

Distributed Consensus and Diffuse Trust Systems

Samuel M. Smith Ph.D.
smith.samuel.m@gmail.com

Why Distributed Consensus

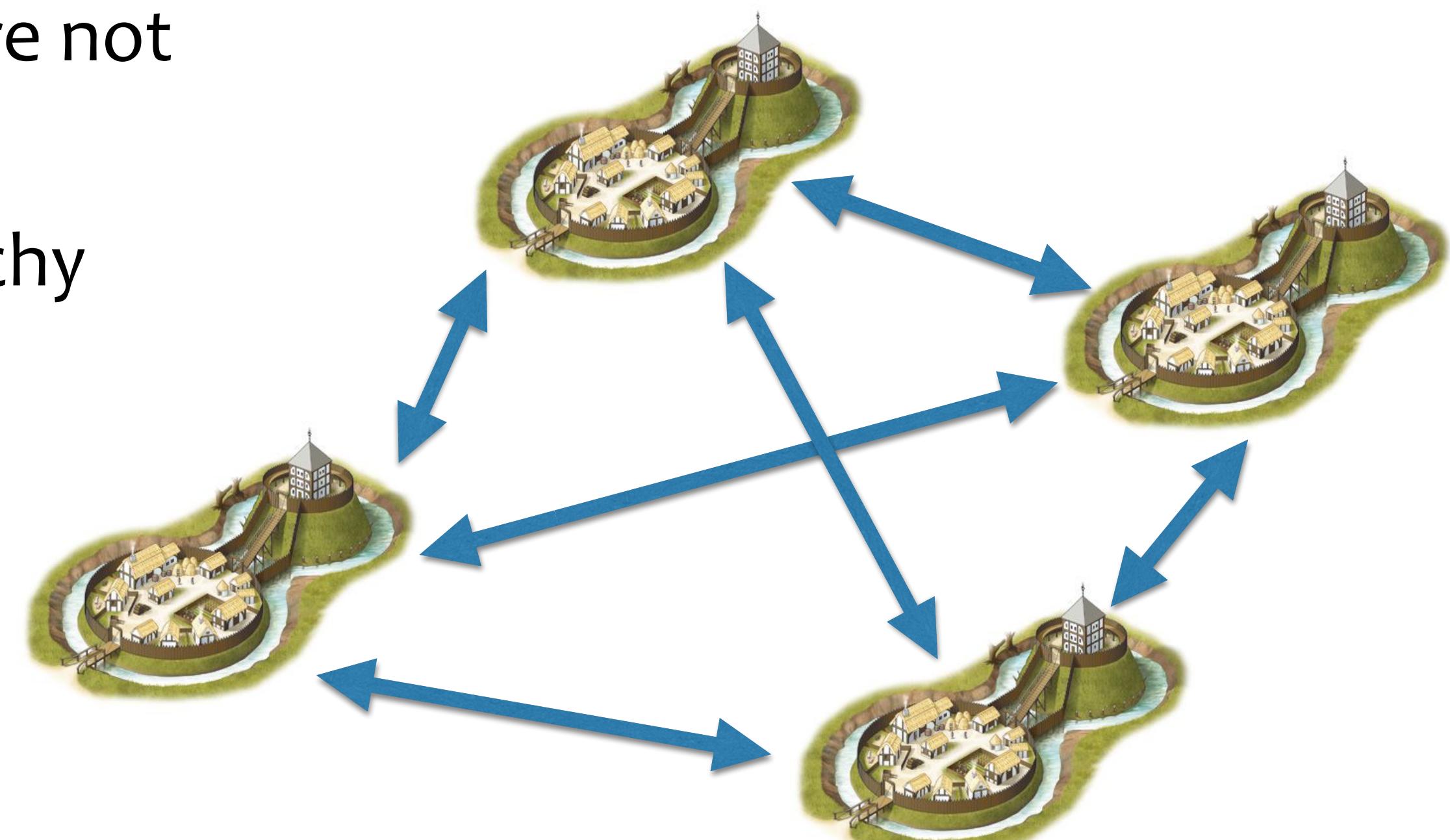
- Distributed consensus enables diffuse trust systems for de-centralized computing infrastructure
- Diffuse trust allows secure infrastructure outside a firewall by distributing the attack surface.
- Diffuse trust systems can disruptively dis-intermediate centralized computing infrastructure.
- Diffuse trust systems enable new business models
 - Computation in the “fog” vs computation in the “cloud”
 - Distributed AI
 - Smart contracts
 - IOT
 - etc

Diffuse Trust

- Cellular distributed topology.
“Think clandestine spy ring or resistance organization”
- Defeating each node (cell, element) requires an independent exploit.
 - No single point of failure. “Universal root privileges”
 - No common mode failures. “Exploit to attain root privileges”
- As long as the majority or super majority of nodes are not exploited the system is trustworthy.
- Cooperation between peers vs. authoritarian hierarchy



vs



Distributed Consensus Types

- Proof of Work: Simple, highly inefficient, high latency, low throughput
- Proof of Stake: More complex, more efficient, lower latency, higher throughput
(Asymmetric Proof of Work)
- Byzantine Agreement: Most complex, most efficient, lowest latency, highest throughput.
- Hybrid:

Permissioned Distributed Consensus

AKA XBFT

- Permissioned Distributed Ledgers
Swanson 2105
 - Higher speed application specific distributed consensus
 - Database
 - Open Storage
 - DHT
 - Transactions
 - Settlement
 - Smart Contracts
 - Identity
 - Reputation
 - etc ...
- Permissioned governance to address
Sybil attack. Censorship.

BYZANTINE GENERALS

- Analogy to war in the Byzantine era.
- Historically Byzantine generals were prone to traitorous or duplicitous acts.
- The Byzantine Generals Problem:

<http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf>

Two separated generals agree to attack a third. If the two unite they both win. If either attacks alone he loses.

The communication of intent to attack and confirmation of intent to attack is subject to delay and error.

- Byzantine agreement:

A valid consensus is guaranteed despite a fraction of the participants behaving in a malicious and/or erroneous manner.

- Consensus despite “Byzantine” faults:

Dropped packets, Erroneous packets, Malicious Packets, Partitioning, Sybil Attack

A consensus algorithm that solves the Byzantine consensus problem in the presence of malicious and/or erroneous behavior is called Byzantine fault tolerant (BFT).

- There are two features of any BFT consensus solution:

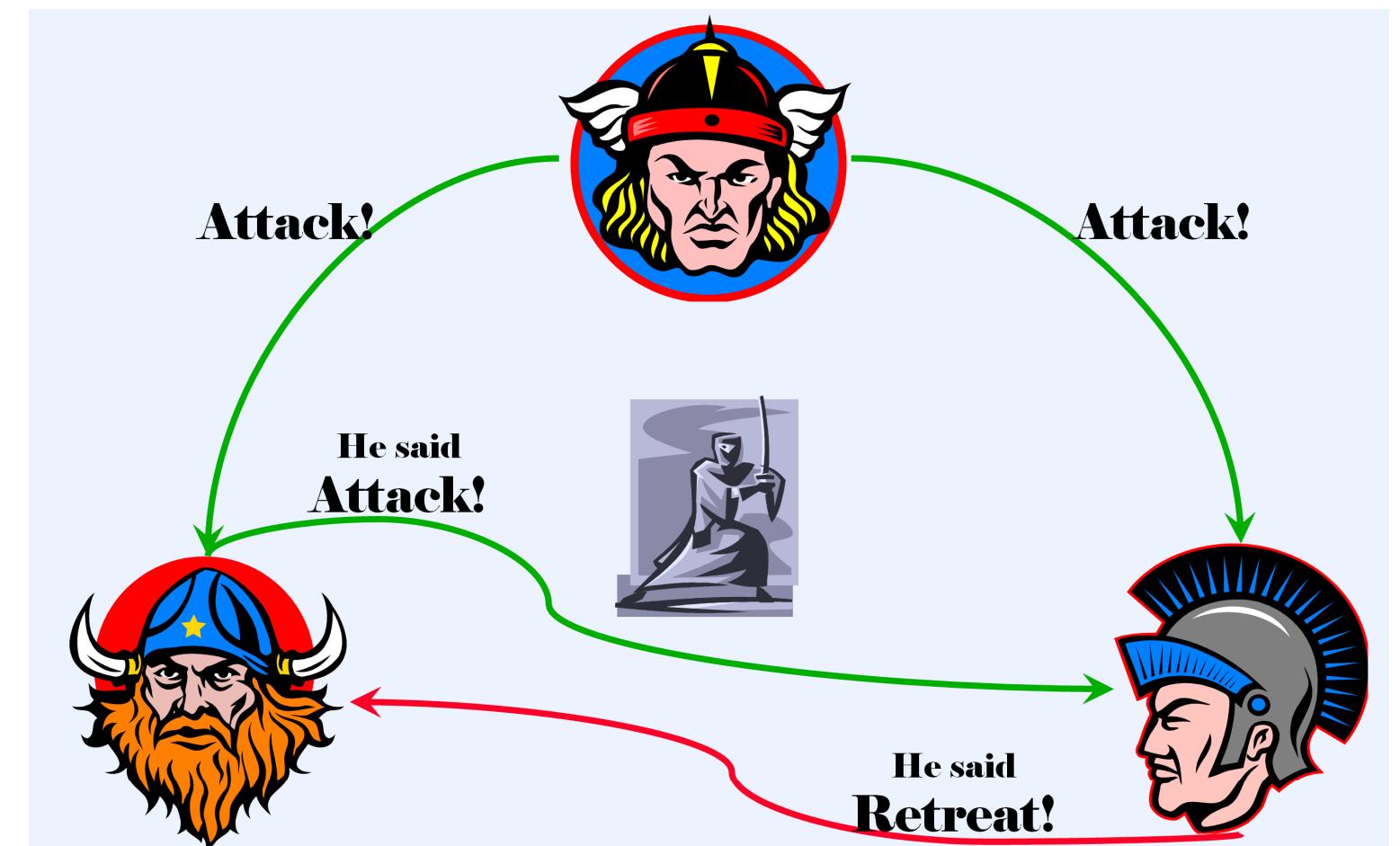
Liveness, that is, the non-faulty nodes make a joint decision in finite time.

