Consesus algorithms

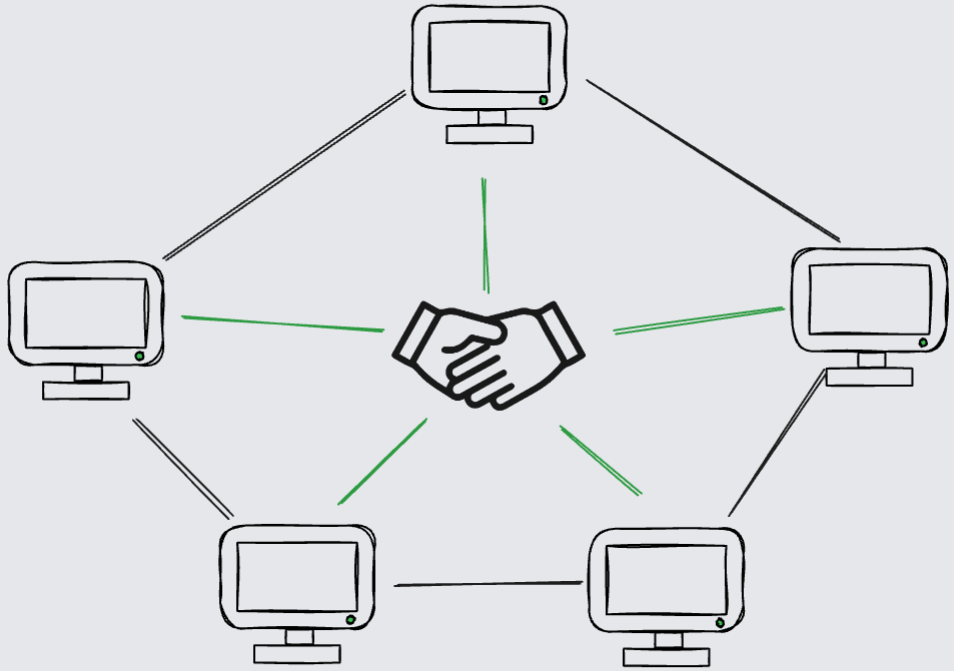
Prepared by Kirill Sizov

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.



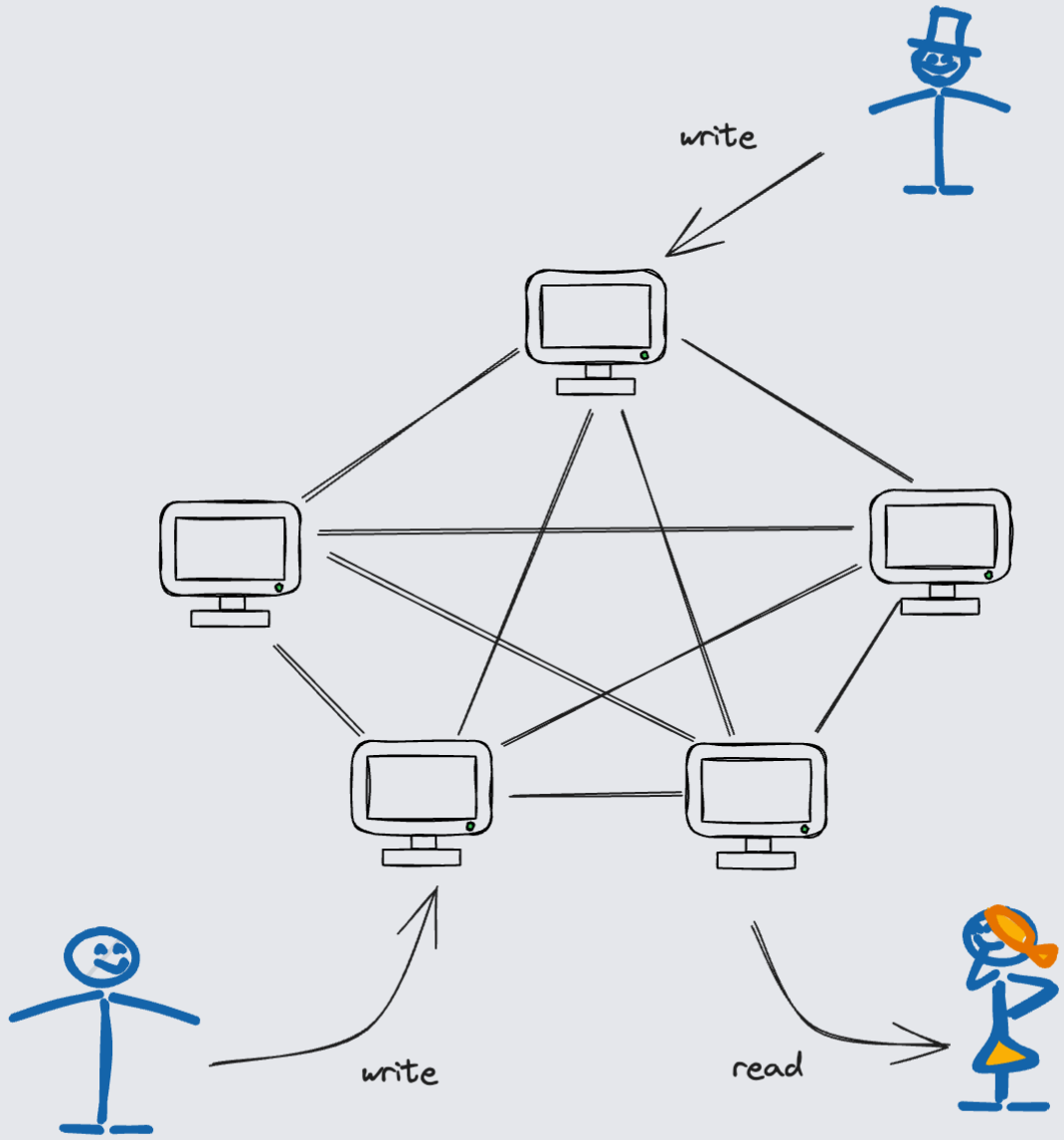
Consensus tasks

There are two types of consensus tasks:

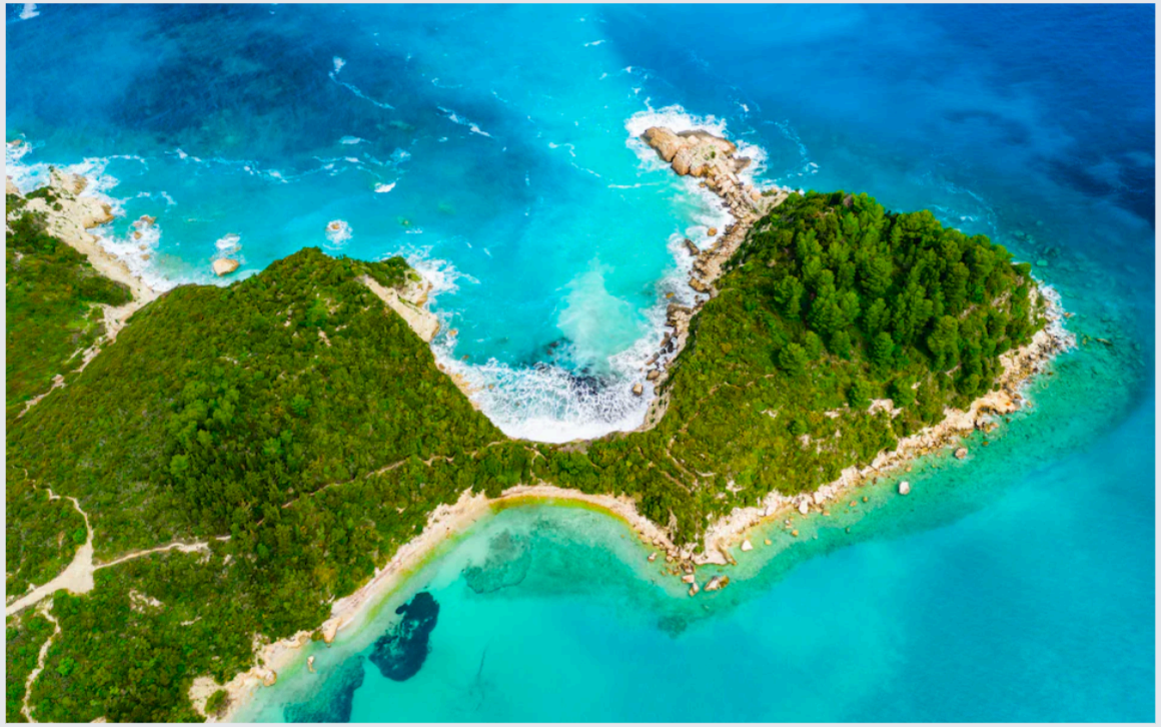
- Crash Fault Tolerance (CFT) – protection against failures of some components.
- Byzantine Fault Tolerance (BFT) – protection against intentional malicious nodes.

