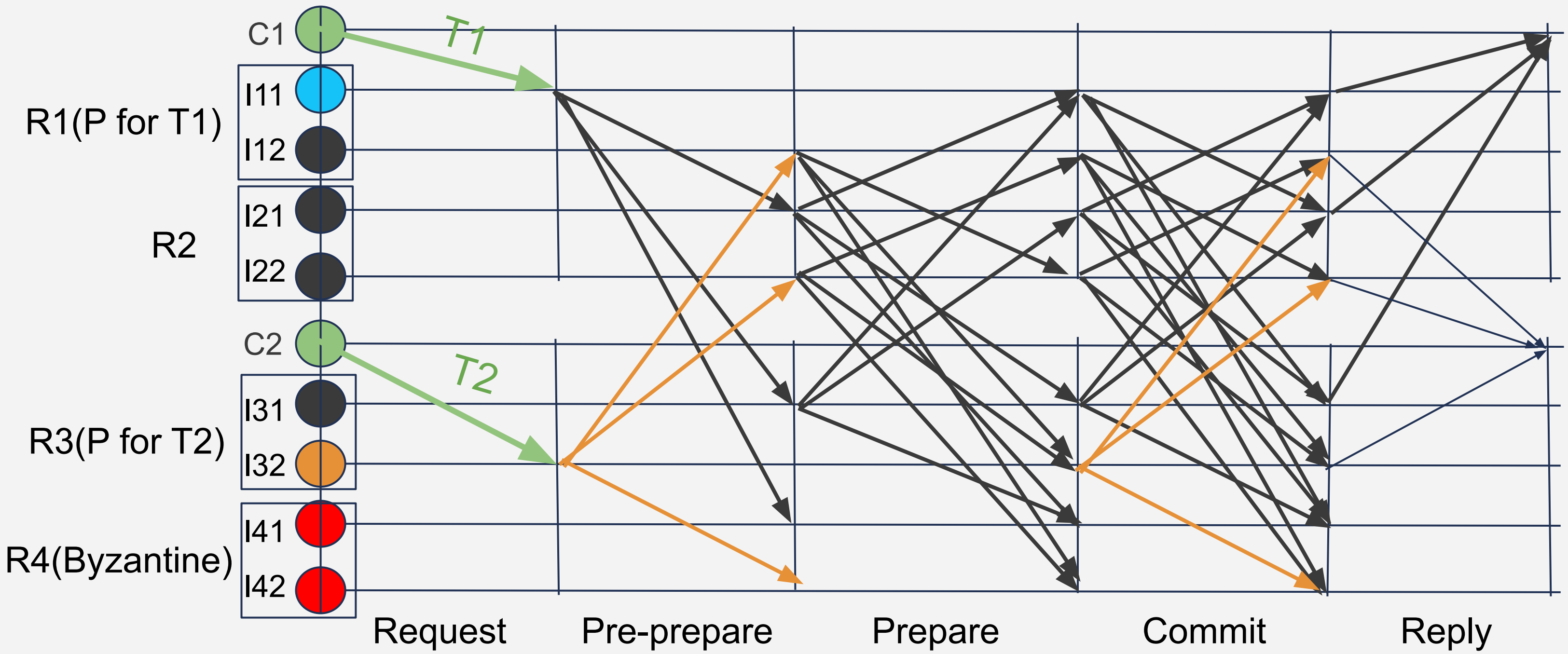
Take PBFT as an example(cont.)

The first transaction T1 comes in, R1 is the primary for T1



Take PBFT as an example(cont.)

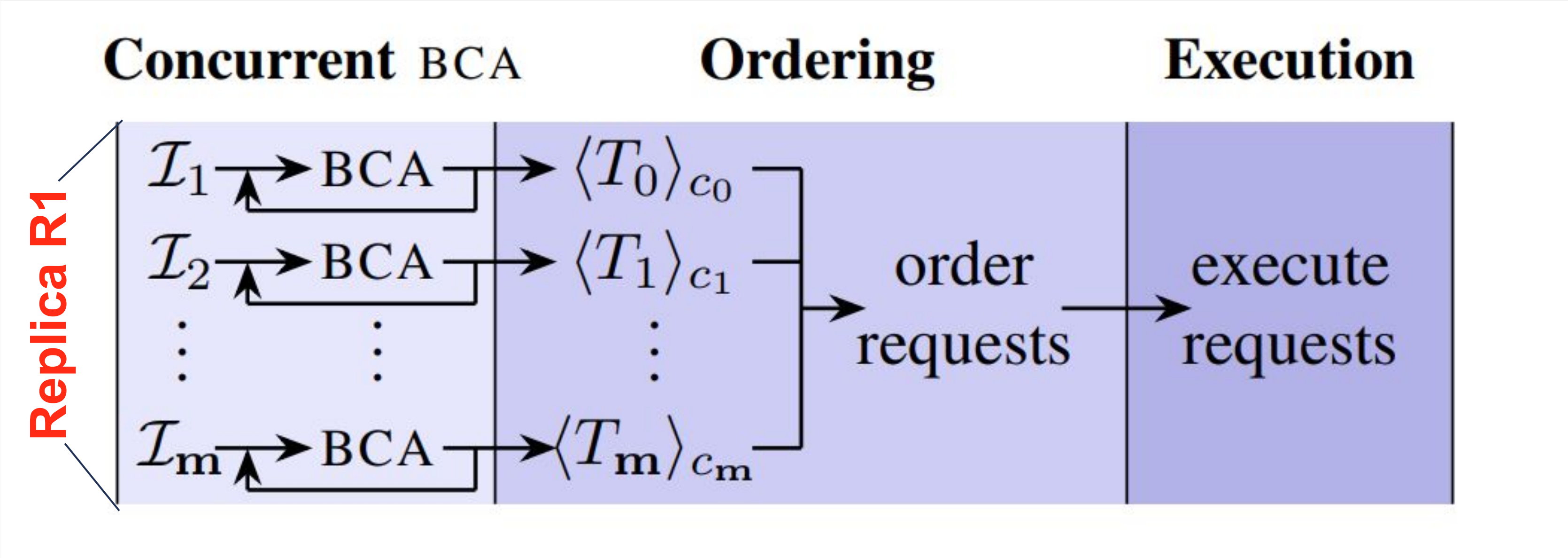
The second transaction T2 comes in, R3 is the primary for T2



High Level RCC Paradigm

If we have m transactions and m replicas:

- Each replica in each round will have m instances participate in m BCA (e.g. PBFT with pre-prepare-prepare-commit)



Detectable Failures

Detectable Failures

- **Primary does not send out proposals**
- **Proposals get lost in the network.**

