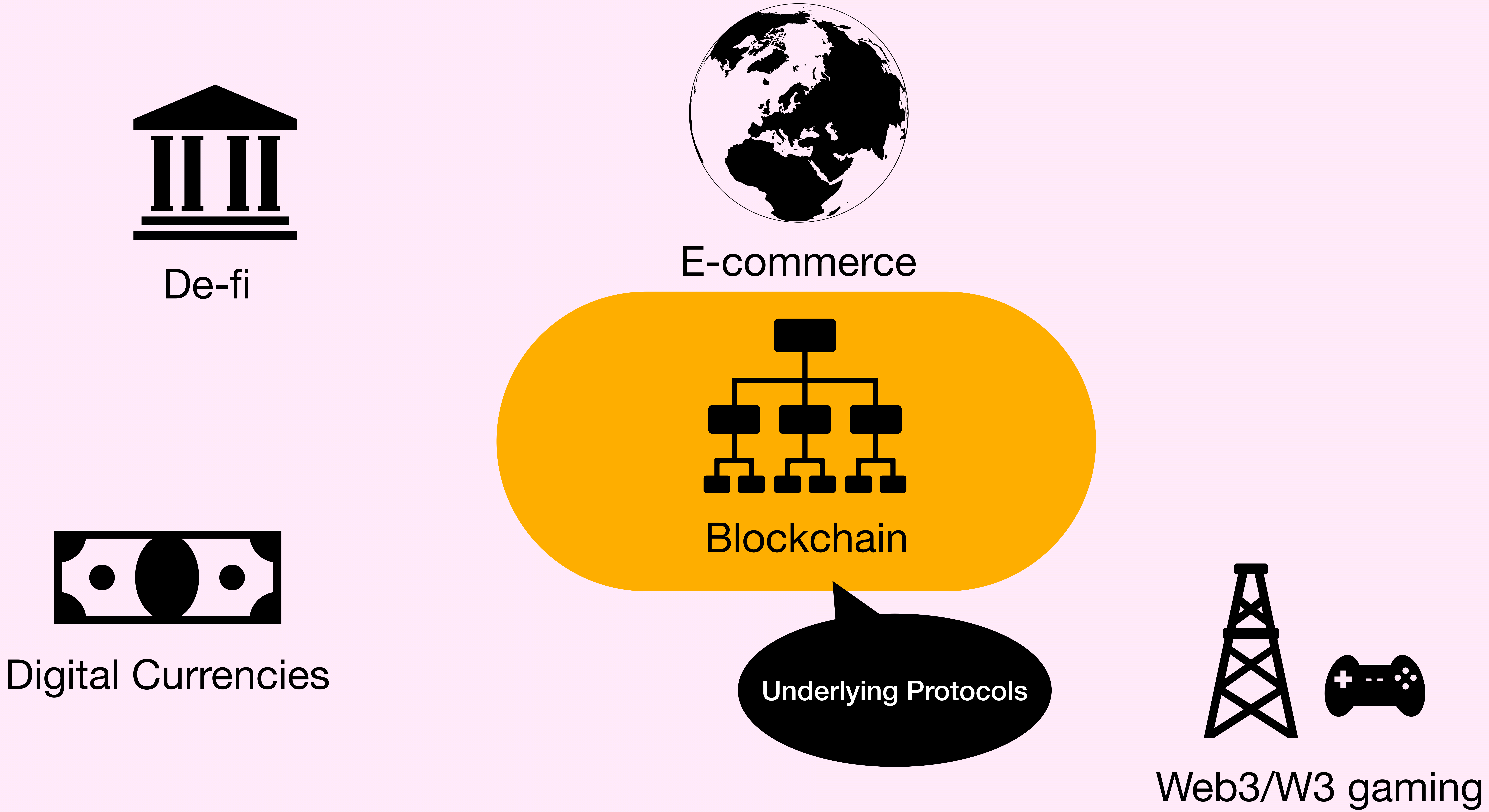
HammerHead

Leader Reputation for Dynamic Scheduling

Giorgos Tsimos, Anastasios Kichidis, Alberto Sonnino, Lefteris Kokoris-Kogias


De-fi
E-commerce
Blockchain
Underlying Protocols
Digital Currencies
Web3/W3 gaming

Byzantine Atomic Broadcast

- n parties, f byzantine (malicious)
- Parties send messages during several rounds
- One designated sender per message sent
- Sender wants to send its input value to all

Byzantine Atomic Broadcast

1. **Agreement**: All good nodes will get the same messages
2. **Integrity**: Every good node gets at most one message per round/sender
3. **Validity**: If a good node sends a message, all good nodes get it
4. **Total Order**: All good nodes get messages in the same order