Safety, that is, the non-faulty nodes all make the same joint decision.

- All the solutions require that a majority or supermajority, of the nodes be non-faulty.

- The problem is usually couched as a problem of distributed replication of a sequence of states, that is, the non-faulty nodes will all replicate the same sequence of states (decisions).

- For example, the sequence of transactions entered into a shared public ledger or shared public database is the sequence of states.



Practical Byzantine Fault Tolerance (PBFT)

http://pmg.csail.mit.edu/~castro/osdi99_html/osdi99.html

Background

Synchronous Systems:

All steps proceed in rounds.

All message transmissions between nodes fully complete within each round.

Consensus at the end of each round.

Asynchronous:

Steps occur at varying times.

Message transmissions happen whenever.

Consensus?

Fundamental Constraint:

In an asynchronous system it is impossible to achieve consensus with one faulty node.

Impossibility of Distributed Consensus with One Faulty Process; Fischer, Lynch, Patterson, JACM Vol. 32, No. 2, April 1985, pp. 374-382.

<https://groups.csail.mit.edu/tds/papers/Lynch/jacm85.pdf>

Desirable Features

Sequential consistency among nodes:

Results of any execution is the same as if the operations on all the nodes were executed in some sequential order.

Linearizable:

Sequential consistency that respects the real time order of events.

State Machine Replication:

Lots of things can be expressed as a state machine

Goals:

Build a linearizable replicated state machine.

Agreement on operations and their order.

Replication to provide availability while preventing faults including malicious ones.

PBFT Features

Byzantine Faults:

Delay, Error, Stop, Duplicate, Reorder, Malicious Data.

Honest or non-faulty nodes:

Dishonest or faulty nodes:

Liveness:

Quorum of non-faulty nodes make a joint decision in finite time.

Weak synchrony via timing bounds $[h, H]$.

Delay does not grow faster than time. $\text{delay}(t) = o(t)$

Safety:

Quorum of non-faulty nodes all make the same joint decision.

Consensus despite Byzantine faults on some of the nodes.

Consensus despite out of bounds delay on some of the non-faulty nodes.

PBFT Assumptions

All messages between nodes are signed ():

Typically some combination of message authentication codes (MACs) with shared secrets and true digital signatures. Modern ECC digital signatures ameliorate many of the performance advantages of using MACS while enhancing security (nonrepudiation).

At most f faults on any step:

Honest nodes behave deterministically:

Honest nodes fail independently:

No systemic or common mode failures

Weakly synchronous:

Bounded time to respond

$$N = Cf + 1$$

N is the total number of nodes.

$N=f+1$ Correct Agreement:

Given at most f faults or f faulty nodes, any agreement of $f+1$ nodes is correct because at least one of the $f+1$ nodes is honest.
No guarantee that any set of $f+1$ nodes will ever be in agreement.

$N=2f+1$ Correct Majority Agreement:

Given $2f+1$ nodes at least $f+1$ are honest ensuring an honest majority and that at least $f+1$ nodes could be in correct agreement.

$N=3f+1$ Correct Majority Live Agreement:

In addition to f responding but faulty nodes, up to f nodes may not respond so need $3f+1-f=2f+1$ nodes to ensure a majority is formed in bounded time with at least $f+1$ honest nodes.

$N=3f+1$ Correct Majority Live Safe Agreement:

$2f+1$ is minimum Quorum size to form correct majority.

$N-f$ is Quorum size that ensures a Quorum is formed in bounded time.

Given Quorum size of $N-f = 2f+1$, N for which the intersection of any quorum partition will include at least one honest node.

Partition Safety

Means that any two quorums will be in agreement.

$2f+1$ is minimum **Quorum** size to form correct majority.

$N-f$ is constraint on **Quorum** size relative to N that ensures a **Quorum** is formed in bounded time.

Given **Quorum** size of $N-f \geq 2f+1$, find minimum N such that the intersection of any quorum partition includes at least one honest node.

Phrased in terms of cardinality (size) of the quorum partitions. For set S , the cardinality $|S|$ is the number of elements of S .

Overlapping
Partition

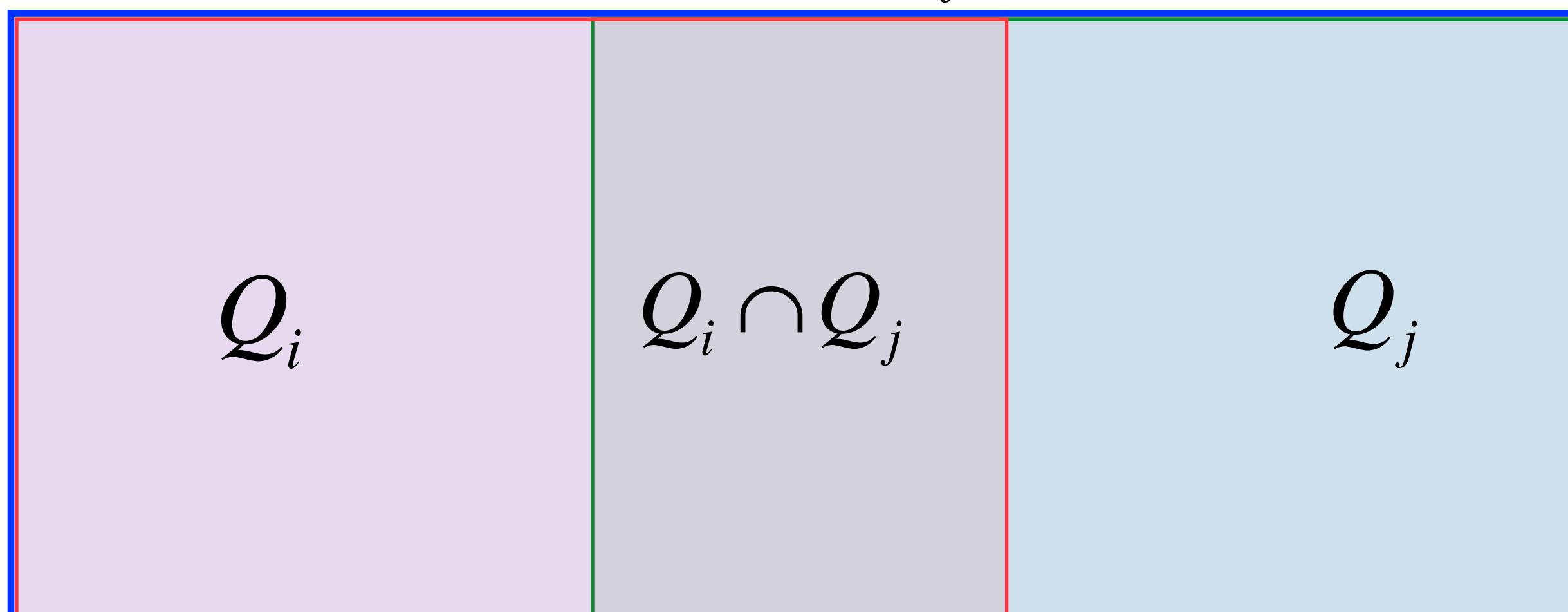
$[Q_i, Q_j]$

$$|Q_i \cap Q_j| \geq f + 1$$

$$Q_i \cup Q_j$$

$$|Q_i| + |Q_j| = |Q_i \cup Q_j| + |Q_i \cap Q_j|$$

$$|Q_i| + |Q_j| - |Q_i \cup Q_j| = |Q_i \cap Q_j| \geq f + 1$$



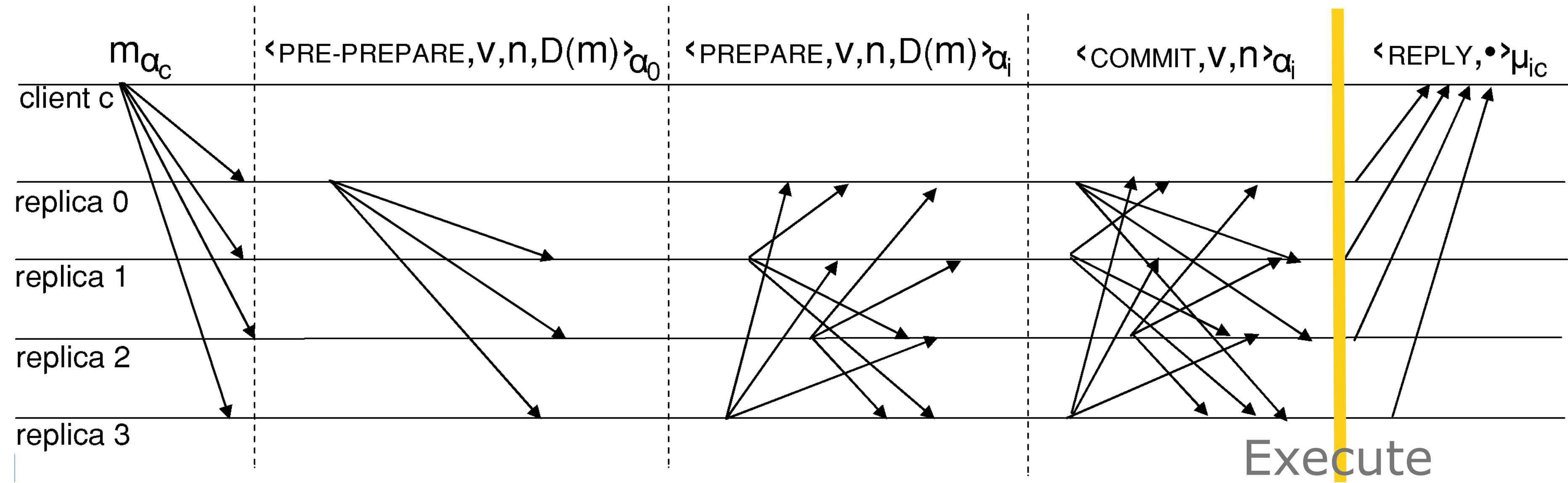
$$(N-f) + (N-f) - N \geq f + 1$$

$$N - 2f \geq f + 1$$

$$N \geq 2f + f + 1 \geq 3f + 1$$

$$\min(N) \rightarrow N = 3f + 1 \rightarrow N - f = 2f + 1$$

Client Request



Client multicasts request to replicas

Primary initiates 3 phase commit

Client collects replies from replicas,
 $f+1$ correct replies before timeout indicates success

