Consesus algorithms

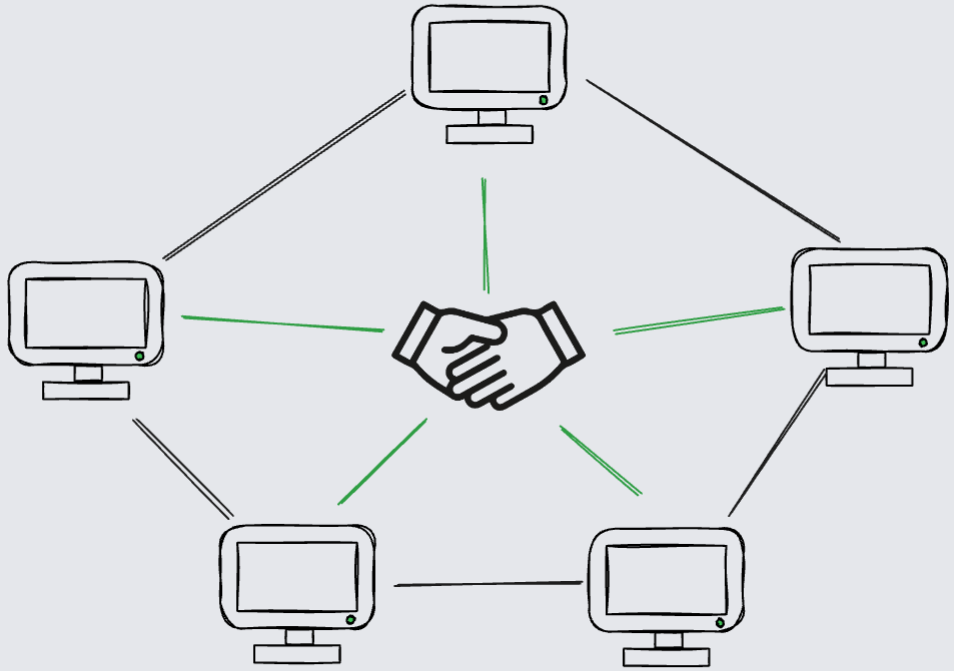
Prepared by Kirill Sizov

Agenda

- Problem definition
- CFT: Paxos review
- Byzantine generals problem
- pBFT review
- Consensus in Blockchain
- Proof-of-Work (PoW)
- Proof-of-Stake (PoS)

Core idea

- There is a a distributed computing system.
- There are a number of faulty processes.
- The goal is to provide a common agreement on the state of the system.



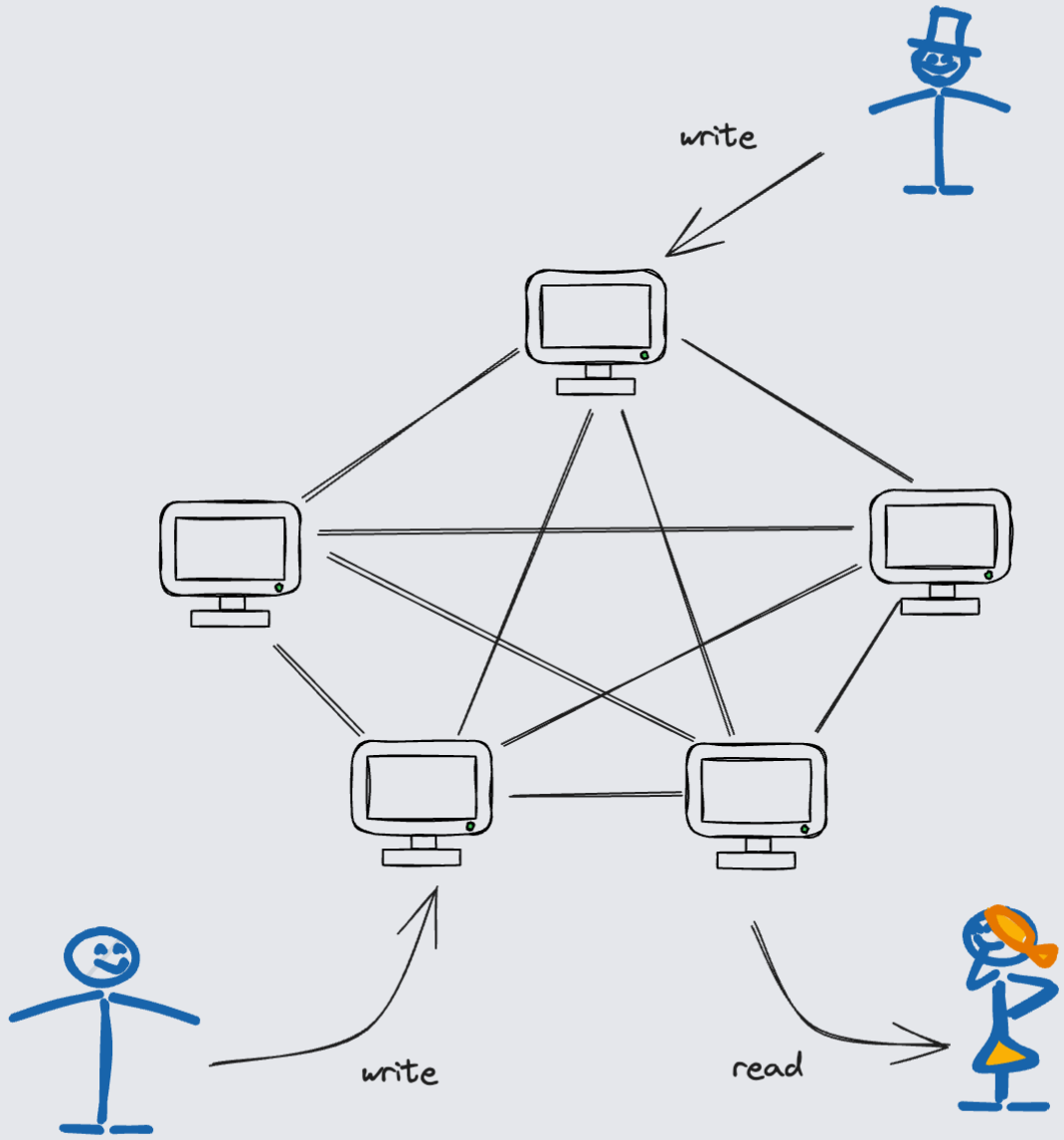
Consensus tasks

There are two types of consensus tasks:

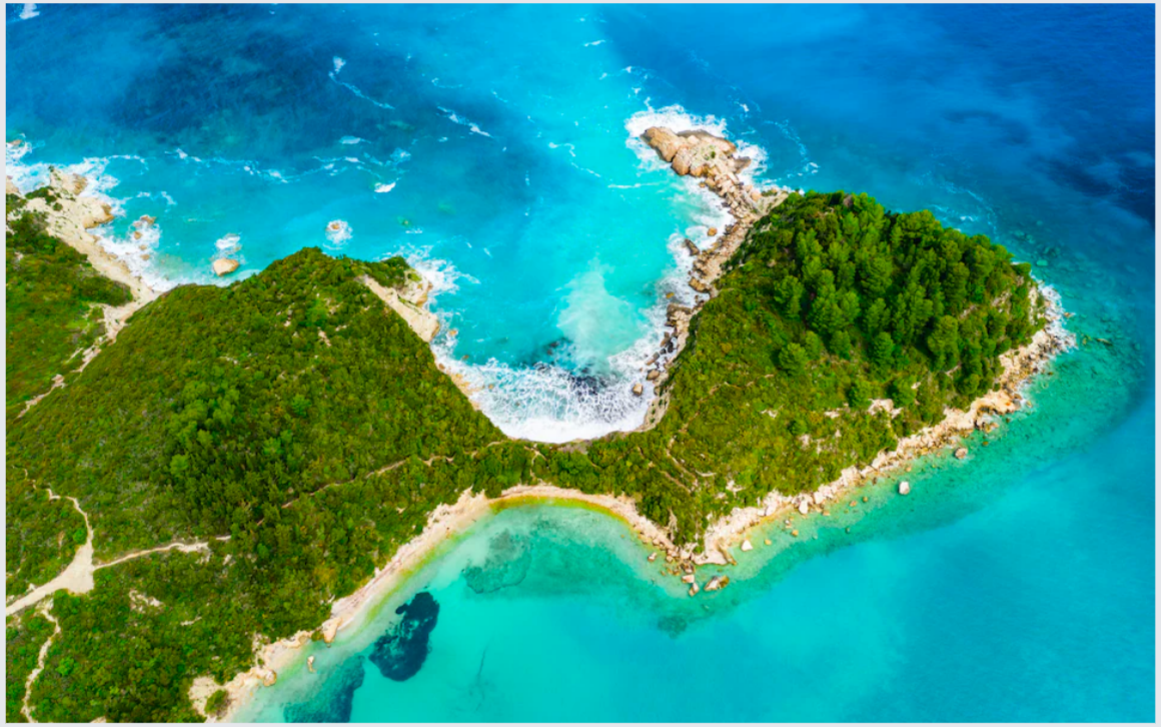
- Crash Fault Tolerance (CFT) – protection against failures of some components.
- Byzantine Fault Tolerance (BFT) – protection against intentional malicious nodes.

CFT. DB replication

- Clients write and read to/from different nodes.
- The nodes eventually have to reach the same state.



Paxos



> “I decided to cast the algorithm in terms of a parliament on an ancient Greek island…I gave the Greek legislators the names of computer scientists working in the field, transliterated with Guibas’s help into a bogus Greek dialect…Writing about a lost civilization allowed me to eliminate uninteresting details and indicate generalizations by saying that some details of the parliamentary protocol had been lost. To carry the image further, I gave a few lectures in the persona of an Indiana-Jones-style archaeologist, replete with Stetson hat and hip flask.” — Leslie Lamport (the author)

Roles

Within the Paxon Parliament

- **Proposer**: legislator, advocates a citizen's requests.
- **Acceptor**: legislator, voter.
- **Learner**: remembers and carries out result for citizen.

Quorum – any majority of acceptors.

