Research Prospective

Consensus Protocols

- Permissionless Blockchain
  - PoW
  - PoS
  - PoET
  - PoB

    .

    .

    .
- Permissioned Blockchain
  - RAFT
  - PAXOS
  - BFT
  - PBFT
  - RBFT (Redundant Byzantine Fault Tolerence)

PoW Vs PBFT

- PoW
  - Open environment
  - Works over a large number of nodes
  - Scalable in terms of number of nodes (+ve)
  - Transaction throughput is low:
    - Very less no. of transactions per block (-Ve)
    - No. of transactions per unit time

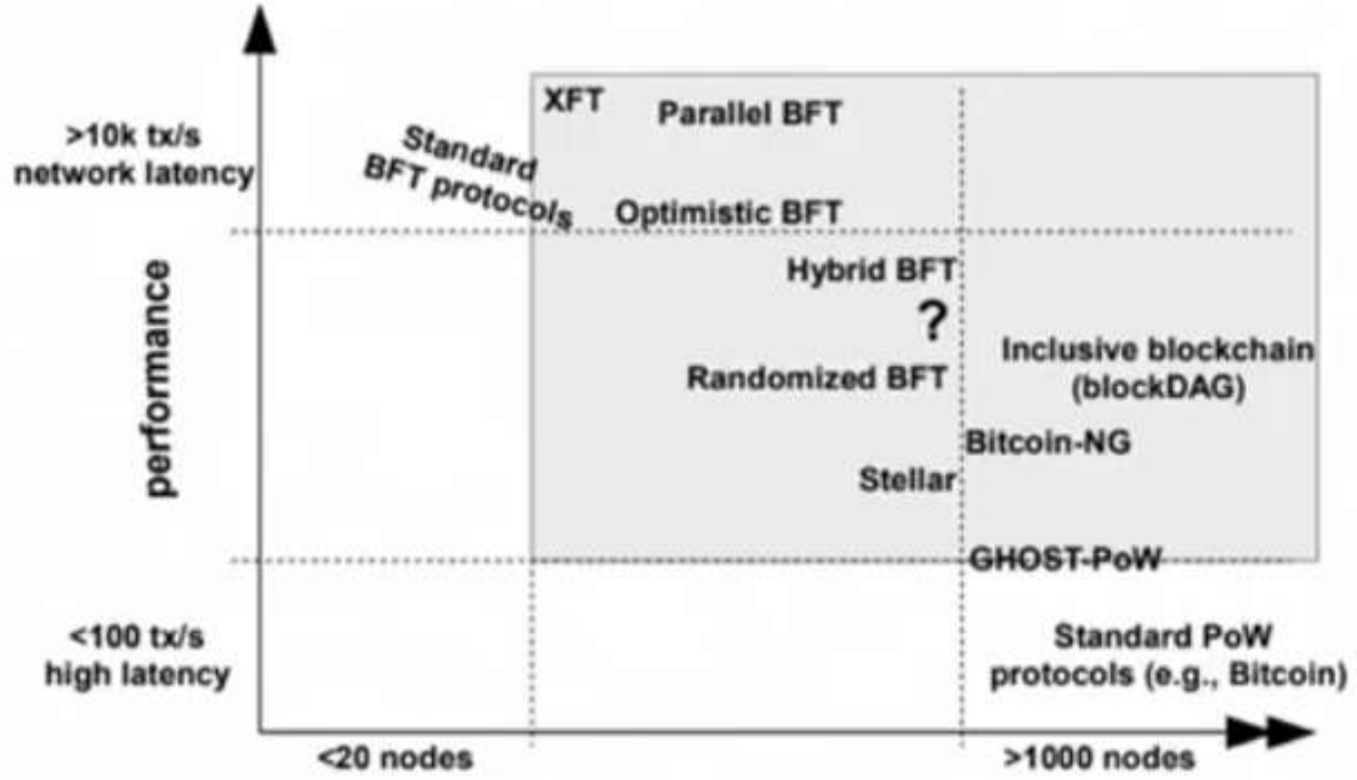
- PBFT
  - Closed
  - Not scalable in terms of number of nodes
  - High transaction throughput (+ve)
  - High message complexity (-ve)

PoW Scalability

- Two magic numbers in PoW
  - **Block frequency** - 10 minutes
  - **Block size** - 1 MB **In 2008, but later it increased upto 8 MB**
- Transaction throughput - 7 transactions per second (with 200-250 bytes transactions)

- VISA or Master card transactions: About 40-45 million per second

Performance vs Scalability for PoW and BFT



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 vs PBFT - Consensus Finality

- *If a correct node p appends block b to its copy of blockchain before appending block b', then no correct node q appends block b' before b to its copy of the blockchain* (Vukolic, 2015)

- PoW is a randomized protocol - does not ensure consensus finality
  - Remember the forks in Bitcoin blockchain

- BFT protocols ensure total ordering of transactions - ensures consensus finality

	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**

Bitcoin-NG

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

Bitcoin-NG

Issues with Nakamoto Consensus (PoW)

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

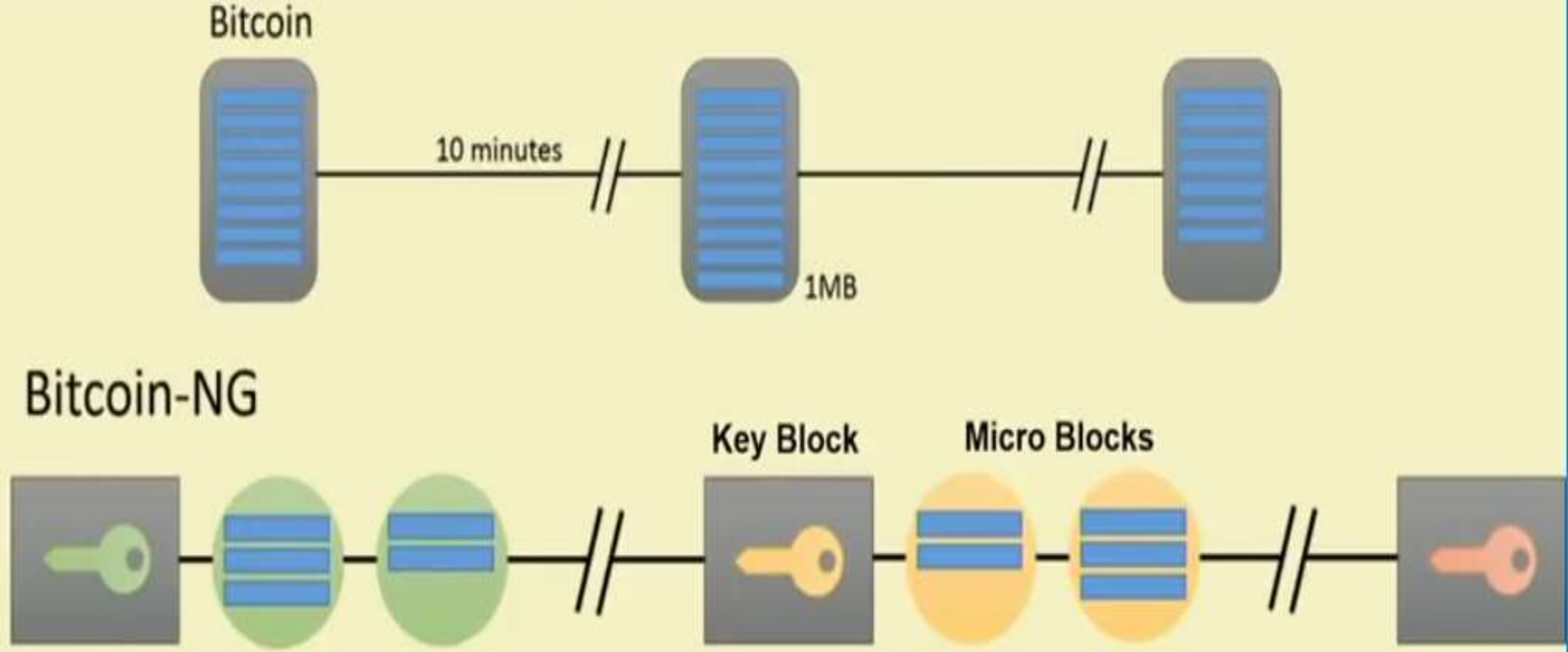
- **Issues with Forks**
  - Prevents consensus finality
  - Makes the system unfair - a miner with poor connectivity has always in a disadvantageous position

Bitcoin-NG: A Scalable PoW Protocol

- 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



Key blocks generated by PoW

Contains key of potential miner

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

Bitcoin-NG: Key Blocks

- For a key block to be valid, the cryptographic hash of its header must be smaller than the target value.

