Data Consistency and Blockchain

Bei Chun Zhou (BlockChainZ)
beichunz@cn.ibm.com

Data Consistency

- Point-in-time consistency
- Transaction consistency
- Application consistency

Strong Consistency

- ACID
  - Atomicity. All of the operations in the transaction will complete, or none will.
  - Consistency. The database will be in a consistent state when the transaction begins and ends.
  - Isolation. The transaction will behave as if it is the only operation being performed upon the database.
  - Durability. Upon completion of the transaction, the operation will not be reversed.

CAP theorem

- A distributed computer system CANNOT simultaneously provide:
  - Consistency
    - all nodes see the same data at the same time
  - Availability
    - every request receives a response about whether it succeeded or failed
  - Partition tolerance
    - the system continues to operate despite arbitrary partitioning due to network failures
- "2 of 3" is misleading!
  - Partitions are rase, so detecting partition failure is more valuable
  - The choice is mainly between C and A
  - CAP are not binary, 0 vs. 1

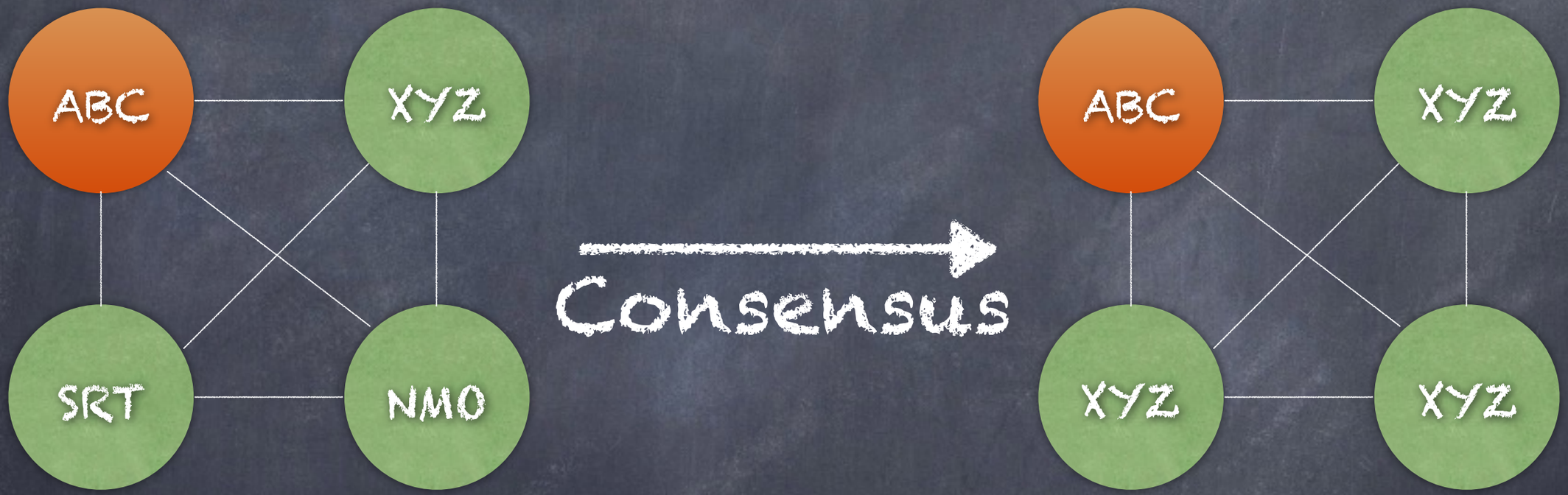
Eventual Consistency

- Based CAP
  - ACID provides Consistency for partitioned database, but give ups availability to some extent as tradeoff
  - In ACID, product availability is lower than components availability
- BASE (An ACID alternative, diametrically opposed)
  - "Trading some consistency for availability can lead to dramatic improvements in scalability"
  - What makes up BASE?
    - Basically Available
    - Soft state
    - Eventual consistency

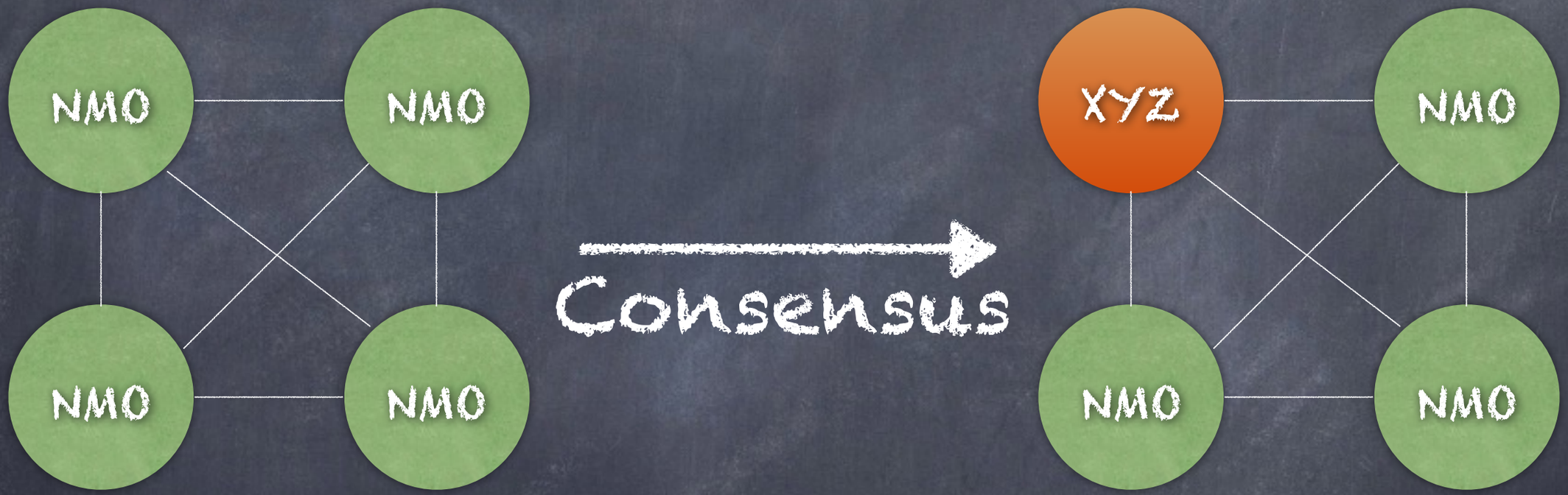
Consensus

- A process of agreeing on one result among a group of participants to guarantee data consistency

