Blockchain Principles and Applications

Amir Mahdi Sadeghzadeh, PhD

Data and Network Security Lab (DNSL)
Trustworthy and Secure AI Lab (TSAIL)

Most slides have been adapted from ECE/COS 470, Pramod Viswanath, Princeton 2024 and CS251, Dan Boneh, Stanford 2023.

Recap

Bitcoin latency

Time from when a transaction was broadcast until the transaction is confirmed in the ledger

- τ_1: Time from when a transaction was broadcast until the transaction is put into a mined block B
- τ_2: Time from when the transaction was put into a mined block B until block B is k-deep in the longest chain

$$\tau = \tau_1 + \tau_2$$

τ_2 is the real bottleneck, depends on how large k is.

Bitcoin latency

Assume low forking ($\lambda\Delta \ll 1$),

$$\tau = \frac{k}{(1-\beta)\lambda}$$

Depth of blocks

Block arrival rate

From Lecture 6, error probability

$$\epsilon = e^{-ck}$$

$$\tau = \frac{\frac{1}{c}\log(\frac{1}{\epsilon})}{(1-\beta)\lambda} = O(\frac{1}{\lambda}\log(\frac{1}{\epsilon}))$$

Latency and security are coupled

Improve Bitcoin latency

Only way to improve latency is to

- reduce k; but this reduces security
- Increase λ; but this also reduces security

Ethereum: $\frac{1}{\lambda}$ = 15s; $k = 100$

- latency = 25 minutes
- Way better than Bitcoin performance; improvement simply by picking better parameters.

Improve Bitcoin latency

Question: can we make relatively small changes to the longest chain protocol and PoW mining while scaling latency?

Key Requirement:

- Do not want latency to depend on security level
- **Decouple security from latency**

Prism

Prism achieves optimal latency

- **Decoupling principle**: separate performance from security

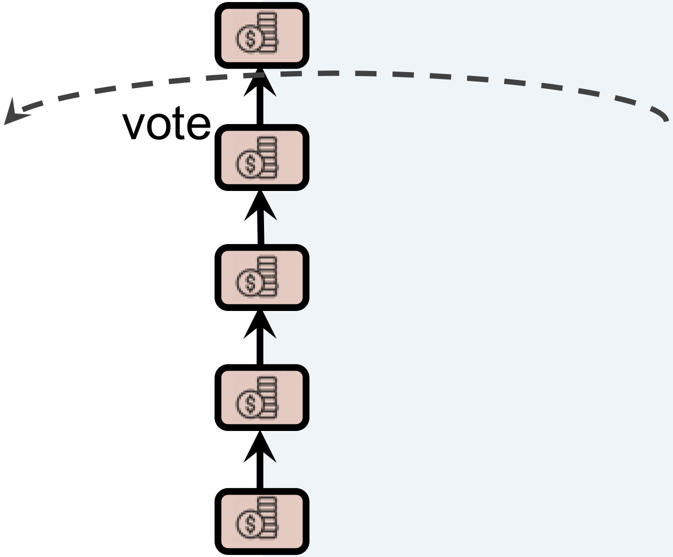
- Prism 1.0 achieves optimal throughput; last lecture

