

Introduction to Ethereum Consensus

Francesco D'Amato
Yann Vonlanthen

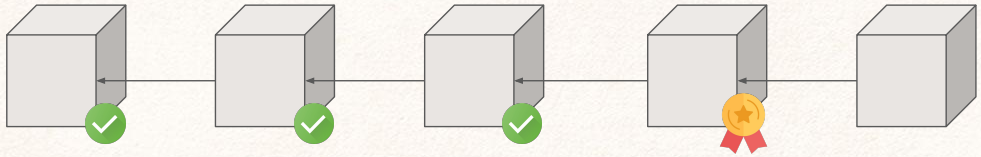
Ethereum Foundation


inactivity leak
→ continuous reconfiguration
→ weak subjectivity period
50% safety
1M validators
→ beacon state
dynamic availability
→ healing from asynchrony
on-chain votes
fork-choice store
sleepy consensus



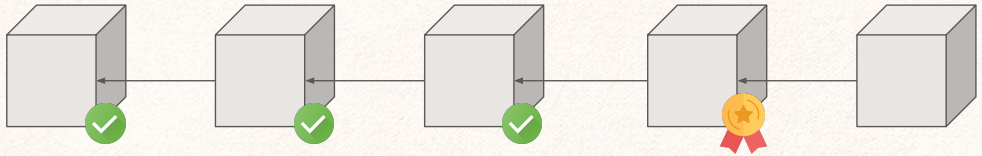
High-Level Overview

Yann Vonlanthen
Ethereum Foundation

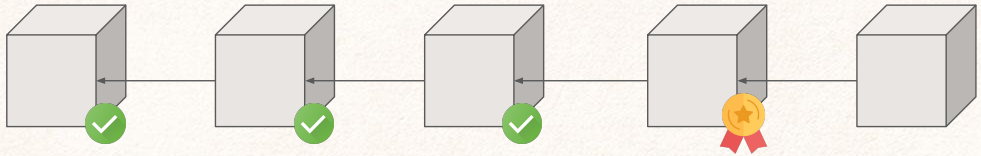






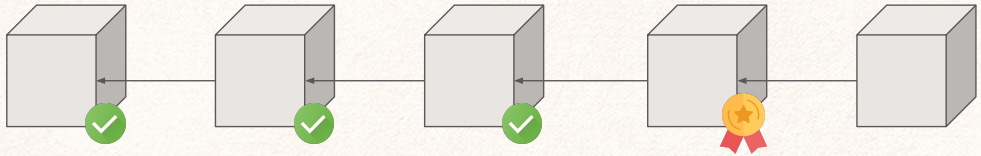




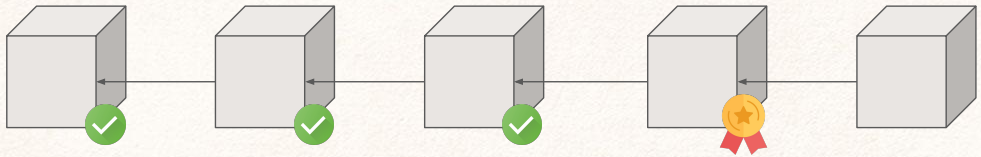


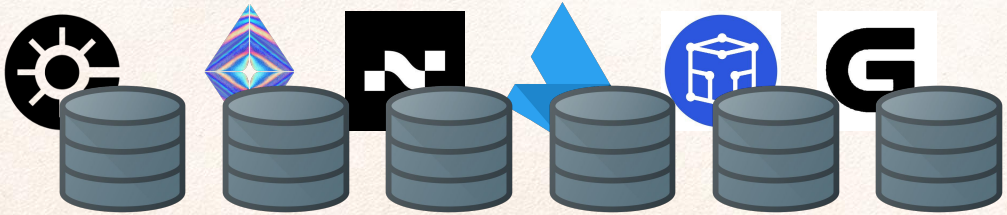


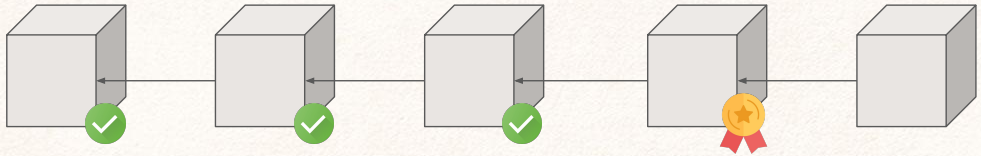


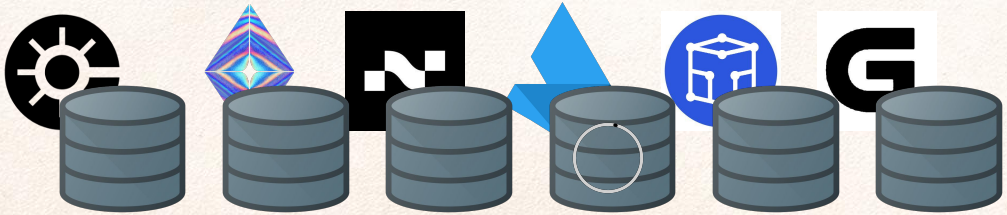




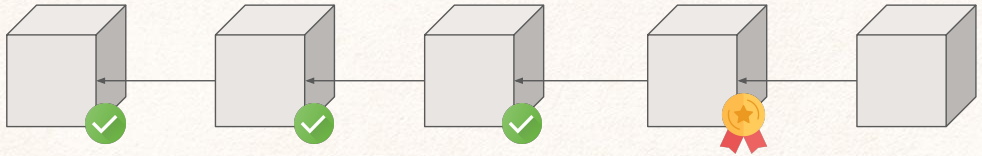


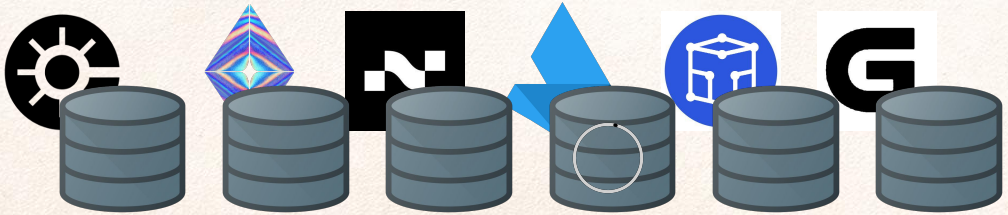










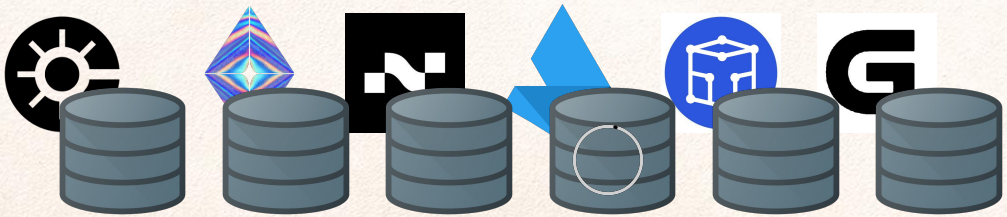


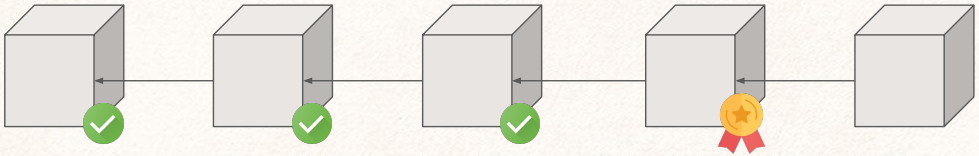
Validator Activity in Ethereum Drops 25% Following Fusaka Deployment

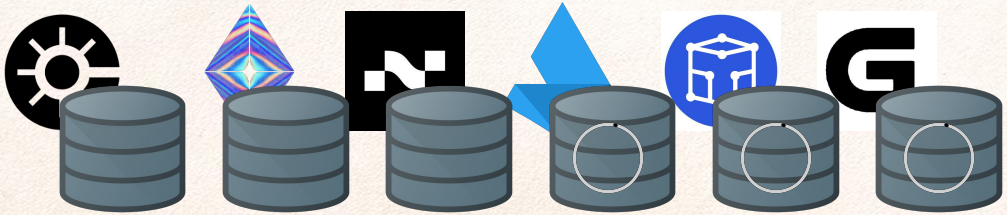
Validator activity in Ethereum drops 25% following Fusaka deployment.

05.12.2025 • ForkLog

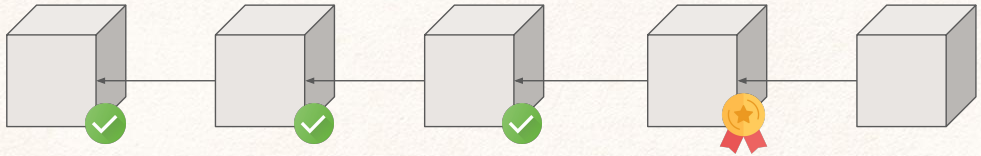
Shortly after the deployment of Fusaka, a malfunction occurred in the popular consensus client Prysm, disabling a portion of Ethereum validators.

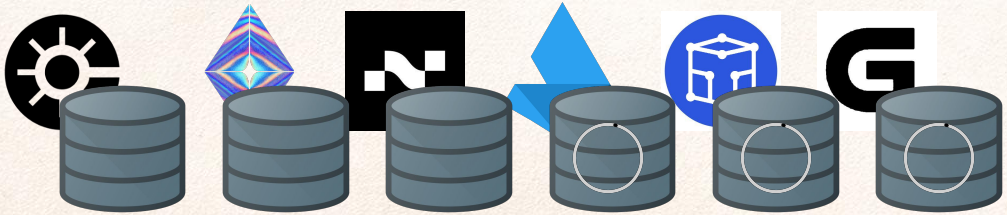




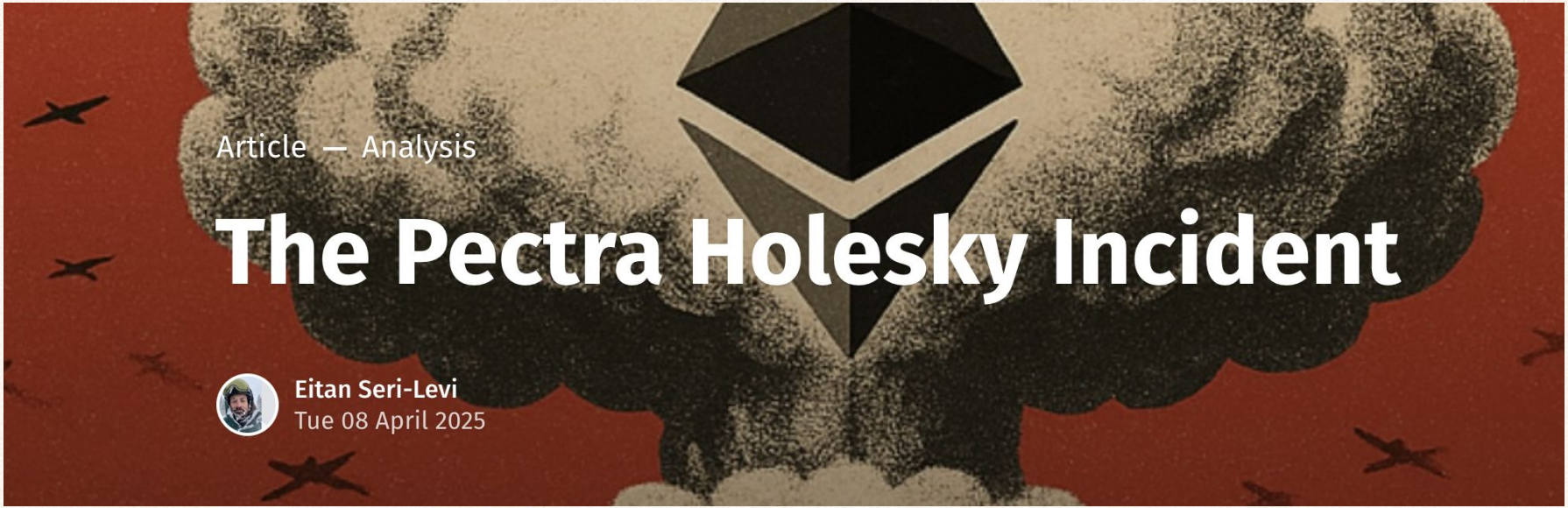




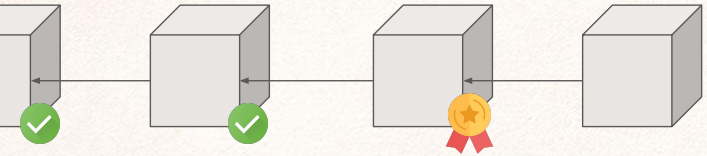




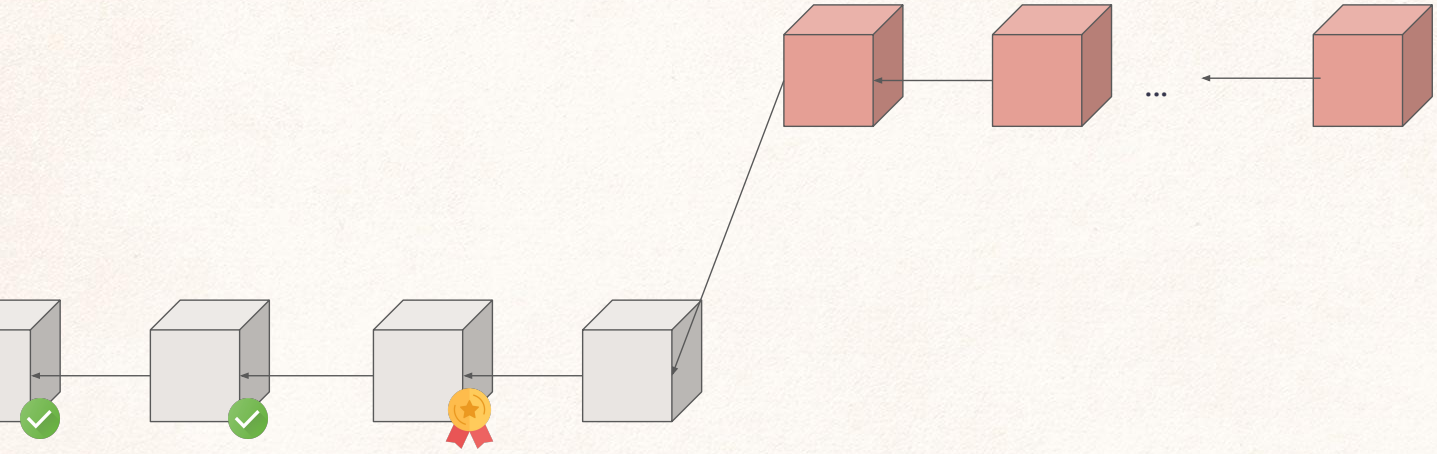
https://blog.sigmaprime.io/pectra-holesky-incident.html



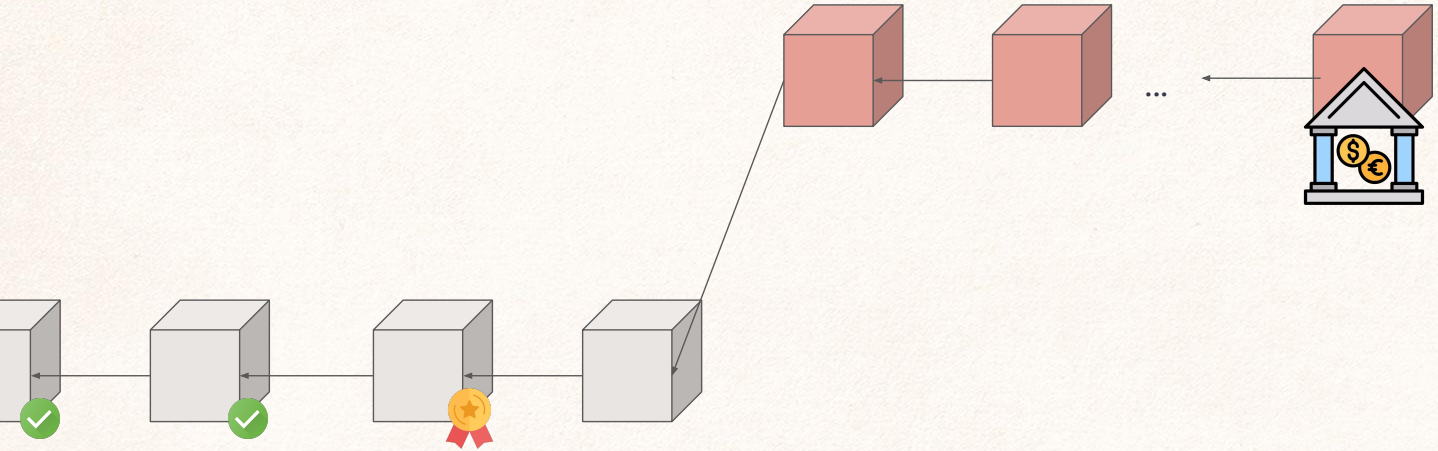






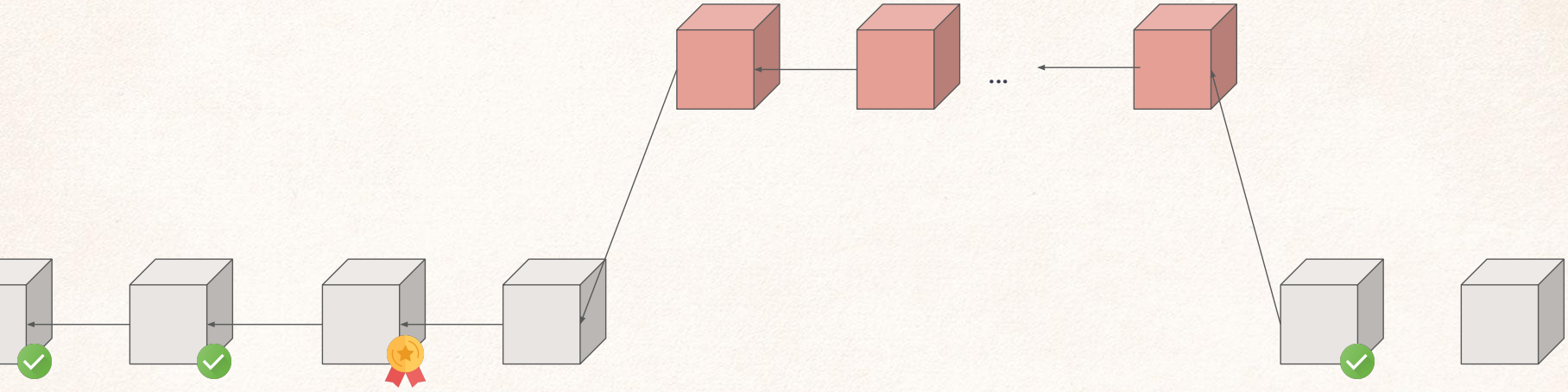




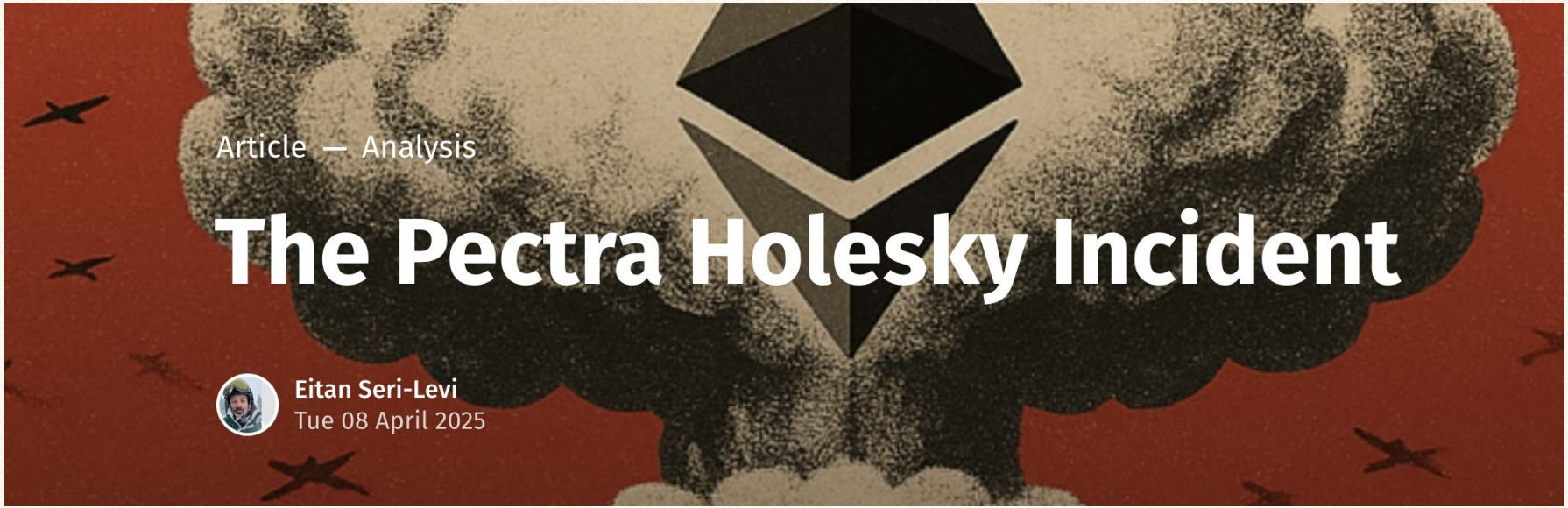




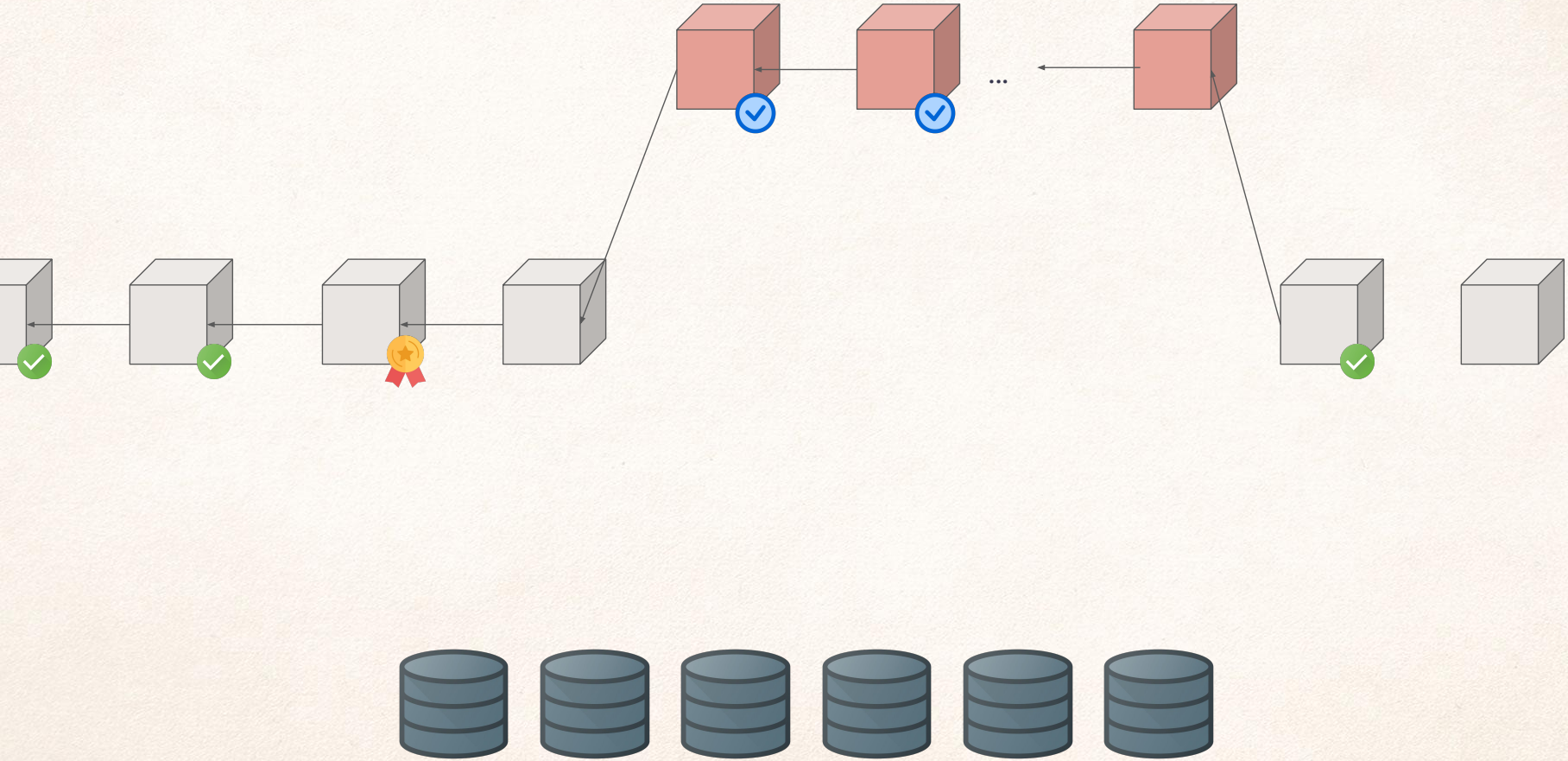


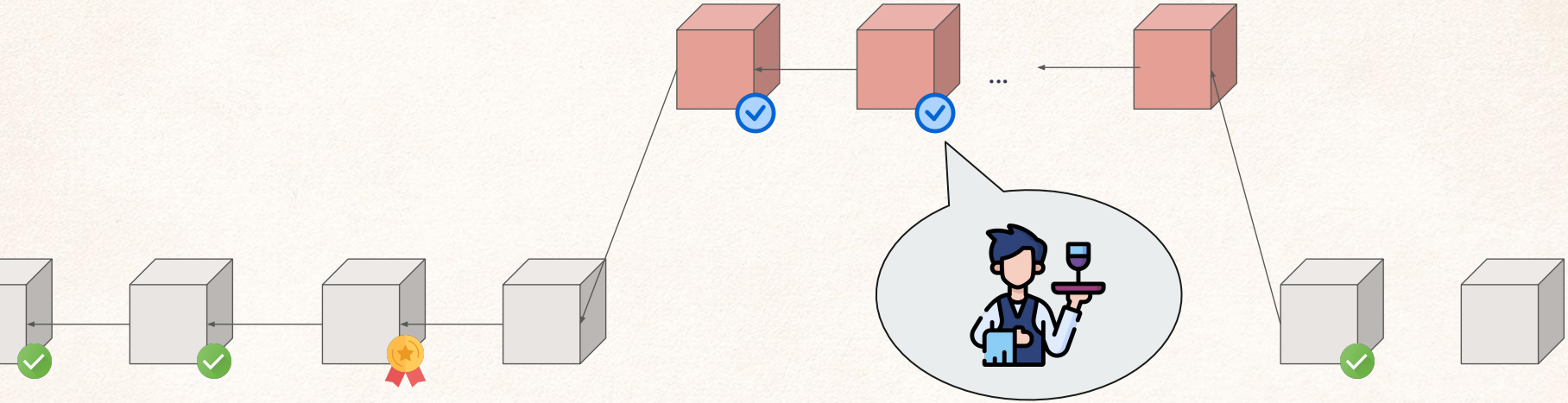


https://blog.sigmaprime.io/pectra-holesky-incident.html



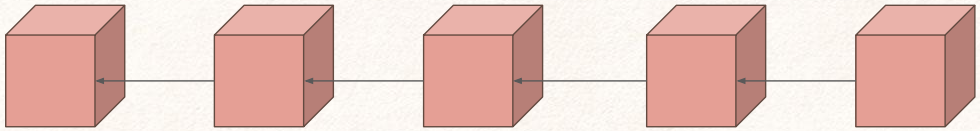


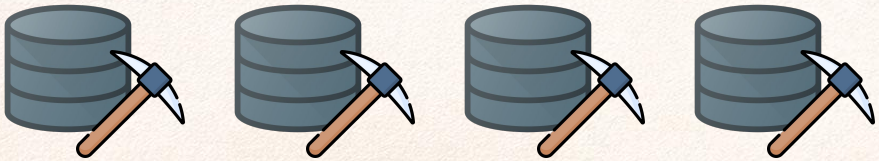





Proof-of-Work

A Short Proof-of-Work Primer

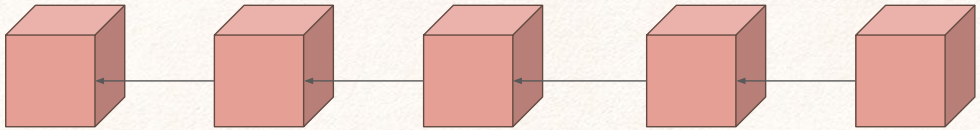


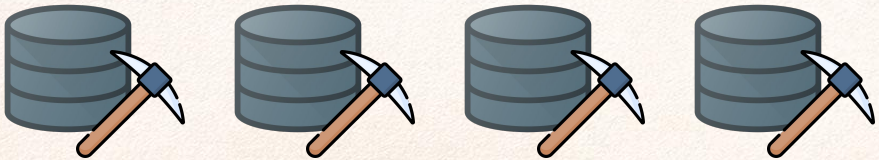


A Short Proof-of-Work Primer

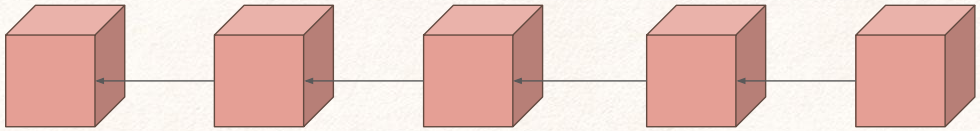
$$\mathcal{F}_d(c, x) \rightarrow \{true, false\}$$

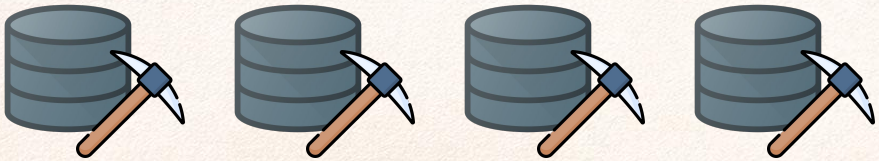
c: challenge
x: nonce
d: difficulty



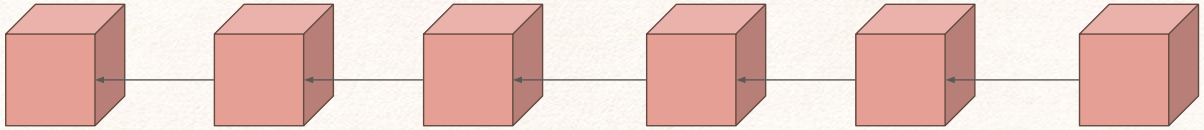


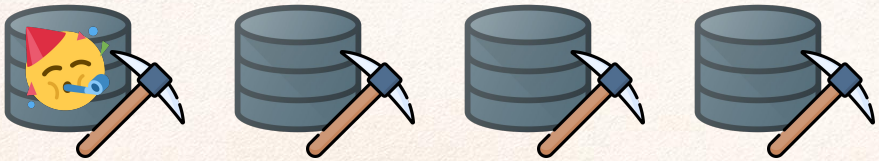
A Short Proof-of-Work Primer





A Short Proof-of-Work Primer

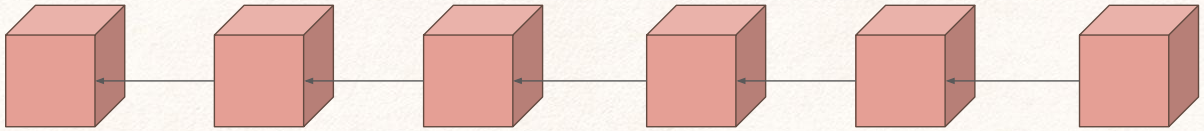


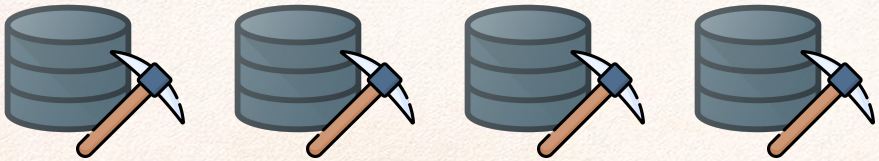


A Short Proof-of-Work Primer

$$\mathcal{F}_d(c, x) \rightarrow \{true, false\}$$

c: challenge
x: nonce
d: difficulty

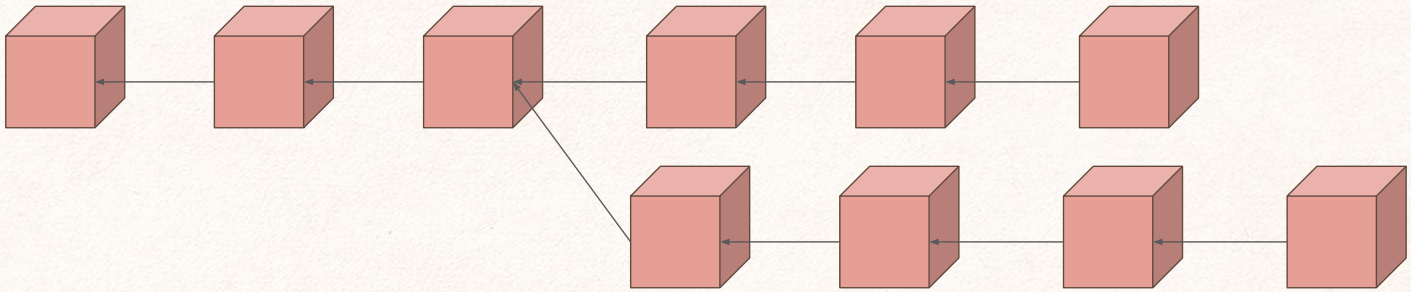


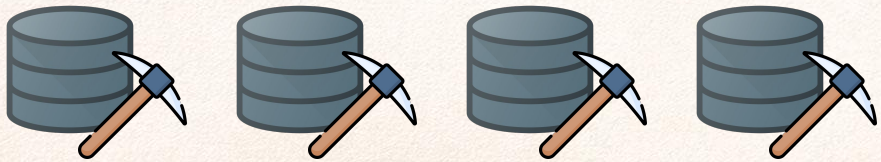


A Short Proof-of-Work Primer

$$\mathcal{F}_d(c, x) \rightarrow \{true, false\}$$

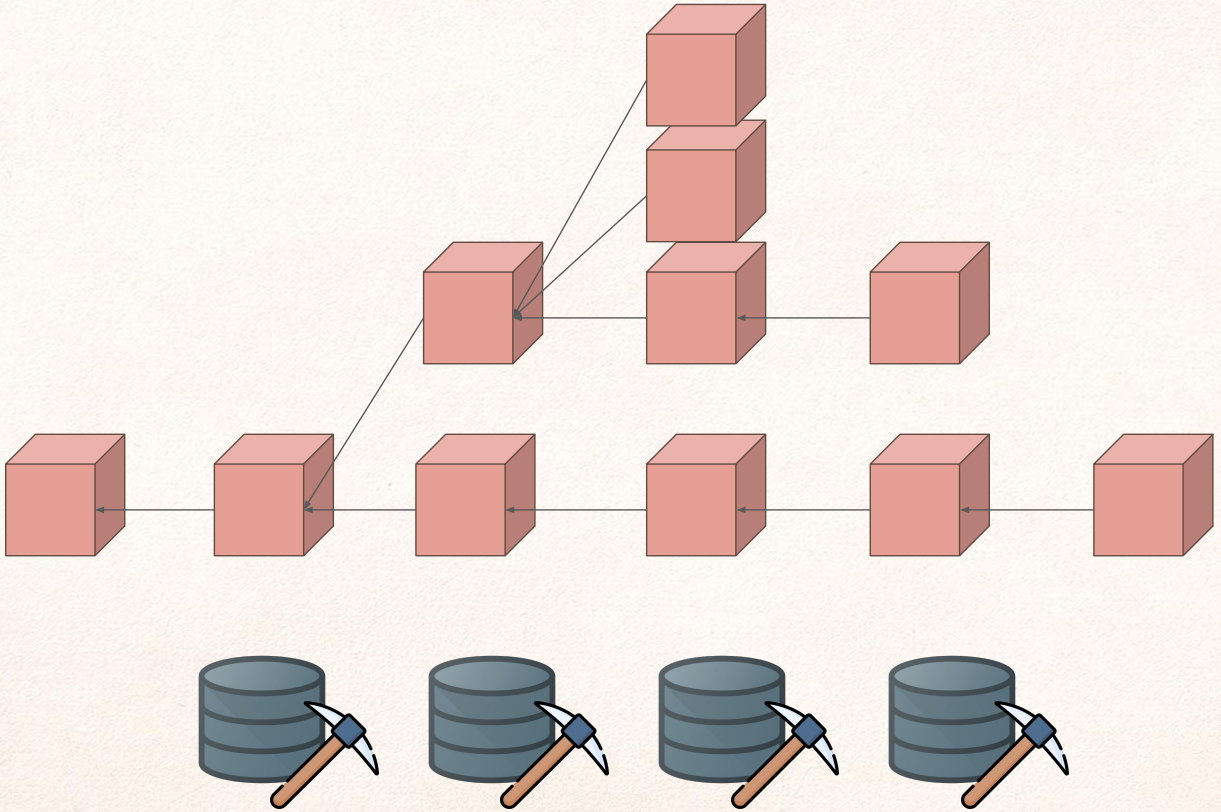
c: challenge
x: nonce
d: difficulty





Ethereum PoW (GHOST)

Greediest Heaviest
Observed SubTree



Ethereum PoW (GHOST)

Secure High-Rate Transaction Processing in Bitcoin

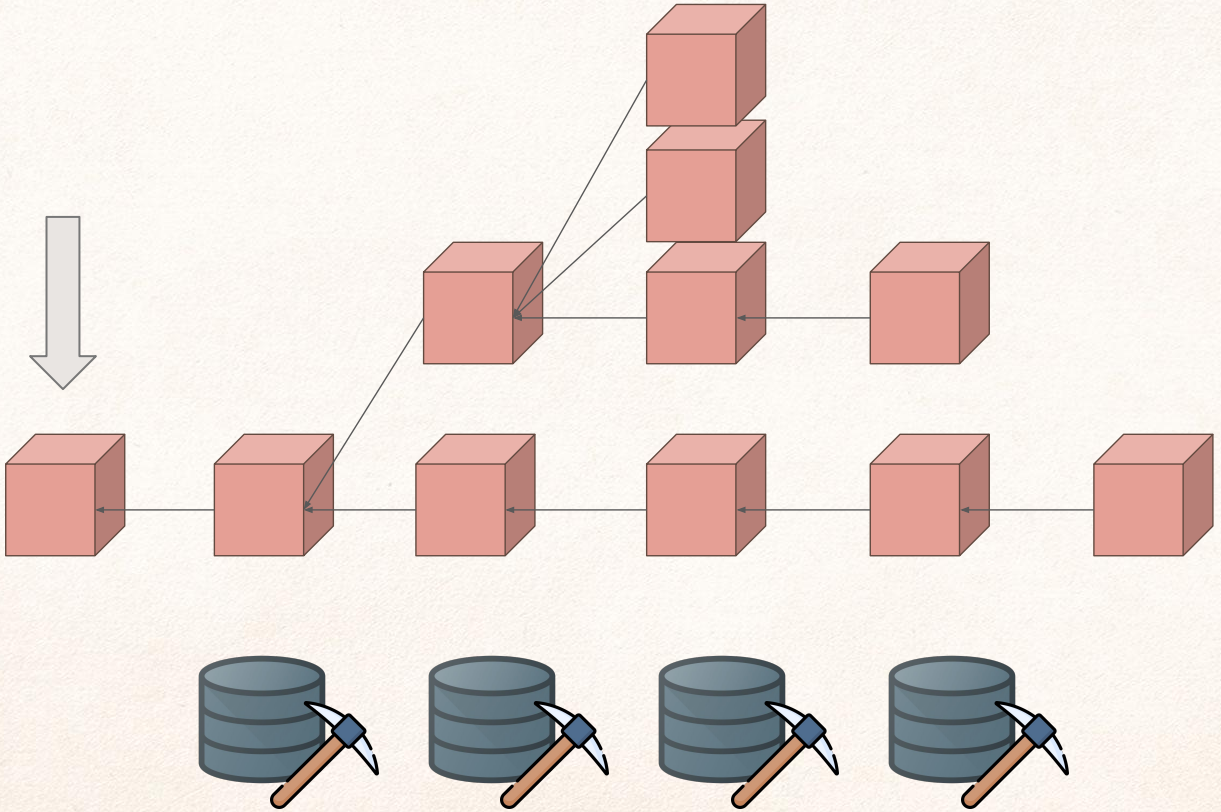
(full version)

Yonatan Sompolinsky[1] and Aviv Zohar[1,2]

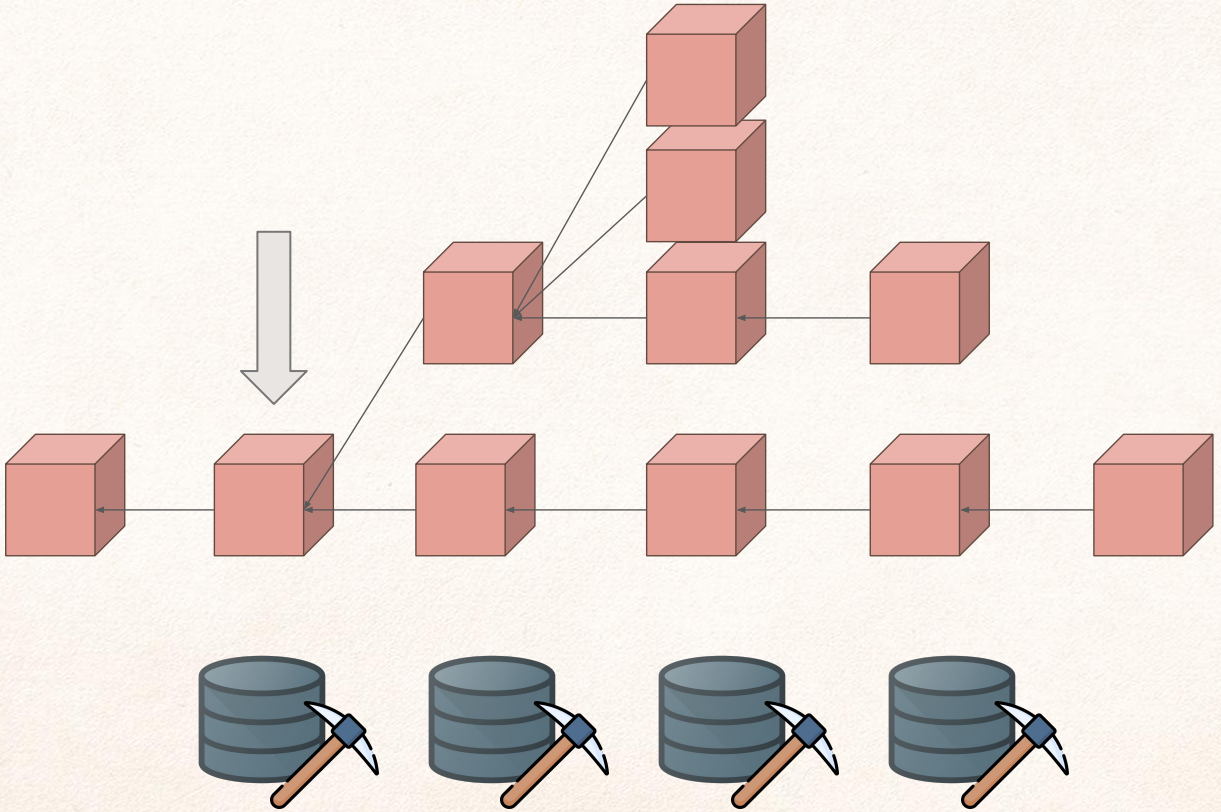
[1] School of Engineering and Computer Science,
The Hebrew University of Jerusalem, Israel
[2] Microsoft Research, Herzliya, Israel
yoni_sompo@cs.huji.ac.il, avivz@cs.huji.ac.il

Abstract. Bitcoin is a disruptive new crypto-currency based on a decentralized open-source protocol which has been gradually gaining momentum. Perhaps the most important question that will affect Bitcoin's success, is whether or not it will be able to scale to support the high volume of transactions required from a global currency system. We investigate the implications of having a higher transaction throughput on Bitcoin's security against double-spend attacks. We show that at high throughput, substantially weaker attackers are able to reverse payments they have made, even well after they were considered accepted by recipients. We address this security concern through the GHOST rule, a modification to the way Bitcoin nodes construct and re-organize the block chain, Bitcoin's core distributed data-structure. GHOST has been adopted and a variant of it has been implemented as part of the Ethereum project, a second generation distributed applications platform.

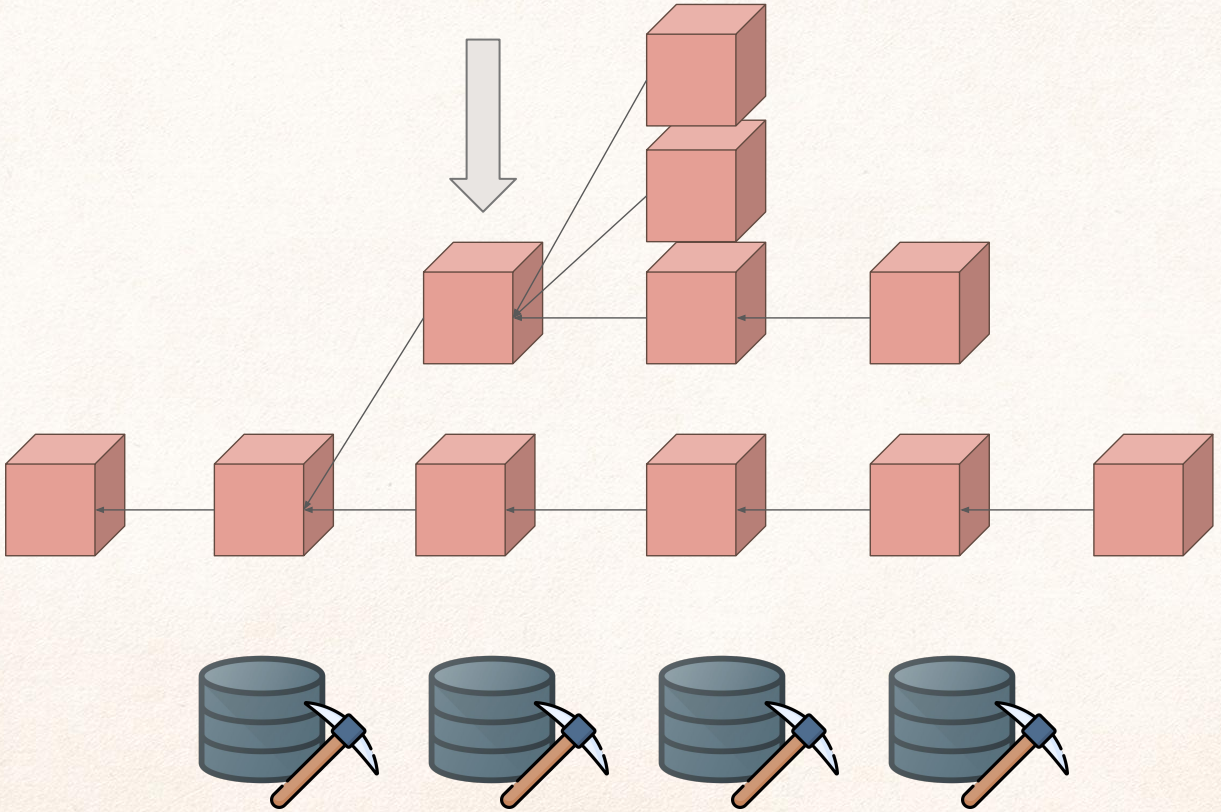
Ethereum PoW (GHOST)



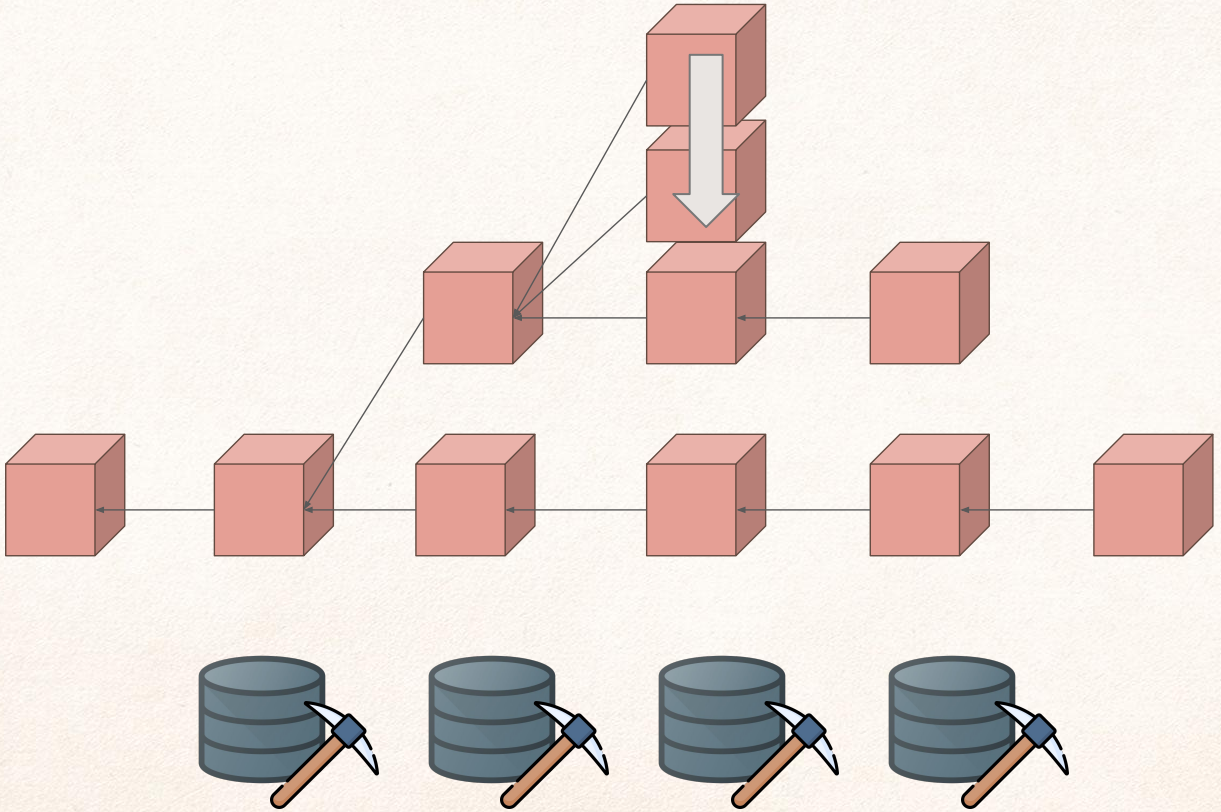
Ethereum PoW (GHOST)



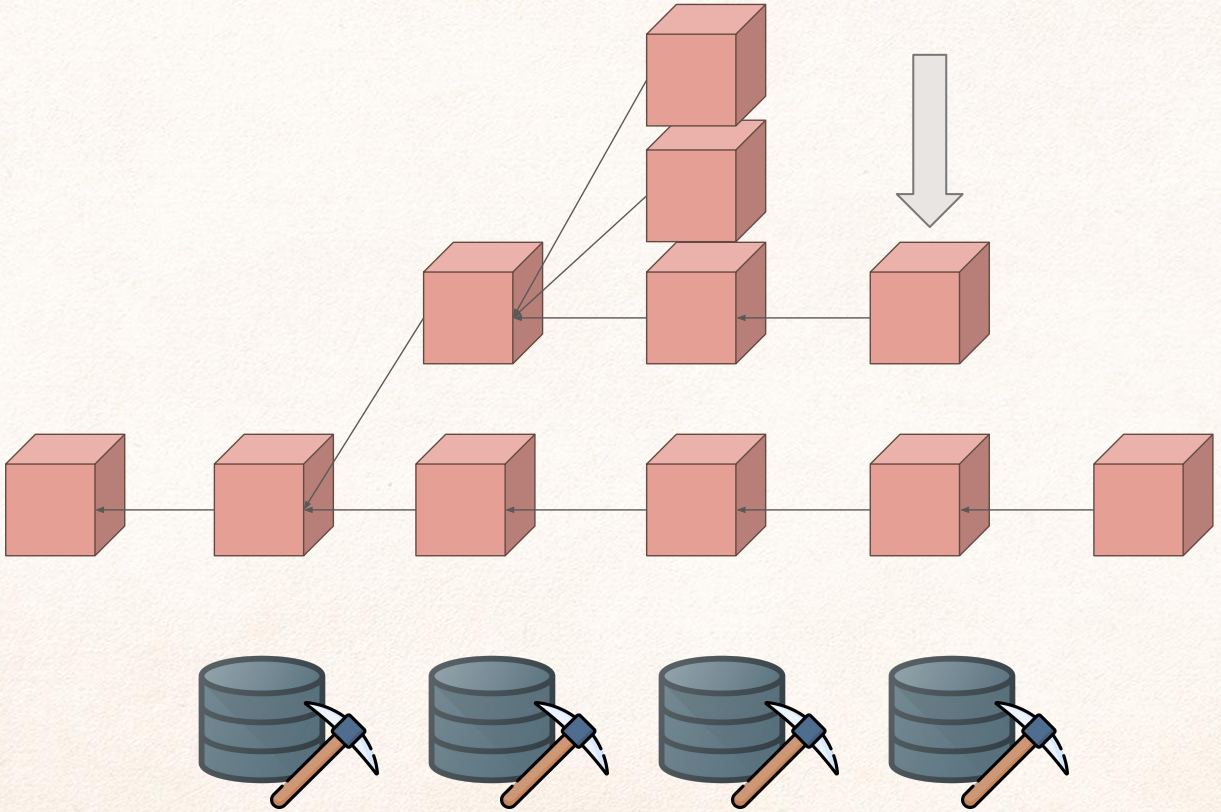
Ethereum PoW (GHOST)



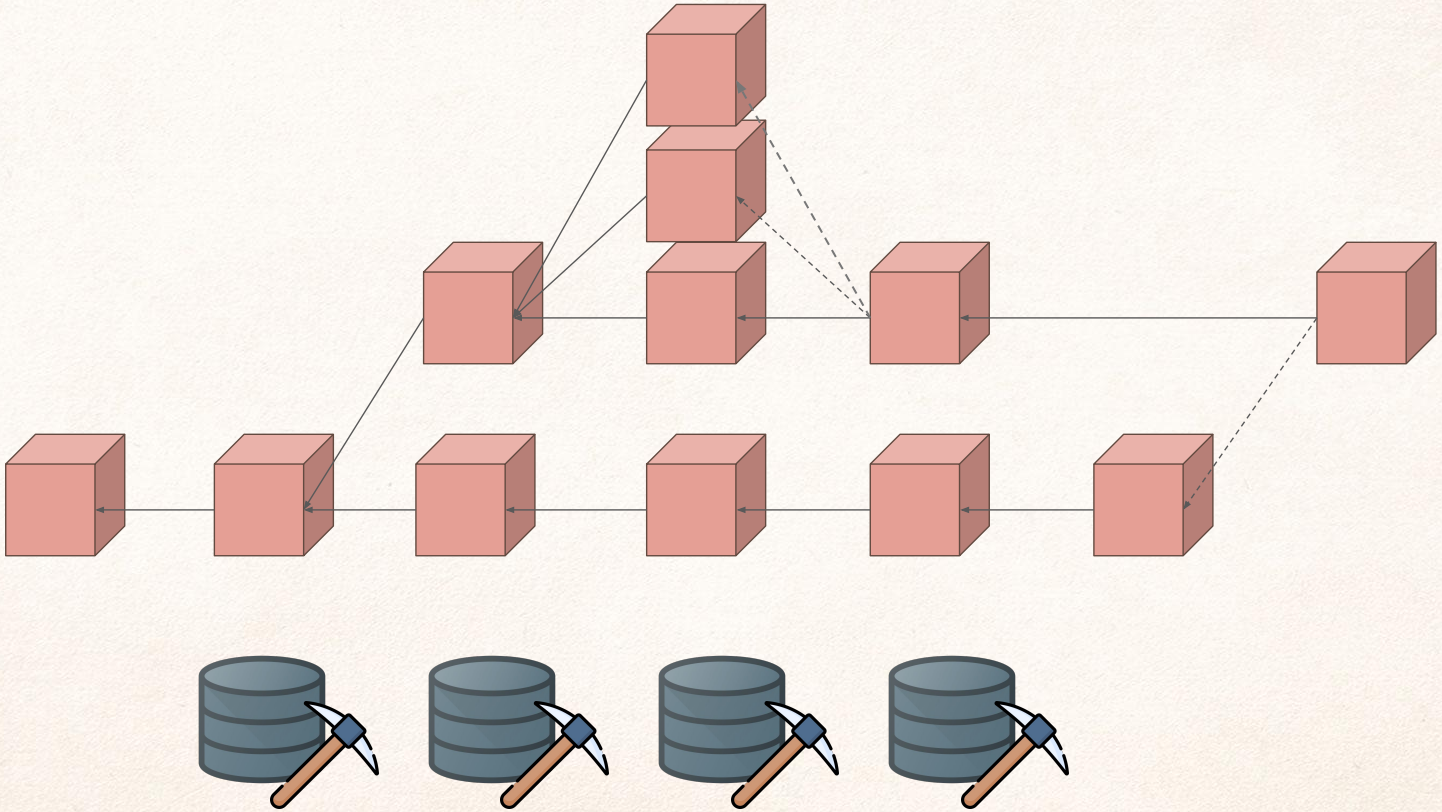
Ethereum PoW (GHOST)



Ethereum PoW (GHOST)



Ethereum PoW - Uncles

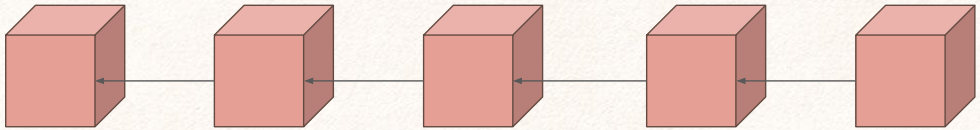


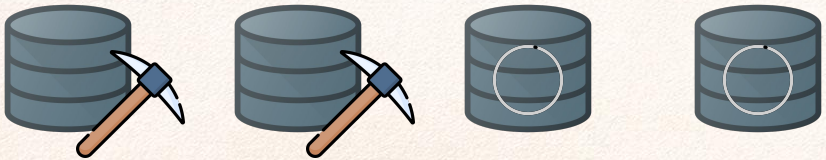
https://ethereum.org/en/whitepaper/

A Short Proof-of-Work Primer

$$\mathcal{F}_d(c, x) \rightarrow \{true, false\}$$

c: challenge
x: nonce
d: difficulty





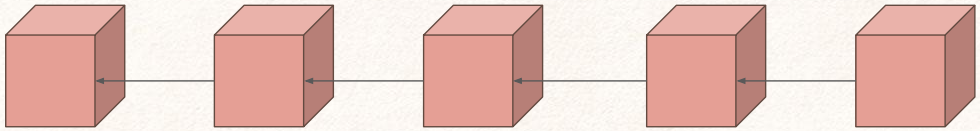
Sleepy Model

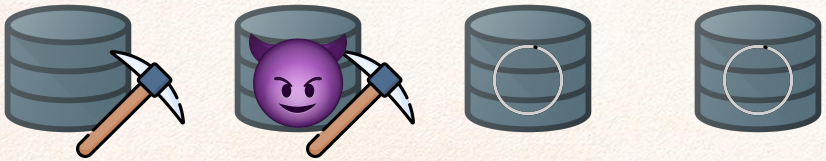


f < 50% of awake



1 honest awake







$f < n / 3$



Δ after **GST** (part. sync)



All online after **GAT**



f < n / 3



Δ after **GST** (part. sync)



All online after **GAT**



f < 50% of awake



Δ (Synchronous)



1 honest awake

Finality Chain

 f < n / 3

 Δ after **GST** (part. sync)

 All online after **GAT**

Available Chain

 f < 50% of awake

 Δ (Synchronous)

 1 honest awake

Finality Chain



f < n / 3 + economic security



Δ after **GST** (part. sync)



All online after **GAT**

Available Chain



f < 50% of awake

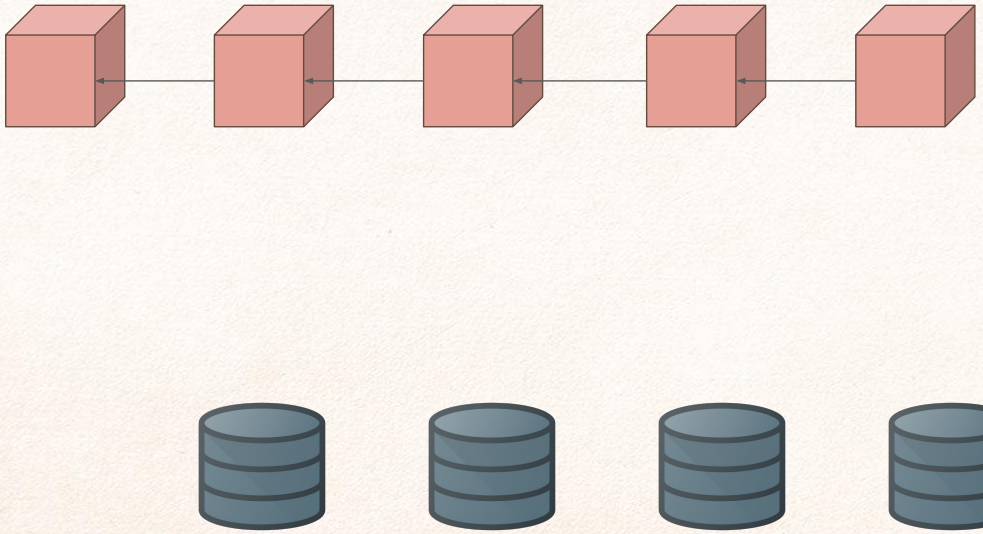


Δ (Synchronous)

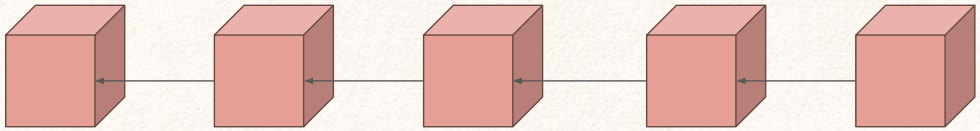


1 honest awake


Switch to Proof-of-Stake

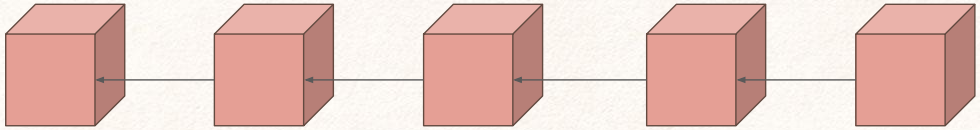


Sybil Resistance - Staking





Leader Election - RANDAO





Leader Election - RANDAO



Optimal RANDAO Manipulation in Ethereum

Kaya Alpturer
Princeton University, NJ, USA

S. Matthew Weinberg
Princeton University, NJ, USA

Abstract

It is well-known that RANDAO manipulation is possible in Ethereum if an adversary controls the proposers assigned to the last slots in an epoch. We provide a methodology to compute, for any fraction α of stake owned by an adversary, the maximum fraction $f(\alpha)$ of rounds that a strategic adversary can propose. We further implement our methodology and compute $f(\cdot)$ for all α. For example, we conclude that an optimal strategic participant with 5% of the stake can propose a 5.048% fraction of rounds, 10% of the stake can propose a 10.19% fraction of rounds, and 20% of the stake can propose a 20.68% fraction of rounds.

