(Almost)
Bringing secure Bitcoin transactions to your smartphone

Davide Frey[1], Marc X. Makkes[2], Pierre-Louis Roman[3],
François Taïani[3], Spyros Voulgaris[2]

[1] Inria Rennes
[2] Vrije Universiteit Amsterdam
[3] University of Rennes 1 – IRISA – ESIR

TL;DR blockchain

Bitcoin: peer-to-peer payment system, backed by a blockchain

Blockchain overview:

- Distributed database: full replica at every node
- Trustless: no authority, nodes verify everything locally
- Probabilistic immutability: very difficult to replace a block
- Resistant: incentives to have honest nodes

TL;DR blockchain

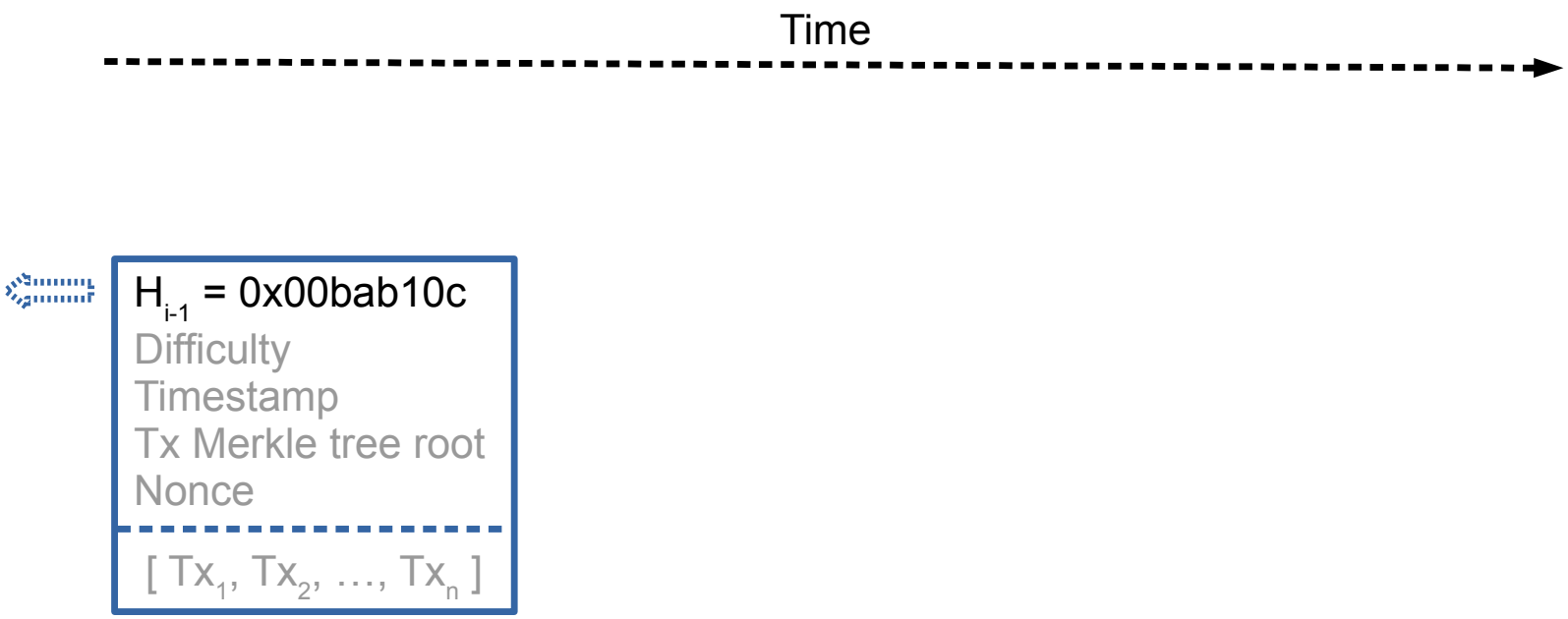
The blockchain data structure

Ledger of transactions
Transactions are encapsulated in blocks
Blocks provide probabilistic immutability
Reverse linked list of blocks

TL;DR blockchain

The blockchain data structure

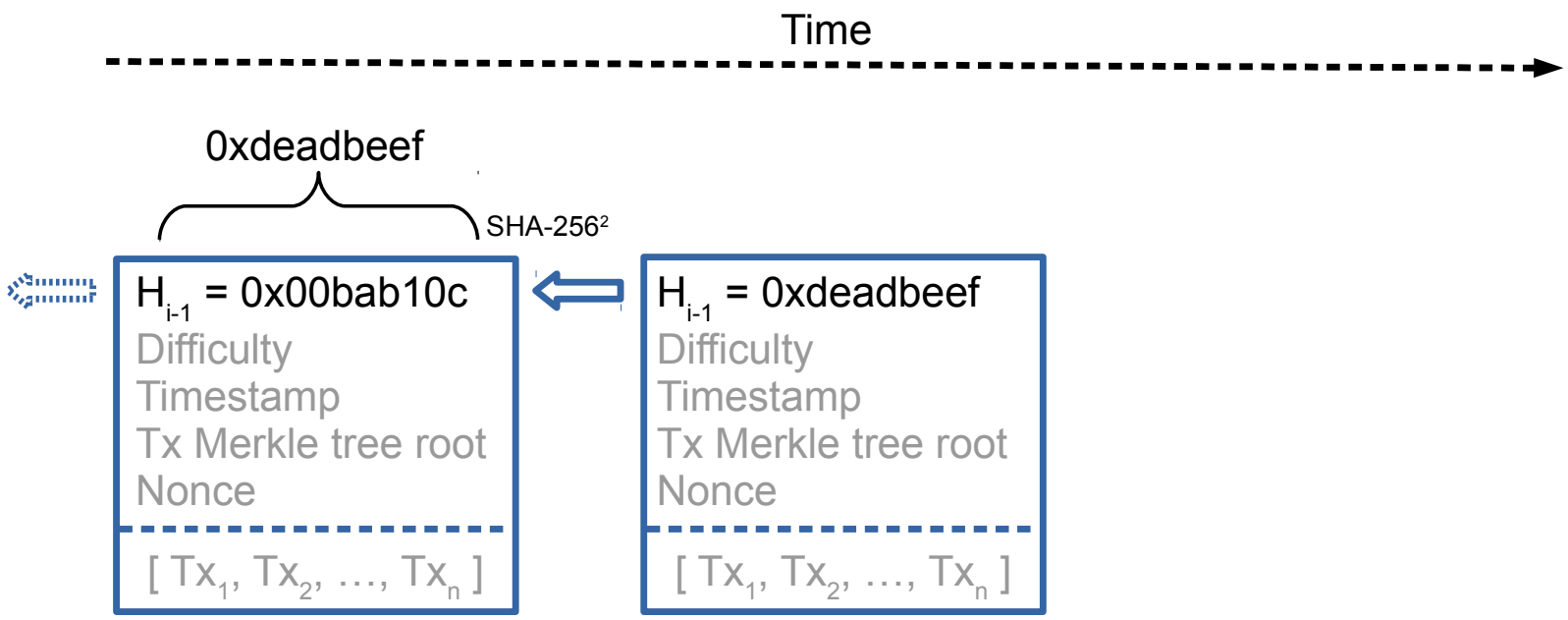
Ledger of transactions
Transactions are encapsulated in blocks
Blocks provide probabilistic immutability
Reverse linked list of blocks



TL;DR blockchain

The blockchain data structure

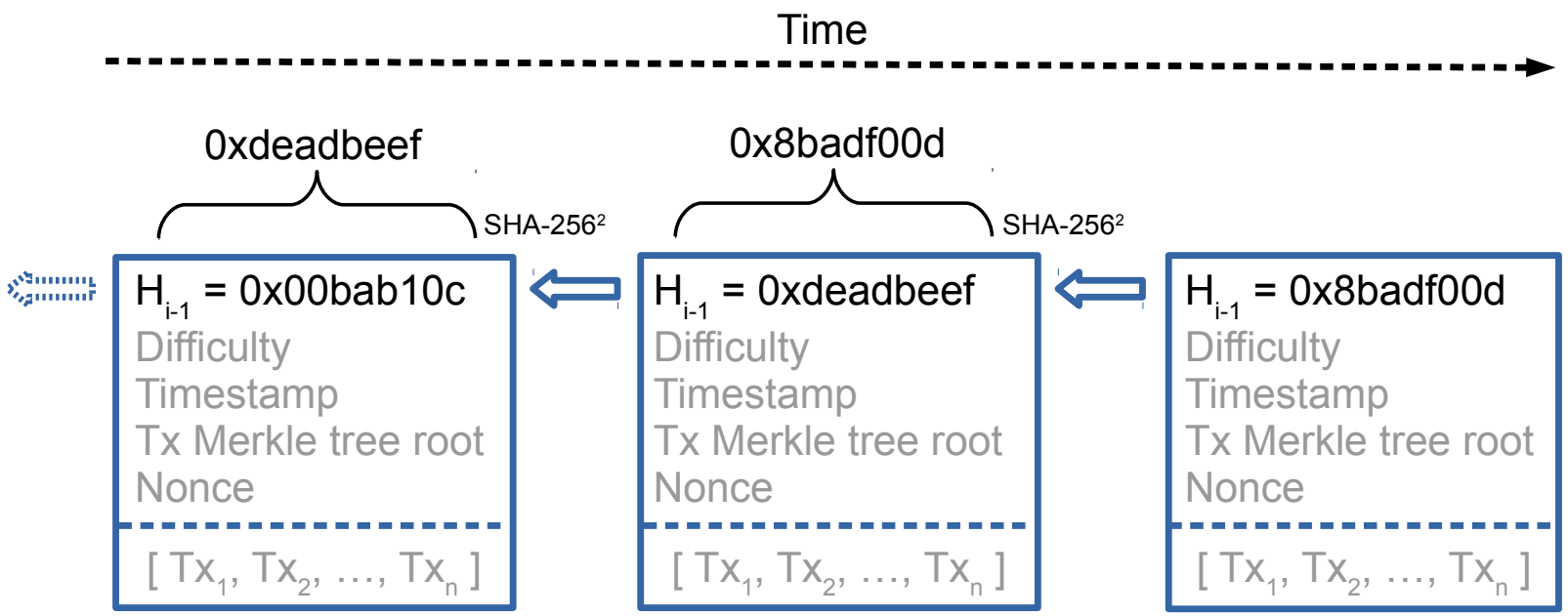
Ledger of transactions
Transactions are encapsulated in blocks
Blocks provide probabilistic immutability
Reverse linked list of blocks



