

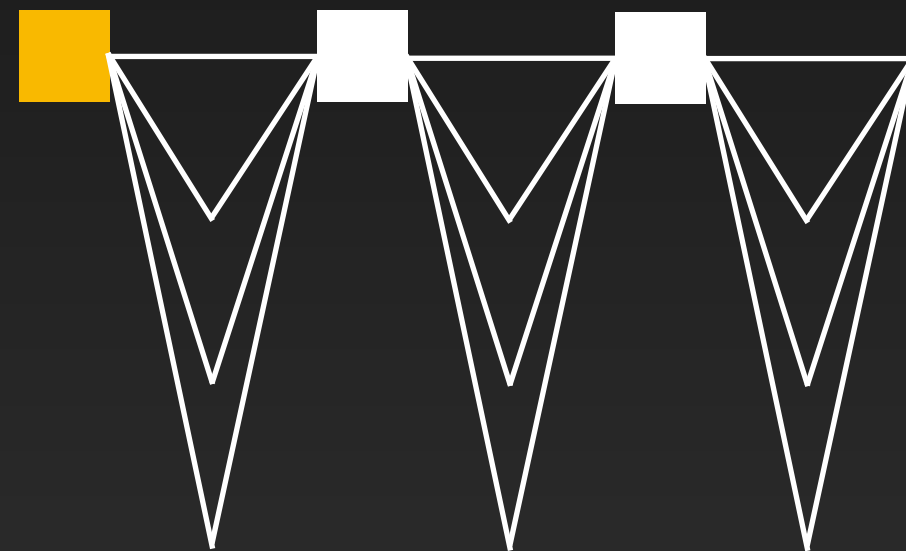
The Evolution of Sui

From Academic Paper to Mainnet

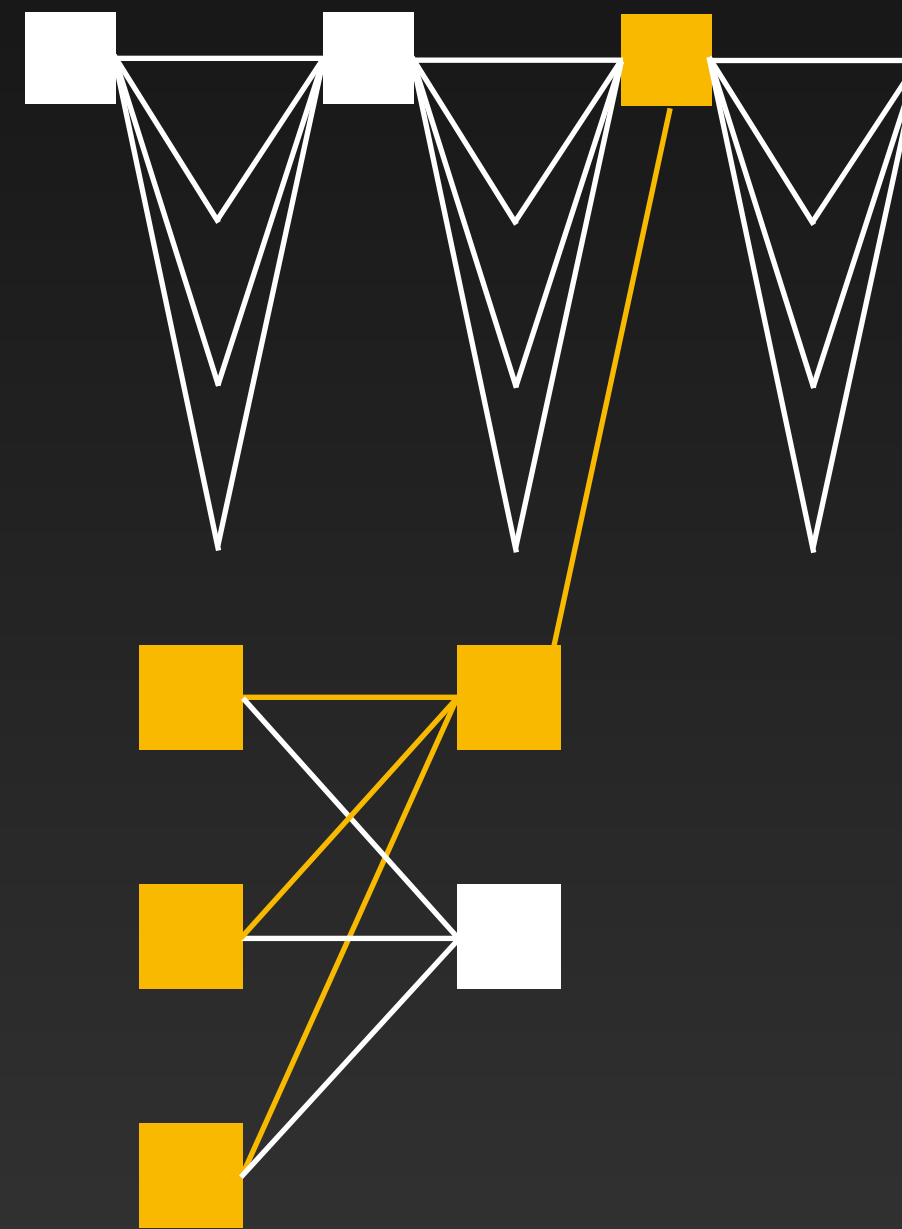
2019



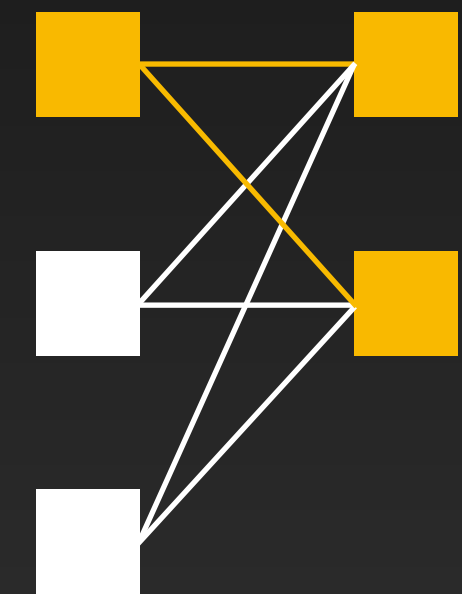
2024



HotStuff



HotStuff + Mempool



**Bullshark,
Mysticeti**

Libra, 2019

HotStuff

HotStuff: BFT Consensus in the Lens of Blockchain

Maofan Yin^{1,2}, Dahlia Malkhi², Michael K. Reiter^{2,3}, Guy Golan Gueta², and Ittai Abraham²

¹Cornell University, ²VMware Research, ³UNC-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



HashGraph

Verifying the Hashgraph Consensus Algorithm

Karl Crary

Carnegie Mellon University

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

Libra, 2019

HotStuff: BFT Consensus in the Lens of Blockchain

Maofan Yin^{1,2}, Dahlia Malkhi², Michael K. Reiter^{2,3}, Guy Golan Gueta², and Ittai Abraham²

¹Cornell University, ²VMware Research, ³UNC-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

HotStuff

✓ Linear

✓ Clearly isolated components

HashGraph

✗ Hard to garbage collect

✗ Unclear block synchroniser

The first 6 months...

SMR in the Libra Blockchain

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

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].

Research Questions

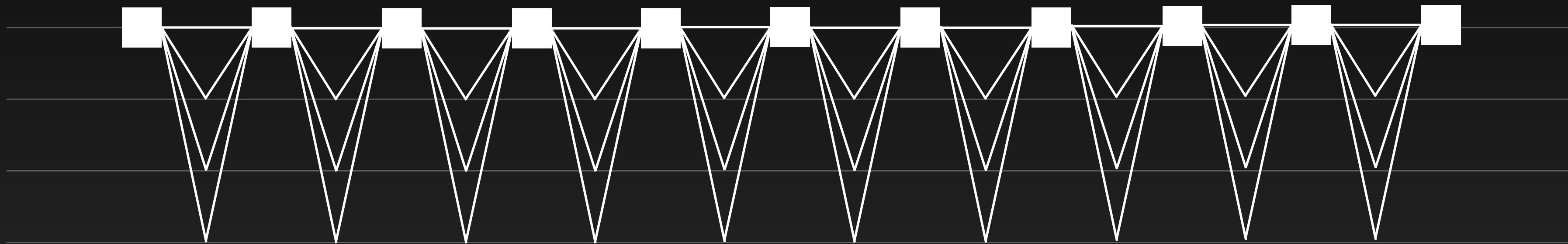
1. Network model?

Lessons Learned

1. Modularisation is a design strategy

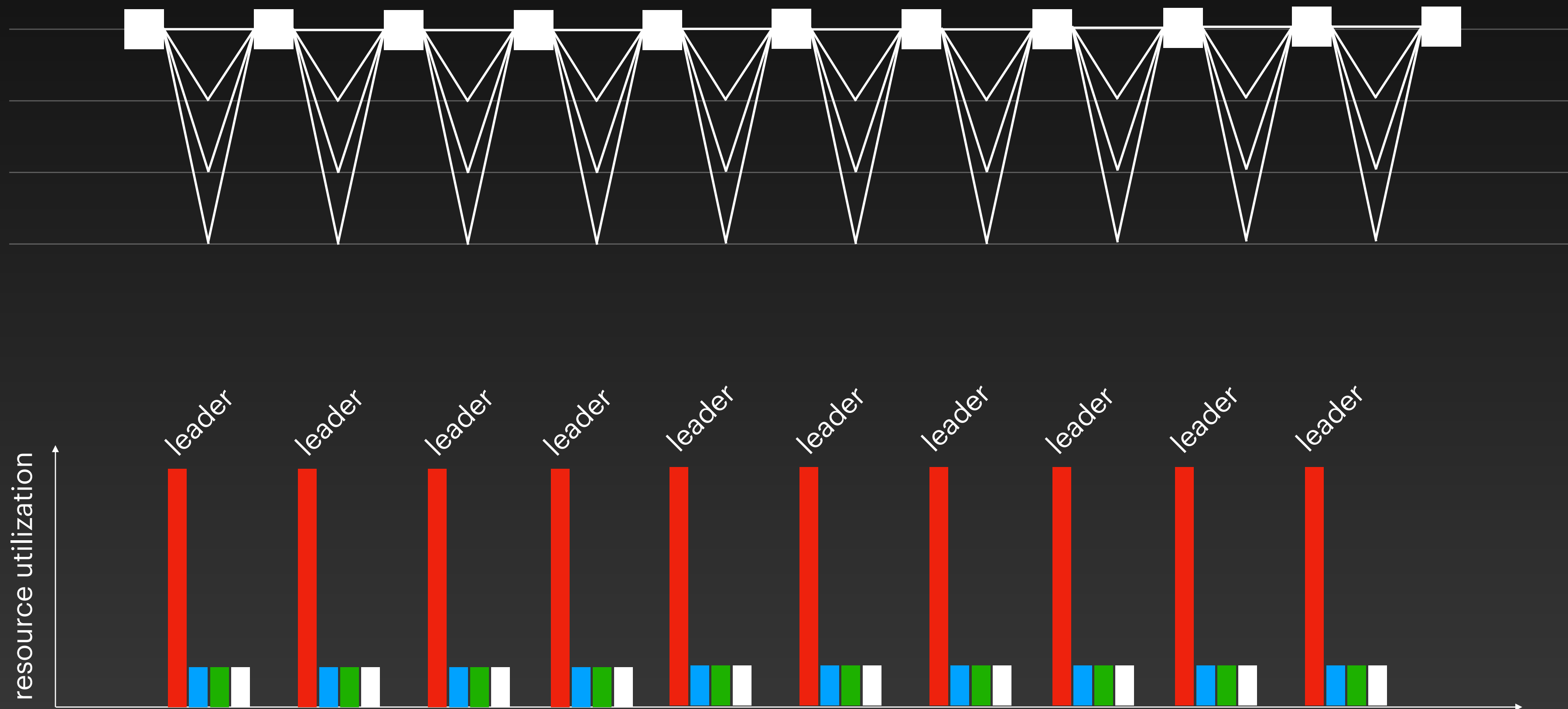
HotStuff

Typical leader-based protocols



Naive Implementation

Uneven resource utilisation



Research Questions

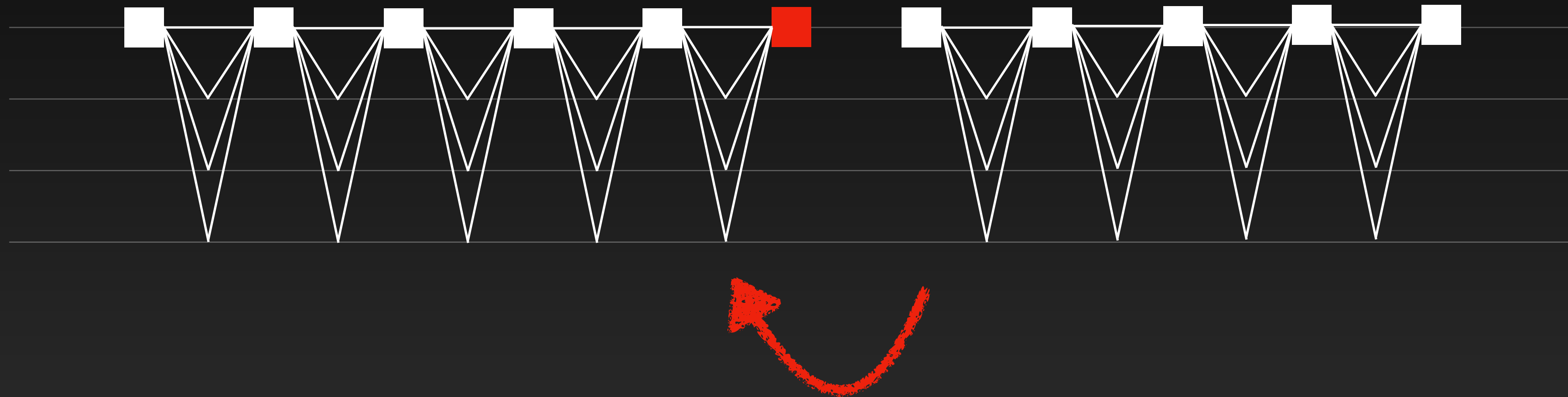
1. Network model?

