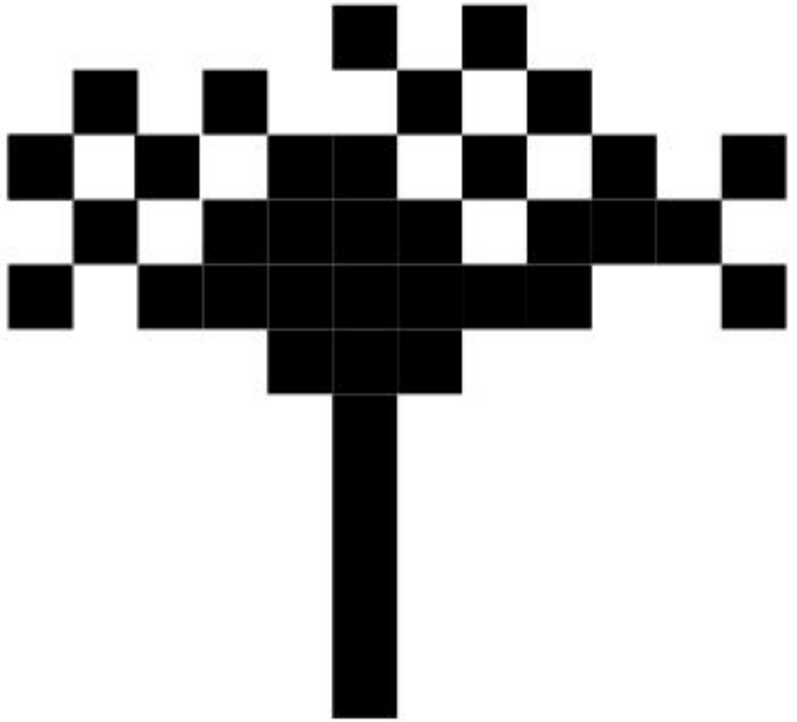


Utreexo - What is it?

What it is and how it works

Seoul Bitcoin Meetup

Where I got started

- Started by Ruben Somsen in 2014

Seoul Bitcoin Meetup

Where I got started

- Started by Ruben Somsen in 2014
- I joined in 2017

Seoul Bitcoin Meetup

Where I got started

- Started by Ruben Somsen in 2014
- I joined in 2017
- We hold in-person meetings on technical topics Bitcoin

Seoul Bitcoin Meetup

Where I got started

About myself

Calvin Kim

- Joined the meetup in 2017

About myself

Calvin Kim

- Joined the meetup in 2017
- Started doing Bitcoin development in 2019

About myself

Calvin Kim

- Joined the meetup in 2017
- Started doing Bitcoin development in 2019
- Was funded by Bitmex since 2020.

About myself

Calvin Kim

- Joined the meetup in 2017
- Started doing Bitcoin development in 2019
- Was funded by Bitmex since 2020.
- Funded by HRF since this May.

About myself

Calvin Kim

- Joined the meetup in 2017
- Started doing Bitcoin development in 2019
- Was funded by Bitmex since 2020.
- Funded by HRF since this May.
- Done Summer of Bitcoin mentorships from 2021 and onwards.

Utreexo

Agenda for today

How does Bitcoin work?

Utreexo

Agenda for today

How does Bitcoin work?

What is Utreexo?

Utreexo

Agenda for today

How does Bitcoin work?

What is Utreexo?

How does Utreexo work?

Utreexo

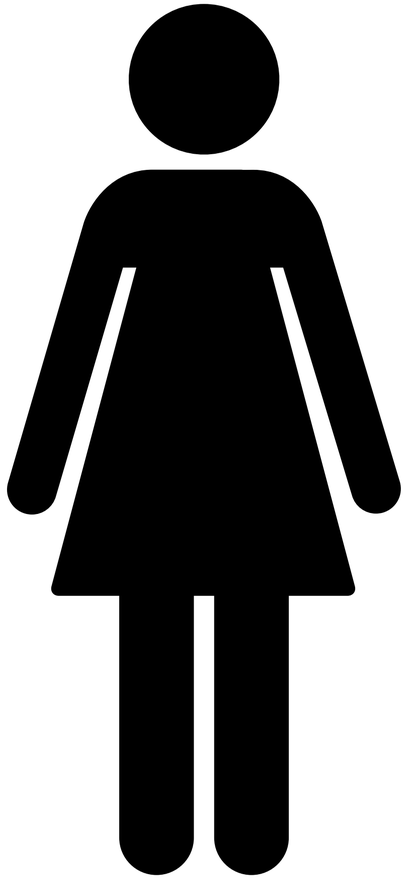
Agenda for today

Questions encouraged!

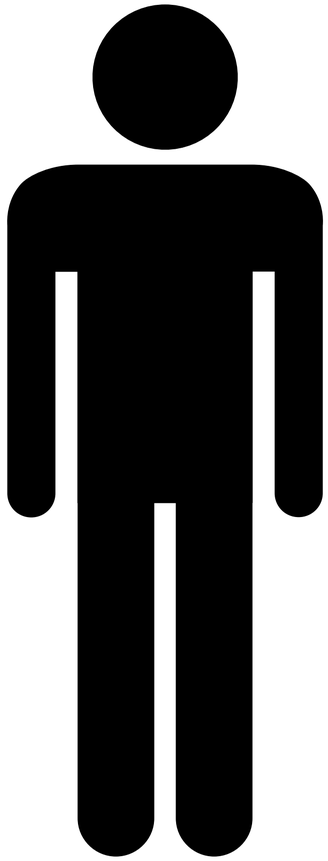
How does Bitcoin work?

Bartering with cash vs with Bitcoin

Barter with cash

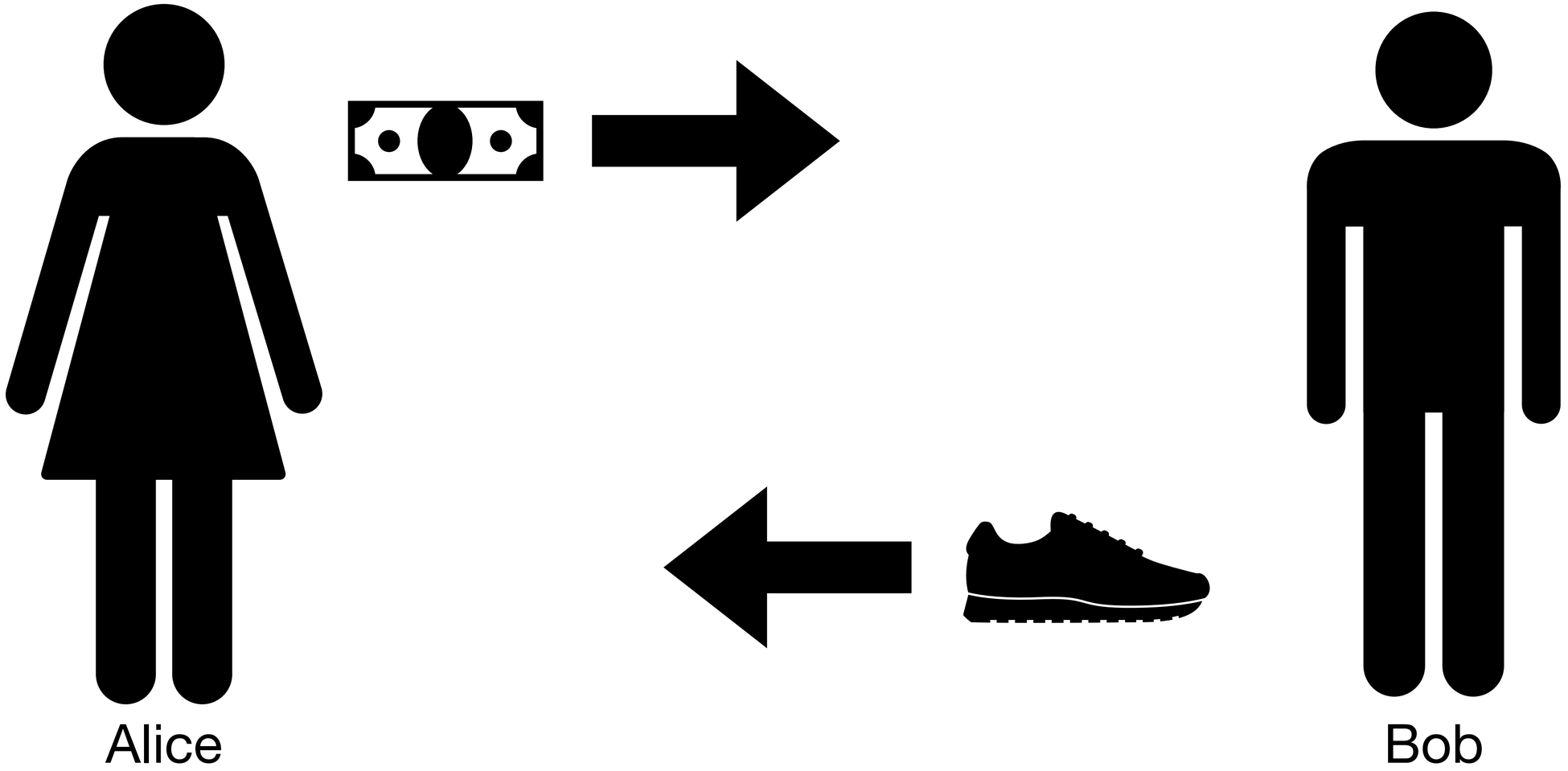


Alice

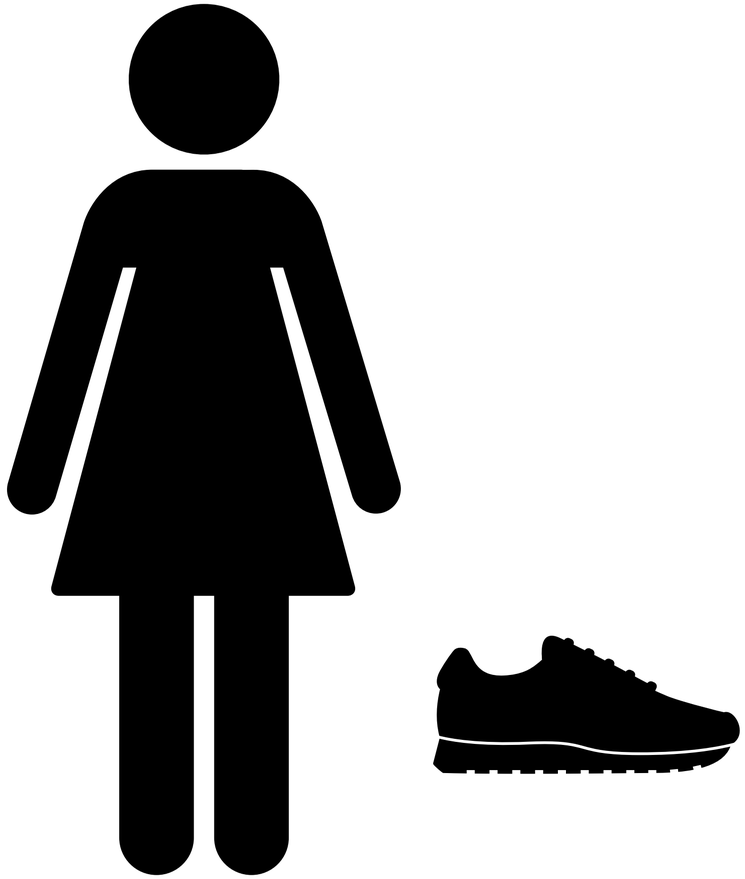


Bob

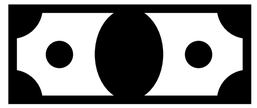
Barter with cash

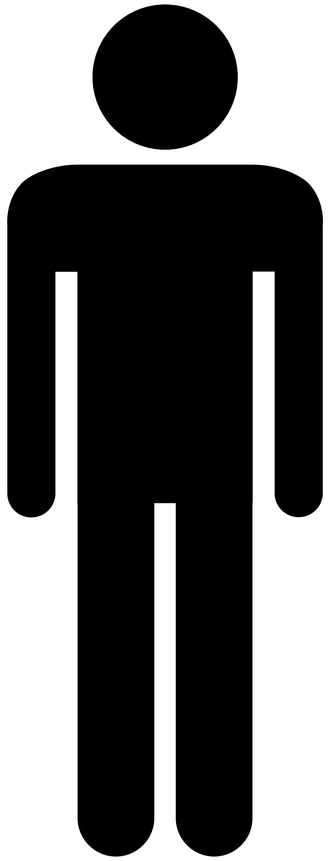


Barter with cash



Alice





Bob

Barter with cash

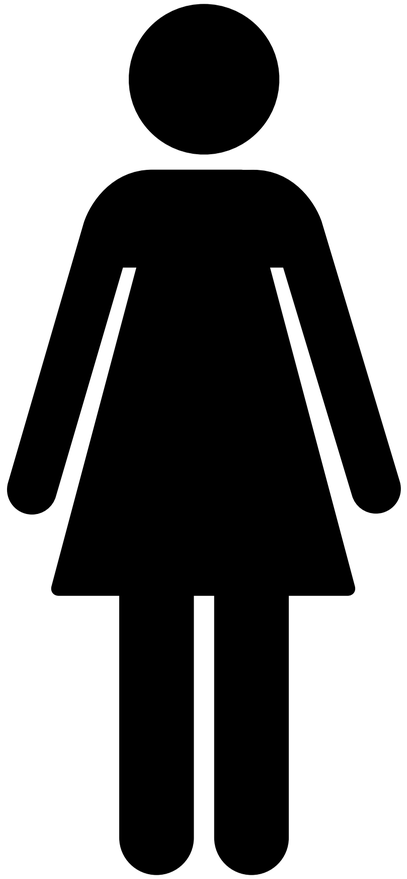
- Alice hands over the dollar
- Bob hands over the shoes

Barter with cash

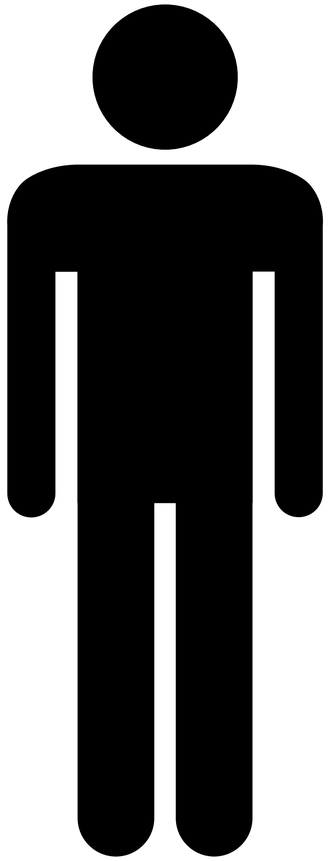
- Alice hands over the dollar
- Bob hands over the shoes
- Physical exchange has been made

