Elastic Scaling Web3

Lefteris Kokoris Kogias — Mysten Labs

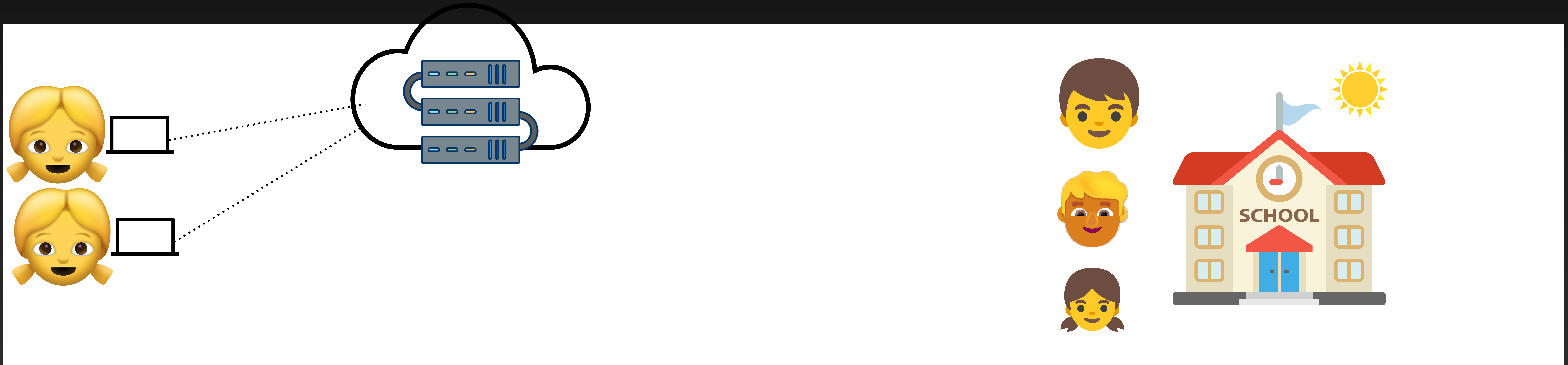
Elastic Scaling

- **Elastic scalability** *is the ability of a system to dynamically adjust its resource usage based on workload demands.*

Example

Video Streaming Lectures

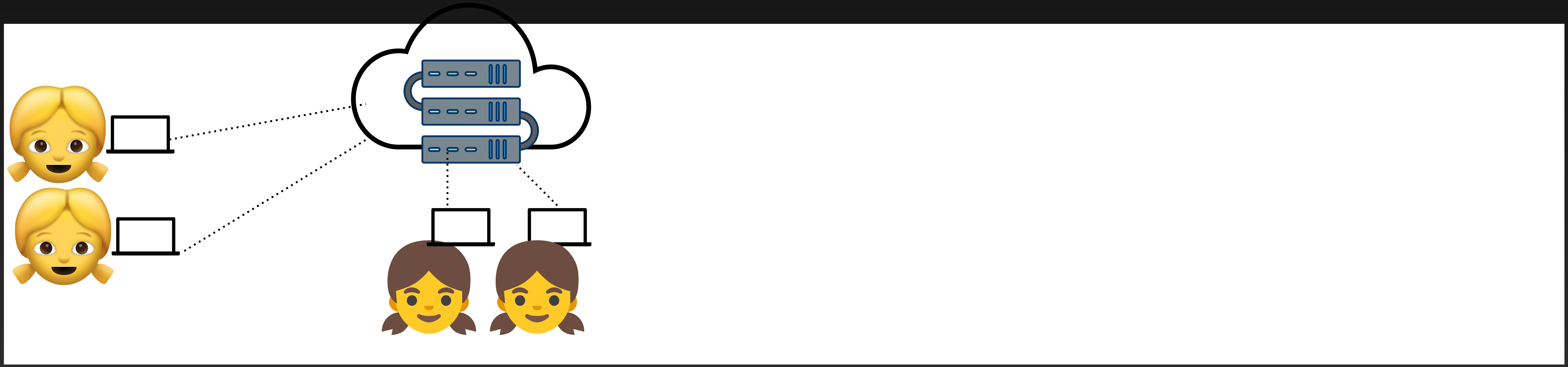
Normal Days



Example

Video Streaming Lectures

Exam Period

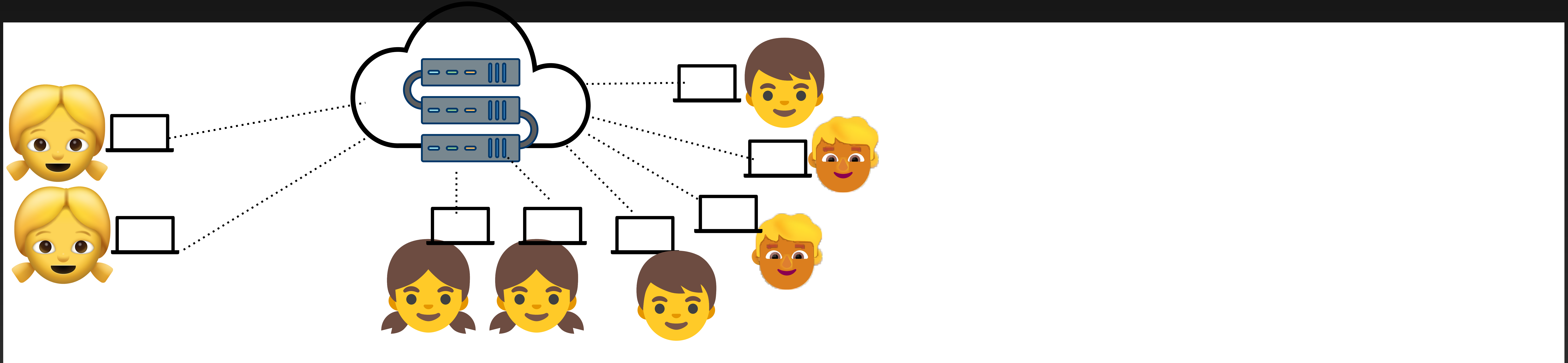


- Video is lower quality
- Students disconnect

Example

Video Streaming Lectures

Morning Before the Exam

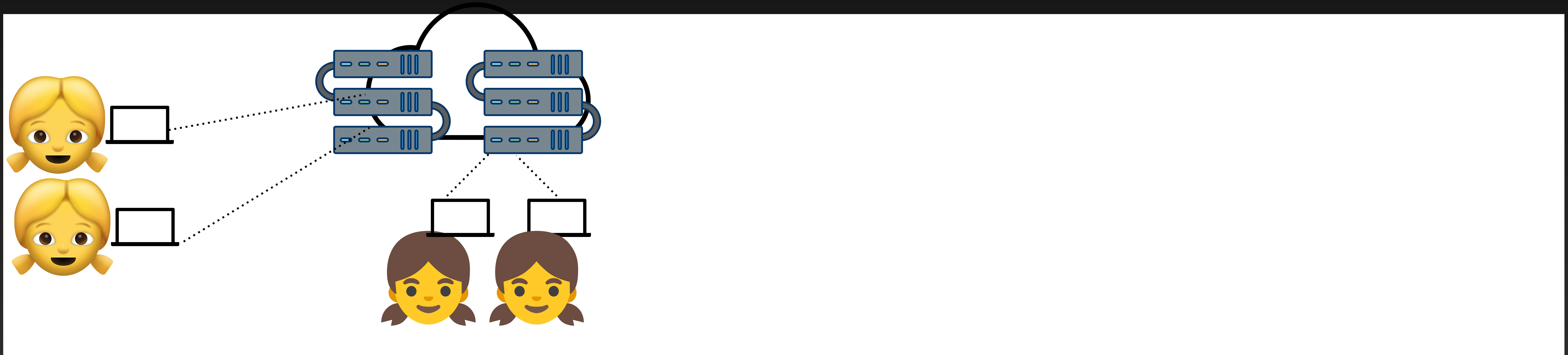


- No Service
- System overload

Elastic Scaling

Video Streaming Lectures

Exam Period

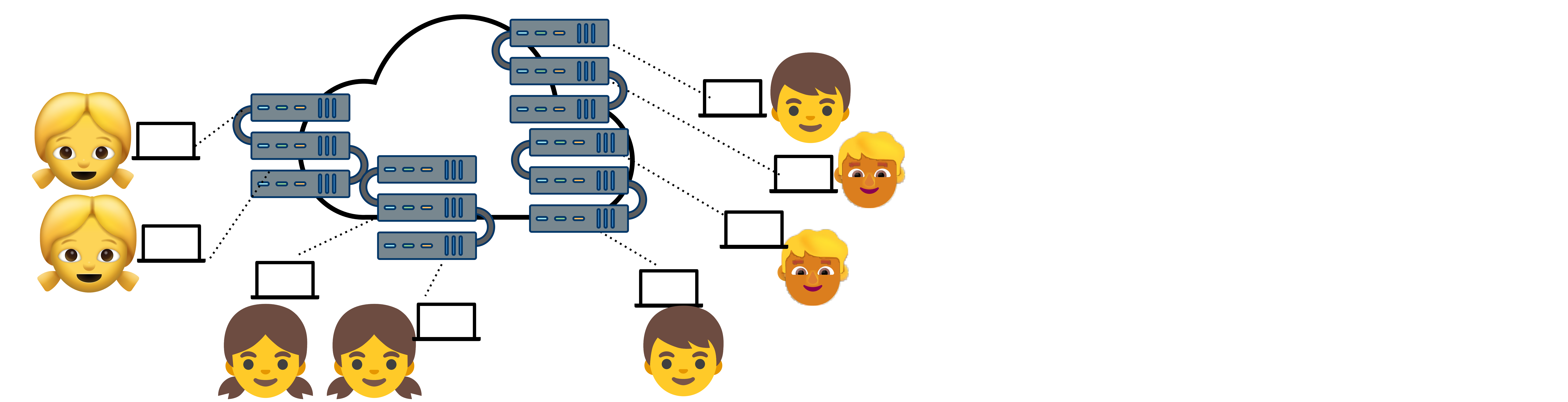


- Add Resources as Demand Increases
- Cost per User remains constant

Elastic Scaling

Video Streaming Lectures

Morning Before the Exam



- Smooth User Experience
- Better load balance even in the presence of faults

Key Components For Elastic Scaling

Autoscaling: Automatically adjust the number of compute resources based on workload demands

Load Balancing: Distributes incoming traffic across multiple instances to ensure optimal resource utilization and performance.

Benefits of Elastic Scaling

Cost Efficiency: Pay only for the resources used, minimizing idle capacity

Performance: Maintain consistent performance levels during peak and off-peak periods.

The state of Web3

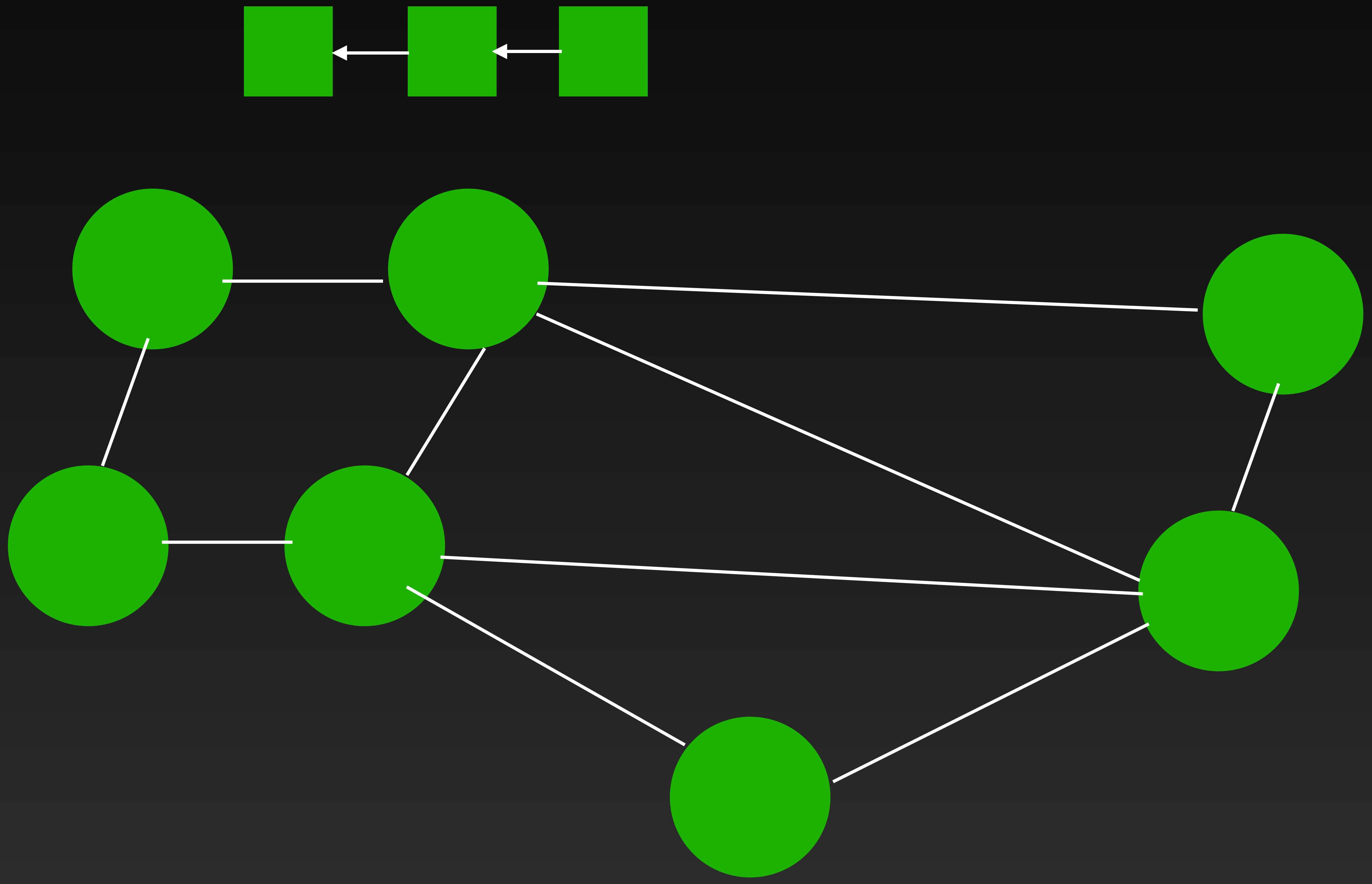
- Minimum Validator Requirements are high
  - Handle load spikes
  - High cost
  - Downward spiral when the load is low —> Increase fees or bankrupt
    1. Invest in a powerful machine to be ready to handle spikes
    2. Load is low, but the cost of buying and running the machine is constant
    3. Need to charge more per transaction to break even
    4. The marginal utility of transaction drops as fees increase
    5. Load drops further

The state of Web3

- When load is higher than the provisioned machine can handle
  - Fees and cost are no longer linked
  - It is an auction —> Pay the premium or leave
    - Stable in the short term, but leads exit the ecosystem in the long term
  - Huge queuing delays —> Horrible UX
    - Also leads to exit the ecosystem

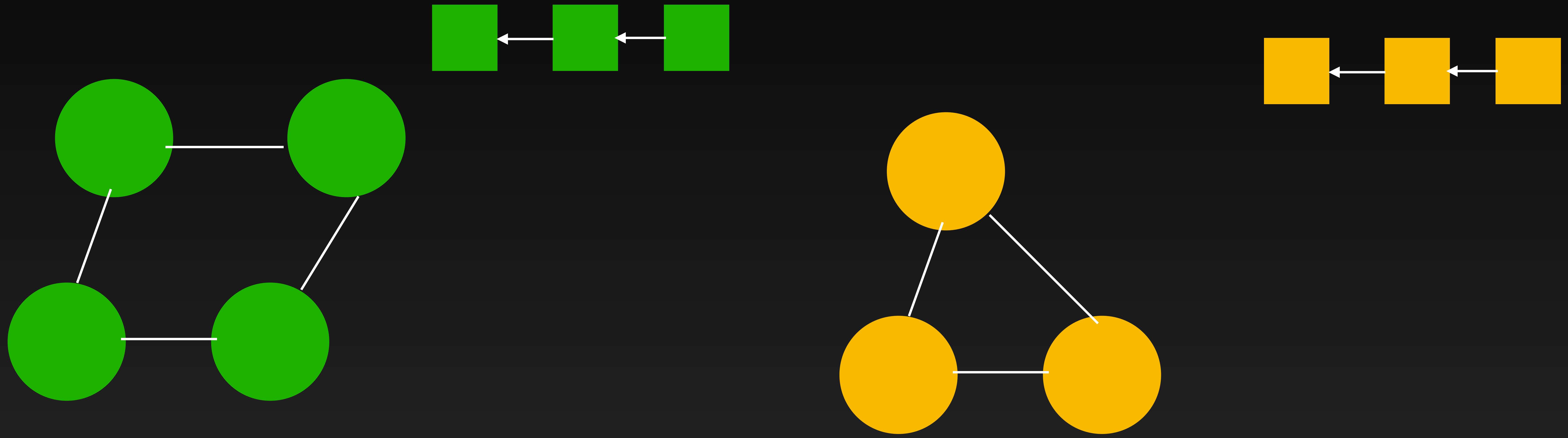
Sharding Blockchains — Design

"Omniledger: A secure, scale-out, decentralized ledger via sharding." IEEE S&P, 2018.



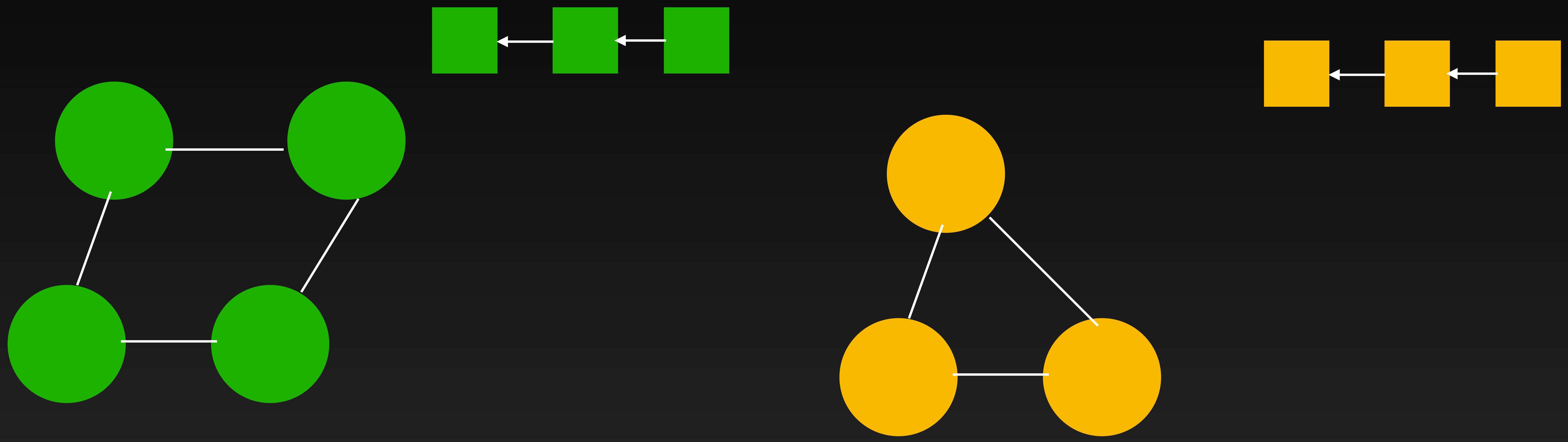
Sharding Blockchains — Design

"Omniledger: A secure, scale-out, decentralized ledger via sharding." IEEE S&P, 2018.



Sharding Blockchains — Properties

"Omniledger: A secure, scale-out, decentralized ledger via sharding." IEEE S&P, 2018.



+ Low Cost per Node

+ Scales-Out

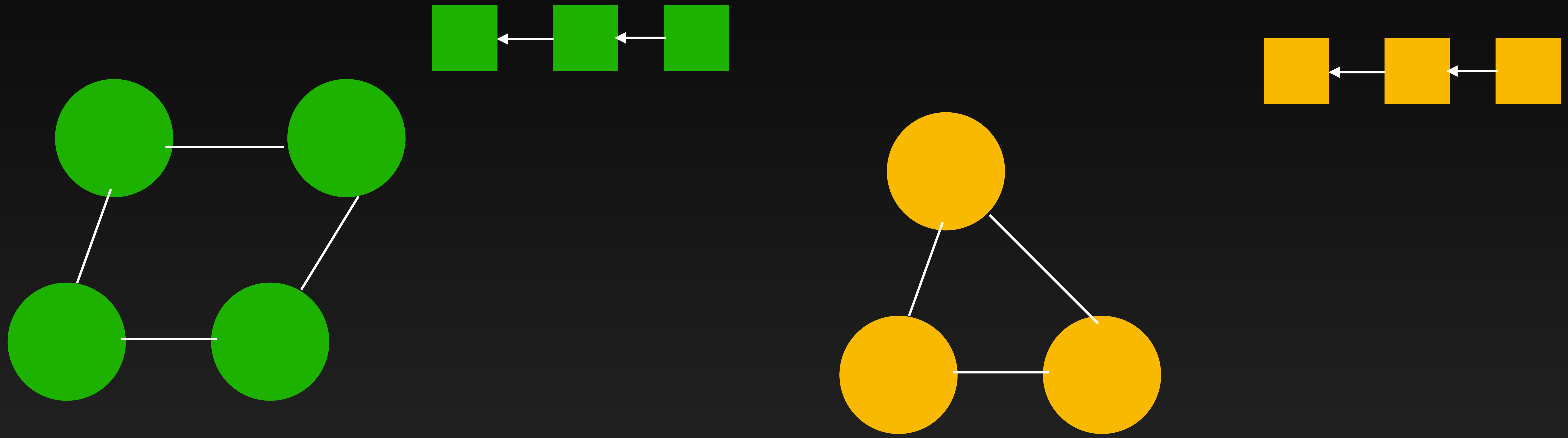
− Fragmenting the state-space — Expensive Atomic Commit

− Susceptible to adaptive adversaries

− Security drop

Sharding Blockchains — Challenges

“Divide and scale: Formalization of distributed ledger sharding protocols” SIROCCO, 2023



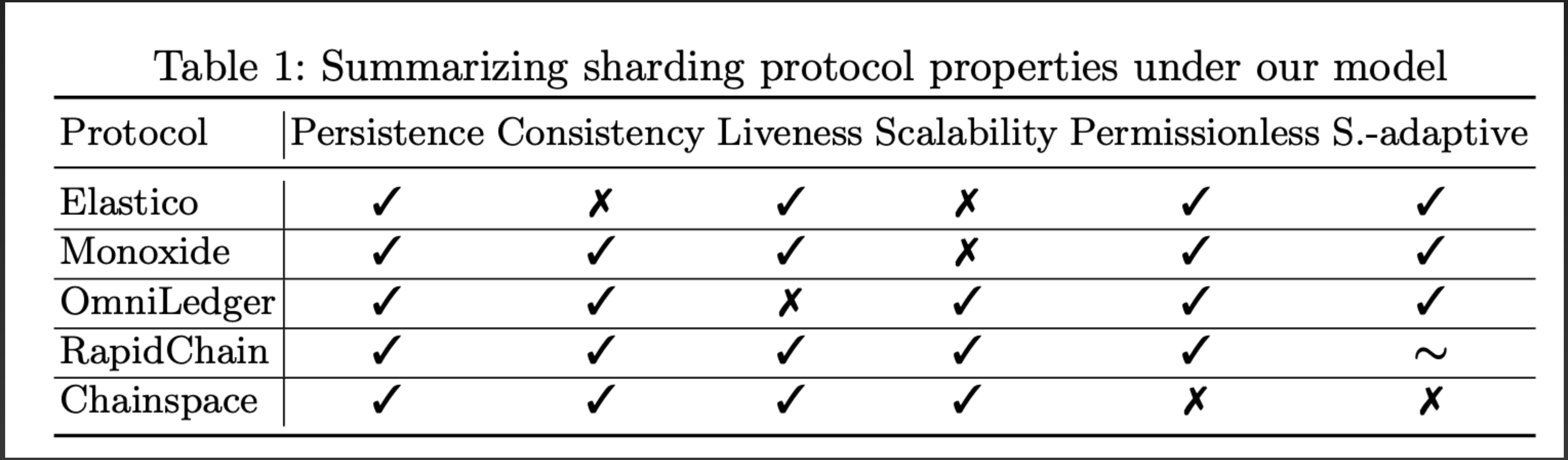
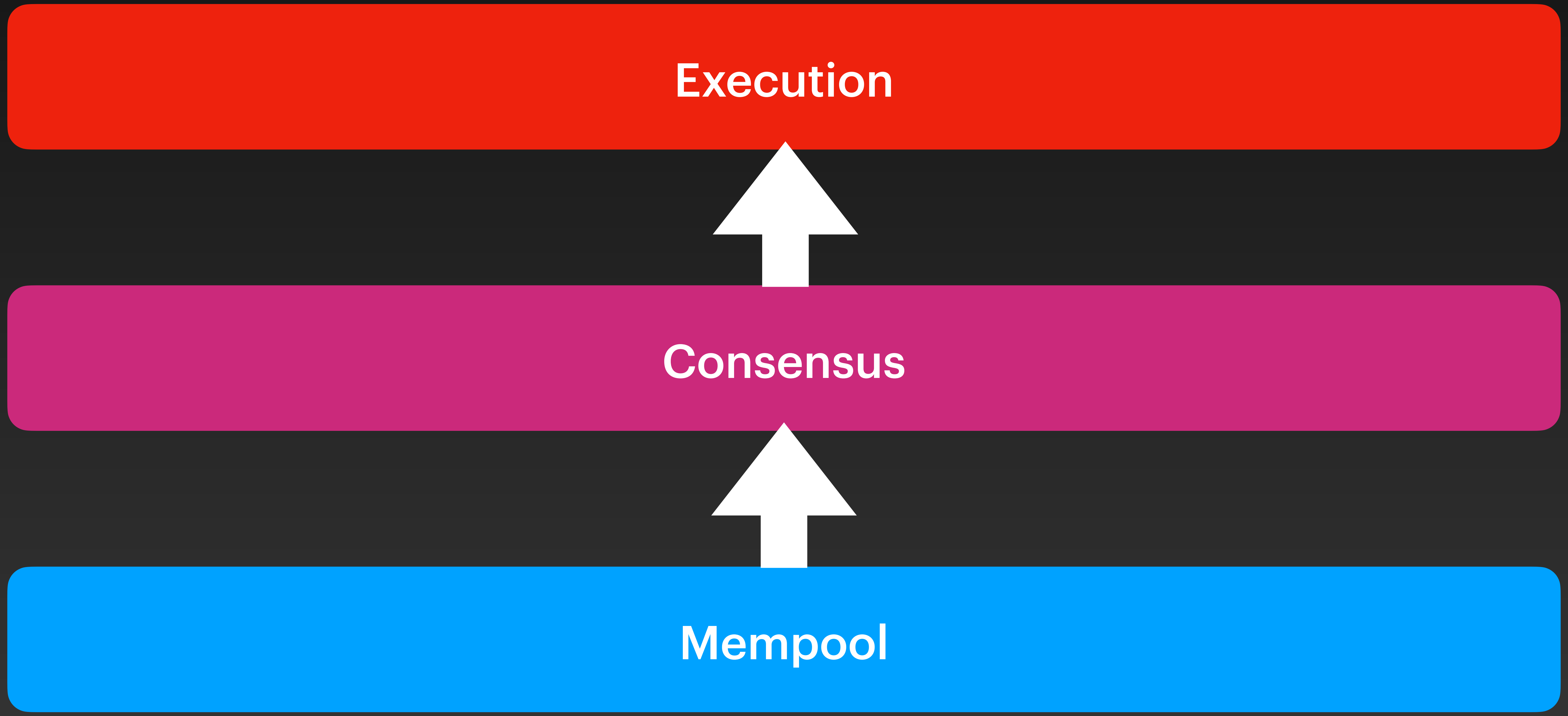


