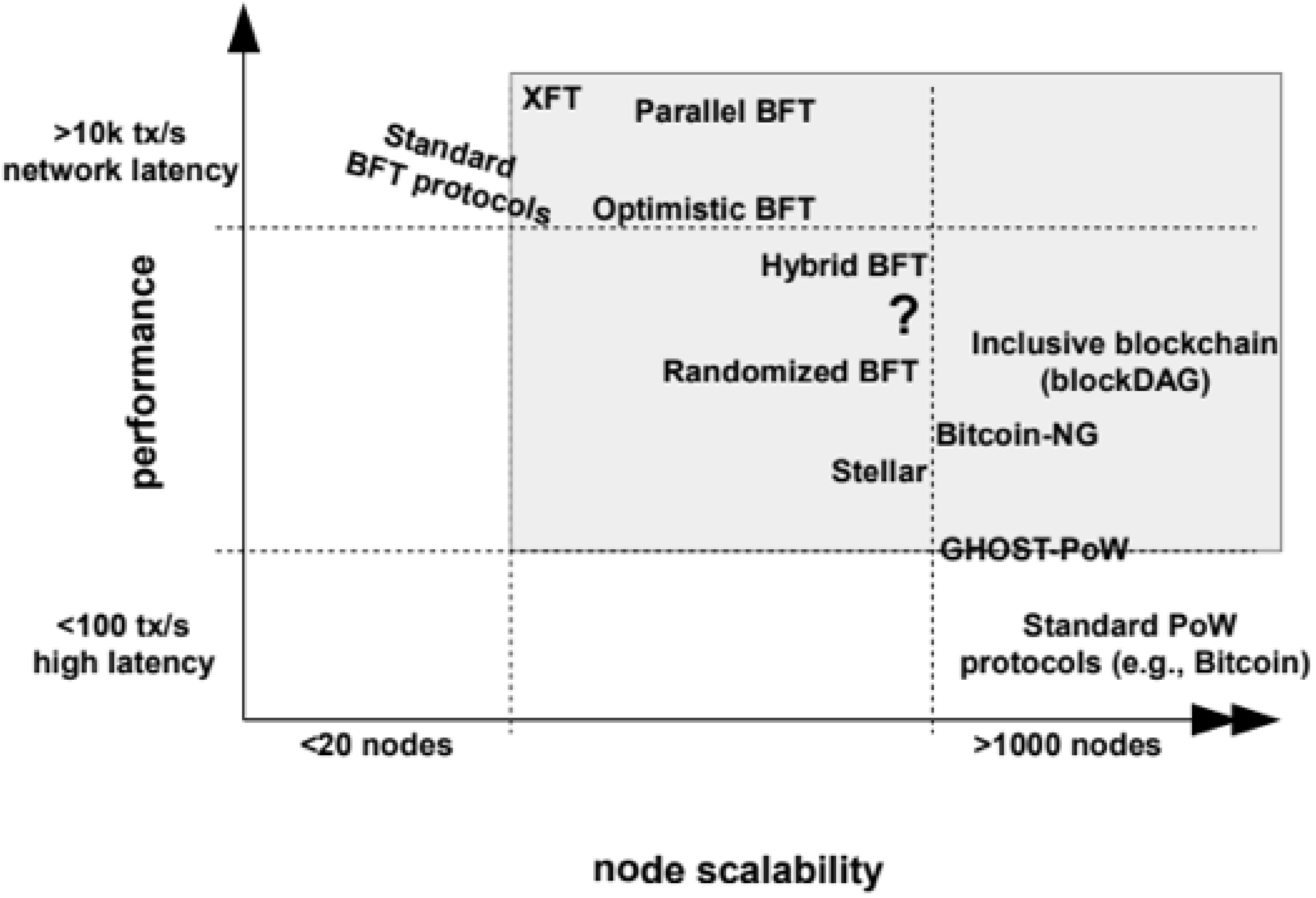
Advanced Consensus Mechanisms

Department of Computer Science and Engineering

INDIAN INSTITUTE OF TECHNOLOGY KHARAGPUR

Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

Consensus Scalability



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.

PoW Consensus vs BFT Consensus

	PoW consensus	BFT consensus
Node identity management	**open, entirely decentralized**	permissioned, nodes need to know IDs of all other nodes
Consensus finality	no	**yes**
Scalability (no. of nodes)	**excellent (thousands of nodes)**	limited, not well explored (tested only up to $n \leq 20$ nodes)
Scalability (no. of clients)	**excellent (thousands of clients)**	**excellent (thousands of clients)**
Performance (throughput)	limited (due to possible of chain forks)	**excellent (tens of thousands tx/sec)**
Performance (latency)	high latency (due to multi-block confirmations)	**excellent (matches network latency)**
Power consumption	very poor (PoW wastes energy)	**good**
Tolerated power of an adversary	$\leq 25\%$ computing power	$\leq 33\%$ voting power
Network synchrony assumptions	physical clock timestamps (e.g., for block validity)	**none for consensus safety** (synchrony needed for liveness)
Correctness proofs	no	**yes**

Towards a Scalable Consensus

Bitcoin-NG

Eyal, I., Gencer, A. E., Sirer, E. G., & Van Renesse, R. (2016, March). **Bitcoin-NG: A Scalable Blockchain Protocol**. In *NSDI 2016*

Issues with Nakamoto Consensus

- **Transaction scalability**
  - Block frequency of 10 minutes and block size of 1 MB during mining reduces the transactions supported per second

- **Issues with Forks**
  - Prevents consensus finality
  - Makes the system unfair - a miner with poor connectivity has always in a disadvantageous position

Bitcoin-NG: Decouple Leader Election

- Bitcoin - think of the winning miner as the **leader** - the leader serializes the transactions and include a new block in the blockchain

- Decouple Bitcoin’s blockchain operations into two planes
  - **Leader election**: Use PoW to randomly select a leader (an infrequent operation)
  - **Transaction Serialization**: The leader serializes the transaction until a new leader is elected