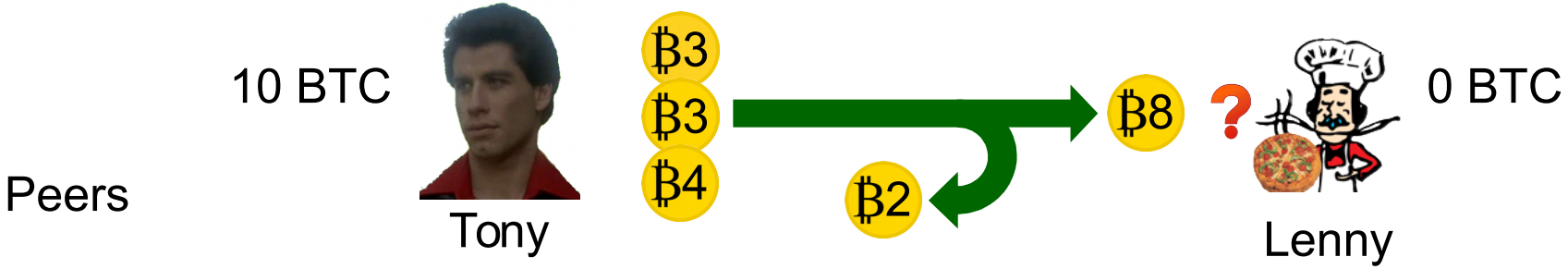
TL;DR blockchain

The blockchain data structure

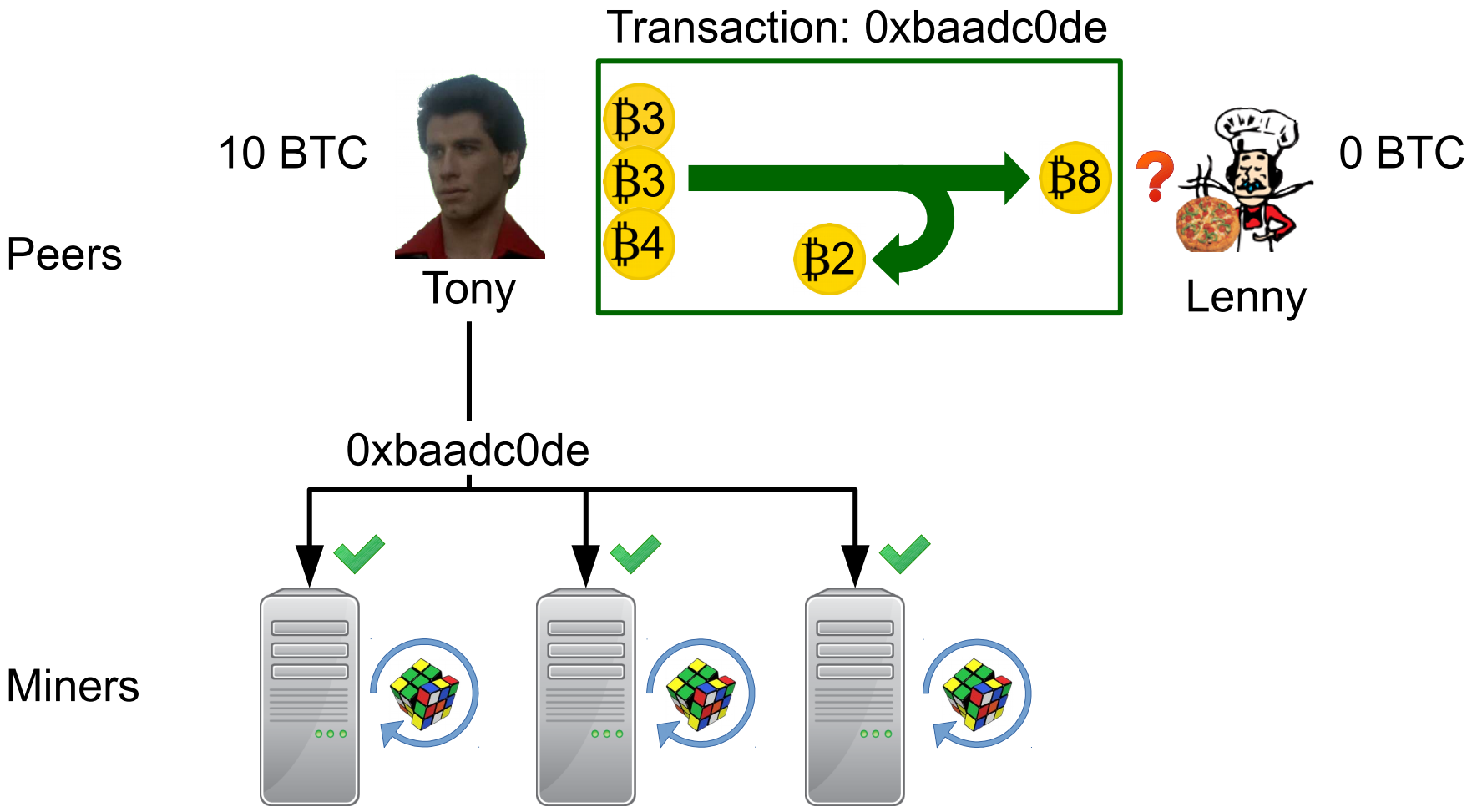
Ledger of transactions
Transactions are encapsulated in blocks
Blocks provide probabilistic immutability
Reverse linked list of blocks



