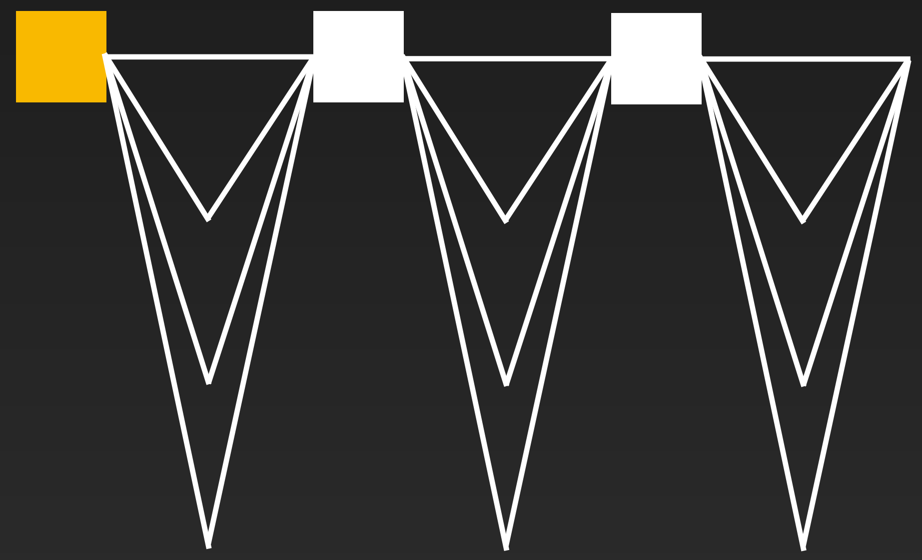
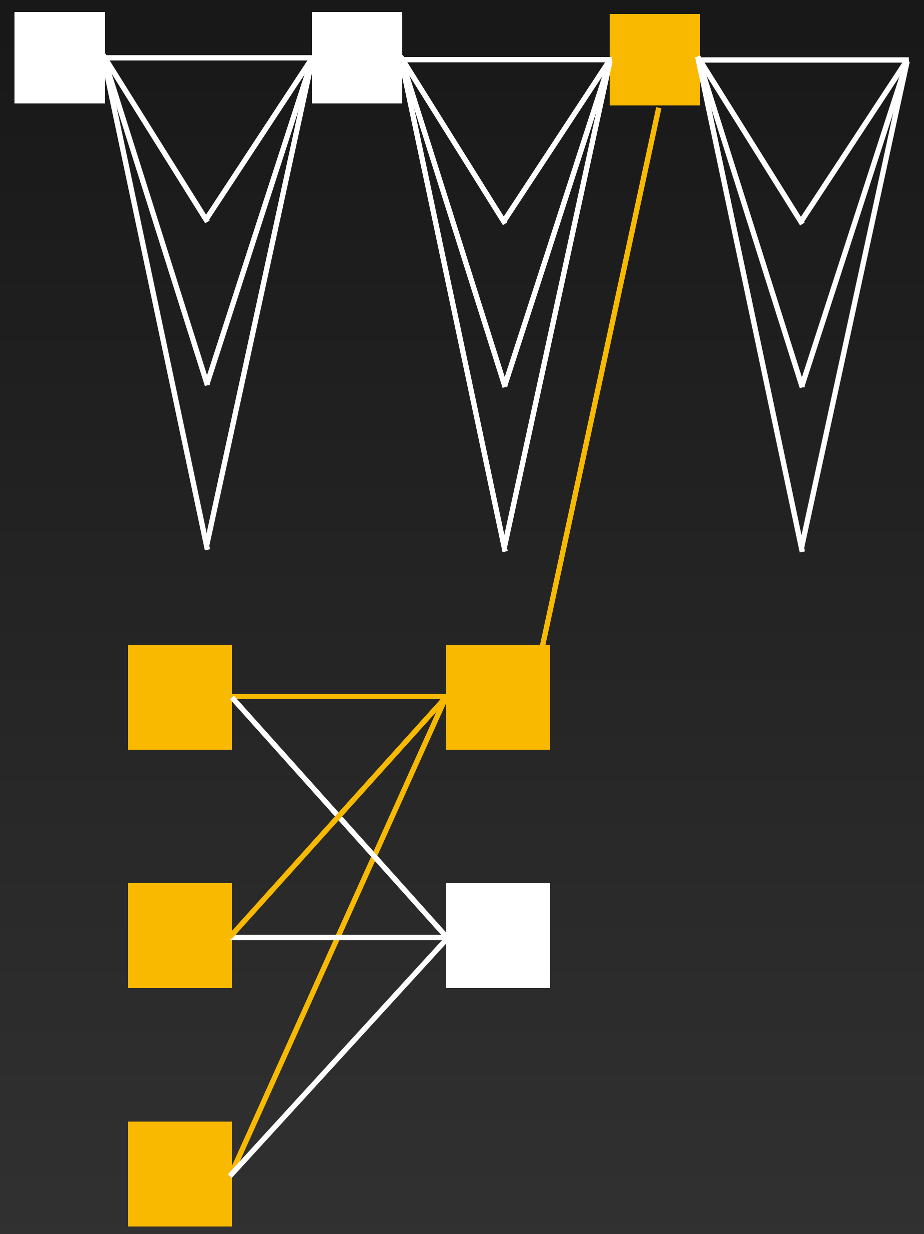
# The Evolution of Sui

From Academic Paper to Mainnet

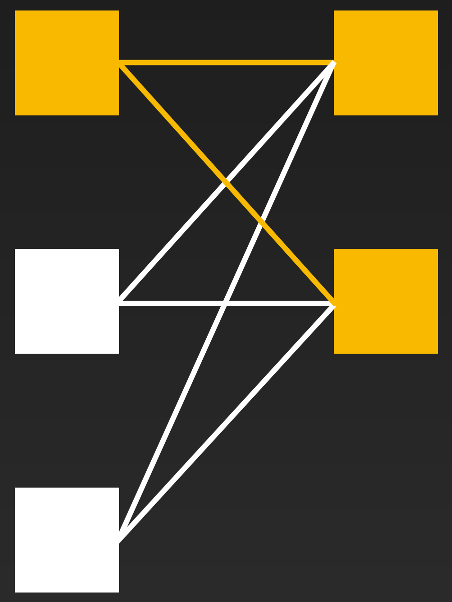
**2019** libra → Sui **2024**



**HotStuff**



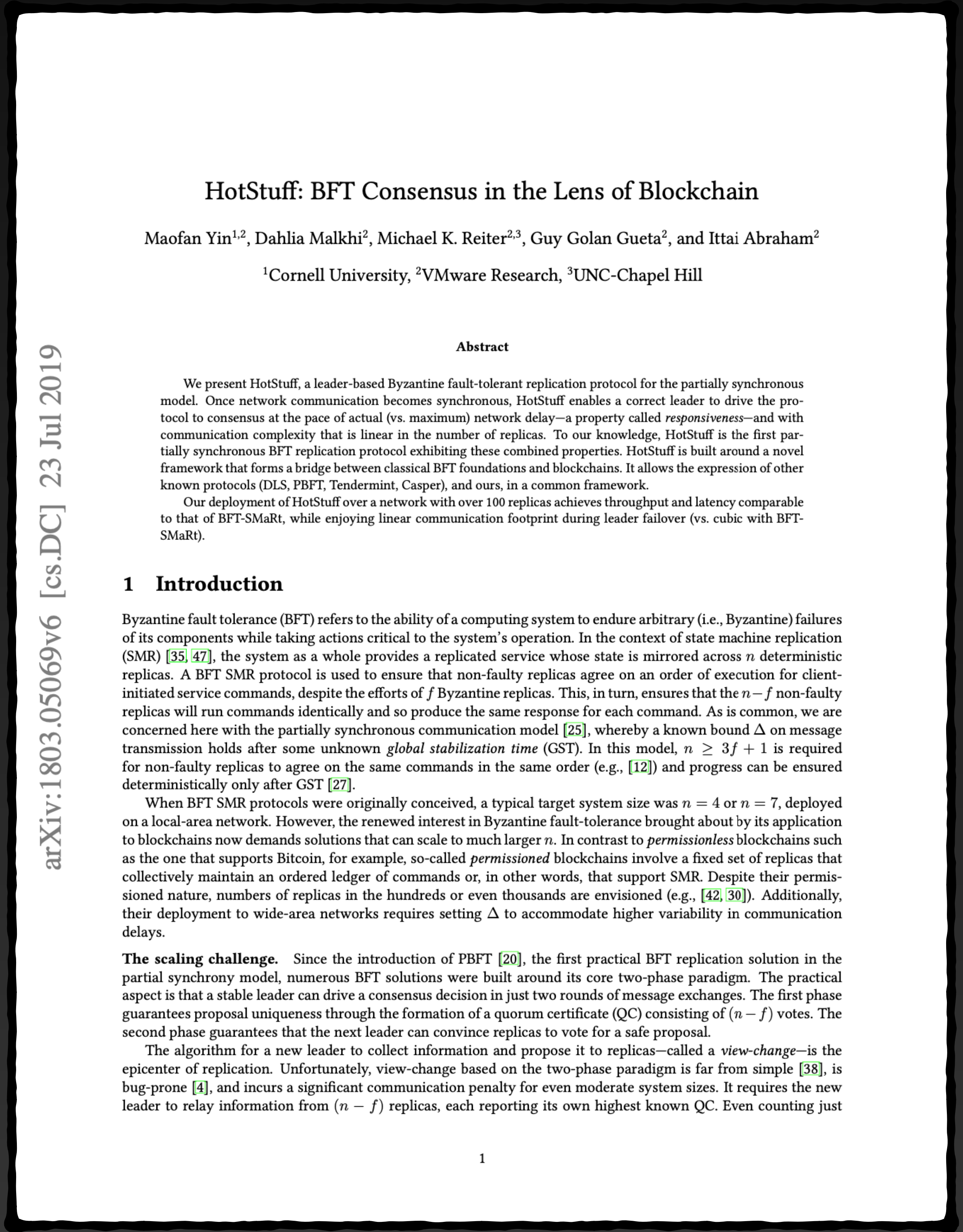
**HotStuff + Mempool**



**Bullshark, Mysticeti**

# Libra, 2019

## HotStuff



HotStuff: BFT Consensus in the Lens of Blockchain

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

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

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

?

## HashGraph

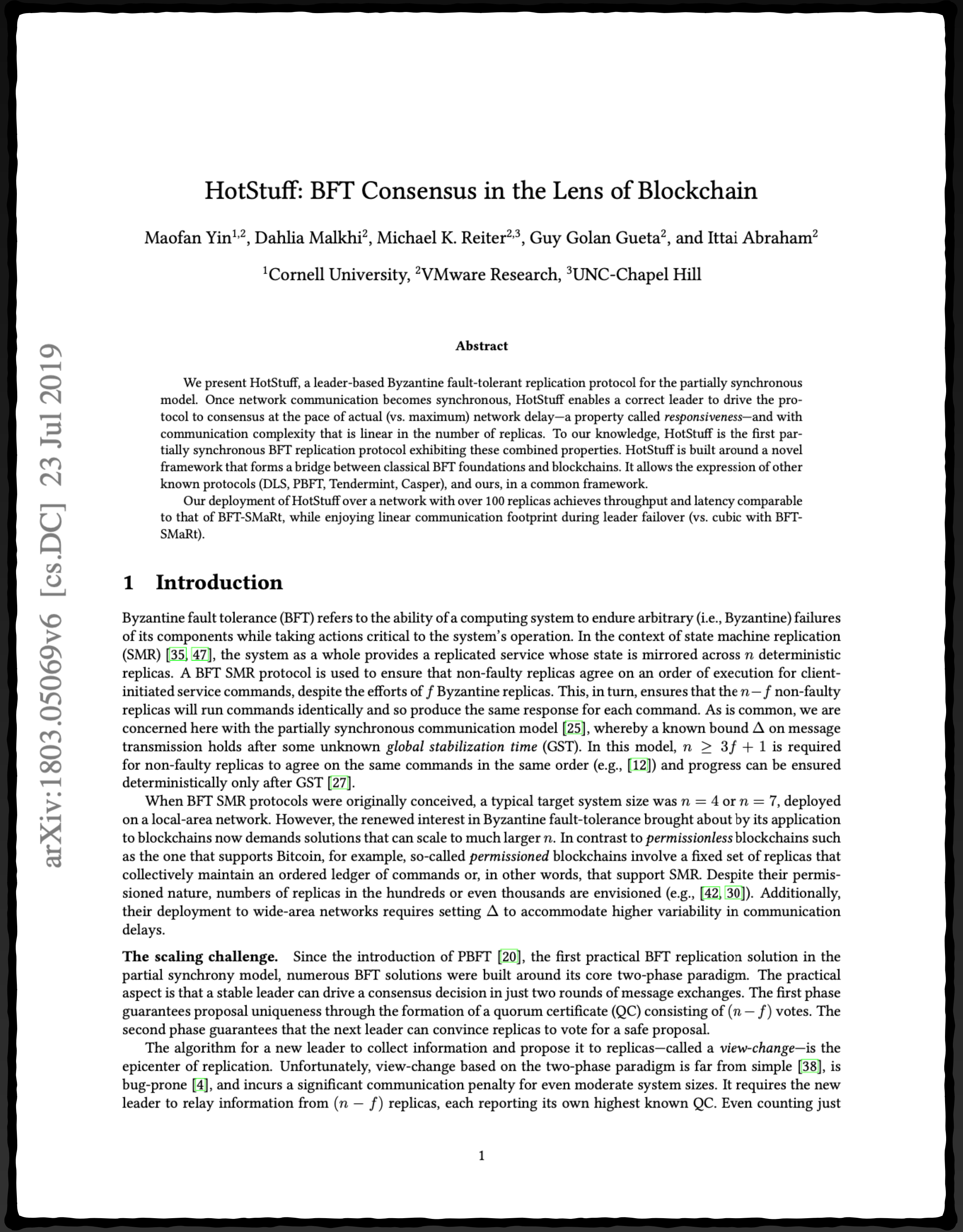
Verifying the Hashgraph Consensus Algorithm

Karl Crary

Carnegie Mellon University

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

# Libra, 2019



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

HotStuff: BFT Consensus in the Lens of Blockchain

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

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

**Abstract**

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

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

**1 Introduction**

Byzantine fault tolerance (BFT) refers to the ability of a computing system to endure arbitrary (i.e., Byzantine) failures of its components while taking actions critical to the system's operation. In the context of state machine replication (SMR) [35, 47], the system as a whole provides a replicated service whose state is mirrored across $n$ deterministic replicas. A BFT SMR protocol is used to ensure that non-faulty replicas agree on an order of execution for client-initiated service commands, despite the efforts of $f$ Byzantine replicas. This, in turn, ensures that the $n-f$ non-faulty replicas will run commands identically and so produce the same response for each command. As is common, we are concerned here with the partially synchronous communication model [25], whereby a known bound $\Delta$ 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 $\Delta$ to accommodate higher variability in communication delays.

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

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

1

## HotStuff

✓ Linear

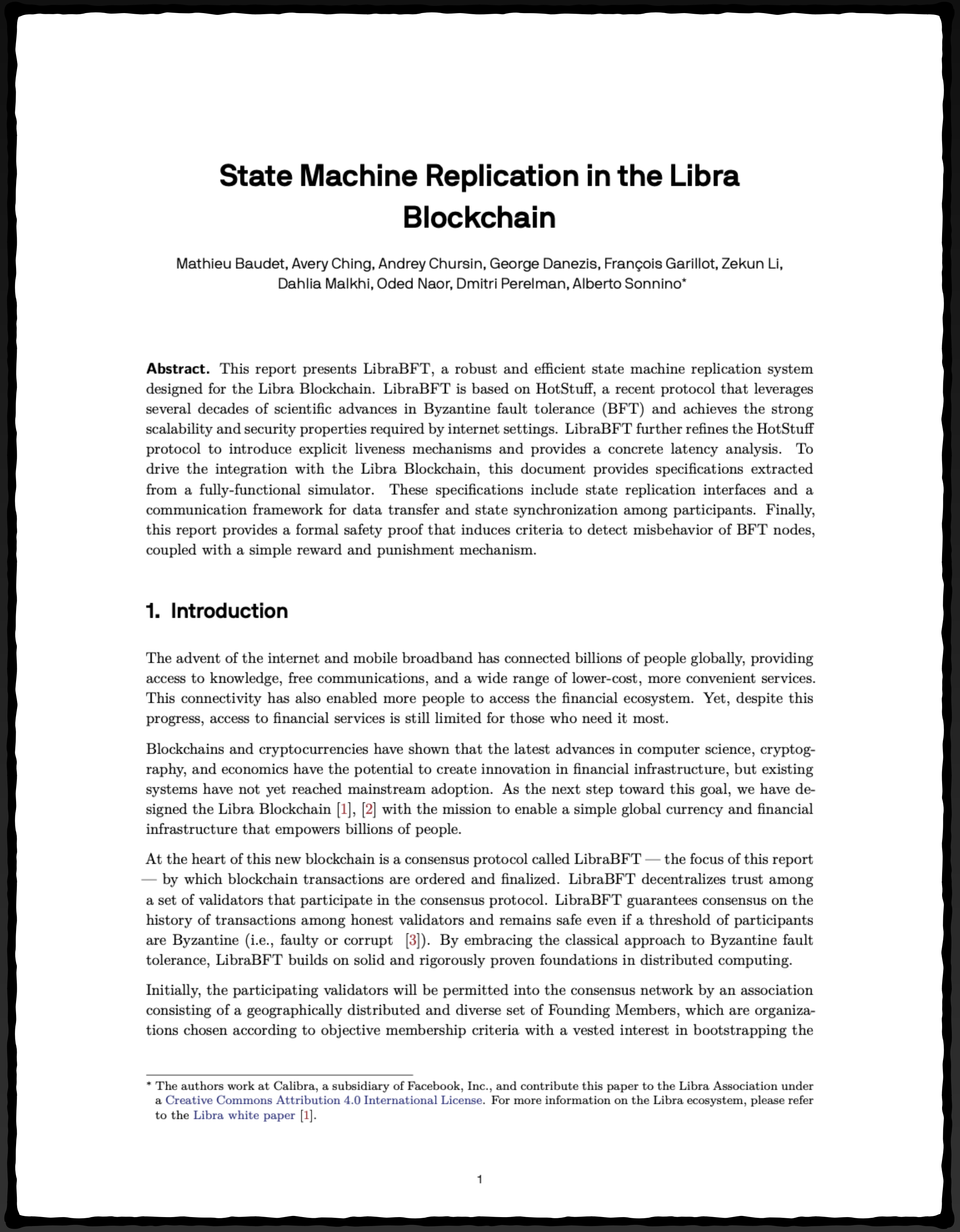
✓ Clearly isolated components

## HashGraph

✗ Hard to garbage collect

✗ Unclear block synchroniser

# The first 6 months...



## State Machine Replication in the Libra Blockchain

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

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

### 1. Introduction

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

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

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

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

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

1

## SMR in the Libra Blockchain

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

## Research Questions

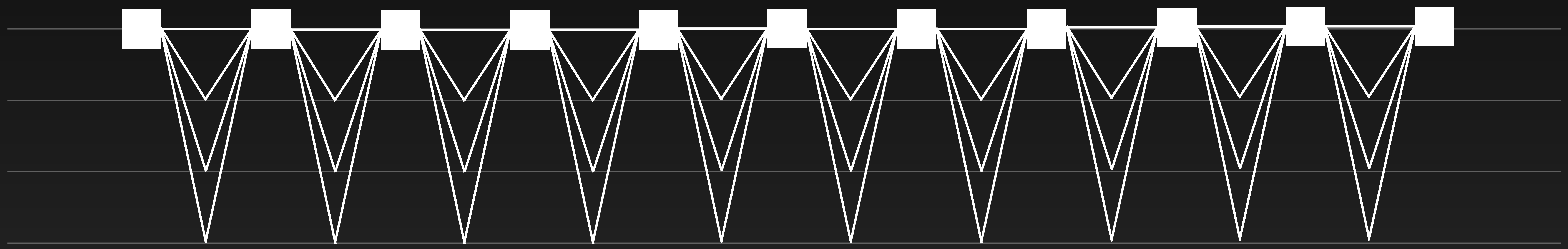
1. Network model?

