# Scaling the Infrastructure of Practical Blockchain Systems

Scan to Download Dissertation Draft

Ted Yin // Cornell University

Ph.D. Dissertation Defense

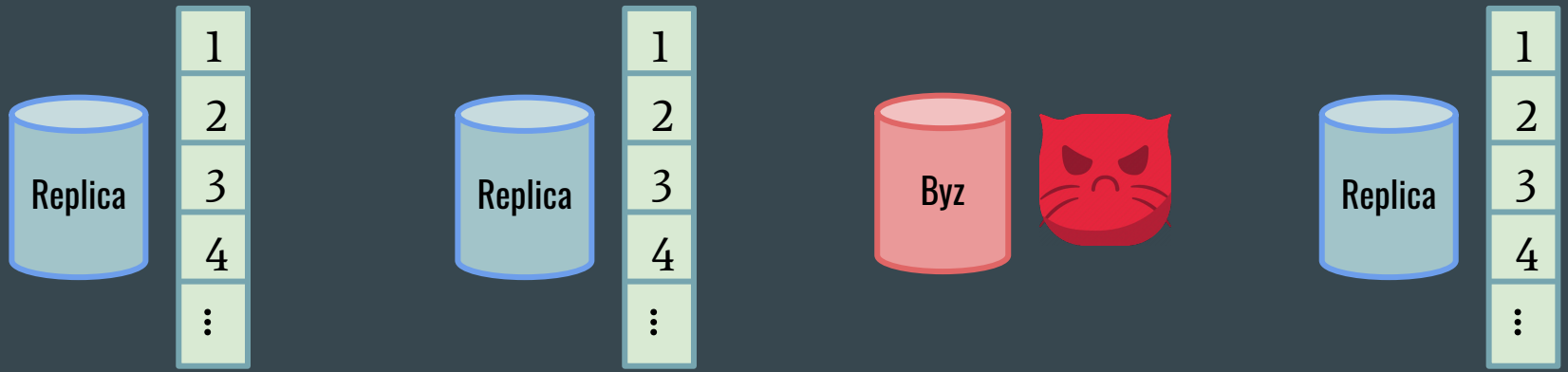
# Works in the Dissertation

- Practical BFT replication (consensus) protocols:
  - HotStuff: BFT Consensus in the Lens of Blockchain (shorter version in PODC '19)
  - Scalable and Probabilistic Leaderless BFT Consensus through Metastability ("Snow/Avalanche")

- The first attempt in persistent key-value stores:
  - CedrusDB: Persistent Key-Value Store with Memory-Mapped Lazy-Trie

# Byzantine Fault Tolerant State Machine Replication

- f out of all n nodes could exhibit arbitrarily faulty behavior
- *(Replica Coordination)* other n-f correct replicas receive and process the same sequence of requests
  - Two replicated sequence $s_1 \subseteq s_2 \vee s_2 \subseteq s_1$
- Safety: agreement
- Liveness: termination
- "Consensus"



# Model & Known Solutions

- Impossibility (FLP '83)

> "In this paper, it is shown that every protocol for this problem has the possibility of non-termination, even with only one faulty process."

# Model & Known Solutions

- Termination → “Probability of 1”

  → Asynchronous model (Ben-Or ’83)

- Always safe no matter what, and terminate when network is synchronized

  → Partially synchronous model (DLS ’88)

- Use synchronous assumption

  → Synchronous model (LSP ’82)

# Model & Known Solutions

- Asynchronous model
  - Randomization to break symmetry
  - Examples: Ben-Or ('83): exponential termination time; HoneyBadgerBFT ('16): $\Omega(n^3)$

- Partially synchronous model
  - Leader-based
  - Examples: PBFT ('99): $O(n^2)$ on a "good" day, Paxos ('90, CFT), Raft ('14, CFT)

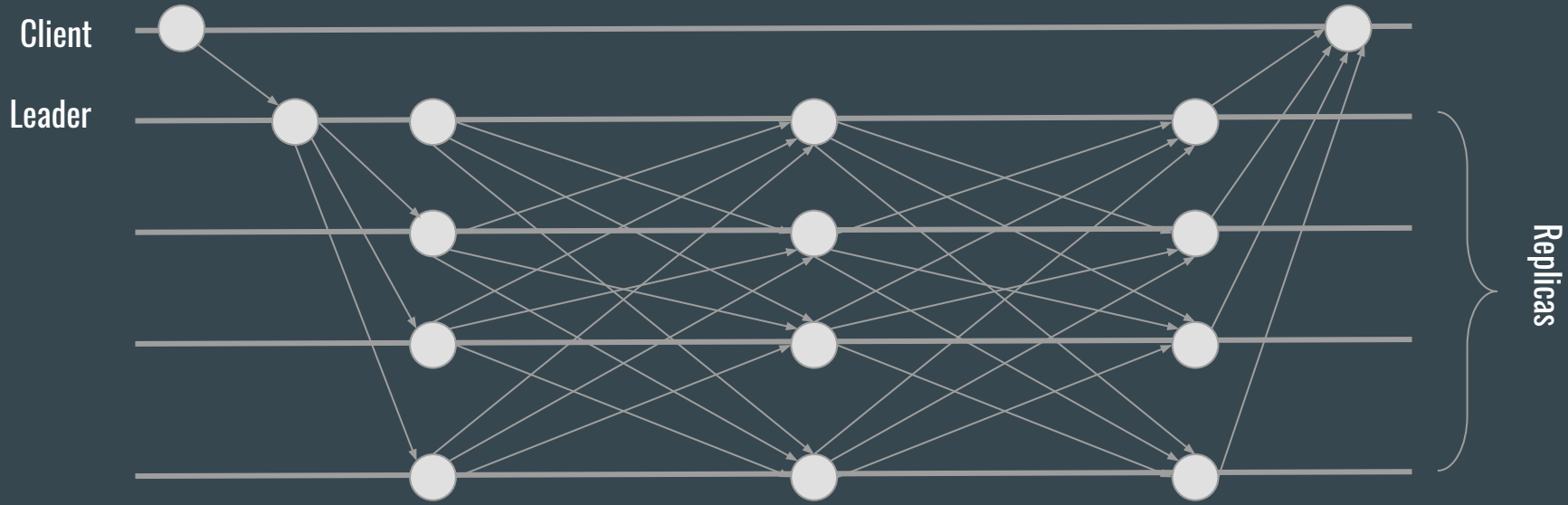
# Theory vs Practice

- Asynchronous model
  - Without a leader — everyone proposes; the best was $O(n^3)$ until recently: VABA ('19)
  - "Expected convergence" — non-deterministic number of rounds

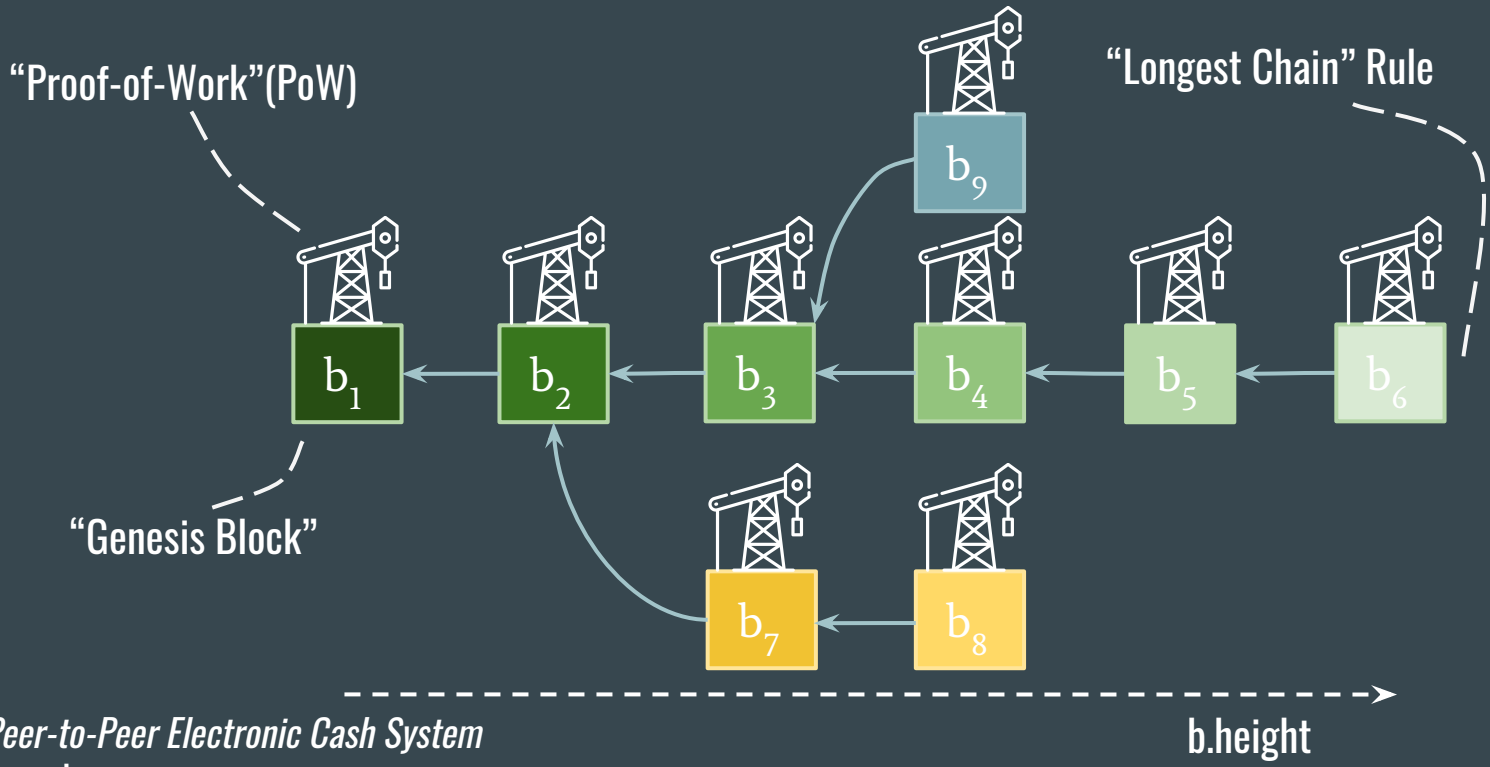
- Partially synchronous model:
  - Constant number of rounds on a good day
  - $O(n^2)$ on a good day is still far from the benign counterpart
  - Complicated and subtle operational logic
  - The leader becomes the new bottleneck

# Paradigm: Quorum/Vote-Based

Example: PBFT



# Paradigm: Nakamoto Consensus



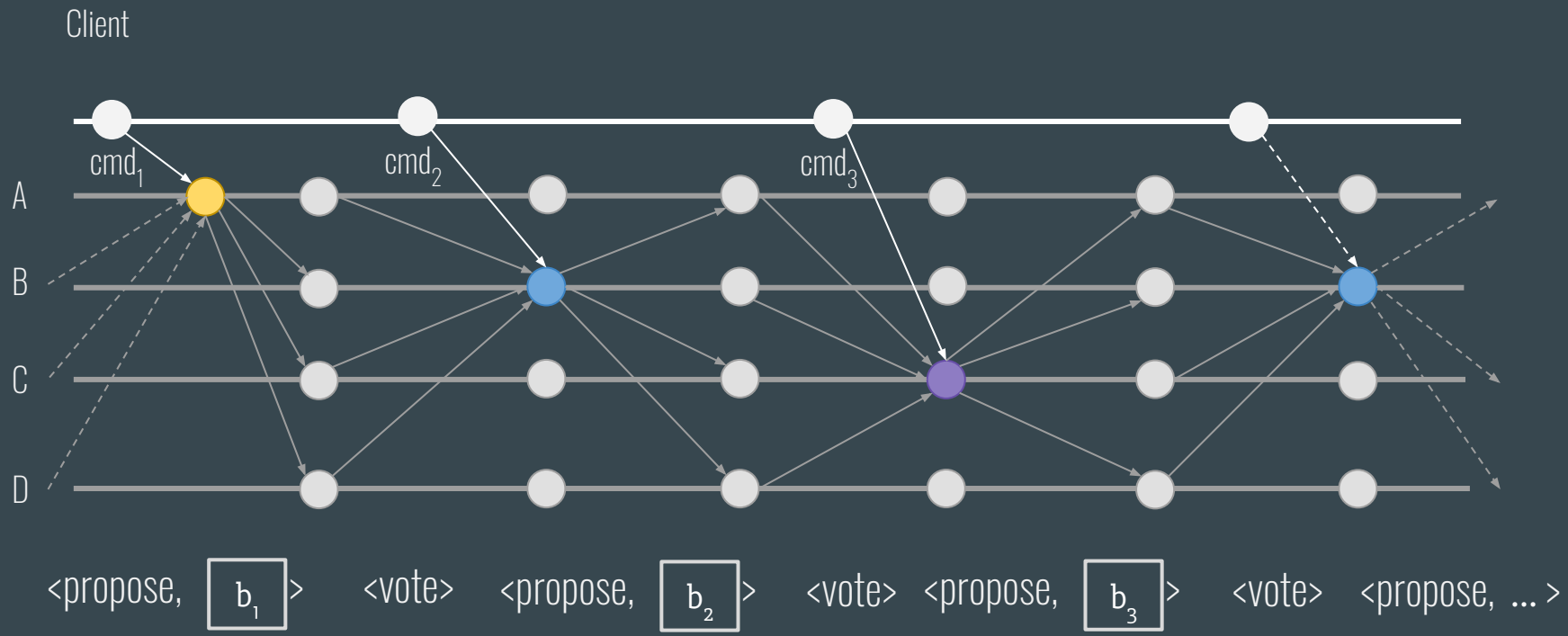

*Bitcoin: A Peer-to-Peer Electronic Cash System*
Satoshi Nakamoto

# HotStuff

Joint work with Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, Ittai Abraham

