Distributed Transaction Processing in Untrusted Environments

Mohammad Javad Amiri Divyakant Agrawal Amr El Abbadi Boon Thau Loo

Stony Brook University UC Santa Barbara University of Pennsylvania

SIGMOD
PODS
2024

Motivation

- A large part of our existence is digital or virtual:
  - Highly interconnected, and
  - Increasingly interdependent
- These interconnections and dependencies rely on myriad actors whose trustworthiness is not always guaranteed.
- Advances in technology have enabled billions of us with:
  - Vast amount of information, and
  - Unlimited connectivity
- Unfortunately, this rapid digitization of our personal lives is vulnerable to malicious activity or malicious intent in the internet.

Paradigm shift

- Large body of work in the area of computer and network security:
  - Focuses, however, on the reactive or monitoring approach
  - Wait for the problem to occur and then weed out the malicious activity
- Lower barrier to entry for malicious activity.
- Need preventive approaches to address the lack of trust.
- Re-evaluate the model that assumes technology components are benign and passive:
  - Develop technology solutions that can withstand malicious or untrusted nature of the underlying infrastructure (network and compute servers)

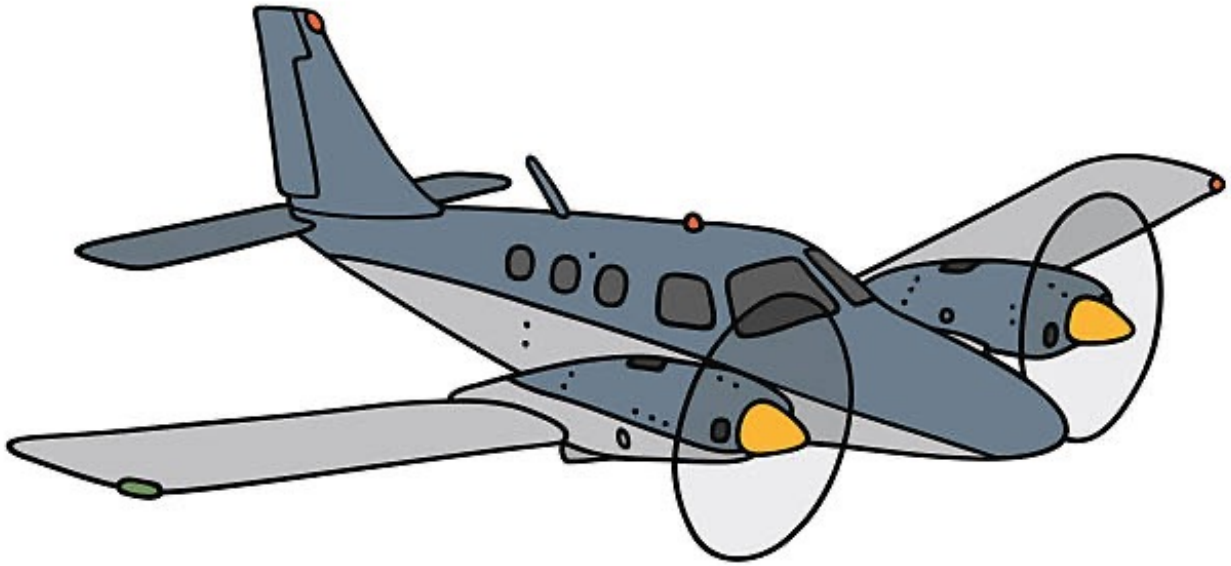
Processing transactions

Tnx_1: (Alice,Bob,10)

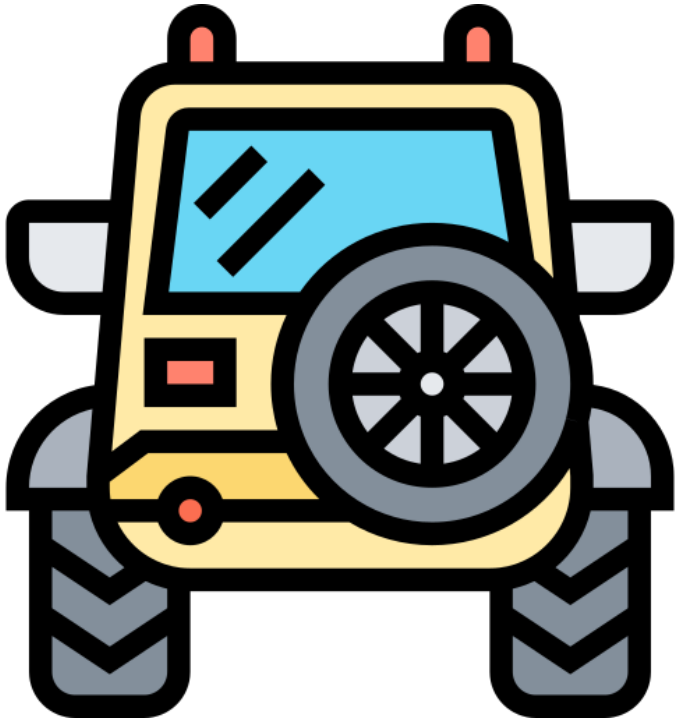
ID	Name	BAL
1	Alice	~~20~~10
2	Bob	~~10~~20
3	Carol	15

What if the server crashes?!

Fault tolerance through redundancy

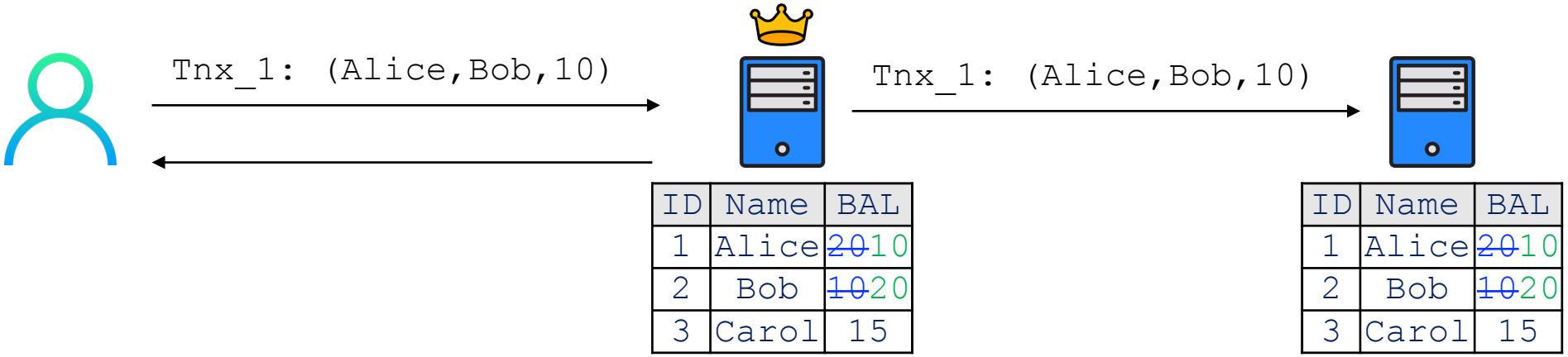


Twin-engine plane



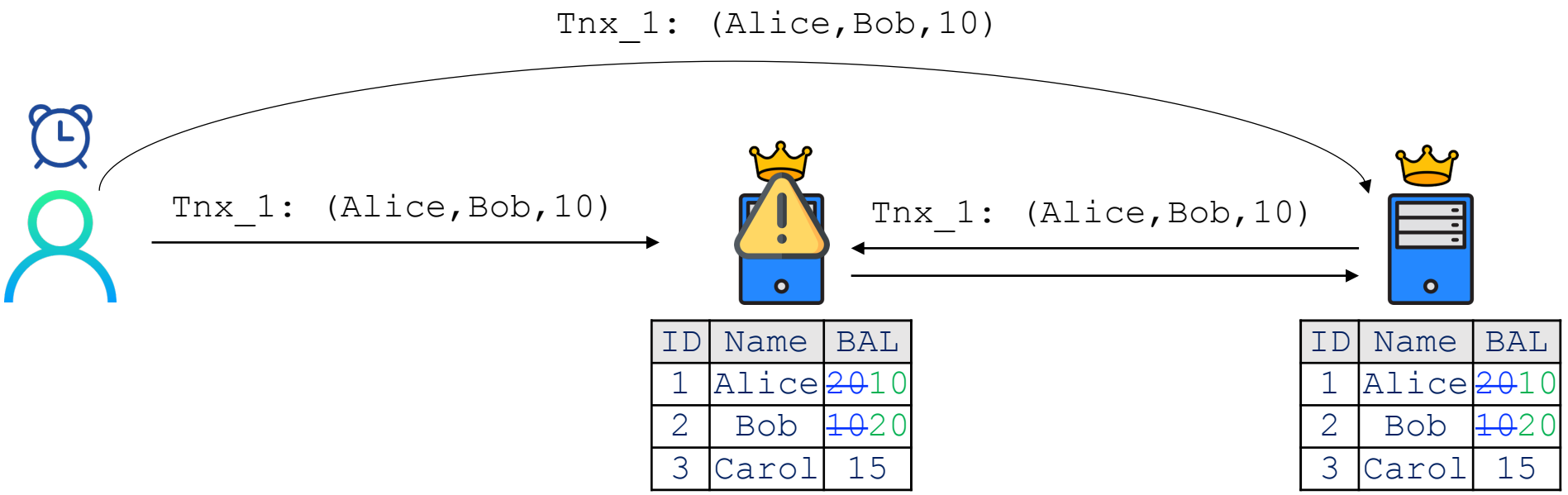
Spare tire

Fault tolerance through redundancy



- Assuming at most one server might fail at each time
- Crash failure: fail by stopping, no malicious behavior
- Order of transactions matters

Fault tolerance through redundancy

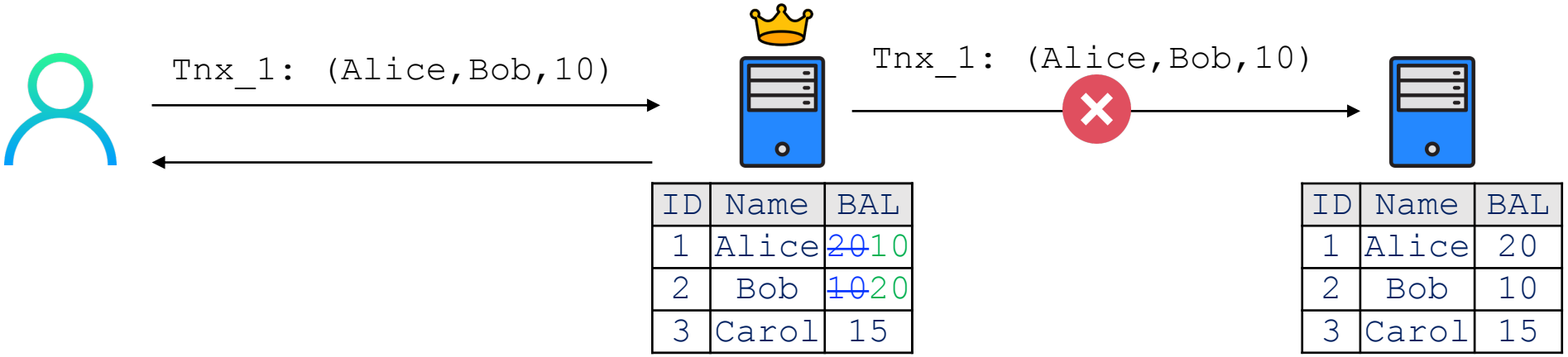


ID	Name	BAL
1	Alice	~~20~~10
2	Bob	~~10~~20
3	Carol	15

ID	Name	BAL
1	Alice	~~20~~10
2	Bob	~~10~~20
3	Carol	15

How to ensure that the state is replicated?!

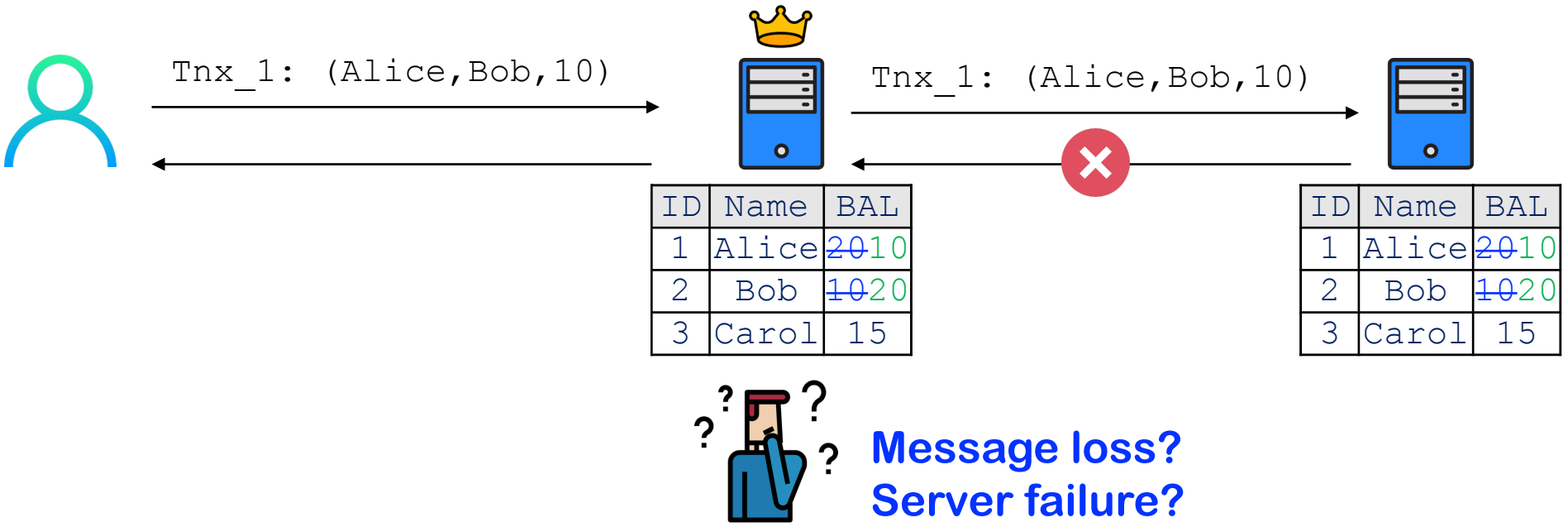
Fault tolerance through redundancy



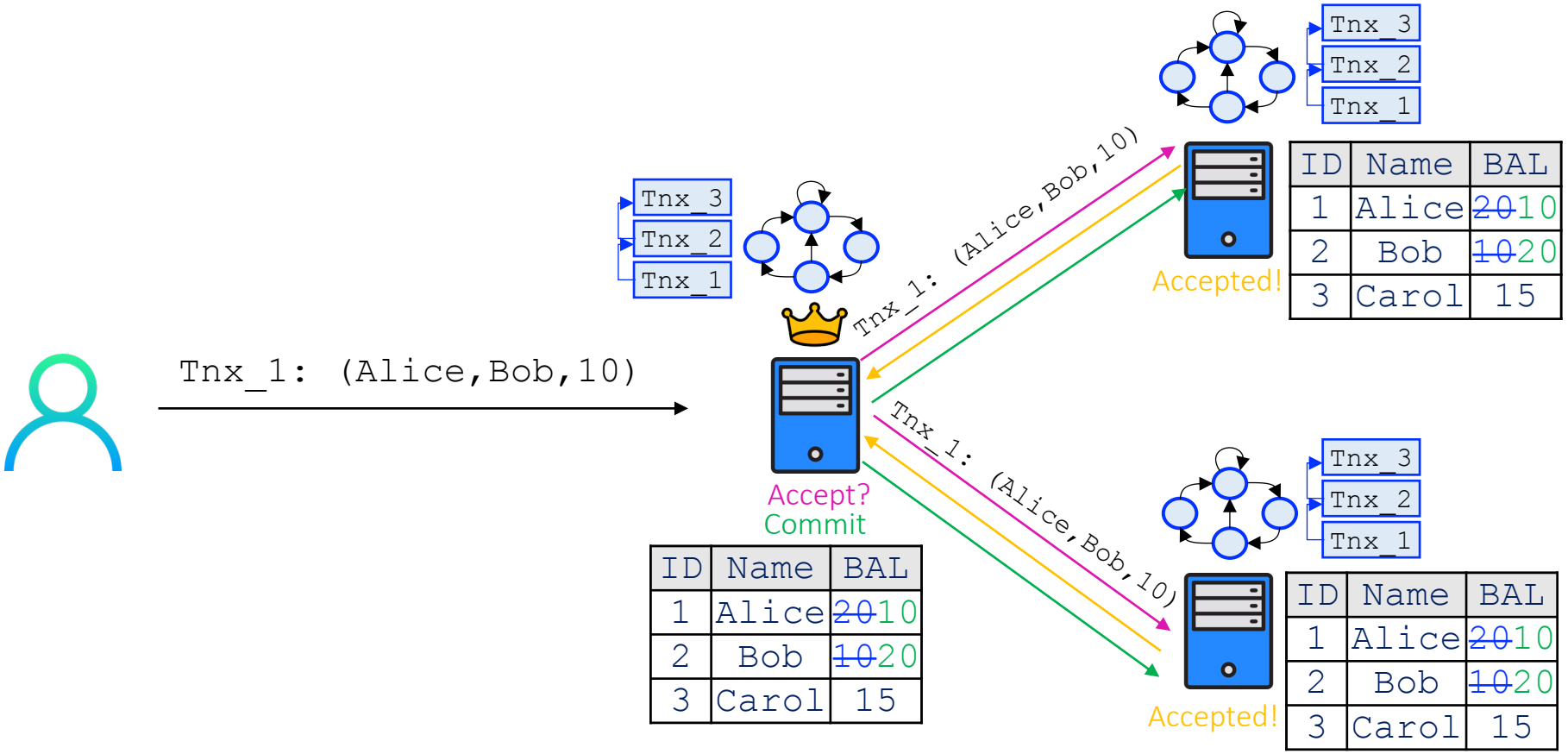
ID	Name	BAL
1	Alice	~~20~~10
2	Bob	~~10~~20
3	Carol	15

ID	Name	BAL
1	Alice	20
2	Bob	10
3	Carol	15

Fault tolerance through redundancy



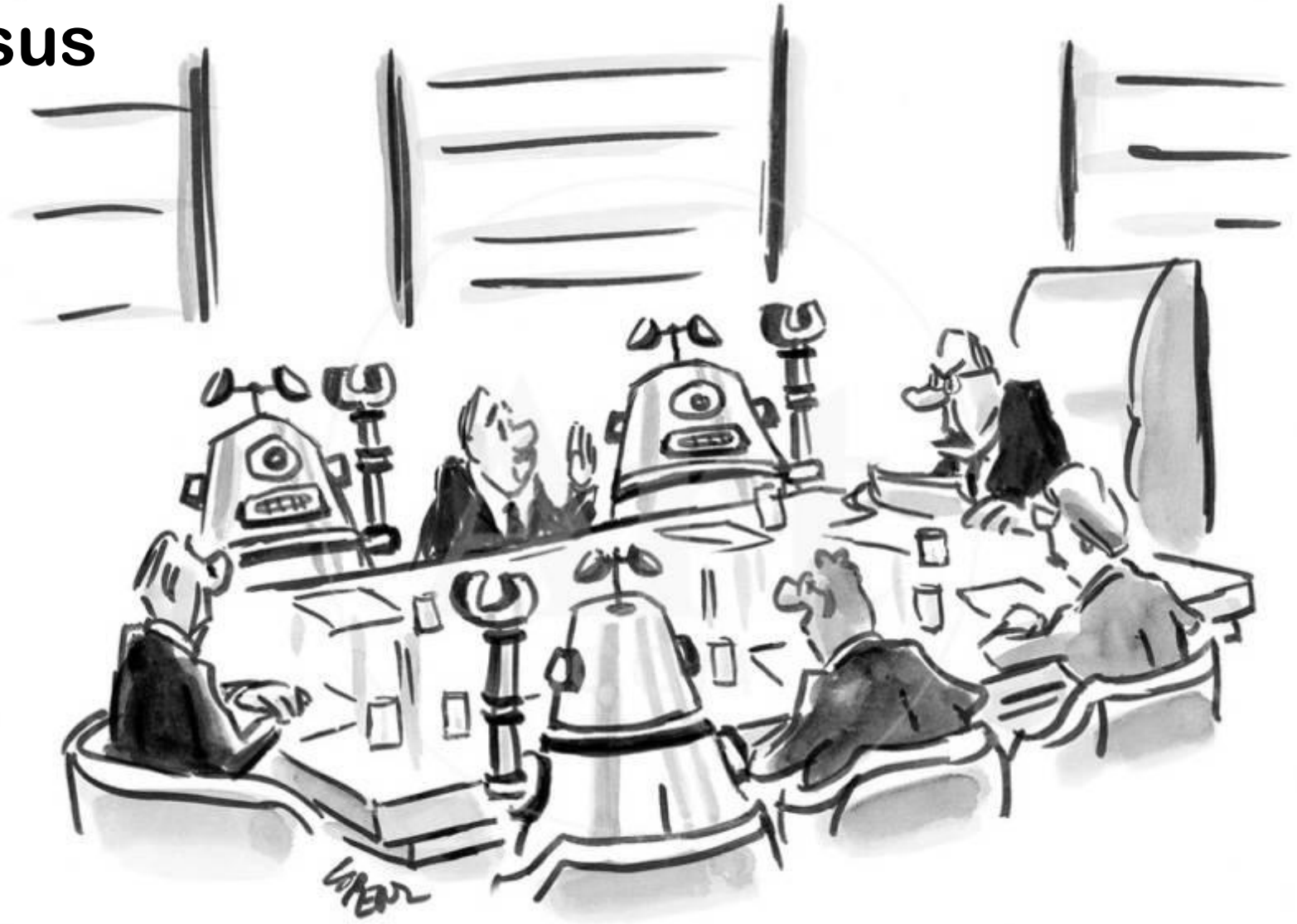
Distributed transaction processing



State Machine Replication: a replicated service whose state is mirrored across different deterministic replicas

- Assign each client request an order in the global service history and execute it in that order