Table 1: Summarizing sharding protocol properties under our model

Protocol	Persistence	Consistency	Liveness	Scalability	Permissionless	S.-adaptive
Elastico	✓	✗	✓	✗	✓	✓
Monoxide	✓	✓	✓	✗	✓	✓
OmniLedger	✓	✓	✗	✓	✓	✓
RapidChain	✓	✓	✓	✓	✓	∼
Chainspace	✓	✓	✓	✓	✗	✗

First Step to the Solution

Layering



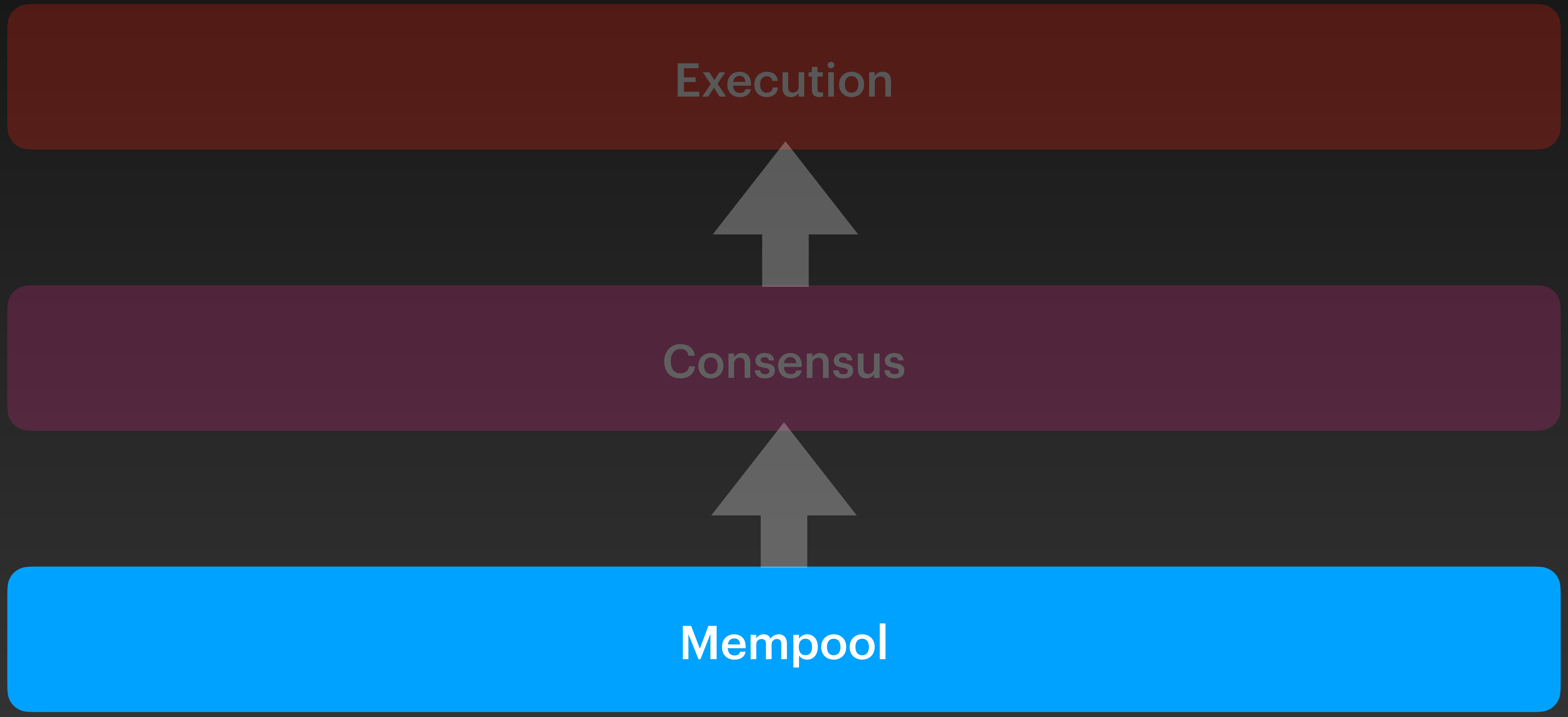
Execution

Consensus

Mempool

First Step to the Solution

Layering



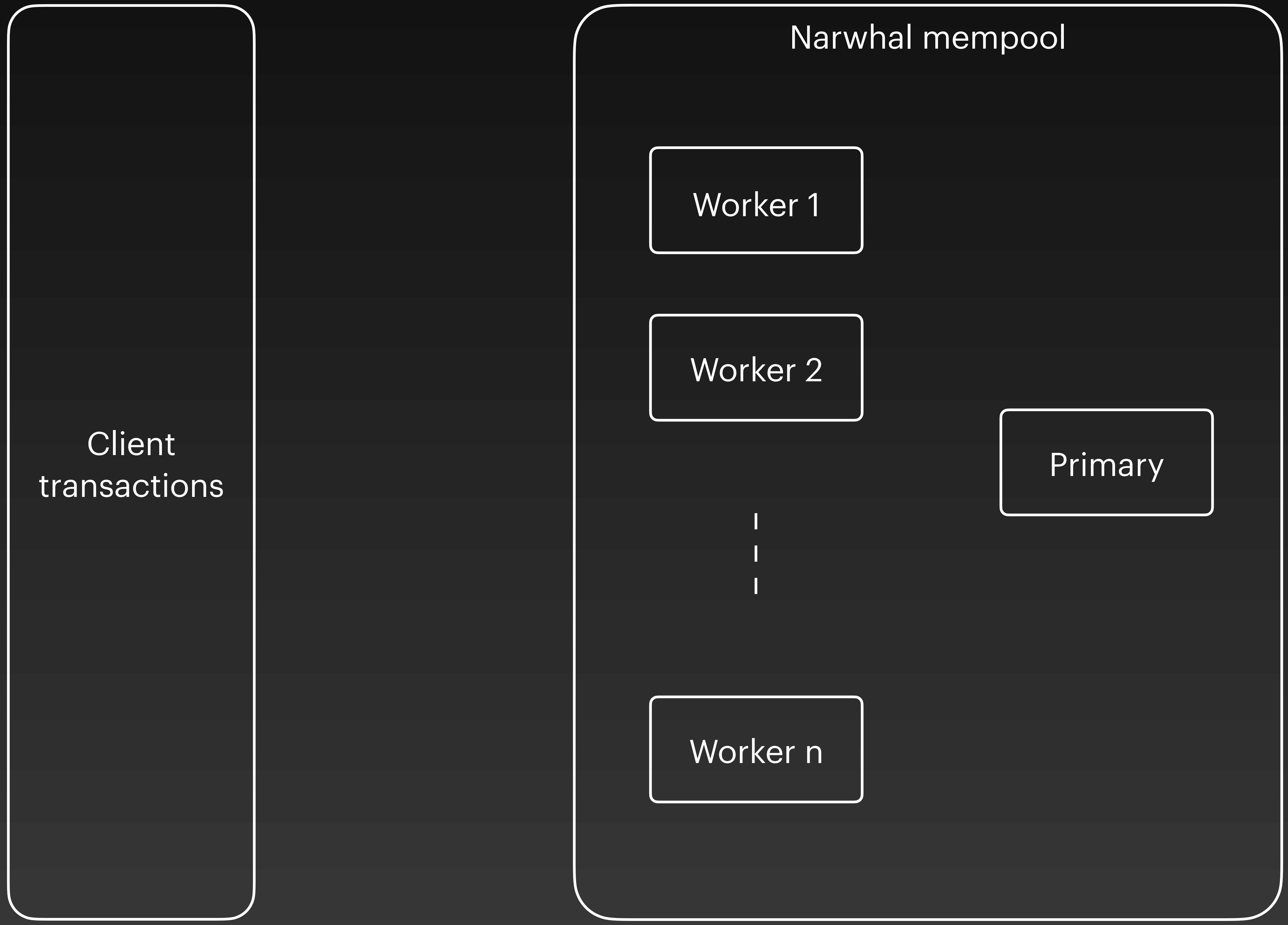
Narwhal

Dag-based mempool

“Narwhal and tusk: a dag-based mempool and efficient bft consensus.” *EuroSys 2022*

