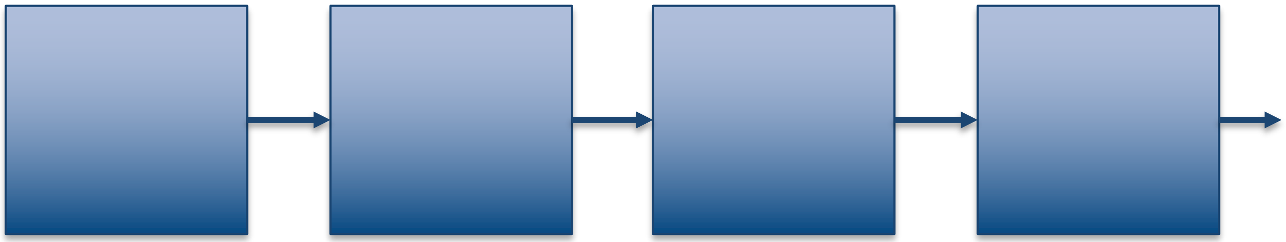
Blockchain

Davide Frey
WIDE Team
Inria

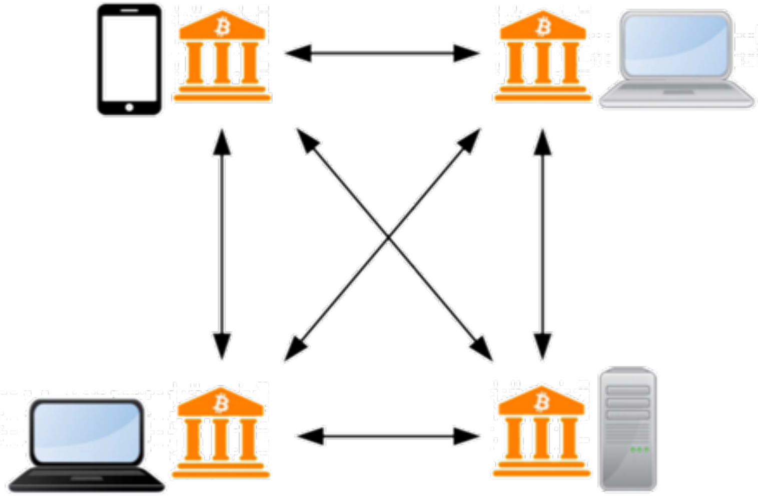
What is a Blockchain

- Distributed Ledger
- Recording Transactions
- Replicated
- Need agreement on the content of the ledger



Bitcoin Cryptocurrency

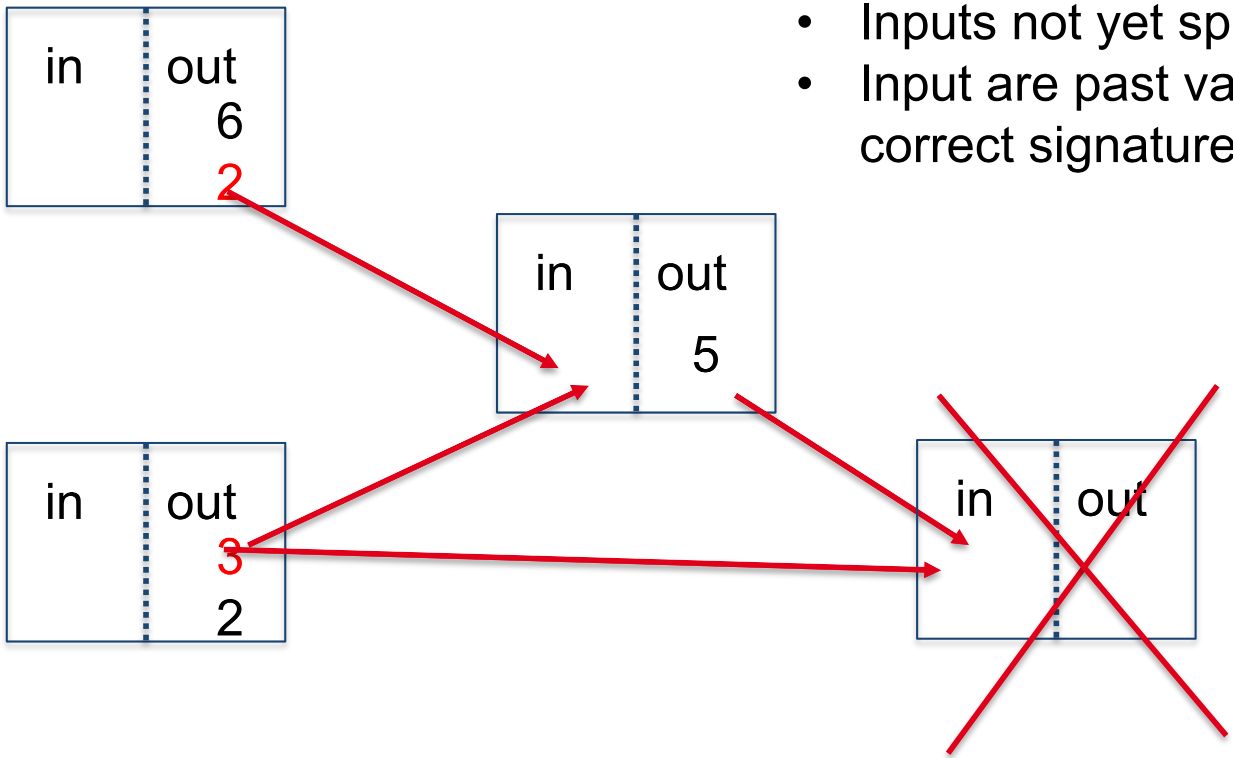
- Be your own bank
  - Public
  - Trustless
  - Decentralized
  - Resistant to Attacks?
  - Scalable?



Bitcoin Blockchain

- Public / Permissionless blockchain
- Trustless: Fully verifiable
- Based on Proof of Work
- Two operations
  - Append
  - Read

Verifying Transactions



Valid Transaction if

- Sum(inputs) >= Sum(outputs)
- Inputs not yet spent
- Input are past valid outputs w/ correct signatures

Bitcoin Block

- Collection of transactions verified as a whole

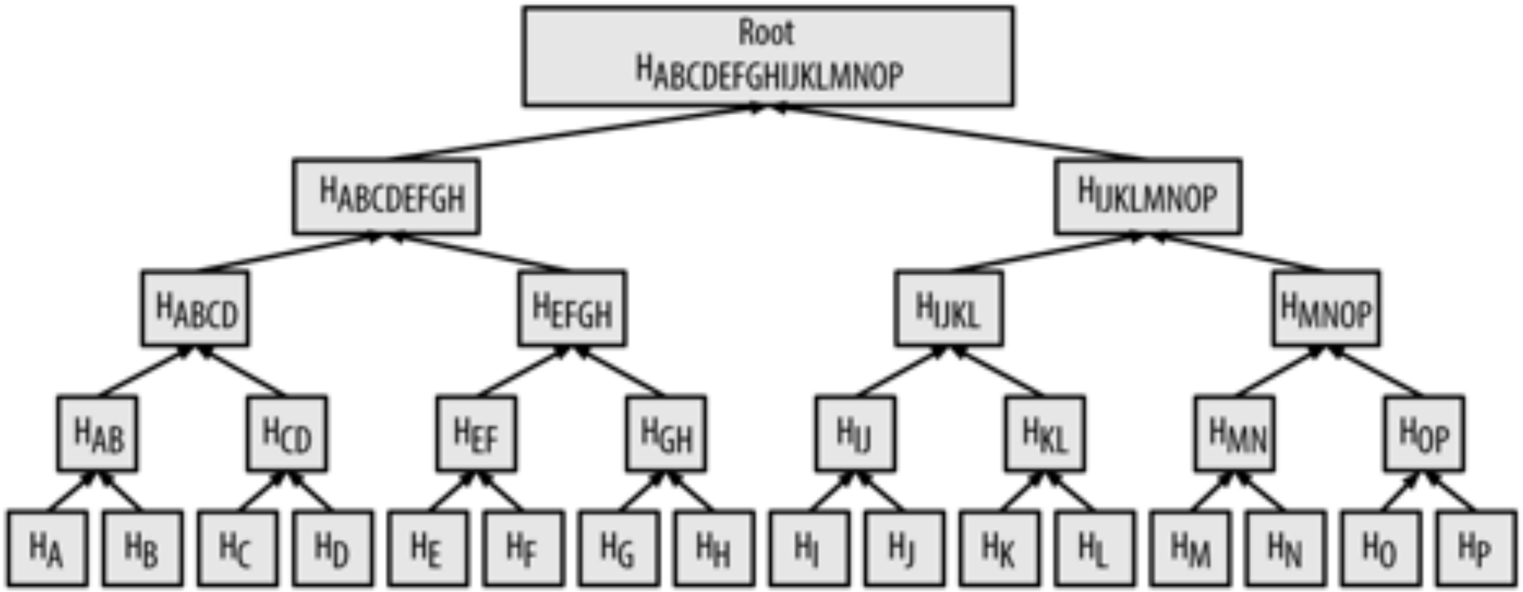
Size	Field	Description
4 bytes	Block Size	The size of the block, in bytes, following this field
80 bytes	Block Header	Several fields form the block header
1–9 bytes (VarInt)	Transaction Counter	How many transactions follow
Variable	Transactions	The transactions recorded in this block

[https://github.com/bitcoinbook/bitcoinbook/blob/develop/ch09.asciidoc#block_structure1]

Bitcoin Block - Header

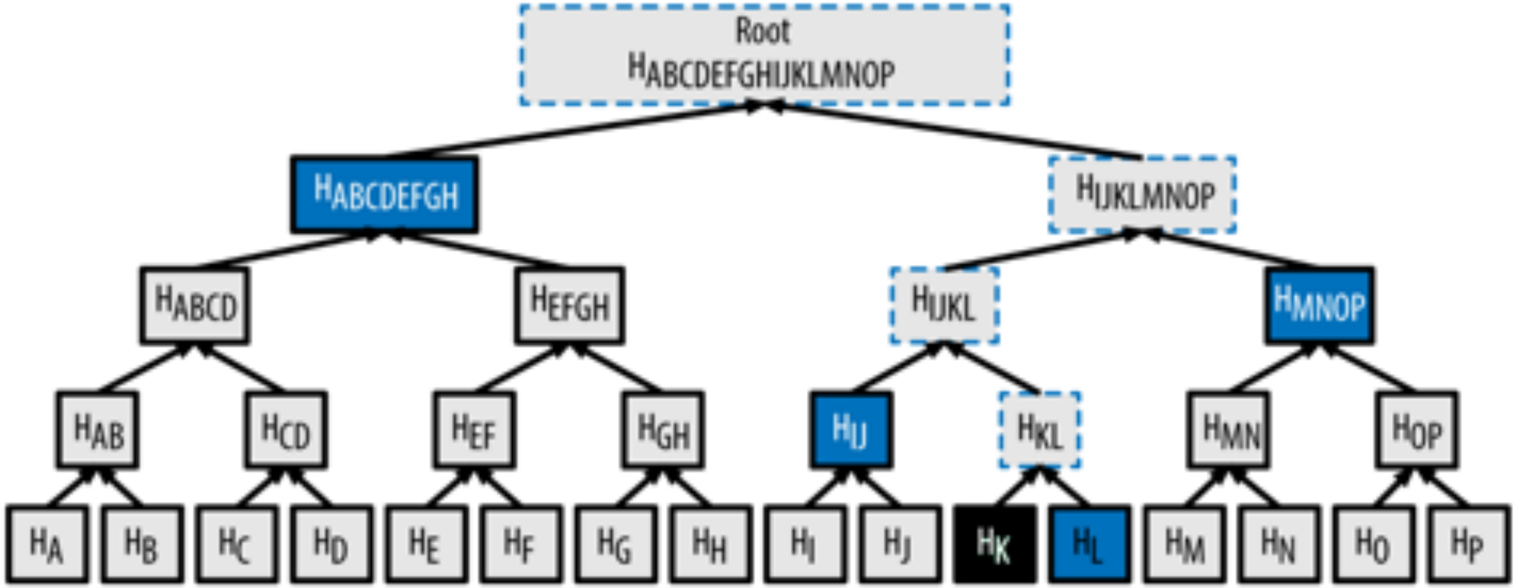
Size	Field	Description
4 bytes	Version	A version number to track software/protocol upgrades
32 bytes	Previous Block Hash	A reference to the hash of the previous (parent) block in the chain
32 bytes	Merkle Root	A hash of the root of the merkle tree of this block's transactions
4 bytes	Timestamp	The approximate creation time of this block (seconds from Unix Epoch)
4 bytes	Difficulty Target	The Proof-of-Work algorithm difficulty target for this block
4 bytes	Nonce	A counter used for the Proof-of-Work algorithm

