

Decentralized Systems Engineering

CS-438 – Fall 2025

DEDIS

Bryan Ford and Pierluca Borsò-Tan

EPFL

Credits: B. Ford

Advanced Consensus Protocols

QuePaxa: Escaping the Tyranny of Timeouts in Consensus

Pasindu Tennage
EPFL

Cristina Basescu
EPFL & Digital Asset

Eleftherios Kokoris Kogias
ISTA & Mysten Labs

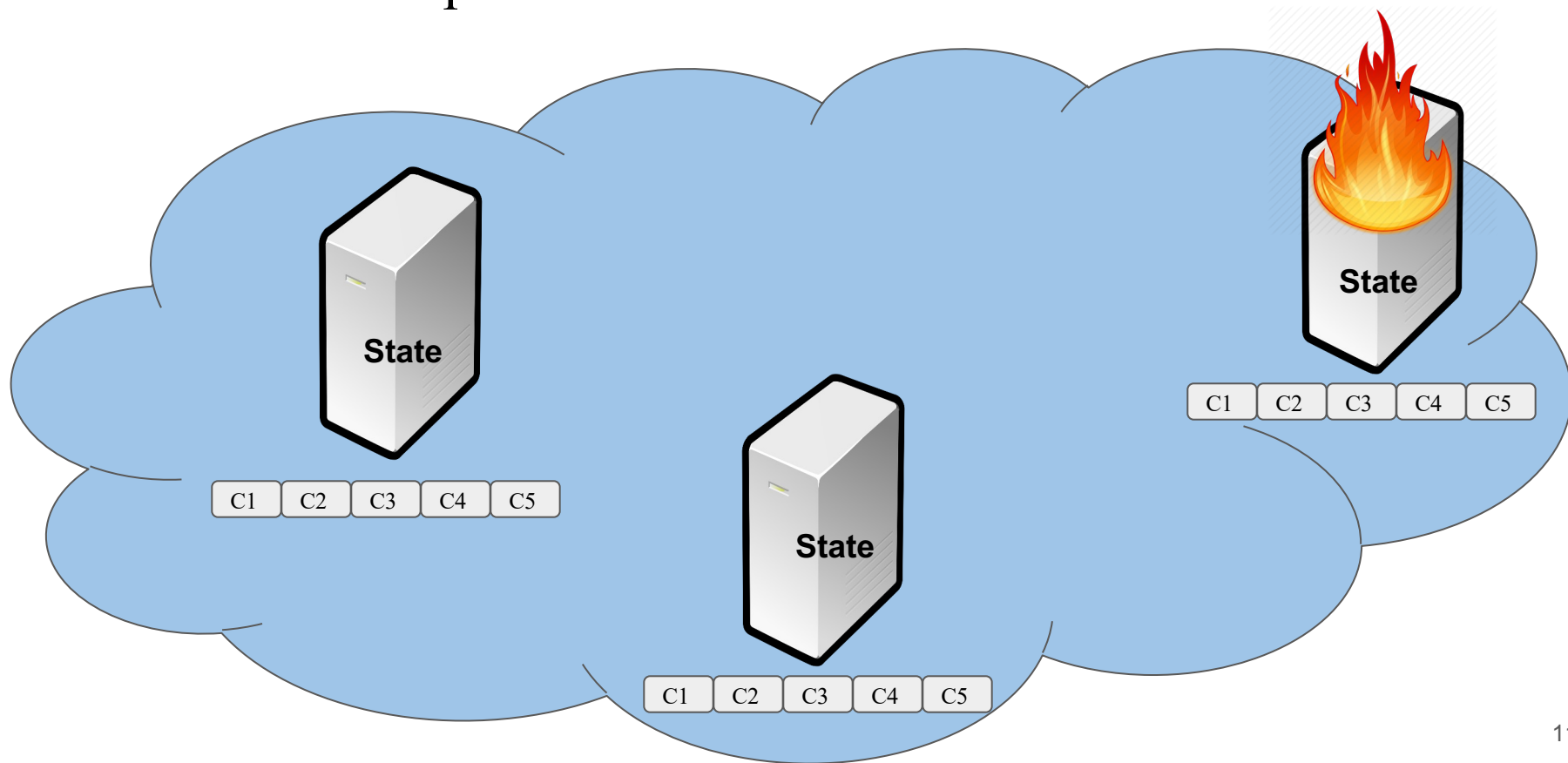
Ewa Syta
Trinity College

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

Semi-Decentralized Geo-replicated State Machines

Quanlu Zhang
Microsoft 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

WPaxos 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



NebulaGraph



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

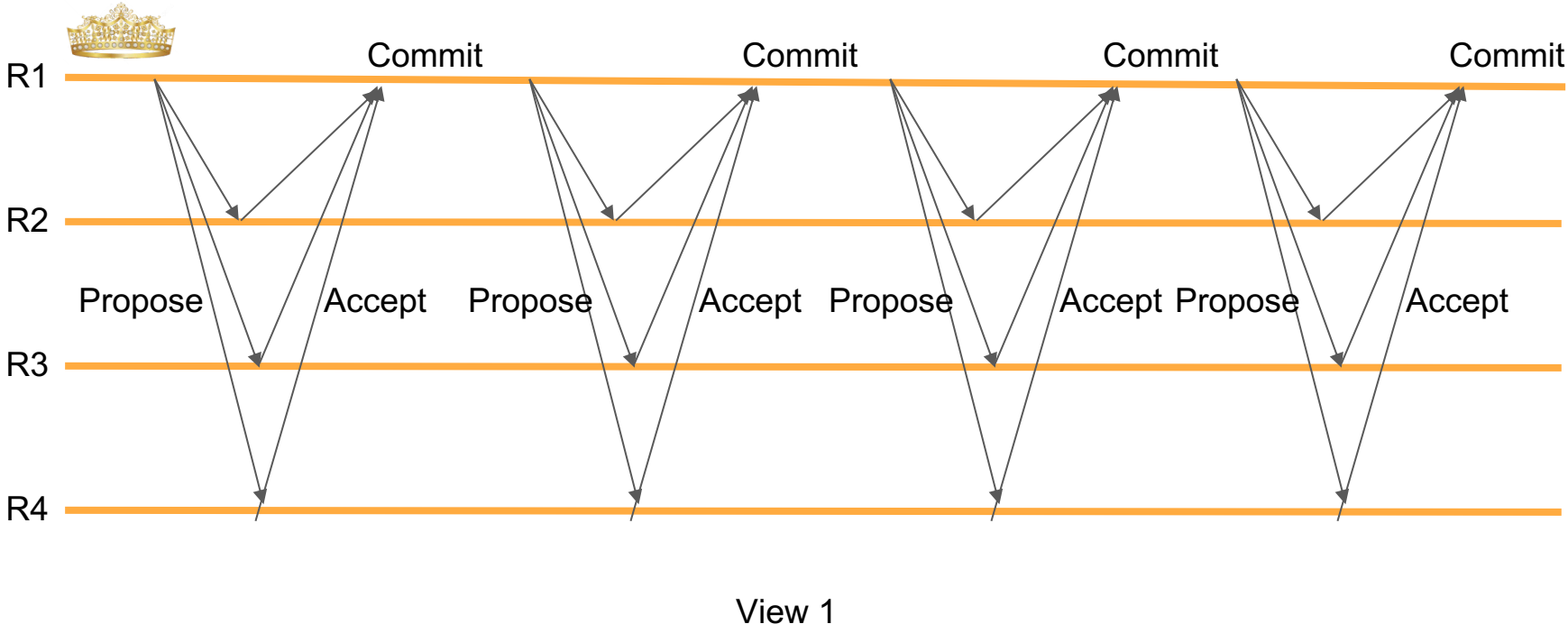
Tyranny of Timeout Problems in Consensus

Timeout based view change

Conservative timeouts

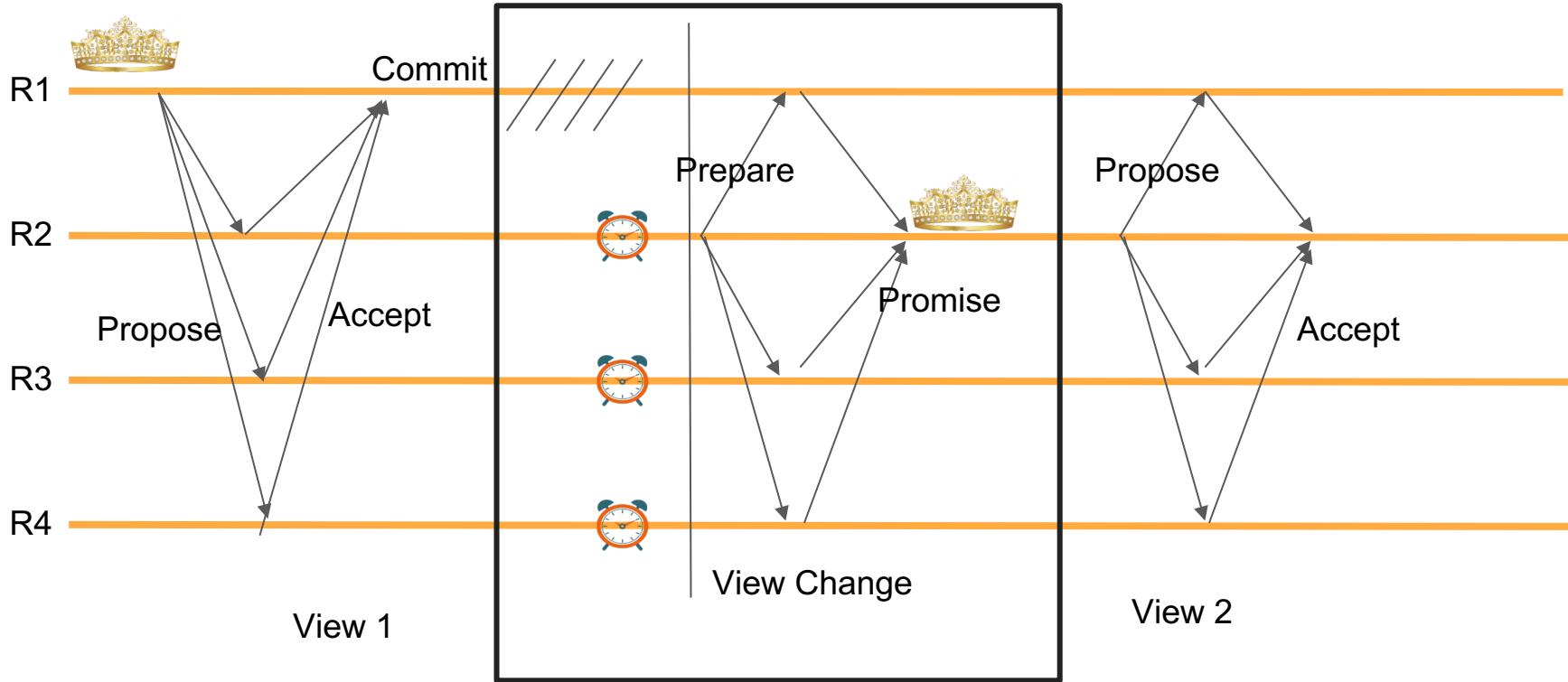
Manually configured timeouts

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

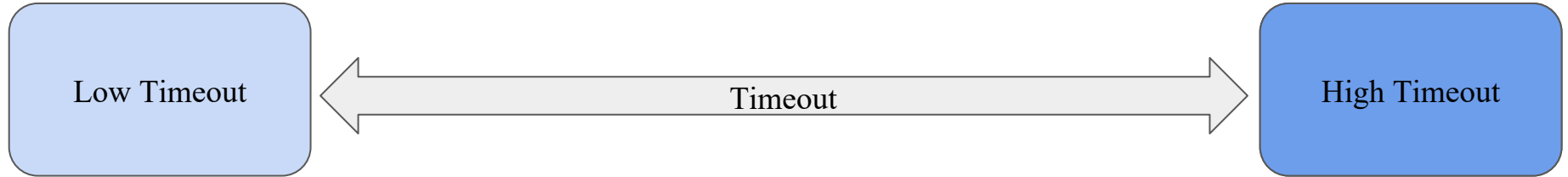
Tyranny of Timeout Problems in Consensus

