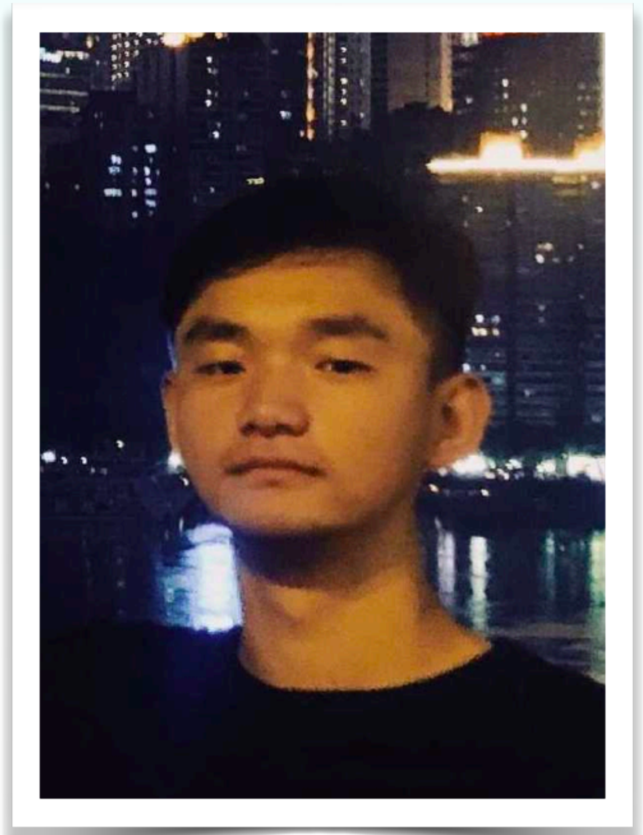
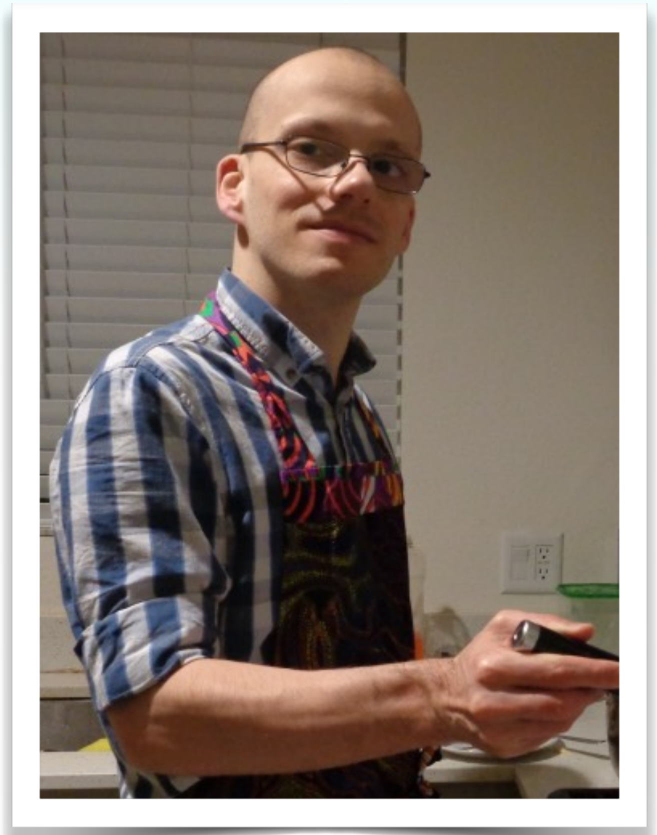
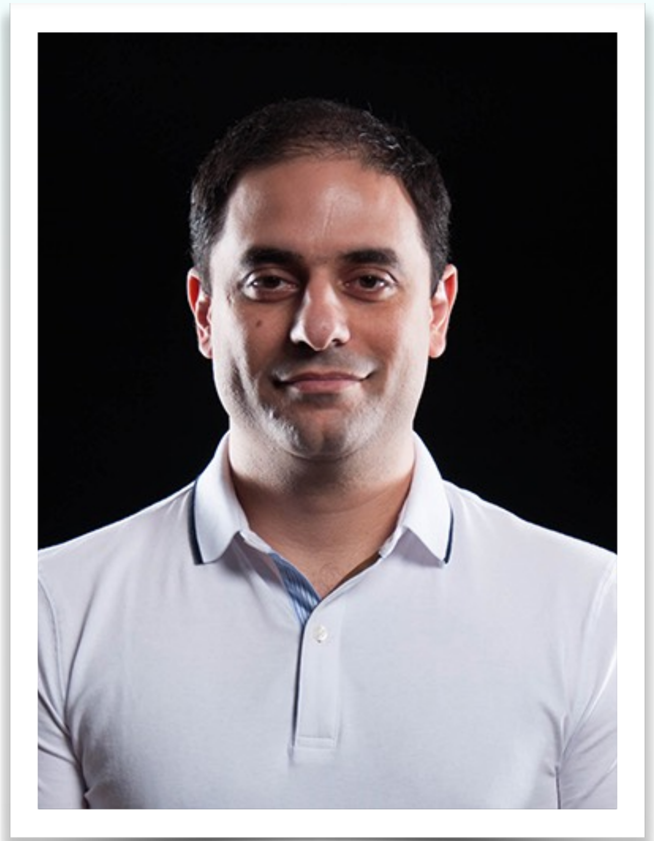
SpotLess: Concurrent Rotational Consensus Made Practical through Rapid View Synchronization

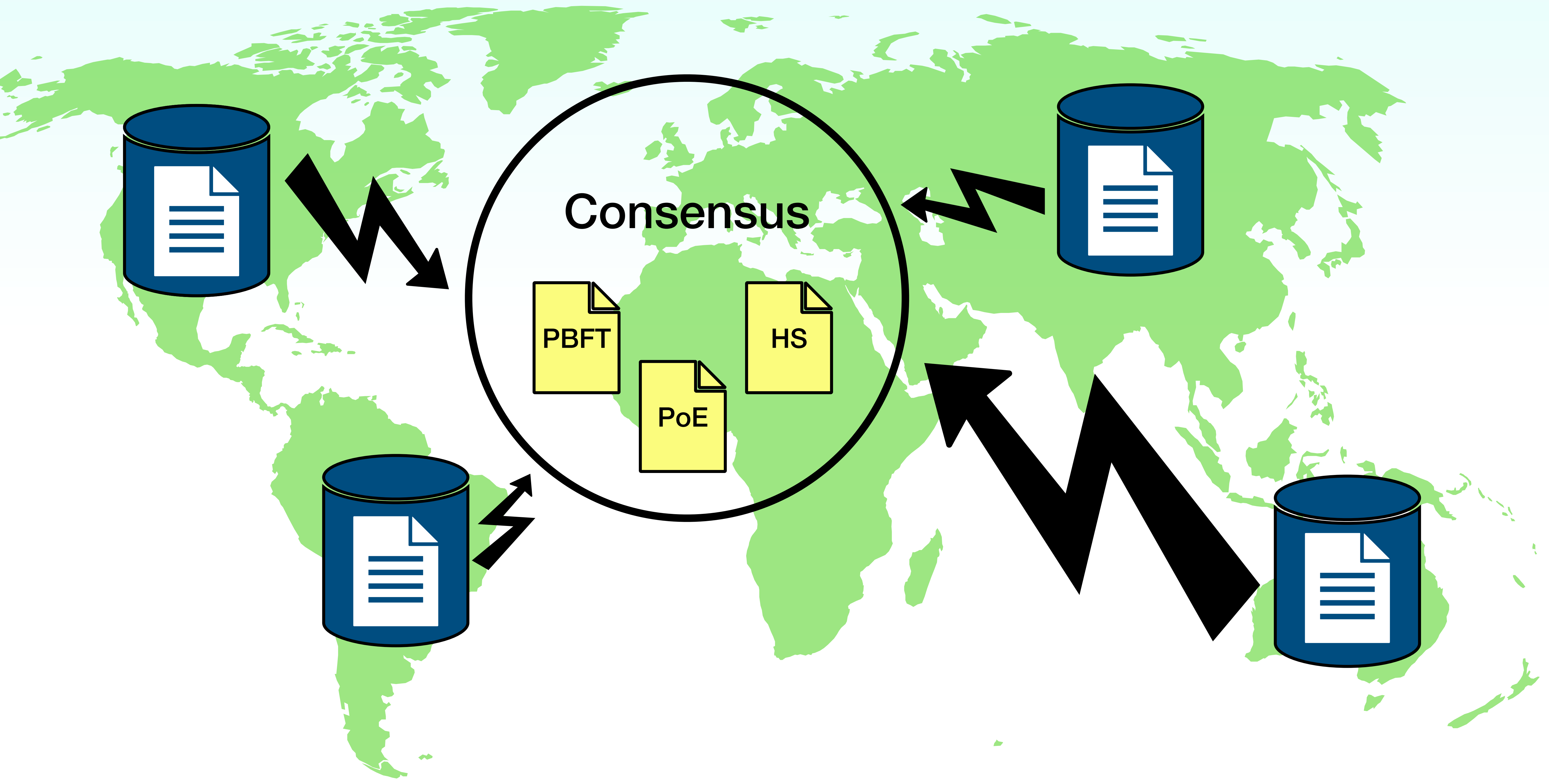


Dakai Kang Sajjad Rahnama Jelle Hellings* Mohammad Sadoghi

Exploratory Systems Lab
Department of Computer Science, University of California, Davis
*Department of Computing and Software, McMaster University

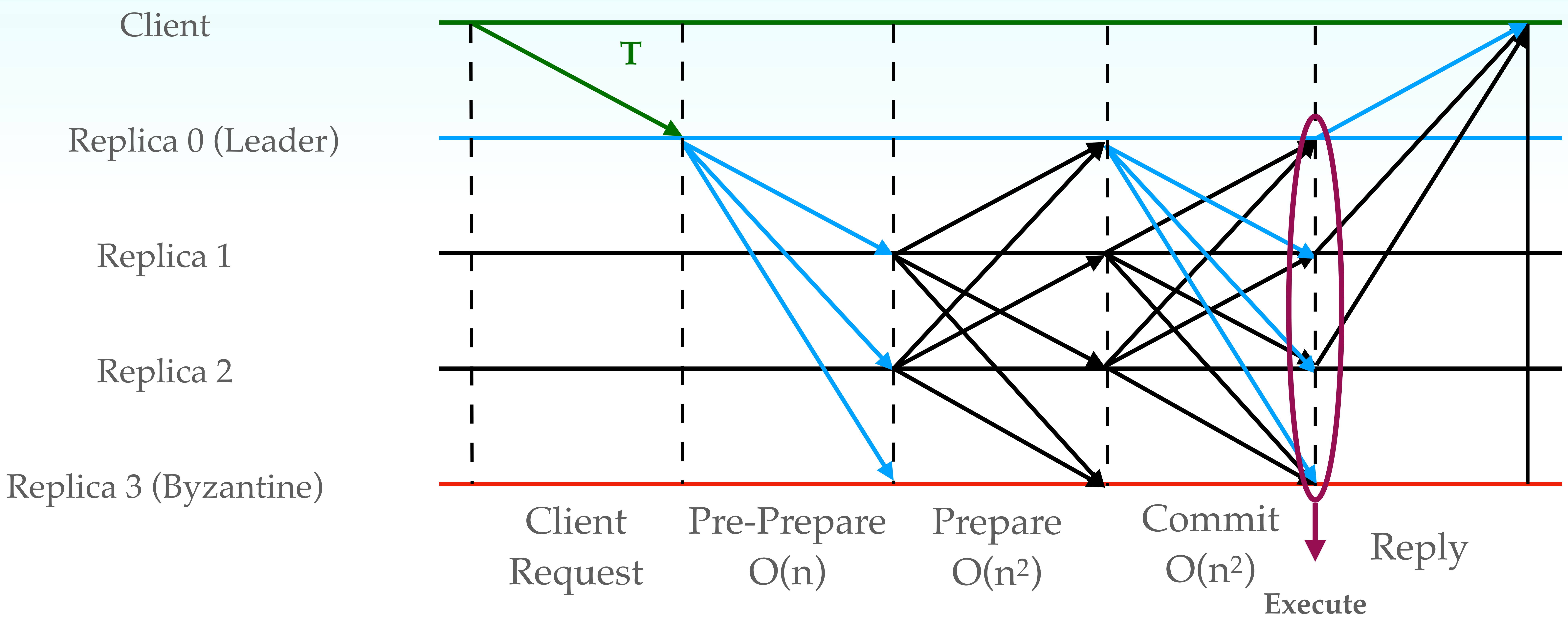
What is Permissioned Blockchain?



