Towards practical and efficient performance robustness: QuePaxa and beyond

Bryan Ford – EPFL

Dagstuhl Seminar 24362
“Next-Generation Secure Distributed Computing”
September 4, 2024

QuePaxa: Escaping the Tyranny of Timeouts in Consensus

Pasindu Tennage
EPFL

Cristina Basescu
EPFL & Digital Asset

Eleftherios Kokoris Kogias
ISTA & Mysten Labs

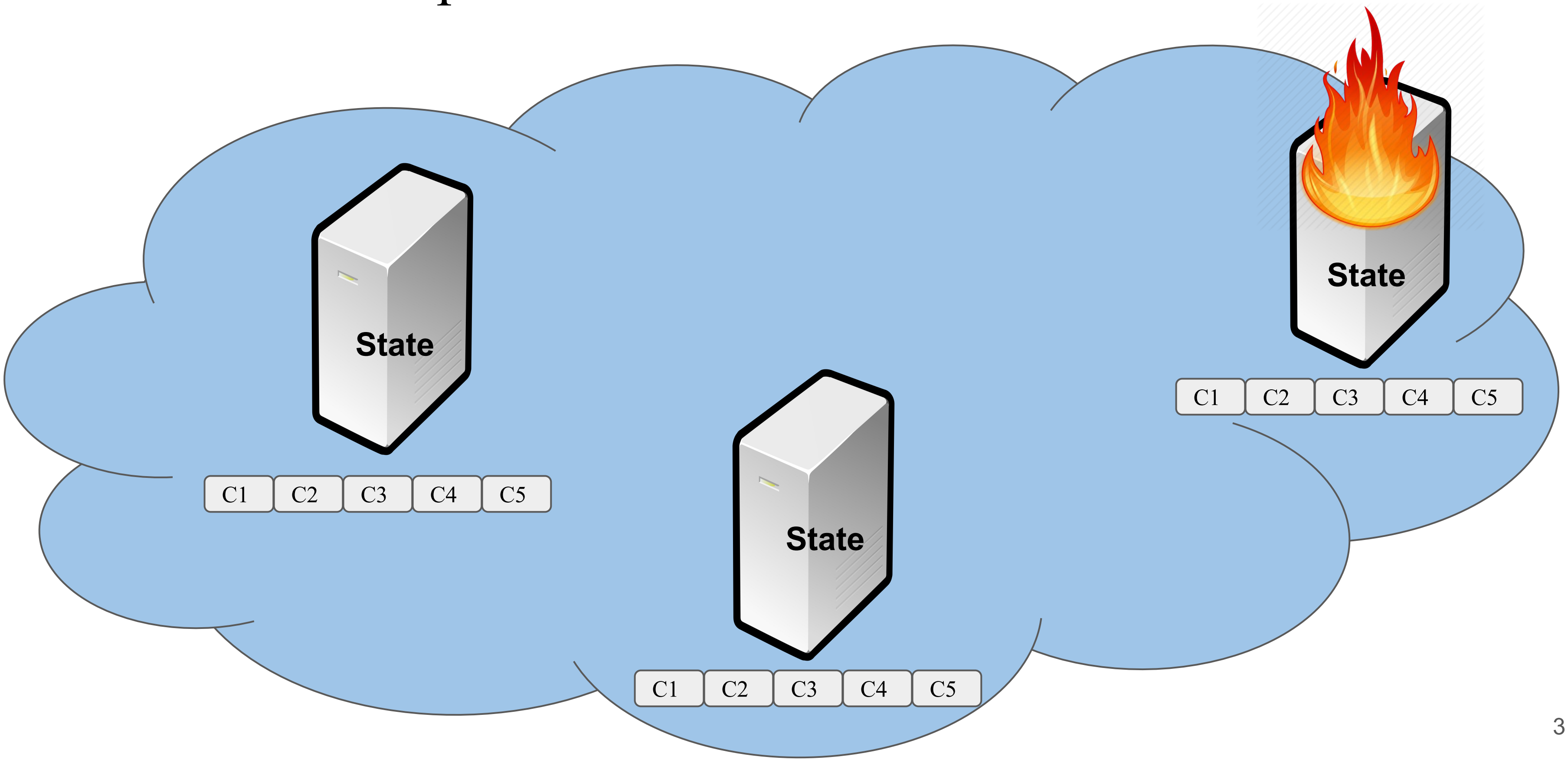
Ewa Syta
Trinity College

Philipp Jovanovic
UCL

Vero Galiñanes
EPFL

Bryan Ford
EPFL

Consensus and Replicated State Machine



Consensus and Replicated State Machine

Paxos Made Simple

Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems

Generalized Consensus and Paxos

Leslie Lamport

3 March 2004
revised 15 March 2005
corrected 28 April 2005

There Is

Iulian Moran
Carne

Semi-Decentralized Geo-replicated
te Machines

Quanlu Zhang
icrosoft Research
.Zhang@microsoft.com

Zhi Yang*
Peking University
yangzhi@pku.edu.cn

Ming Wu
Microsoft Research
miw@microsoft.com

Yafei Dai
Shenzhen Key Lab for Information
Centric Networking & Block Chain

WPax

S-Paxos: Offloading the Leader for High

PigPaxos: Devouring the Communication Bottlenecks in Distributed Consensus

Aleksey Charapko*
University of New Hampshire
Aleksey.Charapko@unh.edu

Ailidani Ailijiang
Microsoft
Ailidani.Ailijiang@microsoft.com

Murat Demirbas
University at Buffalo, SUNY
demirbas@buffalo.edu

Google Chubby lock service

Apache Zookeeper

dragonboat

Spanner

TiKV

NebulaGraph

SOFASTACK

Dimensions of robustness in (permissioned) consensus

Failure model: crash-stop or byzantine? (This talk’s focus: crash-stop)

Threshold: tolerant of how many failures? (Typically $2f < n$ for crash-stop)

Network model: synchronous, partially synchronous, asynchronous?

Normal-case performance (throughput, latency) and efficiency (compute, BW)

Worst-case performance (throughput, latency) and efficiency (compute, BW)

Recovery time after failure, responsiveness, …

What we would like versus what actually gets deployed

What we would like in principle: asynchronous Byzantine consensus everywhere

- Robust to adversarial node failures *and* adversarial network behavior

What actually gets deployed almost everywhere: Paxos, Multi-Paxos, Raft

- Partially synchronous, crash-stop failures only

Why? Paxos et al offers:

- Low latency: 1-round-trip commit in the normal case
- Efficiency: O(n) normal-case bandwidth per commit
- Relatively simple, “good enough” for most deployment scenarios

Introducing QuePaxa – key contribution in a nutshell

QuePaxa is the first crash-stop consensus protocol that achieves:

- Same 1-round-trip normal-case commit latency as Paxos etc.
- Same O(n) normal-case bandwidth consumption as Paxos etc.
- Performance robustness of full asynchronous consensus in the worst case
  - Guaranteed liveness even during periods of asynchrony
  - Protocol makes progress at rate the network communication permits
  - O(1) expected round-trips to commit w.h.p.
- Experimentally performance-robust also in “medium-bad” but non-worst cases
  - Temporary network delays, node slowdowns, DoS attacks against minority of nodes, …
- Not *much* more complex/difficult to implement than Paxos etc.
  - Full pseudocode of QuePaxa algorithm fits easily on 1 page

RoadMap

- Introduction to consensus
- **Tyranny of timeouts**
- Parallels of QuePaxa and hedging
- QuePaxa algorithm
- Evaluation

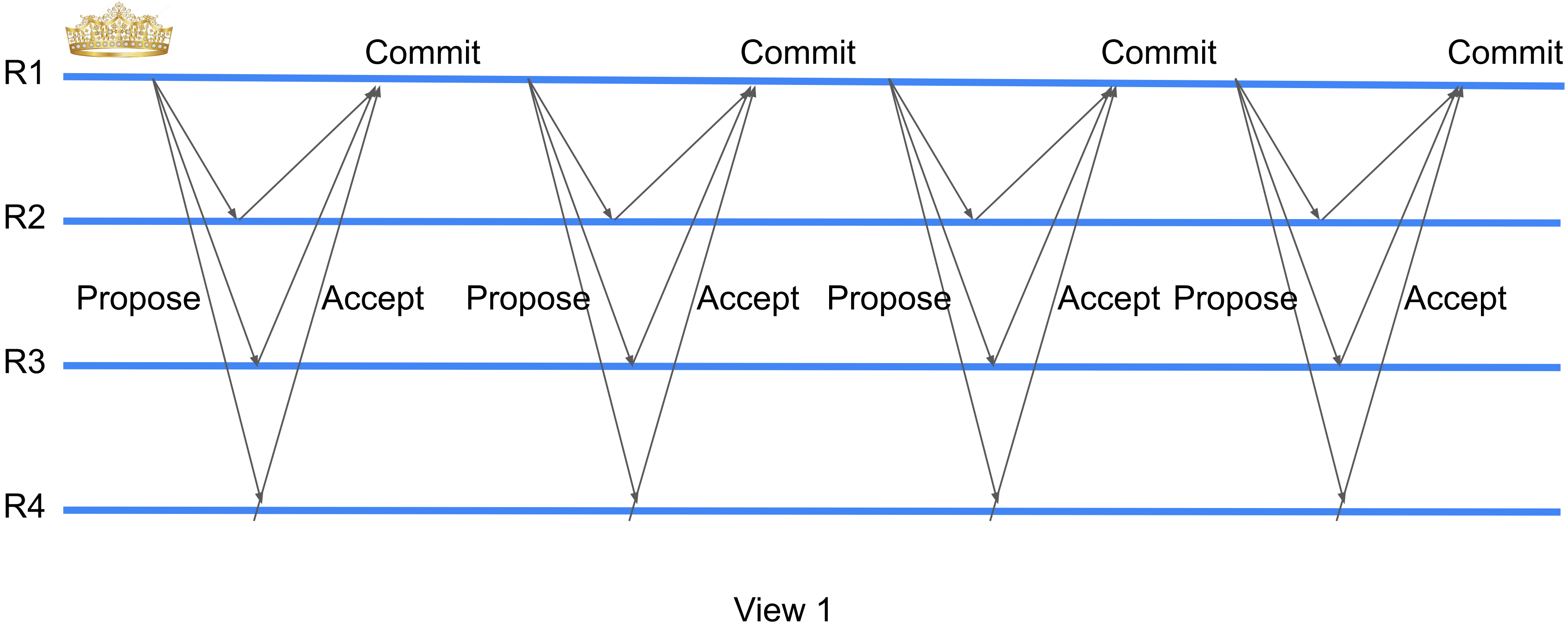
Tyranny of Timeout Problems in Consensus

Timeout based view change

Conservative timeouts

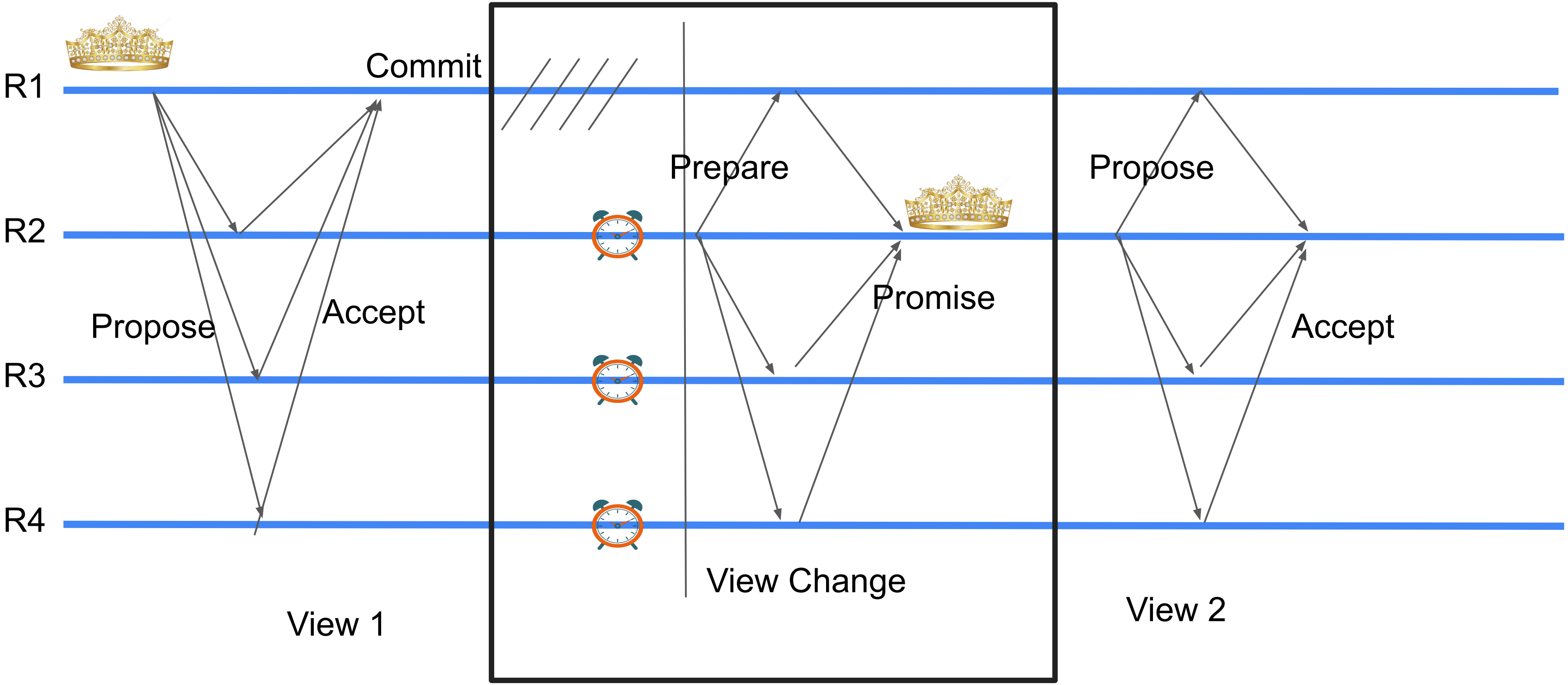
Manually configured timeouts

Timeout based view change [Multi-Paxos]



As long as the network is synchronous, the leader will keep committing new requests

Timeout based view change [Multi-Paxos]



No new commands are committed during view change
Liveness depends on partial synchronous network conditions

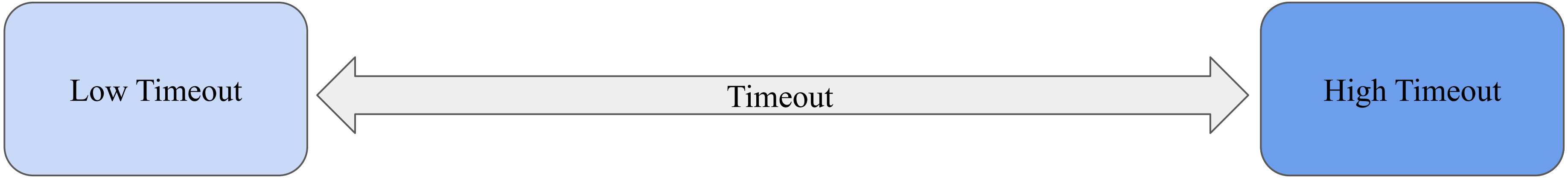
Tyranny of Timeout Problems in Consensus