Timeout based view change

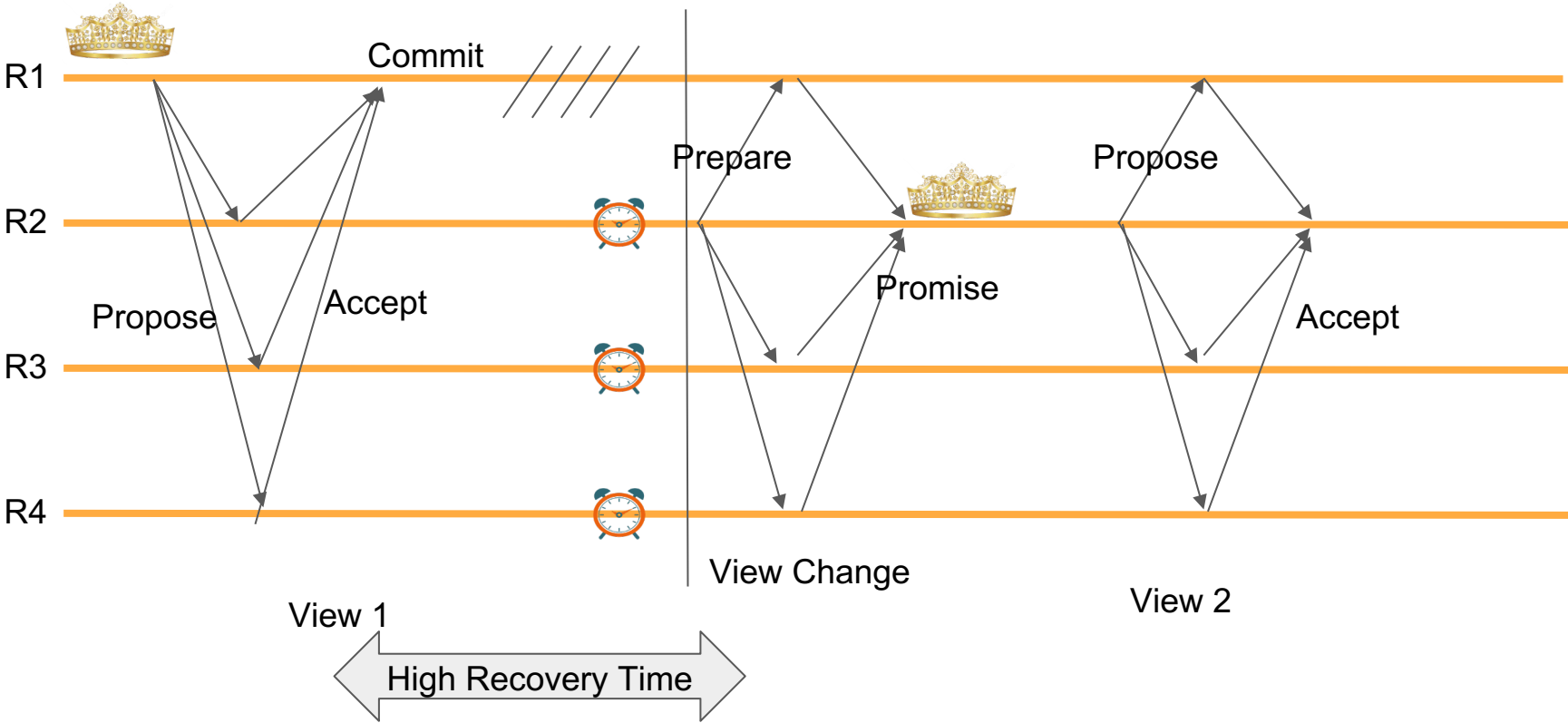
Conservative timeouts

Manually configured timeouts

Choosing Timeouts in leader based protocols

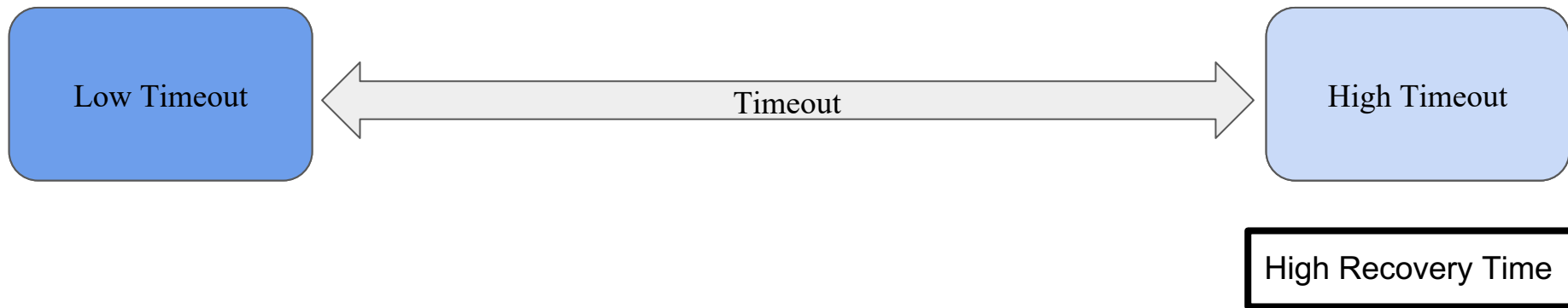


Timeout based view change [Multi-Paxos]

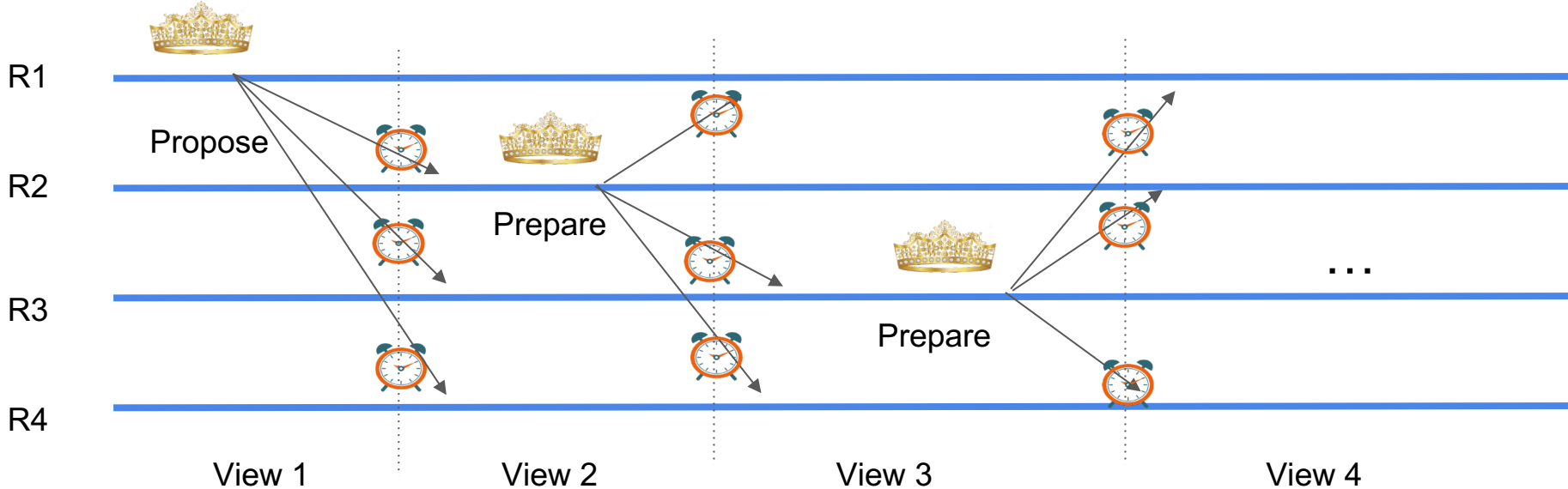


High timeouts result in high recovery time

Choosing Timeouts in leader based protocols

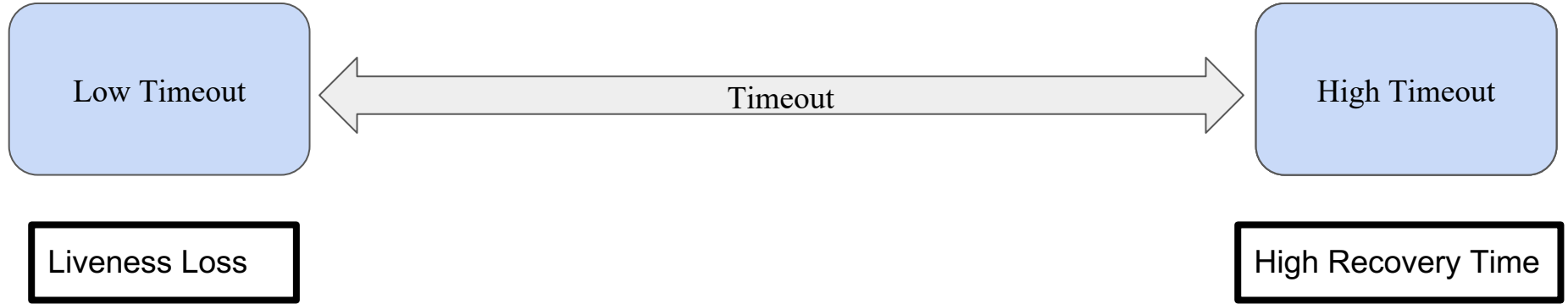


Liveness loss with low timeouts



No commands are committed when the timeout is low

Choosing Timeouts in leader based protocols



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

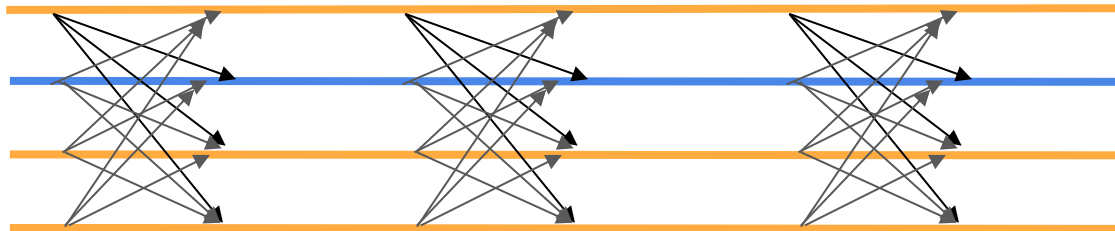
Manual timeouts are sub optimal

Are timeouts necessary for progress?

Can we eliminate the impact of timeout for liveness?

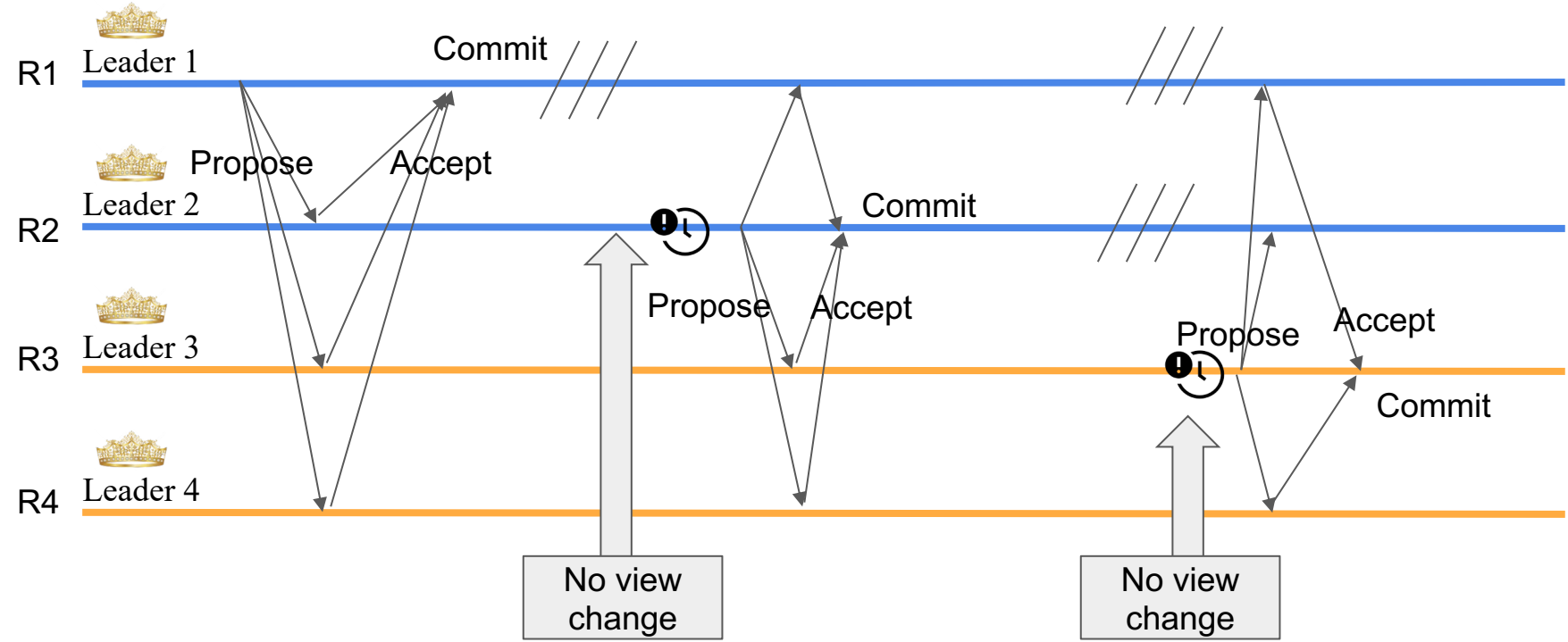
Do asynchronous protocols solve this problem?

- 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



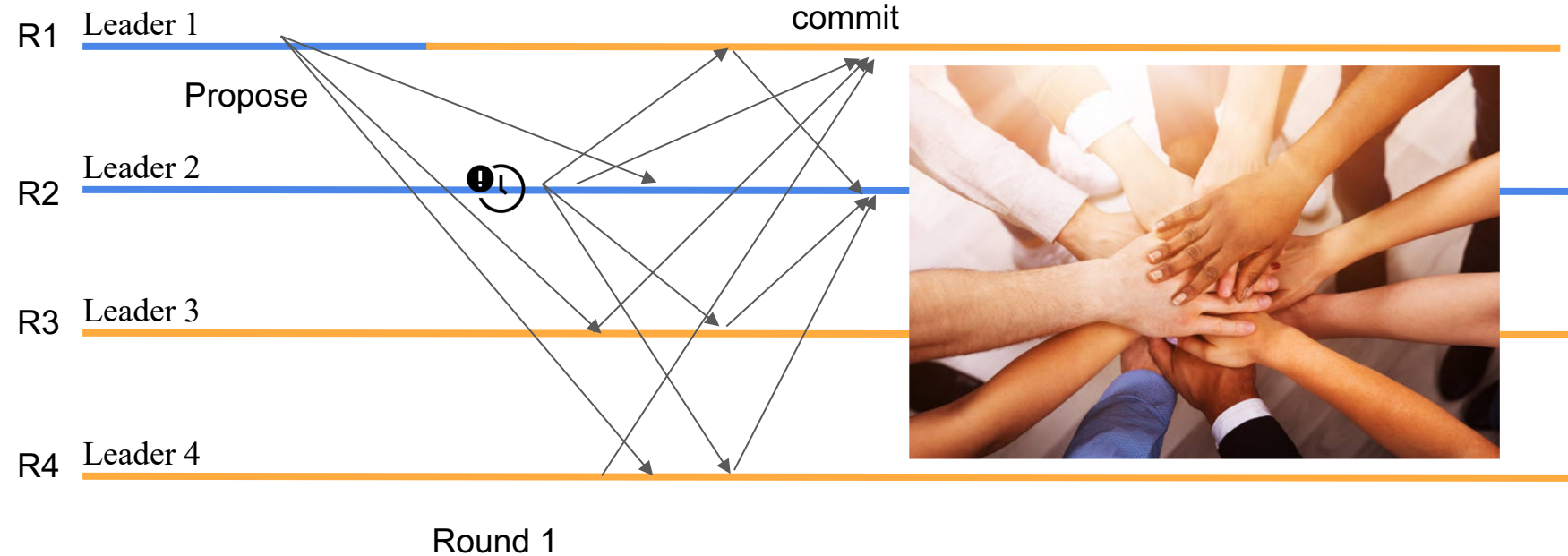
Asynchronous protocols are slow and rarely deployed

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?

