Efficient DAG-Based Consensus

Sui Eng Offsite 22

Alberto Sonnino

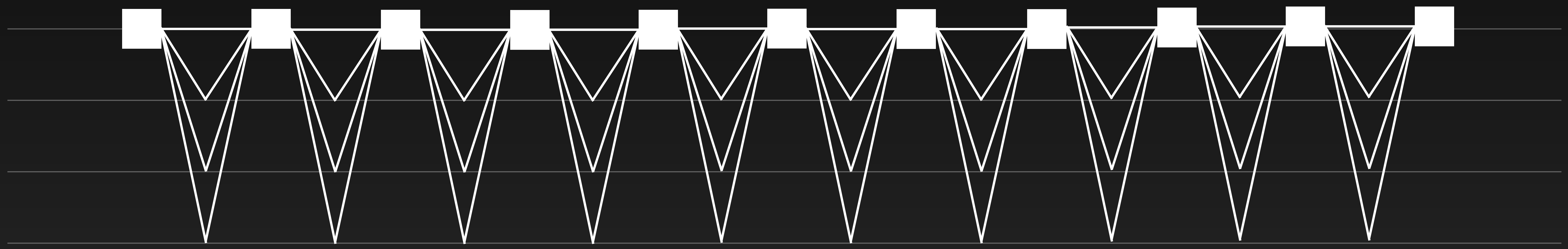
Traditional Designs

Observation

- Monolithic protocol sharing transaction data as part of the consensus
- Optimize overall message complexity of the consensus protocol