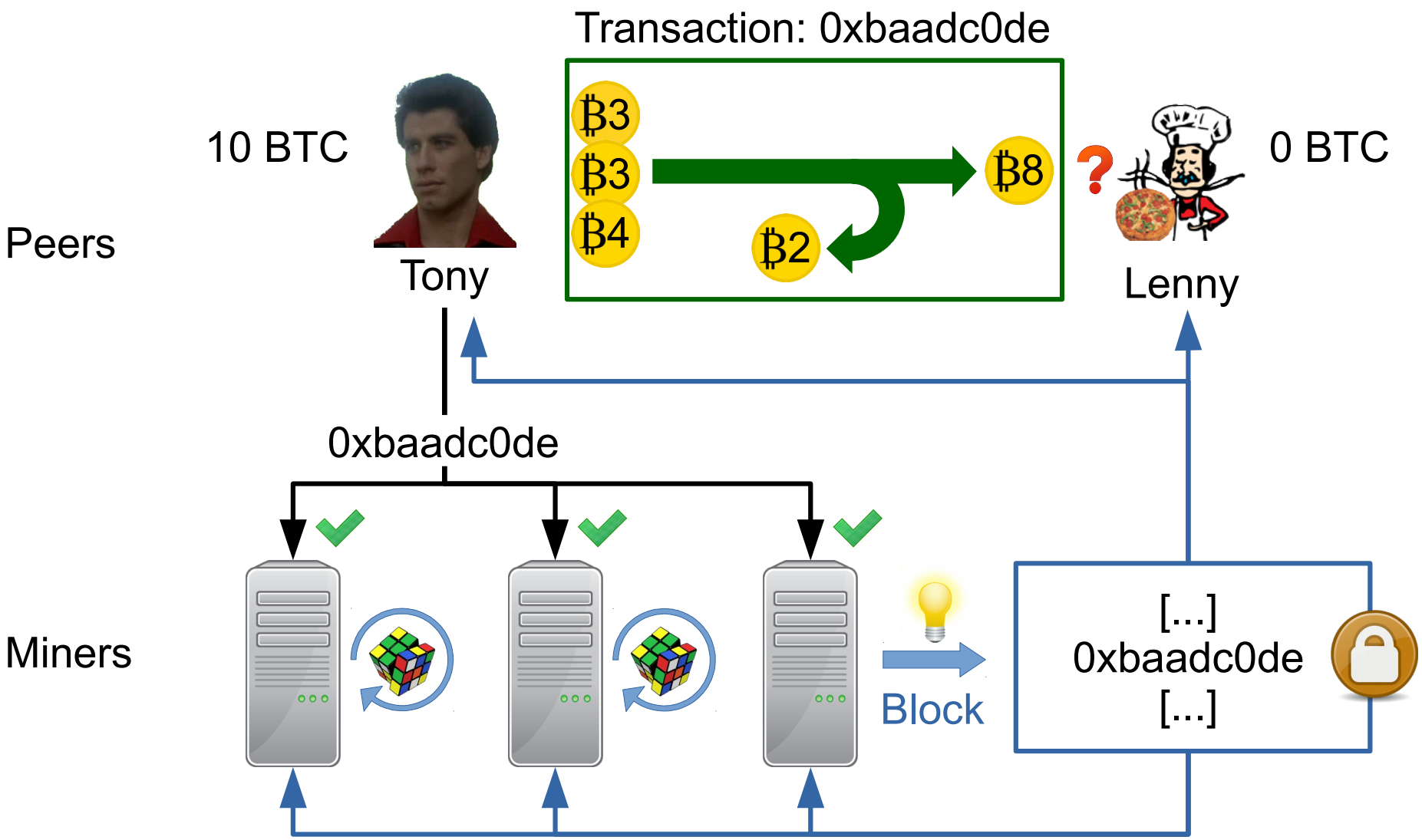
Making a transaction



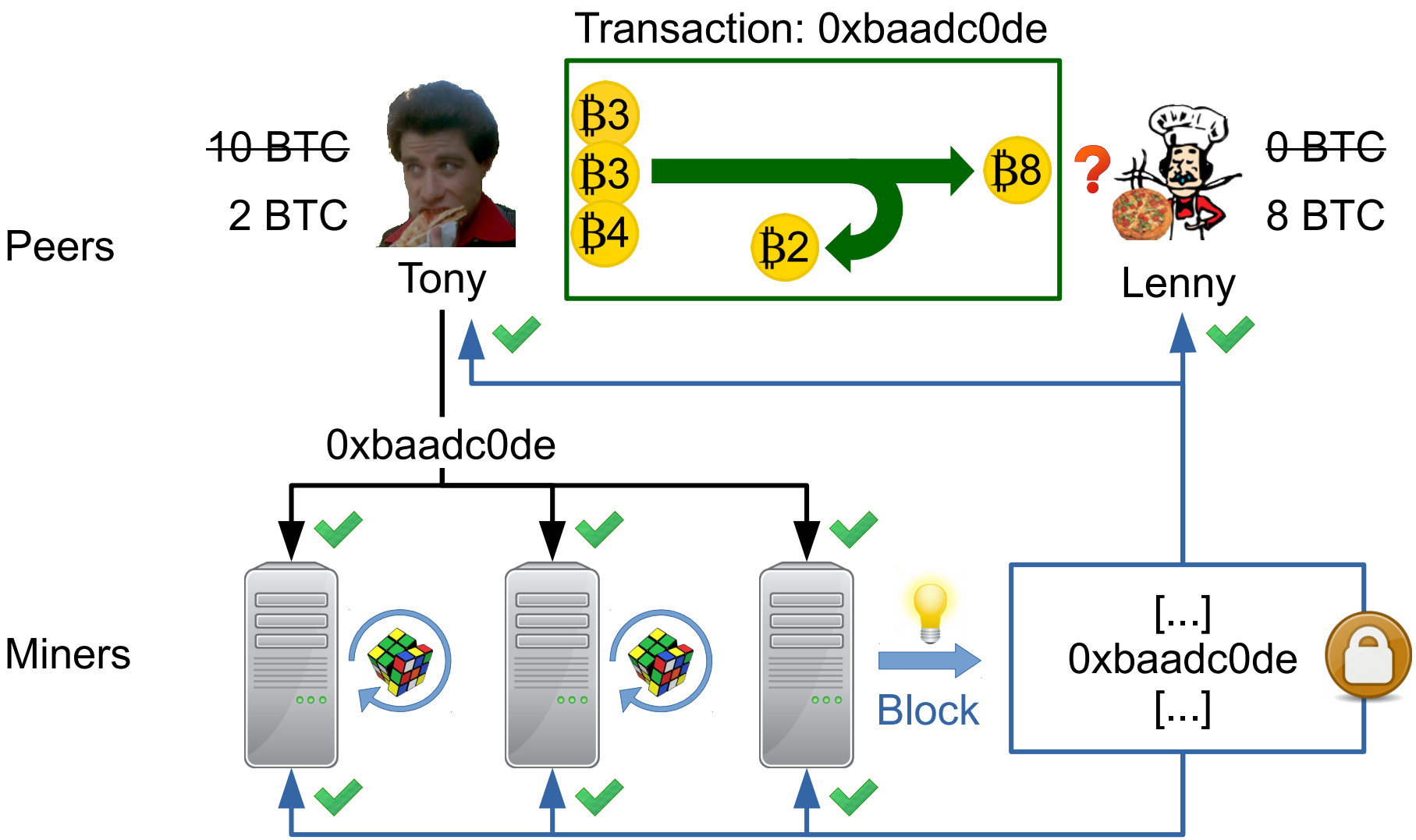
Making a transaction



Making a transaction



Making a transaction



Transactions

Valid transaction:

- Spend $\geqslant$ Receive
- The money exists, and is not spent yet

Money moves from transaction to transaction

Transactions

Valid transaction:

- Spend $\geqslant$ Receive
- The money exists, and is not spent yet

Money moves from transaction to transaction

	List of inputs		List of outputs
0	previous Tx + output index, proof ownership	0	value, ownership challenge
	[...]		[...]
n-1	previous Tx + output index, proof ownership	m-1	value, ownership challenge