Lessons Learned

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

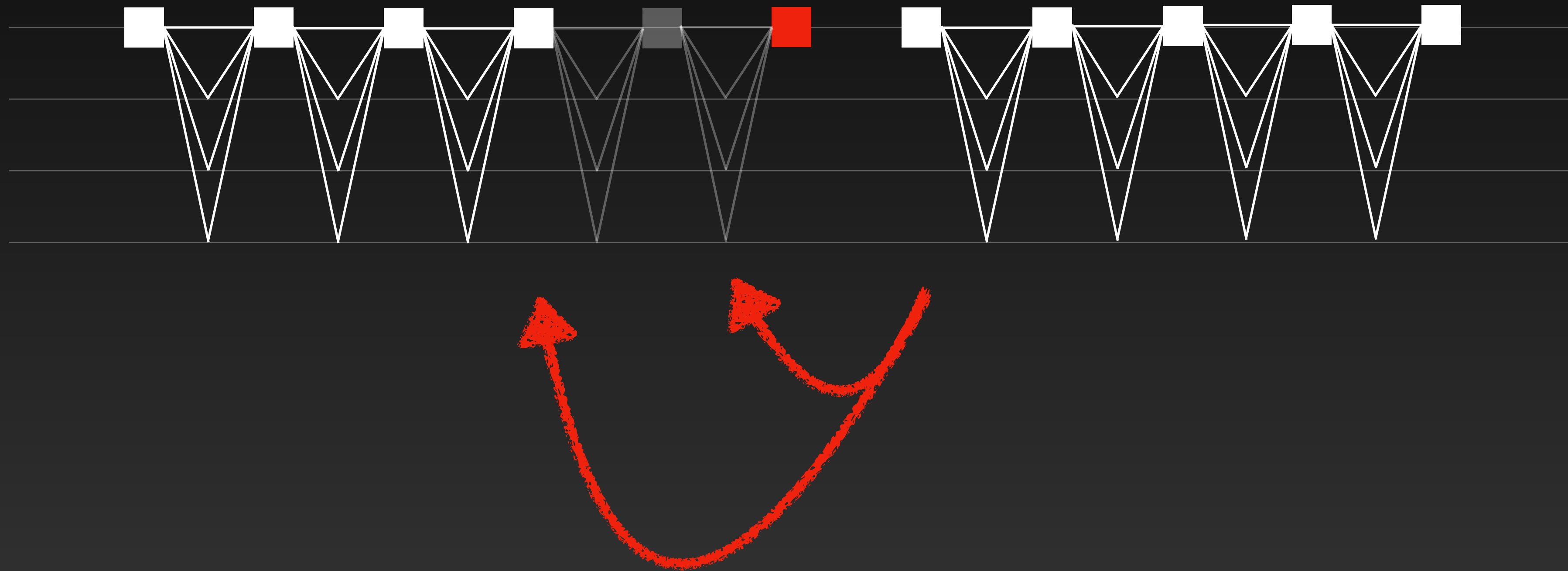
Leader-Driven Consensus

Fragility to faults and asynchrony

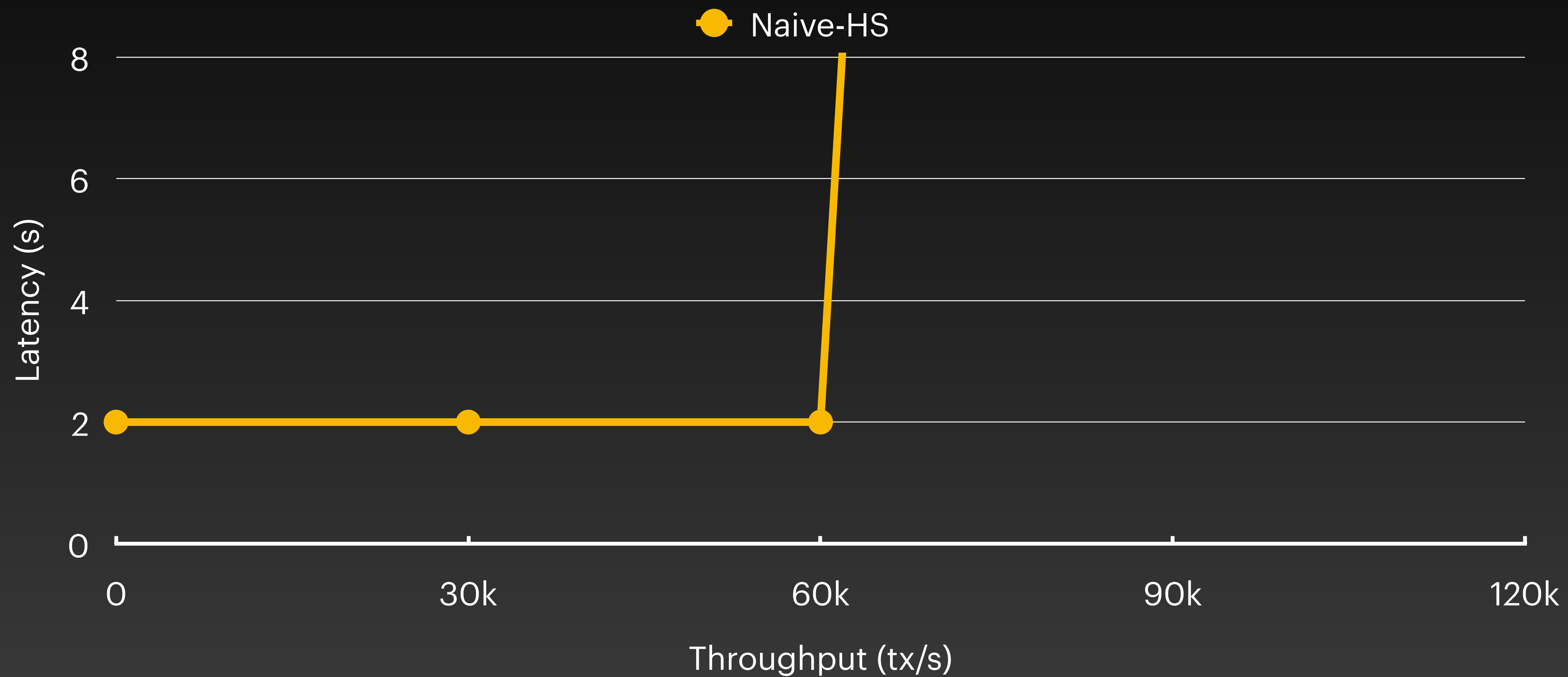


Leader-Driven Consensus

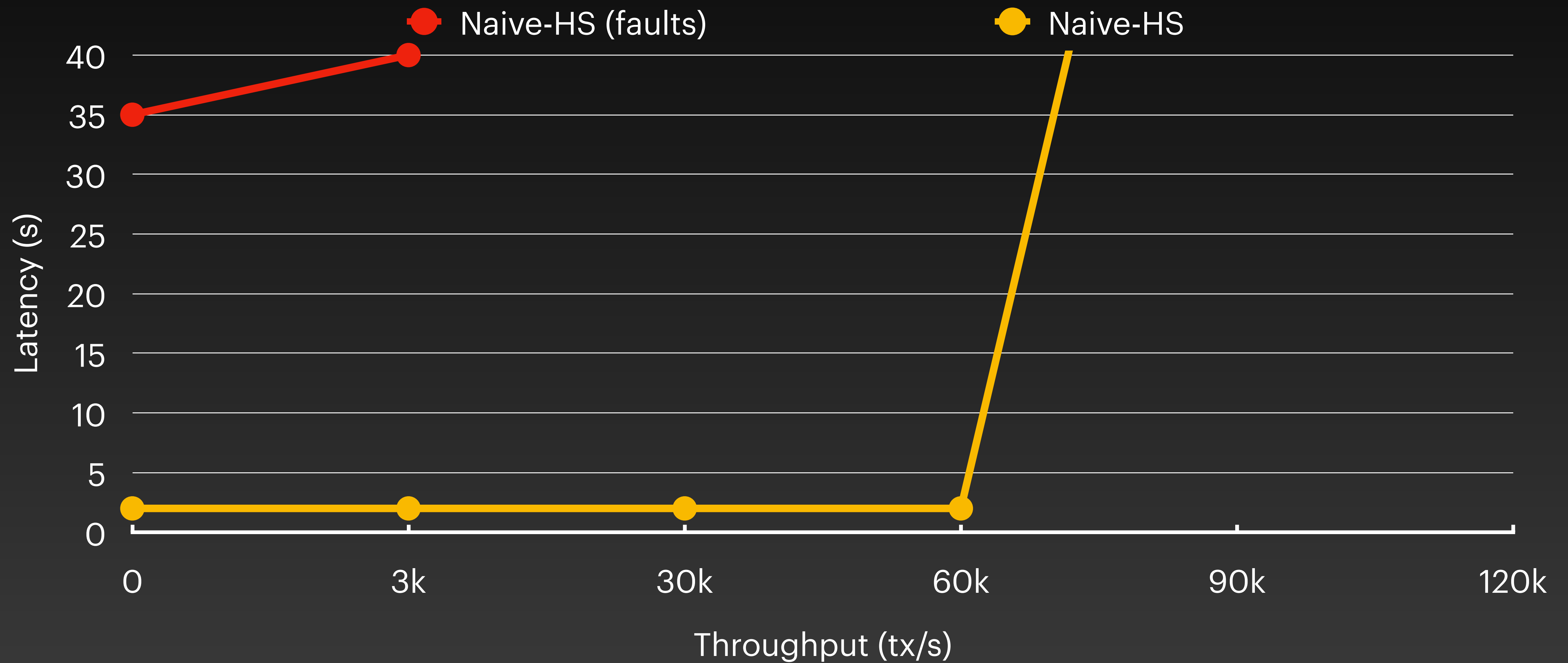
Fragility to faults and asynchrony



Performance



Performance



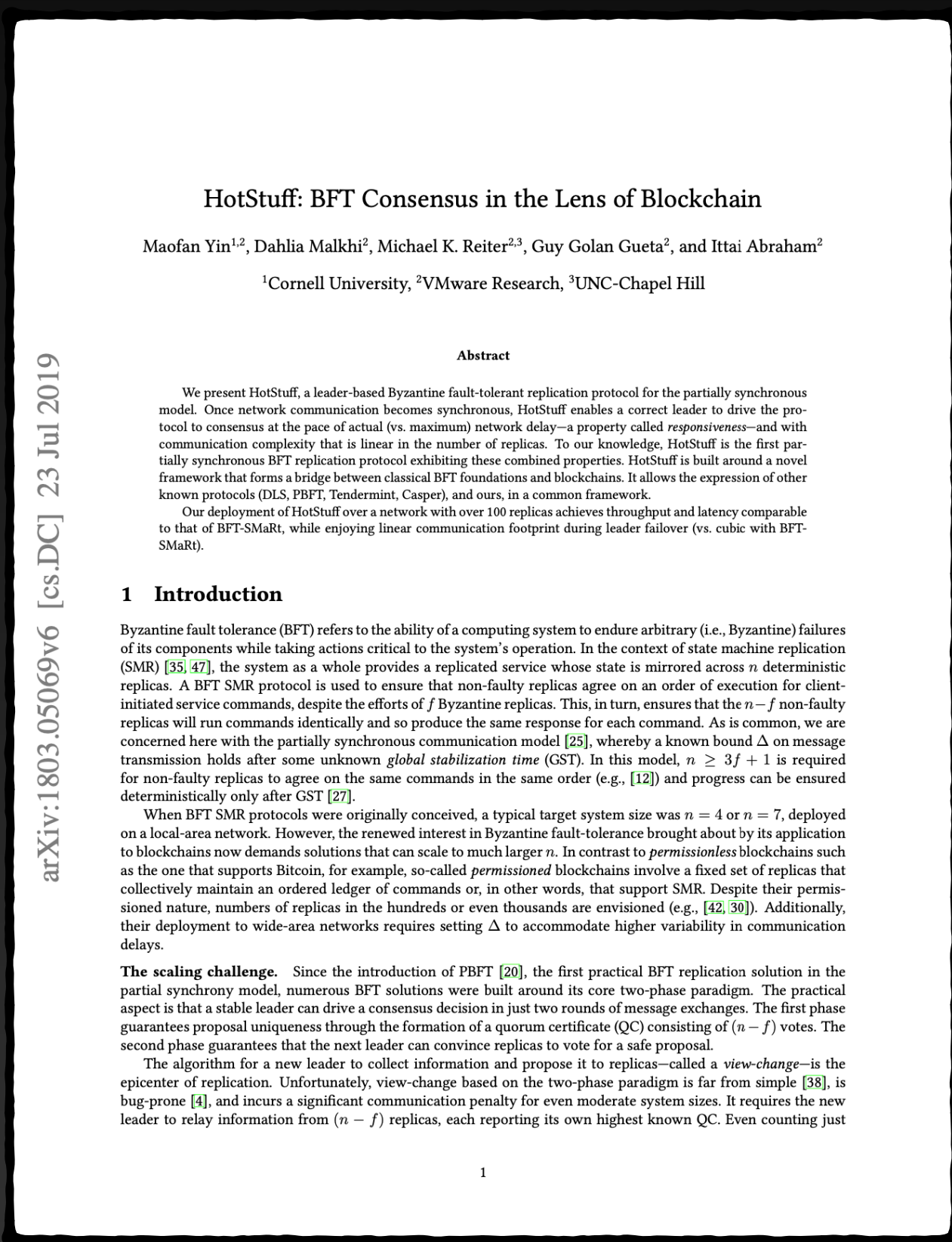
Research Questions

1. Network model?

Lessons Learned

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

Libra, 2019



HotStuff (naive mempool)

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

Libra, 2021

Narwhal

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

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

George Danezis
Mysten Labs & UCL

Alberto Sonnino
Mysten Labs

Lefteris Kokoris-Kogias
IST Austria

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

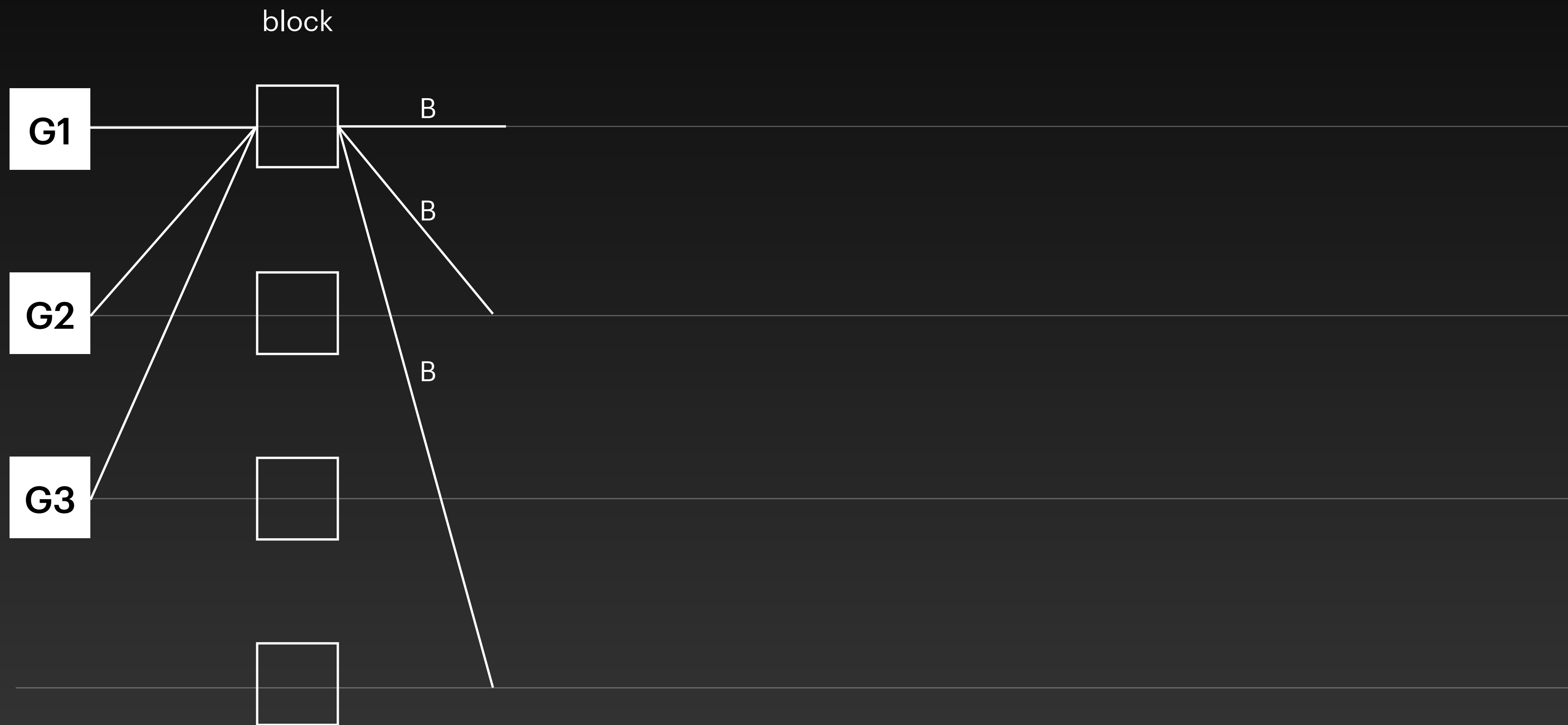
Byzantine consensus protocols [15, 19, 21] and the state machine replication paradigm [13] for building reliable distributed systems have been studied for over 40 years. However, with the rise in popularity of blockchains there has been a renewed interest in engineering high-performance consensus protocols. 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:

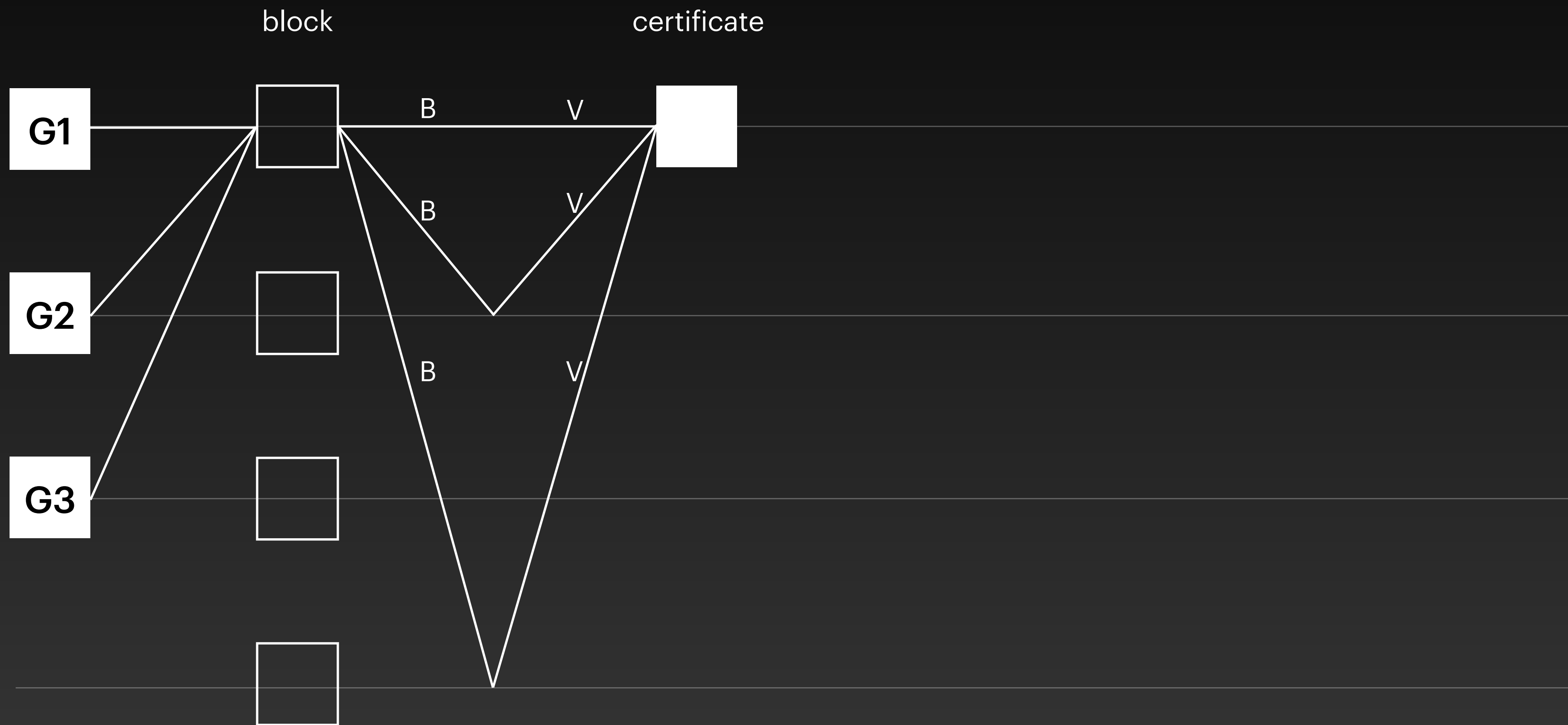
- Any (partially-synchronous) protocol that minimizes overall message number, but relies on a leader to produce proposals and coordinate consensus, fails to capture the high load this imposes on the leader who inevitably becomes a bottleneck.
- Message complexity counts the number of *metadata* messages (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