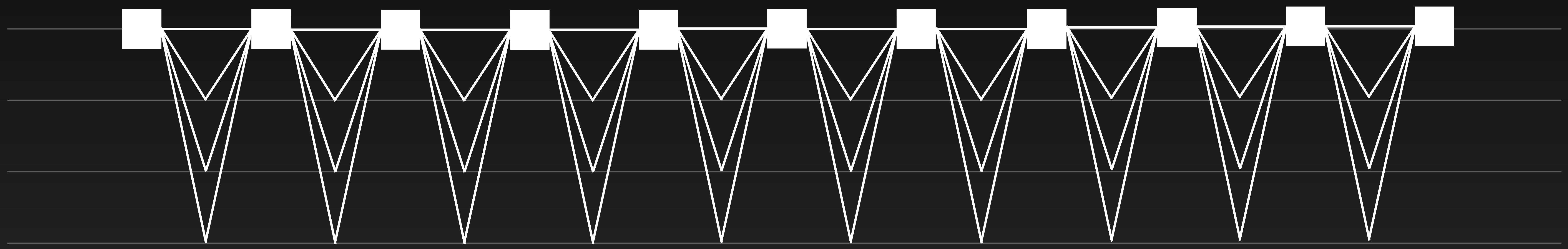
Current Designs

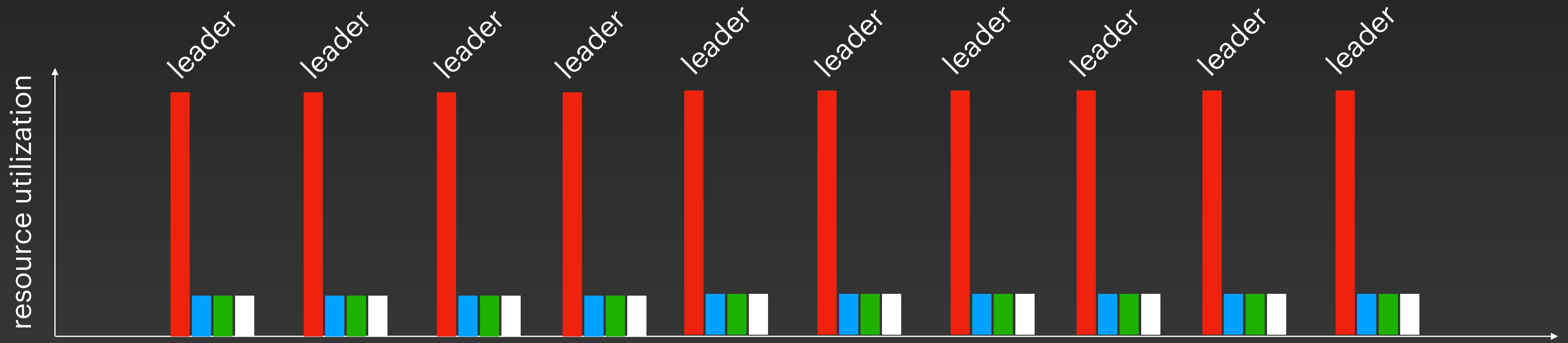
Typical leader-based protocols



Current Designs

Typical leader-based protocols





Data dissemination is the key

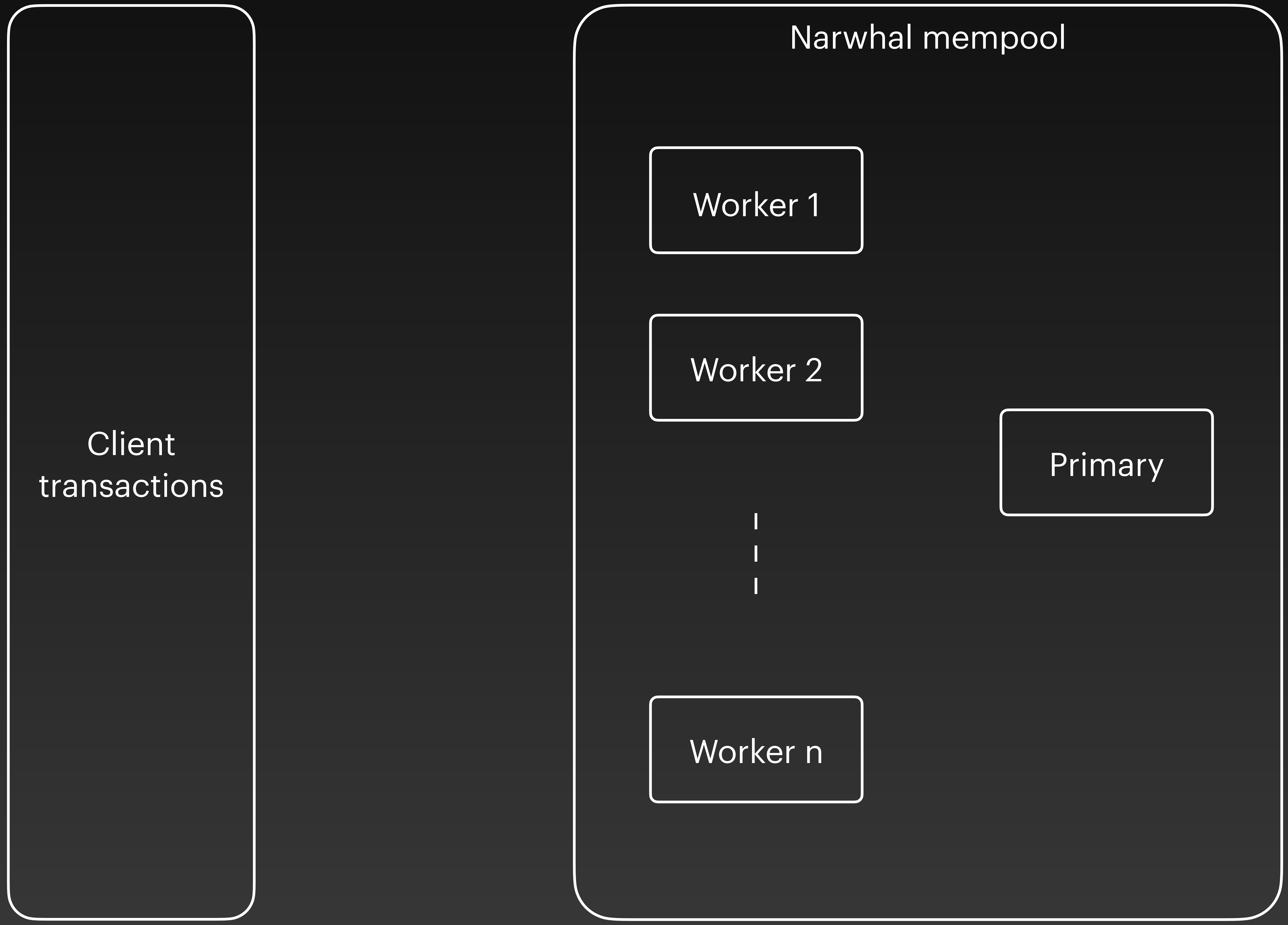
Reaching consensus on metadata is cheap

Narwhal

Dag-based mempool

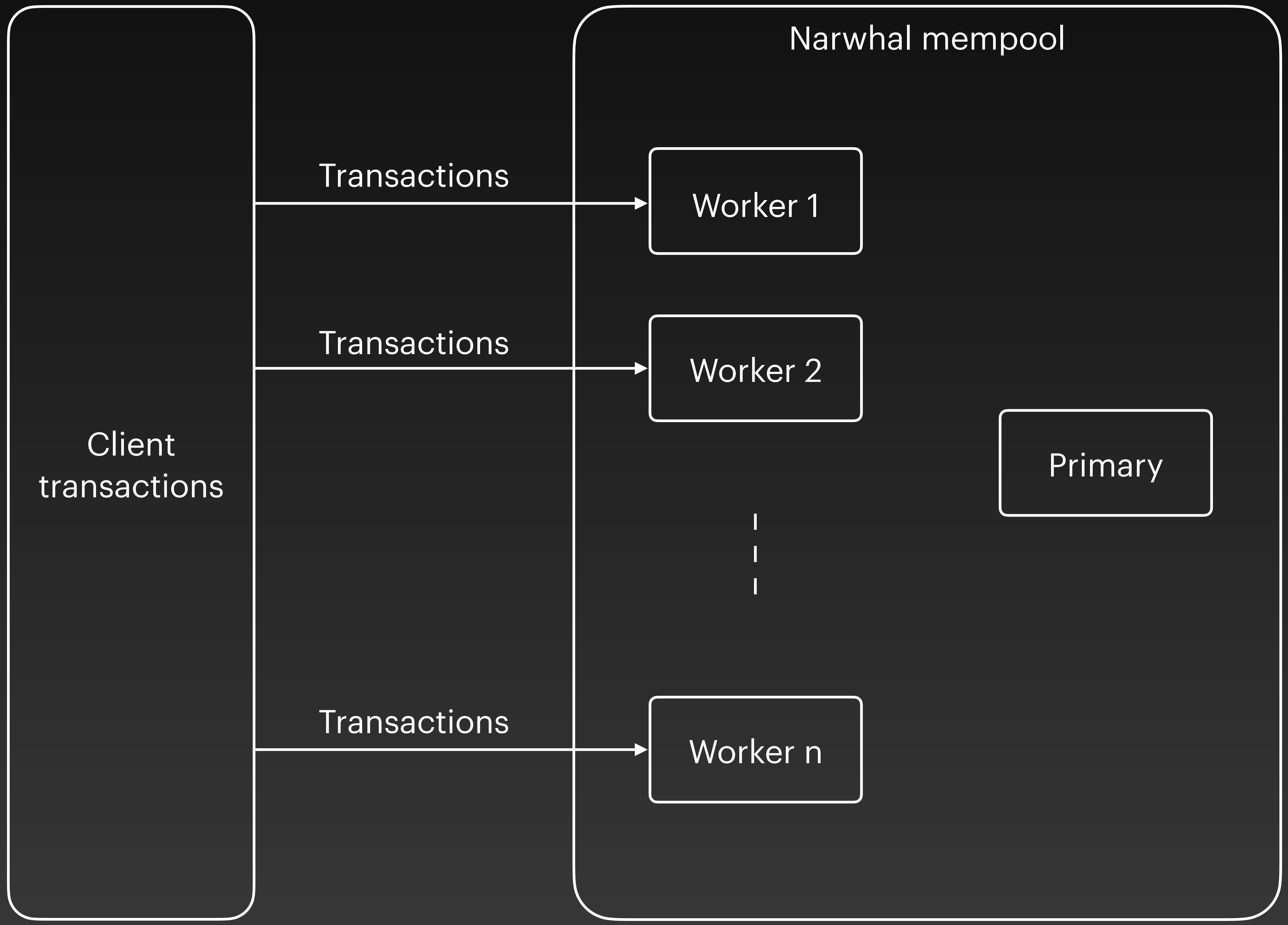
Narwhal

The workers and the primary



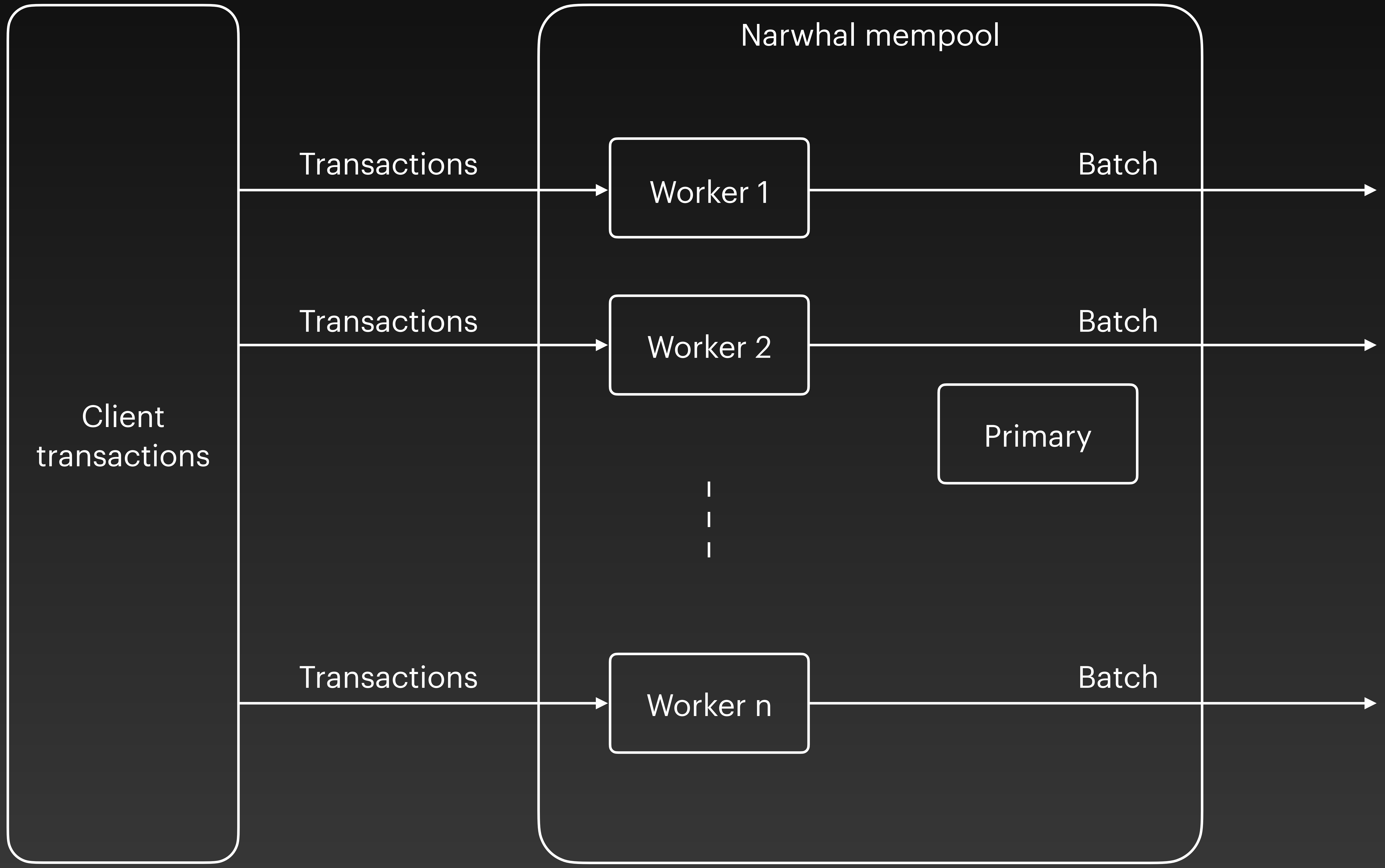
Narwhal

The workers and the primary



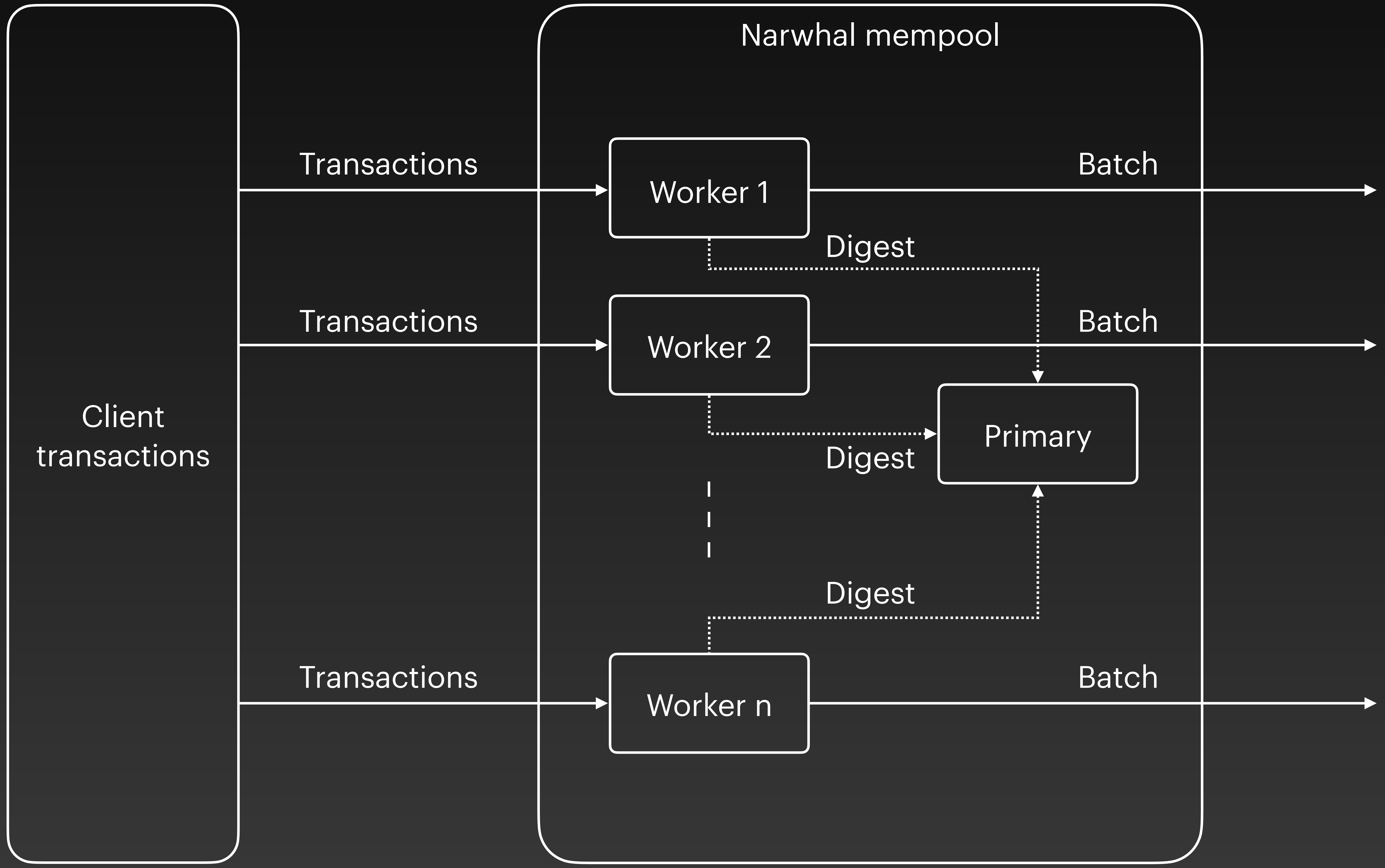
Narwhal

The workers and the primary



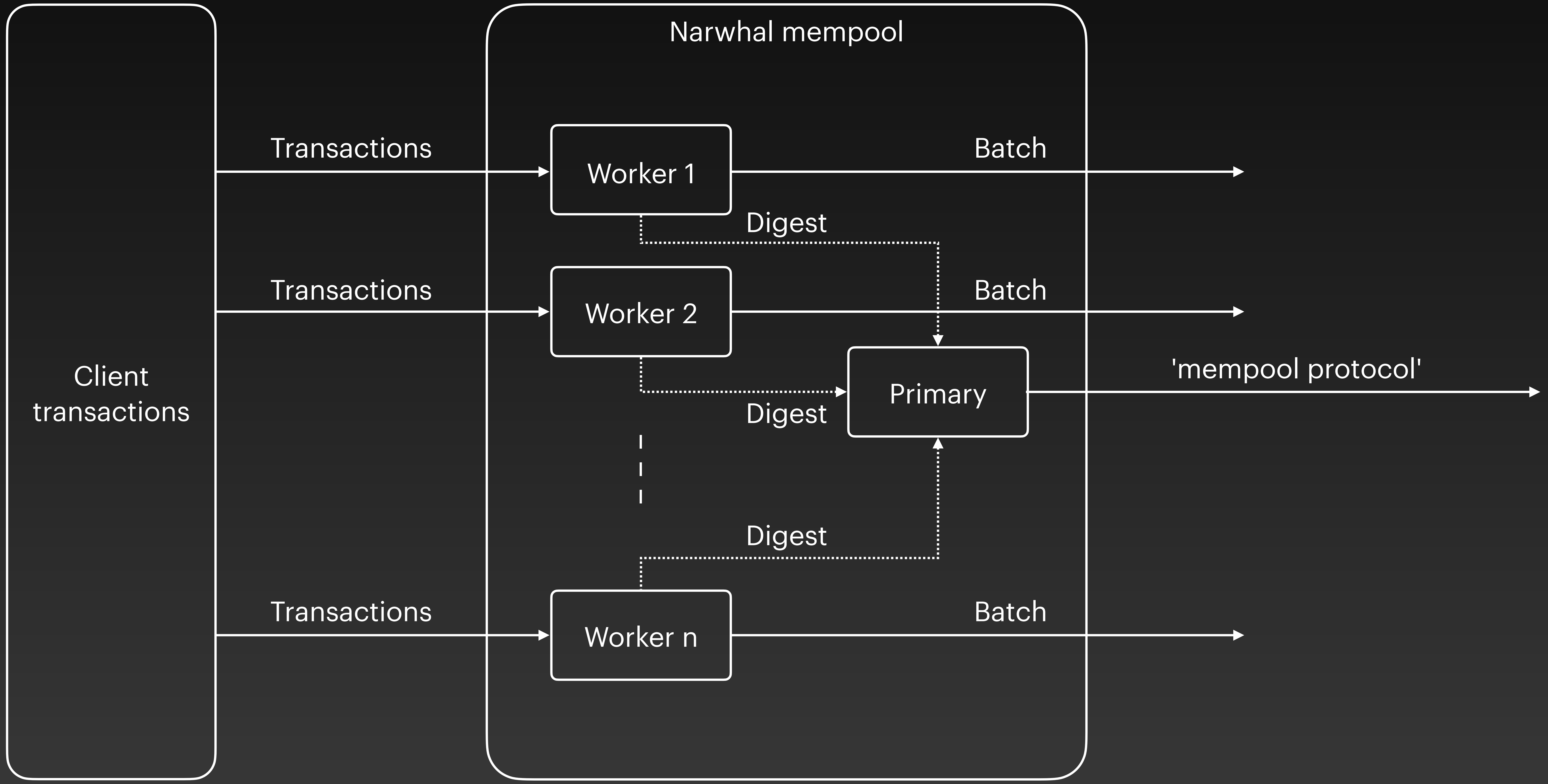
Narwhal

The workers and the primary



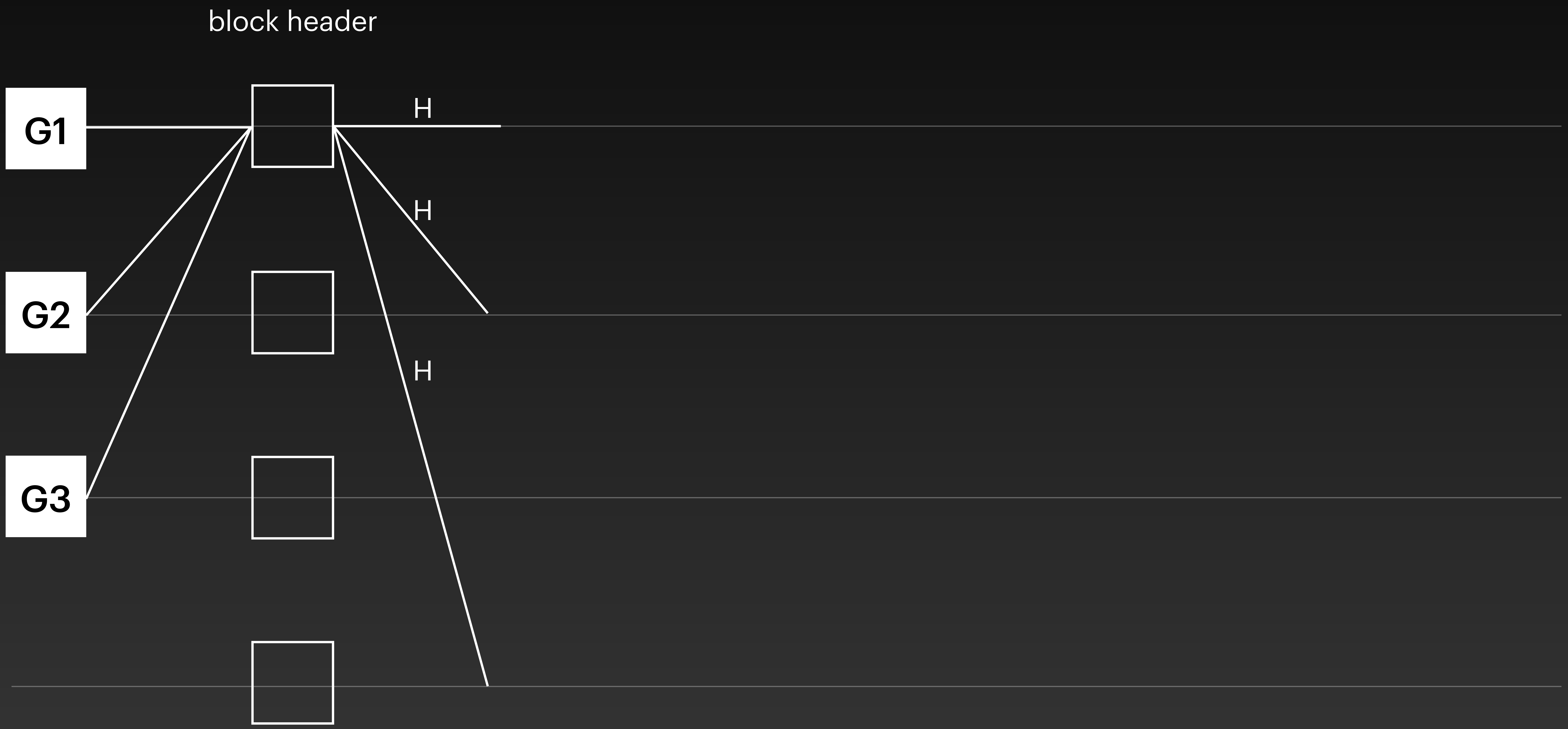
Narwhal

The workers and the primary



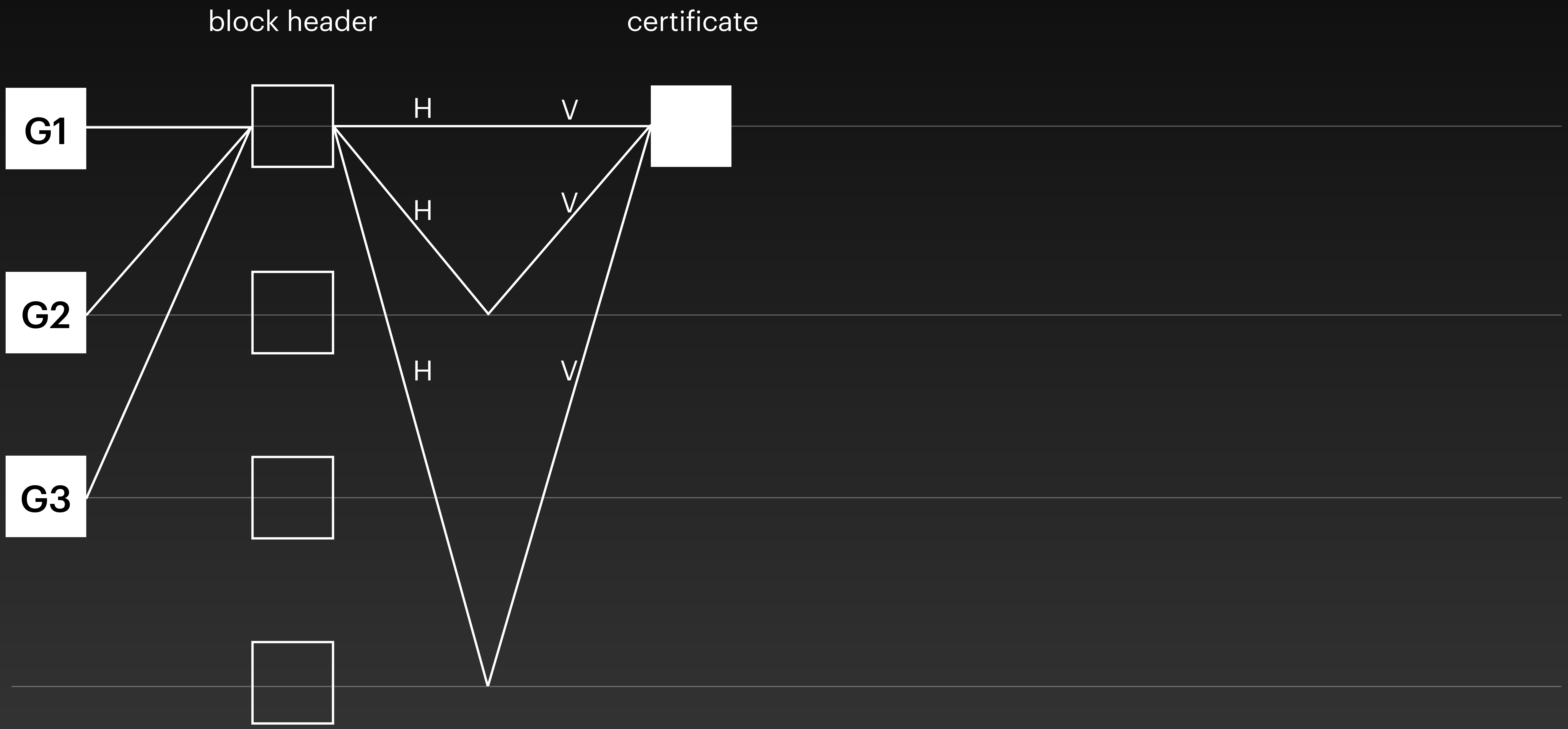
Narwhal

The primary machine



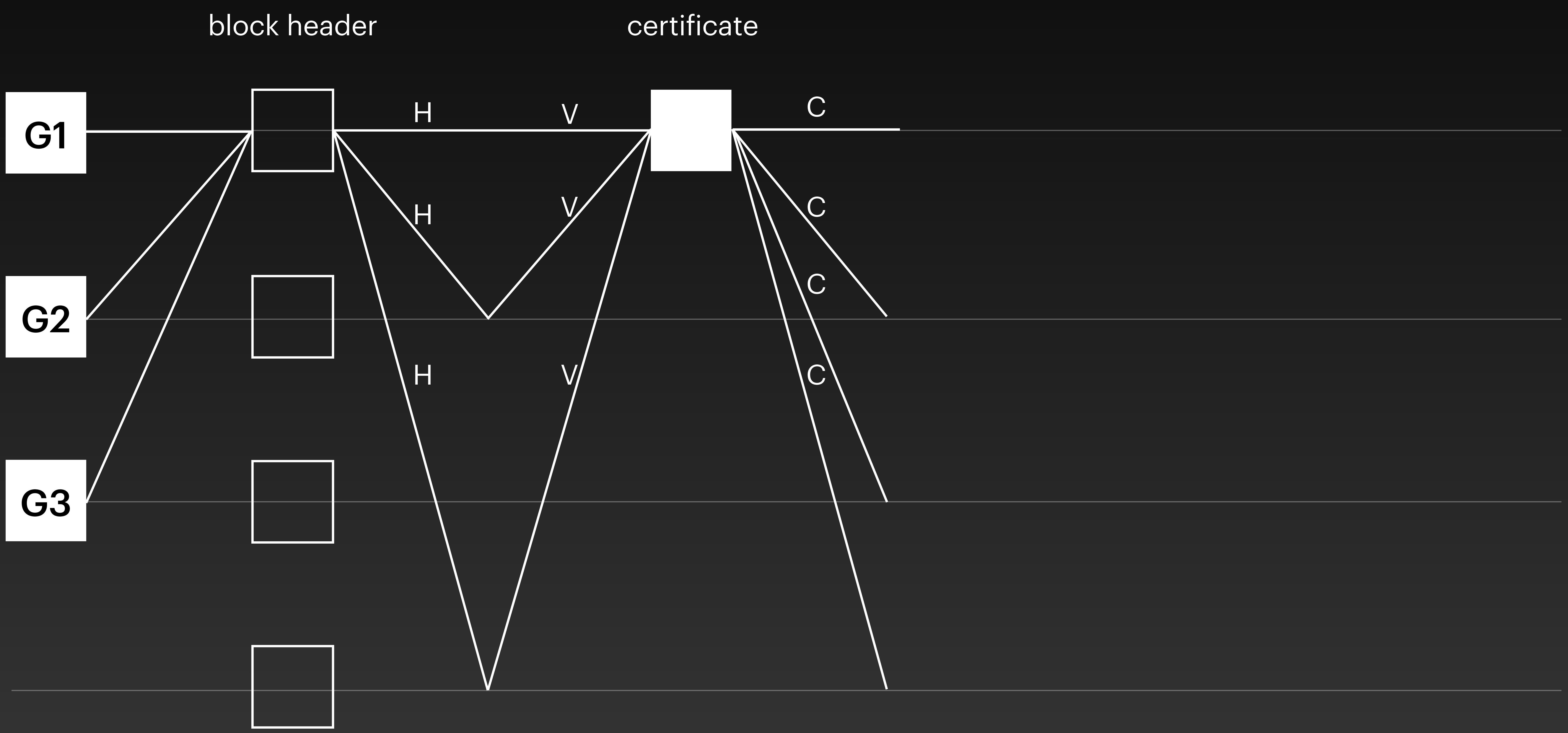
Narwhal

The primary machine



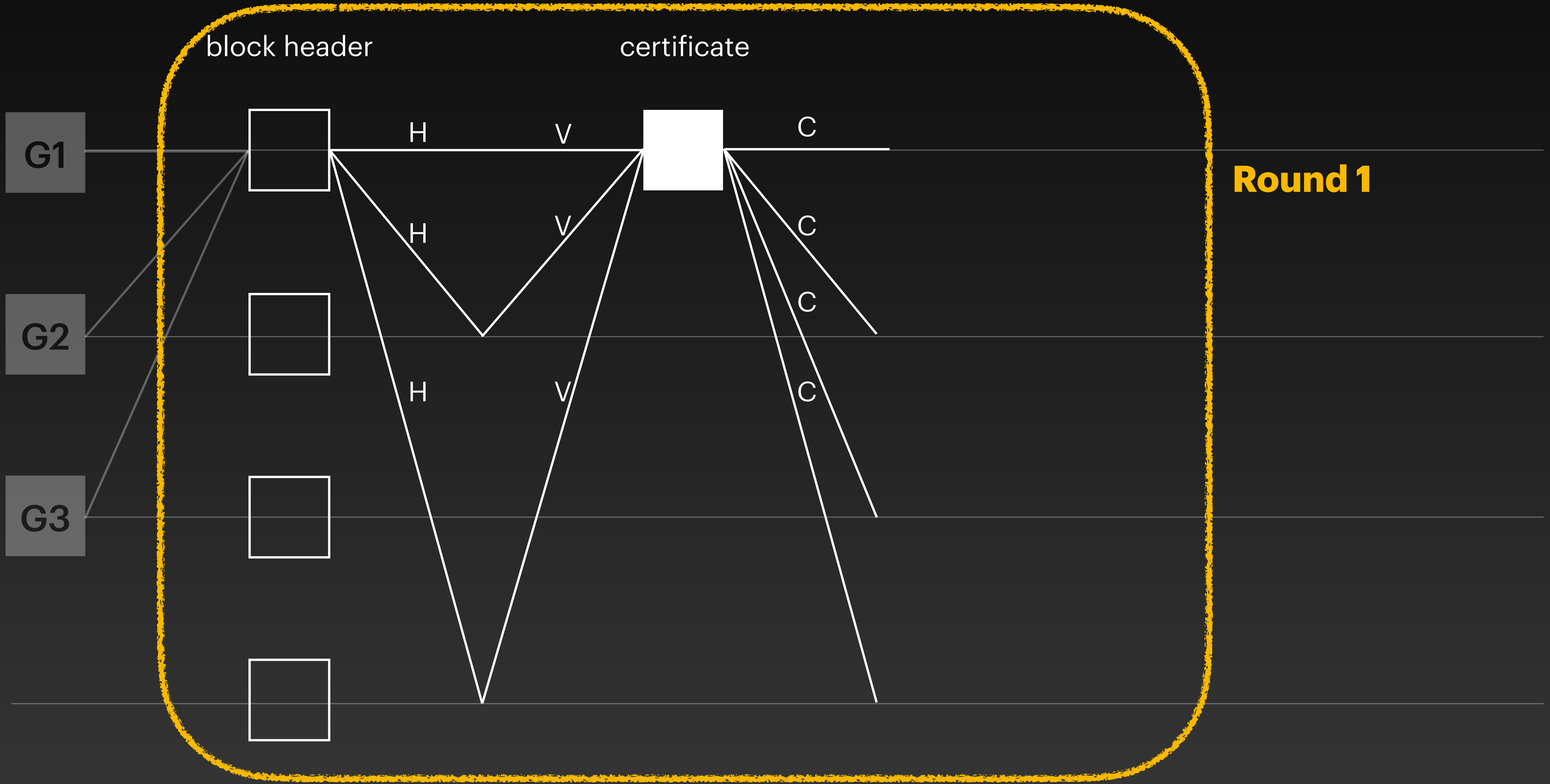
Narwhal

The primary machine



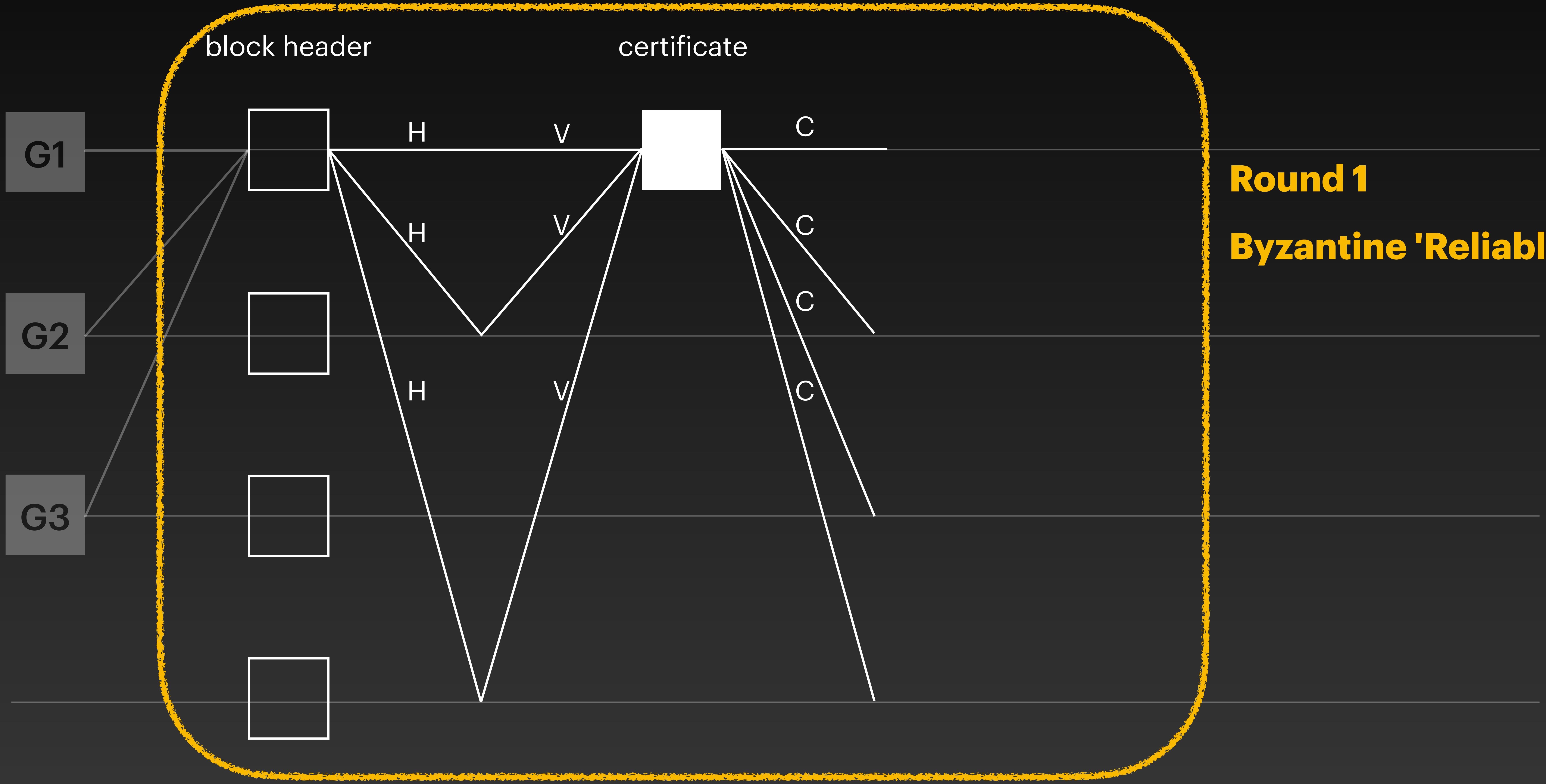
Narwhal

The primary machine



Narwhal

The primary machine

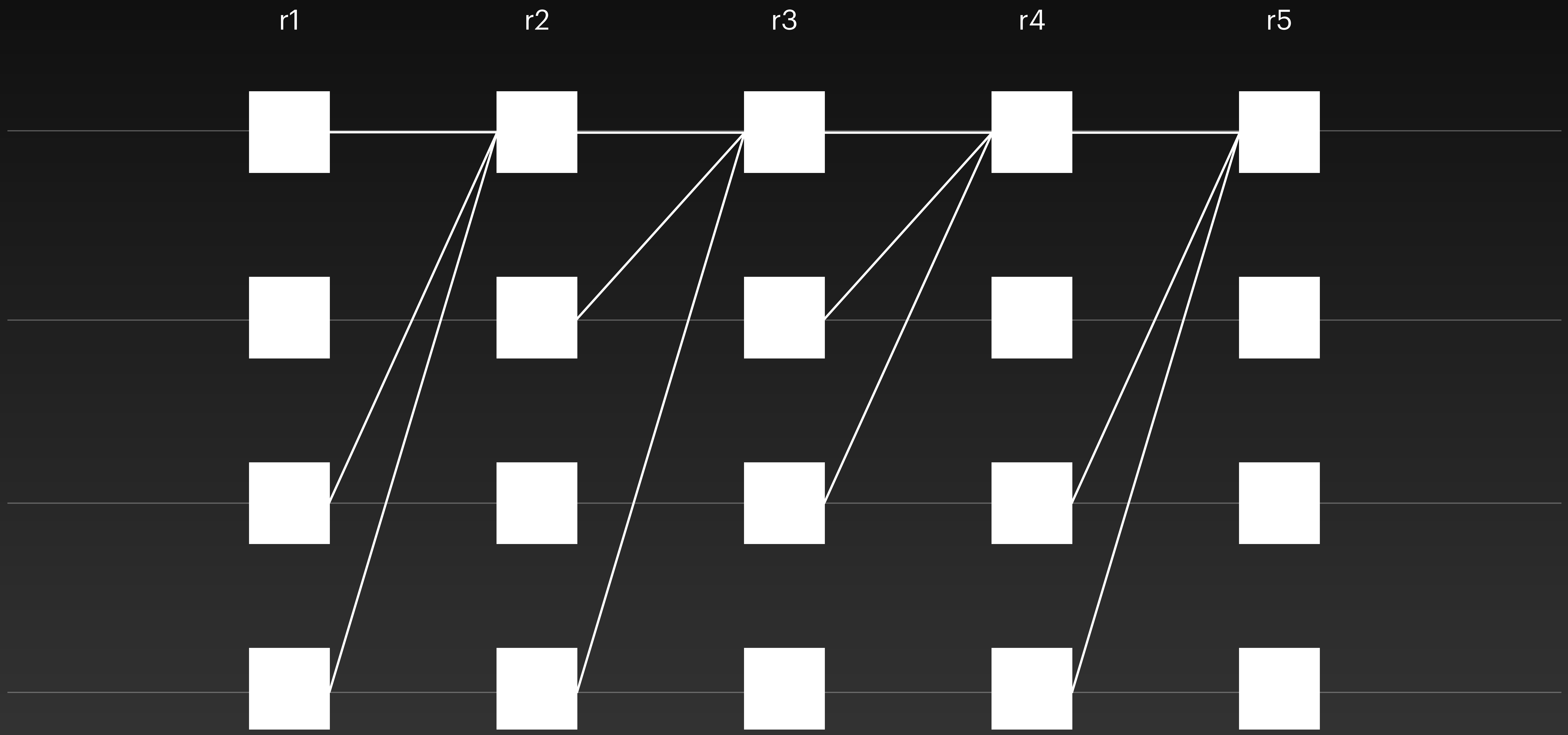


Round 1

Byzantine 'Reliable' Broadcast

Narwhal

The primary machine



Narwhal

Data Dissemination & Proof of Availability

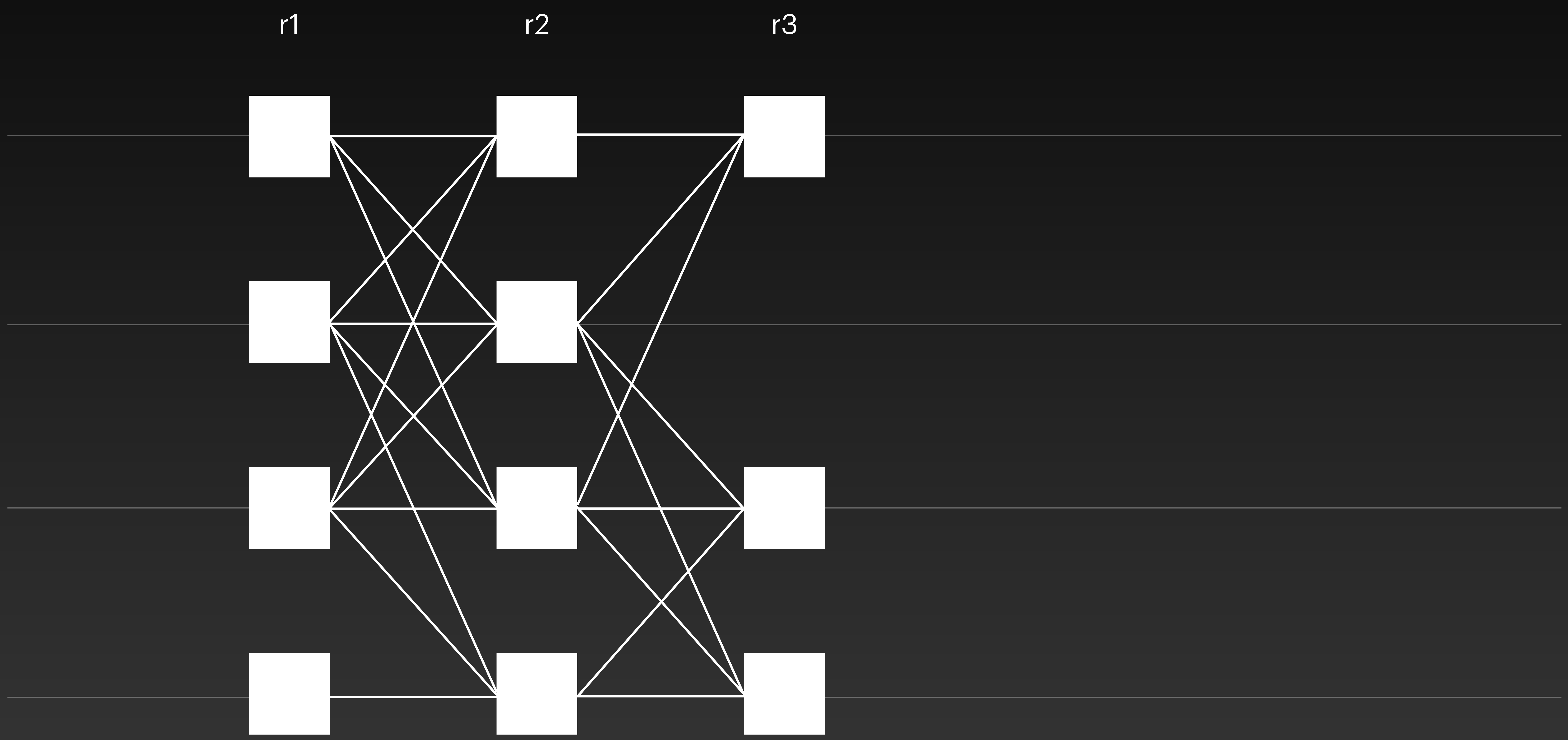
- The workers ship batch of transactions
- Many workers to scale out and use resources concurrently
- The primary constantly broadcasts the batch digests
- Headers at round *r* contains references to *2f+1* certificates of round *r-1*
- Build a structured DAG of certificates