- Bridge the two paradigms!
- Encode consensus knowledge entirely with chain topology
- “Blockchain-style”
  - Immutable and verifiable history
  - Make consensus on the prefix of tentative SMR trace
  - No special treatment of view change
- Solve an open problem: O(n) in both optimistic case and view change (per leader failure, with responsiveness)

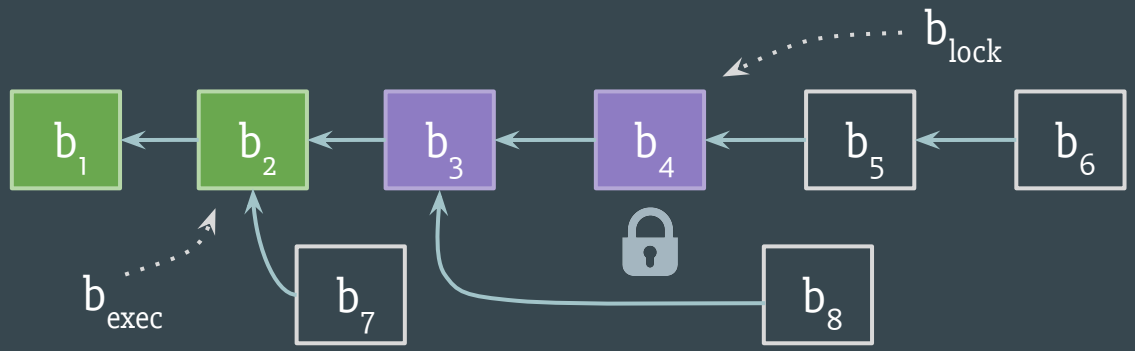
# HotStuff: Communication Pattern



# HotStuff: The Protocol

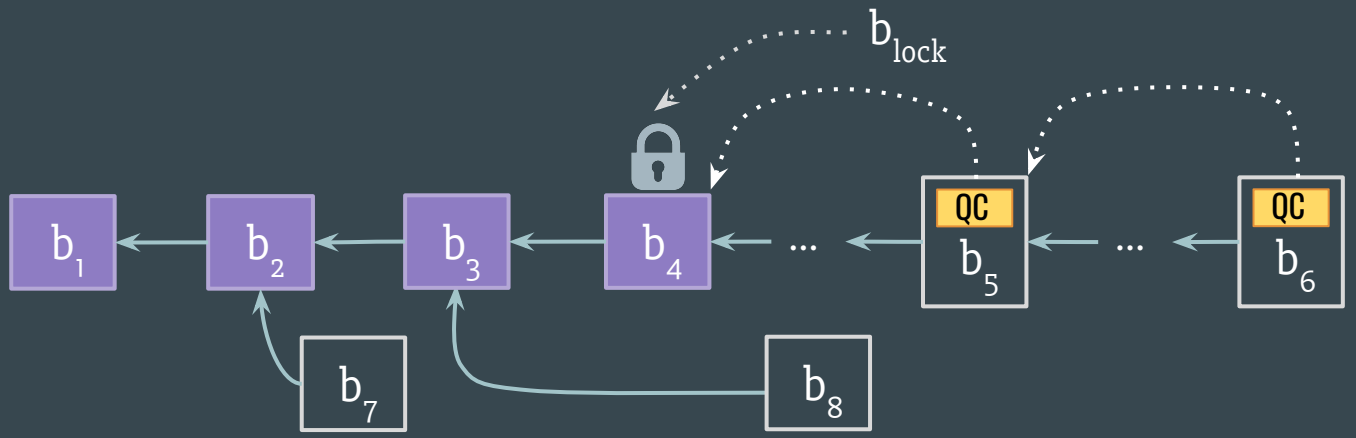
## Protocol State Variables

- $b_{lock}$ = block leading the preferred branch
- $b_{exec}$ = last committed block
- vheight = height of the block last voted for



# “Longest” Chain Rule: Branch Preference

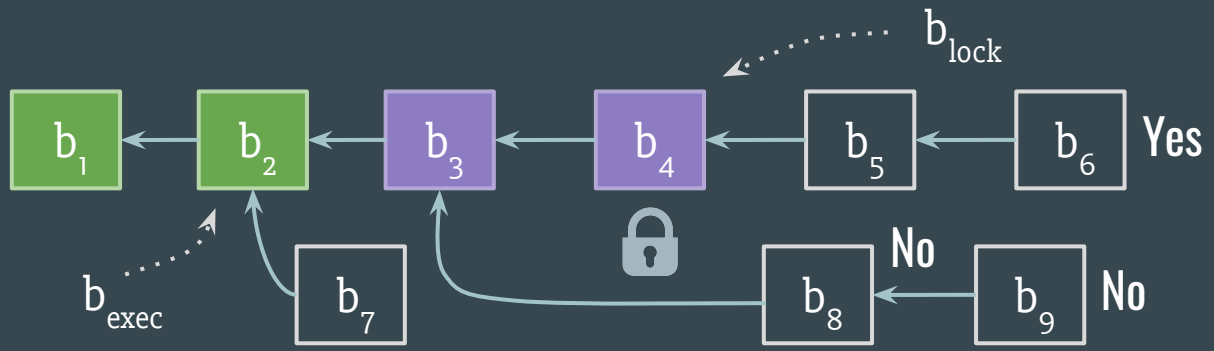
- Quorum Certificate (QC): a proof of 2f+1 votes for a block, a “phase”
- $b_{lock}$: the “locked block” that leads the preferred branch
- Lock mechanism (preference): a replica only votes for the block on the preferred branch...



# HotStuff: Voting

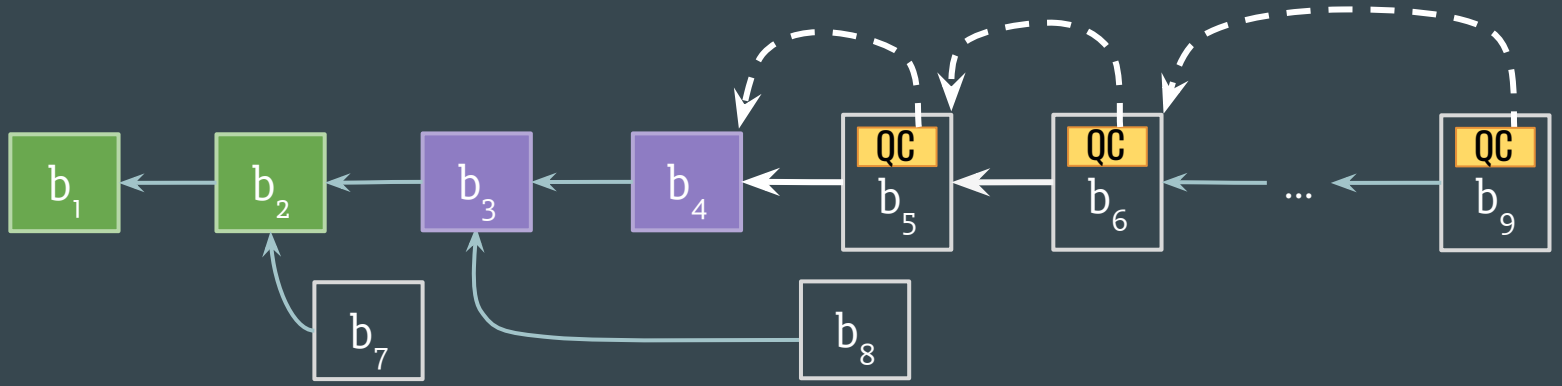
## How to Vote?

- Only vote for $b_{new}$ if the following constraints hold:
  - $b_{new}$.height > vheight
  - ($b_{new}$ is on the same branch as $b_{lock}$ ) or ($b_{new}$ .justify .node.height > $b_{lock}$.height)