Barter with Bitcoin

Barter with Bitcoin

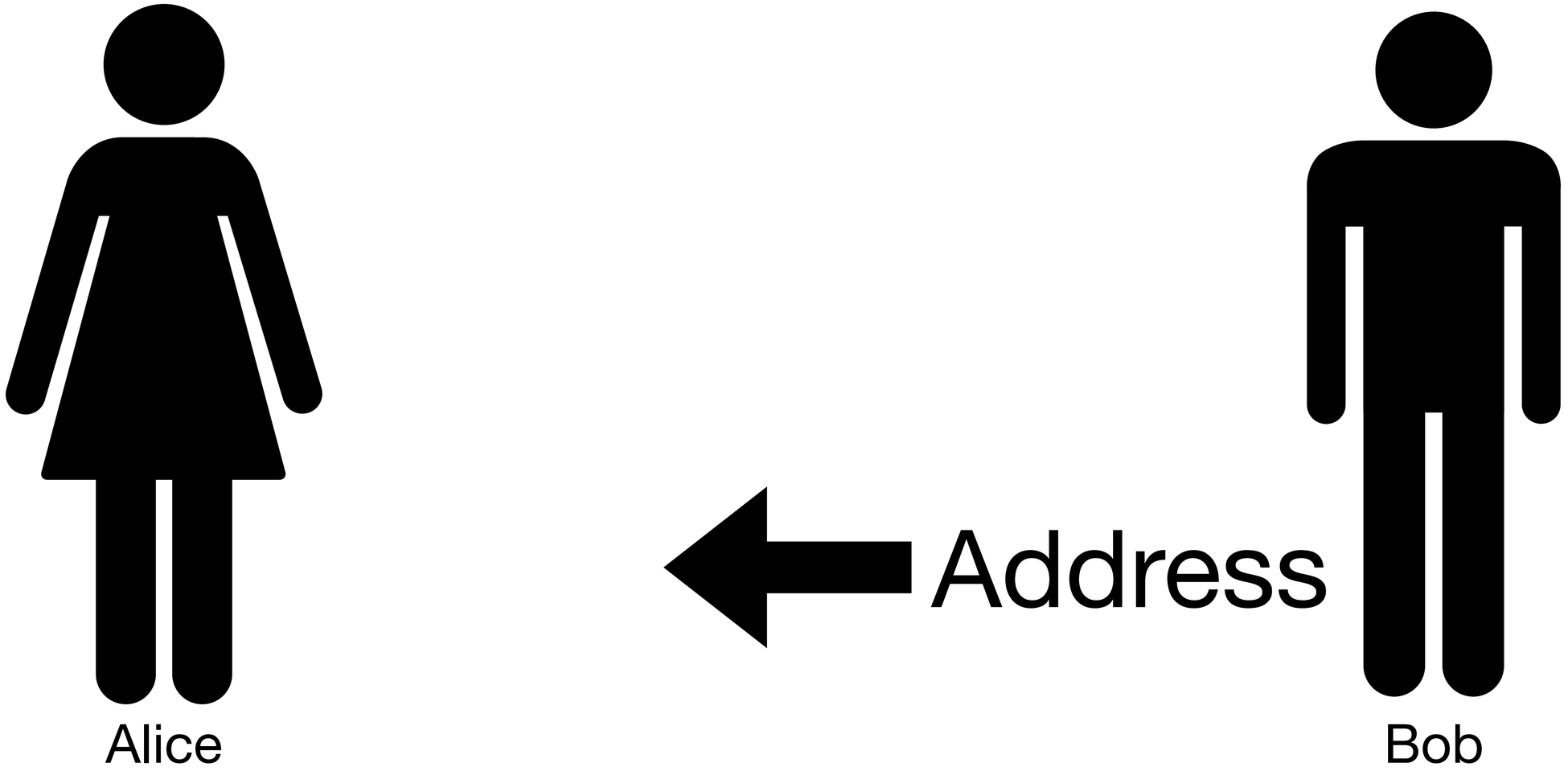


Alice

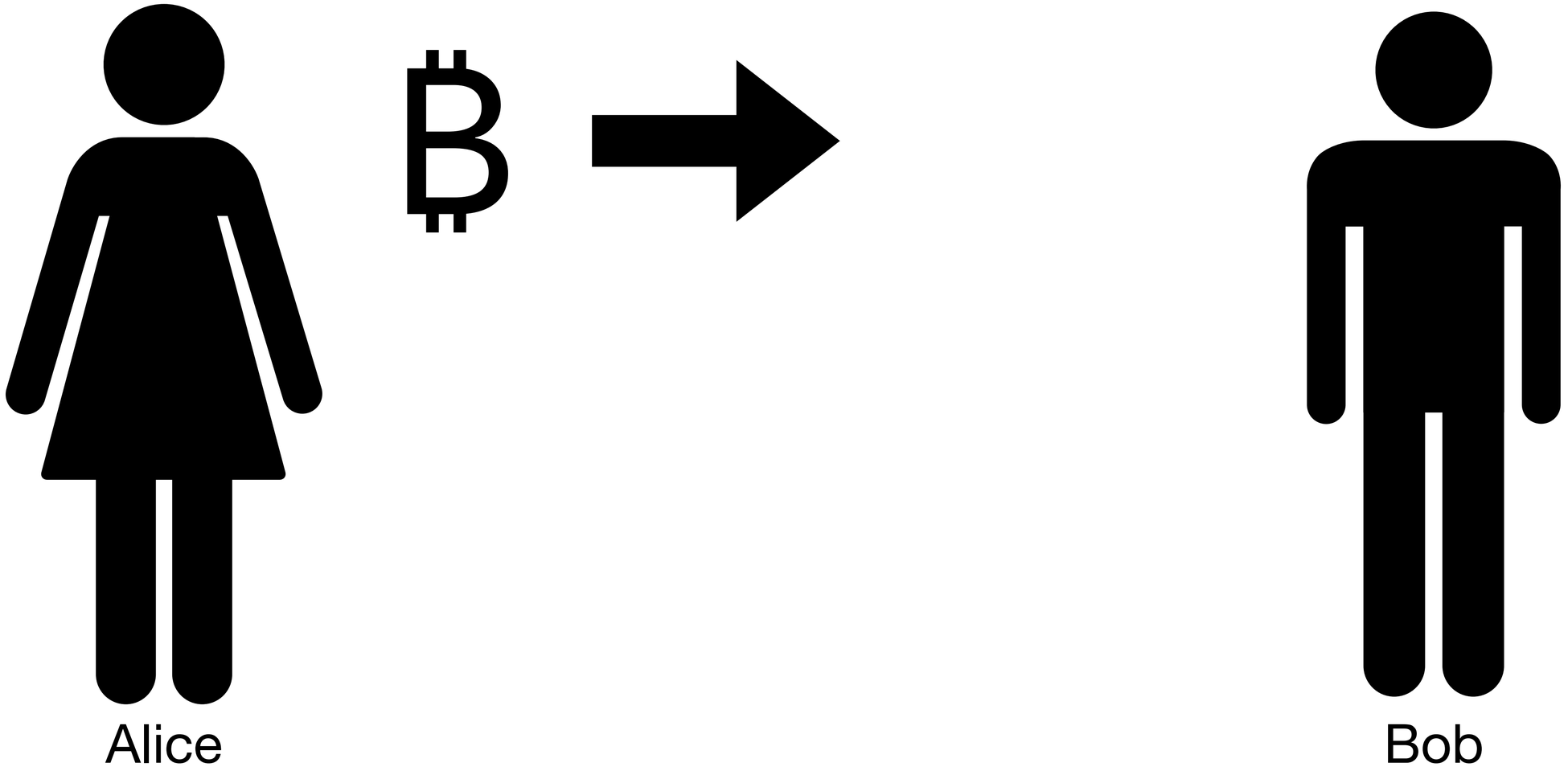


Bob

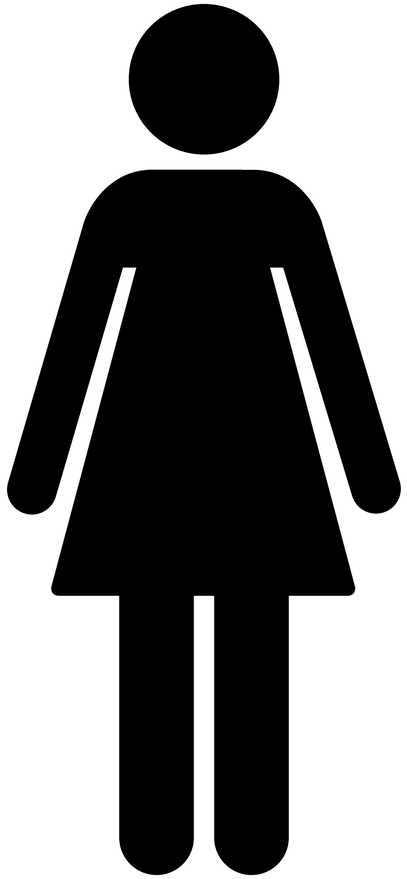
Barter with Bitcoin



Barter with Bitcoin

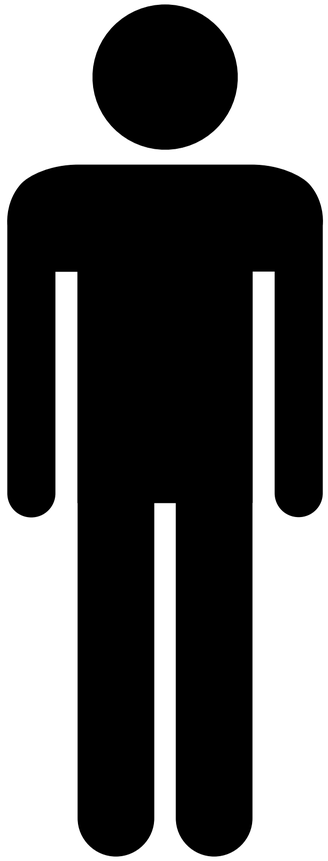


Barter with Bitcoin



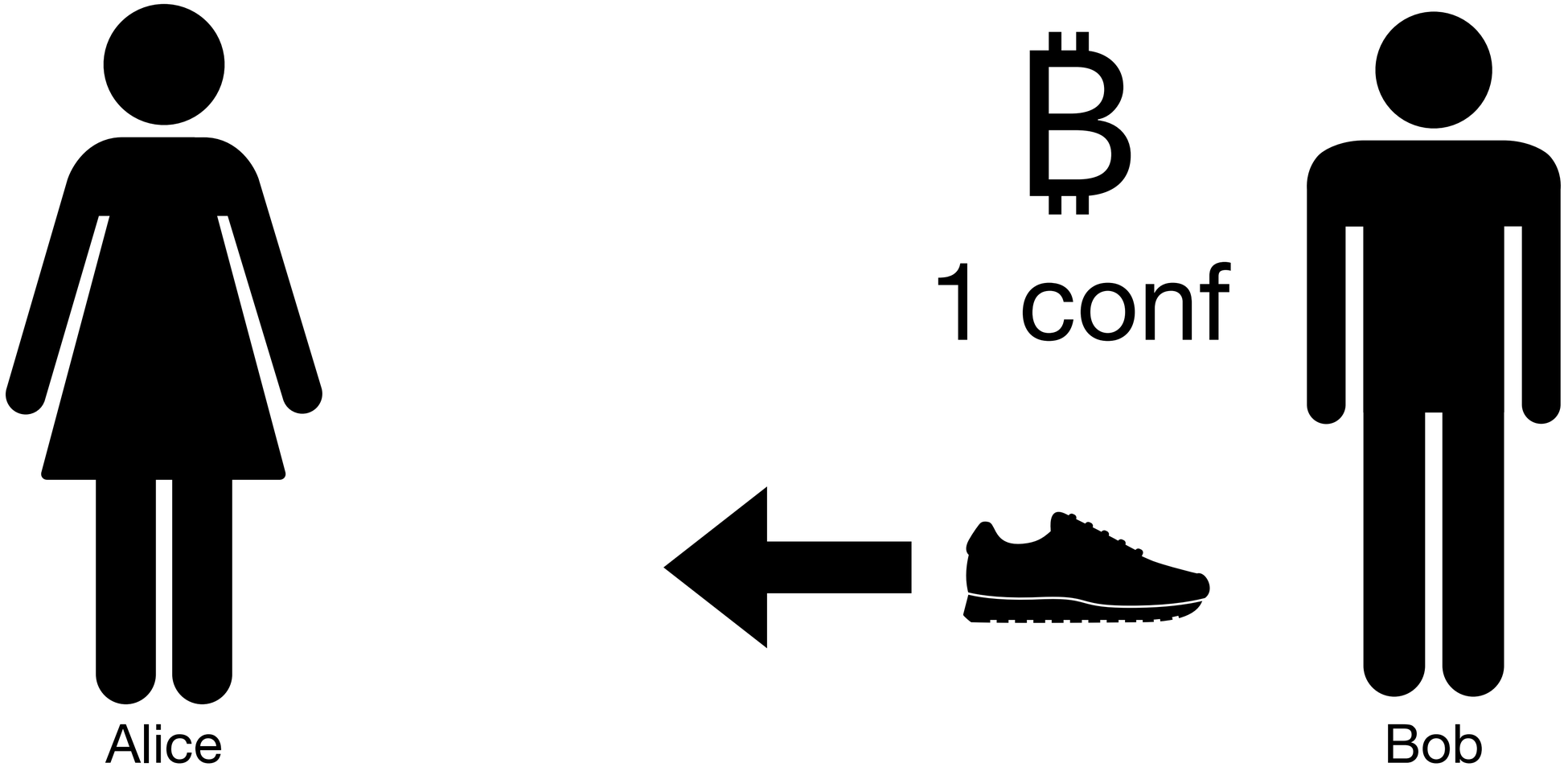
Alice

1 conf

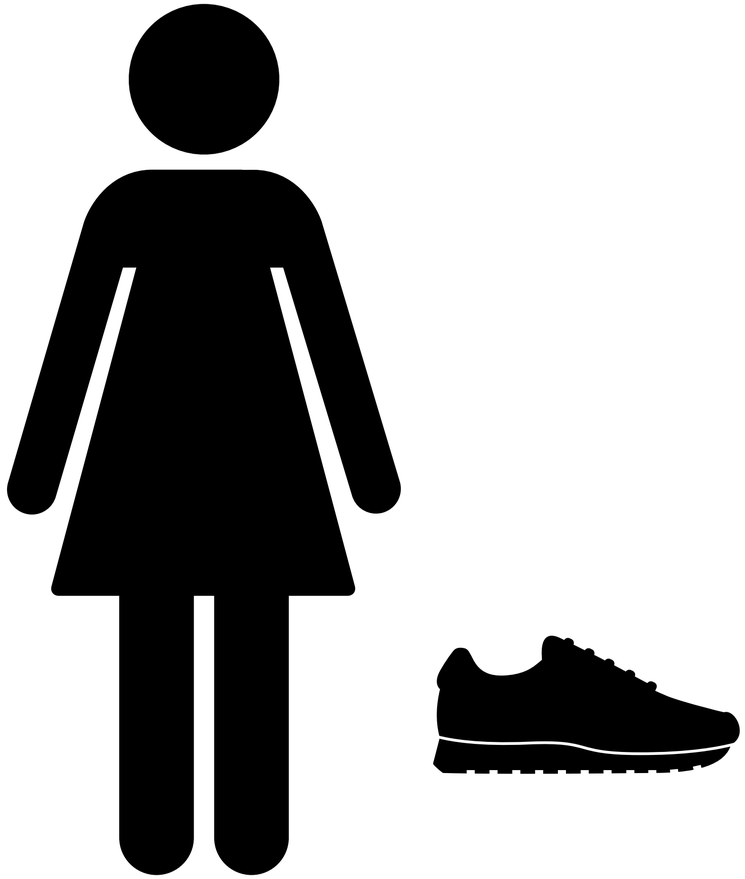


Bob

Barter with Bitcoin

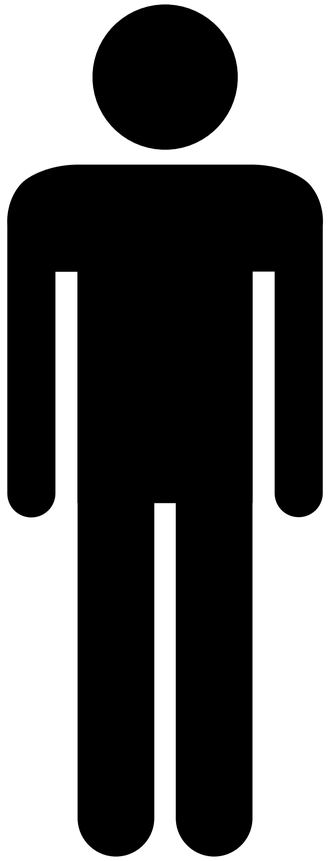


Barter with Bitcoin



Alice

1 conf



Bob

Barter with Bitcoin

- Alice sends Bitcoin to Bob

Barter with Bitcoin

- Alice sends Bitcoin to Bob
- Bob waits for a confirmation of the transaction

Barter with Bitcoin

- Alice sends Bitcoin to Bob
- Bob waits for a confirmation of the transaction
- Bob only accepts the payment after the confirmation

Cash

- Physically hand over the bill

Bitcoin

- Wait for a confirmation of the tx

What's a confirmation?

Bitcoin's rules

- Payment isn't final before a confirmation

What's a confirmation?

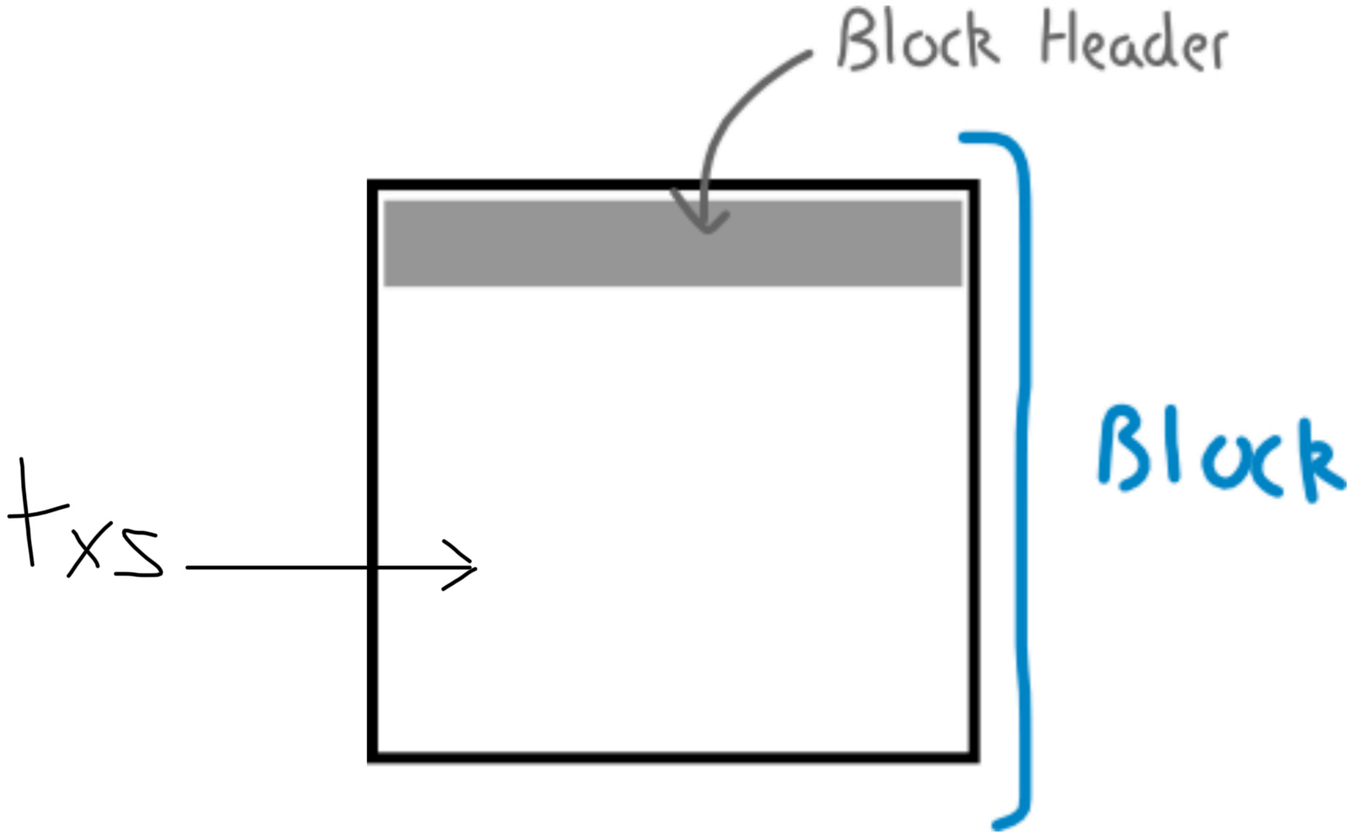
Bitcoin's rules

- Payment isn't final before a confirmation
- Inclusion of your transaction in a block is a confirmation

What's a block?

Bitcoin Block

Bitcoin Block Internals



https://learnmeabitcoin.com/technical/block-header

Bitcoin Block

Bitcoin Block Internals

- Metadata (Block Header)

Bitcoin Block

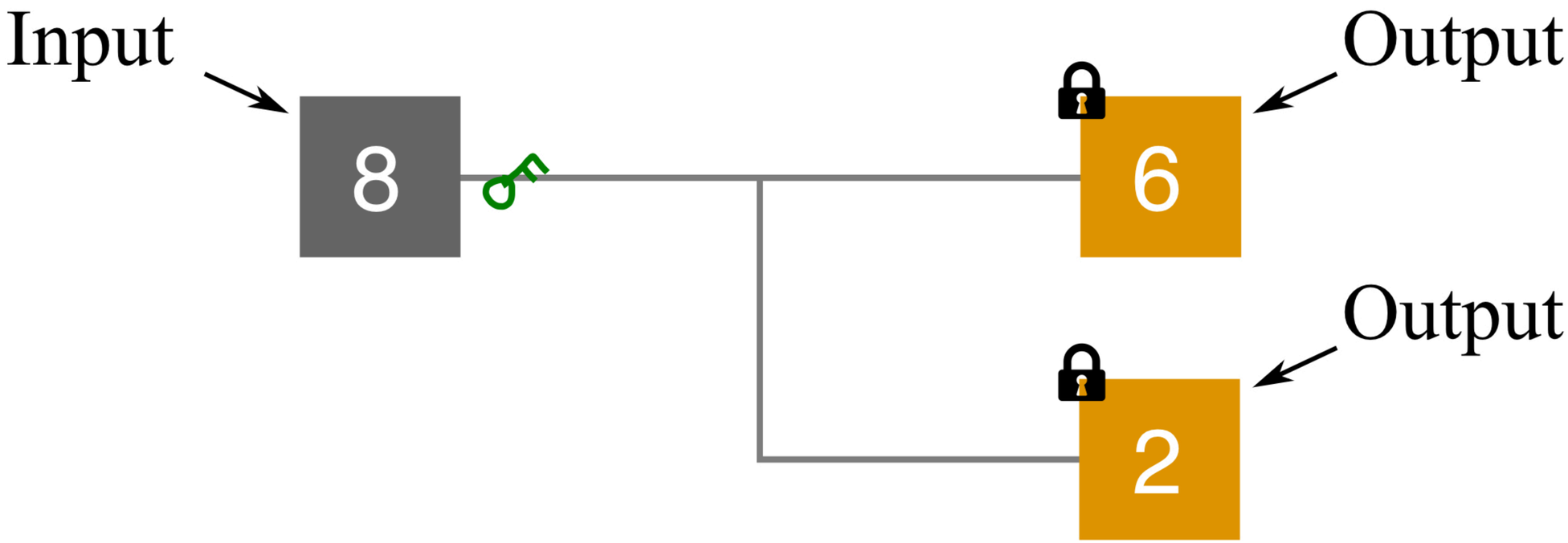
Bitcoin Block Internals

- Metadata (Block Header)
- A collection of transactions

What's a transaction?

A record of spent Bitcoins and newly spendable Bitcoins

Bitcoin Transaction



https://learnmeabitcoin.com/technical/transaction-data

Bitcoin Transaction

- Inputs are spent Bitcoins

Bitcoin Transaction

- Inputs are spent Bitcoins
- Outputs are newly spendable Bitcoins

Bitcoin Transaction

- Inputs are spent Bitcoins
- Outputs are newly spendable Bitcoins
- Output amount < Input amount