Valid signature

Distinct $f+1$ replicas i but same t and r

Each replica accepts request

Verifies signature

logs request

$\text{REQUEST} < o, t, c > \sigma_c$

o = operation

t = request timestamp

c = clientid

σ_c = client signature

$\text{REPLY} < v, t, c, i, r > \sigma_i$

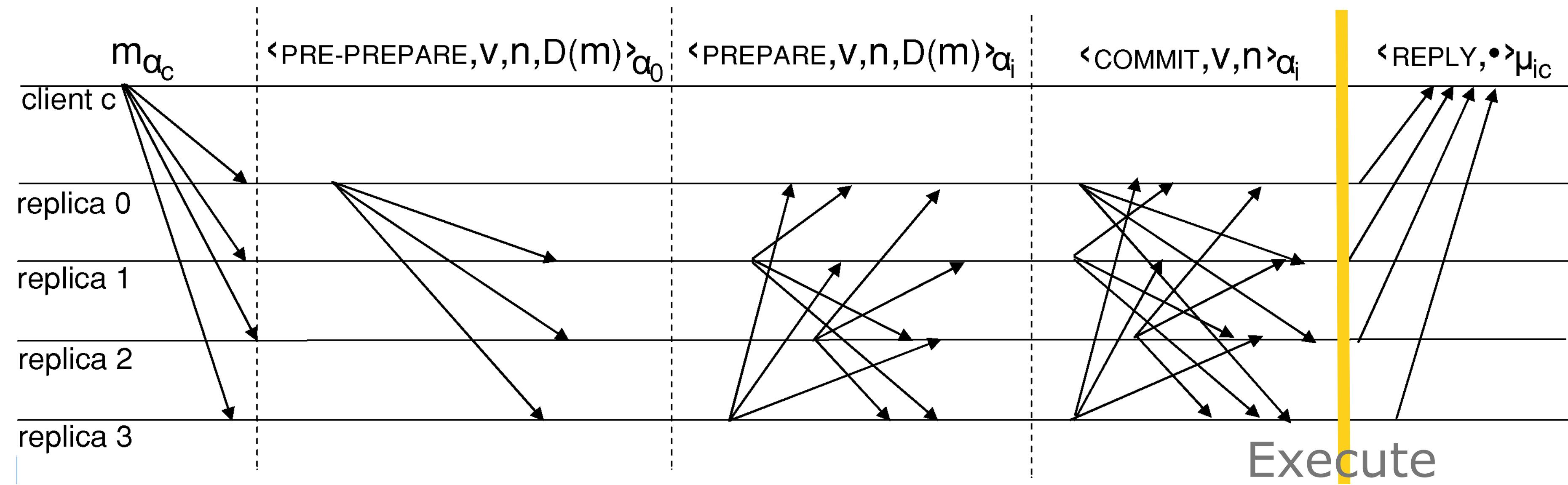
v = view number

i = replica id

r = result of o

σ_i = replica i signature

Pre-Prepare



Primary replica initiates 3 phase commit

$\text{PREPARE} < v, n, d(m) > \sigma_p$

Multicasts pre-prepare with proposed order n

v = view number

Backup replicas accept and log pre-prepare

n = sequence number

Verify signature and digest of request

d = digest

Verify backup is in same view

m = request message

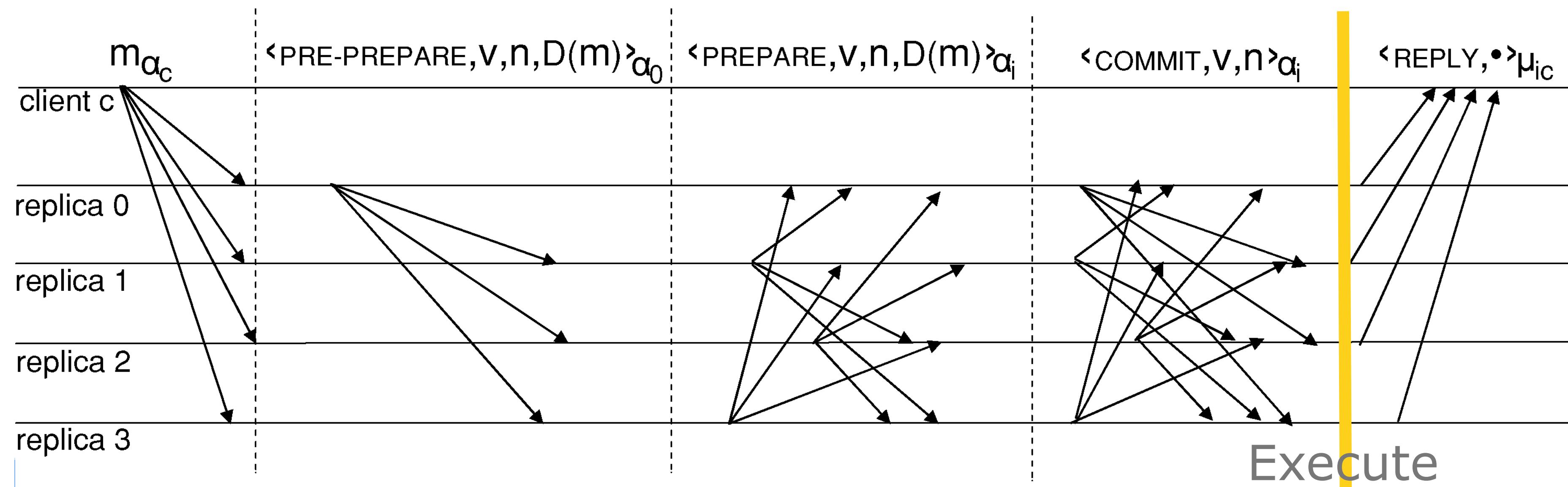
Verify not accepted different request for n

$p = v \bmod N$

Enter prepare phase

σ_p = primary replica signature

Prepare



Backups initiate prepare phase

$PREPARE < v, n, d(m), i > \sigma_i$

Multicast prepare to all replicas

v = view number

Backup replicas accept and log prepare

i = replica id

Verify signatures and v, n, d match

n = sequence number

Backup i is prepared IFF

d = digest

pre-prepare and $2f$ matching prepares

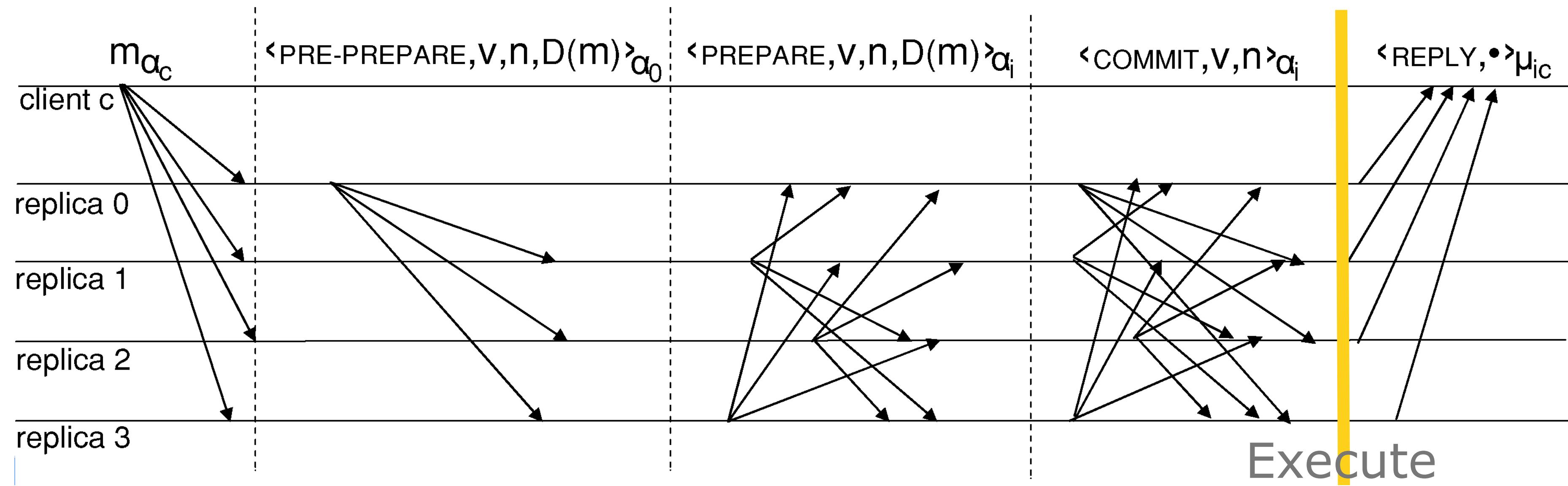
m = request message

knows a safe quorum agrees on n for m in v

σ_i = replica i signature

If $\text{prepared}(m, v, n, i)$ is True, Then $\text{prepared}(m, v, n, j)$ is False for $i \neq j$