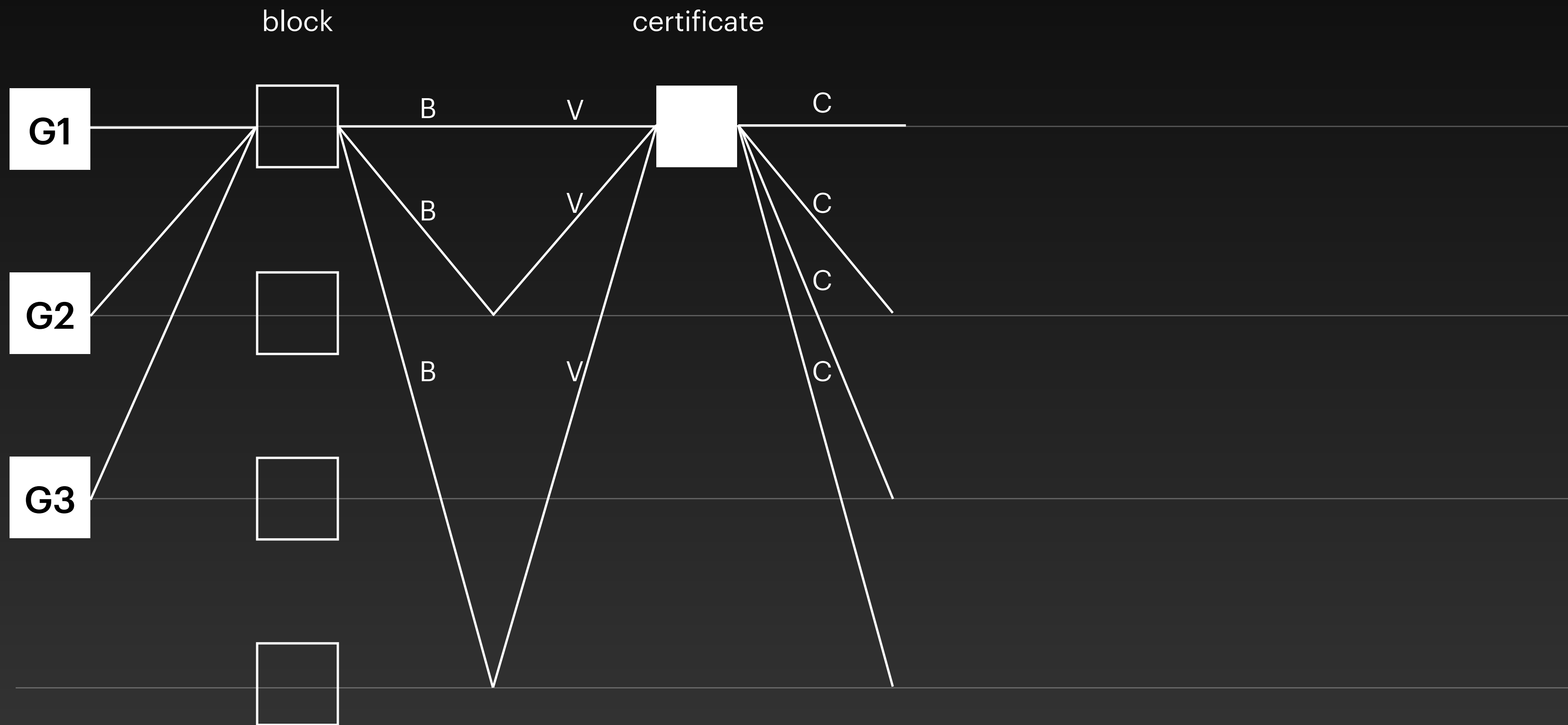
Narwhal



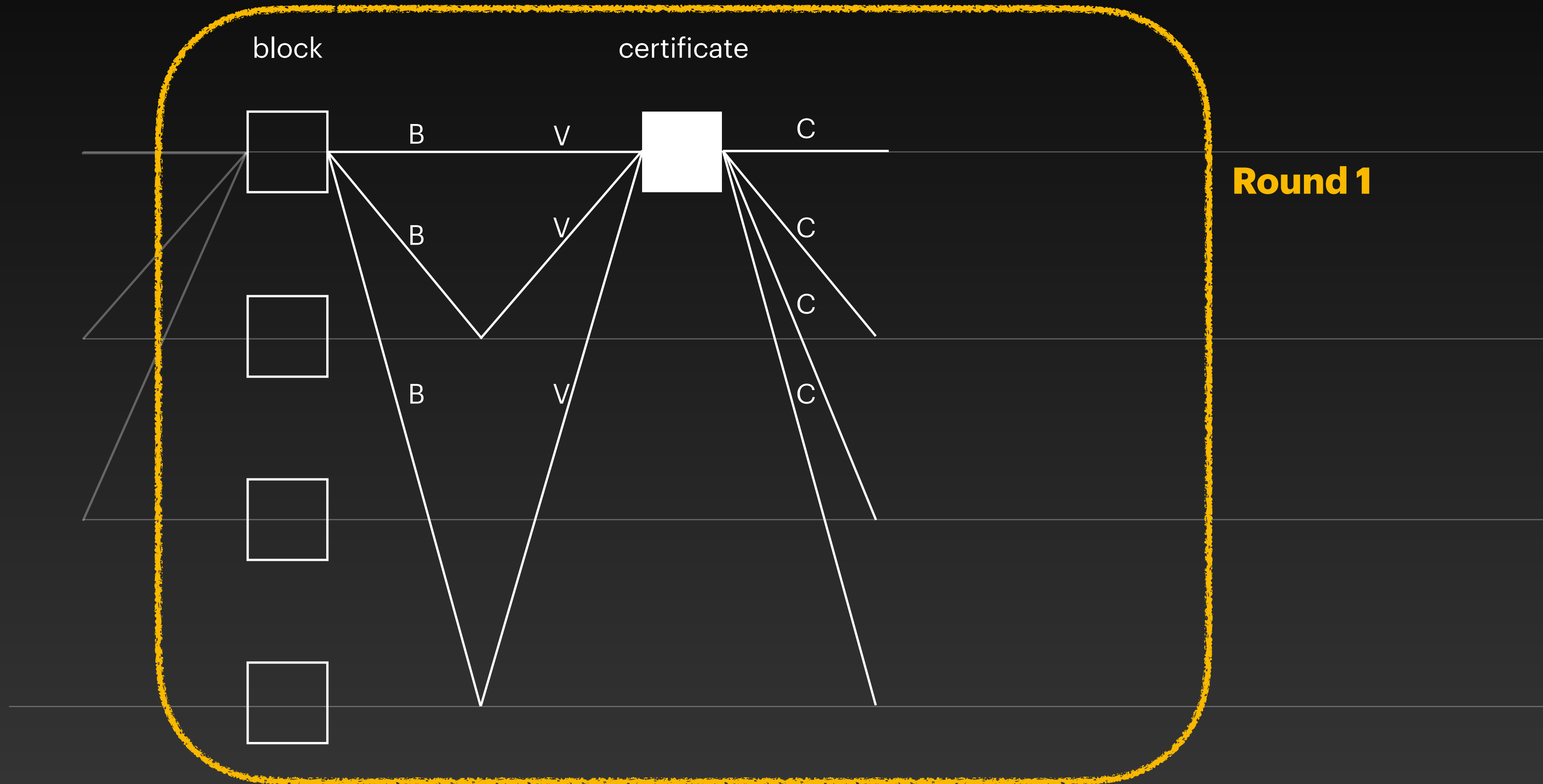
Narwhal



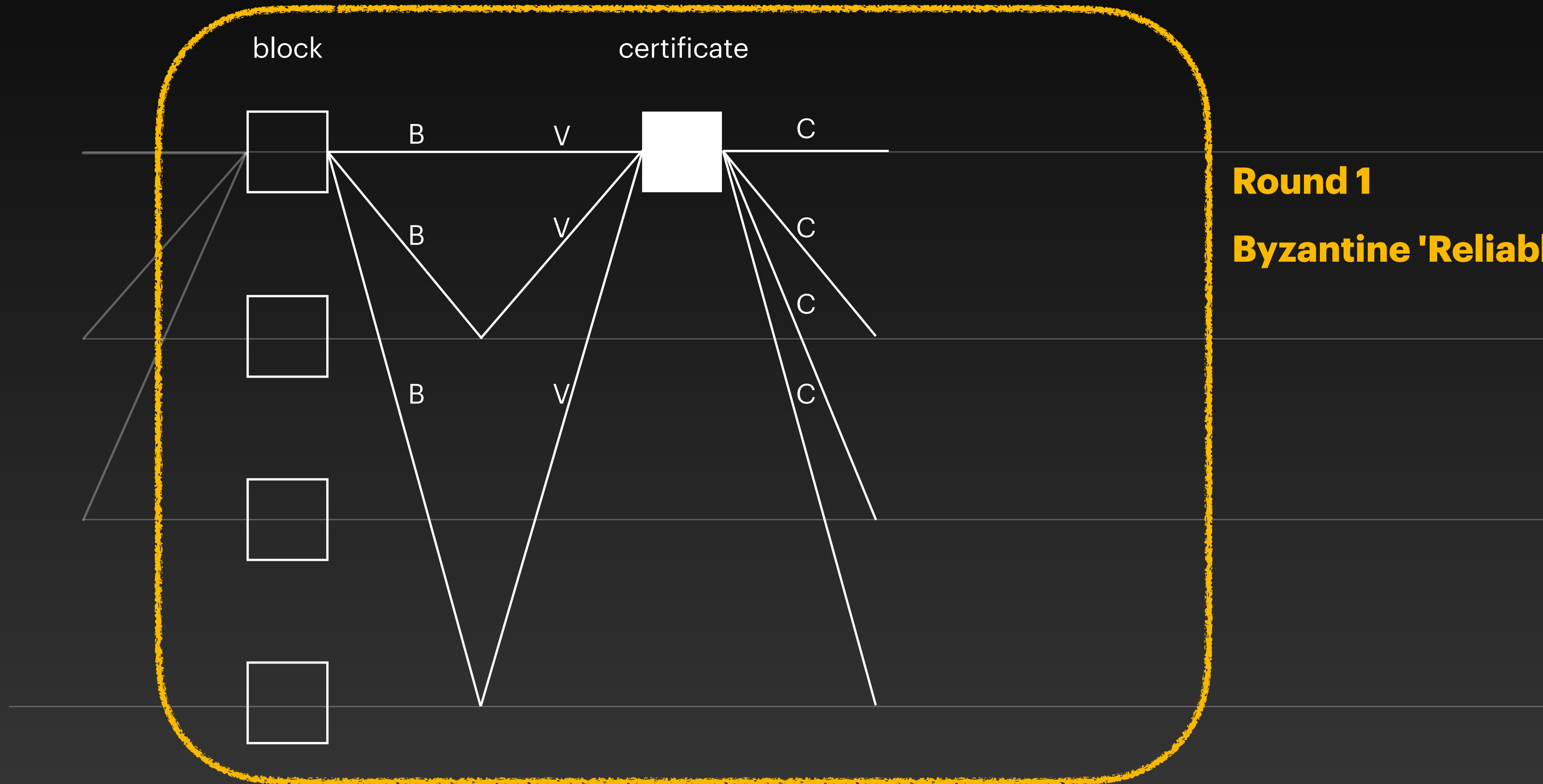
Narwhal



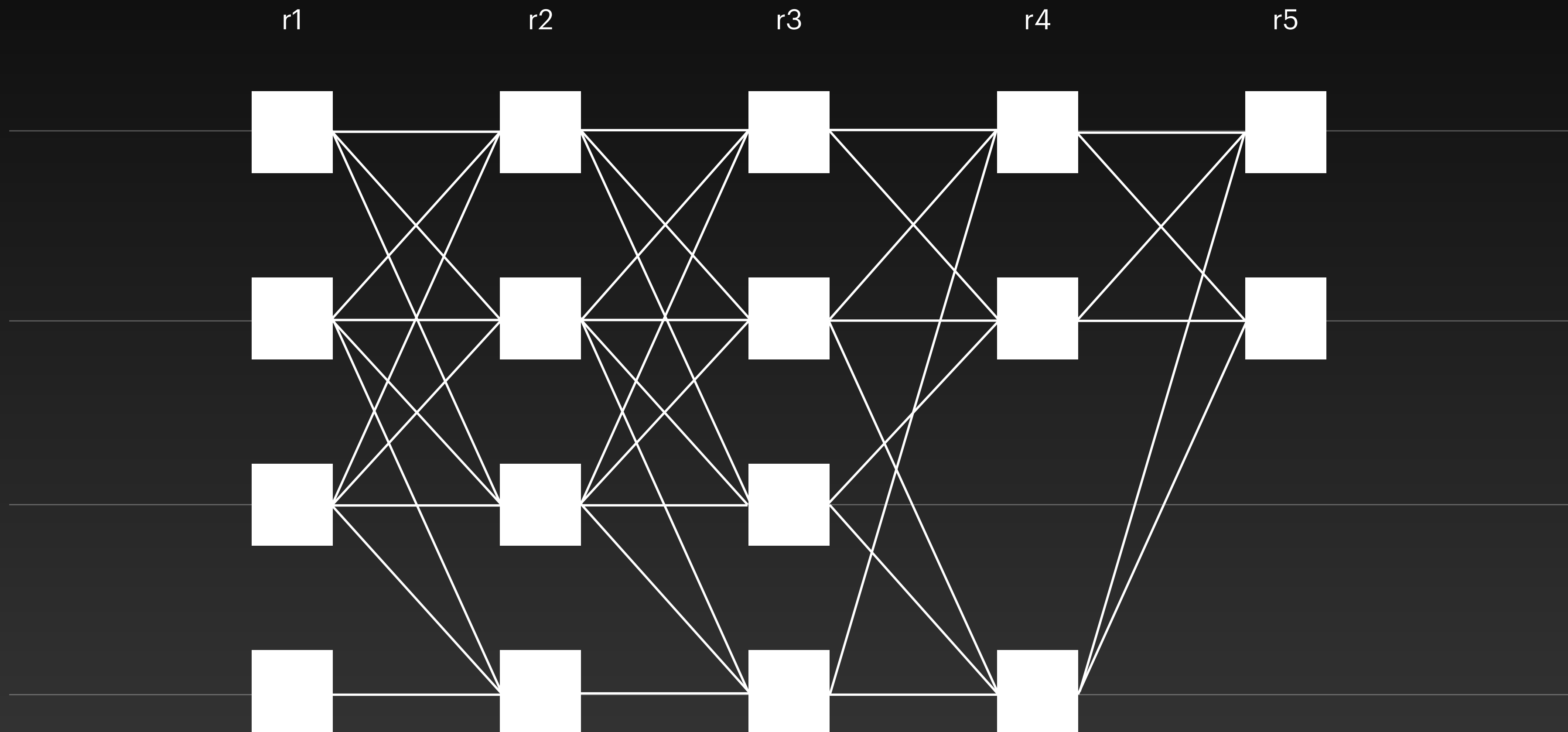
Narwhal



Narwhal



Narwhal



Research Questions

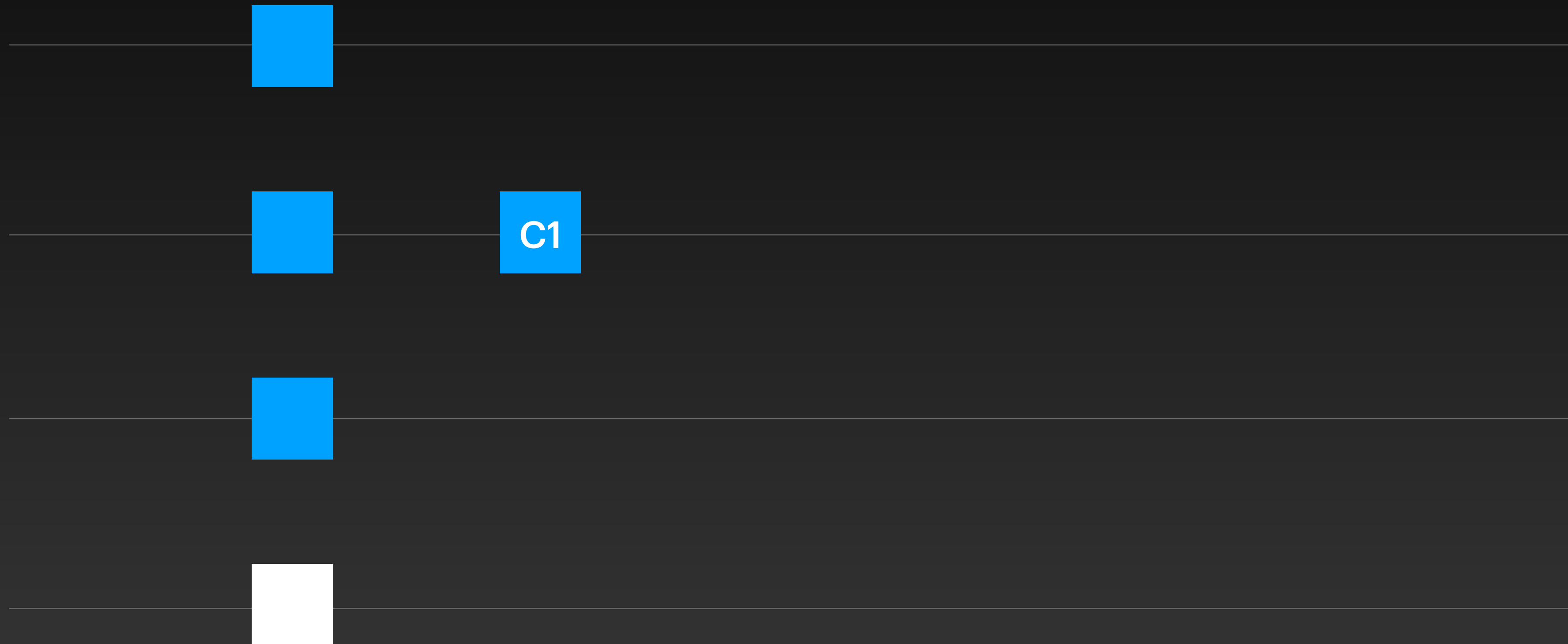
1. Network model?