- **Non-faulty replicas do not receive proposals from a primary**

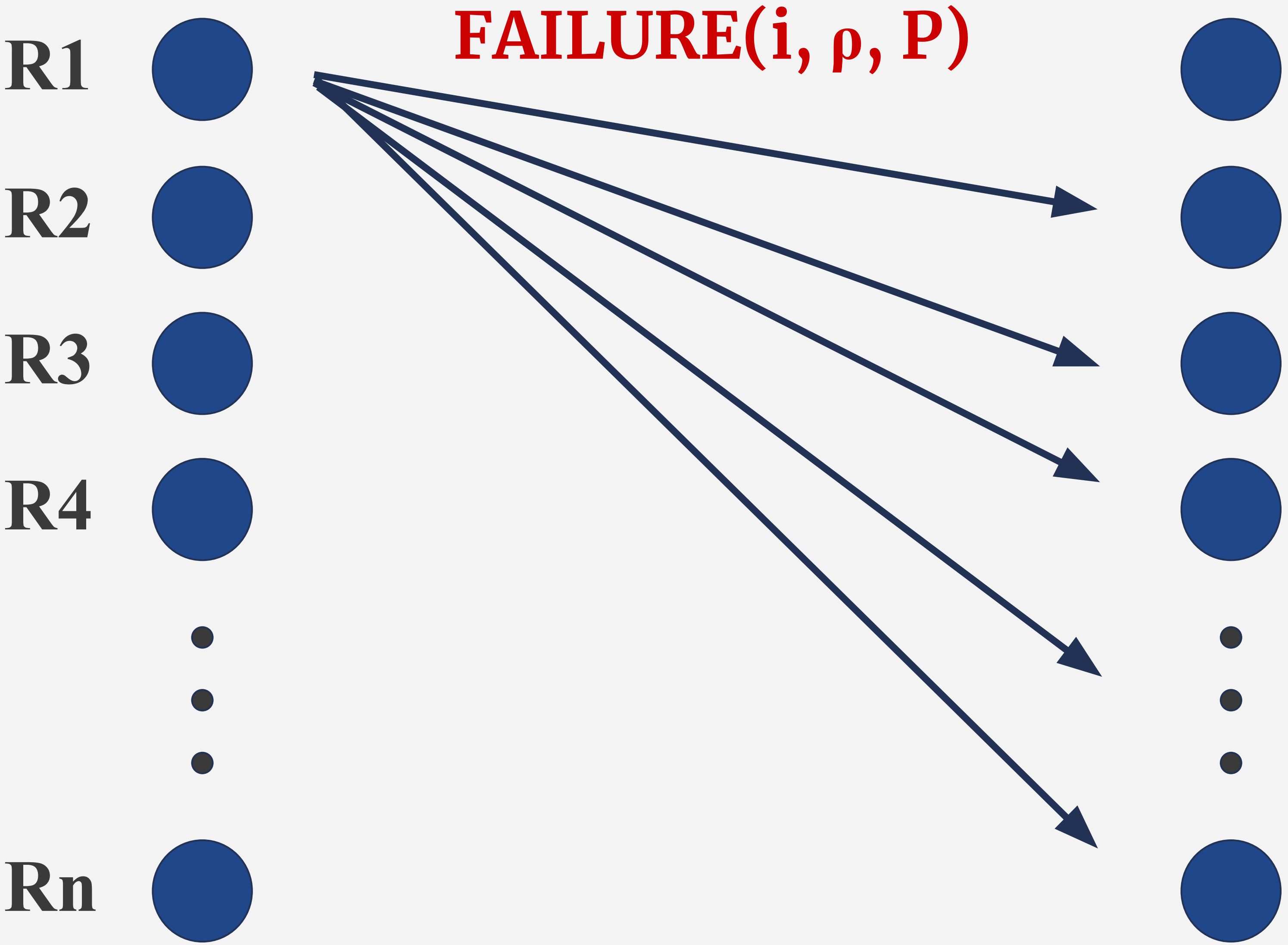
UC DAVIS
UNIVERSITY OF CALIFORNIA

Detectable Failures

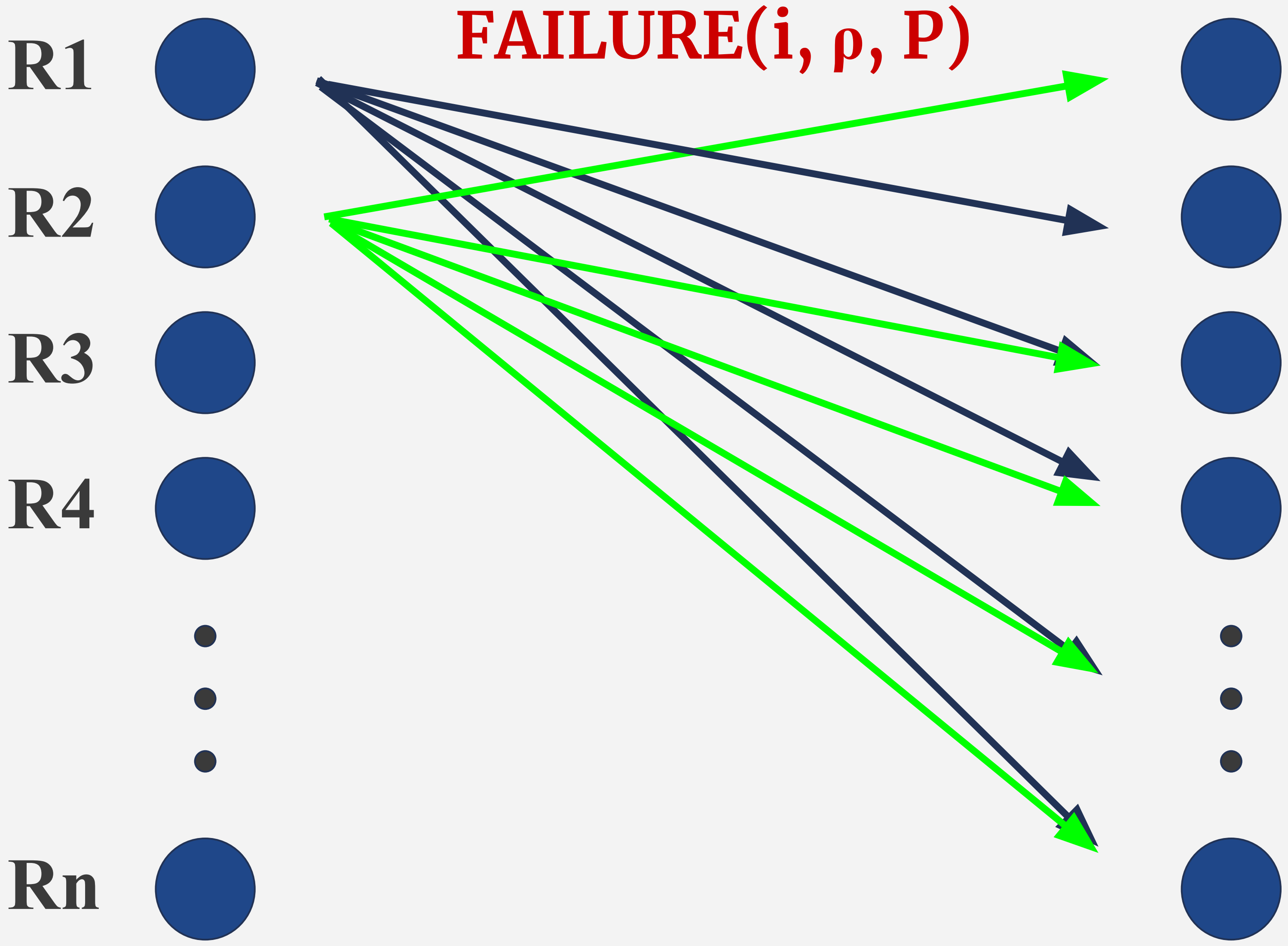
- Assume that primary P1 of I1 fails
- Recovery process

1. **Detect failure** of the primary P1
2. **Reach agreement** on the state of I1
3. **Determine the round** in which P1 can resume its operations