- The key block also contains a public key (so the name, key block)
  - Used in subsequent microblocks

Bitcoin-NG: 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

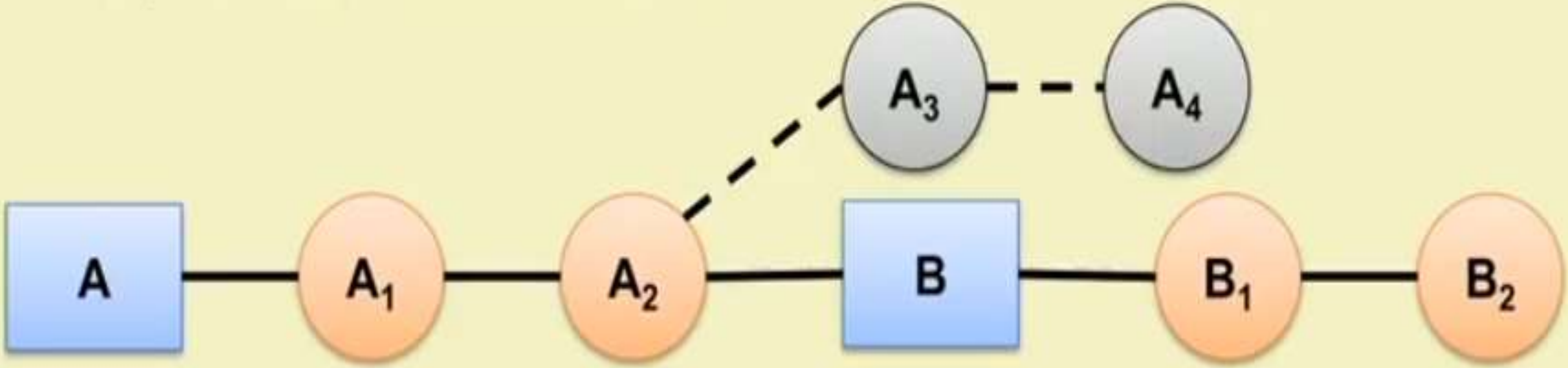
- Once a node generates a key block, it becomes the **leader**
- As a leader, the node is allowed to generate microblocks
  - Microblocks are generates at a set rate smaller than a predefined maximum
  - The rate is much higher than the key block generation rate

Bitcoin-NG: Microblocks

- 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
- The signature uses private key corresponding to the key block public key