Lessons Learned

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

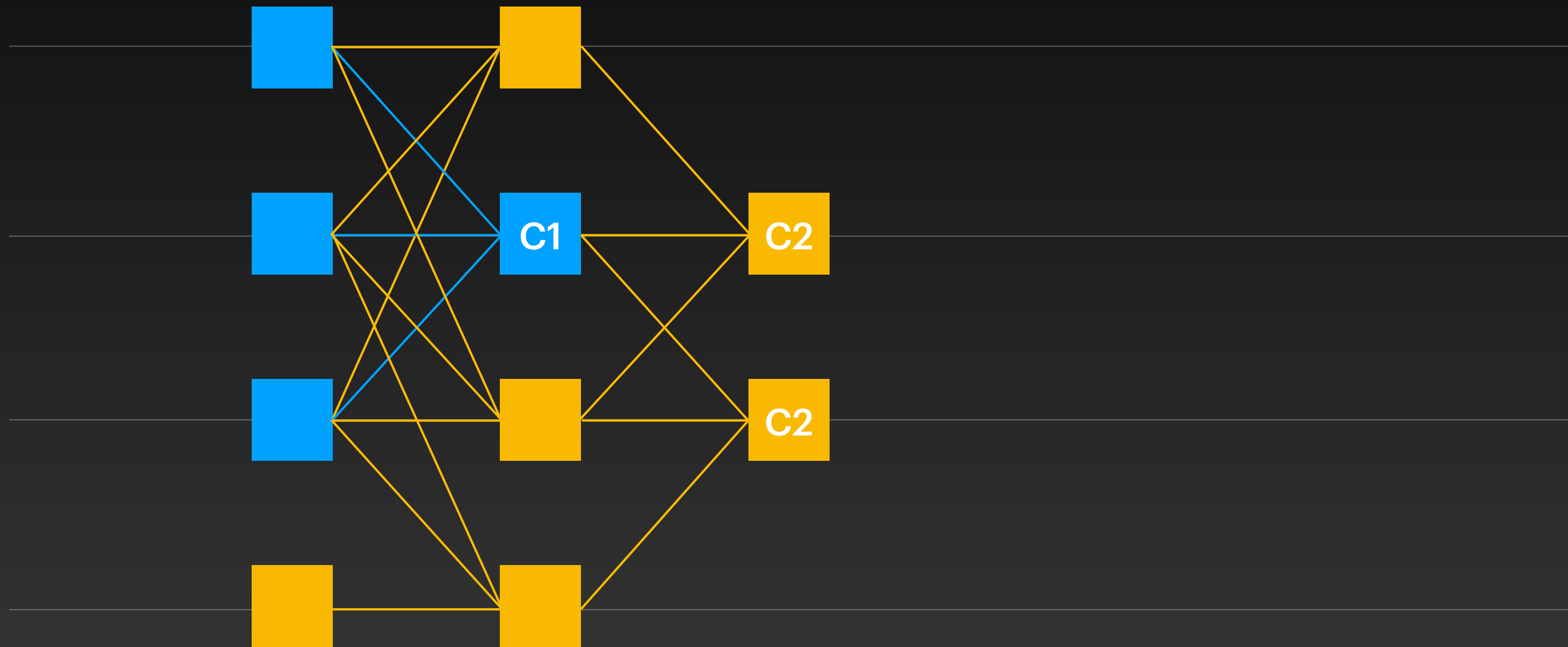
HotStuff on Narwhal

Enhanced commit rule



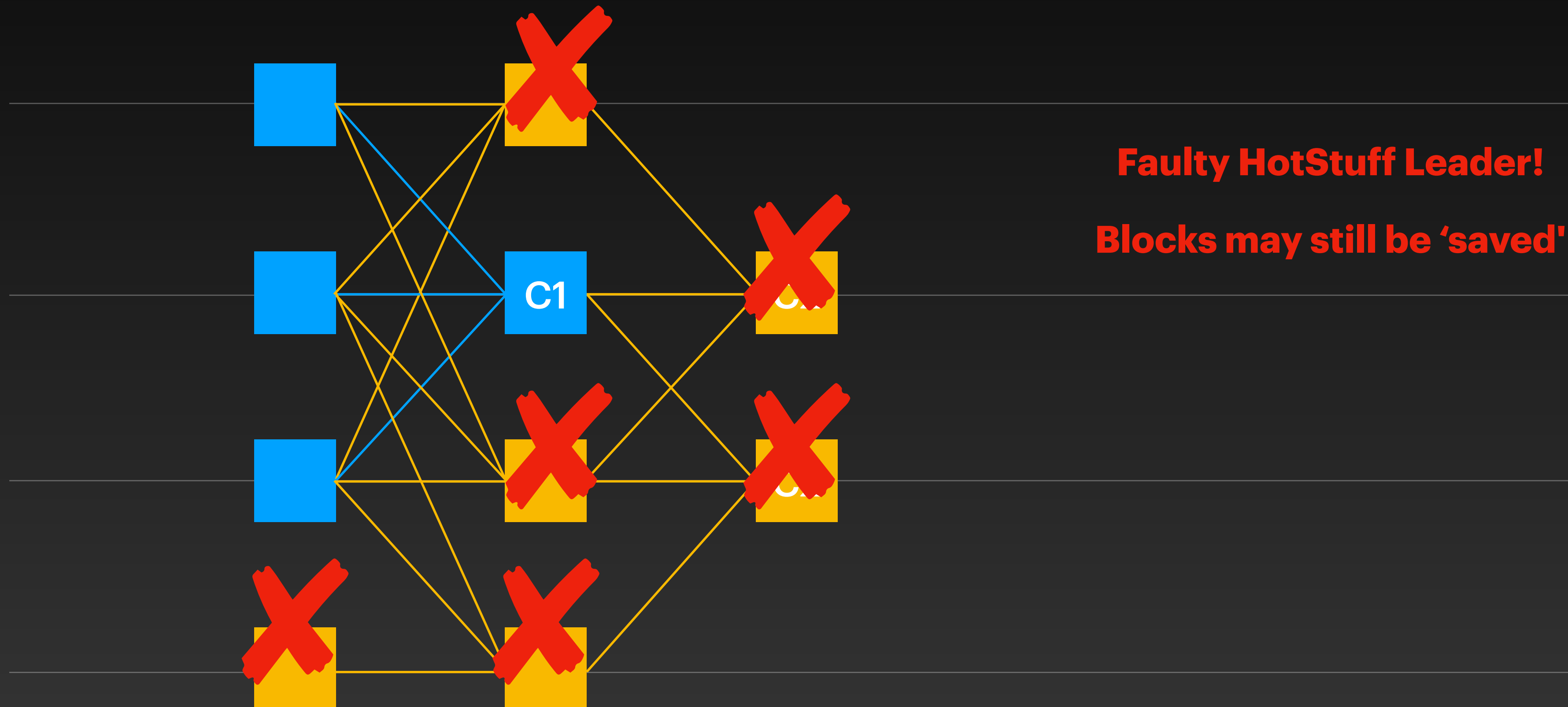
HotStuff on Narwhal

Enhanced commit rule



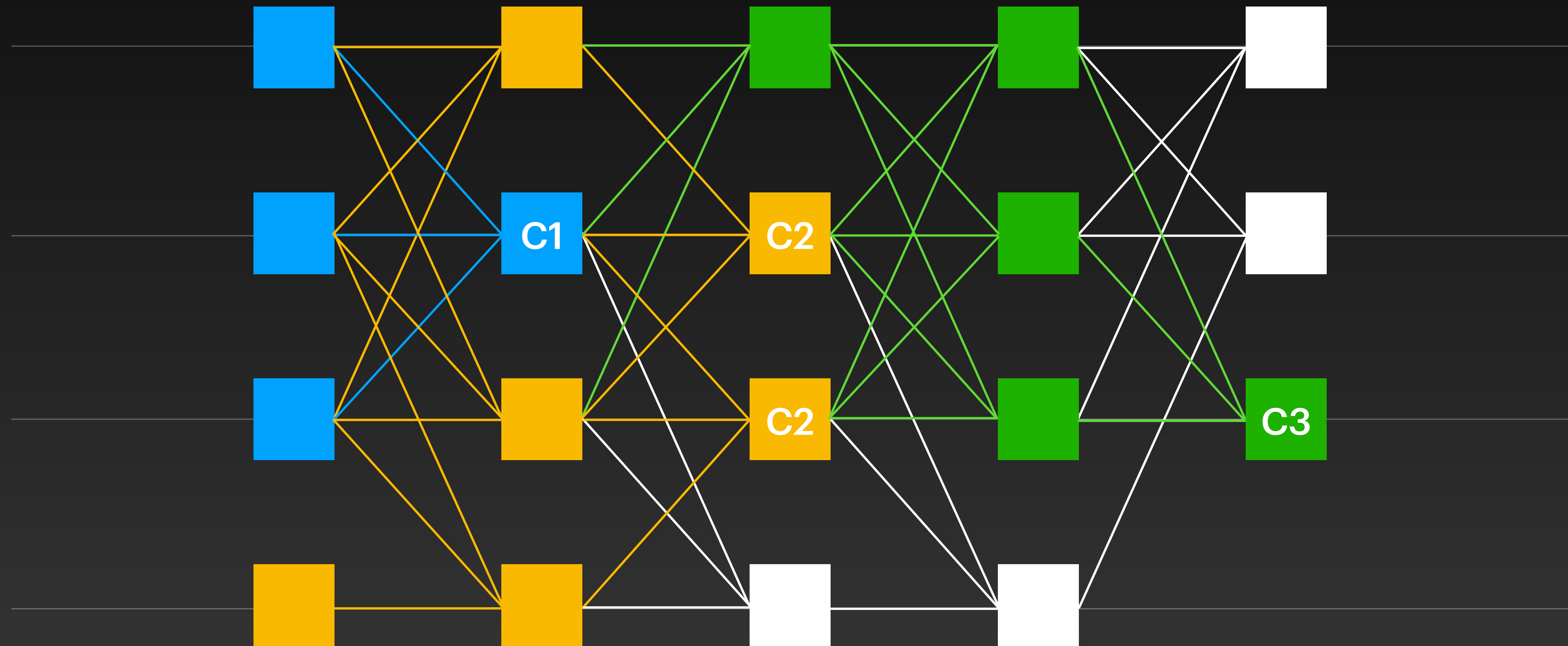
HotStuff on Narwhal

Enhanced commit rule



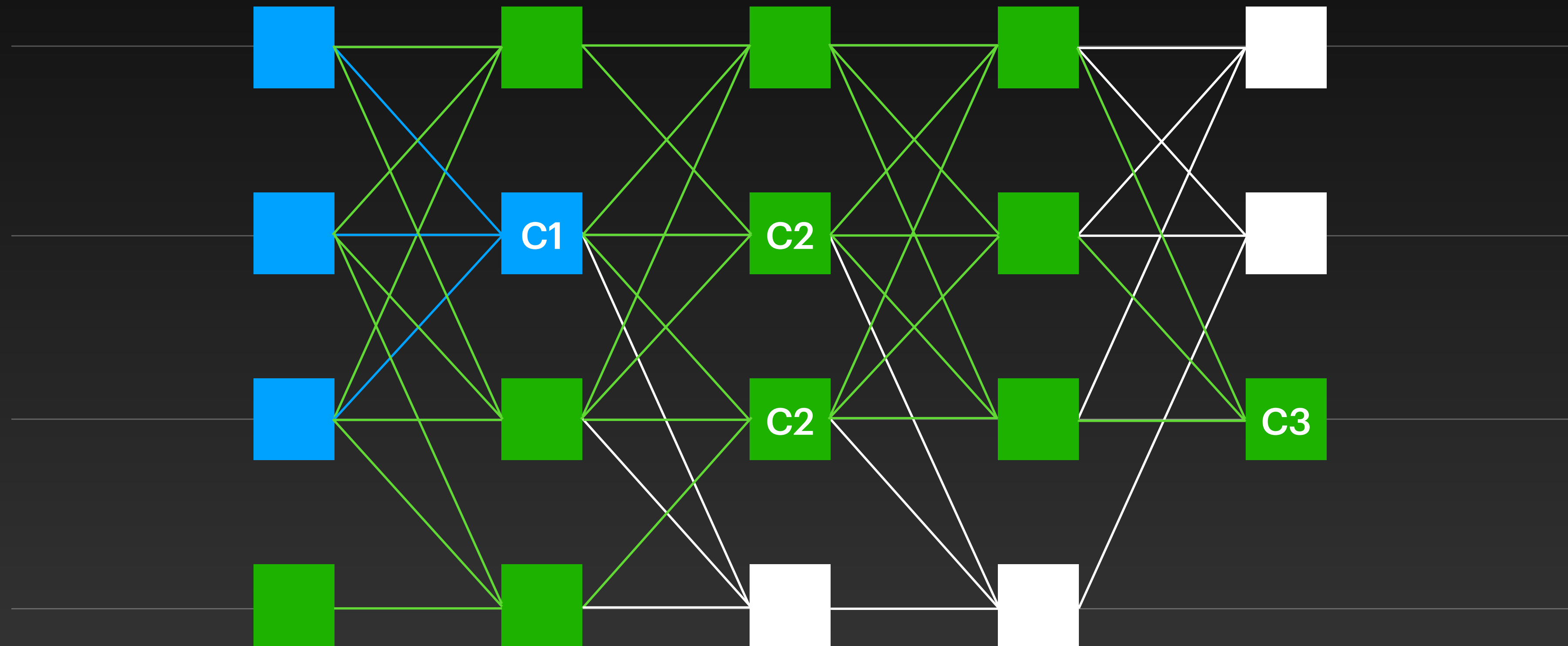
HotStuff on Narwhal

Enhanced commit rule

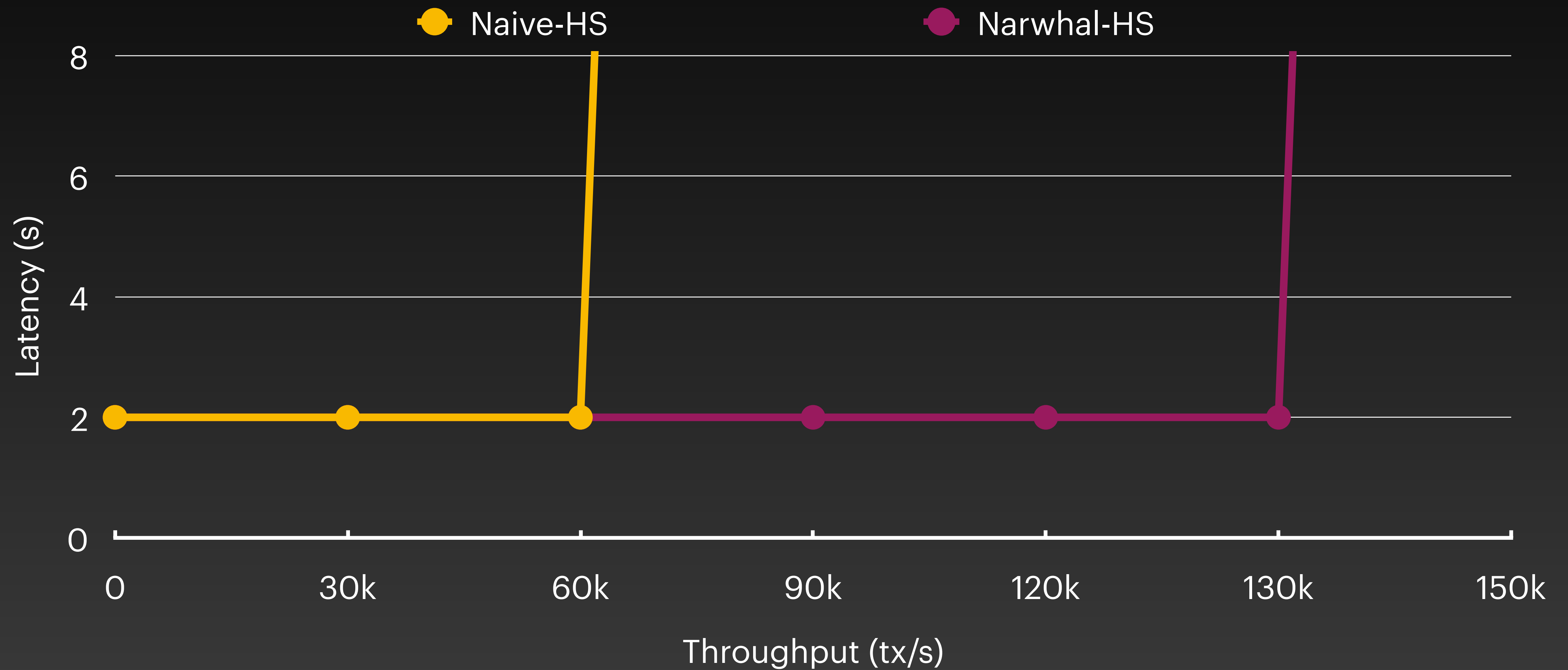


HotStuff on Narwhal

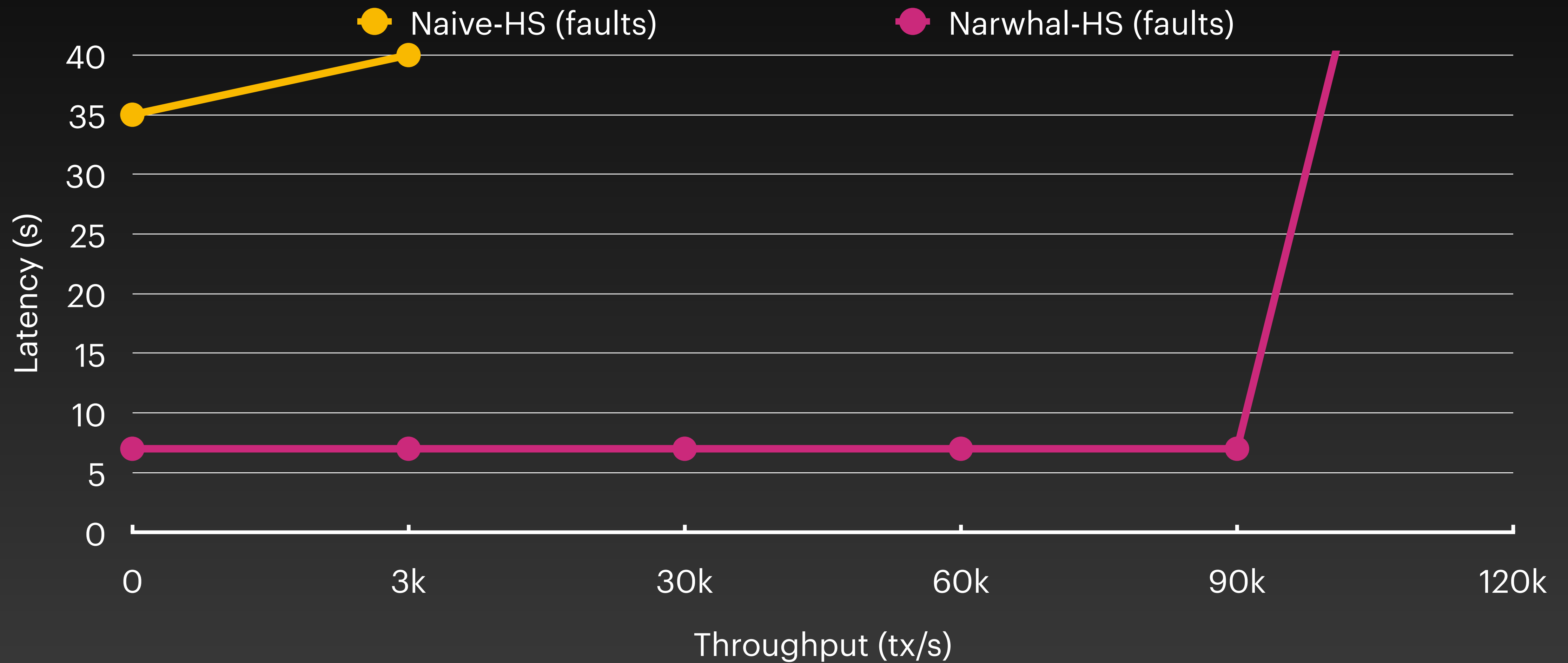
Enhanced commit rule



Performance

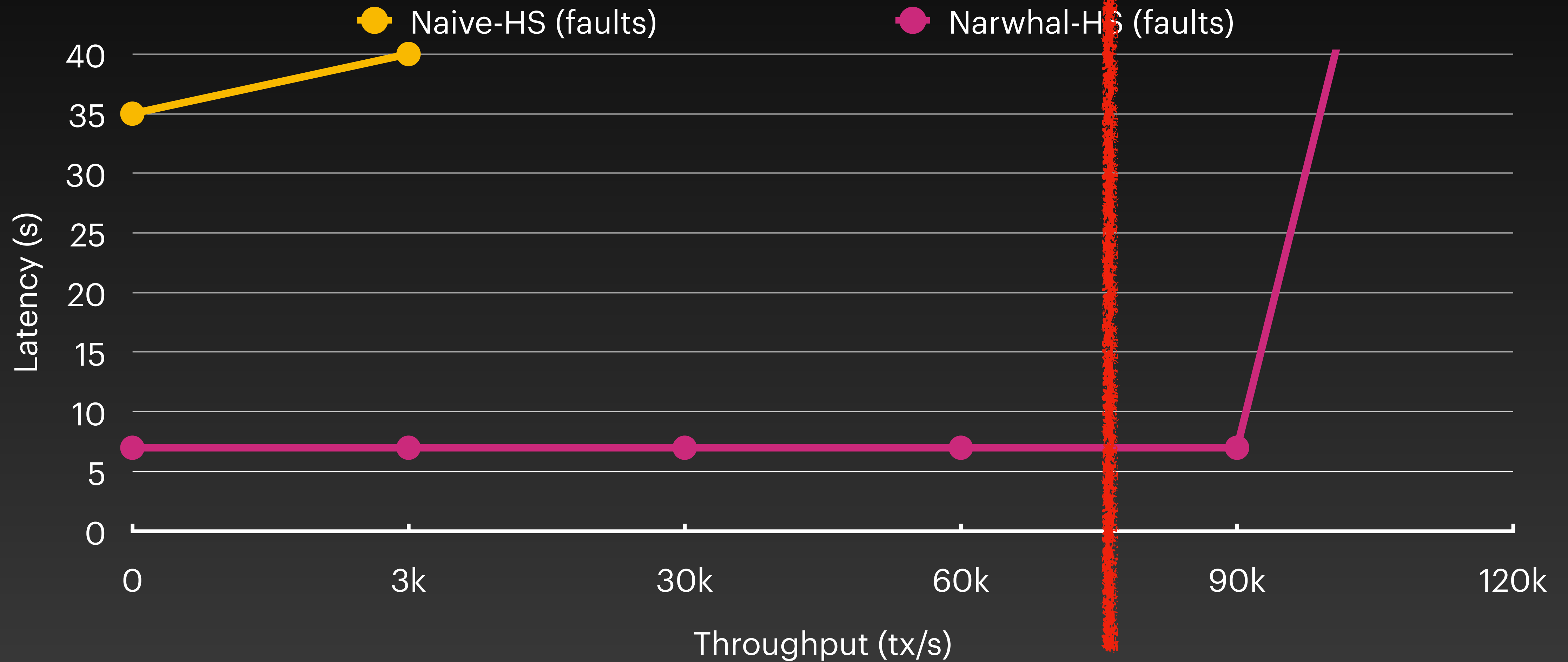


Performance



Performance

visa+mastercard



Libra, 2021

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

George Danezis
Mysten Labs & UCL

Alberto Sonnino
Mysten Labs

Lefteris Kokoris-Kogias
IST Austria

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
- High engineering complexity

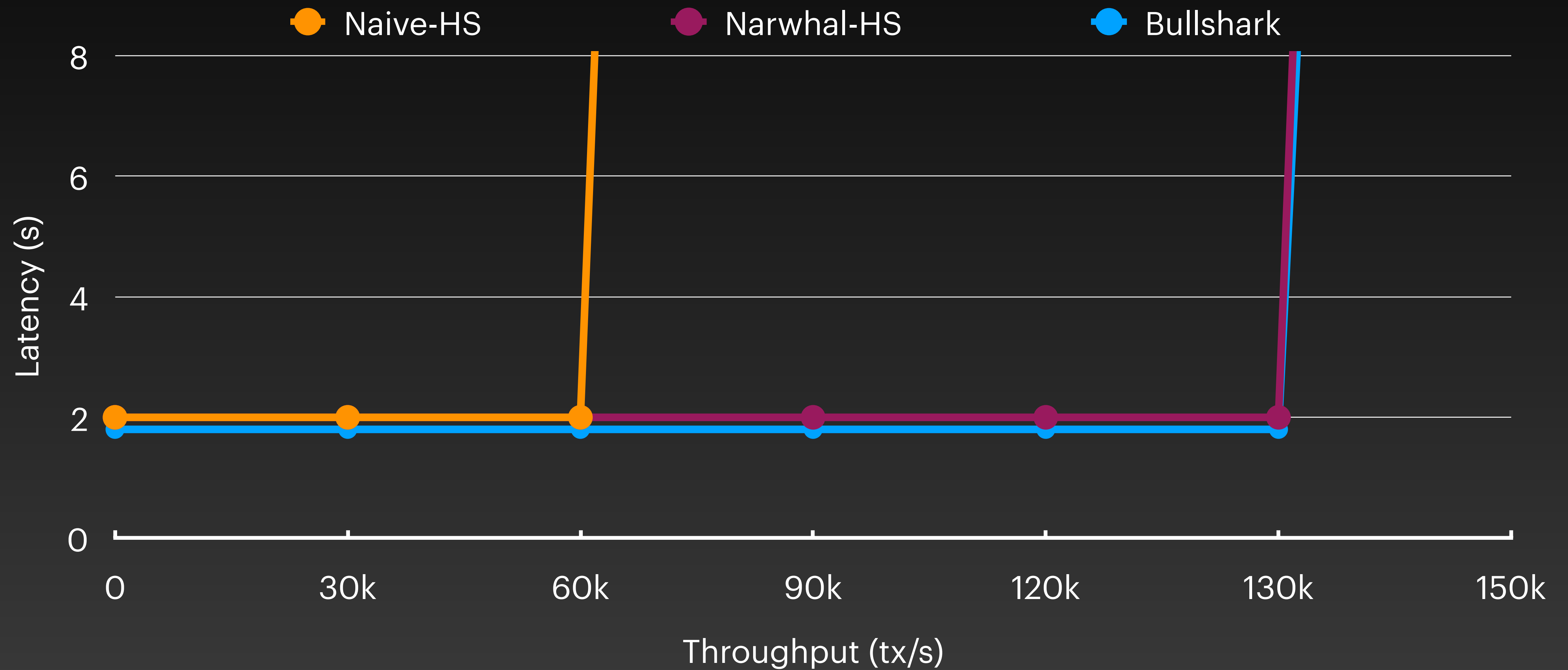
Research Questions

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

Performance



Research Questions

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

Lessons Learned

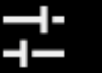
1. Modularisation is a design strategy
2. Tasks-threads allocation
3. Benchmark early
4. Codesign with mem. and storage
5. Core is hard, consensus is easy

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

...

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

Techniques

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

arXiv:2306.0308v2 [cs.DC] 7 Jul 2023

Shoal: Improving DAG-BFT Latency And Robustness

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

Abstract
The Narwhal system is a state-of-the-art Byzantine fault-tolerant scalable architecture that involves constructing a directed acyclic graph (DAG) of messages among a set of validators as a blockchain network. Bullshark is a zero-overhead consensus protocol on top of the Narwhal's DAG that can order over 100k transactions per second. Unfortunately, the high throughput of Bullshark comes with a latency price due to the DAG construction, increasing the latency compared to the state-of-the-art leader-based BFT consensus protocols.

We introduce Shoal, a protocol-agnostic framework for enhancing Narwhal-based consensus. By incorporating leader reputation and pipelining support for the first time, Shoal significantly reduces latency. Moreover, the combination of properties of the DAG construction and the leader reputation mechanism enables the elimination of timeouts in all but extremely uncommon scenarios in practice, a property we name "prevalent responsiveness" (it strictly subsumes the established and often desired "optimistic responsiveness" property for BFT protocols).

We integrated Shoal instantiated with Bullshark, the fastest existing Narwhal-based consensus protocol, in an open-source Blockchain project and provide experimental evaluations demonstrating up to 40% latency reduction in executions with failures against the vanilla Bullshark implementation.

CCS Concepts • Security and privacy → Distributed systems security.

Keywords: Consensus Protocol, Byzantine Fault Tolerance

ACM Reference Format:
Alexander Spiegelman, Balaji Arun, Rati Gelashvili, and Zekun Li. 2023. Shoal: Improving DAG-BFT Latency And Robustness.

1 Introduction
Byzantine fault tolerant (BFT) systems, including consensus protocols [13, 23, 24, 29] and state machine replication [7, 10, 26, 42, 46], have been a topic of research for over four decades as a means of constructing reliable distributed systems. Recently, the advent of Blockchains has underscored the significance of high performance. While Bitcoin handles approximately 10 transactions per second (TPS), the proof-of-stake committee-based blockchains [38–41, 43, 44] are now engaged in a race to deliver a scalable BFT system with the utmost throughput and minimal latency.

Historically, the prevailing belief has been that reducing communication complexity was the key to unlocking high performance, leading to the pursuit of protocols with linear communication. However, this did not result in drastic enough improvements in the throughput, falling significantly short of the current blockchain network targets. For example, the state-of-the-art HotStuff [46] protocol in this line of work only achieves a throughput of 3500 TPS [3].

A recent breakthrough, however, stemmed from the realization that data dissemination is the primary bottleneck for leader-based protocols, and it can benefit from parallelization [4, 17, 37, 45]. The Narwhal system [17] separated data dissemination from the core consensus logic and proposed an architecture where all validators simultaneously disseminate data, while the consensus component orders a smaller amount of metadata. A notable advantage of this architecture is that not only it delivers impressive throughput on a single machine, but also naturally supports scaling out each blockchain validator by adding more machines. The Narwhal paper [17] evaluated the system in a geo-replicated environment with 50 validators and reported a throughput of 160,000 TPS with one machine per validator, which further increased to 400,000 TPS with 10 machines per validator.

These numbers are more in line with the ambitions of modern blockchain systems. Consequently, Narwhal has garnered significant traction within the community, resulting in its deployment in Sui [44] and ongoing development in Aptos [39] and Celo [40].

Developing a production-ready reliable distributed system is challenging, and integrating intricate consensus protocols only adds to the difficulty. Narwhal addresses this issue by abstracting away networking from the consensus protocol. It constructs a non-equivocating round-based directed acyclic graph (DAG), a concept initially introduced by Aleph [21]. In this design, each validator contributes one vertex per round, and each vertex links to $n - f$ vertices in the preceding round. Each vertex is disseminated via an efficient reliable broadcast implementation, ensuring that malicious validators cannot distribute different vertices to different validators within the same round. With networking abstractions separated from the details of consensus, the DAG can be constructed without contending with complex mechanisms like view-change or view-synchronization.