## Lessons Learned

1. Modularisation is a design strategy

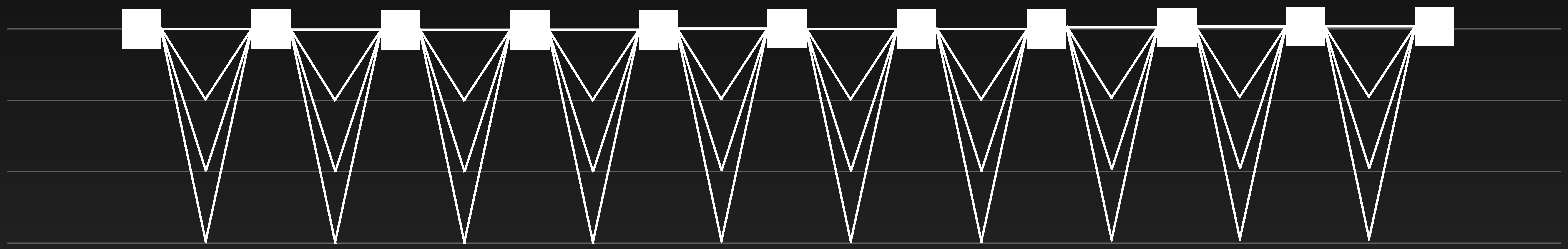
# HotStuff

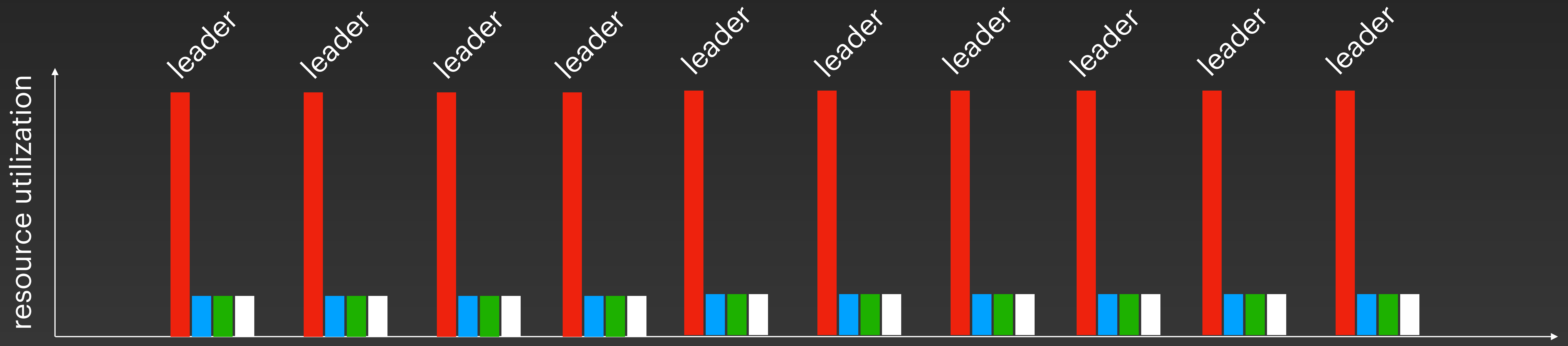
## Typical leader-based protocols



# Naive Implementation

## Uneven resource utilisation





# Research Questions

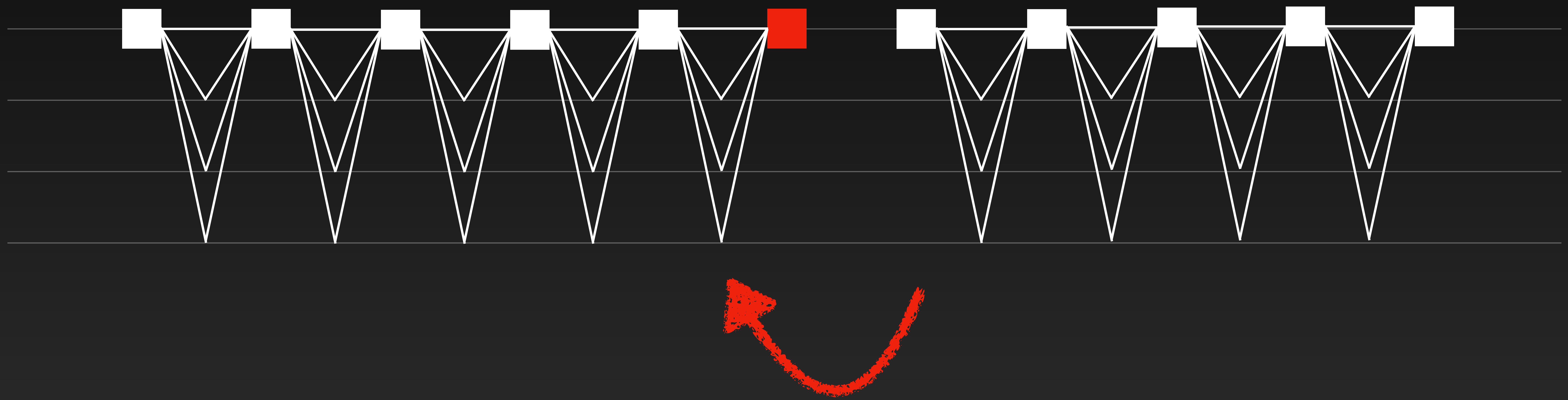
1. Network model?

# Lessons Learned

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

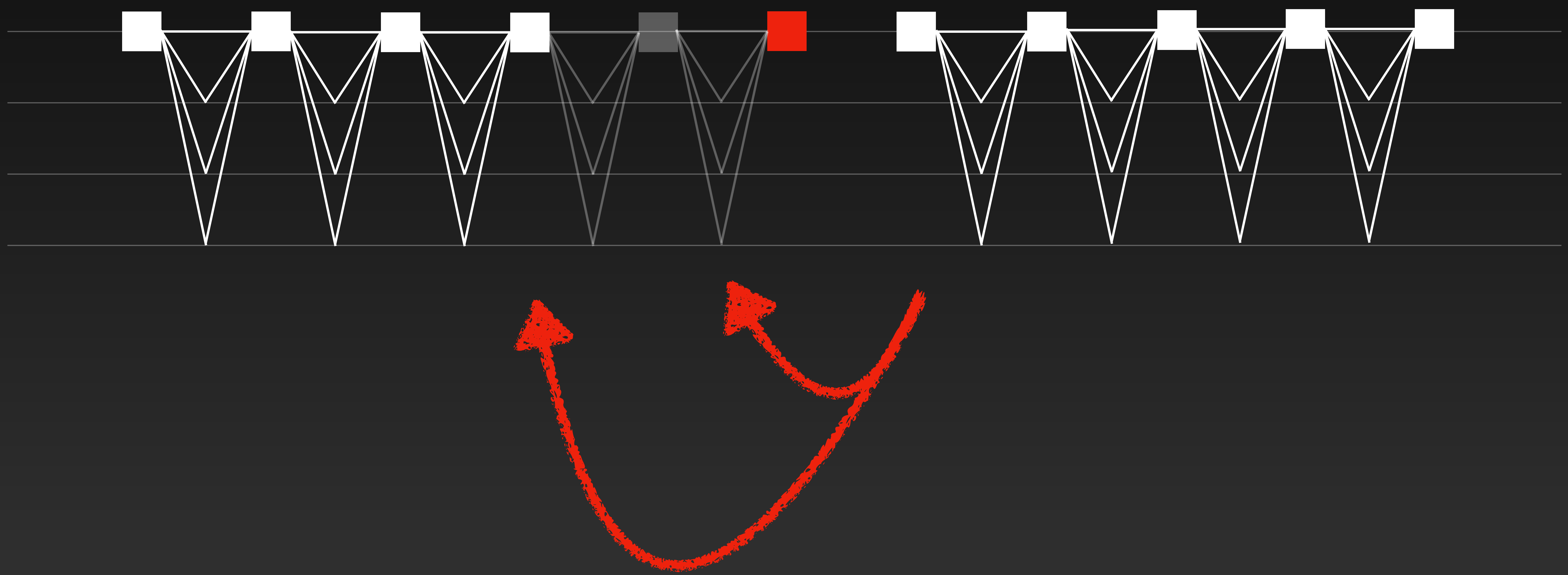
# Leader-Driven Consensus

Fragility to faults and asynchrony

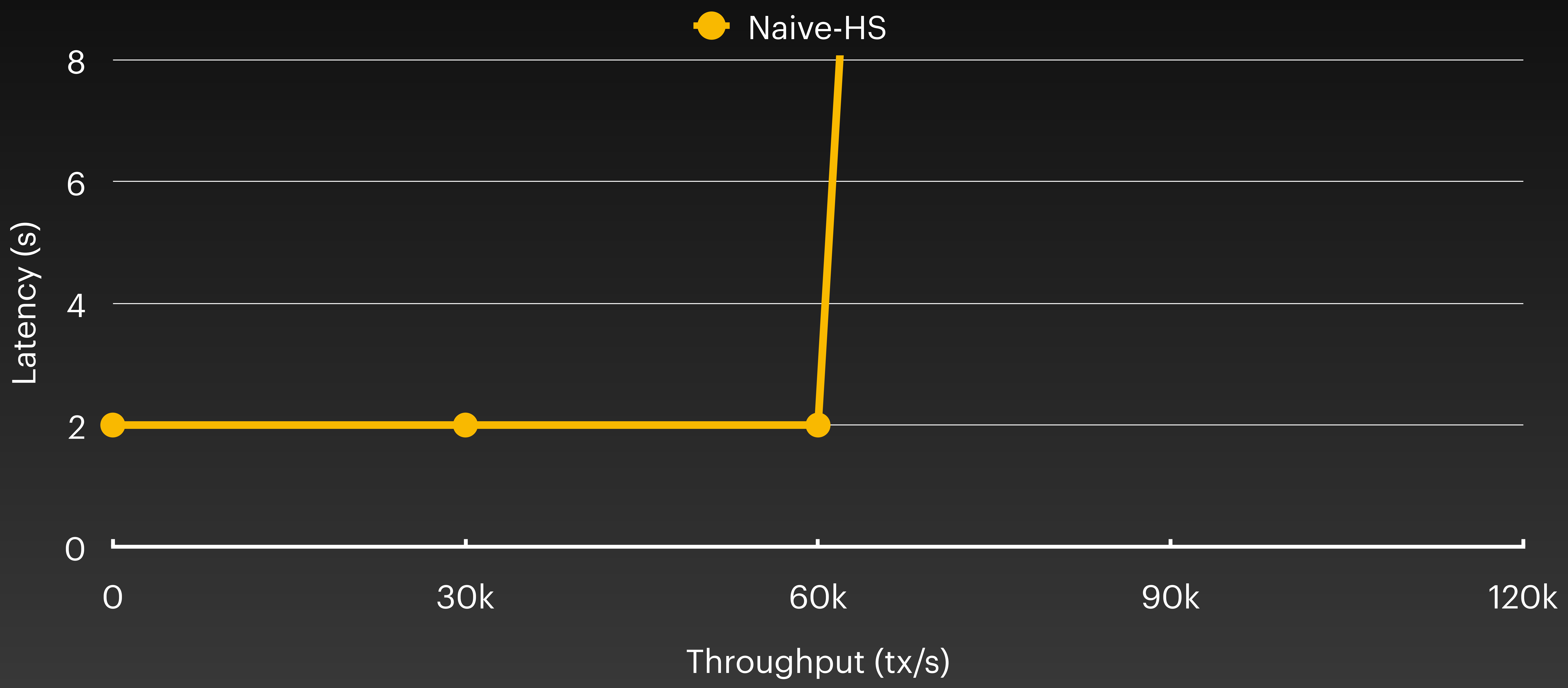


# Leader-Driven Consensus

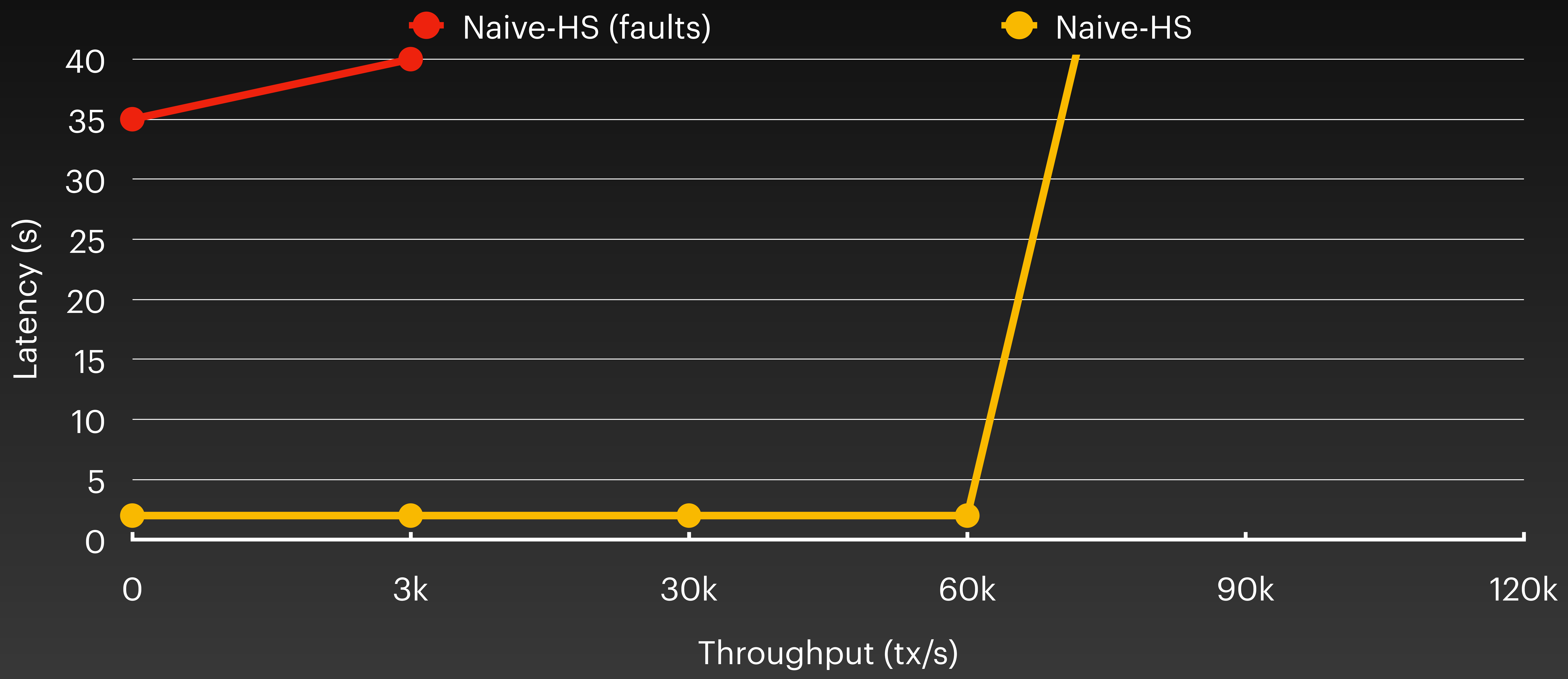
## Fragility to faults and asynchrony



# Performance



# Performance



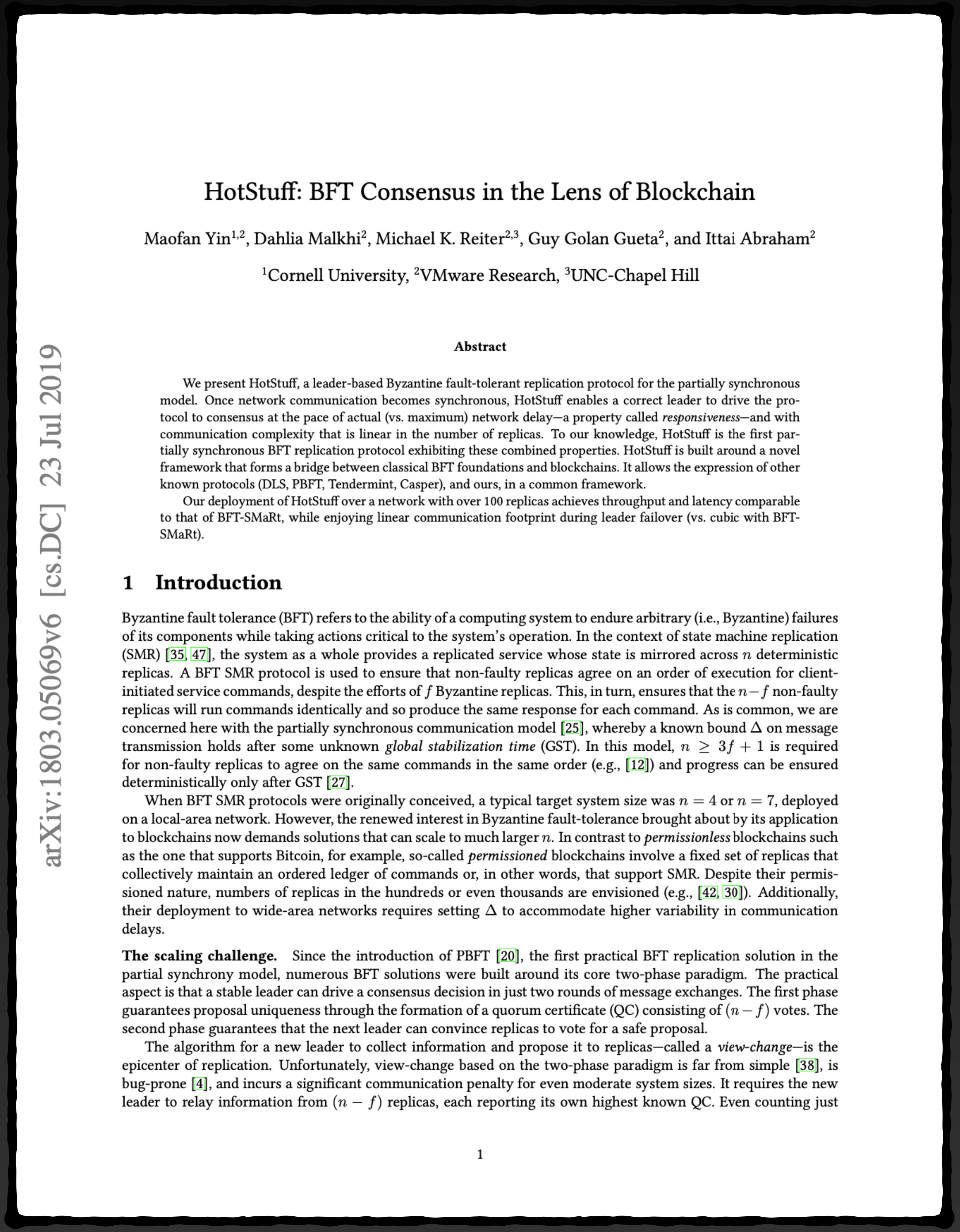
# Research Questions

1. Network model?

# Lessons Learned

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

# Libra, 2019



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

HotStuff: BFT Consensus in the Lens of Blockchain

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

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

**Abstract**

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

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

**1 Introduction**

Byzantine fault tolerance (BFT) refers to the ability of a computing system to endure arbitrary (i.e., Byzantine) failures of its components while taking actions critical to the system's operation. In the context of state machine replication (SMR) [35, 47], the system as a whole provides a replicated service whose state is mirrored across $n$ deterministic replicas. A BFT SMR protocol is used to ensure that non-faulty replicas agree on an order of execution for client-initiated service commands, despite the efforts of $f$ Byzantine replicas. This, in turn, ensures that the $n-f$ non-faulty replicas will run commands identically and so produce the same response for each command. As is common, we are concerned here with the partially synchronous communication model [25], whereby a known bound $\Delta$ 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 $\Delta$ to accommodate higher variability in communication delays.

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

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

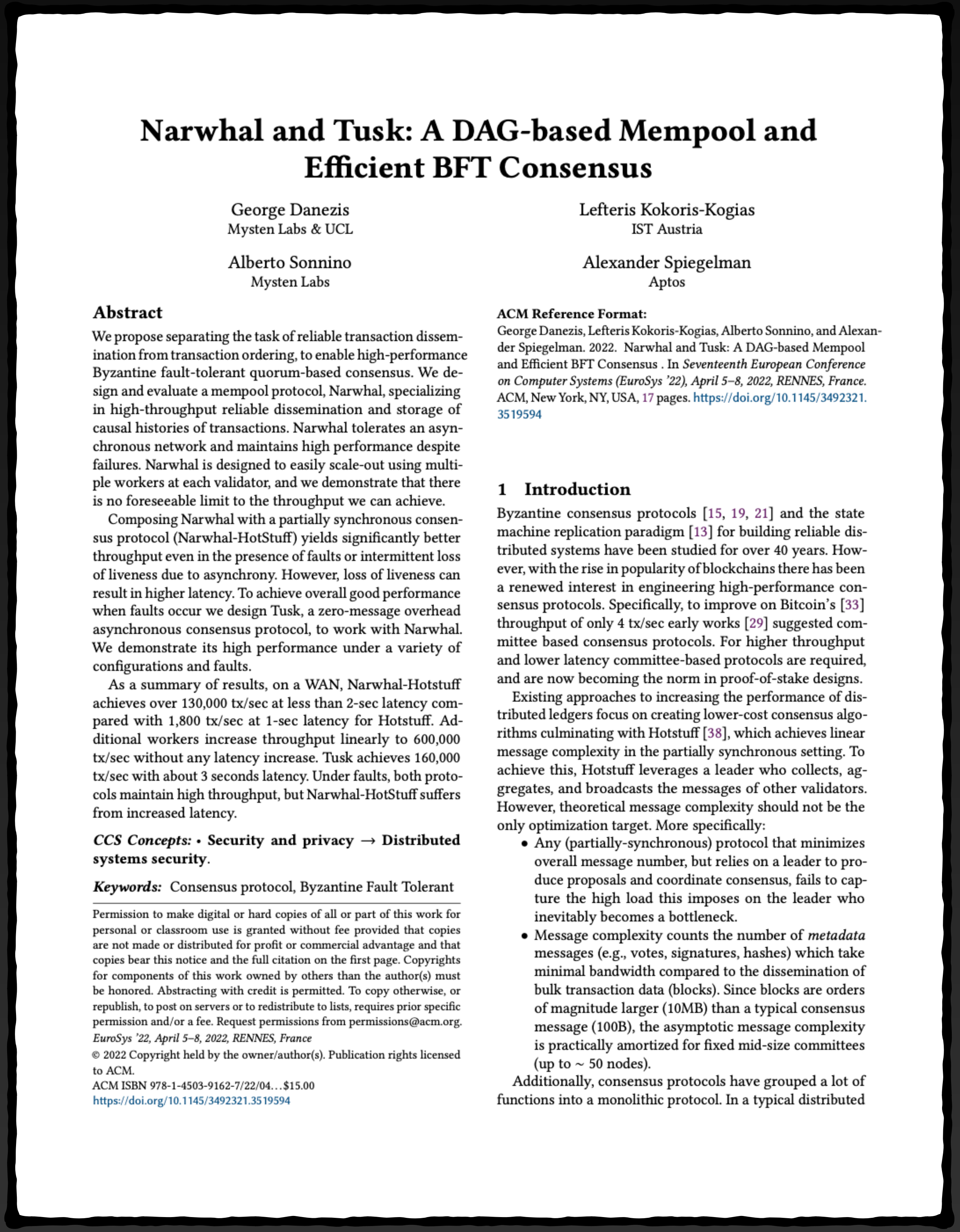
1

## HotStuff (naive mempool)

- Linear
- Clearly isolated components

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

# Libra, 2021



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

George Danezis
Mysten Labs & UCL

Lefteris Kokoris-Kogias
IST Austria

Alberto Sonnino
Mysten Labs

Alexander Spiegelman
Aptos

**Abstract**

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

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

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

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

***Keywords:*** Consensus protocol, Byzantine Fault Tolerant


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

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

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

## 1 Introduction

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

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

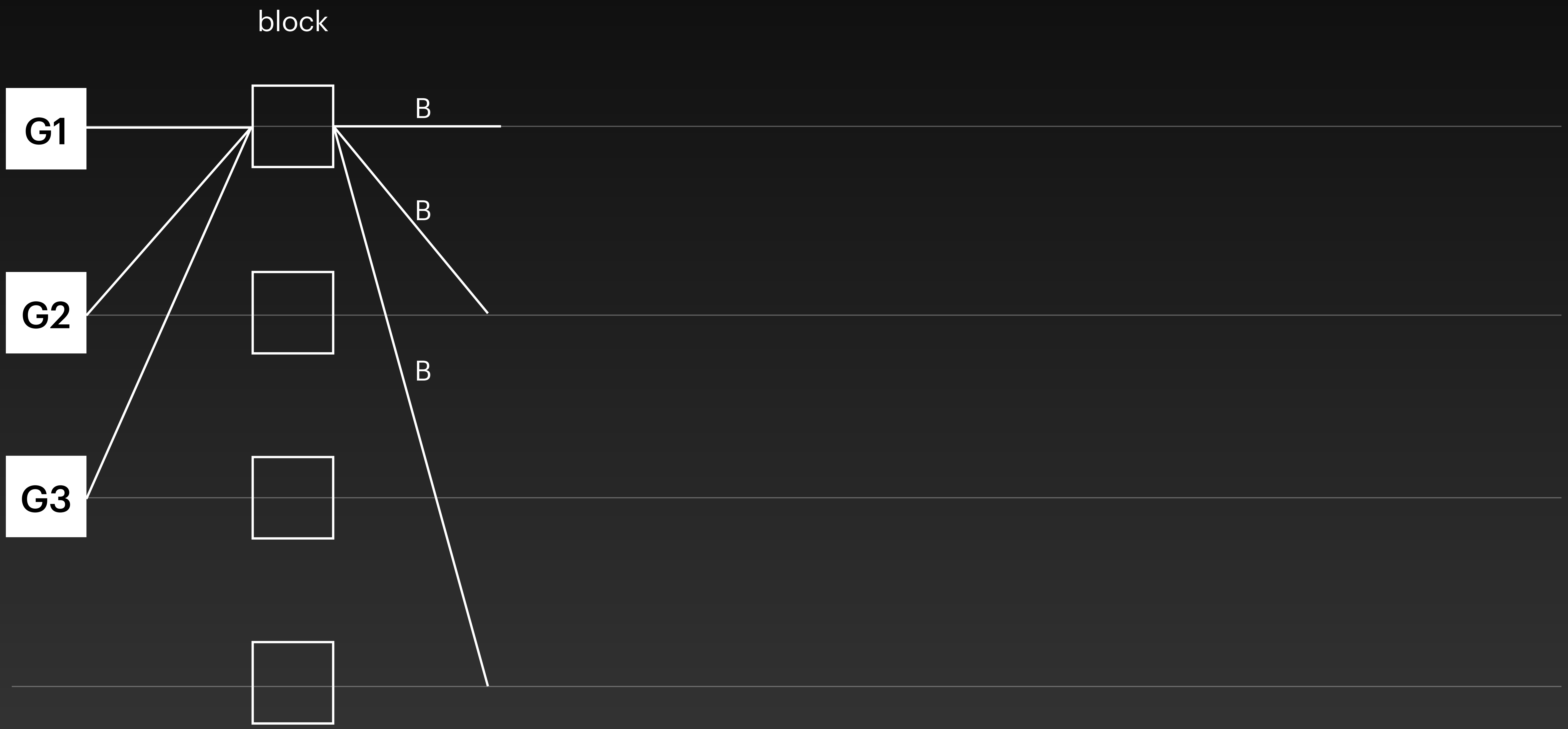
- Any (partially-synchronous) protocol that minimizes overall message number, but relies on a leader to produce proposals and coordinate consensus, fails to capture the high load this imposes on the leader who inevitably becomes a bottleneck.
- Message complexity counts the number of *metadata* messages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