Verifying transactions

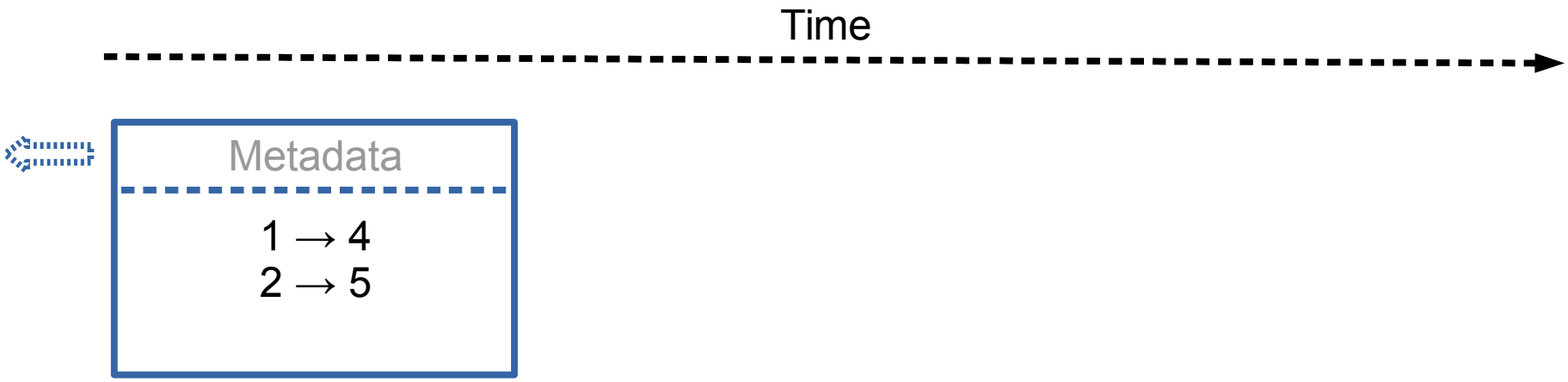
UTxO set

Set of Unspent Transaction Outputs

Verifying transactions

UTxO set

Set of Unspent Transaction Outputs

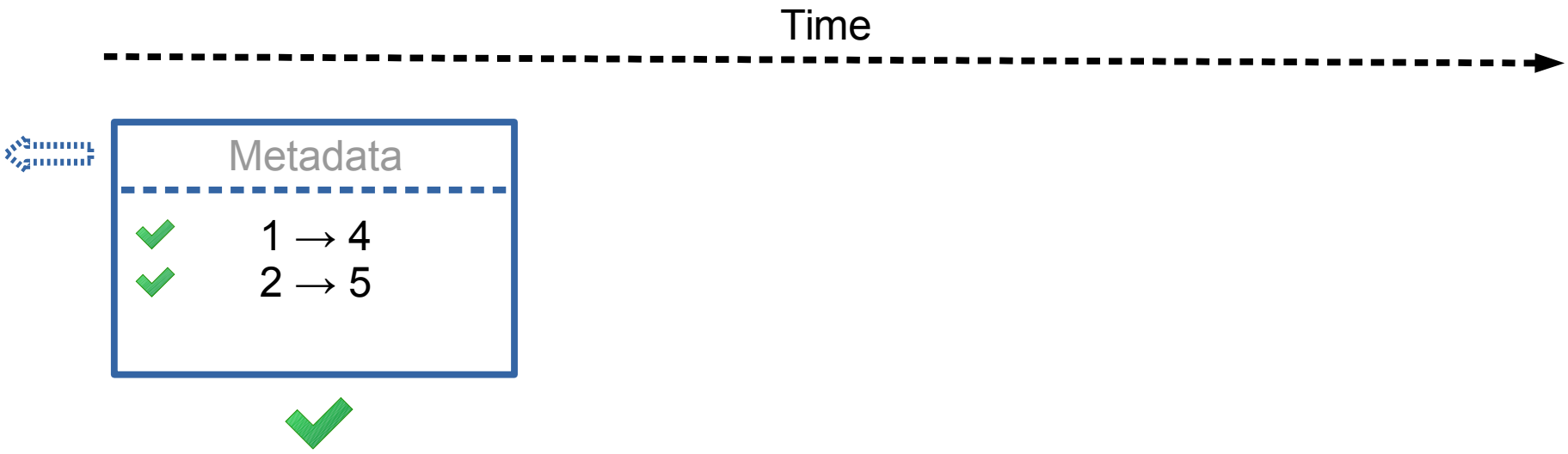


UTxO set = { 1, 2, 3 }

Verifying transactions

UTxO set

Set of Unspent Transaction Outputs

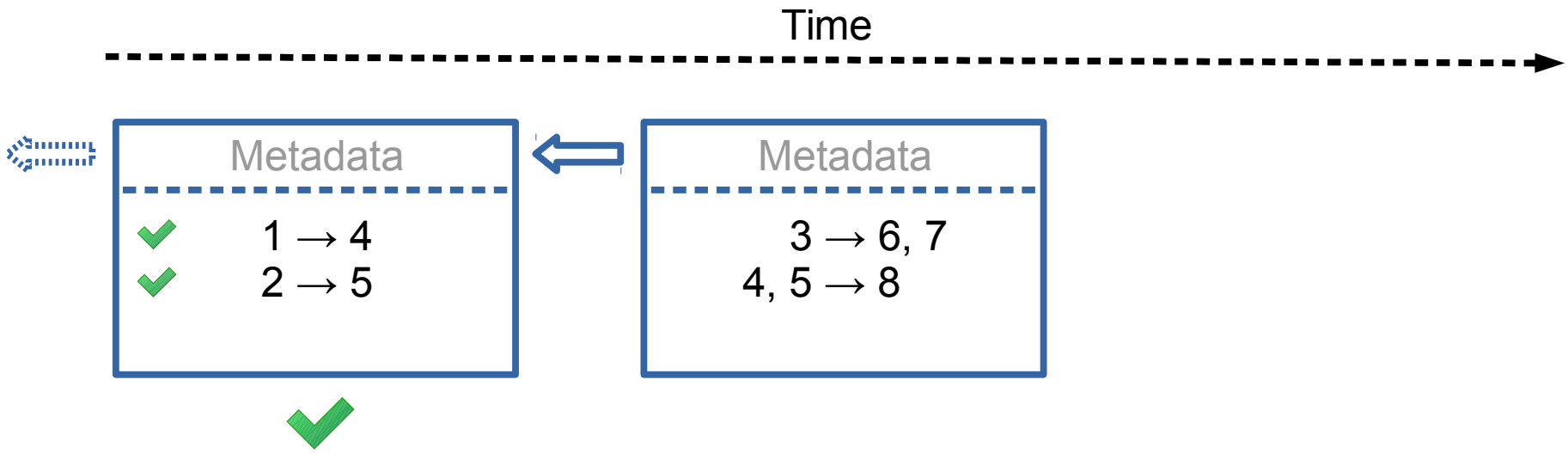


UTxO set = { ~~1~~, ~~2~~, 3, 4, 5 }

Verifying transactions

UTxO set

Set of Unspent Transaction Outputs