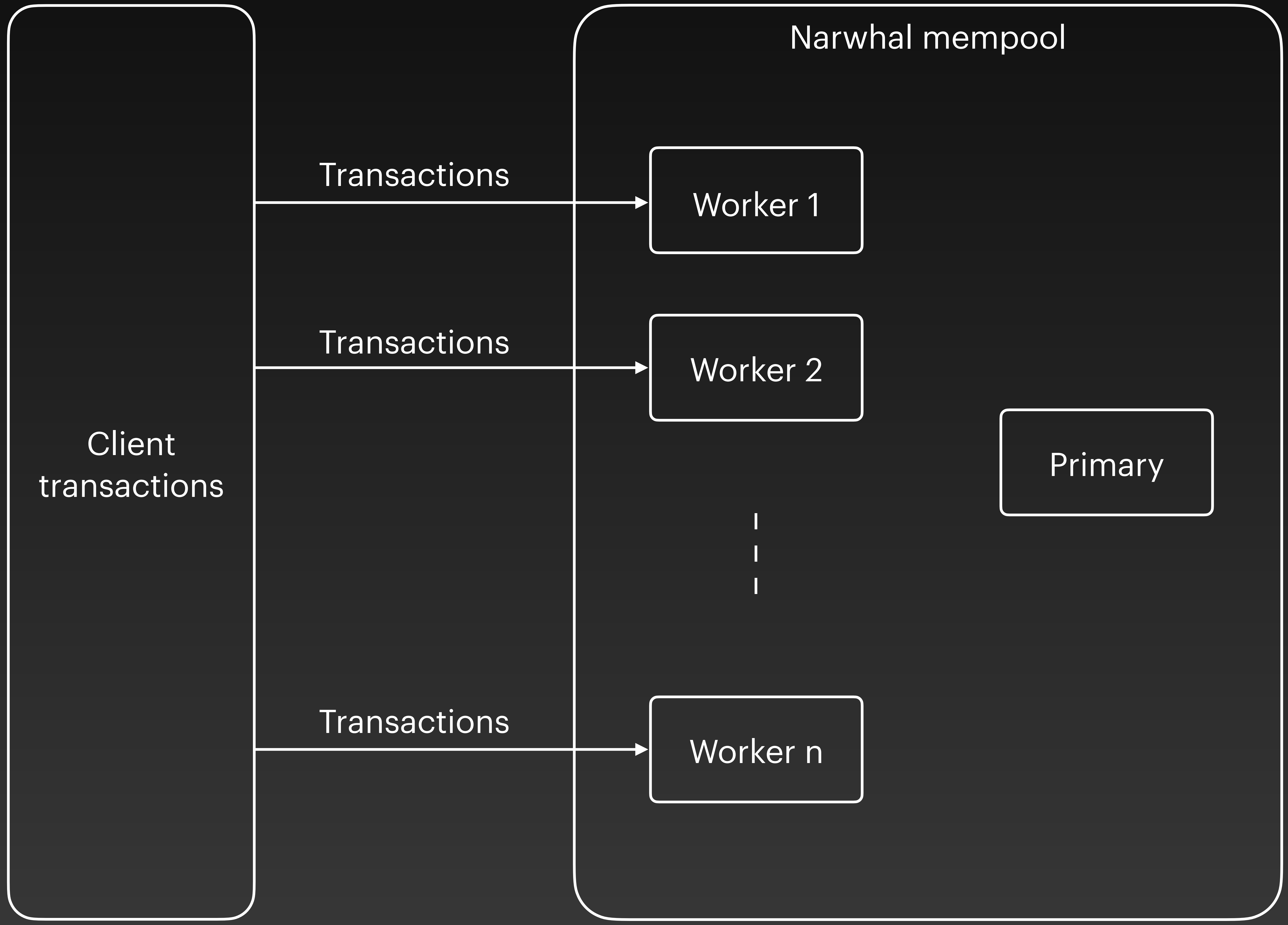
Narwhal

The workers and the primary



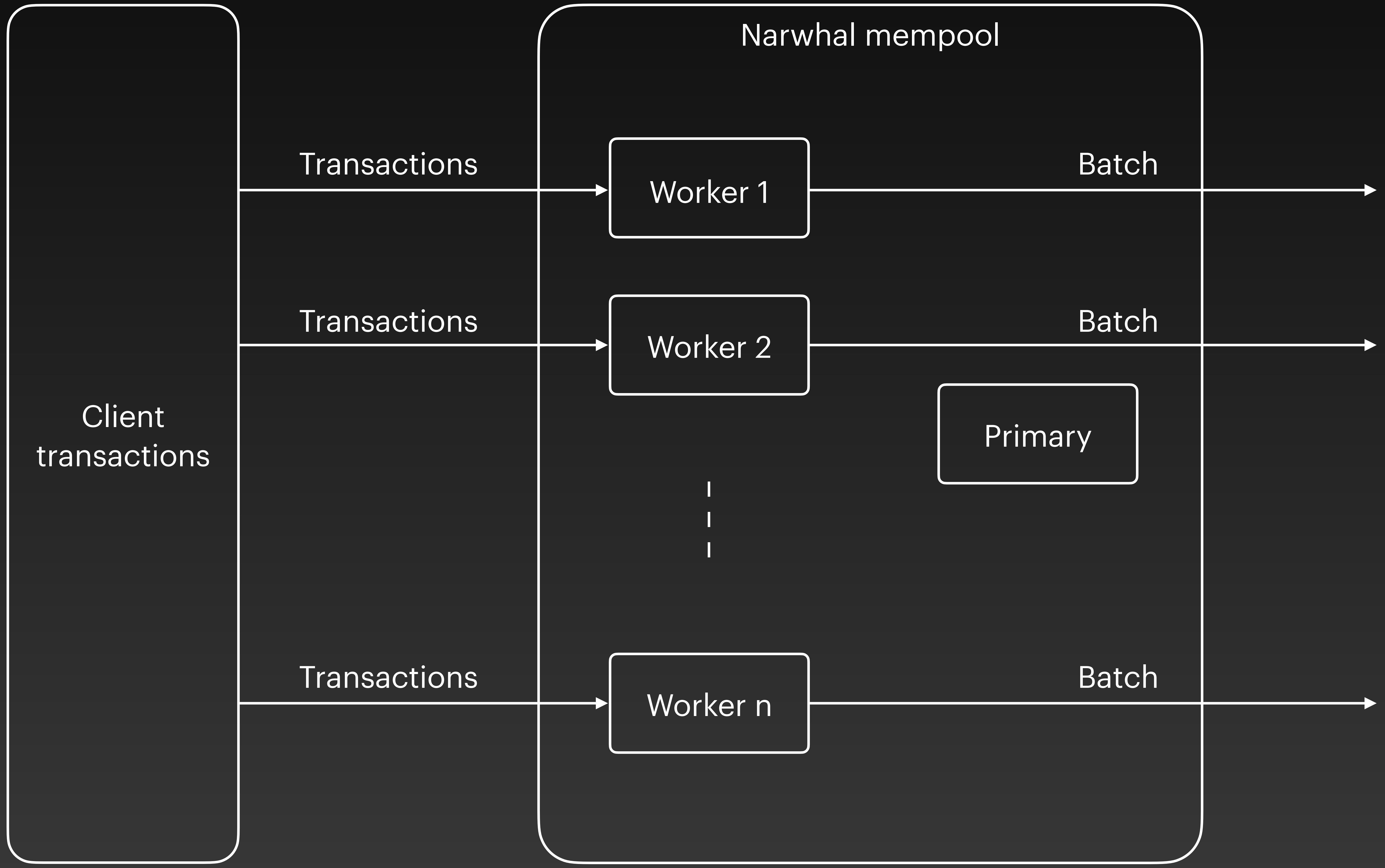
Narwhal

The workers and the primary



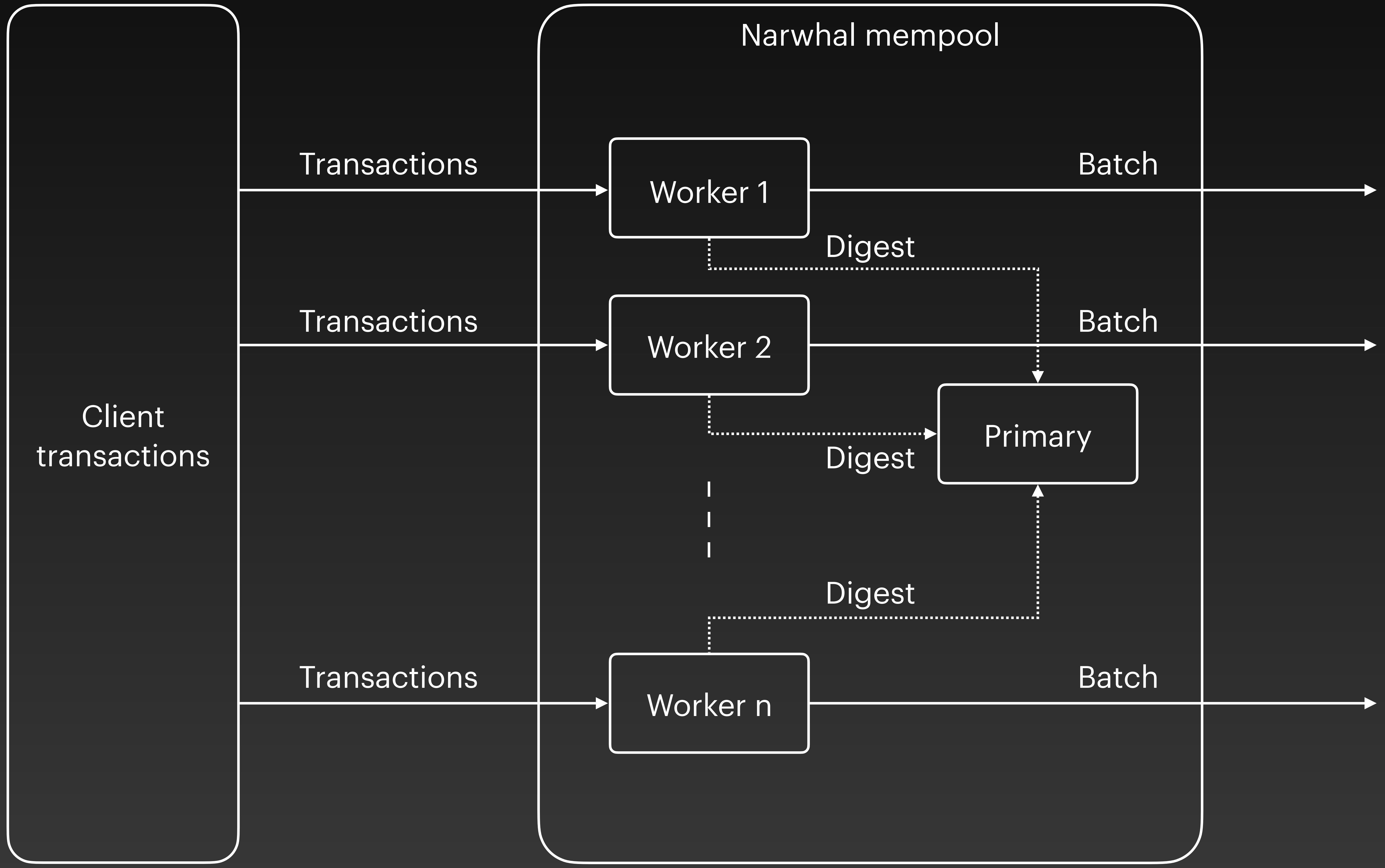
Narwhal

The workers and the primary



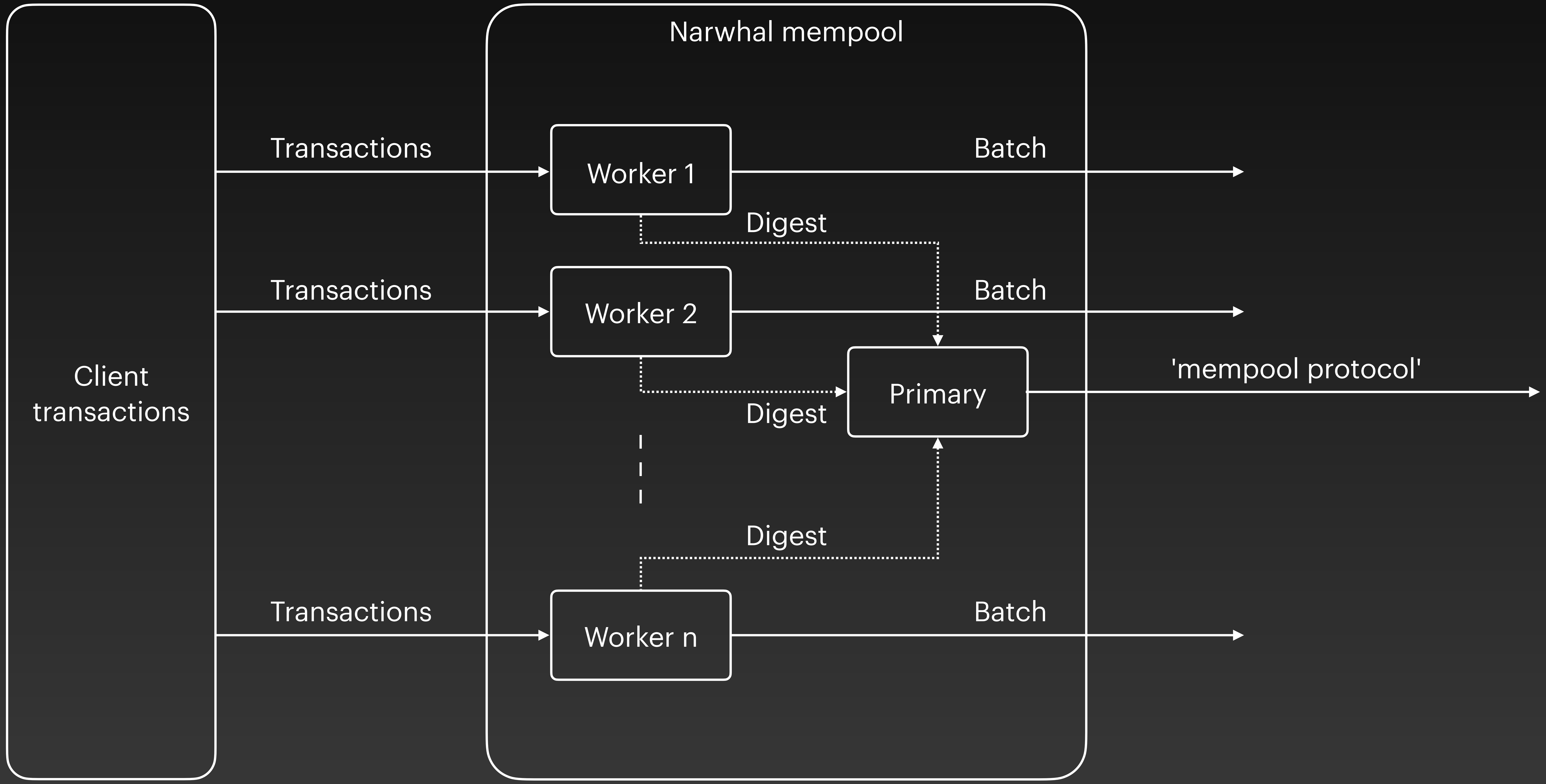
Narwhal

The workers and the primary



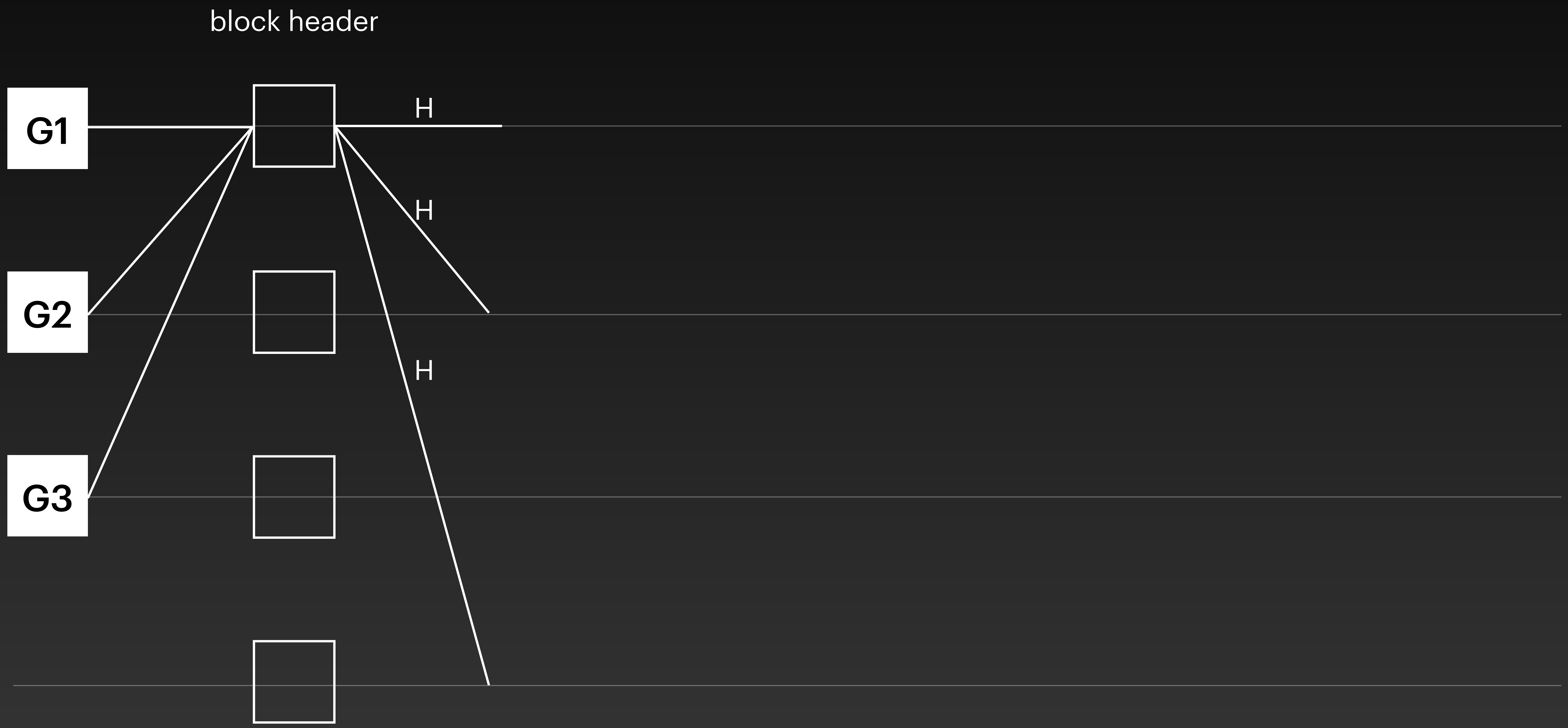
Narwhal

The workers and the primary



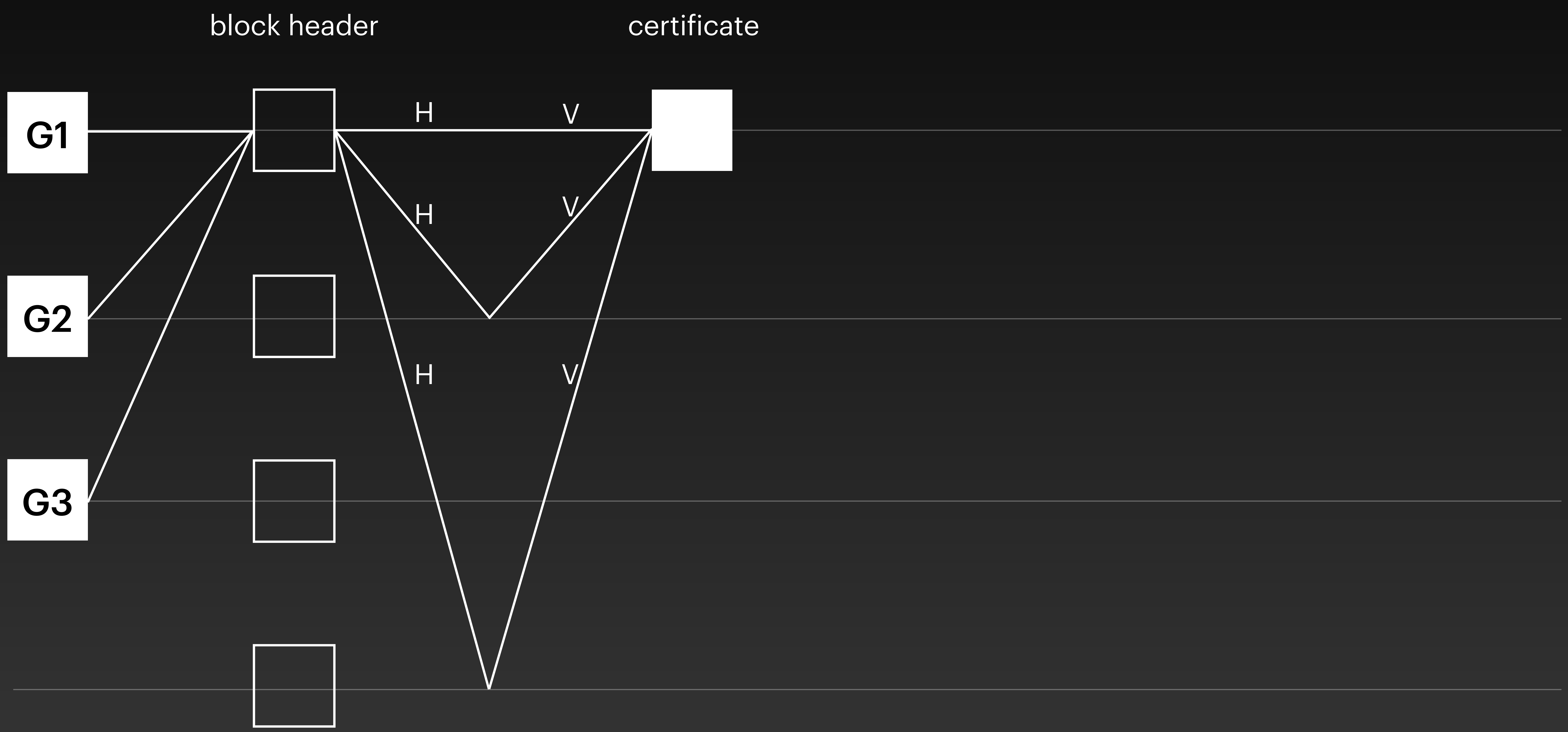
Narwhal

The primary machine



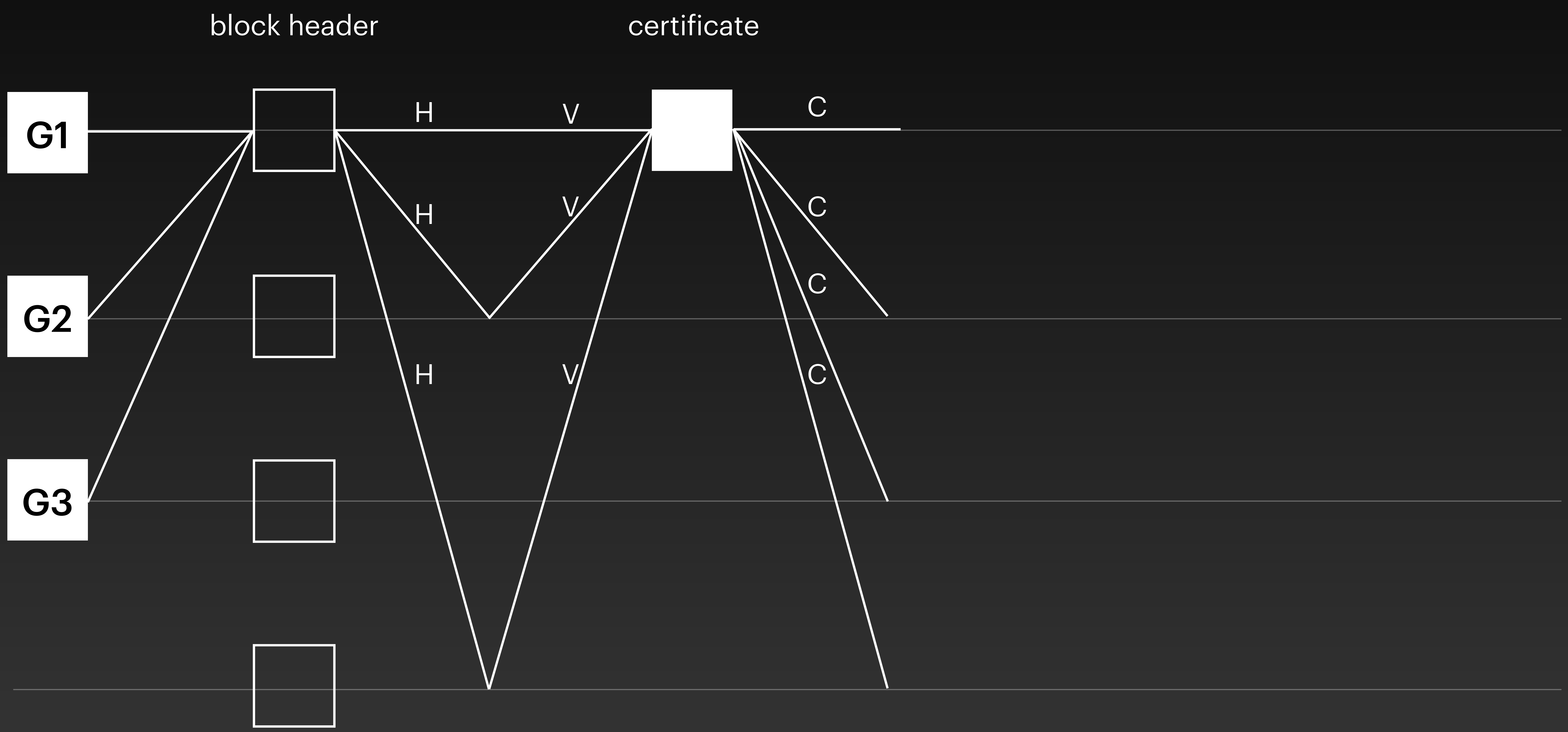
Narwhal

The primary machine



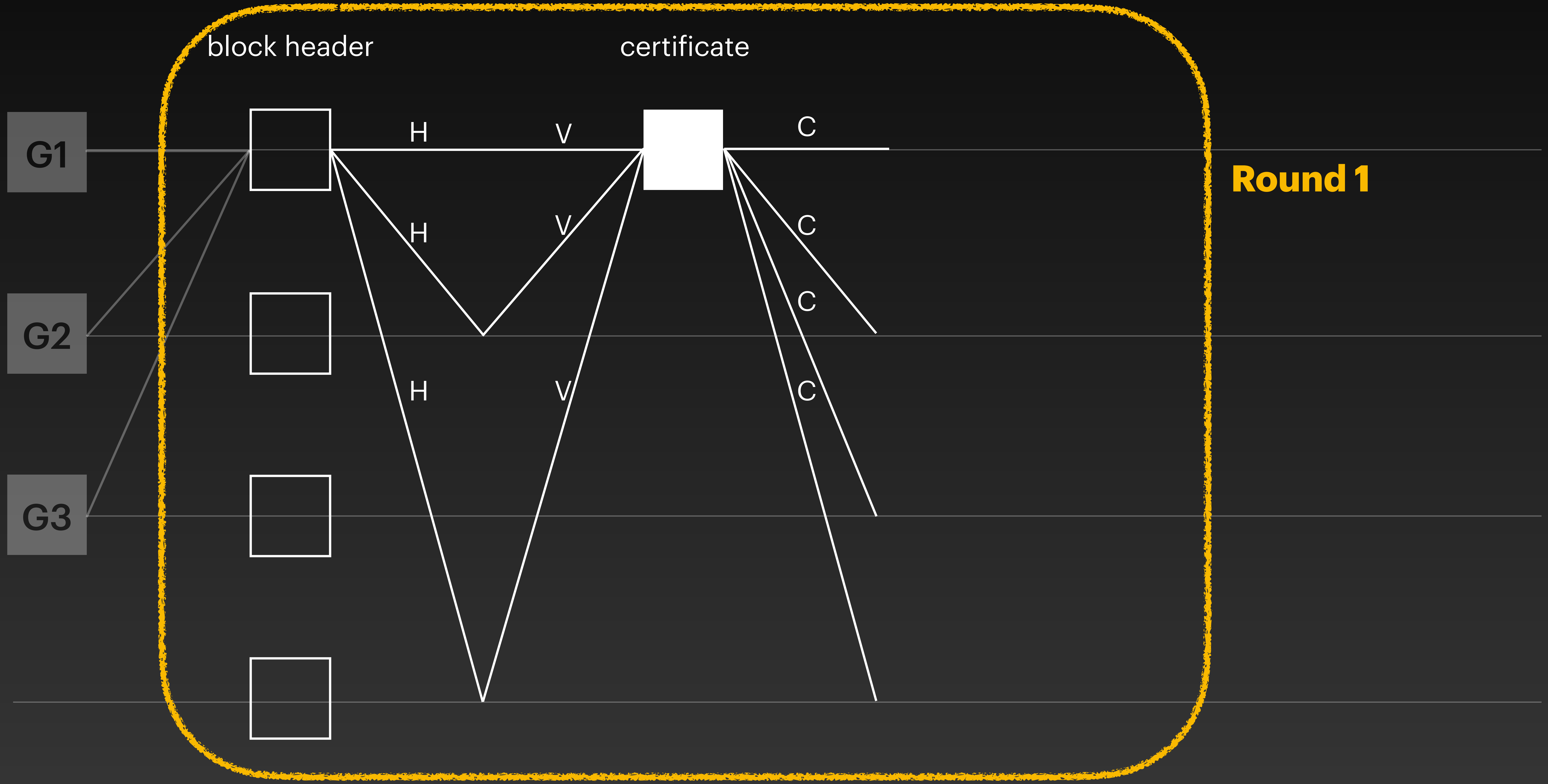
Narwhal

The primary machine



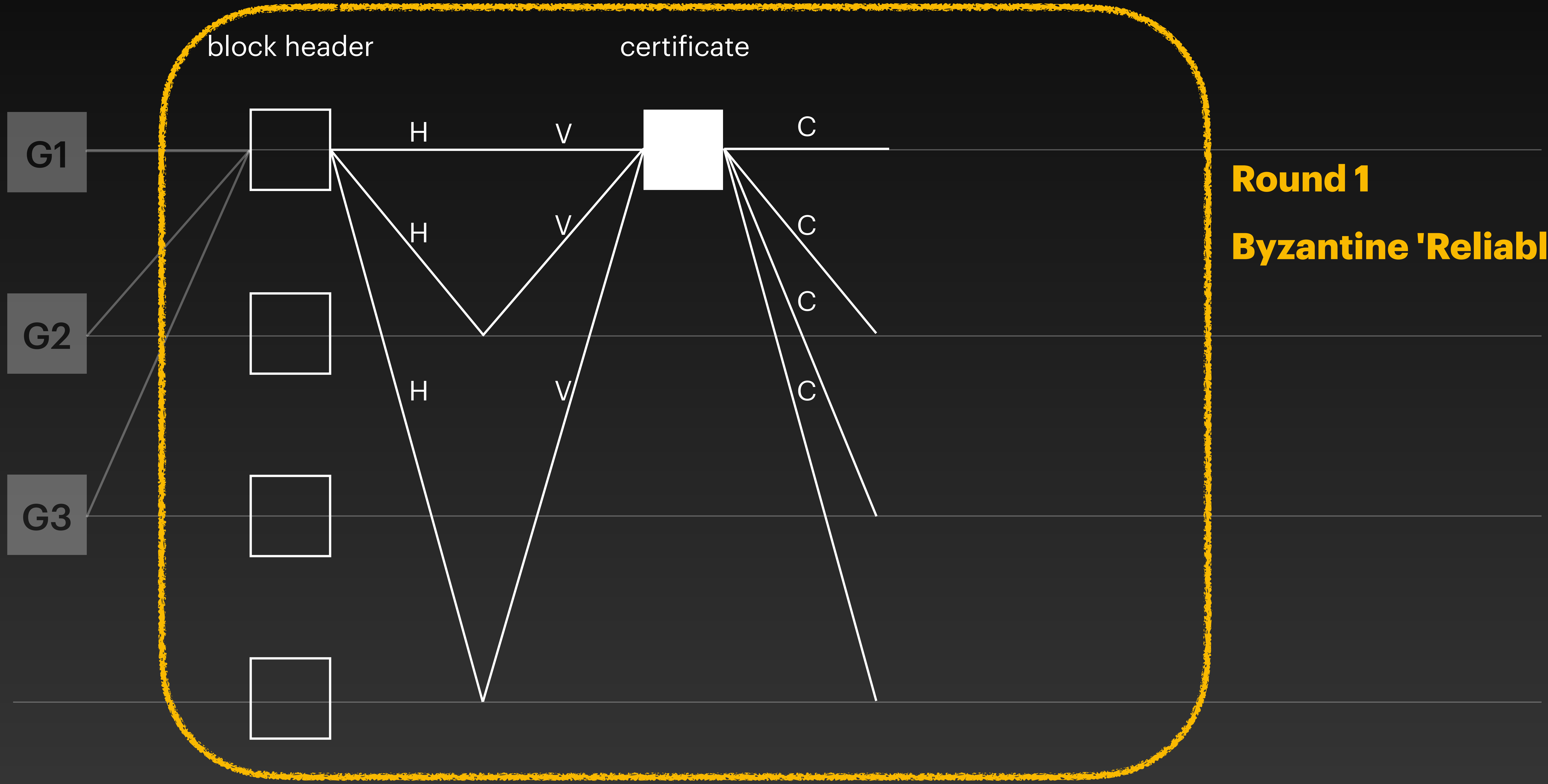
Narwhal

The primary machine



Narwhal

The primary machine

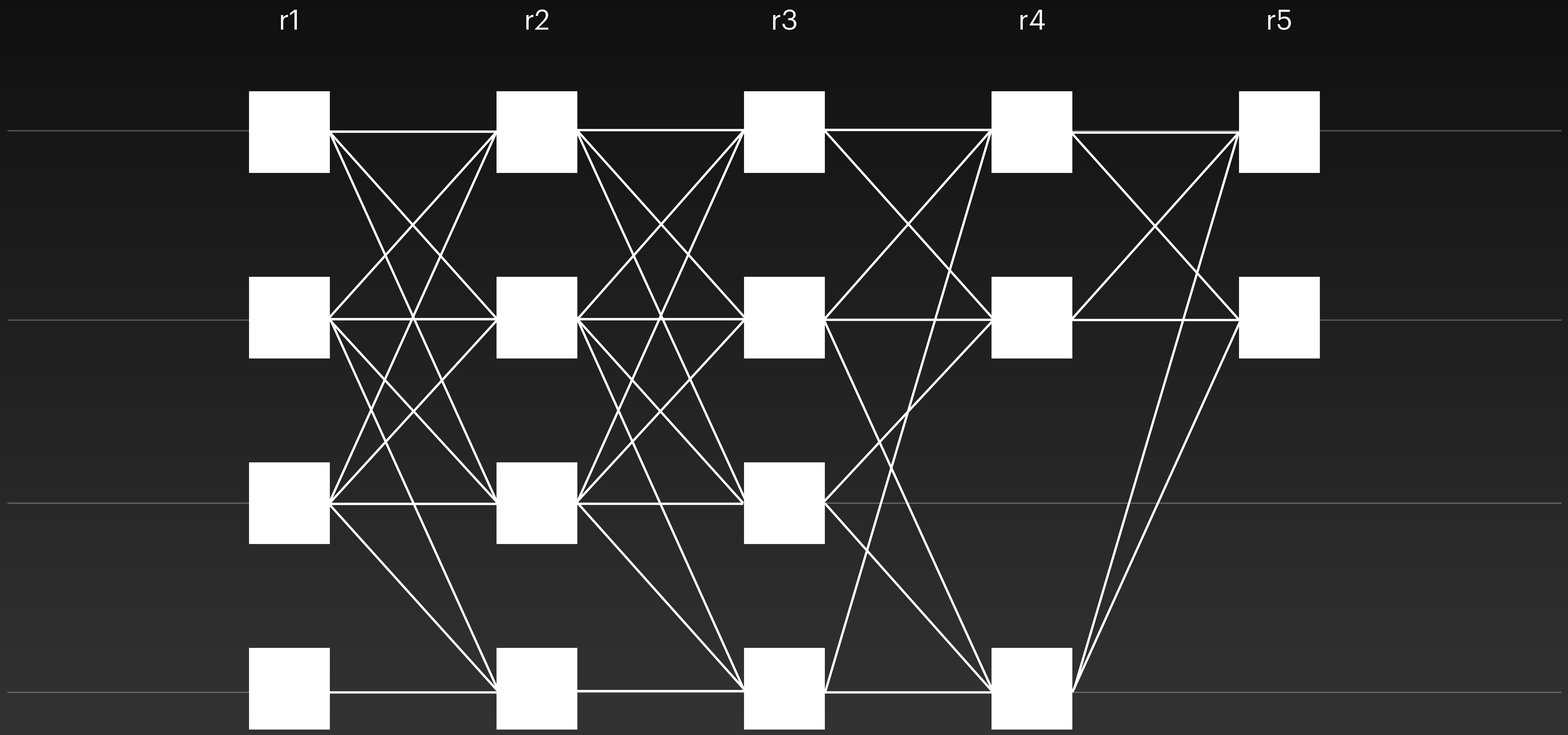


Round 1

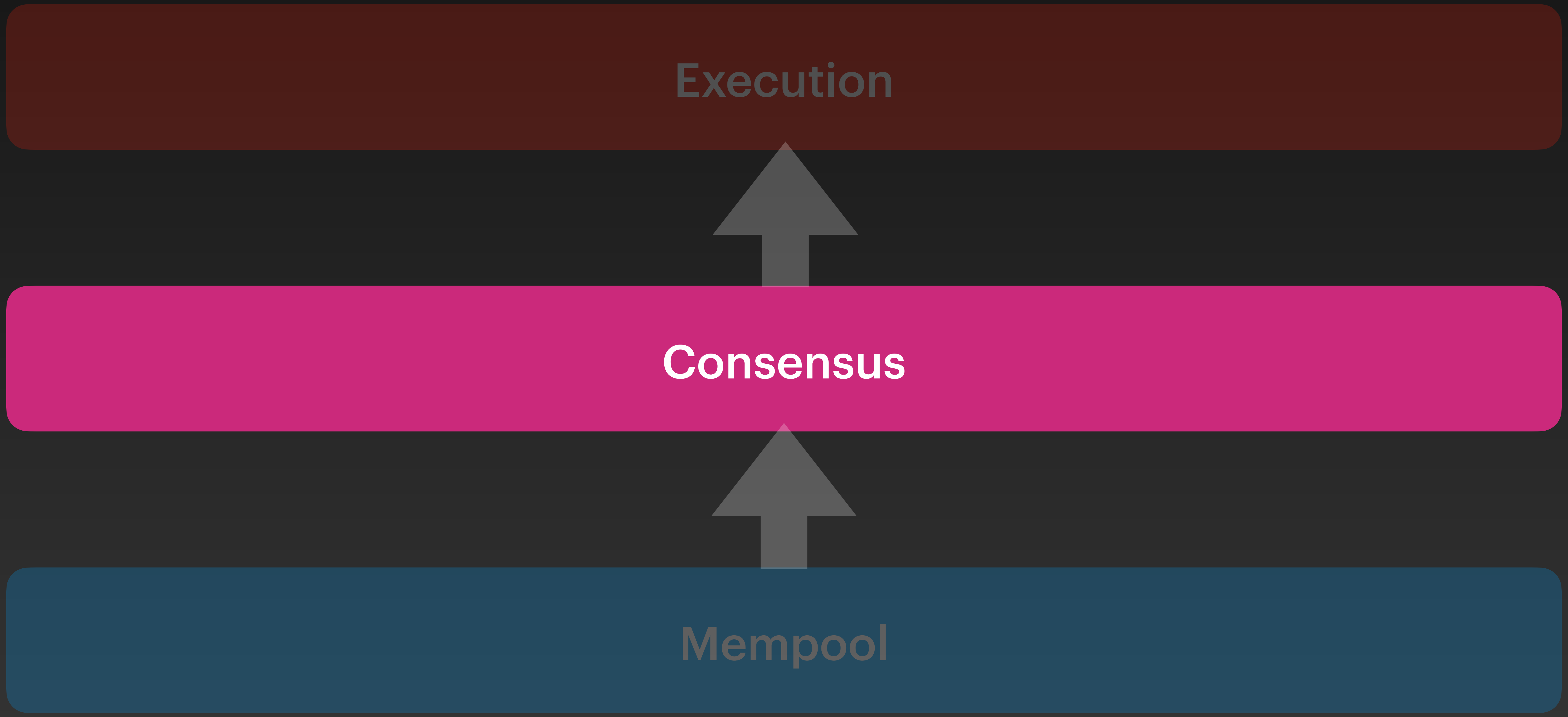
Byzantine 'Reliable' Broadcast

Narwhal

The primary machine



Second Step to the Solution



Execution

Consensus

Mempool

All You Need is DAG—PODC 21’

