

BFT Consensus

From Academic Paper to Mainnet

Alberto Sonnino

Alberto Sonnino

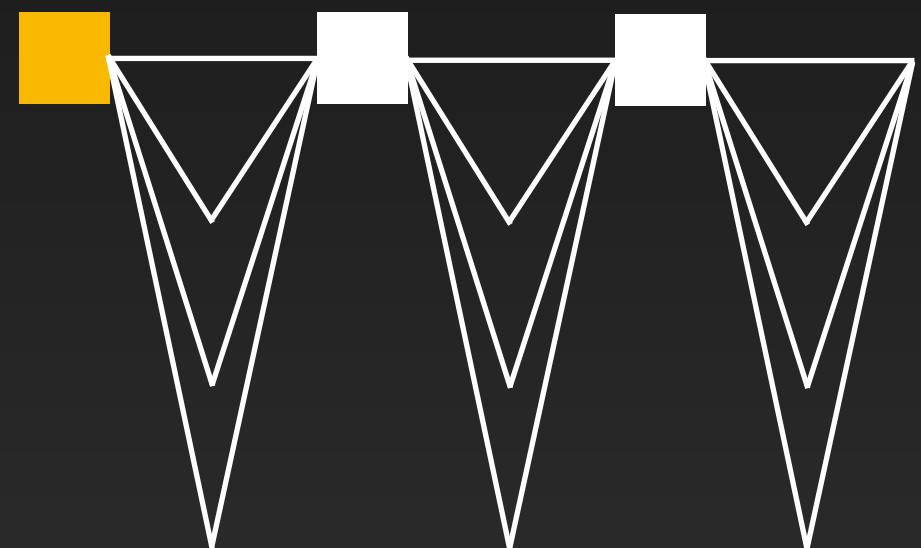
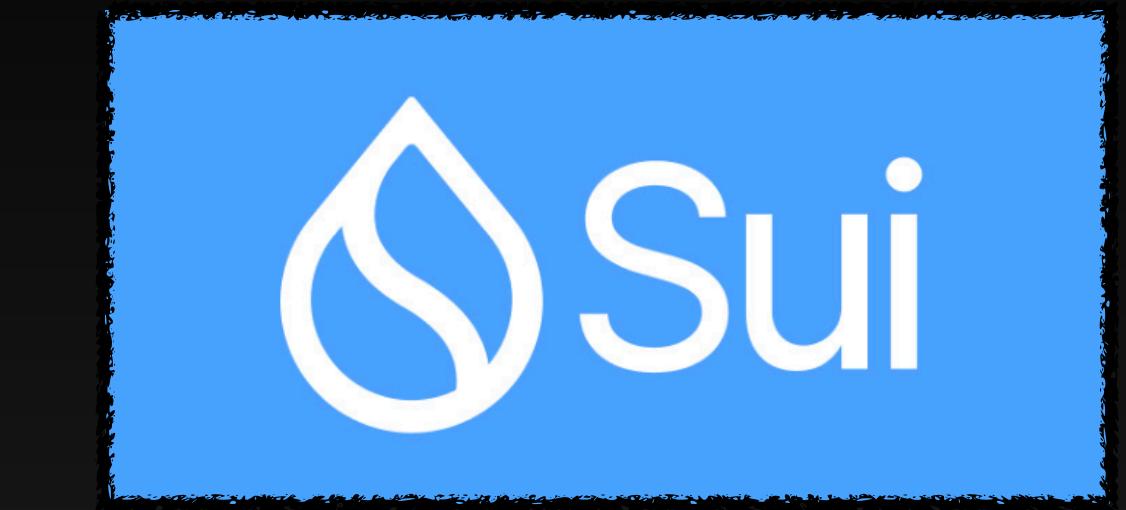
Research Scientist

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

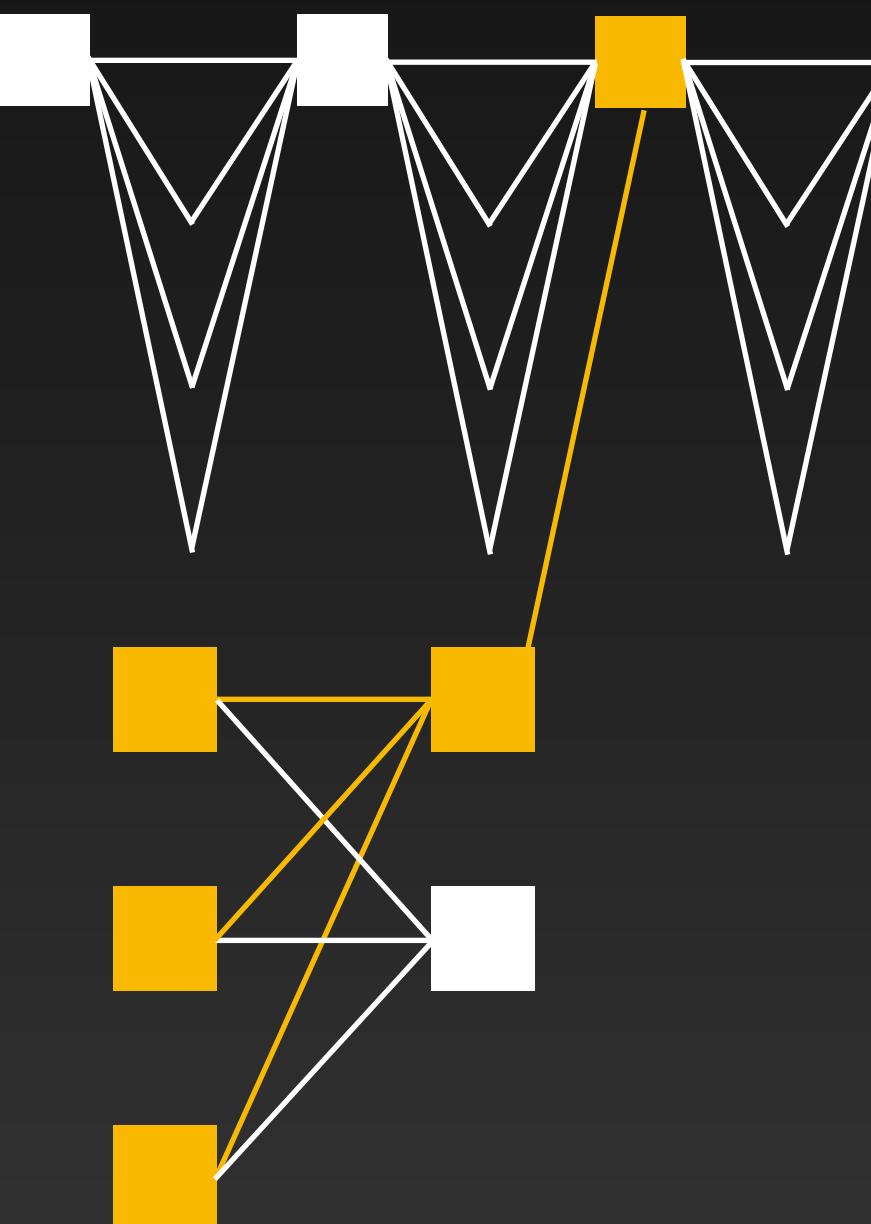
2019



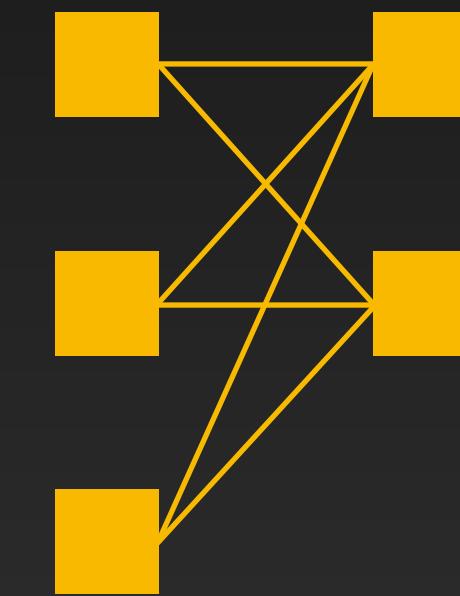
2024



HotStuff



HotStuff + Mempool

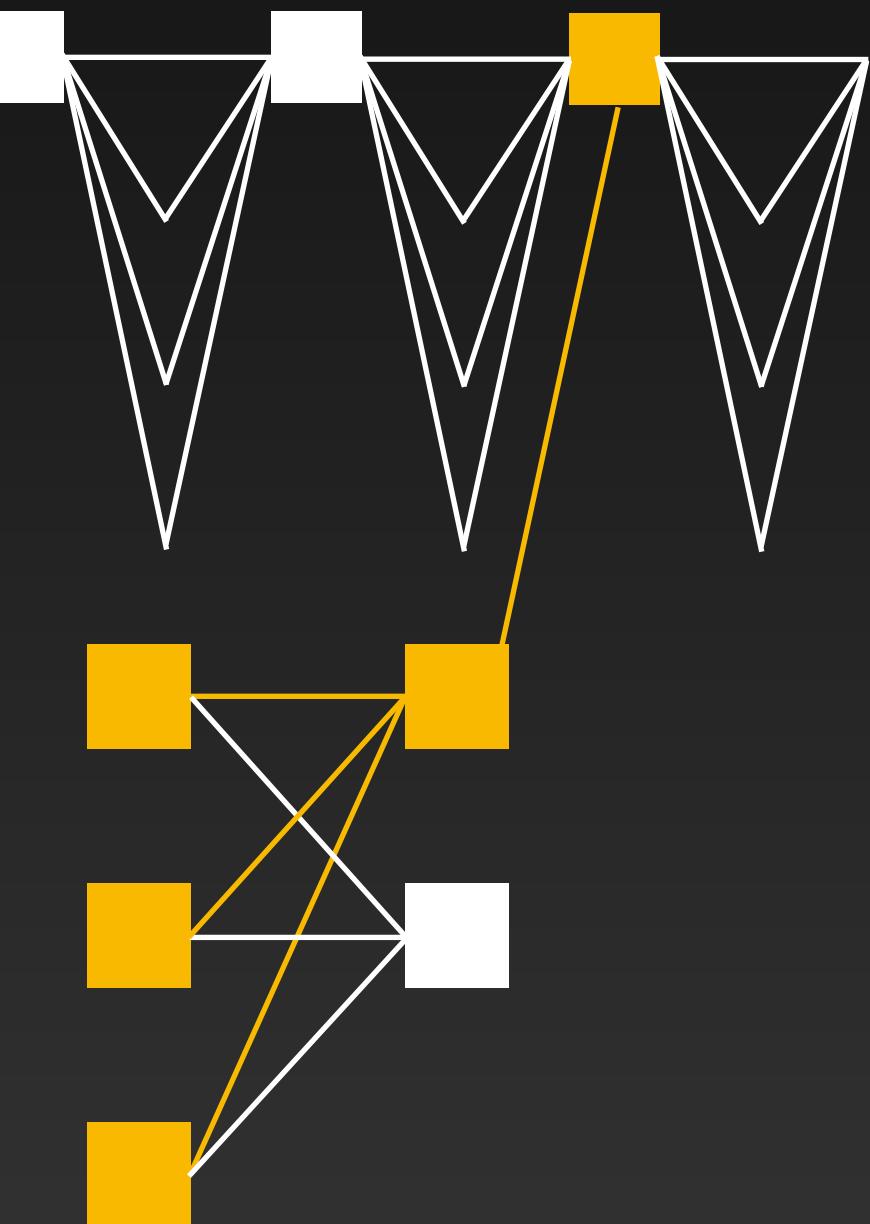
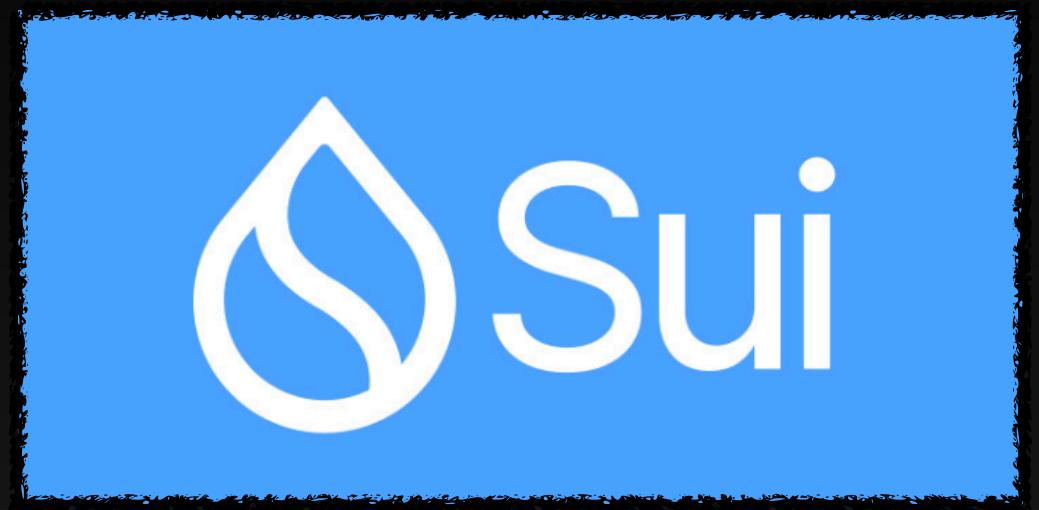


**Bullshark,
Mysticeti**

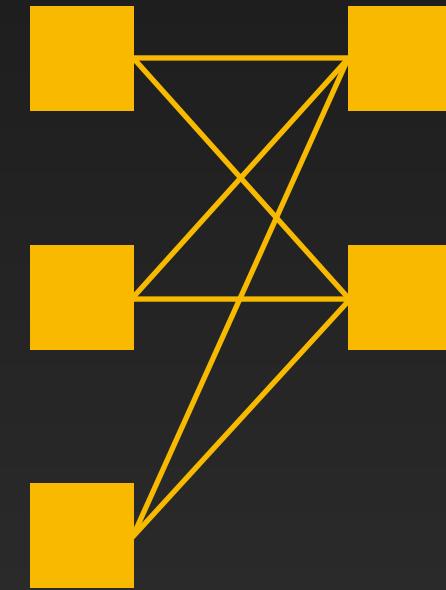
2019



2024



- Lessons learned
- Open research challenges



Research Gifts



(please keep it short)

Byzantine Fault Tolerance



Byzantine Fault Tolerance



$> 2/3$



Partial Synchrony

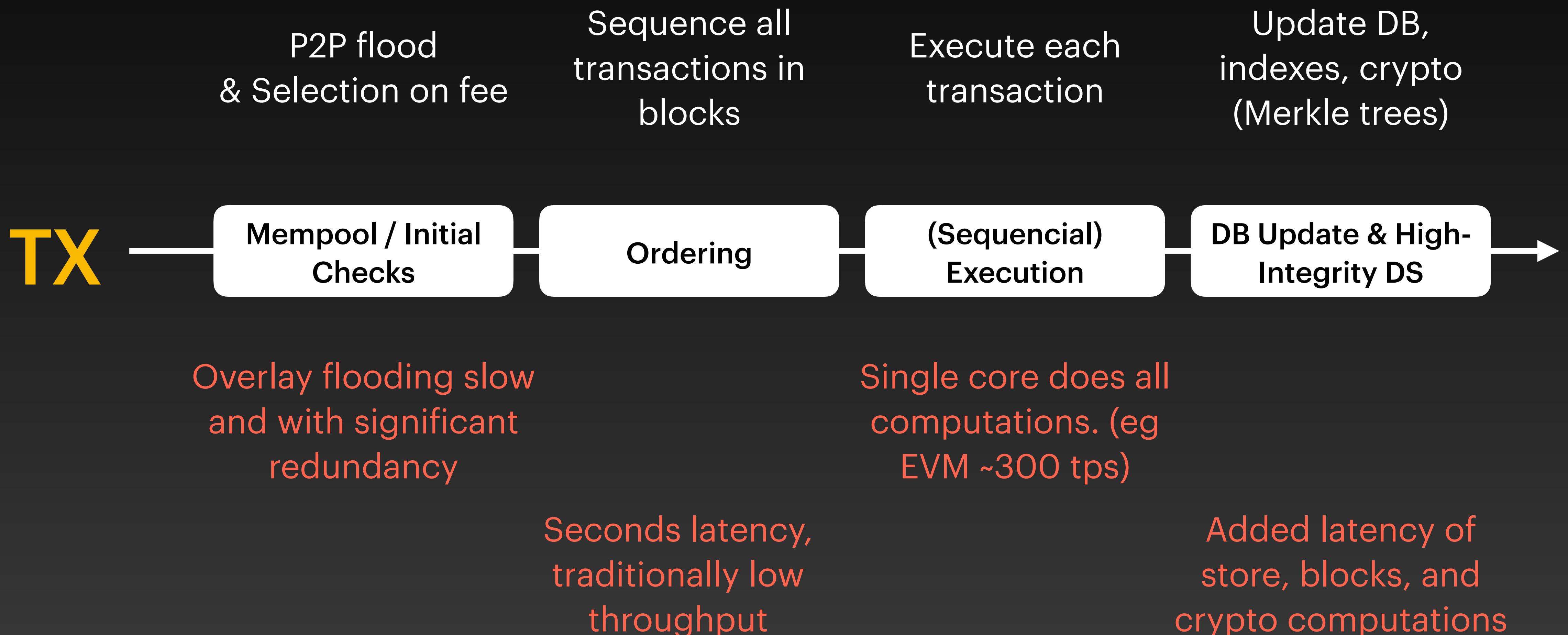


Research Questions

1. Network model?

Lessons Learned

Typical Blockchain



Typical Blockchain

**P2P flood
& Selection on fee**

**Sequence all
transactions in
blocks**

Mempool / Initial
Checks

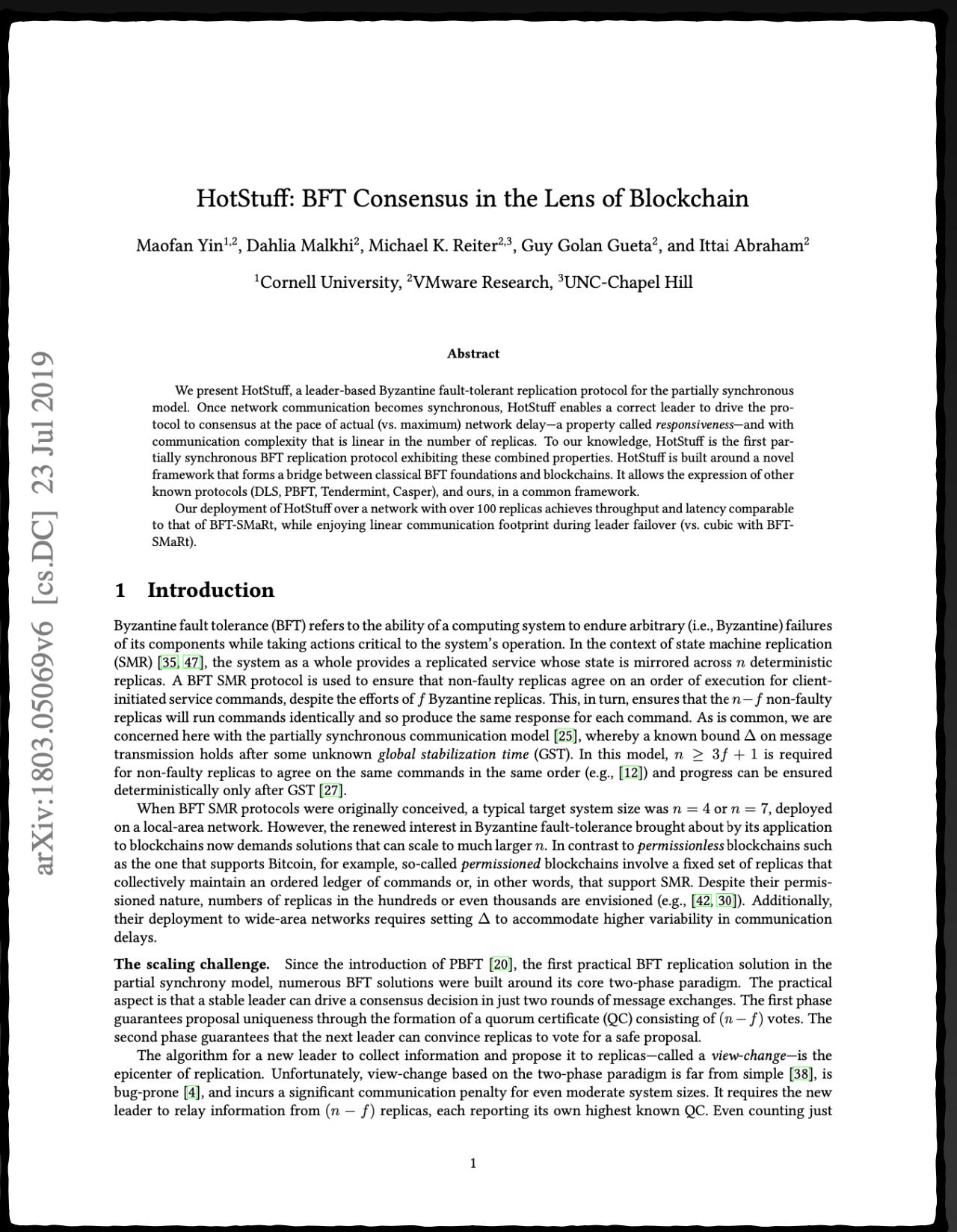
Ordering

**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 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 new 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



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

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 a lack of communication. The challenge is the creation 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 placed 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.