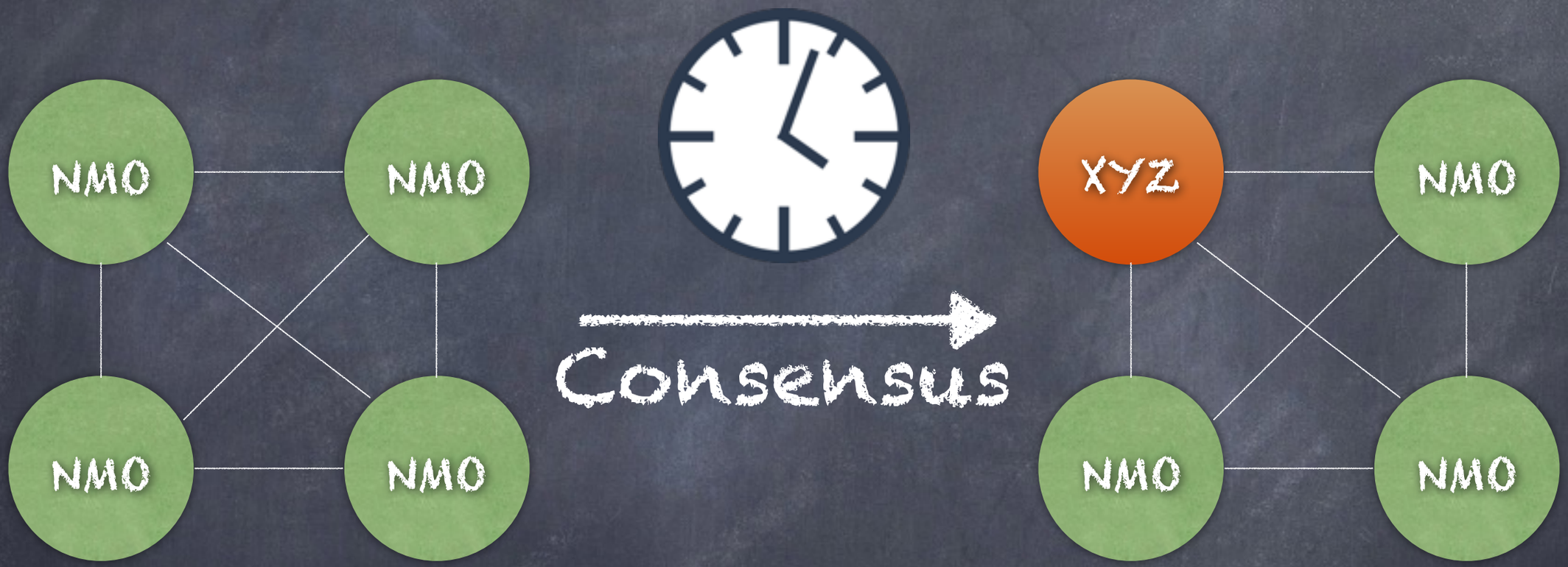
AGREEMENT



VALIDITY



TERMINATION



Consensus Models and Challenges

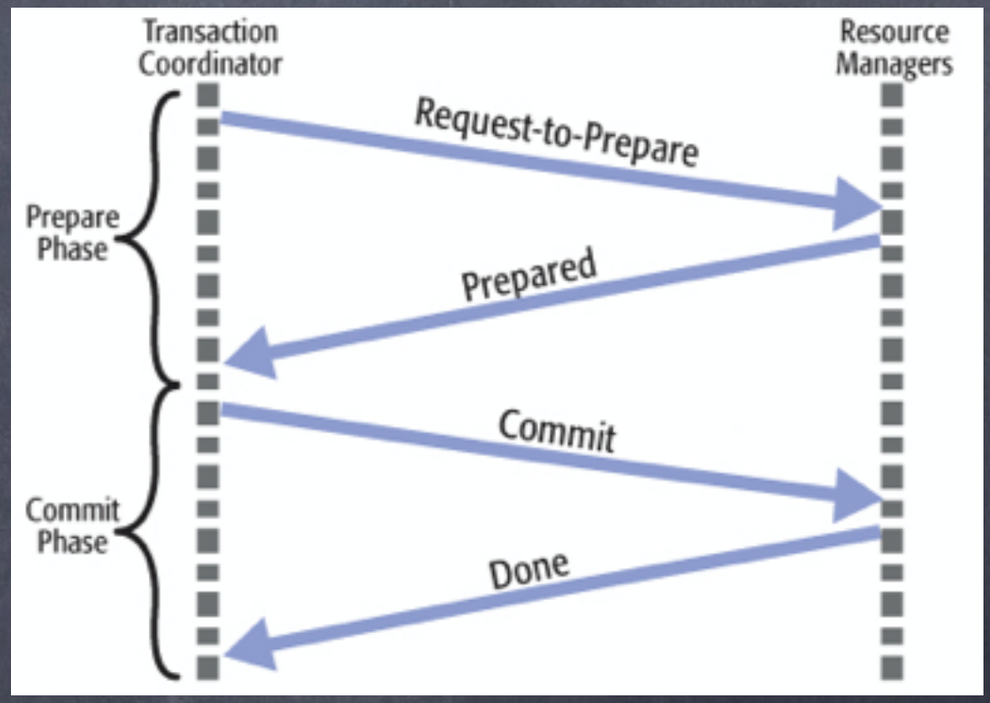
- Synchronous and Asynchronous models
  - Timing control
  - Leader
- Challenges
  - Crash failure
  - Byzantine

FLP Impossibility

- FLP (Fischer, Lynch and Paterson)
  - "No completely asynchronous consensus protocol can tolerate even a single unannounced process death."
    - It is not saying that consensus can never be reached: merely that no algorithm can always reach consensus in bounded time.
- Inspire discussion of:
  - Safety: something bad will never happen
  - Liveness: Something good will eventually happen

