RapidChain: Scaling Blockchain via Full Sharding

Ramesh Adhikari, Graduate Research Assistant
School of Computer and Cyber Sciences, Augusta University

20th September, 2022

AUGUSTA UNIVERSITY

Outline

- Problem in Blockchain (Motivation for this paper)
- Sharding
- RapidChain
- RapidChain Protocols
- Evaluation
- Conclusion

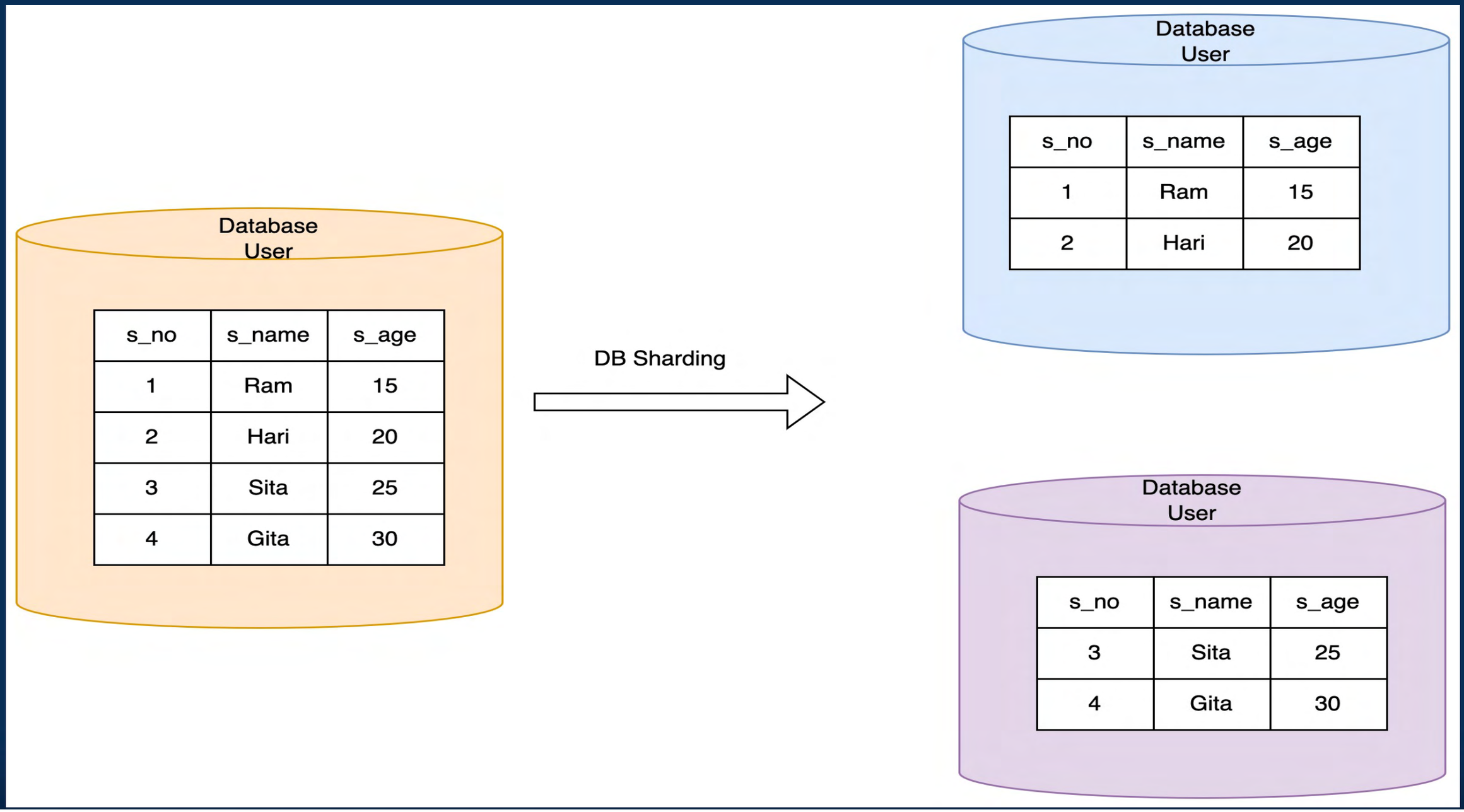
Problem with Blockchain (Motivation for this paper)

- Performance and Scalability
  - Blockchain requires all participants to agree on the validity of transactions
  - Hence every node must store all transactions
  - Scalability reduces with the rise in decentralization
  - As the number of nodes increase, the longer it takes for a transaction to be propagated and consensus to be achieved the more it degrades the overall performance

Transaction in	Transaction Per Second (TPS)	Average Transaction (Block) Confirmation Time
VISA	24,000	Instantly
ETHEREUM	15-20	6 Minute
bitcoin	3-7	60 Minute