Bitcoin → Deconstruct

Proposing

Voting

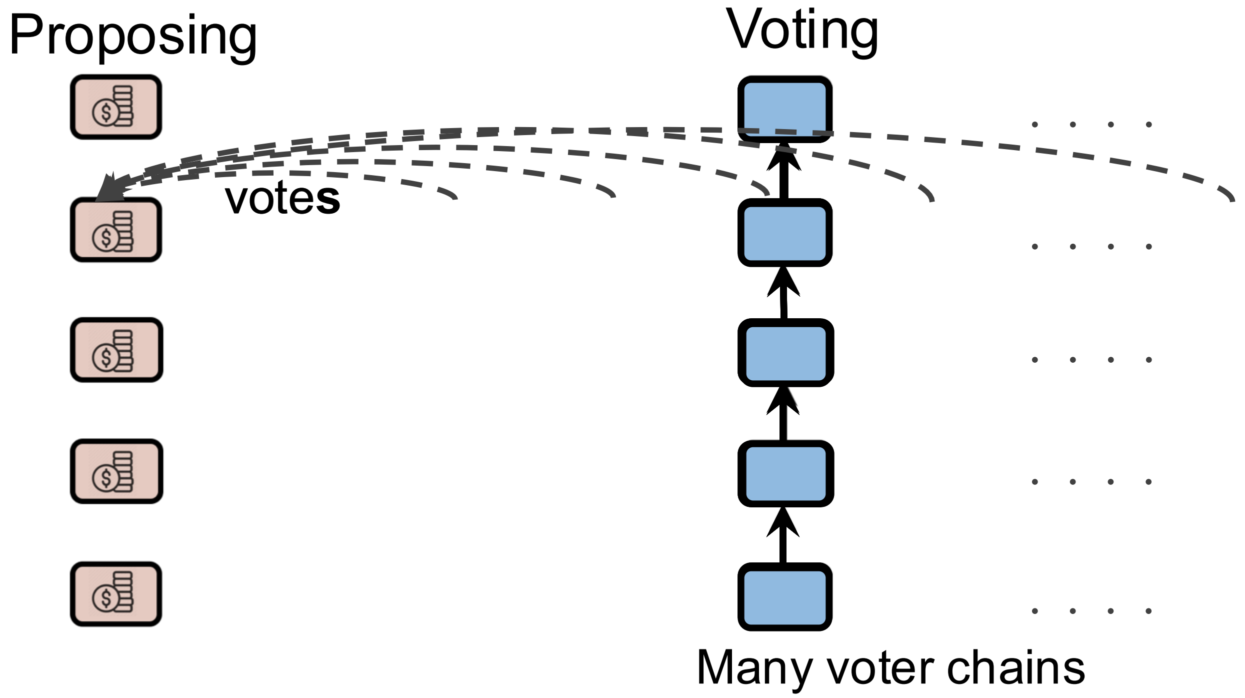
Ledger construction



1. Select votes along longest voter chain
2. Order the proposer blocks by votes

Bitcoin → Deconstruct →Prism

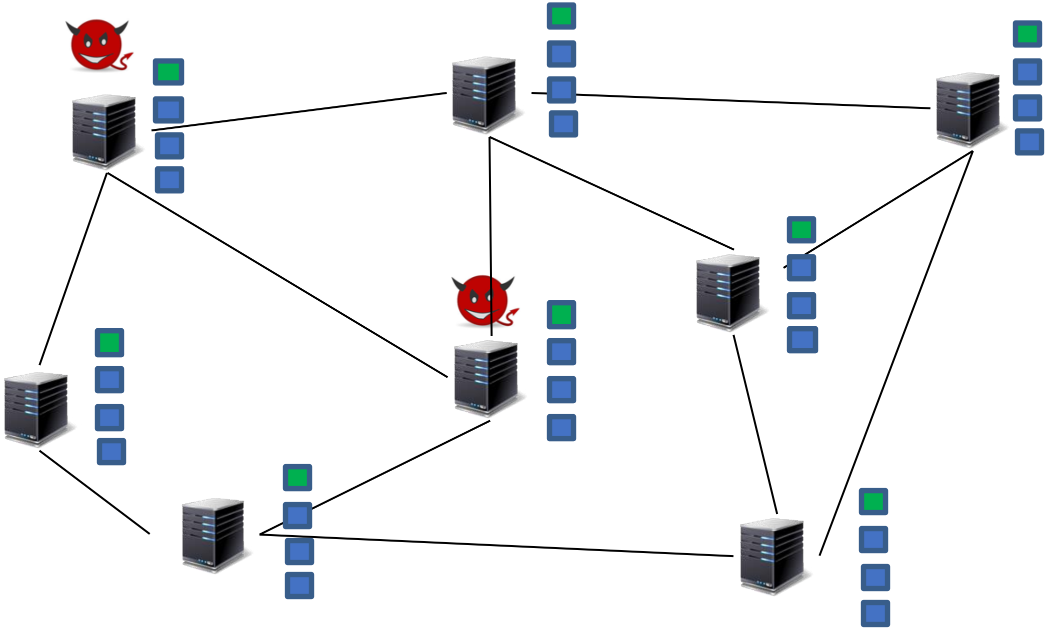
Fast Confirmation

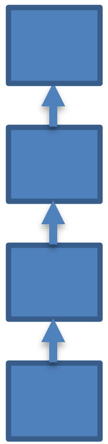


Ledger Construction: For each level choose the proposer block with **maximum** votes