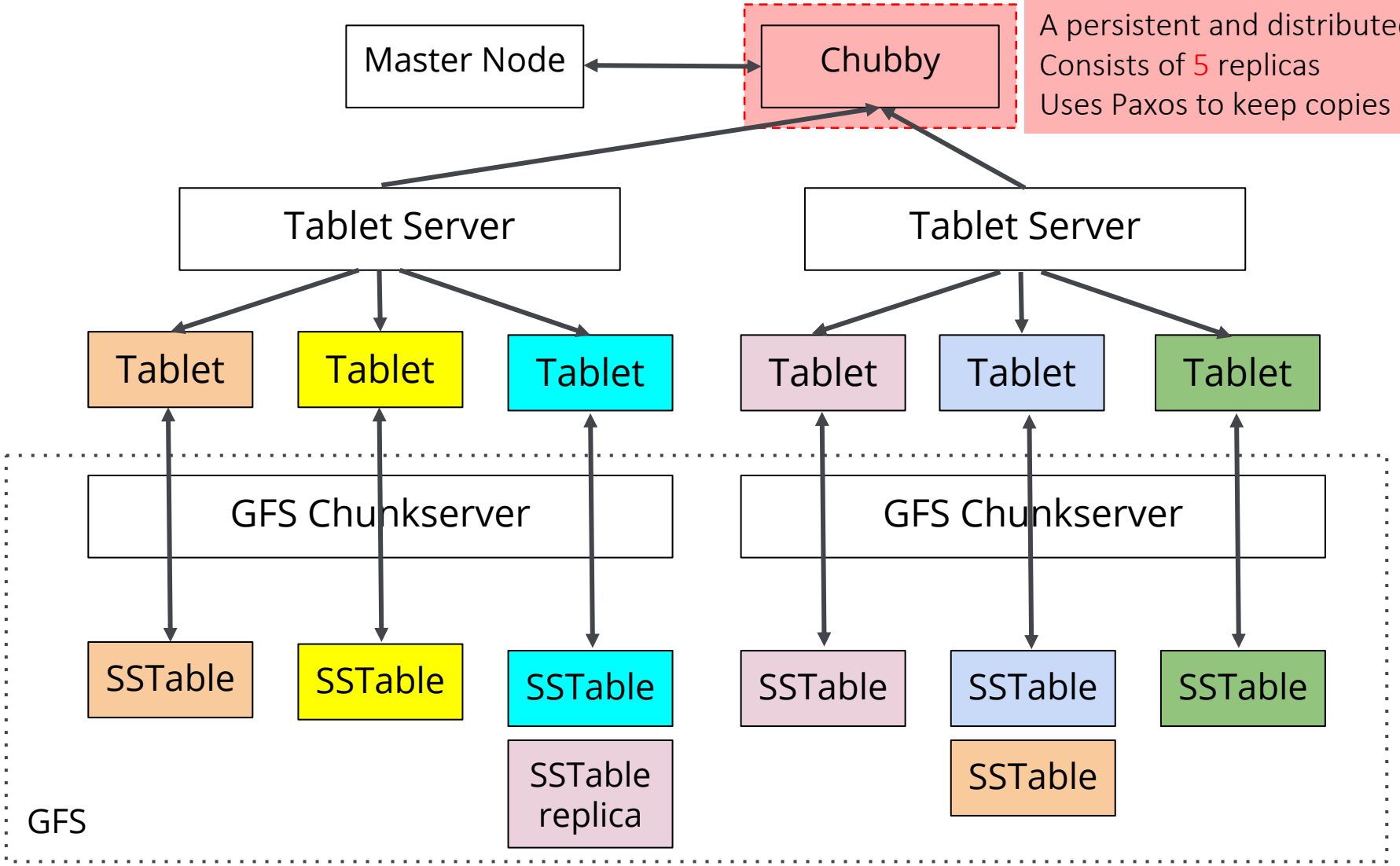
Consensus



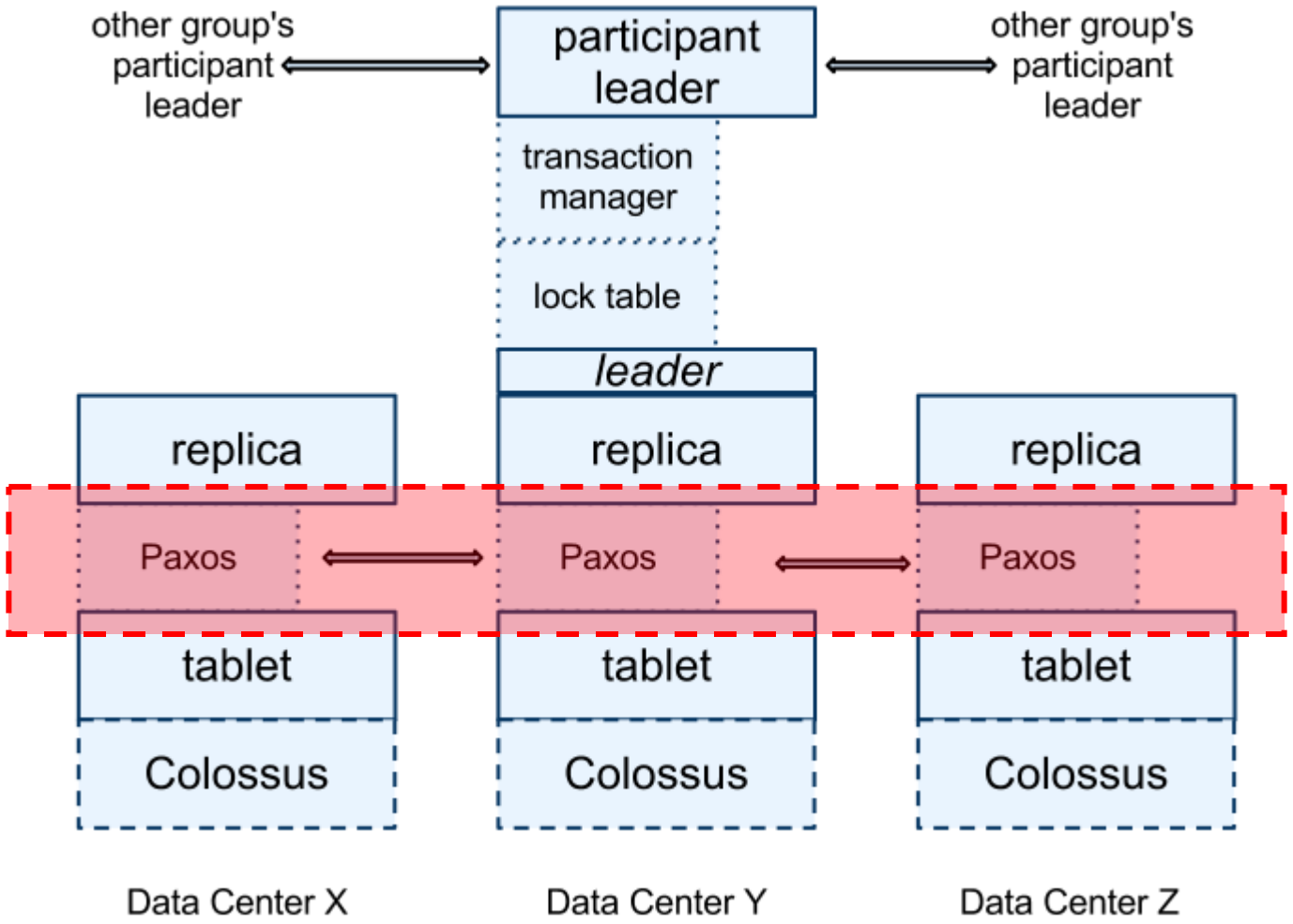
"Jenkins, if I want another yes-man I'll build one."

A set of distributed nodes need to reach agreement on a single value

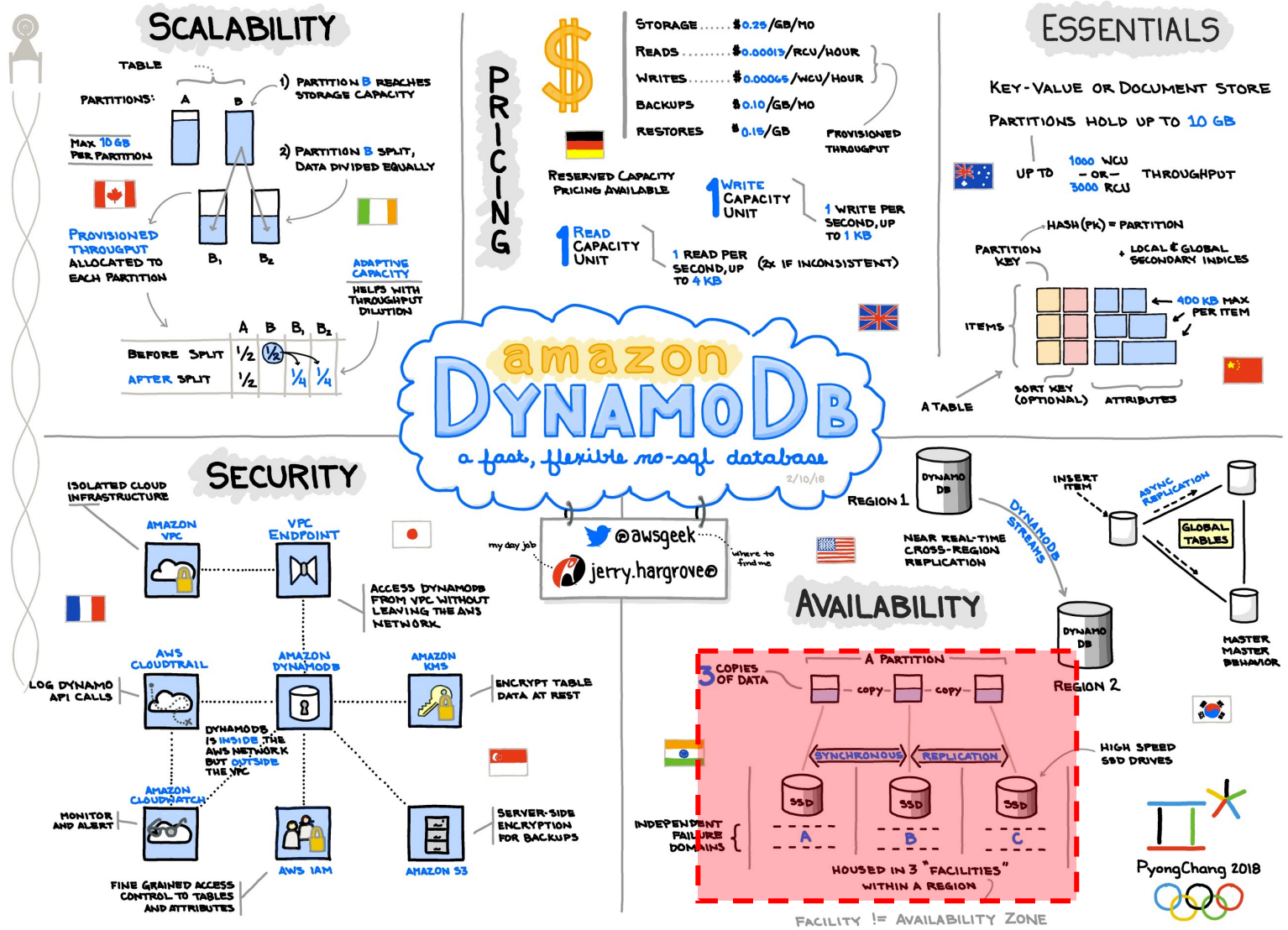
Google Bigtable



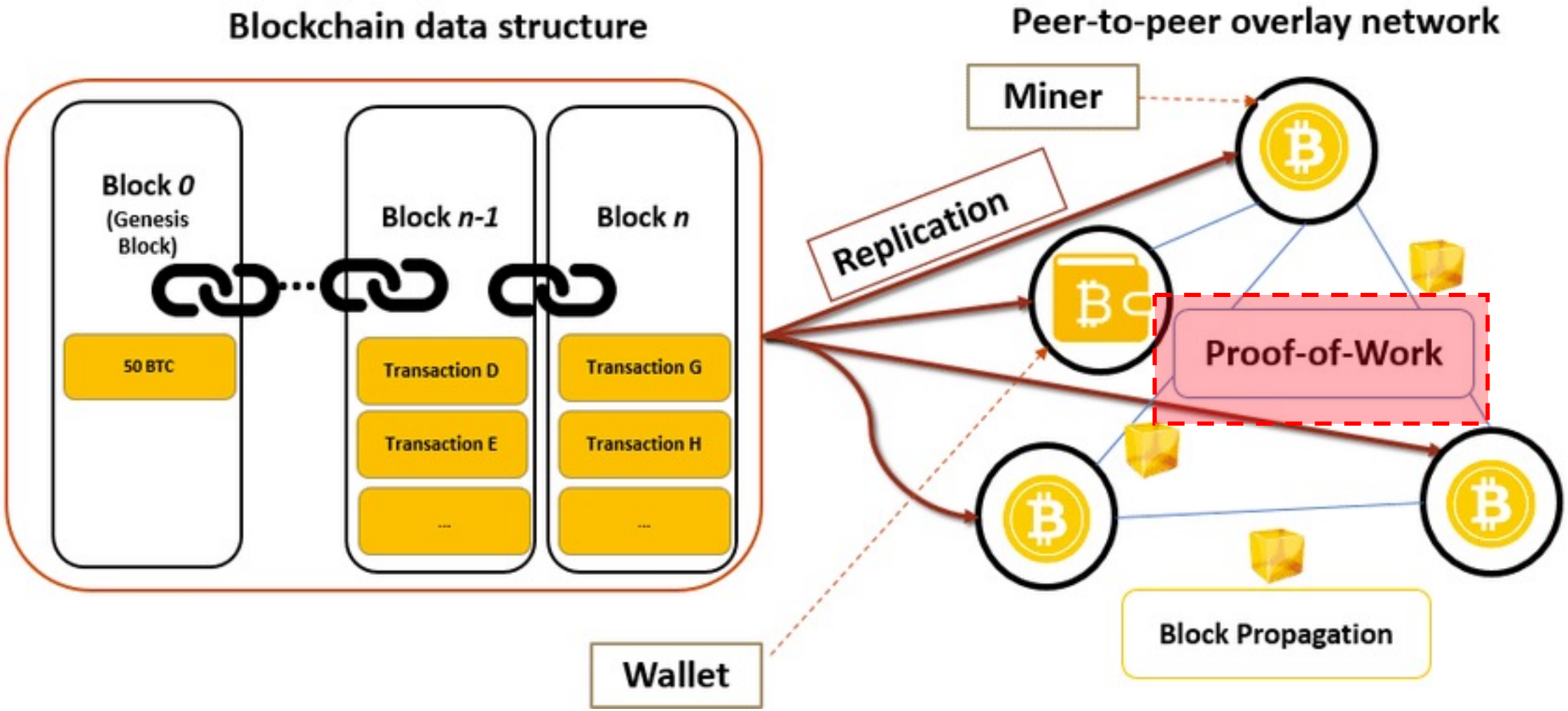
Google Spanner



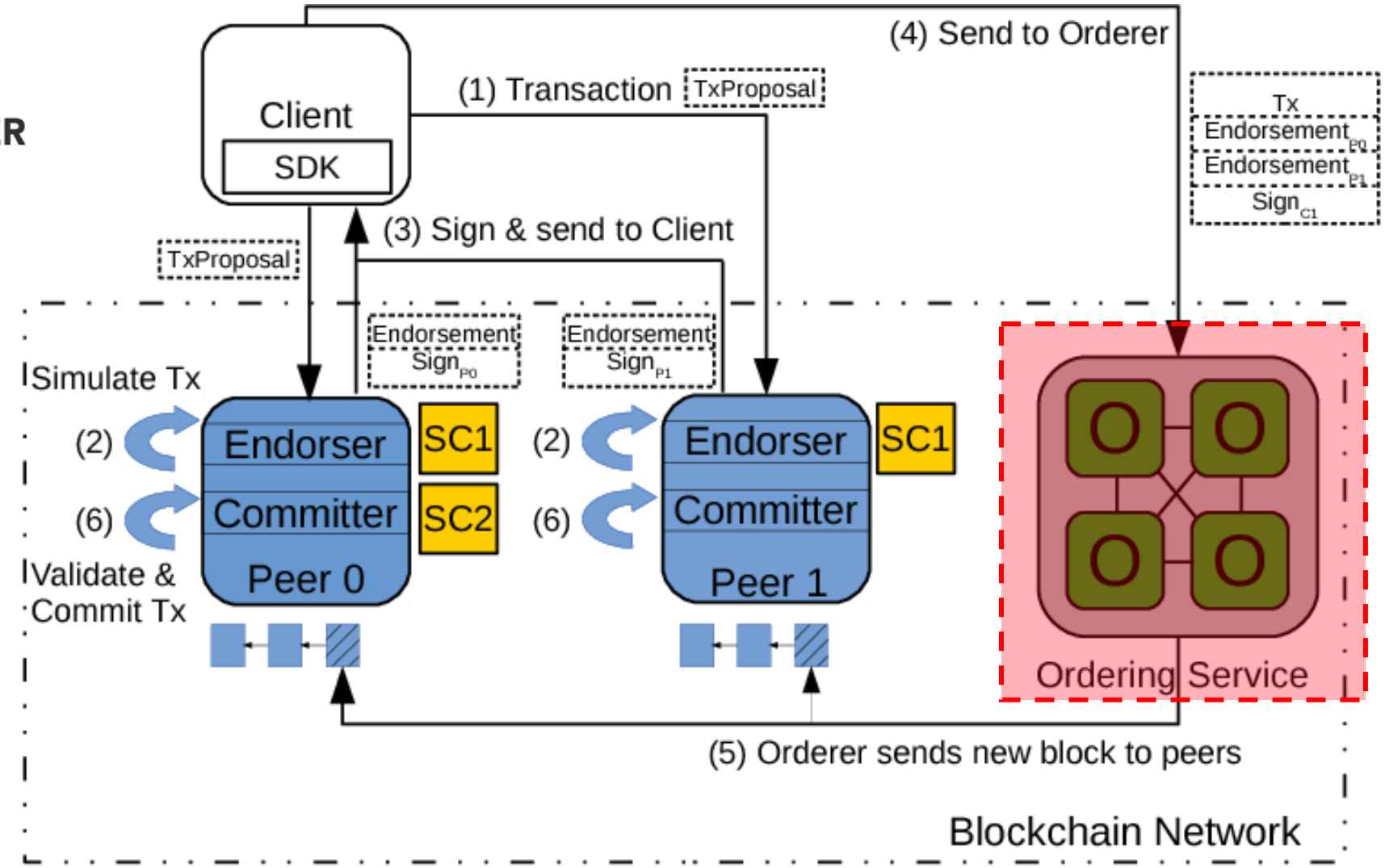
Amazon DynamoDB



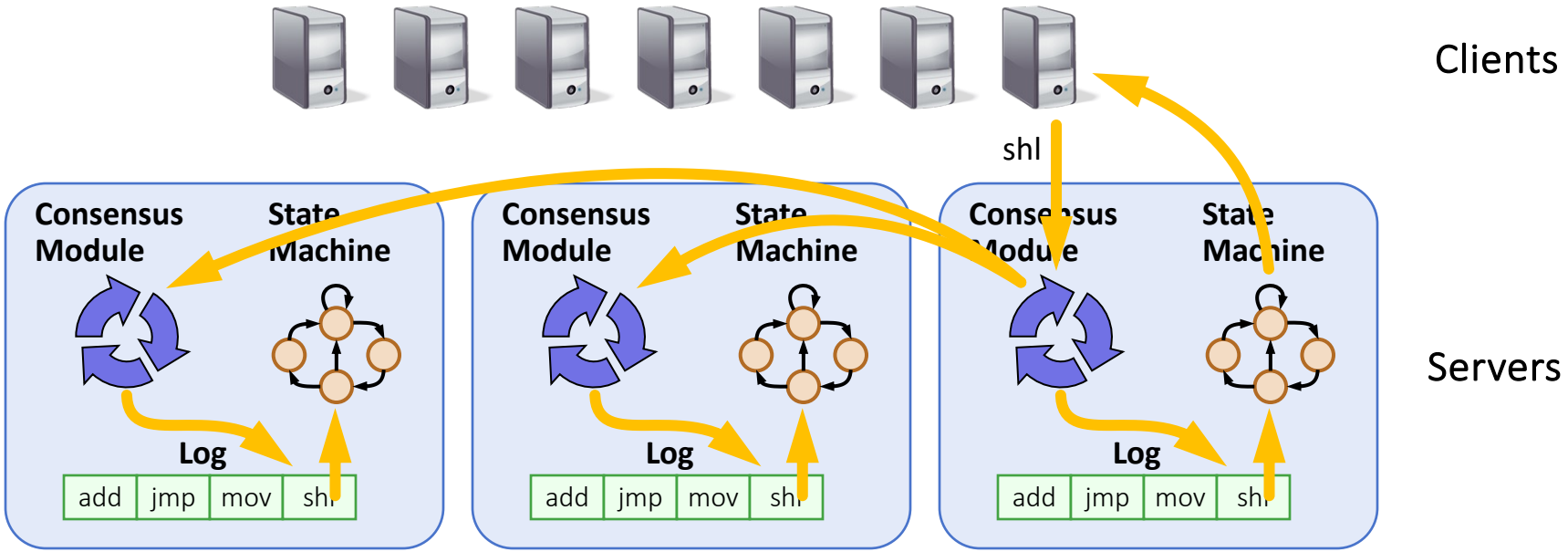
Bitcoin (permissionless blockchain)



Hyperledger Fabric (permissioned blockchain)



State machine replication

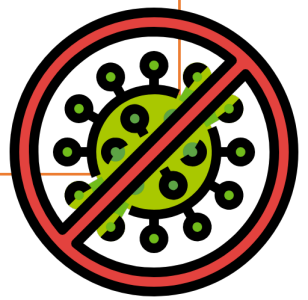


- Replicated log: replicated state machine
  - All servers execute same commands in the same order
  - Commands are deterministic
- Consensus module ensures proper log replication

SMR properties

Safety (bad things never happen)

- Only a value that has been proposed may be chosen
- Only a single value is chosen
- A node never learns that a value has been chosen unless it has been



Liveness (good things eventually happen)

- Some proposed value is eventually chosen
- If a value has been chosen, a node can eventually learn the value

When can consensus be guaranteed?

Determining factors:

- Processors: synchronous vs. asynchronous
- Communication: bounded vs. unbounded
- Messages: ordered vs. unordered
- Transmission: broadcast vs. point-to-point

Processors: synchronous or asynchronous

Synchronous:

- If and only if there exists a constant s >= 1 such that for every s + 1 steps taken by any processor, every other processor will have taken at least one step.

Communication delay: hounded or unbounded

Bounded communication delay:

- If and only if every message sent by a processor arrives at its destination within t real-time steps, for some predetermined t.

Messages: ordered or unordered

Ordered Messages:

- If and only if process P_r receives message m_1 before message m_2 when
  - P_1 sends m_1, to P_r at real time t_1
  - P_2 sends m_2 to P_r at real time t_2, and
  - $t_1 < t_2$.

Transmission mechanism

Point-to-point:

- If a processor can send a message in an atomic step to at most one other processor

Broadcast:

- If a processor can send a message to all the processors in one step.

Conditions under which consensus is possible

Processors	Message Order: Unordered		Message Order: Ordered		Communication
	Transmission: Point-to-point	Transmission: Broadcast	Transmission: Broadcast	Transmission: Point-to-point	
Asynchronous	No	No	Yes	No	Unbounded
	No	No	Yes	No	Bounded
Synchronous	Yes	Yes	Yes	Yes	
	No	No	Yes	Yes	Unbounded

Asynchronous system (realistic): mostly no.

Synchronous system (unrealistic): yes.

John Turek, Dennis Shasha. The many faces of consensus in distributed systems. IEEE Computer, 1992.

Synchronous and asynchronous systems

Synchronous System

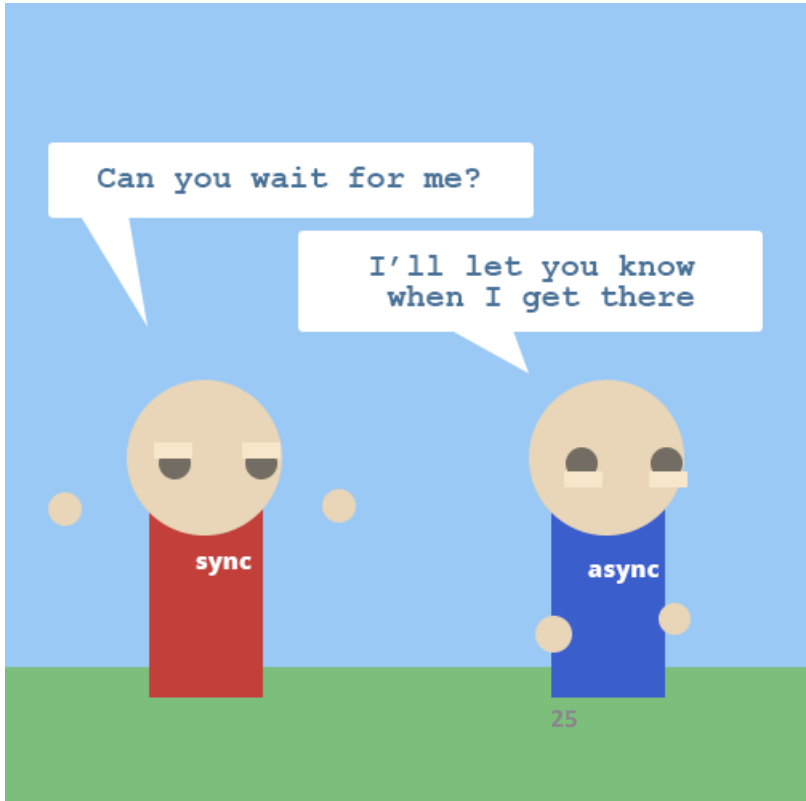
Known bounds on times for message transmission, processing, bounds on local clock drifts, etc.