Timeout based view change

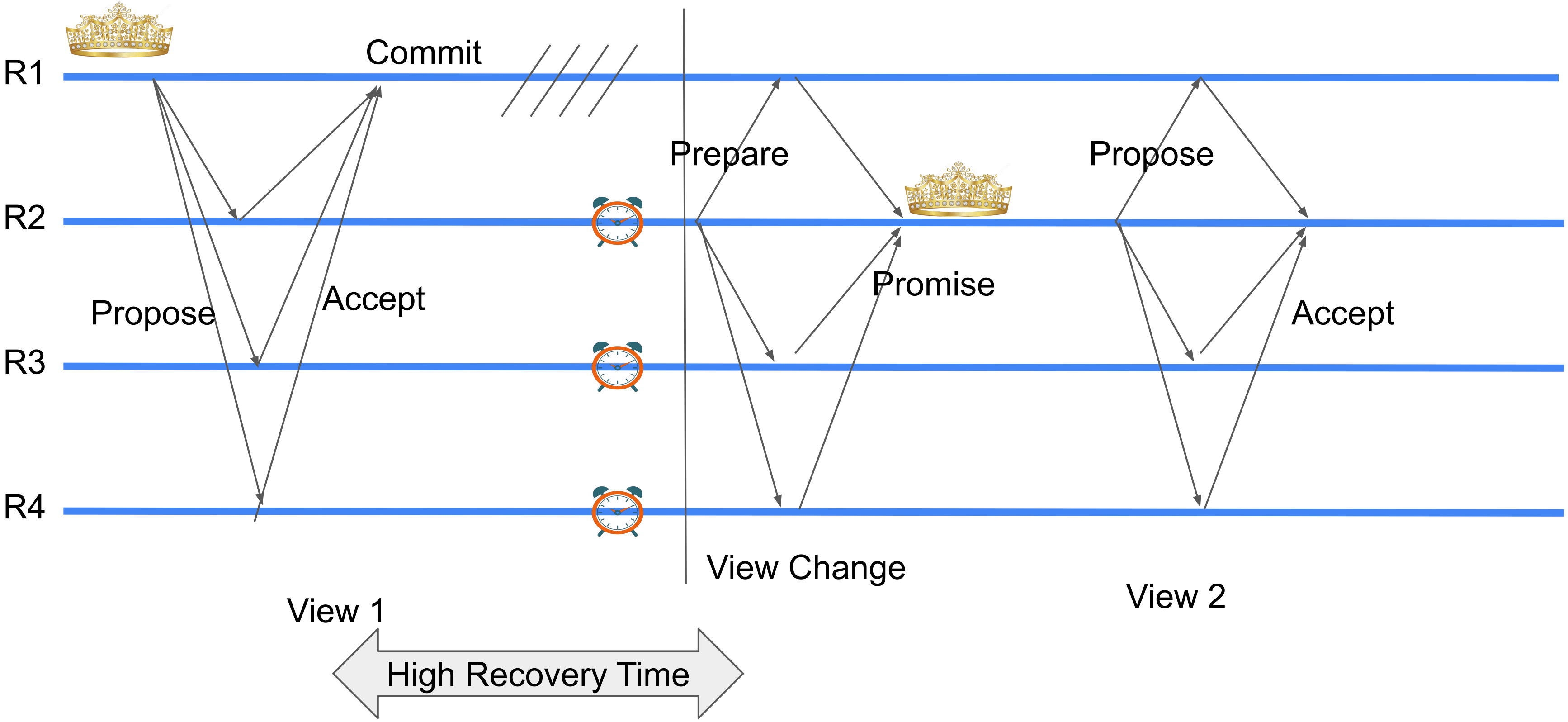
Conservative timeouts

Manually configured timeouts

Choosing Timeouts in leader based protocols



Timeout based view change [Multi-Paxos]



High timeouts result in high recovery time

Choosing Timeouts in leader based protocols

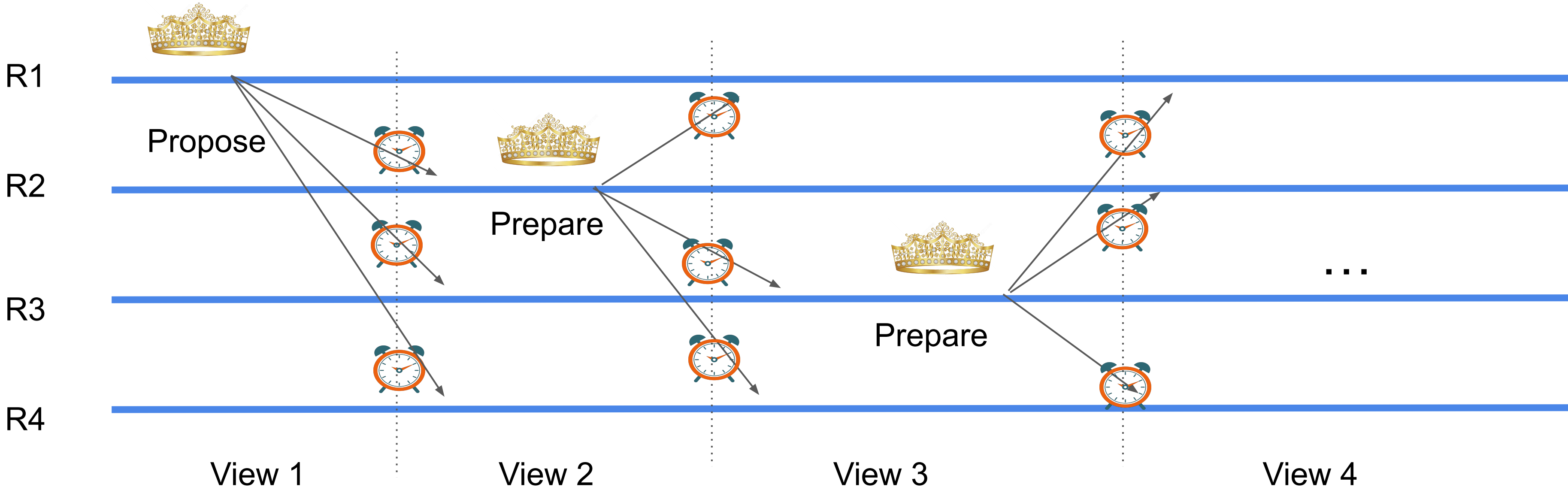
Low Timeout

Timeout

High Timeout

High Recovery Time

Liveness loss with low timeouts



No commands are committed when the timeout is low

Choosing Timeouts in leader based protocols

Low Timeout ⟷ Timeout ⟷ High Timeout

Liveness Loss

High Recovery Time

Both choices of timeouts have negative consequences

Tyranny of Timeout Problems in Consensus

Timeout based view change

Conservative timeouts

Manually configured timeouts

Manual configuration of timeouts

- Stuck with a live but slow leader replica

- Do not consider dynamic network state for leader election

Manual timeouts are sub optimal

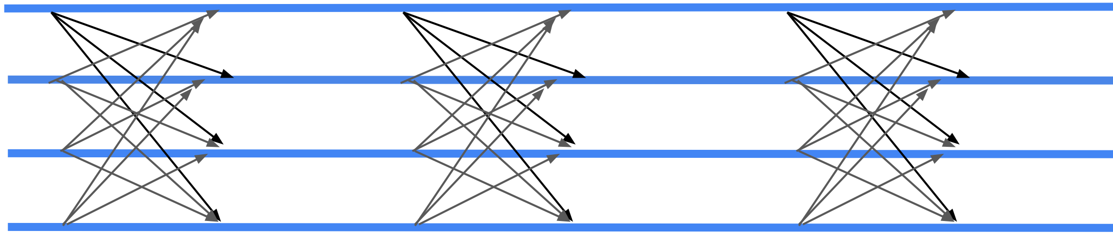
Are timeouts necessary for progress?

Can we eliminate the impact of timeout for liveness?

Do asynchronous protocols solve this problem?

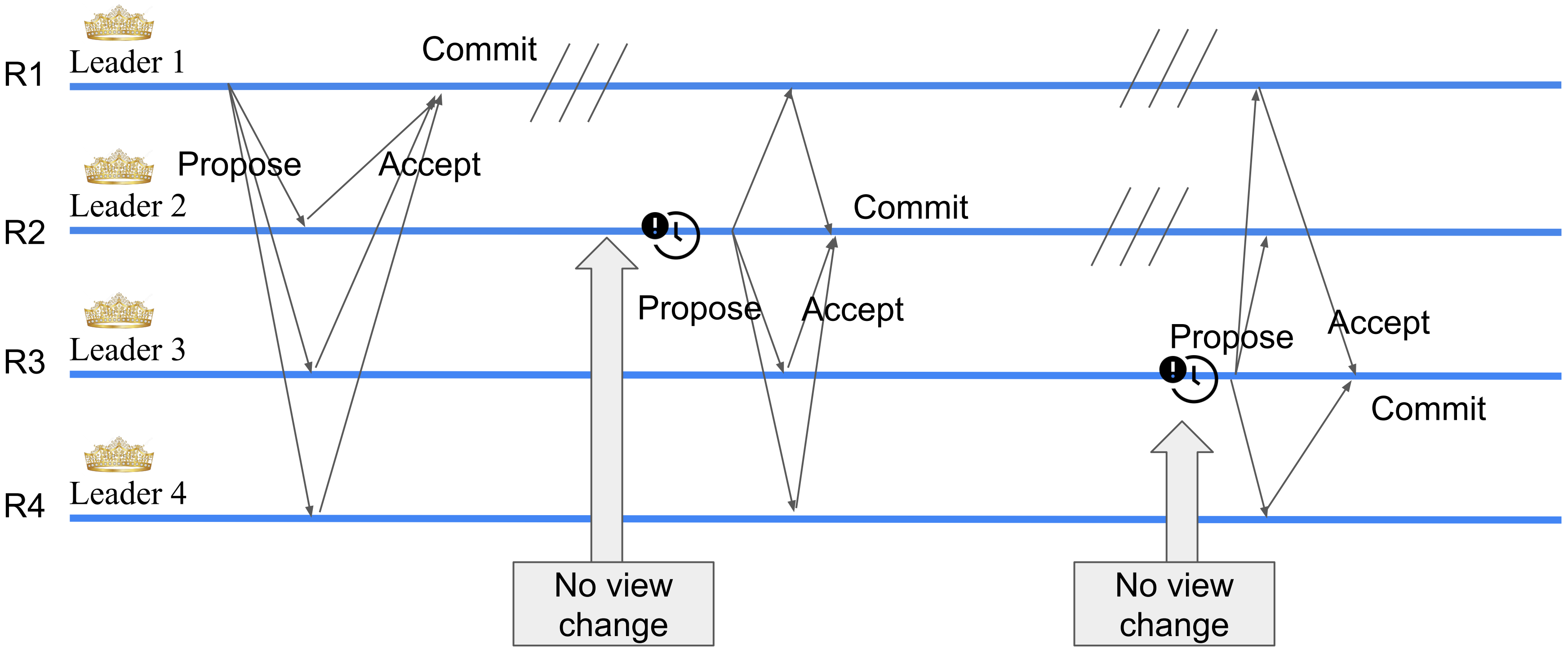
- Asynchronous protocols do not depend on timeout for progress
  - Use randomization to alleviate the FLP impossibility

- Message complexity
  - In general asynchronous protocols have $O(n^2)$ / $O(n^3)$ complexity in the normal case
    - Partially synchronous protocols have $O(n)$ complexity in the normal case
  - Less efficient than leader-based protocols
  - Hence rarely deployed



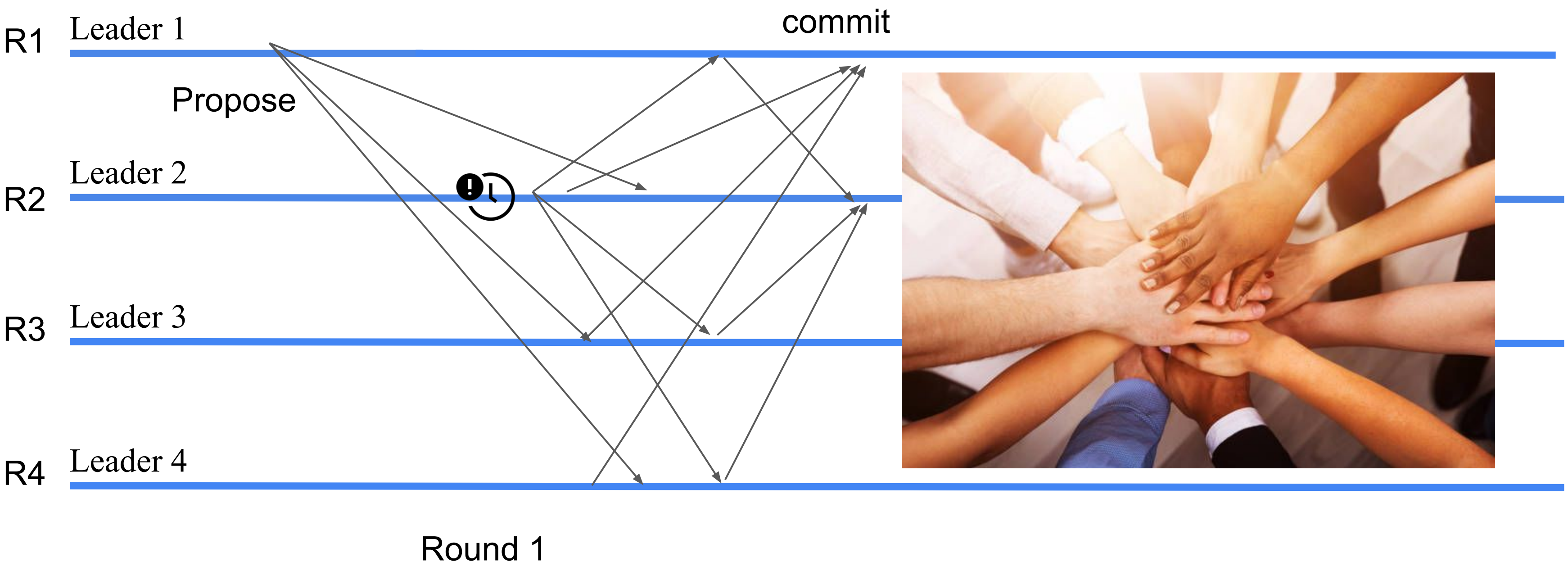
Asynchronous protocols are slow and rarely deployed

What if **multiple** leaders could propose **without** view changes?



Can we change leaders **without view changes** if the current leader is sub optimal?