Sharding

Blockchains & Full replication





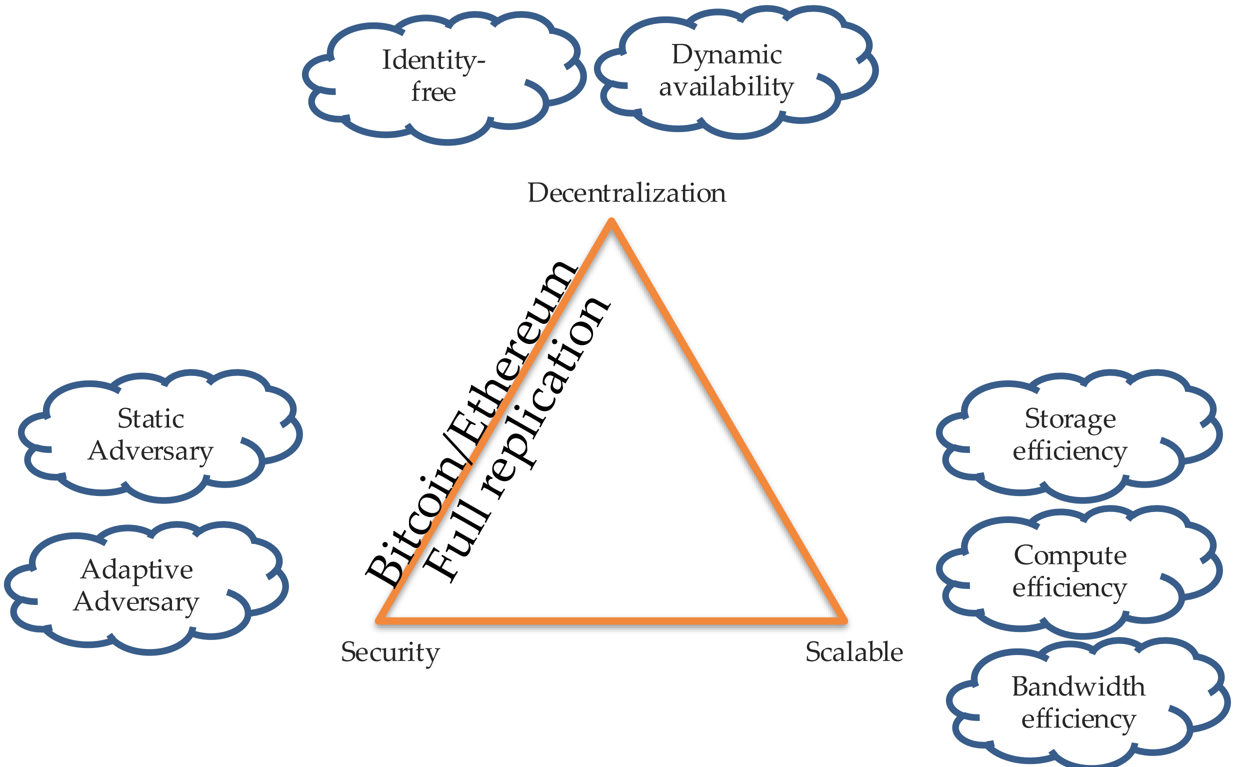
Full replication – Consensus problems

- Throughput of consensus decreases with an increase in size of the number of participating nodes
- For Nakamoto delay $\propto O(\log(N))$
- For Nakamoto, communication load = $O(N^2)$

Full replication – resource usage

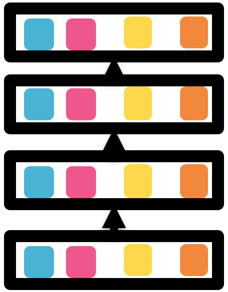
- All nodes process the same transactions
- **Communication**: The transaction has to traverse the complete network at least once
- **Storage**: All nodes have to store the complete state, the account details of everyone!
- **Compute**: All nodes have to validate all transactions and update the ledger every block

Trilemma

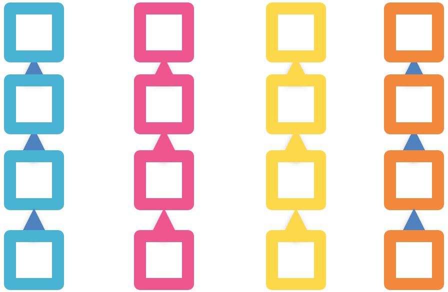

Identity-
free
Dynamic
availability
Decentralization
Bitcoin/Ethereum
Full replication
Static
Adversary
Adaptive
Adversary
Storage
efficiency
Compute
efficiency
Bandwidth
efficiency
Security
Scalable

First Approach– Maintain multiple blockchains

- Divide ledger into K shards
- Each shard is a separate blockchain



Complete ledger



K Shards

Problems

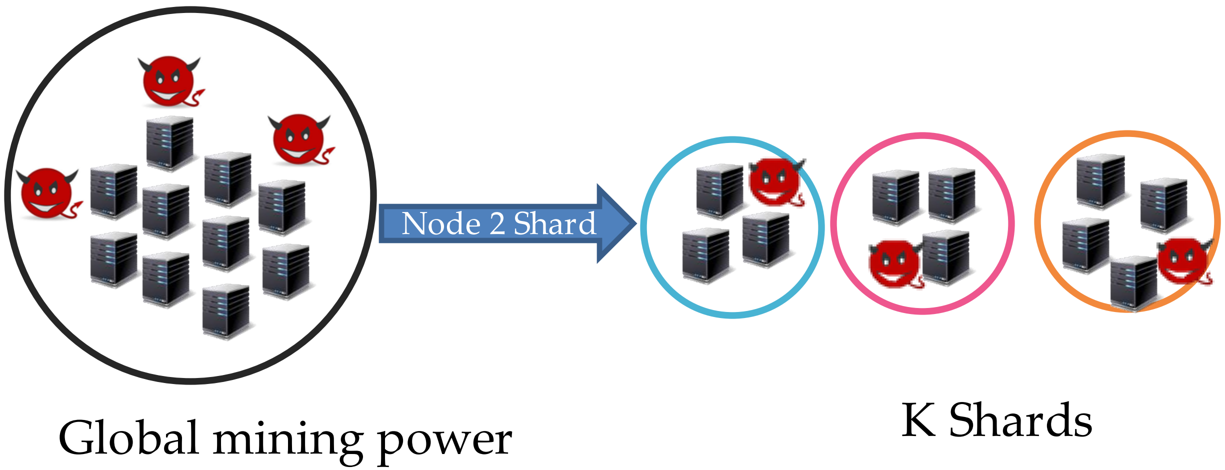
1. Reduced security – An adversary can concentrate on one shard

