UCDAVIS

RapidChain:
Scaling Blockchain via Full Sharding

Mahdi Zamani, Mahnush Movahedi, Mariana Raykova

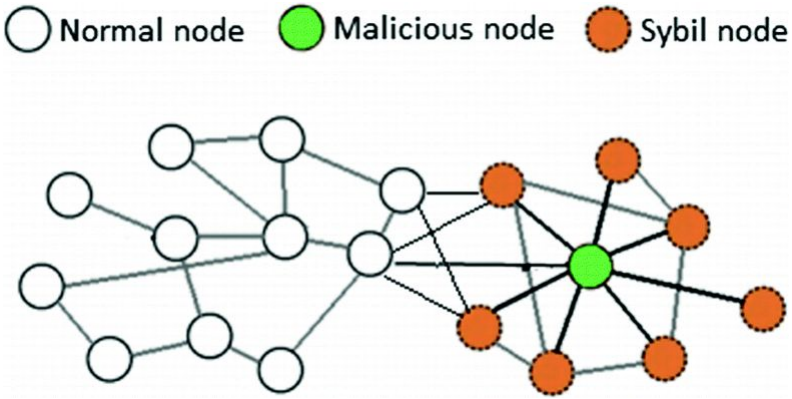
Presenters: Arindaam Roy & Divjeet Singh Jas

Roadmap

- Limitations of Traditional Methods
- Sharding based Consensus
- RapidChain: Overview
- Bootstrapping
- Consensus
- Reconfiguration
- Evaluation

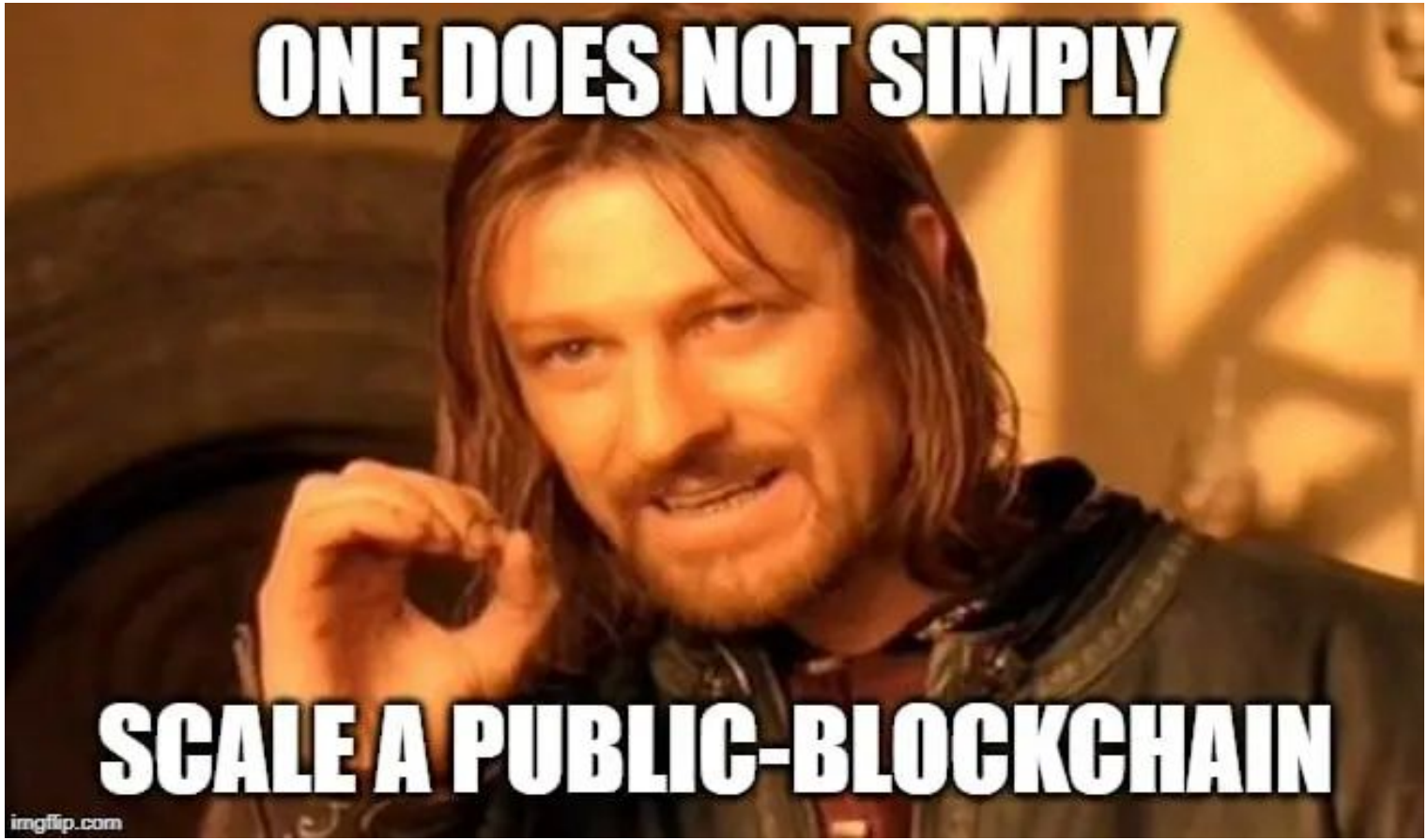
Traditional Consensus Protocols

- Cannot be used in **Open membership** setting
- Vulnerable to **Sybil Attacks** - Explained

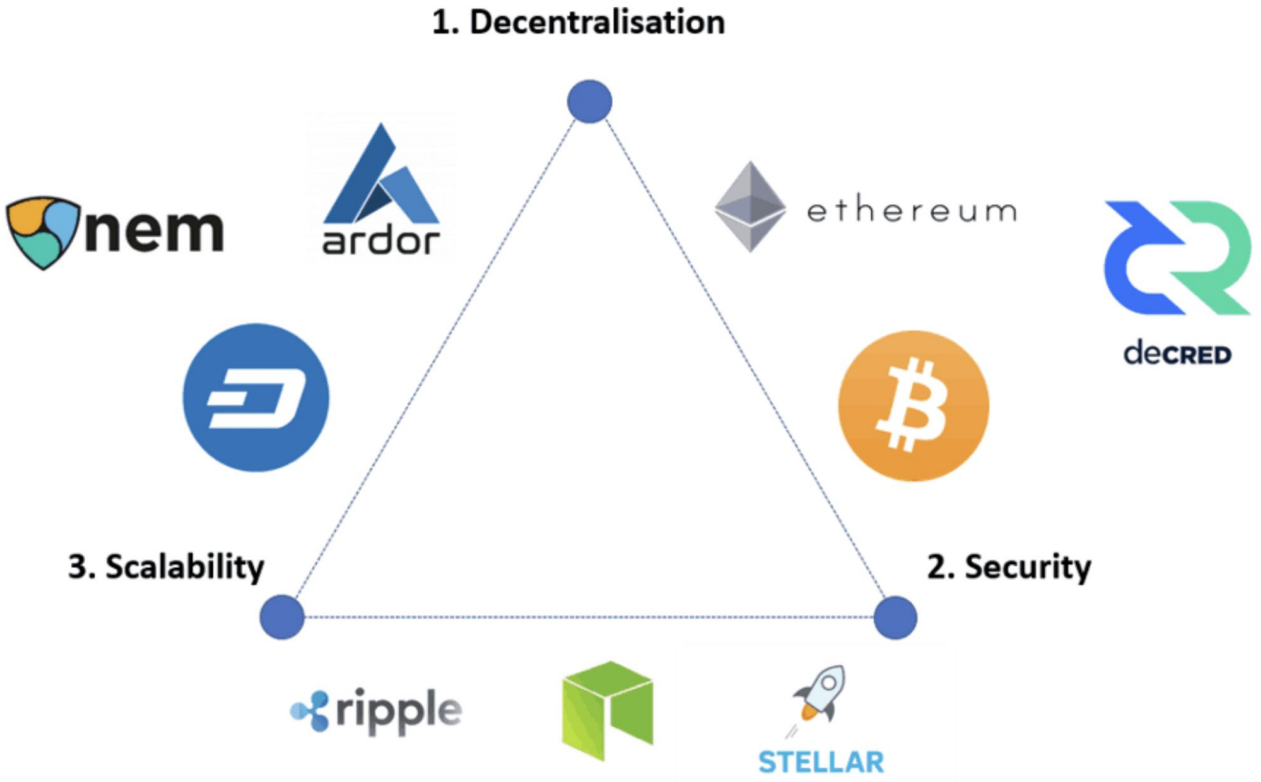


Bitcoin

- Uses **Nakamoto Consensus**
- Inhibits Sybil using **PoW**
- **Full Replication**: Low transaction throughput, high latency, poor scaling
- **Sacrifices** scalability for decentralization

