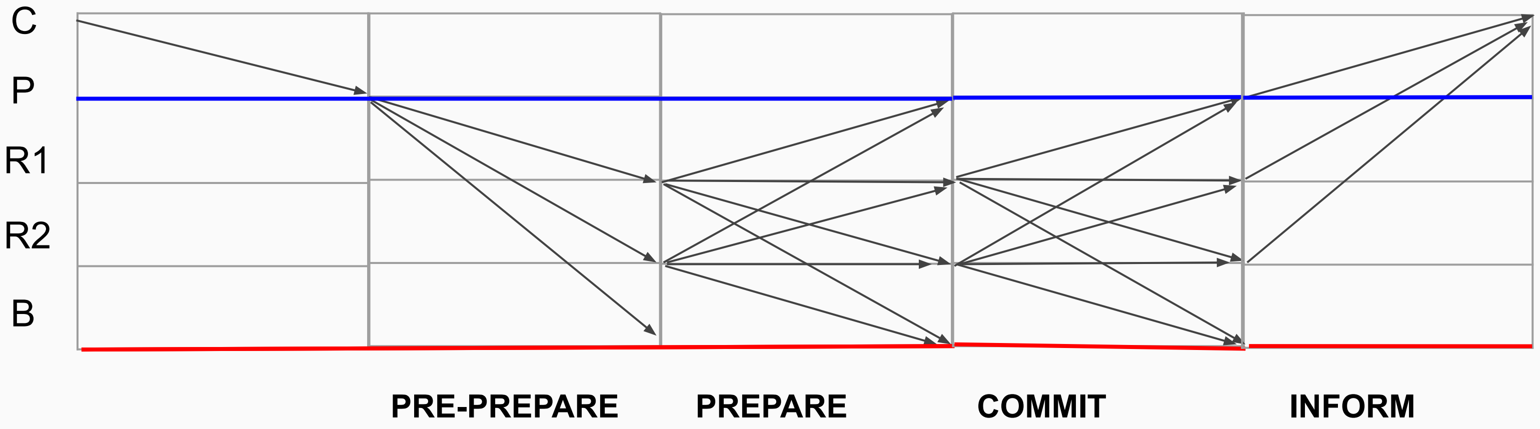
SpotLess: Concurrent Rotational Consensus Made Practical through Rapid View Synchronization

Dakai Kang, Sajjad Rahnama, Jelle Hellings, Mohammad Sadoghi

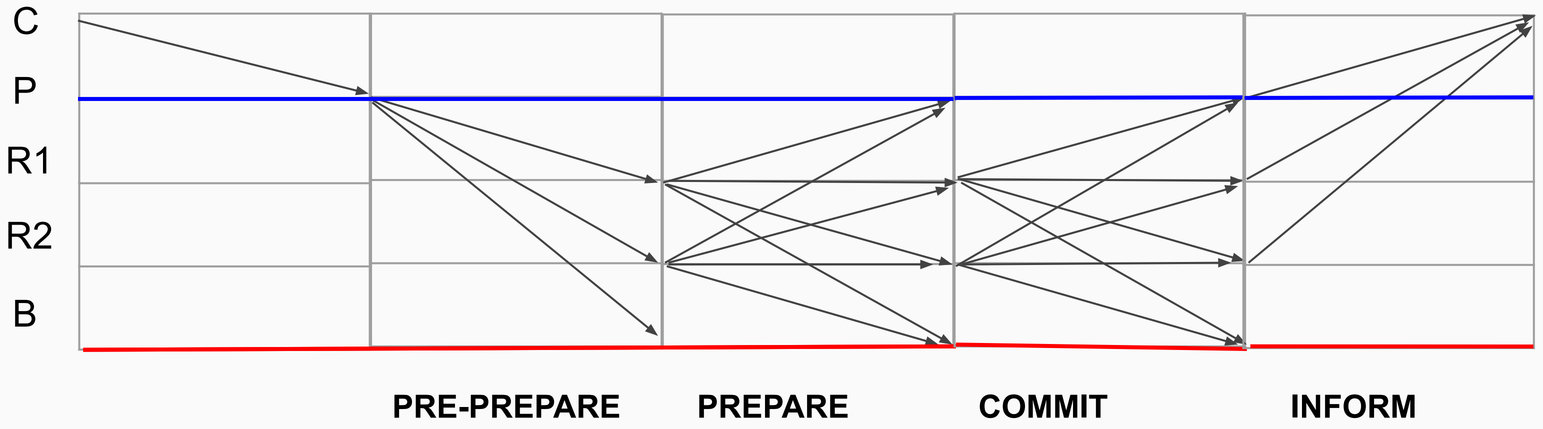
Presenters: Dieu Anh Le, Gopal Nambiar, Rishika Garg, Sasha Pimento