Bitcoin vs Bitcoin-NG

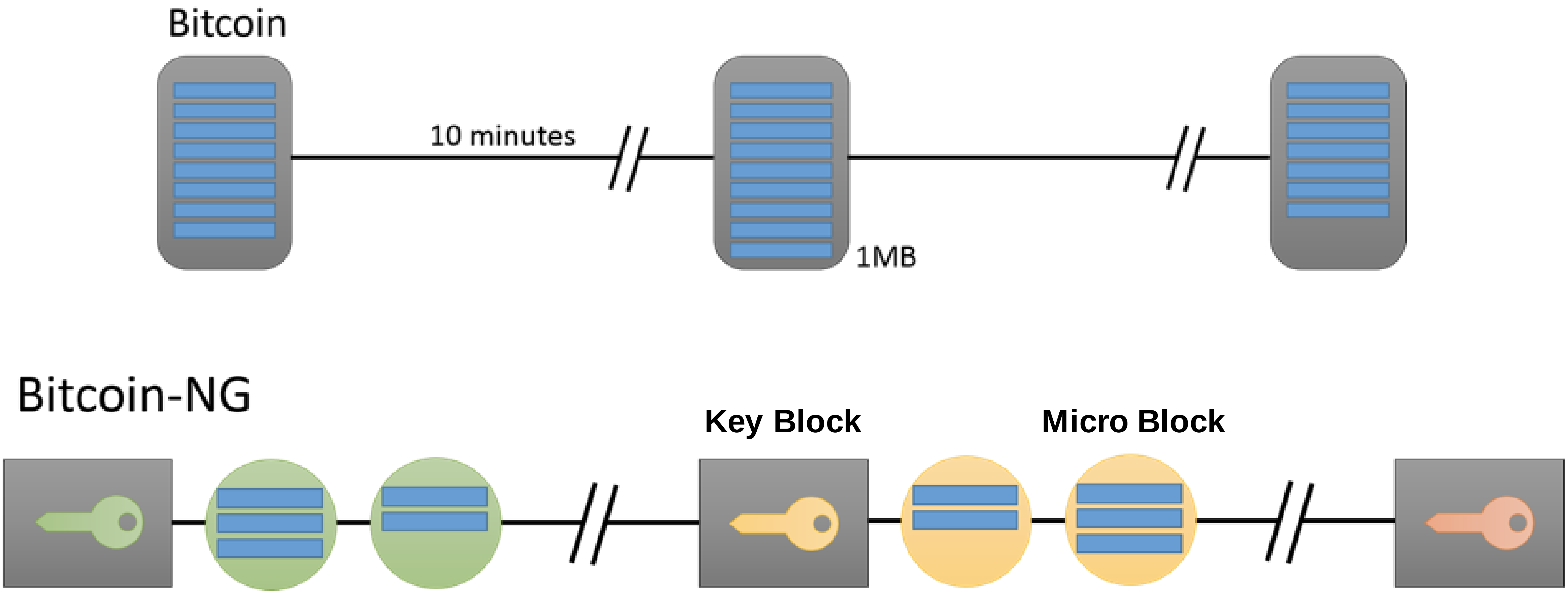


Image Source: http://hackingdistributed.com/2015/10/14/bitcoin-ng/

Bitcoin-NG : Key Blocks

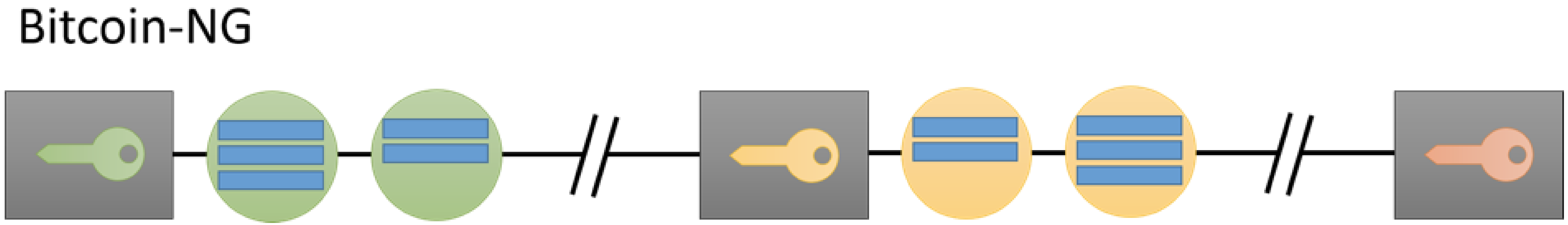
- Key blocks are used to choose a leader (similar to Bitcoin)

- A key block contains
  - The reference to the previous block
  - The current Unix time
  - A coinbase transaction to pay of the reward
  - A target hash value
  - A nonce field

Key Blocks

- Key blocks are generated based on regular Bitcoin mining procedure
  - Find out the nonce such that the block hash is less than the target value

- Key blocks are generated infrequently - the intervals between two key blocks is exponentially distributed



Bitcoin-NG : Microblocks

- Once a node generates a key block, it becomes the **leader** and generates further microblocks
  - Microblocks are generates at a set rate smaller than a predefined maximum
  - The rate is much higher than the key block generation rate

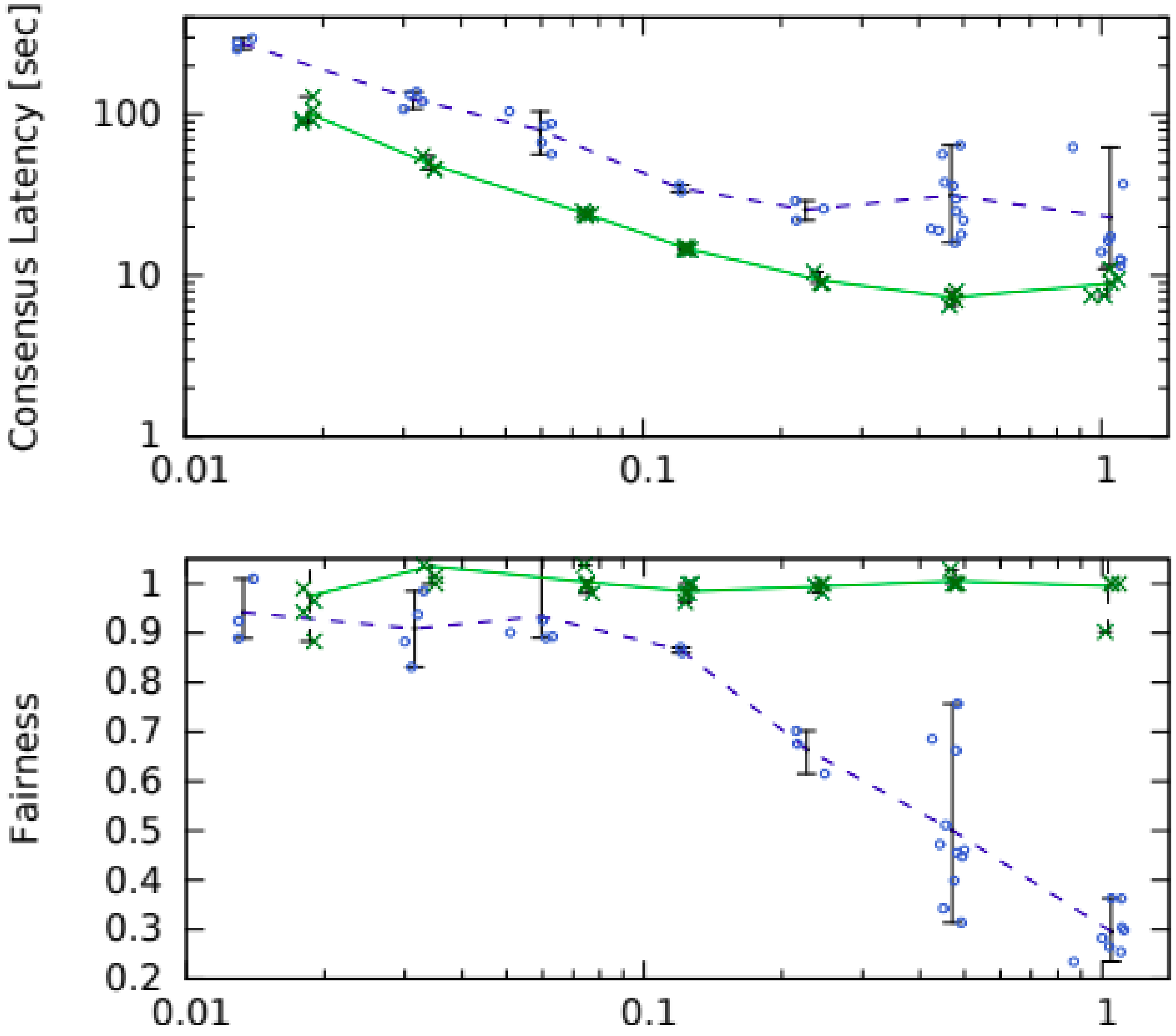
- A microblock contains
  - Ledger entries
  - Header
    - Reference to the previous block
    - The current Unix time
    - A cryptographic hash of the ledger entries (Markle root)
    - A cryptographic signature of the header (signature of the key block miner)