Tusk

Zero-message asynchronous consensus

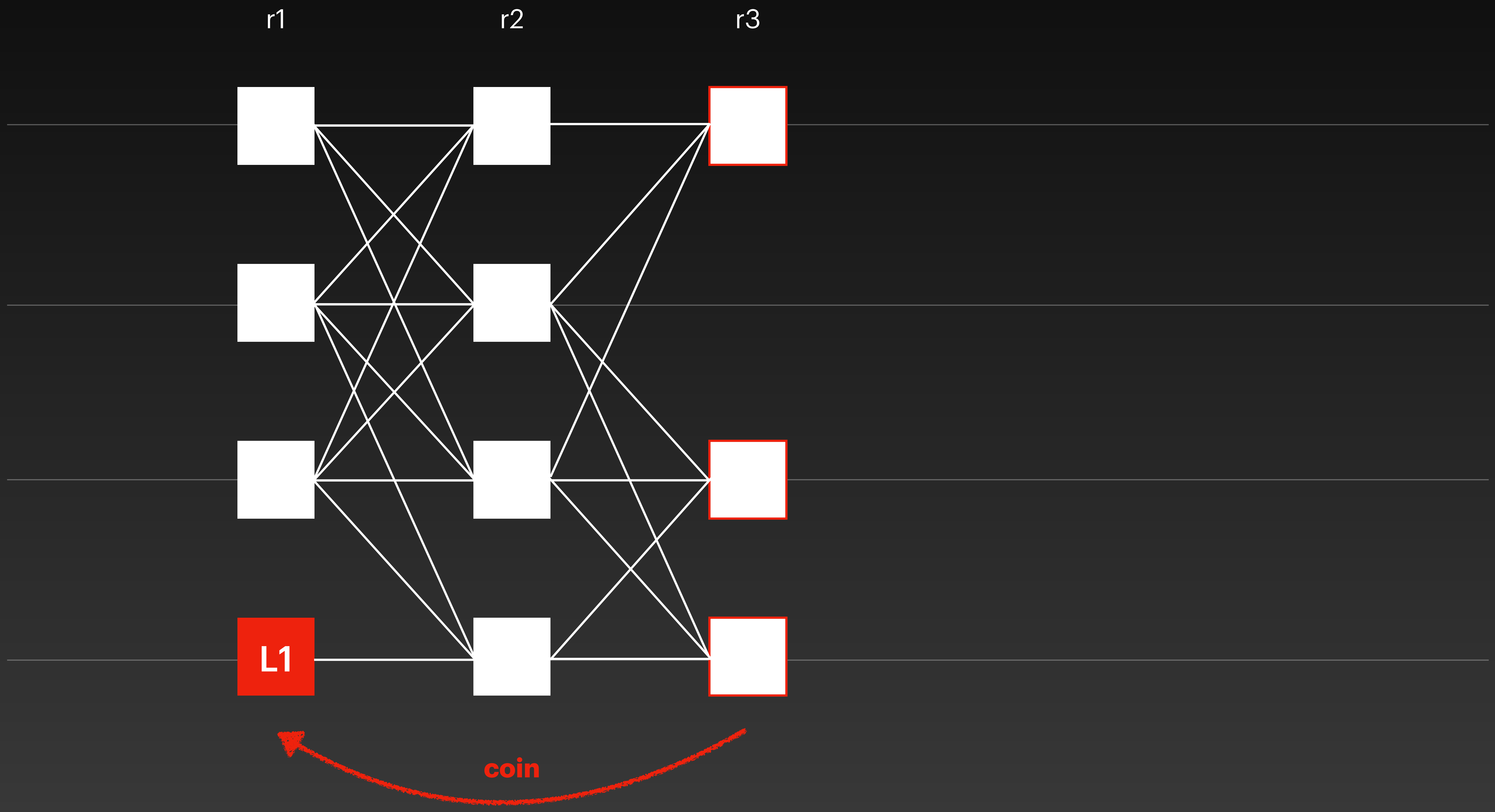
Tusk

Add common coin & Interpret the DAG



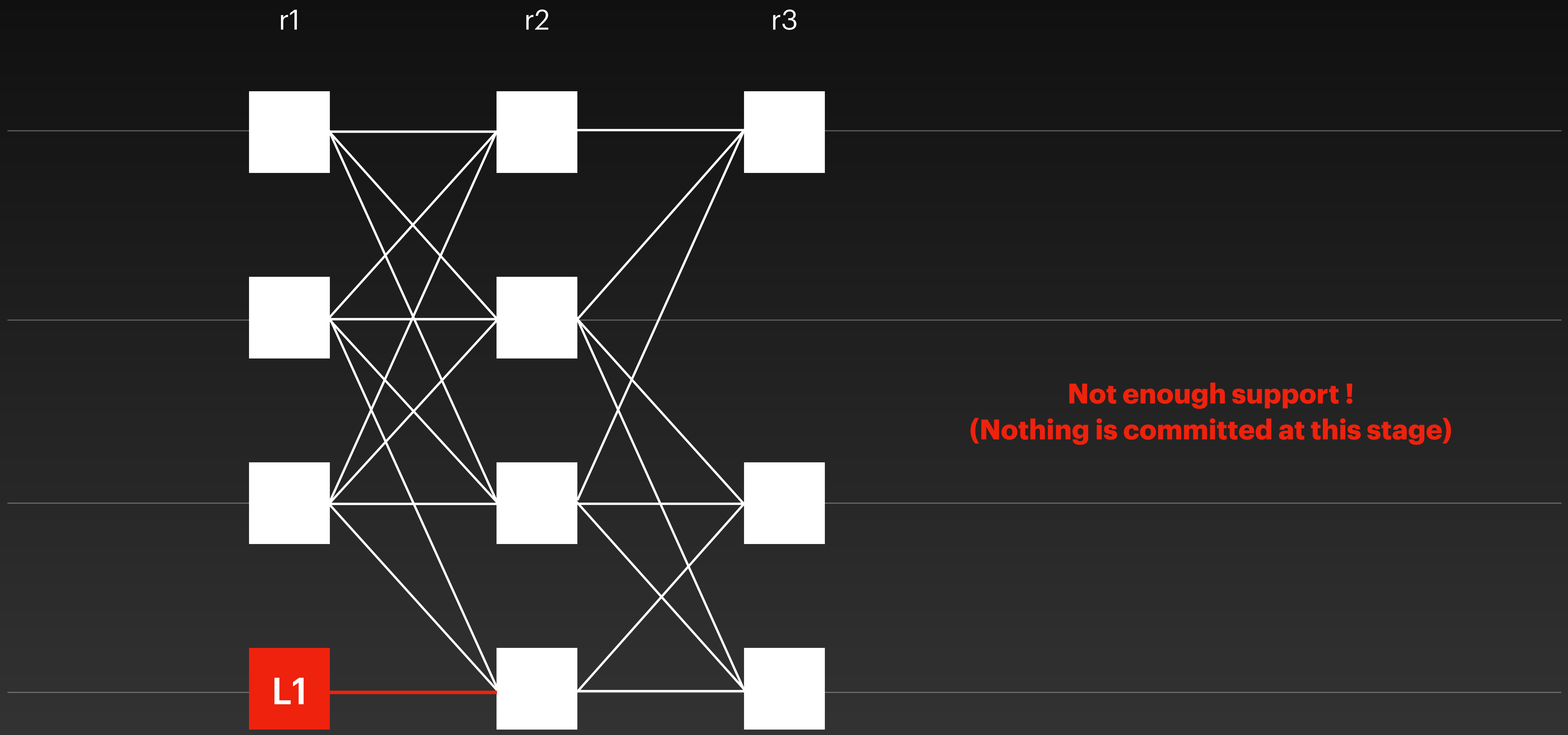
Tusk

The random coin elects the leader of r-2



Tusk

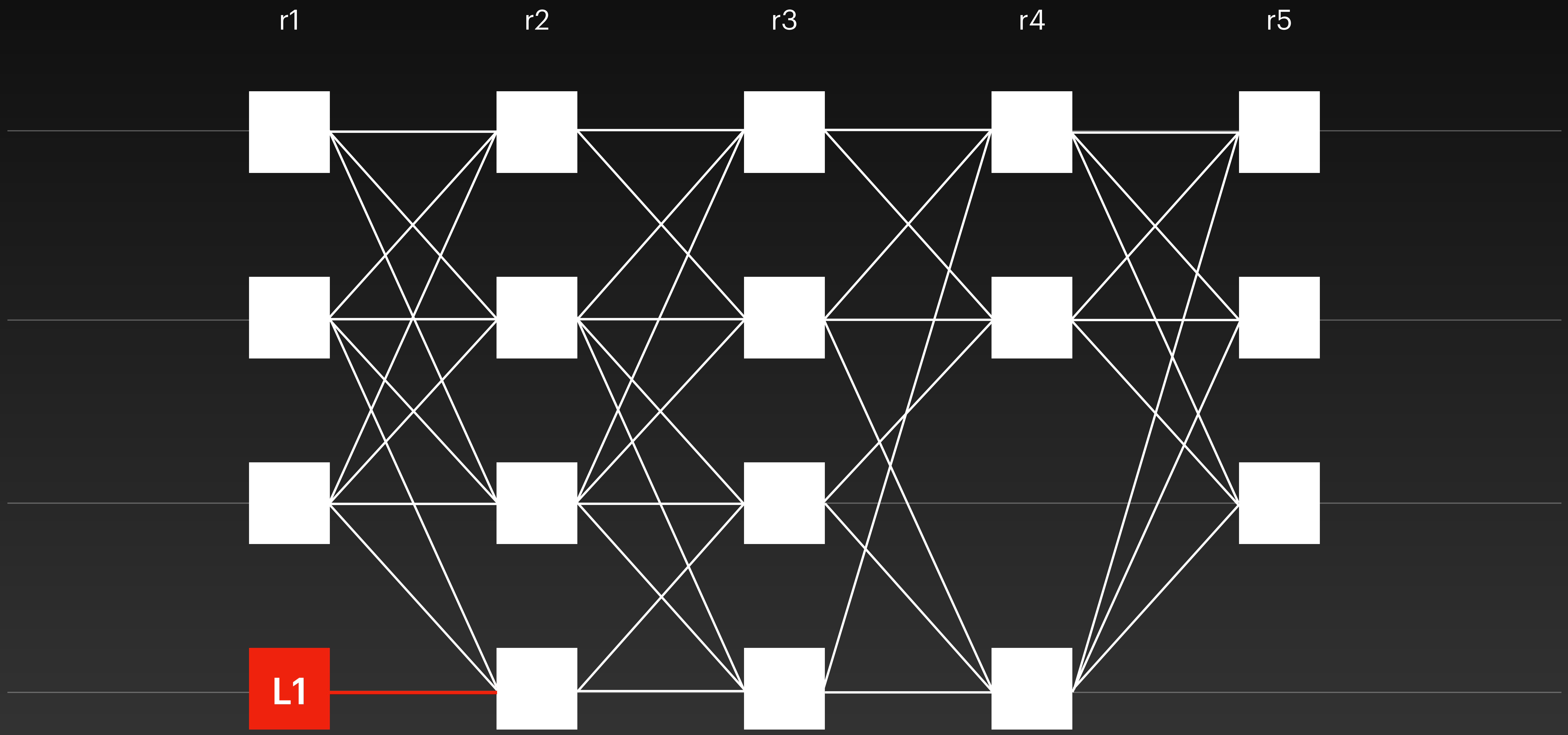
The leader needs f+1 links from round r-1