Forking the RANDAO: Manipulating Ethereum's Distributed Randomness Beacon

Ábel Nagy
Eötvös Loránd University
Budapest, Hungary
nagyabi@gmail.com

János Tapolcai
University of Technology and Economics
Budapest, Hungary
tapolcai@tmit.bme.hu

István András Seres
Eötvös Loránd University
Budapest, Hungary
seresistvanandras@gmail.com

Bence Ladóczki
University of Technology and Economics
HUN-REN Information Systems Research Group
Budapest, Hungary
ladoczki.bence@vik.bme.hu

Abstract

Proof-of-stake consensus protocols often rely on distributed randomness beacons (DRBs) to generate randomness for leader selection. This work analyses the manipulability of Ethereum's DRB implementation, *RANDAO*, in its current consensus mechanism. Even with its efficiency, RANDAO remains vulnerable to manipulation through the deliberate omission of blocks from the canonical chain. Previous research has shown that economically rational players can withhold blocks – known as a block withholding attack or selfish mixing – when the manipulated RANDAO outcome yields greater financial rewards.

We introduce and evaluate a new manipulation strategy, the RANDAO forking attack. Unlike block withholding, whereby validators opt to hide a block, this strategy relies on selectively forking

ACM Reference Format:
Ábel Nagy, János Tapolcai, István András Seres, and Bence Ladóczki. 2025. Forking the RANDAO: Manipulating Ethereum's Distributed Randomness Beacon. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS '25), October 13–17, 2025, Taipei, Taiwan.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3719027.3744852

1 Introduction

Randomness is indispensable for (permissionless) consensus [17, 22]. Ethereum [36], the second-largest cryptocurrency by market capitalisation and the largest by transaction volume, pseudo-randomly selects block proposers using cryptographic algorithms and protocols. Back in the days, Ethereum utilised a purely *longest-chain protocol* to reach consensus, a similar construction to the



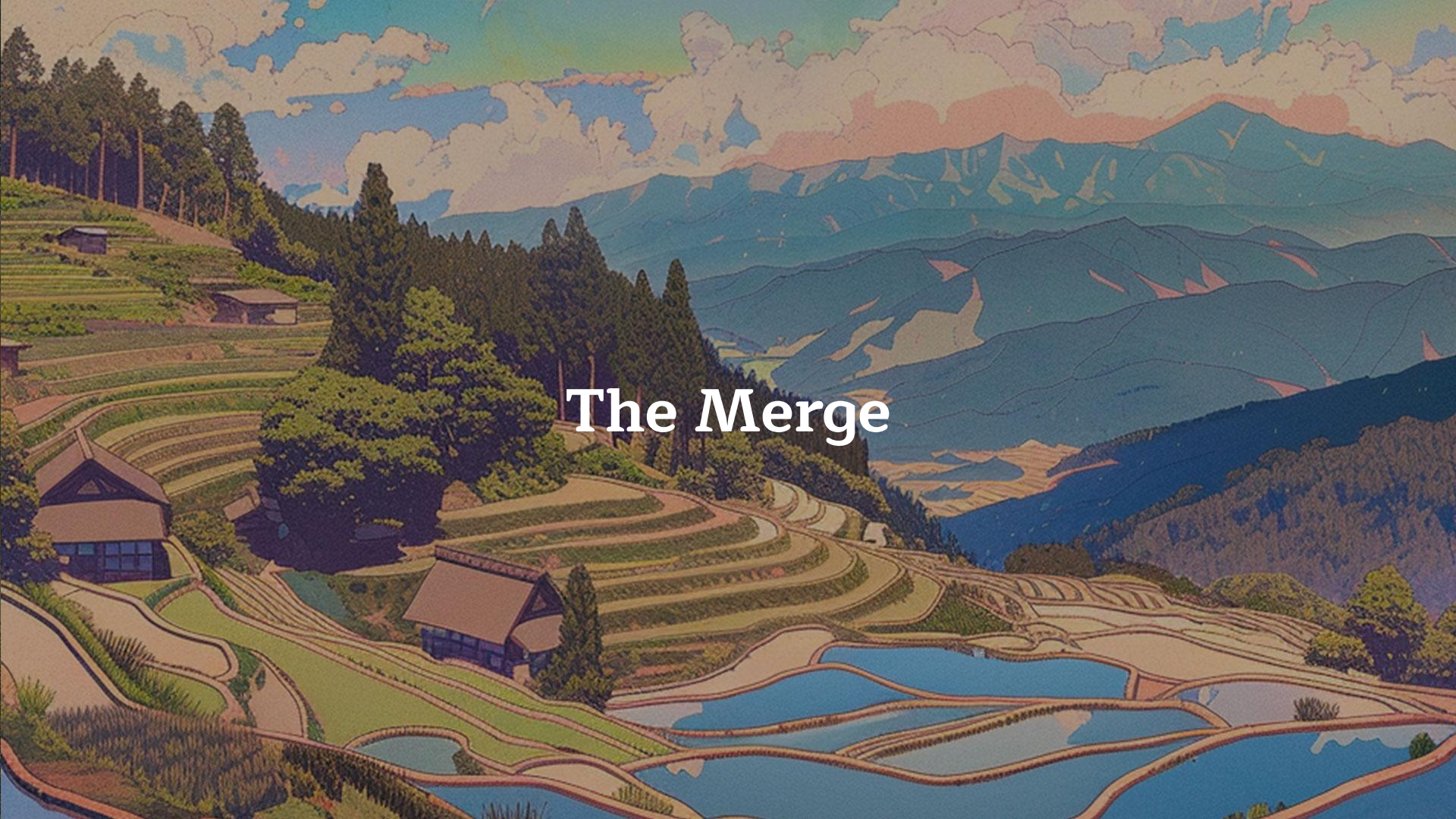
Two Lines of Research

1. **Mimicking GHOST**
    a. Voting based protocol
    b. Available chain (sleepy model)

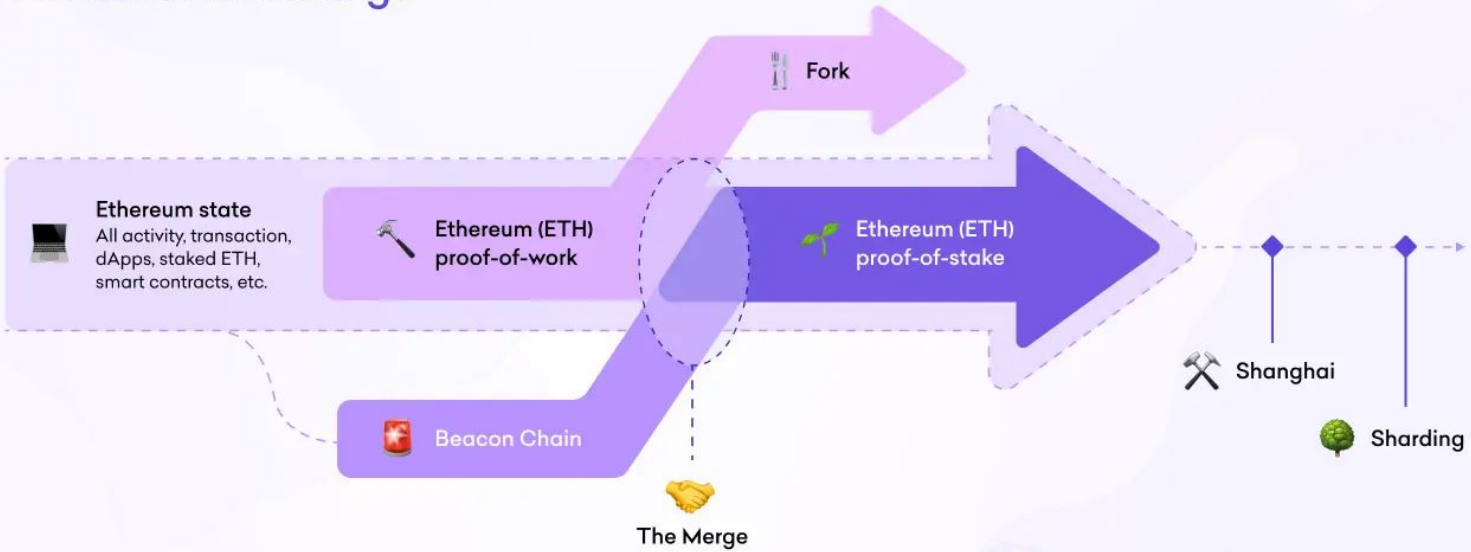
⇒ LMD-GHOST

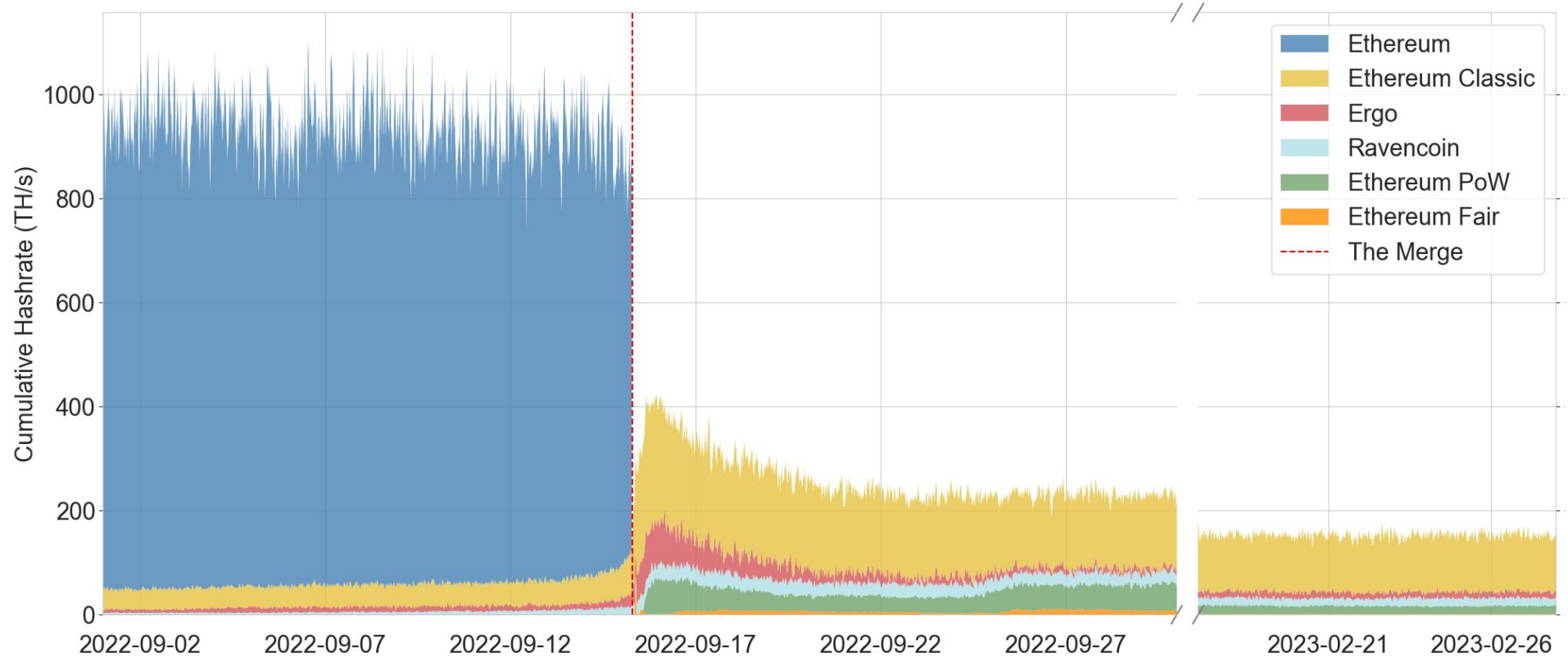
2. **Part. Sync protocol**
    a. Economic security

⇒ Casper FFG

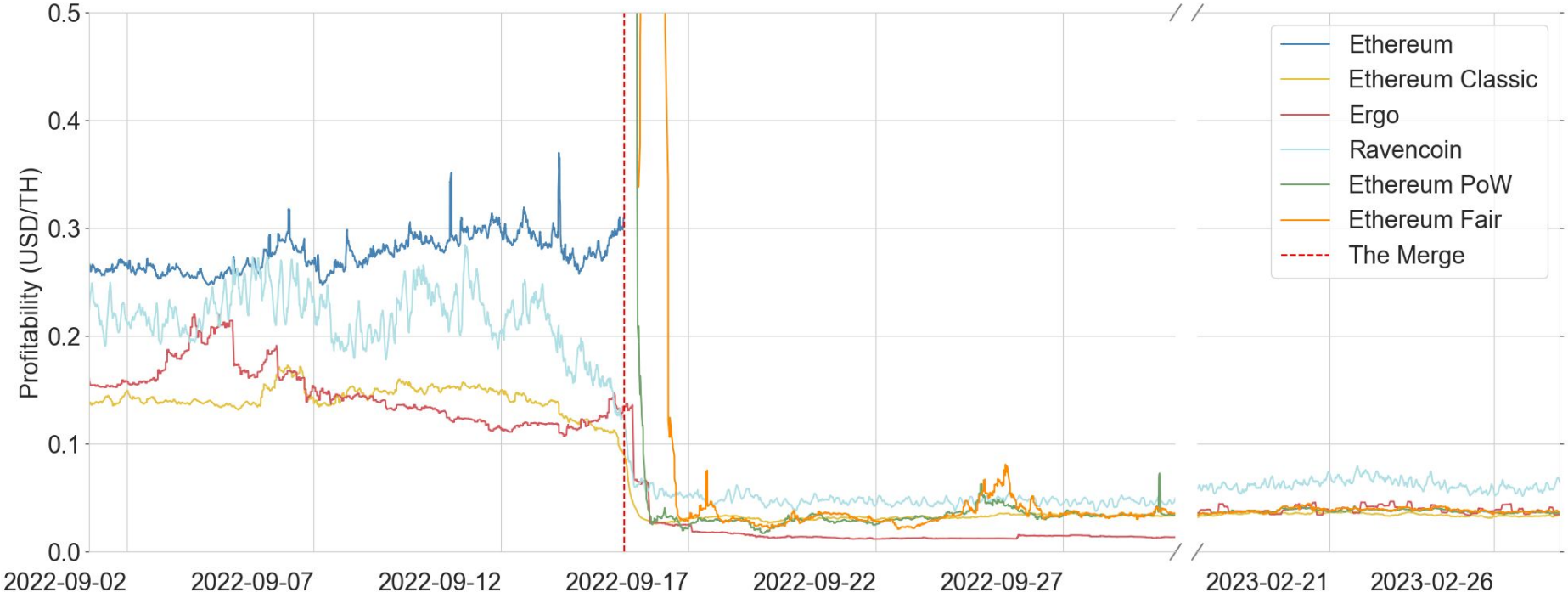

The Merge

The Ethereum Merge





Ethash Equivalent Hashrate

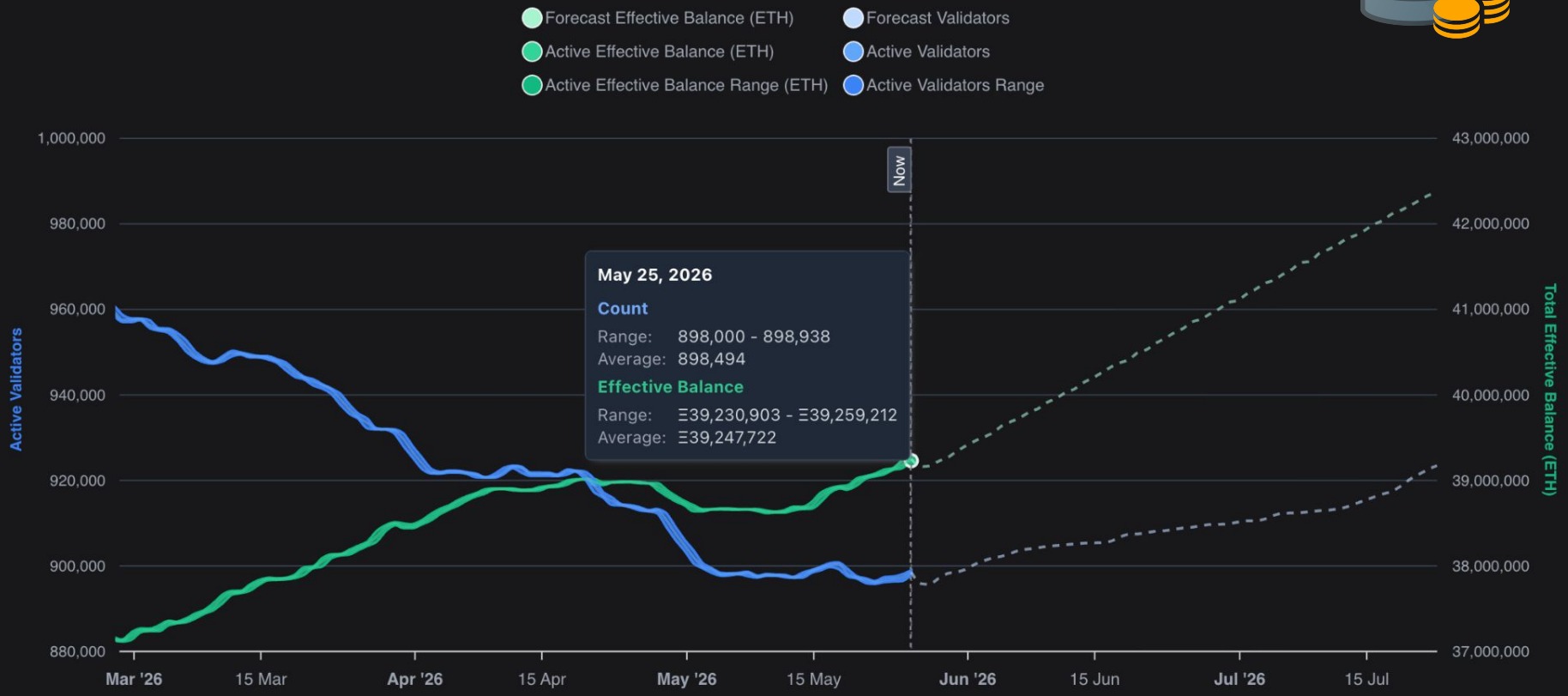


Mining Profitability


Practical Aspects

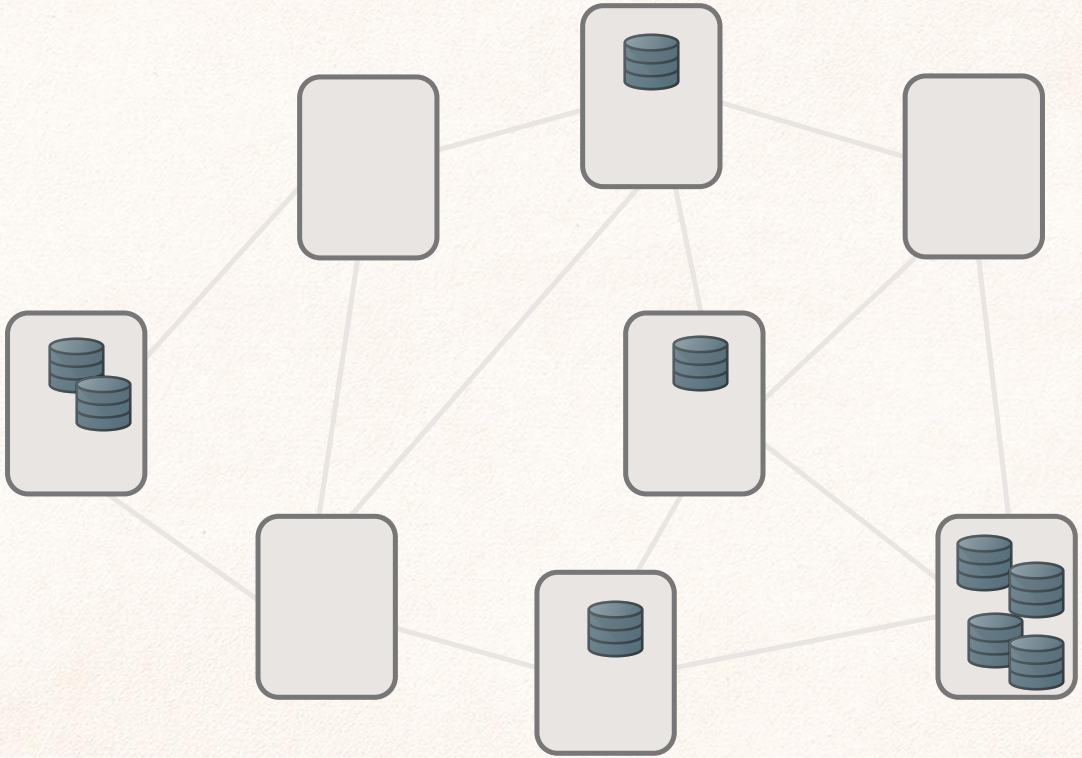
Validators



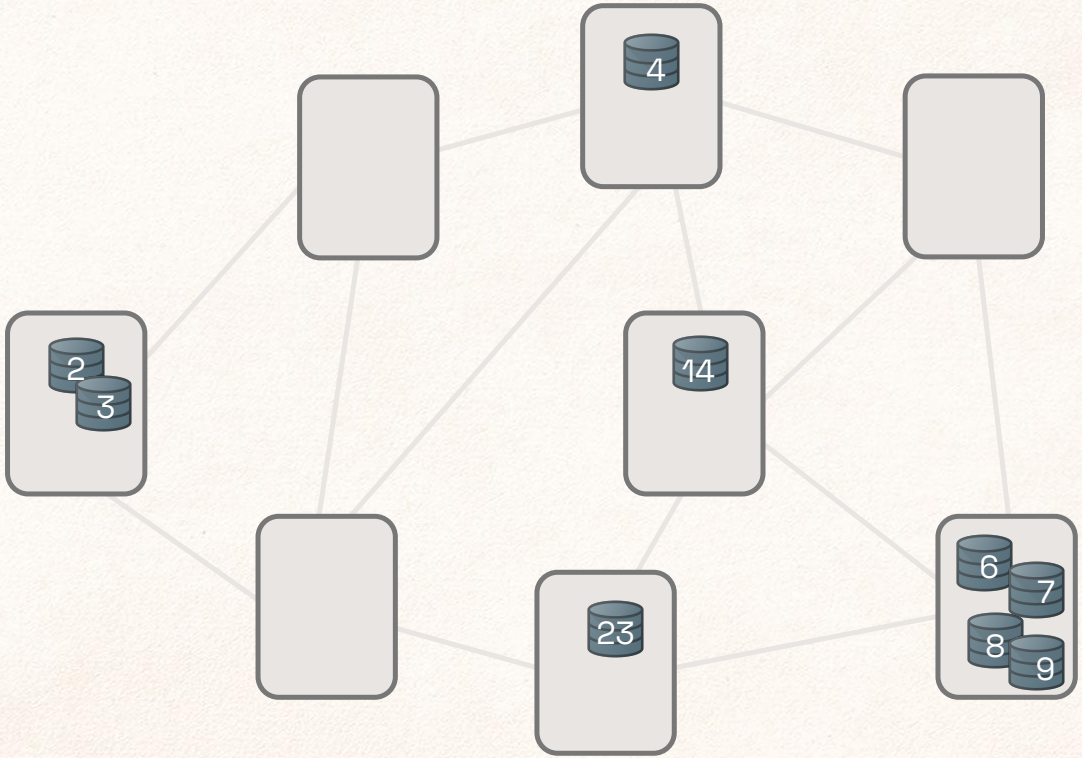


https://www.pectrified.com/mainnet

Clients

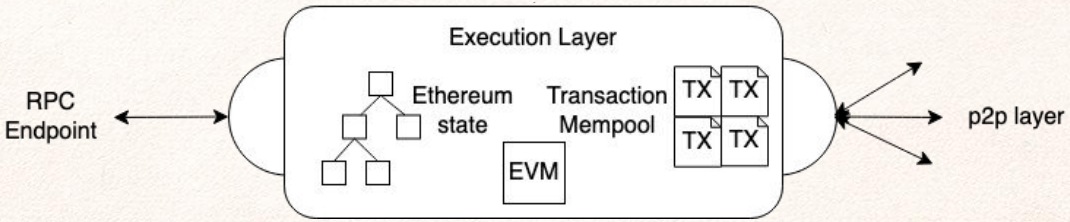


Clients

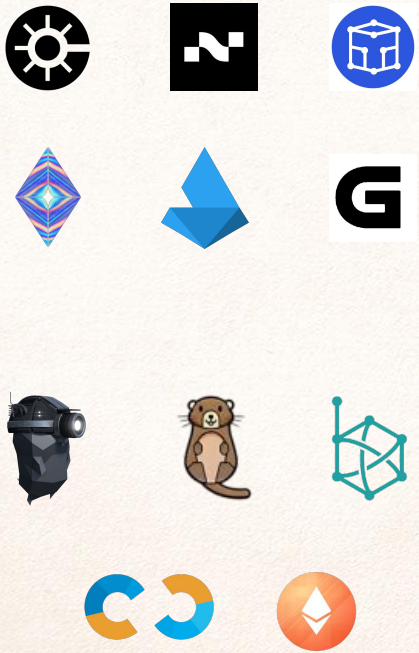


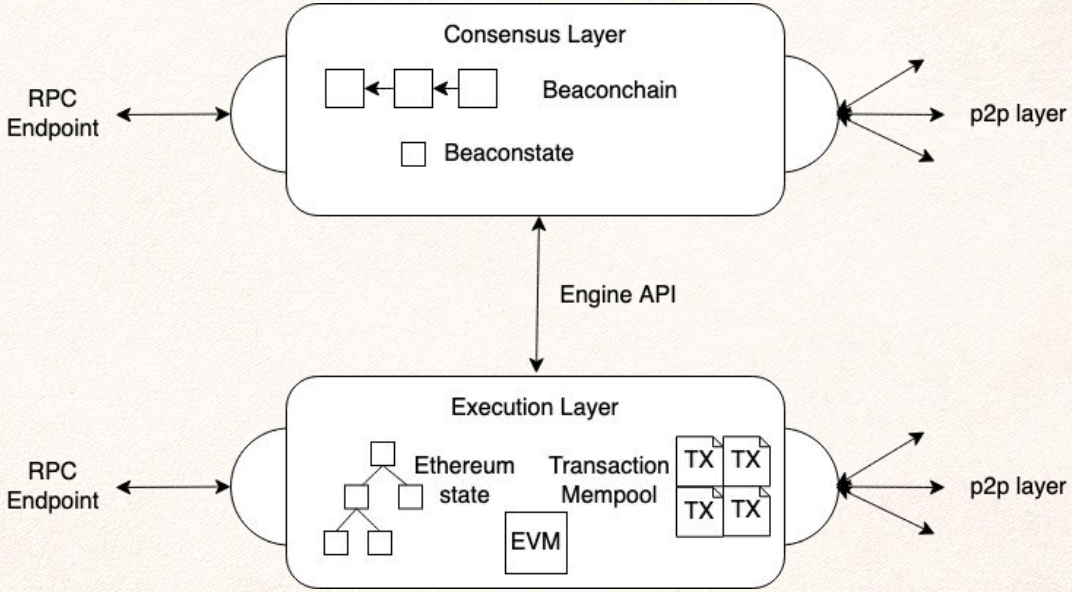
Execution Layer Clients



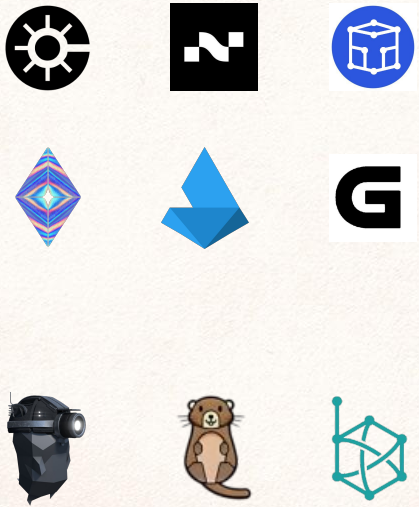


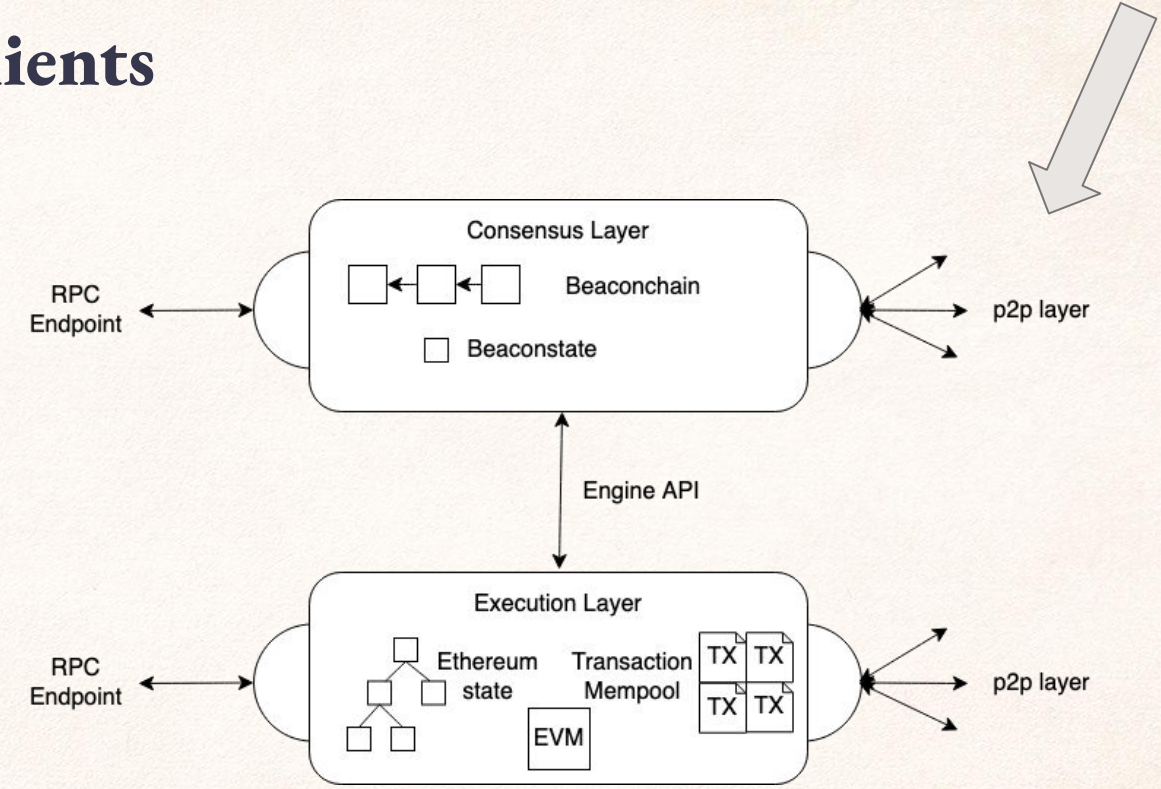
Consensus Layer Clients

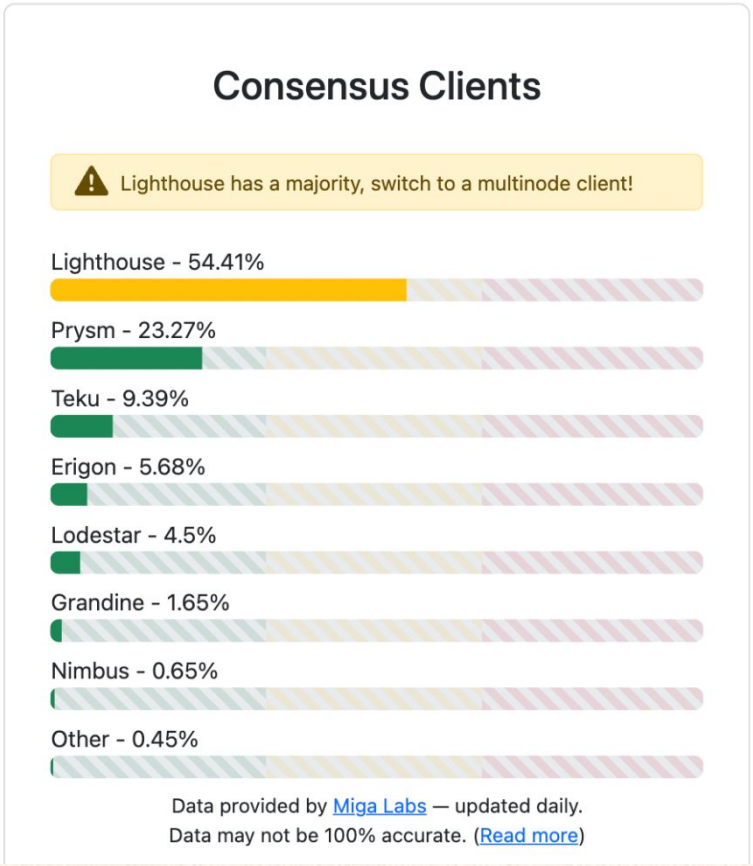




Consensus Layer Clients



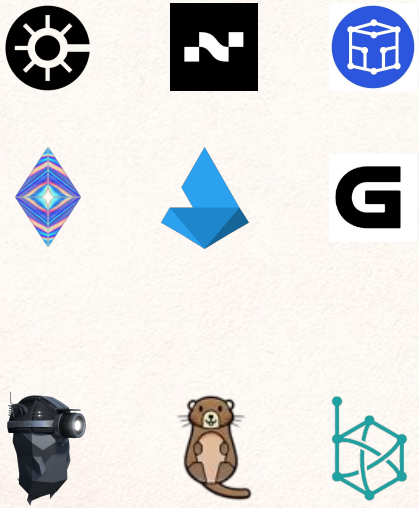


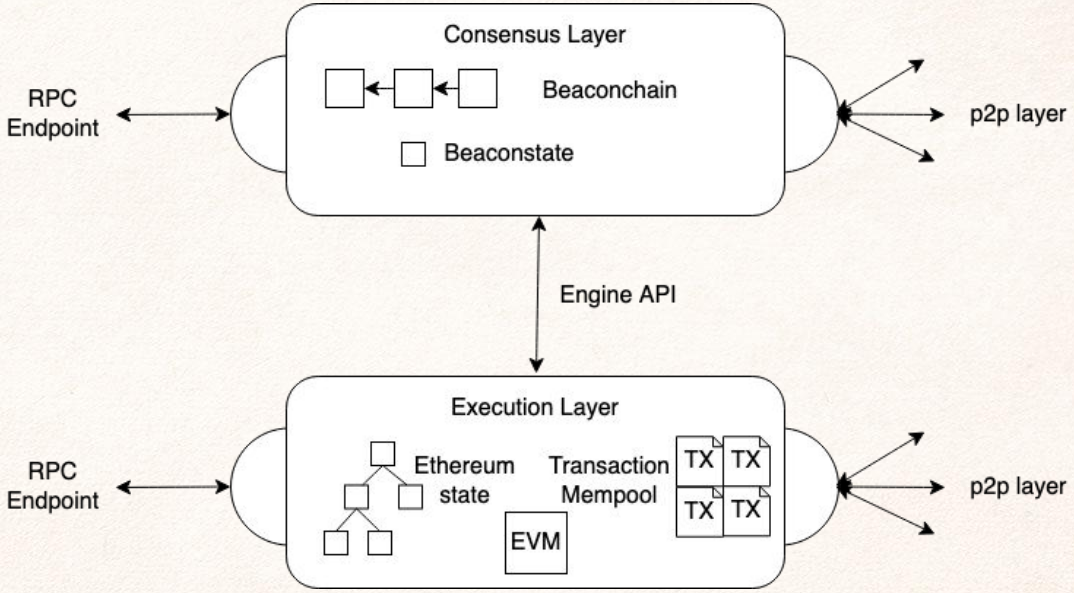




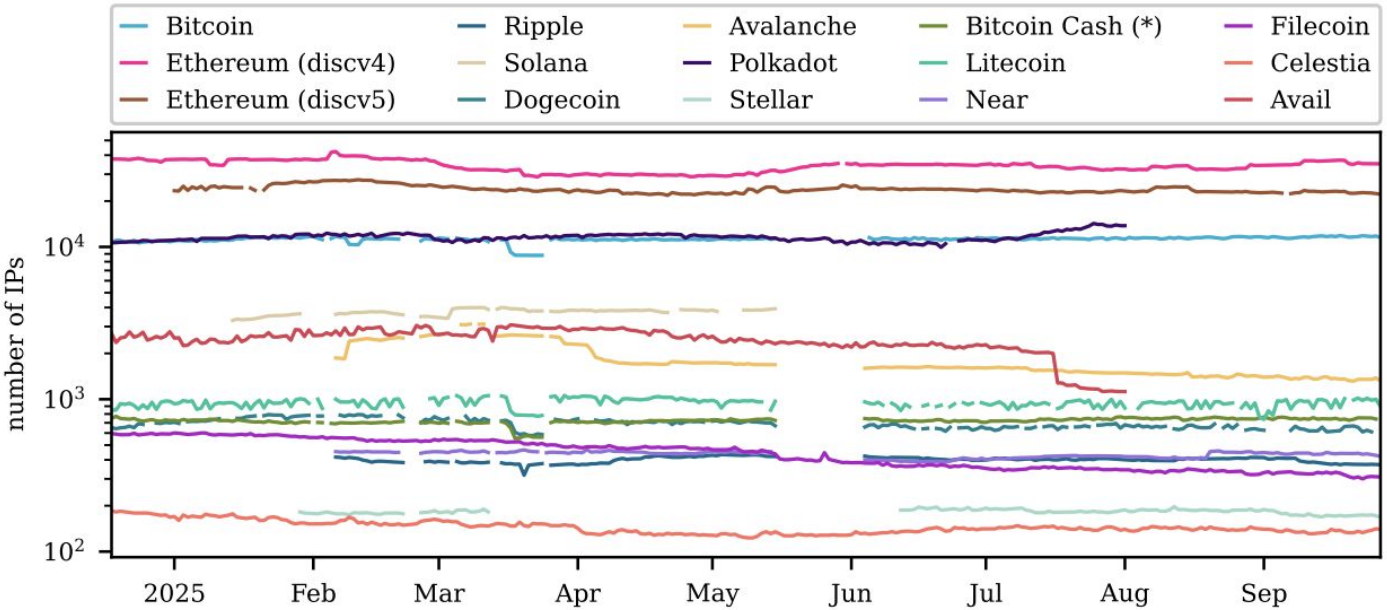
https://clientdiversity.org/#distribution

Clients




Consensus Layer
Beaconchain
Beaconstate
RPC
Endpoint
p2p layer
Engine API
Execution Layer
Ethereum
state
Transaction
Mempool
TX
TX
TX
TX
EVM
RPC
Endpoint
p2p layer

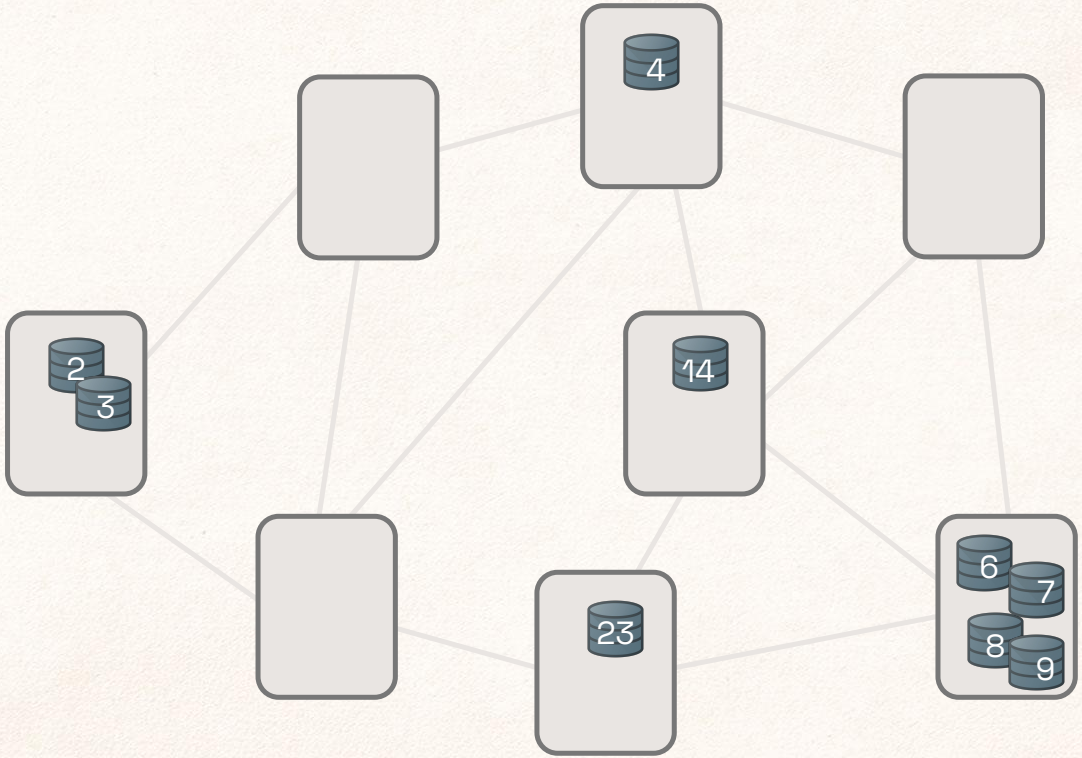
Clients



Daily count of active nodes per blockchain network.

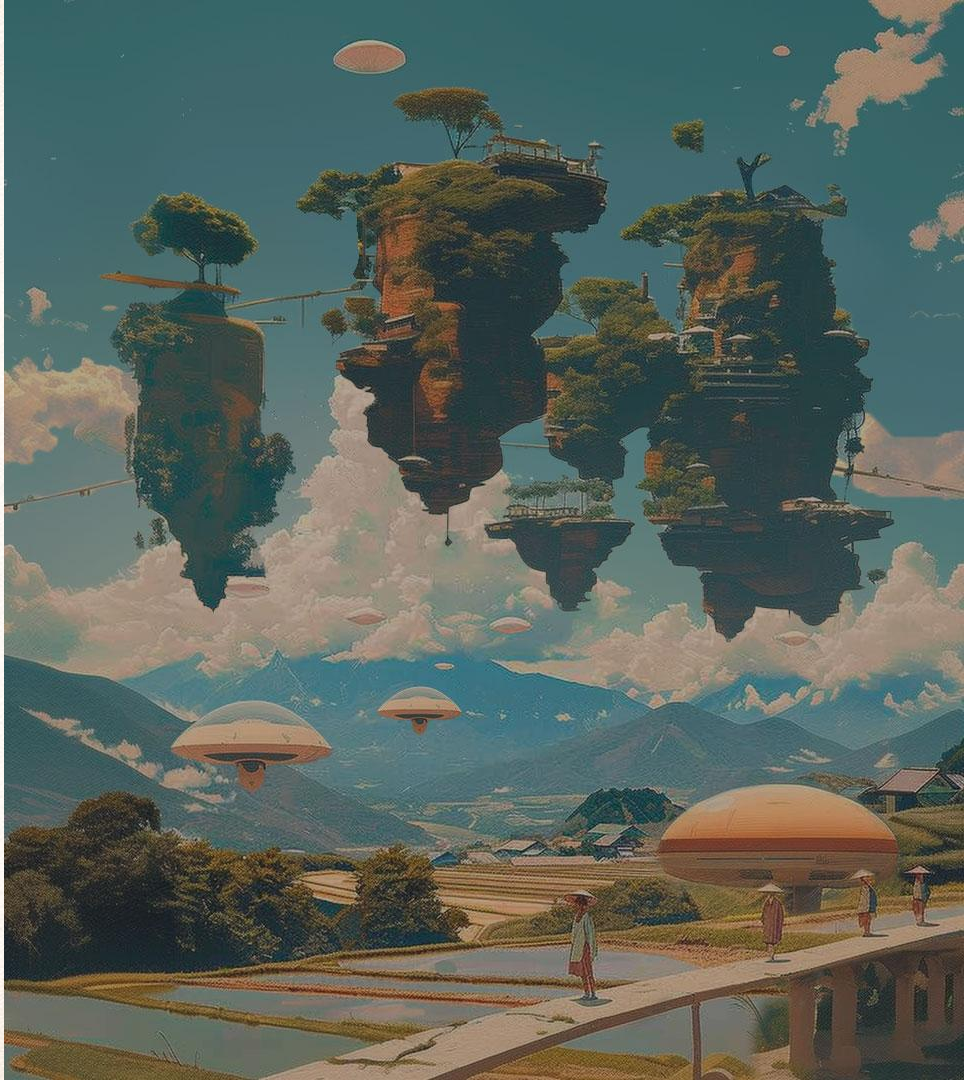
Multiple Sides of 36 Coins: Measuring Peer-to-Peer Infrastructure Across Cryptocurrencies
Lucianna Kiffer, Lioba Heimbach, Dennis Trautwein, Yann Vonlanthen, Oliver Gasser. ACM SIGMETRICS 2026, Ann Arbor, Michigan, USA, 2026

Clients



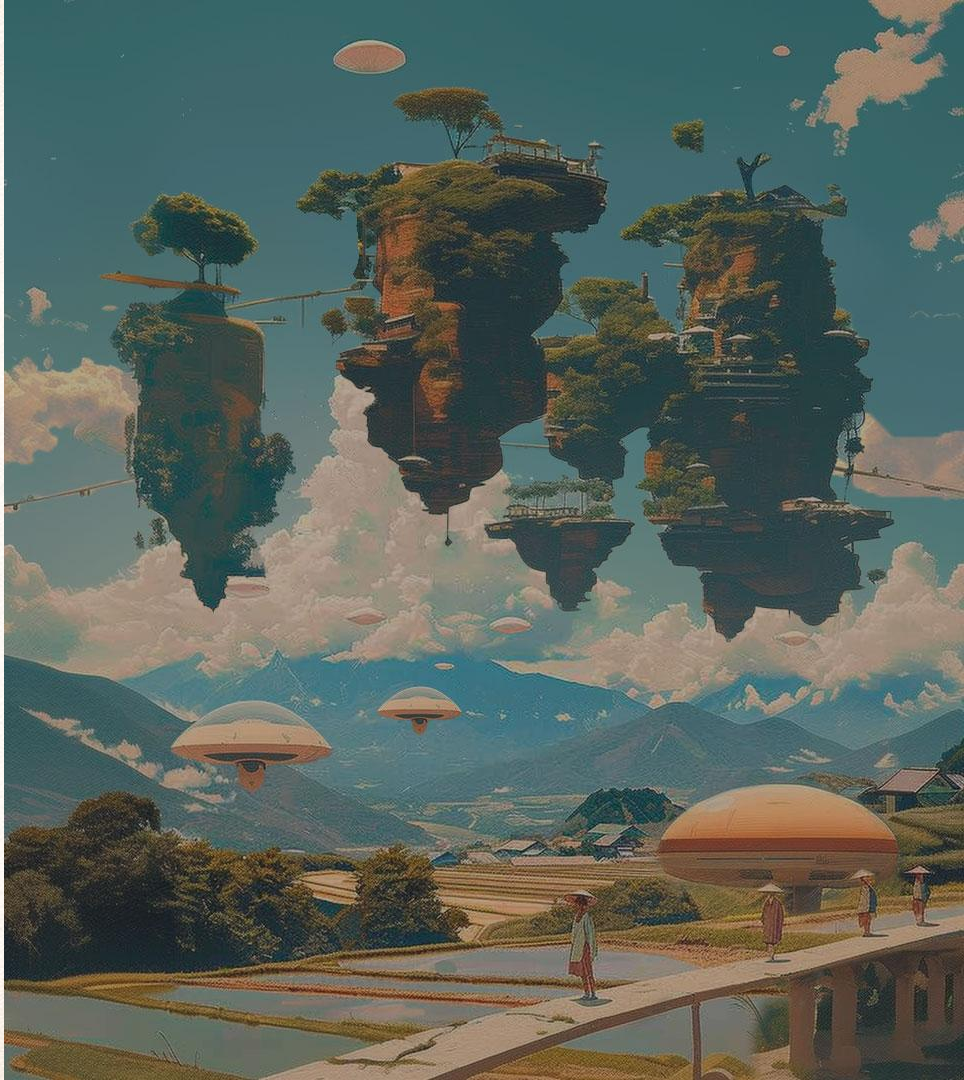
The Network Layer

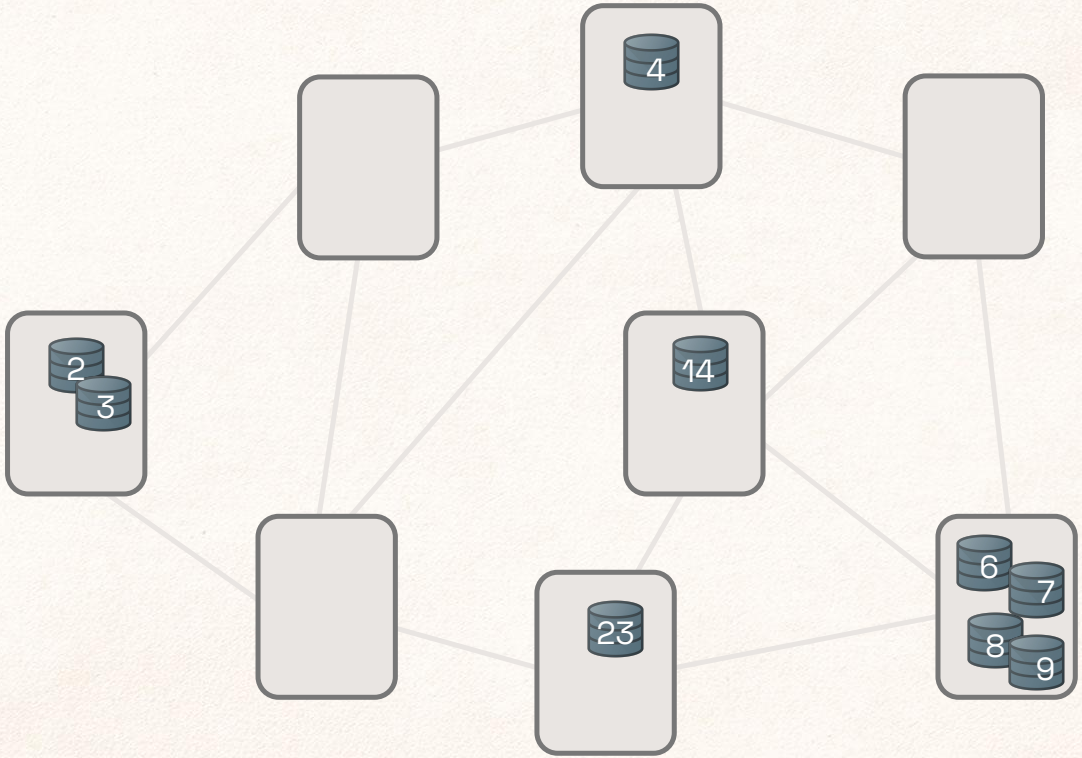
1. Censorship Resistance
   ⇒ anyone can run a client
   (follow consensus, read state, send tx)
2. Openness
   ⇒ anyone can run a validator
   ⇒ fairness (run from anywhere)
3. Privacy
   ⇒ no structured topology
4. Security
   ⇒ high resilience