Algorithmic Byzantine consensus (under various assumptions) have existed for some time, indeed longer than the protocol 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 solve a difficult computational problem that takes the last old header as part of its input. 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 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

arXiv:1803.05069v6 [cs.DC] 23 Jul 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 failure (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.

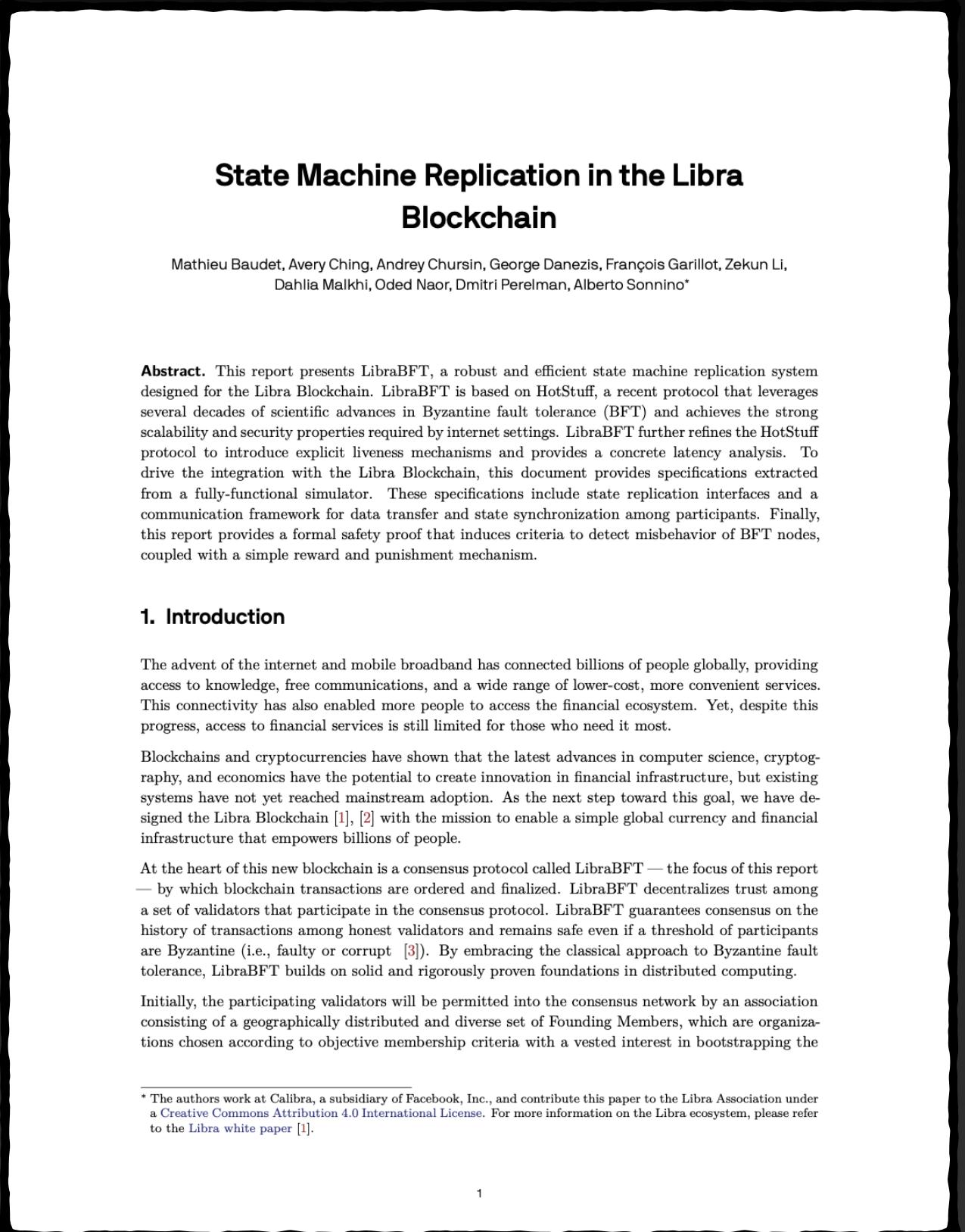
The algorithm for a new leader to collect information and propose it to replicas—called a *view-change*—is the epicenter of replication. Unfortunately, view-change based on the two-phase paradigm is far from simple [38], is bug-prone [4], and incurs a significant communication penalty for even moderate 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

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

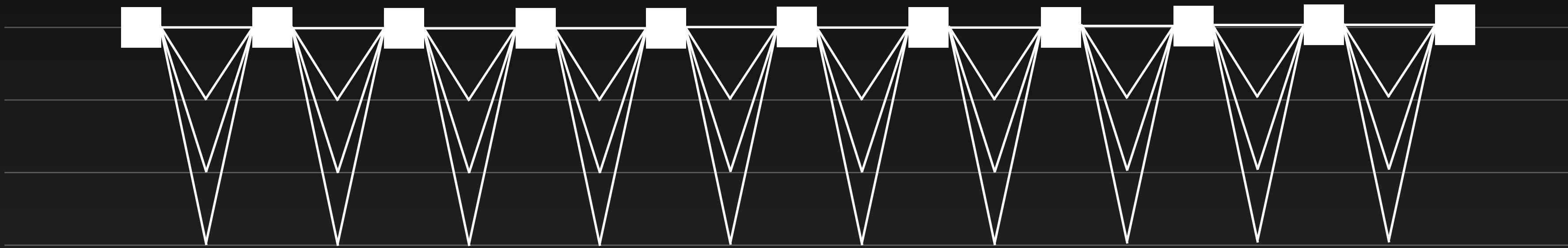
1. Network model?

Lessons Learned

1. Modularisation is a design strategy

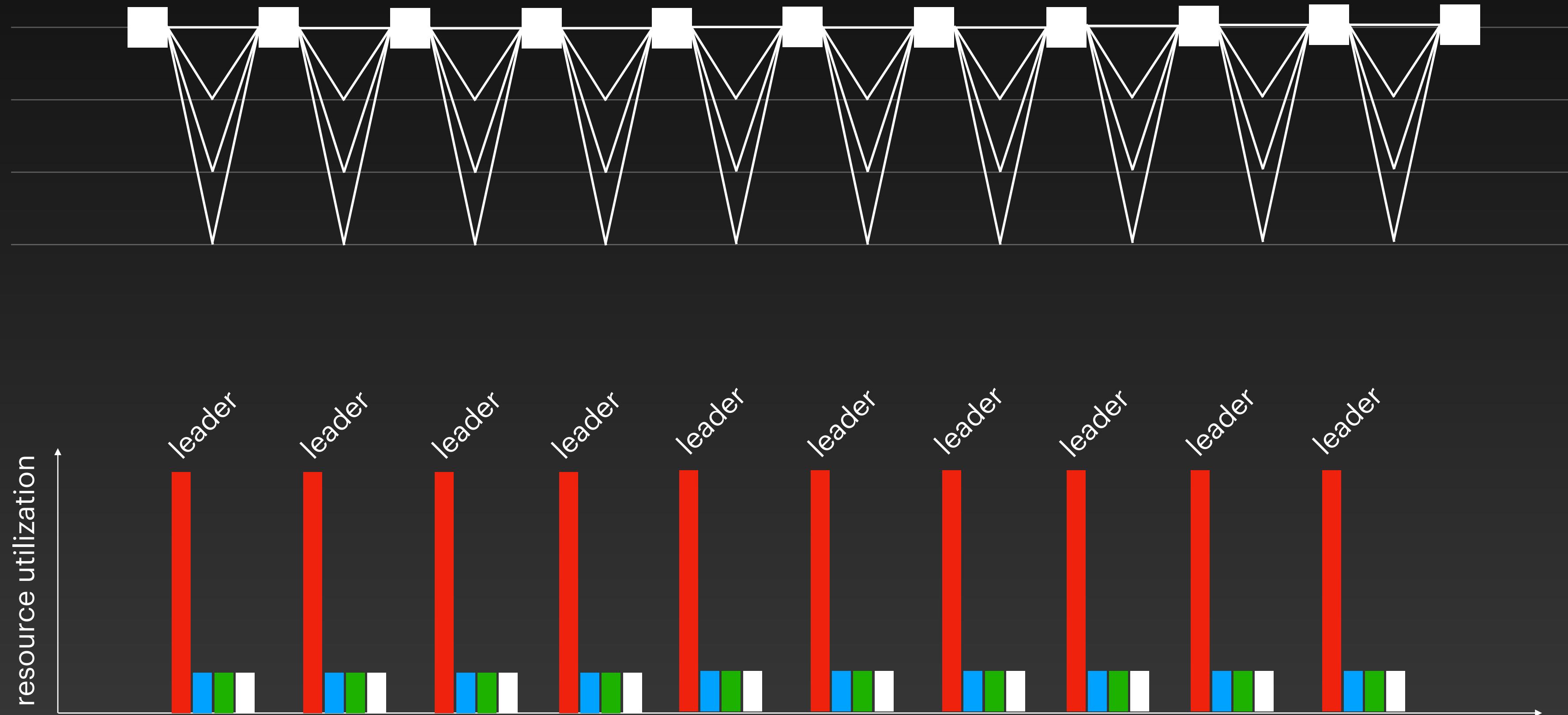
HotStuff

Typical leader-based protocols



HotStuff

Uneven resource utilisation



Research Questions

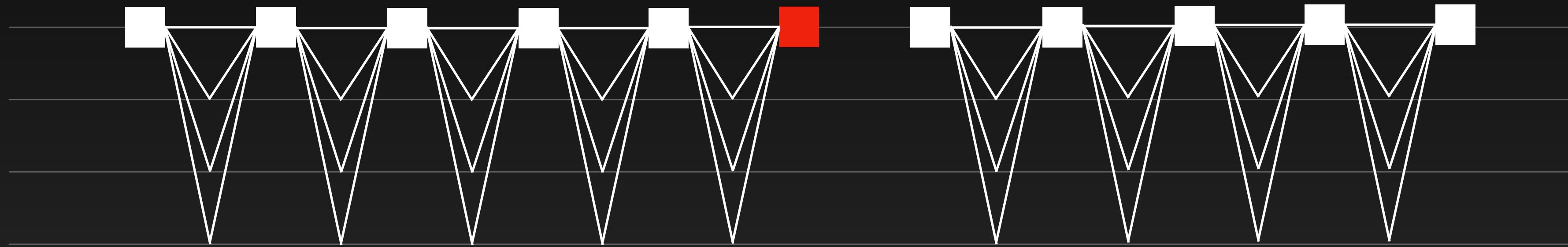
1. Network model?

Lessons Learned

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

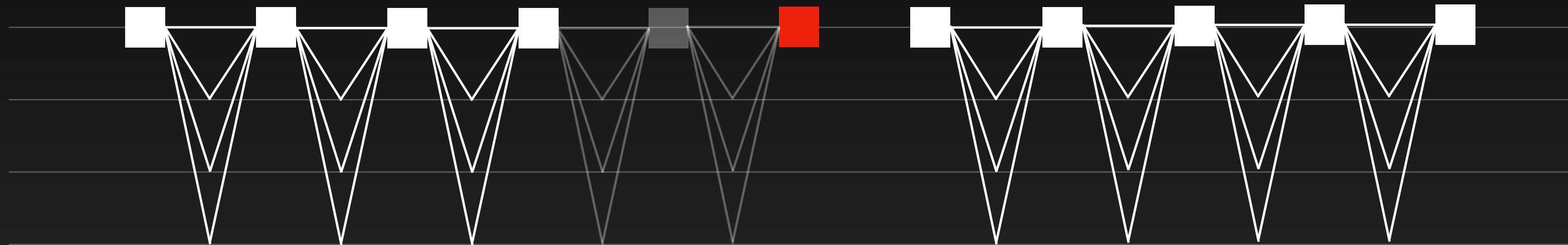
HotStuff

Fragility to faults and asynchrony

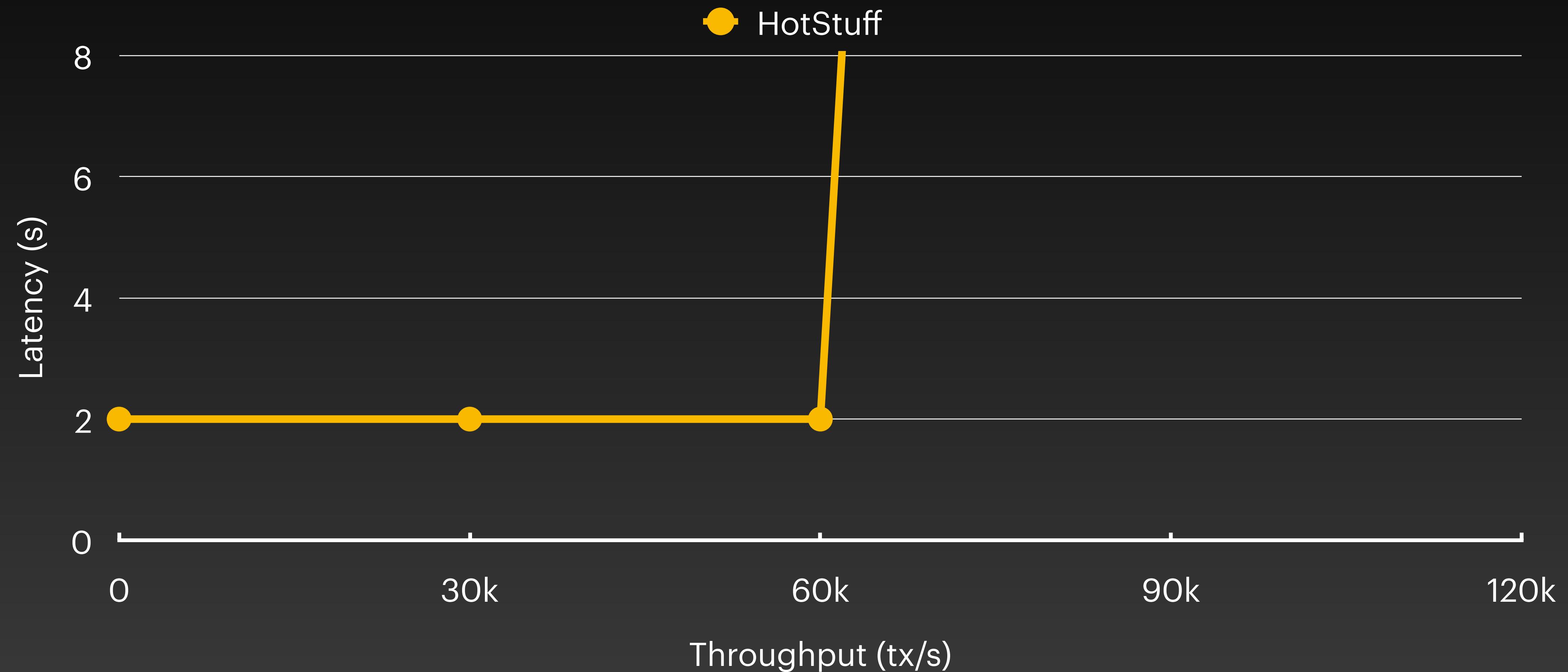


HotStuff

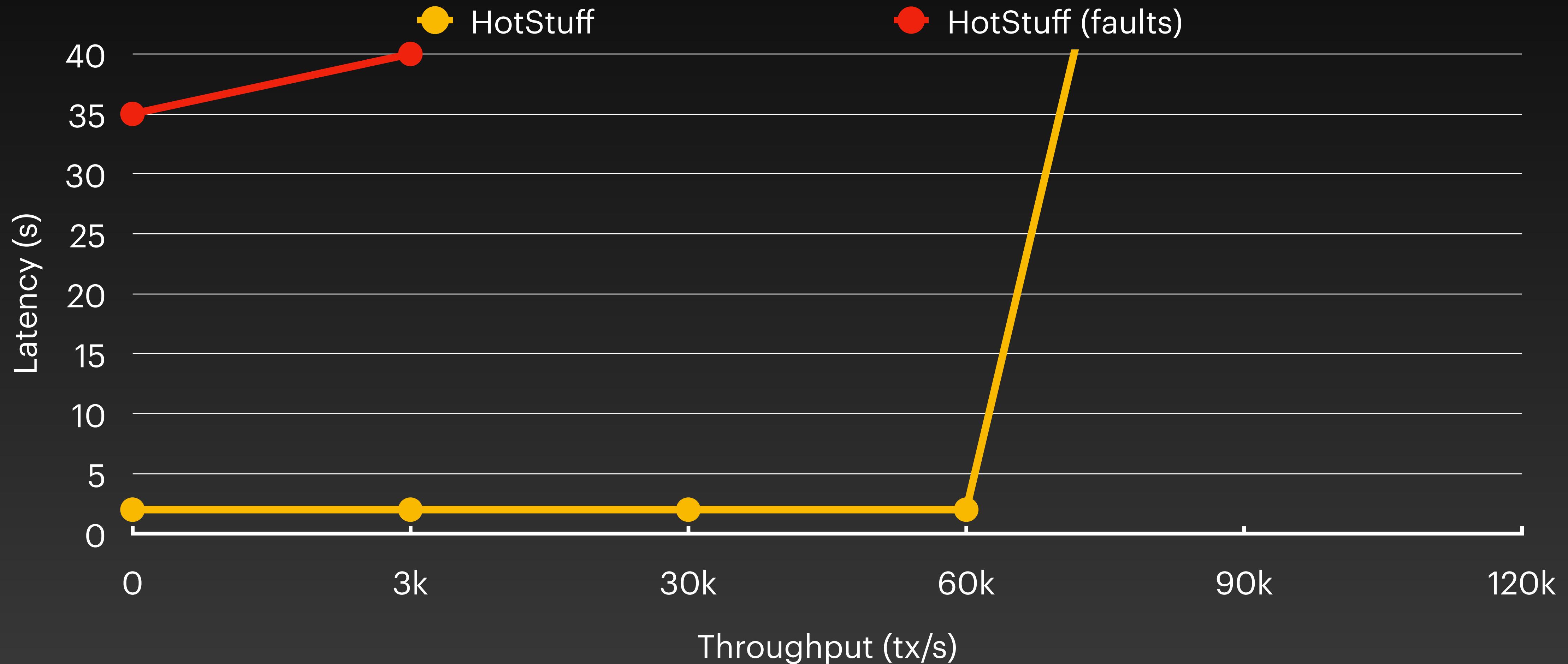
Fragility to faults and asynchrony



Performance



Performance



Research Questions

1. Network model?

Lessons Learned

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

Libra, 2019

arXiv:1803.05069v6 [cs.DC] 23 Jul 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 failure (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
- 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

Alberto Sonnino
Mysten Labs

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).