Merkle Tree of Transactions



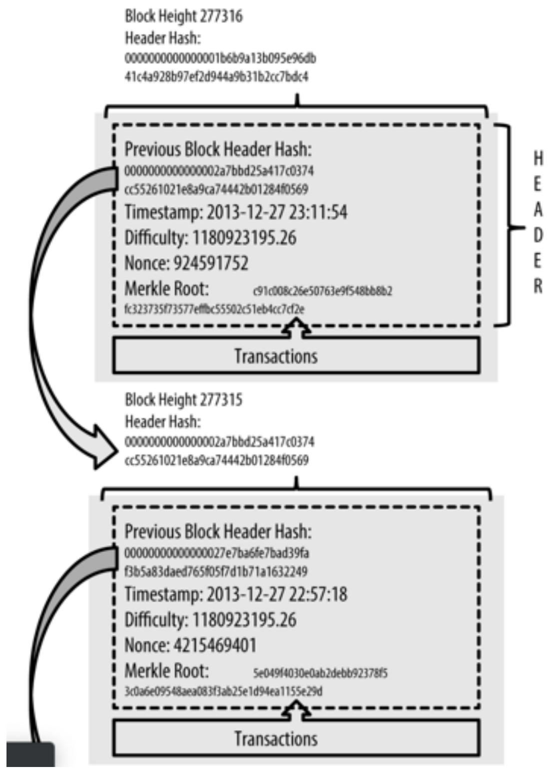
More efficiently prove that transactions are in a block

Merkle Tree of Transactions



Merkle Tree of Transactions

- Header contains root of MT
- Full bitcoin nodes can cache inner tree nodes or recompute them



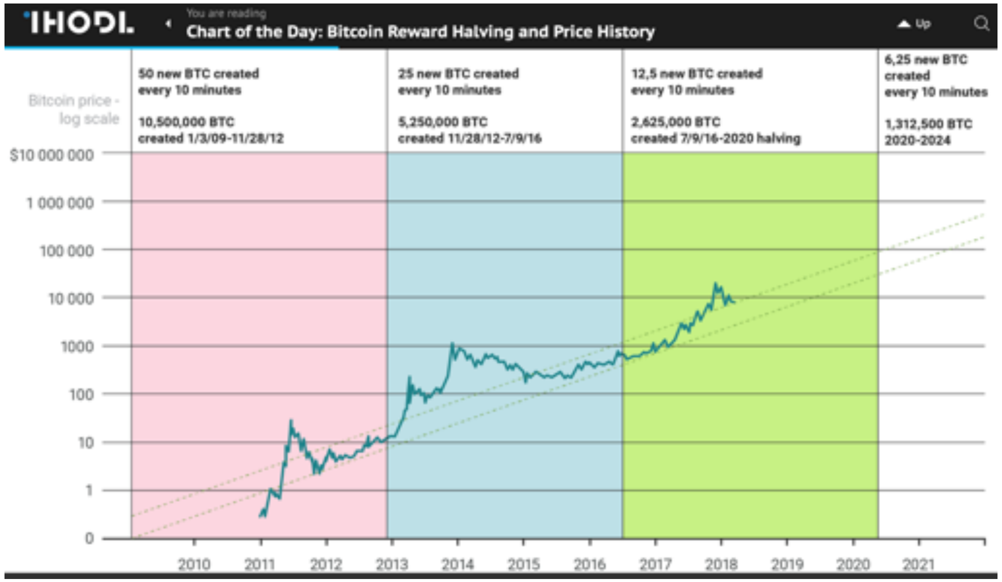
Bitcoin Mining

- Select Transactions:
  - Verify them
  - Put them in block
- Solve Cryptopuzzle:
  - Find a nonce such that
    - block hash starts with given number of leading 0s
- Difficulty: how difficult to find a new block
- Difficulty adjustment
  - Every 2016 blocks
  - So that previous 2016 blocks would take two weeks
    - If took longer reduce difficulty
    - If took shorter increase difficulty

[https://en.bitcoinwiki.org/wiki/Difficulty_in_Mining]

Rewards for Miners

- Miner gets rewards for mining transaction
  - Transaction fees: market driven
  - Bounty: halves every 210000 blocks

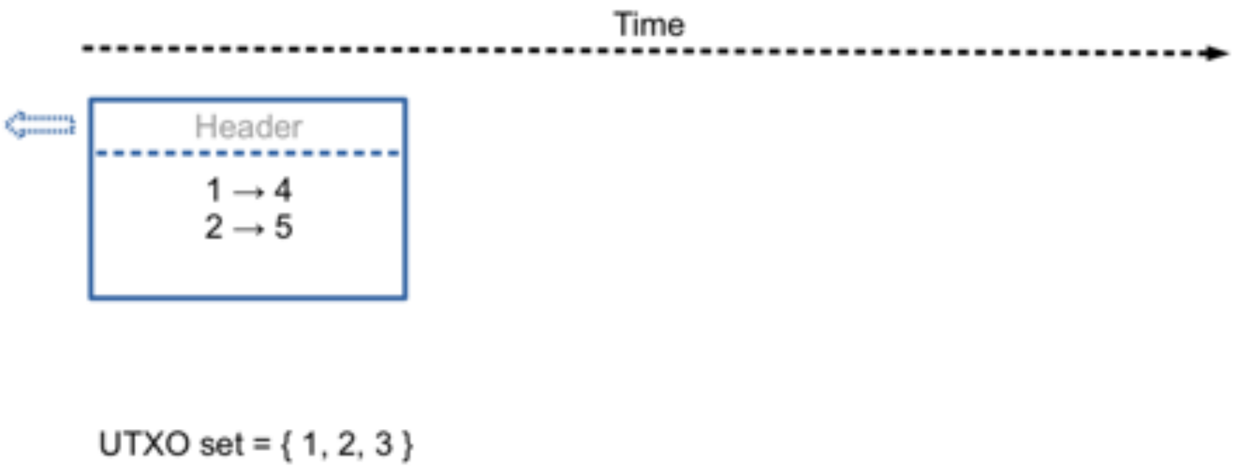


How to Verify Transactions

- Need to replay the whole chain
  - Cannot do it for every transaction
- Maintain local data structure UTxO set
  - Set of unspent transaction outputs
  - Build it by going through the chain at start up
  - Maintain it as new transactions are processed

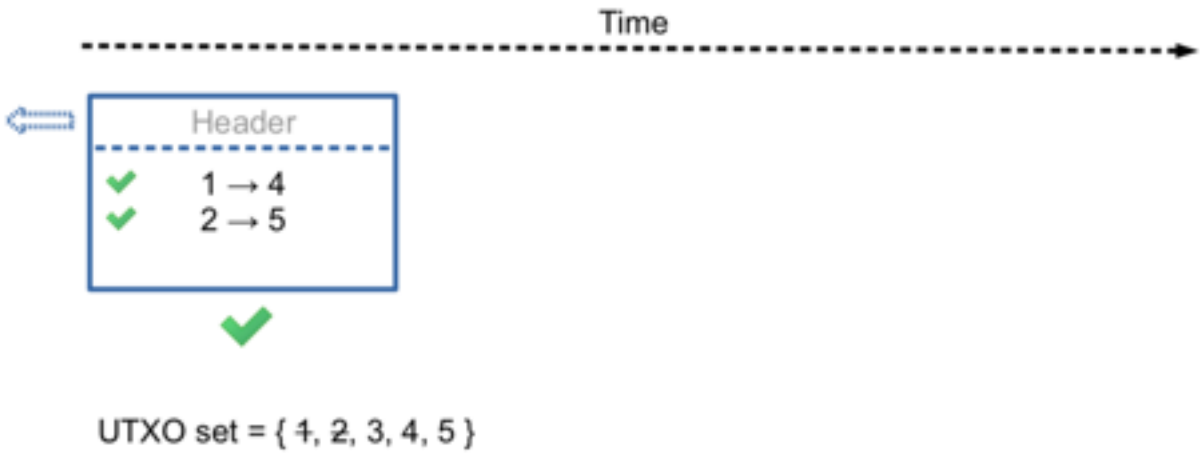
Who Verifies Transactions

- Miners that attempt to create blocks
- All full nodes (even non-miners) that receive a newly mined block



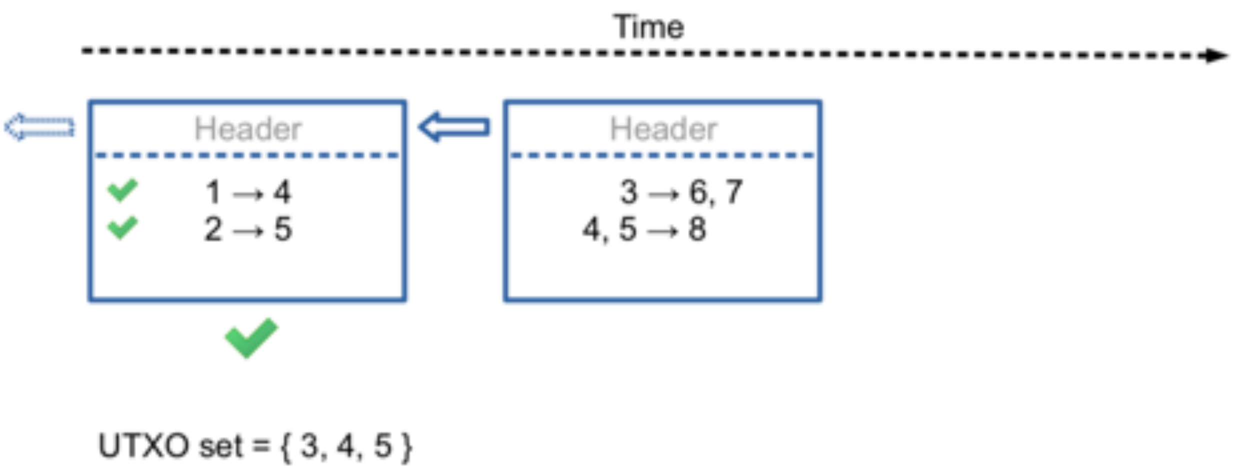
Who Verifies Transactions

- Miners that attempt to create blocks
- All full nodes (even non-miners) that receive a newly mined block



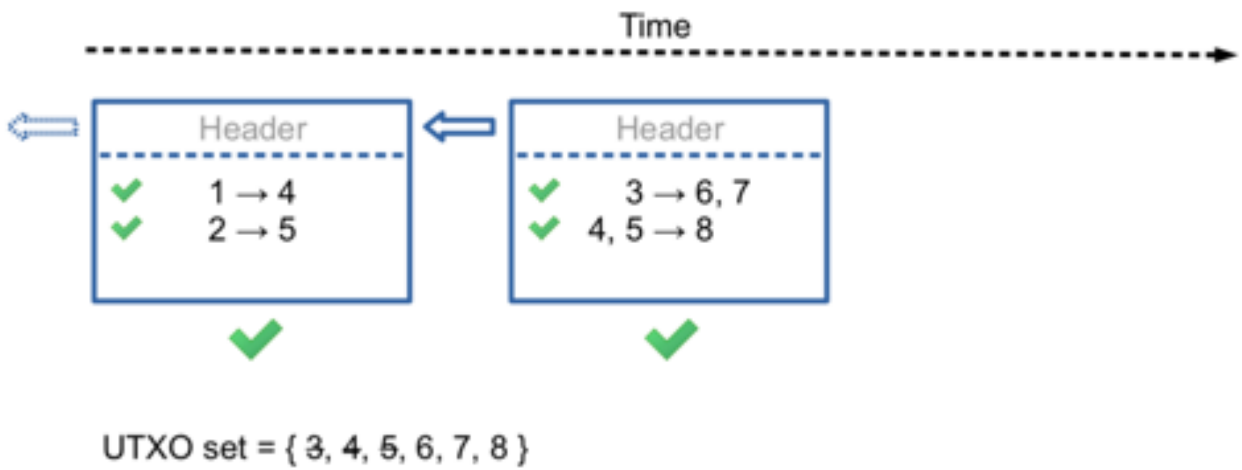
Who Verifies Transactions

- Miners that attempt to create blocks
- All full nodes (even non-miners) that receive a newly mined block