Microblock Fork

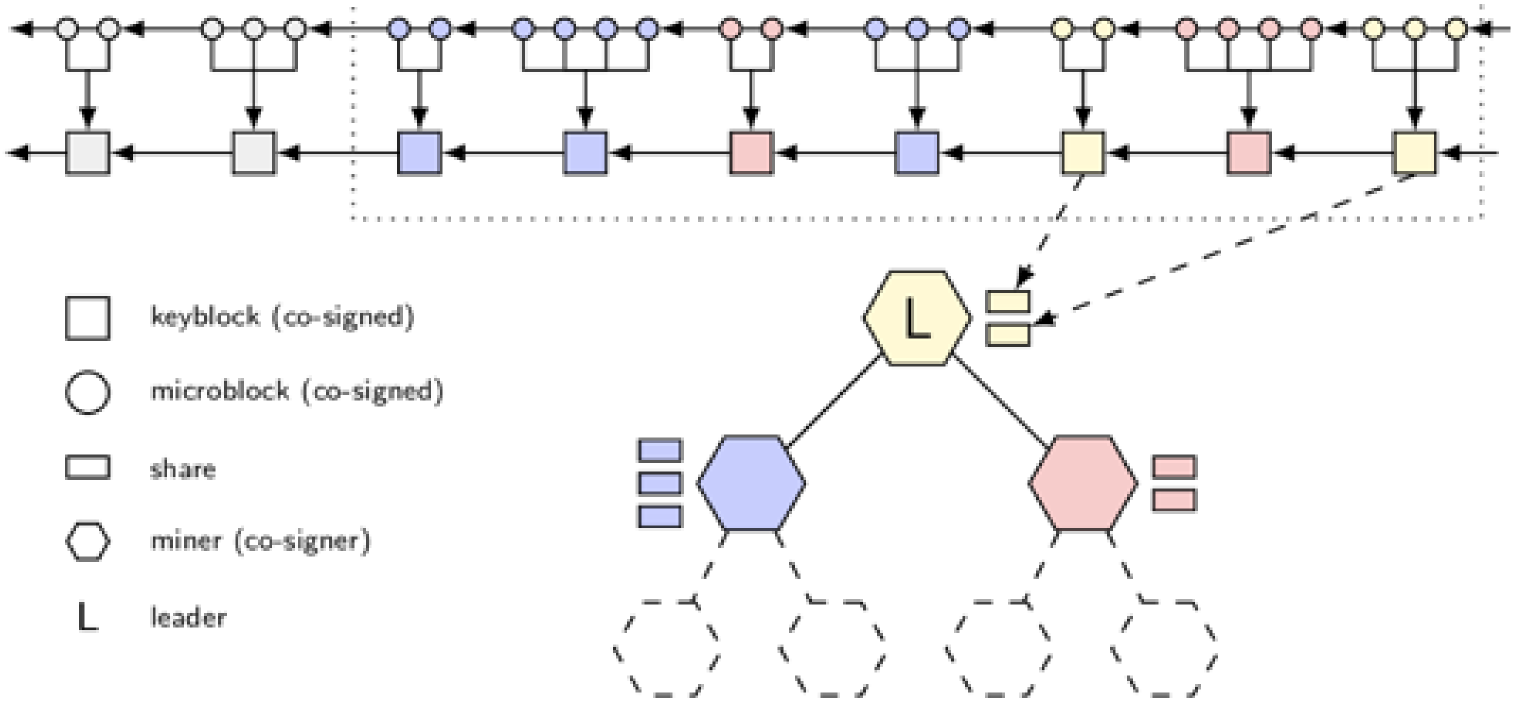
- When a miner generates a key block, he may not have heard of all microblocks generated by the previous leader
  - Common if microblock generation is frequent
  - May result in microblock fork

- A node may hear a forked microblock but not the new key block
  - This can be prevented by ensuring the reception of the key block
  - When a node sees a microblock, it waits for propagation time of the network, to make sure it is not pruned by a new key block

Bitcoin-NG Performance



Combining PoW with PBFT : Byzcoin



Kogias, E. K., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., & Ford, B. (2016, August). **Enhancing bitcoin security and performance with strong consistency via collective signing**. In *25th USENIX Security Symposium 2016*

Collective Signing (CoSi)

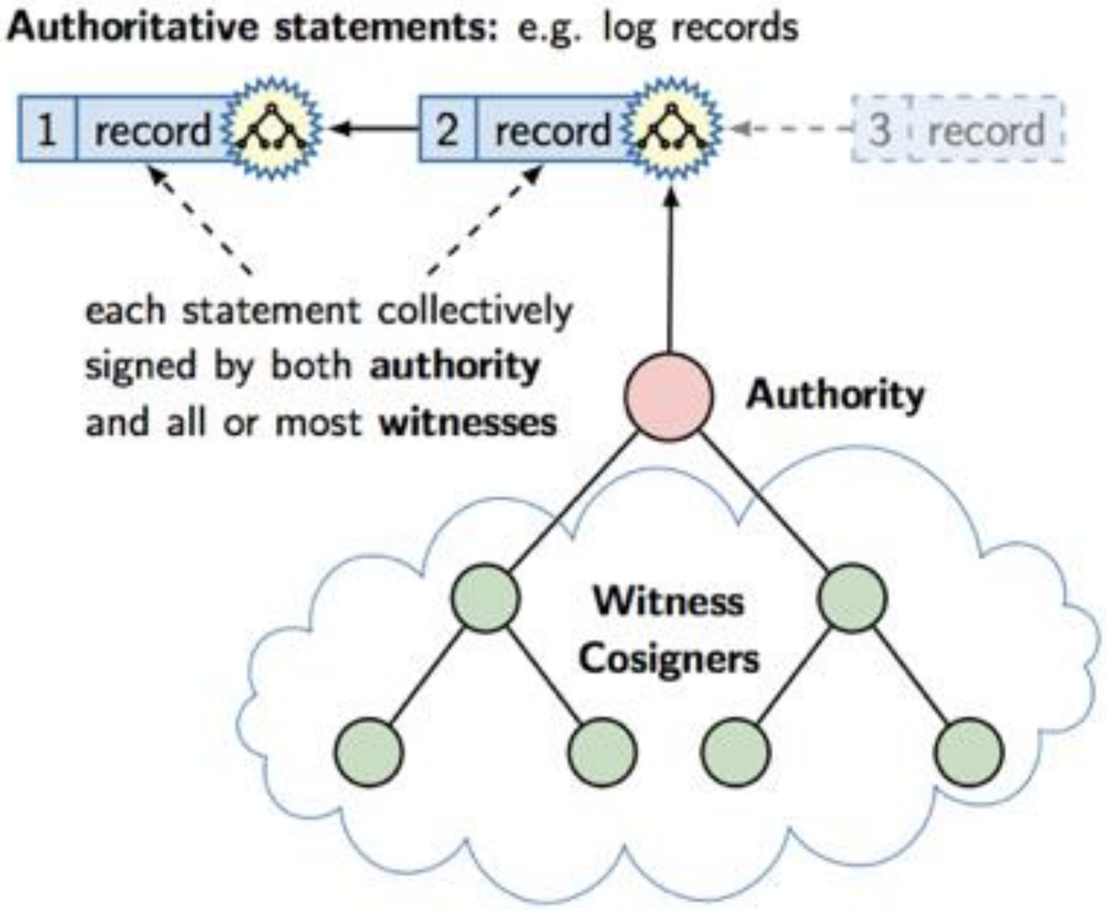
- Method to protect "authorities and their clients" from undetected misuse or exploits
- A **scalable witness cosigning protocol** ensuring that every authoritative statement is validated and publicly logged by a diverse group of witnesses before any client accepts it
- A statement *S* collectively signed by *W* witnesses assures clients that *S* has been seen, and not immediately found erroneous, by those *W* observers.