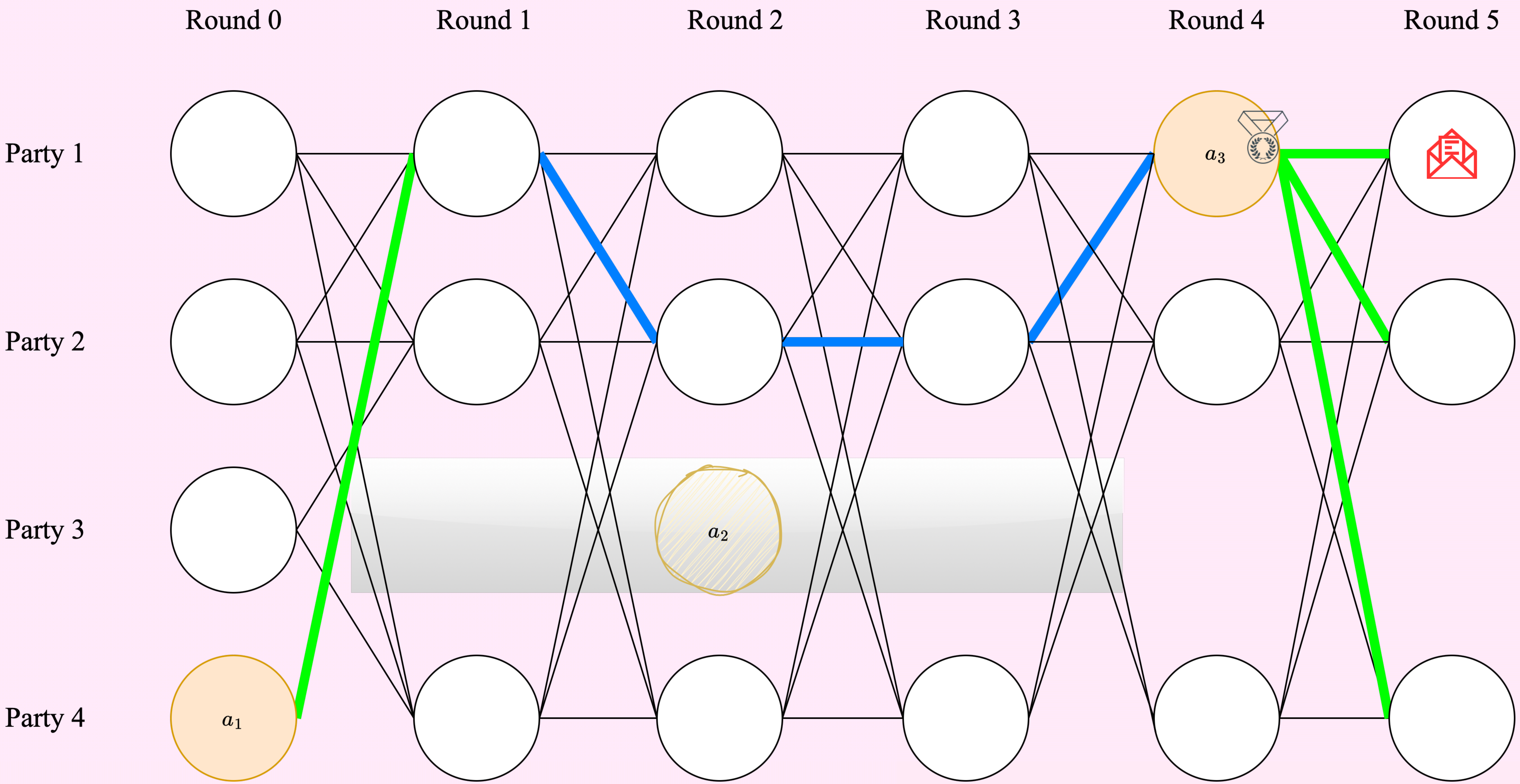
Even with one node sending everything, properties are satisfied!

Fairness and Crash Faults

- Optimally, we would like every node to add proportionally equally many blocks
- Fairness
- **BUT**
- If previously good nodes crash and are chosen as leaders, they cause long latency/low throughput
- Can we achieve fairness and avoid crashed leaders?

DAG consensus

- Consensus protocols with an underlying DAG structure for message dependencies
  - Line of work: Bullshark, Narwhal-Tusk, Shoal, **HammerHead**…
  - Adopted by systems like Sui
  - Allow for blockchains with improved throughput in comparison


Round 0
Round 1
Round 2
Round 3
Round 4
Round 5
Party 1
Party 2
Party 3
Party 4
a_1
a_2
a_3

Chained vs DAG Consensus

- For chained consensus, Carousel [FC'22] provided a solution with dynamic leader election
  - Achieves **Leader Utilization: bounds the amount of faulty leaders**
  - And **Chain Quality: ~ %committed blocks ≡ %good nodes**

- How about DAG-based consensus protocols?