- Can use timeouts (to predict failures)
- After a time threshold, the message will never be received

Asynchronous System

No known bounds on times for message transmission, processing, bounds on local clock drifts, etc.

- More realistic, practical, but no timeout
- The message can be infinitely delayed and still received



The FLP result

No deterministic 1-crash-robust consensus algorithm exists with asynchronous communication

Impossibility of Distributed Consensus with One Faulty Process

MICHAEL J. FISCHER

Yale University, New Haven, Connecticut

NANCY A. LYNCH

Massachusetts Institute of Technology, Cambridge, Massachusetts

AND

MICHAEL S. PATERSON

University of Warwick, Coventry, England

Abstract. The consensus problem involves an asynchronous system of processes, some of which may be unreliable. The problem is for the reliable processes to agree on a binary value. In this paper, it is shown that every protocol for this problem has the possibility of nontermination, even with only one faulty process. By way of contrast, solutions are known for the synchronous case, the "Byzantine Generals" problem.

This work was originally presented at the 2nd ACM Symposium on Principles of Database Systems, March 1983.

Authors' present addresses: M. J. Fischer, Department of Computer Science, Yale University, P.O. Box 2158, Yale Station, New Haven, CT 06520; N. A. Lynch, Laboratory for Computer Science, Massachusetts Institute of Technology, 545 Technology Square, Cambridge, MA 02139; M. S. Paterson, Department of Computer Science, University of Warwick, Coventry CV4 7AL, England

Journal of the Association for Computing Machinery, Vol. 32, No. 2, April 1985, pp. 374–382.

The FLP result

- In an asynchronous system (unordered messages, unbounded communication delays, unbounded processing delays), no protocol can guarantee consensus within a finite amount of time if even a single process can fail by stopping.
  - Network: asynchronous (unordered messages, unbounded communication delays, unbounded processing delays)
  - Failure model: crash failures (maximum ONE failure)
  - Consensus problem: all non faulty processes agree on the same value {0, 1}.

Intuition: in an asynchronous system where there are no bounds on the amount of time a processor might take to complete its work it's impossible to make the distinction between a process that is crashed and one that is taking a long time to respond.

How to circumvent FLP result?

Sacrifice determinism

Randomized Byzantine consensus algorithm

Adding synchrony assumption

Define bound on message delay, etc.

Adding oracle (failure detector)

Adding trusted component

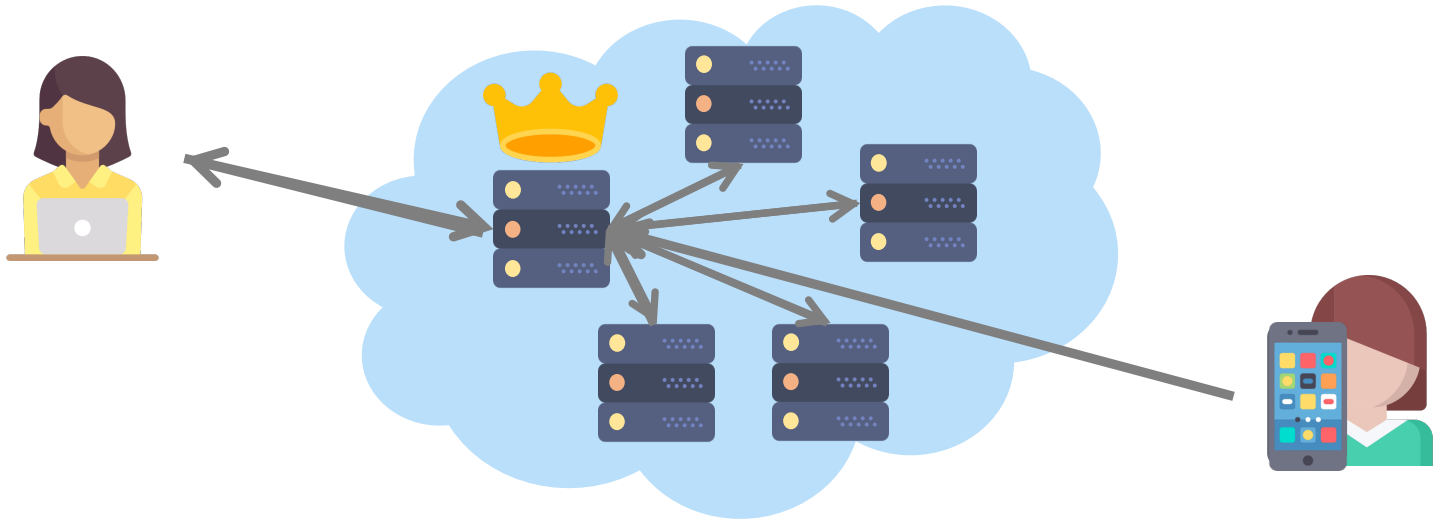
Change the problem domain

range of value or set of values

Correia, M., Veronese, G. S., Neves, N. F., & Verissimo, P. Byzantine consensus in asynchronous message-passing systems: a survey. IJCCBS, 2011

Paxos consensus algorithm

- The clients send updates to the leader
- Leader orders the requests and ‘forwards’ to the replicas
- Leader waits to get acknowledgement of the updates
- Upon receiving ‘enough’ acks, leader sends decision asynchronously



Lamport, L. Paxos made simple. ACM Sigact News, 2001

Basic Paxos skeleton

Phase 1a: **"Prepare"**
Select proposal number *N* and send a ***prepare(N)*** request to all acceptors.

Proposer

Phase 1b: **"Promise"**
If N > *number of any previous promises or acceptances*,
- send a ***ack(N)*** response

Phase 2a: **"Accept!"**
If proposer received ack responses from a majority,
- send an ***accept(N)*** request to all acceptors

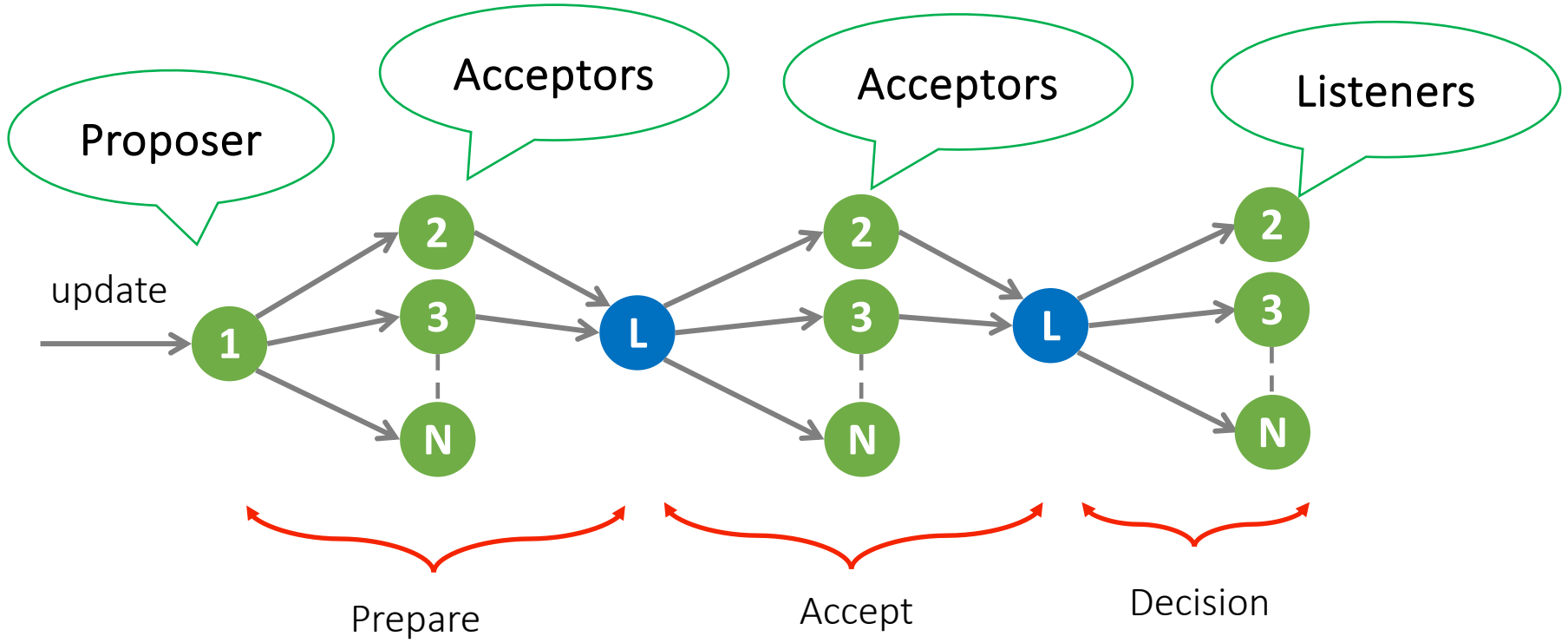
Acceptor

Phase 2b: **"Accepted"**
If N >= *number of any previous promise*,
accept the proposal

The leader announces Decision to all once majority **Accepted** the value

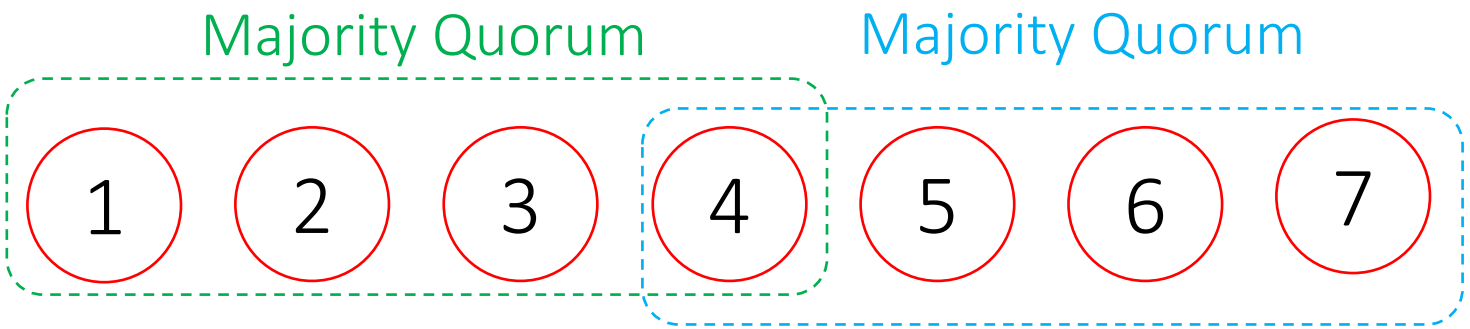
Paxos consensus algorithm

- **Leader Election:** Initially, a leader is elected by a quorum of servers
- **Replication:** Leader replicates new updates on quorum of servers
- **Decision:** Propagates decision to all asynchronously



Safety condition

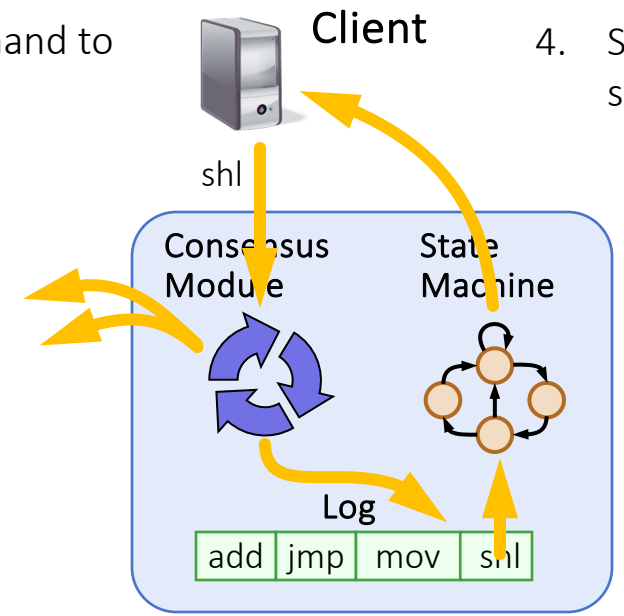
- Any two sets (quorums) of acceptors must have at least one overlapping acceptor
- This way a new leader will know of a value chosen by old leader through the overlapping acceptor



Multi-Paxos

- Separate instance of Basic Paxos for each log entry:
  - Add index argument to Prepare and Accept (selects entry in log)

1. Client sends command to server



4. Server returns result from state machine to client

2. Server uses Paxos to choose command as value for a log entry

3. Server waits for previous log entries to be applied, then applies new command to state machine

Raft

- Equivalent to Paxos in fault-tolerance
- Meant to be more understandable
- Uses a leader approach
- Integrates consensus with log management

Cockroach DB

HYPERLEDGER FABRIC

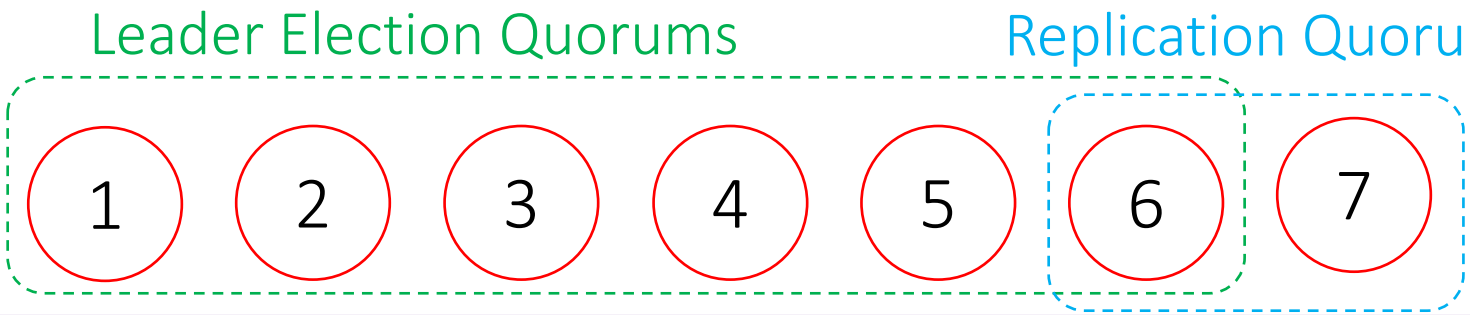
r3 corda

Quorum

Ongaro, D., & Ousterhout, J. In search of an understandable consensus algorithm. USENIX ATC, 2014

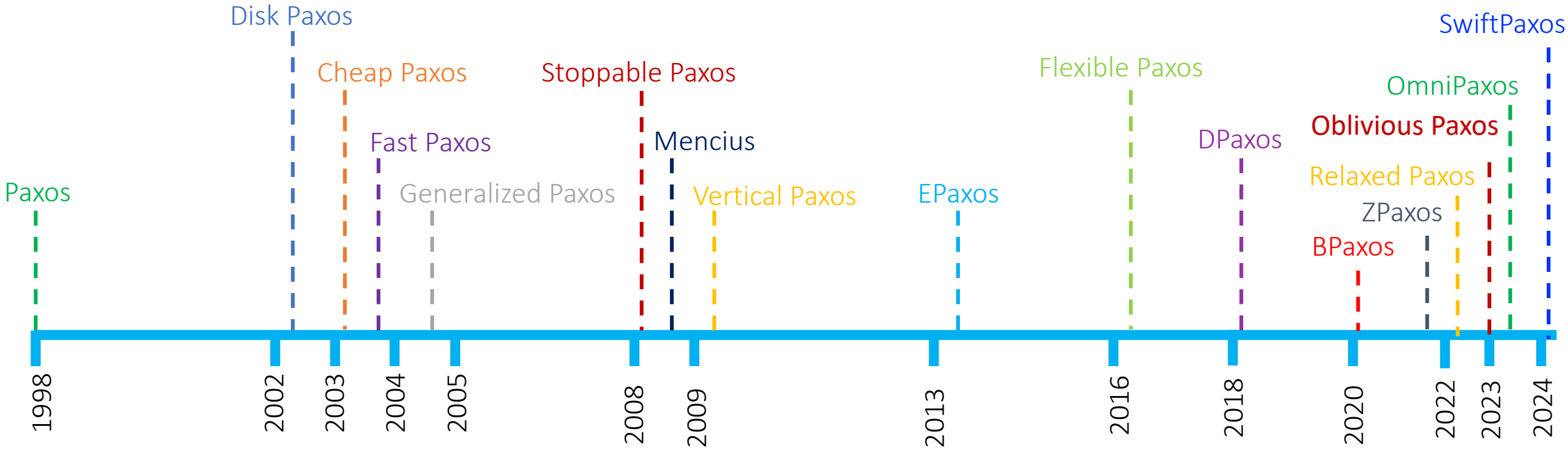
Flexible Paxos

- Majority quorums for BOTH Leader Election AND Replication are too conservative
- It is not necessary to require all quorums in Paxos to intersect
- Generalized Quorum Condition: only Leader Election Quorums and Replication Quorums must intersect.
  - Decouple Leader Election Quorums from Replication Quorums
  - Arbitrarily small replication quorums as long as Leader Election Quorums intersect with every Replication Quorum
- No changes to Paxos algorithms



Howard, H., Malkhi, D., & Spiegelman, A. Flexible Paxos: Quorum Intersection Revisited. OPODIS 2017

Paxos variants



Paxos in real systems

Google MegaStore

Clustrix

ceph

Google Cloud Spanner

Bing

OpenReplica

Doozerd

Google Chubby

wanDISCO

neo4j

XTREEMFS


What if nodes behave maliciously?!

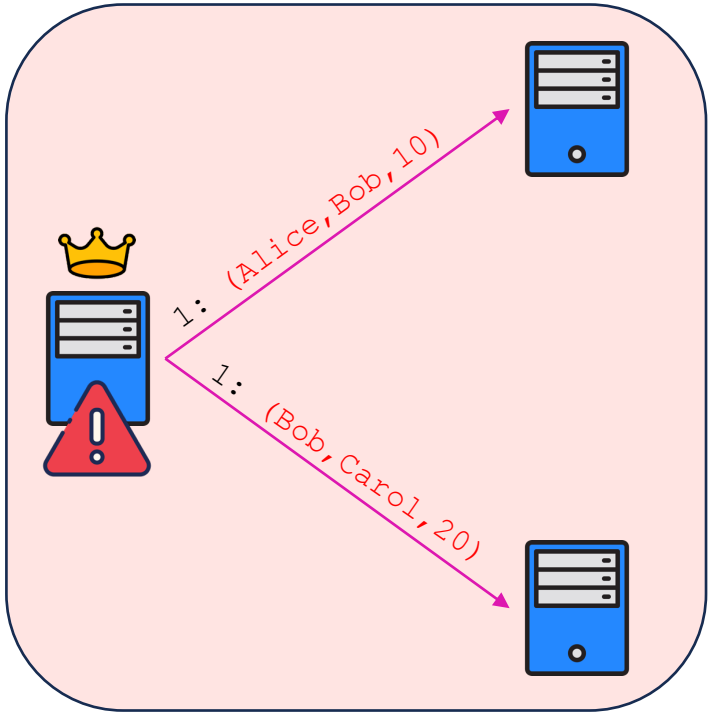
Byzantine faults

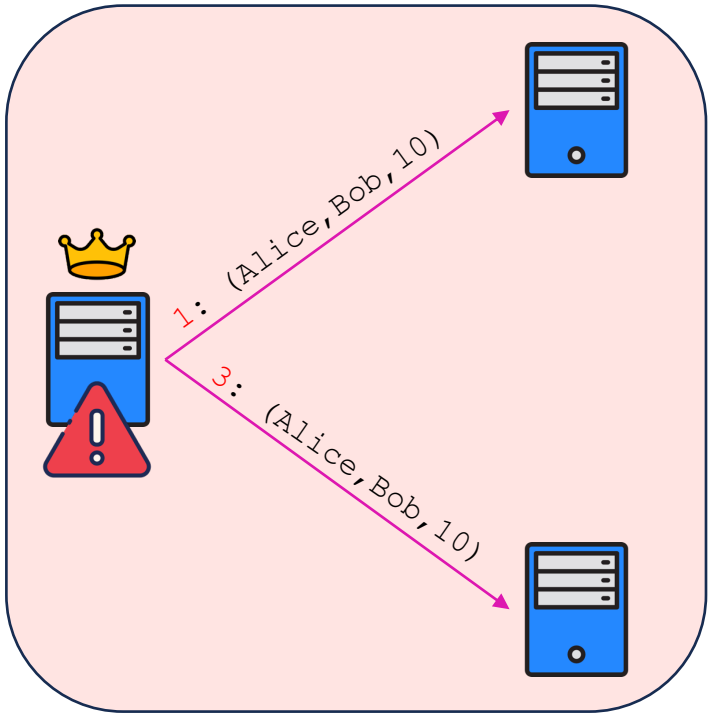
- Nodes can fail arbitrarily, including deviate from the protocol
  - may perform incorrect computation
  - may give conflicting information to different parts of the system
  - may collude with other failed nodes

- Potential causes:
  - software bugs
  - hardware failures
  - malicious attacks

- Can we provide state machine replication for a service in the presence of Byzantine faults?

Consensus in the presence of malicious nodes





Byzantine failure: exhibit arbitrary, potentially malicious, behavior

Reaching agreement in the presence of faults

- In a system with f faulty processes, an agreement can be achieved only if 2f+1 correctly functioning processes are present, for a total of 3f+1.
- i.e., An agreement is possible only if more than two-thirds of the processes are working properly.
- Model:
  - Processes are synchronous
  - Messages are unicast while preserving ordering
  - Communication delay is bounded
  - There are *N* processes, where each process *i* will provide a value v_i to the others
  - There are at most *f* faulty processes
  - It requires f+1 rounds of communication

Pease, Marshall, Robert Shostak, and Leslie Lamport. "Reaching agreement in the presence of faults, JACM, 1980

Practical Byzantine fault tolerance

1. Asynchronous
2. Authentication: messages are authenticated (a.k.a., signed) such that everyone knows who sent them and can do the accounting of responses correctly.
3. Super-majority: majority = 2f+1 now because you need to know that at least f+1 non-faulty nodes have responded.
4. Broadcast: protocol works through broadcast because no individual node (such as a “leader”) can be trusted.
5. Three phases: There are three proper phases in addition to “termination”.

Castro, Miguel, and Barbara Liskov, Practical Byzantine fault tolerance, OSDI 1999 & TOCS 2002

Assumptions

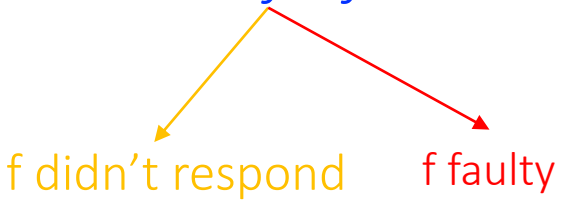
- Asynchronous distributed system where nodes are connected by a network.
- The network may fail to deliver messages, delay, duplicate or deliver them out of order.
- Byzantine failure model: faulty nodes may behave arbitrarily.
- Independent node failures.
- The adversary cannot delay correct nodes indefinitely and cannot subvert the cryptographic mechanisms
  - Public-key signatures.
  - Message authentication codes.
  - Message digest produced by collision-resistant hash functions.