Commit



Backups initiate commit phase

$\text{COMMIT} < v, n, i > \sigma_i$

Multicast commit to all replicas

v = view number

Backup replicas accept and log commit

n = sequence number

Backup i performs operation \circ IFF

i = replica id

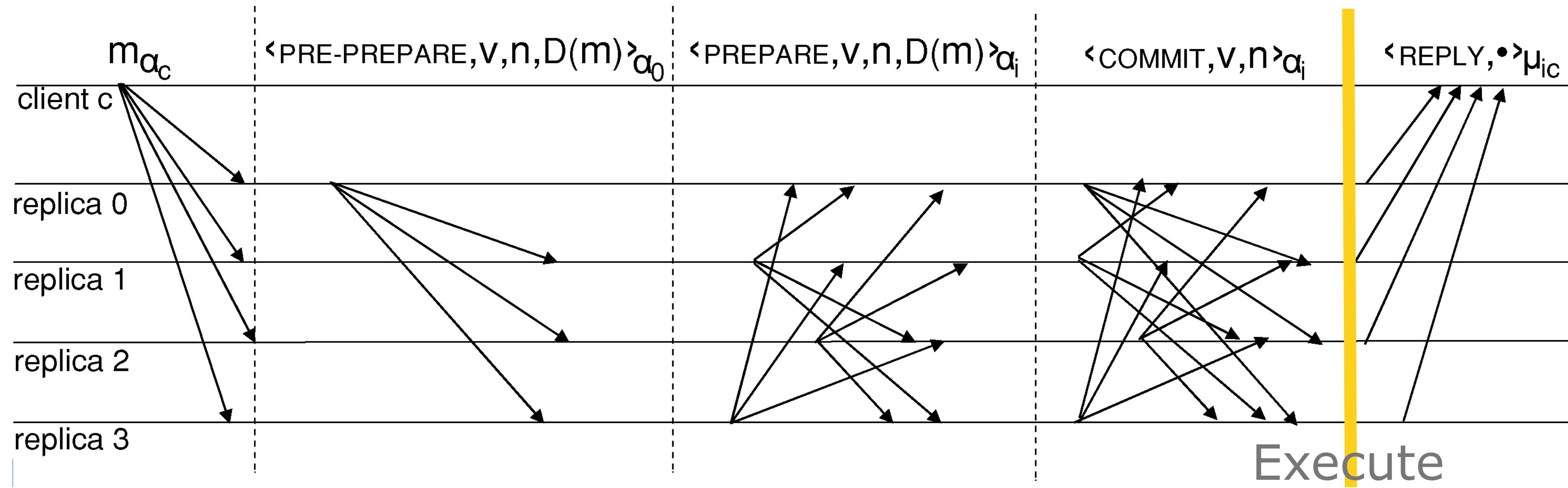
Prepared

$2f+1$ matching commits including own (quorum)

σ_i = replica i signature

lower order n operations have been performed

Reply



Backups reply to client

$$REPLY < v, t, c, i, r > \sigma_i$$

v = view number

i = replica id

r = result of o

σ_i = replica i signature

Log Checkpointing

Infinite log

Checkpoint Log state periodically

Digest of log state

Multicast digest of log state to other replicas

$2f+1$ digests of matching state is quorum

Checkpoint and discard older log entries

$CHECKPOINT < d(s), i > \sigma_i$

d = digest

s = log state

i = replica id

σ_i = replica i signature

View Changes

Faulty Primary

Primary deterministic $p = v \bmod N$

Views may overlap

Restart view by re-issuing pre-prepares from failed view.

View change request triggered by

Client Timeout rebroadcast request which starts replica timer

Replica Timeout generates view change request

If primary given by $v+1$ receives $2f$ view change plus own

$\text{VIEWCHANGE} < v + 1, n_s, C, P, i > \sigma_i$

v = faulty view number

n_s = sequence number of last stable checkpoint s

C = set of $2f + 1$ checkpoint messages

P = set of sets P_m for each m prepared at i with $n_m > n_s$

P_m = 1 preprepare + $2f$ prepare

m = request message

i = replica id

σ_i = replica i signature

New View

If primary given by $v+1$ receives $2f$ view change plus own

Multicasts new view to replicas

O set of pre-prepare messages

After last stable checkpoint

Contained in at least one P of view-change messages

Use no-op for sequence number gaps

$$NEWVIEW < v+1, V, O > \sigma_p$$

v = faulty view number

$$p = (v + 1) \bmod N$$

V = set of view change messages

O = set of pre-prepare messages

σ_p = primary replica signature

Performance

PBFT BFS about 10- 20% slower than NFS under normal operation
90% performance degradation for fault recovery view change.

Implementations

Tendermint <http://tendermint.com/docs/tendermint.pdf>

Aardvark 2009 http://usenix.org/events/nsdi09/tech/full_papers/clement/clement.pdf

Fast Byzantine Consensus 2006 <https://www.cs.utexas.edu/~lorenzo/papers/fab.pdf>

Ripple <https://ripple.com/consensus-whitepaper/>

Stellar 2015 <https://www.stellar.org/papers/stellarconsensusprotocol.pdf>

<https://www.bigchaindb.com/whitepaper/>

<https://www.hyperledger.org>

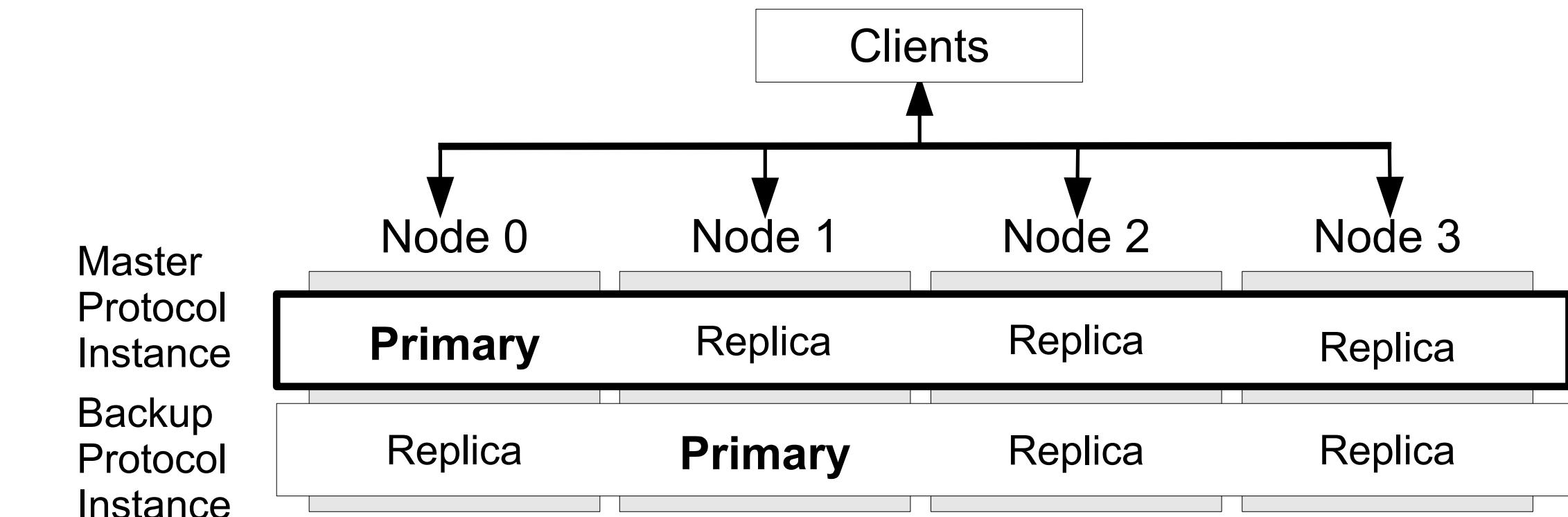
Redundant BFT (RBFT)

Redundant Byzantine Fault Tolerance 2013

<http://pakupaku.me/plaublin/rbft/report.pdf>)

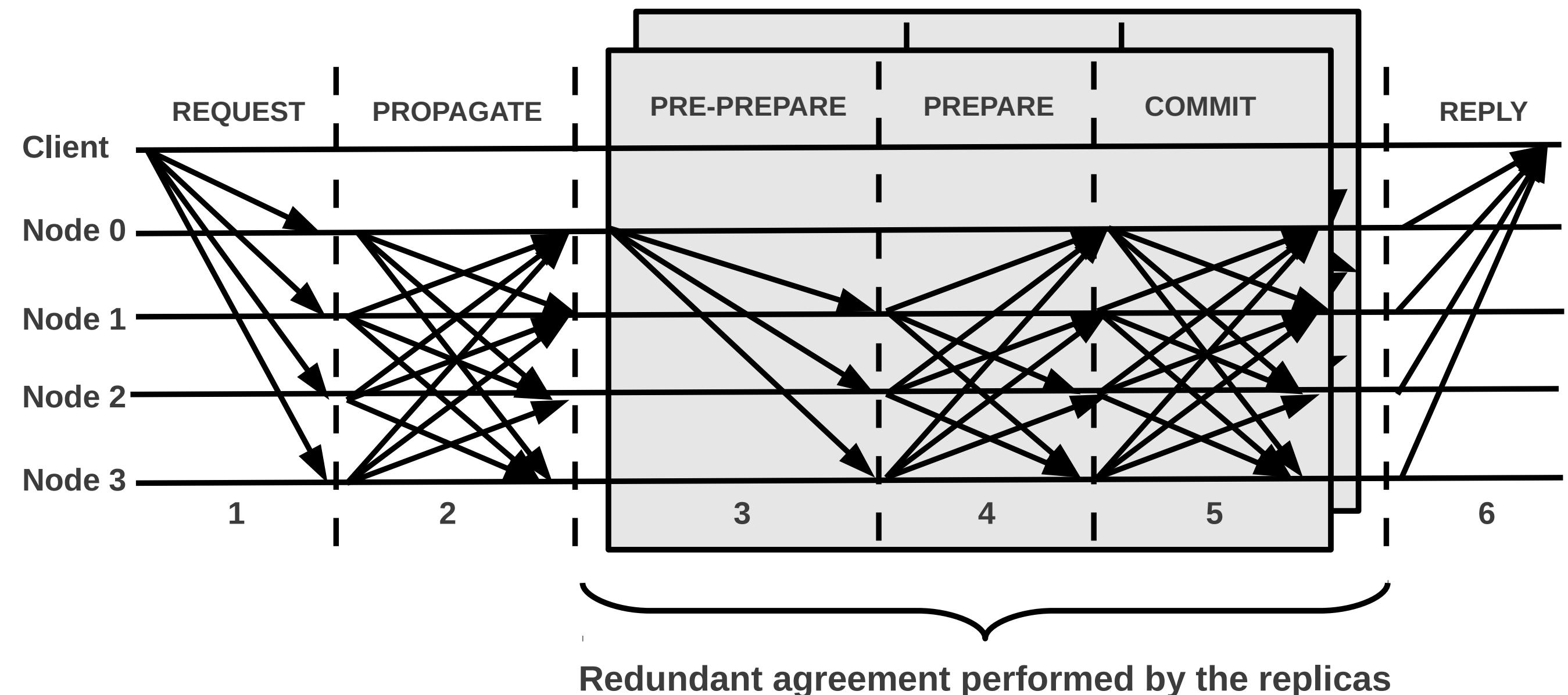
700 BFT Protocols, 2014

<http://www.eurecom.fr/~vukolic/tocs-700.pdf>



Little performance degradation for fault recovery, 5%.