- Distributed database consists of a set of replicas (participants).
- Each replica holds a copy of the ledger, which is a chain of blocks containing transactions.
- Consensus Guarantees: Safety; Liveness.
- Fault Model: Byzantine Replicas may behave arbitrarily.
- Consensus Protocols: PBFT, PoE, HotStuff, etc

The core of Blockchain Applications

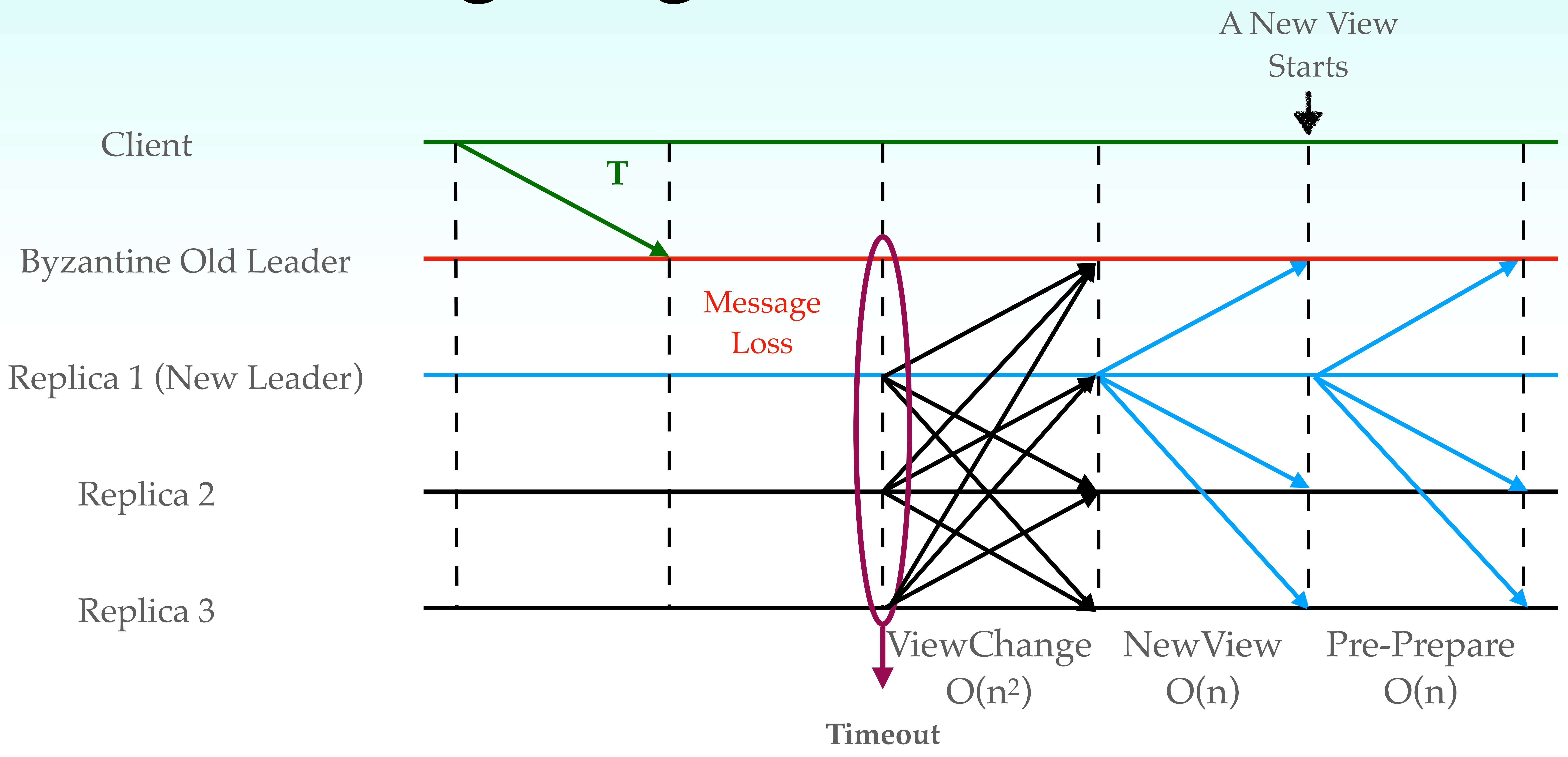
Practical Byzantine Fault Tolerance (PBFT)