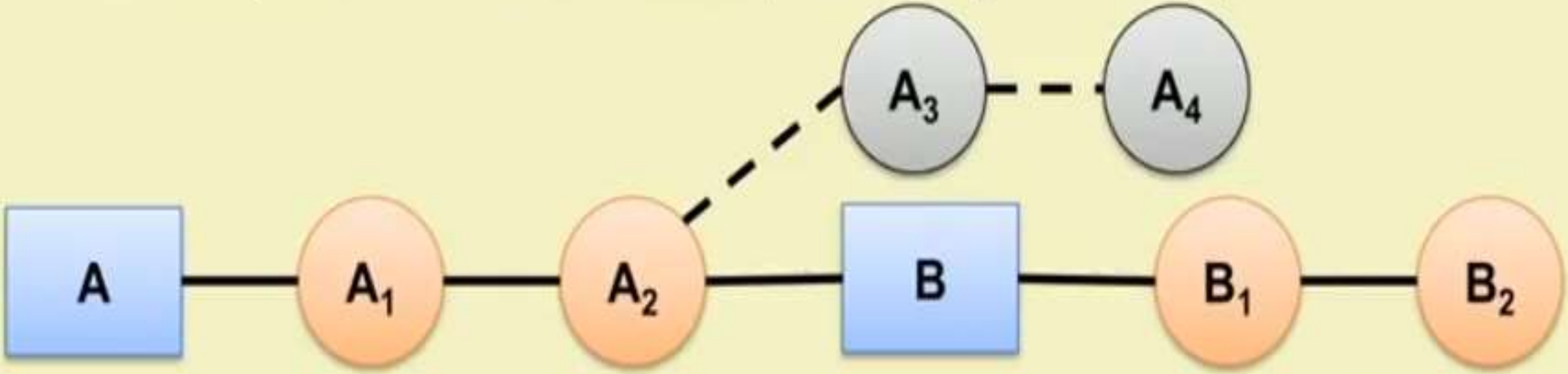
Bitcoin-NG: Confirmation Time

- 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

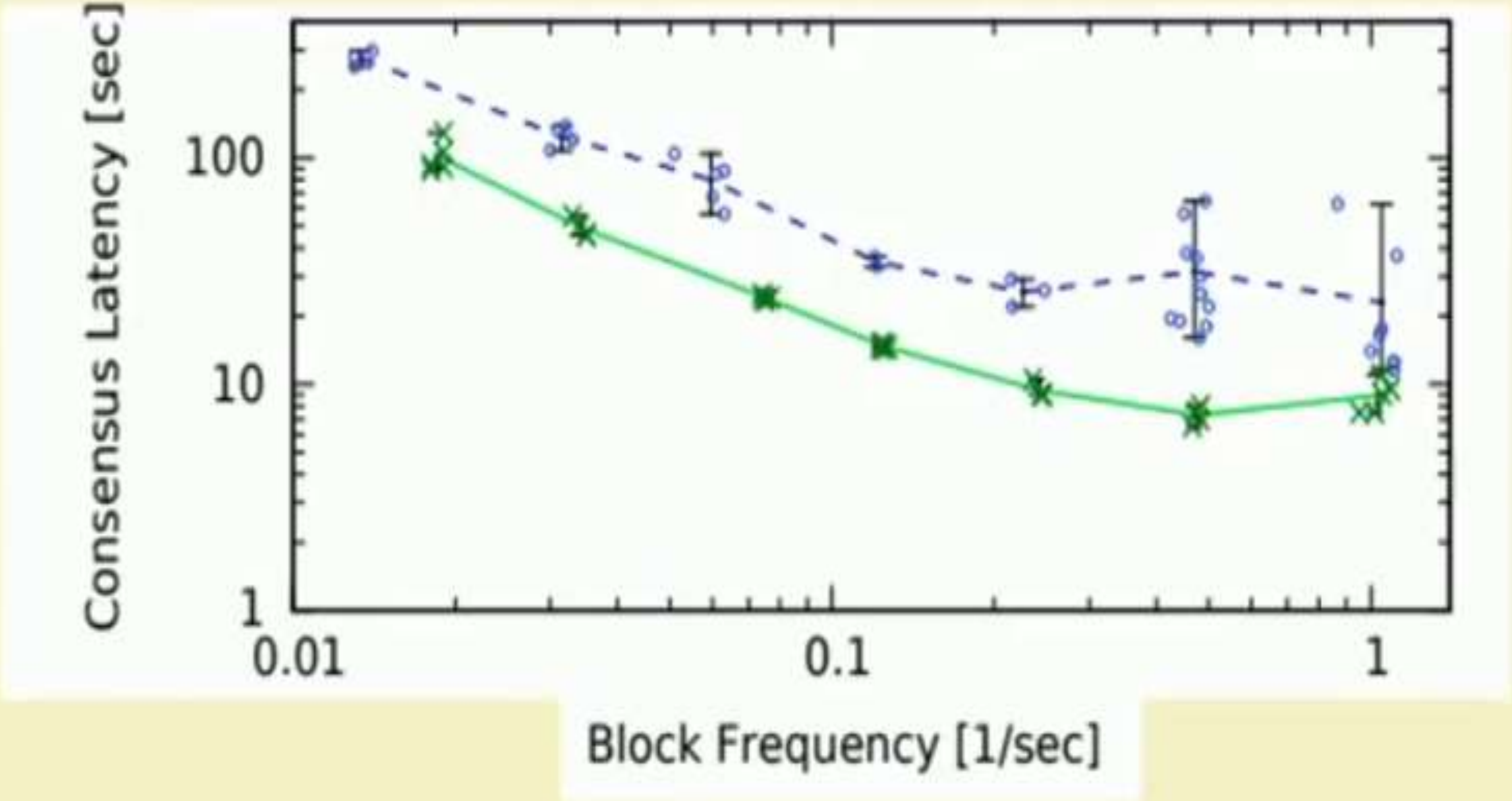


Bitcoin-NG: Confirmation Time

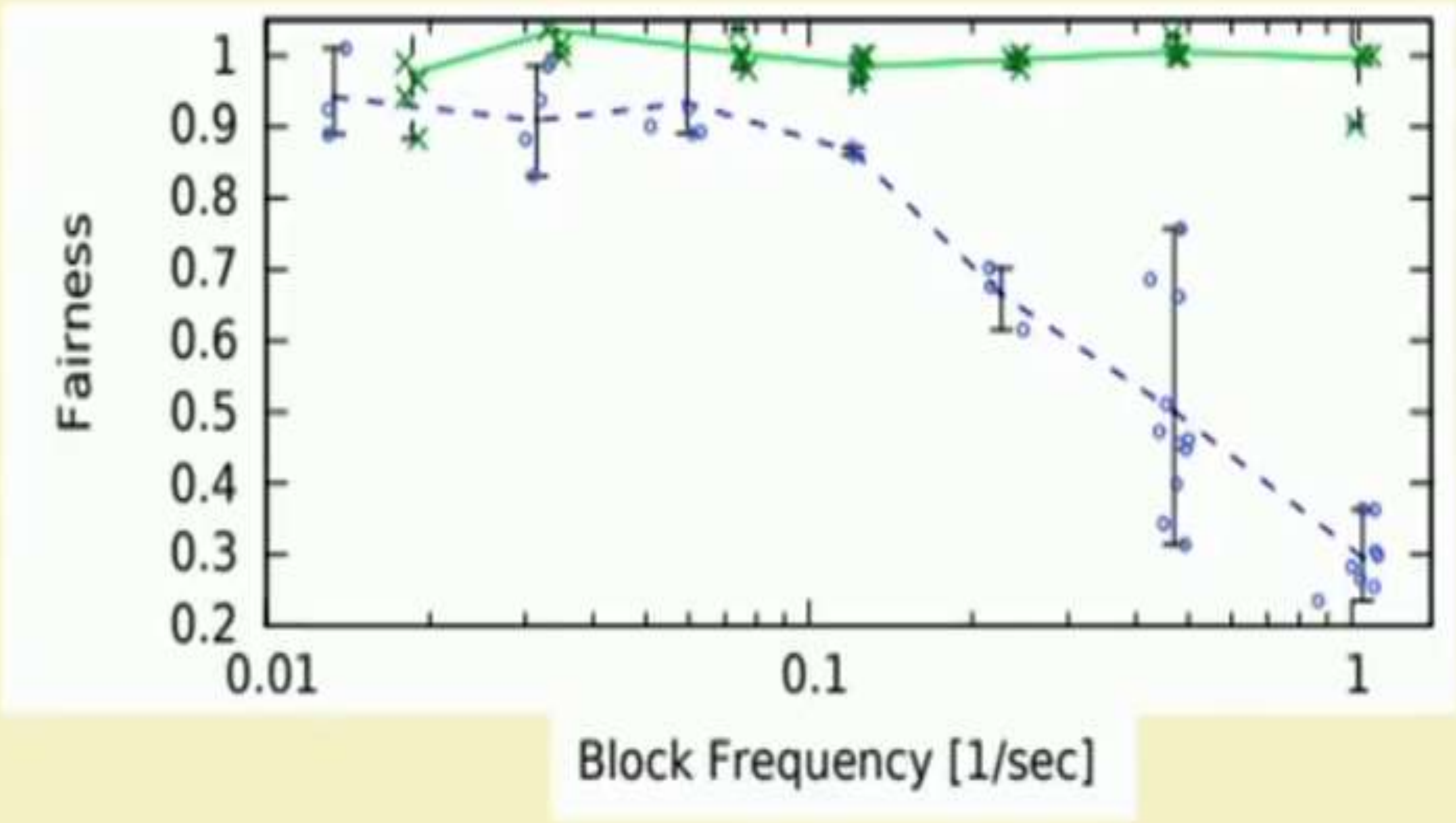
- A node may hear a forked microblock (A_3) but not the new key block (B)
  - This can be prevented by ensuring the reception of the key block
  - When a node sees a microblock, it waits for propagation time of the network, to make sure it is not pruned by a new key block



Bitcoin vs Bitcoin-NG



Bitcoin vs Bitcoin-NG



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**

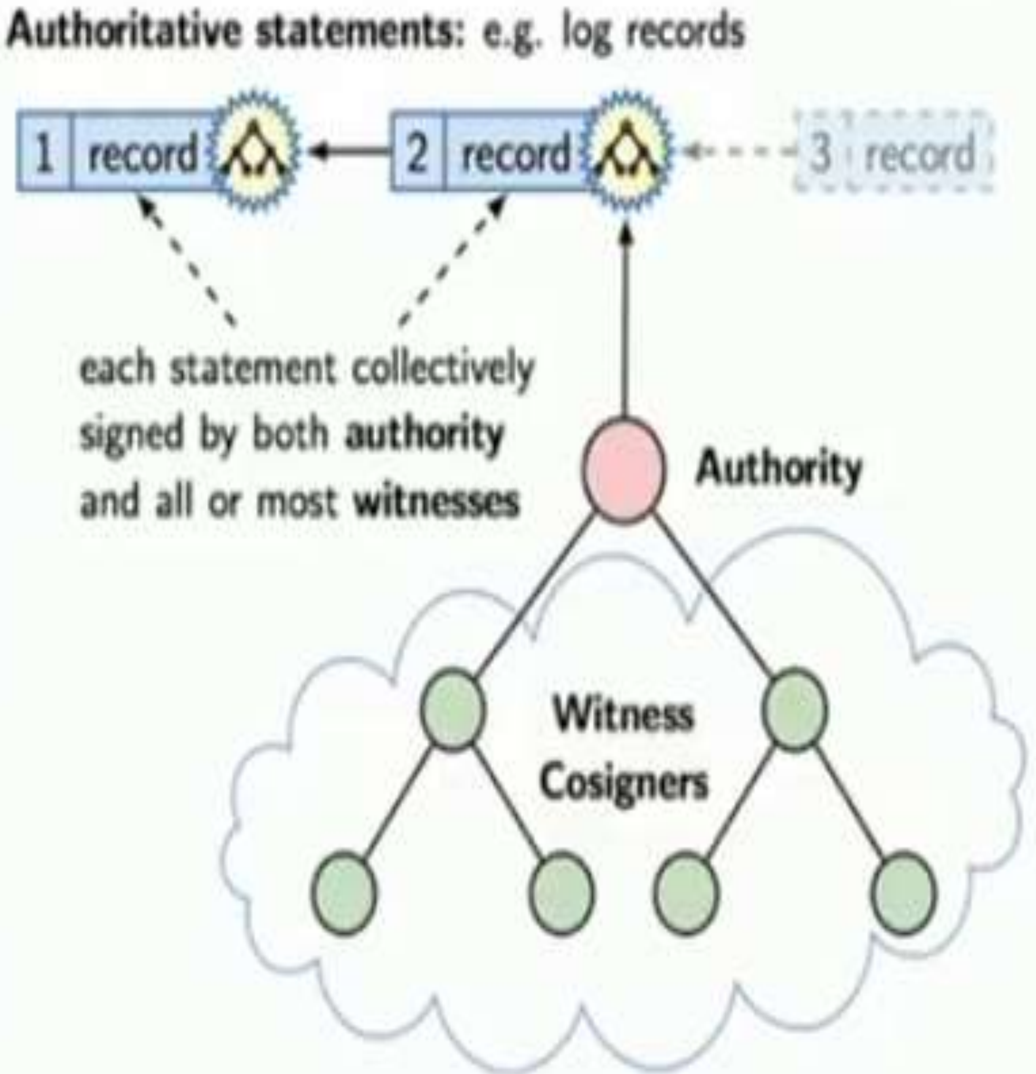
Collective Signing (CoSi)