About 10-20% lower performance under normal conditions.



Plenum RBFT Python Implementation

Plenum RBFT <https://github.com/evernym/plenum>

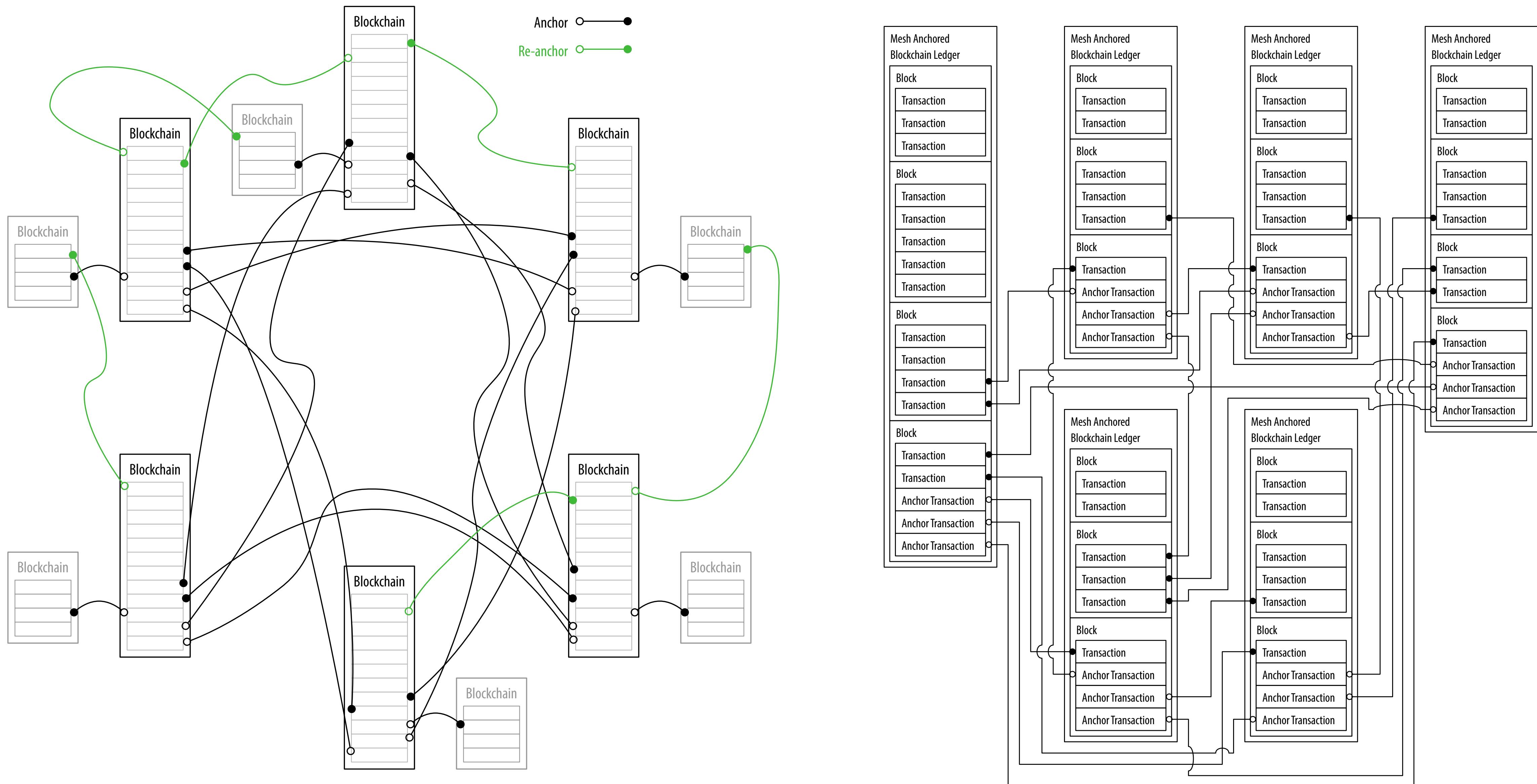
<https://github.com/evernym/plenum/wiki>

Uses RAET (Reliable Asynchronous Event Transport) protocol

<https://github.com/RaetProtocol/raet>

Digital Signatures (no Macs) so don't need digests.