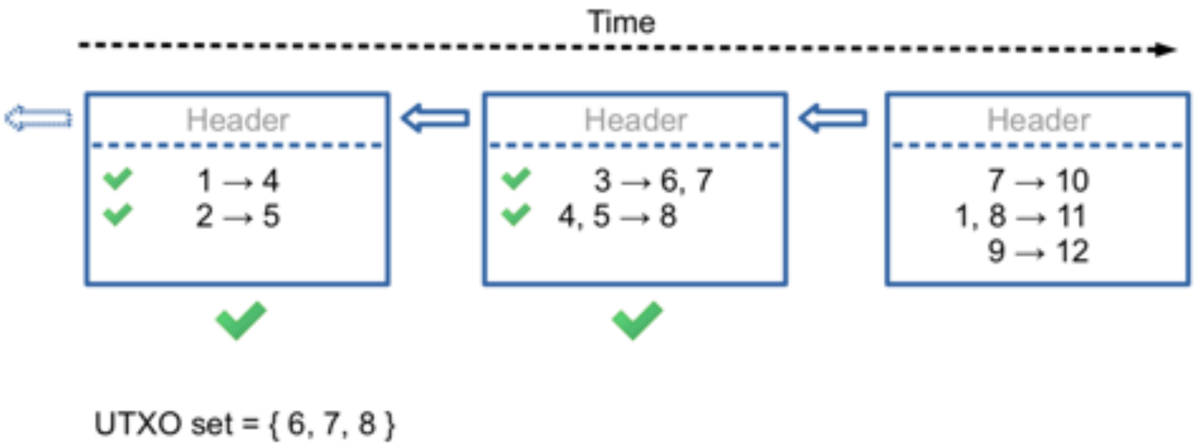
Who Verifies Transactions

- Miners that attempt to create blocks
- All full nodes (even non-miners) that receive a newly mined block



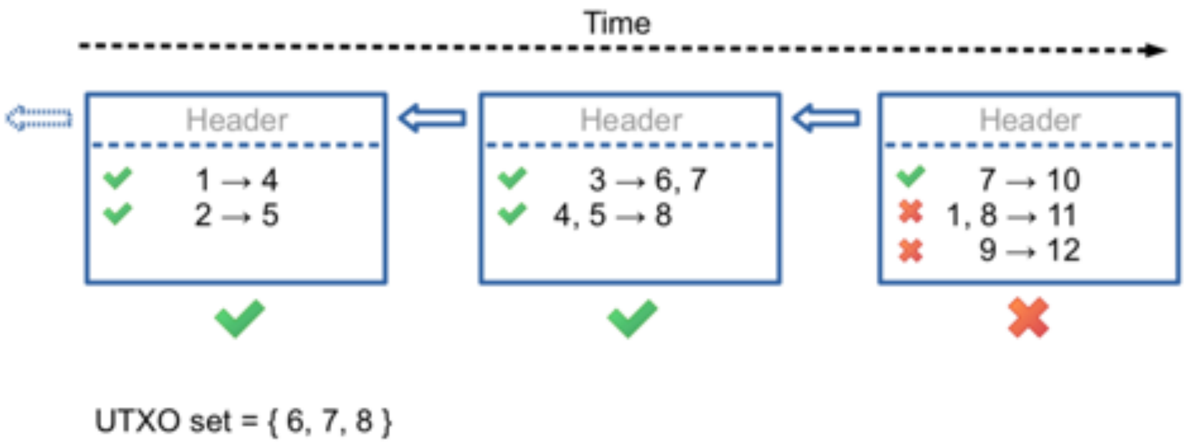
Who Verifies Transactions

- Miners that attempt to create blocks
- All full nodes (even non-miners) that receive a newly mined block



Who Verifies Transactions

- Miners that attempt to create blocks
- All full nodes (even non-miners) that receive a newly mined block



Who Verifies Transactions

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

- Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty $\neq$ Correctness

Simple Payment Verification (SPV)

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

- Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty ≠ Correctness

Blockchain Forks

- What if two miners mine a new block at (approximately) the same time.
- Generally one block propagates faster than the other and fork is resolved quickly

There are also Software Forks

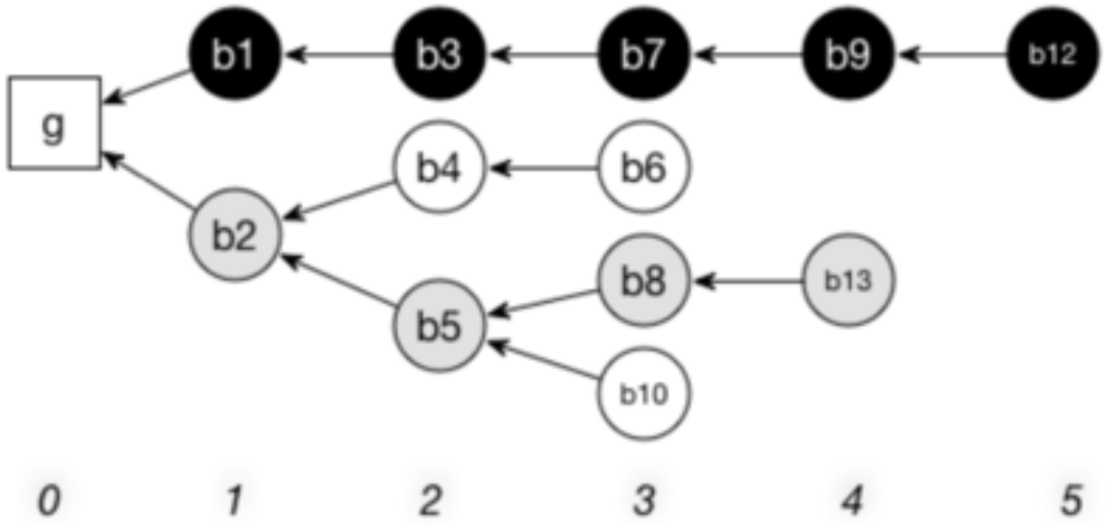
- Soft Fork
  - Backward compatible
  - Old and new version can coexist

- Hard Fork
  - Not backward compatible
  - Split network to form new cryptocurrency

Ethereum

- Faster transaction processing
- One block every 15 seconds
- Longest chain may cause too many forks
- Also Provides Turing Complete Language -> smart contract

Longest Chain vs GHOST



Longest Chain -> Bitcoin
GHOST -> Ethereum (Greedy Heaviest Observed Subtree)

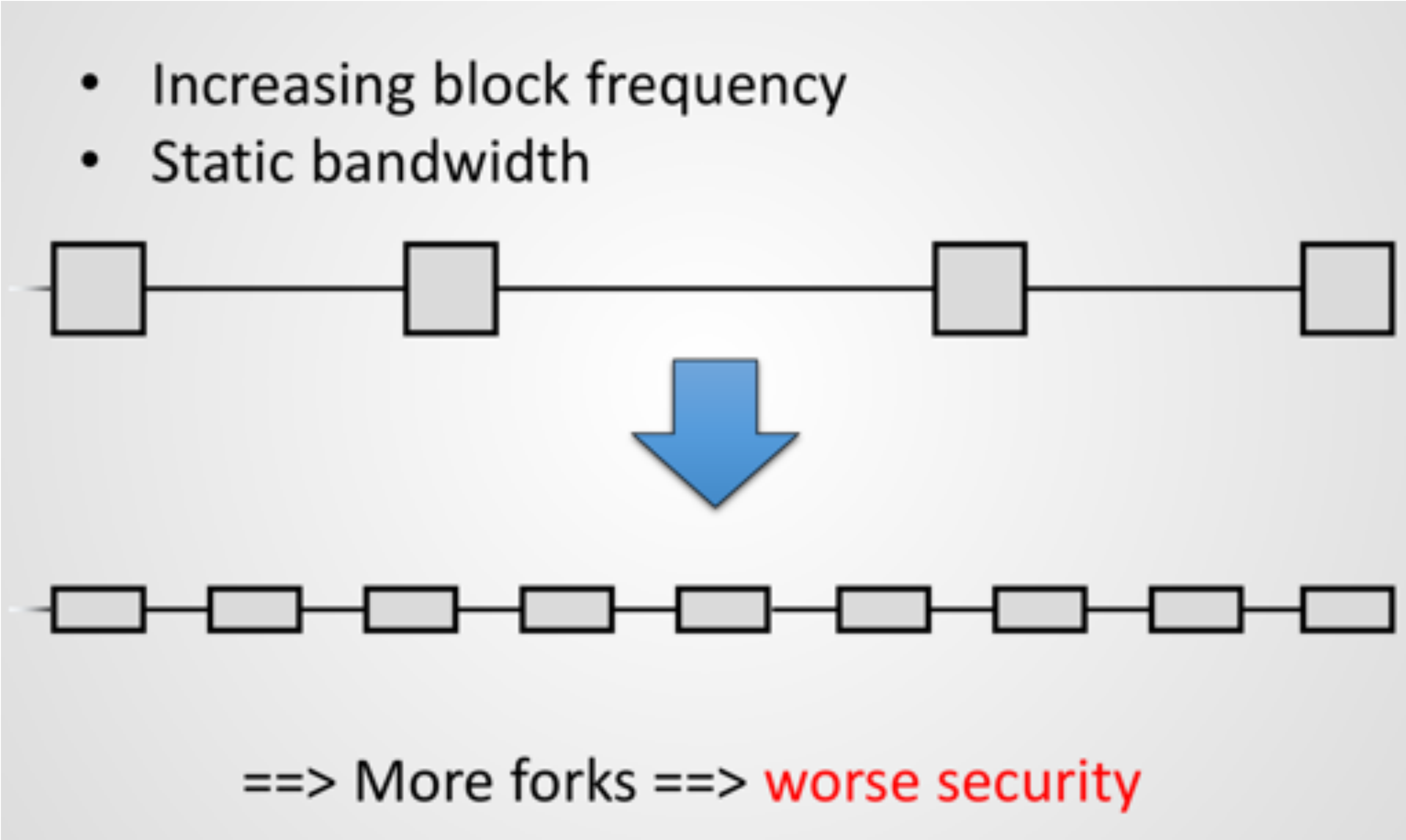
Smart Contracts in Ethereum

- Contract executed by Ethereum Virtual Machine (EVM)
- Written in Solidity scripting language
- Instructions consume GAS
  - GAS has a cost determined by sender
  - GAS limit specified for transactions (default available)
  - GAS avoids infinite loops

Ethereum and SmartContract Lab

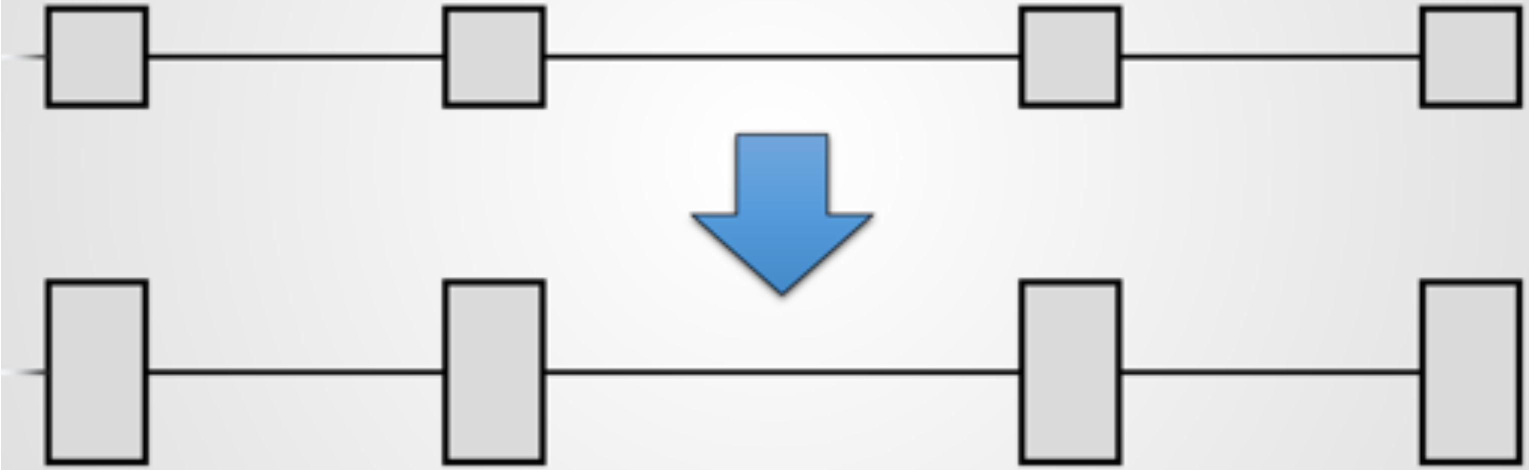
- https://geth.ethereum.org/
- https://solidity.readthedocs.io/en/v0.6.4/installing-solidity.html