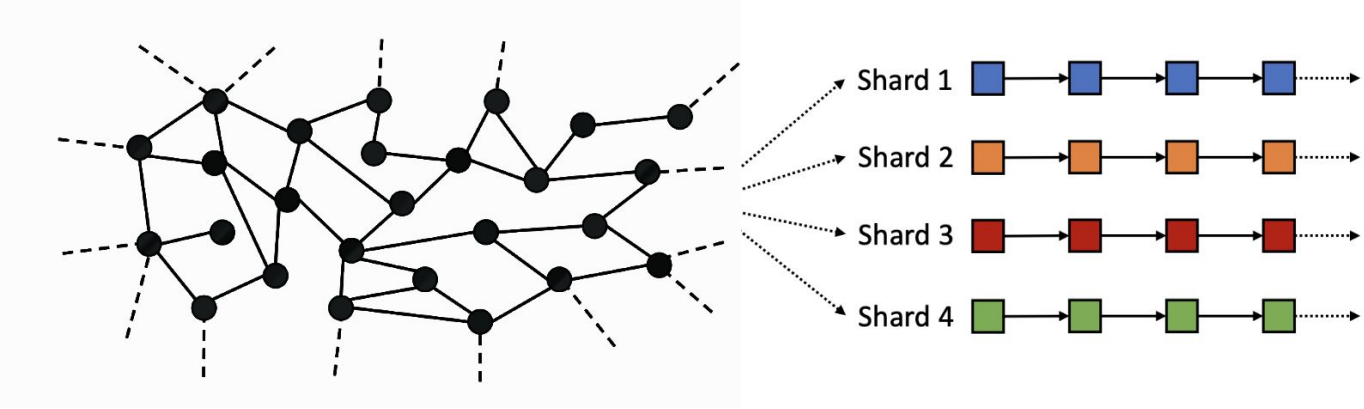

ONE DOES NOT SIMPLY
SCALE A PUBLIC-BLOCKCHAIN
imgflip.com

The Blockchain Trilemma



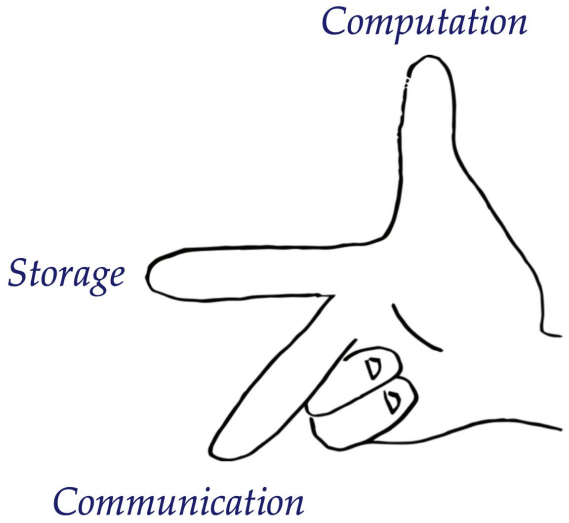
Sharding-Based Consensus

- Electing random small committees
  - Disjoint Blocks Transactions
  - Disjoint Ledgers
- Each committee runs consensus on behalf of all
- And, maintains a disjoint ledger
- Throughput increases linearly with # nodes



Full Sharding

- Computation
- Storage
- Communication



Sharding Challenges

- **Reconfiguration** *to avoid Sybil attack*
- **Cross-shard transactions**
  - Verify transactions located in other committees
- **Decentralized bootstrapping**
  - Creating Initial Random Committees
  - Establish PKI without initial randomness

RapidChain

RapidChain

- Sharding-based public blockchain protocol
- Byzantine faults from up to **1/3** of its participants
- No trusted setup required
- Achieve more than **7,300** tx/sec
- Expected confirmation latency of roughly **8.7** seconds in a network of **4,000** nodes
- A time-to-failure of more than **4,500** years.

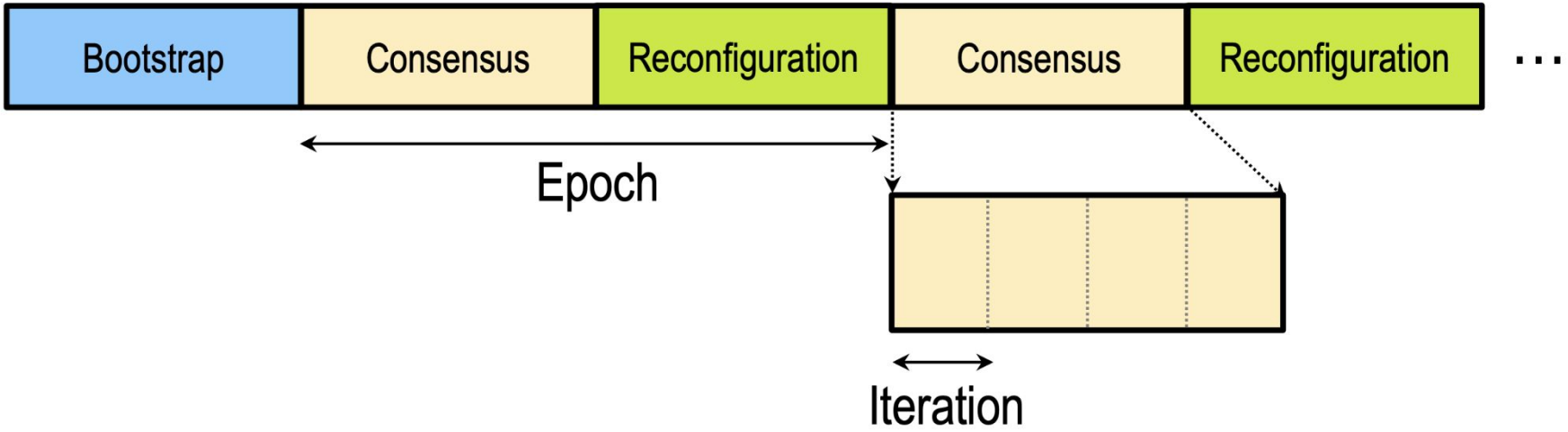
RapidChain Proposes

- A way to bootstrap the initial system
- A way of forming (and re-forming) committees, and allowing nodes to join and leave
- A way of reaching consensus within a committee (shard)
- A way of verifying transactions amongst cross-shards