2. How to transfer money from one shard to another?

3. If such transfer is possible, adversary can transfer non-existent funds from infected shard to non-infected shard.

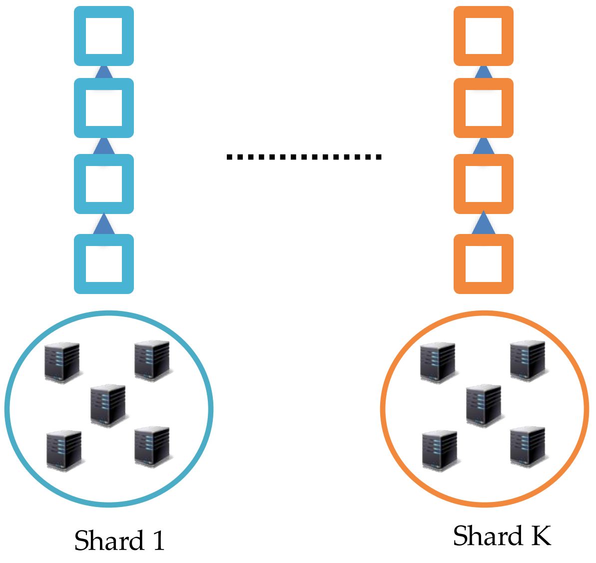
Node to shard allocation (N2S)

- Extension of the first order approach
- Allocates each consensus nodes to one shard randomly