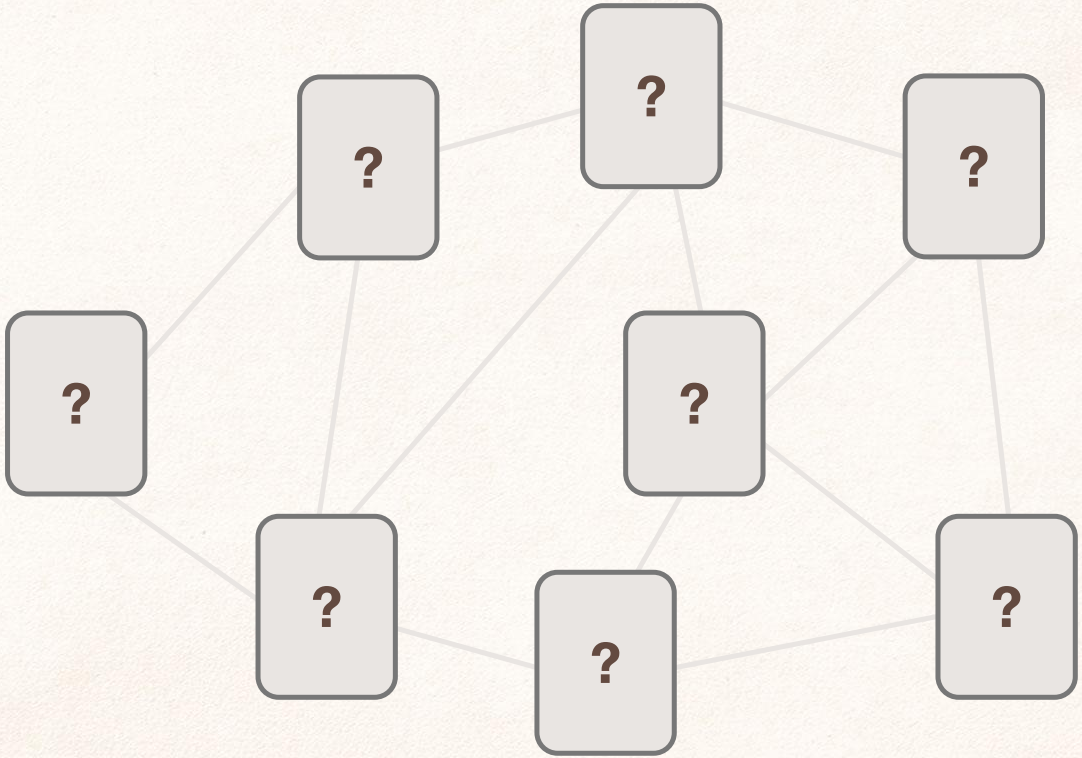


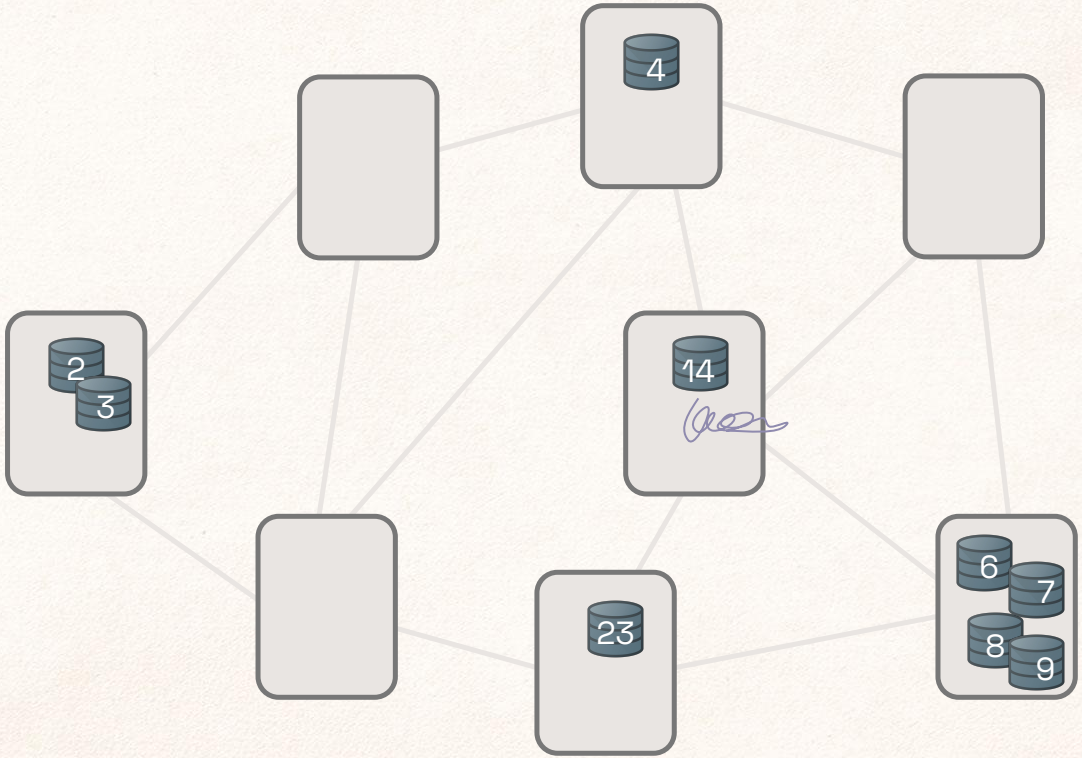
The Network Layer

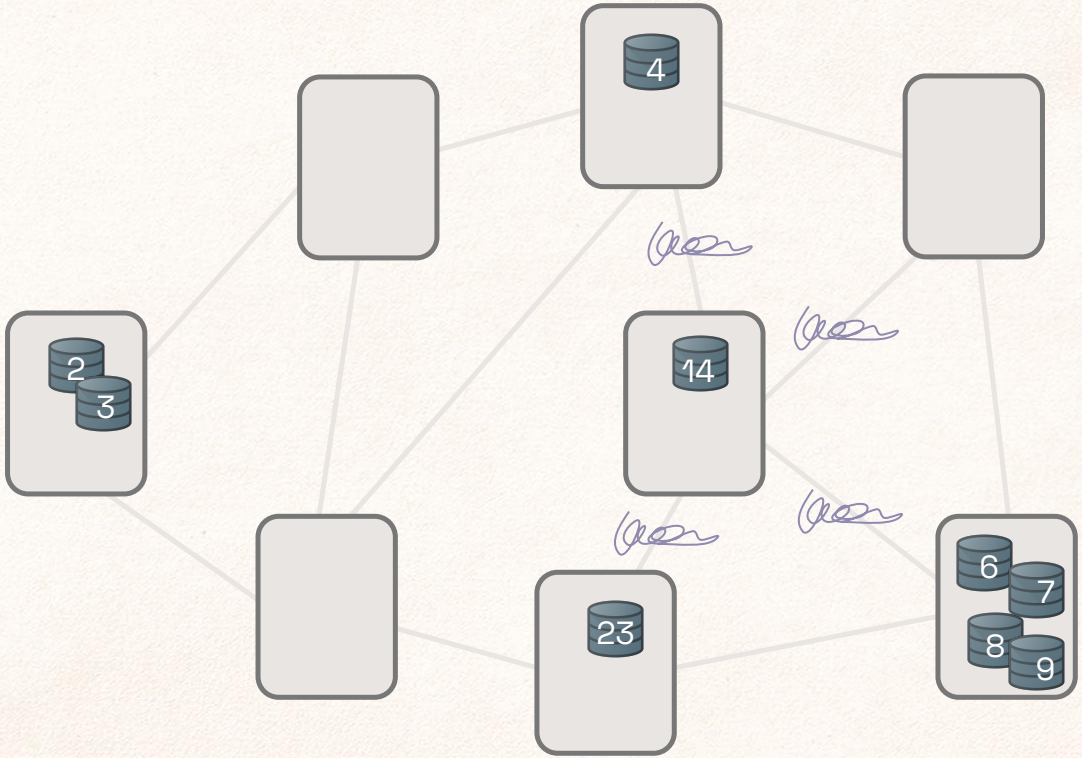
1. Censorship Resistance
   ⇒ anyone can run a client
   (follow consensus, read state, send tx)
2. Openness
   ⇒ anyone can run a validator
   ⇒ fairness (run from anywhere)
3. Privacy
   ⇒ no structured topology ⚠
4. Security
   ⇒ high resilience ⚠

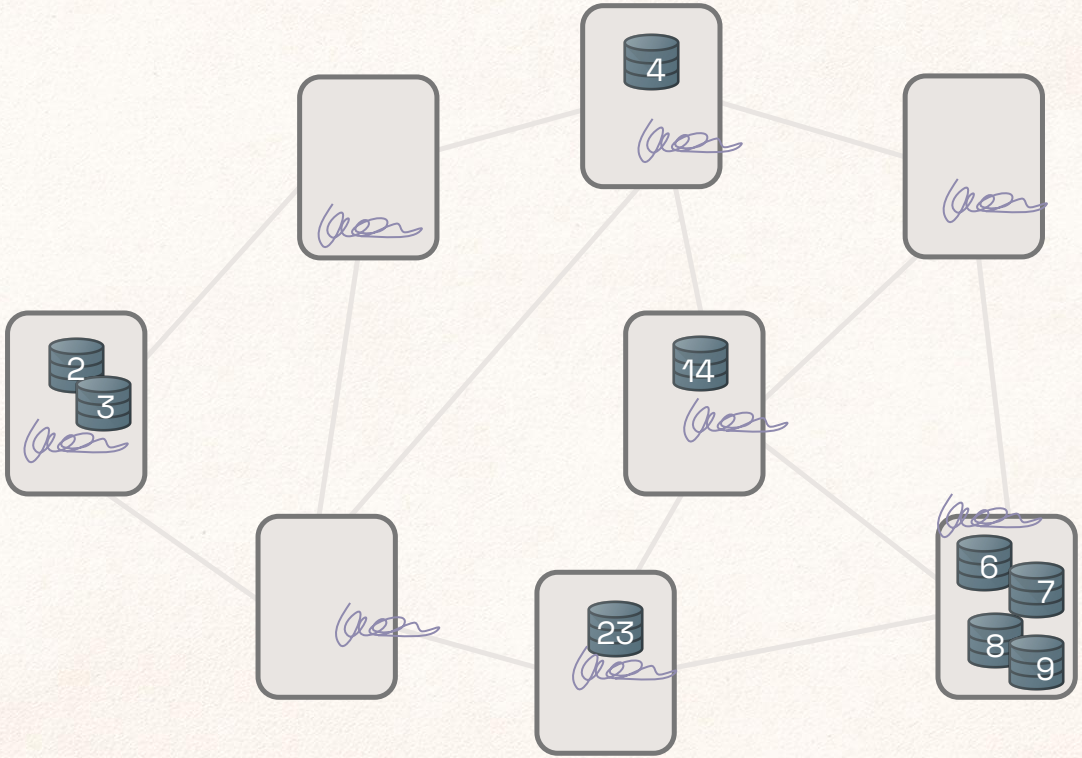


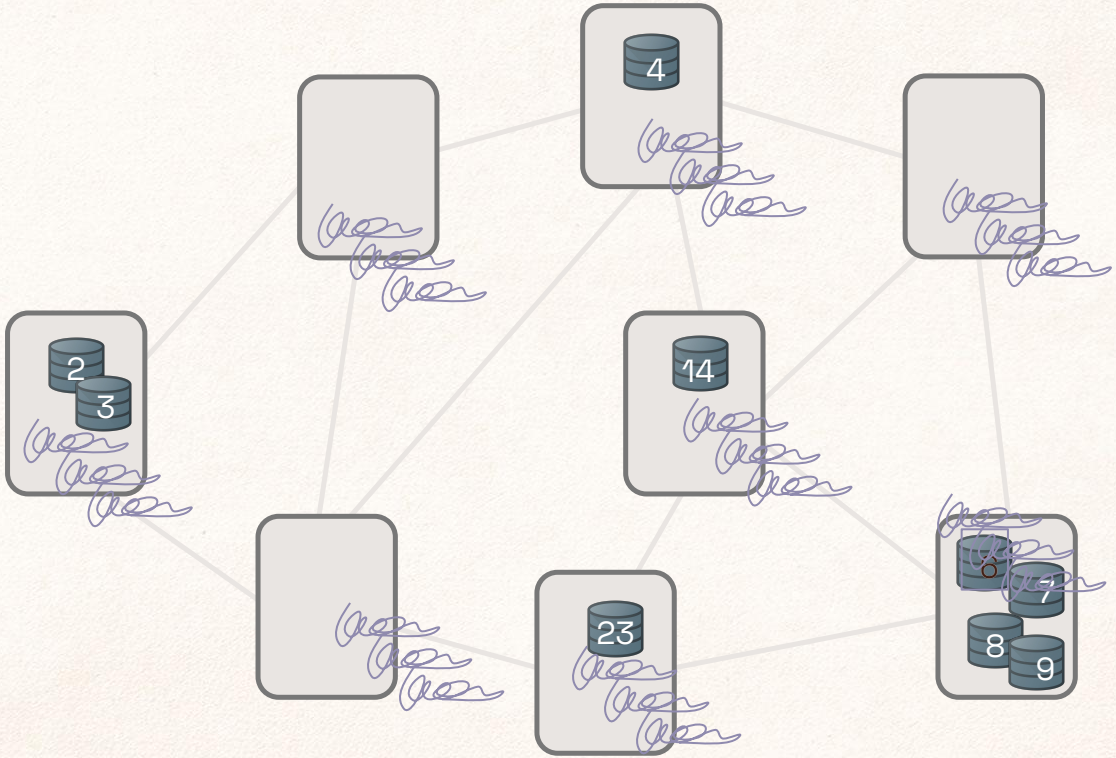

4
2
3
14
6
7
8
9
23

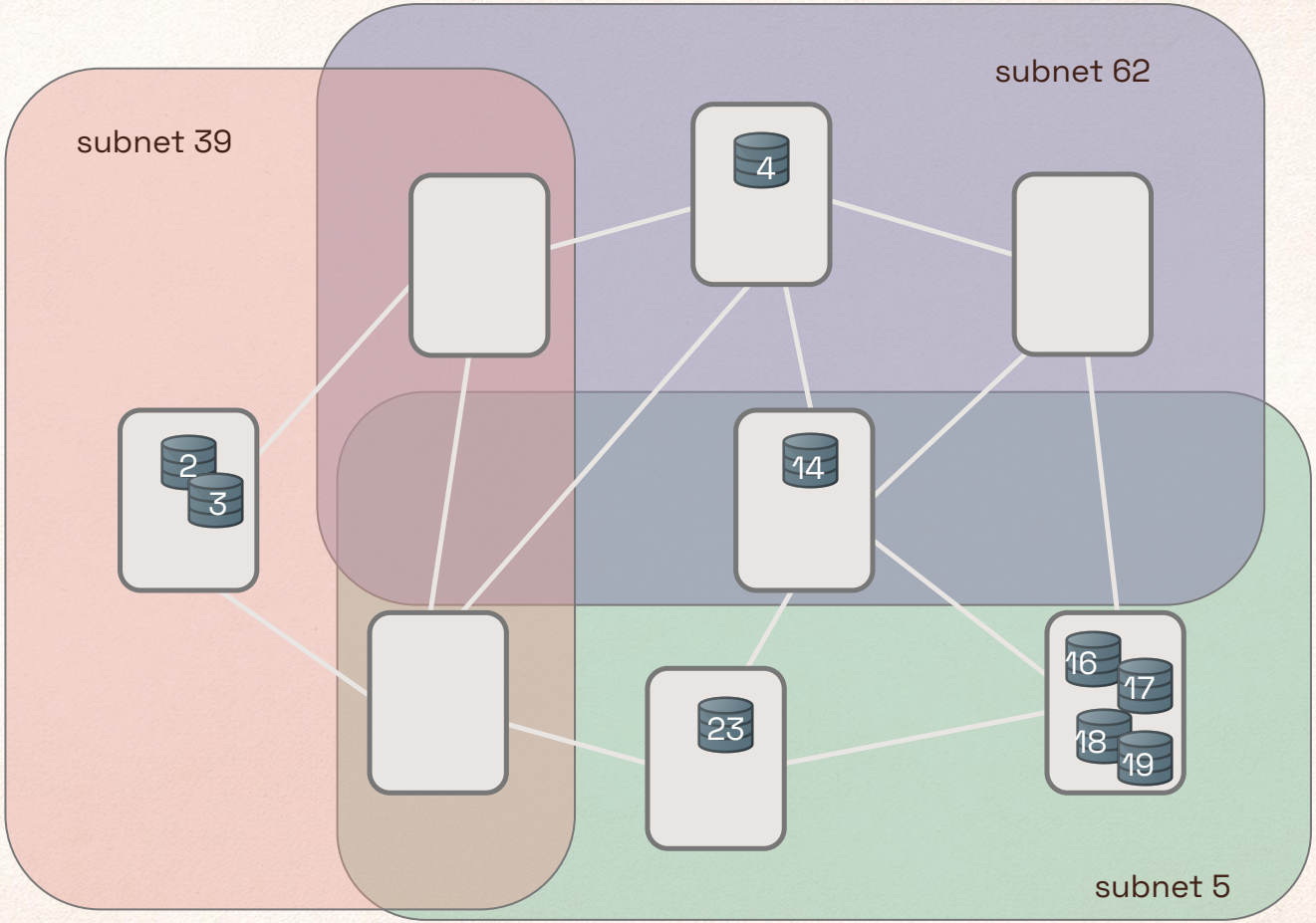


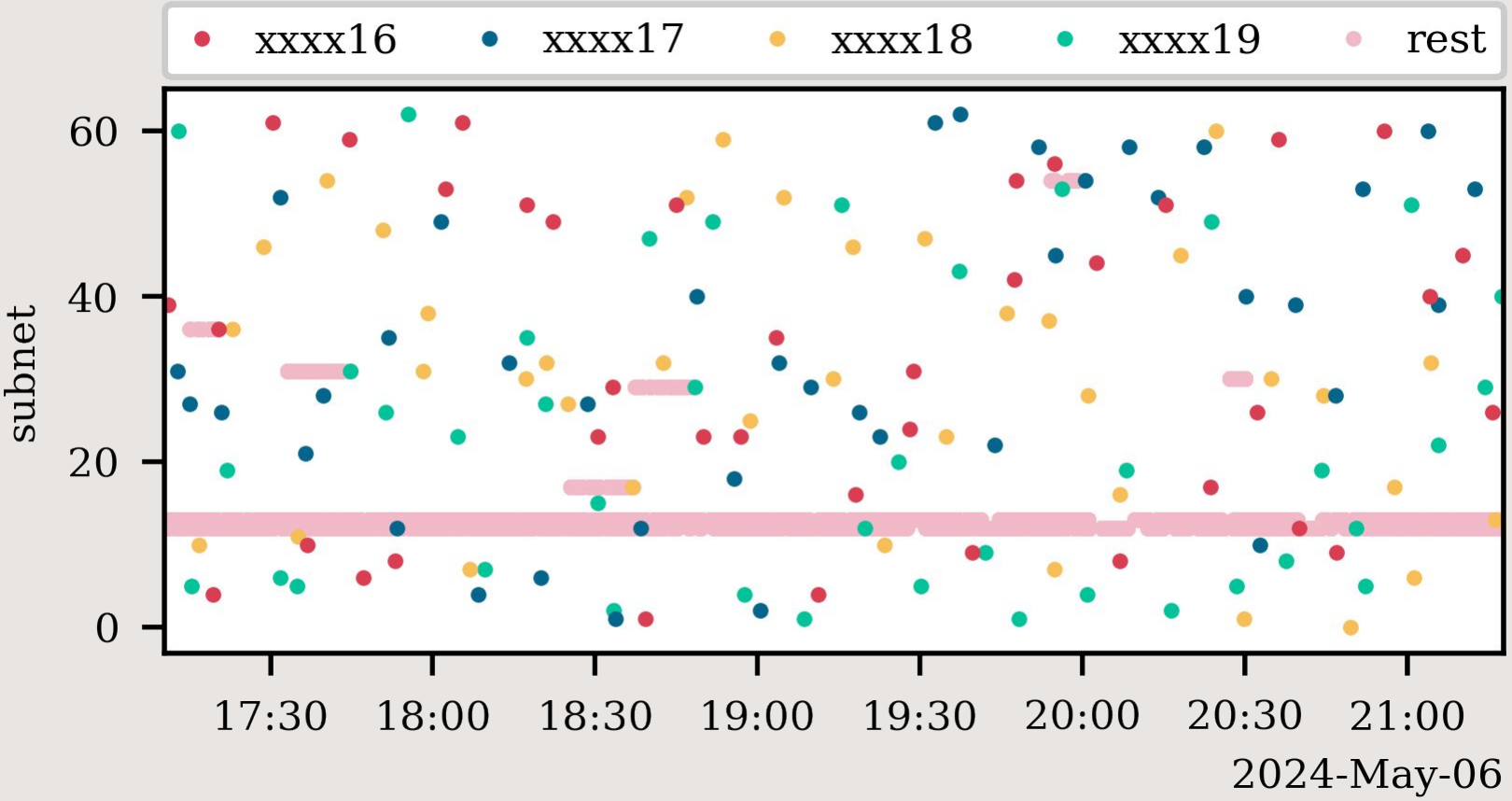

4
2
3
14
6
7
8
9
23


4
2
3
14
6
7
8
9
23


4
2
3
14
6
7
8
9
23


4
2
3
14
23
6
7
8
9


subnet 62
subnet 39
4
2
3
14
16
17
18
19
23
subnet 5



Deanonymizing Ethereum Validators: The P2P Network Has a Privacy Issue.
Lioba Heimbach, Yann Vonlanthen, Juan Villacis, Lucianna Kiffer, Roger Wattenhofer. 34th **USENIX Security** Symposium, Seattle, USA, 2025



Slot 13812415

Epoch 431637

Fulu

6.3 sec

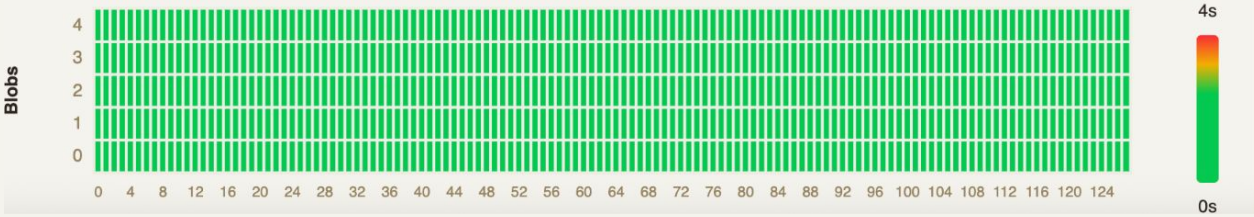
Block | Attestations | Aggregations

0s 4s 8s 12s

6.0s	Attest	3 validators
6.1s	Attest	8 validators
6.2s	Attest	11 validators
6.2s	Attest	1 validator
6.3s	Attest	1 validator
6.3s	Attest	1 validator
6.3s	Attest	2 validators
6.4s	Attest	9 validators
7.5s	Attest	1 validator
9.5s	Attest	1 validator

DATA COLUMN AVAILABILITY

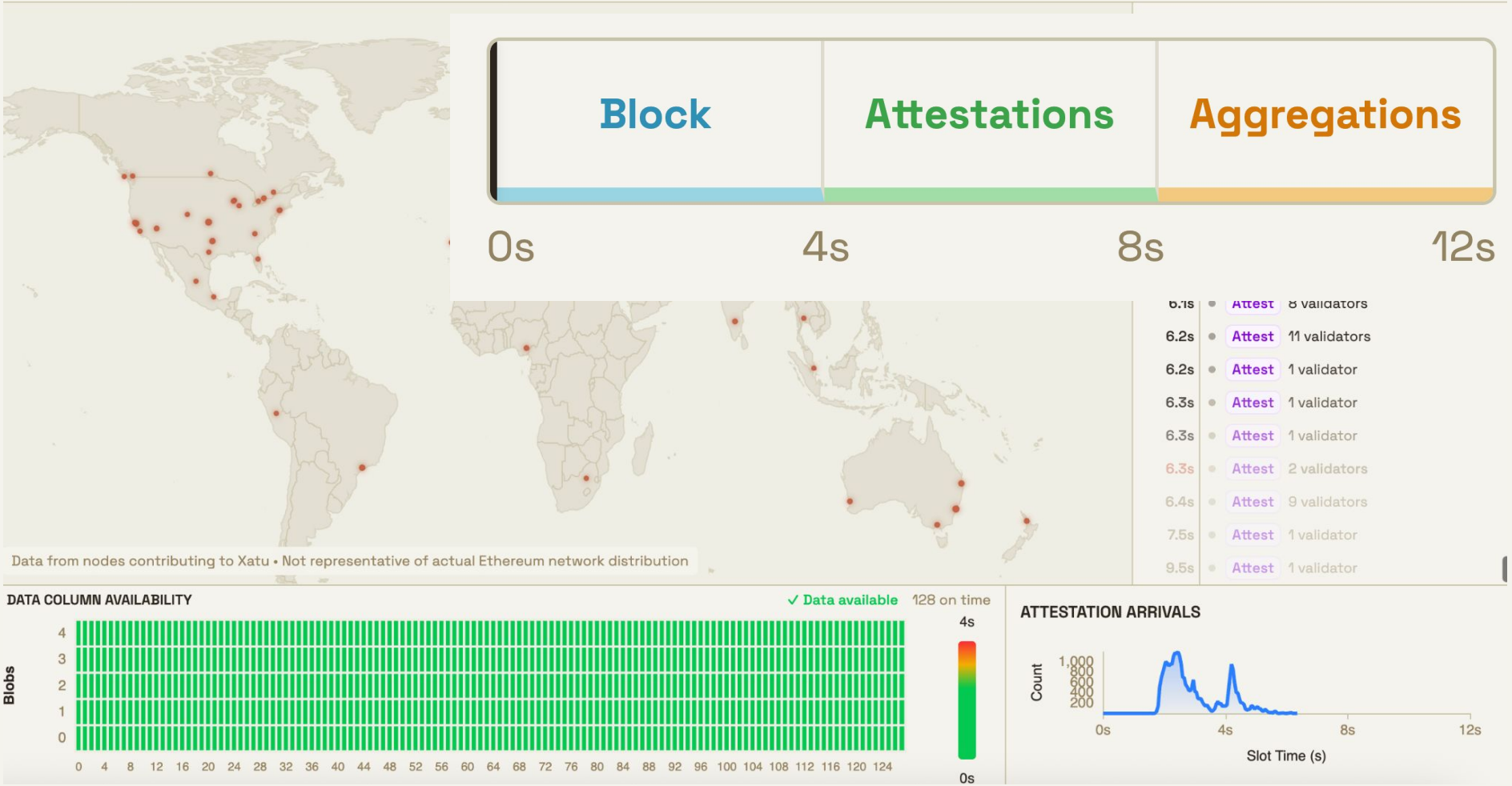
✓ Data available 128 on time



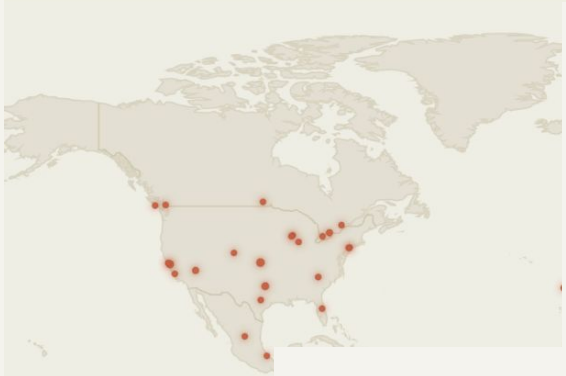
ATTESTATION ARRIVALS



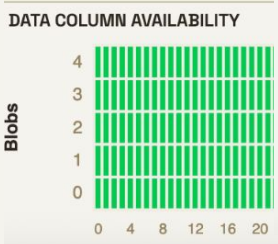
lab.ethpandaops.io

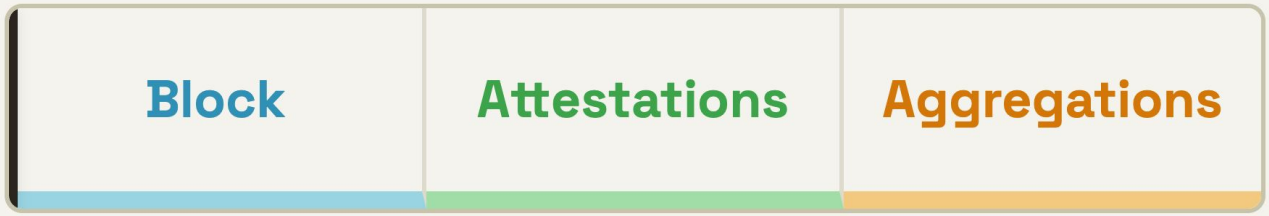


lab.ethpandaops.io



Data from nodes contributing





ATTESTATION ARRIVALS



lab.ethpandaops.io



ethp2p

Follow

Popular repositories

ethp2p — Public

ethp2p is a next-generation p2p networking stack purpose-built for Ethereum

Jupyter Notebook ☆ 36 6

netviz — Public

a very good high-perf time-traveling visualizer for networking data (WebGL2/WebGPU)

TypeScript ☆ 12 2

dasmon — Public

Live custody monitoring engine for PeerDAS [beta]

Go ☆ 8 2

afferent — Public

where Ethereum listens to itself // a collection of probes and instruments to observe the p2p layer

Go ☆ 6

notebooks — Public

Ethereum P2P observatory

Jupyter Notebook ☆ 6 5

wireshark-dissectors — Public

Wireshark Dissectors for libp2p and Ethereum consensus messages

Lua ☆ 4 2

Repositories

View as: Public

You are viewing the README and pinned repositories as a public user.

You can create a README file visible to anyone.

People



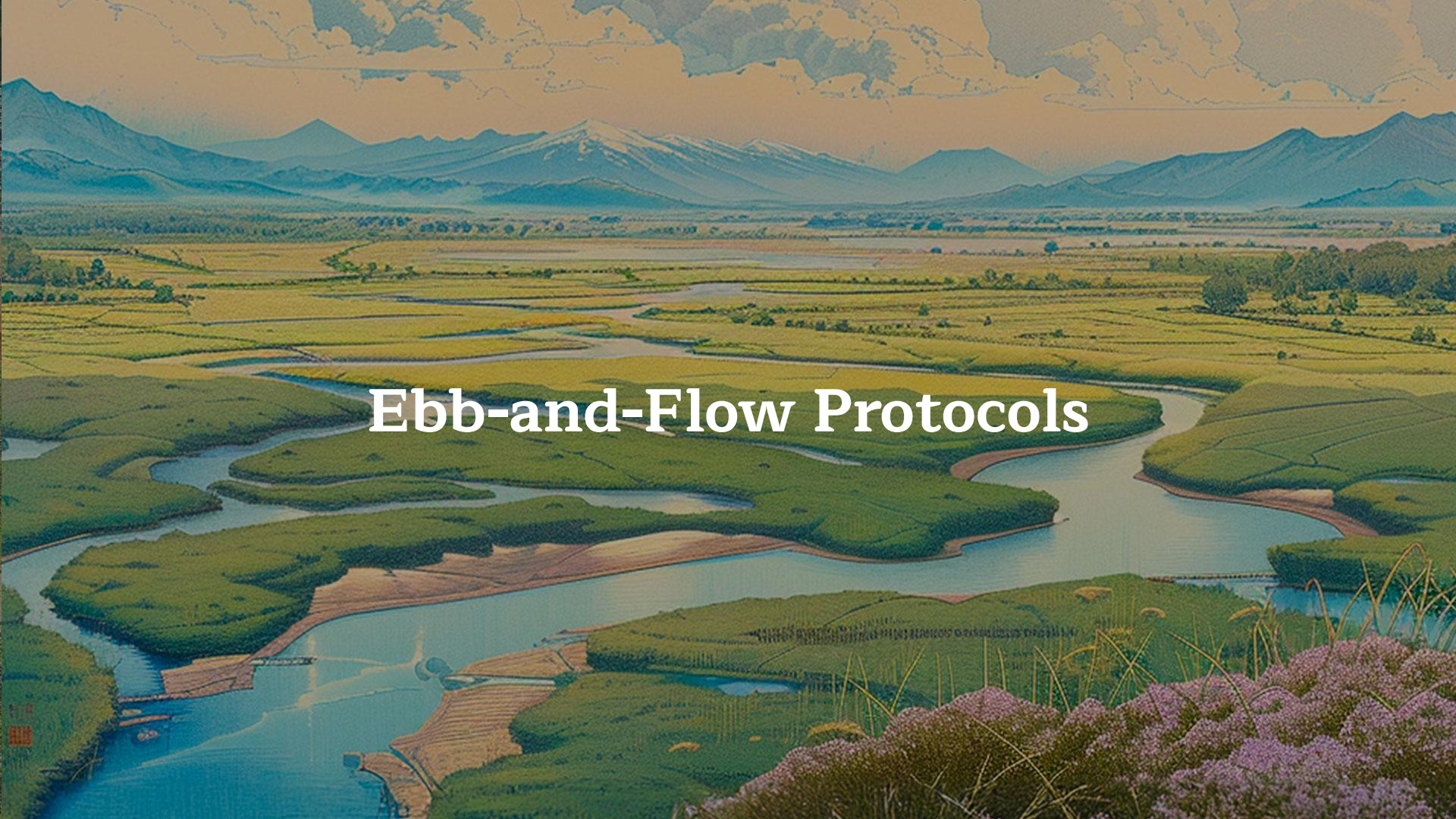
Top languages

Go · Jupyter Notebook · Python · TypeScript · Lua

Most used topics

Manage

ethereum · blockchain · data-availability · data-science · jupyter


Ebb-and-Flow Protocols



Ebb-and-Flow Protocols: A Resolution of the Availability-Finality Dilemma

Joachim Neu
jneu@stanford.edu

Ertem Nusret Tas
nusret@stanford.edu

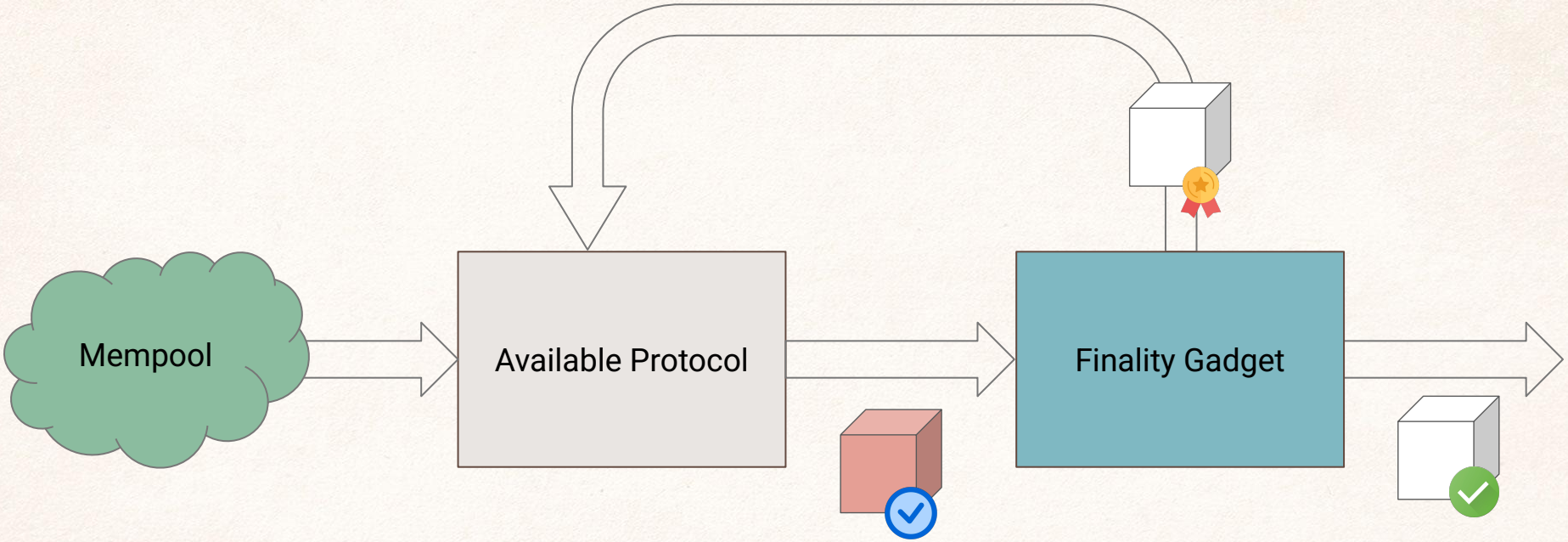
David Tse
dntse@stanford.edu

arXiv:2009.04823 [cs.CR] 4 Feb 2021

***Abstract*—The CAP theorem says that no blockchain can be live under dynamic participation and safe under temporary network partitions. To resolve this availability-finality dilemma, we formulate a new class of flexible consensus protocols, *ebb-and-flow protocols*, which support a full dynamically available ledger in conjunction with a finalized prefix ledger. The finalized ledger falls behind the full ledger when the network partitions but catches up when the network heals. Gasper, the current candidate protocol for Ethereum 2.0's beacon chain, combines the finality gadget Casper FFG with the LMD GHOST fork choice rule and aims to achieve this property. However, we discovered an attack in the standard synchronous network model, highlighting a general difficulty with existing finality-gadget-based designs. We present a construction of provably secure ebb-and-flow protocols with optimal resilience. Nodes run an off-the-shelf dynamically available protocol, take snapshots of the growing available ledger, and input them into a separate off-the-shelf BFT protocol to finalize a prefix. We explore connections with flexible BFT and improve upon the state-of-the-art for that problem.**

need to assume all adversary nodes are awake at the beginning [3], [6] or a trusted setup for nodes to join the network [4], [5], but recently it has been shown that these restrictions can be removed using verifiable delay functions [7].

One limitation of dynamically available protocols is that they are not tolerant to network partition: when the network partitions, honest nodes in a dynamically available protocol will think that many nodes are asleep, continue to confirm transactions, and thus is not safe.[1] This is in contrast to permissioned BFT protocols designed for partially synchronous networks, such as PBFT [8], Tendermint [9], [10], Hotstuff [11] and Streamlet [12]. This type of protocols is the basis for permissioned blockchains such as Libra [13], [14] and PoS blockchains such as Algorand [15], [16]. In these protocols, a quorum of two-thirds of the signatures of all the nodes is required to finalize transactions, and hence is safe under



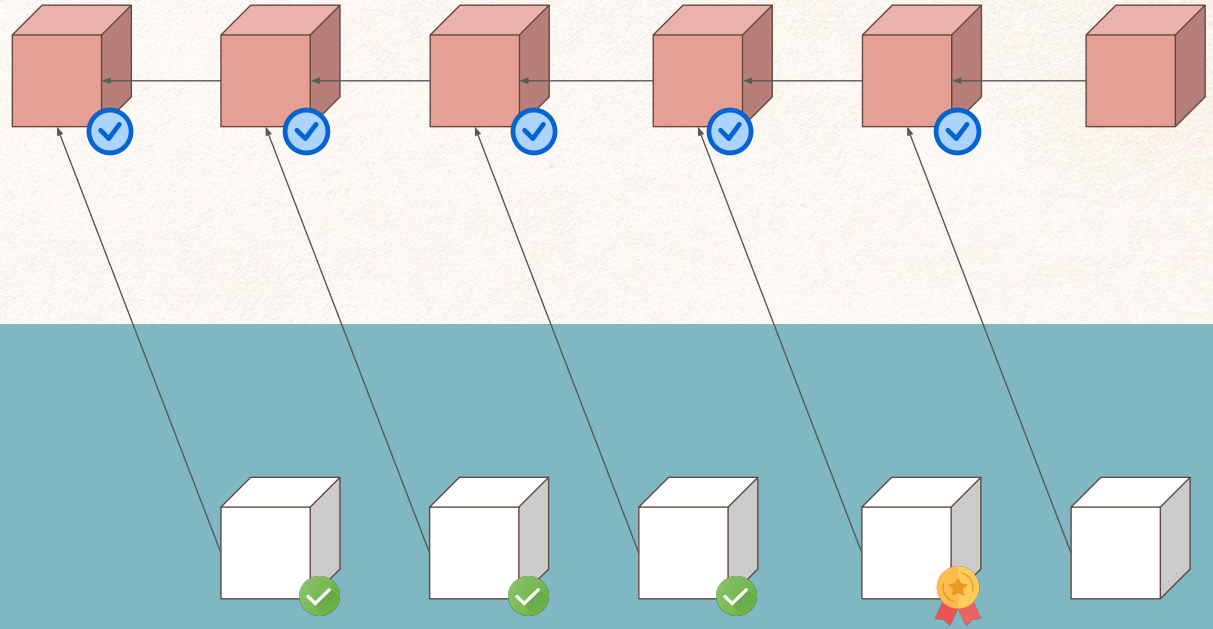
Prefix Property: The available chain is a prefix of the final chain.

Available Chain

f < 50% of awake

Δ (Synchronous)

1 honest awake



Finality Chain

f < n / 3

Δ after **GST** (part. sync)

All online after **GAT**

Advantages of Ebb & Flow

- Circumventing the CAP Theorem
  - Two chains ⇒ two signals
  - Peace time / war time
- Available Chain
  - **Continuous** block production
  - Higher **adversary resilience** under normal network conditions
  - PQ ready?
- Finality Chain
  - **Economic security**
- Circumventing the “slot-length vs time to finality” trade-off



Why Ethereum Needs a Dynamically Available Protocol

Consensus



luca_zanolini 2 12d

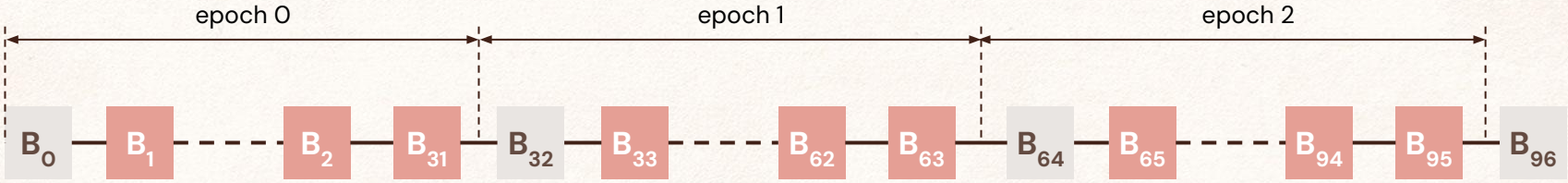
Author: Luca Zanolini 9

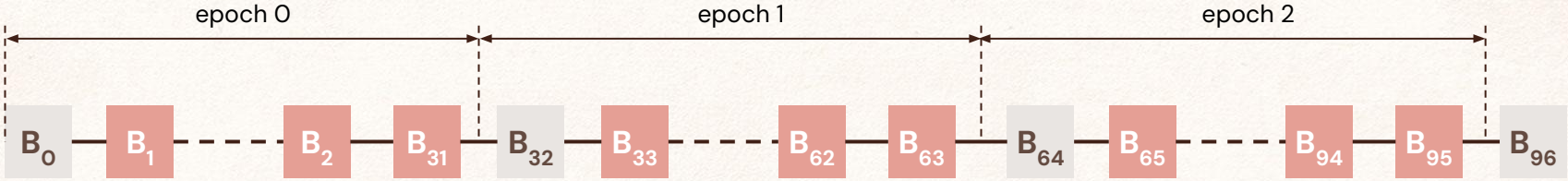
Huge thanks to Ben 9 *, Francesco* 3 *, Joachim* 4 *, Justin* 6 *, Mikhail* 3 *, Roberto* 1 *, Thomas* 3 *, Vitalik* 1 *, and Yann* 4 *for their feedback.*

We are working on a proposal for the next consensus protocol for Ethereum. A central piece of the new design is a two-layer architecture: a fast available chain — the *heartbeat* — produced by a small randomly-sampled committee, and a separate finality mechanism that trails behind, finalizing blocks the heartbeat has already produced — crucially, with the two layers fully **decoupled**, unlike the current Gasper 2 design where LMD-GHOST and Casper FFG interact in ways that have proven difficult to reason about 6 . Vitalik outlined this direction in a recent post 11 .

This post focuses on the first layer — the heartbeat — and on a property we believe should be a strict requirement for it: **dynamic availability**.

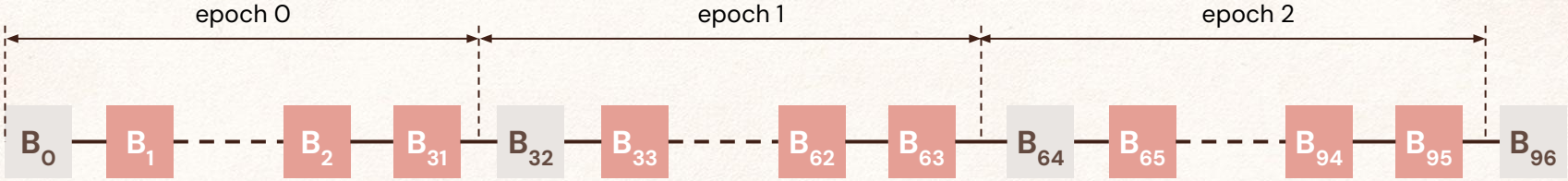

LMD-GHOST & Casper FFG


epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96


epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96

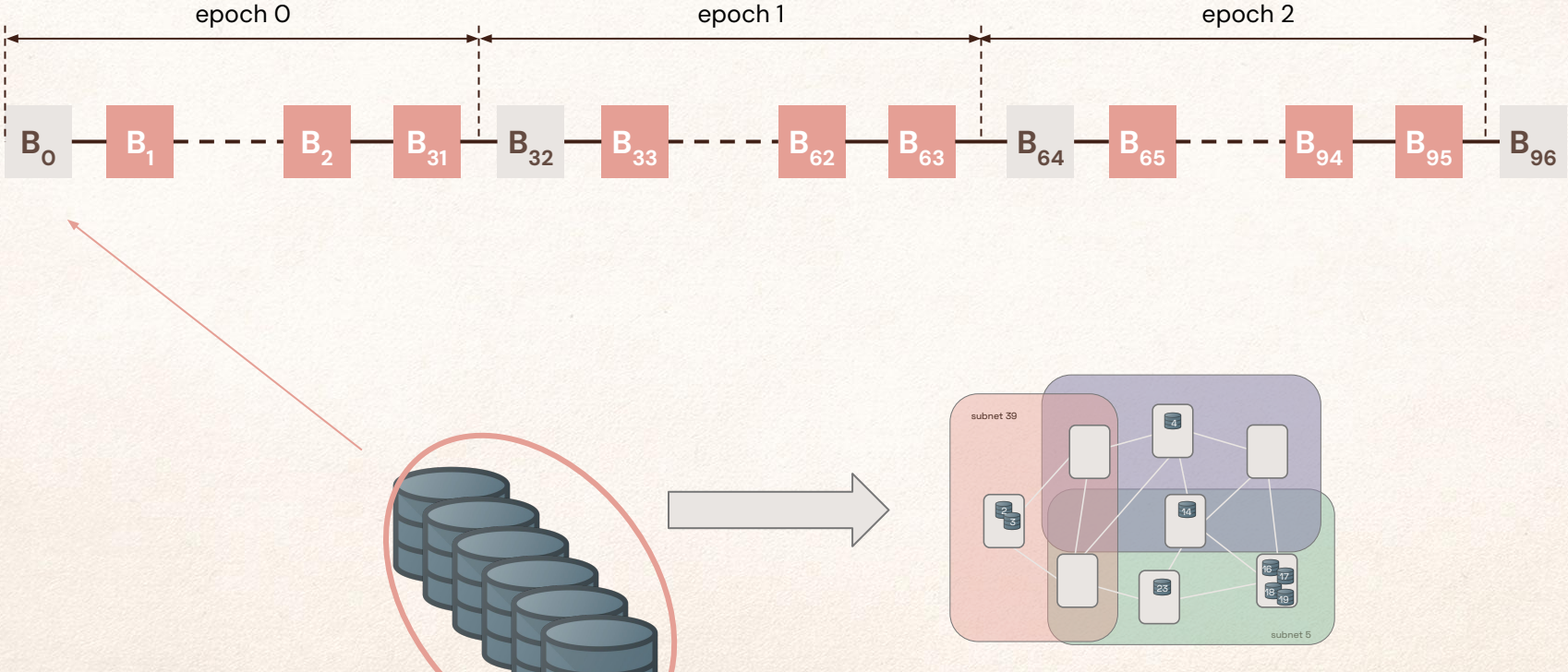


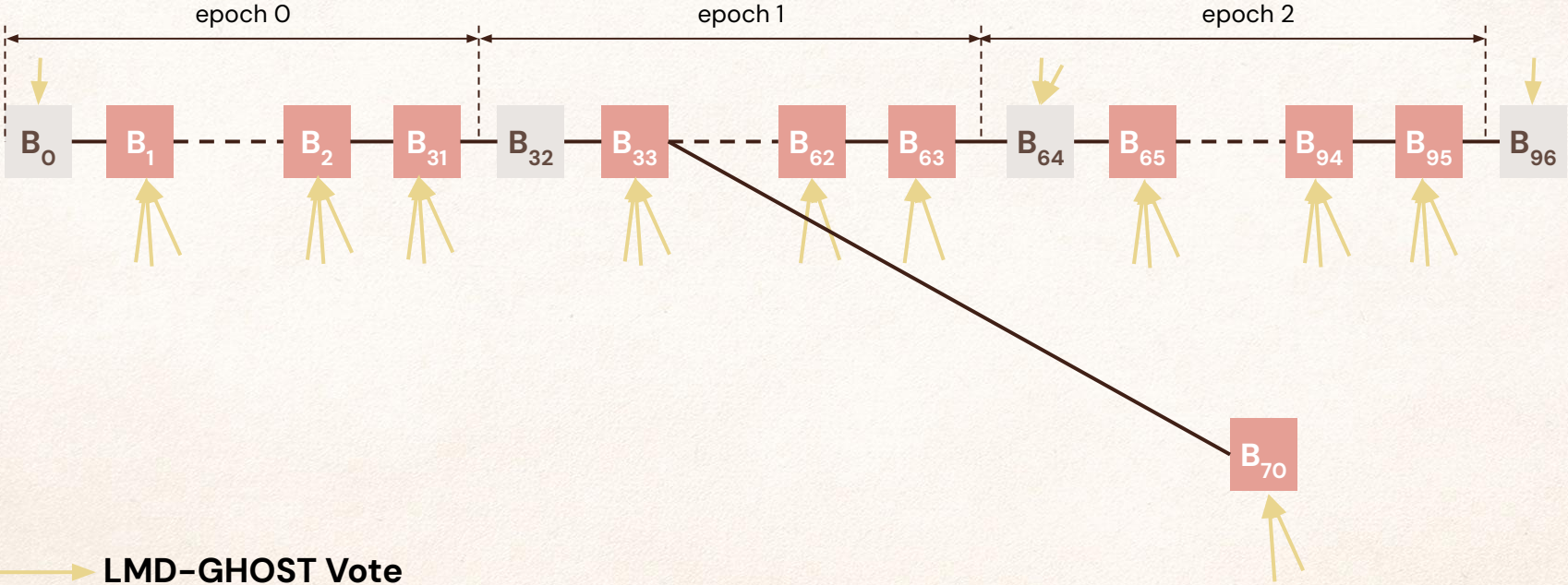
subnet 5


epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96



subnet 5


epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96
subnet 39
subnet 5



LMD-GHOST Vote

Vote on blocks

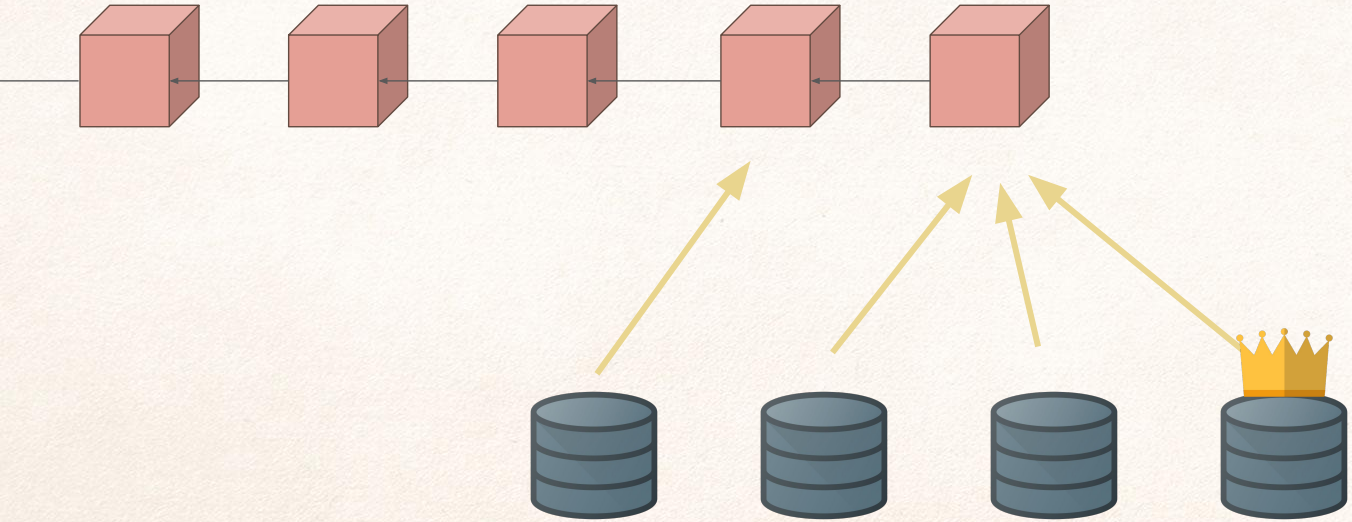
LMD-GHOST = Latest Message Driven Greediest Heaviest Observed SubTree

LMD-GHOST

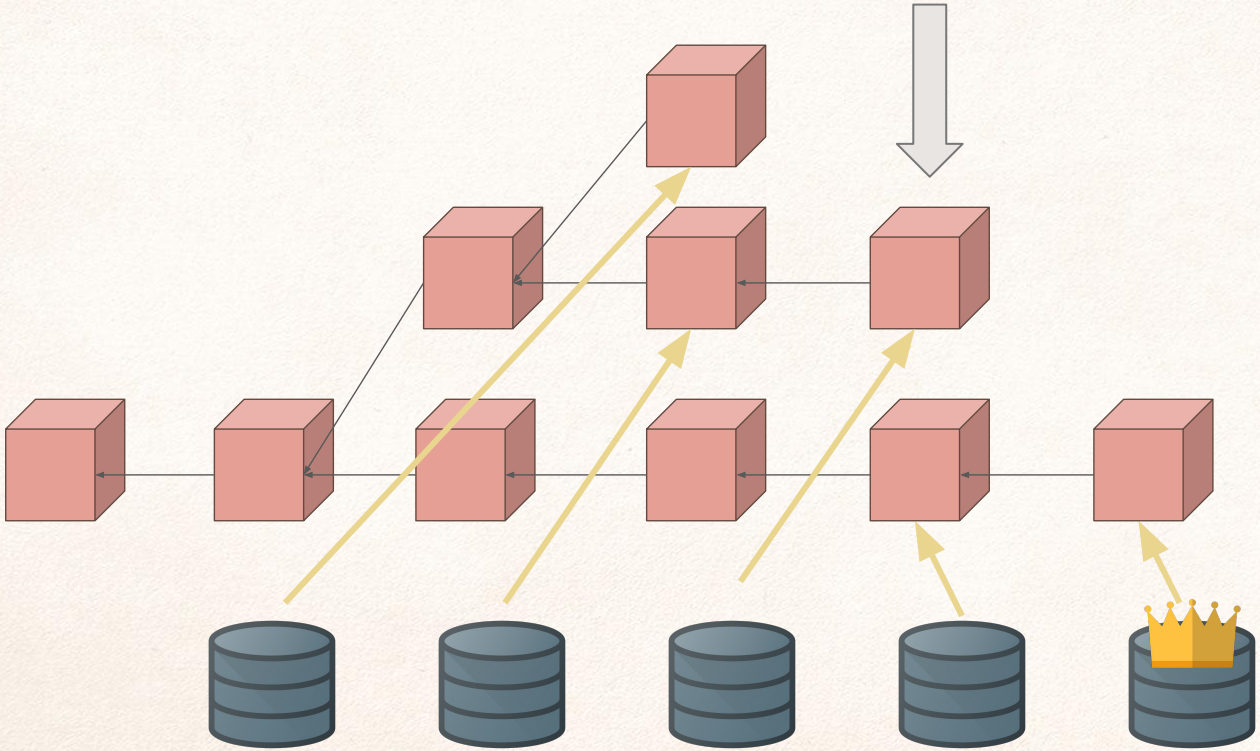
f < 50% of awake

Δ (Synchronous)

1 honest awake



LMD-GHOST

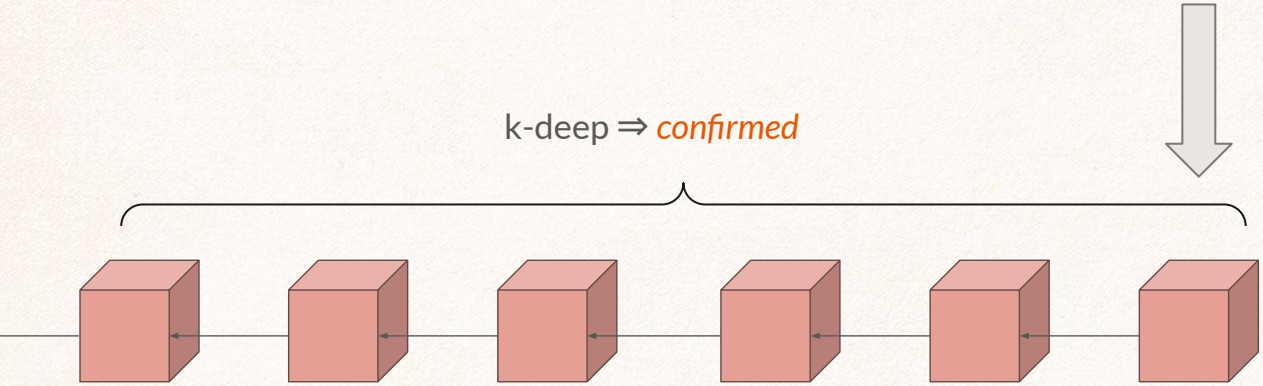


f < 50% of awake

Δ (Synchronous)

1 honest awake

LMD-GHOST





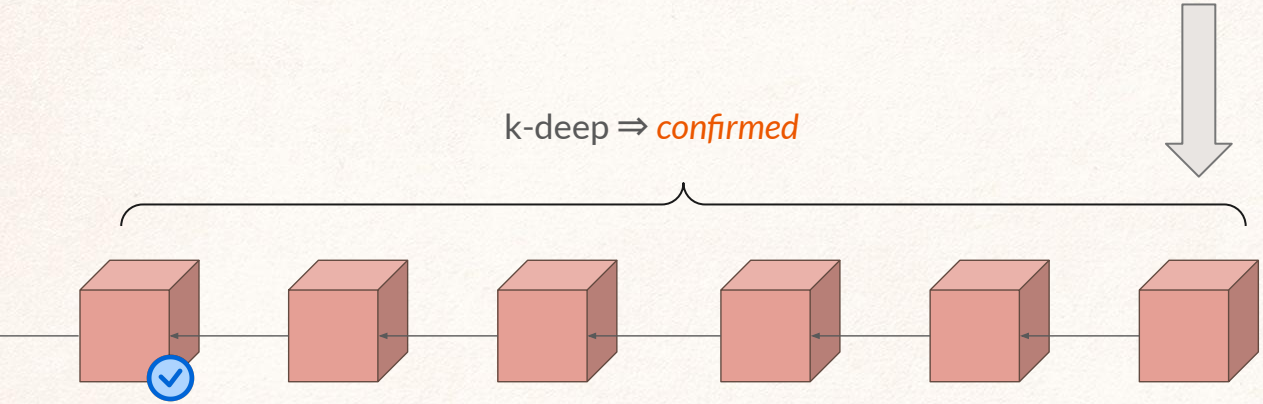
f < 50% of awake

Δ (Synchronous)

1 honest awake

k-deep confirmation

LMD-GHOST





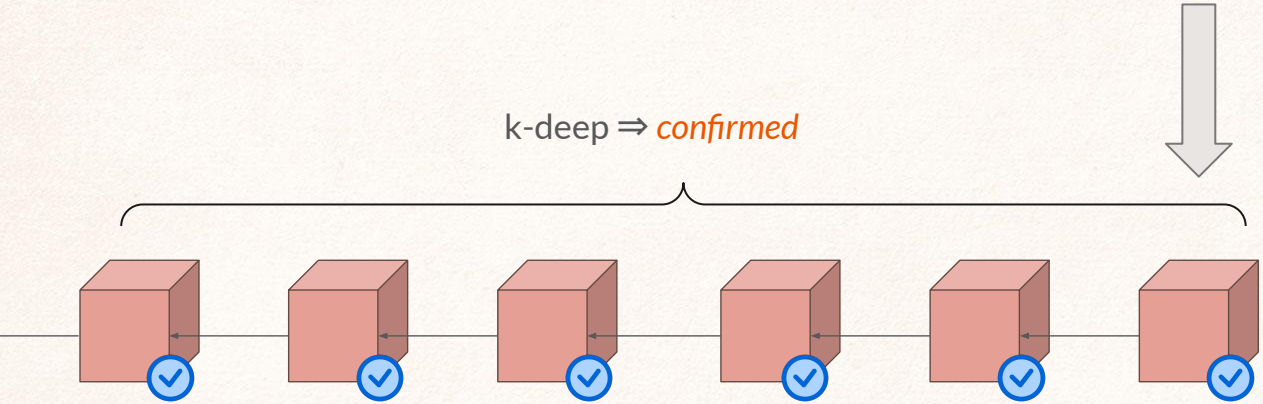
f < 50% of awake

Δ (Synchronous)

1 honest awake

k-deep confirmation

LMD-GHOST







f < 50% of awake



Δ (Synchronous)



1 honest awake

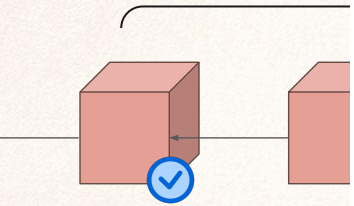


k-deep confirmation

OR

fast confirmation

LMD-GHOST



f < 50% of awake

Δ (Synchronous)

1 honest awake

k-deep confirmation

OR

fast confirmation

A Fast Confirmation Rule (aka Fast Synchronous Finality) for the Ethereum Consensus Protocol

Aditya Asgaonkar
Offchain Labs*

Francesco D'Amato
Ethereum Foundation

Roberto Saltini
Ethereum Foundation†

Luca Zanolini
Ethereum Foundation

Chenyi Zhang
University of Canterbury†

Abstract

A Confirmation Rule, within blockchain networks, refers to an algorithm implemented by network nodes that determines (either probabilistically or deterministically) the permanence of certain blocks on the blockchain. An example of Confirmation Rule is the Bitcoin's *longest chain* Confirmation Rule where a block b is confirmed (with high probability) when it has a sufficiently long chain of successors, its siblings have notably shorter successor chains, the majority of the network's total computation power (hashing) is controlled by honest nodes, and network synchrony holds.