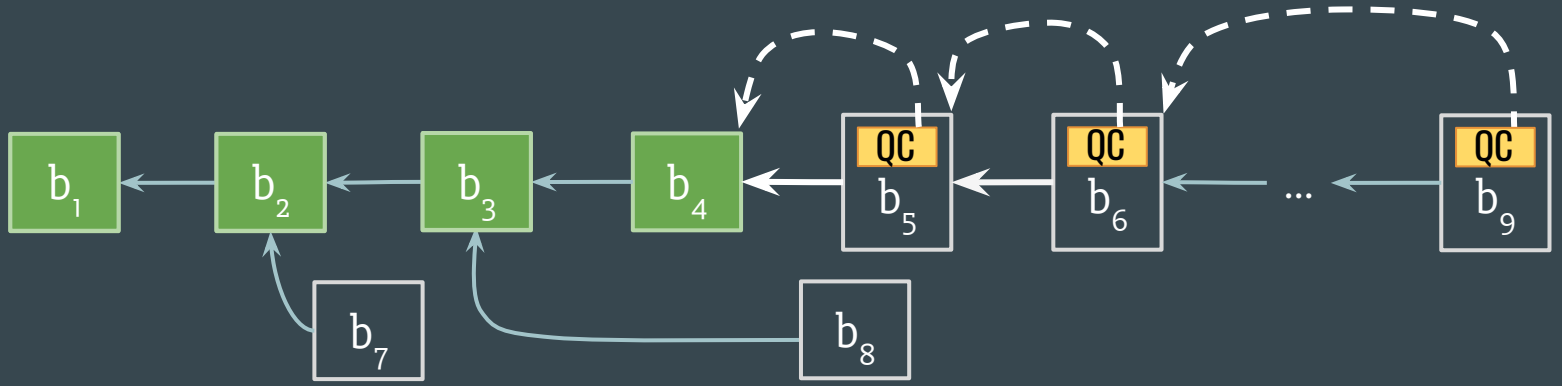
# HotStuff: Decision

## When to Commit? (w.l.o.g. by illustration)

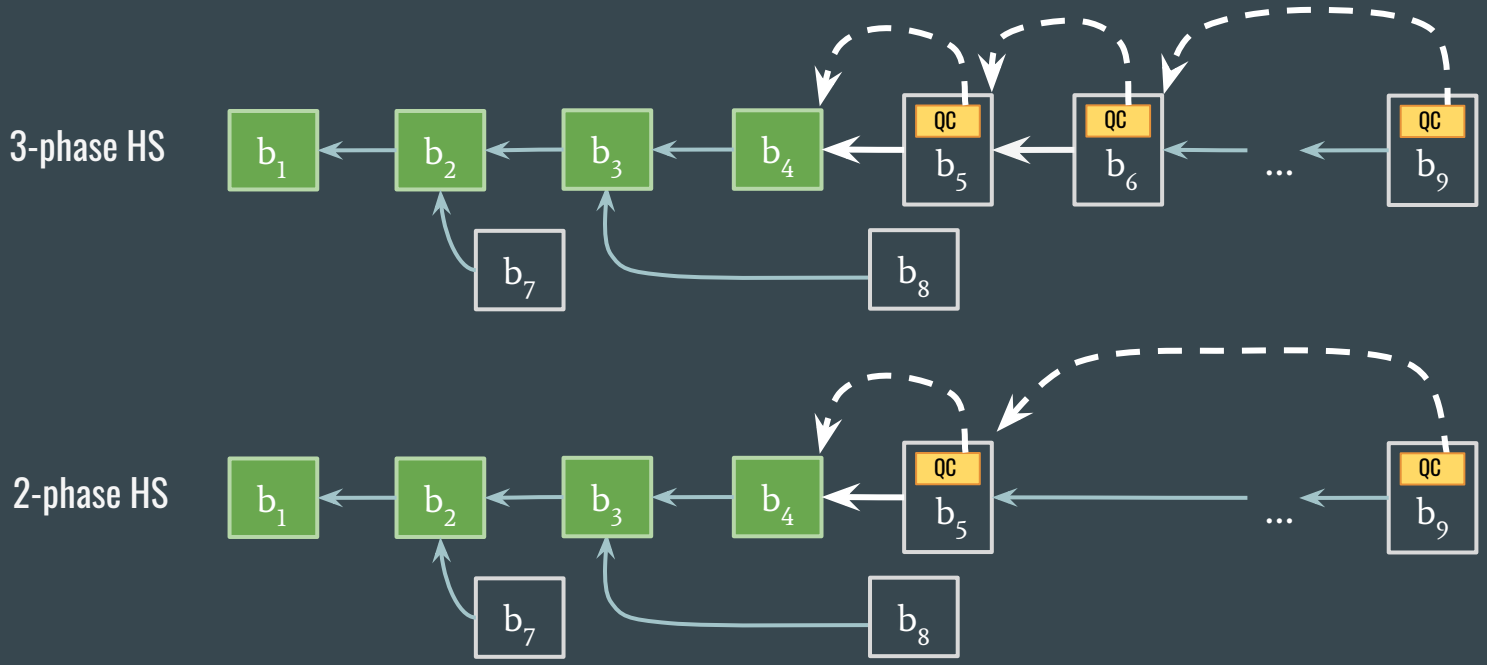


# HotStuff: Decision

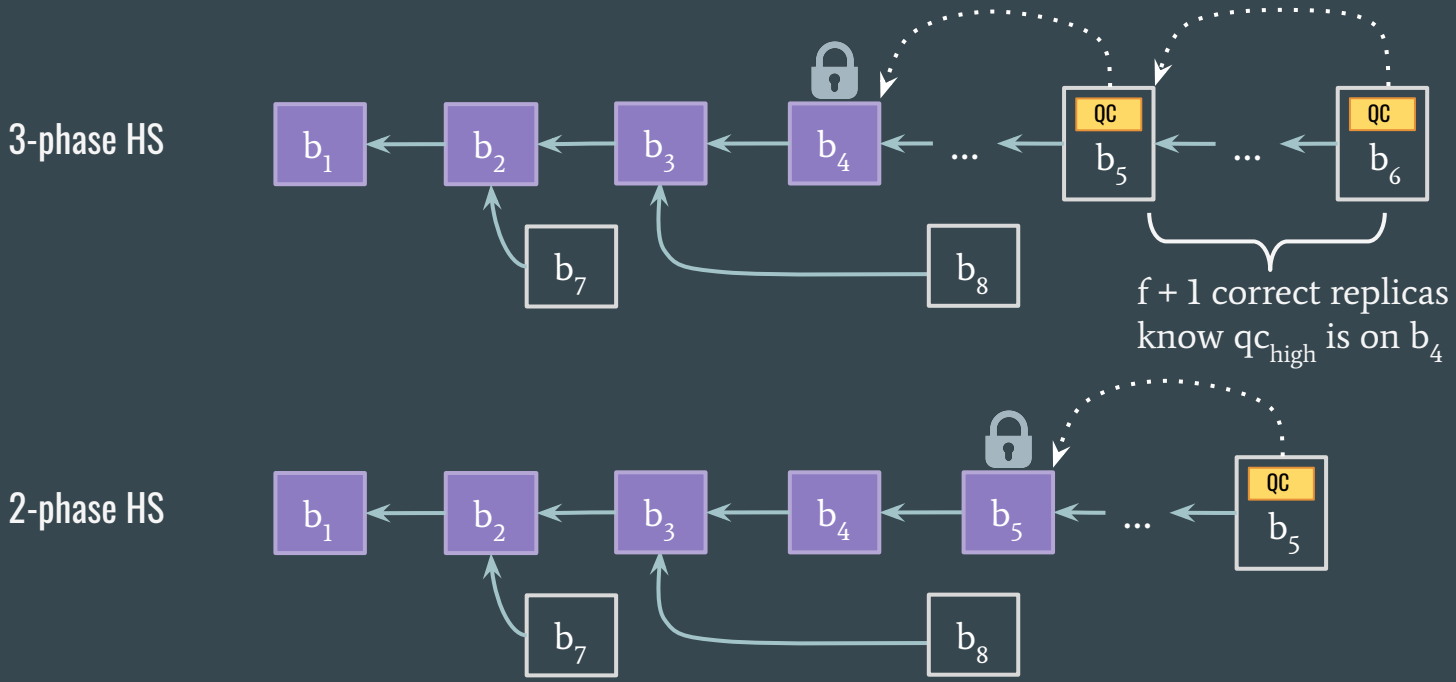
## When to Commit? (w.l.o.g. by illustration)



# HotStuff Framework: Commit Rule

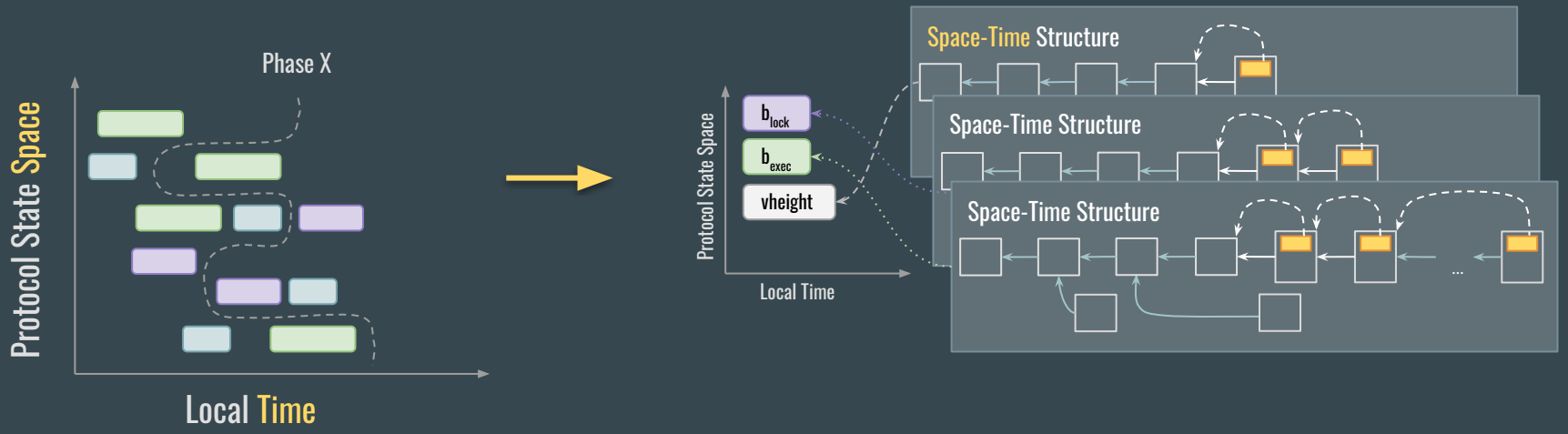


# HotStuff Framework: Branch Preference



# HotStuff: The Secret Sauce

- Keep time-independent data structures
- Reason at a “higher dimension”: space-time structure



# HotStuff: Main Contributions

- Theory:
  - The first partially synchronous protocol that achieves O(n) lower bound in presence of a leader failure (*w/ optimistic responsiveness)
  - Inspired other protocols such as VABA (async.), Sync HotStuff (sync.)
- Engineering:
  - Protocol safety is just about checking conditions of the tree topology
  - No explicit view change handling for safety
  - Safety decoupled from liveness: “Pacemaker”
    - Safety part ~ 200 C code
    - Libra/DiemBFT is an instantiation of the Pacemaker concept!
  - Prototype implementation

# What We Didn't Anticipate...

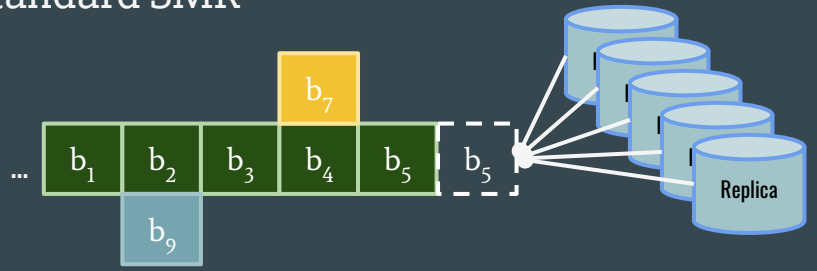
libra

facebook

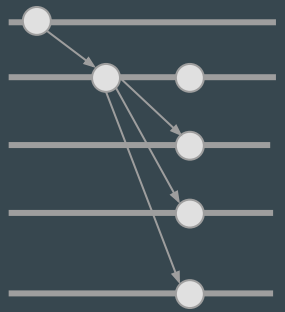
diem

# What Prevents Quorum-based Consensus from Scaling

- The strong consistency requirement by standard SMR
  - Leadership helps
  - Relax the model (e.g. EPaxos '13)




- At least some node needs to exchange messages to the vast majority of others
  - Even in HotStuff, the leader has to broadcast
  - Is it inevitable?...



# Snow/Avalanche

Joint work with Kevin Sekniqi, Robbert van Renesse, Emin Gün Sirer.
The credit of theoretical analysis goes to Kevin.

- Consensus by “the Congress vs. social media”
- Peer-to-peer gossip in a BFT environment
- Sample the network!

US Congress: 100 + 435

US Social Media: 231 Million

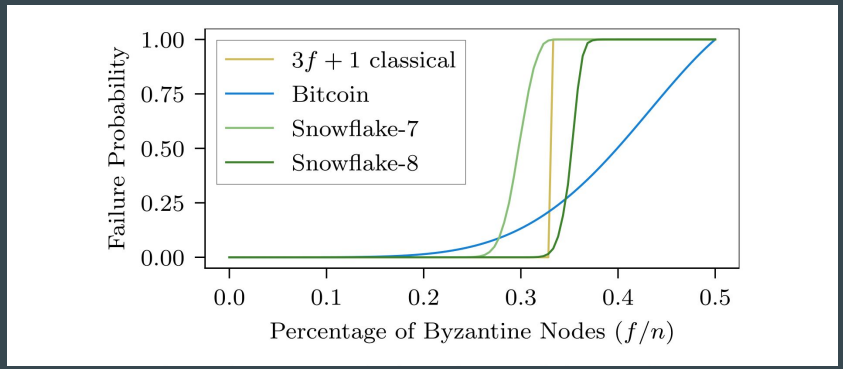
# Snow/Avalanche

Joint work with Kevin Sekniqi, Robbert van Renesse, Emin Gün Sirer.
The credit of theoretical analysis goes to Kevin.

- Consensus by “the Congress vs. social media”
- Peer-to-peer gossip in a BFT environment
- Sample the network!
- Not a “free lunch”:
  - Like Bitcoin: trade off deterministic safety for a probabilistic one
  - Requires more synchrony in the network, but not as strong as in “lockstep” protocols

US Congress: 100 + 435

US Social Media: 231 Million




Bitcoin is with the well-known “six-block” rule.
Snowflake configured with k = 10, β = 150. Snowflake-7,8 uses α = 7 and α = 8 respectively (see the dissertation for the definition of k, α and β).

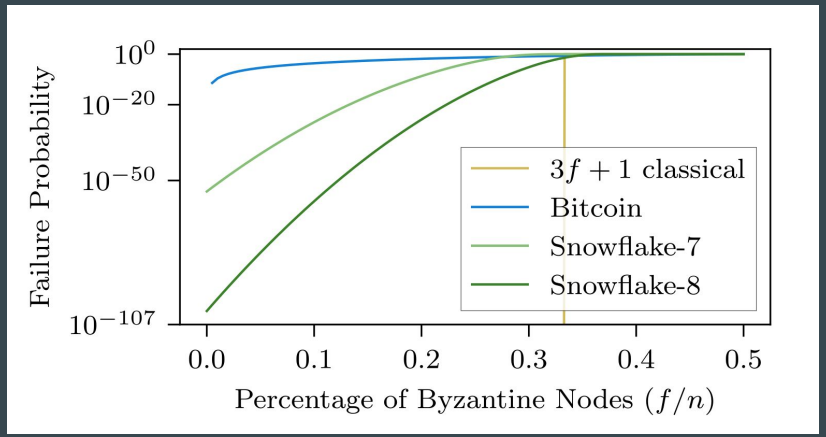
# Snow/Avalanche

Joint work with Kevin Sekniqi, Robbert van Renesse, Emin Gün Sirer.
The credit of theoretical analysis goes to Kevin.

- Consensus by “the Congress vs. social media”
- Peer-to-peer gossip in a BFT environment
- Sample the network!
- Not a “free lunch”:
  - Like Bitcoin: trade off deterministic safety for a probabilistic one
  - Requires more synchrony in the network, but not as strong as in “lockstep” protocols

US Congress: 100 + 435

US Social Media: 231 Million

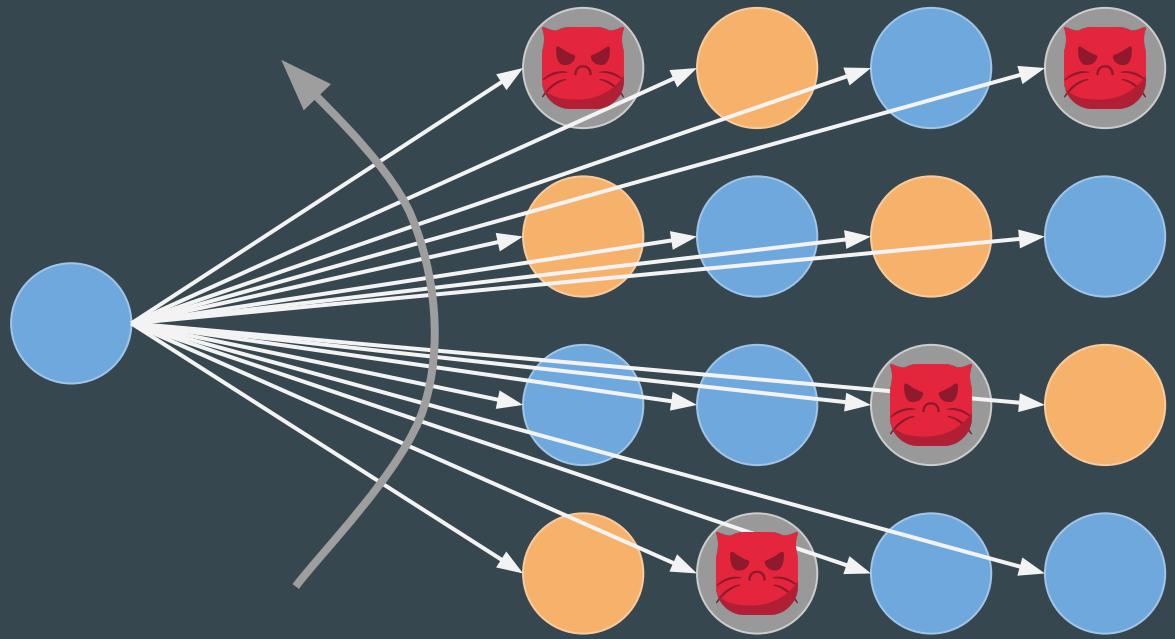


# Snow/Avalanche

- Consensus by “the Congress vs. social media”
- Peer-to-peer gossip in a BFT environment
- Sample the network!
- Not a “free lunch”
- A feedback loop: Sample→Adjust →Sample→...
- Positive feedback: a binary consensus where fluctuation in initial preferences triggers a “toppling” effect