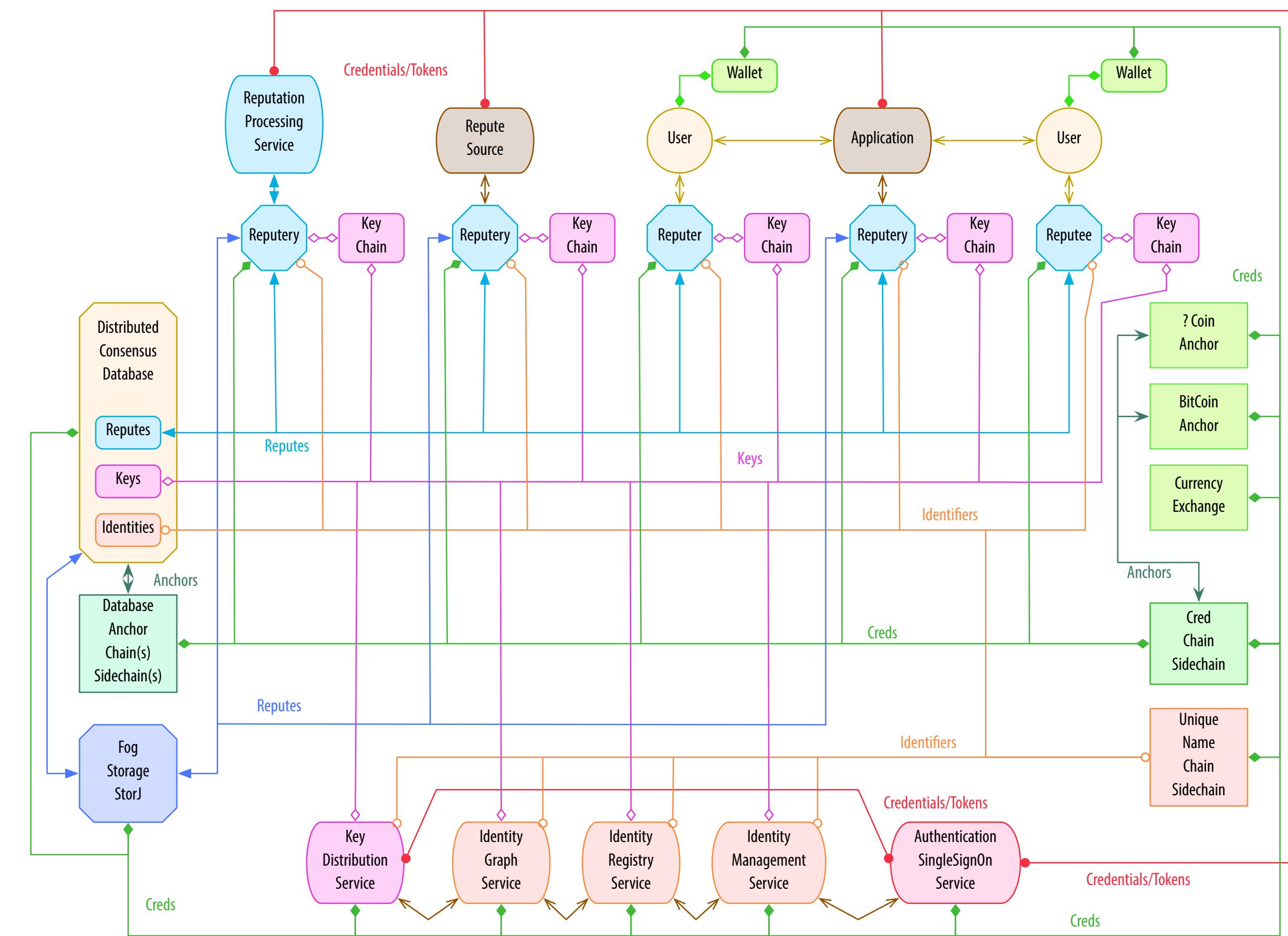
Chainmail or Mesh Anchoring



Reputation and Identity

<https://openreputation.net/open-reputation-low-level-whitepaper.pdf>

<http://evernym.com/assets/doc/Identity-System-Essentials.pdf?v=0825630653>



BACKUP SLIDES

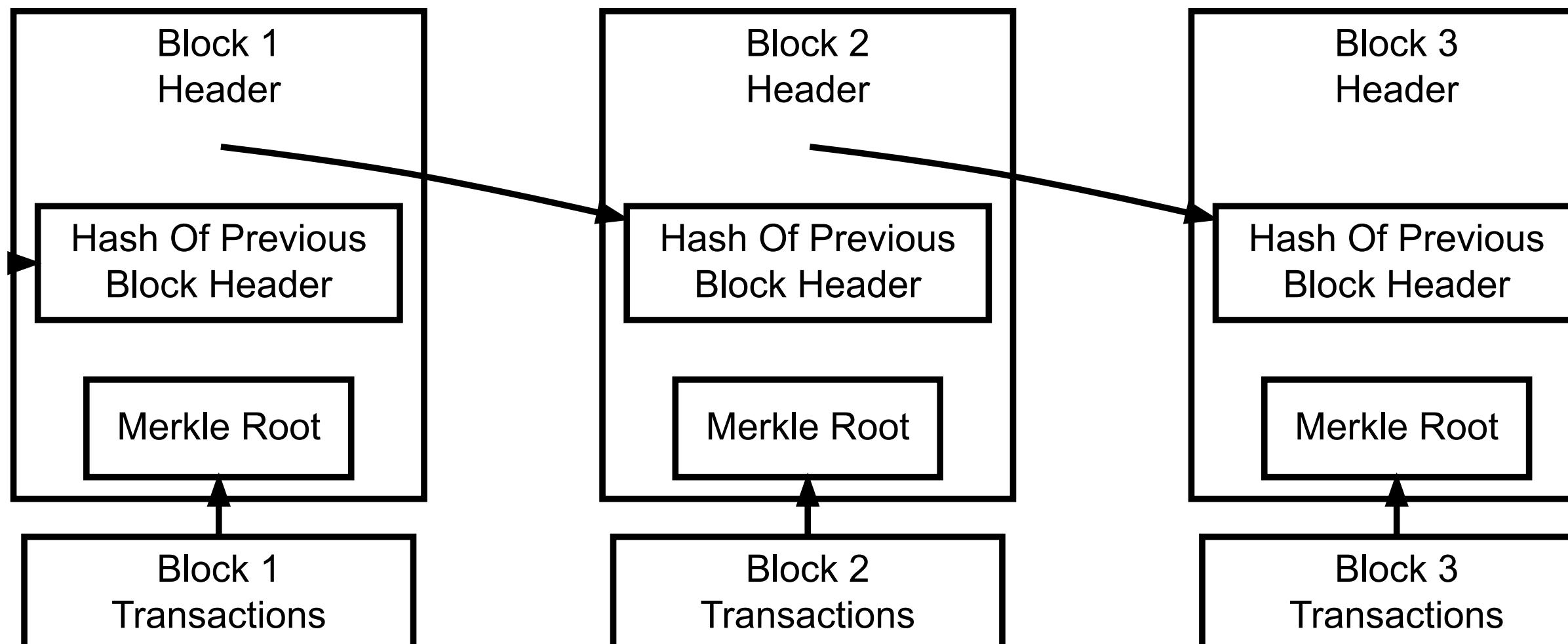
View Change

What if the primary is faulty?

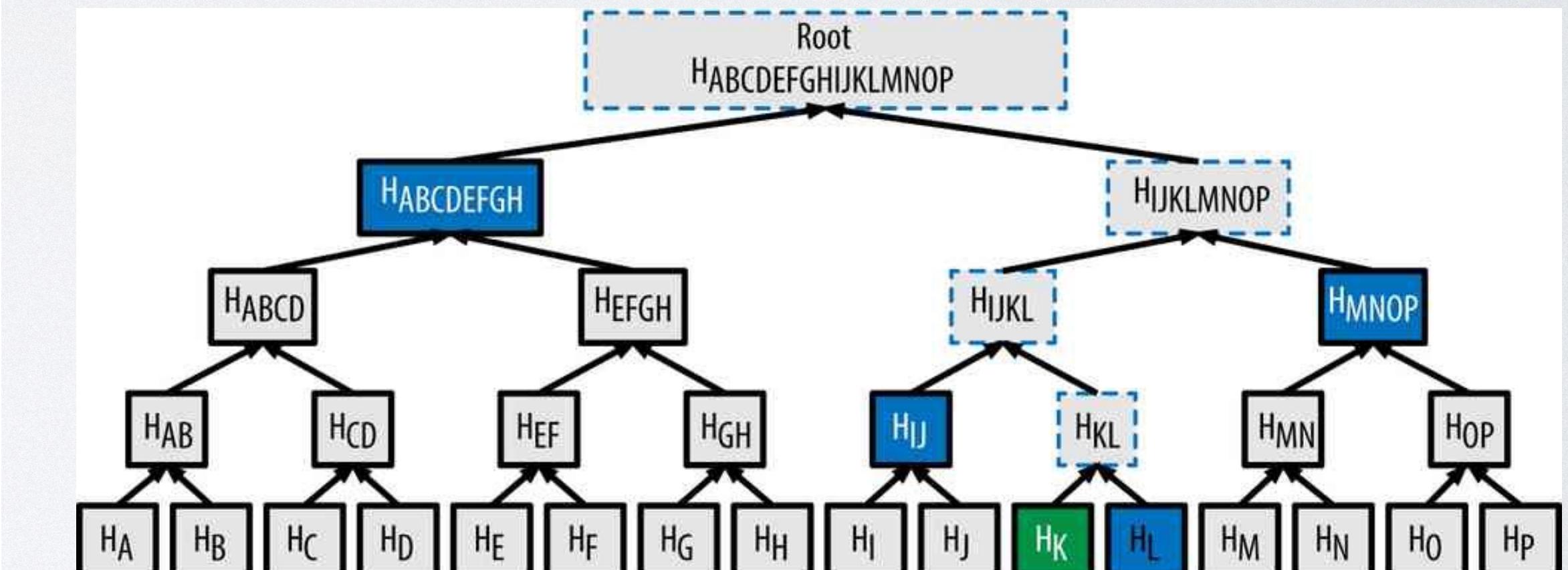
- The client uses a timeout. When this timeout expires, the request is sent to all replicas.
- If a replica already knows about the request, the rebroadcast is ignored.
- If the replica does not know about the request, it will start a timer.
- On timeout of this second timer, the replica starts the view change process.
- If a replica's timer expires, it sends a view change message.
- This message contains the system state (in the form of archived messages) so that other nodes will know that the replica has not failed.
- If the current view is v , node $v+1 \pmod n$ waits for $2f$ valid view-change messages.
- Once $v+1$ has seen $2f$ view-change messages, it multicasts a new-view message
- This message contains all the valid view change messages received by $v+1$ as well as a set O of all requests that may not have been completed yet (due to primary failure).
 - After a replica receives a valid view-change message, it enters view $v+1$ and processes O
 - While view change is occurring, no new requests are accepted.

BLOCKCHAIN I: CRYPTOCURRENCY

- BitCoin: A Peer-to-Peer Electronic Cash System <https://bitcoin.org/bitcoin.pdf>
- Solved in a logically simple but computationally inefficient way the double-spend problem on a distributed ledger with uncensored hosts
- Proof of Work: Do the work up front. Fastest worker (miner) wins. Solve cryptographic hash proof.
- Validation: All the workers (miners) validate. Majority rules.

