BFT Consensus

From Academic Paper to Mainnet

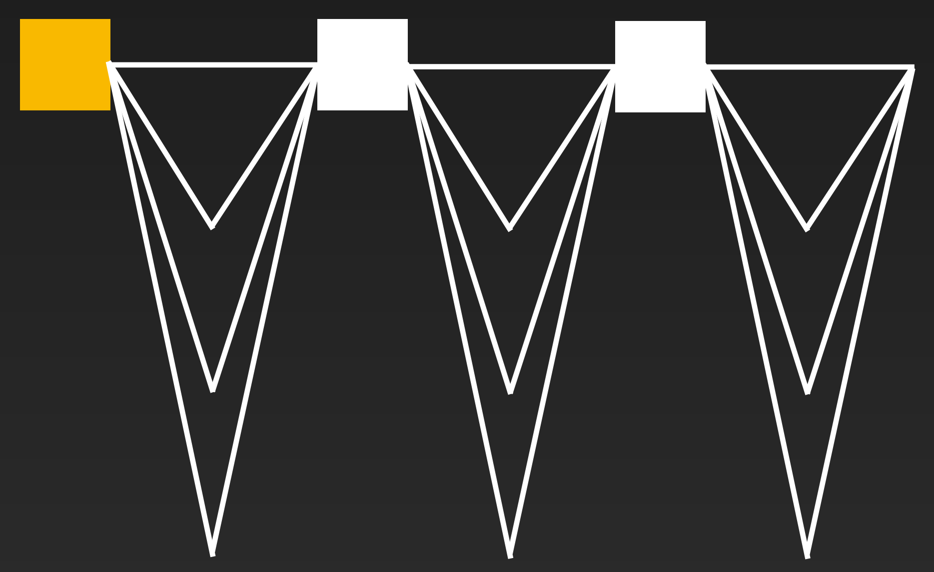
Alberto Sonnino

Alberto Sonnino
Research Scientist

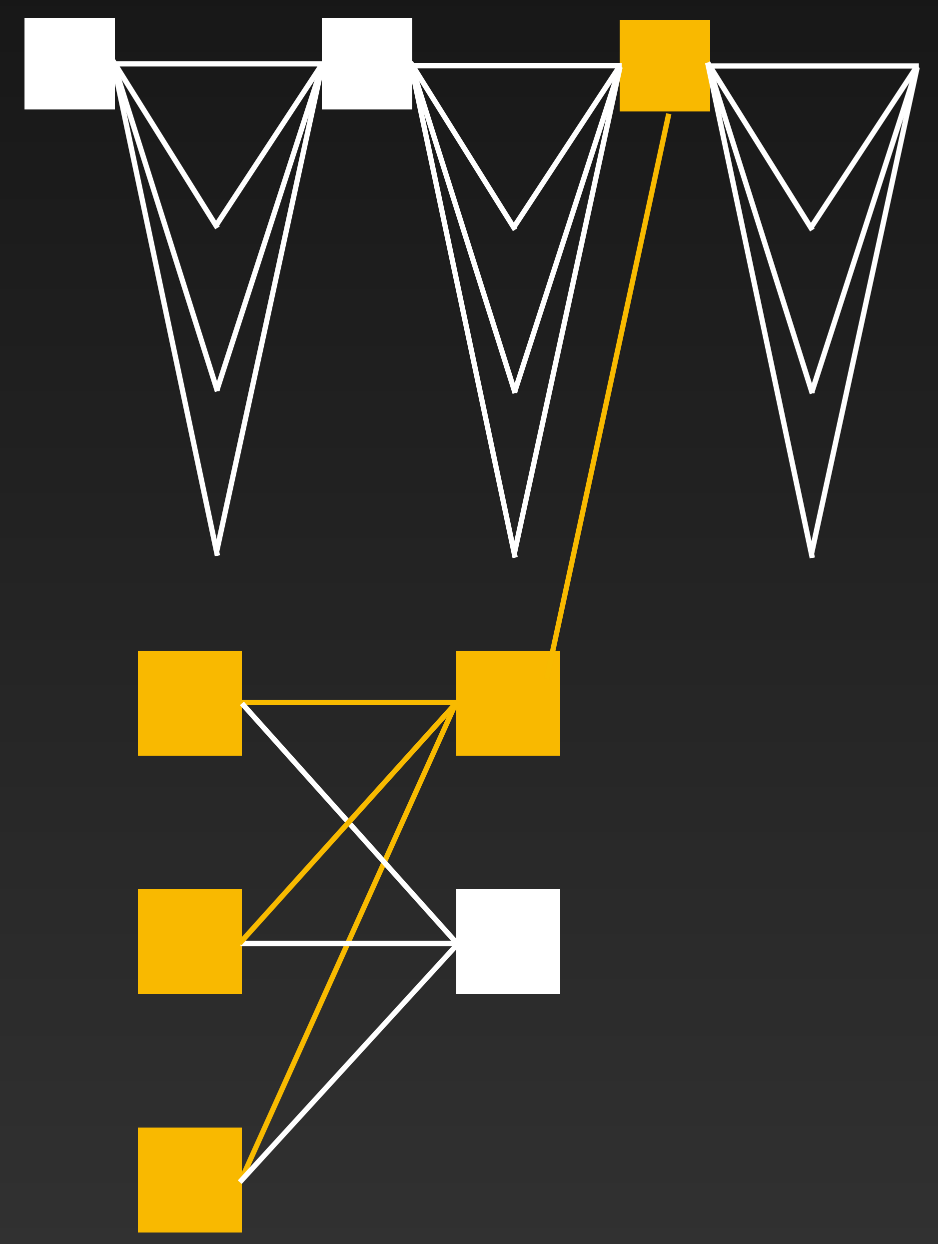
- PhD from UCL (George Danezis & Jens Groth)
- Co-founded Chainspace
- At Libra / Diem from day 1
- Now building the Sui blockchain

2019

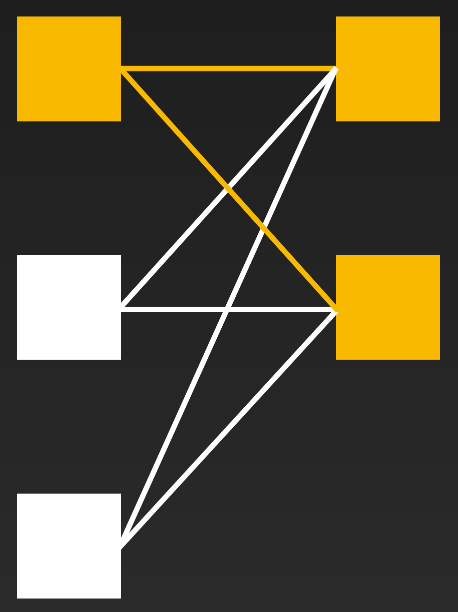
2024



HotStuff



HotStuff + Mempool

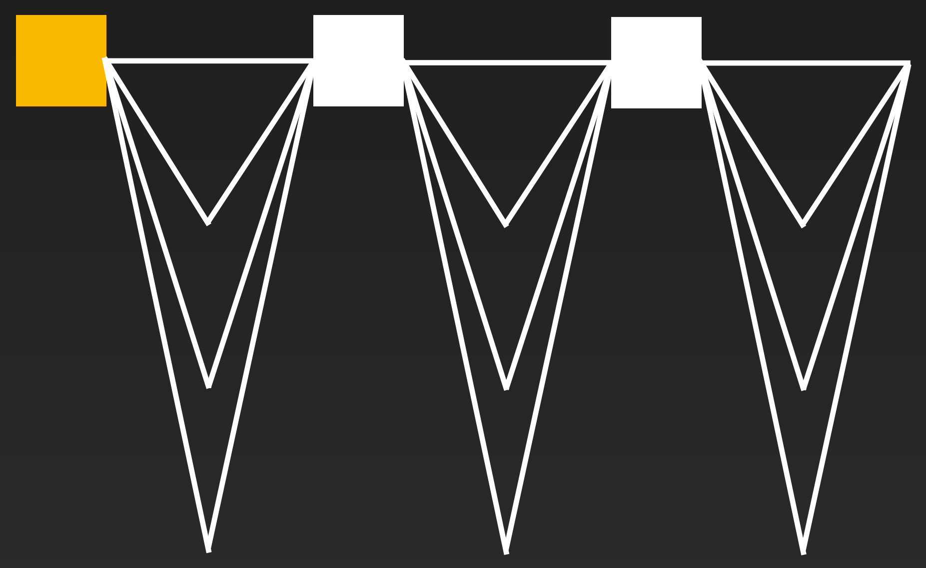


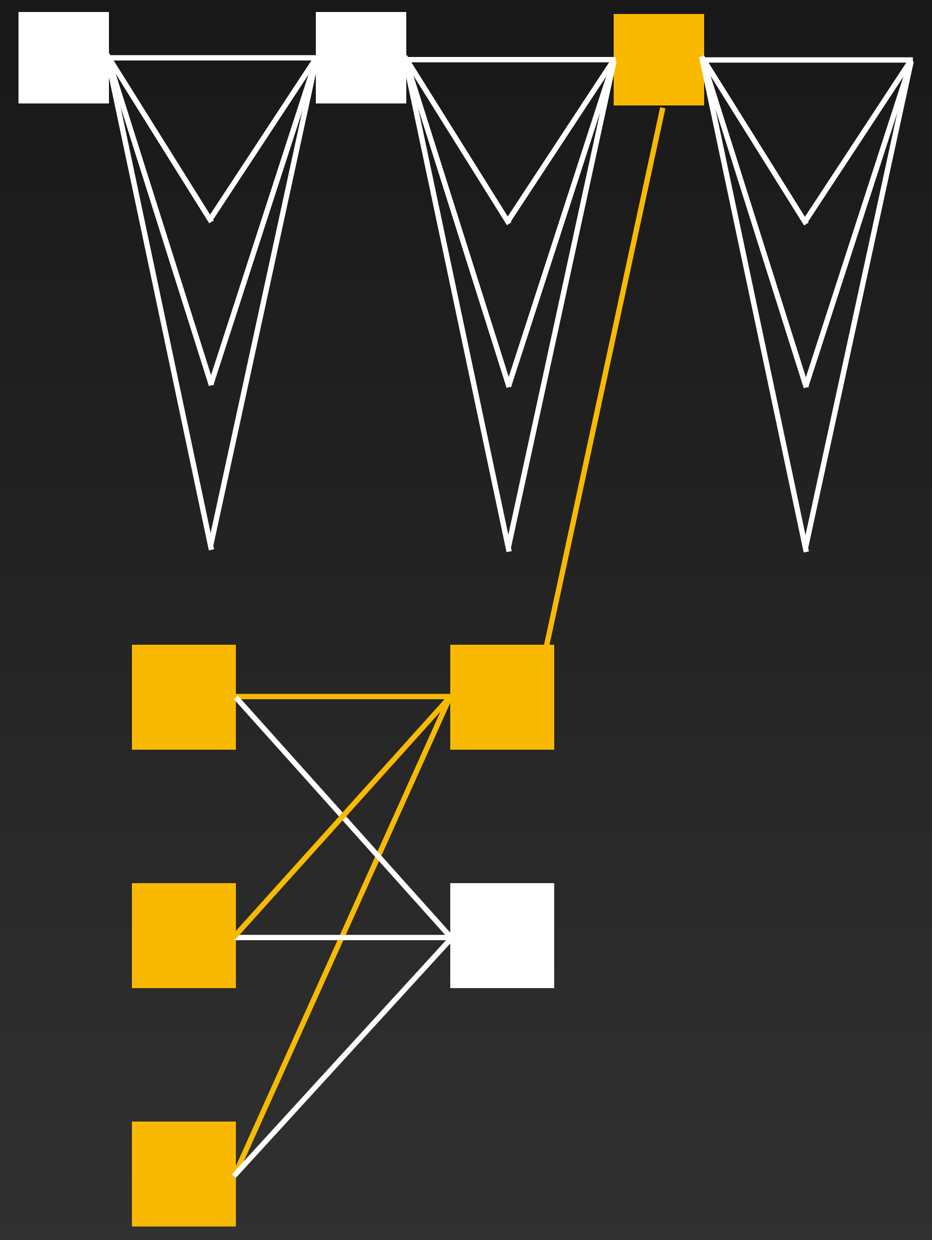
Bullshark,

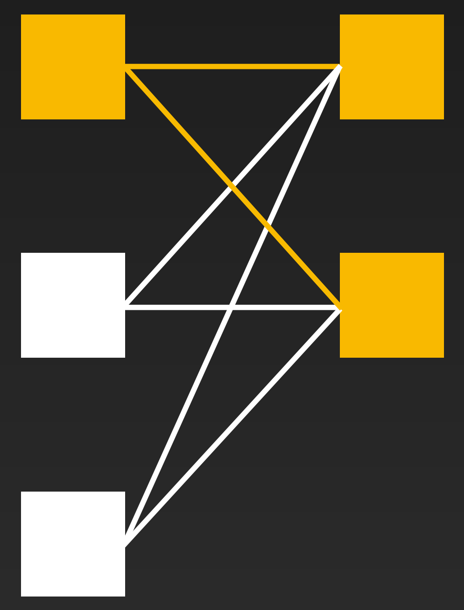
Mysticeti

2019 → 2024

libra → Sui







- Lessons learned
- Open research challenges

Research Gifts

(please keep it short)

Byzantine Fault Tolerance

Byzantine Fault Tolerance

Partial Synchrony

Research Questions

1. Network model?

Lessons Learned

Typical Blockchain

TX →

Mempool / Initial Checks	Ordering	(Sequencial) Execution	DB Update & High-Integrity DS
P2P flood & Selection on fee	Sequence all transactions in blocks	Execute each transaction	Update DB, indexes, crypto (Merkle trees)
Overlay flooding slow and with significant redundancy	Seconds latency, traditionally low throughput	Single core does all computations. (eg EVM ~300 tps)	Added latency of store, blocks, and crypto computations

Typical Blockchain

P2P flood
& Selection on fee

Sequence all transactions in blocks

Mempool / Initial Checks

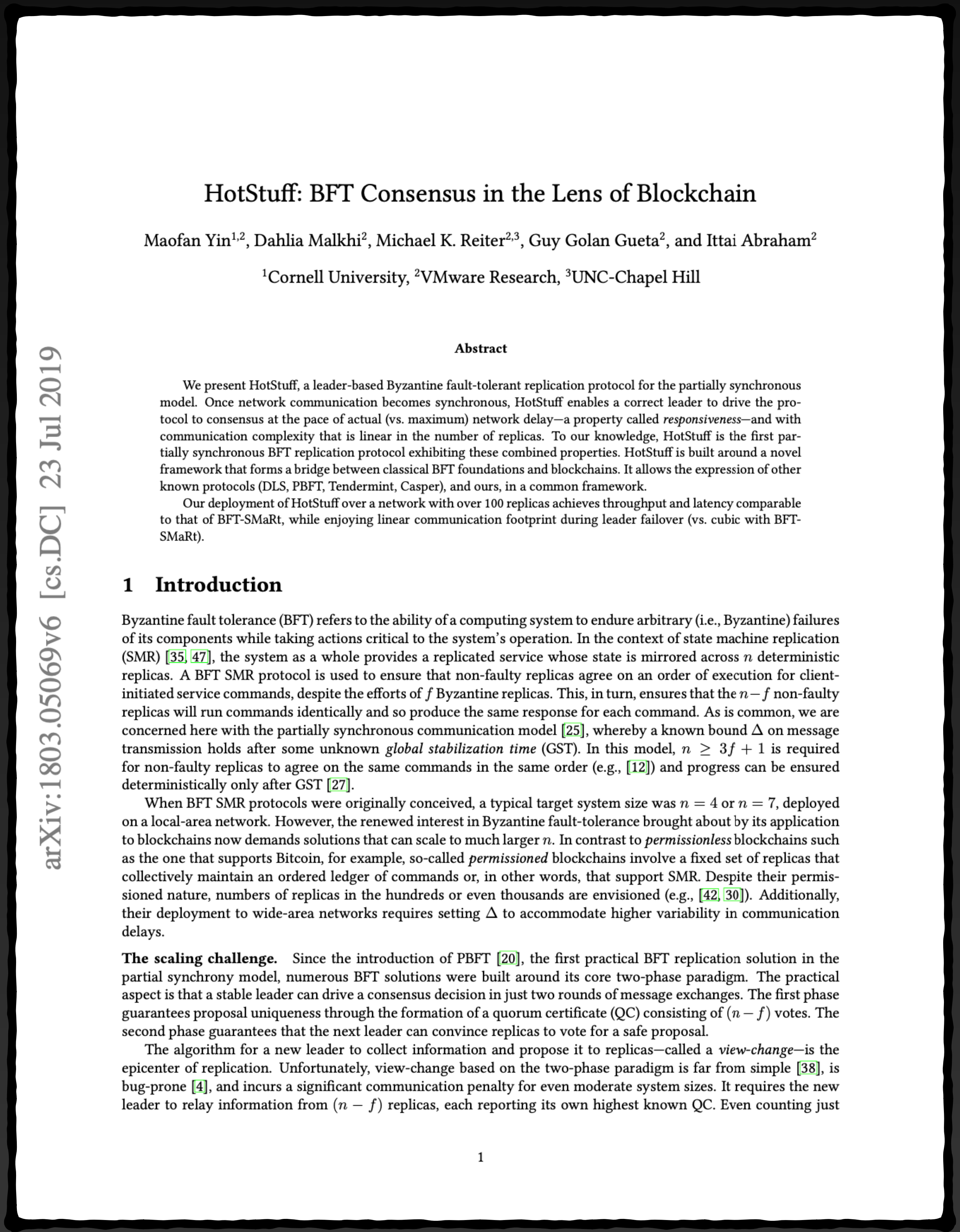
Ordering

Overlay flooding slow and with significant redundancy

Seconds latency, traditionally low throughput

Libra, 2019

HotStuff



arXiv:1803.05069v6 [cs.DC] 23 Jul 2019

HotStuff: BFT Consensus in the Lens of Blockchain

Maofan Yin[1,2], Dahlia Malkhi[2], Michael K. Reiter[2,3], Guy Golan Gueta[2], and Ittai Abraham[2]

[1]Cornell University, [2]VMware Research, [3]UNC-Chapel Hill

Abstract

We present HotStuff, a leader-based Byzantine fault-tolerant replication protocol for the partially synchronous model. Once network communication becomes synchronous, HotStuff enables a correct leader to drive the protocol to consensus at the pace of actual (vs. maximum) network delay—a property called *responsiveness*—and with communication complexity that is linear in the number of replicas. To our knowledge, HotStuff is the first partially synchronous BFT replication protocol exhibiting these combined properties. HotStuff is built around a novel framework that forms a bridge between classical BFT foundations and blockchains. It allows the expression of other known protocols (DLS, PBFT, Tendermint, Casper), and ours, in a common framework.

Our deployment of HotStuff over a network with over 100 replicas achieves throughput and latency comparable to that of BFT-SMaRt, while enjoying linear communication footprint during leader failover (vs. cubic with BFT-SMaRt).

1 Introduction

Byzantine fault tolerance (BFT) refers to the ability of a computing system to endure arbitrary (i.e., Byzantine) failures of its components while taking actions critical to the system's operation. In the context of state machine replication (SMR) [35, 47], the system as a whole provides a replicated service whose state is mirrored across n deterministic replicas. A BFT SMR protocol is used to ensure that non-faulty replicas agree on an order of execution for client-initiated service commands, despite the efforts of f Byzantine replicas. This, in turn, ensures that the $n-f$ non-faulty replicas will run commands identically and so produce the same response for each command. As is common, we are concerned here with the partially synchronous communication model [25], whereby a known bound Δ on message transmission holds after some unknown *global stabilization time* (GST). In this model, $n \geq 3f + 1$ is required for non-faulty replicas to agree on the same commands in the same order (e.g., [12]) and progress can be ensured deterministically only after GST [27].

When BFT SMR protocols were originally conceived, a typical target system size was $n = 4$ or $n = 7$, deployed on a local-area network. However, the renewed interest in Byzantine fault-tolerance brought about by its application to blockchains now demands solutions that can scale to much larger n. In contrast to *permissionless* blockchains such as the one that supports Bitcoin, for example, so-called *permissioned* blockchains involve a fixed set of replicas that collectively maintain an ordered ledger of commands or, in other words, that support SMR. Despite their permissioned nature, numbers of replicas in the hundreds or even thousands are envisioned (e.g., [42, 30]). Additionally, their deployment to wide-area networks requires setting Δ to accommodate higher variability in communication delays.

The scaling challenge. Since the introduction of PBFT [20], the first practical BFT replication solution in the partial synchrony model, numerous BFT solutions were built around its core two-phase paradigm. The practical aspect is that a stable leader can drive a consensus decision in just two rounds of message exchanges. The first phase guarantees proposal uniqueness through the formation of a quorum certificate (QC) consisting of $(n - f)$ votes. The second phase guarantees that the next leader can convince replicas to vote for a safe proposal.

The algorithm for a new leader to collect information and propose it to replicas—called a *view-change*—is the epicenter of replication. Unfortunately, view-change based on the two-phase paradigm is far from simple [38], is bug-prone [4], and incurs a significant communication penalty for even moderate system sizes. It requires the new leader to relay information from $(n - f)$ replicas, each reporting its own highest known QC. Even counting just

1

?

HashGraph

Verifying the Hashgraph Consensus Algorithm

Karl Crary

Carnegie Mellon University

arXiv:2102.01167v1 [cs.LO] 1 Feb 2021

Abstract

The Hashgraph consensus algorithm is an algorithm for asynchronous Byzantine fault tolerance intended for distributed shared ledgers. Its main distinguishing characteristic is it achieves consensus without exchanging any extra messages; each participant's votes can be determined from public information, so votes need not be transmitted.

In this paper, we discuss our experience formalizing the Hashgraph algorithm and its correctness proof using the Coq proof assistant. The paper is self-contained; it includes a complete discussion of the algorithm and its correctness argument in English.

1 Introduction

Byzantine fault-tolerance is the problem of coordinating a distributed system while some participants may maliciously break the rules. Often other challenges are also present, such as unreliable communications. The problem is at the center of a variety of new applications such as cryptocurrencies. Such applications rely on *distributed shared ledgers,* a form of Byzantine fault-tolerance in which a set of transactions are assigned a place in a globally-agreed total order that is *immutable*. The latter means that once a transaction enters the order, no new transaction can enter at an earlier position.

A distributed shared ledger makes it possible for all participants to agree, at any point in the order, on the current owner of a digital commodity such as a unit of cryptocurrency. A transaction transferring ownership is valid if the commodity's current owner authorizes the transaction. (The authorization mechanism—presumably using a digital signature—is beyond the scope of the ledger itself.) Because the order is total, one transaction out of any pair has priority. Thus we can show that a commodity's chain of ownership is uniquely determined. Finally, because the order is immutable, the chain of ownership cannot change except by adding new transactions at the end.

Algorithms for Byzantine consensus (under various assumptions) have existed for some time, indeed longer than the problem has been named [12, 9]. Practical algorithms are more recent; in 1999, Castro and Liskov [6] gave an algorithm that when installed into the NFS file system slowed it only 3%. As Byzantine consensus algorithms have become more practical, they have been tailored to specific applications. Castro and Liskov's algorithm was designed for fault-tolerant state machine replication [13] and probably would not perform well under the workload of a distributed shared ledger.

However, in the last few years there have arisen Byzantine fault-tolerance algorithms suitable for distributed shared ledgers, notably HoneyBadgerBFT [10], BEAT [7], and—the subject of this paper—Hashgraph [2]. Moreover, the former two each claim to be the first practical *asynchronous* BFT algorithm (with different standards of practicality). Hashgraph does not claim to be first, but is also practical and asynchronous.

In parallel with that line of work has been the development of distributed shared ledgers based on *proof of work,* beginning with Bitcoin [11]. The idea behind proof of work is to maintain agreement on the ledger by maintaining a list of blocks of transactions, and to ensure that the list does not become a tree. To ensure this, the rules state that (1) the longest branch defines the list, and (2) to create a new block, one must first solve a mathematical problem that takes the list's old head as one of its inputs. The problem's solution is much easier to verify than to obtain, so when one learns of a new block, one's incentive is to restart work from the new head rather than continue work from the old head.

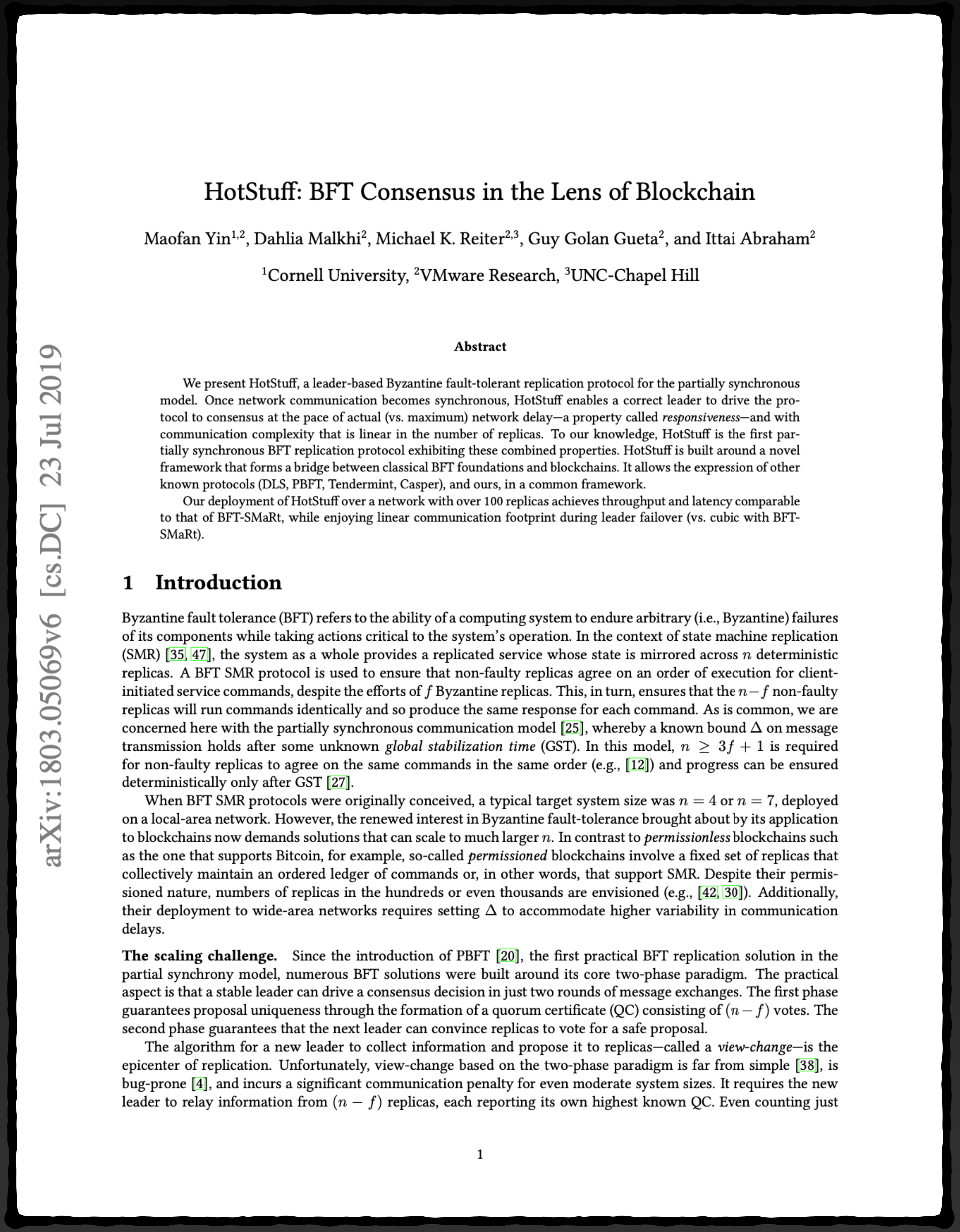
Bitcoin and some of its cousins are widely used, so in a certain sense they are indisputably practical. They are also truly permissionless, in a way that the BFT algorithms, including Hashgraph, cannot quite claim. Nevertheless, they offer severely limited throughput. Bitcoin is limited to seven transactions per second and has a latency of one hour, while its BFT competitors all do several orders of magnitude better. Proof-of-work systems are also criticized for being wasteful: an enormous amount of electricity is expended on block-creation efforts that nearly always fail. Finally—more to the point of this paper—the theoretical properties of proof of work are not well understood.

The Hashgraph consensus algorithm is designed to support high-performance applications of a distributed shared ledger. Like the other BFT systems, it is several orders of magnitude faster than proof of work. Actual performance depends very much on configuration choices (*e.g.,* how many peers, geographic distribution, tradeoff between latency and throughput, etc.), but in all configurations published in Miller, *et. al* [10] (for HoneyBadgerBFT) and Duan, *et al.* [7] (for BEAT), the Hashgraph algorithm equals or exceeds the published performance figures [4]. A frequently cited throughput goal is to equal the Visa credit-card network. According to Visa's published figures, Hashgraph can

Too much impact?

- Patent your work
- Send the patent around
- Ask companies to cite your patented work (ideally in public)

Libra, 2019



arXiv:1803.05069v6 [cs.DC] 23 Jul 2019

HotStuff: BFT Consensus in the Lens of Blockchain

Maofan Yin[1,2], Dahlia Malkhi[2], Michael K. Reiter[2,3], Guy Golan Gueta[2], and Ittai Abraham[2]

[1]Cornell University, [2]VMware Research, [3]UNC-Chapel Hill

Abstract

We present HotStuff, a leader-based Byzantine fault-tolerant replication protocol for the partially synchronous model. Once network communication becomes synchronous, HotStuff enables a correct leader to drive the protocol to consensus at the pace of actual (vs. maximum) network delay—a property called *responsiveness*—and with communication complexity that is linear in the number of replicas. To our knowledge, HotStuff is the first partially synchronous BFT replication protocol exhibiting these combined properties. HotStuff is built around a novel framework that forms a bridge between classical BFT foundations and blockchains. It allows the expression of other known protocols (DLS, PBFT, Tendermint, Casper), and ours, in a common framework.

Our deployment of HotStuff over a network with over 100 replicas achieves throughput and latency comparable to that of BFT-SMaRt, while enjoying linear communication footprint during leader failover (vs. cubic with BFT-SMaRt).

1 Introduction

Byzantine fault tolerance (BFT) refers to the ability of a computing system to endure arbitrary (i.e., Byzantine) failures of its components while taking actions critical to the system's operation. In the context of state machine replication (SMR) [35, 47], the system as a whole provides a replicated service whose state is mirrored across n deterministic replicas. A BFT SMR protocol is used to ensure that non-faulty replicas agree on an order of execution for client-initiated service commands, despite the efforts of f Byzantine replicas. This, in turn, ensures that the $n-f$ non-faulty replicas will run commands identically and so produce the same response for each command. As is common, we are concerned here with the partially synchronous communication model [25], whereby a known bound Δ on message transmission holds after some unknown *global stabilization time* (GST). In this model, $n \geq 3f+1$ is required for non-faulty replicas to agree on the same commands in the same order (e.g., [12]) and progress can be ensured deterministically only after GST [27].

When BFT SMR protocols were originally conceived, a typical target system size was $n = 4$ or $n = 7$, deployed on a local-area network. However, the renewed interest in Byzantine fault-tolerance brought about by its application to blockchains now demands solutions that can scale to much larger n. In contrast to *permissionless* blockchains such as the one that supports Bitcoin, for example, so-called *permissioned* blockchains involve a fixed set of replicas that collectively maintain an ordered ledger of commands or, in other words, that support SMR. Despite their permissioned nature, numbers of replicas in the hundreds or even thousands are envisioned (e.g., [42, 30]). Additionally, their deployment to wide-area networks requires setting Δ to accommodate higher variability in communication delays.

