Optimizing Consensus Algorithms for Permissoned Blockchains

Gengrui (Edward) Zhang and Hans-Arno Jacobsen

Toronto Blockchain Week

UNIVERSITY OF TORONTO

MIDDLEWARE SYSTEM RESEARTCH GROUP
MSRG.ORG

Outline

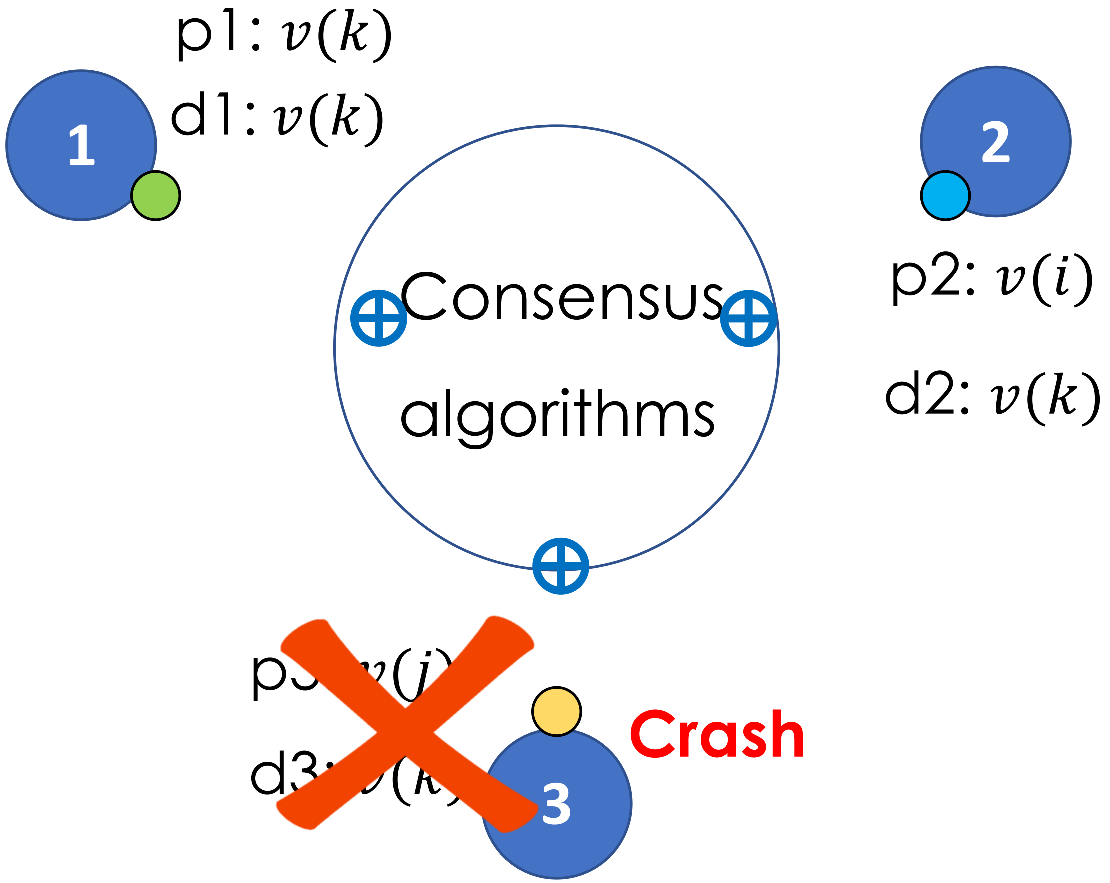
- What is a consensus algorithm?

- Commonly used approaches
  - ✓ Proof-of-Work (PoW)
  - ✓ PBFT
  - ✓ Paxos and Raft

