ACM Middleware 2021

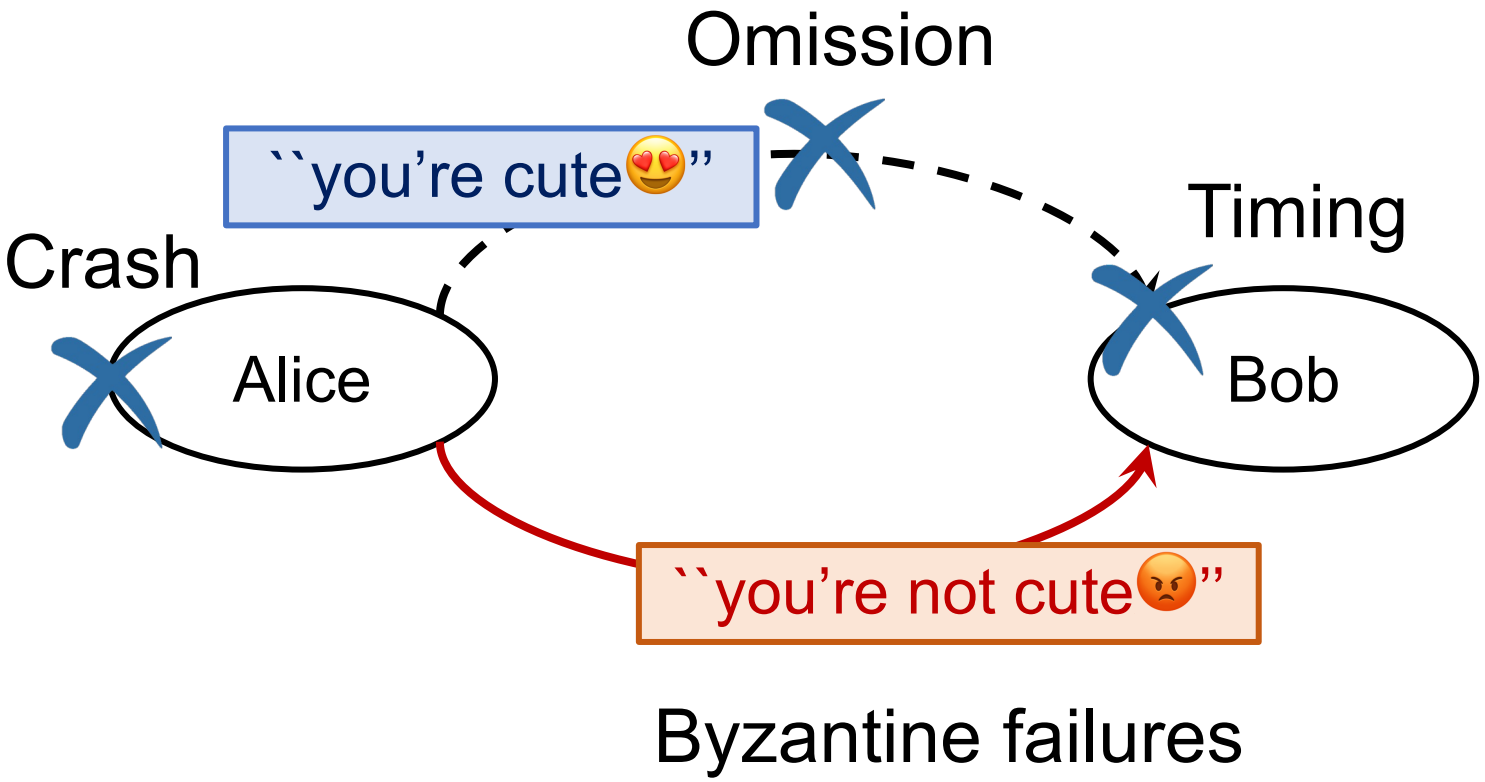
Prosecutor: An Efficient BFT Consensus Algorithm with Behavior-aware Penalization Against Byzantine Attacks

Gengrui Zhang and Hans-Arno Jacobsen

Department of Electrical & Computer Engineering

University of Toronto

Byzantine failures



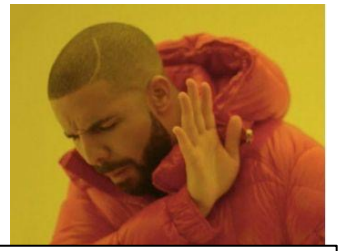
Byzantine failures

Byzantine failures, a.k.a., arbitrary failures:

the new state of the faulty server and the contents of the messages sent are **completely unconstrained**

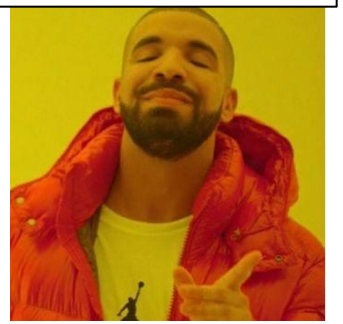
Why tolerating Byzantine failures is important?

- Unreliable hardware
  - Boeing 777[1][2]
  - Space X Dragon[3]
- Growing software scales/bugs and operator mistakes
  - Cloudflare[4]
- Blockchain application
  - Bitcoin
  - Diem



Crash fault tolerance

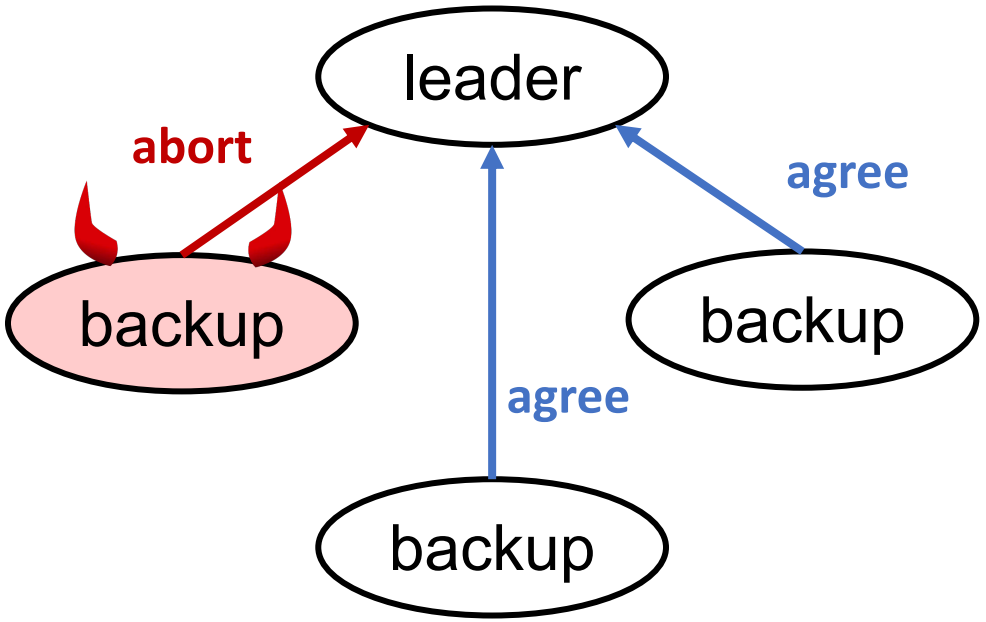
Blockchains



Byzantine fault tolerance

1. M., Paulitsch; Driscoll, K. (9 January 2015). "Chapter 48:SAFEbus". In Zurawski, Richard (ed.). *Industrial Communication Technology Handbook, Second Edition*. CRC Press. pp. 48–1–48–26.
2. Yeh, Y.C. (2001). "Safety critical avionics for the 777 primary flight controls system". 20th DASC. 20th Digital Avionics Systems Conference (Cat. No.01CH37219). 1. pp. 1C2/1–1C2/11
3. ELC: SpaceX lessons learned, LWN.net, https://lwn.net/Articles/540368/
4. A Byzantine failure in the real world, https://blog.cloudflare.com/a-byzantine-failure-in-the-real-world/