The scaling challenge. Since the introduction of PBFT [20], the first practical BFT replication solution in the partial synchrony model, numerous BFT solutions were built around its core two-phase paradigm. The practical aspect is that a stable leader can drive a consensus decision in just two rounds of message exchanges. The first phase guarantees proposal uniqueness through the formation of a quorum certificate (QC) consisting of $(n-f)$ votes. The second phase guarantees that the next leader can convince replicas to vote for a safe proposal.

The algorithm for a new leader to collect information and propose it to replicas—called a *view-change*—is the epicenter of replication. Unfortunately, view-change based on the two-phase paradigm is far from simple [38], is bug-prone [4], and incurs a significant communication penalty for even moderate system sizes. It requires the new leader to relay information from $(n-f)$ replicas, each reporting its own highest known QC. Even counting just

1

HotStuff

✓ Linear

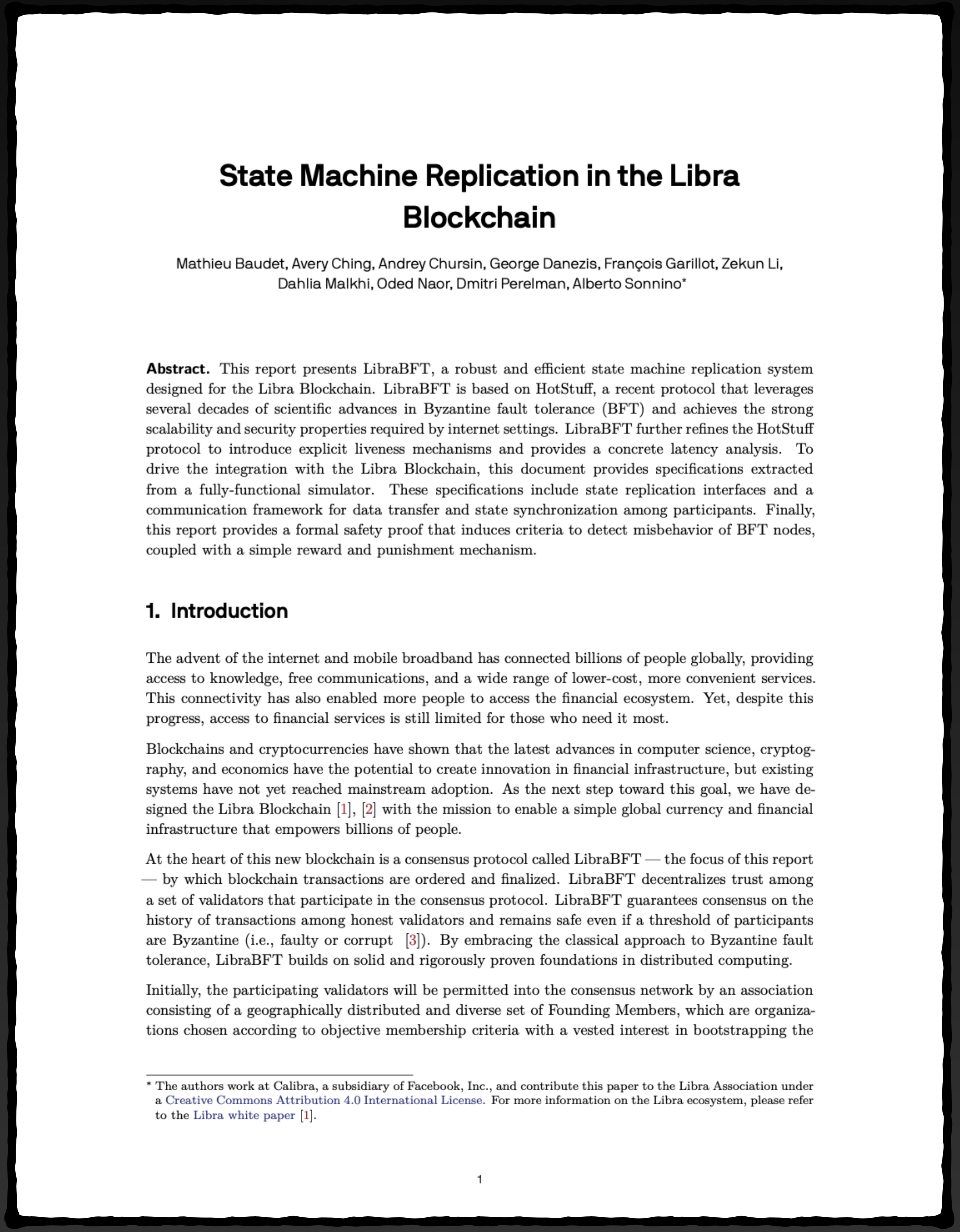
✓ Clearly isolated components

HashGraph

✗ Impossible to garbage collect

✗ Unclear block synchroniser

The first 6 months...



State Machine Replication in the Libra Blockchain

Mathieu Baudet, Avery Ching, Andrey Chursin, George Danezis, François Garillot, Zekun Li, Dahlia Malkhi, Oded Naor, Dmitri Perelman, Alberto Sonnino*

Abstract. This report presents LibraBFT, a robust and efficient state machine replication system designed for the Libra Blockchain. LibraBFT is based on HotStuff, a recent protocol that leverages several decades of scientific advances in Byzantine fault tolerance (BFT) and achieves the strong scalability and security properties required by internet settings. LibraBFT further refines the HotStuff protocol to introduce explicit liveness mechanisms and provides a concrete latency analysis. To drive the integration with the Libra Blockchain, this document provides specifications extracted from a fully-functional simulator. These specifications include state replication interfaces and a communication framework for data transfer and state synchronization among participants. Finally, this report provides a formal safety proof that induces criteria to detect misbehavior of BFT nodes, coupled with a simple reward and punishment mechanism.

1. Introduction

The advent of the internet and mobile broadband has connected billions of people globally, providing access to knowledge, free communications, and a wide range of lower-cost, more convenient services. This connectivity has also enabled more people to access the financial ecosystem. Yet, despite this progress, access to financial services is still limited for those who need it most.

Blockchains and cryptocurrencies have shown that the latest advances in computer science, cryptography, and economics have the potential to create innovation in financial infrastructure, but existing systems have not yet reached mainstream adoption. As the next step toward this goal, we have designed the Libra Blockchain [1], [2] with the mission to enable a simple global currency and financial infrastructure that empowers billions of people.

At the heart of this new blockchain is a consensus protocol called LibraBFT — the focus of this report — by which blockchain transactions are ordered and finalized. LibraBFT decentralizes trust among a set of validators that participate in the consensus protocol. LibraBFT guarantees consensus on the history of transactions among honest validators and remains safe even if a threshold of participants are Byzantine (i.e., faulty or corrupt [3]). By embracing the classical approach to Byzantine fault tolerance, LibraBFT builds on solid and rigorously proven foundations in distributed computing.

Initially, the participating validators will be permitted into the consensus network by an association consisting of a geographically distributed and diverse set of Founding Members, which are organizations chosen according to objective membership criteria with a vested interest in bootstrapping the

* The authors work at Calibra, a subsidiary of Facebook, Inc., and contribute this paper to the Libra Association under a Creative Commons Attribution 4.0 International License. For more information on the Libra ecosystem, please refer to the Libra white paper [1].

1

SMR in the Libra Blockchain

- The LibraBFT/DiemBFT pacemaker
- Codesign the pacemaker with the rest

Research Questions

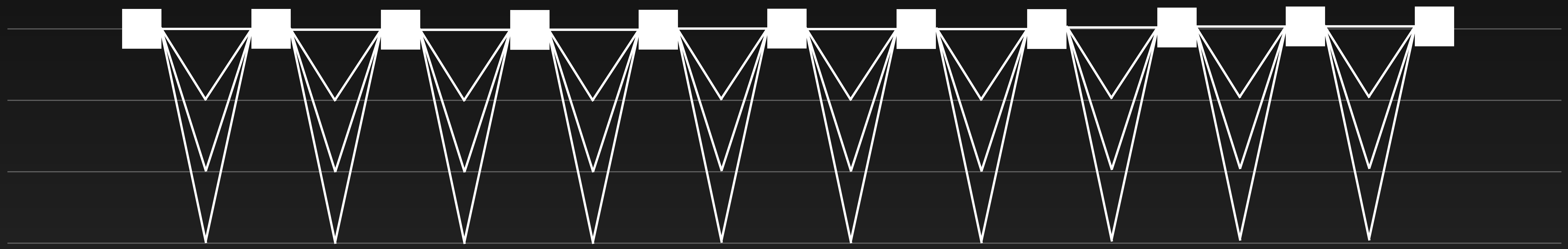
1. Network model?

Lessons Learned

1. Modularisation is a design strategy

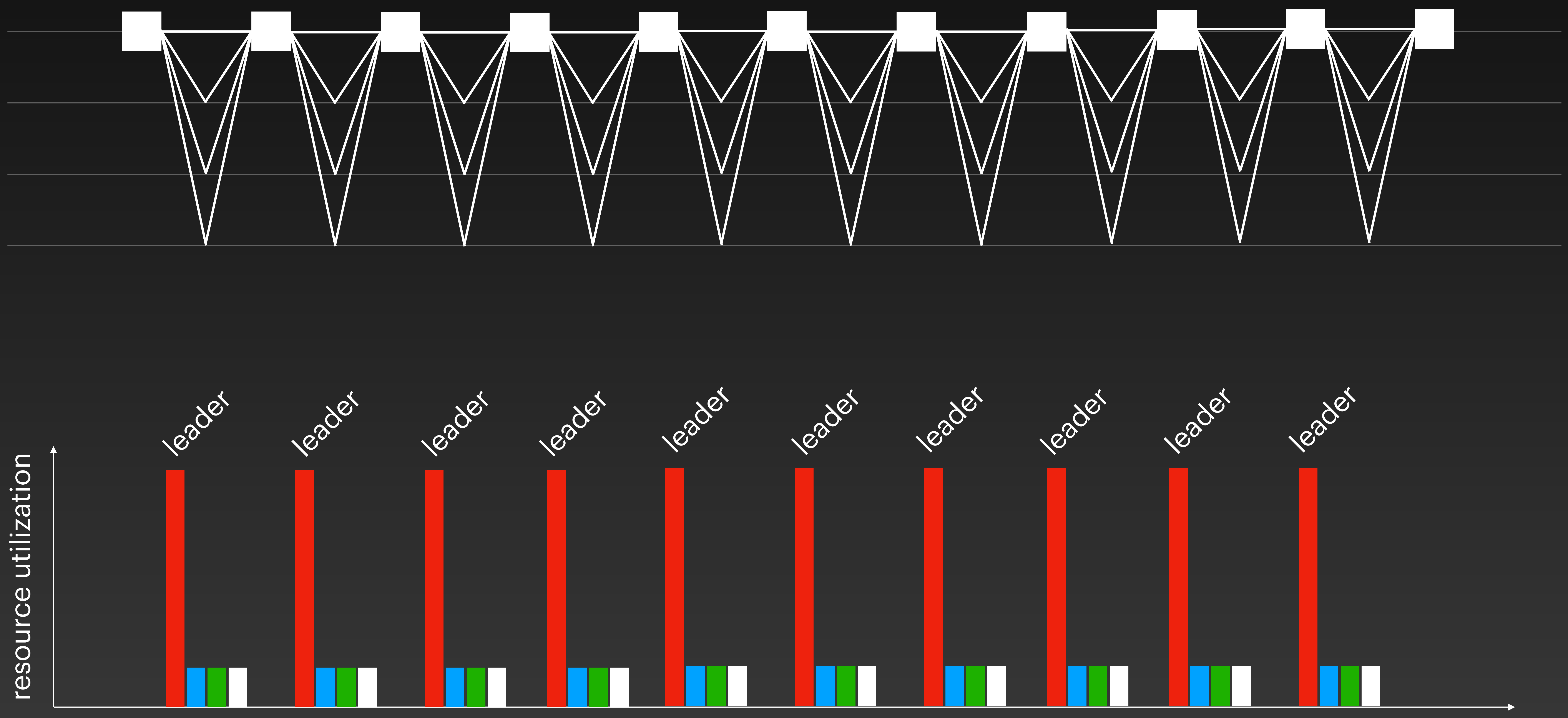
HotStuff

Typical leader-based protocols



Naive Implementation

Uneven resource utilisation



Research Questions

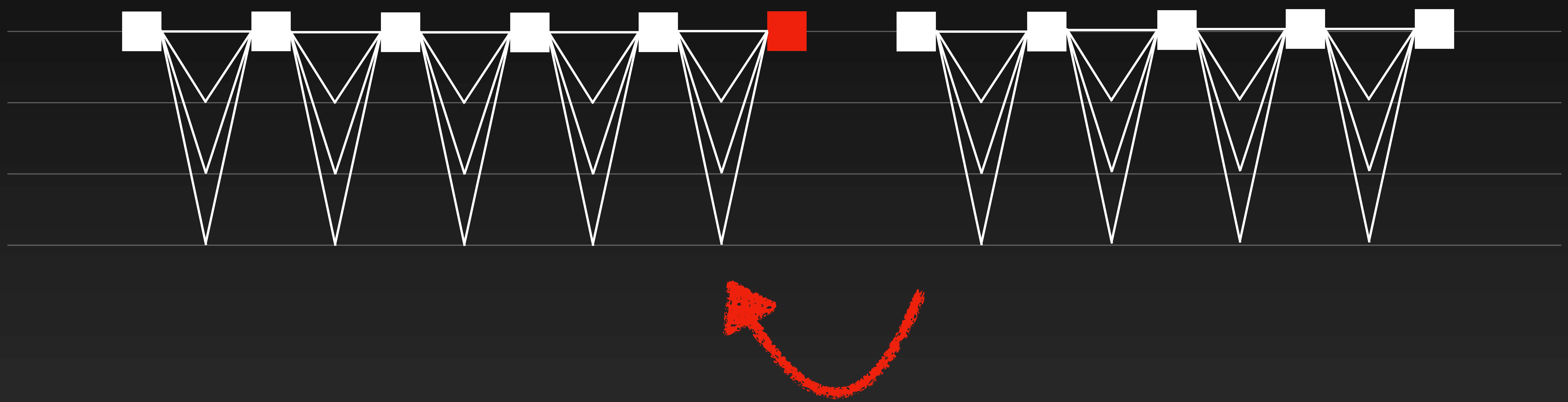
1. Network model?

Lessons Learned

1. Modularisation is a design strategy
2. Tasks-threads allocation

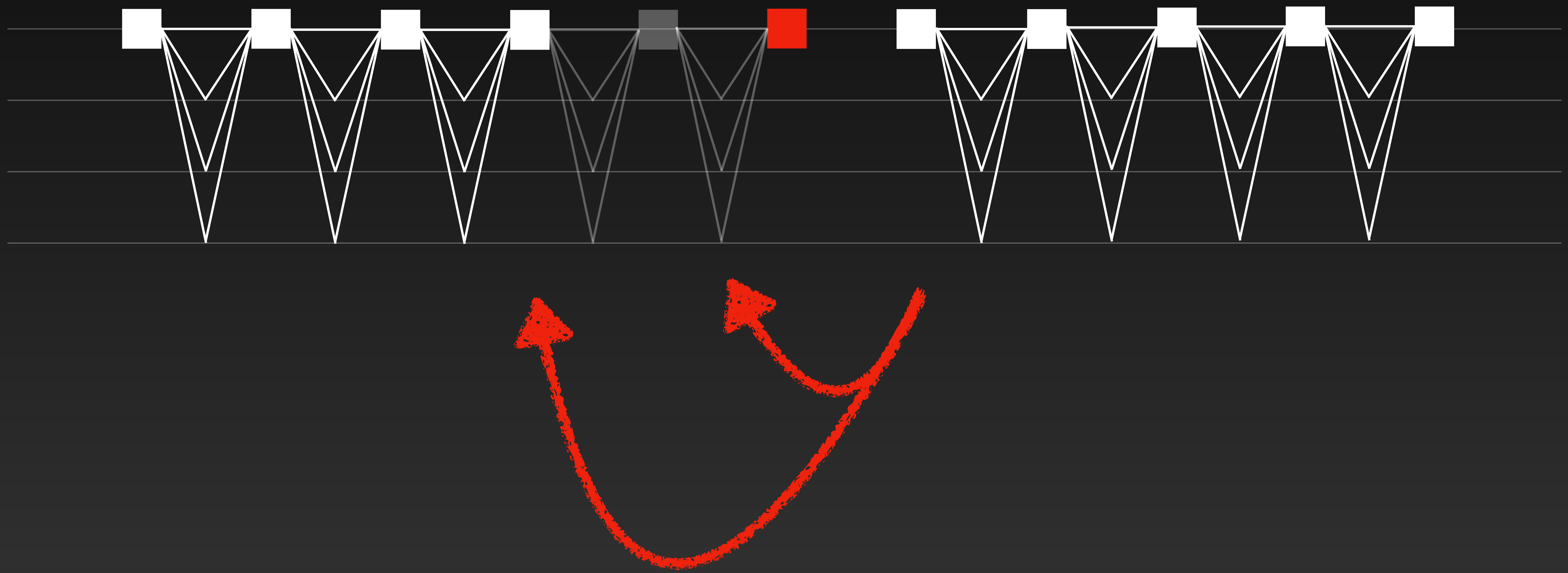
Leader-Driven Consensus

Fragility to faults and asynchrony

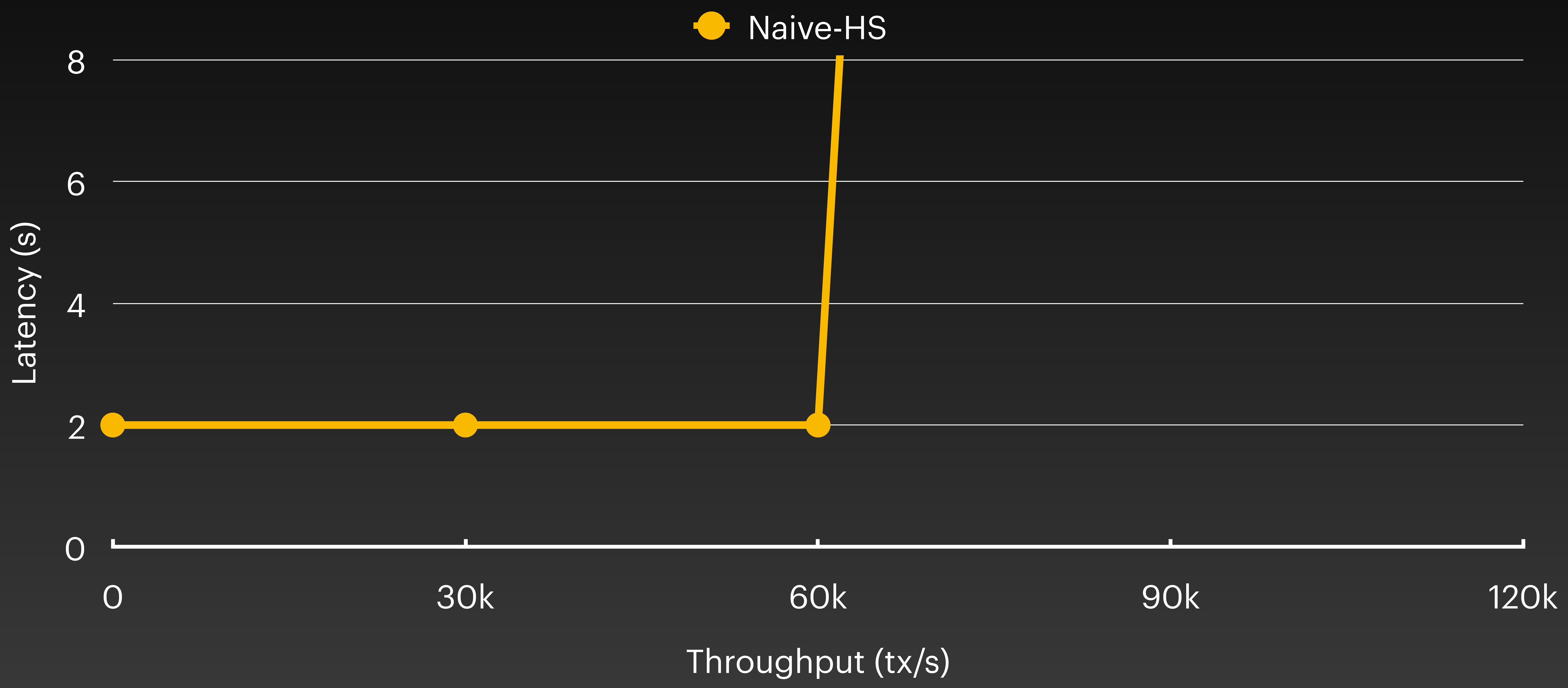


Leader-Driven Consensus

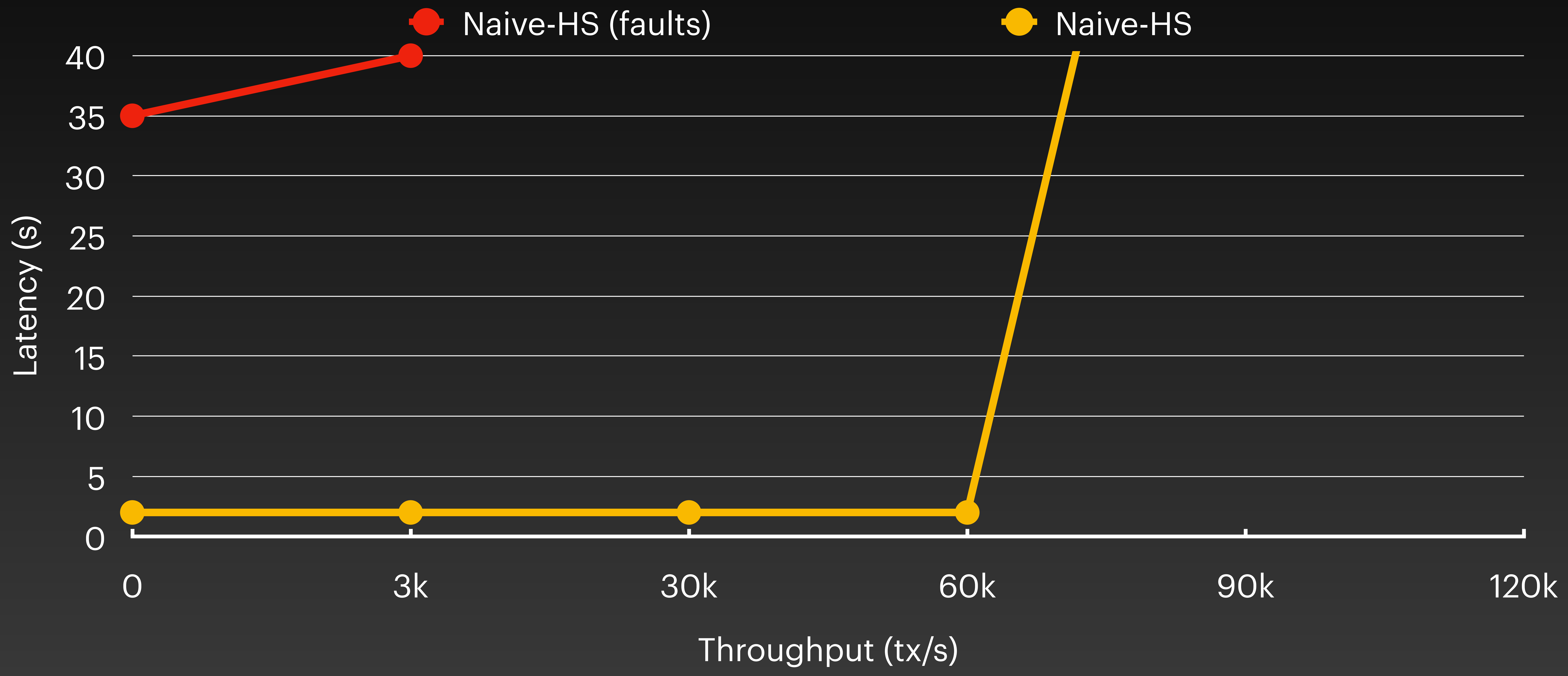
Fragility to faults and asynchrony



Performance



Performance



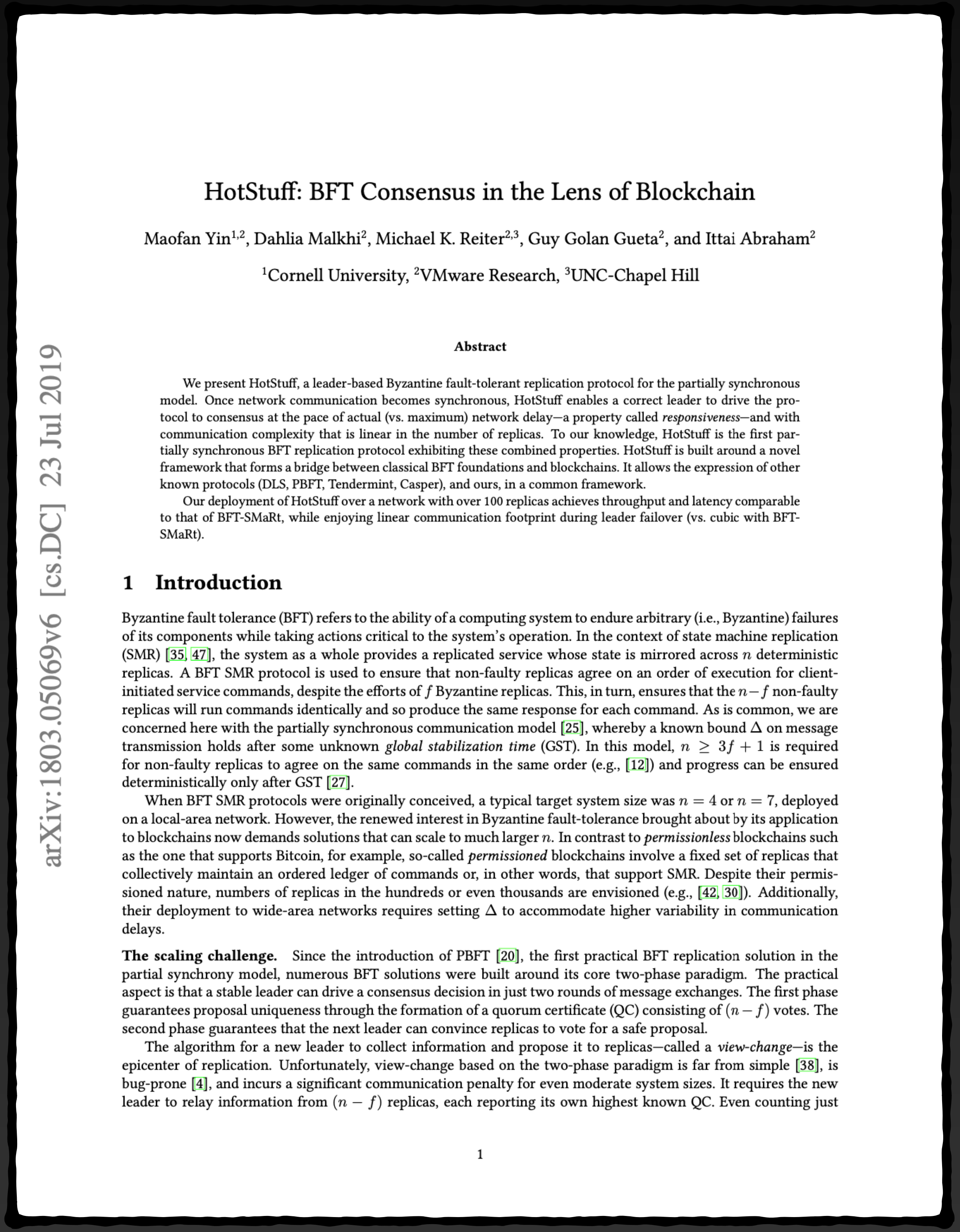
Research Questions

1. Network model?

Lessons Learned

1. Modularisation is a design strategy
2. Tasks-threads allocation
3. Benchmark early

Libra, 2019



HotStuff: BFT Consensus in the Lens of Blockchain

Maofan Yin[1,2], Dahlia Malkhi[2], Michael K. Reiter[2,3], Guy Golan Gueta[2], and Ittai Abraham[2]

[1]Cornell University, [2]VMware Research, [3]UNC-Chapel Hill

arXiv:1803.05069v6 [cs.DC] 23 Jul 2019

Abstract

We present HotStuff, a leader-based Byzantine fault-tolerant replication protocol for the partially synchronous model. Once network communication becomes synchronous, HotStuff enables a correct leader to drive the protocol to consensus at the pace of actual (vs. maximum) network delay—a property called *responsiveness*—and with communication complexity that is linear in the number of replicas. To our knowledge, HotStuff is the first partially synchronous BFT replication protocol exhibiting these combined properties. HotStuff is built around a novel framework that forms a bridge between classical BFT foundations and blockchains. It allows the expression of other known protocols (DLS, PBFT, Tendermint, Casper), and ours, in a common framework.

Our deployment of HotStuff over a network with over 100 replicas achieves throughput and latency comparable to that of BFT-SMaRt, while enjoying linear communication footprint during leader failover (vs. cubic with BFT-SMaRt).

1 Introduction

Byzantine fault tolerance (BFT) refers to the ability of a computing system to endure arbitrary (i.e., Byzantine) failures of its components while taking actions critical to the system's operation. In the context of state machine replication (SMR) [35, 47], the system as a whole provides a replicated service whose state is mirrored across n deterministic replicas. A BFT SMR protocol is used to ensure that non-faulty replicas agree on an order of execution for client-initiated service commands, despite the efforts of f Byzantine replicas. This, in turn, ensures that the $n-f$ non-faulty replicas will run commands identically and so produce the same response for each command. As is common, we are concerned here with the partially synchronous communication model [25], whereby a known bound Δ on message transmission holds after some unknown *global stabilization time* (GST). In this model, $n \geq 3f+1$ is required for non-faulty replicas to agree on the same commands in the same order (e.g., [12]) and progress can be ensured deterministically only after GST [27].

When BFT SMR protocols were originally conceived, a typical target system size was $n = 4$ or $n = 7$, deployed on a local-area network. However, the renewed interest in Byzantine fault-tolerance brought about by its application to blockchains now demands solutions that can scale to much larger n. In contrast to *permissionless* blockchains such as the one that supports Bitcoin, for example, so-called *permissioned* blockchains involve a fixed set of replicas that collectively maintain an ordered ledger of commands or, in other words, that support SMR. Despite their permissioned nature, numbers of replicas in the hundreds or even thousands are envisioned (e.g., [42, 30]). Additionally, their deployment to wide-area networks requires setting Δ to accommodate higher variability in communication delays.

The scaling challenge. Since the introduction of PBFT [20], the first practical BFT replication solution in the partial synchrony model, numerous BFT solutions were built around its core two-phase paradigm. The practical aspect is that a stable leader can drive a consensus decision in just two rounds of message exchanges. The first phase guarantees proposal uniqueness through the formation of a quorum certificate (QC) consisting of $(n-f)$ votes. The second phase guarantees that the next leader can convince replicas to vote for a safe proposal.