The only Confirmation Rule currently available in the Ethereum protocol, Gasper, is the FFG Finalization Rule. While this Confirmation Rule works under asynchronous network conditions, it is quite slow for many use cases. Specifically, best-case scenario, it takes around 13 to 19 min to confirm a transaction, where the actual figure depends on when the transaction is submitted to the network.

In this work, we devise a Fast Confirmation Rule for Ethereum's consensus protocol. Our Confirmation Rule relies on synchrony conditions, but provides a best-case confirmation time of 12 seconds only, greatly improving on the latency of the FFG Finalization Rule.

Users can then rely on the Confirmation Rule that best suits their needs depending on their belief about the network conditions and the need for a quick response.

Confirmation distance over time

Daily roll-up. Gaps are days inside a known xatu pipeline outage window (those days are dropped from the underlying data).

Metric | P50 delay | P95 delay | P99 delay | 1-slot FCR rate

Lighthouse · Lodestar · Grandine · Teku



Gaps in the lines are days that fell inside a xatu pipeline outage window.

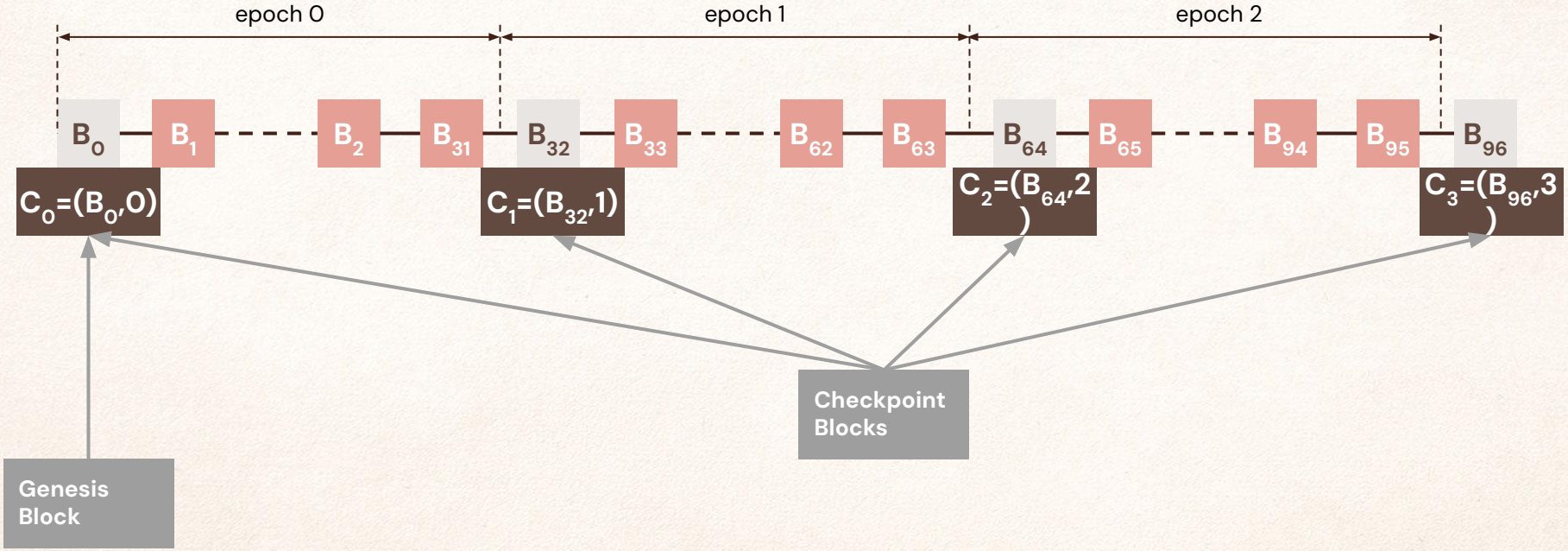


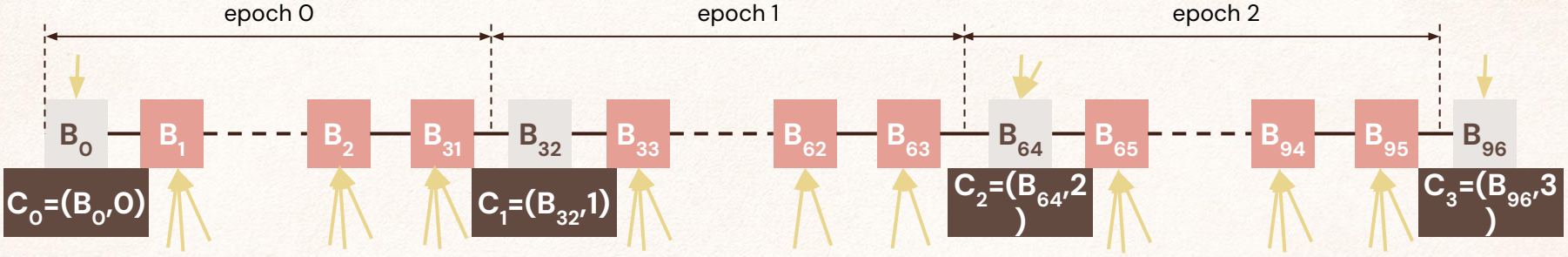
f < 80% of awake



Δ (Synchronous)

ethpandaops.io/posts/fcr-simulator/

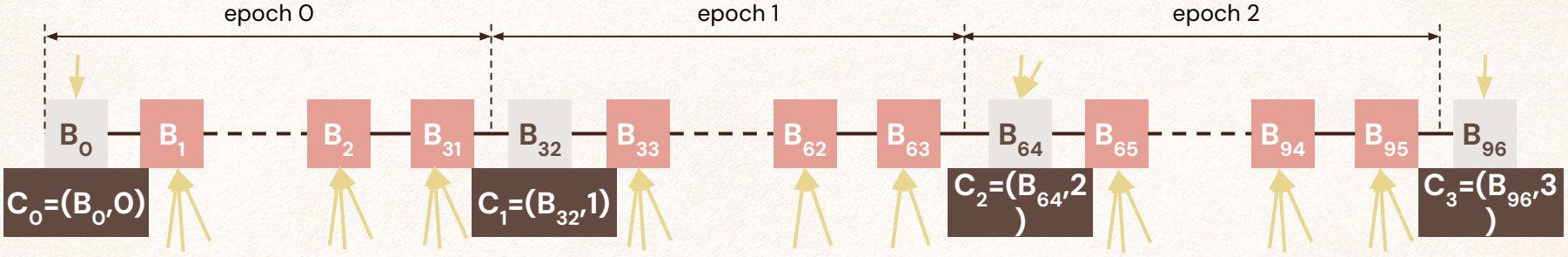

epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96
C0=(B0,0)
C1=(B32,1)
C2=(B64,2)
C3=(B96,3)
Checkpoint Blocks
Genesis Block



LMD-GHOST Vote

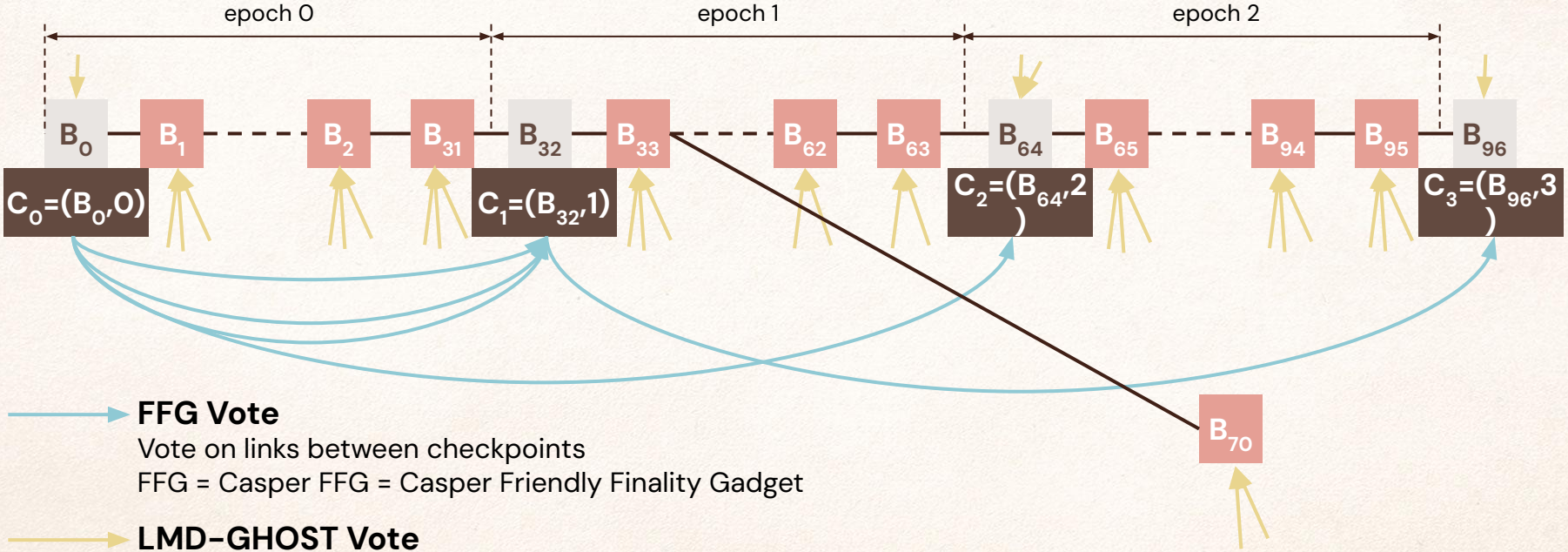
Vote on blocks

LMD-GHOST = Latest Message Driven Greediest Heaviest Observed SubTree



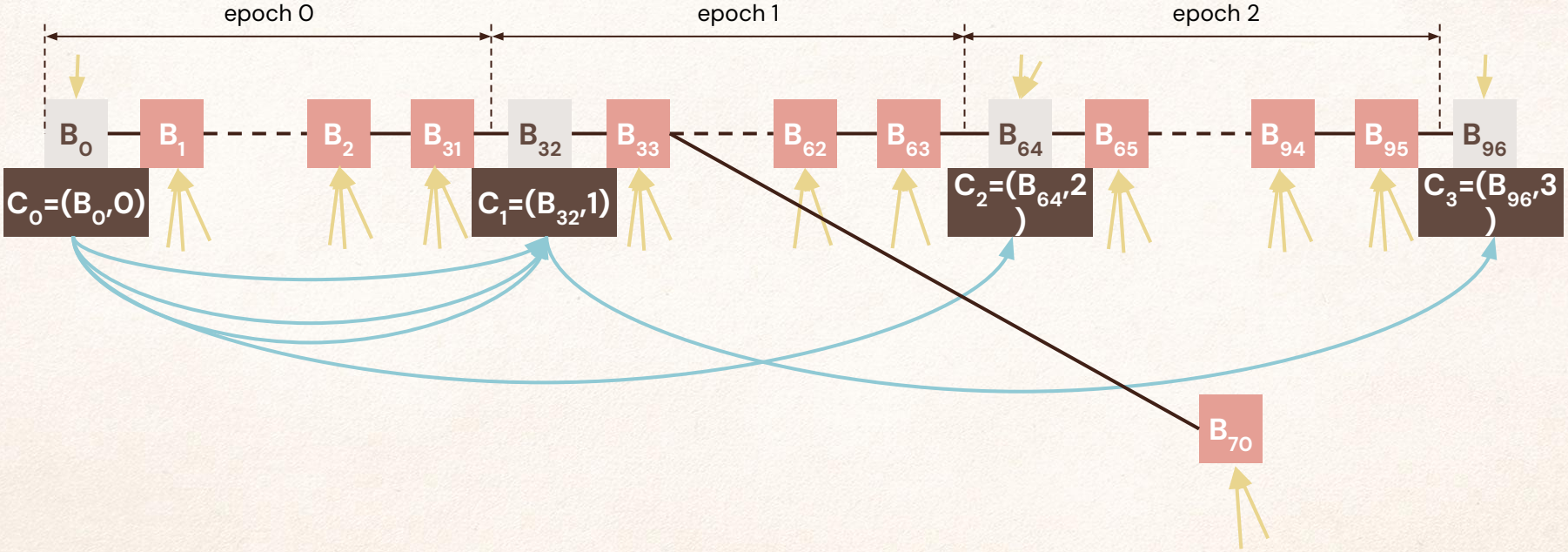
FFG Vote
Vote on links between checkpoints
FFG = Casper FFG = Casper Friendly Finality Gadget

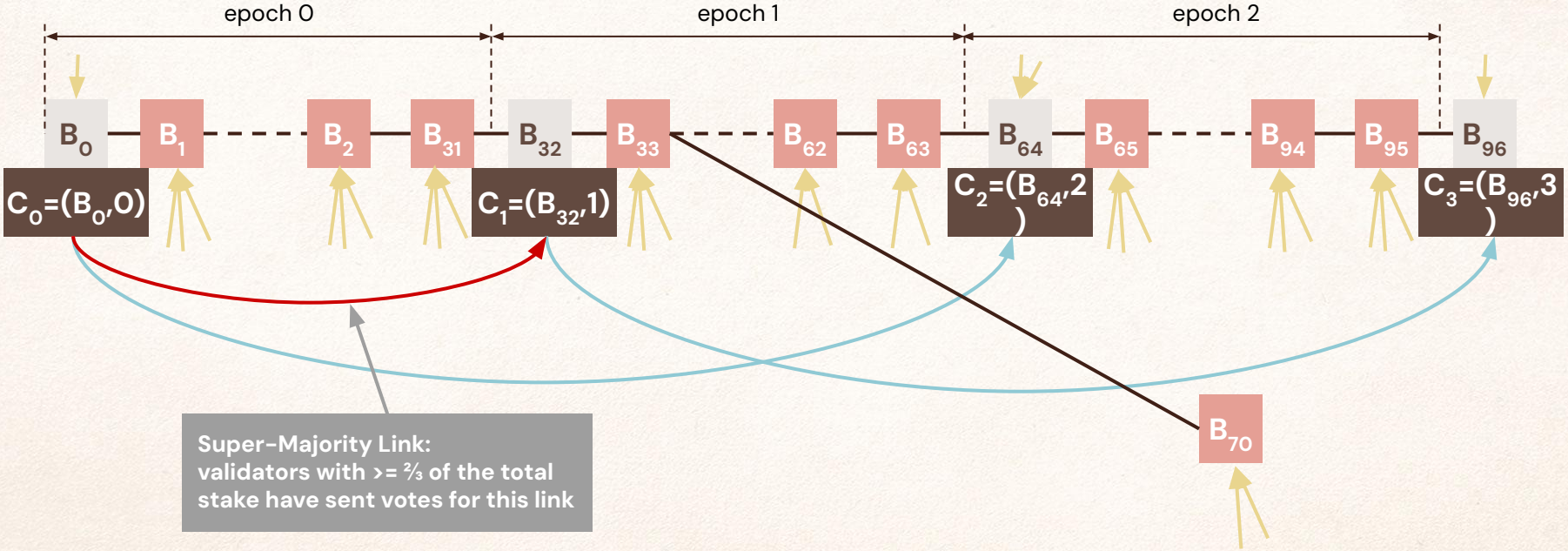
LMD-GHOST Vote
Vote on blocks
LMD-GHOST = Latest Message Driven Greediest Heaviest Observed SubTree

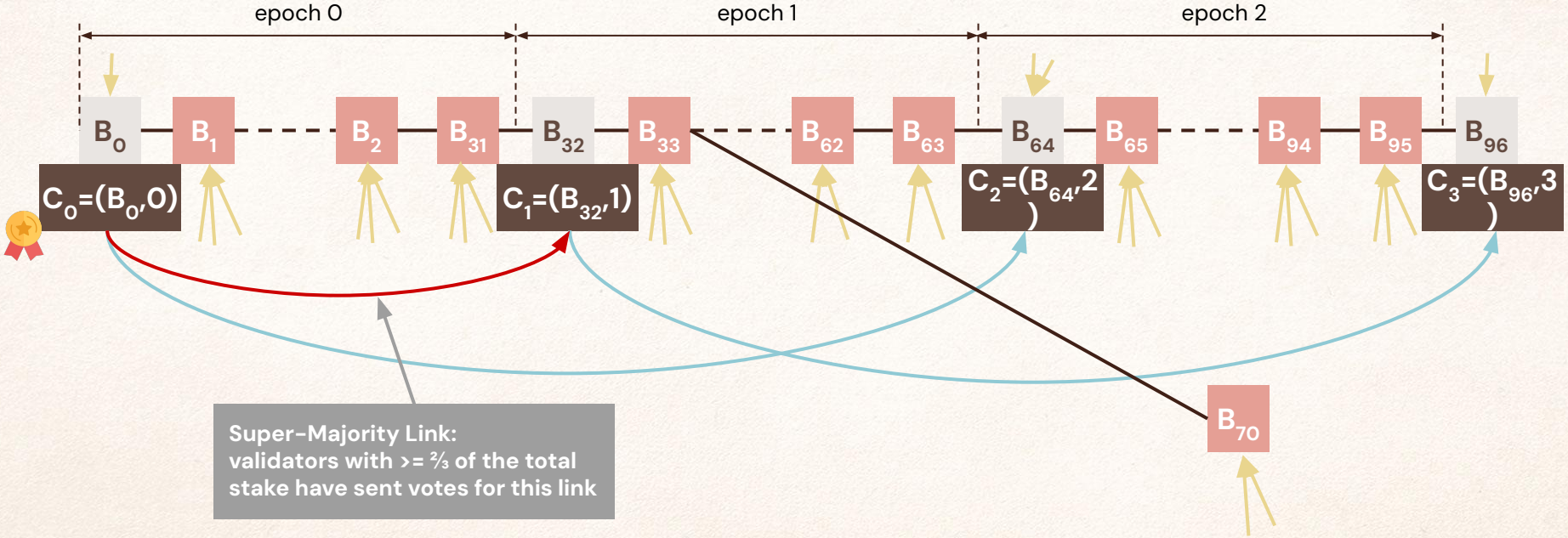


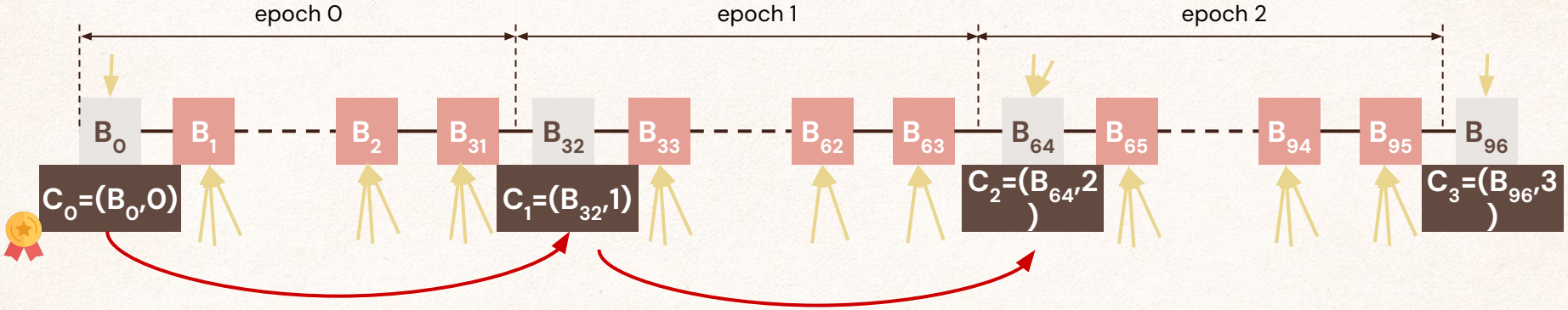
FFG Vote
Vote on links between checkpoints
FFG = Casper FFG = Casper Friendly Finality Gadget

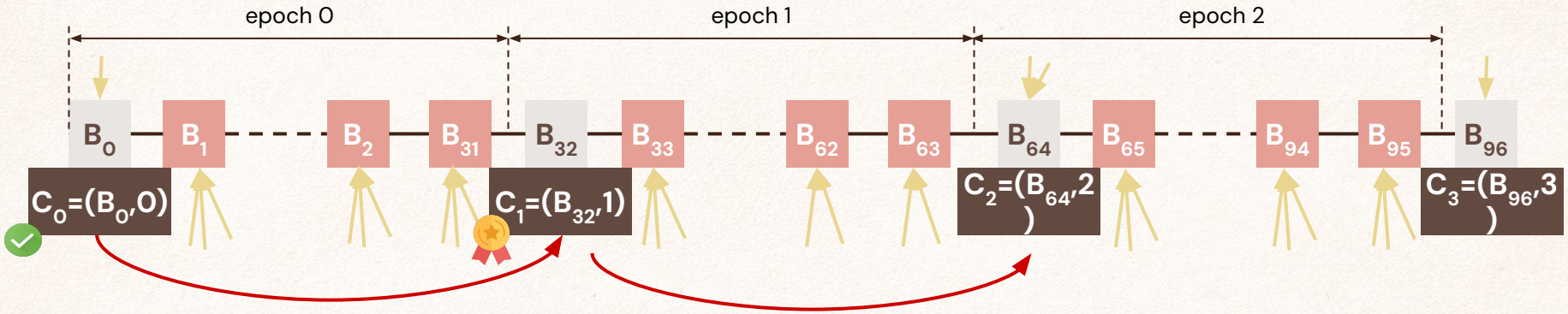
LMD-GHOST Vote
Vote on blocks
LMD-GHOST = Latest Message Driven Greediest Heaviest Observed SubTree


epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96
C0=(B0,0)
C1=(B32,1)
C2=(B64,2)
C3=(B96,3)
B70

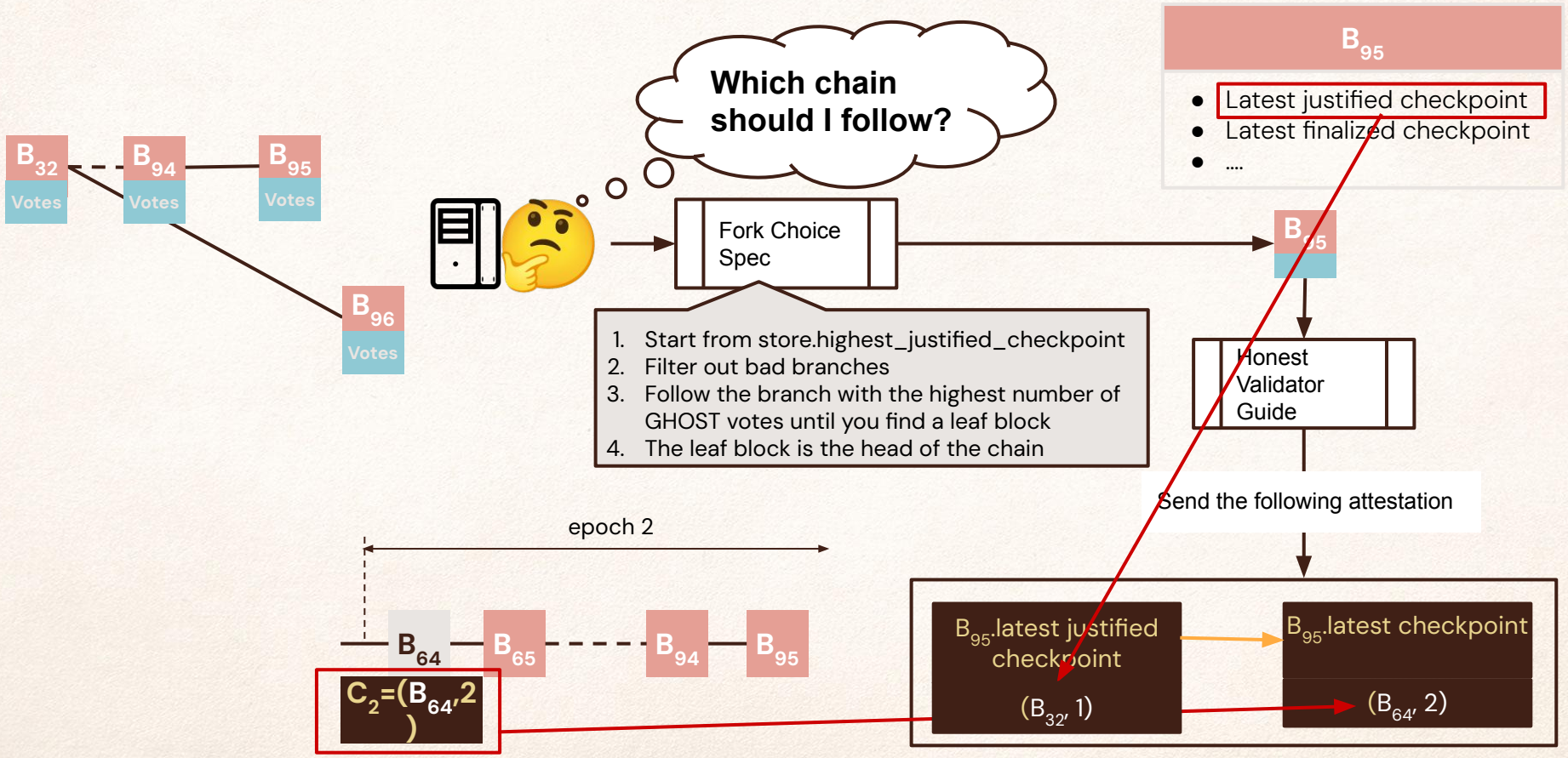

epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96
C0=(B0,0)
C1=(B32,1)
C2=(B64,2)
C3=(B96,3)
B70
Super-Majority Link:
validators with >= ⅔ of the total
stake have sent votes for this link


epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96
C0=(B0,0)
C1=(B32,1)
C2=(B64,2)
C3=(B96,3)
B70
Super-Majority Link:
validators with >= ⅔ of the total
stake have sent votes for this link


epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96
C0=(B0,0)
C1=(B32,1)
C2=(B64,2)
C3=(B96,3)


epoch 0
epoch 1
epoch 2
B0
B1
B2
B31
B32
B33
B62
B63
B64
B65
B94
B95
B96
C0=(B0,0)
C1=(B32,1)
C2=(B64,2)
C3=(B96,3)

How do validators cast FFG votes?



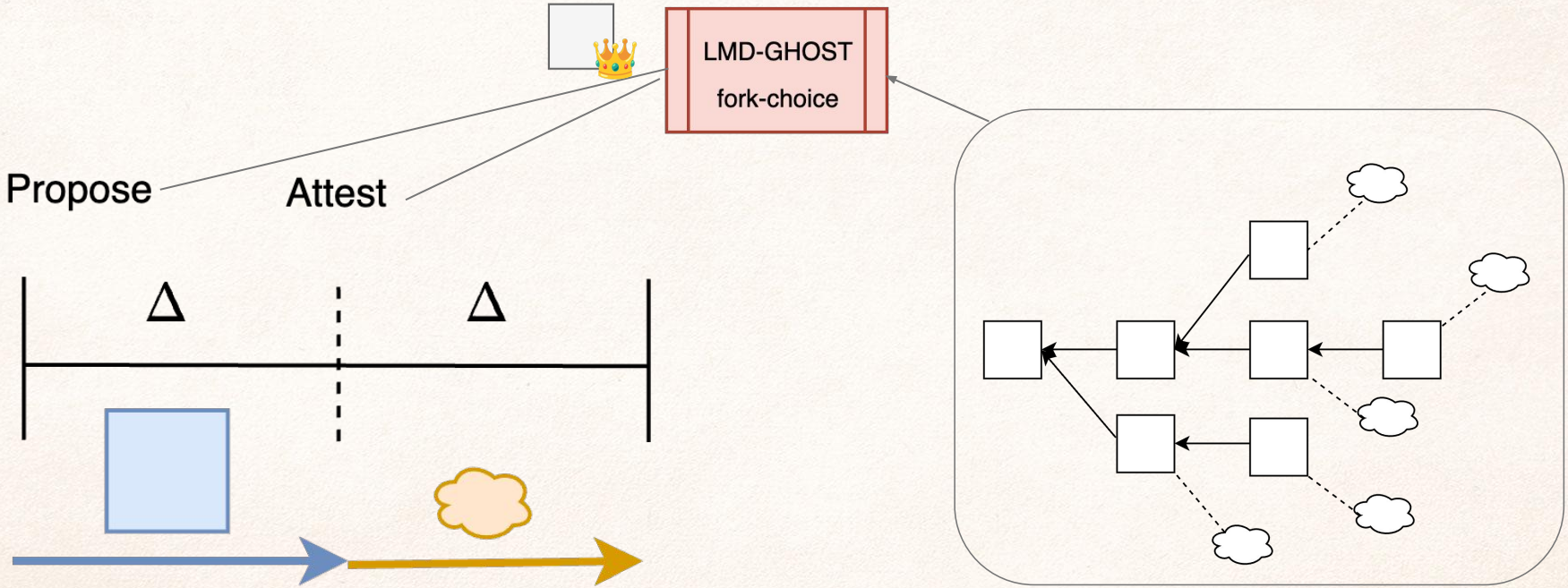


Ethereum Consensus:
The Available Chain

Francesco D'Amato
Ethereum Foundation


LMD-GHOST
(in)security

LMD-GHOST protocol



Security definition

We want a **confirmation rule** that is:

- **safe:** no conflicting blocks are confirmed
- **live:** blocks proposed by honest participants are regularly confirmed

Sleepy model



f < 50% of awake

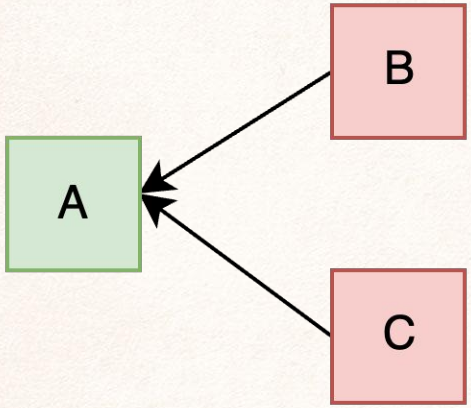


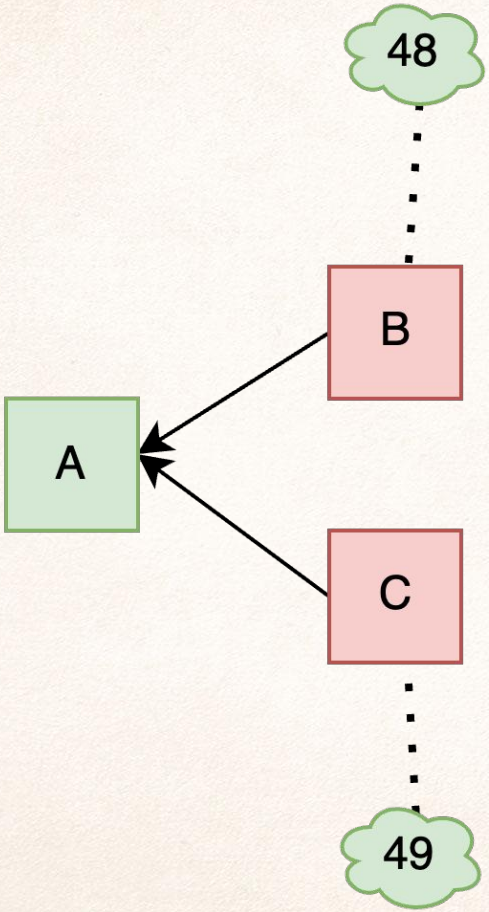
Δ (Synchronous)