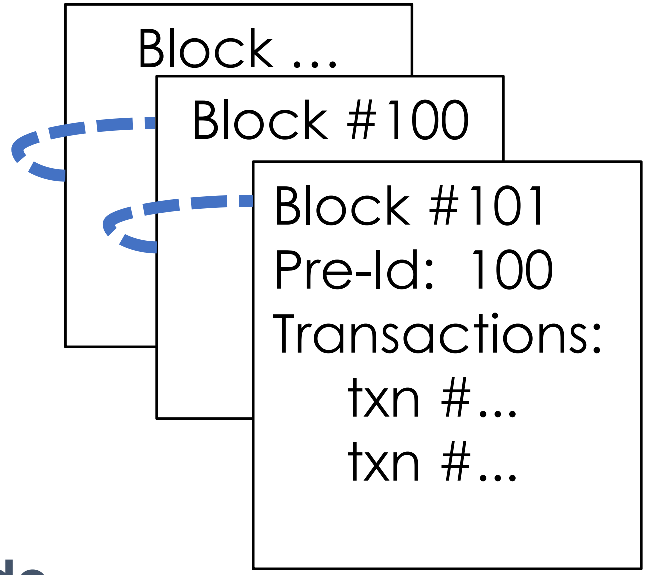
- Overview of our approach
  - PoCRaft: PoW + Raft

What is a consensus algorithm?

- From *undecided* state to *decided* state
  - Propose
  - Communicate
  - Decide
- Handle failures
  - Fail-stop failures (FS)
  - Crash failures (CF)
  - Byzantine failures (BF)



Proof-of-Work(PoW) in Permissionless Blockchains (Bitcoin)