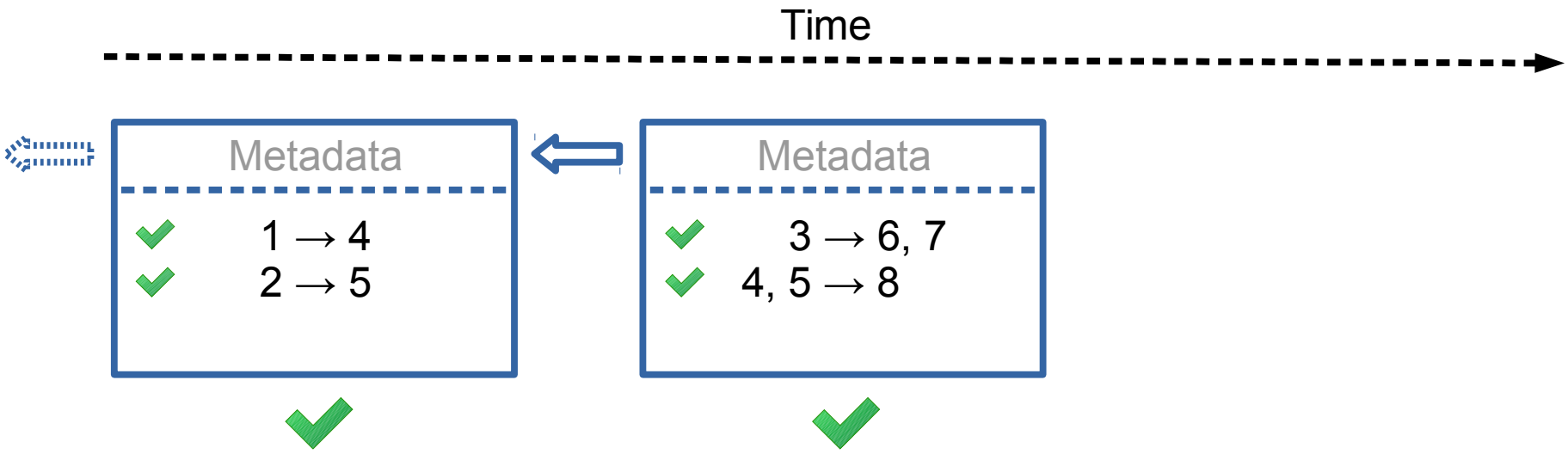


UTxO set = { 3, 4, 5 }

Verifying transactions

UTxO set

Set of Unspent Transaction Outputs

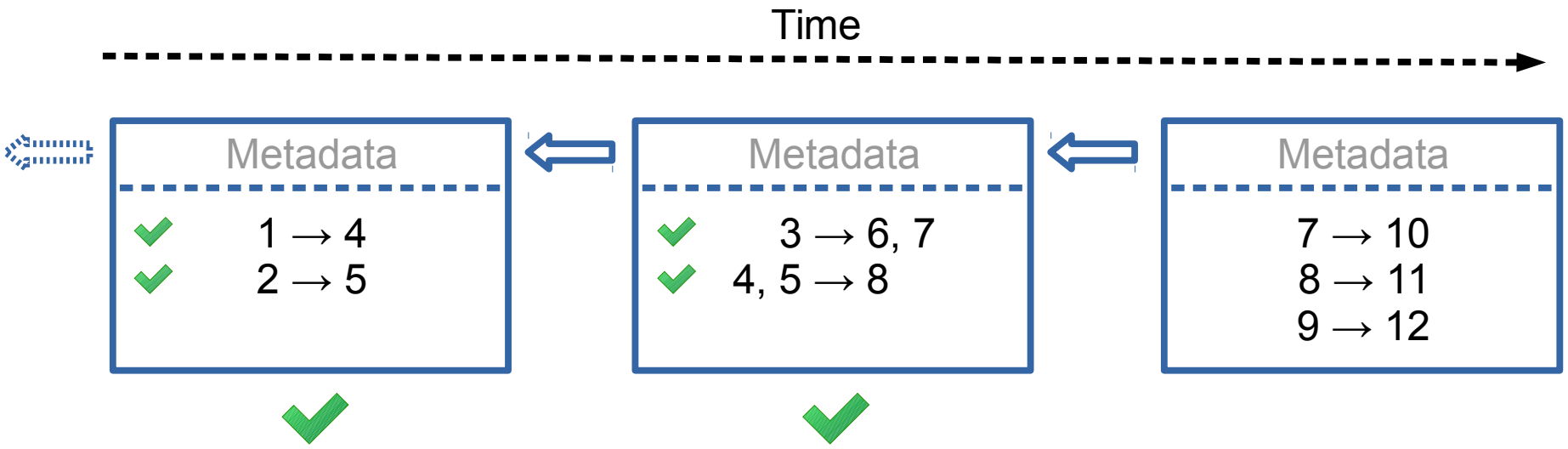


UTxO set = { ~~3~~, 4, ~~5~~, 6, 7, 8 }

Verifying transactions

UTxO set

Set of Unspent Transaction Outputs

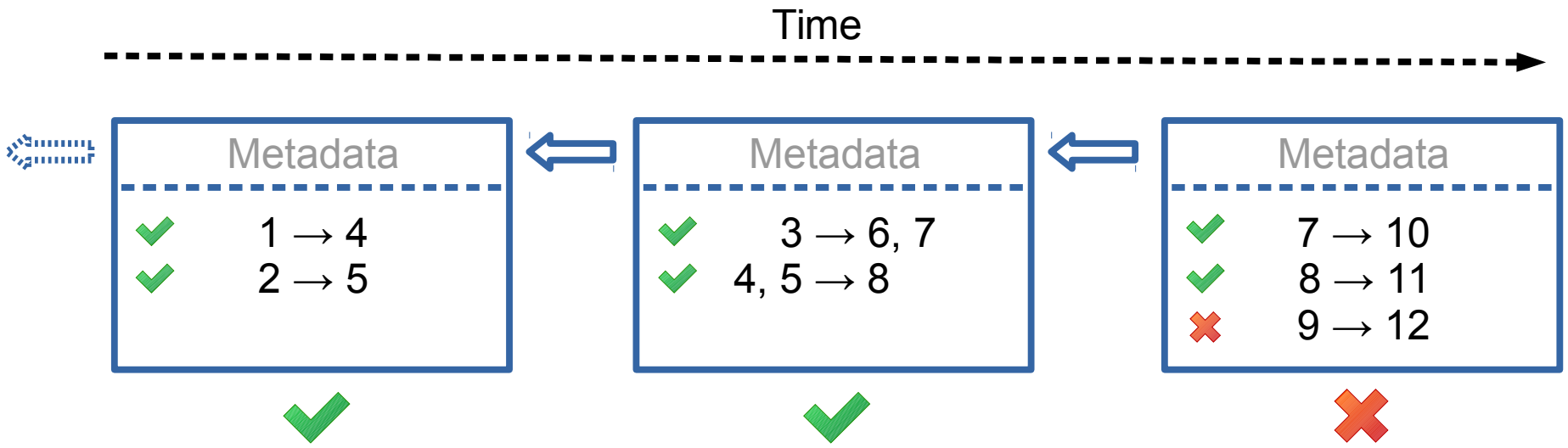


UTxO set = { 6, 7, 8 }

Verifying transactions

UTxO set

Set of Unspent Transaction Outputs

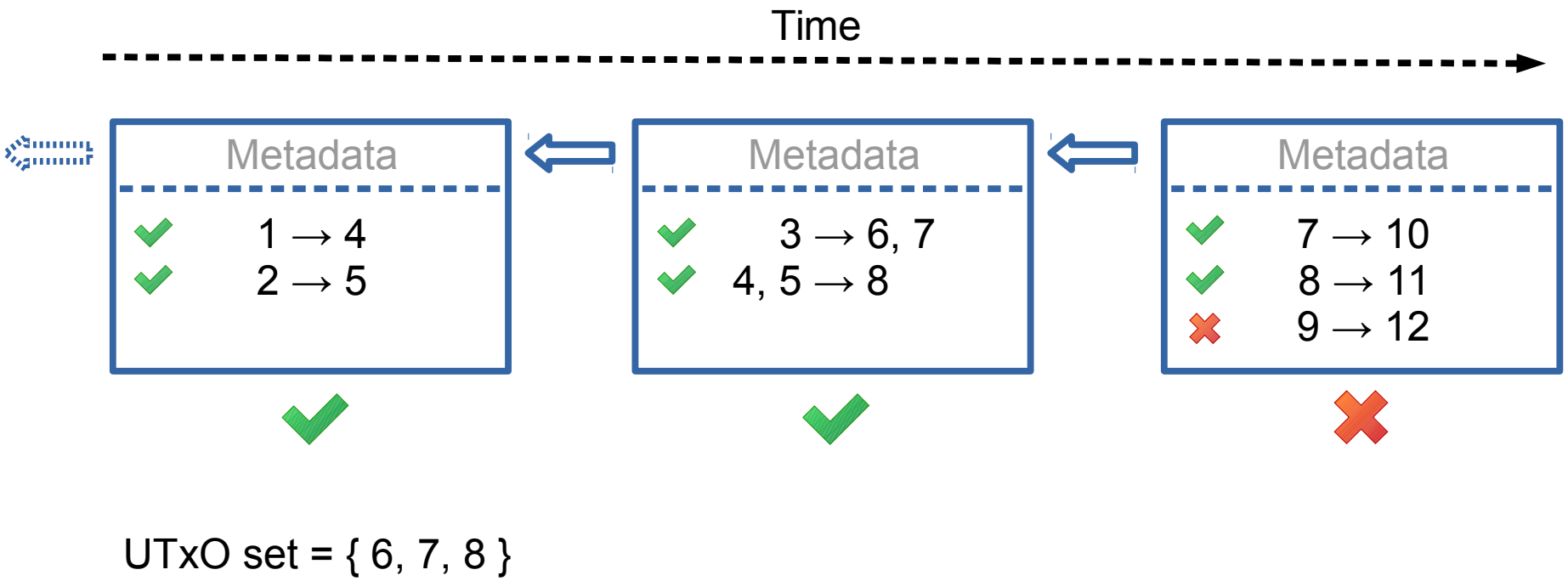


UTxO set = { 6, 7, 8 }