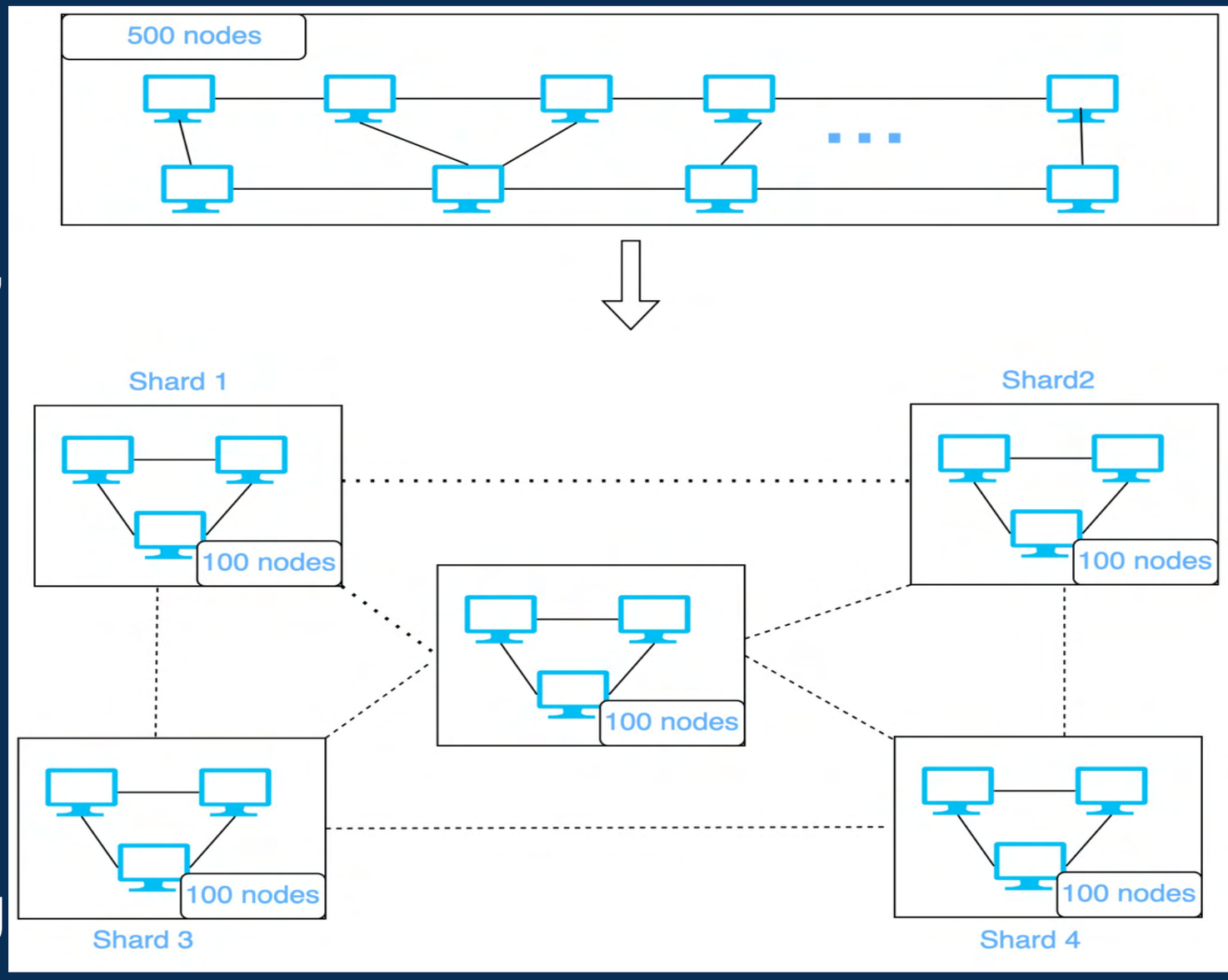
Solution: use Sharding

Concept of DB Sharding



Sharding in Blockchain

- **Sharding** is to split the overheads of processing transactions among multiple, smaller groups of nodes.
- These groups work in parallel to maximize performance
- Goal for a sharded network is to be able to process transactions with quick enough decision by reducing computing and storage redundancies