# Full Broadcast to Partial Sampling.

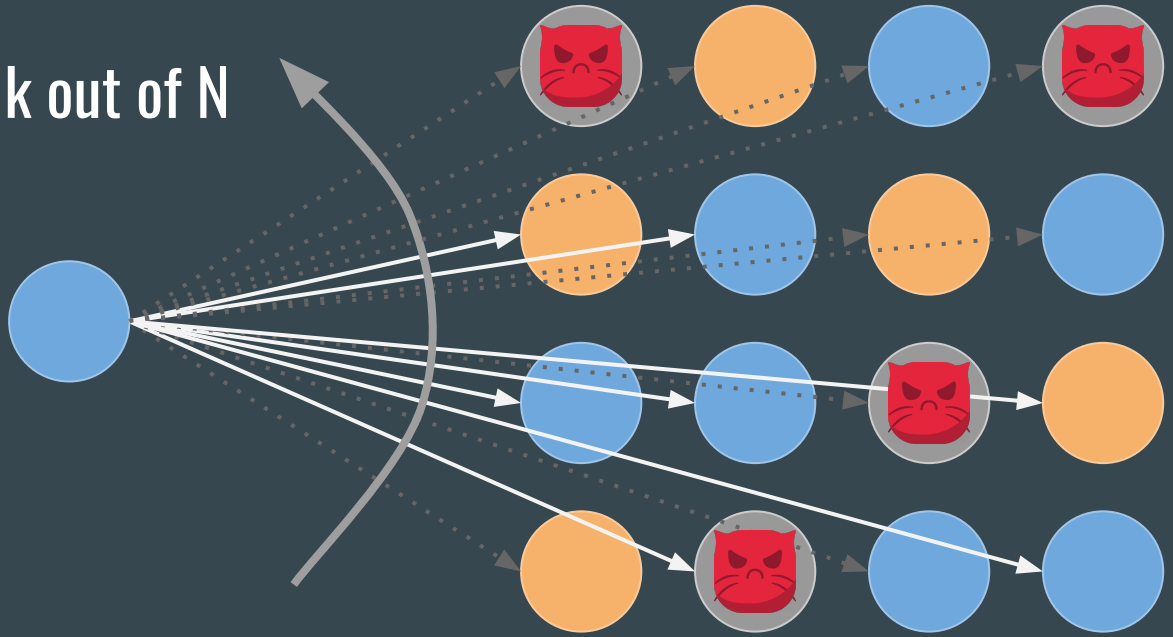
Query all

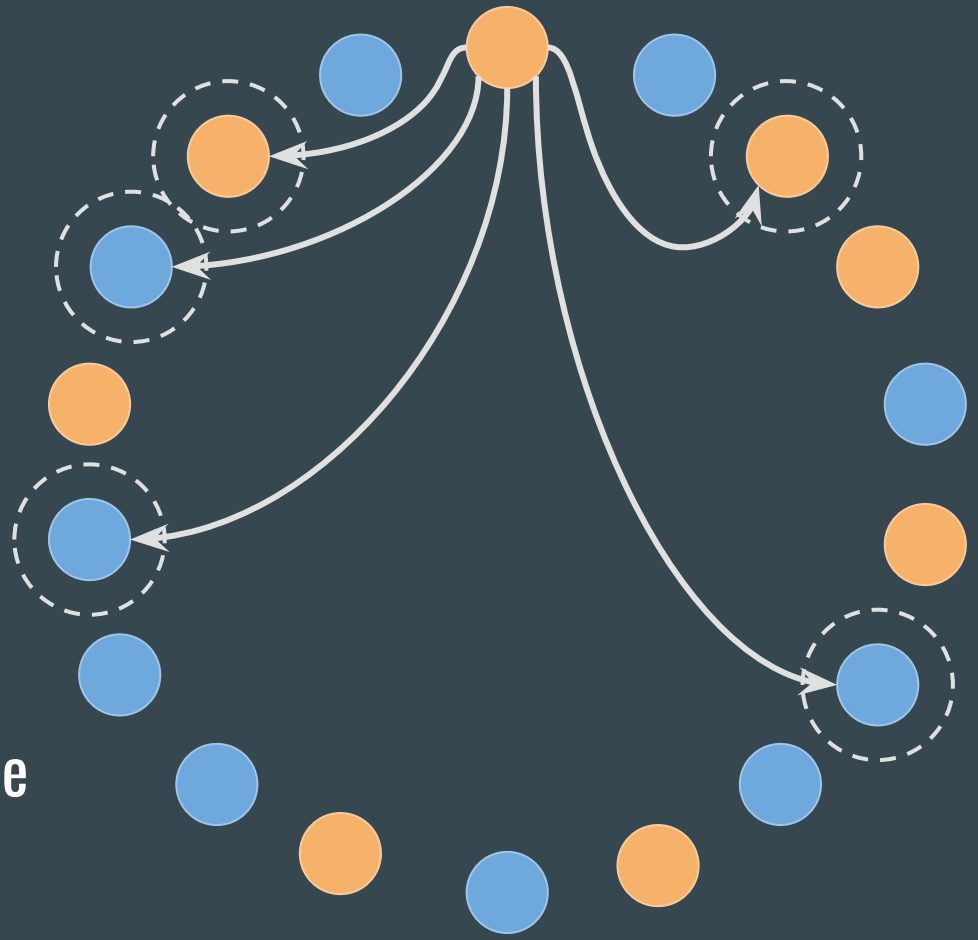


# Full Broadcast to Partial Sampling.

~~Query all~~

Only Sample k out of N





Alice asks
5 other people
randomly

Alice: “Blue is the majority answer!”

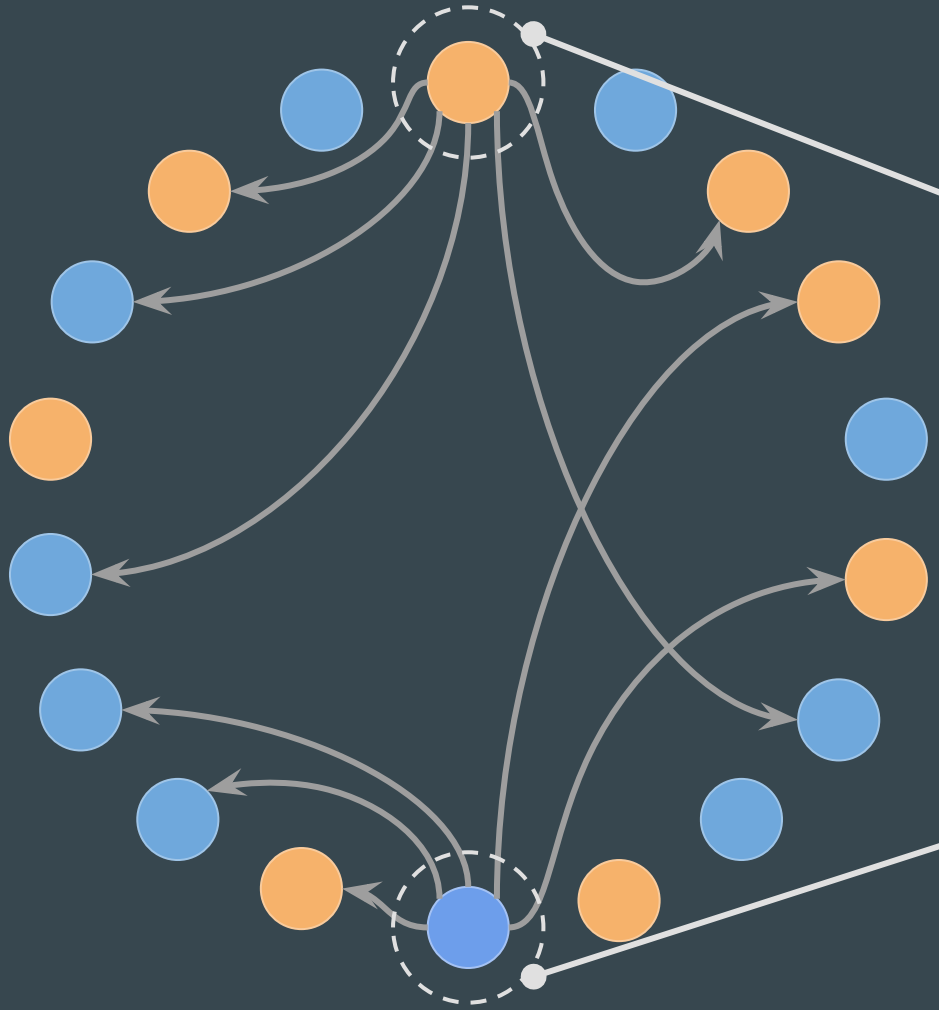
Bob asks
5 other people
randomly

Alice: “Blue is
the majority
answer!”

Alice asks
5 other people
randomly

Bob: “Yellow is
the majority
answer!”

Bob asks
5 other people
randomly

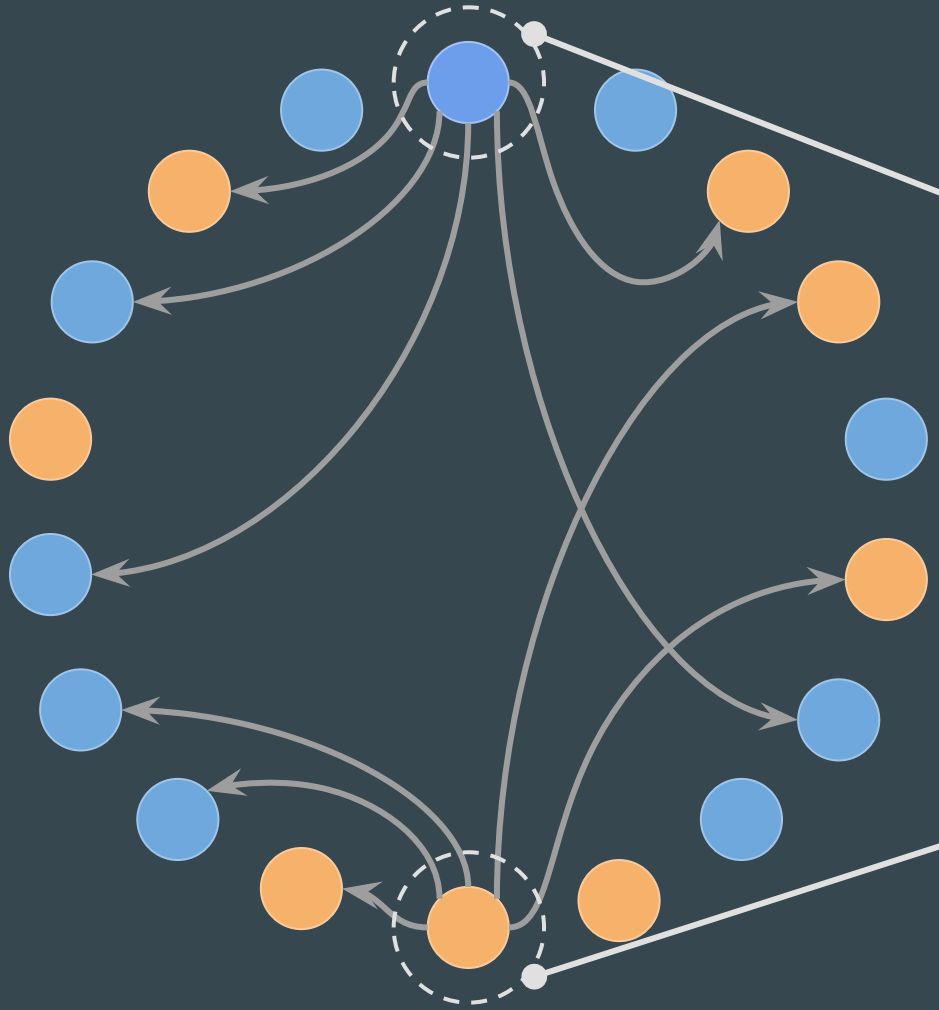


Alice: “Blue is the majority answer!”

Alice asks
5 other people
randomly

Bob: “Yellow is the majority answer!”

Bob asks
5 other people
randomly



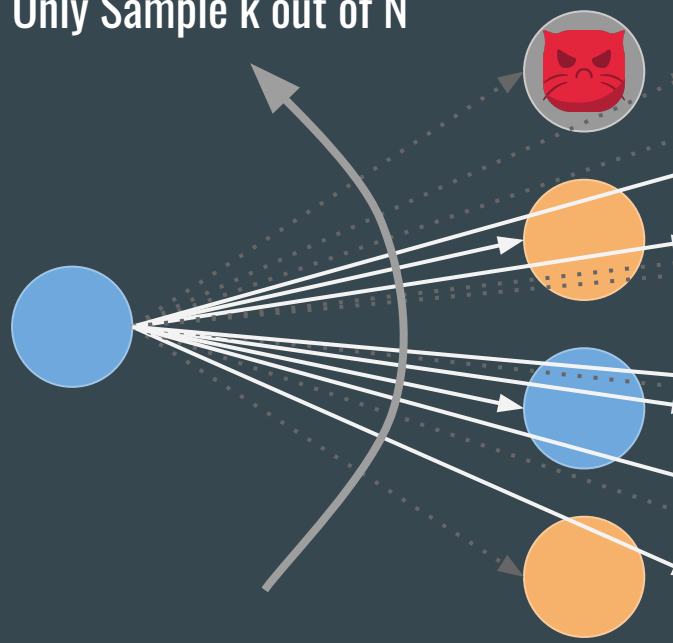
Alice: “Blue is the majority answer!”