Classic BFT Implementation, n = 3f+1

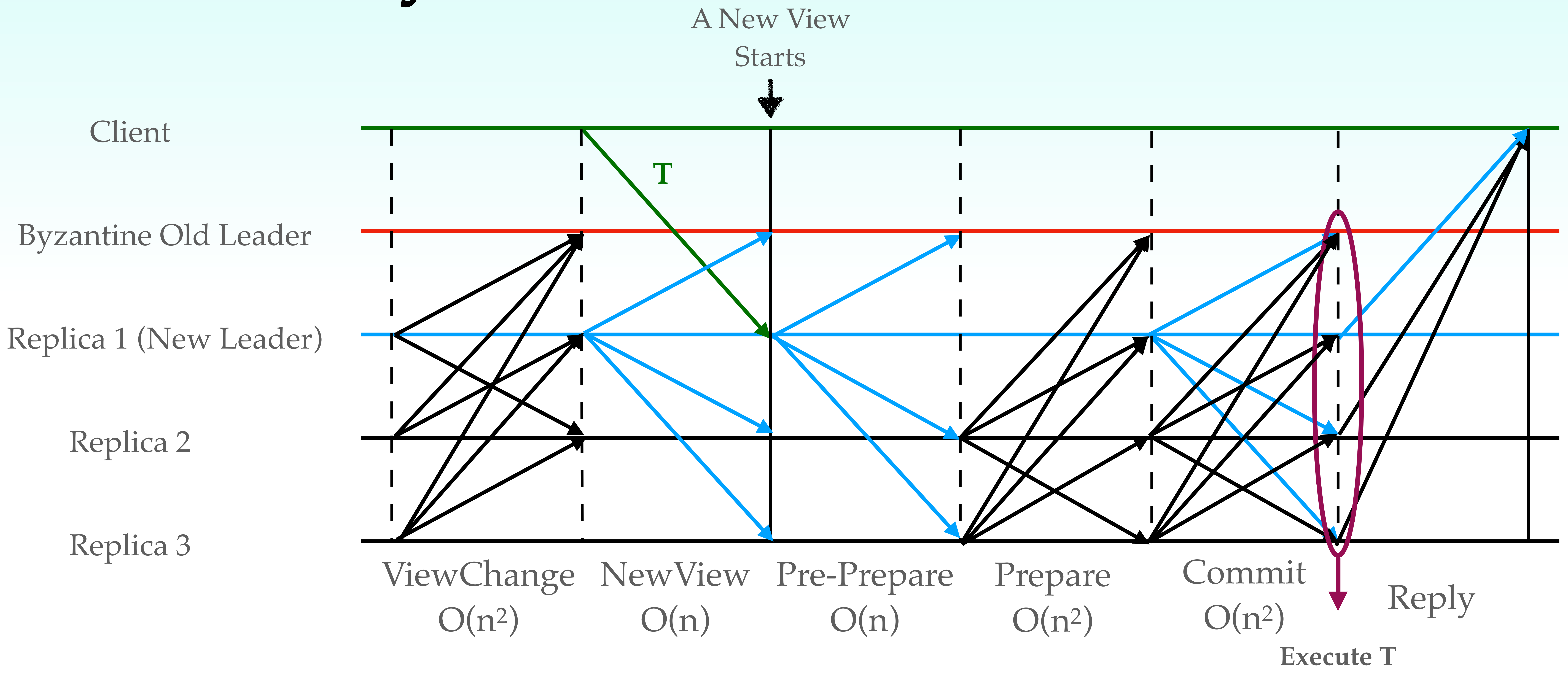
n = 4, f = 1

ViewChange Algorithm in PBFT

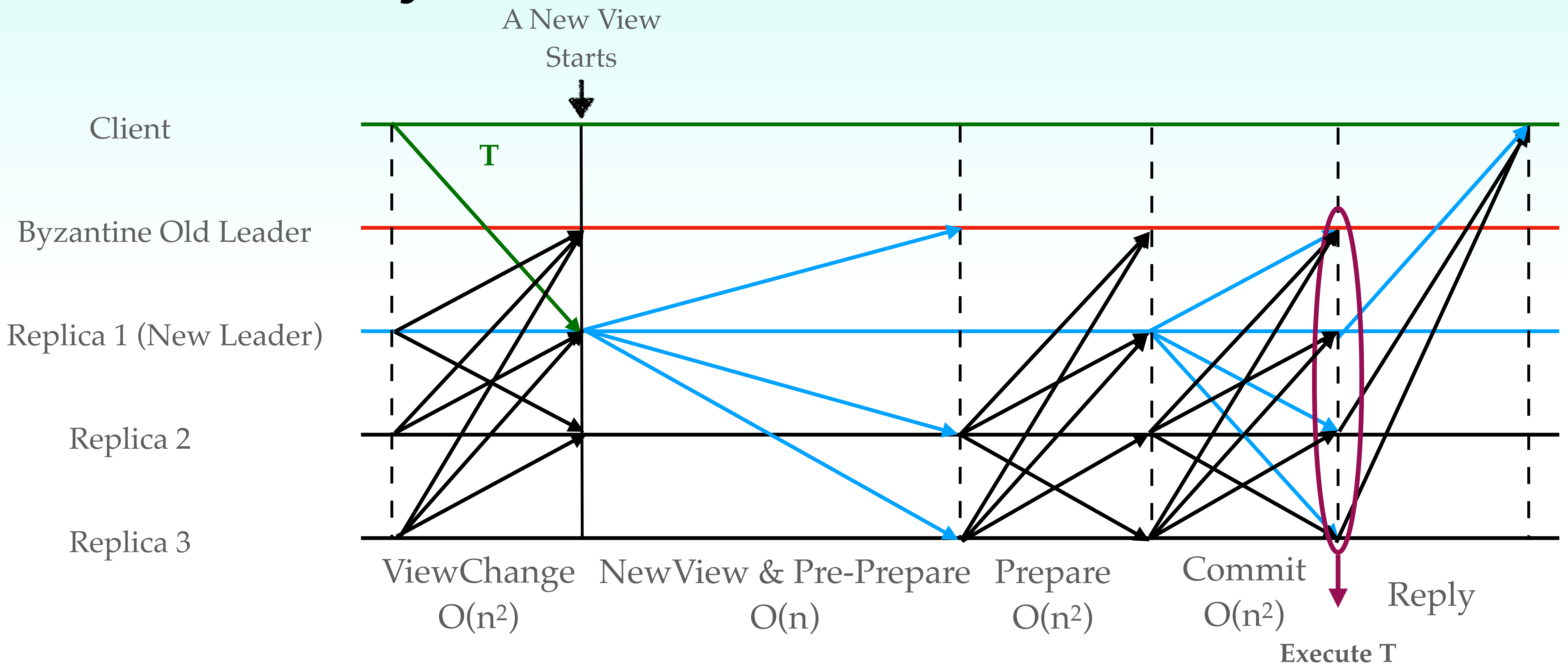


Replacing Leader when things get wrong!

Practical Byzantine Fault Tolerance (PBFT)

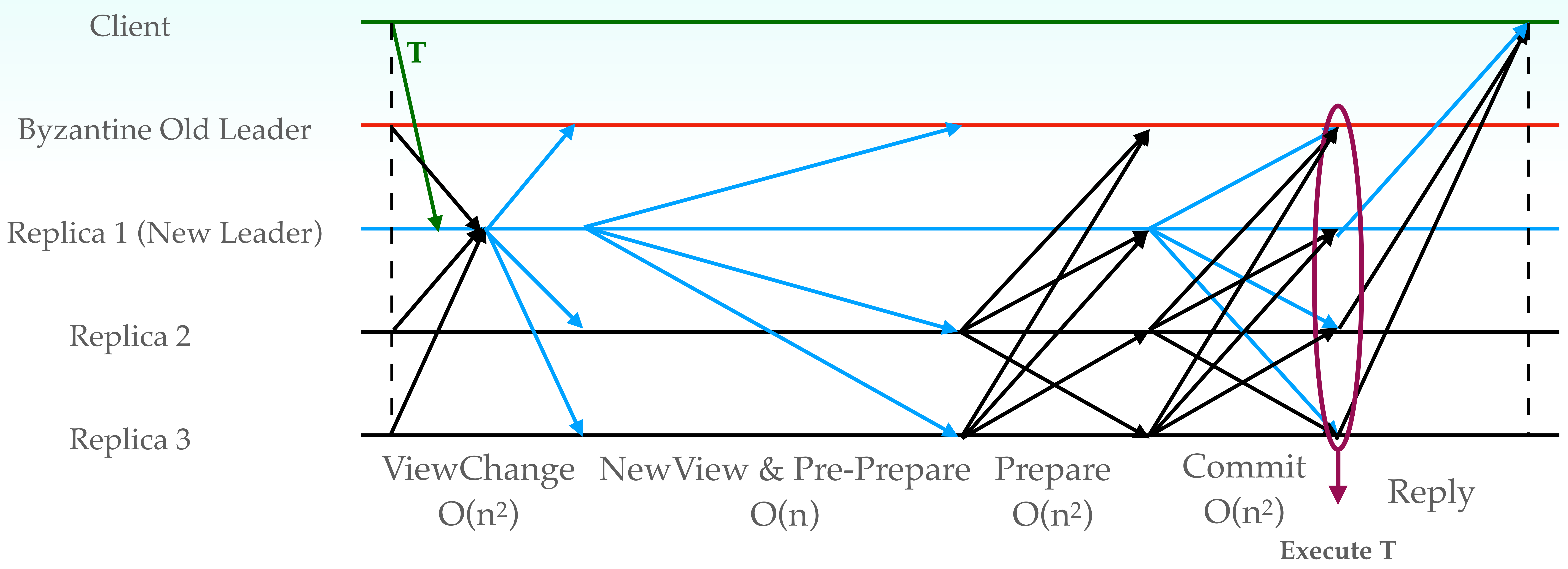


Practical Byzantine Fault Tolerance (PBFT)



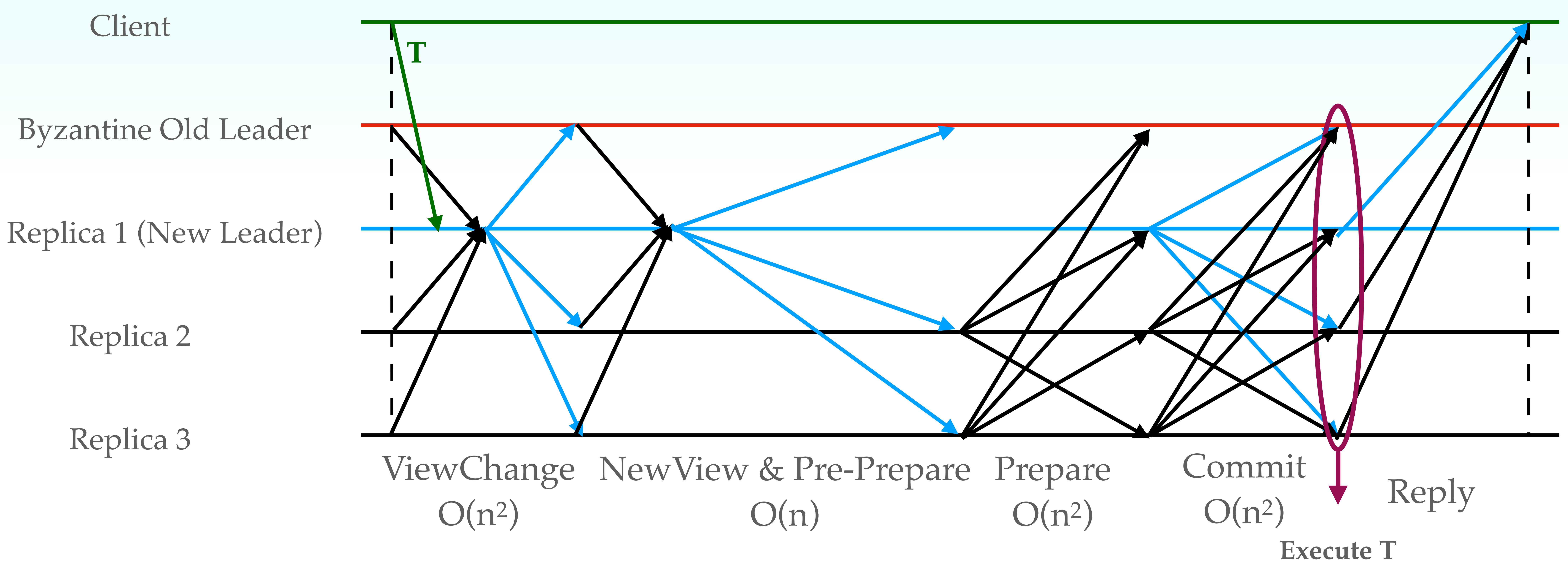
Simplifying ViewChange ==> Rotational Consensus, e.g. HotStuff with Linear Communication Complexity

Practical Byzantine Fault Tolerance (PBFT)



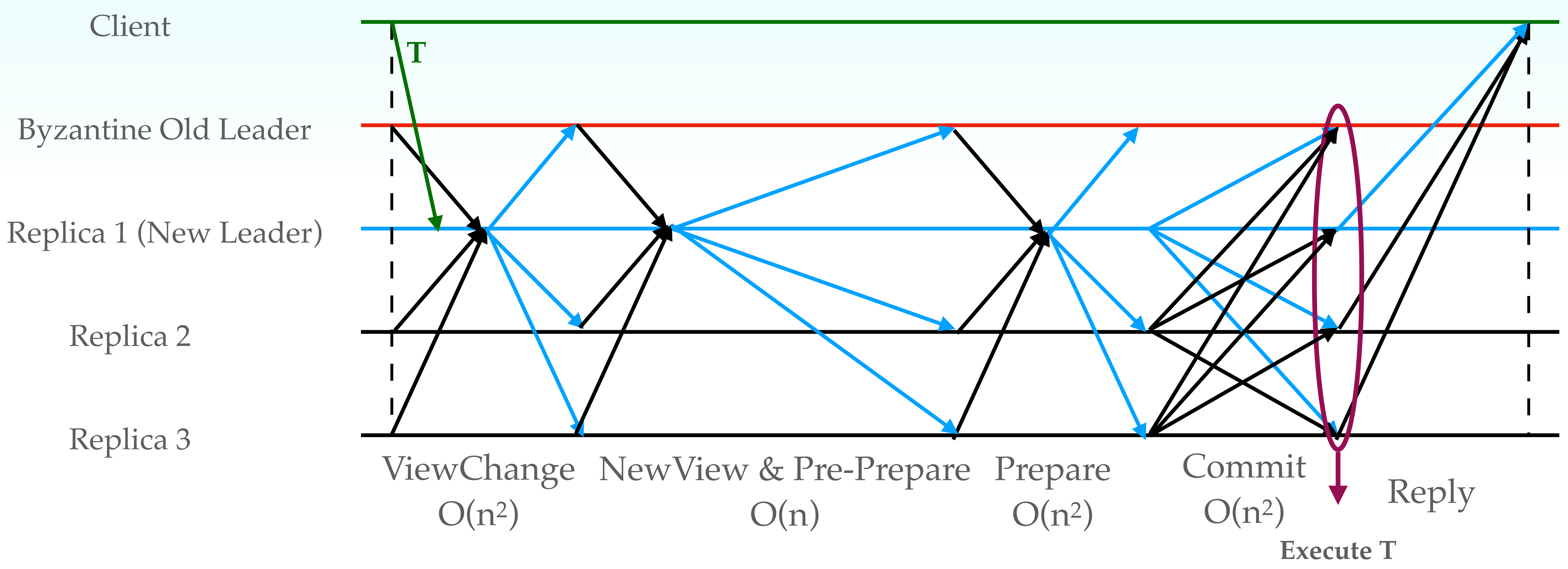
Simplifying ViewChange ==> Rotational Consensus, e.g. HotStuff with Linear Communication Complexity

Practical Byzantine Fault Tolerance (PBFT)