Syta, Ewa, *et al*. "**Keeping authorities "honest or bust" with decentralized witness cosigning**" *2016 IEEE Symposium on Security and Privacy (SP)*, 2016.

CoSi Architecture

- The leader organizes the witnesses in a tree structure – a scalable way of aggregating signatures coming from the children

- Three rounds of PBFT (pre-prepare, prepare and commit) can be simulated using two rounds of CoSi protocol



Schnorr Multi-signature

- The basic CoSi protocol uses **Schnorr multisignatures**, that rely on a group G of prime order
  - *Discrete logarithmic problem is believed to be hard*

- **Key Generation:**
  - Let G be a group of prime order r. Let g be a generator of G.
  - Select a random integer x in the interval $[0, r-1]$. x is the private key and g^x is the public key.
  - N signers with individual private keys $x_1, x_2, \ldots, x_N$, and the corresponding public keys $g^{x_1}, g^{x_2}, \ldots, g^{x_N}$

Schnorr Multi-signature

- **Signing:**
  - Each signer picks up the random secret v_i, generates $V_i = g^{v_i}$
  - The leader collects all such V_i, aggregates them $V = \prod V_i$, and uses a hash function to compute a collective challenge $c = H(V||S)$. This challenge is forwarded to all the signers.
  - The signers send the response $r_i = v_i - cx_i$. The leader computes the aggregated as $r = \sum r_i$. The signature is (c, r).

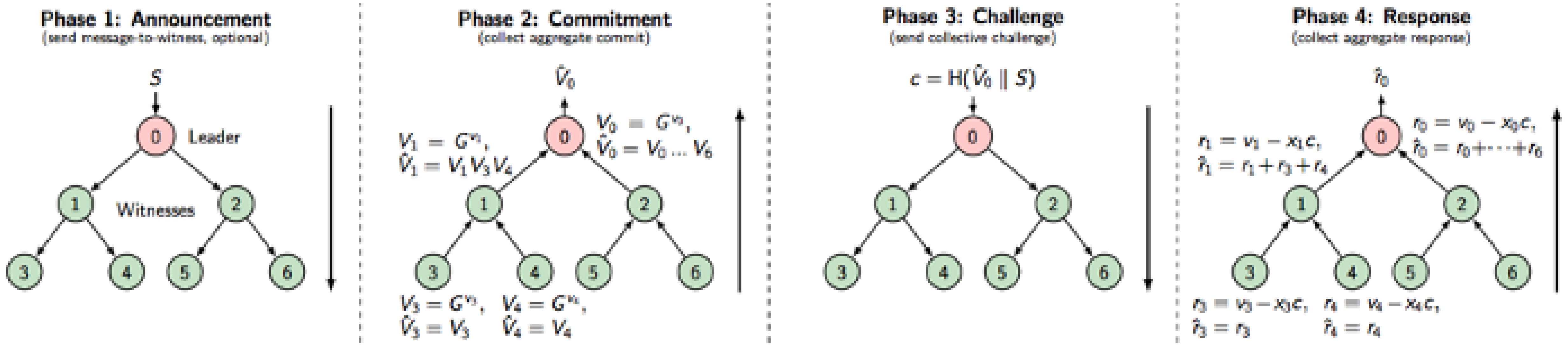
Schnorr Multi-signature

- **Verification:**
  - The verification key is $y = \prod g^{x_i}$
  - The signature is (c, r), where $c = H(V||S)$ and $r = \sum r_i$
  - Let $V_v = g^r y^c$
  - Let $r_v = H(V_v||S)$
  - If $r_v = r$, then the signature is verified

Schnorr Multi-signature

- **Proof:**
  - The verification key is $y = \prod g^{x_i}$
  - The signature is (c, r), where $c = H(V||S)$ and $r = \sum r_i$
  - $V_v = g^r y^c = g^{\sum(v_i - cx_i)} \prod g^{cx_i} = g^{\sum(v_i - cx_i)} g^{\sum cx_i} = g^{\sum v_i} = \prod g^{v_i} = \prod V_i = V$
  - So, $r_v = H(V_v||S) = H(V||S) = r$

CoSi Protocol



- One CoSi round to implement PBFT's pre-prepare and prepare phases
- Second CoSi round to implement PBFT's commit phase

Revisiting the Requirements for Blockchain Consensus

- **Byzantine fault tolerant** – the system should work even in the presence of malicious users while operating across multiple administrative domains
- Should provide **strong consistency guarantee** across replicas
- Should **scale well to increasing workloads** in terms of transactions processed per unit time
- Should **scale well to increasing network size**

Bitcoin-NG: The Issue with a Faulty Key Block

- **Problem with Bitcoin-NG:** A faulty key block is verified only after end of the round
  - A faulty miner can introduce a number of correct microblocks following a faulty microblock in the system - certainly a overhead for the application - **a fork alleviates the problem further**

Bitcoin-NG: The Issue with a Faulty Key Block

- **Problem with Bitcoin-NG:** A faulty key block is verified only after end of the round
  - A faulty miner can introduce a number of correct microblocks following a faulty microblock in the system - certainly a overhead for the application - **a fork alleviates the problem further**

Solve this problem by a set of PBFT verifiers - who will verify a block and then only the block is added in the Blockchain

Issues with PBFT

- PBFT requires a **static consensus group** (because of message passing)

- **Scalability** (in terms of nodes) is a problem for PBFT
  - $O(n^2)$ communication complexity
  - $O(n)$ verification complexity
  - Absence of third-party verifiable proofs (PBFT uses MAC - need to share the keys among the miners)