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

## Narwhal

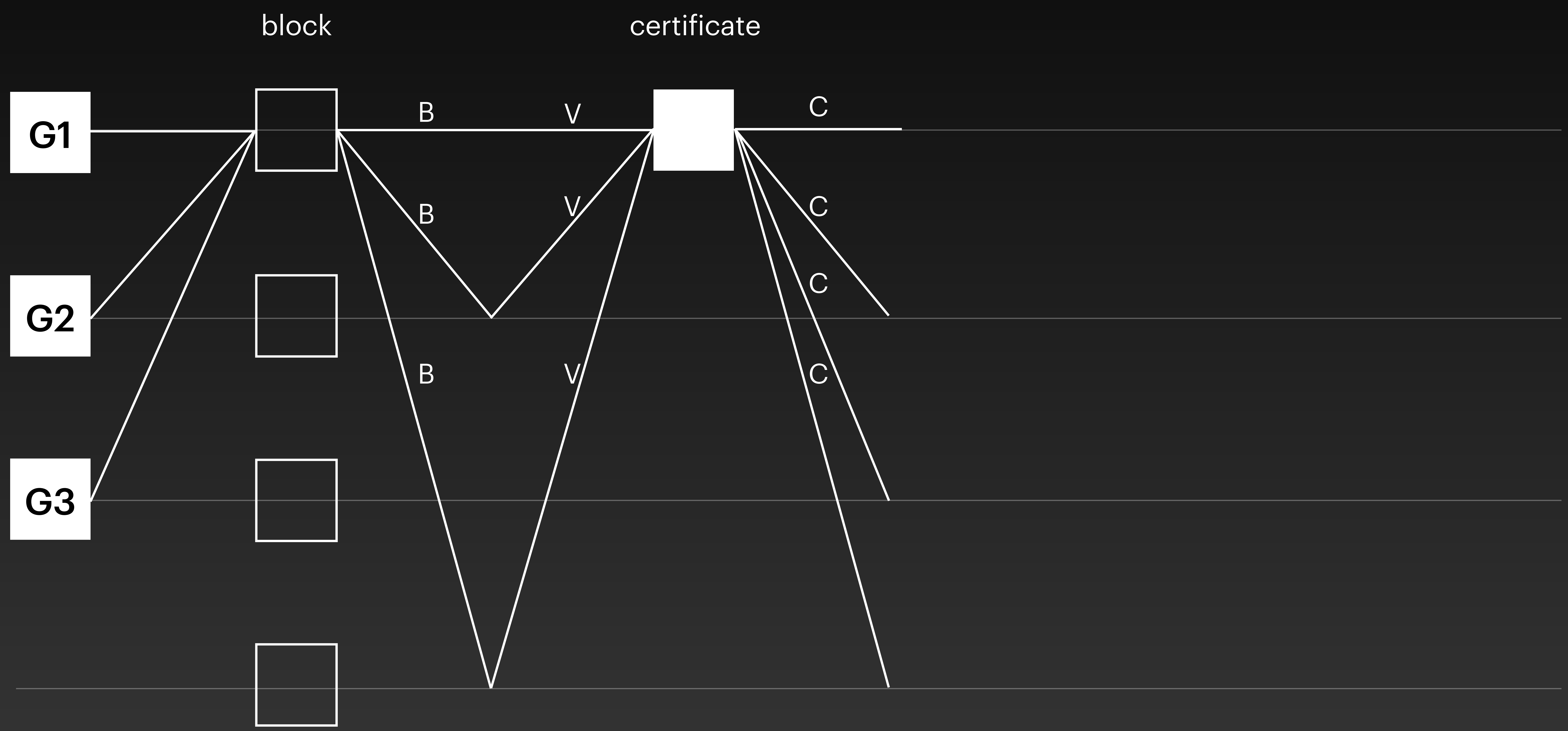
- Quadratic but even resource utilisation
- Separation between consensus and data dissemination

# Narwhal

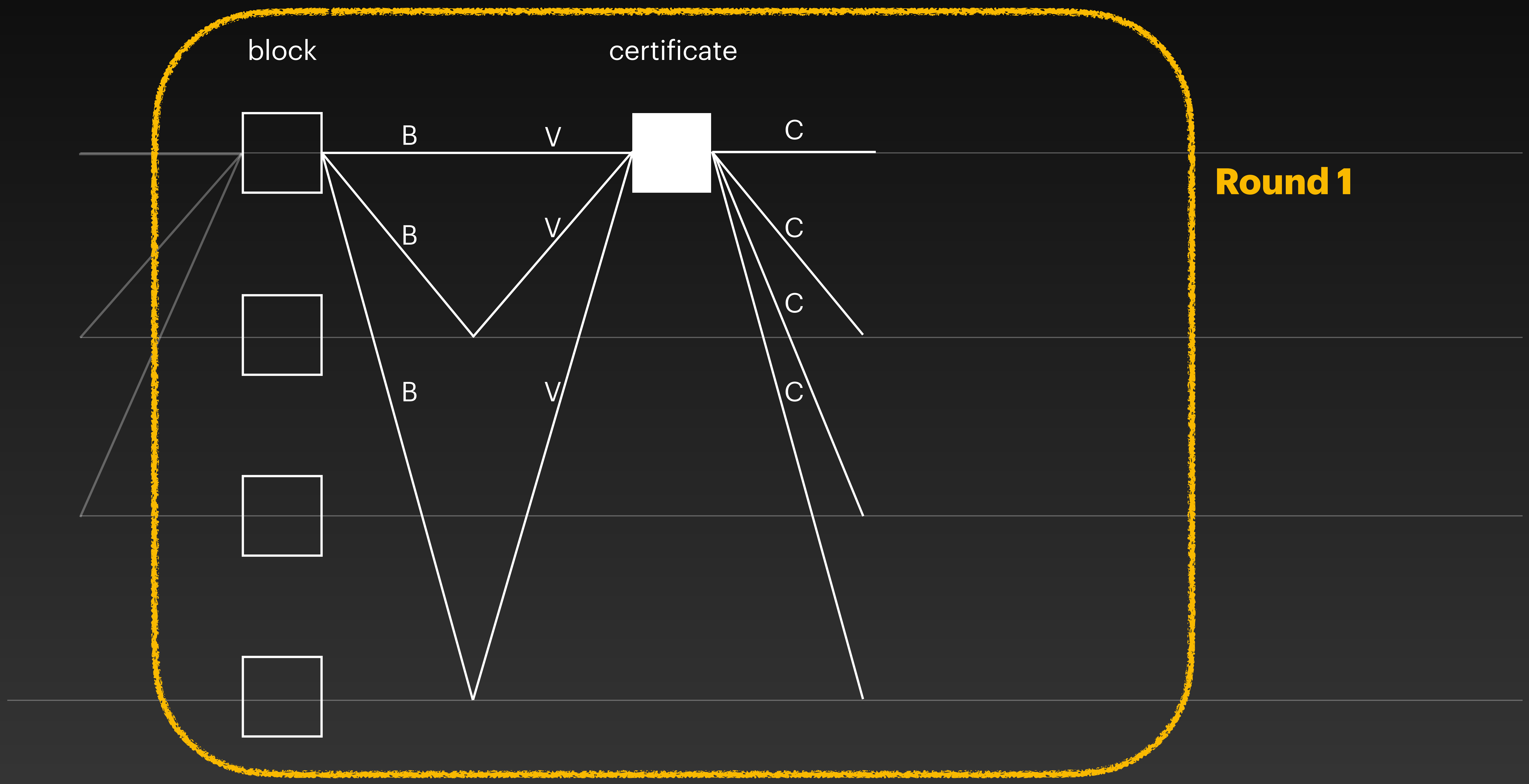


# Narwhal

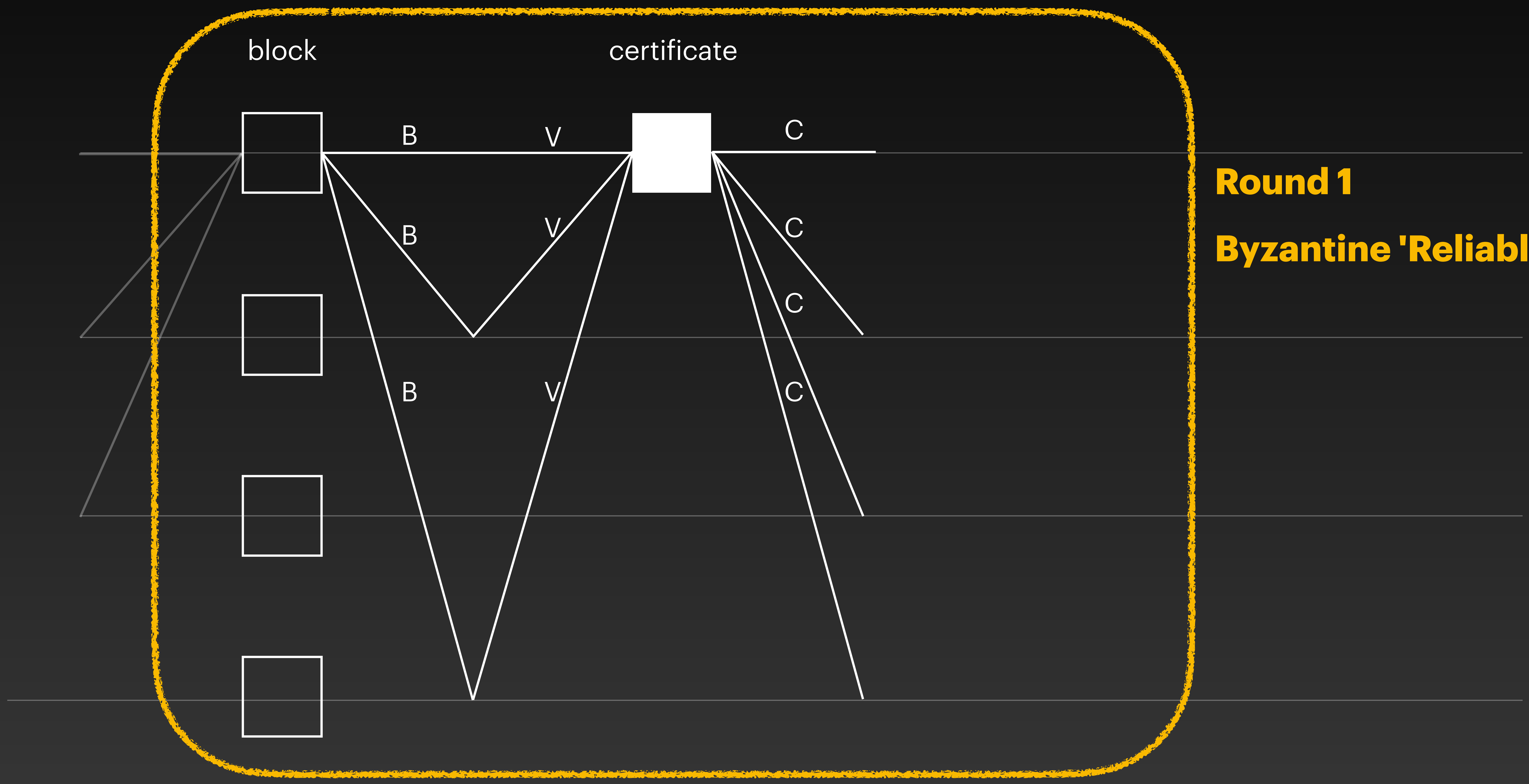
# Narwhal



# Narwhal



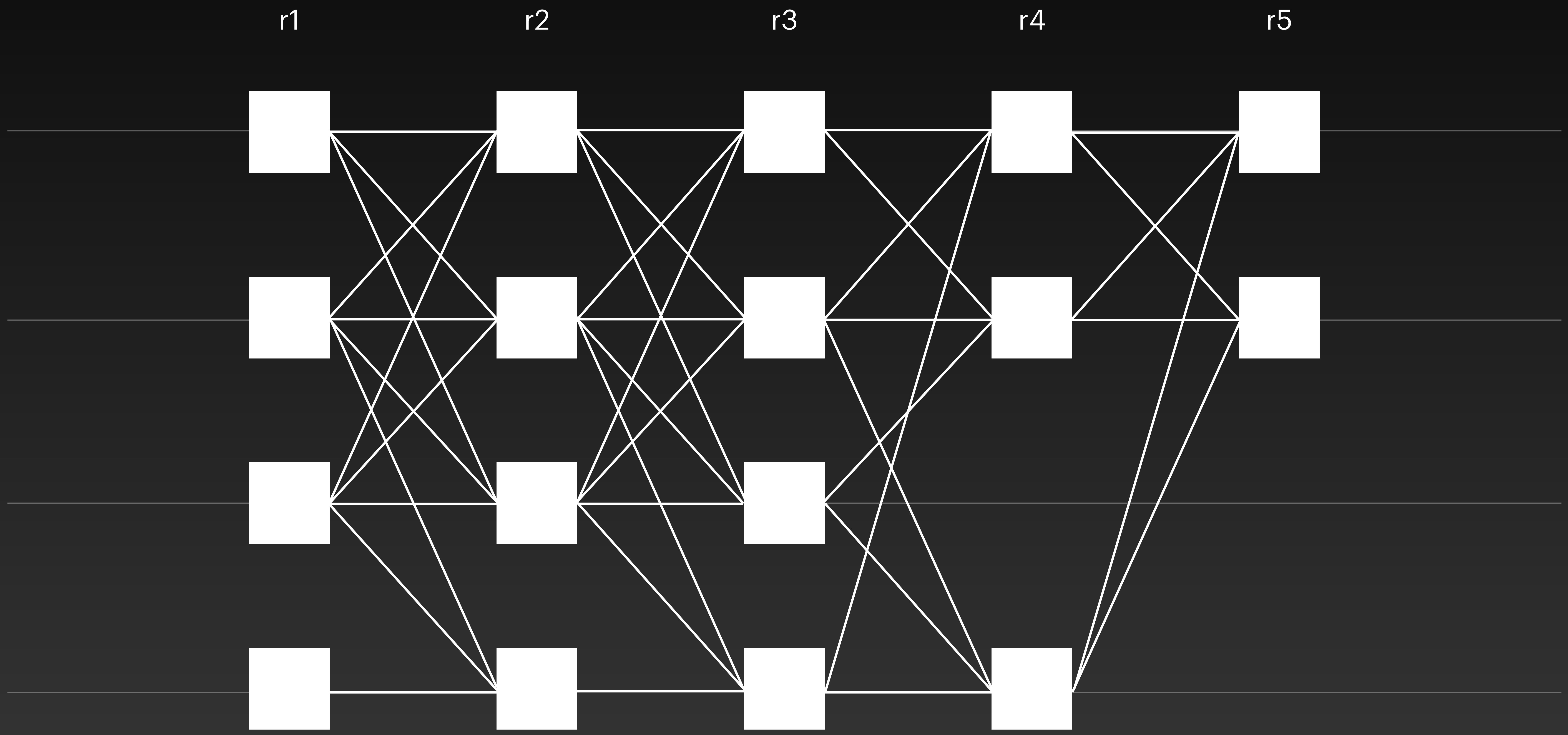
# Narwhal




**Round 1**

**Byzantine 'Reliable' Broadcast**

# Narwhal



# Research Questions

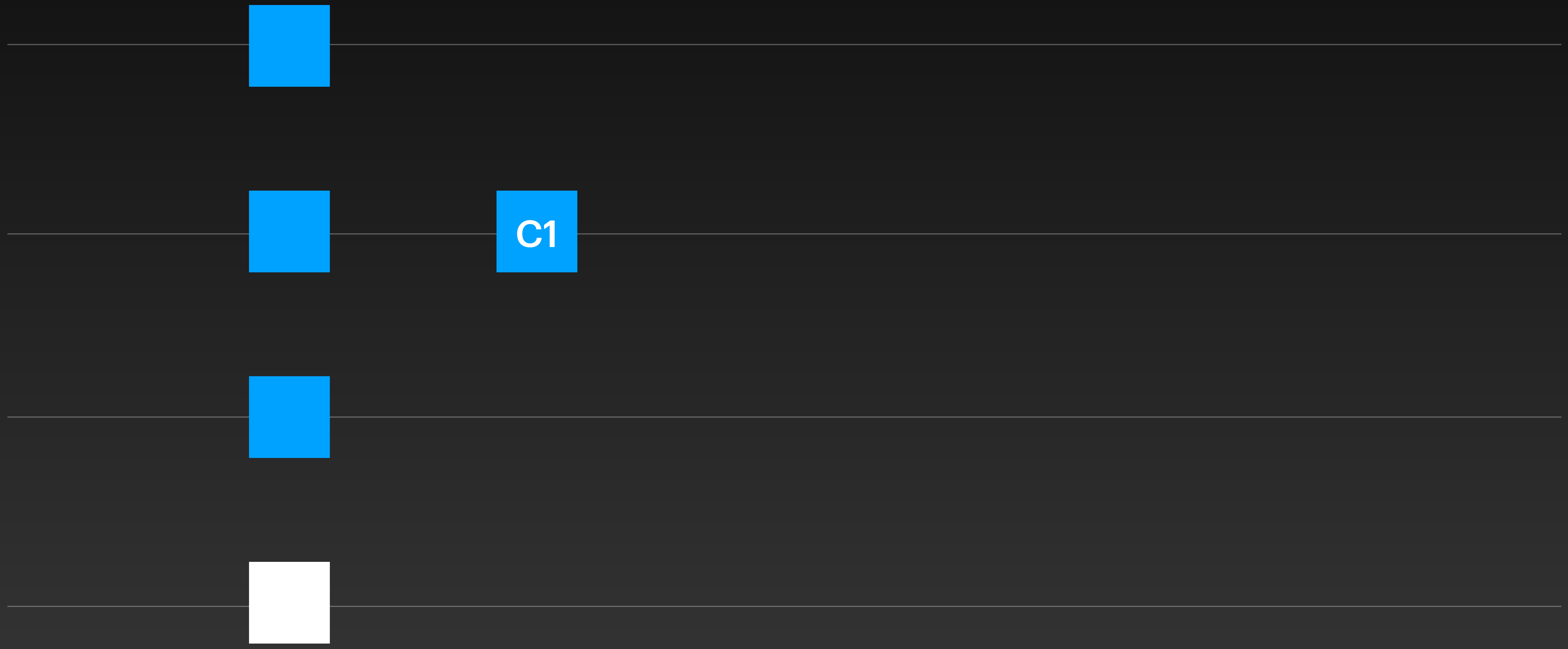
1. Network model?