Introduction

Practical Byzantine Fault-Tolerant Consensus



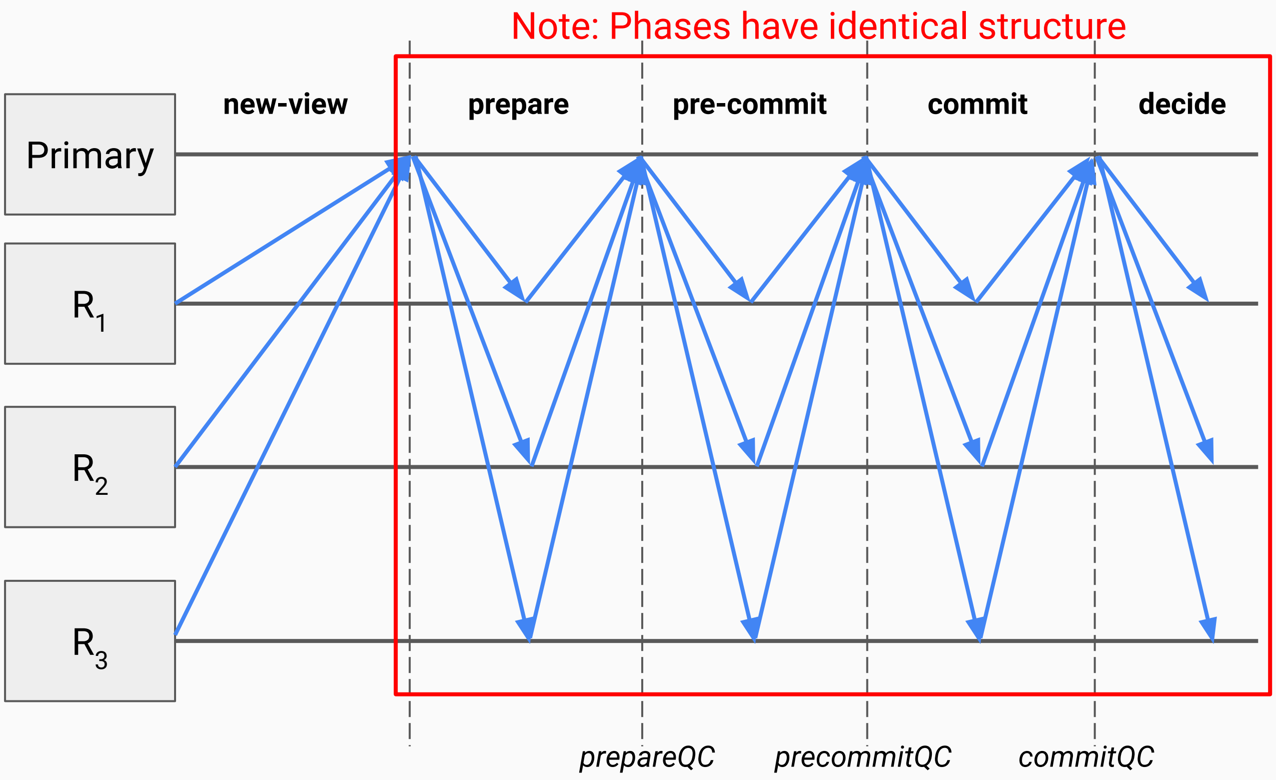
Limitations of PBFT



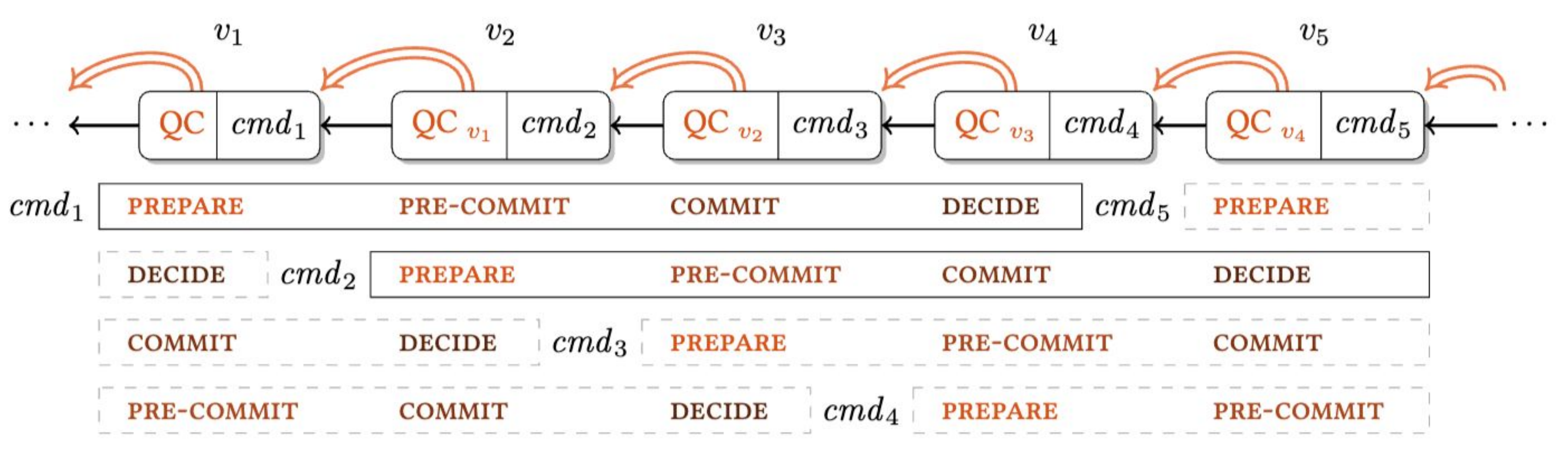
- Performance is bottlenecked by the network bandwidth of primary
- Reduced Scalability
- Techniques used to reach high throughput, have complex implementations

How does HotStuff work?