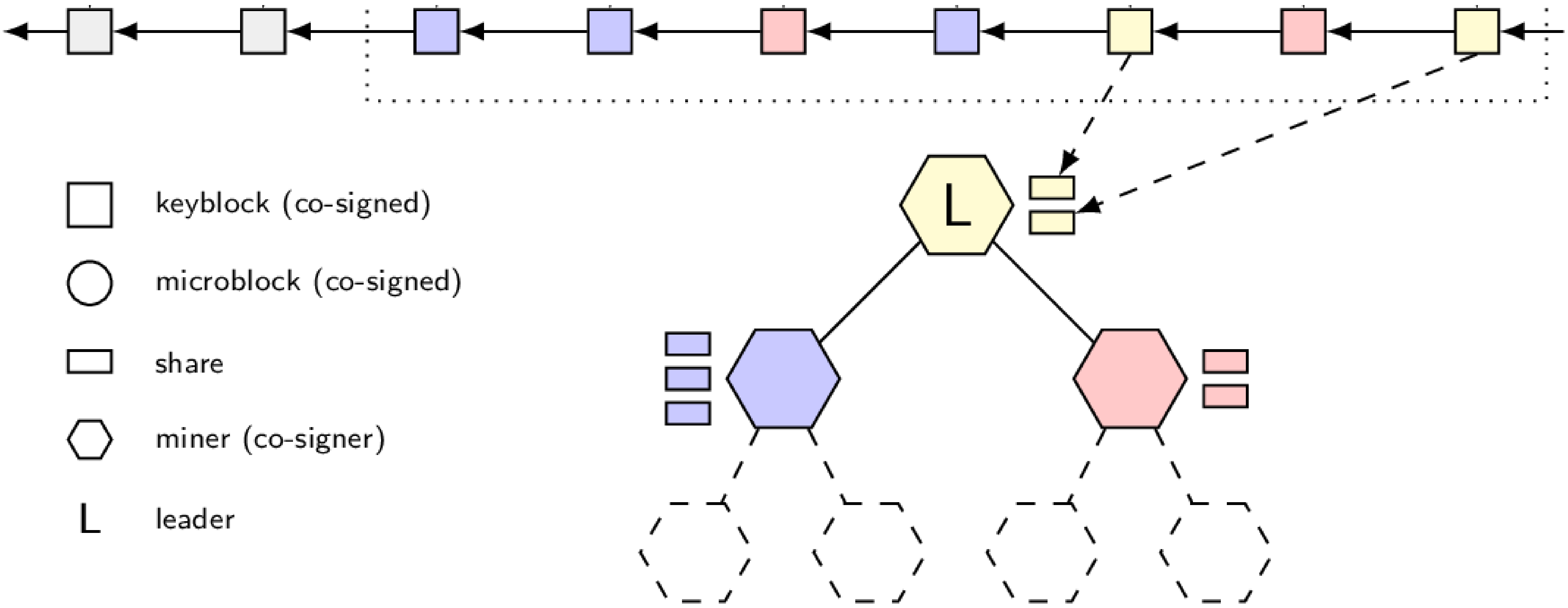
- **Sybil attack** - create multiple pseudonymous identities to subvert the ***3f+1*** requirements of PBFT

Open the Consensus Group

- Use PoW based system to give a *proof of membership* of a miner as a part of the trustees

- Maintains a “balance of power” within the BFT consensus group
  - Use a fixed-size sliding window
  - Each time a miner finds a new block, it receives a *consensus group share*
  - The share proves the miner’s membership in the trustee group

Merging BFT Consensus with PoW



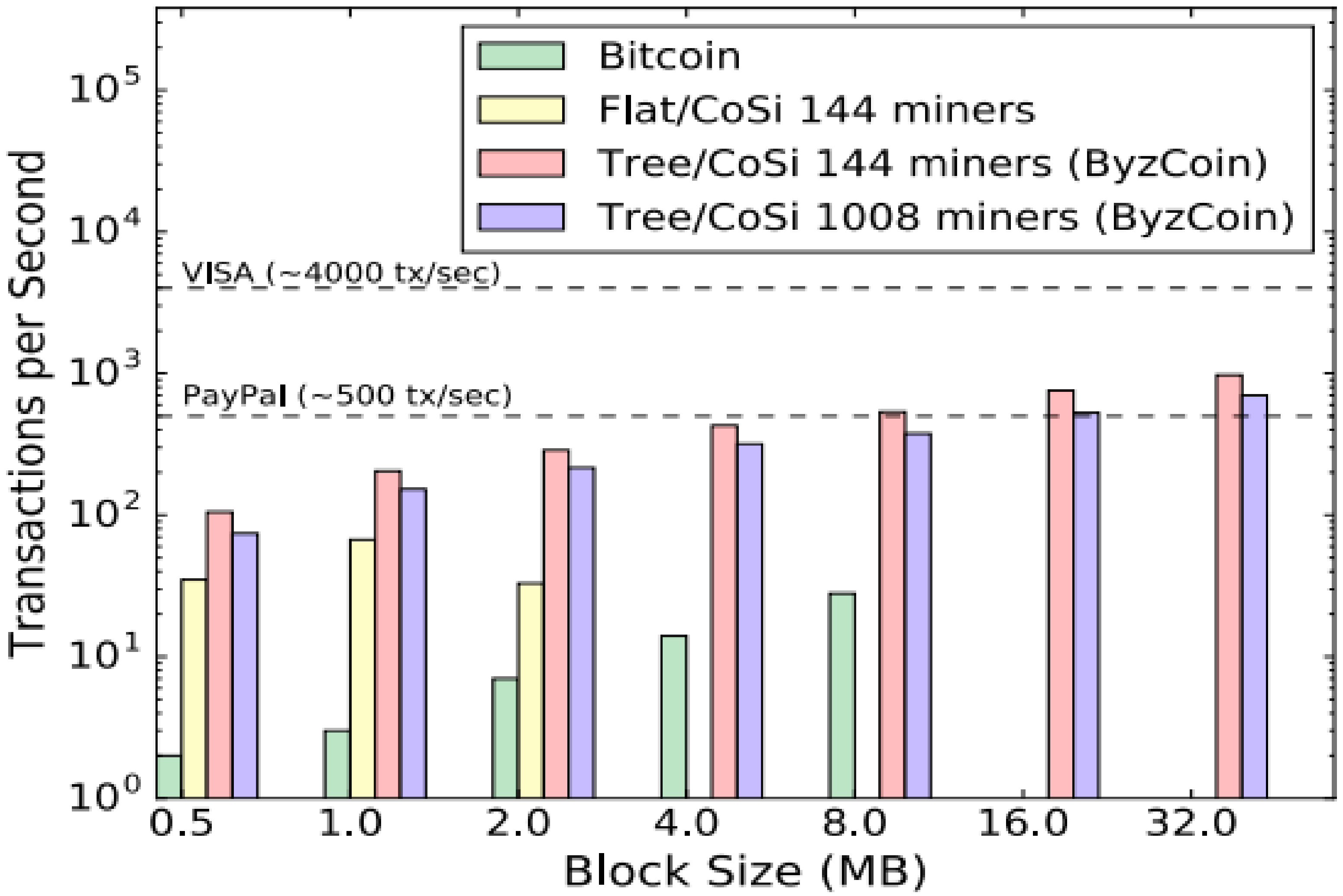
Improving the Efficiency of BFT Consensus

- **Improve O(n) communication complexity**
  - Use tree-based multicast protocol - share information with O(log n)

- **Improve O(n) complexity for verification**
  - Use Schnorr multi-signatures
  - Verification can be done in O(1) through signature aggregation

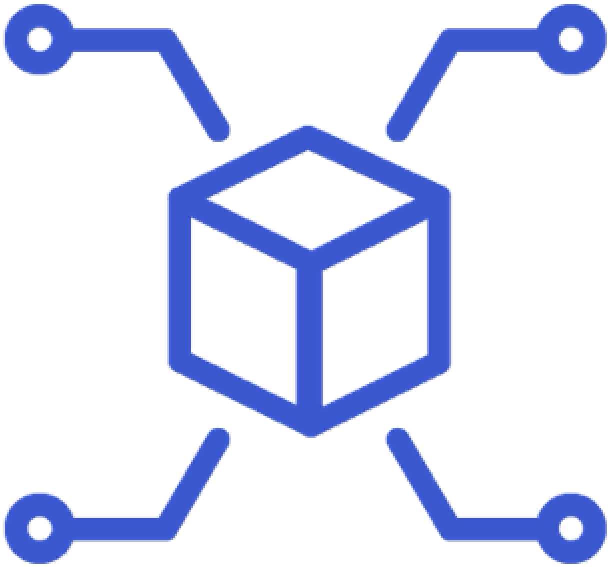
- Multi-signatures + Communication trees = **CoSi**

Byzcoin Performance



Algorand

Gilad, Y., Hemo, R., Micali, S., Vlachos, G., & Zeldovich, N. (2017, October). *Algorand: Scaling byzantine agreements for cryptocurrencies.* In *Proceedings of the 26th Symposium on Operating Systems Principles* (pp. 51-68). ACM.