Model Overview

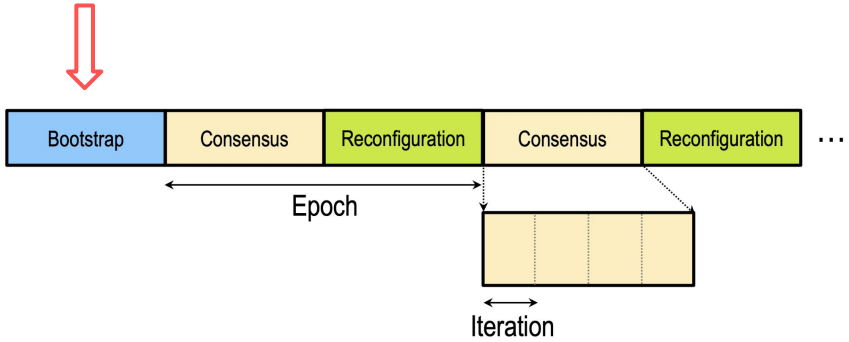
- n nodes with public (pk_i) and secret key (sk_i)
- m committees (or shards), each of size m = c*log(n), c is a security parameter
- P2P network with gossiping
- Bounded message delay Δ
- t < n/3, where t is byzantine nodes in the entire network
- Less than 5% churn in every epoch

Top Level Diagram



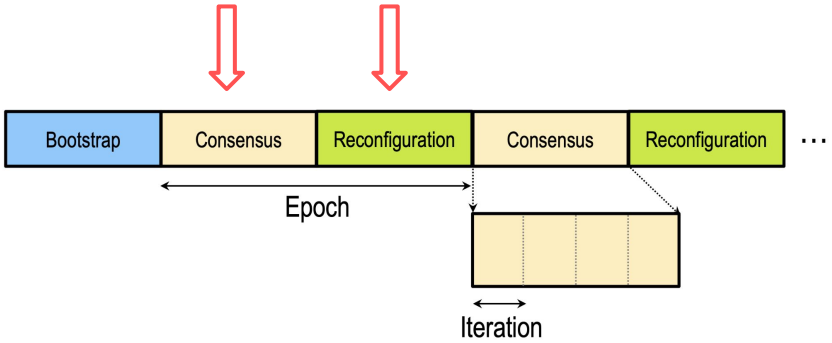
RapidChain Overview

- Proceed in *epochs*
- Bootstrapping Phase - Establishing a reference committee
  - Epoch Randomness
    - Samples sharding committees
    - Challenges for new nodes joining the system
  - Reconfiguration Block



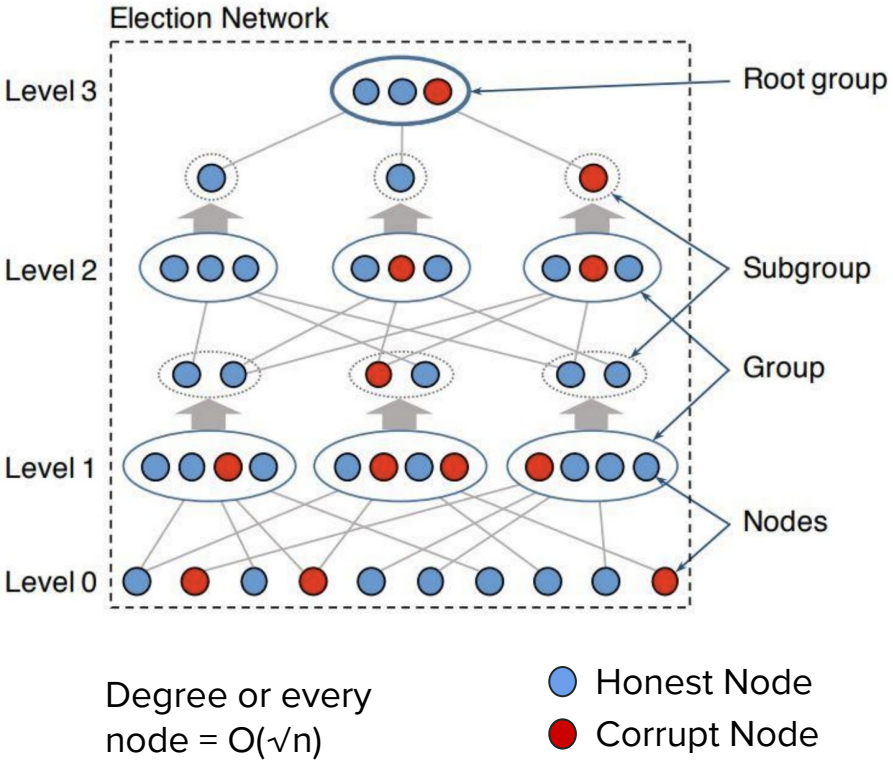
RapidChain Overview (cont'd)

- Consensus Phase
  - Each tx is sent to a random node
  - Tx sent to the output committee, C_{out}
  - Members of C_{out} verifies and adds tx to block
- Reconfiguration Phase
  - Reconfiguration block generated
    - Fresh epoch randomness
    - New list of participants
  - Cuckoo rule used to reconfigure existing committees



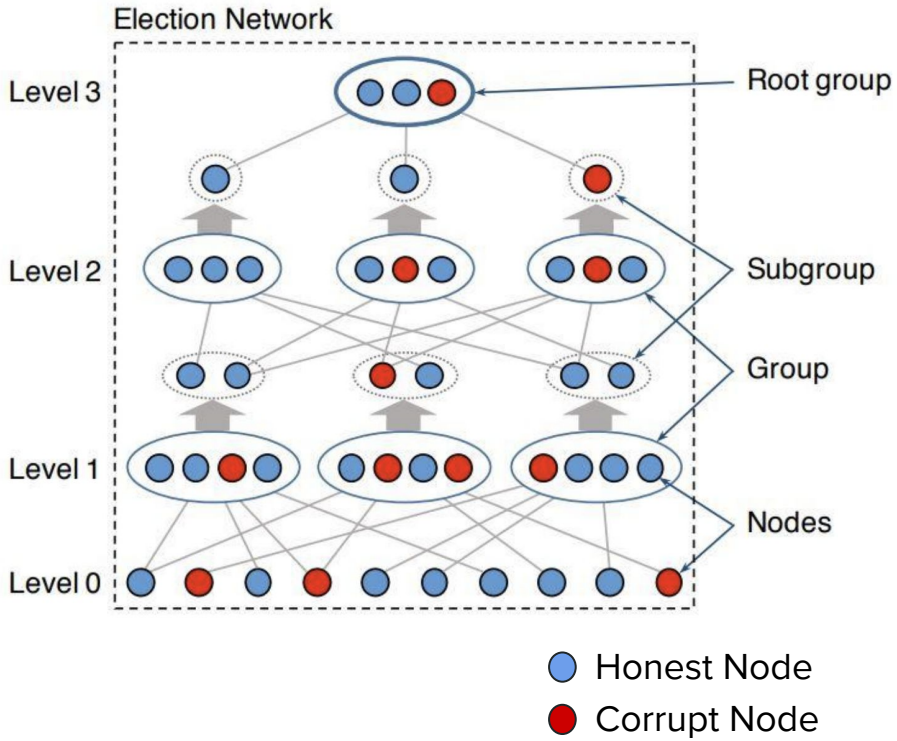
Bootstrapping

- Runs an election committee protocol
- Initial nodes have hardcoded seed (s), ID and knows initial network size
- Group formation
- Running subgroup election protocol
  - h = H(s || ID)
  - h <= 2^{256-e}
- Subgroup Peer Discovery
- Committee Formation
  - root group of size O(√n)
  - First shard or reference committee



Bootstrapping (cont'd)

- Reference Committee formation
  - Root group generating a sequence of random bits to establish a reference committee of size O(logn).
- Establish Committees
  - Reference committee are responsible to create other committees- size O(logn)

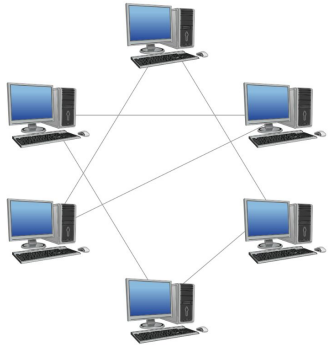


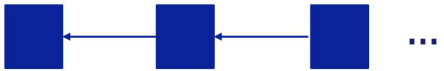
Consensus within committees

Idea: Gossip the block, then agree on the hash of the block

Consists of two parts:

- A gossiping protocol to propagate the messages (such as transactions and blocks) within a committee
- A synchronous consensus protocol to agree on the header of the block





Let’s Gossip

Visualisation : https://flopezluis.github.io/gossip-simulator/

Gossiping Large Blocks

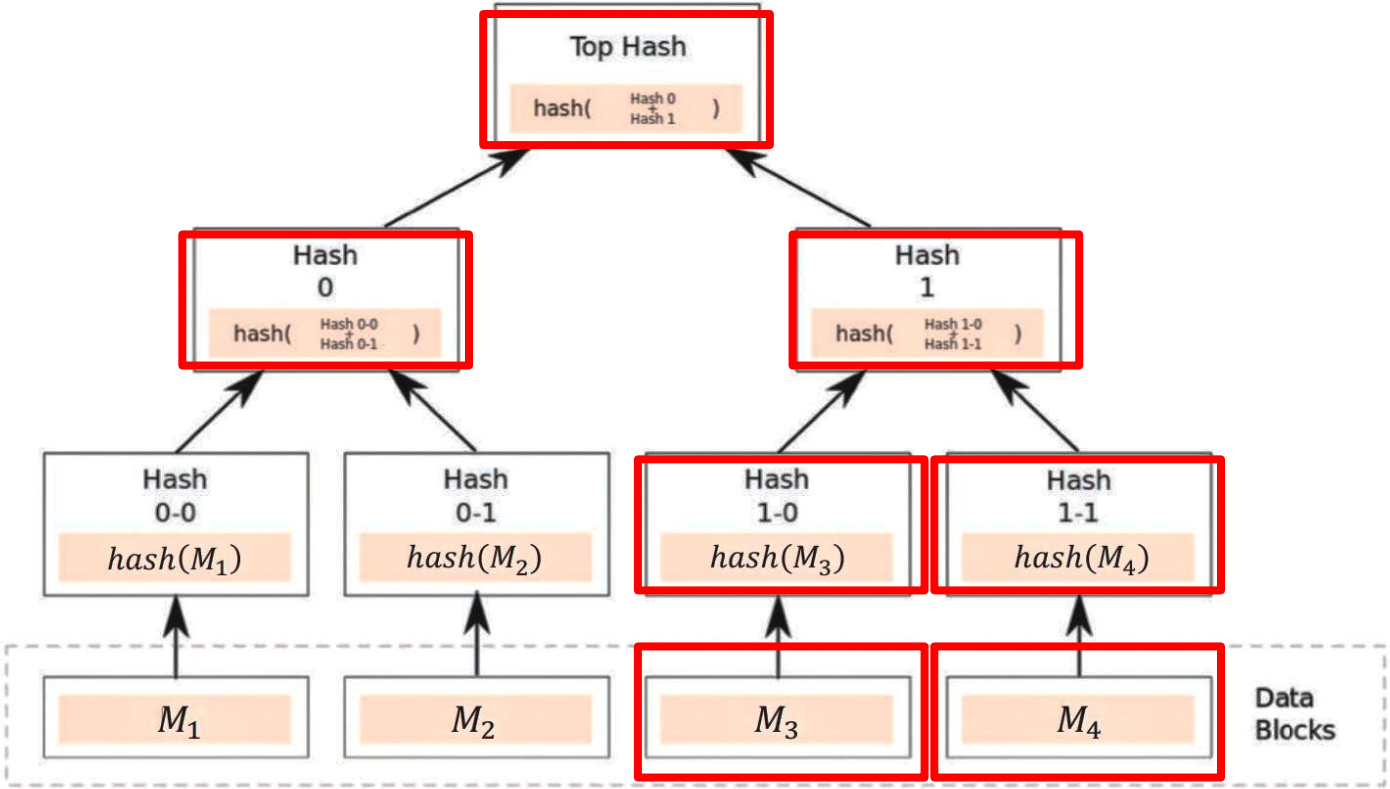
Information dispersal algorithm (IDA)

- Sender divides message M into (1−φ)κ-equal sized chunks M1,M2,...,M(1−φ)κ
- Apply ***erasure coding mechanism*** for additional φκ parity chunk to obtain M1, M2, . . . , Mκ
- Leader node computes a Merkle tree with leaves M1, . . . Mκ .
- Give each neighbor k/d chunks (gossips M_i and its Merkle proof)
- The message can be reconstructed from any set of (1 − φ)κ valid chunks

Φ - fraction of corrupt neighbors
K - no. of chunks of message M
d - no. of neighbors

Gossiping Merkle Hash

- Compute a Merkle hash tree over message chunks M1, ..., Mk
- Send Merkle proof along with message chunk to neighbors
- Each node verifies the message using the Merkle proof and the Merkle root.


Top Hash
hash(Hash 0 + Hash 1)
Hash 0
hash(Hash 0-0 + Hash 0-1)
Hash 1
hash(Hash 1-0 + Hash 1-1)
Hash 0-0
$hash(M_1)$
Hash 0-1
$hash(M_2)$
Hash 1-0
$hash(M_3)$
Hash 1-1
$hash(M_4)$
M_1
M_2
M_3
M_4
Data Blocks

Consensus Protocol

- Can tolerate $f < \frac{1}{2}$ m
- It's a synchronous consensus protocol with a constant delay Δ
- Each committee picks a leader randomly using the epoch randomness
- The leader gathers all the transactions it has received (from users or other committees) in a block B_i
- Leader gossips the block using IDA-gossip and creates the block header H_i that contains the iteration number as well as the root of the Merkle tree from IDA-Gossip.

Consensus Protocol (cont'd)

1

Leader Gossips:
H_i + *propose tag*

2

Nodes *echo* received H_i:
H_i + *echo tag*

3

If nodes see different versions of H_i

Leader is malicious and gossip:
H_i^l + gossip message with *pending tag*

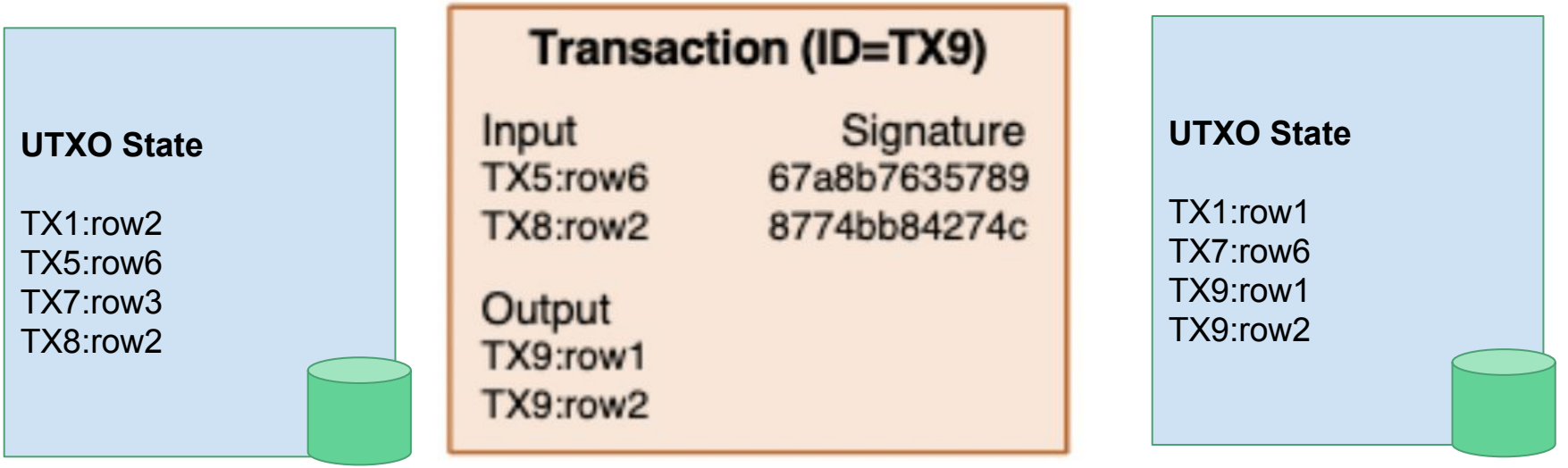
4

If nodes see mf+1 echoes and there is only 1 version of H_i

Accept the header and gossip:
H_i + *accept tag* + proof

Cross-Shard Transactions

- Tx has a unique id, list of inputs, list of outputs
- Node verifies
  - Input is unspent
  - Sum of the outputs < sum of the inputs