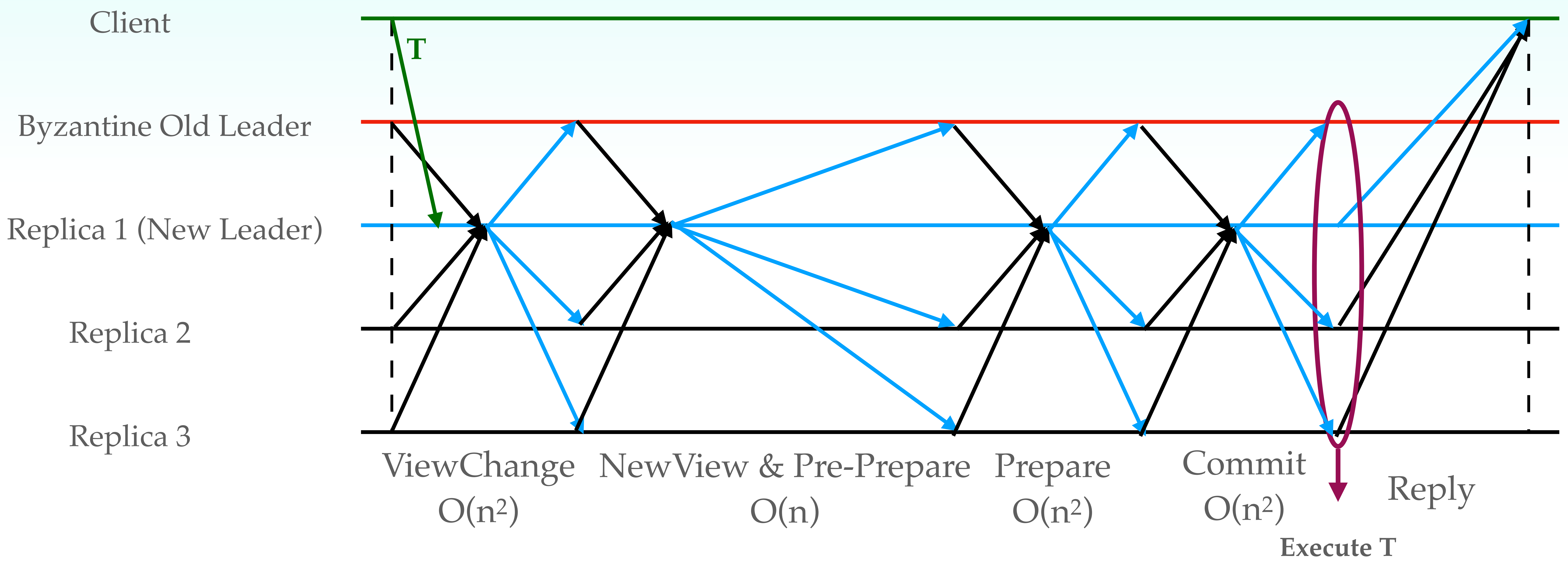
Simplifying ViewChange ==> Rotational Consensus, e.g. HotStuff with Linear Communication Complexity

Practical Byzantine Fault Tolerance (PBFT)



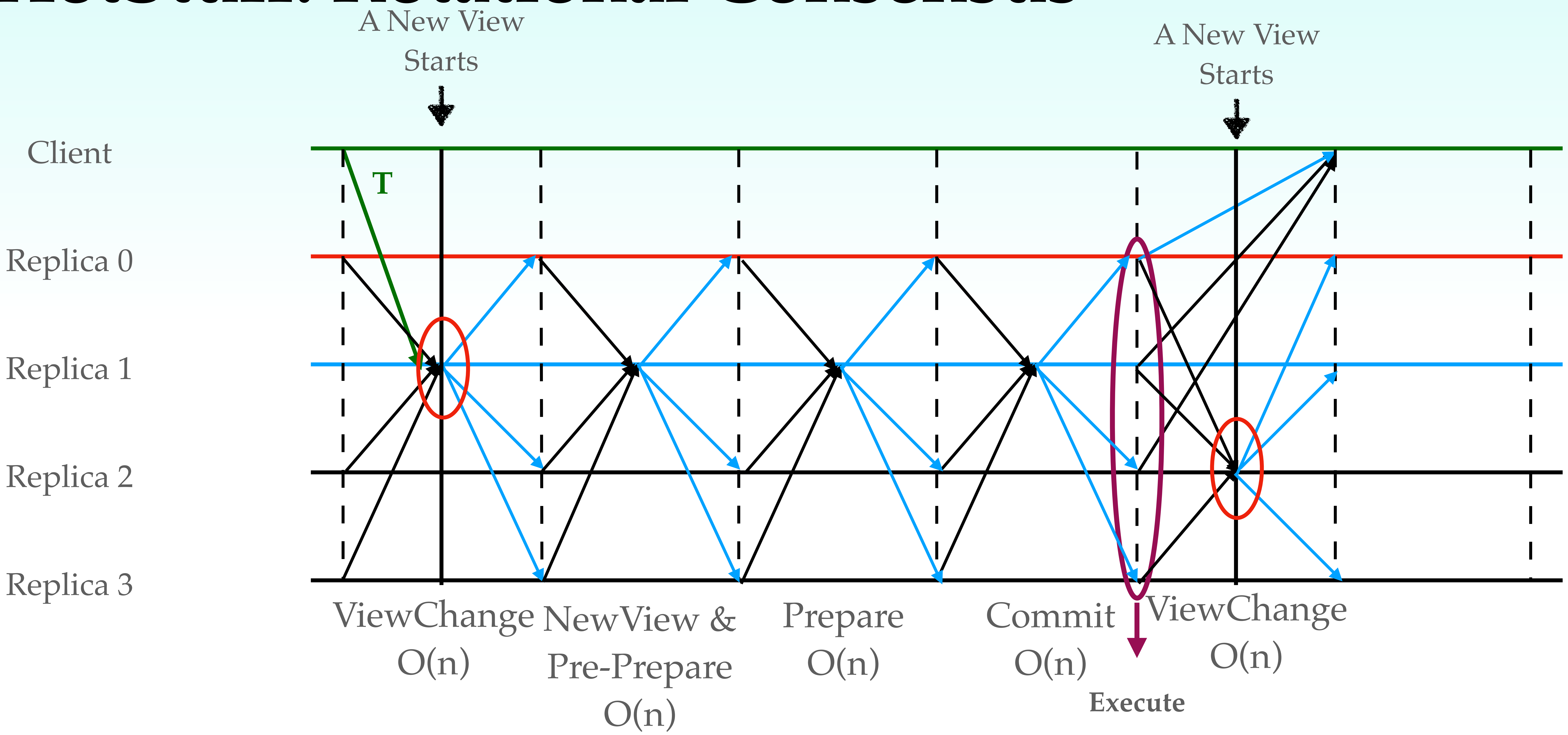
Simplifying ViewChange ==> Rotational Consensus, e.g. HotStuff with Linear Communication Complexity

Practical Byzantine Fault Tolerance (PBFT)



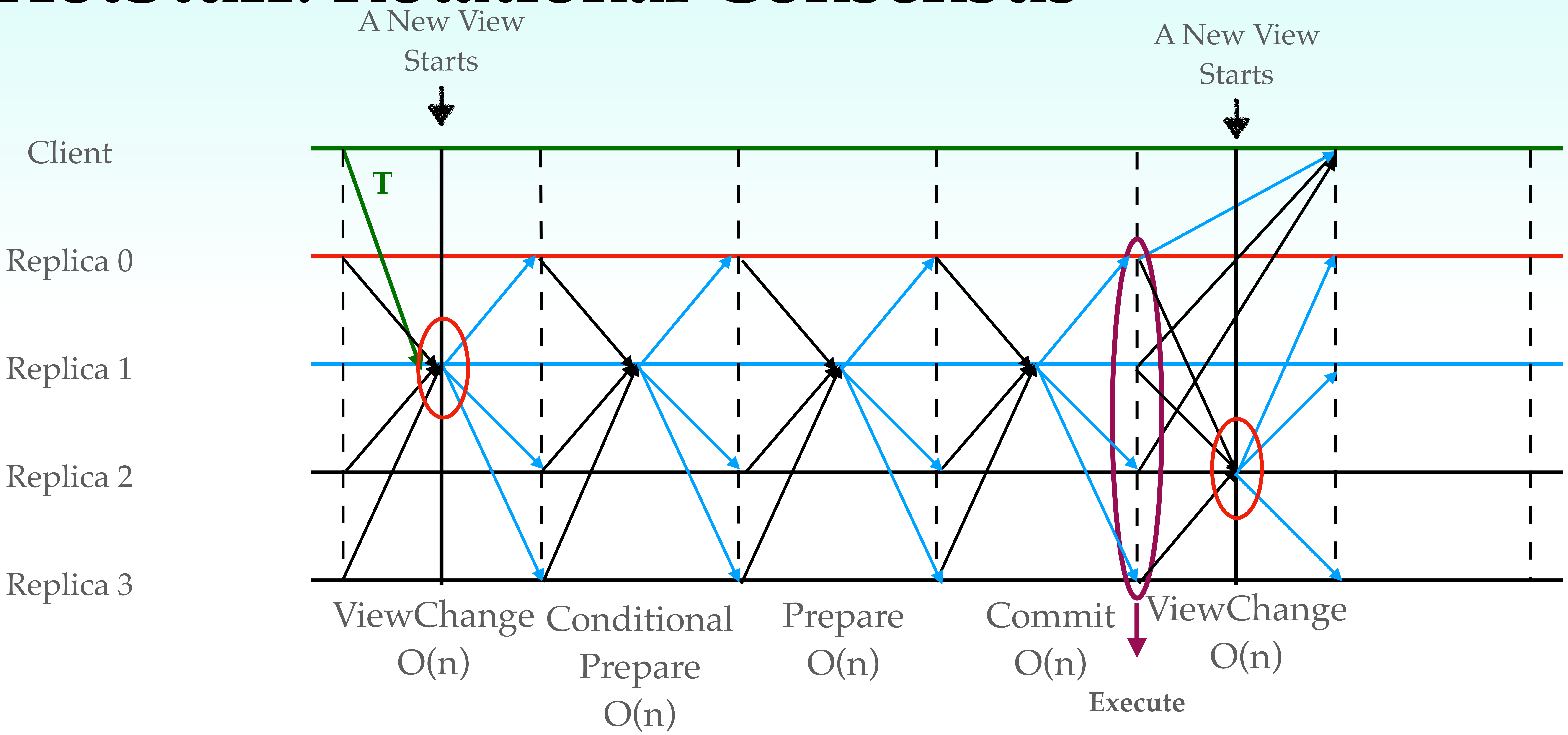
Simplifying ViewChange ==> Rotational Consensus, e.g. HotStuff with Linear Communication Complexity