- New-view
- Prepare phase
- Pre-commit phase
- Commit phase
- Decide phase

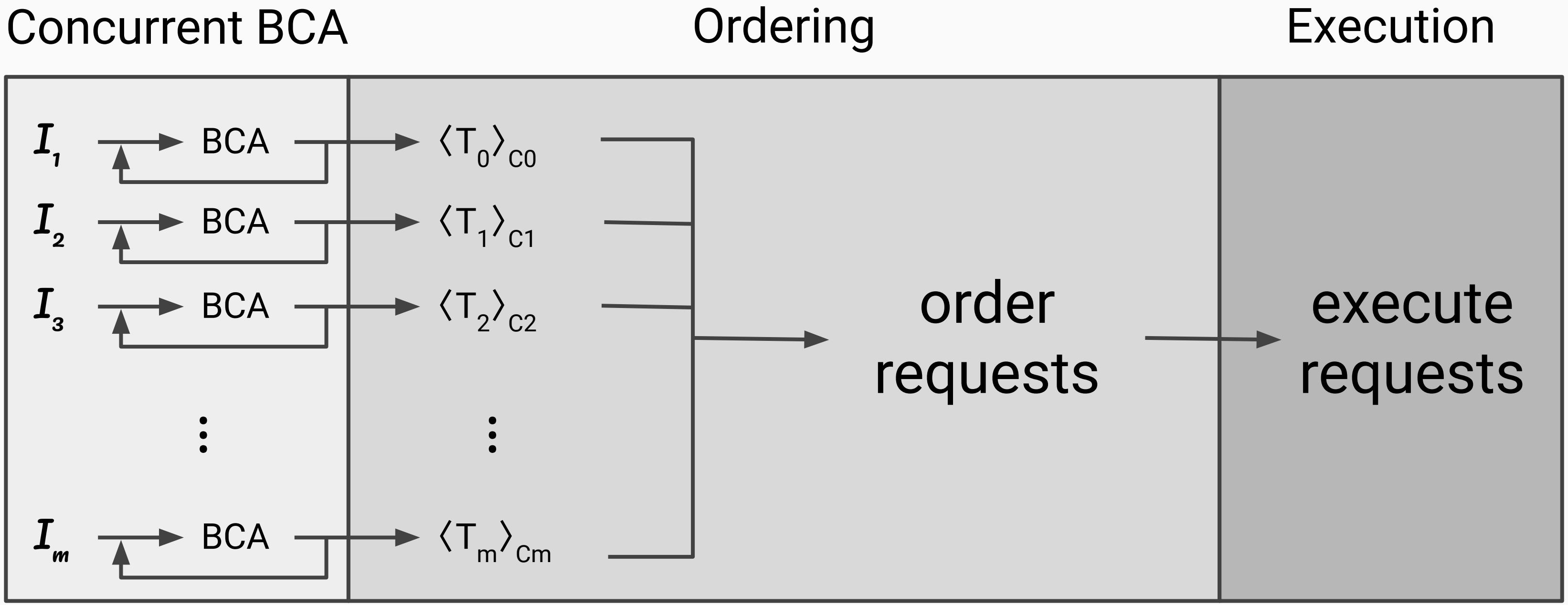


Chained HotStuff



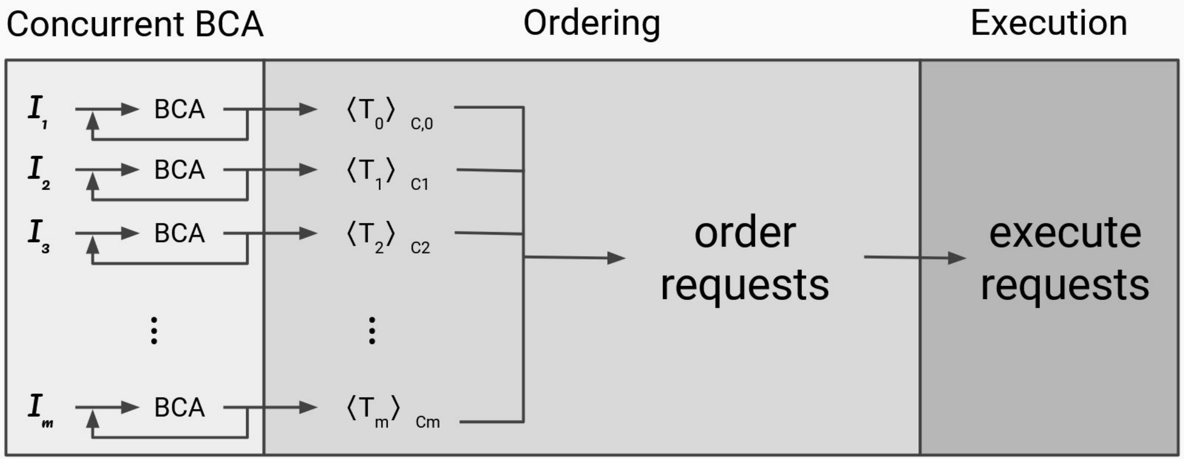
- **Improve** and **simplify** basic HotStuff protocol
- View changes with every **PREPARE** phase
- All phases have identical structure
- Reduced message types - **NEW-VIEW** and **GENERIC** message

Resilient Concurrent Consensus (RCC)



I_i refers to instance i
T_i refers to transaction i
C_m refers to m client transaction

Issues with RCC



- Eliminates single-primary bottlenecks and improved resilient, but:
- Introduces complexity and increased implementation costs,

Introduction to SpotLess

SpotLess:

- A concurrent rotational consensus protocol addressing challenges in HotStuff and RCC
- Chained + Concurrent consensus -> Reduced message complexity and latency
- Eliminates threshold signatures and error-prone view-change protocols
- Maintains robustness during unreliable communication through Rapid View Synchronization (RVS)
- Promises improved scalability, reduced message complexity, and high throughput
- Ideal for high-performance Resilient Data Management Systems (RDMSs) and reliability-focused distributed systems

Preliminaries

PRELIMINARIES: System

- Fixed set of replicas R
- **n= |R|** denotes the number of replicas
- Every replica r in R has a unique identifier id with $\mathbf{0 \leq id(R) < n}$
- **f** denotes the number of faulty replicas
- $\mathbf{n > 3f}$
- All clients can be malicious

PRELIMINARIES: Consensus

- SpotLess decides the sequence of client requests executed by non-faulty replicas in the system $\mathfrak{R}$.

- Three Consensus guarantees:

  - **Termination** - *If non-faulty replica* $R \in \mathfrak{R}$ *decides upon an* ϱ*-th client request, then all non-faulty replicas* $Q \in \mathfrak{R}$ *will decide upon an* ϱ*-th client request*

  - **Non-Divergence** - *If non-faulty replicas* $R_1, R_2 \in \mathfrak{R}$ *make* ϱ*-th decisions* $\tau 1$ *and* $\tau 2$*, respectively, then* $\tau 1 = \tau 2$ *(they decide upon the same* ϱ*-th client request).*

  - **Service** - *Whenever a non-faulty client* c *requests execution of* τ*, then all non-faulty replicas will eventually decide on a client request of* c*.*

PRELIMINARIES: Communication

- Asynchronous communication is assumed, i.e., messages can get lost or arbitrarily delayed.