CFT. DB replication

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



Paxos



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

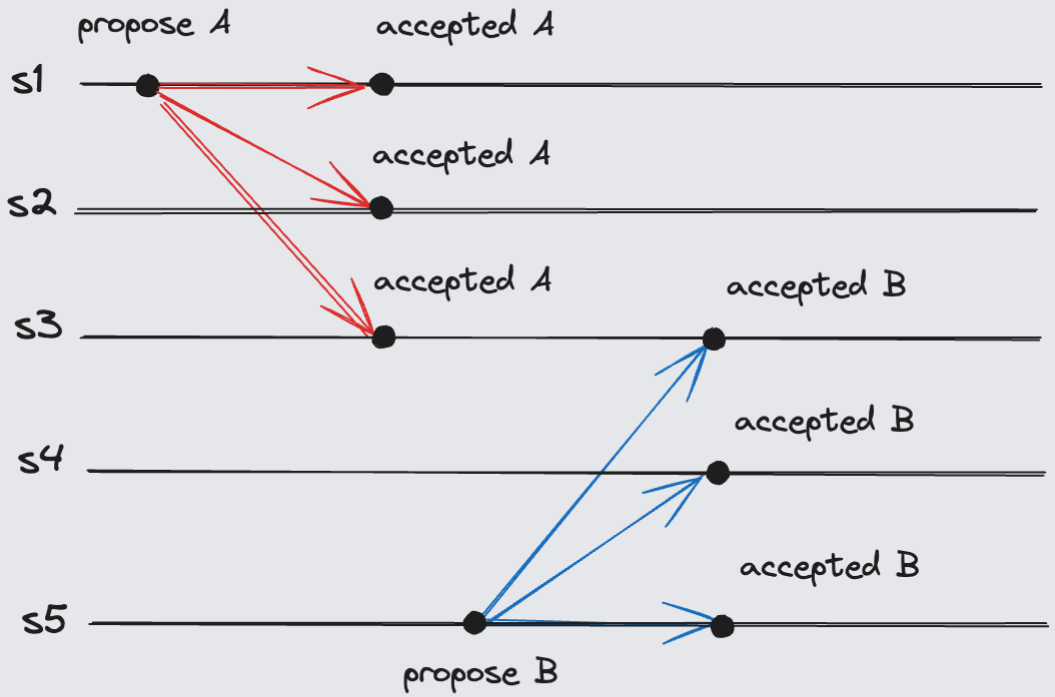
- Server
- Client program
- Database state

Requirements for Paxos

- **Safety**
  - Only a single value may be chosen.
  - A server never learns that a value has been chosen unless it really has been.
- **Liveness**
  - Some proposed value is eventually chosen.
  - If a value is chosen, servers eventually learn about it.

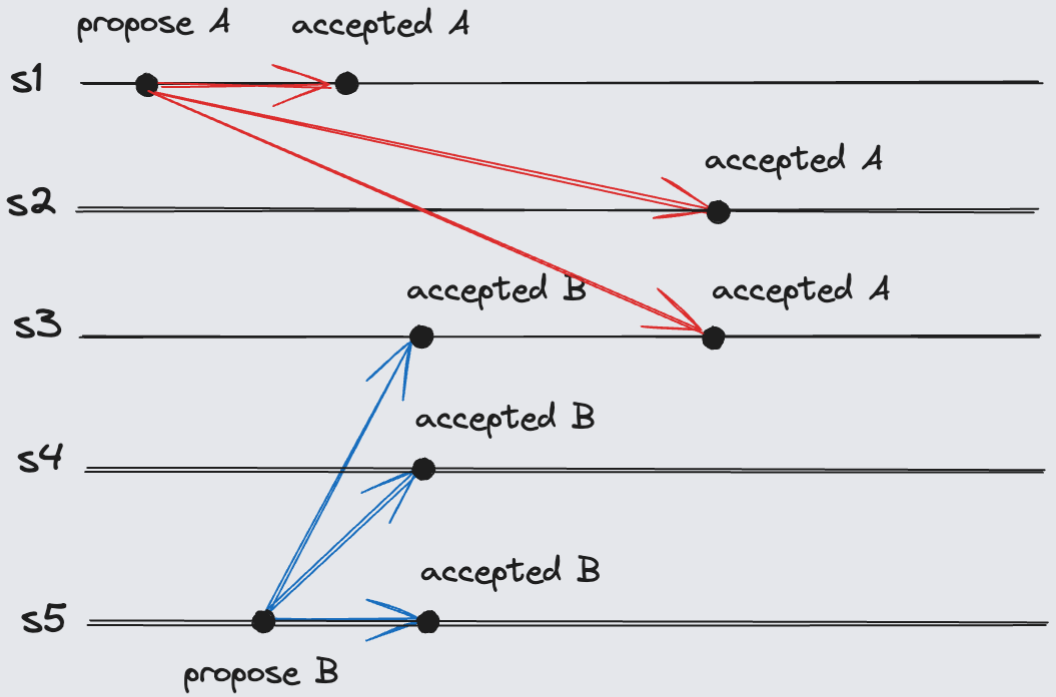
Conflicting choices

- What if acceptor accepts every value it receives?
  - Contradicts safety requirement.
- Once a value is chosen, future proposals must agree on it.
- This is why protocol has 2 phase.