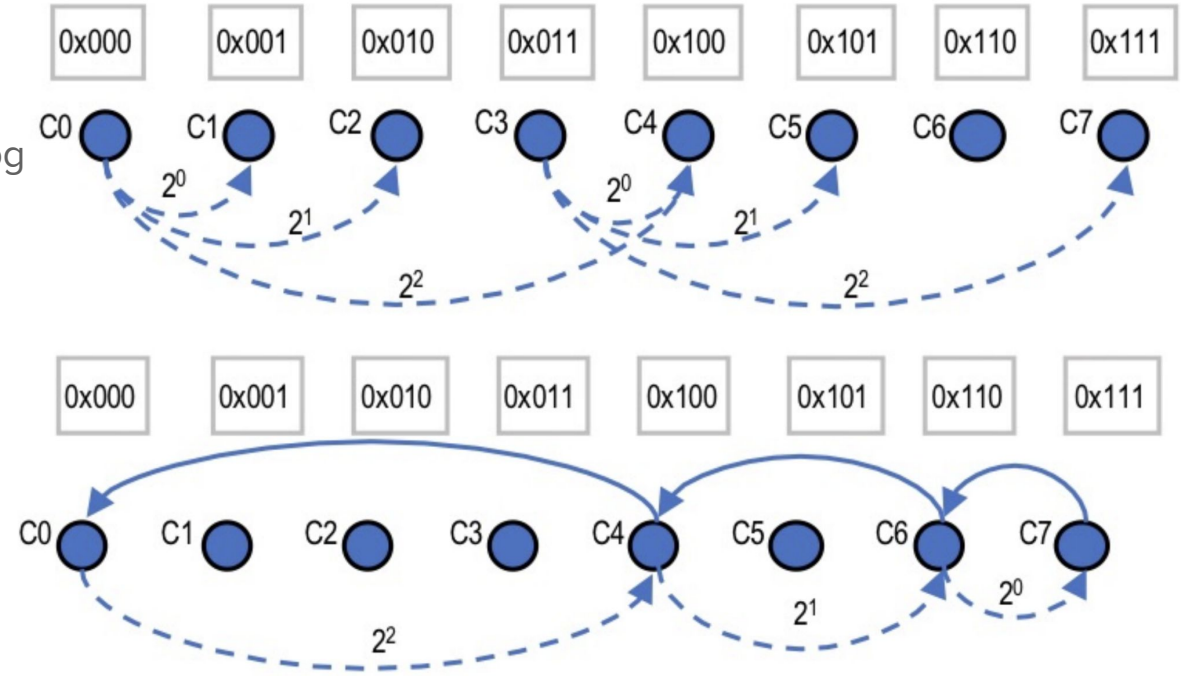


Inter-Committee Routing

Kademlia routing algorithm

Each committee maintains a routing table of log n records that point to log n different committees

Distance of $\mathbf{2^i}$ 0<= i<= logn - 1



Committee C0 wants to locate committee C7 (via C4 and C6) responsible for transactions with prefix 0x111.

Problem with Committees

- Join/Leave attacks: Corrupt nodes could strategically rejoin the network to take control of a committee.
- Malicious nodes can corrupt the good nodes

RapidChain's Defense

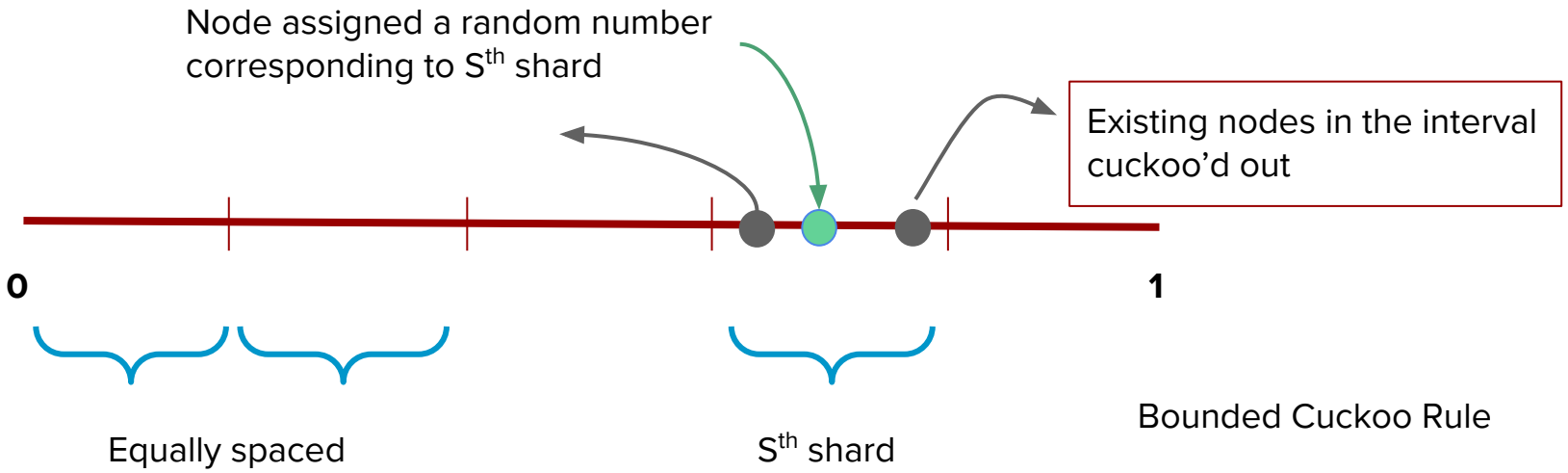
- A 'pay-to-play' scheme (used by bitcoin)
- Selective random shuffling based on the Cuckoo rule

Reconfiguration

- **Offline PoW**
  - Rely on Pow to protect against Sybil attacks
  - Reference committee is responsible to verify PoW result
  - $O = H(\text{timestamp} \,||\, PK \,||\, r_i \,||\, x)$
- **Randomness Generation**
  - Reference Committee run a Distributed random generation protocol
- **Cuckoo Rule**
  - Randomly assign new node
  - Assign a number of members in the committee to another committee