# Lessons Learned

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

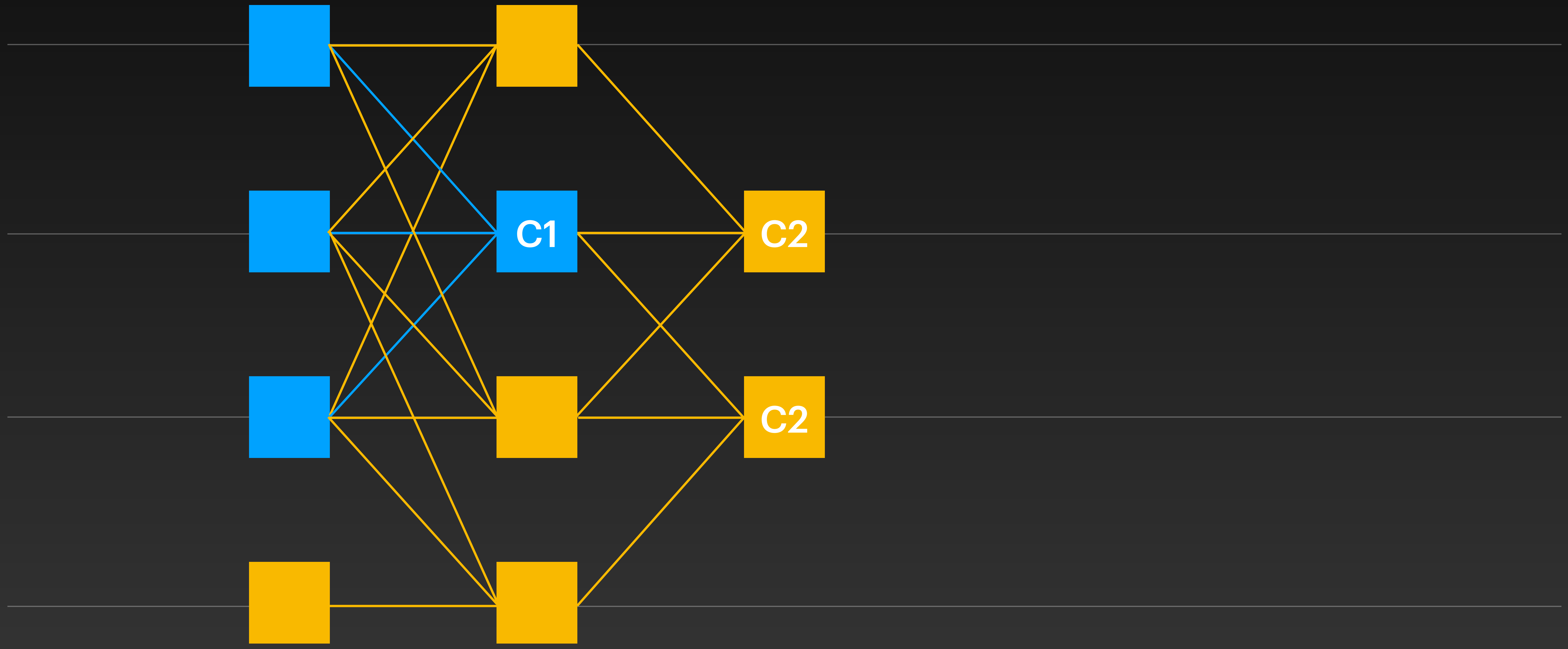
# HotStuff on Narwhal

## Enhanced commit rule



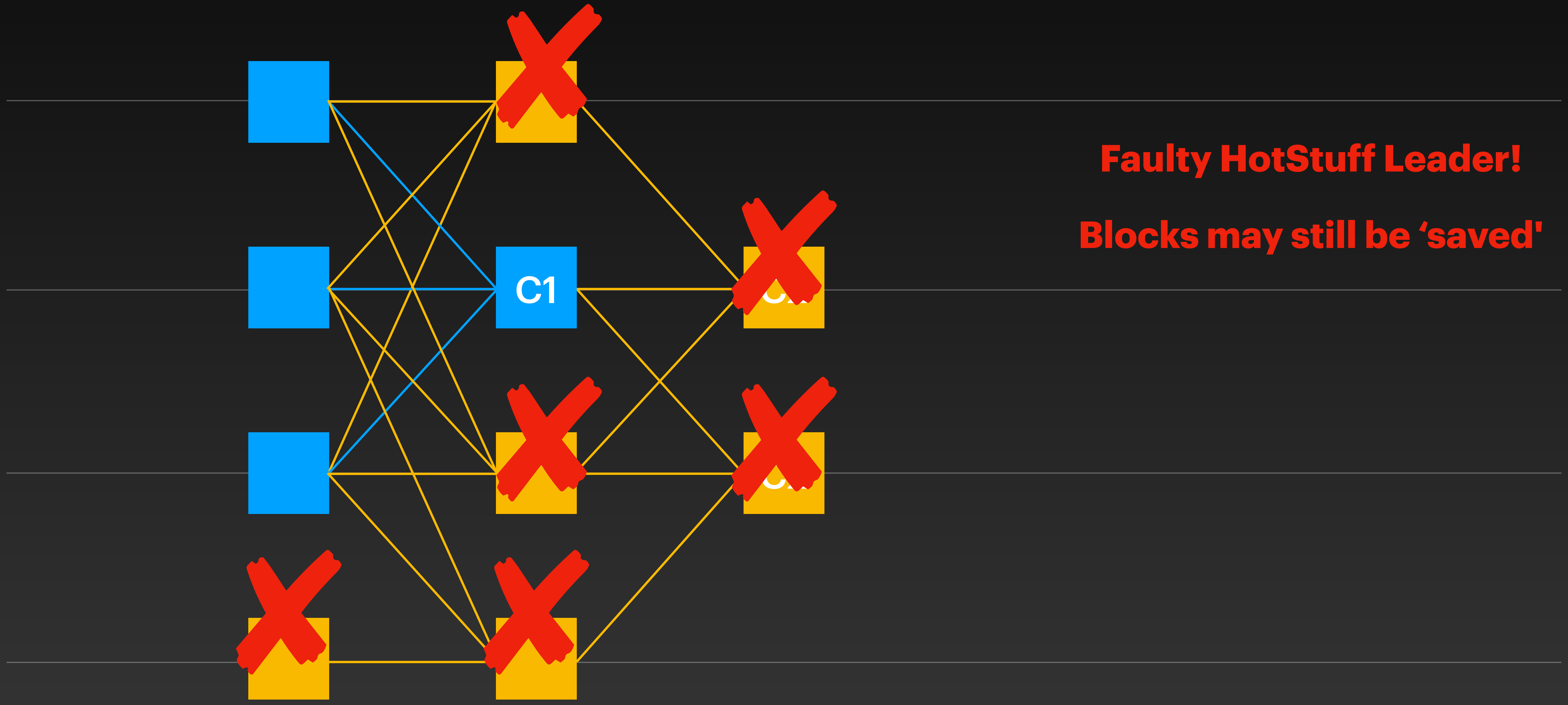
# HotStuff on Narwhal

## Enhanced commit rule



# HotStuff on Narwhal

## Enhanced commit rule

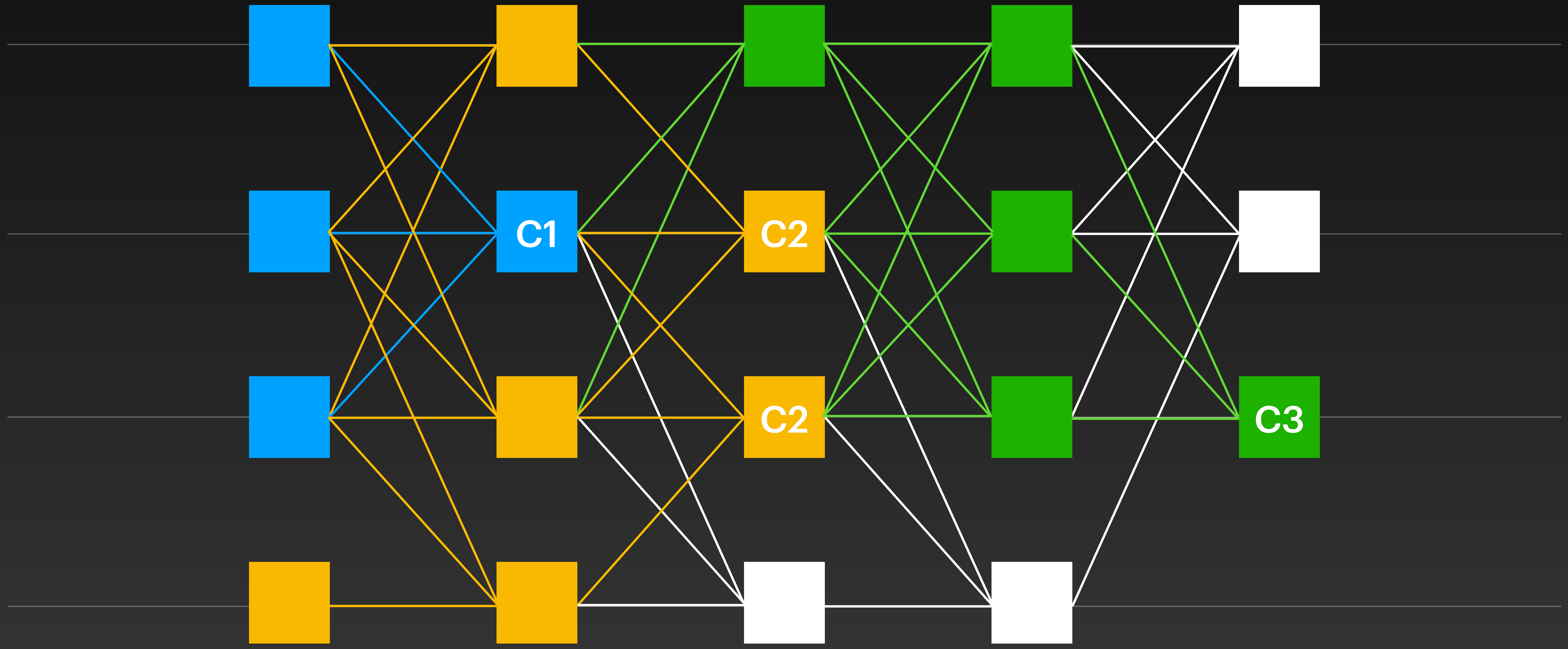



**Faulty HotStuff Leader!**

**Blocks may still be ‘saved'**

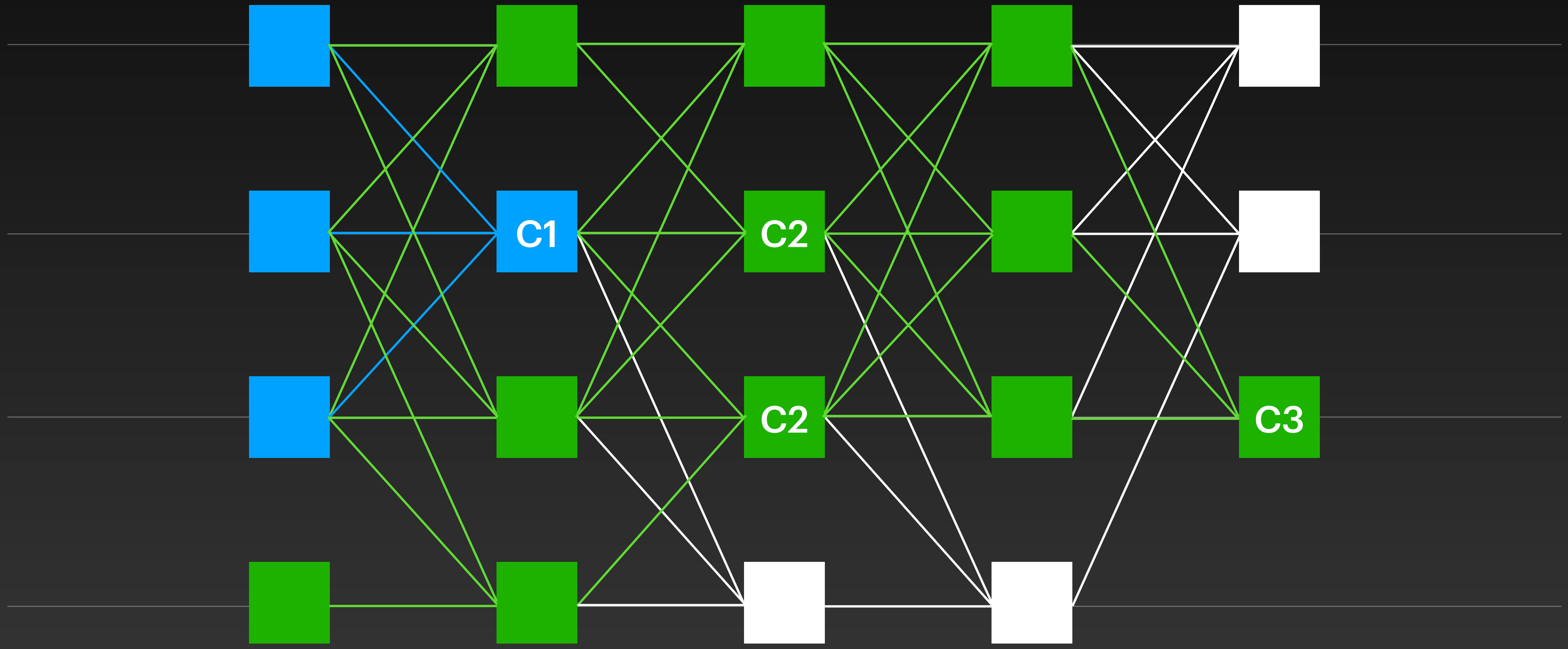
# HotStuff on Narwhal

## Enhanced commit rule

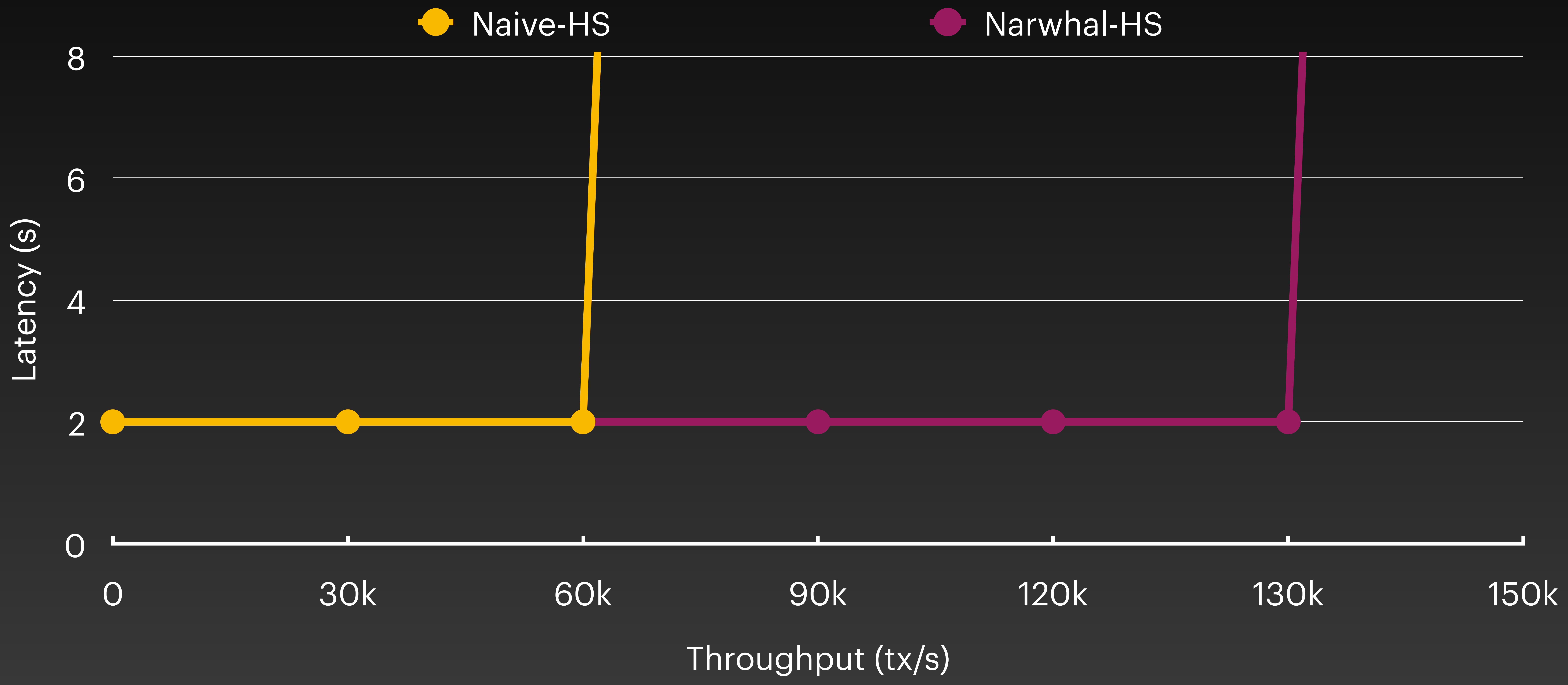


# HotStuff on Narwhal

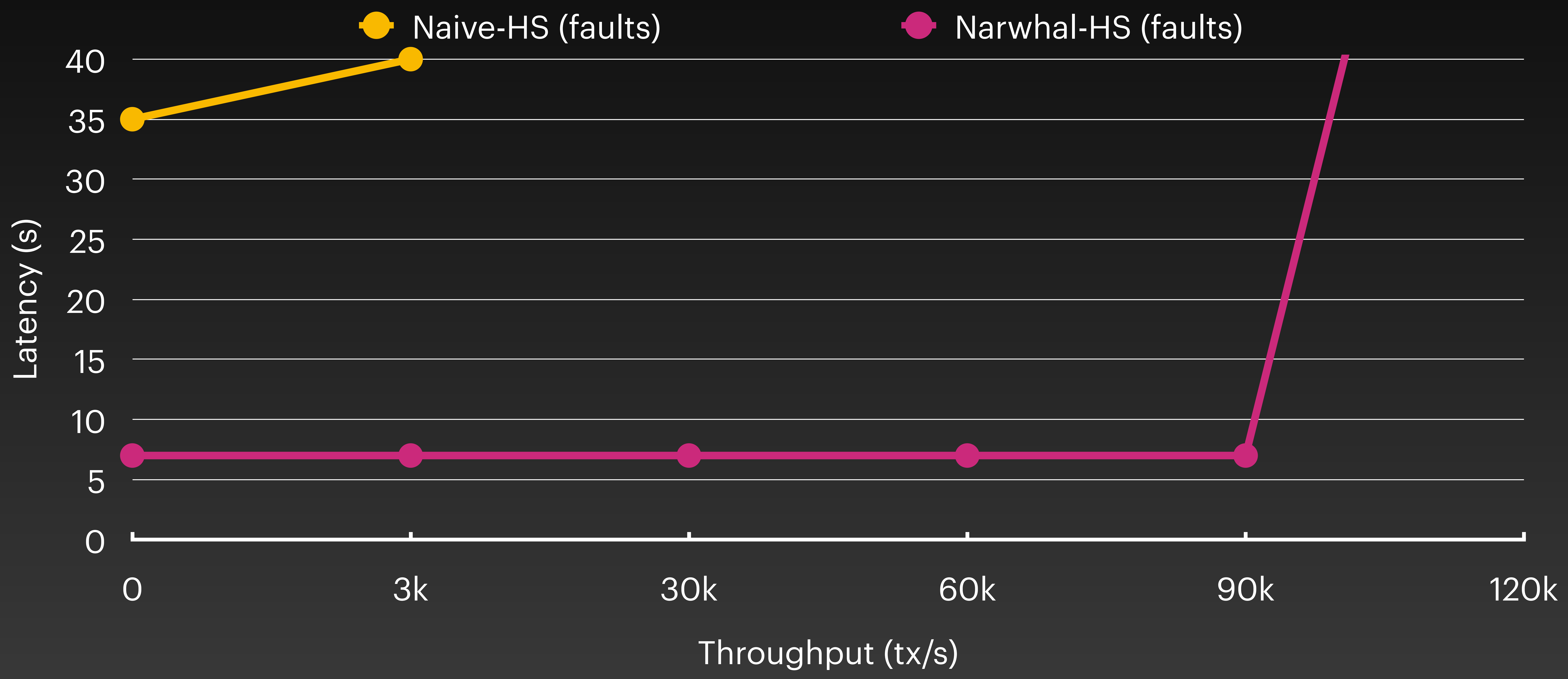
## Enhanced commit rule



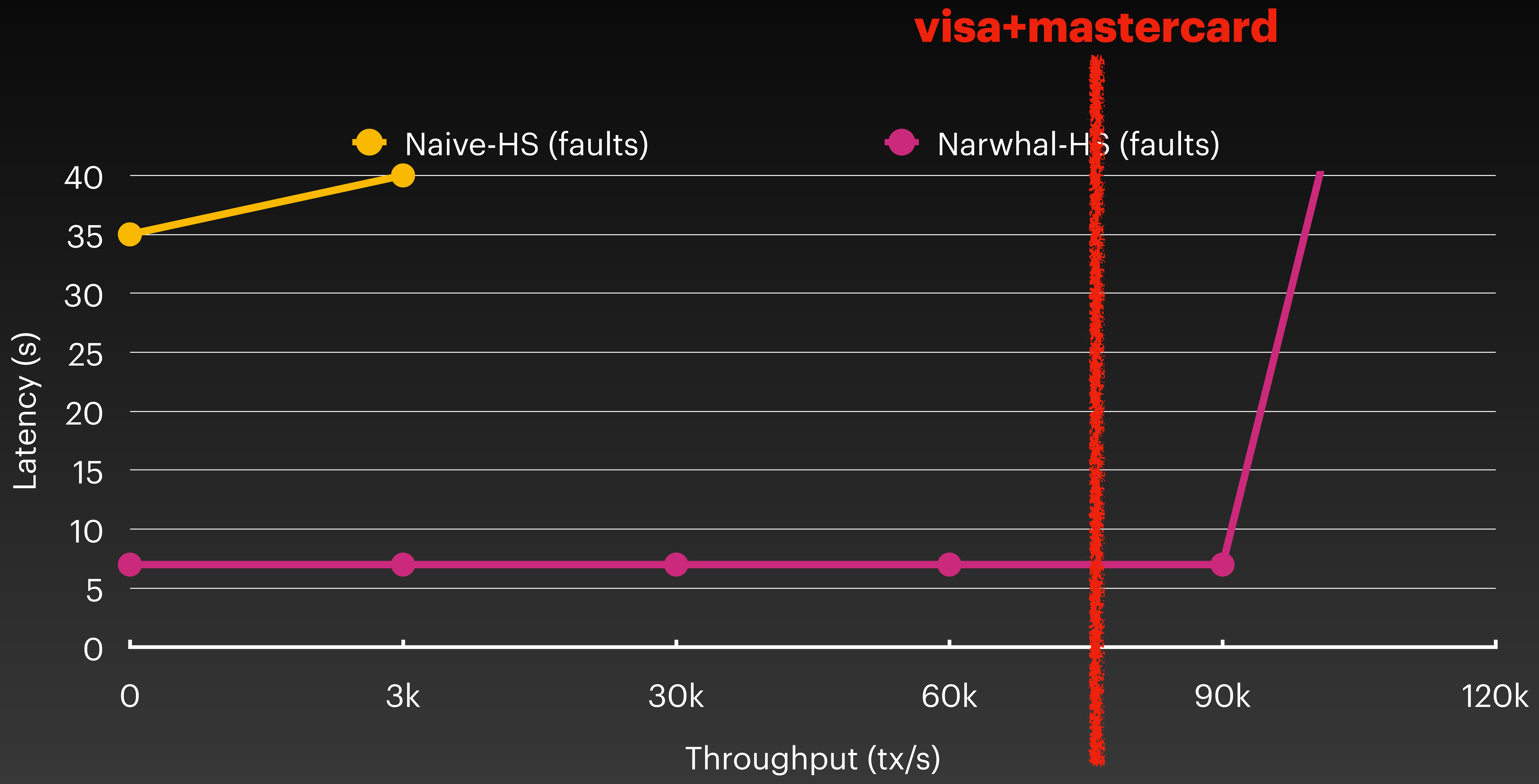
# Performance



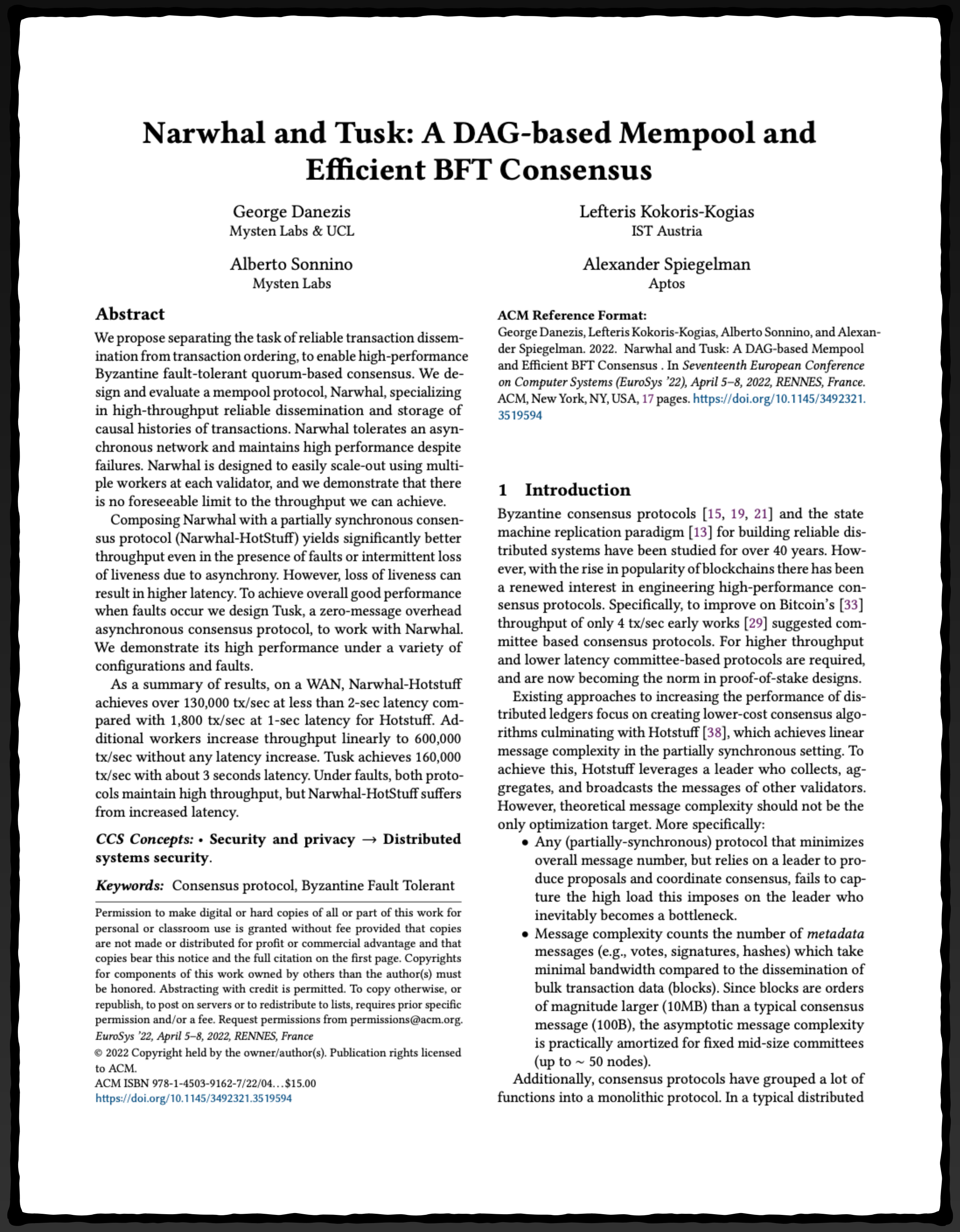
# Performance



# Performance



# Libra, 2021



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


George Danezis
Mysten Labs & UCL

Lefteris Kokoris-Kogias
IST Austria

Alberto Sonnino
Mysten Labs

Alexander Spiegelman
Aptos



### Abstract

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

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

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



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

***Keywords:*** Consensus protocol, Byzantine Fault Tolerant





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





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

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


### 1 Introduction

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

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

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

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

## Narwhal

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