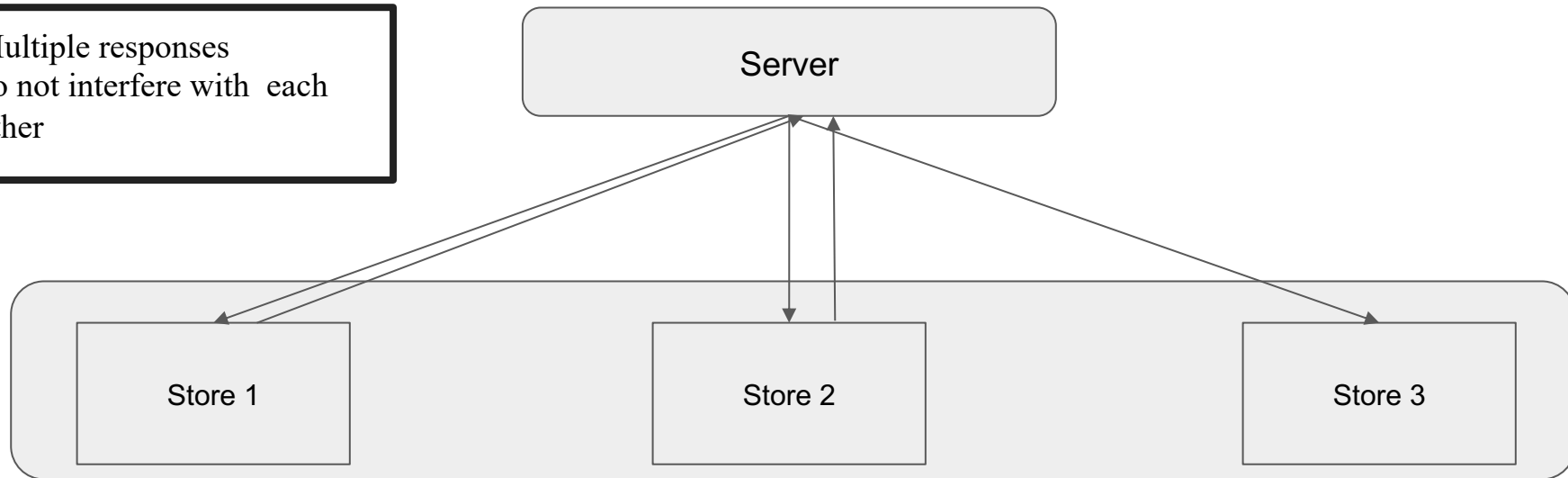
RoadMap

- 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?³³

RoadMap

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

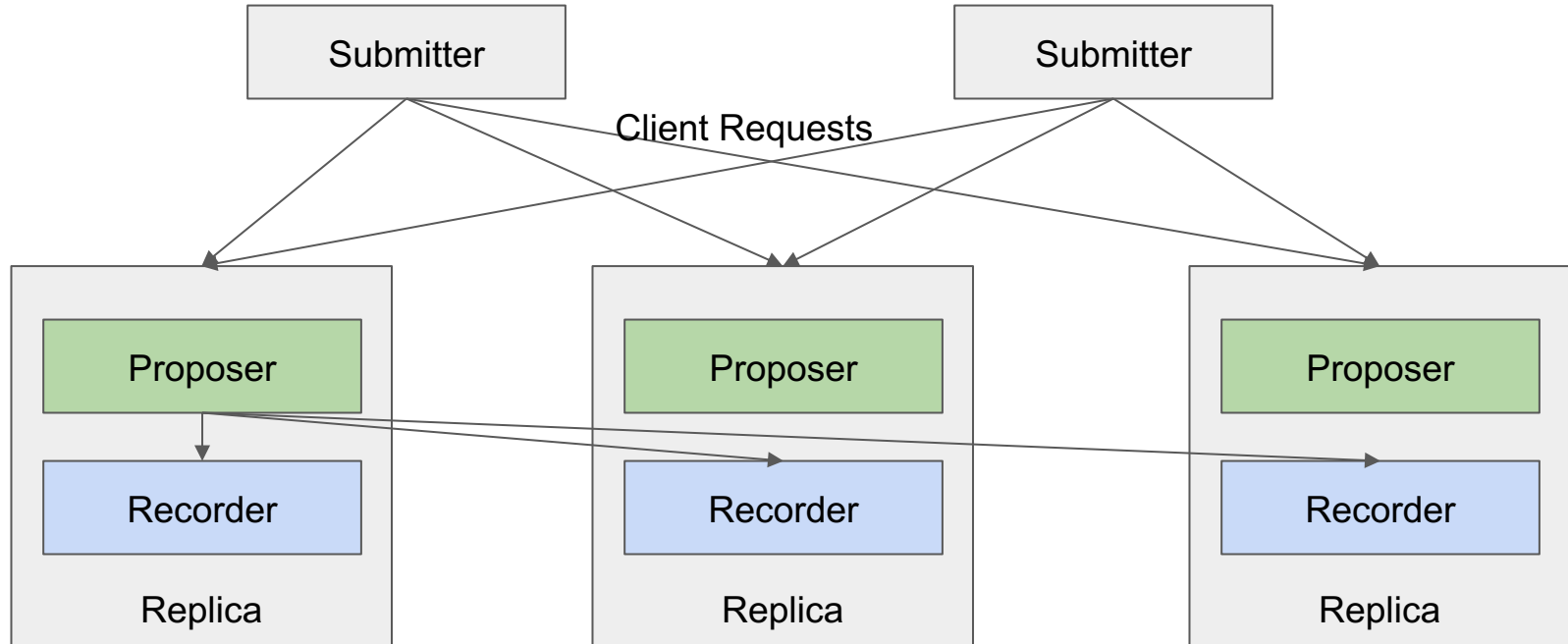
QuePaxa Contributions

- 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

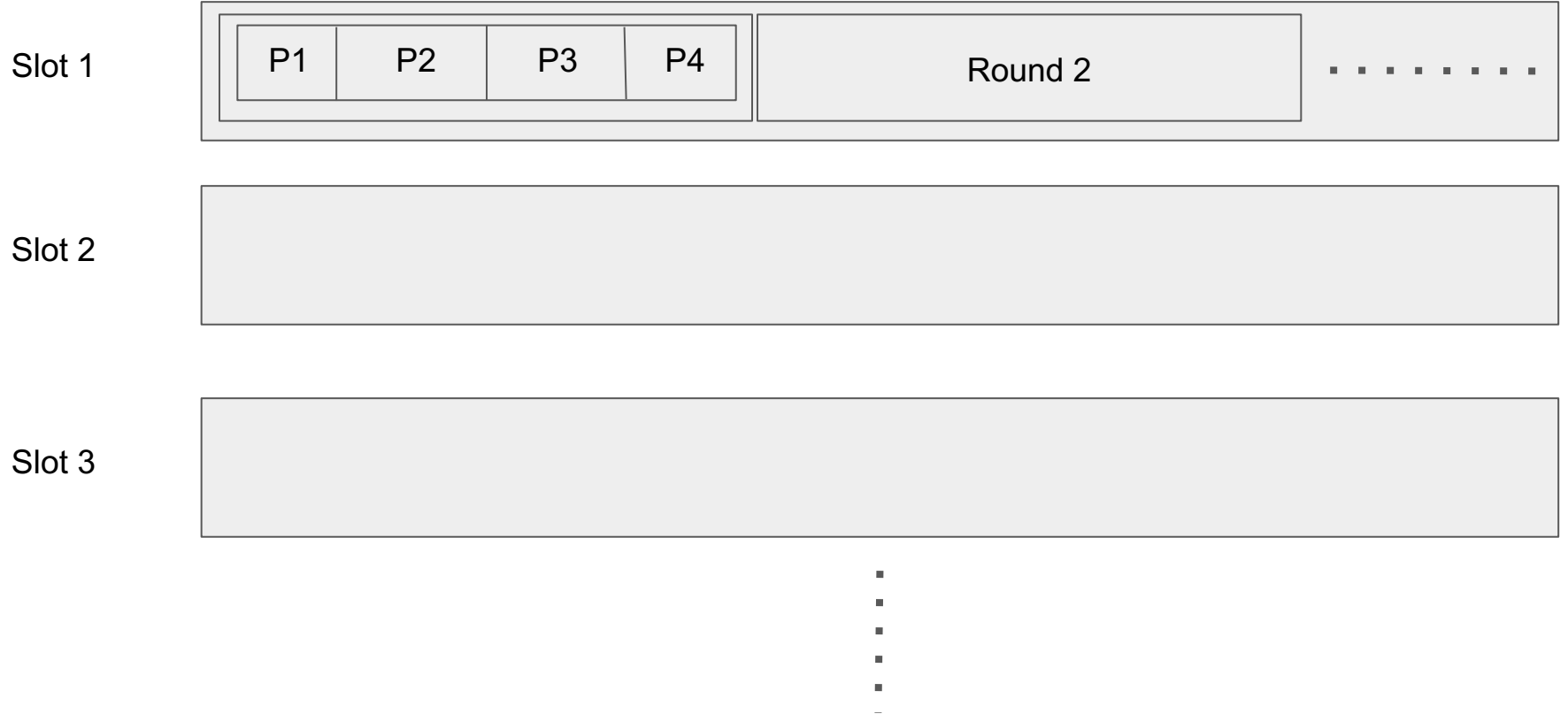
QuePaxa RoadMap

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

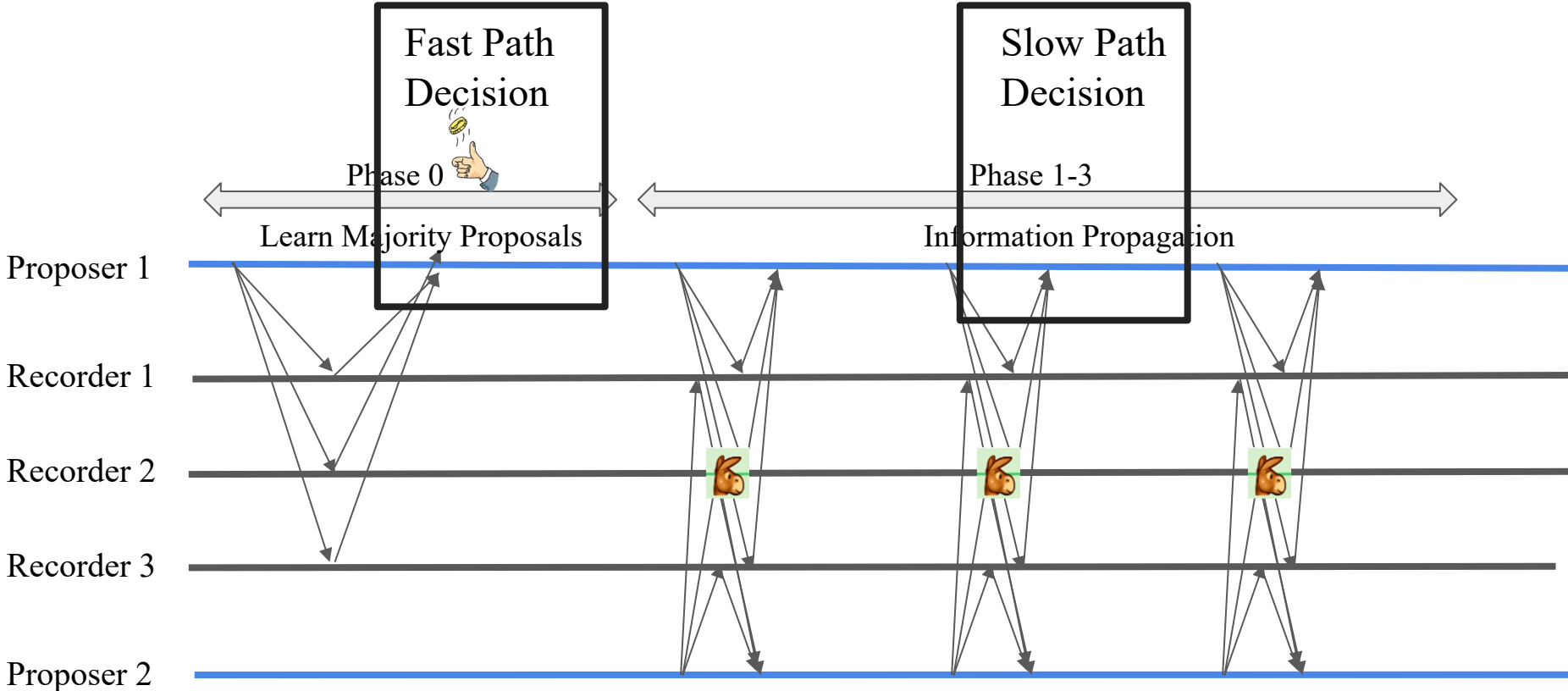
QuePaxa Architecture



QuePaxa Log Structure

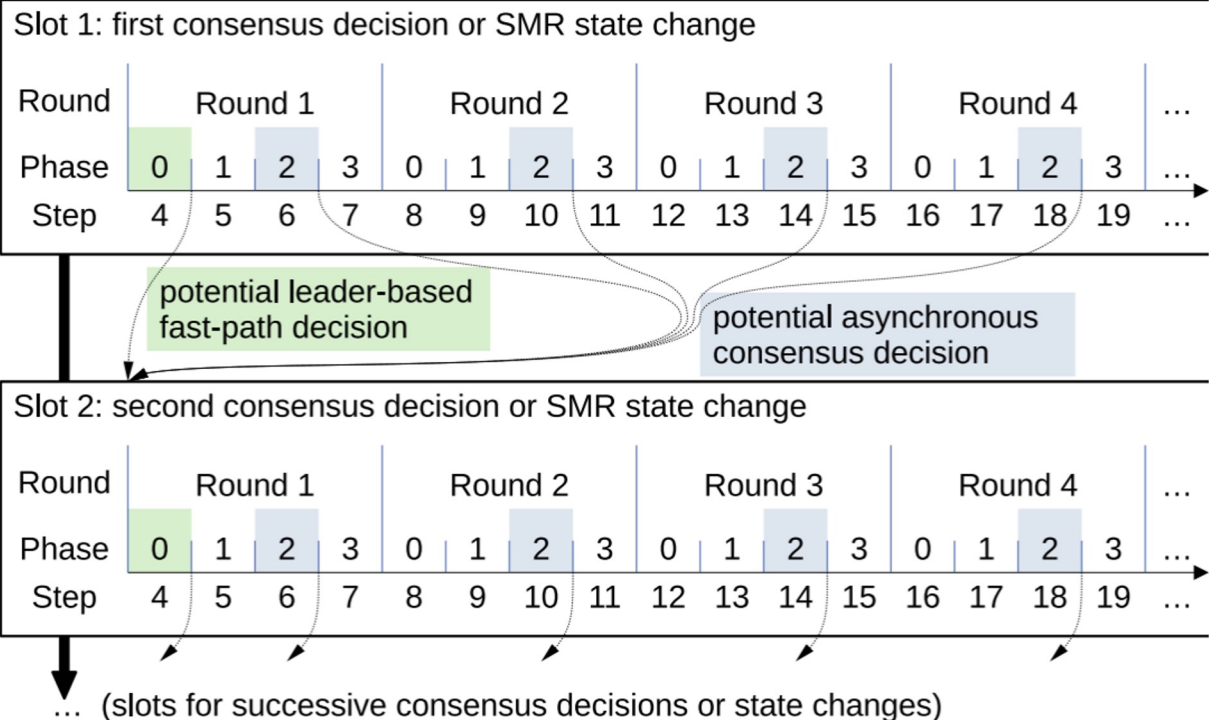


QuePaxa Protocol Diagram



QuePaxa has a fast path decision and a slow path decision

QuePaxa Log Structure



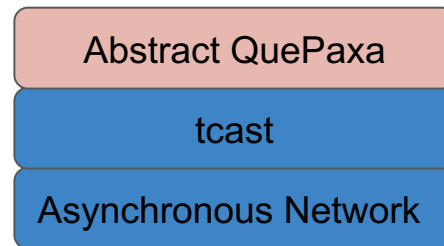
QuePaxa RoadMap

- 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

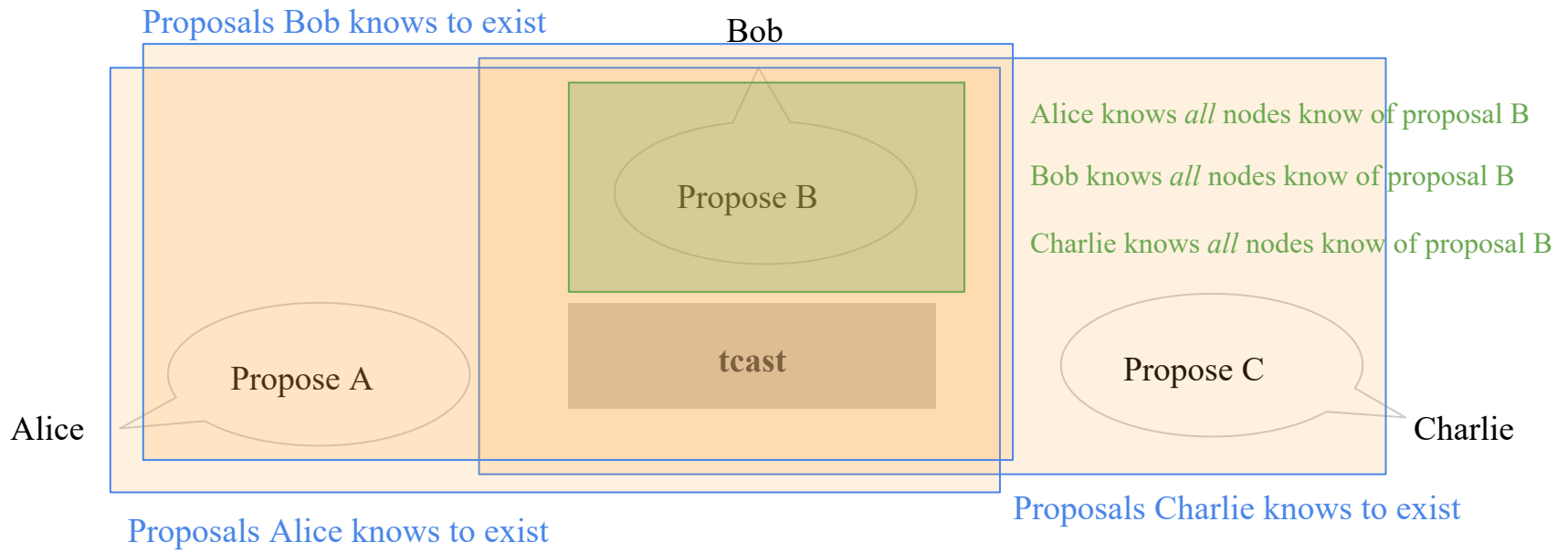
Abstract QuePaxa Algorithm

Algorithm 1: Abstract QuePaxa consensus algorithm

Input: $v \leftarrow$ value preferred by this replica

```
repeat                                     // iterate through rounds
|  $p \leftarrow \langle v, \text{random}() \rangle$       // prioritized proposal
|  $(P, \_) \leftarrow \text{tcast}(\{p\})$            // propagate our proposal
|  $(E, P') \leftarrow \text{tcast}(P)$              // propagate existent sets
|  $(C, U) \leftarrow \text{tcast}(P')$            // propagate common sets
|  $v \leftarrow \text{best}(C).\text{value}$              // next candidate value
| if  $\text{best}(E) = \text{best}(U)$  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

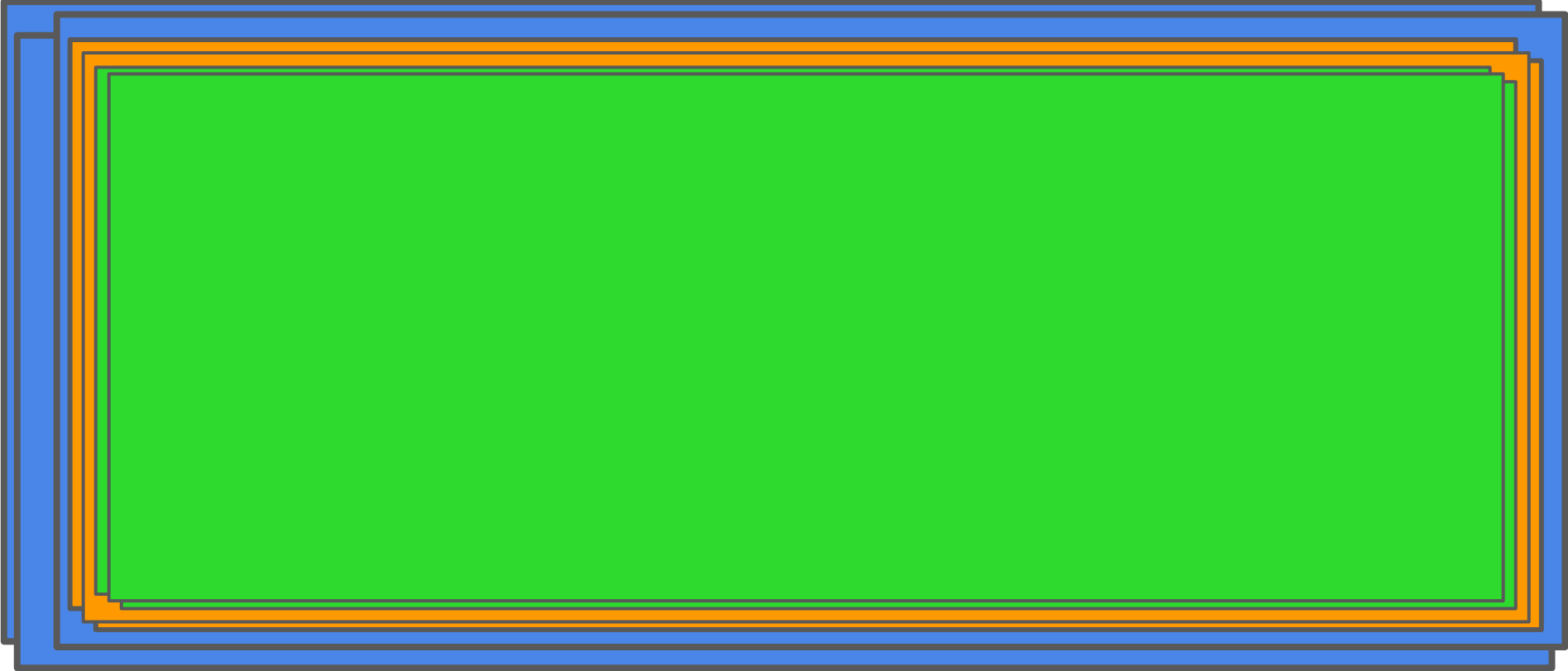
No guarantee that nodes learn the same subsets! (no consensus yet)

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 , $E_i \supseteq C_j \supseteq U_k$**

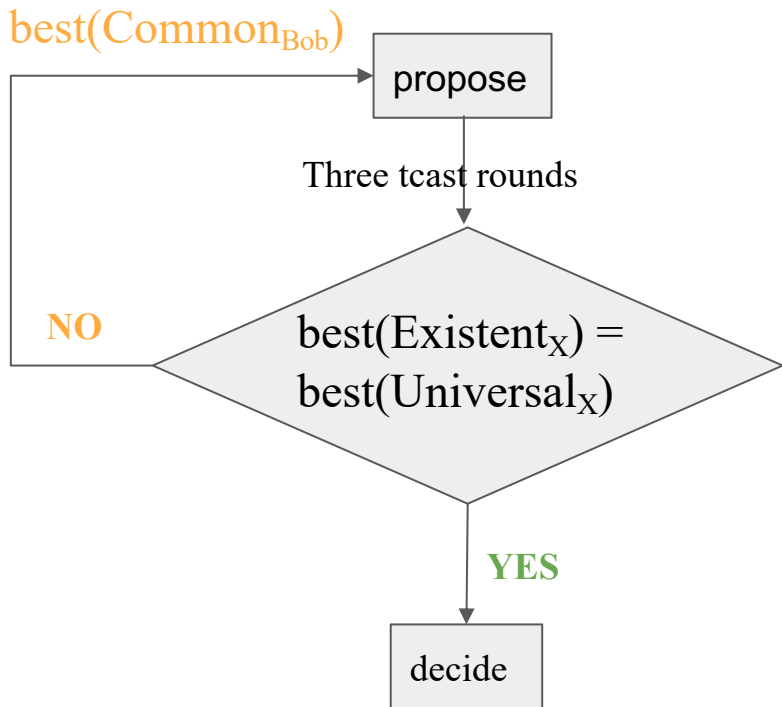
Existent_i \supseteq Common_j \supseteq Universal_k



QuePaxa RoadMap

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

Consensus: reaching a safe decision