Narwhal and Tusk: A DAG-based Mempool and Efficient BFT Consensus— Eurosys 22’

Bullshark: Dag bft protocols made practical — CCS 22’

Hammerhead: Leader reputation for dynamic scheduling — ICDCS 24’

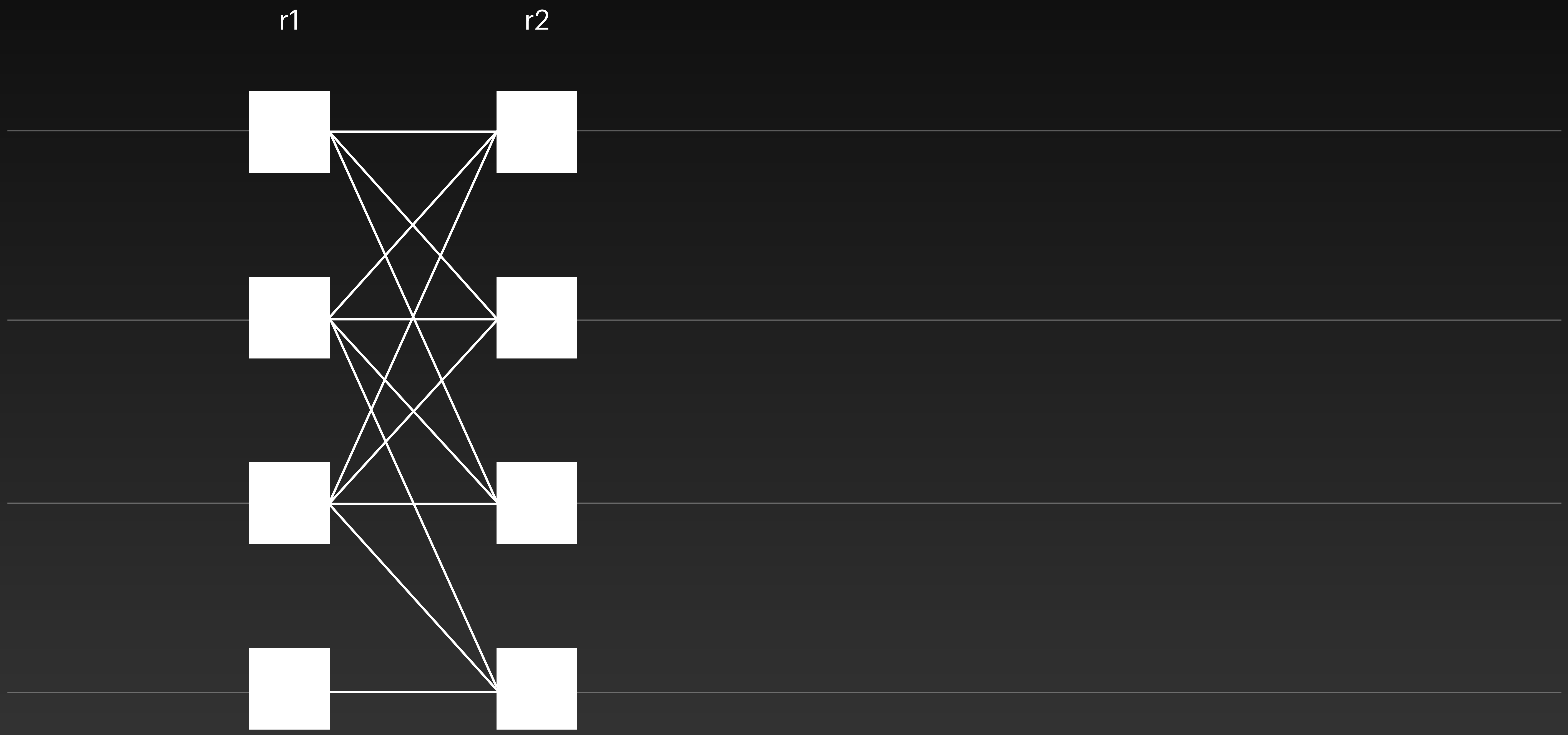
Bullshark

Zero-message partially-synchronous consensus

* without asynchronous fallback

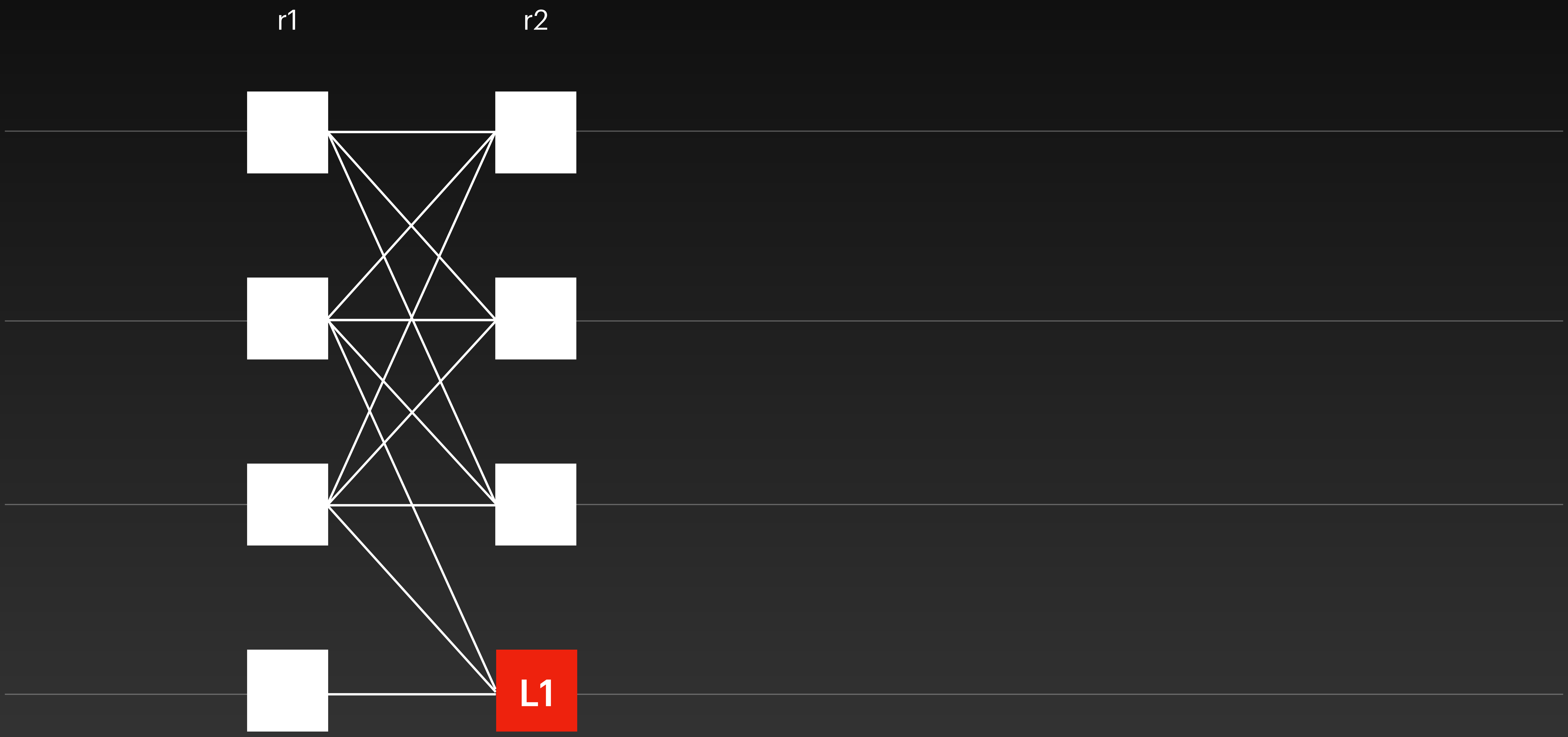
Bullshark

Just interpret the DAG



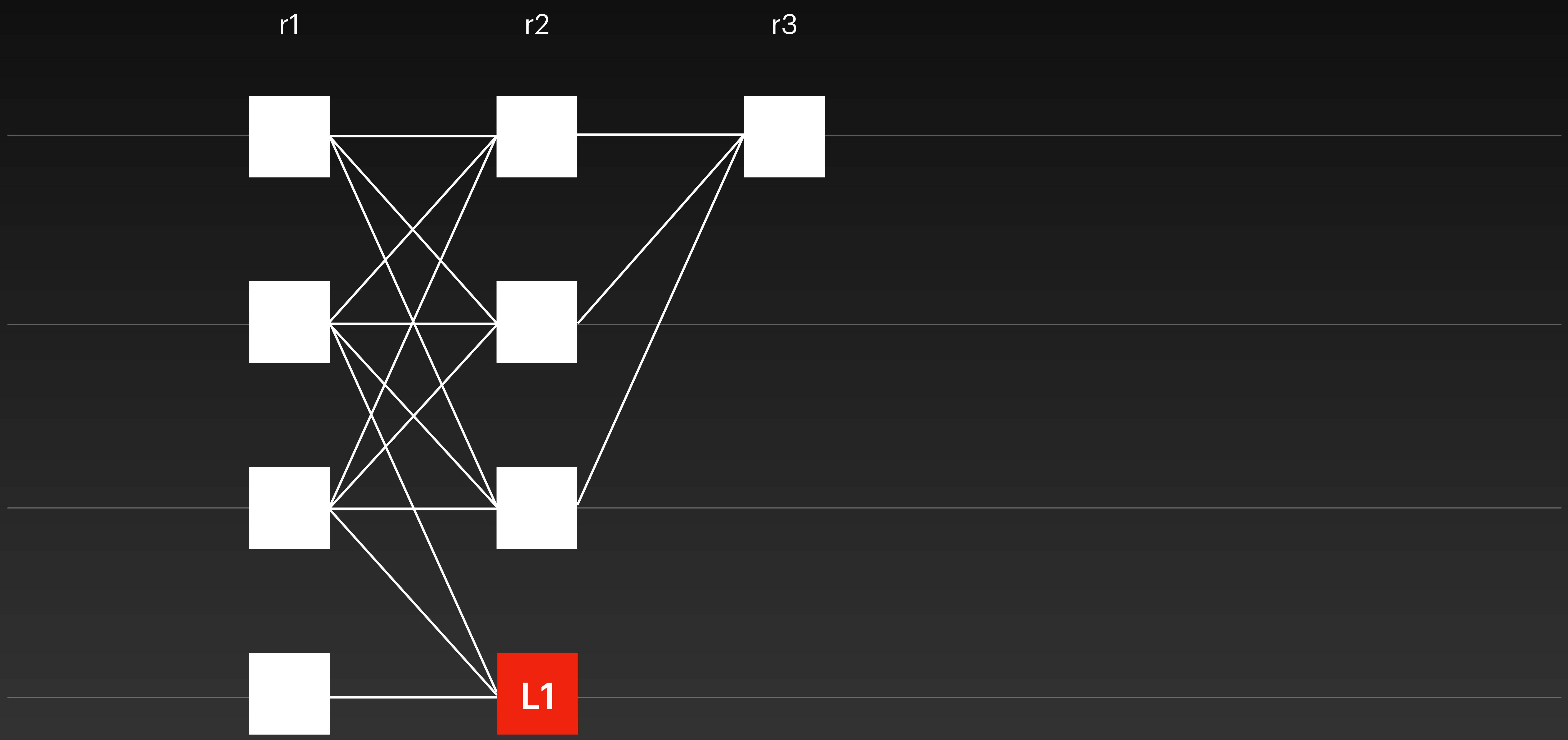
Bullshark

Deterministic leader every 2 rounds



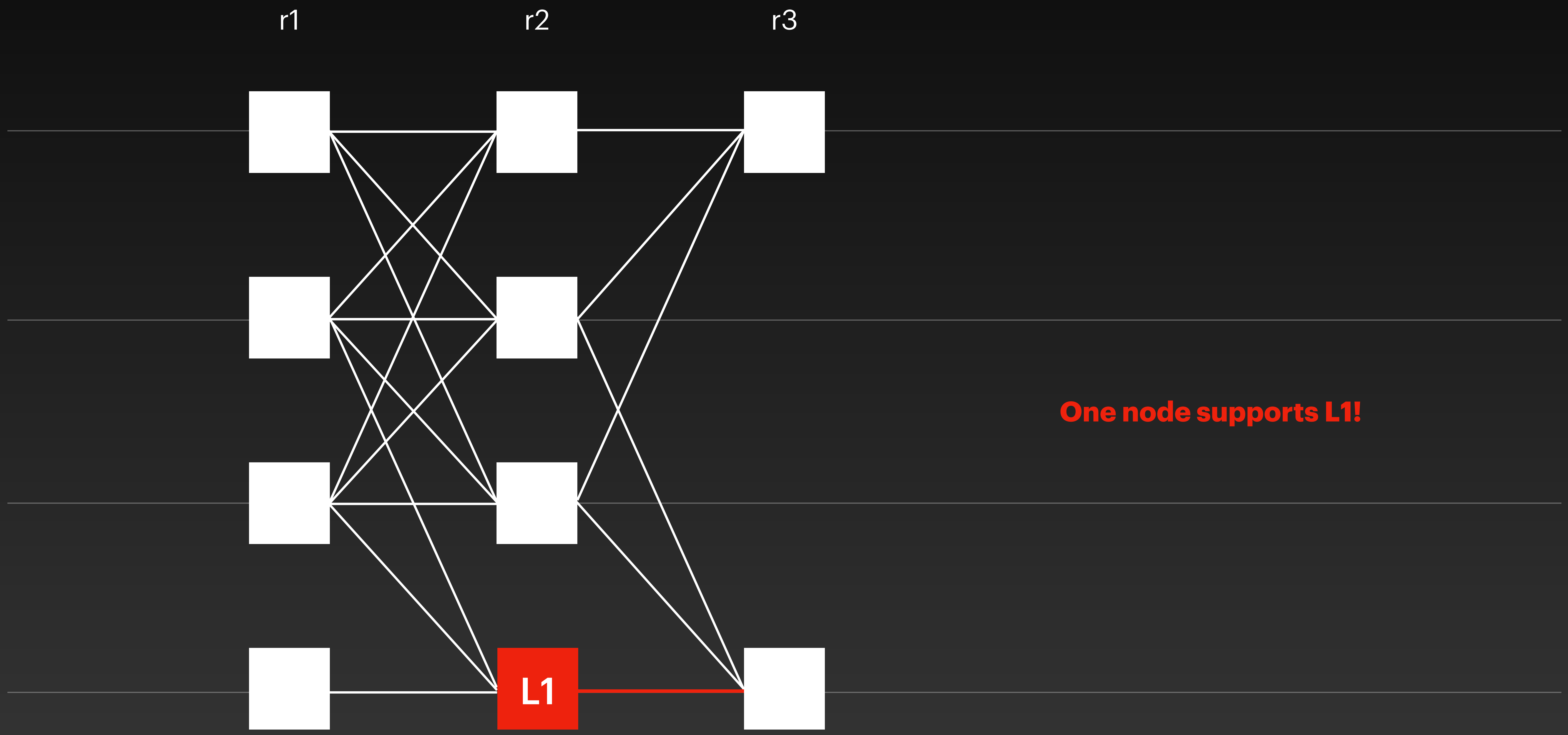
Bullshark

The leader needs f+1 links from round r



Bullshark

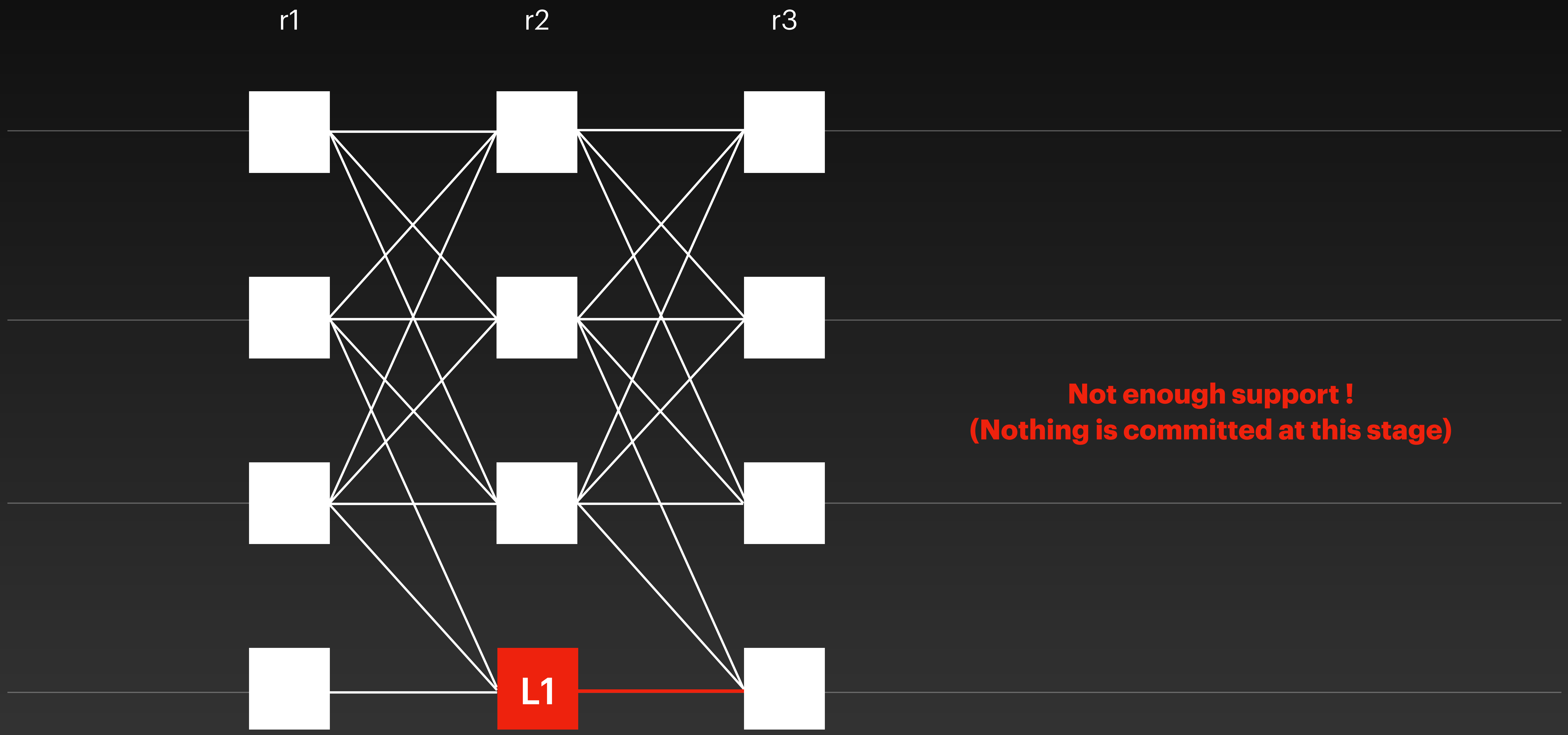
The leader needs f+1 links from round r



One node supports L1!

Bullshark

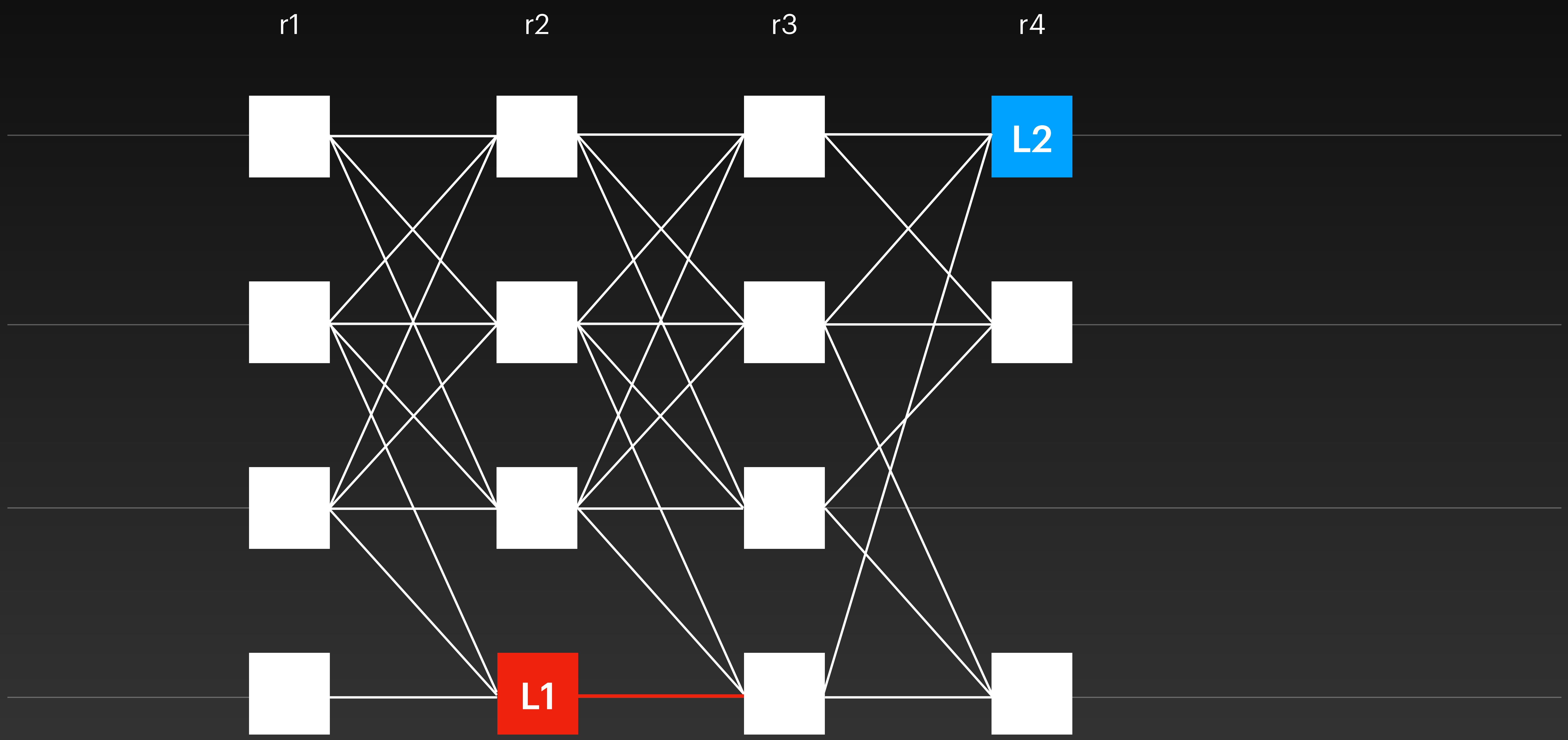
The leader needs f+1 links from round r



Not enough support !
(Nothing is committed at this stage)