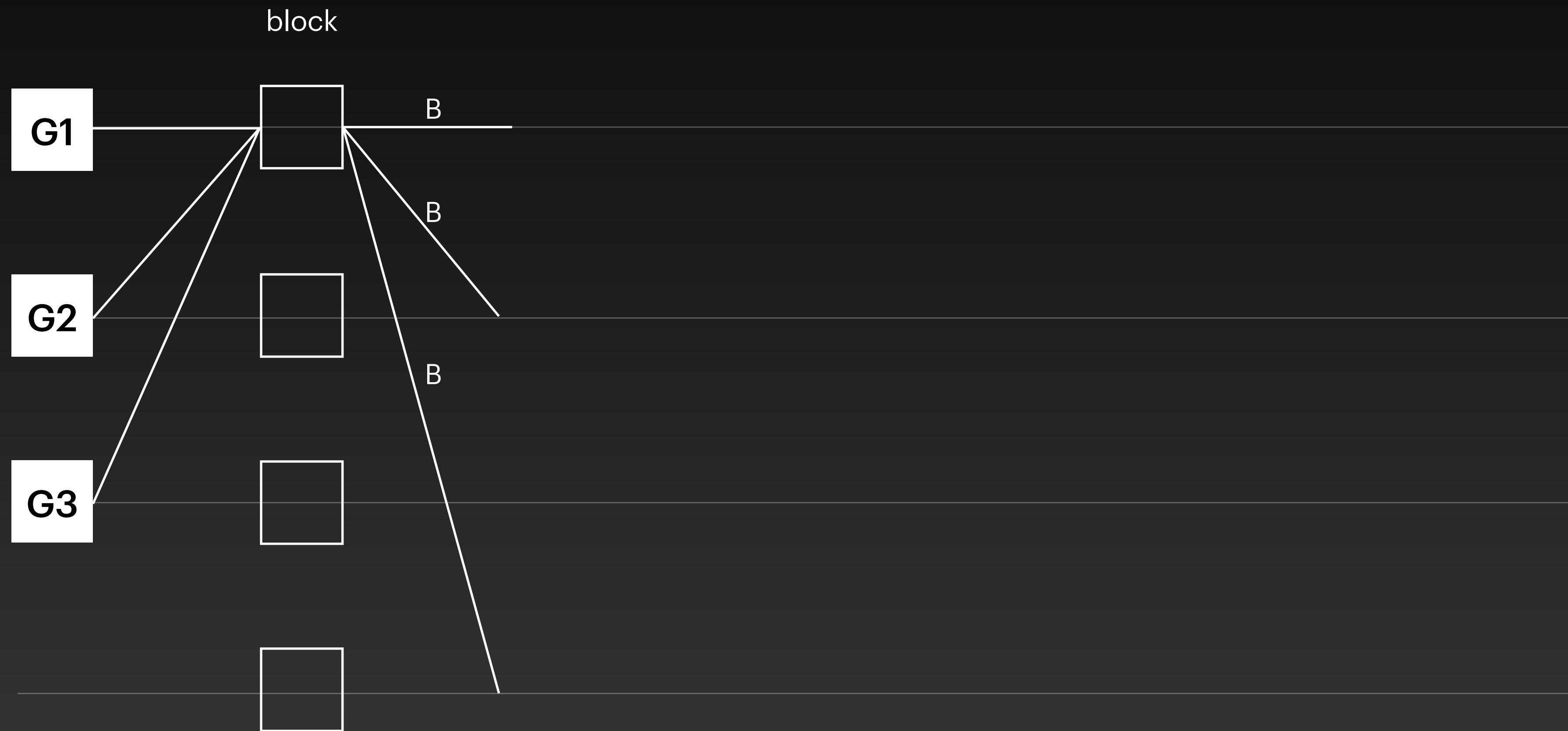
Additionally, consensus protocols have grouped a lot of functions into a monolithic protocol. In a typical distributed

Narwhal

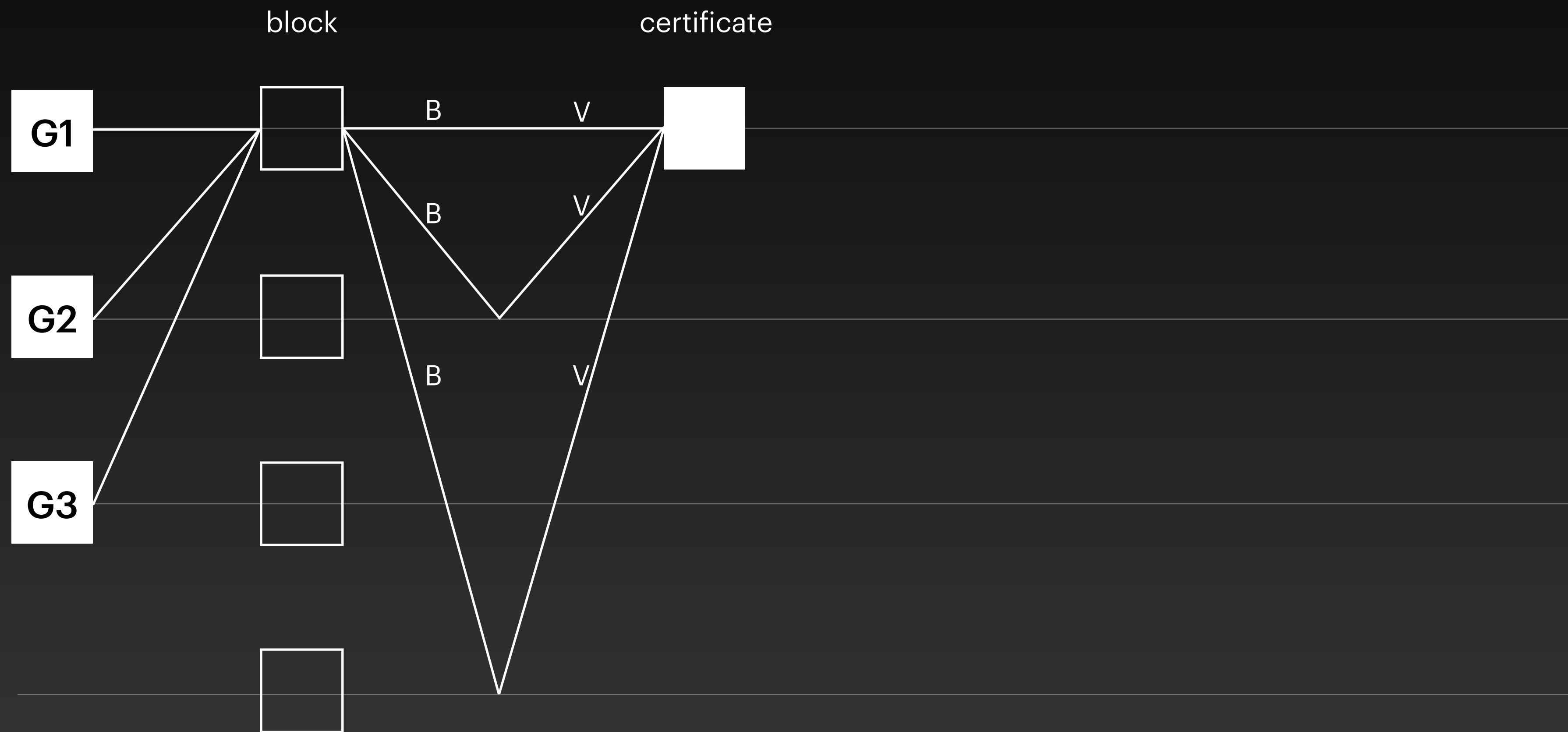
- Quadratic but even resource utilisation
- Separation between consensus and data dissemination