Bob doesn't decide, proposes V'

$$\text{best}(\text{Existent}_{\text{Bob}}) \neq \text{best}(\text{Universal}_{\text{Bob}})$$

$$V' =$$

$$\text{best}(\text{Common}_{\text{Bob}})$$

$$\text{Existent}_{\text{Alice}} \supseteq \text{Common}_{\text{Bob}} \supseteq \text{Universal}_{\text{Alice}}$$

Alice decides V

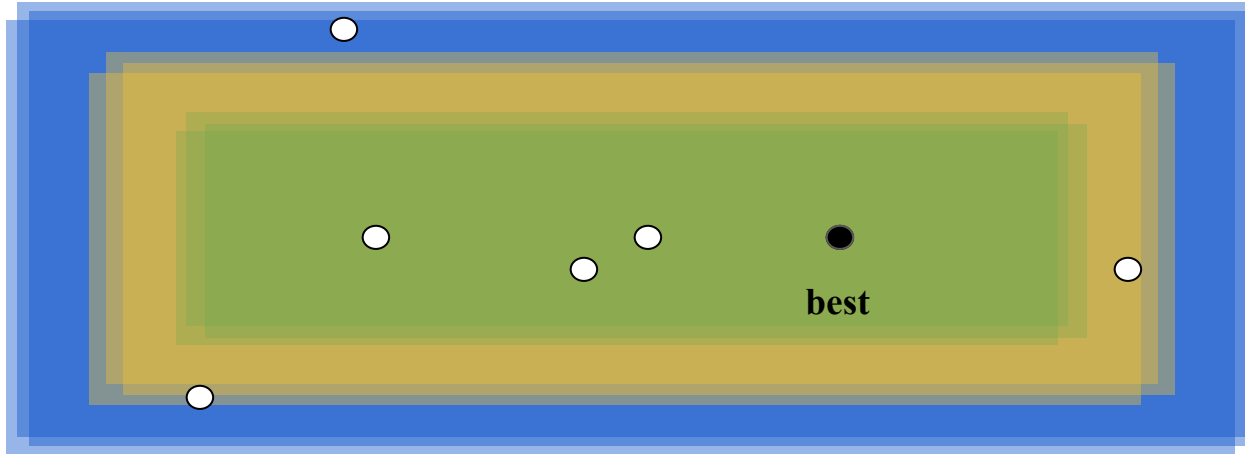
$$\text{best}(\text{Existent}_{\text{Alice}}) = V = \text{best}(\text{Universal}_{\text{Alice}})$$

Only possible decision in future is $V' = \text{best}(\text{Common}_{\text{Bob}}) = \text{best}(\text{Existent}_{\text{Alice}}) = V$

Efficiency: How many rounds until consensus

Probability that Alice decides

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



Each set contains
 $> \frac{1}{2}$ of proposals

Decision probability is $\geq \frac{1}{2} \Rightarrow$ in expectation two rounds until decision

Abstract QuePaxa

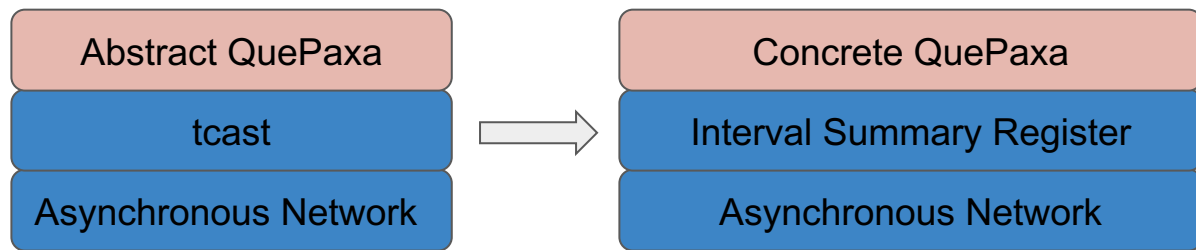
- Avoids timeout from liveness because the protocol is randomized 😊
- Robust against adversarial networks 😊
- $O(n^2)$ message complexity hence slow 😞
- Does not support hedging 😞

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

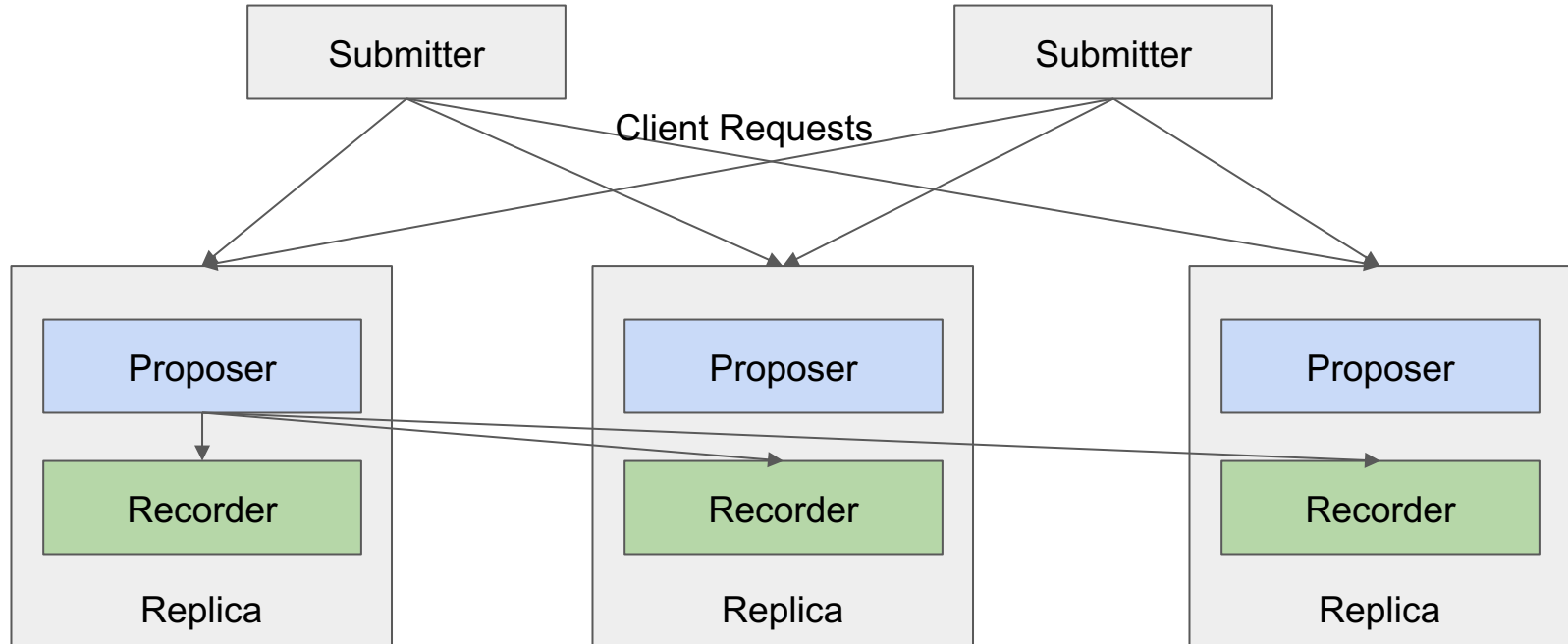
From abstract to concrete QuePaxa



- $O(n)$ 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  $(s, v) \rightarrow (s', f', a')$ :           // handle an invocation
┌
│ if  $s > S$  then                          // advance to a higher step
│    $S \leftarrow s$                           // update current step number
│    $F[s] \leftarrow v$                        // record first value in this step
│
│ if  $s = S$  then                          // aggregate all values
│    $A[s] \leftarrow \mathbf{aggregate}(A[s], v)$  // 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

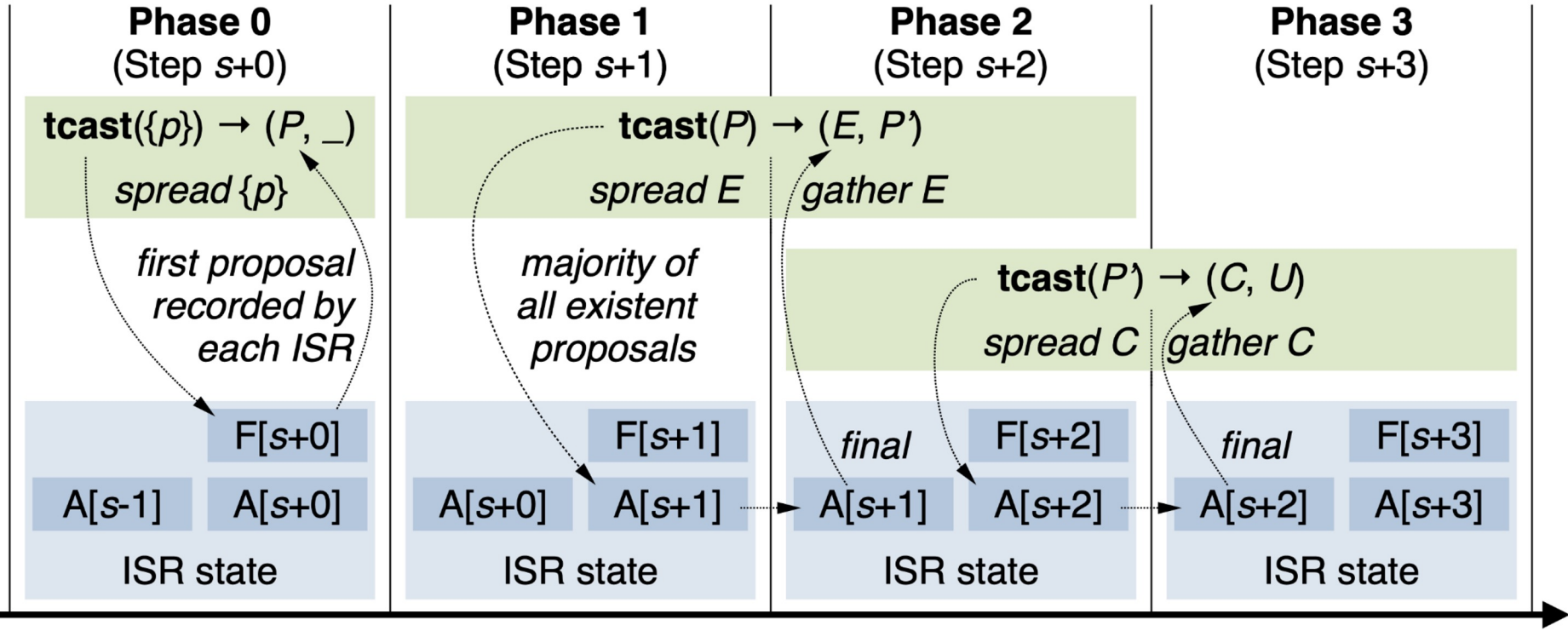
QuePaxa Recorder is a constant space interval summary register

Proposer Code