During periods of network asynchrony, each validator may observe a slightly different portion of the DAG at any

Sailfish: Towards Improving the Latency of DAG-based BFT

Nibesh Shrestha
n.shrestha@supranetmex.com
Supra Research
Rohan Shrothrium
rohan@kurilabs.xyz
Kura Labs
Aniket Kate
aniket@purdue.com
Purdue University | Supra Research
Karrik Nayak
karrik@cs.duke.edu
Duke University

Abstract—Directed Acyclic Graph (DAG) based BFT protocols balance consensus efforts across different parties and maintain high throughput even when some designated parties fail. However, existing DAG-based BFT protocols exhibit long latency to commit decisions, primarily because they have a leader every 2 or more "rounds". Recent works, such as Shoal [C'23] and Mysticeti, have deemed supporting a leader vertex in each round particularly difficult, if not impossible. Consequently, even under honest leaders, these protocols require high latency (or communication complexity) to commit the proposal submitted by the leader (leader vertex) and additional latency to commit other proposals (non-leader vertices).

In this work, we present Sailfish, the first DAG-based BFT that supports a leader vertex in each round. Under honest leaders, Sailfish maintains a commit latency of one reliable broadcast (RBC) round plus 14 to commit the leader vertex (where 4 is the actual transmission latency of a message) and only an additional RBC round to commit non-leader vertices. We also extend Sailfish to Multi-leader Sailfish, which facilitates multiple leaders within a single round and commits all leader vertices in a round with a latency of one RBC round plus 14. Our experimental evaluation demonstrates that our protocols introduce significantly lower latency overhead compared to existing DAG-based protocols, with similar throughput.

1. Introduction

Byzantine fault-tolerant state machine replication (BFT/SMR) protocols form the core underpinning for blockchains. At a high level, a BFT-SMR enables a group of n parties to agree on a sequence of values, even if a bound of up to f of these parties is Byzantine (arbitrarily malicious). Owing to the need for efficient blockchains in practice, there has been a lot of recent progress in improving the key efficiency metrics namely, latency, communication complexity, and throughput under various network conditions. Assuming the network is partially synchronous, existing SMR protocols can commit with a latency overhead of 34 (where 4 represents the actual network delay) [13], [12], [22] and also achieve linear communication complexity [19], [31] under optimistic conditions (such as in honest leaders).

Most of these protocol designs rely on a designated leader who is the party responsible for proposing transactions and driving the protocol forward while other parties

agree on the proposed values and ensure that the leader keeps making progress. From an efficiency standpoint, this approach results in two key drawbacks. First, there is an uneven scheduling of work among the parties. While the leader is sending a proposal, the other parties' processors and their network are not used, leading to uneven resource usage across parties. Second, in typical leader-based protocols progress stops if the leader fails and until it is replaced. Several techniques proposed in the literature can potentially mitigate these concerns. These include the use of erasure coding techniques [9], [43] or the data availability committees [29], [27], [49] to disseminate the data more efficiently.

Recently, a novel approach known as DAG-based BFT has emerged [9], [19], [23], [33], [39], [49], [47]. These protocols enable all participating parties to propose in parallel, maximizing bandwidth utilization and ensuring equitable distribution of workload. Additionally, because each party is responsible for disseminating its own transactions, the protocol continues to progress in constructing the DAG even if a party fails during a round. Consequently, these protocols have demonstrated improved throughput compared to their leader-based counterparts under moderate network sizes [9], [49]. However, existing DAG-based protocols incur a high latency compared to their "leader-heavy" counterparts [12], [22], [33], [37], [61]. Is high latency inherent for such DAG-based protocols? Addressing this question is the key goal of this paper.

All existing DAG-based protocols progress in rounds. In each round, every party can create a potential DAG vertex containing transactions, with edges pointing to vertices from previous rounds. These protocols rely on committing a designated "leader vertex" and order other non-leader vertices in the DAG. Therefore, the frequency with which leaders are designated and how fast the leader vertices are committed directly influences the commit latency.

Supporting a leader vertex in each round. State-of-the-art protocols designate leaders once every two or more rounds, and in fact, deem supporting a leader vertex in each round particularly difficult. In their words, Shoal [49] writes, "Our attempt to solve the problem by delving into the inner workings of the protocol and exploring complex quorum intersection ordering rules have not been fruitful. Intuitively, this is because ...". Similarly, Mysticeti [9]

Cordial Miners: Fast and Efficient Consensus for Every Eventuality

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

Abstract
Cordial Miners are a family of efficient Byzantine Atomic Broadcast protocols, with instances for asynchrony and eventual synchrony. They improve the latency of state-of-the-art DAG-based protocols by almost $2\times$ and achieve optimal good-case complexity of $O(n)$ by forging Reliable Broadcast as a building block. Rather, Cordial Miners use the Modulo- n partially-ordered counterpart of the totally-ordered blockchain data structure—to implement the three algorithmic components of consensus: Dissemination, equivocation-exclusion, and ordering.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases Byzantine Fault Tolerance, State Machine Replication, DAG, Consensus, Blockchain, Blockade, Cordial Dissemination

Related Version Cordial Miners: Fast and Efficient Consensus for Every Eventuality
Full Version: <https://arxiv.org/abs/2206.09174>

Acknowledgements Oded Naor is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship, and to the Technion Hiroshi Fujiwara Cyber-Security Research Center for providing a research grant. Ehud Shapiro is the Inumbrent of The Harry Weinreb Professorial Chair of Computer Science and Biology at the Weizmann Institute.

1 Introduction

The problem of ordering transactions in a permissioned Byzantine distributed system, also known as *Byzantine Atomic Broadcast (BAB)*, has been investigated for four decades [30], and in the last decade, has attracted renewed attention due to the emergence of cryptocurrencies.

Recently, a line of works [4, 14, 20, 33, 21, 27] suggests ordering transactions using a distributed Directed Acyclic Graph (DAG) structure, in which each vertex contains a block of transactions as well as references to previously sent vertices. The DAG is distributively constructed from messages of *miners* running the consensus protocol. While building the DAG structure, each miner also totally orders the vertices in its DAG locally. That is, as the DAG is being constructed, a consensus on its ordering emerges without additional communication among the miners.

The two state-of-the-art protocols in this context are DAG-Rider [21] and Bullshark [33]. DAG-Rider works in the asynchronous setting, in which the adversary controls the finite delay on message delivery between miners, and Bullshark works in the Eventual Synchrony (ES) model, in which eventually all messages between correct miners are delivered within a known time-bound.

MYSTICETI: Reaching the Latency Limits with Uncertified DAGs

Kushal Babel^{1*}, Andrey Chursin¹, George Danezis^{1,3}, Anastasios Kichidis¹, Lefteris Kokoris-Kogias^{1*}, Arun Koshy¹, Alberto Sonnino^{1,3}, Mingwei Tian¹

¹Cornell Tech, ¹C3, ³Mysten Labs, ¹University College London (UCL), ¹IST Austria

Abstract—We introduce MYSTICETI-C, the first DAG-based Byzantine consensus protocol to achieve the lower bounds of latency of 3 message rounds. Since MYSTICETI-C is built over DAGs it also achieves high resource efficiency and censorship resistance. MYSTICETI-C achieves the latency improvement by avoiding explicit certification of the DAG blocks and by proposing a novel commit rule such that every block can be committed without delays, resulting in optimal latency in the steady state and under crash failures. We further extend MYSTICETI-C to MYSTICETI-EPC, which incorporates a fast commit path that achieves even lower latency for transferring assets. Unlike prior fast commit path protocols, MYSTICETI-EPC minimizes the number of signatures and messages by weaving the fast path transactions into the DAG. This free up resources, which subsequently result in better performance. We prove the safety and liveness in a Byzantine context. We evaluate both MYSTICETI protocols and compare them with state-of-the-art consensus and fast path protocols to demonstrate their low latency and resource efficiency, as well as their more graceful degradation under crash failures. MYSTICETI-C is the first Byzantine consensus protocol to achieve WAN latency of 0.5 for consensus commit while simultaneously maintaining state-of-the-art throughput of over 200k TPS. Finally, we report on integrating MYSTICETI-C as the consensus protocol into the Sui blockchain [67], resulting in over 4x latency reduction.

1. INTRODUCTION

Several recent blockchains, such as Sui [67], [12], have adopted consensus protocols based on certified directed acyclic graphs (DAG) of blocks [25], [55], [56], [34], [30], [70], [52], [58], [44]. By design, these consensus protocols scale well in terms of throughput, with a performance of 100k txs of new transactions and are robust against faults and network asynchrony [33], [25]. This, however, comes at a high latency of around 2-3 seconds, which can hinder user experience and prevent low-latency applications.

MYSTICETI-C: the power of uncertified DAGs Certified DAGs [34], [25], where each vertex is delivered through consistent broadcast [14], have high latency for three main reasons: (1) the certification process requires multiple round-trips to broadcast each block between validators, get signatures, and re-broadcast certificates. This leads to higher latency than traditional consensus protocols [11], [64], [15]; (2) blocks commit on a "per-wave" basis, which means that only once every two rounds (for Bullshark [53]) there is a chance to commit. Hence, some blocks have to wait for the wave to be decided. MYSTICETI-C solves all of the above challenges as (1) it is the first safe DAG-based consensus protocol that does not require explicit certificates, committing blocks within the



Fig. 1: P50 latency of a major blockchain switching from Bullshark (1900ms) to MYSTICETI-C (990ms) consensus on 106 independently run validators

be signed by a supermajority of validators, signature generation and verification consume a large amount of CPU on each validator, which grows with the number of validators [42], [16]. This burden is particularly heavy for a crash-recovered validator that typically needs to verify thousands of signatures when trying to catch up with the rest. Although at a first glance, certification seems to have the benefit that in adversarial cases nodes can advance the DAG without needing to synchronize the full-history, production experience of deploying Bullshark shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits only Byzantine Atomic Broadcast protocols but not if used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This comes in stark contrast to the early protocols for BFT consensus, such as PBFT [11], which requires only 3 message delays to commit a proposal (instead of the 6 in Bullshark) and facilitates the pipeline of proposals to commit one block every round [38]. They, however, require a high number of authenticated messages to coordinate, which consumes a lot of resources and results in low throughput. Additionally, they are fragile to faults and implementation mistakes due to their complexity, especially the view-change sub-protocols.

This work presents MYSTICETI, a family of DAG-based protocols allowing to safely commit distributed transactions in a Byzantine setting that focuses on low-latency and low-CPU operation, achieving the best of both worlds. MYSTICETI-C is a consensus protocol based on a threshold logical clock [29] DAG of blocks, that commits every block as early as it can be decided. MYSTICETI-C solves all of the above challenges as (1) it is the first safe DAG-based consensus protocol that does not require explicit certificates, committing blocks within the

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

Shoal: Improving DAG-BFT Latency And Robustness

Alexander Spiegelman
AptosRati Gelashvili
AptosBalaji Arun
AptosZekun Li
Aptos

Abstract

The Narwhal system is a state-of-the-art Byzantine fault-tolerant scalable architecture that involves constructing a directed acyclic graph (DAG) of messages among a set of validators in a blockchain network. Bullshark is a zero-overhead consensus protocol on top of the Narwhal’s DAG that can order over 100k transactions per second. Unfortunately, the high throughput of Bullshark comes with a latency price due to the DAG construction, increasing the latency compared to the state-of-the-art leader-based BFT consensus protocols.

We introduce Shoal, a protocol-agnostic framework for enhancing Narwhal-based consensus. By incorporating leader reputation and pipelining support for the first time, Shoal significantly reduces latency. Moreover, the combination of properties of the DAG construction and the leader reputation mechanism enables the elimination of timeouts in all but extremely uncommon scenarios in practice, a property we name “prevalent responsiveness” (it strictly subsumes the established and often desired “optimistic responsiveness” property for BFT protocols).

We integrated Shoal instantiated with Bullshark, the fastest existing Narwhal-based consensus protocol, in an open-source Blockchain project and provide experimental evaluations demonstrating up to 40% latency reduction in the failure-free executions, and up-to 80% reduction in executions with failures against the vanilla Bullshark implementation.

These numbers are more in line with the ambitions of modern blockchain systems. Consequently, Narwhal has generated significant traction within the community, resulting in its deployment in Sui [44] and ongoing development in Aptos [39] and Celo [40].

Developing a production-ready reliable distributed system is challenging, and integrating intricate consensus protocols only adds to the difficulty. Narwhal addresses this issue by abstracting away networking from the consensus protocol. It constructs a non-equivocating round-based directed acyclic graph (DAG), a concept initially introduced by Alperin [21]. In this design, each validator contributes one vertex per round, and each vertex links to $n - f$ vertices in the preceding round. Each vertex is disseminated via an efficient reliable broadcast implementation, ensuring that malicious validators cannot distribute different vertices to different validators within the same round. With networking abstraction separated from the details of consensus, the DAG can be constructed without contending with complex mechanisms like view-change or view synchronization.

During periods of network asynchrony, each validator may observe a slightly different portion of the DAG at any

Historically, the prevailing belief has been that reducing communication complexity was the key to unlocking high performance, leading to the pursuit of protocols with linear communication. However, this did not result in drastic enough improvements in the throughput, falling significantly short of the current blockchain network targets. For example, the state-of-the-art HotStuff [46] protocol in this line of work only achieves a throughput of 3500 TPS [3].

A recent breakthrough, however, stemmed from the realization that data dissemination is the primary bottleneck for leader-based protocols, and it can benefit from parallelization [4, 17, 37, 45]. The Narwhal system [17] separated data dissemination from the core consensus logic and proposed an architecture where all validators simultaneously disseminate data, while the consensus component orders a smaller amount of metadata. A notable advantage of this architecture is that not only it delivers impressive throughput on a single machine, but also naturally supports scaling on multiple machines, but also naturally supports scaling on each blockchain validator by adding more machines. The Narwhal paper [17] evaluated the system in a geo-replicated environment with 50 validators and reported a throughput of 160,000 TPS with one machine per validator, which further increased to 600,000 TPS with 10 machines per validator.

These numbers are more in line with the ambitions of modern blockchain systems. Consequently, Narwhal has generated significant traction within the community, resulting in its deployment in Sui [44] and ongoing development in Aptos [39] and Celo [40].

Developing a production-ready reliable distributed system is challenging, and integrating intricate consensus protocols only adds to the difficulty. Narwhal addresses this issue by abstracting away networking from the consensus protocol. It constructs a non-equivocating round-based directed acyclic graph (DAG), a concept initially introduced by Alperin [21]. In this design, each validator contributes one vertex per round, and each vertex links to $n - f$ vertices in the preceding round. Each vertex is disseminated via an efficient reliable broadcast implementation, ensuring that malicious validators cannot distribute different vertices to different validators within the same round. With networking abstraction separated from the details of consensus, the DAG can be constructed without contending with complex mechanisms like view-change or view synchronization.

During periods of network asynchrony, each validator may observe a slightly different portion of the DAG at any

Sailfish: Towards Improving the Latency of DAG-based BFT

Nihesh Shrestha
n.shrestha@supernodes.com
Supra ResearchRohan Shrothrium
rohan@luridale.xyz
Rura LabsAniket Kate
aniket@pauldub.com
Pauldub University / Supra ResearchKarik Nayak
karik@cs.duke.edu
Duke University

Abstract—Directed Acyclic Graph (DAG) based BFT protocols balance consensus efforts across different parties and maintain high throughput even when some designated parties fail. However, existing DAG-based BFT protocols exhibit long latency to commit decisions, primarily because they have a leader every 2 or more “rounds”. Recent works, such as Shoal [37, 45] and Mysticeti, have deemed supporting a leader vertex in each round particularly difficult, if not impossible. Consequently, even under honest leaders, these protocols require high latency (or communication complexity) to commit the proposal submitted by the leader (leader vertex) and additional latency to commit other proposals (non-leader vertices).

In this work, we present Sailfish, the first DAG-based BFT that supports a leader vertex in each round. Under honest leaders, Sailfish maintains a commit latency of one reliable broadcast (RBC) round plus 14 to commit the leader vertex where 4 is the actual transmission latency of a message and only an additional RBC round to commit non-leader vertices. We also extend Sailfish to Multi-leader Sailfish, which facilitates multiple leaders within a single round and commits all leader vertices in a round with a latency of one RBC round plus 14. Our experimental evaluation demonstrates that our protocol introduces significantly lower latency overhead compared to existing DAG-based protocols, with similar throughput.

1. Introduction

Byzantine fault-tolerant state machine replication (BFT/SMR) protocols form the core underpinning for blockchains. At a high level, a BFT-SMR enables a group of n parties to agree on a sequence of values, even if a bound of up to f of these parties is Byzantine (arbitrarily malicious). Owing to the need for efficient blockchains in practice, there has been a lot of recent progress in improving the key efficiency metrics namely, latency, communication complexity, and throughput under various network conditions. Assuming the network is partially synchronous, existing SMR protocols can commit with a latency overhead of 3 (where 4 represents the actual network delay) [13], [12], [22] and also achieve linear communication complexity [19], [31] under optimistic conditions (such as an honest leader).

Most of these protocol designs rely on a designated leader who is the party responsible for proposing transactions and driving the protocol forward while other parties

agree on the proposed values and ensure that the leader keeps making progress. From an efficiency standpoint, this approach results in two key drawbacks. First, there is an uneven scheduling of work among the parties. While the leader is sending a proposal, the other parties’ processors and their network are not used, leading to uneven resource usage across parties. Second, in typical leader-based protocols progress stops if the leader fails and until it is replaced. Several techniques proposed in the literature can potentially mitigate these concerns. These include the use of erasure coding techniques [25], [41] or the data availability committees [29], [27], [49] to disseminate the data more efficiently.

Recently, a novel approach known as DAG-based BFT has emerged [9], [19], [23], [13], [29], [49], [27]. These protocols enable all participating parties to prepare in parallel, maximizing bandwidth utilization and ensuring equitable distribution of workload. Additionally, because each party is responsible for disseminating its own transactions, the protocol continues to progress in constructing the DAG even if a party fails during a round. Consequently, these protocols have demonstrated improved throughput compared to their leader-based counterparts under moderate network sizes [9], [49]. However, existing DAG-based protocols incur a high latency compared to their “leader-heavy” counterparts [12], [23], [29], [13], [61]. Is high latency inherent for such DAG-based protocols? Addressing this question is the key goal of this paper.

All existing DAG-based protocols progress in rounds. In each round, every party can create a potential DAG vertex containing transactions, with edges pointing to vertices from previous rounds. These protocols rely on committing a designated “leader vertex” and order other non-leader vertices in the DAG. Therefore, the frequency with which leaders are designated and how fast the leader vertices are committed directly influences the commit latency.

Supporting a leader vertex in each round. State-of-the-art protocols designate leaders once every two or more rounds, and in fact, deem supporting a leader vertex in each round particularly difficult. In their words, Shoal [23] writes, “Our attempt to solve the problem by delving into the inner workings of the protocol and exploring complex quorum intersection ordering rules have not been fruitful. Intuitively, this is because ...”. Similarly, Mysticeti [9]

Cordial Miners: Fast and Efficient Consensus for Every Eventuality

Idit Keidar

Technion

Oded Naor

Technion and StartWare

Ouri Poupko

Ben-Gurion University

Ehud Shapiro

Weizmann Institute of Science

Abstract

Cordial Miners are a family of efficient Byzantine Atomic Broadcast protocols, with instances for asynchrony and eventual synchrony. They improve the latency of state-of-the-art DAG-based protocols by almost $2\times$ and achieve optimal good-case complexity of $O(n)$ by forging Bullshark Broadcast as a building block. Rather, Cordial Miners use the blockchain—a partially-ordered counterpart of the totally-ordered blockchain data structure—to implement the three algorithmic components of consensus: Dissemination, equivocation-exclusion, and ordering.

2012 ACM Subject Classification Computing methodologies → Distributed algorithms

Keywords and phrases: Byzantine Fault Tolerance, State Machine Replication, DAG, Consensus, Blockchain, Blockless, Cordial Dissemination

Related Version: Cordial Miners: Fast and Efficient Consensus for Every Eventuality
Full Version: <https://arxiv.org/abs/2205.09174>

Acknowledgments: Oded Naor is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship, and to the Technion Ilirakli Fugwara Cyber-Security Research Center for providing a research grant. Ehud Shapiro is the Incumbent of The Harry Weinstock Professorial Chair of Computer Science and Biology at the Weizmann Institute.

1 Introduction

The problem of ordering transactions in a permissioned Byzantine distributed system, also known as *Byzantine Atomic Broadcast (BAB)*, has been investigated for four decades [30], and in the last decade, has attracted renewed attention due to the emergence of cryptocurrencies.

Recently, a line of works [4, 14, 20, 33, 21, 27] suggests ordering transactions using a distributed Directed Acyclic Graph (DAG) structure, in which each vertex contains a block of transactions as well as references to previously sent vertices. The DAG is distributively constructed from messages of *miners* running the consensus protocol. While building the DAG structure, each miner also totally orders the vertices in its DAG locally. That is, as the DAG is being constructed, a consensus on its ordering emerges without additional communication among the miners.