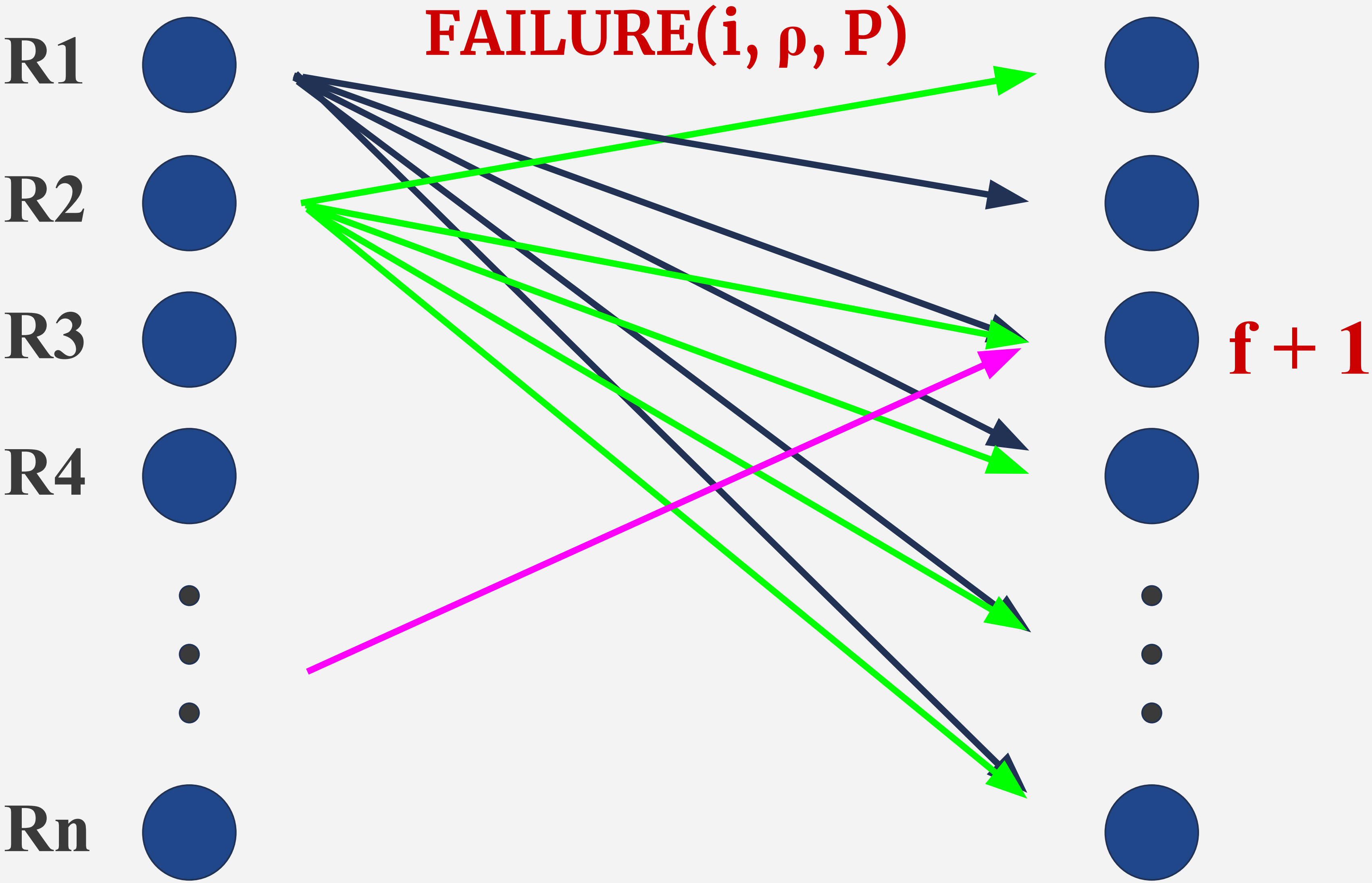
Detect Failures



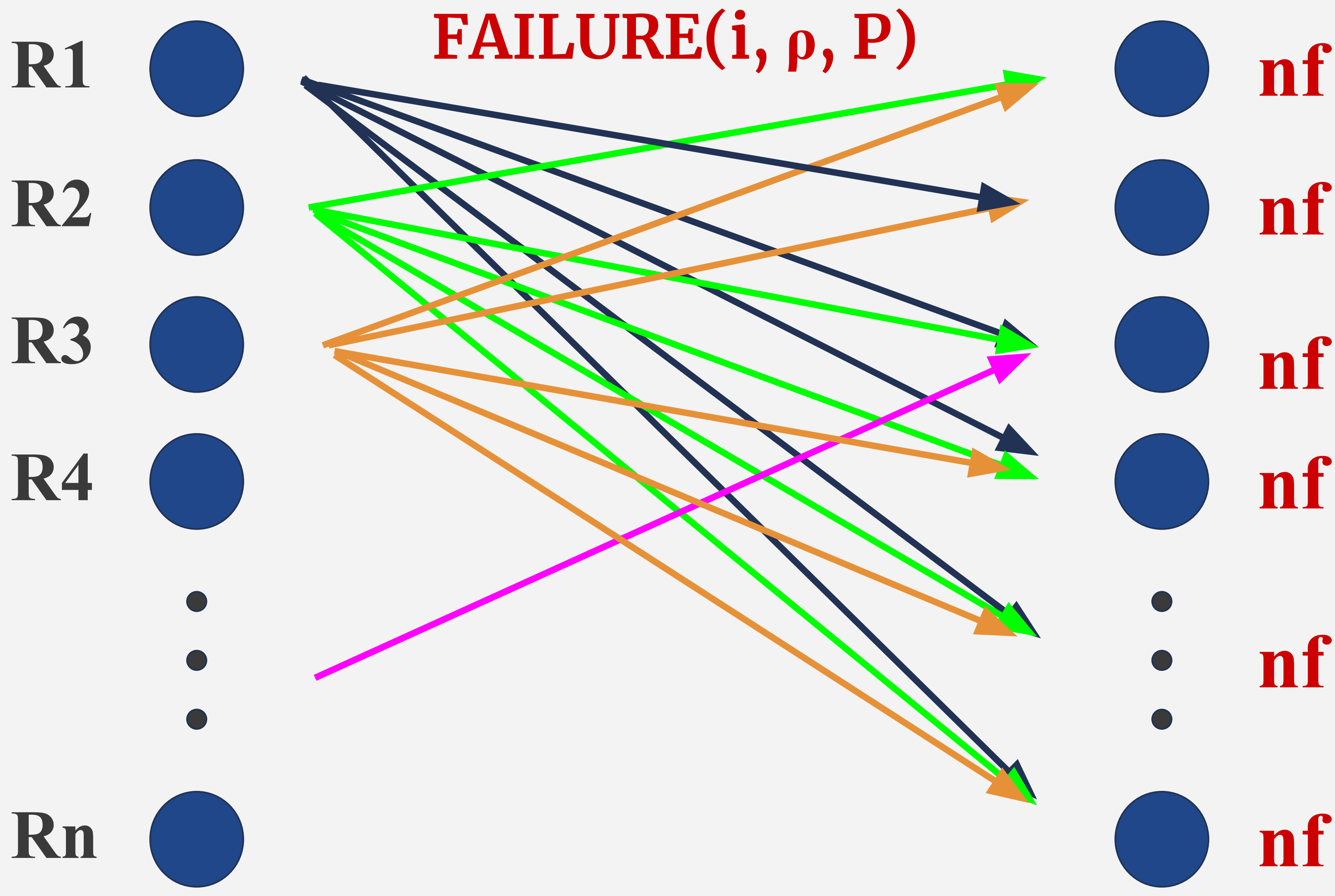
Detect Failures



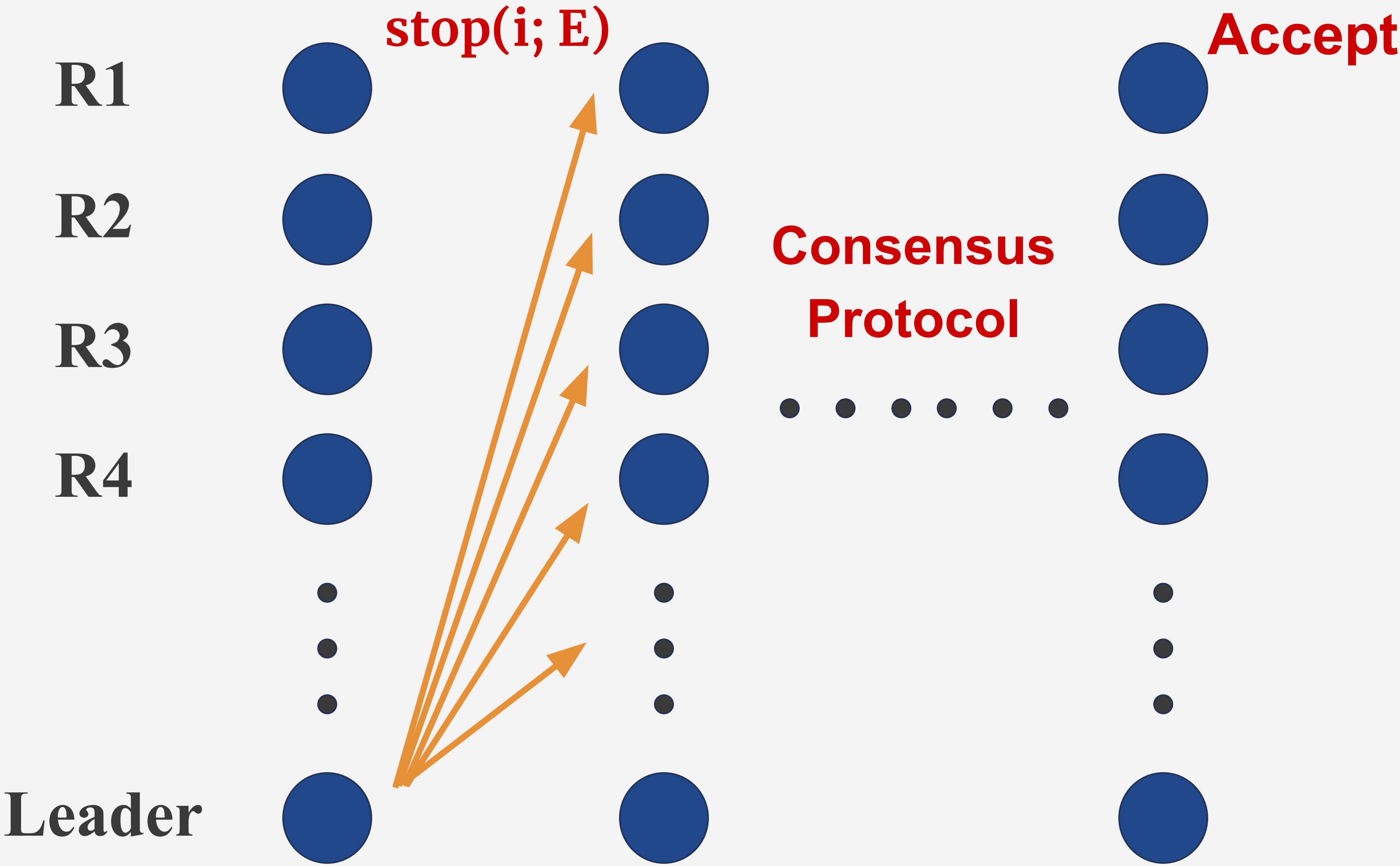
Detect Failures



Detect Failures



Reach Agreement

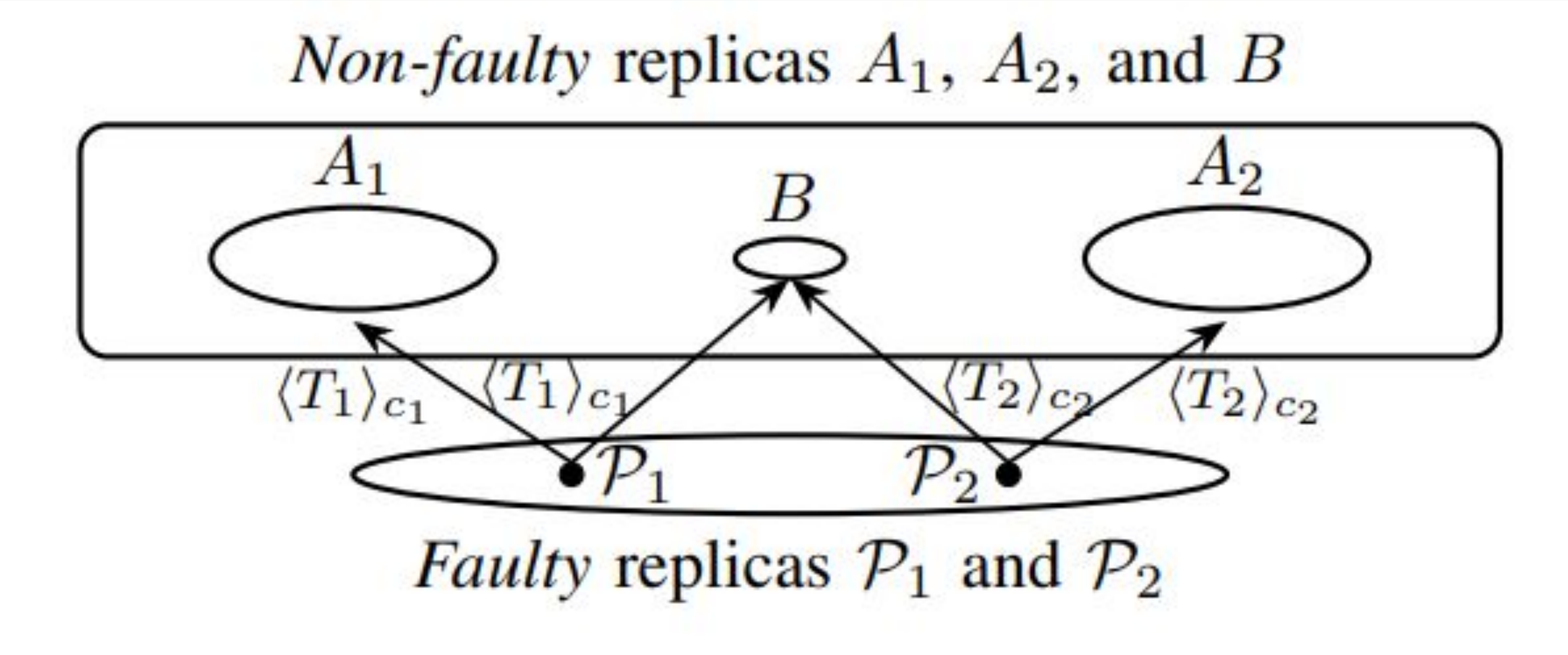


Reach Agreement

- **Each replica can recover to a common state of Instance**
- **Set ρ + 2^f as the next valid round number for Instance.**