Alice asks
5 other people
randomly

Bob: “Yellow is the majority answer!”

# Snowflake: Epidemic Loop

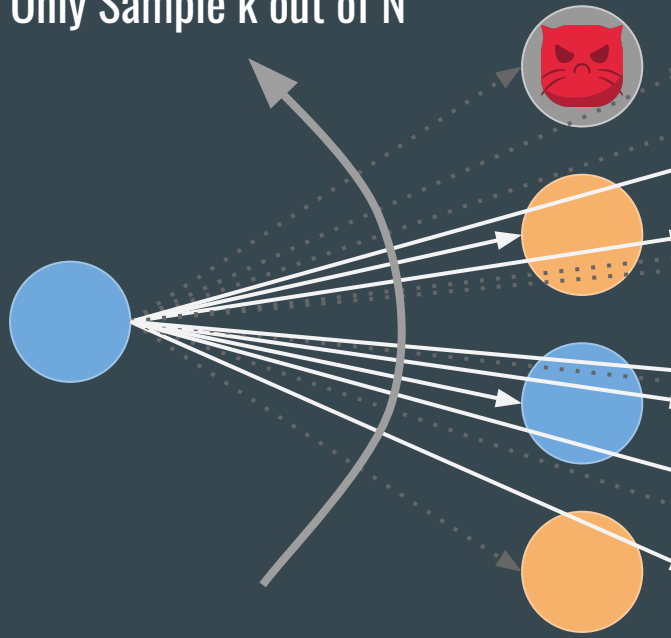
1. Sample k peers uniformly at random
2. If αk agrees on a color c':
   a. If c' is the same as the current c:
      i. Increase the *counter*
   b. Else:
      i. c := c'
      ii. Reset the *counter*
3. Go to line 1



# Snowball: Adding Confidence

1. Sample k peers uniformly at random
2. If αk agrees on a color c':
   a. Increase the *confidence* d[c'] for c'
   b. If d[c'] > d[c]
      i. c := c'
   c. If c' is not the last color gets αk
      i. Reset the counter
   d. Else:
      i. Increase the counter
3. Go to line 1

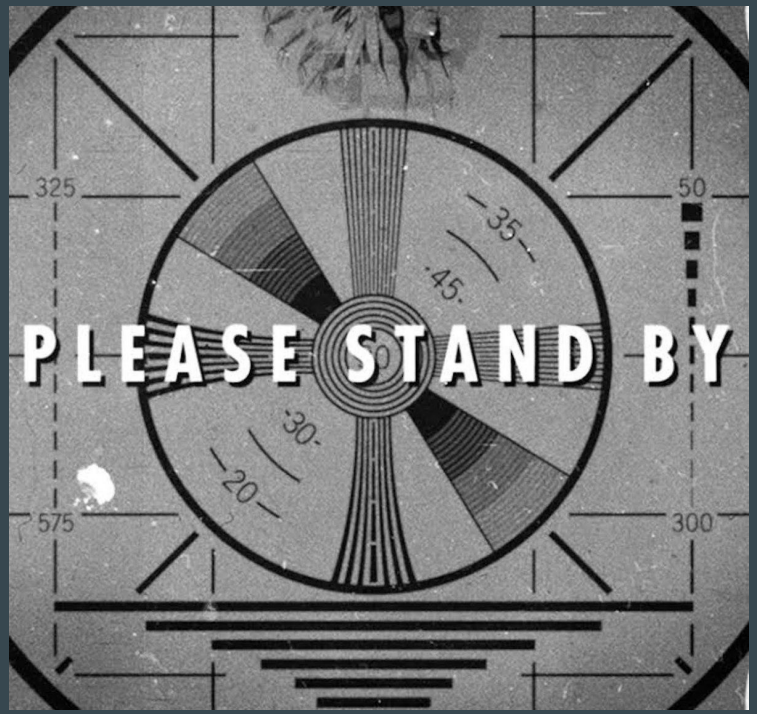
Only Sample k out of N



Converge in O(logN) time
Better complexity?

# Snowball: Demo

https://tedyin.com/archive/snow-bft-demo

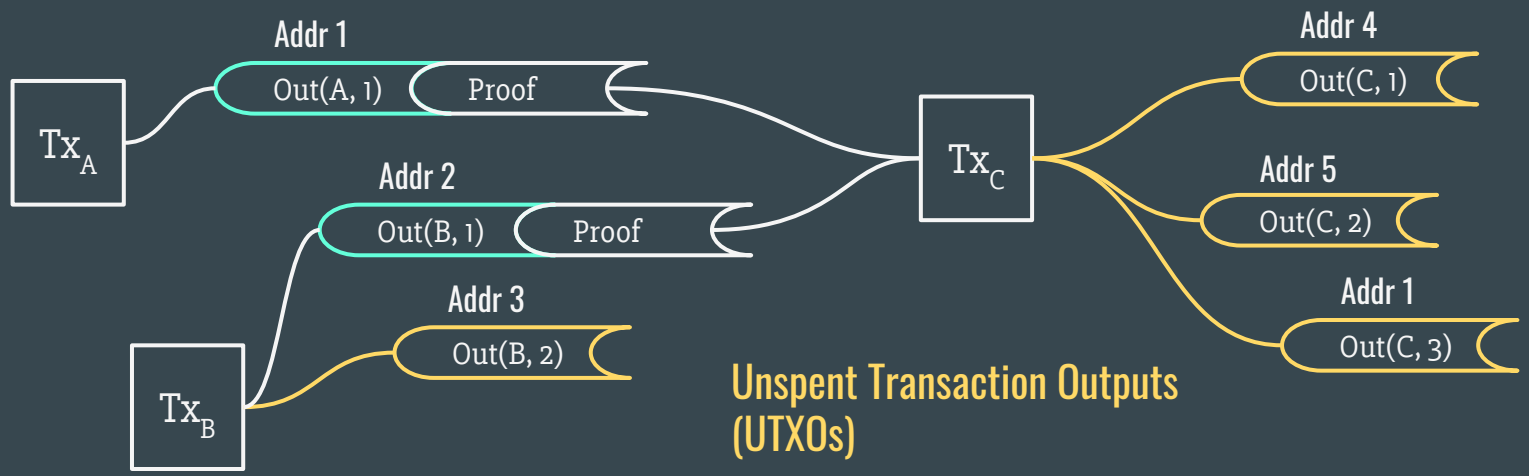


# Snow → Avalanche

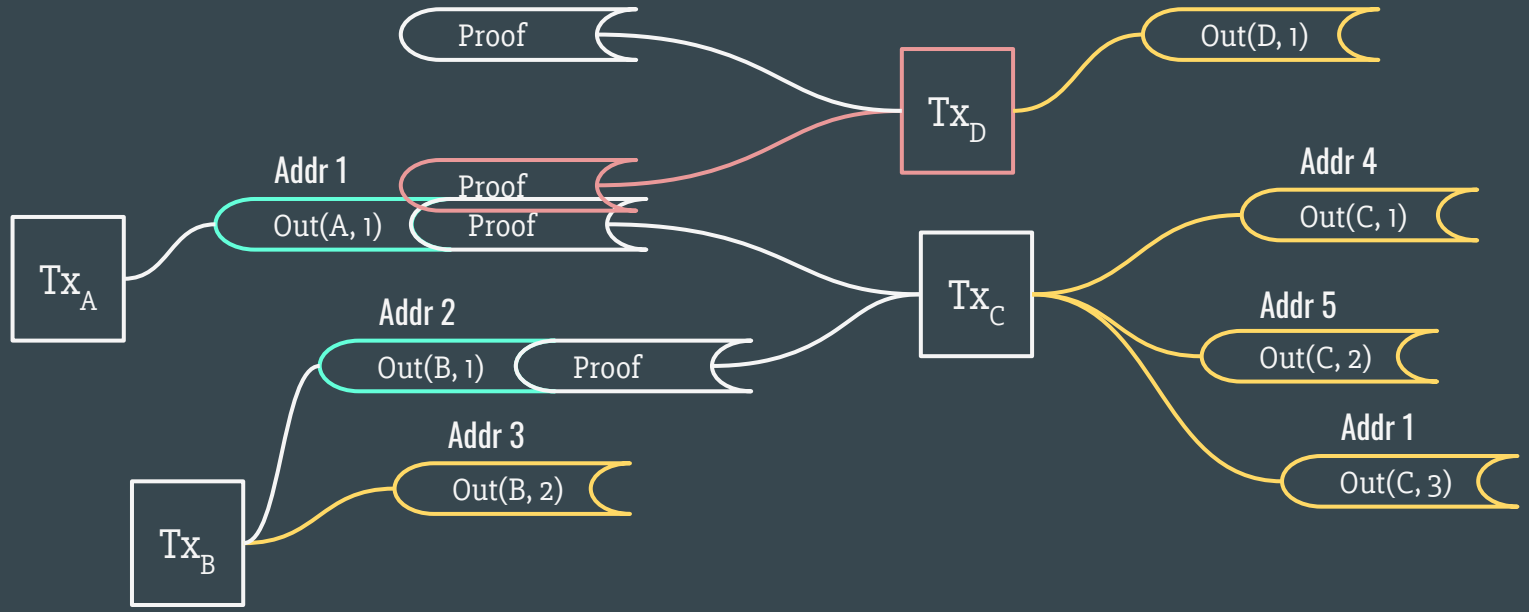
- Key observation: standard replication is not necessary in a payment system
  - UTXO ensures a verifiable flow of spending
  - The only missing piece is “conflict resolution” — weaker than “consensus”

# Snow → Avalanche

- Key observation: standard replication is not necessary in a payment system

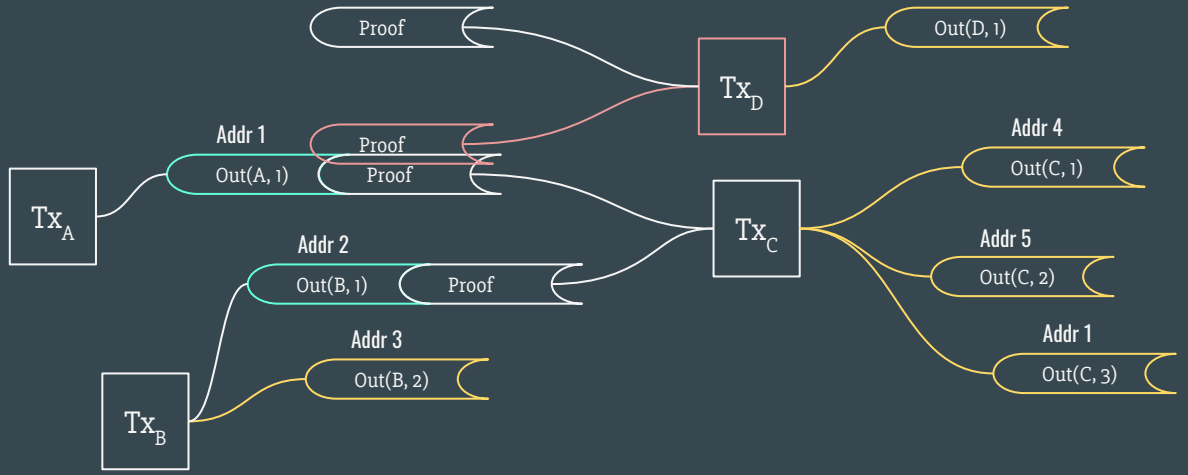


# Snow → Avalanche

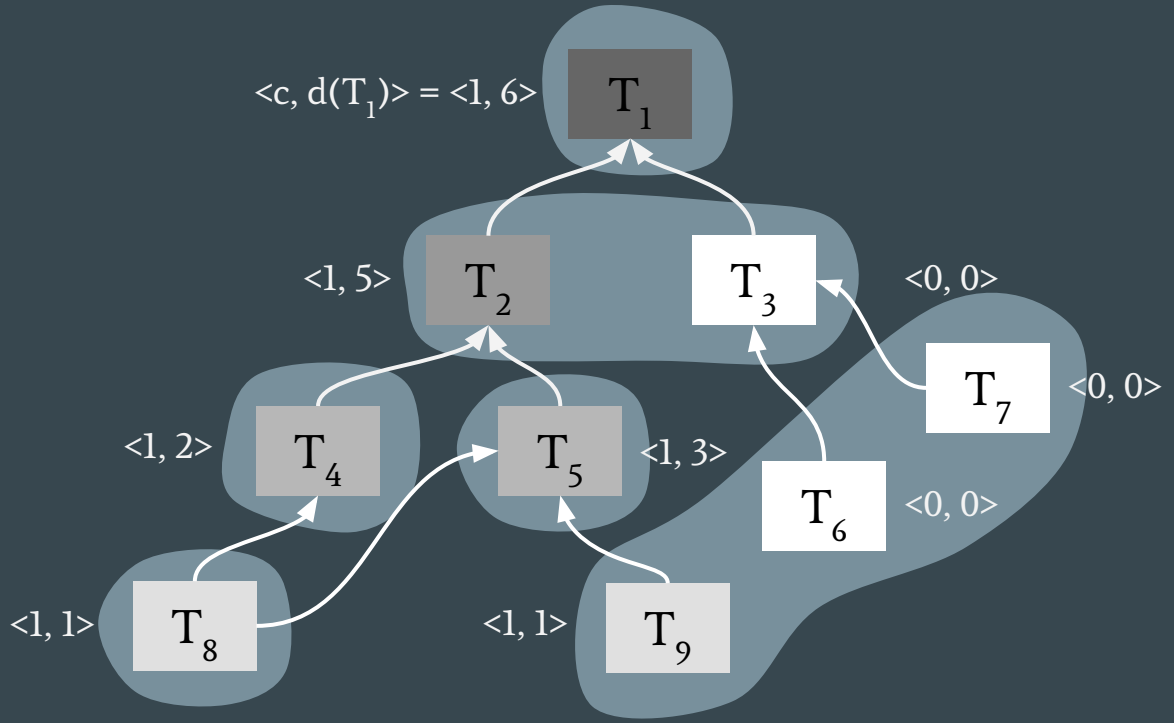


# Snow → Avalanche

- Key observation: standard replication is not necessary in a payment system
- Weakening the liveness:
  - Honest spenders → “triviality case” in consensus
  - Malicious spenders → may get stuck forever!