- 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

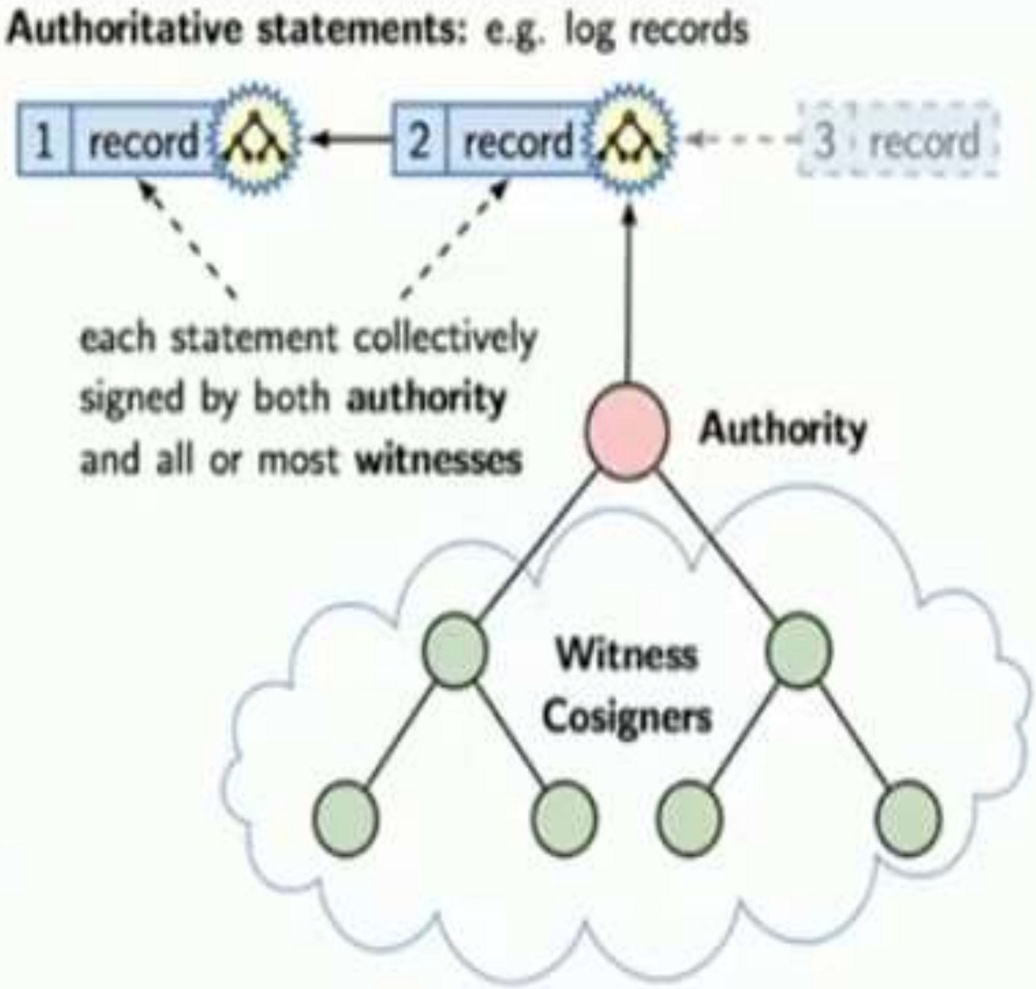
- A statement *S* collectively signed by *W* witnesses assures clients that *S* has been seen, and not immediately found erroneous, by those *W* observers.

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

CoSi Architecture



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

CoSi based on Schnorr Multisignature

- **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 $x_1, x_2, \dots, x_N$, and the corresponding public keys $g^{x_1}, g^{x_2}, \dots, g^{x_N}$

CoSi based on Schnorr Multisignature

- **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).

Xi is the private key. We sign using the private key.

CoSi based on Schnorr Multisignature

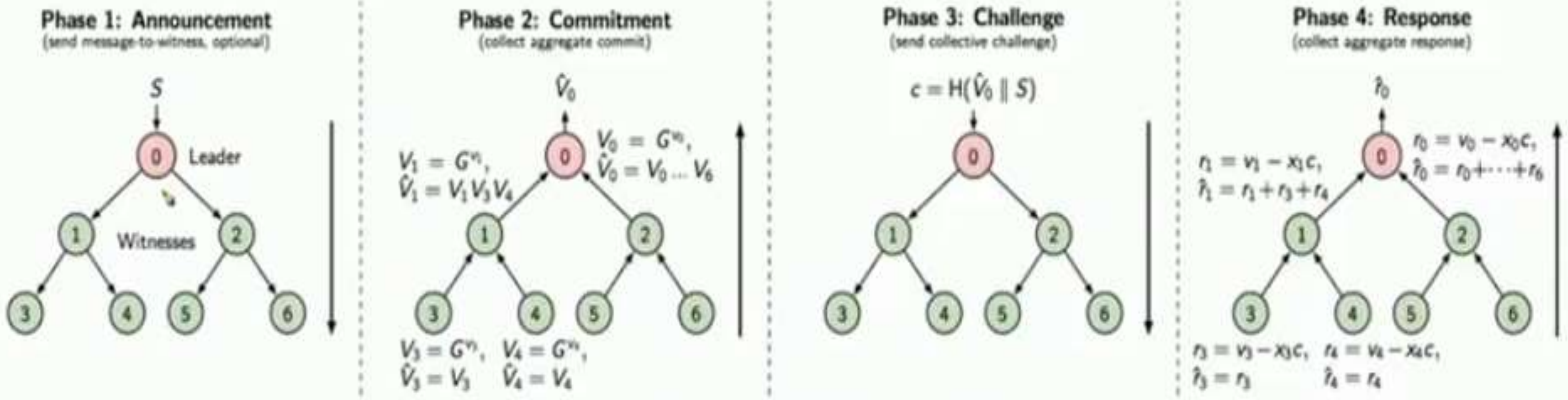
- **Verification:**
  - The verification key is $y = \prod g^{x_i}$
  - The signature is (c, r), where $c = H(V||S)$ and $r = \sum r_i$
  - Let $V_v = g^r y^c$
  - Let $r_v = H(V_v||S)$
  - If $r_v = r$, then the signature is verified

Gxi is the public key. We verify using public key.