Algorand: Scaling Byzantine Agreements for Cryptocurrencies

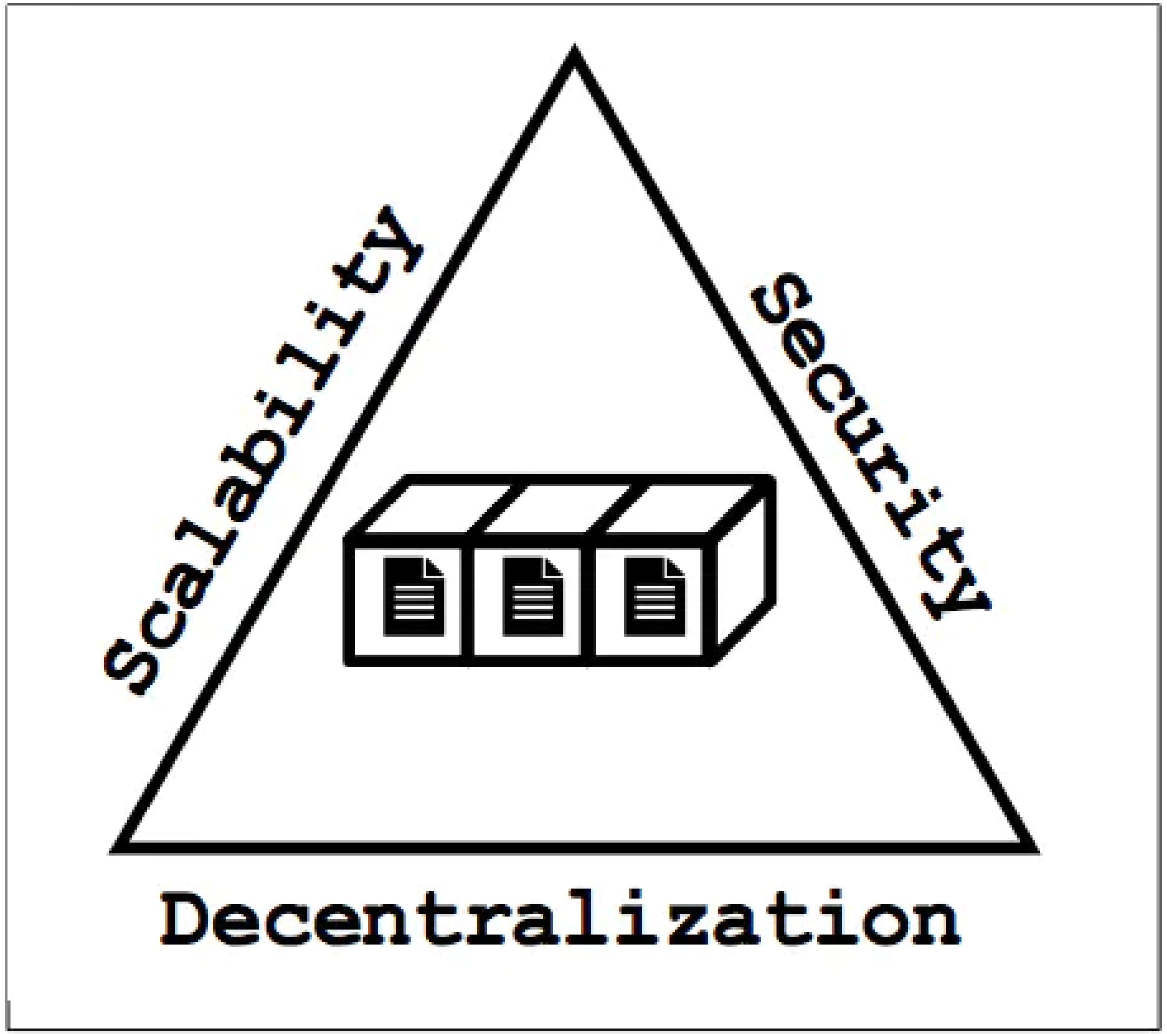
Bitcoin: Recap

- **Key Idea**:
  - Consensus through proof-of-work (PoW)
- **Communication**:
  - Gossip protocol
- **Key Assumption**:
  - Honest majority of mining computation power

Bitcoin: Limitations

- **Resource wastage**:
  - high computational, electricity cost
- **Concentration of power**
  - only ~5 mining pools control the entire system
- **Vulnerable**
  - easy to track miners, concentrated to a few mining pools - https://www.blockchain.com/btc/blocks?page=1
- **Scalability**
  - number of users not clear (1M, 10M, 100M??), high latency(~10minutes)
- **Ambiguity**
  - fork in blockchain

The Blockchain Performance Triangle



Algorand: Overview

- **Key Idea**:
  - Consensus through Byzantine Agreement Protocol
- **Communication**:
  - Gossip protocol
- **Key Assumption**:
  - Honest majority of money

Algorand: Technical Advancement

- **Trivial computation**
  - simple operation like add, count
- **True decentralization**
  - no concentration of mining pool power, all equal miners and users
- **Finality of payment**
  - fork with very low probability, block appears and payment fixed forever
- **Scalability**
  - millions of users, only network latency (~1minute)
- **Security**
  - against bad adversary

