Lemonshark

Asynchronous DAG-BFT With Early Finality

Michael Yiqing Hu, Alvin Hong Yao Yan, Yang Yihan, Liu Xiang, Li Jialin

Lemonshark 

- Fully Asynchronous Byzantine Atomic Broadcast Protocol.
  - Optimal resilience ($N = 3f + 1$).
  - For **SMR** (State Machine Replication)

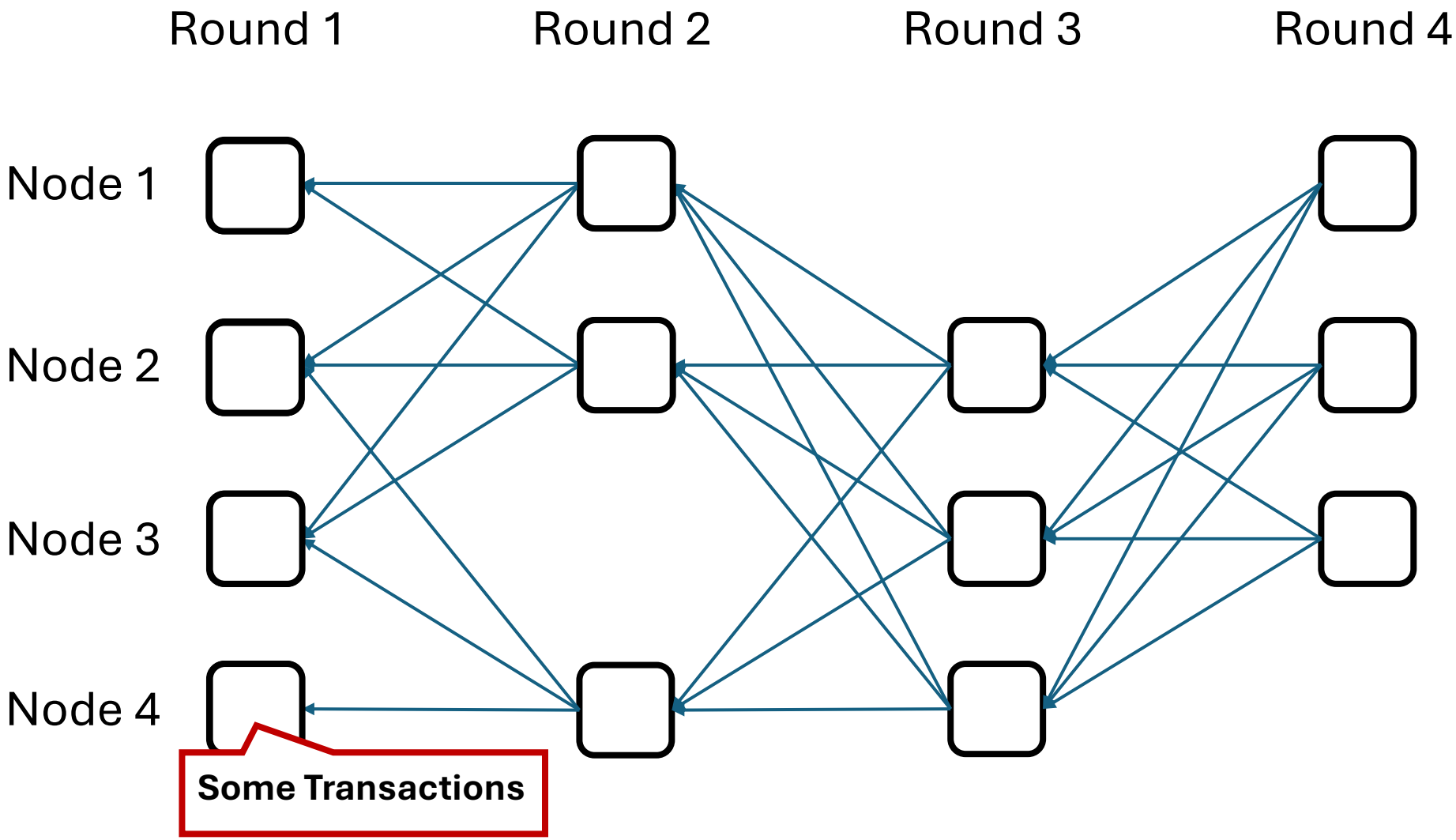
- Utilizes a Directed Acyclic Graph (DAG-BFT) structure
  - Akin to DAG-Rider (PODC'21), Tusk (EuroSys'22), Bullshark (CCS'22).

- **Challenges** traditional paradigm:
  - Leader commitment is **required** for safe results.

- Providing **safe, finalized results** ***before*** leader election!
  - Reducing latency by up to 65%.

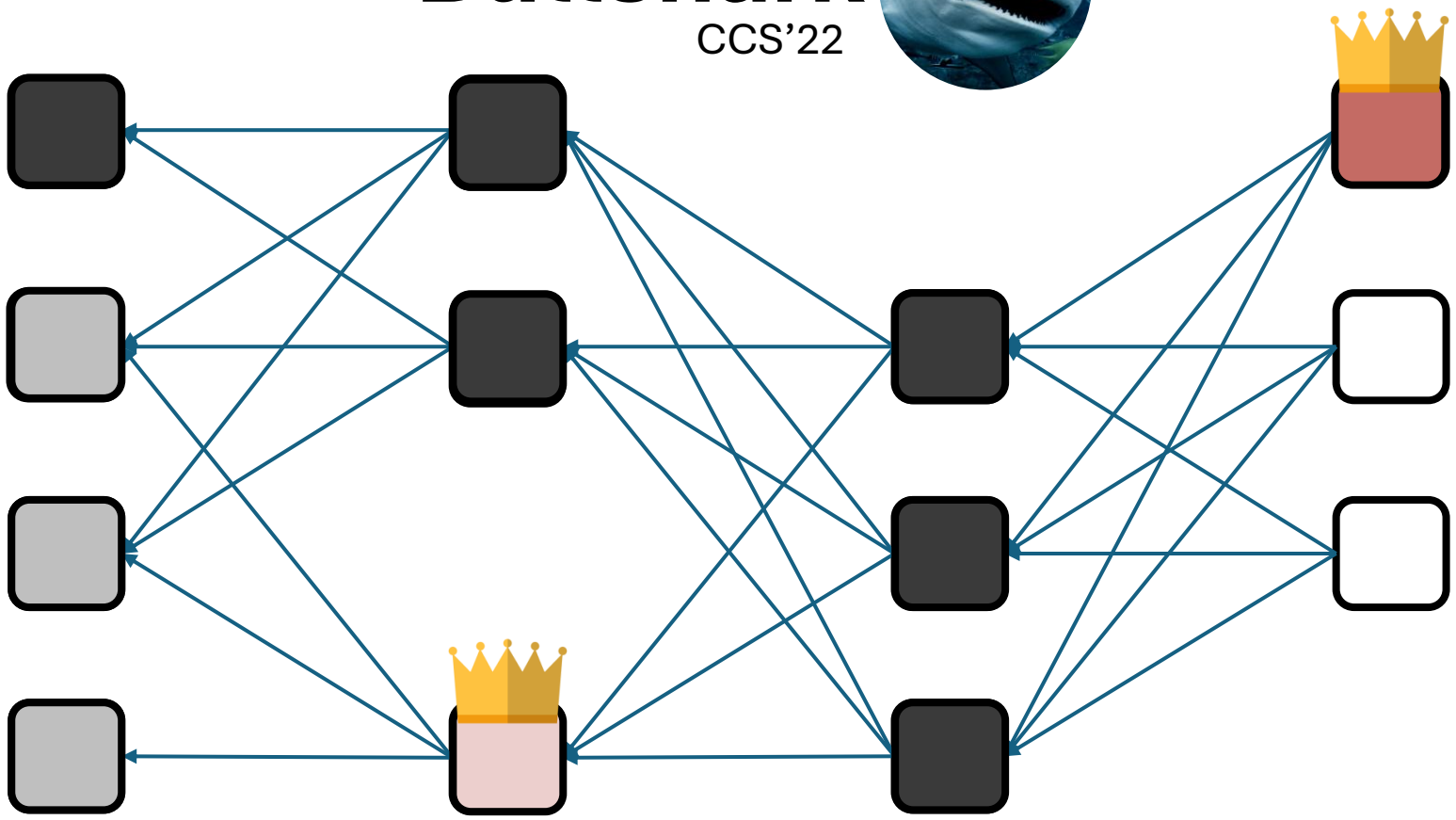
Bullshark

CCS'22



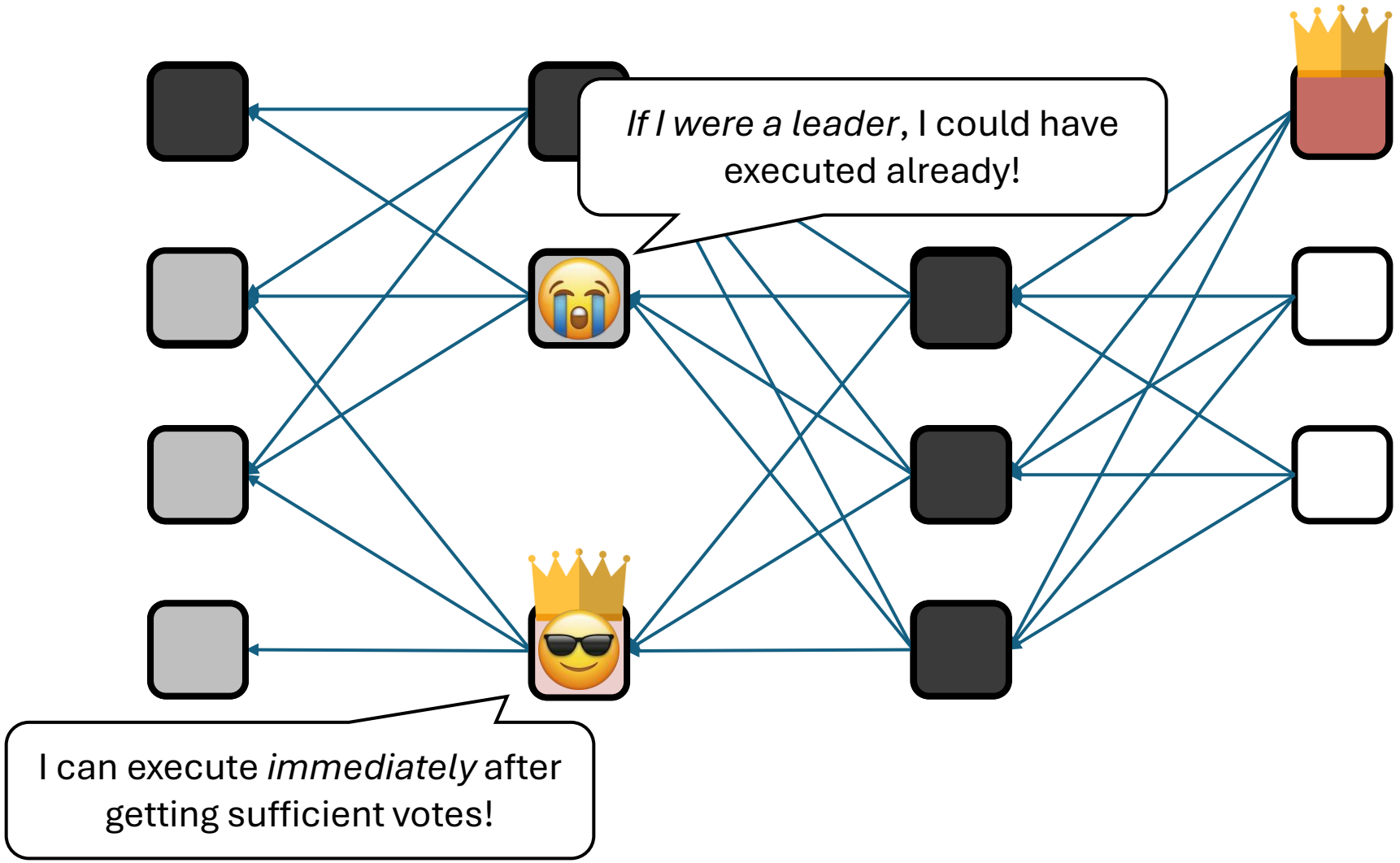
Bullshark

CCS'22

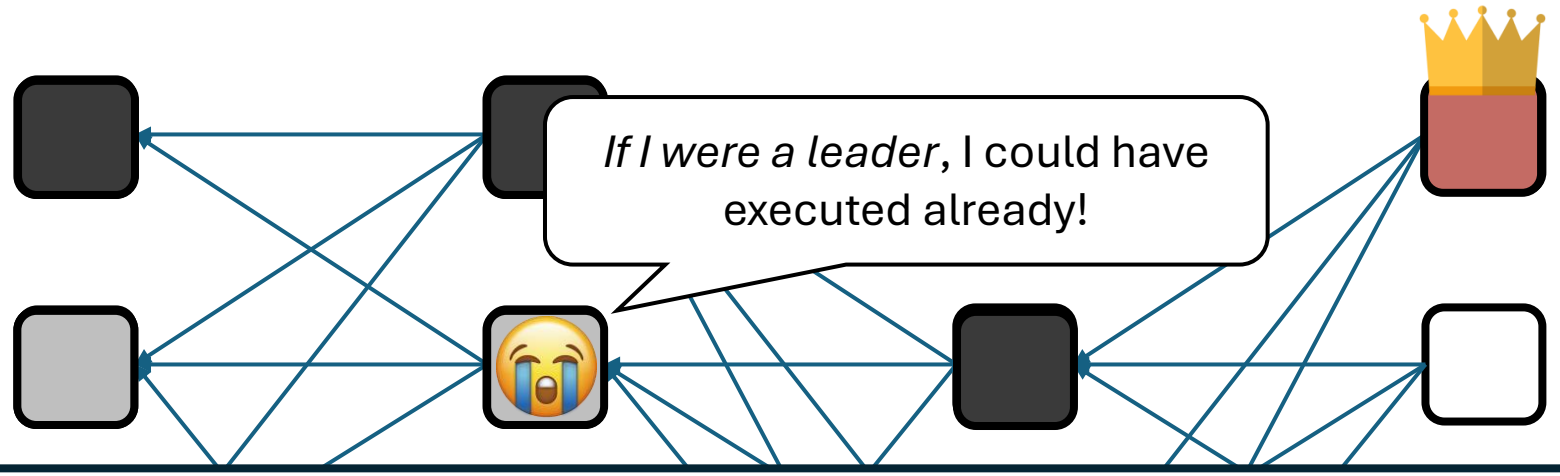


Leader election → Order blocks → Execute Transactions → Safe results

Oppressed by the hegemony of Leaders



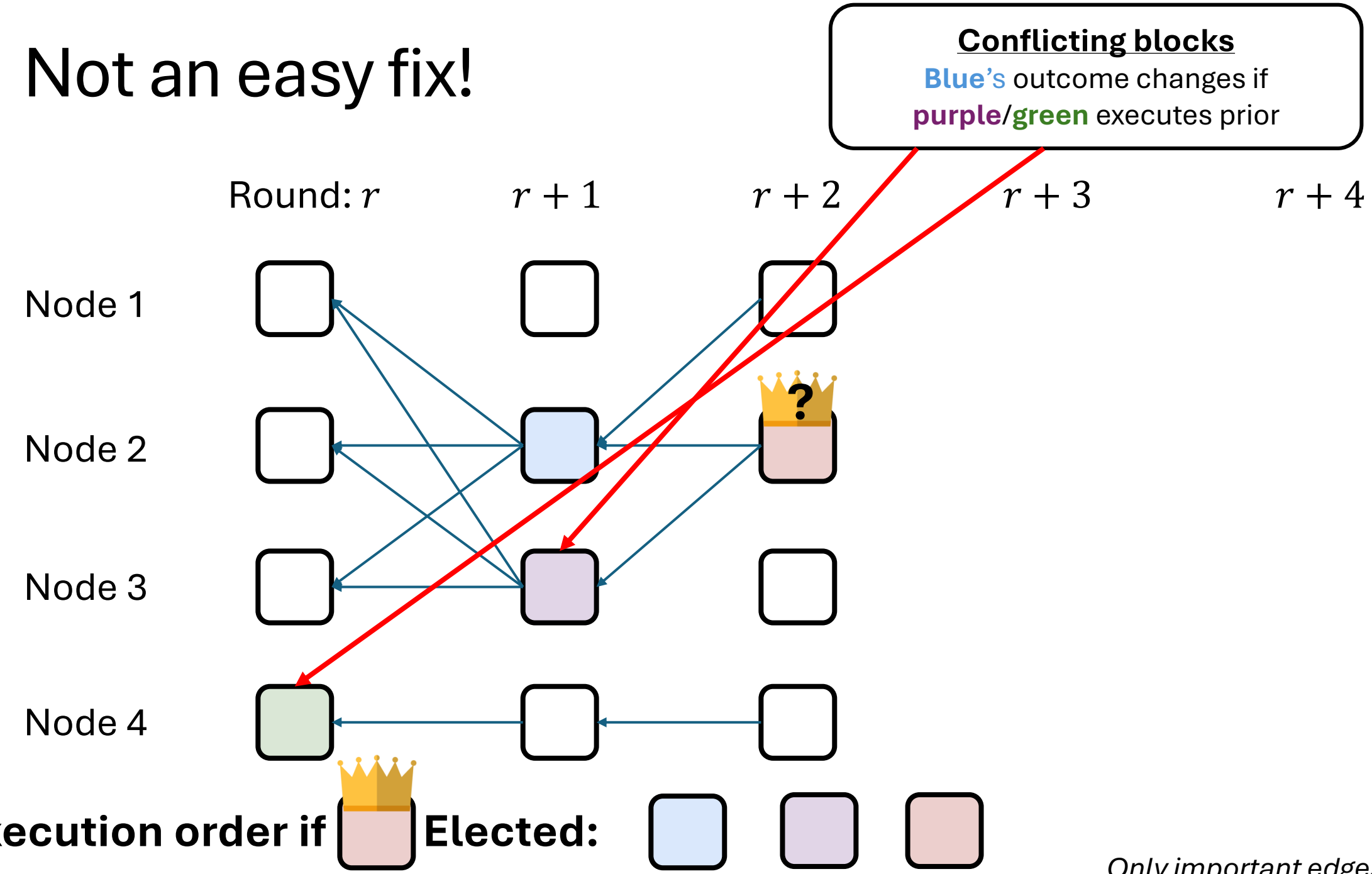
Oppressed by the hegemony of Leaders



[*Early Finality*](): Can we obtain safe results for non-leader blocks before they are committed by a leader?

I can execute *immediately* after getting sufficient votes!

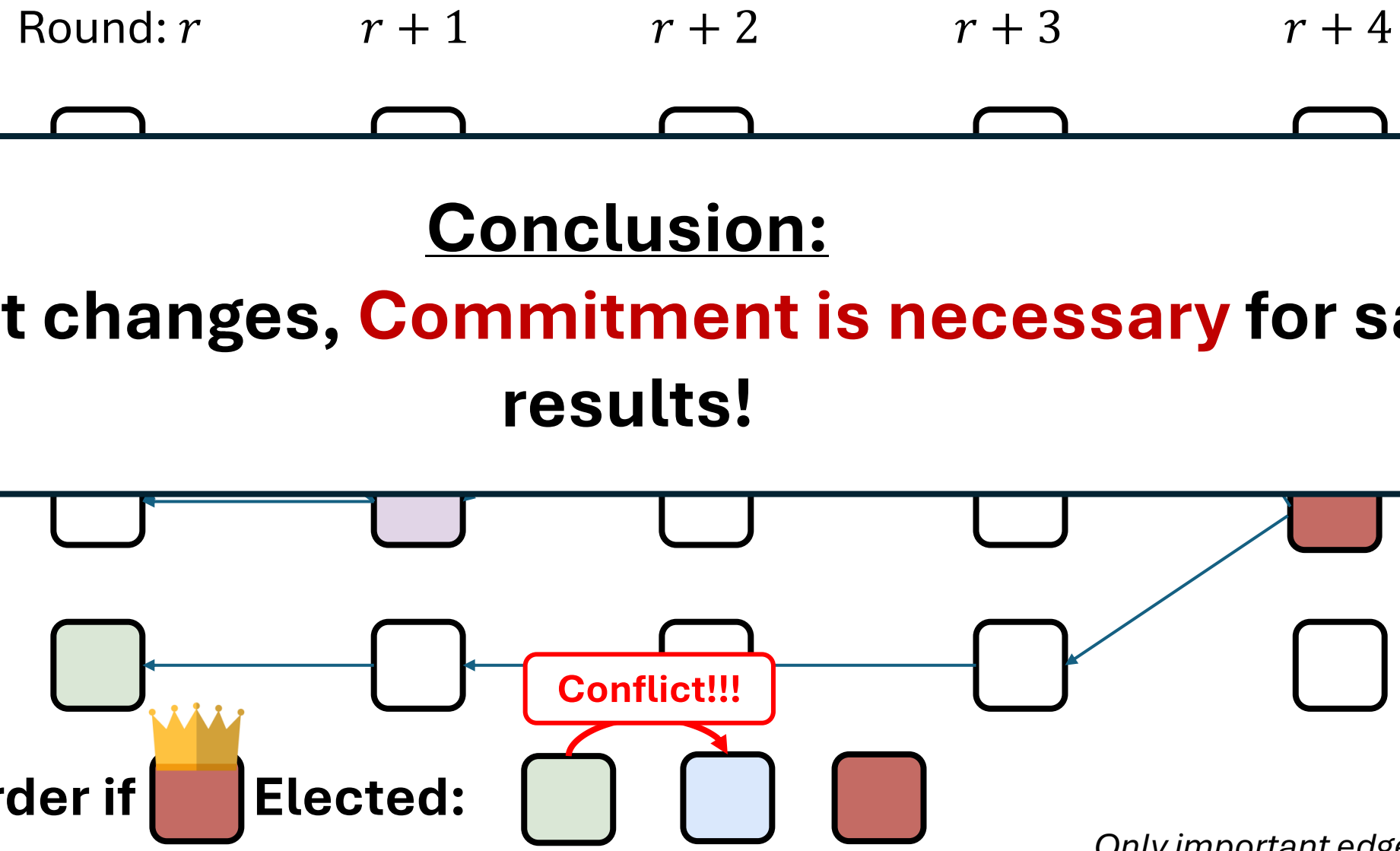
Not an easy fix!



Only important edges illustrated

Not an easy fix!

Conflicting blocks
Blue's outcome changes if purple/green executes prior



Conclusion:

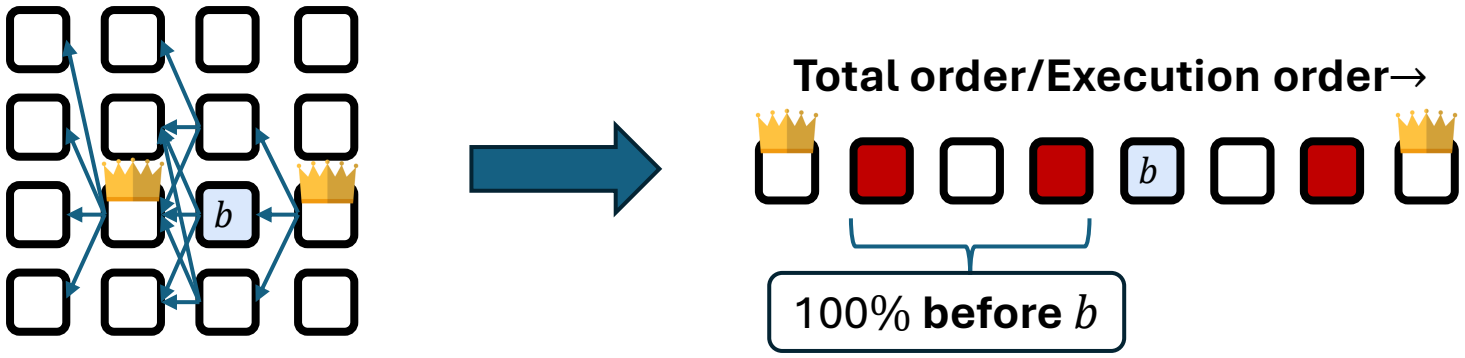
Without changes, Commitment is necessary for safe results!

Only important edges illustrated

Lemonshark's Insight

- Determining a block's (b) **finalized result** **before commitment** requires:

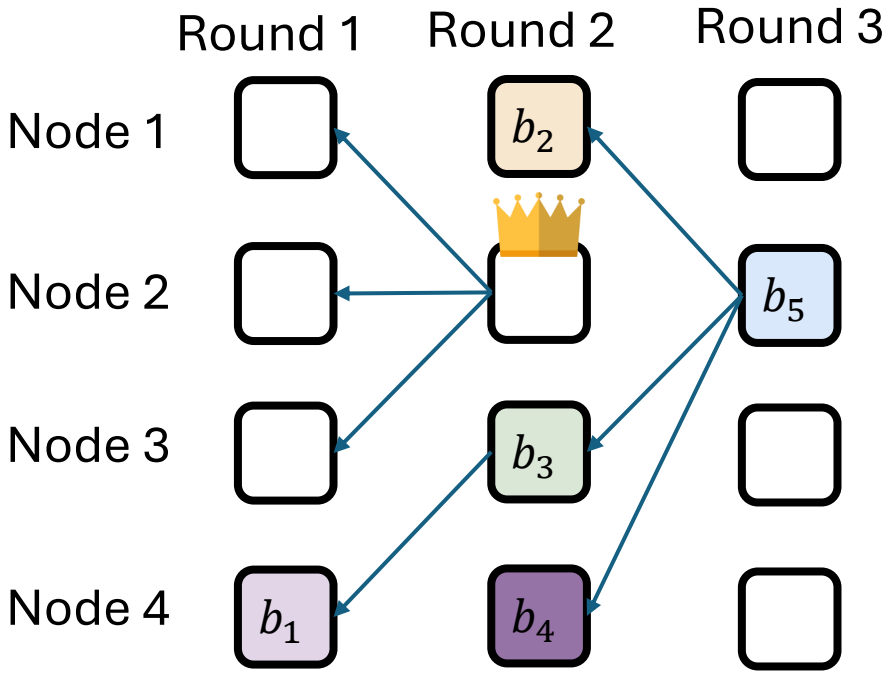
1. Knowing b will commit eventually.
2. Identifying all **conflicting blocks** that will commit concurrently, **but ordered before b** in the eventual execution order.



Change 1: Causal History Sorting Algorithm

Result:

Blocks from future rounds less likely to supersede ordering.



Sorted causal history of b_5 $= H_{b_5} = \{b_1, b_2, b_4, b_3, b_5\} =$

$\{$ b_1 b_2 b_3 b_4 b_5 $\}$

Change 2: Key-space partition

- Each transaction read-writes to a particular key

Result:

1. **No conflicting actions within the same round.**
2. **Realistic tracking of conflicts.**

Mo' Shards, Mo' problems

- Key-space partitioning introduces cross-shard transaction complexity.
- To show generality, Lemonshark handles 3 types of transactions:

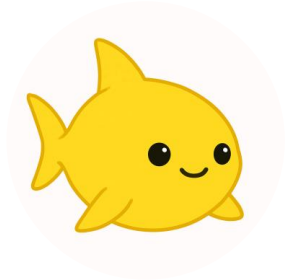
Type α Transactions
(Single-shard)
- Intra-shard transactions that read and write exclusively within k_i.

Type β Transactions
(cross-shard reads)
- Cross-shard transactions that read from multiple shards but write exclusively to k_i.

Type γ Transactions
(Atomic Coordination)
- Atomic multi-shard transactions consisting of multiple α, β sub-transactions that maintain serializability as a pair.

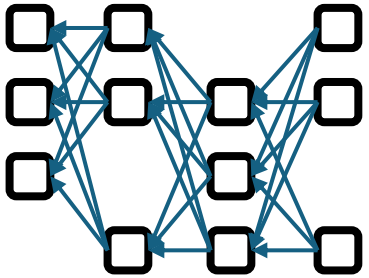
Essential Database operations (more types possible)

What Lemonshark does



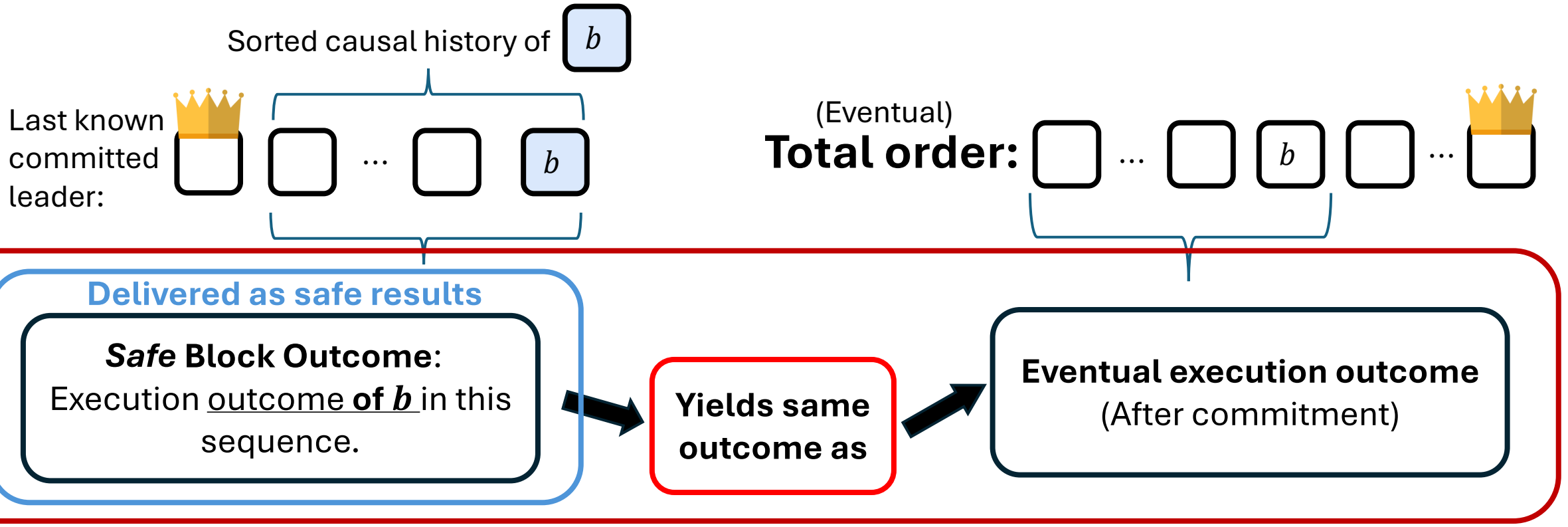
-  Bullshark's consensus core* for leader election.

- Each node utilizes its own local view of the DAG
  - Possibly incomplete!



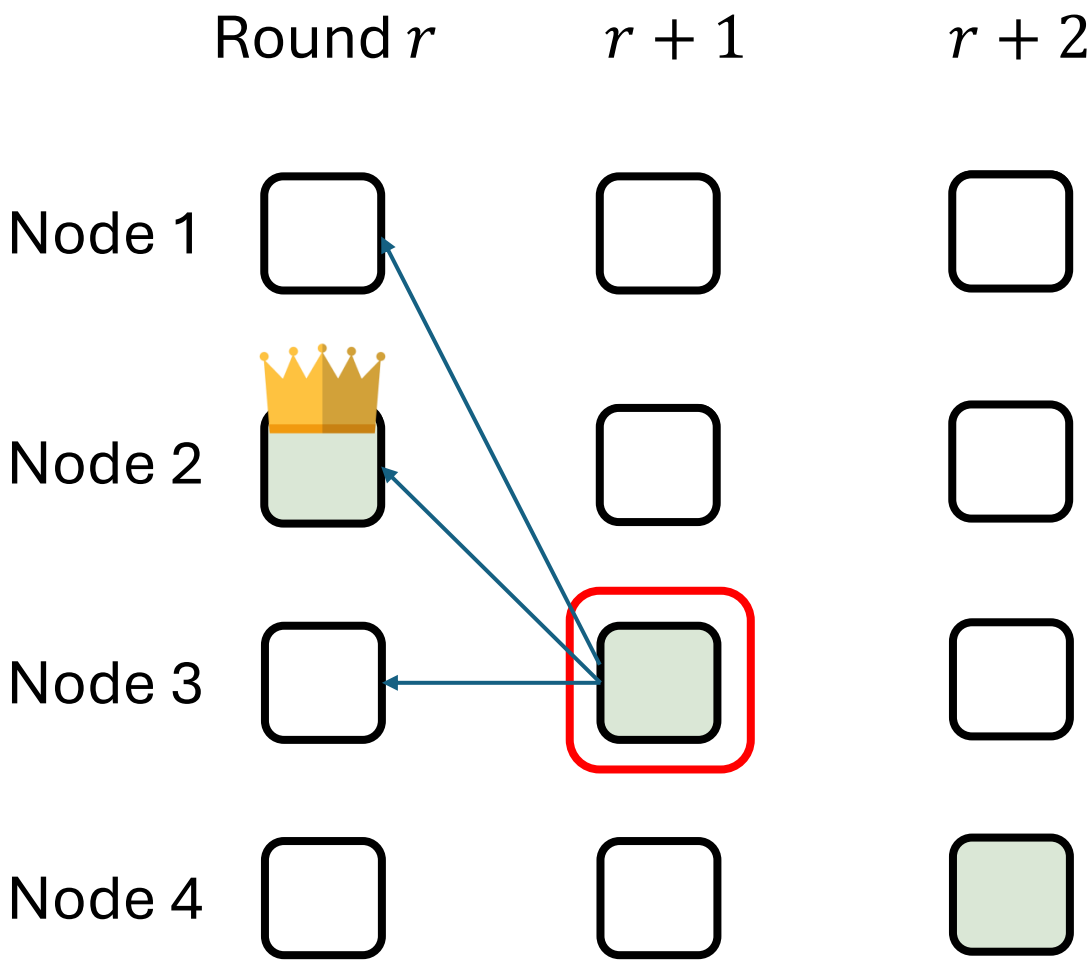
- **Identify** which transactions/blocks meet the criteria for **Early Finality**; **no enforcement**.

How is Early Finality evaluated?



Determined before commitment!

Type α (Single-shard) transactions: **Base case**

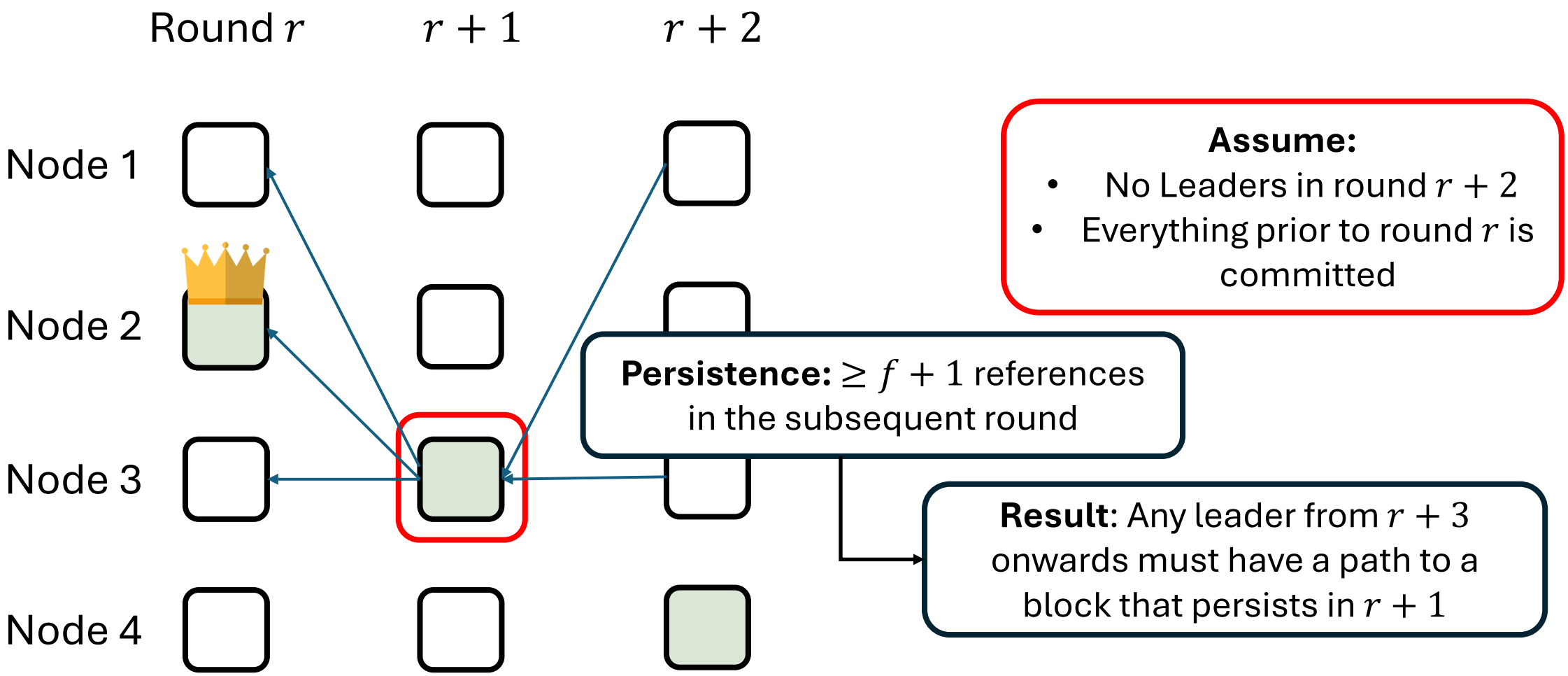


Assume:

- No Leaders in round $r+2$
- Everything prior to round r is committed

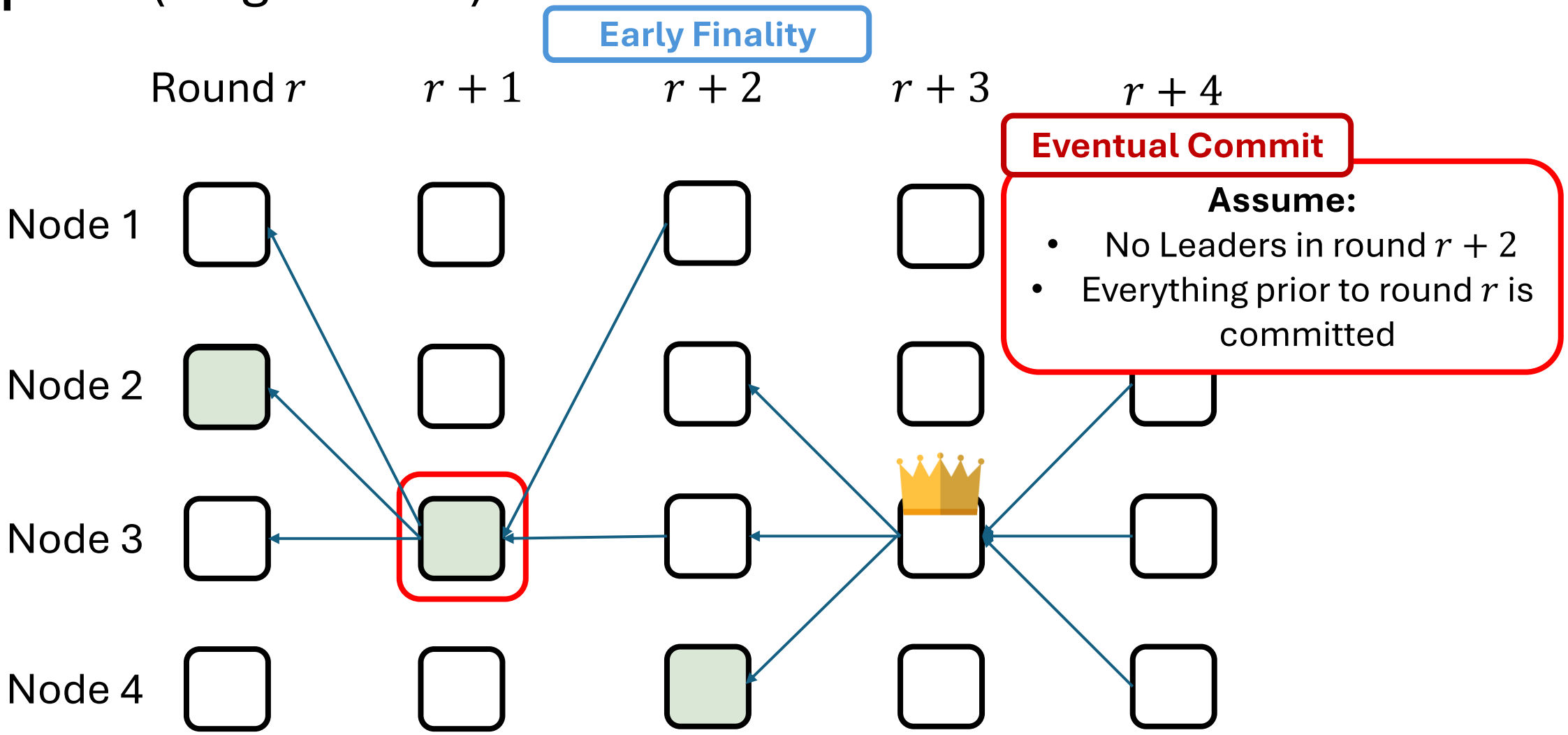
Only important edges illustrated

Type α (Single-shard) transactions: **Base case**



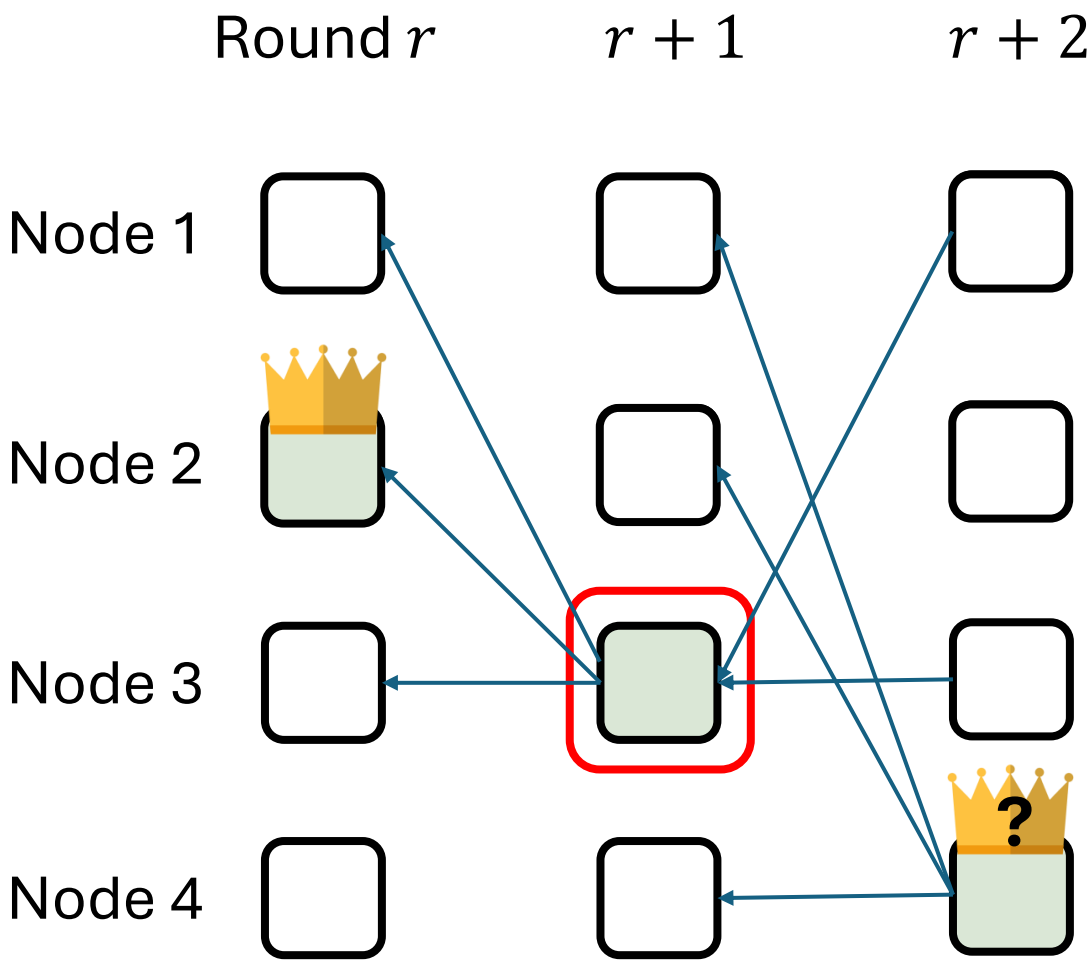
Only important edges illustrated

Type α (Single-shard) transactions: **Base case**



Only important edges illustrated

Type α (Single-shard) transactions: **leaders?** (1)



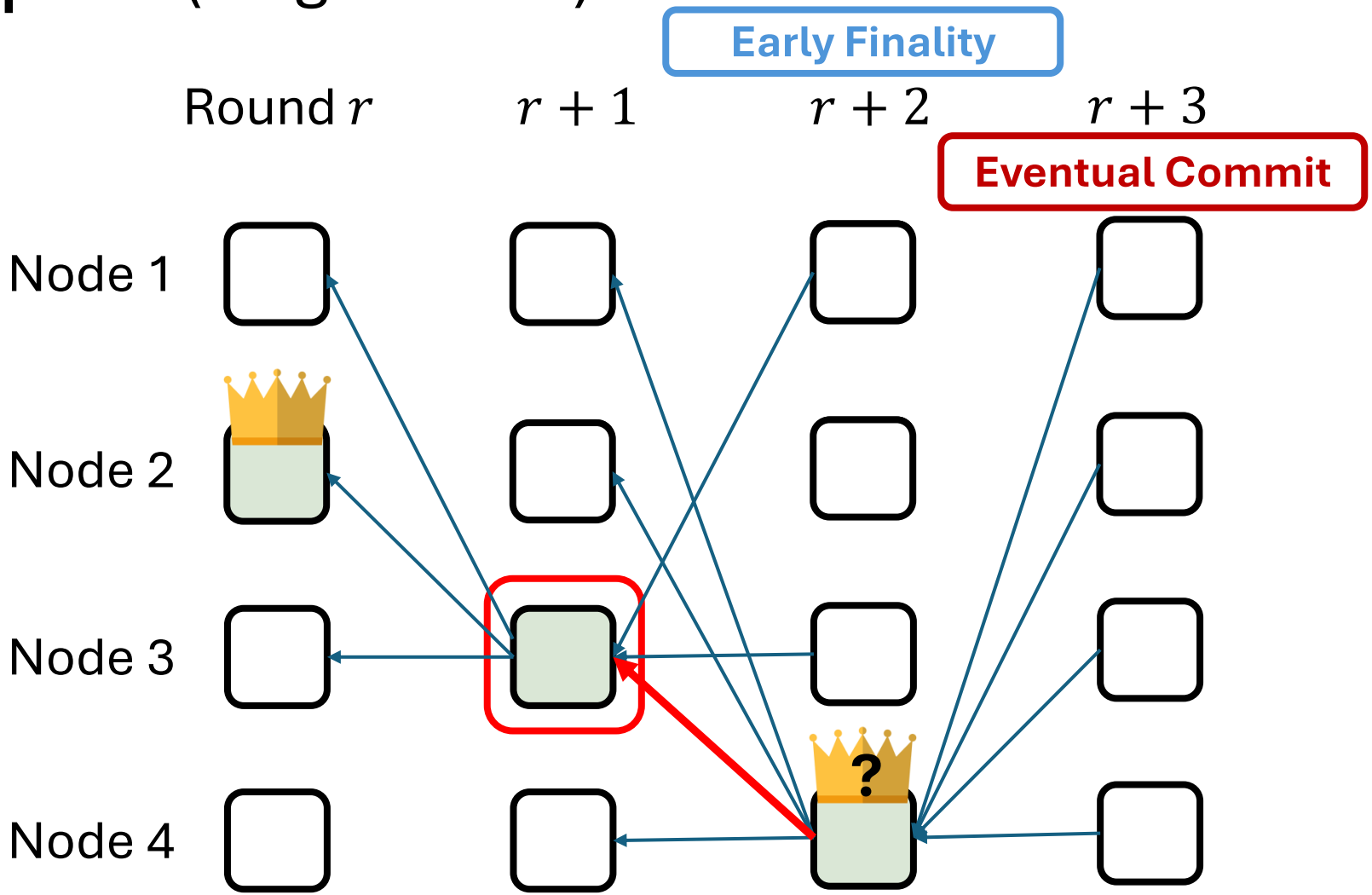
Assume:

- ~~No Leaders in round $r+2$~~
- Everything prior to round r is committed

Problem!
This potential leader might **execute first!**

Only important edges illustrated

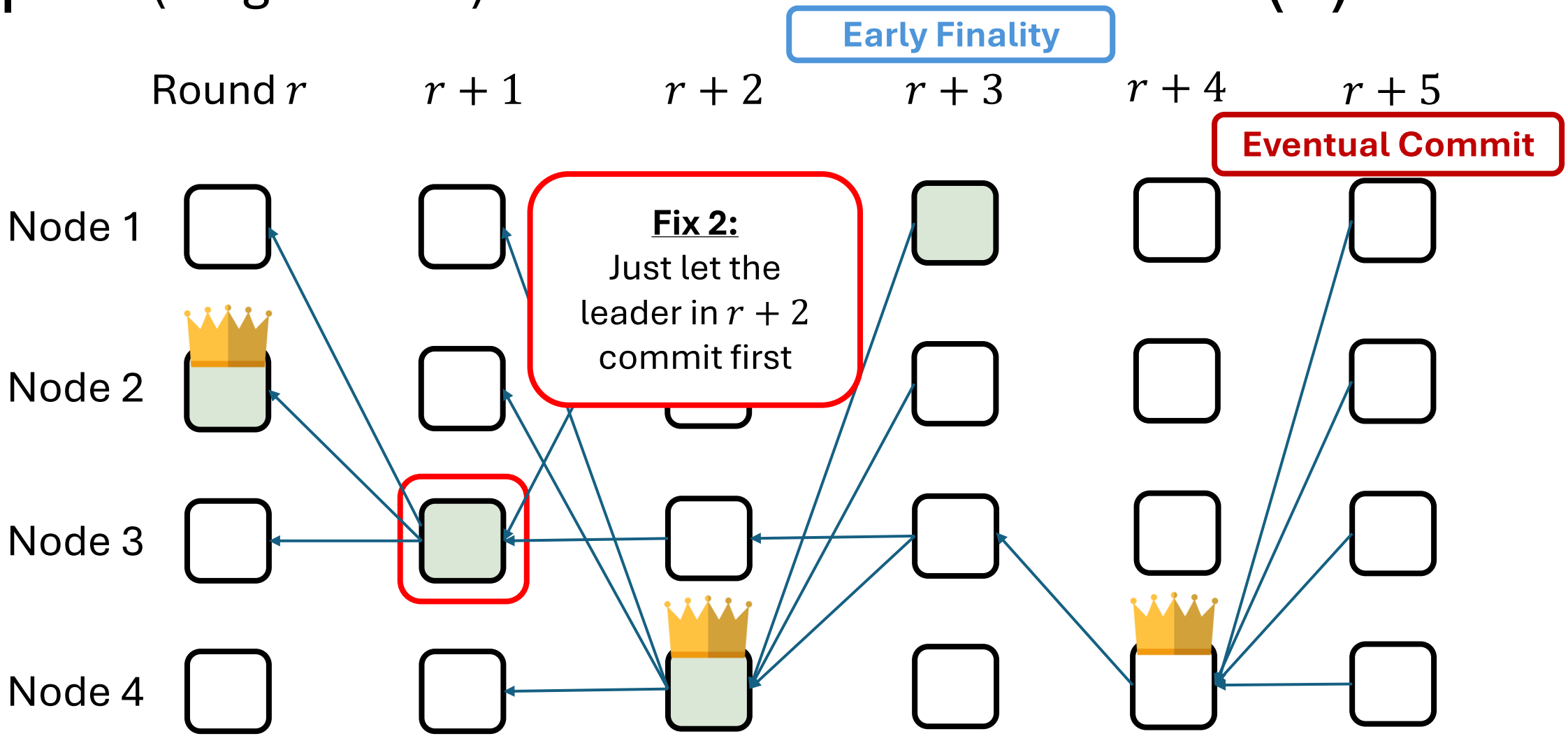
Type α (Single-shard) transactions: **leaders?** (2)



Fix 1:
Green block in the next round having a path to our green block

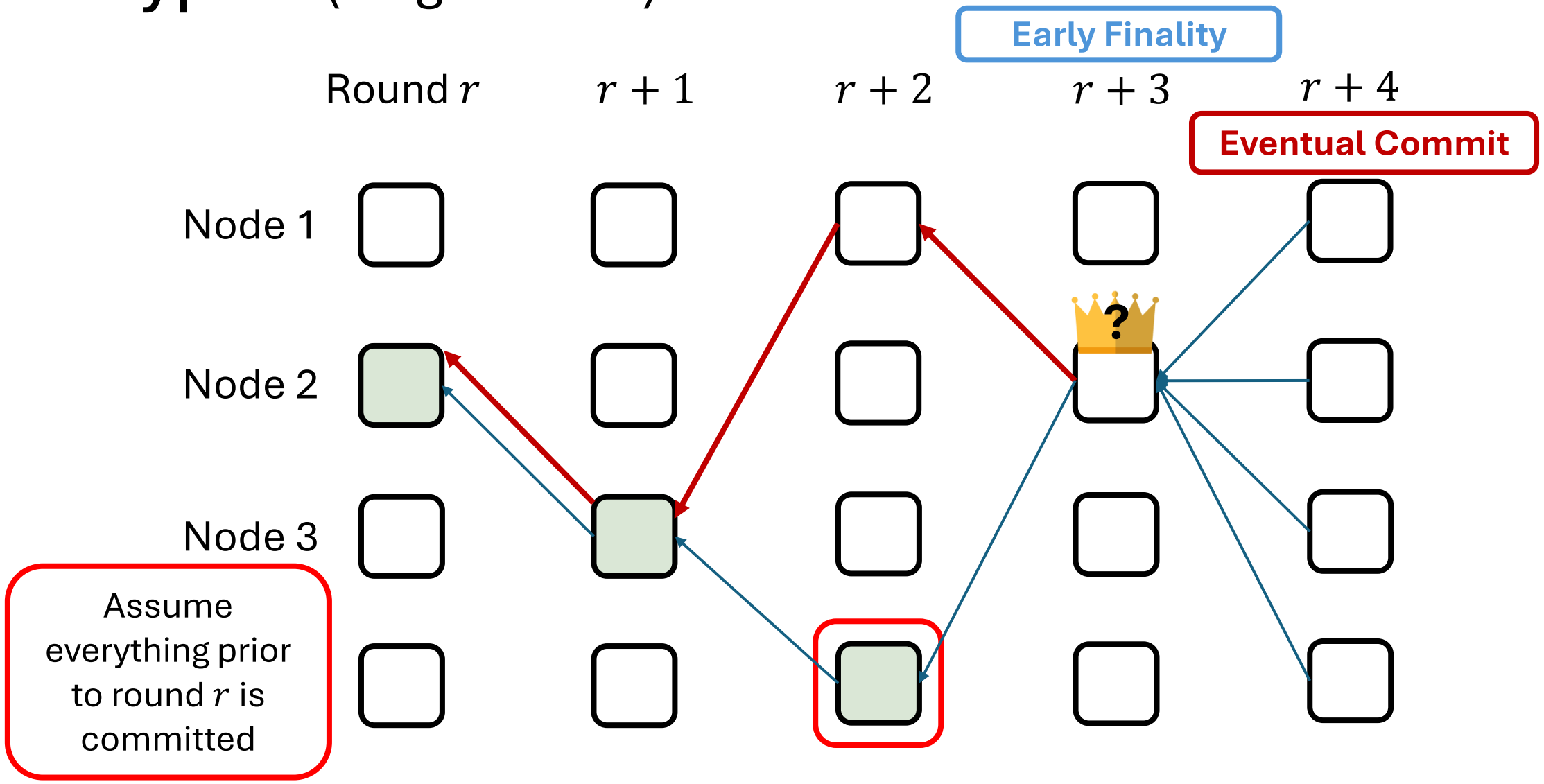
Only important edges illustrated

Type α (Single-shard) transactions: **leaders?** (3)



Only important edges illustrated

Type α (Single-shard) transactions: **Other blocks?**



Only important edges illustrated

Lemonshark transaction types

Just covered in our presentation

Details can be found in the paper

Type α Transactions
(Single-shard)
- Intra-shard transactions that read and write exclusively within k_i.

Type β Transactions
(Cross-shard reads)
- Cross-shard transactions that read from multiple shards but write exclusively to k_i.

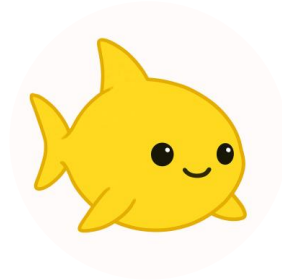
Type γ Transactions
(Atomic Coordination)
- Atomic multi-shard transactions consisting of multiple α, β sub-transactions that maintain serializability as a pair.

Essential Database operations (more types possible)

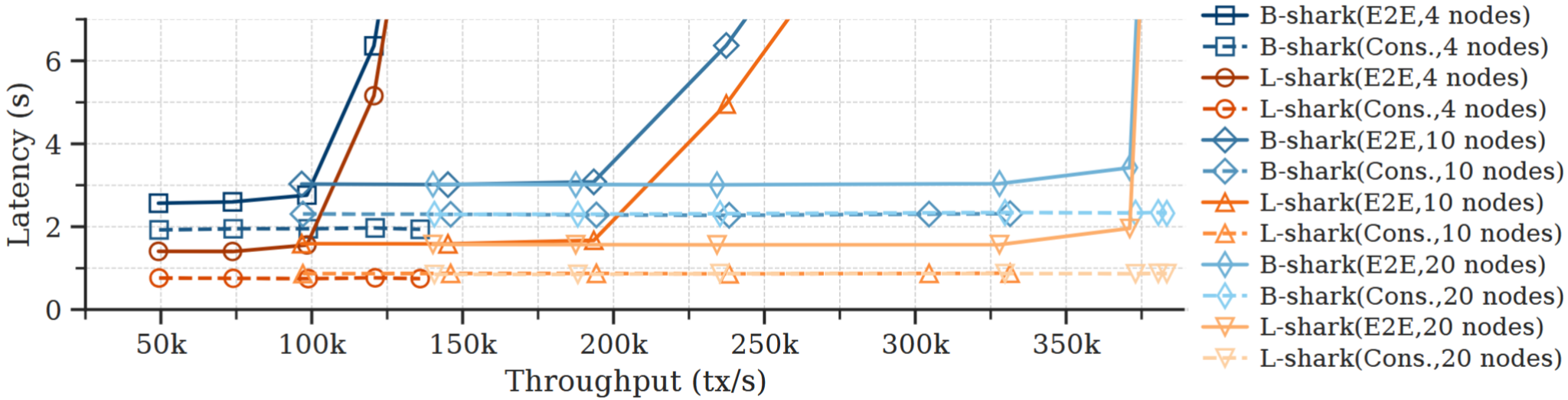
Evaluation Details

- We forked Bullshark's (asynchronous) repository.
  - Simulated cross-shard behavior to isolate consensus latency.
- Compared our performance against Bullshark.
- Replicated the network settings:
  - AWS **`m5.8xlarge`** instances across 5 regions.
  - **`US-East-1, US-West-1, Ap-southeast-2, Eu-north-1, Ap-northeast-1`**
- Averaged results of 3 independent runs.

Evaluation results

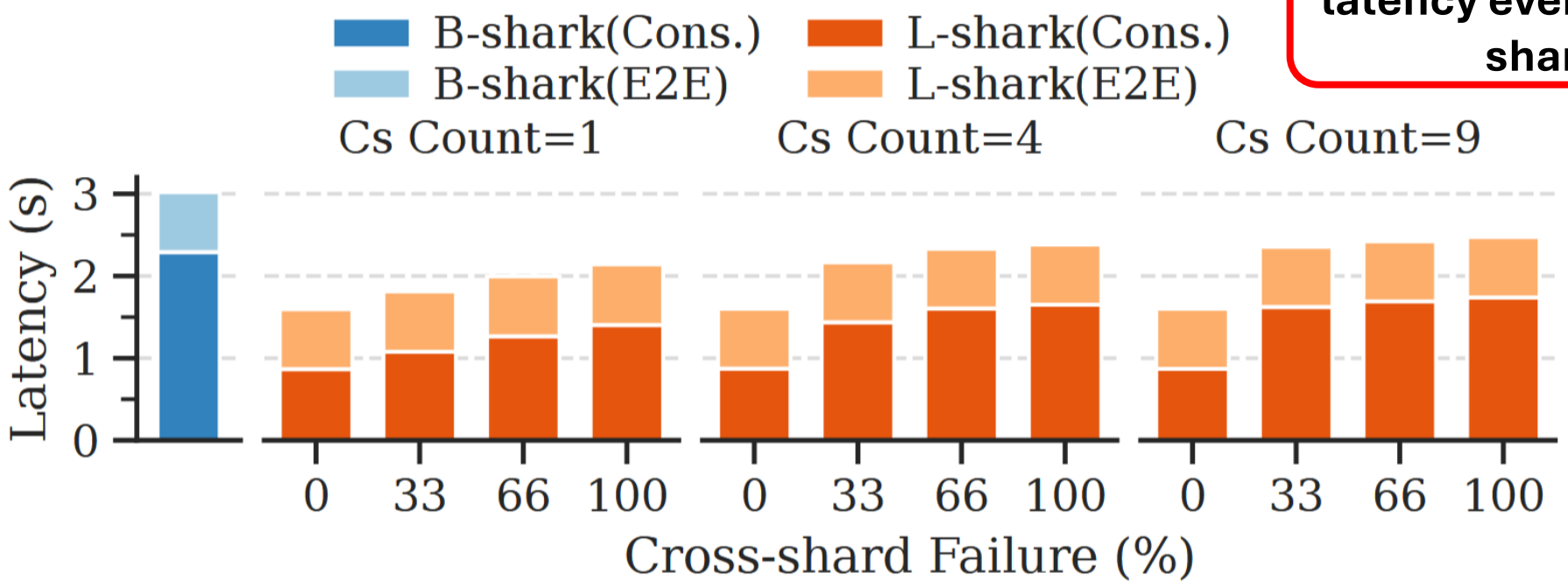


65% lower consensus latencies when scaling transaction load



Lemonshark vs Bullshark (Single-shard only)

Evaluation results

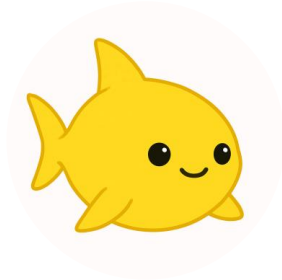


Lemonshark vs Bullshark (Cross-shard)

>25% lower consensus latency even with high cross-shard activity

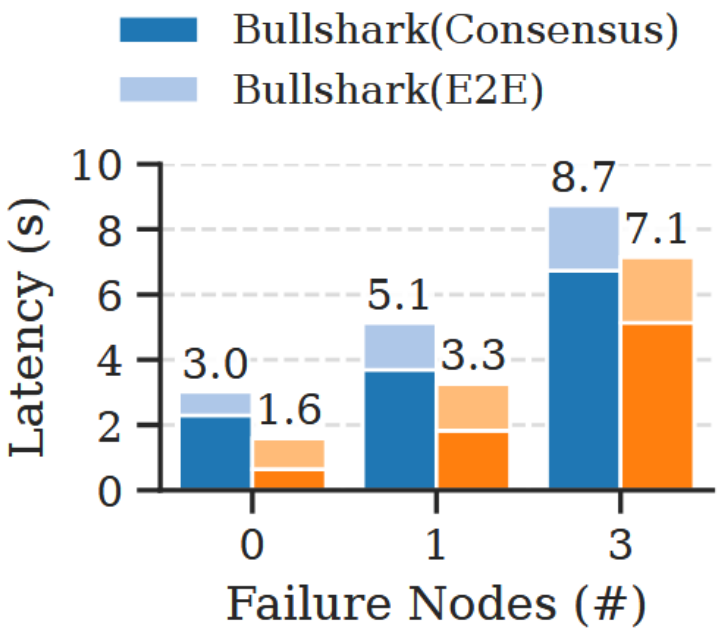
10 nodes, 100k transactions per second load

Evaluation results

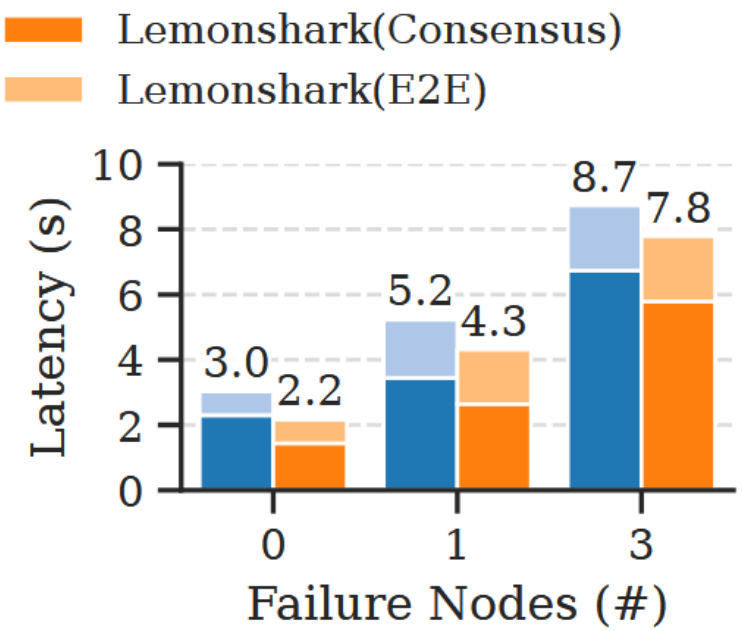


Lemonshark vs Bullshark (Crash Failures)

>24% improvement!



(Single-shard only)



(Cross-shard)

>14% improvement!

10 nodes, 100k transactions per second load

Summary

- Lemonshark is the first Asynchronous DAG-BFT protocol that provides early finality.
  - **Finalized results before finality**!

- Additional Details in the paper:
  - Conditions for early finality for β, γ transactions.
  - Full proofs.
  - Pipelining sequentially dependent transactions with speculation

Thanks for listening!