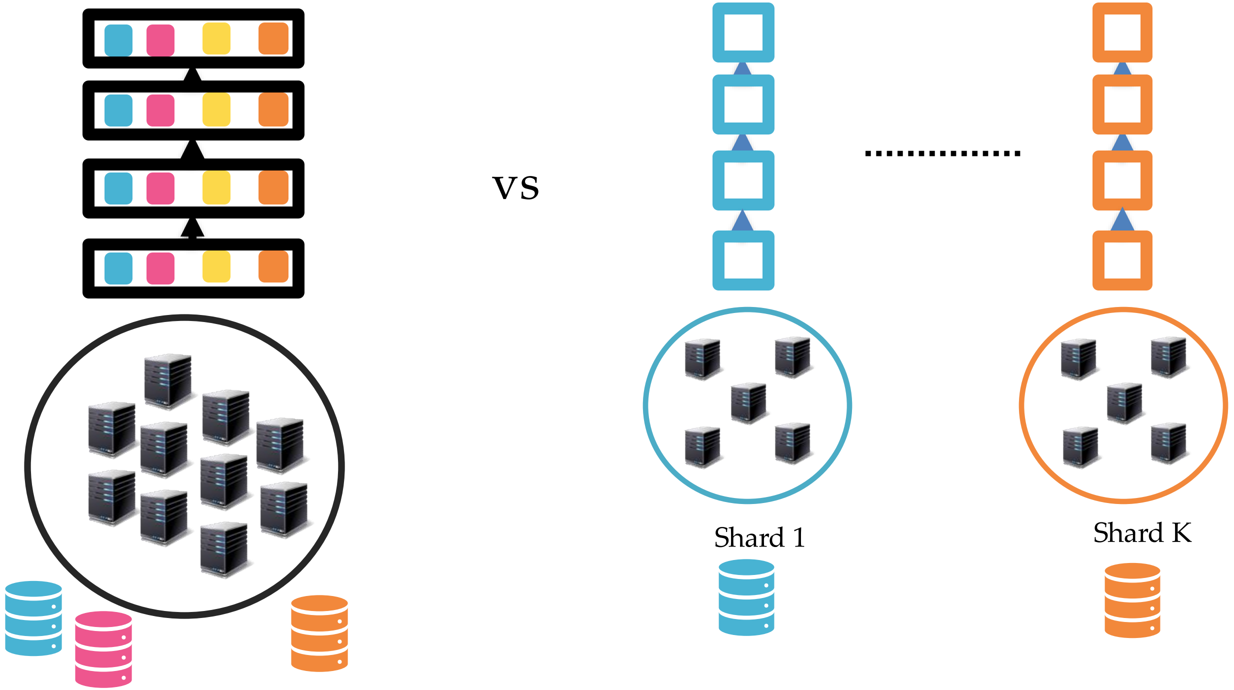
Multiconsensus

- Each shard runs its own consensus



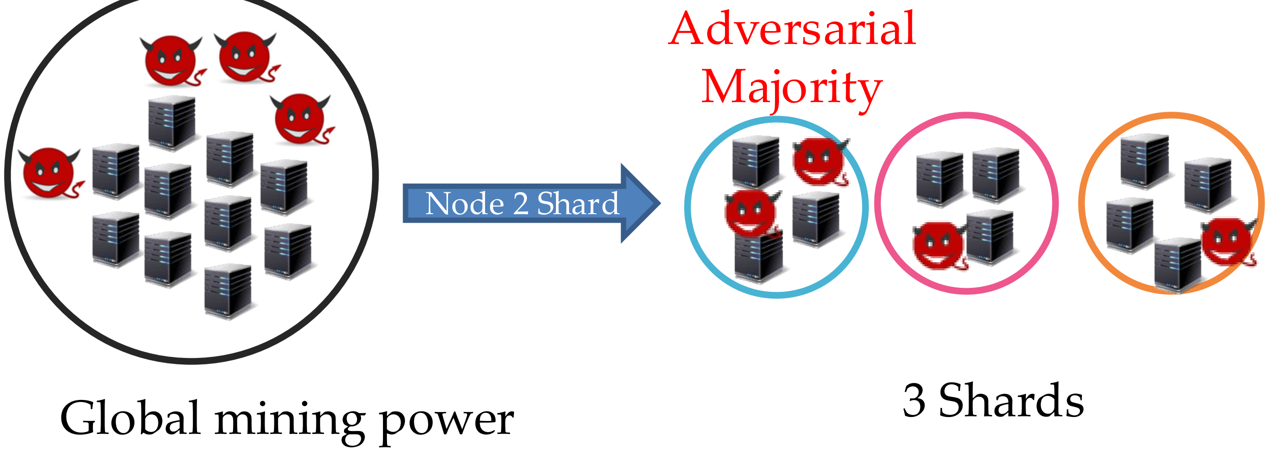
Scaling: O(K)