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

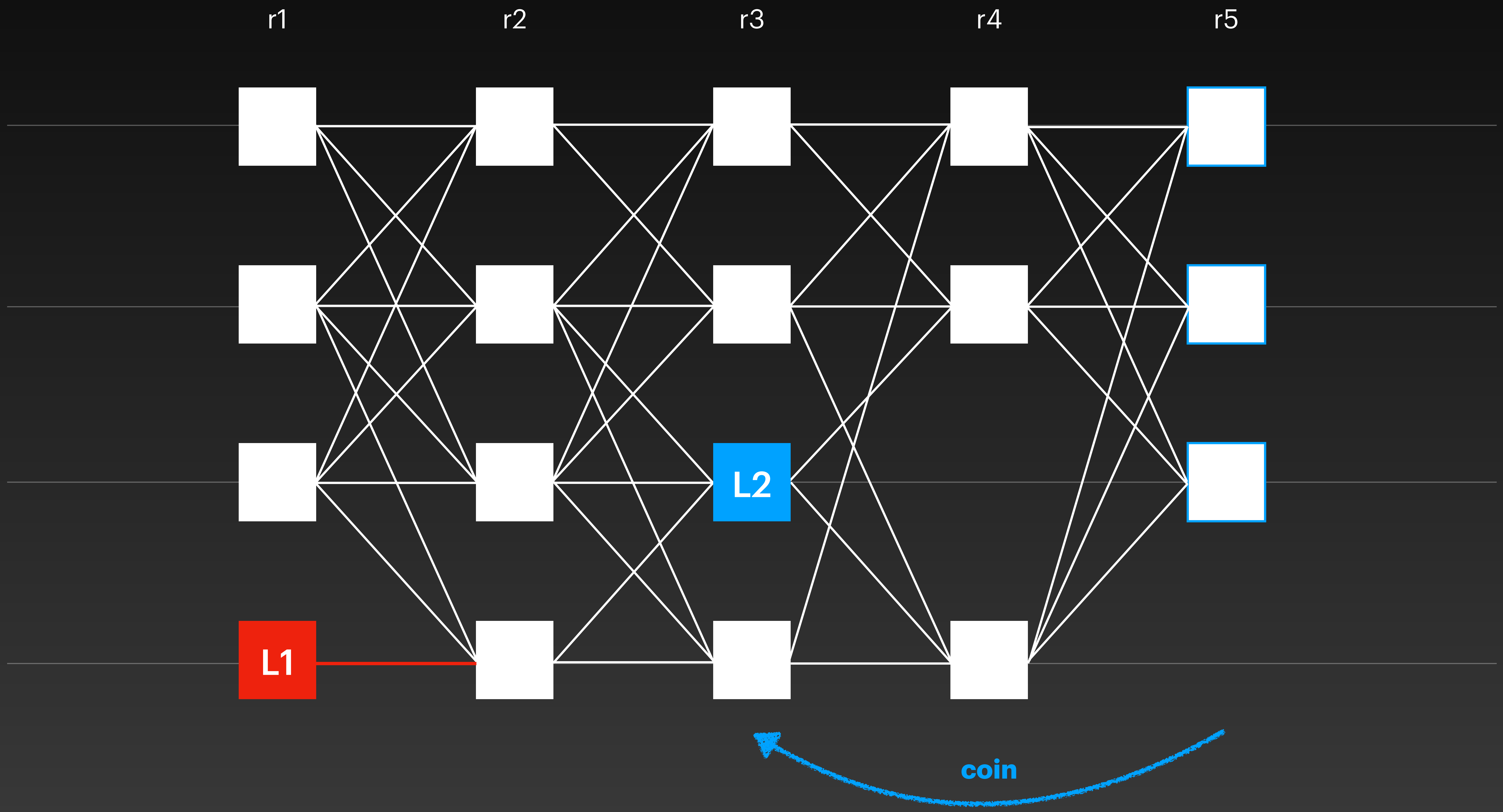
Tusk

Nothing is committed and we keep build the DAG



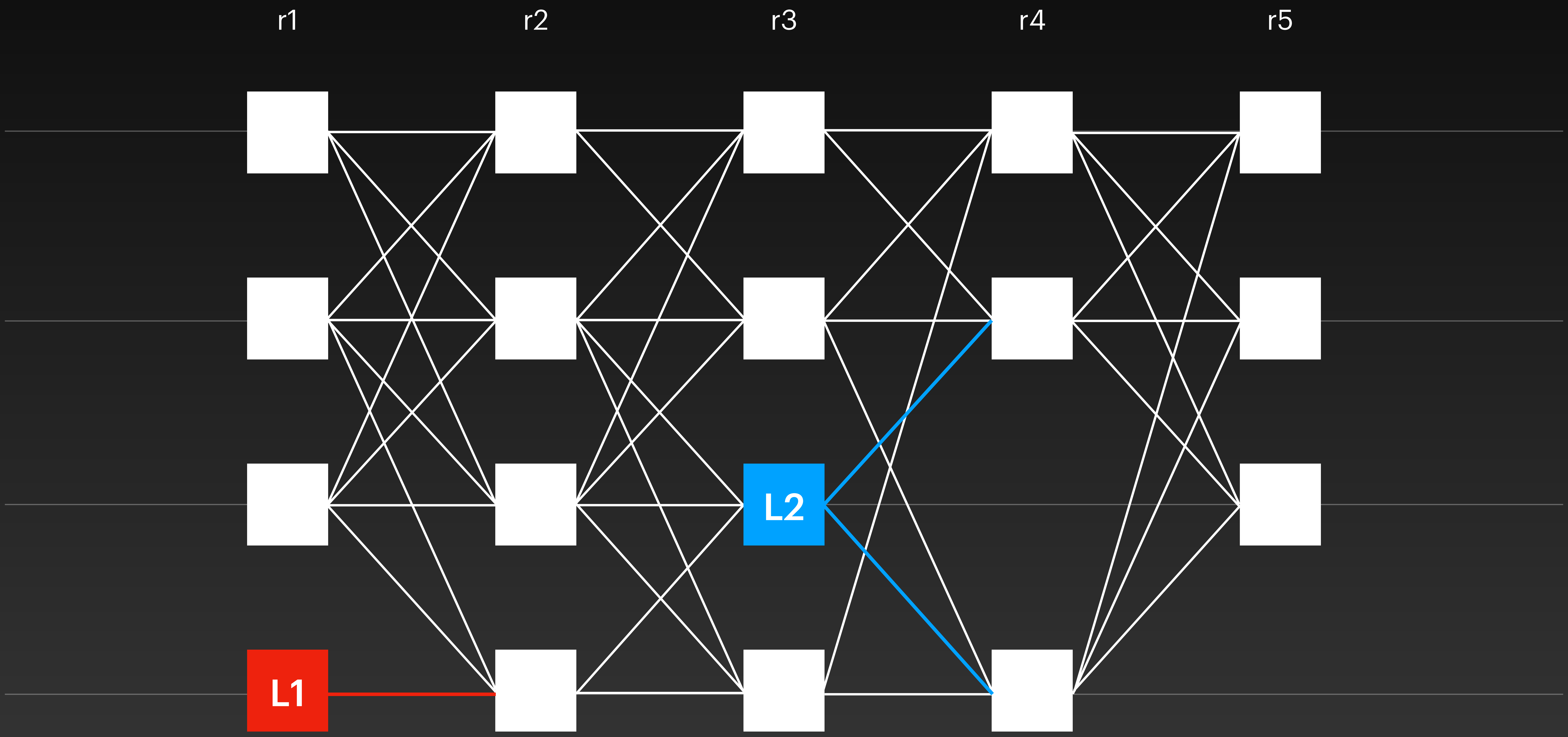
Tusk

Elect the leader of r3



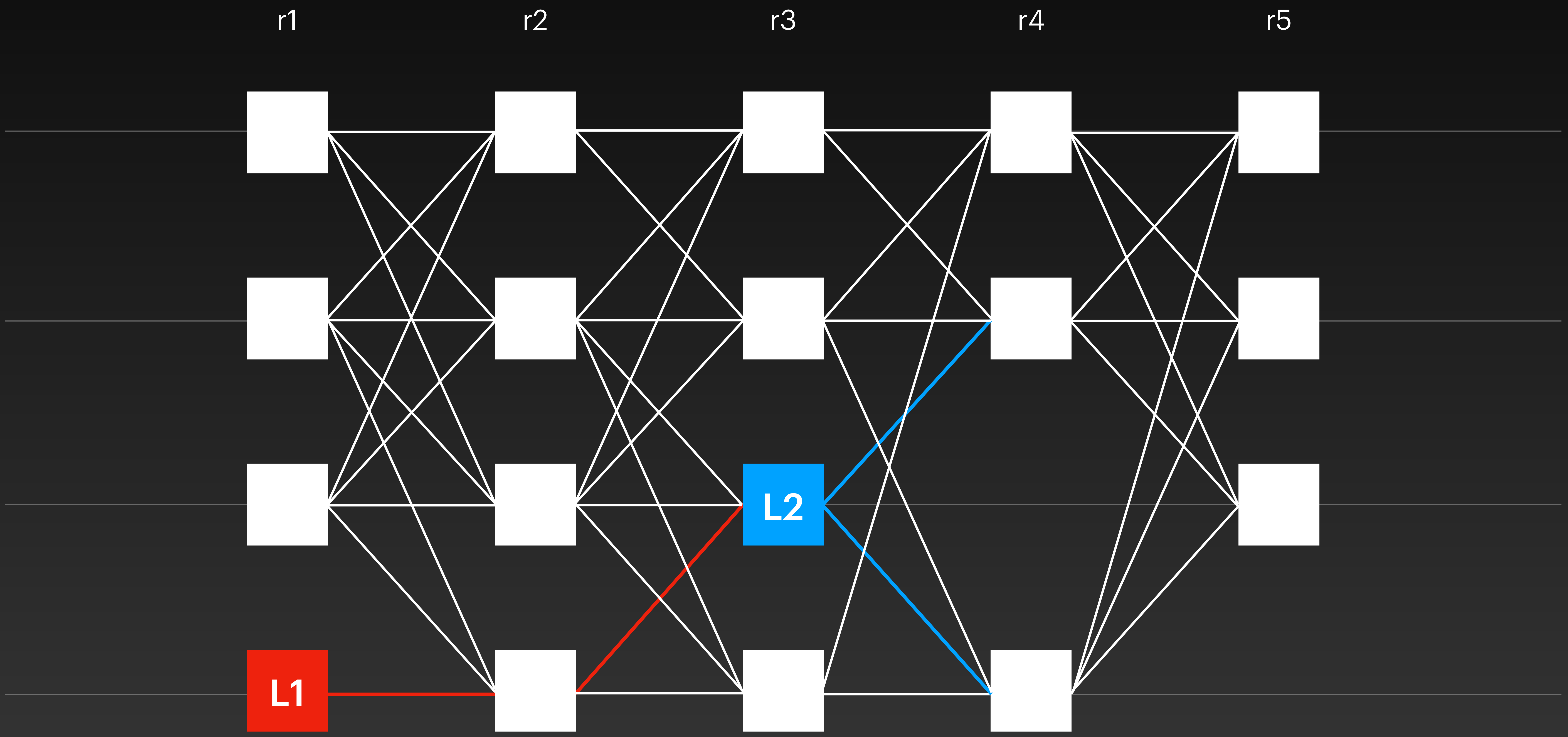
Tusk

Leader L2 has enough support



Tusk

Leader L2 has links to leader L1

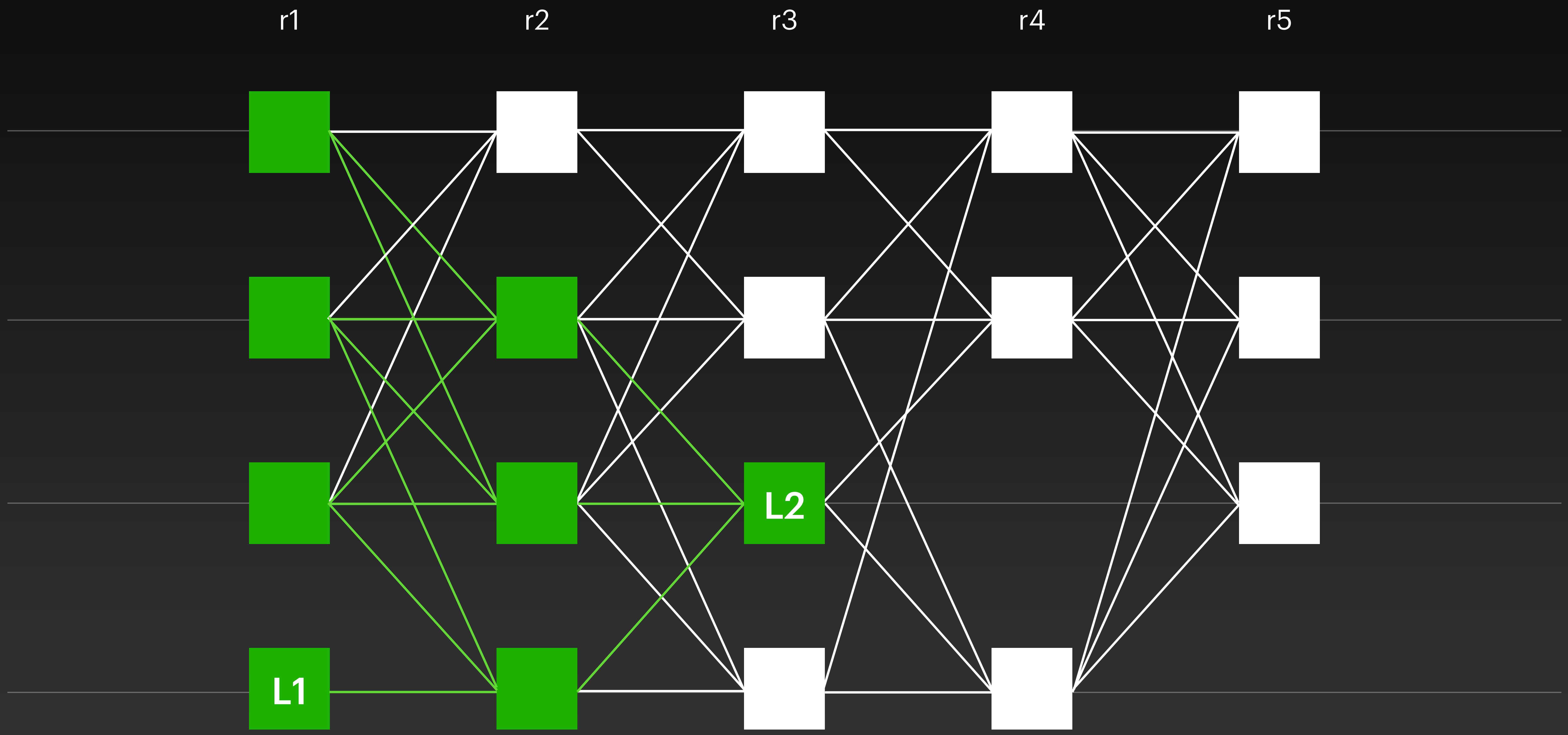


First commit L1

Then commit L2

Tusk

Commit all the sub-DAG of the leader



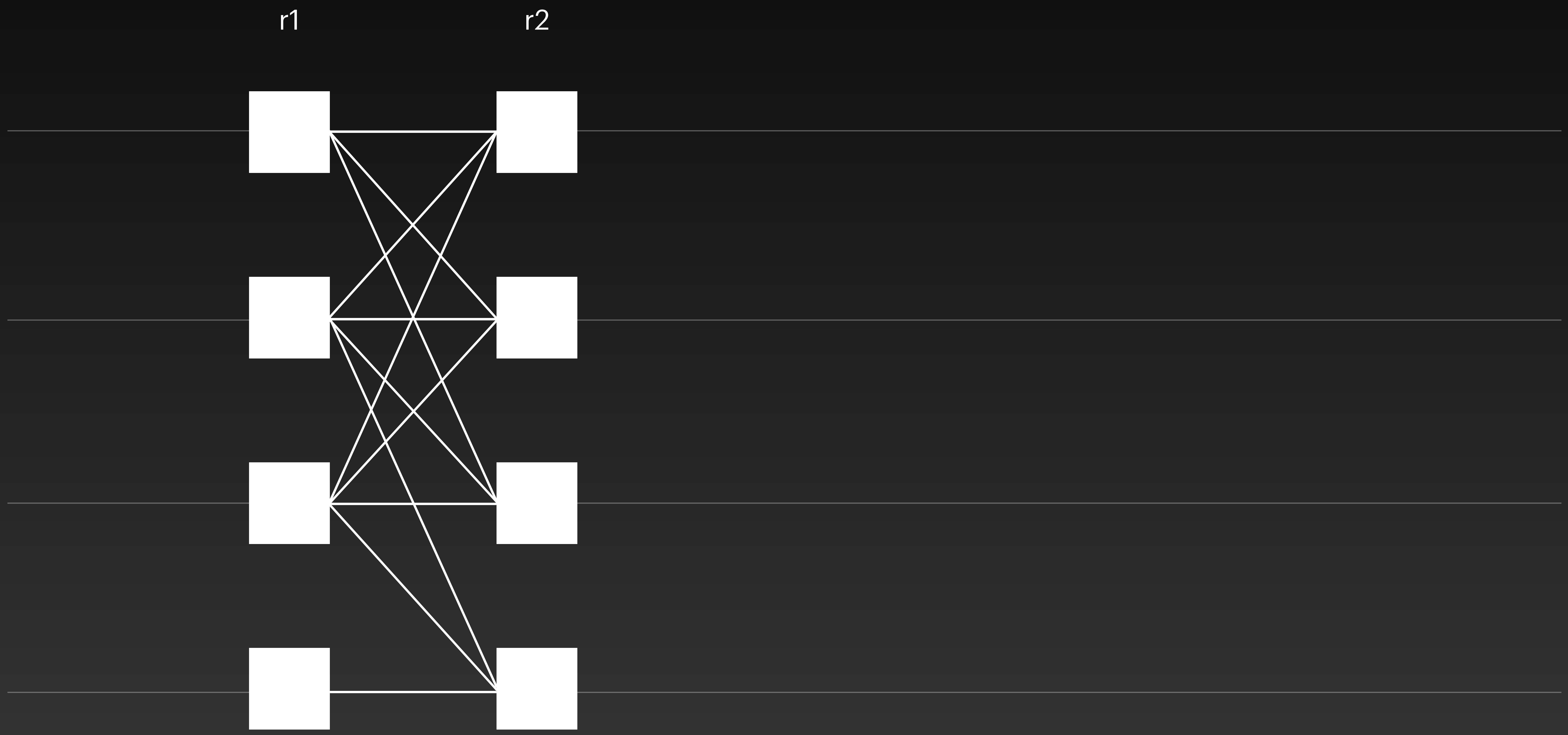
Bullshark

Zero-message partially-synchronous consensus

* without asynchronous fallback

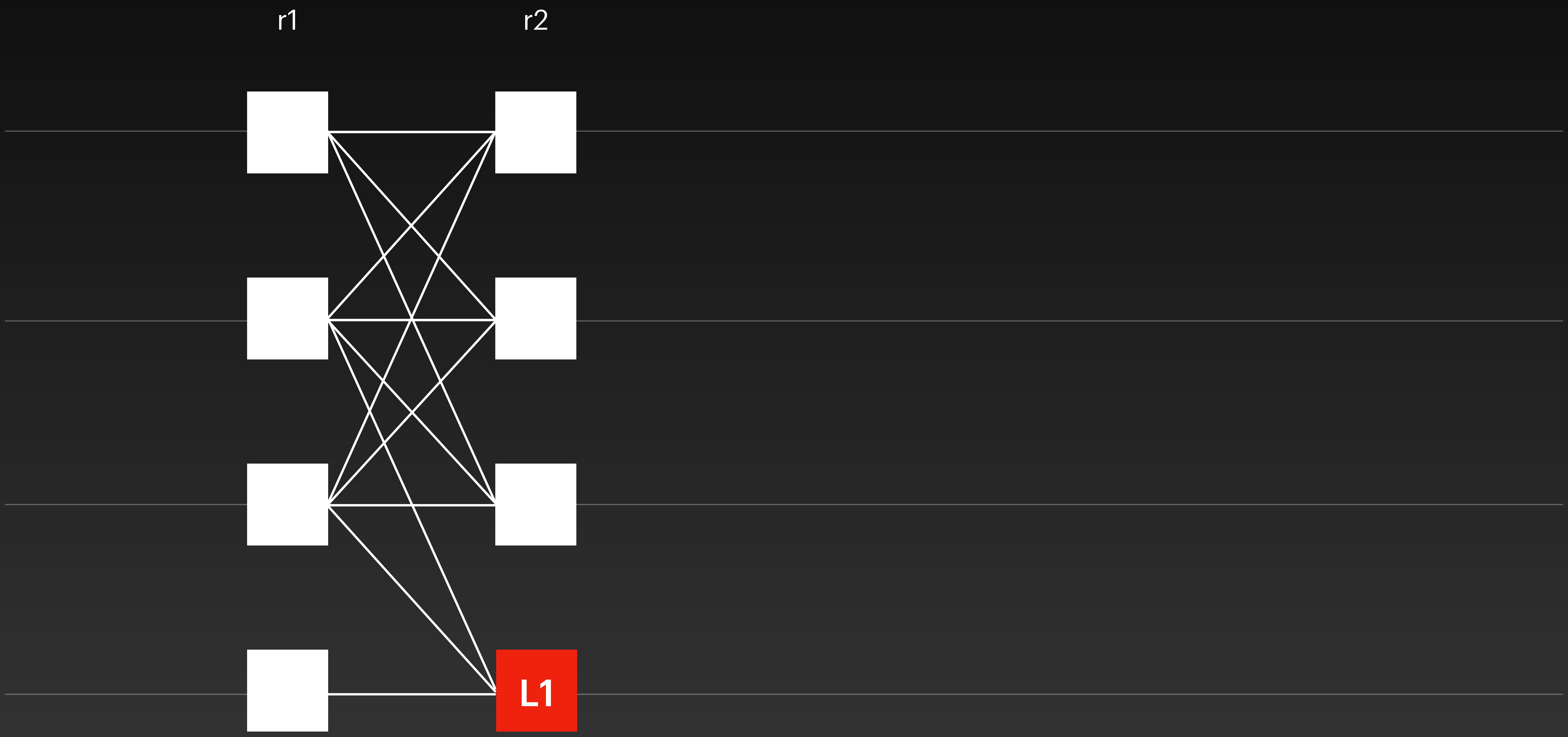
Bullshark

Just interpret the DAG



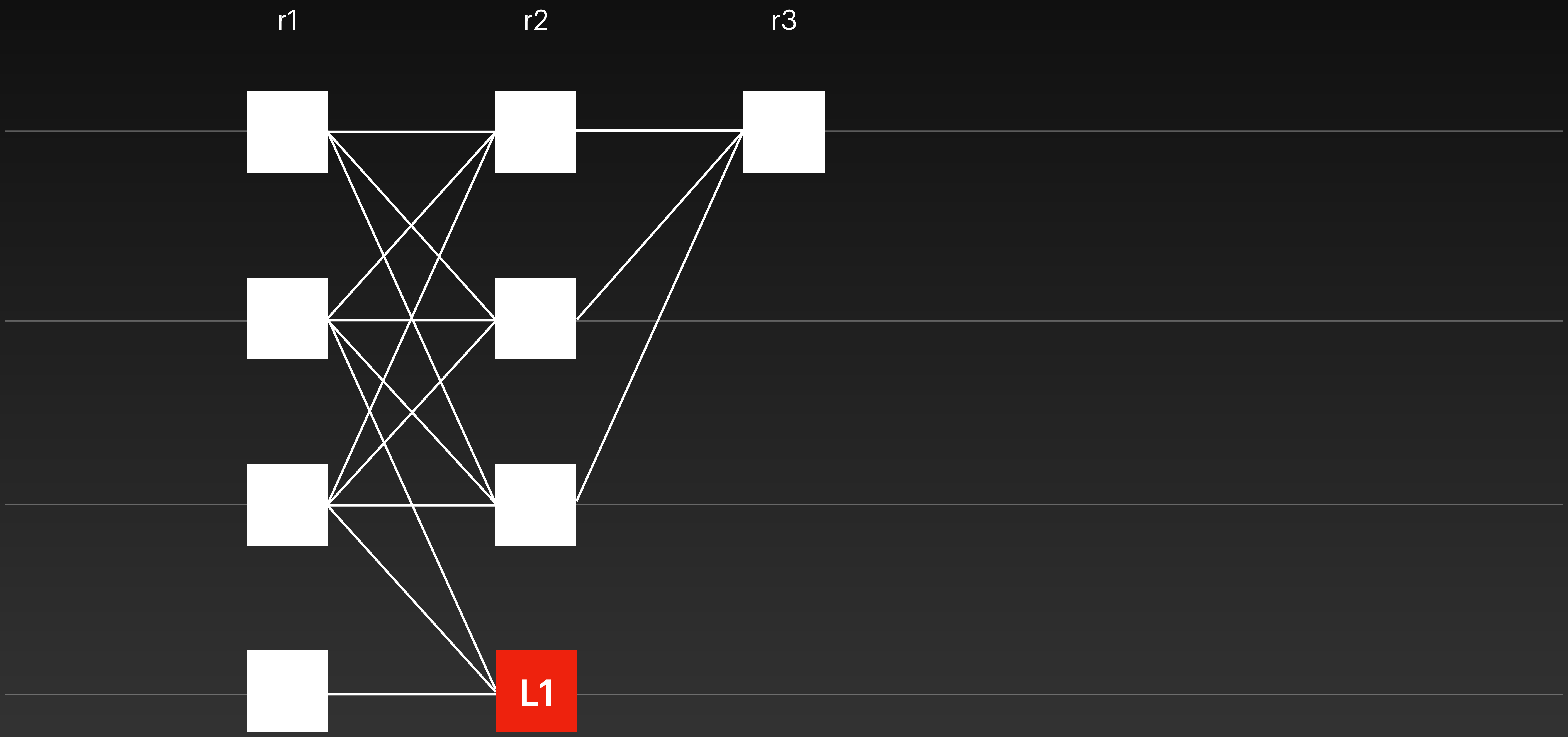
Bullshark

Deterministic leader every 2 rounds



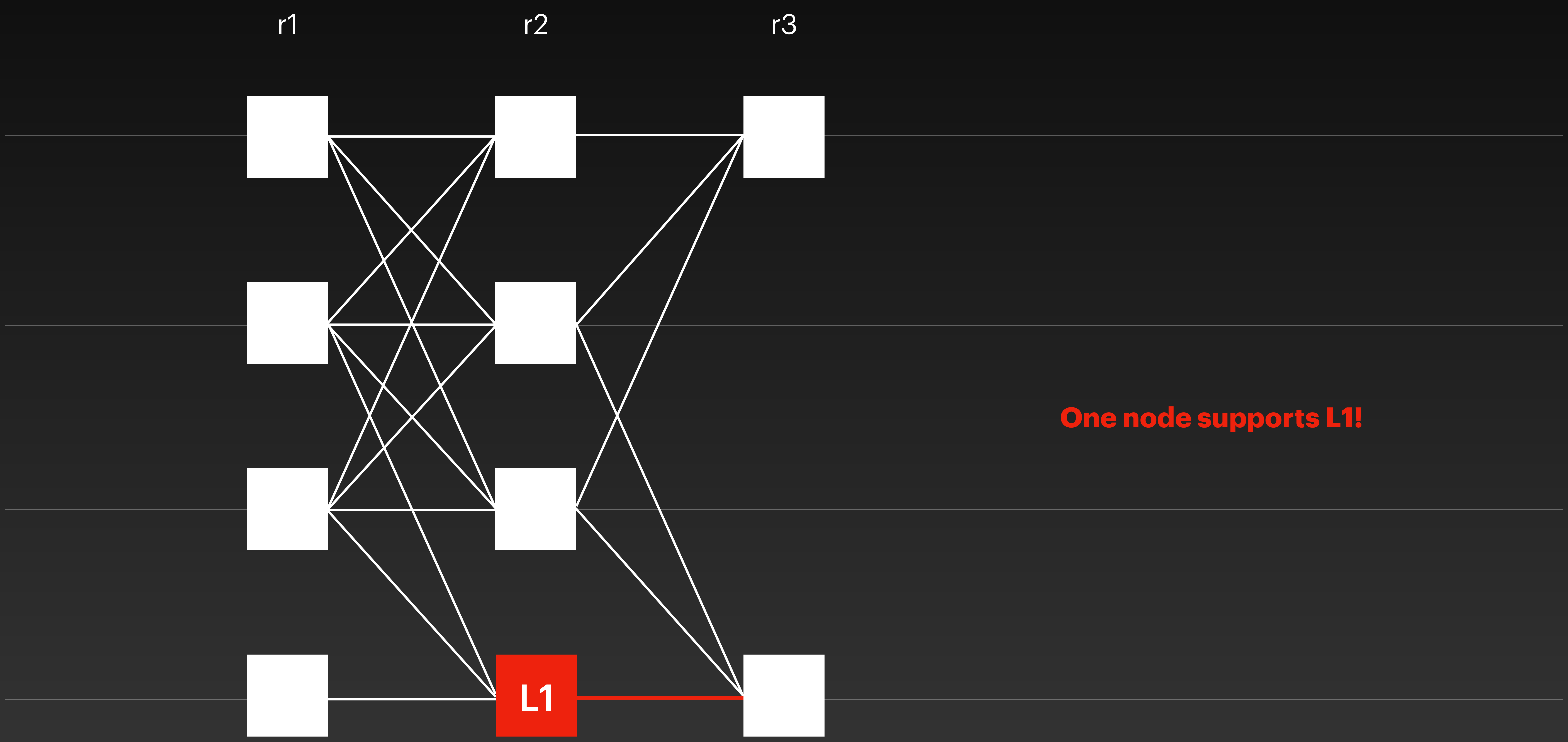
Bullshark

The leader needs f+1 links from round r



Bullshark

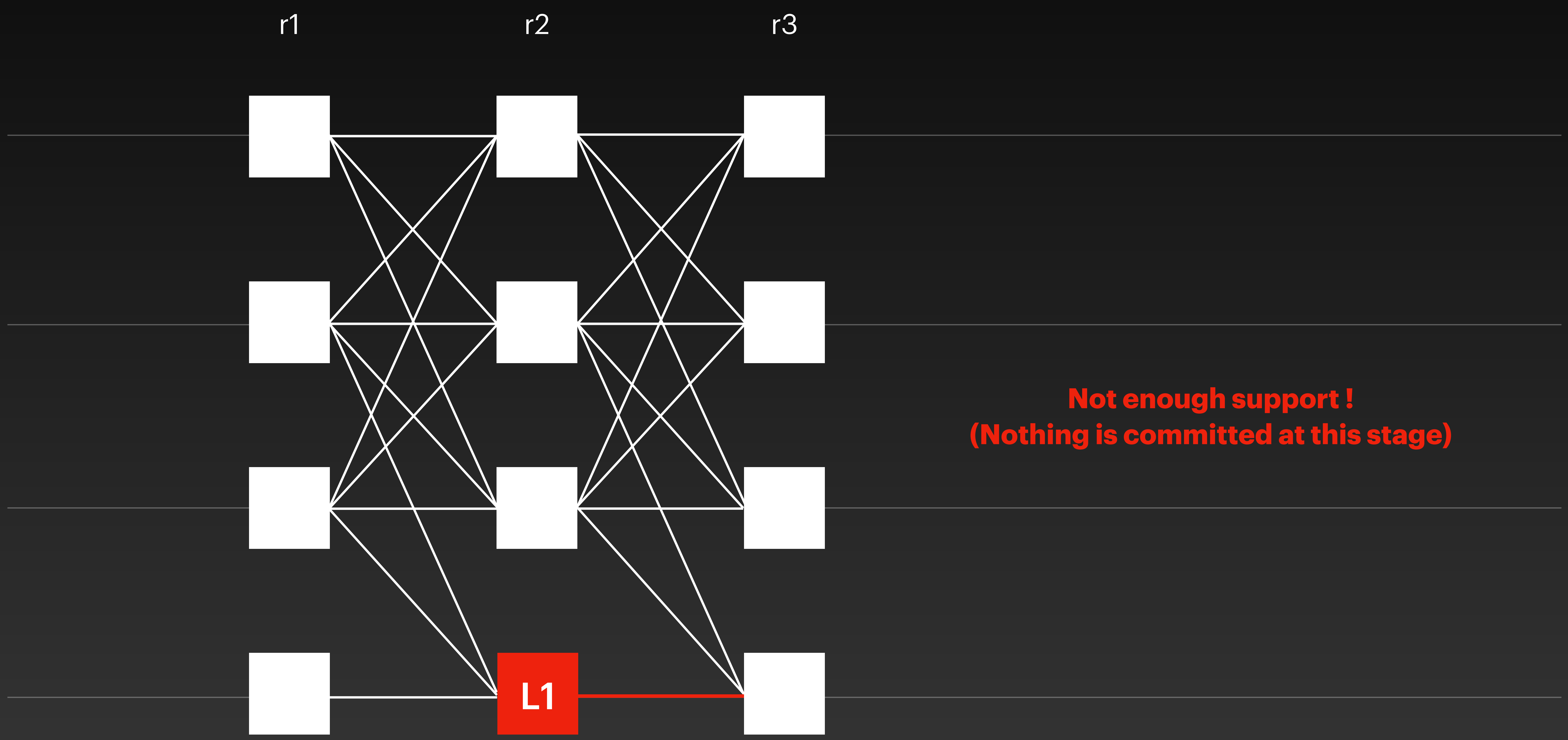
The leader needs f+1 links from round r



One node supports L1!