The algorithm for a new leader to collect information and propose it to replicas—called a *view-change*—is the epicenter of replication. Unfortunately, view-change based on the two-phase paradigm is far from simple [38], is bug-prone [4], and incurs a significant communication penalty for even moderate system sizes. It requires the new leader to relay information from $(n-f)$ replicas, each reporting its own highest known QC. Even counting just

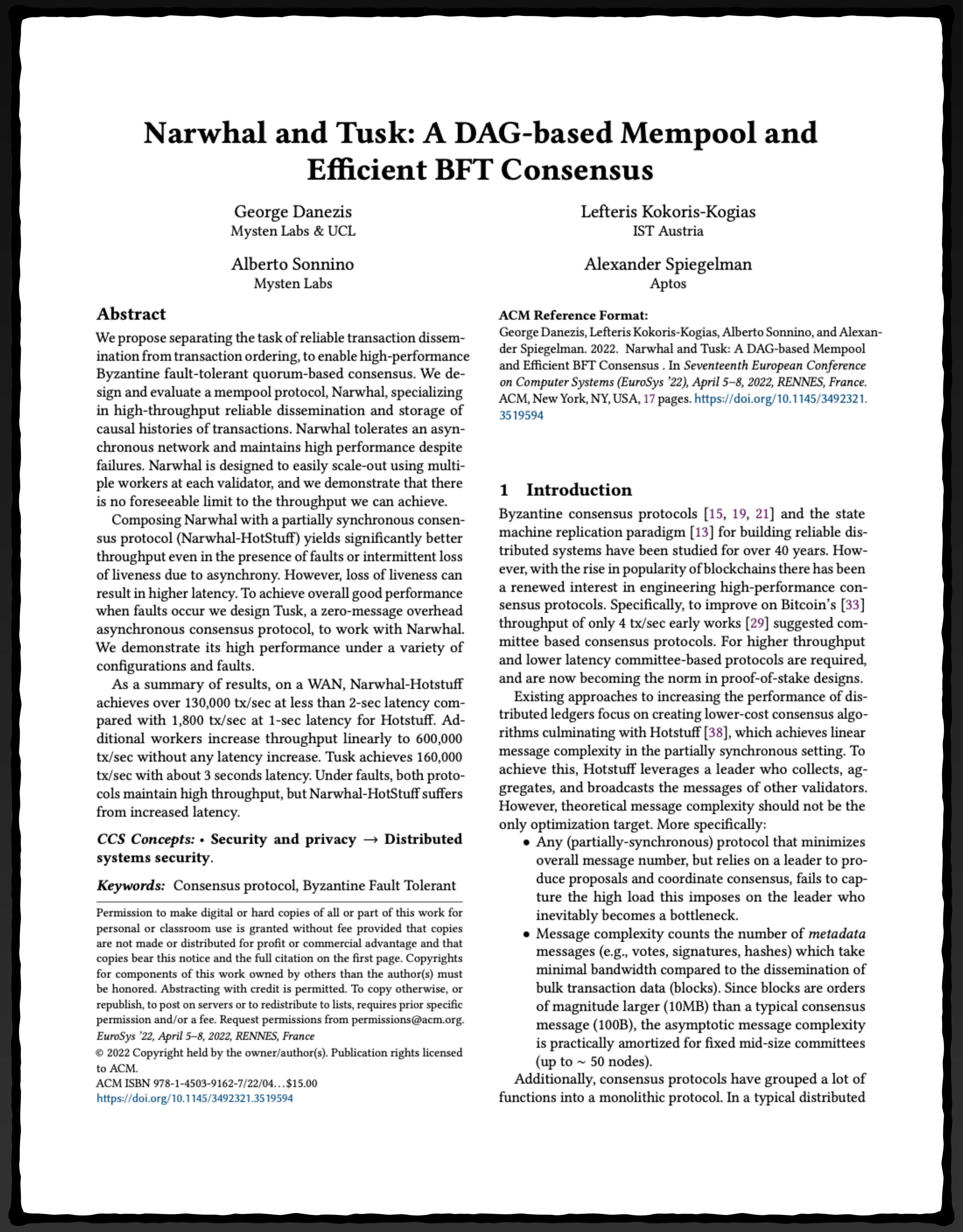
1

HotStuff (naive mempool)

- Linear
- Clearly isolated components

- Uneven resource utilisation
- Fragile to faults and asynchrony
- Unspecified components (pacemaker)

Libra, 2021



Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus

George Danezis, Mysten Labs & UCL
Lefteris Kokoris-Kogias, IST Austria
Alberto Sonnino, Mysten Labs
Alexander Spiegelman, Aptos

Abstract

We propose separating the task of reliable transaction dissemination from transaction ordering, to enable high-performance Byzantine fault-tolerant quorum-based consensus. We design and evaluate a mempool protocol, Narwhal, specializing in high-throughput reliable dissemination and storage of causal histories of transactions. Narwhal tolerates an asynchronous network and maintains high performance despite failures. Narwhal is designed to easily scale-out using multiple workers at each validator, and we demonstrate that there is no foreseeable limit to the throughput we can achieve.

Composing Narwhal with a partially synchronous consensus protocol (Narwhal-HotStuff) yields significantly better throughput even in the presence of faults or intermittent loss of liveness due to asynchrony. However, loss of liveness can result in higher latency. To achieve overall good performance when faults occur we design Tusk, a zero-message overhead asynchronous consensus protocol, to work with Narwhal. We demonstrate its high performance under a variety of configurations and faults.

As a summary of results, on a WAN, Narwhal-Hotstuff achieves over 130,000 tx/sec at less than 2-sec latency compared with 1,800 tx/sec at 1-sec latency for Hotstuff. Additional workers increase throughput linearly to 600,000 tx/sec without any latency increase. Tusk achieves 160,000 tx/sec with about 3 seconds latency. Under faults, both protocols maintain high throughput, but Narwhal-HotStuff suffers from increased latency.

CCS Concepts: • **Security and privacy → Distributed systems security.**

Keywords: Consensus protocol, Byzantine Fault Tolerant

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
EuroSys '22, April 5–8, 2022, RENNES, France
© 2022 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-9162-7/22/04…$15.00
https://doi.org/10.1145/3492321.3519594

ACM Reference Format:
George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus . In *Seventeenth European Conference on Computer Systems (EuroSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3492321.3519594

1 Introduction

Byzantine consensus protocols [15, 19, 21] and the state machine replication paradigm [13] for building reliable distributed systems have been studied for over 40 years. However, with the rise in popularity of blockchains there has been a renewed interest in engineering high-performance consensus protocols. Specifically, to improve on Bitcoin's [33] throughput of only 4 tx/sec early works [29] suggested committee based consensus protocols. For higher throughput and lower latency committee-based protocols are required, and are now becoming the norm in proof-of-stake designs.

Existing approaches to increasing the performance of distributed ledgers focus on creating lower-cost consensus algorithms culminating with Hotstuff [38], which achieves linear message complexity in the partially synchronous setting. To achieve this, Hotstuff leverages a leader who collects, aggregates, and broadcasts the messages of other validators. However, theoretical message complexity should not be the only optimization target. More specifically:

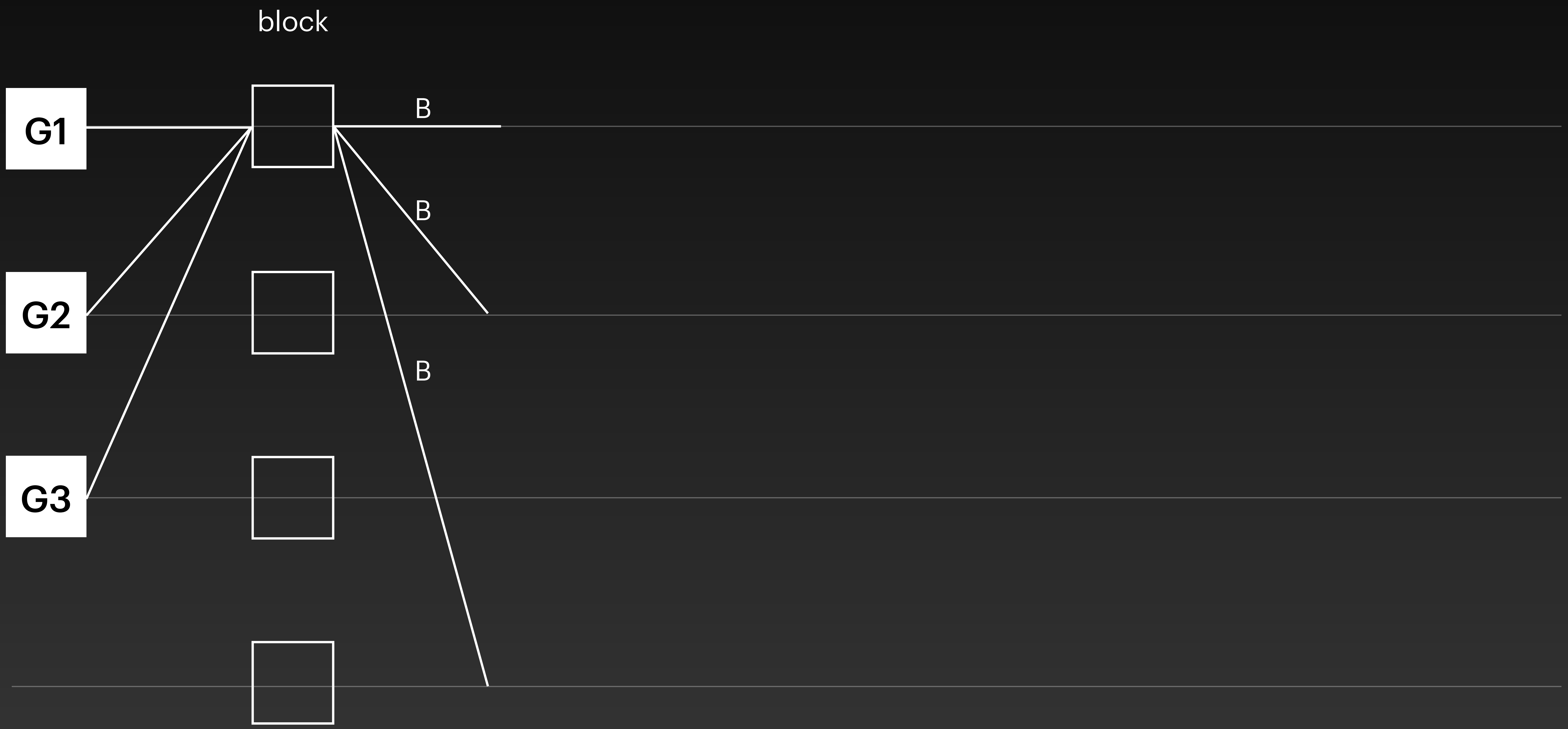
- Any (partially-synchronous) protocol that minimizes overall message number, but relies on a leader to produce proposals and coordinate consensus, fails to capture the high load this imposes on the leader who inevitably becomes a bottleneck.
- Message complexity counts the number of *metadata* messages (e.g., votes, signatures, hashes) which take minimal bandwidth compared to the dissemination of bulk transaction data (blocks). Since blocks are orders of magnitude larger (10MB) than a typical consensus message (100B), the asymptotic message complexity is practically amortized for fixed mid-size committees (up to ~ 50 nodes).

Additionally, consensus protocols have grouped a lot of functions into a monolithic protocol. In a typical distributed

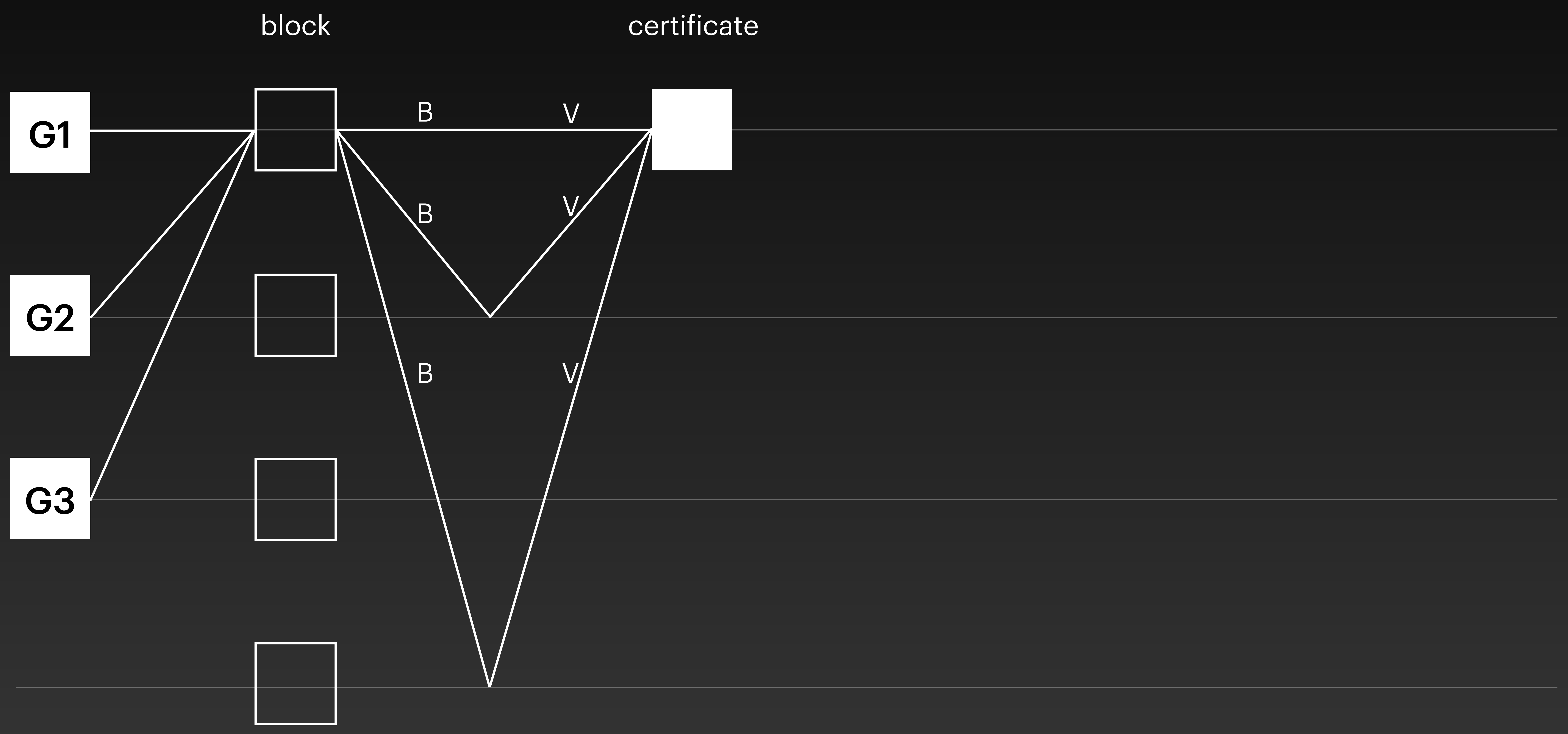
Narwhal

- Quadratic but even resource utilisation
- Separation between consensus and data dissemination

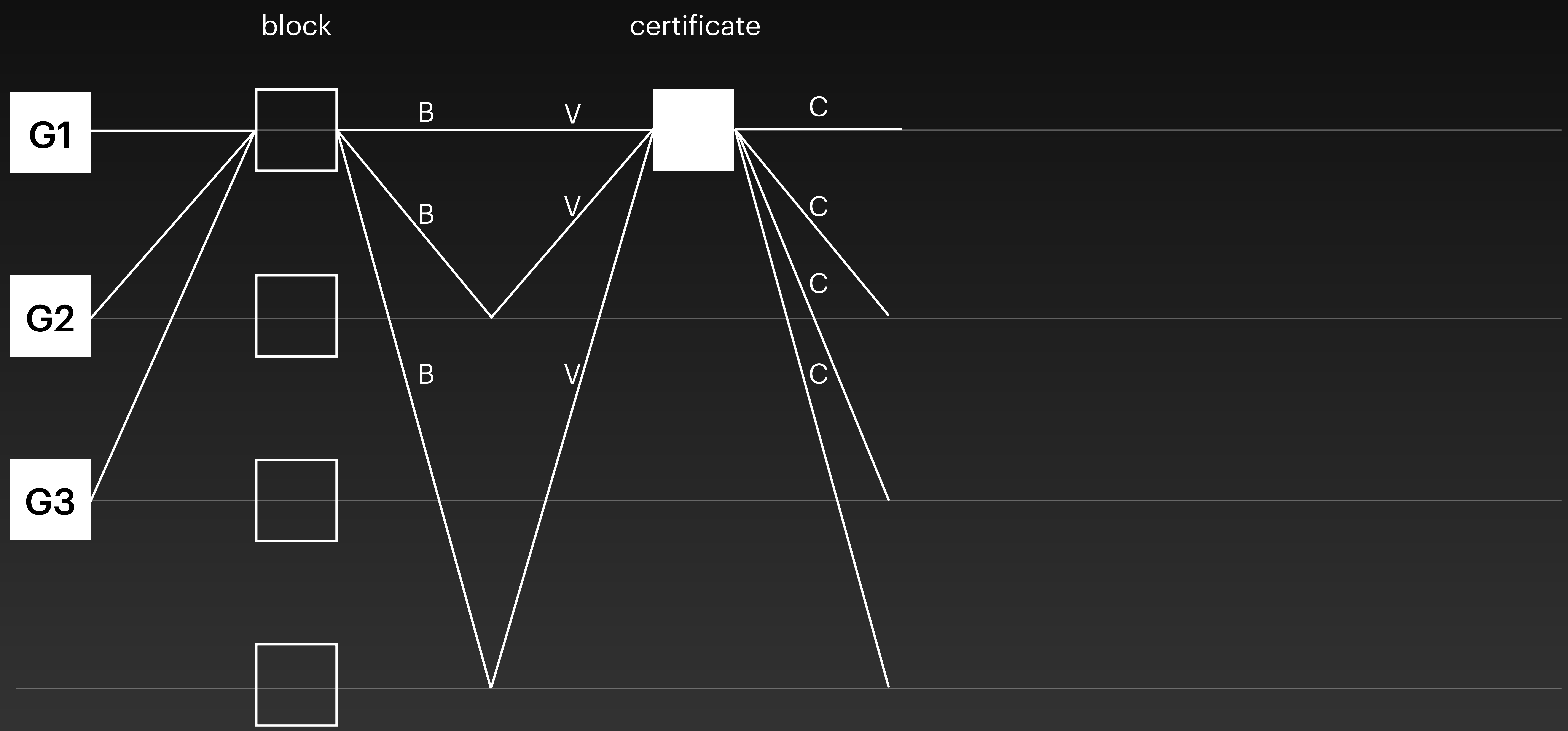
Narwhal



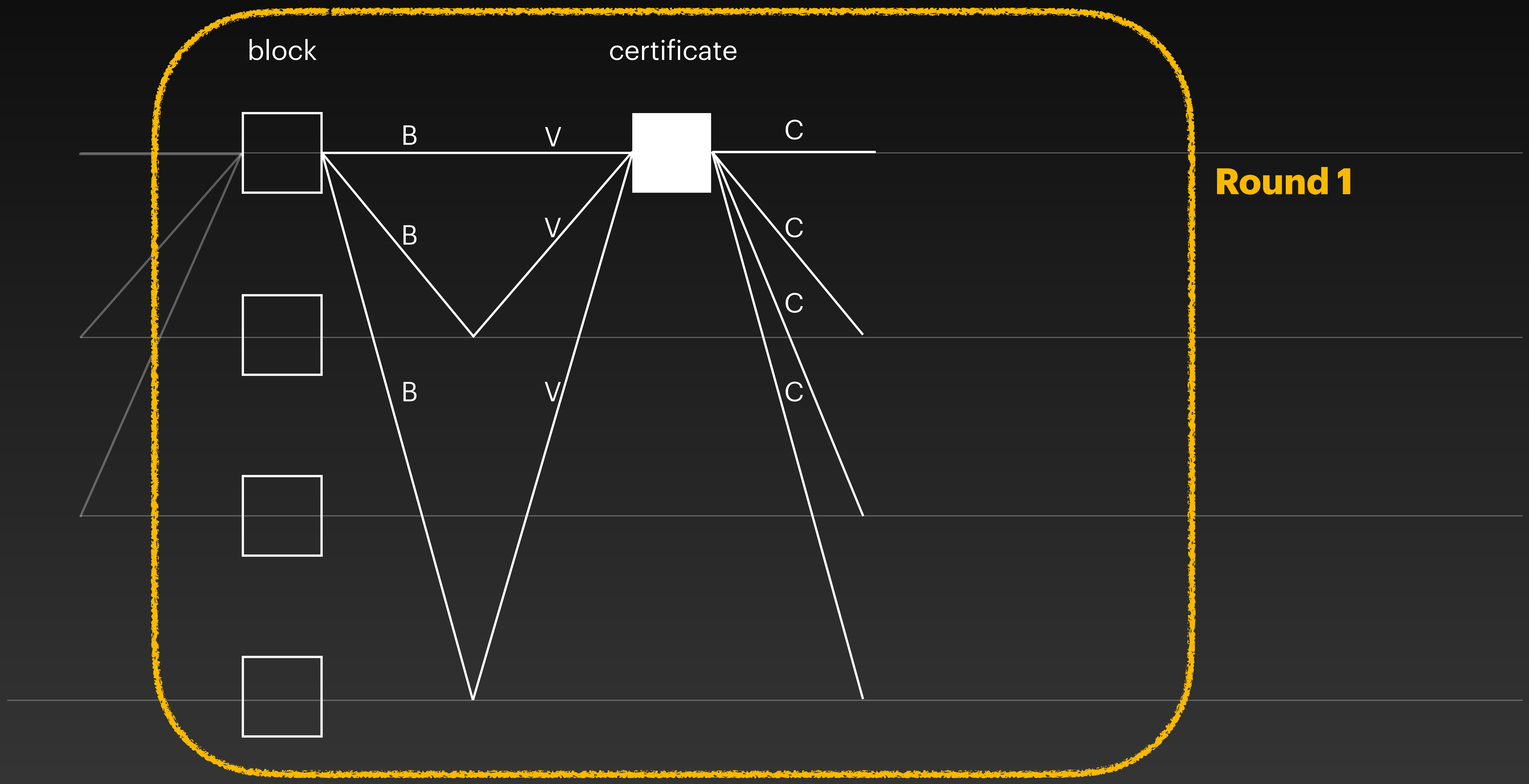
Narwhal



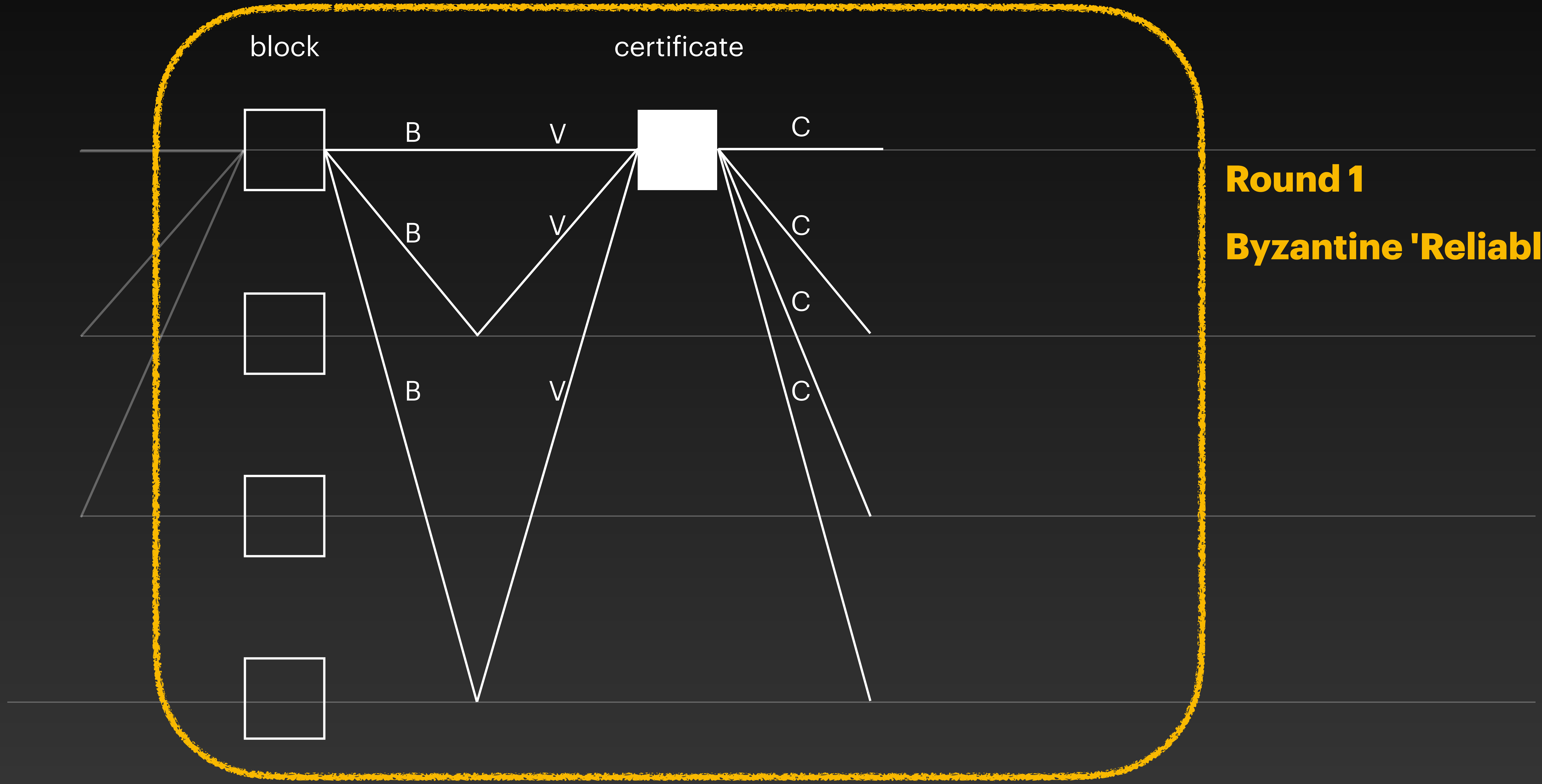
Narwhal



Narwhal



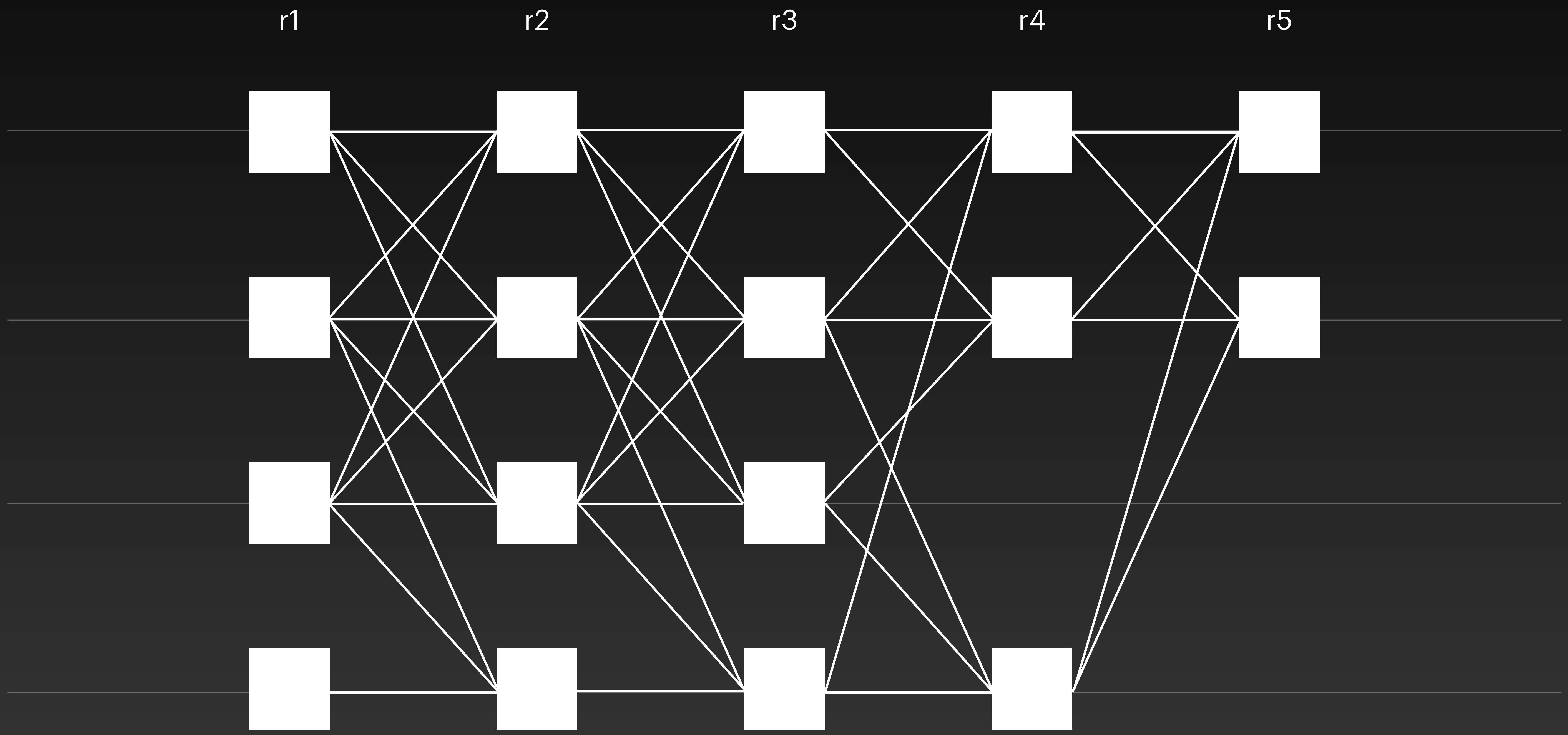
Narwhal



Round 1

Byzantine 'Reliable' Broadcast

Narwhal



Research Questions

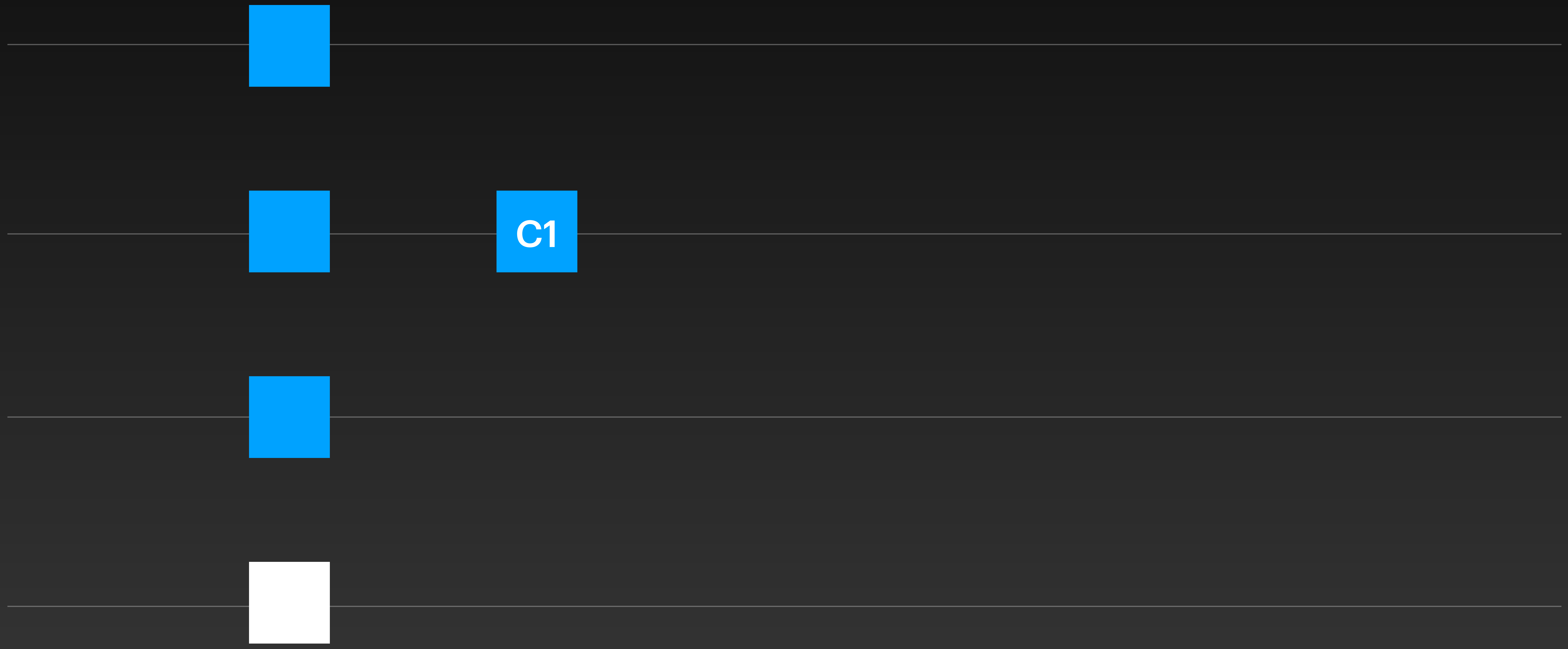
1. Network model?