Replication in leader-based BFT protocols

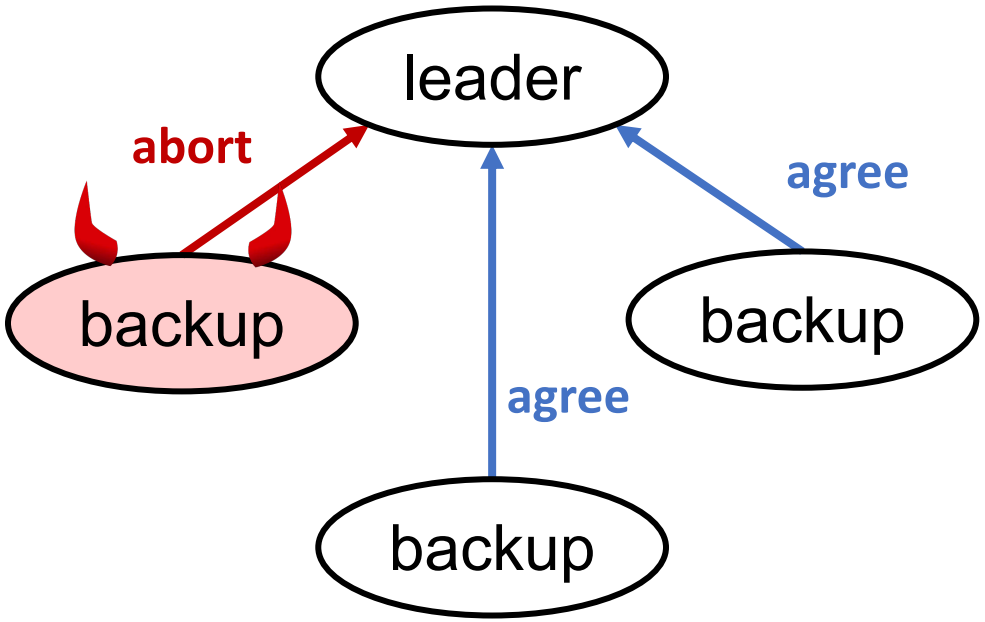


Backups: <abort, agree, agree>

Consensus proceeds

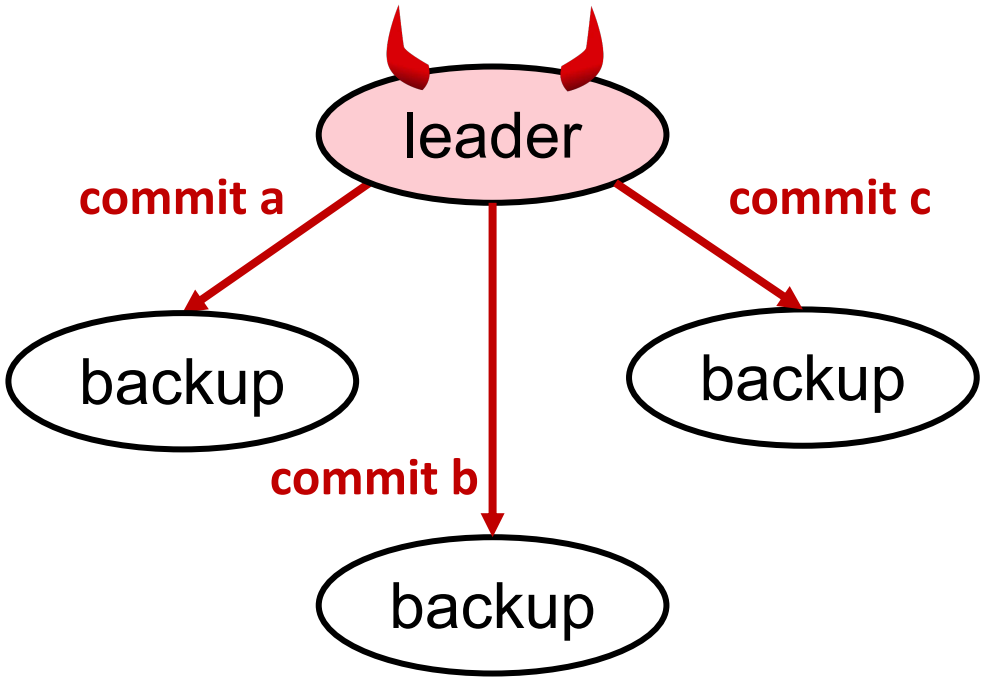
- Utilizing a leader increases efficiency
  - Avoids conflicts
  - Less message passing
- Quorum certificates
  - Message authentication
  - Lower bound: tolerating f faults out of a total $3f + 1$ replicas
    - PBFT ($O(n^2)$)
    - SBFT and HotStuff ($O(n)$)

Tolerating failures in leader-based BFT protocols



Backups: <abort, agree, agree>

Consensus proceeds



Consensus stops! Need to replace the faulty leader! View change starts

Failures on backups & leaders

- Failures on backups and leaders have completely different impact
- A correct leader and f backup failures: **No problem! Go ahead!**
- Leader failures are more catastrophic
  - Internally, no server manages consensus
  - Externally, no server handles client requests
  - The system cannot provide any service (out-of-service condition)

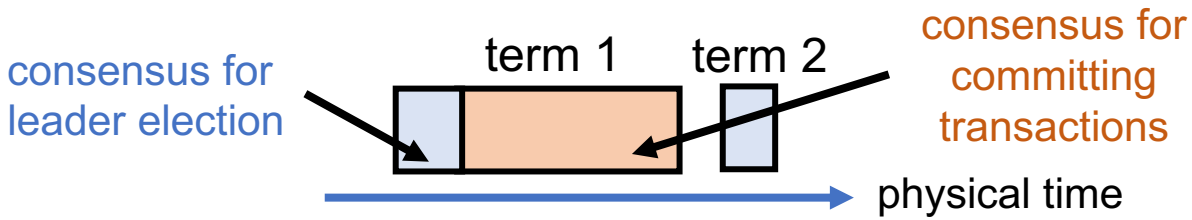
Question:

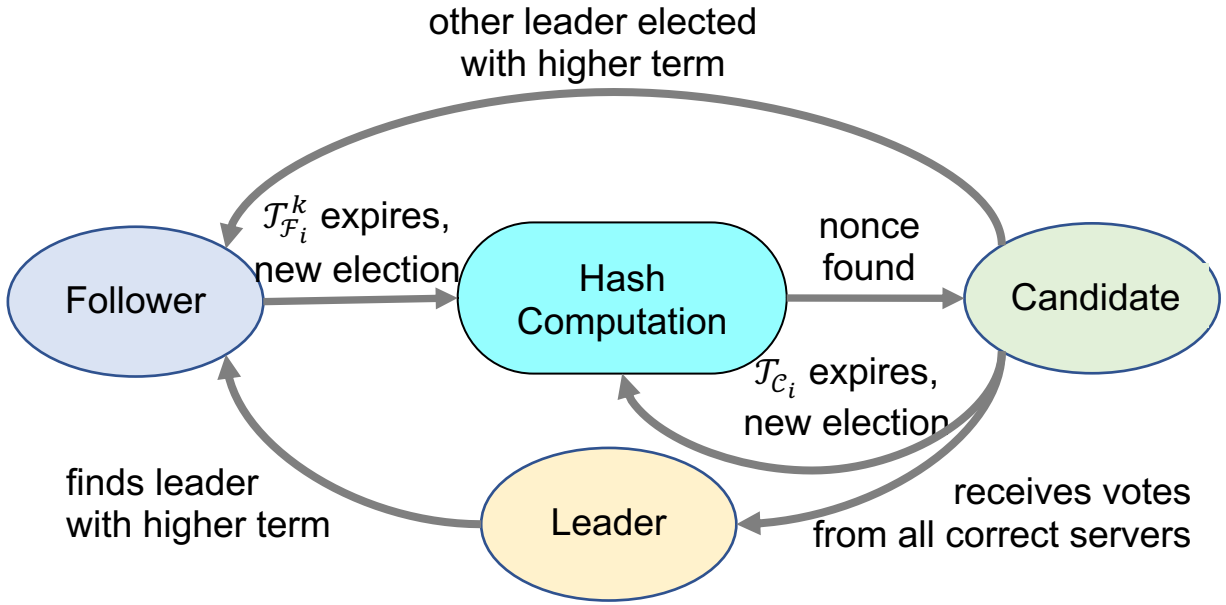
Is there any way to mitigate the impact of leader failures?

Prosecutor: Suppressing Byzantine servers

- Prosecutor aims to fortify the leader position
  - Suspects leader failures as potential malicious attacks
  - Penalizes suspected servers by imposing computation work
  - Diminishes the probability of faulty servers becoming leaders
  - Achieves consensus in linear messaging complexity
- Therefore, Byzantine servers **pay the price for attacking** the system, and the system **suppresses Byzantine servers** over time

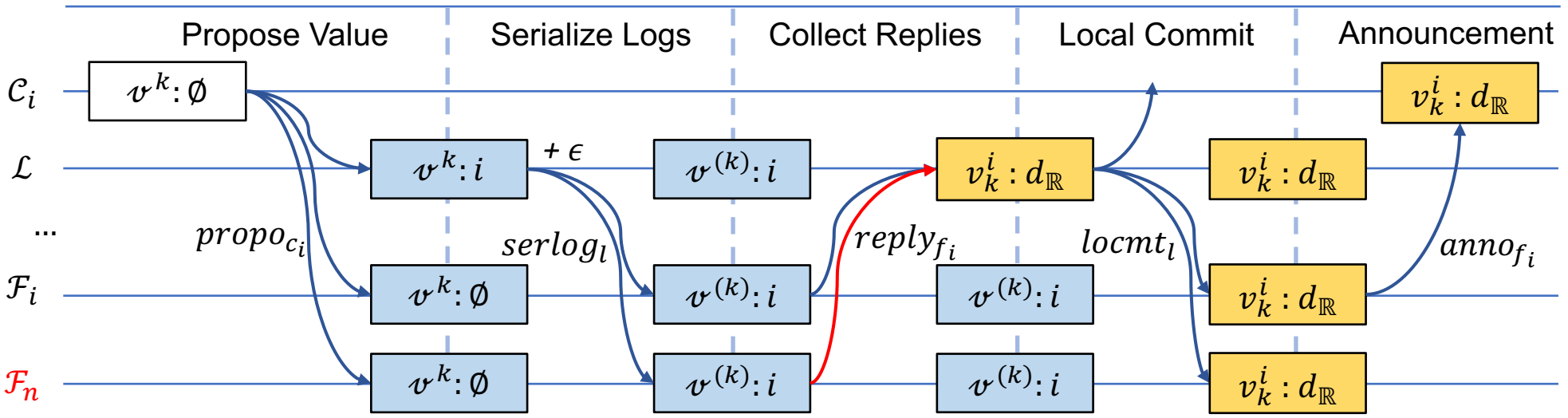
Terms and server state transitions





- Inspired by Raft [Ongaro et al., ATC'14], Prosecutor adapts *terms* and *server state transitions*.
- Terms are logical clocks
- *Follower*, *candidate*, and *leader*
- Election campaigns take place when followers trigger timeouts
- A leader comes from the candidate that has collected $2f + 1$ votes in the same term

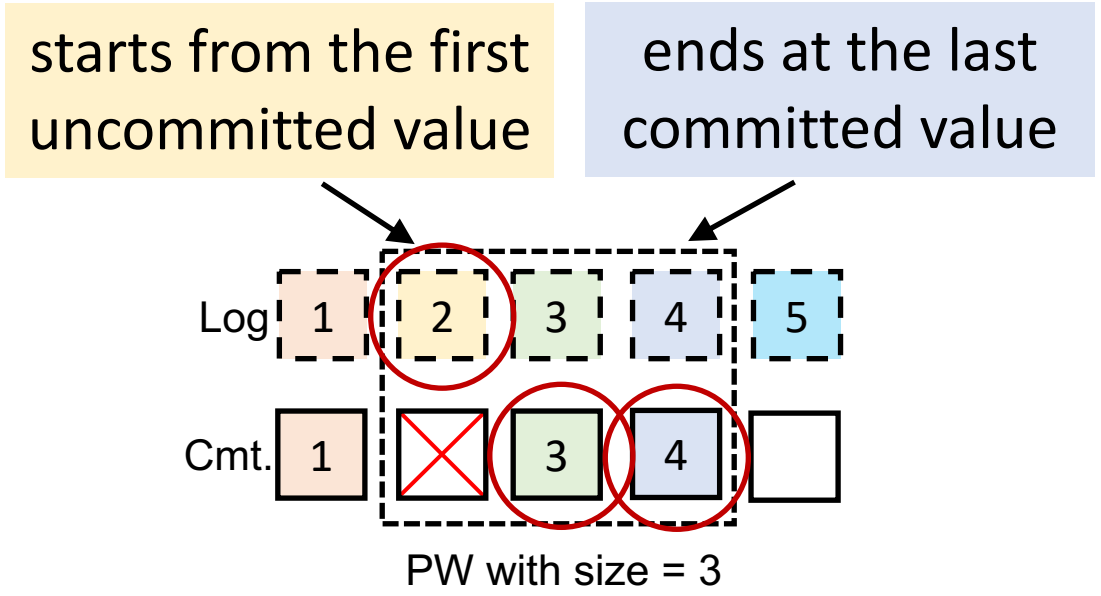
Transaction replication in normal operation