Bitcoin Transaction

Real world example

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

Bitcoin Transaction

Real world example

8.84354951

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

Bitcoin Transaction

Real world example

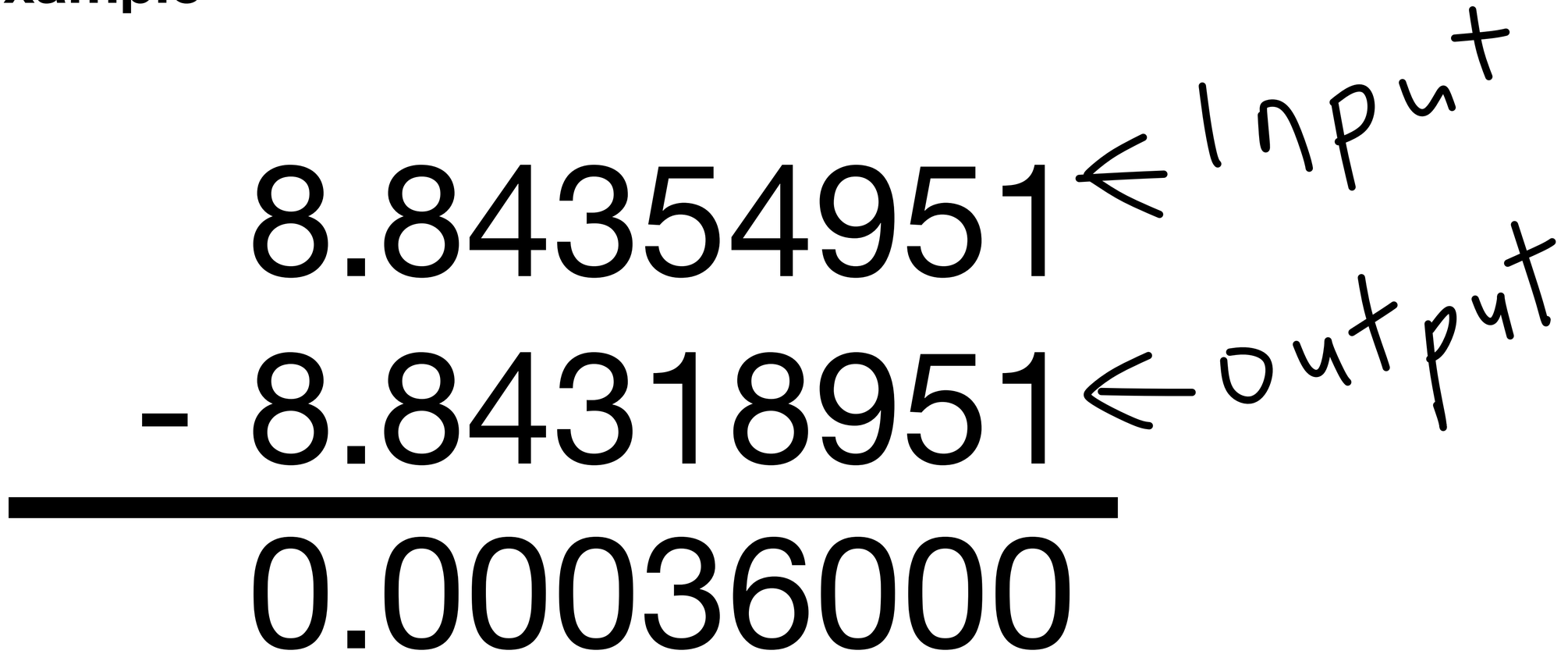
8.84354951

8.84318951

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

Bitcoin Transaction

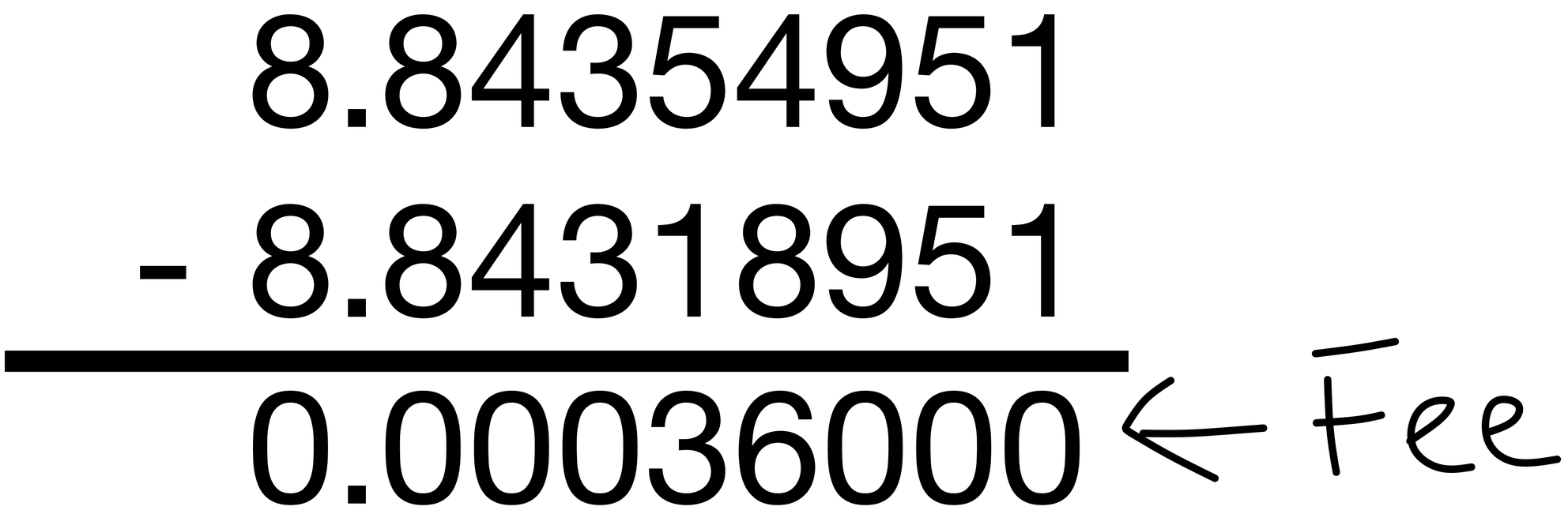
Real world example



https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

Bitcoin Transaction

Real world example



https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

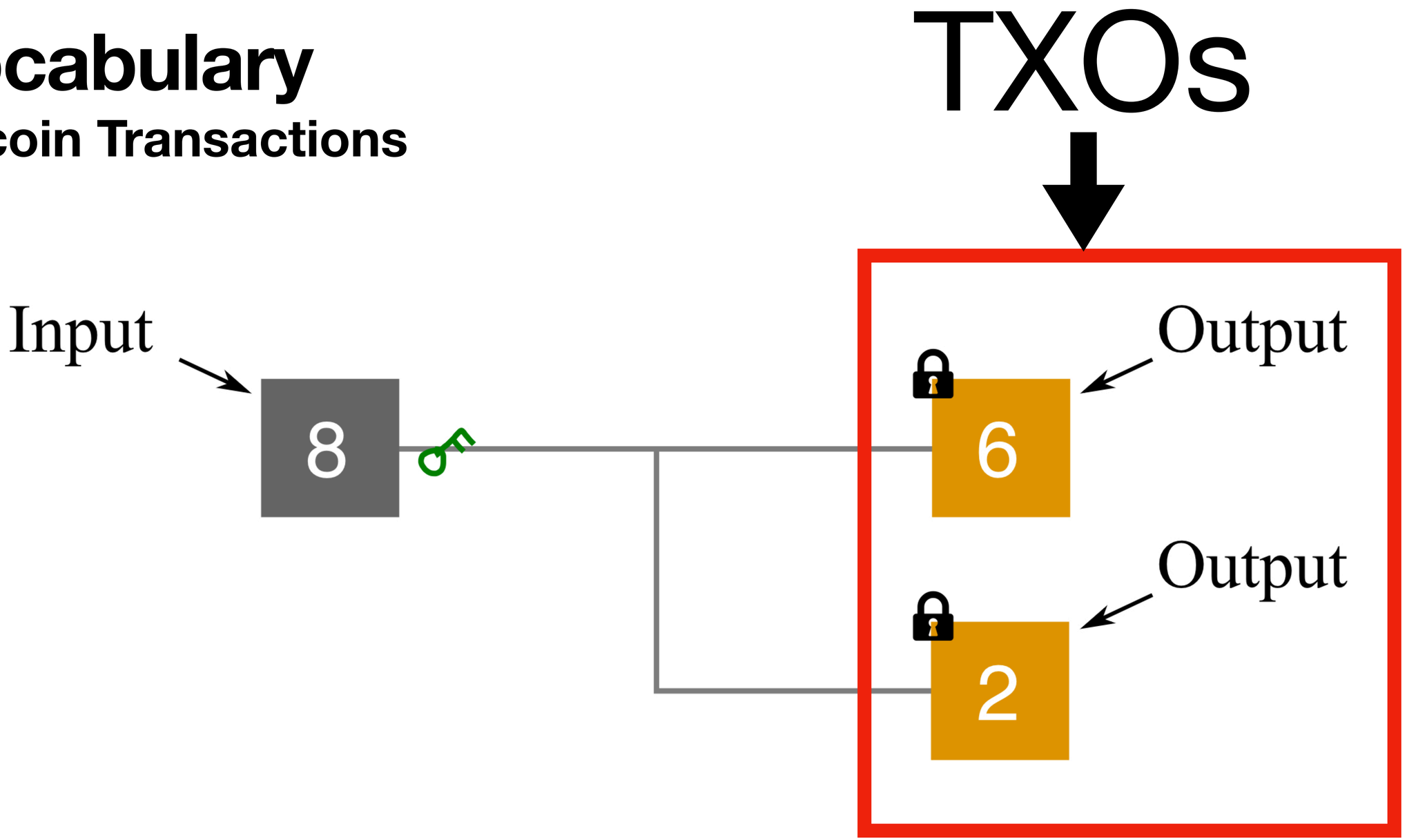
Vocabulary

Bitcoin Transactions

- TXO - An Output of a TX (transaction)

Vocabulary

Bitcoin Transactions



https://learnmeabitcoin.com/technical/transaction-data

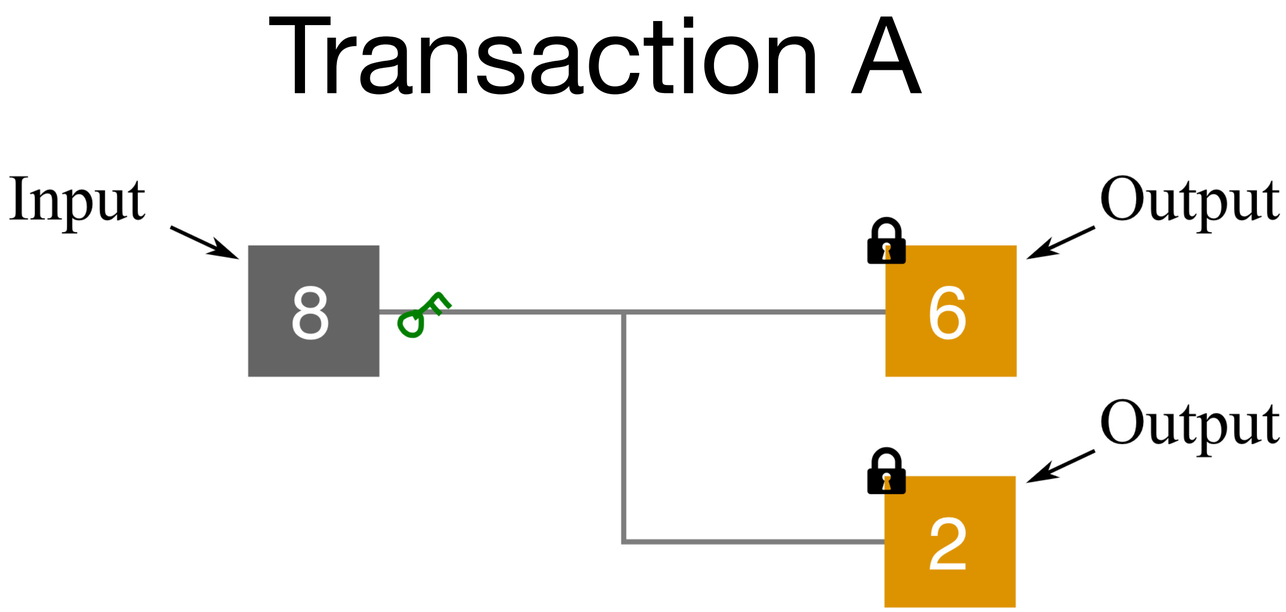
Vocabulary

Bitcoin Transactions

- TXO - An Output of a TX (transaction)
- UTXO - Unspent TXO

Vocabulary

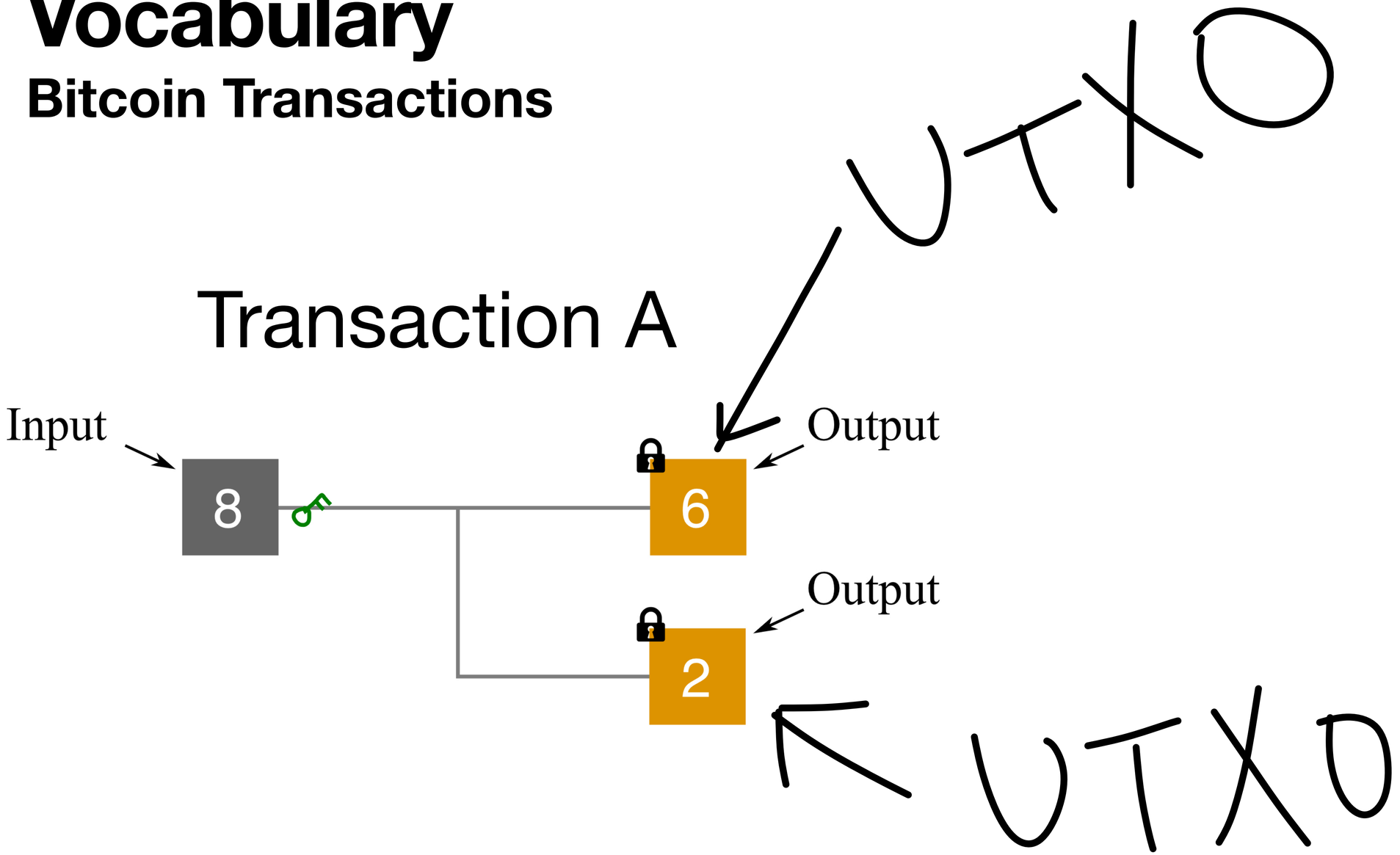
Bitcoin Transactions



https://learnmeabitcoin.com/technical/transaction-data

Vocabulary

Bitcoin Transactions



https://learnmeabitcoin.com/technical/transaction-data

Vocabulary

Bitcoin Transactions

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

Vocabulary

Bitcoin Transactions

https://mempool.space/tx/c3b12e33a5dfc80b2e15cb920e588af6ce12f98ed16544e498bf634b63310ec3