The two state-of-the-art protocols in this context are DAG-Rider [21] and Bullshark [33]. DAG-Rider works in the asynchronous setting, in which the adversary controls the finite delay on message delivery between miners, and Bullshark works in the Eventual Synchrony (ES) model, in which eventually all messages between correct miners are delivered within a known time-bound.

Mysticeti

MYSTICETI: Reaching the Latency Limits with Uncertified DAGs

Kushal Babel^{1*}, Andrey Chursin¹, George Danezis^{1§}, Anastasios Kichidis¹, Lefteris Kokoris-Kogias^{1*}, Arun Koshy¹, Alberto Sonnino^{1§}, Mingwei Tian¹¹Cornell Tech, ¹UC3, ¹Mysten Labs, ¹University College London (UCL), ¹IST Austria

Abstract—We introduce MYSTICETI-C, the first DAG-based Byzantine consensus protocol to achieve the lower bounds of latency of 3 message rounds. Since MYSTICETI-C is built over DAGs it also achieves high resource efficiency and censorship resistance. MYSTICETI-C achieves the latency improvement by avoiding explicit certification of the DAG blocks and by proposing a novel commit rule such that every block can be committed without delays, resulting in optimal latency in the steady state and under crash failures. We further extend MYSTICETI-C to MYSTICETI-EPC, which incorporates a fast commit path that achieves even lower latency for transferring assets. Unlike prior fast commit path protocols, MYSTICETI-EPC minimizes the number of signatures and messages by weaving the fast path transactions into the DAG. This frees up resources, which subsequently result in better performance. We prove the safety and liveness in a Byzantine context. We evaluate both MYSTICETI protocols and compare them with state-of-the-art consensus and fast path protocols to demonstrate their low latency and resource efficiency, as well as their more graceful degradation under crash failures. MYSTICETI-C is the first Byzantine consensus protocol to achieve WAN latency of 0.5s for consensus commit while simultaneously maintaining state-of-the-art throughput of over 200k TPS. Finally, we report on integrating MYSTICETI-C as the consensus protocol into the Sui blockchain [67], resulting in over 4x latency reduction.

I. INTRODUCTION

Several recent blockchains, such as Sui [67], [12], have adopted consensus protocols based on certified directed acyclic graphs (DAG) of blocks [25], [55], [56], [34], [30], [70], [52], [58], [44]. By design, these consensus protocols scale well in terms of throughput, with a performance of 100k txs of new transactions and are robust against faults and network asynchrony [33], [25]. This, however, comes at a high latency of around 2-3 seconds, which can hinder user experience and prevent low-latency applications.

MYSTICETI-C: the power of uncertified DAGs Certified DAGs [34], [25], where each vertex is delivered through consistent broadcast [14], have high latency for three main reasons: (1) the certification process requires multiple round-trips to broadcast each block between validators, get signatures, and re-broadcast certificates. This leads to higher latency than traditional consensus protocols [11], [64], [15]; (2) blocks commit on a “per-wave” basis, which means that only once every two rounds (for Bullshark [55]) there is a chance to commit. Hence, some blocks have to wait for the wave to finish increasing the latency of transactions proposed by the block. This phenomenon is similar to committing big batches of $2^l + 1$ blocks. Finally, (3) since all certified blocks need to



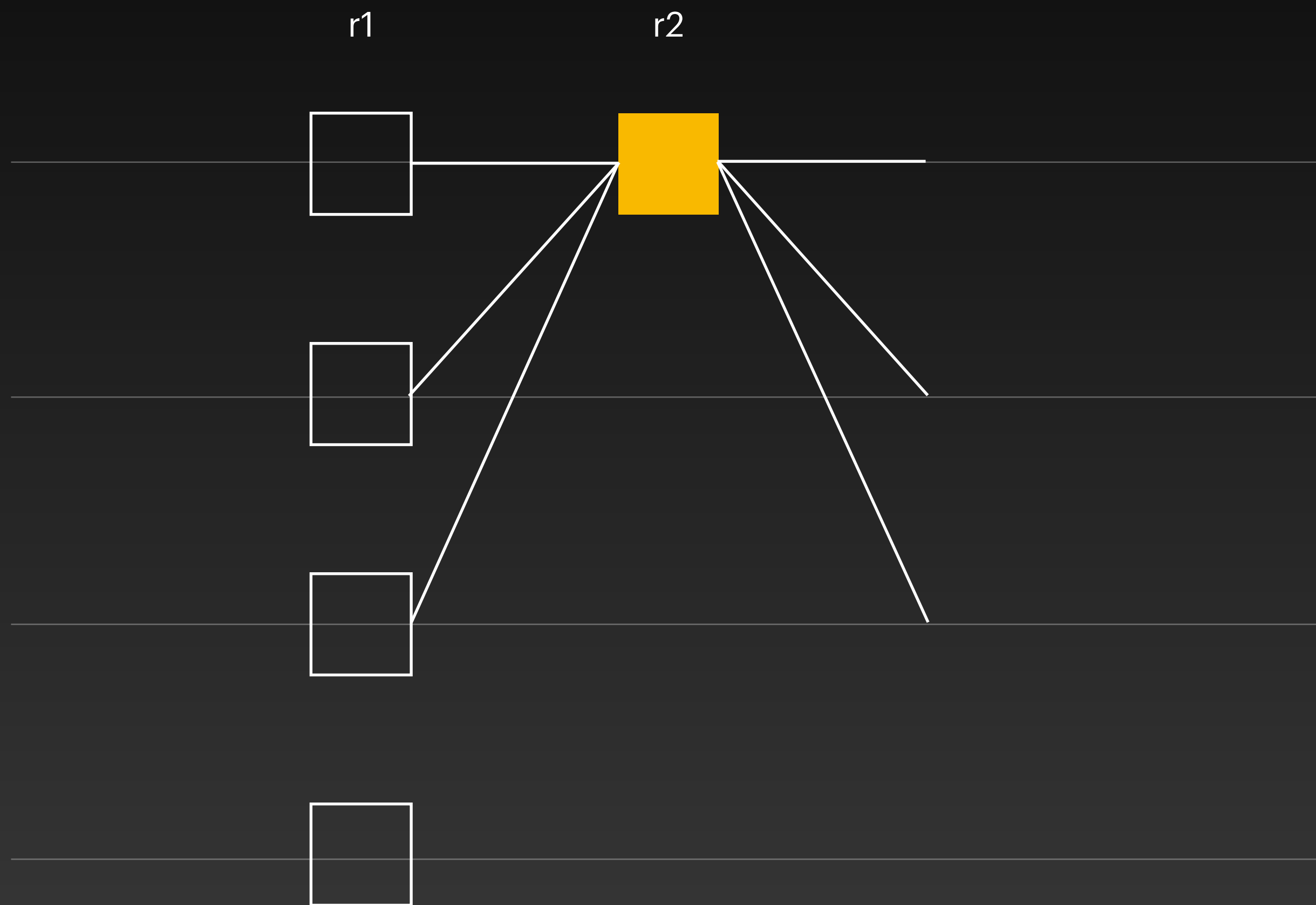
Fig. 1: P50 latency of a major blockchain switching from Bullshark (1900ms) to MYSTICETI-C (99ms) consensus on 106 independently run validators

be signed by a supermajority of validators, signature generation and verification consume a large amount of CPU on each validator, which grows with the number of validators [42], [16]. This burden is particularly heavy for a crash-recovered validator that typically needs to verify thousands of signatures when trying to catch up with the rest. Although at a first glance, certification seems to have the benefit that in adversarial cases nodes can advance the DAG without needing to synchronize the full-history, production experience of deploying Bullshark shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits only Byzantine Atomic Broadcast protocols but not if used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This comes in stark contrast to the early protocols for BFT consensus, such as PBFT [11], which requires only 3 message delays to commit a proposal (instead of the 6 in Bullshark) and facilitates the pipeline of proposals to commit one block every round [38]. They, however, require a high number of authenticated messages to coordinate, which consumes a lot of resources and results in low throughput. Additionally, they are fragile to faults and implementation mistakes due to their complexity, especially the view-change sub-protocols.

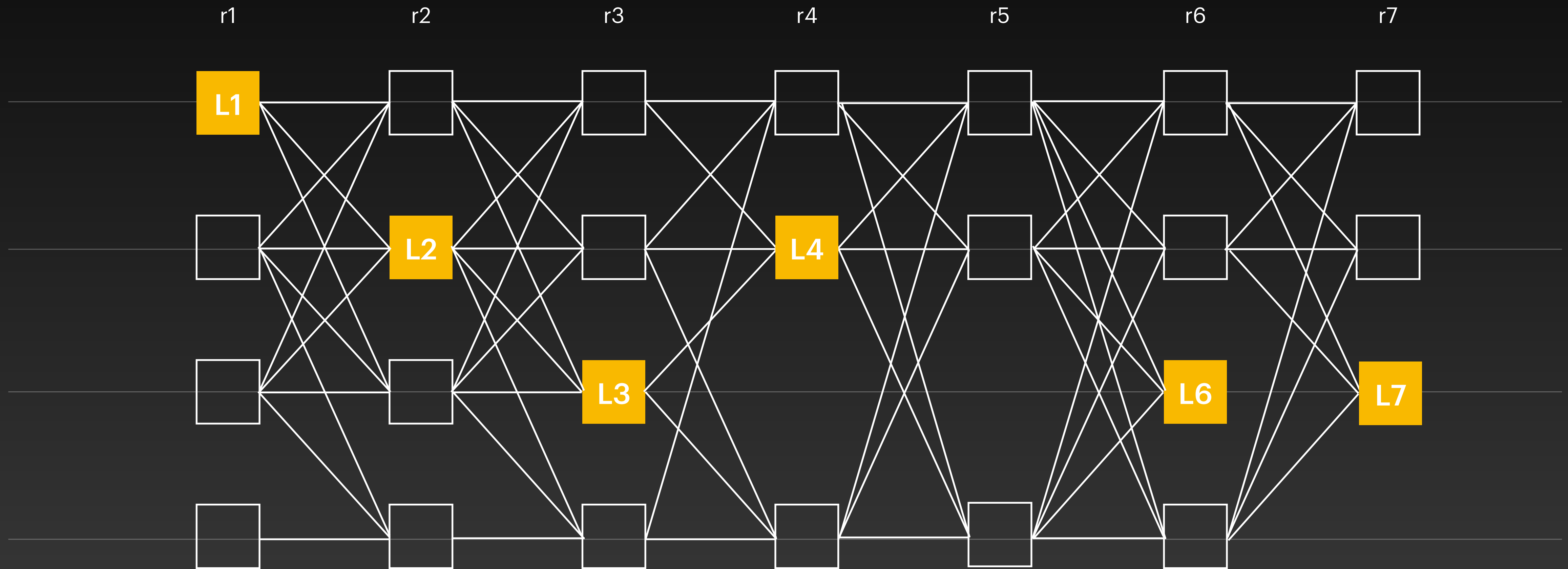
This work presents MYSTICETI, a family of DAG-based protocols allowing to safely commit distributed transactions in a Byzantine setting that focuses on low-latency and low-CPU operation, achieving the best of both worlds. MYSTICETI-C is a consensus protocol based on a threshold logical clock [29] DAG of blocks, that commits every block as early as it can be decided. MYSTICETI-C solves all of the above challenges as (1) it is the first safe DAG-based consensus protocol that does not require explicit certificates, committing blocks within the