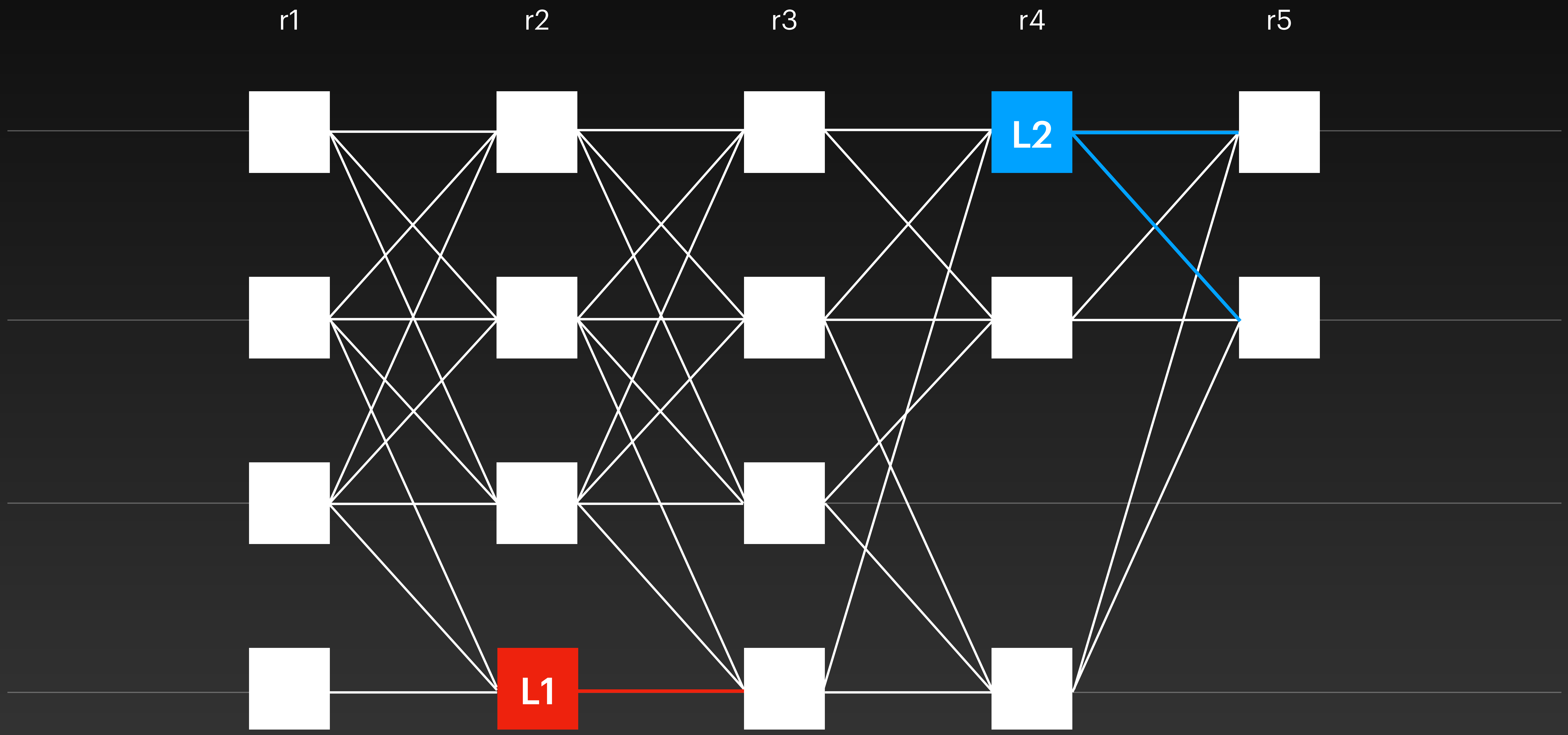
Bullshark

Elect the leader of r4



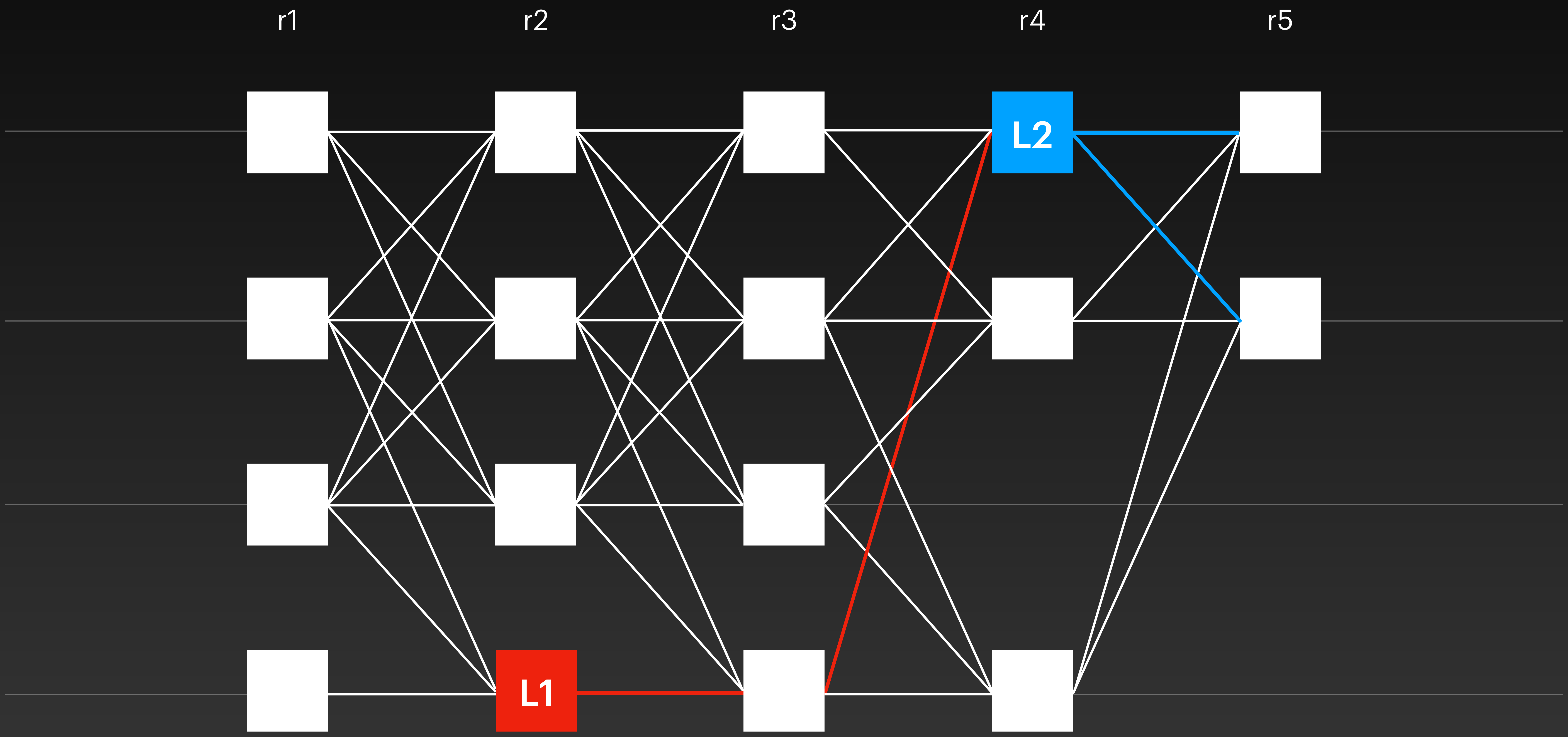
Bullshark

Leader L2 has enough support



Bullshark

Leader L2 has links to leader L1

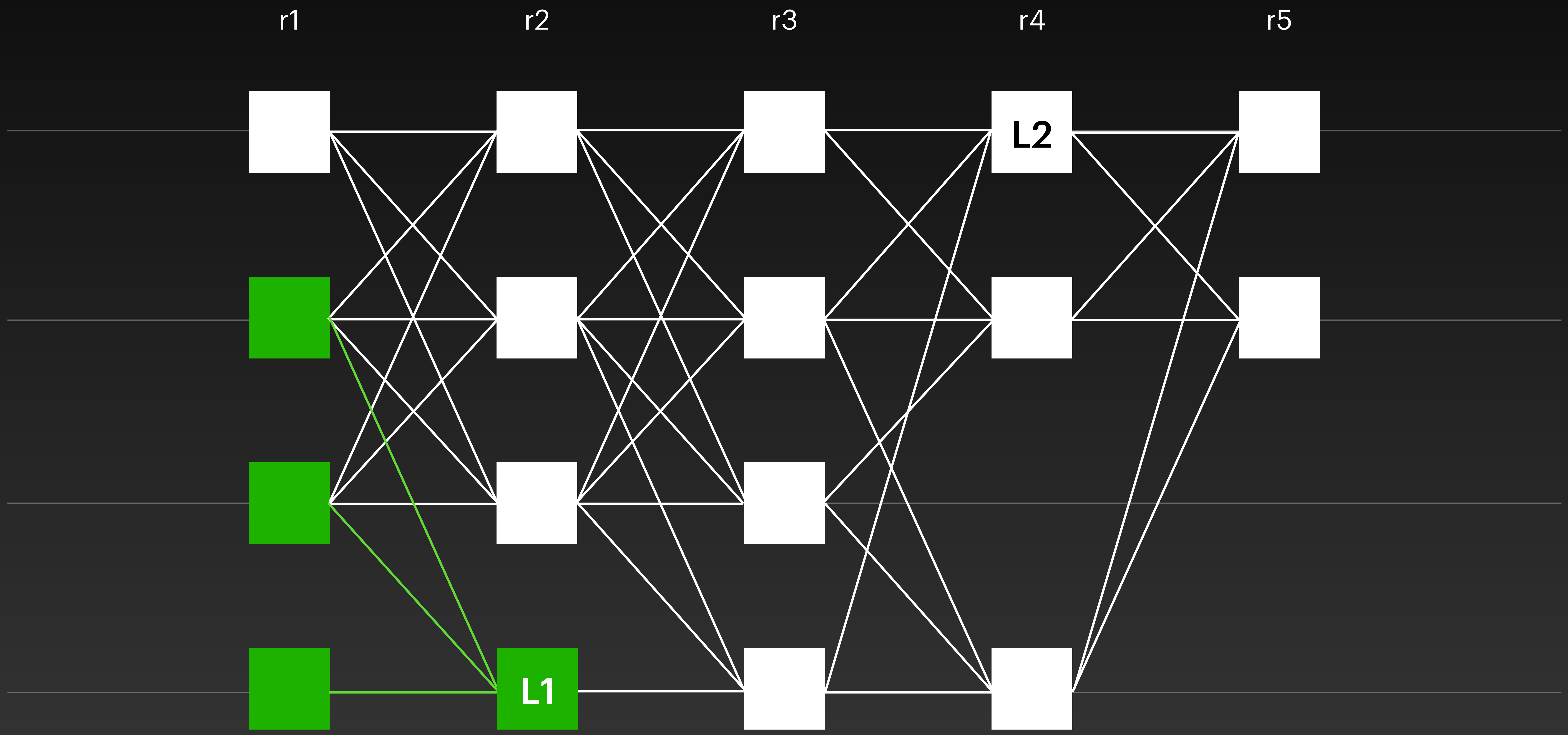


First commit L1

Then commit L2

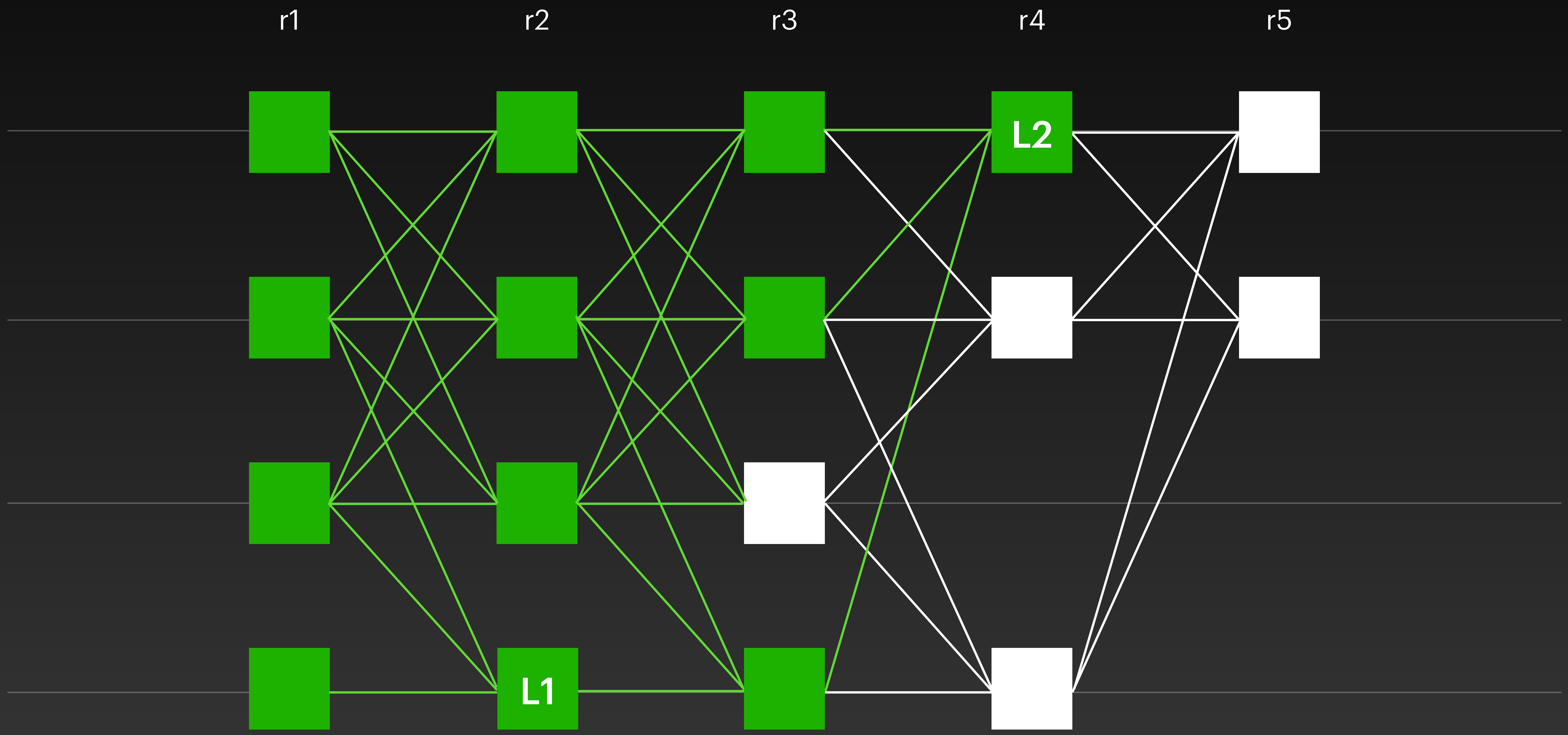
Bullshark

Commit all the sub-DAG of the leader



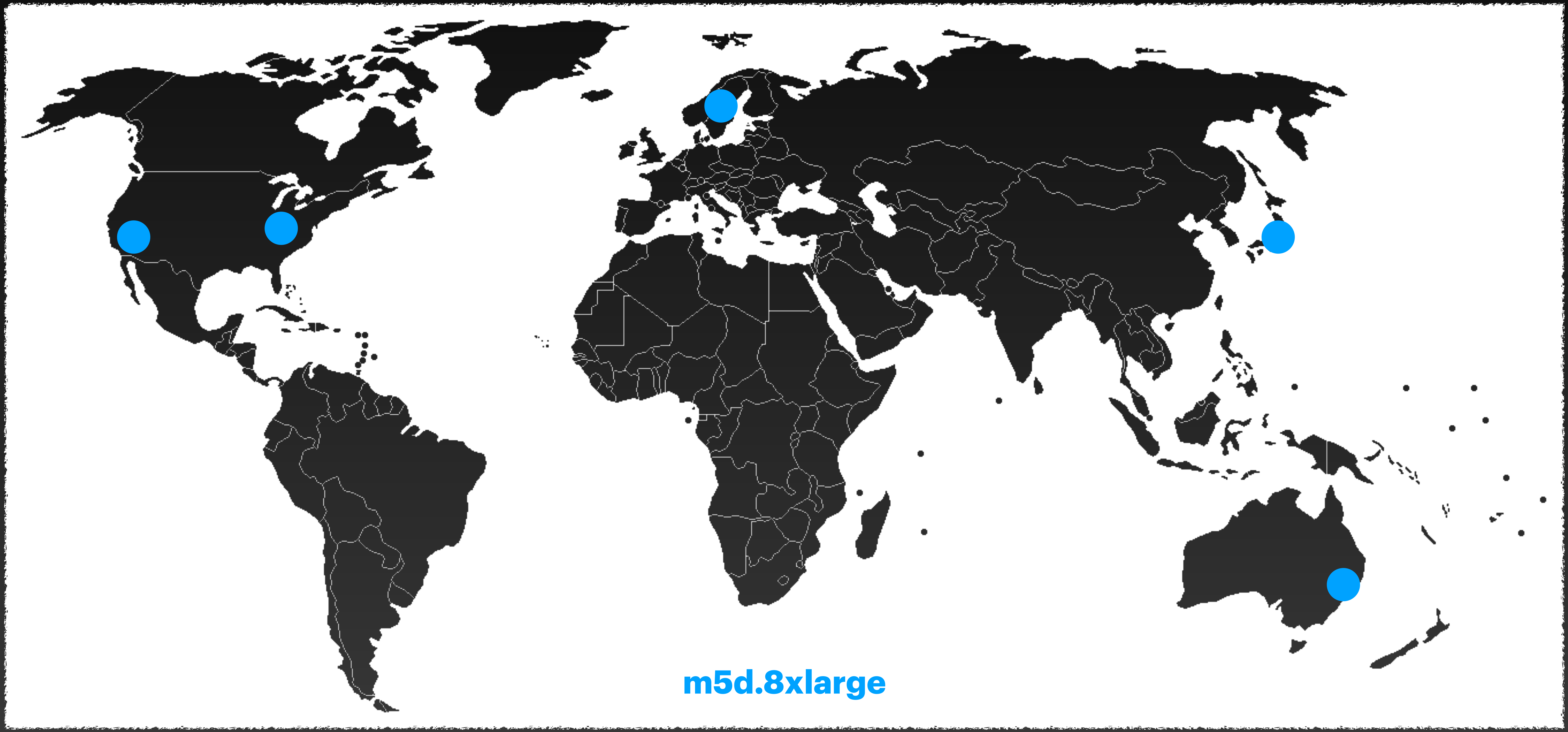
Bullshark

Commit all the sub-DAG of the leader



Evaluation

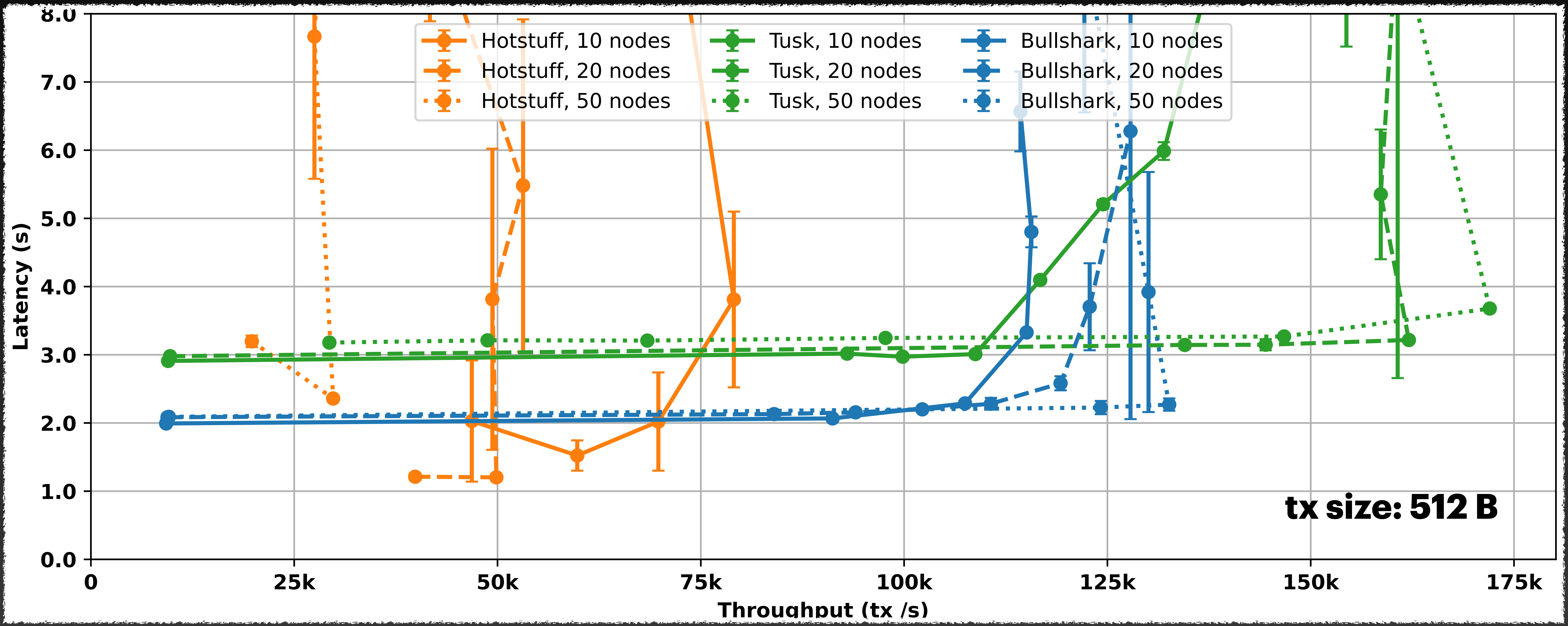
Experimental setup on AWS



m5d.8xlarge

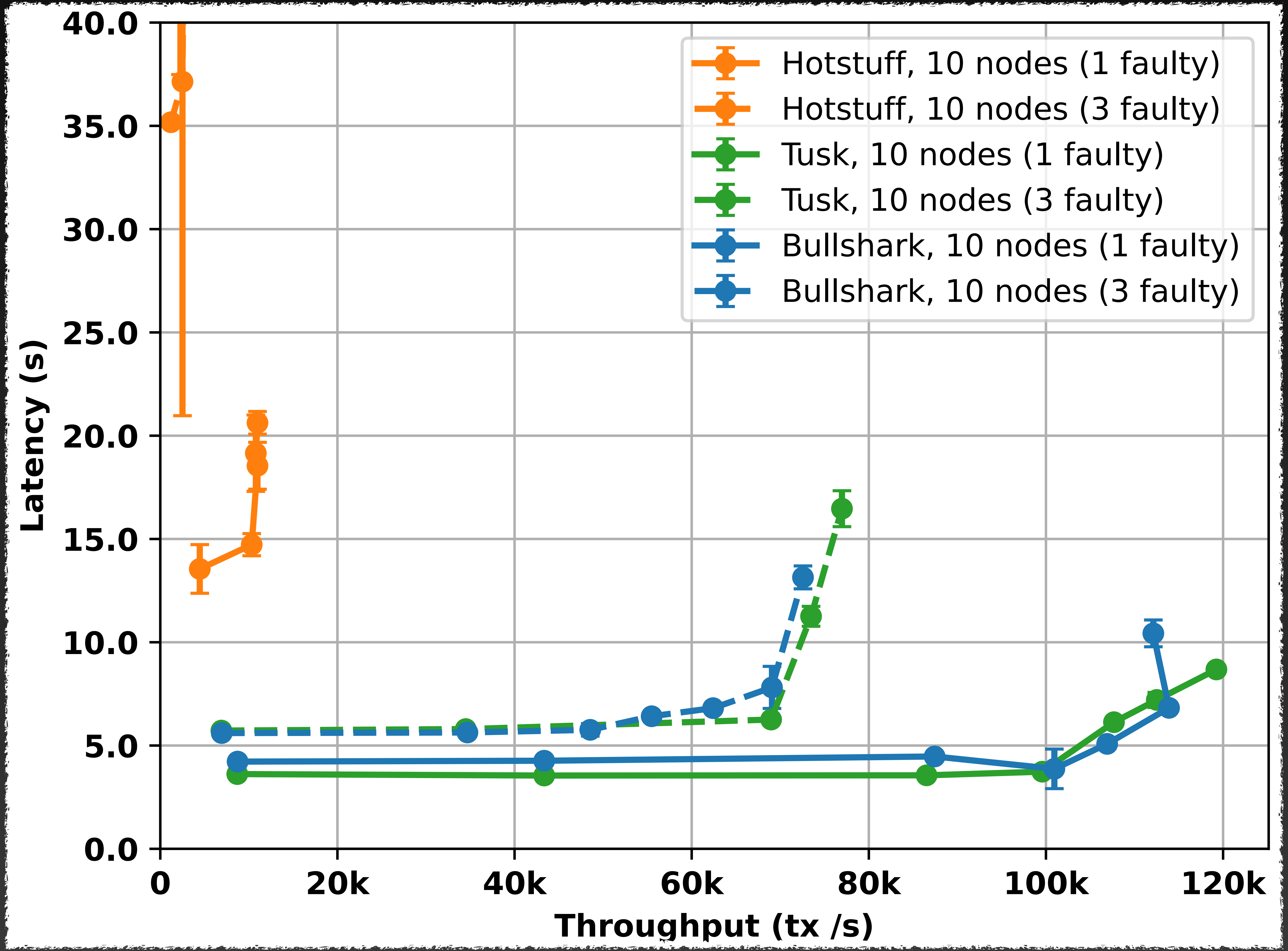
Evaluation

Throughput latency graph



Evaluation

Performance under faults



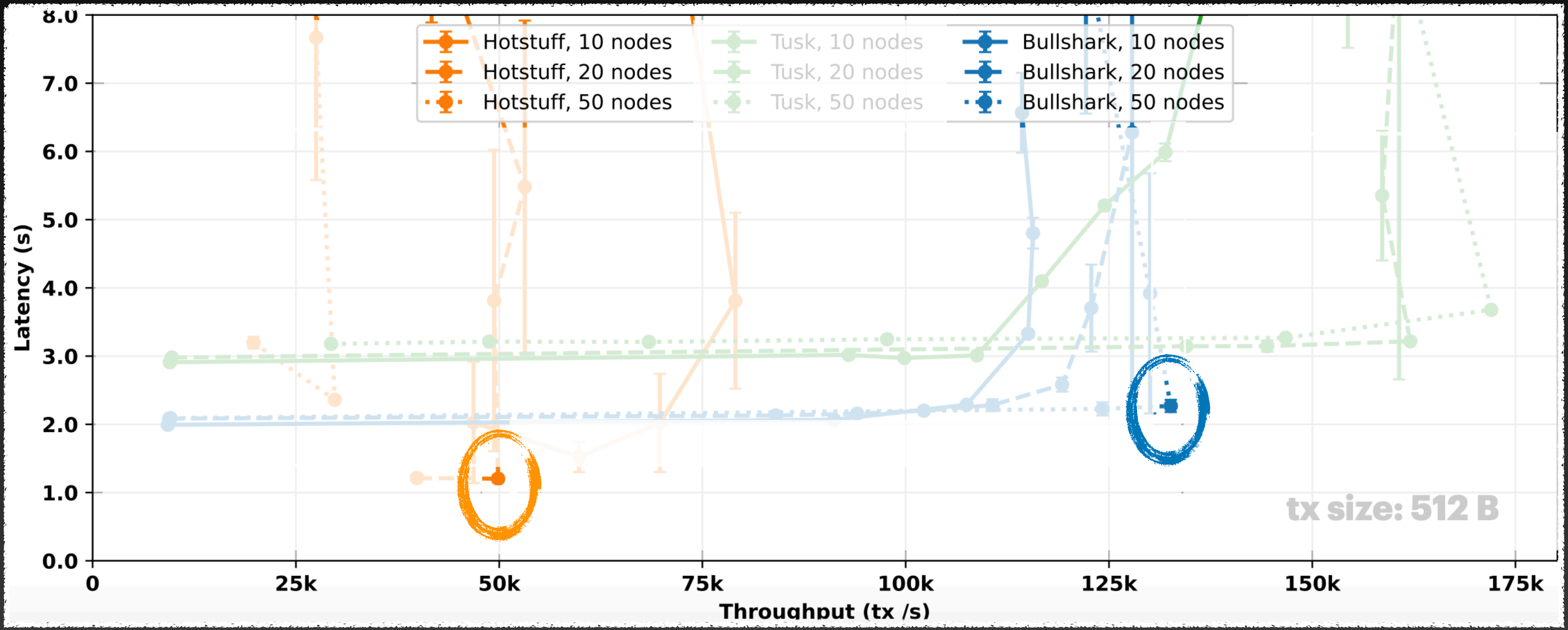
Summary

Bullshark

- Zero-message overhead, no view-change, no common-coin
- Disseminate data with Narwhal, exploits periods of synchrony

Are we done?