What if multiple leaders could **cooperate** instead of **interfere**?



Can we support multiple proposers to be non destructive?

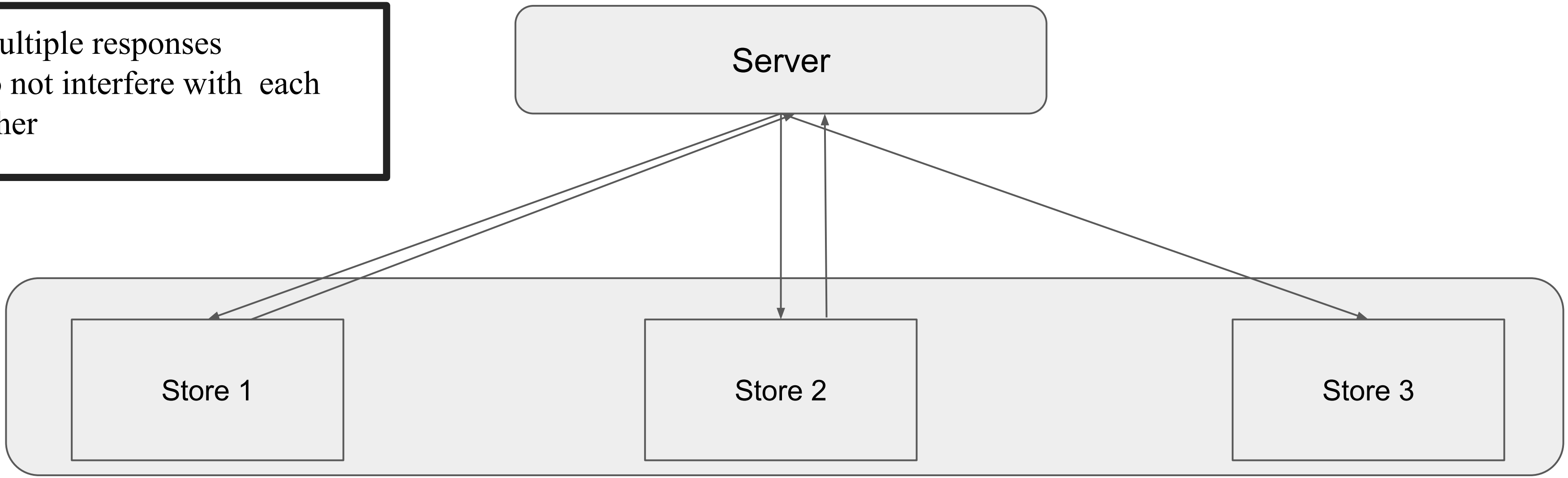
RoadMap

- Introduction to consensus
- Tyranny of timeouts
- **Parallels of QuePaxa and hedging**
- QuePaxa algorithm
- Evaluation

Hedging

- Hedging is a way to curb latency variability
  - Key idea: issue the same request to multiple replicas and use the results from whichever replica responds first

Multiple responses do not interfere with each other



Can we apply hedging to consensus so that multiple proposers don't interfere?

RoadMap

- Introduction to consensus
- Tyranny of timeouts
- Parallels of QuePaxa and hedging
- **QuePaxa algorithm**
- Evaluation

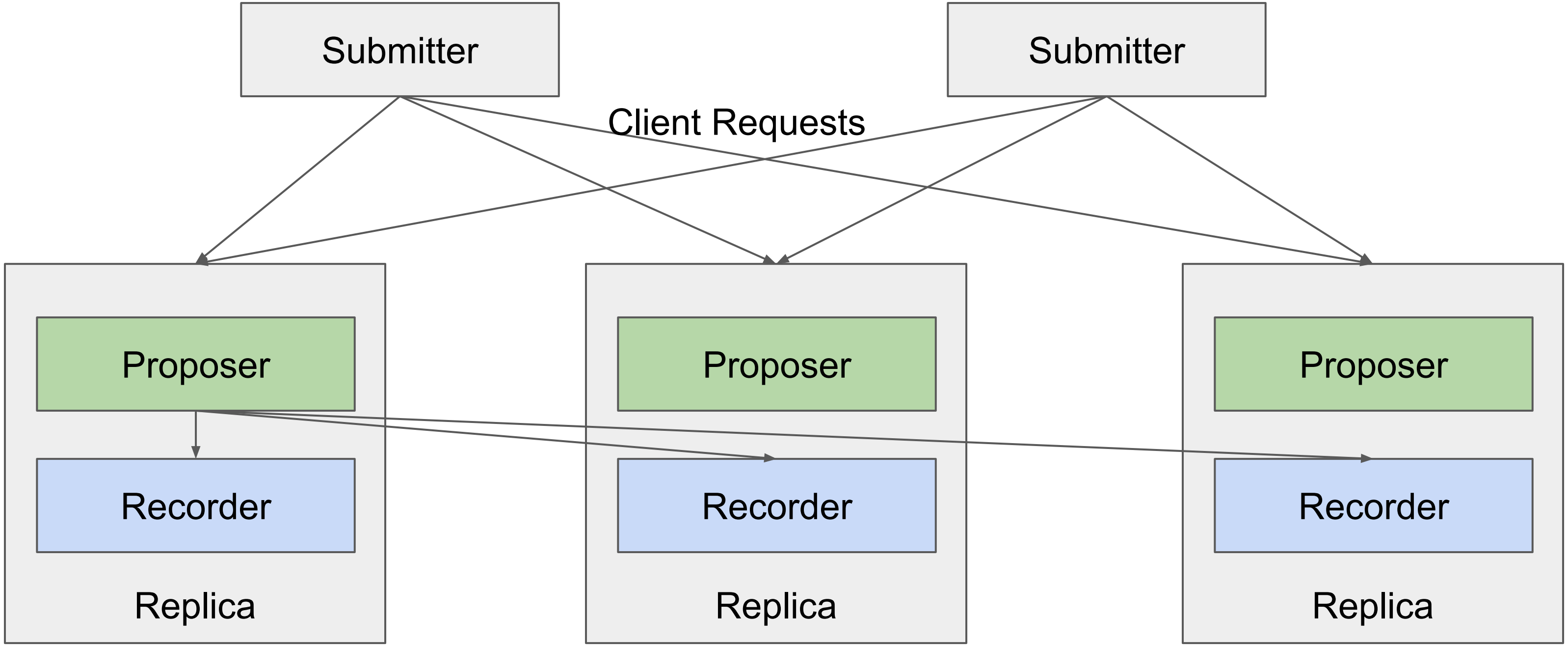
QuePaxa Contributions

- Eliminates the “tyranny of timeouts” for consensus liveness
- First consensus protocol to support hedging in consensus
- First protocol offering **efficiency** with **performance-robustness**
  - Under normal network conditions, just as efficient as Multi-Paxos/Raft
  - Under bad/high-delay/noisy network conditions, maintains performance
  - Under worst-case adversarial network conditions, maintains liveness

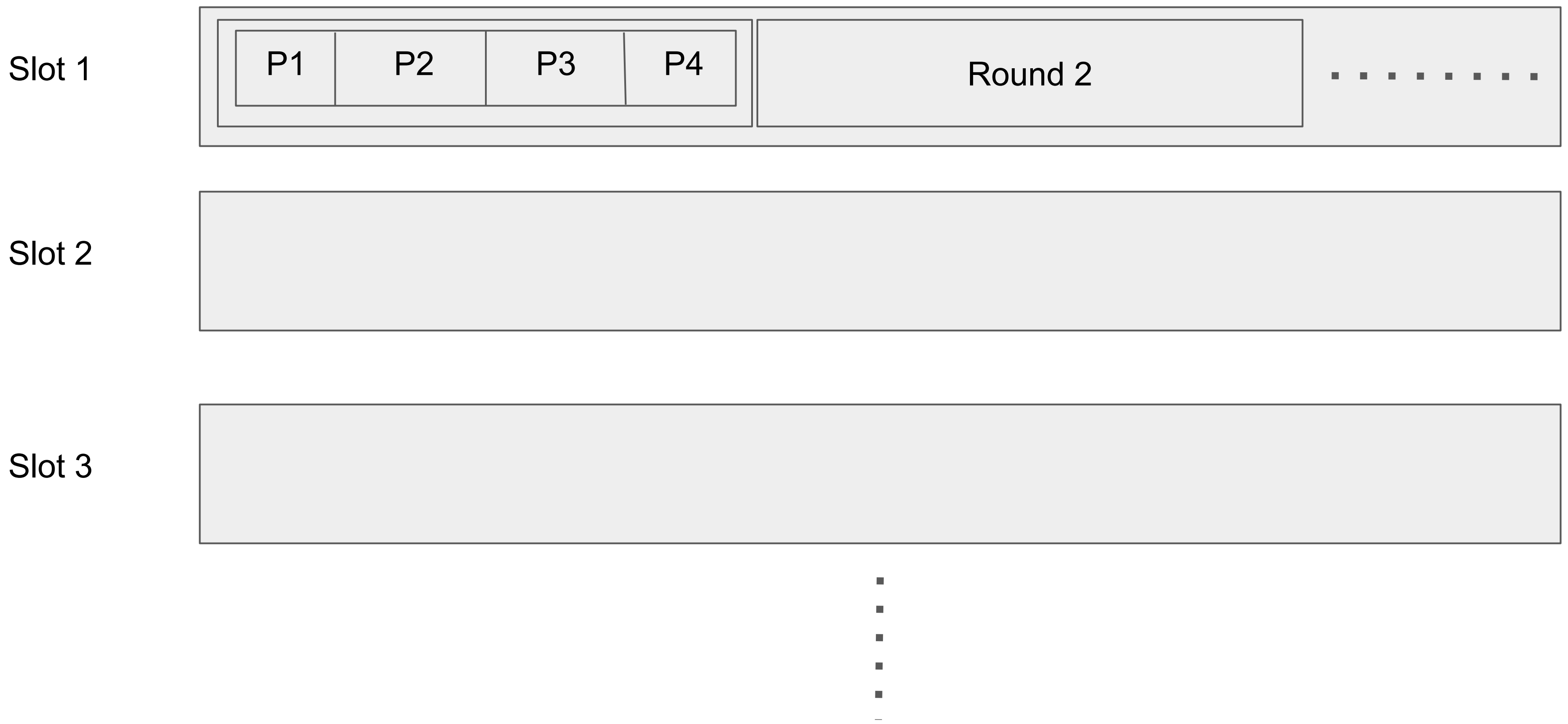
QuePaxa RoadMap

- Operation Overview
- Abstract QuePaxa – *a simplified version*
- Safety and liveness of abstract QuePaxa
- Concrete QuePaxa overview
- The QuePaxa fast path