Correspondence

Parliament

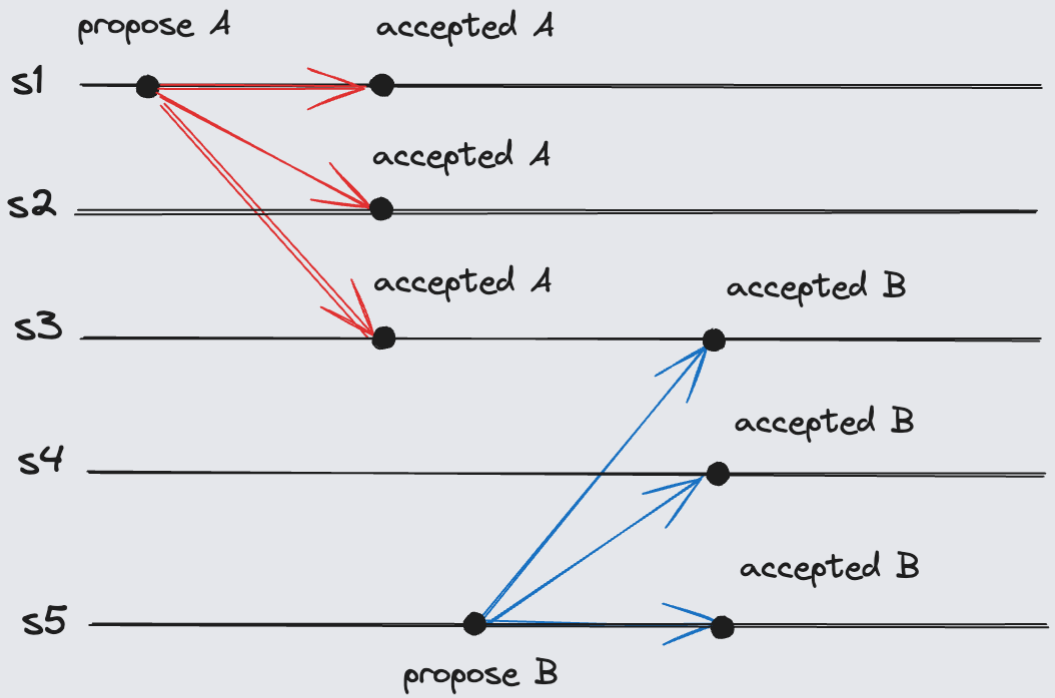
- Legislator
- Citizen
- Current law

Distributed Database

- Server
- Client program
- Database state

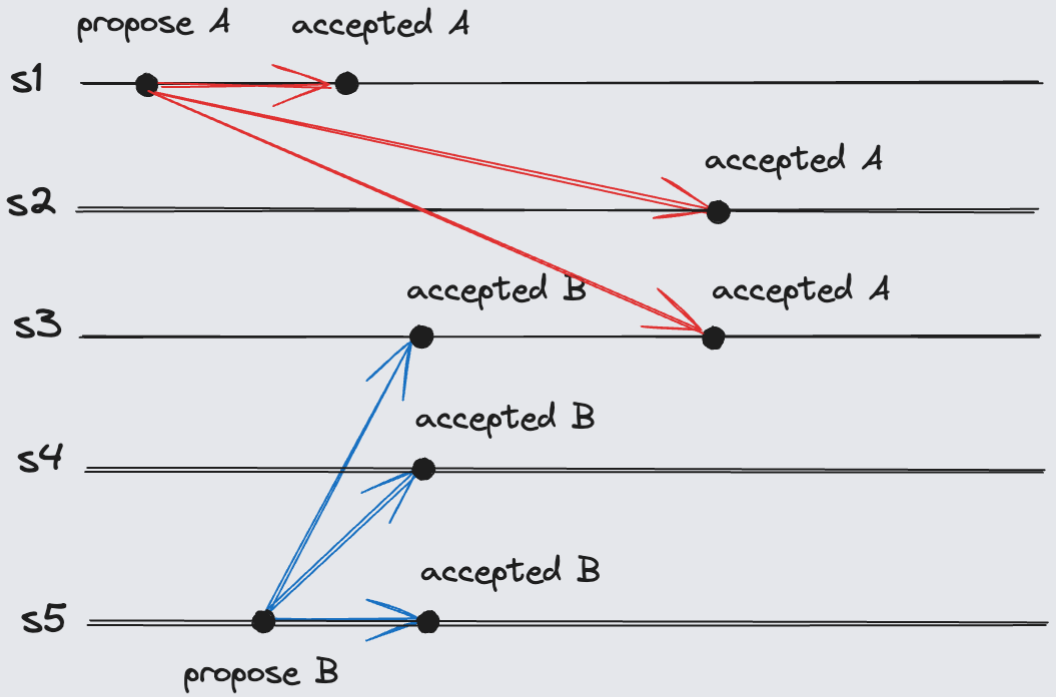
Conflicting choices

- Once a value is chosen, future proposals must agree on it.
- This is why protocol has 2 phase.



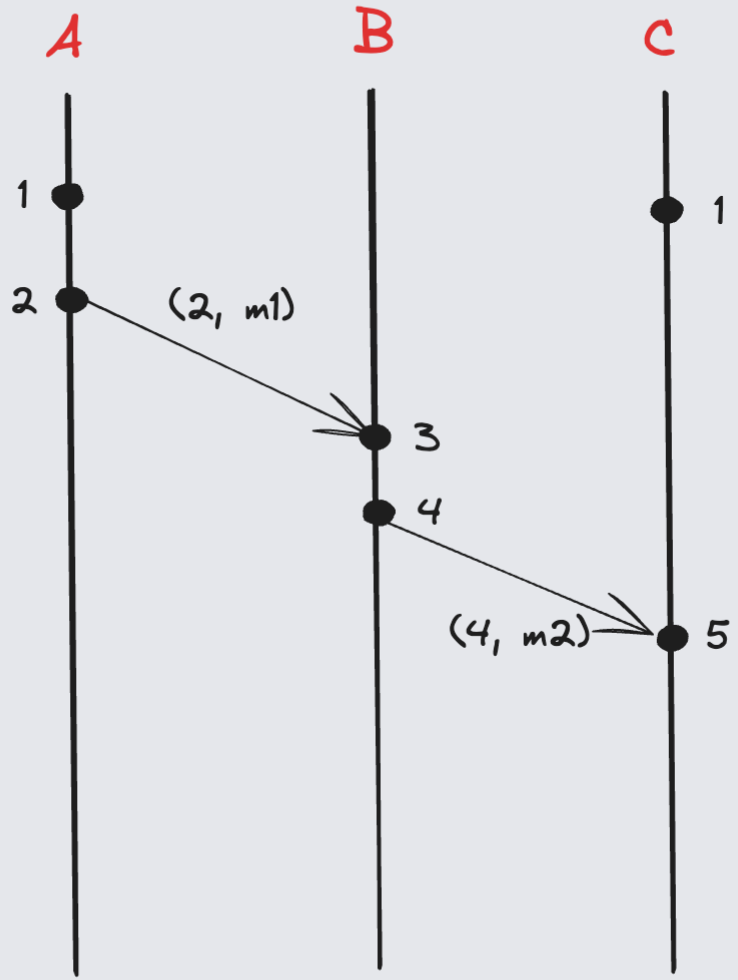
Conflicting choices

- There is a need for ordering proposals and reject older ones.
- Lamport timestamp used as a proposal number.



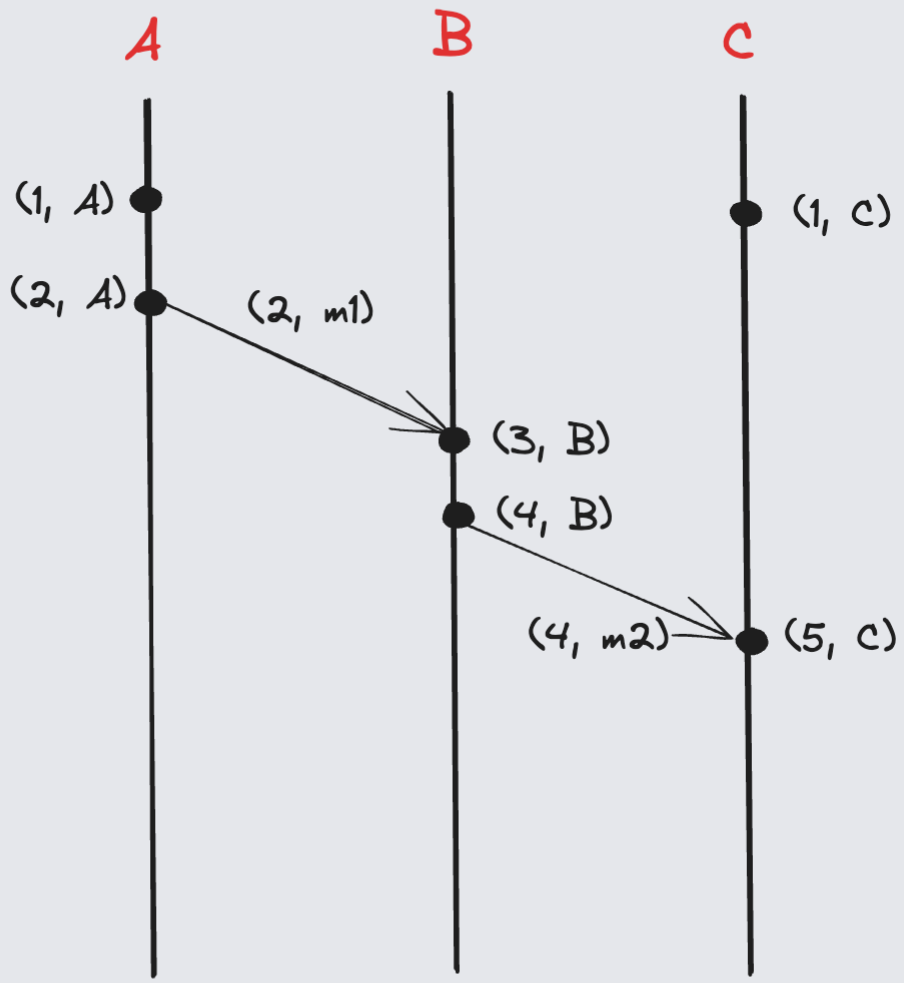
Lamport clocks

- Each node maintain a counter $L(e)$ incremented on every local event e.
- Attach $L(e)$ to messages sent over the network.
- Recipient moves its clock forward to timestamp in the message.