- The partial synchrony model of PBFT is adopted to guarantee:
  - Non-Divergence or Safety
  - Liveness during periods of reliable (synchronous) communication

- Another assumption: Periods of unreliable communication are always followed by sufficiently long periods of synchronous communication.

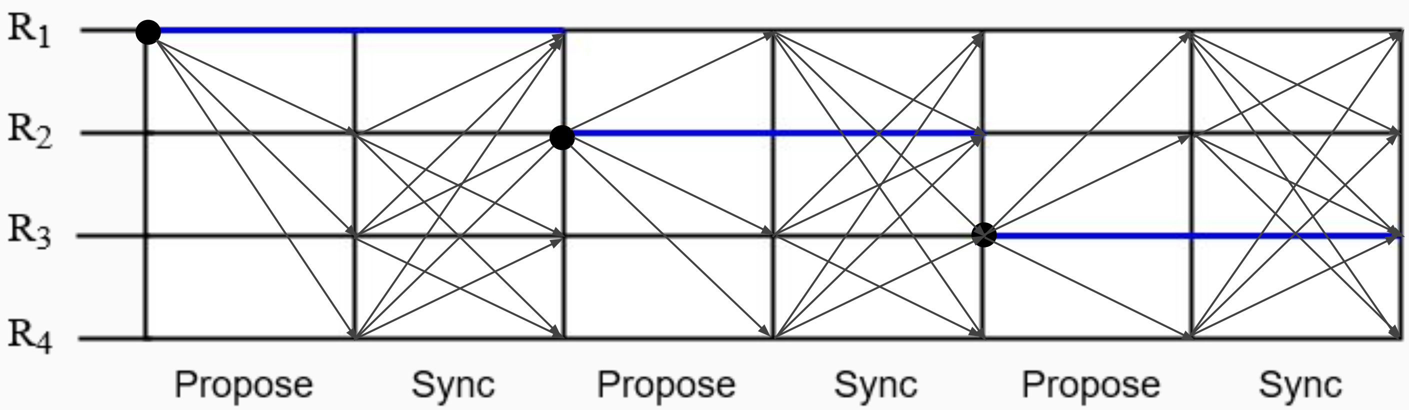
PRELIMINARIES: Authentication

- Assume that faulty replicas can impersonate each other but cannot impersonate non-faulty replicas

- Mechanisms:
    - Message authentication codes (MACs) does not guarantee that messages are tamper-free
    - Digital Signatures (DS)

Spotless Design Principles

1. Normal-Case Replication Protocol



view $\mathbf{v}$ - 1 ,
request τ_1 ,
conditionally prepare τ_1

2. Rules Guaranteeing Safety

Definitions:

1. **Safety** — Non-Divergence:
   If non-faulty replicas $R_1, R_2 \in \mathfrak{R}$ make ϱ^{th} decisions τ_1 and τ_2, respectively, then $\tau_1 = \tau_2$ (they decide upon the same ϱ^{th} client request).

2. **Preceding** — Proposal P_1 precedes proposal P_2 if $\exists$ proposal P' such that P_1 precedes P' and P' is the preceding proposal of P_2.

   If precedes(m) = {Proposals preceding m}, depth(m) = |precedes(m)|.

2. Rules Guaranteeing Safety

Definitions:

3. **Record** — Replica R records proposal m if it decides that m = PROPOSE(v, τ, cert($\mathbb{P}$)) is a **well-formed** proposal, which is the case if:
   - $(|m|)_P$ is a valid digital signature;
   - τ is a valid client request;
   - View v is the current view;
   - If R agrees with cert($\mathbb{P}$) (if R has not conditionally prepared $\mathbb{P}$)

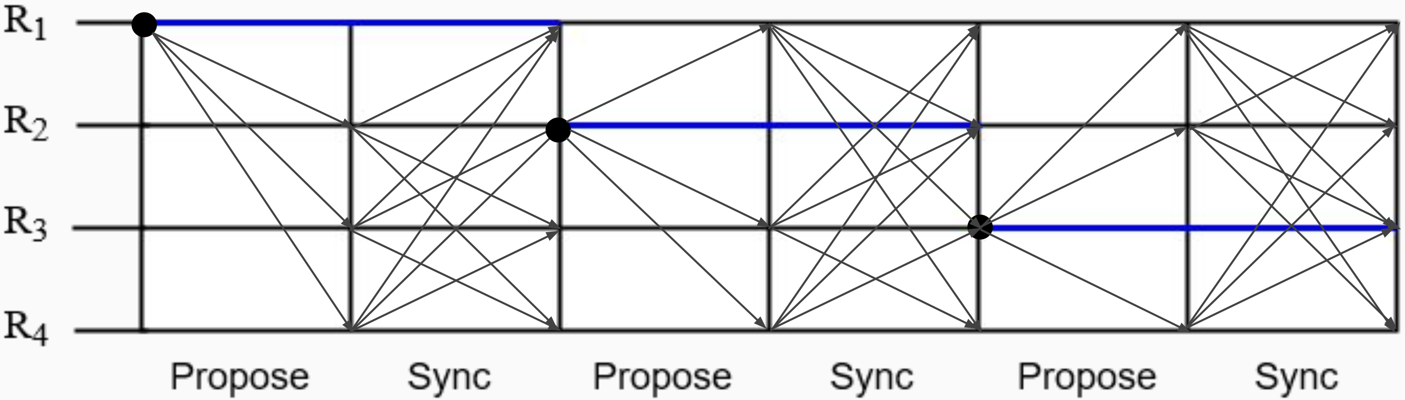
4. **Accept** — Replica R accepts proposal m if it broadcasts Sync messages with claim(m); a way of saying that m is acceptable.