Bitcoin NG: Motivation



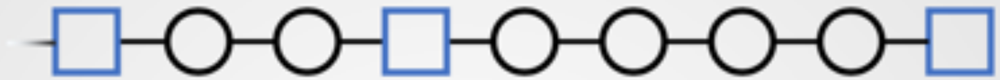
Bitcoin NG: Motivation

- Static block frequency
- Increasing block size



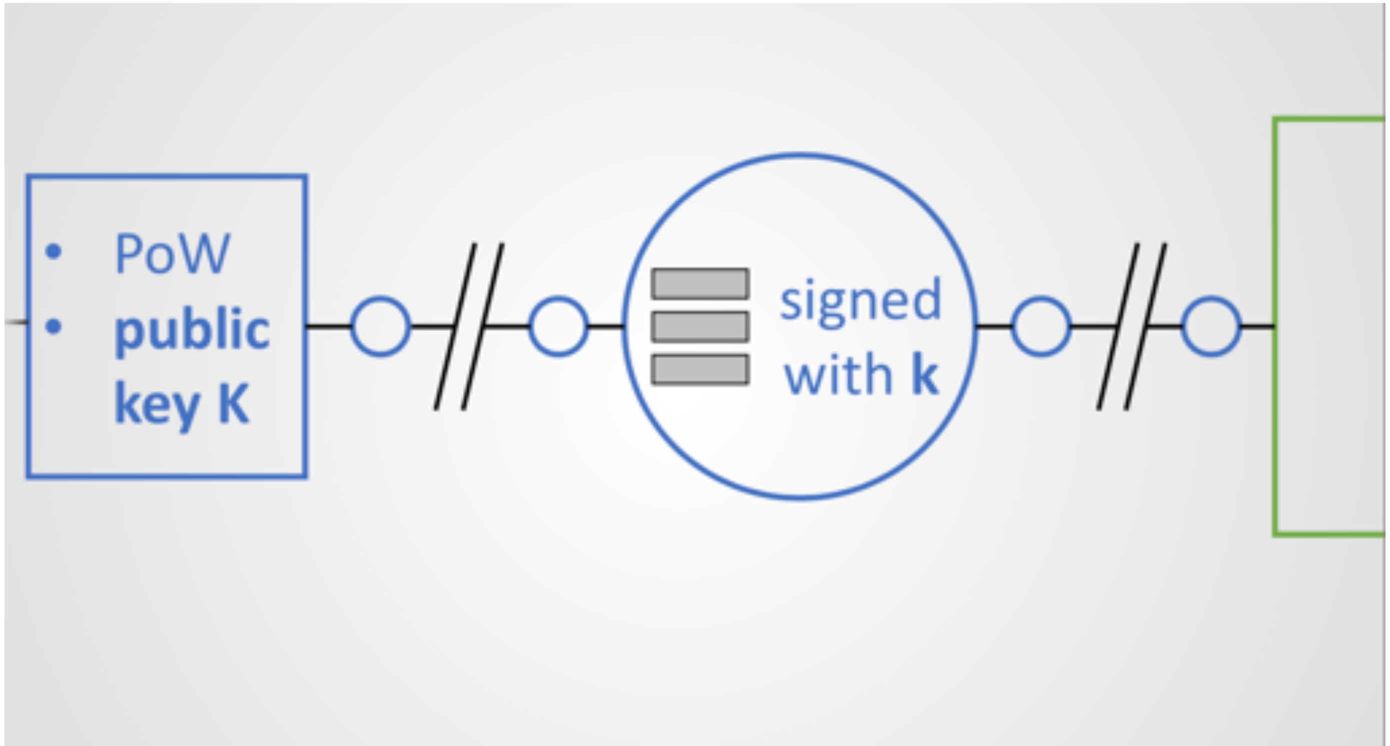
==> More forks ==> worse security

Bitcoin NG

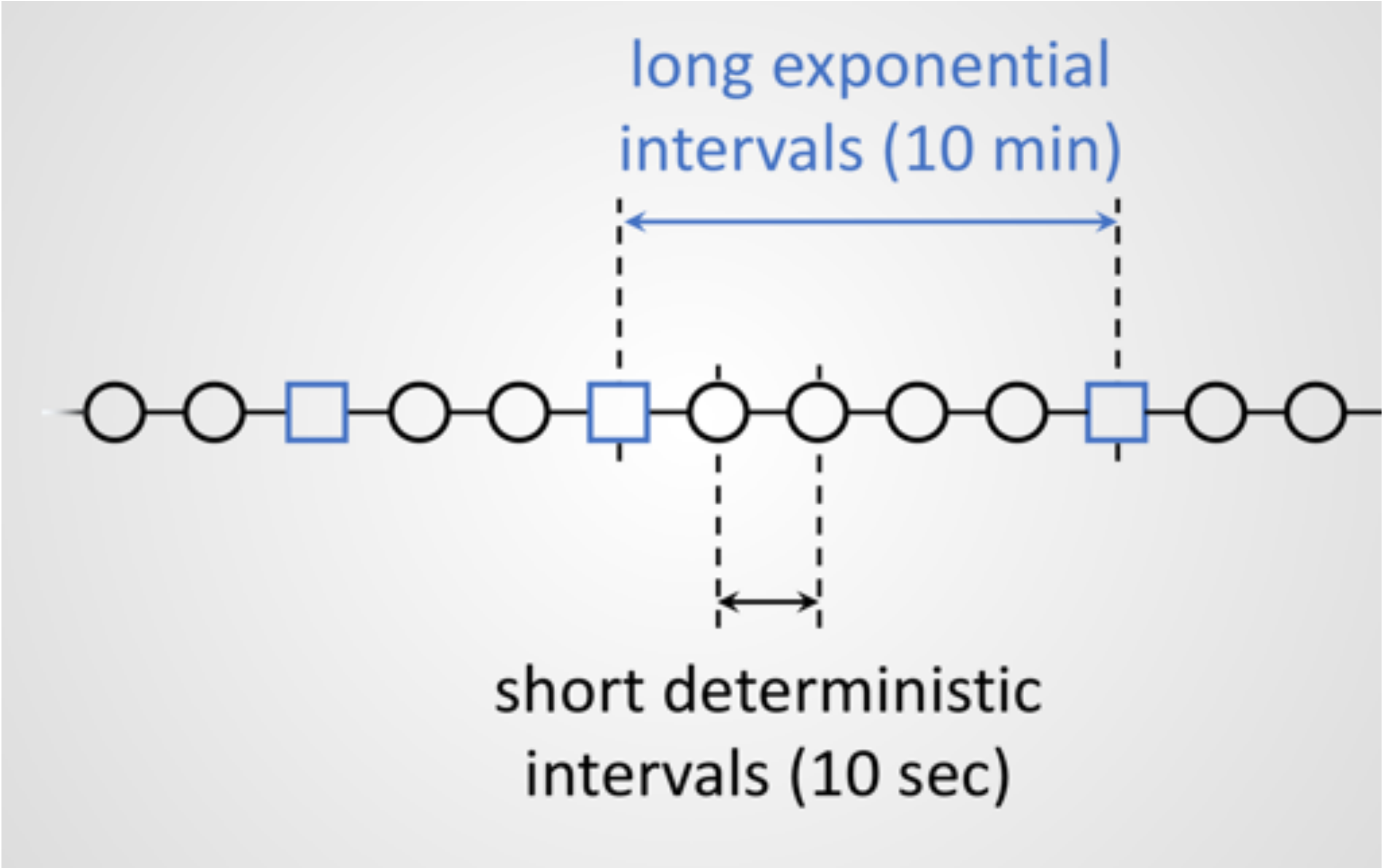


- Key blocks:
  - No content
  - Leader election
- Microblocks:
  - Only content
  - No contention

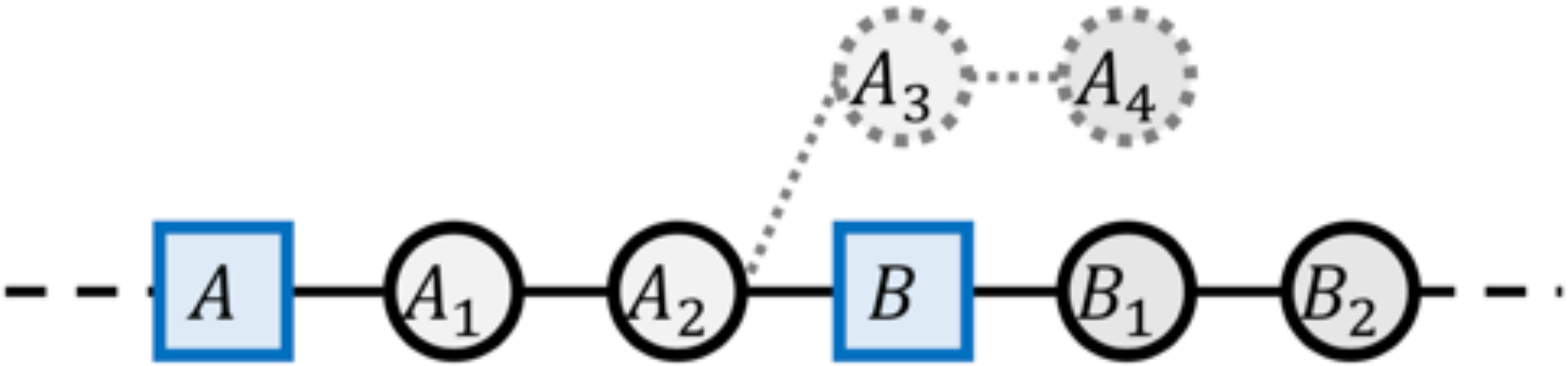
Bitcoin NG



Bitcoin NG


long exponential
intervals (10 min)
short deterministic
intervals (10 sec)

Microblock forks



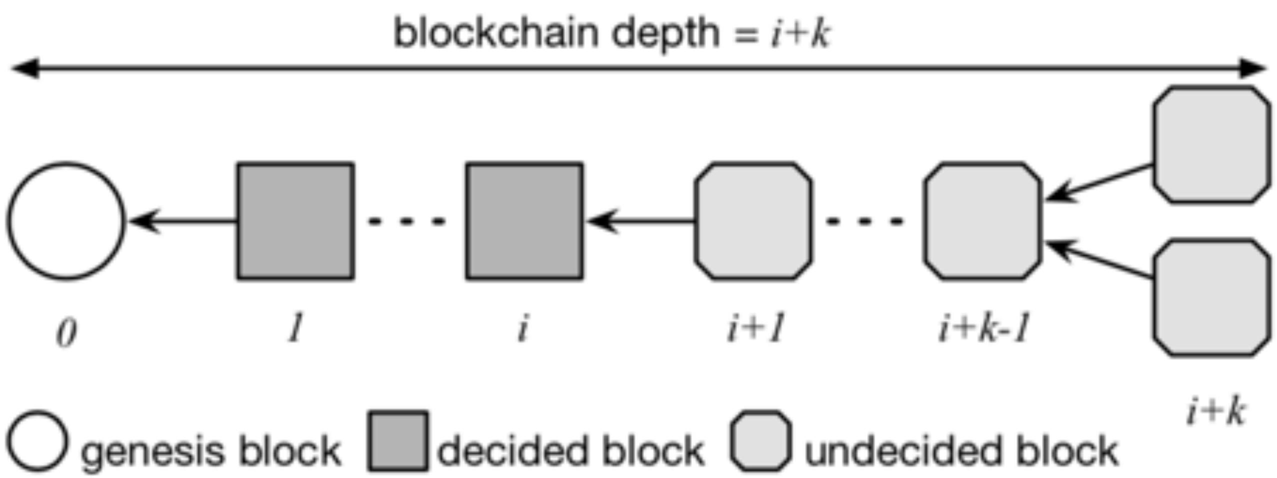
Quickly resolved when node receives new block