Narwhal

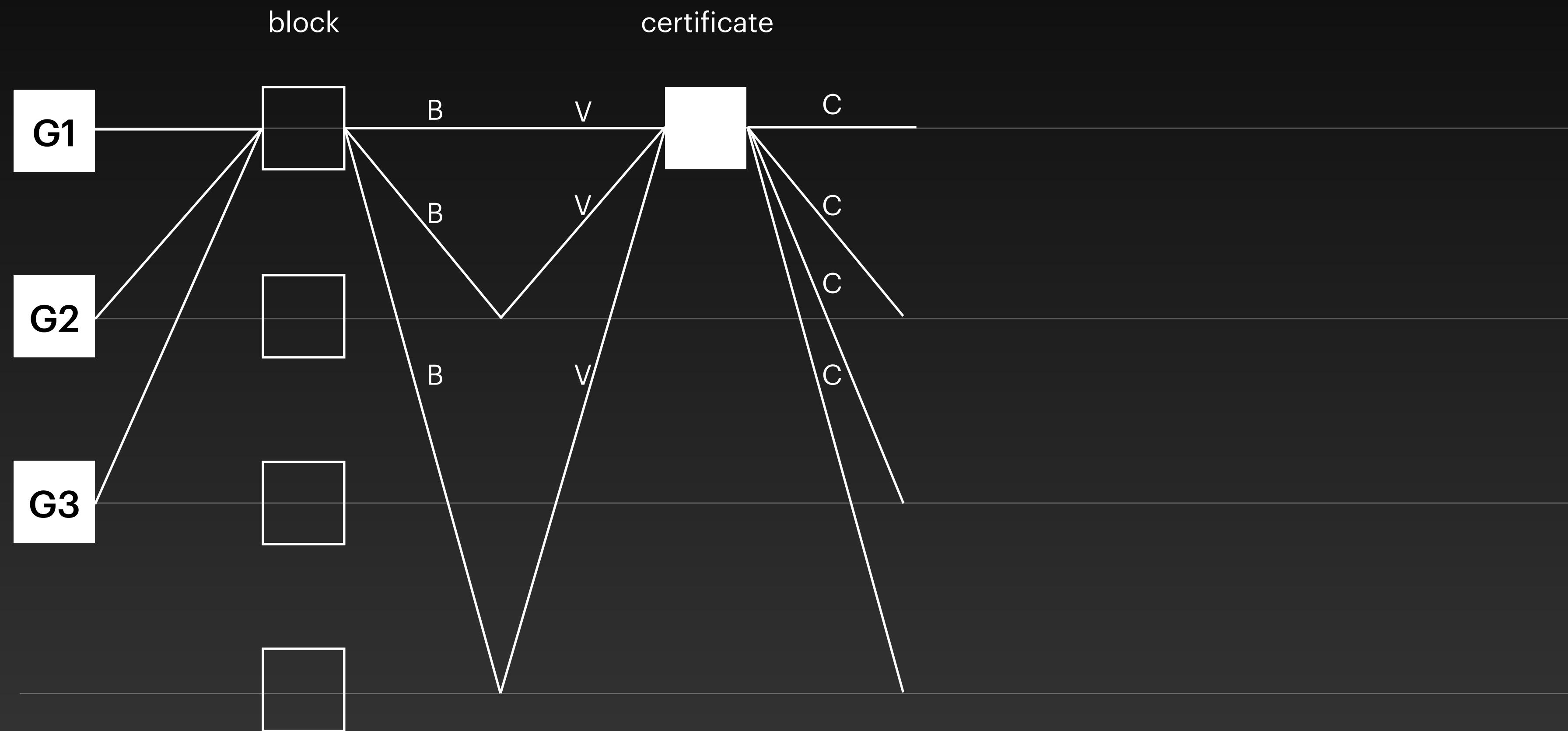
The primary machine



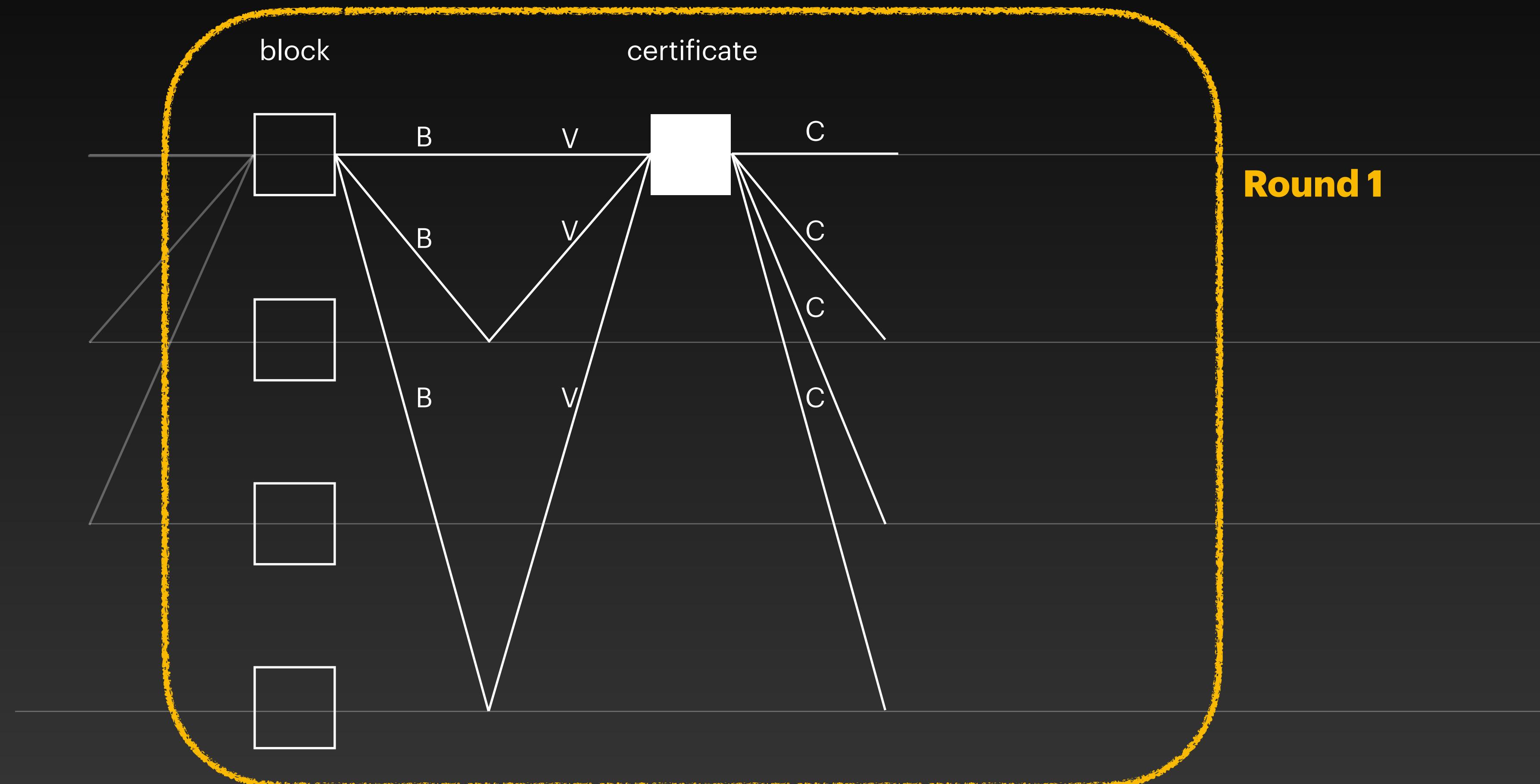
Narwhal



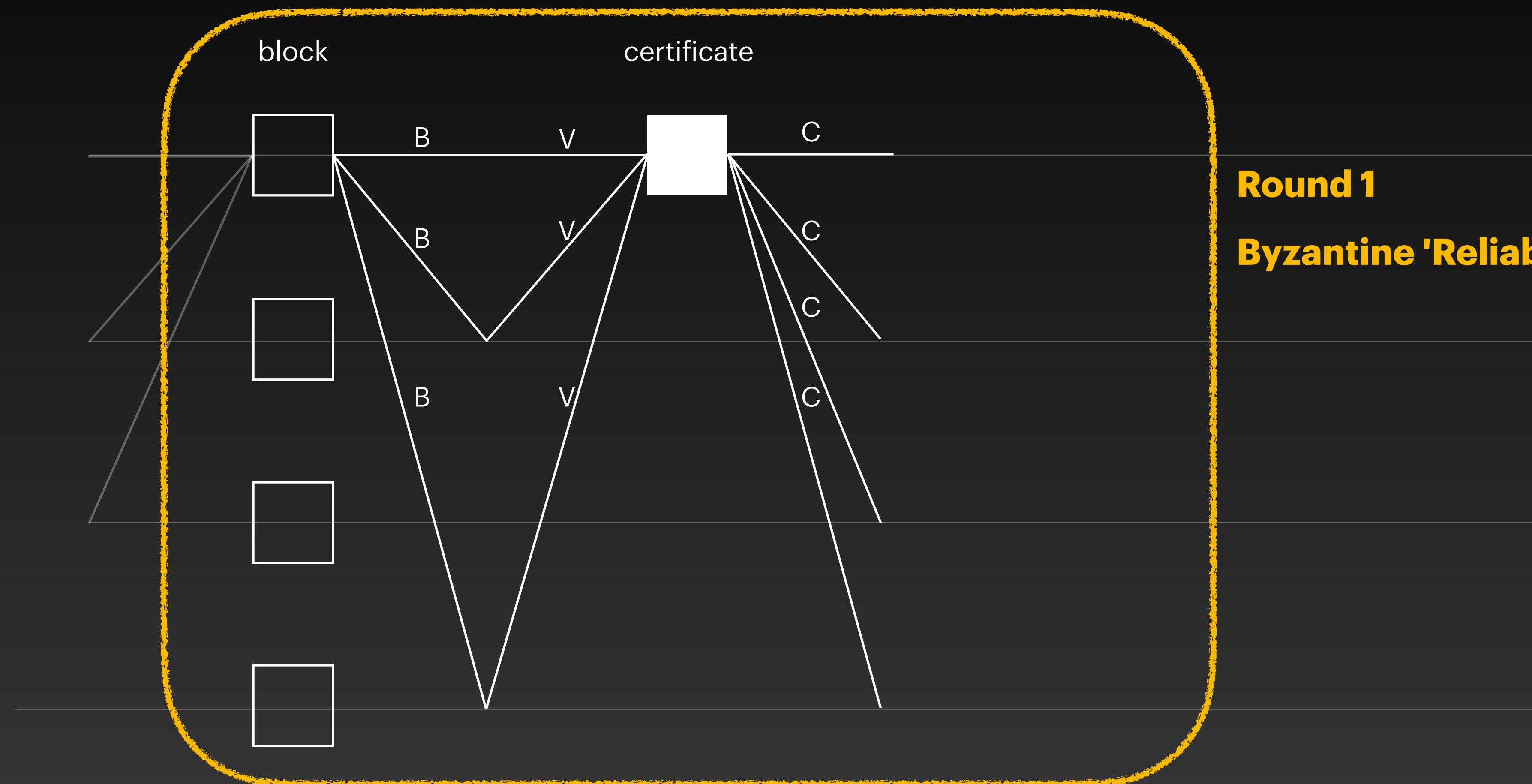
Narwhal



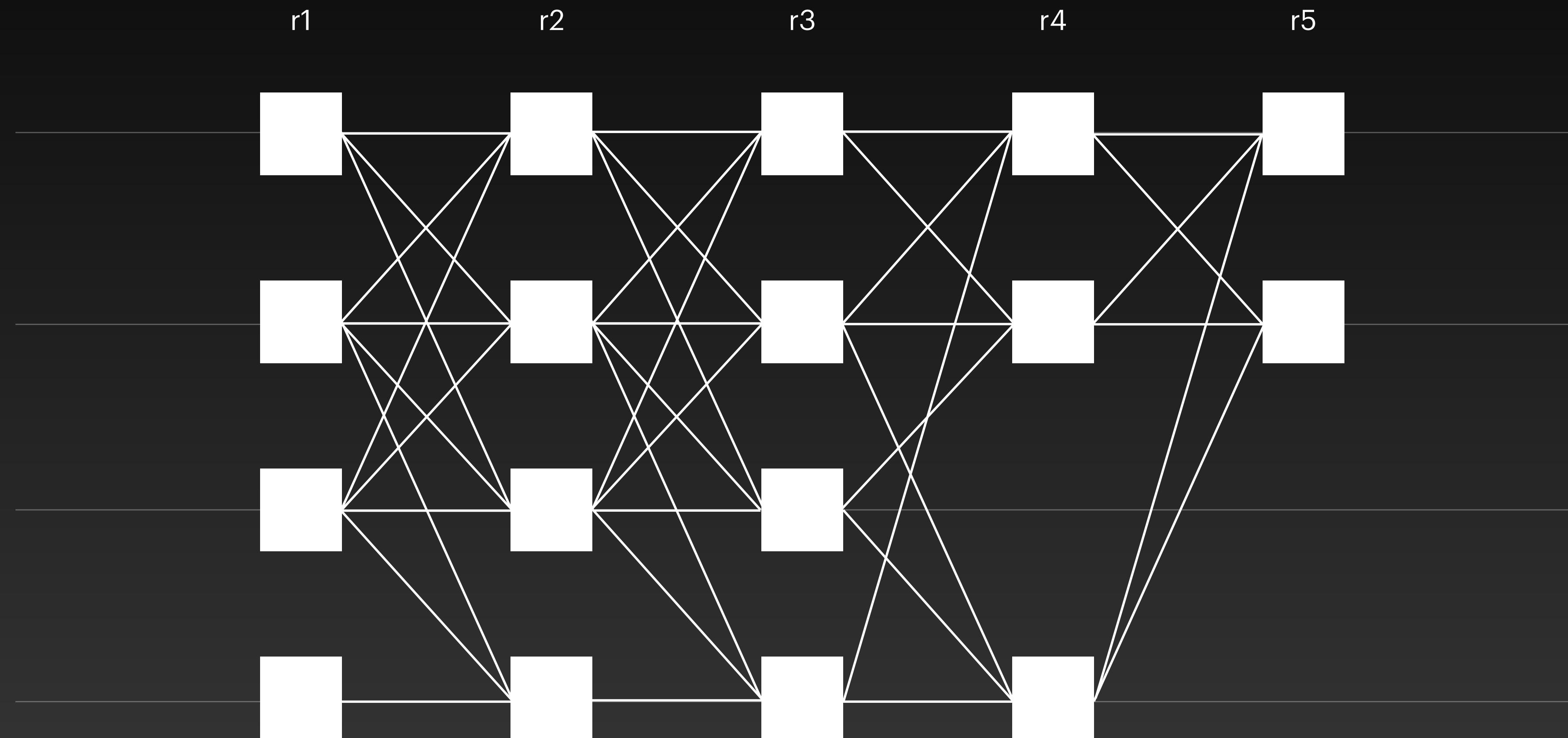
Narwhal



Narwhal



Narwhal



Research Questions

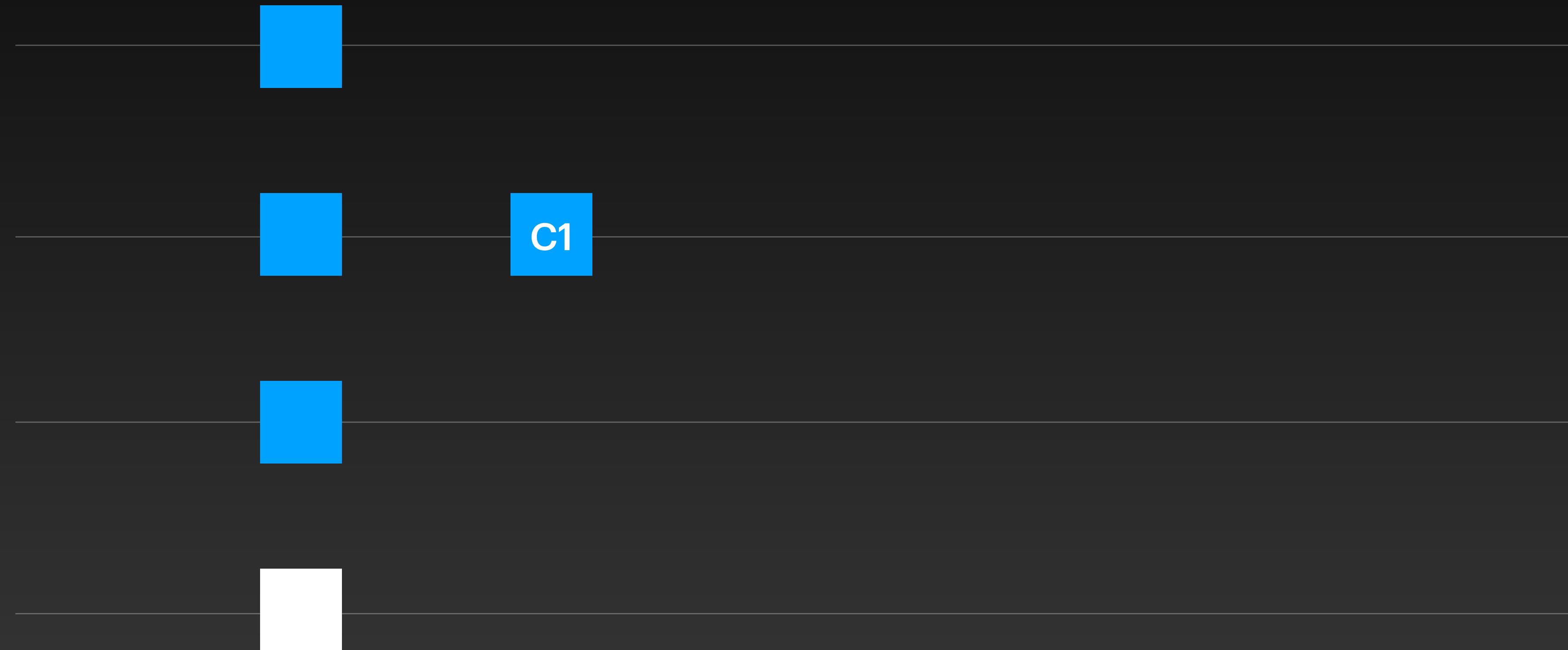
1. Network model?