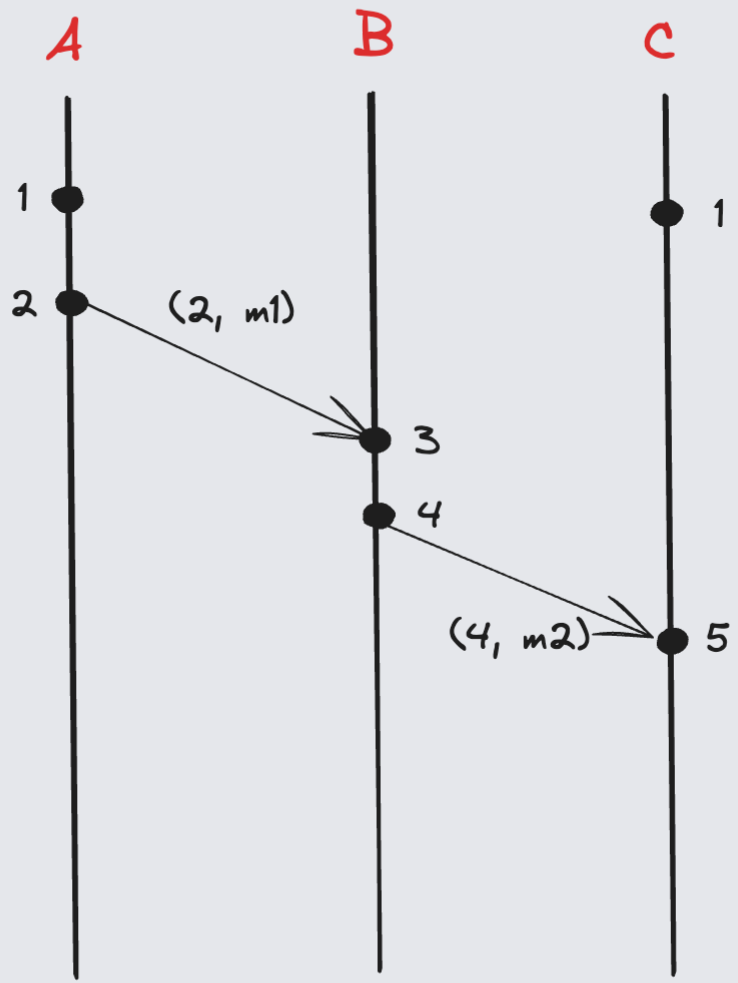
Conflicting choices

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



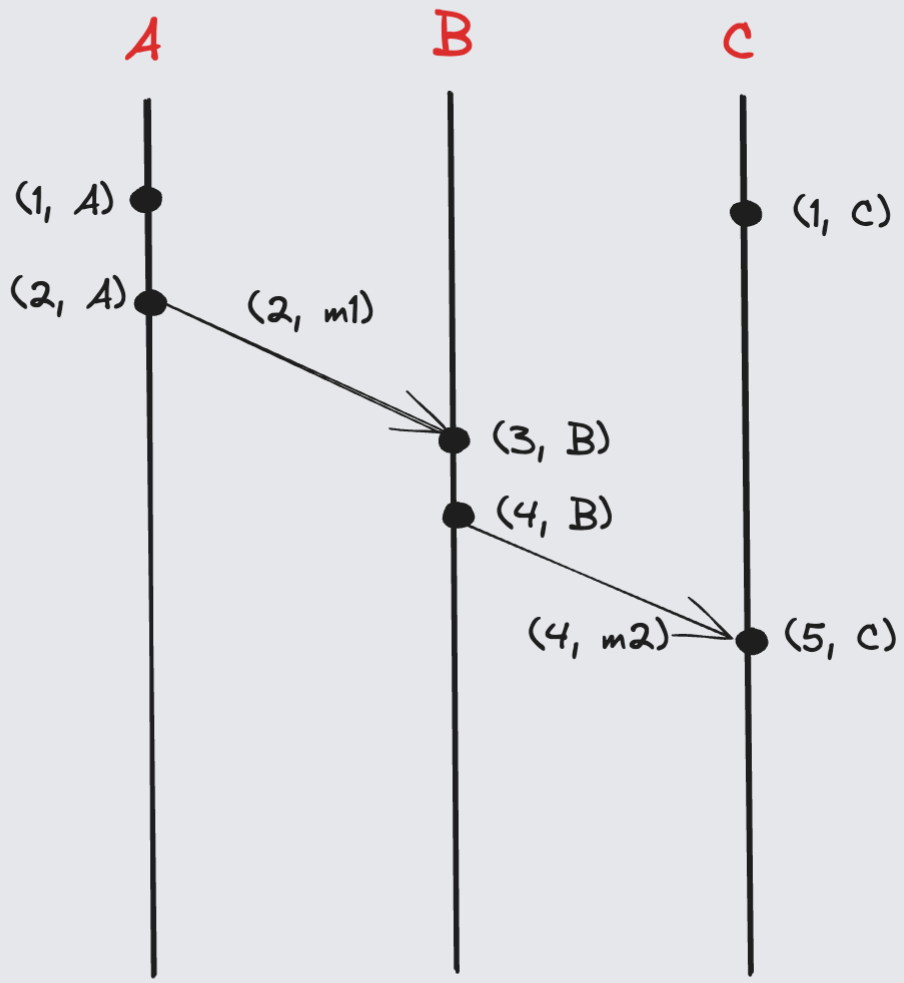
Lamport clocks

- 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 (Prepare)

- **Proposer:**
  - Send current lamport clock number `<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 (Accept)

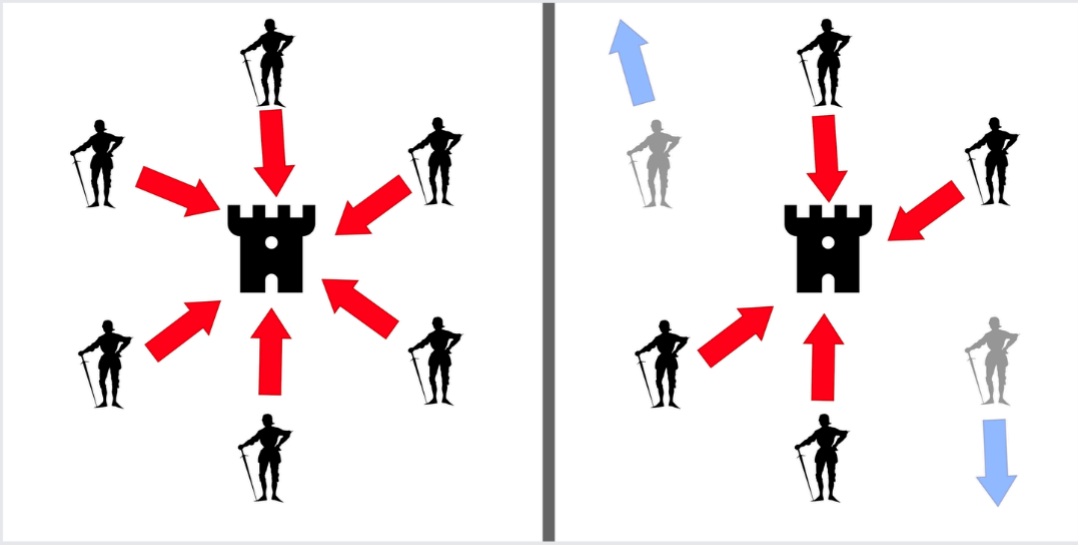
- **Proposer:**
  - if promise from majority
    - determine `v_acc` with highest `n_acc`
    - send `<accept, n, v_acc, v>` to all acceptor
- **Acceptors:**
  - If $n \geq n_{highest}$
    - $n_{accepted} = n_{highest} = n$
    - $v_{accepted} = v$

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.

BFT in Blockchain

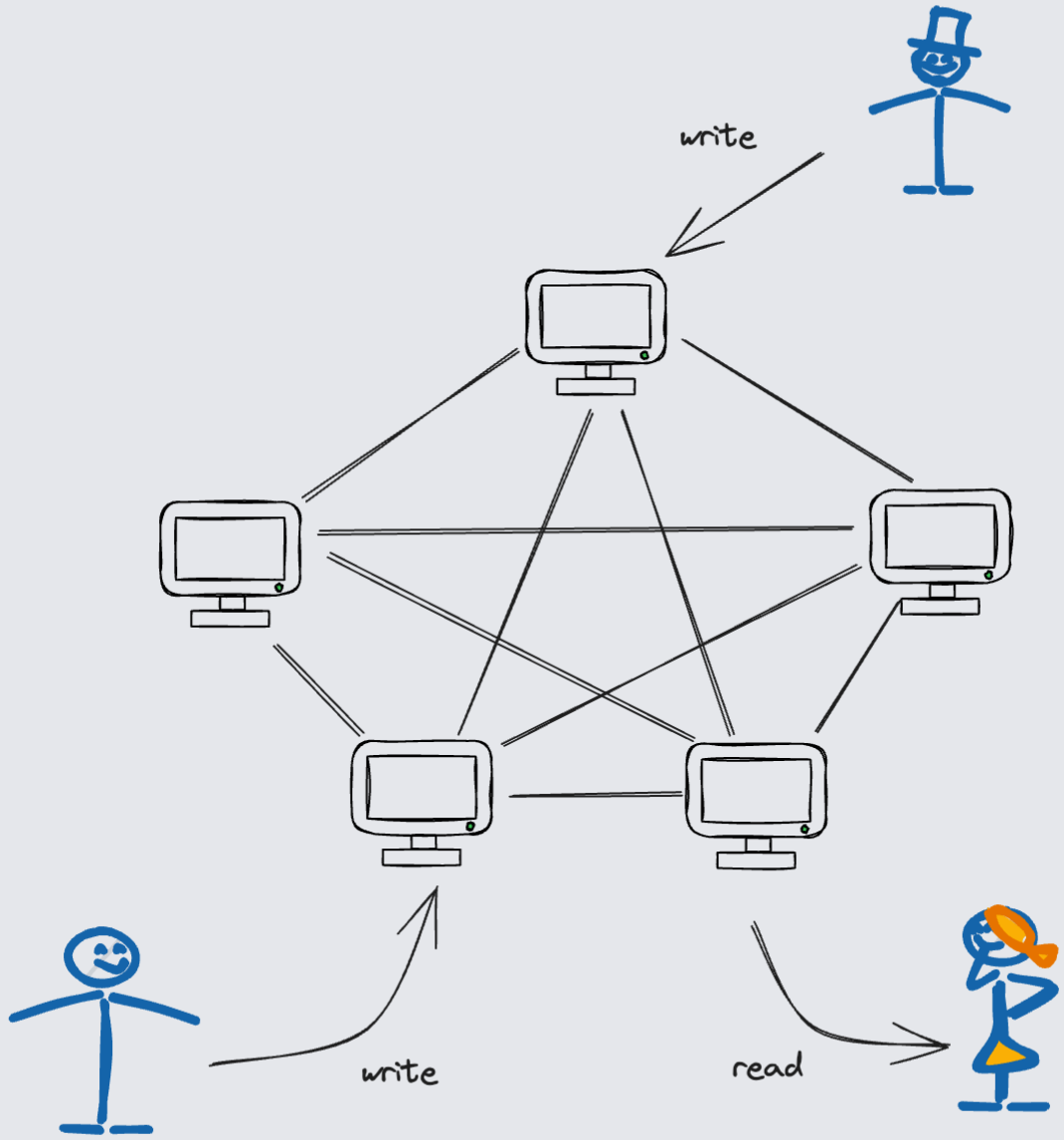
- Blockchain is decentralized.
- Anyone can become a participant.
- How to ensure security?