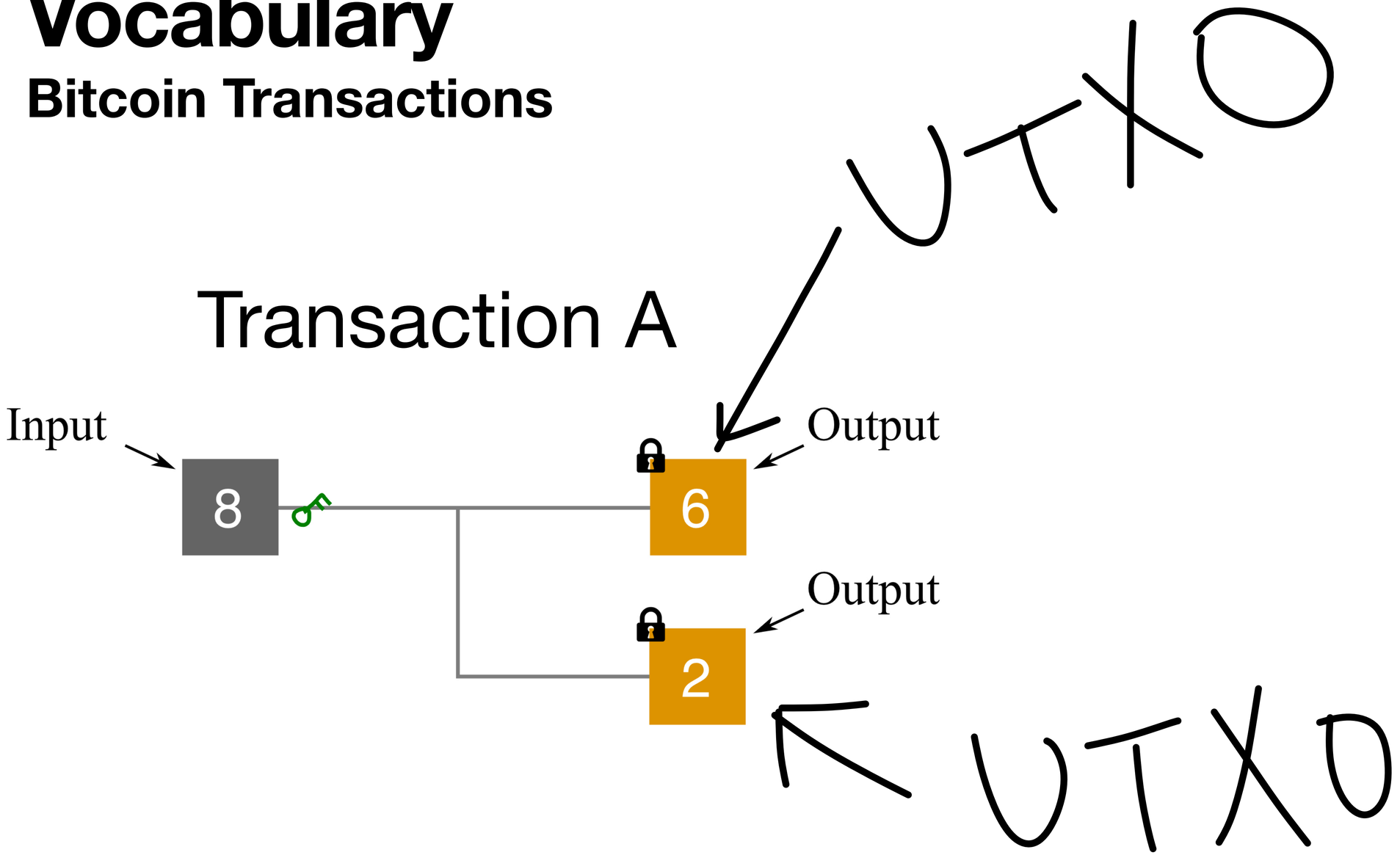
Vocabulary

Bitcoin Transactions

- TXO - An Output of a TX (transaction)
- UTXO - Unspent TXO
- STXO - Spent TXO

Vocabulary

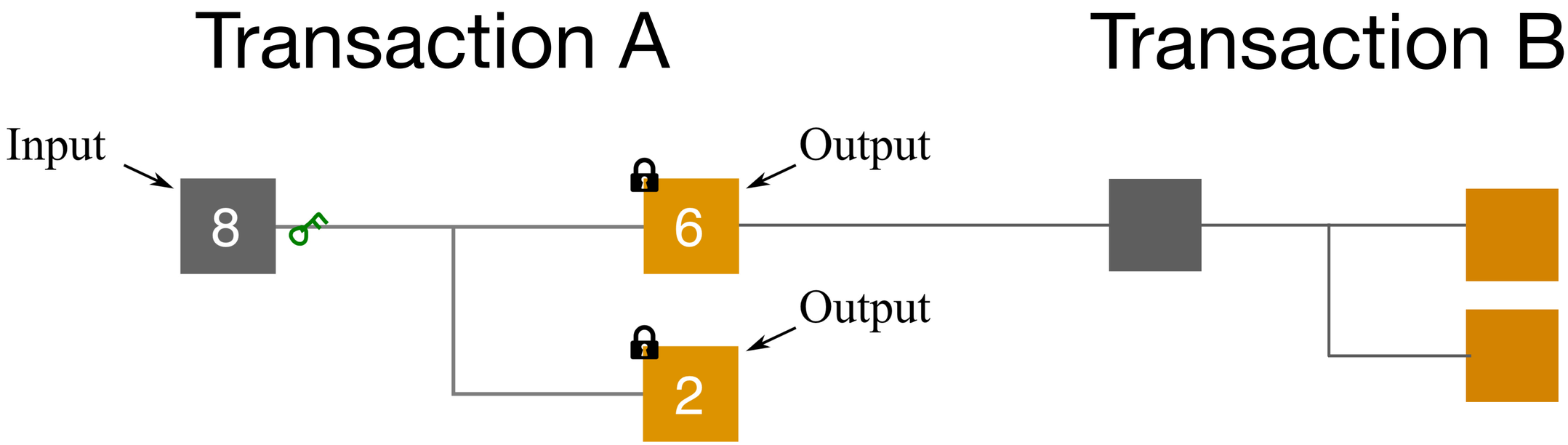
Bitcoin Transactions



https://learnmeabitcoin.com/technical/transaction-data

Vocabulary

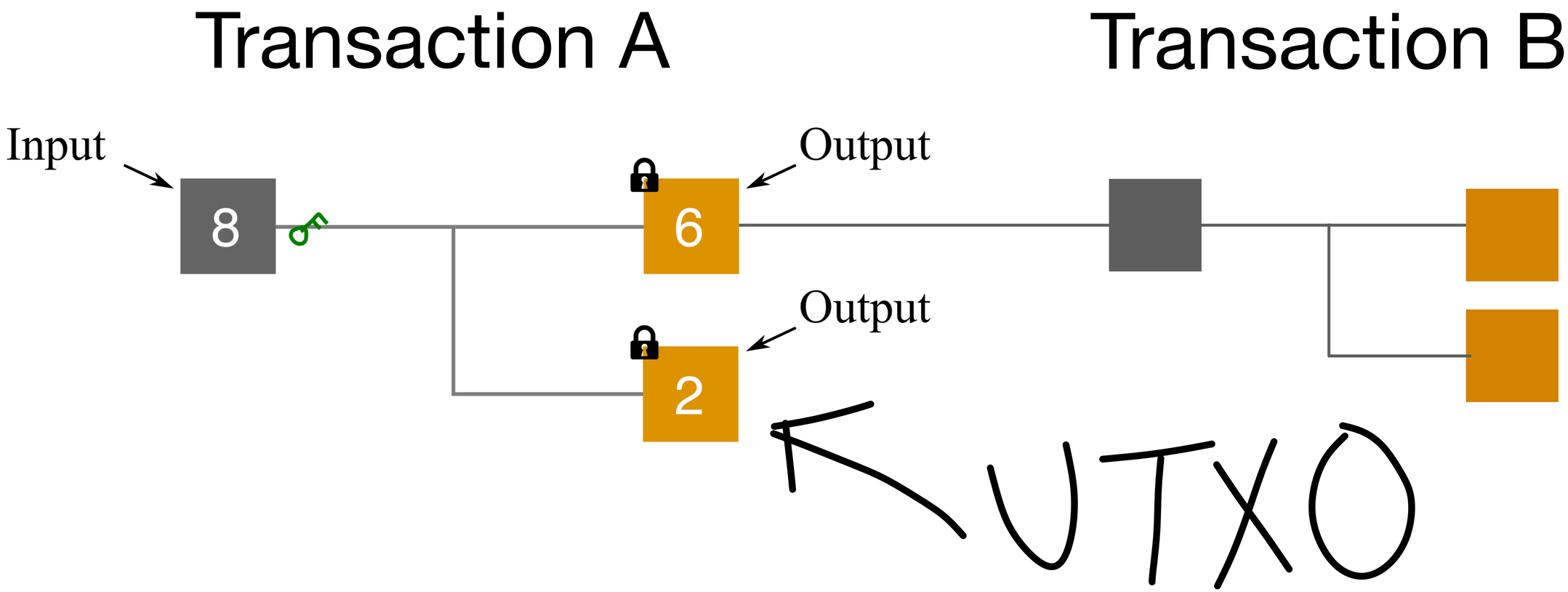
Bitcoin Transactions



https://learnmeabitcoin.com/technical/transaction-data

Vocabulary

Bitcoin Transactions

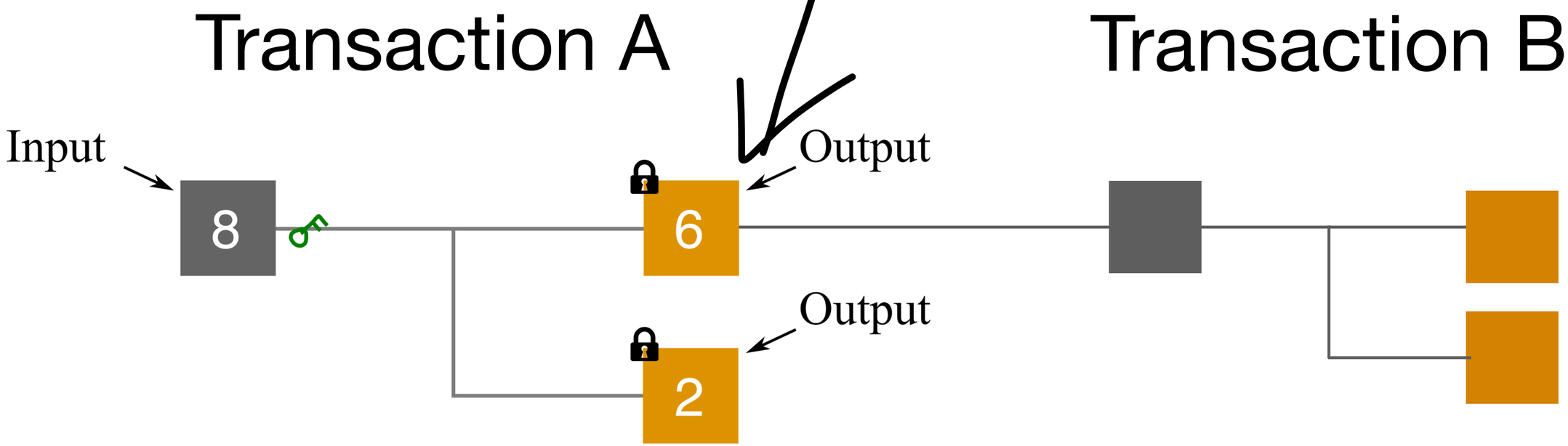


https://learnmeabitcoin.com/technical/transaction-data

Vocabulary

Bitcoin Transactions

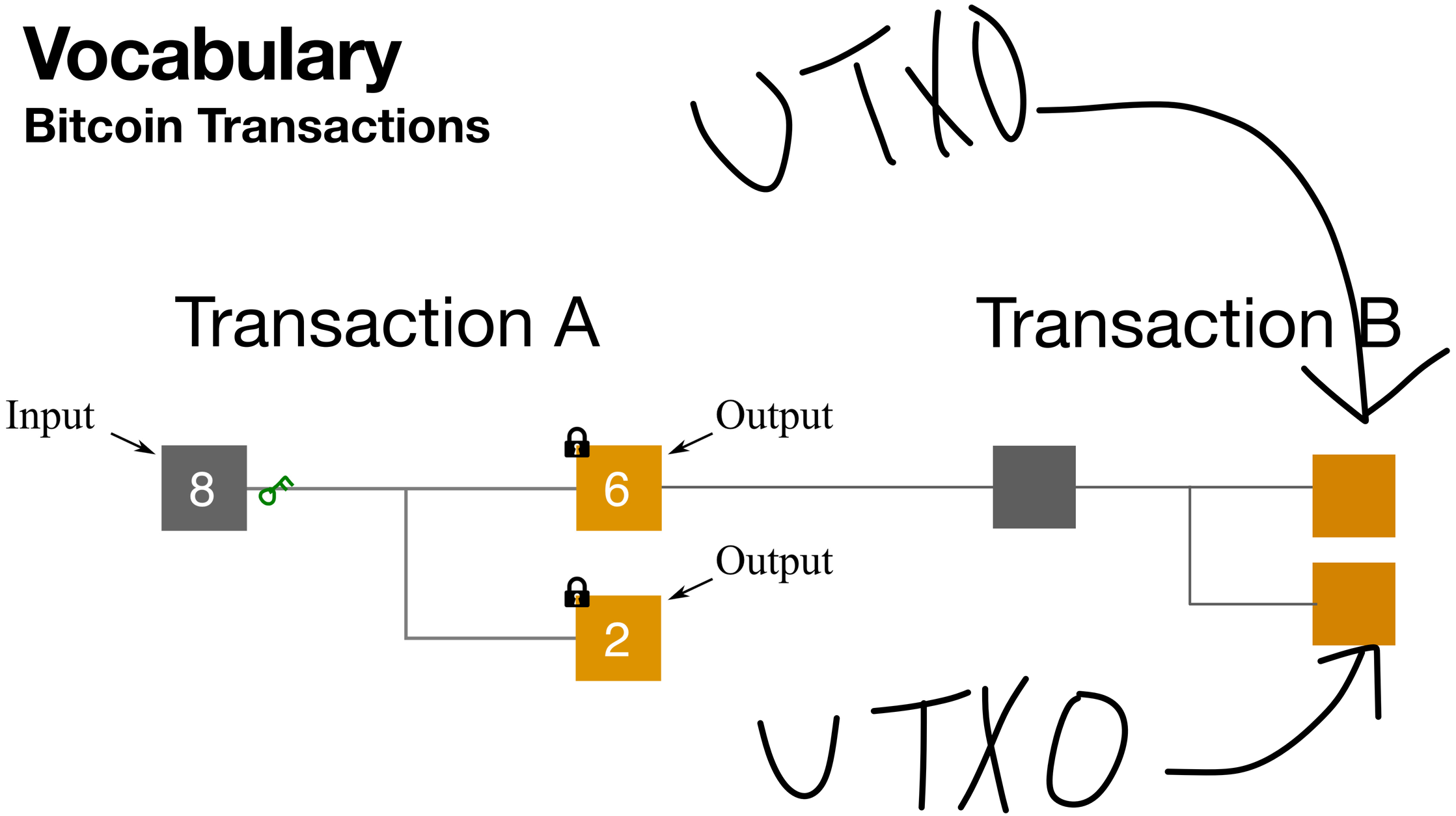
STXO



https://learnmeabitcoin.com/technical/transaction-data

Vocabulary

Bitcoin Transactions



https://learnmeabitcoin.com/technical/transaction-data

Whitepaper section 5

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Whitepaper section 5

5. Network

The steps to run the network are as follows:

1) New transactions are broadcast to all nodes.
2) Each node collects new transactions into a block.
3) Each node works on finding a difficult proof-of-work for its block.
4) When a node finds a proof-of-work, it broadcasts the block to all nodes.
5) Nodes accept the block only if all transactions in it are valid and not already spent.
6) Nodes express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash.

Nodes accept the block only if all the transactions are valid and not already spent

Nodes accept the block only if all the transactions are valid and not already spent

Not Already Spent

Transaction validation

- Inputs must be spending UTXOs not STXOs

Not Already Spent

Transaction validation

- Inputs must be spending UTXOs not STXOs
- Invalid if the referenced TXO is spent or doesn't exist

What's referenced when checking for a UTXO?

The UTXO set!

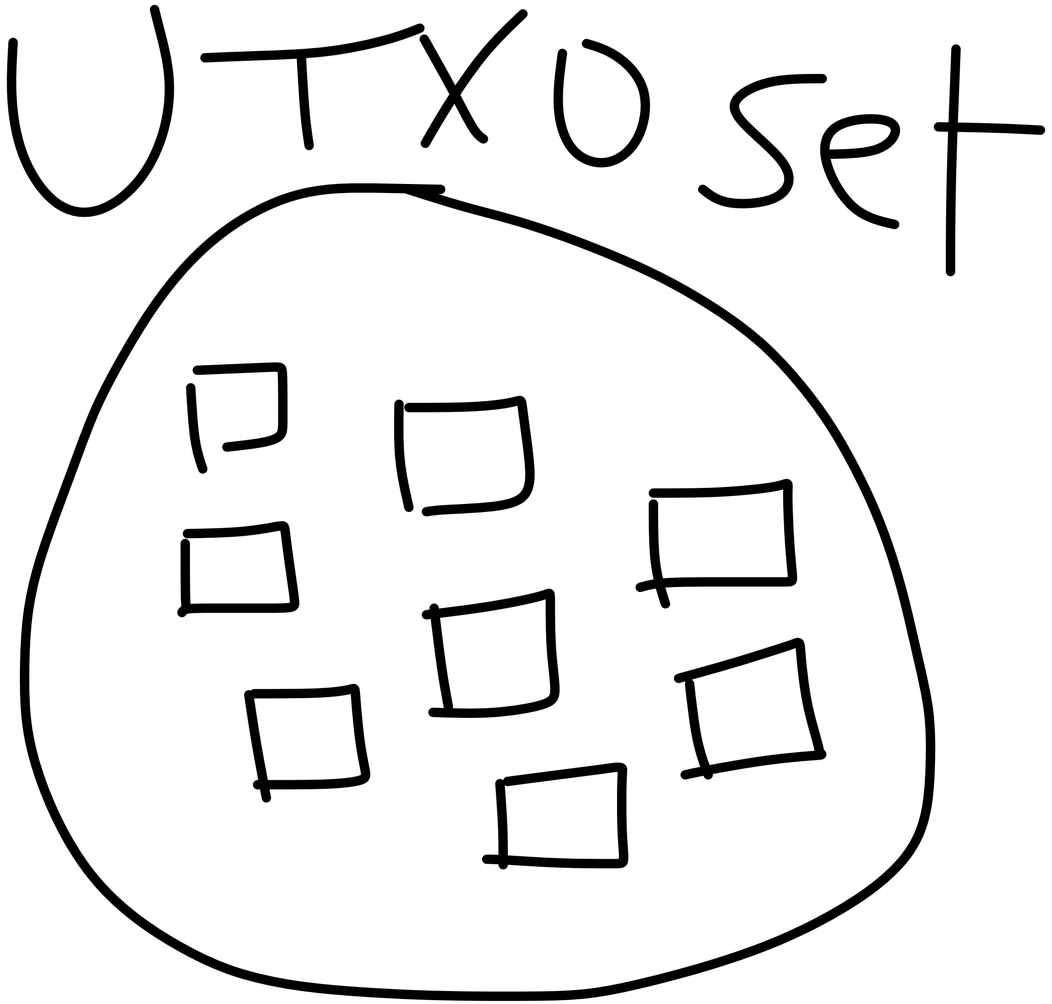
Vocabulary

Bitcoin Transactions

- TXO - An Output of a TX (transaction)
- UTXO - Unspent TXO
- STXO - Spent TXO
- UTXO set - All UTXOs

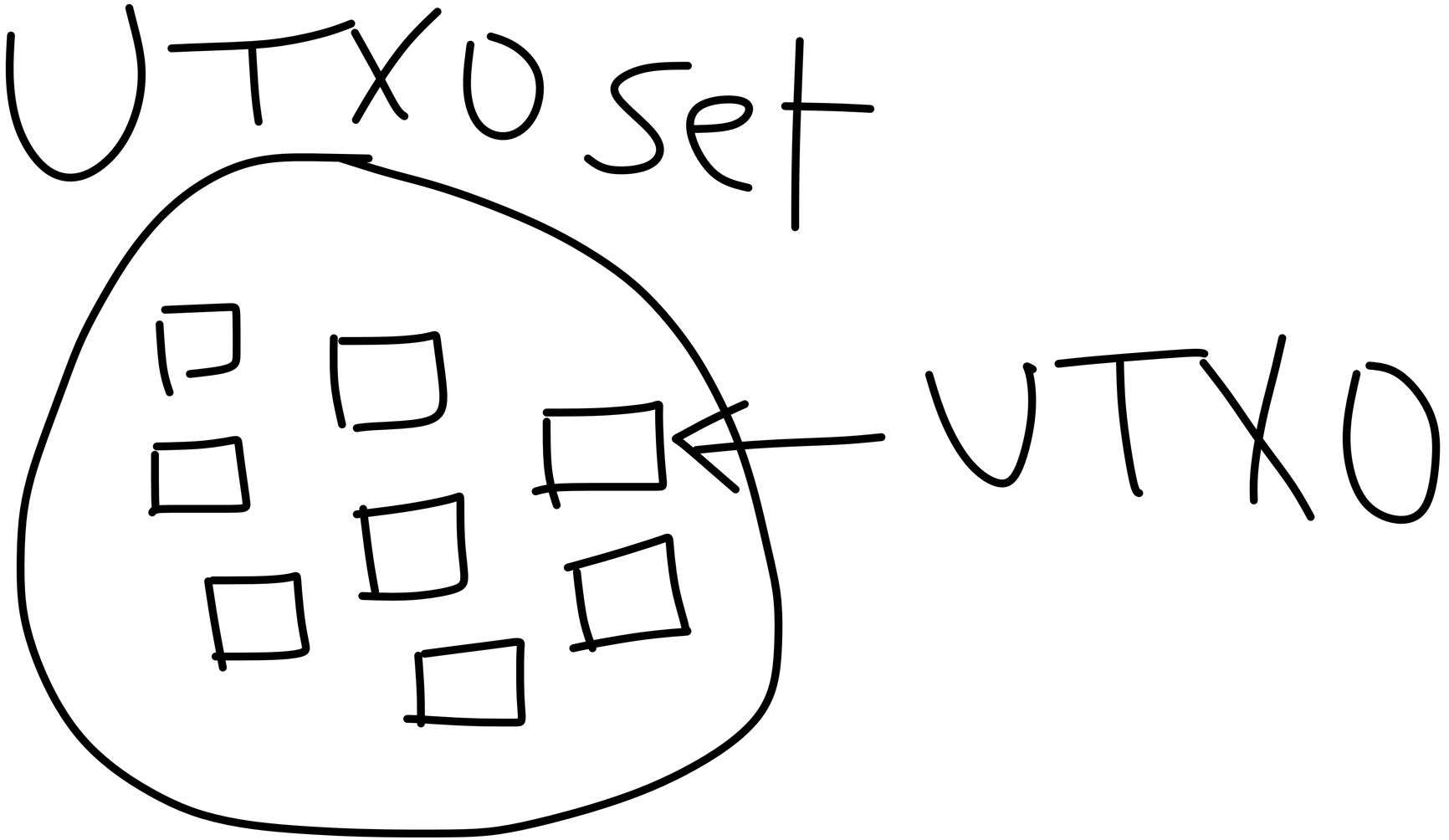
The UTXO set

All the Bitcoins that are available to be spent



The UTXO set

All the Bitcoins that are available to be spent



A single UTXO

What a UTXO is

- Referenced by TXID:Index

A single UTXO

What a UTXO is

- Referenced by TXID:Index
- Contains all the data needed for verification (amount, pkscript, height, whether or not it's a coinbase tx)

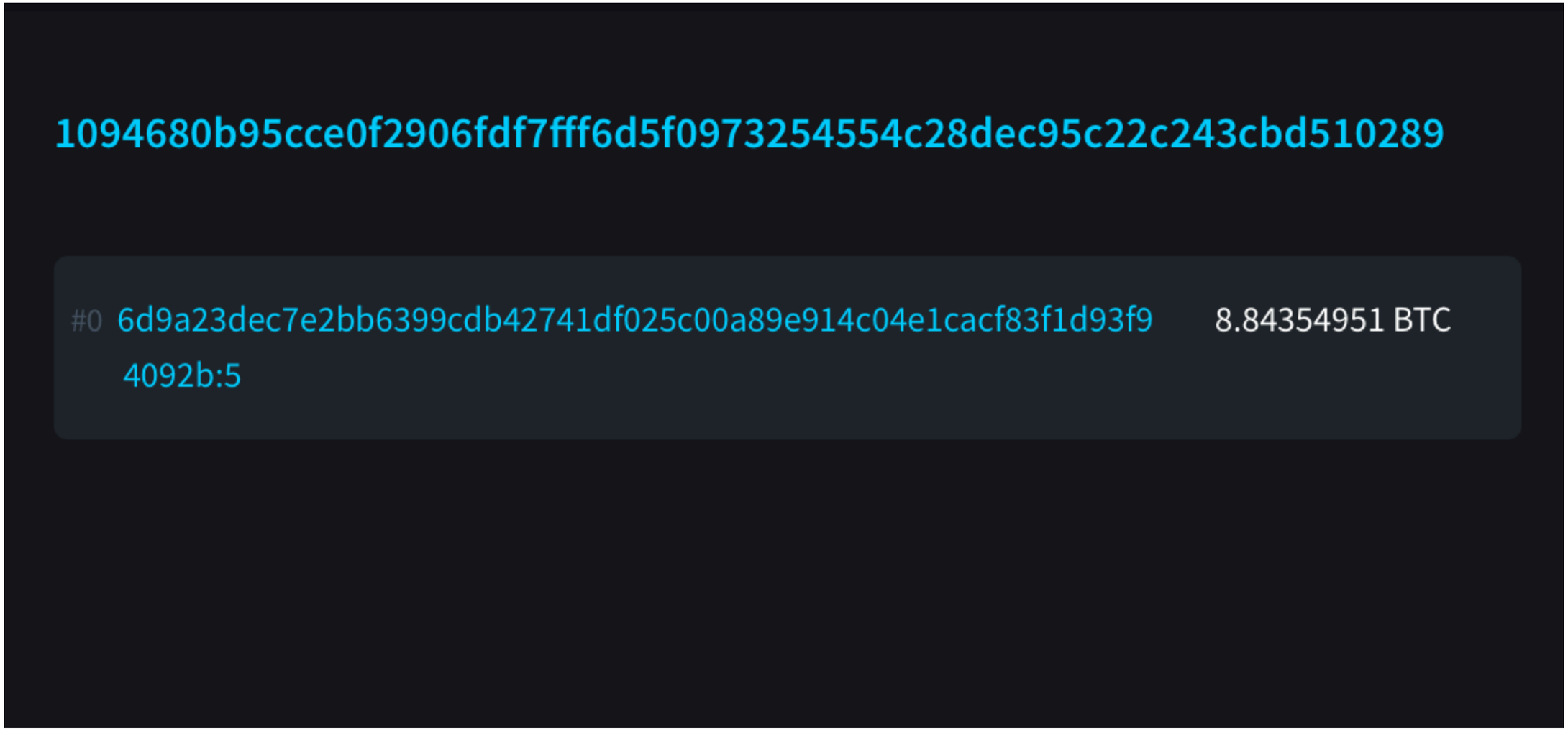
A single UTXO

What a UTXO is

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

A single UTXO

What a UTXO is



https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

A single UTXO

What a UTXO is

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

A single UTXO

What a UTXO is

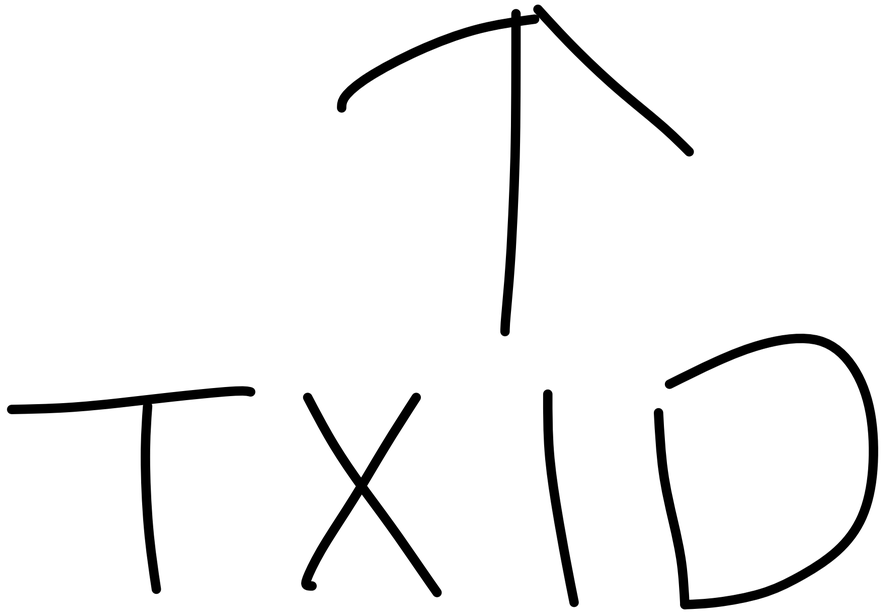
6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f94092b:5