Lamport clocks

- Let $N(e)$ be the node at which event e occurred.
- $(L(e), N(e))$ uniquely identifies event e.



Basic Paxos

Phase 1

- **Proposer:**
  - Choose proposal n, send `<prepare, n>` to all acceptors
- **Acceptors:**
  - If $n > n_{highest}$
    - $n_{highest} = n$
    - reply `<promise, n, (n_acc, v_acc)>`
  - else reply `<prepare failed>`

Phase 2

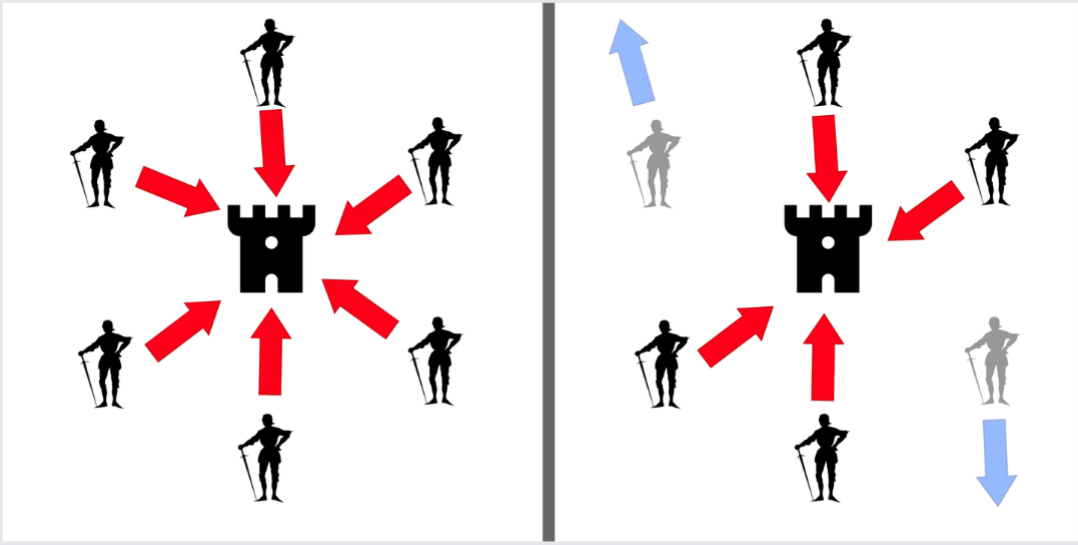
- **Proposer:**
  - if promise from majority
    - determine `v_acc` with highest `n_acc`
    - send `<accept, n, v_acc, v>` to all acceptor
- **Acceptors:**
  - If $n \geq n_{highest}$
    - $n_{accepted} = n_{highest} = n$
    - $v_{accepted} = v$

Paxos outro

- Has many variations and is widely used in practice.
- Each change is a separate round.
- Complex to understand.
- Requires $2f + 1$ replicas to survive f failures.
- Not applicable when there are malicious node or messages can be corrupted.

Byzantine generals problem

- Allegory used to describe Byzantine Fault Tolerance problem (Lamport, 1982).
- The challenge is for the loyal generals to reach a consensus on their strategy, despite the presence of unreliable parties.



If all generals attack in coordination, the battle is won (left).
If two generals falsely declare that they intend to attack,
but instead retreat, the battle is lost (right).

Goal

- **Agreement**: No two loyal generals take different actions.
- **Validity**: If the commander is loyal, then all loyal generals must take the action suggested by the commander.
- **Termination**: All loyal generals must eventually take some action.

Solution

- No solution for $\geq \frac{1}{3}$ traitors.
- Solution for $< \frac{1}{3}$ traitors – **Practical Byzantine Fault Tolerance** (Castro, Liskov, 1999)

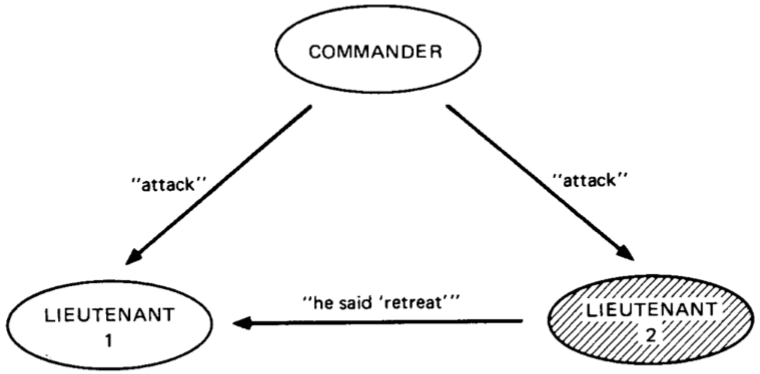


Fig. 1. Lieutenant 2 a traitor.

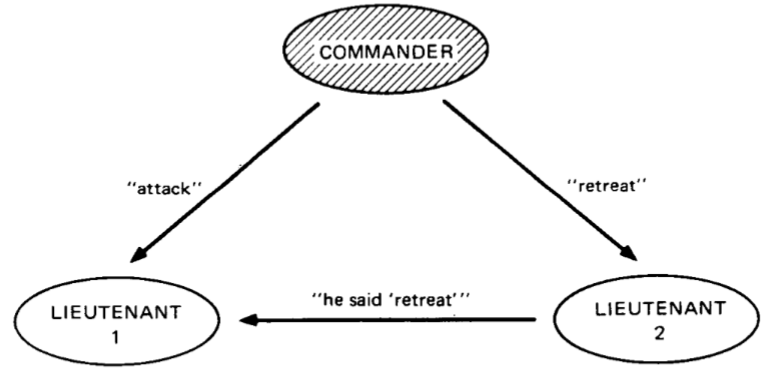


Fig. 2. The commander a traitor.

pBFT

Practical Byzantine Fault Tolerance (Castro, Liskov, 1999)

- $3f + 1$ replicas to survive f failures.
- 3 phases algorithm.
- Efficient.
- Tolerates Byzantine-faulty clients.

pBFT

- Assume:
  - operations are deterministic
  - replicas start in the same state
- If replicas execute the same requests in the same order
- Correct replicas will produce identical results