# Avalanche: Cascading the Sampling Process




# Avalanche: Multiple Inputs

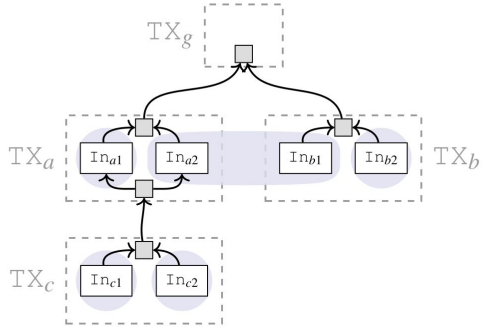
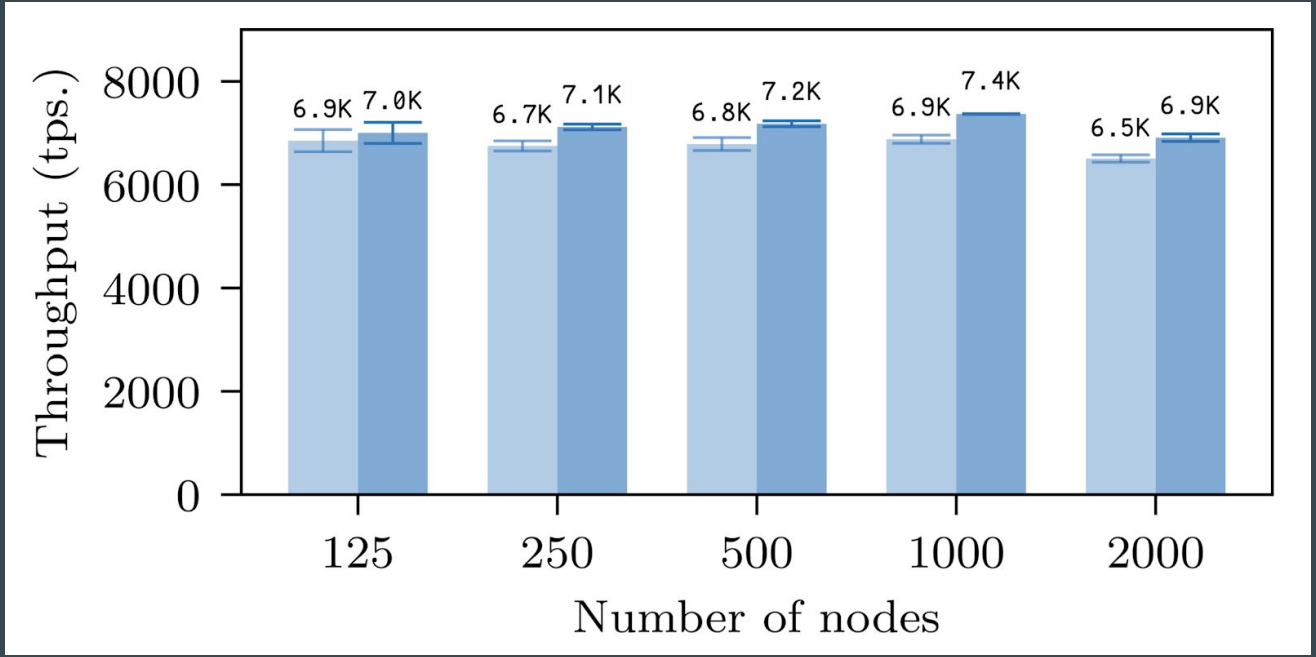
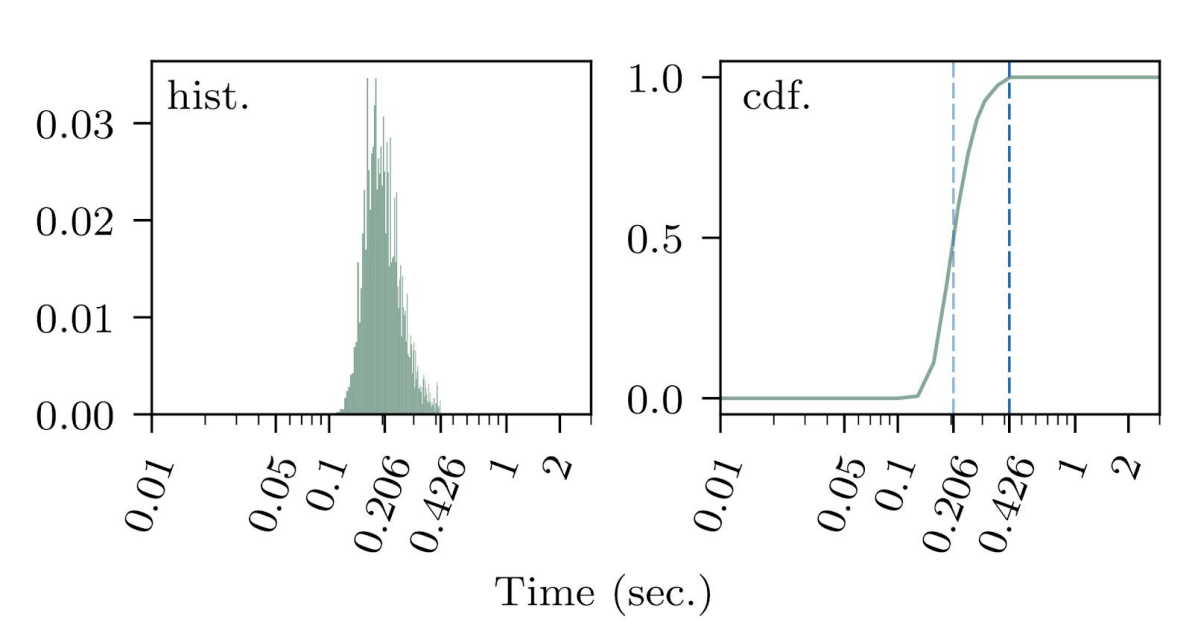



Figure 5.5: The underlying logical DAG structure used by Avalanche. The tiny squares with shades are dummy vertices which just help form the DAG topology for the purpose of clarity, and can be replaced by direct edges. The rounded gray regions are the conflict sets.

# Avalanche: Throughput



# Avalanche: Latency



# Avalanche: Geo-replication

In a more realistic setting:

- 2000 nodes in 20 cities across the globe
- All nodes directly participate in consensus
- Full signature verification

Our evaluation results:

- ~3400 tps
- ~1.35 sec

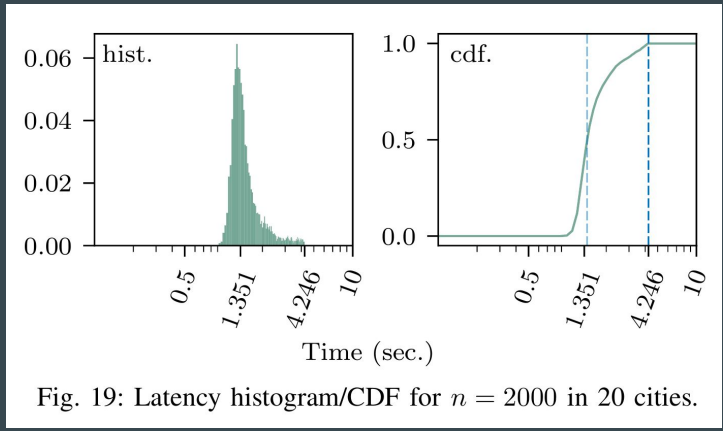


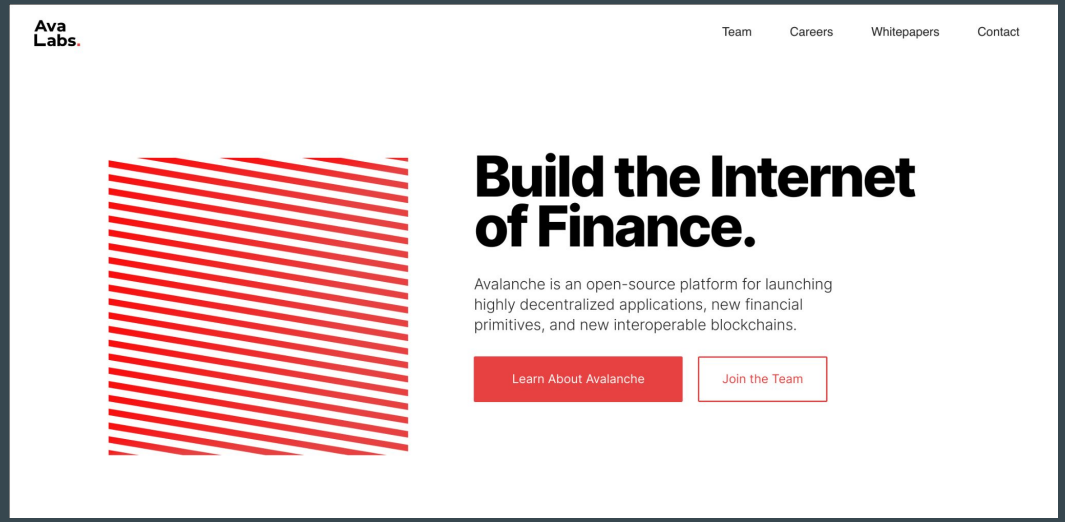

Fig. 19: Latency histogram/CDF for $n = 2000$ in 20 cities.

# Snow/Avalanche: Main Contributions

- Snow, a family of binary BFT protocols
  - Consensus-by-sampling, a new paradigm and perspective
  - Quiescent and green, like quorum-based protocols
  - Probabilistic safety, loosely organized, like Nakamoto consensus
  - Scalable because of the sampling!

- Avalanche, a decentralized, peer-to-peer payment system
  - Combine multi-value Snowball with UTXO to allow a fully functional payment system
  - Prototype implementation
  - Got commercialized and deployed worldwide with ~1000 validators