PBFT main ideas

- Configuration
  - Use 3f+1 nodes
- To deal with malicious primary
  - Use a 3-phase protocol
- To deal with loss of agreement
  - Use a bigger quorum (2f+1 out of 3f+1 nodes)
- Need to authenticate communications
  - Message m signed by node i is denoted as $<m>\sigma_i$

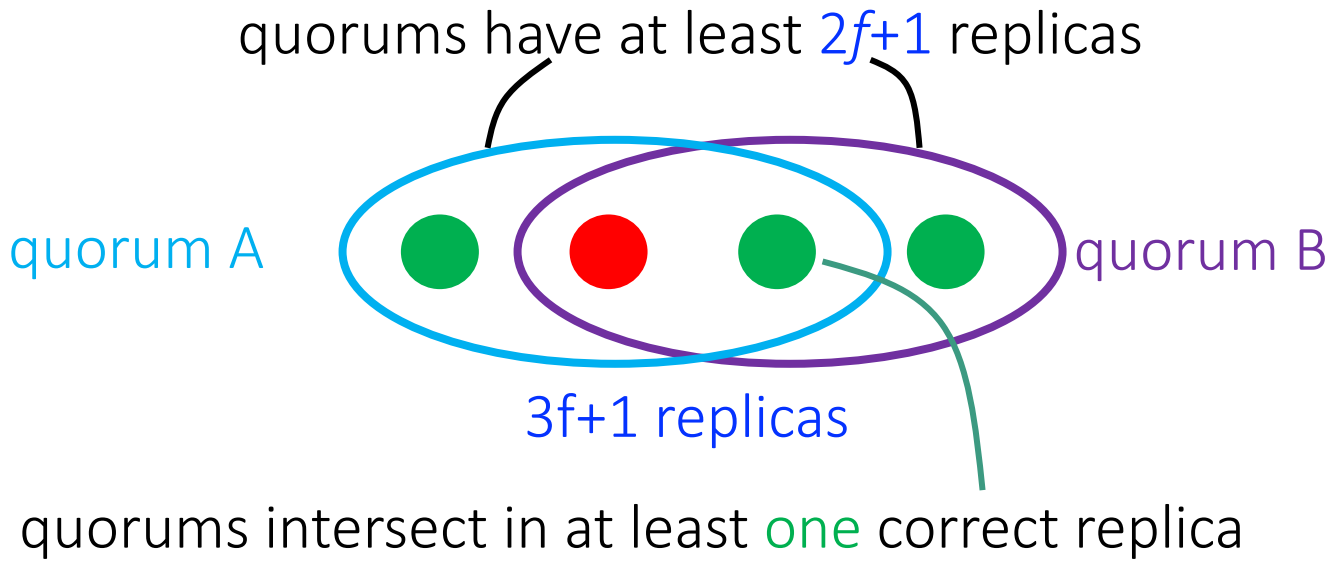
Network size

- $N > 3f$ Why?
- To make any progress must be able to tolerate f failures, i.e., must be able to make progress if only $n-f$ processes respond.
- BUT maybe the f that did not respond are not faulty, but slow (asynchronous systems), and among $n-f$ that responded f are faulty!
- Must have enough responses from non-faulty to outnumber faulty
  - $n-2f > f \Rightarrow n > 3f$



Quorum and network size

- $N > 3f$ Why? (Another Argument!)
- Any two Quorums of responses Q_1 and Q_2 need to intersect in at least f+1 nodes
  - $Q_1 + Q_2 > N + f$
  - $(N-f) + (N-f) > N + f \Rightarrow N > 3f$



Nodes

- Maintain a state
  - Log
  - View number
  - State (e.g., database)
- Can perform a set of operations
  - Need not be simple read/write
  - Must be deterministic
- Well behaved nodes must
  - Start at the same state
  - Execute requests in the same order

Views

- Operations occur within views
- For a given view, a particular node in is designated the primary node, and the others are backup nodes
- Primary = v mod n
  - n is number of nodes
  - v is the view number

PBFT protocol components

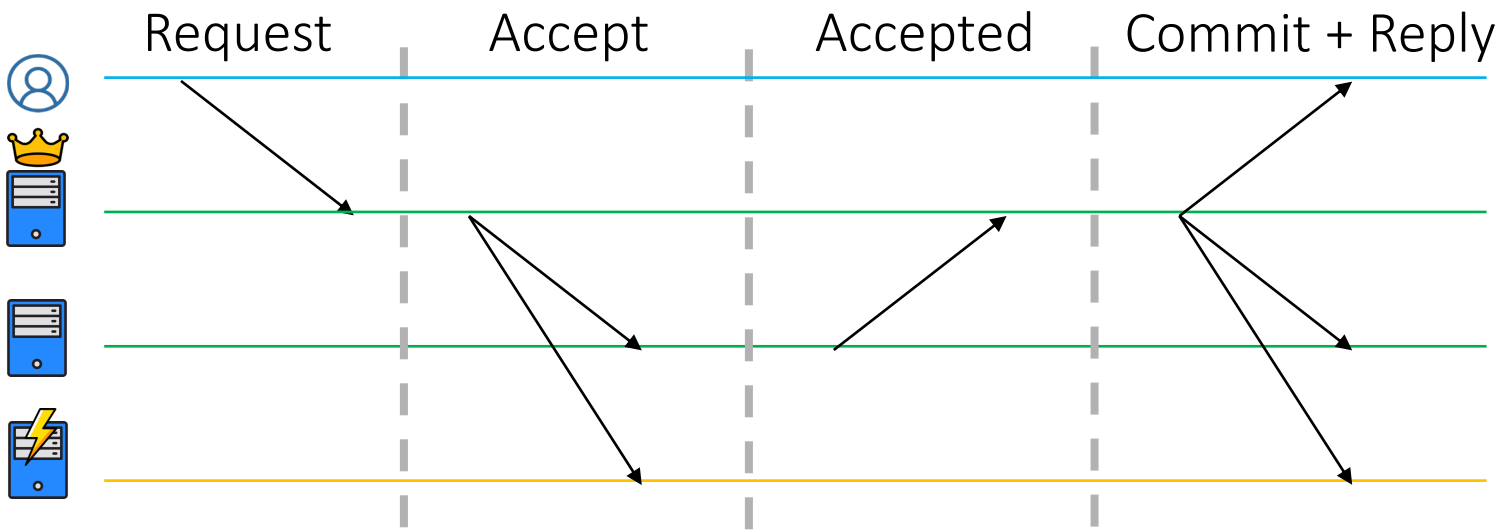
- Normal Case operation
  - order and execution
- View-change
  - Replace the current primary node
- Garbage collection
  - Checkpointing

PBFT normal case operation

- A client sends a request to invoke a service operation to the primary
- The primary multicasts the request to the backups
- Replicas reach agreement, execute the request and send a reply to the client
- The client waits for f +1 replies from different replicas with the same result; this is the result of the operation.

(Multi-)Paxos review

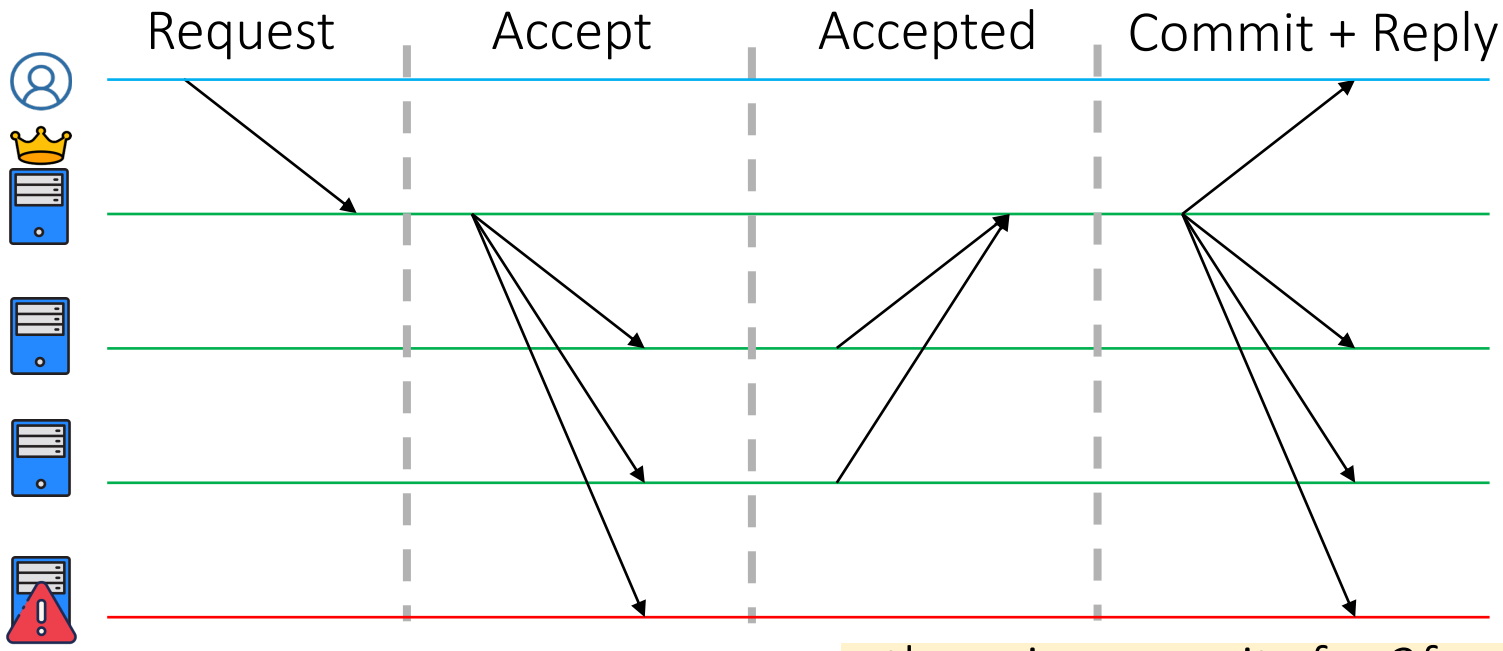
- Let's assume the leader is already elected



If the primary receives f matching Accepted messages (including itself f+1), it sends commit and reply messages

(Multi-)Paxos with malicious backups

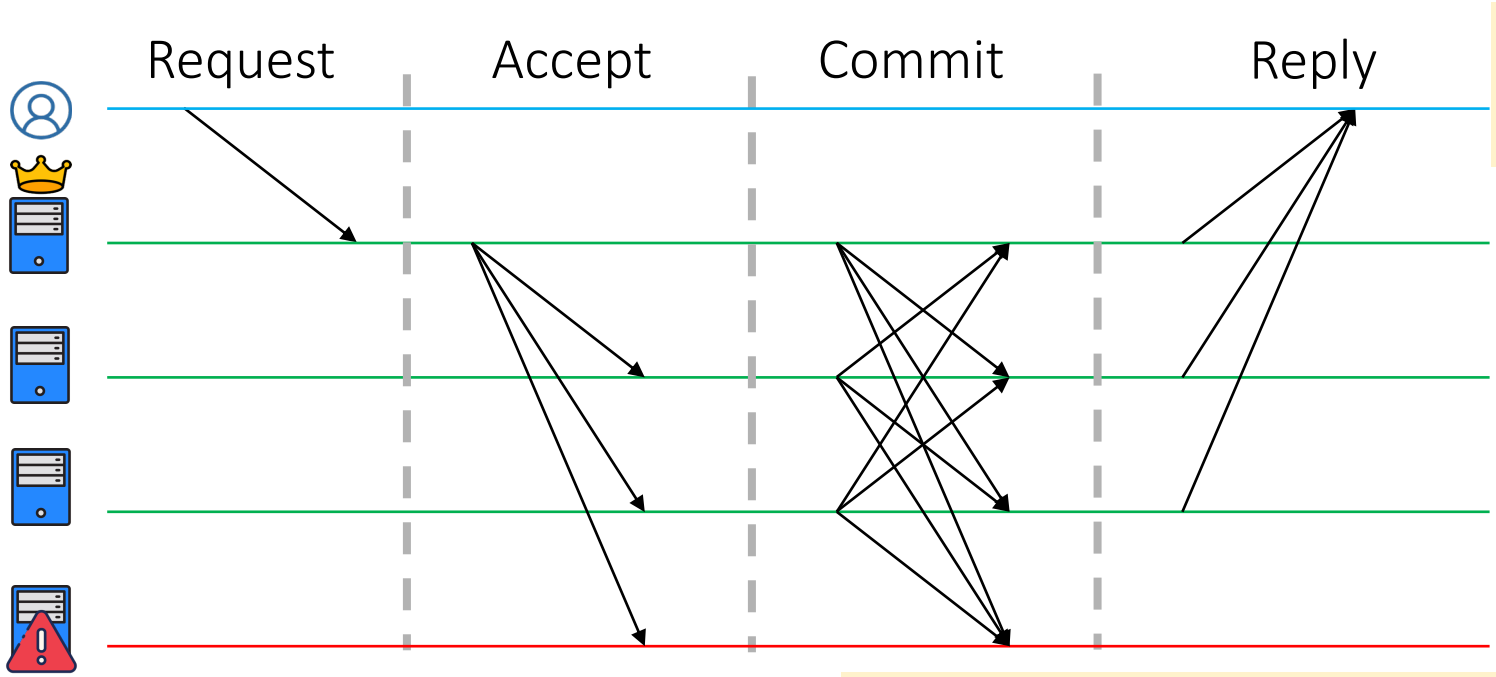
- What if f of the backups (not the primary) are malicious?!
  - 3f+1 nodes needed!



the primary waits for 2f matching Accepted messages (including itself 2f+1)

(Multi-)Paxos optimization

- Can nodes commit earlier?!



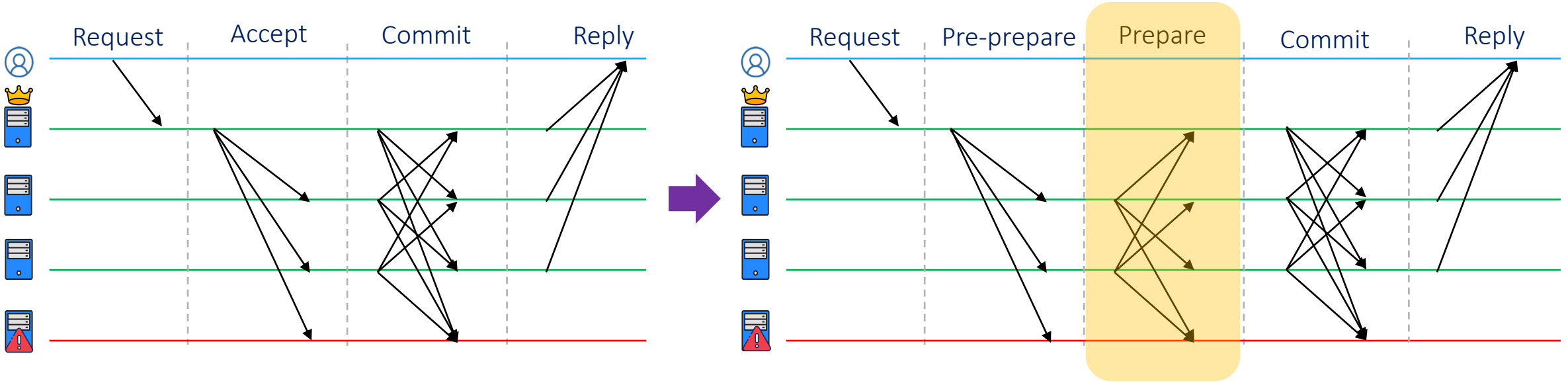
Client waits for f+1 matching Reply messages

If a replica receives 2f+1 matching Commit messages, it sends a Reply

What if the leader is malicious?

- When a replica receives an Accept message, it knows every replica receives the same message
- What if the leader is malicious?!
  - Sends invalid messages
  - Does not send the message to some replicas
  - Assigns different sequence numbers to the same request
  - Assigns the same sequence number to different requests
- One more phase of communication is needed
  - Majority of non-faulty nodes agree on the value that they received from the leader

From (Multi-)Paxos to PBFT



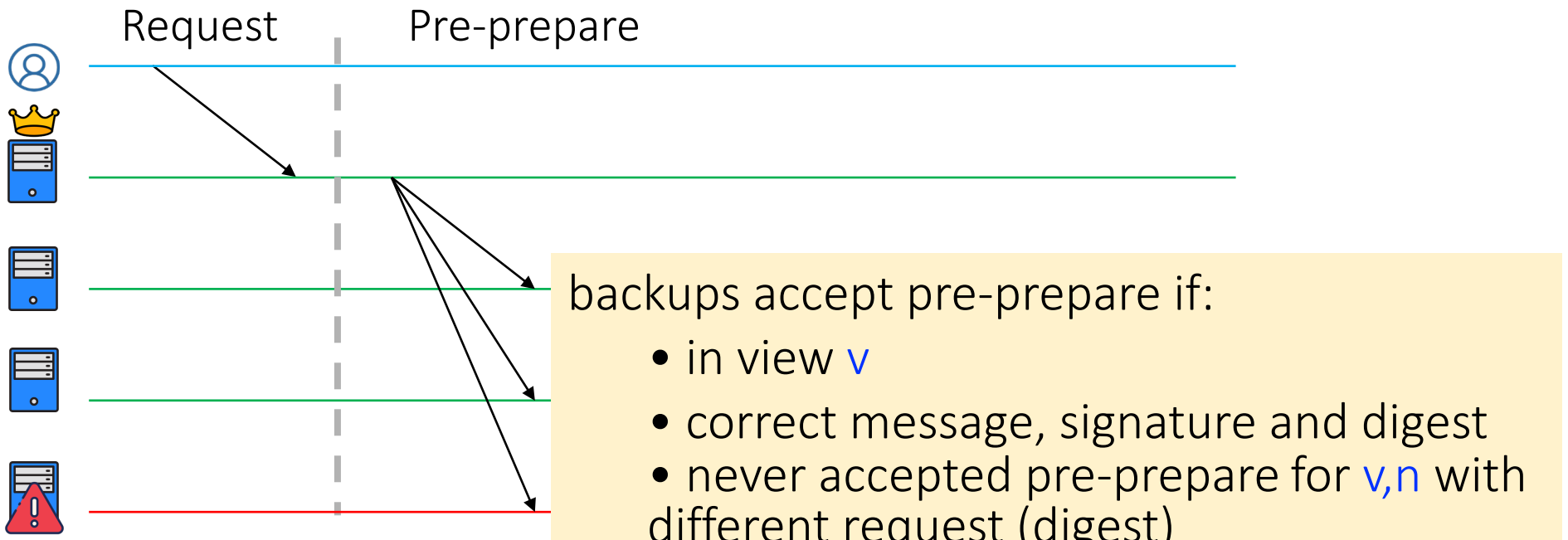
Normal case operation

- The algorithm has three phases:
  - pre-prepare picks order of requests
  - prepare ensures order within views
  - commit ensures order across views
- A replica executes a request m if
  - m is committed
  - all requests with sequence number less than n have been executed
- Replicas send a reply to the client
  - Client waits for f +1 matching replies

Pre-prepare phase

- The primary node p assigns sequence number n to request m in view v
  - The primary multicasts <PRE-PREPARE,v,n,d>σ_p,m>

d is the digest of m

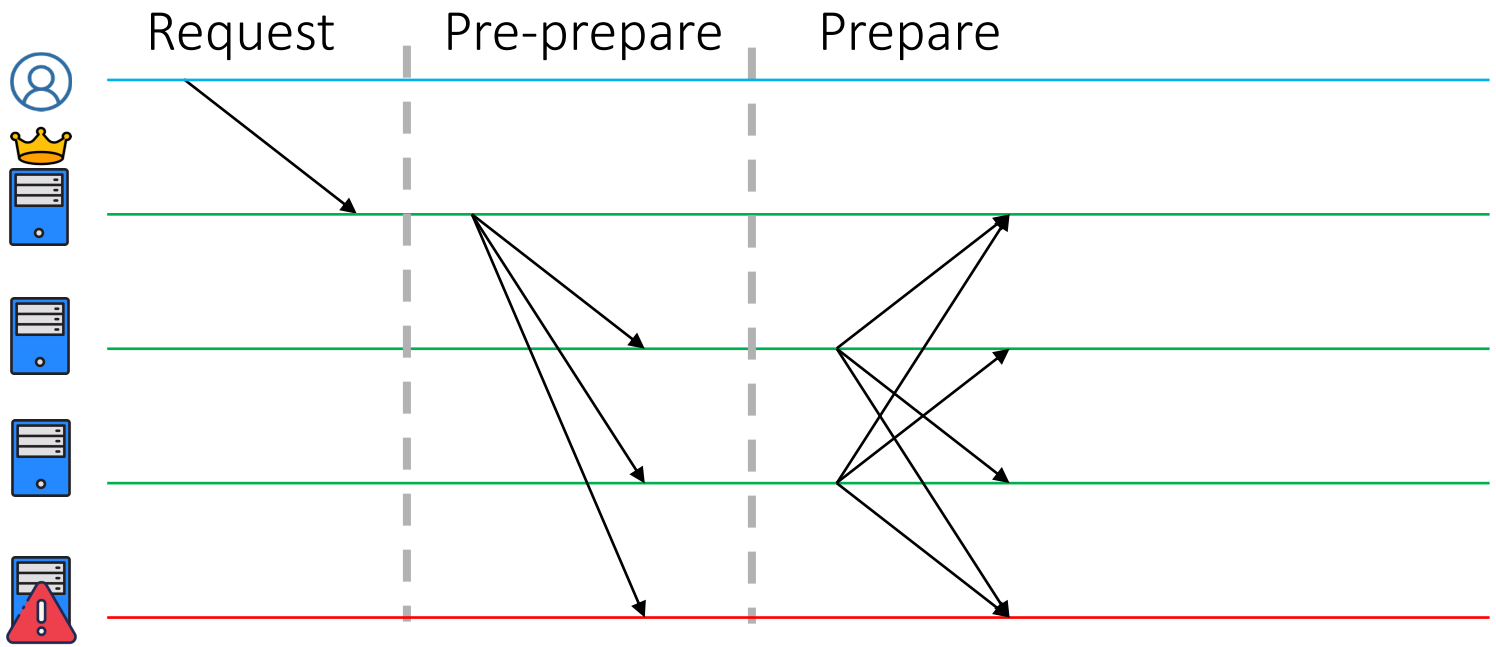


backups accept pre-prepare if:

- in view v
- correct message, signature and digest
- never accepted pre-prepare for v,n with different request (digest)

Prepare phase

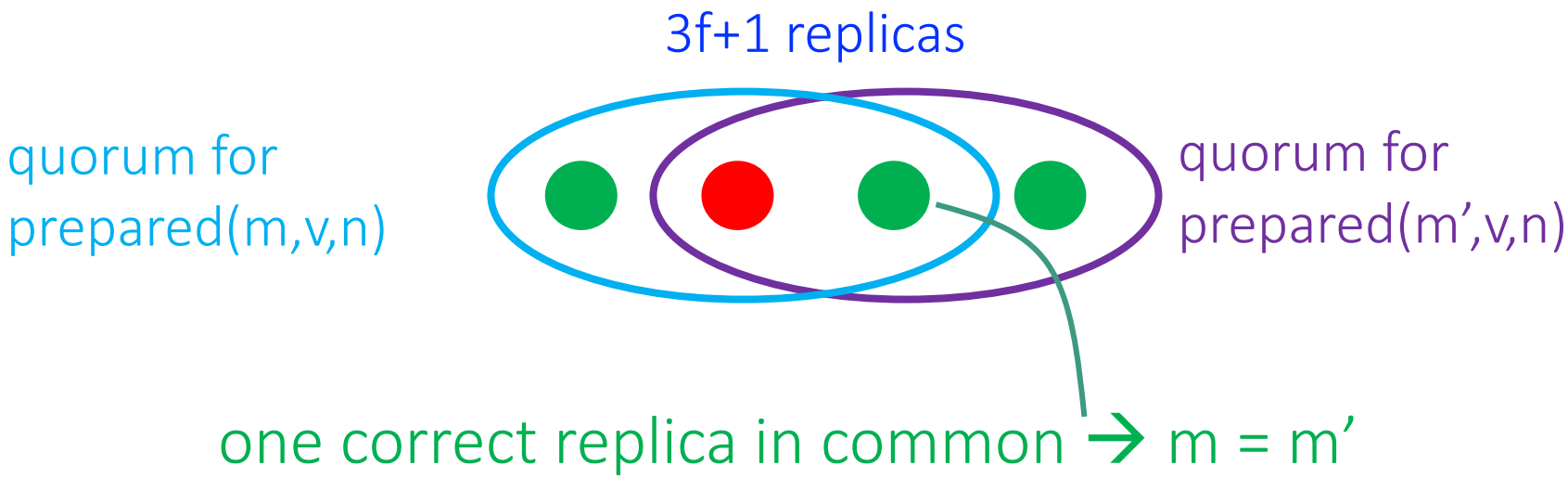
- Upon accepting a pre-prepare message
  - each backup replica i multicasts <PREPARE,v,n,d,i>σ_i



prepared(m,v,n,i) is true if replica i has corresponding pre-prepare in log and has received at least 2f matching prepares (possibly including its own) that match the pre-prepare