HotStuff: Rotational Consensus



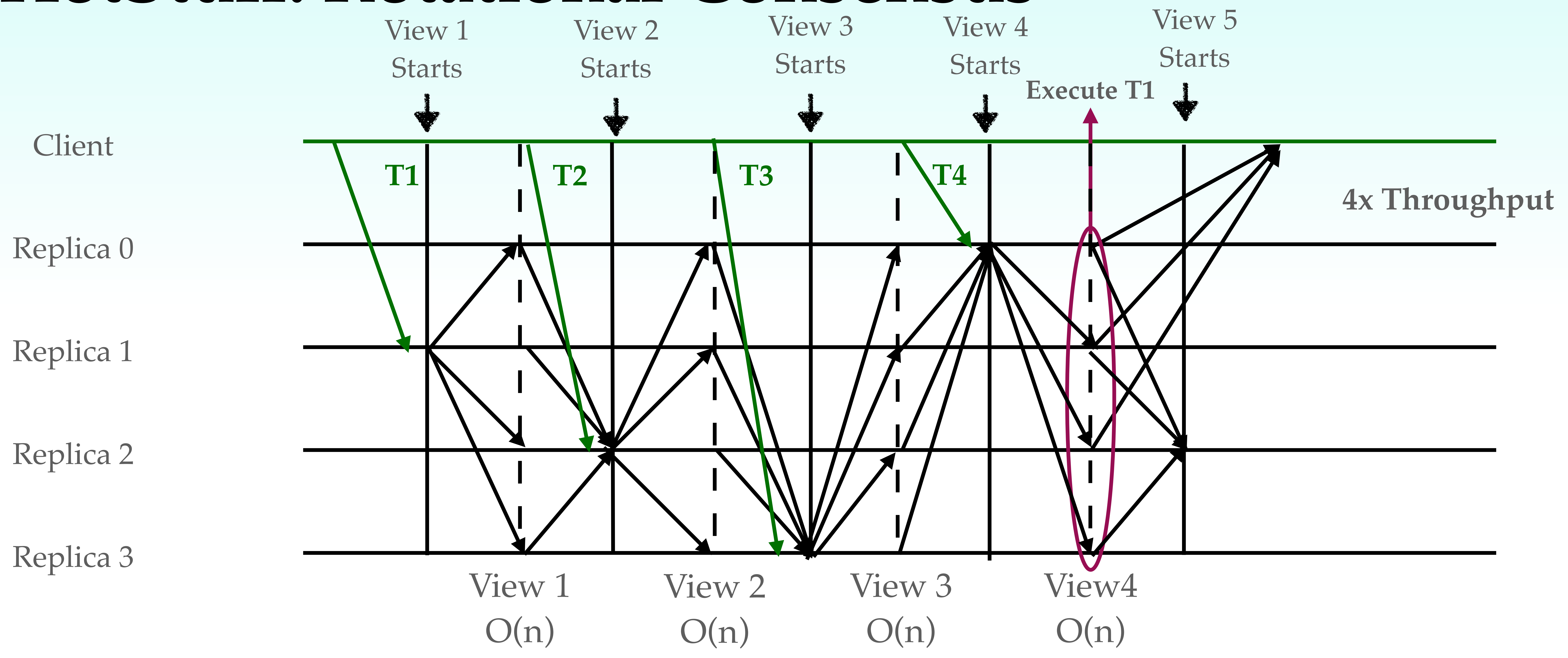
Basic HotStuff

HotStuff: Rotational Consensus



Basic HotStuff

HotStuff: Rotational Consensus



Chained HotStuff: Overlapping phases of consecutive views

Weaknesses of Rotational Consensus

View Synchronization:

- To guarantee Liveness, at least 2f+1 well-behaving replicas should be in the same view for sufficiently long (necessary condition)
- HotStuff uses an unspecified black-box PaceMaker
- Impractical for implementation

Low Throughput and Scalability:

- Lack of Out-of-Order Processing
- Single-Leader Bottleneck

Our Solution: SpotLess

Simple ViewChange: Keep the Low-Complexity ViewChange via Rotational Consensus

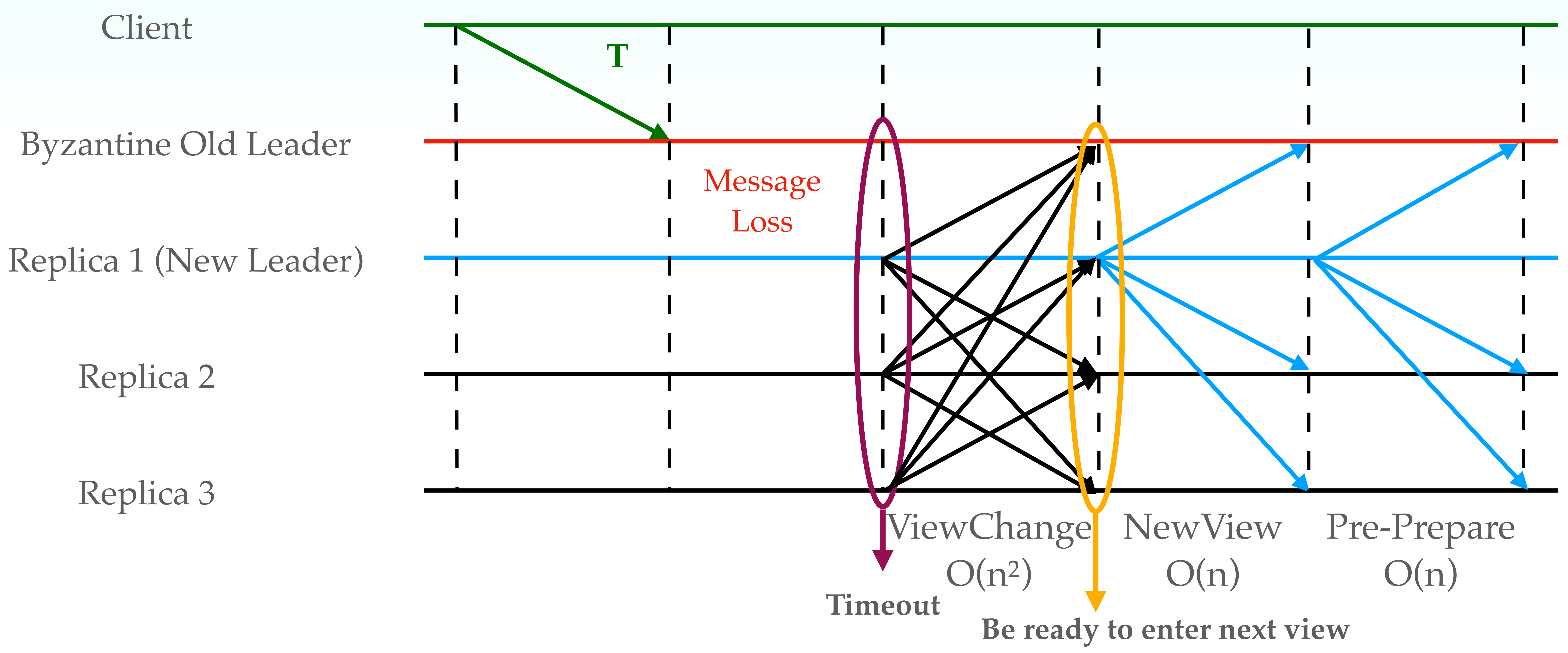
Rapid View Synchronization: An Explicit Mechanism to Synchronize View of Replicas

High Throughput: Break the single-leader bottleneck using Concurrent Consensus

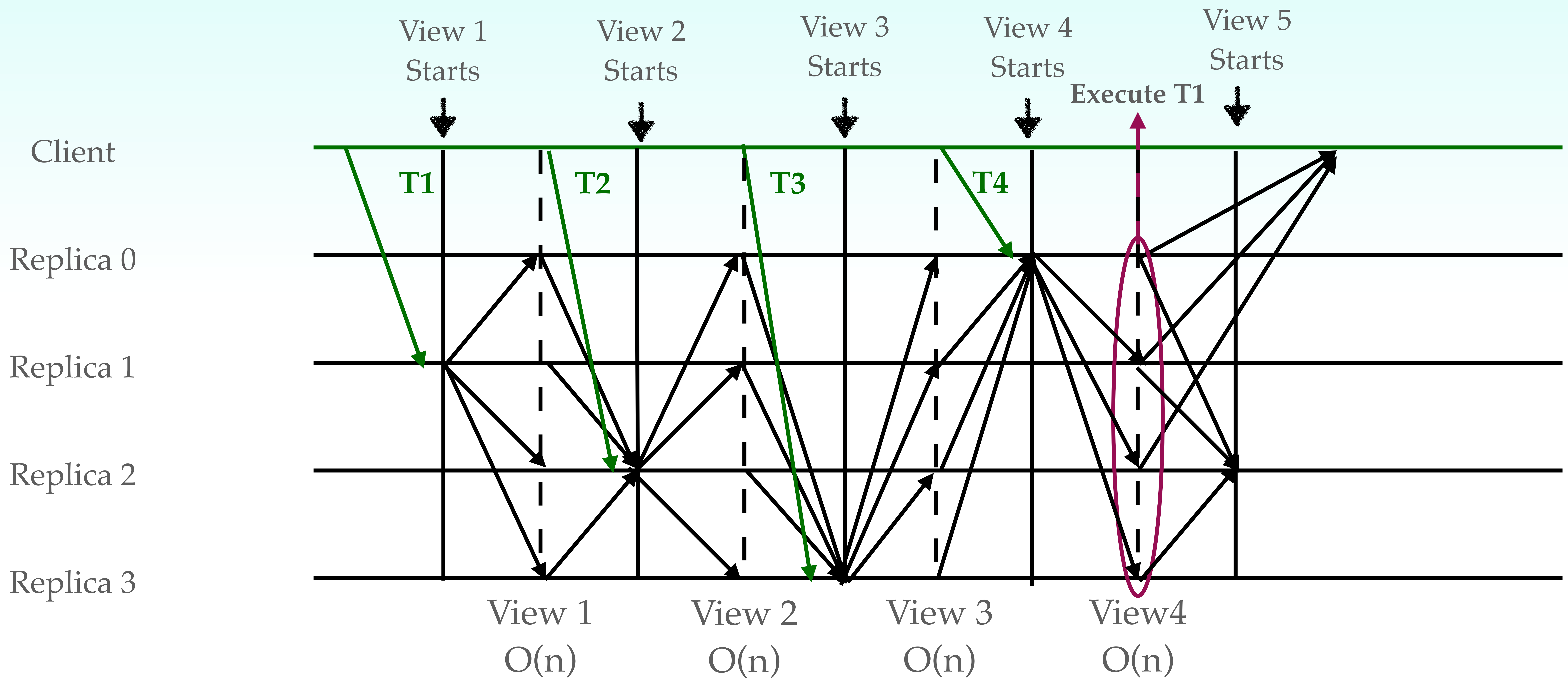
Rapid View Synchronization

Looking backwards at PBFT:

- Advance View only after Seeing 2f+1 ViewChange messages

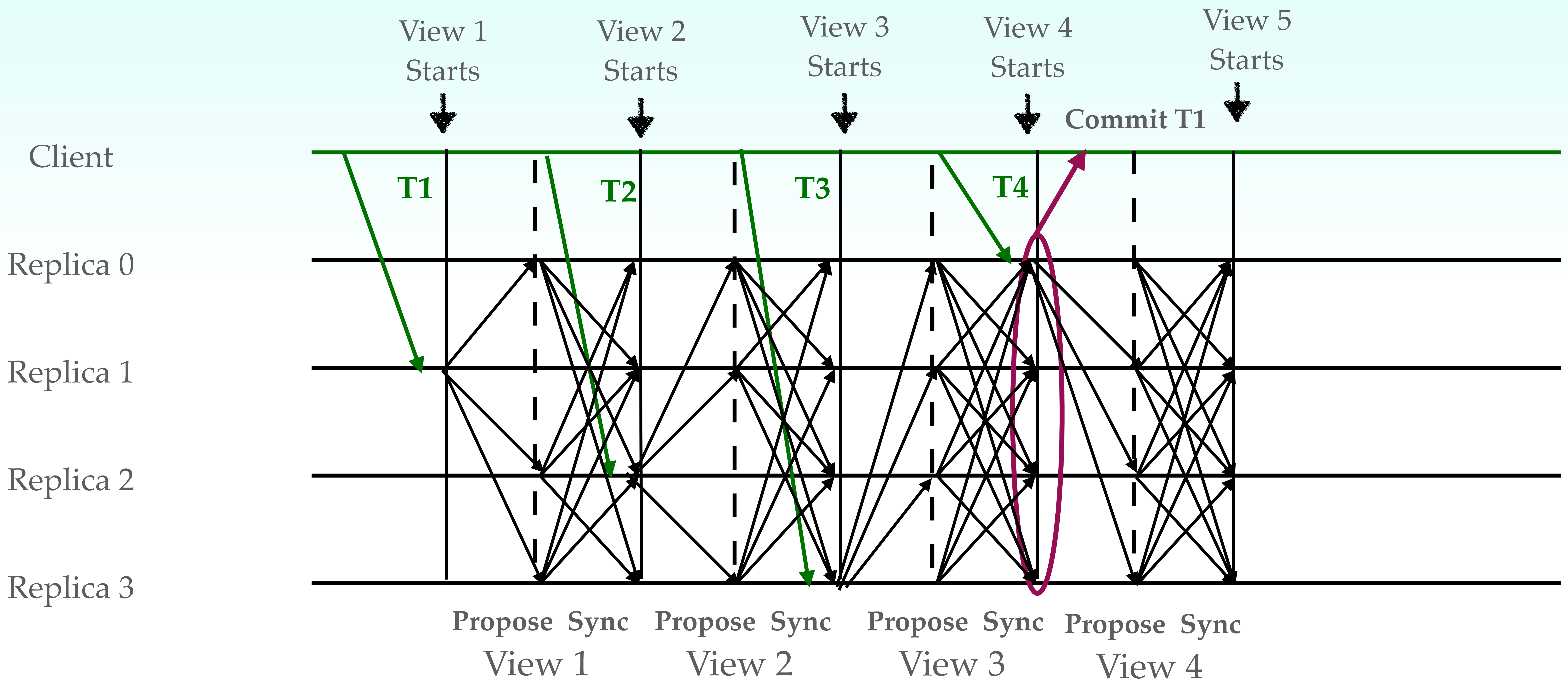


SpotLess: Back to All-to-all Communication



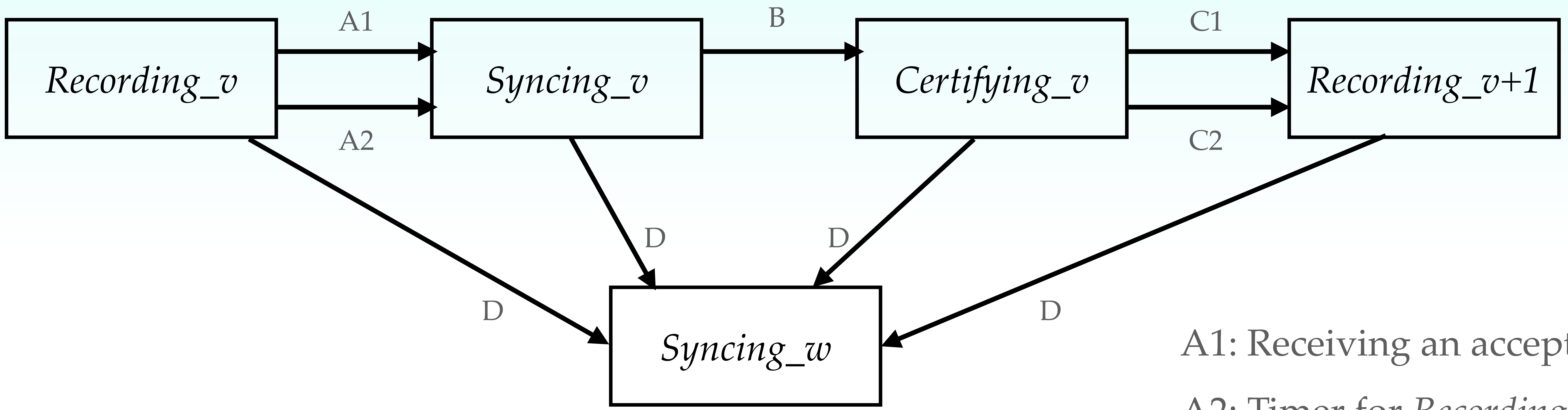
Chained HotStuff: Linear *Propose* Phase + Linear *Vote* Phase