Bullshark

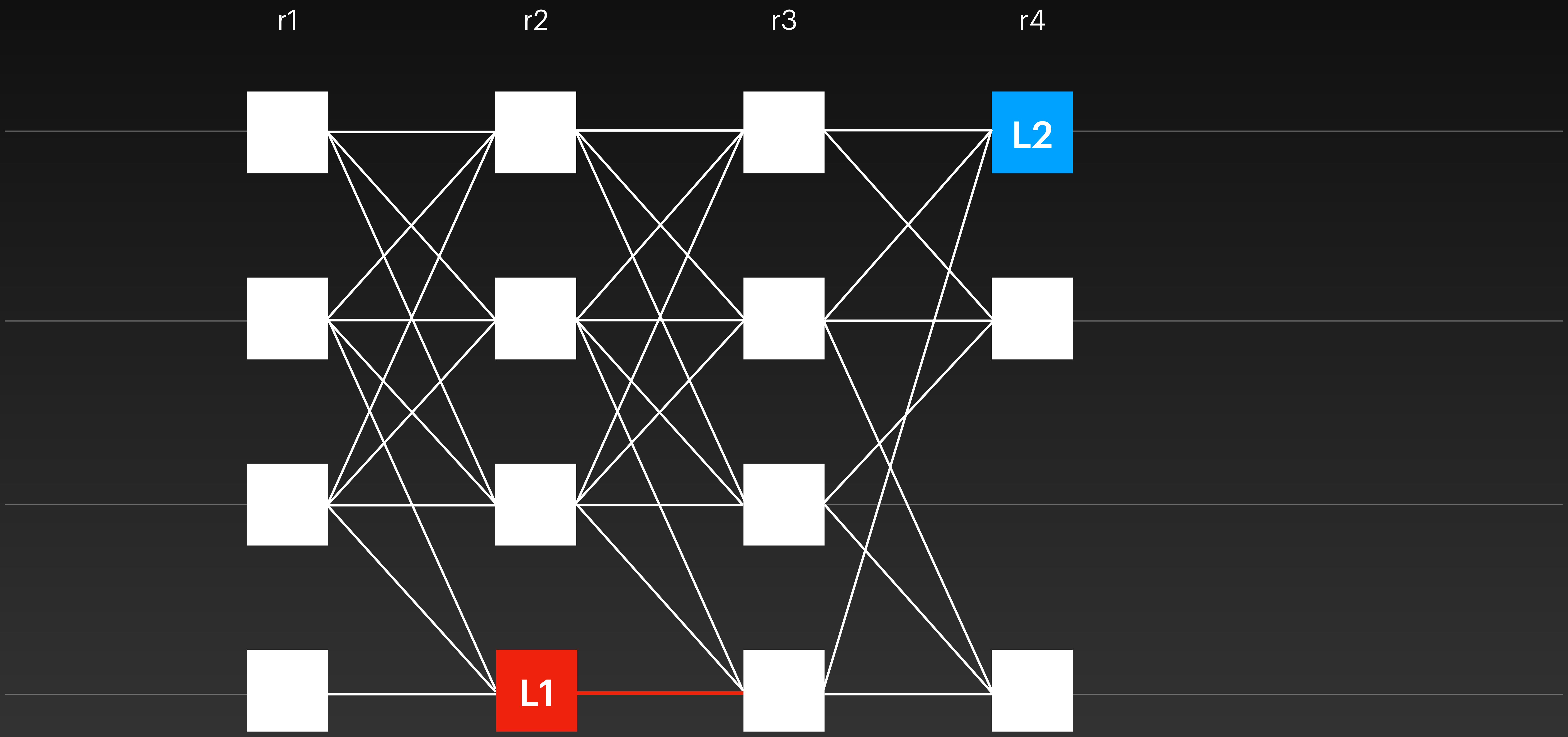
The leader needs f+1 links from round r



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

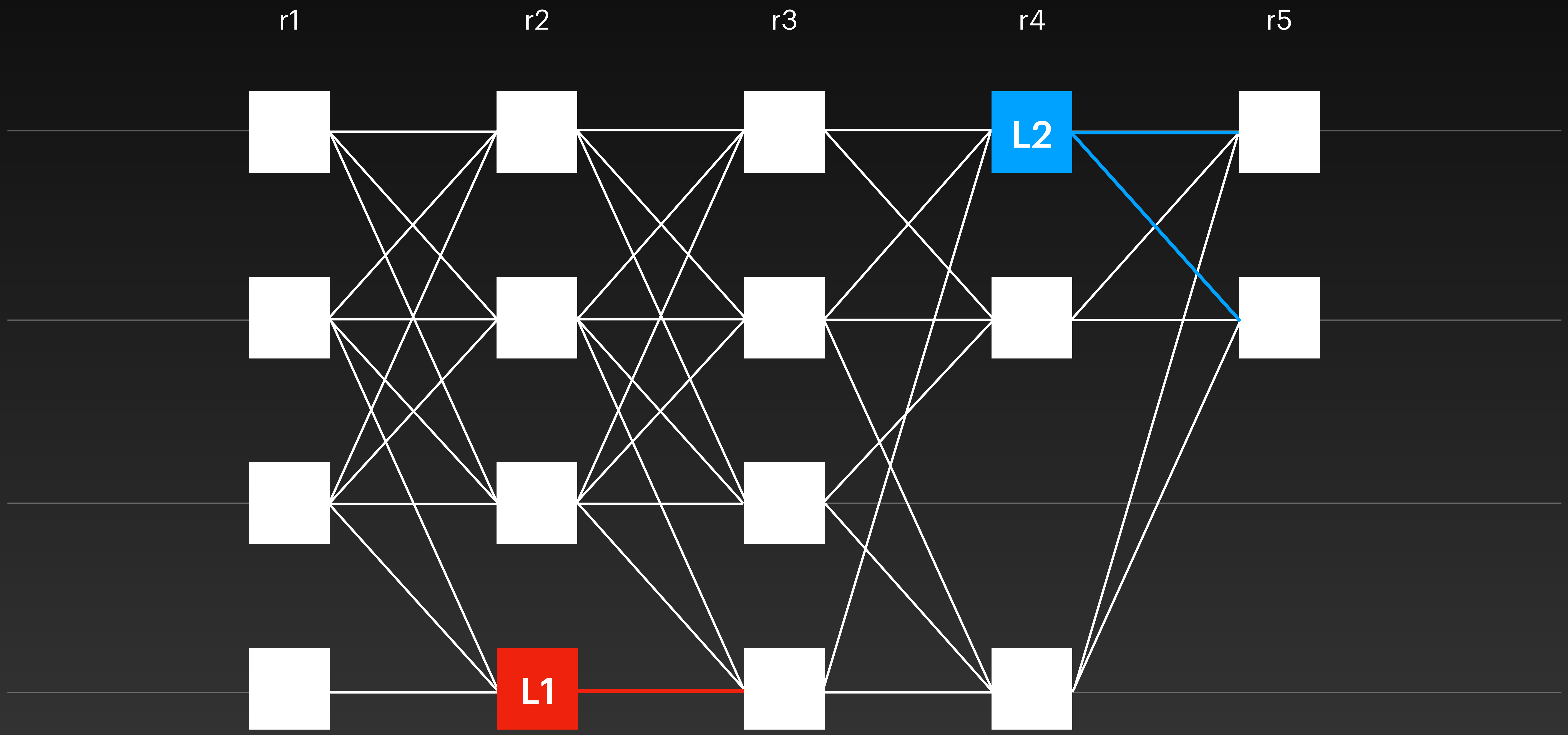
Bullshark

Elect the leader of r4



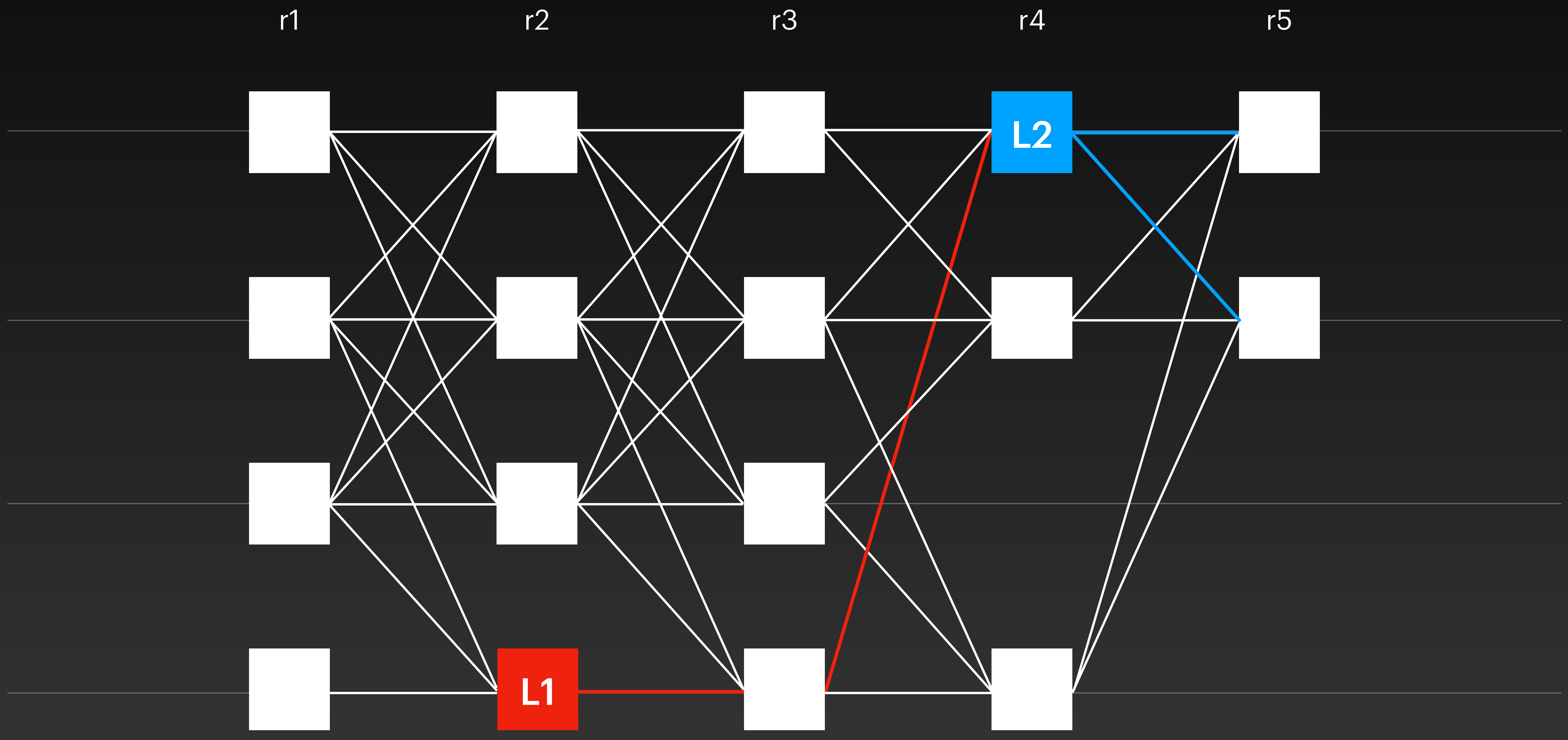
Bullshark

Leader L2 has enough support



Bullshark

Leader L2 has links to leader L1

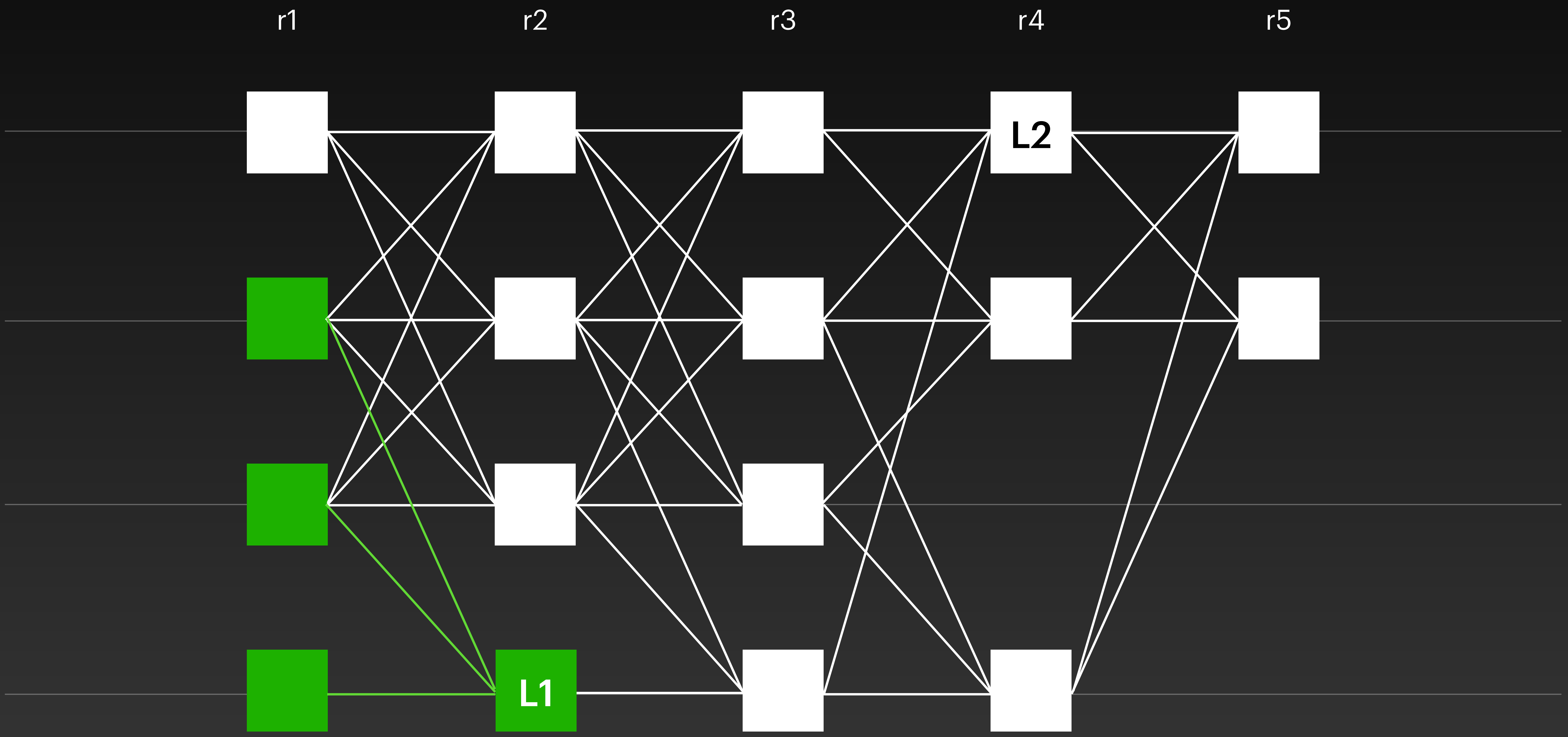


First commit L1

Then commit L2

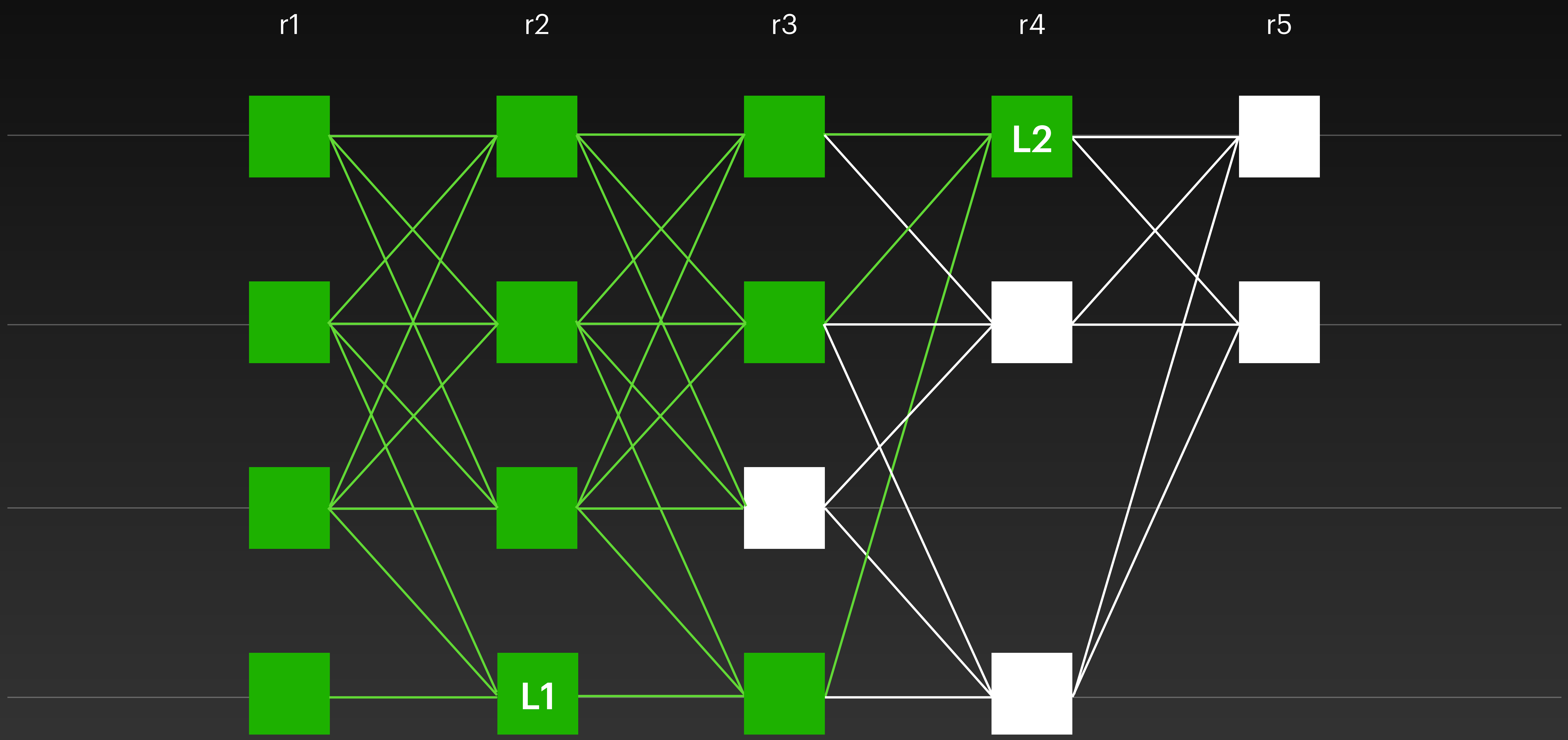
Bullshark

Commit all the sub-DAG of the leader



Bullshark

Commit all the sub-DAG of the leader



Evaluation

How to properly benchmark consensus protocols

Evaluation

Typical mistakes

- 😫 Forgo persistent storage
- 😫 Do not sanitise messages
- 😫 Local/LAN benchmark + ping
- 😫 Many nodes on same machine
- 😫 Change parameters across runs
- 😫 Set transaction size to zero
- 😫 Preconfigure nodes with txs
- 😫 Send a single burst of transactions
- 😫 Benchmark for a few seconds
- 😫 Start timer in the batch maker
- 😫 Evaluate latency w/ only the first tx
- 😫 Separate latency and throughput
- 😫 Only benchmark happy path

Evaluation

Set the benchmark parameters

```
Faults: 0 node(s)

Committee size: 10 node(s)

Transaction size: 512 B
```

Evaluation

Set the benchmark parameters

```
Faults: 0 node(s)
Committee size: 10 node(s)
Transaction size: 512 B
```

```
Header size: 1,000 B
Max header delay: 200 ms
GC depth: 50 round(s)
Sync retry delay: 5,000 ms
Sync retry nodes: 3 node(s)
batch size: 500,000 B
Max batch delay: 200 ms
```

Evaluation

Typical mistakes

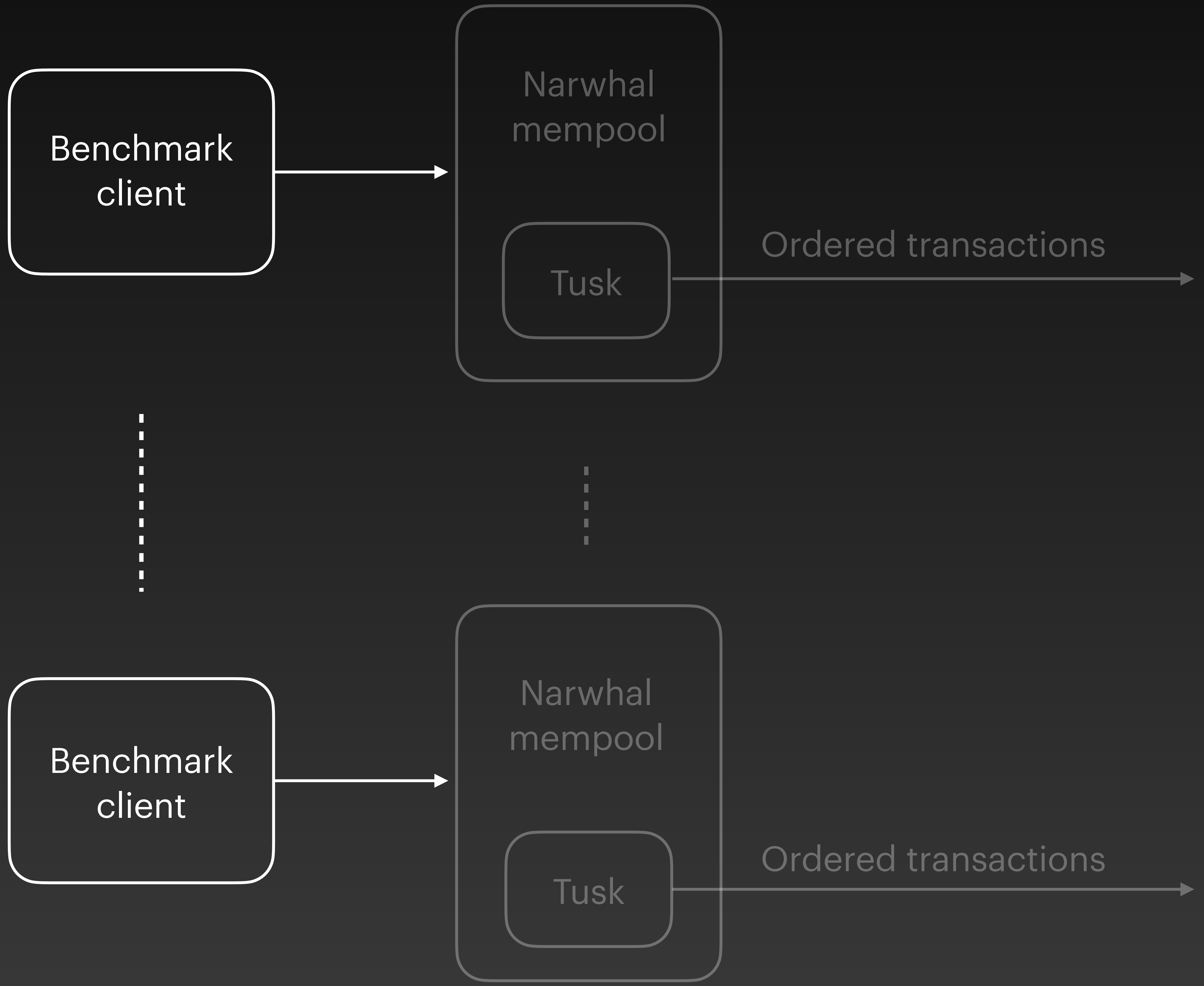
- 😫 Forgo persistent storage
- 😫 Do not sanitise messages
- 😫 Local/LAN benchmark + ping
- 😫 Many nodes on same machine
- 😫 Change parameters across runs
- 😫 Set transaction size to zero
- 😫 Preconfigure nodes with txs
- 😫 Send a single burst of transactions
- 😫 Benchmark for a few seconds
- 😫 Start timer in the batch maker
- 😫 Evaluate latency w/ only the first tx
- 😫 Separate latency and throughput
- 😫 Only benchmark happy path