Order within view

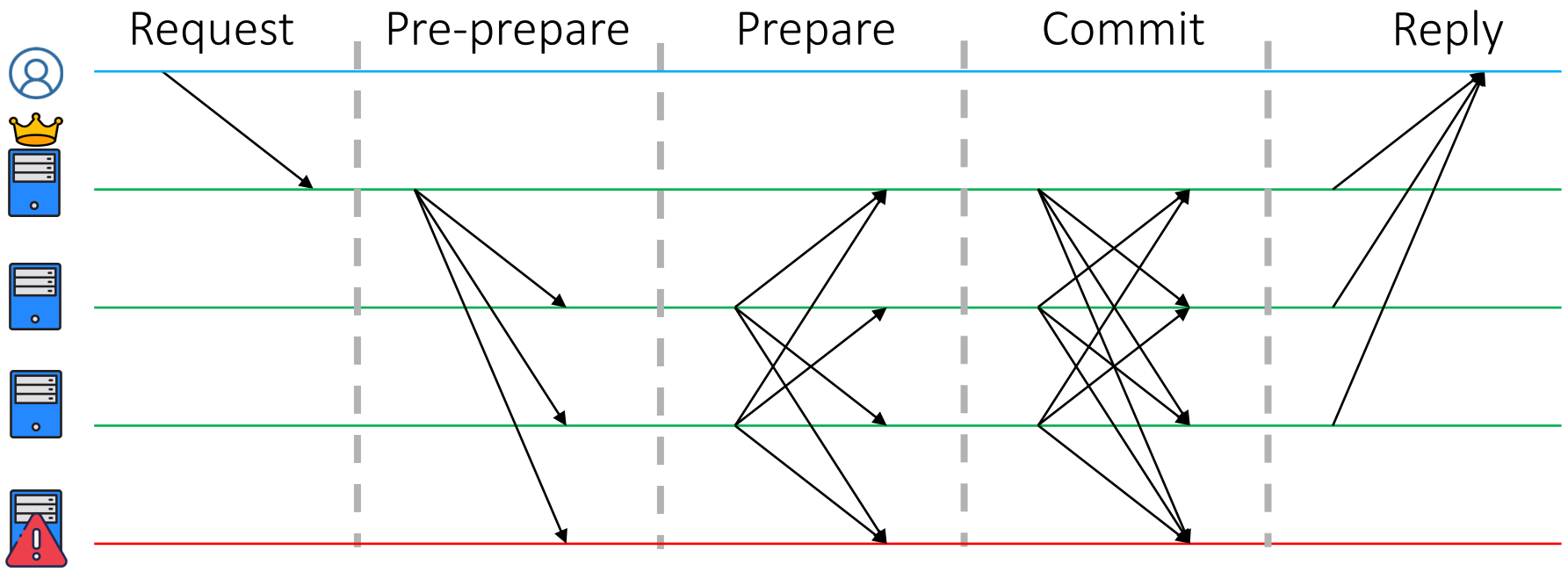
if prepared(m,v,n) is true then prepared(m’,v,n) is false for any m’



Commit phase

If replica i has prepared(m,v,n,i), it multicasts <COMMIT,v,n,D(m),i>σ_i

Client waits for f+1 matching replies



committed(m,v,n,i) is true if prepared(m,v,n,i) is true and replica i has has received at least 2f+1 matching commit messages (possibly including its own) that match the pre-prepare message

Request execution

- A replica executes a request m if
  - committed(m,v,n) is true
  - executed all requests with sequence number less than n
- Each replica send a reply to client
- Client collects replies from replicas
  - Returns result that applies in at least f+1 replies.

View-change

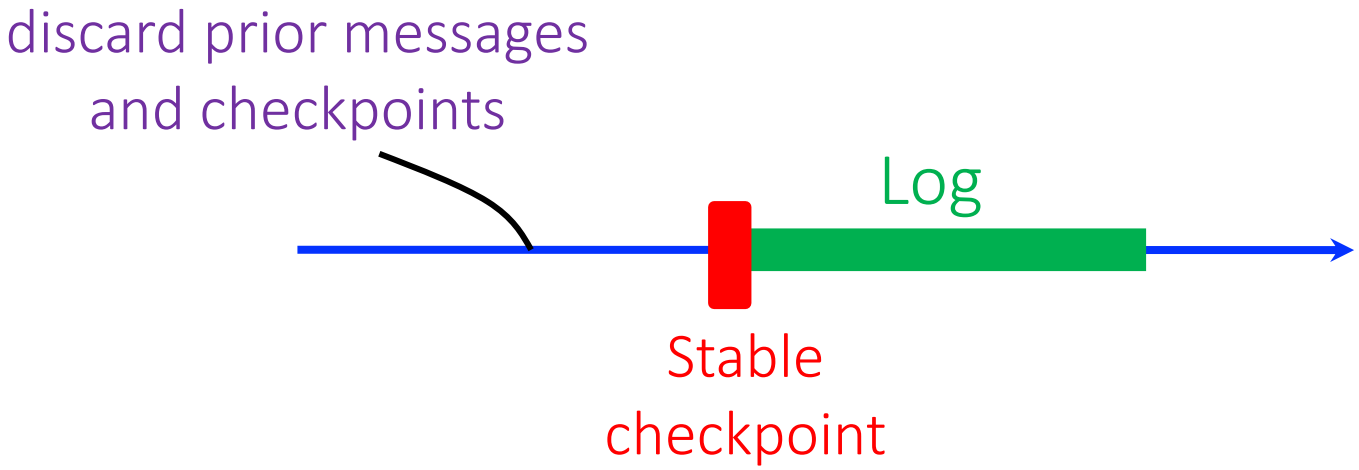
- Provide liveness when primary fails
  - timeouts trigger view-changes
  - select new primary ≡ view number mod n, where n is the number of replicas.

View-change

- Replicas watch the primary
- Request a view-change
  - send a view-change request to all
  - new primary requires 2f+1 view-change messages to accept new role
  - sends new-view with proof that it got the 2f+1 view-change messages

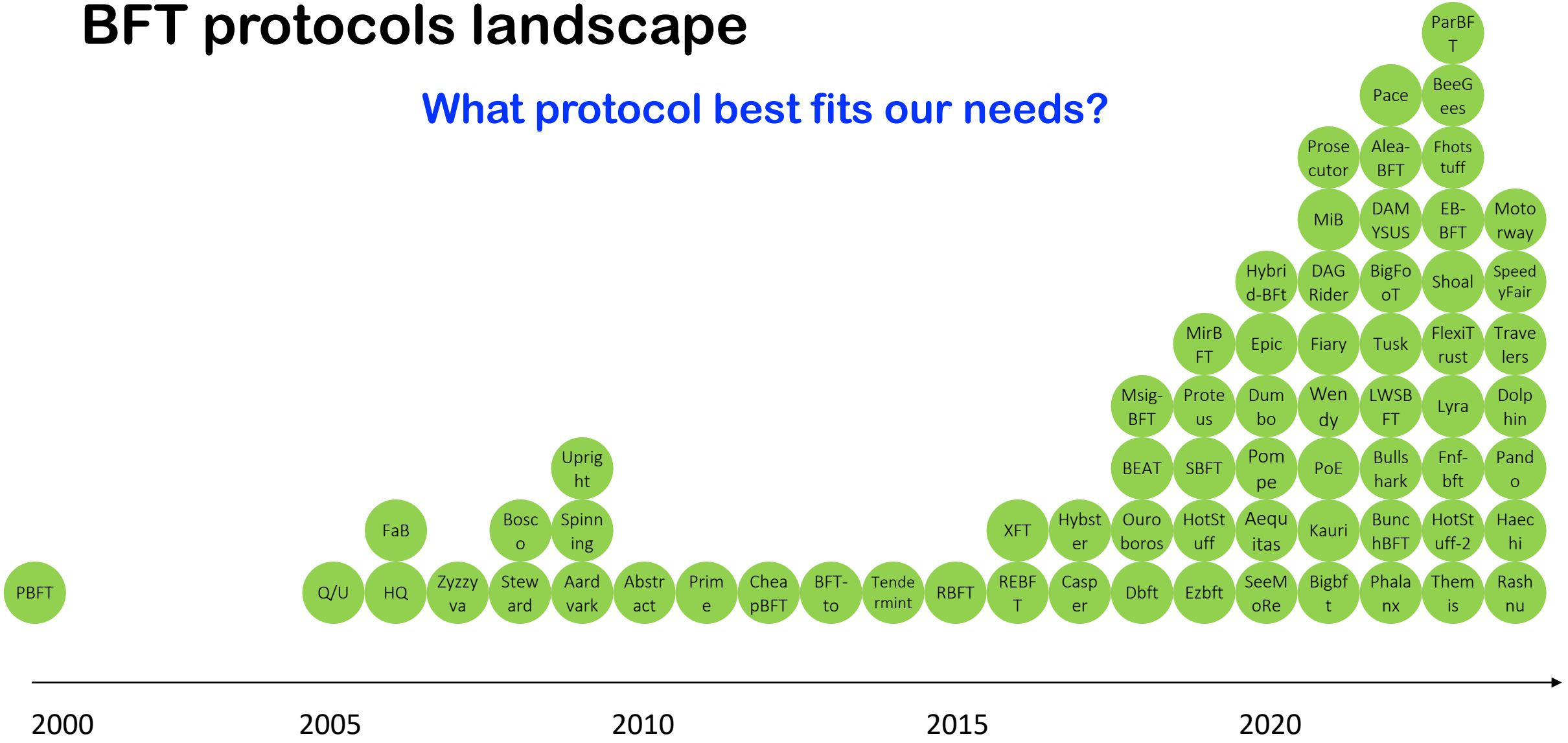
Garbage collection

- When to discard messages in the log?
  - periodically checkpoint the state by multicasting CHECKPOINT messages
  - Each node collects 2*f*+1 checkpoint messages: proof of correctness



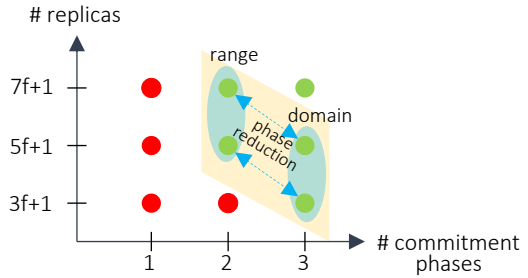
BFT protocols landscape

What protocol best fits our needs?



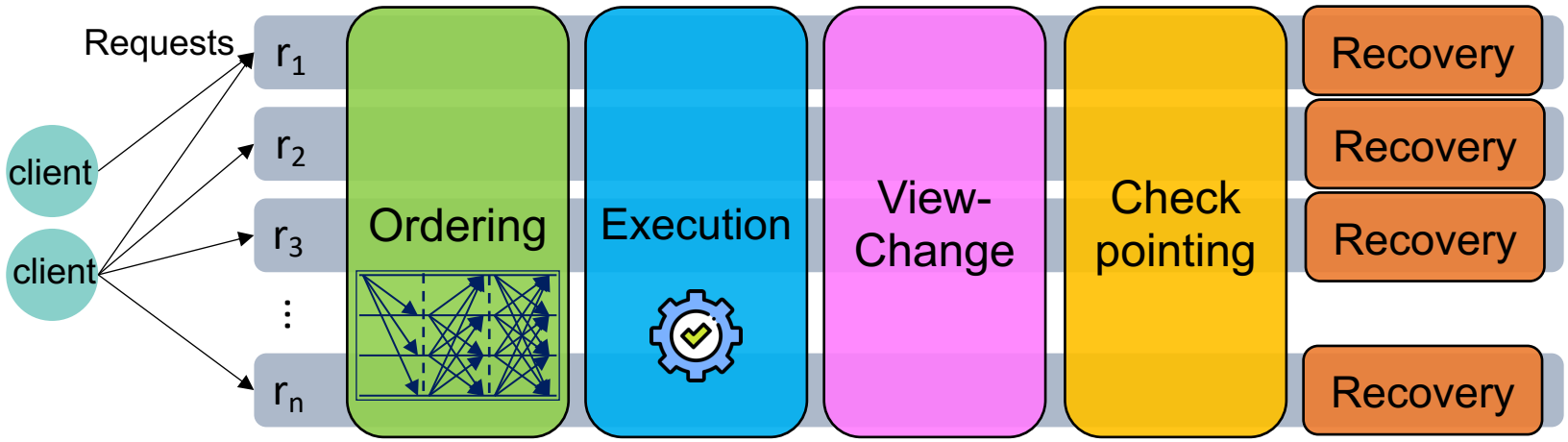
BFT protocols design space and design dimensions

- There are too many BFT protocols
- Design space
  - Consisting of a set of dimensions to analyze BFT protocols
- Design choices
  - Trade-offs between dimensions
  - A one-to-one function that maps protocols in its domain to protocols in its range

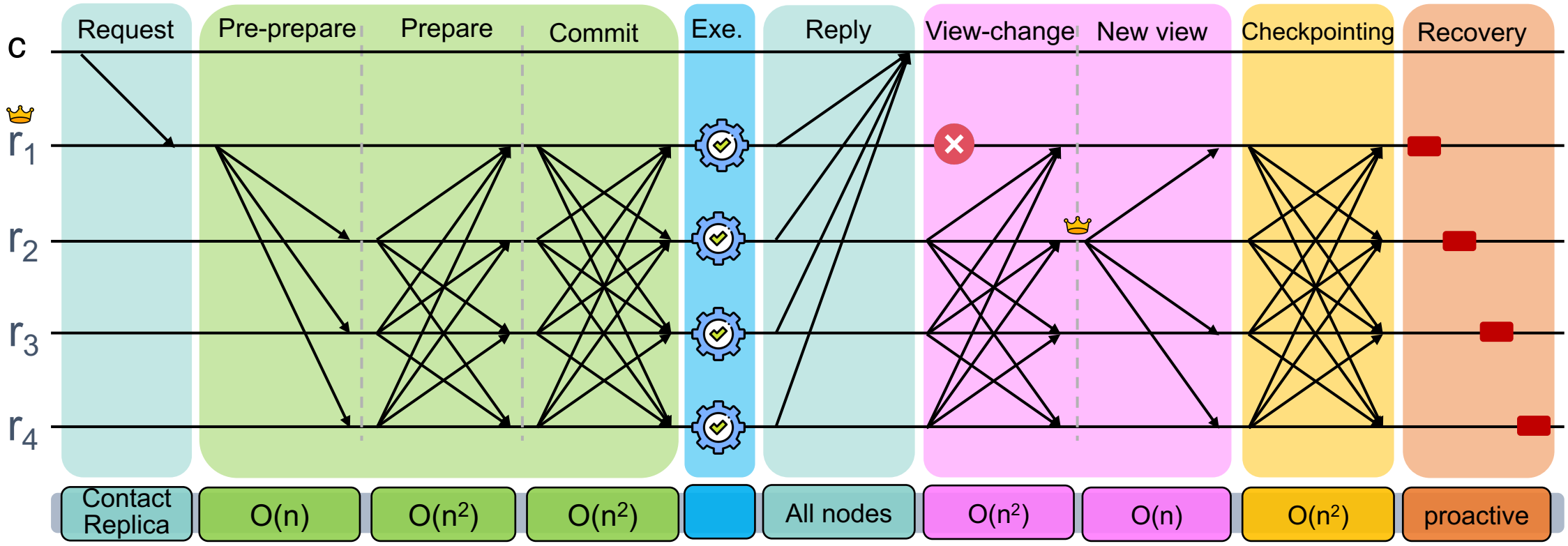


Amiri, M. J., Wu, C., Agrawal, D., El Abbadi, A., Loo, B. T., & Sadoghi, M. The Bedrock of Byzantine Fault Tolerance: A Unified Platform for BFT Protocol Analysis, Implementation and Experimentation, NSDI'24 **[Outstanding Paper Award]**

Different stages of replicas in a BFT protocol



PBFT


c
r1
r2
r3
r4
Request
Pre-prepare
Prepare
Commit
Exe.
Reply
View-change
New view
Checkpointing
Recovery
Contact Replica
O(n)
O(n²)
O(n²)
All nodes
O(n²)
O(n)
O(n²)
proactive

Design space of BFT protocols

Protocol structure

P1. Commitment strategy
P2. Number of commitment phases
P3. View-change
P4. Checkpointing
P5. Recovery
P6. Types of clients

Environmental Settings

E1. Number of replicas
E2. Communication topology
E3. Authentication
E4. Responsiveness, synchronization, and timers

Quality of Service

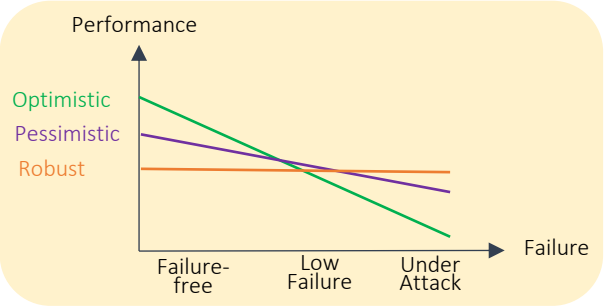
Q1. Order-fairness
Q2. Load balancing

Performance Optimization

O1. Out-of-order processing
O2. Request pipelining
O3. Parallel ordering
O4. Parallel execution
O5. Read-only requests processing
O6. Separating ordering and execution
O7. Trusted hardware
O8. Request/reply dissemination

Protocol structure

- P1. Commitment strategy
  - Optimistic
    - Assumptions:
      - a1. The leader is non-faulty, assigns a correct order to requests and sends it to all backups, e.g., Zyzzyva
      - a2. The backups are non-faulty and actively and honestly participate in the protocol, e.g., CheapBFT
      - a3. All non-leaf replicas in a tree topology are non-faulty, e.g., Kauri
      - a4. The workload is conflict-free and concurrent requests update disjoint sets of data objects, e.g., Q/U
      - a5. The clients are honest, e.g., Quorum, and
      - a6. The network is synchronous (in a time window), and messages are not lost or delayed, e.g., Tendermint
    - Speculative vs non-speculative
  - Pessimistic: make no assumption on failures, synchrony, or data contention
  - Robust: suitable for scenarios where the system is under attack
    - e.g., Prime, Aardvark, R-Aliph, Spinning, RBFT



Protocol structure (Cont.)

- P2. Number of commitment phases (good-case latency)
  - The number of phases needed for all non-faulty replicas to commit when the leader is non-faulty, and the network is synchronous (after GST)
- P3. View-change
  - Stable leader: replaces the leader when the leader is suspected to be faulty
  - Rotating leader: replace the leader
    - periodically, e.g., after a single attempt,
    - insufficient performance
    - an epoch (multiple requests)
- P4. Checkpointing
  - Garbage-collect data of completed consensus instances to save space
  - Restore in-dark replicas (due to network unreliability or leader maliciousness)
- P 5. Recovery
  - Reactive: detect faulty replica behavior and recover the replica by applying software rejuvenation techniques
  - Proactive: recover replicas in periodic time intervals

Protocol structure (Cont.)

- P6. Types of Clients
- Requester
  - Communicates with replicas by sending requests and receiving replies, e.g., PBFT
  - May need to verify the results by waiting for a number of matching replies
- Proposer
  - Proposes a sequence number (acting as the leader) for its request, e.g., Q/U
- Repairer
  - Detects the failure of replicas, e.g., Zyzzyva
  - Changes the protocol configuration, e.g., Scrooge

Environmental settings

- E1. Number of replicas
  - 3f+1
  - Trusted hardware: 2f+1
  - Active/passive replication: 2f+1(active)
  - 5f+1 (5f-1)
  - 7f+1
- E2. Communication topology
  - Star topology (linear message complexity)
  - Clique topology (quadratic message complexity)
  - Tree topology (logarithmic message complexity)
  - Chain topology (constant message complexity)
- E3. Authentication
  - Signatures, e.g., RSA
  - Authenticators, i.e., MACs
  - Threshold signatures

Environmental settings (Cont.)