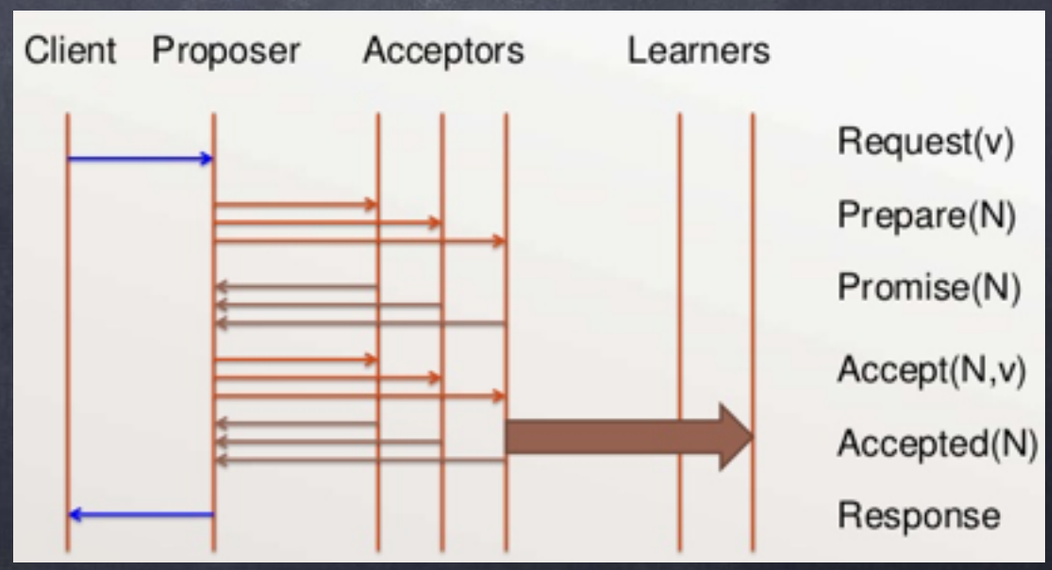
Consensus - 2PC

- 2PC (Two Phase Commit)
  - Agree on a decision for transaction atomic
  - Coordinator ensures Safety by rollbacking transactional update
  - The failure of Coordinator blocks the consensus and thus heuristic action needs to be designed.
  - 3N-1 message and 3 force writes to log, probably most efficient consensus protocol



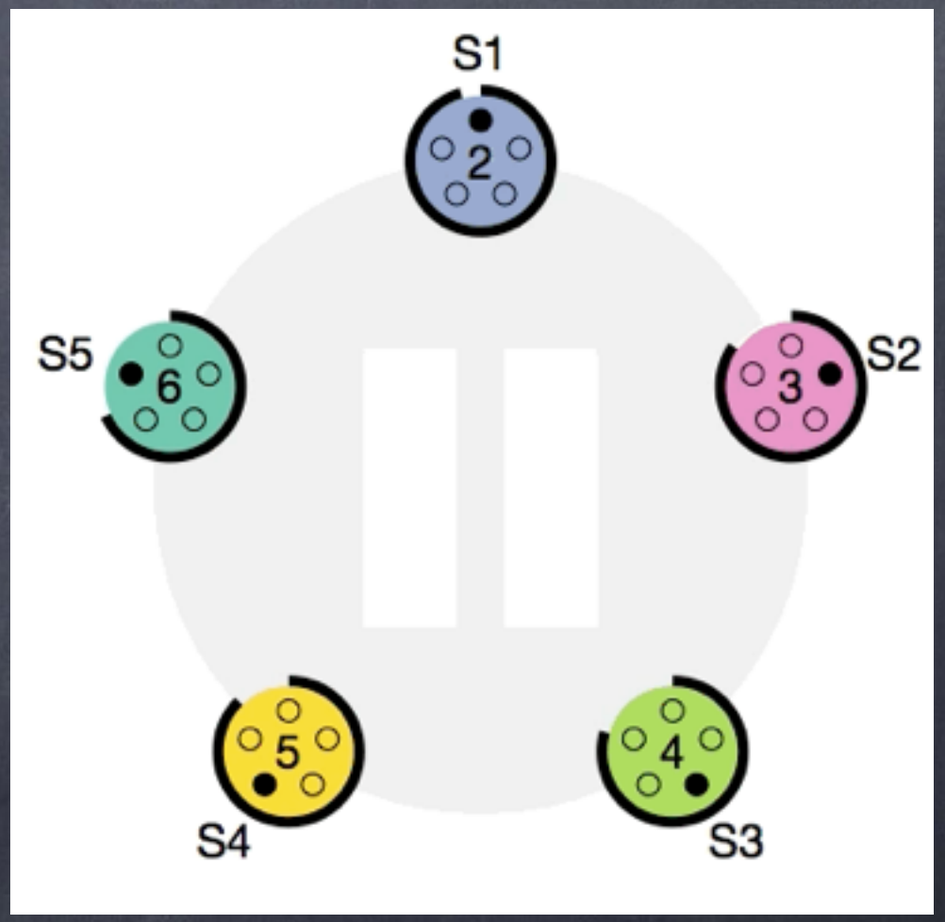
Consensus - Paxos

- Paxos
  - "Paxon Parliament": Agree on a value or a decision
  - Customary asynchronous, non-Byzantine model with three roles and two phases
    - Proposer, Acceptor and Learner
    - Prepare, Accept
- Tolerate non-malicious Failure among 2F+1 acceptor
- Non-blocking via leader election
- Multi-Paxos to facilitate a seriers of decisions



Consensus - Raft

- Raft
  - Equivalent to Paxos in terms of Performance
  - But better Understandability and Completeness
    - Leader election
      - Heartbeats and random timeouts
    - Log replication
      - Strong leadership: receive command from Client, append log and replicates its log to followers
    - Safety
      - Only leader can decide to COMMIT
- Roles: Leader, Follower, Candidates



"One Lie is enough
to question all
truths"

Anonymous

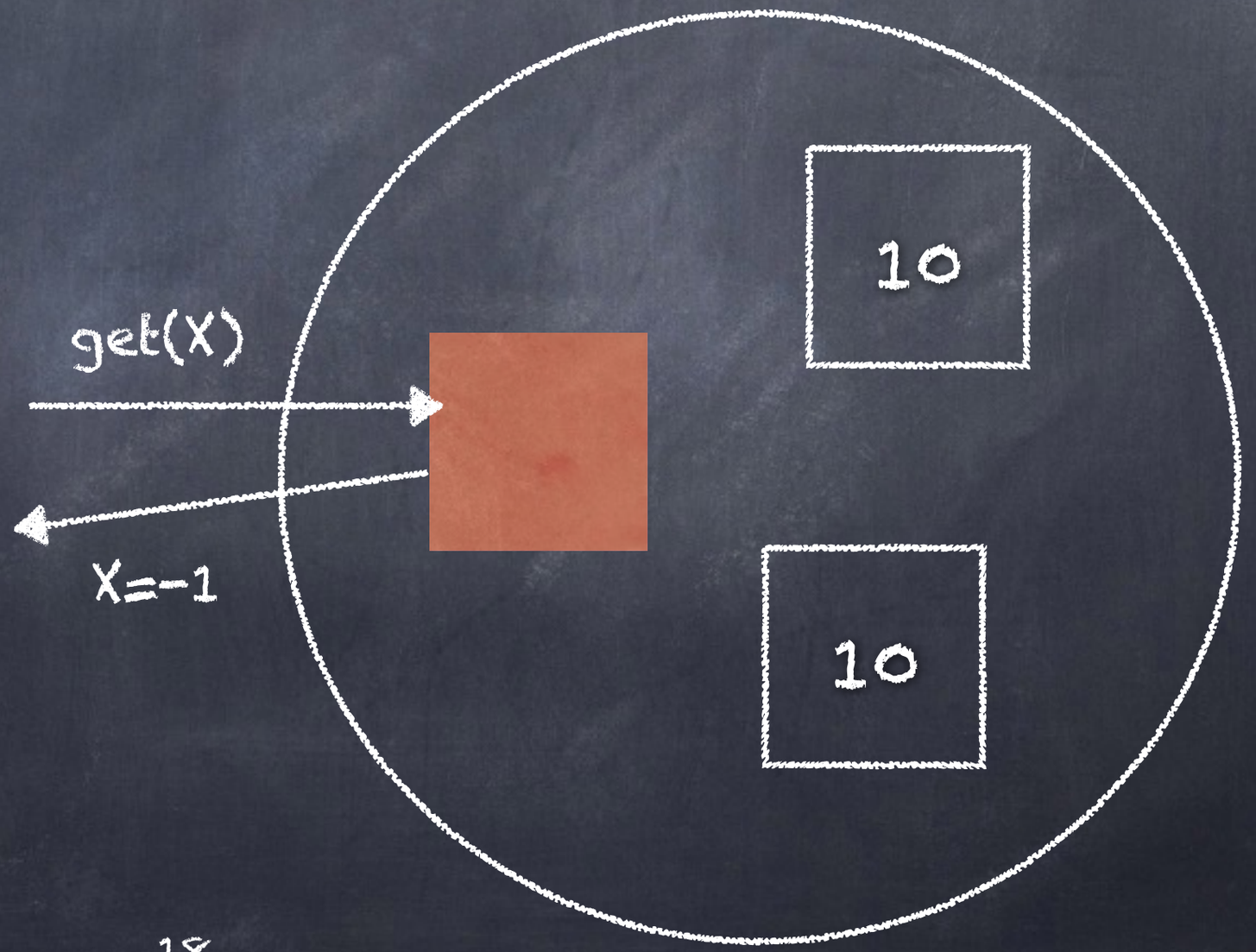
Byzantine Empire (height AD 555)