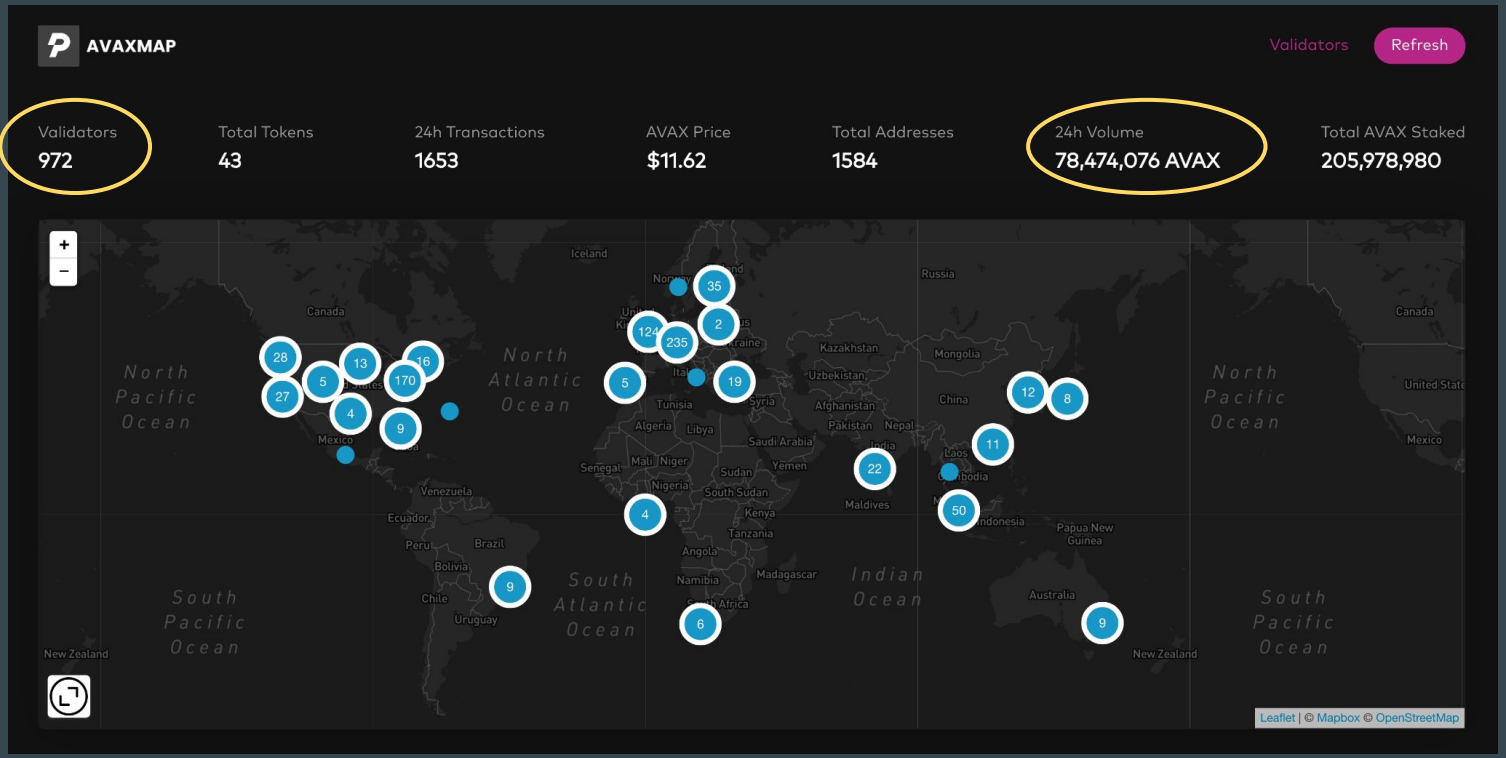
# What We Didn't Anticipate...Again



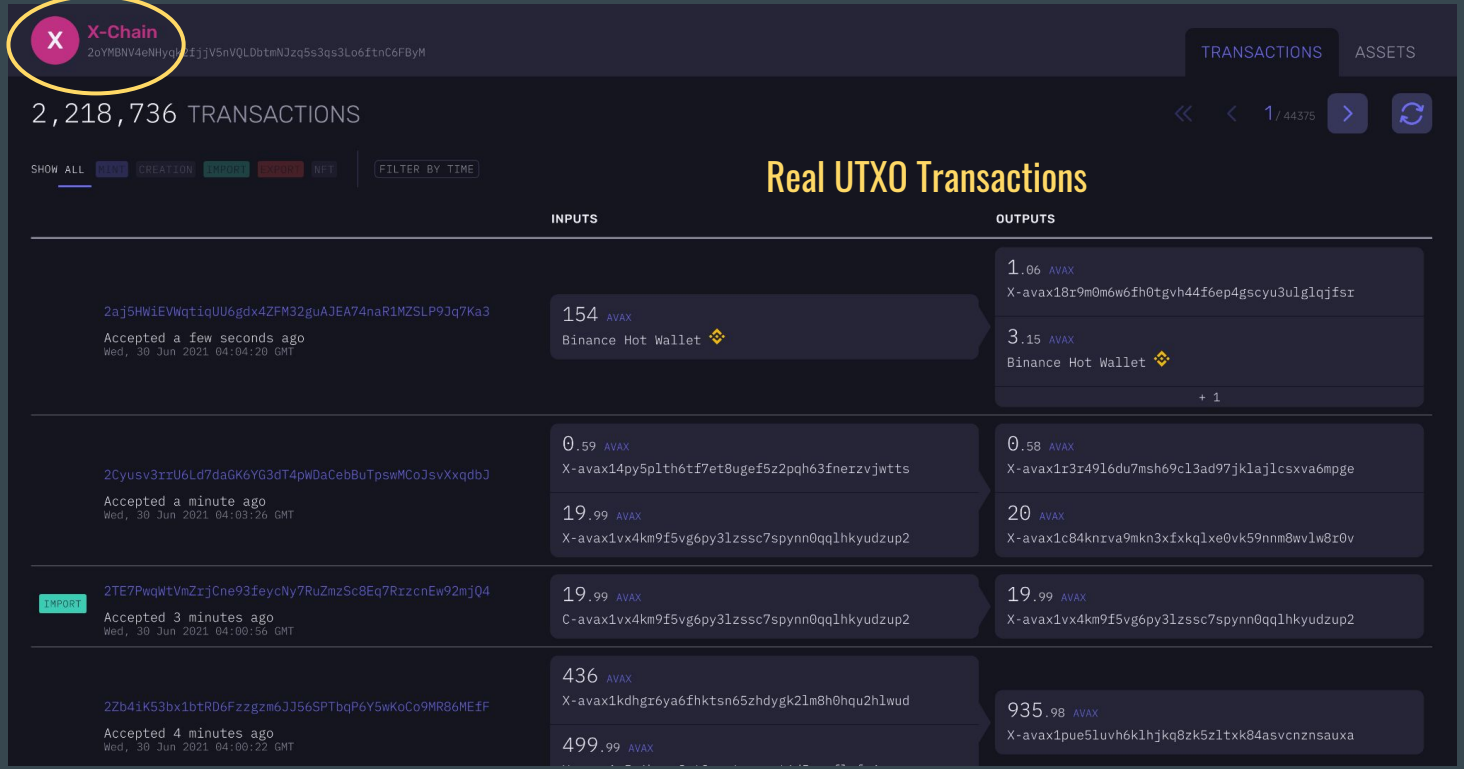

Ava Labs.

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| ☆ | 46 | Avalanche AVAX | $11.59 | ▲3.33% | ▲1.68% | $2,004,824,268 | $64,400,677<br>5,538,563 AVAX | 172,418,164 AVAX |

# What We Didn't Anticipate...

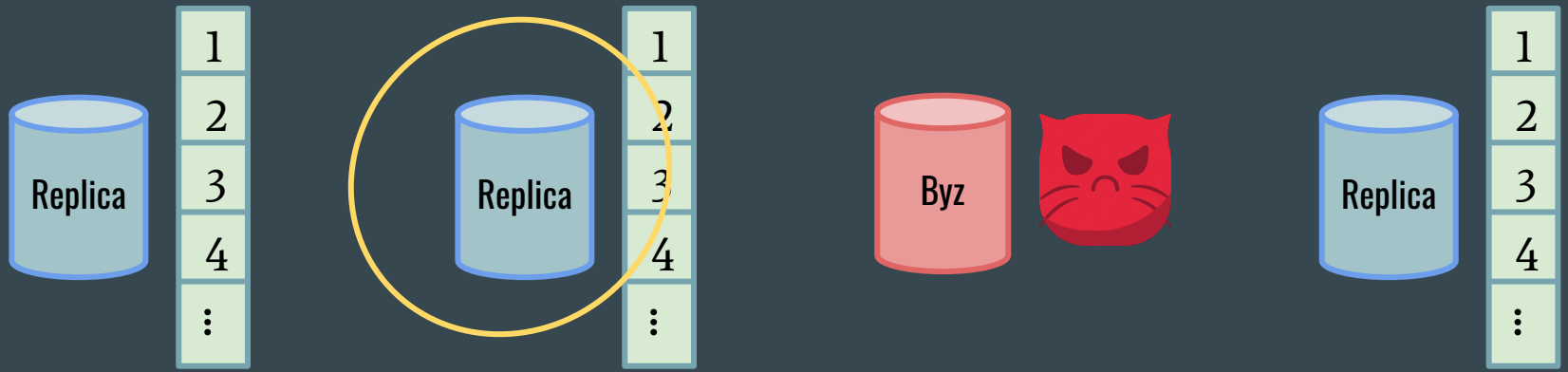


# What We Didn't Anticipate...



# Byzantine Fault Tolerant State Machine Replication

- f out of all n nodes could exhibit arbitrarily faulty behavior
- *(Replica Coordination)* other n-f correct replicas receive and process the same sequence of requests
  - Two replicated sequence $s_1 \subseteq s_2 \vee s_2 \subseteq s_1$
- Safety: agreement
- Liveness: termination



# CedrusDB

Joint work with Hongbo Zhang, Robbert van Renesse, Emin Gün Sirer.

- On-disk indexing:
  - B/B$^+$ trees (KyotoCabinet, LMDB, etc.)
    - Stable, predictable degradation
    - Non-sequential small writes
    - Tree balance maintenance amplification

# CedrusDB

Joint work with Hongbo Zhang, Robbert van Renesse, Emin Gün Sirer.

- On-disk indexing:
  - B/B$^+$ trees (KyotoCabinet, LMDB, etc.)
  - Log-Structured Merge trees (LevelDB, RocksDB, etc.)
    - Optimized for write-intensive workloads
    - Sequential writes, the storage is compact
    - Periodic, merge sorted compaction
    - Higher read amplification

# CedrusDB

Joint work with Hongbo Zhang, Robbert van Renesse, Emin Gün Sirer.

- On-disk indexing:
  - B/B$^+$ trees (KyotoCabinet, LMDB, etc.)
  - Log-Structured Merge trees (LevelDB, RocksDB, etc.)

Intended for the scenario where

1. data cannot (largely) fit in memory
2. the secondary storage is much slower.

# CedrusDB

Joint work with Hongbo Zhang, Robbert van Renesse, Emin Gün Sirer.

- On-disk indexing
- Log/Slab-based: (Masstree, KVell, FASTER)
  - Fast in-memory index
  - Records all writes to a log
  - Performance very well in the failure-free case
  - Recovery time can be expensive

# CedrusDB

Joint work with Hongbo Zhang, Robbert van Renesse, Emin Gün Sirer.

- On-disk indexing
- Log/Slab-based: (Masstree, KVell, FASTER)
- Many applications do not require the “sorted” property:
  - User configurations of apps
  - Cloud infrastructure settings/states
  - Blockchain systems!

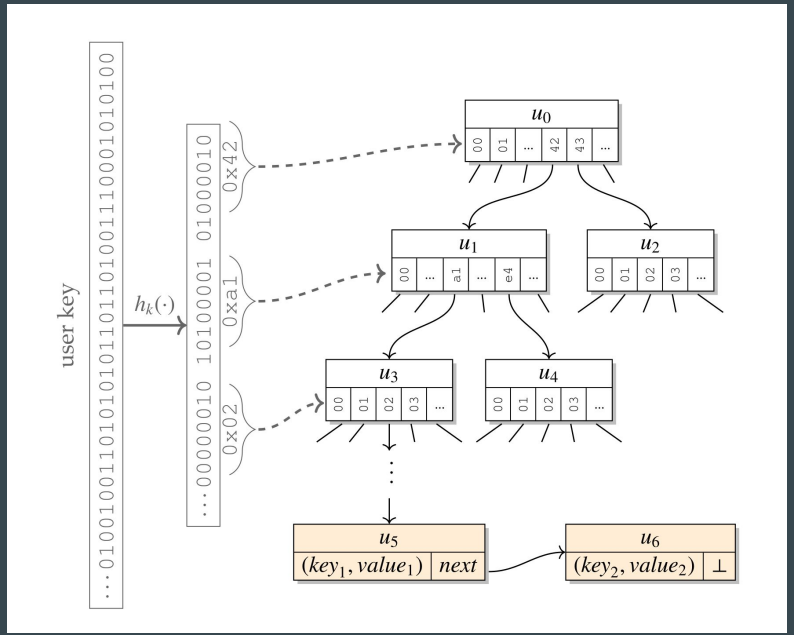
# CedrusDB

Joint work with Hongbo Zhang, Robbert van Renesse, Emin Gün Sirer.

- On-disk indexing
- Log/Slab-based: (Masstree, KVell, FASTER)
- Many applications do not require the “sorted” property
- Based on Lazy-Trie, a data structure that is:
  - An in-memory index
  - Also Storage-friendly
- Achieving both:
  - Comparable performance to FASTER
  - Fast recovery like RocksDB

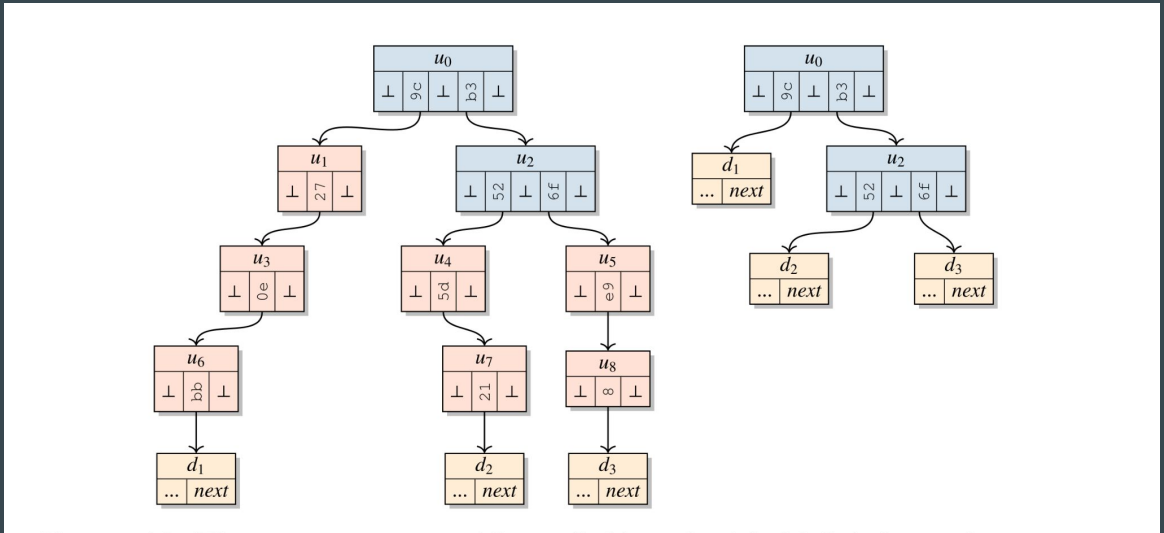
# Lazy-Trie: Started with a Simple Idea

- Hash a key into a fix-length character sequence
- Use a radix-tree to organize all the keys
- Attach the value as leaves
- Properties:
  - Logarithmic tree-height
  - Almost no collision if the hash function is strong