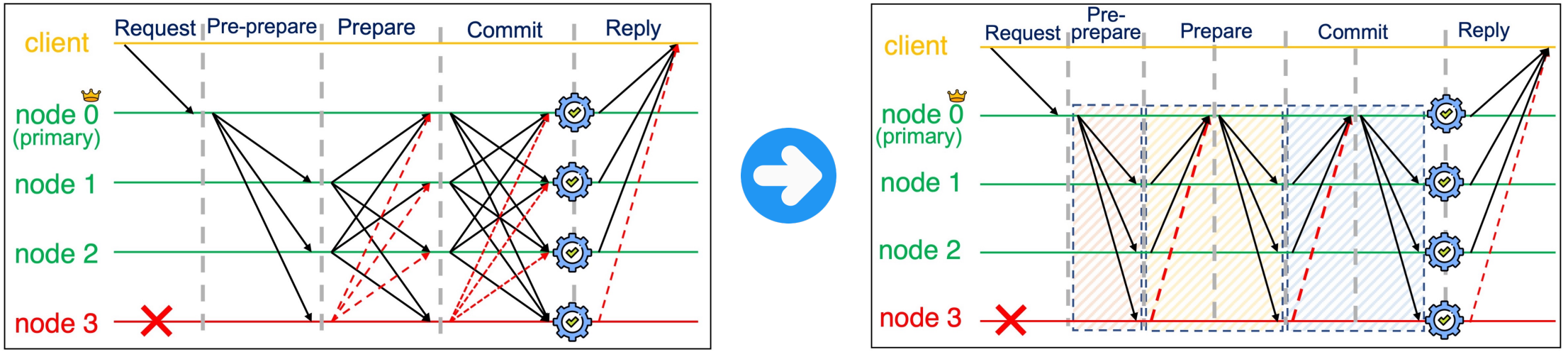
- E4. Responsiveness, Synchronization and Timers
  - Responsive: if its normal case commit latency depends only on the actual network delay rather than any predefined upper bound on message transmission delay
  - View synchronization: all non-faulty replicas need to eventually be synchronized to the same view with a non-faulty leader
  - Timers
    - τ1. Waiting for reply messages
    - τ2. Triggering (consecutive) view-change
    - τ3. Detecting backup failures
    - τ4. Quorum construction in an ordering phase
    - τ5. View synchronization
    - τ6. Finishing a (preordering) round
    - τ7. Performance check (heartbeat)
    - τ8. Atomic recovery (watchdog timer) to periodically hand control to a recovery monitor

Quality of service

- Q1. Order-fairness
  - Preventing adversarial manipulation of request ordering
- Q2. Load balancing
  - The performance is limited by the computing and bandwidth capacity of the leader

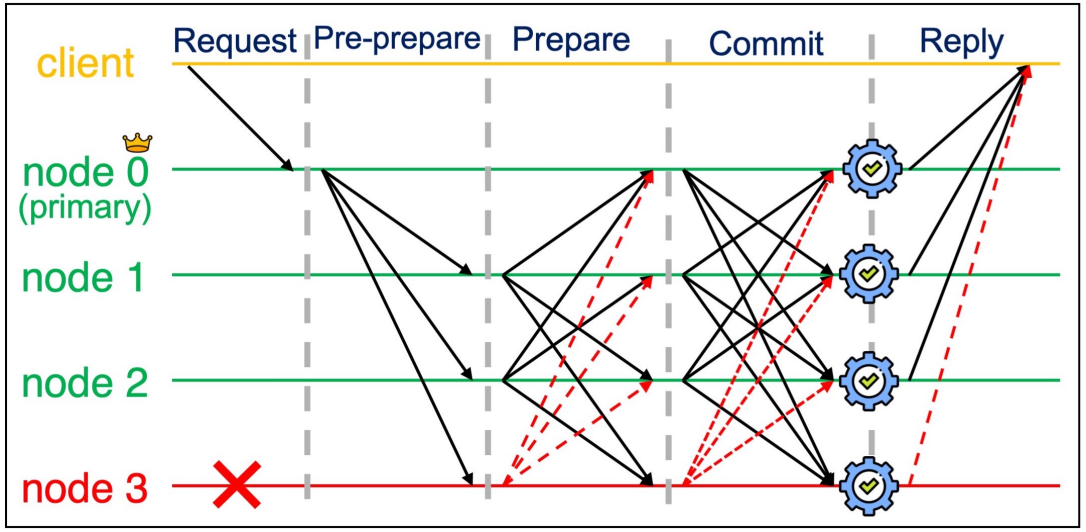
Design choice 1: Linearization

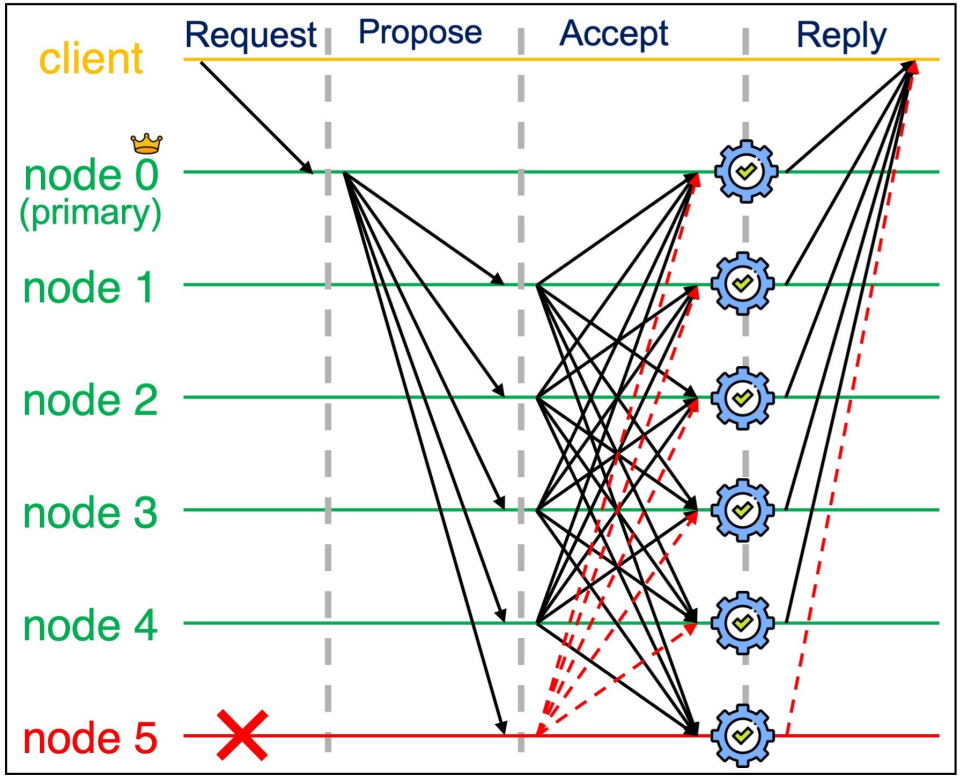
- Trade-off between communication topology and communication phases.
  - Linear PBFT
    - The collector needs to send a certificate of having received the required signatures.



Design choice 2: Phase reduction through redundancy

- Trade-off between the number of ordering phases and the number of replicas
  - FaB

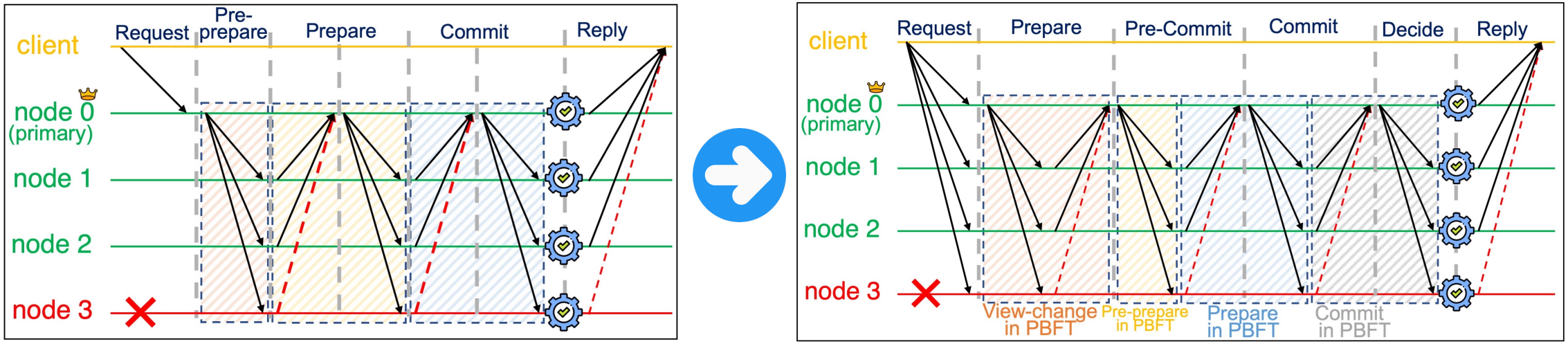




Martin, J.P. and Alvisi, L., Fast byzantine paxos, SDN 2004

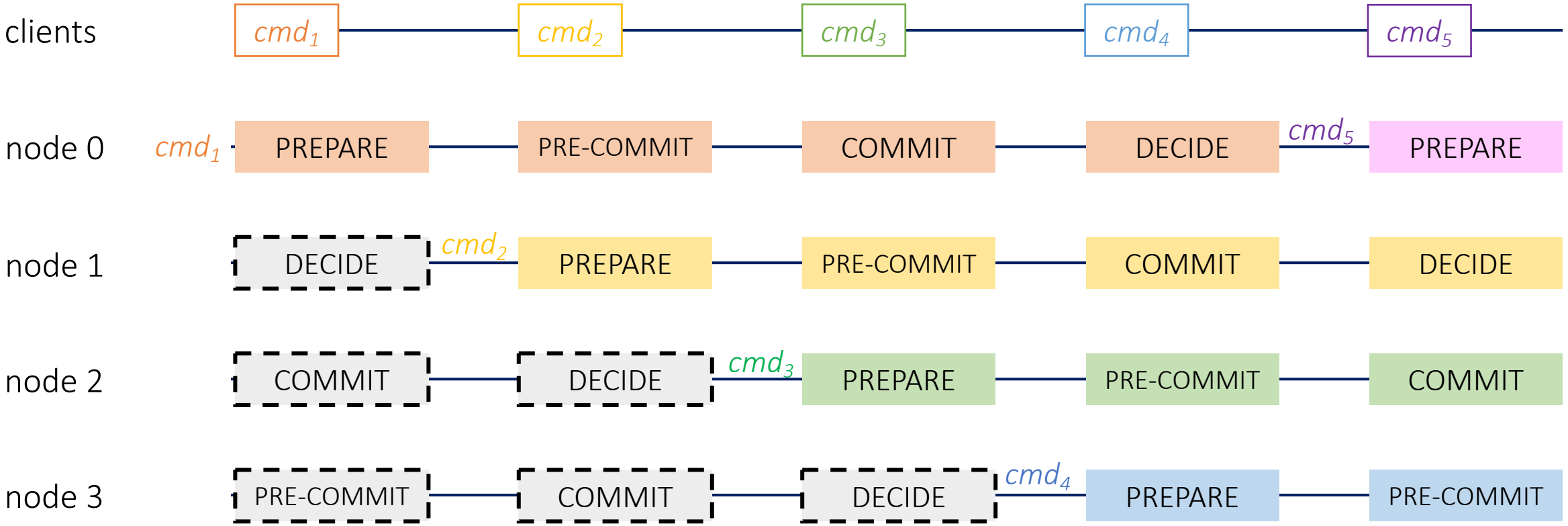
Design choice 3: Leader rotation

- Replace the stable leader with the rotating leader mechanism by adding one phase
  - HotStuff



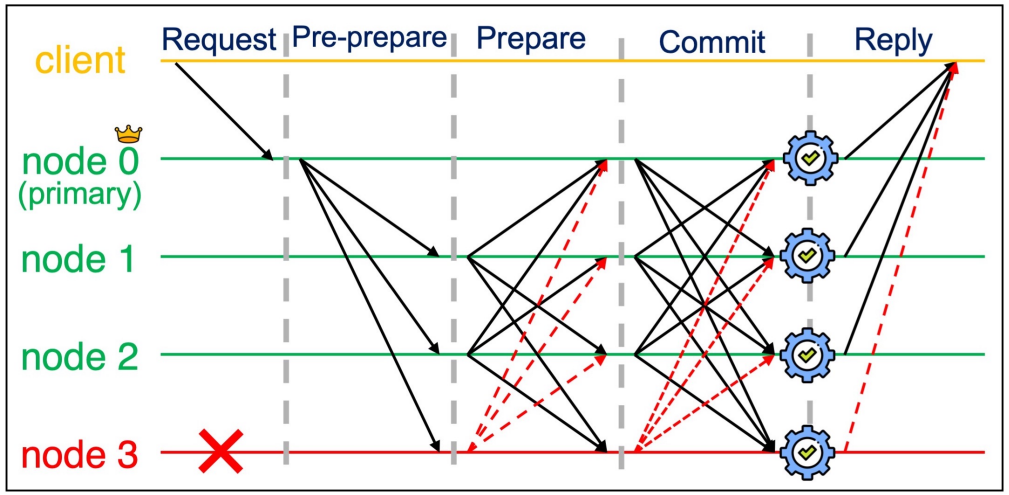
Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G. and Abraham, I., HotStuff: BFT consensus with linearity and responsiveness, PODC 2019

The Pipeline of HotStuff



Design choice 4: Non-responsive leader rotation

- Replace the stable leader with the rotating leader without adding a new ordering phase
  - Sacrifice responsiveness: the new leader waits for a pre-defined known upper bound Δ
  - Tendermint



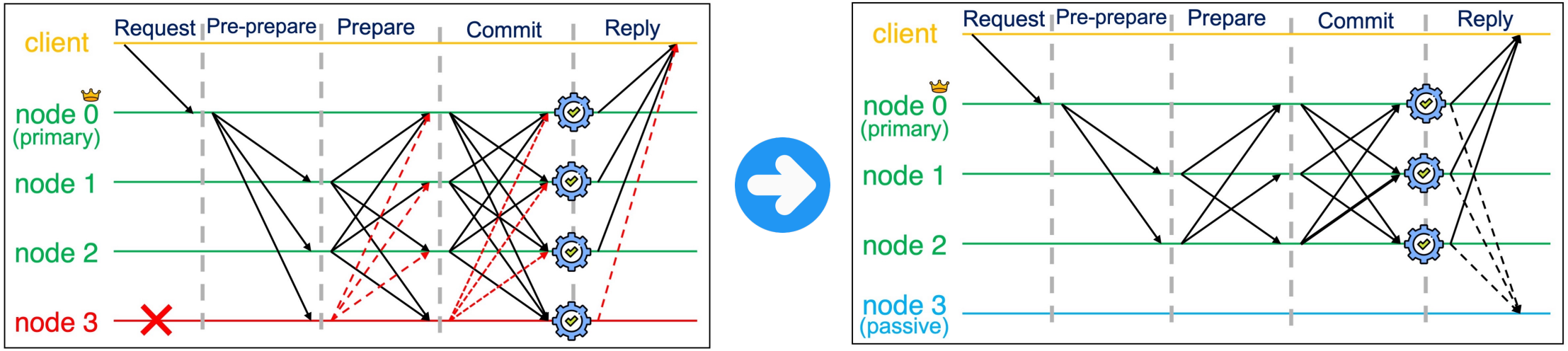
Buchman, E.. Tendermint: Byzantine fault tolerance in the age of blockchains, Doctoral dissertation, University of Guelph, 2016.

Proof of Stake in Tendermint

- Proof of Stake (PoS): a person can mine or validate block transactions according to how many coins he or she holds.
  - What is the difference with PBFT?
- A node can participate in the consensus protocol (become a validator (orderer)) by having coins locked in a bond deposit.
- A validator has voting power equal to the amount of the bonded coins.
- A set of validators with at least 2/3 of total voting power is called a 2/3 majority of validators.

Design choice 5: Optimistic replica reduction

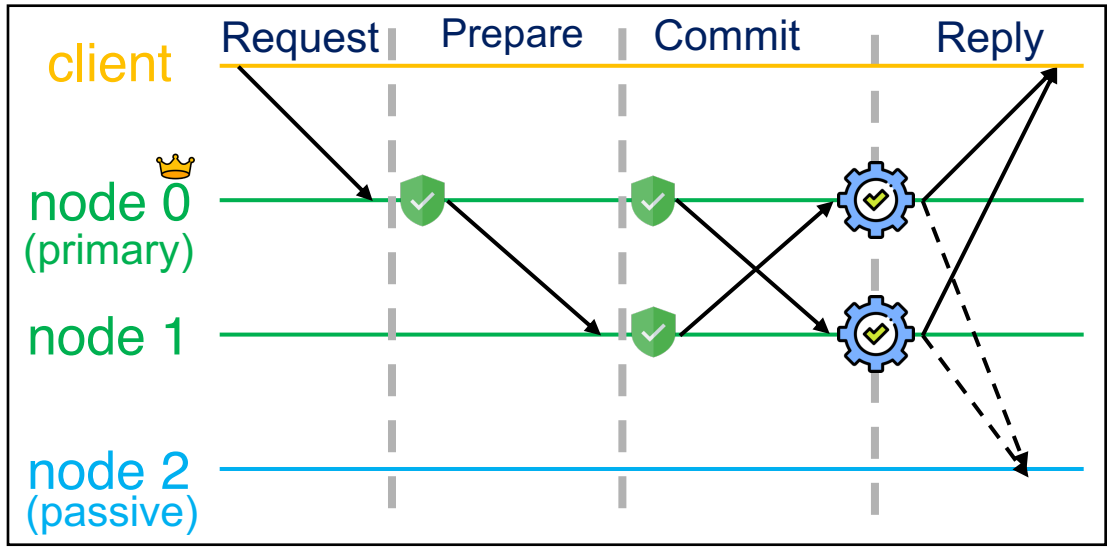
- Reduce the number of involved replicas in consensus from 3f +1 to 2f +1 while optimistically assuming all 2f + 1 active replicas are non-faulty.
  - CheapBFT



Kapitza, R., Behl, J., Cachin, C., Distler, T., Kuhnle, S., Mohammadi, S. V., ... & Stengel, K. CheapBFT: resource-efficient byzantine fault tolerance. EuroSys, 2012

Active-passive replication with trusted hardware

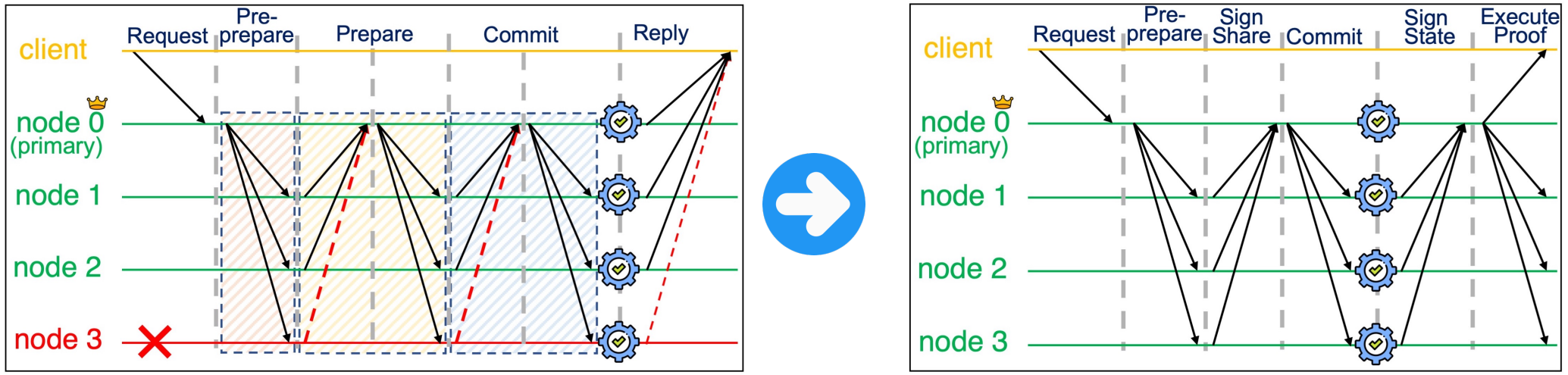
- What if trusted hardware is used?
  - Assigns a unique counter value to each request
  - Creates message certificate and checks message certificate
  - Trusted hardware fails only by crashing
  - Reduces BFT to CFT



Design choice 6: Optimistic phase reduction

- Optimistically eliminate two linear phases assuming all replicas are non-faulty
  - SBFT

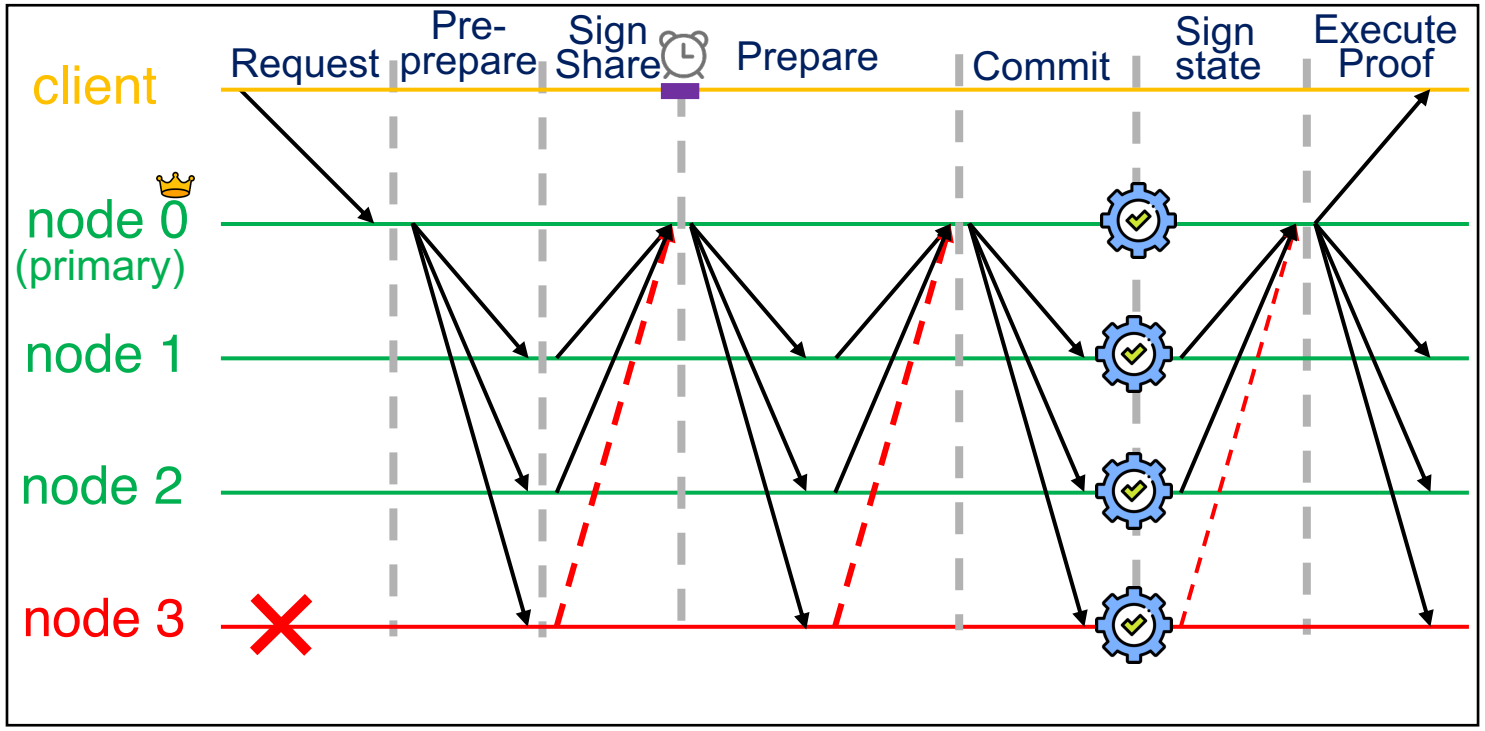
This can be replaced with a regular reply phase



Gueta, G.G., Abraham, I., Grossman, S., Malkhi, D., Pinkas, B., Reiter, M., Seredinschi, D.A., Tamir, O. and Tomescu, A., 2019, June. SBFT: A scalable and decentralized trust infrastructure, DSN 2019

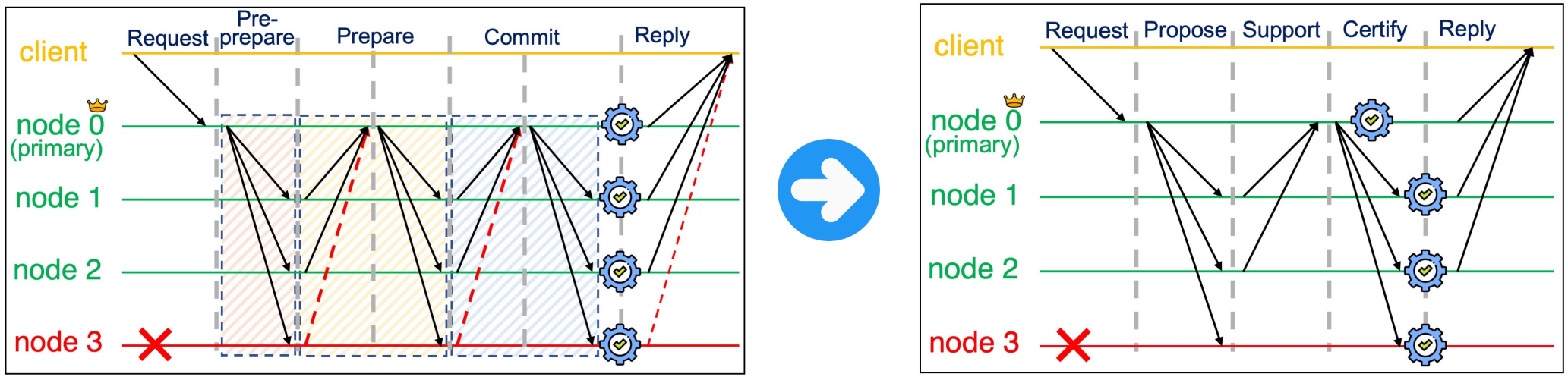
SBFT slow path

- If the fast path fails, it becomes similar to linear-PBFT (with higher latency)