Consensus - PBFT

- Byzantine fault
  - Failure to return a result
  - Return of an incorrect result
  - Return of a deliberately misleading result
  - Return of a differing result to different parts of the system
- Pratical Byzantine Fault Tolerance (PBFT)
  - "A big step for Byzantine Fault"
  - Further research
    - e.g. Q/U, HQ, Zyzzyva and ABsTRACTs to lower costs and improve performance, Aardvark
    - RBFT to improve robustness.

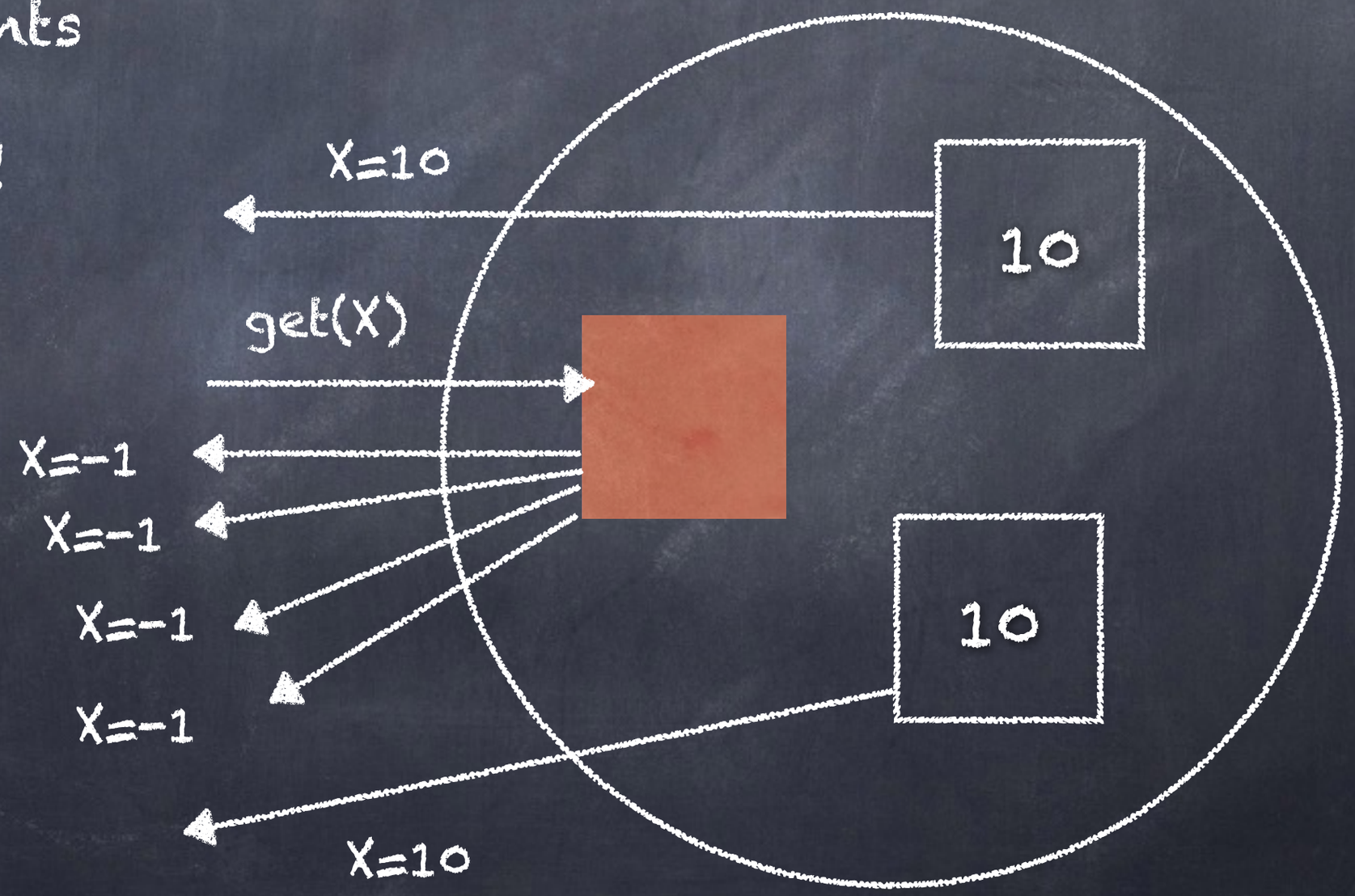
PBFT Cont.

- Malicious Primary?
  - Might Lie!