Verifying transactions

UTxO set

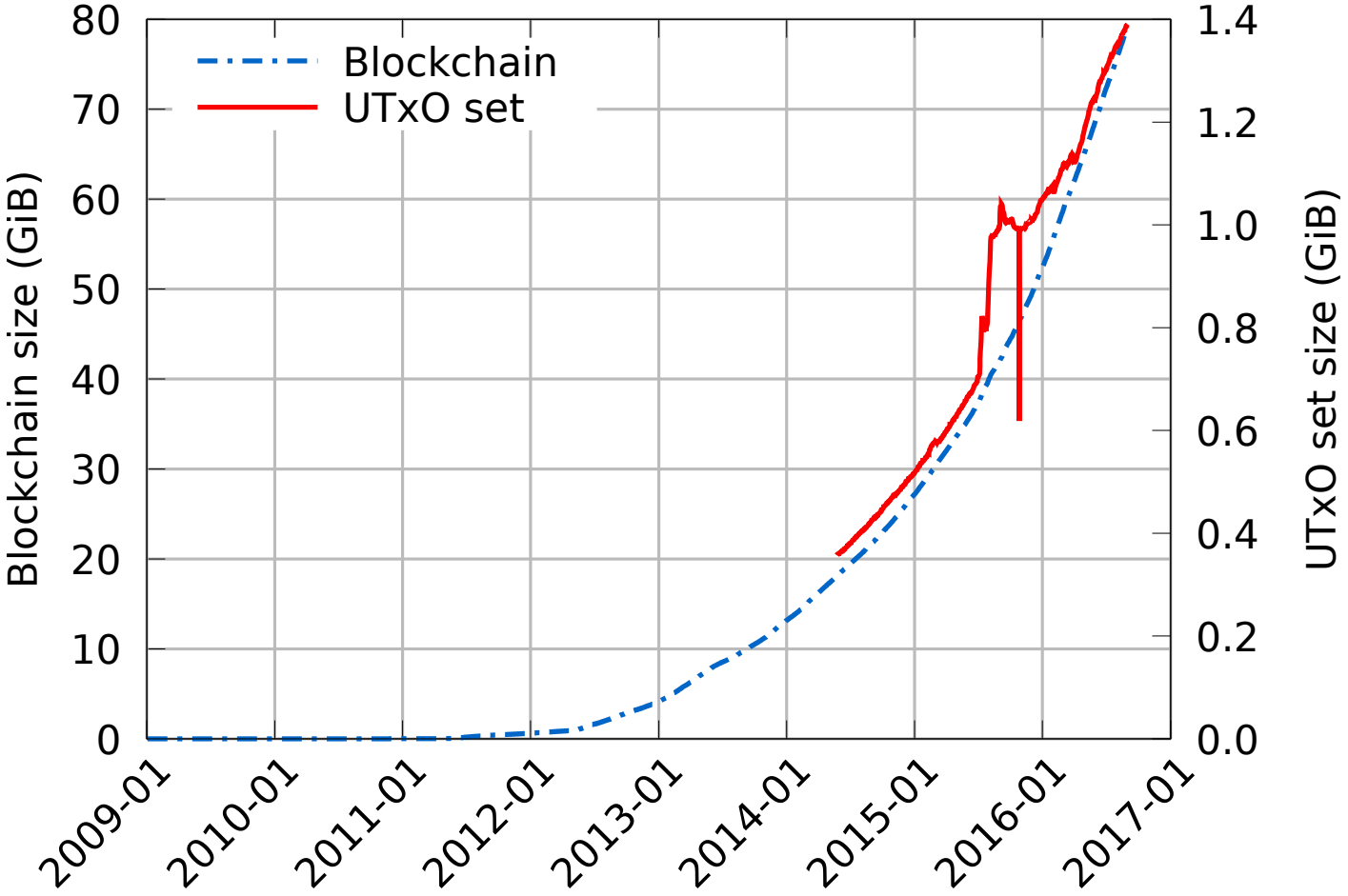
Set of Unspent Transaction Outputs



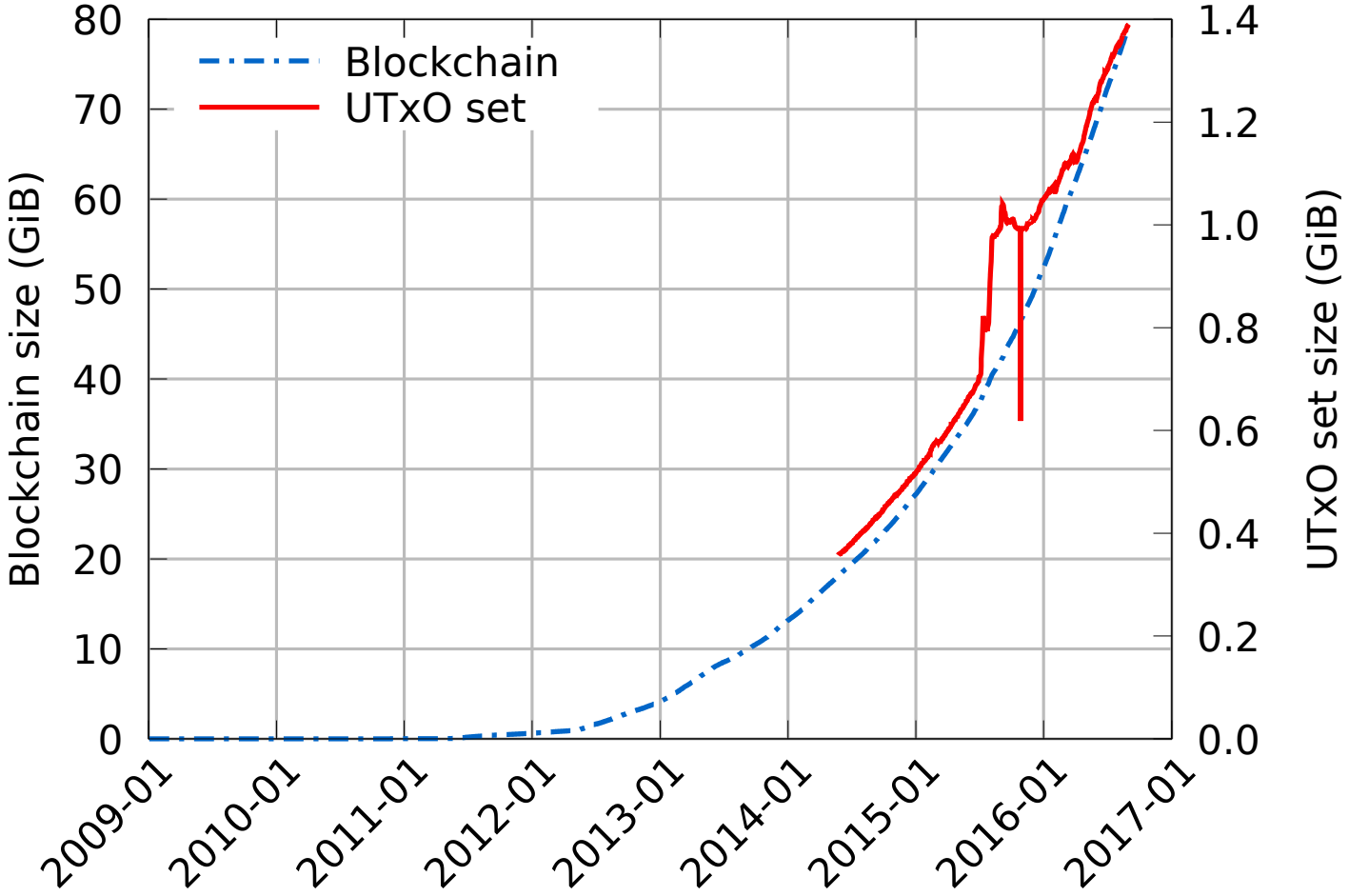
Problem 1: time

Building the UTxO is long, can't skip-ahead

Not getting any better



Not getting any better



Problem 2: resources

Building the UTxO is bandwidth and storage heavy

Problem and solution

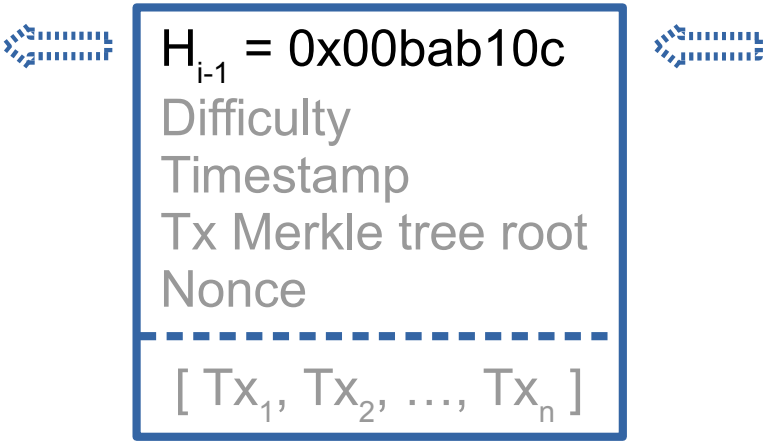
Fully secured node = long bootstrap, high storage, high bandwidth