Permissioned vs Permissionless

- Permissionless
  - Public
  - Anyone can join
  - Completely decentralized
- Permissioned
  - Private consortia (banks, etc.)
  - Closed ecosystem
  - May be partially centralized
- Private (special case of permissioned)
  - Single trust domain

Consensus

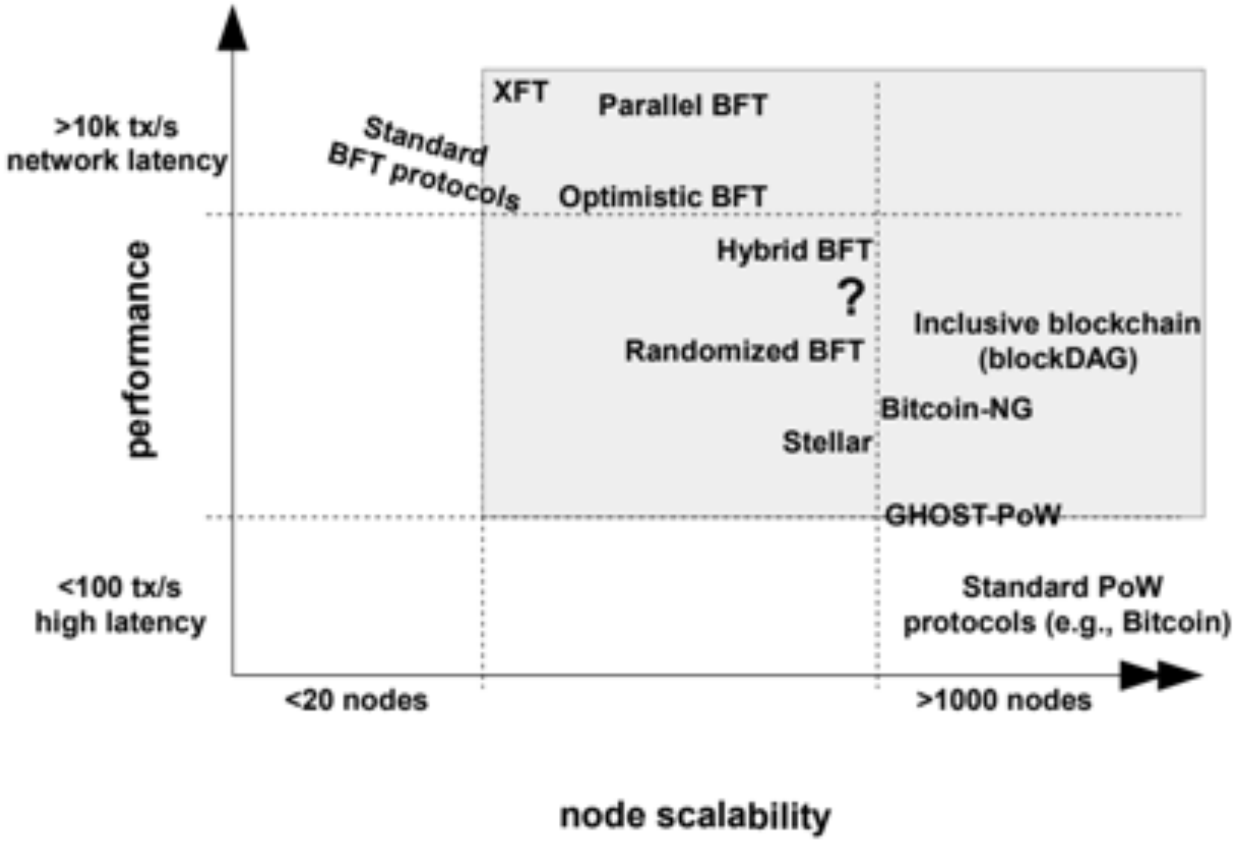
- Agreement
- Validity
- Termination



Proof of Work and Consensus

- Blockchain requires consensus with malicious participants in asynchronous system
- Consensus is impossible in asynchronous system even with just one process that may crash
- ???????

Consensus Protocols



[Marko Vukolic: The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication]

BFT vs Proof of Work

	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**

[Marko Vukolic: The Quest for Scalable Blockchain Fabric: Proof-of-Work vs. BFT Replication]

Proof of Stake

- Combine
  - random block selection
  - Coin-age-based selection
    - Stake represented by coins that have been there for X days
    - Once stake used to sign a block its age is reset

- Nothing at stake
  - In case of fork, validators have interest in mining on both chains
  - Makes double spend easier
  - Solutions exist
  - Casper: Security Deposit

Some Attacks on Proof of Work

- Double Spending
- Easy if you control 51% of the network
- But is it the only case?

Attack Rationale

- Proof of work (and others) only give non-deterministic guarantees
- Cannot be sure that a committed transaction won't be reverted

Blockchain Anomaly

- Delay can cause miners to ”agree” on different branches
- Leads to anomaly
  - Bob will transfer money to Carol only after receiving money from Alice
  - Ta = Alice sends money to Bob
  - Tb = Bob sends money to Carol
- Miner 1 mines block-1 with Ta, Bob sees transaction and issues Tb
- Miner 2 mines another block without Ta and then links another block with Tb.
- Chain of Miner 2 wins -> dependency violated

[Christopher Natoli, Vincent Gramoli: “The Blockchain Anomaly. NCA 2016: 310-317”]

Blockchain Anomaly

Attack Sketch:

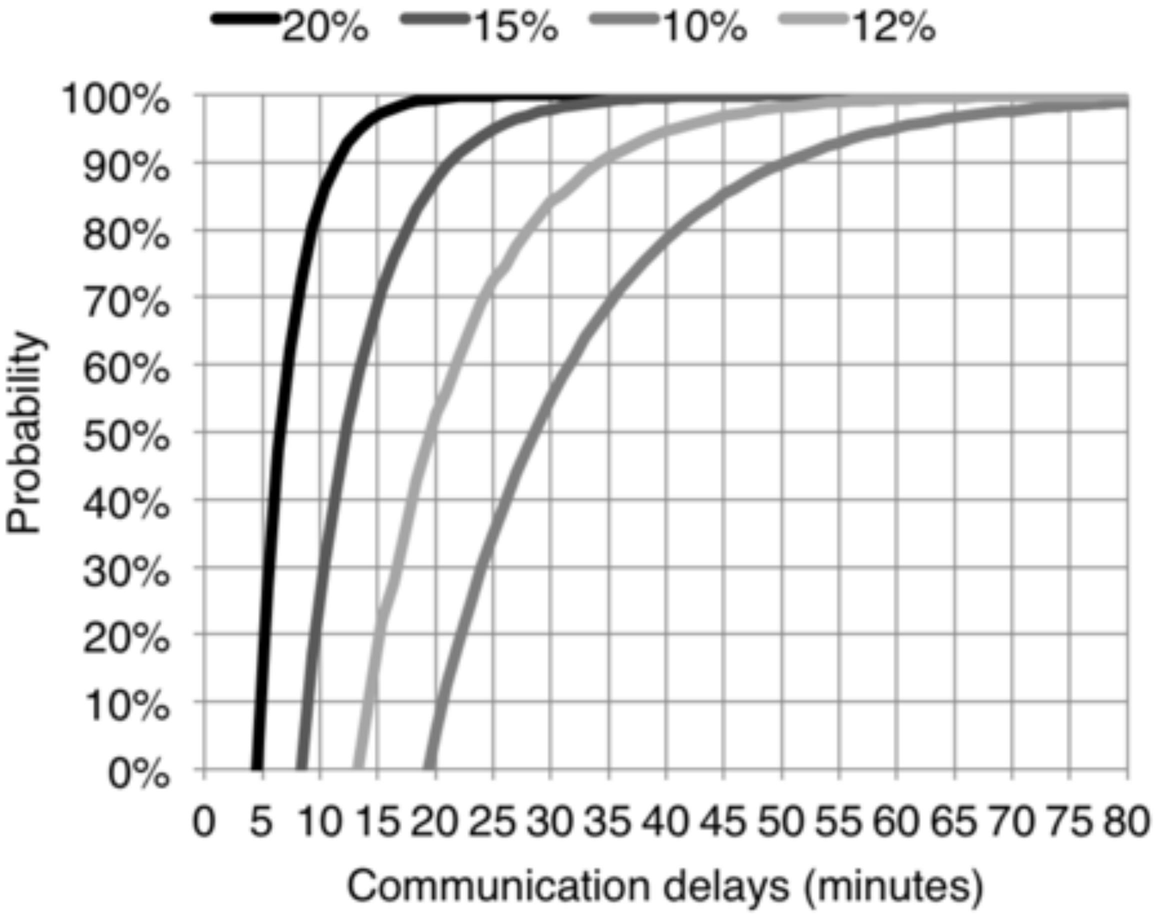
- Powerful Miner 1
- Miner1 buys stuff and waits for his transaction to be in a block
- Miner1 then starts mining in isolation from the previous block
- Then commits lots of blocks but waits until his transaction is 6 blocks deep in the other chain

[Christopher Natoli, Vincent Gramoli: “The Blockchain Anomaly. NCA 2016: 310-317”]

Balance Attack

- Consortium Blockchain
- Attacker can isolate two subgroups
- Operation
  - Isolate two subgroups of equivalent power
  - Issue transaction in one subgroup
  - Mine many blocks in other subgroup
  - Revert transaction when everybody in first subgroup thought it'd be permanent

Balance Attack



Selfish Mining

- The attacker keeps track of its own “private chain”
- Attacker always mines on the private chain keeping blocks private
- Publish blocks when probability of winning is high

Selfish Mining

- State 0: private chain length same as public
  - Mine on private -> if lucky get ahead -> state 1
- State 1: 1 block ahead
  - Mine on private -> if lucky -> state 2

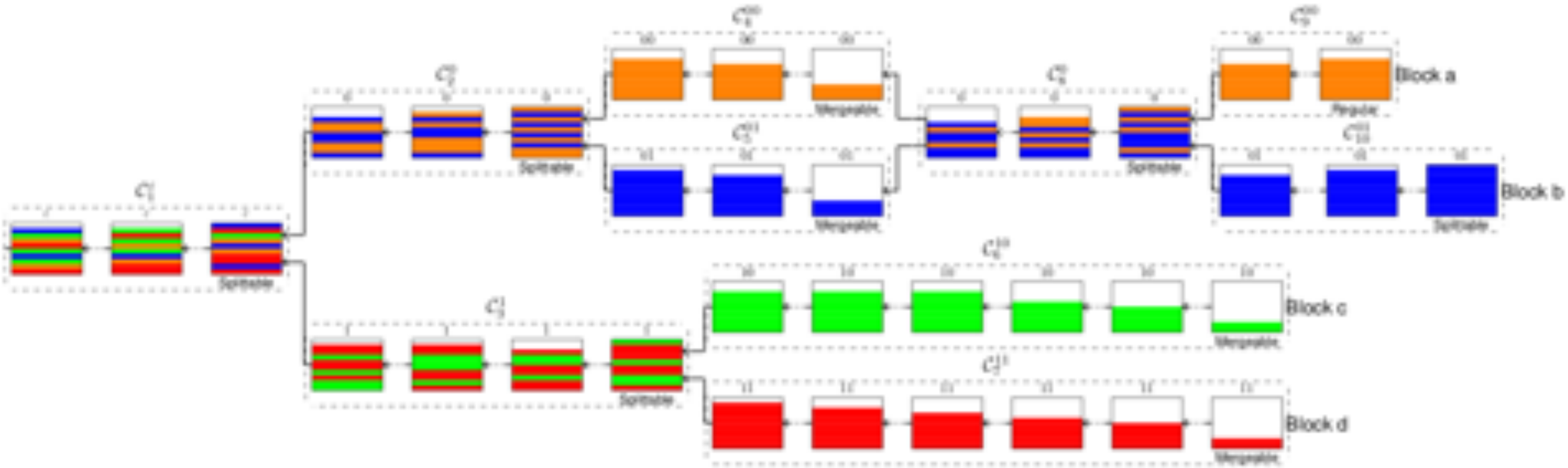
    If not state 0'
- State 0': publish block: two competing chains
  - if lucky attacker chain wins
- But 25% of mining power enough to have good probability of success but can be avoided
- But no defense if attacker has 50%+1 of the network.