Lessons Learned

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

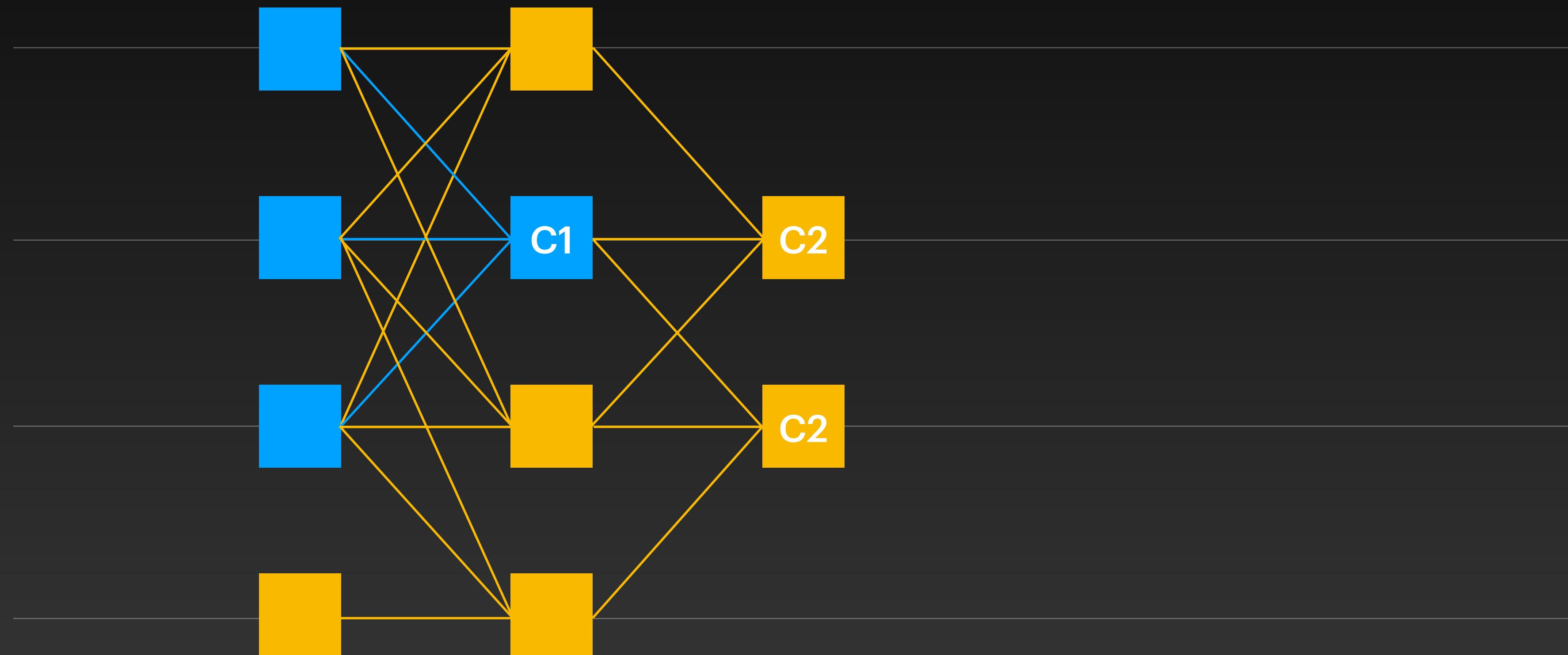
HotStuff on Narwhal

Enhanced commit rule



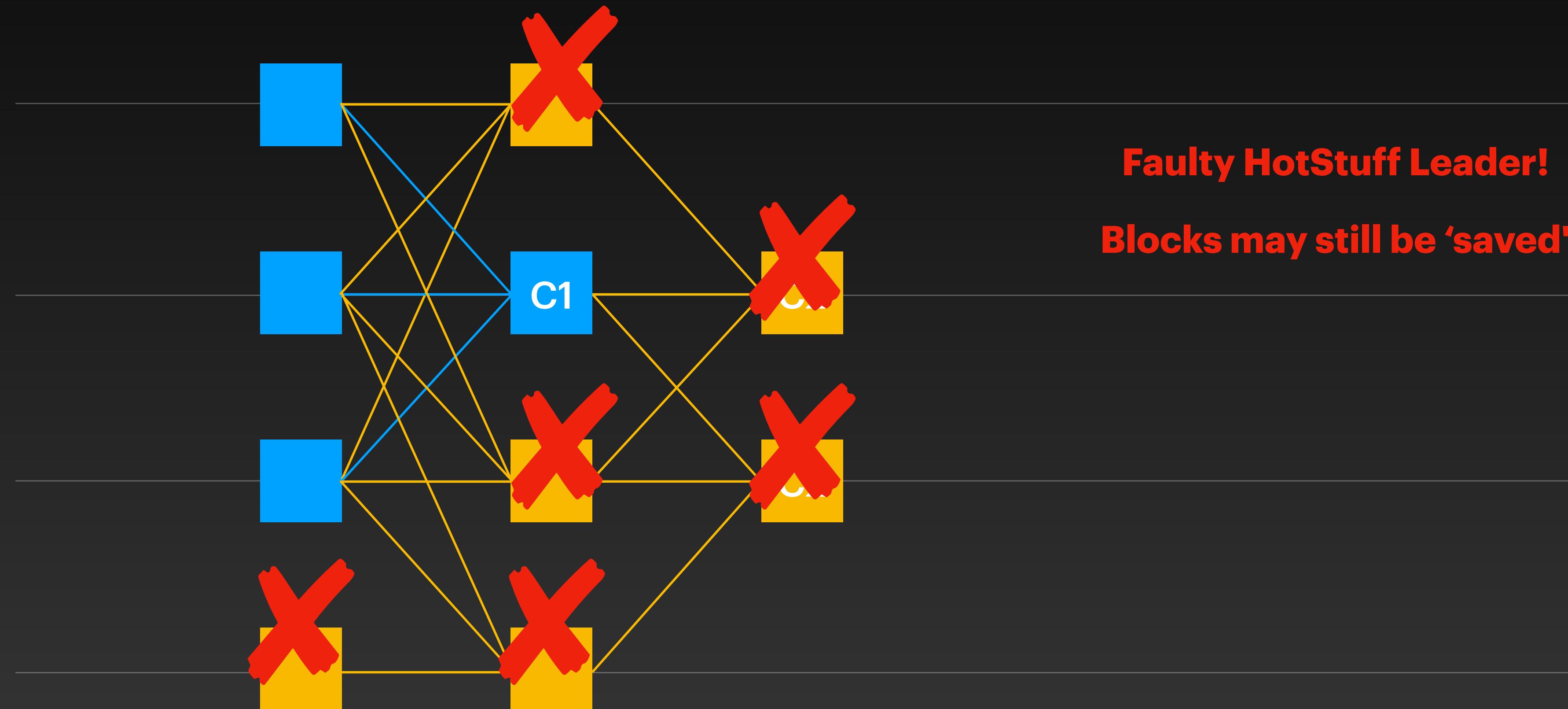
HotStuff on Narwhal

Enhanced commit rule



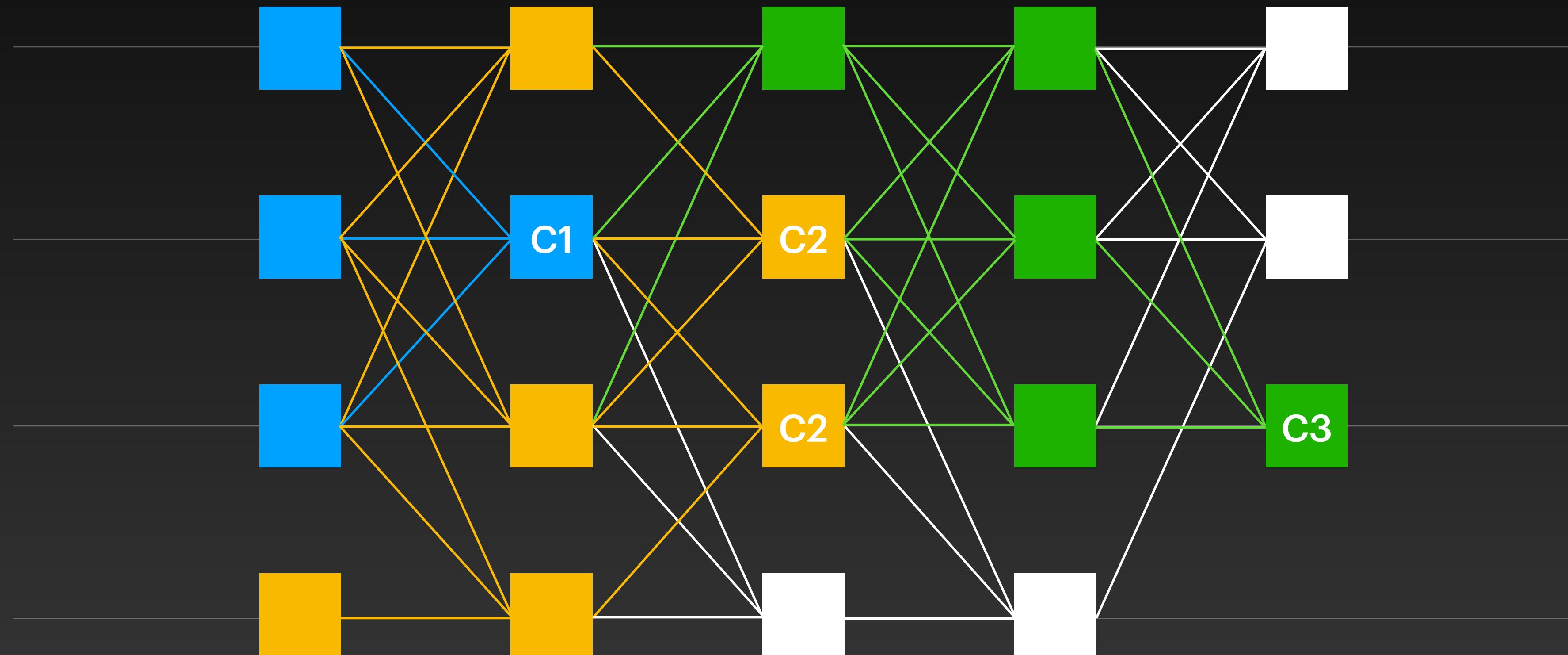
HotStuff on Narwhal

Enhanced commit rule



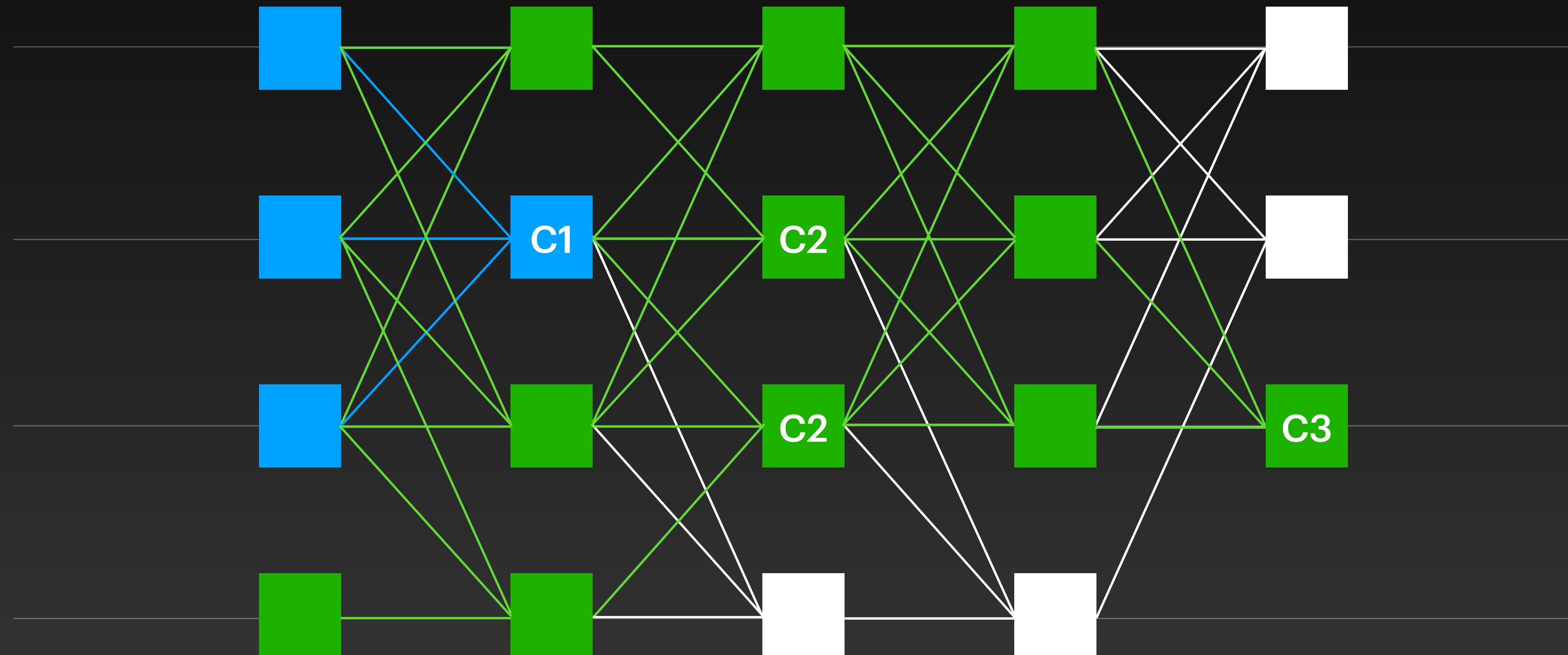
HotStuff on Narwhal

Enhanced commit rule

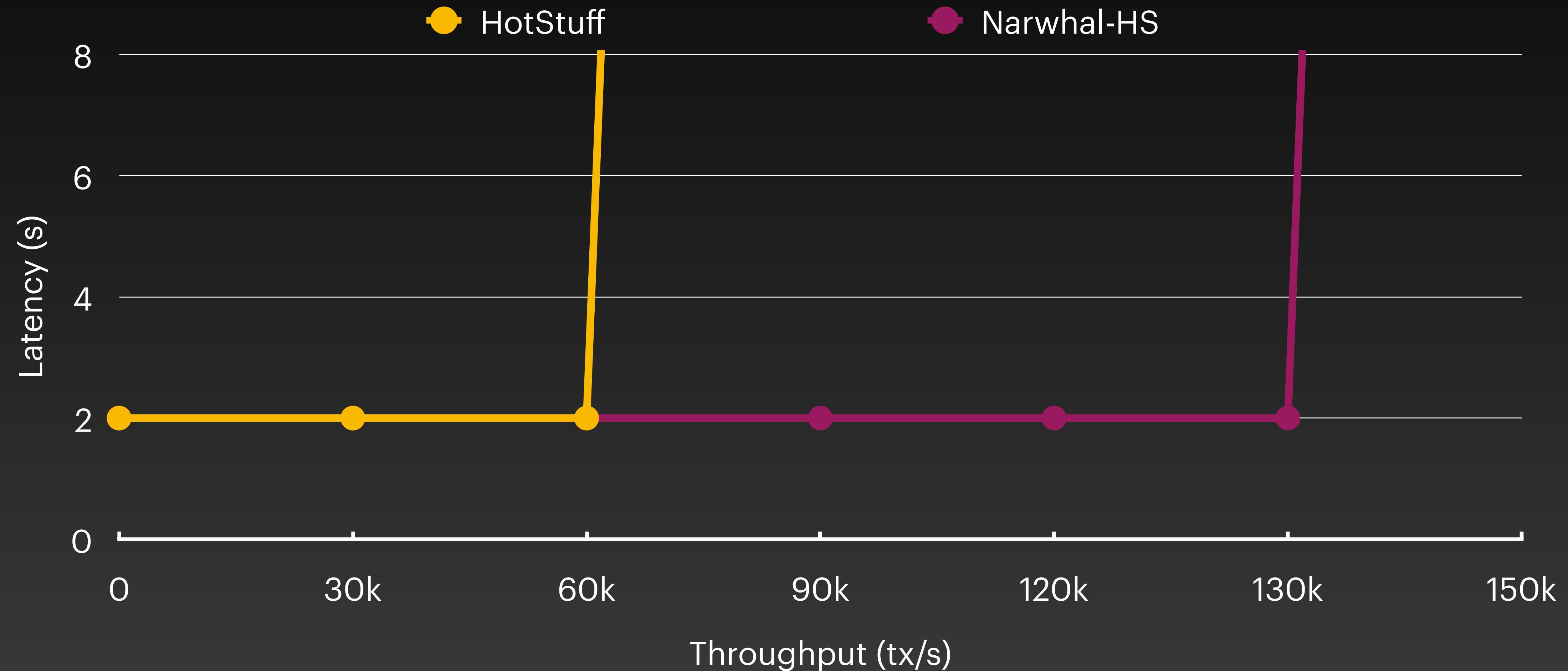


HotStuff on Narwhal

Enhanced commit rule



Performance



Libra, 2021

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

George Danezis
Mysten Labs & UCL

Alberto Sonnino
Mysten Labs

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

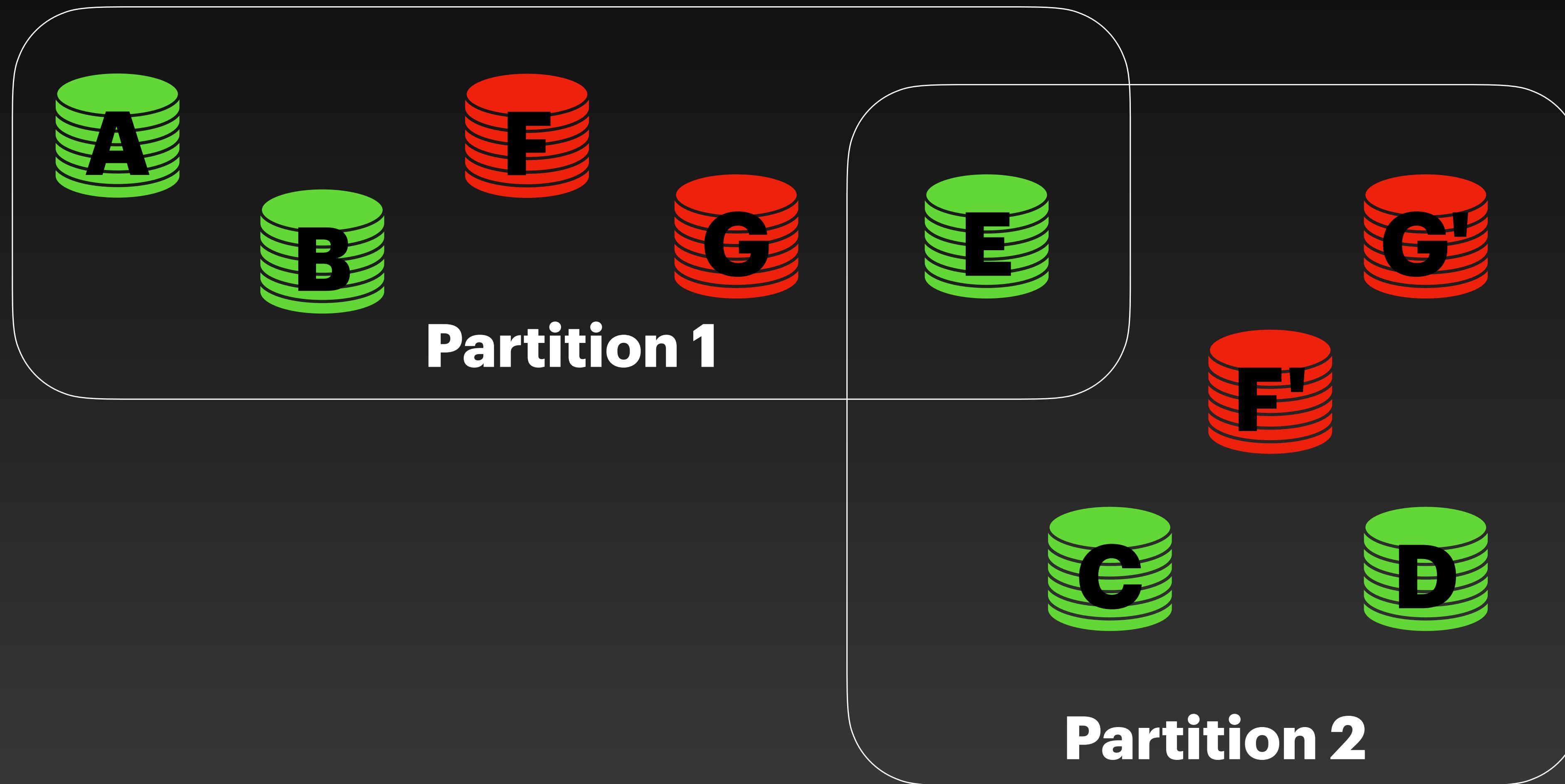
Research Questions

1. Network model?
2. BFT testing?

Lessons Learned

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

Twins



DagRider

arXiv:2102.08325v2 [cs.DC] 4 Jun 2021

Tusk

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](http://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

Bullshark

Bullshark: DAG BFT Protocols Made Practical

Alexander Spiegelman
sasha@aptoslabs.com
Aptos

Alberto Sonnino
alberto@mystenlabs.com
Mysten Labs

Neil Giriadharan
giridhar@berkeley.edu
UC Berkeley

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 Giriadharan, 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/nnnnnnnn.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

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. Copyright © 2022 Association for Computing Machinery (ACM) 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.

Conference '22, November 2022, Los Angeles, CA, USA

© 2022 Association for Computing Machinery.

ACM ISBN 978-1-xxxx-xxxx-x/YY-MM \$15.00

<https://doi.org/10.1145/nnnnnnnn.nnnnnnn>

logic required to safely order them. The idea is simple. To propose transactions, parties send them in a way that forms a causal 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 encode 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 break 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

arXiv:2209.00750v3 [cs.CR] 1 Feb 2024

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

Yingzi Gao*

ISCAS & UCAS

yingzi2019@iscas.ac.cn

Qiang Tang*

USYD

qiang.tang@sydney.edu.au

Yuan Lu*

ISCAS

luyuan@iscas.ac.cn

Jing Xu*

ISCAS

xujing@iscas.ac.cn

Zhenliang Lu*

USYD

zhu9620@uni.sydney.edu.au

Zhenfeng Zhang*

ISCAS

zhenfeng@iscas.ac.cn

ABSTRACT

Despite recent progresses of practical asynchronous Byzantine-fault tolerant (BFT) consensus, the state-of-the-art designs still suffer from suboptimal performance. Particularly, to obtain maximum throughput, most existing protocols with guaranteed linear amortized communication complexity require each participating node to broadcast a huge batch of transactions, which dramatically sacrifices latency. Worse still, the f slowest nodes' broadcasts might never be agreed to output and thus can be censored (where f is the number of faults). Implementable mitigation to the threat either uses computationally costly threshold encryption or incurs communication blow-up by letting the honest nodes to broadcast redundant transactions, thus causing further efficiency issues.

We present Dumbo-NG, a novel asynchronous BFT consensus (atomic broadcast) to solve the remaining practical issues. Its technical core is a non-trivial *direct reduction* from asynchronous atomic broadcast to multi-valued validated Byzantine agreement (MVBA) with *quality* property (which ensures the MVBA output is from honest nodes with $1/2$ probability). Most interestingly, the new protocol structure empowers completely concurrent execution of transaction dissemination and asynchronous agreement. This brings about two benefits: (i) the throughput-latency tension is resolved to approach peak throughput with minimal increase in latency; (ii) the transactions broadcasted by any honest node can be agreed to output, thus conquering the censorship threat with no extra cost.

We implemented Dumbo-NG with the current fastest GLL+22 MVBA with quality (NDSS'22) and compare it to the state-of-the-art asynchronous BFT with guaranteed censorship resilience including Dumbo (CCS'20) and Speeding-Dumbo (NDSS'22). Along the way, we apply the techniques from Speeding-Dumbo to DispersedLedger (NSDI'22) and obtain an improved variant of DispersedLedger called sDumbo-DL for comprehensive comparison. Extensive experiments (over up to 64 AWS EC2 nodes across all 16 AWS regions) reveal: Dumbo-NG realizes a peak throughput 4-8x over Dumbo, 2-4x over Speeding-Dumbo, and 2-3x over sDumbo-DL (for varying scales); More importantly, Dumbo-NG's latency, which is lowest among all tested protocols, can almost remain stable when throughput grows.

CCS CONCEPTS

- Security and privacy → Systems security; Distributed systems security;
- Computer systems organization → Reliability.

*Authors are listed alphabetically. Yingzi, Yuan & Zhenliang made equal contributions. An abridged version of the paper will appear in ACM CCS 2022.

KEYWORDS

Asynchronous consensus, Byzantine-fault tolerance, blockchain

1 INTRODUCTION

The huge success of Bitcoin [63] and blockchain [19, 24] leads to an increasing tendency to lay down the infrastructure of distributed ledger for mission-critical applications. Such decentralized business is envisioned as critical global infrastructure maintained by a set of mutually distrustful and geologically distributed nodes [1], and thus calls for consensus protocols that are both secure and efficient for deployment over the Internet.

Asynchronous BFT for indispensable robustness. The consensus of decentralized infrastructure has to thrive in a highly adversarial environment. In particular, when the applications atop it are critical financial and banking services, some nodes can be well motivated to collude and launch malicious attacks. Even worse, the unstable Internet might become part of the attack surface due to network fluctuations, misconfigurations and even network attacks. To cope with the adversarial deployment environment, asynchronous Byzantine-fault tolerant (BFT) consensus [4, 20, 35, 47, 58, 60] are arguably the most suitable candidates. They can realize high security-assurance to ensure liveness (as well as safety) despite an asynchronous adversary that can arbitrarily delay messages. In contrast, many (partial) synchronous consensus protocols [5, 6, 8, 15, 27, 44, 45, 64, 73] such as PBFT [26] and HotStuff [75] might sustain the inherent *loss of liveness* (i.e., generate unbounded communications without making any progress) [36, 60] when unluckily encountering an asynchronous network adversary.

1.1 Practical obstacles of adopting asynchronous BFT consensus

Unfortunately, it is fundamentally challenging to realize practical asynchronous BFT consensus, and none of such protocols was widely adopted due to serious efficiency concerns. The seminal FLP “impossibility” [36] proves that no *deterministic* consensus exists in the asynchronous network. Since the 1980s, many attempts [1, 12, 13, 21, 25, 65, 67] aimed at circumventing the “impossibility” by *randomized* protocols, but most of them focused on theoretical feasibility, and unsurprisingly, several attempts of implementations [22, 61] had inferior performance.

Until recently, the work of HoneyBadger BFT (HBBFT) demonstrated the first asynchronous BFT consensus that is performant in the wide-area network [60]. As shown in Figure 1, HBBFT was

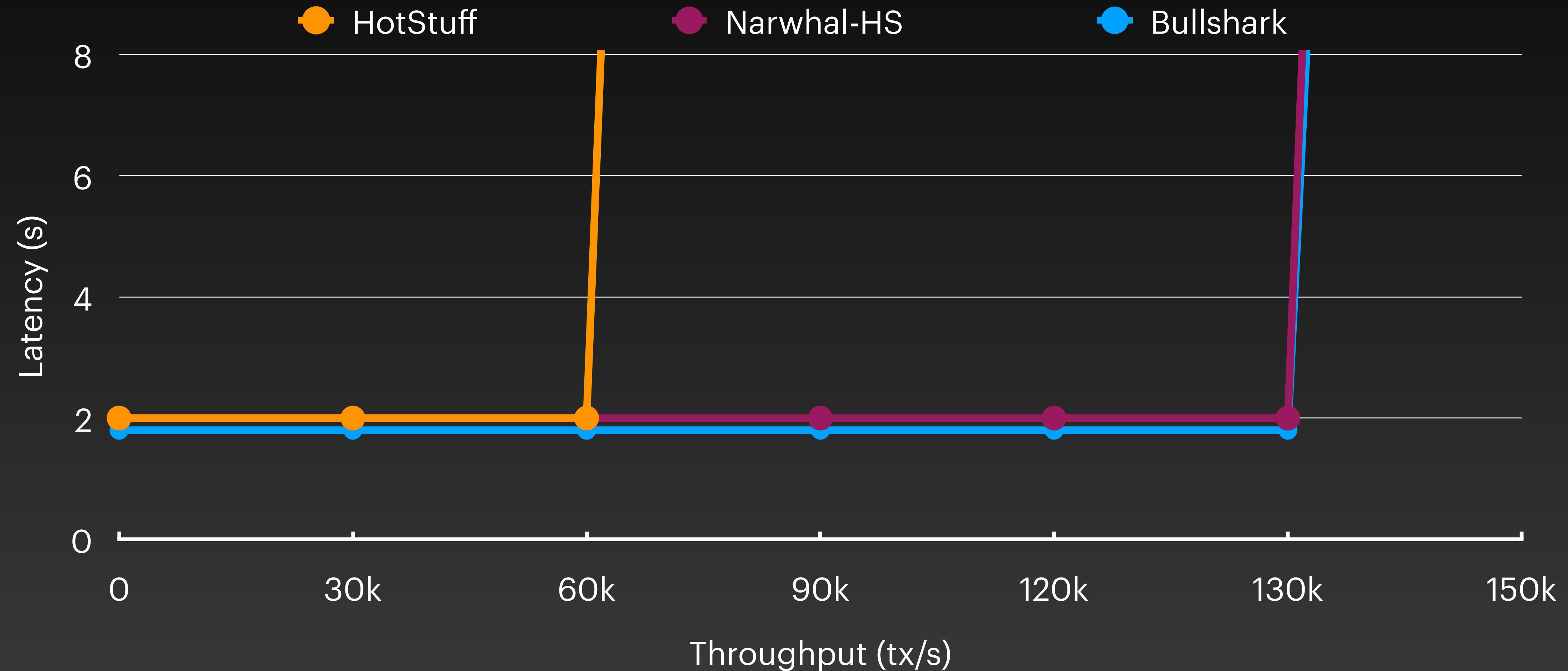
Core

- Hard to make efficient
 - 99% of the code

Consensus

- Error prone
Isolated, easy to maintain

Performance



Research Questions

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

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

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 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

Sui, 2023

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

Shoal

Shoal: Improving DAG-BFT Latency And Robustness

Alexander Spiegelman
Aptos
Rati Gelashvili
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 (or Blockleaders). Recently, it was proposed as a consensus protocol on top of the Narwhal's DAG that can support over 100K transactions per second. Unfortunately, the high throughput of Bullshark comes with a latency price due to the DAG-based consensus mechanism, which compares 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 replacement, Shoal achieves a round time that is significantly reduced. Moreover, the combination of properties of the DAG construction and the leader replacement mechanism enables the protocol to scale in all but extremely adversarial scenarios in practice: a property we name "prevalent responsiveness" (it strictly subsumes the established and often desired "optimistic responsiveness" property for BFT).

We evaluate 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 execution. We also evaluate the execution 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, 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, 22–24], have been designed to tolerate up to t faulty nodes 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 state-of-the-art blockchain [50, 51, 52, 53, 54, 55, 56, 57, 58, 59] are engaged in a race to deliver a scalable BFT system with the utmost throughput and minimal latency.

Sailfish

Sailfish: Towards Improving the Latency of DAG-based BFT

Nibesh Shrestha
n.shrestha@supraclouds.com
Supra Research
Rohan Shrothium
rohan@kuralabs.xyz
Kuru Labs
Aniket Kate
aniket@purdue.edu
Purdue University / Supra Research
Kartik Nayak
kartik@cs.duke.edu
Duke University

Abstract

Historically, the prevailing belief has been that reducing communication complexity was the key to unlocking high performance, low-latency consensus. However, this did not result in dramatic improvements in the throughput. Instead, it was a shortcoming 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 100 TPS.

A recent breakthrough, however, stemmed from the realization that data dissemination is the primary bottleneck for leader-based protocols, and it can be lifted by utilizing a novel approach. Specifically, a leader-based protocol such as RoundRobin [47, 48] can achieve a high throughput by separating data dissemination from the consensus logic and proposed an architecture where all validators simultaneously process data, while only one validator performs a smaller number of leader-related tasks. 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 for handling more users. The paper proposes Sailfish, a leader-based protocol 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 compared to 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 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 t up to $\frac{n}{2}$ of these parties are Byzantine (arbitrarily malicious). Over the last two decades, significant progress has been made in improving the efficiency of these consensus protocols. In particular, non-equivocable round-based directed acyclic graph (DAG), a concept initially introduced by Aleph [21]. In this model, each validation step is represented by a vertex and each vertex has up to n vertices in the preceding round. Each vertex is disseminated via an efficient reliable broadcast implementation, ensuring that malicious validators cannot distinguish different vertices to different validators within the same round. While serving as a building block for consensus, the DAG is also used 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 state-of-the-art blockchain [50, 51, 52, 53, 54, 55, 56, 57, 58, 59] are engaged in a race to deliver a scalable BFT system with the utmost throughput and minimal latency.

Supporting a leader vertex in each round, State-of-the-art consensus protocols can either use a leader or a more decentralized approach. In the former, each party can elect a potential DAG vertex containing transactions and edges pointing to vertices from previous rounds. These protocols rely on committing a designated "leader vertex" and order other non-leader vertices around it. This leads to a high latency overhead since it is not clear which leaders are designated and how fast the leader vertices are committed directly influences the commit latency.

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, the miners must ensure that the DAG is partially synchronous, existing SMR protocols can commit with a latency overhead of δ (where δ represents the round trip time between miners) and achieve linear communication complexity [17, 18] under optimistic conditions (such as honest leader).

Most of these protocols depend on a designated leader vertex that is mainly responsible for proposing transactions and driving the protocol forward while other parties

2023

arXiv:2205.09174v6 [cs.DC] 22 Sep 2023

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

Abstract

Cordial Miners are a family of efficient Byzantine Atomic Broadcast protocols, with instances of latency 3 rounds. Our main contribution is to achieve the lowest bounds of latency for consensus protocols that are robust to both Byzantine and crash failures. Cordial Miners also achieve high resource efficiency and censorship resistance. MYSTICETI-C achieves its latency improvement by avoiding the need for a consensus DAG, and instead uses a novel commit rule such that every block can be committed without delay, resulting in a fast commit path in the presence of both Byzantine and under crash failures. We further extend MYSTICETI-C to MYSTICETI-FPC, which incorporates a fast commit path that allows for fast consensus even in the presence of both Byzantine and crash failures. Finally, we propose MYSTICETI-FPC, a fast and efficient consensus protocol that minimizes the number of signatures and messages by weaving the fast path and slow path together. Our results show that our approach subsequently result in better performance. We prove the safety and liveness of our protocols in the presence of both Byzantine and crash failures. We evaluate the both MYSTICETI and fast path protocols to demonstrate their low latency and resource efficiency, making them suitable for real-world applications.

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 Tel Aviv Hashiri Fujwars Cyber-Security Research Center for providing a research grant. Ehud Shapiro is the incumbent of The Harry Weinrebe Professorial Chair of Computer Science and Biology at the Weizmann Institute.

1

Introduction

Several recent blockchains, such as Sui [67], [12], have adopted DAG-based consensus [31, 55, 56, 54, 50, 19, 12, 58, 44]. By design, these consensus protocols scale well in terms of throughput, with a performance of 100K TPS of transaction processing per second (TPS) and less than 10ms latency [23]. This, however, comes at a high latency of around 2-3 seconds, which can hinder user experience and prevent low-latency applications.

MYSTICETI-C: A family of uncertified DAG-Certified DAGs [14], where each vertex is delivered through consistent propagation [14], have high latency for three main reasons: (1) the consensus process requires multiple rounds to reach a consensus between validators; (2) the signatures and re-broadcast certificates. This leads to higher latency than traditional consensus protocols [31, 64], [15]; (2) blocks commit on a "per block" basis, which is slow and limits the throughput of the system. This is because each block needs to wait for the previous block to finish increasing the latency of transactions proposed by the block. This phenomena is similar to committing big batches of $2^j + 1$ blocks. Finally, (3) since all certified blocks need to

Mysticeti

MYSTICETI: Reaching the Latency Limits with Uncertified DAGs

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

¹Cornell Tech, ^{1‡}C3, ¹Mythen 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 for consensus protocols that are robust to both Byzantine and crash failures. MYSTICETI-C achieves its latency improvement by avoiding the need for a consensus DAG, and instead uses a novel commit rule such that every block can be committed without delay, resulting in a fast commit path in the presence of both Byzantine and under crash failures. We further extend MYSTICETI-C to MYSTICETI-FPC, which incorporates a fast commit path that allows for fast consensus even in the presence of both Byzantine and crash failures. Finally, we propose MYSTICETI-FPC, a fast and efficient consensus protocol that minimizes the number of signatures and messages by weaving the fast path and slow path together. Our results show that our approach subsequently result in better performance. We prove the safety and liveness of our protocols in the presence of both Byzantine and crash failures. We evaluate the both MYSTICETI and fast path protocols to demonstrate their low latency and resource efficiency, making them suitable for real-world applications.