```
Algorithm 4: Protocol for QuePaxa proposer  $i$   
Input:  $v$  preferred value of this proposer  $i$   
 $s \leftarrow 4 \times 1 + 0$  // start at round 1, phase 0  
 $p \leftarrow \langle H, i, v \rangle$  // initial proposal template  
repeat  
   $p_j \leftarrow p$  for all recorders  $j$  // prepare proposals  
  if  $s \bmod 4 = 0$  and ( $s > 4$  or  $i$  is not leader) then  
     $p_j.\text{priority} \leftarrow \text{random}(1..H - 1)$  for all  $j$   
  Send record( $s, p_i$ ) in parallel to each recorder  $j$   
  Await  $R \leftarrow$  quorum of replies ( $s'_j, f'_j, a'_j$ )  
  if  $s'_j = s$  in all replies received in  $R$  then  
    if  $s \bmod 4 = 0$  then // phase 0: propose  
      if  $f'_j.\text{priority} = H$  in all replies then  
         $\text{return } f'_j.\text{value}$  from any reply in  $R$   
       $p \leftarrow \text{best}_j$  of  $f'_j$  from all replies in  $R$   
    if  $s \bmod 4 = 1$  then // phase 1: spread  $E$   
      // no action required  
    if  $s \bmod 4 = 2$  then // phase 2: gather  $E$ , spread  $C$   
      if  $p = \text{best}_j$  of  $a'_j$  from all replies in  $R$  then  
         $\text{return } p.\text{value}$  // report decision  
    if  $s \bmod 4 = 3$  then // phase 3: gather  $C$   
       $p \leftarrow \text{best}_j$  of  $a'_j$  from all replies in  $R$   
     $s \leftarrow s + 1$  // advance to next step  
  else if any reply in  $R$  has  $s'_j > s$  then  
     $s, p \leftarrow s'_j, f'_j$  // catch up to step  $s'_j$ 
```

QuePaxa proposer uses RPC in 4 phases to contact Recorders

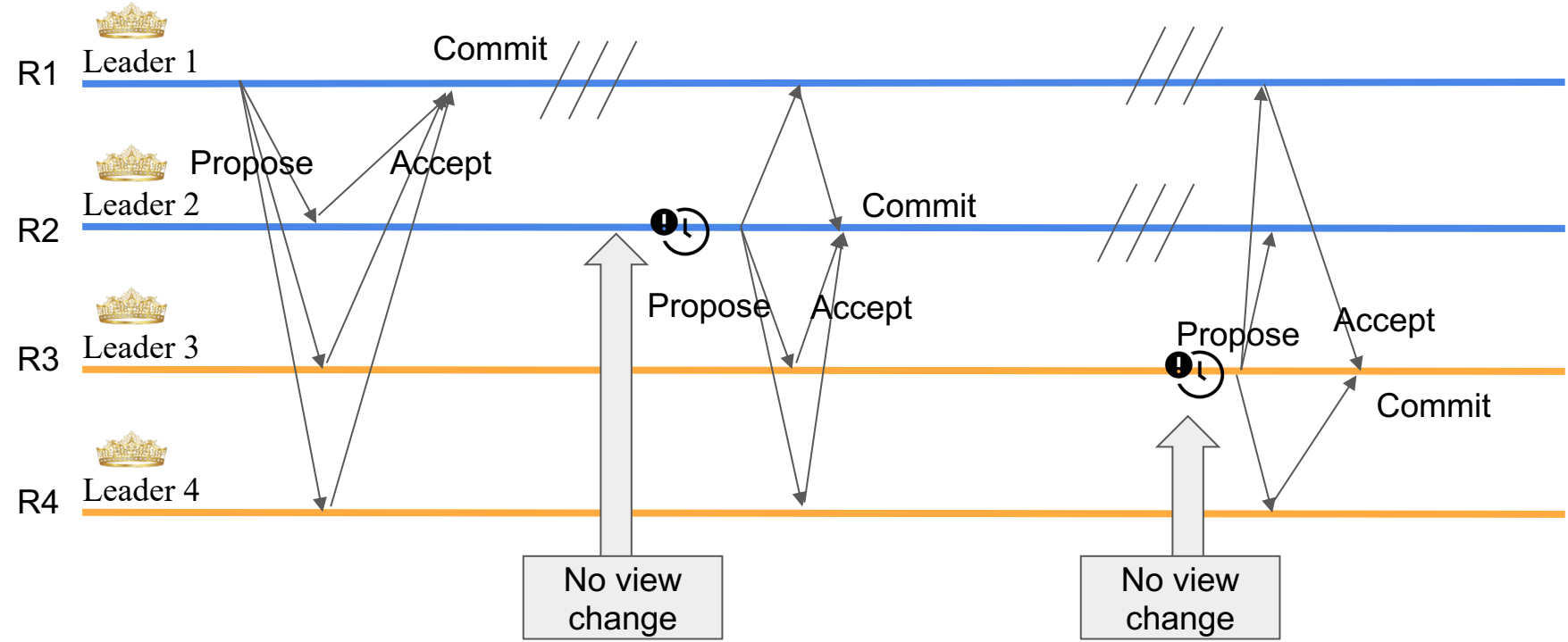
How tcast abstraction maps to concrete QuePaxa phases



QuePaxa RoadMap

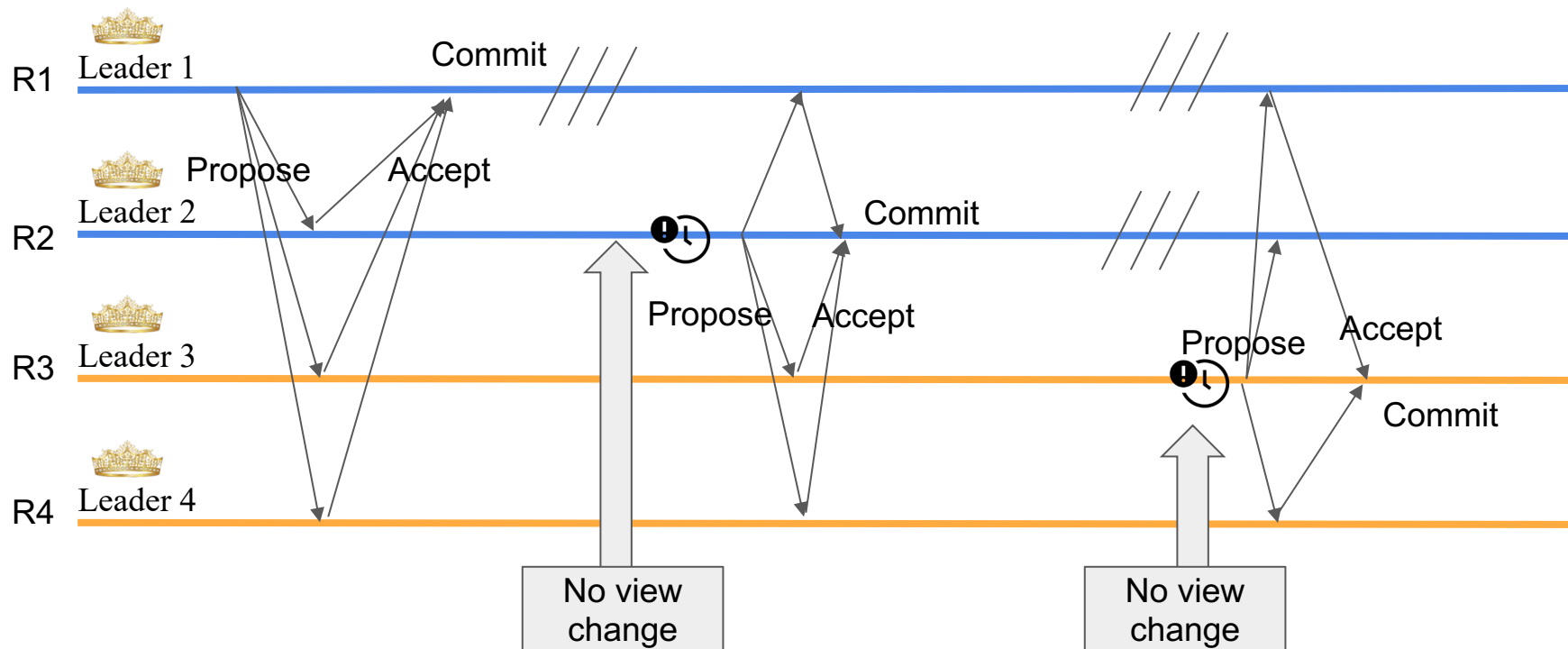
- Operation Overview
- Abstract QuePaxa
- Safety and liveness of abstract QuePaxa
- Concrete QuePaxa overview
- **The QuePaxa fast path**

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

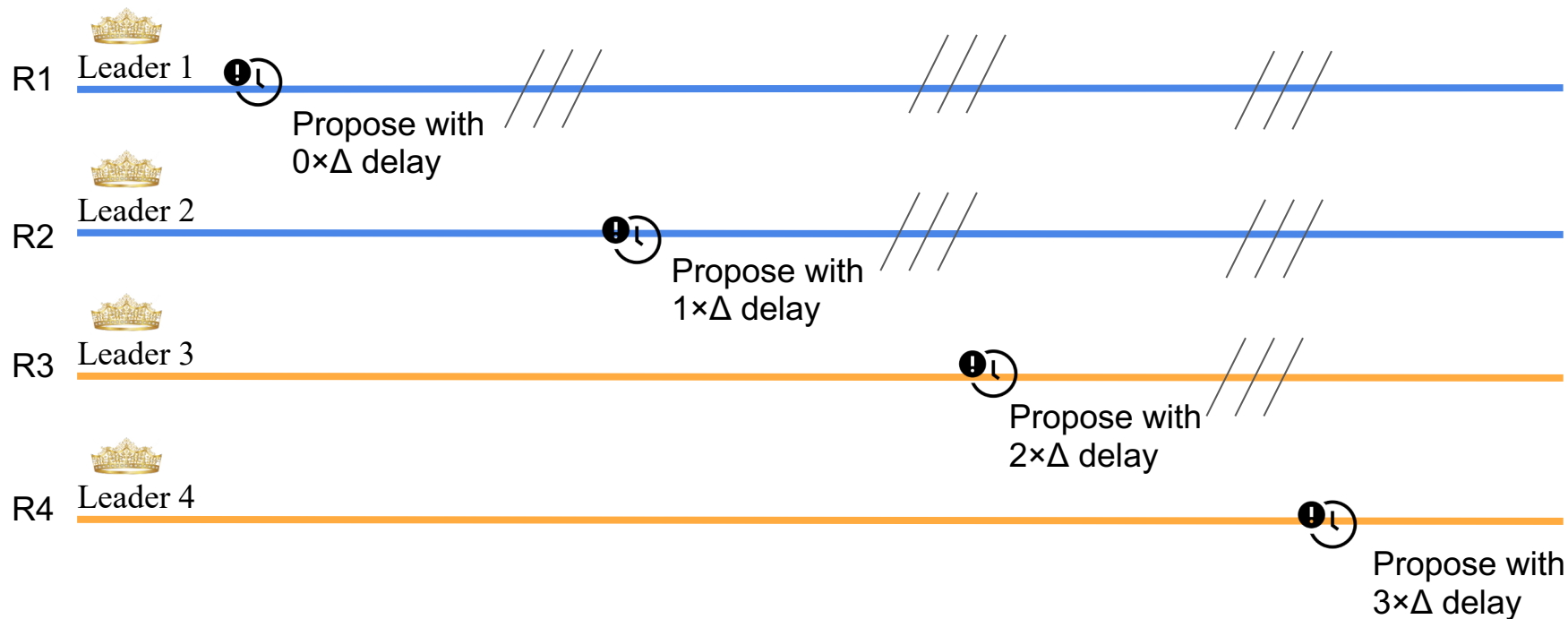


Can we change leaders **without** view changes if the current leader is sub optimal?