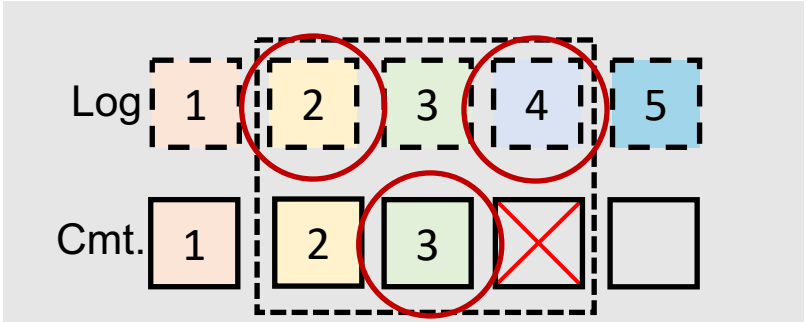


- Clients communicate with all servers (at least $2f + 1$)
- Leader collects $2f + 1$ replies and combines them using threshold signatures
- The replication scheme requires $3f + 1$ servers to tolerate f failures

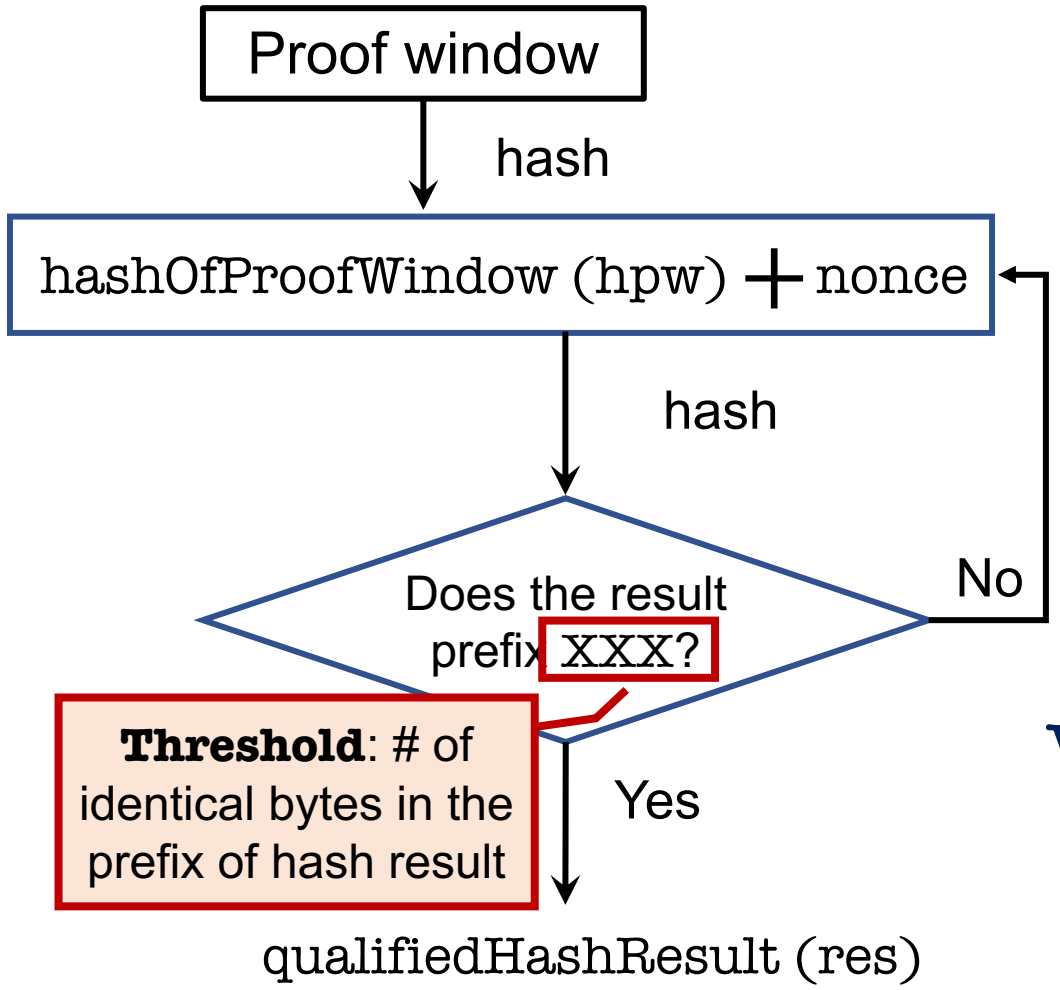
Proof windows (PWs) – log consistency requirement

- PWs illustrate servers' replication status (how up-to-date they are)
- Consensus can be issued in parallel
  - Some values committed on S_i can still be uncommitted on S_j
  - But they **must be** logged on S_j (quorum certificate)
- Validation of PWs treats log and commit equally