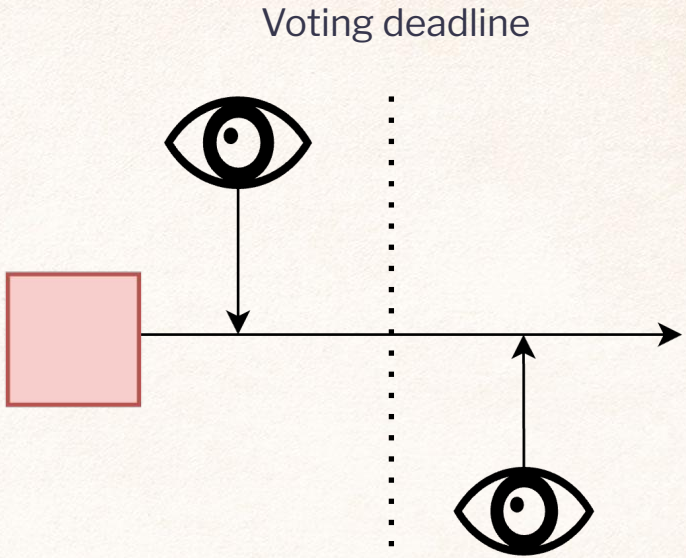


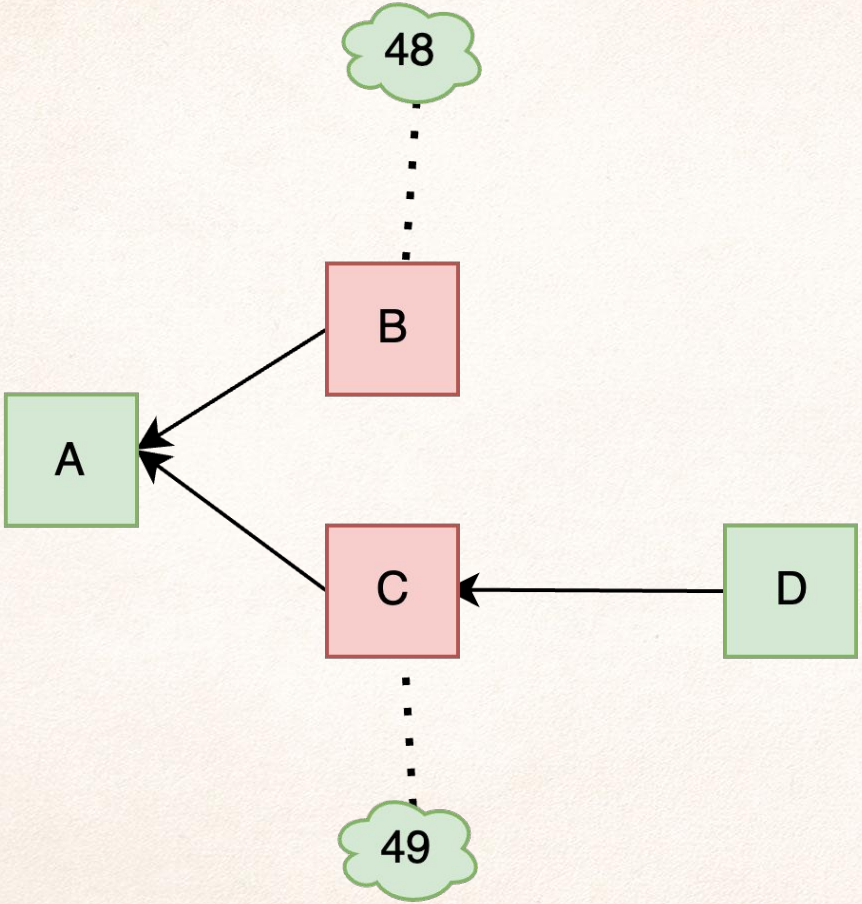
1 honest awake

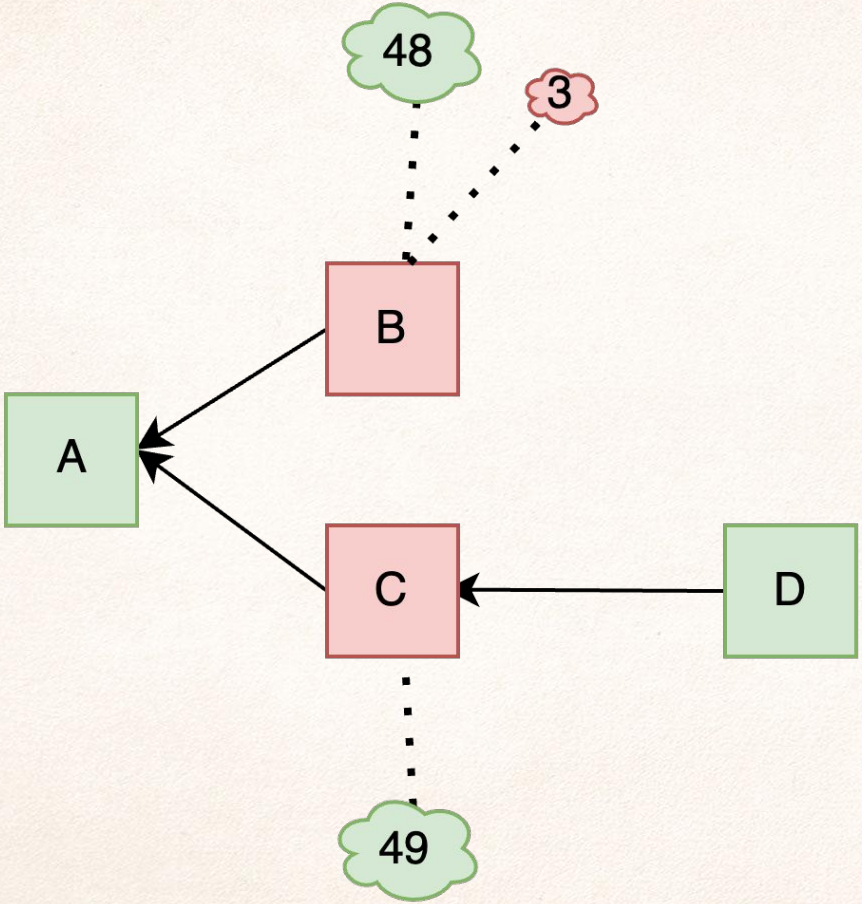
Balancing attack

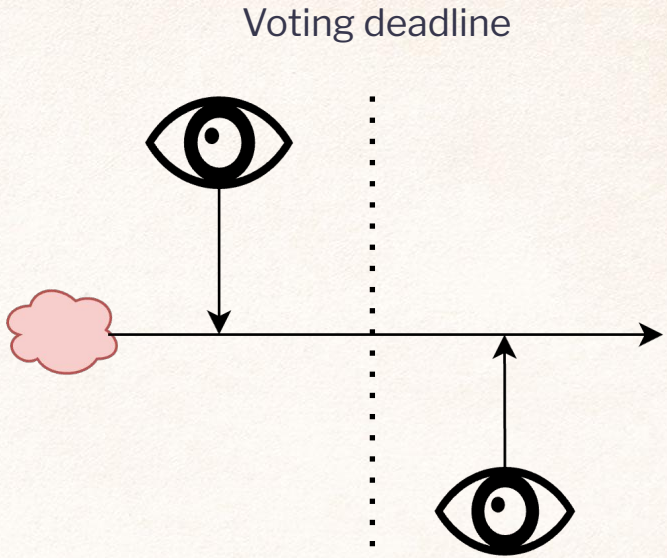


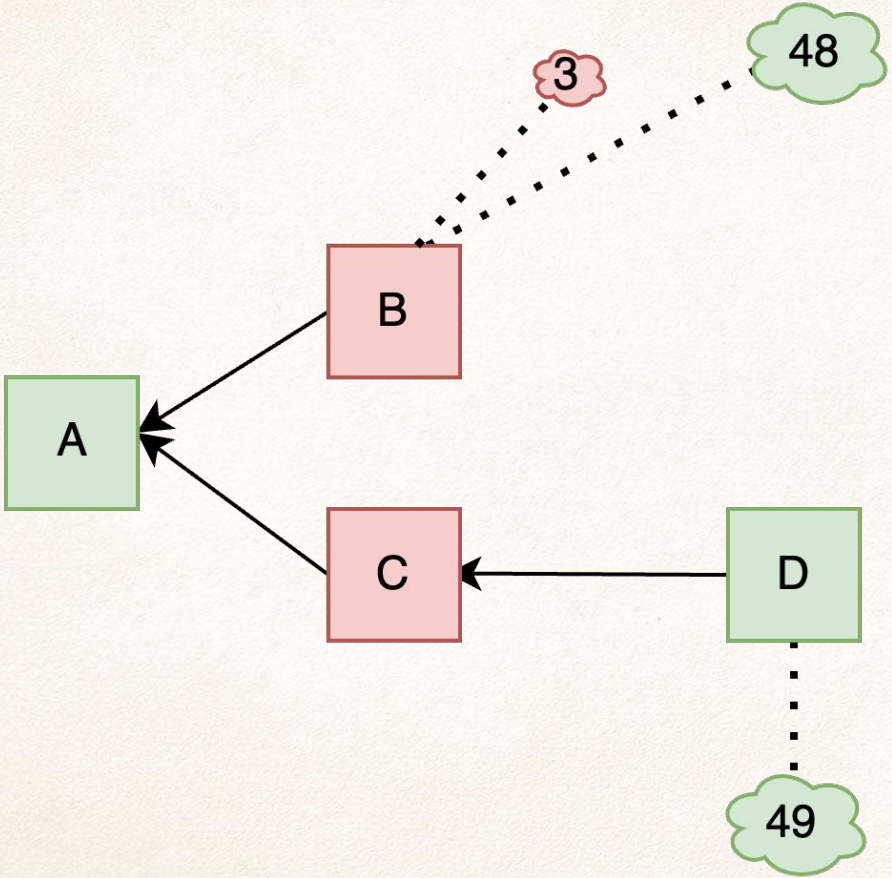

48
B
A
C
49

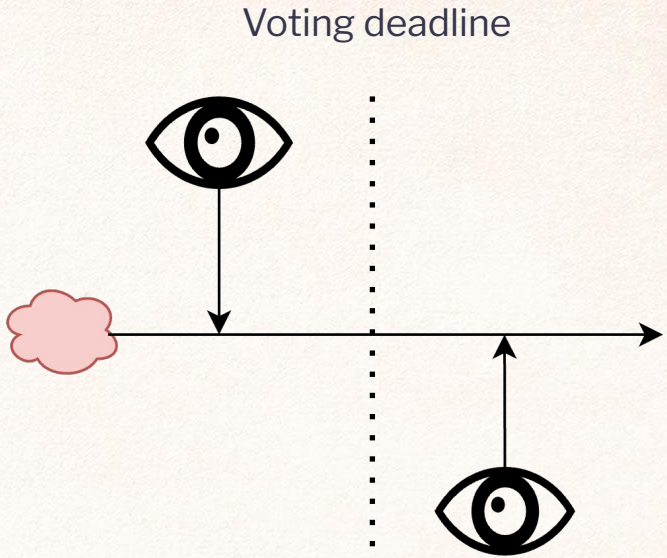

Voting deadline

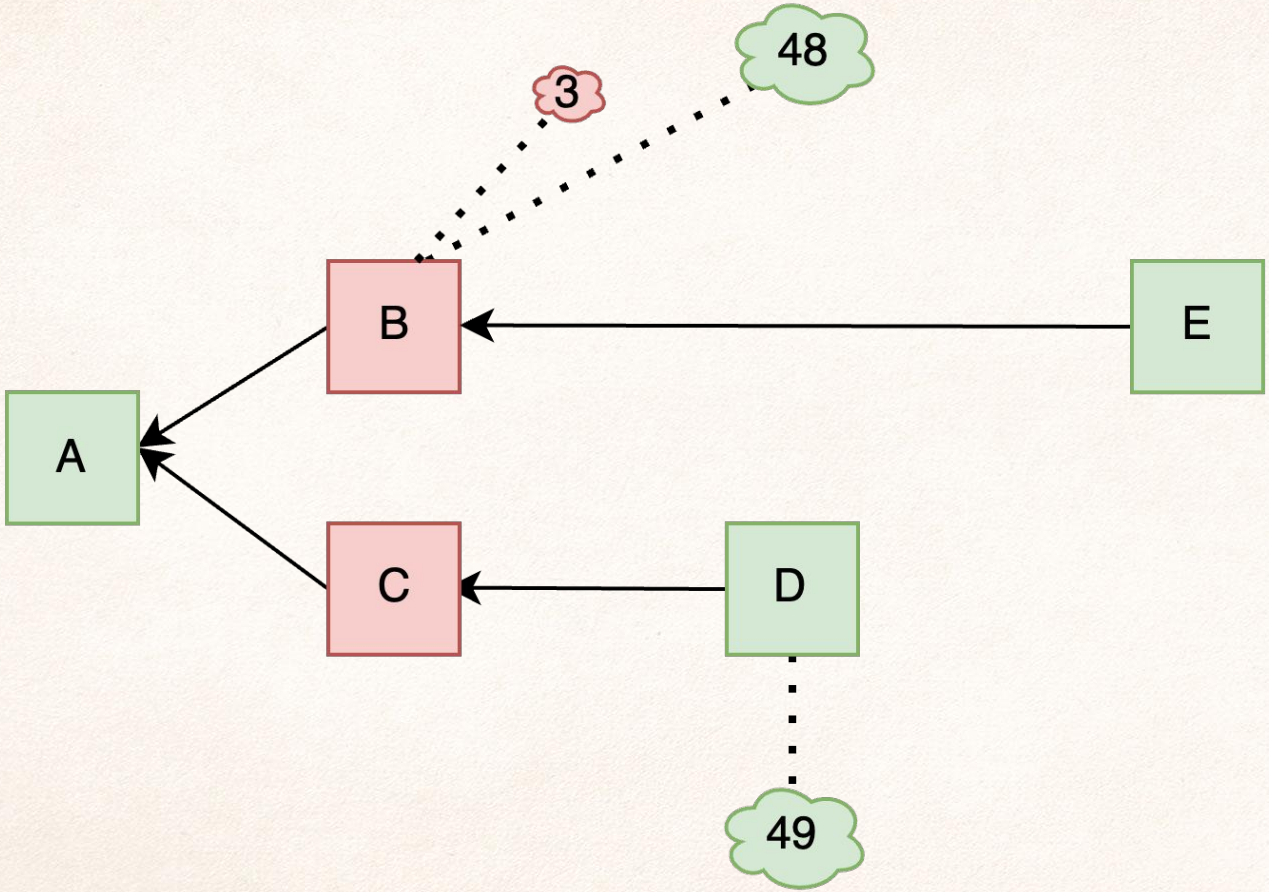

48
B
A
C
D
49

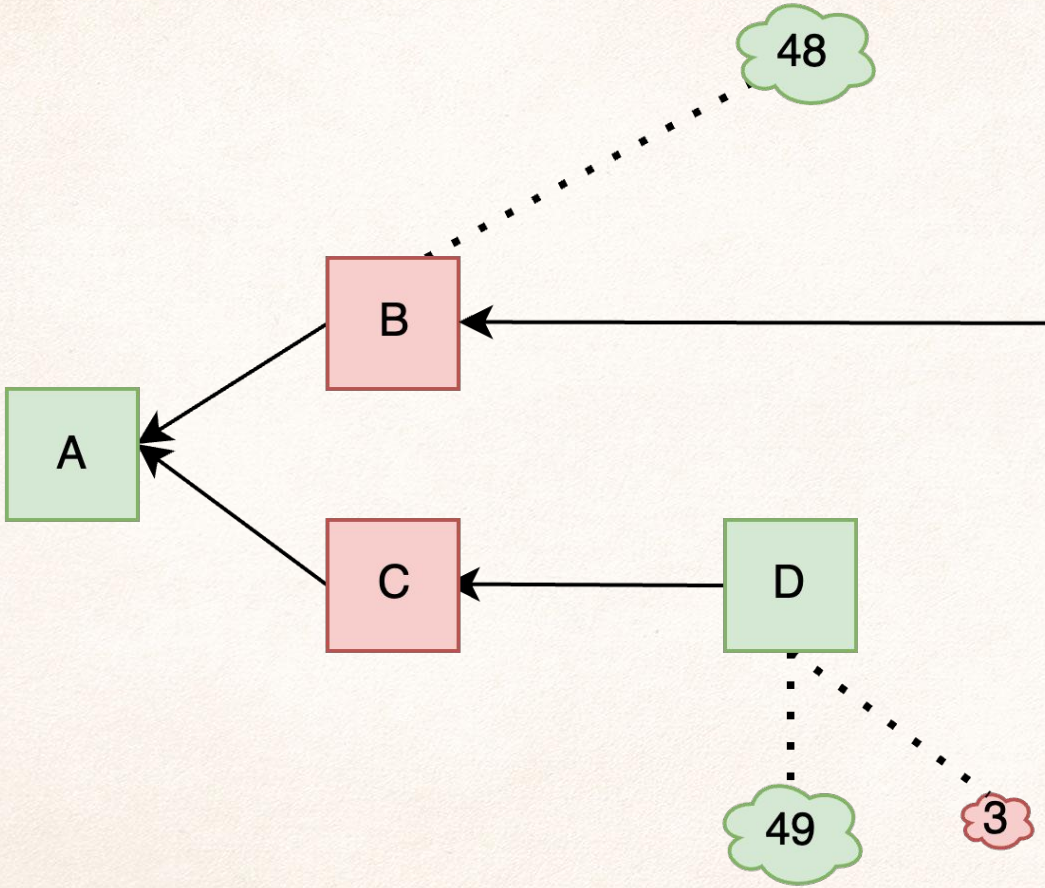

48
3
B
A
C
D
49

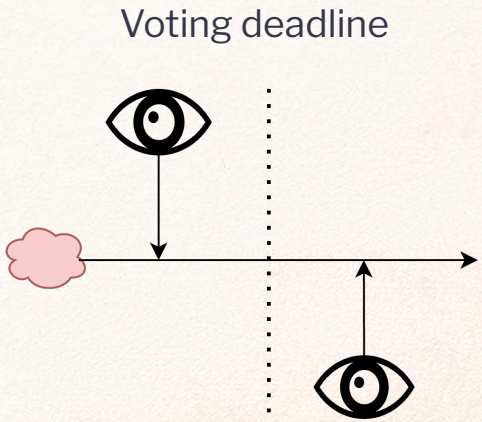

Voting deadline

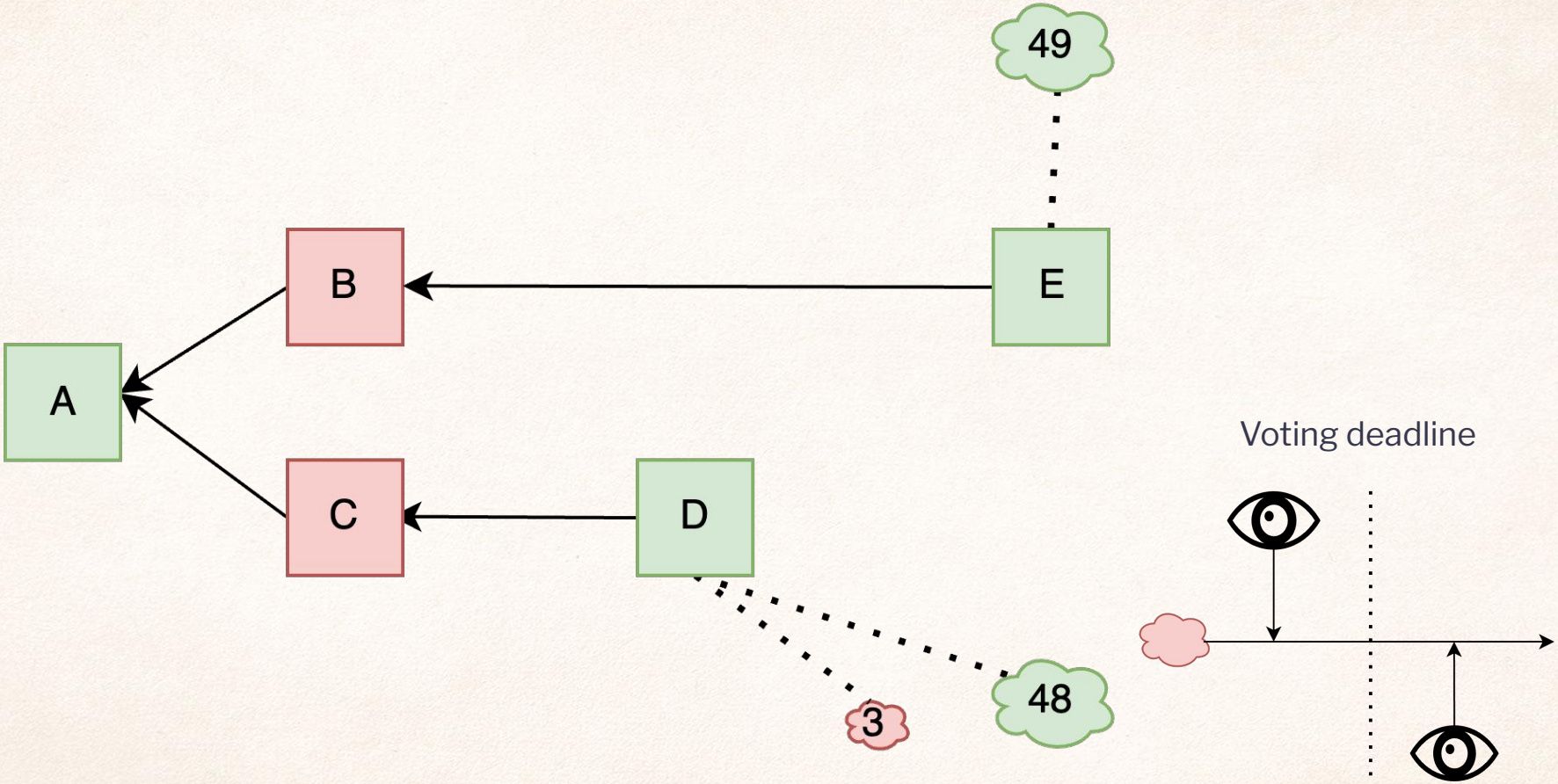

48
3
B
A
C
D
49

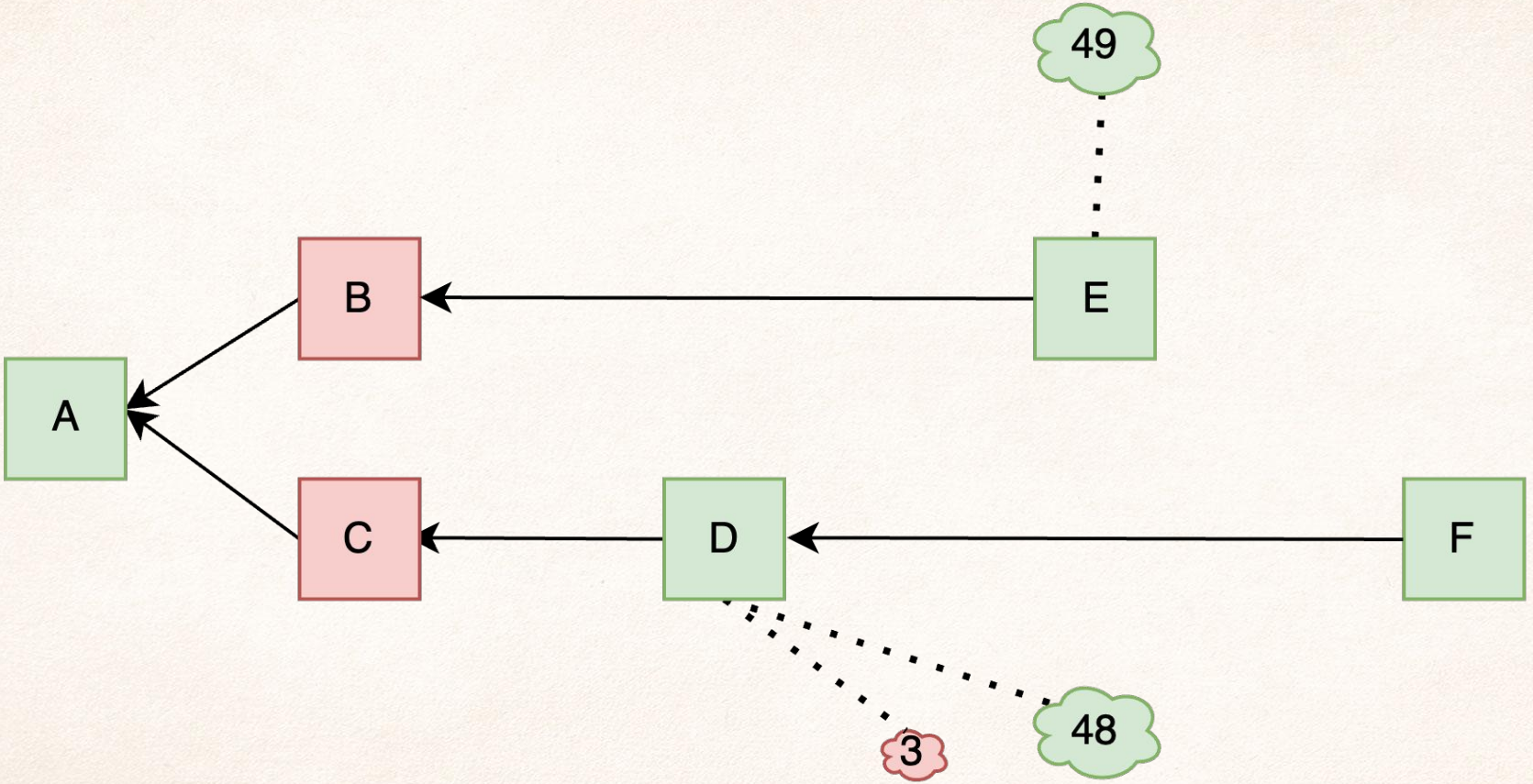

Voting deadline

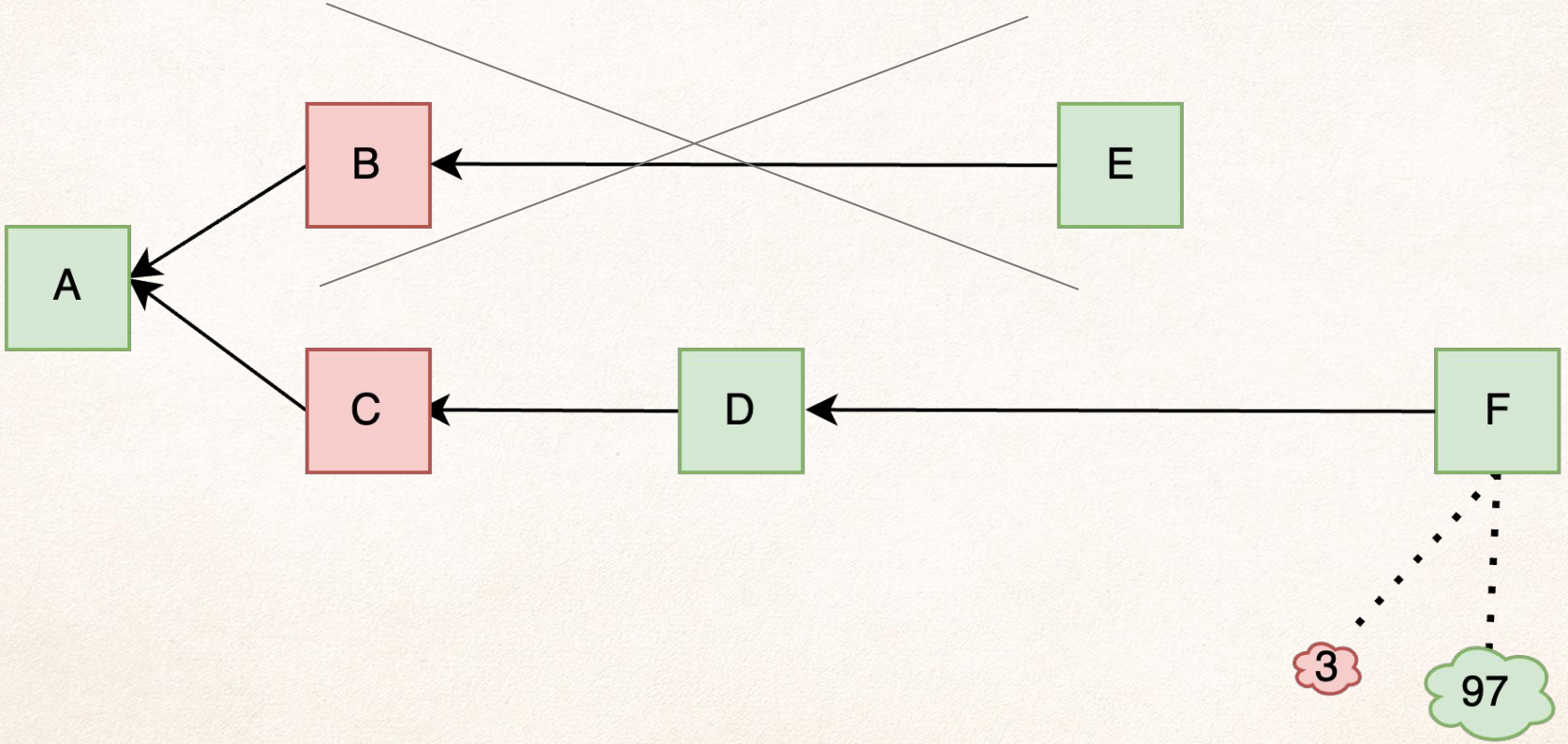

48
3
B
E
A
C
D
49


48
B
E
A
C
D
49
3

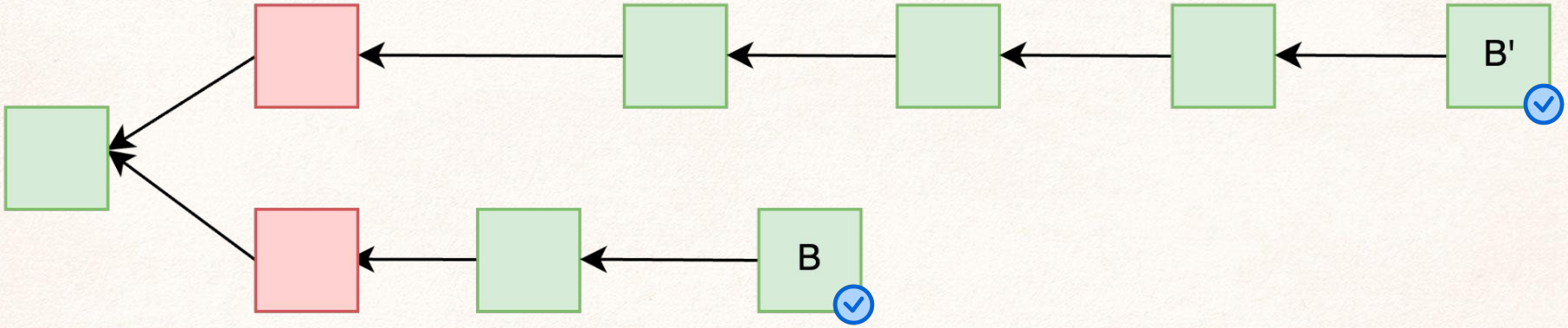

Voting deadline


49
B
E
A
C
D
Voting deadline
48
3


49
B
E
A
C
D
F
3
48


A
B
C
D
E
F
3
97

Balancing attack ⇒ no secure confirmation rule for LMD-GHOST



1. a live confirmation rule would eventually confirm some block B during the attack
2. the attacker collapses the fork-choice to the opposite branch
3. eventually a conflicting B’ is confirmed


Constructing a Secure
Available Chain

Longest chain

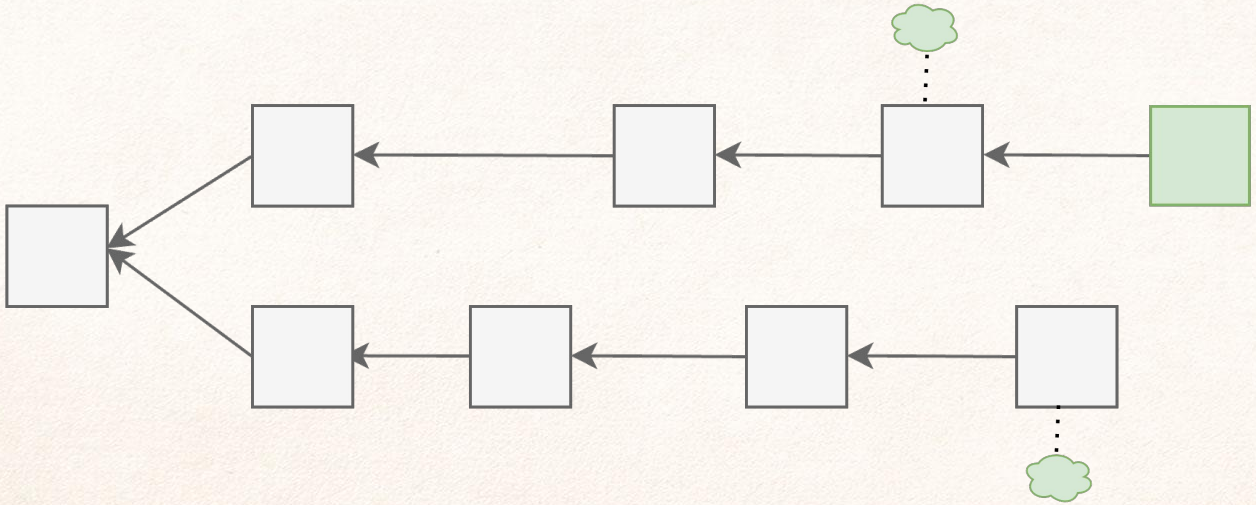
Original dynamically available protocol, but doesn't have:

- Deterministic safety (probabilistic instead)
- Constant latency (κ-deep confirmation, dependent on security parameter)

With parallel votes, we want to do better

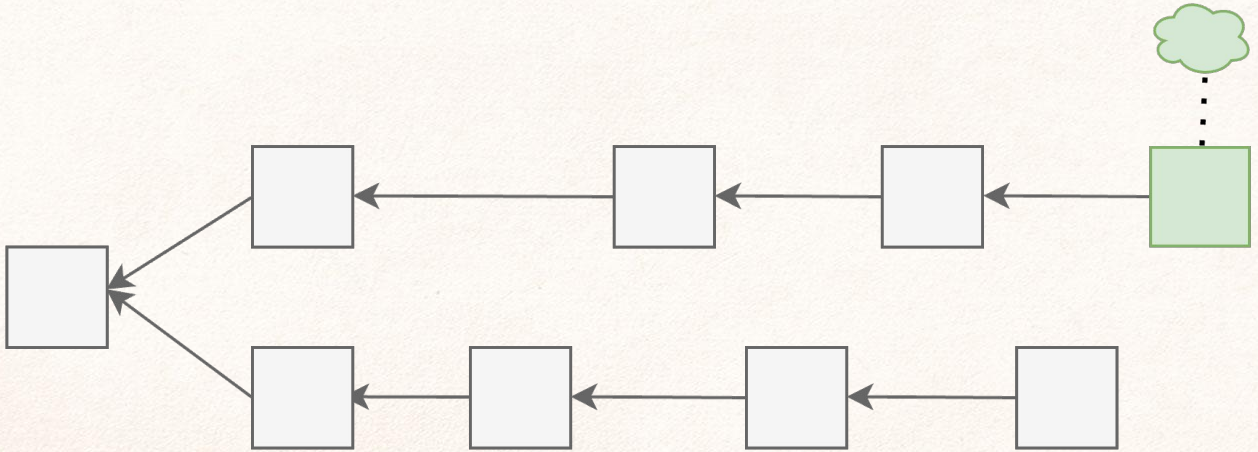
Basic idea: proposer-driven synchronization

Can synchronize how validators vote at least in slots with honest proposers?
To temporarily break the balancing



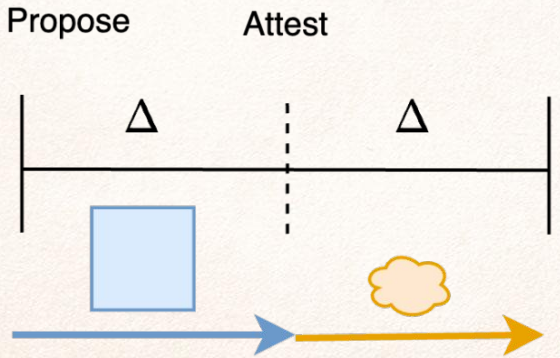
Basic idea: proposer-driven synchronization

Can synchronize how validators vote at least in slots with honest proposers?
To temporarily break the balancing

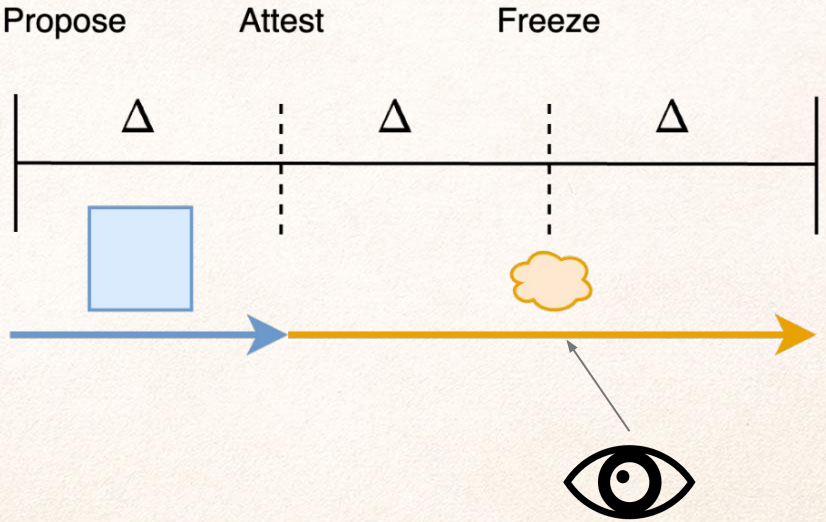


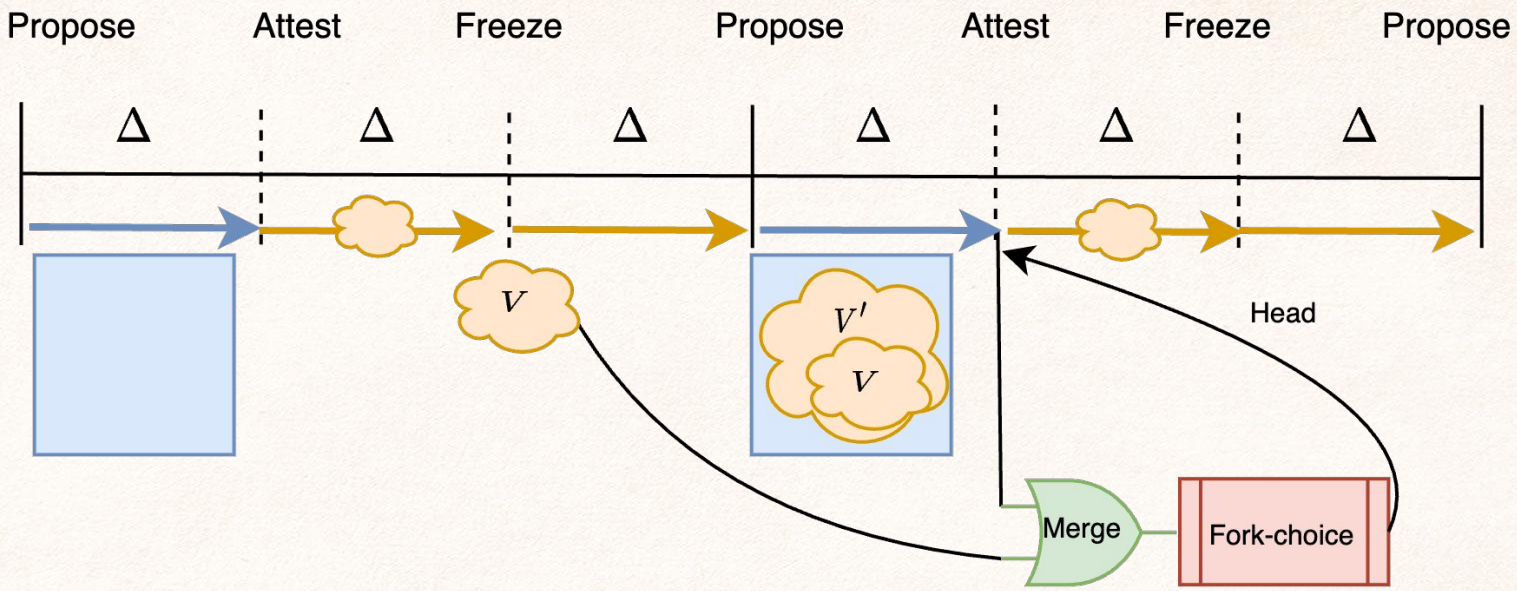
Proposer-driven synchronization: view-merge

LMD-GHOST



LMD-GHOST with view-merge





1. Validator view at freeze time is V
2. Proposer view at proposal time is V', $V \subset V'$
3. Merged view at voting time: $V \cup V' = V'$

Head block:
LMD-GHOST(V')

More proposer synchronization: time-shifted quorum

Constant Latency in Sleepy Consensus

Atsuki Momose
Intelligent Systems Laboratory, SECOM CO., LTD.
Mitaka, Tokyo, Japan
a-momose@secom.co.jp

Ling Ren
University of Illinois at Urbana-Champaign
Urbana, IL, USA
renling@illinois.edu

ABSTRACT

Dynamic participation support is an important feature of Bitcoin's longest-chain protocol and its variants. But these protocols suffer from long latency as a fundamental trade-off. Specifically, the latency depends at least on the following two factors: 1) the desired security level of the protocol, and 2) the actual participation level of the network. Classic BFT protocols, on the other hand, can achieve constant latency but cannot make progress under dynamic participation. In this work, we present a protocol that simultaneously supports dynamic participation and achieves constant latency. Our core technique is to extend the classic BFT approach from static quorum size to *dynamic quorum size*, i.e., according to the current participation level, while preserving important properties of static quorum. We also present a recovery mechanism for rejoining nodes that is efficient in terms of both communication and storage. Our experimental evaluation shows our protocol has much lower latency than a longest-chain protocol, especially when there is a sudden

longest-chain protocols must be noticeably larger than Δ [14, 30]. The block interval further increases if the actual participation level suddenly drops, reducing the total hash rate or active stakeholders (which is why the latency is inverse proportional to γ).

On the other hand, classic BFT protocols can achieve (expected or amortized) latency of $O(\Delta)$ [1, 19]. This motivates the following natural question:

Can we design BFT consensus protocols that simultaneously support dynamic participation and achieve $O(\Delta)$ latency?

In order to answer this question, we must formalize the consensus problem and the model of dynamic participation. We will focus on the BFT atomic broadcast problem [8, 12], where parties agree on a ledger (a linearizable log). As for the model, our starting point will be the *sleepy model* of Pass and Shi [31] (and we will further relax the model later). The sleepy model assumes that the number of honest participants can fluctuate at the adversary's control, with the constraint that there are more honest parties than Byzantine

More proposer synchronization: time-shifted quorum

Constant Latency in Sleepy Consensus

Atsuki Momose
Intelligent Systems Laboratory, SECOM CO., LTD.
Mitaka, Tokyo, Japan
a-momose@secom.co.jp

Ling Ren
University of Illinois at Urbana-Champaign

ABSTRACT

Dynamic participation support is an important feature of Bitcoin's longest-chain protocol and its variants. But these protocols suffer from long latency as a fundamental trade-off. Specifically, the latency depends at least on the following two factors: 1) the desired security level of the protocol, and 2) the actual participation level of the network. Classic BFT protocols, on the other hand, can achieve constant latency but cannot make progress under dynamic participation. In this work, we present a protocol that simultaneously supports dynamic participation and achieves constant latency. Our core technique is to extend the classic BFT approach from static quorum size to *dynamic quorum size*, i.e., according to the current participation level, while preserving important properties of static quorum. We also present a recovery mechanism for rejoining nodes that is efficient in terms of both communication and storage. Our experimental evaluation shows our protocol has much lower latency than a longest-chain protocol, especially when there is a sudden

Towards Practical Sleepy BFT

Dahlia Malkhi[†], Atsuki Momose[*‡], and Ling Ren[‡]

[†]Chainlink Labs
[‡]University of Illinois at Urbana-Champaign

September 26, 2023

Abstract

Bitcoin's longest-chain protocol pioneered consensus under dynamic participation, also known as sleepy consensus, where nodes do not need to be permanently active. However, existing solutions for sleepy consensus still face two major issues, which we address in this work. First, existing sleepy consensus protocols have high latency (either asymptotically or concretely). We tackle this problem and achieve 4Δ latency (Δ is the bound on network delay) in the best case, which is comparable to classic BFT protocols without dynamic participation support. Second, existing protocols have to assume that the set of corrupt participants remains fixed throughout the lifetime of the protocol due to a problem we call *costless simulation*. We resolve this problem and support growing participation of corrupt nodes. Our new protocol also offers several other important advantages, including support for arbitrary fluctuation of honest participation as well as an efficient recovery mechanism for new active nodes.

More proposer synchronization: time-shifted quorum

Constant Latency in Sleepy Consensus

Atsuki Momose
Intelligent Systems Laboratory, SECOM CO., LTD.
Mitaka, Tokyo, Japan
a-momose@secom.co.jp

Ling Ren
University of Illinois at Urbana-Champaign

ABSTRACT

Dynamic participation support is an important feature of Bitcoin's longest-chain protocol and its variants. But these protocols suffer from long latency as a fundamental trade-off. Specifically, the latency depends at least on the following two factors: 1) the desired security level of the protocol, and 2) the actual participation level of the network. Classic BFT protocols, on the other hand, can achieve constant latency but cannot make progress under dynamic participation. In this work, we present a protocol that simultaneously supports dynamic participation and achieves constant latency. Our core technique is to extend the classic BFT approach from static quorum size to *dynamic quorum size*, i.e., according to the current participation level, while preserving important properties of static quorum. We also present a recovery mechanism for rejoining nodes that is efficient in terms of both communication and storage. Our experimental evaluation shows our protocol has much lower latency than a longest-chain protocol, especially when there is a sudden

Towards Practical Sleepy BFT

Dahlia Malkhi[†], Atsuki Momose[*‡], and Ling Ren[‡]

[†]Chainlink Labs
[‡]University of Illinois at Urbana-Cham

September 26, 2023

Abstract

Bitcoin's longest-chain protocol pioneered consensus under dynami
sleepy consensus, where nodes do not need to be permanently active.
sleepy consensus still face two major issues, which we address in this work.
protocols have high latency (either asymptotically or concretely). We t
4Δ latency (Δ is the bound on network delay) in the best case, which
protocols without dynamic participation support. Second, existing protoc
of corrupt participants remains fixed throughout the lifetime of the pro
costless simulation. We resolve this problem and support growing parti
new protocol also offers several other important advantages, including s
of honest participation as well as an efficient recovery mechanism for nev

TOB-SVD: Total-Order Broadcast with Single-Vote Decisions in the Sleepy Model

Francesco D'Amato
Ethereum Foundation
francesco.damato@ethereum.org

Roberto Saltini
Ethereum Foundation
roberto.saltini@ethereum.org

Thanh-Hai Tran
Independent Researcher
thanhhai1302@gmail.com

Luca Zanolini
Ethereum Foundation
luca.zanolini@ethereum.org

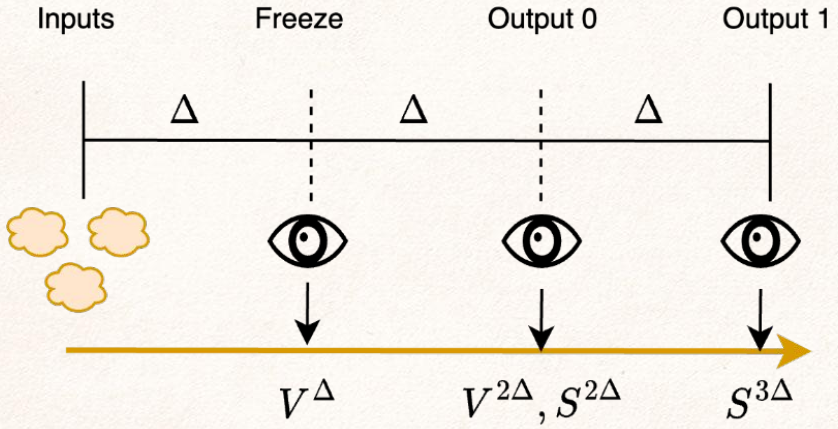
Abstract

Over the past years, distributed consensus research has expanded its focus to address challenges in large-scale, permissionless systems, such as blockchains. This shift reflects the need to accommodate dynamic participation, in contrast to the traditional model of a static set of continuously online validators. Works like Bitcoin and the sleepy model have laid the groundwork for this evolving framework.

Notable contributions by Momose and Ren (CCS 2022), along with subsequent research, have introduced Total-Order Broadcast protocols that leverage Graded Agreement primitives and support dynamic participation. However, these approaches often require multiple phases of voting per decision, which can create a bottleneck for real-world, large-scale systems.

To address this, our paper introduces TOB-SVD, a novel Total-Order Broadcast protocol in the

More proposer synchronization: time-shifted quorum



Output $(\lambda, 0)$ if $|V_\lambda^{2\Delta}| > |S^{2\Delta}|/2$

Output $(\lambda, 1)$ if $|V_\lambda^{\Delta}| > |S^{3\Delta}|/2$

If someone outputs $(\lambda, 1)$ then everyone outputs $(\lambda, 0)$

Graded Agreement

Security?

LMD-GHOST with view-merge is secure with respect to the kappa-deep confirmation rule under the normal synchronous model (GST = 0, GAT = 0)

Security?

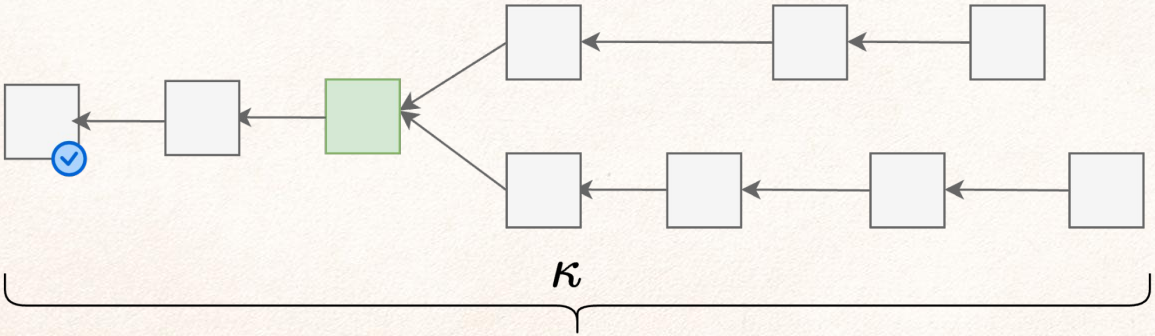
LMD-GHOST with view-merge is secure with respect to the kappa-deep confirmation rule under the normal synchronous model (GST = 0, GAT = 0)

Key result:

Reorg-resilience
Blocks proposed by honest proposers never leave the canonical chain (are never reorged)

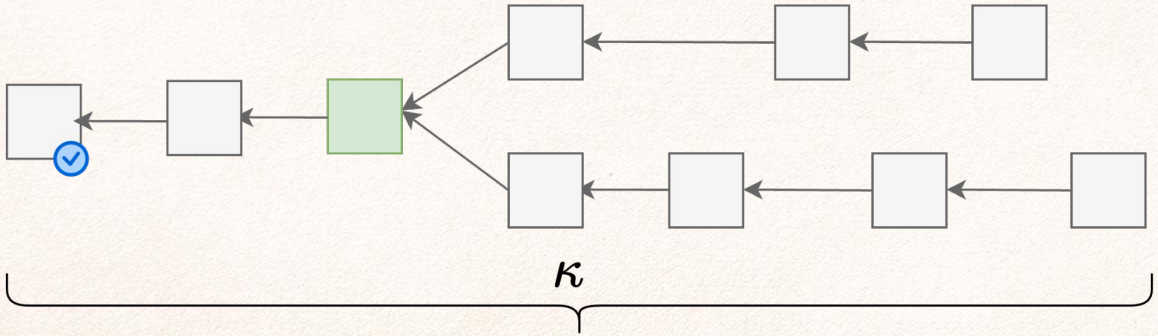
Reorg resilience => κ-deep confirmation is secure

With high probability, every range of κ blocks has an honest proposer, and so a “stable block” that never gets reorged.



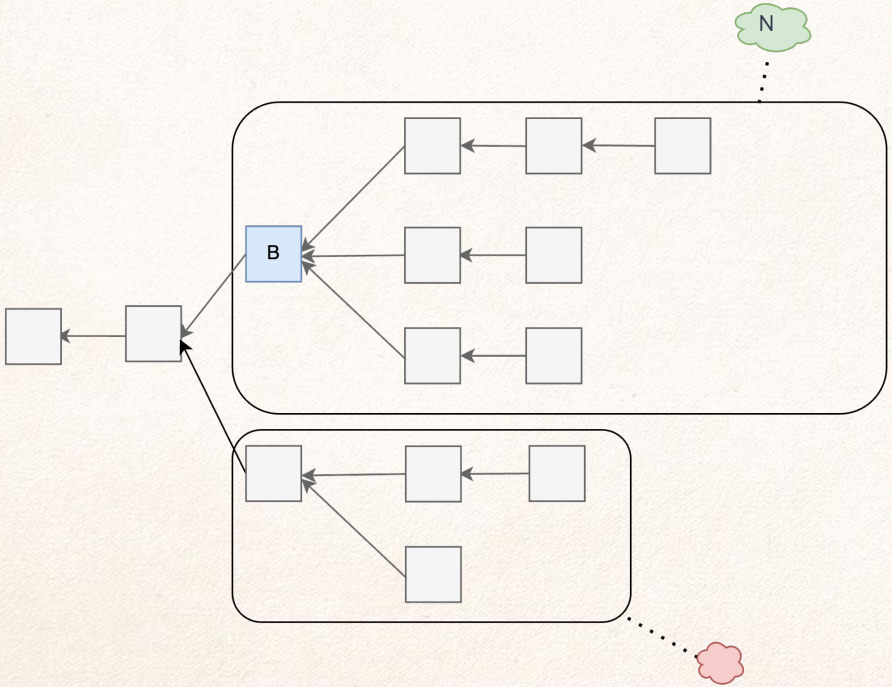
Reorg resilience => κ-deep confirmation is secure

With high probability, every range of κ blocks has an honest proposer, and so a “stable block” that never gets reorged.



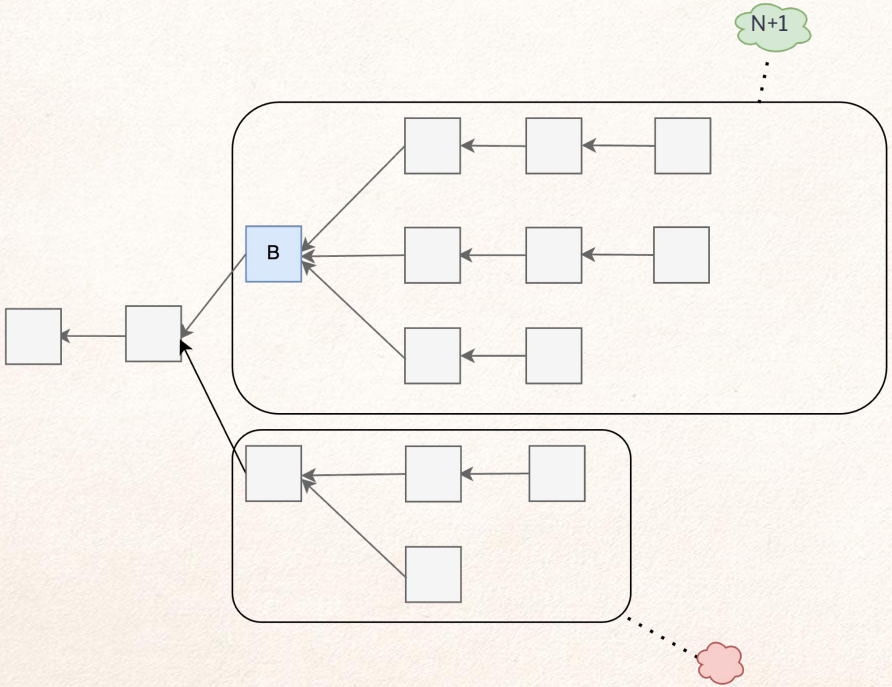
Note: latency O(κ) like longest chain. Can do O(1) as well.

Reorg resilience



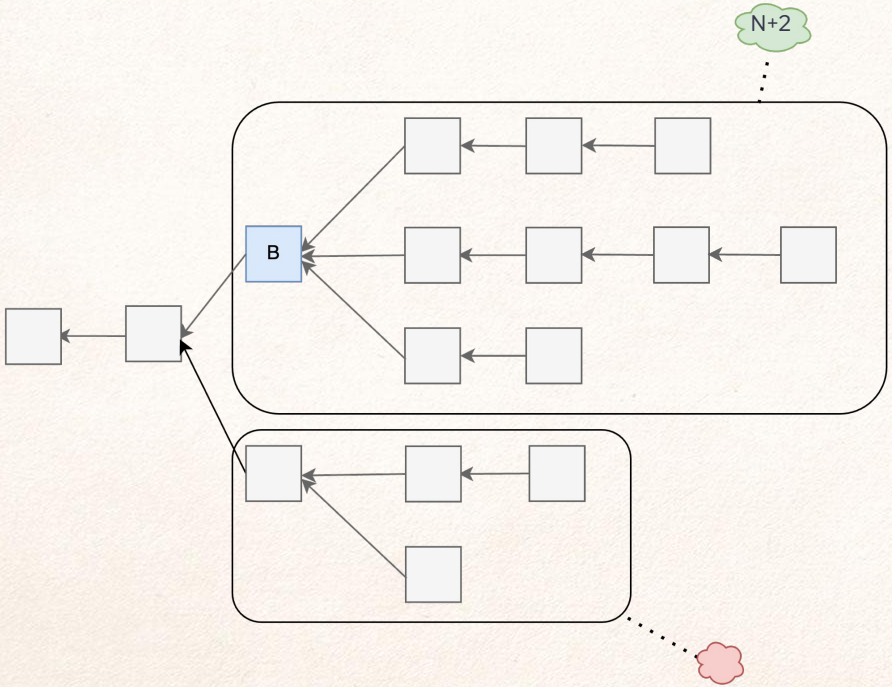
1. *If all honest participants vote on the subtree of block B at slot N, then B is canonical: LMD-GHOST outputs a descendant of B*

Reorg resilience



1. *If all honest participants vote on the subtree of block B at slot N, then B is canonical: LMD-GHOST outputs a descendant of B*
2. *They all vote for a descendant of B at N+1 too, then at N+2 etc…*

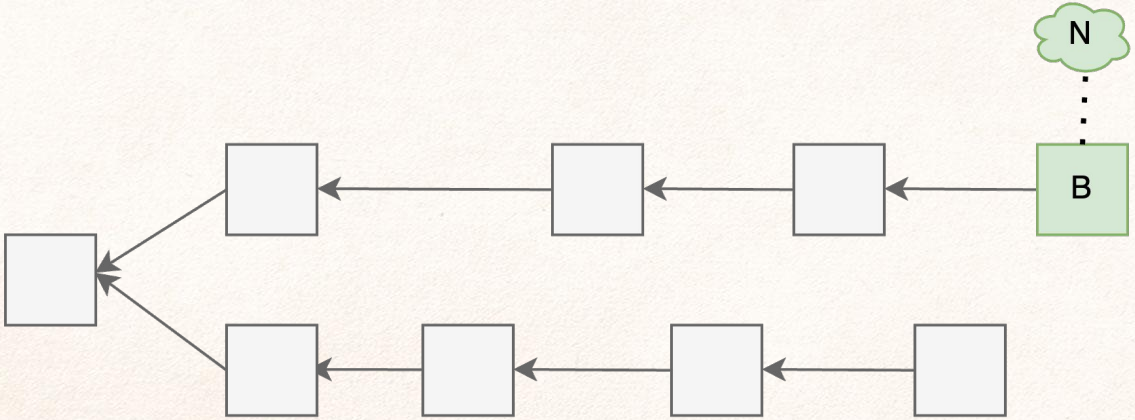
Reorg resilience



1. *If all honest participants vote on the subtree of block B at slot N, then B is canonical: LMD-GHOST outputs a descendant of B*
2. *They all vote for a descendant of B at N+1 too, then at N+2 etc…*
3. *B is never reorged*

Reorg resilience

View-merge gives us the base case: an honest block receives all honest votes



Security!

Security!

… however:

- not yet in the sleepy model
- does not work with committees

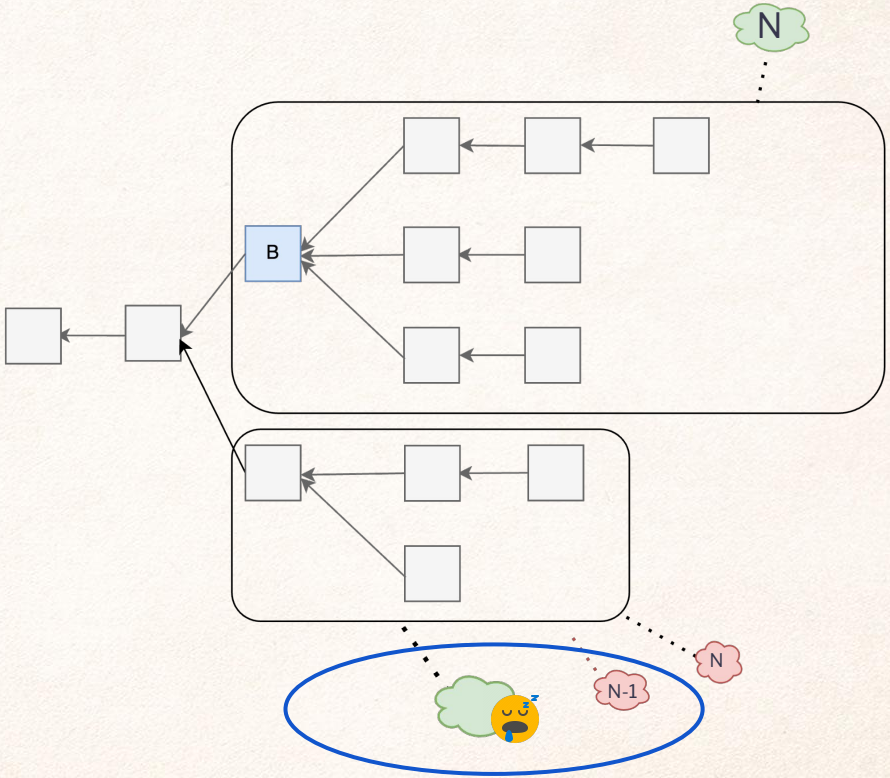
Security!

... however:

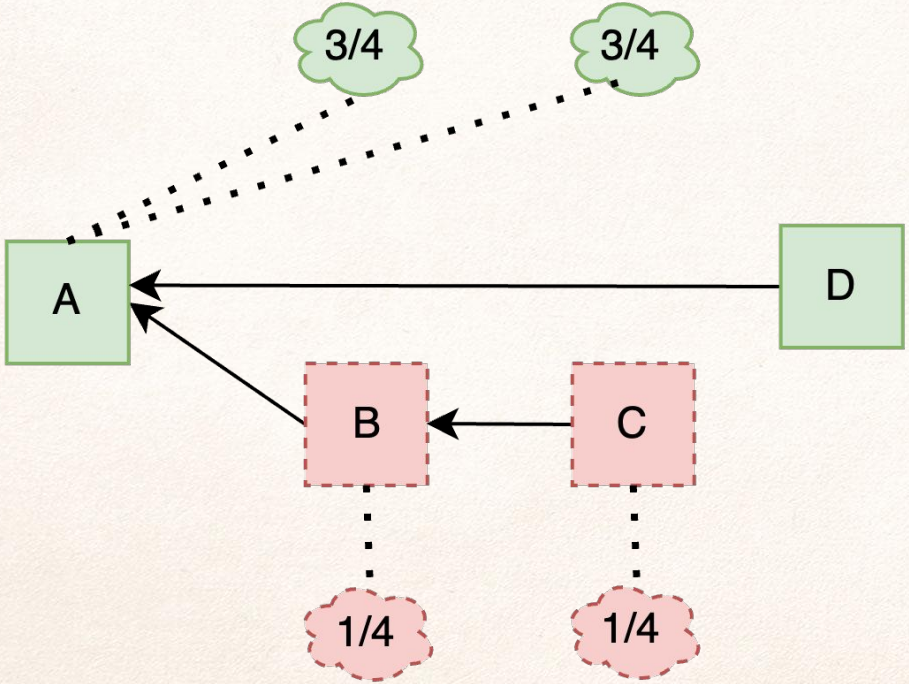
- not yet in the sleepy model
- does not work with committees

For the same reason!
The honest votes at slot N are not a majority of **all** weight if there are:

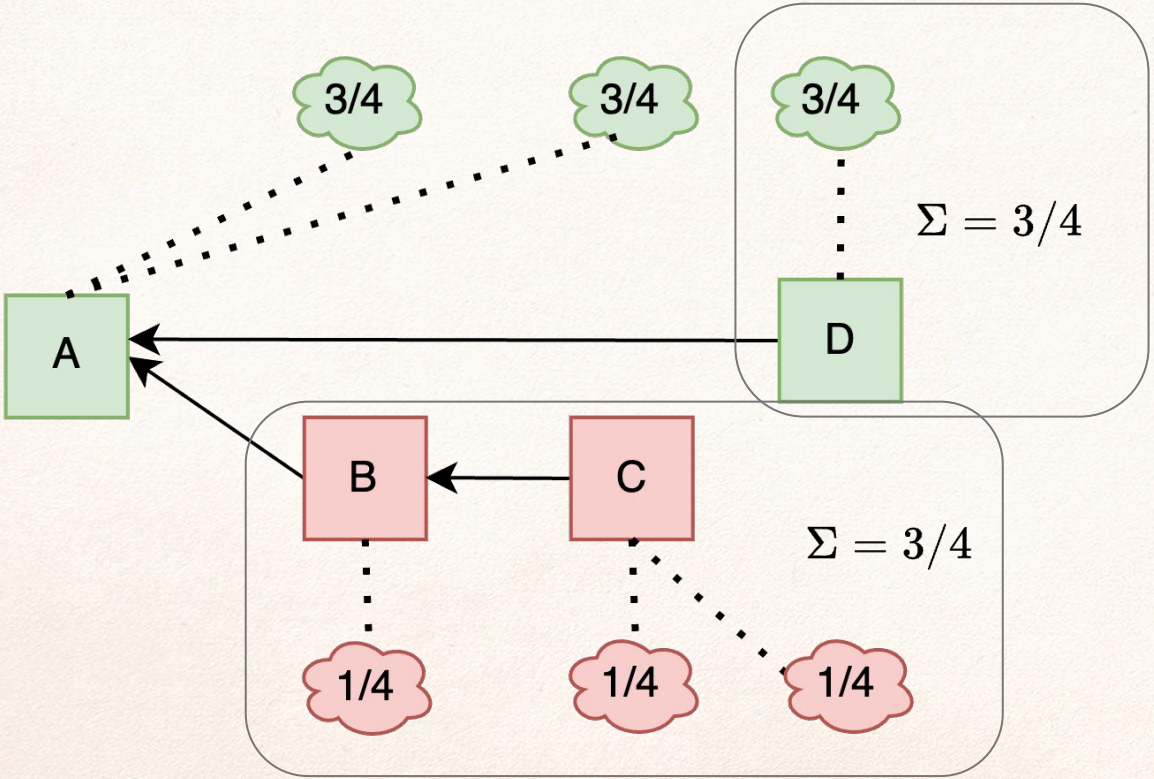
- Votes of asleep validators
- Votes of previous committees



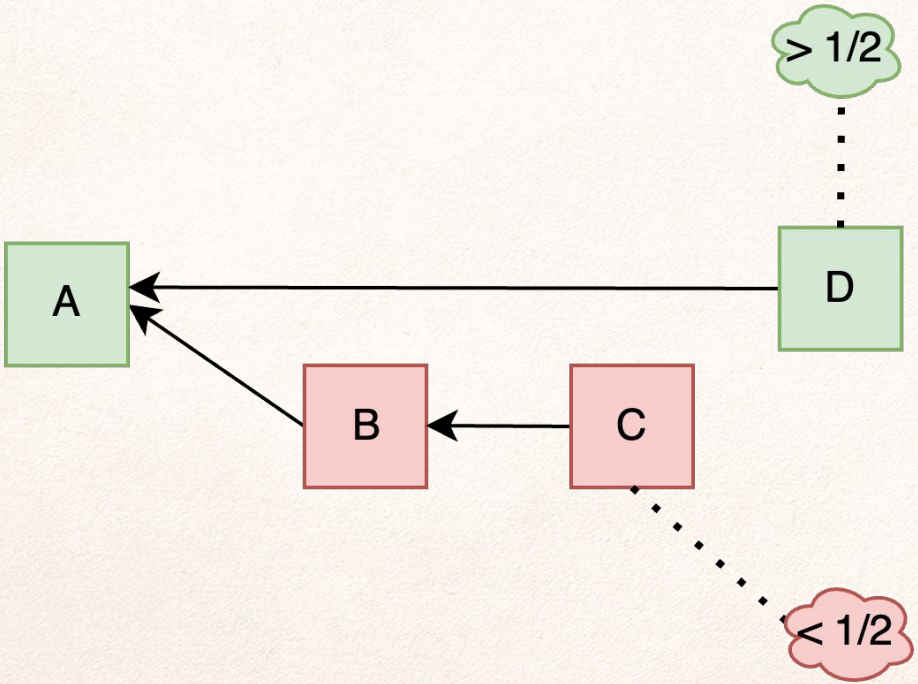
Ex-ante Reorgs



Ex-ante Reorgs



Ex-ante Reorgs

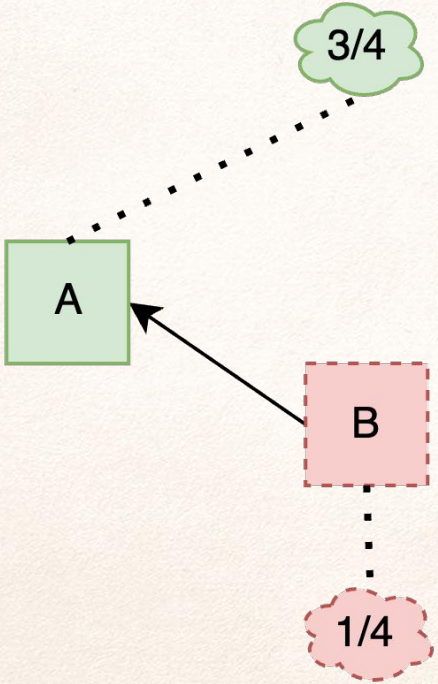


Without committees, **only the latest voting round matters**

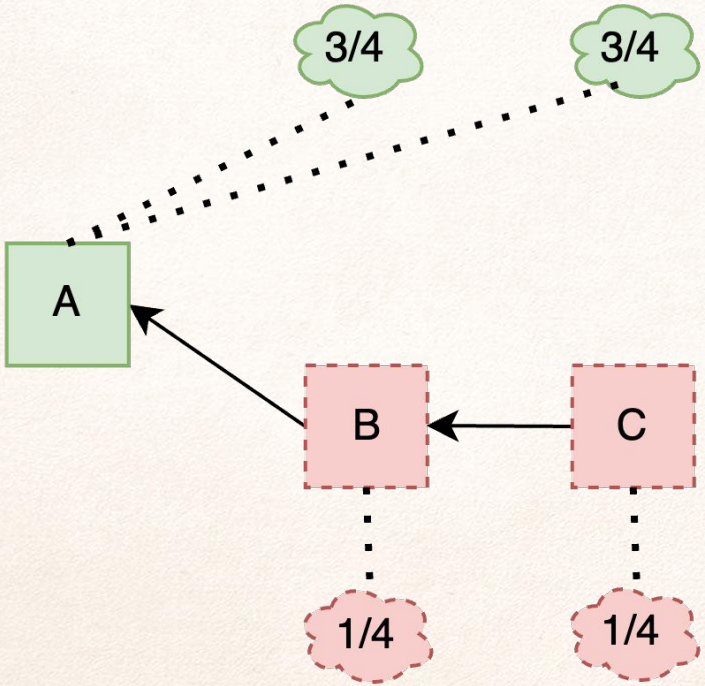
Same trouble with asleep validators! Their latest votes are “stale”, not from the latest round


Goldfish

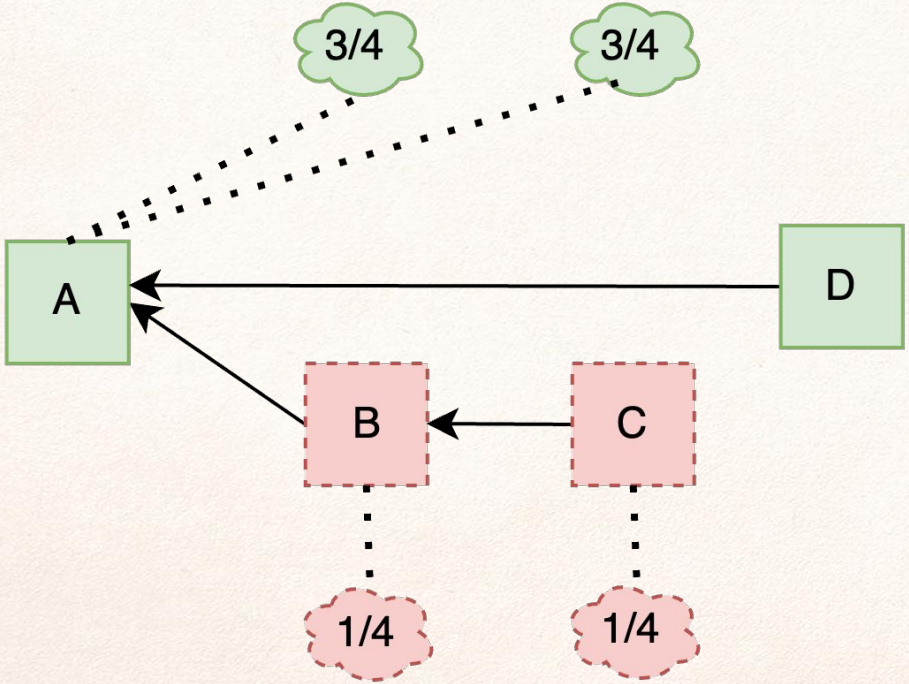
Ex-ante Reorgs (with committees)



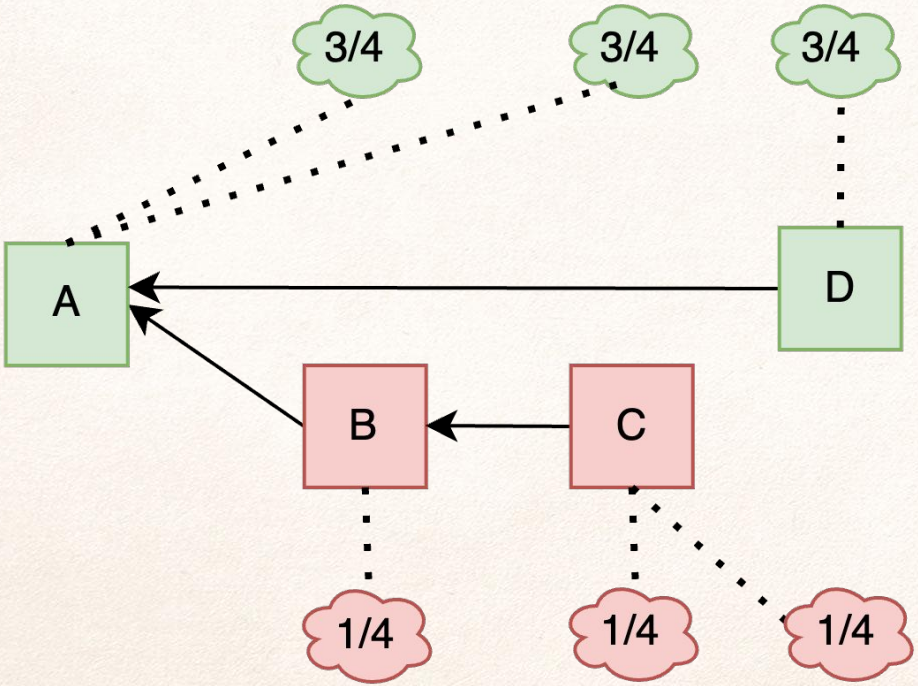
Ex-ante Reorgs



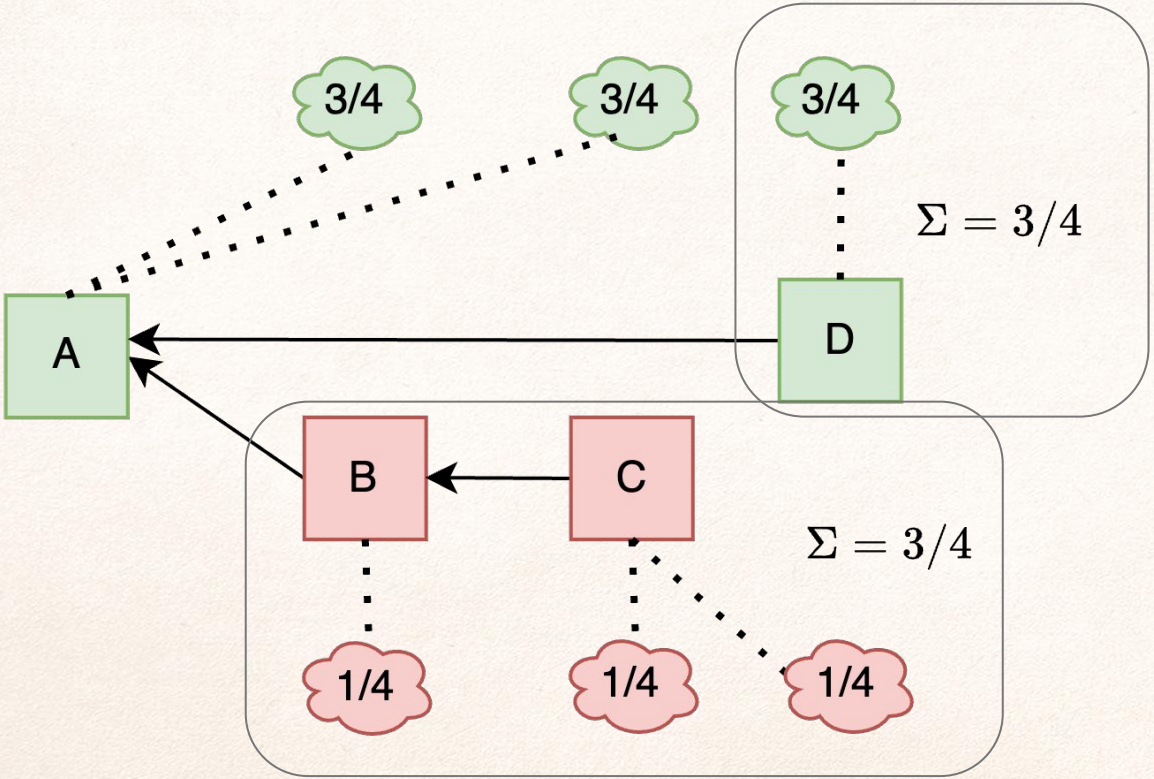
Ex-ante Reorgs



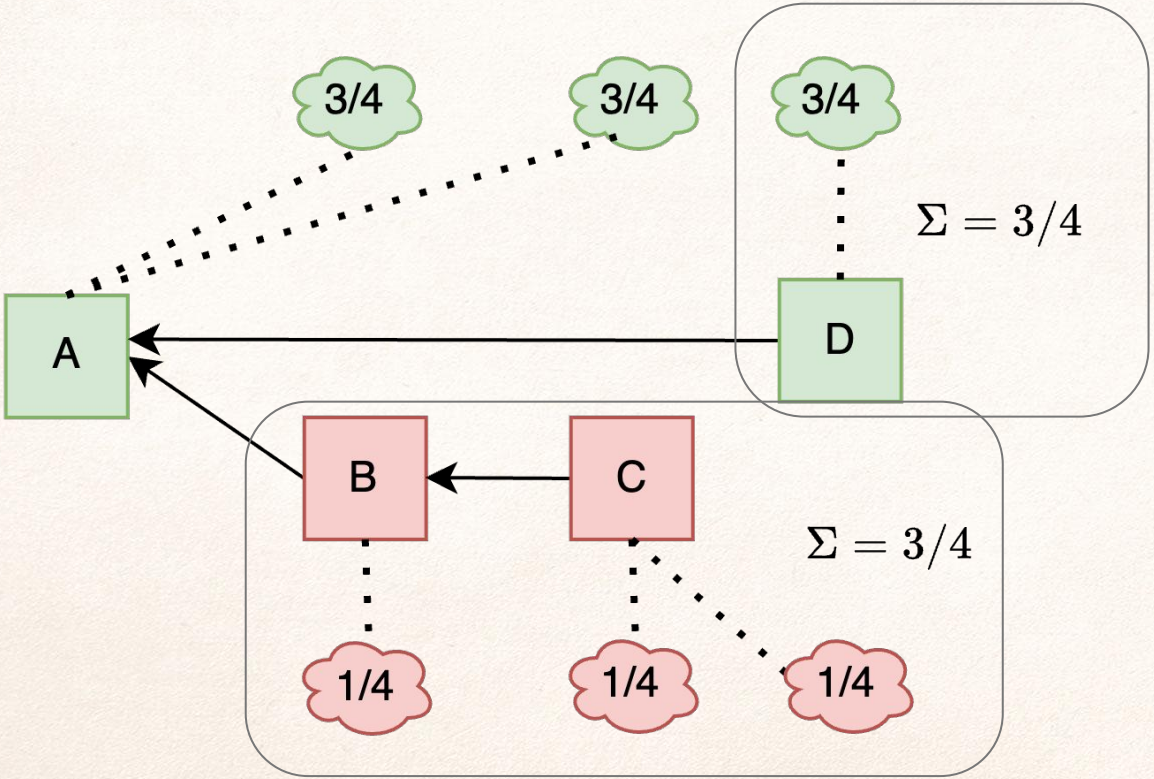
Ex-ante Reorgs



Ex-ante Reorgs



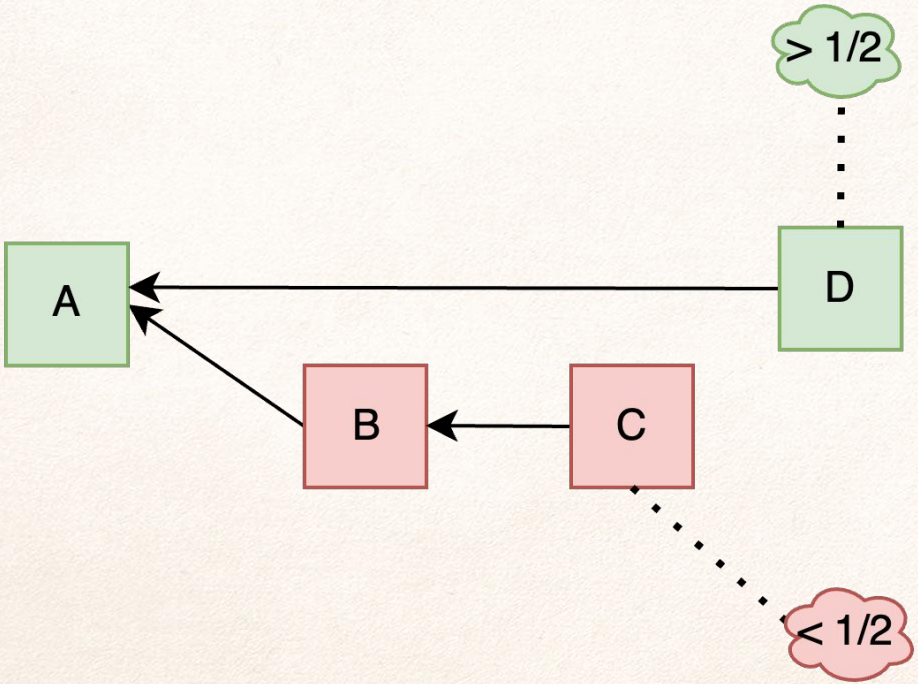
Ex-ante Reorgs



Votes accumulate across committees.