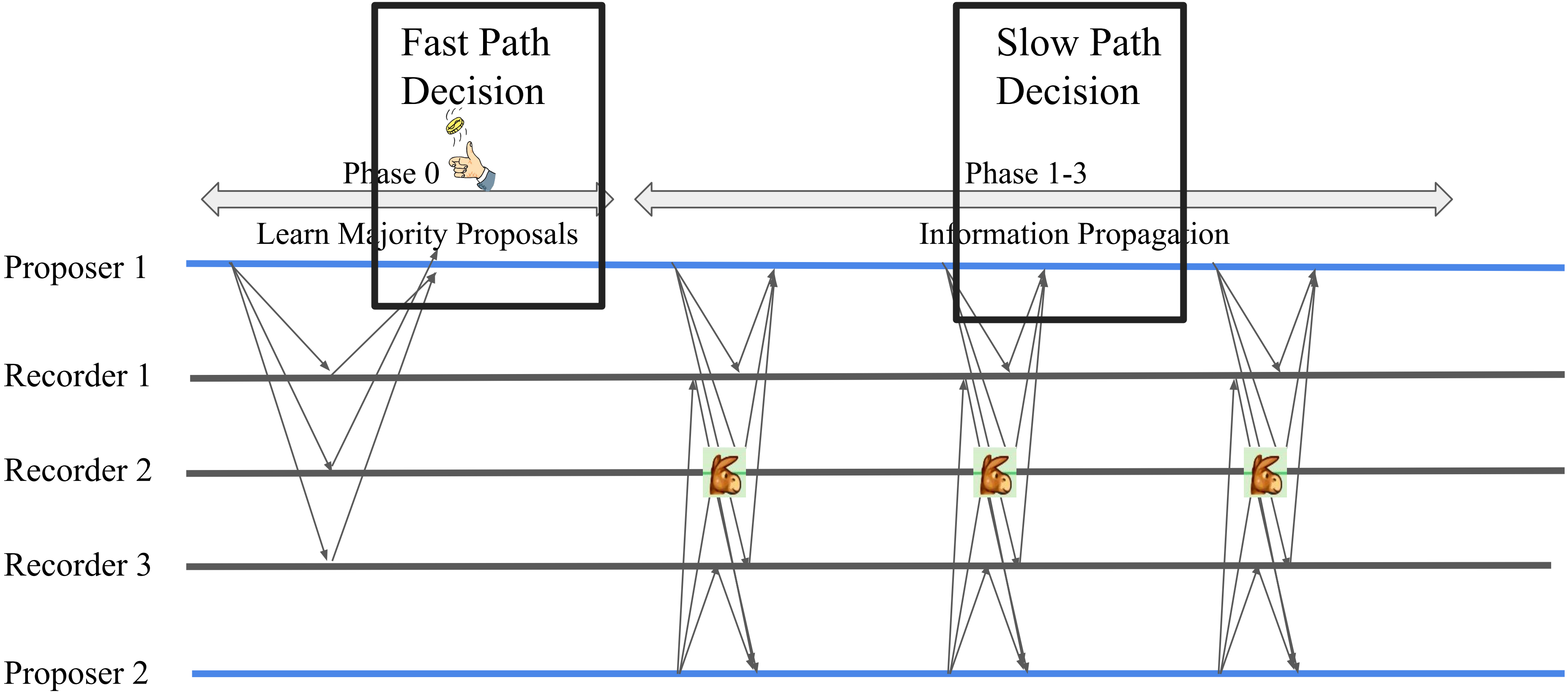
QuePaxa Architecture



QuePaxa Log Structure

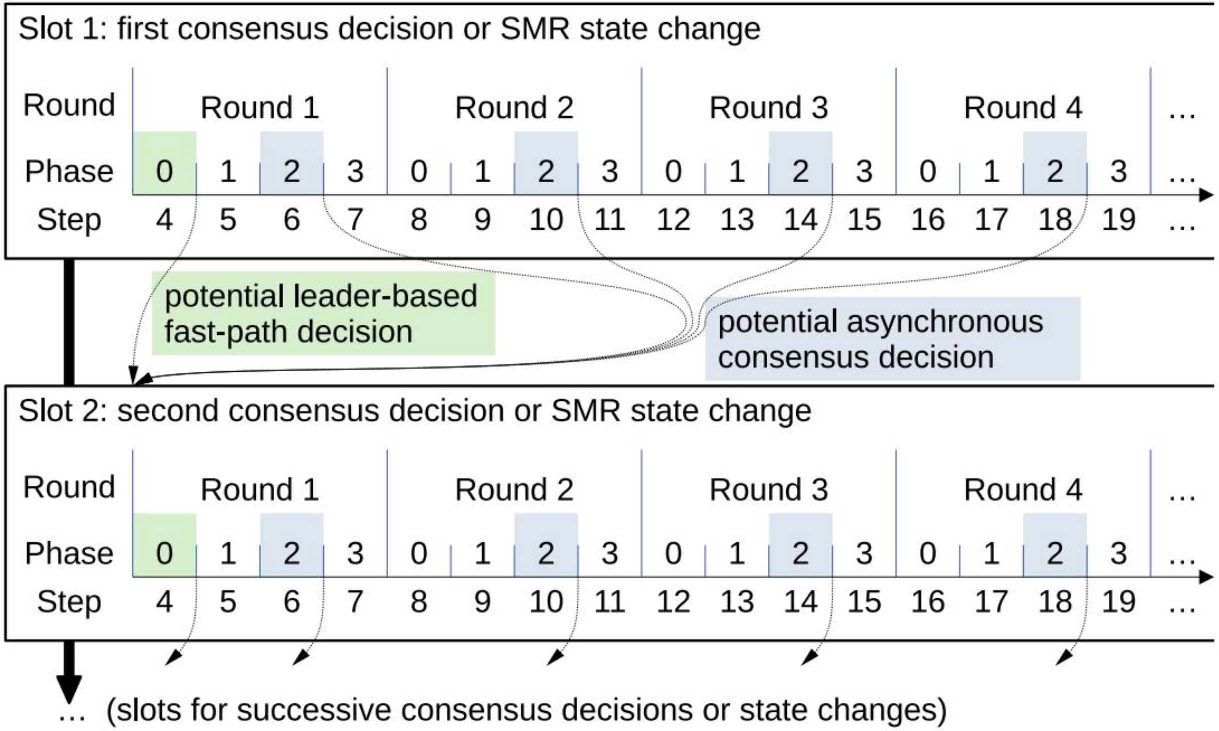


QuePaxa Protocol Diagram



QuePaxa has a fast path decision and a slow path decision

QuePaxa Log Structure



QuePaxa RoadMap

- Operation Overview
- **Abstract QuePaxa - *a simplified version***
- Safety and liveness of abstract QuePaxa
- Concrete QuePaxa overview
- The QuePaxa fast path

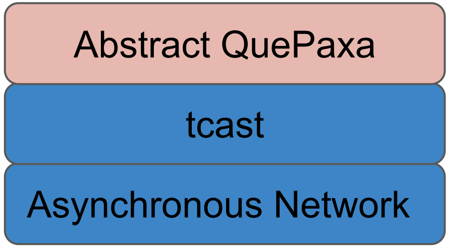
Abstract QuePaxa is a simplified version of QuePaxa

Introducing threshold broadcast (**tcast**)

- Divide the problem in to two parts
  - Handling replica failures
  - Handling asynchrony

- **First ignore asynchrony** and focus on replica failures
  - Assume an abstract synchronous **lock-step** network

- **tcast** (threshold synchronous broadcast): an abstraction providing lock-step synchrony to the consensus layer



Abstract QuePaxa assumes synchrony and solves the replica failure challenge

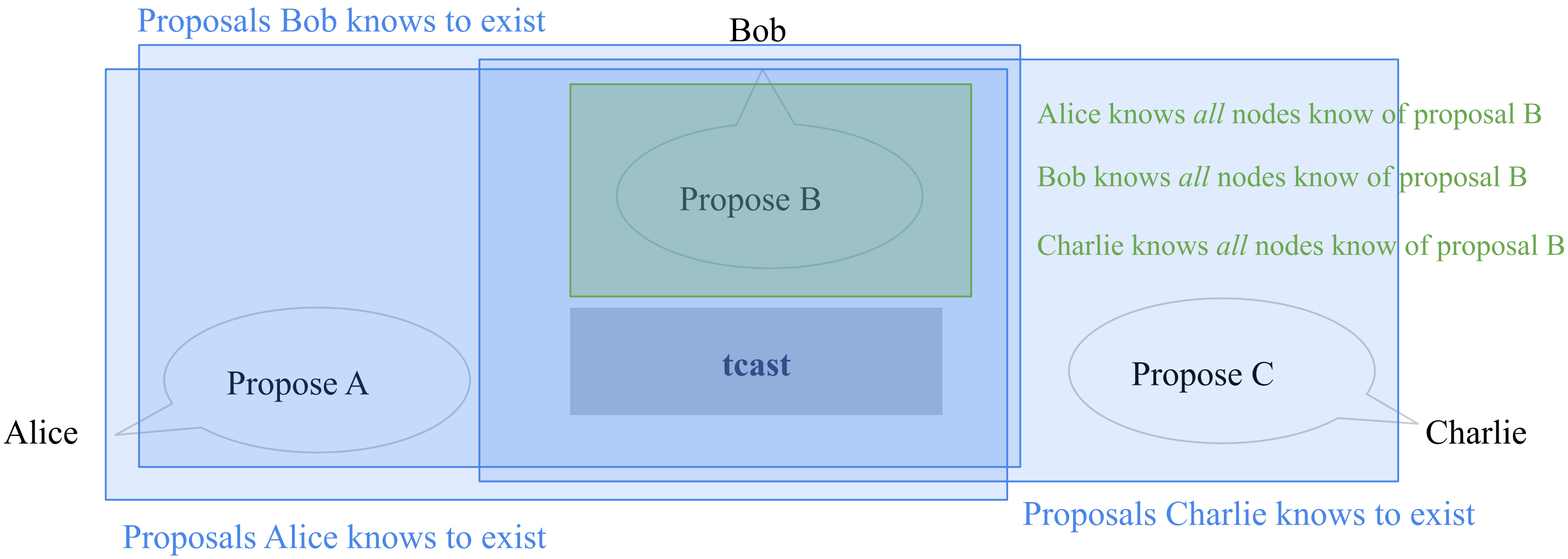
Abstract QuePaxa Algorithm

Algorithm 1: Abstract QuePaxa consensus algorithm

```
Input: v ← value preferred by this replica

repeat                                  // iterate through rounds
    p ← ⟨v, random()⟩                   // prioritized proposal
    (P, _) ← tcast({p})                 // propagate our proposal
    (E, P′) ← tcast(P)                  // propagate existent sets
    (C, U) ← tcast(P′)                  // propagate common sets
    v ← best(C).value                   // next candidate value
    if best(E) = best(U) then           // detect consensus
        deliver(v)                      // deliver decision
```

Abstract QuePaxa is just a few lines of pseudocode!



- **tcast property 1**: each node learns the existence of a majority of proposals
- **tcast property 2**: each node learns *some* proposal that has reached *all* nodes

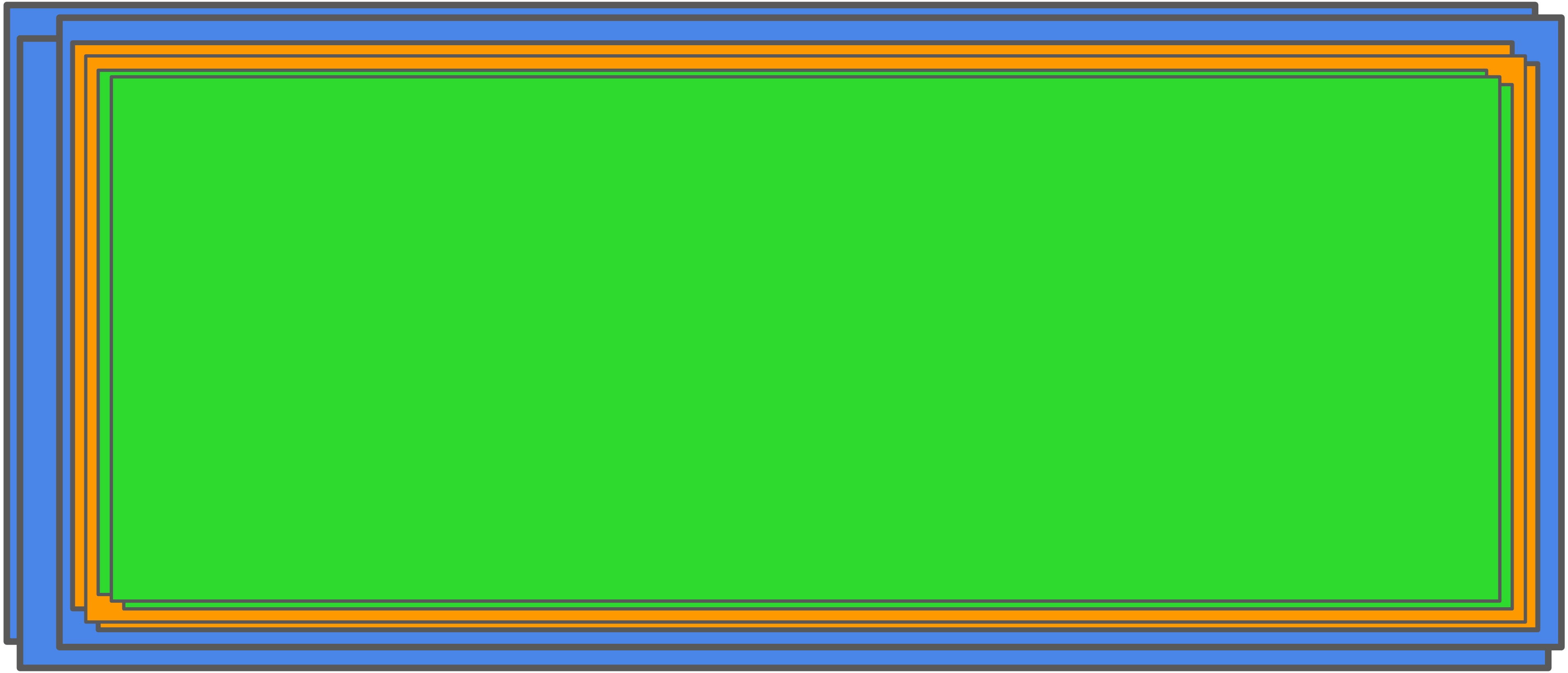
No guarantee that nodes learn the same subsets! (no consensus yet)

Towards consensus: approximating what others know

- Sets from one tcast invocation are **insufficient for consensus**
- **Repeat**: **three tcast invocations**, giving each node *i* sets with increasing guarantees
  - E_i: If Alice knows proposal *P* exists, then *P* is in her ***existent*** set E_i
  - C_i: If Alice knows *all* nodes know *P* exists, *P* is in her ***common*** set C_i
  - U_i: If Alice knows *all* nodes know *P* is common, *P* is in her ***universal*** set U_i

Key relationship for consensus: **for all nodes *i,j,k*, $E_i \supseteq C_j \supseteq U_k$**

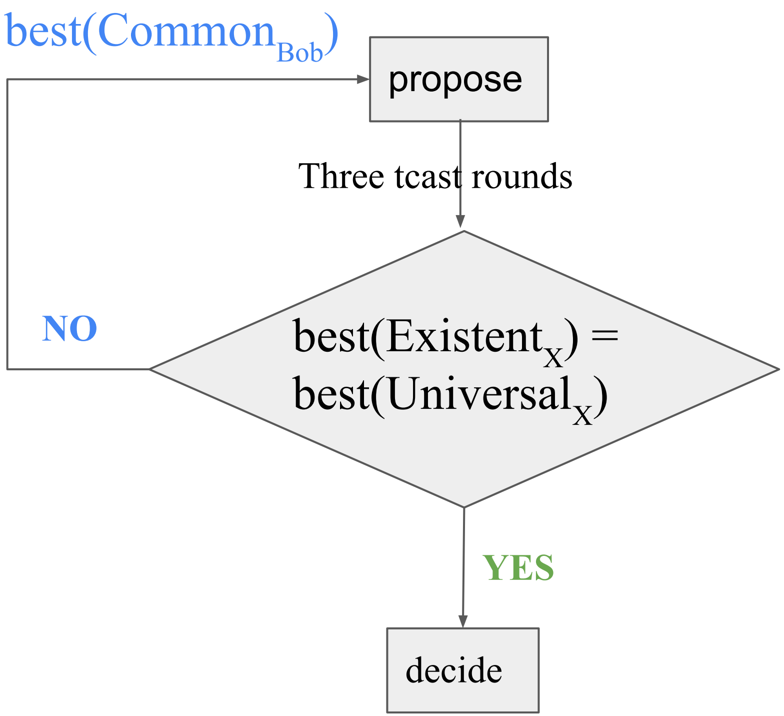
$\text{Existent}_i \supseteq \text{Common}_j \supseteq \text{Universal}_k$

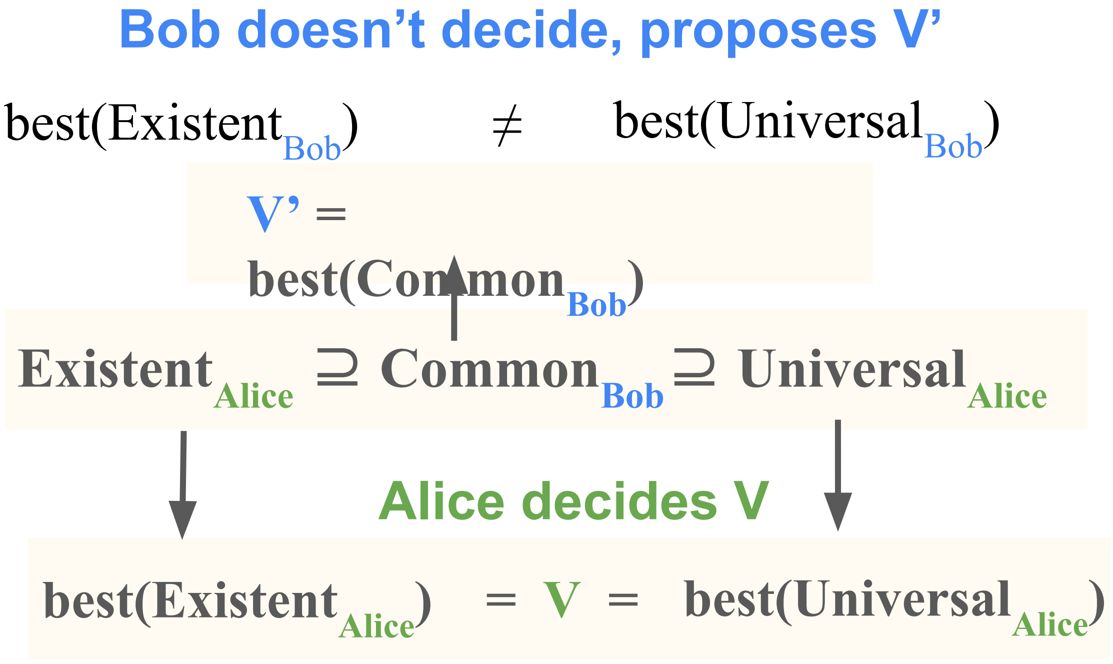


QuePaxa RoadMap

- Operation Overview
- Abstract QuePaxa
- **Safety and liveness of abstract QuePaxa**
- Concrete QuePaxa overview
- The QuePaxa fast path