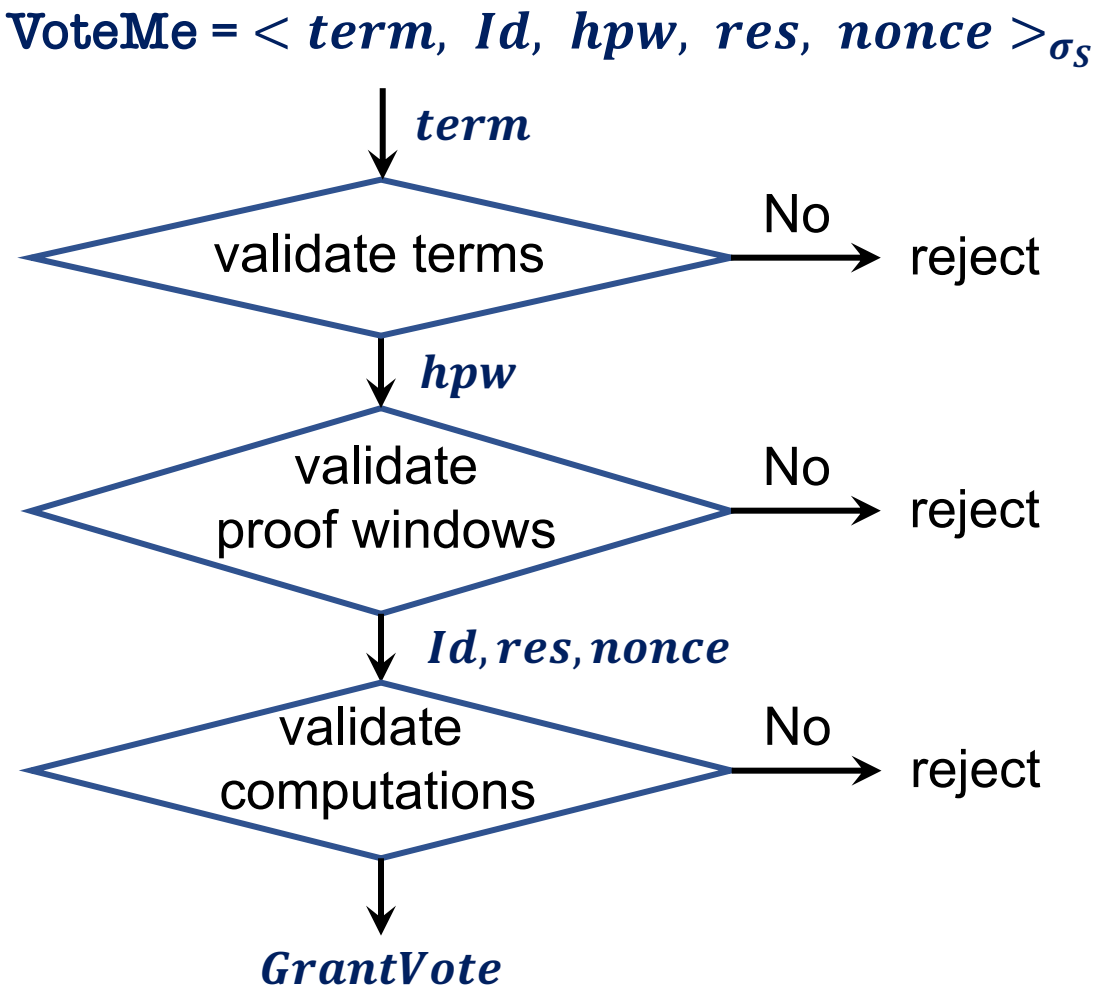
Proof-of-Commit leader election



- Similar to Proof-of-Work[Nakamoto, Bitcoin 2008], hashing is a brutal-force process
- The higher the *threshold* is, the more computation the server needs to perform
- Candidates broadcast **VoteMe** messages to all the other servers

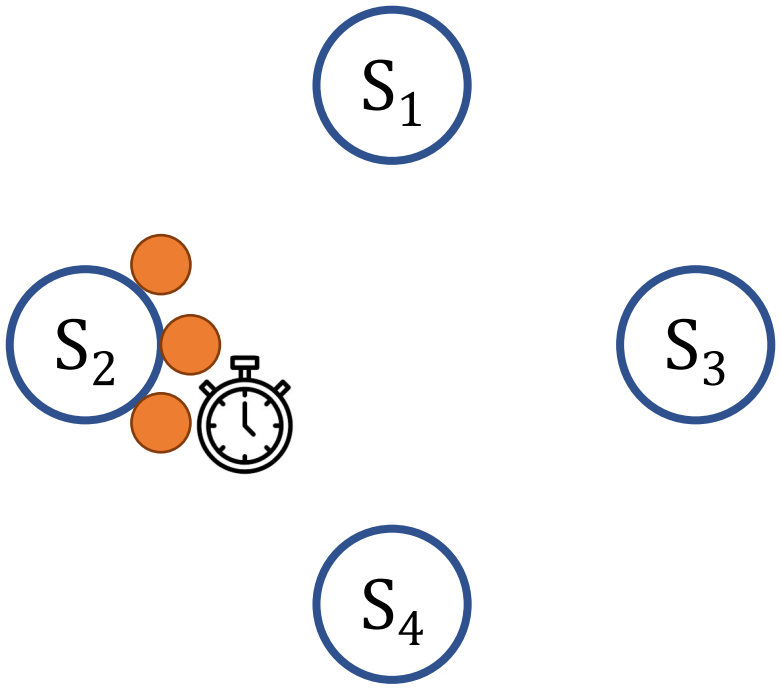
$$\textbf{VoteMe} = < \boldsymbol{term},\ \boldsymbol{Id},\ \boldsymbol{hpw},\ \boldsymbol{res},\ \boldsymbol{nonce} >_{\sigma_s}$$

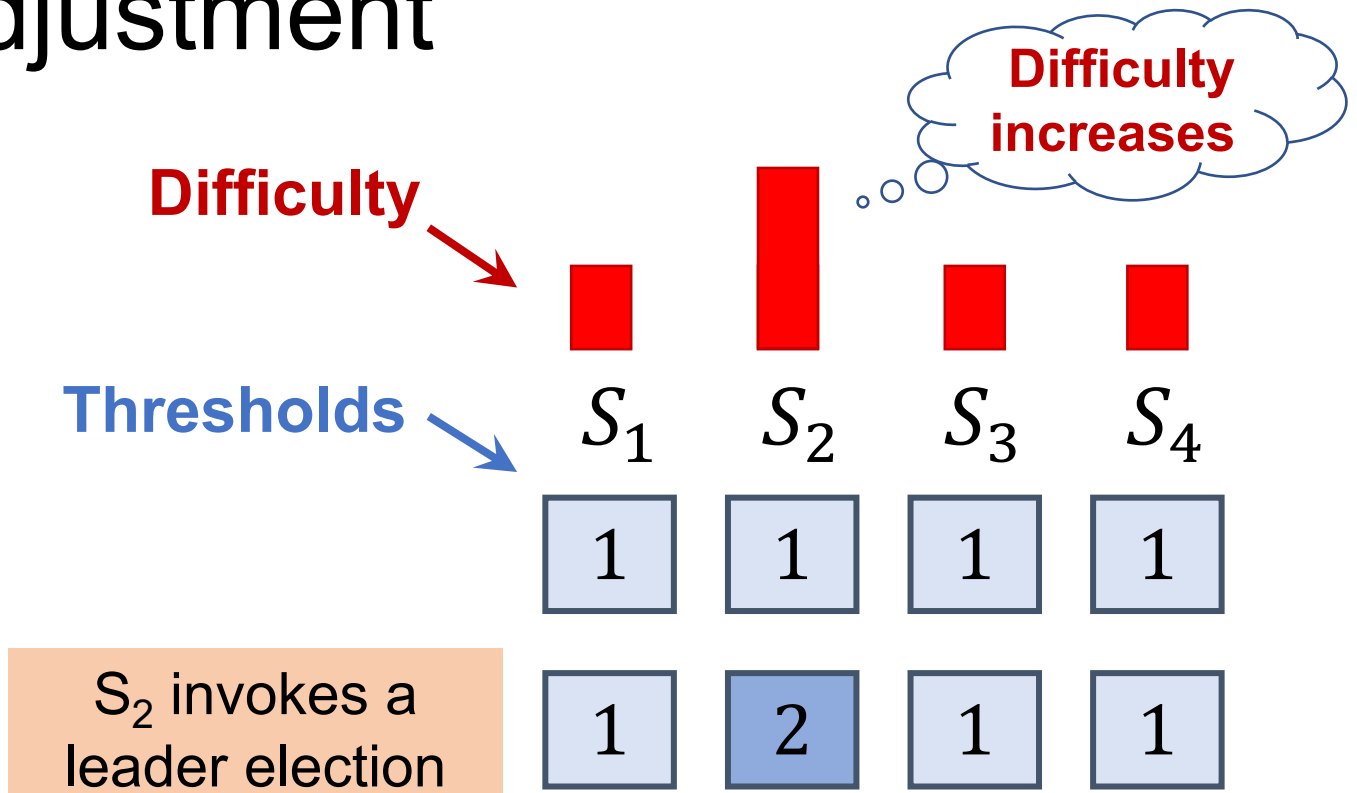
Threshold-determined computation penalty