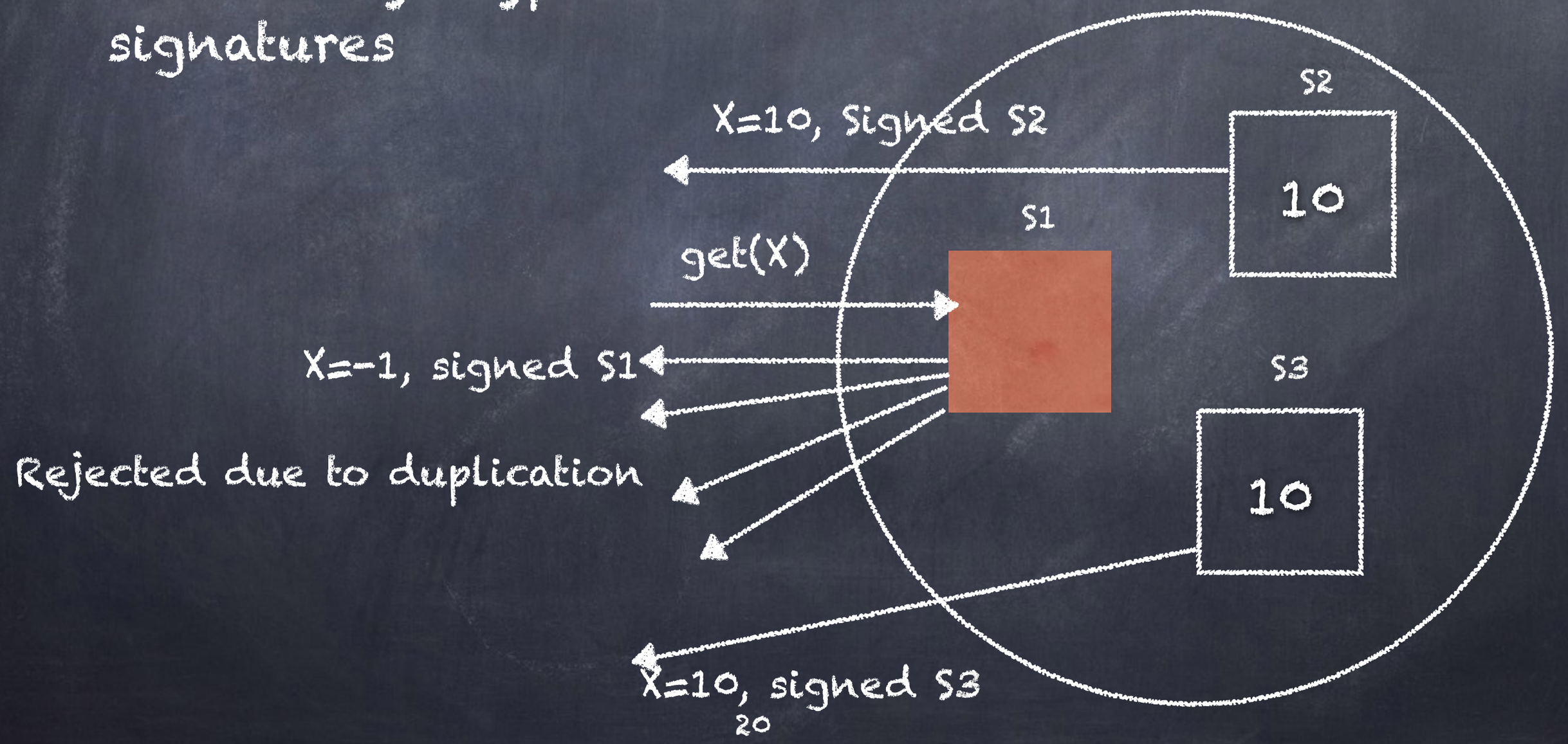
PBFT Cont.

- Malicious Primary?
  - Solution: direct response from participants
- Might lie more!



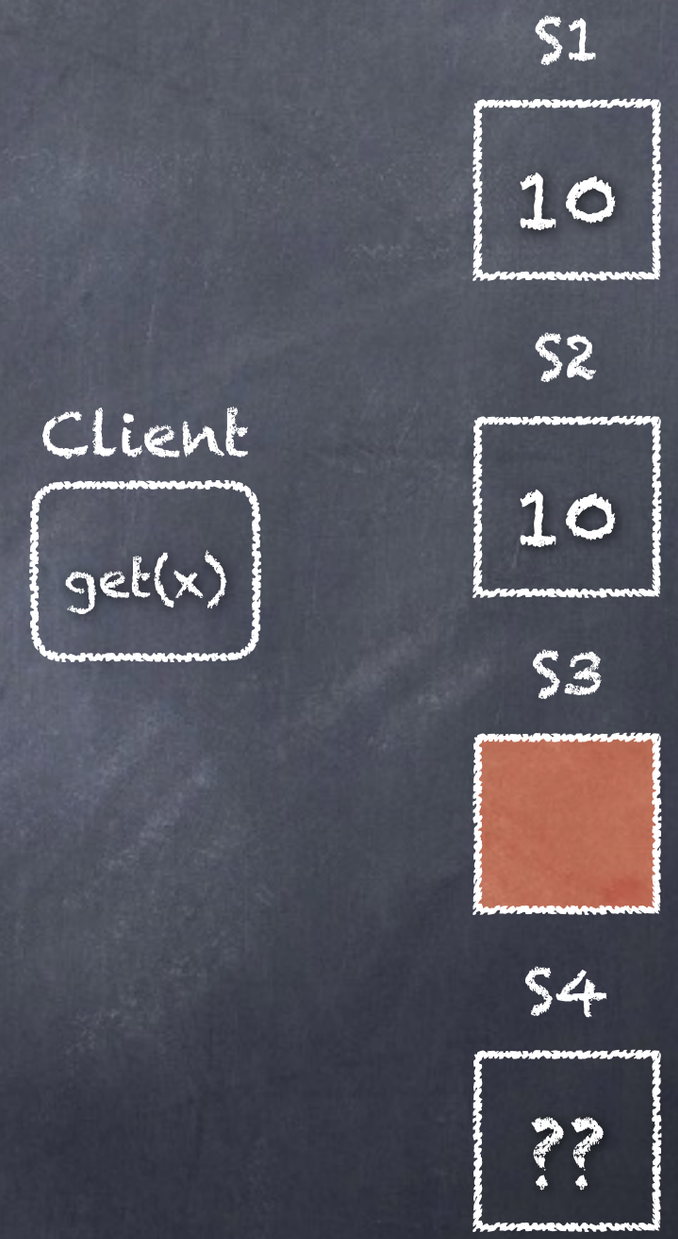
PBFT Cont.

- Ok, please sign your name!!!
  - Public-key crypto for signatures



PBFT Cont.