Consensus: reaching a safe decision

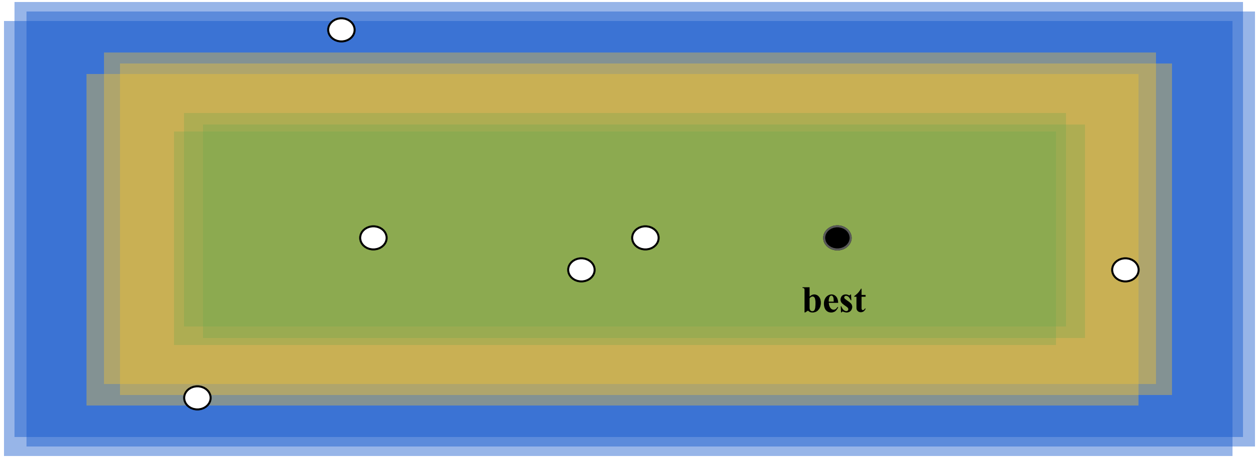




Only possible decision in future is V' = best($Common_{Bob}$) = best($Existent_{Alice}$) = V

Efficiency: How many rounds until consensus

Probability that Alice decides Prob (best($\text{Existent}_{\text{Alice}}$) = best($\text{Universal}_{\text{Alice}}$))



Each set contains
> ½ of proposals

Decision probability is ≥ ½ ⇒ in expectation two rounds until decision

Abstract QuePaxa

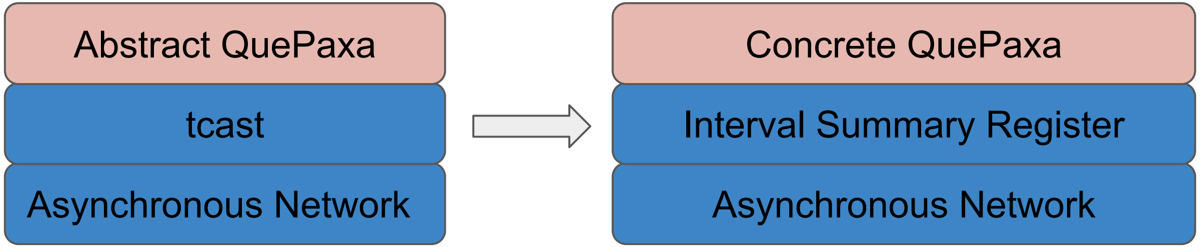
- Avoids timeout from liveness because the protocol is randomized
- Robust against adversarial networks
- $O(n^2)$ message complexity hence slow
- Does not support hedging

Abstract QuePaxa is robust but inefficient

QuePaxa RoadMap

- Operation Overview
- Abstract QuePaxa
- Safety and liveness of abstract QuePaxa
- **Concrete QuePaxa overview**
- The QuePaxa fast path

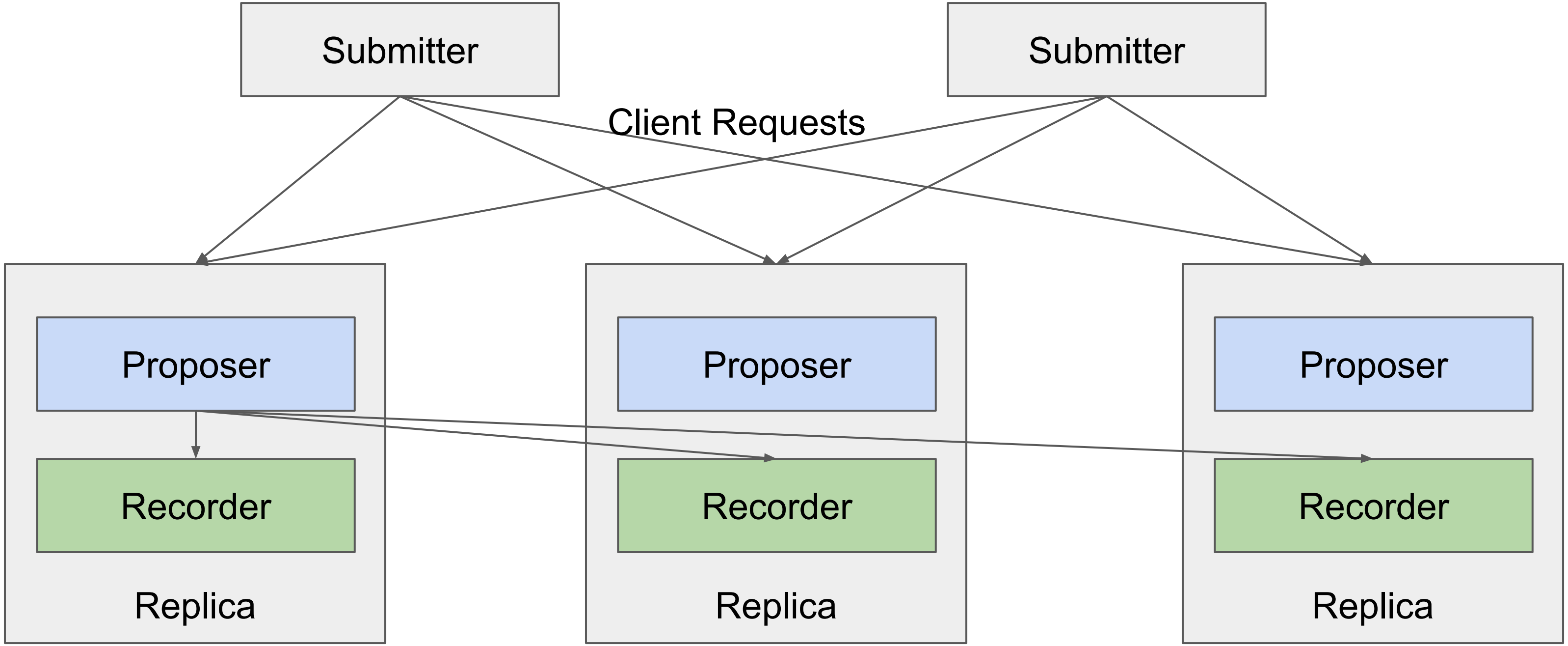
From abstract to concrete QuePaxa



- O(n) complexity in the normal case
- Robust against asynchrony
- Support hedging
- Implementation ready (4368 LOC)

Concrete QuePaxa has all we need!

QuePaxa Architecture



Concrete Recorder Protocol (ISR)

Algorithm 2: Interval summary register (ISR)

State : S current logical clock step, initially 0
State : $F[s]$ first value recorded at each step, default **nil**
State : $A[s]$ aggregate of values in each step, default **nil**

```
record (s, v) → (s′, f′, a′):              // handle an invocation
    if s > S then                          // advance to a higher step
        S ← s                              // update current step number
        F[s] ← v                           // record first value in this step
    if s = S then                          // aggregate all values
        A[s] ← aggregate(A[s], v)          // seen in this step
    return (S, F[S], A[S − 1])             // return a summary
```

- Simulates lock step synchrony using a threshold logical clock
- For each step, records the the first value and the aggregate of the values submitted in the previous step
- Constant space

QuePaxa Recorder is a constant space interval summary register

Proposer Code

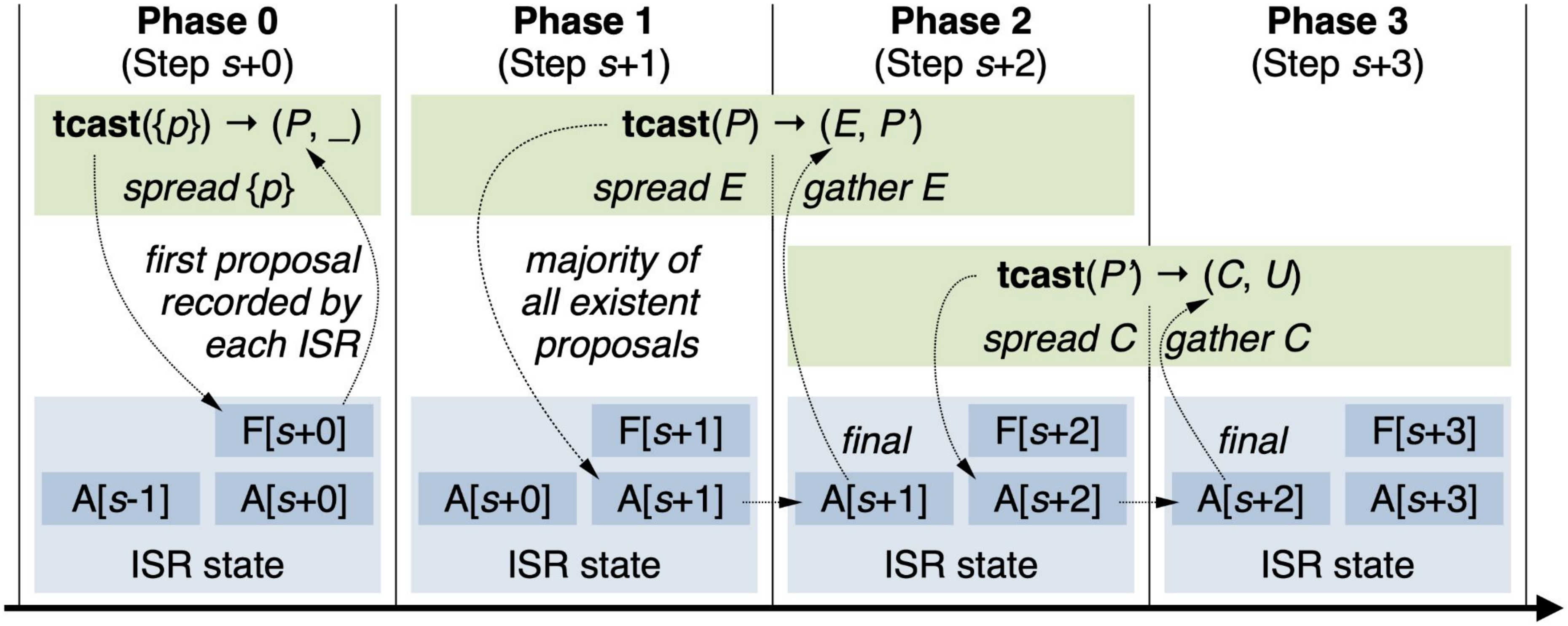
Algorithm 4: Protocol for QuePaxa proposer i

Input: v preferred value of this proposer i

```
s ← 4 × 1 + 0                                    // start at round 1, phase 0
p ← ⟨H, i, v⟩                                    // initial proposal template
repeat
    p_j ← p for all recorders j                  // prepare proposals
    if s mod 4 = 0 and (s > 4 or i is not leader) then
        p_j.priority ← random(1..H − 1) for all j
    Send record(s, p_i) in parallel to each recorder j
    Await R ← quorum of replies (s'_j, f'_j, a'_j)
    if s'_j = s in all replies received in R then
        if s mod 4 = 0 then                      // phase 0: propose
            if f'_j.priority = H in all replies then
                return f'_j.value from any reply in R
            p ← best_j of f'_j from all replies in R
        if s mod 4 = 1 then                      // phase 1: spread E
                                                 // no action required
        if s mod 4 = 2 then                      // phase 2: gather E, spread C
            if p = best_j of a'_j from all replies in R then
                return p.value                   // report decision
        if s mod 4 = 3 then                      // phase 3: gather C
            p ← best_j of a'_j from all replies in R
        s ← s + 1                                // advance to next step
    else if any reply in R has s'_j > s then
        s, p ← s'_j, f'_j                        // catch up to step s'_j
```

QuePaxa proposer uses RPC in 4 phases to contact Recorders

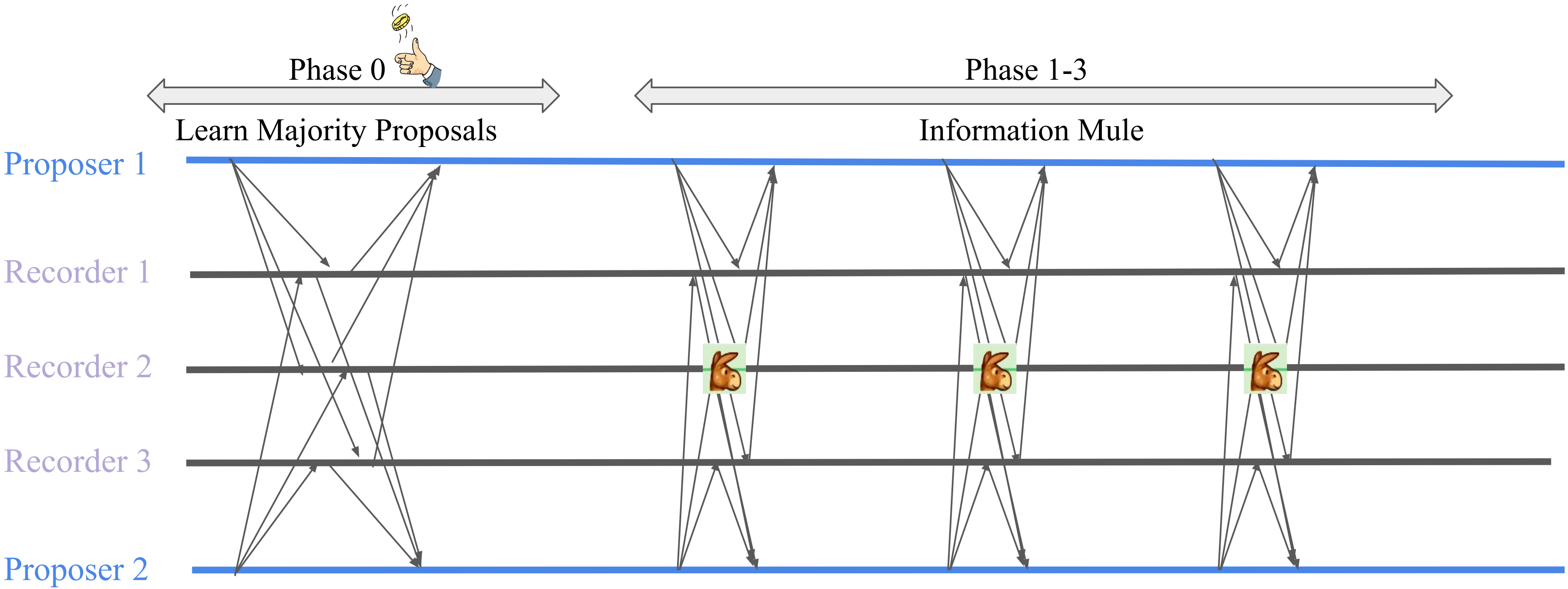
How tcast abstraction maps to concrete QuePaxa phases