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).



Solution

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

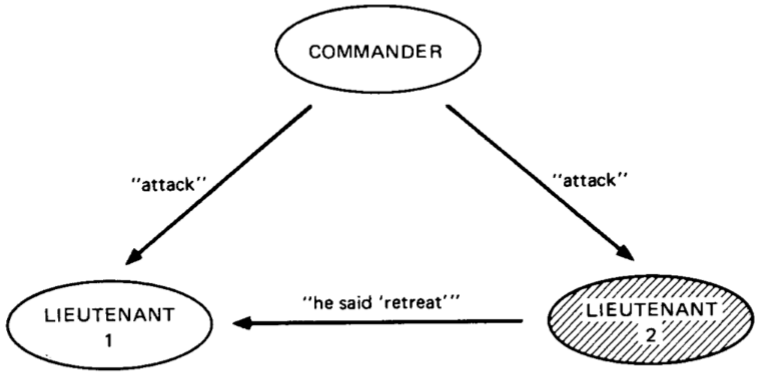


Fig. 1. Lieutenant 2 a traitor.

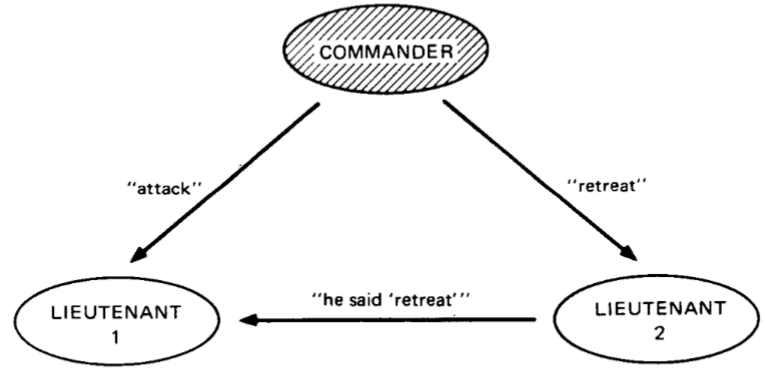


Fig. 2. The commander a traitor.

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

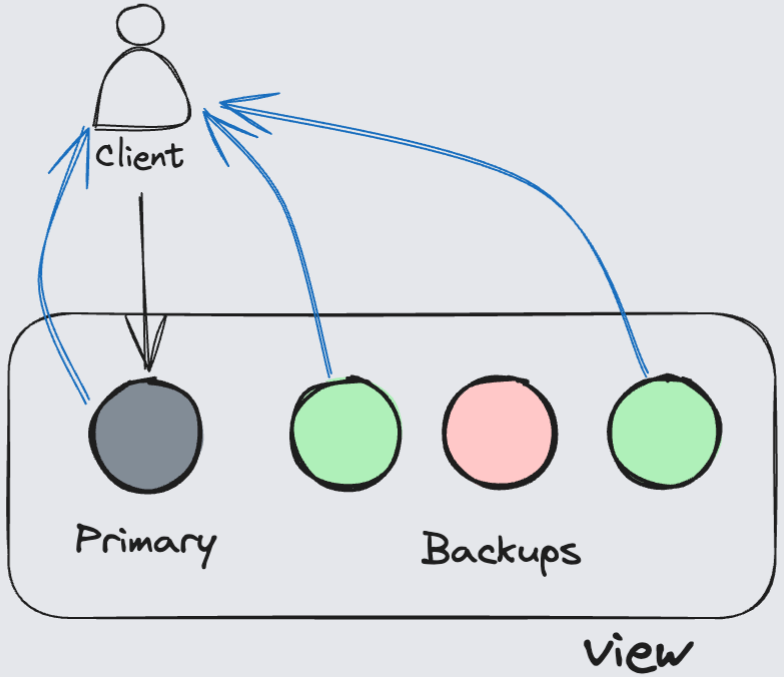
pBFT. What clients do

Clients pipeline:

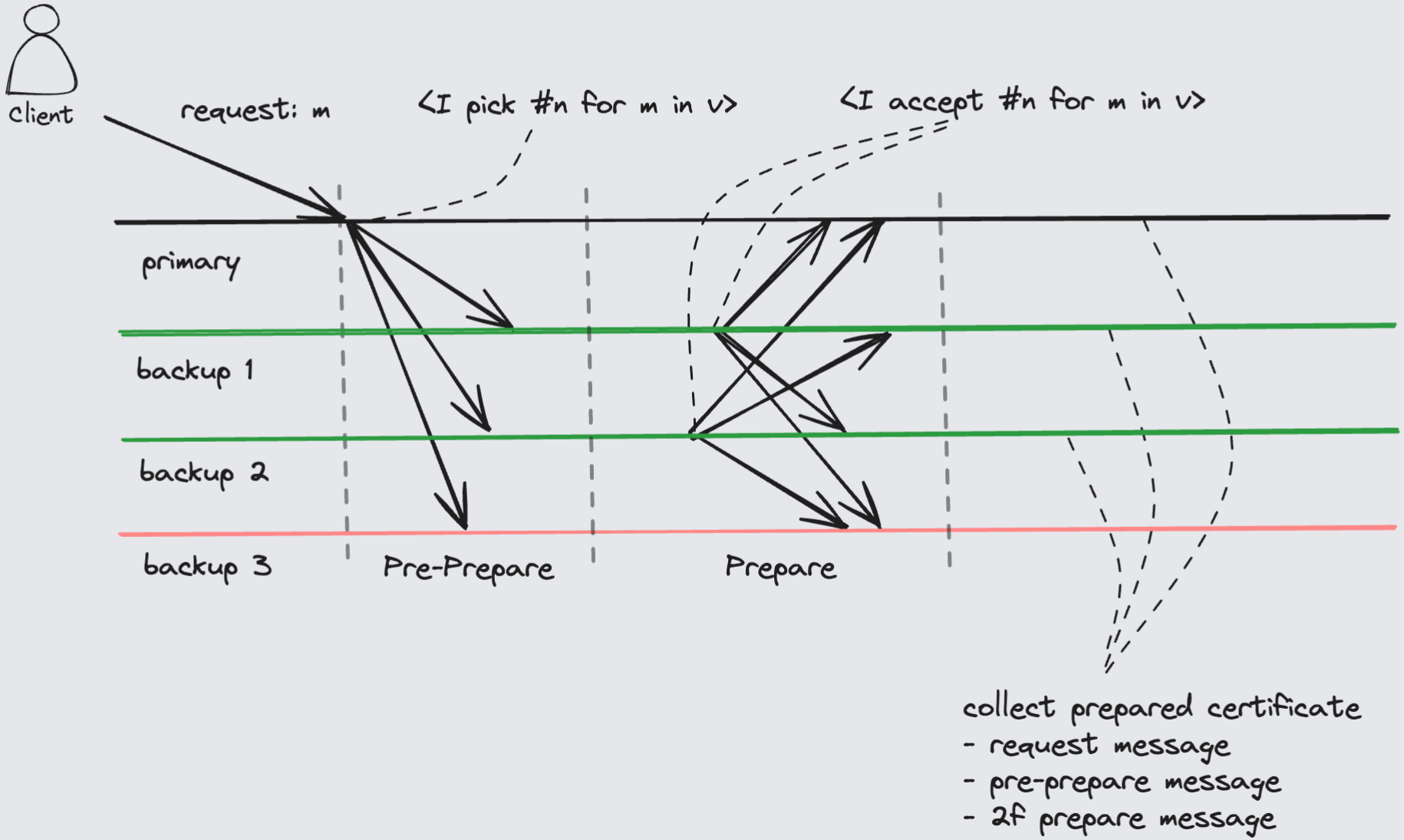
- send requests to all replicas
- wait for $f + 1$ identical results
- works because at least one reply is from a non-faulty replica