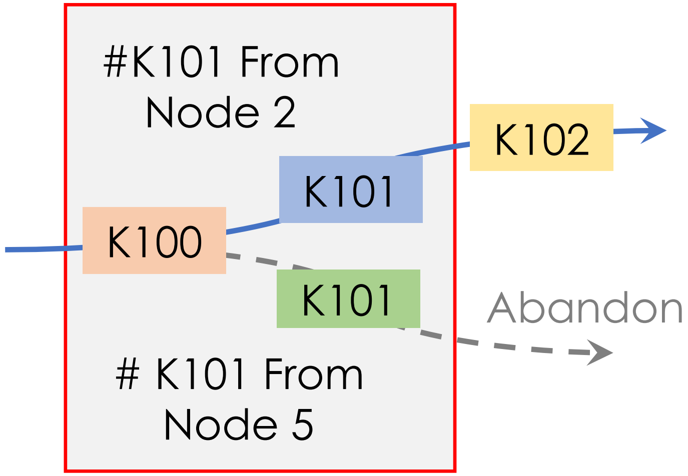


+ **Nonce**:
Prj%ld*…XXX

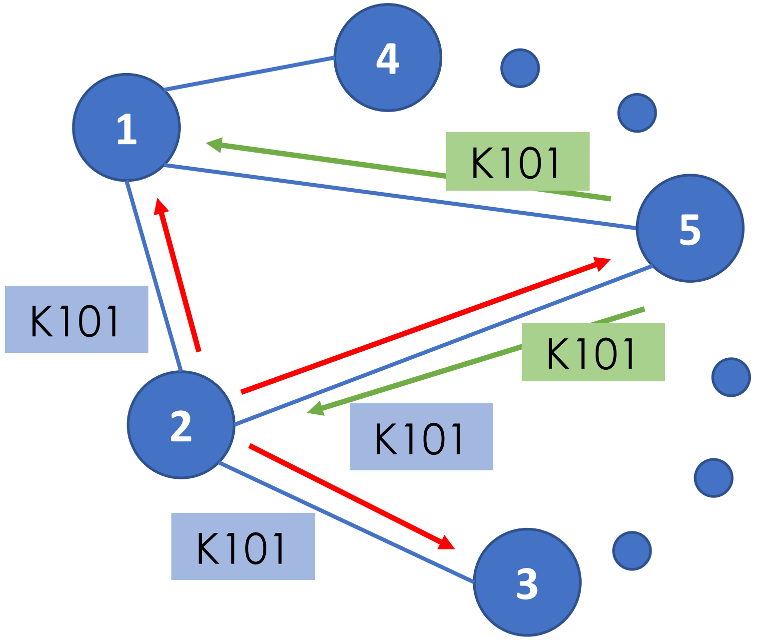
Hash block with a nonce

0003nFsn#$rG..... XXX

do
hash work
until
result meets an
output requirement

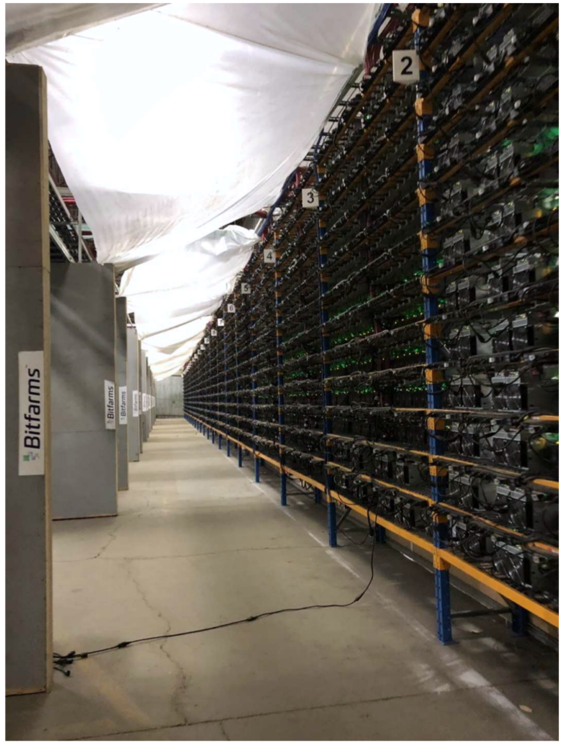


- Longest chain rule
- Open, entirely decentralized
- Good scalability



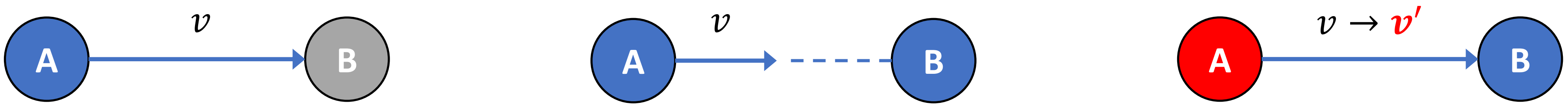
Limitations of Proof-of-Work

- Limited throughput
  - Due to protocol design, e.g., block size, varying proof difficulty
- High latency
  - Due to multi-block confirmations
- Wasted power
  - Due to redundant hash computation



Consensus Protocols in Permissioned Blockchains

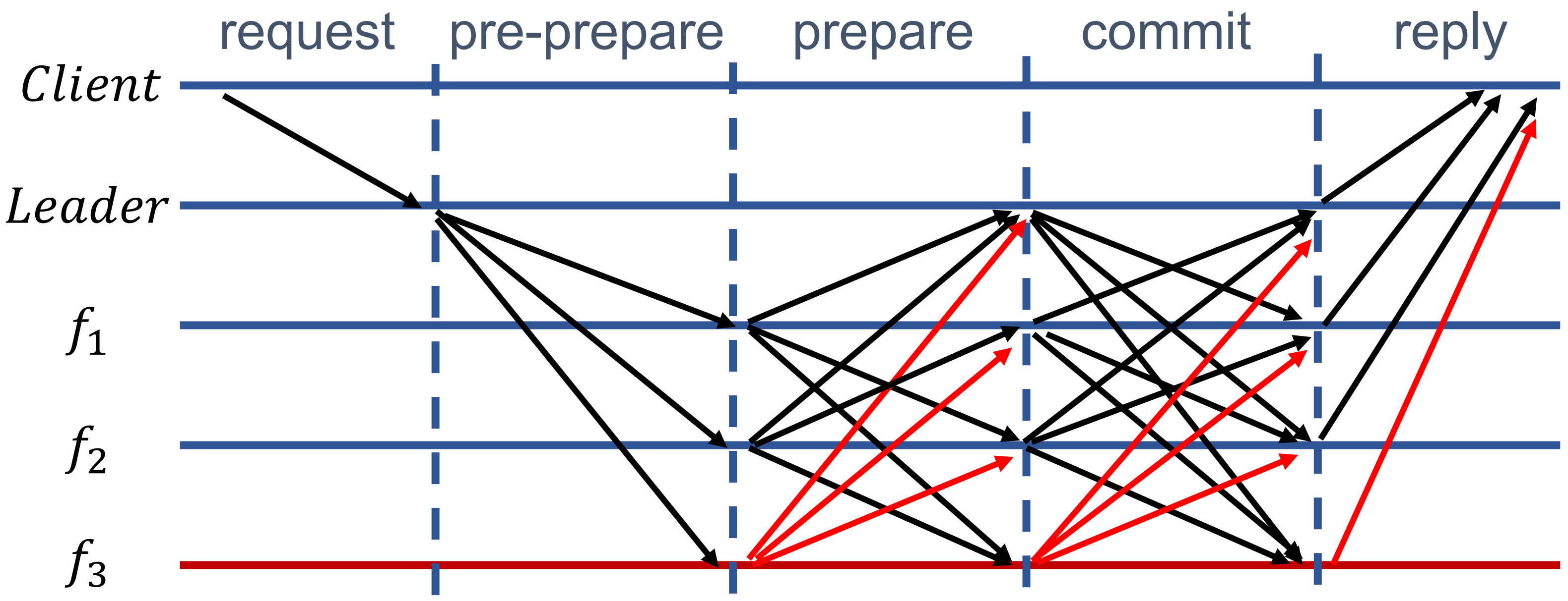
- System context
  - Cooperating participants
  - Verified identities
- Fault tolerance
  - Non-Byzantine conditions: **(Paxos, Raft)**
    - Crash failures, omission failures such as network delays, partitions, packet loss, duplication, and reordering
  - Byzantine conditions: processes may exhibit arbitrary failures **(PBFT)**