- High engineering complexity

## Research Questions

1. Network model?
2. BFT testing?

## Lessons Learned

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

**DagRider** **Tusk** **Bullshark** **Dumbo-NG** …

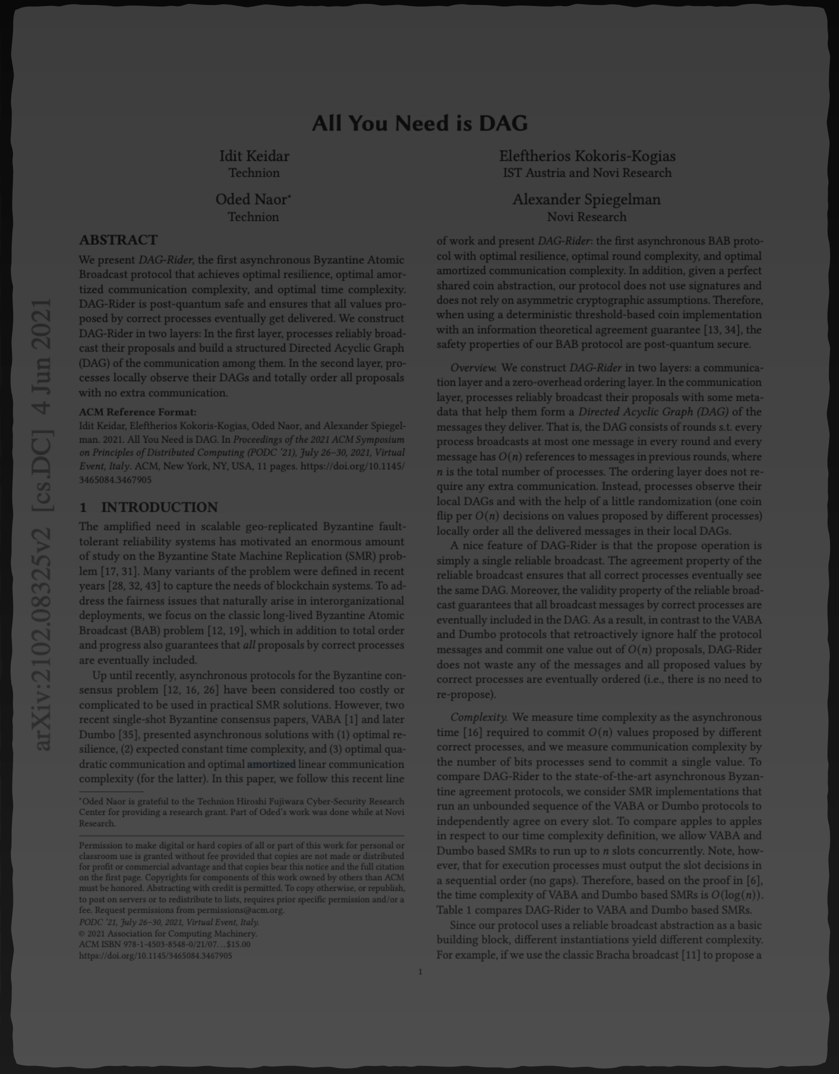
## Data Dissemination

- Hard to make efficient
- 99% of the code

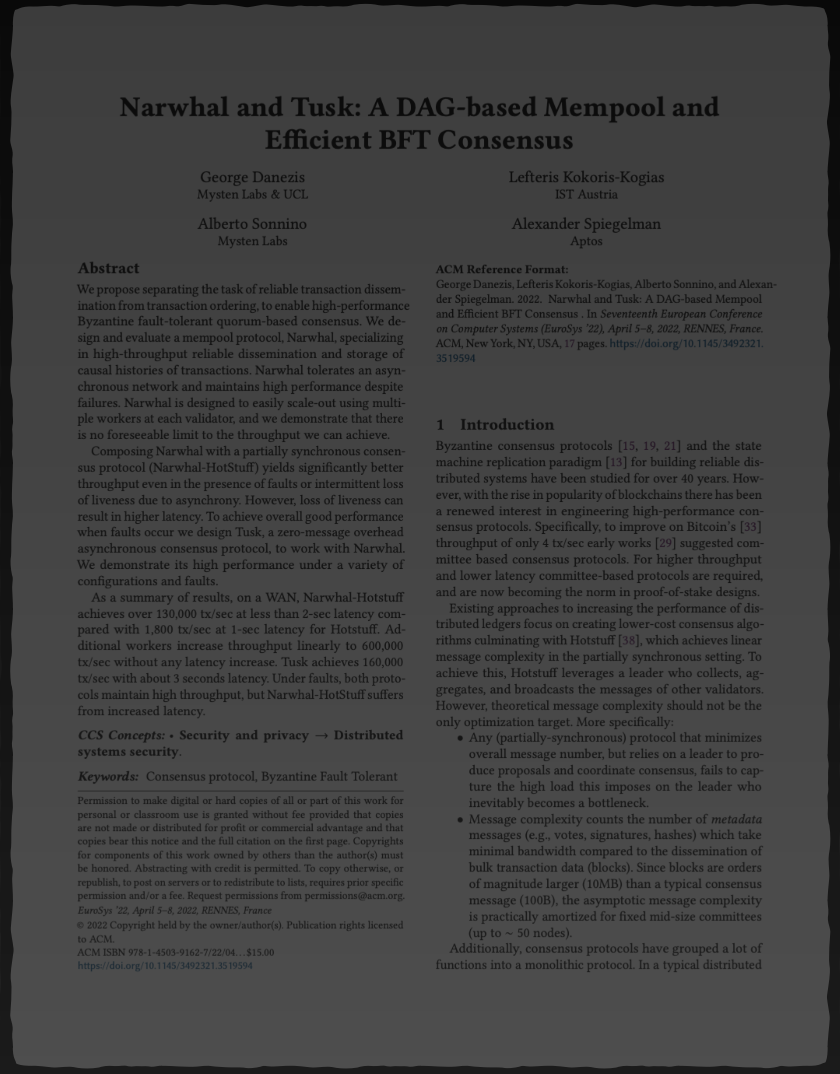
## Consensus

- Error prone
- Isolated, easy to maintain

# Bullshark


All You Need is DAG

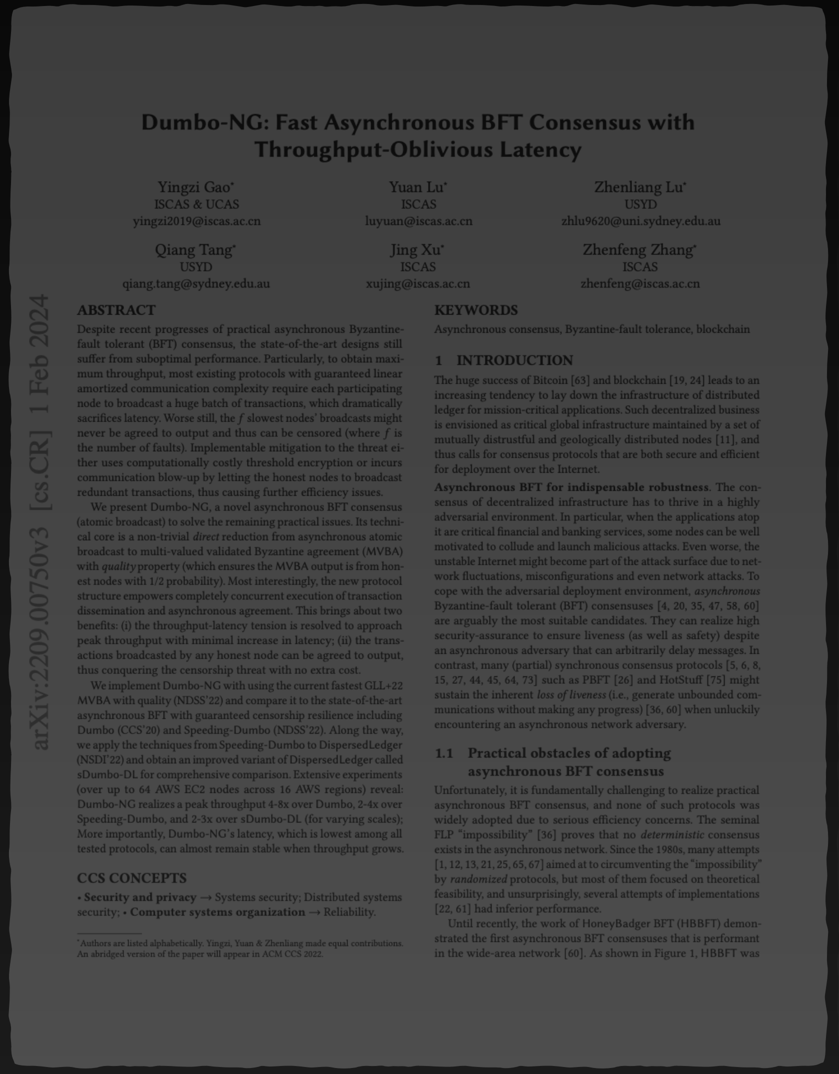
Idit Keidar, Technion
Eleftherios Kokoris-Kogias, IST Austria and Novi Research
Oded Naor, Technion
Alexander Spiegelman, Novi Research


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

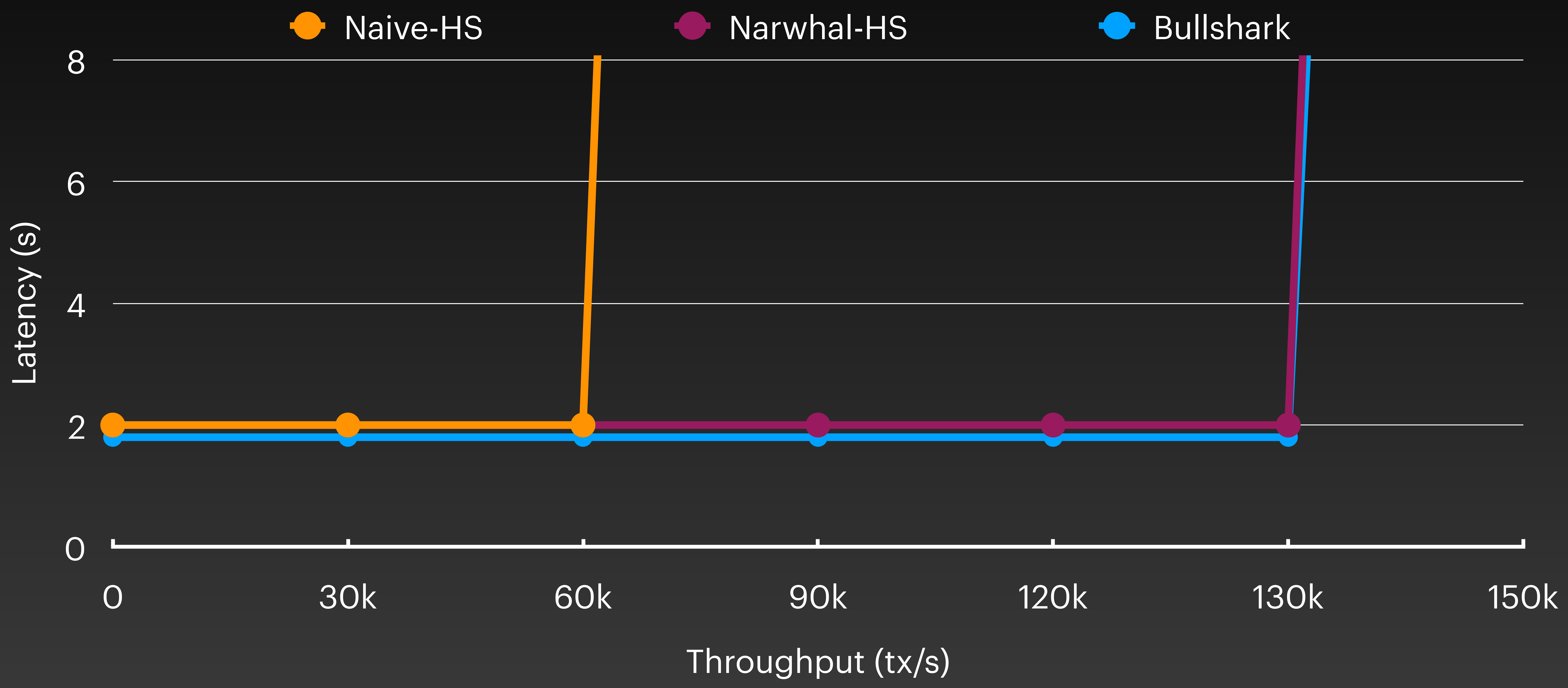
George Danezis, Mysten Labs & UCL
Lefteris Kokoris-Kogias, IST Austria
Alberto Sonnino, Mysten Labs
Alexander Spiegelman, Aptos


Bullshark: DAG BFT Protocols Made Practical

Alexander Spiegelman, sasha@aptoslabs.com, Aptos
Neil Giridharan, giridhn@berkeley.edu, UC Berkeley
Alberto Sonnino, alberto@mysternlabs.com, Mysten Labs
Lefteris Kokoris-Kogias, ekokoris@ist.ac.at, IST Austria


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

# 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

## Shoal

Shoal: Improving DAG-BFT Latency And Robustness

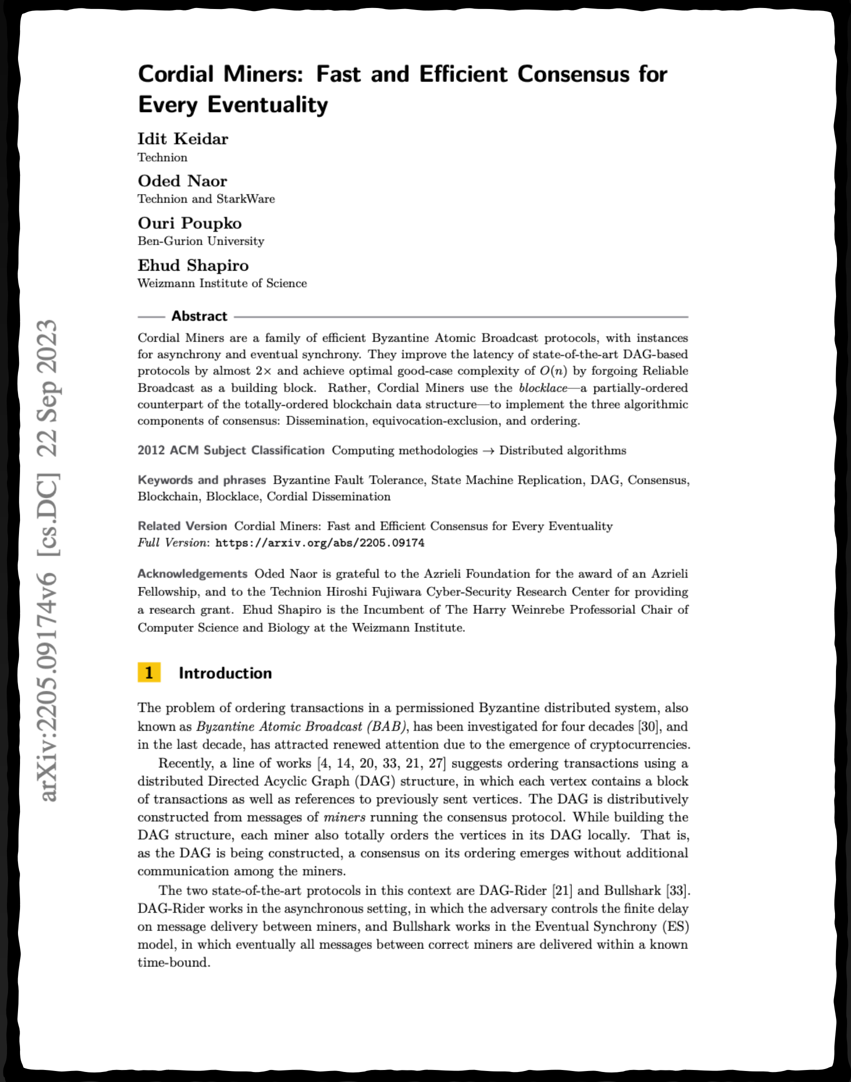
Alexander Spiegelman, Balaji Arun, Rati Gelashvili, Zekun Li

## Sailfish

Sailfish: Towards Improving the Latency of DAG-based BFT

Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, Kartik Nayak

## CM



Cordial Miners: Fast and Efficient Consensus for Every Eventuality

Idit Keidar, Oded Naor, Ouri Poupko, Ehud Shapiro

## Mysticeti

MYSTICETI: Reaching the Latency Limits with Uncertified DAGs

...