QuePaxa RoadMap

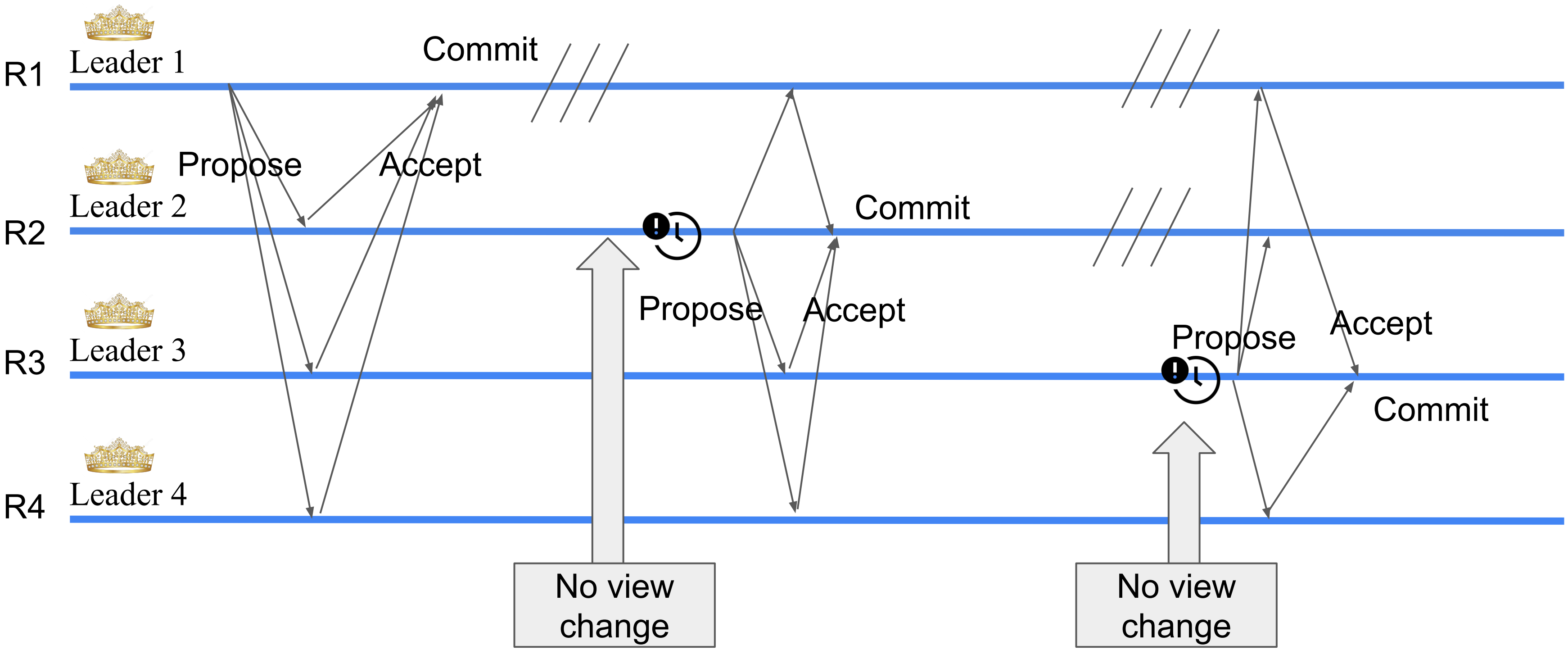
- Operation Overview
- Abstract QuePaxa
- Safety and liveness of abstract QuePaxa
- Concrete QuePaxa overview
- **The QuePaxa fast path**

Hedging in QuePaxa



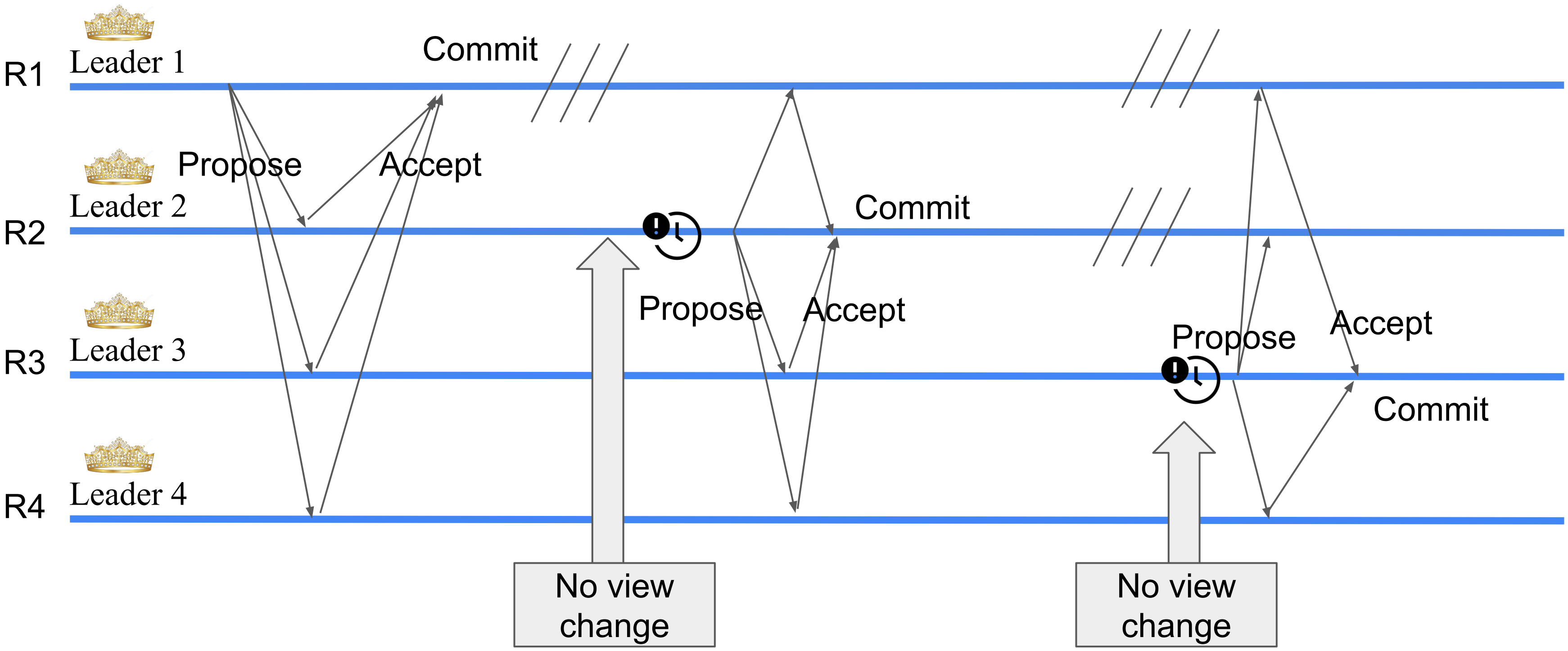
QuePaxa supports hedging because multiple proposers do not cancel each other

What if **multiple** leaders could propose **without** view changes?

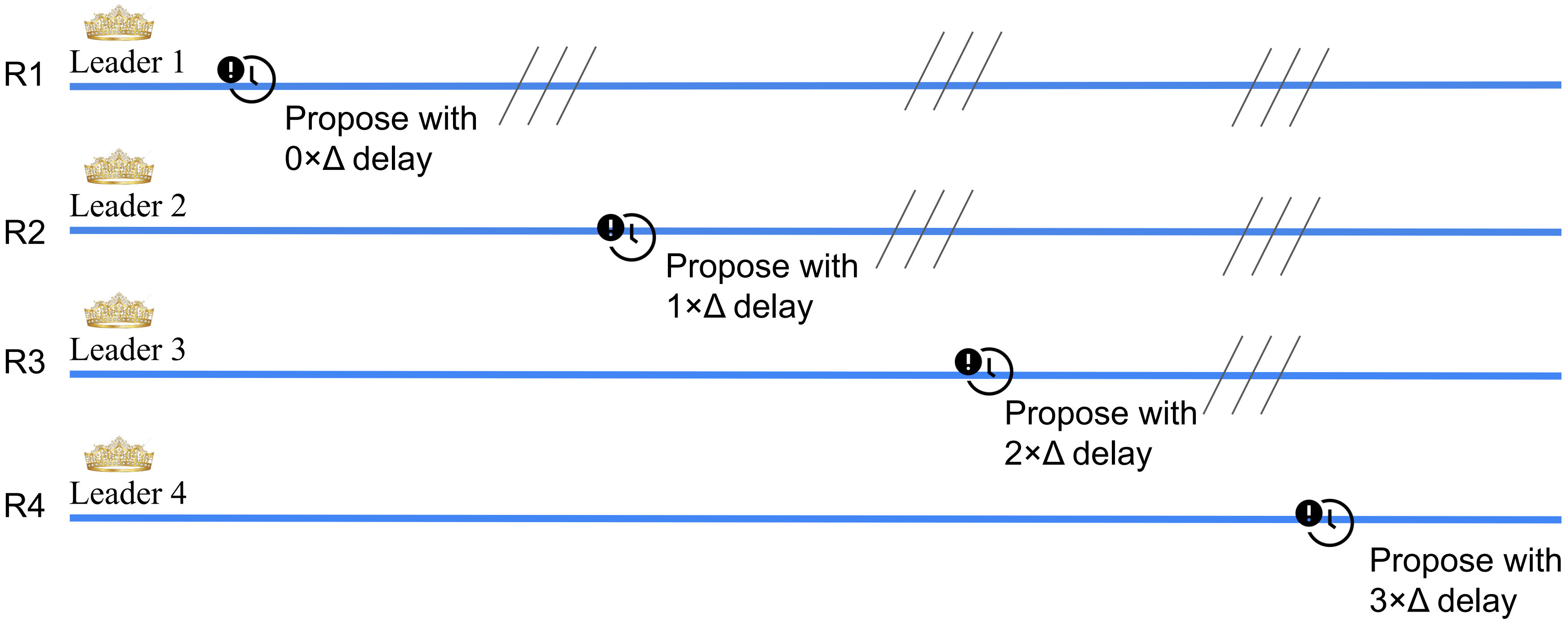


Can we change leaders **without view changes** if the current leader is sub optimal?

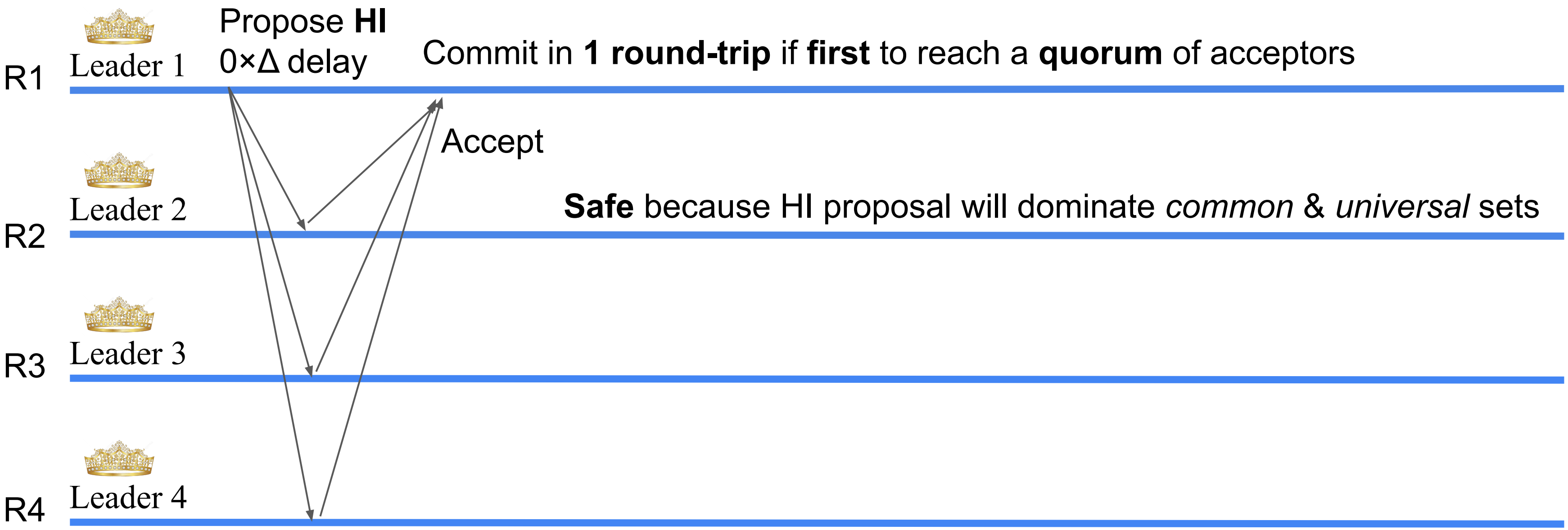
In QuePaxa, multiple leaders **can** propose without view changes