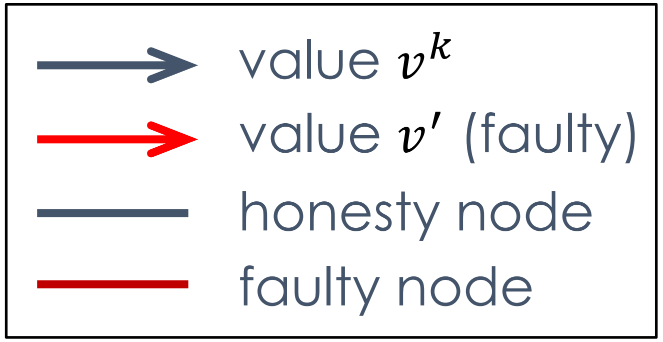
crash failures (Node B crashes) | omission (packet loss) | Byzantine (send a modified value)

Practical Byzantine Fault Tolerance (PBFT)

- Byzantine broadcast to reach a Byzantine agreement
- Client waits for $f + 1$ replies from different replicas with the same result
- View change (deal with a faulty leader)



L, f_1, f_2 $commit\ a\ value$
$= majority\{v_{f_2}^k, v_{f_3}^{k'}, v_l^k\}$
$= k$



Practical Byzantine Fault Tolerance (PBFT)

- Features
  - Compares the received value with others' value
  - Involves $\mathcal{O}(n^2)$ message transmissions
  - Requires $3f + 1$ nodes to reach Byzantine agreement, where f represents number of failures that can be tolerated
- PBFT in blockchains
  - Hyperledger Fabric, Zilliqa, R3 Corda, Symbiont