Double Spending Attack Probability

q	1	2	3	4	5	6	7	8	9	10
2%	4%	0.237%	0.016%	0.001%	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
4%	8%	0.934%	0.120%	0.016%	0.002%	≈ 0	≈ 0	≈ 0	≈ 0	≈ 0
6%	12%	2.074%	0.394%	0.078%	0.016%	0.003%	0.001%	≈ 0	≈ 0	≈ 0
8%	16%	3.635%	0.905%	0.235%	0.063%	0.017%	0.005%	0.001%	≈ 0	≈ 0
10%	20%	5.600%	1.712%	0.546%	0.178%	0.059%	0.020%	0.007%	0.002%	0.001%
12%	24%	7.949%	2.864%	1.074%	0.412%	0.161%	0.063%	0.025%	0.010%	0.004%
14%	28%	10.662%	4.400%	1.887%	0.828%	0.369%	0.166%	0.075%	0.034%	0.016%
16%	32%	13.722%	6.352%	3.050%	1.497%	0.745%	0.375%	0.190%	0.097%	0.050%
18%	36%	17.107%	8.741%	4.626%	2.499%	1.369%	0.758%	0.423%	0.237%	0.134%
20%	40%	20.800%	11.584%	6.669%	3.916%	2.331%	1.401%	0.848%	0.516%	0.316%
22%	44%	24.781%	14.887%	9.227%	5.828%	3.729%	2.407%	1.565%	1.023%	0.672%
24%	48%	29.030%	18.650%	12.339%	8.310%	5.664%	3.895%	2.696%	1.876%	1.311%
26%	52%	33.530%	22.868%	16.031%	11.427%	8.238%	5.988%	4.380%	3.220%	2.377%
28%	56%	38.259%	27.530%	20.319%	15.232%	11.539%	8.810%	6.766%	5.221%	4.044%
30%	60%	43.200%	32.616%	25.207%	19.762%	15.645%	12.475%	10.003%	8.055%	6.511%
32%	64%	48.333%	38.105%	30.687%	25.037%	20.611%	17.080%	14.226%	11.897%	9.983%
34%	68%	53.638%	43.970%	36.738%	31.058%	26.470%	22.695%	19.548%	16.900%	14.655%
36%	72%	59.098%	50.179%	43.330%	37.807%	33.226%	29.356%	26.044%	23.182%	20.692%
38%	76%	64.691%	56.698%	50.421%	45.245%	40.854%	37.062%	33.743%	30.811%	28.201%
40%	80%	70.400%	63.488%	57.958%	53.314%	49.300%	45.769%	42.621%	39.787%	37.218%
42%	84%	76.205%	70.508%	65.882%	61.938%	58.480%	55.390%	52.595%	50.042%	47.692%
44%	88%	82.086%	77.715%	74.125%	71.028%	68.282%	65.801%	63.530%	61.431%	59.478%
46%	92%	88.026%	85.064%	82.612%	80.480%	78.573%	76.836%	75.234%	73.742%	72.342%
48%	96%	94.003%	92.508%	91.264%	90.177%	89.201%	88.307%	87.478%	86.703%	85.972%
50%	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

[Meni Rosenfeld. Analysis of Hashrate-Based Double Spending]

From chain to DAGs: Sycomore



[Anceaume et al Sycomore : a Permissionless Distributed Ledger that self-adapts to Transactions Demand]

Avalanche

- DAG of Transactions
- Gossip-based probabilistic consensus
- Three protocols for binary consensus
  - Slush
  - Snowflake
  - Snowball

[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

Avalanche

- DAG of Transactions
- Gossip-based probabilistic consensus
- Three protocols
  - Slush
  - Snowflake
  - Snowball

[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

Avalanche - Slush

- Sample values of k random nodes for m times
  - If more than α have different values than own

    Flip value
- Decide value at round m

Key concept: Metastability

- Once one value gains majority, all quickly choose it

[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

Avalanche - Snowflake

- Repeat
  - Sample k nodes and record value if > α votes
- until β consecutive samples yield same value
- Decide value

[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

Avalanche - Snowball

- Repeat
  - Sample *k* nodes

    If > α for same value, increment counter for value

    If counter v1 > counter v2 select value v1

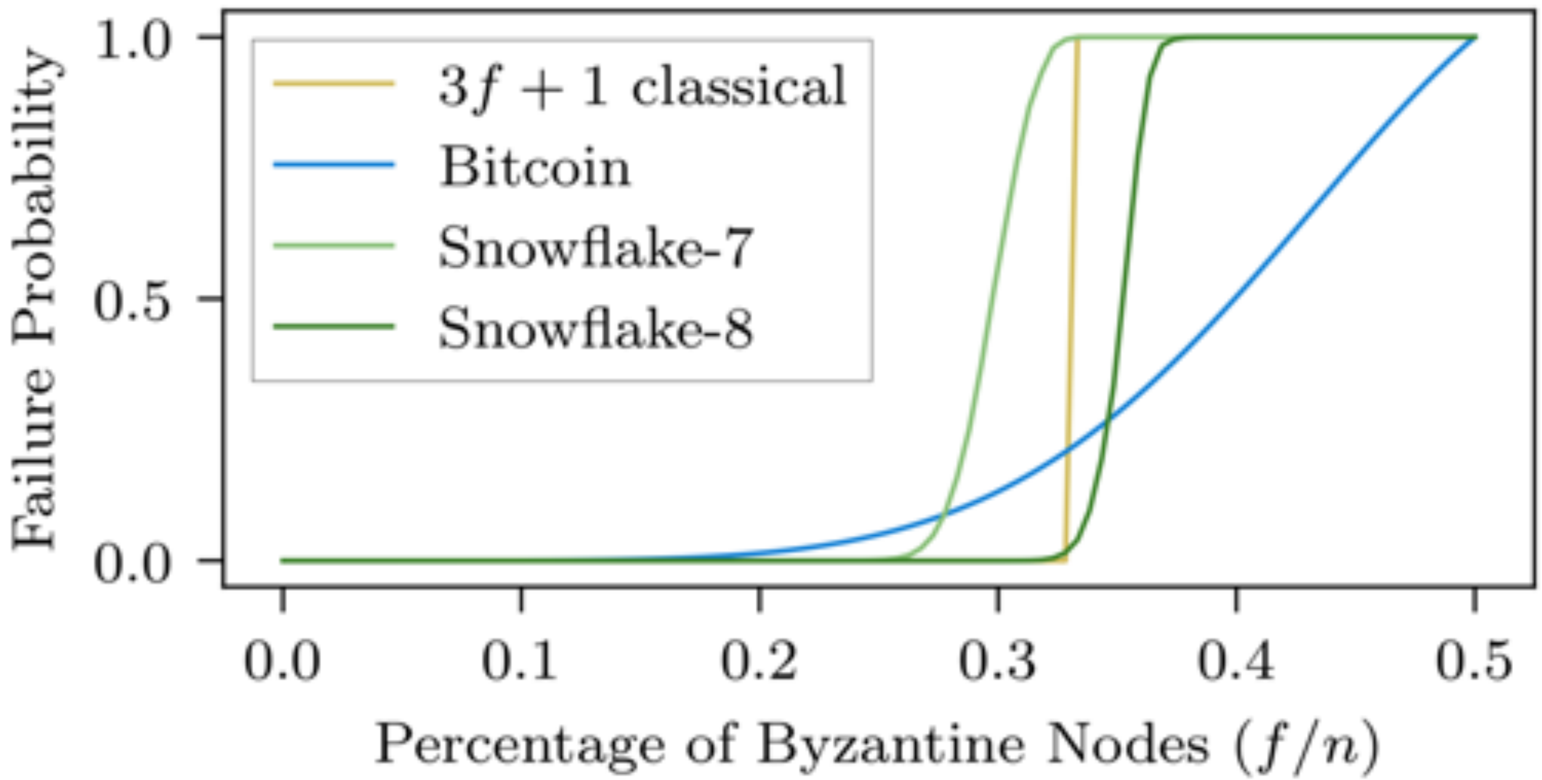
    If counter v2 > counter v1 select value v1

    same color for

  - until β consecutive iterations select same value
- Decide value

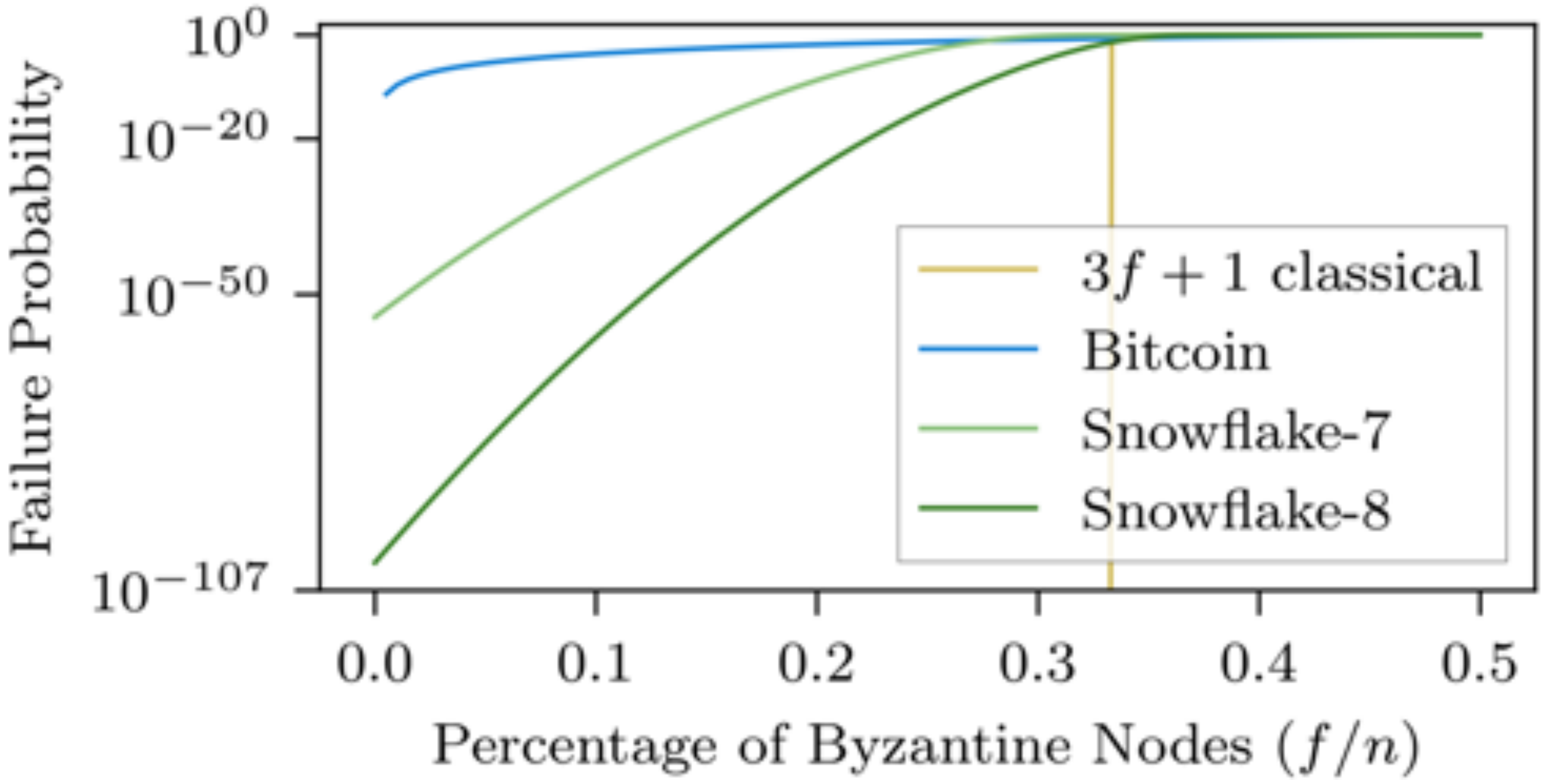
[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