SpotLess: Back to All-to-all Communication



Chained SpotLess: Linear *Propose* Phase + Quadratic *Sync* Phase

View Synchronization States in SpotLess



Recording_v: Start of View

Syncing_v: Pace Synchronization

Certifying: Forming Certificate

A1: Receiving an acceptable proposal

A2: Timer for *Recording* State expires

B : Receiving 2f+1 Sync messages

C1: Forming a certificate

C2: Timer for *Certifying* State expires

D : Receiving f+1 Sync message of view w, w > v

Rapid View Synchronization Guarantees

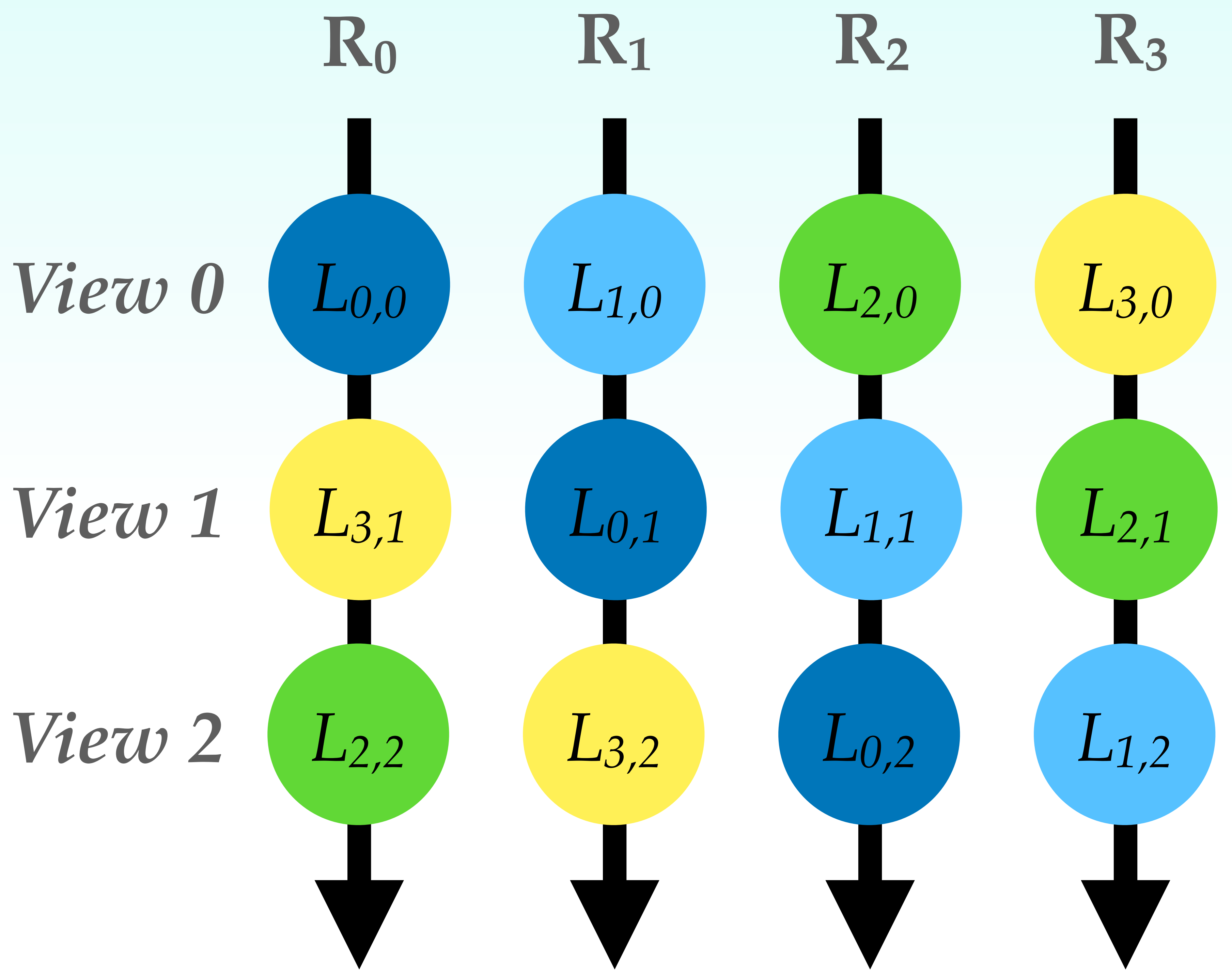
Network becomes Synchronous

Any Future Non-Faulty Leader can form a certificate

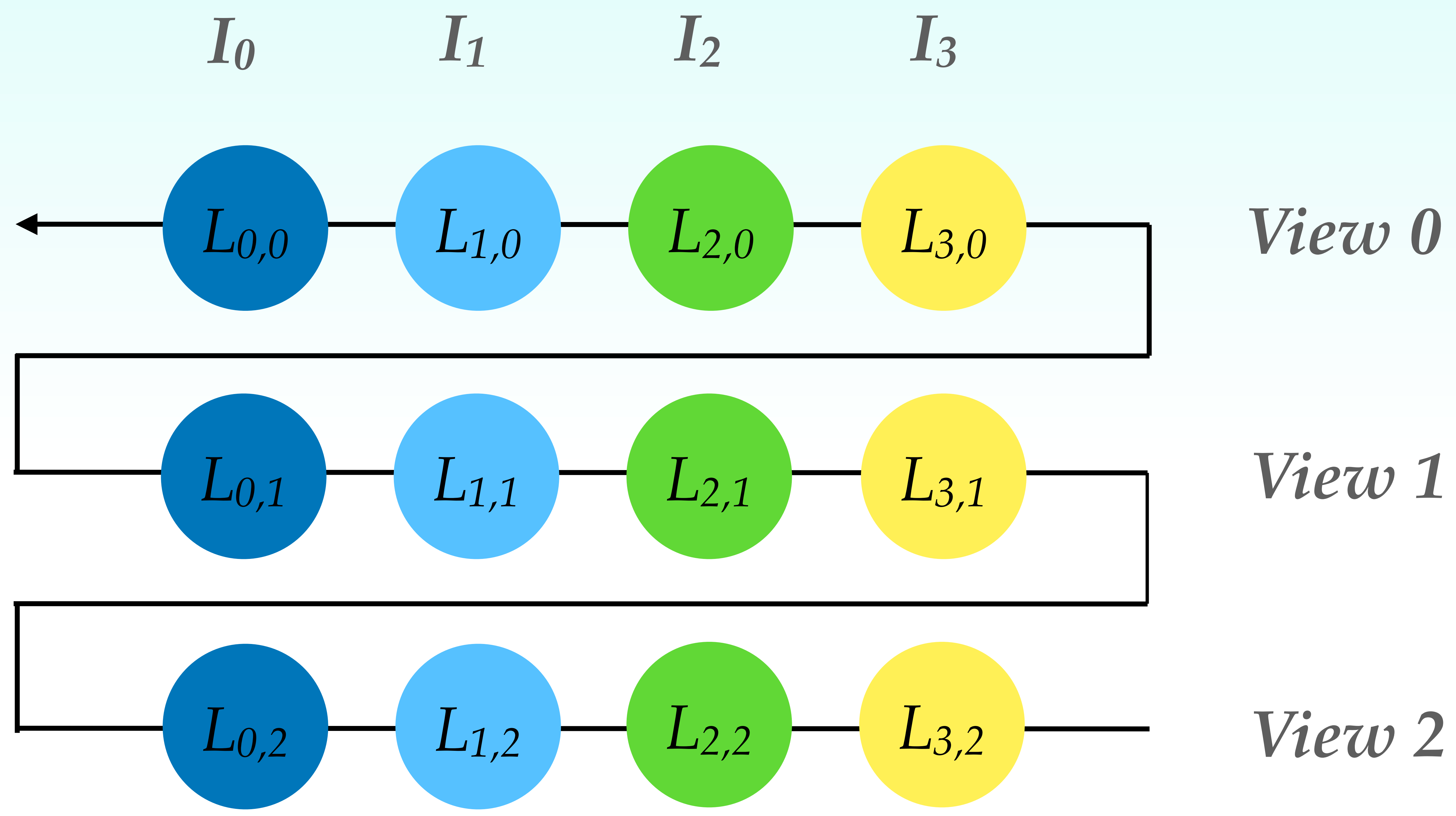
Three consecutive non-faulty leaders commit a transaction

Liveness

Concurrent Consensus in SpotLess



SpotLess with 4 replicas and 4 instances;
$L_{i,v}$ is the Leader of instance i in view v;
$L_{i,v} = (i+v) \bmod n$

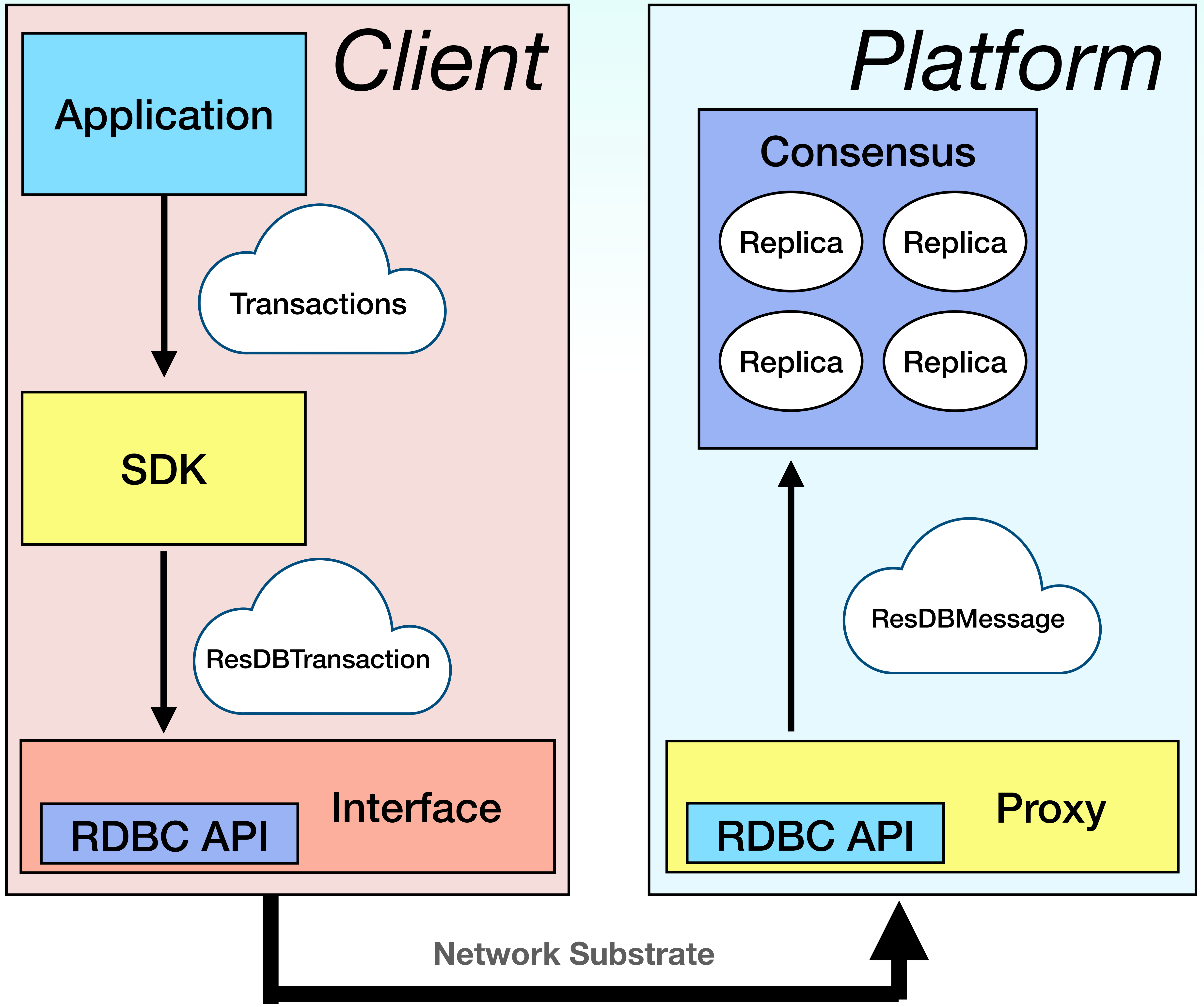


Globally Order transactions in
view-increasing and instance-increasing order.

UCDAVIS

Apache ResilientDB Incubating

ResilientDB Architecture Overview

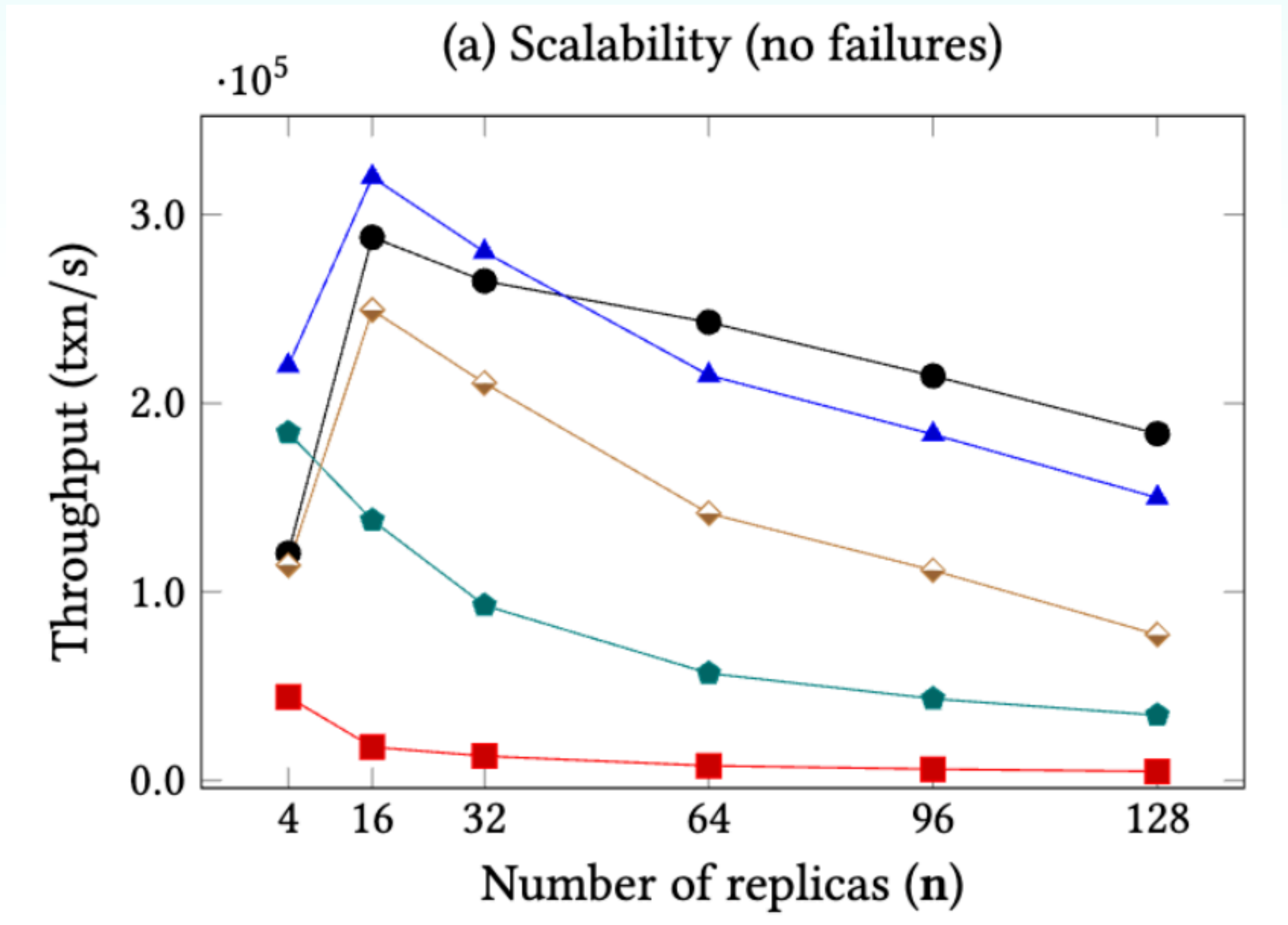


1. **Applications** submit **client transactions** to SDK;
2. **SDK** transforms the client transactions into **ResDBTransaction** objects;
3. Sends the ResDBTransaction to **Proxy** by invoking the **RDBC API**;
4. The **ResDBTransaction** is delivered from the client to the **Proxy** via the **Network Substrate**;
5. The Proxy packs the ResDBTransaction into **ResDBMessage** and forwards it to **Replicas**