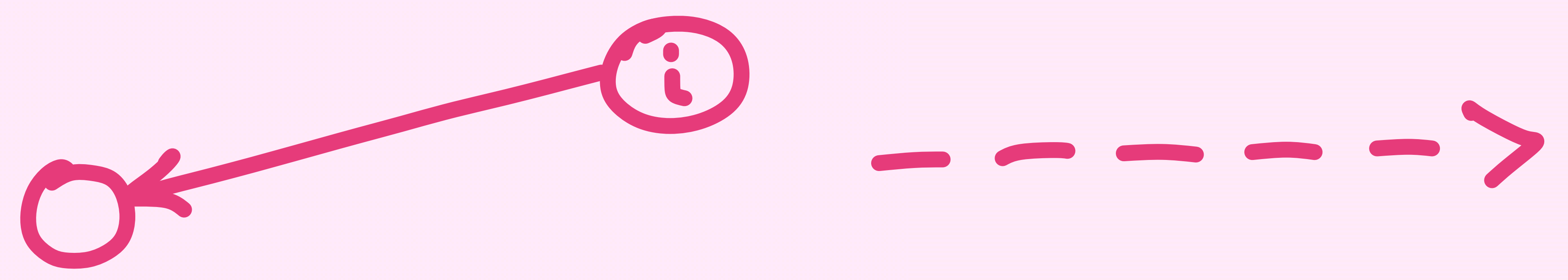
In this talk

- We design a **DAG-based BAB protocol** on top of Bullshark, with **dynamic leader election** via **reputation**, that achieves **Leader Utilization**
- We show
  - How to **instantiate leader reputation** in DAG consensus
  - How to **utilize leader reputation** for BAB protocols with:
    - **Leader Utilization**
    - Improved practical **crash-recovery**

Reputation

- Each node keeps track of a score for every other node
- The score depends on events upon the DAG
  - It captures the "responsiveness" of nodes during current epoch
  - If P_i votes in round r for the proposal of the leader of round $r-1$, then set scores$(P_i) += 1$

DAG view of party P_j

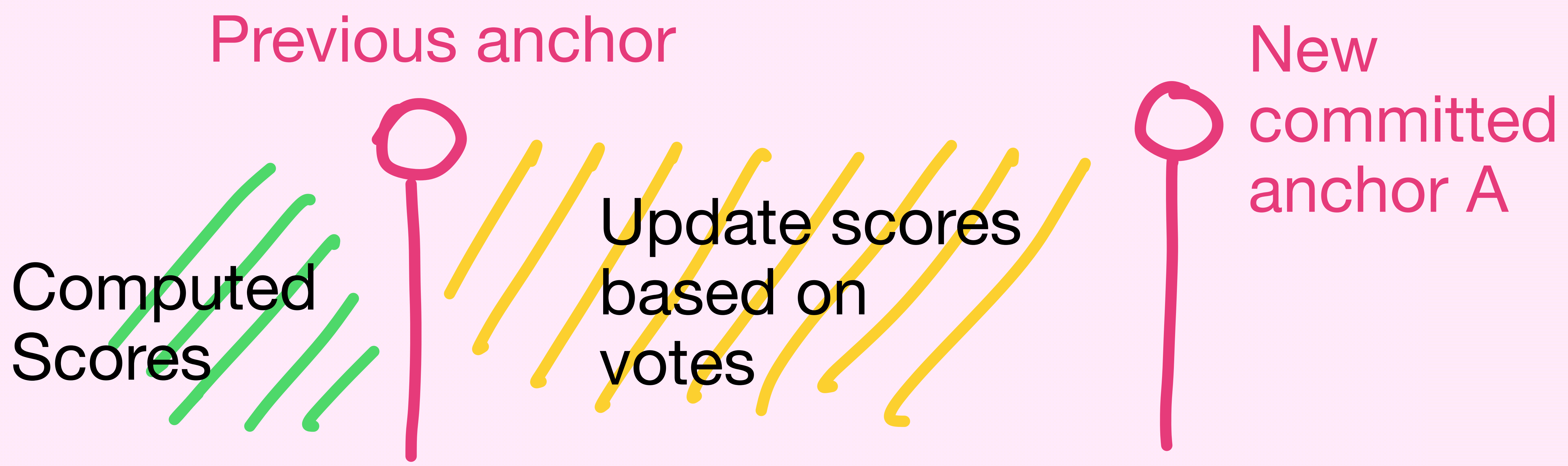


Set scores$_j(i)$ += 1

Round $r-1$ Round r

Reputation

- View of different parties might differ
- Remember **DAG property**: If vertex u is added to DAG_i , then its entire causal history is also in DAG_i
    - So, when committing anchor A, recompute scores based on the new subdag history up to A (which is committed and fixed).