- Nodes only maintain the state of their own shard



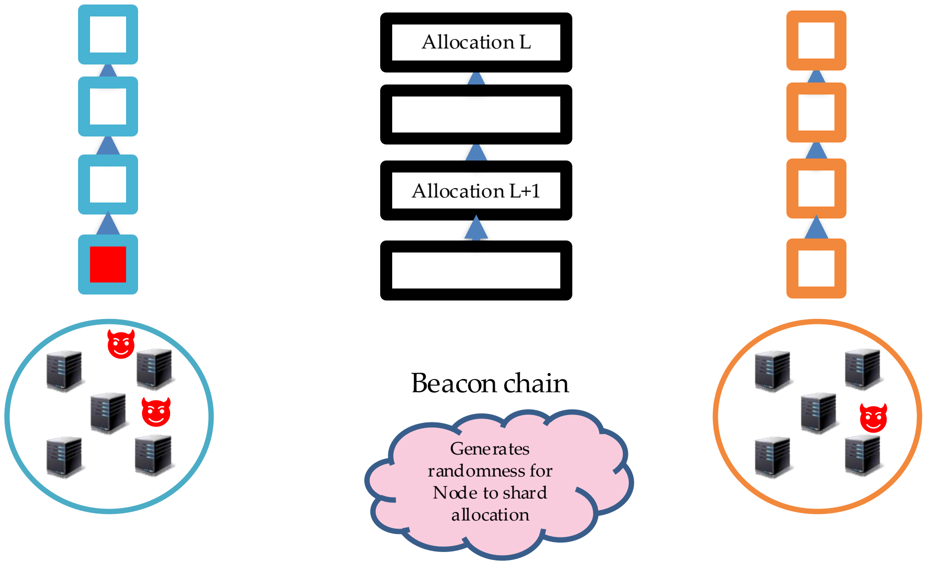
Drawback 1: Proportional representation

- Need large number of nodes per shard to ensure honest majority in a shard



Drawback 2: Security

- Not resilient to O(1/K) adaptive adversary

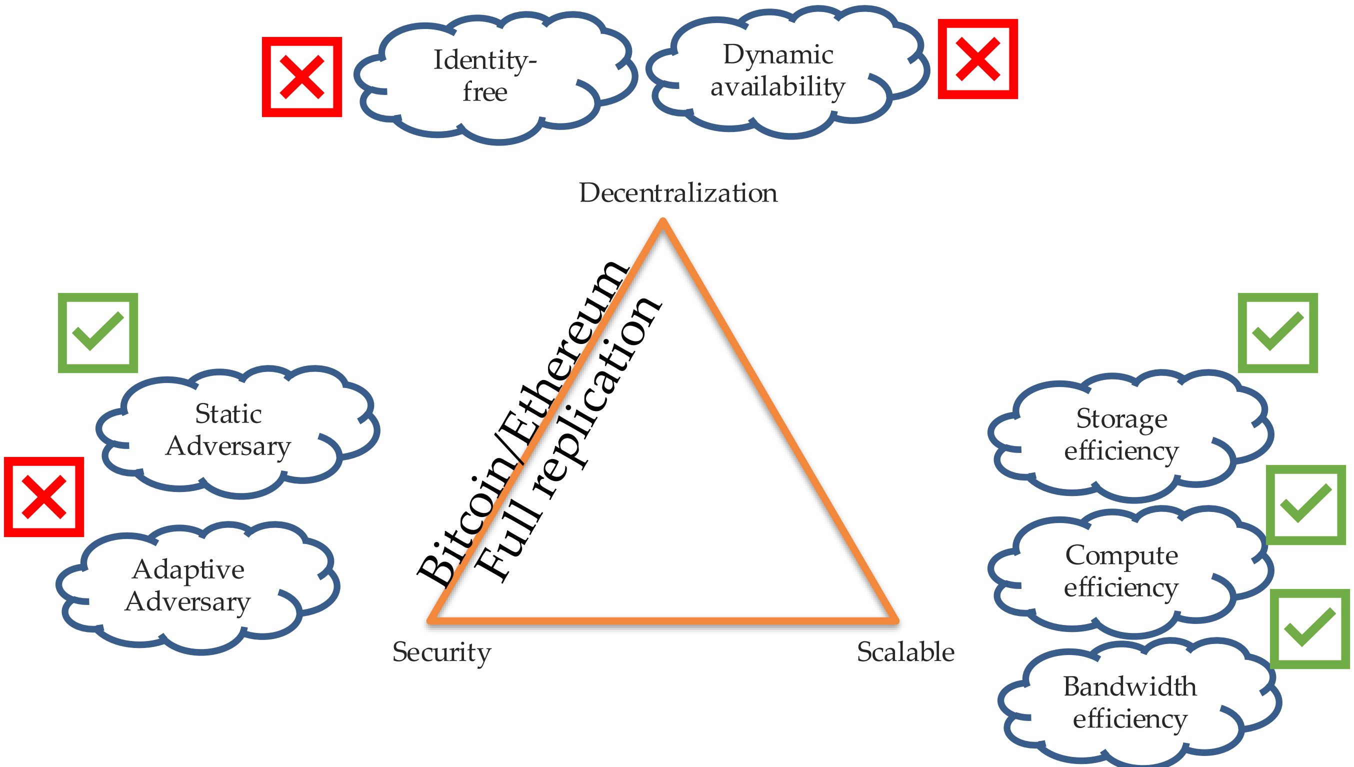


Drawback 3: Node identity

- Existing works vary in the implementation of Node to shard allocation
  - Different ways of randomness generation
  - Different rate of re-allocation

- All Node to shard allocation algorithms require consensus node identity

Trilemma revisited

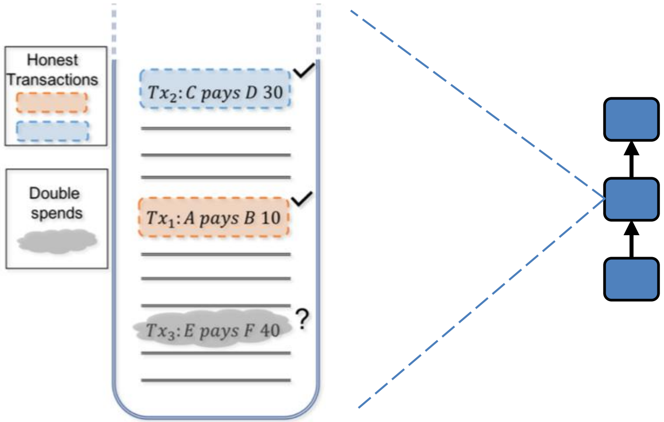


Identity-free sharding

- If the protocol is identity free, we cannot use Node to Shard allocation algorithm
- Only choice is to allow nodes to **self-allocate**
- Adversaries can congregate on one shard; safety and liveness can be easily broken
- Solution: Uniconsensus architecture

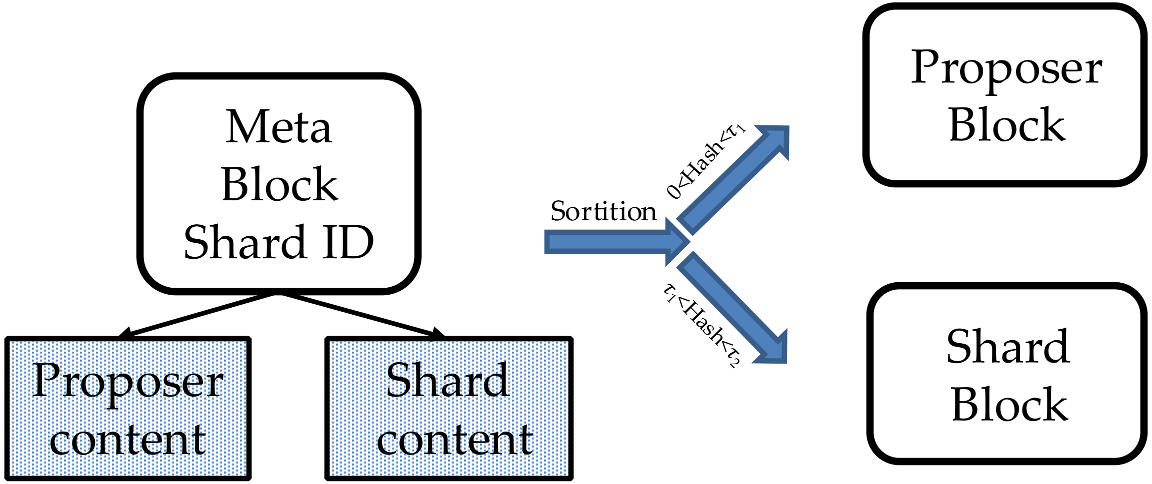
Uniconsensus: Decoupled validation

- Shard transaction ordering can be decoupled from validation
- Extension of the deconstruction ideas from Fruitchains and Prism