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

A single UTXO

What a UTXO is

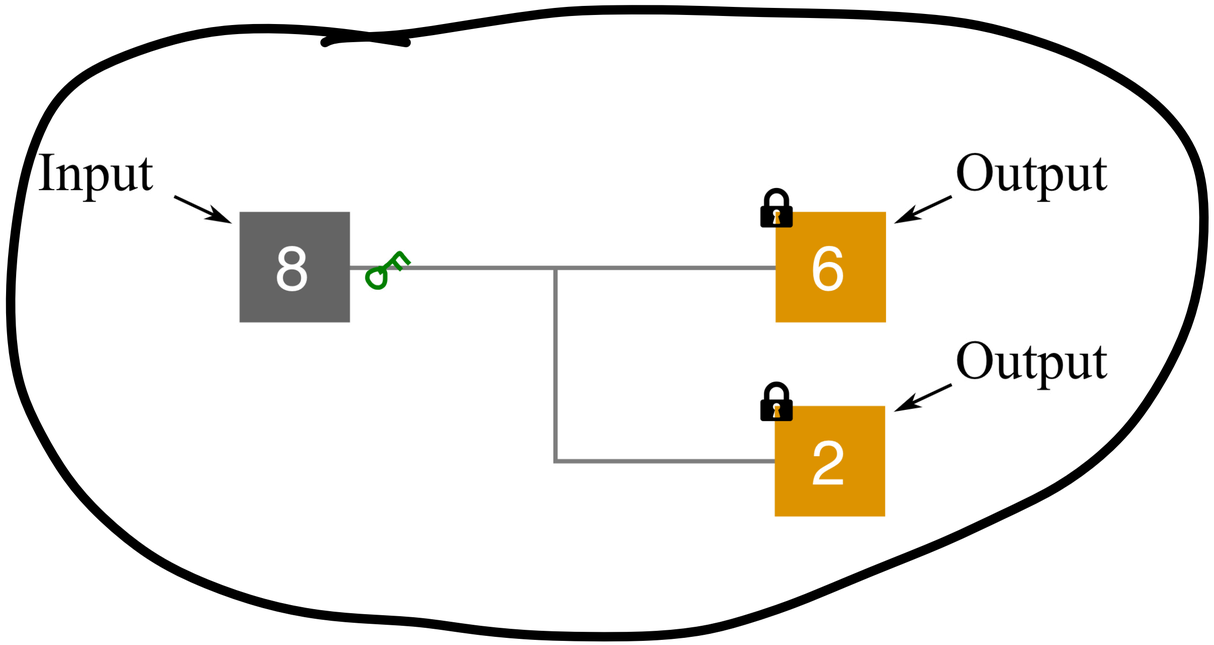
6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f94092b:5



https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

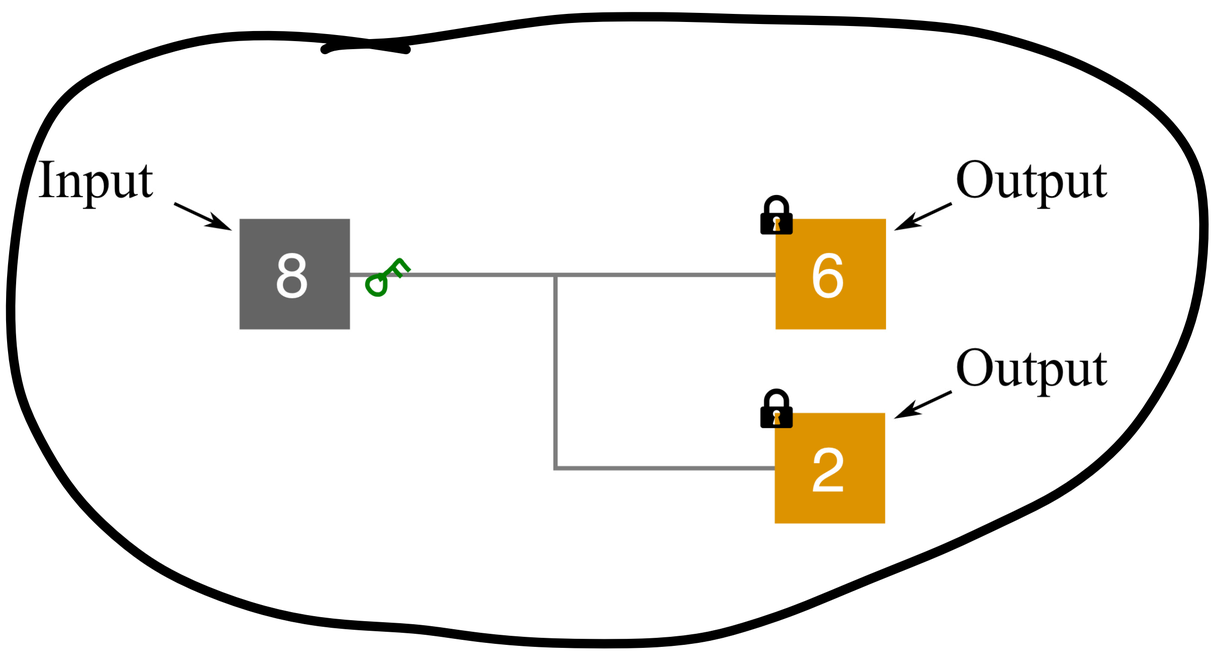
A single UTXO

What a UTXO is



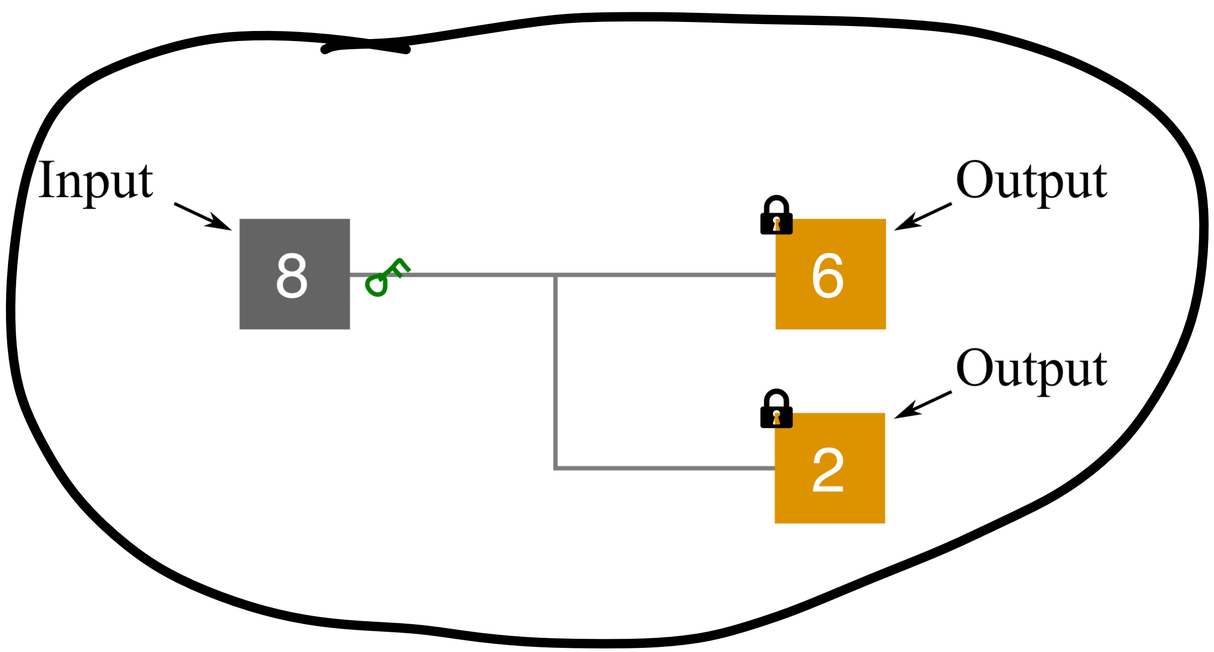
A single UTXO

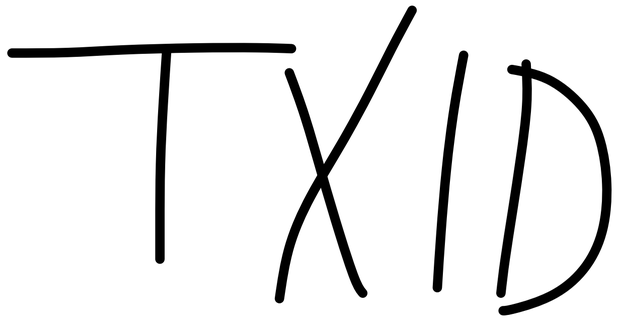
What a UTXO is



A single UTXO

What a UTXO is





A single UTXO

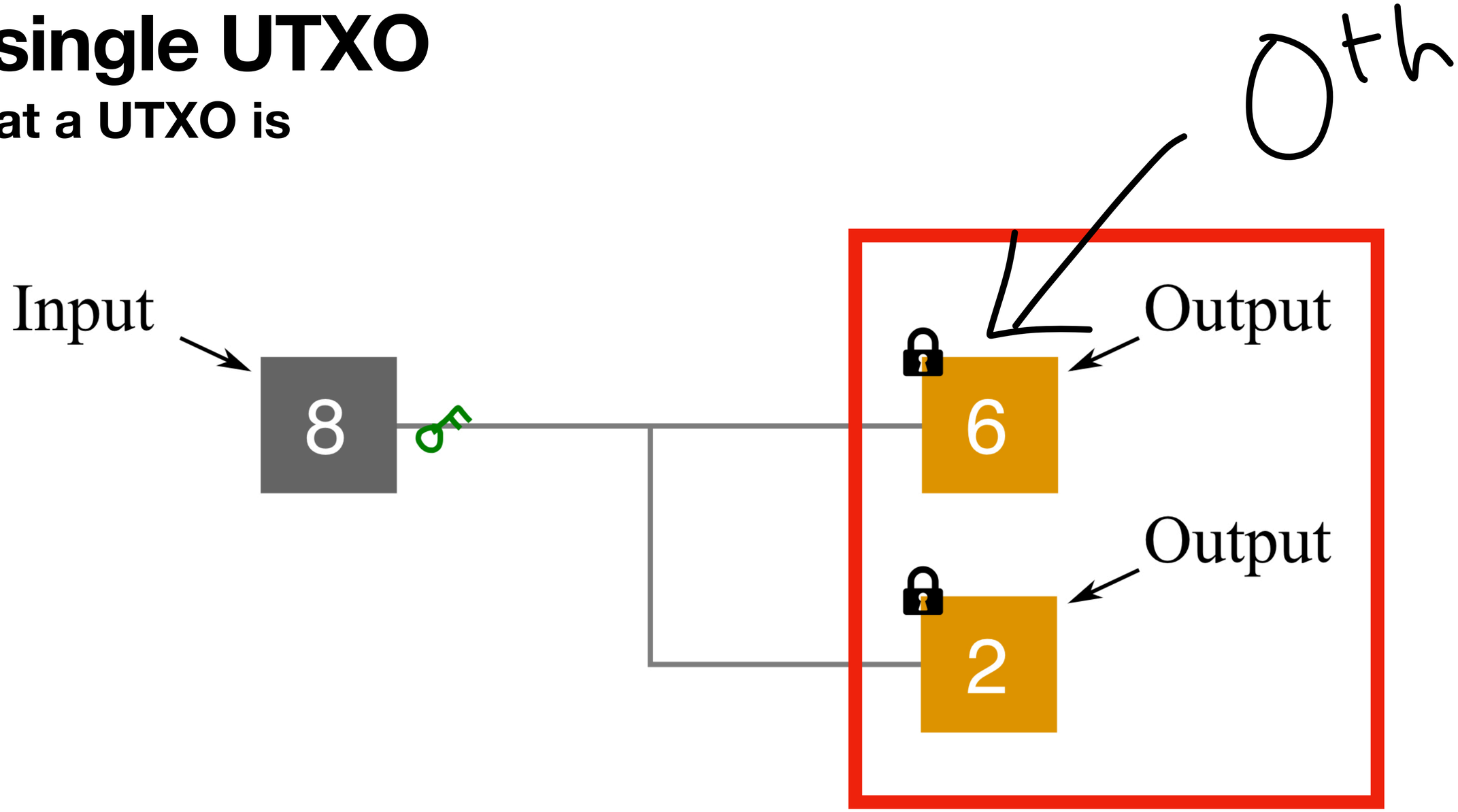
What a UTXO is

6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f94092b:5

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

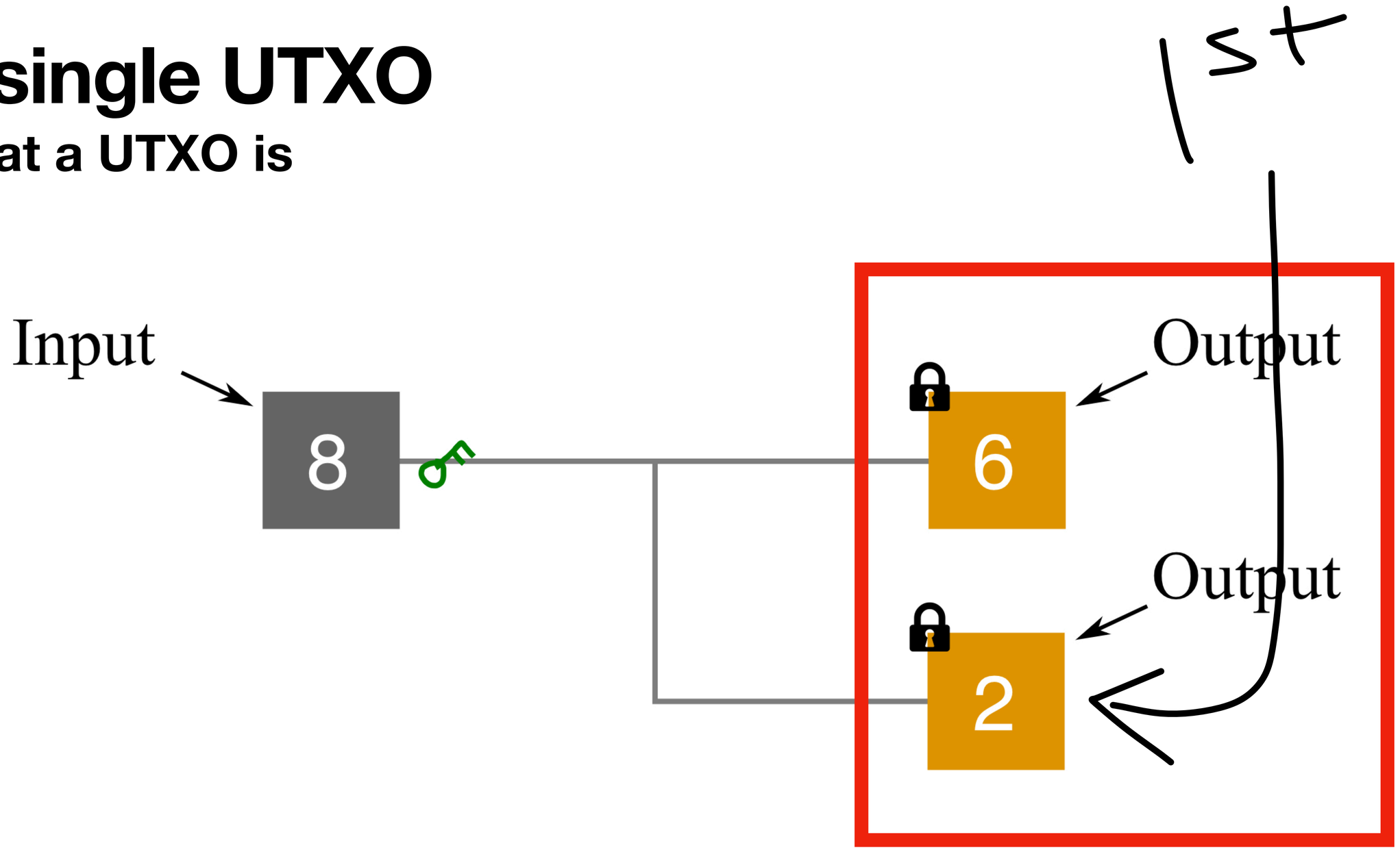
A single UTXO

What a UTXO is



A single UTXO

What a UTXO is



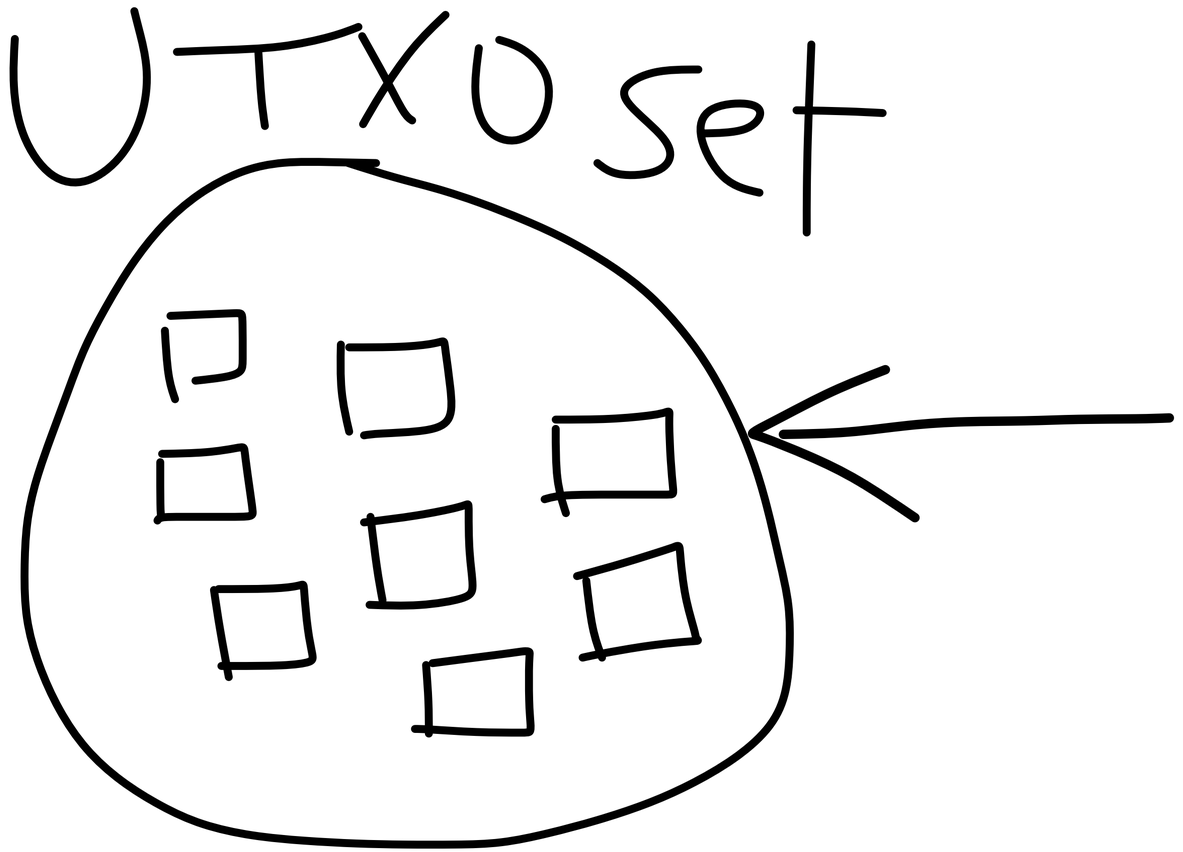
Validating a TX

Real world example

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

Validating a TX

Real world example



6d9a23dec7e2bb6399cdb
42741df025c00a89e914c0
4e1cacf83f1d93f94092b:5

https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289

Validating a TX

Real world example

- Valid if: UTXO exists in the UTXO set

Validating a TX

Real world example

- Valid if: UTXO exists in the UTXO set
- Invalid if: UTXO doesn't exist

What is Utreexo?


Hello
I'm Tadge Dryja & I've started some fun things in bitcoin:
Lightning network <- popular
Discreet Log Contracts <- up & coming
Utreexo <- not yet

Utreexo?

What is it?

- A fancy merkle tree proposed by Tadge Dryja

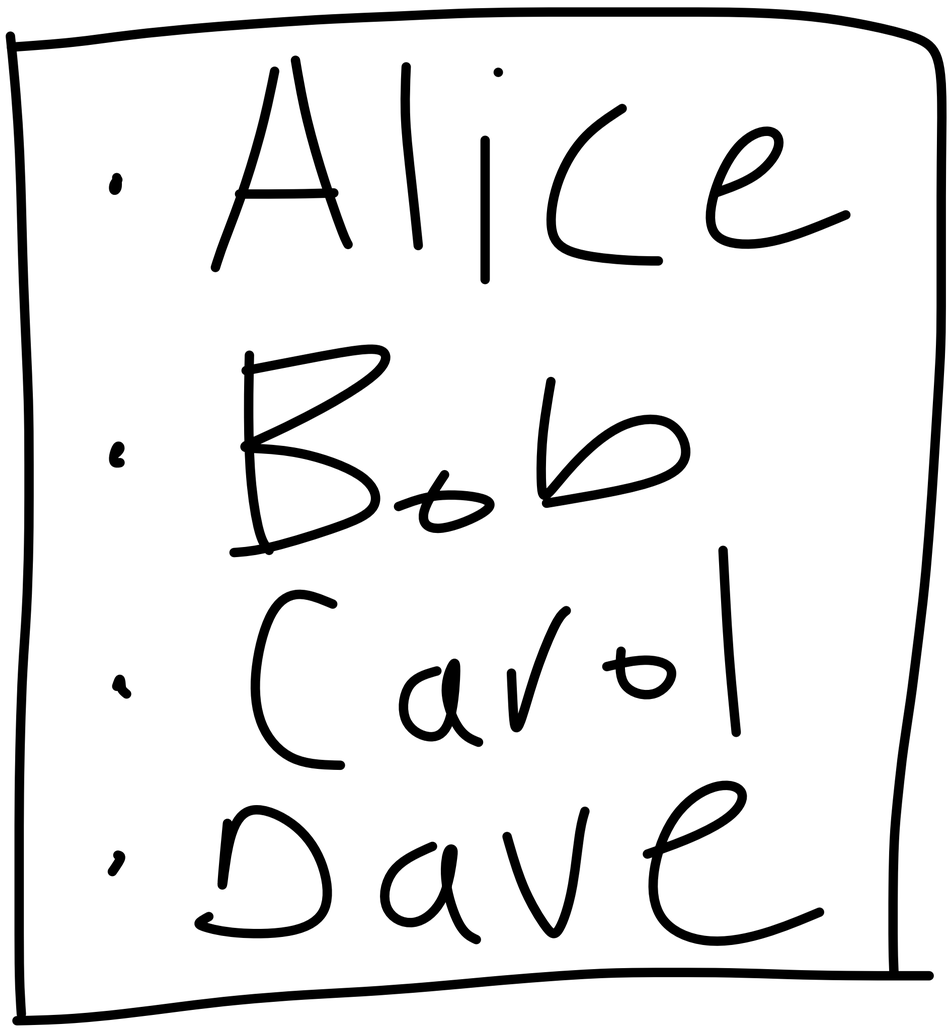
Merkle Tree

What is it?