## Techniques

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

# Discussion

**Certified DAG** ⟷ **Uncertified DAG**

**Shoal/shoal++**

- Low latency
- Easier synchroniser
- Leverage existing code

**Sailfish/BBCA**

- Lower latency
- Easy synchroniser
- Flexible

**CM/Mysticeti**

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

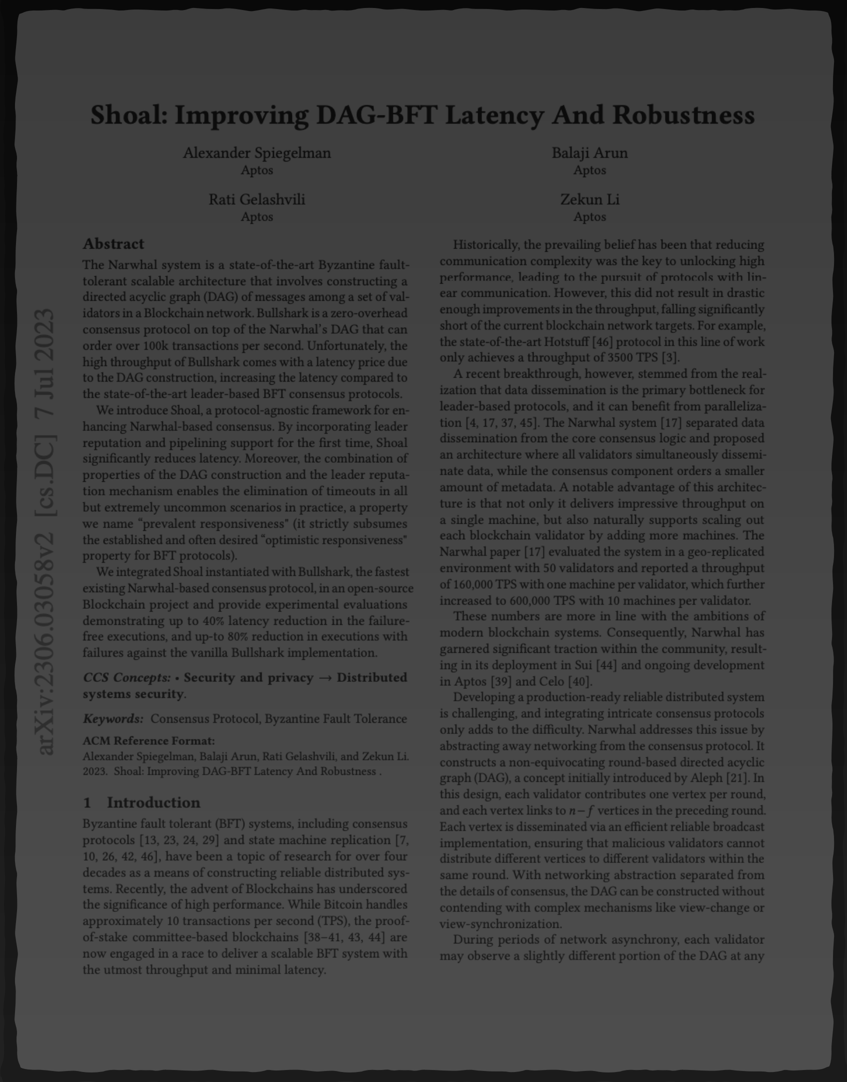
## Research Questions

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

## Lessons Learned

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

# Mysticeti

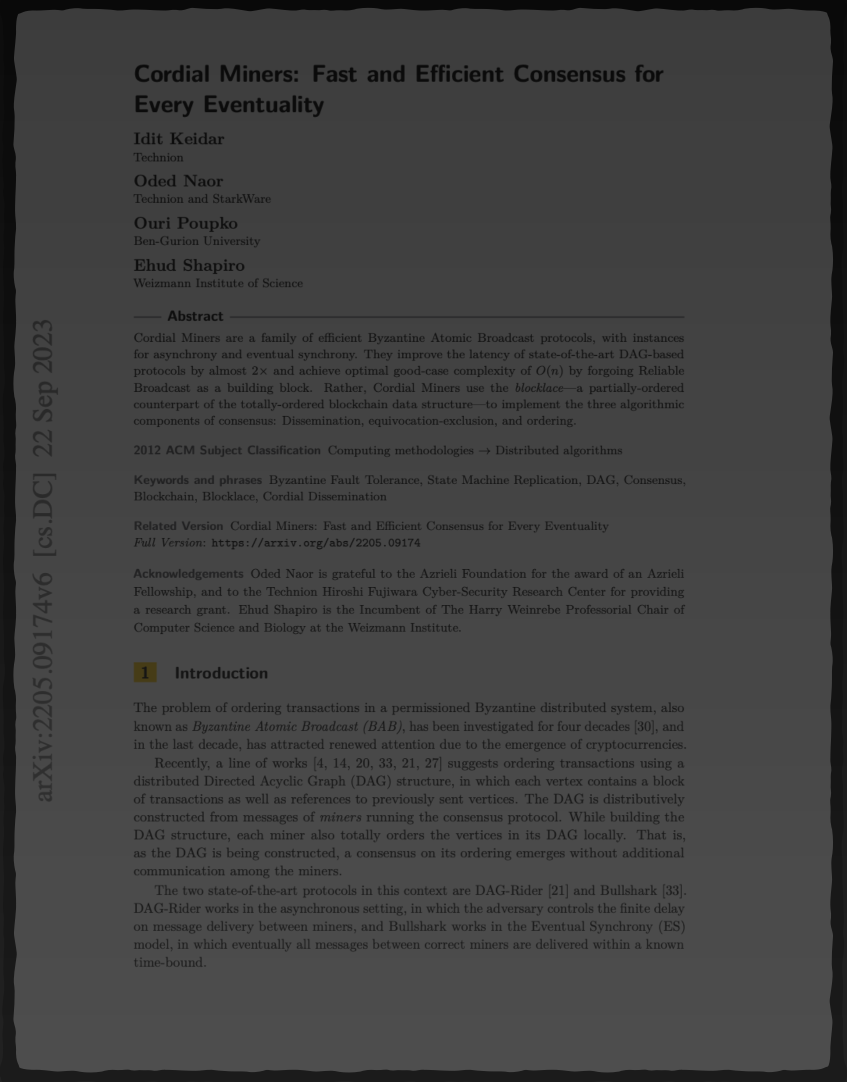


Shoal: Improving DAG-BFT Latency And Robustness

Alexander Spiegelman, Balaji Arun, Rati Gelashvili, Zekun Li

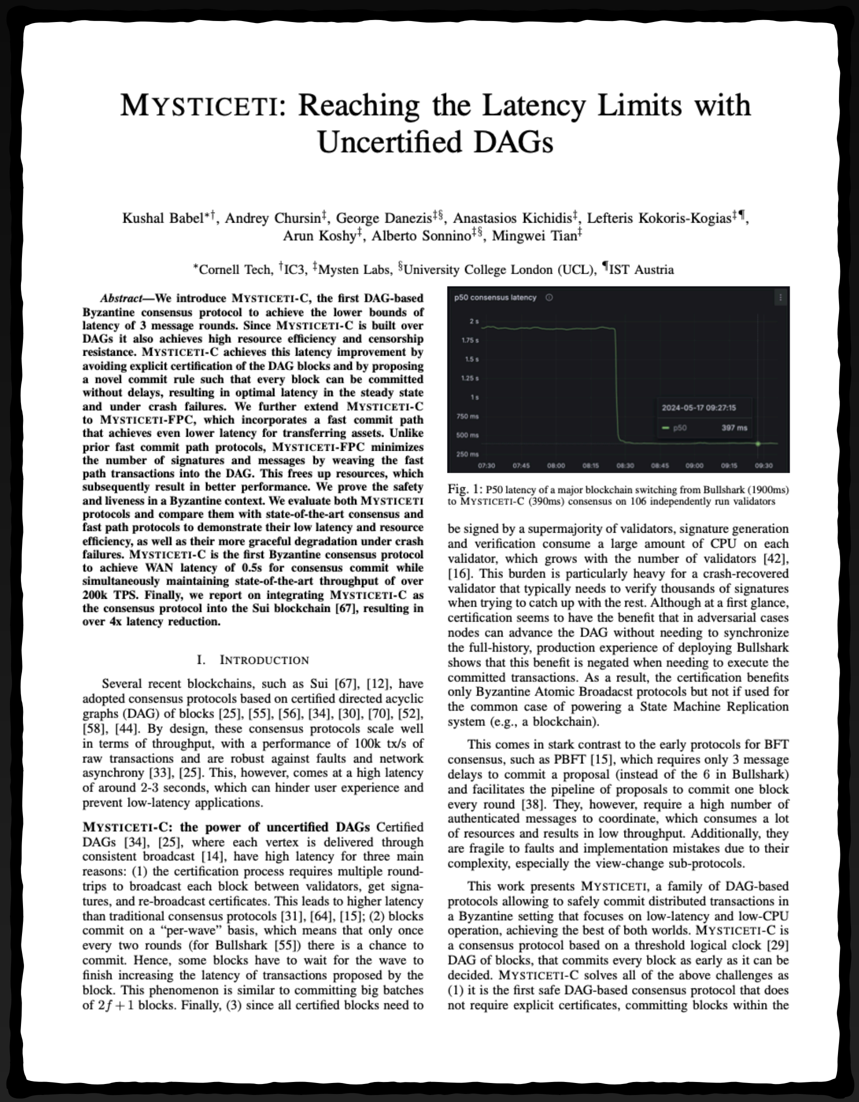
Sailfish: Towards Improving the Latency of DAG-based BFT

Nibesh Shrestha, Rohan Shrothrium, Aniket Kate, Kartik Nayak



Cordial Miners: Fast and Efficient Consensus for Every Eventuality

Idit Keidar, Oded Naor, Ouri Poupko, Ehud Shapiro

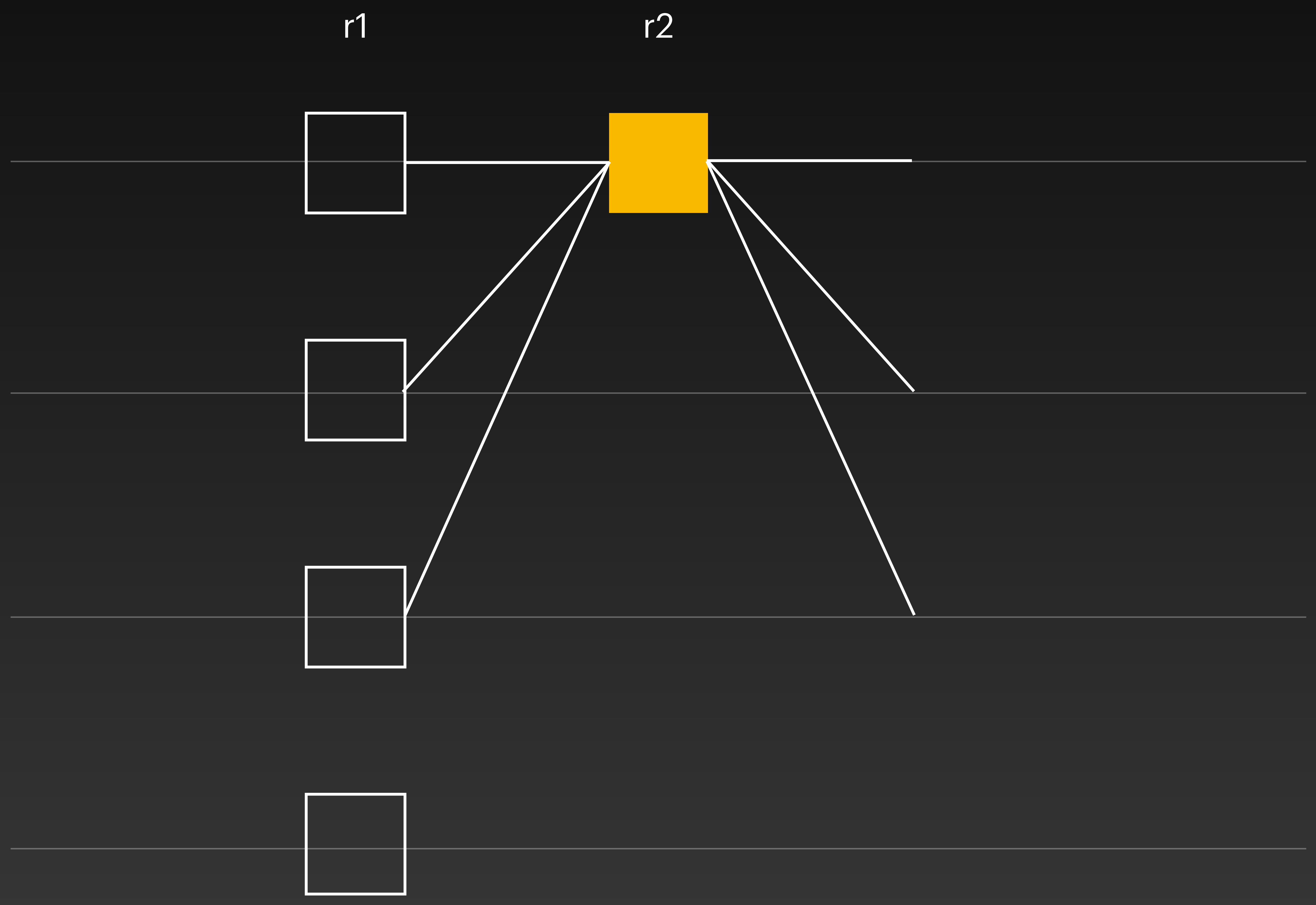


MYSTICETI: Reaching the Latency Limits with Uncertified DAGs

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

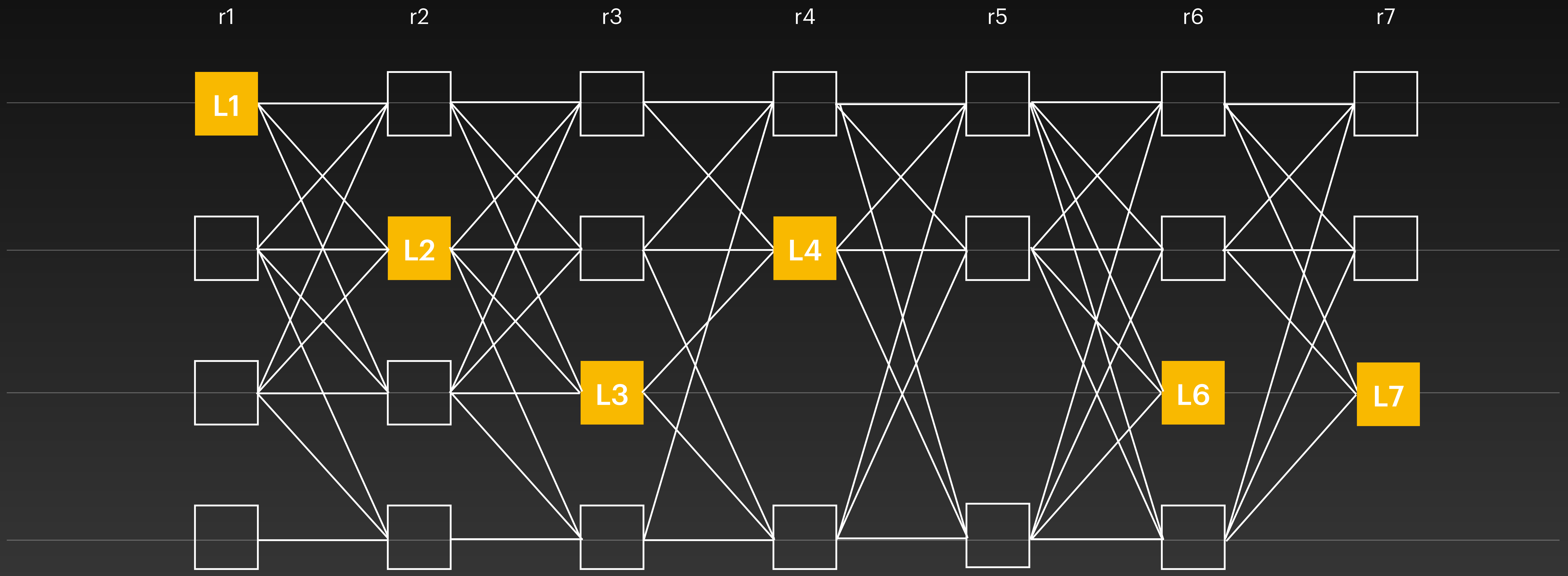
Cornell Tech, IC3, Mysten Labs, University College London (UCL), IST Austria