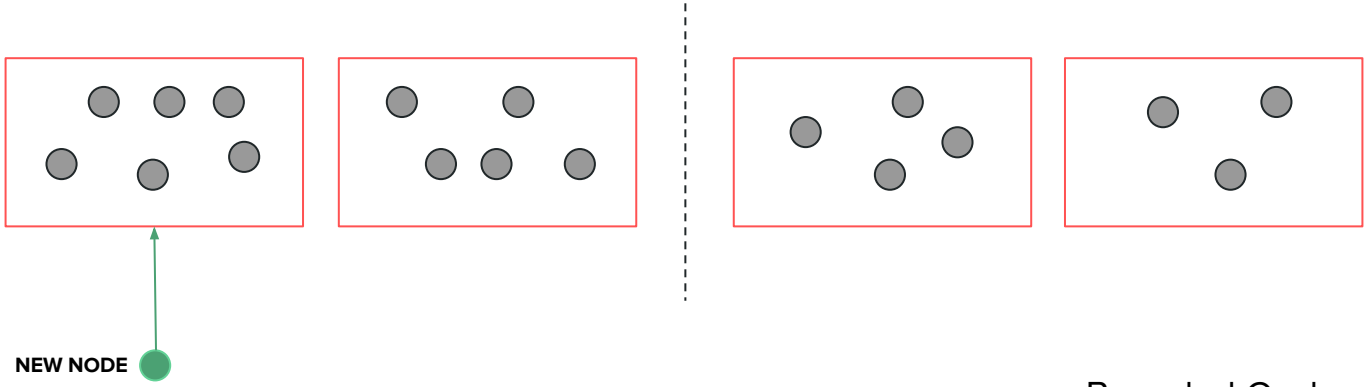
Cuckoo Rule

- New node assigned a random shard
- Evict k nodes from the shard, not including the new node
- Assign these k nodes to another committee



Cuckoo Rule

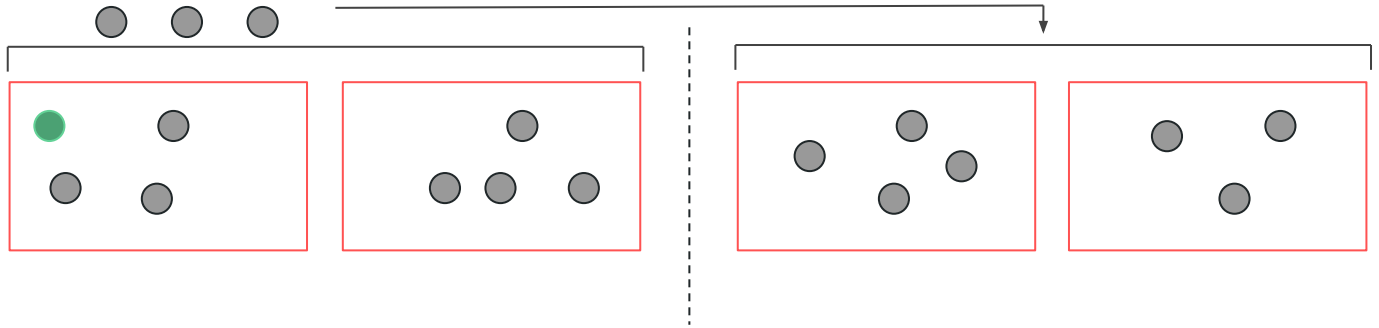
- New node assigned a random shard
- Evict k nodes from the shard, not including the new node
- Assign these k nodes to another committee



Bounded Cuckoo Rule

Cuckoo Rule

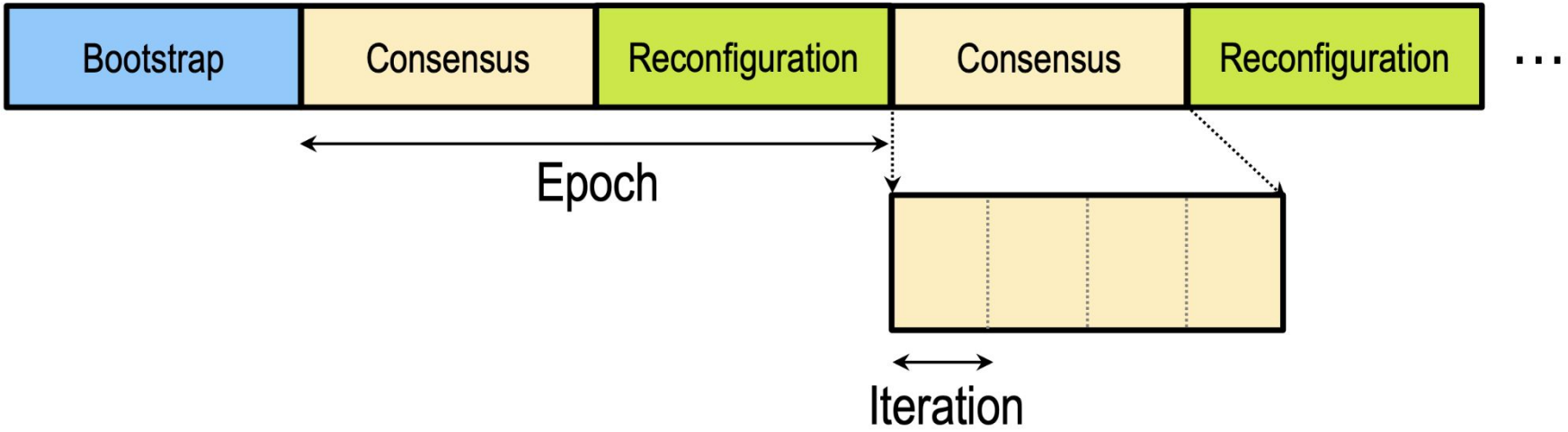
- New node assigned a random shard
- Evict k nodes from the shard, not including the new node
- Assign these k nodes to another committee



Bounded Cuckoo Rule

What we have learned?

Top Level Diagram



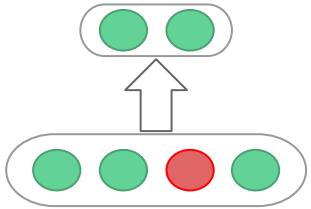
Bootstrap

Initial nodes with hard coded seed (s), ID, network knowledge

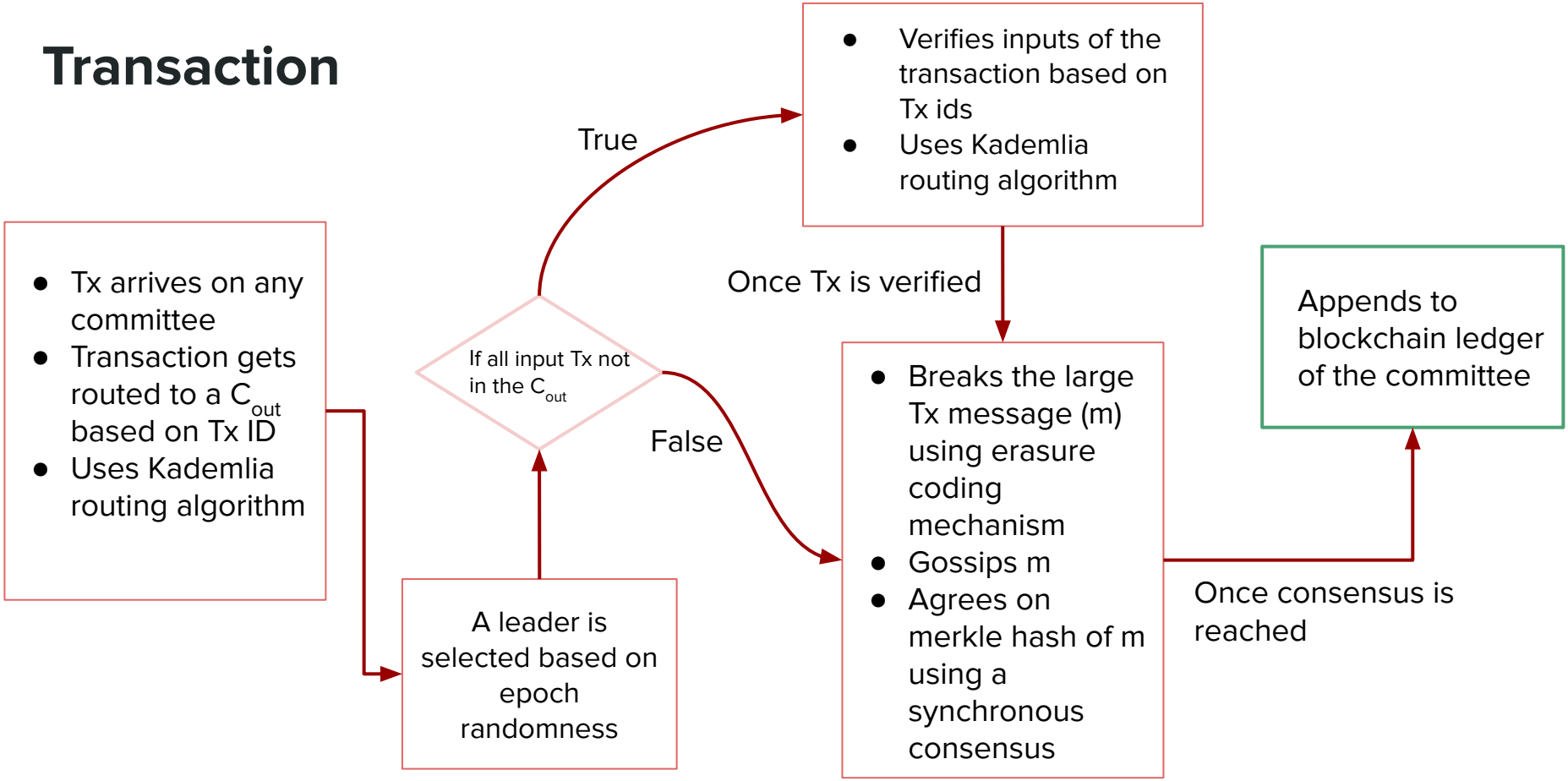
- Random sampler graph
- Election committee protocol
- Forms root group of size O(√n)