- 3f + 1?
  - At f + 1 we can tolerate f failures and hold on to data.
  - At 2f + 1 we can tolerate f failures and remain available.
- What do we get for 3f + 1?
  - We can tolerate f malicious or arbitrarily nasty failures

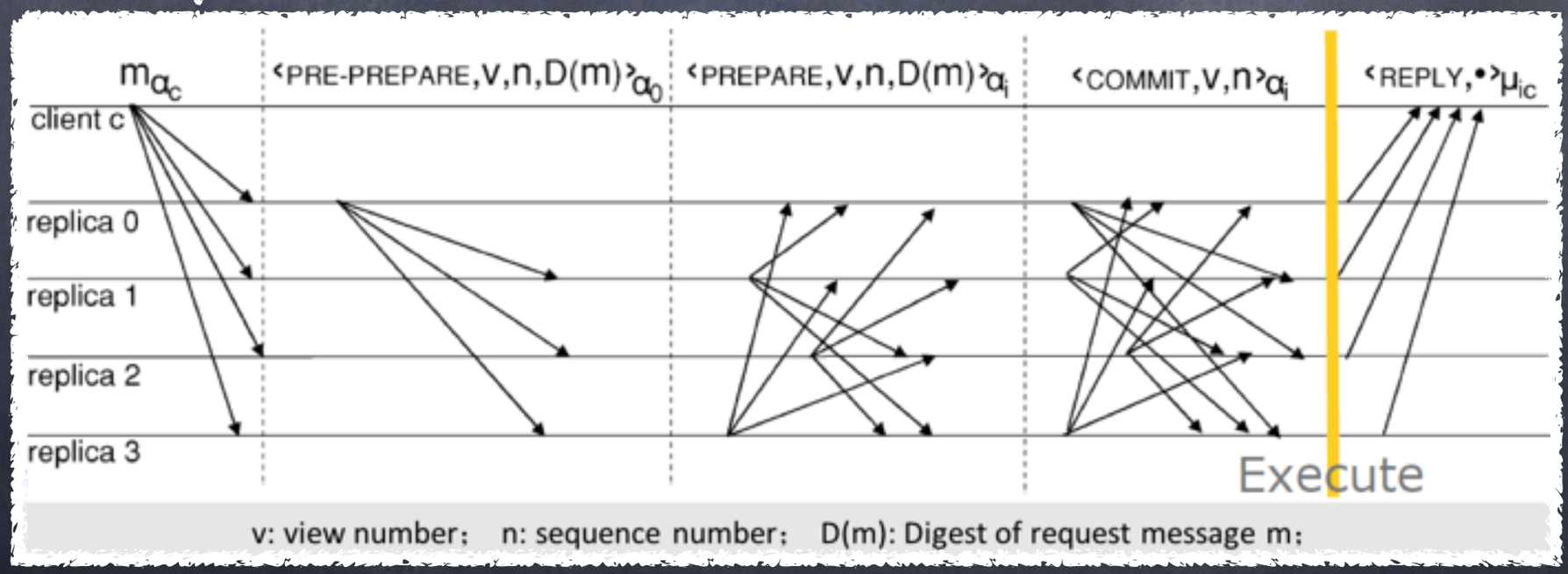


PBFT Cont.

- Three Phases Commit for agreement
  - Pre-Prepare, Prepare and Commit
    - Pre-prepare and Prepare: quibble about order
    - Prepare and Commit: atomicity, and thus safety
- Checkpoint for Garbage Collection
  - log is a must to guarreetee Safety, and GC is a must to manage log
- View Changes for crashed or malicious leader
  - Triggered by replicas' timeout
  - Provide Liverness by Re-electing a new leader (thus new view)

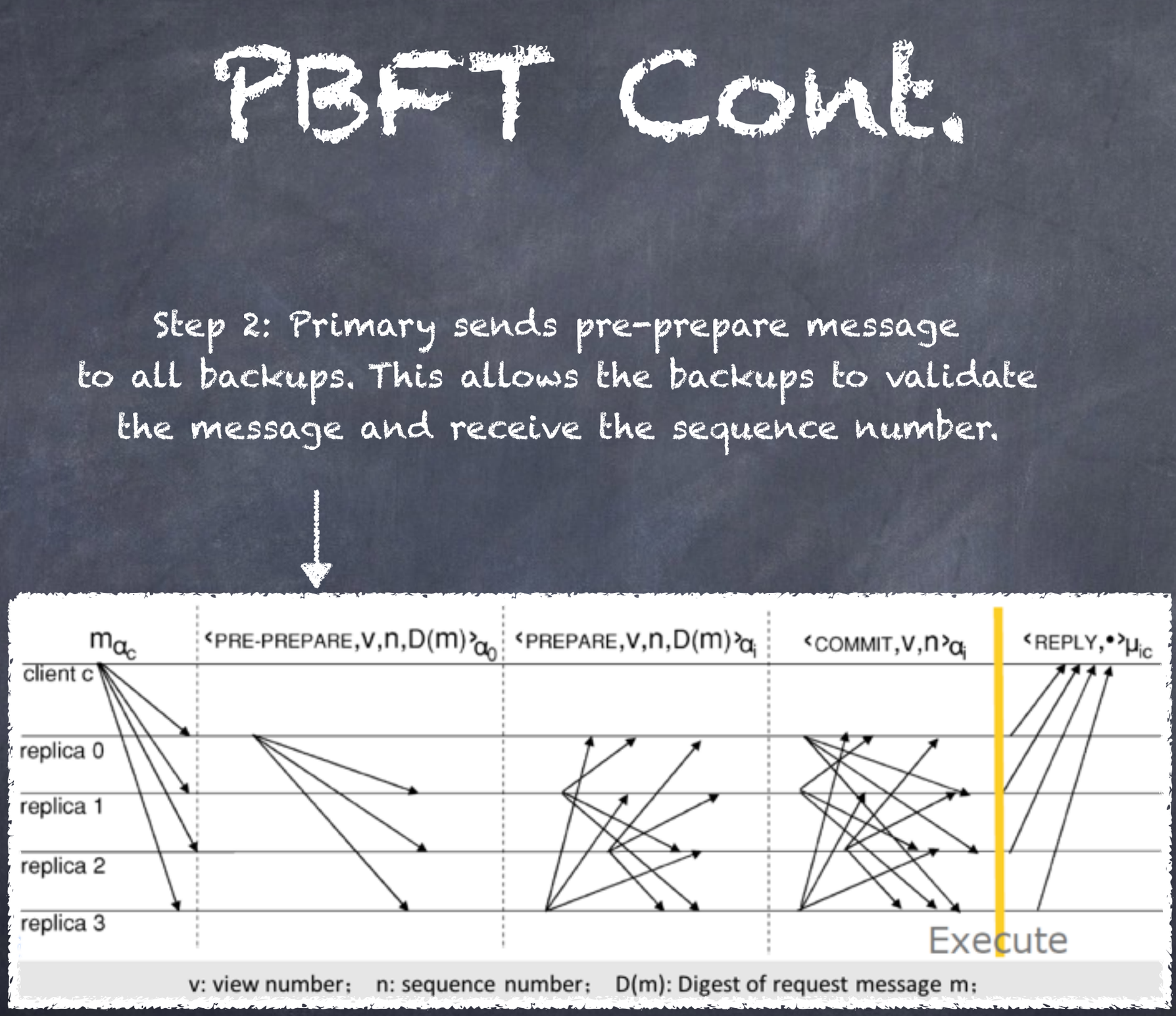
PBFT Cont.