Evaluation

Benchmark clients

Fixed input rate

For a long time (minutes)



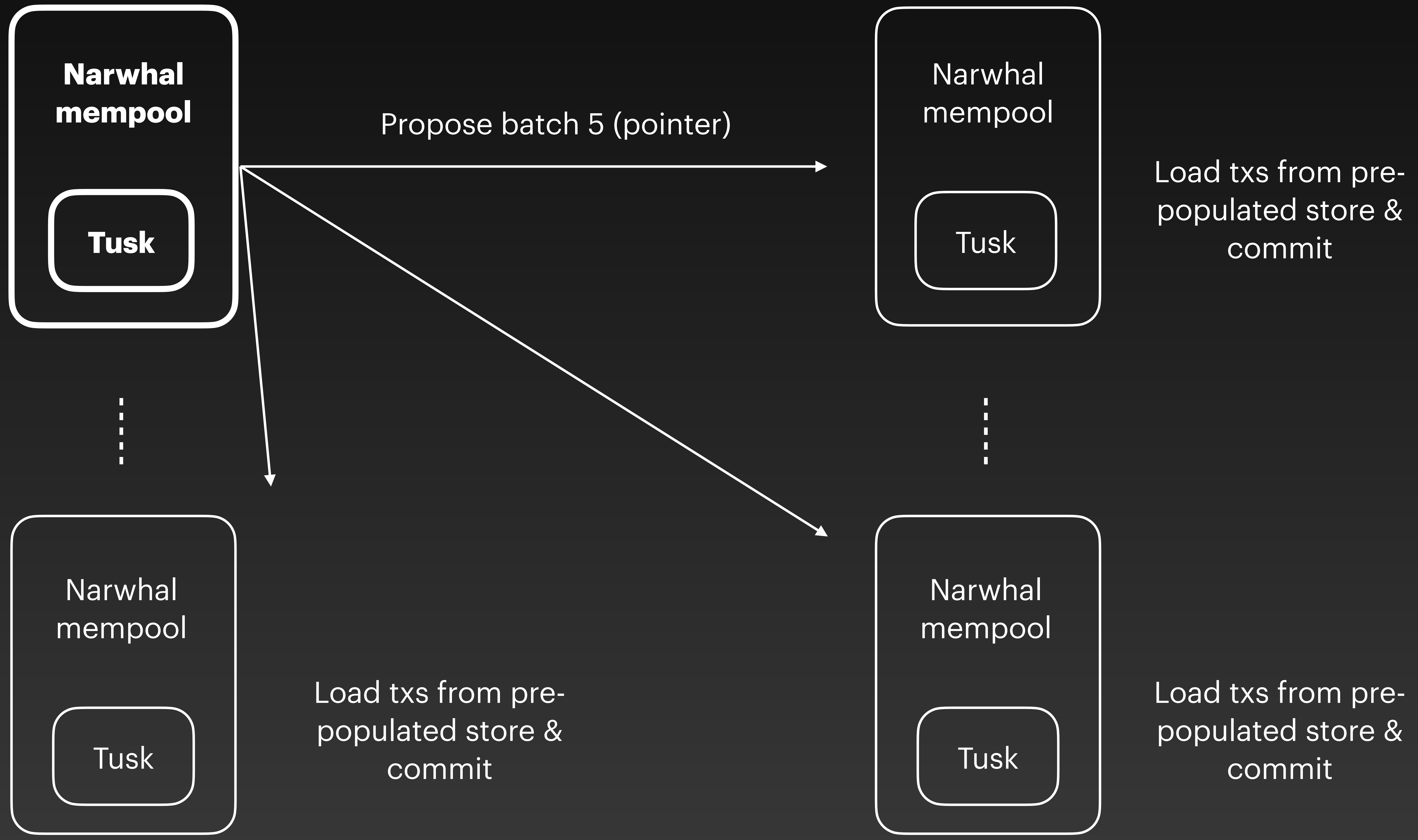
Evaluation

Typical mistakes

- 😫 Forgo persistent storage
- 😫 Do not sanitise messages
- 😫 Local/LAN benchmark + ping
- 😫 Many nodes on same machine
- 😫 Change parameters across runs
- 😫 Set transaction size to zero
- 😫 Preconfigure nodes with txs
- 😫 Send a single burst of transactions
- 😫 Benchmark for a few seconds
- 😫 Start timer in the batch maker
- 😫 Evaluate latency w/ only the first tx
- 😫 Separate latency and throughput
- 😫 Only benchmark happy path

Evaluation

Typical mistake



Narwhal mempool
Tusk
Propose batch 5 (pointer)
Narwhal mempool
Tusk
Load txs from pre-populated store & commit
Narwhal mempool
Tusk
Load txs from pre-populated store & commit
Narwhal mempool
Tusk
Load txs from pre-populated store & commit

Evaluation

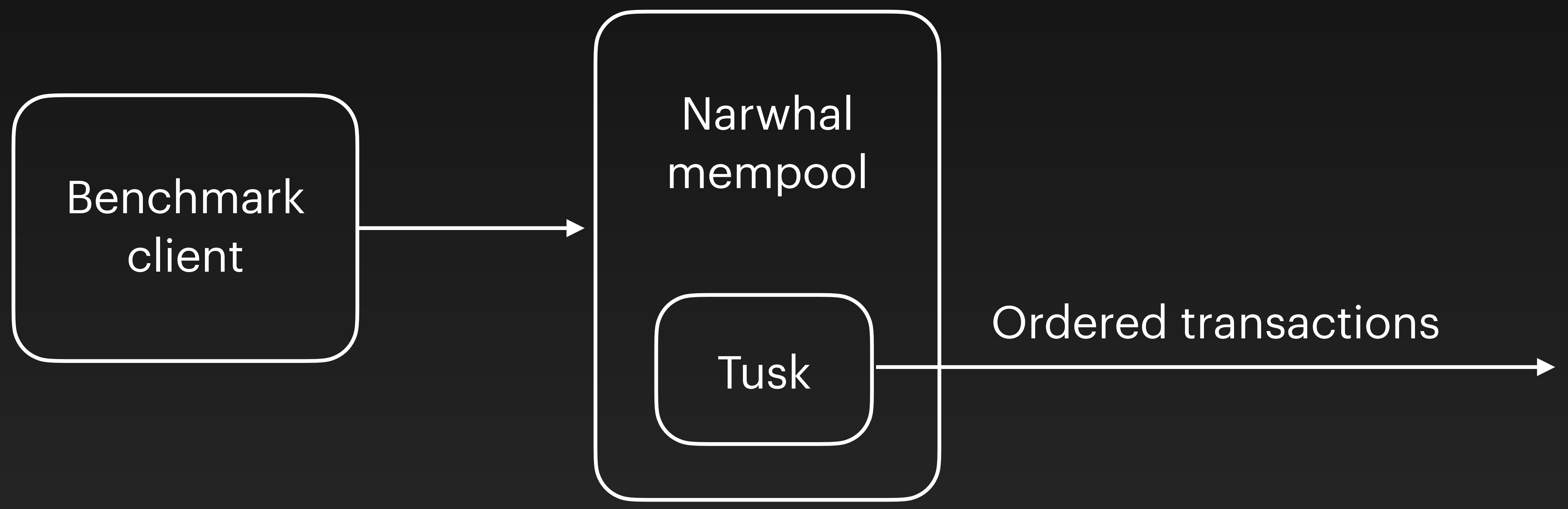
Typical mistakes

- 😫 Forgo persistent storage
- 😫 Do not sanitise messages
- 😫 Local/LAN benchmark + ping
- 😫 Many nodes on same machine
- 😫 Change parameters across runs
- 😫 Set transaction size to zero
- 😫 Preconfigure nodes with txs
- 😫 Send a single burst of transactions
- 😫 Benchmark for a few seconds
- 😫 Start timer in the batch maker
- 😫 Evaluate latency w/ only the first tx
- 😫 Separate latency and throughput
- 😫 Only benchmark happy path

Evaluation

Typical mistake

send 50k txs (once)

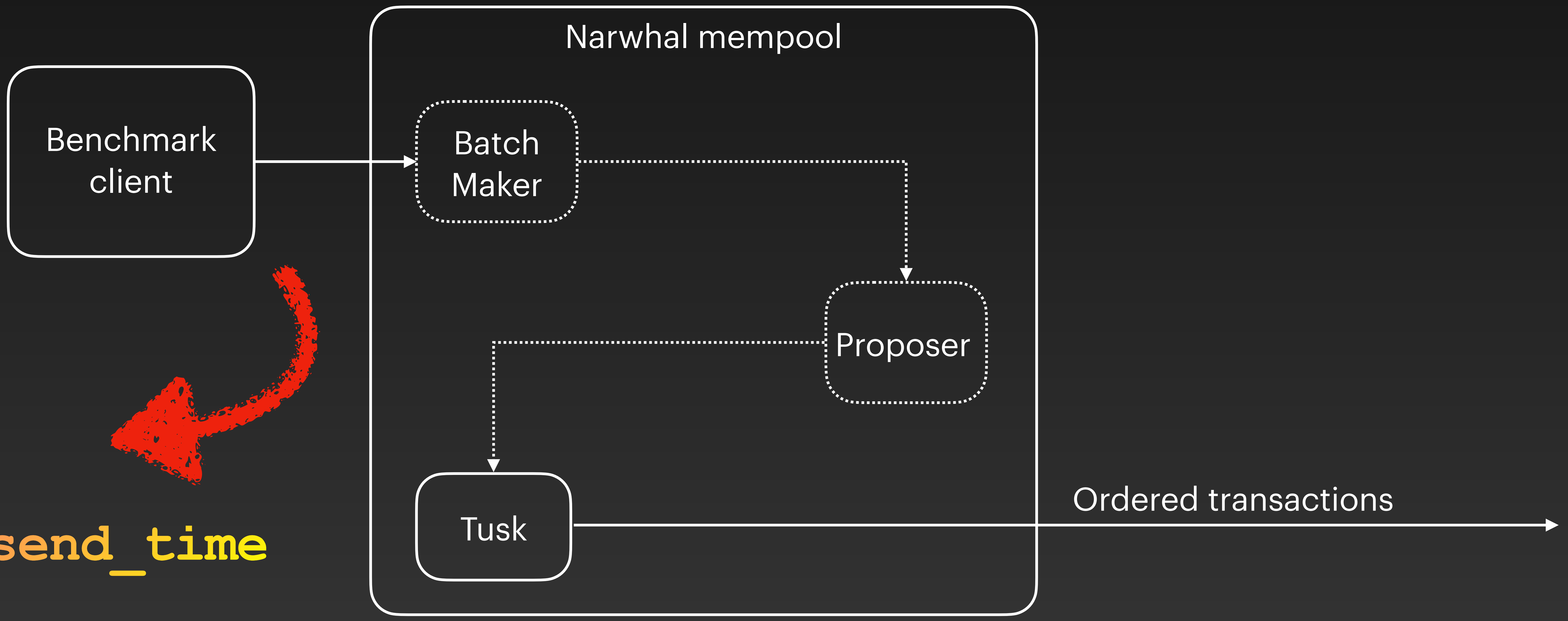


output after 400 ms

😫 TPS = 50k / 400ms = 125k tx/s 😫

Evaluation

Instrument the codebase



`bench_start_time`

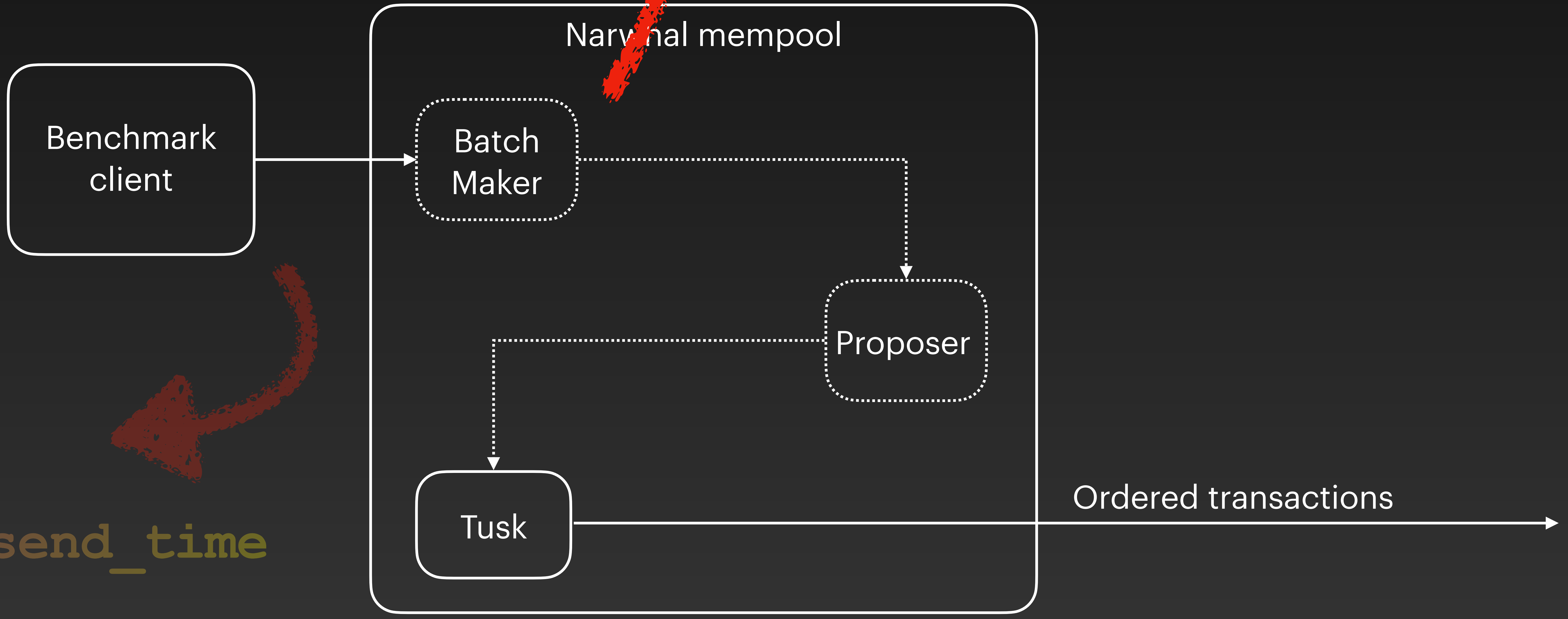
`sample_tx_id -> send_time`

Evaluation

Instrument the codebase

batch_digest -> sample_tx_id

batch_digest -> batch_bytes



bench_start_time

sample_tx_id -> send_time

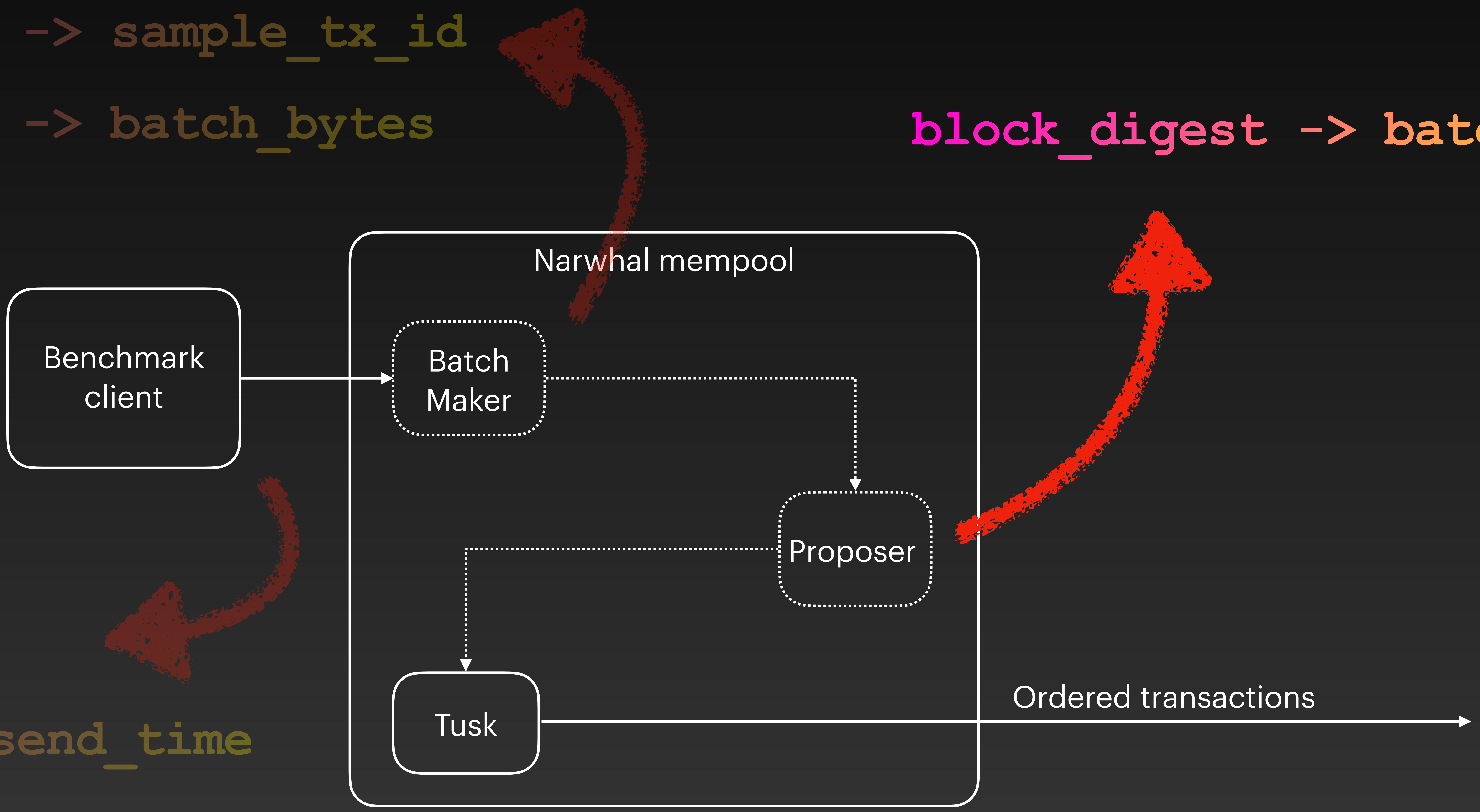
Evaluation

Instrument the codebase

batch_digest -> sample_tx_id

batch_digest -> batch_bytes

block_digest -> batch_digest

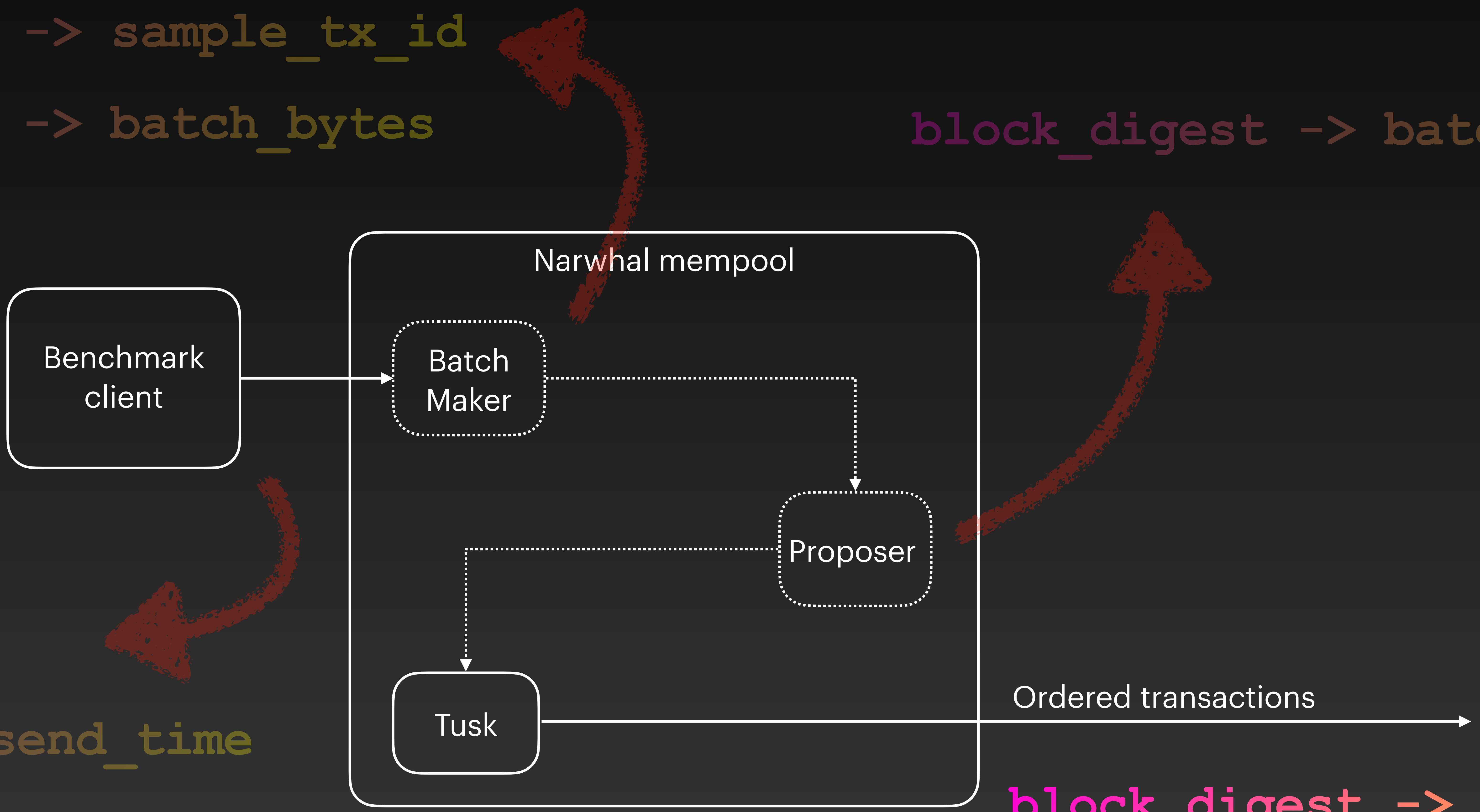


bench_start_time

sample_tx_id -> send_time

Evaluation

Instrument the codebase



```
batch_digest -> sample_tx_id

batch_digest -> batch_bytes
```

```
block_digest -> batch_digest
```