Undetectable Failures

Undetectable Failures

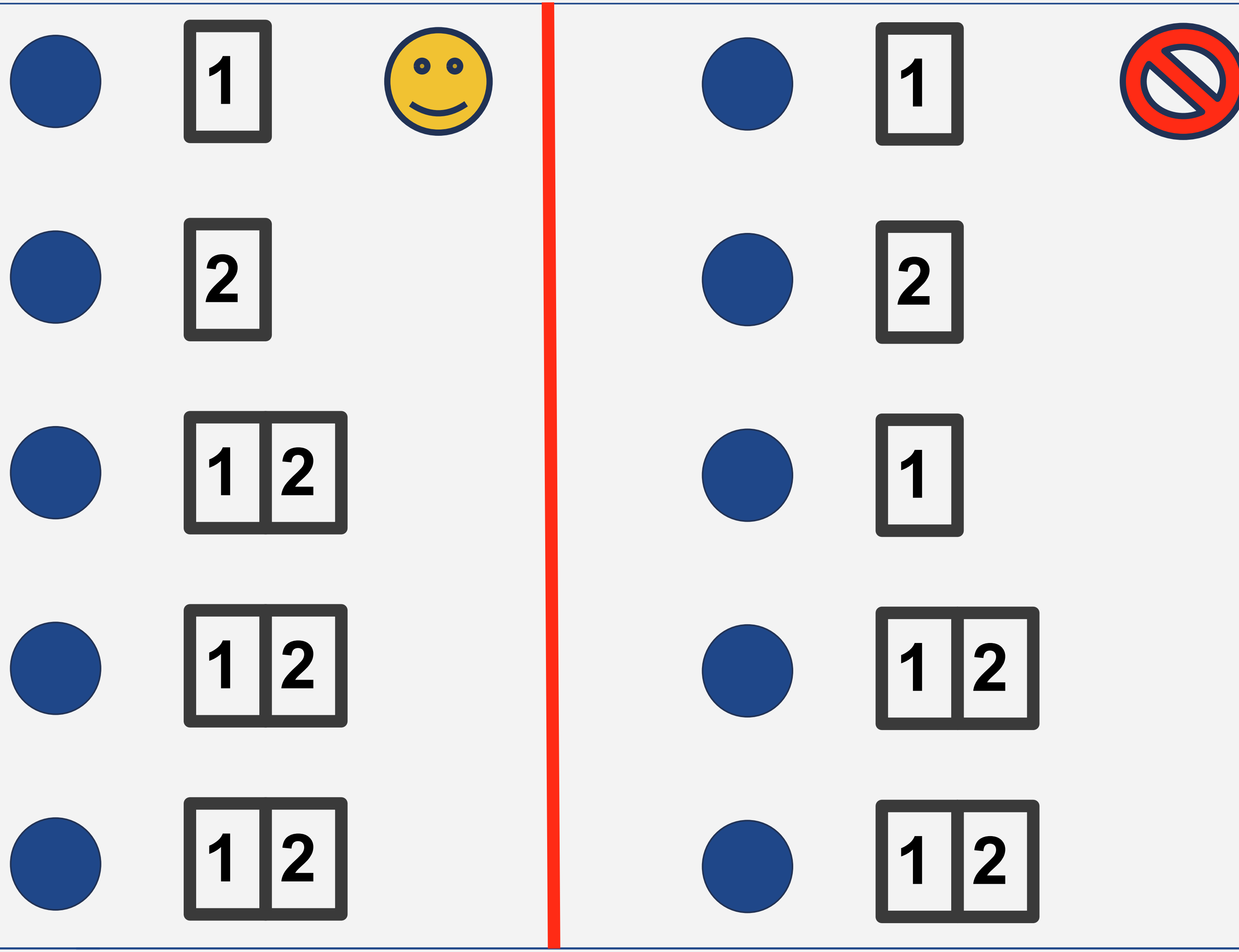


From: RCC: Resilient Concurrent Consensus for High-Throughput Secure Transaction Processing. ICDE'21

Solution

- **Run a standard checkpoint algorithm for each BCA instance**
- **Reduce the cost of checkpoints**
  - **Do checkpoints when a replica receives f+1 claims of failure of all primaries**

Solution



Client Interactions

Client Interaction

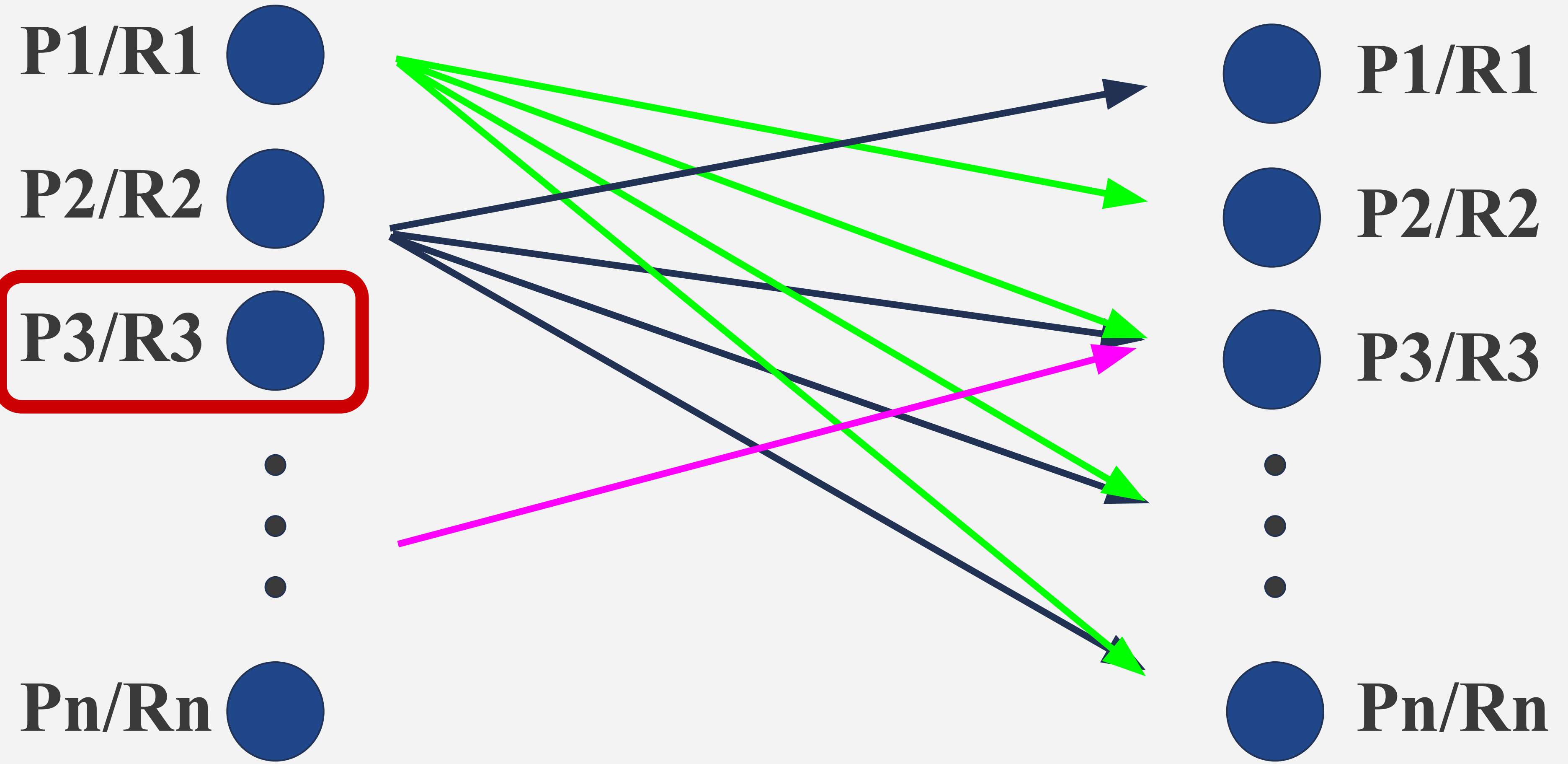
- **Every instance proposes distinct client transactions**
- **More concurrent clients than replicas**