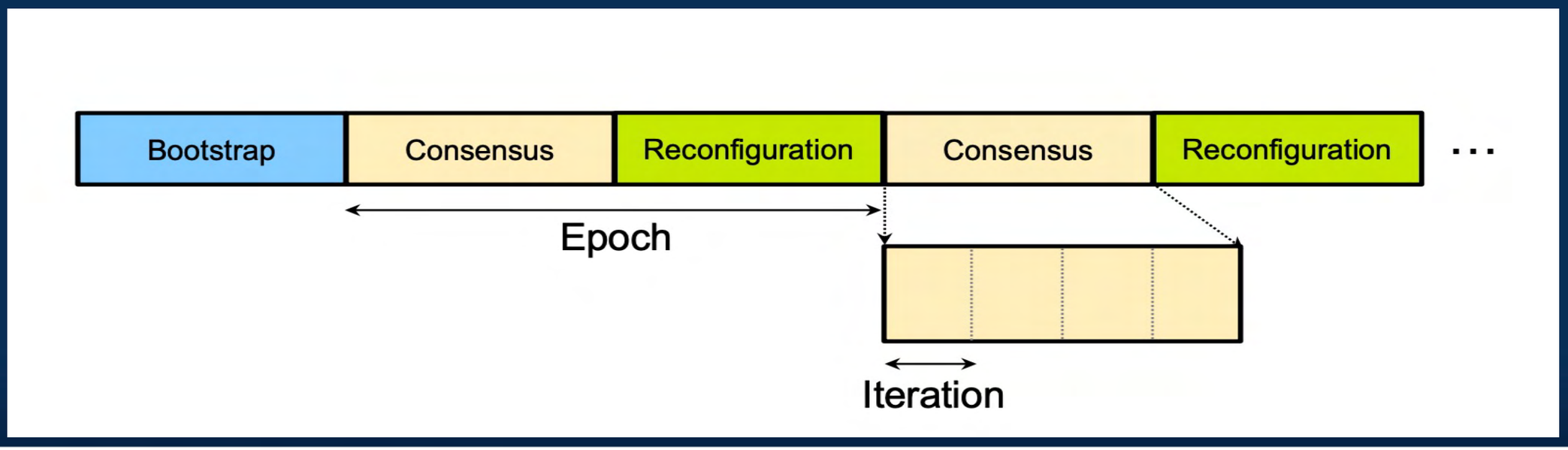


Rapidchain

- **Sharding-based public blockchain protocol**
- Partitions the set of nodes into multiple smaller groups of nodes called committees (or shards)
- Committees operate in parallel on disjoint blocks of transactions and maintain disjoint ledgers
- Aiming on full sharding i.e. on communication, computation and storage

Rapidchain Protocol

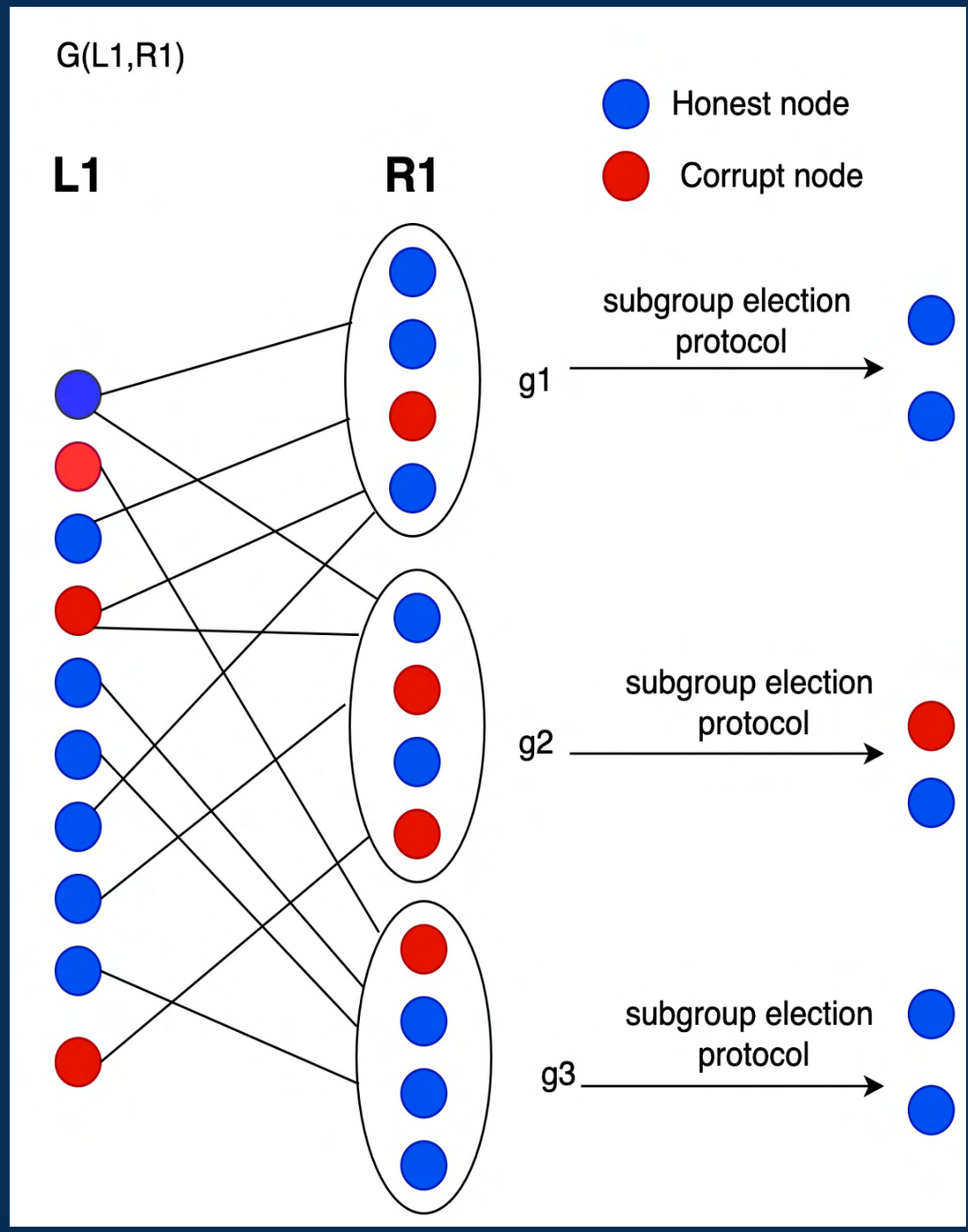
- Bootstrapping: Create Root Group and Establishing a reference committee
- Consensus: Gossip the block, then agree on the hash of the block
- Reconfiguration: allows new nodes to establish identities and join the existing committees