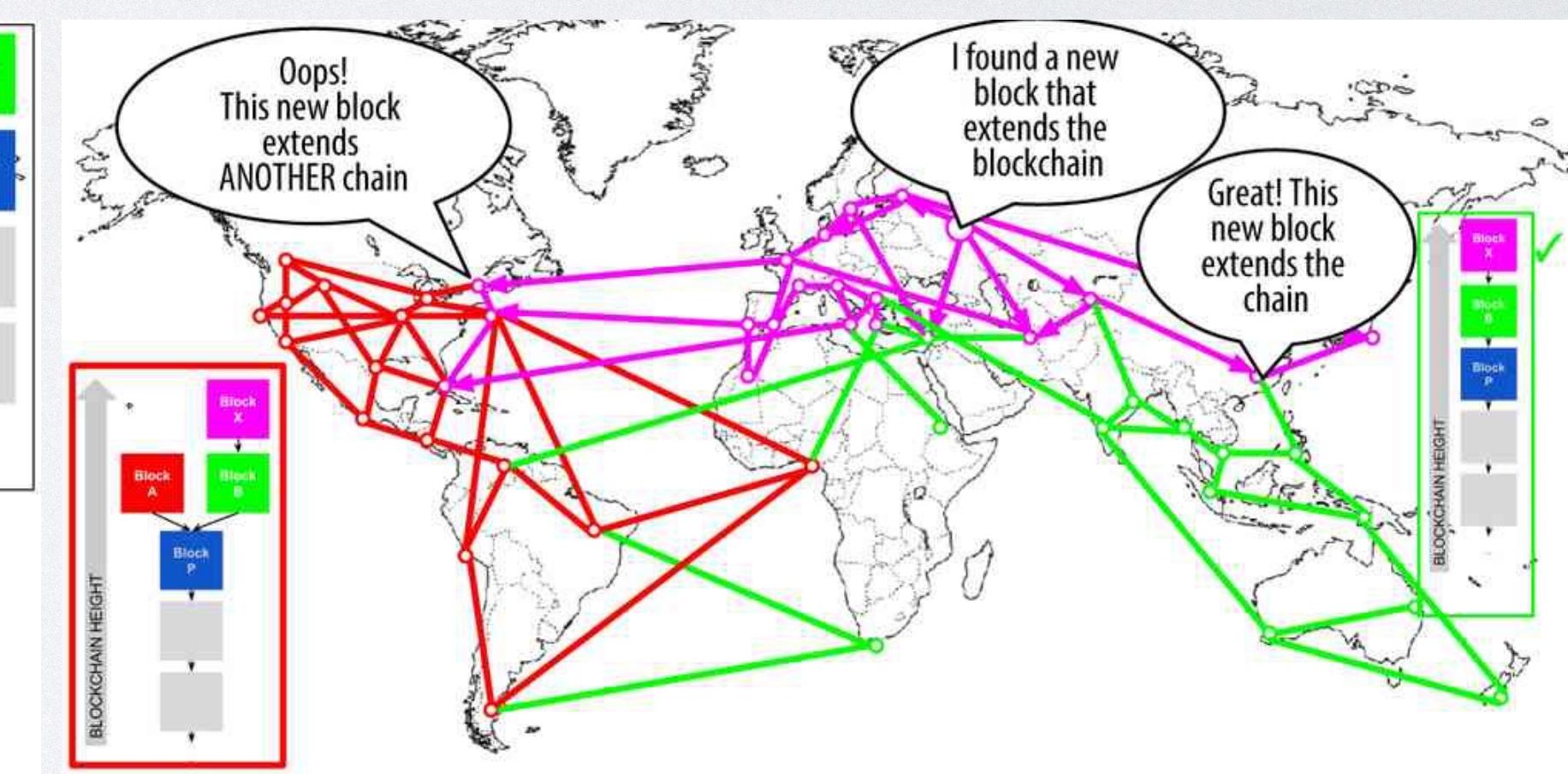
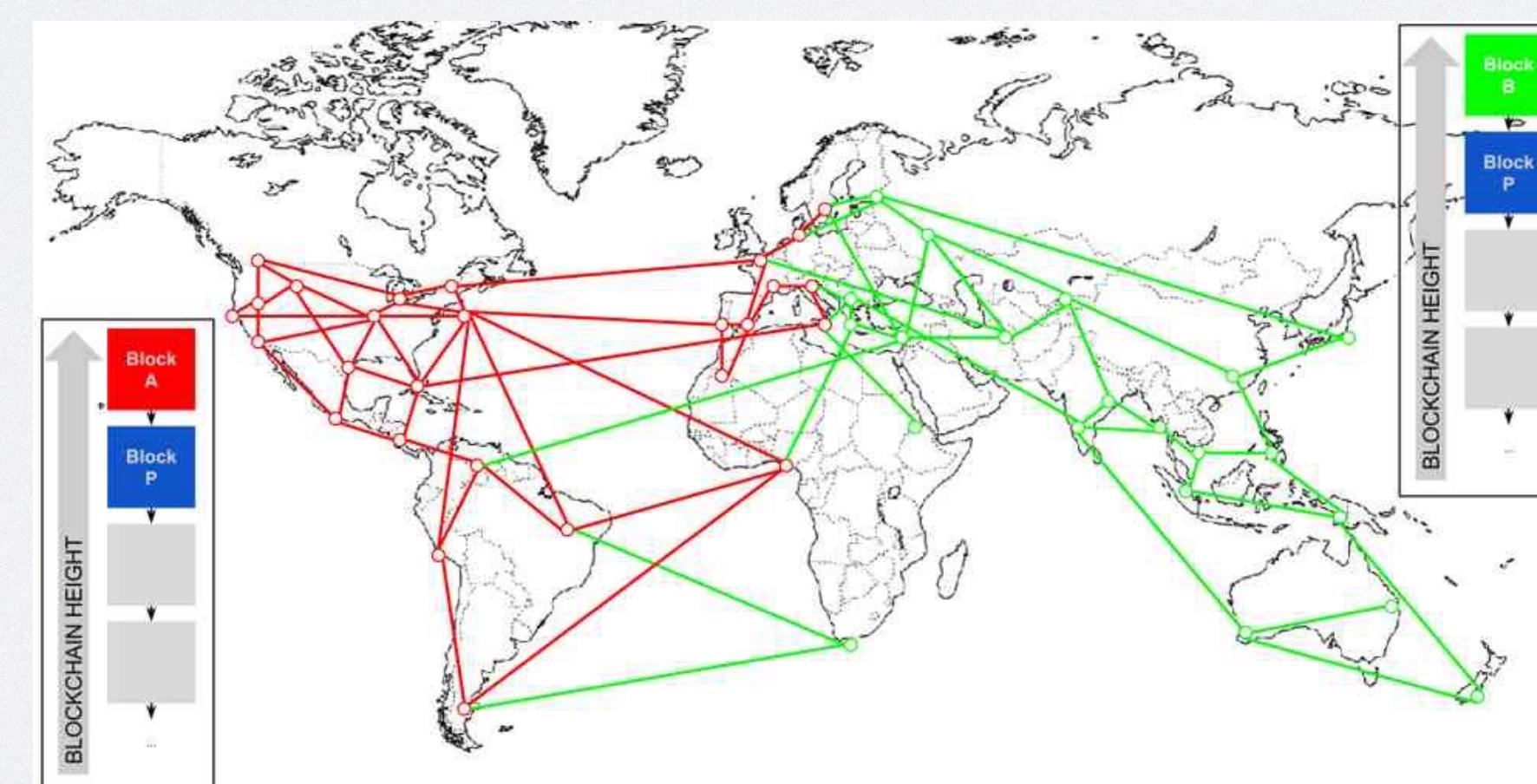
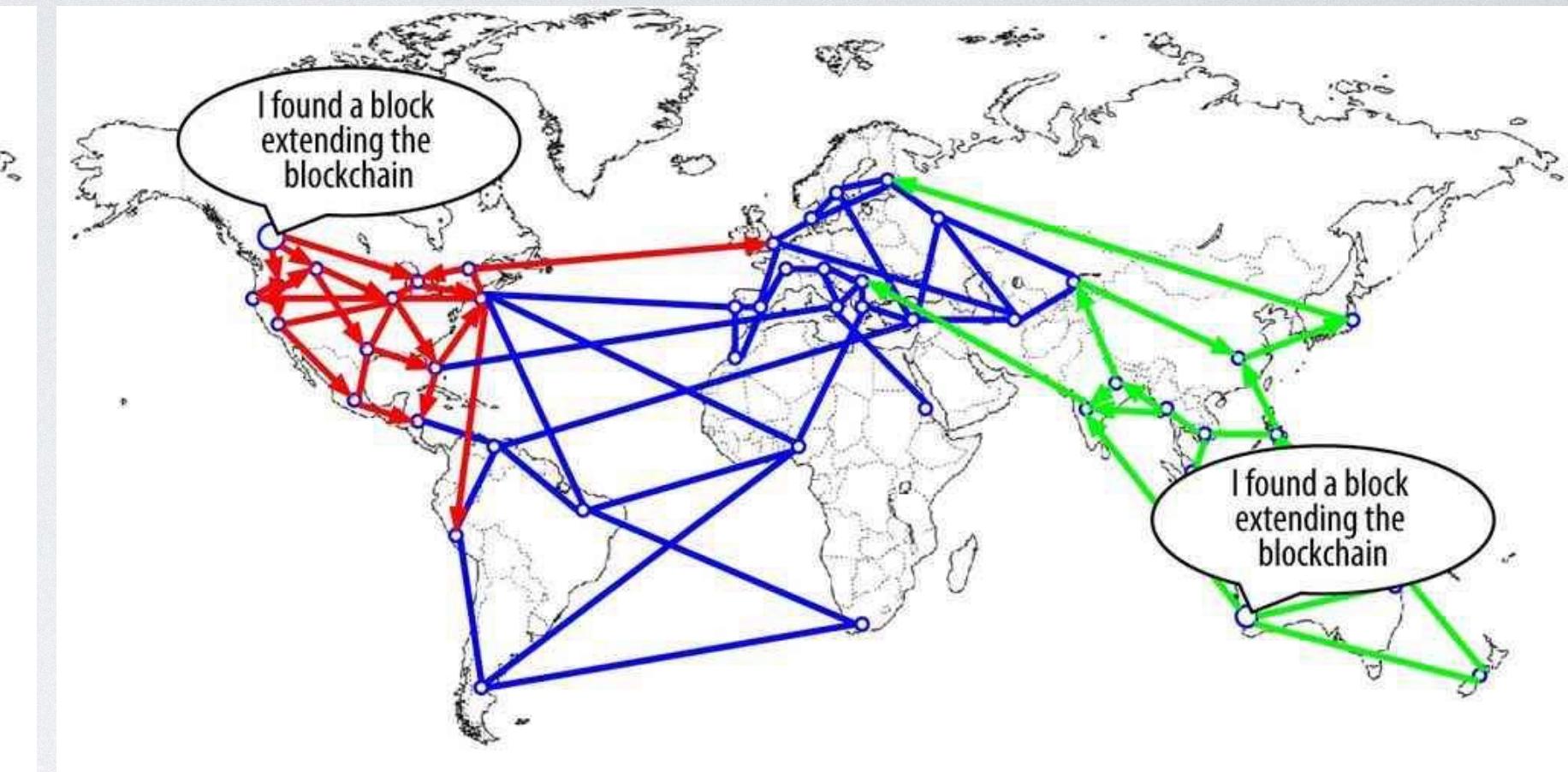
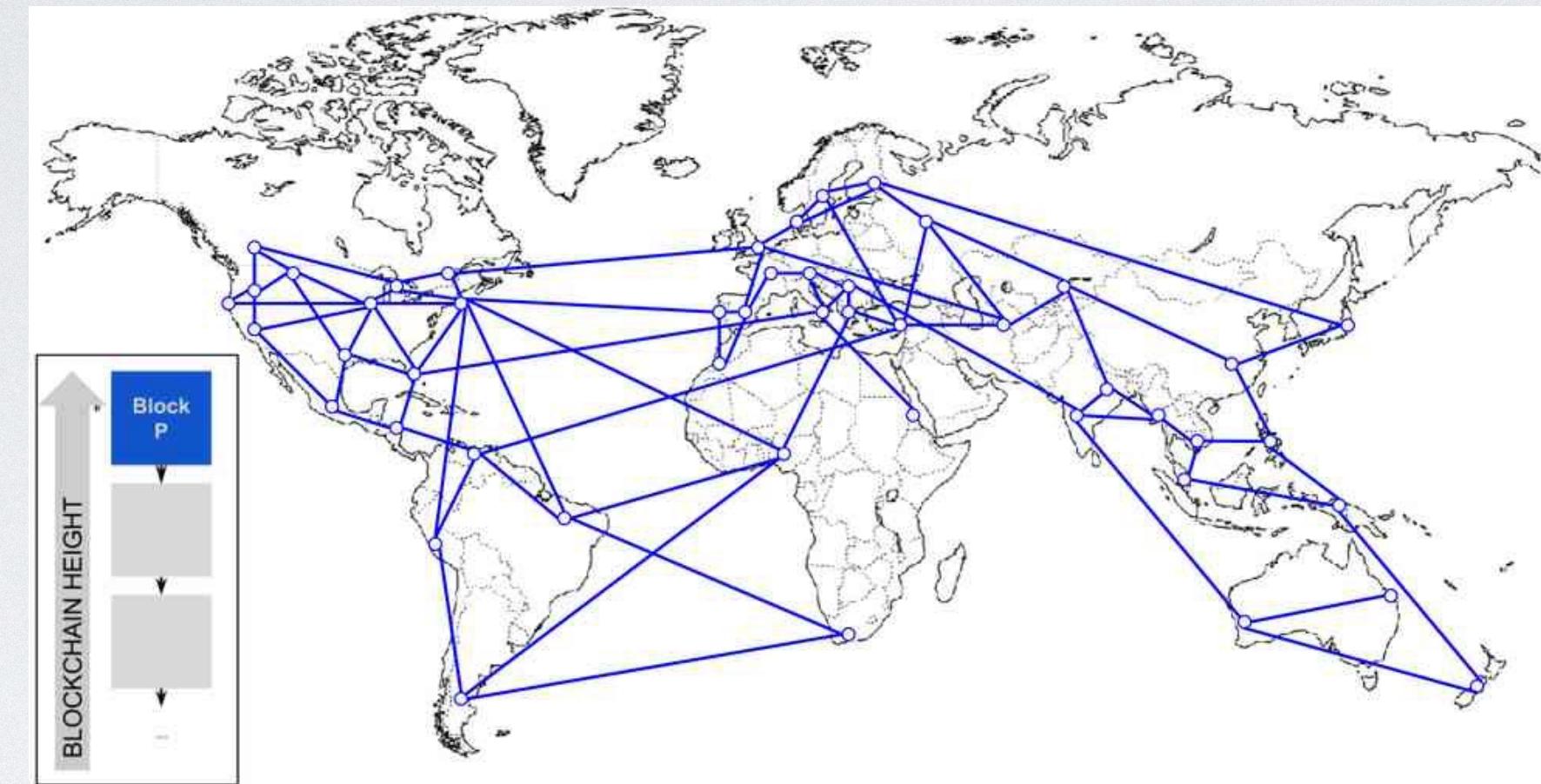
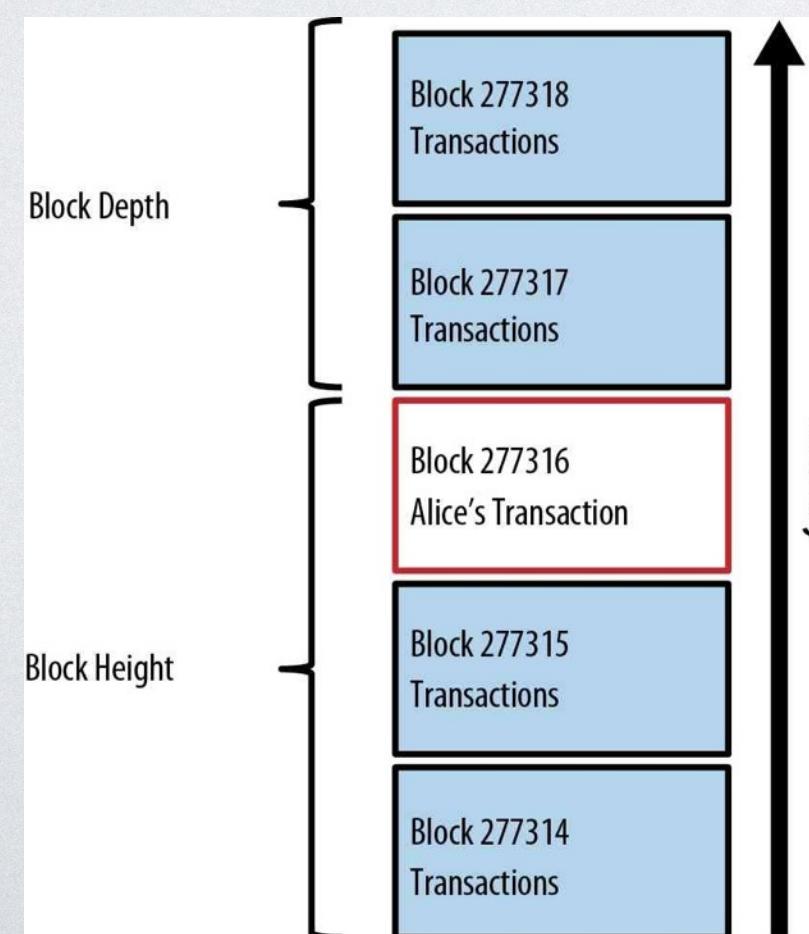