2. Rules Guaranteeing Safety

Definitions:

5. **<u>Conditional Prepare</u>** — A replica R conditionally prepares proposal m if:
   - R received m
   - During view v, R receives Sync messages from n−f (non-faulty) replicas.

6. **<u>Conditional Commit</u>** — A replica R conditionally commits proposal m if, in a future view $w > v$, R conditionally prepares a proposal m' that extends m.



view v
request m'
conditionally prepare m

view $w = v+1$
request m''
conditionally commit m
conditionally prepare m'

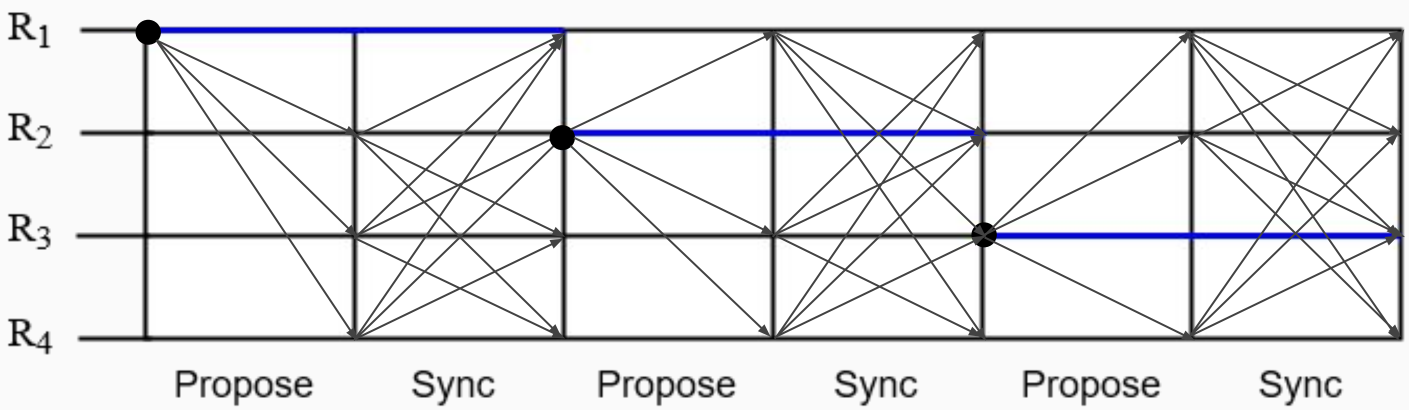
view $u = w+1 = v+2$
request m'''
commit m
conditionally commit m'
conditionally prepare m''

2. Rules Guaranteeing Safety

Definitions:

7. **Lock** — Replica R locks proposal m if $m = \mathbb{P}_{\text{lock}}$, the highest proposal that R has conditionally committed so far.

8. **Commit** — Replica R commits proposal m if in a future view $u > v$, R conditionally prepares a proposal m'' that extends m', with $u = w + 1 = v + 2$.



view v
request m'
conditionally prepare m

view w = v+1
request m''
conditionally commit m
conditionally prepare m'

view u = w+1 = v+2
request m'''
commit m
conditionally commit m'
conditionally prepare m''

2. Rules Guaranteeing Safety

Definitions:

9. **<u>Conflict</u>** — Two proposals are conflicting if the sequence of preceding proposals of these two proposals are disjoint.

10. **<u>Extendability</u>** — A non-faulty primary P considers a proposal $\mathbb{P}'$ to be extendable if:
    (E1) P has a valid certificate for $\mathbb{P}'$; or
    (E2) P has received a set of Sync messages from n − f replicas that claim to have conditionally prepared $\mathbb{P}'$.

2. Rules Guaranteeing Safety

Safety Rules:

(A1) **Validity Rule:** R has conditionally prepared $\mathbb{P}$.

(A2) **Safety Rule:** m extends R's locked proposal $\mathbb{P}_{lock}$, $\mathbb{P}_{lock} \in (\{\mathbb{P}\} \cup \text{precedes}(\mathbb{P}))$.