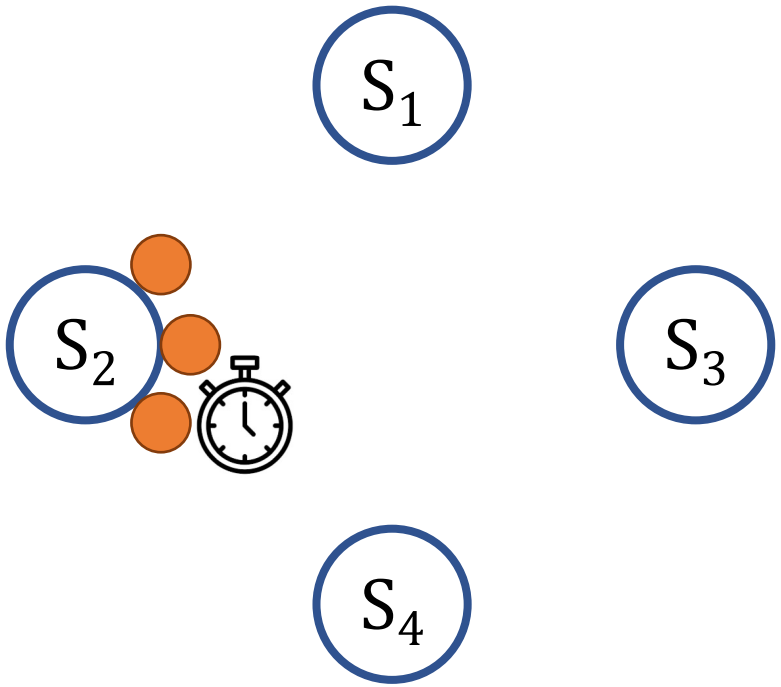
- Validation process takes O(1) time only; *computation puzzle is hard to solve but easy to verify*
- A `grantVote` indicates that
  - Candidate is up-to-date
  - Proof windows are identical
  - Performed computations meet corresponding threshold penalty