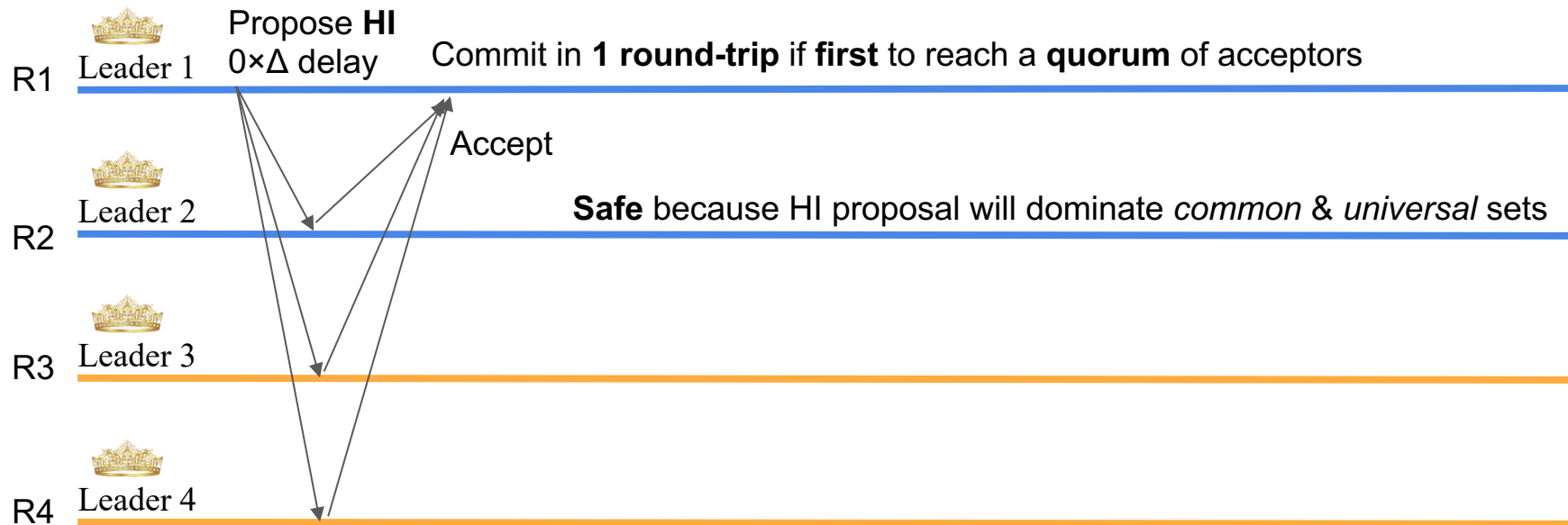
In QuePaxa, multiple leaders **can** propose without view changes



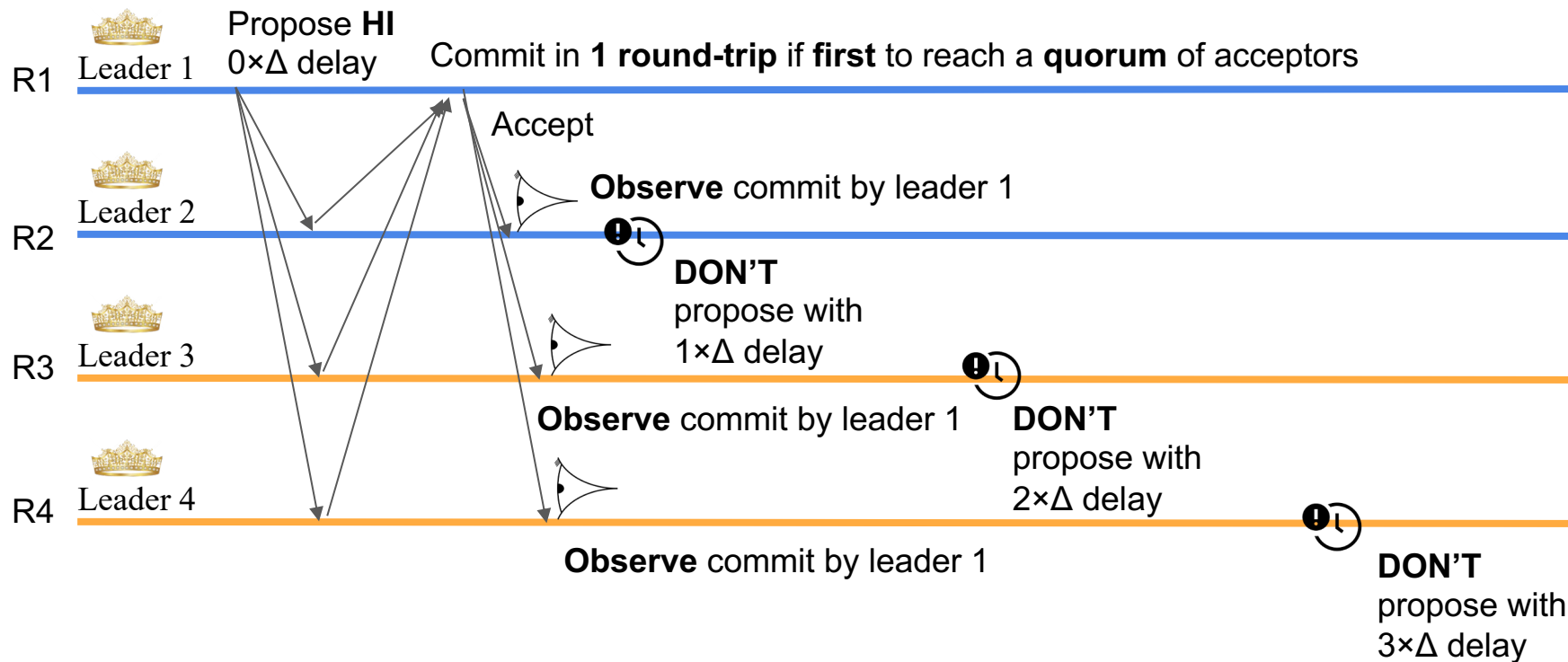
All potential leaders propose on well-known **hedging schedule**



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



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



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

Performance robustness in challenging network situations

What if:

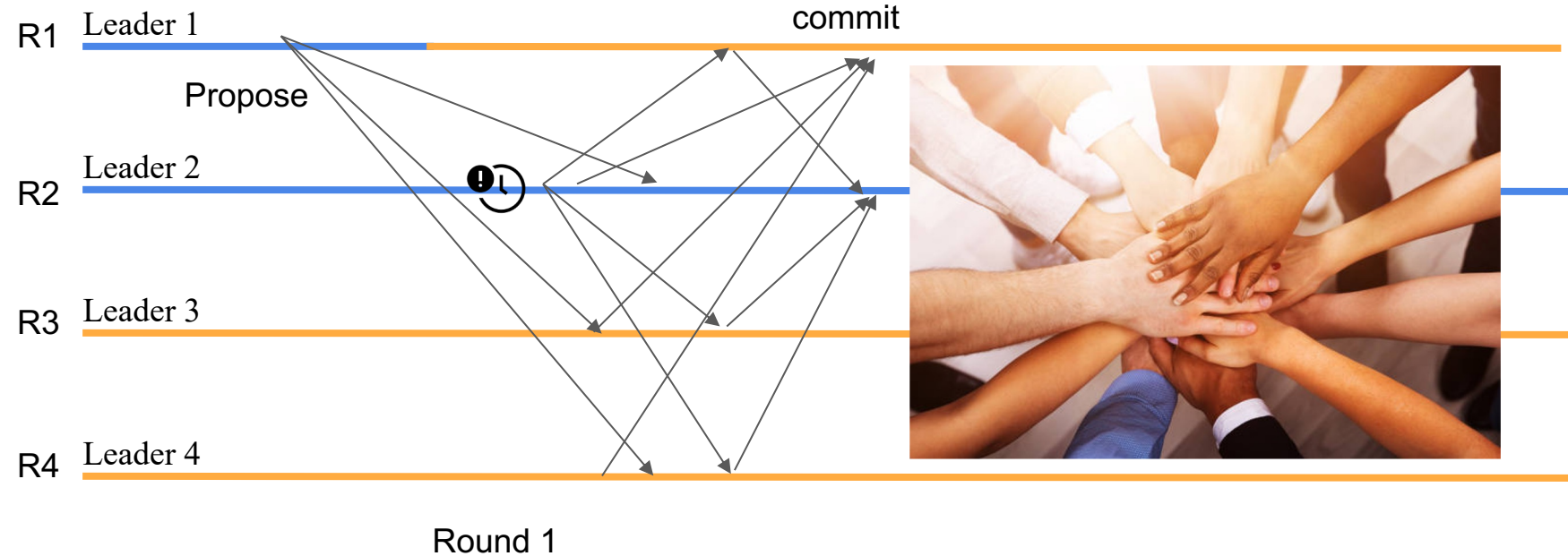
- Network experiences periods of **high delay** (e.g., due to congestion)?
- Network exhibits **high jitter** or **delay unpredictability** (e.g., bursty loads)?
- Timeouts or hedging delays **misconfigured** too low for actual network?

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*

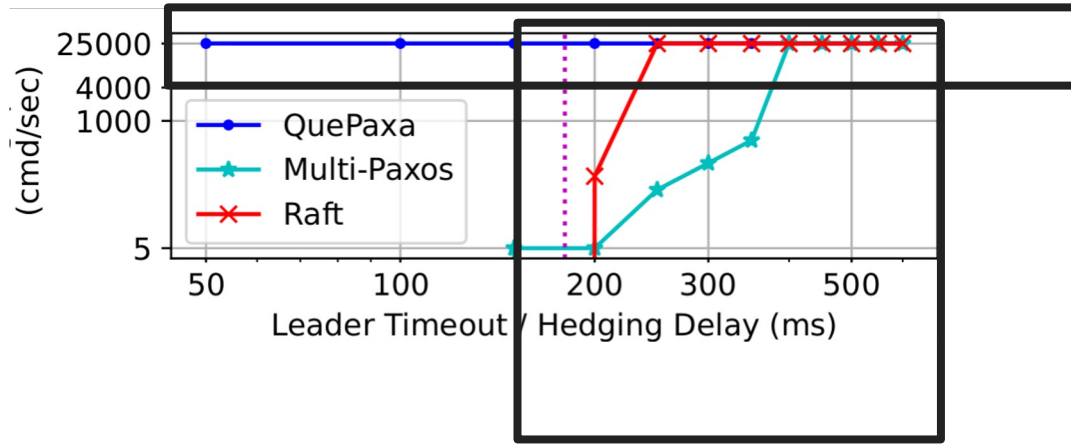
Setup

- 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



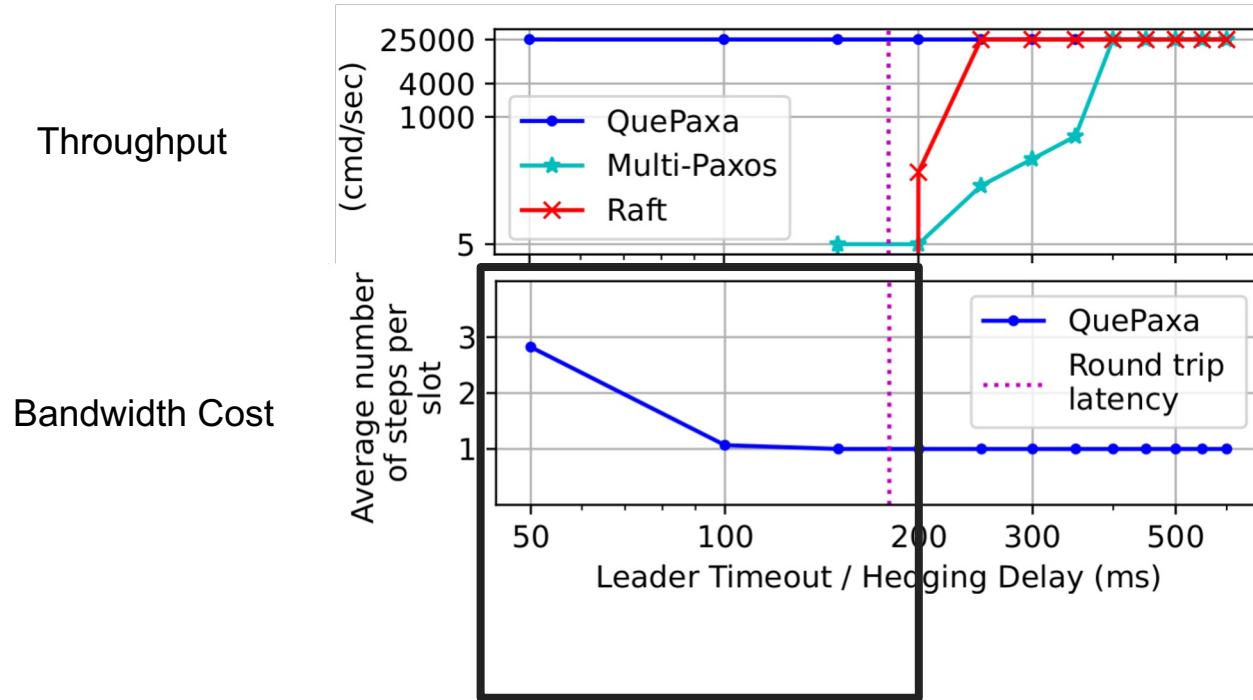
Effect of Hedging in Quepaxa

Throughput



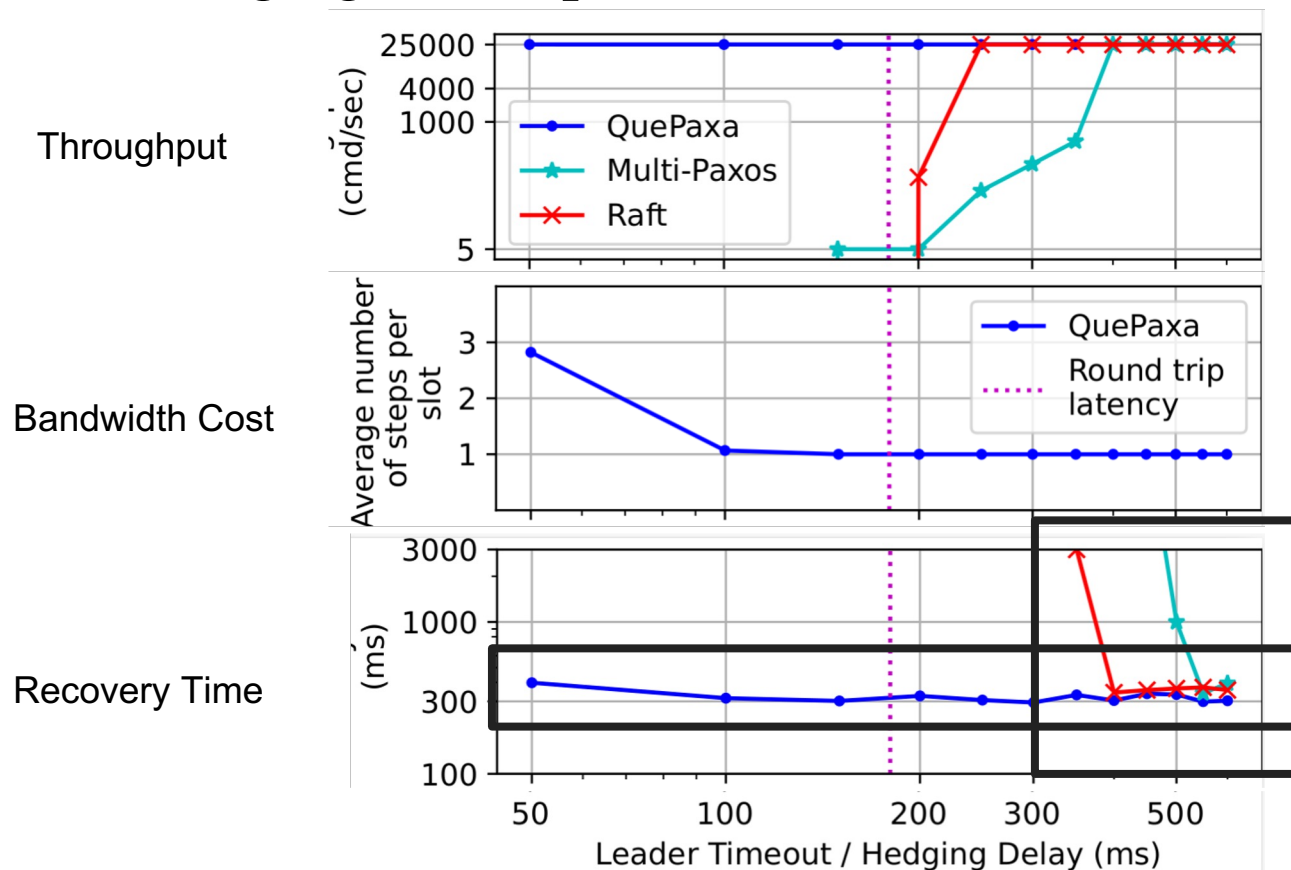
QuePaxa is live for any hedging delay

Effect of Hedging in Quepaxa



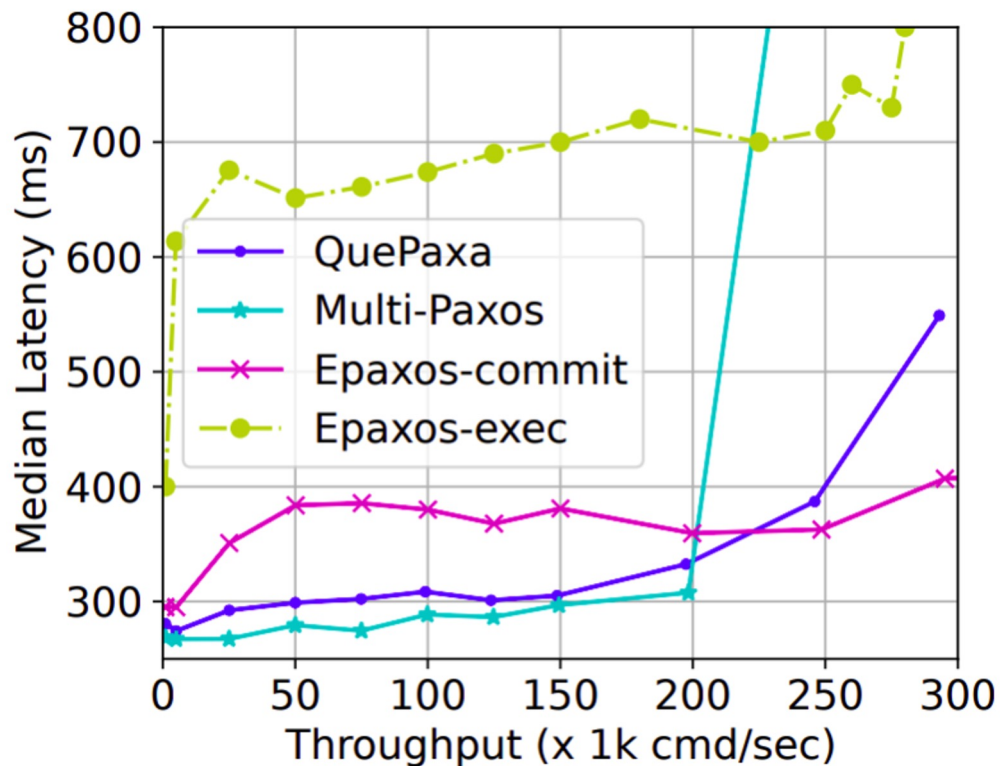
QuePaxa has an additional overhead only when hedging delay < RTT

Effect of Hedging in Quepaxa



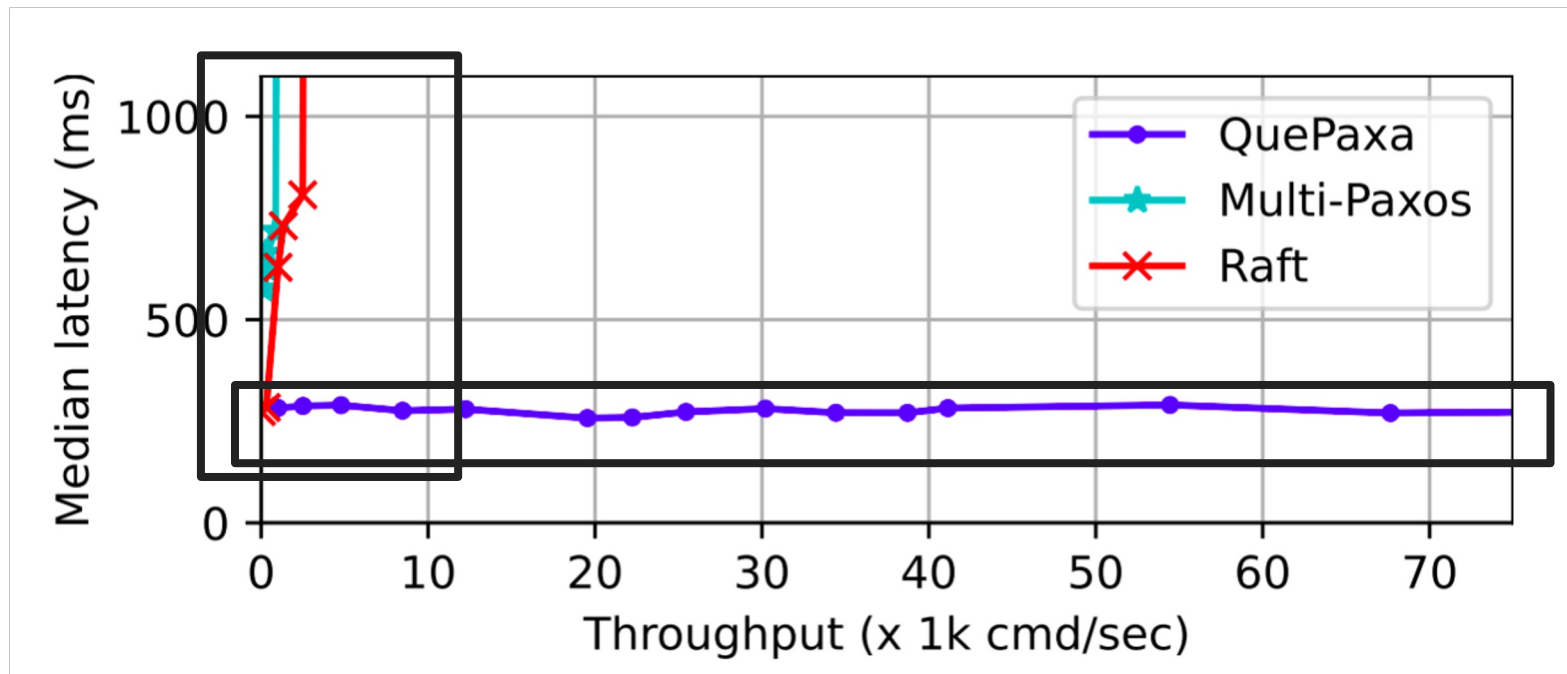
QuePaxa has low recovery time

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>



Advanced Blockchain Architectures

Limitations of mainstream blockchain platforms

- Weak finality – probabilistic, ~1h for Bitcoin
- Transaction throughput & cost – few TPS
- Transaction commit latency
- Fairness – Power distributed by investment, not by people
- Energy waste (PoW)
- Privacy – TX identities, amounts, data, computations, etc.
- Following blockchains efficiently
- Bridging between chains
- Governance – upgrade, replace

Current State of Throughput

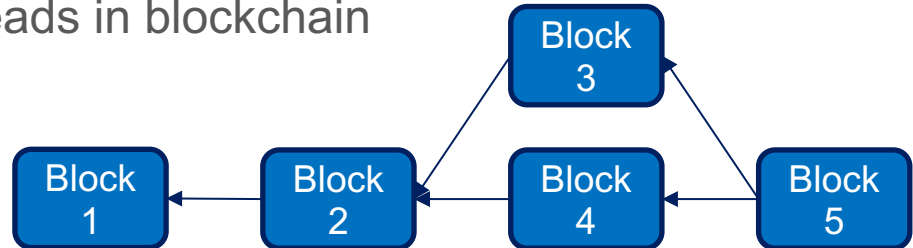
- Bitcoin:
4 – 7 TPS (transactions per second)
- Ethereum:
20 – 30 TPS
- VISA:
1500 – 4000 TPS (pre-2020)
~8000 TPS (now, can scale up to 65'000 TPS)

Solutions: ?

Scaling Throughput – Approaches

Tweaks:

- Scale up of nodes