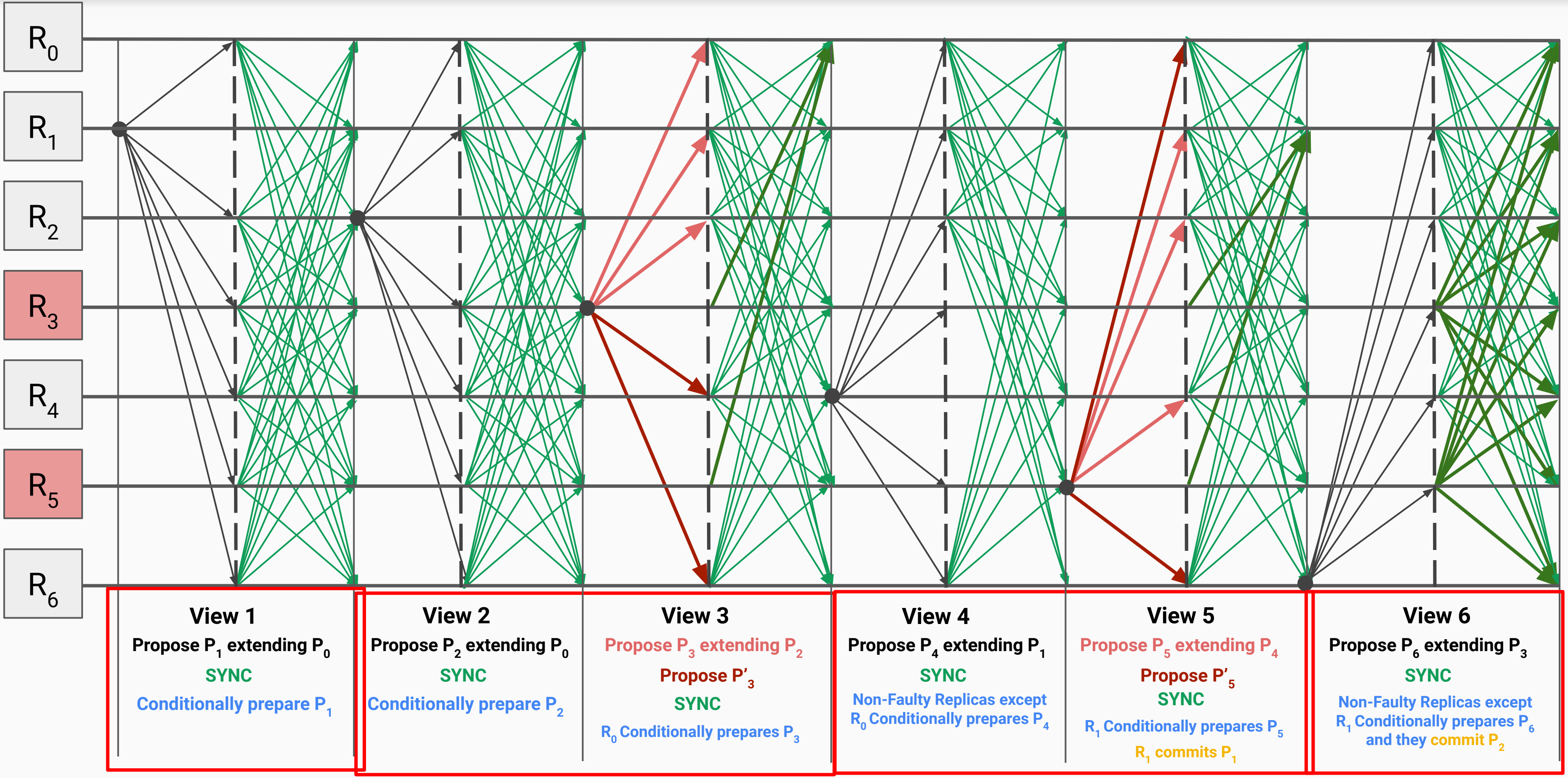
(A3) **Liveness Rule:** $\mathbb{P}$ is a proposal for a higher view than $\mathbb{P}_{lock}$.

2. Rules Guaranteeing Safety

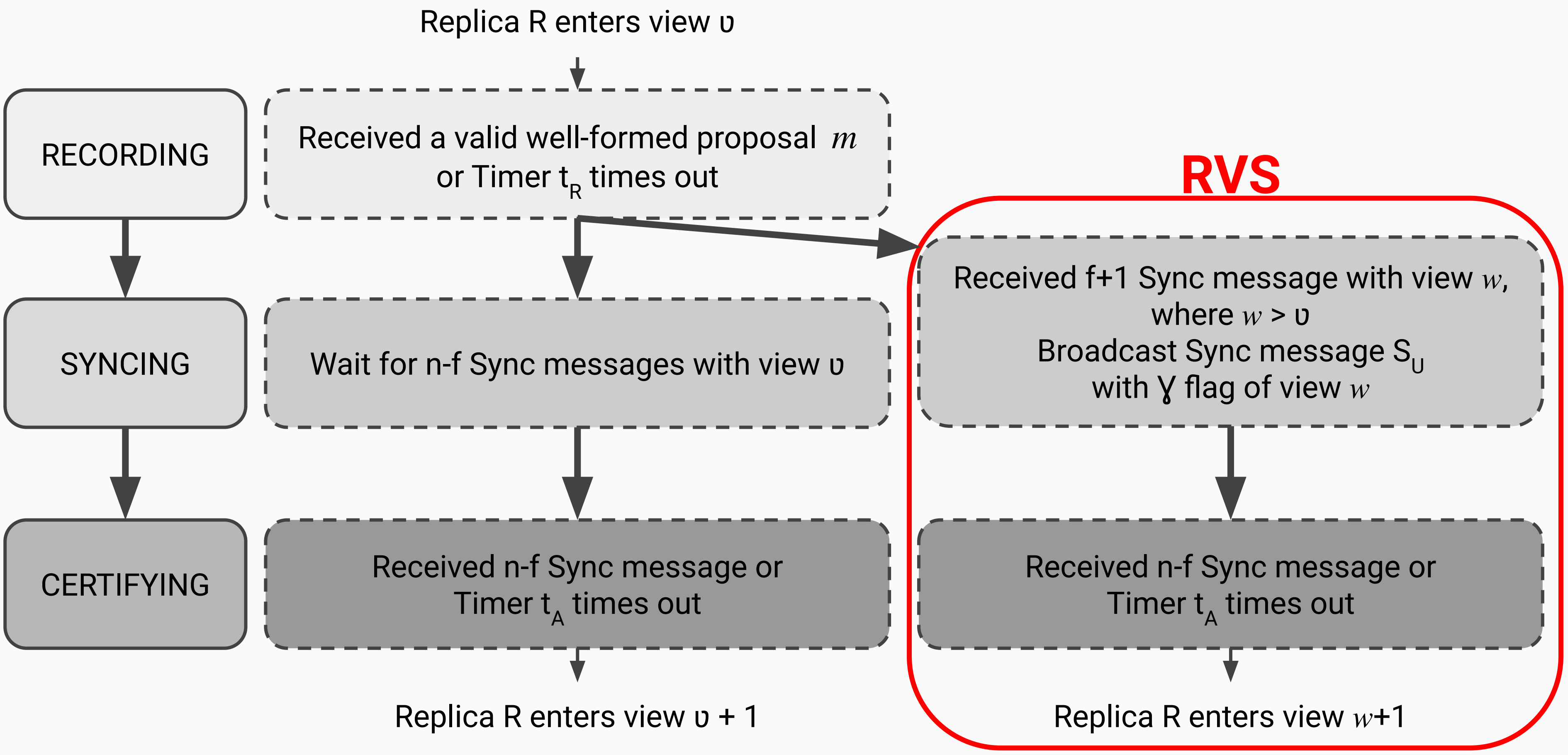
Lemma: If a non-faulty replica R conditionally prepares m, then for each proposal $\mathbb{P}'$ preceding m, at least $n - 2f \geq f + 1$ non-faulty replicas will have conditionally prepared $\mathbb{P}'$ and sent Sync messages with $\mathbb{P}' \in C\mathbb{P}$.