Issues

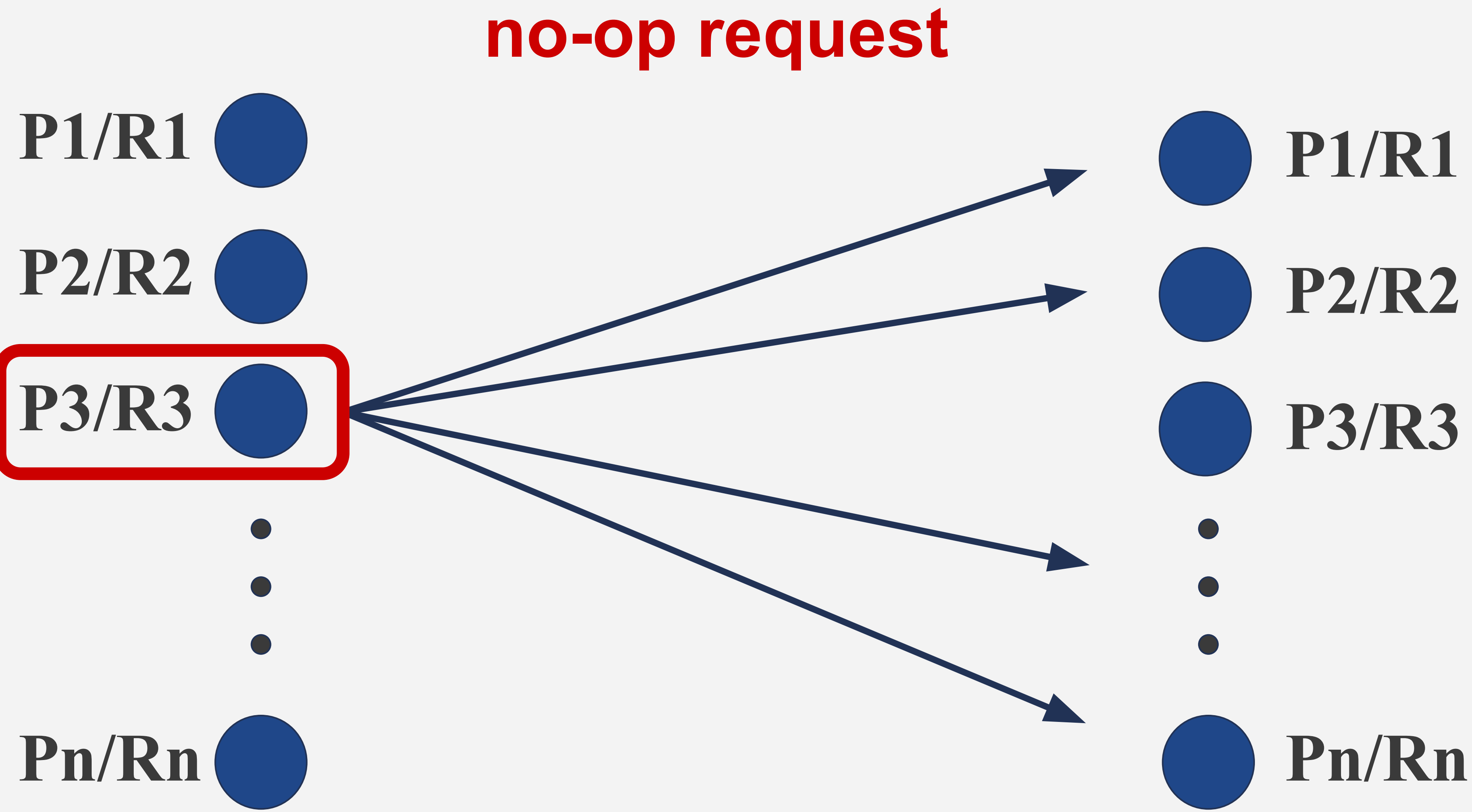
- **Primaries do not receive client requests**
  - **Less concurrent clients than replicas**
- **Faulty primaries refuse to propose requests of some clients**