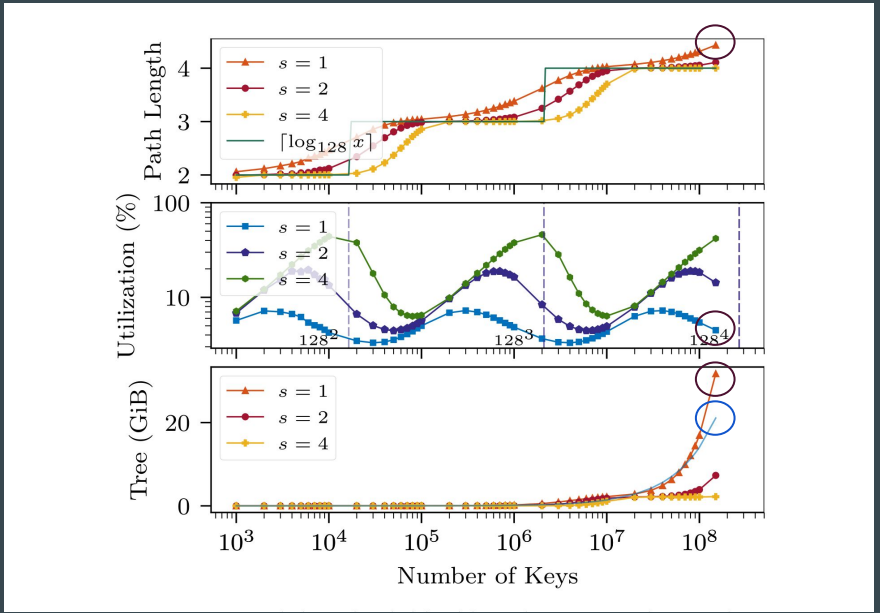
# Lazy-Trie: Making It Storage-Friendly

- Path-compression (suffix) is very effective in reducing the average tree height:
  - Unlikely to share a long prefix for two keys
  - Long “chain-like” structure in the hash-trie



# Lazy-Trie: Sluggish Splitting

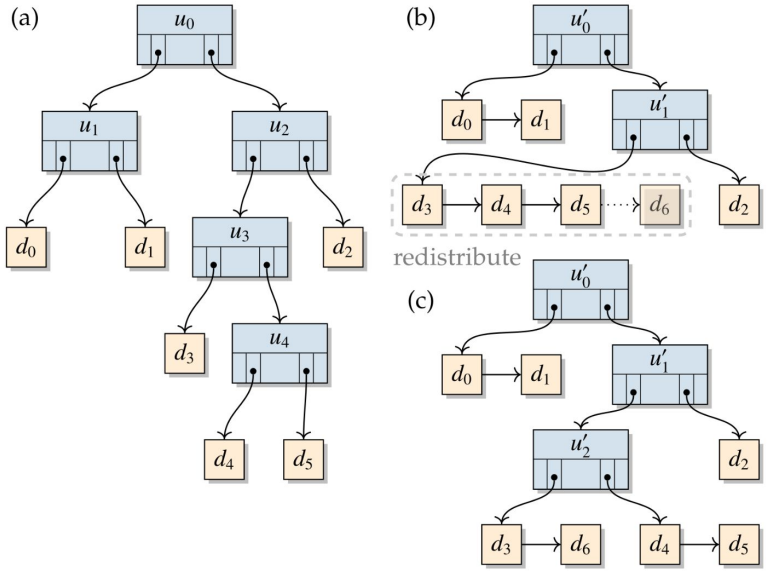
- Path lengths for different keys vary
- Small average path length
- Still a large tree height (maximum length)
- Low utilization of tree nodes close to leaves
- Bizarrely high tree footprint

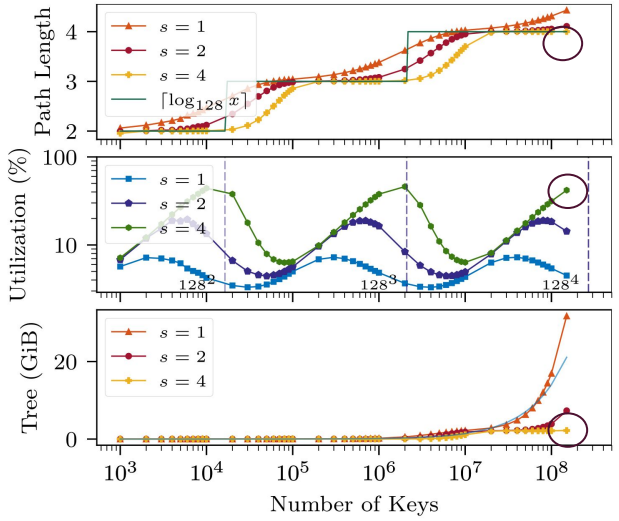



128-ary lazy-trie. The solid blue line in the bottom graph shows the user data footprint for 23-byte keys and 128-byte values.

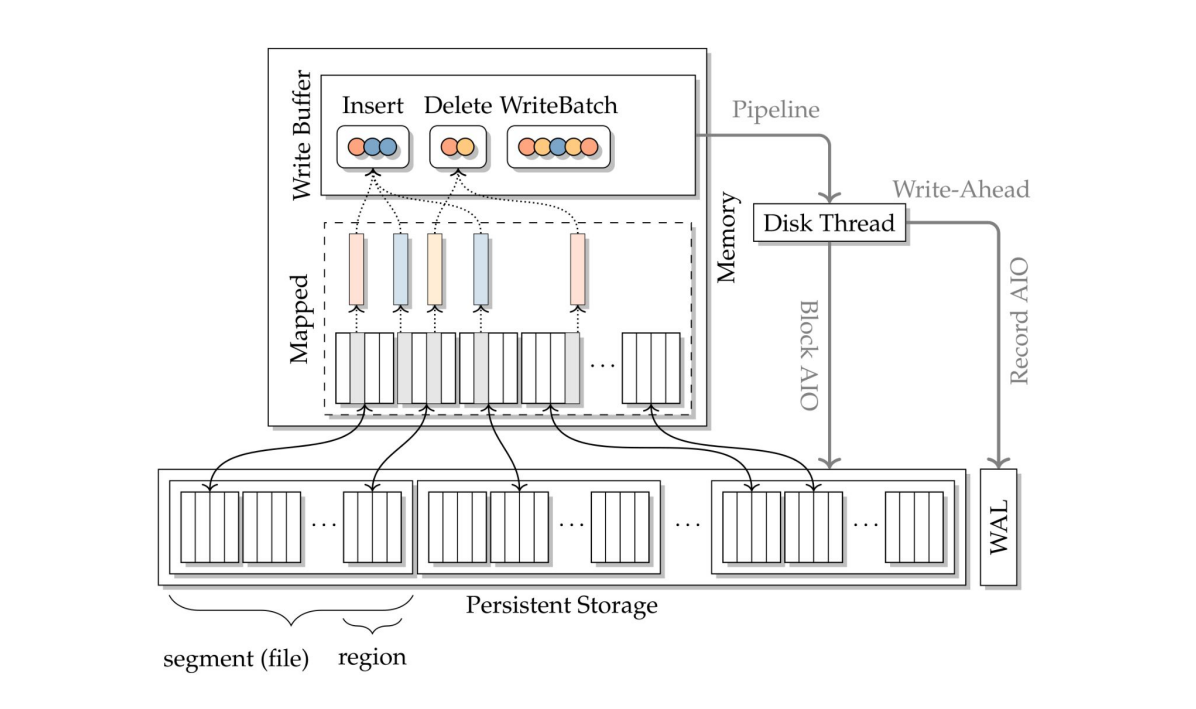
# Lazy-Trie: Sluggish Splitting

Solution: split “lazily” to allow some “sluggishness” under a node



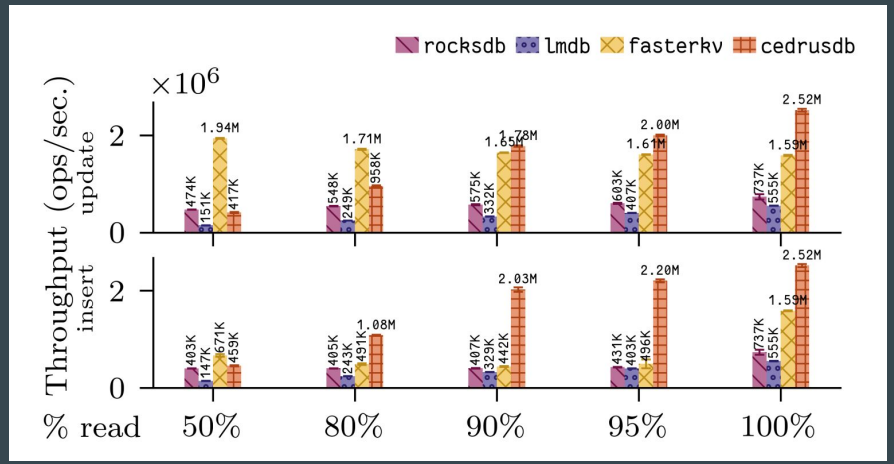



# CedrusDB: a Memory-Mapped Store

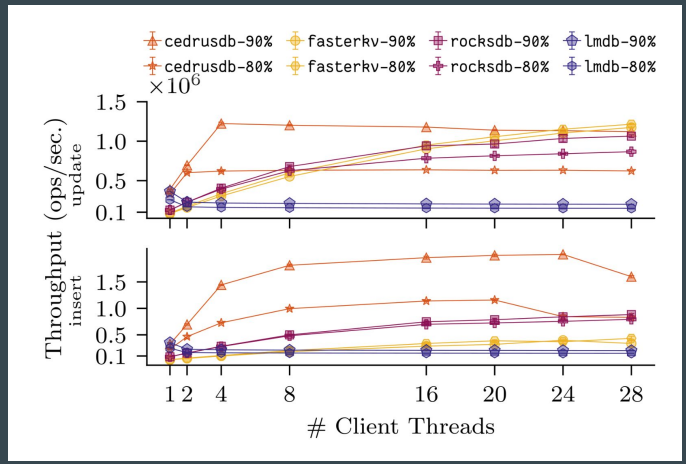



Storage hierarchy with mapped memory and write buffer. The smallest rectangle represents a page.

# CedrusDB: Performance




YCSB evaluation with $10^8$ 128-byte values and operations (4 threads).




Concurrent YCSB evaluation with $10^8$ 128-byte values and operations (AWS NVMe SSD).

# CedrusDB: Performance

| | Persistence | Indexing | Recovery (sec.) | Checkpoint (sec.) | Throughput (Kops/sec.) |
|---|---|---|---|---|---|
| CedrusDB | always, disk-indexed | hashed | 0.645 | - | 391 |
| FASTER | manual, log-based | hashed | 28.16 | 19.38 | 446 (disk)<br>1125 (volatile) |
| LMDB | always, disk-indexed | sorted | 0.003 | - | 198 |
| RocksDB | always, disk-indexed | sorted | 0.267 | - | 186 |
| Masstree | always, log-based | sorted | 31.8 per $10^8$ ops | $\approx 33.5^*$ | 1123 |
| KVell | always, slab-based | sorted | 91.9 | - | 410 |

YCSB (50% updates), Zipfian, 128-byte values and $10^8$ items. * Extrapolated value (see dissertation)

# CedrusDB: Main Contributions

- Lazy-Trie data structure
  - Friendly to both in-memory and on-disk access
  - Dynamically grows with near-optimal tree height
  - Efficient concurrent access
- Design and the implementation of CedrusDB
  - Completeness and conciseness: ~8K pure Rust code, a ready-to-use Rust crate (library)
  - WAL library
  - C/Go bindings
  - Will be open-sourced soon!

# Final Remarks and Future Career

- “Long Live the Consensus!”

- The next battle ground: State Machines
  - Verifiable/Authenticated Storage
    - Data structures (Crypto + Systems)
    - End-to-end solutions (e.g., mLSM, “CedrusDB+”)
  - Execution
    - Parallelism
    - Speculative execution
    - Just-In-Time (JIT) technologies (e.g., WASM)

# Final Remarks and Future Career

- My PhD journey started with uncertainty, went on with surprises and finally finished up with joy and self-fulfillment.
- I see myself both as a researcher and an engineer, but more of a bridging role that gules theory and practice together. Location or position does not affect my continuing effort in systems research.
- I'm looking for collaboration, and when I get older and an opportunity arises, a faculty job.
- There are too few blockchain system researchers (most are in crypto/security/theory) and the importance of building decentralized infrastructures is undervalued by the academia!

# Special Thanks


- Professor Emin Gün Sirer (my “captain”)
- Professor Robbert van Renesse (my “captain”)
- Dr. Dahlia Malkhi (best external collaborator)
- Professor Lorenzo Alvisi (inspired me to this PhD journey of computer systems)
- Professor Adrian Sampson (advisor of my ECE minor)
- Professor Robert D. Kleinberg (early Avalanche discussions and DGS)
- Research collaborators: Ittai Abraham, Kartik Nayak, Michael K. Reiter, Ling Ren, Kevin Sekniqi, and Hongbo Zhang (*alphabetical)
- My friends and family who supported me in this journey.

(The list is not comprehensive due to the slide limit)