CoSi based on Schnorr Multisignature

- **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



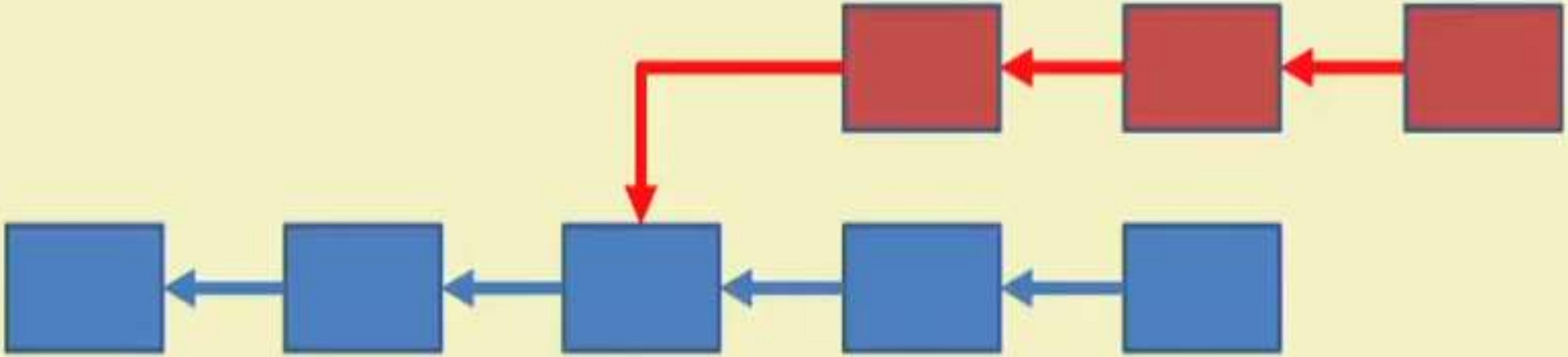
- One CoSi round to implement PBFT's pre-prepare and prepare phases
- Second CoSi round to implement PBFT's commit phase

Problems with Bitcoin

- There is **no verifiable commitment** of the system that a block would exist
  – Probability of successful fork attack decreases as the size of the Blockchain increases

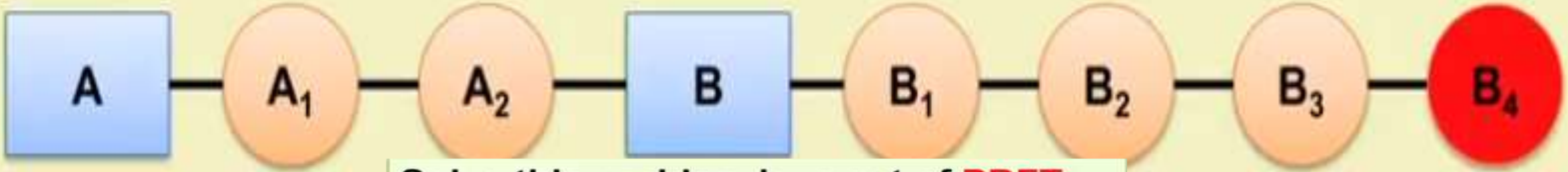
Problems with Bitcoin

- There is **no verifiable commitment** of the system that a block would exist



Problems with Bitcoin-NG

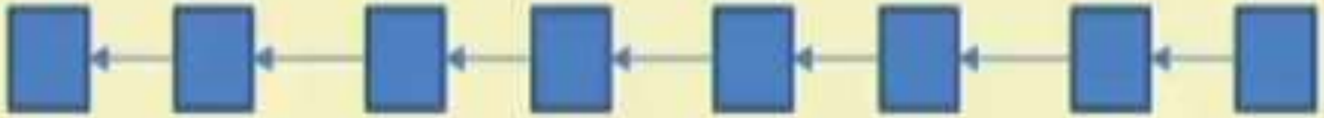
- A faulty key block is verified only after end of the round
  - A faulty miner can introduce a number of correct microblocks following a faulty microblock in the system - certainly a overhead for the application - **a fork alleviates the problem further**



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

PBFTCoin - A Strawman Design

- **Assumption**: 3f+1 fixed "trustees" are there, who will run the PBFT to withstand f failures
  - Avoid the probabilistic strong consistency - introduces low latency in the system
  - No forks in the system - Blocks are added only after verification from the trustees



Problems of PBFT

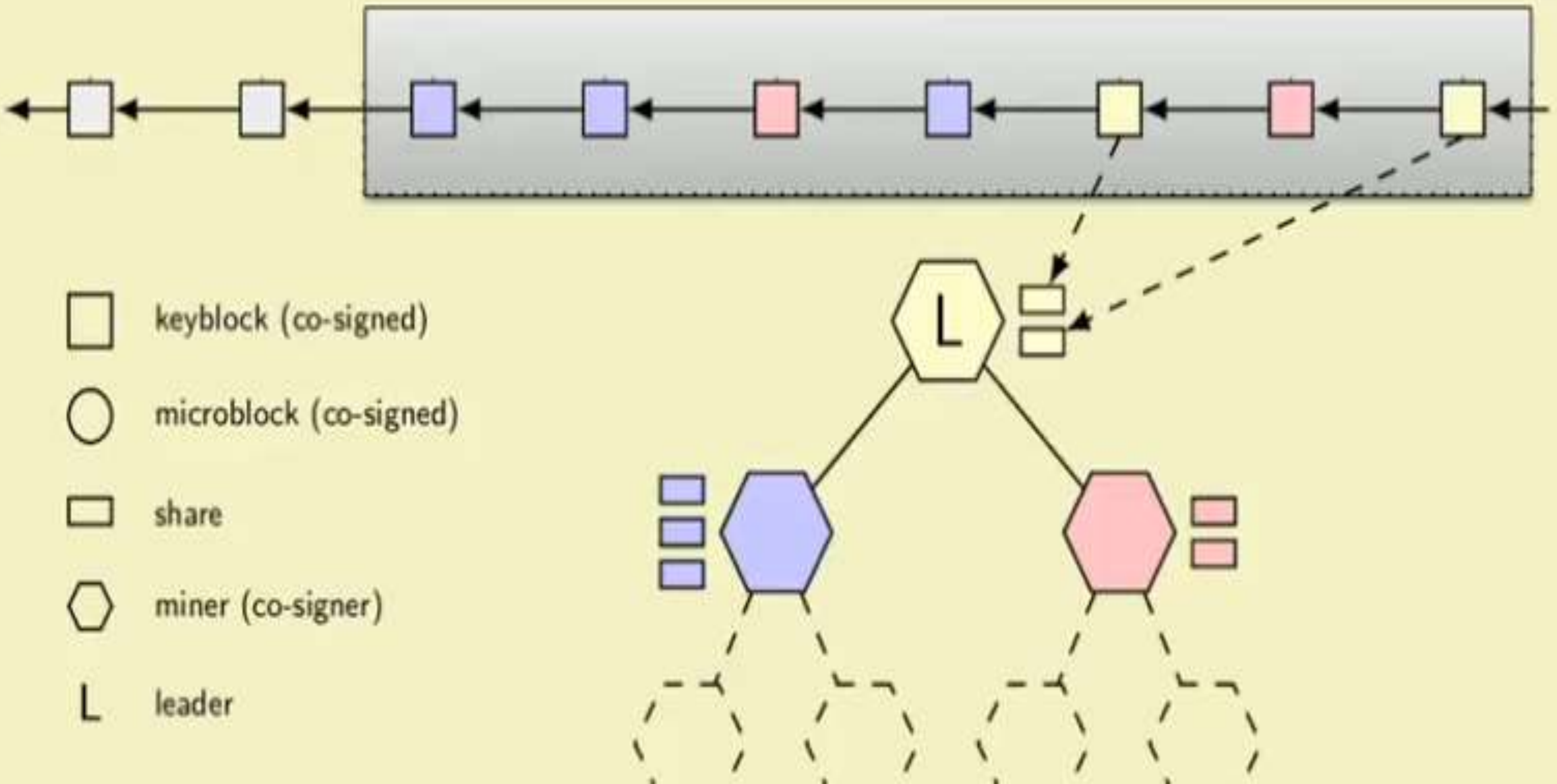
- PBFT requires a **static consensus group** (because of message passing)
- **Scalability** (in terms of nodes) is a problem for PBFT
  - $O(n^2)$ communication complexity
  - $O(n)$ verification complexity
  - Absence of third-party verifiable proofs (PBFT uses MAC - need to share the keys among the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