Solution

- Primaries do not receive client requests

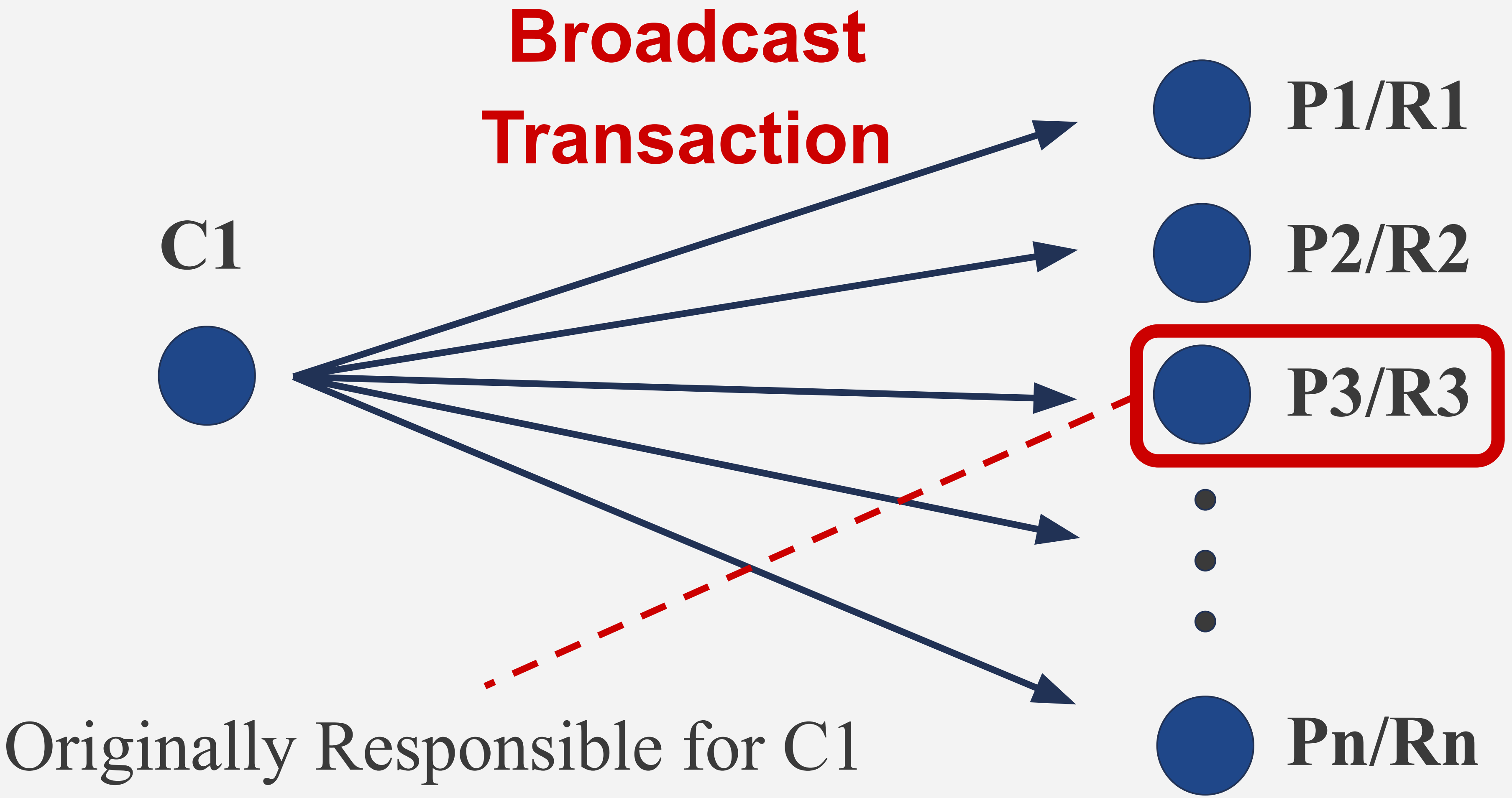


Solution

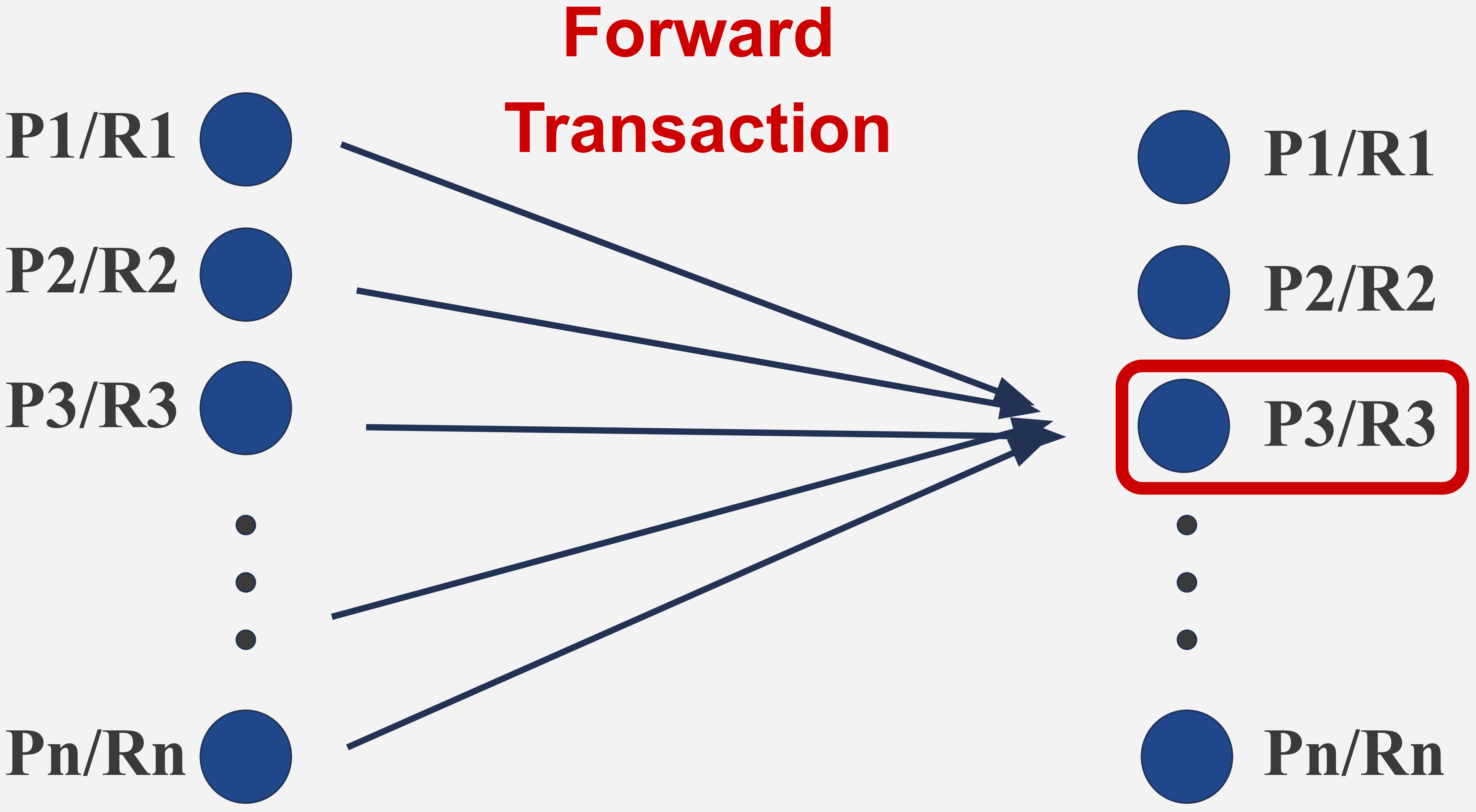


Refuse to propose requests

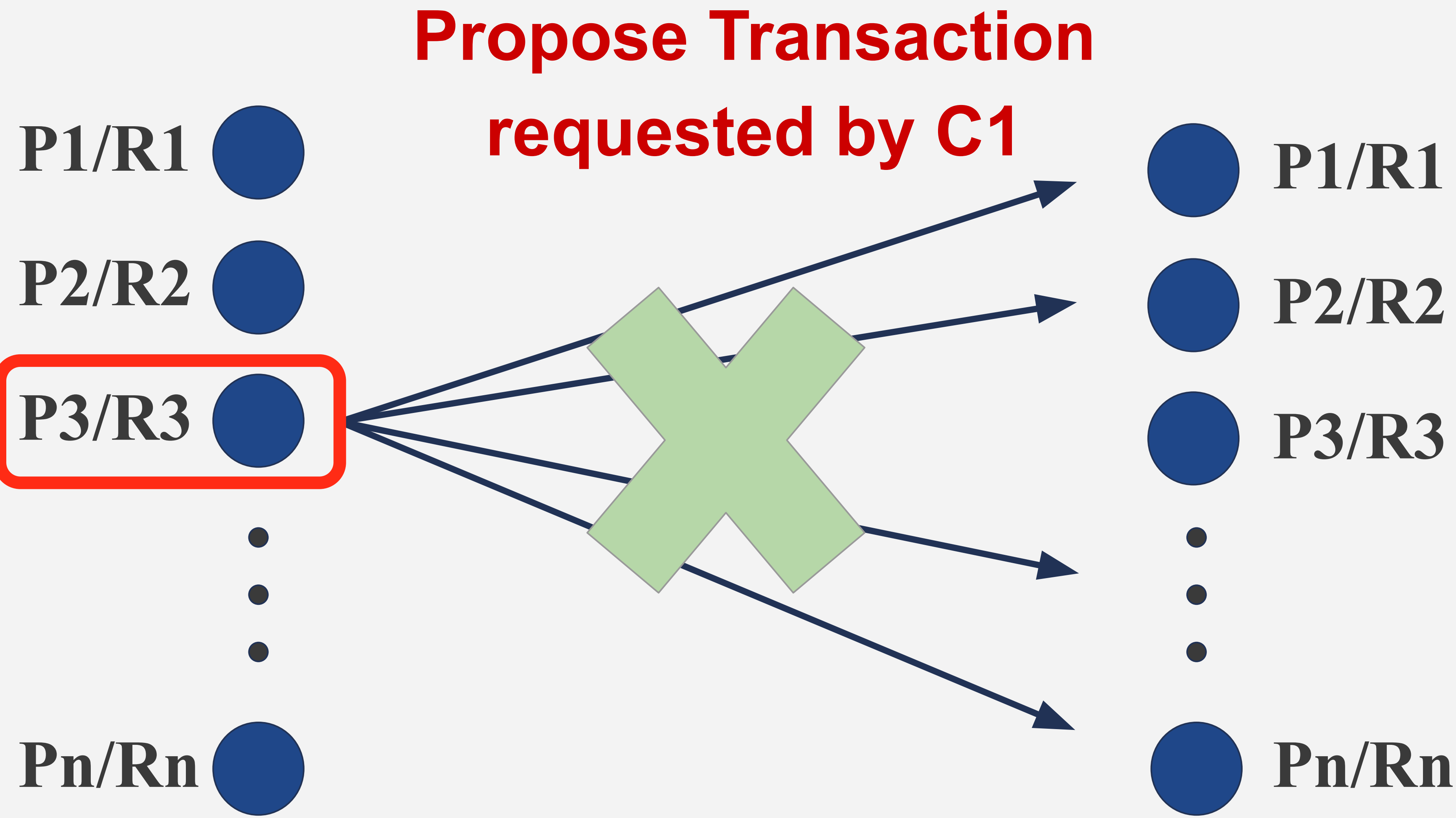
- Detect failure



Detect failure

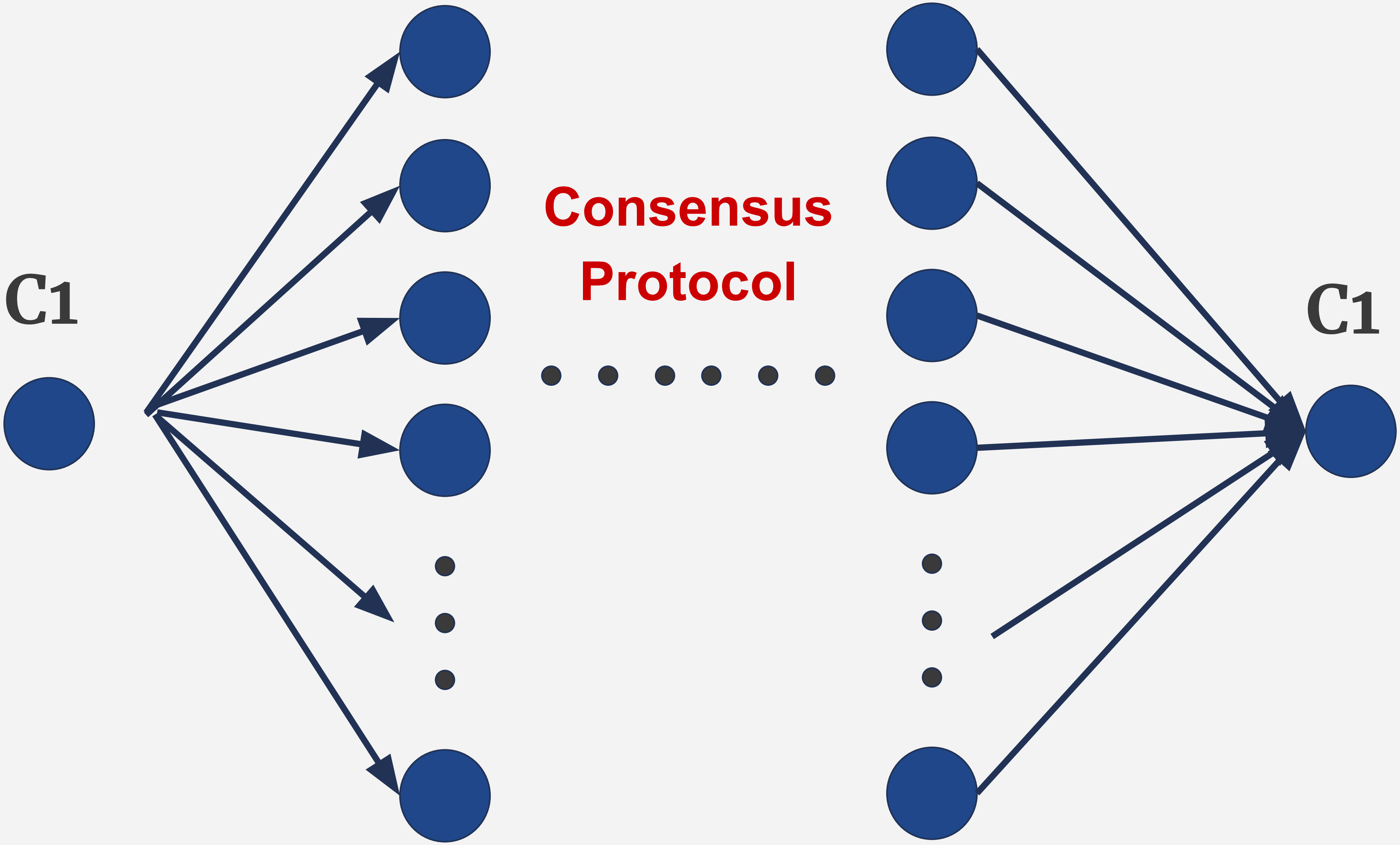


Detect failure



Reassignment

SWITCHINSTANCE(c, j)



Improving Resilience of Consensus
: security challenges

Security challenges

- Traditional primary-backup consensus protocols are designed to deal with **primaries fail**
  - resilient to failure of the primary
- Malicious attacks possible in the consensus protocol
  - Malicious **primary**
    - Ordering attack, Throttling attack
  - Malicious **entities**
    - Targeted attack