Bootstrapping

- To create Root Group, election network is constructed bipartite graph
- Initially n nodes are in L_1 and every node is assigned to set of groups R_1
- Each group runs a subgroup election protocol to select a random subset of its members.
- Elected members will then serve as nodes in L_2
- To construct the election network, we set $|L_i| = |L_{i-1}|^{\alpha_i + \beta_i \gamma_i}$ and $|R_i| = |L_i|^{\alpha_i}$

where $|L_1| = n$, $|R_1| = n^{\alpha_i}$, $0<\alpha_i,\beta_i,\gamma_i< 1$, and $i = \{2, ..., \ell\}$



Bootstrapping contd

- Running subgroup election protocol
  - Members of each group run distributed random generator(DRG) to generate random string s
  - Each node with identification ID compute h=H(s||ID) and announces itself elected if $h<=2^{256-e}$, where H is hash function, in practice set e=2.
  - All nodes sign the (ID,s) of e elected nodes who have the smallest h and gossip their signatures in group as a proof of election for the elected node.
  - This process continues to last sampler graphs

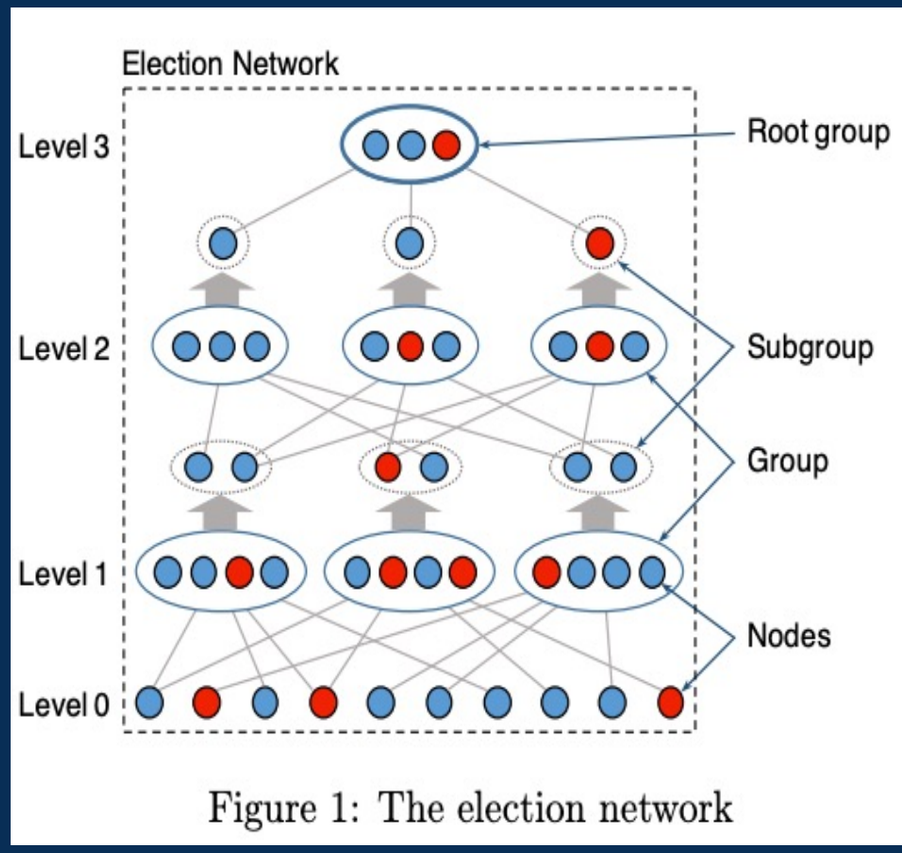


Figure 1: The election network

Bootstrapping Contd..