# Uncertified DAG

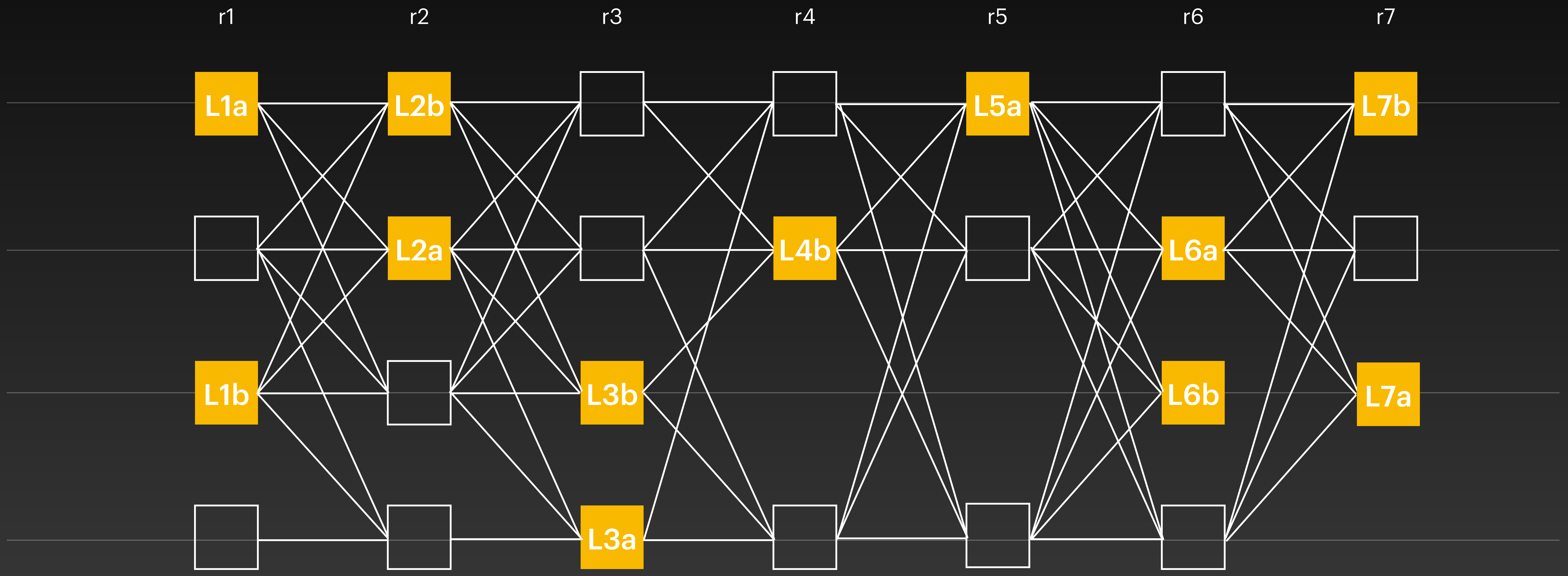



- Round number
- Author
- Payload (transactions)
- Signature

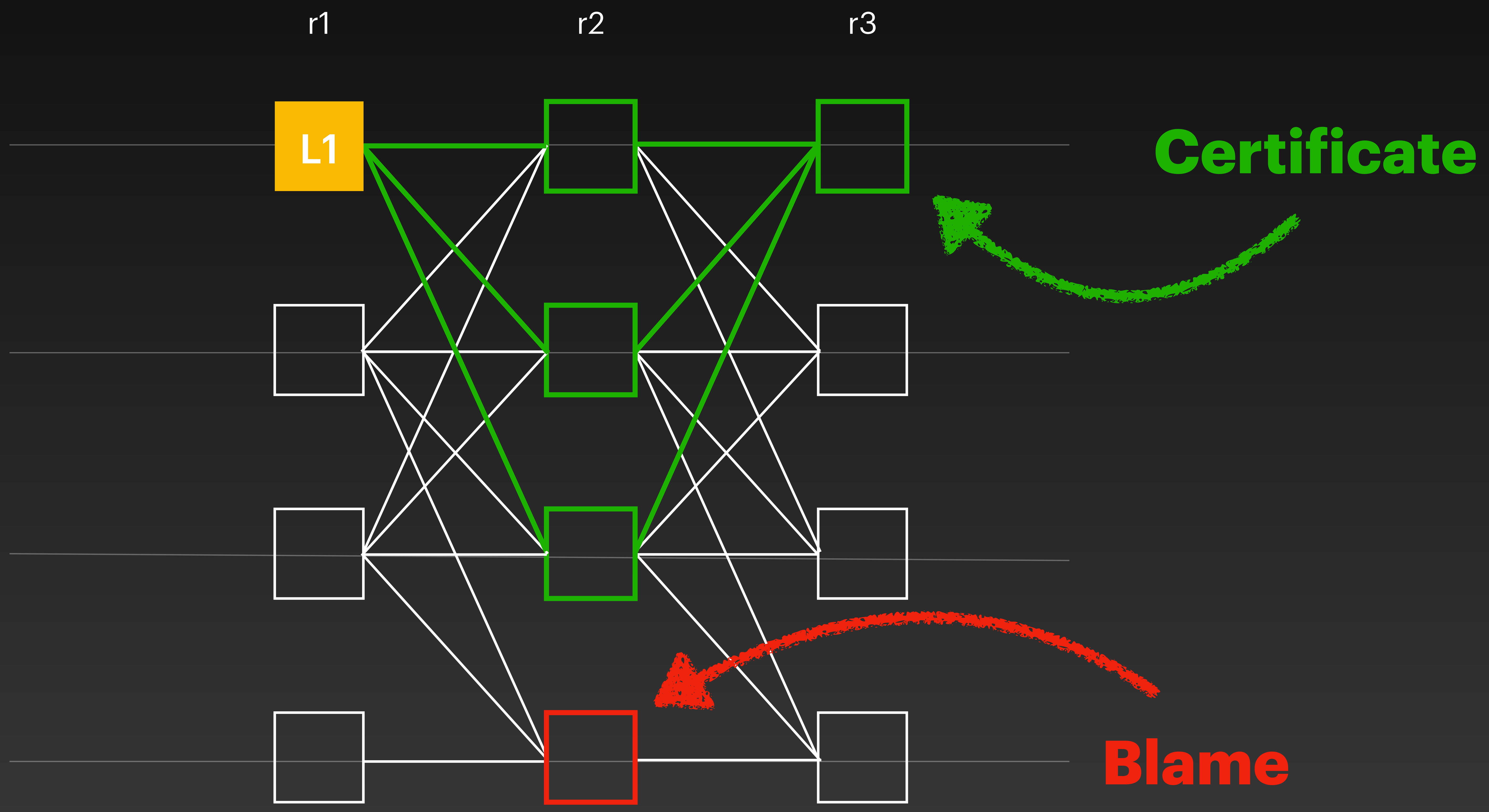
# Uncertified DAG



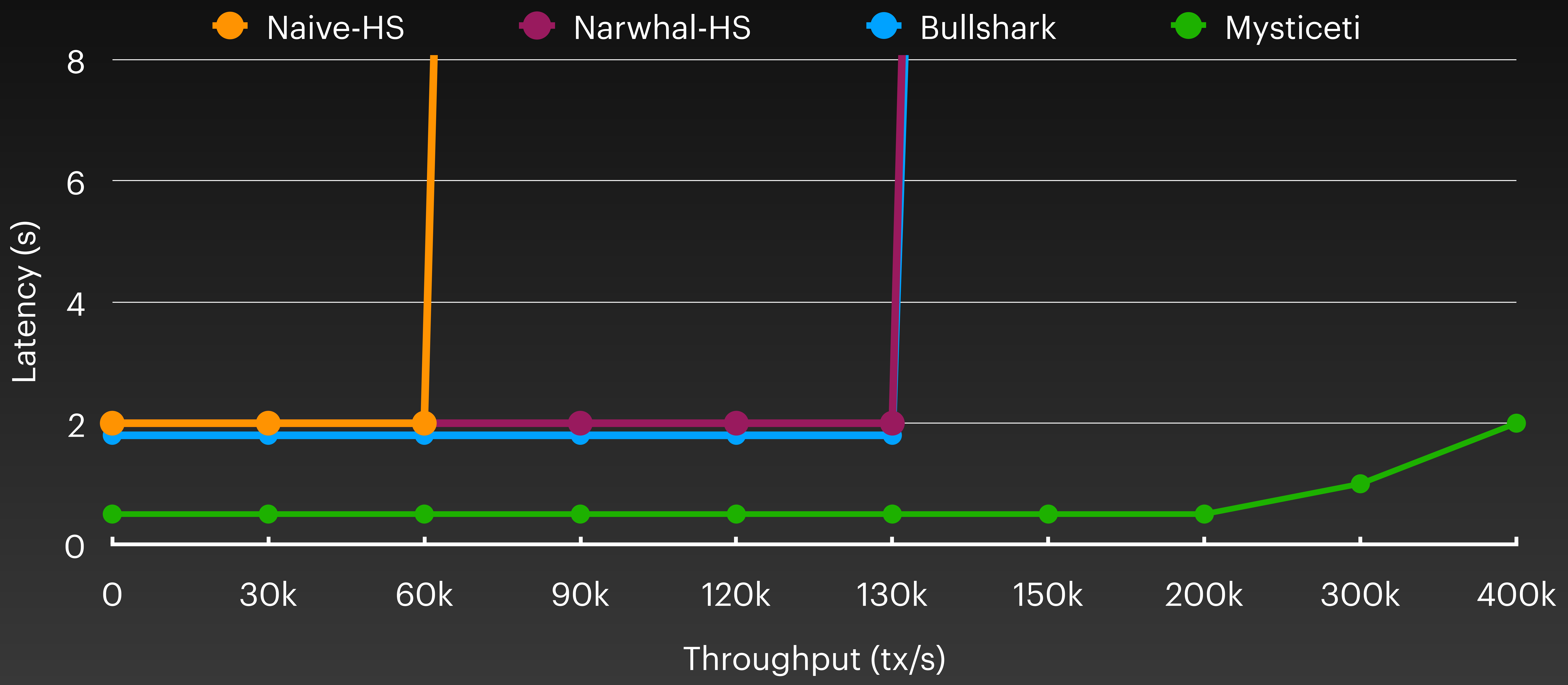
# Uncertified DAG



# Interpreting DAG Patterns



# Performance



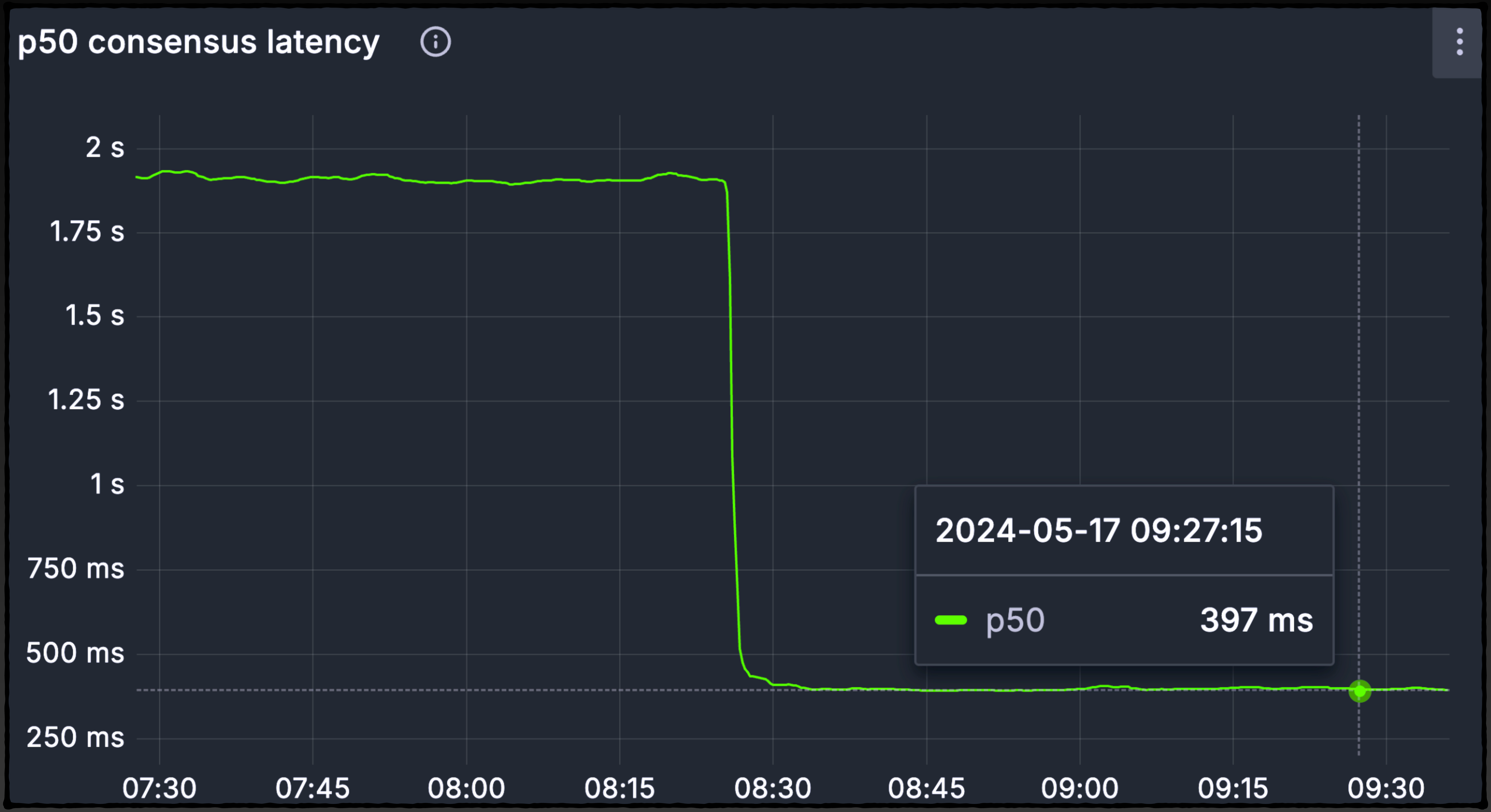
# Research Questions

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

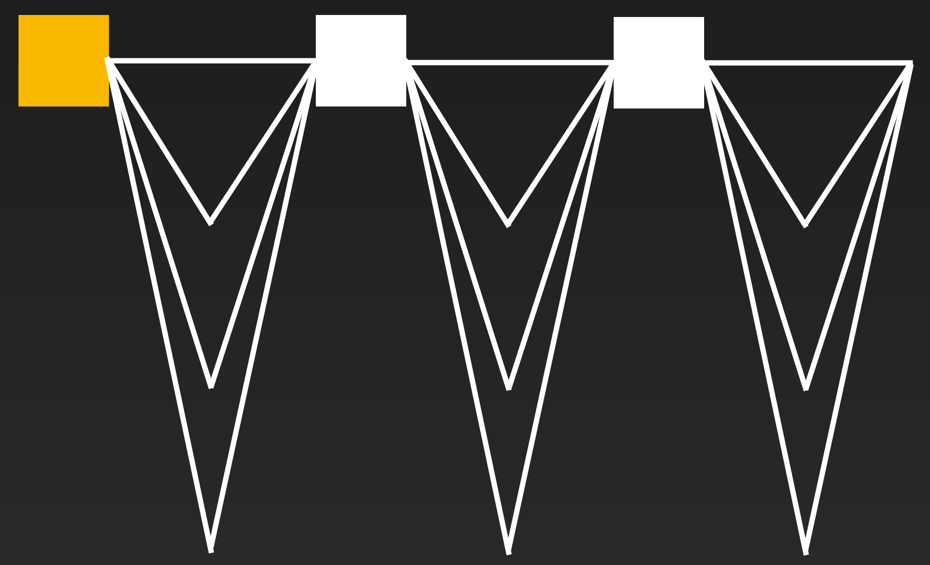
# The Sui Mainnet



# The Roadmap

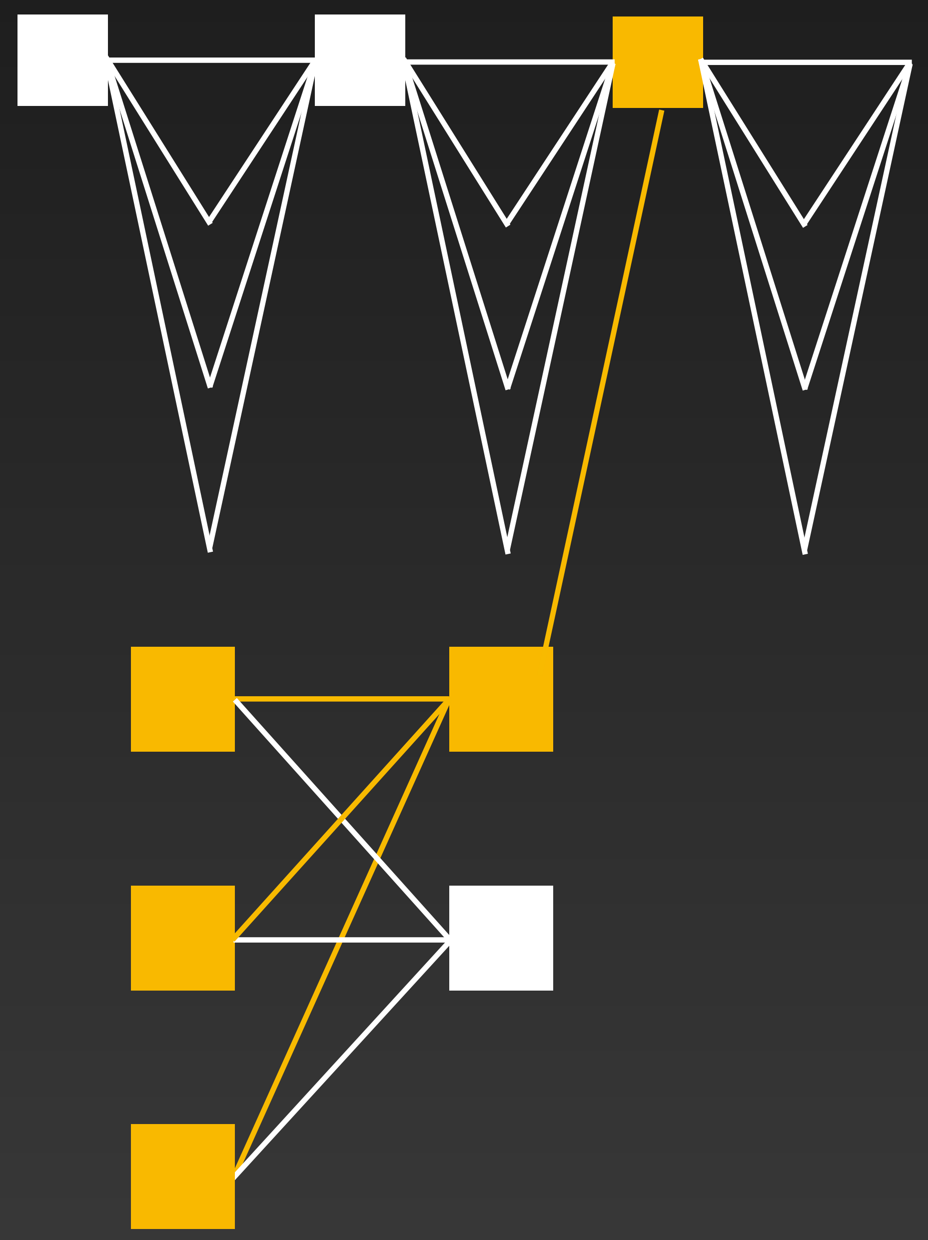
**2019**

**naive consensus**



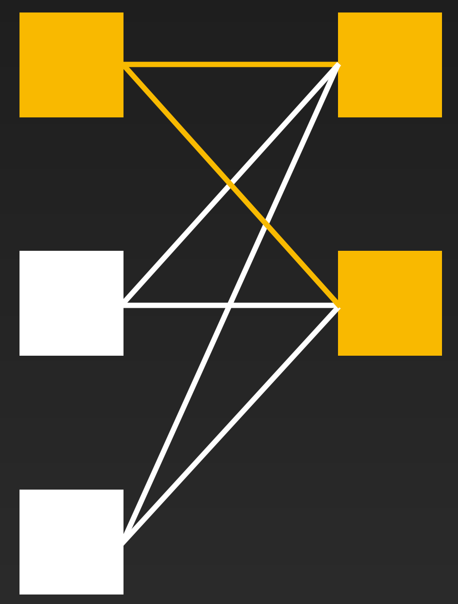
**2020-2021**

**mempool ❤️consensus**



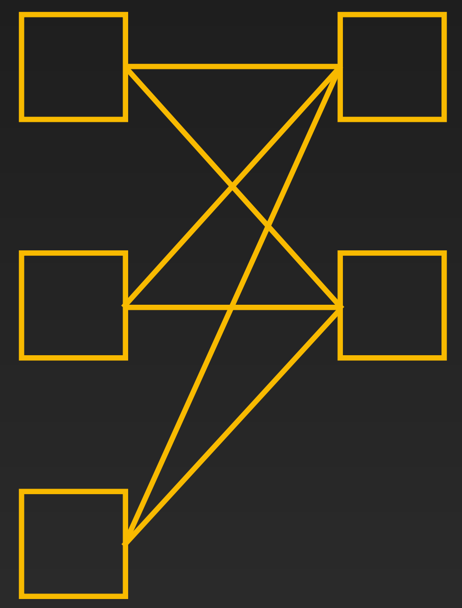
**2022-2023**

**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

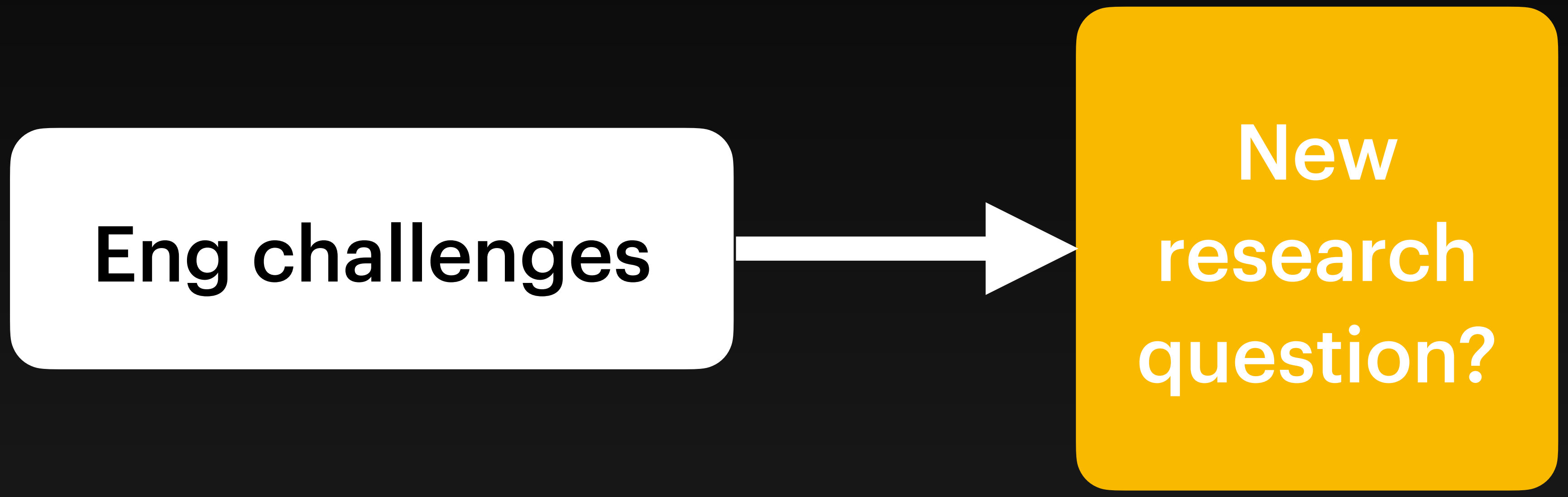
**Thread** # sui-core-internal

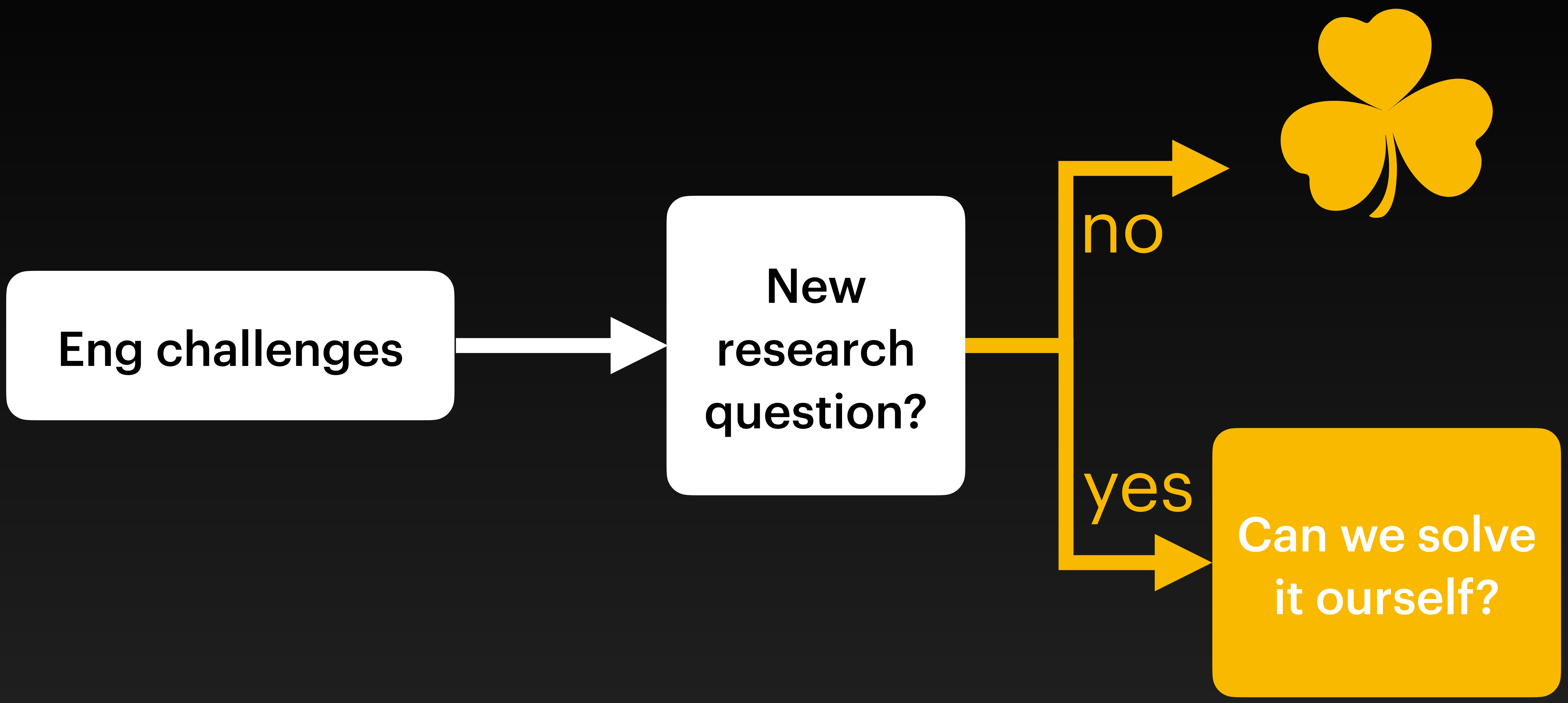
**Dmitri Perelman** Oct 18th at 5:55 AM
In tomorrow's Research <> Core Eng syncup, @Mark Logan is going to share top of mind of Core Eng pain points and current struggles. See you 🙌
👍 2