Simplified Bitcoin Block Chain



BLOCKCHAIN I: PROOF OF WORK

- Every new block requires proof of work.
- To change a block down in the chain requires reworking all the blocks above it.
- Deep enough blocks become exponentially difficult to change.
- In order to double spend, one must control a majority of the compute power to create a forked chain and then collude with spenders to reverse and then respond prior spends.
- Transactions made by other spenders cannot be double spent.
- Multi Billions of currency outside of any firewall. Safe from attack



ELLIPTIC CURVE CRYPTOGRAPHY

- Fast, efficient, most secure.
- Both encryption/decryption and signing/validation
- Public Private Key exchange
- Secret Sharing
- Key Blinding
- End to End Encryption with Authentication after Encrypt

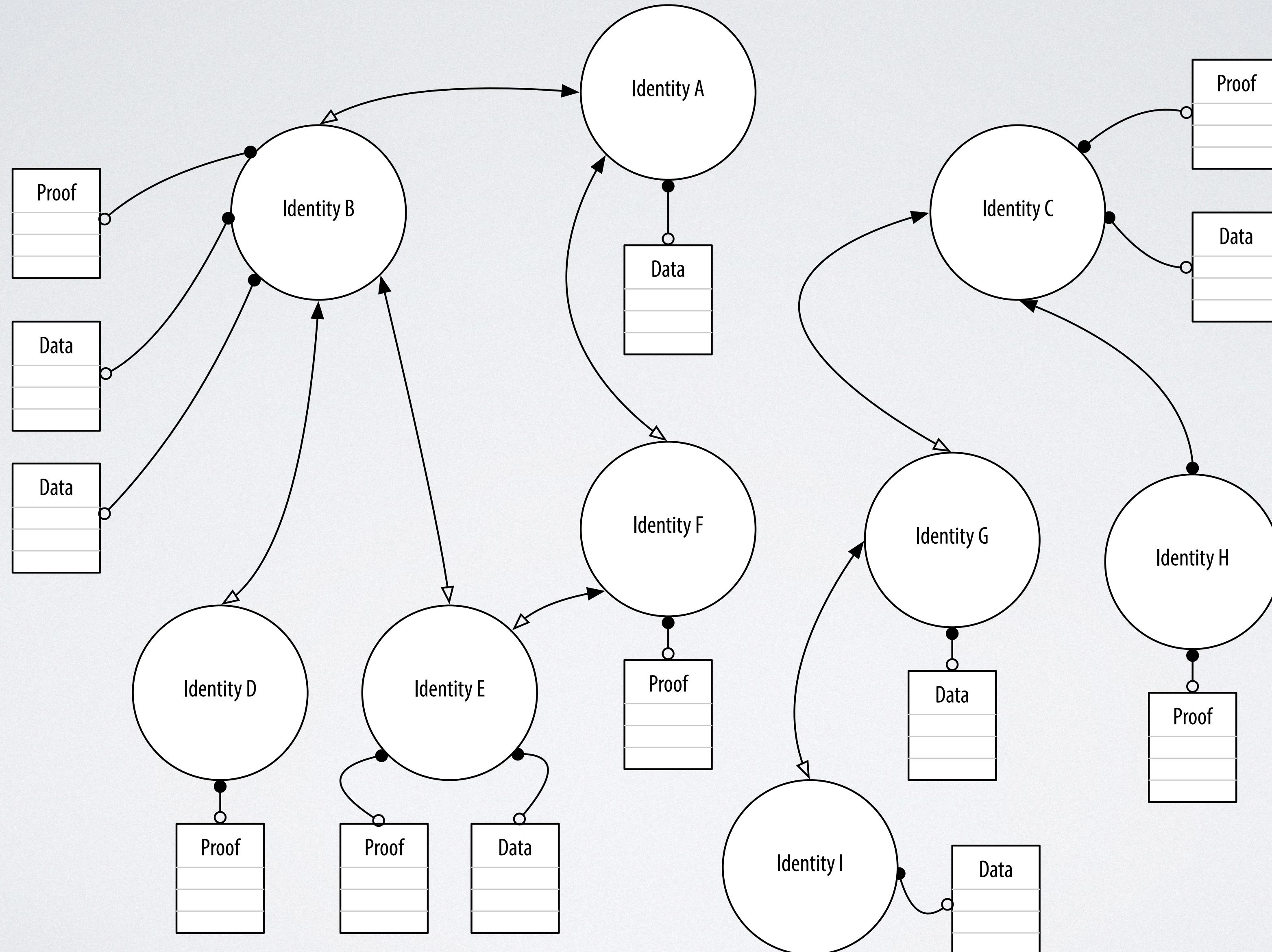
BLOCKCHAIN 2

- Smart Contracts:
 - Terms and conditions codified in the ledger.
 - Distributed consensus enforces terms and conditions
- Other assets managed by distributed consensus
- Identity managed by distributed consensus
- Distributed Autonomous Organizations and Services.

IDENTITY

- Entity: Uniquely Identifiable, Individual or Group
- Identifier: Symbol uniquely associated with an entity
 - Cryptonym: Globally unique cryptographic key
 - Aliases: Human friendly globally unique string
 - Email
 - Phone number
 - URL
 - Self certifying URL
- Proofs: Verifiable information that associates an identifier with an entity
- Data: Relevant information about entity
- Identity Graph: Relationships between identifiers, proofs, and descriptors
- Identity: Data structure: Identity graph, proofs, data
- Hierarchical Deterministic Cryptonyms

IDENTITY GRAPH



PRIVACY

- Three Degrees of Privacy:
 - Full Public: Public (non-anonymous) Identifiers and Public (unencrypted) Descriptors/Proofs
 - Semi-Private: Public (non-anonymous) Identifiers and Private (encrypted) Descriptors
 - Full Private: Private (anonymous) Identifiers and Private (encrypted) Descriptors
- Group Privacy: Class or Attribute Identifiers
 - Identifier associates entity as a member of a group of entities essentially anonymizing the entity
 - Allows sharing of data without correlation
 - Principle only share enough data to enable the transaction and no more
- Controllability, Observability, Traceability

KEY MANAGEMENT

- Multi-Signature
- Hierarchical Deterministic Keys
- Key Recovery
- Secret Sharing
- Group Keys