pBFT. View

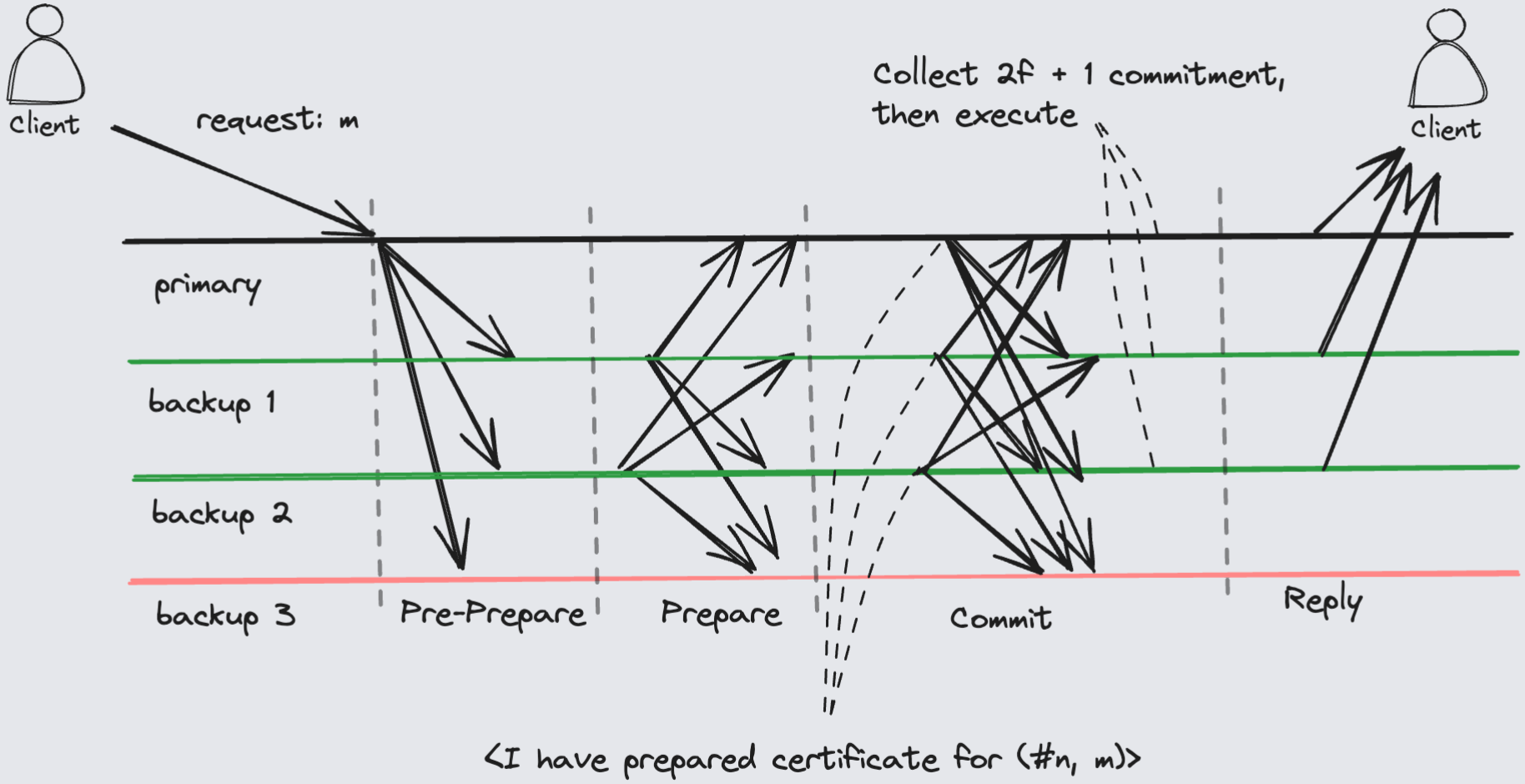
- **View** designates the primary replica.
- Primary picks the ordering.
- Backups ensure primary behaves correctly.
- Primary setting by formula: $p = v \bmod |\mathcal{R}|$



pBFT. Pre-Prepare & Prepare



pBFT. Commit & Reply



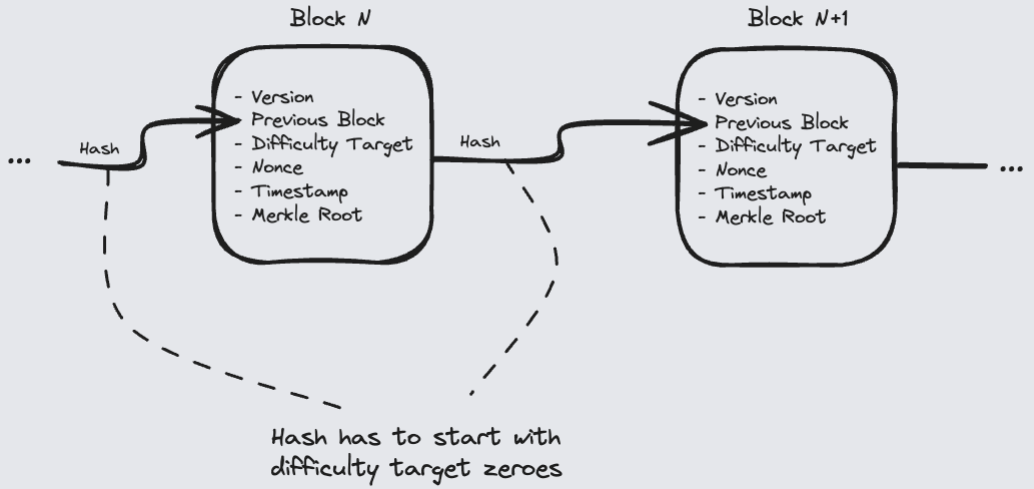
Consensus in blockchains

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.

