Narwhal and Tusk:

A DAG-based Mempool and Efficient BFT Consensus

Presenter: Aditya, Howard, Karamjeet, and Divyanshu

The slides closely follow the format of Alberto Sonnino's presentation slides in ConsensusDays 21.

Tusk - asynchronous consensus

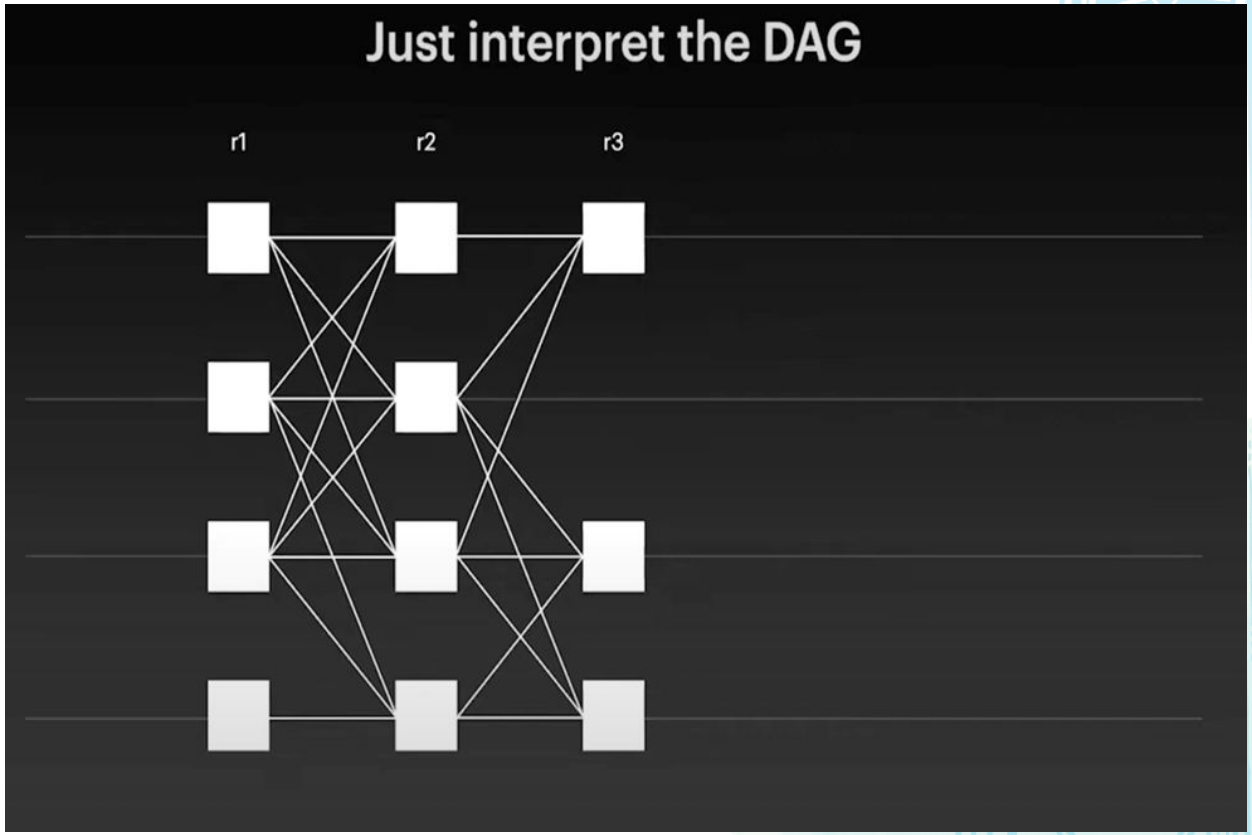
- A **zero-message overhead** asynchronous consensus protocol
- Tusk's theoretical starting point is DAGRider, from which it inherent its safety guarantees.
- Tusk modifies DAG-Rider into an implementable system and improves its latency.
- To remain live under asynchronous or DDoS attacks, **Tusk** was proposed in the paper.

Tusk - asynchronous consensus

- DAG-Rider is the first asynchronous Byzantine Atomic Broadcast protocol with optimal resilience, optimal communication complexity, and optimal time complexity.
- It's ensures all correct processes' messages are decided. Its design is notable for its efficiency, modularity, and concise logic.
- The protocol consists of two layers:
  - a communication layer for broadcasting proposals and forming a Directed Acyclic Graph (DAG), and
  - an ordering layer for local observation and ordering of proposals with no extra communication.

Tusk - asynchronous consensus

- Tusk validators operate a Narwhal mempool, but also include in each of their blocks information to generate a distributed perfect random coin.



Tusk - asynchronous consensus

•Every validator interprets the DAG based on its view. To interpret the DAG, validators divide it into waves, each of which consists of 3 consecutive rounds.

•What is a wave ? - Three consecutive rounds form a wave

- ❏ First round: Each validator proposes its block and **causal** history (PROPOSE)
- ❏ Second round: Each validator votes on the proposal by including them in their block (VOTE)
- ❏ Third round: Validators produce randomness to elect a random leader's block.

•The order may be different since it is asynchronous.

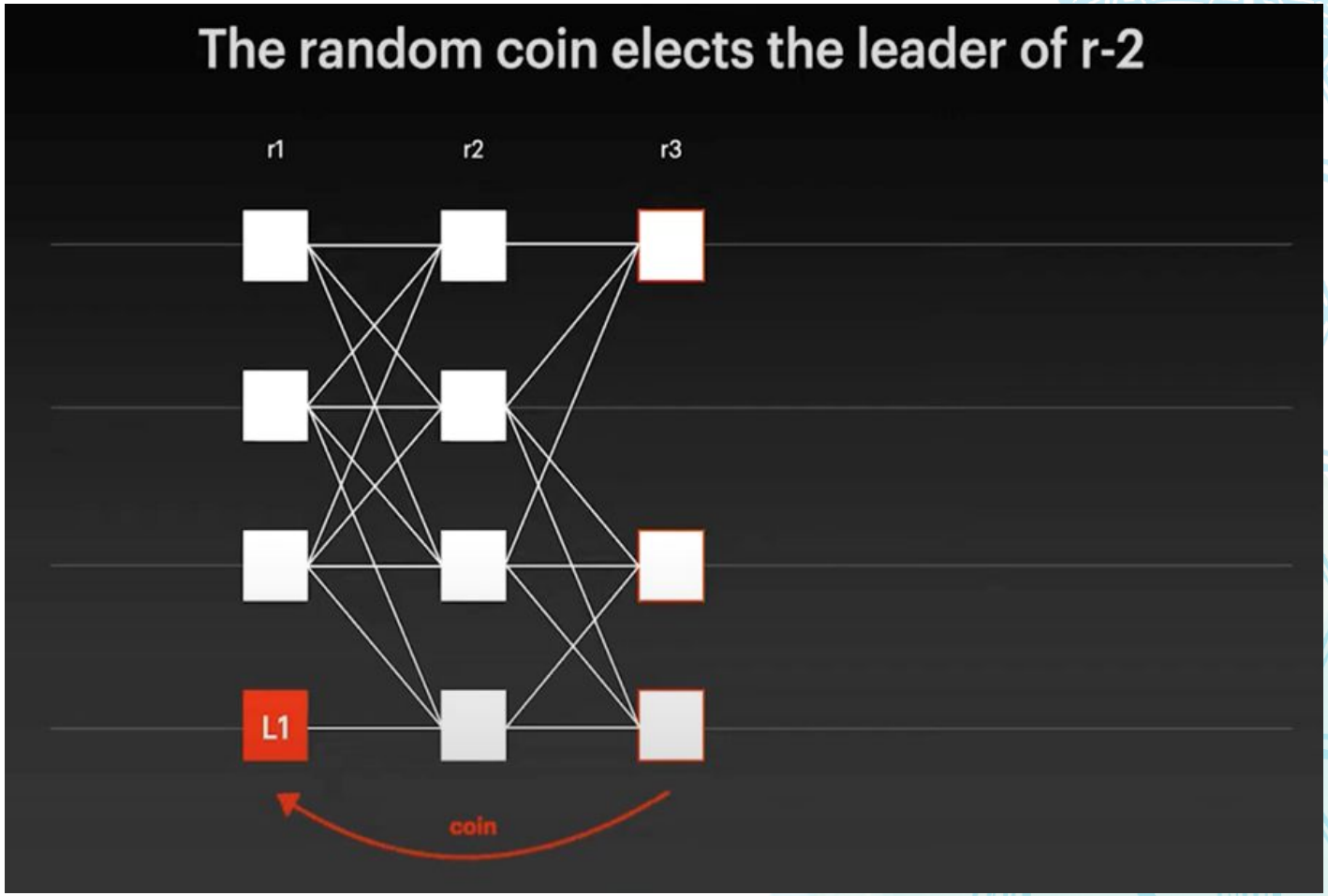
•Leader is elected and all its Sub-DAGs are committed

Tusk - asynchronous consensus

- The generation of distributed perfect random coin only happens at odd rounds such as round 3, round 5 and so on.
- The elected validators are called as Leaders

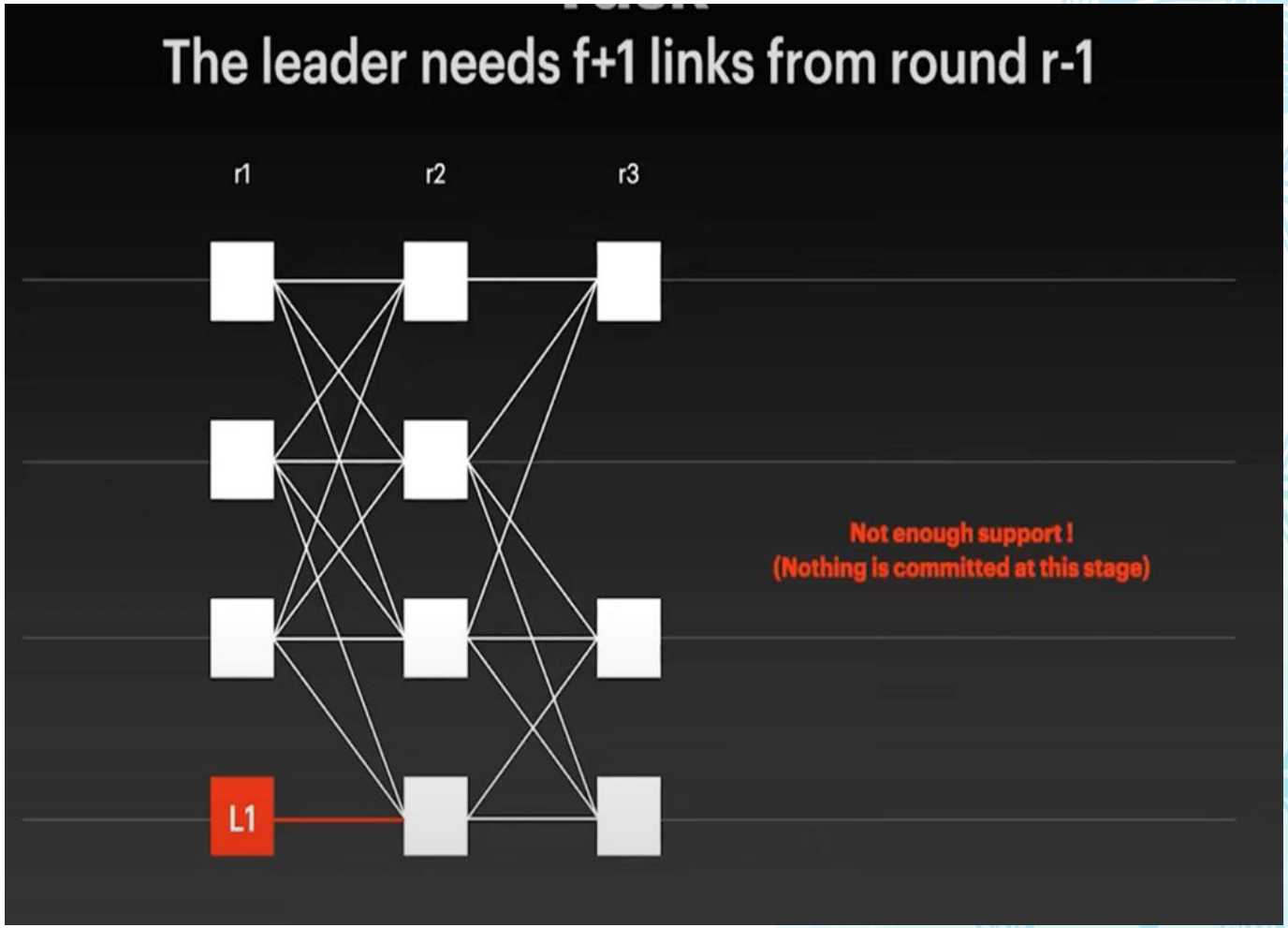
Tusk - asynchronous consensus

Here you can see when we are at round 3 which is an odd round, there is random coin generation which takes place for round r-2 i.e the round 1. Here we elect a leader and name it as L1

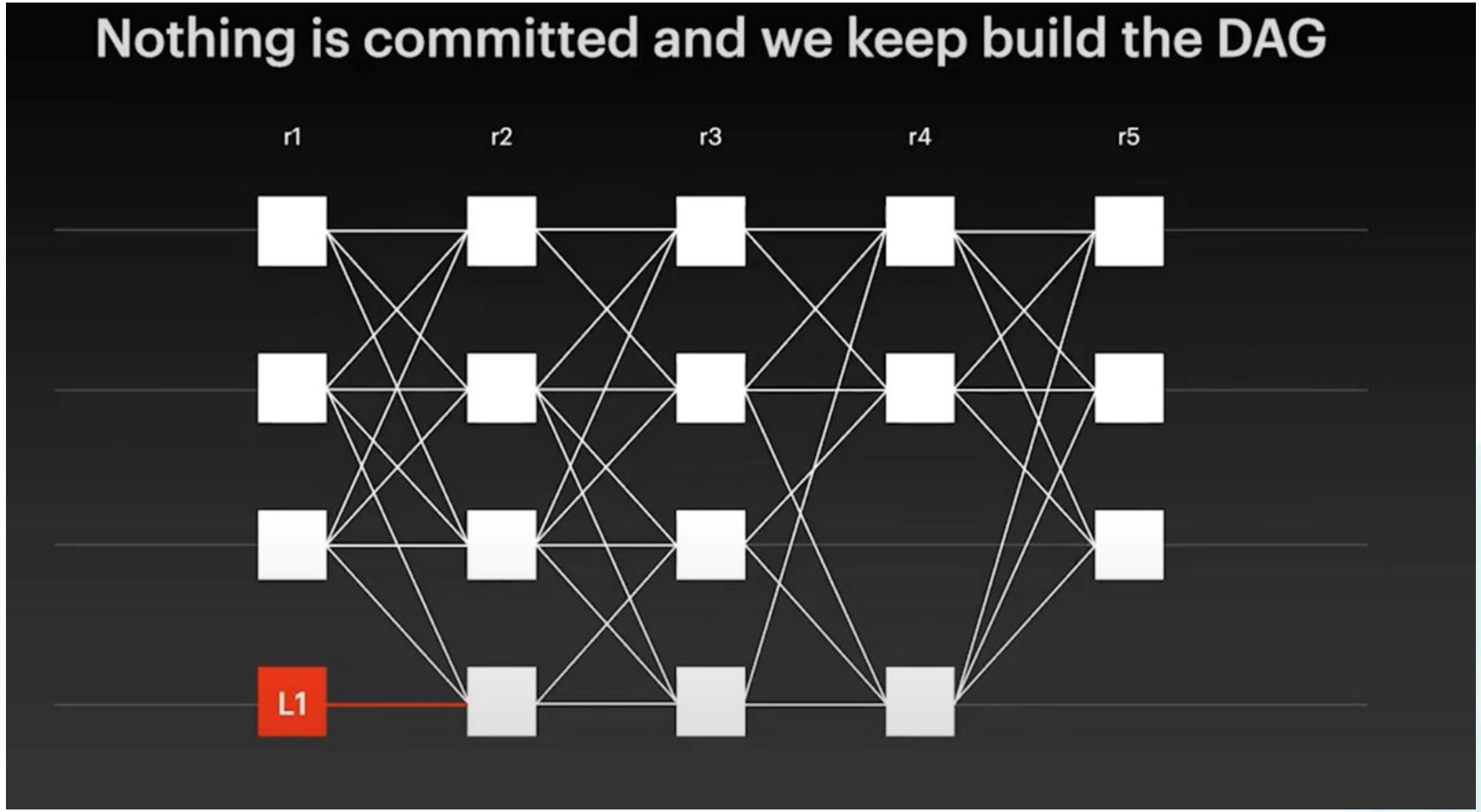


Tusk - asynchronous consensus

There are an insufficient number of blocks in round 2 (less than $f + 1$) that refer to/vote for $L1$ and thus $L1$ is not committed when round 3 is interpreted.

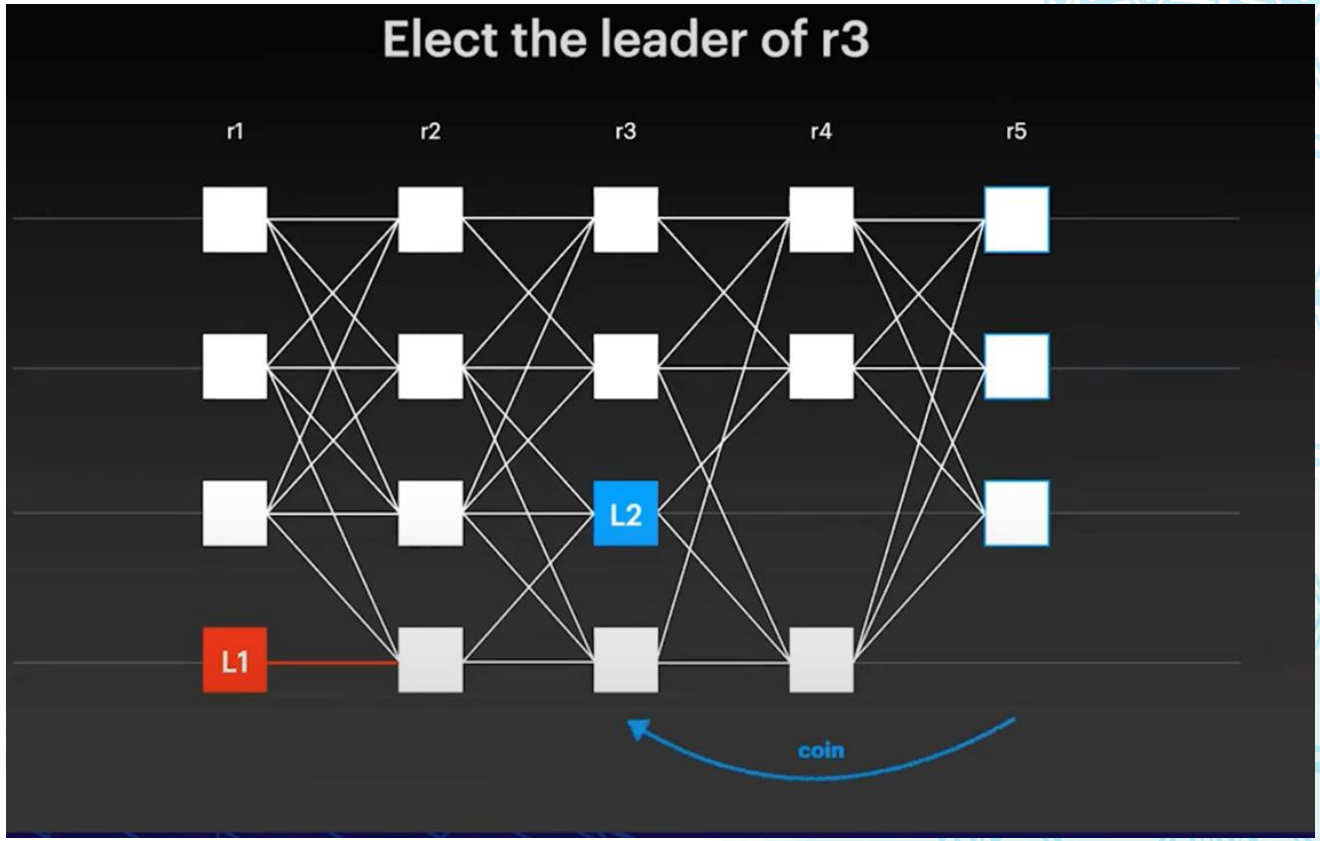


Tusk - asynchronous consensus



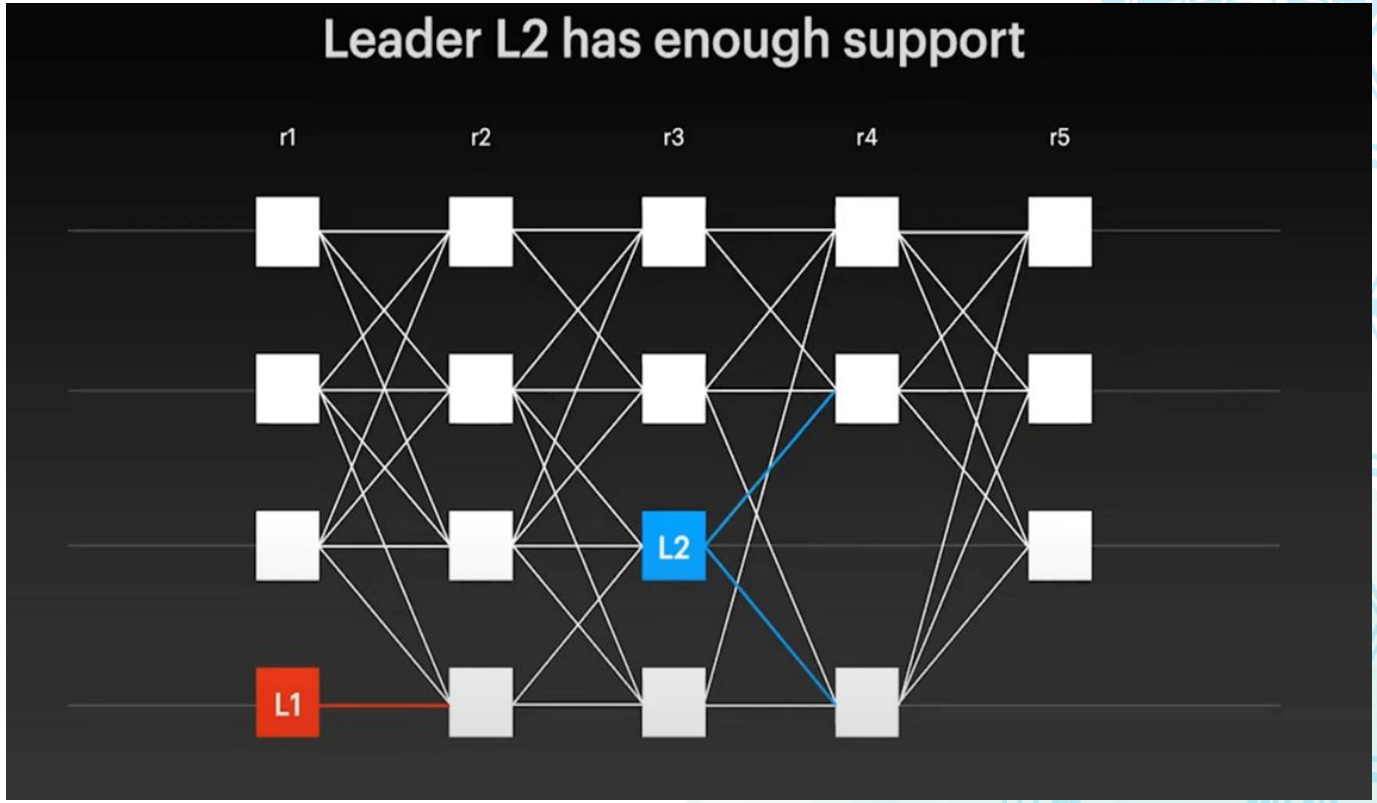
Tusk - asynchronous consensus

The elected leaders of waves 1 and 2 are determined at rounds 3 and 5, and we denote them $L1$ and $L2$, respectively



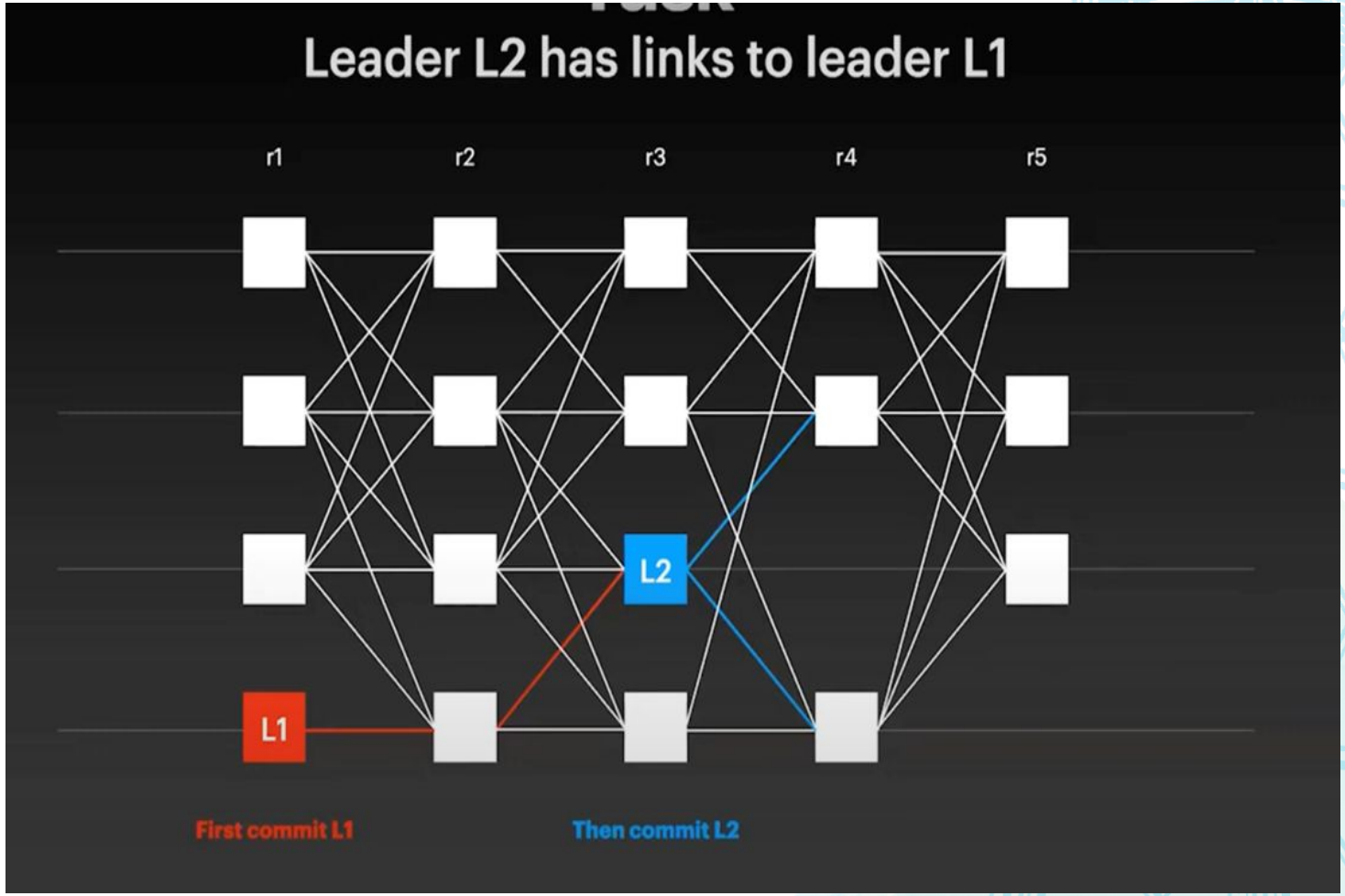
Tusk - asynchronous consensus

Since there are $f + 1 = 2$ blocks in round 4 that refer to $L2$, and as a result $L2$ is eventually committed.



Tusk - asynchronous consensus

Since there is a path between $L2$ and $L1$, $L1$ is ordered before $L2$. Meaning that the sub-DAG causally dependent on $L1$ is ordered first (by some deterministic rule), and then the same rule is applied to the sub-DAG causally dependent on $L2$.



Tusk - asynchronous consensus

So whenever we have enough support and Commit can take place, then a lot of blocks are committed altogether(Sub-DAGs) and not just a single block, this way we are able to commit a gigantic number of transactions all at once.