Latency?

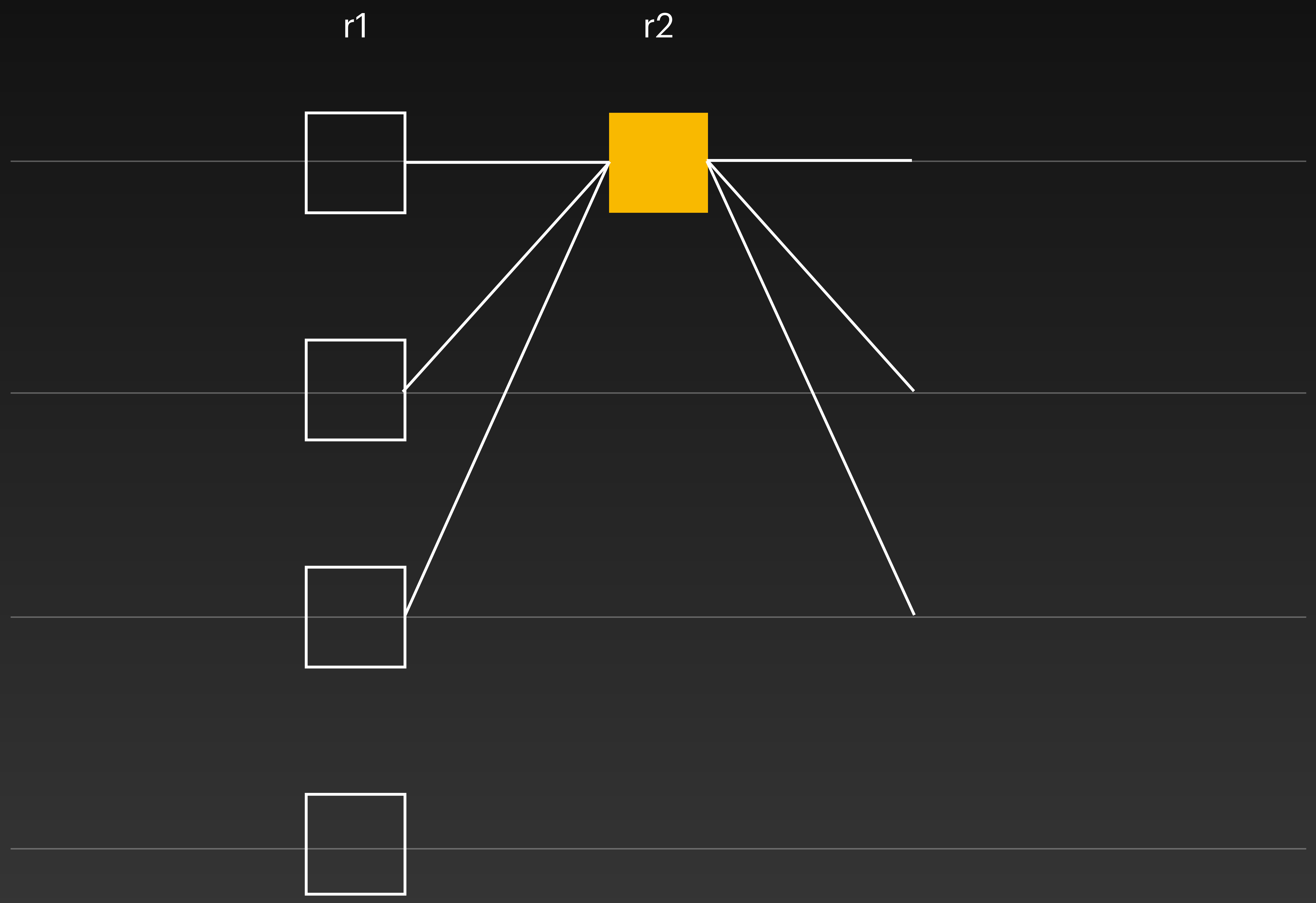


The Mysticeti DAG

Uncertified DAG

The Mysticeti DAG

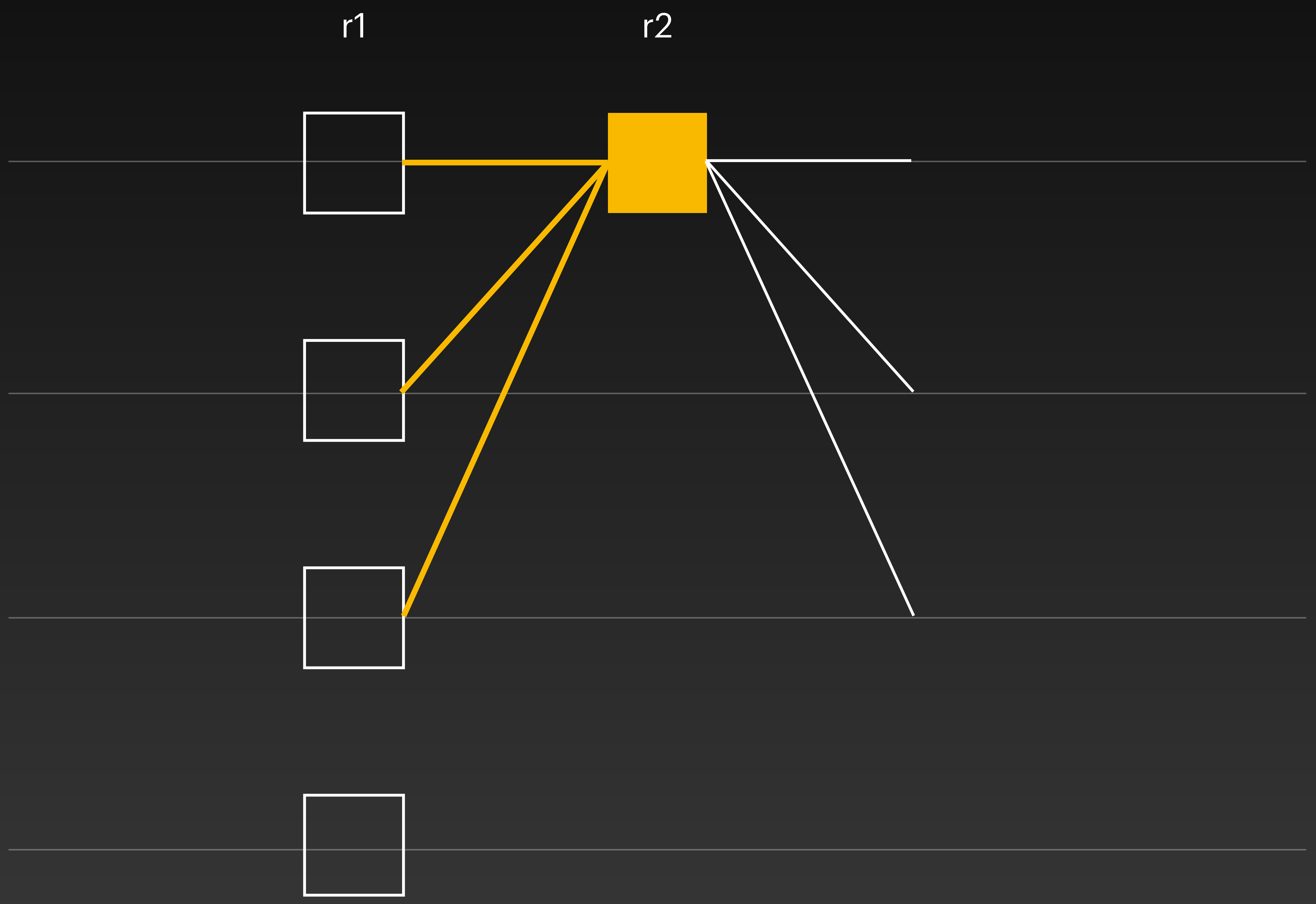
Block Creation



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

The Mysticeti DAG

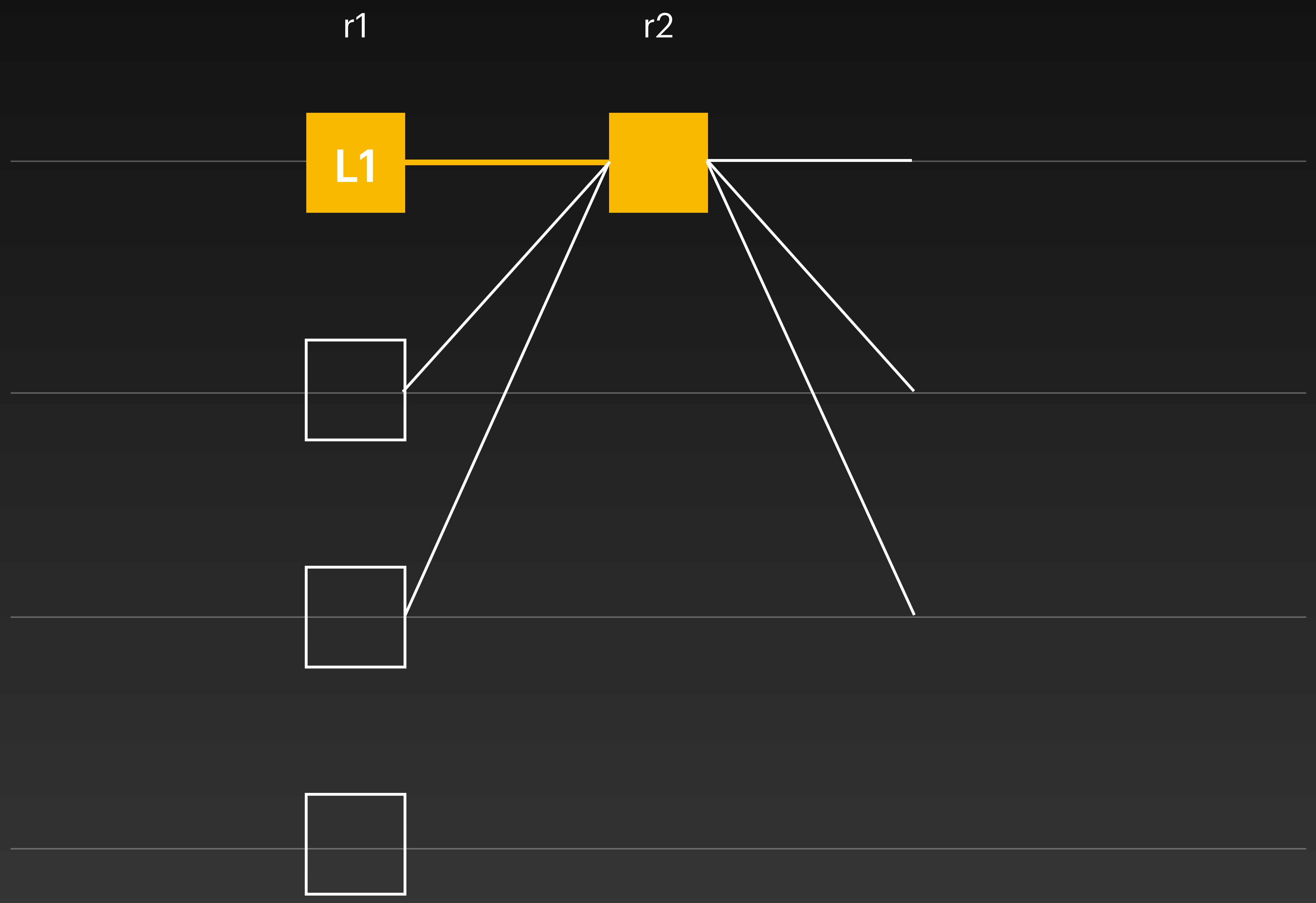
Rule 1: Link to 2f+1 parents



- Total nodes: **3f+1 = 4**
- Quorum: **2f+1 = 3**

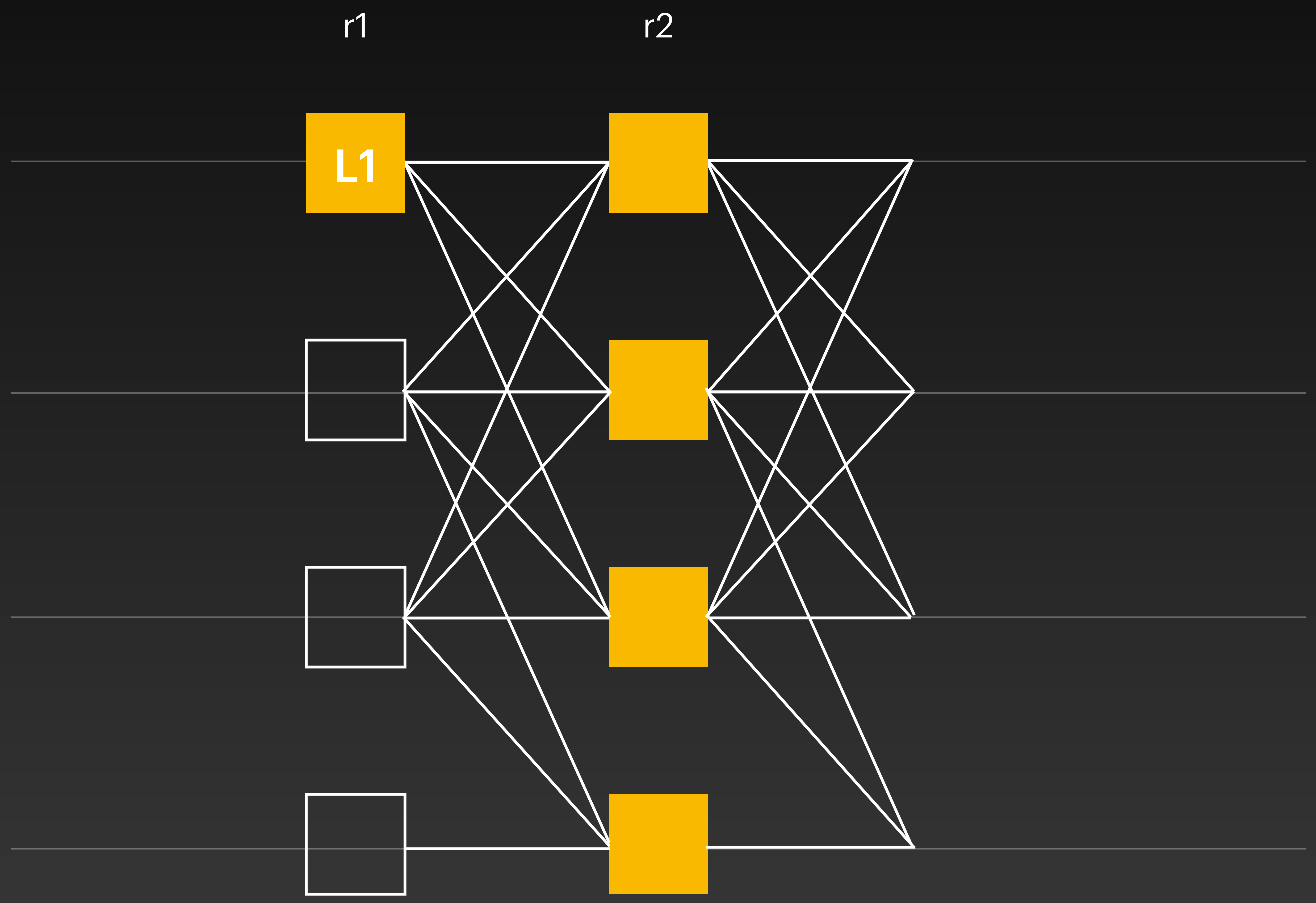
The Mysticeti DAG

Rule 2: Every node waits and links to leaders

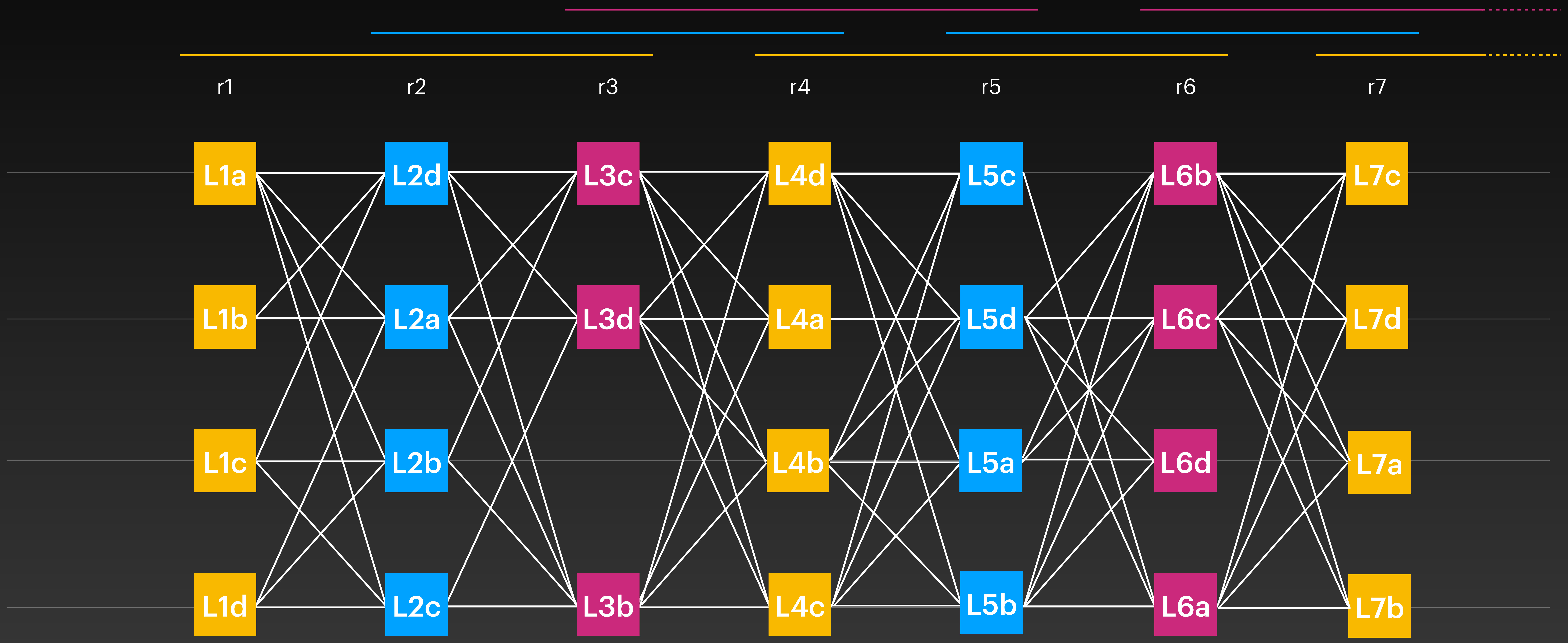


The Mysticeti DAG

Rule 3: All node run in parallel

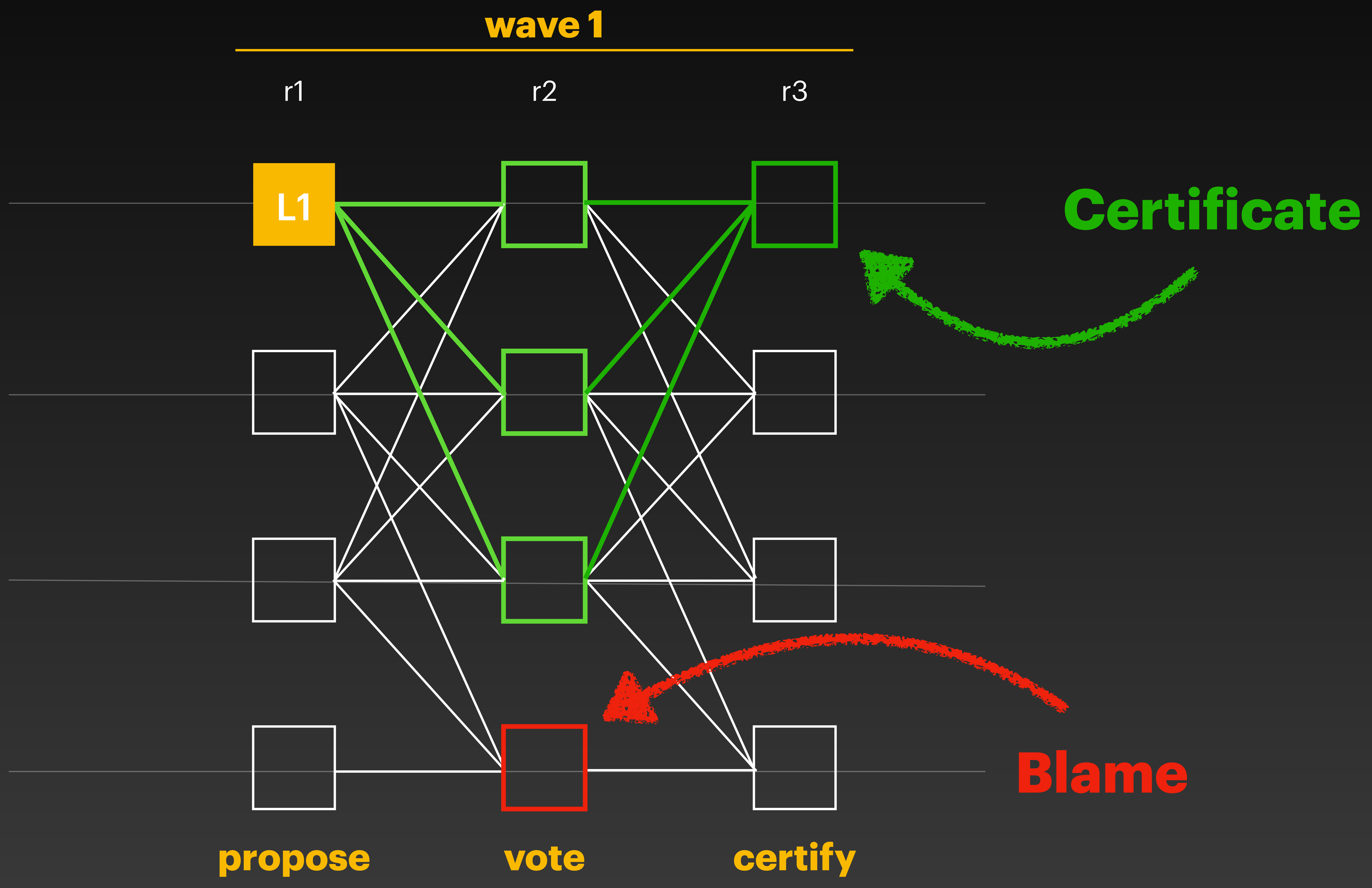


DAG Structure



Interpreting DAG Patterns

Reminder



Direct Decision Rule

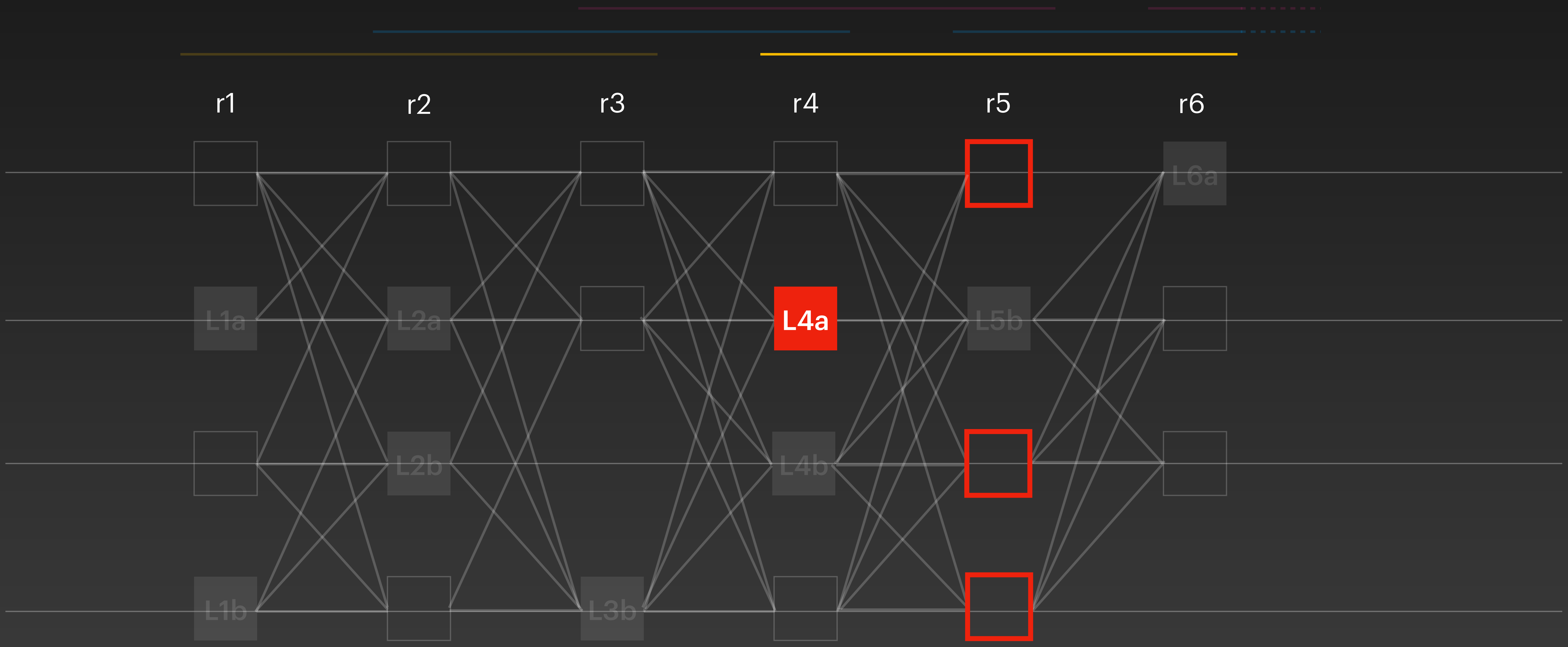
On each leader starting from highest round:

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

Direct Decision Rule

On each leader starting from highest round:

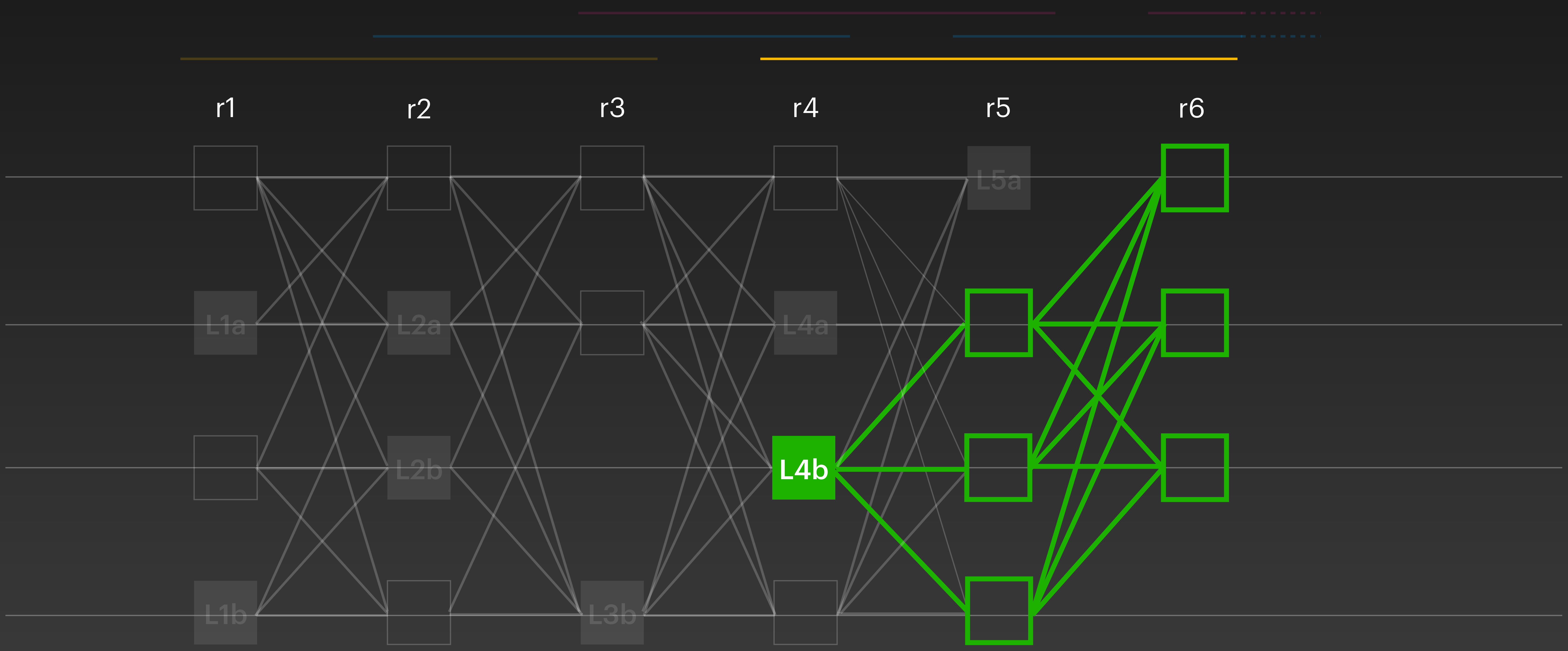
- **Skip** if 2f+1 blames
- **Commit** if 2f+1 certificates
- **Undecided** otherwise



Direct Decision Rule

On each leader starting from highest round:

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



Why?

Crash Faults

In a year of running Sui:

- How many Byzantine faults? 0

Why?

Crash Faults

In a year of running Sui:

- How many Byzantine faults? 0
- How many Crash faults?

Why?

Crash Faults

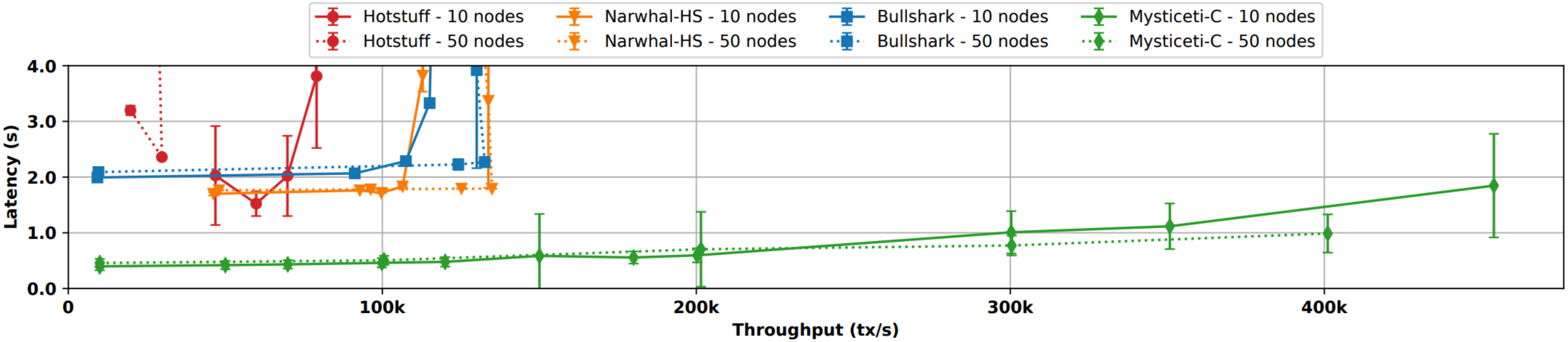
In a year of running Sui:

- How many Byzantine faults? 0
- How many Crash faults? 😭

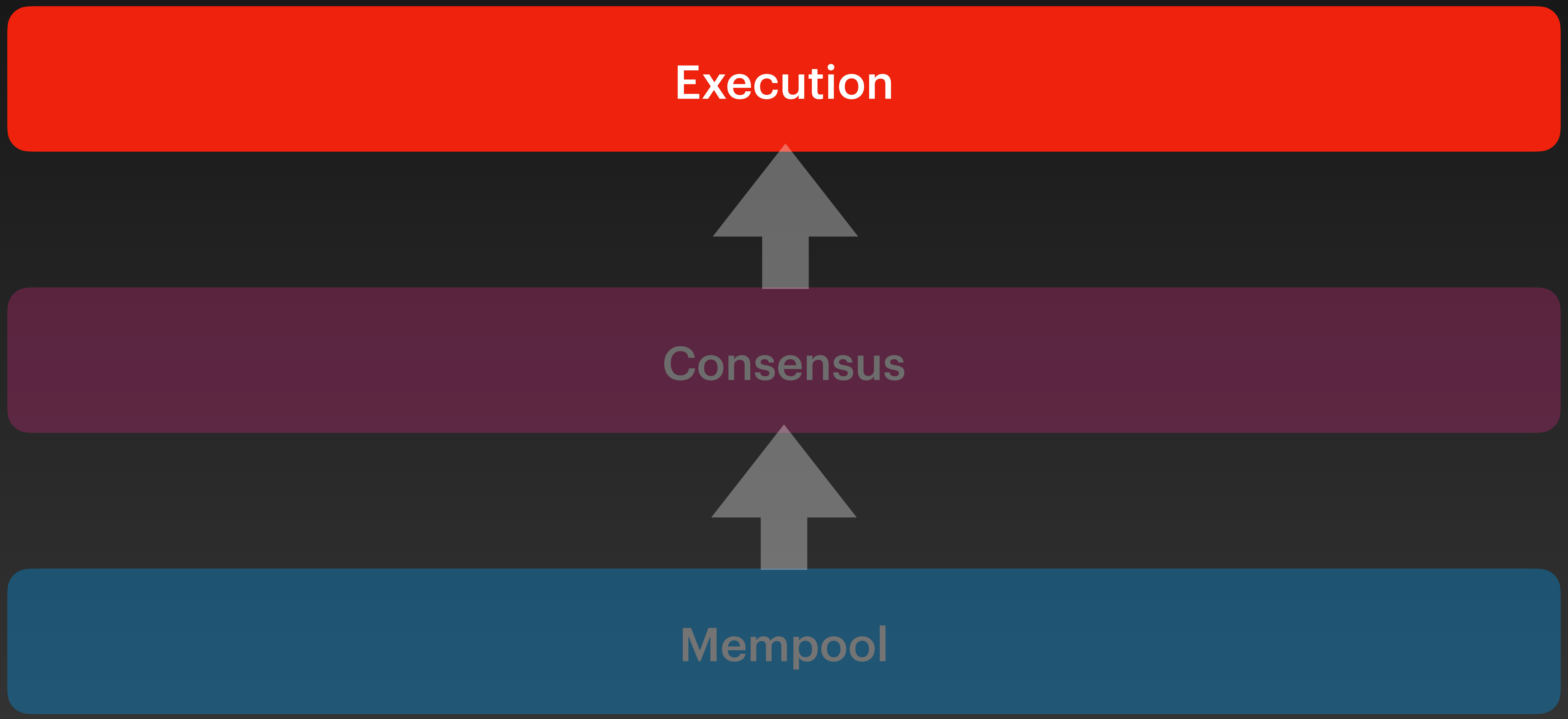
Resources

- **Paper:** https://arxiv.org/pdf/2310.14821
- **Presentation:** https://www.youtube.com/watch?v=JhhCxyZylx8

Evaluation


Hotstuff - 10 nodes
Hotstuff - 50 nodes
Narwhal-HS - 10 nodes
Narwhal-HS - 50 nodes
Bullshark - 10 nodes
Bullshark - 50 nodes
Mysticeti-C - 10 nodes
Mysticeti-C - 50 nodes
Latency (s)
4.0
3.0
2.0
1.0
0.0
0
100k
200k
300k
400k
Throughput (tx/s)

Last Step to the Solution



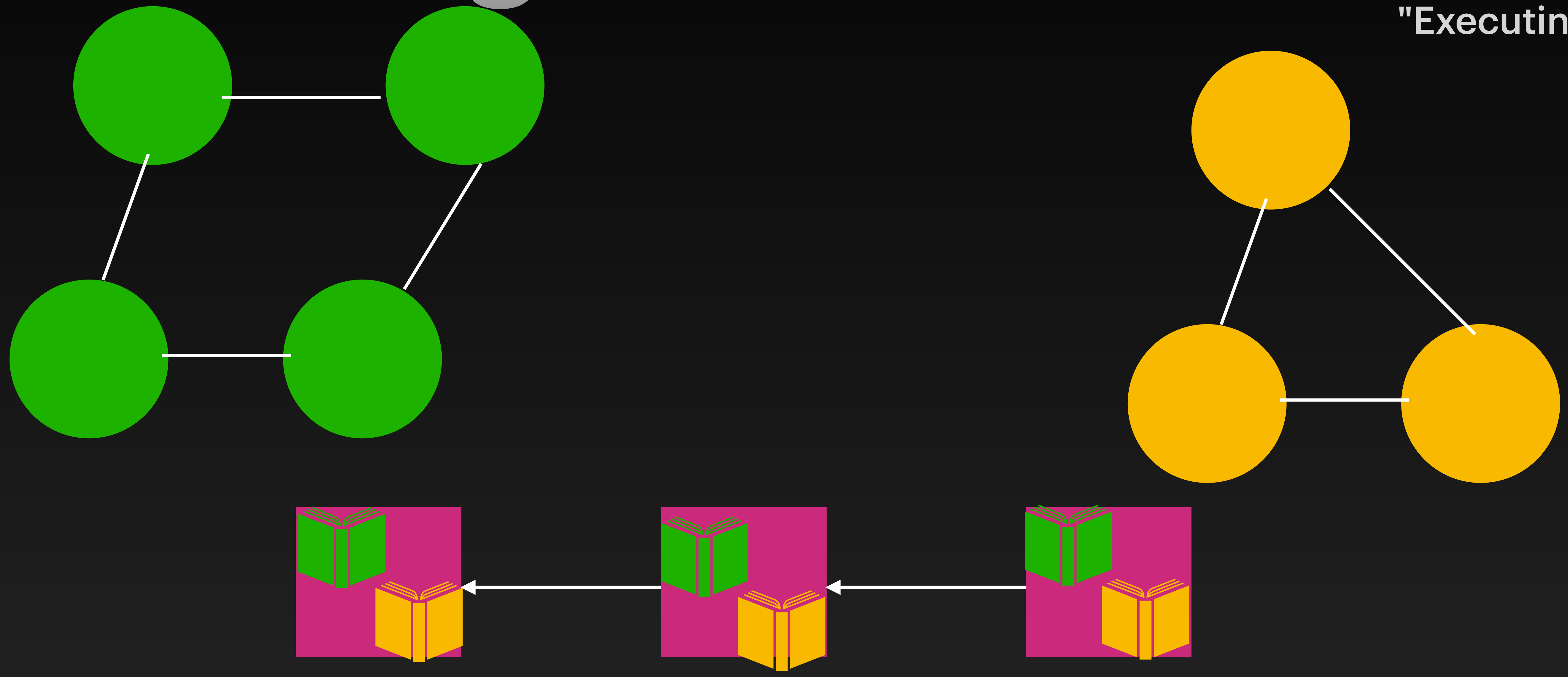
Execution

Consensus

Mempool

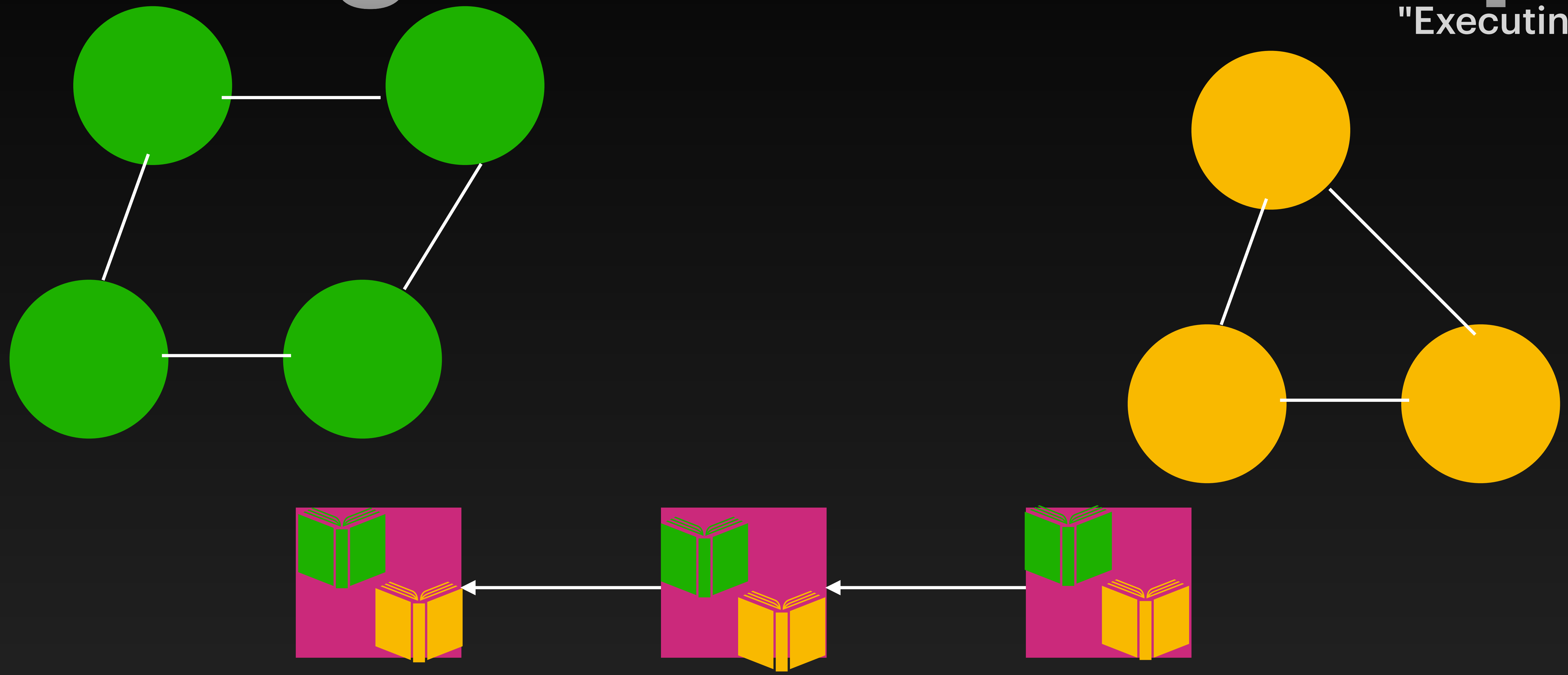
Sharding Over DAGs—Design

"Executing and proving over dirty ledgers." *FC*, 2023.



Sharding Over DAGs — Properties

"Executing and proving over dirty ledgers." *FC*, 2023.



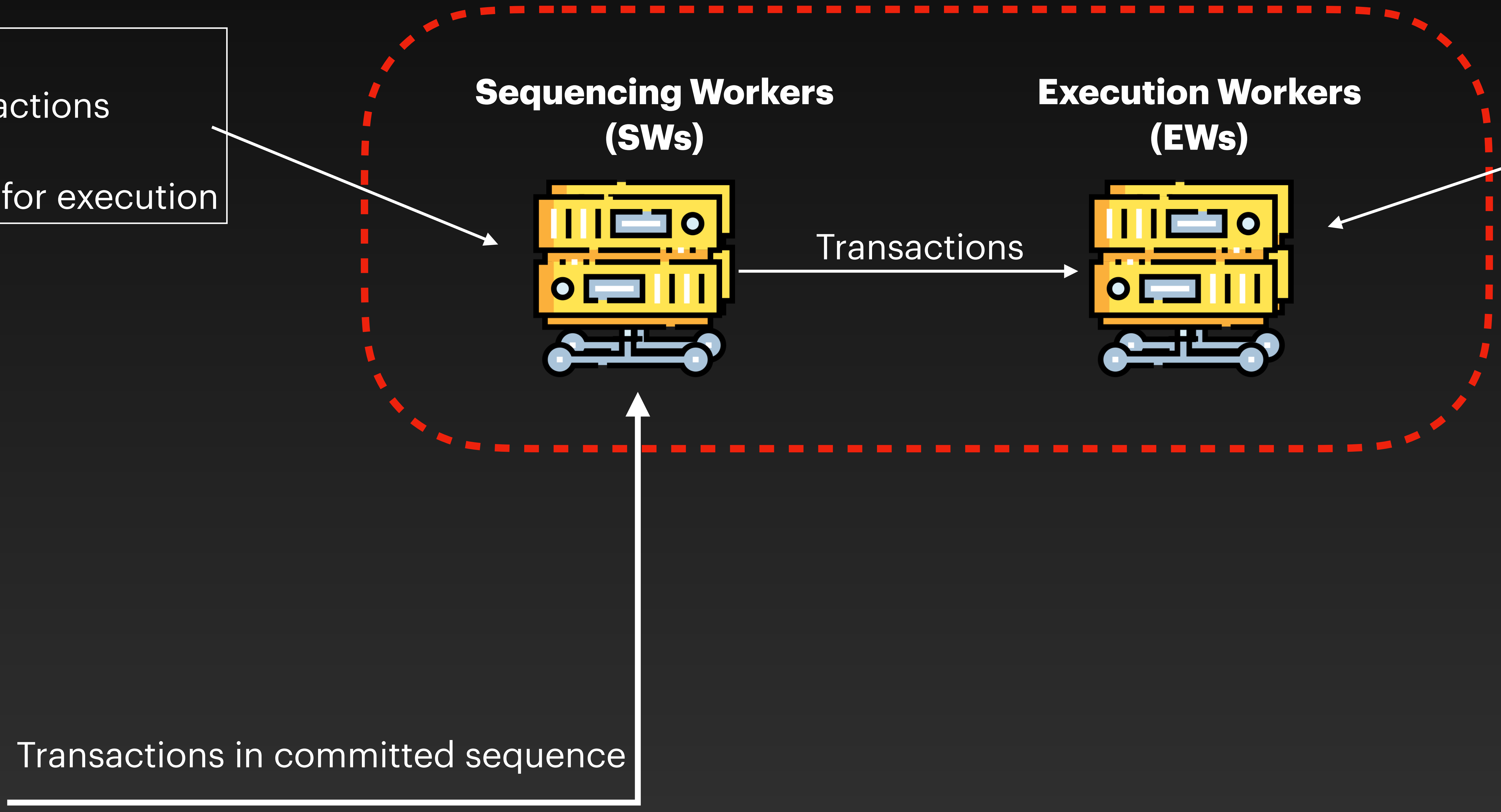
- + 51% security threshold per Shard
- + Scales-Out
- + Low Cost per Execution Node
- − Fragmenting the state-space — Expensive Atomic Commit
- − Susceptible to adaptive adversaries

Pilotfish

Distributed Transaction Execution for Lazy Blockchains

Sequence Worker

- Owns a shard of transactions
- Stores txs it owns
- Dispatches txs to EWs for execution



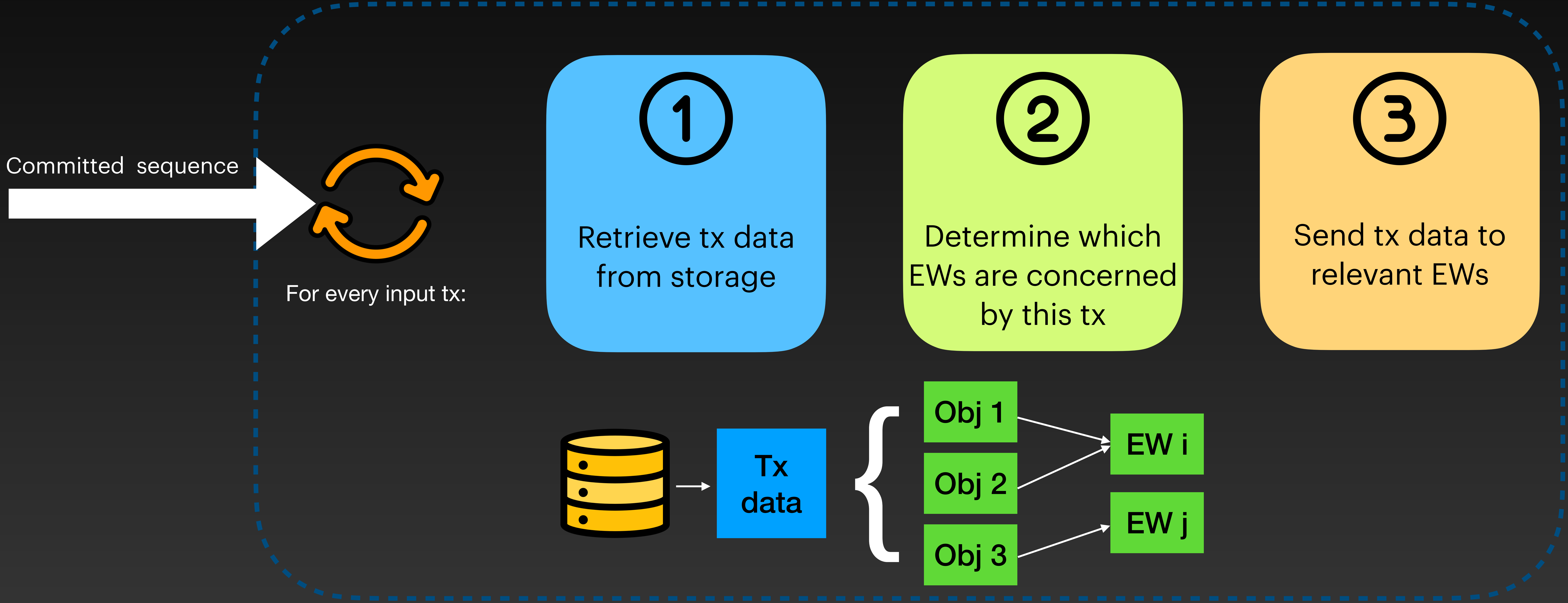
From consensus or checkpoints

Execution Worker

- Owns a shard of objects
- Stores objects it owns
- Executes txs on objects it owns
- Coordinates with other EWs

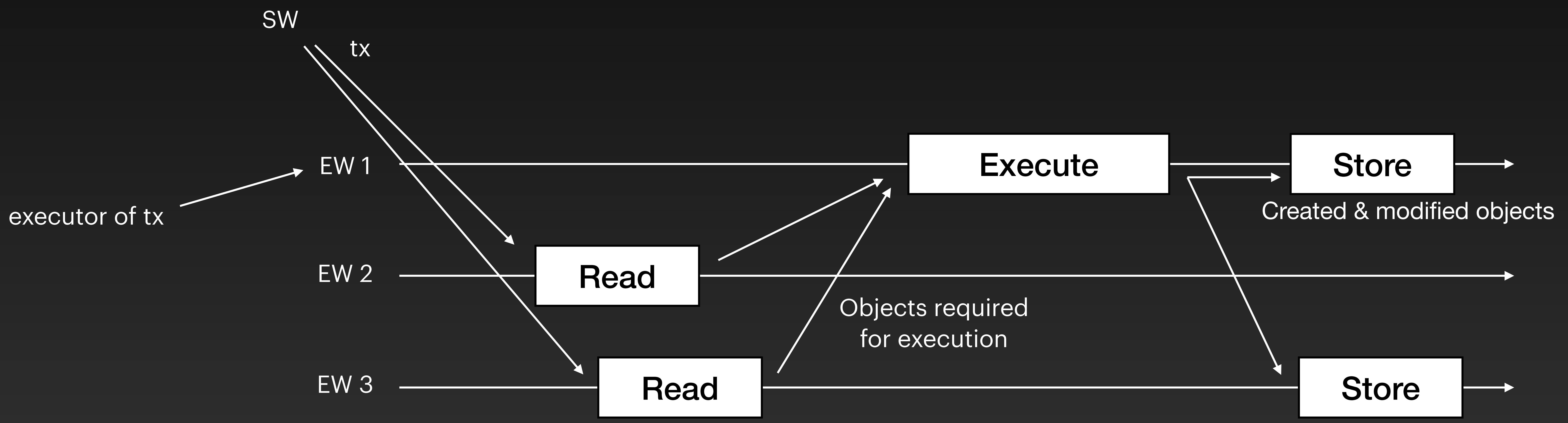
“Pilotfish: Distributed Transaction Execution for Lazy Blockchains.” *arXiv preprint arXiv:2401.16292*.