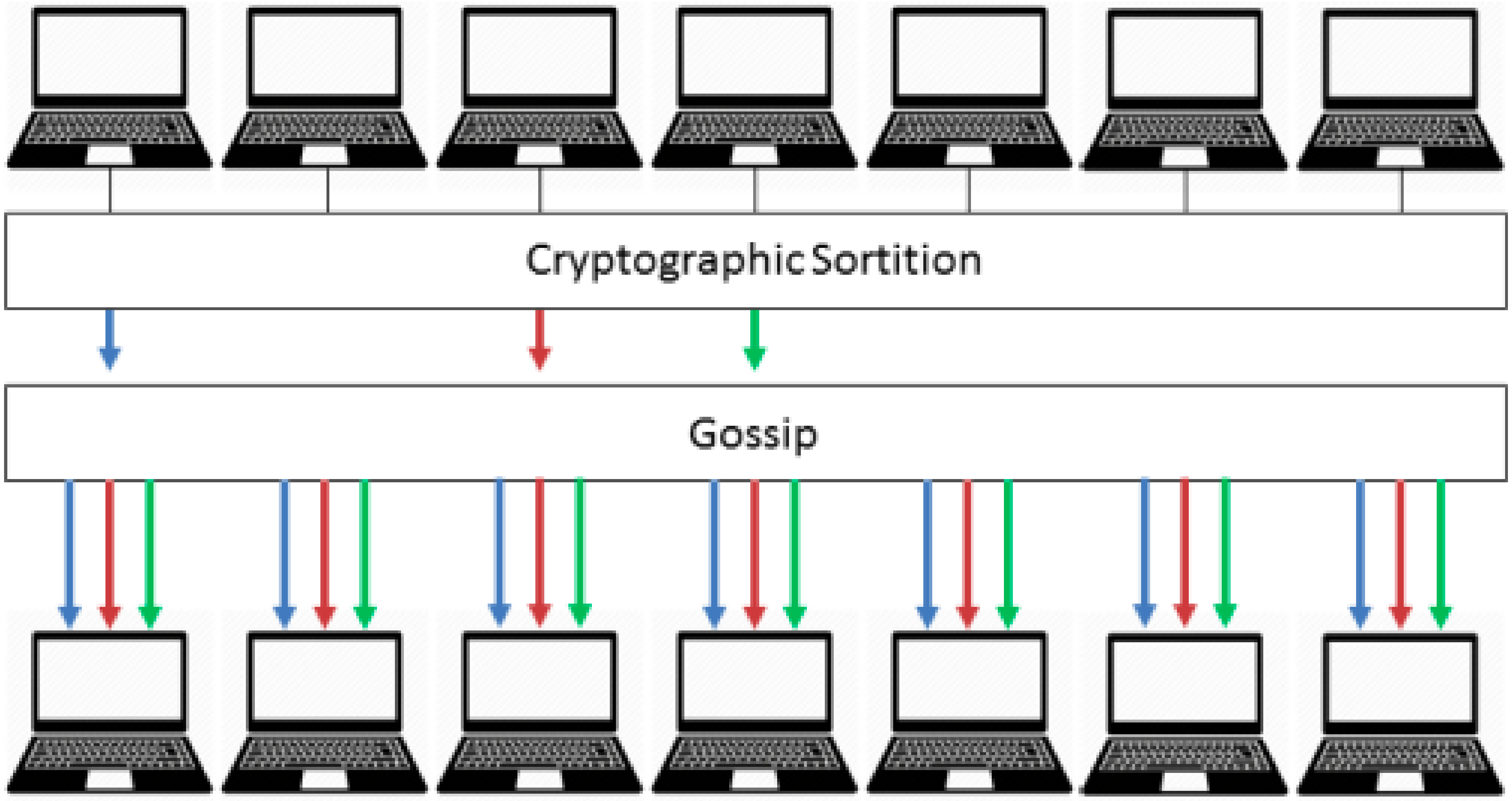
Architecture of Algorand

- Select a random user
  - prepare a block
  - propagate block through gossiping

- Select random committee with small number of users (~10k)
  - run Byzantine Agreement on the block
  - digitally sign the result
  - propagate digital signatures

- **Who select the committee?**

Cryptographic Sortition in Algorand

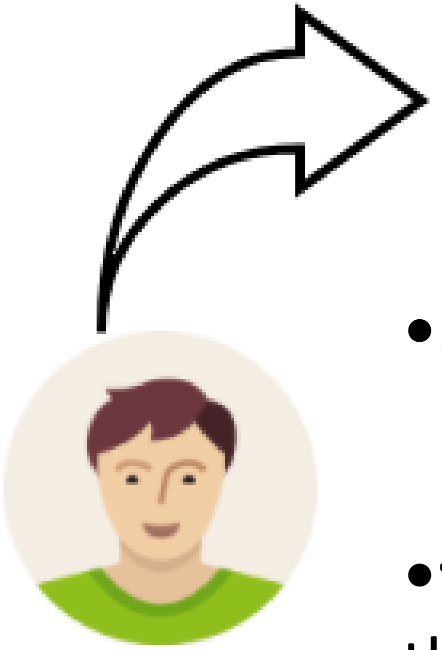


Cryptographic Sortition

- Each committee member selects himself according to per-user weights
  - Implemented using **<u>Verifiable Random Functions</u> (VRFs)**

- <hash,proof> $\leftarrow$ $VRF_{sk}(x)$
  - **x:** input string
  - **(pk_i,sk_i):** public/private key pair
  - **hash:** hashlenbit-long value that is uniquely determined by sk and x
  - **proof:** enables to check the hash indeed corresponds to x

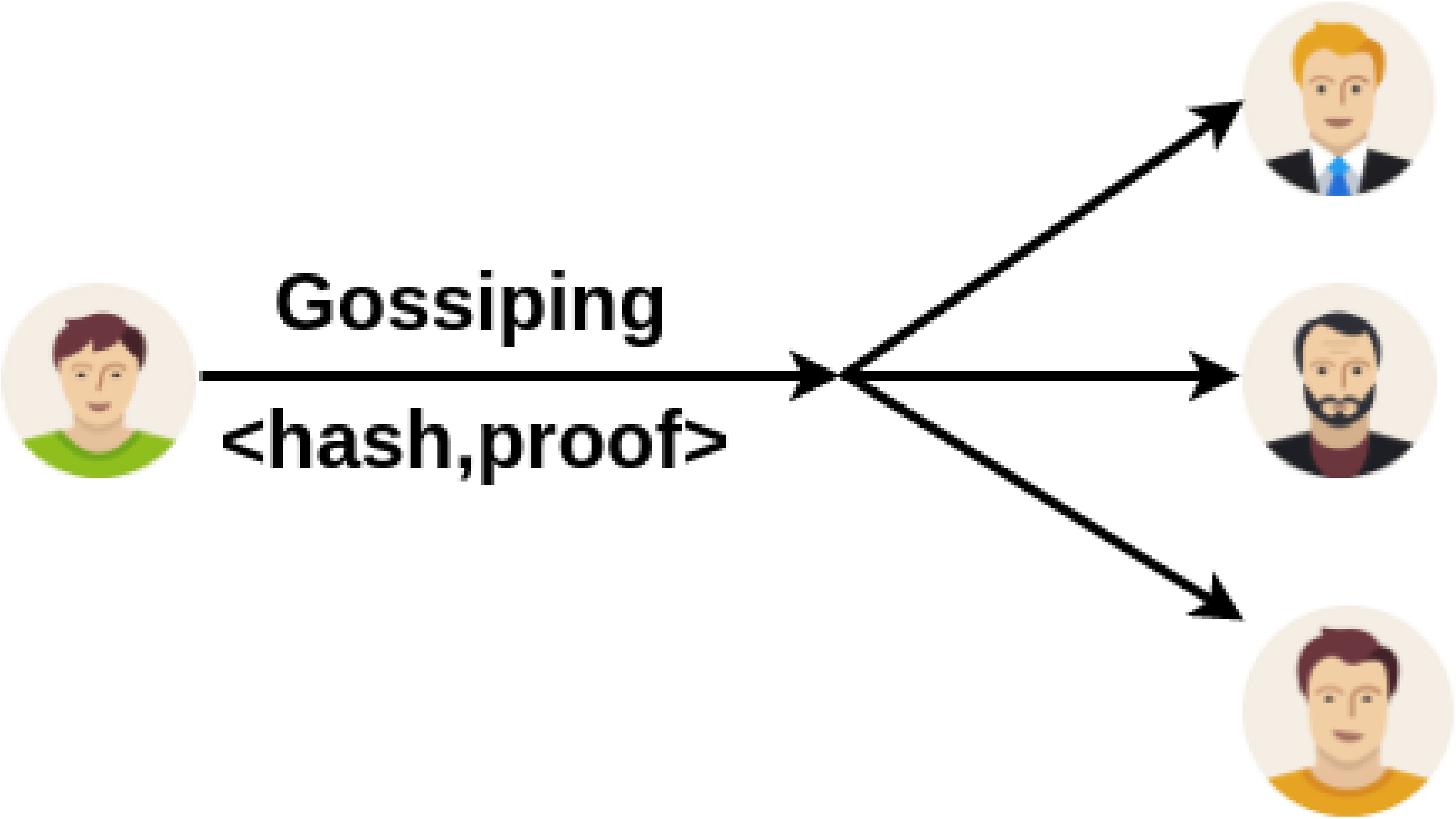
Cryptographic Sorition in Algorand: Committee Member Selection



<hash,proof,j> <--- Sortition(sk,seed,threshold,role,w,W)

- **seed:** publicly known random value
  - seed published at Algorand's round r using VRFs with the seed of the previous round r − 1
- **threshold:** determines the expected number of users selected for that role
- **role:** user for proposing a block/ committee member
- **w:** weight of a user
- **W:** weight of all users
- **j:** user gets to participate as j different "sub-users."