Lessons Learned

1. Modularisation is a design strategy
2. Tasks-threads allocation
3. Benchmark early
4. Codesign with mem. and storage

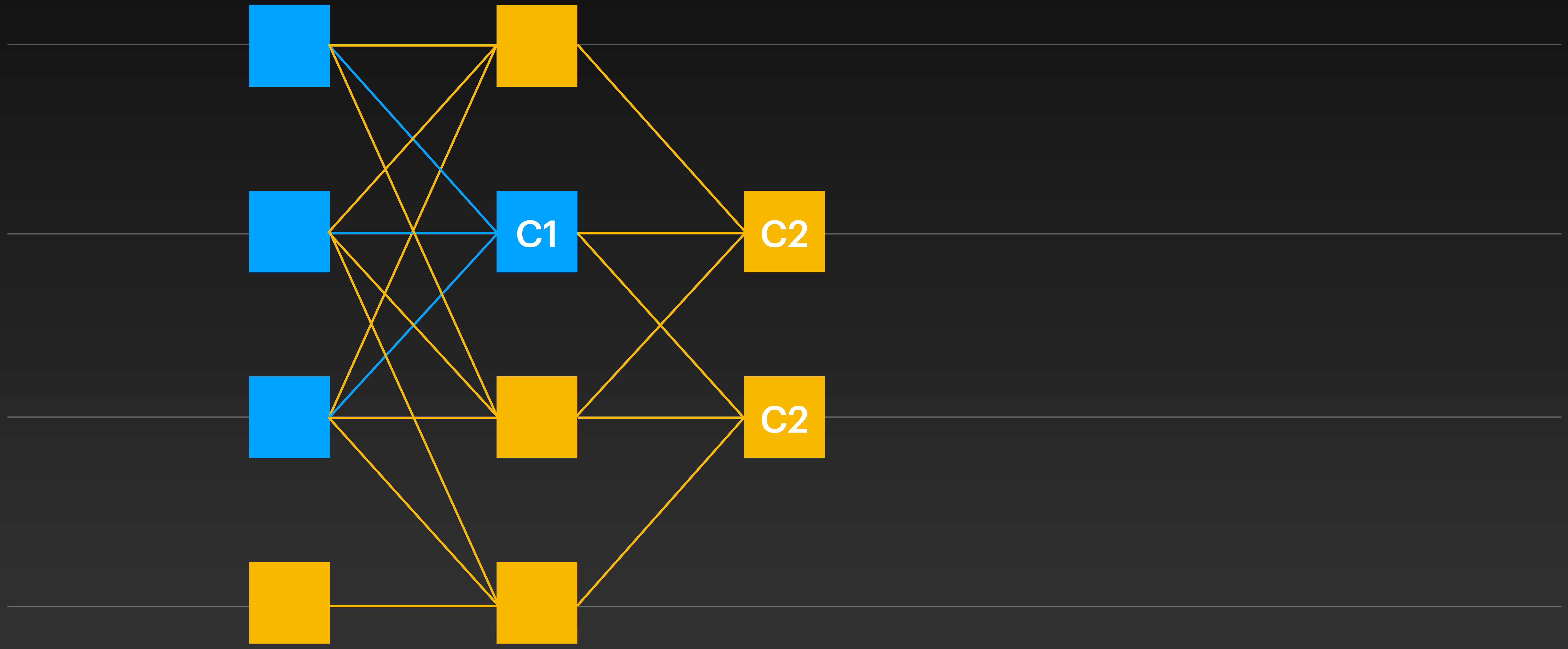
HotStuff on Narwhal

Enhanced commit rule



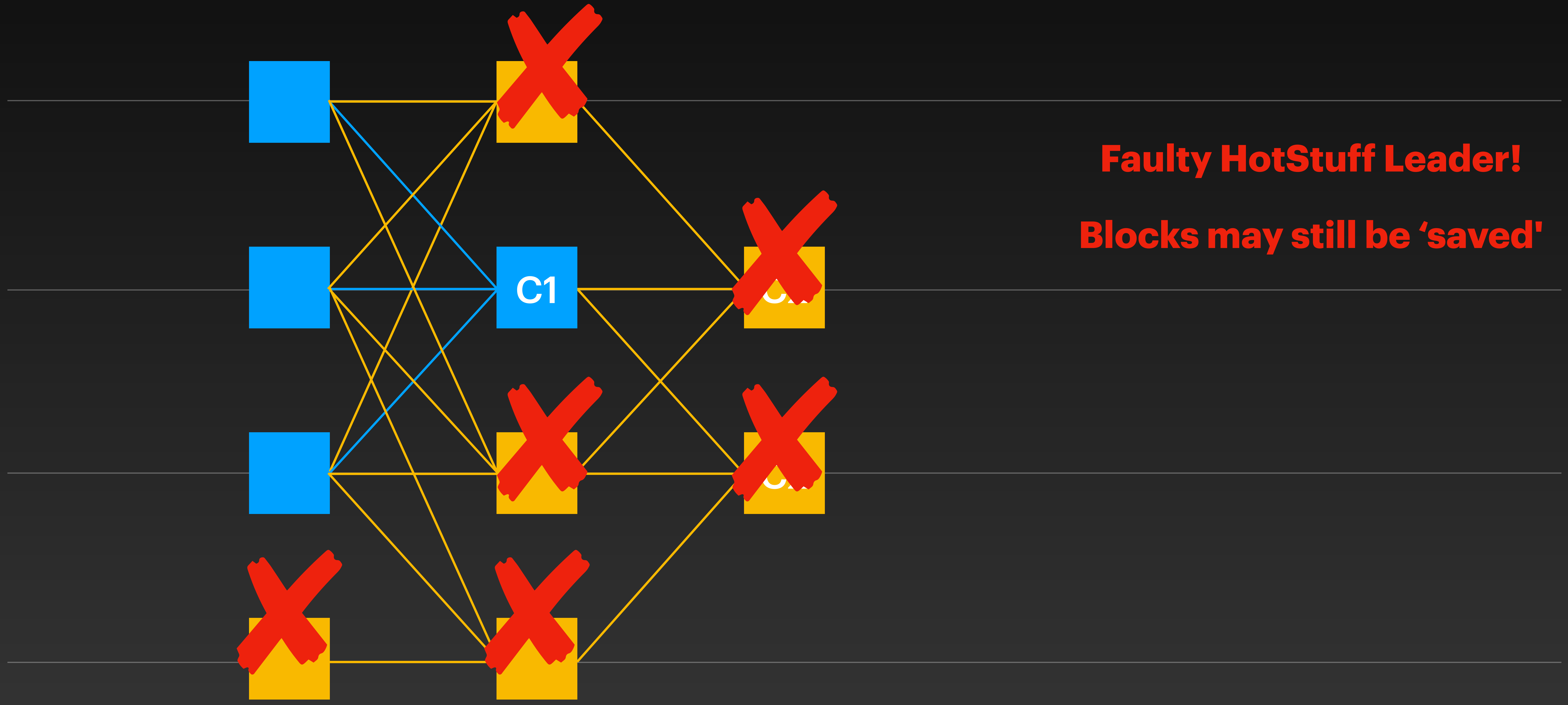
HotStuff on Narwhal

Enhanced commit rule



HotStuff on Narwhal

Enhanced commit rule

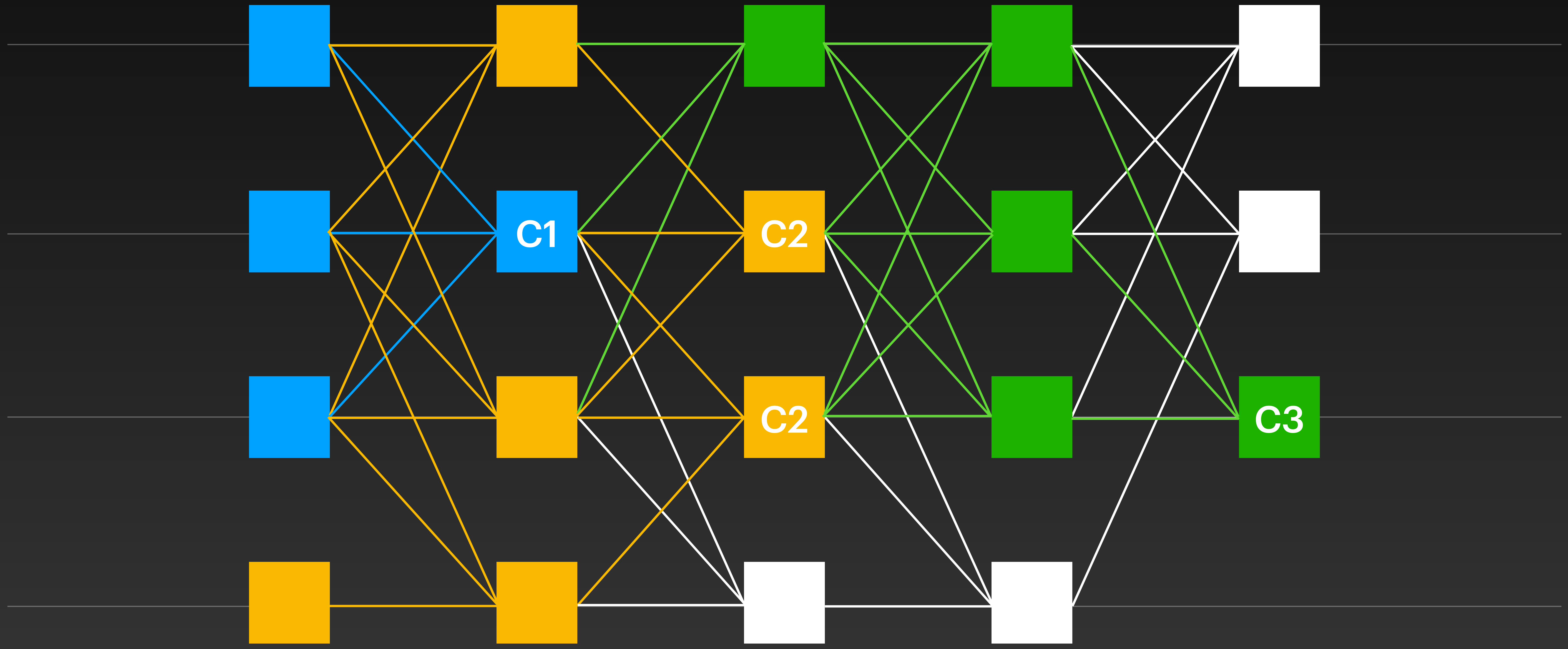


Faulty HotStuff Leader!

Blocks may still be ‘saved'

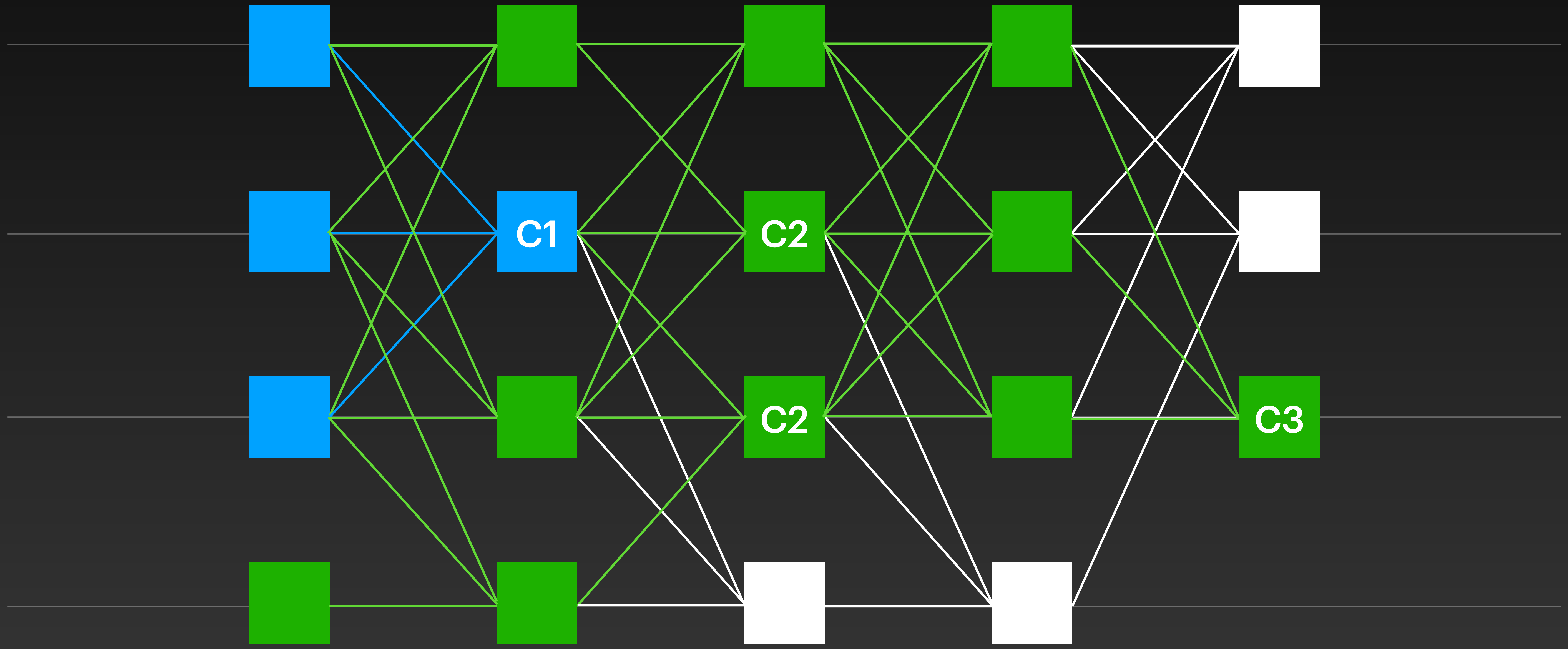
HotStuff on Narwhal

Enhanced commit rule

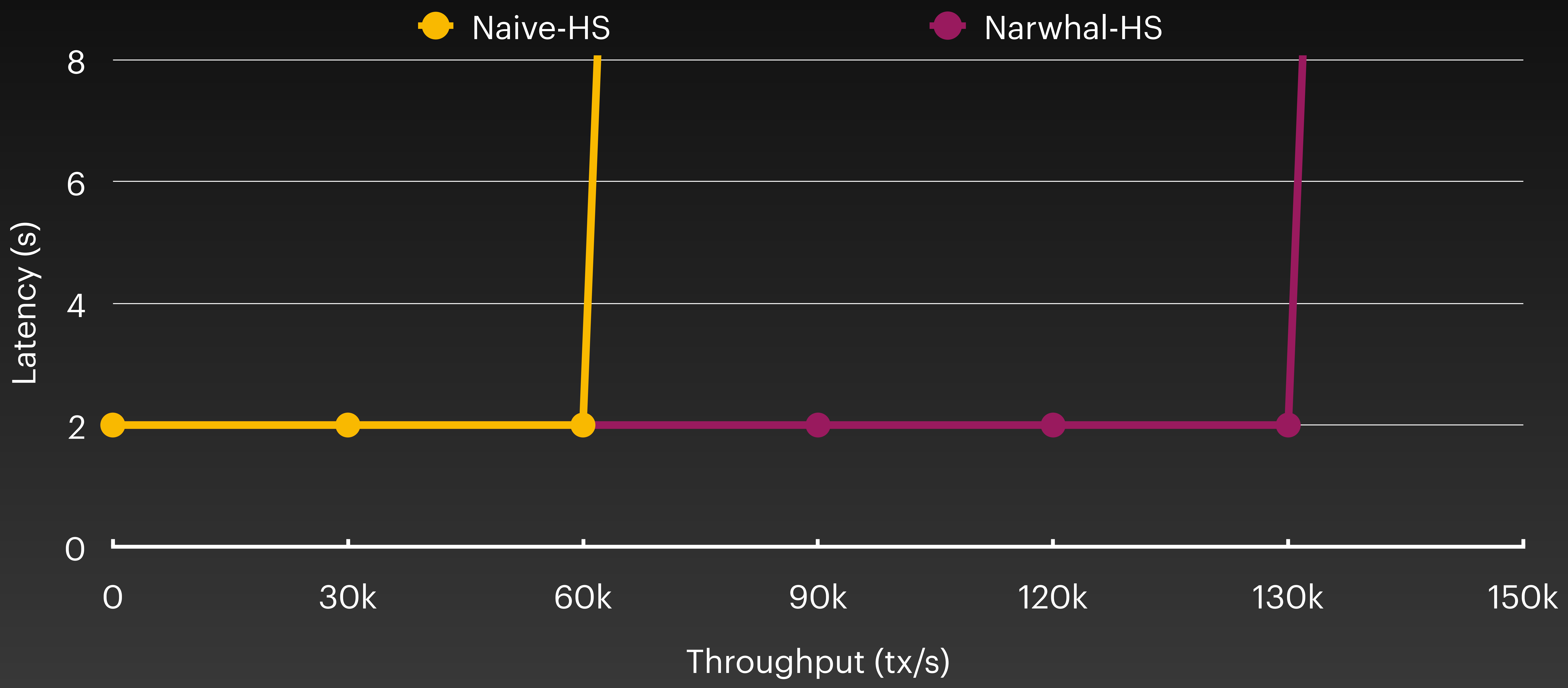


HotStuff on Narwhal

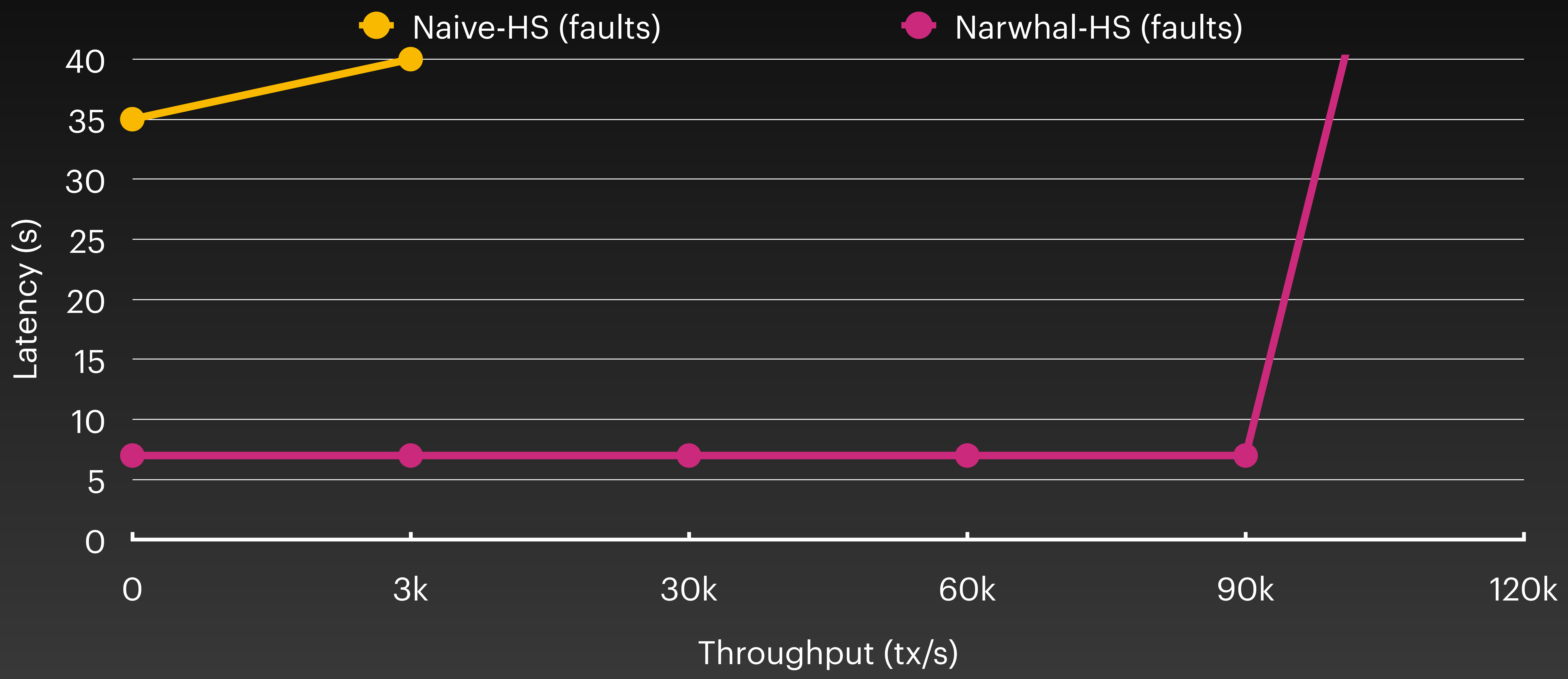
Enhanced commit rule



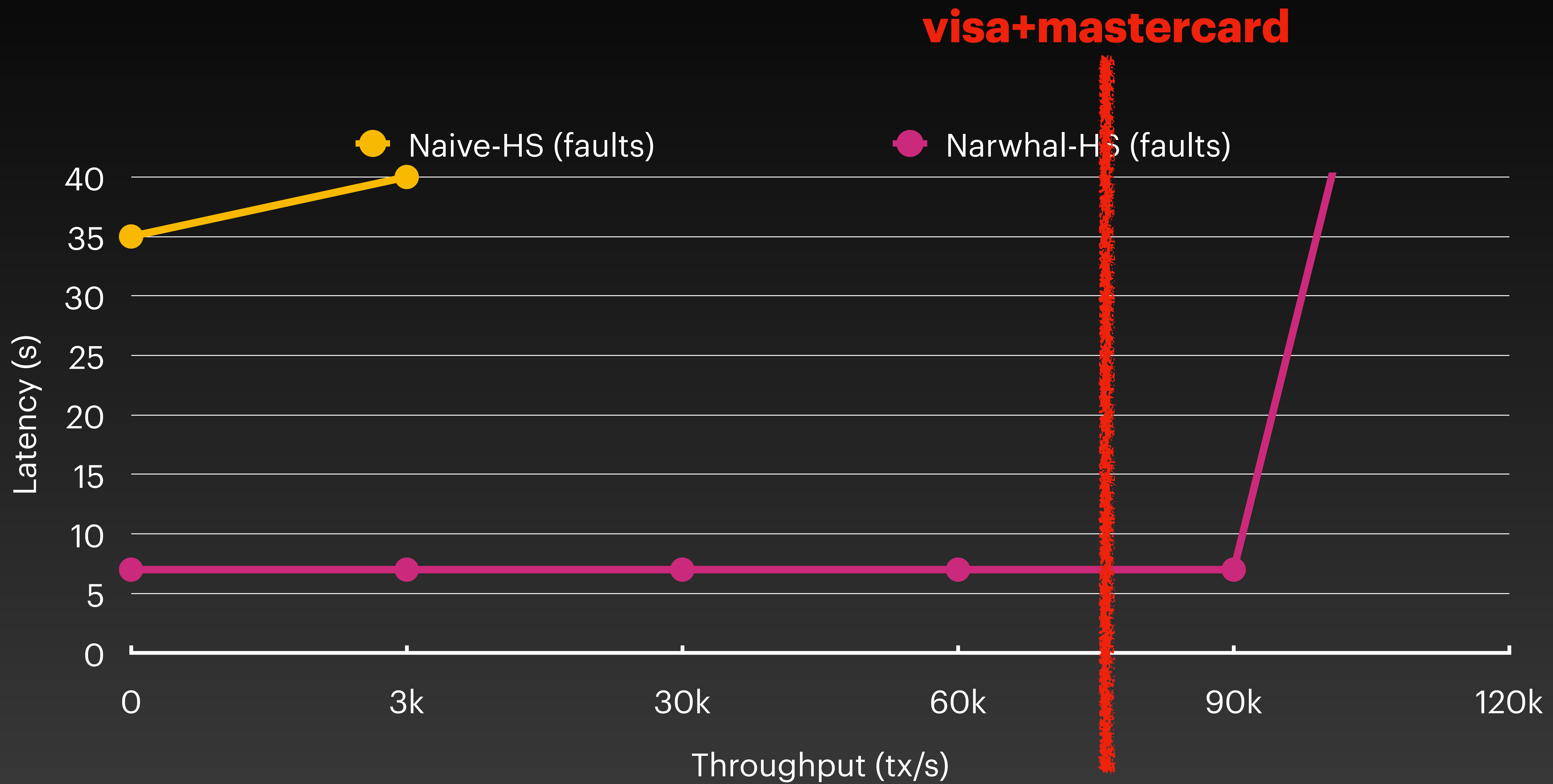
Performance



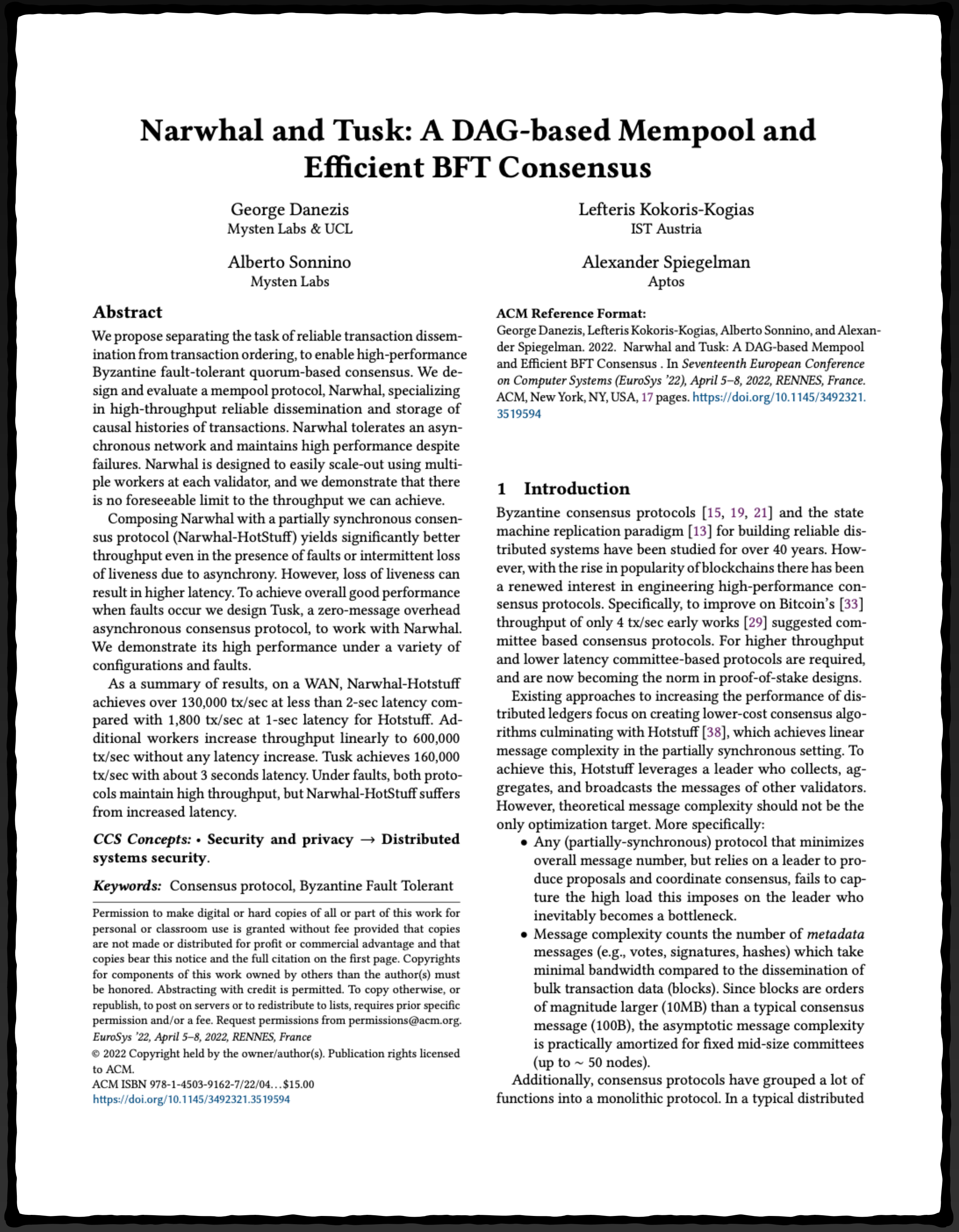
Performance



Performance



Libra, 2021



Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus

George Danezis
Mysten Labs & UCL

Lefteris Kokoris-Kogias
IST Austria

Alberto Sonnino
Mysten Labs

Alexander Spiegelman
Aptos

Abstract

We propose separating the task of reliable transaction dissemination from transaction ordering, to enable high-performance Byzantine fault-tolerant quorum-based consensus. We design and evaluate a mempool protocol, Narwhal, specializing in high-throughput reliable dissemination and storage of causal histories of transactions. Narwhal tolerates an asynchronous network and maintains high performance despite failures. Narwhal is designed to easily scale-out using multiple workers at each validator, and we demonstrate that there is no foreseeable limit to the throughput we can achieve.

Composing Narwhal with a partially synchronous consensus protocol (Narwhal-HotStuff) yields significantly better throughput even in the presence of faults or intermittent loss of liveness due to asynchrony. However, loss of liveness can result in higher latency. To achieve overall good performance when faults occur we design Tusk, a zero-message overhead asynchronous consensus protocol, to work with Narwhal. We demonstrate its high performance under a variety of configurations and faults.

As a summary of results, on a WAN, Narwhal-Hotstuff achieves over 130,000 tx/sec at less than 2-sec latency compared with 1,800 tx/sec at 1-sec latency for Hotstuff. Additional workers increase throughput linearly to 600,000 tx/sec without any latency increase. Tusk achieves 160,000 tx/sec with about 3 seconds latency. Under faults, both protocols maintain high throughput, but Narwhal-HotStuff suffers from increased latency.

CCS Concepts: • **Security and privacy → Distributed systems security**.

Keywords: Consensus protocol, Byzantine Fault Tolerant

EuroSys '22, April 5–8, 2022, RENNES, France

ACM ISBN 978-1-4503-9162-7/22/04…$15.00
https://doi.org/10.1145/3492321.3519594

ACM Reference Format:
George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, and Alexander Spiegelman. 2022. Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus . In *Seventeenth European Conference on Computer Systems (EuroSys '22), April 5–8, 2022, RENNES, France.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3492321.3519594

1 Introduction

Byzantine consensus protocols [15, 19, 21] and the state machine replication paradigm [13] for building reliable distributed systems have been studied for over 40 years. However, with the rise in popularity of blockchains there has been a renewed interest in engineering high-performance consensus protocols. Specifically, to improve on Bitcoin's [33] throughput of only 4 tx/sec early works [29] suggested committee based consensus protocols. For higher throughput and lower latency committee-based protocols are required, and are now becoming the norm in proof-of-stake designs.