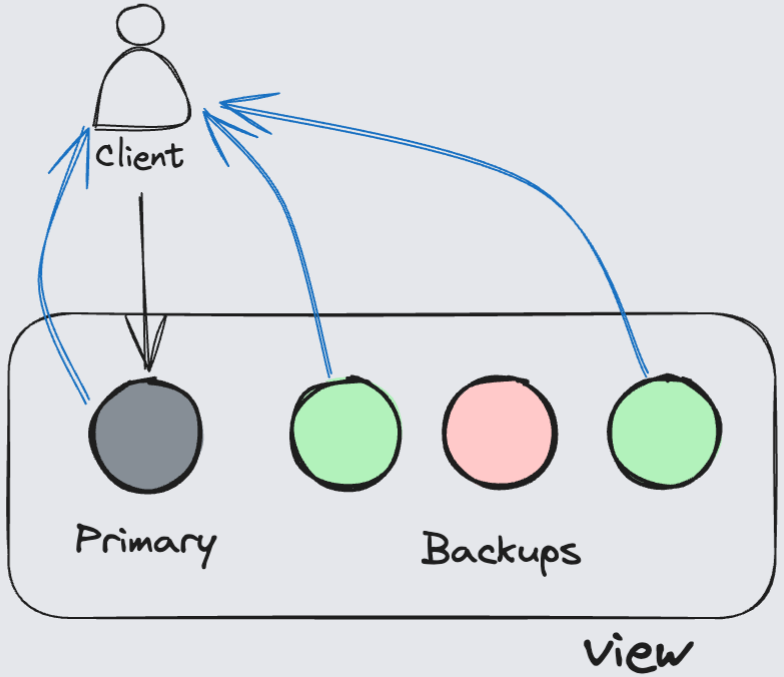
pBFT. What clients do

Clients pipeline:

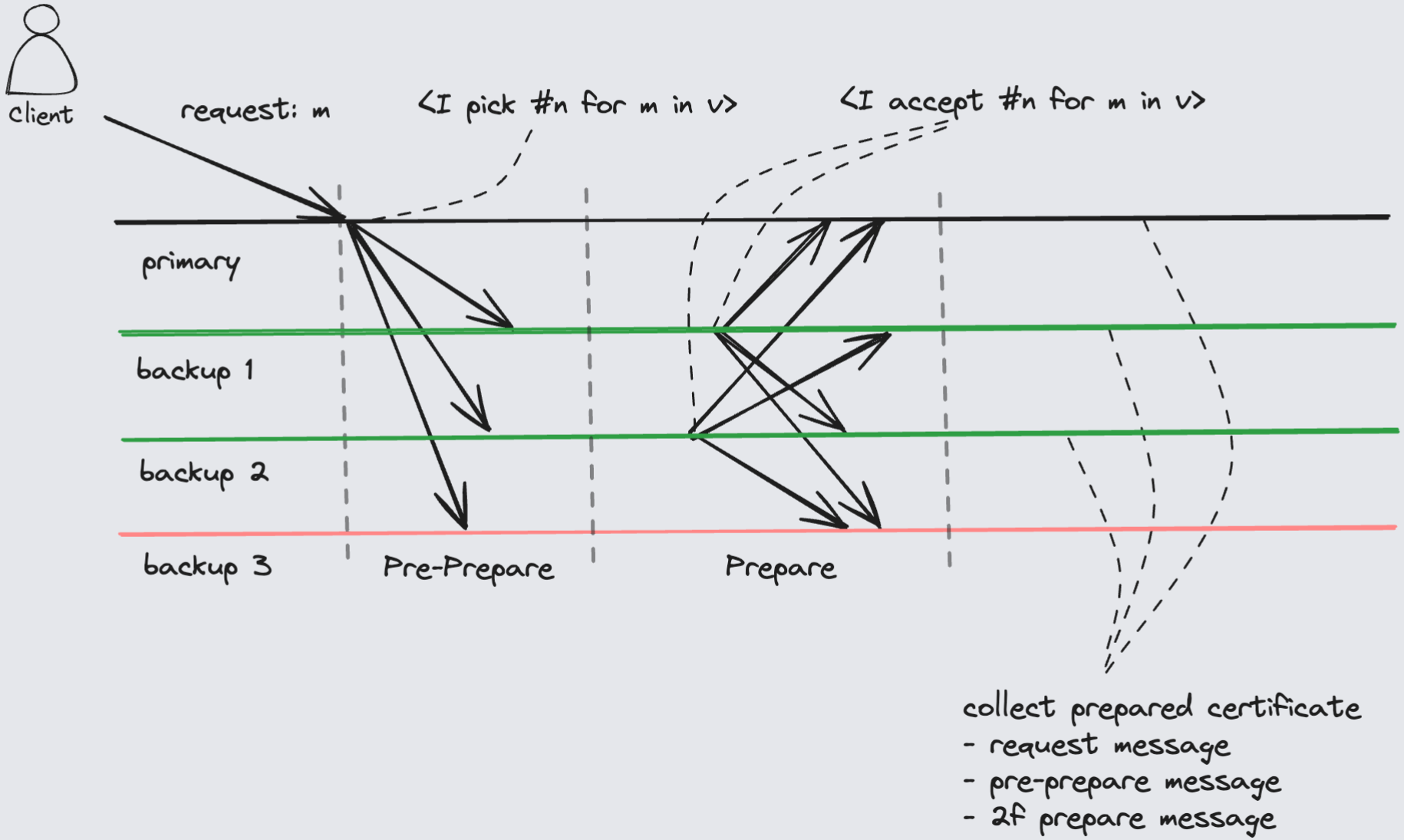
- send requests to all replicas
- wait for $f + 1$ identical results
- works because at least one reply is from a non-faulty replica