Theorem: No two non-faulty replicas can commit two conflicting proposals.

3. Why three-consecutive-view ?



3. Rapid View Synchronization



4. Timers and Resending Mechanism

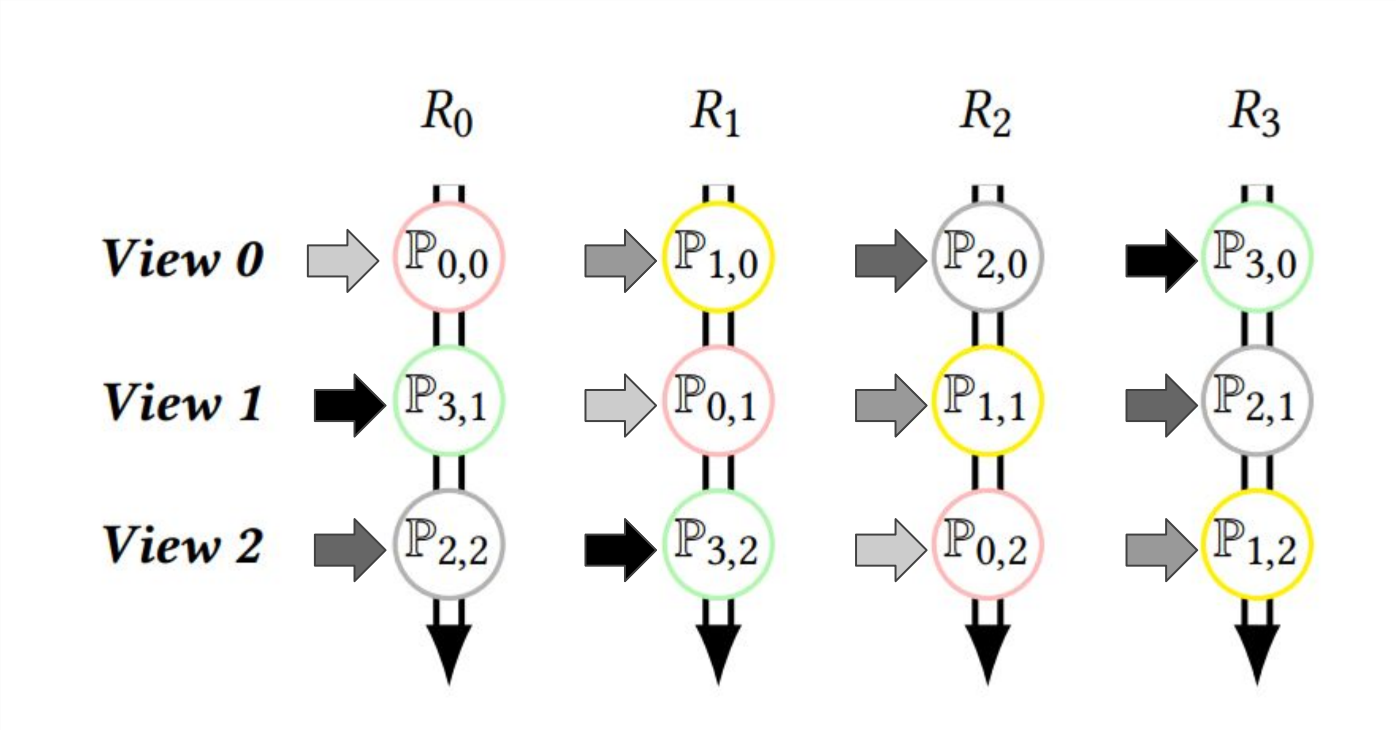
- R requires n-f Sync messages to switch from Syncing to Certifying state
- R cannot learn about a path until it receives f+1 replies to its Sync messages with flag ɣ.
- R cannot record a proposal it did not receive from the primary unless it receives a reply to its ASK message

4. Timers and Resending Mechanism

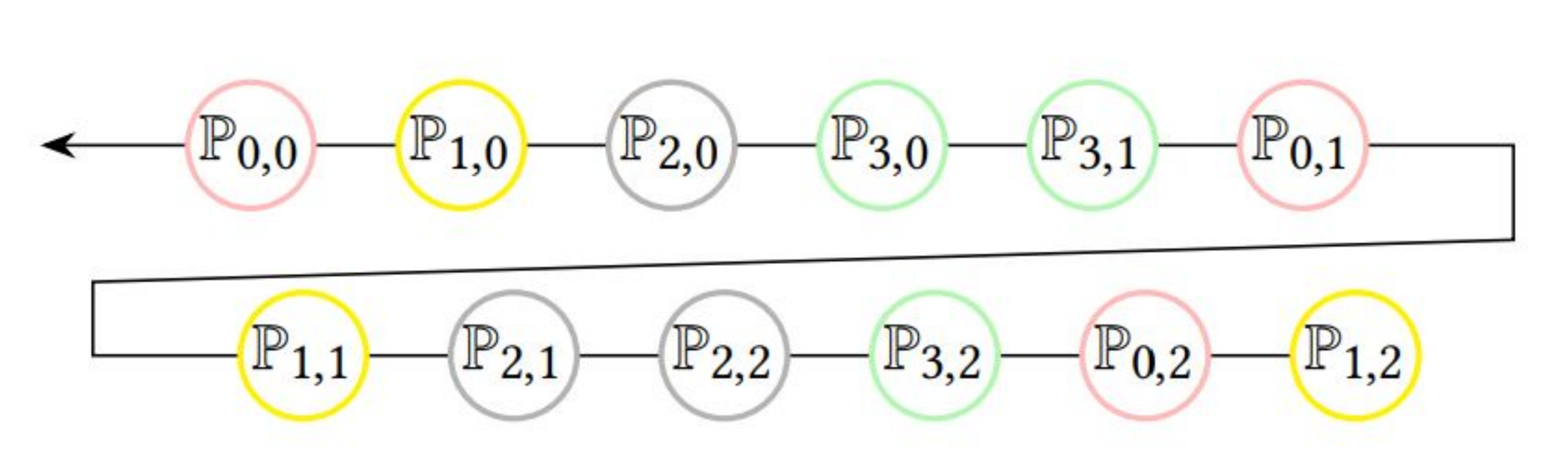
- Due to unreliable communication, R may fail to receive replies to Sync messages
- R periodically resends the message, until it receives necessary replies
- Spotless adjusts the timeout interval to detect replica failures
- For consecutive timeouts of the same timer in consecutive views, we only increase the timeout interval by a constant ε (after each consecutive view).
- If a replica receives an expected message for which the timeout interval was Δ before 0.5Δ, then the replica reduces the timeout by half.