Existing approaches to increasing the performance of distributed ledgers focus on creating lower-cost consensus algorithms culminating with Hotstuff [38], which achieves linear message complexity in the partially synchronous setting. To achieve this, Hotstuff leverages a leader who collects, aggregates, and broadcasts the messages of other validators. However, theoretical message complexity should not be the only optimization target. More specifically:

- Any (partially-synchronous) protocol that minimizes overall message number, but relies on a leader to produce proposals and coordinate consensus, fails to capture the high load this imposes on the leader who inevitably becomes a bottleneck.
- Message complexity counts the number of *metadata* messages (e.g., votes, signatures, hashes) which take minimal bandwidth compared to the dissemination of bulk transaction data (blocks). Since blocks are orders of magnitude larger (10MB) than a typical consensus message (100B), the asymptotic message complexity is practically amortized for fixed mid-size committees (up to ~ 50 nodes).

Additionally, consensus protocols have grouped a lot of functions into a monolithic protocol. In a typical distributed

Narwhal

- Quadratic but even resource utilisation
- Separation between consensus and data dissemination
- High engineering complexity

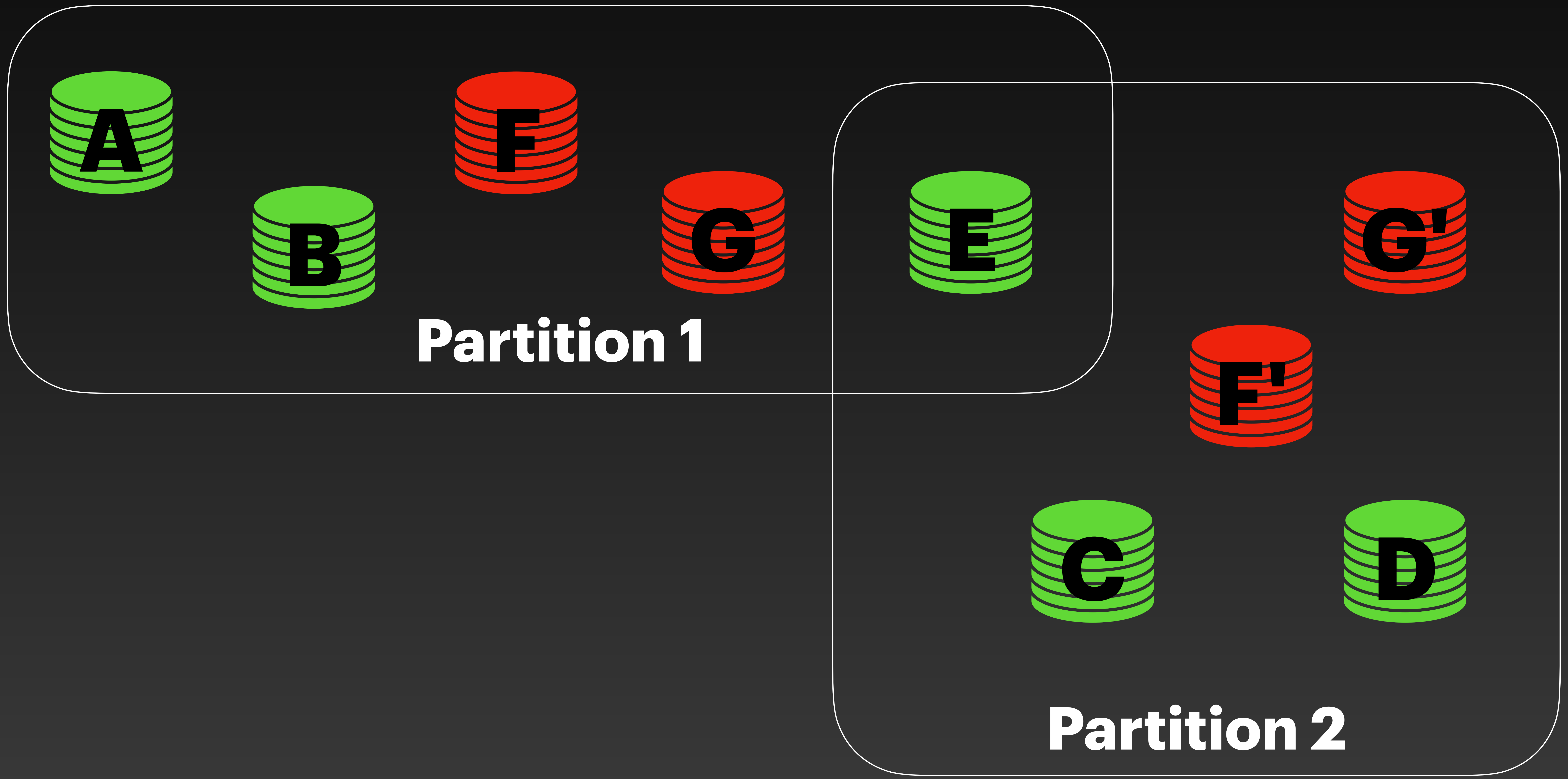
Research Questions

1. Network model?
2. BFT testing?

Lessons Learned

1. Modularisation is a design strategy
2. Tasks-threads allocation
3. Benchmark early
4. Codesign with mem. and storage

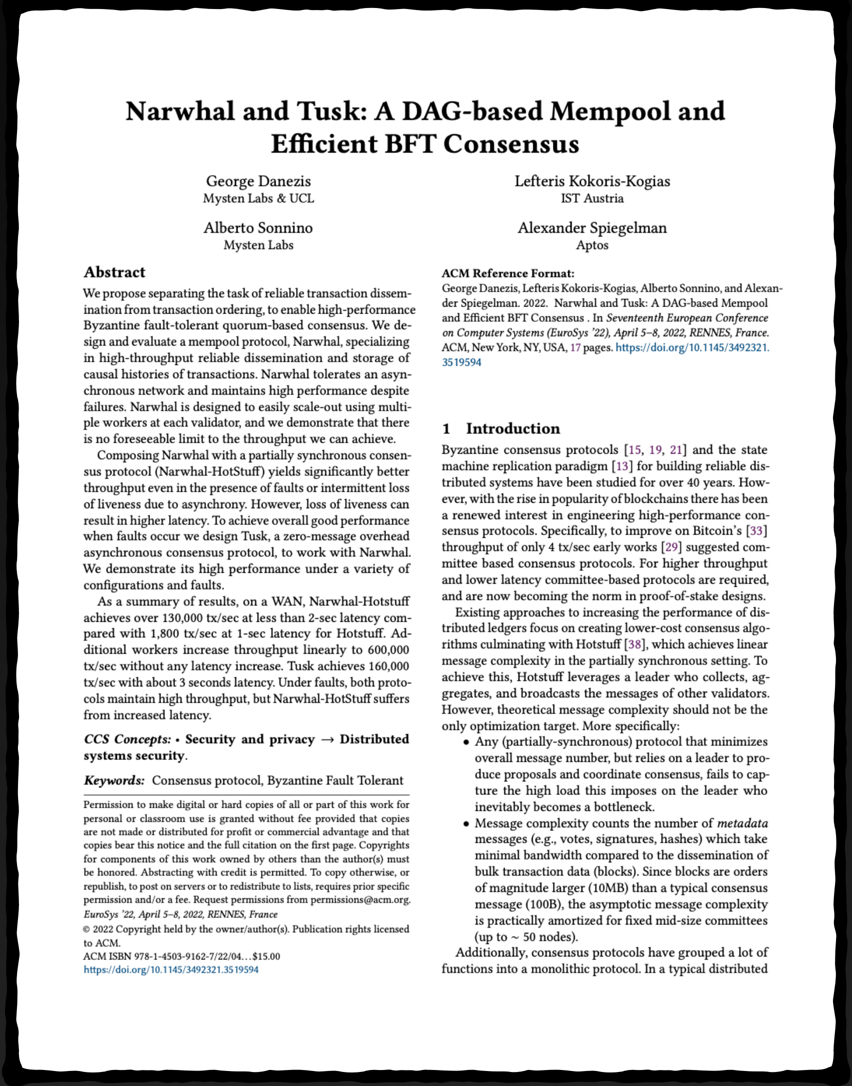
Twins



DagRider


All You Need is DAG

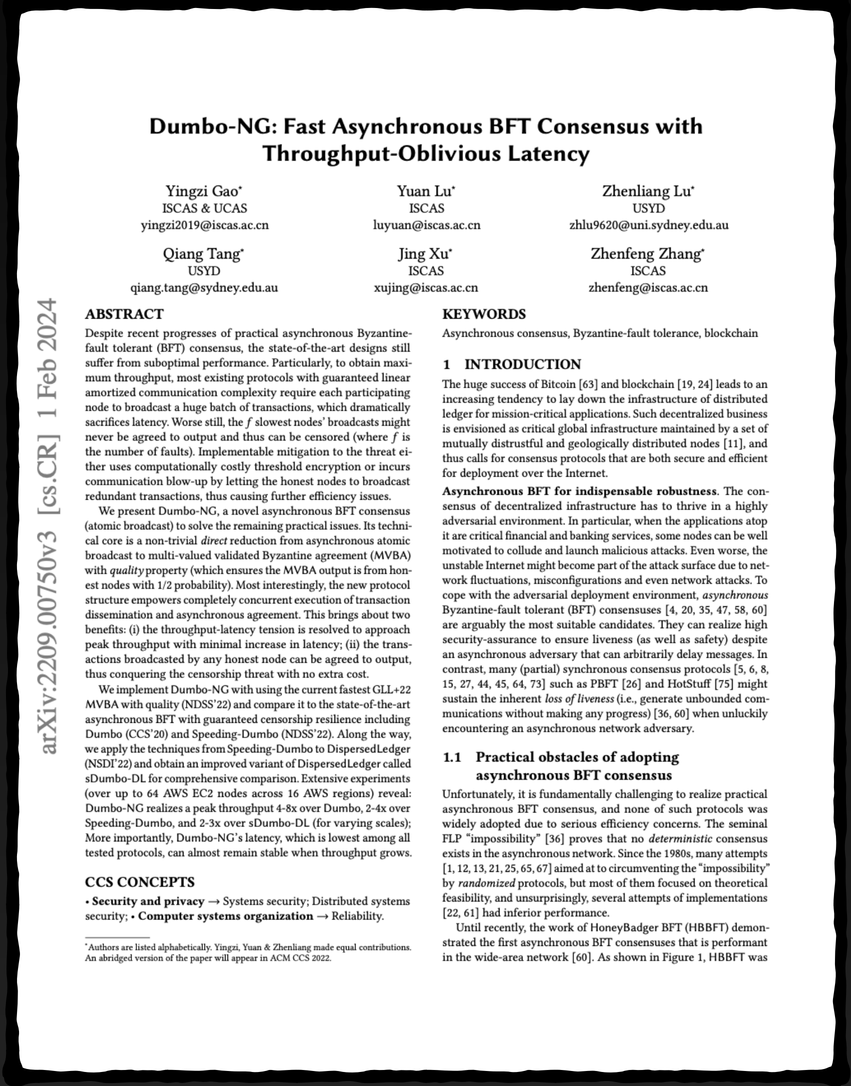
Tusk


Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus

Bullshark


Bullshark: DAG BFT Protocols Made Practical

Dumbo-NG


Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency

...

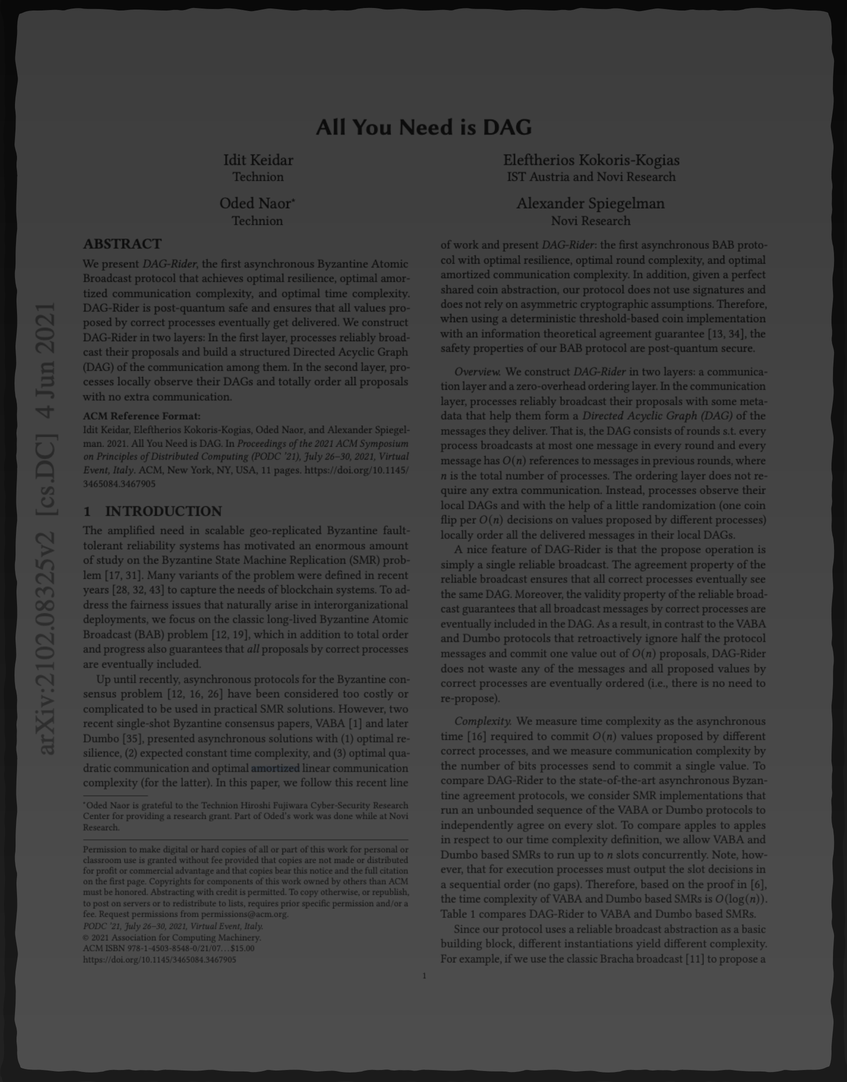
Data Dissemination

- Hard to make efficient
- 99% of the code

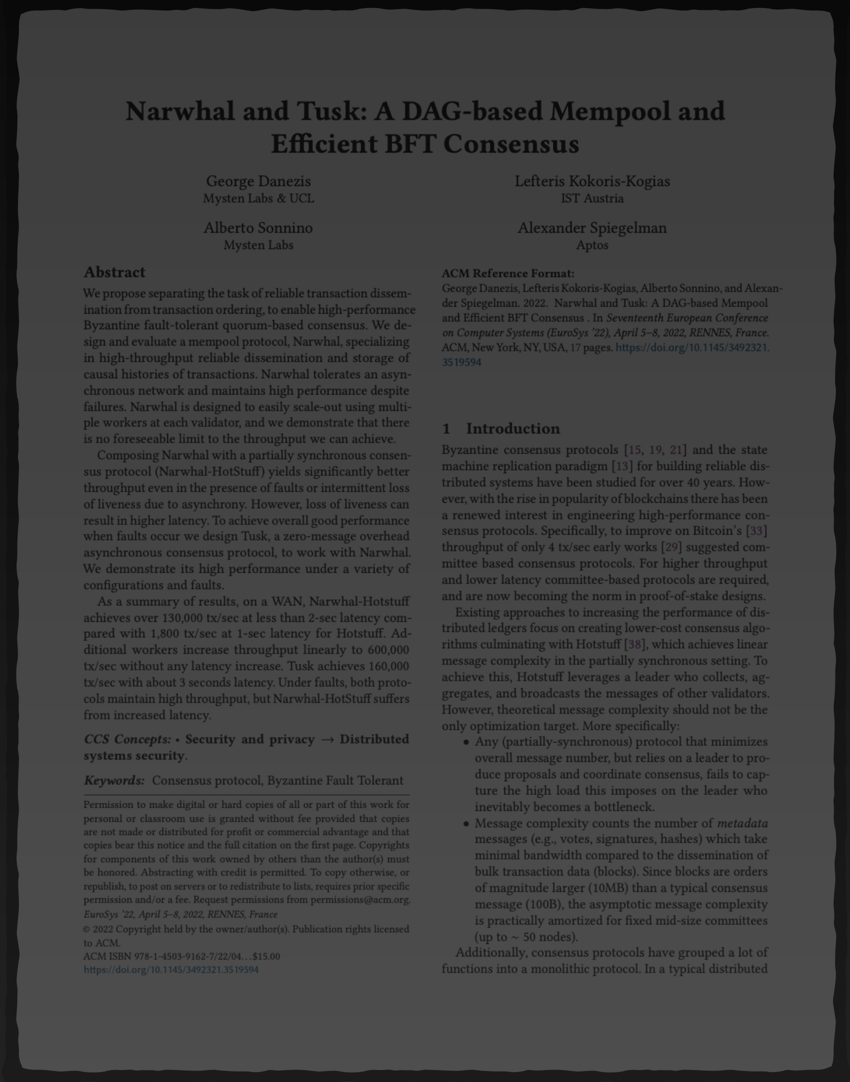
Consensus

- Error prone
- Isolated, easy to maintain

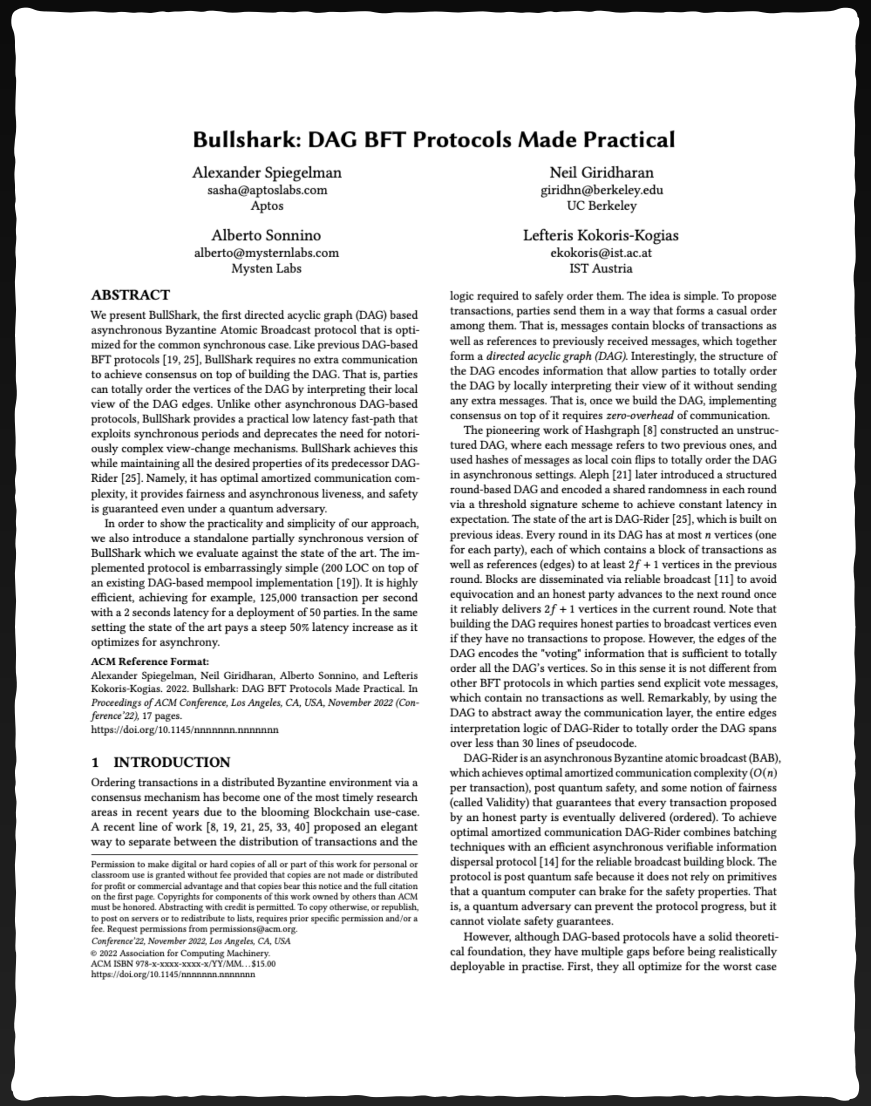
Bullshark


All You Need is DAG

Idit Keidar, Technion
Eleftherios Kokoris-Kogias, IST Austria and Novi Research
Oded Naor, Technion
Alexander Spiegelman, Novi Research


Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus

George Danezis, Mysten Labs & UCL
Lefteris Kokoris-Kogias, IST Austria
Alberto Sonnino, Mysten Labs
Alexander Spiegelman, Aptos


Bullshark: DAG BFT Protocols Made Practical

Alexander Spiegelman
sasha@aptoslabs.com
Aptos

Neil Giridharan
giridhn@berkeley.edu
UC Berkeley

Alberto Sonnino
alberto@mysternlabs.com
Mysten Labs

Lefteris Kokoris-Kogias
ekokoris@ist.ac.at
IST Austria

ABSTRACT

We present BullShark, the first directed acyclic graph (DAG) based asynchronous Byzantine Atomic Broadcast protocol that is optimized for the common synchronous case. Like previous DAG-based BFT protocols [19, 25], BullShark requires no extra communication to achieve consensus on top of building the DAG. That is, parties can totally order the vertices of the DAG by interpreting their local view of the DAG edges. Unlike other asynchronous DAG-based protocols, BullShark provides a practical low latency fast-path that exploits synchronous periods and deprecates the need for notoriously complex view-change mechanisms. BullShark achieves this while maintaining all the desired properties of its predecessor DAG-Rider [25]. Namely, it has optimal amortized communication complexity, it provides fairness and asynchronous liveness, and safety is guaranteed even under a quantum adversary.

In order to show the practicality and simplicity of our approach, we also introduce a standalone partially synchronous version of BullShark which we evaluate against the state of the art. The implemented protocol is embarrassingly simple (200 LOC on top of an existing DAG-based mempool implementation [19]). It is highly efficient, achieving for example, 125,000 transaction per second with a 2 seconds latency for a deployment of 50 parties. In the same setting the state of the art pays a steep 50% latency increase as it optimizes for asynchrony.

ACM Reference Format:
Alexander Spiegelman, Neil Giridharan, Alberto Sonnino, and Lefteris Kokoris-Kogias. 2022. Bullshark: DAG BFT Protocols Made Practical. In *Proceedings of ACM Conference, Los Angeles, CA, USA, November 2022 (Conference'22)*, 17 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

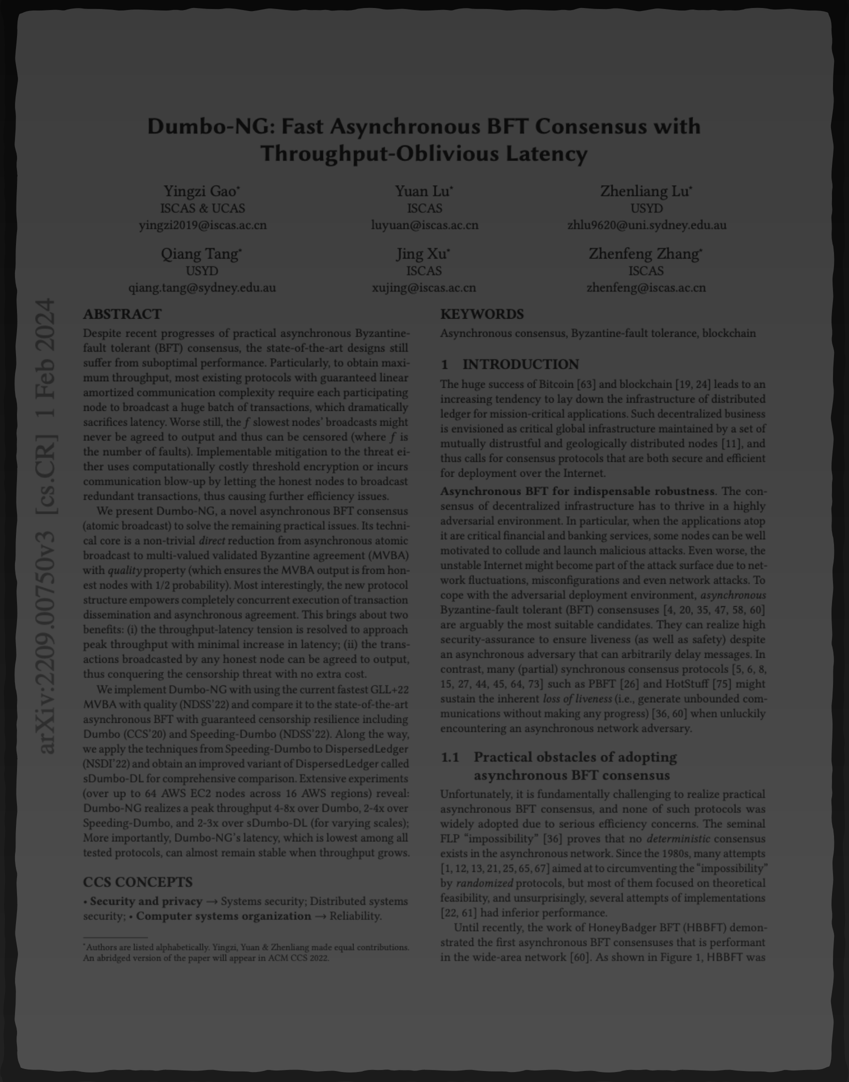
1 INTRODUCTION

Ordering transactions in a distributed Byzantine environment via a consensus mechanism has become one of the most timely research areas in recent years due to the blooming Blockchain use-case. A recent line of work [8, 19, 21, 25, 33, 40] proposed an elegant way to separate between the distribution of transactions and the logic required to safely order them. The idea is simple. To propose transactions, parties send them in a way that forms a casual order among them. That is, messages contain blocks of transactions as well as references to previously received messages, which together form a *directed acyclic graph (DAG)*. Interestingly, the structure of the DAG encodes information that allow parties to totally order the DAG by locally interpreting their view of it without sending any extra messages. That is, once we build the DAG, implementing consensus on top of it requires *zero-overhead* of communication.

The pioneering work of Hashgraph [8] constructed an unstructured DAG, where each message refers to two previous ones, and used hashes of messages as local coin flips to totally order the DAG in asynchronous settings. Aleph [21] later introduced a structured round-based DAG and encoded a shared randomness in each round via a threshold signature scheme to achieve constant latency in expectation. The state of the art is DAG-Rider [25], which is built on previous ideas. Every round in its DAG has at most n vertices (one for each party), each of which contains a block of transactions as well as references (edges) to at least $2f + 1$ vertices in the previous round. Blocks are disseminated via reliable broadcast [11] to avoid equivocation and an honest party advances to the next round once it reliably delivers $2f + 1$ vertices in the current round. Note that building the DAG requires honest parties to broadcast vertices even if they have no transactions to propose. However, the edges of the DAG encodes the "voting" information that is sufficient to totally order all the DAG's vertices. So in this sense it is not different from other BFT protocols in which parties send explicit vote messages, which contain no transactions as well. Remarkably, by using the DAG to abstract away the communication layer, the entire edges interpretation logic of DAG-Rider to totally order the DAG spans over less than 30 lines of pseudocode.

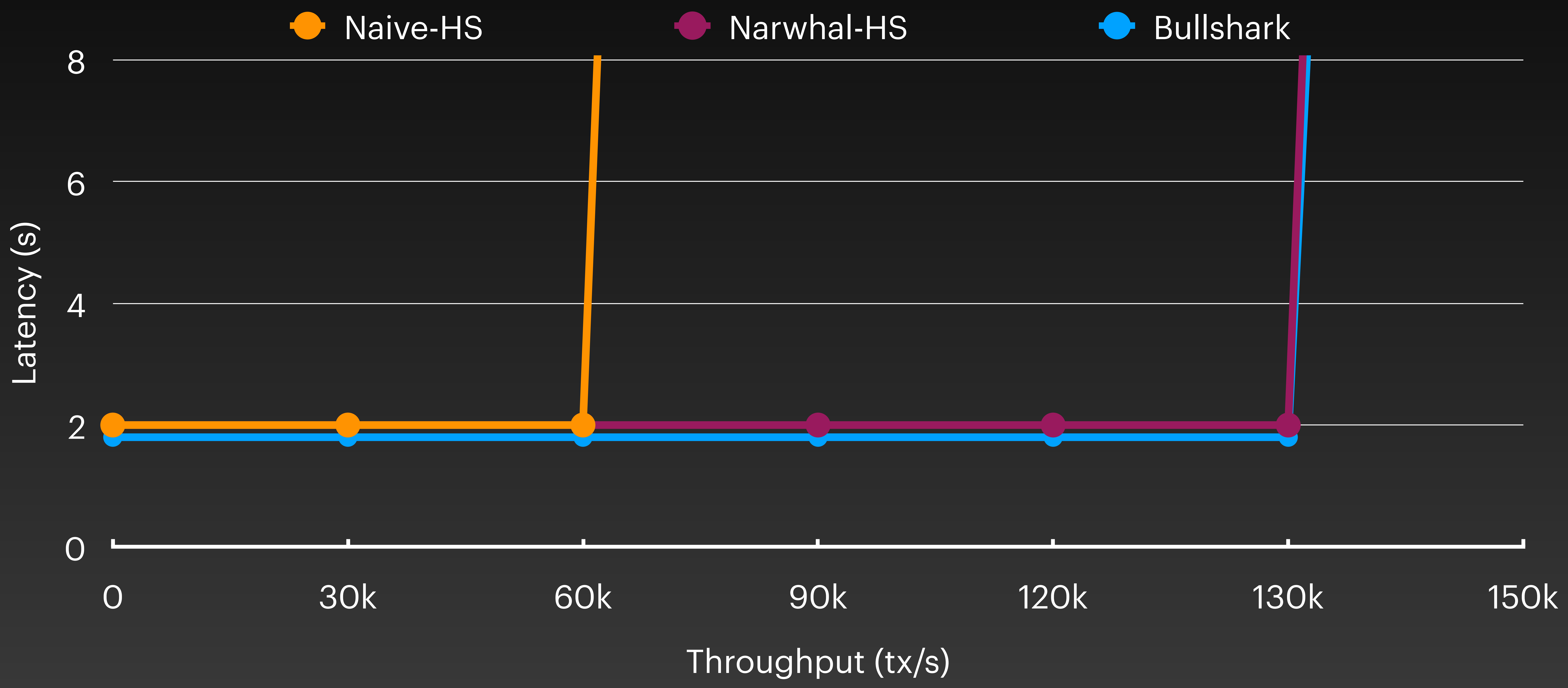
DAG-Rider is an asynchronous Byzantine atomic broadcast (BAB), which achieves optimal amortized communication complexity ($O(n)$ per transaction), post quantum safety, and some notion of fairness (called Validity) that guarantees that every transaction proposed by an honest party is eventually delivered (ordered). To achieve optimal amortized communication DAG-Rider combines batching techniques with an efficient asynchronous verifiable information dispersal protocol [14] for the reliable broadcast building block. The protocol is post quantum safe because it does not rely on primitives that a quantum computer can brake for the safety properties. That is, a quantum adversary can prevent the protocol progress, but it cannot violate safety guarantees.