- Reference Committee formation
  - Root group generates and distributes a sequence of random bits that are in turn used to establish a *reference committee* of size O (log n)
- Establish Committees
  - Reference committee then creates K committees (Shard)
  - The bootstrap phase runs only once at the start of RapidChain

Rapidchain contd…

- Let n denote the number of participants in the protocol at any given time
- $m \ll n$ denote the size of each committee
- It creates $k = n/m$ committees each of size $m = c \log n$ nodes, E.g. with 1,000 nodes we'll have around 17 committees of 60 nodes each
- where c is a constant depending only on the security parameter (in practice, c is roughly 20)

Consensus within committees

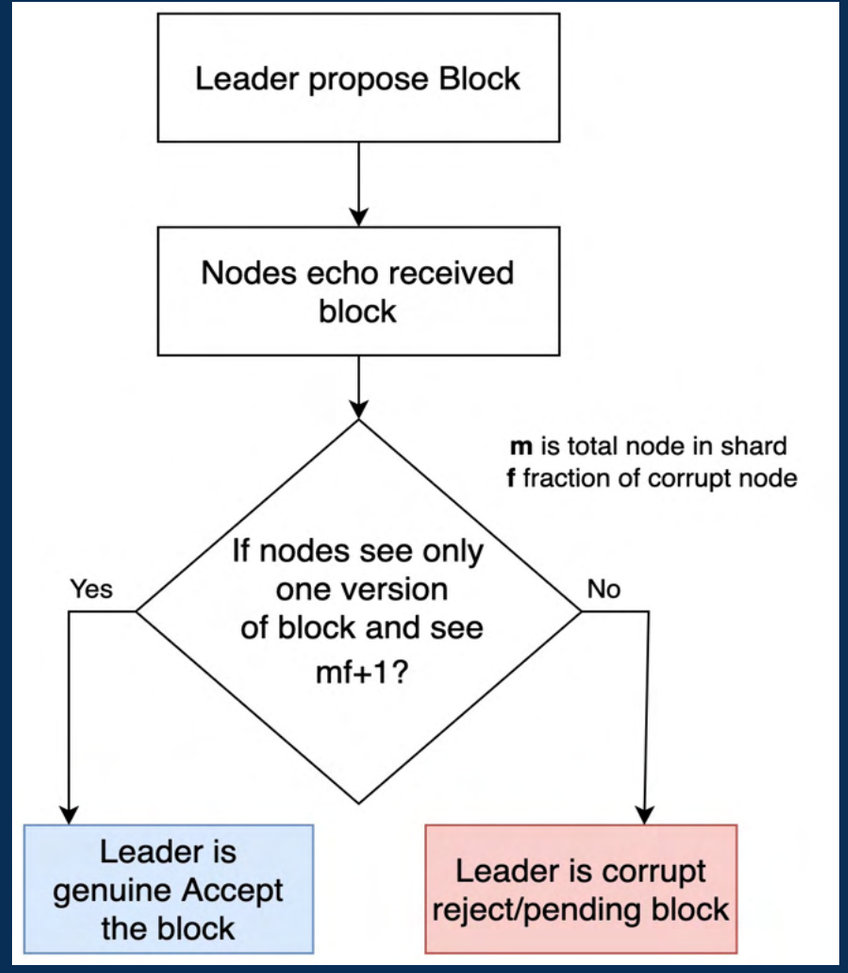
- Idea: Gossip the block, then agree on the hash of the block
- Consists of two parts:
  - A gossiping protocol to propagate the messages (such as transactions and blocks) within a committee
  - A synchronous consensus protocol to agree on the header of the block