Let's you save *a lot* of space but still let you verify existence

Merkle Tree

What is it?



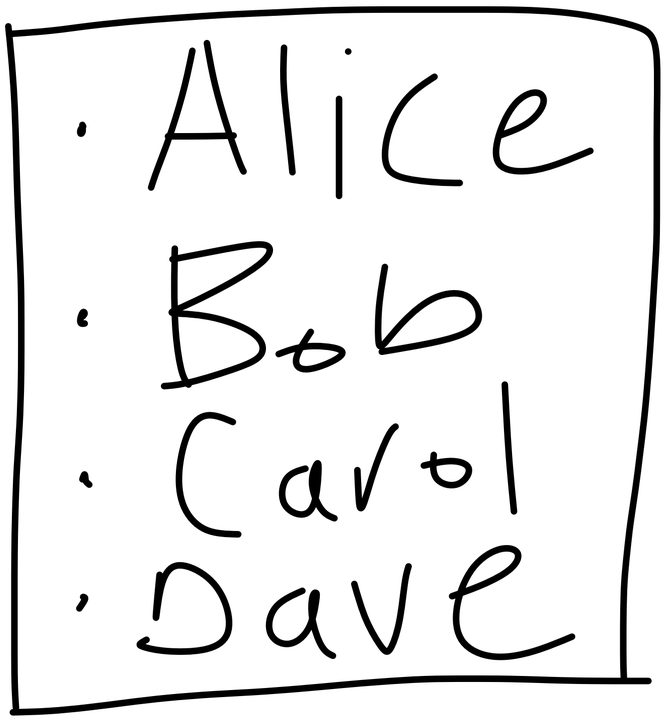
Merkle Tree

What is it?

- If the list is huge, then it's *a lot* of paper that the bouncer needs

Merkle Tree

What is it?

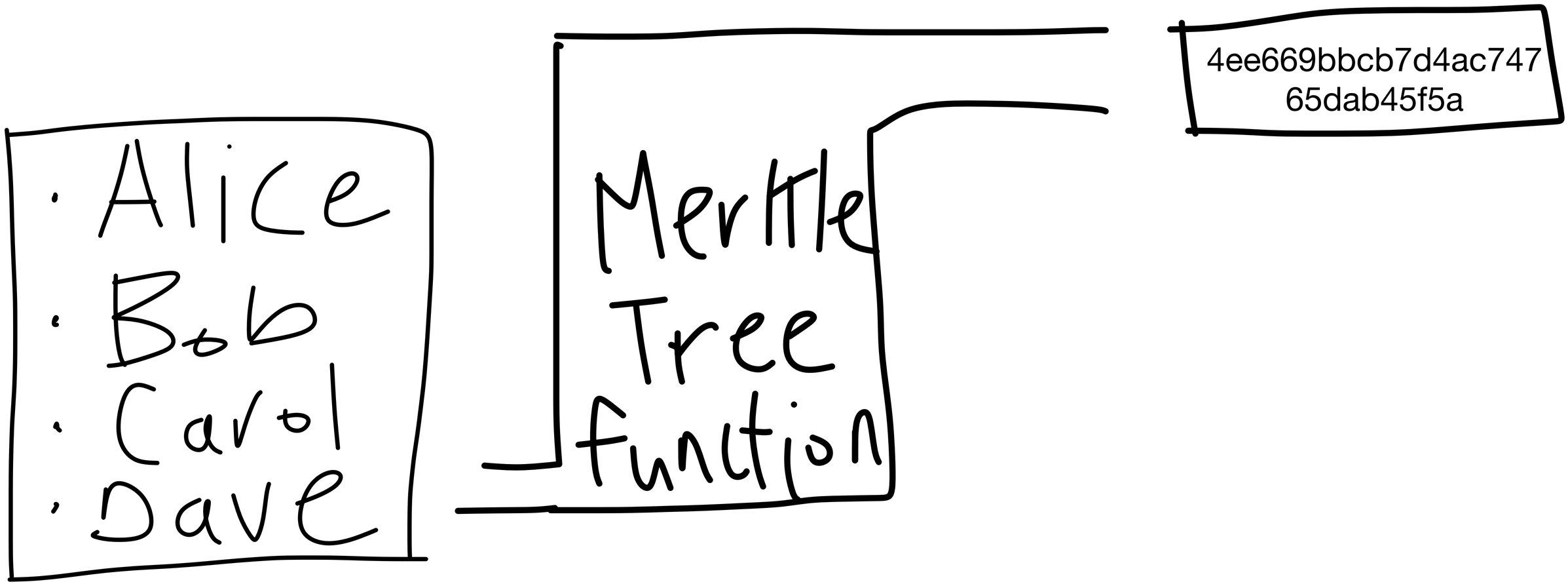


Merkle Tree

What is it?

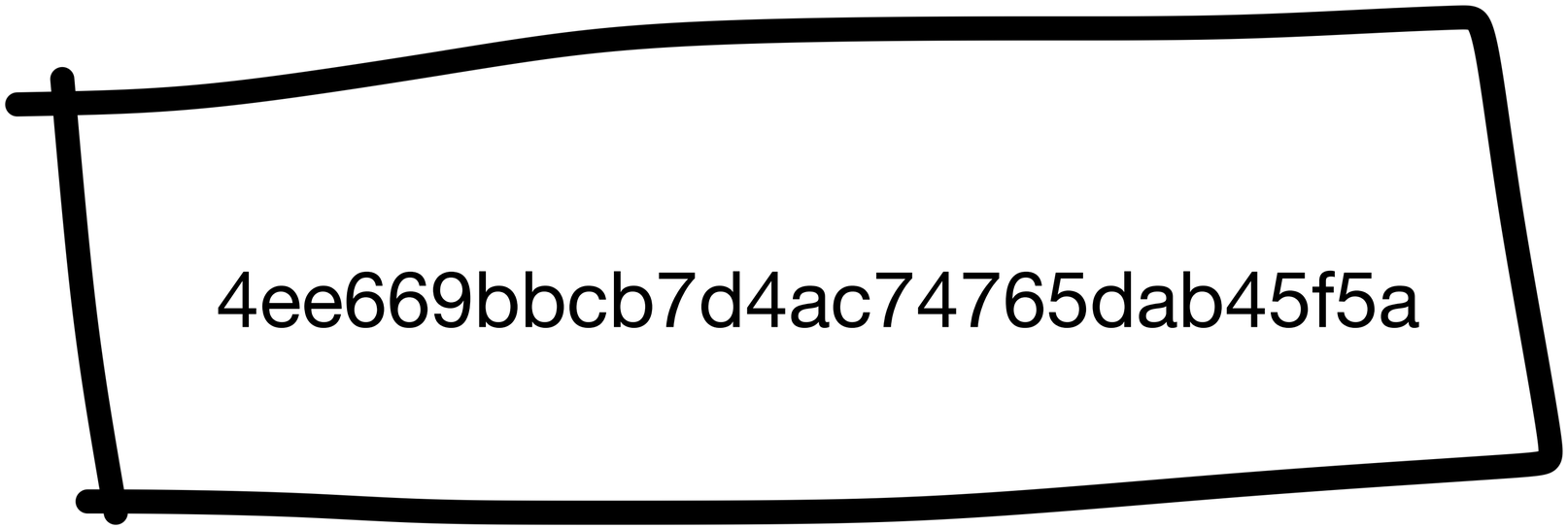
Merkle Tree

What is it?



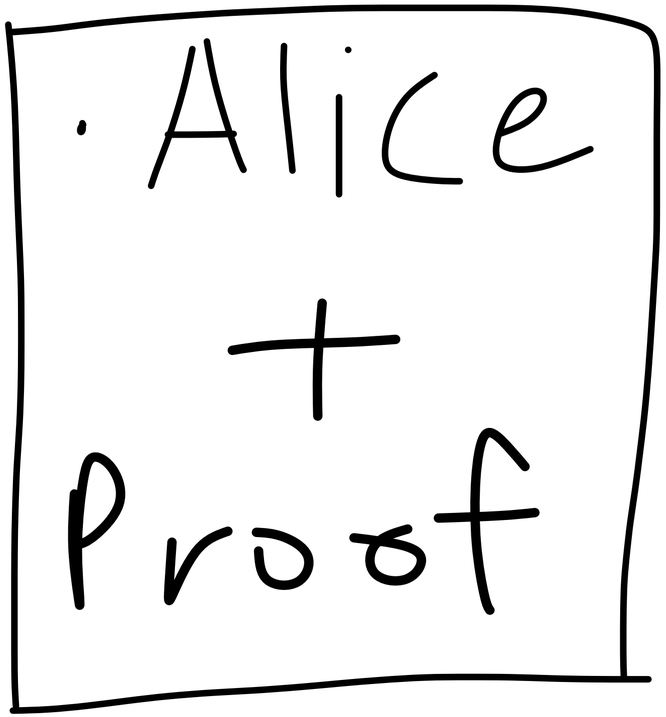
Merkle Tree

What is it?



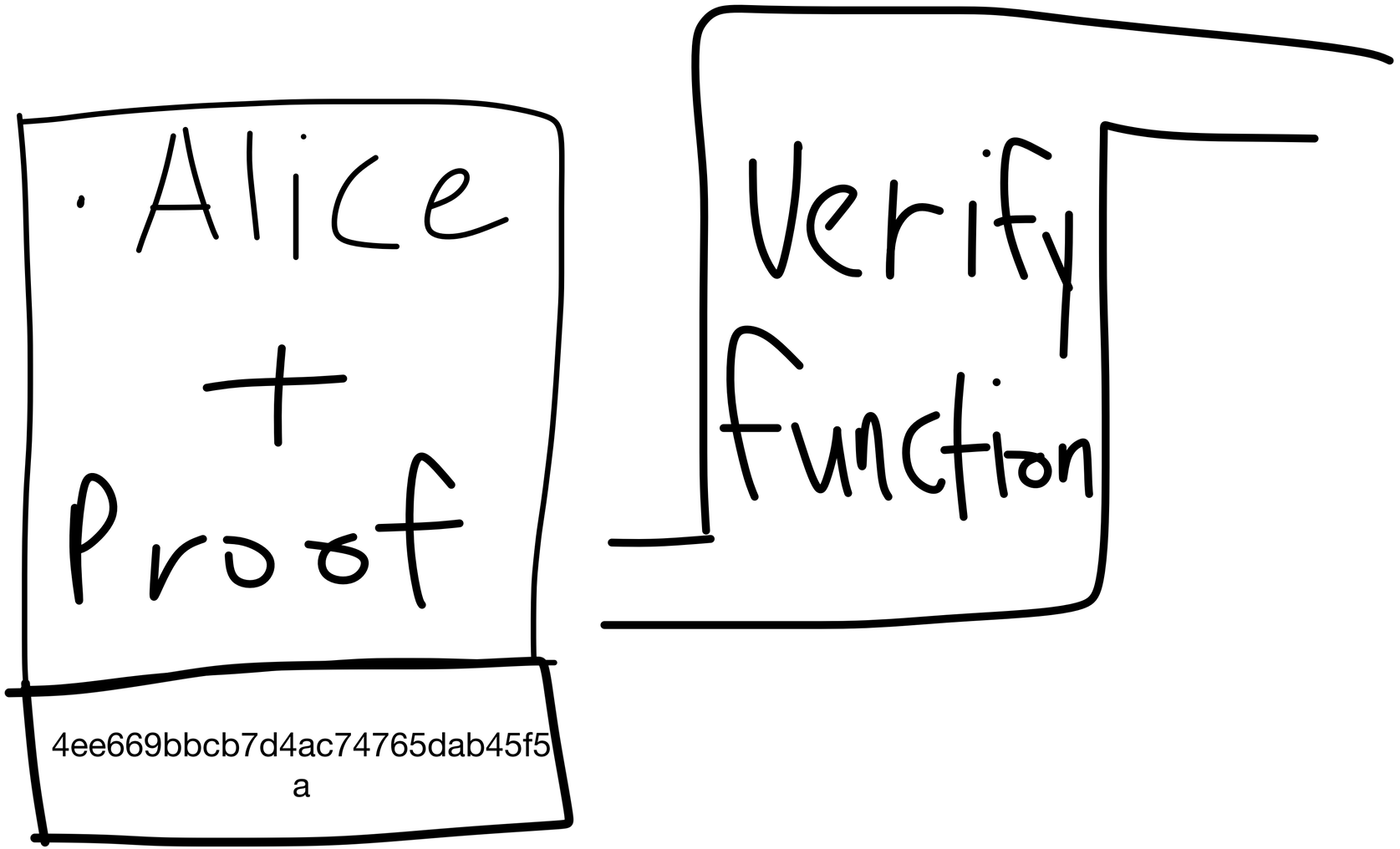
Merkle Tree

What is it?



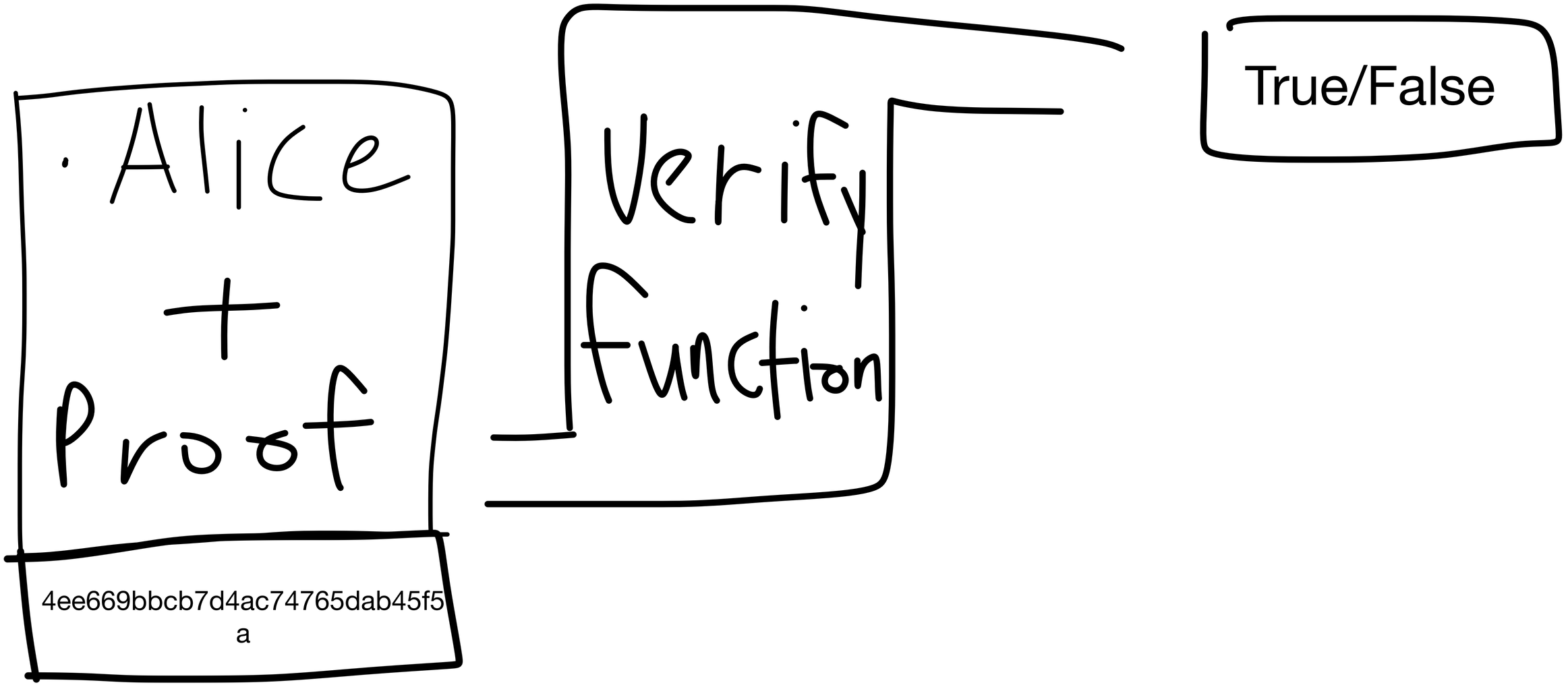
Merkle Tree

What is it?



Merkle Tree

What is it?



Utreexo?

What is it?

- A fancy merkle tree proposed by Tadge Dryja
- Makes pruned nodes smaller

Pruned nodes?

What are pruned nodes?

Small nodes

- Nodes that only keep the UTXO set

What are pruned nodes?

Small nodes

- Nodes that only keep the UTXO set
- Historical block data is removed (~500GB)

What are pruned nodes?

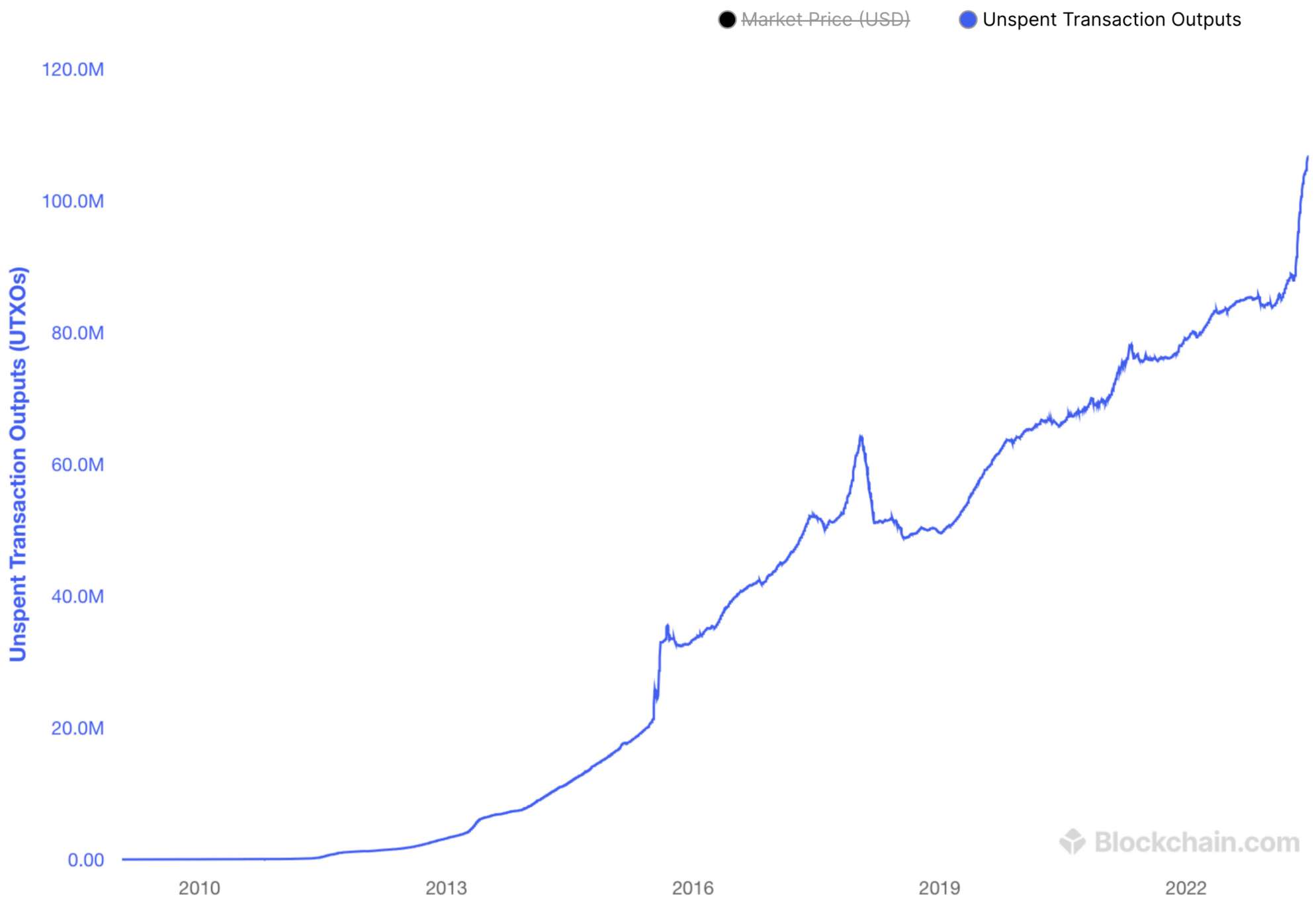
Small nodes

- Nodes that only keep the UTXO set
- Historical block data is removed (~500GB)
- Total size around 6GBs and growing