However, although DAG-based protocols have a solid theoretical foundation, they have multiple gaps before being realistically deployable in practise. First, they all optimize for the worst case


Dumbo-NG: Fast Asynchronous BFT Consensus with Throughput-Oblivious Latency

Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, Zhenfeng Zhang

Performance



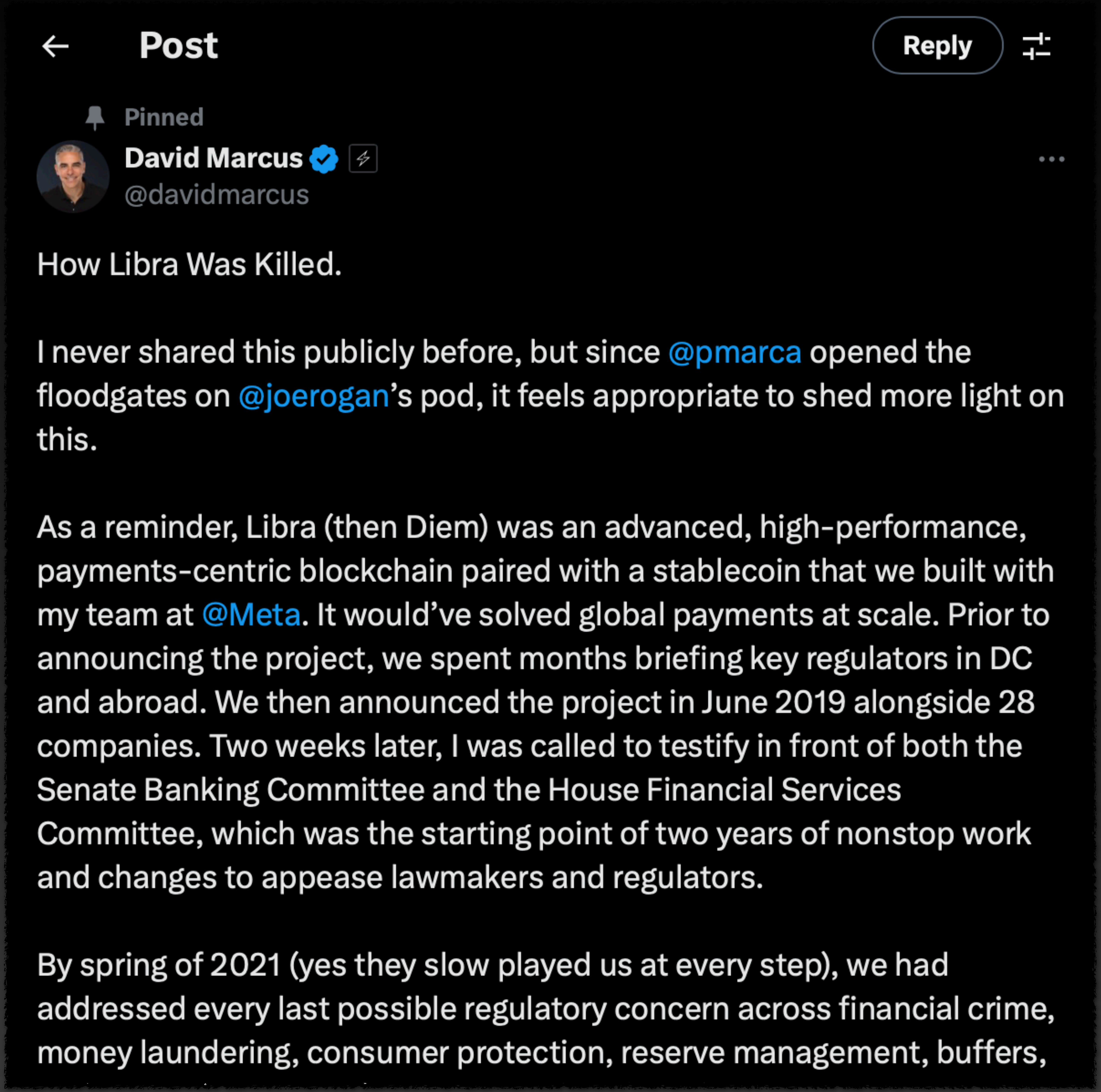
Research Questions

1. Network model?
2. BFT testing?
3. Consensus-exec interface?

Lessons Learned

1. Modularisation is a design strategy
2. Tasks-threads allocation
3. Benchmark early
4. Codesign with mem. and storage
5. Core is hard, consensus is easy

By that time...



Post

Reply

Pinned

David Marcus
@davidmarcus

How Libra Was Killed.

I never shared this publicly before, but since @pmarca opened the floodgates on @joerogan's pod, it feels appropriate to shed more light on this.

As a reminder, Libra (then Diem) was an advanced, high-performance, payments-centric blockchain paired with a stablecoin that we built with my team at @Meta. It would've solved global payments at scale. Prior to announcing the project, we spent months briefing key regulators in DC and abroad. We then announced the project in June 2019 alongside 28 companies. Two weeks later, I was called to testify in front of both the Senate Banking Committee and the House Financial Services Committee, which was the starting point of two years of nonstop work and changes to appease lawmakers and regulators.

By spring of 2021 (yes they slow played us at every step), we had addressed every last possible regulatory concern across financial crime, money laundering, consumer protection, reserve management, buffers,

By that time...

Sui

Aptos

Linera

...

Fundraising with papers seems to work

Sui, 2022

Over a year for mainnet

- Lack of checkpoints
- Lack of epoch-change
- Lack of crash-recovery

Research Questions

1. Network model?
2. BFT testing?
3. Consensus-exec interface?
4. Storage architecture?

Lessons Learned

1. Modularisation is a design strategy
2. Tasks-threads allocation
3. Benchmark early
4. Codesign with mem. and storage
5. Core is hard, consensus is easy
6. Epoch change is not an add-on

Sui, 2023

- Latency was too high
- Crash faults were the predominant faults
- Building Bullshark was still too complex

Shoal


Shoal: Improving DAG-BFT Latency And Robustness

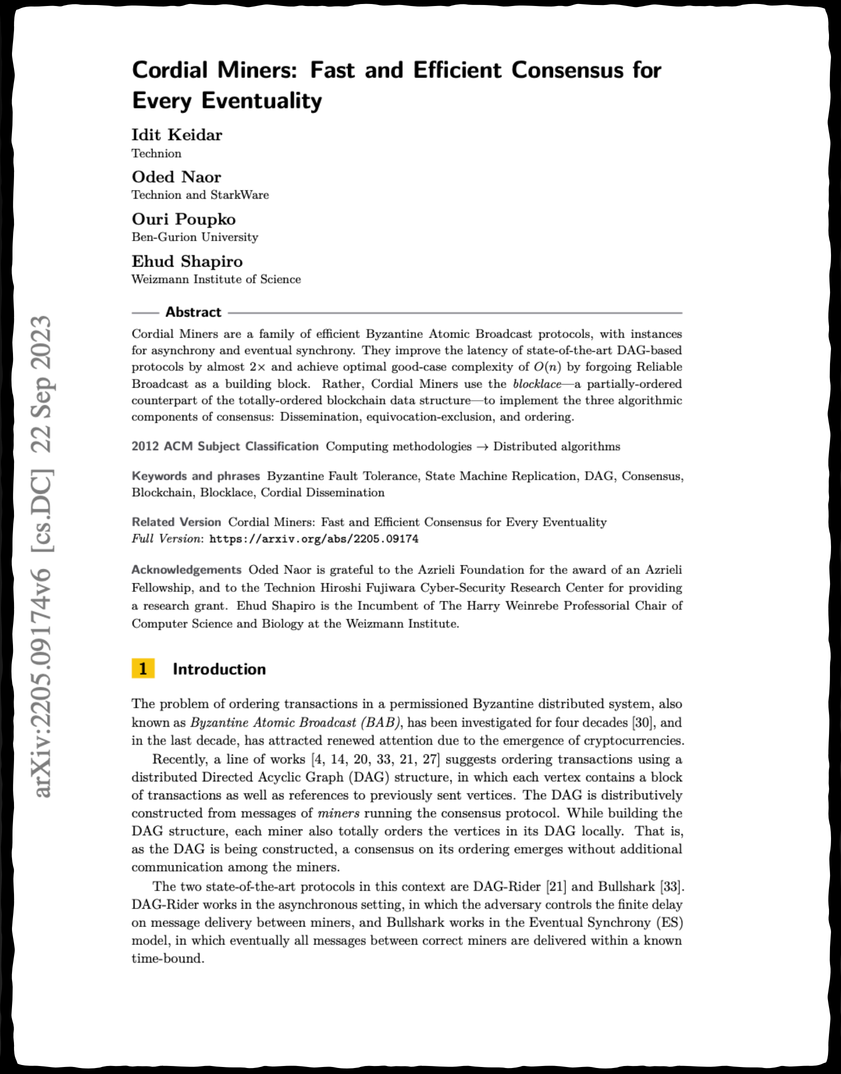
Alexander Spiegelman, Aptos; Balaji Arun, Aptos; Rati Gelashvili, Aptos; Zekun Li, Aptos

Sailfish


Sailfish: Towards Improving the Latency of DAG-based BFT

Nibesh Shrestha, Supra Research; Rohan Shrothrium, Kuru Labs; Aniket Kate, Purdue University / Supra Research; Kartik Nayak, Duke University

CM


Cordial Miners: Fast and Efficient Consensus for Every Eventuality

Idit Keidar, Technion; Oded Naor, Technion and StarkWare; Ouri Poupko, Ben-Gurion University; Ehud Shapiro, Weizmann Institute of Science

Mysticeti


MYSTICETI: Reaching the Latency Limits with Uncertified DAGs

...

Techniques

- Many leaders per round
- Leaders every round
- Uncertified DAG

Discussion

Certified DAG ⟷ **Uncertified DAG**

Shoal/shoal++

- Low latency
- Easier synchroniser
- Leverage existing code

Sailfish/BBCA

- Lower latency
- Easy synchroniser
- Flexible

CM/Mysticeti

- Lowest latency
- Graceful crash faults
- Simpler, less CPU

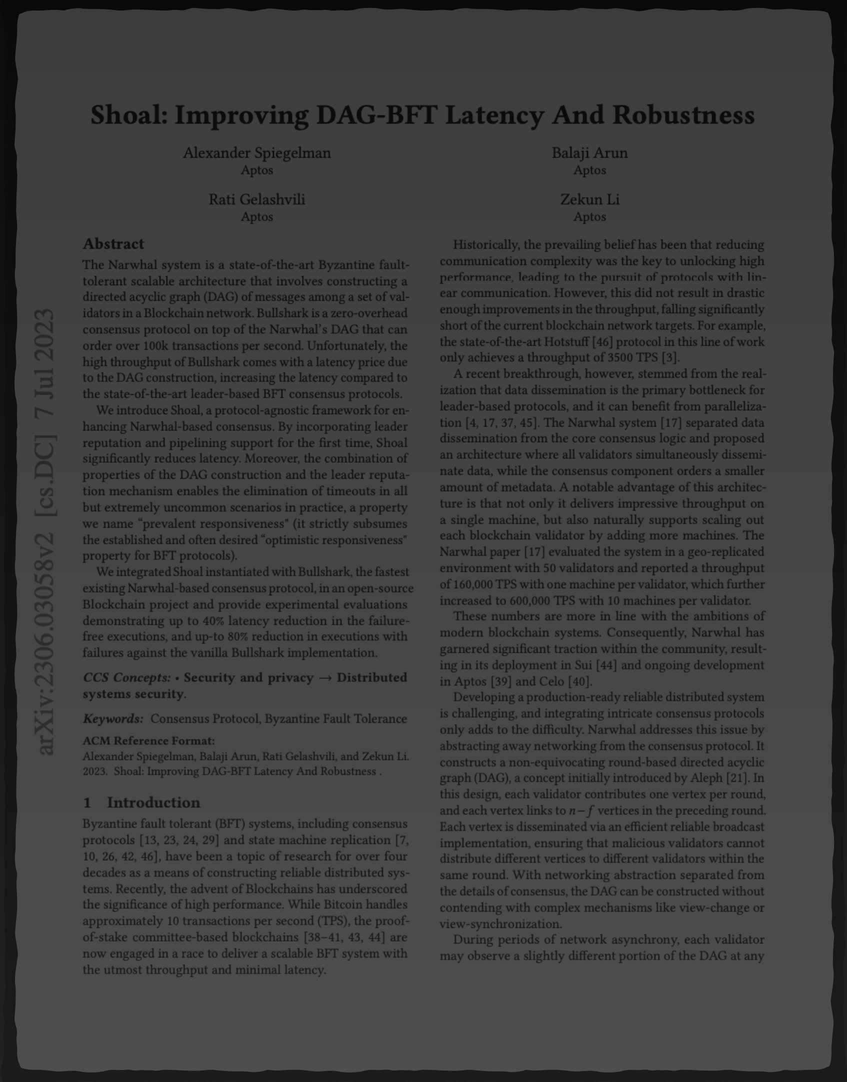
Research Questions

1. Network model?
2. BFT testing?
3. Consensus-exec interface?
4. Storage architecture?
5. Block synchroniser?

Lessons Learned

1. Modularisation is a design strategy
2. Tasks-threads allocation
3. Benchmark early
4. Codesign with mem. and storage
5. Core is hard, consensus is easy
6. Epoch change is not an add-on

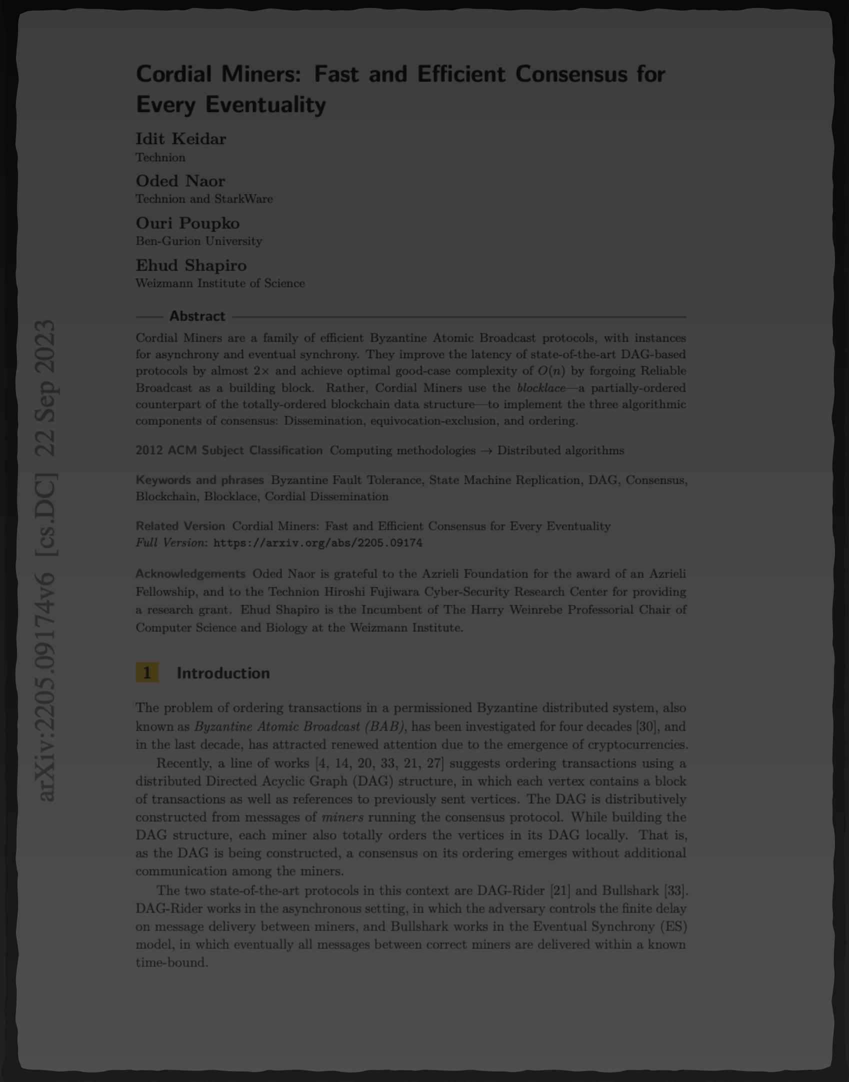
Mysticeti


Shoal: Improving DAG-BFT Latency And Robustness

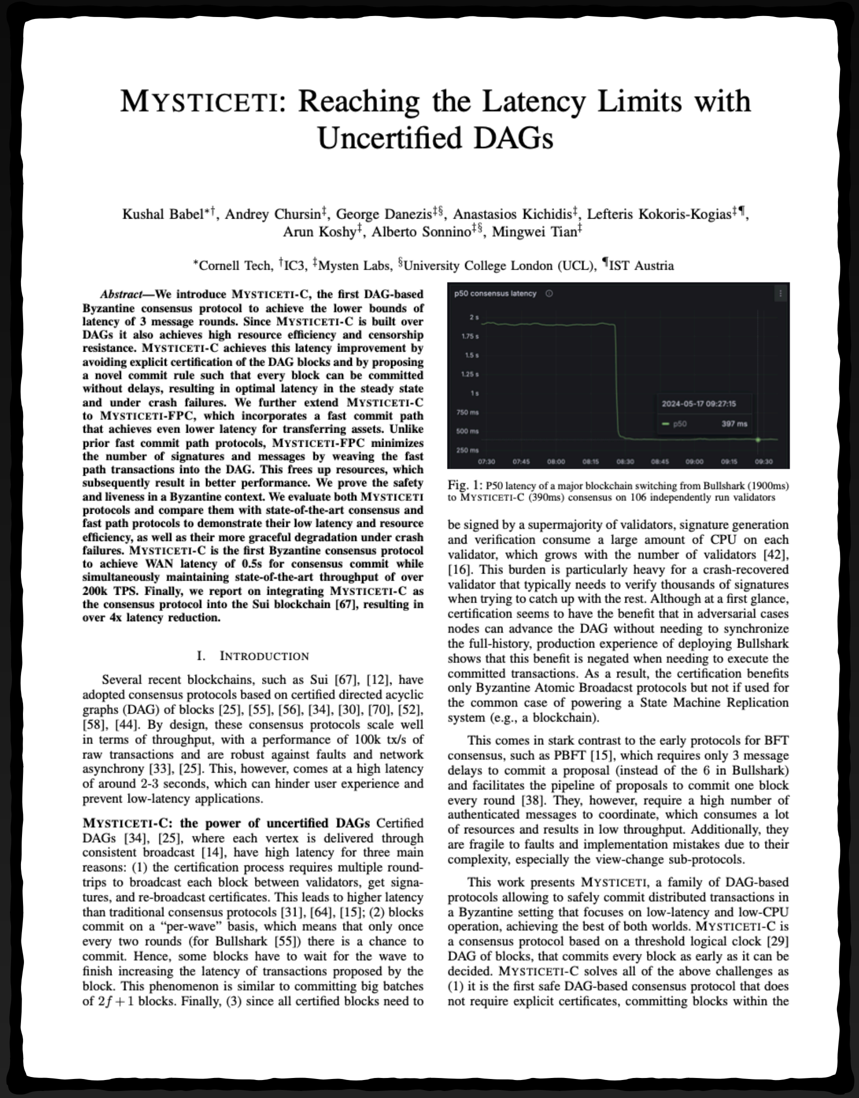
Alexander Spiegelman, Balaji Arun, Rati Gelashvili, Zekun Li


Sailfish: Towards Improving the Latency of DAG-based BFT

Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, Kartik Nayak

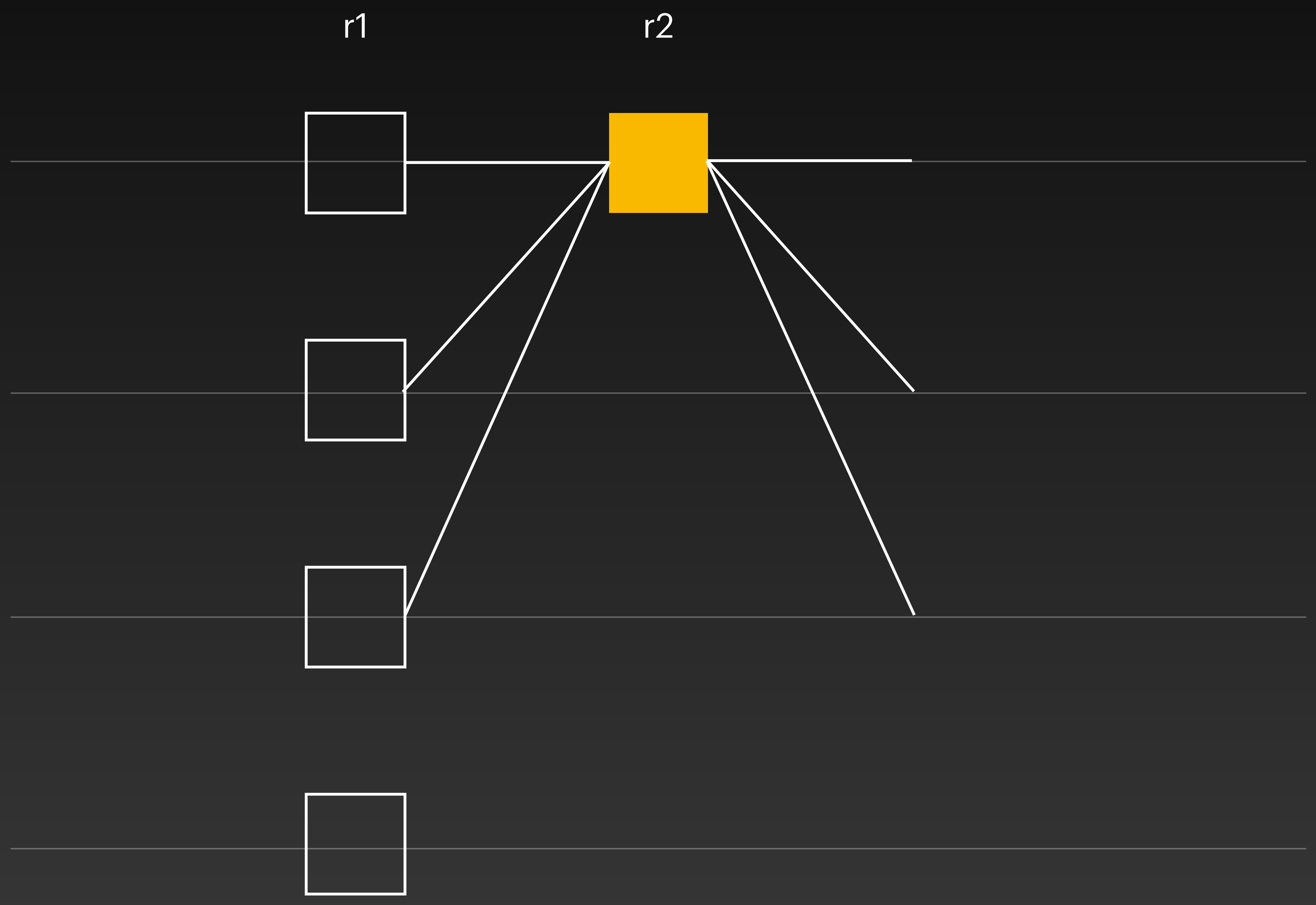

Cordial Miners: Fast and Efficient Consensus for Every Eventuality

Idit Keidar, Oded Naor, Ouri Poupko, Ehud Shapiro


MYSTICETI: Reaching the Latency Limits with Uncertified DAGs

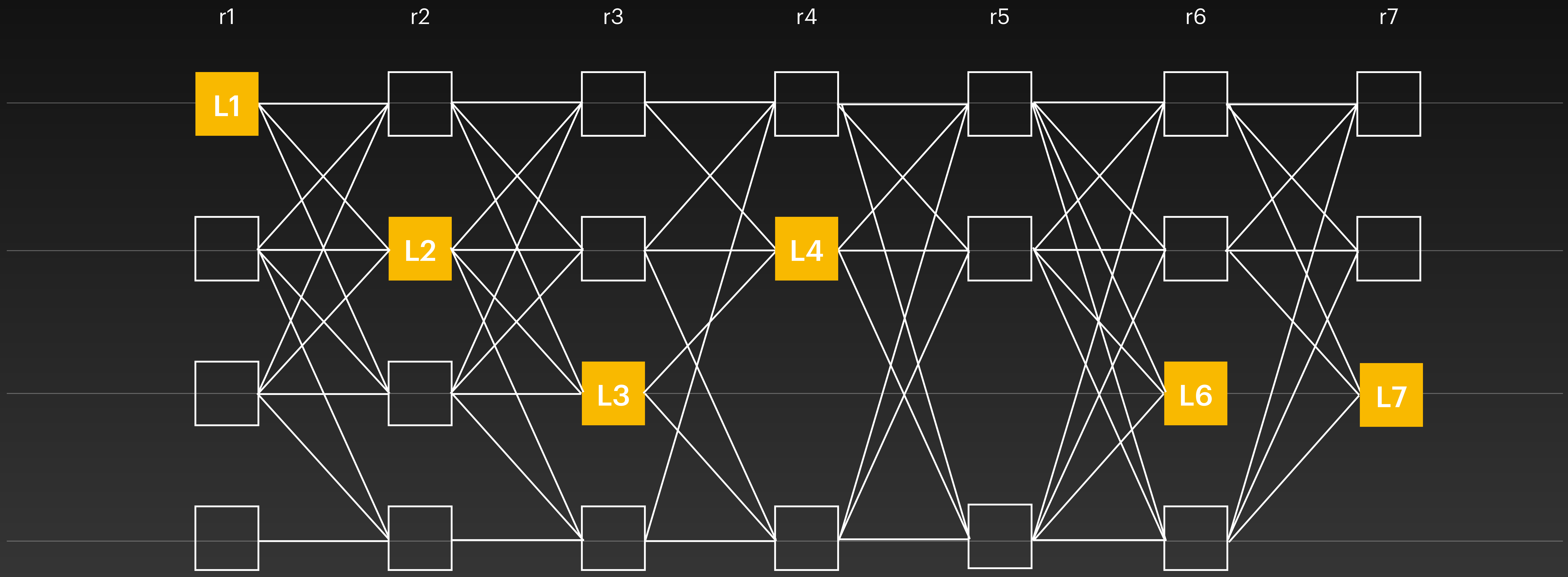
Kushal Babel, Andrey Chursin, George Danezis, Anastasios Kichidis, Lefteris Kokoris-Kogias, Arun Koshy, Alberto Sonnino, Mingwei Tian

Uncertified DAG

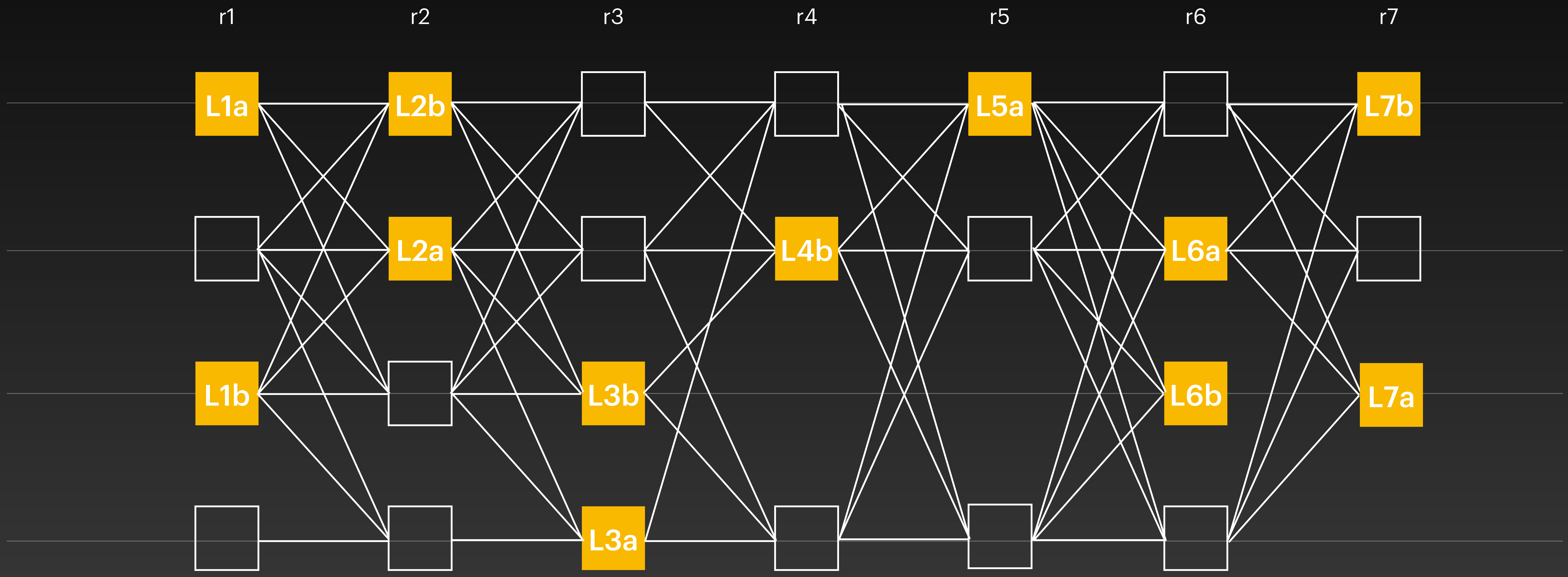


- Round number
- Author
- Payload (transactions)
- Signature

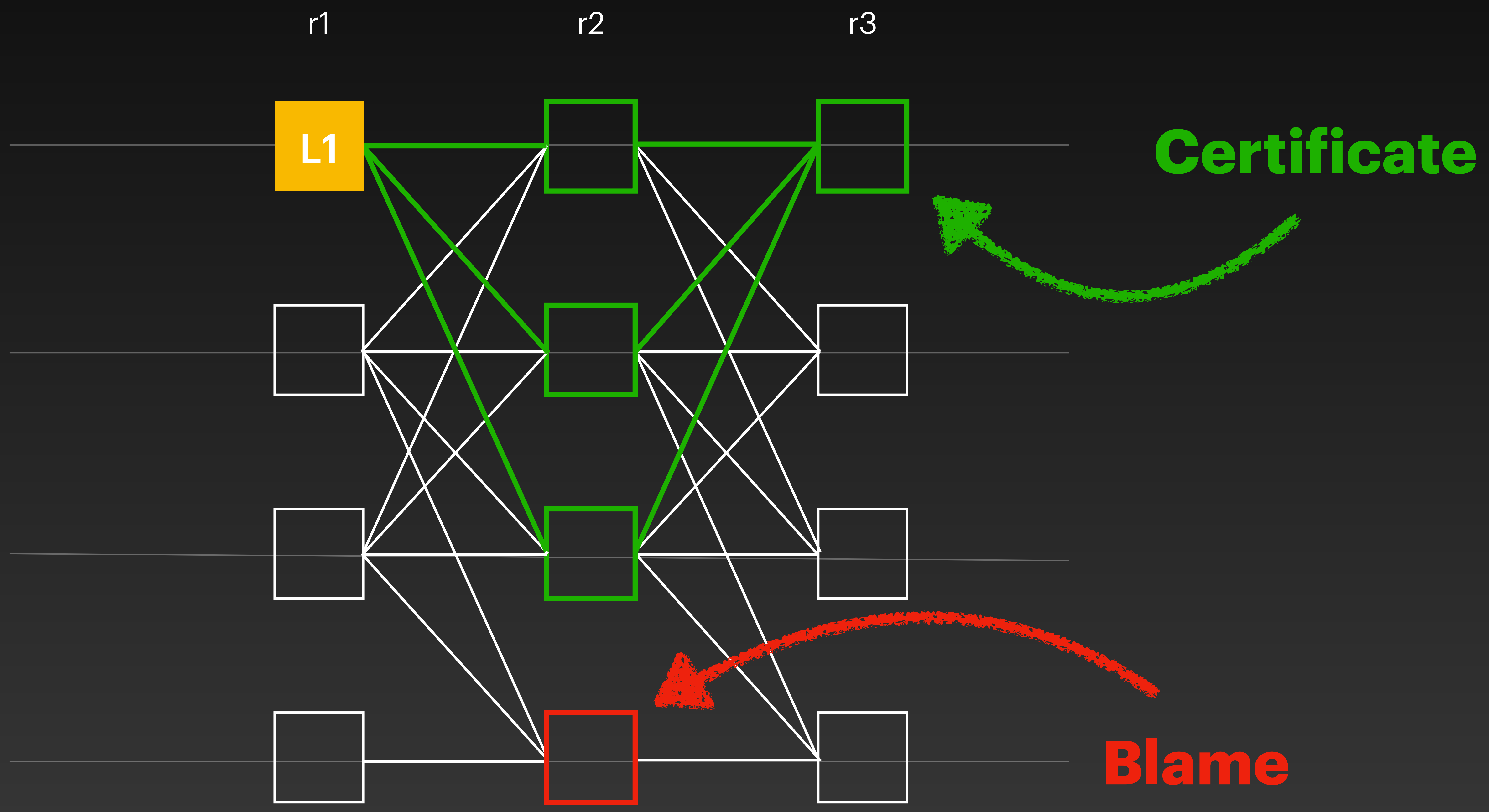
Uncertified DAG



Uncertified DAG



Interpreting DAG Patterns



Direct Decision Rule

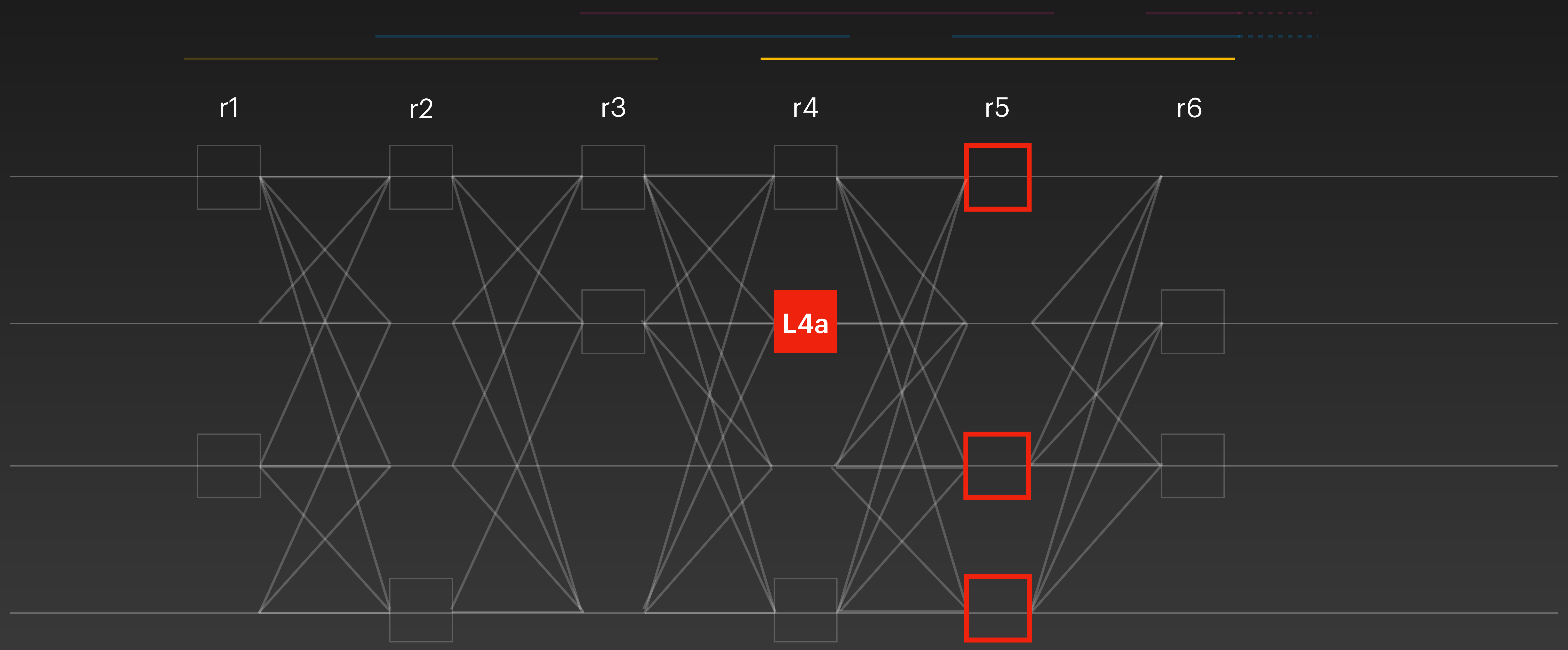
On each leader starting from highest round:

- **Skip** if 2f+1 blames
- **Commit** if 2f+1 certificates
- **Undecided** otherwise

Direct Decision Rule

On each leader starting from highest round:

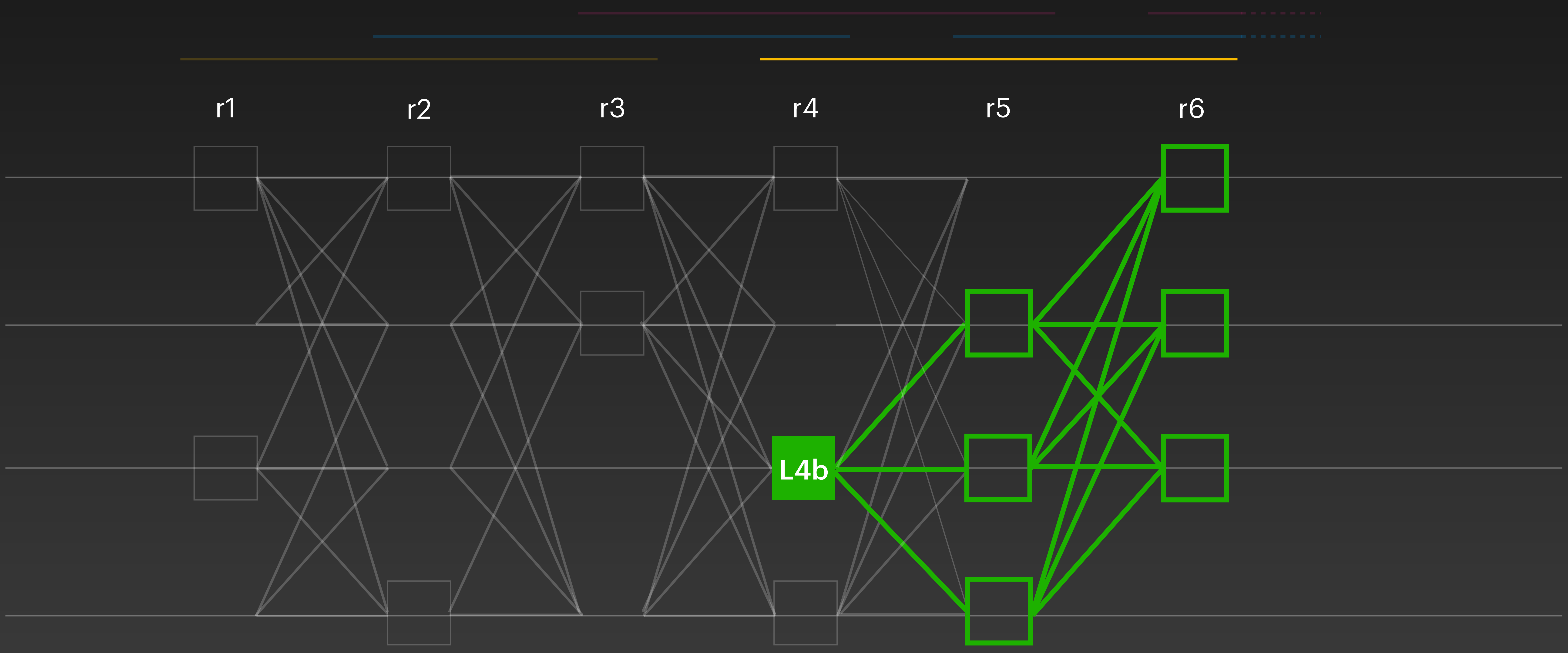
- **Skip** if 2f+1 blames
- **Commit** if 2f+1 certificates
- **Undecided** otherwise



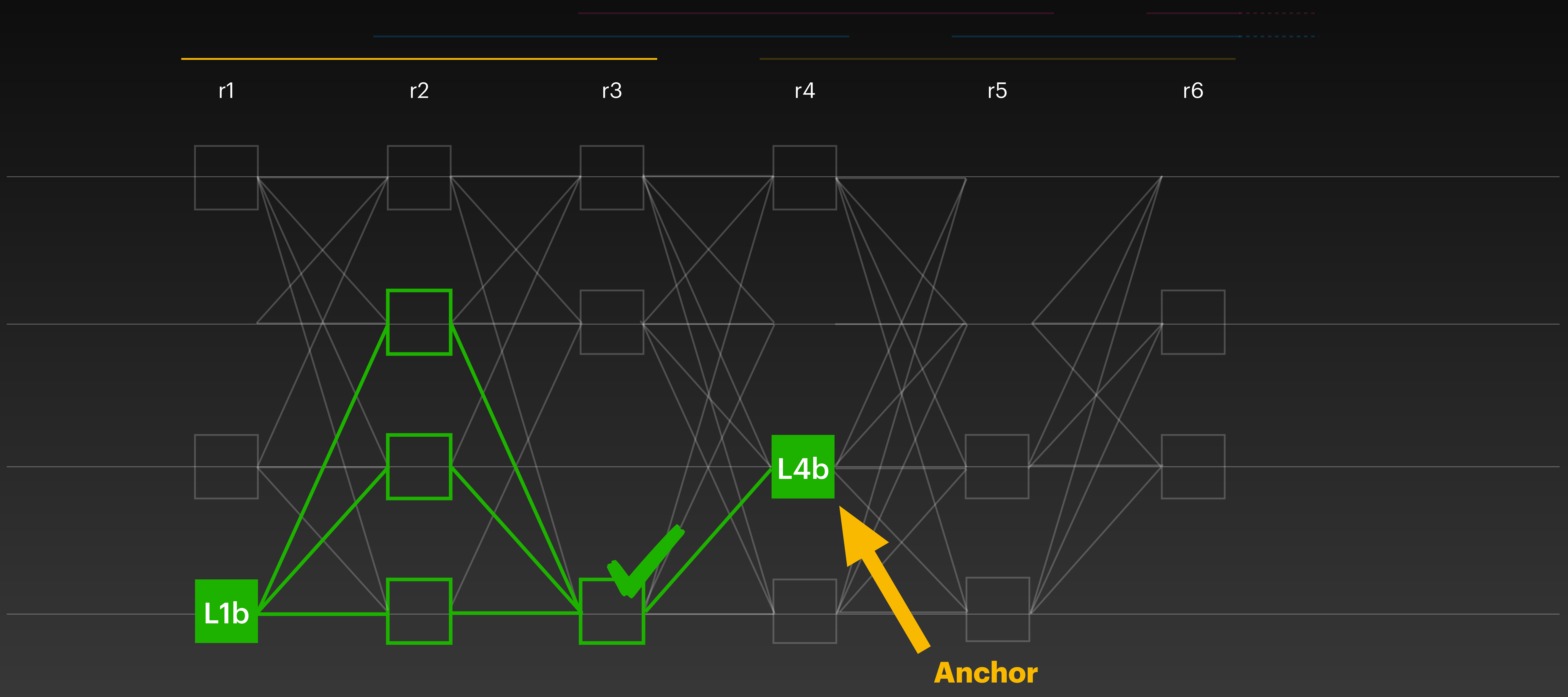
Direct Decision Rule

On each leader starting from highest round:

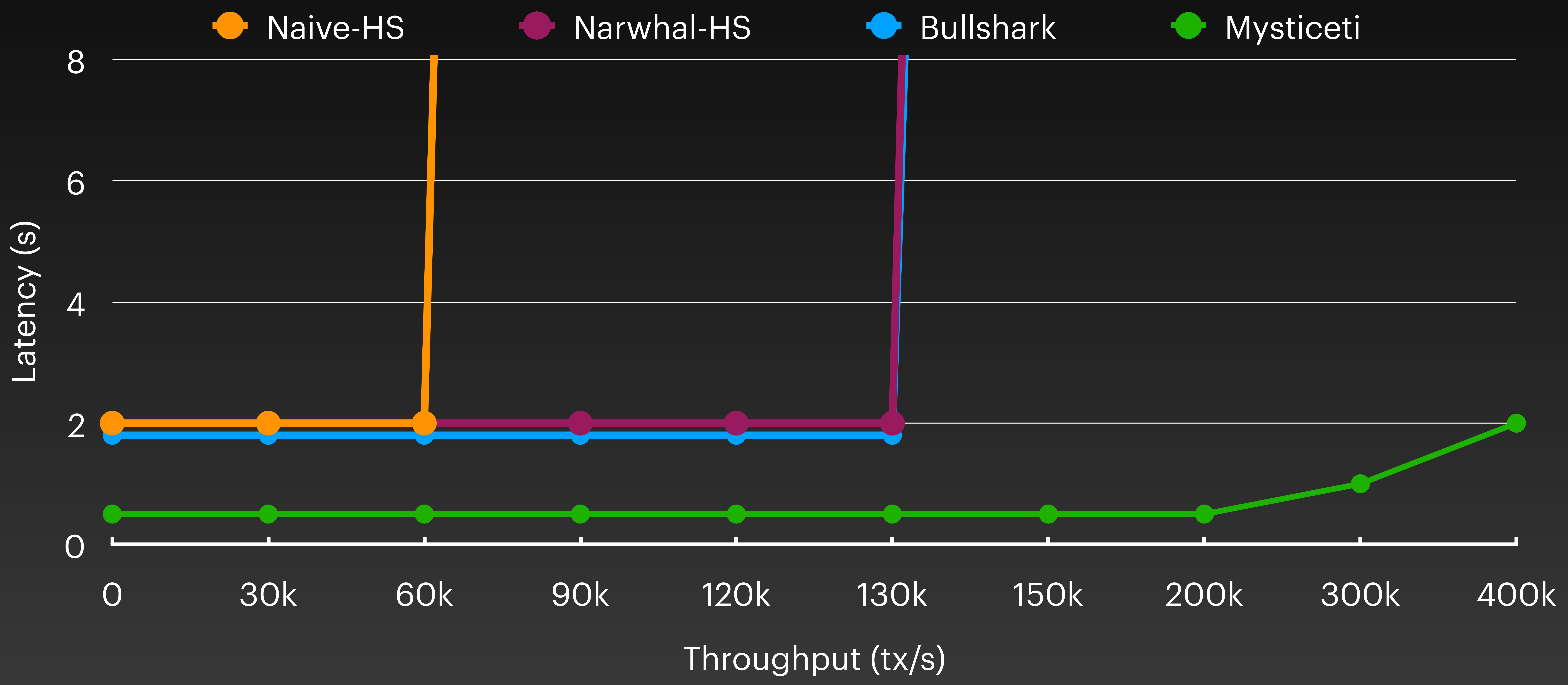
- **Skip** if 2f+1 blames
- **Commit** if 2f+1 certificates
- **Undecided** otherwise



Indirect Decision Rule



Performance



Engineering Benchmarks

Protocol	Committee	Load/TPS	P50	P95
Bullshark	137	5k	2.89 s	4.60 s
Mysticeti	137	5k	397 ms	690 ms

We ran it for 24h and it looks good 👍

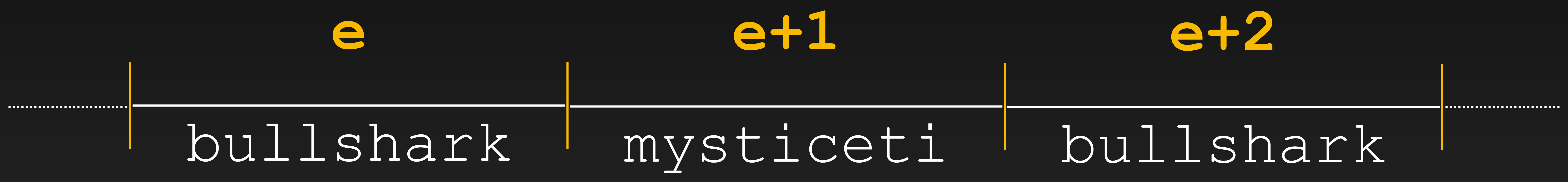
Research Questions

1. Network model?
2. BFT testing?
3. Consensus-exec interface?
4. Storage architecture?
5. Block synchroniser?
6. Realistic benchmarks?
7. Efficient reads?

Lessons Learned

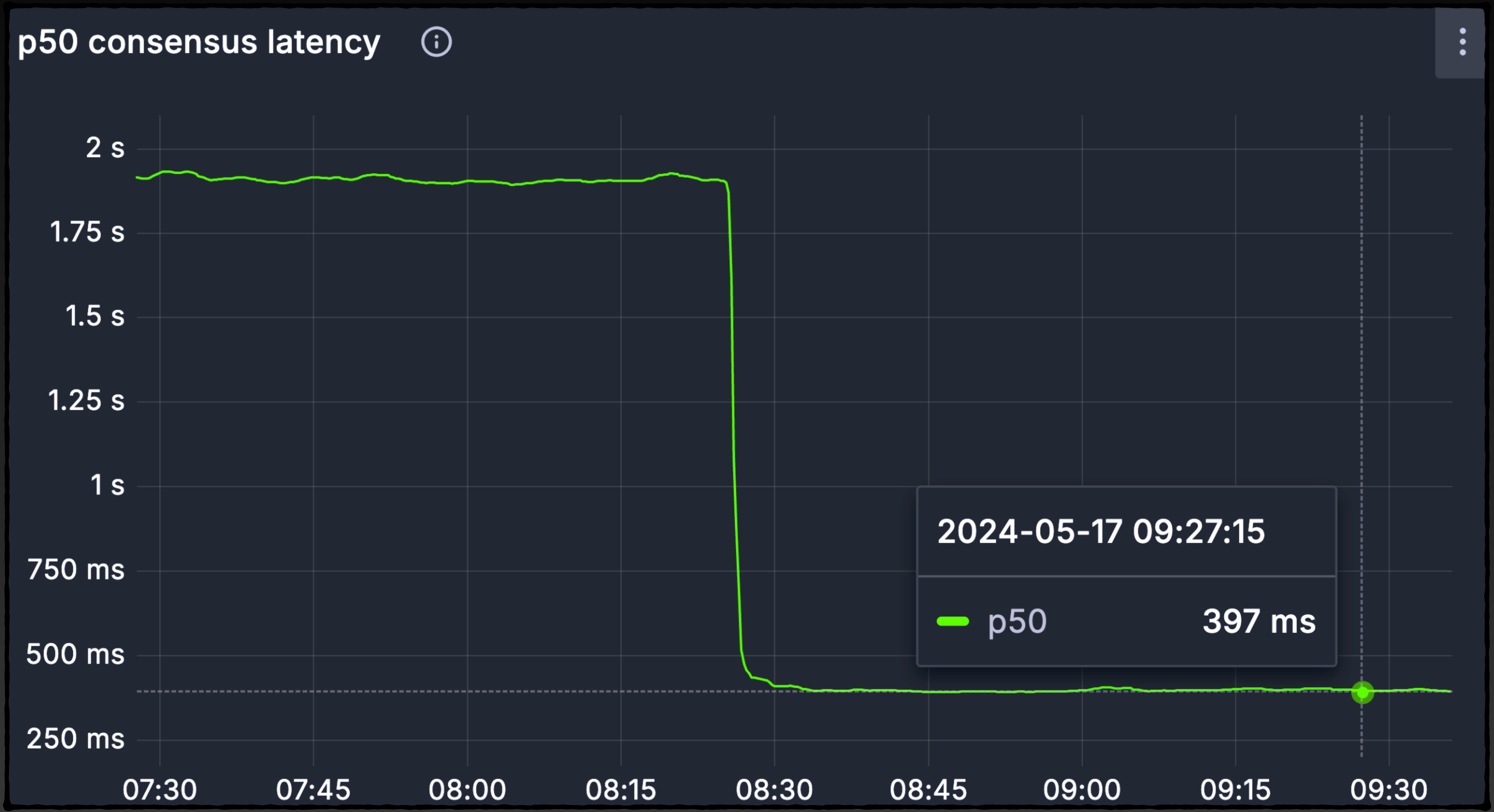
1. Modularisation is a design strategy
2. Tasks-threads allocation
3. Benchmark early
4. Codesign with mem. and storage
5. Core is hard, consensus is easy
6. Epoch change is not an add-on

Testing Strategy



- Compare performance & robustness
- Test mainnet change `bullshark -> mysticeti`
- Prepare for the worst `mysticeti -> bullshark`