Gossiping Large Blocks

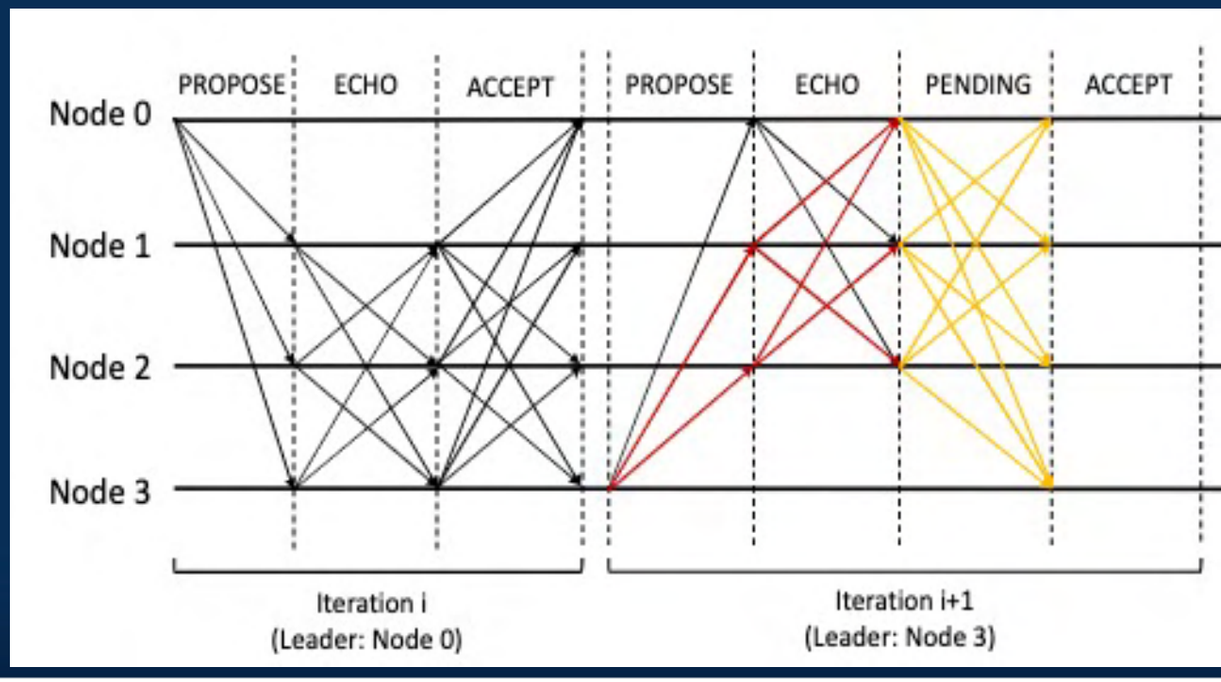
- Information dispersal algorithm (IDA)
  - Encode message (M) into k chunks M_1, M_2, . . . , M_k using an erasure coding mechanism
  - Give each neighbor k/d chunks (d is no. of neighbors)
  - The message can be reconstructed from any set of $(1 - \varphi)\kappa$ valid chunks. (φ is fraction of corrupt neighbors)