Ordering attack

Malicious primary

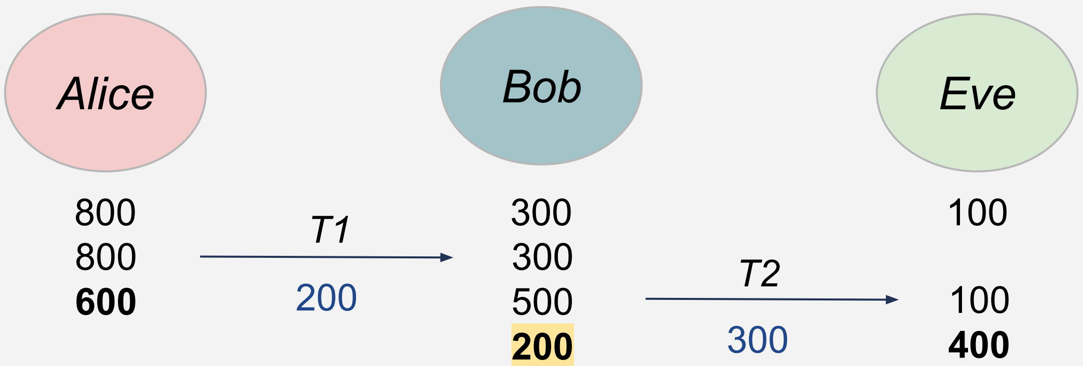
- Primary sets the order for transactions
- Malicious primary can **select the order** that satisfy its own goals

Ordering attack

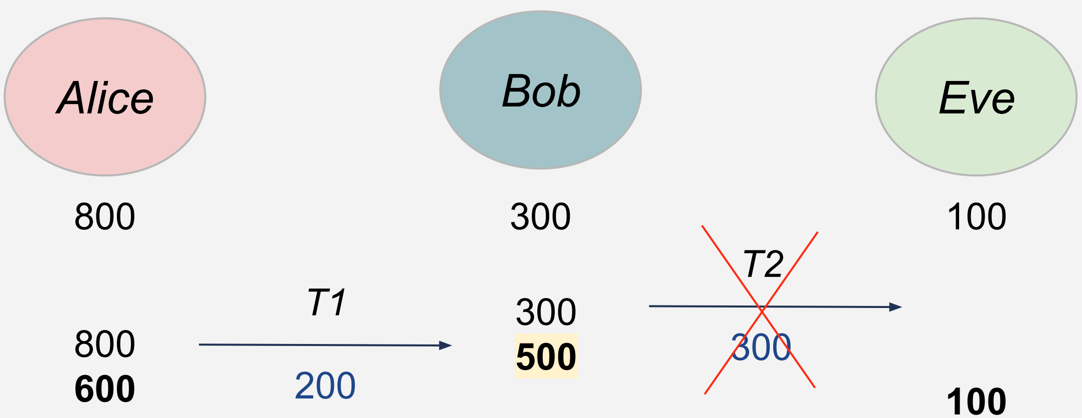
- T1 = transfer(Alice, Bob, 500, 200)
- T2 = transfer(Bob, Eve, 400, 300)

```
transfer(A, B, n, m) := if amount(A) > n
                        then withdraw(A,m);
                             deposit(B,m)
```

Case *T1* then *T2*



Case *T2* then *T1*

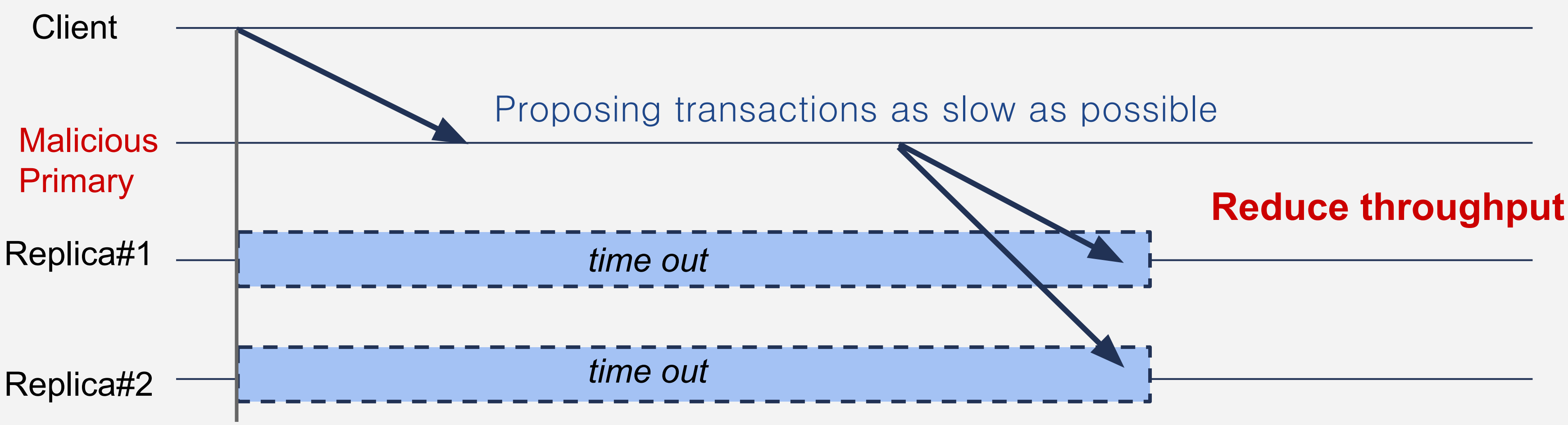


Benefit to *Bob*

Throttling attack

Malicious primary

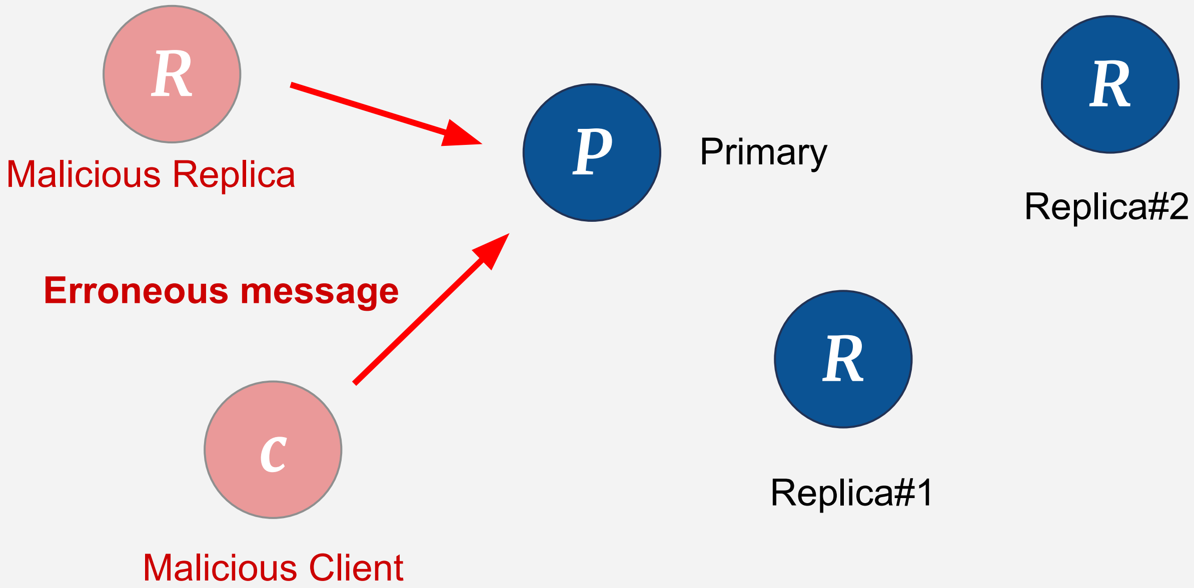
- Replicas rely on **time-outs** to detect malicious behavior of the primary
- Primaries **fail to detect or handle malicious primaries** reduce throughput
  - Proposing transactions as slow as possible
  - Can't detect malicious primary due to **time-outs**



Targeted attack

Malicious entities

- Primary-backup system is entirely determined by **Primary**
  - Attackers send arbitrary messages to the primary
- Primary uses resources to interpret the erroneous message, and throughput decreases
  - Similar to DoS(Denial of Service) attacks



Mitigate attacks

Ordering attacks

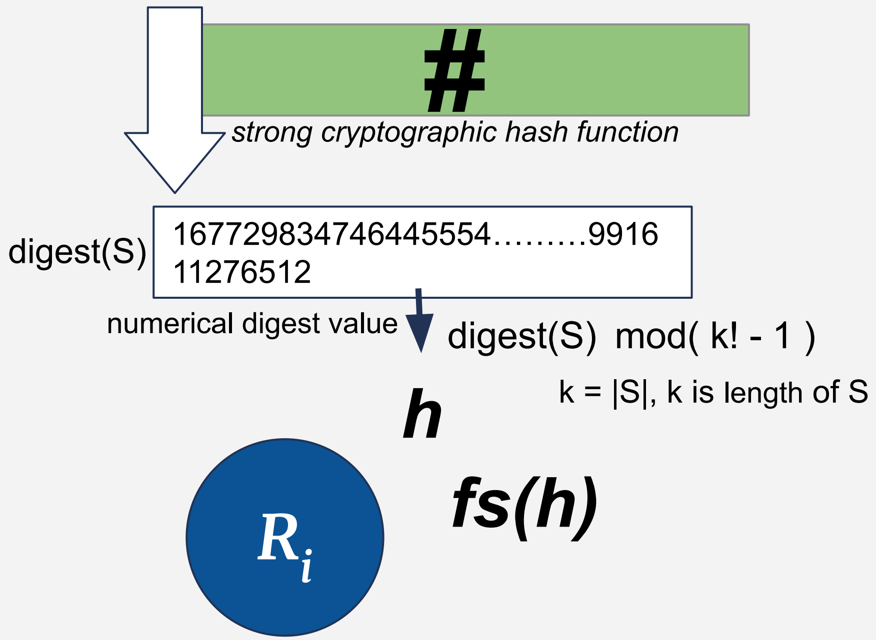
- Concurrent design of RCC can be used to **mitigate** these **attacks**