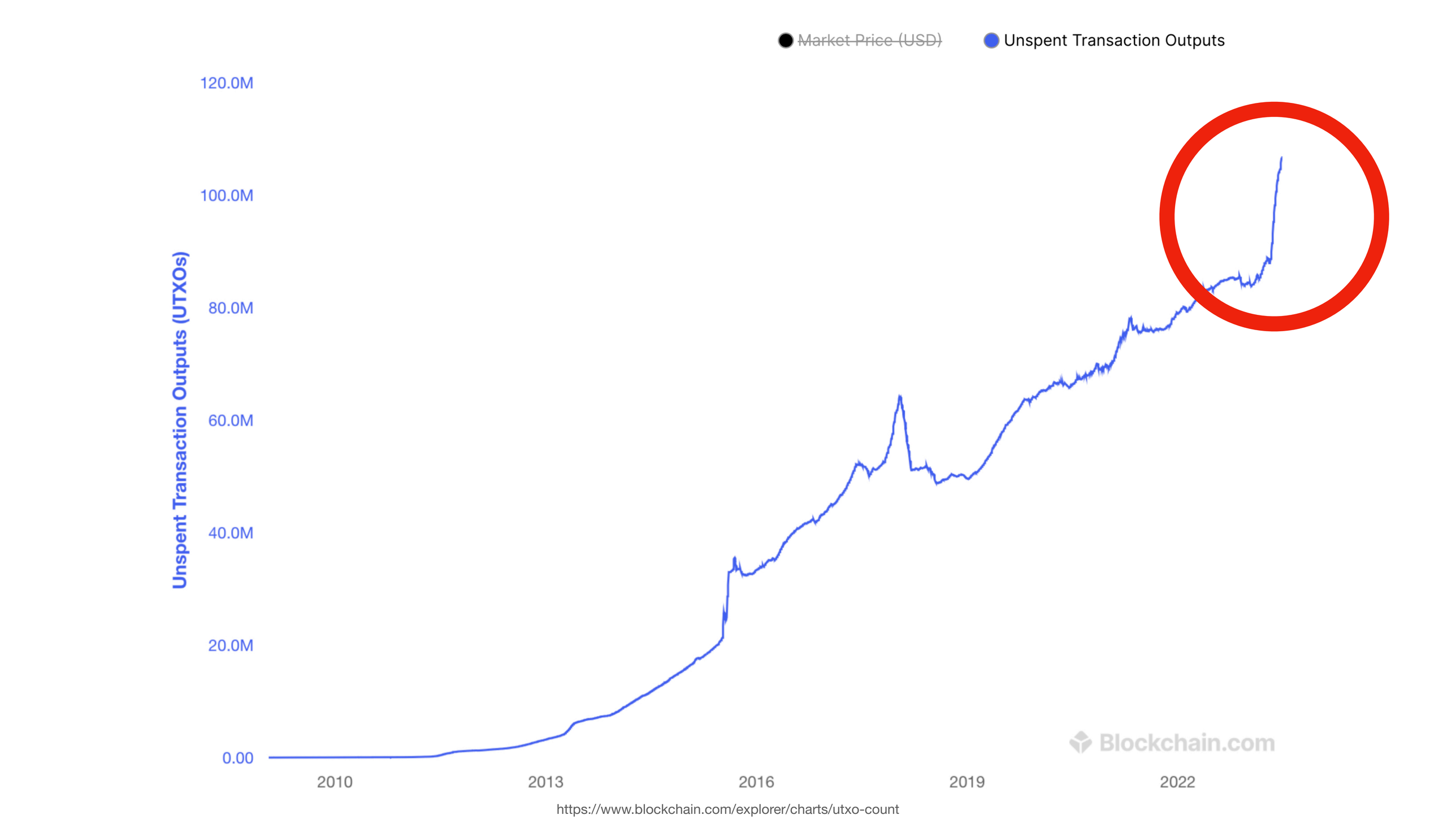
The UTXO set

Why merklize it?

- Puts a bound to the UTXO set growth



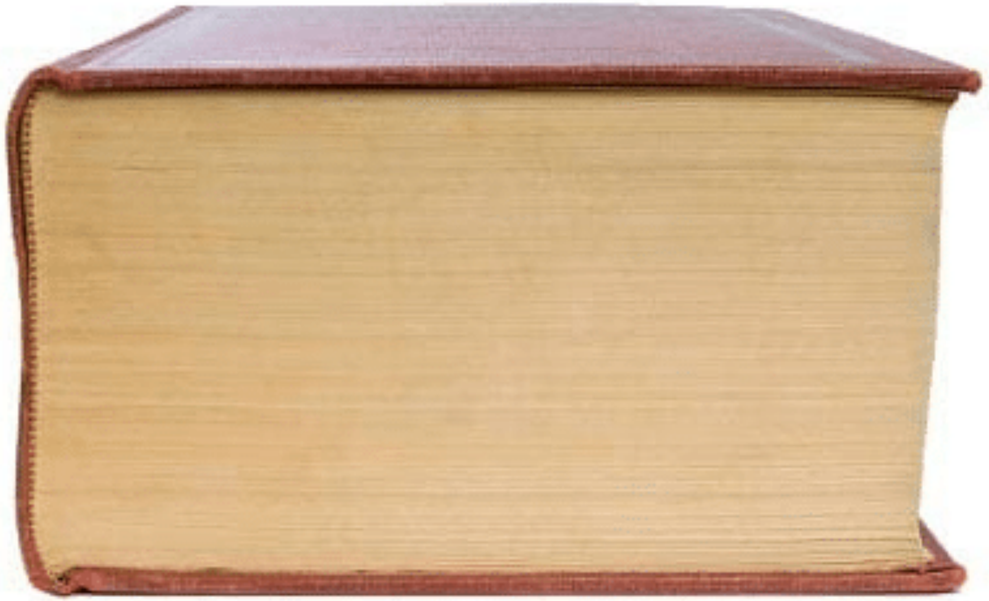
https://www.blockchain.com/explorer/charts/utxo-count



https://www.blockchain.com/explorer/charts/utxo-count

O(N) -> O(log2 N)

Pruned nodes



Without Utreexo

With Utreexo

N=8 billion UTXOs, at 59B per entry

472GB -> 416B

Outpoint: 33B, p2pkh script: 21B, height+iscoinbase: 4, amount: 1

The UTXO set

Why merklize it?

- Puts a bound to the UTXO set growth
- Allows for tiny nodes

6.3G chainstate/

384B chainstate/

Tiny Utreexo full nodes

We can run them anywhere

- Anywhere with *low* storage

Tiny Utreexo full nodes

We can run them anywhere

- Anywhere with *low* storage
- Routers, web browsers, etc.

Tiny Utreexo full nodes

We can run them anywhere

- Anywhere with *low* storage
- Routers, web browsers, etc.
- First good client side Bitcoin nodes

Web Browser full Nodes

Theoretically possible

- Easily accessible. Grandma friendly.

Web Browser full Nodes

Theoretically possible

- Easily accessible. Grandma friendly.
- A full node client for users.

Web Browser full Nodes

Theoretically possible

- Easily accessible. Grandma friendly.
- A full node client for users.
- Not currently implemented but maybe?

Router full Nodes

Always connected full nodes

- Plug & play. Your node is pre-installed on the router.

Router full Nodes

Always connected full nodes

- Plug & play. Your node is pre-installed on the router.
- Everyone has a router so why not have a node in them?

Endless possibilities but I’m not creative enough

Immediate node bootstrap

AssumeUTXO

Quick way to sync up

- Receive a UTXO set hash commitment with the binary

AssumeUTXO

Quick way to sync up

- Receive a UTXO set hash commitment with the binary
- Download the ~6GB of UTXO set from torrents

AssumeUTXO

Quick way to sync up

- Receive a UTXO set hash commitment with the binary
- Download the ~6GB of UTXO set from torrents
- Download blocks from peers

AssumeUtreexo

Quicker way to sync up

- Receive the UTXO set with the binary

AssumeUtreexo

Quicker way to sync up

- Receive the UTXO set with the binary
- Download blocks/tx from peers


Download Bitcoin Core

The UTXO set

Why merklize it?

- Puts a bound to the UTXO set growth
- Allows for tiny nodes
- Faster block validation

Tiny full nodes

Why these can make ibd faster

- Anywhere with *slow* storage

Tiny full nodes

Why these can make ibd faster

- Anywhere with *slow* storage
- We can cache the entire UTXO set representation in memory

Tiny full nodes

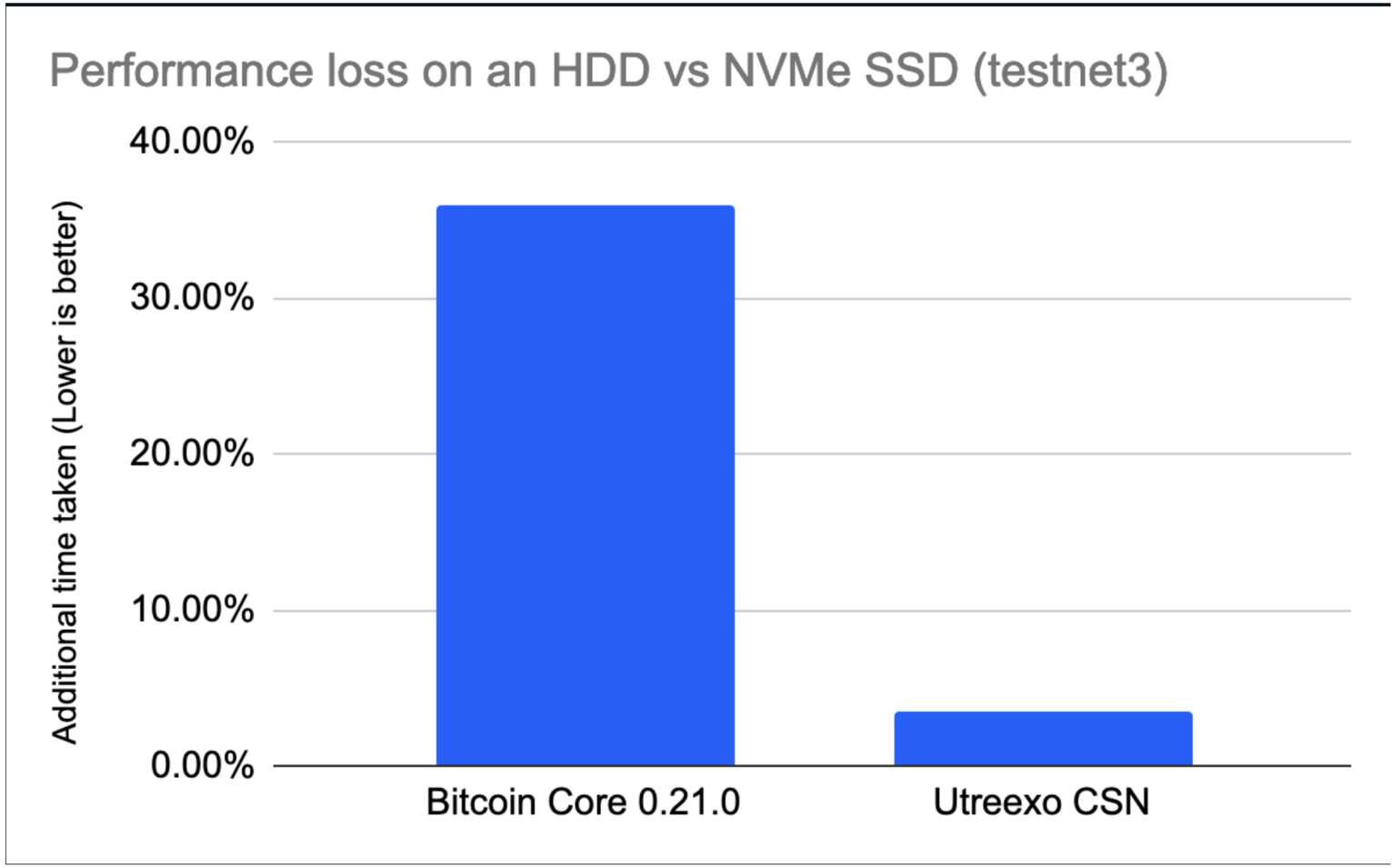
Why these can make ibd faster

- Anywhere with *slow* storage
- We can cache the entire UTXO set representation in memory
- Any computer with slow HDD now has a faster ibd

Tiny full nodes

Why these can make ibd faster

- Less performance degradation for pruned nodes



Tiny full nodes

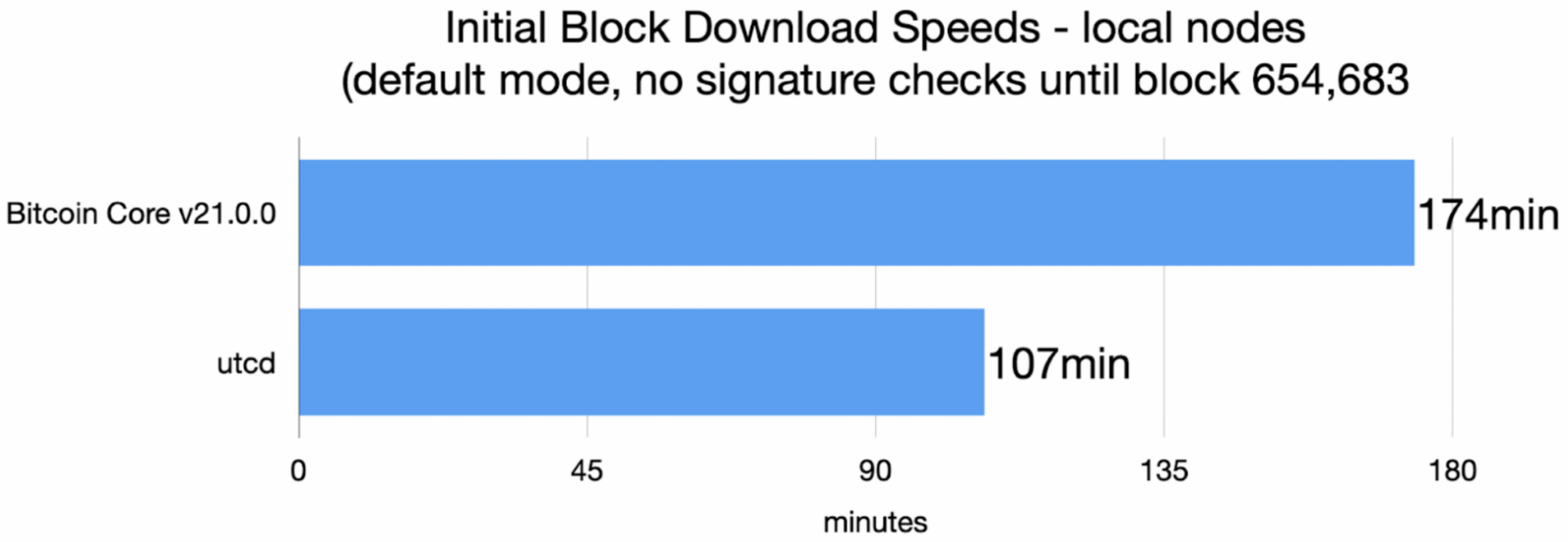
Why these can make ibd faster

- Tiny UTXO set representation allows for a parallel block validation.

Tiny full nodes

Why these can make ibd faster

- Tiny UTXO set representation allows for a parallel block validation.
- Unoptimized implementation ~63% faster than Bitcoin Core

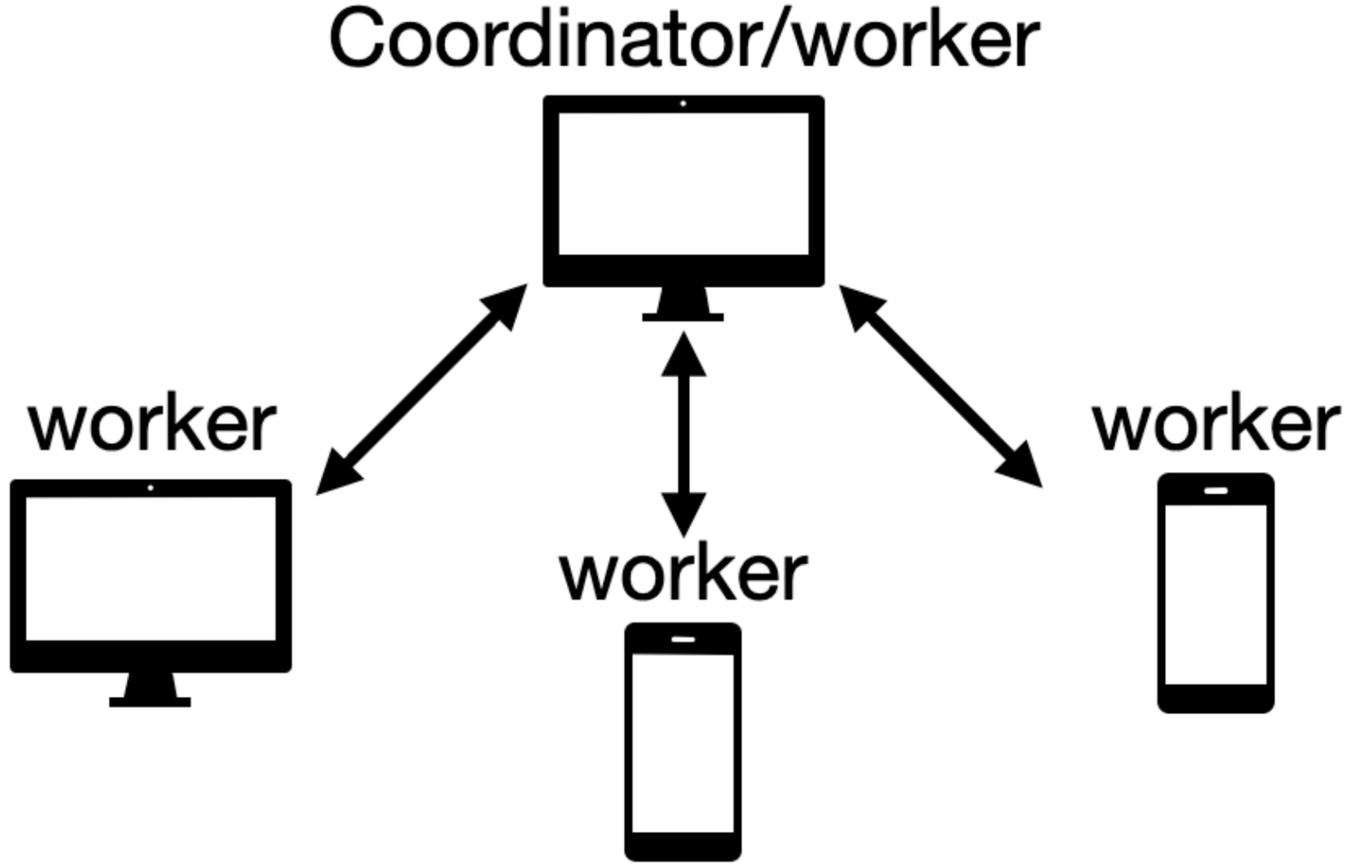


https://blog.bitmex.com/faster-blockchain-validation-with-utreexo-accumulators/

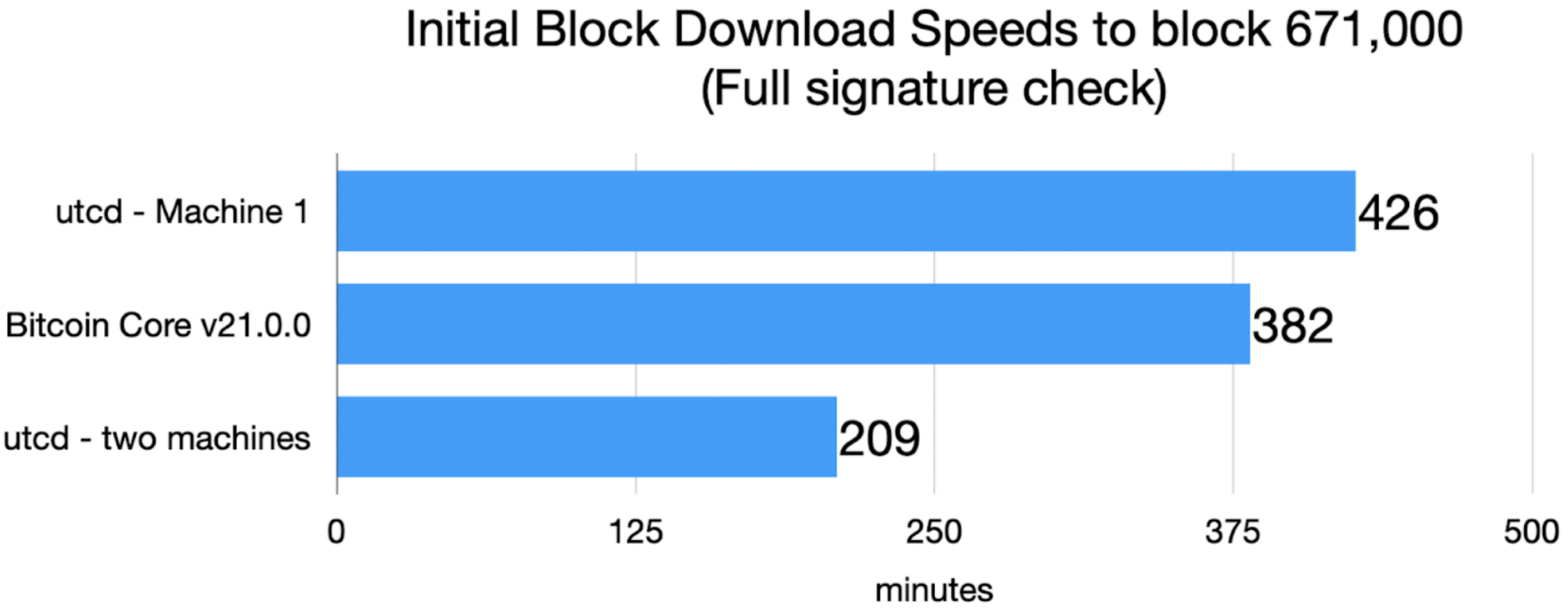
Multiple device sync

Moar processing power

- Take advantage of all the processing power you have



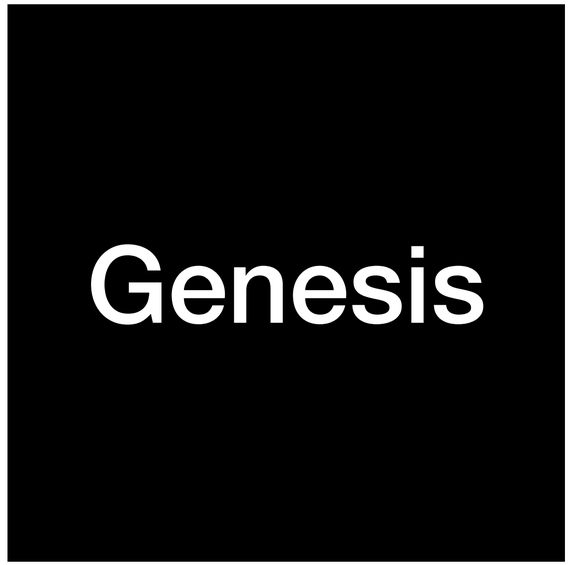
https://blog.bitmex.com/faster-blockchain-validation-with-utreexo-accumulators/