Related Version: MYSTICETI: Reaching the Latency Limits with Uncertified DAGs

Full Version: <https://arxiv.org/abs/2205.09174>

Acknowledgments: Kushal Babel is grateful to the Azrieli Foundation for the award of an Azrieli Fellowship, and to the Tel Aviv Hashiri Fujwars Cyber-Security Research Center for providing a research grant. Ehud Shapiro is the incumbent of The Harry Weinrebe Professorial Chair of Computer Science and Biology at the Weizmann Institute.

1

I. INTRODUCTION

This paper addresses the problem of ordering transactions in a permissioned Byzantine distributed system, also known as *Byzantine Atomic Broadcast (BAB)*. It 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, the miners must ensure that the DAG is partially synchronous, existing SMR protocols can commit with a latency overhead of δ (where δ represents the round trip time between miners) and achieve linear communication complexity [17, 18] under optimistic conditions (such as honest leader).

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.

This comes at the cost of significantly higher latency. For example, in the case of Bullshark [33], which requires only 3 messages 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 signatures and re-broadcast certificates. This leads to higher latency than traditional consensus protocols [31, 64], [15]. This burden is particularly heavy for a crash-recovered validator that typically needs to verify thousands of signatures before it can commit a proposal at each round. Signature verification seems to have the benefit that in adversarial cases one can advance the DAG without needing to synchronize the full block, provided that the adversary does not exploit this benefit. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This paper proposes MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols allowing to safely commit distributed transactions in a Byzantine setting that focuses on low-latency and low-CPU overhead. MYSTICETI-C is the first DAG-based consensus protocol based on a threshold logical clock [29] that achieves consensus faster than any other DAG-based consensus protocol based on a consensus DAG [31].

Fig. 1: P95 latency of a major blockchain switching from Bullshark (190ms) to MYSTICETI-C (390ms) consensus on 106 independently run validators

be signed by all validators. We evaluate the both MYSTICETI and fast path protocols to demonstrate their low latency and resource efficiency, making them suitable for real-world applications.

The full paper, including experimental deployment details, shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols that are robust to both Byzantine and crash failures. We evaluate the performance of MYSTICETI-C, which shows that this benefit is negated when needing to execute the committed transactions. As a result, the certification benefits of DAG-based consensus are not fully realized if it is not used for the common case of powering a State Machine Replication system (e.g., a blockchain).

This work presents MYSTICETI, a family of DAG-based consensus protocols

Discussion



Shoal/shoal++

- Low latency
- Easier synchroniser
- Leverage existing code

Sailfish/BBCA

- Lower latency
- Easy synchroniser
- Flexible

CM/Mysticeti

- Even Lower latency
- Graceful crash faults
- Simpler, less CPU

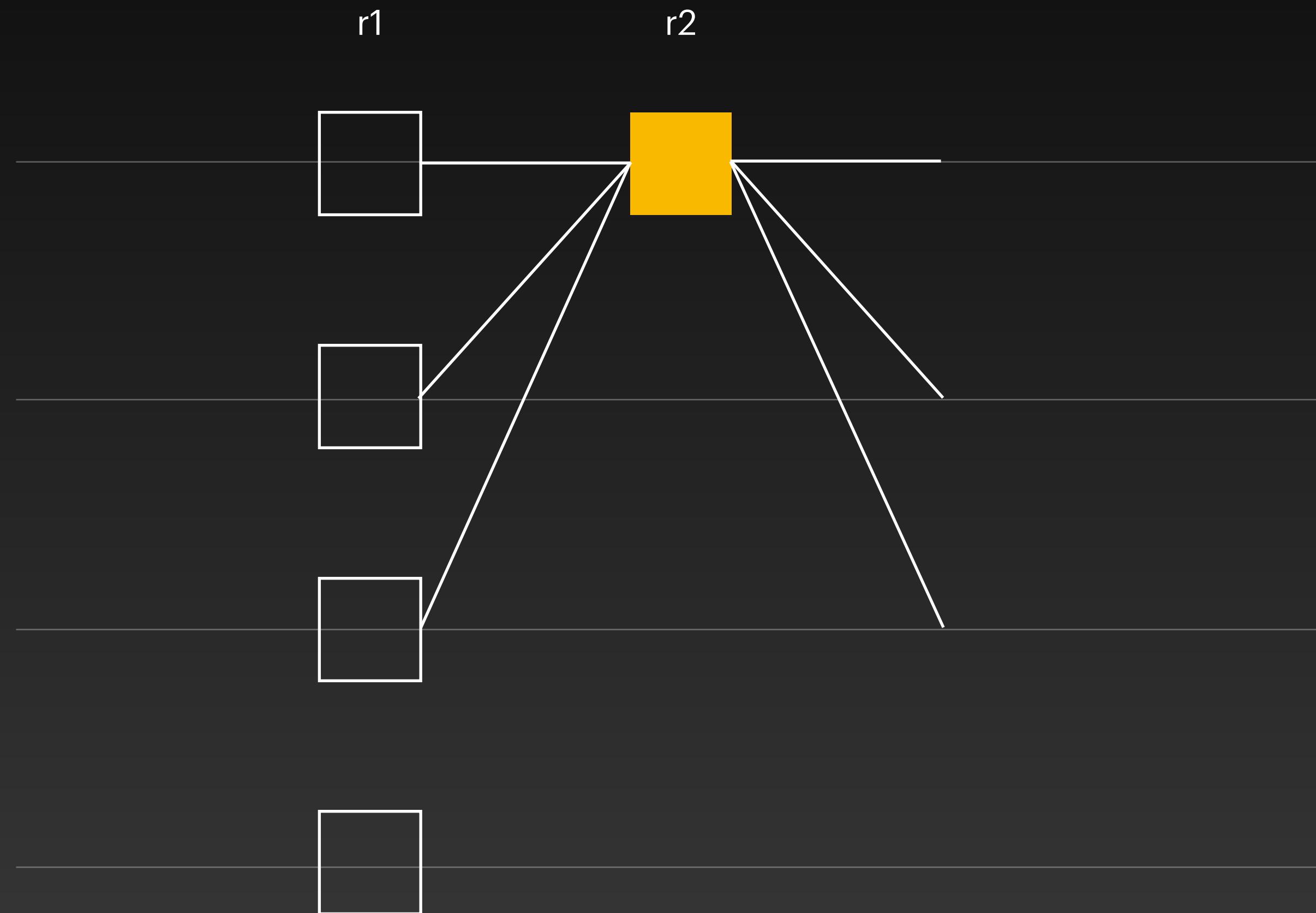
Research Questions

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

Lessons Learned

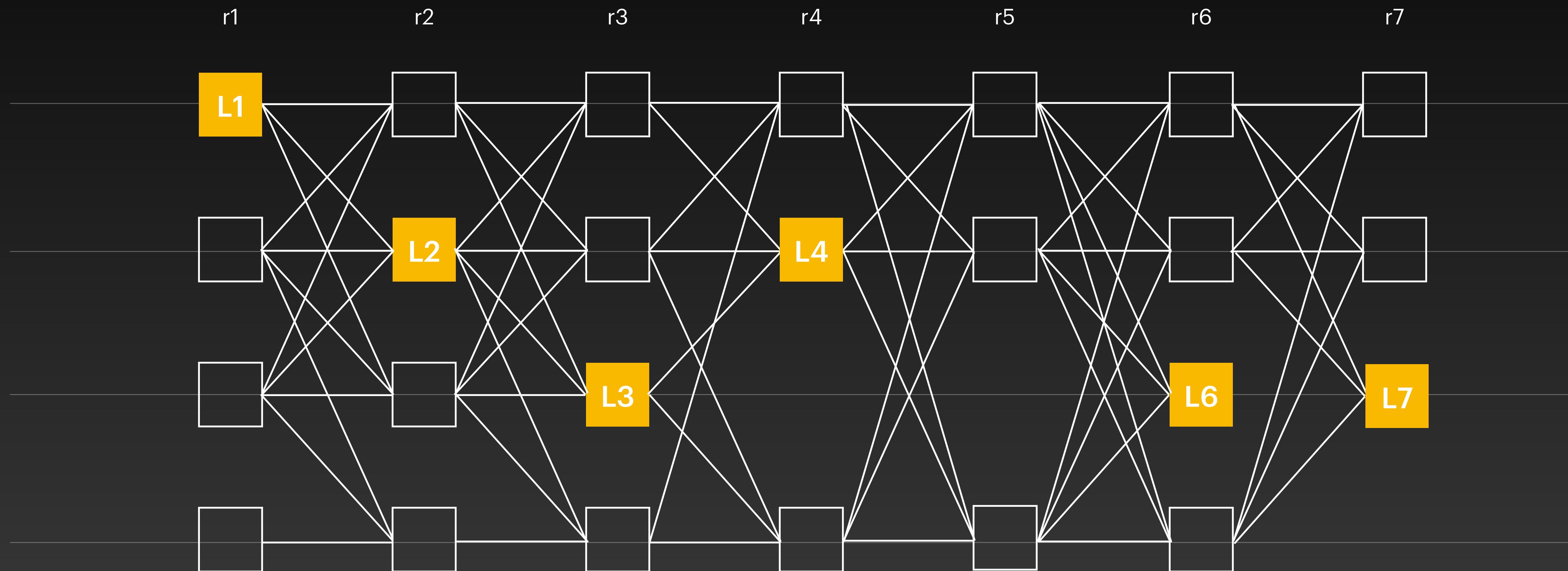
1. Modularisation is a design strategy
2. Tasks-threads relationship
3. Benchmark early
4. Codesign with mem. and storage
5. Core is hard, consensus is easy
6. Epoch change is not an add-on

Uncertified DAG

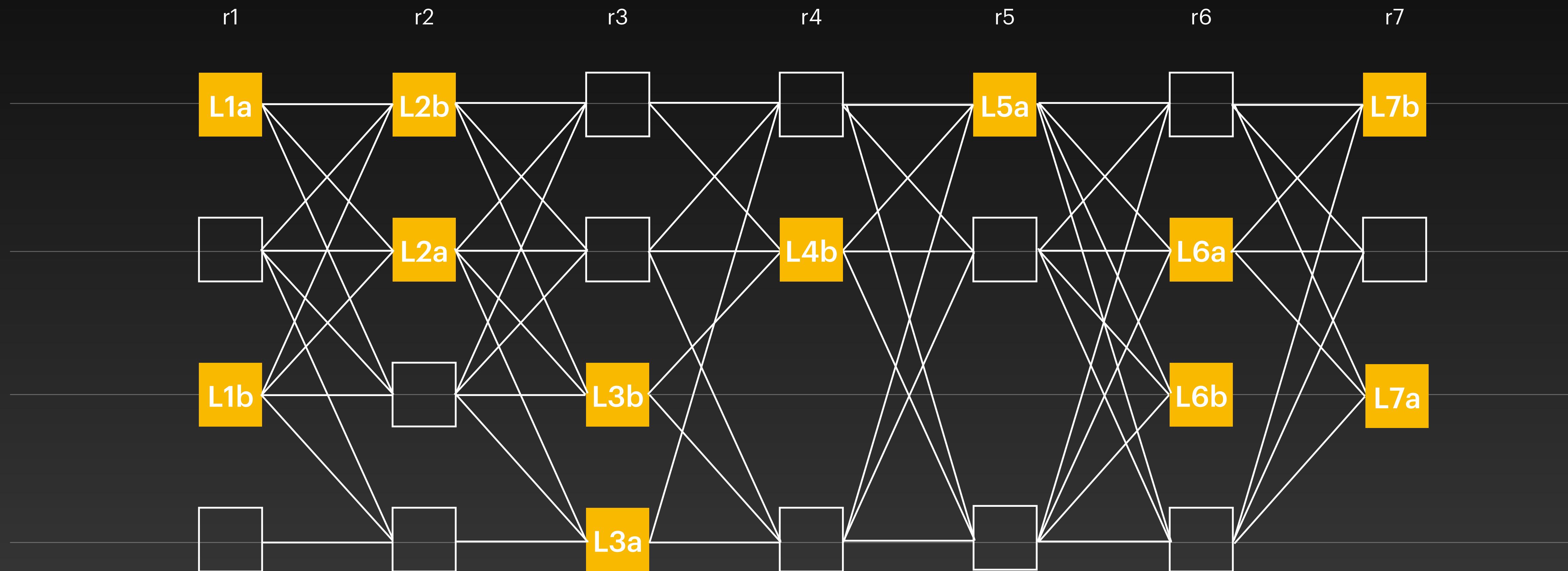


- Round number
- Author
- Payload (transactions)
- Signature

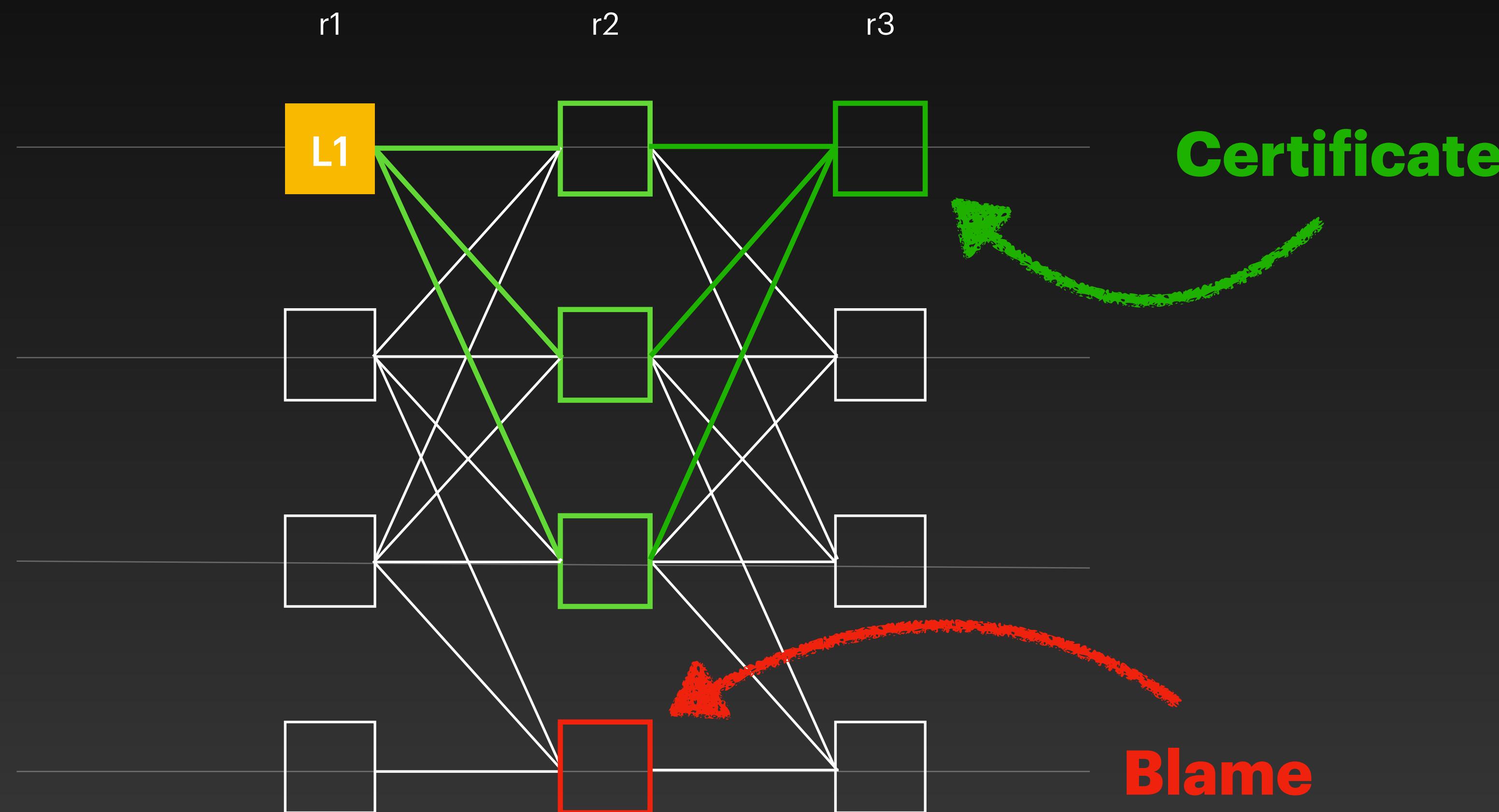
Uncertified DAG



Uncertified DAG



Interpreting DAG Patterns



Direct Decision Rule

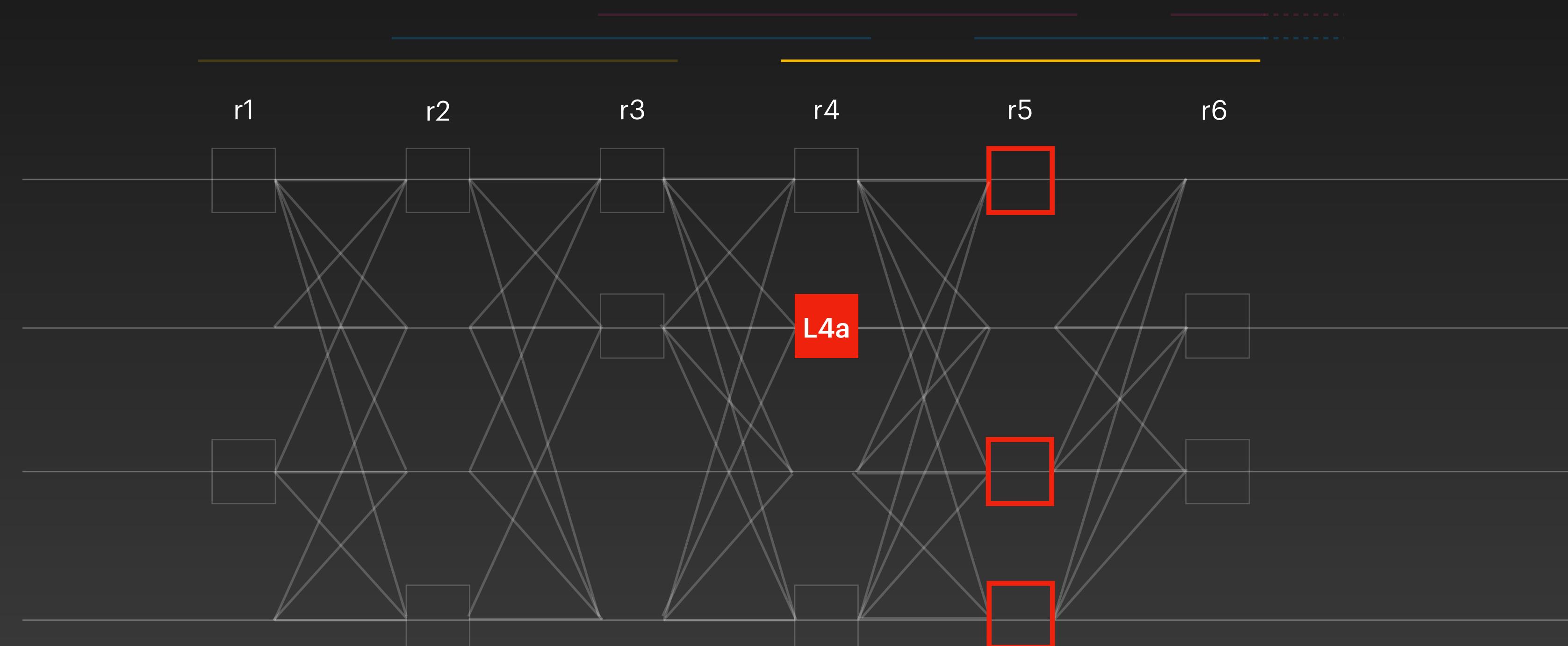
On each leader starting from highest round:

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

Direct Decision Rule

On each leader starting from highest round:

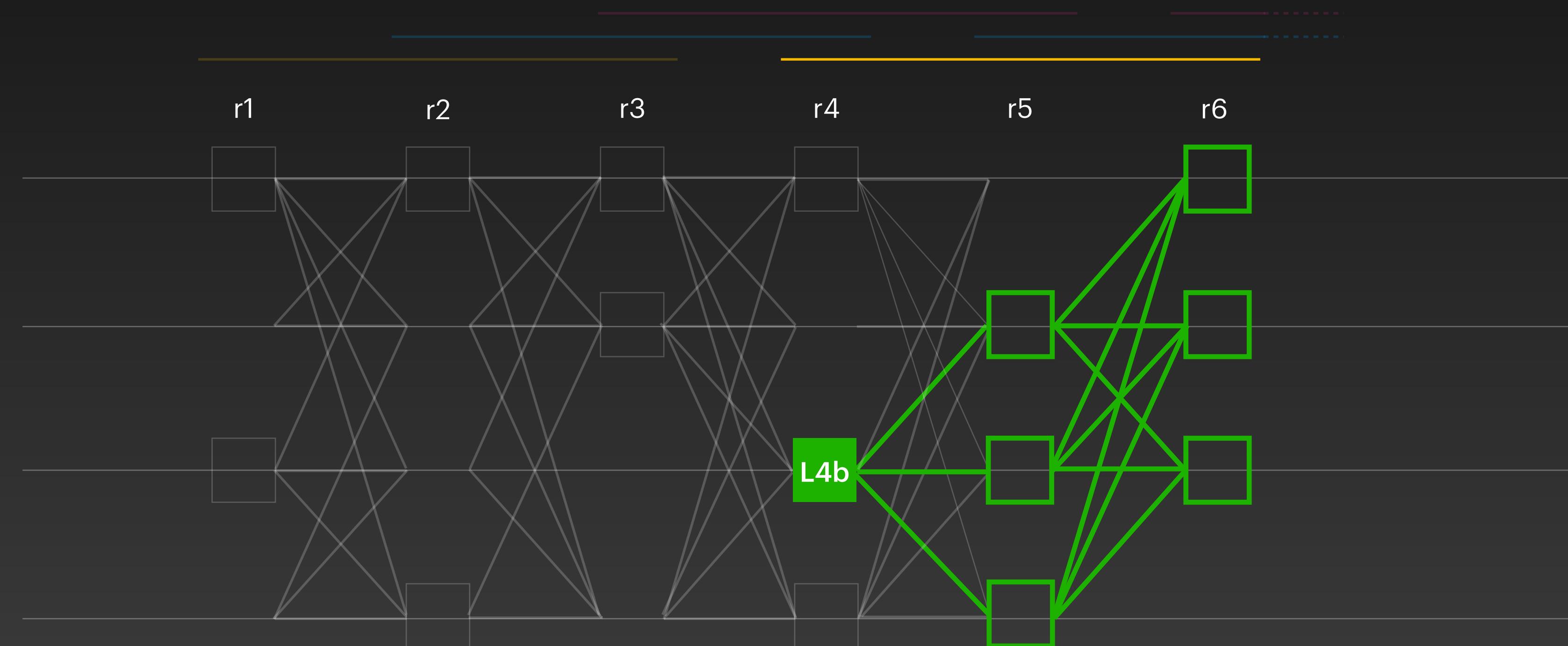
- **Skip** if $2f+1$ blames
- **Commit** if $2f+1$ certificates
- **Undecided** otherwise



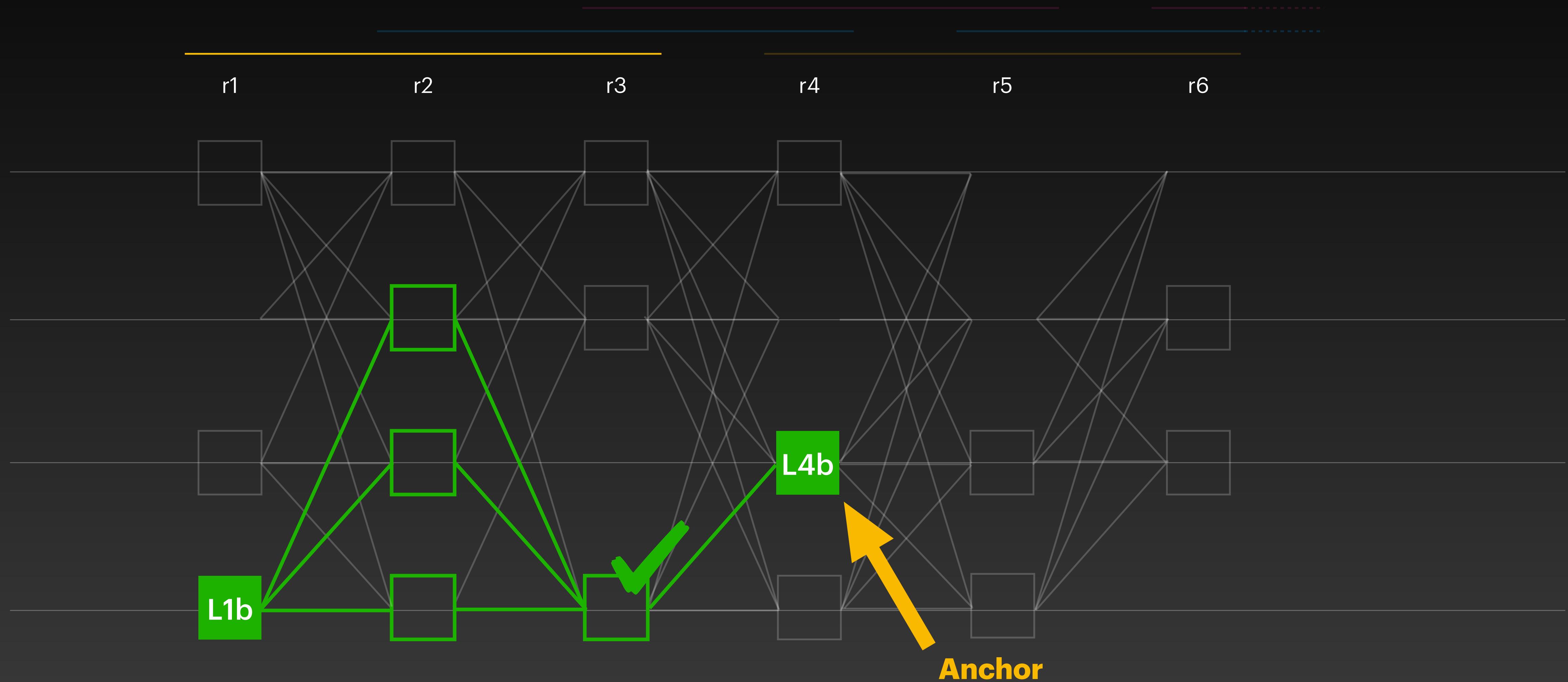
Direct Decision Rule

On each leader starting from highest round:

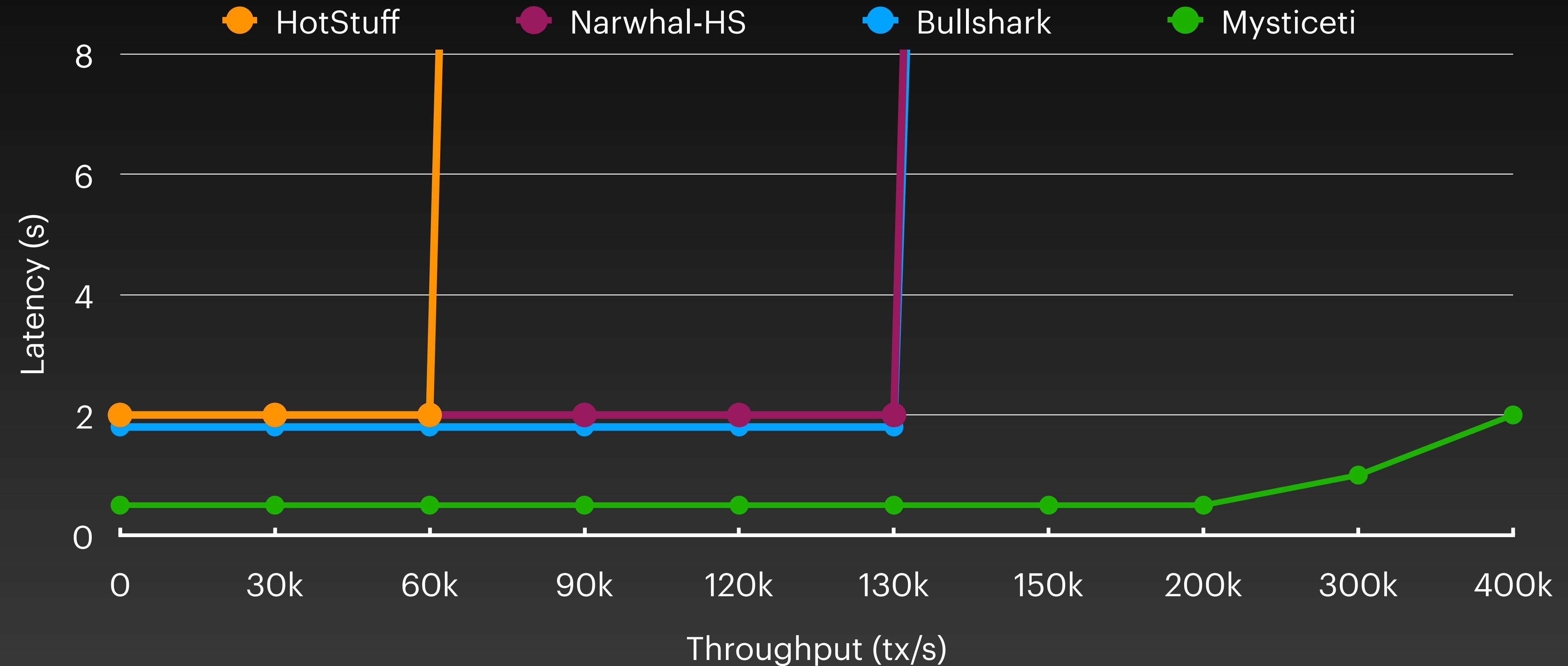
- **Skip** if $2f+1$ blames
- **Commit** if $2f+1$ certificates
- **Undecided** otherwise



Apply Indirect 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

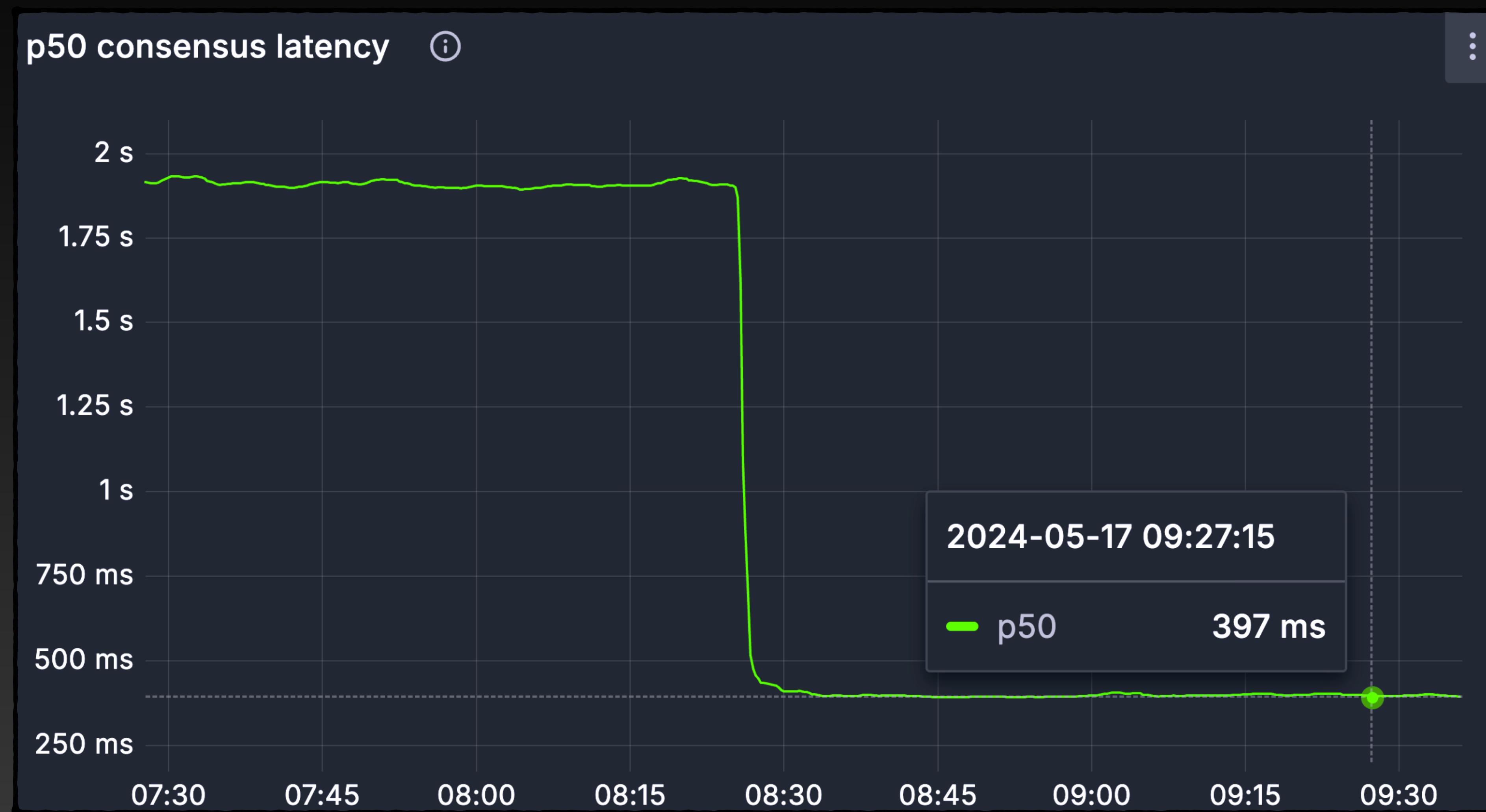
1. Modularisation is a design strategy
2. Tasks-threads relationship
3. Benchmark early
4. Codesign with mem. and storage
5. Core is hard, consensus is easy
6. Epoch change is not an add-on

Testing Strategy



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

The Sui Mainnet



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 



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 

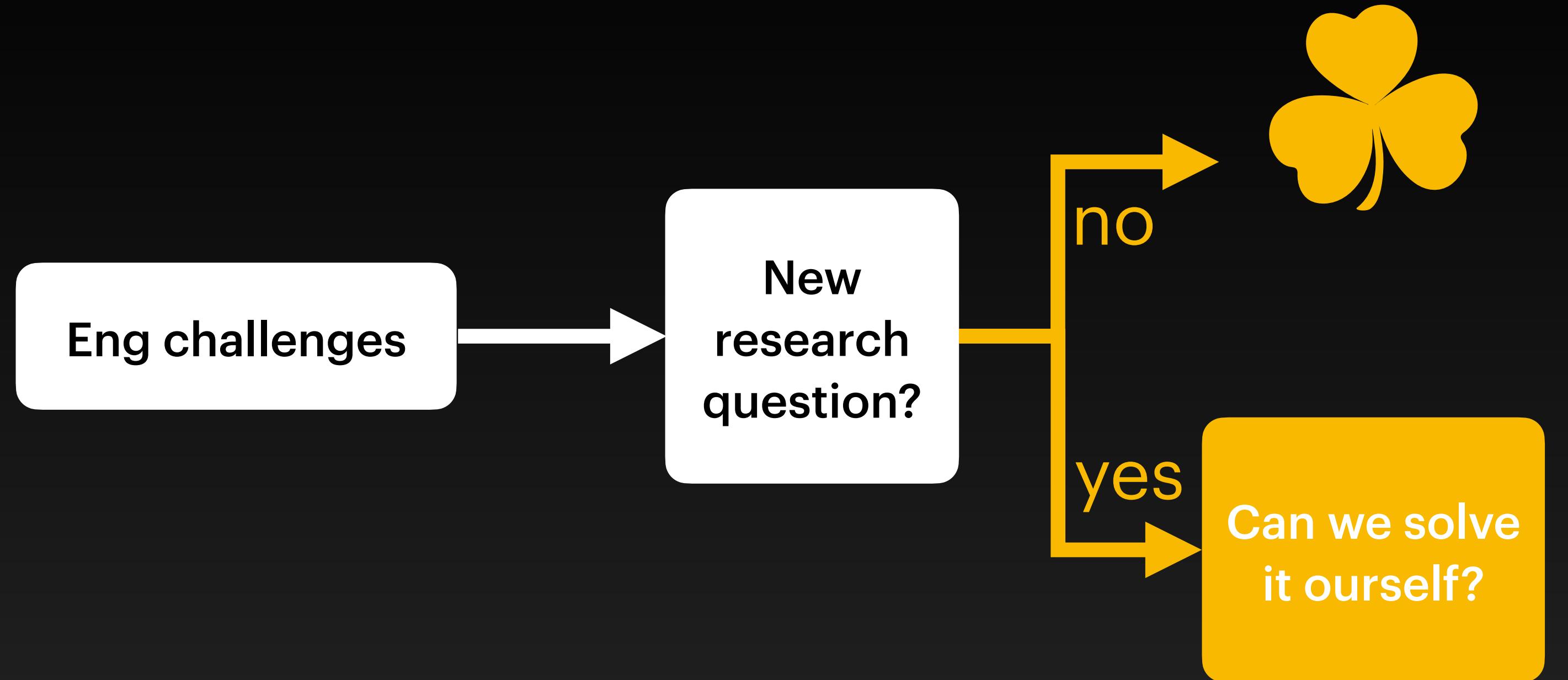
 **Dmitri Perelman** Oct 18th at 7:36 AM
You're in the invite list!

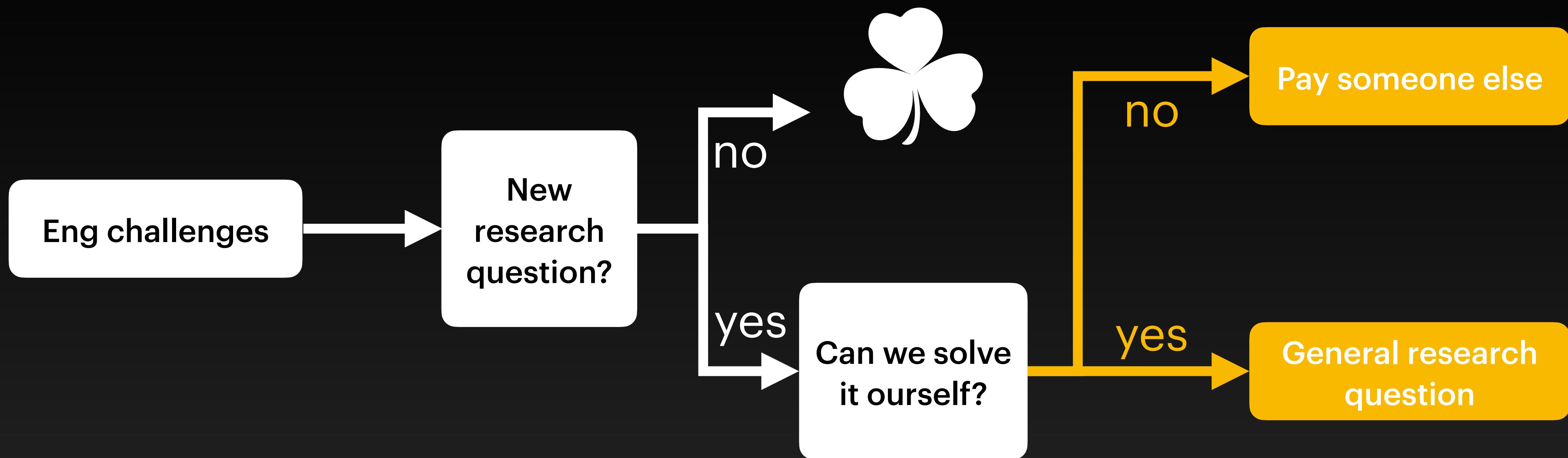
 1 

Eng challenges



New
research
question?