pBFT. View

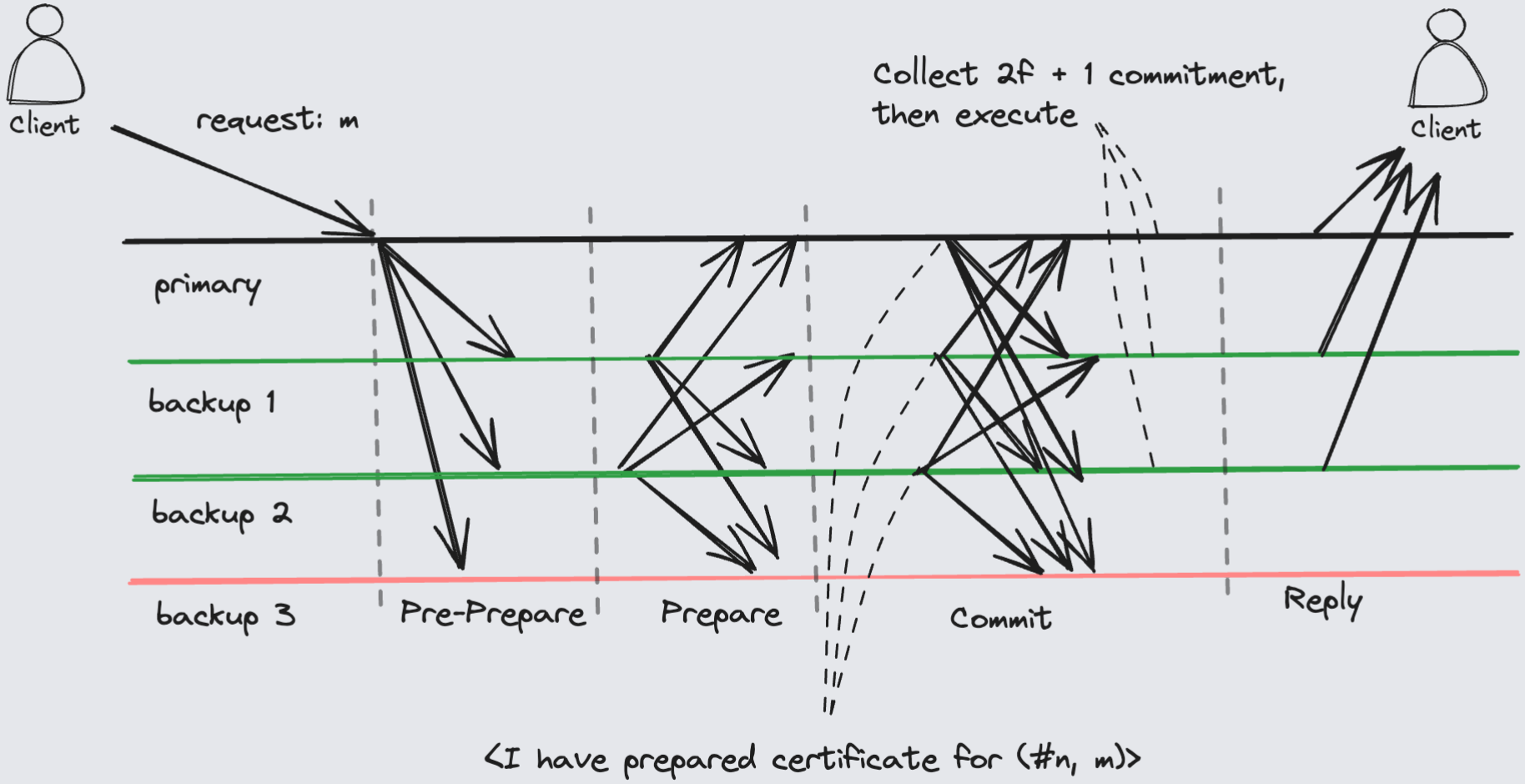
- **View** designates the primary replica.
- Primary picks the ordering.
- Backups ensure primary behaves correctly.
- Primary setting by formula: $p = v \bmod |\mathcal{R}|$



pBFT. Pre-Prepare & Prepare



pBFT. Commit & Reply



BFT in Blockchain

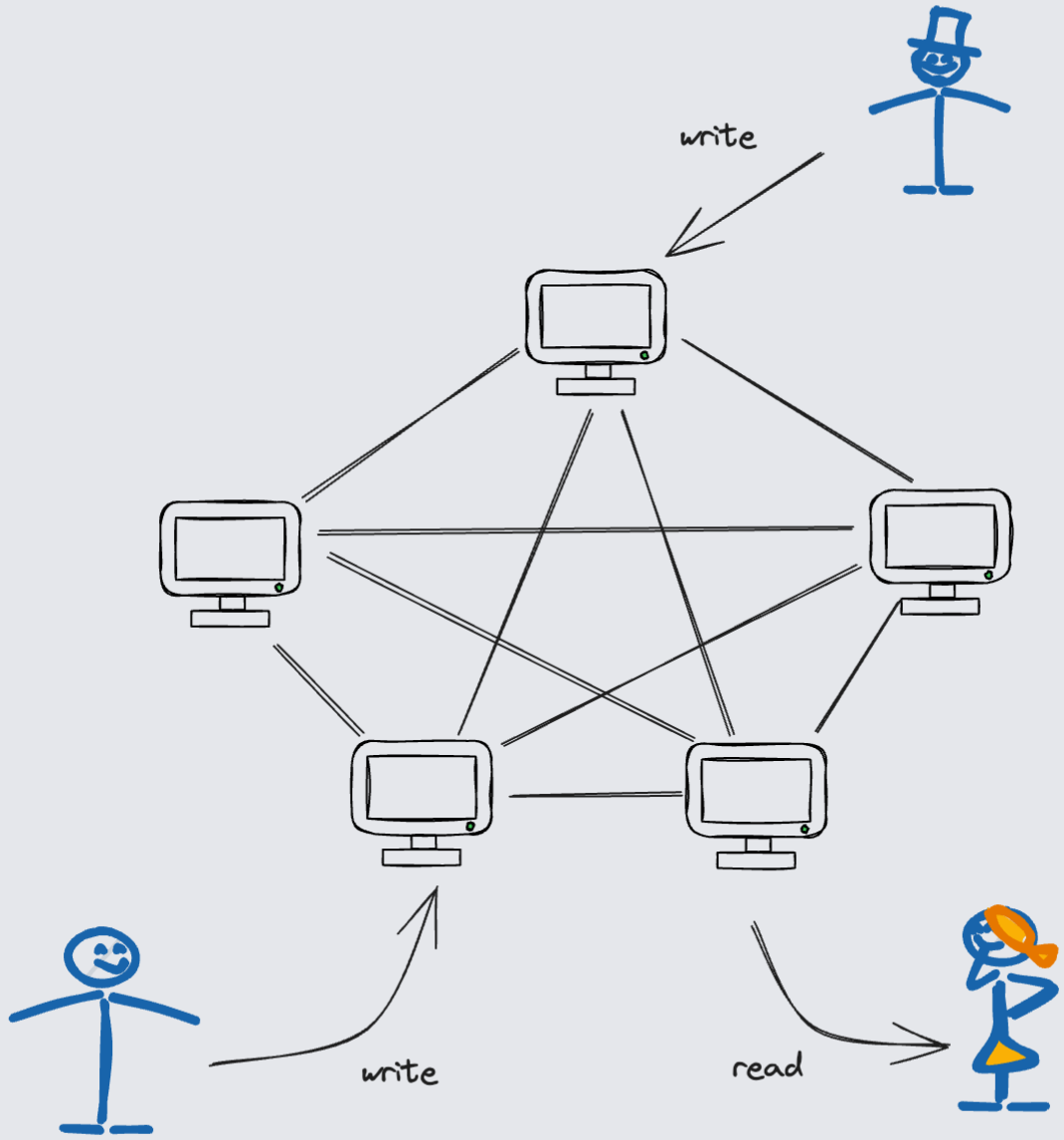
- Blockchain is decentralized.
- Anyone can become a participant.
- How to ensure security?

BFT in Blockchain

Byzantine Generals	Blockchain
Geographic distance	Distributed network
Generals	Nodes
Traitor generals	Faulty or malicious nodes
Unreliable messengers	Messages between nodes dropped or corrupted; unreliable network
Attack or retreat	Consensus on transactions

Analogy with DB replication

- Clients write by sending transaction and read by checking balances or other info.
- The nodes have to reach consensus by agreement rules (e.g., no double spending).