Open the Consensus Group

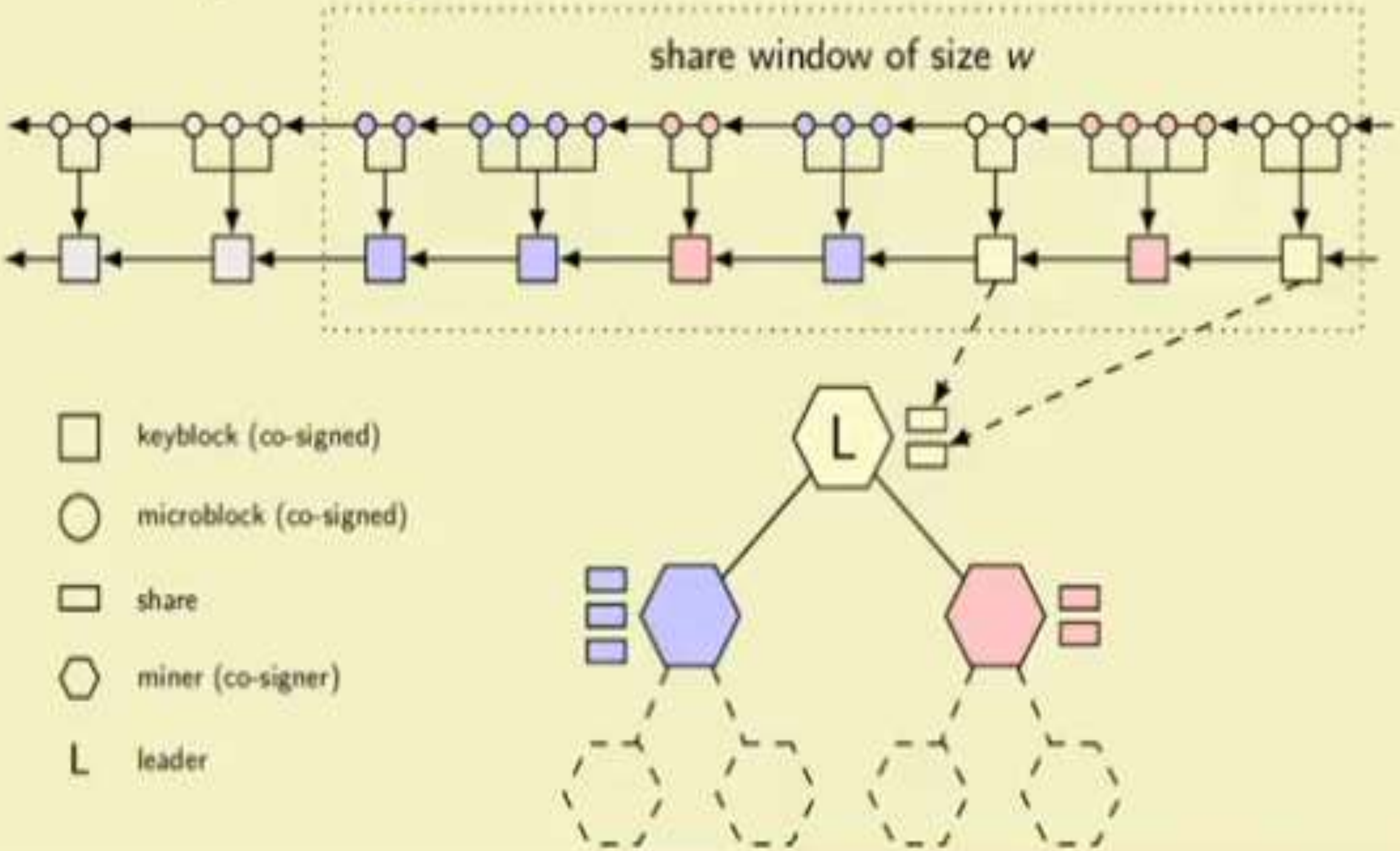


Improve Efficiency

- **Improve O(n) communication complexity**
  - Use tree based multicast protocol - share information with O(log n)
- **Improve O(n) complexity for verification**
  - Use Schnorr multisignatures or BLS for verification
  - Verification can be done in O(1) through signature aggregation
- Multisignatures + Communication trees - **CoSi**

Further Improvement

- Inherit Bitcoin NG's idea of separating out **transaction verification** and **leader election**