Root group elects the reference committee C_R

Reference committee creates K committee of size O(logn)



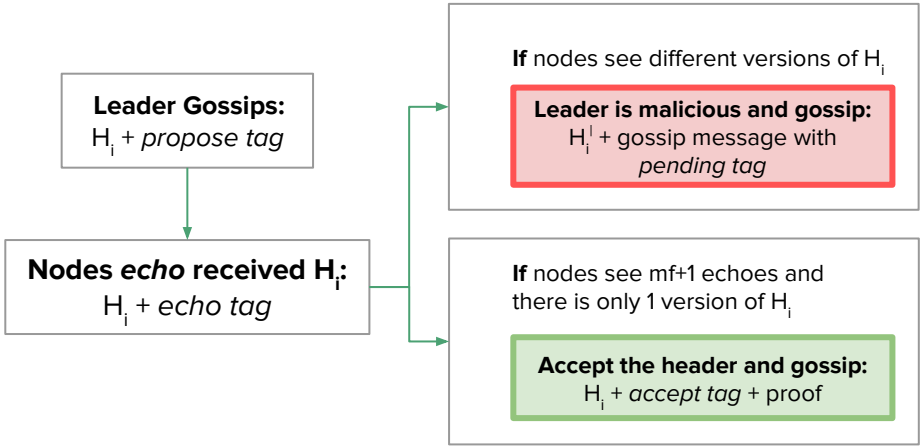
Transaction



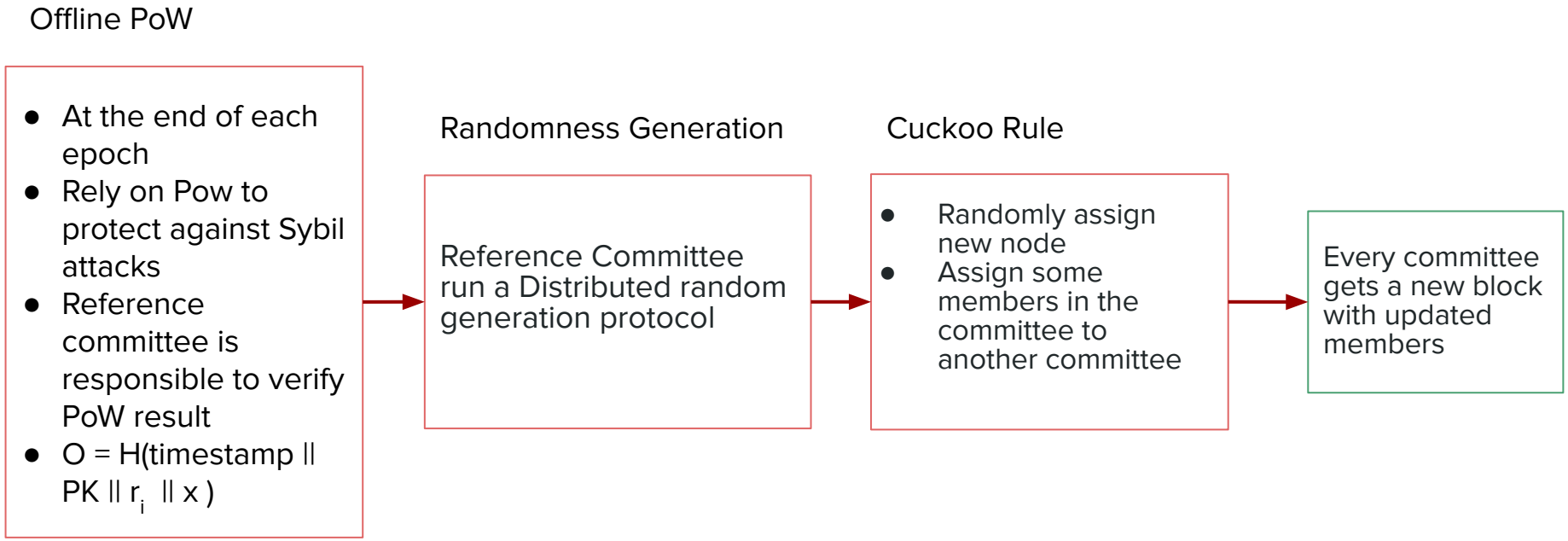
Consensus

1. A **gossiping protocol** to propagate the messages (such as transactions and blocks) within a committee

2. A **synchronous consensus protocol** to agree on the header of the block

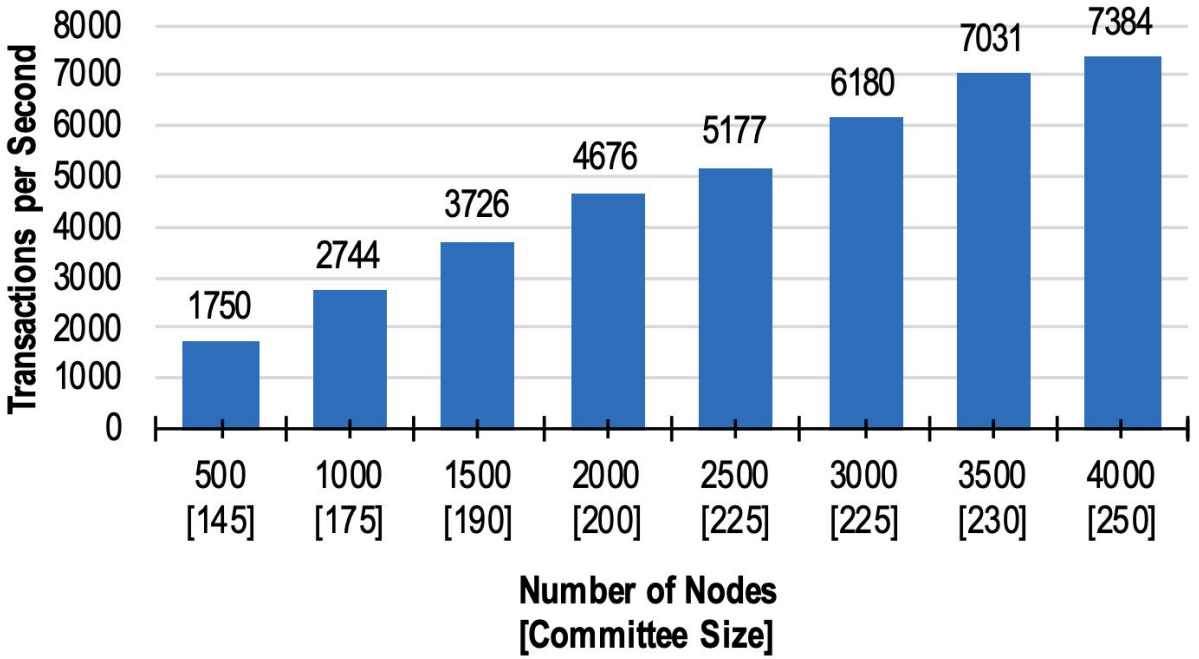


Reconfiguration

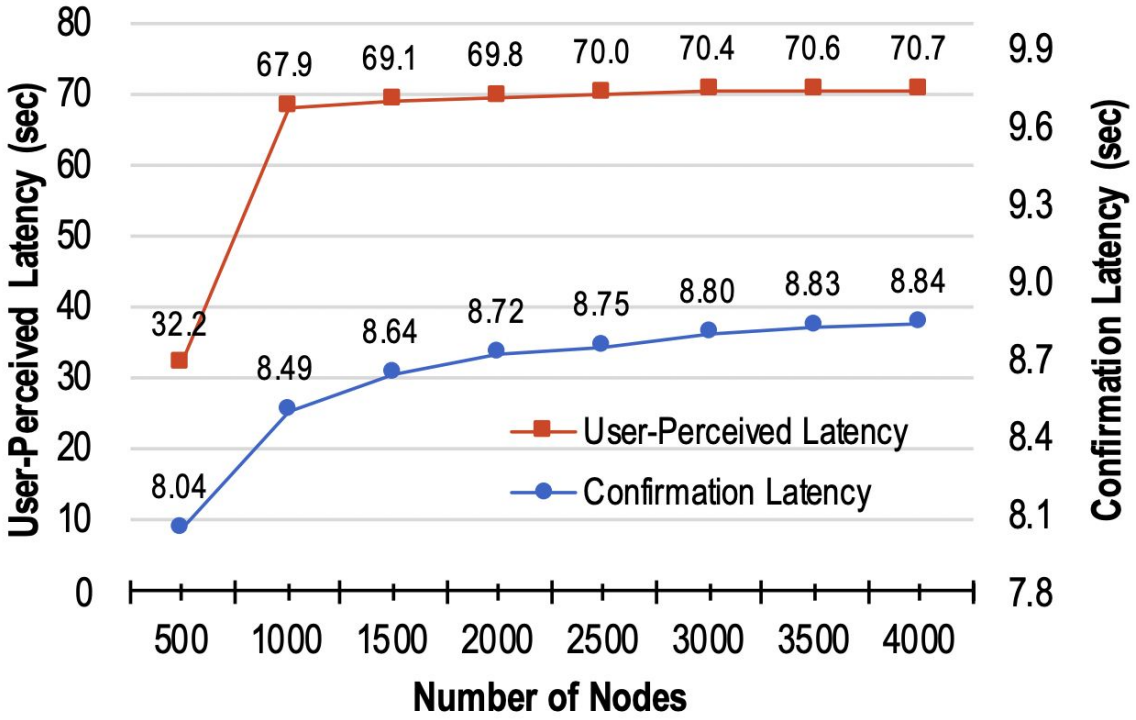


Evaluation

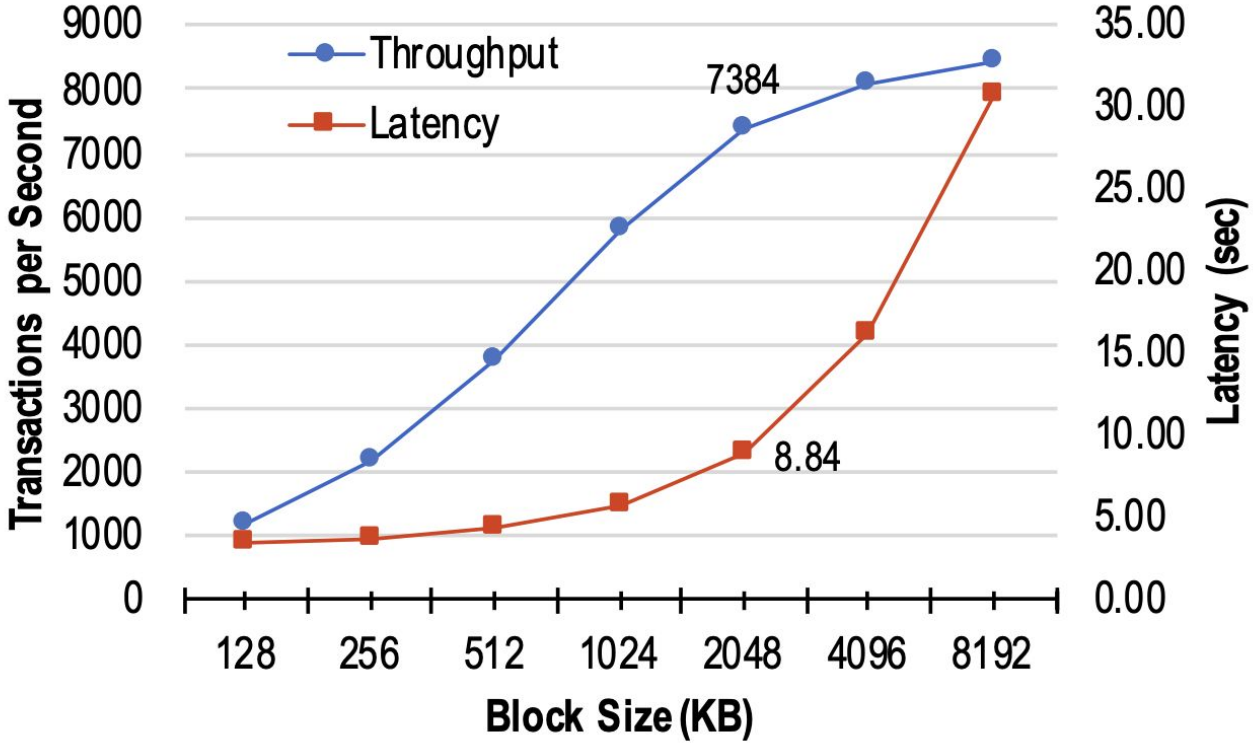
Throughput vs n



Latency vs n


User-Perceived Latency (sec)
Confirmation Latency (sec)
Number of Nodes
User-Perceived Latency
Confirmation Latency
80
70
60
50
40
30
20
10
0
9.9
9.6
9.3
9.0
8.7
8.4
8.1
7.8
500
1000
1500
2000
2500
3000
3500
4000
32.2
67.9
69.1
69.8
70.0
70.4
70.6
70.7
8.04
8.49
8.64
8.72
8.75
8.80
8.83
8.84

Impact of Block Size



Where does RapidChain stand?

Protocol	# Nodes	Resiliency	TPS	Latency	Storage	Shard Size	Time to Failure
Elastico	1,600	$n/4$	40	800 *sec*	1x	100	1 *hour*
OmniLedger	1,800	$n/4$	500	14 *sec*	1/3x	600	230 *years*
OmniLedger	1,800	$n/4$	3,500	63 *sec*	1/3x	600	230 *years*
RapidChain	1,800	$n/3$	4,220	8.5 *sec*	1/9x	200	1,950 *years*
RapidChain	4,000	$n/3$	7,380	8.7 *sec*	1/16x	250	4,580 *years*

References

1. https://eprint.iacr.org/2018/460.pdf
2. https://cbr.stanford.edu/seminarTalks/zamani.pdf
3. https://blog.acolyer.org/2018/12/07/rapidchain-scaling-blockchain-via-full-sharding

THANK YOU!