HYPERLEDGER zilliqa r3. corda symbiont

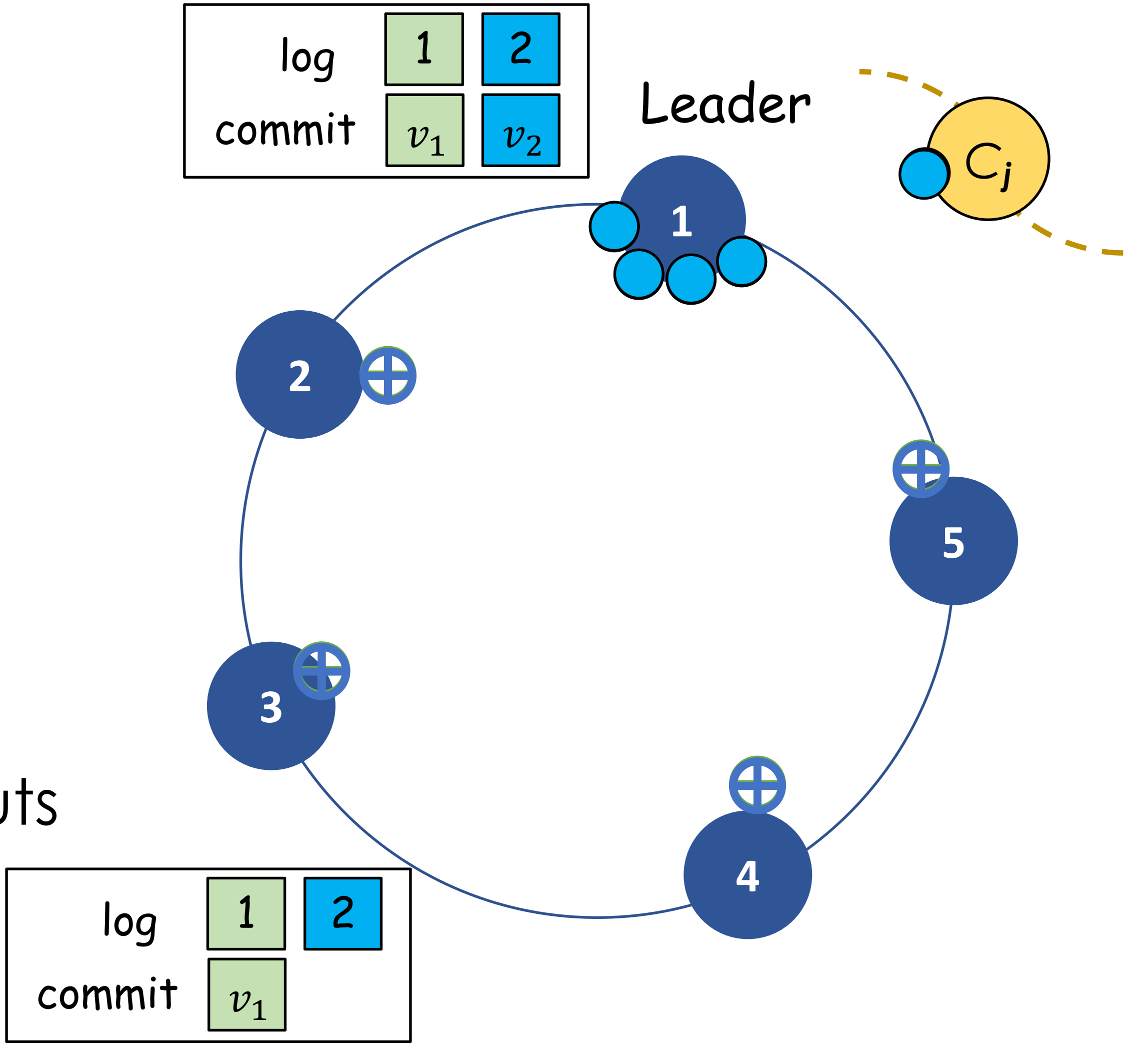
Qualitative Comparisons between PoW and BFT

	Proof-Of-Work	BFT-based protocols
Node identity management	Open, entirely decentralized	Permissioned, nodes need to know IDs of all other nodes
Throughput	Limited (due to possible forks)	Good (tens of thousands of TPS)
Latency	High latency (due to multi-block confirmations)	Excellent (effected by network latency)
Power consumption	Poor (redundant hash computation)	Good (no needless computation)
Scalability	Excellent (like Bitcoin)	Limited (not well explored)
Correctness proofs	no	yes

Adapted from: Vukolić, Marko. "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication." *International workshop on open problems in network security*. Springer, Cham, 2015