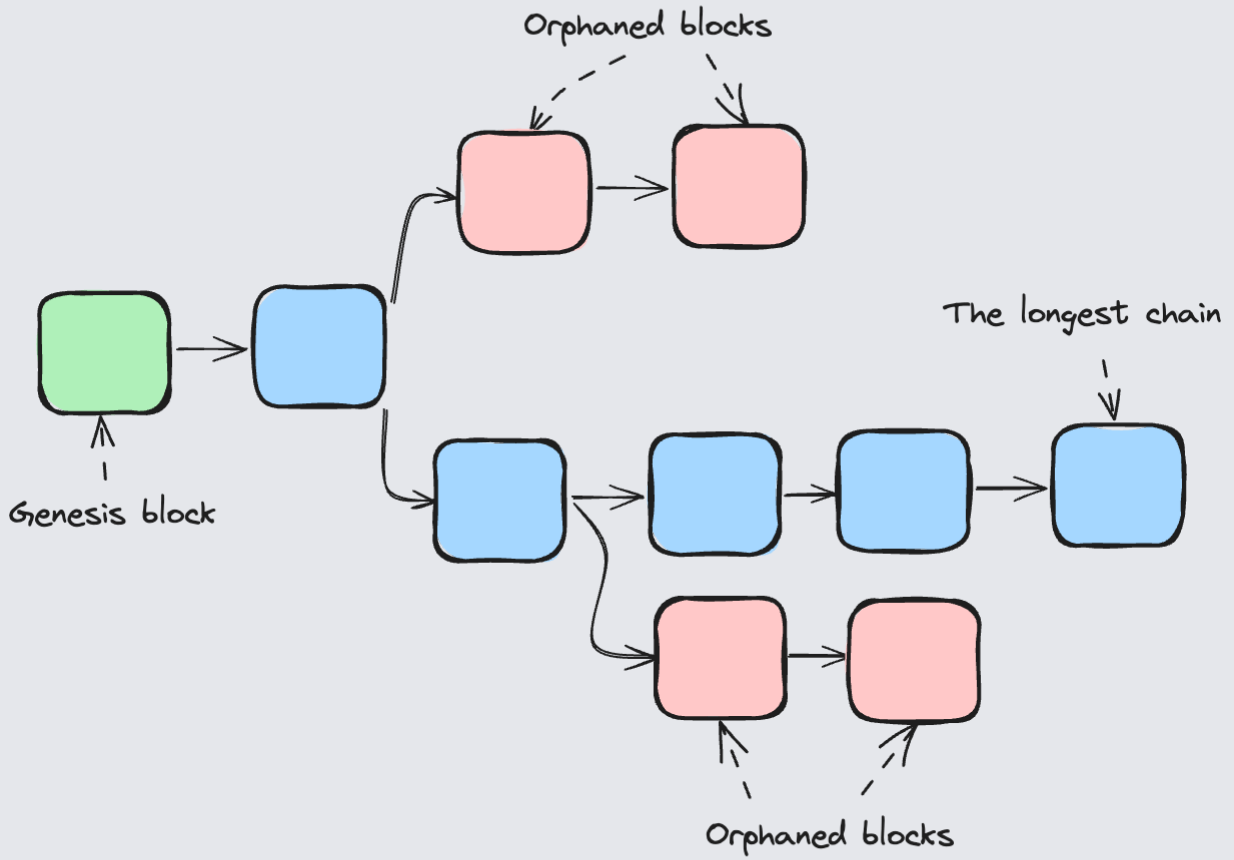
Bitcoin consensus

- 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

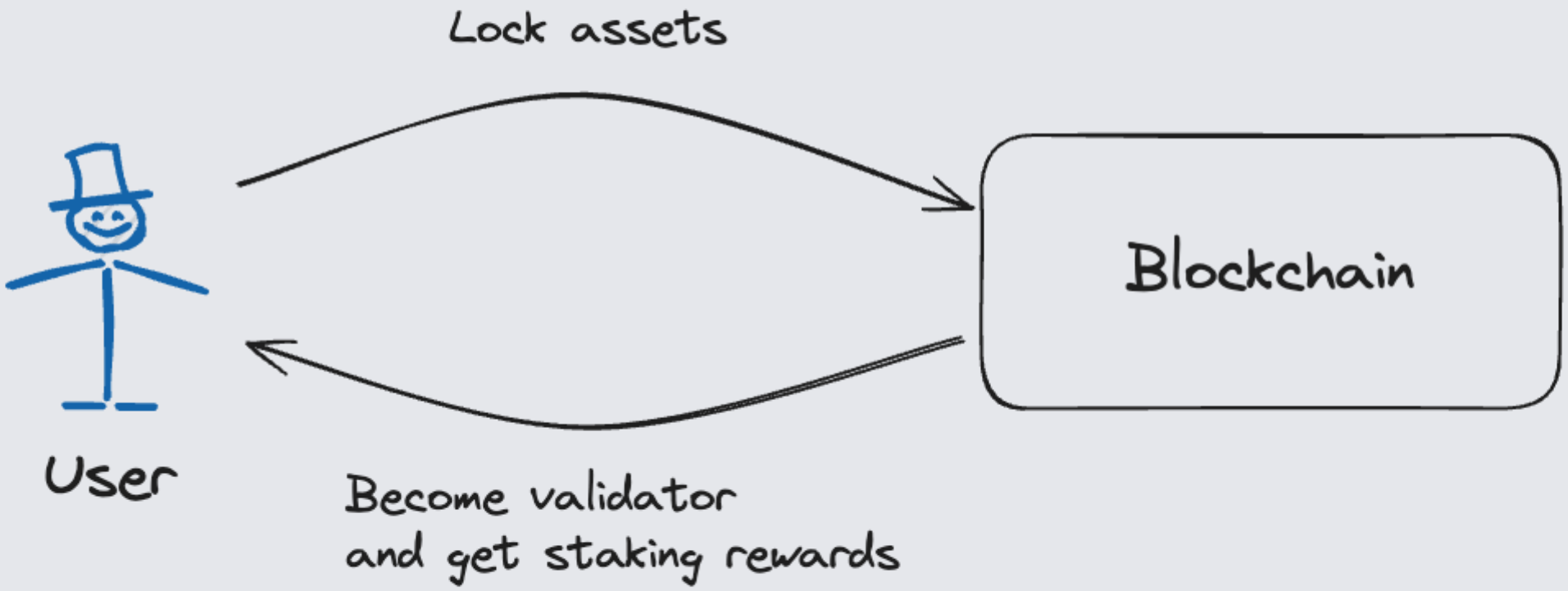
- High level of security.
- More decentralized than PoS.

Cons

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

Proof of Stake

Staking


Lock assets
Blockchain
User
Become validator
and get staking rewards

Proof of Stake

Pros

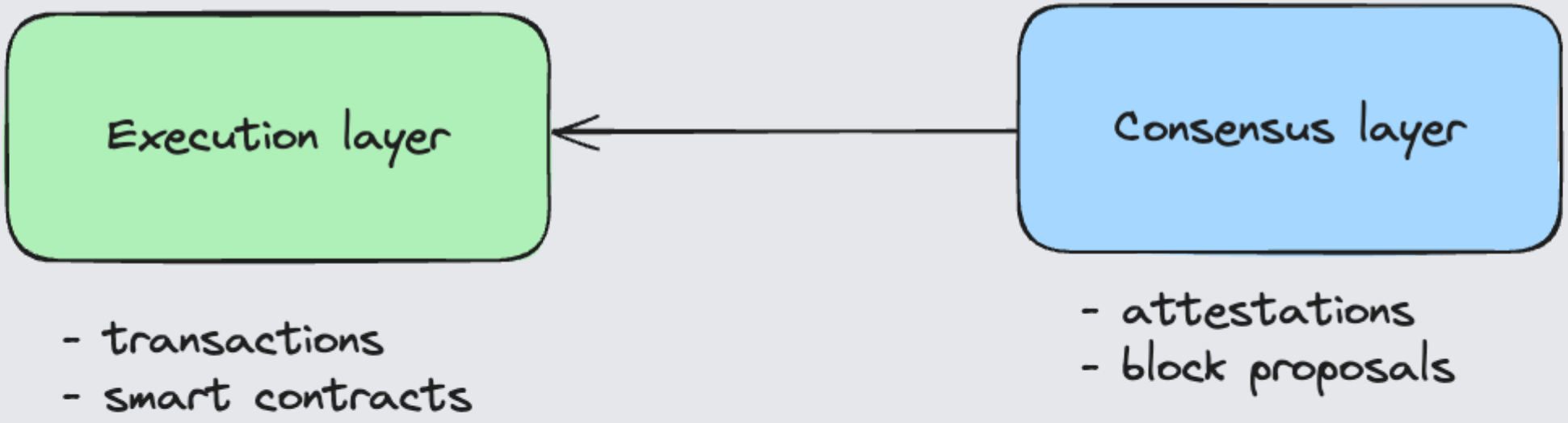
- Relatively fast.
- Energy efficient.
- Block finality.

Cons

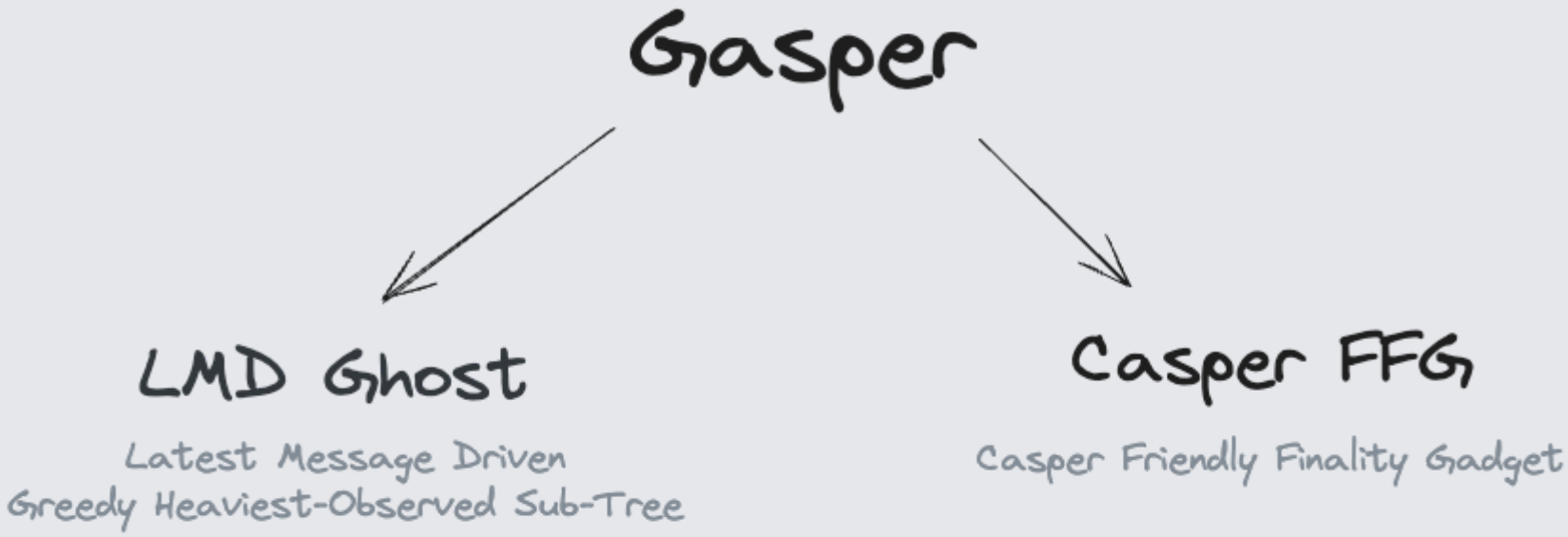
- Centralization risks.
- Less time tested.