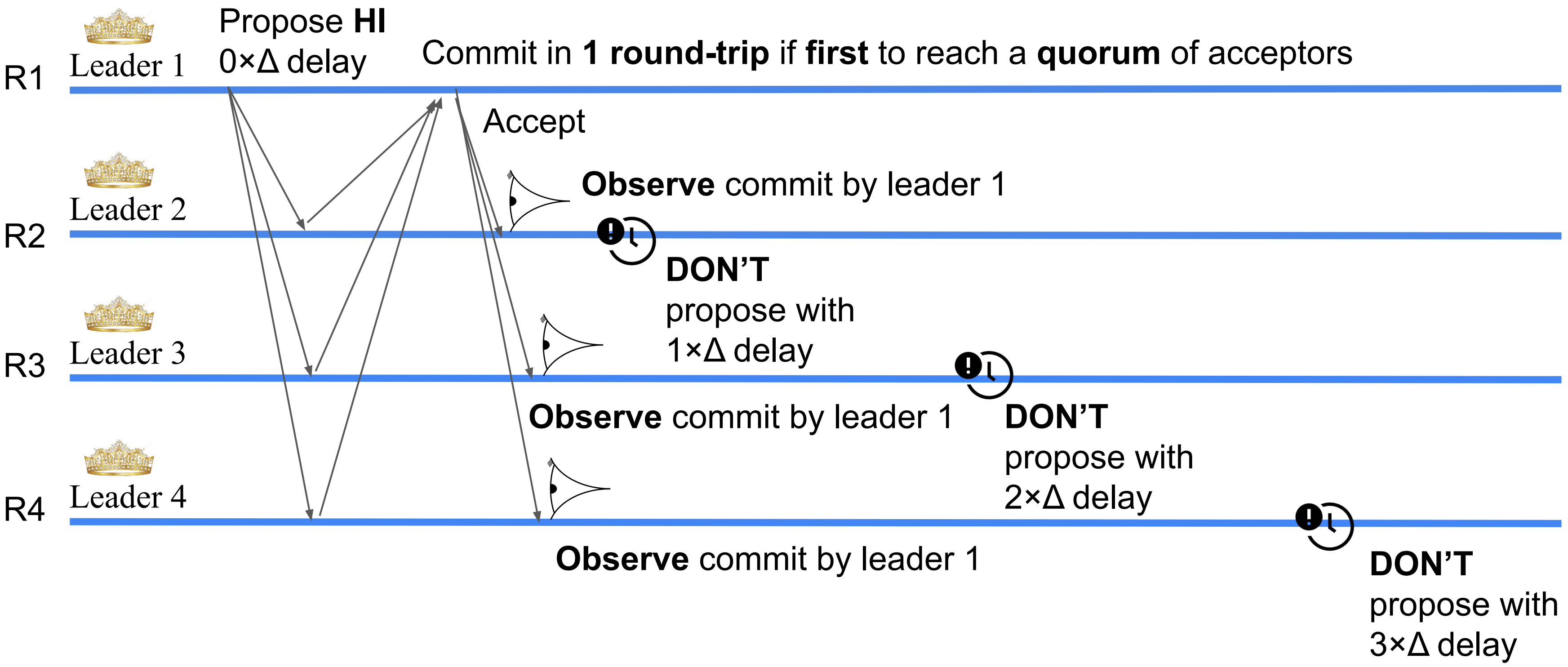
All potential leaders propose on well-known **hedging schedule**



Round 0: first leader proposes with special reserved **HI** priority



First leader's commit **suppresses** remaining leaders' proposals



Normal case: **only** leader 1 proposes → complexity is $O(n)$ instead of $O(n^2)$ per slot

Performance robustness in challenging network situations

What if:

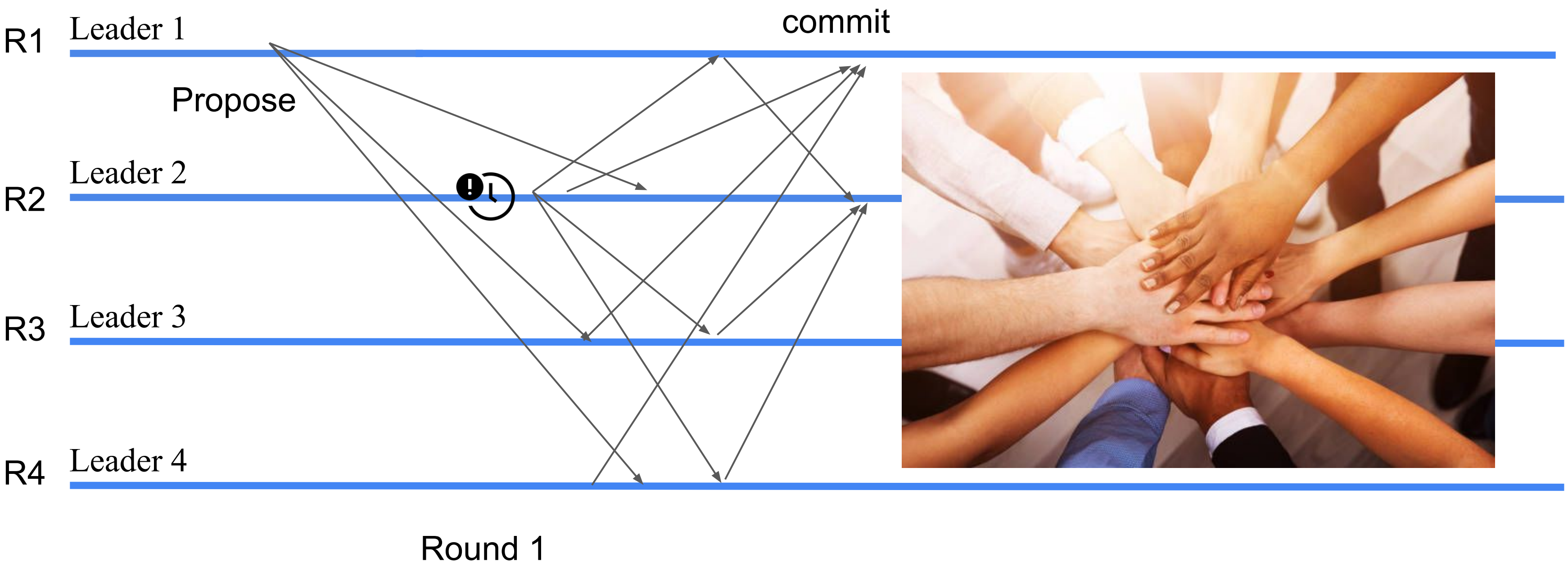
- Network experiences periods of **high delay** (e.g., due to congestion)?
- Network exhibits **high jitter** or **delay unpredictability** (e.g., bursty loads)?
- Timeouts or hedging delays **misconfigured** too low for actual network?

Multi-Paxos/Raft: can **slow drastically** or **lose liveness** entirely

QuePaxa: usually **maintains full performance** even in such situations

- Two or more leaders propose per round, but Leader 1 usually “wins” anyway
- Cost is only **extra unnecessary messaging** (bandwidth use), no extra delay!

Performance robustness in challenging network situations



Leader 2 starts proposing concurrently, but **does not interfere** with Leader 1

Other Contributions

- Multi-Armed-Bandit based hedging sequence tuning for maximum performance
- Optimizations for reducing leader bandwidth bottleneck for high performance

RoadMap

- Introduction to consensus
- Tyranny of timeouts
- Parallels of QuePaxa and hedging
- QuePaxa algorithm
- **Evaluation**

Evaluation

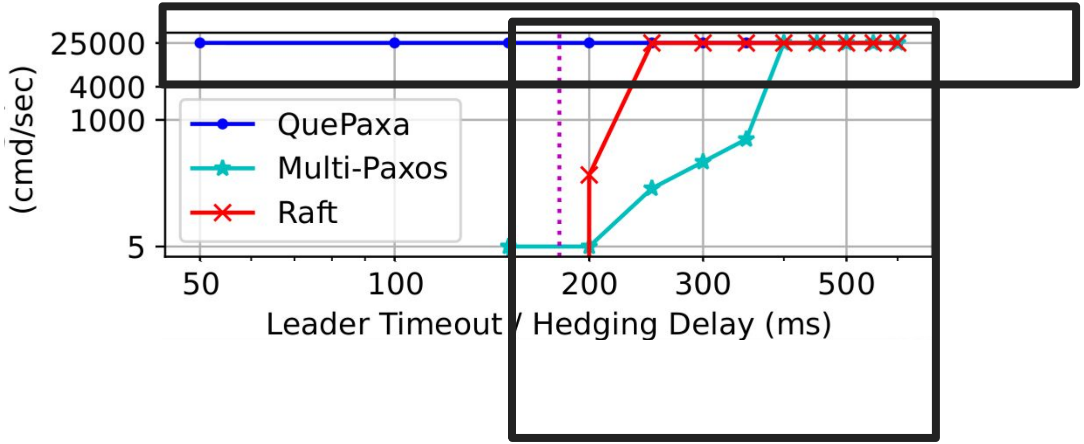
- Can QuePaxa guarantee liveness under any hedging schedule?
- Under normal case, how does QuePaxa compare with leader-based protocols?
- Under adversarial conditions, can QuePaxa maintain liveness?
- Can QuePaxa converge to the best hedging schedule? – *please refer the paper*

Setup

- LAN (N. Virginia)
- WAN (Tokyo, Mumbai, Singapore, Ireland, and São Paulo)
- Replicas: c4.4xlarge
  - 16 virtual CPUs, 30 GB memory
- Submitters: c4.2xlarge
  - 8 virtual CPUs, 15 GB memory

Effect of Hedging in Quepaxa

Throughput

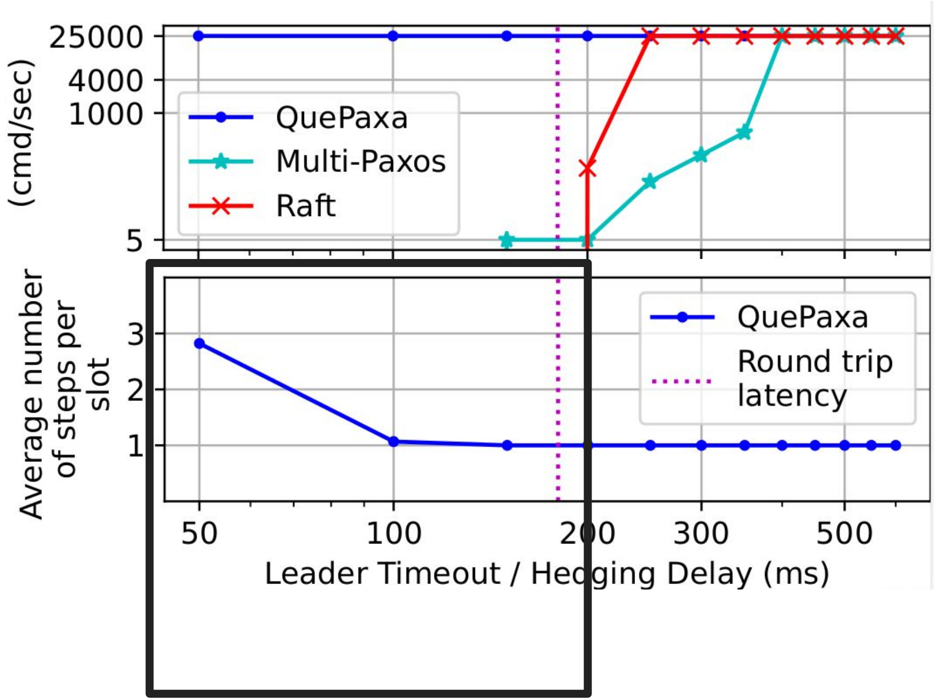


QuePaxa is live for any hedging delay

Effect of Hedging in Quepaxa

Throughput

Bandwidth Cost



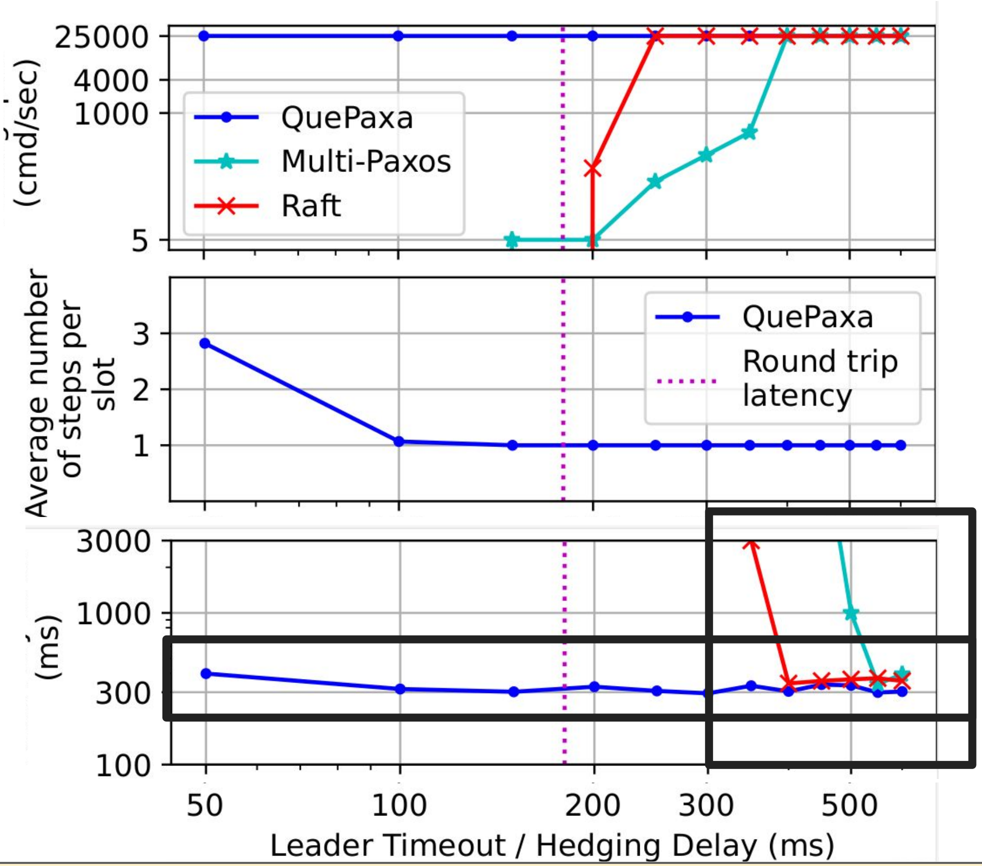
QuePaxa has an additional overhead only when hedging delay < RTT