Uncertified DAG

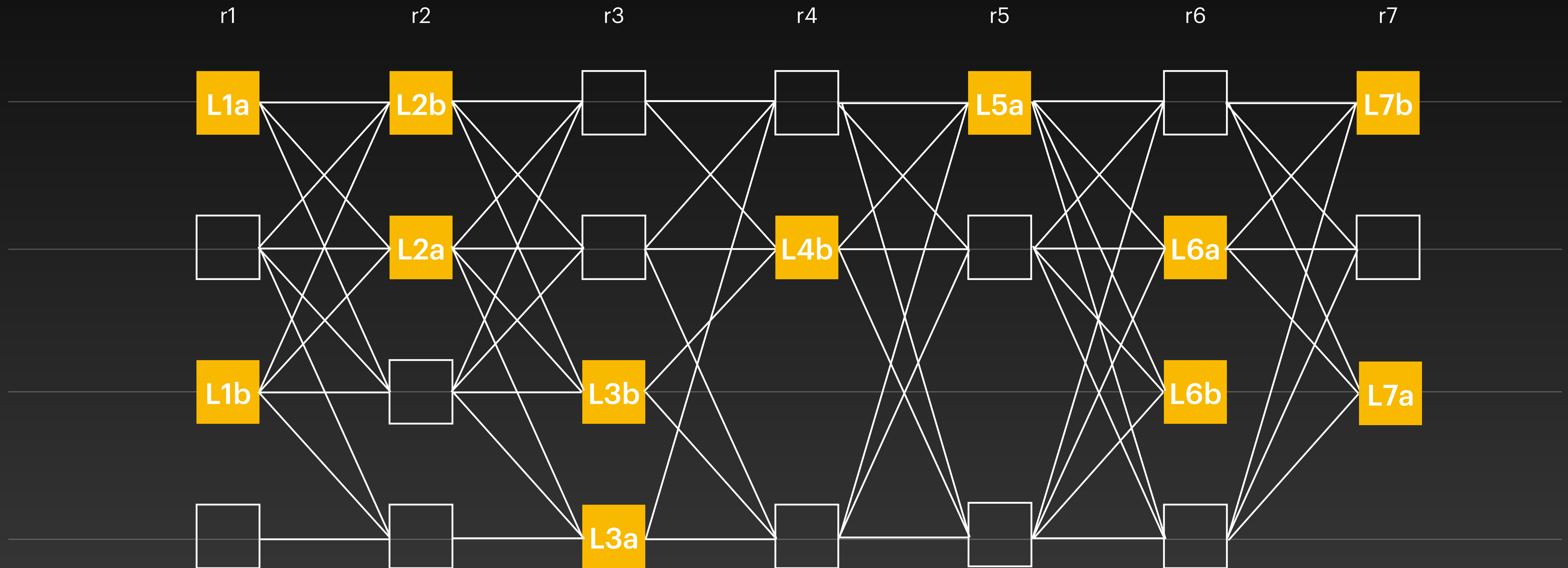


- Round number
- Author
- Payload (transactions)
- Signature

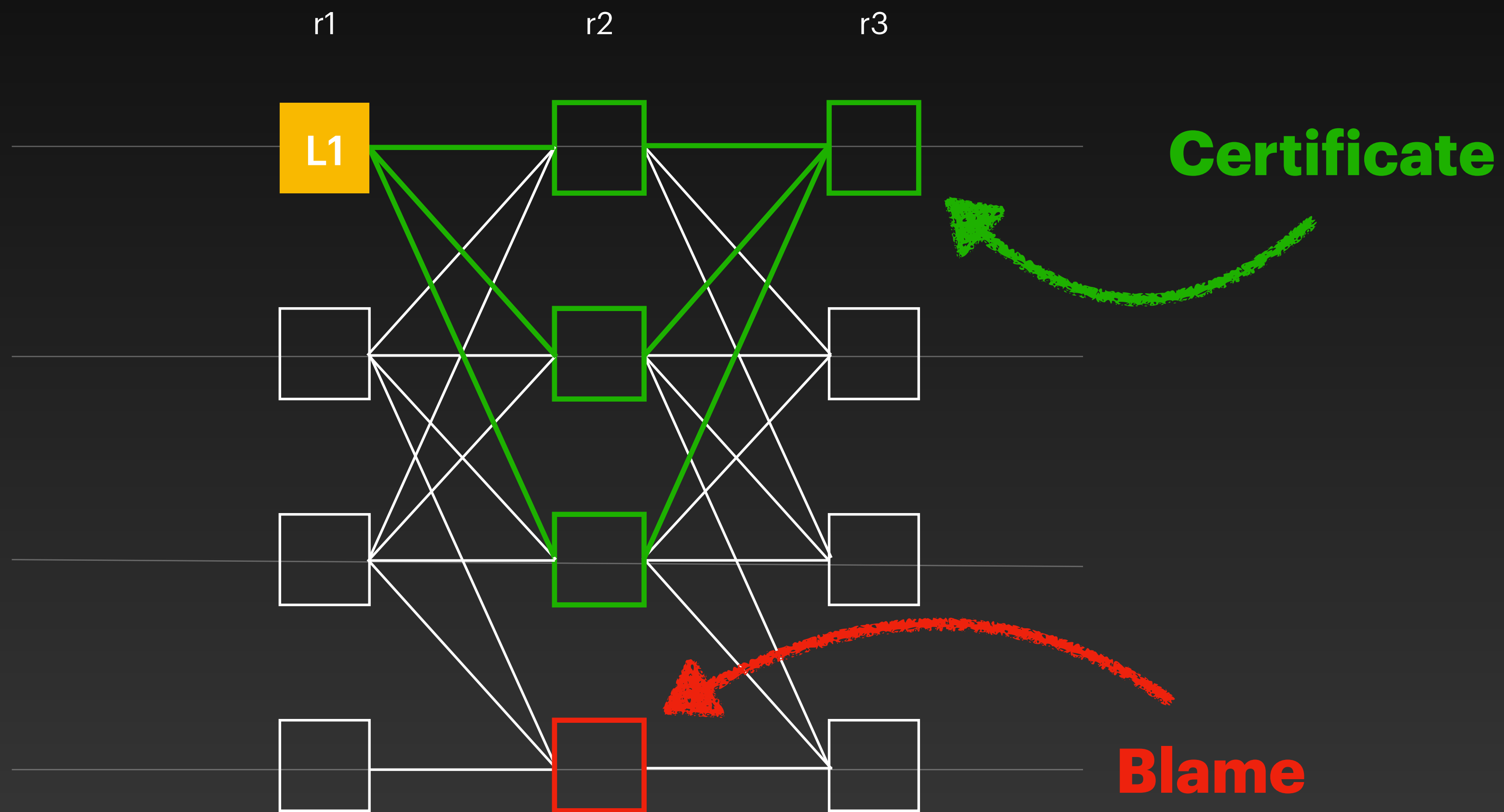
Uncertified DAG



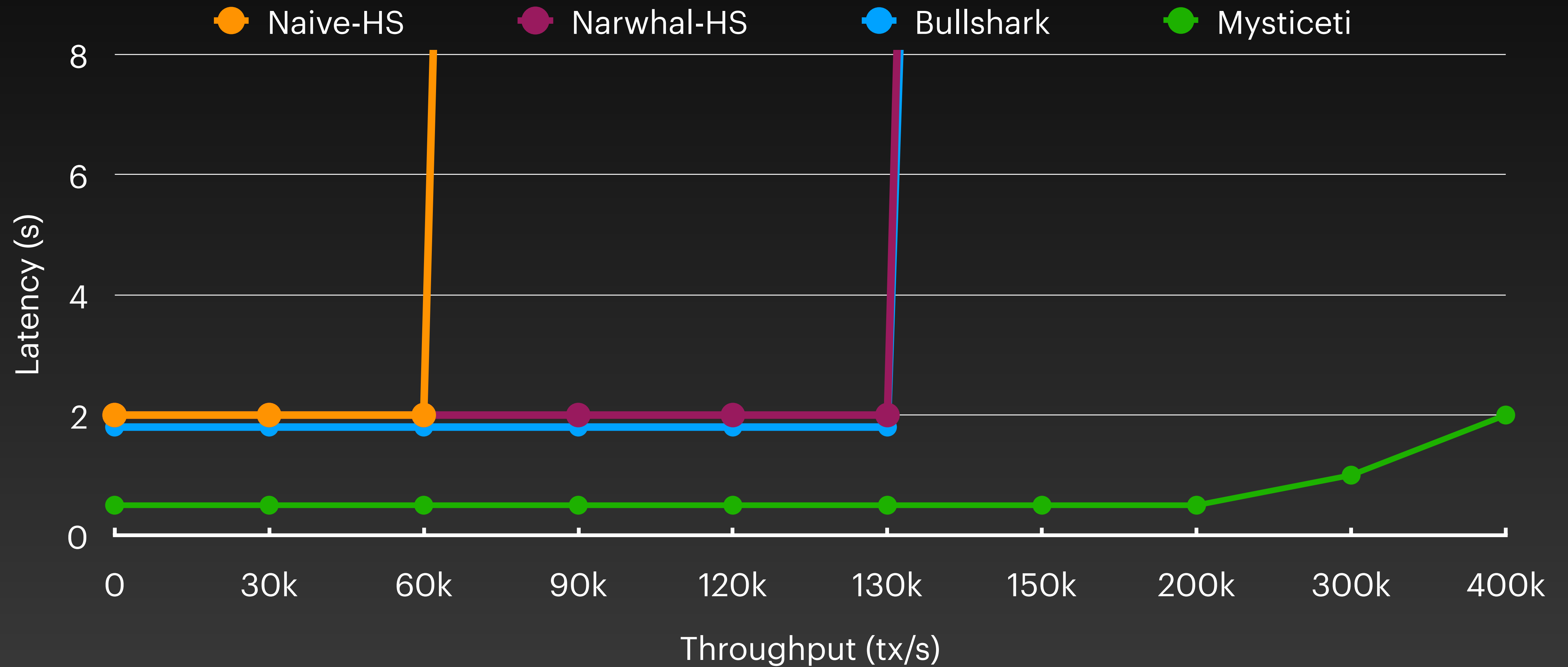
Uncertified DAG



Interpreting DAG Patterns



Performance



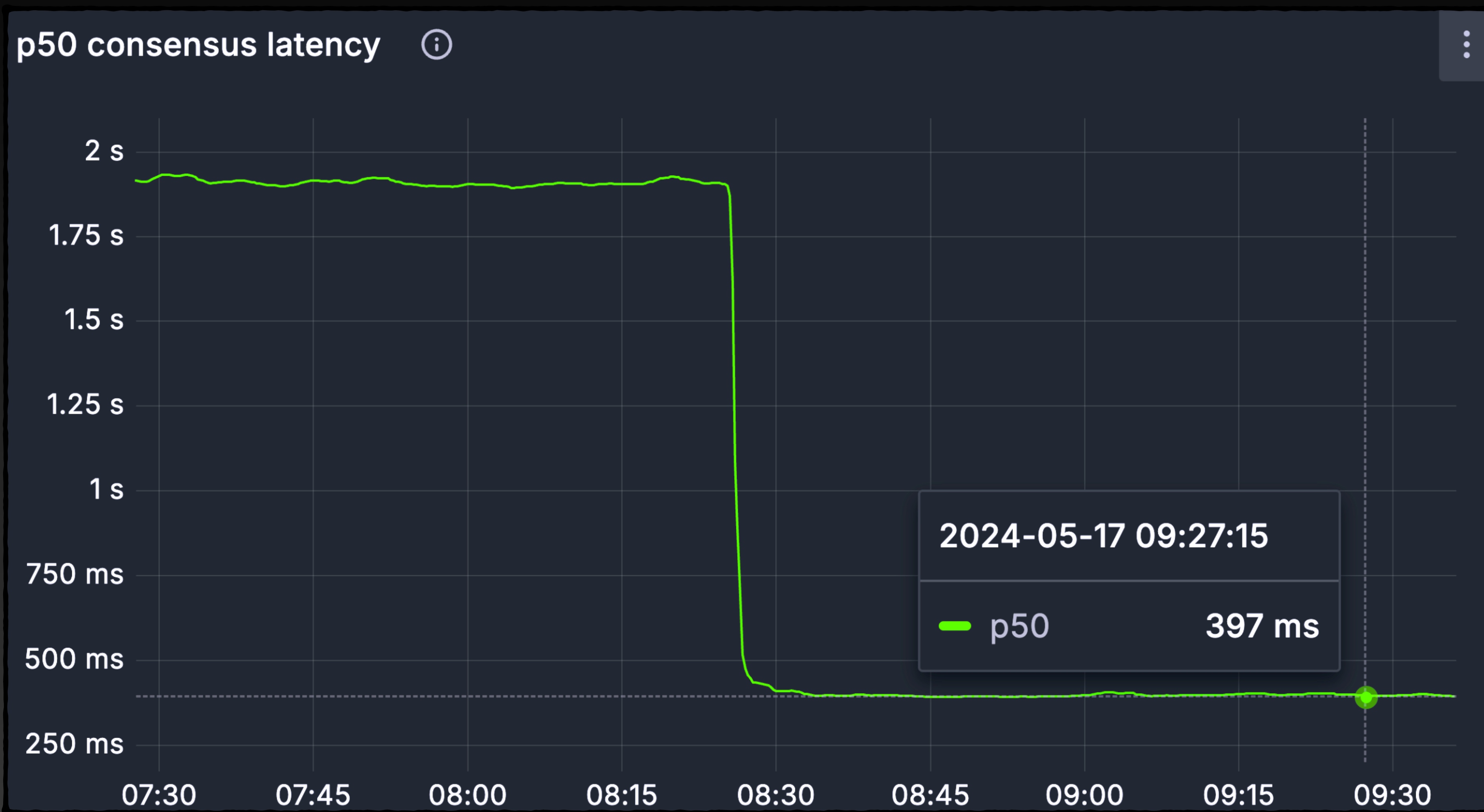
Research Questions

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

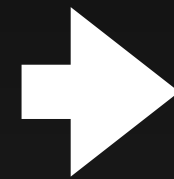
The Sui Mainnet



The Roadmap

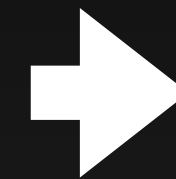
2019

naive consensus



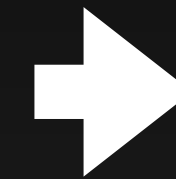
2020-2021

mempool ❤️ consensus



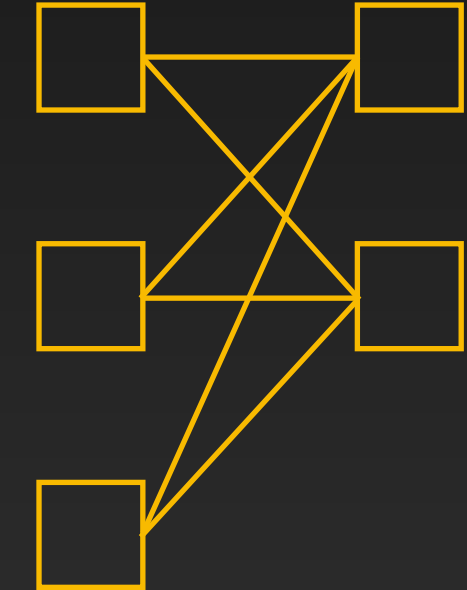
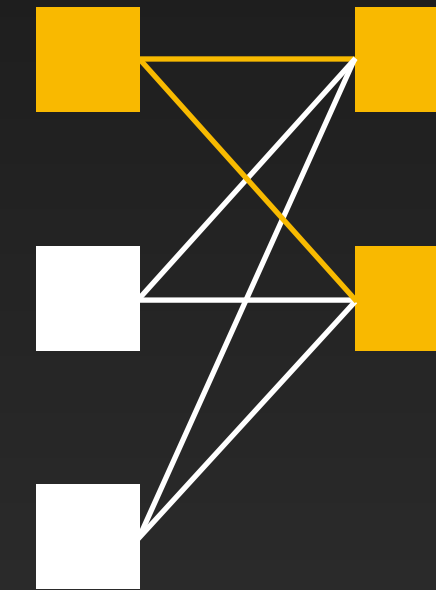
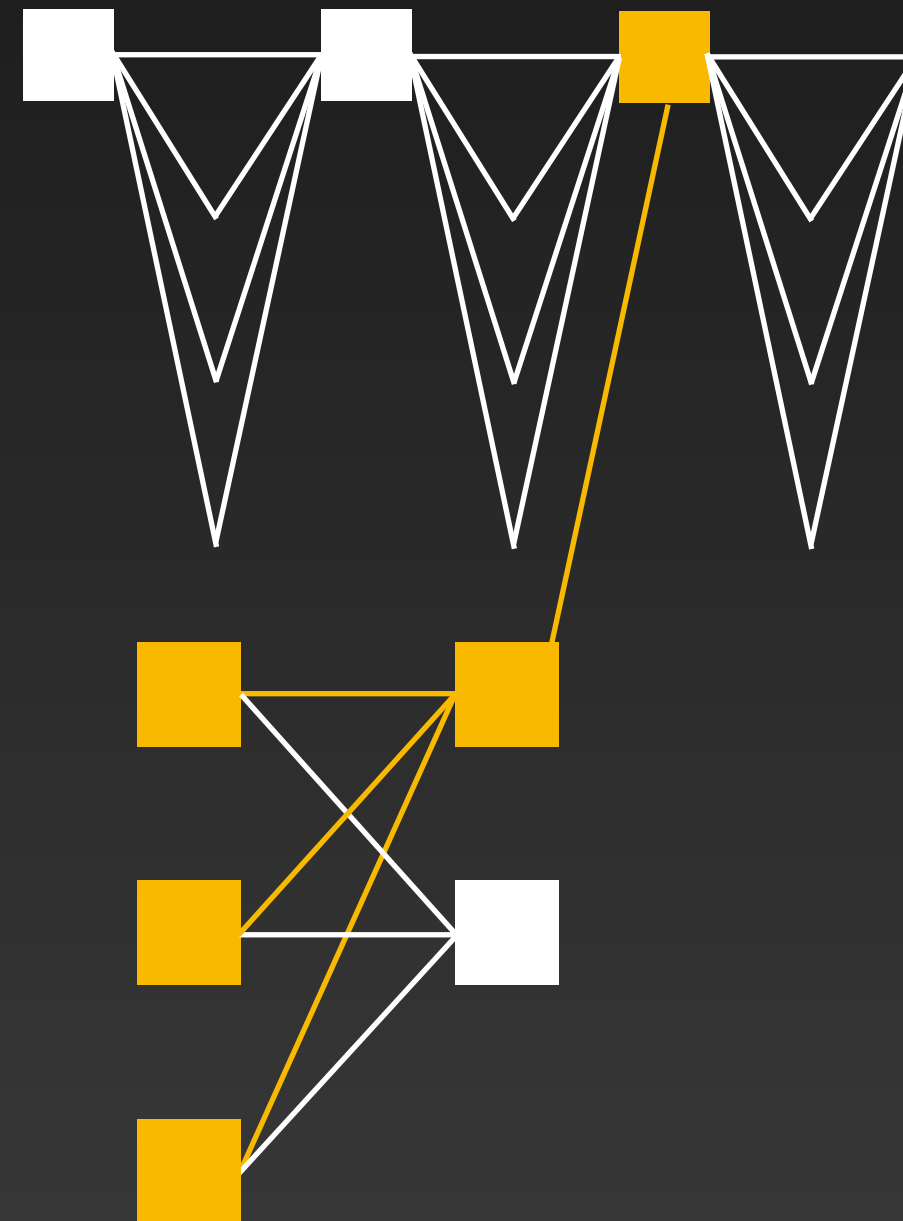
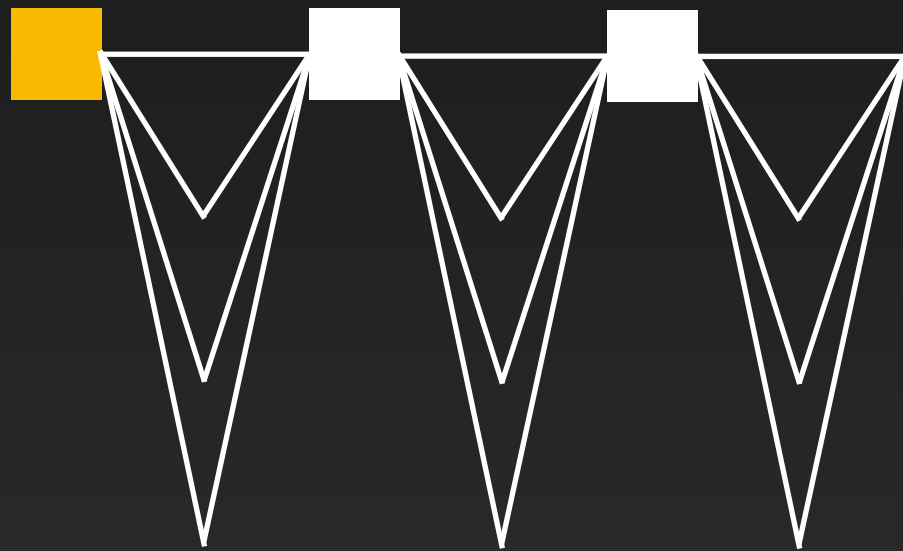
2022-2023

Fold into DAG



2024

Remove overhead



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



Dmitri Perelman Oct 18th at 5

In tomorrow's Research <> C
going to share top of mind of
struggles. See you 🙌



2



Thread

sui-core-internal



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



2 replies



John Martin Oct 18th at 6:16 AM

Can I get an invite to this 🙏



2



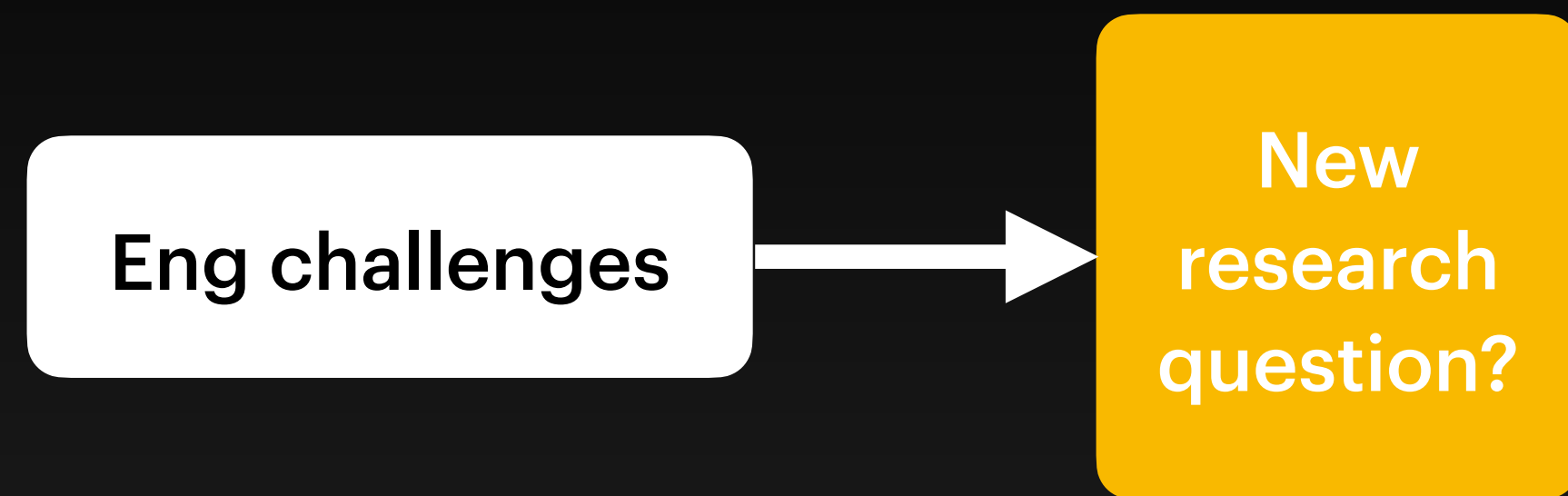
Dmitri Perelman Oct 18th at 7:36 AM

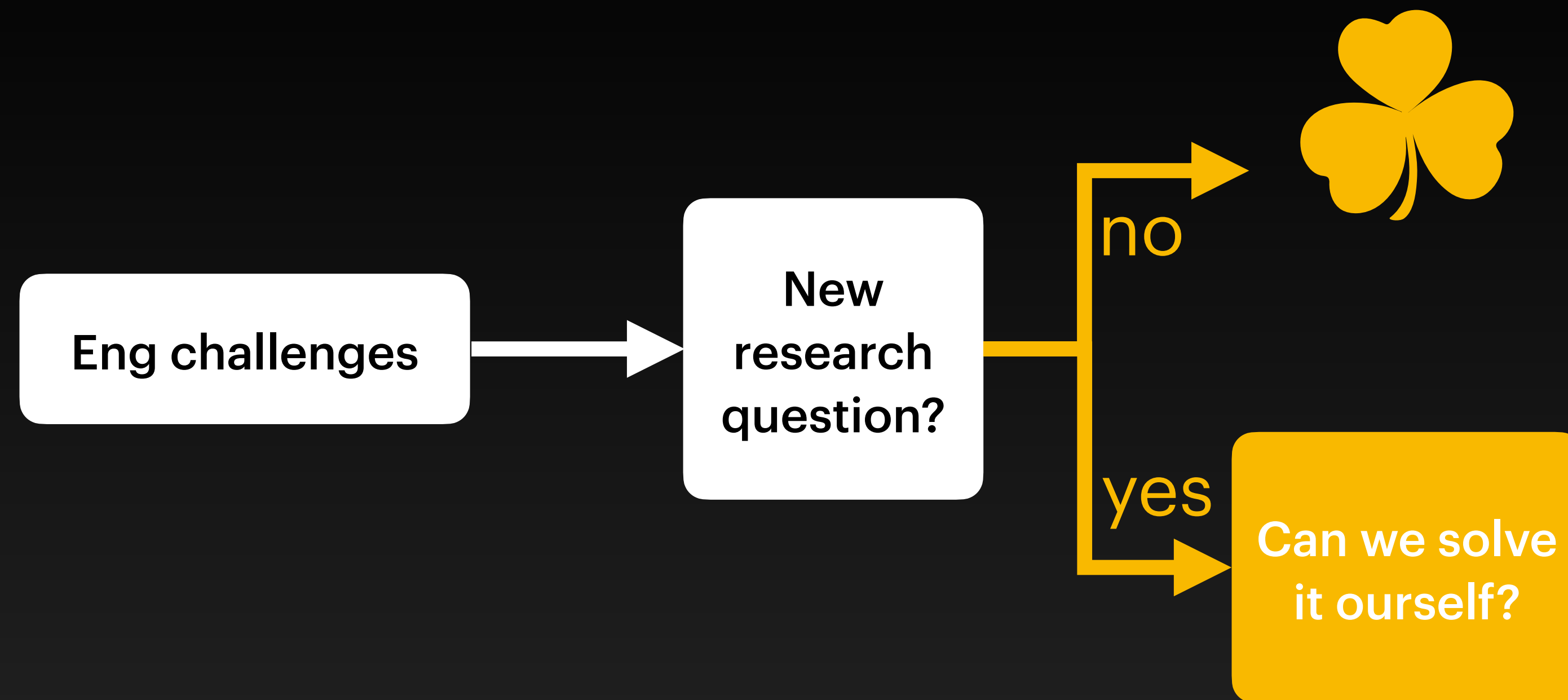
You're in the invite list!