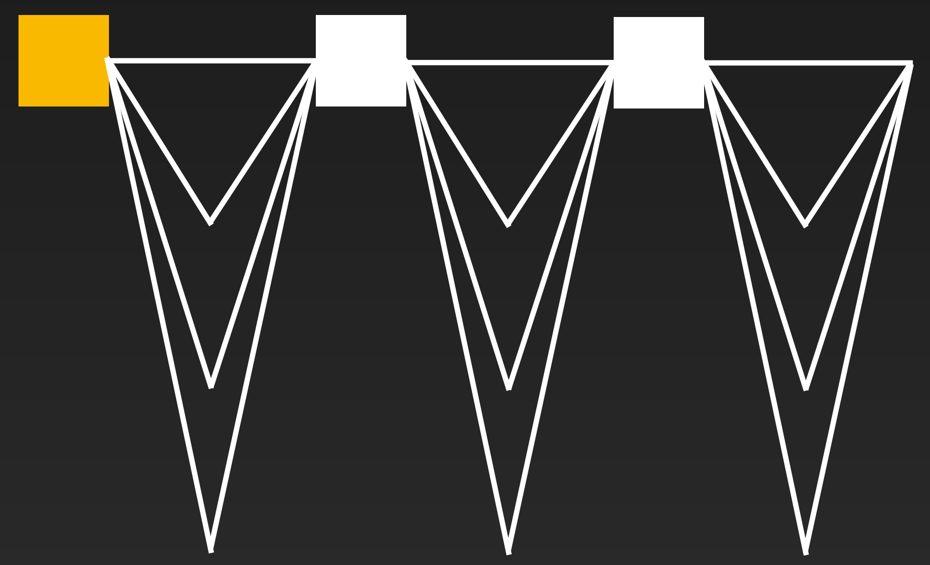
The Sui Mainnet



Conclusion

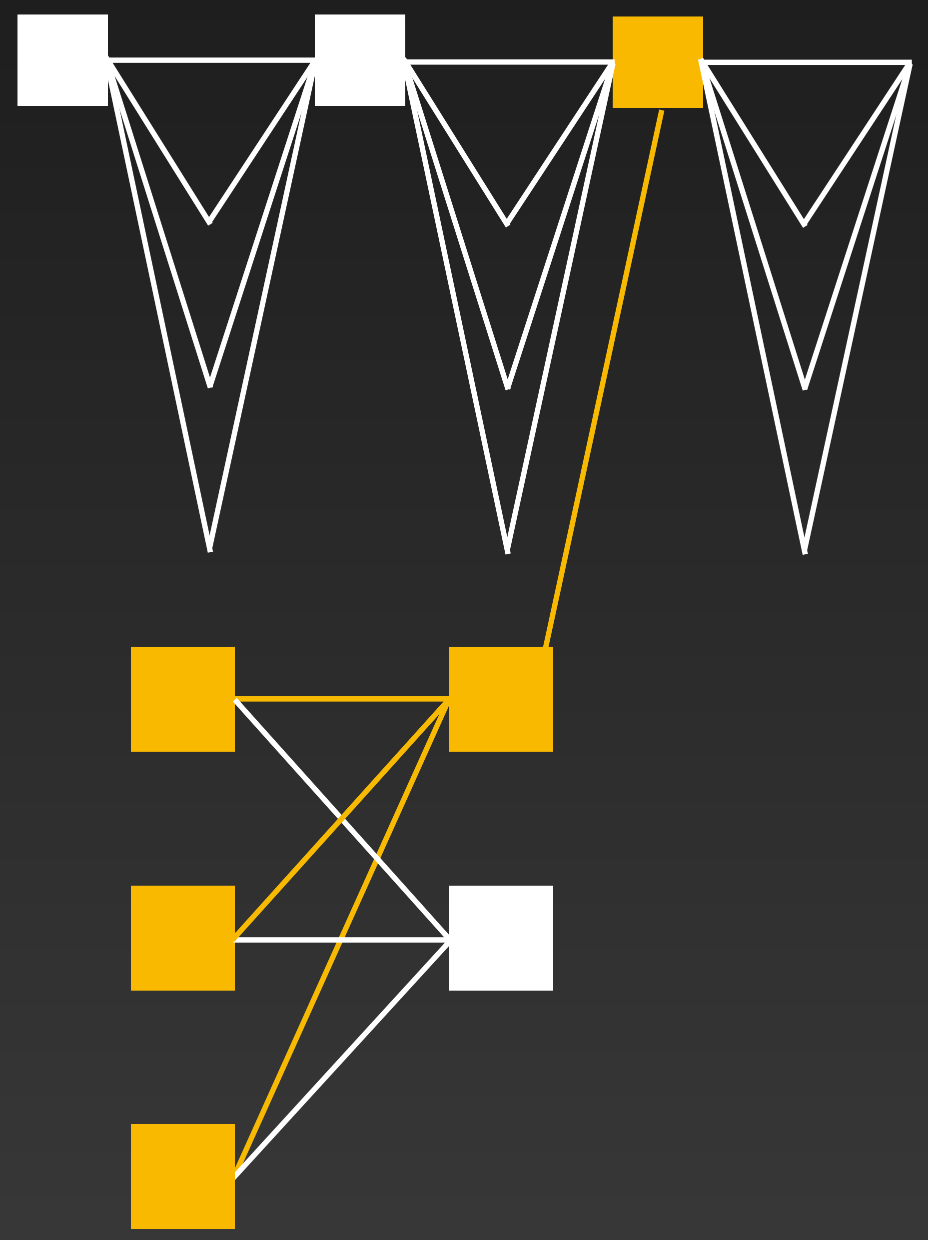
2019

naive consensus



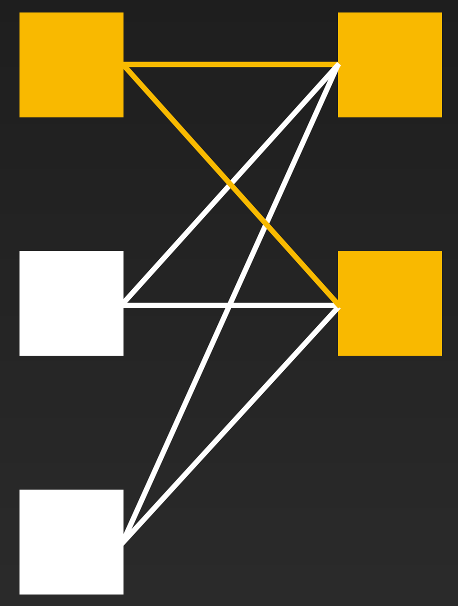
2020-2021

mempool ❤️consensus



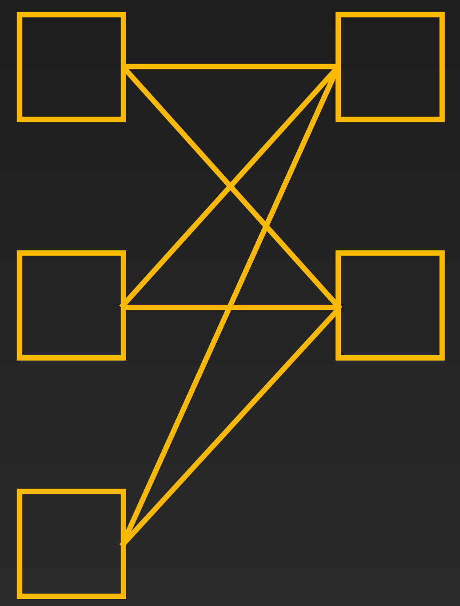
2022-2023

Fold into DAG



2024

Remove overhead



alberto@mystenlabs.com

EXTRA:
Research in Industry

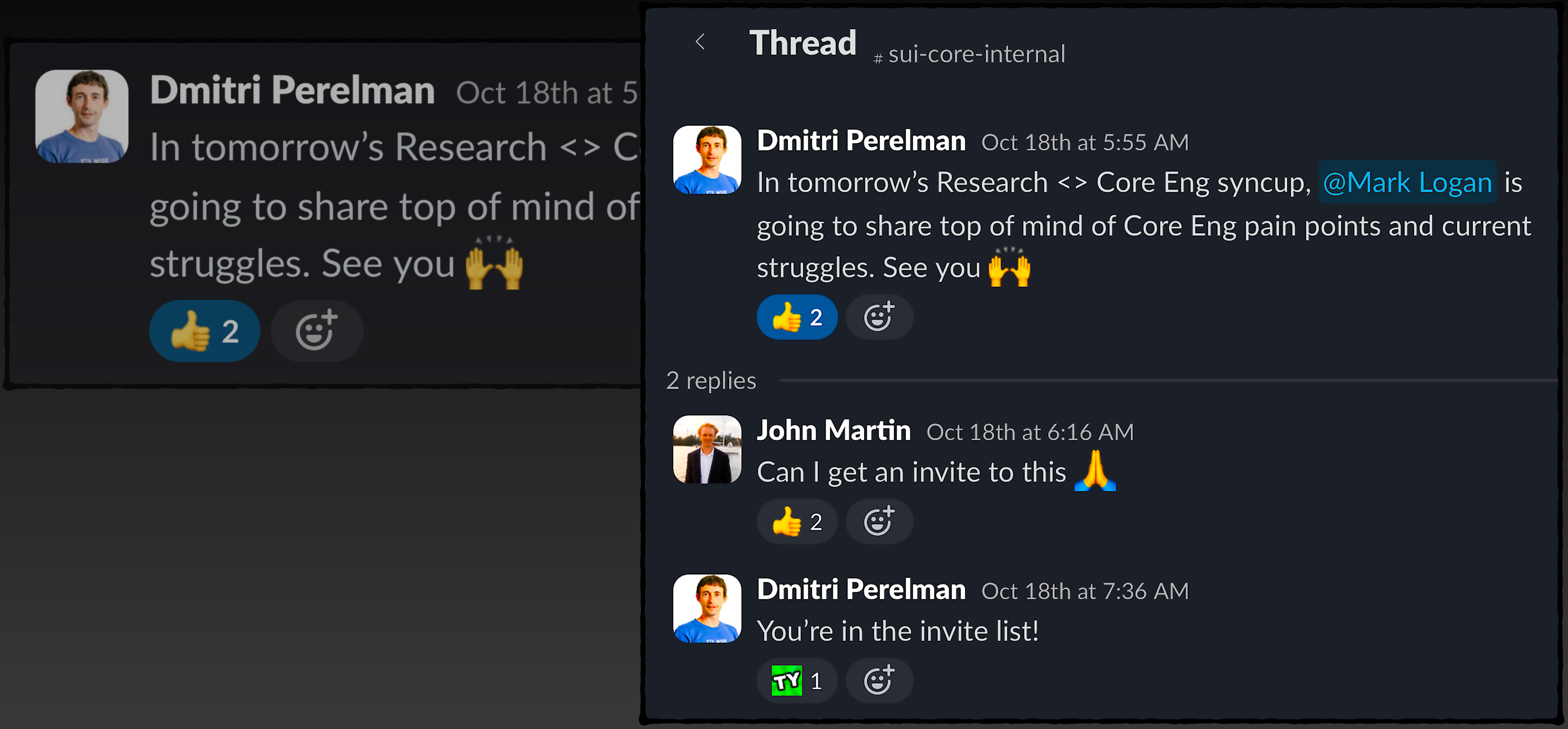
Projects Roadmap

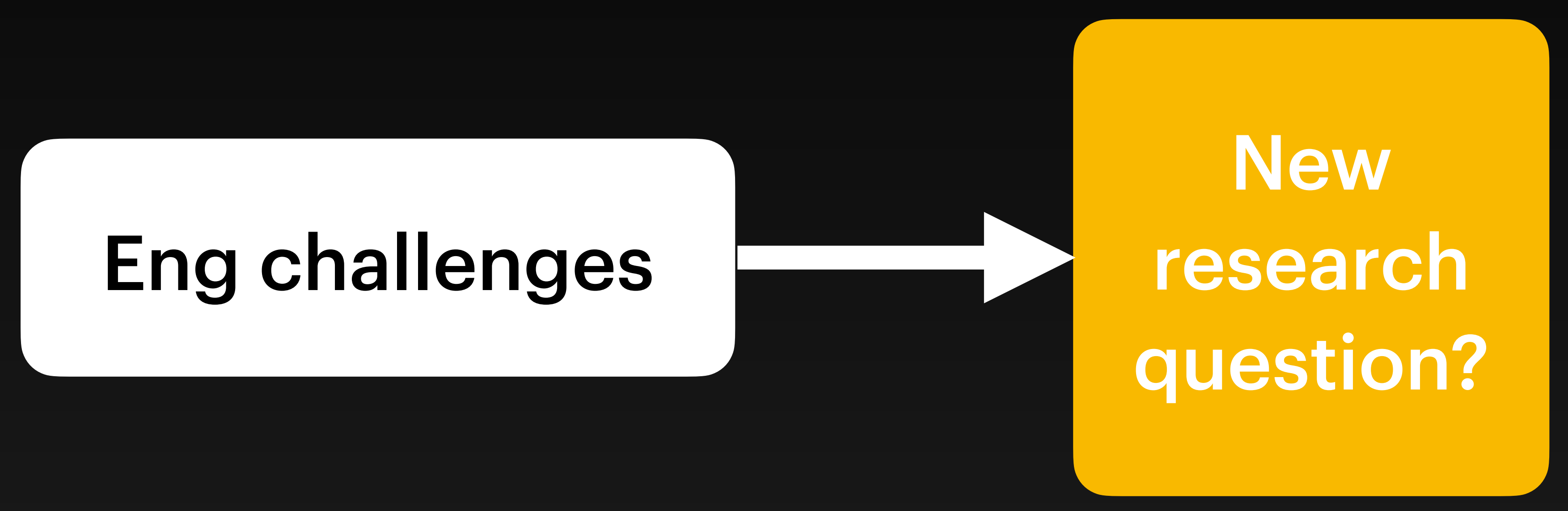
Dmitri Perelman Oct 18th at 5:55 AM

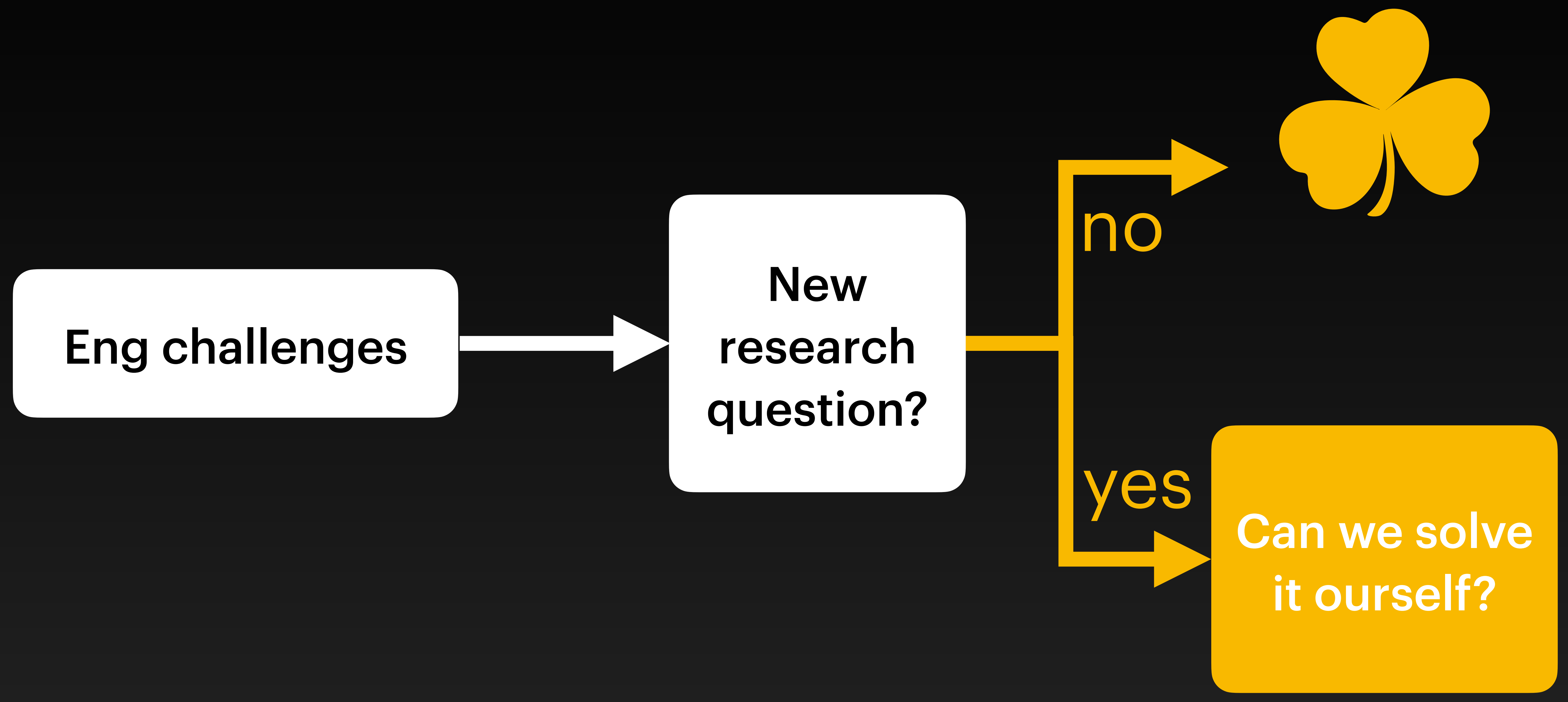
In tomorrow's Research <> Core Eng syncup, @Mark Logan is going to share top of mind of Core Eng pain points and current struggles. See you 🙌

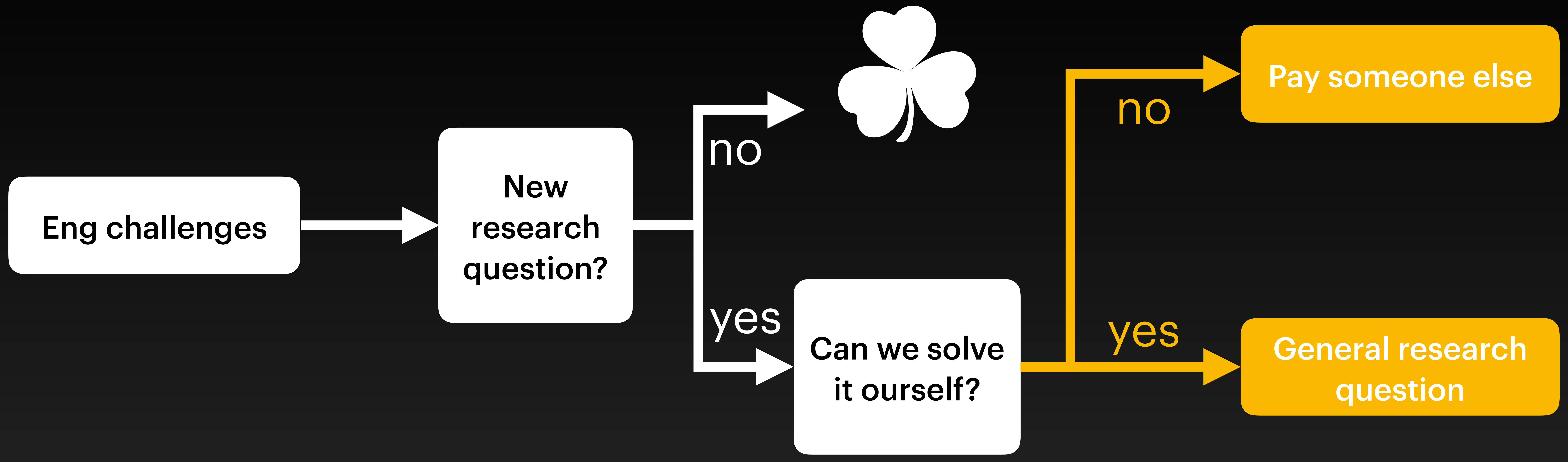
👍 2

Projects Roadmap



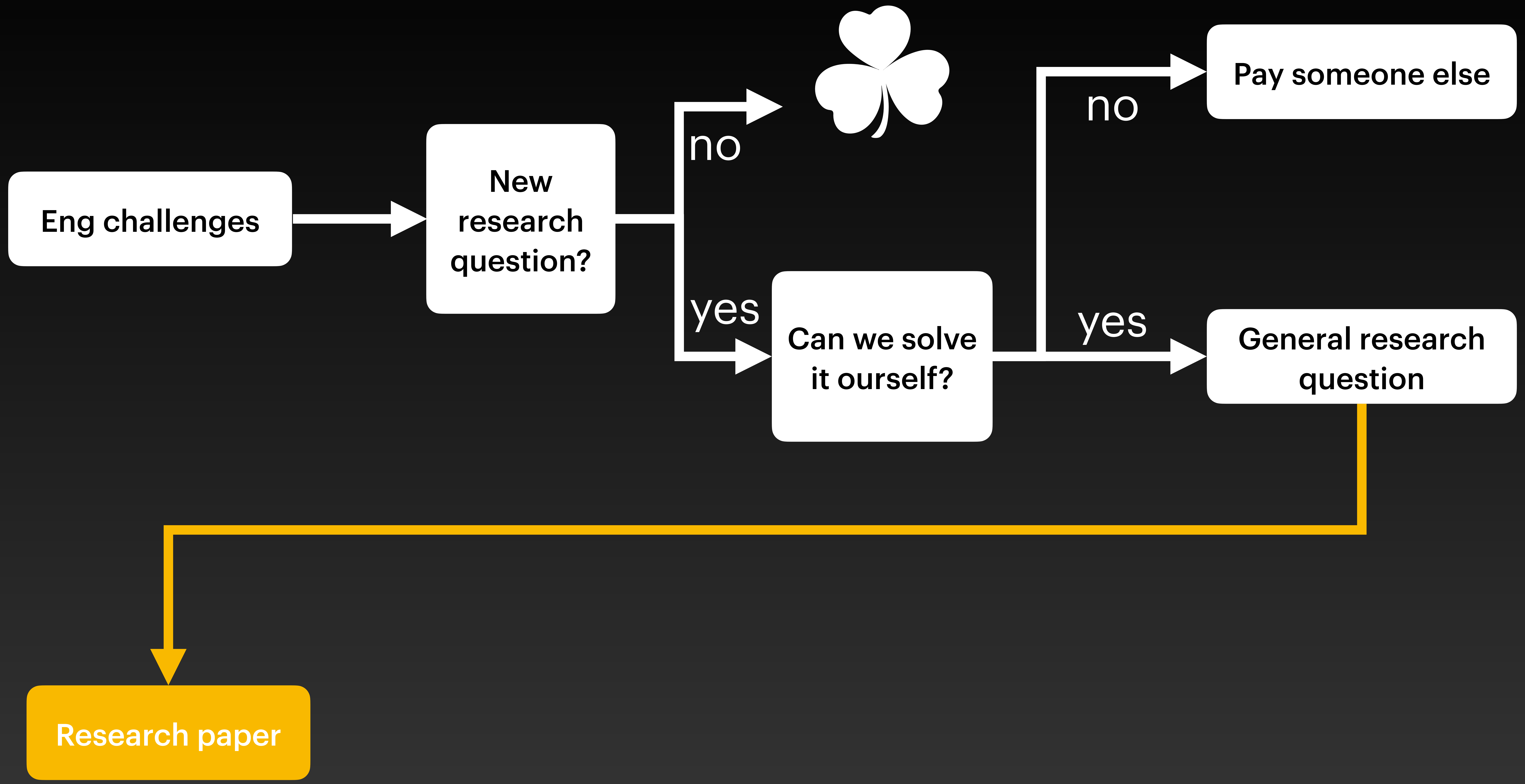

Eng challenges
New research question?

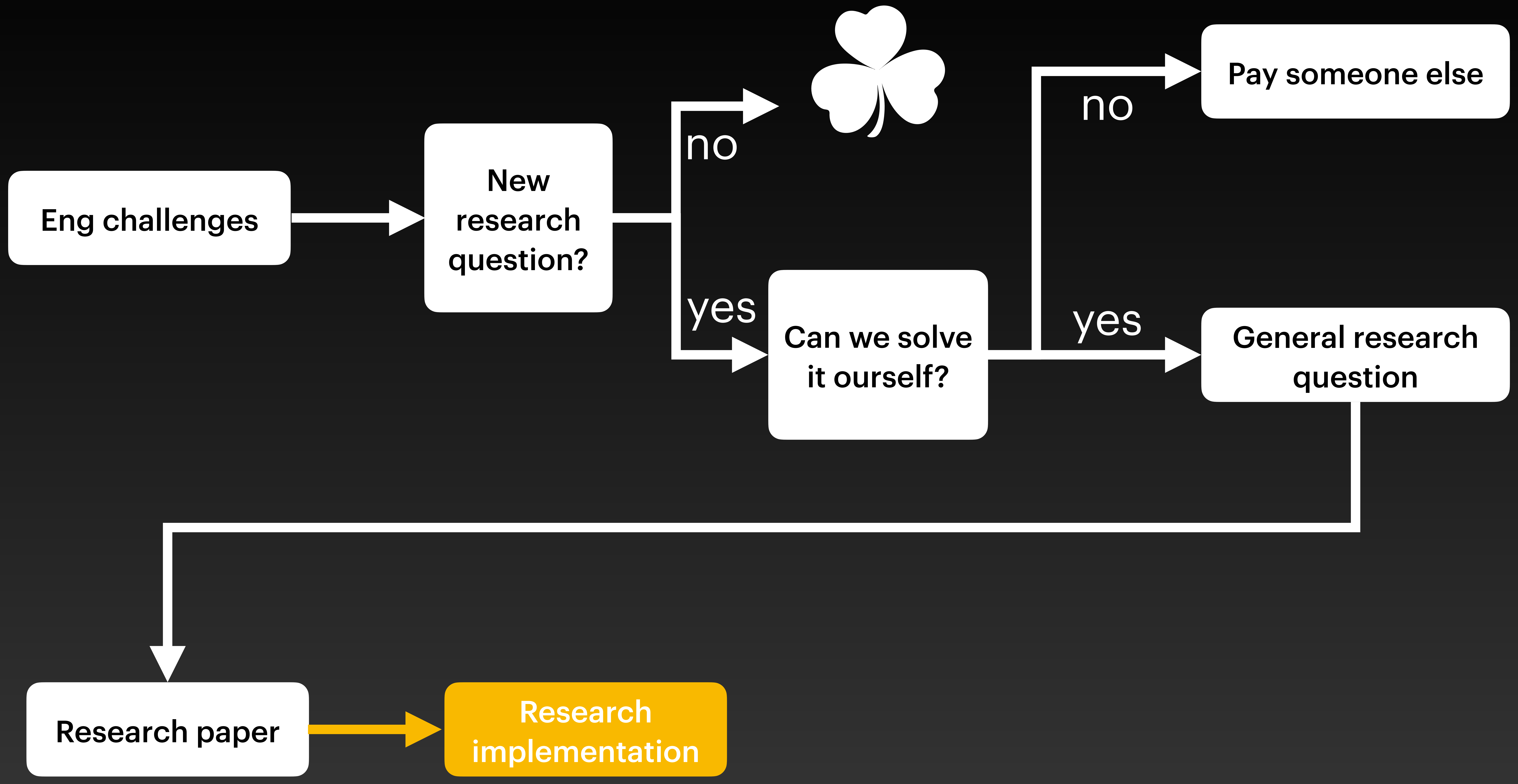

Eng challenges
New research question?
no
yes
Can we solve it ourself?

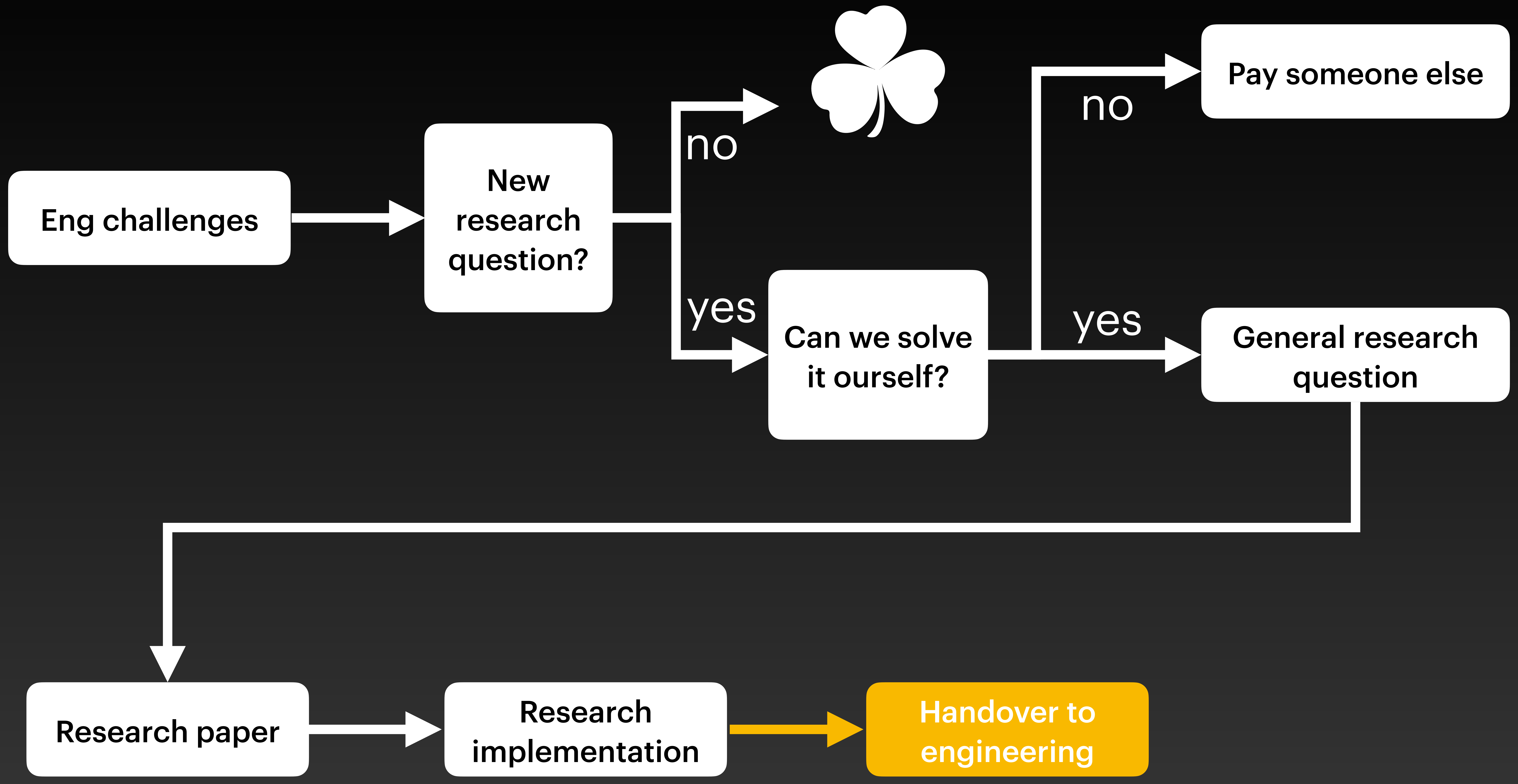

Eng challenges
New research question?
no
yes
Can we solve it ourself?
no
Pay someone else
yes
General research question

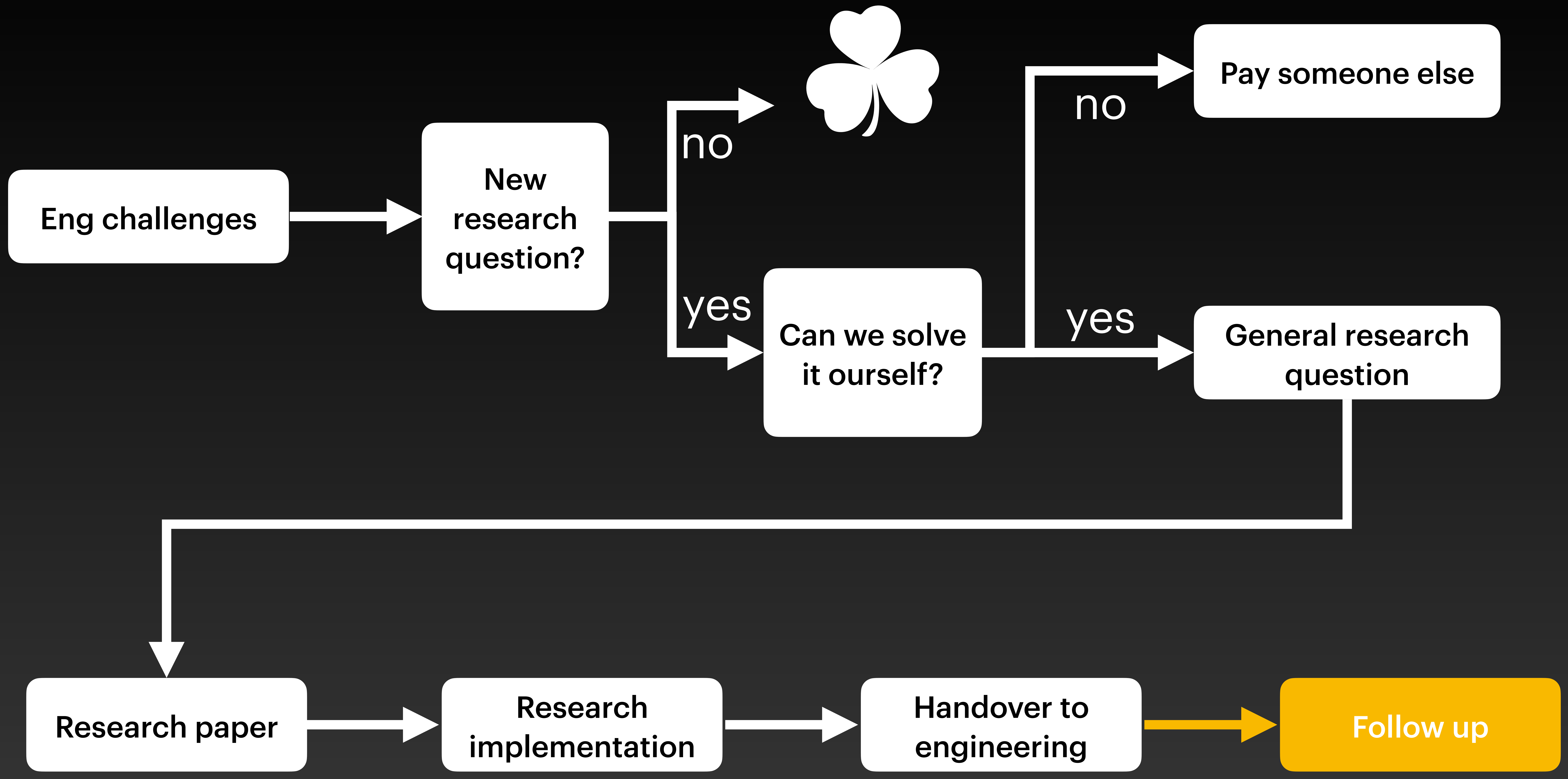
Research Gifts

(please keep it short)


Eng challenges
New research question?
no
yes
Can we solve it ourself?
no
Pay someone else
yes
General research question
Research paper


Eng challenges
New research question?
no
yes
Can we solve it ourself?
no
Pay someone else
yes
General research question
Research paper
Research implementation


Eng challenges
New research question?
no
yes
Can we solve it ourself?
no
Pay someone else
yes
General research question
Research paper
Research implementation
Handover to engineering


Eng challenges
New research question?
no
yes
Can we solve it ourself?
no
Pay someone else
yes
General research question
Research paper
Research implementation
Handover to engineering
Follow up

Chainspace: A Sharded Smart Contracts Platform
NDSS • Adopted by chainspace.io

Coconut: Threshold Issuance Selective Disclosure …
NDSS • Adopted by chainspace.io, Ketl, Nym, …

Replay Attacks and Defenses against Cross-shard …
EuroS&P • Adopted by chainspace.io

FastPay: High-Performance Byzantine Fault Tolerant …
AFT • Adopted by Sui, Linera

Twins: BFT Systems Made Robust
OPODIS • Adopted by Diem, Aptos, Chainlink

Fraud Proofs: Maximising Light Client Security and …
FC • Adoped by Ethereum, Celestia

Jolteon and Ditto: Network-Adaptive Efficient Consensus …
FC • Adopted by Flow, Diem, Aptos, Monad

Be Aware of Your Leaders
FC • Adopted by Diem, Aptos

Subset-optimized BLS Multi-signature with Key Aggregation
FC • Adopted by Fastcrypto

Narwhal and Tusk: A DAG-based Mempool and Efficient …
EuroSys • Adopted by Sui, Aptos, Fleek, Aleo
Best paper award

Bullshark: DAG BFT Protocols Made Practical
CCS • Adopted by Sui, Aleo, Fleek

Zef: Low-latency, Scalable, Private Payments
WPES • Adopted by Linera

Parakeet: Practical Key Transparency for End-to-End …
NDSS • Adopted by WhatsApp
IETF Applied Networking Research Prize

HammerHead: Leader Reputation for Dynamic Scheduling
ICDCS • Adopted by Sui

Fastcrypto: Pioneering Cryptography via Continuous …
LTB • Adopted by Fastcrypto

Sui Lutris: A Blockchain Combining Broadcast and …
CCS • Adopted by Sui
Distinguished paper award

Mysticeti: Reaching the Limits of Latency with Uncertified …
NDSS • Adopted by Sui

Research Questions

1. Network model?
2. BFT testing?
3. Consensus-exec interface?
4. Storage architecture?
5. Block synchroniser?
6. Realistic benchmarks?
7. Efficient reads?

Lessons Learned

1. Modularisation is a design strategy
2. Tasks-threads relationship
3. Benchmark early
4. Codesign with mem. and storage
5. Core is hard, consensus is easy
6. Epoch change is not an add-on
7. Writing papers to explore designs

EXTRA: Benchmarks

Implementation

- Written in Rust
- Networking: Tokio (TCP)
- Storage: custom WAL
- Cryptography: ed25519-consensus

https://github.com/mystenlabs/mysticeti

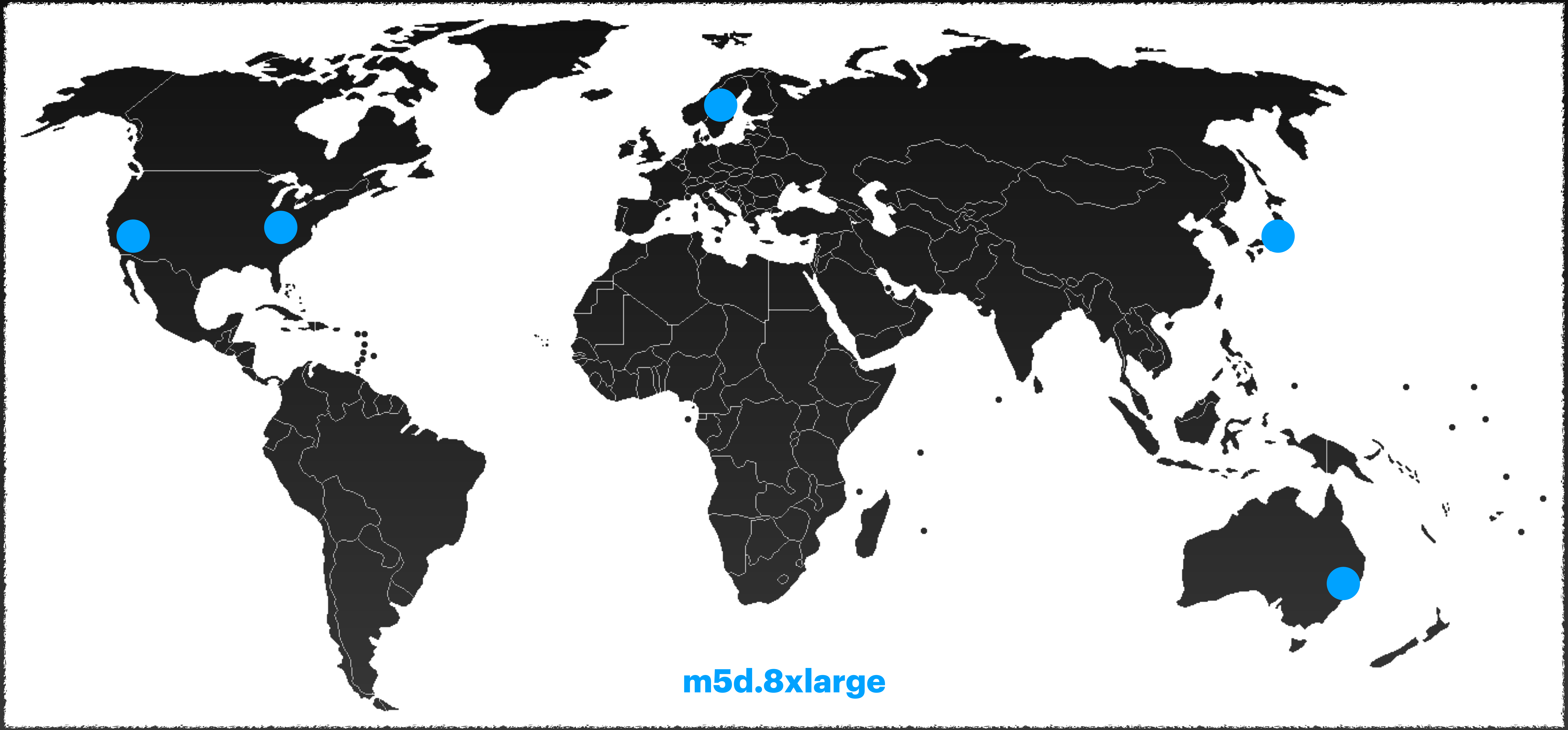
Implementation

- Synchronous core
- One Tokio task per peer (limiting resource usage)
- DTE simulator

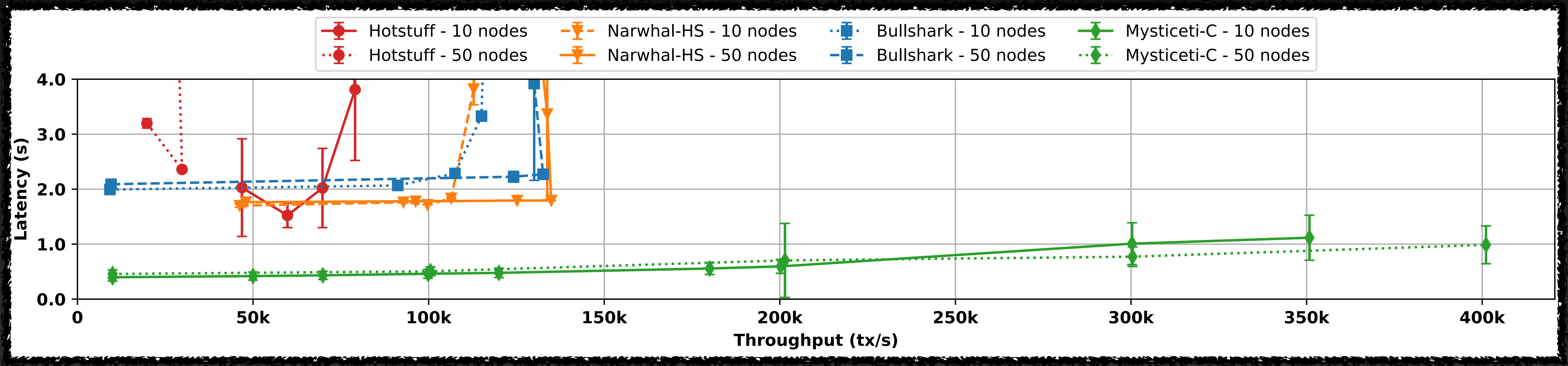
https://github.com/mystenlabs/mysticeti

Evaluation

Experimental setup on AWS



Prototype Benchmarks



Prototype Benchmarks