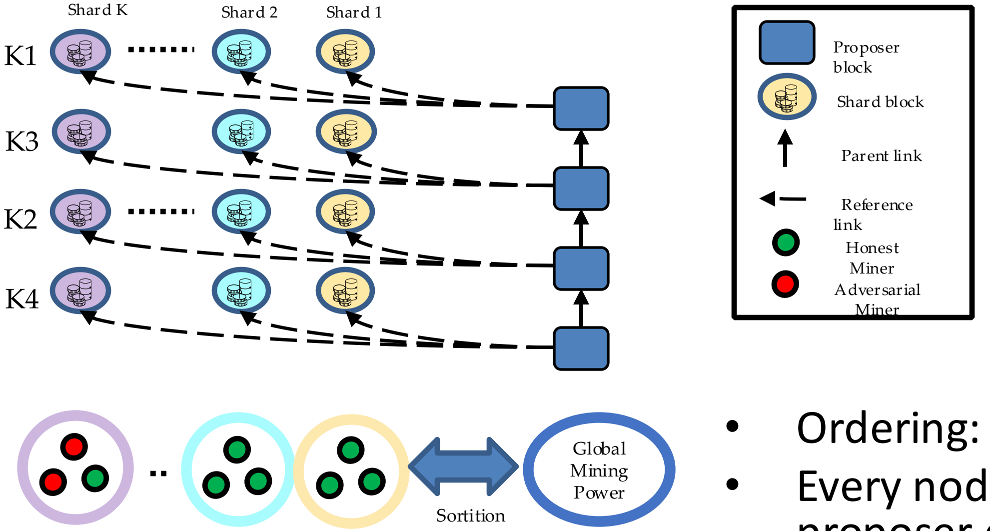


Uniconsensus: Sortition

- A node can maintain any shard of its choice
- It will maintain an ordering chain in parallel
- All nodes mine shard block and proposer block together