- Mitigate **ordering attacks**
  - Method to deterministically choose a **different permutation** of the **execution order** in every round
  - Ordering is almost impossible to predict or influence by faulty replicas

Mitigate attacks

Ordering attacks

sequence of all transactions $S = \{ <T_1>, <T_2>, <T_3>, \dots, <T_m> \}$



- final value ***h*** is **only known** after completion of each round
- After select ***h***, all replicas execute the transactions in ***S*** in the order given by $fs(\mathbf{h})$

Mitigate attacks

Ordering attacks

sequence of all transactions $S = \{ <T_1>, <T_2>, <T_3>, \ldots, <T_m> \}$

\#

strong cryptographic hash function

digest(S) 1677298347464455 54.........9916 11276512

numerical digest value

digest(S) mod(k! - 1)

k = |S|, k is length of S

h

R_i

fs(h)

modified sequence

$S' = \{ <T_1>, <T_3>, <T_2>, \ldots, <T_m> \}$

\#

strong cryptographic hash function

626934618456723.........5293729 1257499083878

digest(**S'**) mod(k! - 1)

h'

R_i

faulty replica

fs(h')

- final value ***h*** is **only known** after completion of each round
- After selecting ***h***, all replicas execute the transactions in ***S*** in the order given by $fs(h)$

- practically impossible to find another value S′, S ≠ S′, with *digest(S) = digest(S′)*

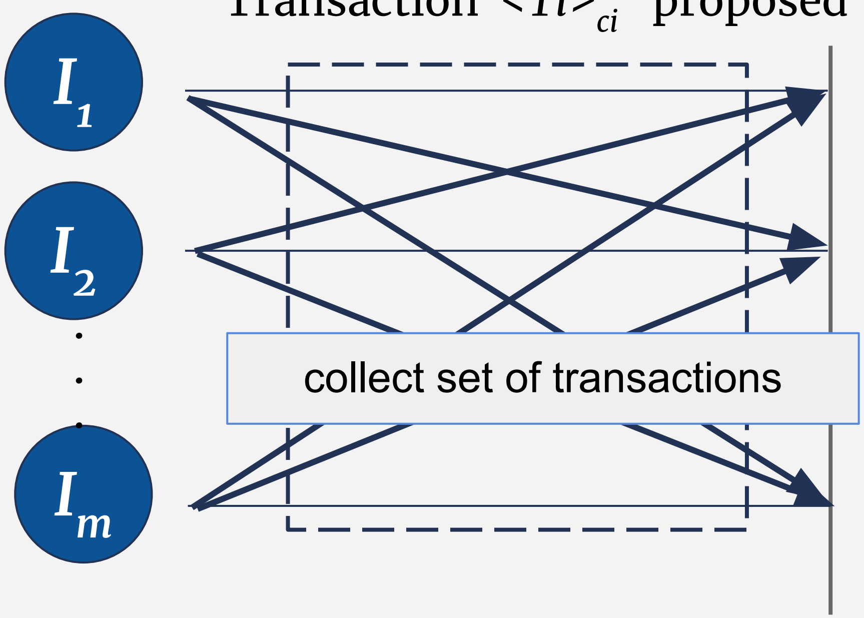
Mitigate attacks

throttling attacks

- Mitigate **throttling attacks**
  - If any other instances lag behind the constant rounds set by the system, Replica detects the failure

***Concurrent BCA* Step**

Transaction $<Ti>_{ci}$ proposed by Pi in round p



- Set constant round : σ
- Current maximum round : $\rho(m, R)$

if *Ii* round value lag behind *p(m, R)* - σ then Replica detects failure of corresponding Primary

Mitigate attacks

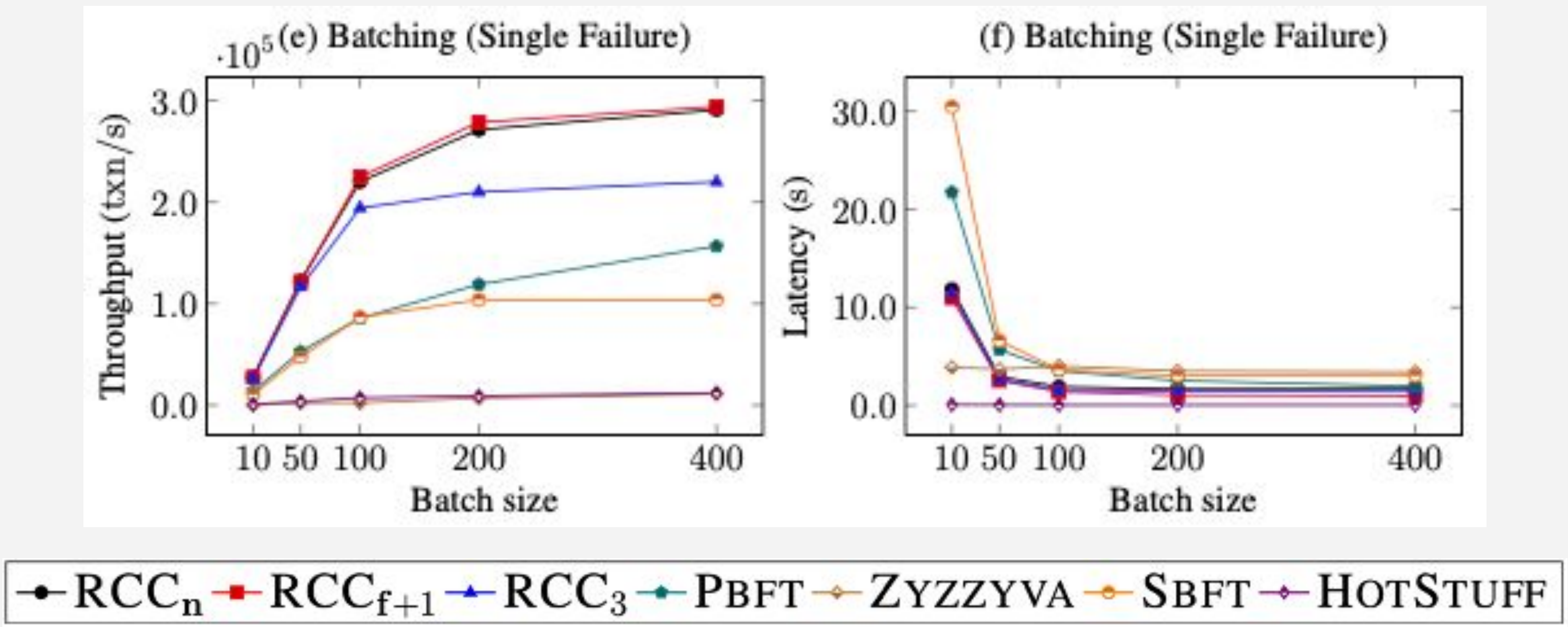
- Mitigate ***targeted attacks***
  - RCC provides *load balancing* for tasks of the primary
    - Reduce load on singe primary
  - RCC distributed the total workload of the system to multiple primaries

Evaluation

Evaluation

Varying batching size

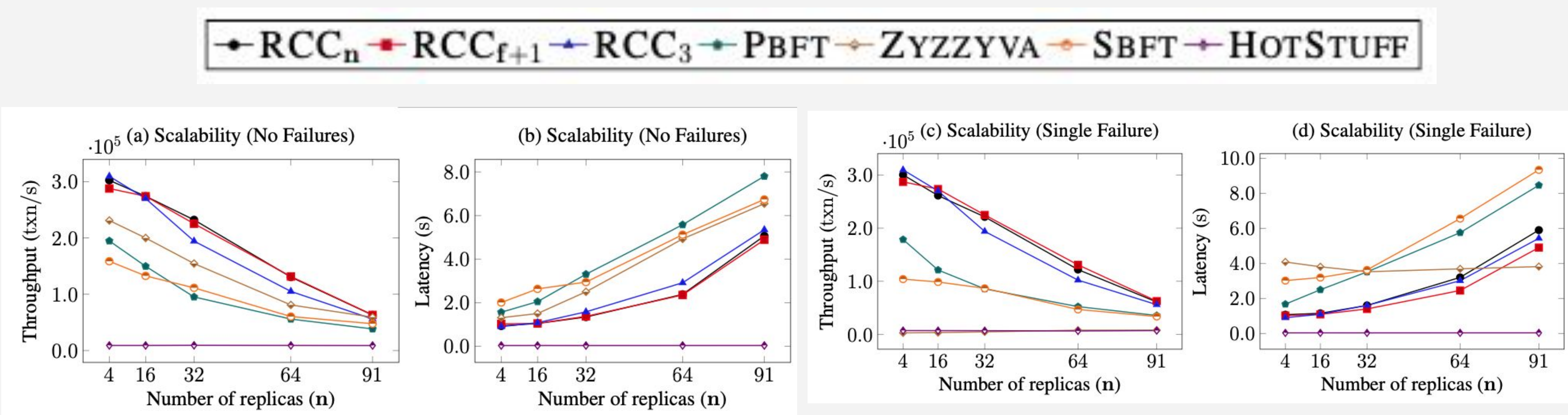
- Performance increases when batch size increases



Evaluation

Varying Replicas

- RCC easily outperforms ZYZZYVA, even in the best-case scenario of no failures
- Three versions of RCC outperform all other protocols
- Adding concurrency by adding more instances improves performance



Conclusion

- RCC allows every replicas to be primary (parallel consensus)
- RCC provides enhanced security features
- RCC guarantees a set of ordered client requests
- RCC provides more throughput than primary-backup consensus

Q & A

Thank you