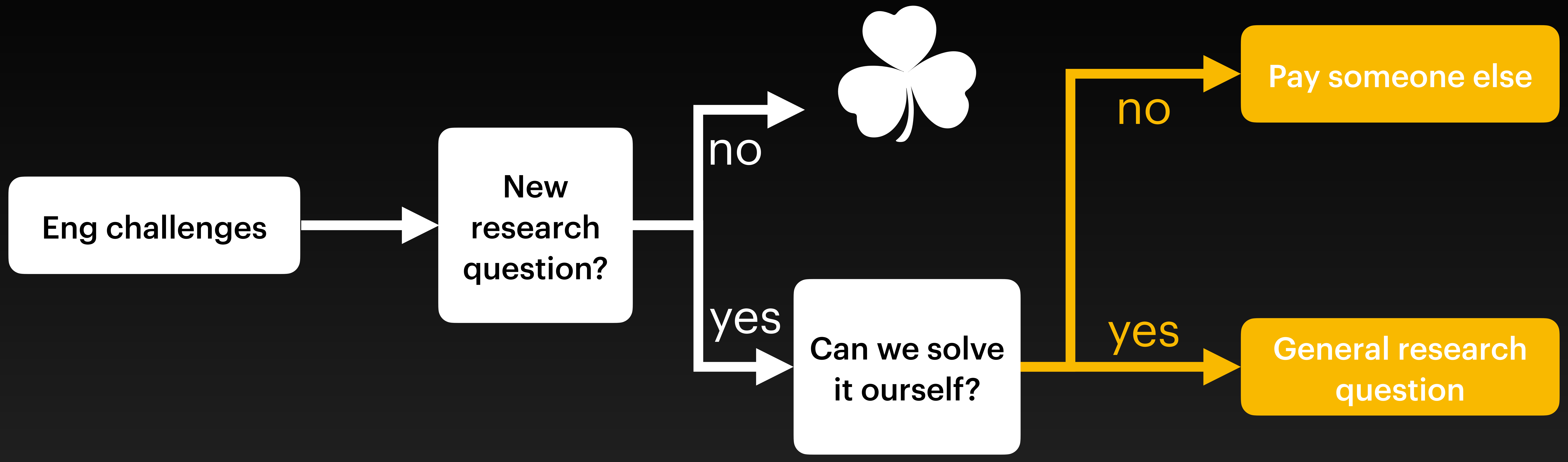
2 replies

**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!
TY 1

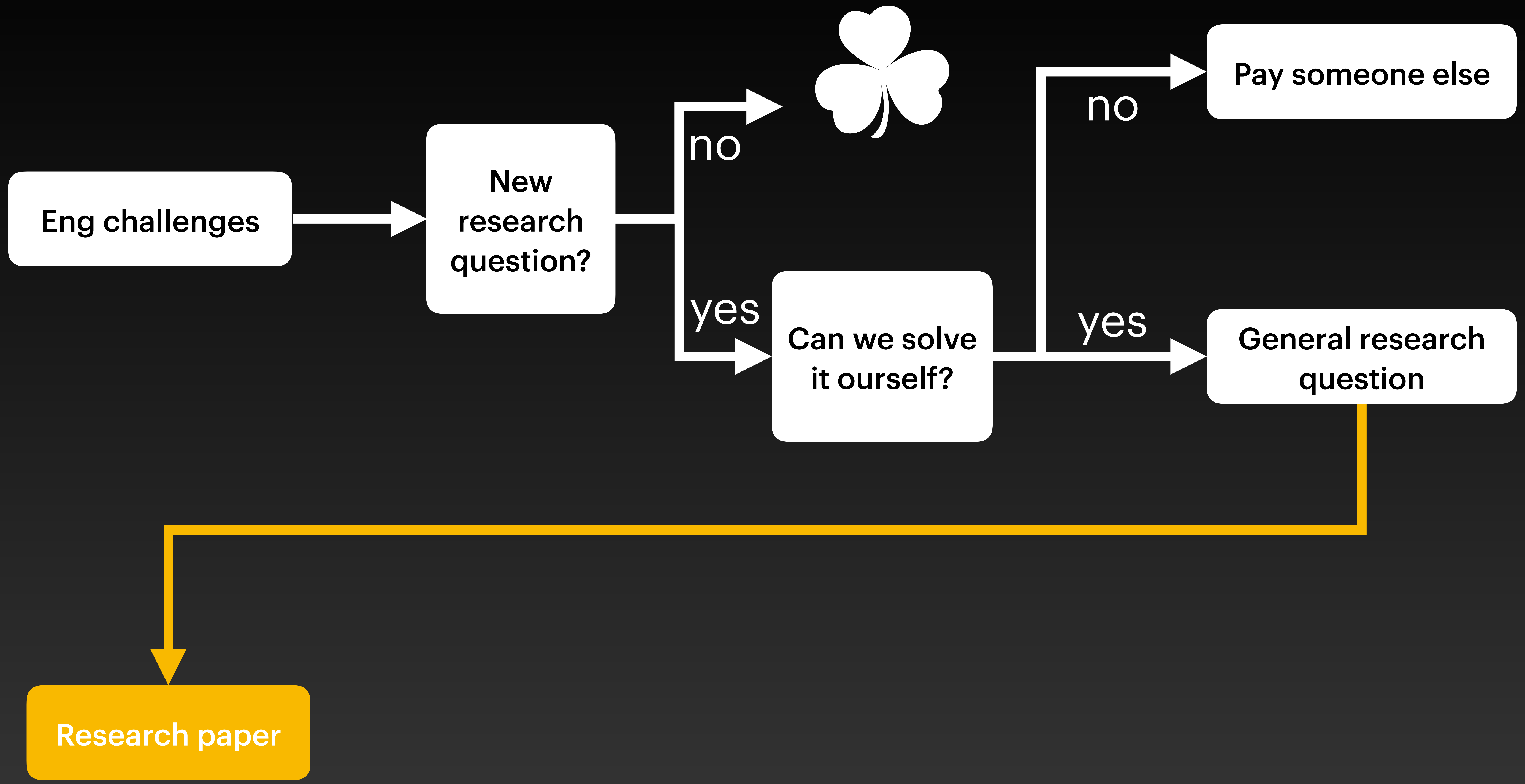

Eng challenges
New research question?

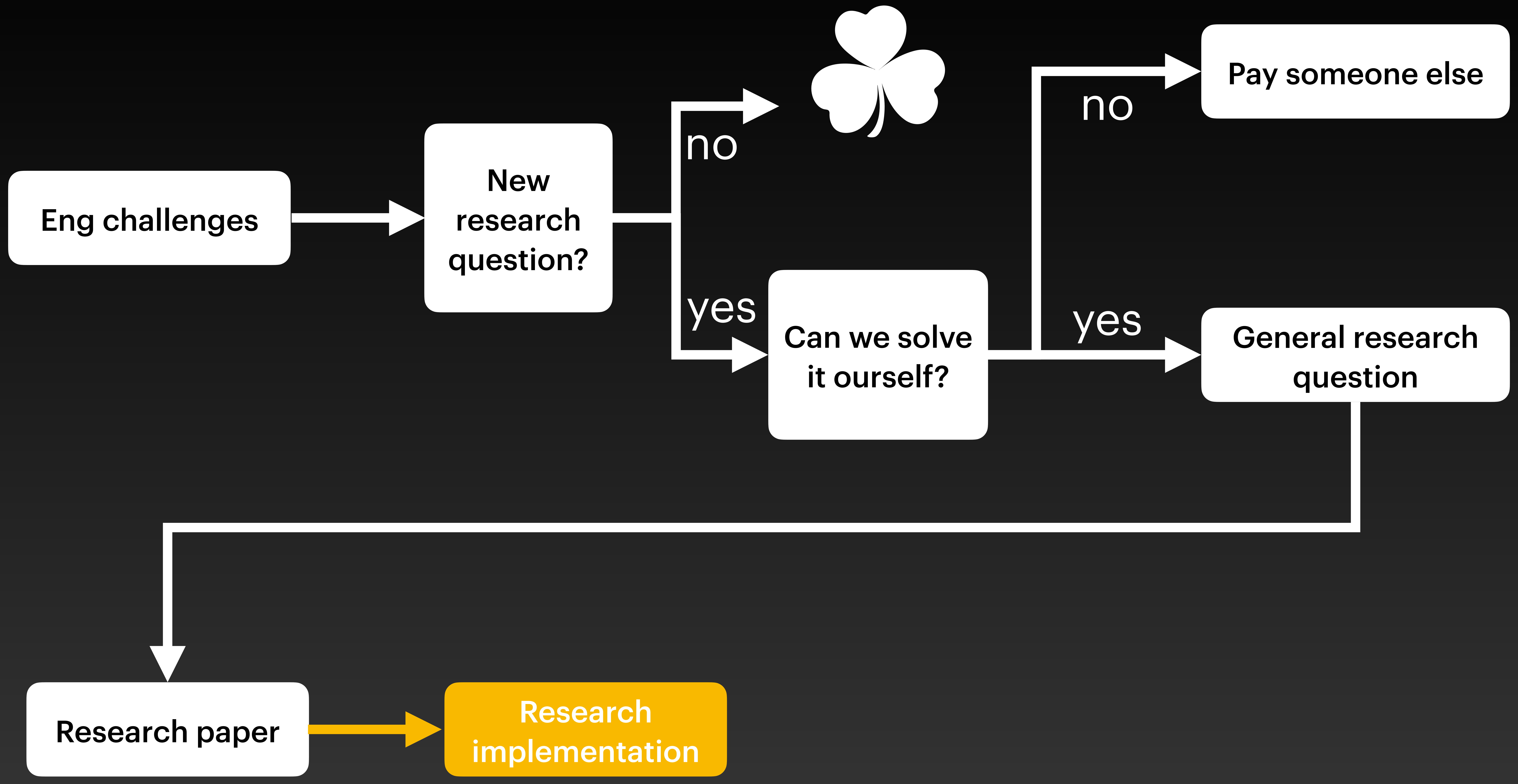

Eng challenges
New research question?
no
yes
Can we solve it ourself?

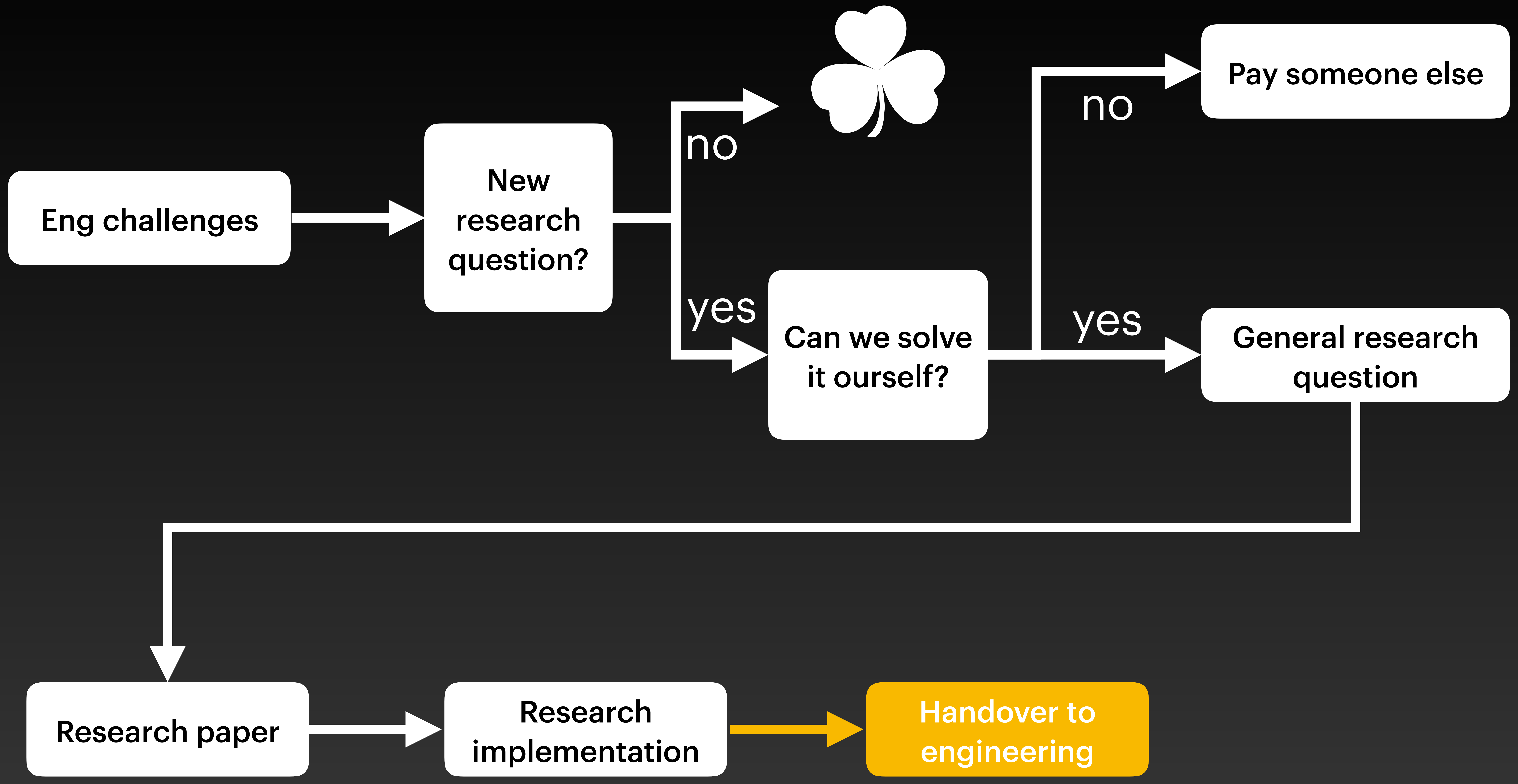

Eng challenges
New research question?
no
yes
Can we solve it ourself?
no
Pay someone else
yes
General research question

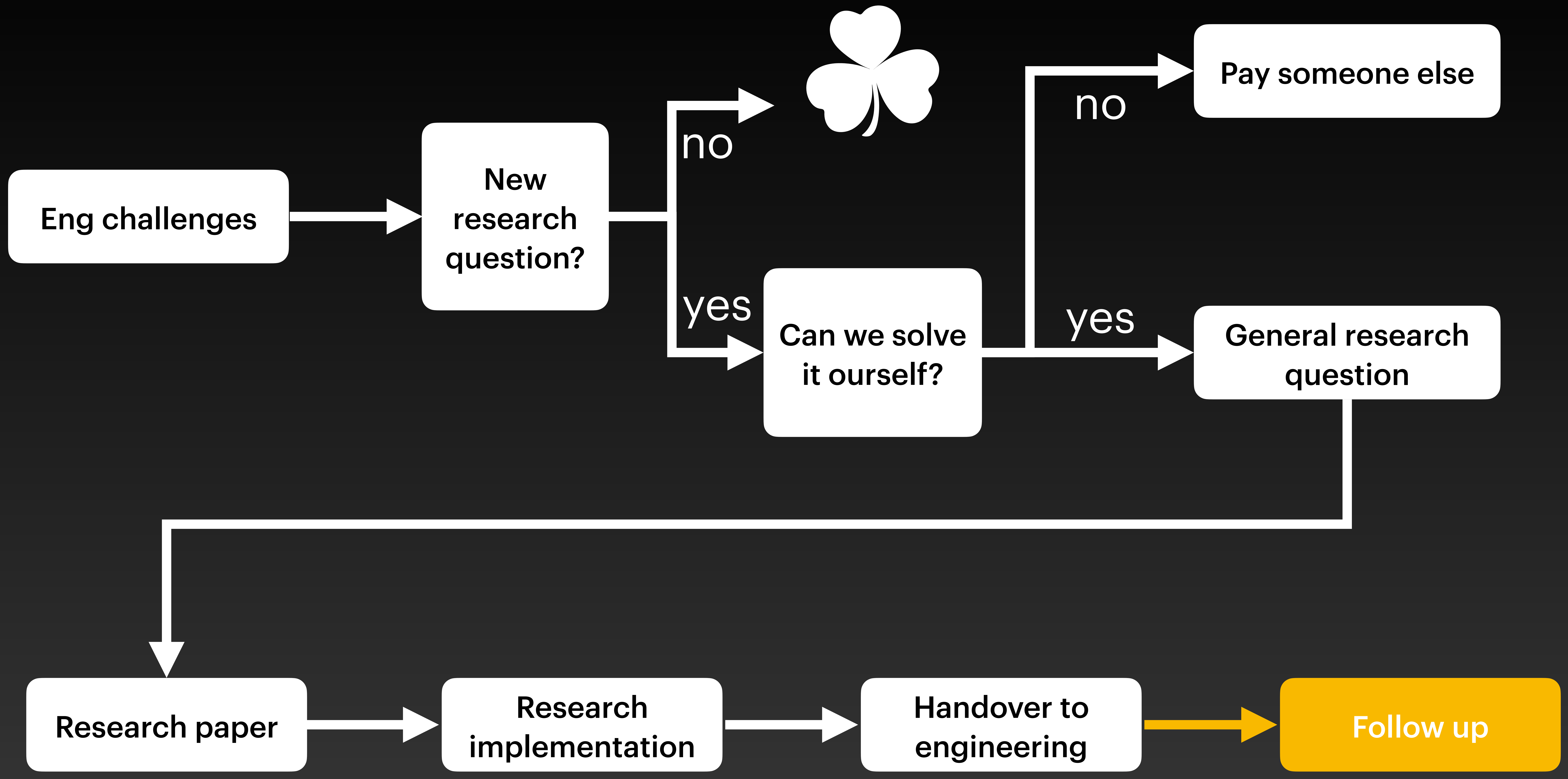
# Research Gifts

(please keep it short)


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


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


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


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

## 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**

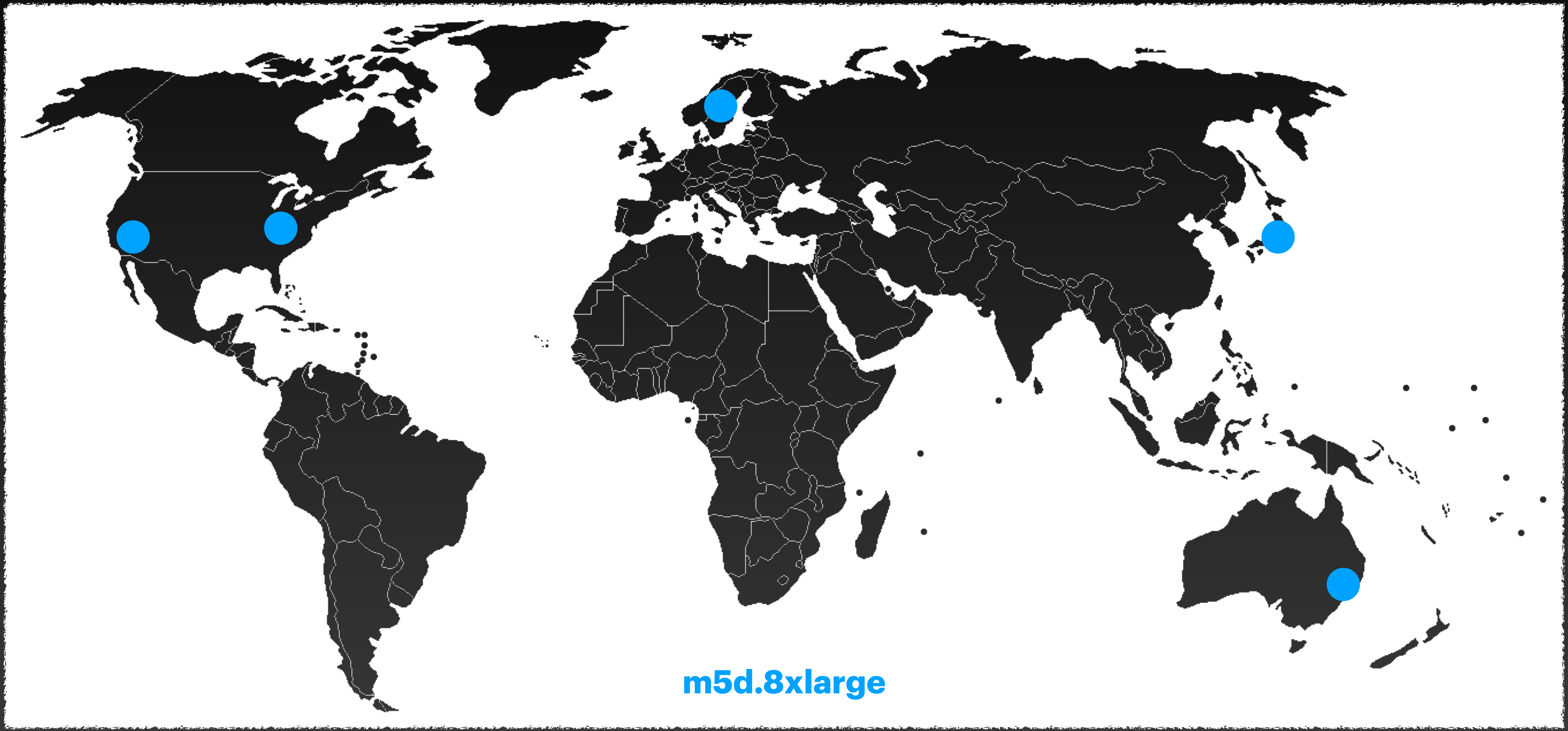
# Implementation

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

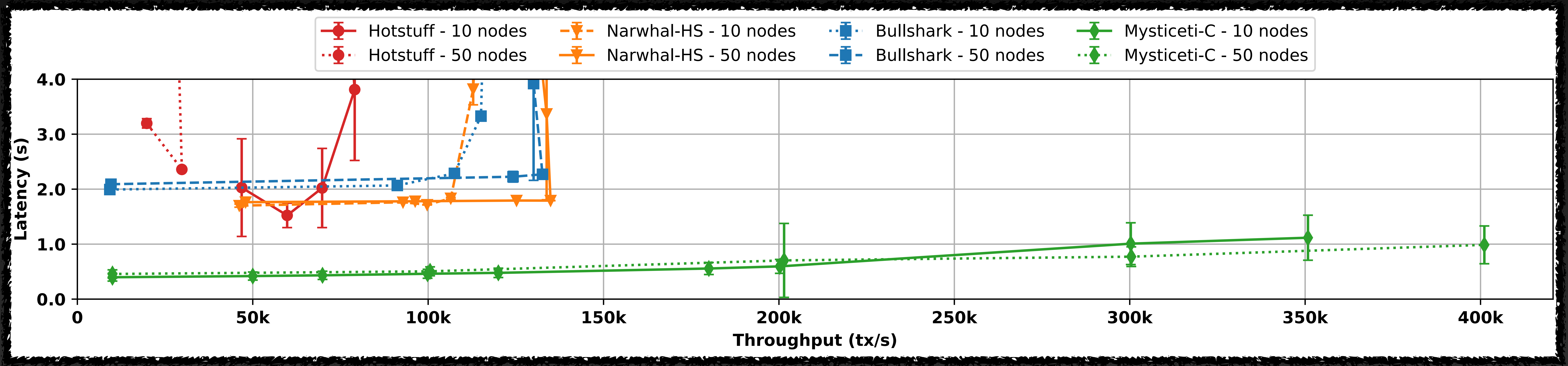
**https://github.com/mystenlabs/mysticeti**

# Evaluation

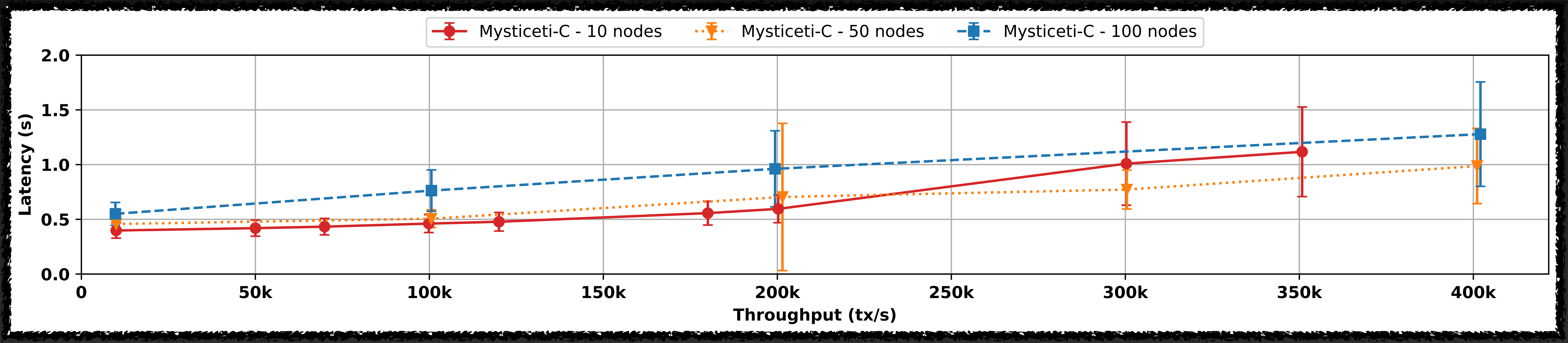
## Experimental setup on AWS



# Prototype Benchmarks



# Prototype Benchmarks



# Mysticeti

## Key Limitations

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