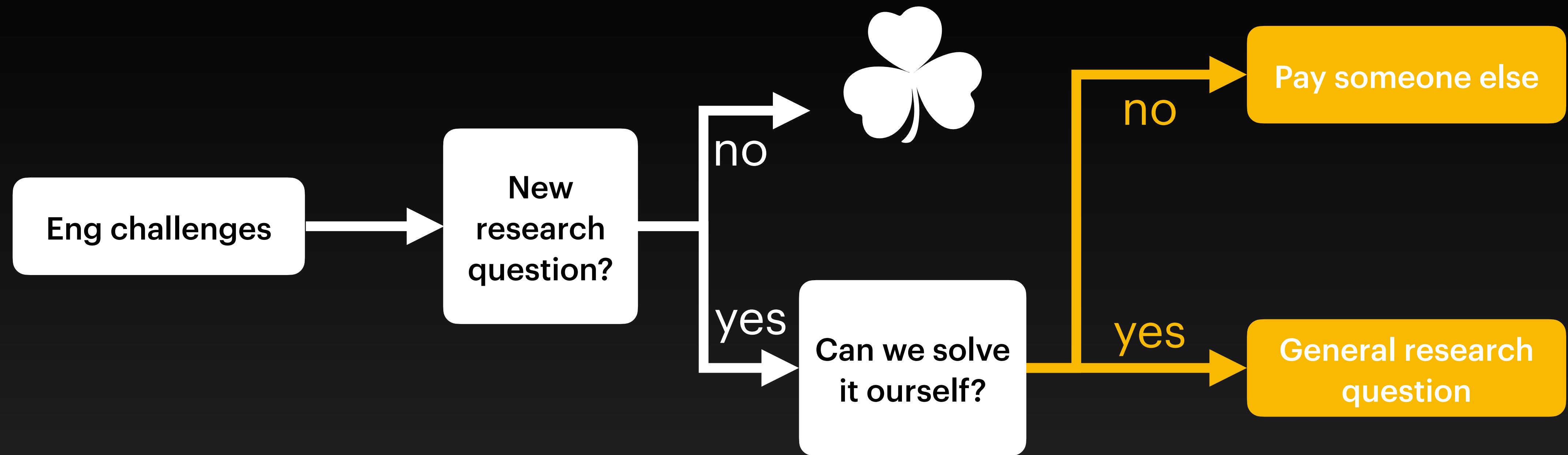


1





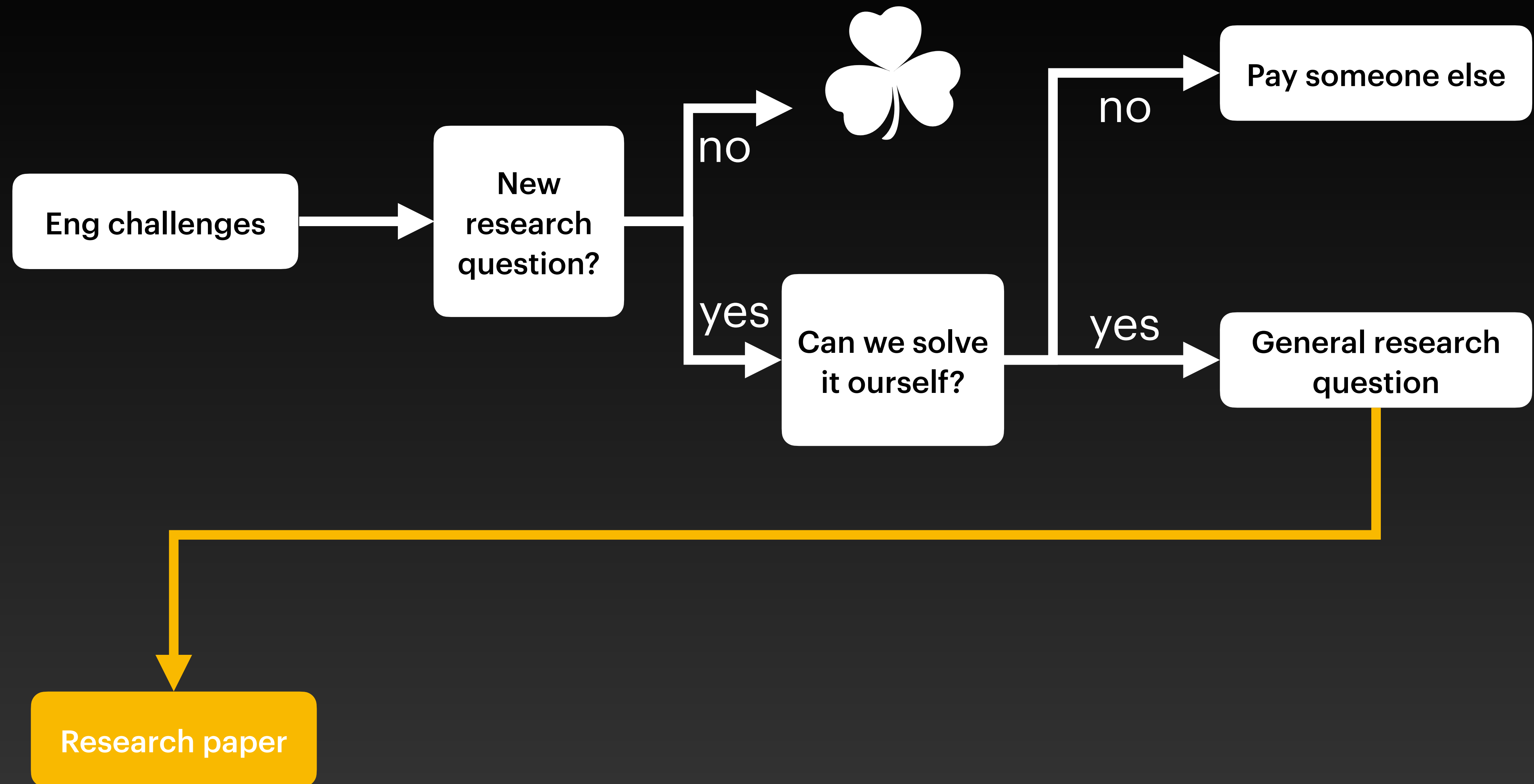


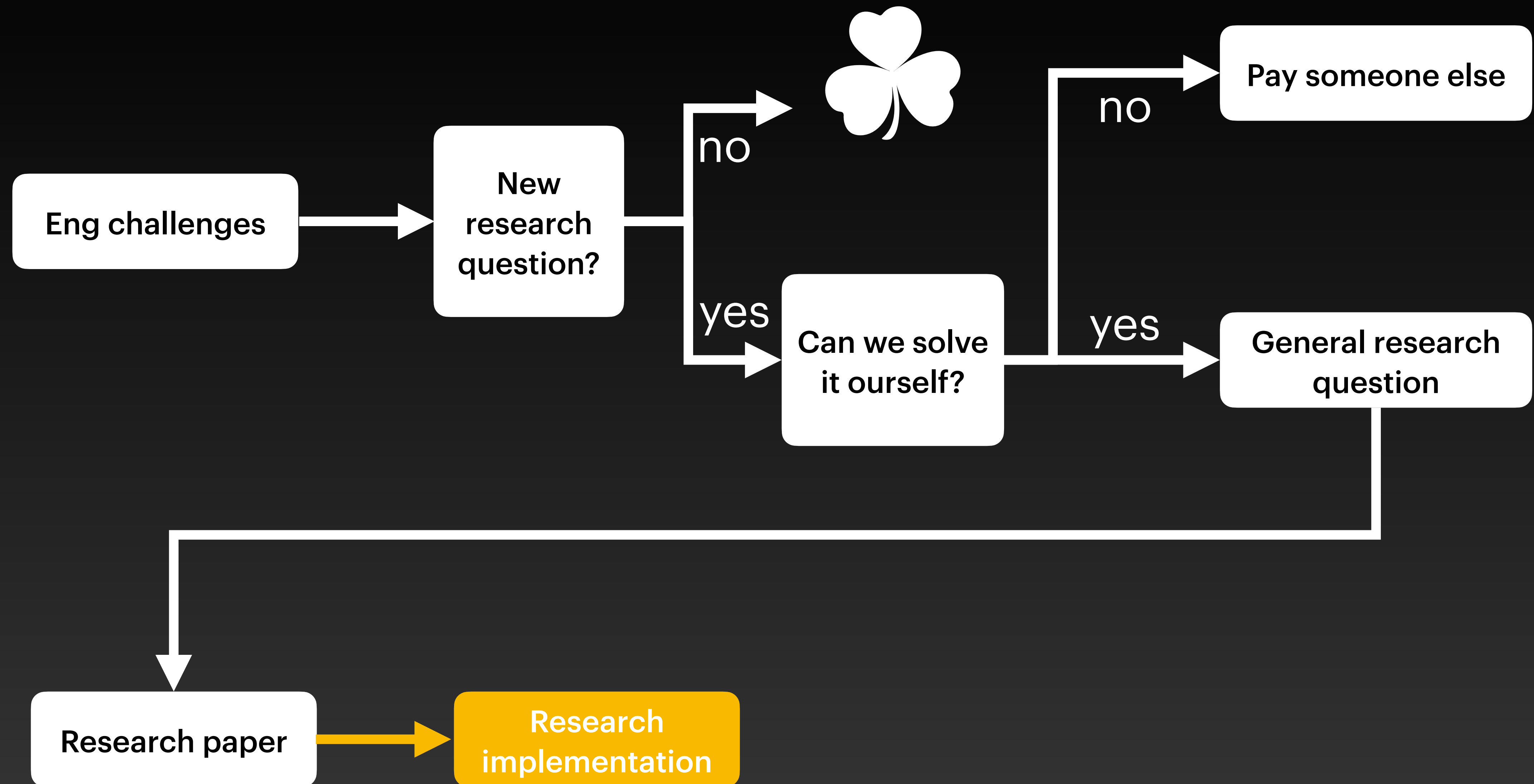


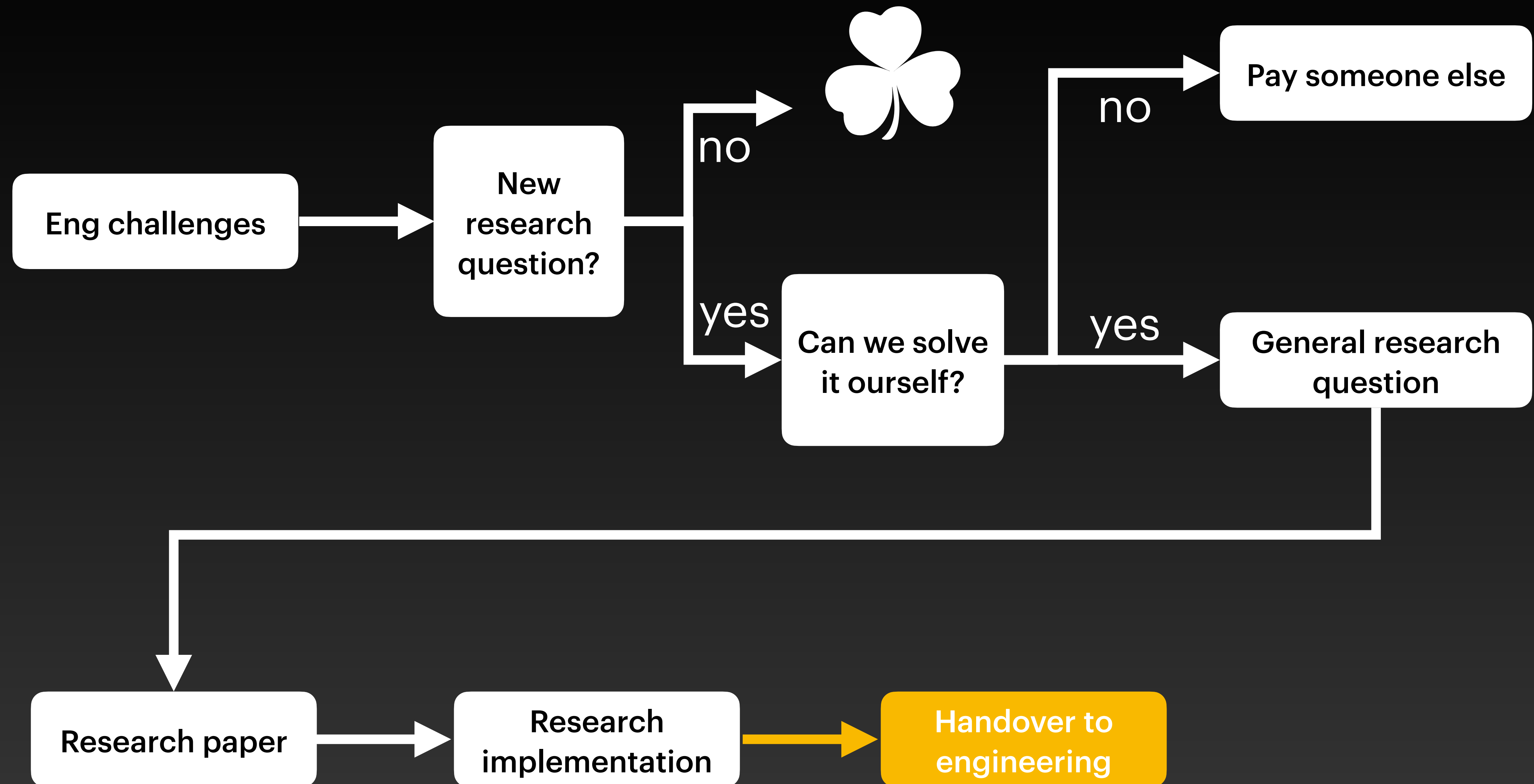
Research Gifts

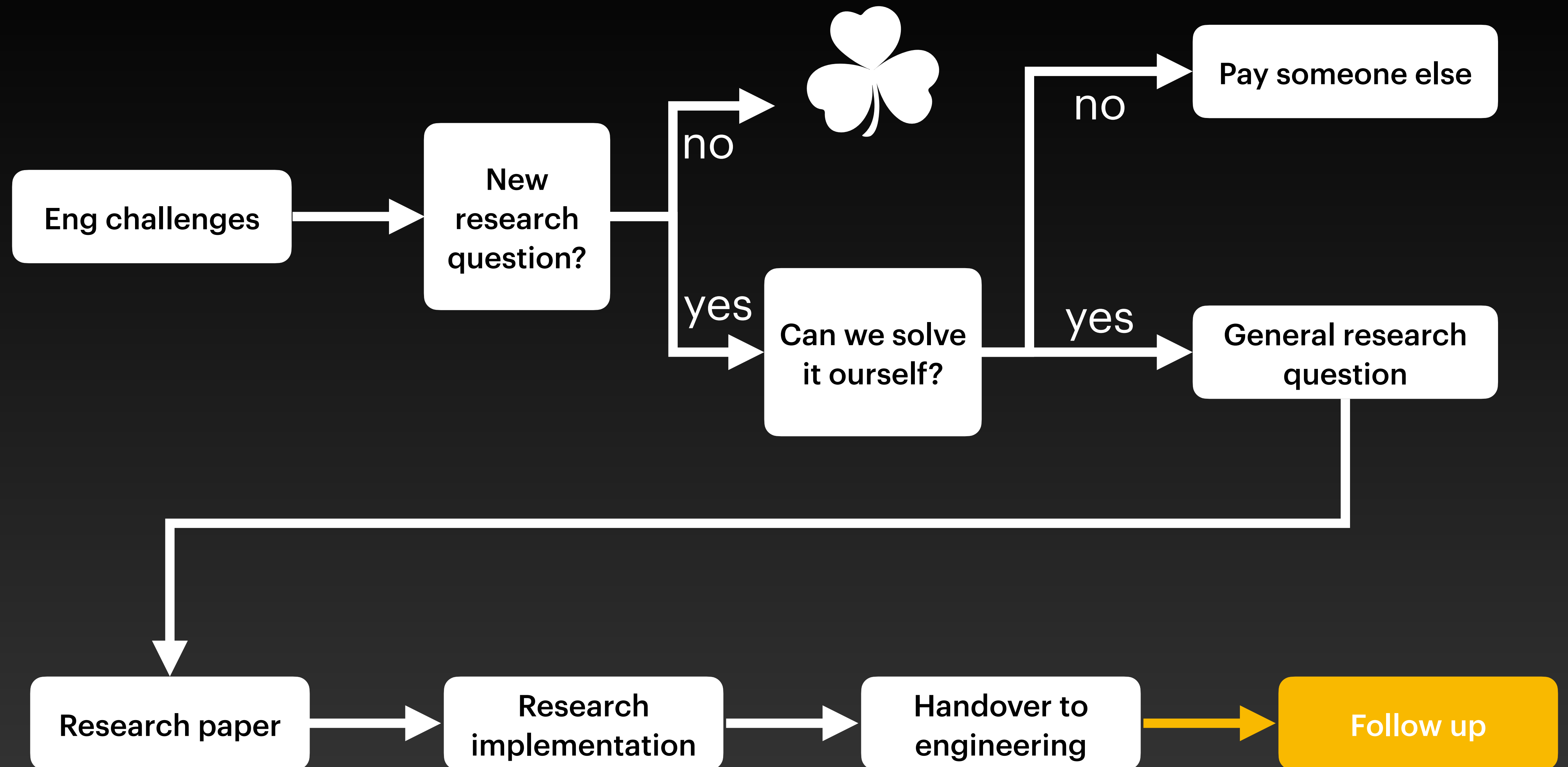


(please keep it short)









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>

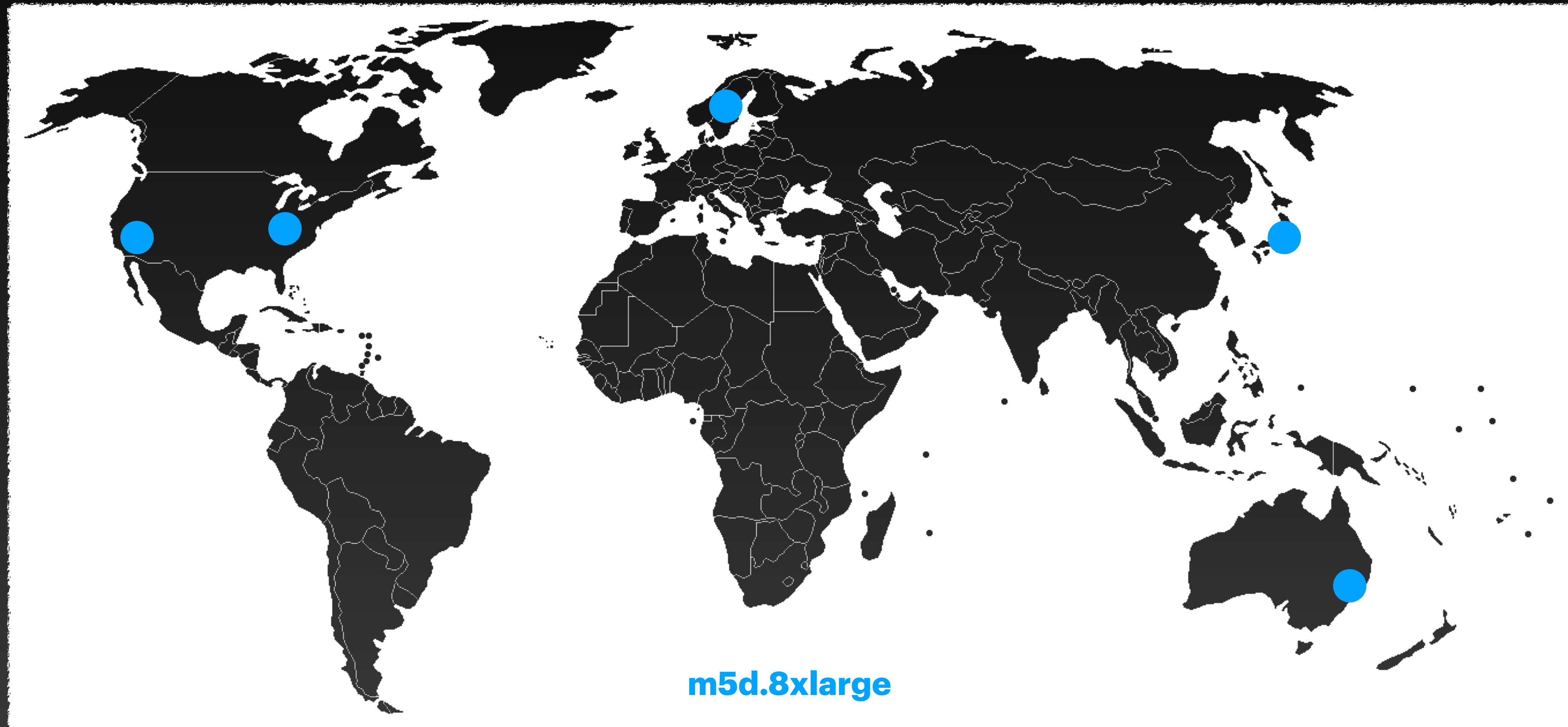
Implementation

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

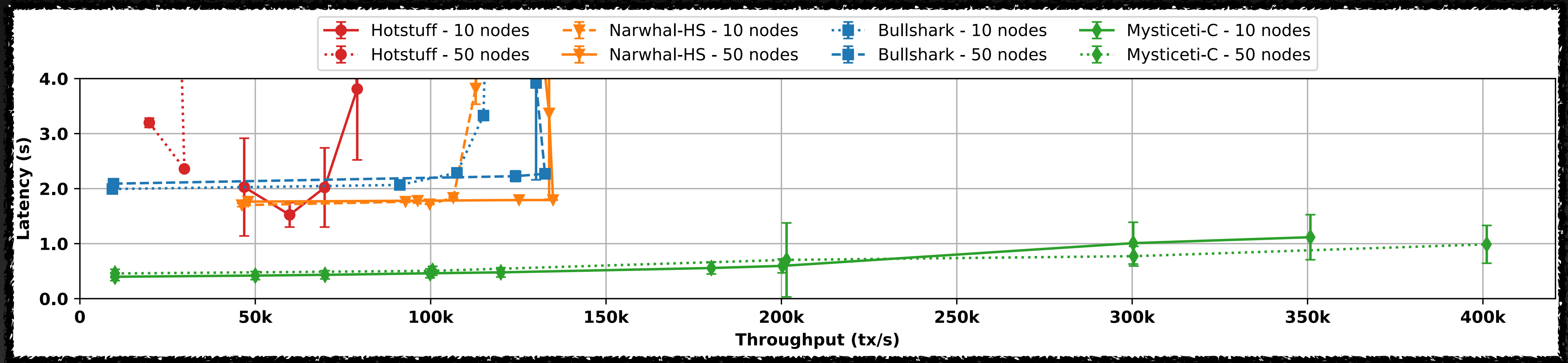
<https://github.com/mystenlabs/mysticeti>

Evaluation

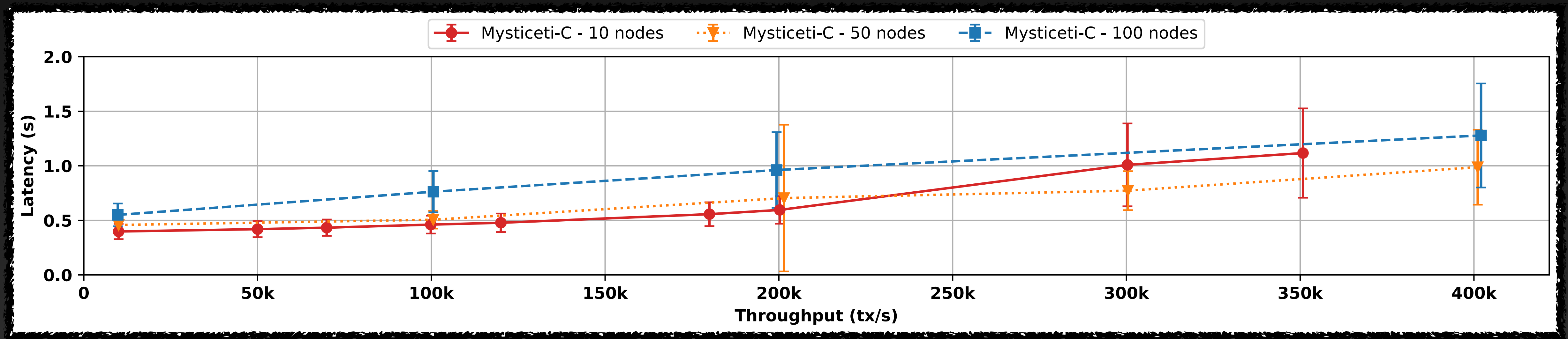
Experimental setup on AWS



Prototype Benchmarks



Prototype Benchmarks



Mysticeti

Key Limitations

- Block Synchroniser
- Parallelise block creation and synchronisation
- Rigid DAG structure?