Concurrent Consensus

1. Concurrent instance in SpotLess



1. Concurrent instance in SpotLess



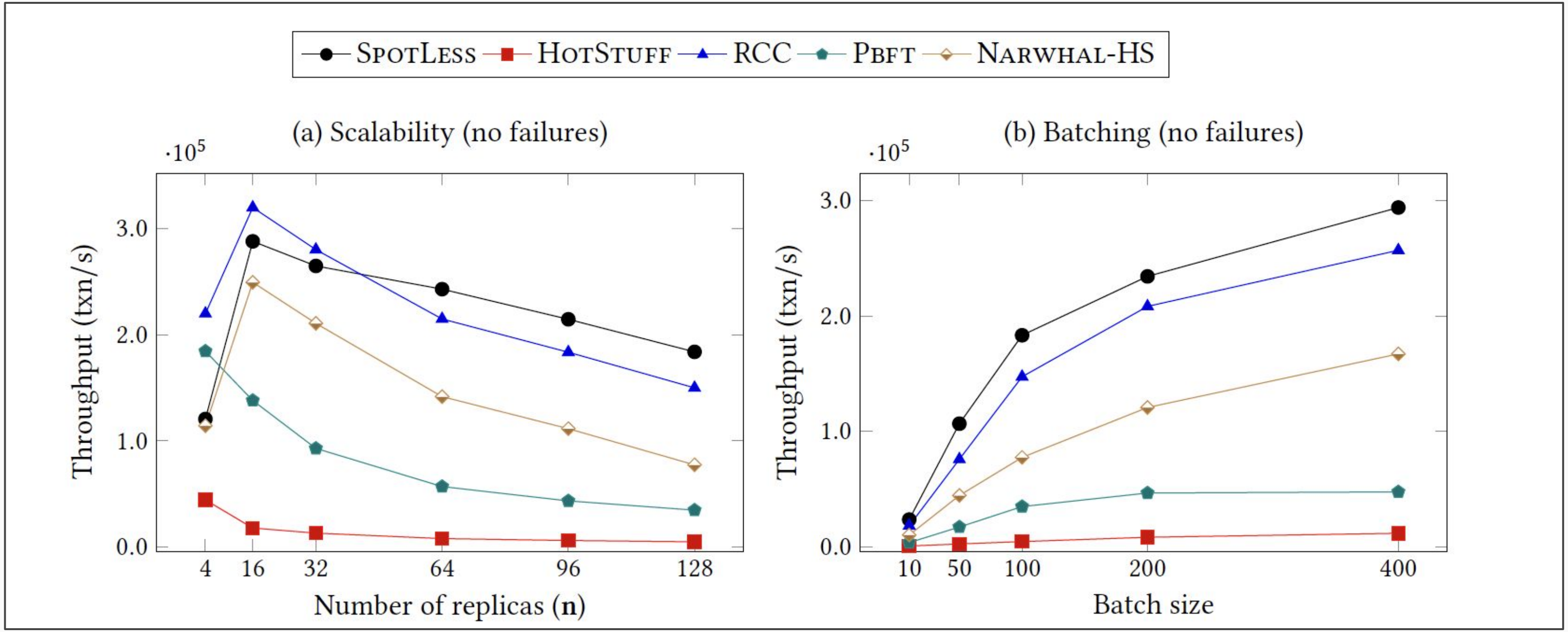
2. Benefits of concurrent processing

- **Scalability:** If m ($m \leq n$) instances run concurrently, SpotLess will achieve m times more throughput comparing to that of a single instance

- **Lower communication cost:** chained design of SpotLess allows instances to send few messages per decision than PBFT

Evaluation Of SpotLess

Evaluation Of SpotLess



Conclusion

SpotLess - Conclusion

- Combines concurrent consensus architectures and chained consensus architecture
- Low-cost recovery via Rapid View Synchronization Protocol
- Spotless outperforms traditional primary-backup consensus protocols
- Maintains stable latency and high throughput during failures

Introduction (Sasha)

Intro and preliminary

Overview of PBFT, HotStuff, RCC (Focus on **PBFT**)

PBFT design schema (3 phases)

- Transition to spotless design schema

Guidelines

Minimal text

Check that every slide has a purpose

- Go straight to the main point

On your own review PBFT phases (why 3f+1, 2f+1, f+1)

Spotless Design (Rishika and Anh, Sasha and Gopal join later)

Diagrams (focus on this)

- Broken into different parts with slide transition (phases animation)
- Show how it differs from PBFT phases

Example 3.6: three-consecutive-view requirements

- Why two-consecutive-view doesn't work?
- diagram

Rapid view synchronization

Timers and resending (briefly)

Concurrent consensus (1 person)

Figure 5 and 6

Section 5-8 (all 4)