Ethereum dive in

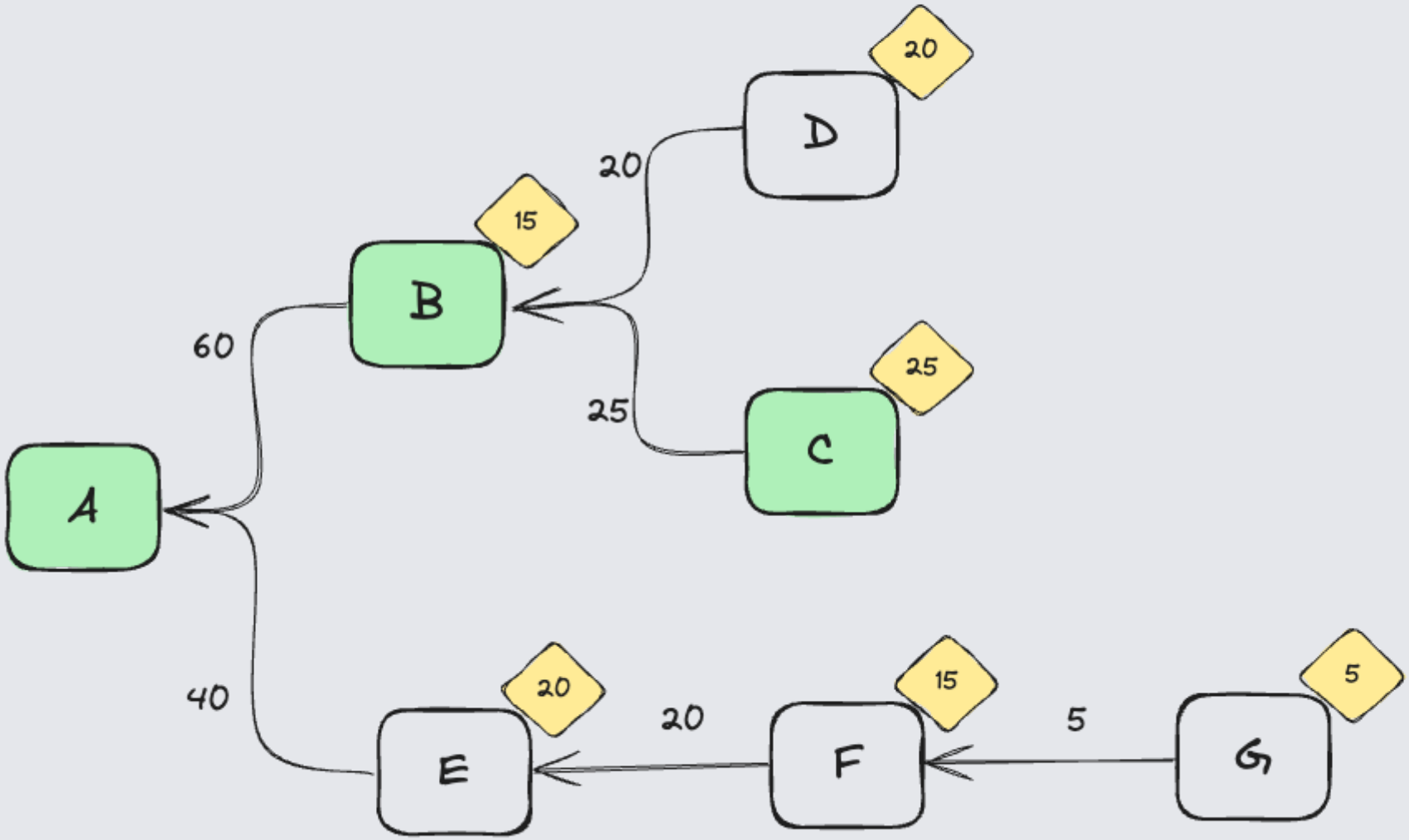
Ethereum


Execution layer
Consensus layer
- transactions
- smart contracts
- attestations
- block proposals

Gasper


Gasper
LMD Ghost
Latest Message Driven
Greedy Heaviest-Observed Sub-Tree
Casper FFG
Casper Friendly Finality Gadget

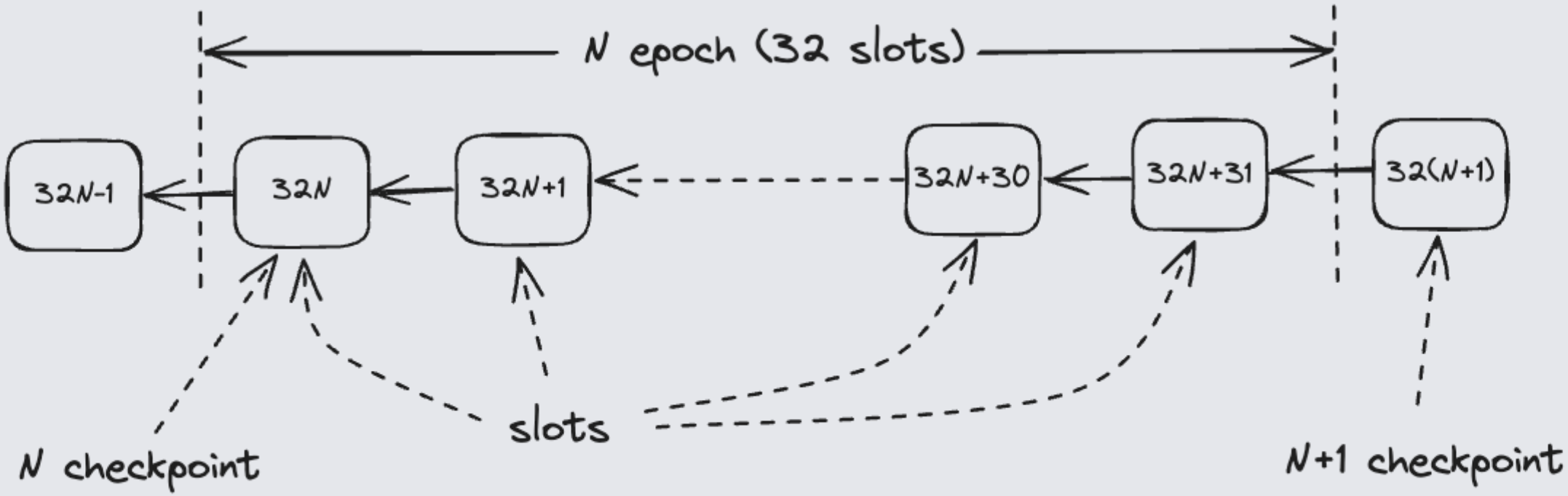
LMD Ghost



Casper FFG: finality

- The part of the consensus algorithm responsible for block fixation.
- A fixed block can't be rolled back.
- Finality is needed to prevent major reorgs.
- In Ethereum, finality is not guaranteed.