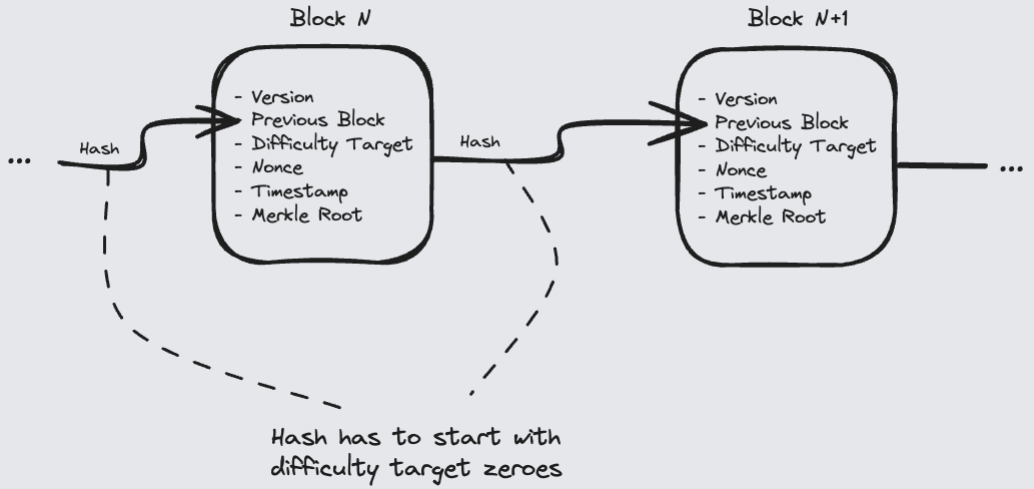


Proof of Work

- Core idea: The next "accepted" block will be the block from the node that did the most work.
- Block creator is rewarded with coins.

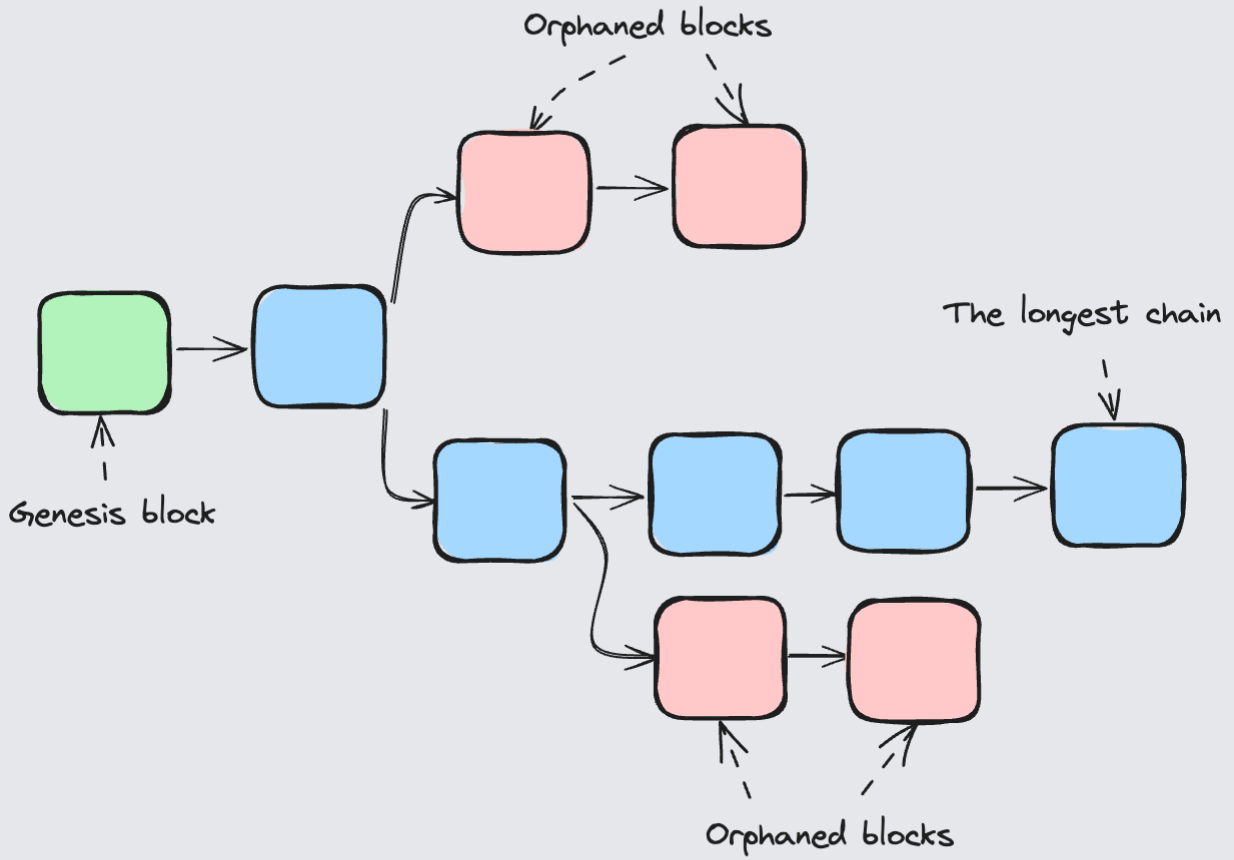
Bitcoin consensus

- Each node brute force the nonce of the block to find the smallest possible hash.
- The size of the minimum acceptable hash is determined by the difficulty target.
- Difficulty target is adjusted every 2016 blocks (~14 days).



Bitcoin consensus

- The longest chain is considered the valid one.
- When temporary forks occur, nodes follow the chain with the most accumulated work (typically the longest).



Proof of Work

Pros

- High level of security.
- More decentralized than PoS.

Cons

- Consumes a lot of resources, sometimes more than it gives itself.
- Slow transaction speed.
- No absolute block finality.

Proof of Stake

Proof of Stake

- Staking is locking cryptocurrencies to participate in network operations.
- Validator selection is determined by the amount staked.
- Validators are responsible for proposing and choosing main chain.