Consensus Protocol

- 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

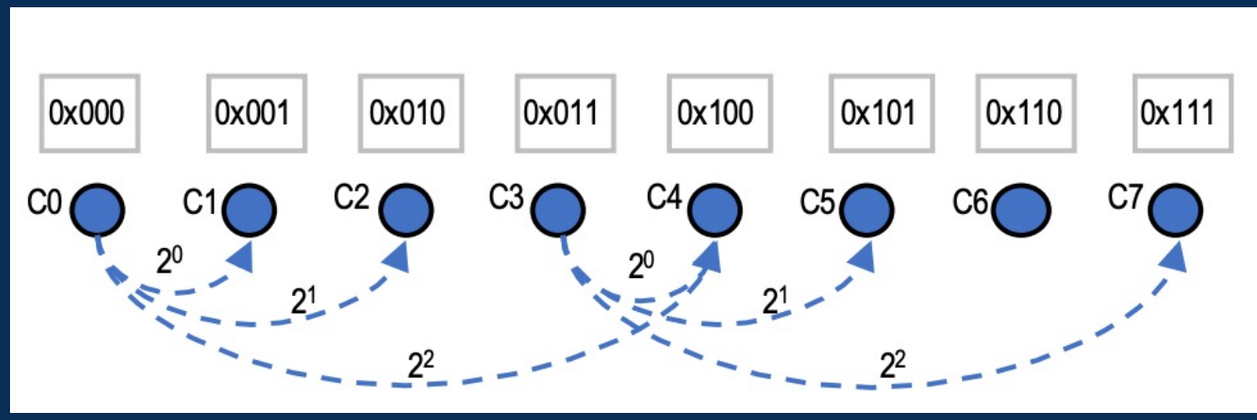


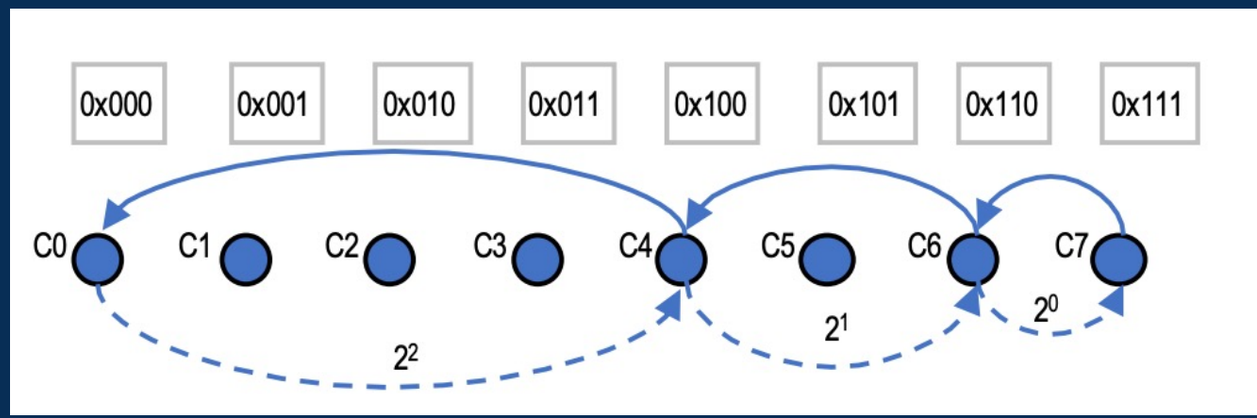
- Leader gossips the block using IDA-gossip and creates the block header H_i that contains the iteration number as well as the root of the Merkle tree from IDA-Gossip.



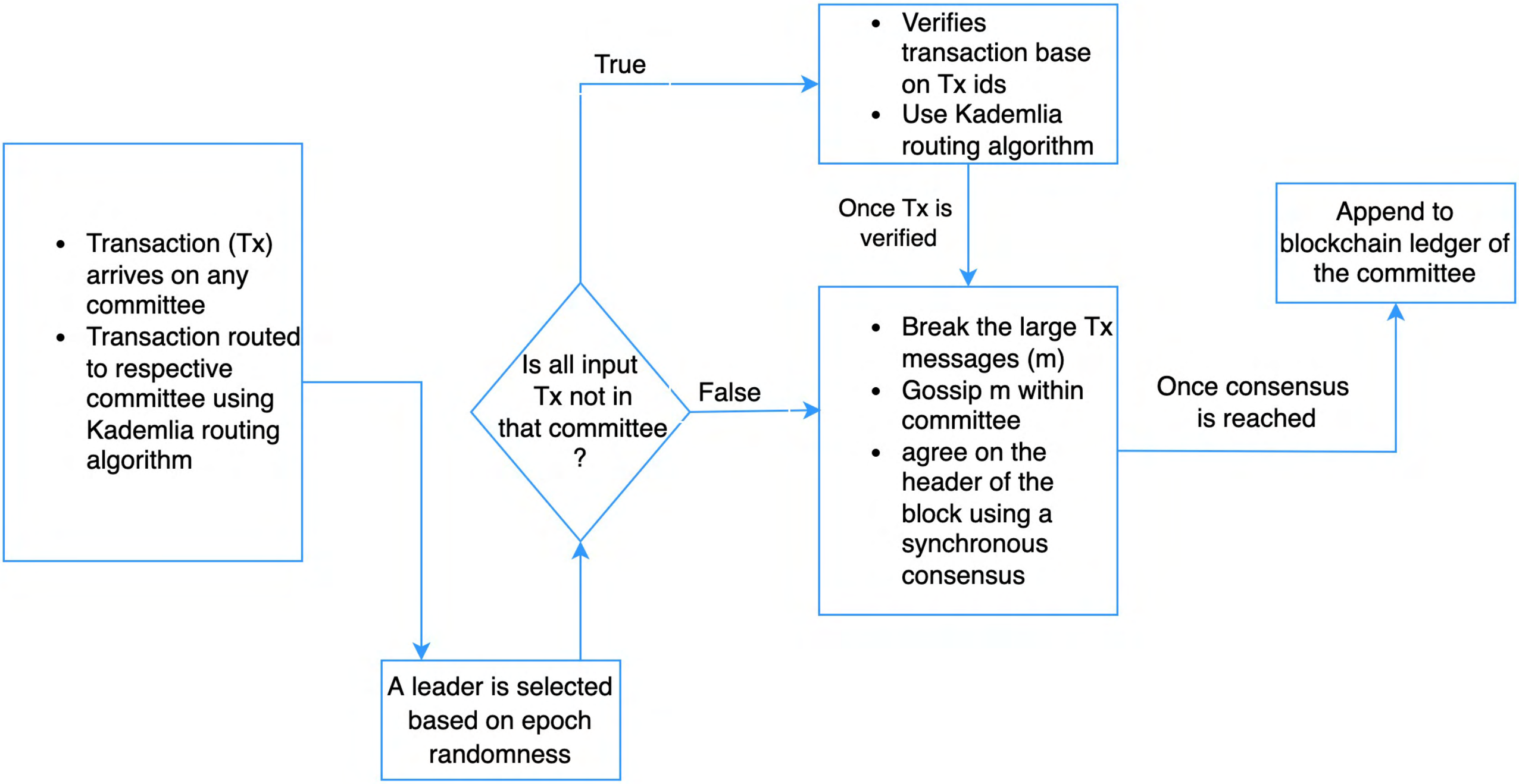
Inter-Committee Routing

- Kademlia routing algorithm
- Each committee maintains a routing table of log n records that point to log n different committees
- Committee C0 wants to locate committee C7 (via C4 and C6) responsible for transactions with prefix 0x111.