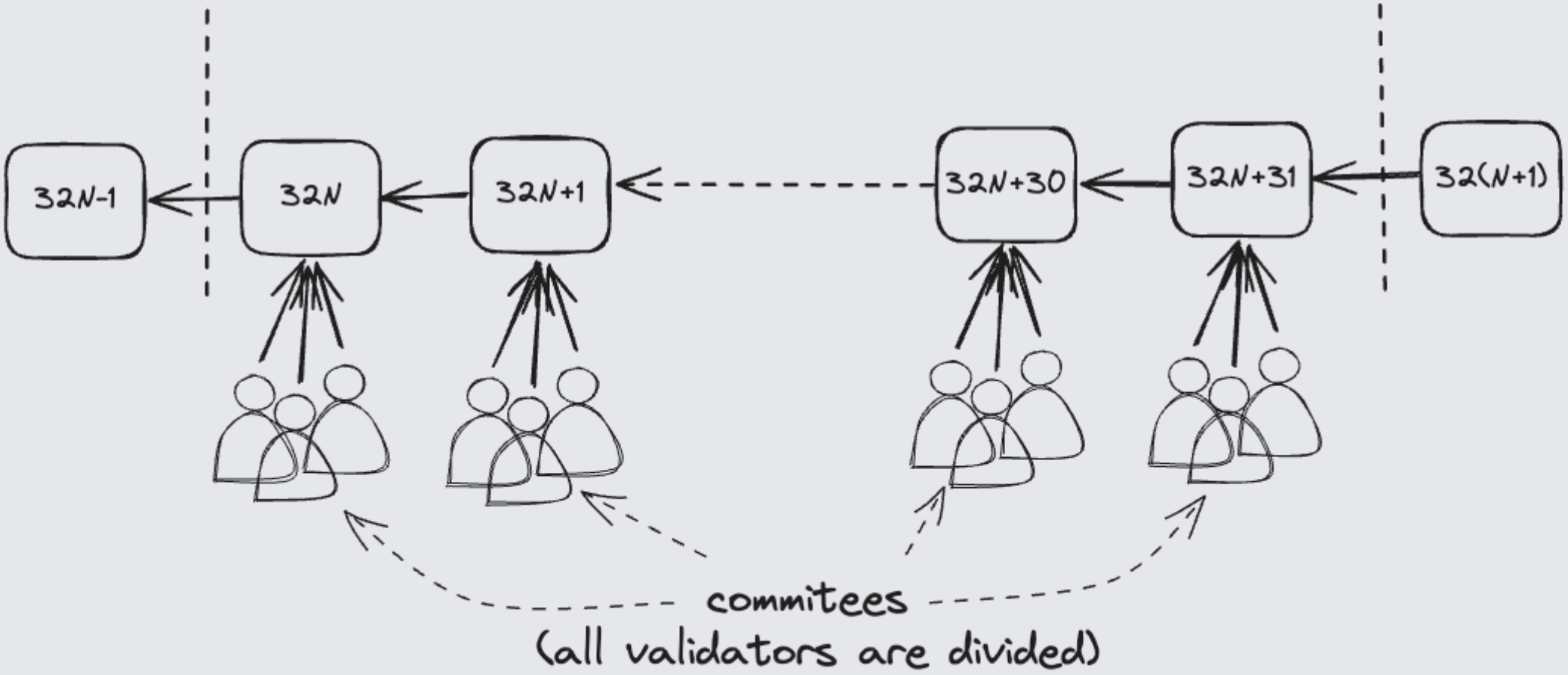
Dividing into epochs



Block proposals

- For each slot, a block is proposed by a "randomly" selected validator.
- The selection process is based on RANDAO, a cryptographic source of randomness.
- Selected validator collects transactions from the transaction pool and proposes valid block.

Attestations



Attestations data

- Head: validator's vote for that slot.
- Source: for finalisation process.
- Target: for finalisation process.

Justification and finalization

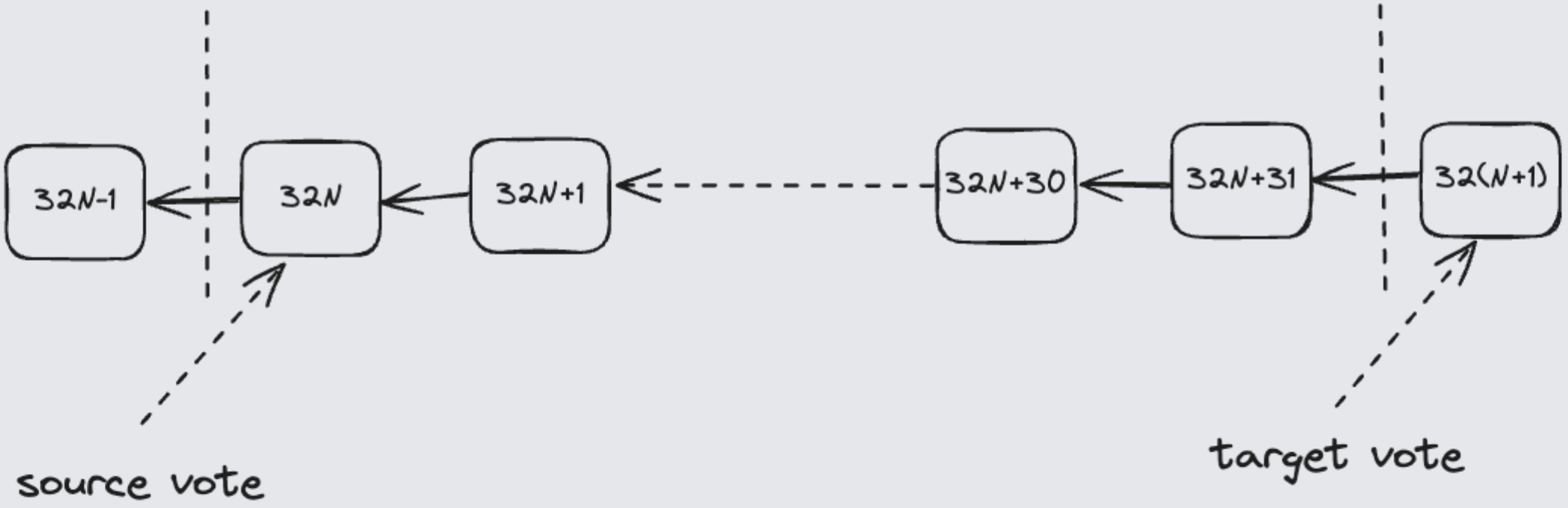
1. **Round 1** (ideally leading to justification):
    - I tell the network what I think is the best checkpoint.
    - I hear from the network what all the other validators think is the best checkpoint.
    - If I hear that 2/3 of the validators agree with me, I justify the checkpoint.
2. **Round 2** (ideally leading to finalisation):
    - I tell the network my justified checkpoint, the collective view I gained from 1.
    - I hear from the network what all the other validators think the collective view is, their justified checkpoints.
    - If I hear that of the validators agree with me, I finalise the checkpoint.

Source: https://eth2book.info/capella/part2/consensus/casper_ffg/

Justification and finalization

- Under ideal conditions, each round lasts an epoch, so it takes an epoch to justify a checkpoint and a further epoch to finalise a checkpoint.
- At the start of epoch N we are aiming to have justified checkpoint $N-1$ and to have finalised checkpoint $N-2$.

Source and target



Source and target

- Source and target votes are made simultaneously in the form of a link $s \rightarrow t$
- The role of my target vote is to broadcast my view of what the I think should be the next checkpoint to be justified.
- It is my round 1 vote. My target vote is a soft commitment not to revert that checkpoint as long as I hear from 2/3 of validators that they also commit to that checkpoint.
- An honest validator's target vote will be the checkpoint of the current epoch that descends from the source checkpoint.

Source and target

- The role of my source vote is to broadcast that I've seen support from 2/3 of the network for checkpoint s, and that it is the most recent such checkpoint that I know about.
- It is my round 2 vote announcing the collective view that I've heard. By making this source vote I am upgrading my previous soft commitment not to revert the checkpoint to a hard commitment never to revert it.
- An honest validator's source vote will always be the highest justified checkpoint in its view of the chain.

The Casper commandments

- **Commandment 1**: a validator must not publish distinct votes $s_1 \to t_1$ and $s_2 \to t_2$ such that $h(t_1) = h(t_2)$
- **Commandment 2**: a validator must not publish distinct votes $s_1 \to t_1$ and $s_2 \to t_2$ such that $h(s_1) < h(s_2) < h(t_2) < h(t_1)$

$h(c)$ is the height of checkpoint c, in Ethereum $h(c) = epoch(c)$

Slashing

- Any validator that violates either of the Casper commandments is liable to being slashed.
- There are also minor penalties for missed attestations / sync committee

Quiz time!