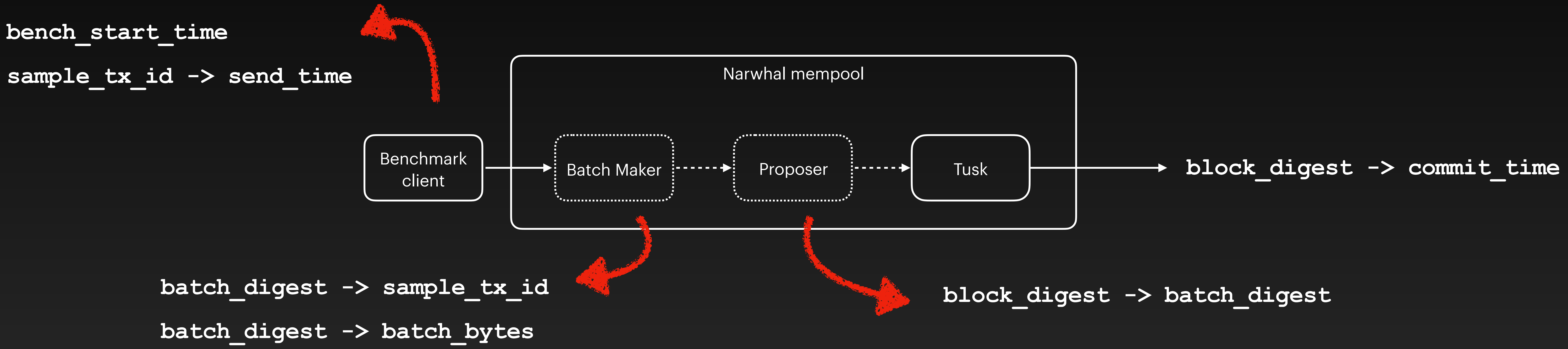
```
bench_start_time

sample_tx_id -> send_time
```

```
block_digest -> commit_time
```

Evaluation

Compute throughput



bench_start_time

sample_tx_id -> send_time

Narwhal mempool

Benchmark client

Batch Maker

Proposer

Tusk

block_digest -> commit_time

batch_digest -> sample_tx_id

batch_digest -> batch_bytes

block_digest -> batch_digest

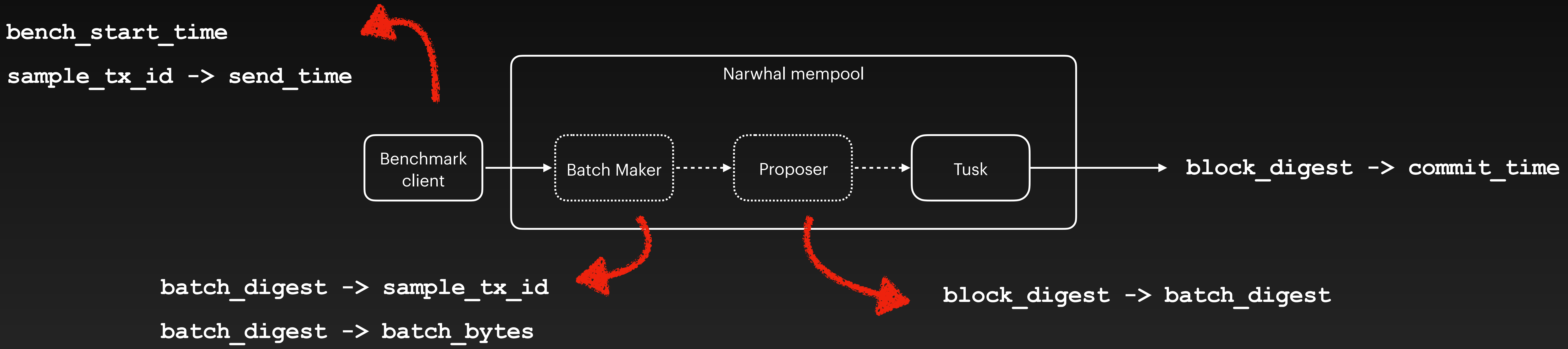
total_time = last_commit_time - bench_start_time

BPS = total_bytes / total_time

TPS = BPS / transaction_size

Evaluation

Compute latency



```
bench_start_time
sample_tx_id -> send_time
```

```
block_digest -> commit_time
```

```
batch_digest -> sample_tx_id
batch_digest -> batch_bytes
```

```
block_digest -> batch_digest
```

```
samples = commit_time - send_time

latency = average(samples)
```

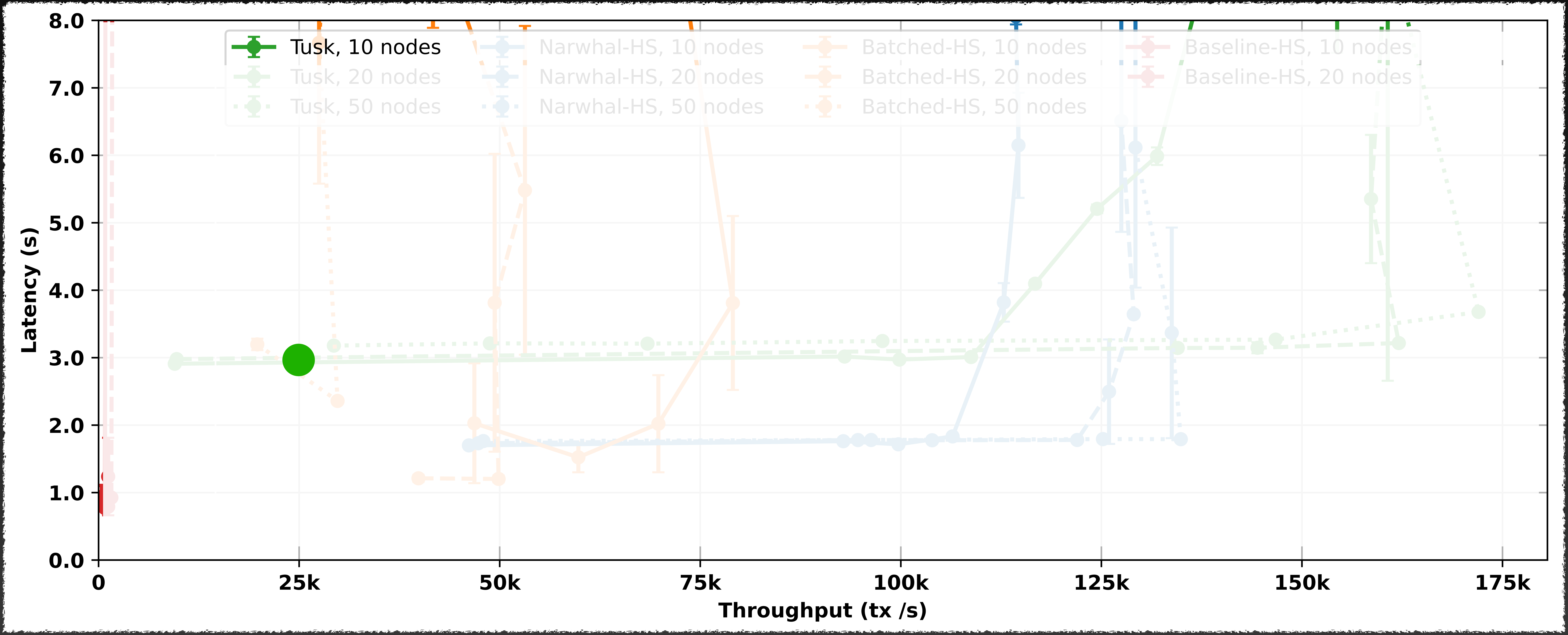
Evaluation

Typical mistakes

- 😫 Forgo persistent storage
- 😫 Do not sanitise messages
- 😫 Local/LAN benchmark + ping
- 😫 Many nodes on same machine
- 😫 Change parameters across runs
- 😫 Set transaction size to zero
- 😫 Preconfigure nodes with txs
- 😫 Send a single burst of transactions
- 😫 Benchmark for a few seconds
- 😫 Start timer in the batch maker
- 😫 Evaluate latency w/ only the first tx
- 😫 Separate latency and throughput
- 😫 Only benchmark happy path