Design choice 7: Speculative phase reduction

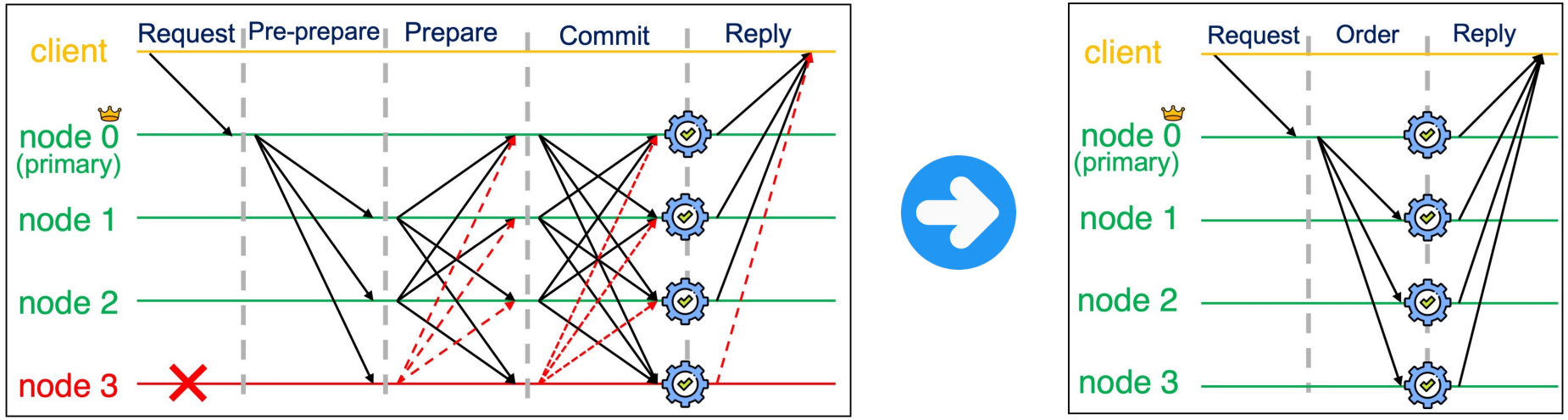
- Optimistically eliminate two linear phases of the ordering stage assuming that non-faulty replicas construct the quorum of responses
  - PoE



Gupta, S., Hellings, J., Rahnama, S. and Sadoghi, M., Proof-of-execution: Reaching consensus through fault-tolerant speculation, EDBT 2021

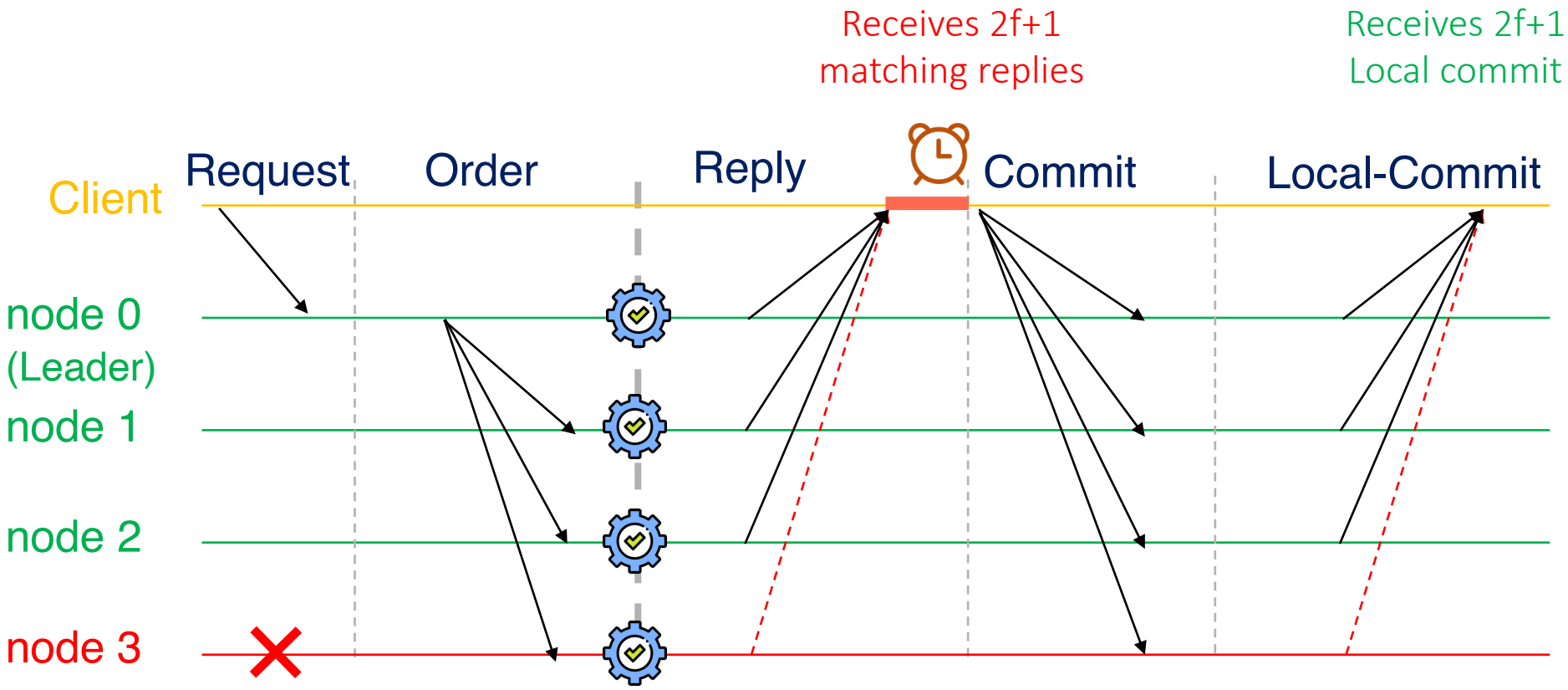
Design choice 8: Speculative execution

- Eliminate the prepare and commit phases while optimistically assuming that all replicas are non-faulty
  - Zyzzyva



Kotla, R., Alvisi, L., Dahlin, M., Clement, A., & Wong, E. Zyzzyva: speculative byzantine fault tolerance, SOSP 2007.

Zyzzyva Agreement Protocol

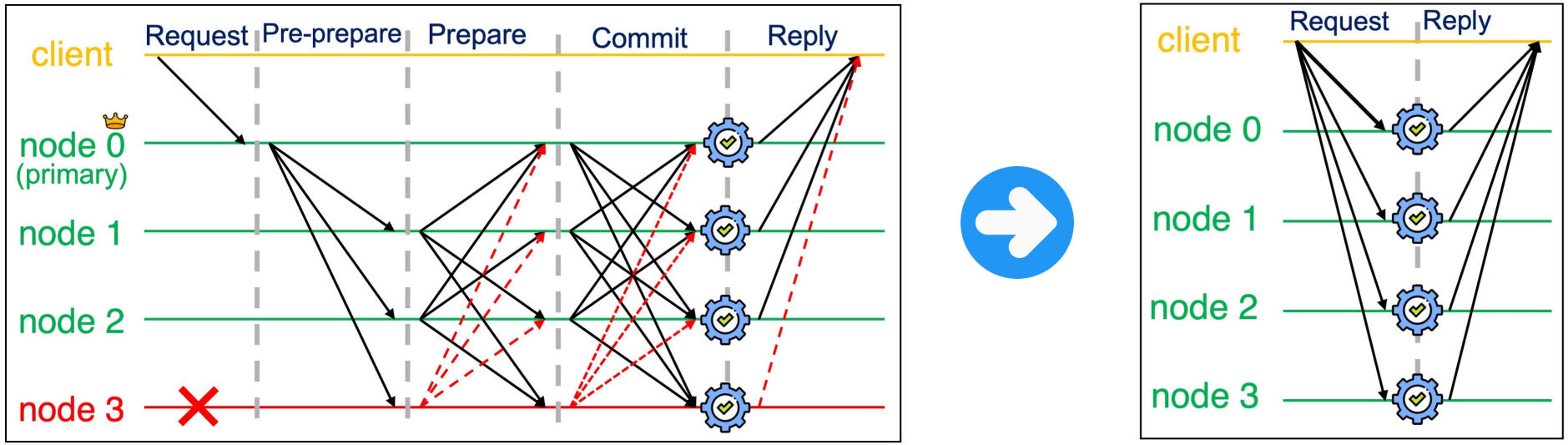


Commit message contains a commit certificate:

A list of 2f+1 replica ids and their signed messages

Design choice 9: Optimistic conflict-free

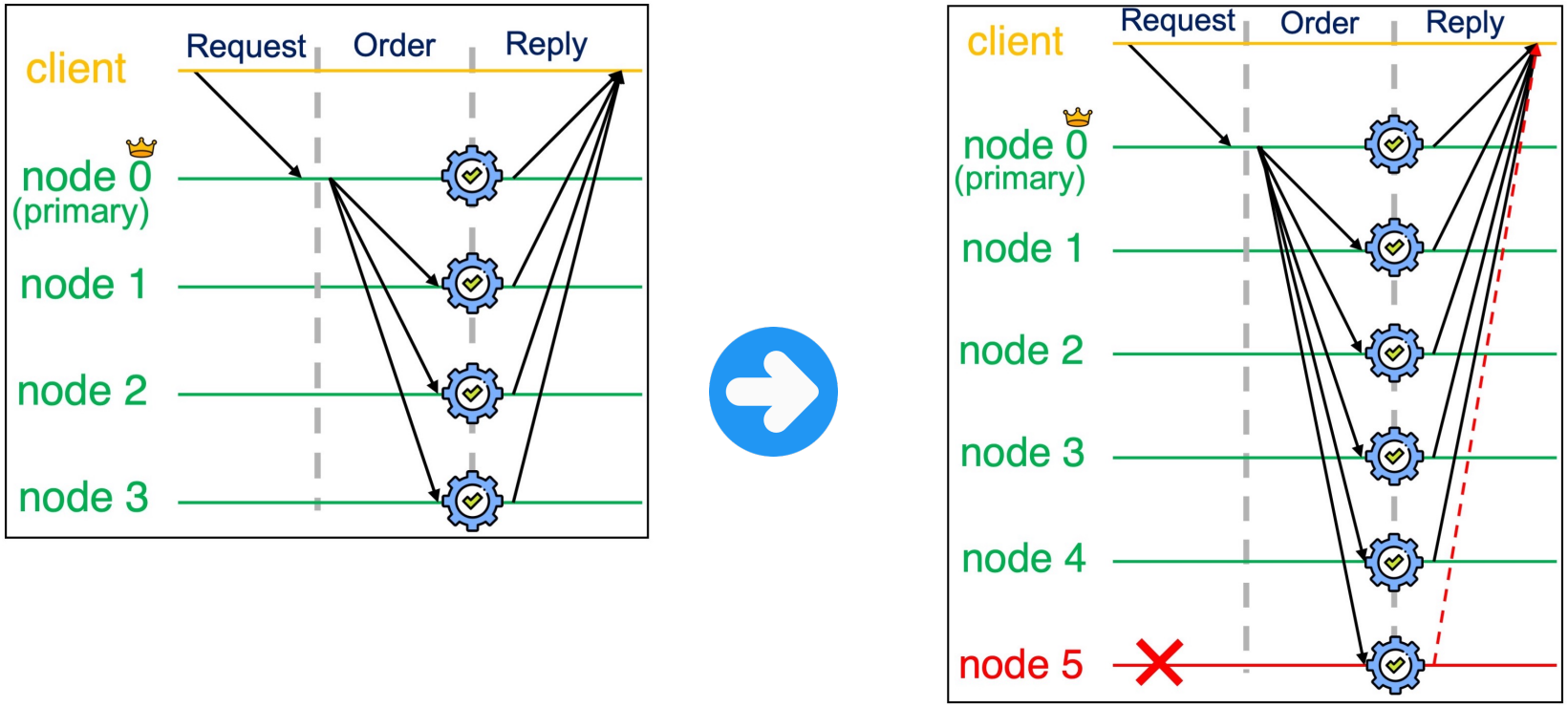
- Eliminate all ordering phases while optimistically assuming that requests are conflict-free and all replicas are non-faulty.
  - The client becomes the proposer
  - HQ



Cowling, J., Myers, D., Liskov, B., Rodrigues, R. and Shrira, L., HQ replication: A hybrid quorum protocol for Byzantine fault tolerance, OSDI 2006

Design choice 10: Resilience

- Increase the number of replicas by 2f enabling the protocol to tolerate f more failure with the same safety guarantees.
  - Zyzzyva 5



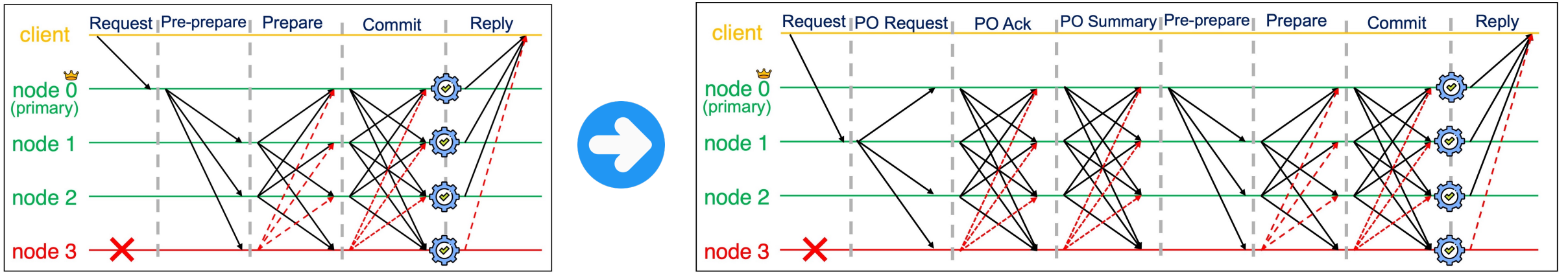
Kotla, R., Alvisi, L., Dahlin, M., Clement, A., & Wong, E. Zyzzyva: speculative byzantine fault tolerance, SOSP 2007.

Design choice 11: Authentication

- Replace MACs with signatures for a given stage
  - Signatures are typically more costly than MACs
  - Signatures provide non-repudiation
  - Threshold signature: replace k signatures (a quorum of signatures)
    - Star communication topology

Design choice 12: Robust

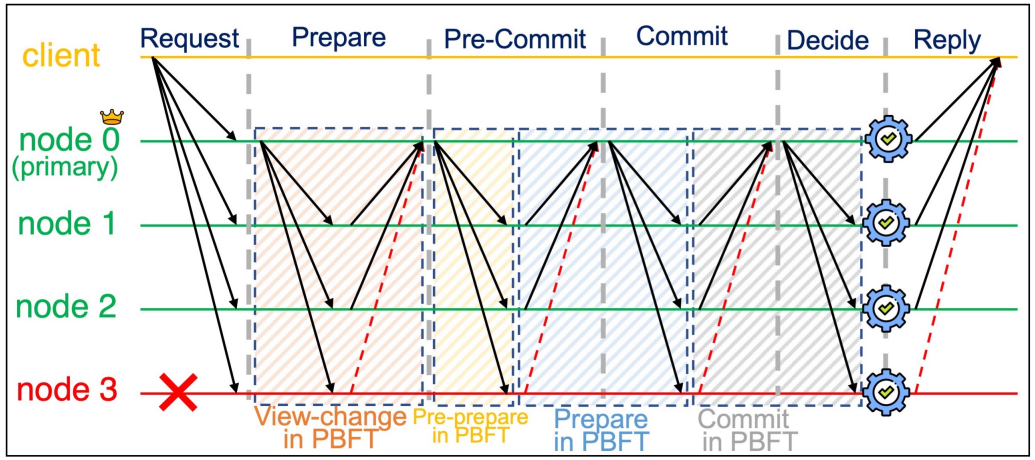
- Make a pessimistic protocol robust (to attacks) by adding a preordering stage
  - Prime
- Preordering stage:
  - Each replica locally orders and broadcasts the request to all other replicas
  - All replicas acknowledge the receipt of the request in an all-to-all communication phase and add the request to their local request vector.
  - Replicas periodically share their vectors with each other.

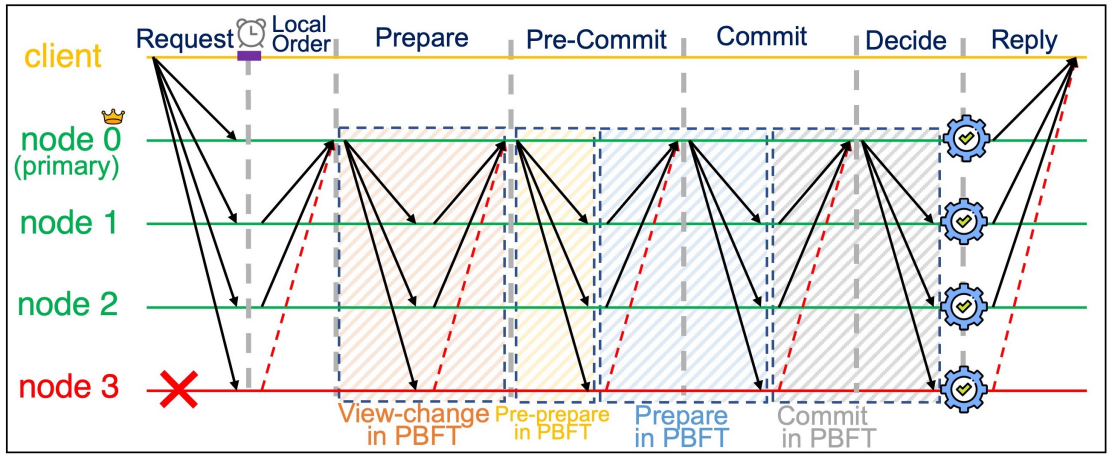


Amir, Y., Coan, B., Kirsch, J. and Lane, J., Prime: Byzantine replication under attack, IEEE TDSC 2010

Design choice 13: Fair

- Transform an unfair protocol, e.g., PBFT, into a fair protocol, e.g., Themis, by adding a preordering phase to the protocol.
  - Clients send requests to all replicas
  - Each replica sends a batch of requests in the received order to the leader
  - The leader initiates consensus following the order of requests in the received batches.
- Require at least 4f +1 replicas ($n > \frac{4f}{2\gamma - 1}$) where order-fairness parameter γ is $0.5 < \gamma \leq 1$

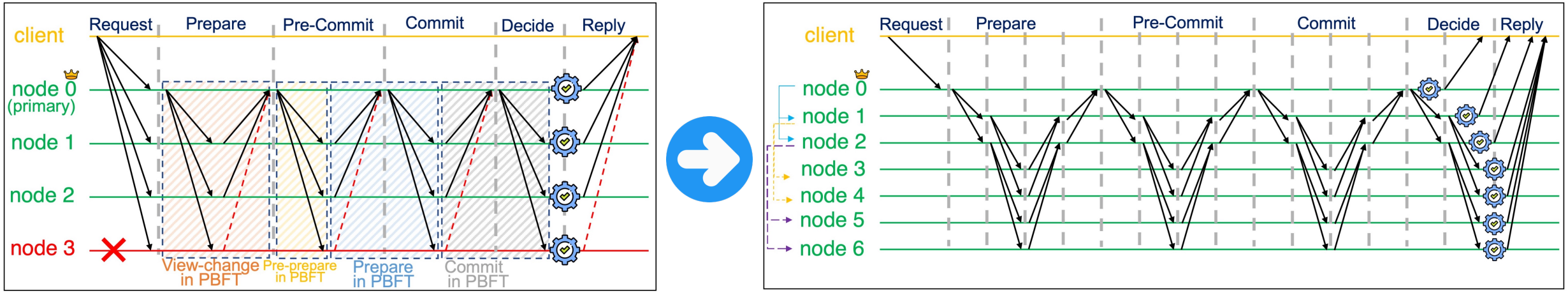




Kelkar, M., Deb, S., Long, S., Juels, A. and Kannan, S., Themis: Fast, strong order-fairness in byzantine consensus, CCS 2023

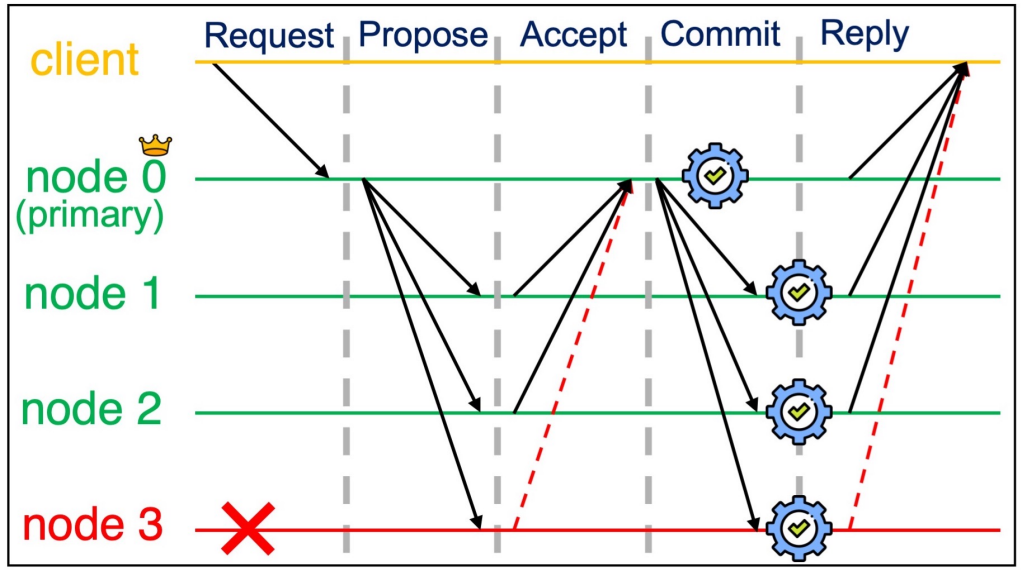
Design choice 14: Tree-based LoadBalancer

- A trade-off between the communication topology and load balancing
  - Load balancing is supported by organizing replicas in a tree topology
  - Splits a linear communication phase into h phases where h is the tree's height
  - Kauri



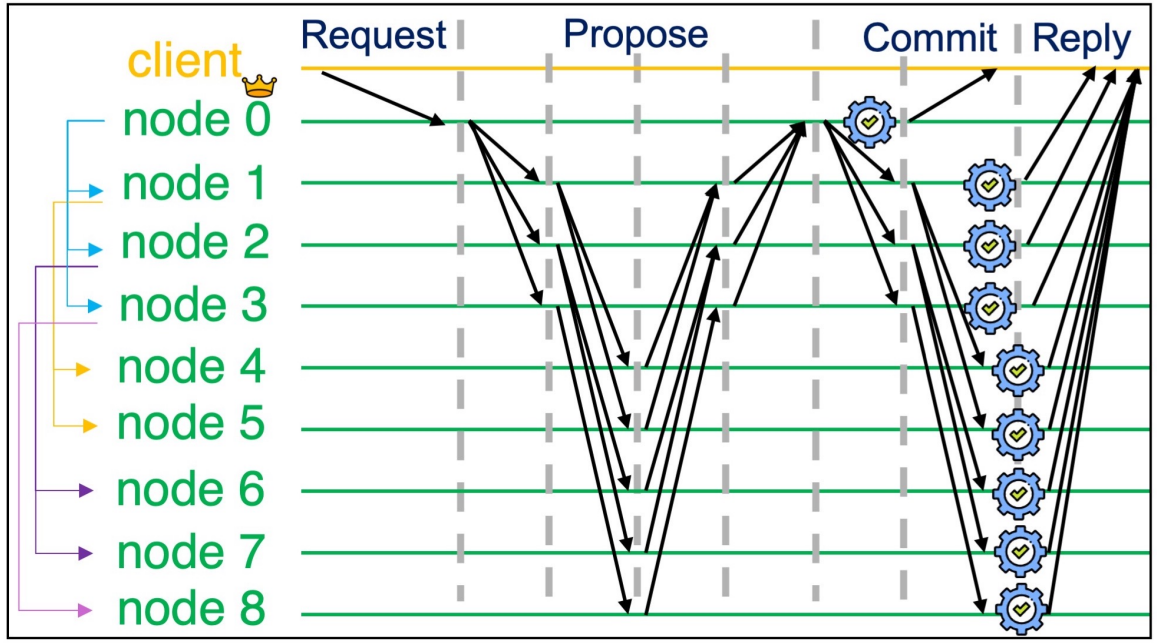
Neiheiser, R., Matos, M. and Rodrigues, L., Kauri: Scalable bft consensus with pipelined tree-based dissemination and aggregation, SOSP 2021

New protocols: FLB and FTB



FLB: Fast Linear BFT protocol

- Commits transactions in two phases
- Linear message complexity
- 5f-1 nodes (f=1)