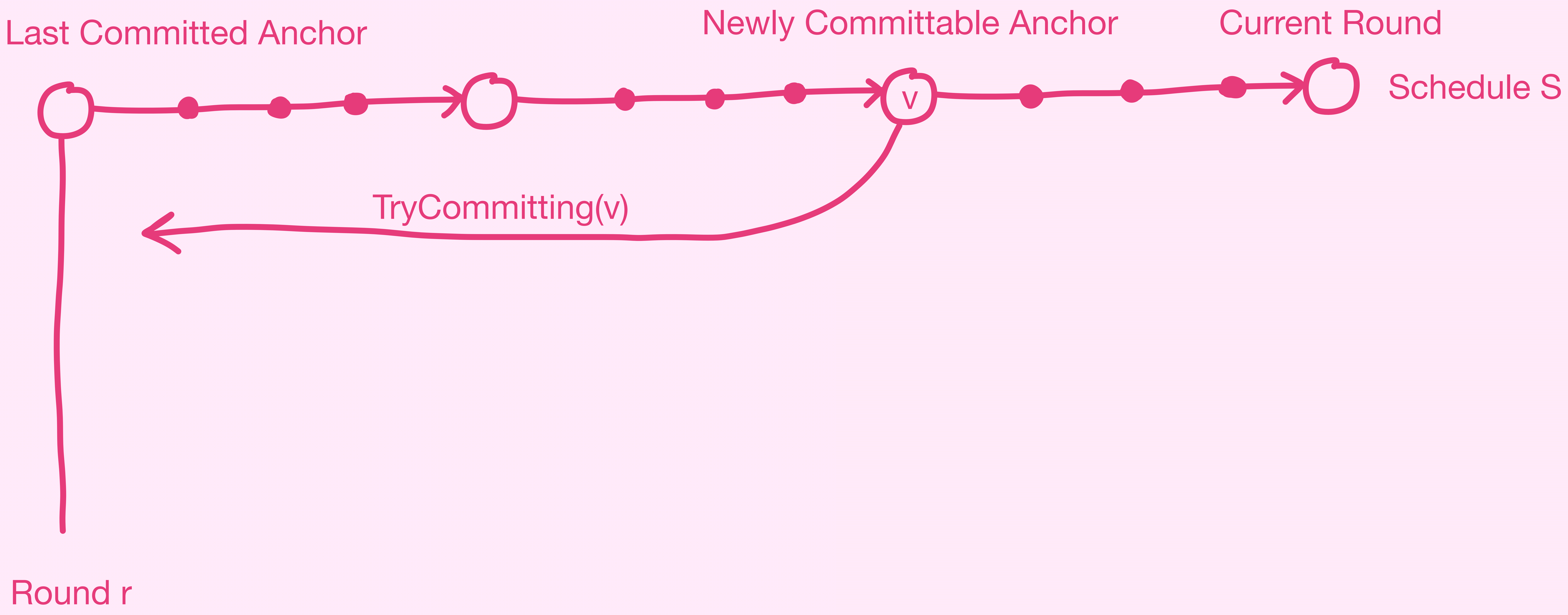
Leader Selection

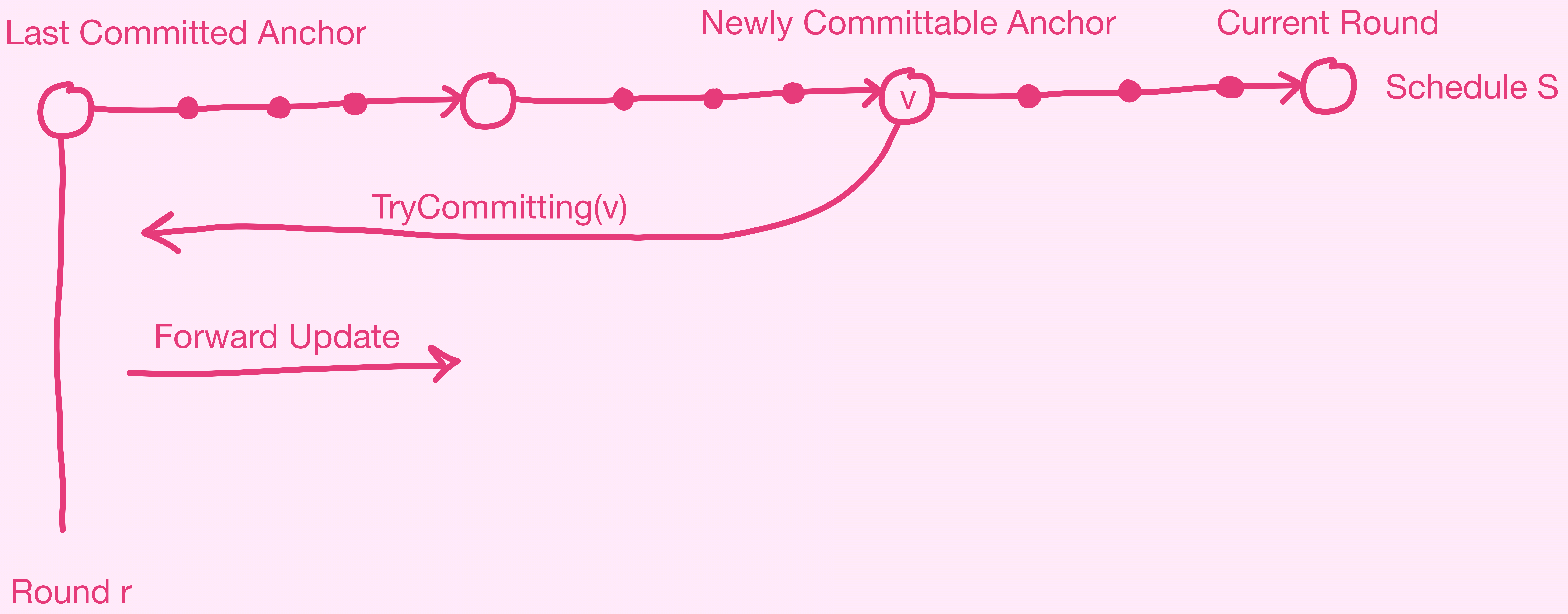
- We want to disallow recently unresponsive nodes from leaders
- We utilize the reputation scores
- If a node has low score for past epoch, it was unresponsive/crashed
- Disallow node from being elected for the next epoch
- **Leader set** for epoch: $\geq 2f+1$ **most responsive** nodes of previous epoch
    - **based on reputation**
    - The $\leq f$ worst are disallowed
    - the rest are elected per round with some (stake-based) probability

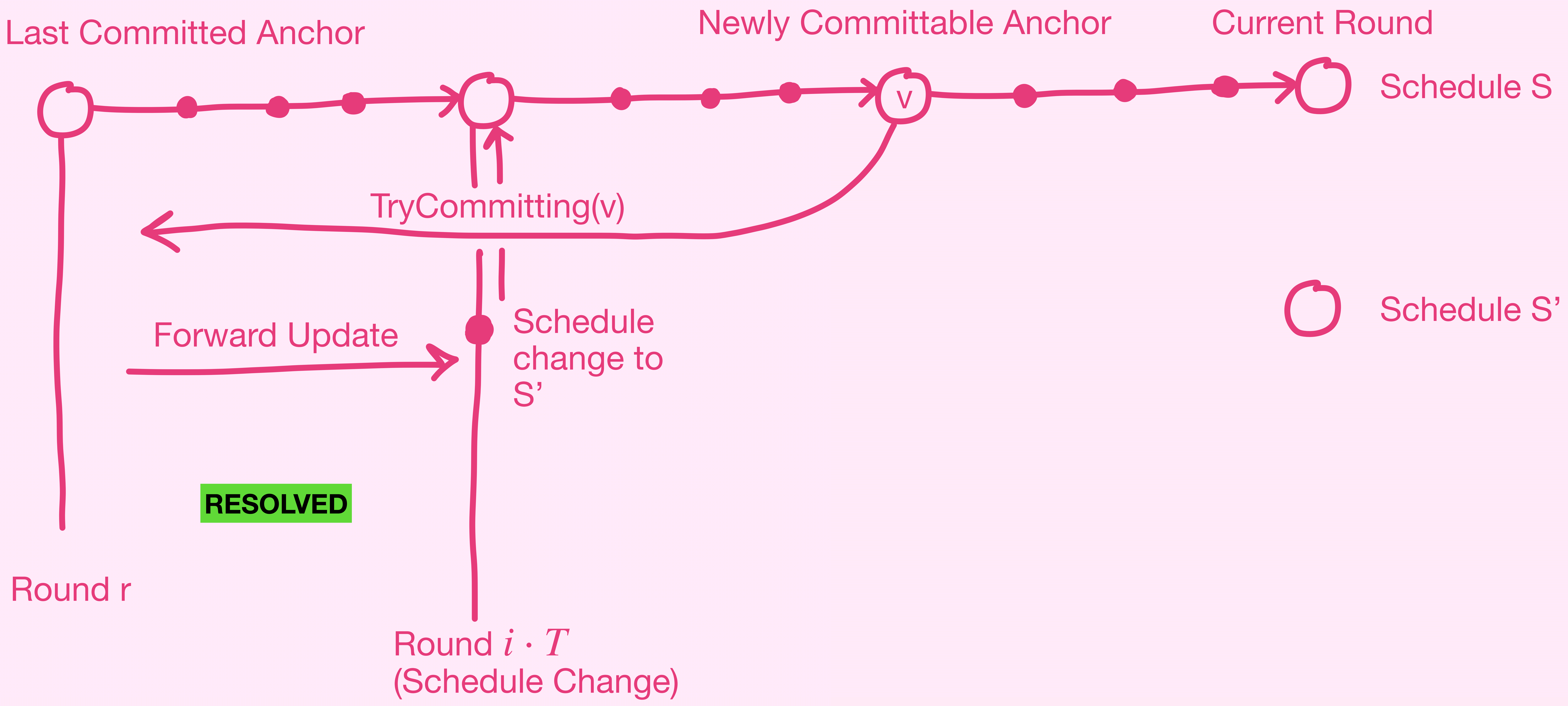
Schedule updates

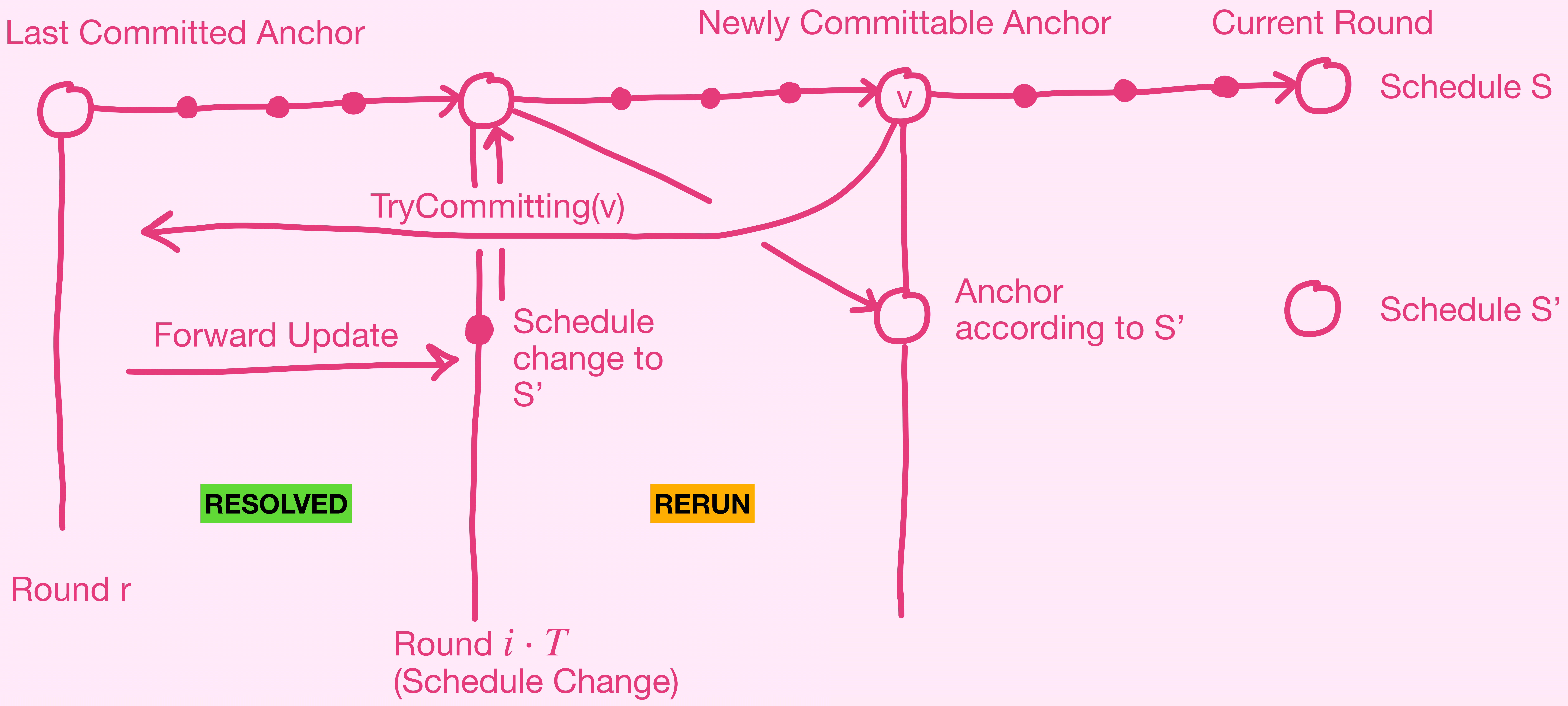
- Schedule-change frequency: **T**
- **activeSchedule :** contains auxiliary info related to the current schedule, i.e.
  - **initialRound**
  - **LeaderSet :** information regarding the set of active leaders for the epoch, i.e. the set of good leaders and their scores (for weighted draw).
- **scores :** DS that maintains a score value for each node. When a schedule change is to occur, scores are updated according to the subDAG of that epoch.

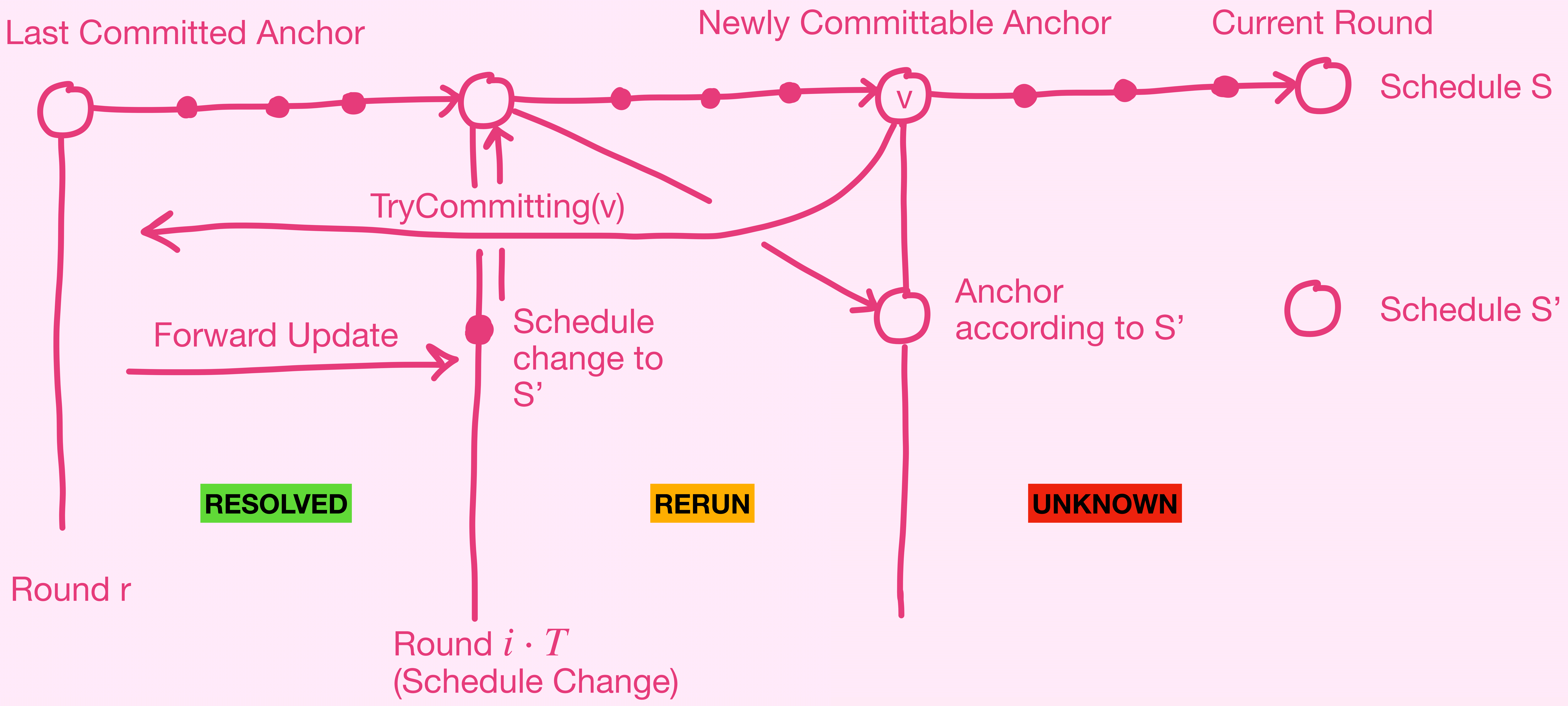
How about safety-liveness issues?


Last Committed Anchor
Newly Committable Anchor
Current Round
v
Schedule S
TryCommitting(v)
Round r


Last Committed Anchor
Newly Committable Anchor
Current Round
v
Schedule S
TryCommitting(v)
Forward Update
Round r


Last Committed Anchor
Newly Committable Anchor
Current Round
v
Schedule S
TryCommitting(v)
Schedule S'
Forward Update
Schedule change to S'
RESOLVED
Round r
Round $i \cdot T$ (Schedule Change)

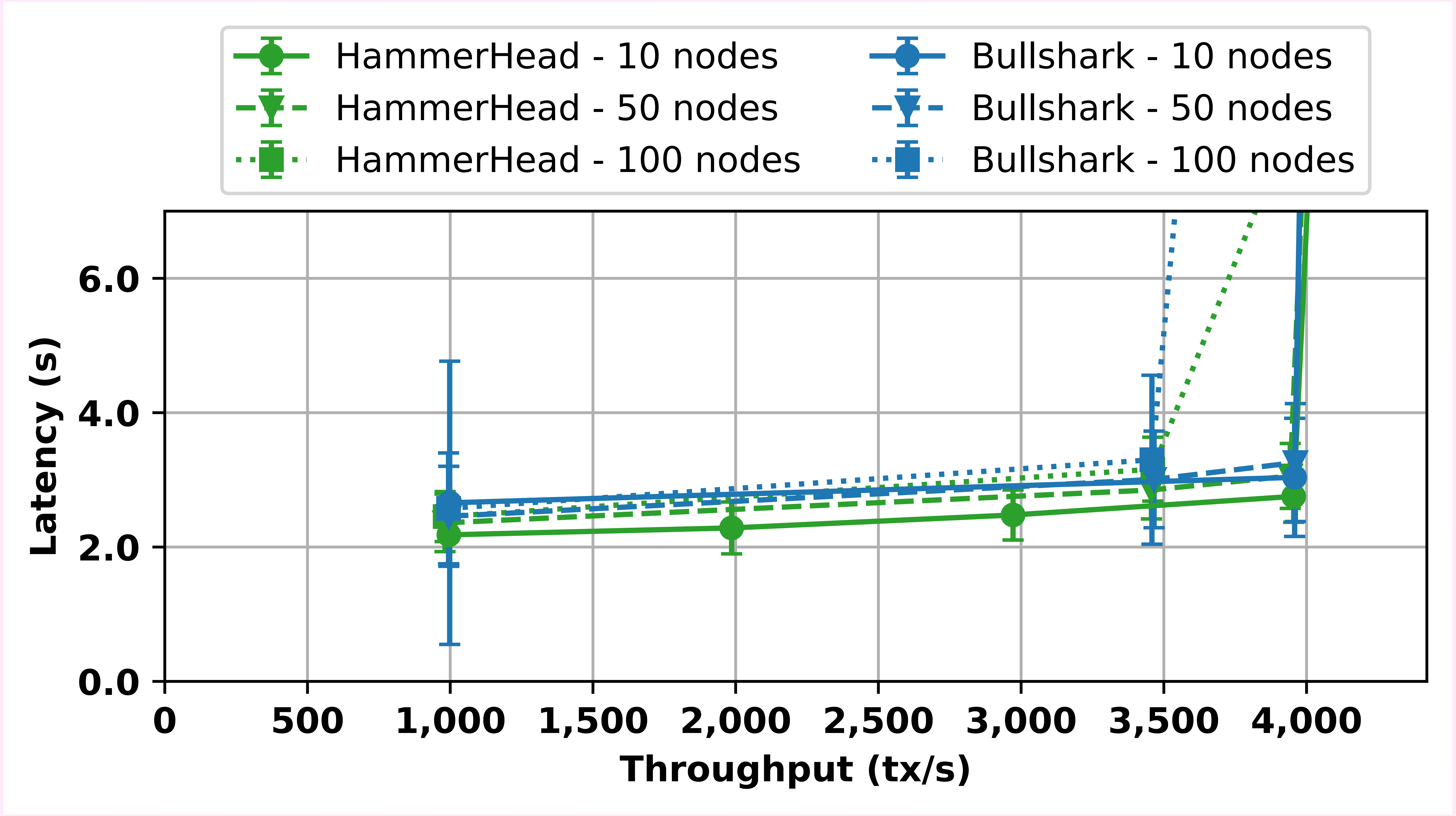

Last Committed Anchor
Newly Committable Anchor
Current Round
v
Schedule S
TryCommitting(v)
Anchor according to S'
Schedule S'
Forward Update
Schedule change to S'
RESOLVED
RERUN
Round r
Round $i \cdot T$ (Schedule Change)


Last Committed Anchor
Newly Committable Anchor
Current Round
v
Schedule S
TryCommitting(v)
Anchor
according to S'
Schedule S'
Forward Update
Schedule
change to
S'
RESOLVED
RERUN
UNKNOWN
Round r
Round $i \cdot T$
(Schedule Change)

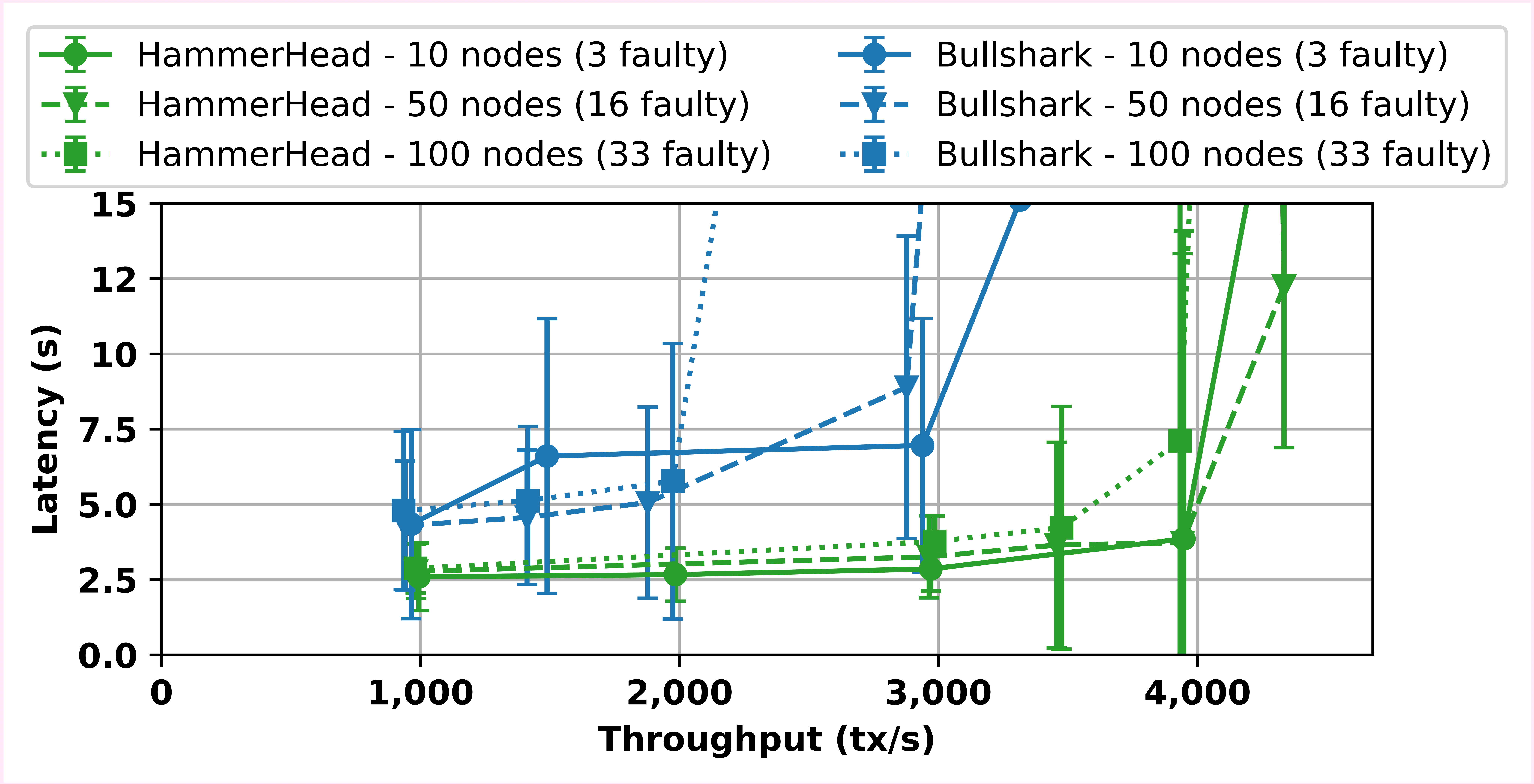
Implementation

- HammerHead is deployed in Sui mainnet (since v1.9.1)
    - Open-source: https://github.com/asonnino/sui/tree/hammerhead (commit 03c96a3)
- Tests show:
    - No throughput loss in ideal conditions (no faults)
    - Improved latency/throughput against crash faults
    - No persistent throughput loss when crash faults occur

Without crash faults



With crash faults



In this work

- We designed a **DAG-based BAB protocol** with **dynamic leader election** via **reputation**, that achieves **Leader Utilization**
- We **instantiated leader reputation** in DAG consensus
- We **utilized leader reputation** for BAB protocols with **Leader Utilization** and improved practical **crash-recovery**
- For more details check out our paper: https://arxiv.org/abs/2309.12713

Questions?

Contact us: tsimos at umd dot edu