- Reduce block time
 - Bitcoin's 10min
 - Ethereum 15s
- Make use of wasted effort
 - Use orphaned / competing heads in blockchain
 - Turn blockchain into a DAG



Scaling Throughput – Approaches

Bypassing the blockchain / batching transactions :

- Payment channels – e.g. Lightning network
- Layer 2 chains – separate consensus mechanism
 - Rapid transactions, instant finality, periodically committed to main chain
 - ZK rollups: verify transaction history, not just hash
 - Optimistic rollups: claim, can be challenged during time window
- Side-chains (→ bridging problem)

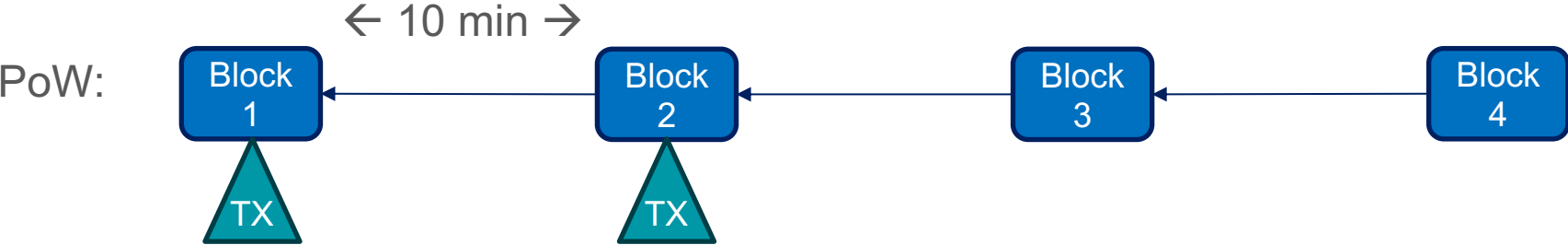
Scaling Throughput – Approaches

Rethinking consensus :

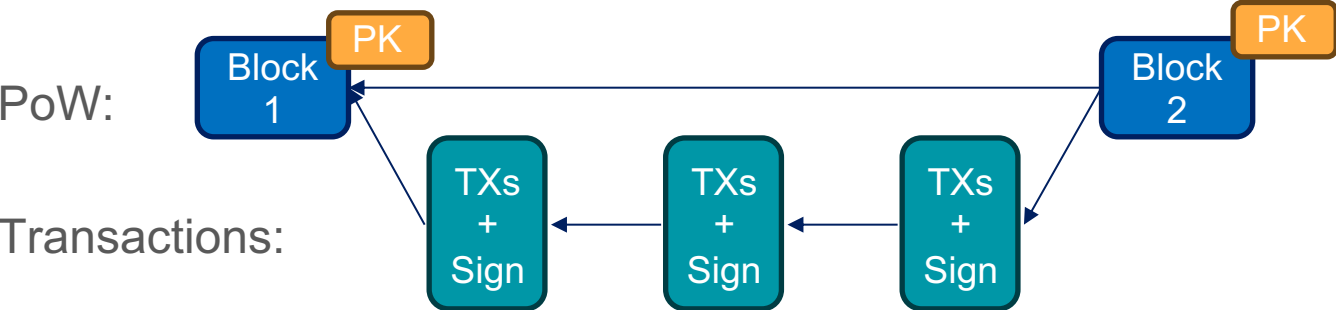
- Hybrid consensus mechanisms – Bitcoin-NG, ByzCoin
 - Can we achieve (permissionless) performance closer to PBFT ?
- Sharding – OmniLedger, Cardano (Ouroboros), Ethereum (past roadmap)
 - Can we scale out ?

Scaling Throughput – Bitcoin-NG (Next Generation)

- Bitcoin



- Bitcoin-NG



Scaling Throughput – Bitcoin-NG (Next Generation)

- PoW becomes a form of leader election
- Transaction processing is separate from PoW
- Incentives need to be adjusted

Pro:

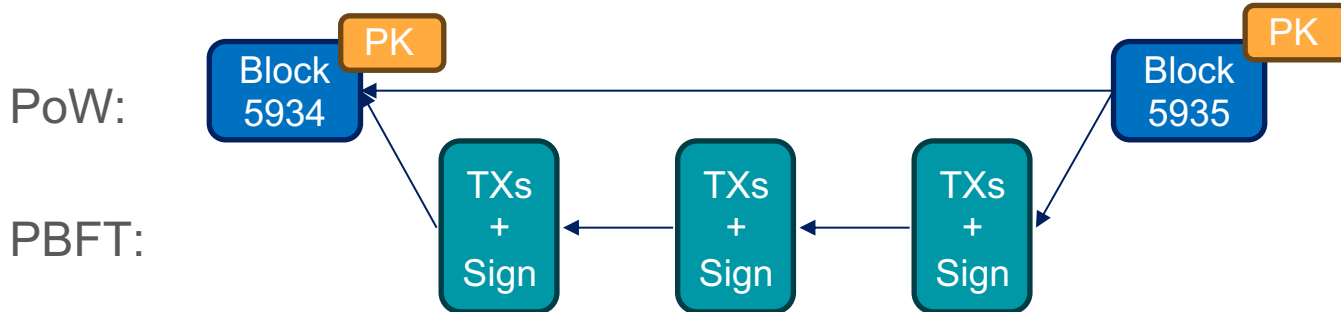
- higher TPS

Con:

- “temporary dictator” (DoS-able)

Scaling Throughput – ByzCoin

- Built on Bitcoin-NG
- Avoid “temporary dictator”
- Form a consensus committee out of 100-1000 last PoW winners
- Operate with BFT consensus within the committee (e.g. PBFT)



Scaling Throughput – ByzCoin

Challenges:

How do clients know & verify which micro-blocks were committed ?

- Collective signing (threshold of committee)

How do you rotate the committee ?

- Option 1: via PBFT, “new committee” TX
→ guarantees safety
- Option 2: via PoW alone
→ guarantees liveness

Scaling Throughput – Sharding

Problem:

- no “scale out” in blockchains, everyone processes everything

Desired state:

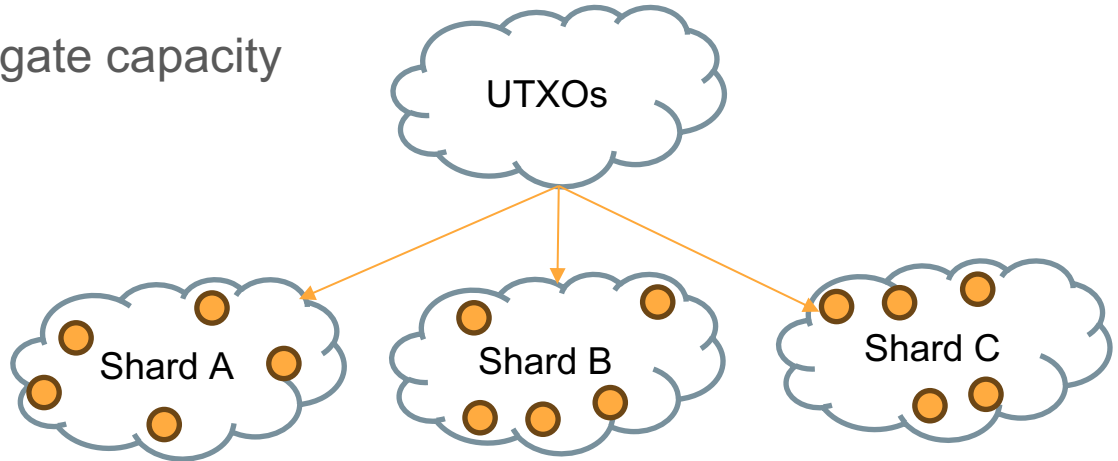
- more miners = more aggregate capacity

Potential solution:

- Shard across UTXO

Challenges:

- How big a shard?