FTB: Fast Tree-based BFT protocol

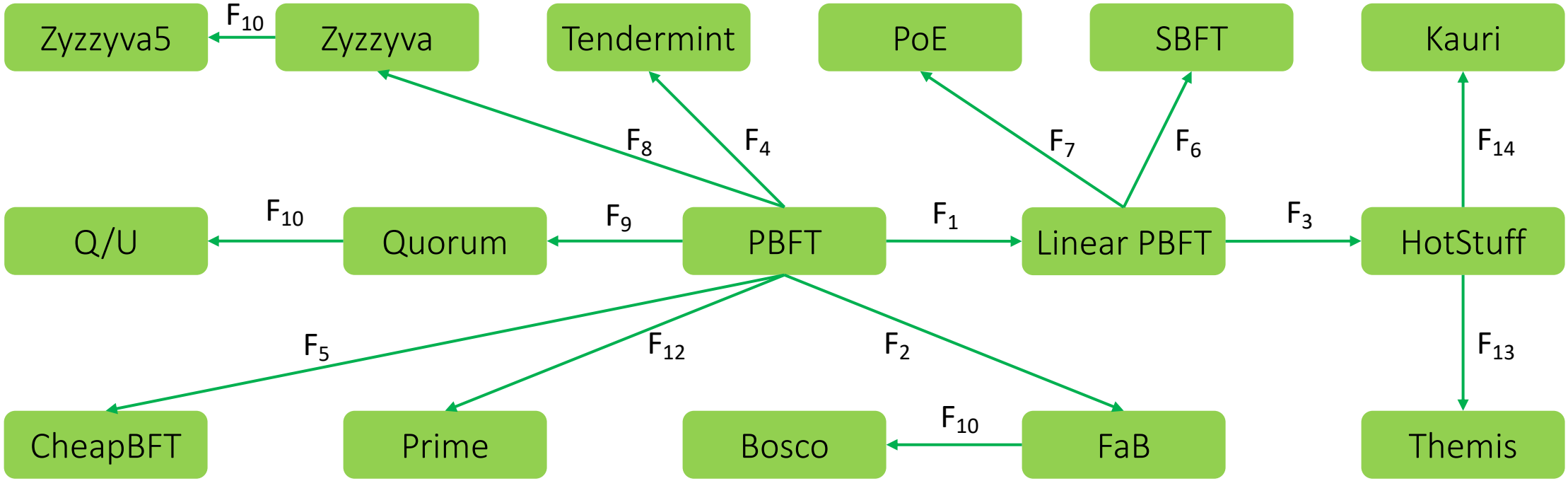
- Commits transactions in 3h phases
- Logarithmic message complexity
- 5f-1 nodes (f=2)

Overview of BFT protocols

Protocol	preordering	o_1: pre-prepare	o_2: prepare	o_3: commit		v_1: View-change	v_2: New view	checkpoint
PBFT		o_1: O(n)	o_2: $O(n^2)$	o_3: $O(n^2)$		v_1: $O(n^2)$	v_2: O(n)	$O(n^2)$
Zyzzyva		o_1: O(n)		o_2: 2*O(n)		v_1: $O(n^2)$	v_2: O(n)	$O(n^2)$
PoE		o_1: O(n)	o_2: 2*O(n)			v_1: $O(n^2)$	v_2: O(n)	O(n)
SBFT		o_1: O(n)	o_2: 2*O(n)	2* O(n)		v_1: O(n)	v_2: O(n)	$O(n^2)$
HotStuff		v_1: 2* O(n)	v_2: O(n)	o_1: O(n)	o_2: 2*O(n)	o_3: 2*O(n)		
Tendermint		o_1: O(n)	o_2: $O(n^2)$	o_3: $O(n^2)$				
Themis	$O(n^2)$	v_1: 2*O(n)	v_2: O(n)	o_1: O(n)	o_2: 2*O(n)	o_3: 2*O(n)		
Kauri		2h*O(n)	h * O(n)	o_1: h*O(n)	o_2: 2h*O(n)	o_3: 2h*O(n)		v_1: O(n)
CheapBFT		o_1: O(n)	o_2: $O(n^2)$	o_3: $O(n^2)$		v_1: $O(n^2)$	v_2: O(n)	$O(n^2)$
FaB		o_1: O(n)	o_2: $O(n^2)$			v_1: $O(n^2)$	v_2: O(n)	$O(n^2)$
Prime	$O(n)+2*O(n^2)$	o_1: O(n)	o_2: $O(n^2)$	o_3: $O(n^2)$		v_1: $O(n^2)$	v_2: O(n)	$O(n^2)$
Q/U						v_1: 2*O(n)	v_2: O(n)	unspecified
FLB		o_1: O(n)	o_2: 2*O(n)			v_1: O(n)	v_2: O(n)	$O(n^2)$
FTB		o_1: h*O(n)	o_2: 2h*O(n)			v_1: O(n)	v_2: O(n)	$O(n^2)$

Derivation of protocols from PBFT

1. Linearization
2. Phase reduction through redundancy
3. Leader rotation
4. Non-responsive leader rotation
5. Optimistic replica reduction
6. Optimistic phase reduction
7. Speculative phase reduction
8. Speculative execution
9. Optimistic conflict-free
10. Resilience
11. Authentication
12. Robust
13. Fair
14. Tree-based LoadBalancer



Comparing selected BFT protocols

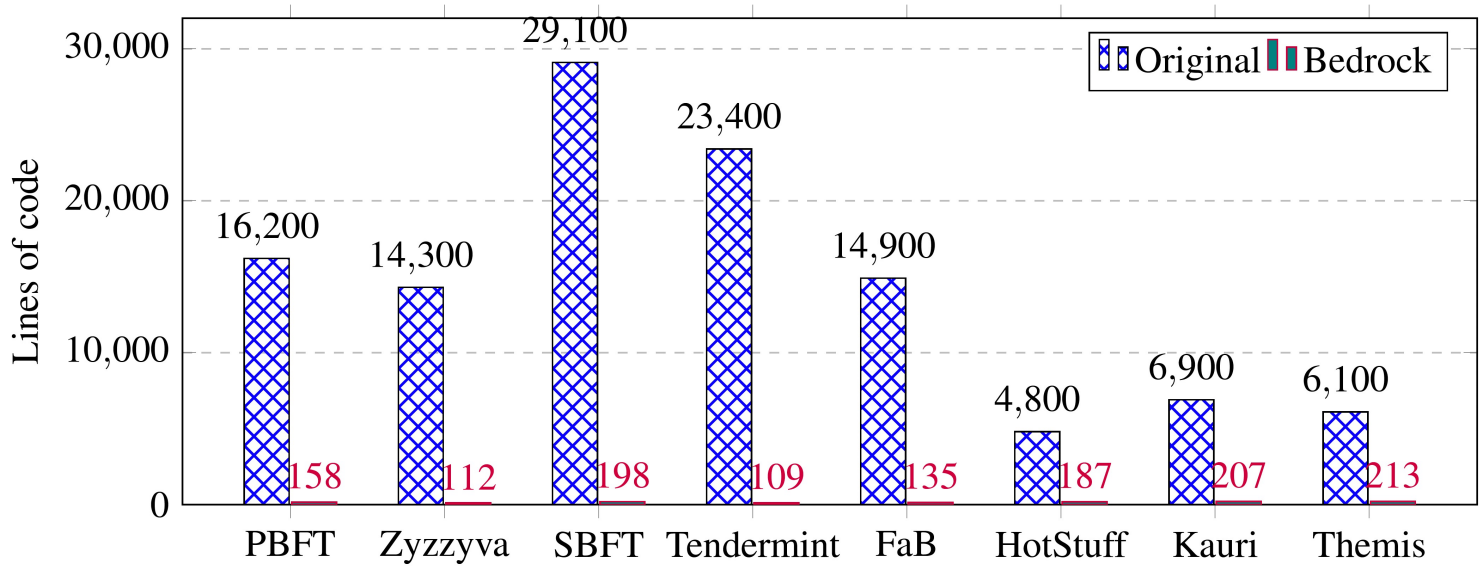
Protocol	E1. Nodes	E2. Topo.	E3. Auth.	E4. Timers	P1. Strategy	P2. Phases	P3. V-change	P5. Rec.	P6. Client	Q1. Fair.	Q2. Load.	Design Choices
PBFT [68]	$3f+1$	clique	MAC || Sign	τ_1, τ_2, τ_8	pessimistic	3	stable	pro.	Req.	☐	☐	**(11)**
Zyzzyva [138]	$3f+1$	star	MAC || Sign	τ_1, τ_2	optimistic (spec): a_1, a_2	1 (3)	stable	-	Rep.	☐	☐	**8**, **(11)**
Zyzzyva5 [138]	$5f+1$	star	MAC || Sign	τ_1, τ_2	optimistic (spec): a_1	1 (3)	stable	-	Rep.	☐	☐	**8**, **10**, **(11)**
PoE [118]	$3f+1$	star	MAC || T-Sign	τ_1, τ_2	optimistic (spec): a_2	3	stable	-	Req.	☐	☐	**1**, **7**, **11**
SBFT [115]	$3f+1$	star	T-Sign	τ_1, τ_2, τ_3	optimistic: a_2	3 (5)	stable	-	Req.	☐	☐	**1**, **6**, **11**
HotStuff [217]	$3f+1$	star	T-Sign	τ_1, τ_2	pessimistic	7	rotating	-	Req.	☐	☐	**1**, **3**, **11**
Tendermint [59]	$3f+1$	clique	Sign	τ_1, τ_2, τ_5, τ_6	optimistic: a_6	3	rotating	-	Req.	☐	☐	**4**, **11**
Themis [130]	$4f+1$	star	T-Sign	τ_1, τ_2, τ_6	pessimistic	1 + 7	rotating	-	Req.	■	☐	**1**, **3**, **13**, **11**
Kauri [175]	$3f+1$	tree	T-Sign	τ_1, τ_2	optimistic: a_3	$7h$	stable*	-	Req.	☐	■	**(3)**, **14**, **11**
CheapBFT[128]	$2f+1$	clique	MAC	τ_1, τ_2	optimistic: a_2	3	stable	-	Req.	☐	☐	**5**
FaB [165]	$5f+1$	clique	(Sign)	τ_1, τ_2	pessimistic	2	stable	-	Req.	☐	☐	**2**
Prime [23]	$3f+1$	clique	Sign	τ_1, τ_2, τ_6, τ_7	robust	6	stable	-	Req.	◨	☐	**11**, **12**
Q/U [4]	$5f+1$	star	MAC	τ_1, τ_2	optimistic: a_4, a_5	1 (3)	stable	-	Rep.	☐	☐	**9**, **10**
FLB	$5f-1$	clique	Sign	τ_1, τ_2	pessimistic	2	stable	-	Req.	☐	☐	**1**, **2**, **11**
FTB	$5f-1$	tree	T-Sign	τ_1, τ_2	optimistic: a_3	$3h$	stable	-	Req.	☐	■	**1**, **2**, **14**, **11**

Bedrock implementation

- The core unit
  - Defines entities, e.g., clients and nodes, and maintains the application logic and data
  - Defines workloads and benchmarks
- The state manager
  - Enables the core unit to track the states and transitions of each entity according to the protocol
  - Defines a domain-specific language (DSL) to rapidly prototype BFT protocols
- The plugin manager
  - Implements protocol-specific behaviors that cannot be handled by the protocol config
  - Enables users to define their own dimensions/values or to update existing dimensions without requiring changes to the platform code or rebuilding the platform binaries
- The coordination unit
  - manages the run-time execution
  - E.g., manages benchmarks, setups all entities, enables plugins to run, reports results

Bedrock DSL code

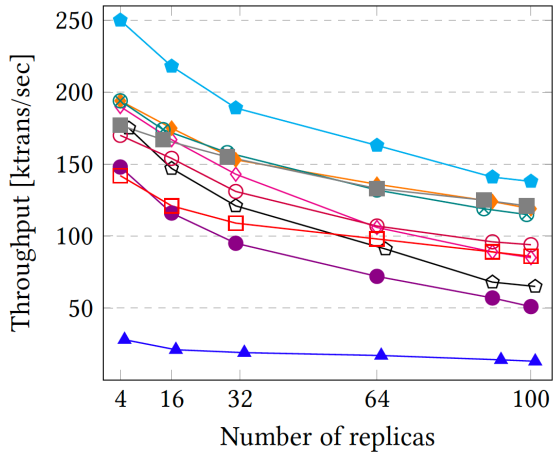
- Written in the protocol config
- Defines different dimensions and their chosen values, the list of roles, phases, states, messages, quorum conditions, and plugins
- Rapid prototyping of BFT protocols



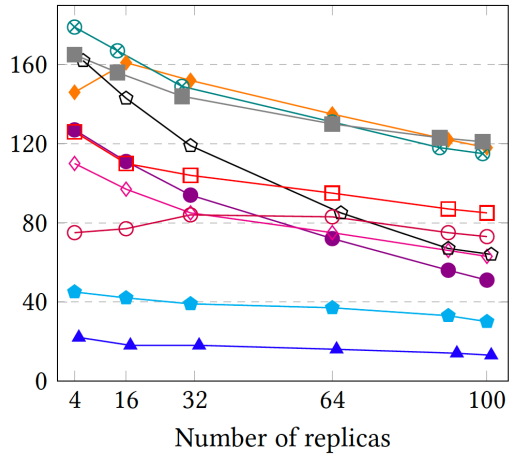

```
protocol:
  general:
    leader: stable
    requestTarget: primary

  roles:
    - primary
    - nodes
    - client

  phases:
    - name: normal
      states:
        - idle
        - wait_prepare
        - wait_commit
        - executed
      messages:
        - name: request
          requestBlock: true
        - name: reply
          requestBlock: true
        - name: preprepare
          requestBlock: true
        - prepare
        - commit
    - name: view_change
      states:
        - wait_view_change
        - wait_new_view
      messages:
        - view_change
        - new_view
    - name: checkpoint
      messages:
        - checkpoint

  transitions:
    from:
      - role: client
        state: idle
        to:
          - state: executed
            update: sequence
            condition:
              type: msg
              message: reply
              quorum: 2f + 1
```

Bedrock platform

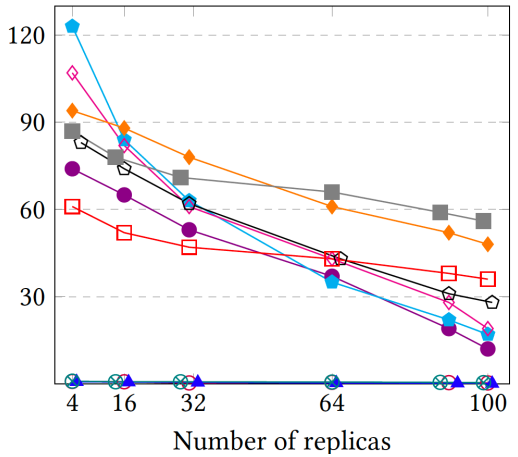
- Fair and Efficient experimental evaluation of BFT protocols



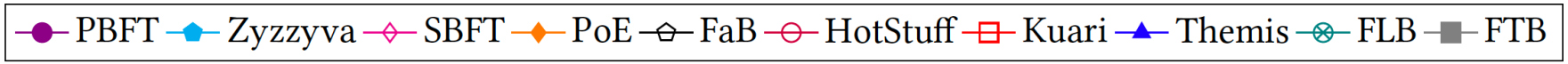
No faulty replica



Single faulty backup



Geo-distributed setup



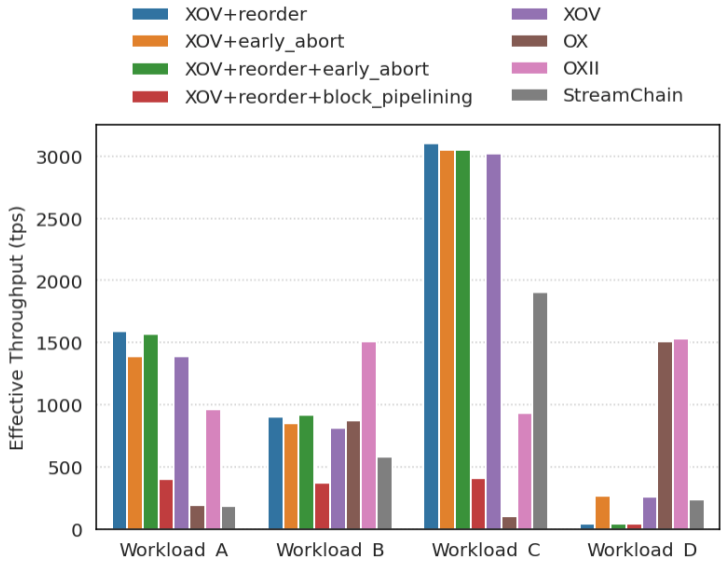
Beyond consensus protocols

- Transaction processing: ordering and execution
  - Concurrency control mechanism
  - Transaction reordering algorithms
  - Block size adaptation

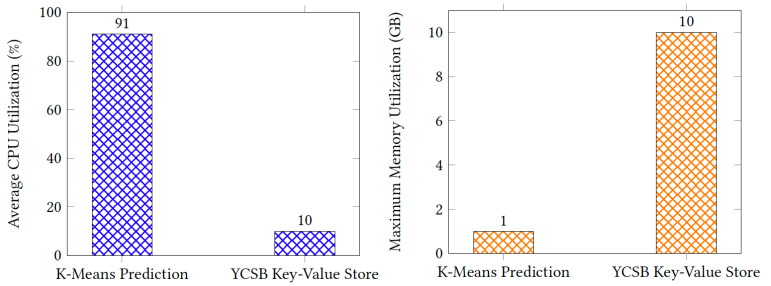
Wu, C., Mehta, B., Amiri, M. J., Marcus, R., & Loo, B. T. AdaChain: A Learned Adaptive Blockchain, VLDB'23

- Hardware resource management
  - Elasticity of disaggregated data center (DDC) infrastructure
  - Switching between DDC vs. non-DDC traditional setup
  - How to deal with the high overhead of switching?

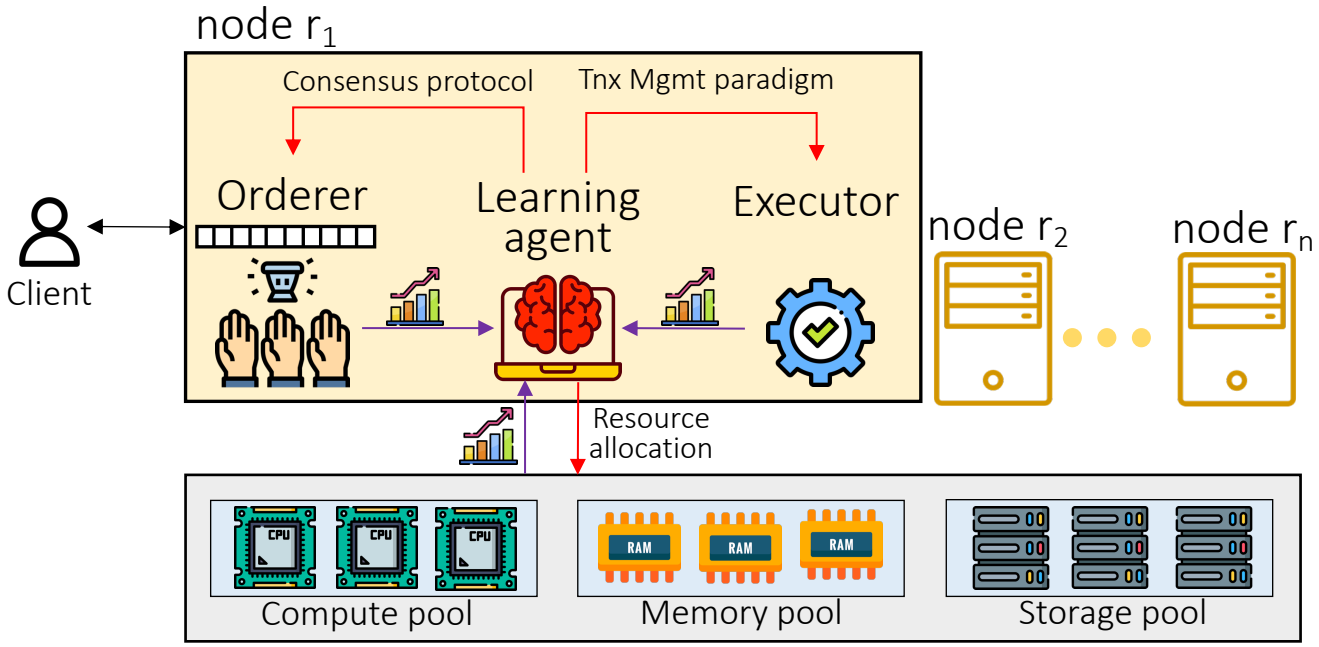
Wu, C., Amiri, M. J., Asch, J., Nagda, H., Zhang, Q., & Loo, B. T. FlexChain: an elastic disaggregated blockchain, VLDB'23



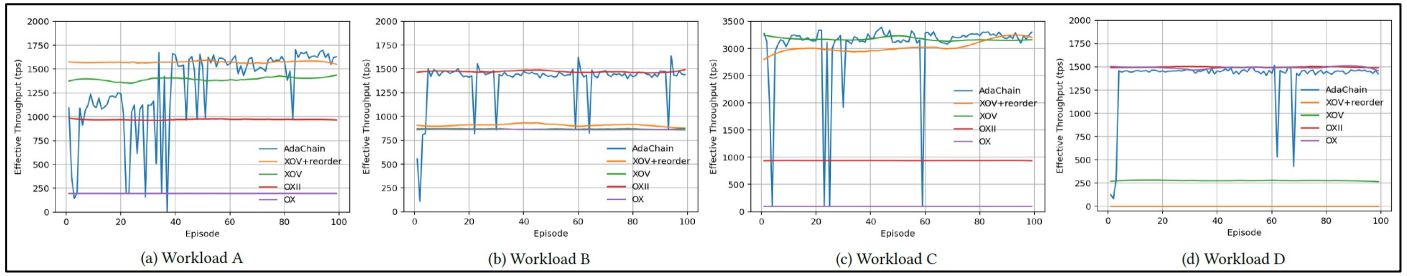
Workload	Write Ratio	Contention Level	Load	Compute Intensity
A	low	high	high	high
B	moderate	high	moderate	low
C	moderate	low	high	Very high
D	high	very high	moderate	Very low

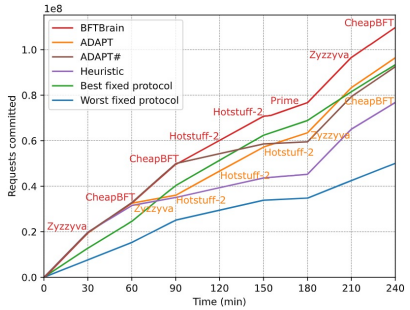


Full Stack Adaptivity



- Robust online data collection
  - No single trusted entity
  - Each node has a learning agent
  - Nodes agree on decisions
  - Featurizing faults and protocols
- Switch at runtime
  - Protocol, paradigm or resources
- Cross-layer adaptivity
  - Identify performance bottlenecks
  - Protocol/paradigm Compatibility





Wu, C., Amiri, M. J., Qin, H., Mehta, B., Marcus, R., & Loo, B. T. , Towards Full Stack Adaptivity in Permissioned Blockchains. VLDB'24

Future work

- Enabling users to check the correctness of their written protocols
  - Transforming the DSL code written in Bedrock to the language used by verification tools
- Diversifying replica implementation using n-version programming
  - To ensure the independent failure of replicas
- Designing a constraint checker to automatically find all plausible points
  - Given a user query what are the valid combinations of design choices in the design space
- Incorporating automatic selection strategies in Bedrock
  - Using machine learning to select the appropriate BFT protocol, or switch protocols at runtime
- Extending the supported protocols
  - E.g., adding synchronous and fully asynchronous protocols


Questions?