Step 1: Client sends a request to the primary.
The primary can then validate the message and
propose a sequence number for it.



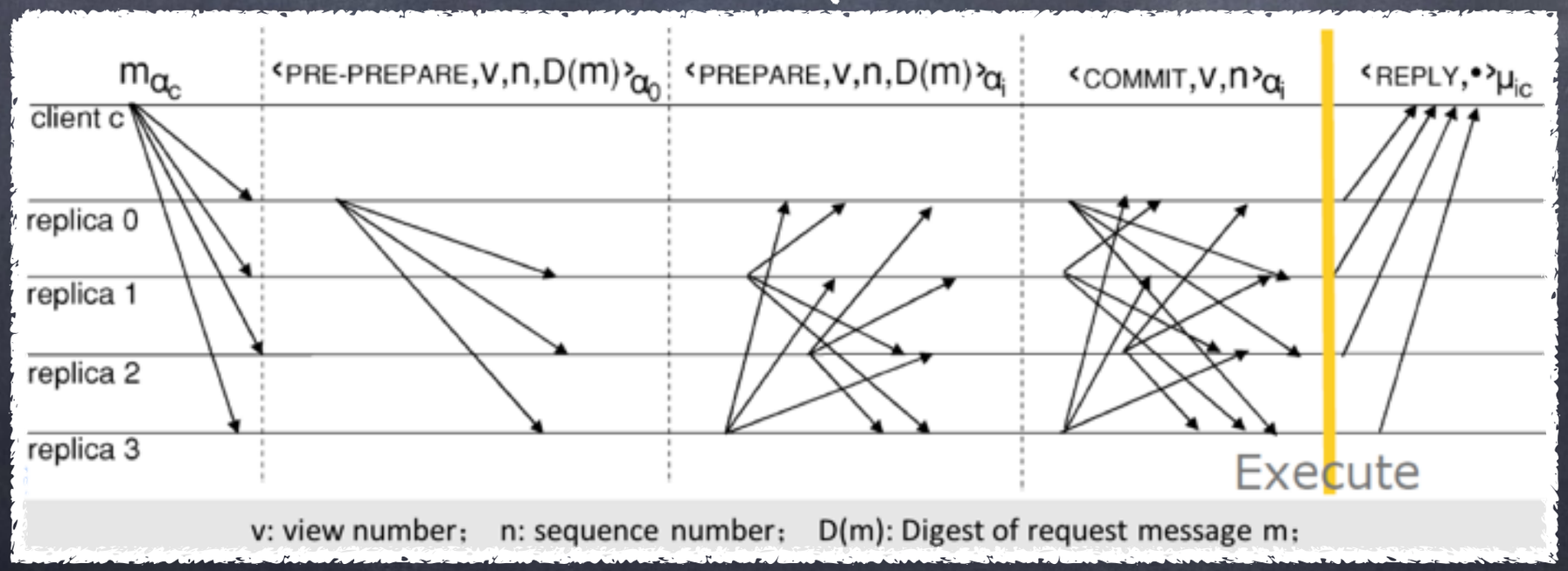
PBFT Cont.

Step 2: Primary sends pre-prepare message
to all backups. This allows the backups to validate
the message and receive the sequence number.



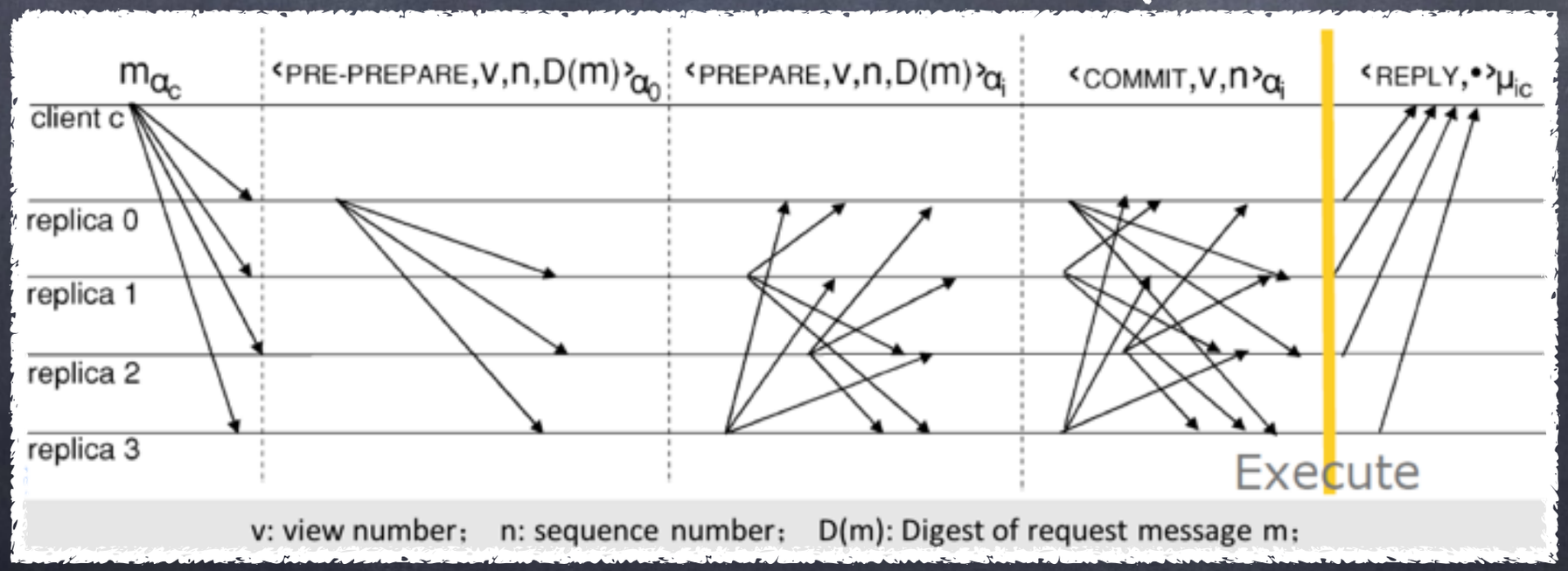
PBFT Cont.

Step 3: All functioning backups send prepare message to all other backups. This allows replicas to agree on a total ordering.