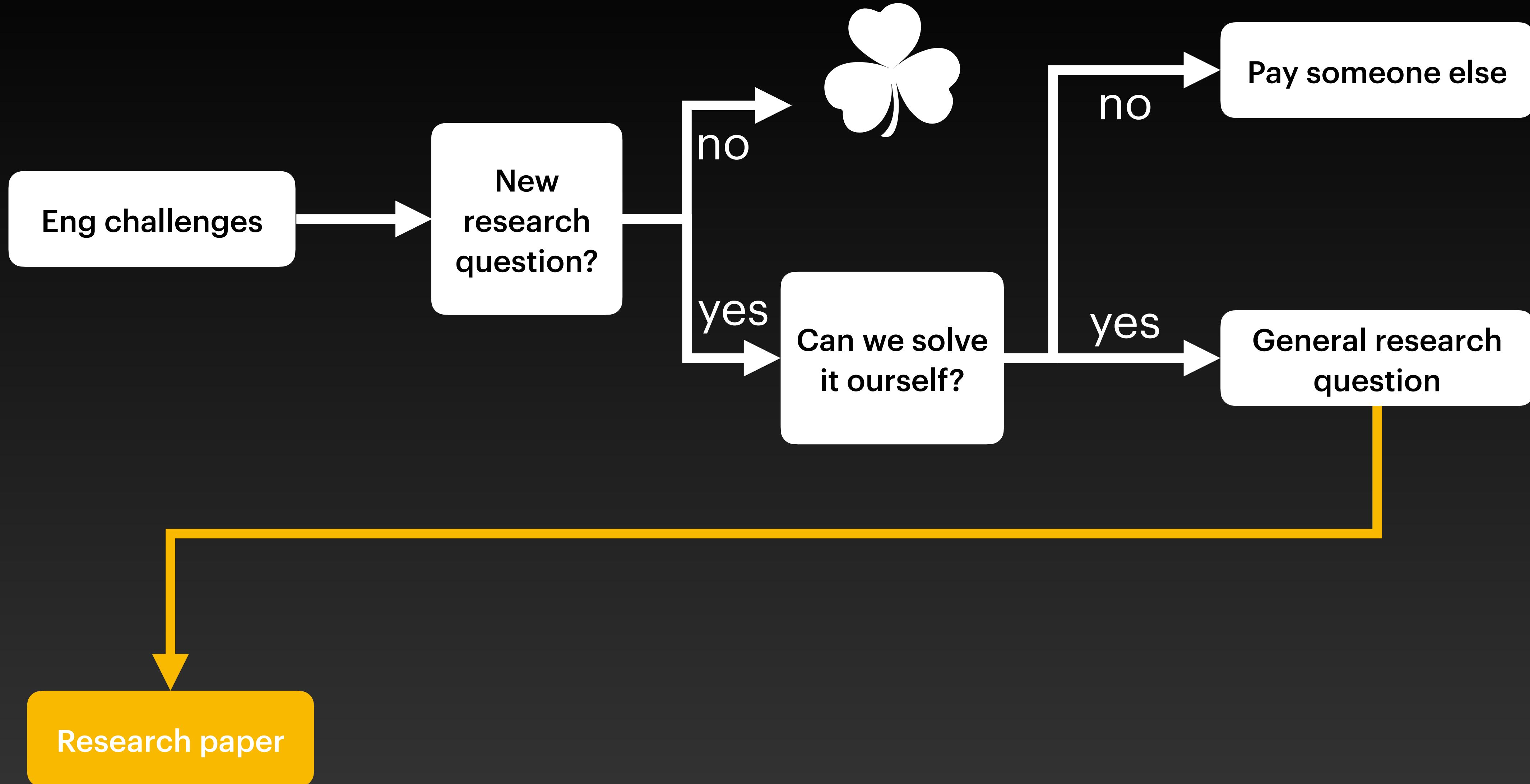


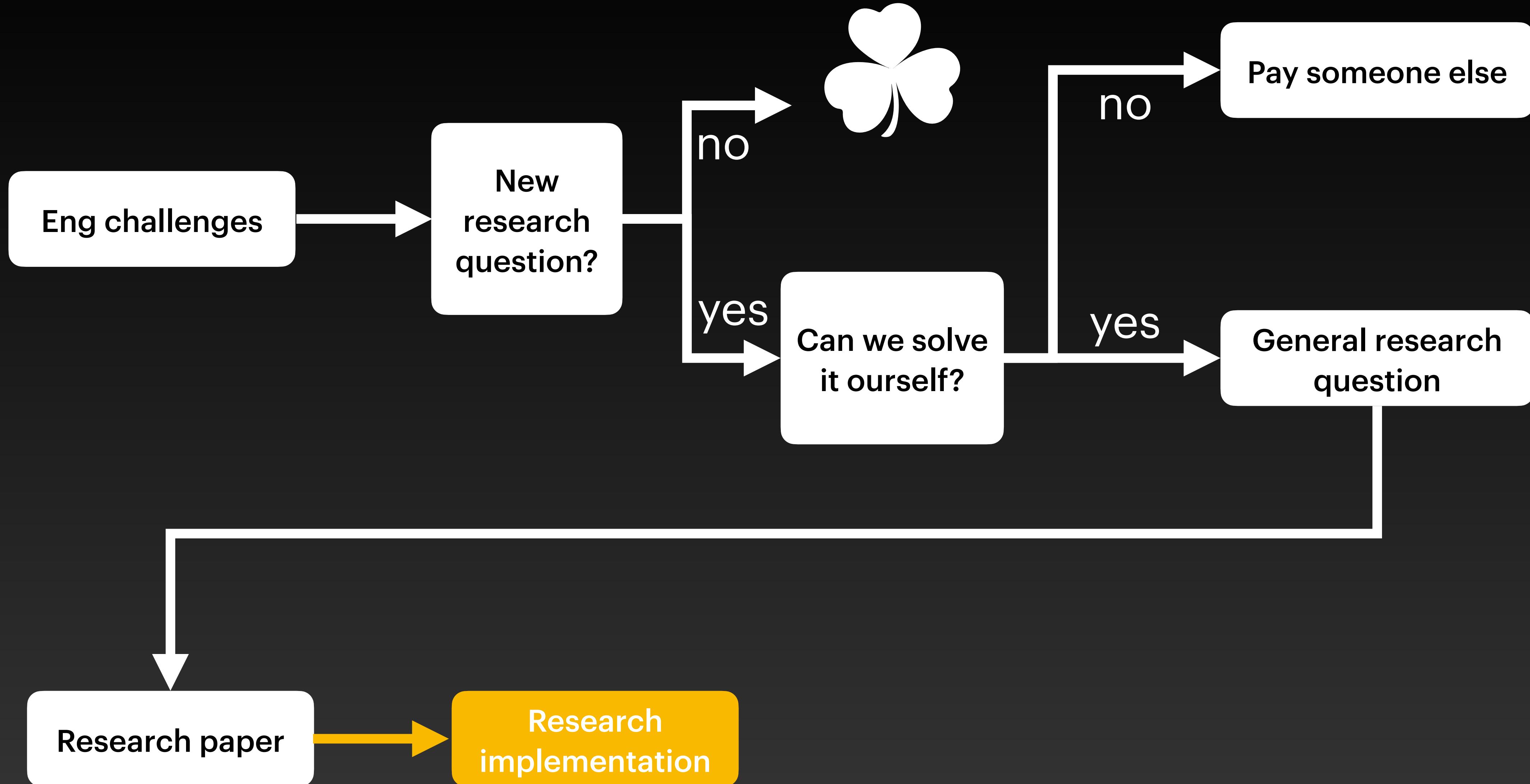


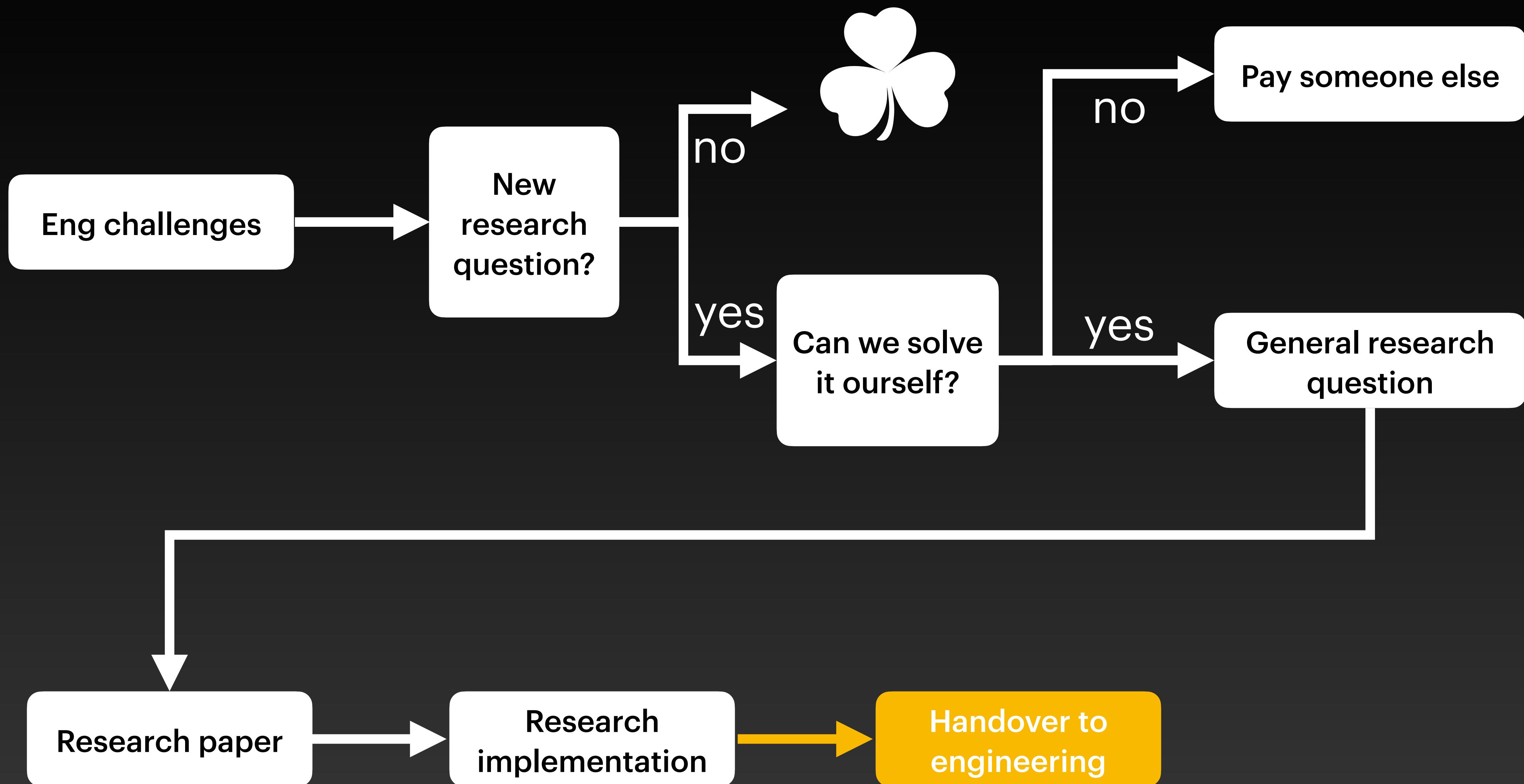
Research Gifts

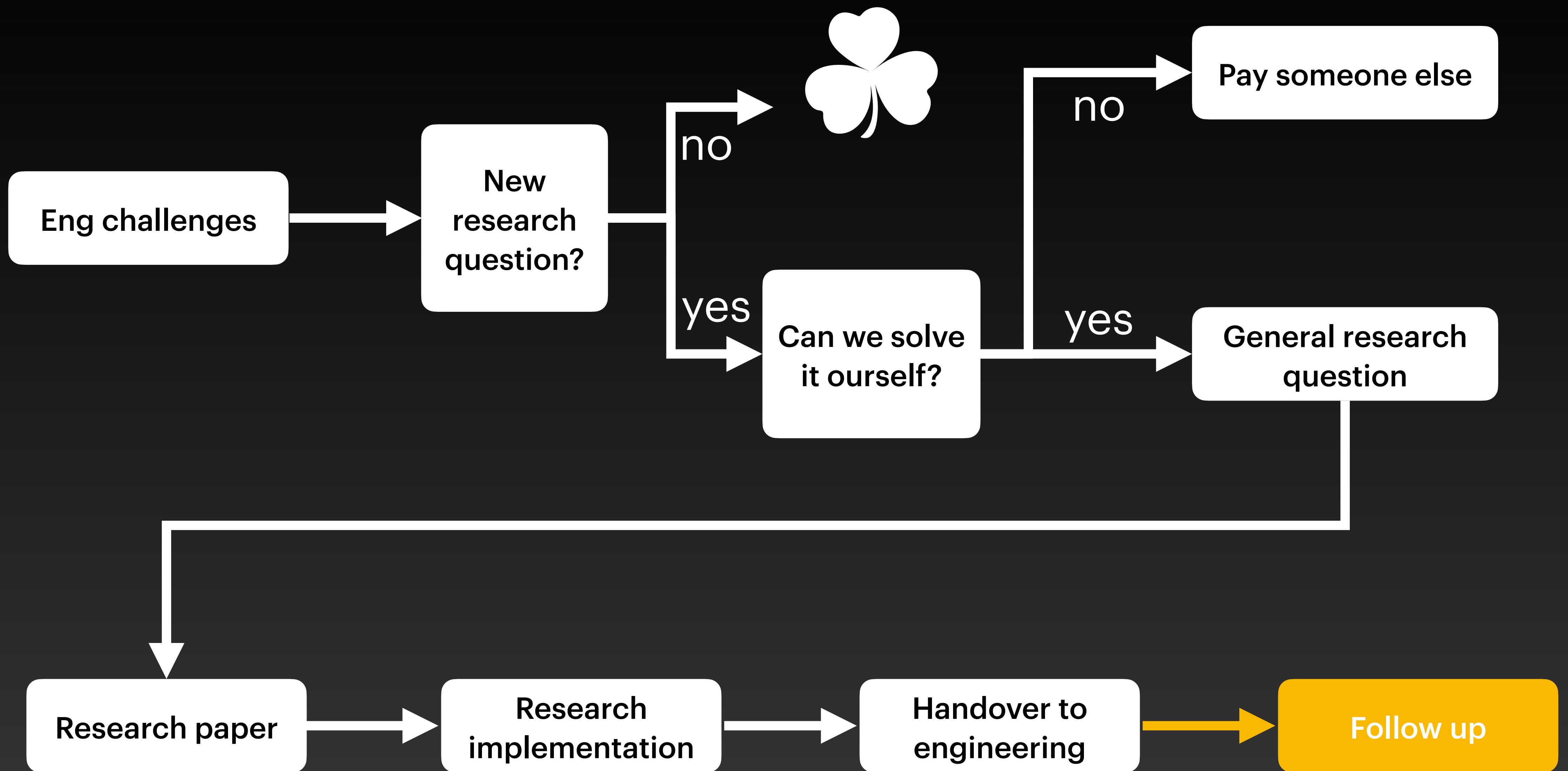


(please keep it short)









[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 • Adopted 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>

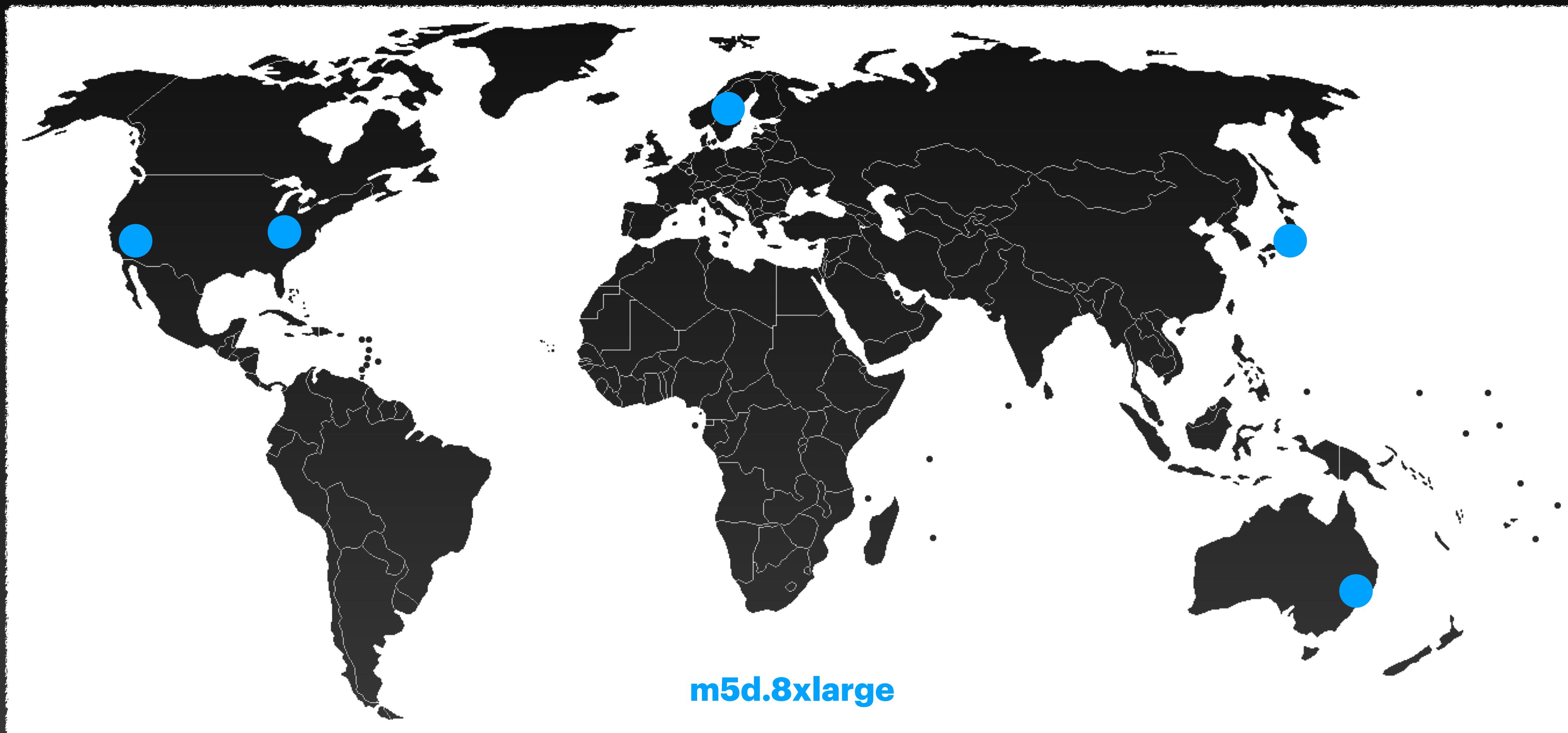
Implementation

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

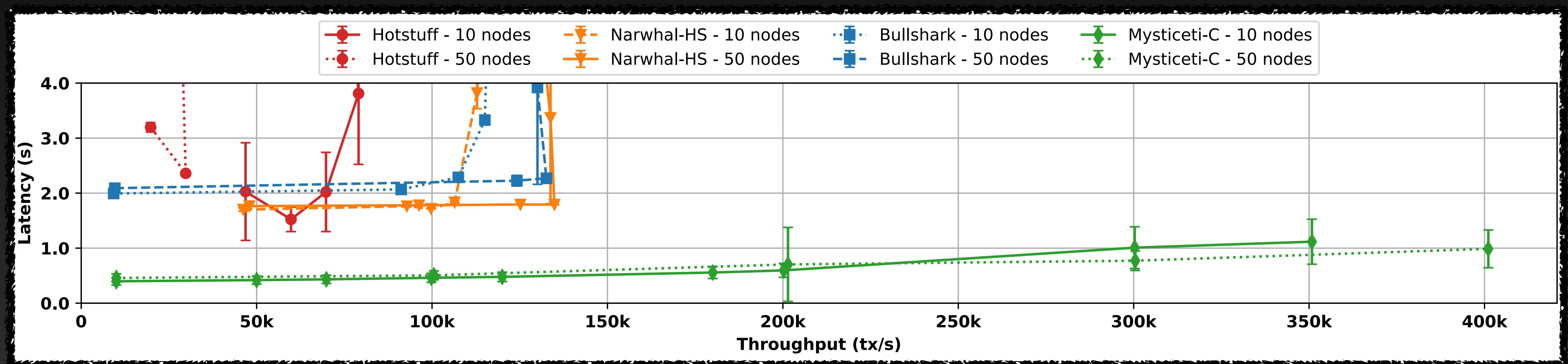
<https://github.com/mystenlabs/mysticeti>

Evaluation

Experimental setup on AWS



Prototype Benchmarks



Prototype Benchmarks