How Transaction process



Why Reconfiguration?

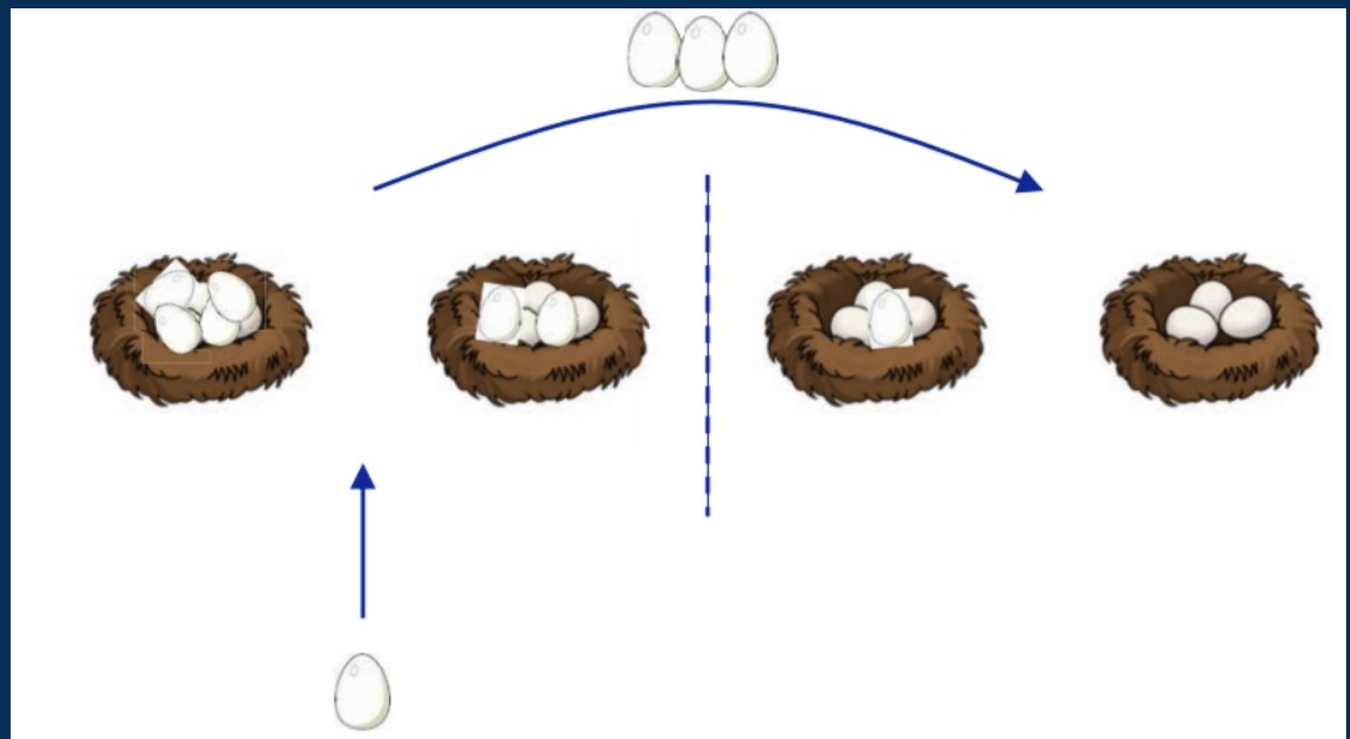
- Join/Leave attacks: Corrupt nodes could rejoin the network to take control of a committee.
  - Cuckoo rule
- Malicious nodes can corrupt the good nodes
  - POW

Reconfiguration

- Every committee gets new block with updated members
- Offline PoW
  - In each epoch, every node that wants to join or stay in the protocol must solve a PoW puzzle during 10 minute.
  - Reference committee is responsible to verify PoW result
- 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
- Move k nodes from the shard, not including the new node
- Assign these k nodes to another committee
- New node needs to download only the set of unspent transactions (UTXOs) from that committee



Evaluation

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*

Conclusion

- Byzantine faults from up to 1/3 of its participants
- Achieve 7,300 tx/sec in a network of 4,000 nodes
- Fast gossip of large messages
- Sharding with storage, communication, computation

Thank you!

AUGUSTA UNIVERSITY