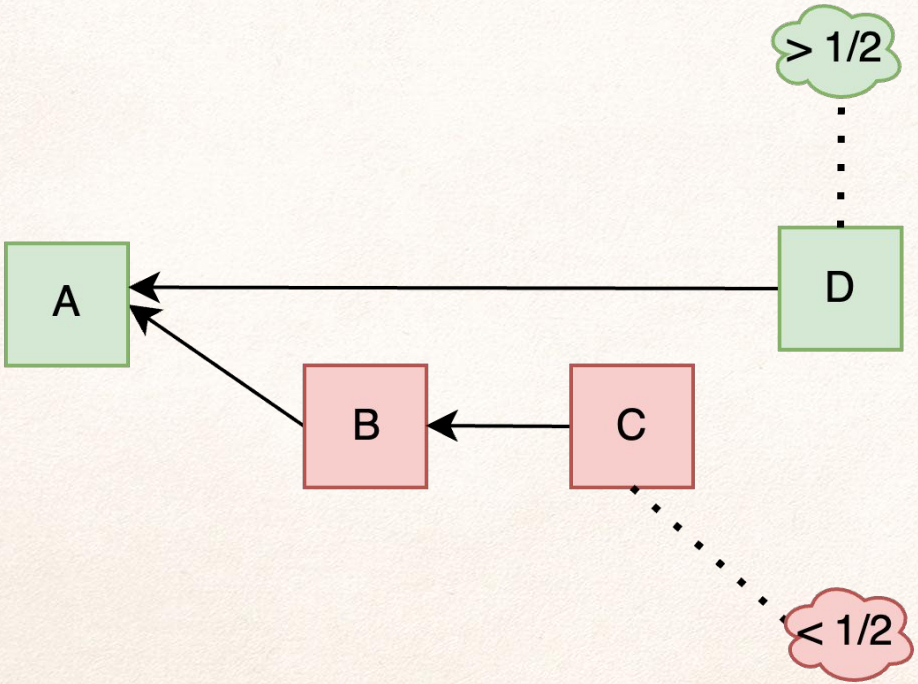
The 3/4 voting for D is 3/4 of one committee, not of the whole validator set!

Ex-ante Reorgs



Without committees, **only the latest voting round matters**

Ex-ante Reorgs

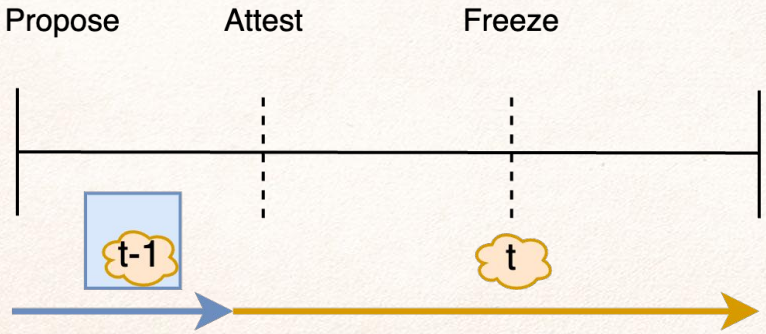


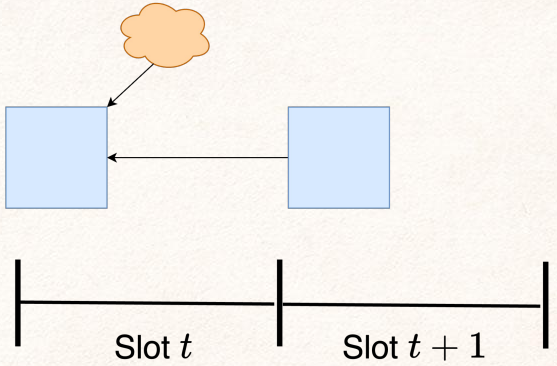
Without committees, **only the latest voting round matters**

Same trouble with asleep validators! Their latest votes are “stale”, not from the latest round

Goldfish = LMD-GHOST + view-merge + vote-expiry

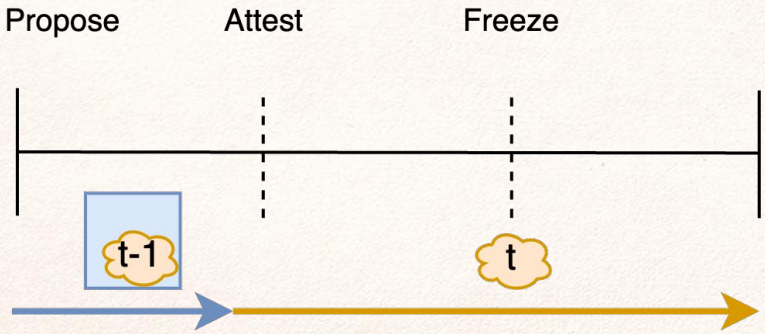
At slot t, only run the fork-choice (GHOST) with votes from slot t-1.
Old votes (previous committees, asleep validators…) have no influence.

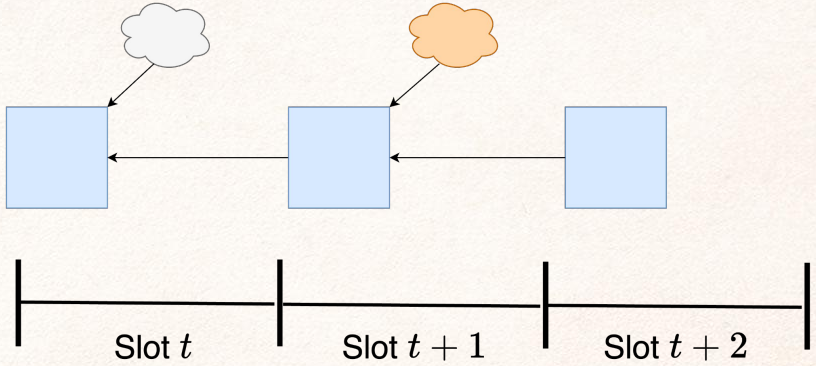




Goldfish = LMD-GHOST + view-merge + vote-expiry

At slot t, only run the fork-choice (GHOST) with votes from slot t-1.
Old votes (previous committees, asleep validators…) have no influence.



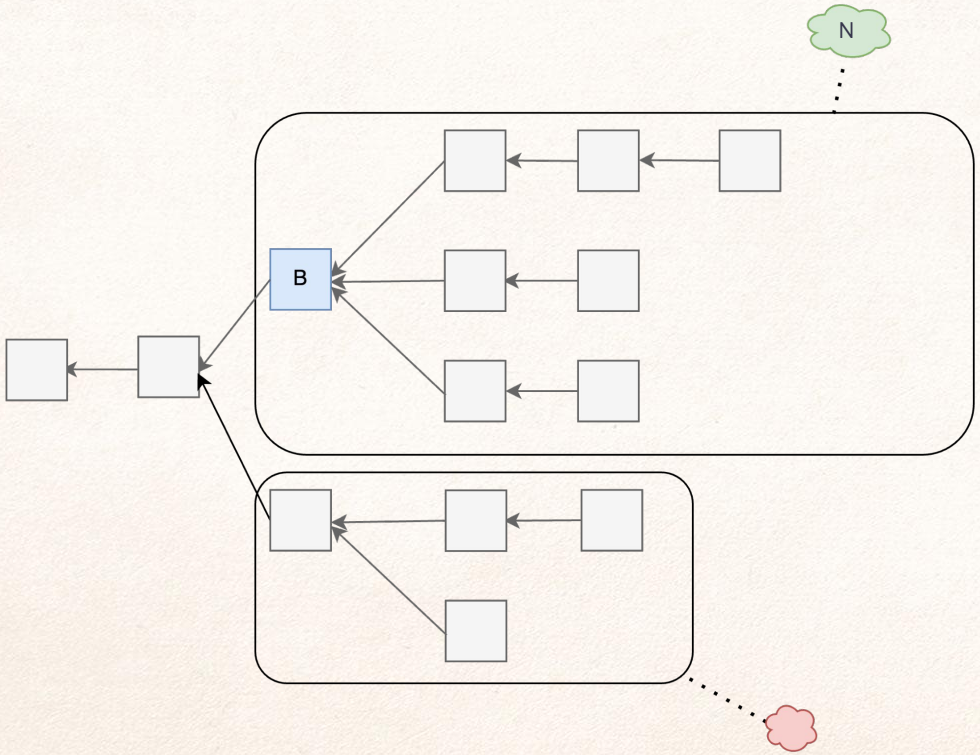


Goldfish



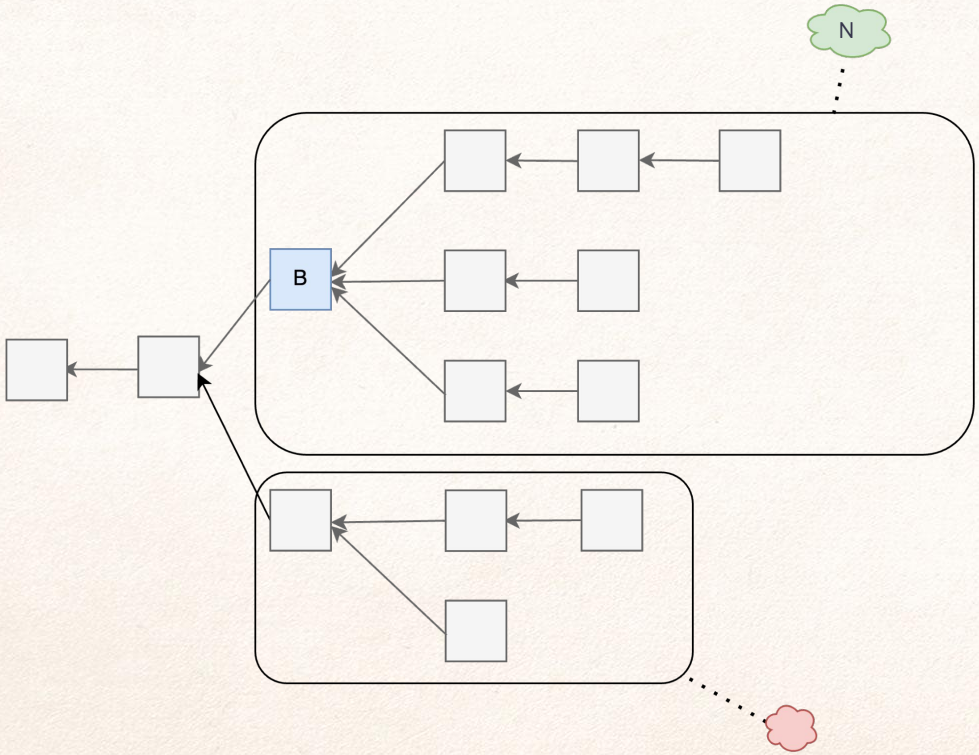
Can imagine that completely new participants show up in each slot, and signal is relayed from a set of awake participants to the next via votes

Goldfish: reorg resilience



Same as before: if all honest votes from slot N are on a subtree, that subtree wins *in slot N+1 (and so on…)*

Goldfish: reorg resilience



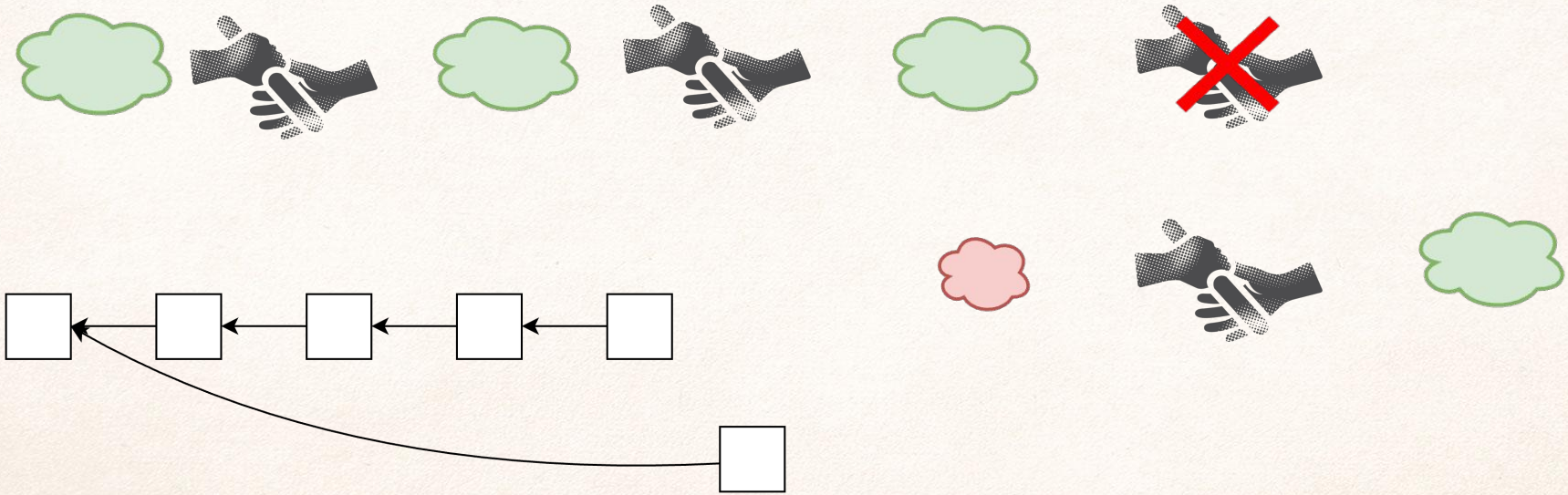
Same as before: if all honest votes from slot N are on a subtree, that subtree wins *in slot N+1 (and so on…)*



Goldfish: (lack of) asynchrony resilience



Goldfish: (lack of) asynchrony resilience



Goldfish: (lack of) asynchrony resilience

Not surprising:

- Dynamic availability requires making new decisions with whoever is online
- Asynchrony resilience requires holding on to previous decisions

Similar to the availability-finality dilemma, but we want *the available chain itself to have some resilience to asynchrony*

RLMD-GHOST (Recent Latest Message Driven - GHOST)

Family of protocols obtained by letting the vote expiry period $\eta \geq 1$ vary:

- At slot t, use votes from slots $\geq t - \eta$
- Goldfish = RLMD-GHOST($\eta=1$) => previous slot votes
- LMD-GHOST = RLMD-GHOST($\eta=\infty$) => all votes

$\eta \uparrow$ => asynchrony resilience $\uparrow$, tolerance of fluctuating participation $\downarrow$

RLMD-GHOST (Recent Latest Message Driven - GHOST)

Family of protocols obtained by letting the vote expiry period $\eta \geq 1$ vary:

- At slot t, use votes from slots $\geq t - \eta$
- Goldfish = RLMD-GHOST($\eta=1$) => previous slot votes
- LMD-GHOST = RLMD-GHOST($\eta=\infty$) => all votes

$\eta \uparrow$ => asynchrony resilience $\uparrow$, tolerance of fluctuating participation $\downarrow$

Votes persist for η slots:

- Good: periods of asynchrony of up to η slots do not cause “full collapse”
- Bad: fluctuating participation within the last η slots can still cause trouble

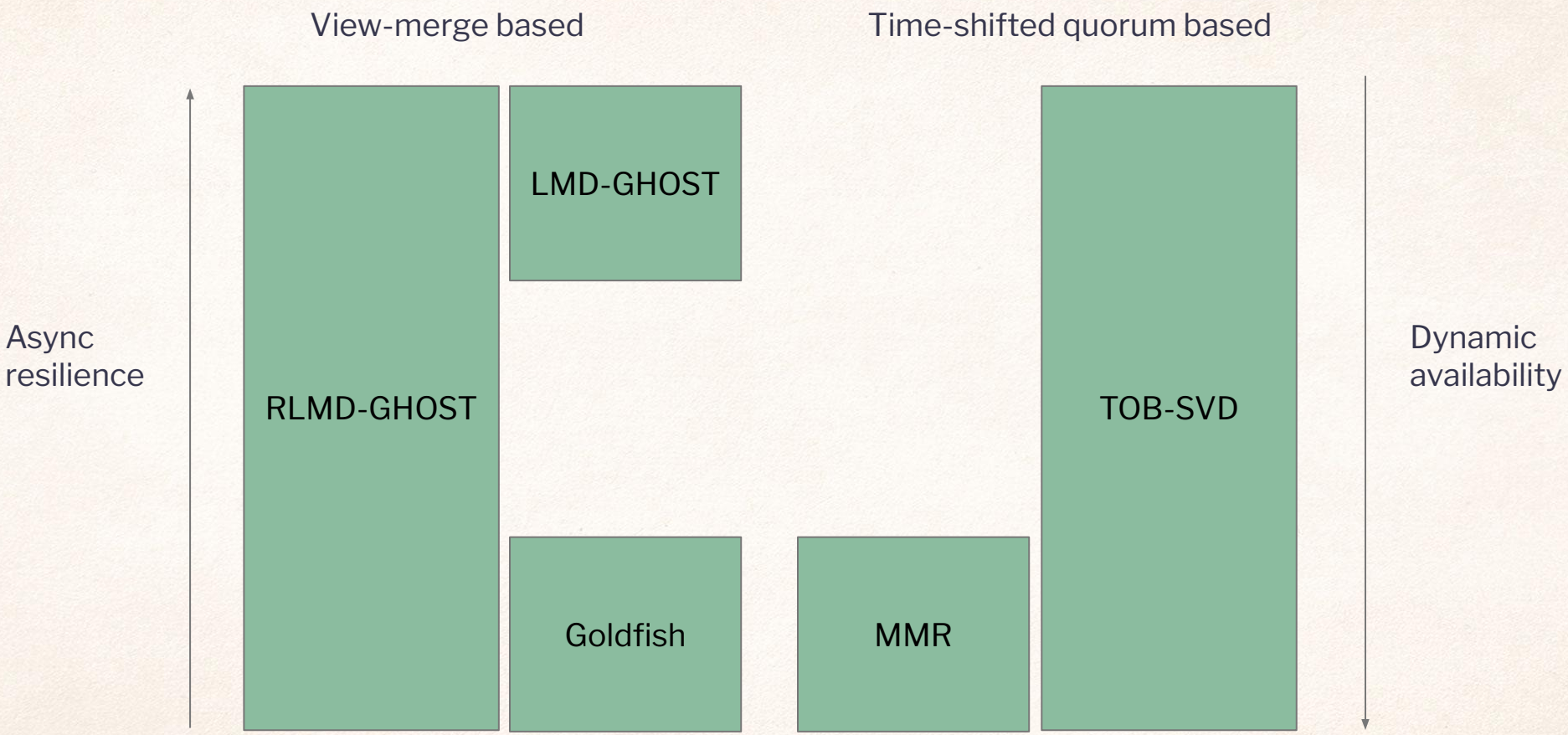
RLMD-GHOST (Recent Latest Message Driven - GHOST)

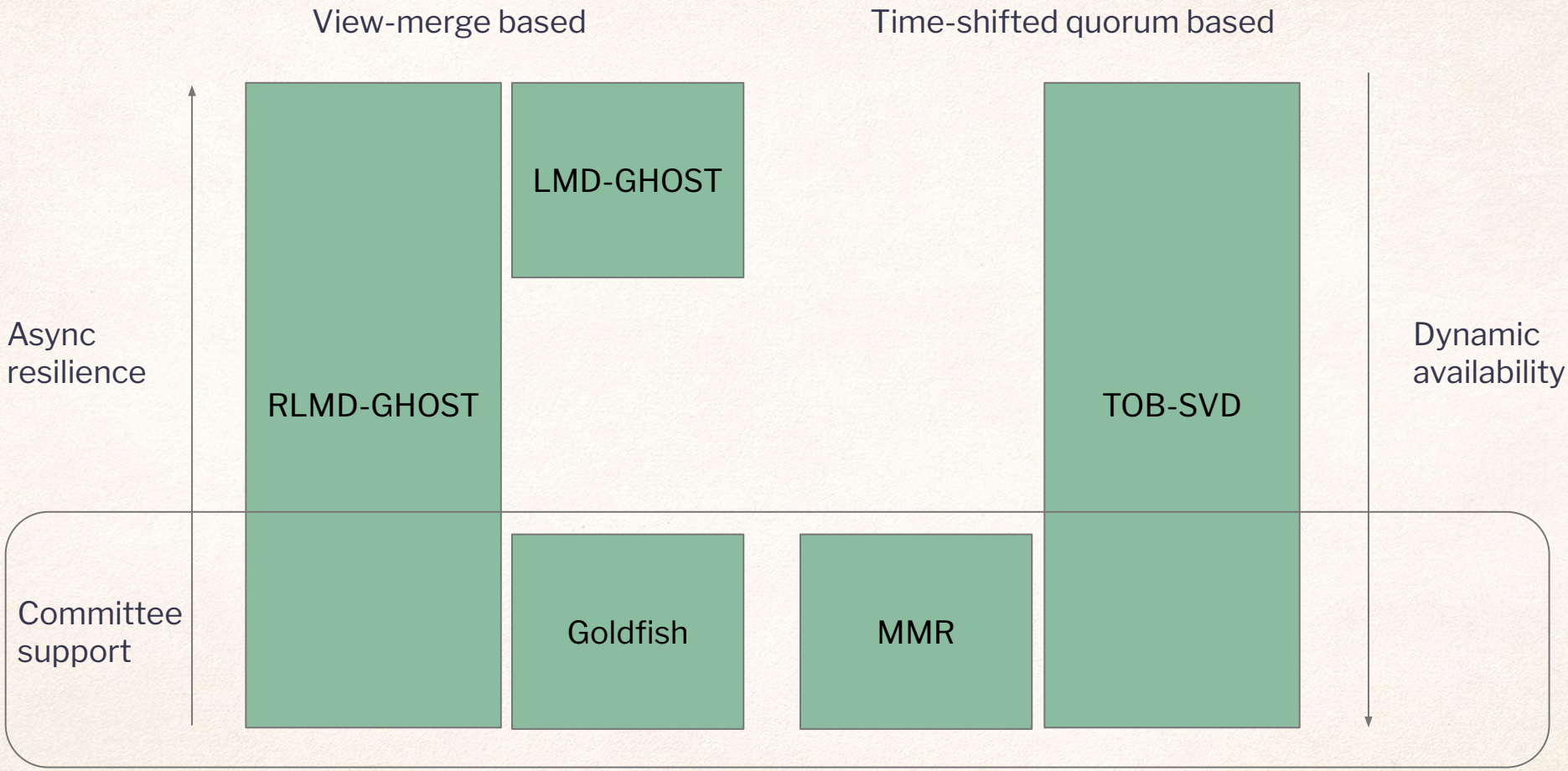
Family of protocols obtained by letting the vote expiry period $\eta \geq 1$ vary:

- At slot t, use votes from slots $\geq t - \eta$
- Goldfish = RLMD-GHOST($\eta=1$) => previous slot votes
- LMD-GHOST = RLMD-GHOST($\eta=\infty$) => all votes

$\eta \uparrow$ => asynchrony resilience $\uparrow$, tolerance of fluctuating participation $\downarrow$

- Goldfish has no asynchrony resilience, but tolerates fully fluctuating participation (sleepy model) *and committees*
- LMD-GHOST can even achieve asynchronous safety, but requires full participation


View-merge based
Time-shifted quorum based
Async resilience
Dynamic availability
RLMD-GHOST
LMD-GHOST
Goldfish
MMR
TOB-SVD


View-merge based
Time-shifted quorum based
Async resilience
Committee support
Dynamic availability
RLMD-GHOST
LMD-GHOST
Goldfish
MMR
TOB-SVD

Combining the two: committees + asynchrony resilience



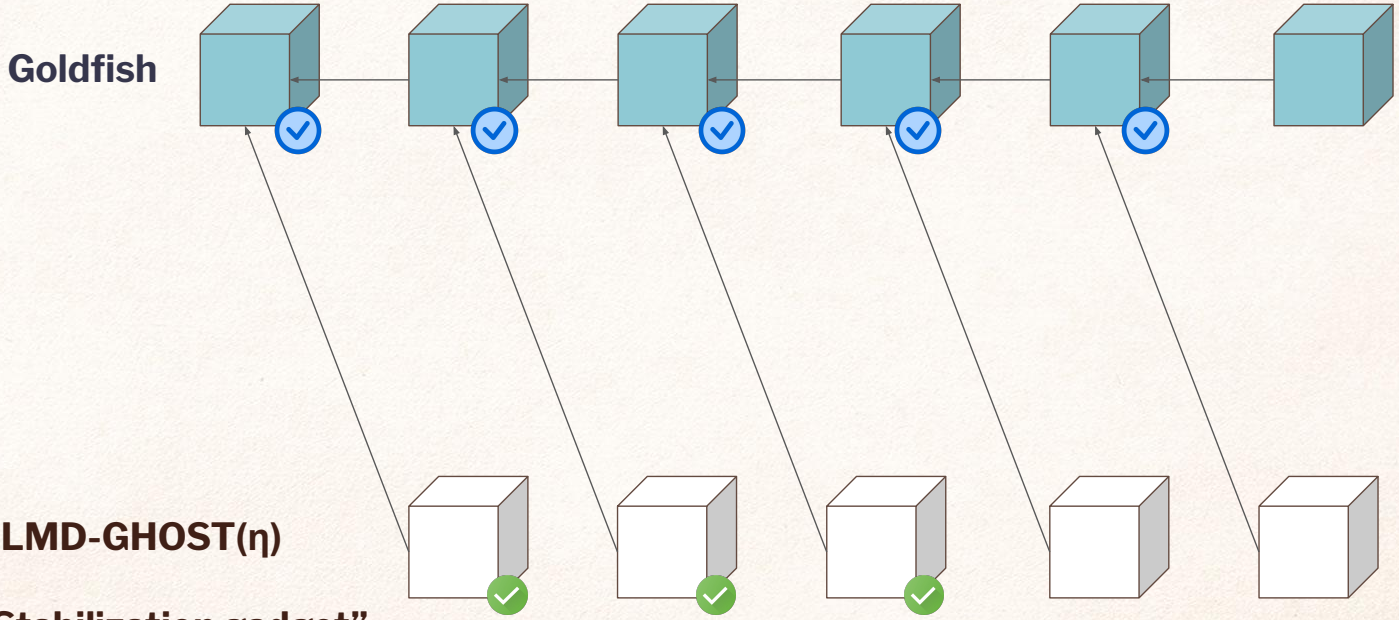
f < 50% of awake



Δ (Synchronous)

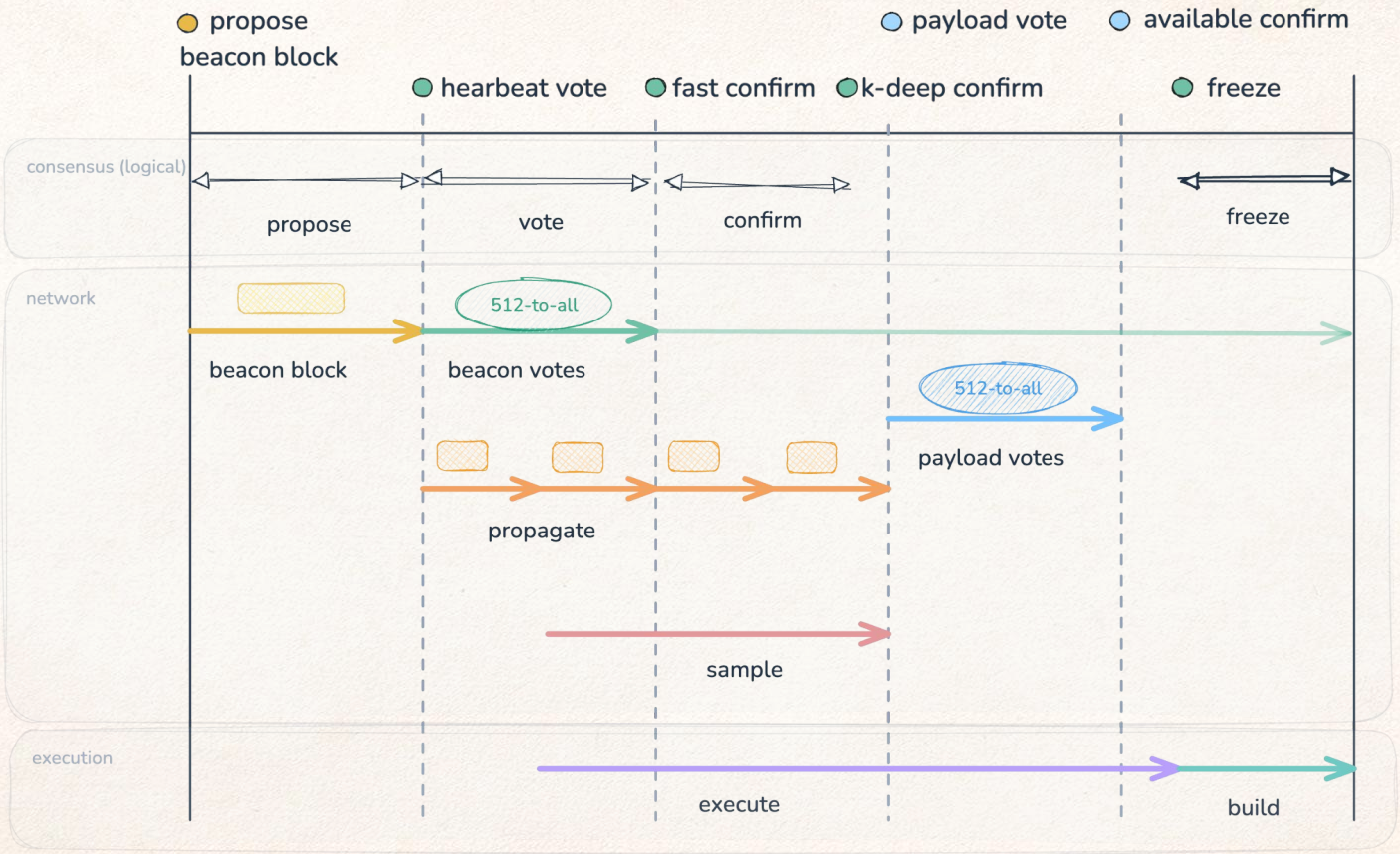


Honest majority of awake stake



Open Questions

- Can we have committee support (*subsampling*) and asynchrony resilience in a simpler way?
- Do we have to pay an additional latency cost for synchronization? (e.g. one more Δ for view-merge or time shifted quorum)
- Do we have to pay an additional latency cost for fast confirmation/constant latency?


propose
beacon block
payload vote
available confirm
hearbeat vote
fast confirm
k-deep confirm
freeze
consensus (logical)
propose
vote
confirm
freeze
network
512-to-all
beacon block
beacon votes
512-to-all
payload votes
propagate
sample
execution
execute
build



Ethereum Consensus:
The Finality Gadget

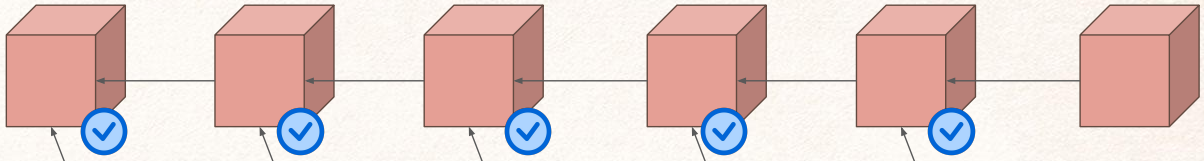
Francesco D'Amato
Ethereum Foundation

Available Chain

f < n / 2

Δ (Synchronous)

1 honest awake

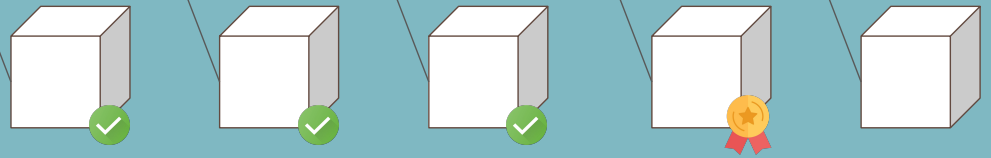


Finality Chain

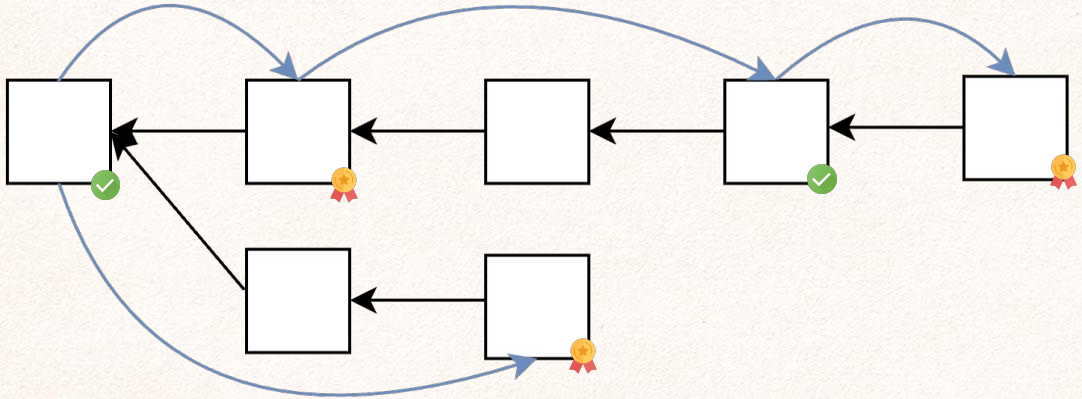
f < n / 3

Δ after **GST** (part. sync)

All online after **GAT**



Casper-FFG recap

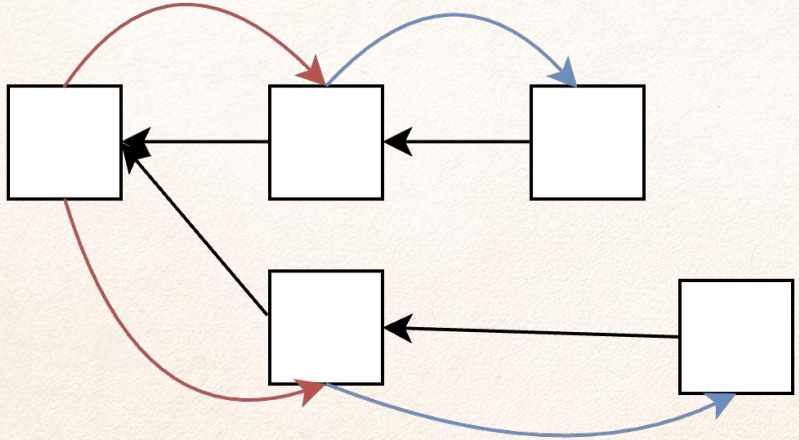


- **Checkpoint:** (Block, epoch) pair
- **FFG Votes:** Checkpoint -> Checkpoint links
- **Supermajority links:** ⅔ stake supports a link
- **Justification:** a chain of links from Genesis
- **Finality:** length 1 justification

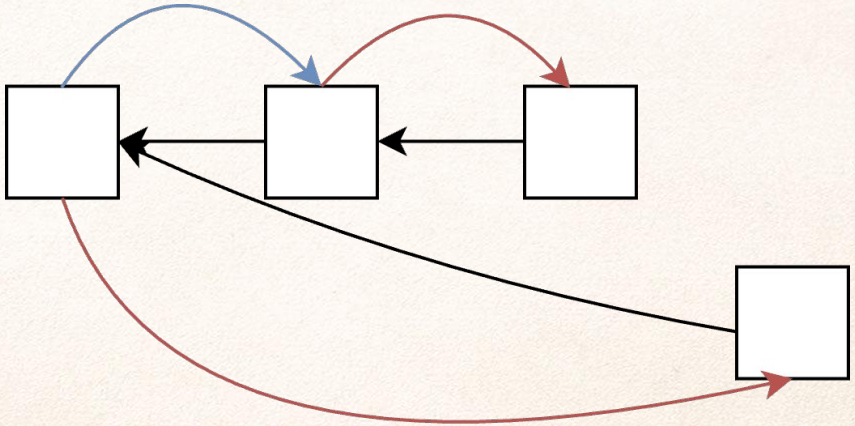
Slashing

Only ways to “bypass a finalization” and justify a conflicting higher epoch checkpoint

Equivocation
Violates the voting rule: only one vote per epoch (two targets, same epoch)



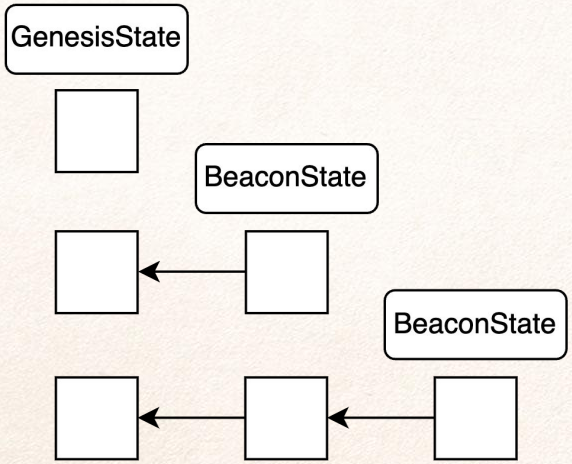
Surround voting
Violates the voting rule: your source (your lock) should only move forward



System-level perspective: the state

First building block is the state transition, which associates a state to every block

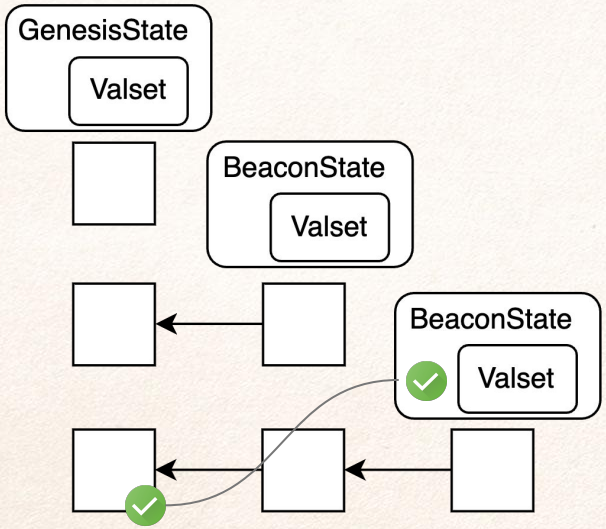
```
def state_transition(state: BeaconState, block: BeaconBlock) -> None
```



System-level perspective: the state

First building block is the state transition, which associates a state to every block

```
def state_transition(state: BeaconState, block: BeaconBlock) -> None
```



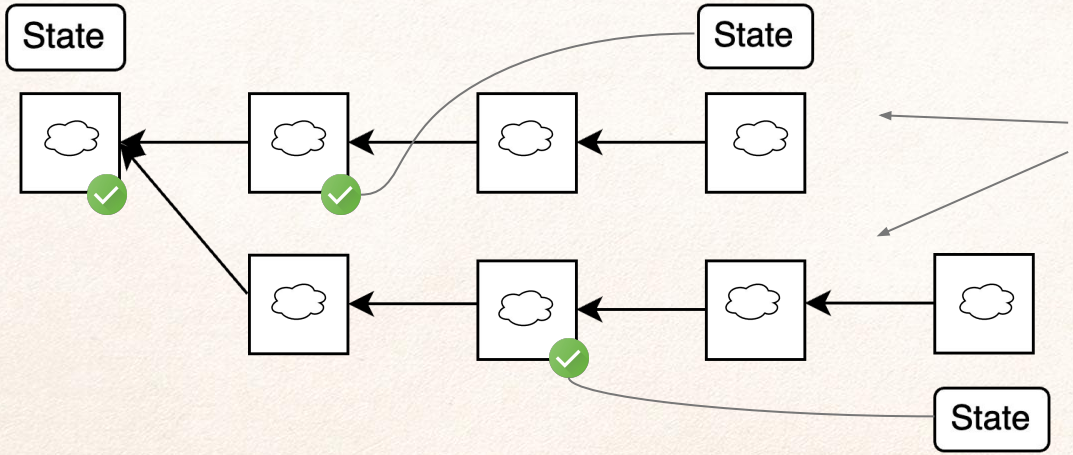
Handles:

- **achieving finality**: a block is finalized *if it finalized in some BeaconState (of another block)*
- **validator set changes**: no global validator set, there is only *the validator set of a given state,* constantly evolving
- **slashing**

Accountable safety

If two conflicting blocks are finalized, we have evidence to slash ≥ 1/3 of the stake

Purely a property of the state transition function

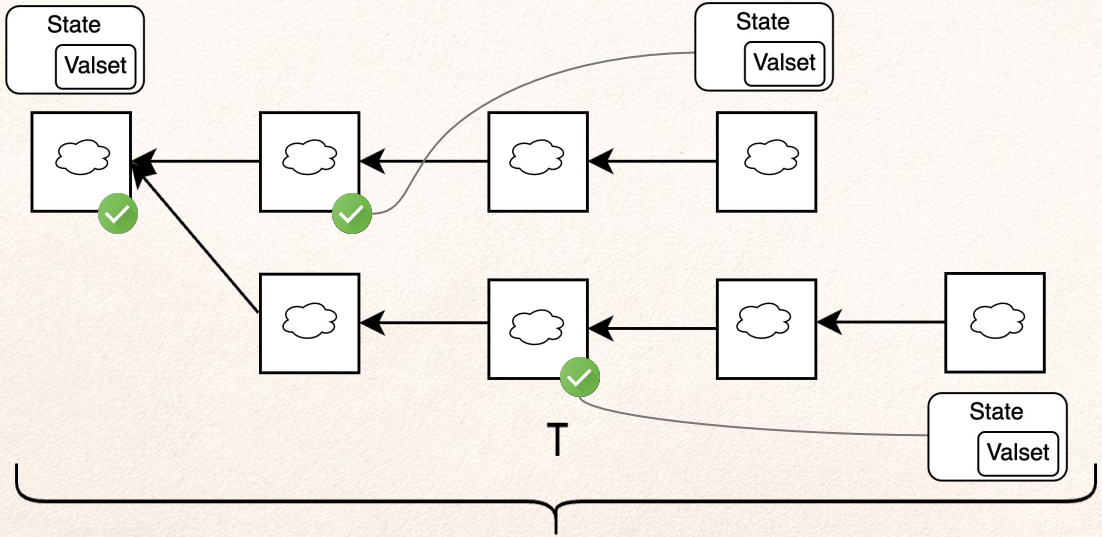


Slashing evidence is contained in the two chains

Weak subjectivity: accountable safety with a dynamic validator set

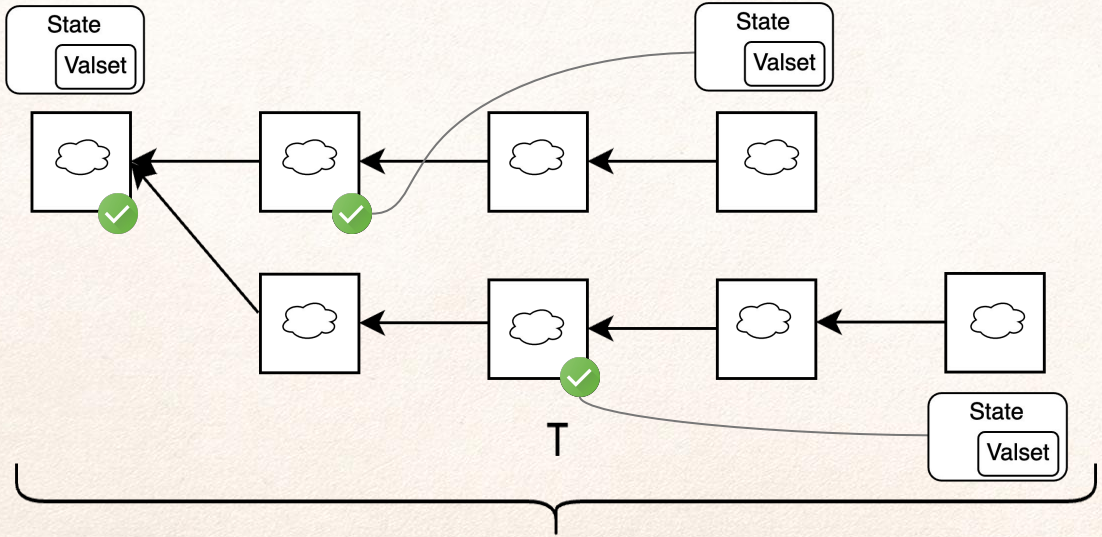
If two conflicting blocks are finalized within T epochs of their highest common ancestor B we have evidence to slash ≥ (1 - ∂•T) total_stake(state(B)) / 3

Safety decay ∂ depends on the max allowed rate of change of the validator set

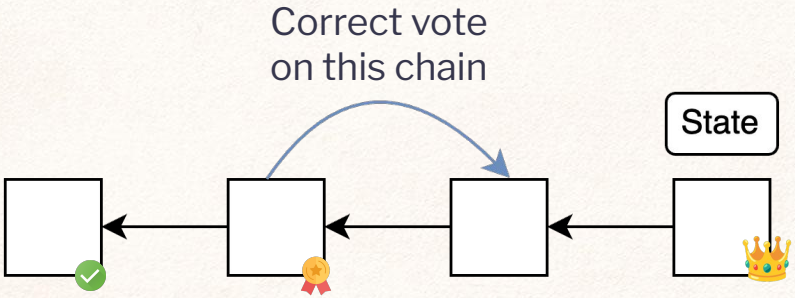


Weak subjectivity: accountable safety with a dynamic validator set

∂ defines the **weak subjectivity period (WSP):** the time it takes for the slashing guarantee (quorum intersection) to drop by 0.1



Inactivity leak (“accountable liveness”)



If the chain is not finalizing, we punish validators that are not participating

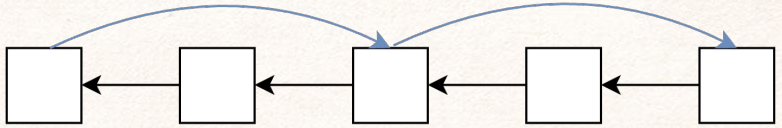
Also part of the state transition:

- On a given chain in a given epoch, there’s a unique correct FFG vote:
  - Source = latest justified
  - target = epoch boundary block
- Voting correctly exempts you from the leak

Inactivity leak (“accountable liveness”)

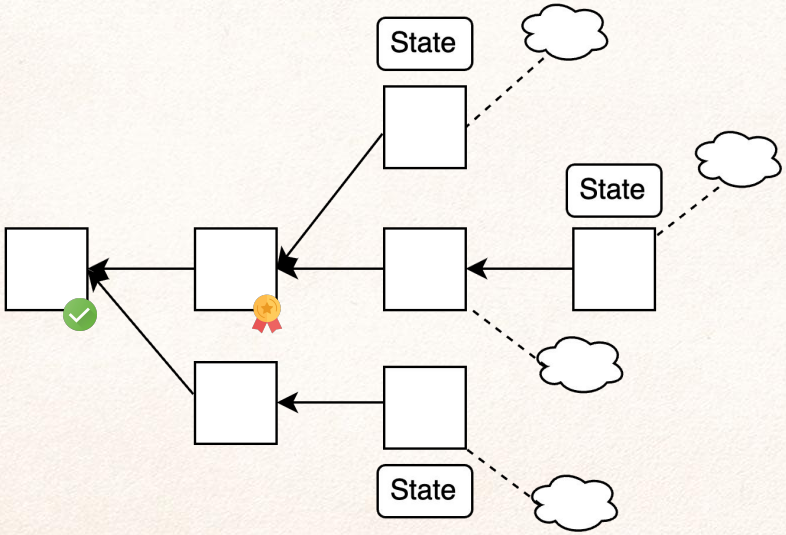
Leak tightness

In every epoch of nonfinality in a given state, ≥ ⅓ of the stake is subject to leak



(Currently Ethereum only guarantees ≥ ⅙)

System-level perspective: the fork-choice store



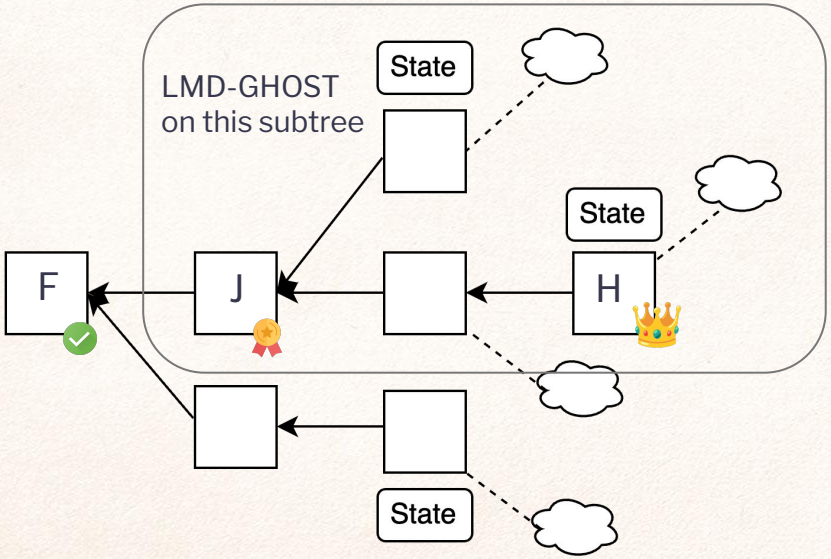
A node needs to:

- process new blocks and votes
- run the fork-choice to determine the head of the chain

To do so, it keeps:

- A block tree
- Associated states, and the finality gadget information they carry
- latest votes for LMD-GHOST

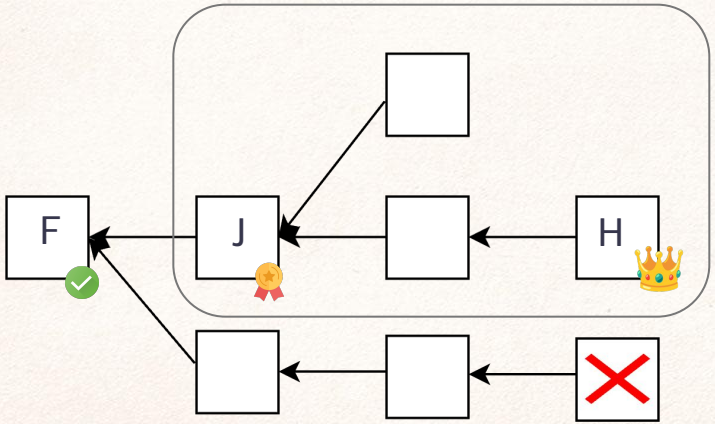
System-level perspective: the fork-choice store



F = latest finalized checkpoint
J = latest justified checkpoint
H = head of the chain

We enforce J < H by running LMD-GHOST from J

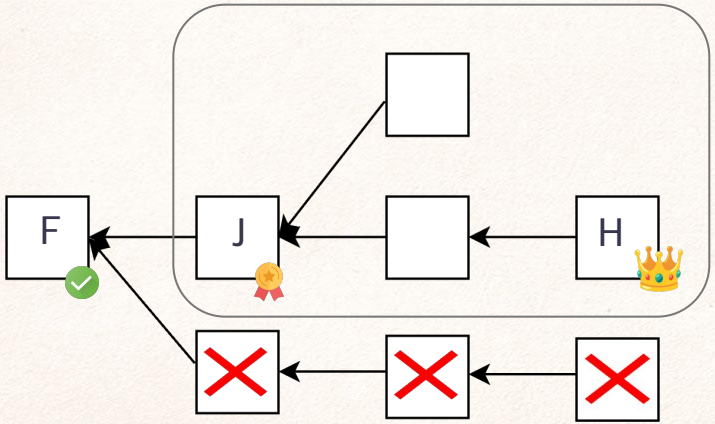
System-level perspective: the fork-choice store



Finalizations are treated as irreversible locally:

- New blocks conflicting with a finalization are not accepted

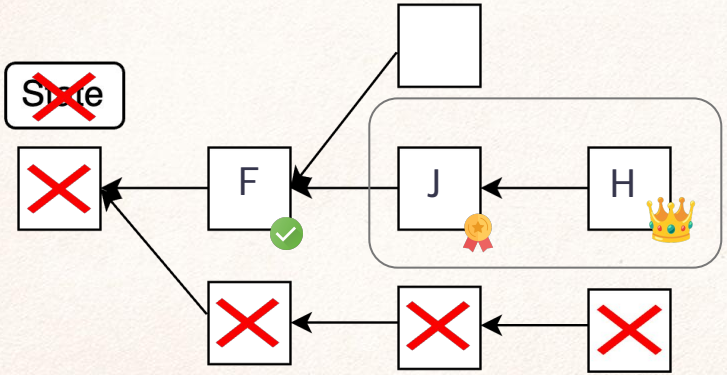
System-level perspective: the fork-choice store



Finalizations are treated as irreversible locally:

- New blocks conflicting with a finalization are not accepted
- Even existing conflicting blocks can be pruned

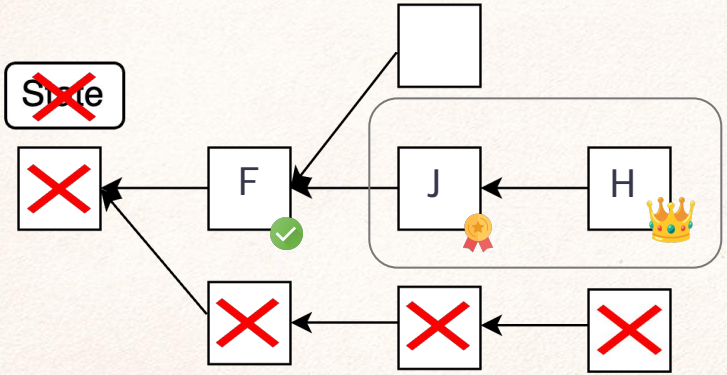
System-level perspective: the fork-choice store