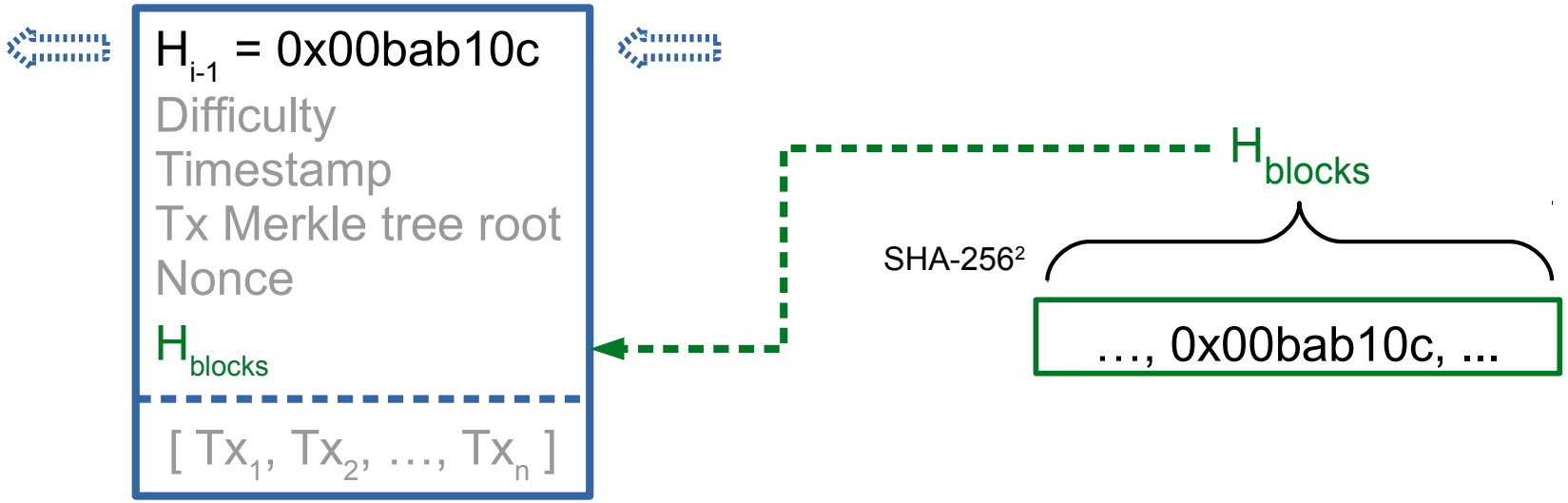
Problem and solution

Fully secured node = long bootstrap, high storage, high bandwidth

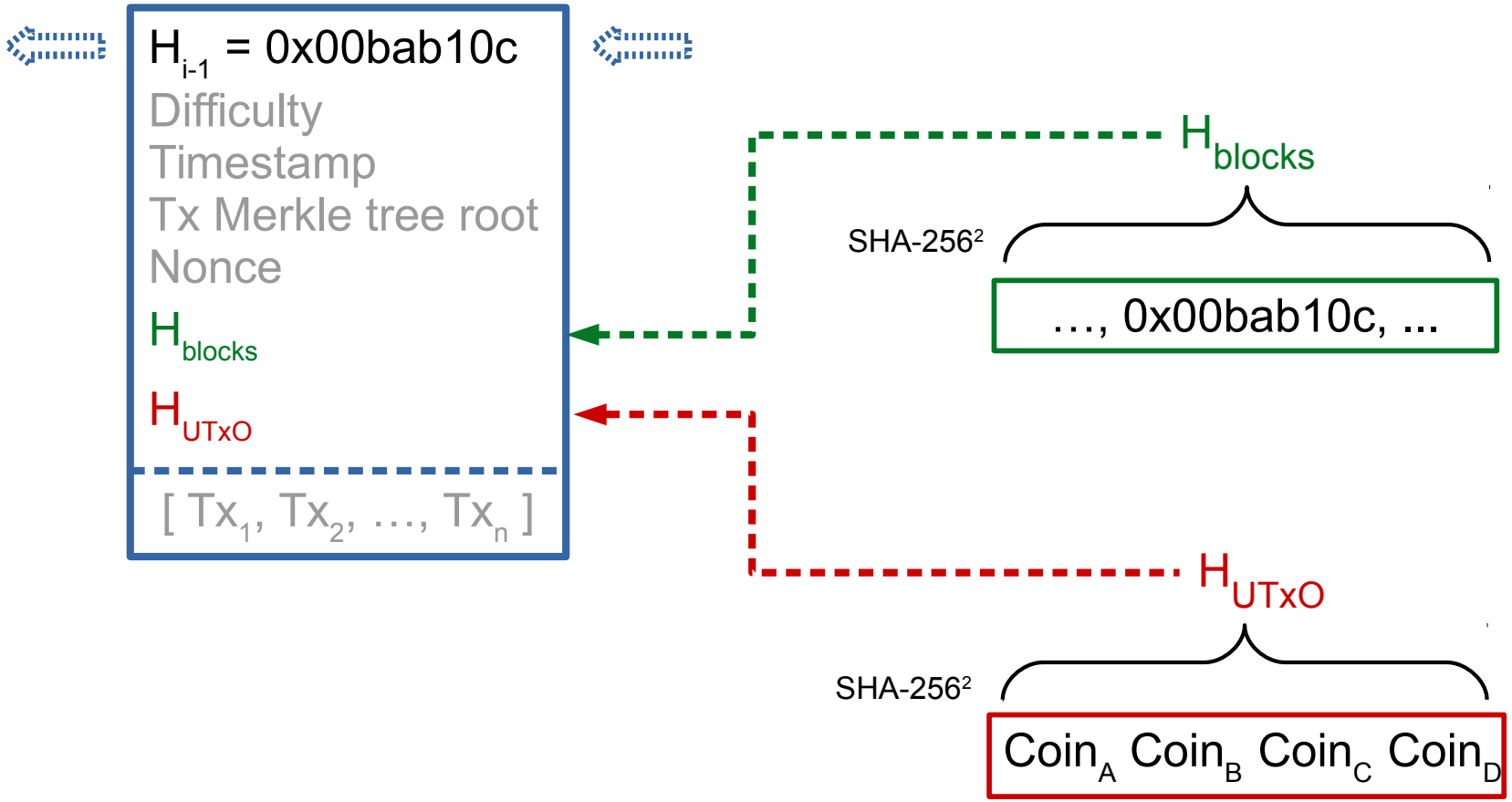
Directly download the UTxO

How do you trust the downloaded set?
Puts its hash in the blockchain


H_{i-1} = 0x00bab10c
Difficulty
Timestamp
Tx Merkle tree root
Nonce
[Tx_1, Tx_2, …, Tx_n]

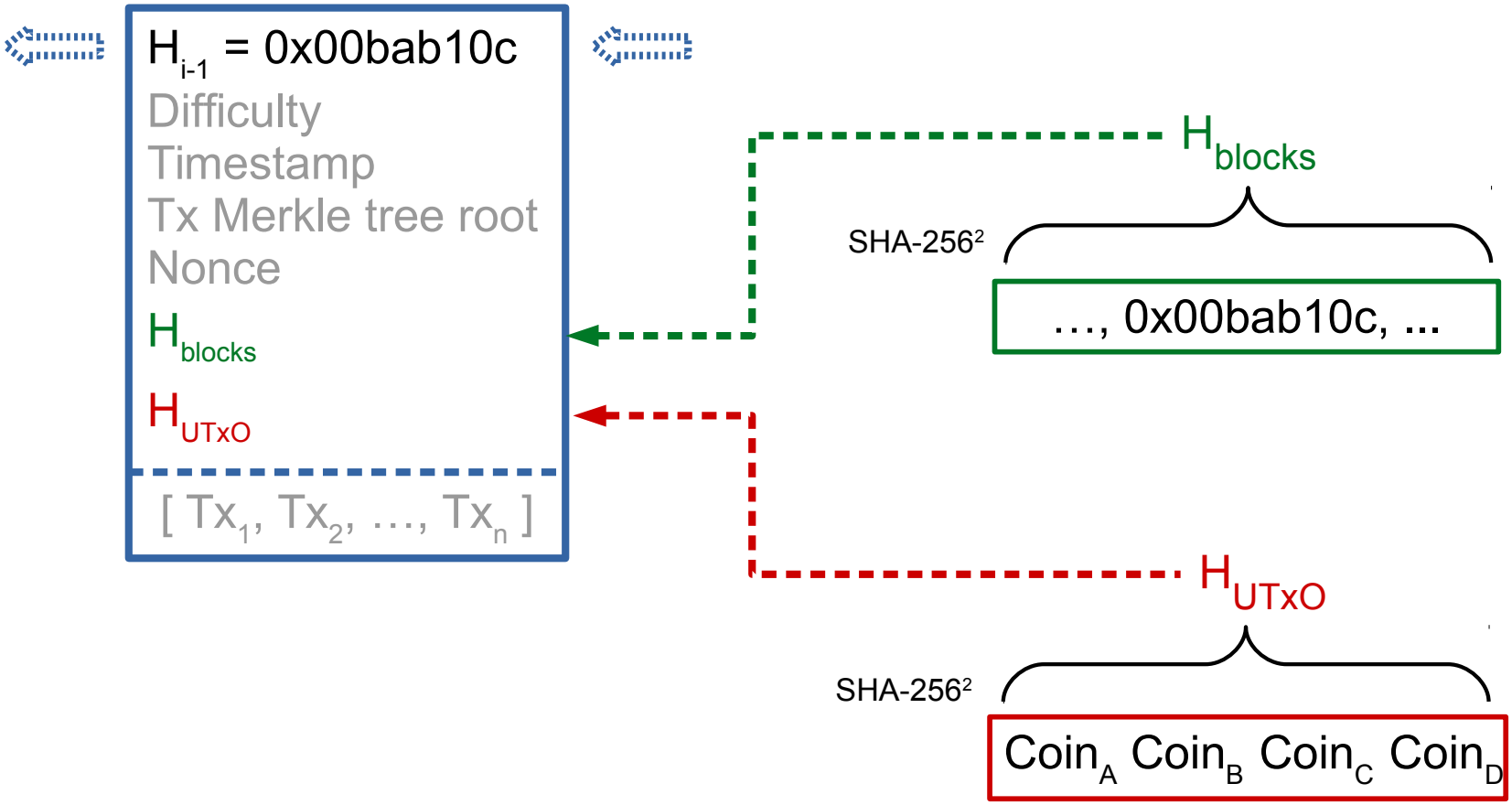


H_{blocks} to dig in the history



H_{blocks} to dig in the history

H_{UTxO} to verify transactions



H_{blocks} to dig in the history

H_{UTxO} to verify transactions

UTxO updates?

Need to "trust" the block

Reducing time and resource needs

Make the UTxO downloadable: enable skip-aheads!

Reducing time and resource needs

Make the UTxO ~~downloadable~~ queriable: skip-aheads and lower costs!

Broaden the spectrum of possible verification strategies
Adaptive strategies for different storage/bandwidth/security trade-offs

Reducing time and resource needs

Make the UTxO ~~downloadable~~ queriable: skip-aheads and lower costs!

Broaden the spectrum of possible verification strategies
Adaptive strategies for different storage/bandwidth/security trade-offs

With a partial UTxO, a client can:

- Verify new transactions
- Verify some transactions in a block
- Verify every transactions in a block (same as full node)

Verifying new transactions

"Show me the money"

Prevents:

- Spending fake coins

Possible attacks:

- Double spends

Verifying some transactions in a block

Verify transactions to you

Prevents:

- Spending fake coins
- Double spends
- Cheap attacks

Possible attacks:

- Blocks with valid transactions to you, invalid to others

Verifying every transactions in a block

Similar security level as full nodes

Prevents:

- Spending fake coins
- Double spends
- Blocks with valid transactions to you, invalid to others
- Basically everything

Vulnerable against:

- $\emptyset$

Partial UTxO

With a partial UTxO, a client can:

- ~~Verify new transactions~~
- ~~Verify some transactions in a block~~
- Verify every transactions in a block (same as full node)