- Cross-shard transactions
- Threat: shard takeover



Scaling Throughput – Sharding

How big should a shard be ?

- 1 shard for everything (= no sharding)
- 1 shard per miner
 - Dictator over those transactions
 - Risk of malicious takeover

Solution ?

- “big enough” shards
- Random & representative shards (Drand to the rescue!)

Scaling Throughput – Sharding

How can we handle cross-shard transactions ?

- Two-phase transaction lock
- Request all shards to lock a transaction, then execute
- On failure / lack of responses: abort
- On success: commit

If shards are untrusted:

- Time lock
- Revert unless transaction goes through
- Much more complex protocol, guarantees safety + liveness

Approaches to energy efficiency

- Proof of Burn
based on other PoW blockchains
- Proof of Storage
e.g. Chia
- Proof of Sequential Work
Verifiable Delay Functions (VDF)
- Proof of Elapsed Time
requires trusted execution environments (hardware)
- Proof of Stake

Proof of Stake

How to form a consensus group ?

Ethereum: RANDAO

Algorand:

- Random sample of stakeholders
- Use VRF to generate a unique “lottery ticket” per node, per round
- Ticket “wins” if below a threshold value
- Reveal winning ticket along with consensus vote

Why change consensus group at every block ?

- Protect against fast, adaptive adversary
- Prevents DoS attacks, nodes hacking

Following blockchains efficiently – Skipchains

Problem:

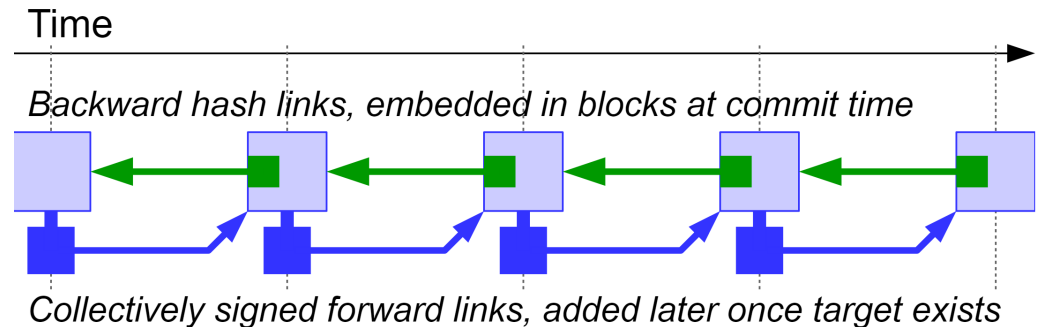
- Retrieving information from a blockchain requires following it actively, fully

Solution:

- ByzCoin-like collective signatures
- Add “forward links” in addition to backward (hash) links

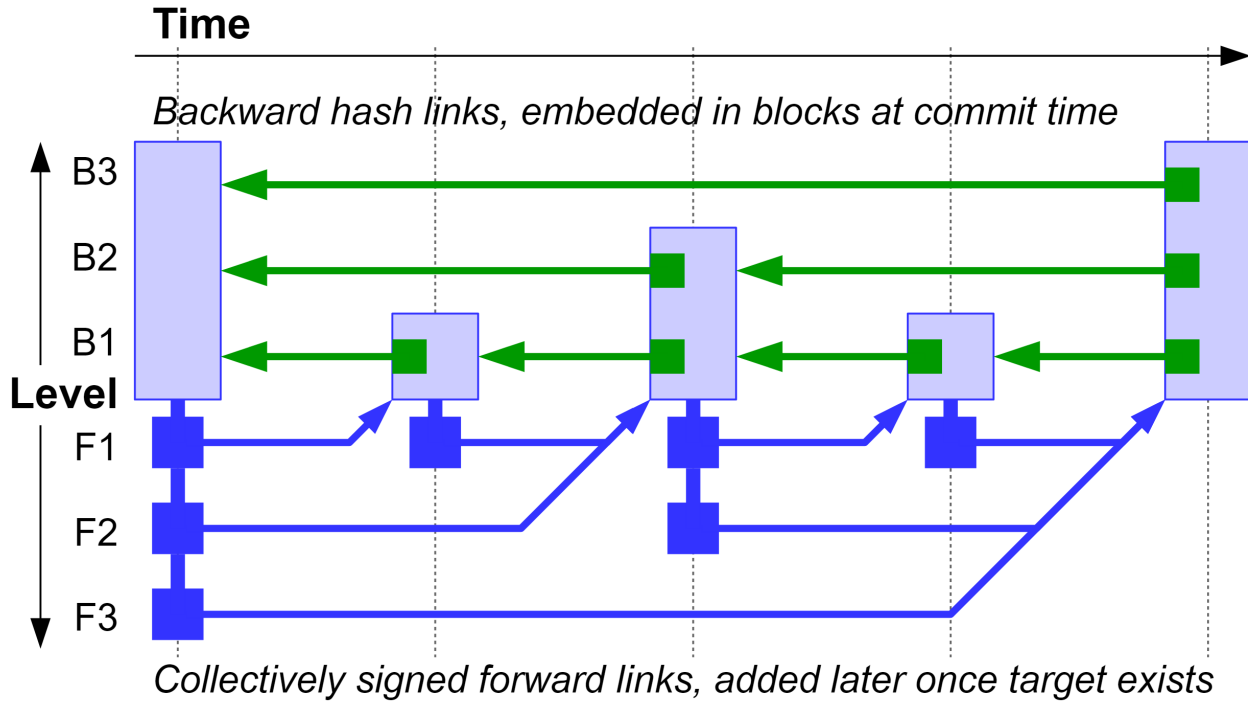
Still requires $O(n)$

How can we do better ?



Following blockchains efficiently – Skipchains

Solution:



- Multi-level links to enable $O(\log(n))$ retrieval

Next steps

Mandatory reading:

- Algorand
- OmniLedger

Optional readings:

- Calypso: Private Data Management for Decentralized Ledgers
- Enhancing Bitcoin Security and Performance with Strong Consistency via Collective Signing