Demo: Threshold adjustment

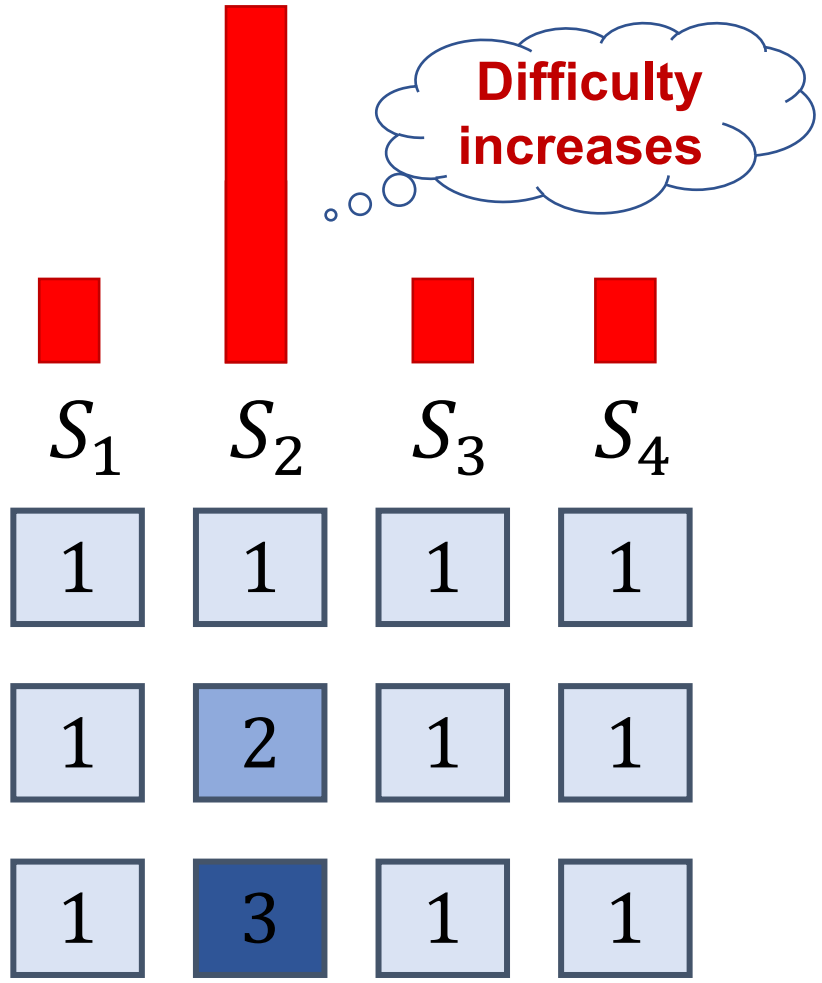




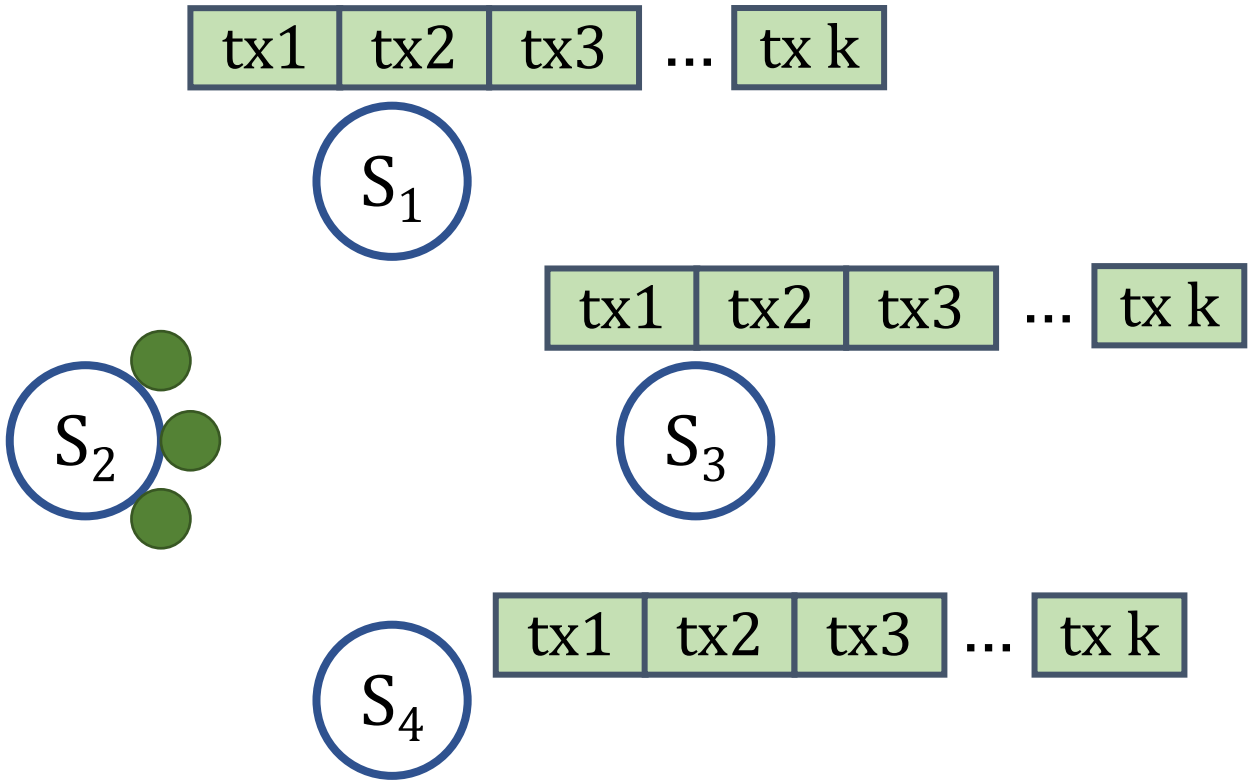
Demo: Threshold adjustment

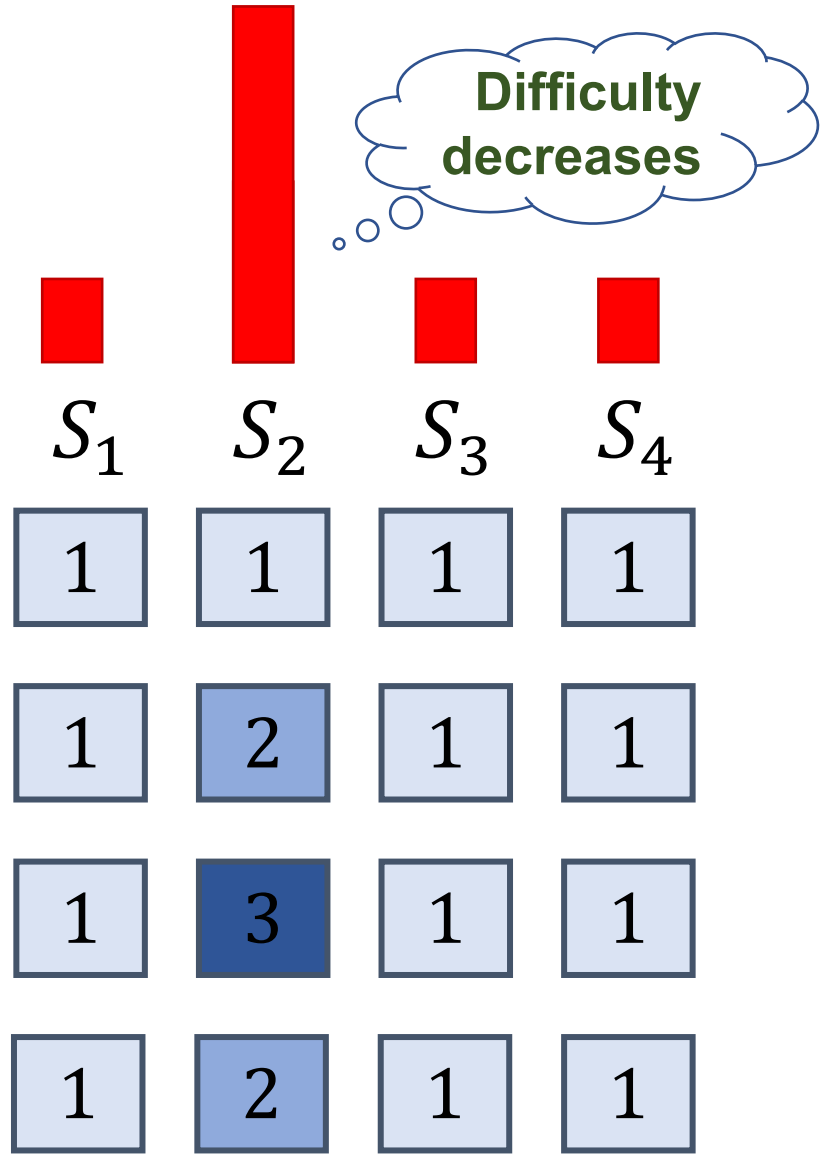


S_2 invokes a leader election again



Demo: Threshold adjustment



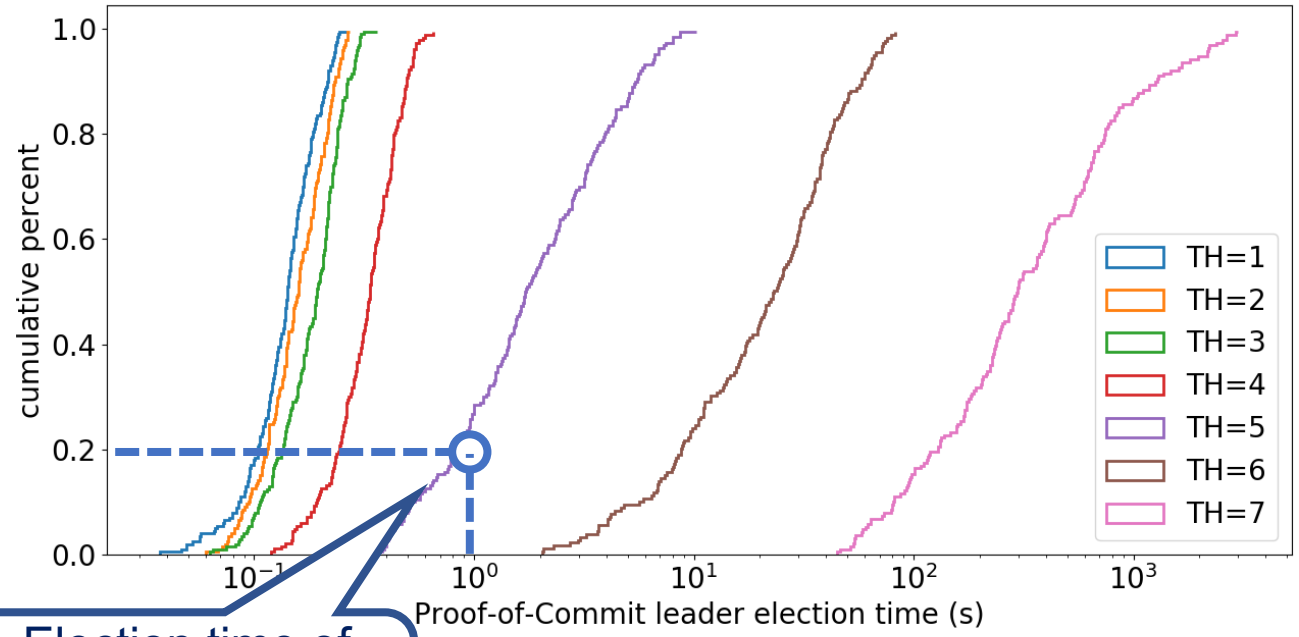


S_2 conducted k consensus processes

Leader election time under varying thresholds

Correct servers' thresholds remain low

Byzantine servers' thresholds increase

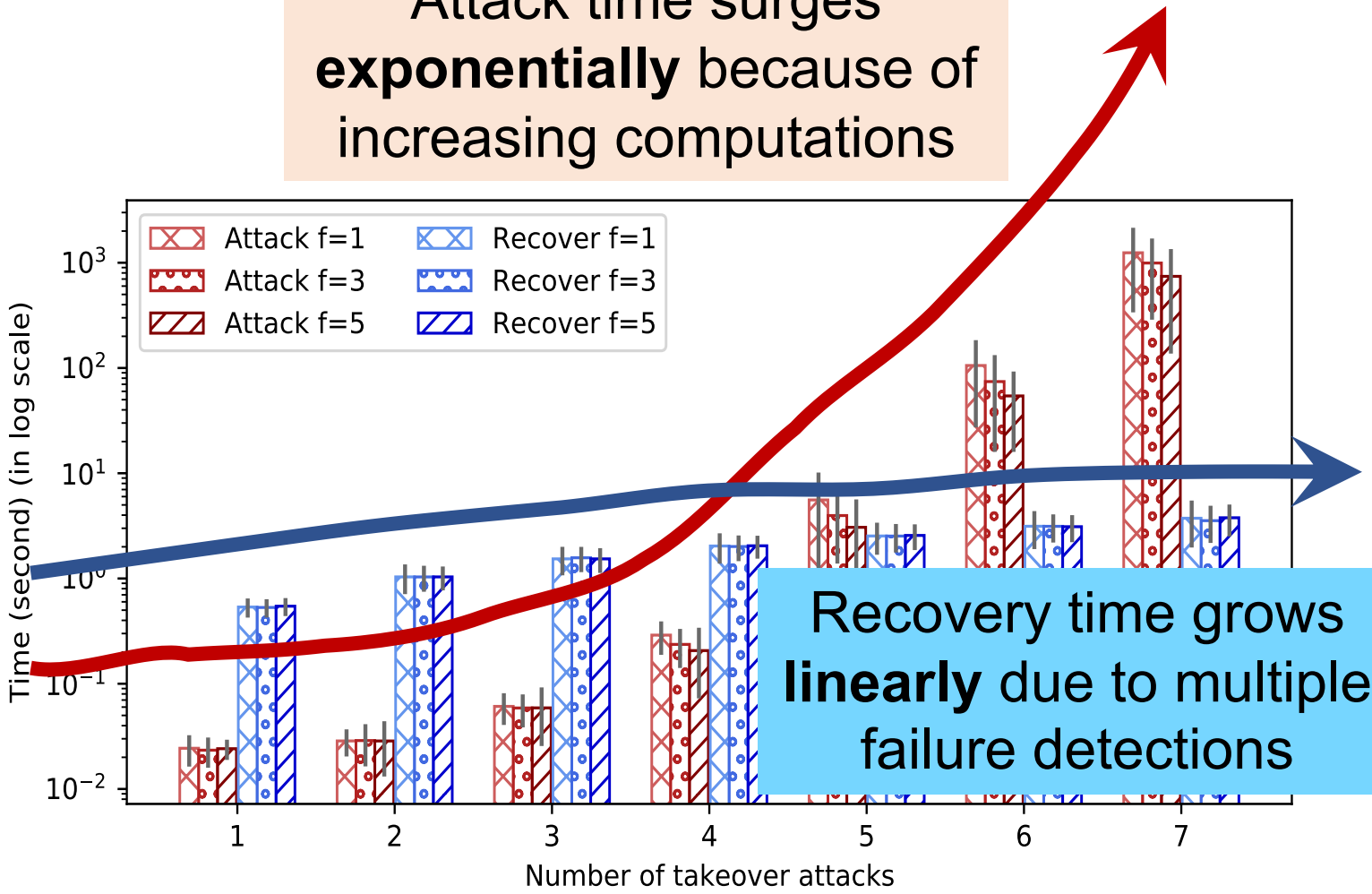


Election time of 20% of the runs is below 1 second

- Experiments conducted in a 16-server cluster on Compute Canada Cloud
- When threshold is less than 4, computation times are less than 100ms
- When threshold exceeds 5, time costs surge and diverge
- When threshold exceeds 7, performing computation takes hours

Malicious attacks vs. system recovery

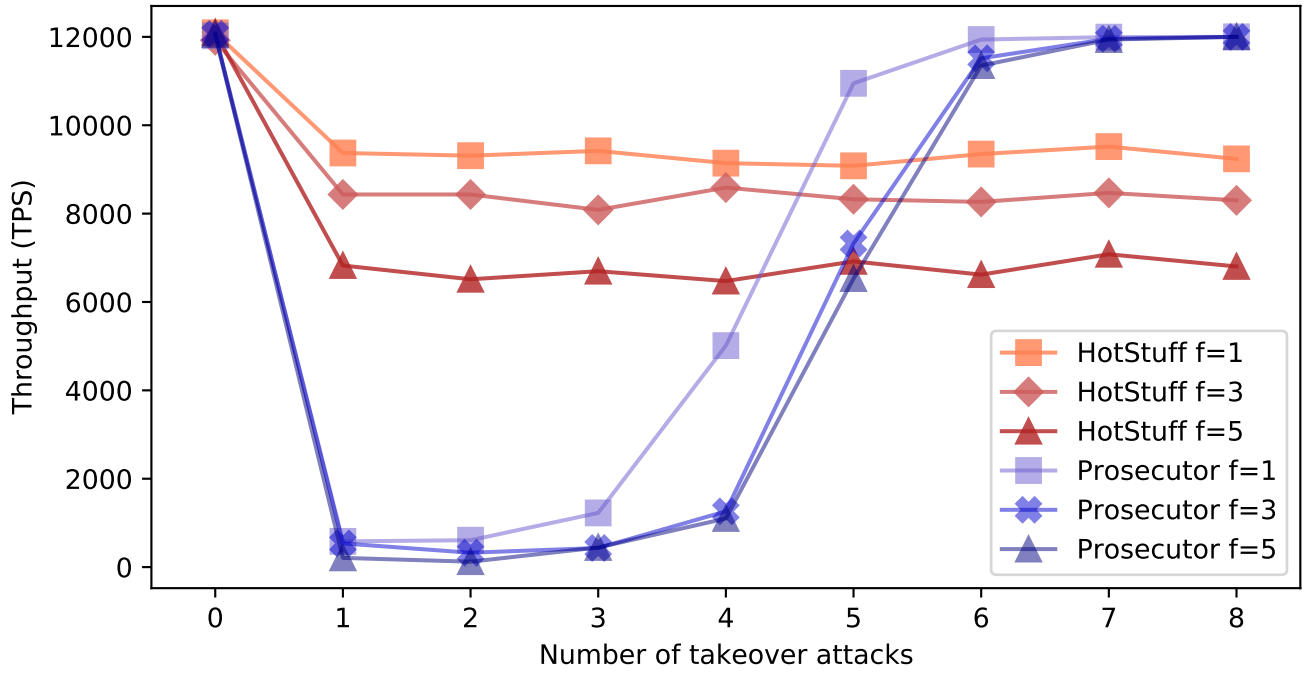
Attack time surges **exponentially** because of increasing computations



Recovery time grows **linearly** due to multiple failure detections

- The more attacks a server has launched, the more computation the server needs to perform for launching the next attack
- Time costs for system recovery are subjected to detecting faulty leaders (multiple timeouts) along with the number of attacks

Throughput under takeover attacks



- HotStuff rotates leadership in view changes; it mitigates the impact of faulty leaders but suffers from sustained throughput deductions
- Prosecutor gradually penalizes faulty servers and suppresses Byzantine servers over time
  - Byzantine servers vanish by having to perform time- and cost-consuming computations

Conclusions

- Inspired by Proof-of-Work and Raft leader election, Prosecutor establishes a penalization-based election mechanism that imposes computation work for leadership candidacy
- The amount of penalty (work) is determined by servers' past behavior
  - The more takeover attacks a server mounted in the past, the higher the penalty (i.e., the more computation work the server needs to bear)
- Byzantine servers vanish into performing computations after exhausting their computation capability

Thank you! Questions?

Gengrui Zhang

Email: gengrui.zhang@mail.utoronto.ca

Website: gengruizhang.github.io