Raft

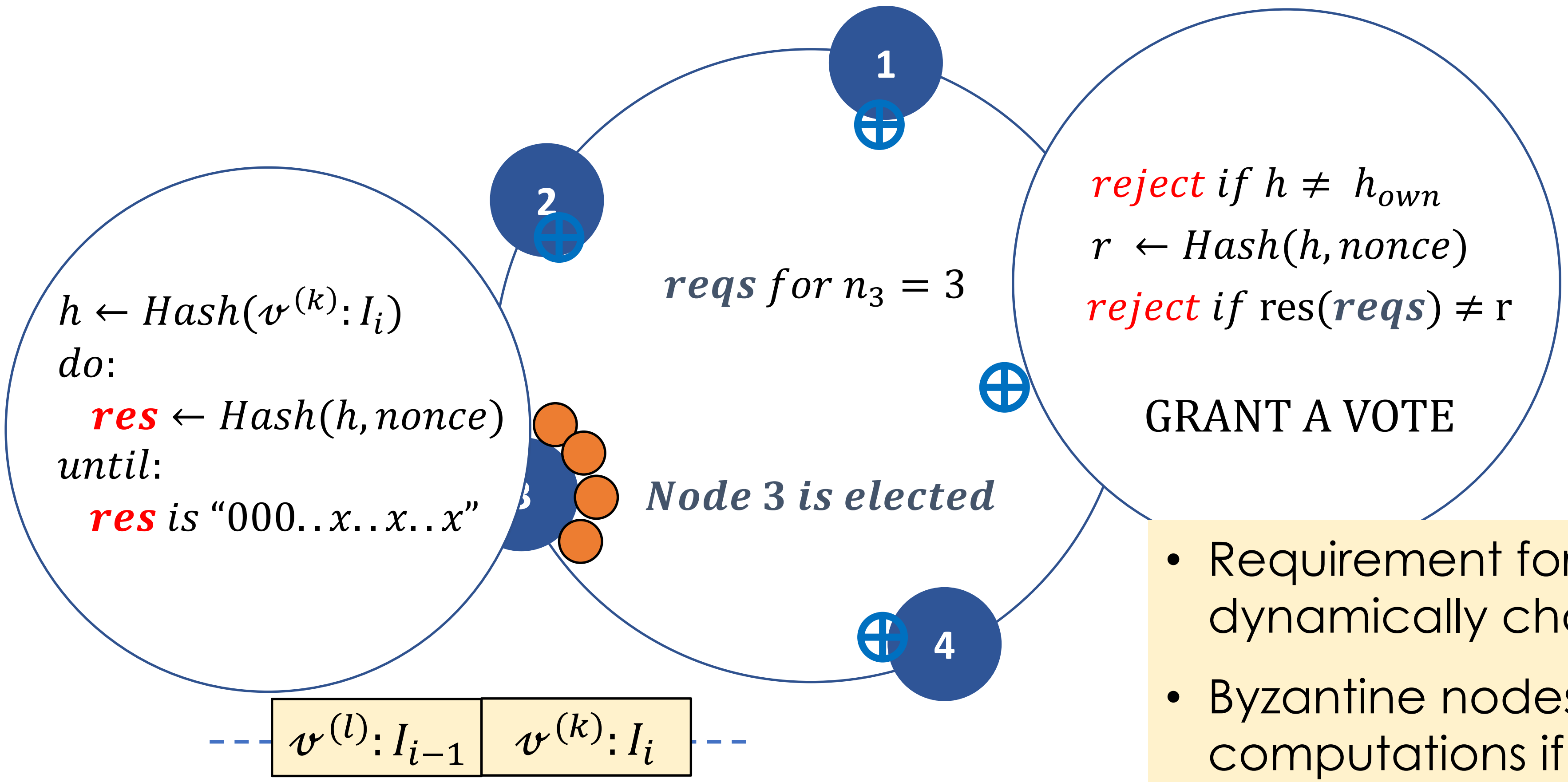
- Features
  - Strong leadership in log replication
  - New elections when leader crashes
  - Involves $\mathcal{O}(n)$ message transmissions
    - Better than PBFT's $\mathcal{O}(n^2)$
- Limitations
  - Vulnerable to Byzantine faults
  - Elections are only initiated by timeouts