Avalanche - Safety



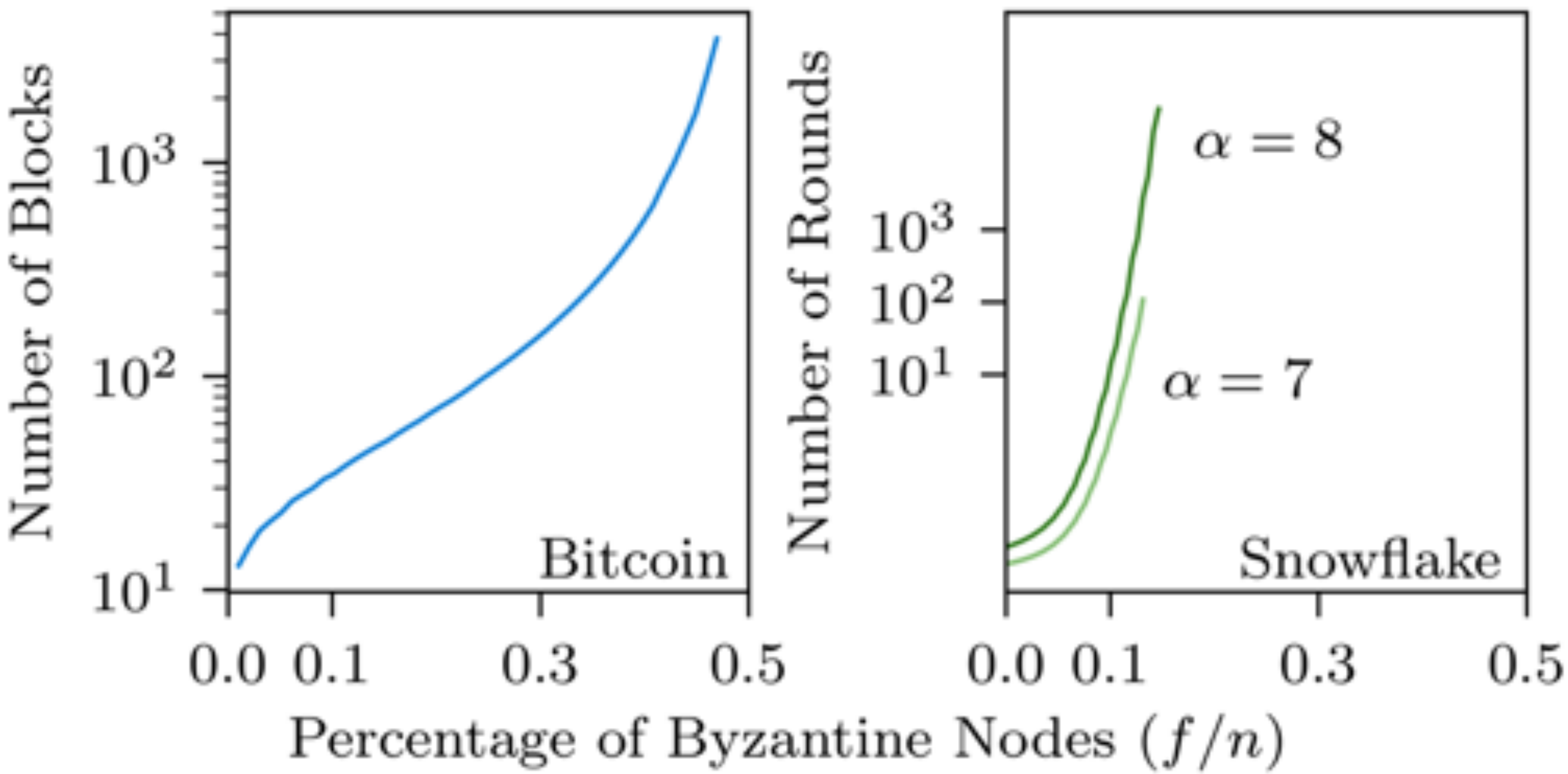
[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

Avalanche - Safety



[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

Avalanche – Latency / Liveness



For failure prob < $\varepsilon = 10^{-20}$

[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

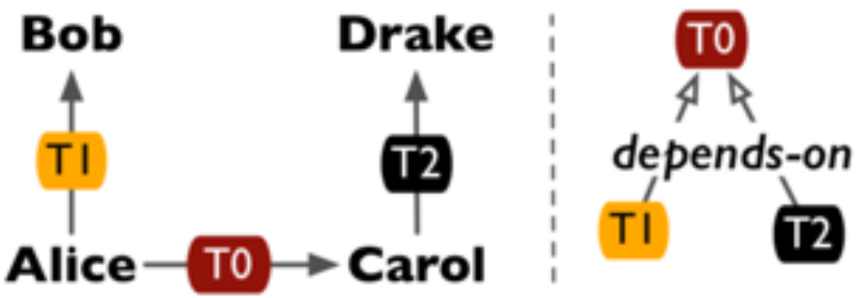
Avalanche

- DAG of Transactions
- Uses variant with Multi valued consensus
  - To arbitrate among conflicting transactions
- Available at: https://github.com/ava-labs/gecko

[Team Rocket, Maofan Yin, Kevin Sekniqi, Robbert van Renesse, and Emin Gun Sirer
« Scalable and Probabilistic Leaderless BFT Consensus through Metastability »]

Is Consensus Really Needed?

- Guerraoui et al: AT2 [PODC 2019] / [DISC 2019]
  - Consensus unnecessary for cryptocurrency
  - Some form of ordered reliable broadcast is enough
    - Causality-like property
  - DAG of transactions



Consensus Number of a Cryptocurrency

PODC 2019

- Consensus number: maximum number of nodes that can reach consensus given an object
- Asset transfer has consensus number
  - 1, if accounts are held by one person each
  - K, for accounts held by k persons.

Consensus Number of a Cryptocurrency

Shared variables:

AS, atomic snapshot, initially $\{\perp\}^N$

Local variables:

$ops_p \subseteq \mathcal{A} \times \mathcal{A} \times \mathbb{N}$, initially $\emptyset$

Upon *transfer*(a, b, x)

$S = AS.snapshot()$
if $p \notin \mu(a) \vee balance(a, S) < x$ **then**
  return *false*
$ops_p = ops_p \cup \{(a, b, x)\}$
$AS.update(ops_p)$
return *true*

Upon *read*(a)

$S = AS.snapshot()$
return $balance(a, S)$

In shared memory model

- Atomic snapshot object

Proves that consensus number is 1 if account held by 1 user

Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. 2019. The Consensus Number of a Cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (*PODC '19*). Association for Computing Machinery, New York, NY, USA, 307–316. DOI:https://doi.org/10.1145/3293611.3331589

Consensus Number of a Cryptocurrency

- With accounts owned by k users:
  - Can implement consensus among k processes
  - => Consensus number = k

Shared variables:

$R[i]$, $i \in 1, \ldots, k$, k registers, initially $R[i] = \bot, \forall i$

AT, k-shared asset-transfer object containing:

– an account a with initial balance $2k$ owned by processes $1, \ldots, k$

– some account s

Upon *propose*(v):

$R[p].write(v)$

$AT.transfer(a, s, 2k - p)$

return $R[AT.read(a)].read()$

Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. 2019. The Consensus Number of a Cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (*PODC '19*). Association for Computing Machinery, New York, NY, USA, 307–316. DOI:https://doi.org/10.1145/3293611.3331589

Consensus Number of a Cryptocurrency

- In Message-Passing model
  - Use a reliable broadcast primitive

- **Integrity:** A benign process delivers a message m from a process p at most once and, if p is benign, only if p previously broadcast m.
- **Agreement:** If processes p and q are correct and p delivers m, then q delivers m.
- **Validity:** If a correct process p broadcasts m, then p delivers m.
- **Source order:** If p and q are benign and both deliver m from r and m' from r, then they do so in the same order.

Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovič, and Dragos-Adrian Seredinschi. 2019. The Consensus Number of a Cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing* (*PODC '19*). Association for Computing Machinery, New York, NY, USA, 307–316. DOI:https://doi.org/10.1145/3293611.3331589

How about Smart Contracts?

- In the general case (Turing complete)
  - Need consensus
    - No difference from classical distributed state machine
- Maybe there are intermediate cases
  - Open research avenue

Scalable Byzantine Reliable Broadcast

- Probabilistic Sample-based algorithm
- Inspired by Bracha’s Byzantine Reliable broadcast algorithm
- Unlike Bracha’s, it is suitable for open/permissionles systems

[Rachid Guerraoui, Petr Kuznetsov, Matteo Monti, Matej Pavlovic, Dragos-Adrian Seredinschi: **Scalable Byzantine Reliable Broadcast.** DISC 2019: 22:1-22:16]

To Take Away

- Bitcoin introduced a new concept
  - Great engineering feat
- Ethereum generalized to Byzantine State-Machine Replication in open systems
- Still Poorly understood in theory
  - 10 years to show that blockchain not needed for cryptocurrency
- Several open topics
  - Specification of distributed ledger
  - Characterization of distributed ledger
  - Weaker byzantine objects
  - Generalizing BFT algorithms to open systems

Dietcoin:

Hardening Bitcoin Transaction Verification Process For Mobile Devices.

Davide Frey, Marc X. Makkes, Pierre-Louis Roman, François Taïani, Spyros Voulgaris:

Inria

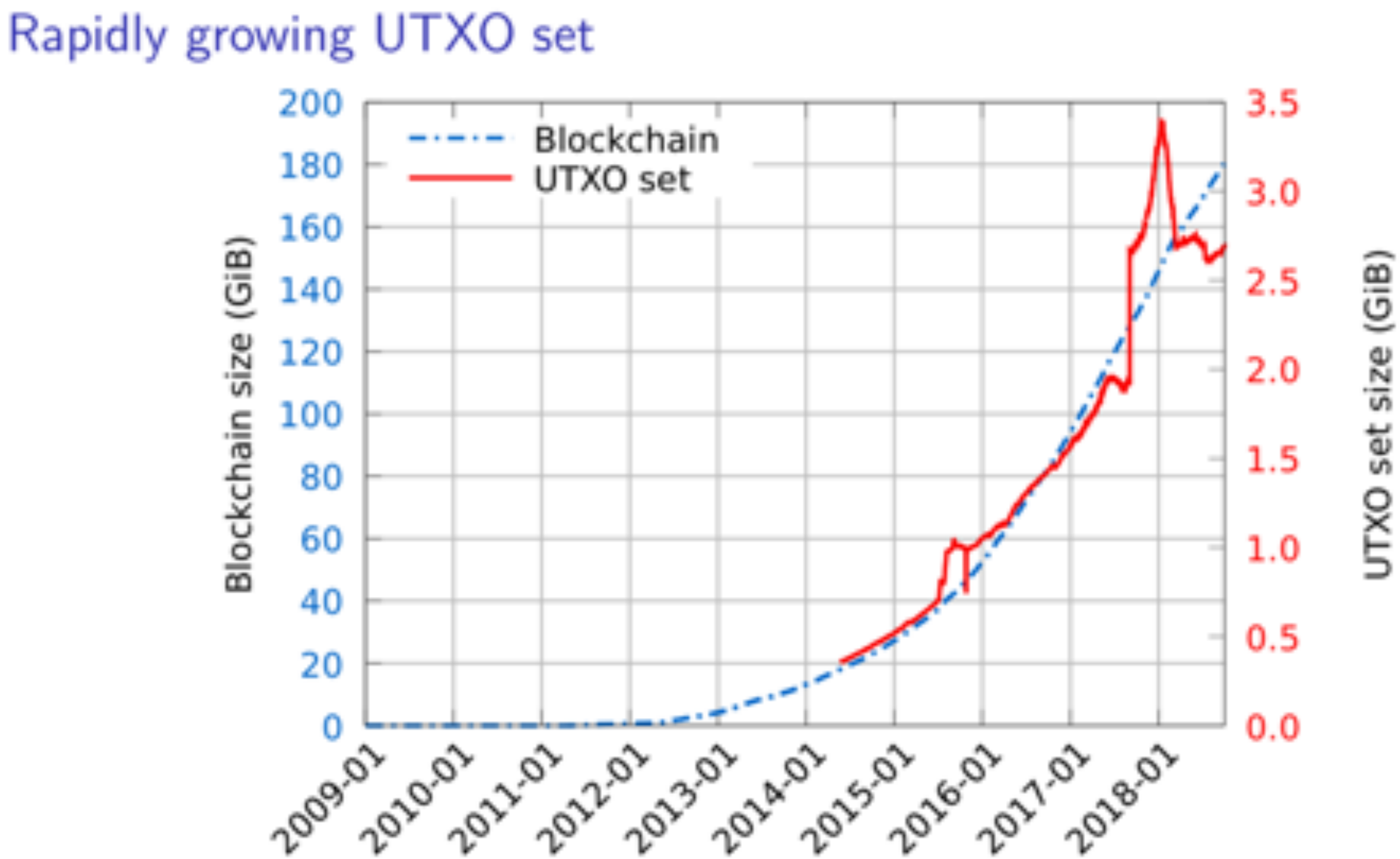
Light Nodes

- Replicate chain locally
- Rule-based verification
- Verify headers and transactions