Effect of Hedging in Quepaxa

Throughput

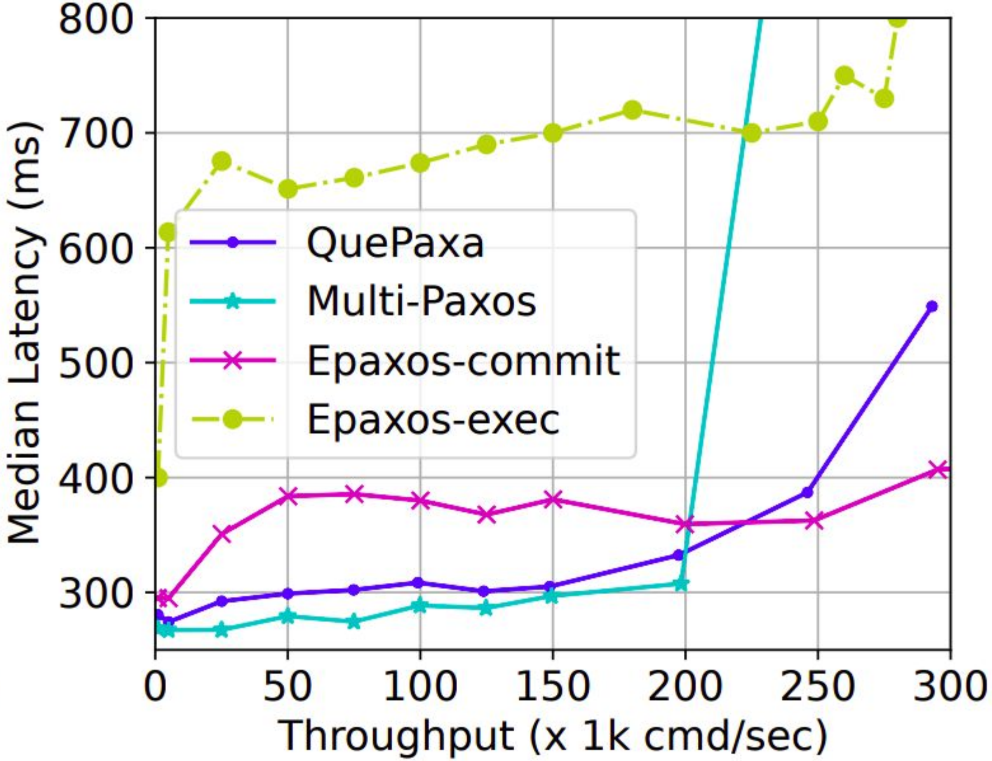
Bandwidth Cost

Recovery Time



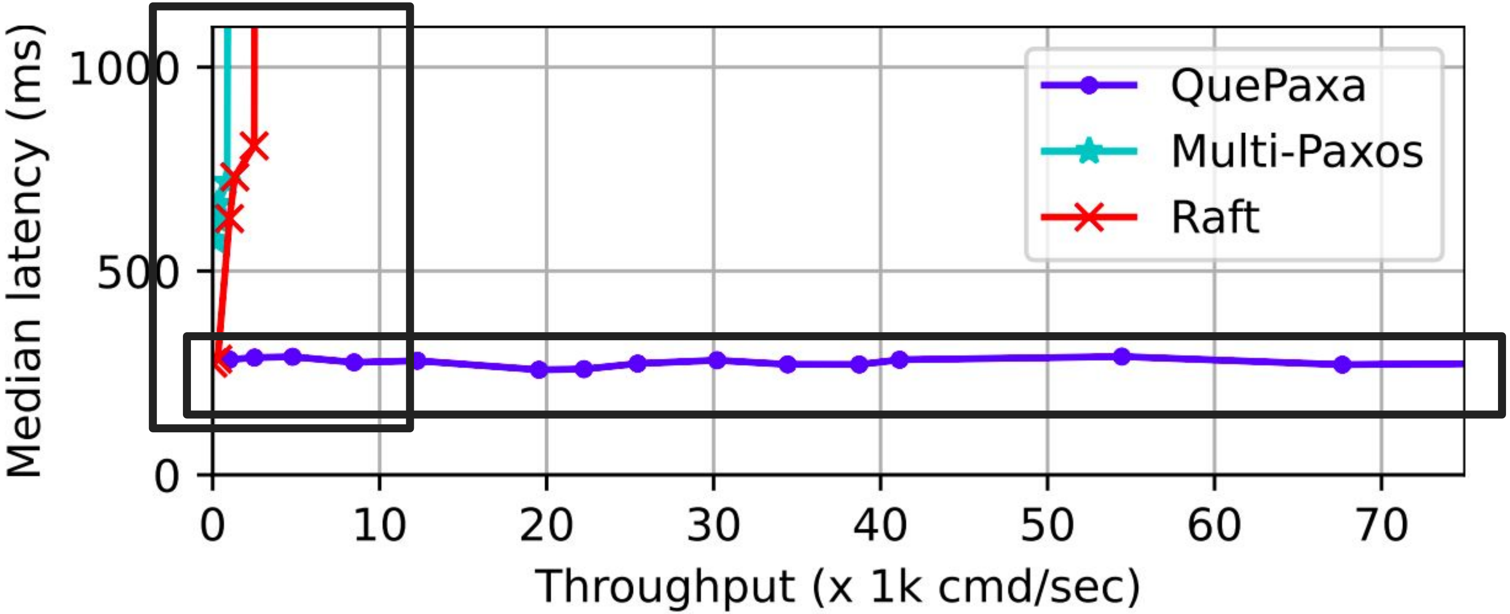
QuePaxa has low recovery time

Normal case execution in a WAN (see paper for LAN)



QuePaxa performs comparable to Multi Paxos

Performance under adversarial networks



QuePaxa is live under asynchrony

Conclusion

- QuePaxa eliminates timeout from liveness guarantees and supports hedging
- QuePaxa provides Multi-Paxos / Raft equivalent performance under normal case
- QuePaxa is performance robust and resilient to adversarial network conditions
- https://github.com/dedis/quepaxa

Supplementary

Hedging delay vs Timeout

- Timeouts initiate failure-recovery processes that interfere with normal progress if triggered early
  - a premature Raft view change halts the prior leader's progress.
- Hedging initiates non-destructive concurrency:
  - launching a second QuePaxa proposer does not prevent the first from still completing the round.
- QuePaxa hedging delays can be zero without losing liveness
  - but the cost is redundant messaging

tCast vs other Broadcast flavours

- Best effort broadcast: If a correct process broadcasts a message m, then every correct process eventually delivers m.
- Reliable broadcast: : If a message m is delivered by some correct process, then m is eventually delivered by every correct process.
- Uniform reliable broadcast: If a message m is delivered by some process (whether correct or faulty), then m is eventually delivered by every correct process.
- Byzantine consistent broadcast: delivered m is the same for all receivers.
- Byzantine reliable broadcast: all correct parties deliver some request or none delivers any (Bracha’s broadcast)

tCast

- tcast property 1: each node learns a majority of proposals
- tcast property 2: each node learns a proposal that all nodes know to exist

Que Sera Consensus:
Simple Asynchronous Agreement with
Private Coins and Threshold Logical Clocks

Bryan Ford[1], Philipp Jovanovic[2], and Ewa Syta[3]

[1]Swiss Federal Institute of Technology Lausanne (EPFL)
[2]University College London (UCL)
[3]Trinity College Hartford

QuePaxa vs Common Core

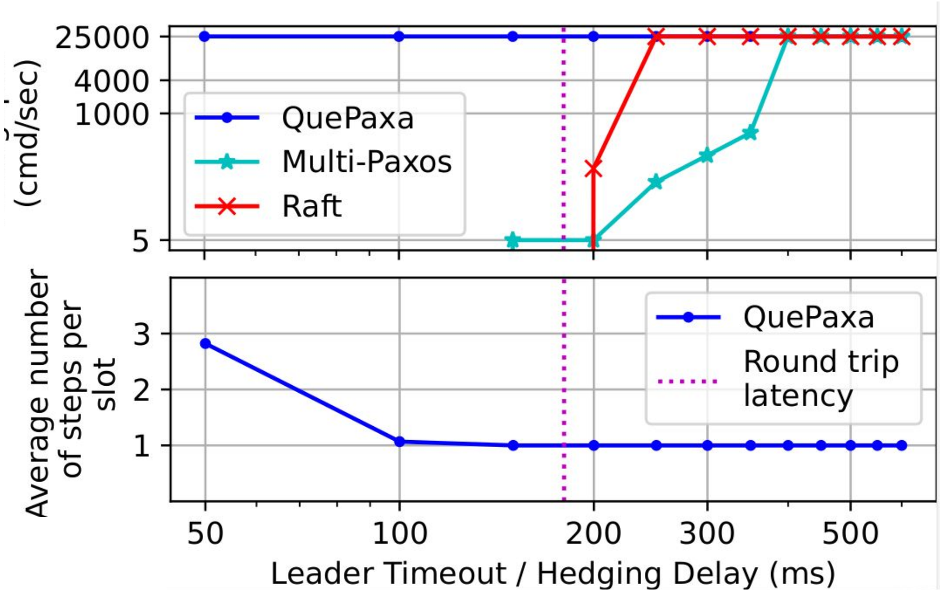
- Common core allows all replicas to create a common core (n-f proposals), such that each node knows that there are n-f proposals known by everyone, however, no node exactly knows which n-f proposals are common. In the literature, common core is used in binary consensus.

- In contrast, tcast-based QuePaxa allows nodes to not only create a common core but also pinpoint which n-f proposals are common. Nodes reach multi-valued consensus using the set relationship we mentioned.

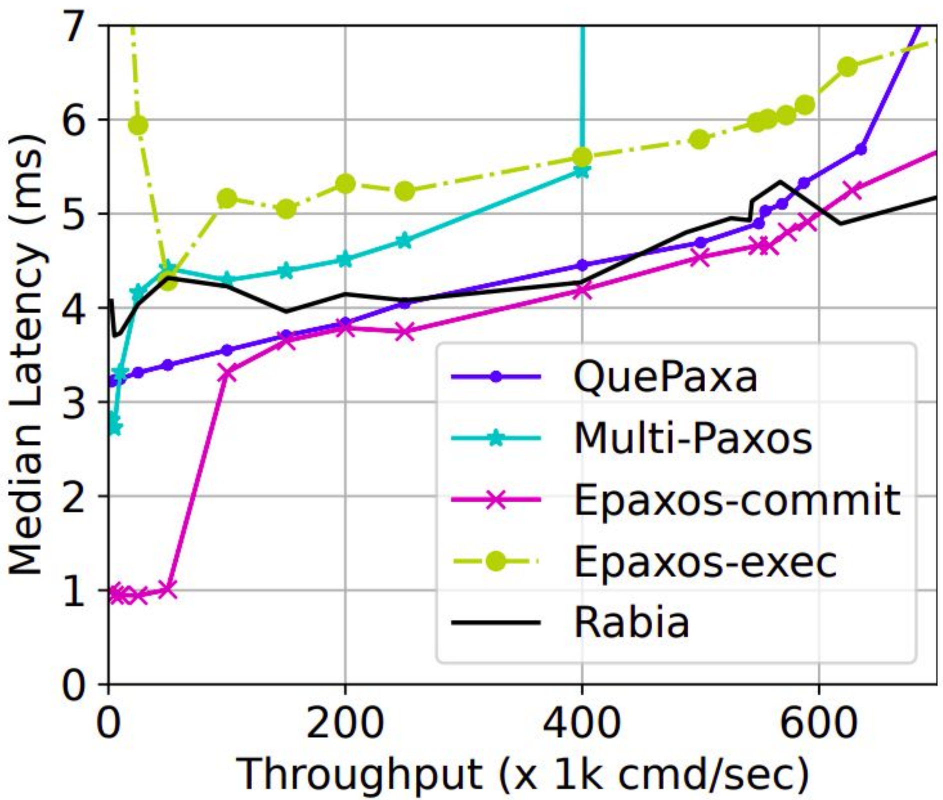
Overhead of Multiple Proposers

Throughput

Bandwidth Cost



Normal Case LAN performance



FLP impossibility and QuePaxa

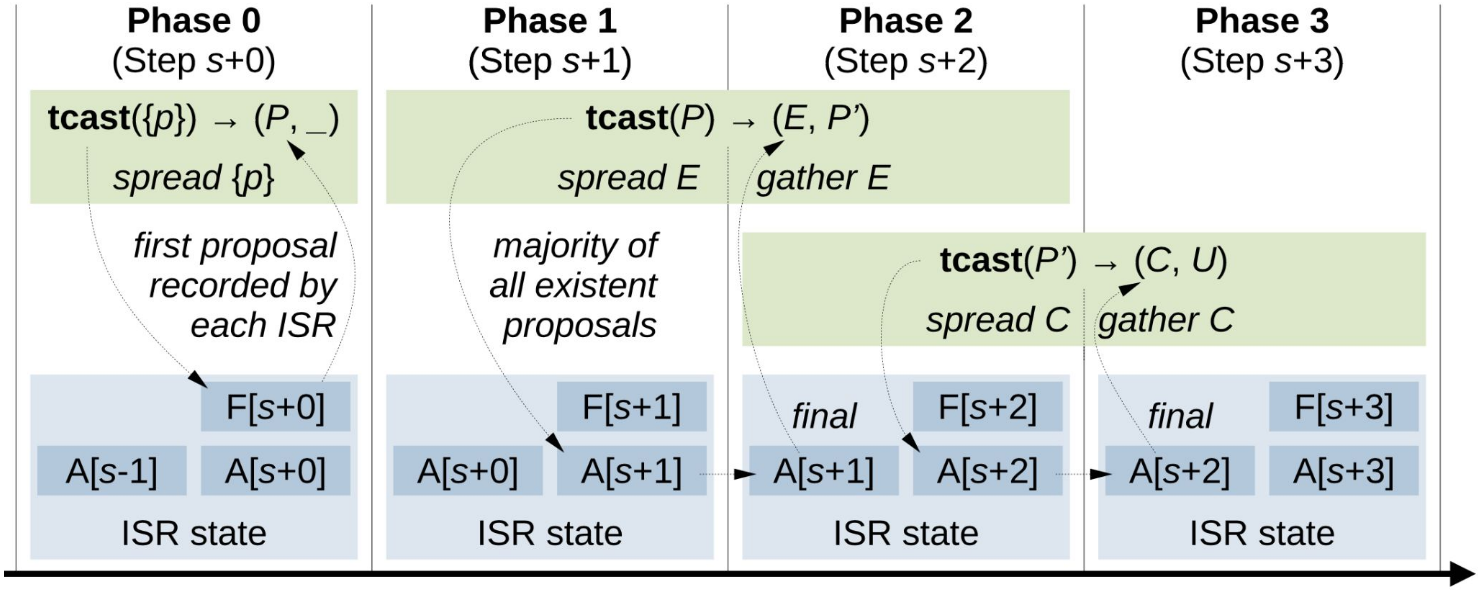
- QuePaxa uses randomization to alleviate FLP
    - However, when the network is synchronous, QuePaxa uses that to provide 1 round trip fast path
- QuePaxa uses private randomness, and that enables hedging

Fast path of 1 RTT in concrete QuePaxa

- How does concrete quePaxa reduce the fast path to just 1 RTT, given that one tcast is several round trips, and one abstract QuePaxa is two tcasts?

- The first tcast of abstract QuePaxa corresponds to a spread phase in concrete QuePaxa in 1 RTT: Each proposer records its proposal at a recorder. In contrast to abstract QuePaxa, however, in concrete QuePaxa only a few nodes propose. If the leader is the fastest, i.e., faster than the few other proposers, then its proposal gets adopted by most recorders. Upon observing this, no other decision is possible and nodes decide after the spread phase, i.e., in 1 RTT.

Correspondence between concrete and abstract QuePaxa (1)



Correspondence between concrete and abstract QuePaxa (2)

- **Concrete QuePaxa phase 0**
    - Computes p = best(P); in abstract QuePaxa P is the output set of the first tcast
- **Concrete QuePaxa phases 1 and 2**
    - Computes a = best(E); in abstract QuePaxa E is the first output of the second tcast
    - Computes p = best(P'), in abstract QuePaxa E P' is the second output set of the second tcast
- **Concrete QuePaxa phases 2 and 3**
    - Computes a = best(C); in abstract QuePaxa C is the first output of the third tcast
    - Computes p = best(U); in abstract QuePaxa U is the second output set of the third tcast