ByzCoin Performance

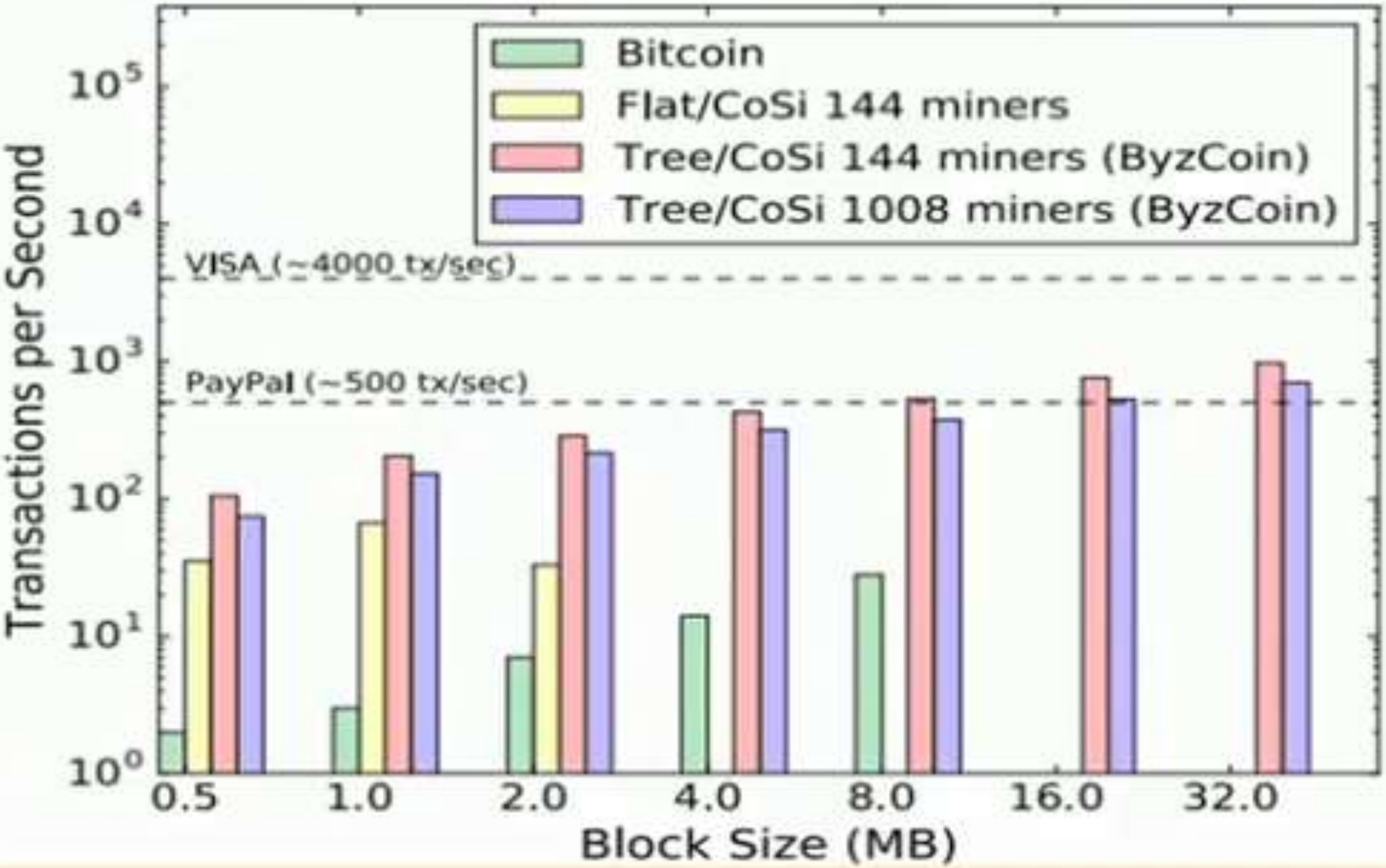
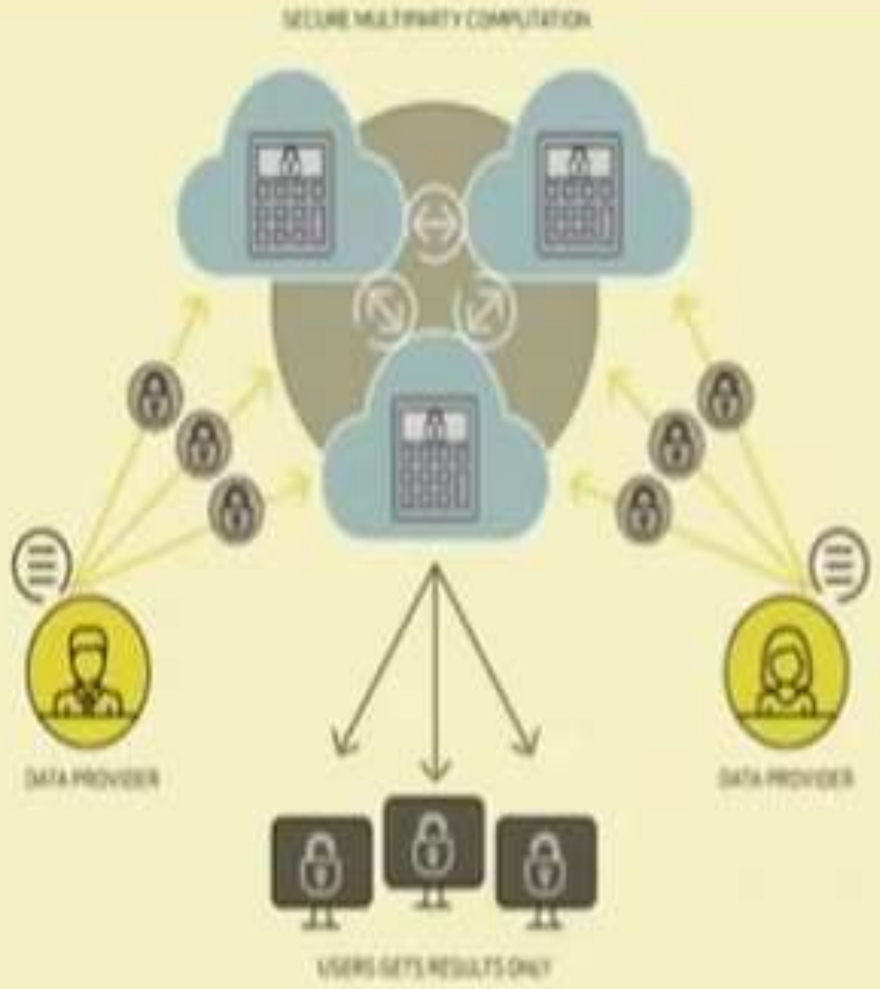
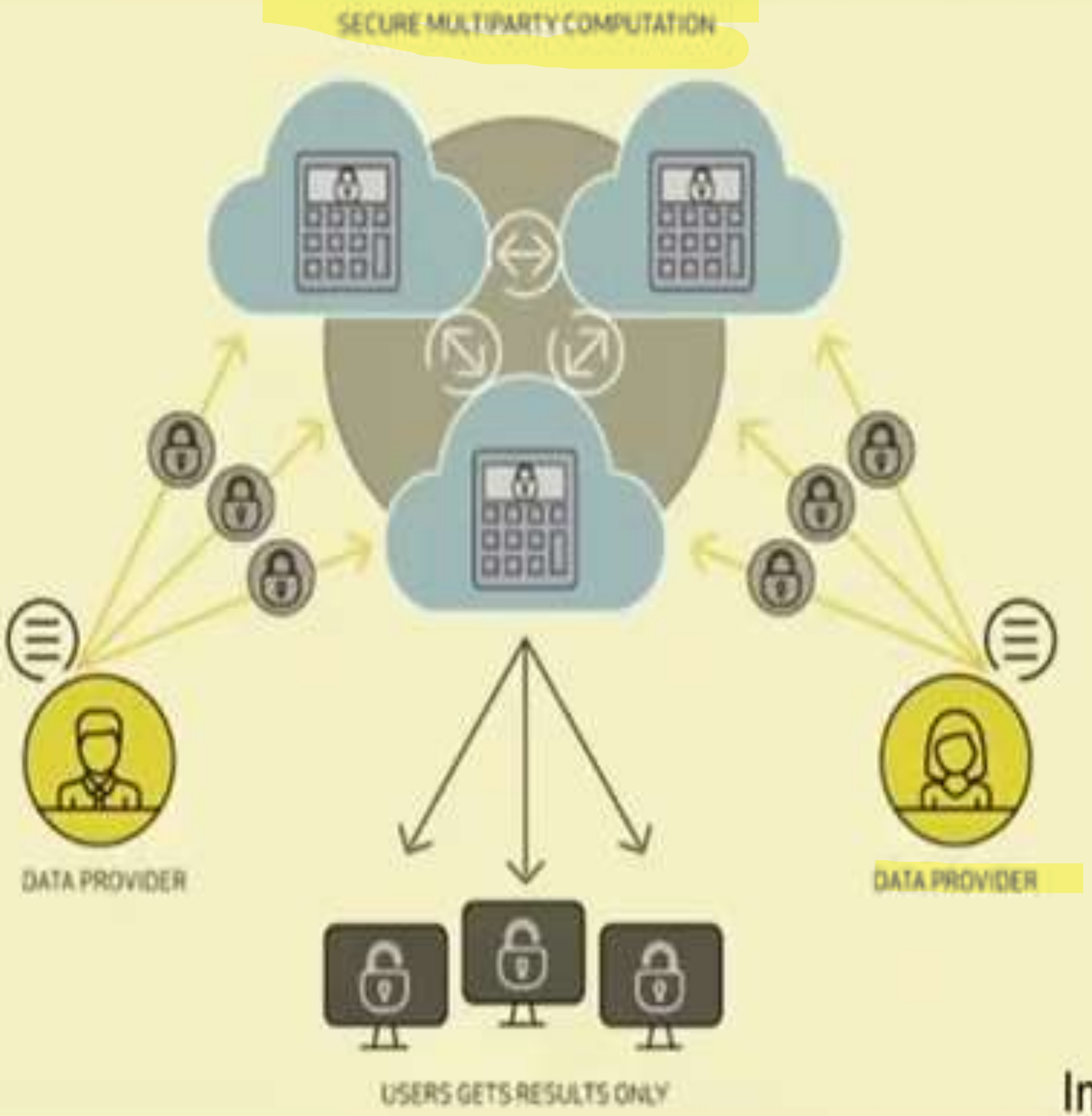