Sequence Worker (SW)



Committed sequence
For every input tx:
1
Retrieve tx data from storage
2
Determine which EWs are concerned by this tx
3
Send tx data to relevant EWs
Tx data
Obj 1
Obj 2
Obj 3
EW i
EW j

Execution Workers (EWs)



Why is this Safe in Concurrency

- The ordering of dependencies is predefined from the consensus output
- Every EW knows the version of the objects they are supposed to read/write and back pressure the SW when it is not available yet

Pilotfish —> Elastic Scaling for Web3

- + Cost scales with load, but so does profit
- + Scales-Out
- + Flat state-space
- + Consistent Threat Model

Evaluation

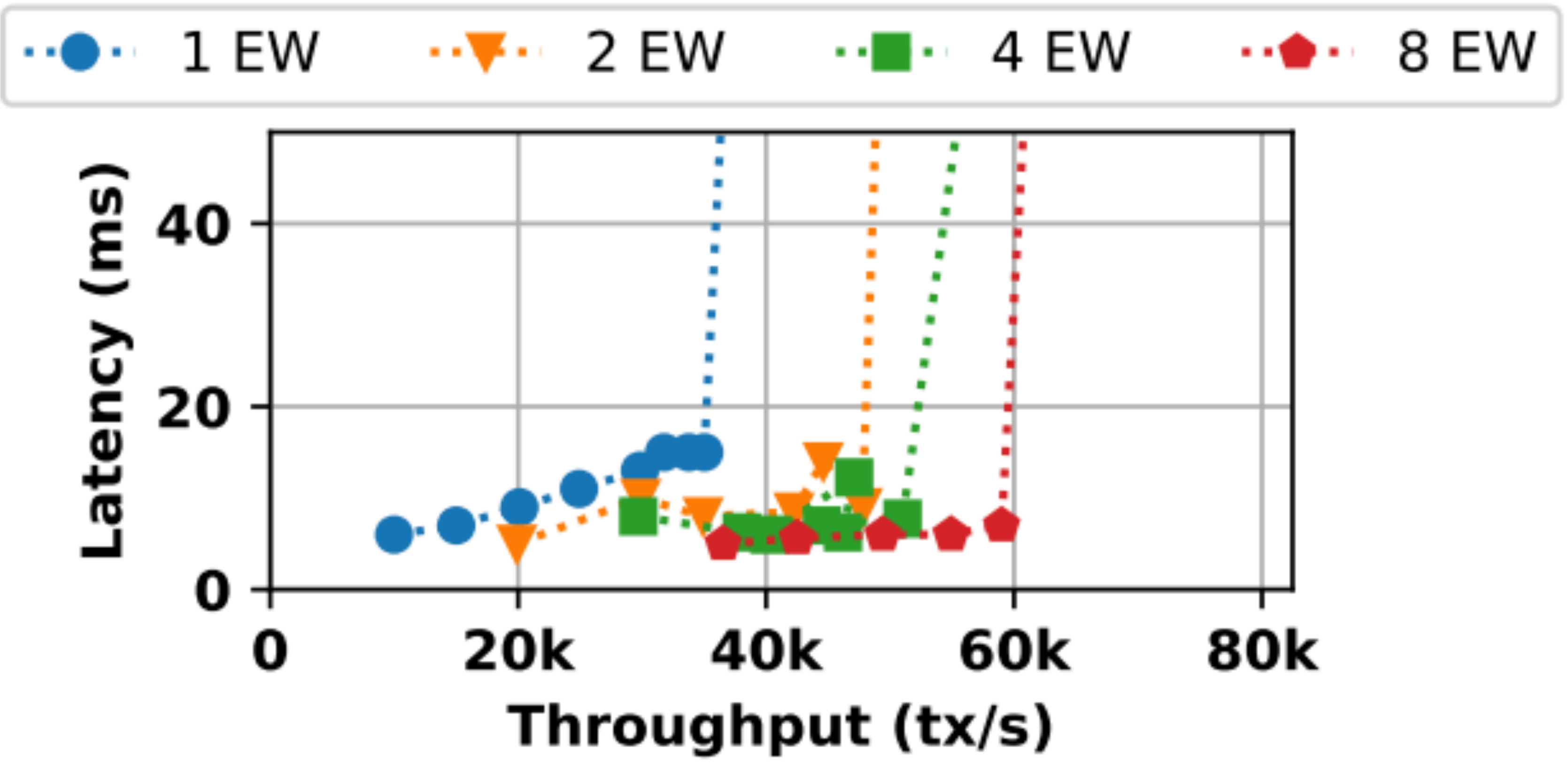


Figure 9: Pilotfish latency vs. throughput with simple transfers.

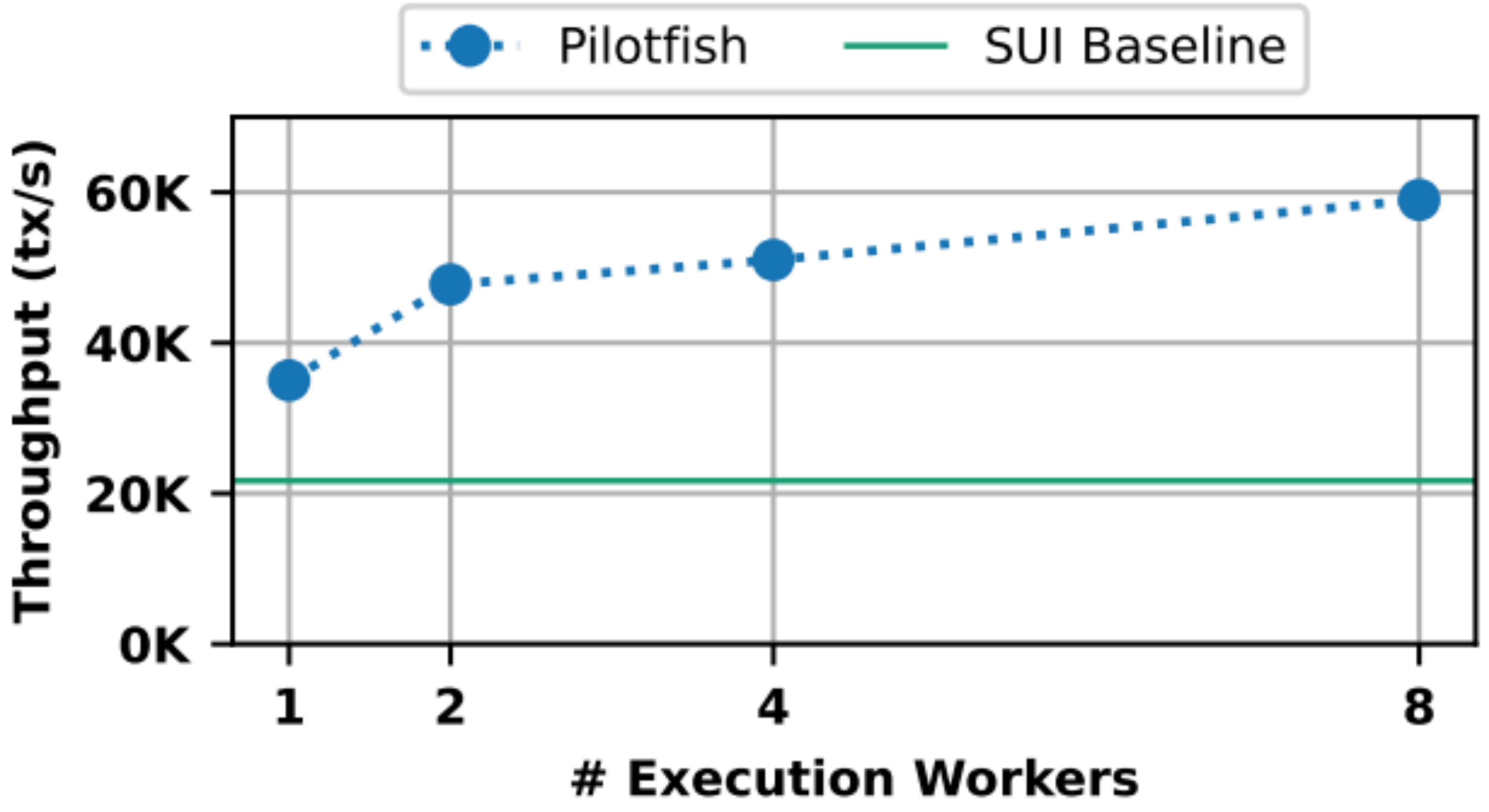


Figure 10: Pilotfish scalability with simple transfers.

Evaluation

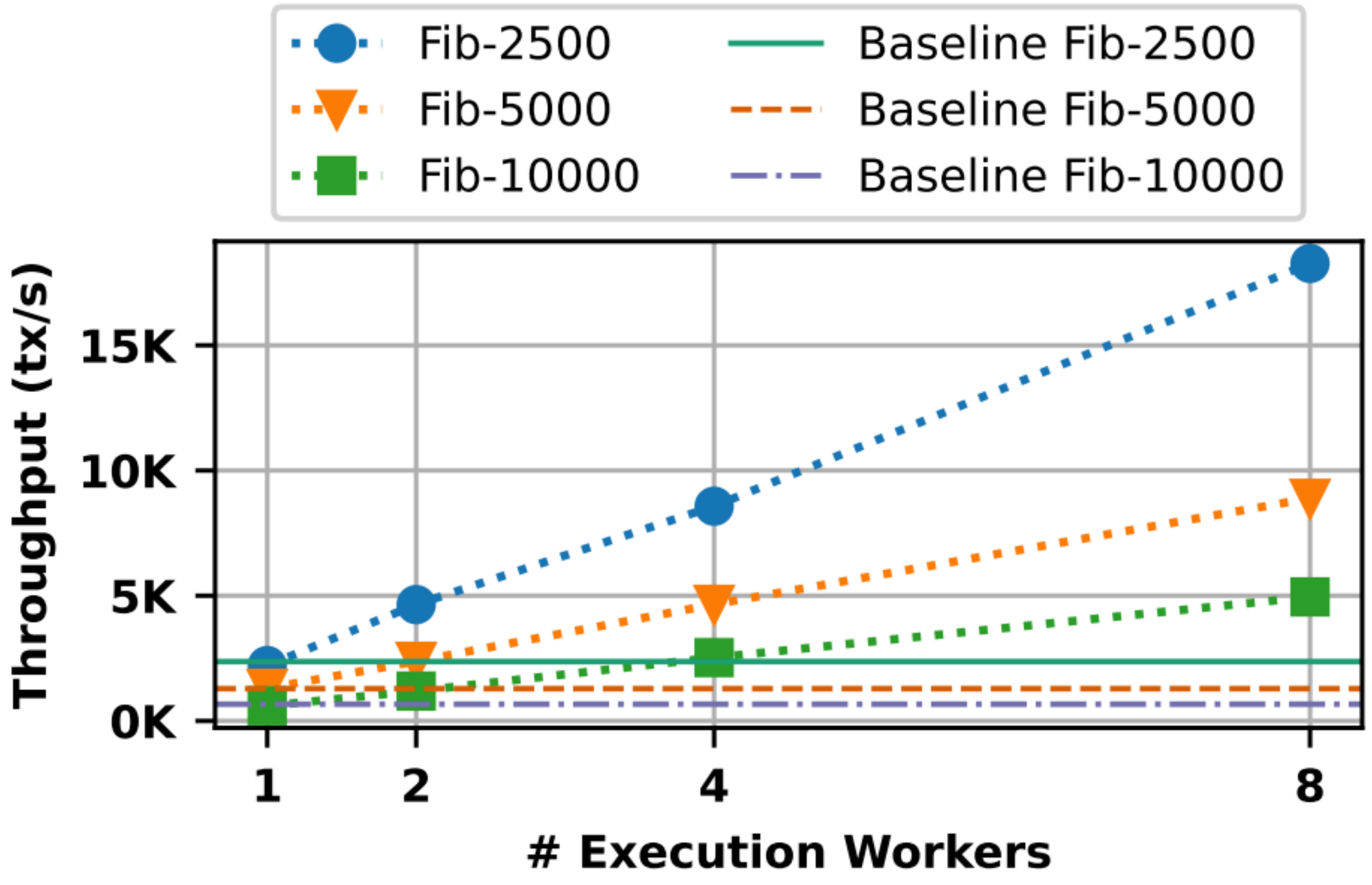
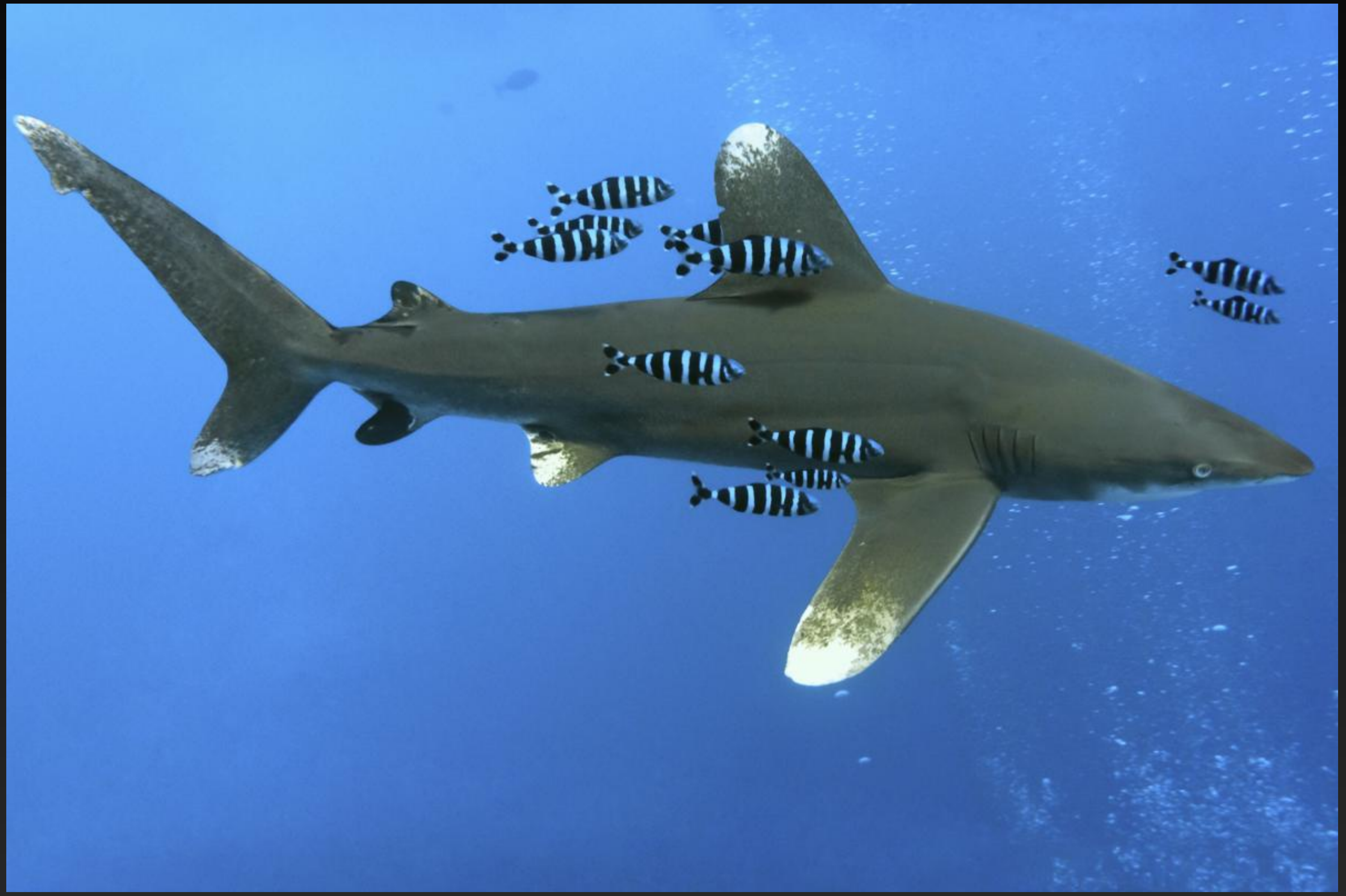


Figure 12: Pilotfish scalability with computationally heavy transactions. `Fib-X` means that each transaction computes the X-th Fibonacci number. The horizontal lines show the single-machine throughput of the baseline on the same workloads.

- Pilotfish over Bullshark provides the first ***end-to-end Elastic Distributed Ledger***
- Pilotfish does not employ batching —> *Latencies of 20-50ms post-consensus*
- Pilotfish is co-designed with the blockchain —> *Light worker recovery*



Side-Stepping Consensus

Consensus is not required

- Coins, balances, and transfers
- NFTs creation and transfers
- Game logic allowing users to combine assets
- Inventory management for games / metaverse
- Auditable 3rd party services not trusted for safety
- ...

New Architecture

The Sui System

Consensus only when you need to

New Architecture

Architecture

Owned Objects

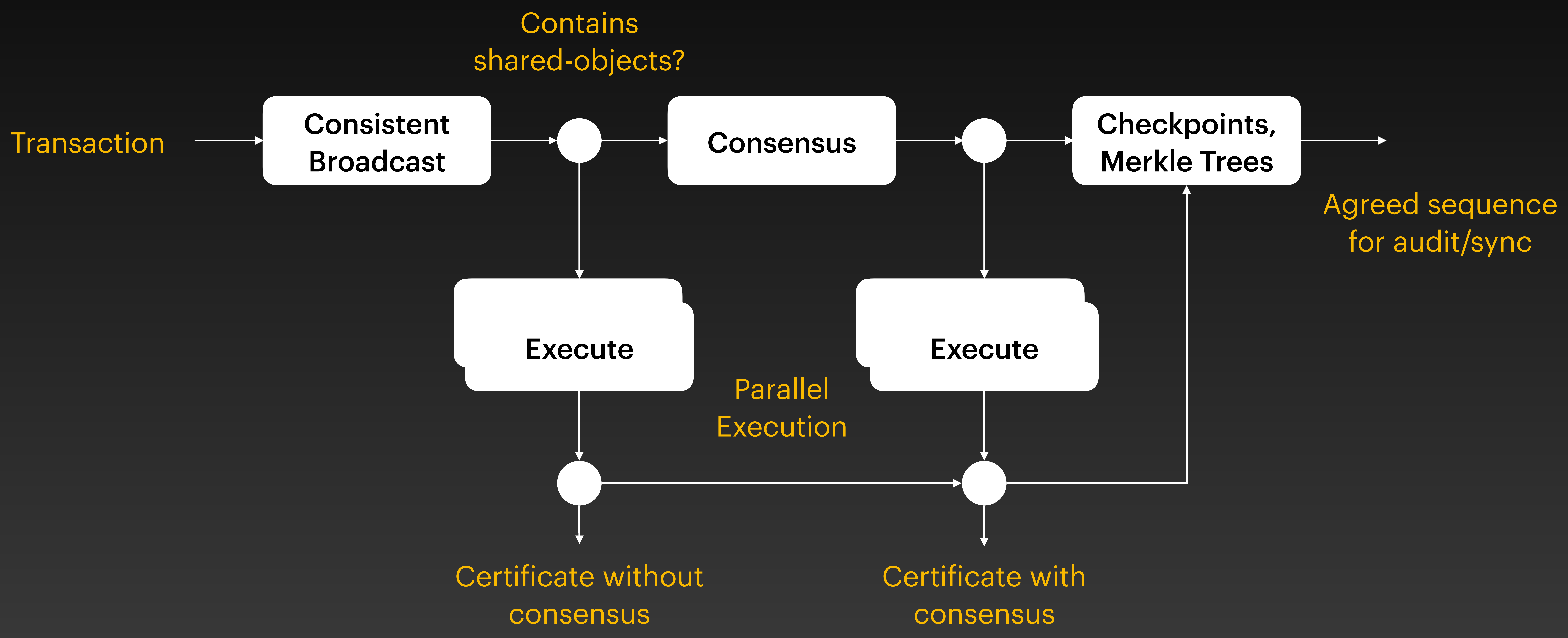
- Objects that can be mutated by a single entity
- e.g., My bank account
- **Do not need consensus**

Shared Objects

- Objects that can be mutated my multiple entities
- e.g., A global counter
- **Need consensus**

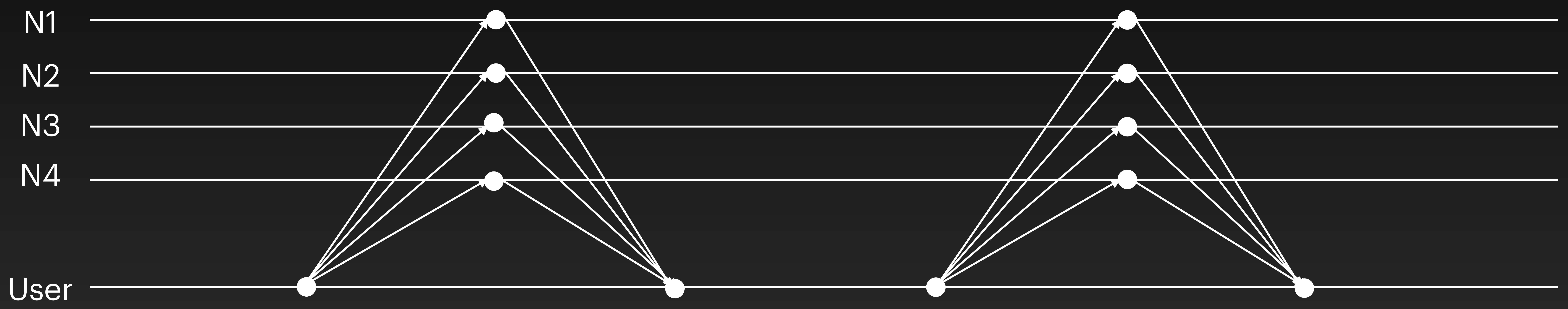
The Sui System

Architecture



The Sui System

Consensus-less Path



Send T1:

Disseminate the transaction

Echo T1:

Nodes check and sign T1

Cert T1:

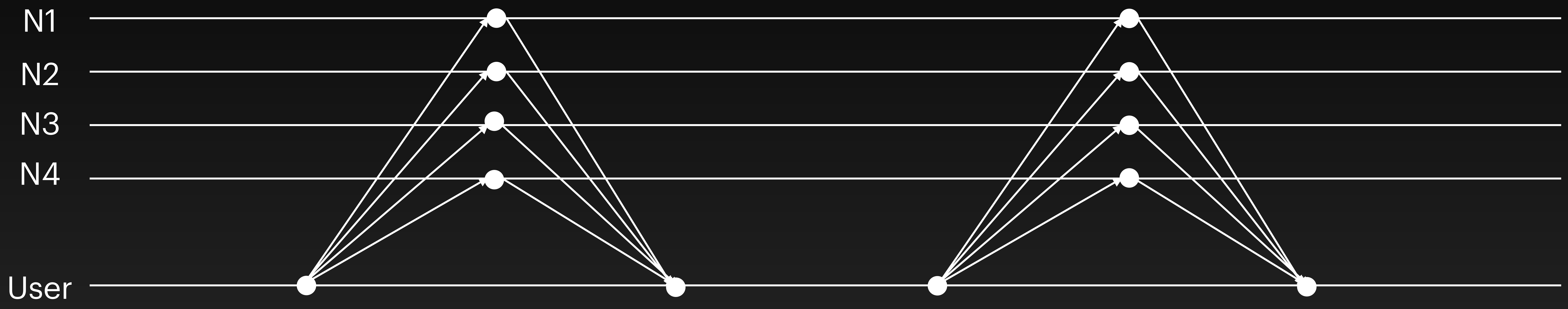
User gather >2/3 signatures into a certificate and disseminate it

Effect T1:

User gather >2/3 effect signatures for finality

The Sui System

Consensus-less Path



- ➕ Low Latency
- ➕ Trivial to Scale Out
- ➖ Reconfiguration
- ➖ Equivocation results in loss of liveness

Side-Stepping Consensus

Safe reconfiguration

Side-Stepping Consensus

Equivocation Tolerence