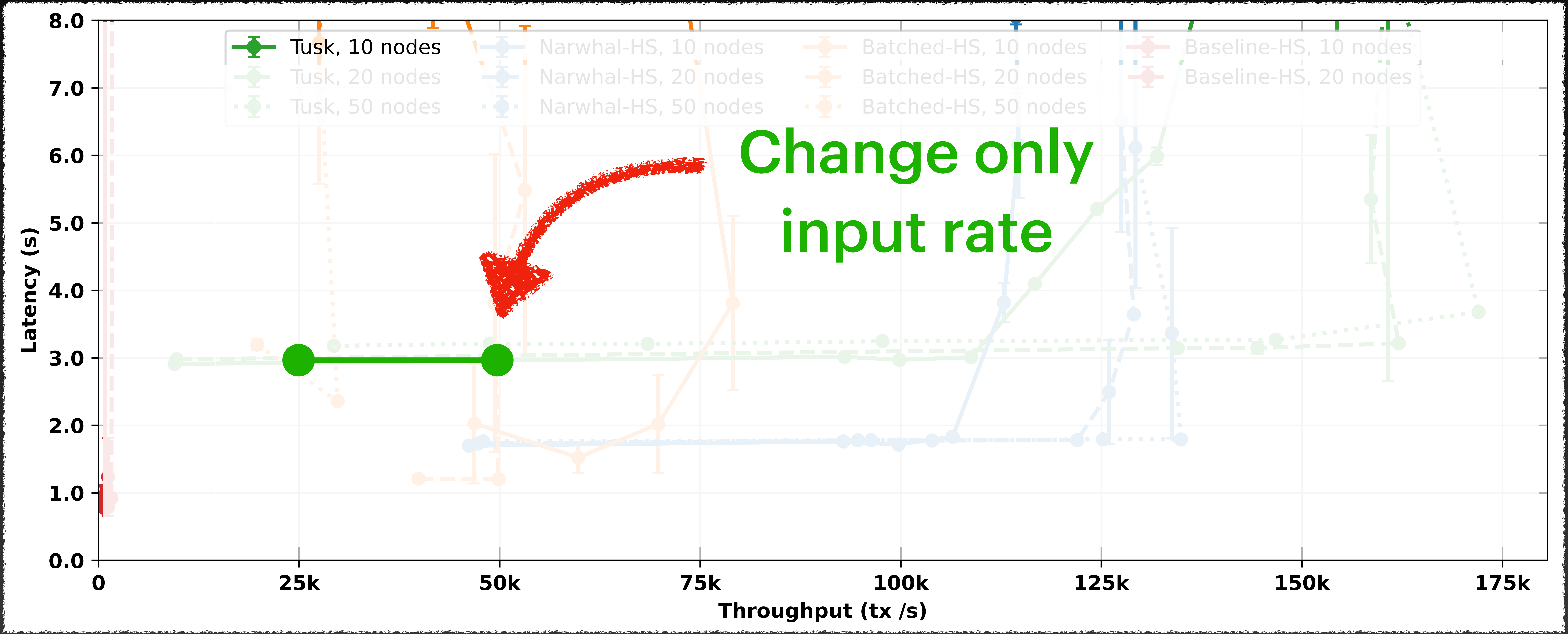
Evaluation

Throughput latency graph



Evaluation

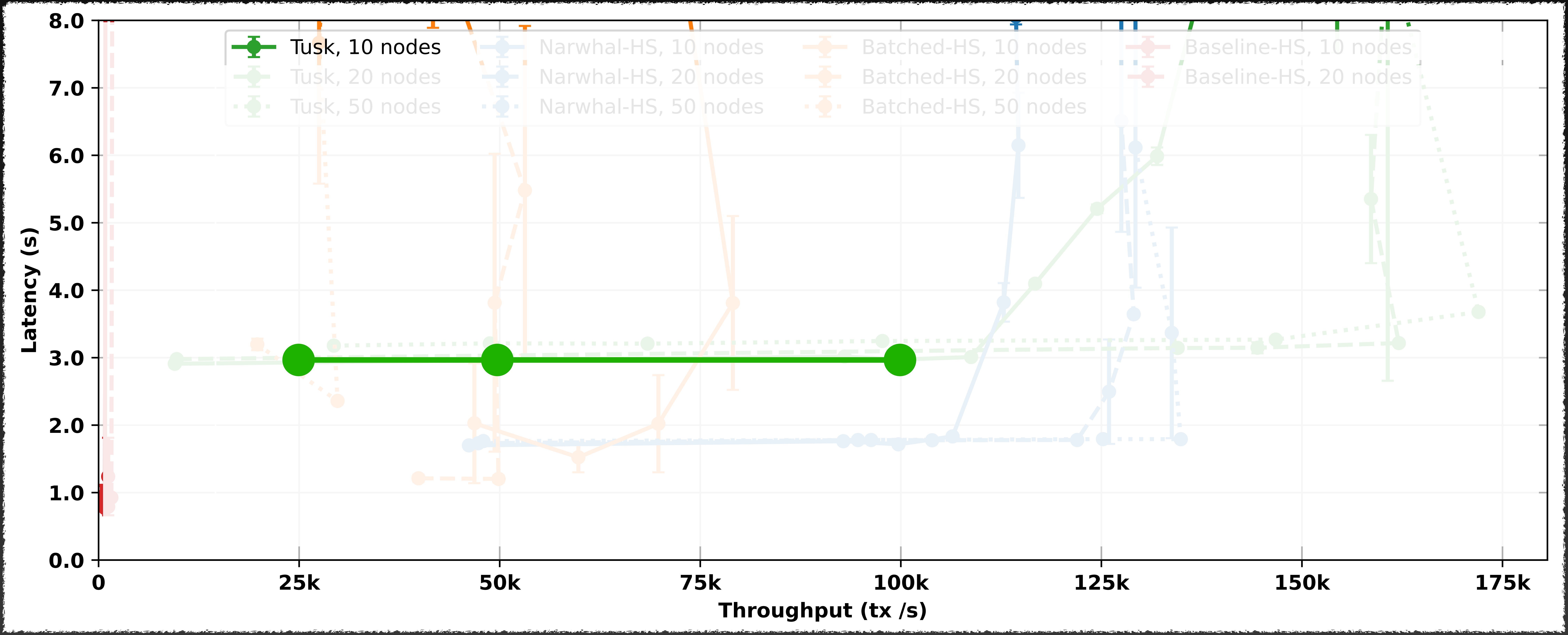
Throughput latency graph



Change only input rate

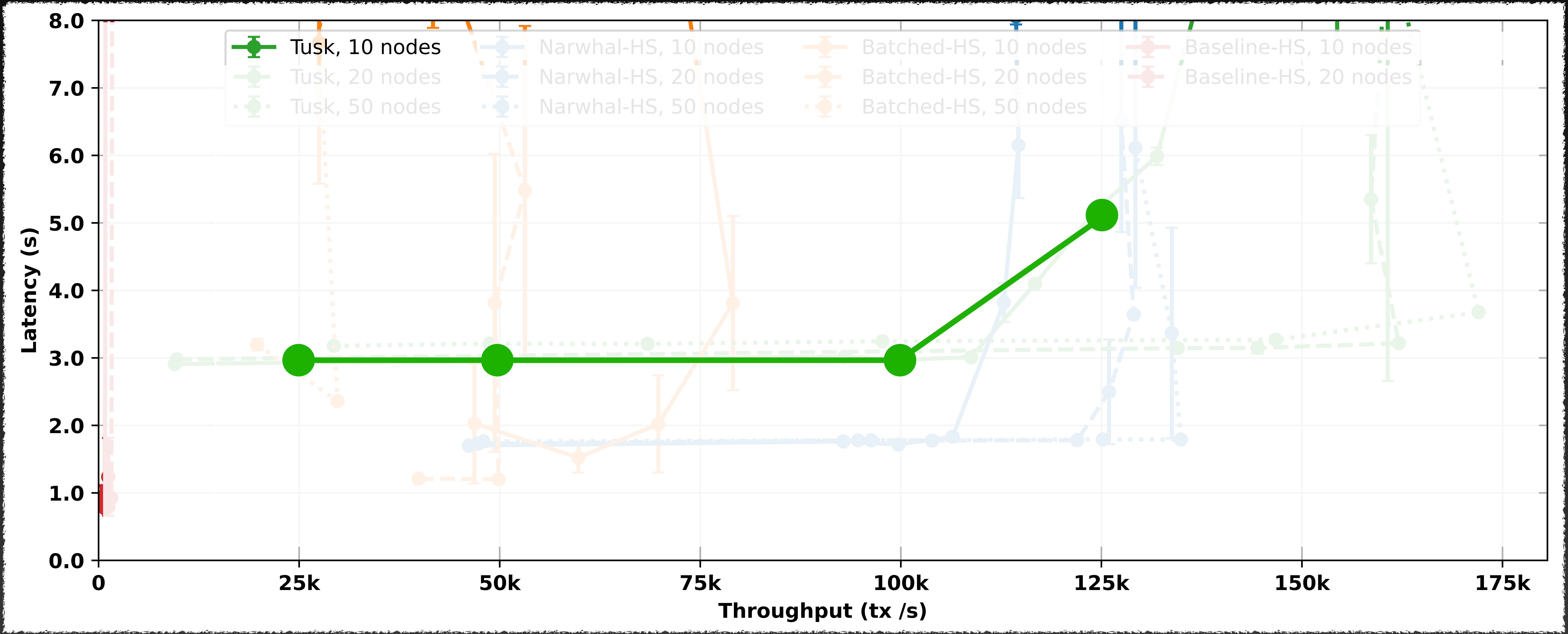
Evaluation

Throughput latency graph



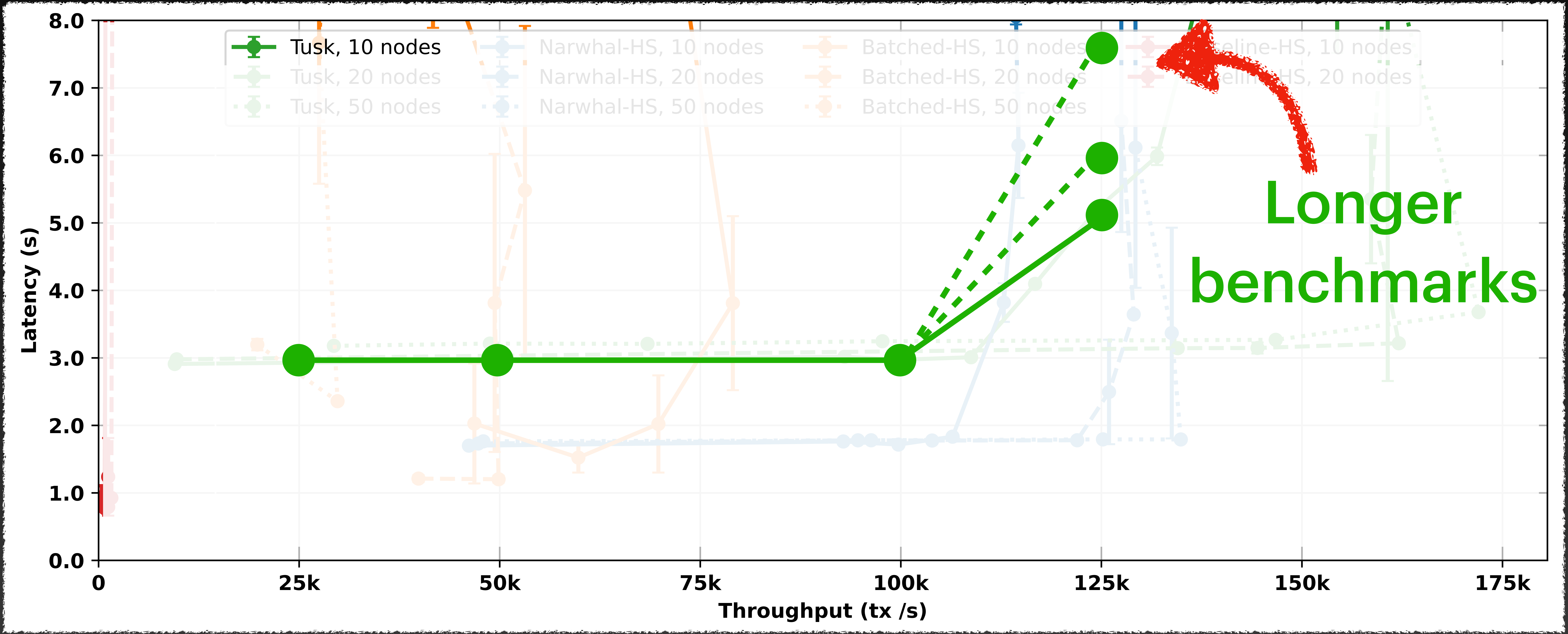
Evaluation

Throughput latency graph



Evaluation

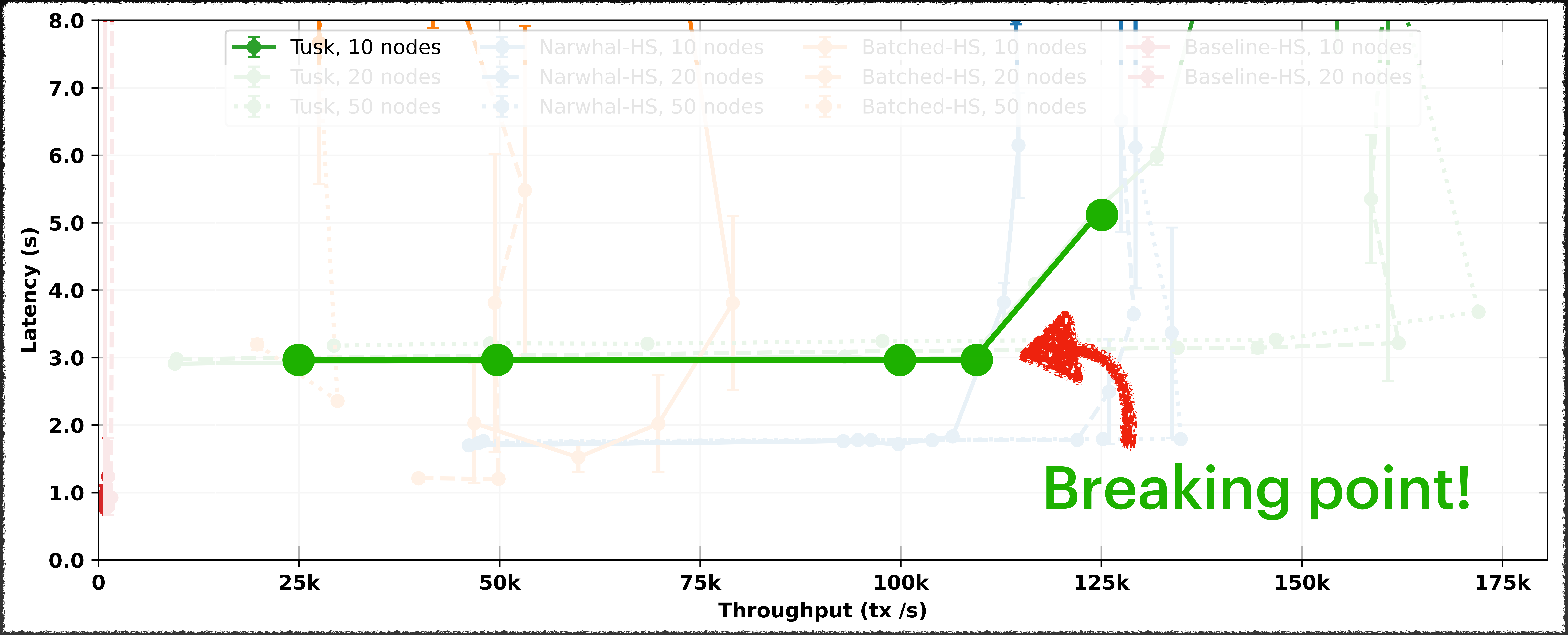
Throughput latency graph



Longer benchmarks

Evaluation

Throughput latency graph



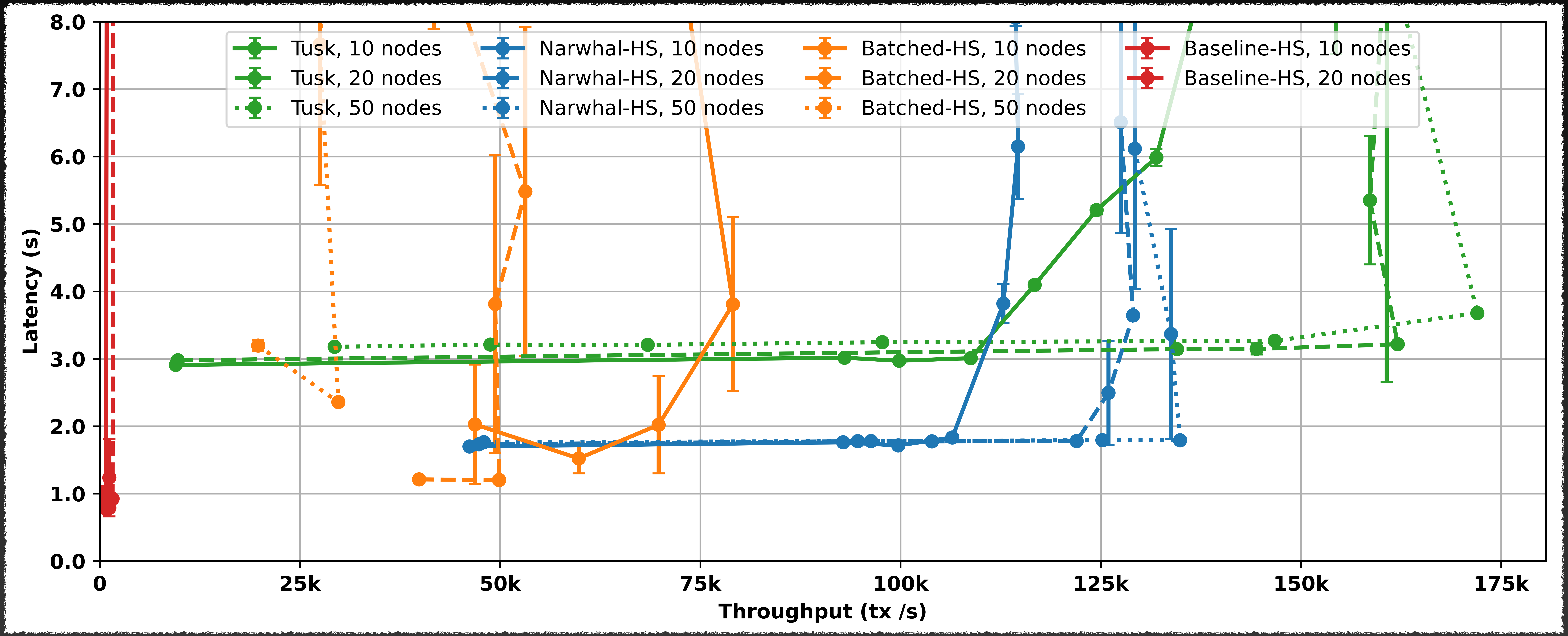
Evaluation

Typical mistakes

- 😫 Forgo persistent storage
- 😫 Do not sanitise messages
- 😫 Local/LAN benchmark + ping
- 😫 Many nodes on same machine
- 😫 Change parameters across runs
- 😫 Set transaction size to zero
- 😫 Preconfigure nodes with txs
- 😫 Send a single burst of transactions
- 😫 Benchmark for a few seconds
- 😫 Start timer in the batch maker
- 😫 Evaluate latency w/ only the first tx
- 😫 Separate latency and throughput
- 😫 Only benchmark happy path

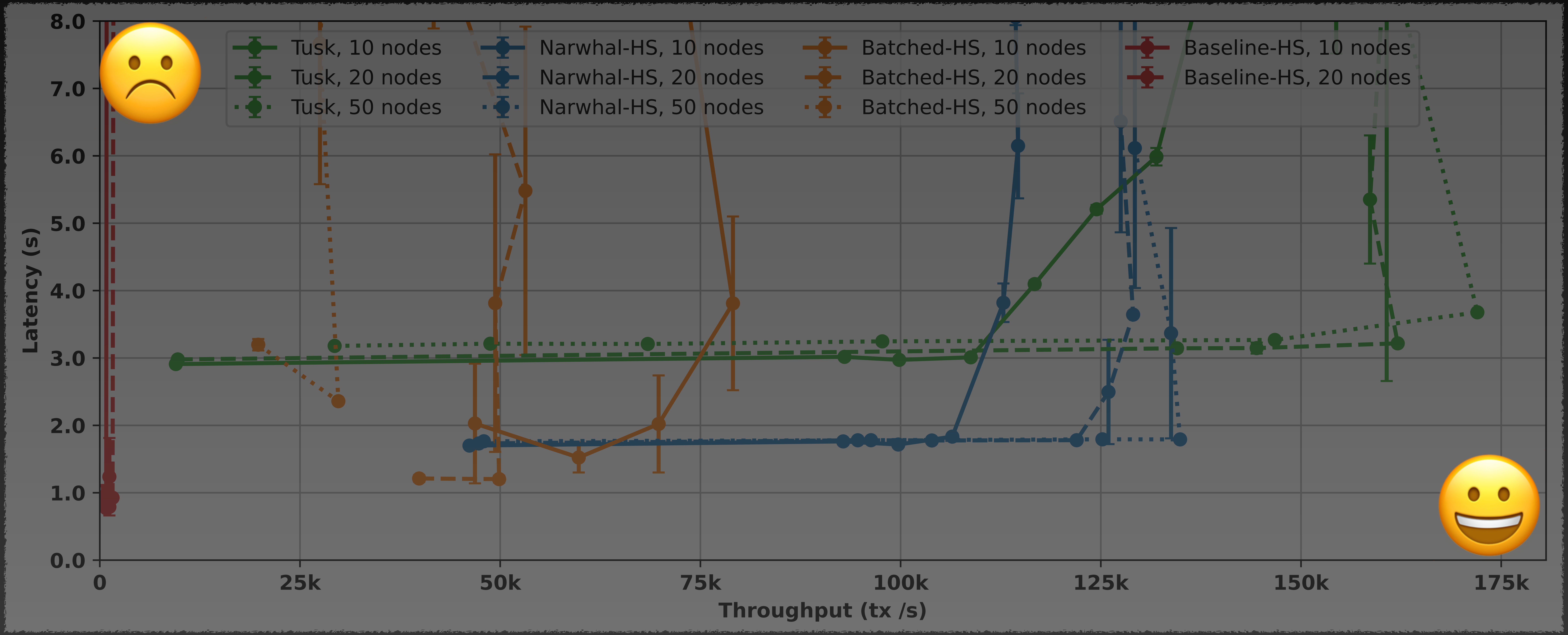
Evaluation

Throughput latency graph



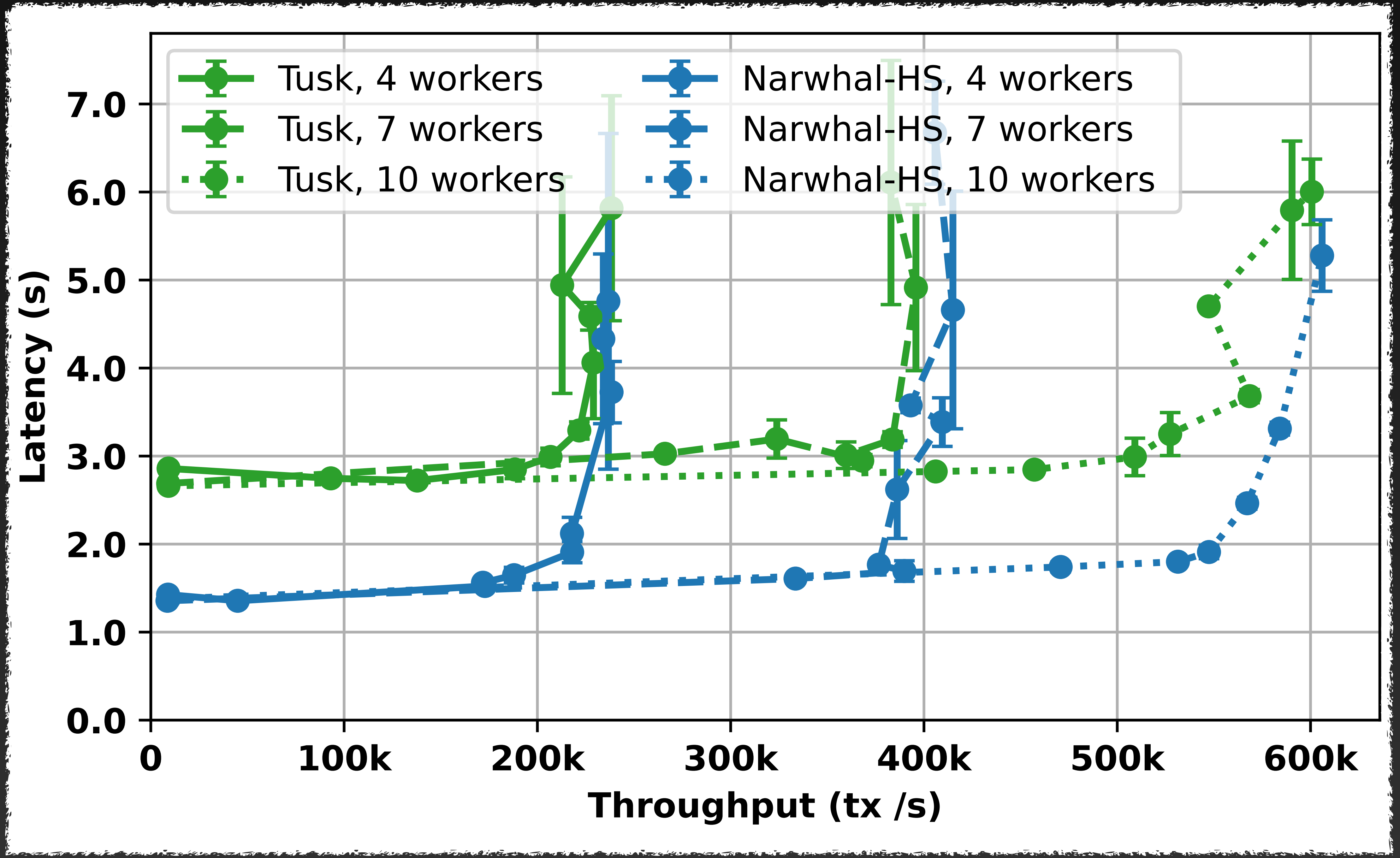
Evaluation

Throughput latency graph



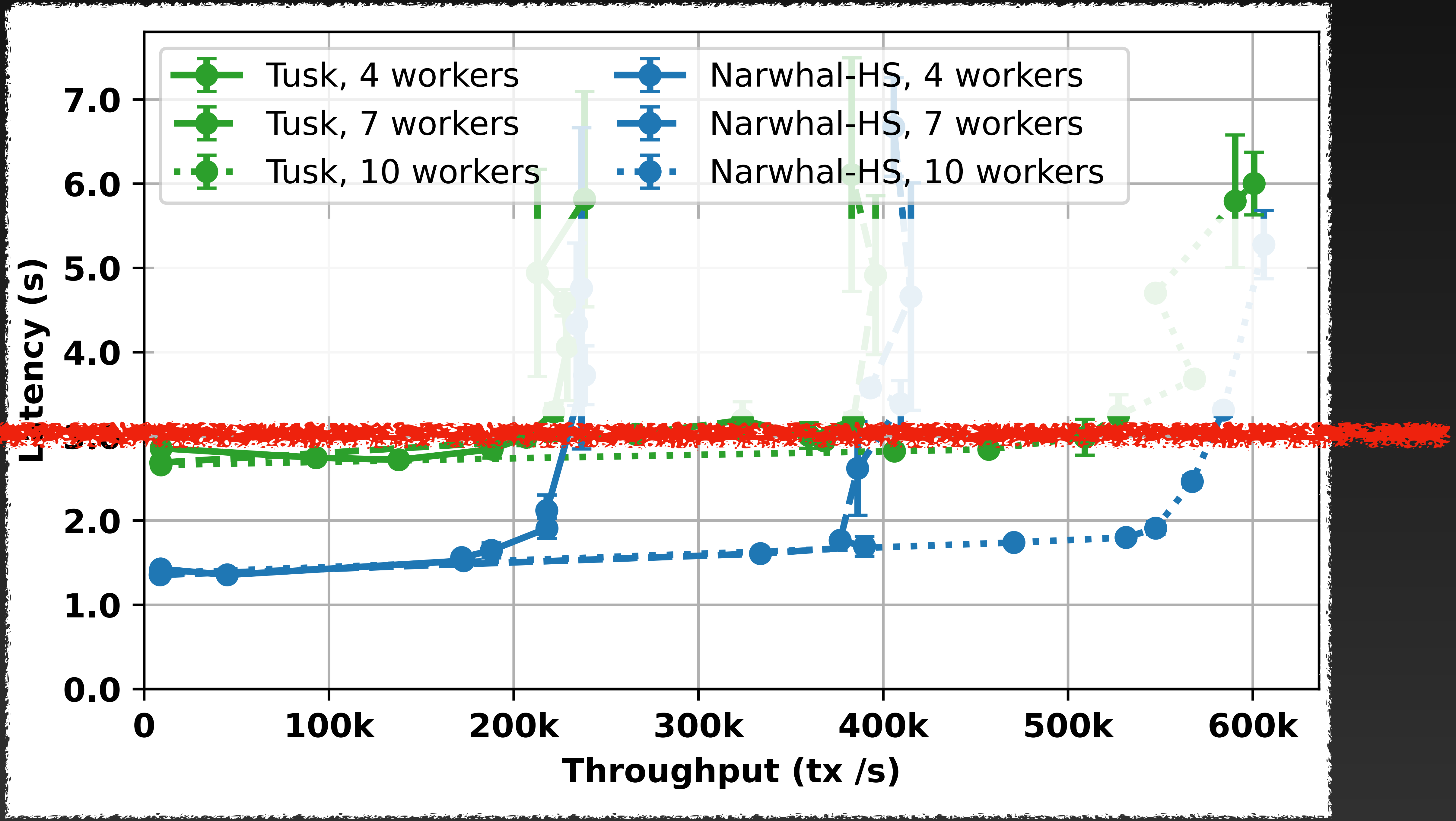
Evaluation

Scalability



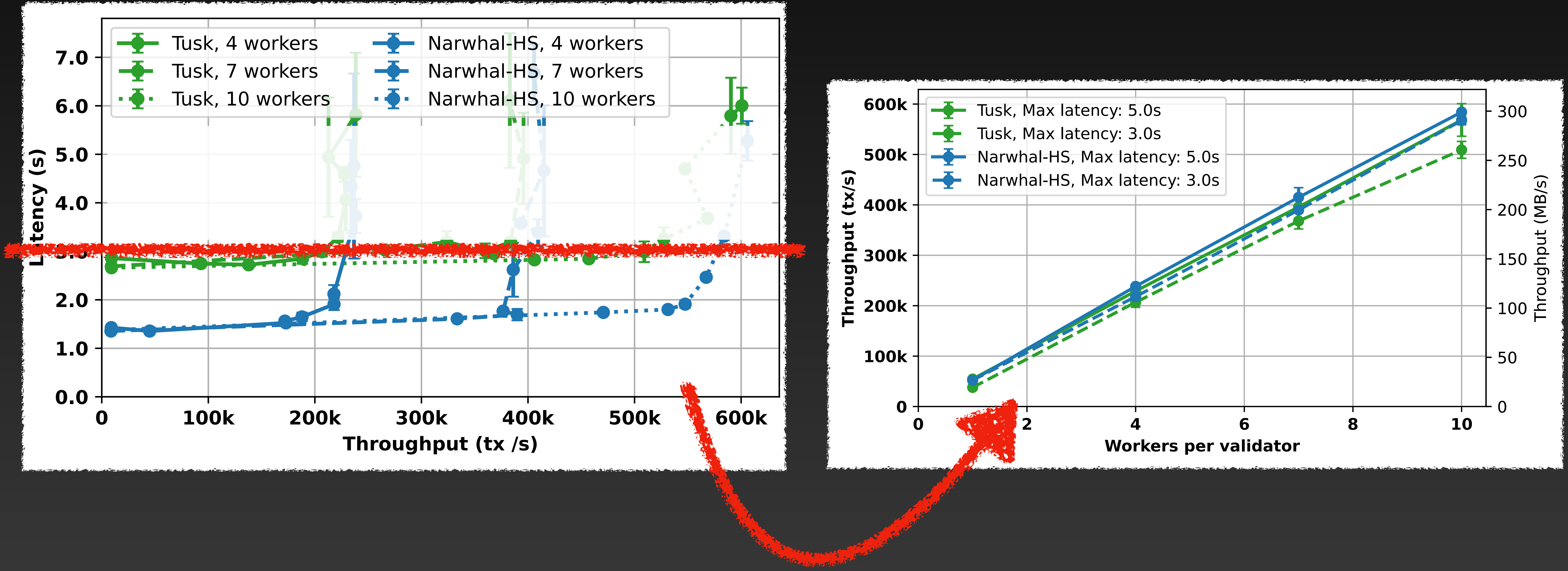
Evaluation

Scalability



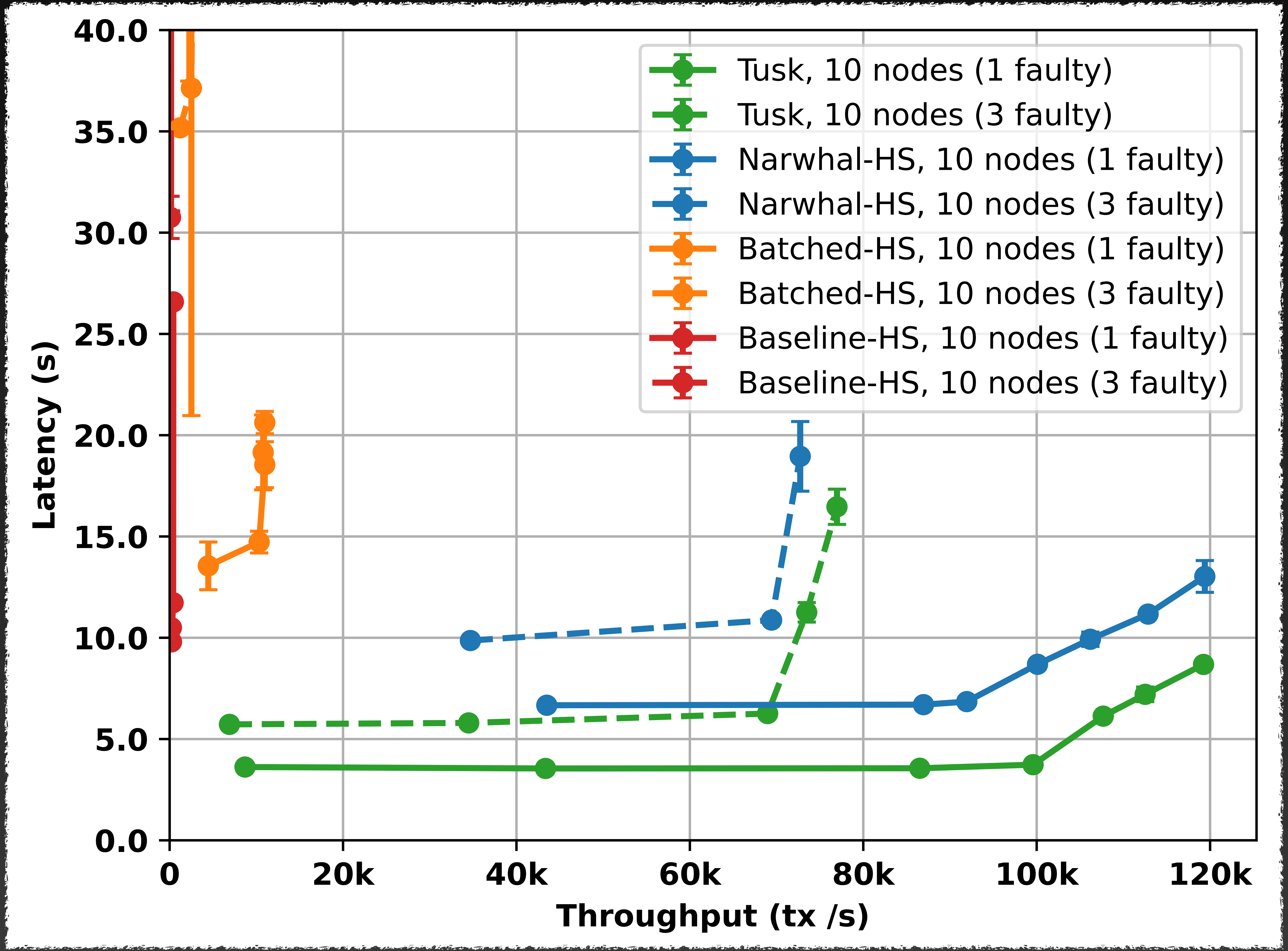
Evaluation

Scalability



Evaluation

Performance under faults



Evaluation

Typical mistakes

- 😫 Forgo persistent storage
- 😫 Do not sanitise messages
- 😫 Local/LAN benchmark + ping
- 😫 Many nodes on same machine
- 😫 Change parameters across runs
- 😫 Set transaction size to zero
- 😫 Preconfigure nodes with txs
- 😫 Send a single burst of transactions
- 😫 Benchmark for a few seconds
- 😫 Start timer in the batch maker
- 😫 Evaluate latency w/ only the first tx
- 😫 Separate latency and throughput
- 😫 Only benchmark happy path

Evaluation

Still many caveats

- Perfect load balance
- Transaction deduplication
- Synthetic load
- No Byzantine adversary
- No network adversary
- Only AWS network

Conclusion

Narwhal & Tusk

- Separate consensus and data dissemination for high performance
- Scalable design, egalitarian resource utilisations

- **Paper:** https://arxiv.org/pdf/2105.11827.pdf
- **Code:** https://github.com/asonnino/narwhal

Acknowledgements

George
Danezis

Lefteris
Kokoris-Kogias

Alexander
Spiegelman

Alberto
Sonnino

Work done at Facebook Novi

Future Works

Come talk to us!

- Performance under DDoS attack?
- How to implement scalable execution?

alberto@mystenlabs.com

Alberto Sonnino