- Sequential verification
- Time and bandwidth expensive bootstrap

Difficulty ≠ Correctness

UTxO is Growing Large Too



Intuition

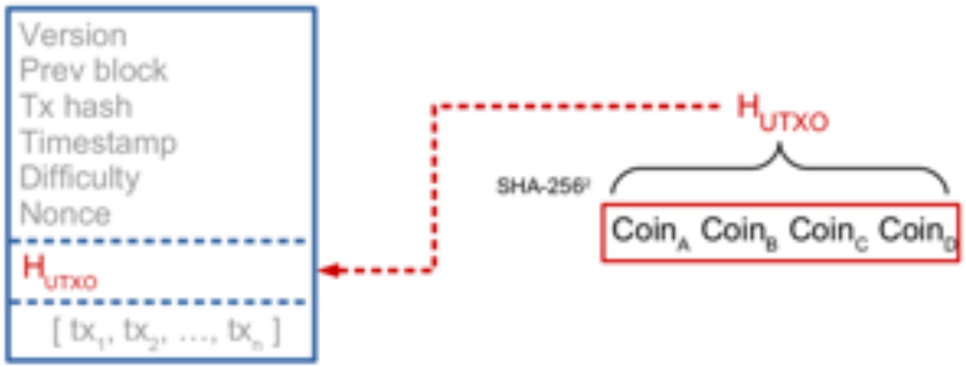
Make the UTXO set queriable by light nodes

- Diet node = light node + transaction verification
- Fast bootstrap, improved security

- Diet nodes consume more bandwidth than light nodes

a diet node receives a fake UTXO set?

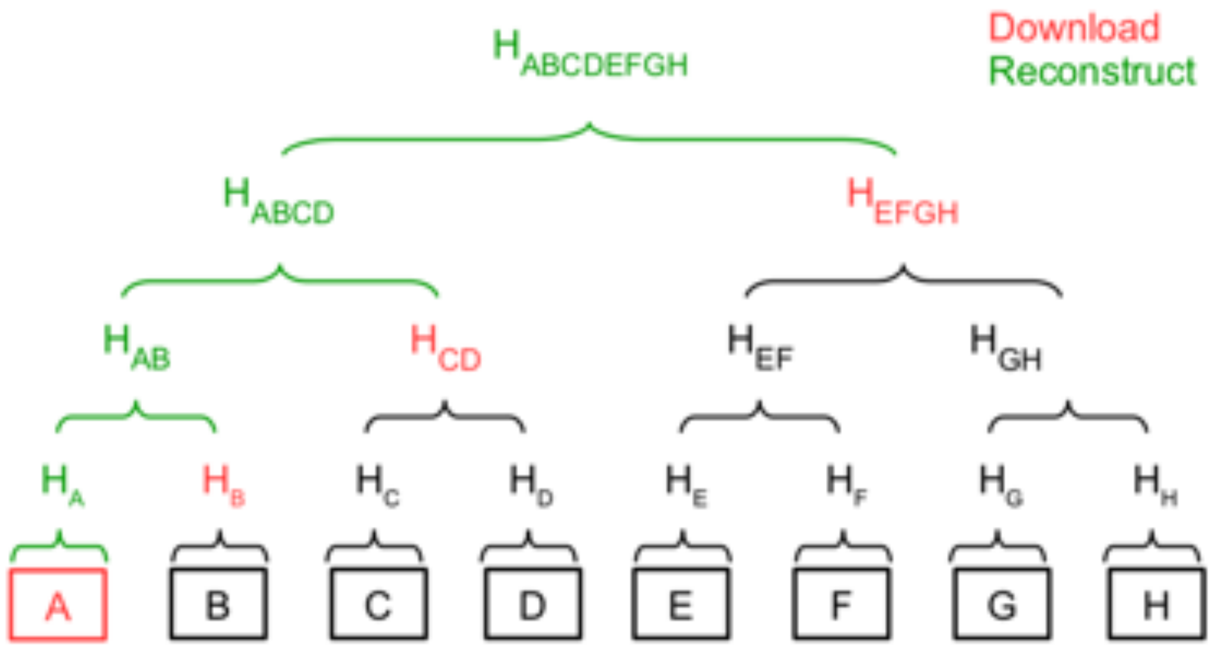
Hash of UTxO Set



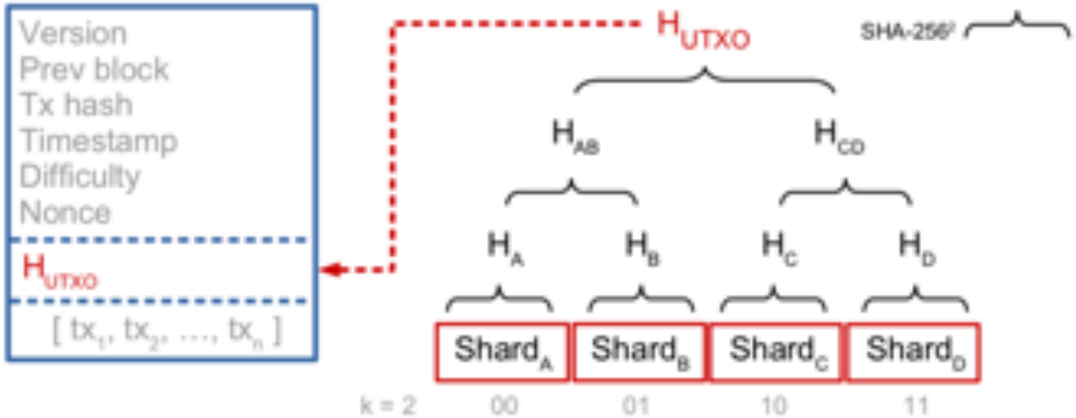
- It works!

- A node has to download the entire UTXO set, even for small queries

Shard UTxO and use Merkle Tree

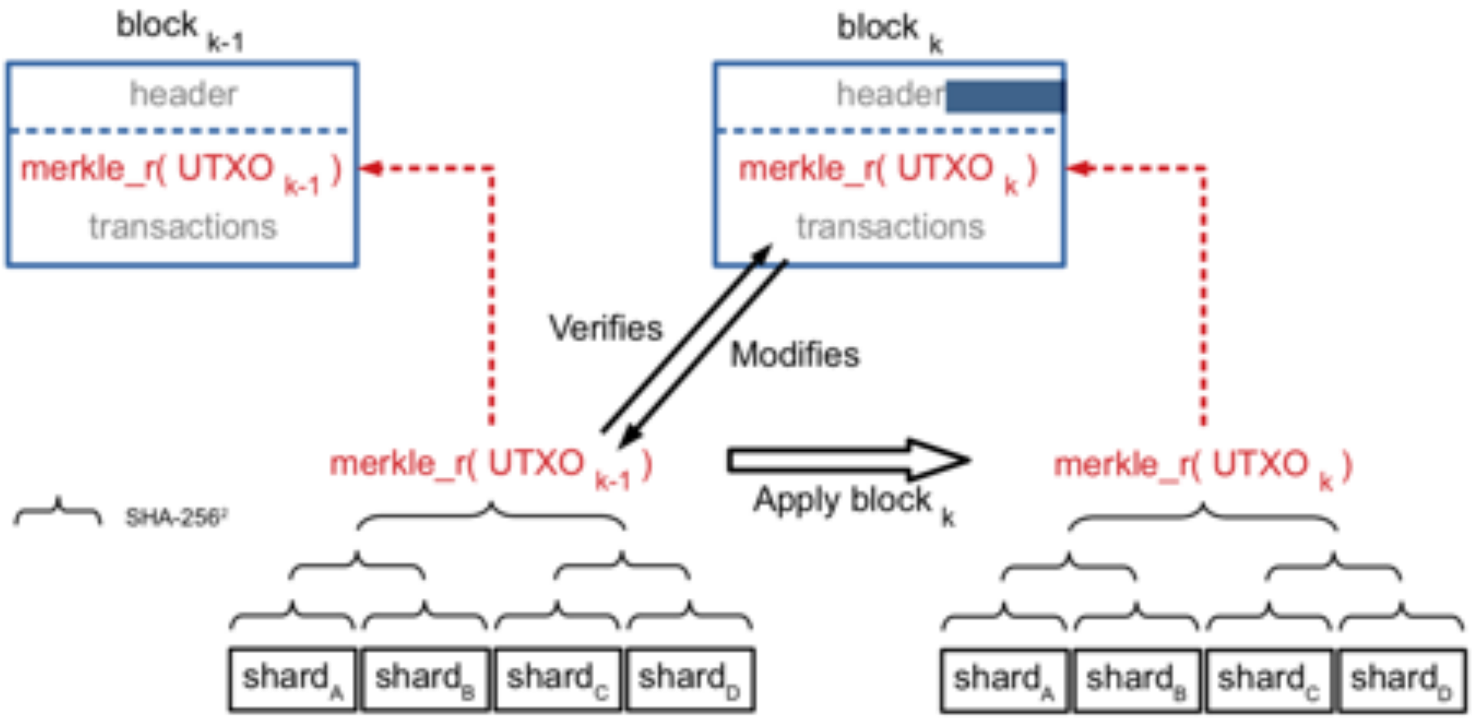


Dietcoin



- Full node storage overhead: 128 MiB $(k = 22 \Rightarrow |\text{hashes}| = 2^{22})$
- Diet node bandwidth consumption: 12.8 MiB of query per block
  $10000 \text{ shards} \times (0.64\,\text{KiB} + 22 \times 32\,\text{B}) = 12.8\,\text{MiB}$
- Parameterized trade-off k: bandwidth consumption vs storage overhead
- Stable tree: no insertion, no deletion $\Rightarrow$ Enable subchain verification

Dietcoin



block k-1
header
merkle_r(UTXO k-1)
transactions
block k
header
merkle_r(UTXO k)
transactions
Verifies
Modifies
merkle_r(UTXO k-1)
Apply block k
merkle_r(UTXO k)
SHA-256²
shardA
shardB
shardC
shardD
shardA
shardB
shardC
shardD

Subchain Verification

Trust a block, verify all the next ones
Shift the trust from the genesis block to any block

A diet node verifying the UTXO Merkle root in block B_k

- Queries the UTXO Merkle root in B_{k-1} (B_{k-1} is trusted)
- Queries UTXO shards for transaction inputs and outputs of B_k
- Verifies transactions in B_k, updating its local copies of UTXO shards
- Recomputes its UTXO Merkle root, check it against the one in B_k
- Repeat for the next block

Effectively shortcuts the verification process

Dietcoin Summary

- Diet nodes can verify the correctness of blocks and subchains
- Diet nodes shortcut the verification process

- Inherent overhead for full nodes
- Non-optimal bandwidth consumption

Future work

- Evaluation
- Decoupled storage ⇒ DHT
- Shard compression
- Combine with Non-interactive Proofs of Proof-of-Work (NiPoPoWs)?
- Combine with Ethereum Casper?

- C. Natoli and V. Gramoli, "The Blockchain Anomaly," 2016 IEEE 15th International Symposium on Network Computing and Applications (NCA), Cambridge, MA, 2016, pp. 310-317. doi: 10.1109/NCA.2016.7778635