Partial UTxO

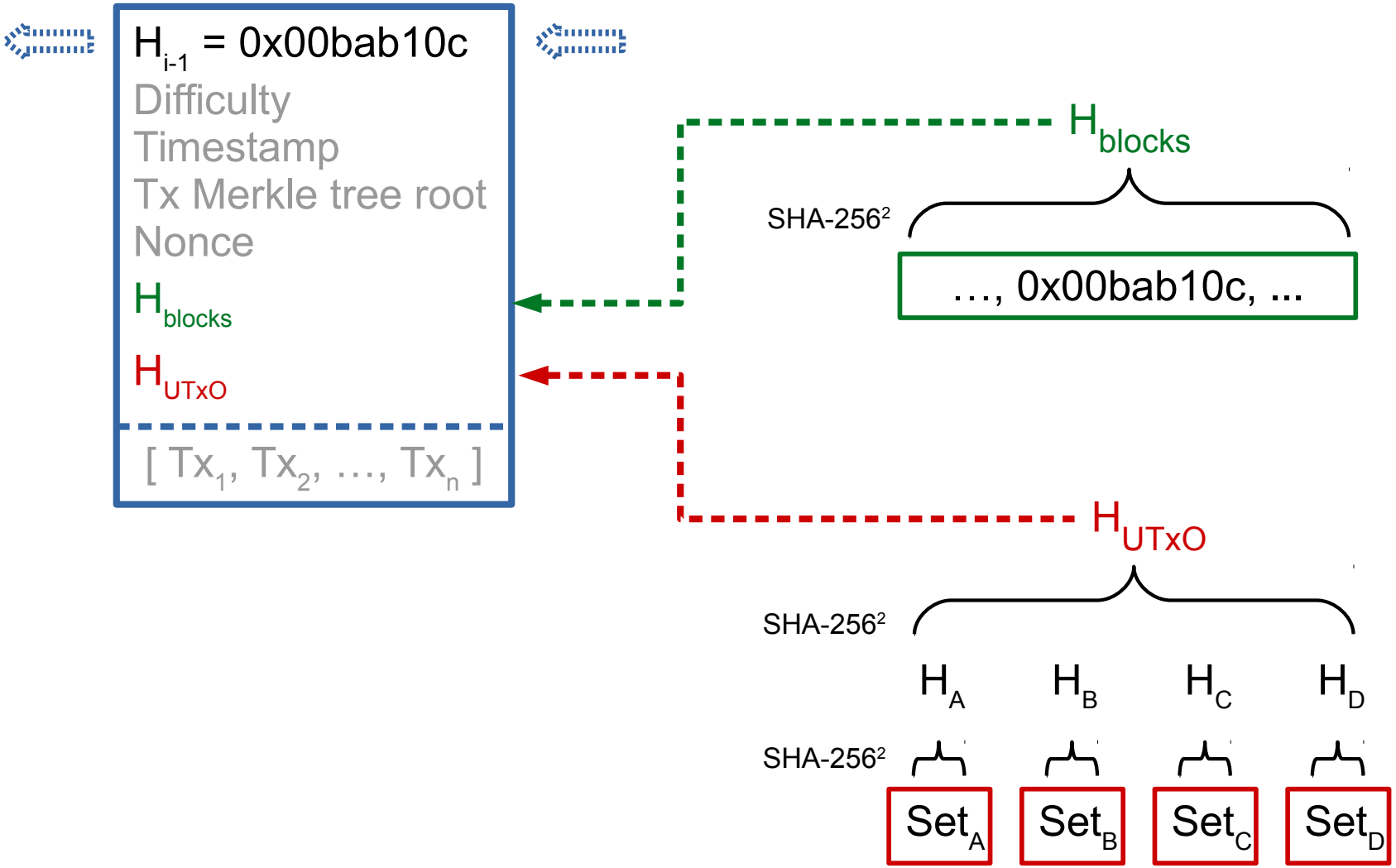
With a partial UTxO, a client can:

- ~~Verify new transactions~~
- ~~Verify some transactions in a block~~
- Verify every transactions in a block (same as full node)

Splitting strategies:

- Sharding: query subsets of the UTxO (our proposal)
- Merkle tree: query individual coins in the UTxO

UTxO - Sharding



UTxO - Sharding

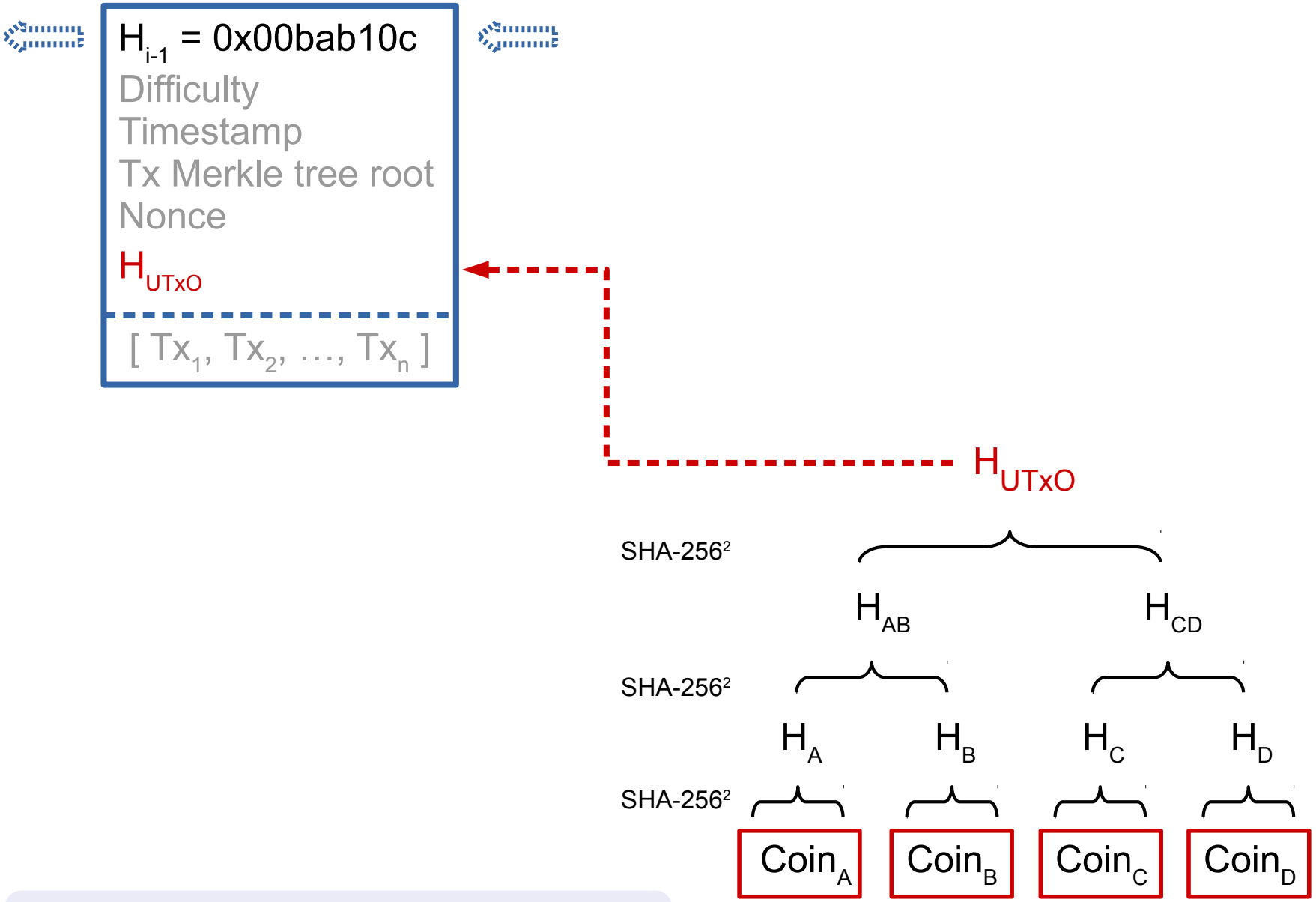
UTxO set size = 1.5 GiB
Sharding policy: random, first k bits, e.g., k = 16
Number of shards = 2^{16}
Shard size = 1.5 GiB $\div$ 2^{16} $\approx$ 25 KiB

Server storage overhead: $2^{16} \times 32\,\mathrm{B} = 2\,\mathrm{MiB}$

Typical block verification:

- 2000 inputs minimum
- With random sharding, needs 2000 shards
- Client bandwidth consumption: $2000 \times 25\,\mathrm{KiB} \approx 49\,\mathrm{MiB} + 2\,\mathrm{MiB}$

High bandwidth



Author: Andrew Miller (UIUC)

UTxO - Merkle tree

Number of items in the UTxO $= 15\,\mathrm{M}$
Number of nodes in the tree $= 2 \times 15\,\mathrm{M} = 30\,\mathrm{M}$
Number of hashes to reconstruct the merkle root: $\log_2(30\,\mathrm{M}) \approx 25$

Server storage overhead: $30\,\mathrm{M} \times 32\,\mathrm{B} \approx 915\,\mathrm{MiB}$

Typical block verification:

- 2000 inputs minimum
- Needs 2000 proof of existence in the tree
- Client bandwidth consumption: $2000 \times 25 \times 32\,\mathrm{B} \approx 1.5\,\mathrm{MiB}$

Balancing the tree, storage overhead on servers

Summary of solutions

	Client bandwidth	Server storage overhead
Full UTxO	1.5 GiB	0
Sharded UTxO	49+2 MiB	2 MiB
Merkle tree UTxO	1.5 MiB	915 MiB

Sharding is the most-balanced, but not ideal

The good, the bad and the unicorn

Decoupled storage
Quick client bootstrap
Light storage client

The good, the bad and the unicorn

Decoupled storage
Quick client bootstrap
Light storage client

High bandwidth clients?
Light clients are not as safe as full nodes
Low overhead for full nodes

The good, the bad and the unicorn

Decoupled storage
Quick client bootstrap
Light storage client

High bandwidth clients?
Light clients are not as safe as full nodes
Low overhead for full nodes

Still looking for the optimal trade-off

Any questions?