Image source: https://thehub.dk/jobs/company/partisia



Secured Multiparty Computation over Blockchain

Multiparty Computation (MPC)



- Information/data is distributed among multiple authorities with different data share or data distribution policies
- Users want to run a computation - however, the computation involves access to data from multiple sources

User doesn’t want to reveal the private data, but wants to result of certain computation on the data

Image source: https://thehub.dk/jobs/company/partisia

Dining Cryptographer Problem

- Three cryptographers are sitting down to dinner at their favorite restaurant

- Any of the cryptographer can pay the bill, or the bill can be directly paid by the National Security Council (NSC)

Dining Cryptographer Problem

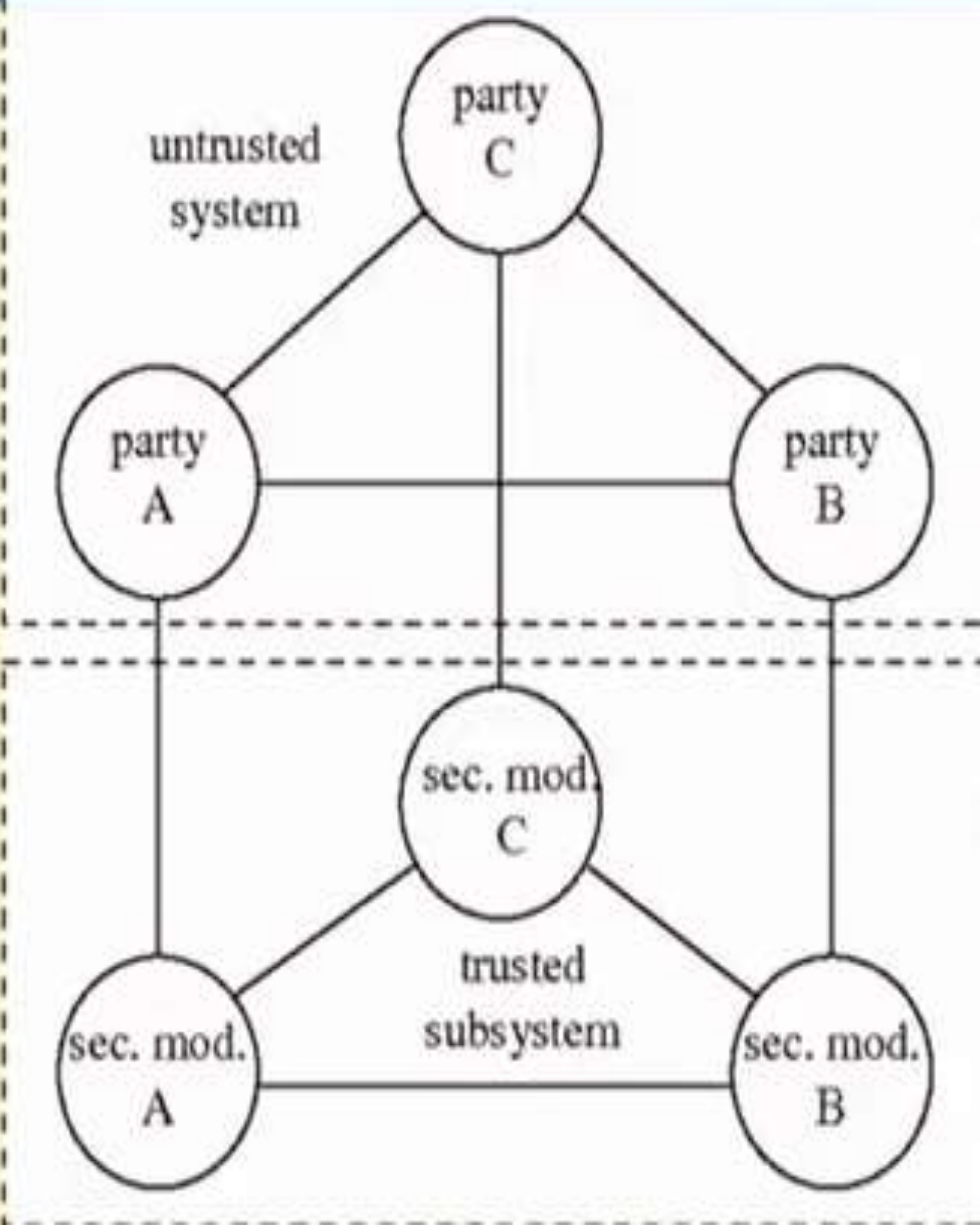
- The three cryptographers respect each other's right to make an anonymous payment
  - But they wonder if NSA is paying

- This payment protocol can be designed using secured

Multiparty computation

Formal Definition

- There are n players $p_1, p_2, \ldots, p_n$
- They wish to evaluate a function $f(x_1, x_2, \ldots, x_n)$
- x_i is a secret value provided by p_i
- **Goal:**
  - Preserve the privacy of the player's input
  - Guarantee the correctness of the computation



Decentralized Solution

- The problem is trivial if we assume the pretense of a trusted third party - **but we do not want to have one**

- Two types of faulty behaviors in a decentralized system
  - Players may try to learn additional information (private computation)
  - Faulty players may try to disrupt the computation (secure computation)

Yao's Millionaire Problem

- Two millionaires wish to find out who is wealthier
- They do not want to reveal any other information

Preconditions

- We know the range of the inputs: (0,N)

- A: Public key **e**, Private key **d**
- B: Can access **e**, not **d**

- $D_d(E_e(X)) = X$
- $D_d(E_e(X)+Y)$ = *some random looking thing if you do not know d*

Protocol Step 1

- A has *i* and B has *j*
- B generates a random *x* of *m* bits
- $C=E_e(X)$
- $u=C-(j-1)$
- Send *u* to *A*

Protocol Step 2

- A Computes: *for* $(t = 1 \text{ to } N)\ y_m = D_d(u+t)$

- A takes a prime p of size $sqrt(m)$ and computes
  - $z_i = y_i \bmod p \quad \text{for } i = 1 \text{ to } N$

- p is chosen such that $|z_m - z_n| >= 2$ for any m,n in [1 to N]

Protocol Step 3

- A sends B the following list
  - $p, z_1, z_2, \ldots, z_i, (z_{i+1}+1), (z_{i+2}+1), \ldots, (z_N+1)$

- B compares the j^{th} entry of this list excluding prime p with $(x \bmod p)$

- If $(x \bmod p) = j^{th}$ entry of the list, then $i >= j$

Problems with MPC

- MPC has poor scaling properties
  - Performance in the malicious setting is worse than the semi-honest case
- Depends on the assumption that majority of the parties are always honest and share the correct information
  - How will you ensure that the parties have shared the correct information?
  - The parties can deny the sharing of information as well

Fair MPC

- Either all parties receive the protocol output or no party does
  - Extremely important for applications like auctions or contract signing

- Example:
  - Alice participates in an auction
  - She **learns first** that she did not win the auction
  - She aborts and claims a network failure - tries again with a new bid

Fair MPC

- [Cleve '86] Fair MPC is impossible to realize for general functions when a majority of the parties are dishonest.
  - Also holds when the parties have access to a trusted setup, such as a common reference string

Richard Cleve. Limits on the security of coin flips when half the processors are faulty (extended abstract). In STOC, pages 364–369, 1986

Solve Fair MPC - Use a Public Bulletin Board

- Parties have access to a *public ledger*
  - Allows anyone to publish arbitrary strings - used for MPC protocol
  - The strings contain proof about who has published the string - anyone can verify

- Run an unfair MPC protocol to compute an encryption of the function output - design a fair decryption protocol using the public ledger - either everyone can decrypt or no one can