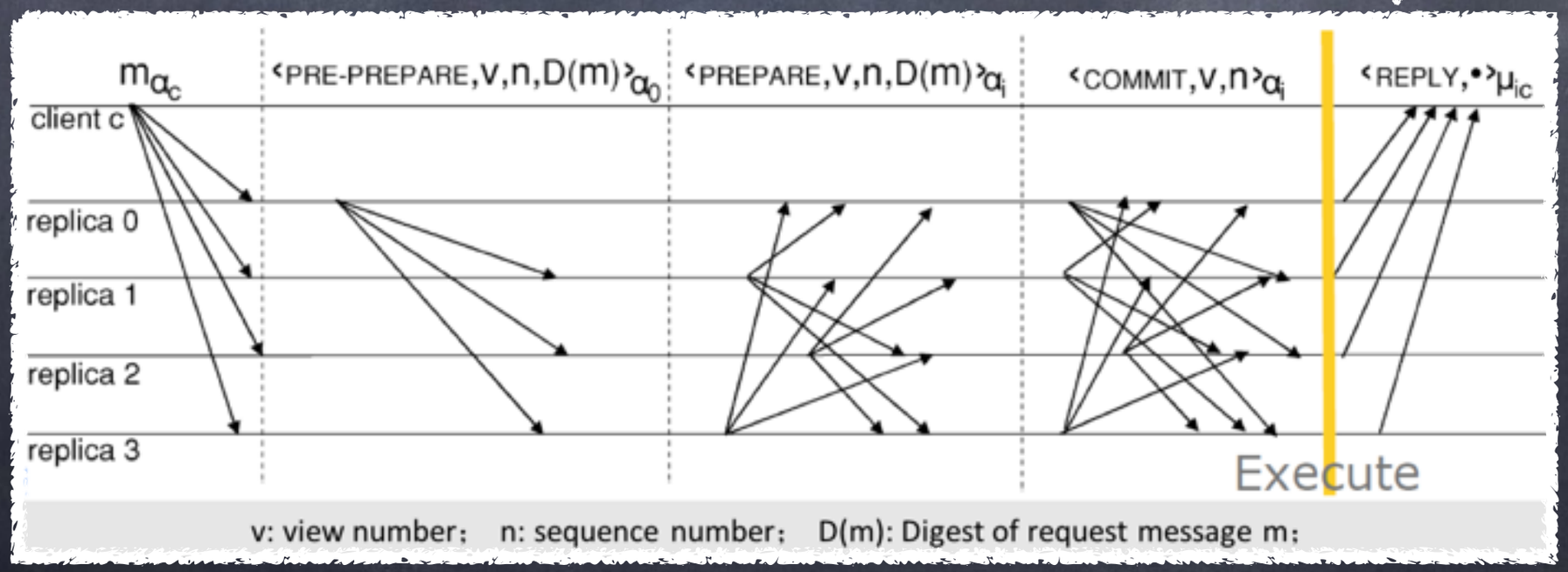
PBFT Cont.

Step 4: All replicas multicast a commit. The replicas have agreed on an ordering and have acknowledged the receipt of the request.



PBFT Cont.

Step 5: Each functioning replica sends a reply directly to the client. This bypasses the case where the primary fails between request and reply.



PBFT Cont.

- Summary of PBFT
  - MODEL : Safe under asynchrony but needs periods of synchrony to proceed (relies upon message timeouts).
  - PROS : Uses a leader, and is fast while network synchronous and leader is correct. Complete system sequential writes.
  - CONS : Leader reelected on difficulties. A Byzantine leader can fwd messages on edge timeout window and cause massive slow down. The adversary can attack the network to make correct leader be seen as faulty and cause endless leader election.

SEIVE

- PBFT is Order-then-Execute
  - Atomic broadcast forms bottleneck
  - Output is NOT deterministic (same operations generate same output)
- Sieve is Execute-then-Order with a designated leader
  - Operation requests are sent to leader as INVOKE
  - Leader orders the requests, and asks peers to EXECUTE operations in order (batch multiple operations like Eve)
  - Each peer executes operation speculatively, and send output back to leader as APPROVE
  - Commit if at least f+1 same output among at least 2f+1 APPROVE, otherwise abort, and decision is broadcasted as an ORDER
  - start-epoch/NEW-SIEVE-CONFIG for selecting new Sieve leader

Consensus - RPCA

- Ripple Protocol Consensus Algorithm (RPCA)
  - Design Goal: Correctness, Agreement and Utility
  - Consensus at ledger-close
    - Form transaction "candidate set"
    - Amalgamates the set to UNL (Unique Node List) and vote on the veracity of all transactions
    - Reject transaction with a minimum percentage by voting (aviod impact from slow nodes)
    - Final round consensus requires 80% agreement among UTL
    - Apply "agreed" transactions to ledger and the ledger is closed as new last-closed ledger (represent the state of the network)
- Tolerate f <= (n-1)/5
- Carefully selected UNL to avoid fork and to guarantee correctness

Public Blockchains require Openness

Consensus - POW

- Proof Of Work in Bitcoin:
  - Problem: Double Spend electronic coin in a p2p network
  - Implicit Consensus for block generation
    - Random Miner consumes huge computing resource to mine next block by seeking a nonce for a hush puzzle
      - A block is then validated and accepted by others
      - POW difficulty is moving to control black rate
  - Majority decision is represented by the longest chain
  - Essentially one-CPU-one-vote, and thus attacher needs 51% CPUs
  - Sacrifice safety to achieve openness and liveness
    - Eventually consistency, 10m for block generate, hours to confirm
    - forks make thing worse
- Ethereum customised POW to consume storage as the work instead.

Steps of Bitcoin network

- New transactions are broadcast to all nodes
- Each node collects new transactions into a block
- Each node works on finding a difficult proof-of-work for its block
- When a node finds a proof-of-work, it broadcasts the block to all nodes
- Nodes accept the block only if all transactions in it are valid and already spent
- Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash

Consensus - POS and others

- Thoughts about POW
  - Do we really need to waste so much computing power?
  - Is there better ways to make an individual miner finds the task expensive?
- Proof Of Stake (POS)
  - Alternative to POW proposed by Quantum Mechanic
  - The probability of mining a block depends on the amount of Bitcoin a miner holds
- Proof of Burn (POB)
  - Alternative to POW and POS
  - Miners should show proof that they burned some coins - that is, sent them to a verifiably unspendable address

A Byzantine summary...

	Safety	Liveness	Openness	Fault Tolerance	Throughput	Consumption
2PC	Good	Week	No	Participant crash	Good	Low
Paxos	Good	Ok	Week	f/2f+1 crash	Good	Medium
Raft	Good	Ok	Week	f/2f+1 crash	Good	Medium
PBFT	Good	Ok	Week	f/3f+1 Byzantine	ok	High Bandwidth
Sieve	Good	Ok	Week	f/3f+1 Byzantine	Good	Better than PBFT via Eve
RPCA	Ok	Ok	Week	f <= (n-1)/5 Byzantine	Good	High Bandwidth
POW	Week	Good	Good	49%	Week	High CPU
POS	Week	Good	Good	49%	Week	Low

"Understand the requirement,
decide the tradeoff,
Plug-in Consensus"

-BC's suggestion, 2016 July