Finalizations are treated as irreversible locally:

- New blocks conflicting with a finalization are not accepted
- Even existing conflicting blocks can be pruned
- Same for all ancestors **and their states**

System-level perspective: the fork-choice store

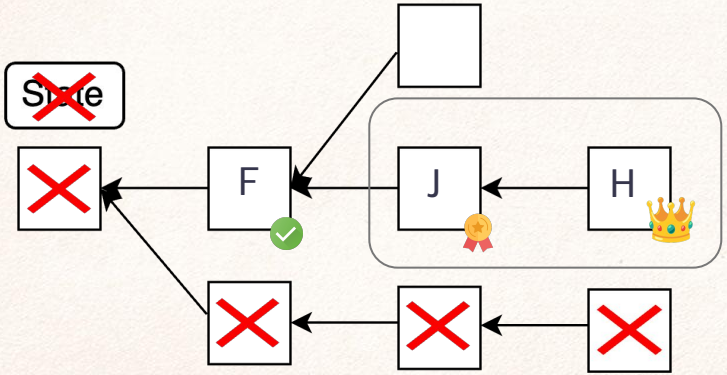


Finalizations are treated as irreversible locally:

- New blocks conflicting with a finalization are not accepted
- Even existing conflicting blocks can be pruned
- Same for all ancestors **and their states**

Lack of pruning is a large reason why nonfinality is problematic for clients

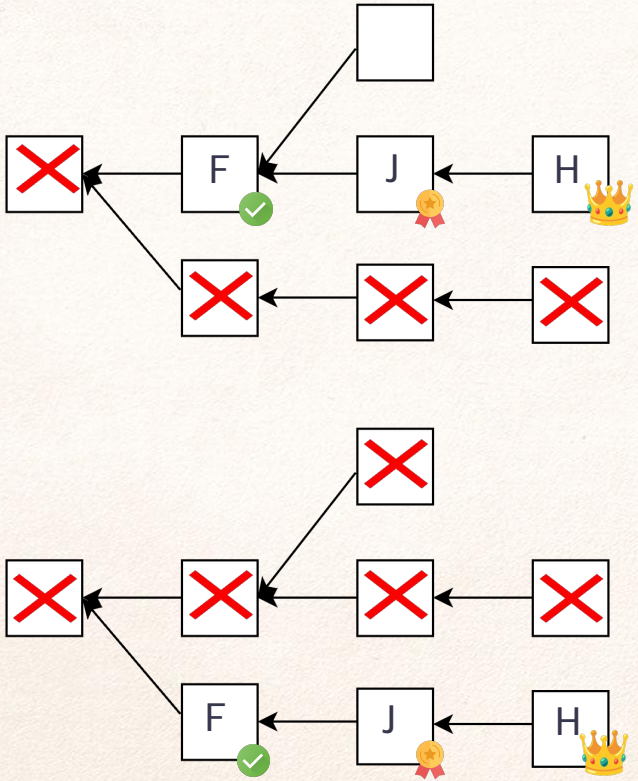
System-level perspective: the fork-choice store



If most of the network processes a finalization, a conflicting finalized block released later has no effect, **even if the whole validator set is corrupted**

Safety of finality can be violated by the validator set, **but not once the whole network has locked on a finalization**

System-level perspective: the fork-choice store

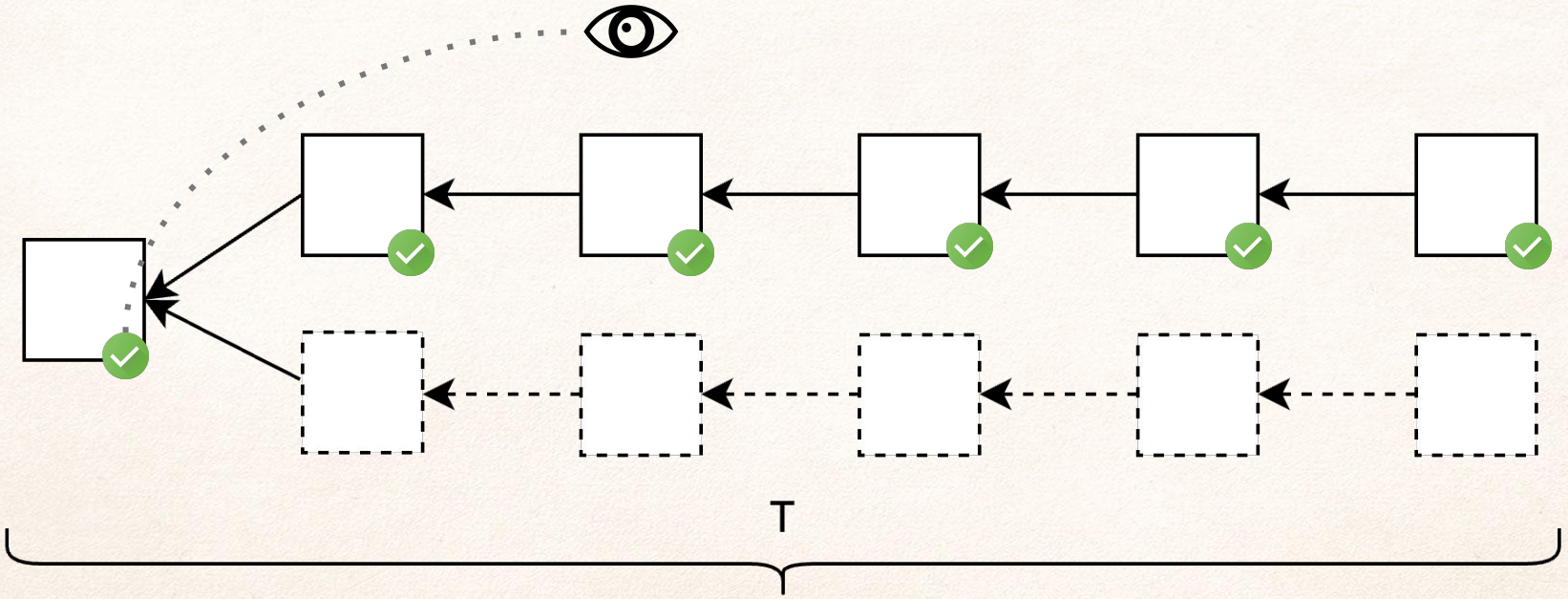


Other side of the coin: deadlocks

In case of conflicting finalizations, nodes can end up with incompatible views of the world and manual recovery is required

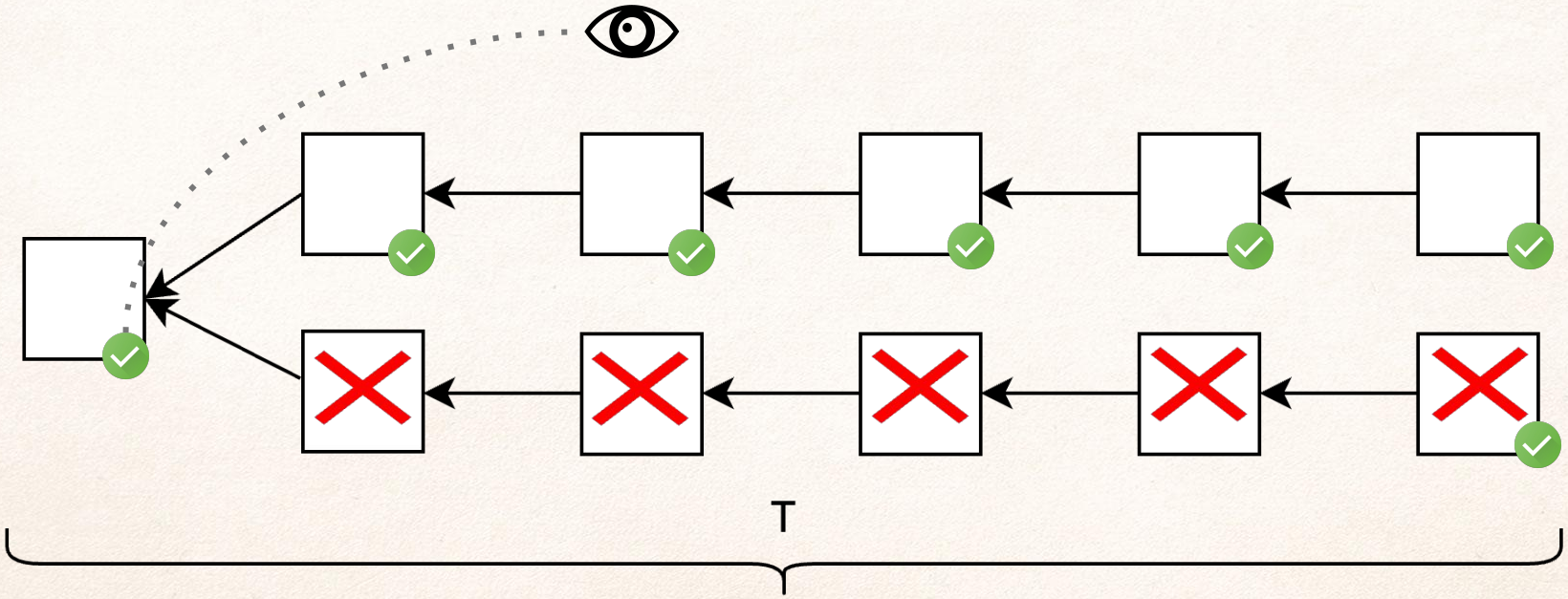
Weak subjectivity: accountable safety with a dynamic validator set

Building a private chain to finalize a conflicting checkpoint after safety decay is not effective



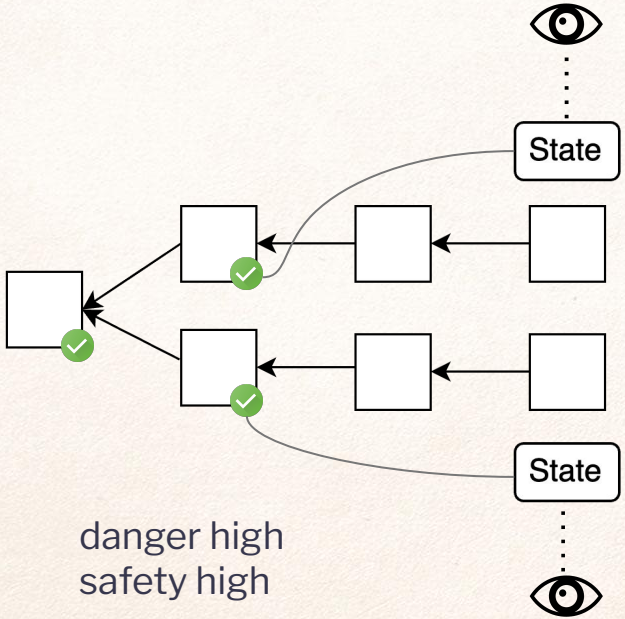
Weak subjectivity: accountable safety with a dynamic validator set

Building a private chain to finalize a conflicting checkpoint after safety decay is not effective

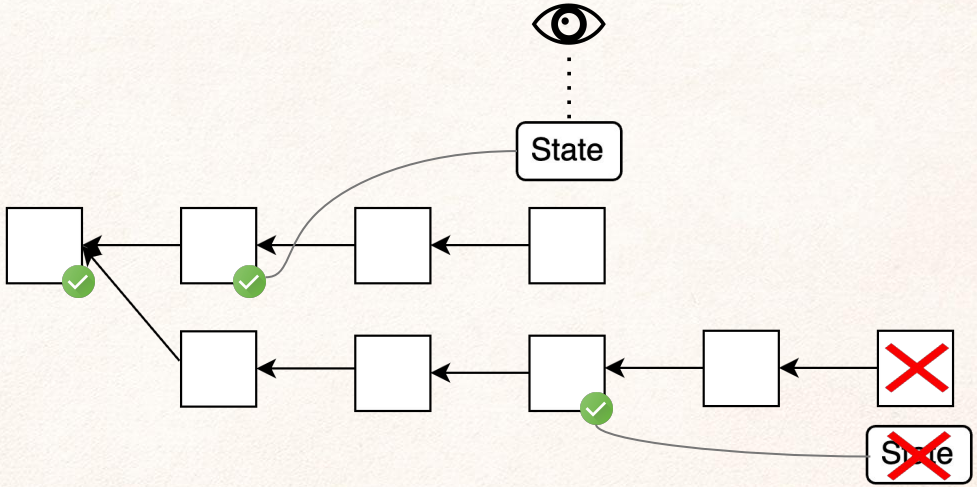


Weak subjectivity: accountable safety with a dynamic validator set

Danger of conflicting finalizations decays if they are revealed far apart from each other. If nodes know about the first finalization, they won't accept the second.

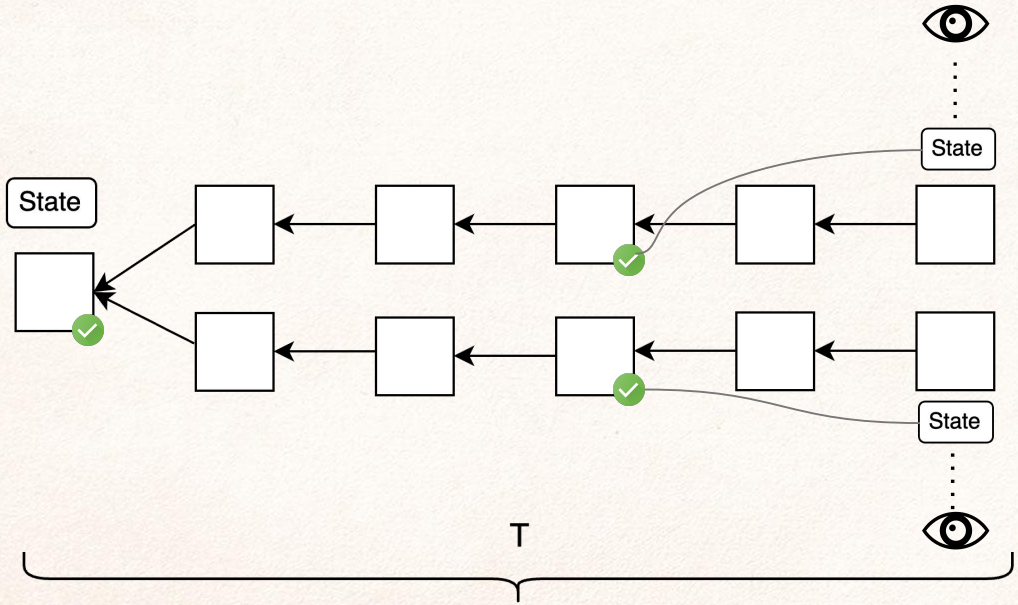


danger high
safety high



danger low
safety low

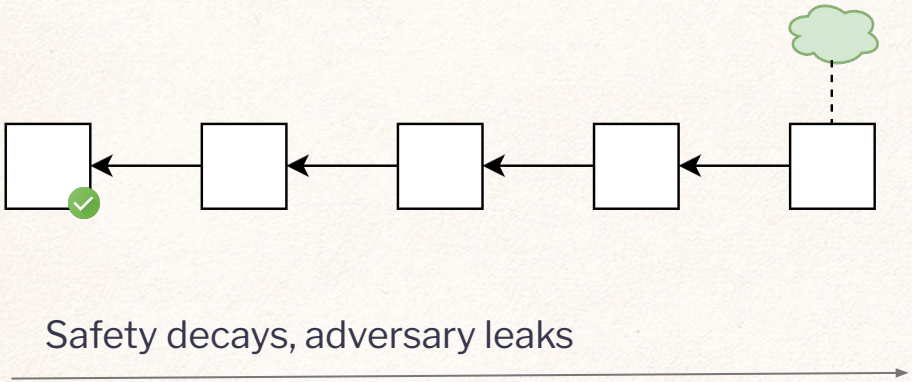
Weak subjectivity: accountable safety with a dynamic validator set



However the safety baseline continuously decays when we are not finalizing

The inactivity leak makes waiting for safety to decay costly

Inactivity leak



(1) Honest majority, synchrony
(2) Secure Ebb-and-flow protocol

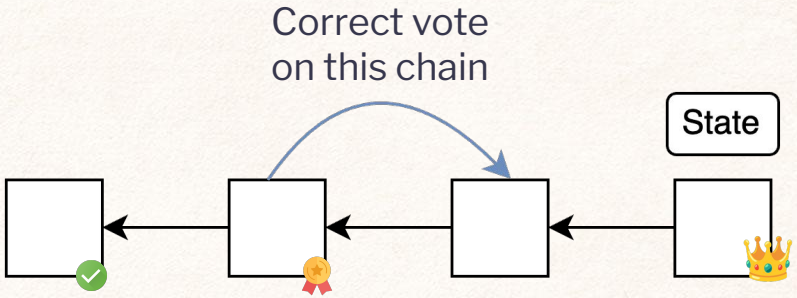
1 + 2 => safe and live available chain, but finality can stall if adversary does not cooperate.

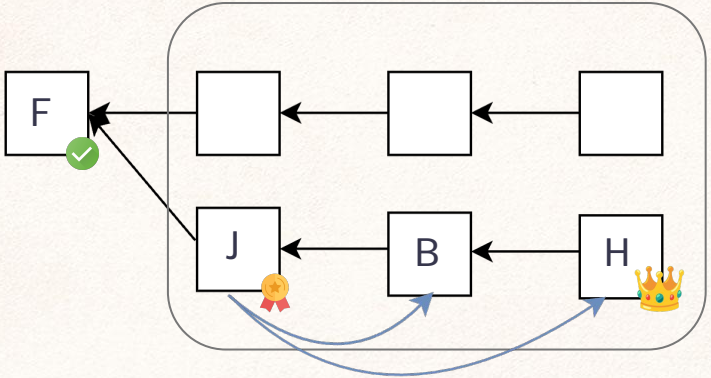
If so, the adversary leaks **on the canonical chain**

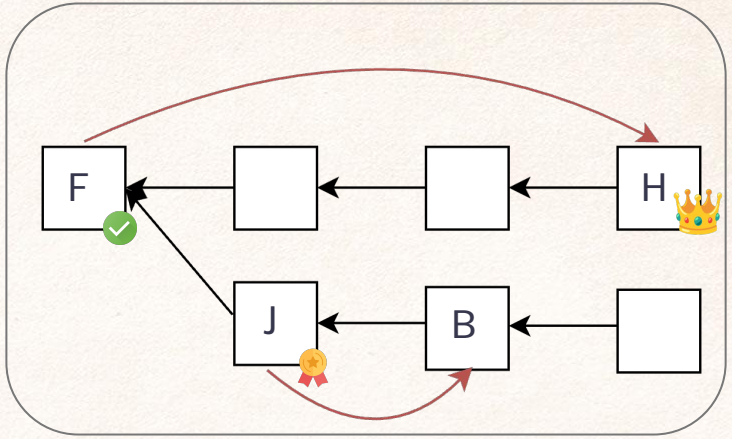
Inactivity leak

Leak fairness

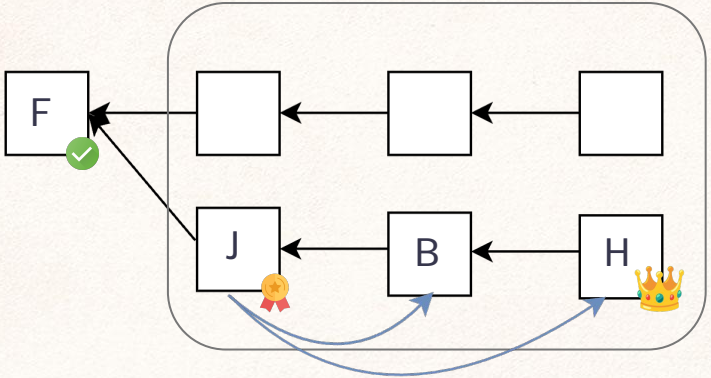
If the canonical chain is sufficiently stable, a validator that follows the voting rules and whose votes is quickly included onchain does not get leaked.

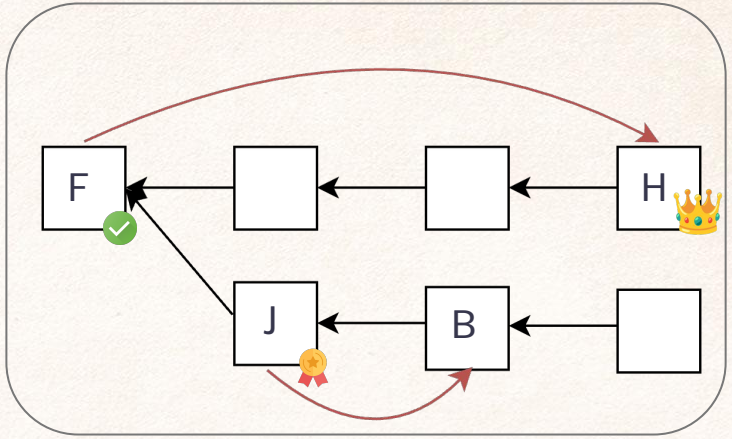






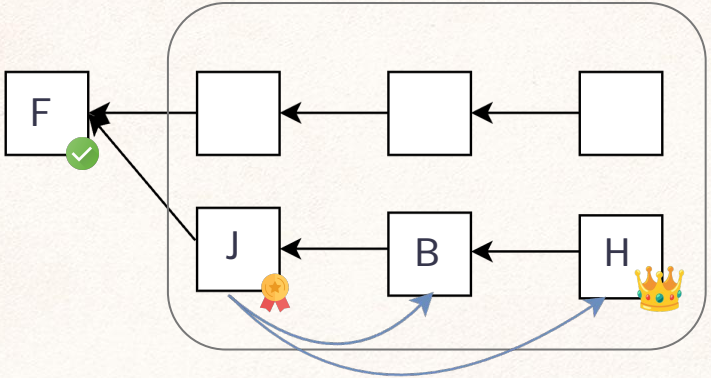
Leak fairness is part of why we use J. If you are locked on J (voted J -> B):

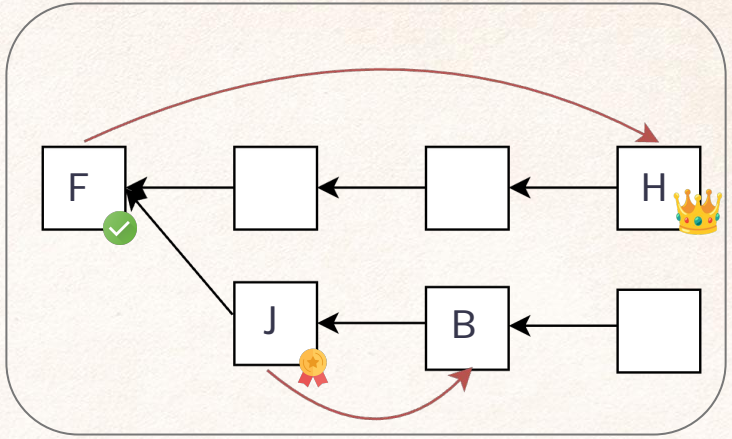




Leak fairness is part of why we use J. If you are locked on J (voted J -> B):

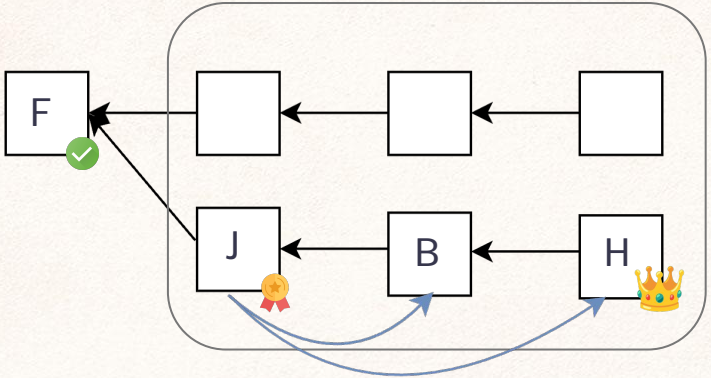
- Fork-choice anchored in J: can vote J -> H and not get leaked *on the canonical chain*

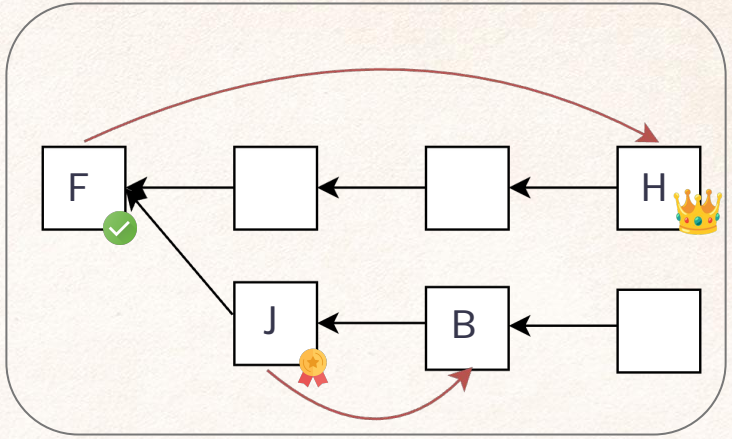




Leak fairness is part of why we use J. If you are locked on J (voted J -> B):

- Fork-choice anchored in J: can vote J -> H and not get leaked *on the canonical chain*
- Fork-choice anchored in F: voting F -> H is a surround vote, *can either get slashed or leaked on the canonical chain*



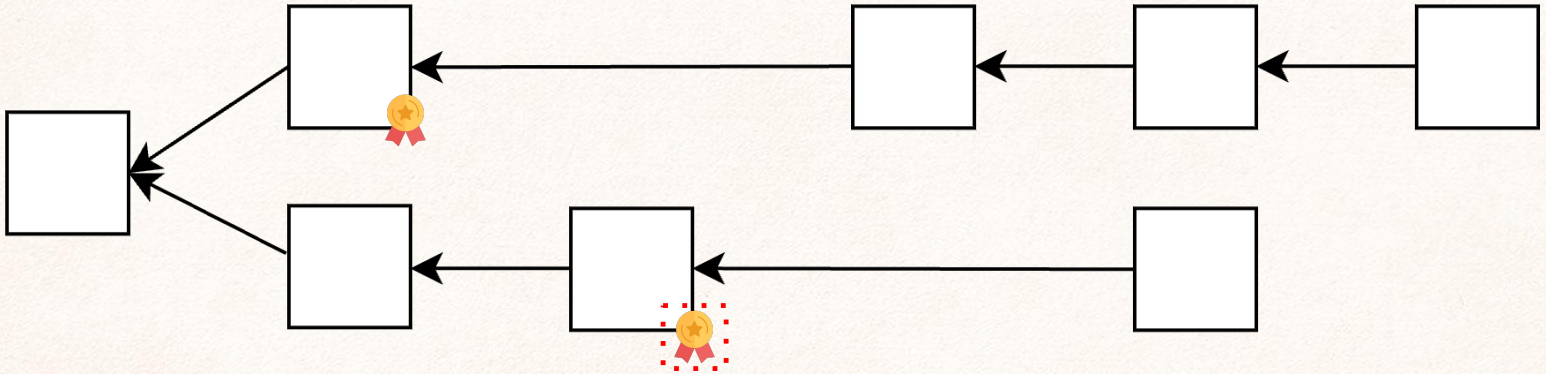


Leak fairness is part of why we use J. If you are locked on J (voted J -> B):

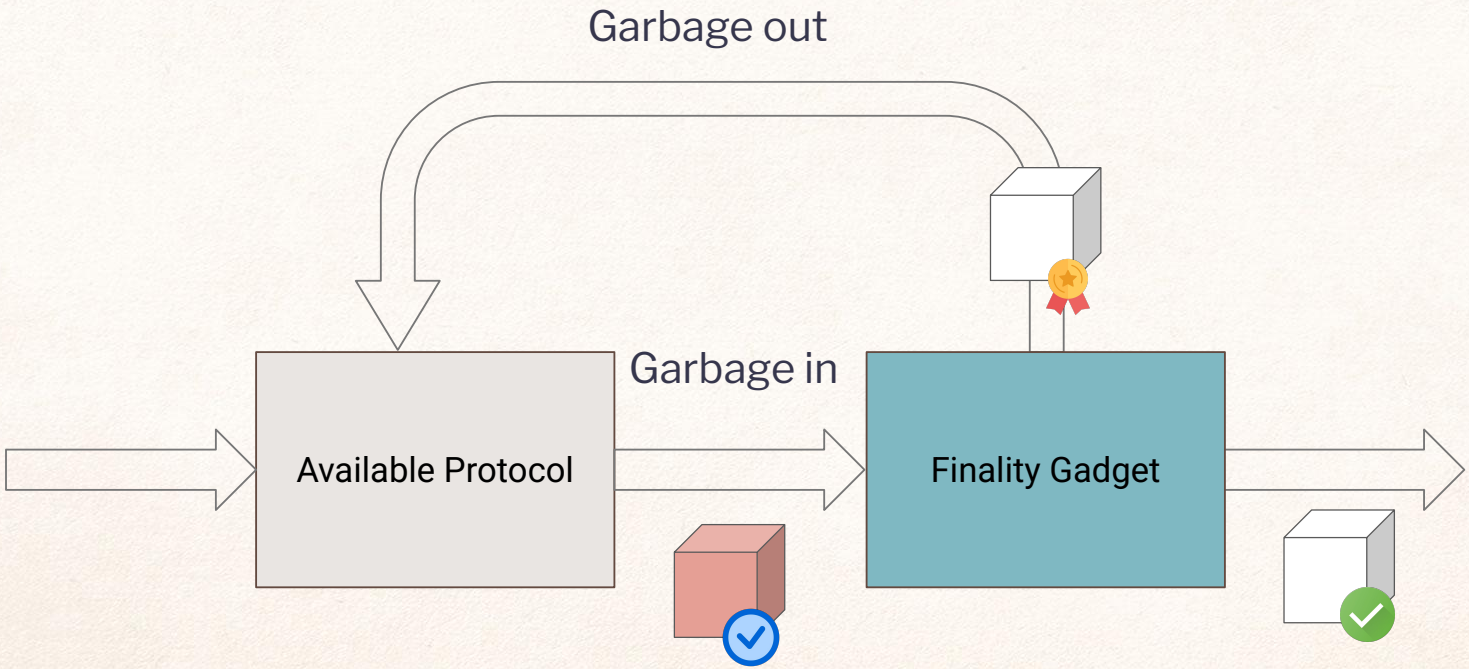
- Fork-choice anchored in J: can vote J -> H and not get leaked *on the canonical chain*
- Fork-choice anchored in F: voting F -> H is a surround vote, *can either get slashed or leaked on the canonical chain*

Generally interaction between FG and AC is a major source of problems and one of the main challenges in the design of the future consensus protocol.

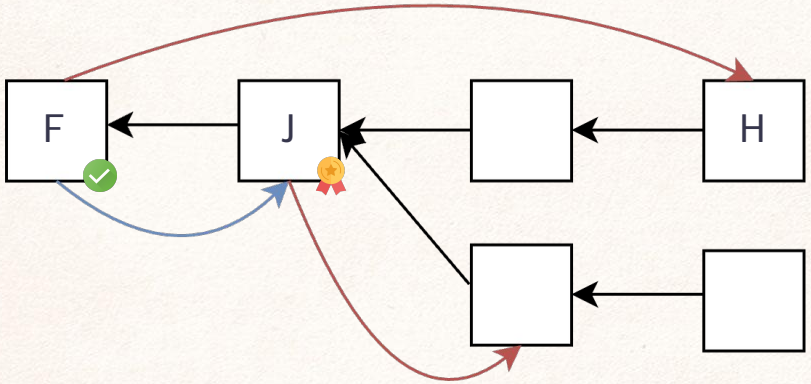
Interaction: healing after asynchrony



Interaction: healing after asynchrony



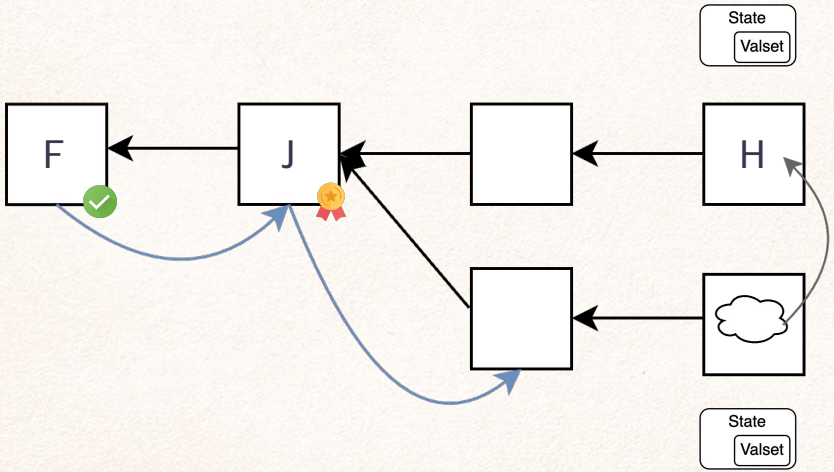
Inactivity leak



Even when using J as fork-choice anchor:

- **Canonical chain might not know about the latest justification: the latest justified for H is F**
- If you're locked on J, you're unable to vote correctly on the canonical chain: again, either get slashed or leaked

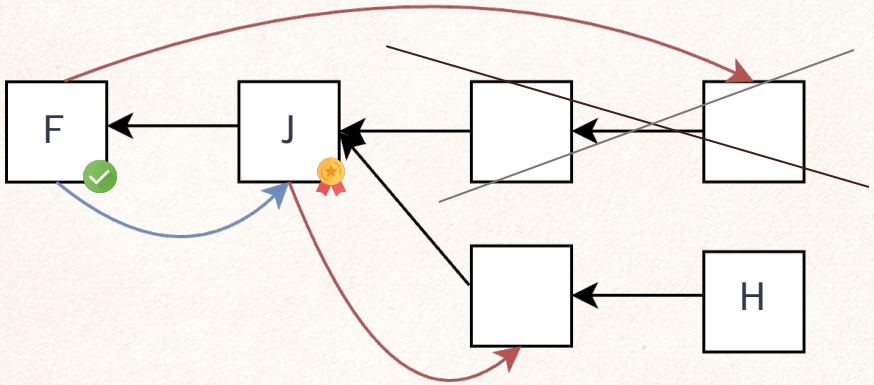
Inactivity leak



Should be fixed by transferring the “justification certificate” from one chain to the other, but not so simple:

1. Different validator sets, so votes don’t count the same
2. We could fix a validator set snapshot to use, only dependent on (source, target), but during a long period of nonfinality we’d have to hold on to many of these

Inactivity leak

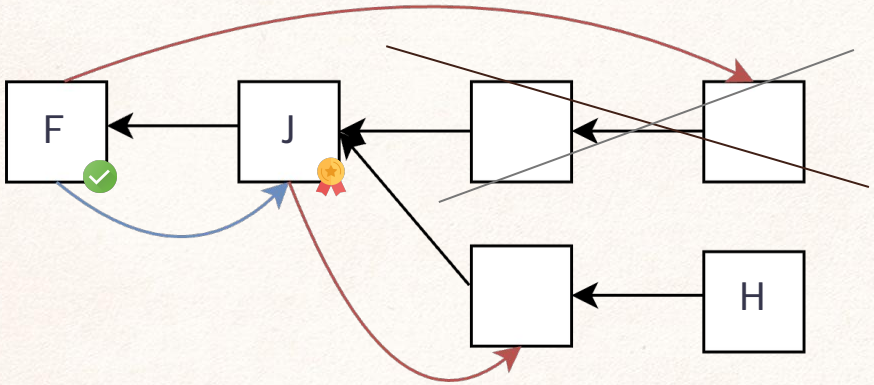


Even when using J as fork-choice anchor:

- **Canonical chain might not know about the latest justification: the latest justified for H is F**
- If you're locked on J, you're unable to vote correctly on the canonical chain: either get slashed or leaked

In Gasper, resolved by **branch filtering:** only branches up to date with the latest justification are **viable.**

Inactivity leak



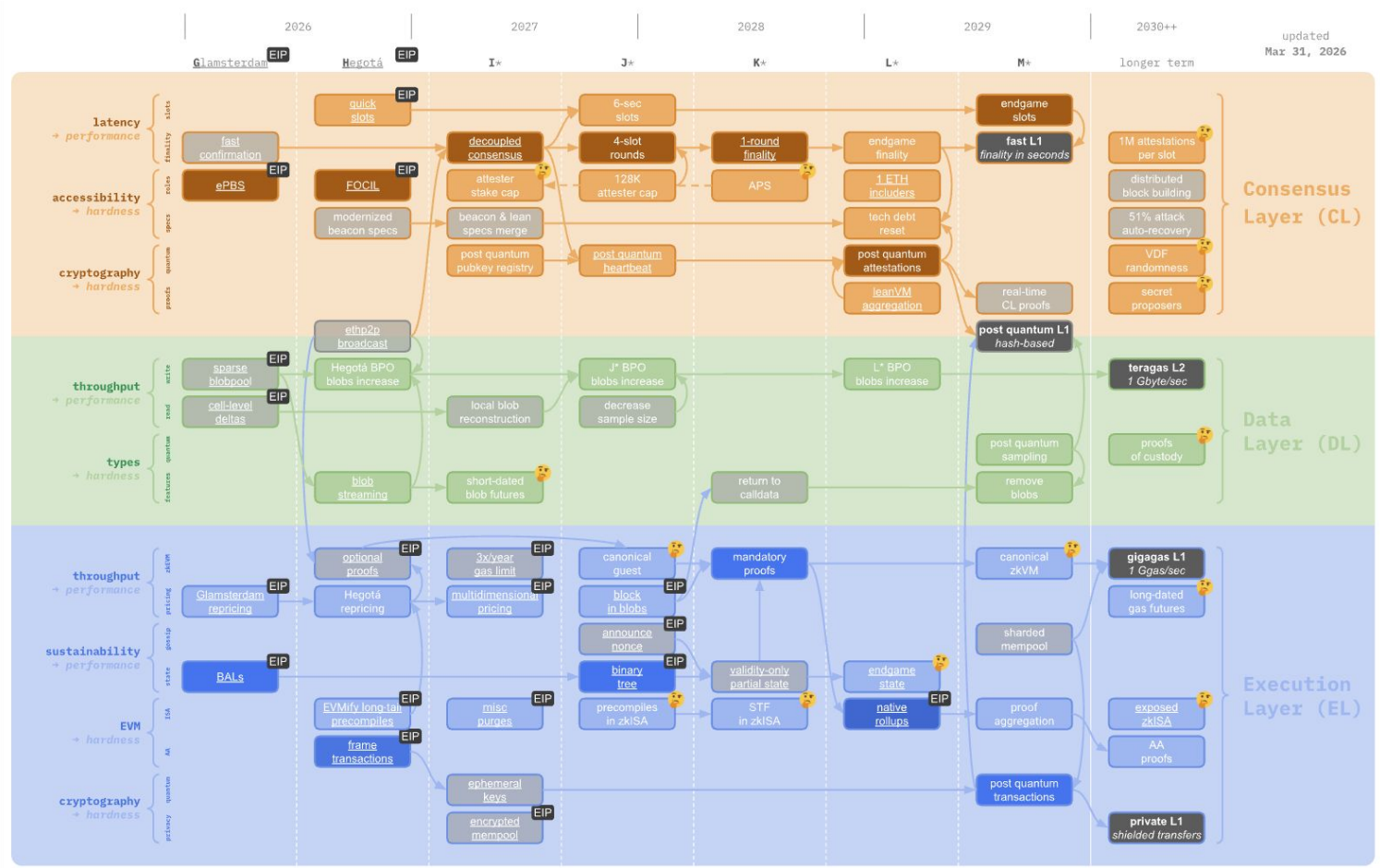
Even when using J as fork-choice anchor:

- **Canonical chain might not know about the latest justification: the latest justified for H is F**
- If you're locked on J, you're unable to vote correctly on the canonical chain: either get slashed or leaked

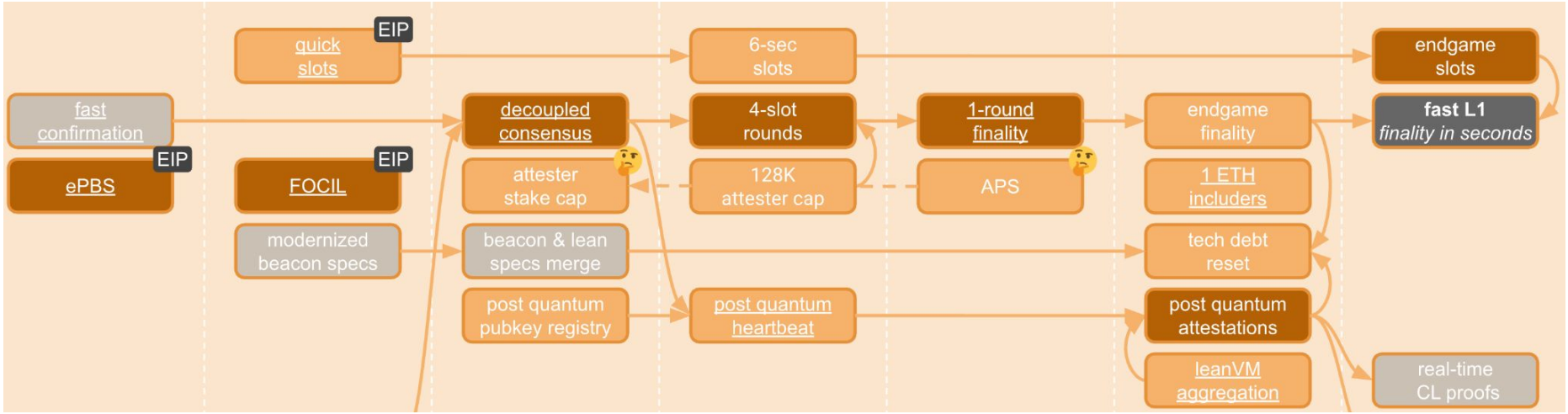
In Gasper, resolved by **branch filtering:** only branches up to date with the latest justification are **viable.**

Major source of issues in the interaction between LMD-GHOST and FFG, will be removed in the future.

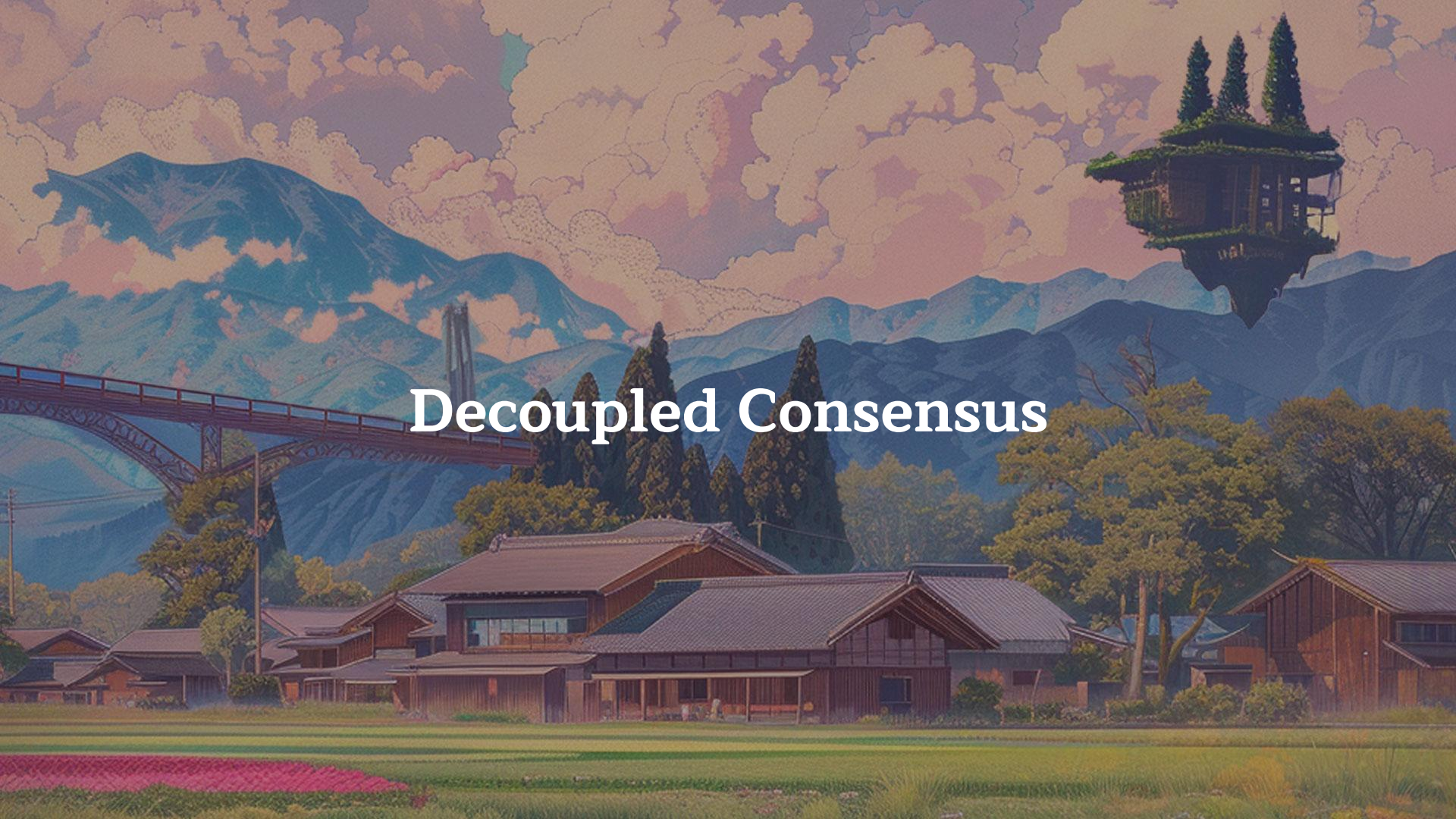

Future of Ethereum Consensus
Yann Vonlanthen
Ethereum Foundation



strawmap.org

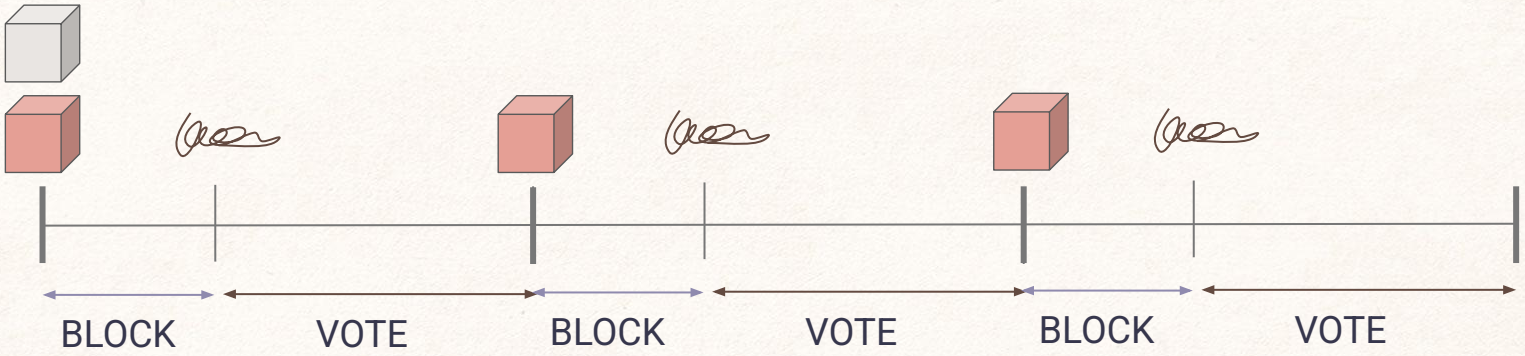


Consensus Layer


Decoupled Consensus

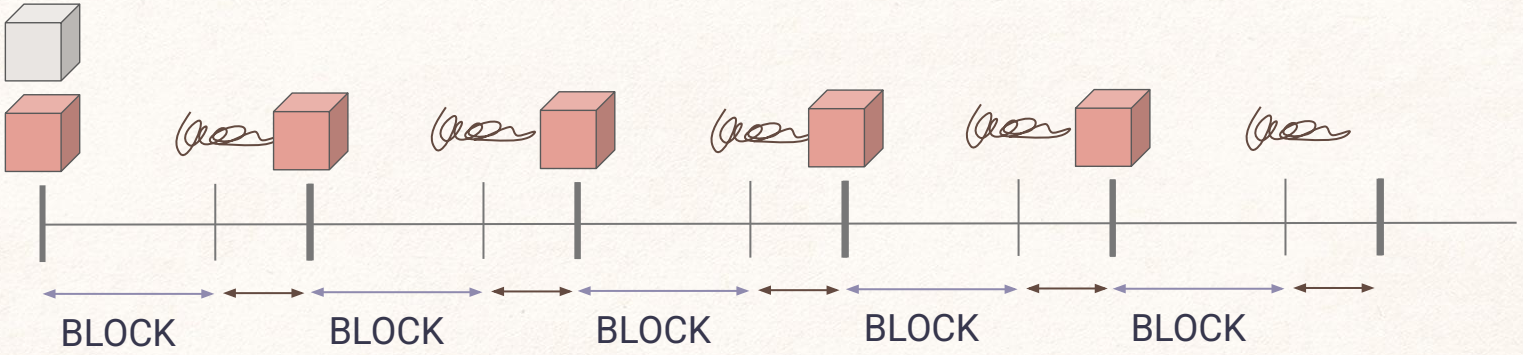
Coupled timelines in Gasper

1M Attestations
of ~100B over 32 slots!



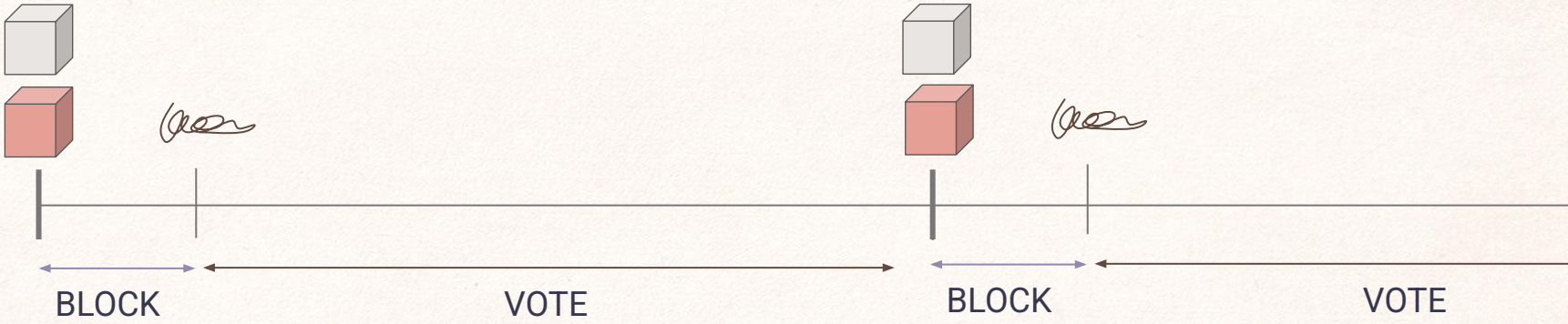
Coupled timelines in Gasper

Small committees
→ long epochs
→ slow finality



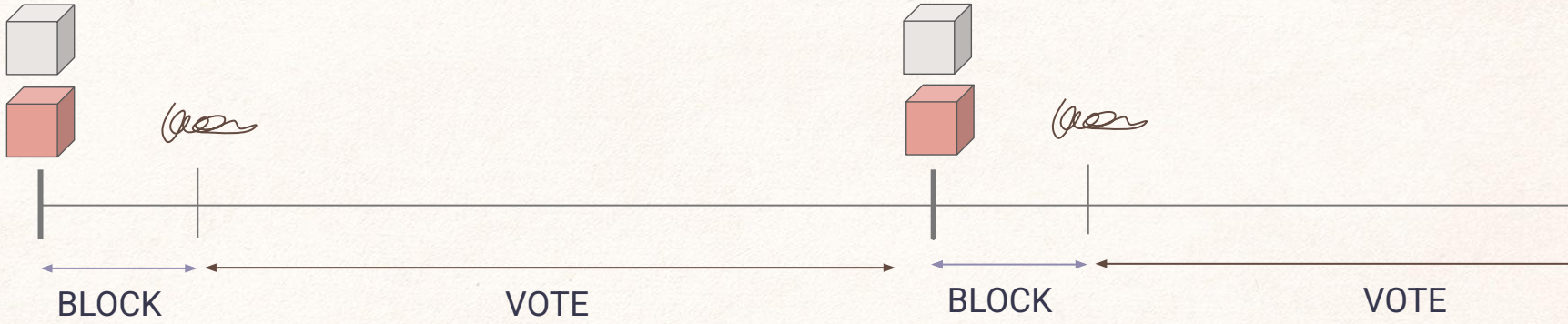
Coupled timelines in Gasper

Large committees
→ long slots
→ slow block production



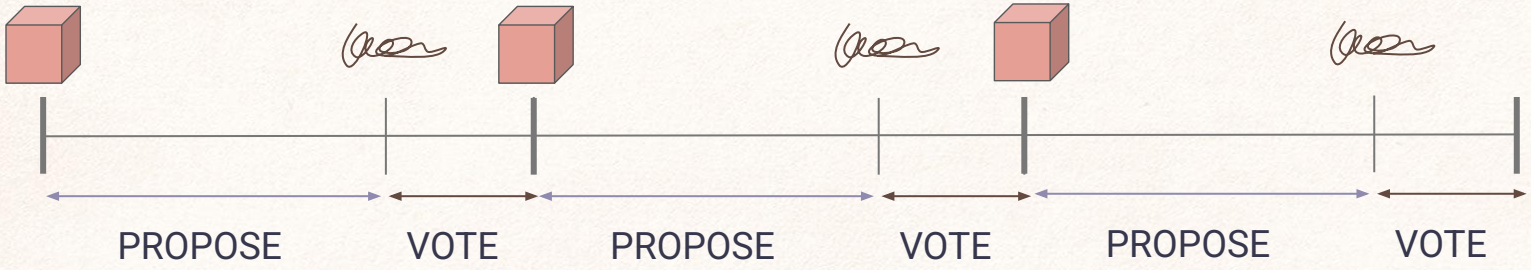
Coupled timelines in Gasper

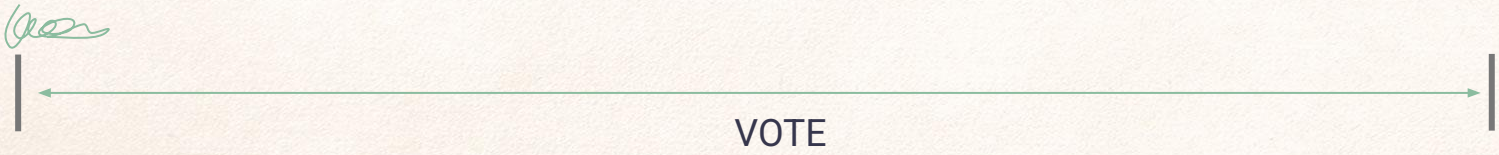
Large committees
→ long slots
→ slow block production

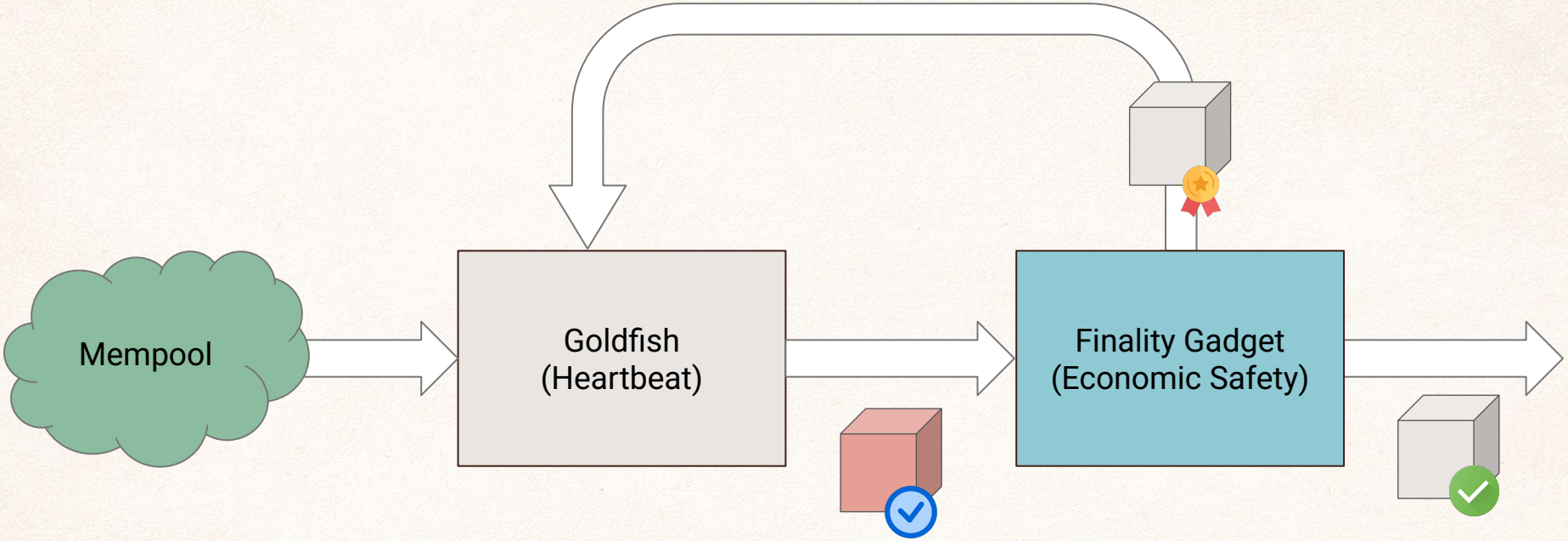


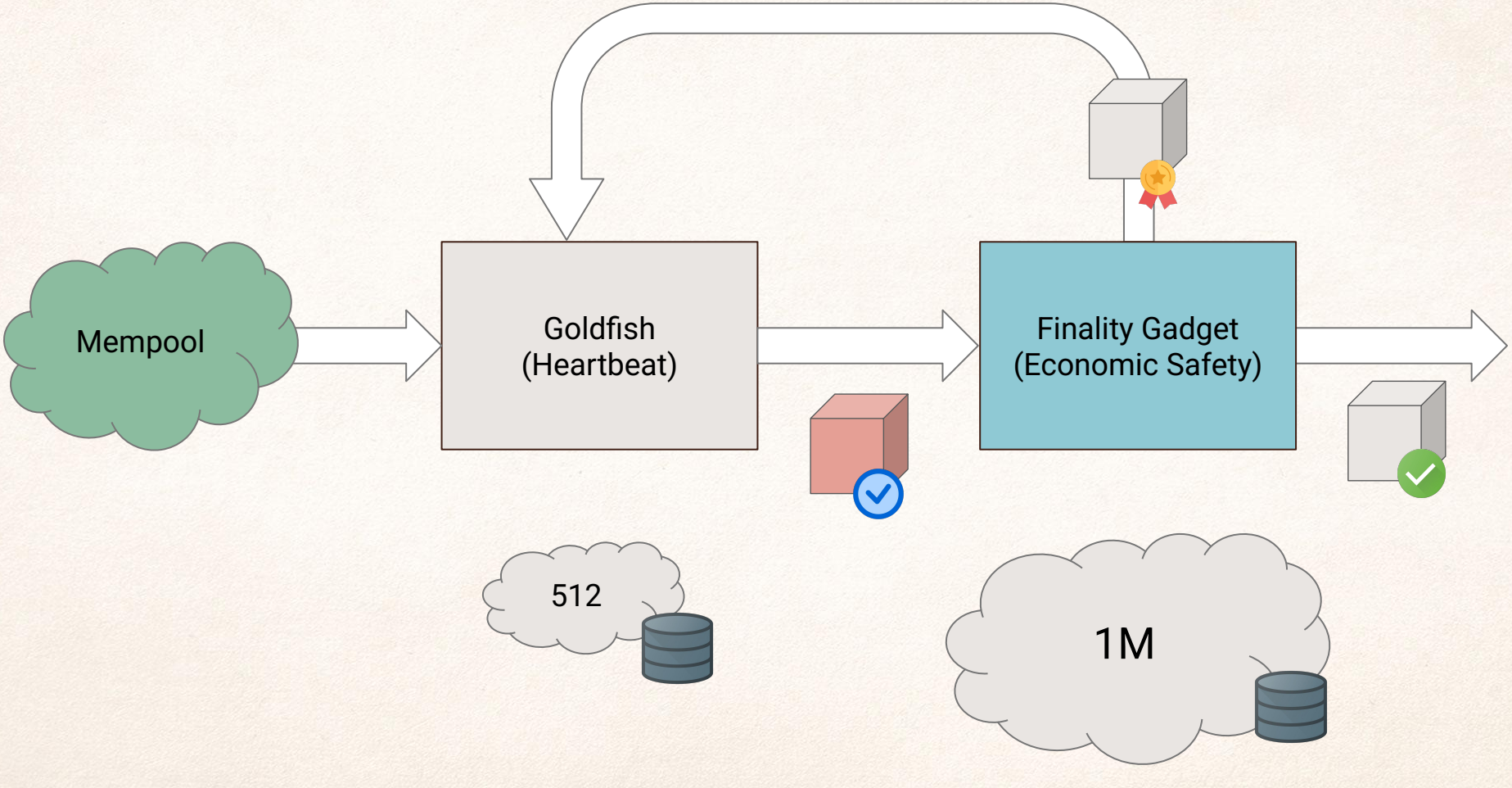
Block production and finality timeline are coupled!