Cryptographic Sortition in Algorand



Cryptographic Sortition: Proof Verification

<j> <---

VerifySort(pk,hash,proof,seed,threshold,role,w,W)

- Check if that user was selected
  - number of selected sub-users
  - zero if the user was not selected at all

Byzantine Agreement in Algorand – BA*

- **Two phase**:
  - Two phase agreement – *Final Consensus* and *Tentative Consensus*

- **Strong Synchrony**: Most honest users (say, 95%) can send message that will be received by most other honest users within a known time bound
  - Adversary can not control the network for long
  - Ensures liveness of the protocol

- **Weak Synchrony**: The network can be asynchronous for long (entirely controlled by adversary) but bounded period of time
  - **There must be a strong synchrony period after a weak synchrony period**
  - Algorand is **safe** under weak synchrony

Byzantine Agreement in Algorand – BA*

- **Two phase:**
  - Two phase agreement – *Final Consensus* and *Tentative Consensus*

- **Strong Synchrony**: Most honest users (say, 95%) can send message that will be received by most other honest users within a known time bound
  - Adversary can not control the network for long
  - Ensures liveness of the protocol

- **Weak Synchrony**: The network can be asynchronous for long (entirely controlled by adversary) but bounded period of time
  - **There must be a strong synchrony period after a weak synchrony period**
  - Algorand is **safe** under weak synchrony

Byzantine Agreement in Algorand – BA*

- **Two phase**:
  - Two phase agreement – *Final Consensus* and *Tentative Consensus*

- **Strong Synchrony**: Most honest users (say, 95%) can send message that will be received by most other honest users within a known time bound
  - Adversary can not control the network for long
  - Ensures liveness of the protocol

- **Weak Synchrony**: The network can be asynchronous for long (entirely controlled by adversary) but bounded period of time
  - **There must be a strong synchrony period after a weak synchrony period**
  - Algorand is **safe** under weak synchrony

Final Consensus

- One user reaches final consensus
  - Any other user that reaches final or tentative consensus in the same round must agree on the same block value (**ensures safety**)
  - Confirm a transaction when the block reaches to the final consensus

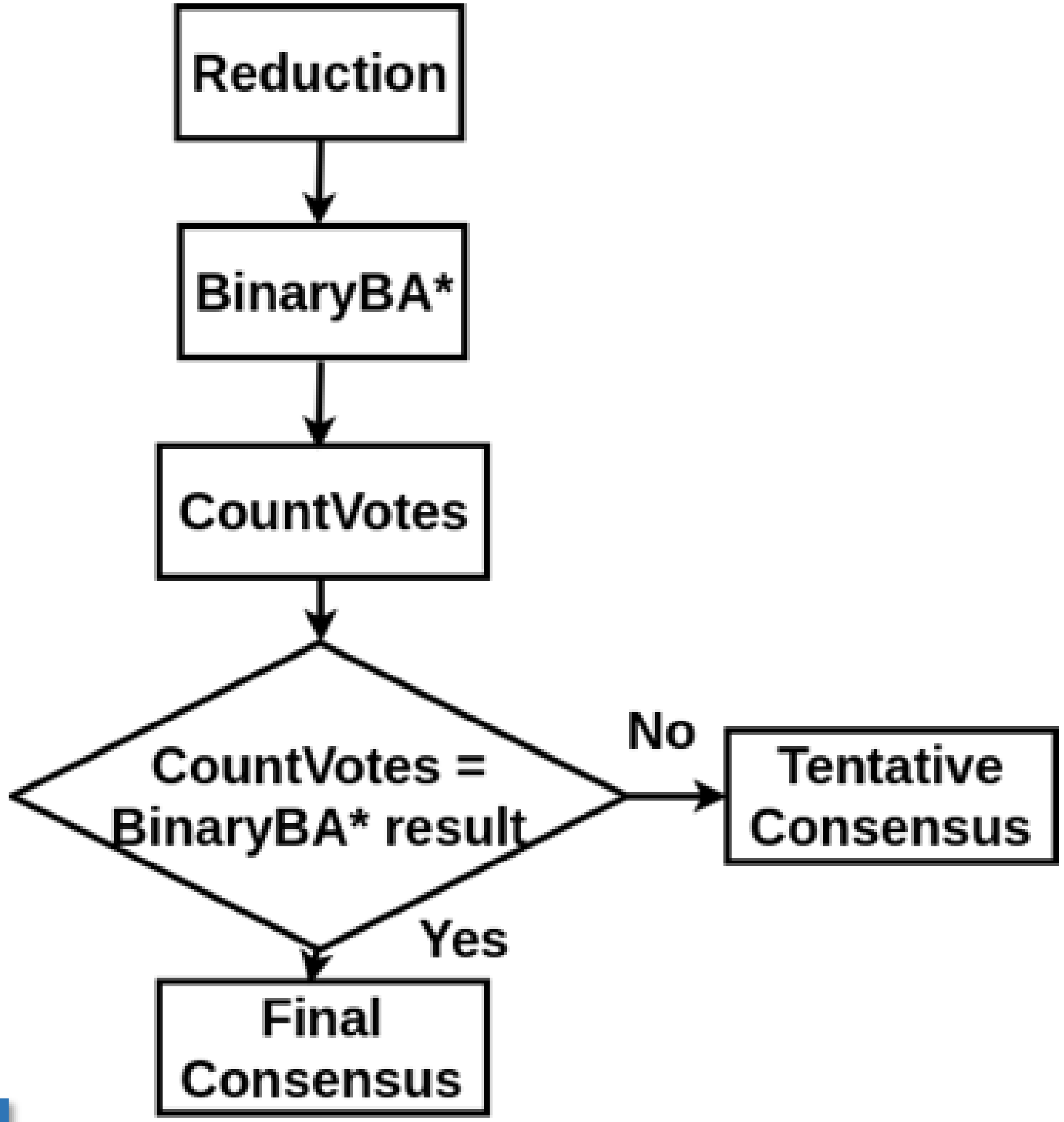
Tentative Consensus

- One user reaches tentative consensus
  - Other users may have reached consensus on a **<u>different (but correct)</u>** block
  - Can be in two cases
    - **<u>The network is strongly synchronous</u>** - adversary may be able to cause BA* to reach tentative consensus on a block - BA* is unable to confirm that the network was strongly synchronous
    - **<u>The network was weakly synchronous</u>** - BA* can form multiple forks and reach tentative consensus on two different blocks - users are split into groups

Coming Out of Tentative Consensus

- Run BA* periodically to come out of tentative consensus - run the next round
  - Network can not be under weak synchrony all the times
  - Cryptographic sortition ensures different committee members at different rounds of the BA*

BA* Overview


Reduction
BinaryBA*
CountVotes
CountVotes = BinaryBA* result
No
Tentative Consensus
Yes
Final Consensus

thank you!