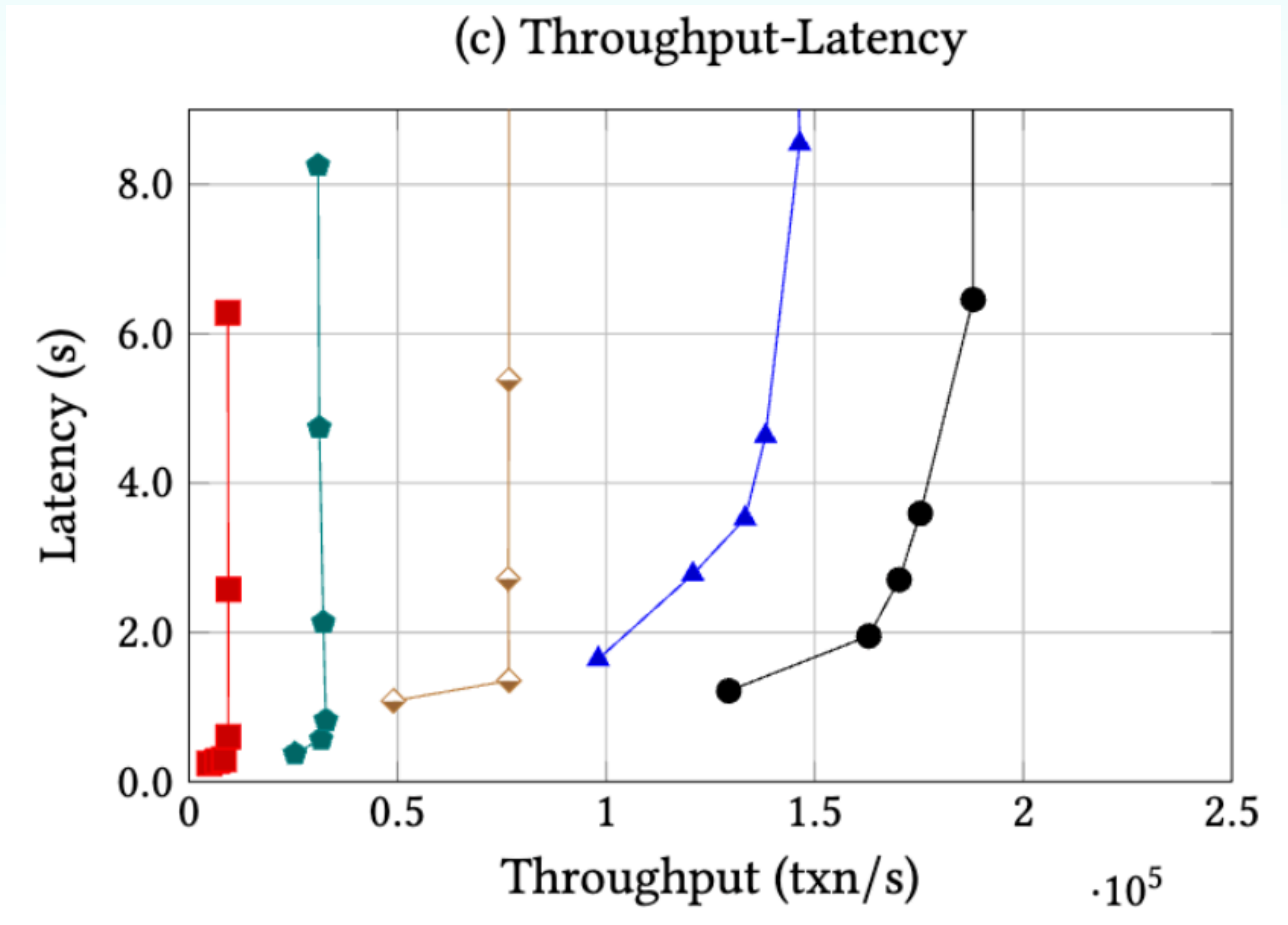
Evaluation

Concurrent PBFT

●— SpotLess ■— HotStuff ▲— RCC ⬟— Pbft ◇— Narwhal-HS



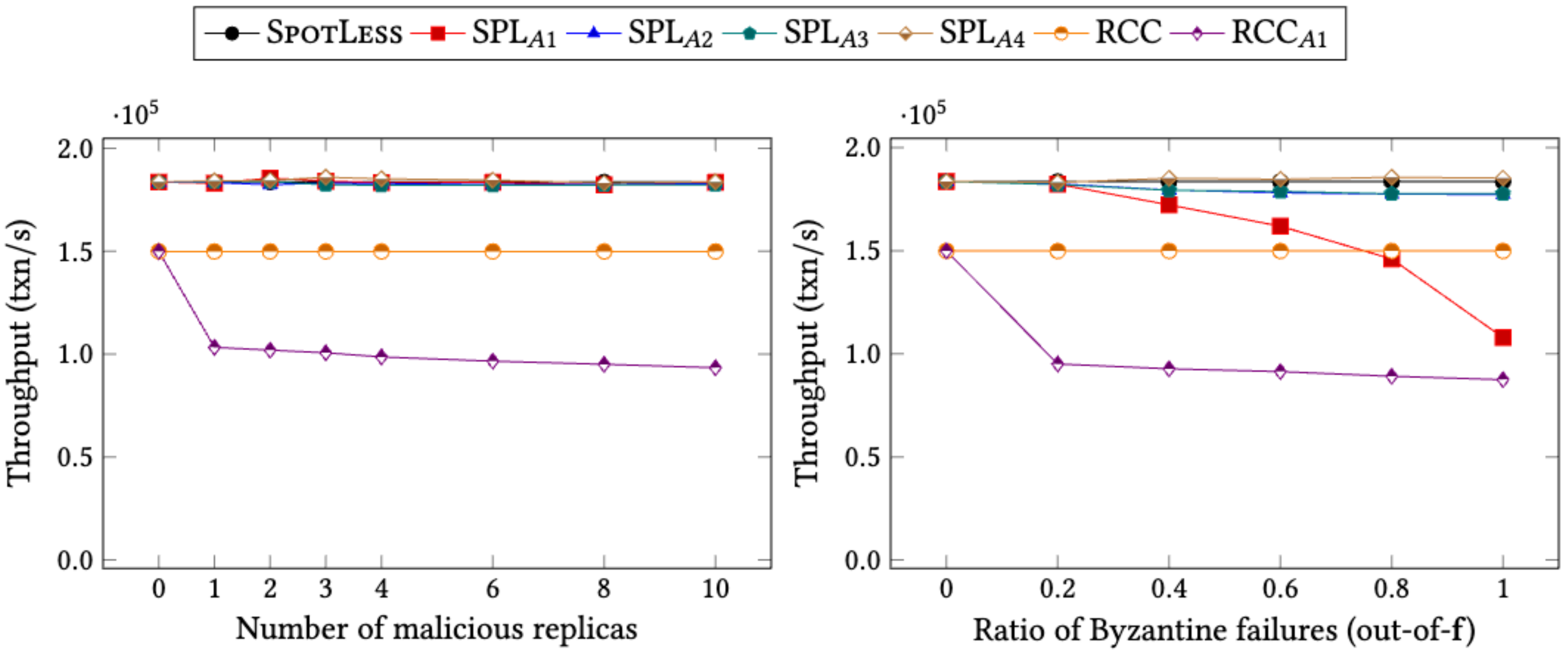
Normal Case Throughput



Throughput-latency (128 replicas)

OutStanding Performance of SpotLess in Normal Cases

Evaluation



A1 Always Non-Responsive

A2 Keeping in Dark

A3 Equivocation

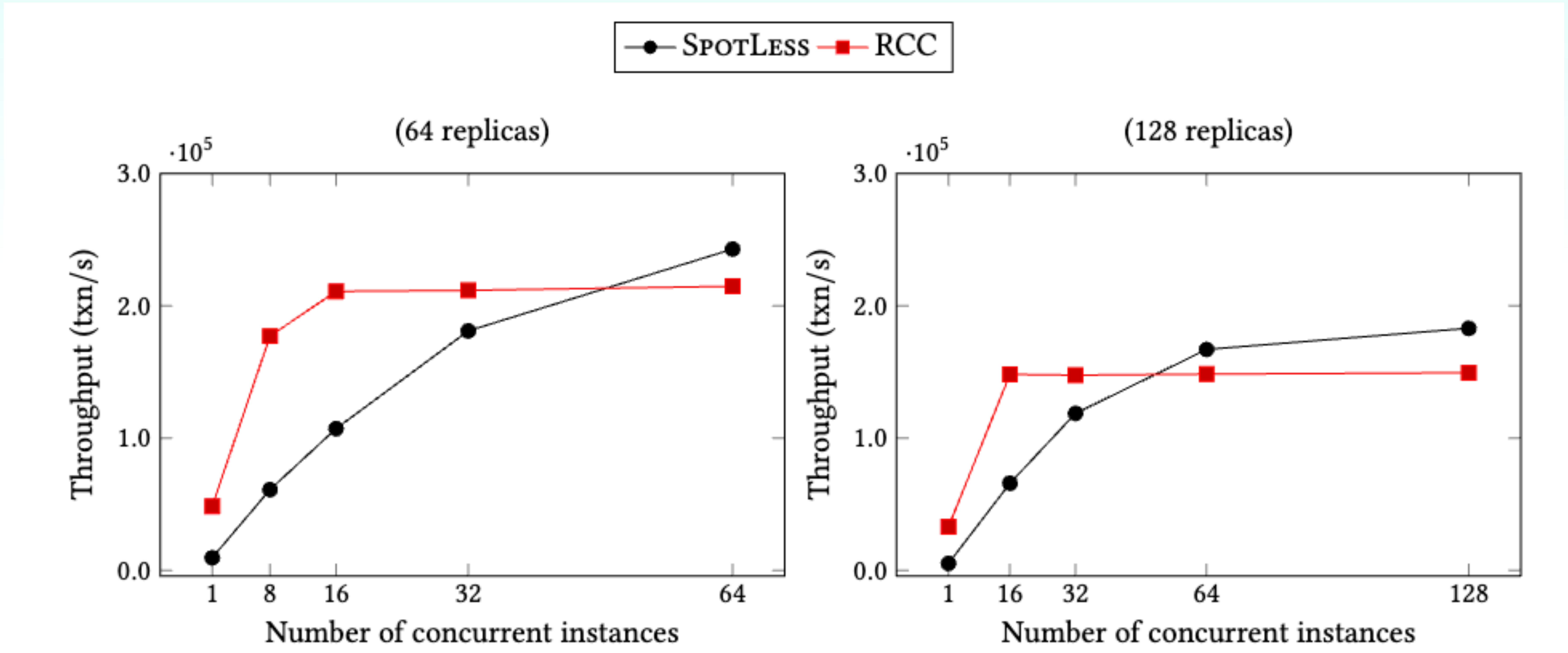
A4 Non-Responsive to non-faulty leaders

Resilient Performance of SpotLess under Attacks

UCDAVIS McMaster University

Apache ResilientDB Incubating

Evaluation



SpotLess benefits more than RCC from Concurrent Consensus

Thank You

https://resilientdb.com/

Publications: