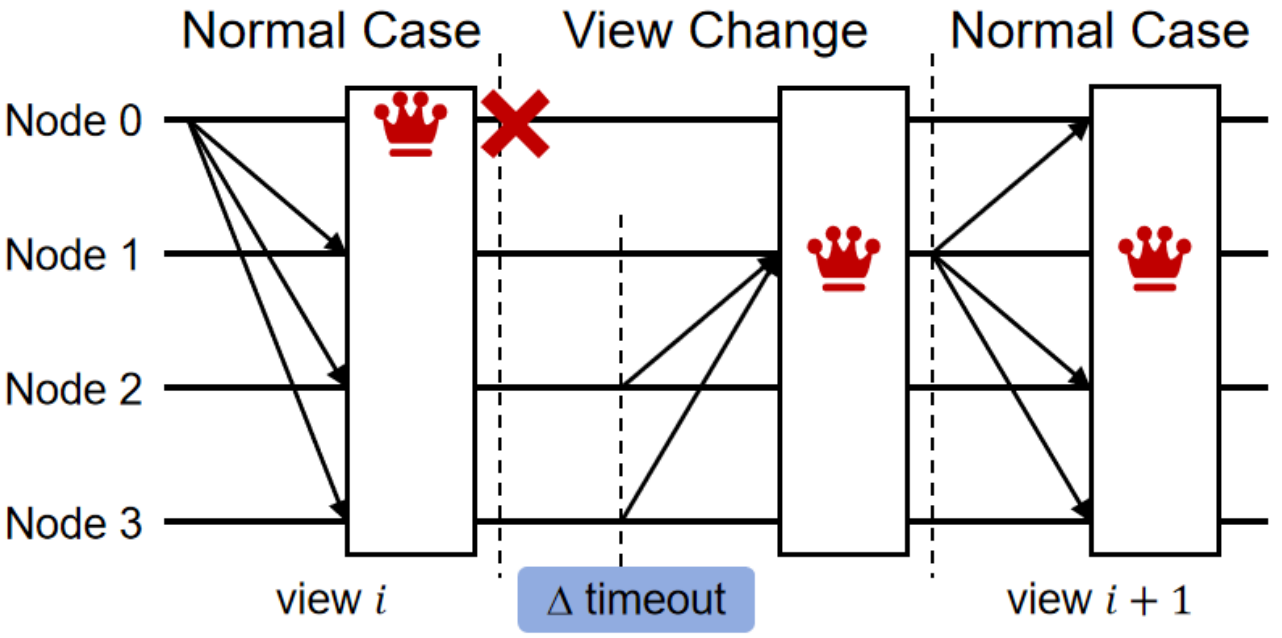
Chitu: Avoiding Unnecessary Fallback in Byzantine Consensus

Rongji Huang[1], Xiangzhe Wang[1], Xiaofeng Yan[1],
Lei Fan[1], Guangtao Xue[1,2], Shengyun Liu[1,2]

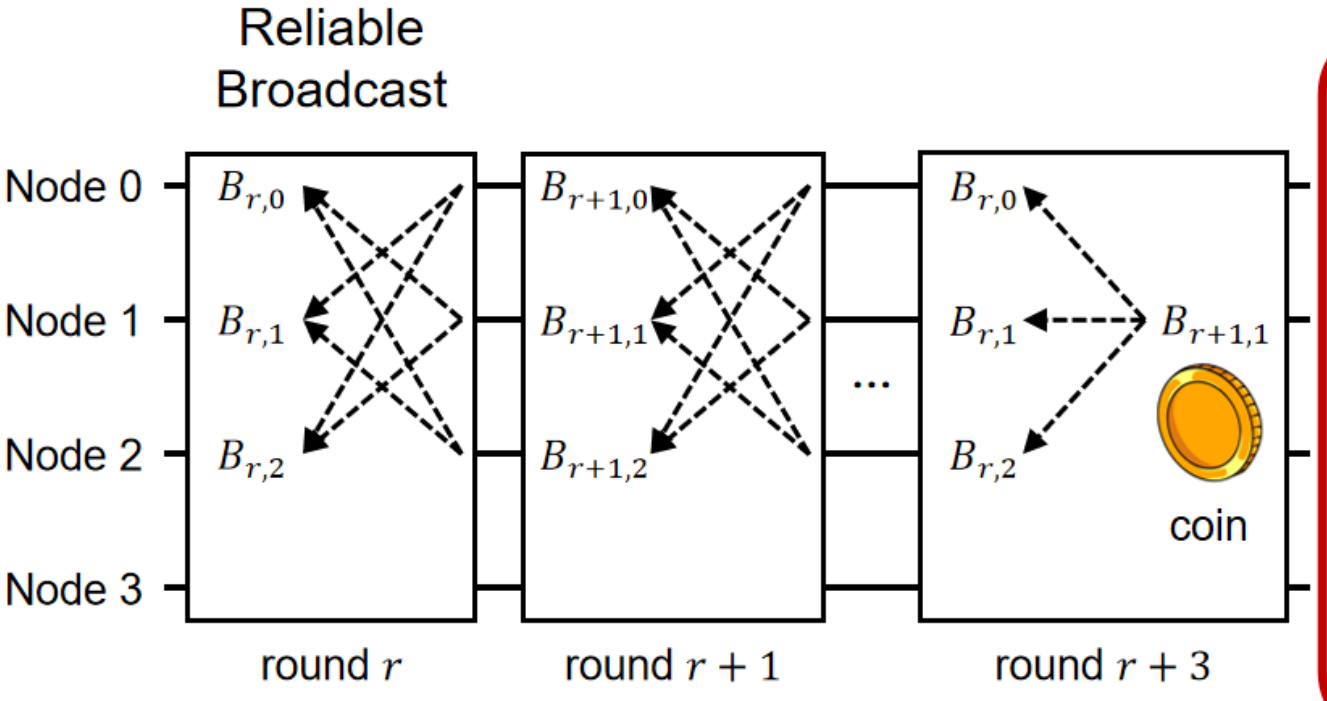
[1]Shanghai Jiao Tong University [2]Shanghai Key Laboratory of Trusted Data Circulation, Governance and Web3

Partially synchronous protocols



- Leader election relying on synchrony assumption
- Unavailibility during view change
- Liveness forfeited when assumption does not hold (commonly in WAN)

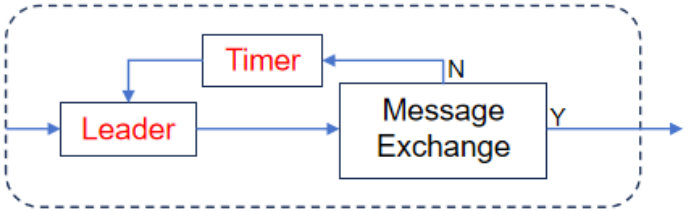
Asynchronous protocols



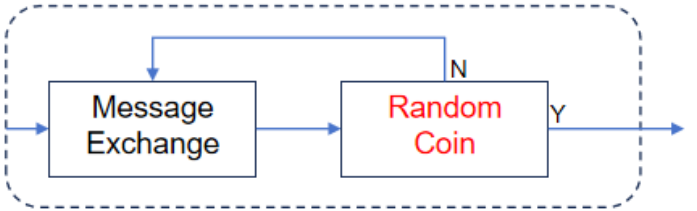
- Non-faulty nodes converge by coin with probability 1
- DAG-based protocols:
  - use coin to select a leader
  - commit blocks by edges in DAG

First principles of consensus

Most partially synchronous protocols



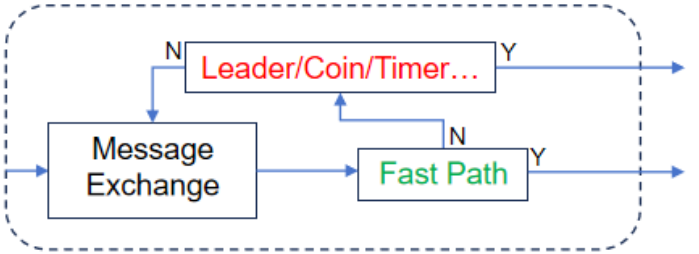
Most asynchronous protocols



Neither any special role (or the maximum network delay Δ) nor randomization is intrinsic to the consensus problem

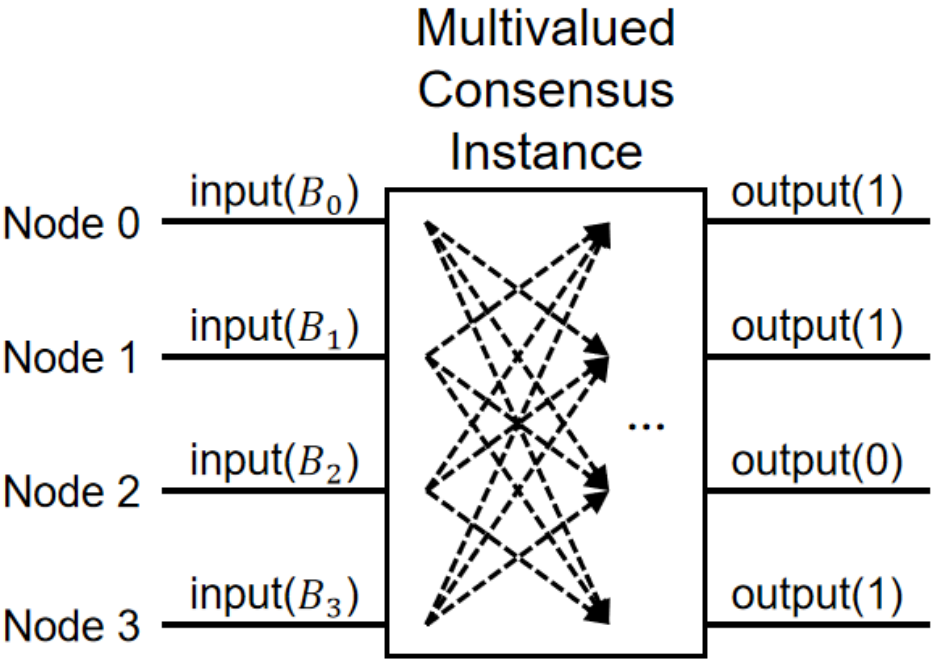
The framework we study

Inspired by MyTumbler (SOSP '23) and Red Belly (S&P '21)



First principles of consensus

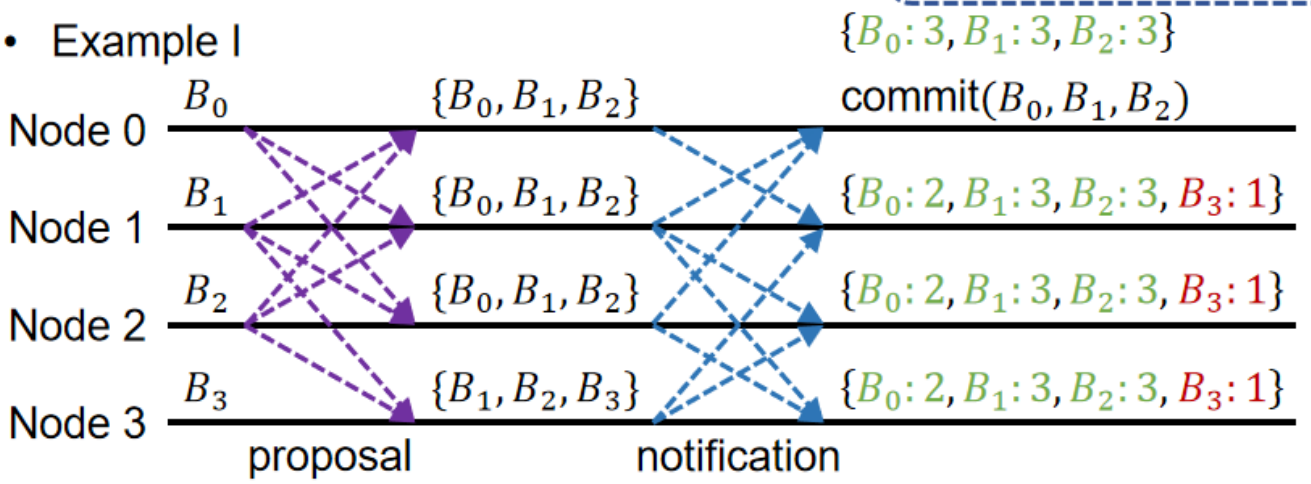
- For any multivalued consensus problem



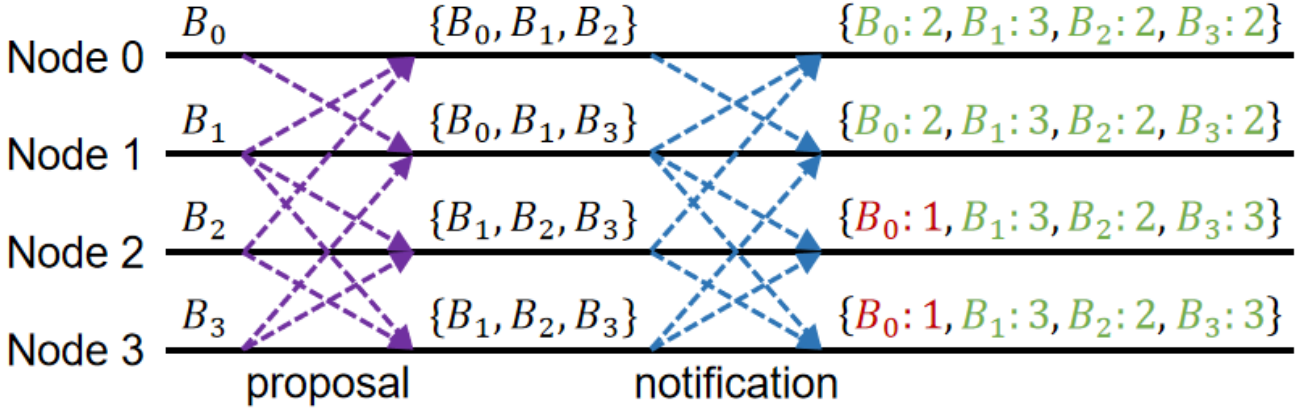
A concrete example

B_i candicate to be committed
(notified by $\geq f + 1$ nodes)
B_i aborted proposal
(notified by $\leq f$ nodes)

- Example I



- Example II

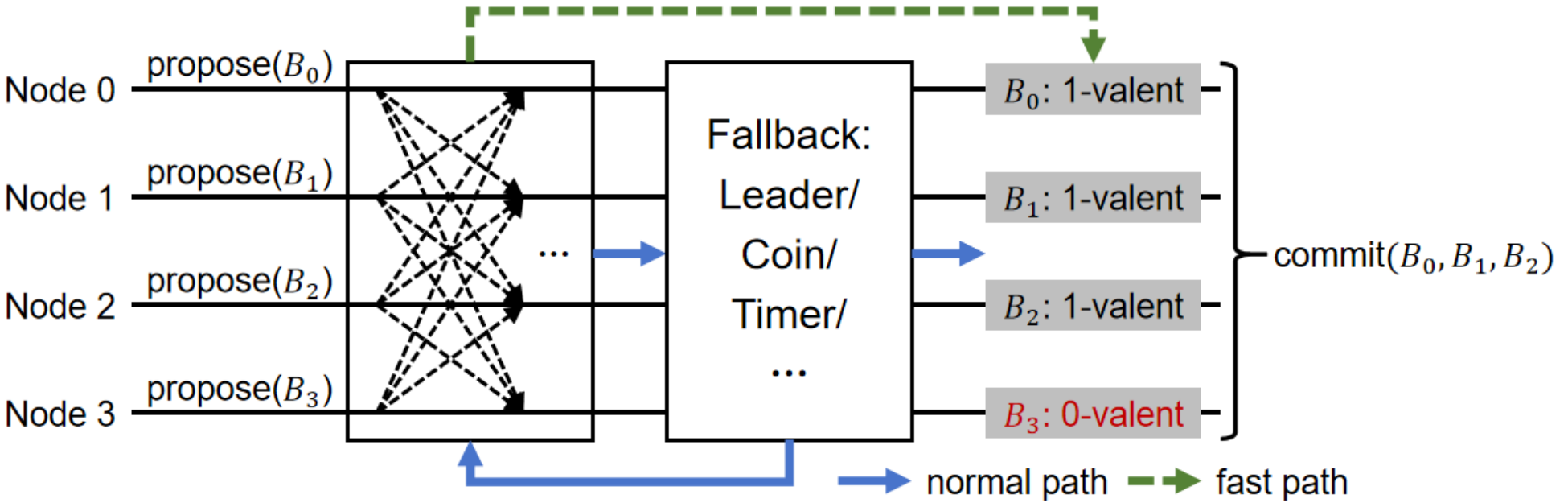


Each proposal B is decided

- **1-valent**, if notified by at least $n - f$ nodes;
- **0-valent**, if not notified by at least $n - f$ nodes;
- **bivalent**, if neither of the above conditions is met.

⚠ A fallback mechanism is needed when fast path fails due to bivalent proposals

The Fair-Fallback framework



- Fast Path + Random Coins → Chitu protocol
- Fast Path + Δ Timer → Applying fast path to Bullshark (CCS '22)
- Fast Path + Pre-selected Leaders → Asynchronous protocol under fair schedulers

Generalization and challenges

- To instantiate the fast path
  - CON_C: the conditions for the fast path to commit a (1-valent) proposal — Ex I. B_0, B_1, B_2
  - CON_A: the conditions for the fast path to abort a (0-valent) proposal — Ex I. B_3
  - CON_F: the conditions for the fallback to commit a proposal — Ex II. B_0

① CON_C satisfied ⇒ CON_F satisfied ② CON_A satisfied ⇒ CON_F not satisfied

- Challenges:
  - How to ensure safety properties under Byzantine faults ? **Byzantine Reliable Broadcast (BRB)**
  - How to efficiently integrate two paths with no switch and guarantee each path commits the same set of proposals ? **Chitu protocol**

Overview of Chitu

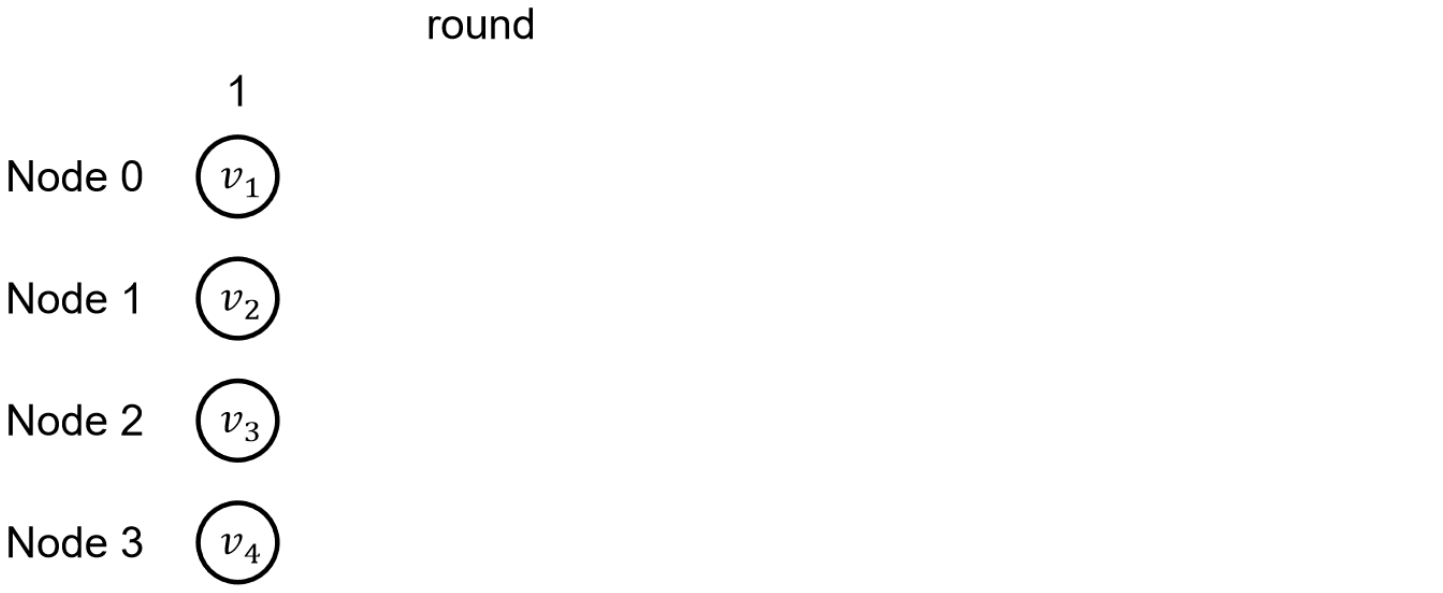
- Dynamically construct a DAG in a round-by-round manner by BRB
  - Inspired by Tusk (EuroSys ’22) and DAG-Rider (PODC ’21)
- Commit rules:
  - Fast path: vertices of round $r+1$ help decide round r **leaderlessly**
  - Normal path: a selected leader of round $r' > r+1$ helps decide round r
- Adaptive wait mechanism:
  - Increase the chance of fast-path univalent commits

Commit rules: Fast path

- A vertex v in round r is
  - A proposal of round r
  - A notification of the proposals observed by v

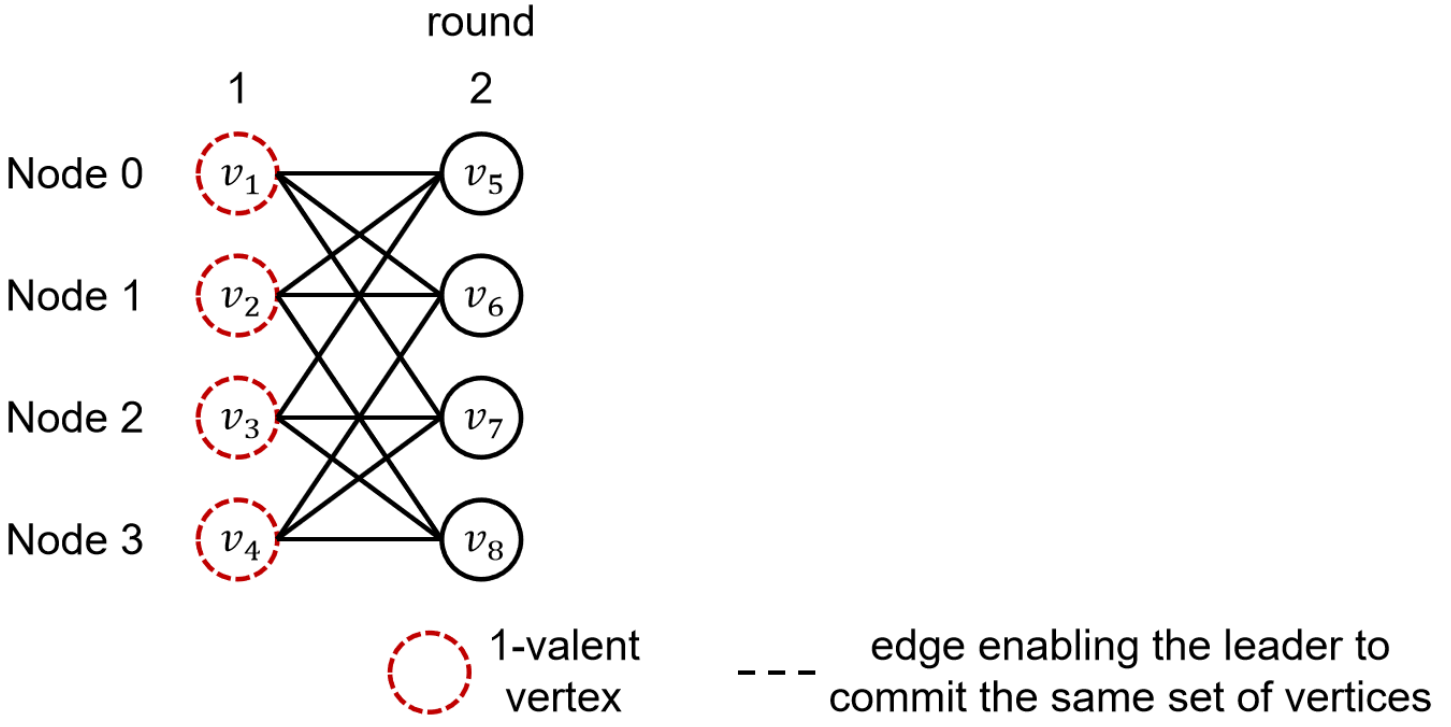
Commit rules: Fast path

- A vertex v in round r is
  - A proposal of round r
  - A notification of the proposals observed by v
- Case I: all 1-valent



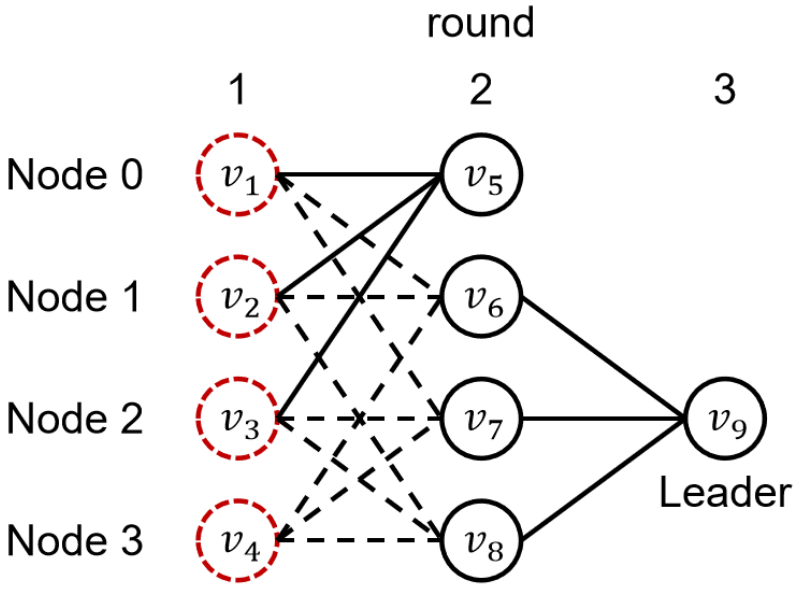
Commit rules: Fast path

- A vertex v in round r is
  - A proposal of round r
  - A notification of the proposals observed by v
- Case I: all 1-valent



Commit rules: Fast path

- A vertex v in round r is
  - A proposal of round r
  - A notification of the proposals observed by v
- Case I: all 1-valent



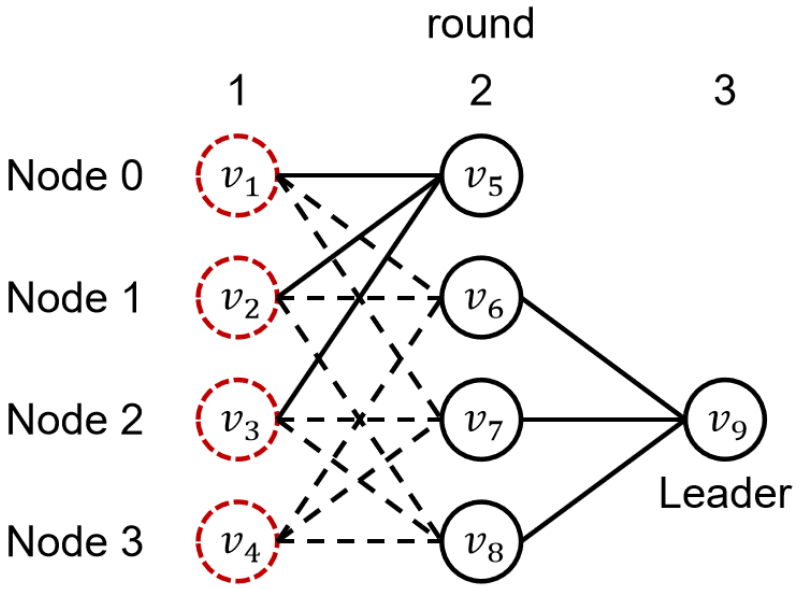
1-valent vertex

edge enabling the leader to commit the same set of vertices

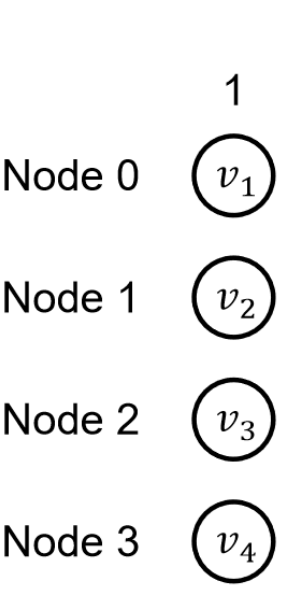
Commit rules: Fast path

- A vertex v in round r is
  - A proposal of round r
  - A notification of the proposals observed by v

- Case I: all 1-valent



- Case II: some 0-valent (v_1)



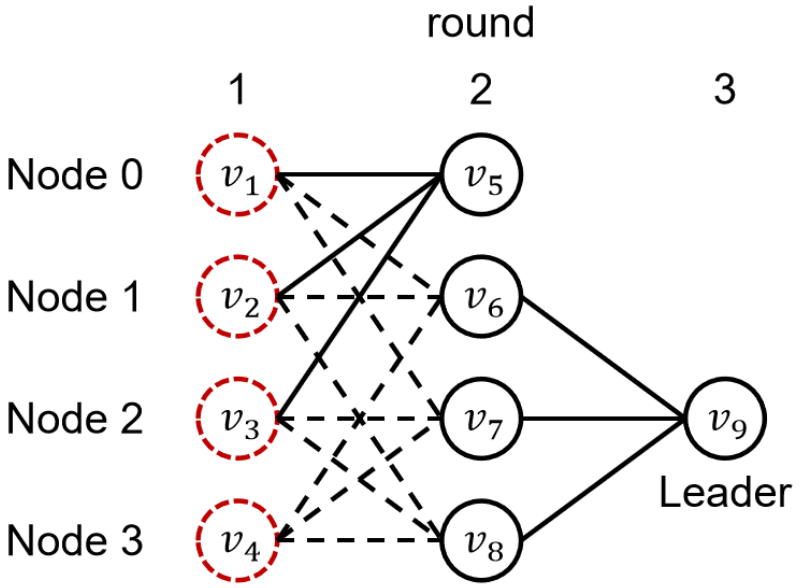
1-valent vertex

edge enabling the leader to commit the same set of vertices

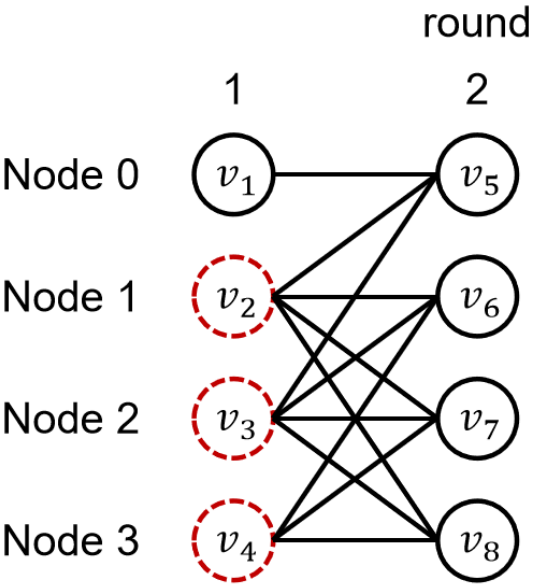
Commit rules: Fast path

- A vertex v in round r is
  - A proposal of round r
  - A notification of the proposals observed by v

- Case I: all 1-valent



- Case II: some 0-valent (v_1)



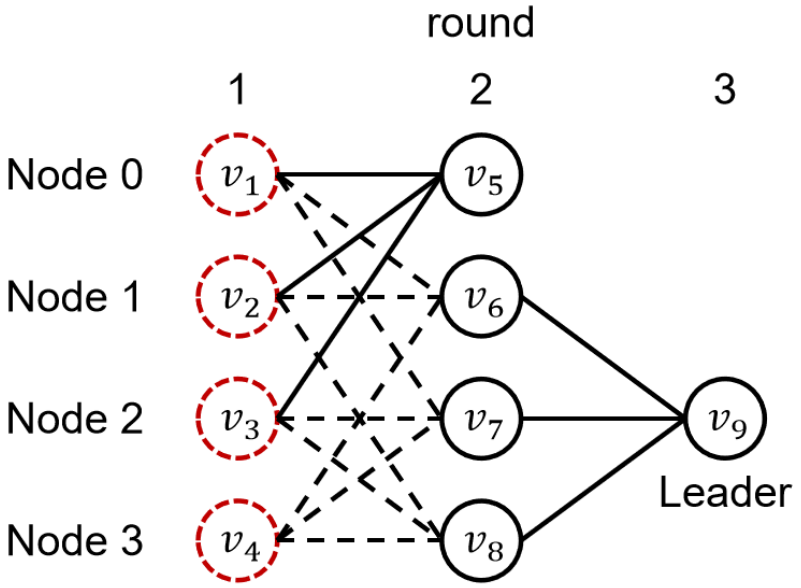
1-valent vertex

edge enabling the leader to commit the same set of vertices

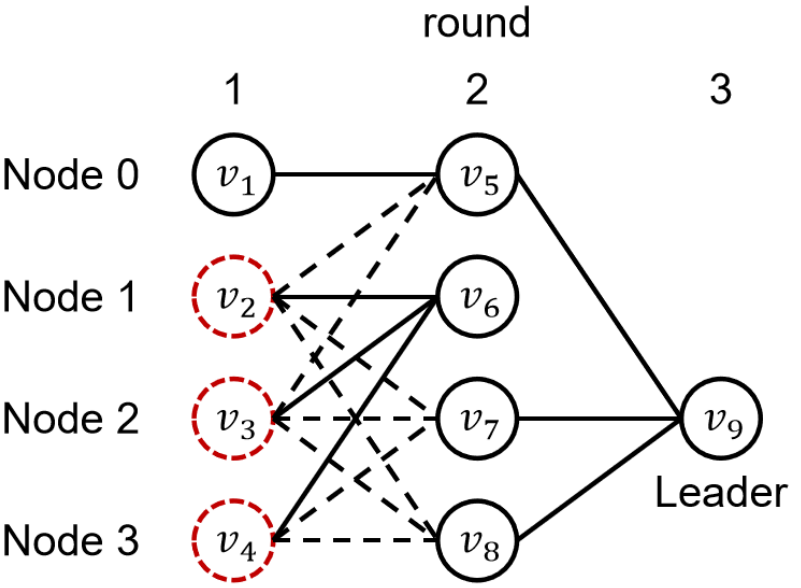
Commit rules: Fast path

- A vertex v in round r is
  - A proposal of round r
  - A notification of the proposals observed by v

- Case I: all 1-valent



- Case II: some 0-valent (v_1)



1-valent vertex

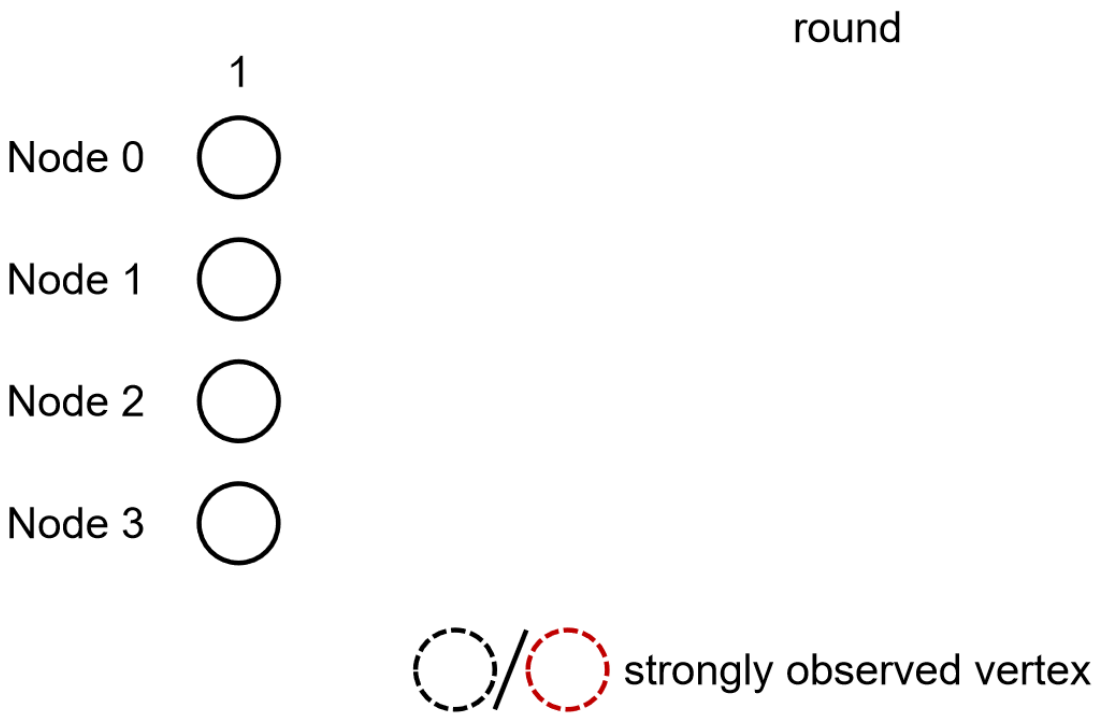
edge enabling the leader to commit the same set of vertices

Commit rules: Normal path

- Select leaders by random coins (like Tusk)
- Leader vertices recursively decide preceding rounds

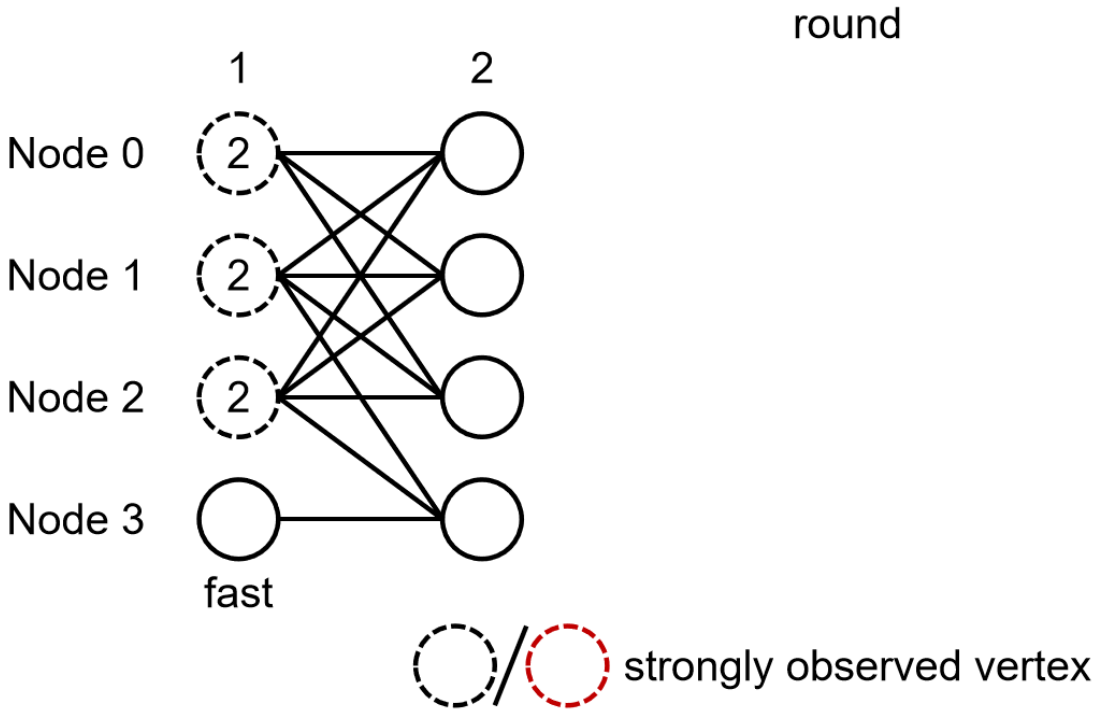
Commit rules: Normal path

- Select leaders by random coins (like Tusk)
- Leader vertices recursively decide preceding rounds



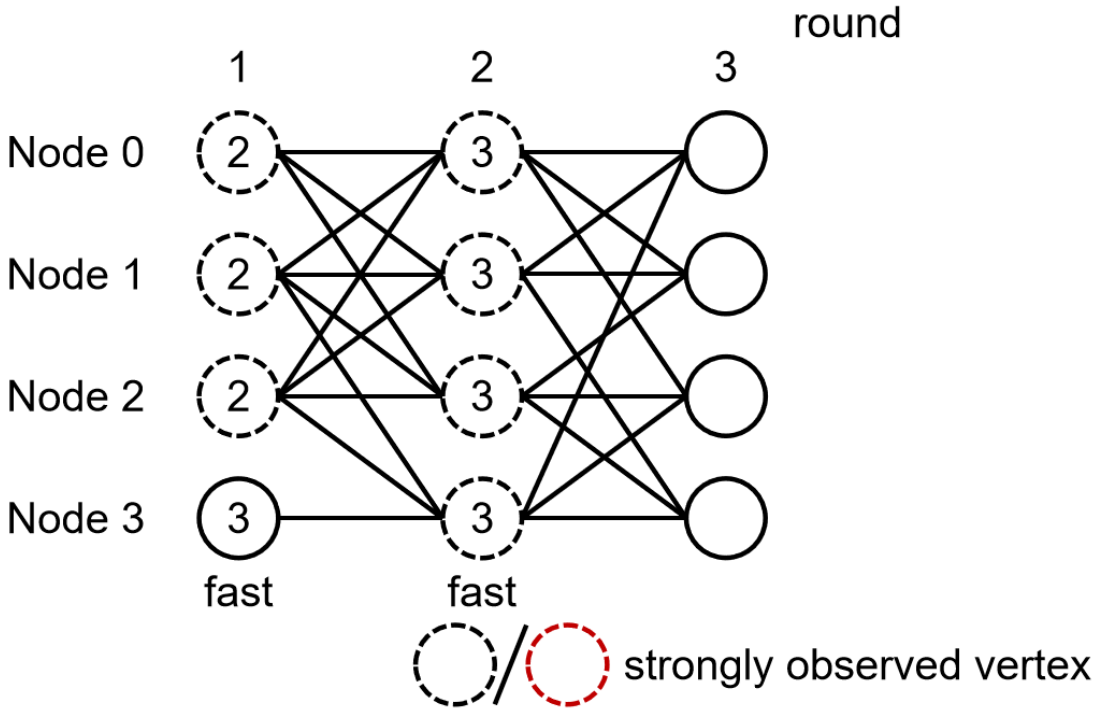
Commit rules: Normal path

- Select leaders by random coins (like Tusk)
- Leader vertices recursively decide preceding rounds



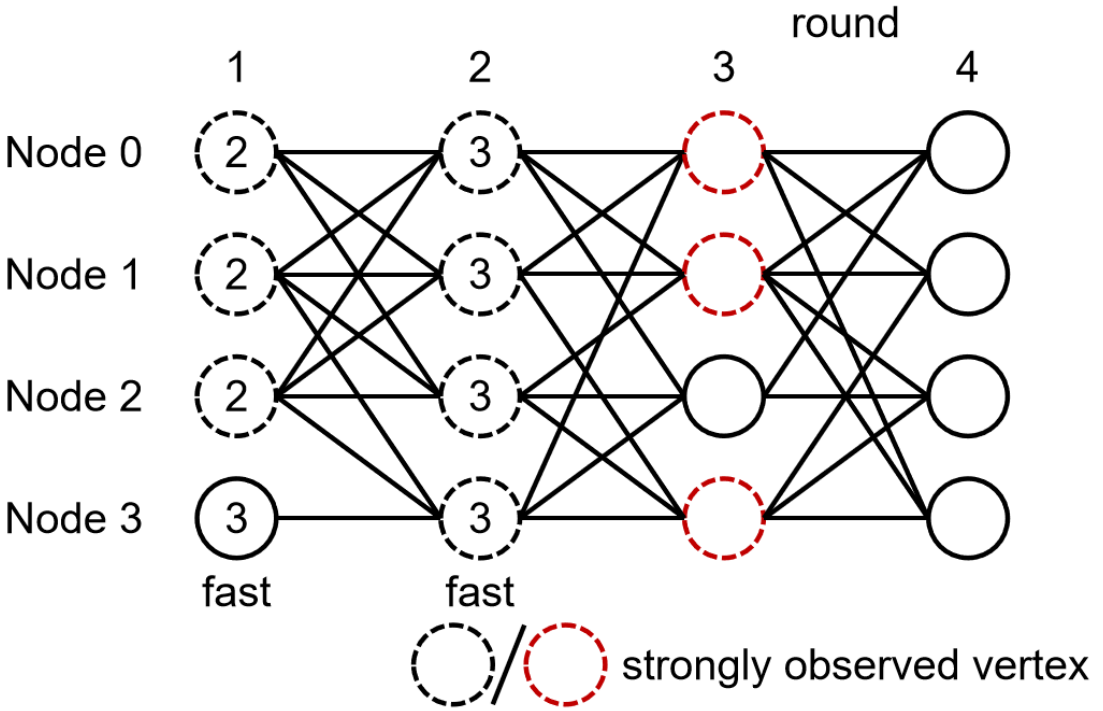
Commit rules: Normal path

- Select leaders by random coins (like Tusk)
- Leader vertices recursively decide preceding rounds



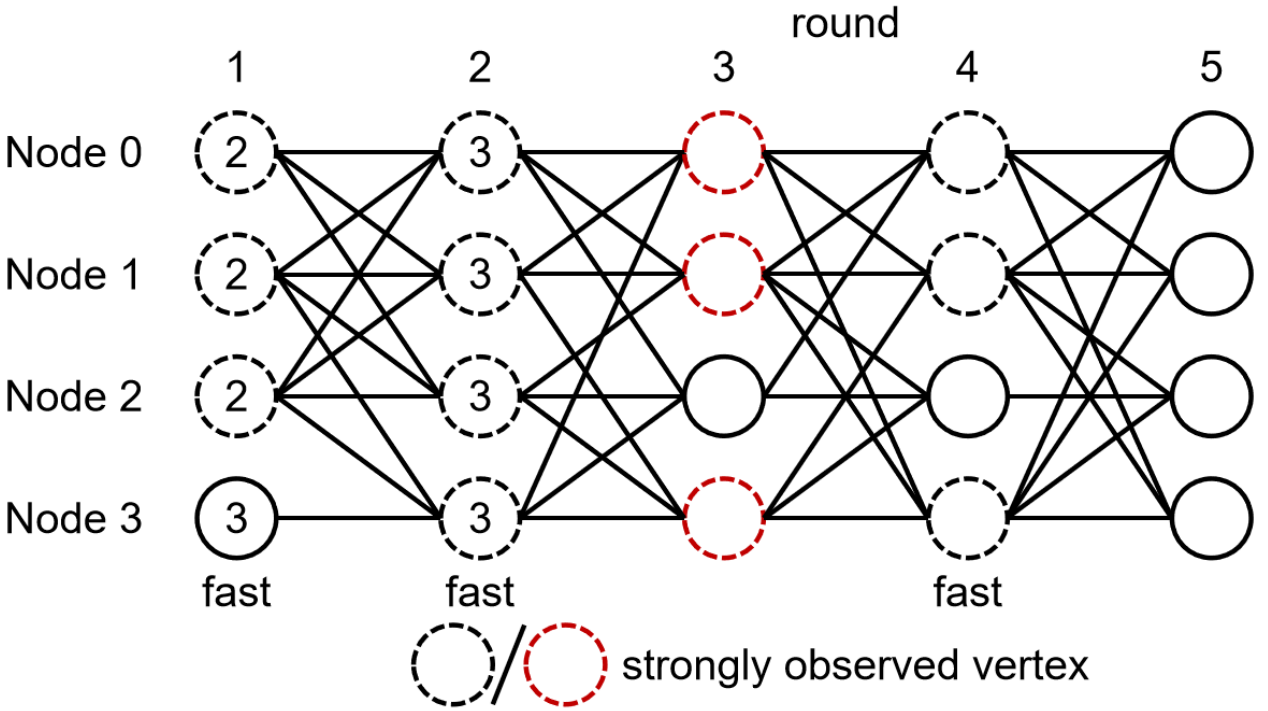
Commit rules: Normal path

- Select leaders by random coins (like Tusk)
- Leader vertices recursively decide preceding rounds



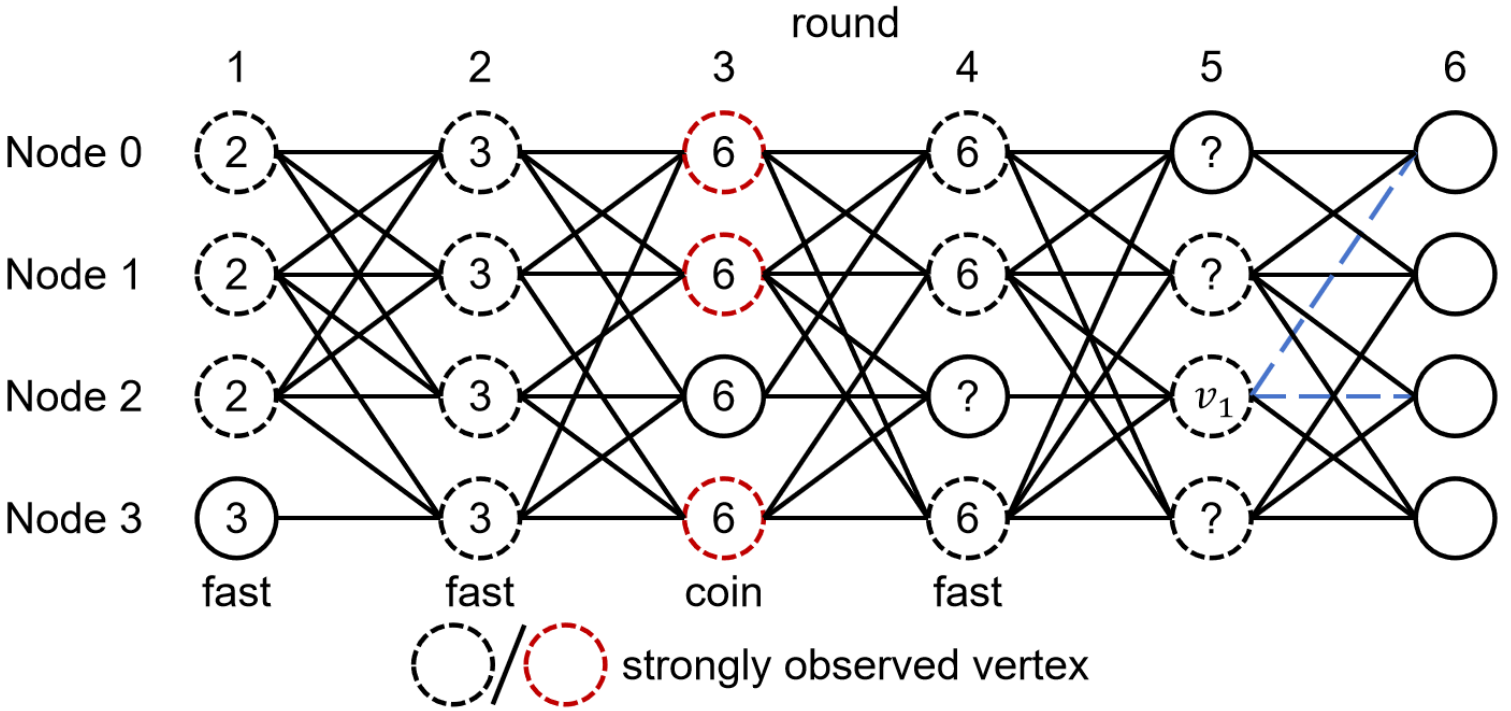
Commit rules: Normal path

- Select leaders by random coins (like Tusk)
- Leader vertices recursively decide preceding rounds



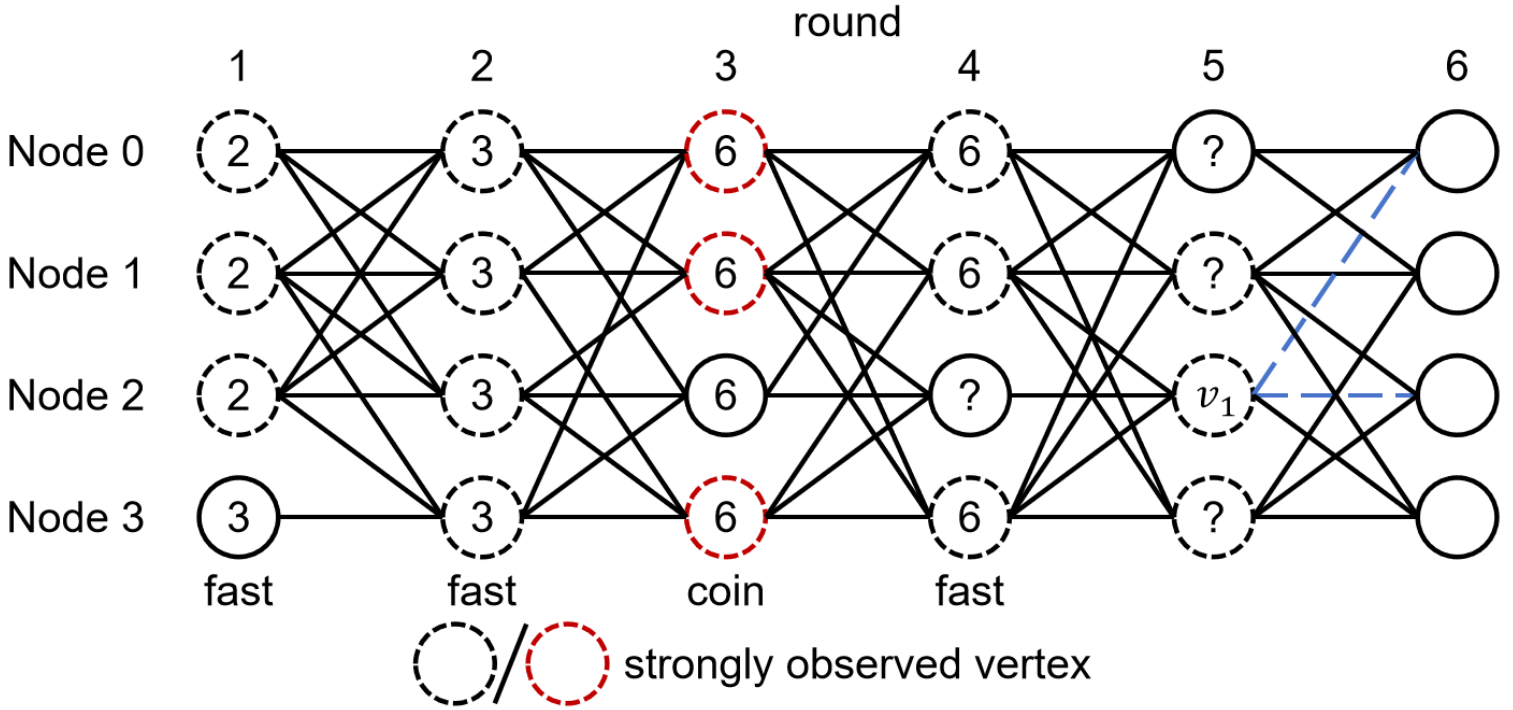
Commit rules: Normal path

- Select leaders by random coins (like Tusk)
- Leader vertices recursively decide preceding rounds



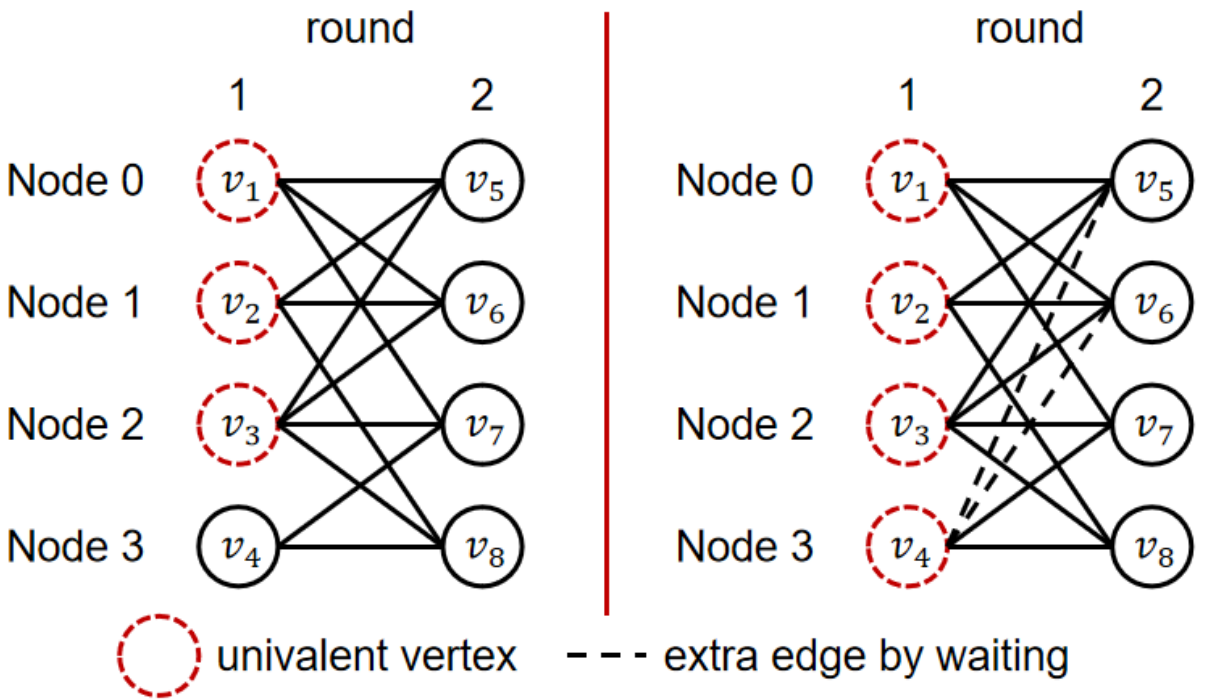
Commit rules: Normal path

- Select leaders by random coins (like Tusk)
- Leader vertices recursively decide preceding rounds
- Every round is decided either by the fast path or by the same leader vertex



Adaptive wait

- Without wait: fast path fails due to bivalent (v_4)
- With wait: more edges turn v_4 into univalent



- Intuition: more edges are needed to help bivalent become univalent
- **DO NOT** introduce Δ timer on the fast path
- Adaptive wait:
  the vertex received by $f + 1$ nodes definitely can be delivered

Experimental setup

Cloud Platform:	Amazon EC2
Hardware:	8 vCPUs, 32GiB RAM, 5Gbps Network bandwidth } t3.2xlarge
OS:	Ubuntu 22.04.4 LTS
Regions:	Ohio, Singapore, Tokyo, Canada (Central) and Frankfurt
Comparison:	**Chitu**[1] vs. Tusk[2] vs. Bullshark[3]
Payload:	1000 B
Test mode:	Open-loop

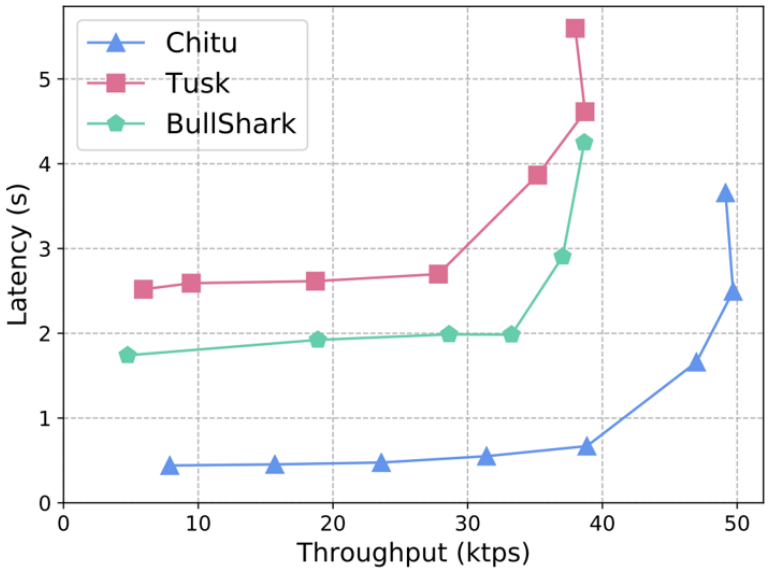
[1]https://github.com/Decentralized-Computing-Lab/ChituBFT

[2]https://github.com/facebookresearch/narwhal

[3]https://github.com/MystenLabs/narwhal

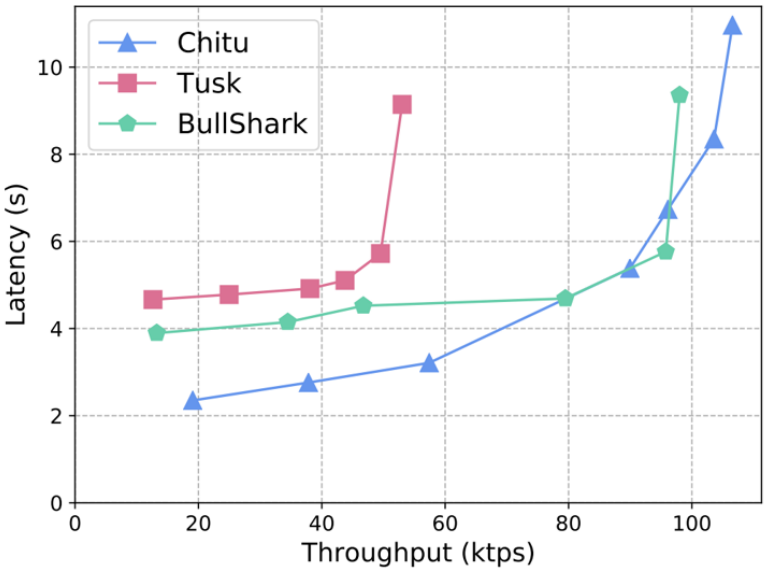
Fault-free performance

$n = 4$



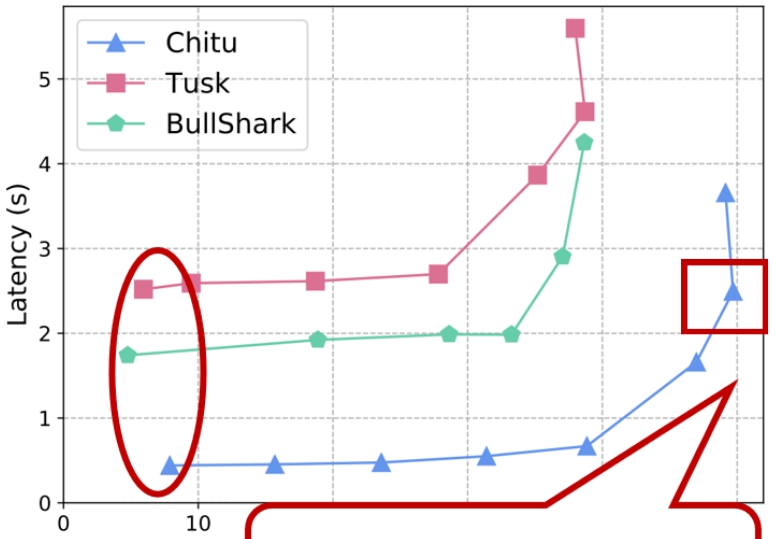
Increase client request sending rate

$n = 100$



Fault-free performance

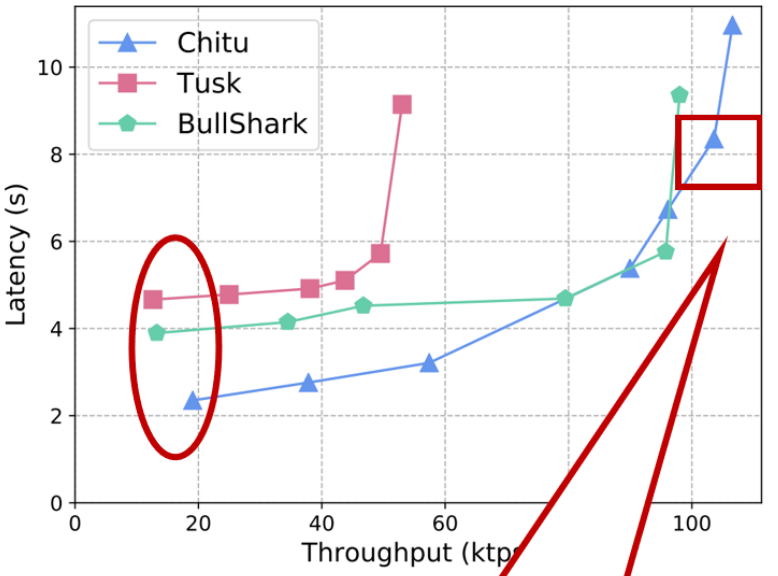
$n = 4$



49.7K txs/sec

Increase client request sending rate

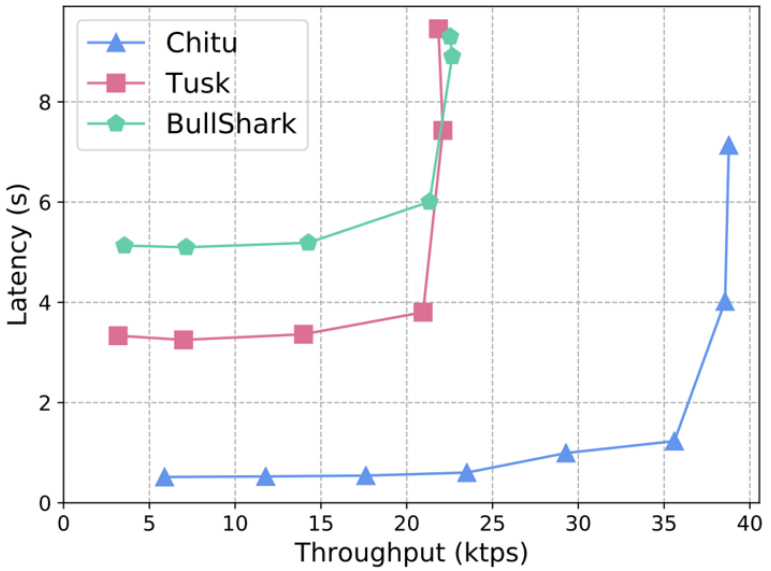
$n = 100$



103.6K txs/sec

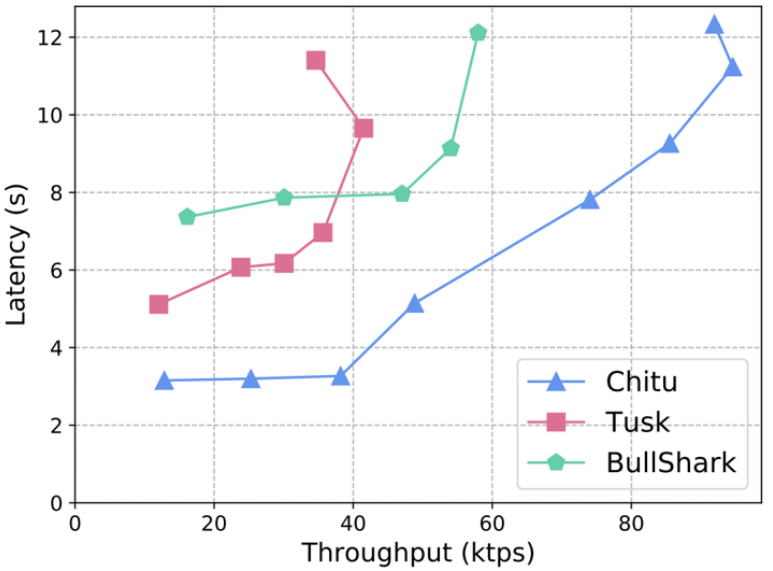
Performance under crash faults

$n = 4$



Increase client request sending rate

$n = 100$

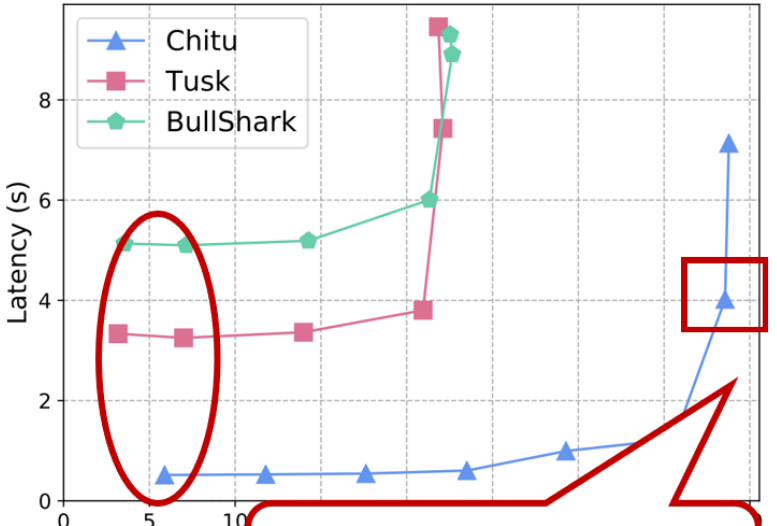


Performance under crash faults

$n = 4$

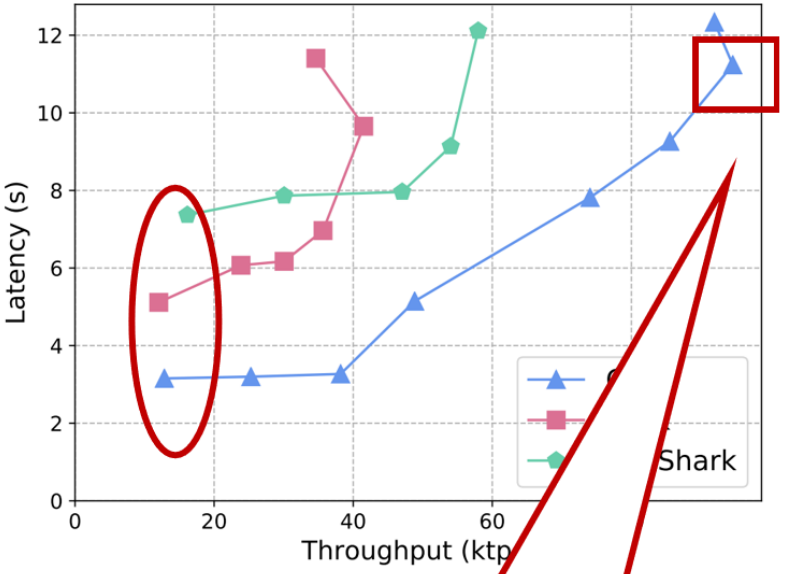
All vertices become univalent committed through the fast path

$n = 100$



38.6K txs/sec

Increase client request sending rate



94.6K txs/sec

More results in our paper

- Performance analysis
  - compared to Tusk, MyTumbler and Red Belly
- Performance comparison between with and without adaptive wait
  - Average and 95%ile latency
  - Percentage of fast-path univalent commits
- Performance under Byzantine faults
- Average and 95%ile latency with skewed distribution
- Pseudocode and correctness proof attached in appendices

Conclusion

Fair-Fallback Framework, a generic framework for reducing latency in BFT consensus protocols in a **leaderless** manner

- ✓ Apply to partially synchronous / asynchronous, BFT / CFT / fair scheduler protocols

Chitu, a DAG-based asynchronous protocol following this framework

- ☐ Retain the simplicity feature of DAG protocols
  - ✓ Easy to implement
- ☐ Bypass expensive random coins and achieve consensus efficiently
  - ✓ Commit proposals in four message delays in the best case
  - ✓ Reduce latency in several typical scenarios

Thank you!

Contact: ryan_huang@sjtu.edu.cn