Decoupled Consensus






Mempool
Goldfish
(Heartbeat)
Finality Gadget
(Economic Safety)


Mempool
Goldfish
(Heartbeat)
Finality Gadget
(Economic Safety)
512
1M

Goldfish



f < n / 2

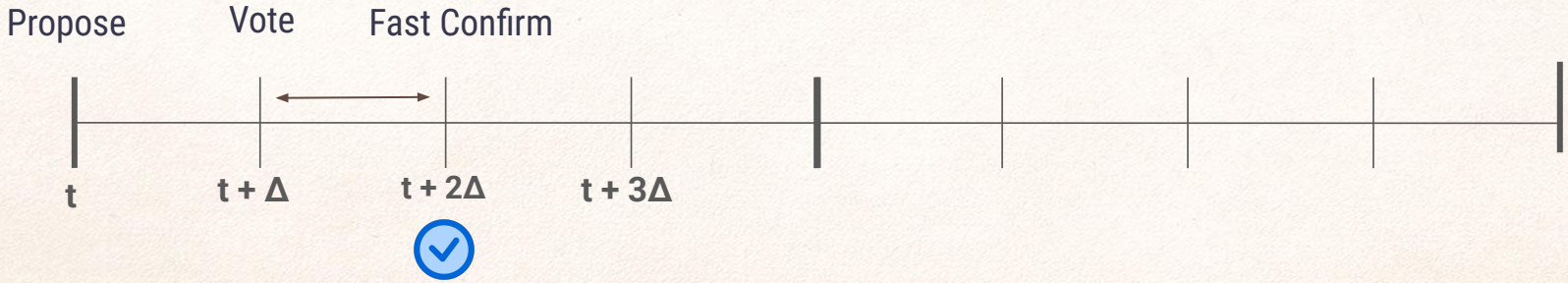


Δ (Synchronous)

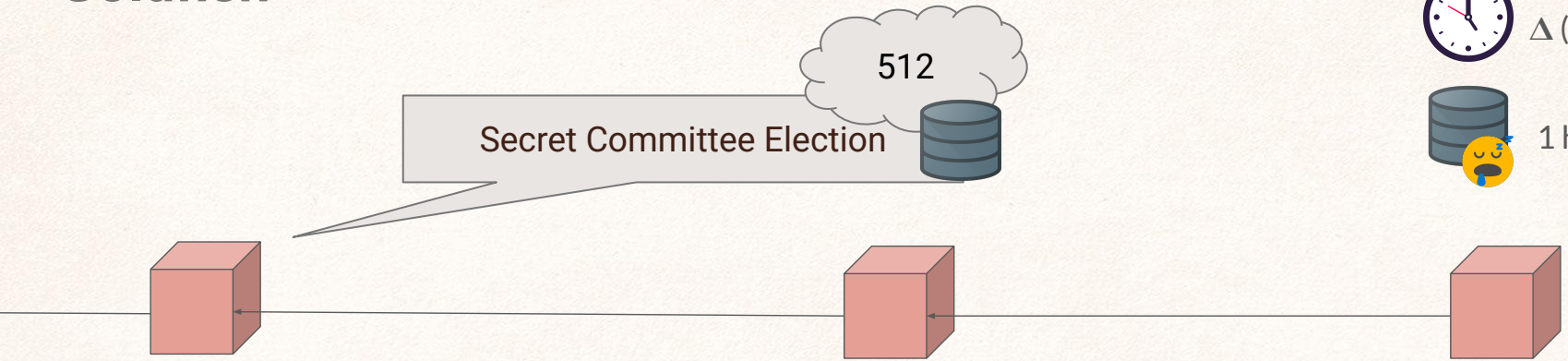


1 honest awake





Goldfish

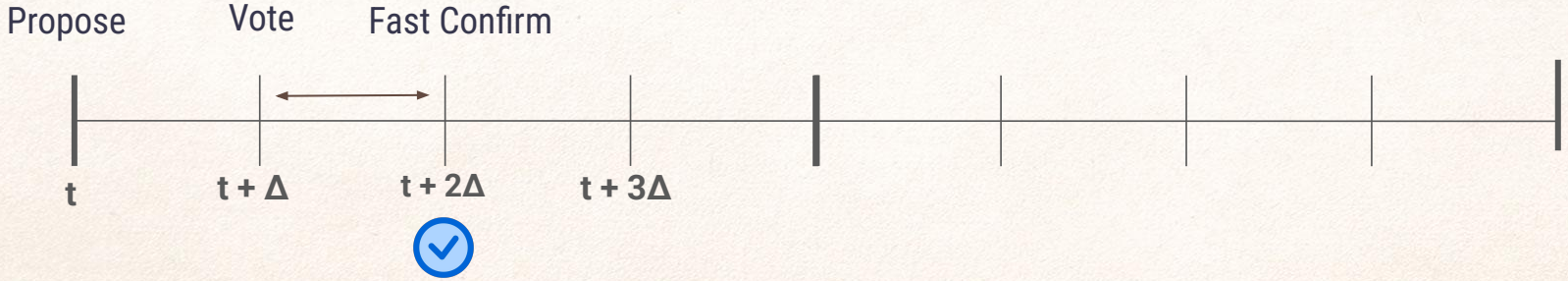




f < n / 2

Δ (Synchronous)

1 honest awake



Goldfish



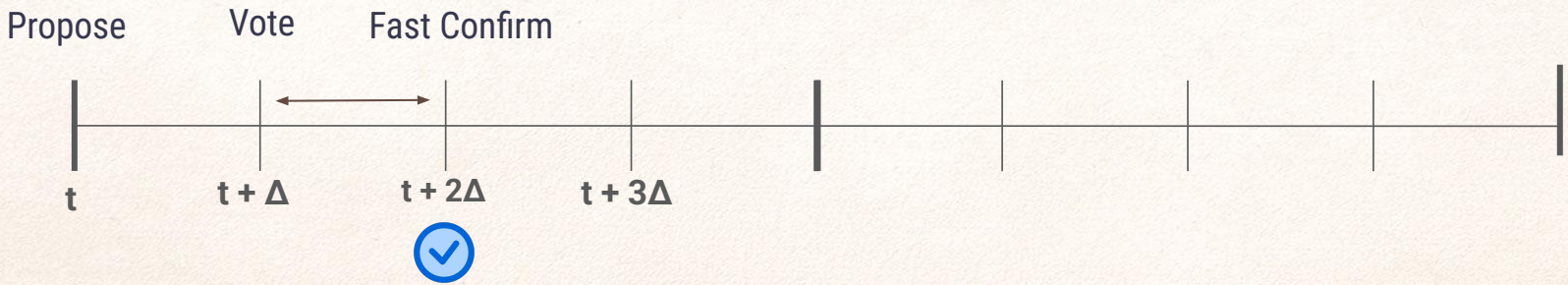
f < n / 2



Δ (Synchronous)



1 honest awake



Goldfish



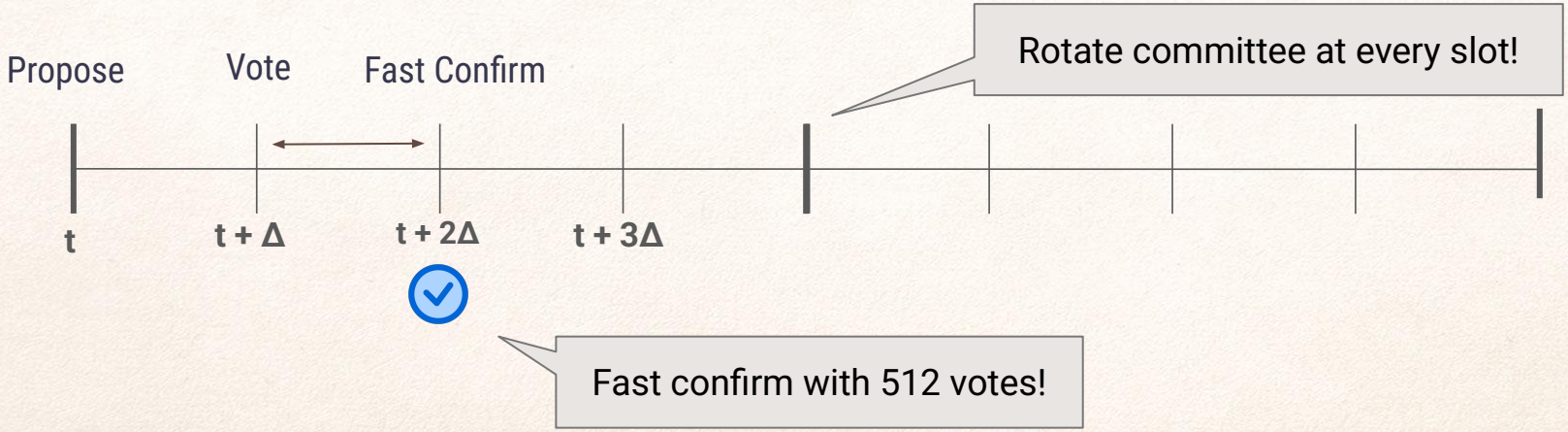
f < n / 2



Δ (Synchronous)



1 honest awake



Goldfish



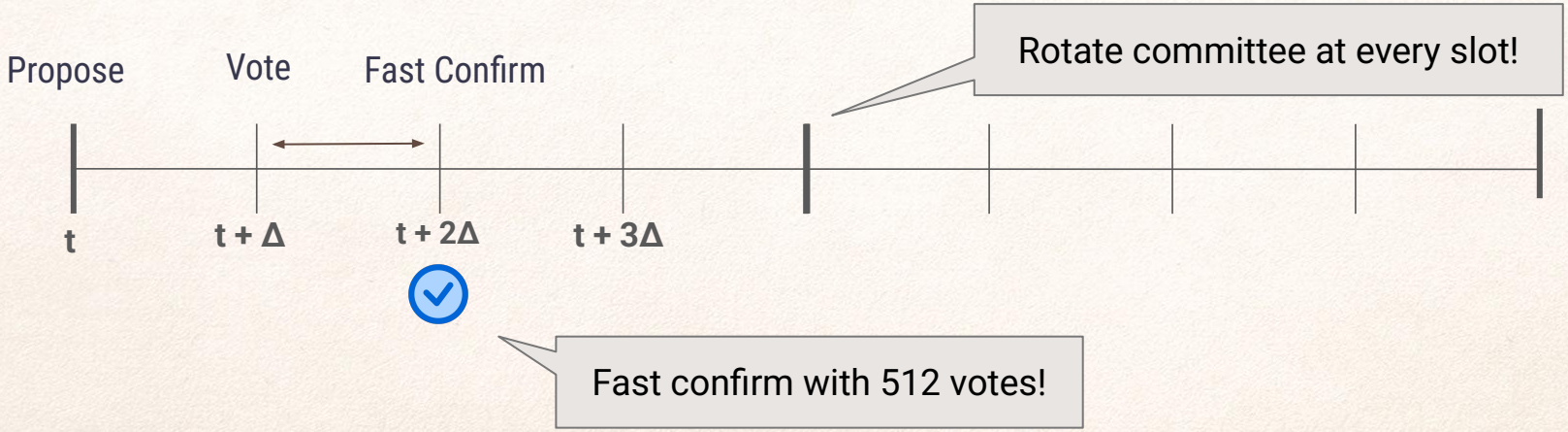
f < n / 2



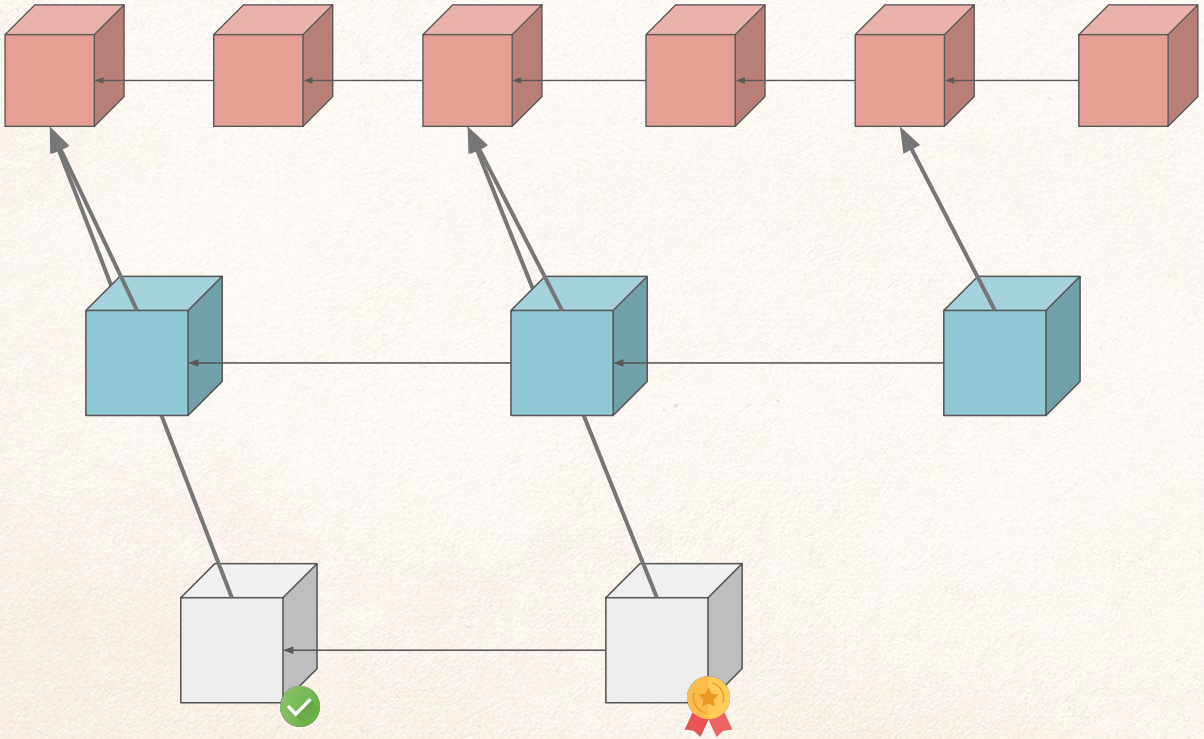
Δ (Synchronous)



1 honest awake



Stabilization Gadget



Goldfish (AC)

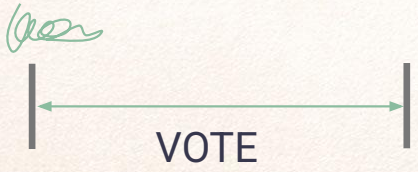
RLMD-GHOST (AR)

BFT (FG)

Advantages

Incremental Improvements

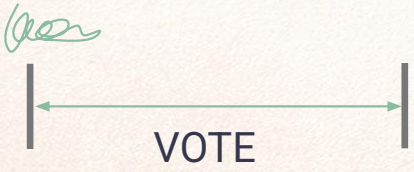
→ consolidation
→ p2p
→ privacy



Advantages

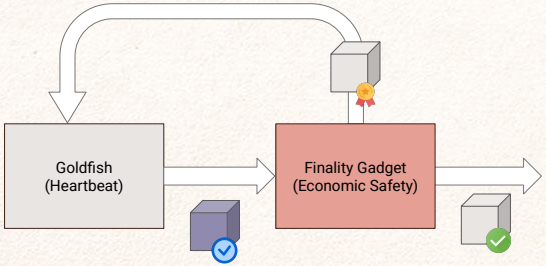
Incremental Improvements

→ consolidation
→ p2p
→ privacy



Flexible

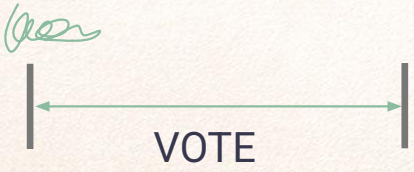
→ hard work up front
→ swap components later



Advantages

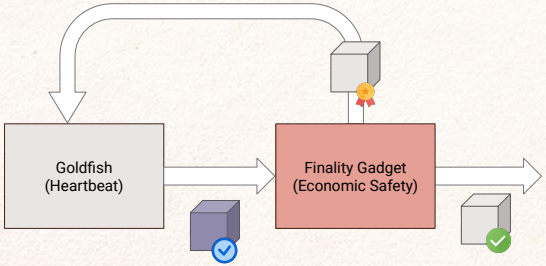
Incremental Improvements

→ consolidation
→ p2p
→ privacy



Flexible

→ hard work up front
→ swap components later

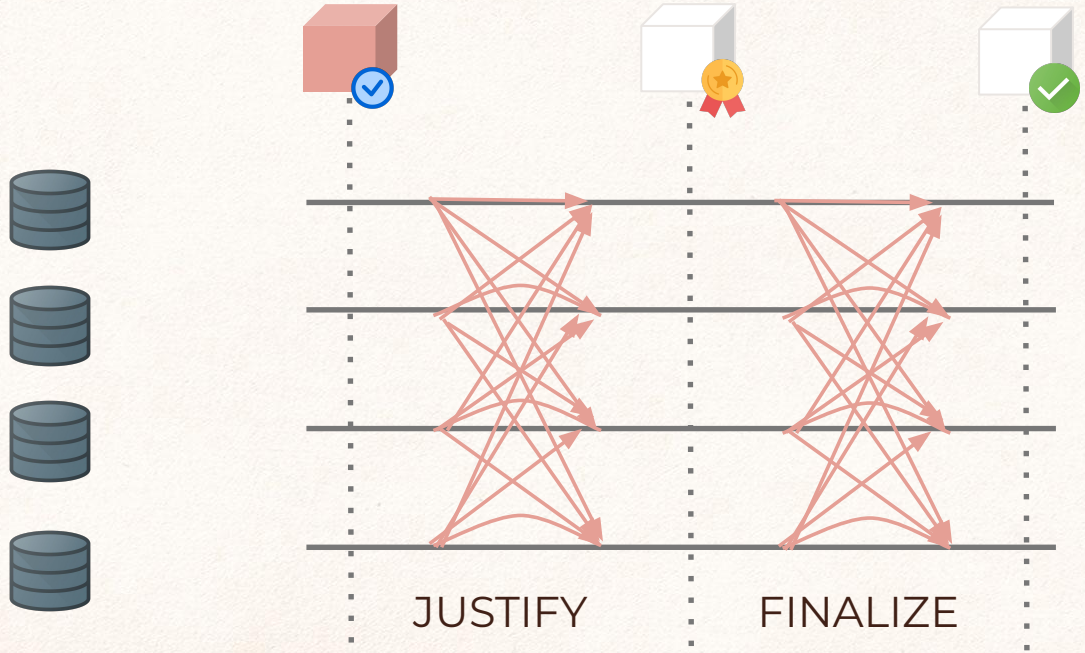


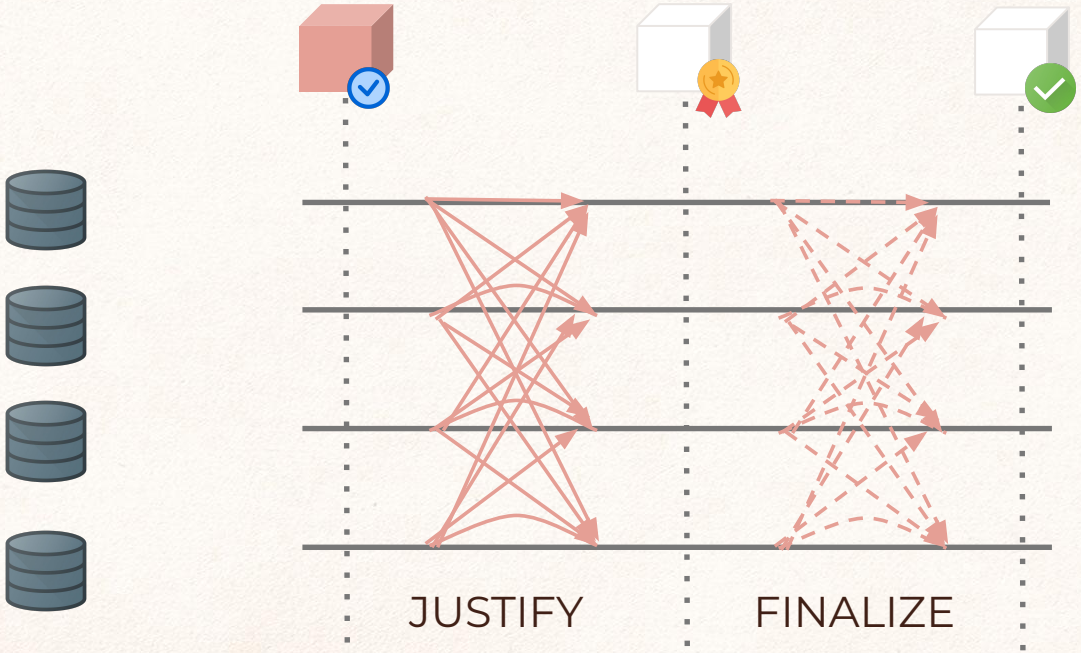
PQ Ready

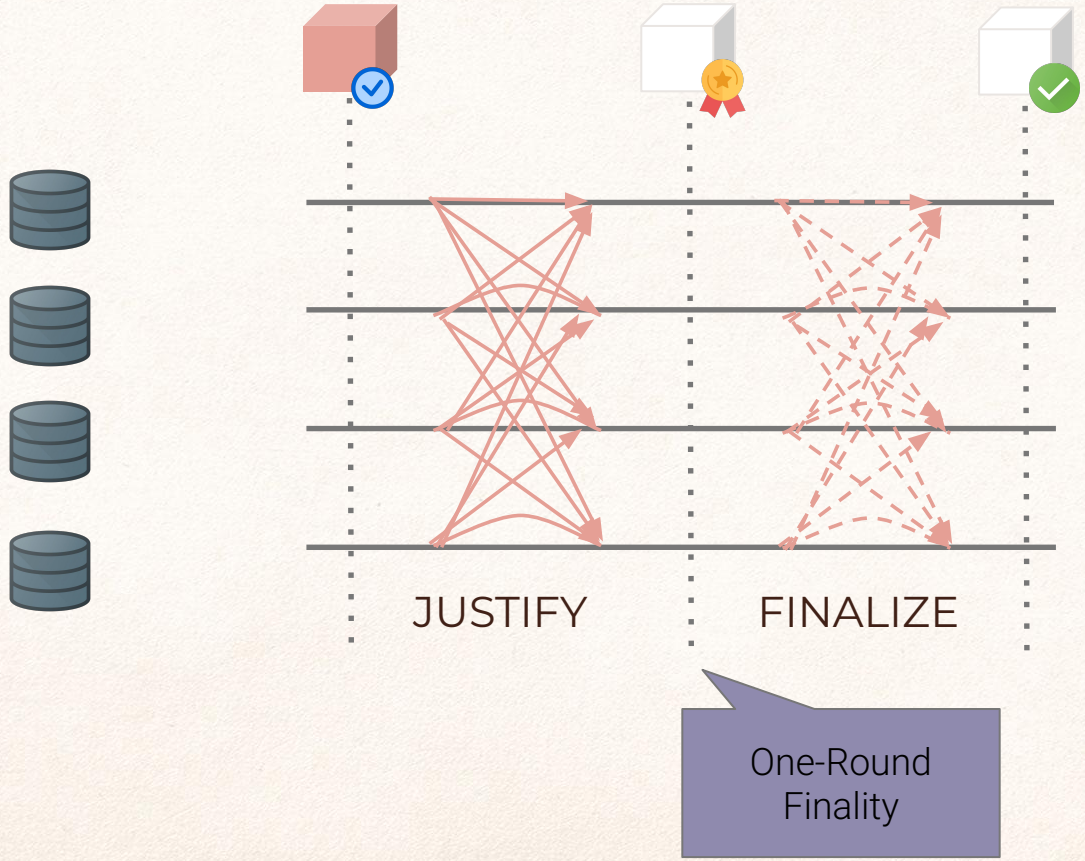
→ Available chain doesn't need aggregation
→ possible short term alternative



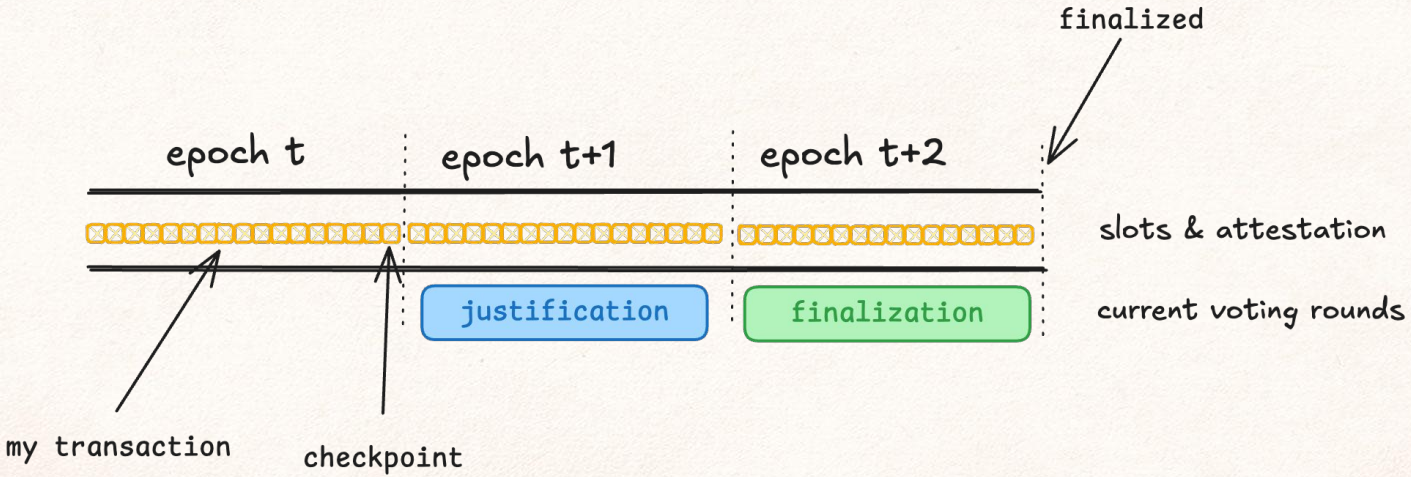

One-Round Finality


JUSTIFY
FINALIZE

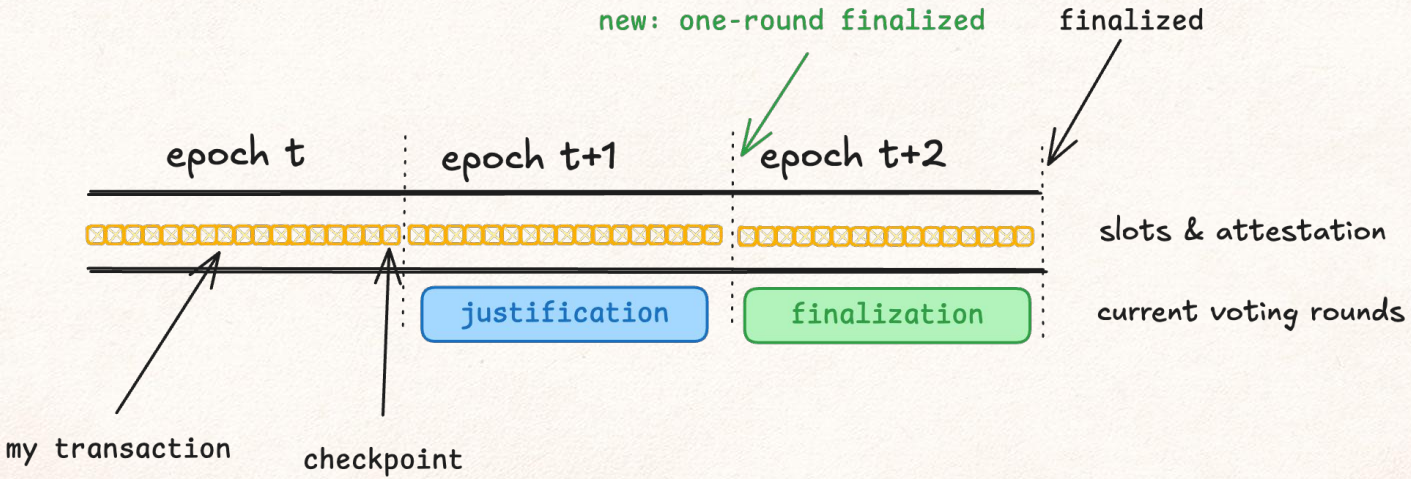

JUSTIFY
FINALIZE


JUSTIFY
FINALIZE
One-Round
Finality

One-Round Casper FFG



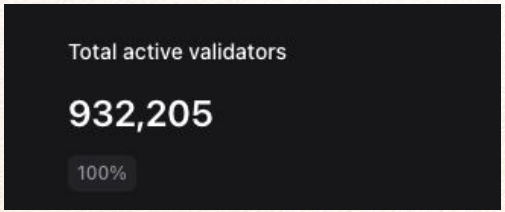
One-Round Casper FFG



Why Ethereum Is Uniquely Positioned

Large Validator Set

→ removing a round is meaningful



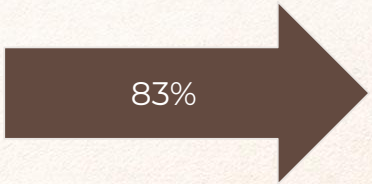
High Participation

→ we can benefit from optimistic conditions

Participation %
99.81%
99.57%

Synergies with High Finality Threshold

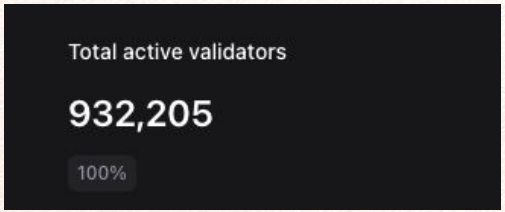
→ we benefit from raising the finality threshold from 67% to e.g., 83%



Why Ethereum Is Uniquely Positioned

Large Validator Set

→ removing a round is meaningful



High Participation

→ we can benefit from optimistic conditions

Participation %
99.81%
99.57%

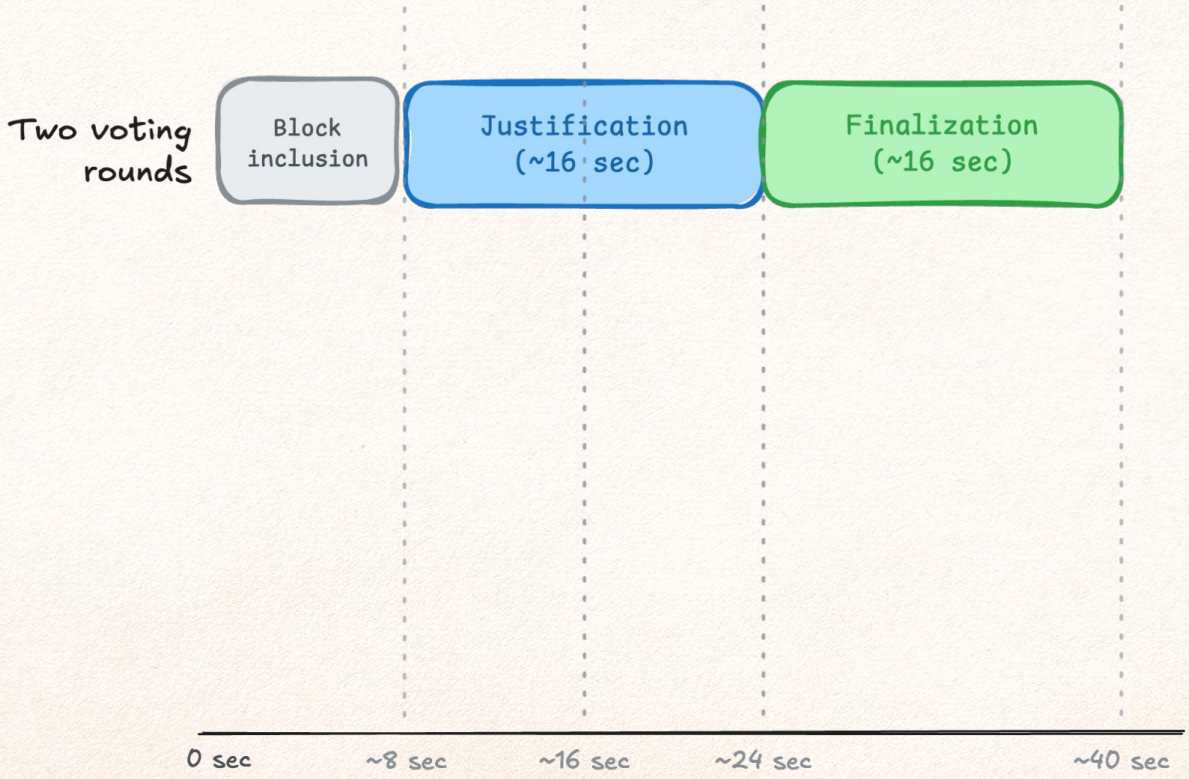
Synergies with High Finality Threshold

→ we benefit from raising the finality threshold from 67% to e.g., 83% because:

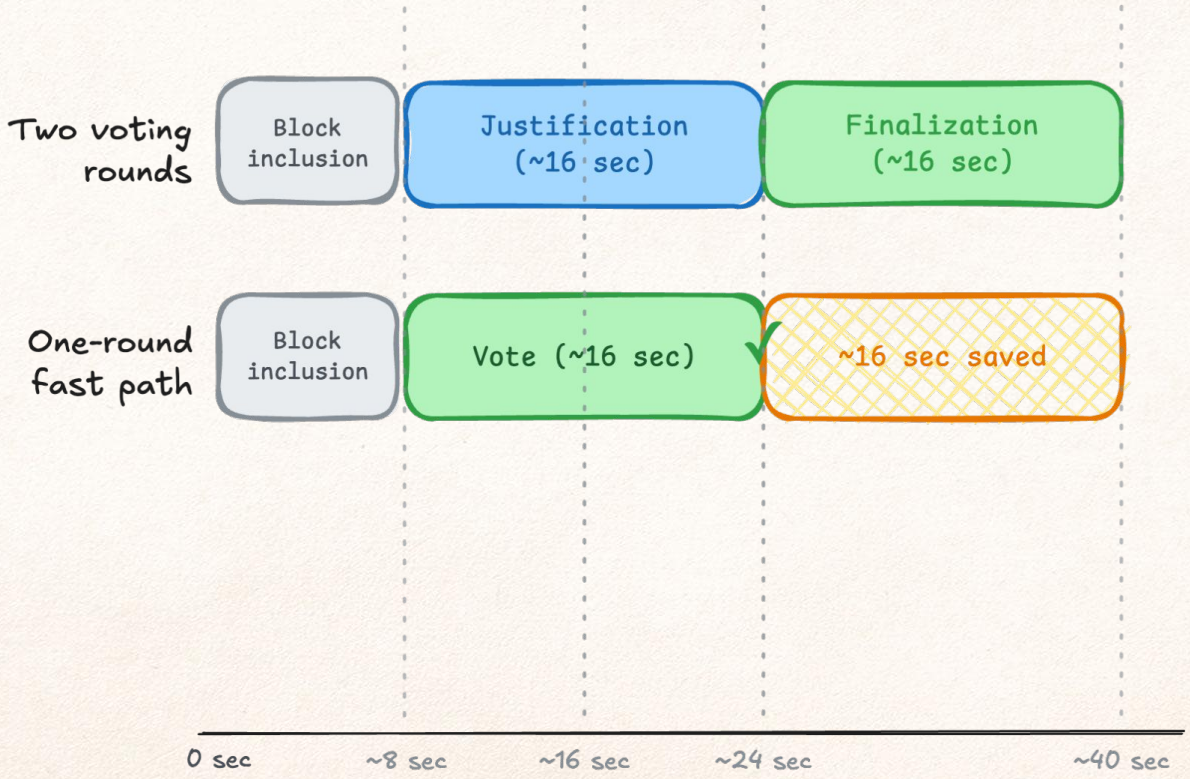
1. Minority can force non-finality
2. Tolerance to bugs
3. Possibly even higher safety



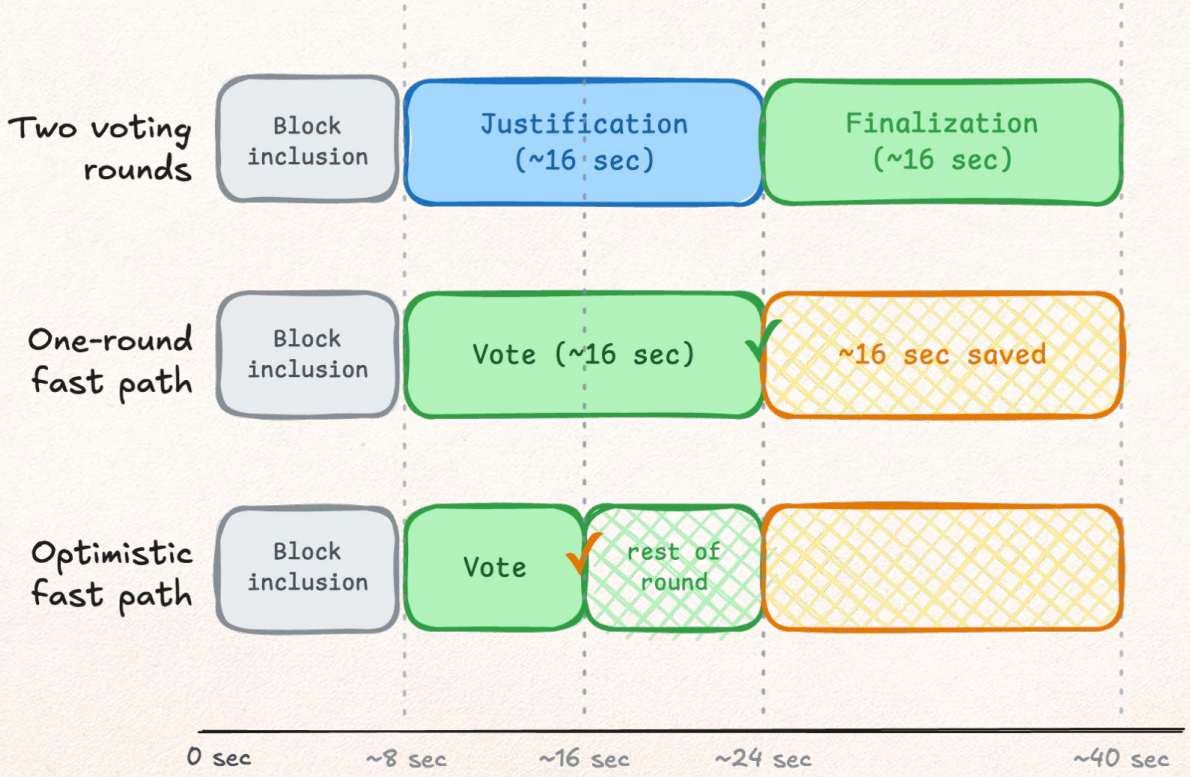
One-Round Decoupled Consensus



One-Round Decoupled Consensus

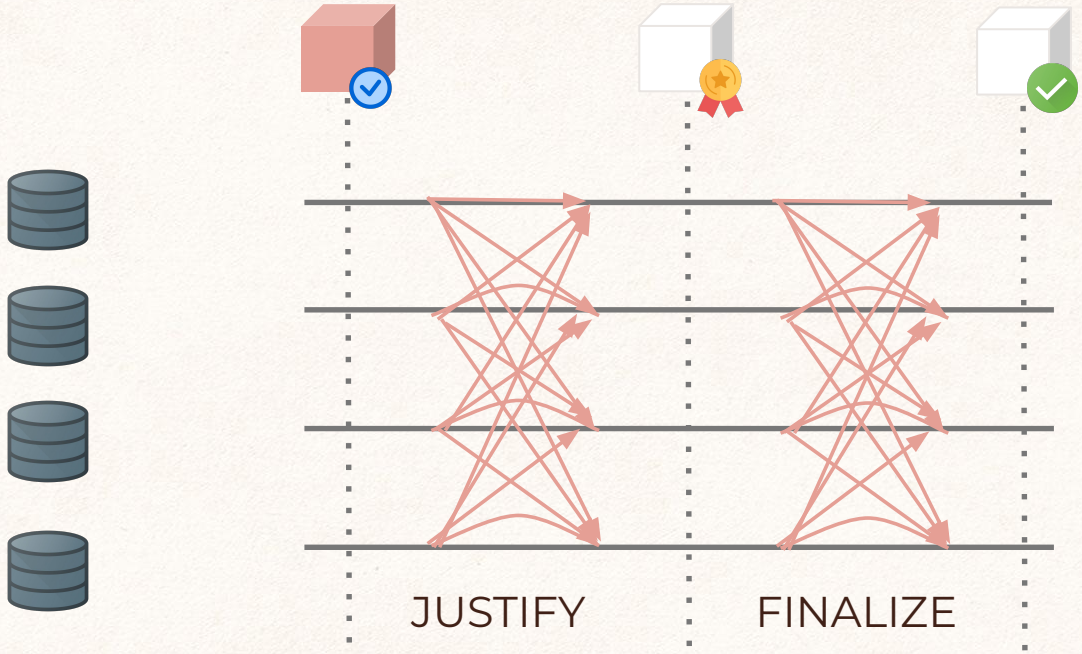


One-Round Decoupled Consensus



Optimistic

f = 33%

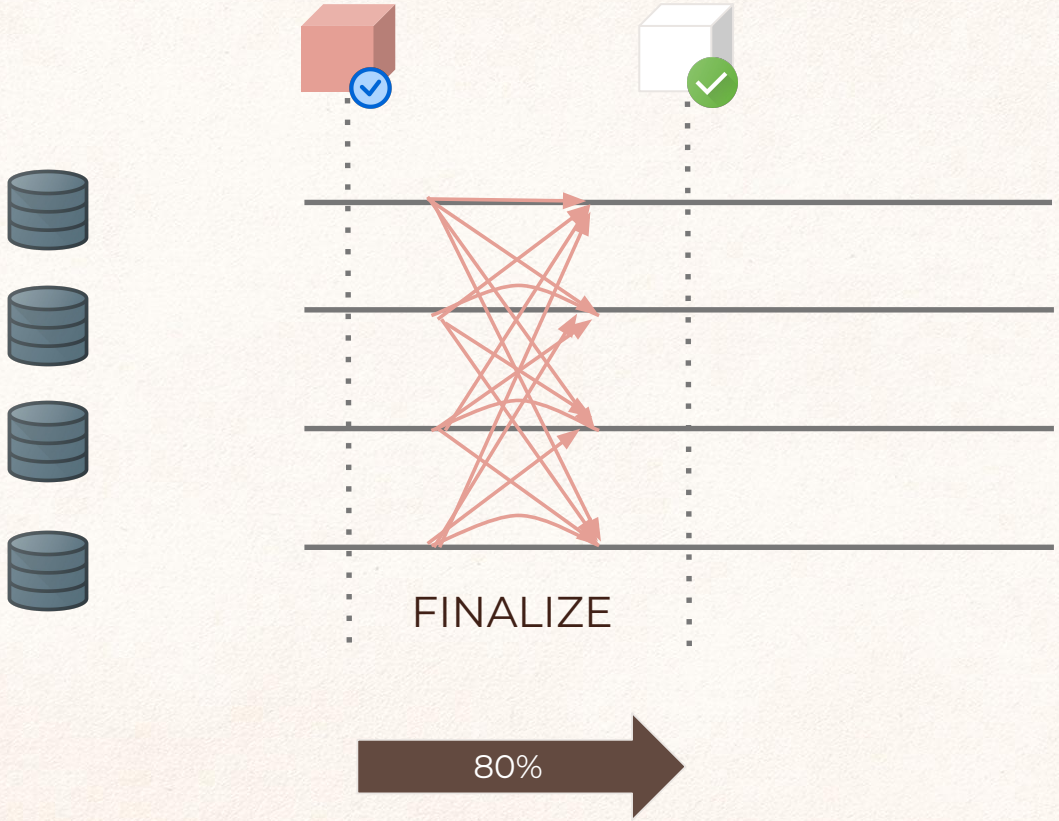


Fast Path: 100%

Slow Path: 67% 67%

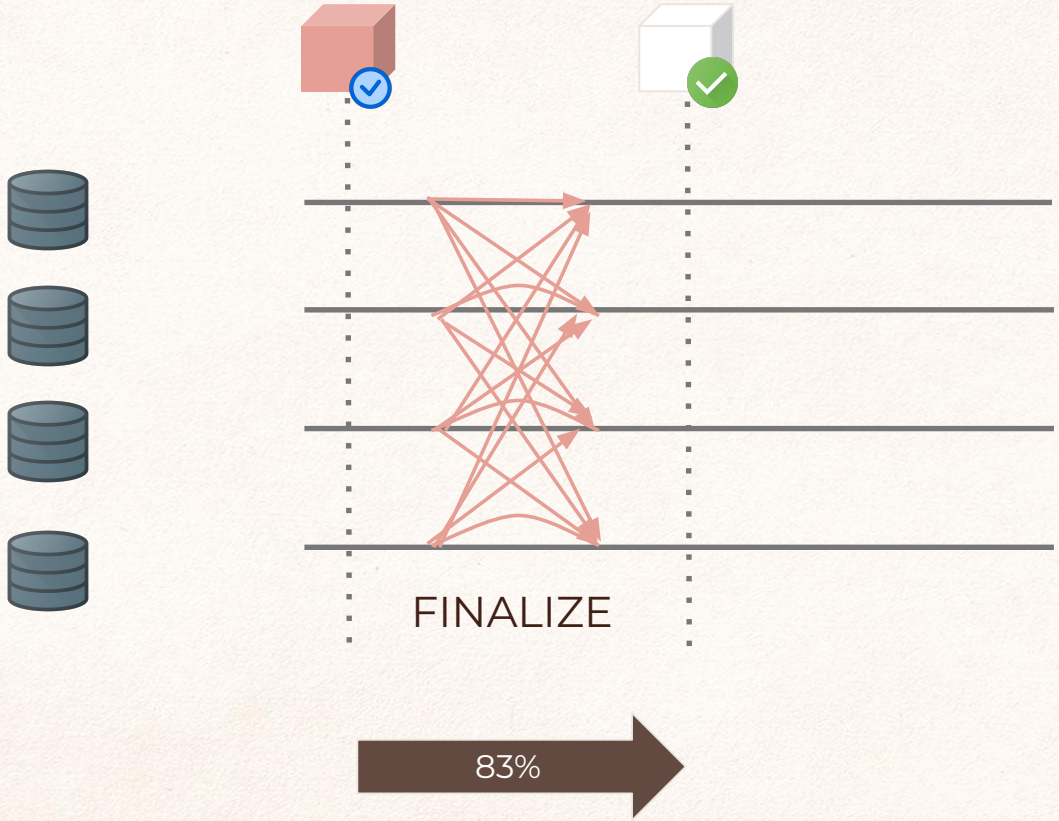
Pure

f = 20%



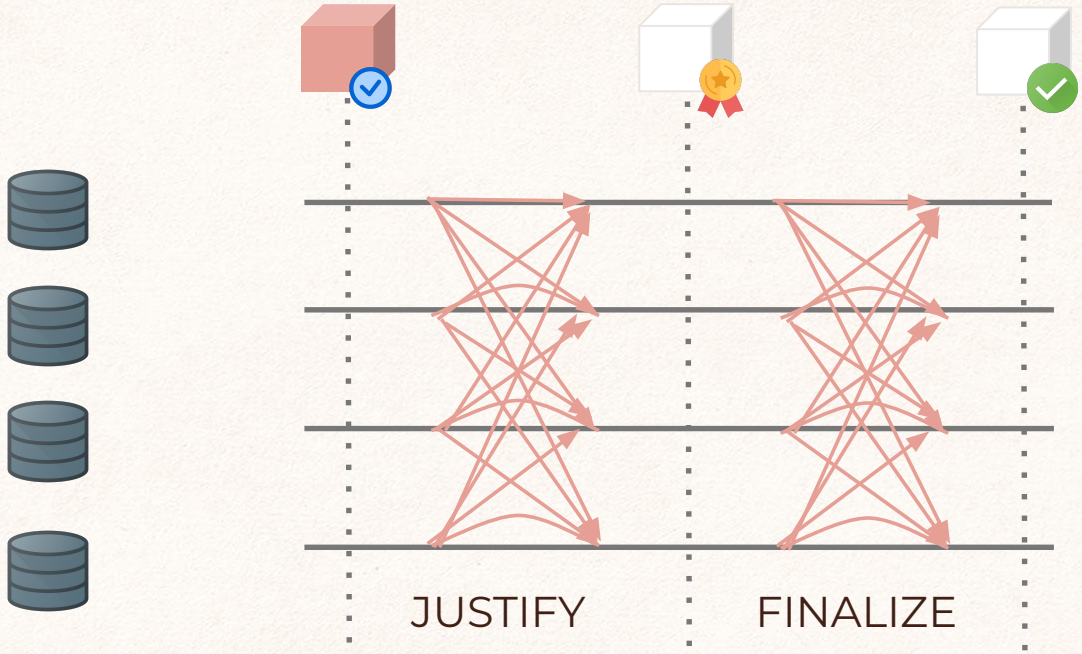
Pure

f = 25%



Hybrid

f = 33%



Fast Path: 92%

Slow Path: 51% 83%

Consensus Goals*

50% Safety Under Sync
⇒ strongest security



Accountable Safety (17%-33%)
⇒ economic security



Accountable Liveness
(programatically recover from partitions, exits and forks)



No Downtime: tolerate, e.g., crash faults by multiple clients
⇒ dynamic availability



Fairness & Decentralization
⇒ geographic decentralization
⇒ no optimistic responsiveness



Veto Power by Minority
No bug can finalize a wrong chain
⇒ liveness threshold of 83%?



Low Latency in spite of

- 128’000+ validators
- Validator privacy
- PQ signatures



Fast Blocks in spite of

- zkEVM
- FOCIL/ePBS
- PQ signatures



Consensus Goals*

Asynchrony Resilience of Available Chain

50% Safety Under Sync
⇒ strongest security

No Downtime: tolerate, e.g., crash faults by multiple clients
⇒ dynamic availability

Low Latency in spite of
- 128’000+ validators
- Validator privacy
- PQ signatures

Accountable Safety (17%-33%)
⇒ economic security

Fairness & Decentralization
⇒ geographic decentralization
⇒ no optimistic responsiveness

Fast Blocks in spite of
- zkEVM
- FOCIL/ePBS
- PQ signatures

Accountable Liveness
(programatically recover from partitions, exits and forks)

Veto Power by Minority
No bug can finalize a wrong chain
⇒ liveness threshold of 83%?

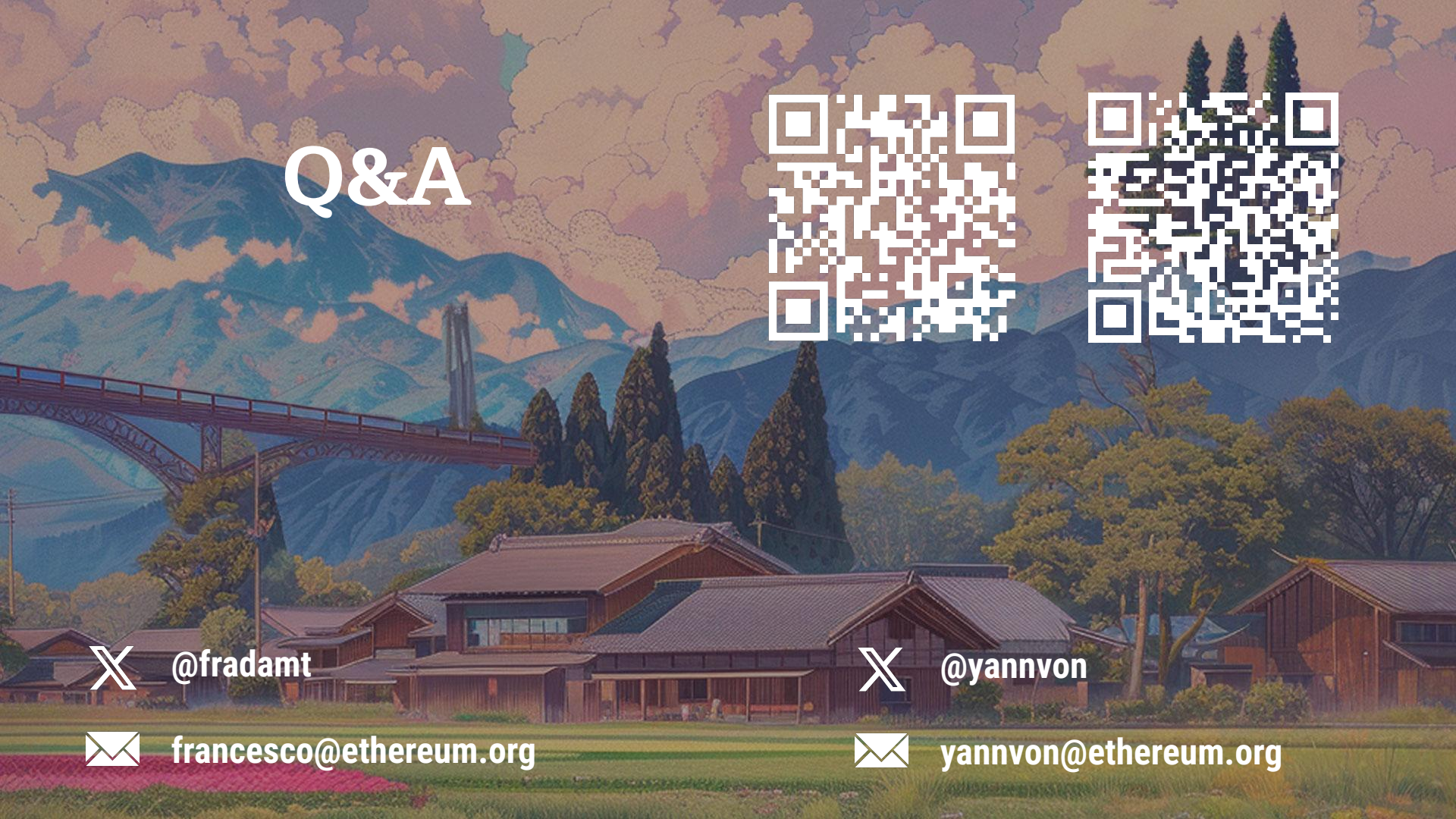
https://consensus.ethereum.foundation/



ethereum foundation
protocol consensus

Open Problems

Available Protocol (heartbeat)	2	+
Finality Protocol (finality gadget)	1	+
Protocol Composition (ebb-and-flow)	2	+
Networking / P2P	1	+
Cross-cutting	1	+


Q&A
@fradamt
francesco@ethereum.org
@yannvon
yannvon@ethereum.org