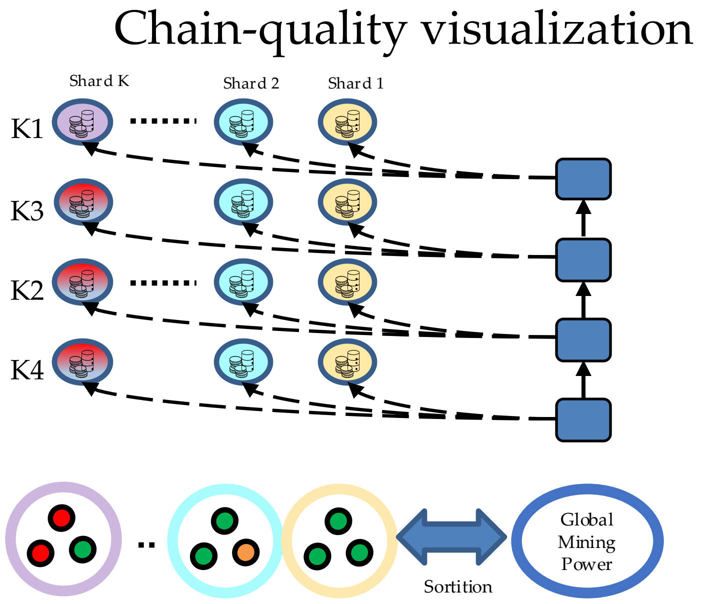


Uniconsensus: Safety



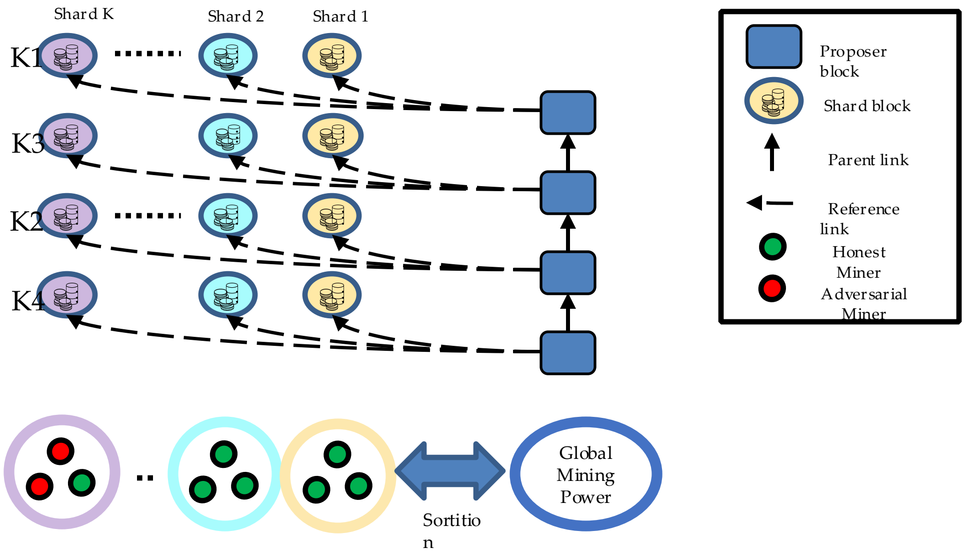
- Ordering: Proposer chain
- Every node maintains proposer chain
- Adversarial majority in a shard does not violate safety

Uniconsensus: Liveness



- Adversaries congregate: drown out honest miners
- Dynamic self allocation can prevent such attacks

Uniconsensus Architecture



- Safety shared
- Liveness is sacrificed

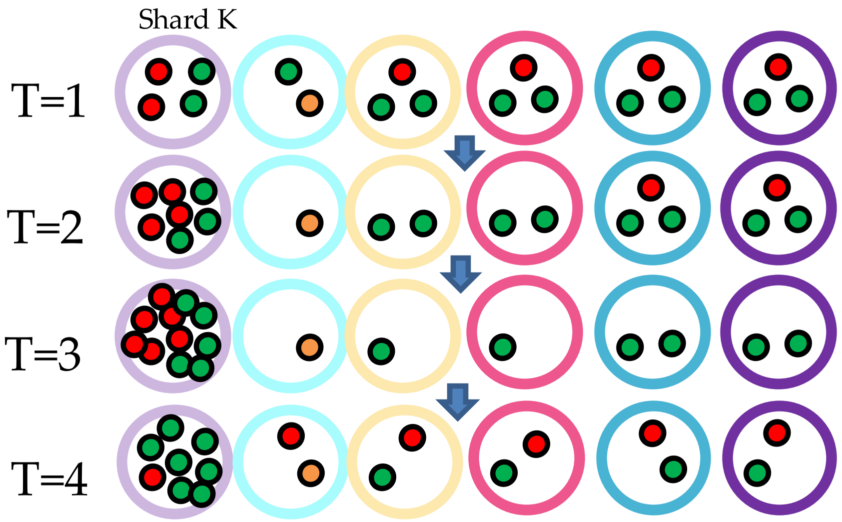
Dynamic Self-allocation

- Honest nodes adapt to adversaries and allocate themselves to new shards

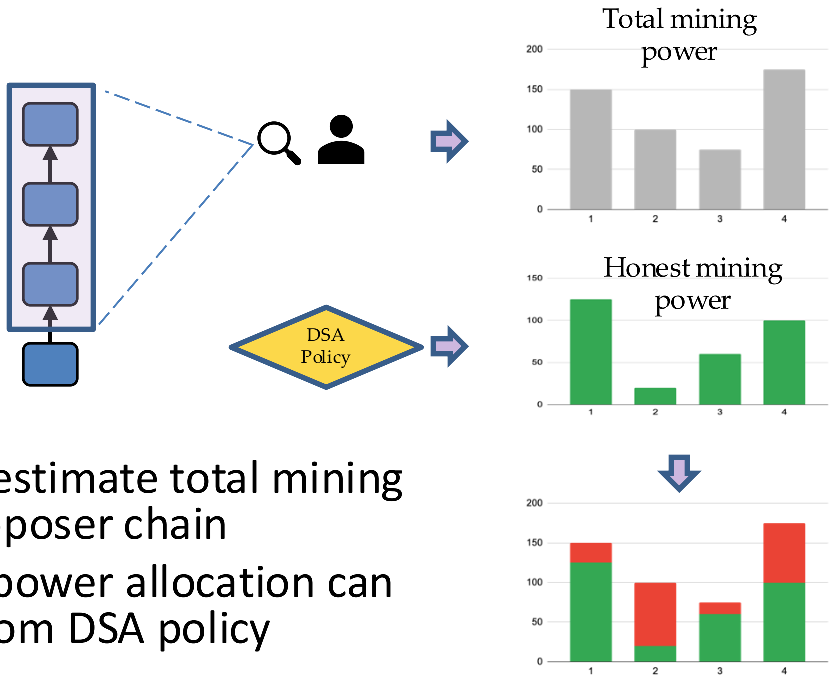
- We want honest nodes to (re)allocate themselves to shards under attack

Simple DSA

- Adapt to the adversary by following the adversary's last move
- Assume that the adversary's past allocations are known



Practical Implementation



- Honest miners estimate total mining power from proposer chain
- Honest mining power allocation can be estimated from DSA policy

Resources

- ECE/COS 470, Pramod Viswanath, Princeton 2024