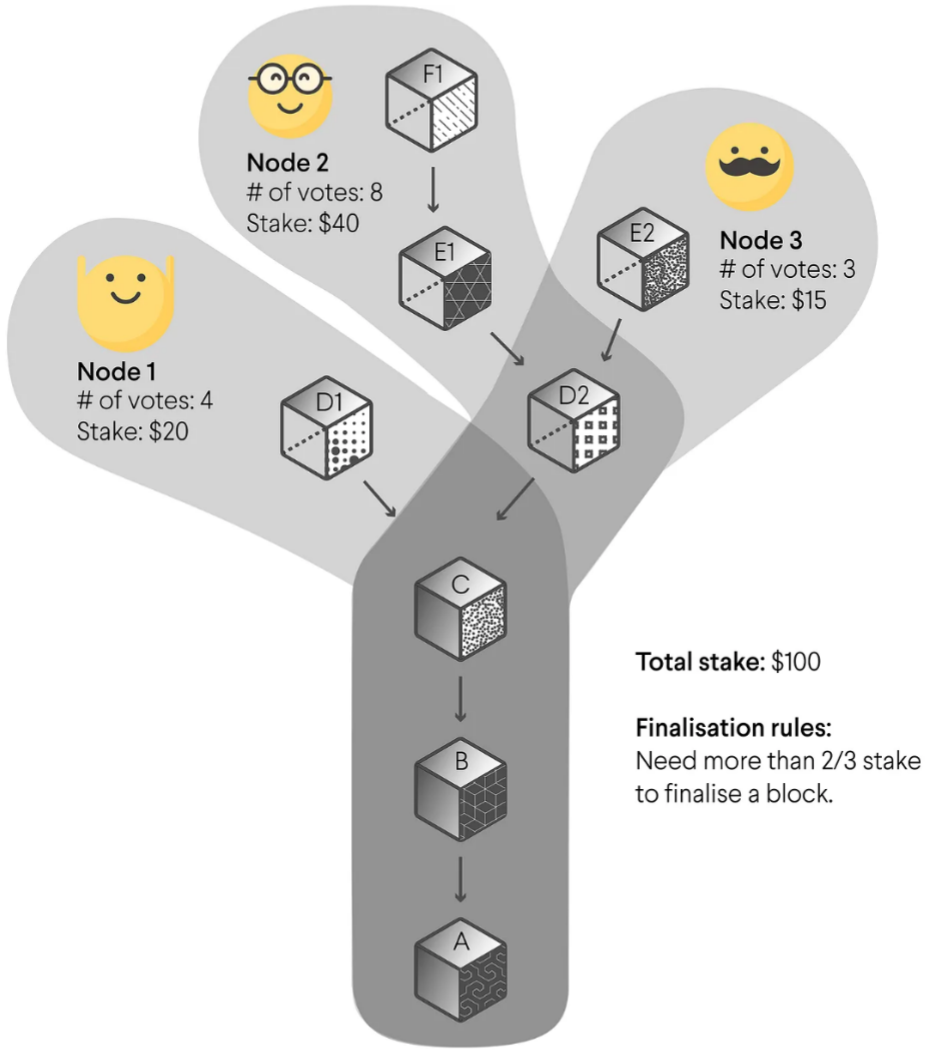
Proof of Stake

PoS algorithm is divided into two main parts:

- Block creation.
- Block finality.

Finality gadget

- The part of the consensus algorithm responsible for block fixation.
- A fixed block can't be rolled back.
- Examples:
  - GRANDPA
  - Casper
  - Tendermint
  - etc



Proof of Stake

Pros

- Relatively fast.
- Energy efficient.
- Block finality.

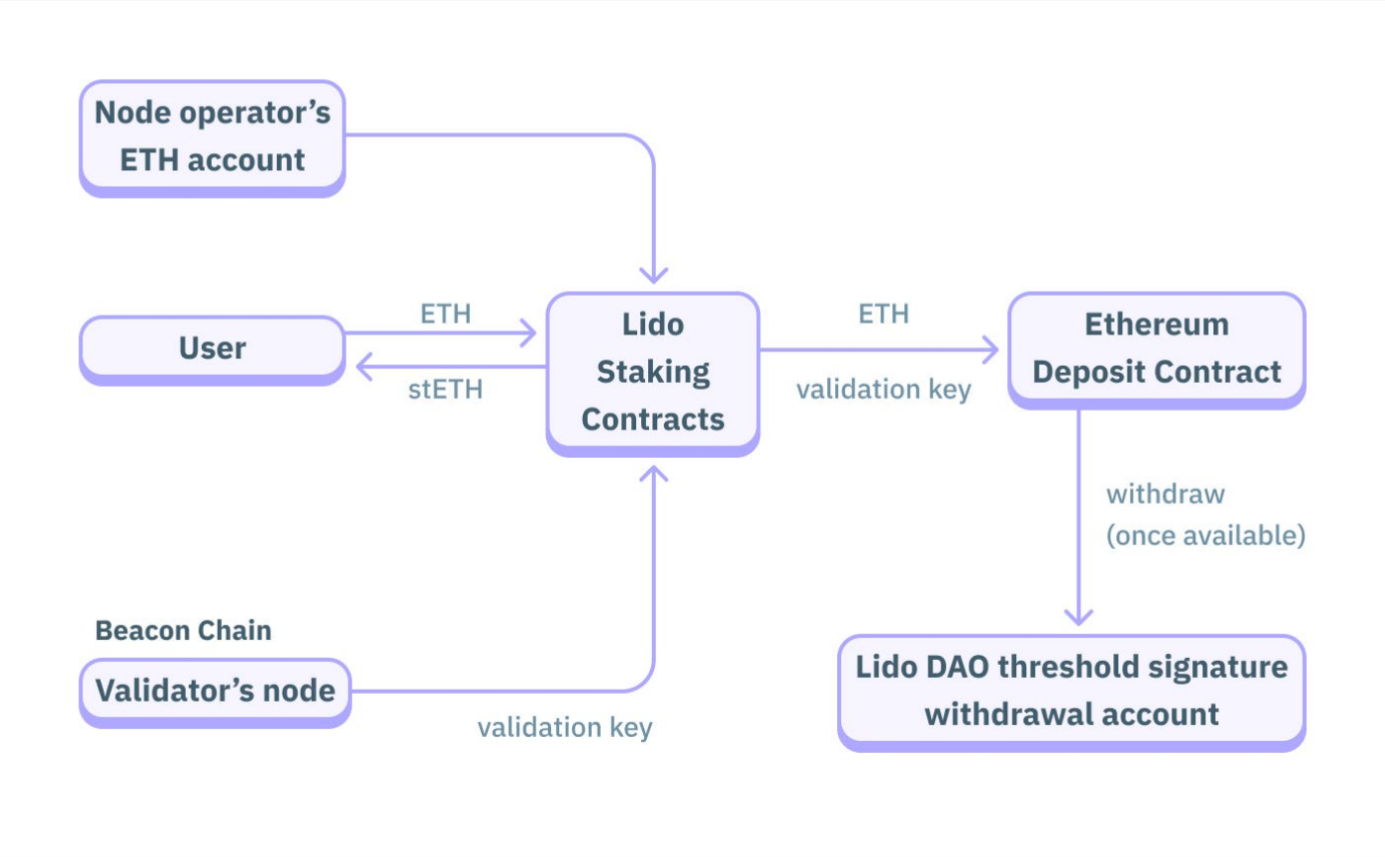
Cons

- Centralization risks.
- Less time tested.

Liquid staking

- Liquid staking is a process that allows users to stake their cryptocurrency without becoming validator
- Users receive a liquid derivative token in return, which can be used as liquid active.

Lido


Node operator's ETH account
User
ETH
stETH
Lido Staking Contracts
ETH
validation key
Ethereum Deposit Contract
withdraw (once available)
Beacon Chain
Validator's node
validation key
Lido DAO threshold signature withdrawal account