Our Approach ---- PoCRaft

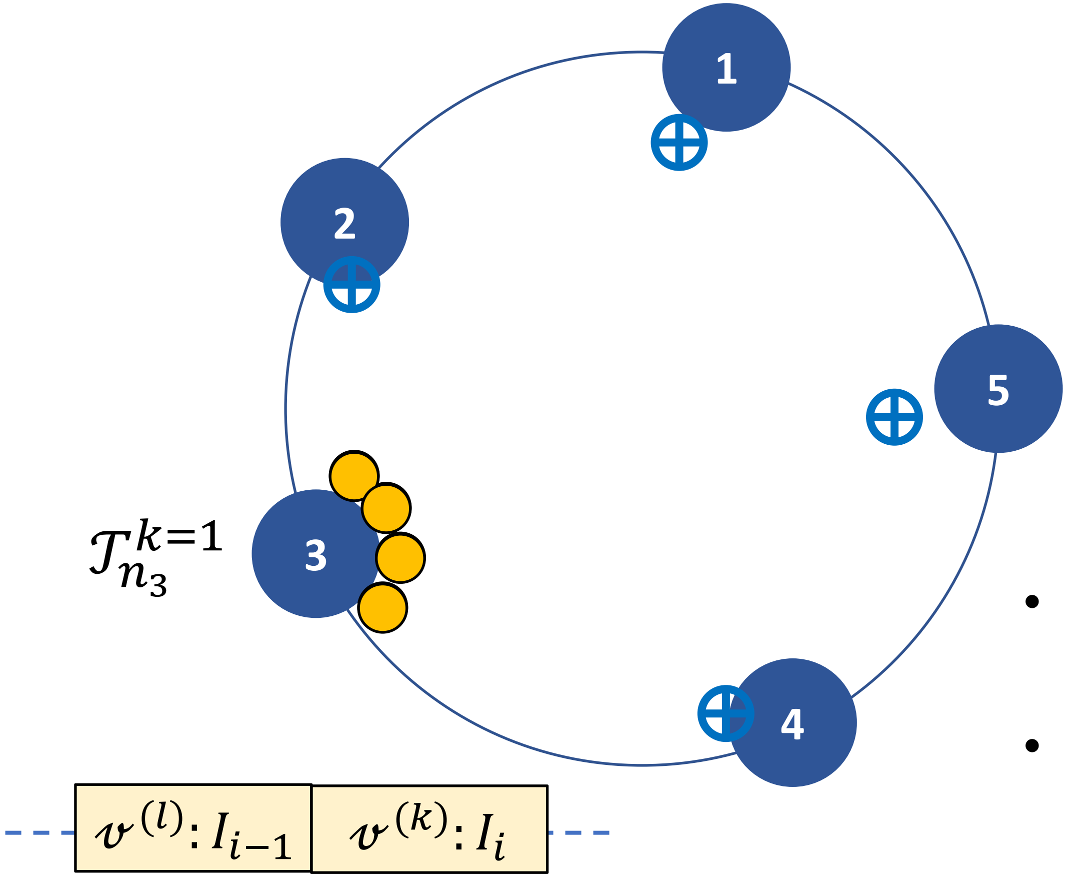
- Enable BFT in Raft, reduce message transmissions
- Proof-based leader election
  - Proof-of-Commit (PoC)
  - Byzantine nodes do more hash computation if they continuously start elections
- Encrypted log replication
  - Leader collects signed replies from followers indicating what they have received
  - Ensure log safety even the current leader is faulty
- Fault tolerance
  - Requires $3f + 1$ nodes to tolerate f Byzantine failures
  - Involves $\mathcal{O}(n)$ messages rather than $\mathcal{O}(n^2)$ as in PBFT under normal operations

PoCRaft: Proof-of-Commit Elections



- Requirement for the hash work dynamically changes
- Byzantine nodes will do more hash computations if they continuously start elections

PoCRaft: Encrypted Log Replications



Leader sends: $<\delta_{\mathcal{L}}(d_{Tx}, d_{LTx}, I_i, t_{\mathcal{L}})>$

Followers reply: $\mathcal{R}_i = <\delta_{\mathcal{F}_1}(d_{Tx}, I_i, t_{\mathcal{R}_i})>$

Leader collects: $\mathbb{R} = \{\mathcal{R}_1, \ldots, \mathcal{R}_k\}\ until\ |\mathbb{R}| = n/2$

then leader sends: $<\delta_{\mathcal{L}}(d_{\mathbb{R}}, I_i, t_{\mathcal{L}}),\ \mathbb{R},\ t_{\mathcal{L}}>$

- Followers check signed replies in $\mathbb{R}$ to acquire what other nodes have previously received
- Client waits for $n/2$ replies with an identical value and index $\{v^{(k)}: I_i\}$ to confirm the proposed value has been committed

Conclusions

- Efficient consensus protocols lead to higher throughput and lower latency blockchains
- State-of-the-art consensus protocols ensure correctness
- Optimizations based on message transmission complexity and efficient leader election could be considered in the future

Thank you

Gengrui (Edward) Zhang and Hans-Arno Jacobsen

Email: gengrui.zhang@mail.utoronto.ca