https://blog.bitmex.com/faster-blockchain-validation-with-utreexo-accumulators/

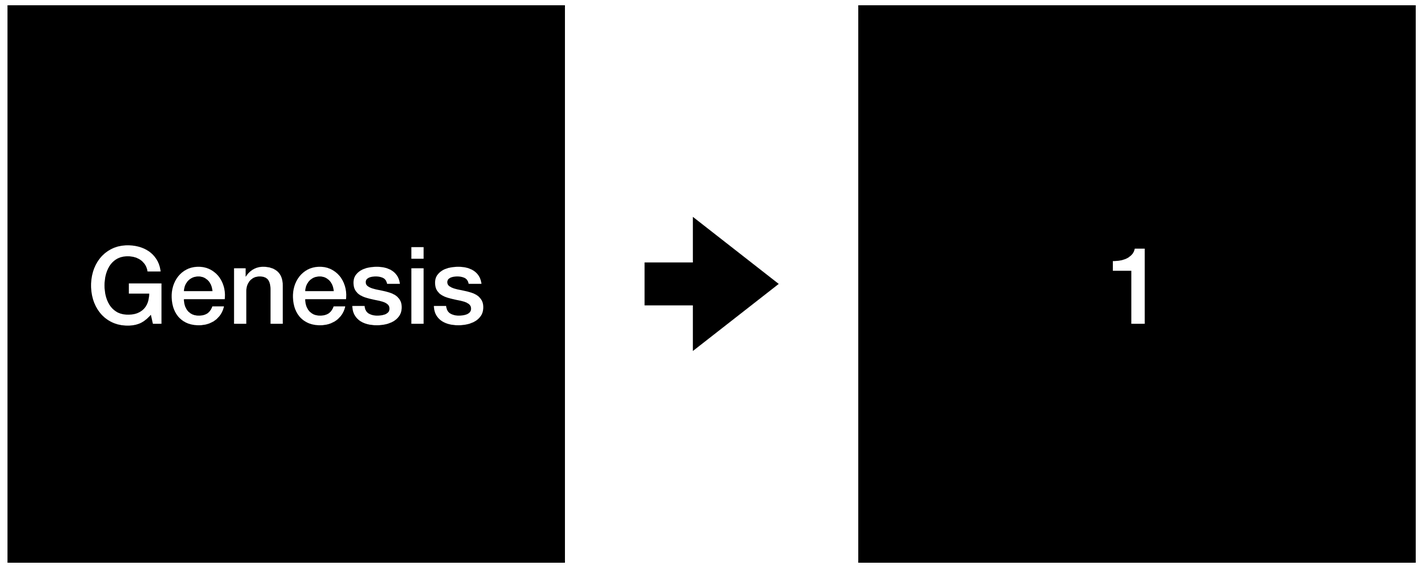
Current block validation

Sequential validation



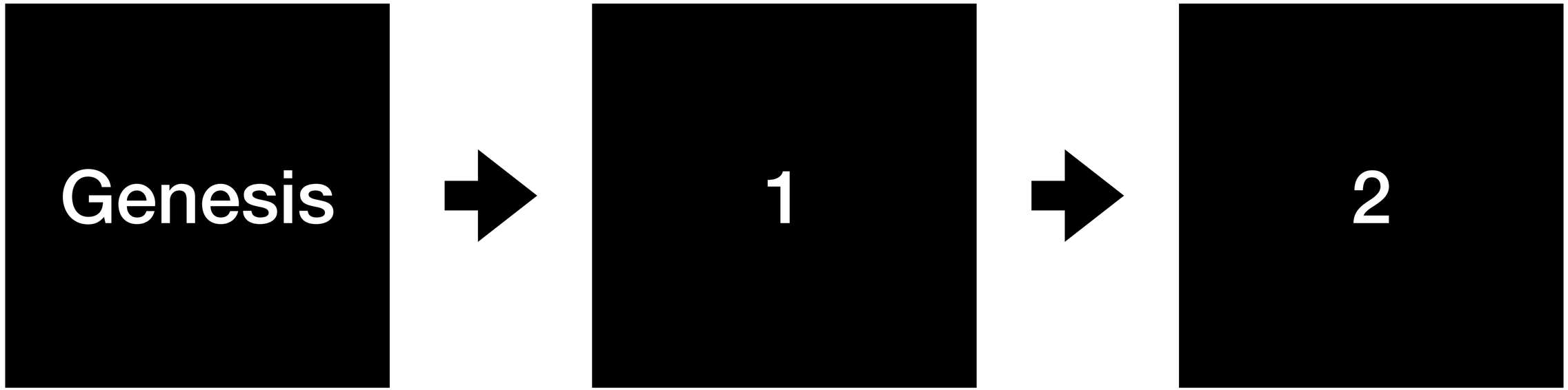
Current block validation

Sequential validation



Current block validation

Sequential validation



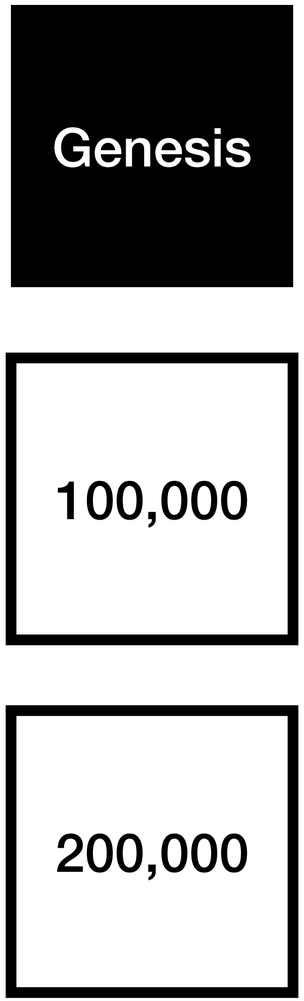
Parallel block validation

Replay blocks in any order

- Utreexo roots committed into the binary
- Process blocks starting from any of the roots that are committed
- https://github.com/mit-dci/utcd/blob/master/chaincfg/mainnetroots.go

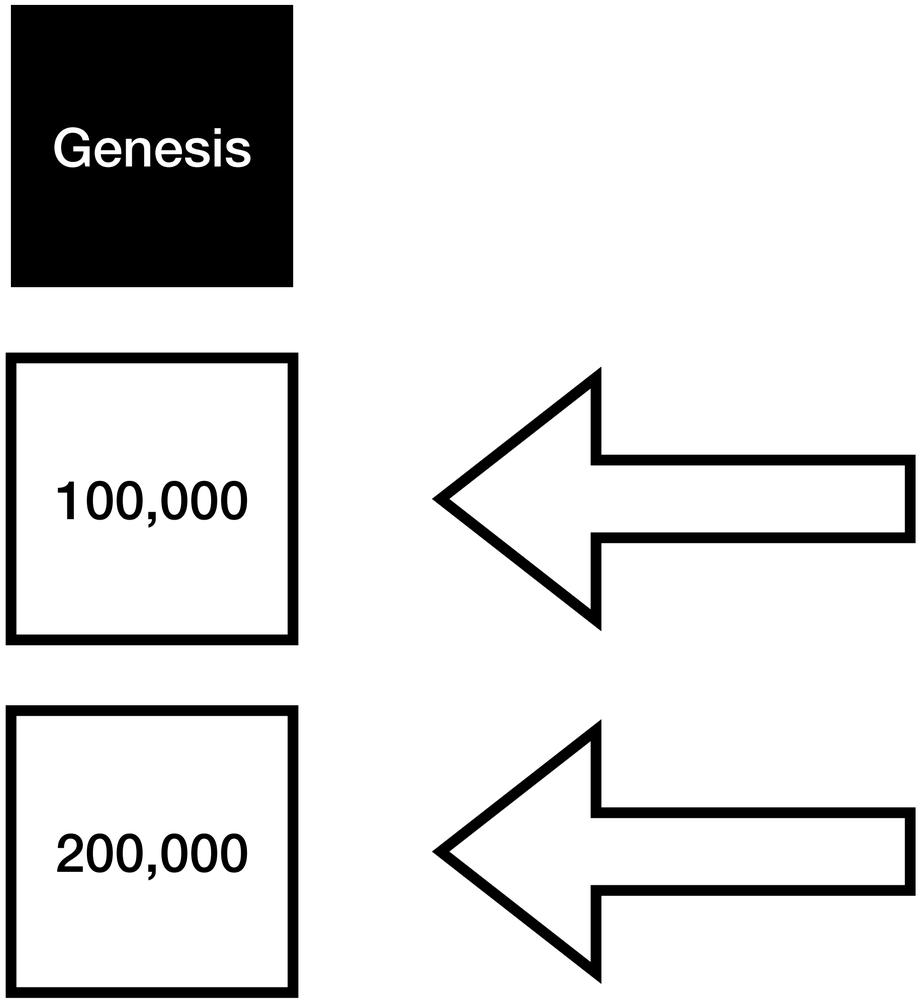
With Utreexo

Efficient parallel validation



With Utreexo

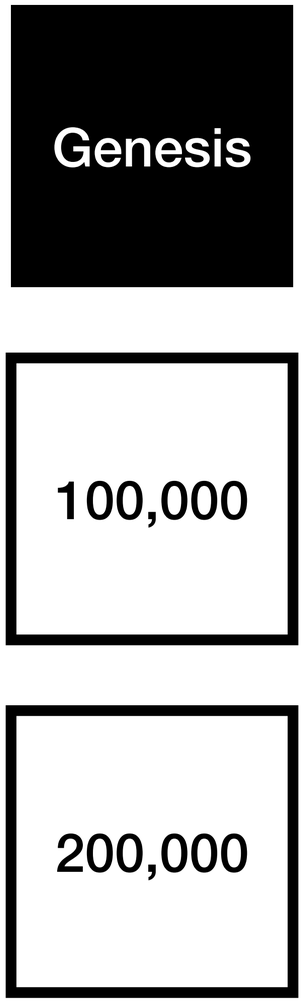
Efficient parallel validation



Untrusted roots at the start

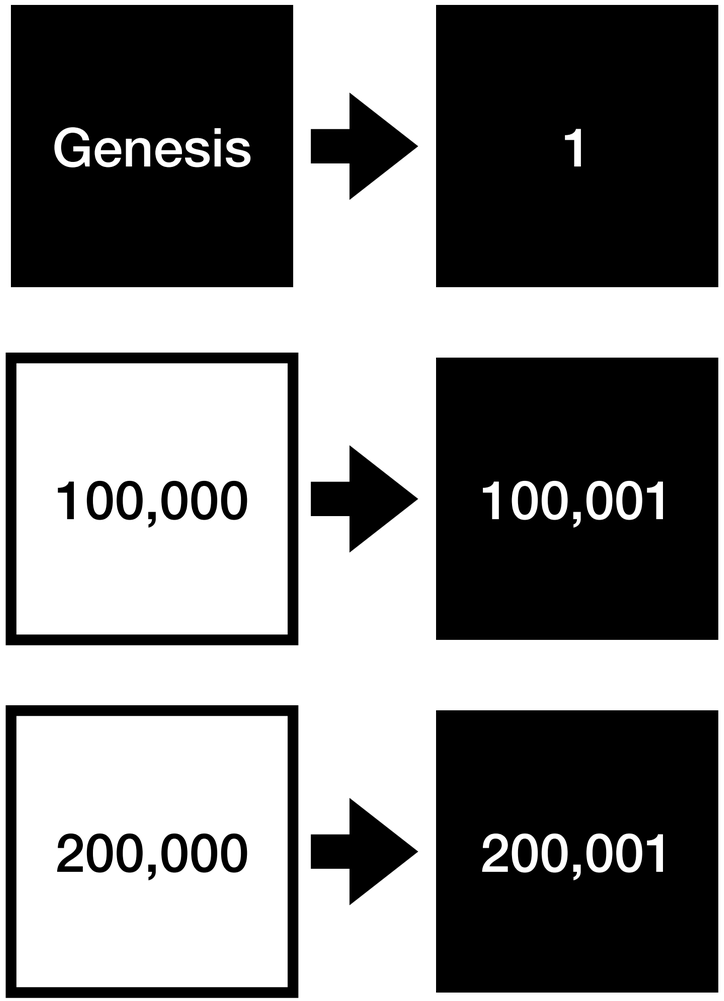
With Utreexo

Efficient parallel validation



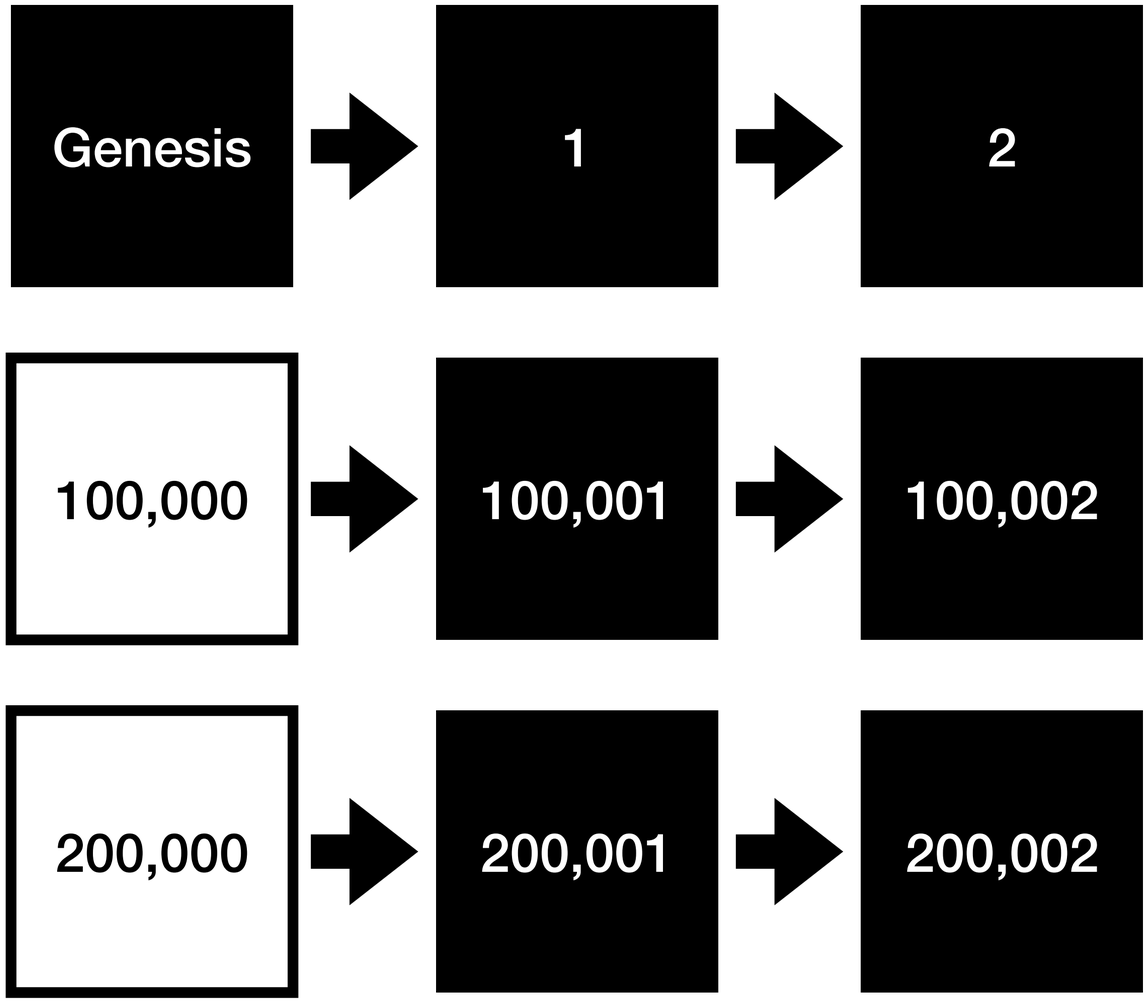
With Utreexo

Efficient parallel validation



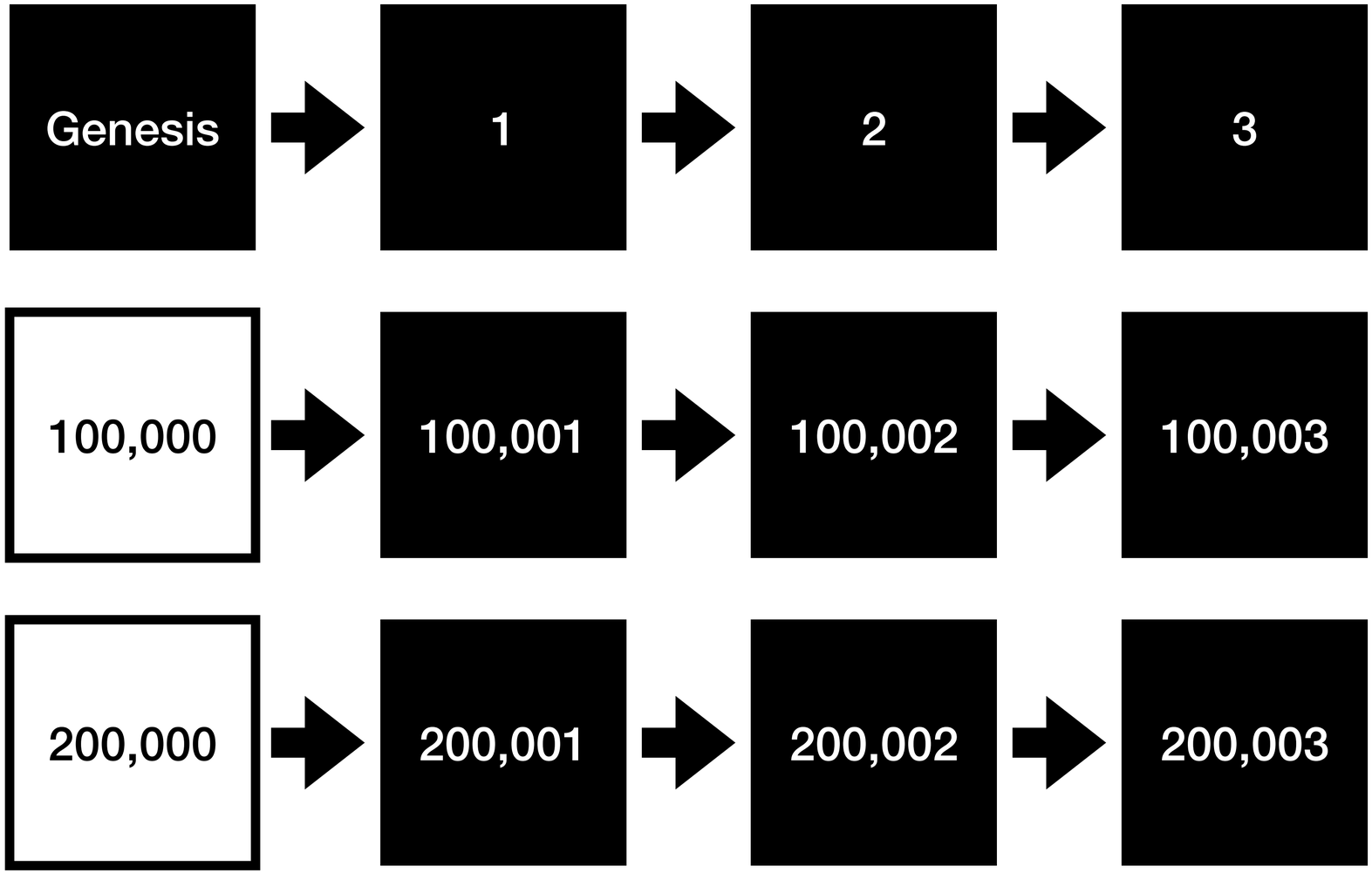
With Utreexo

Efficient parallel validation



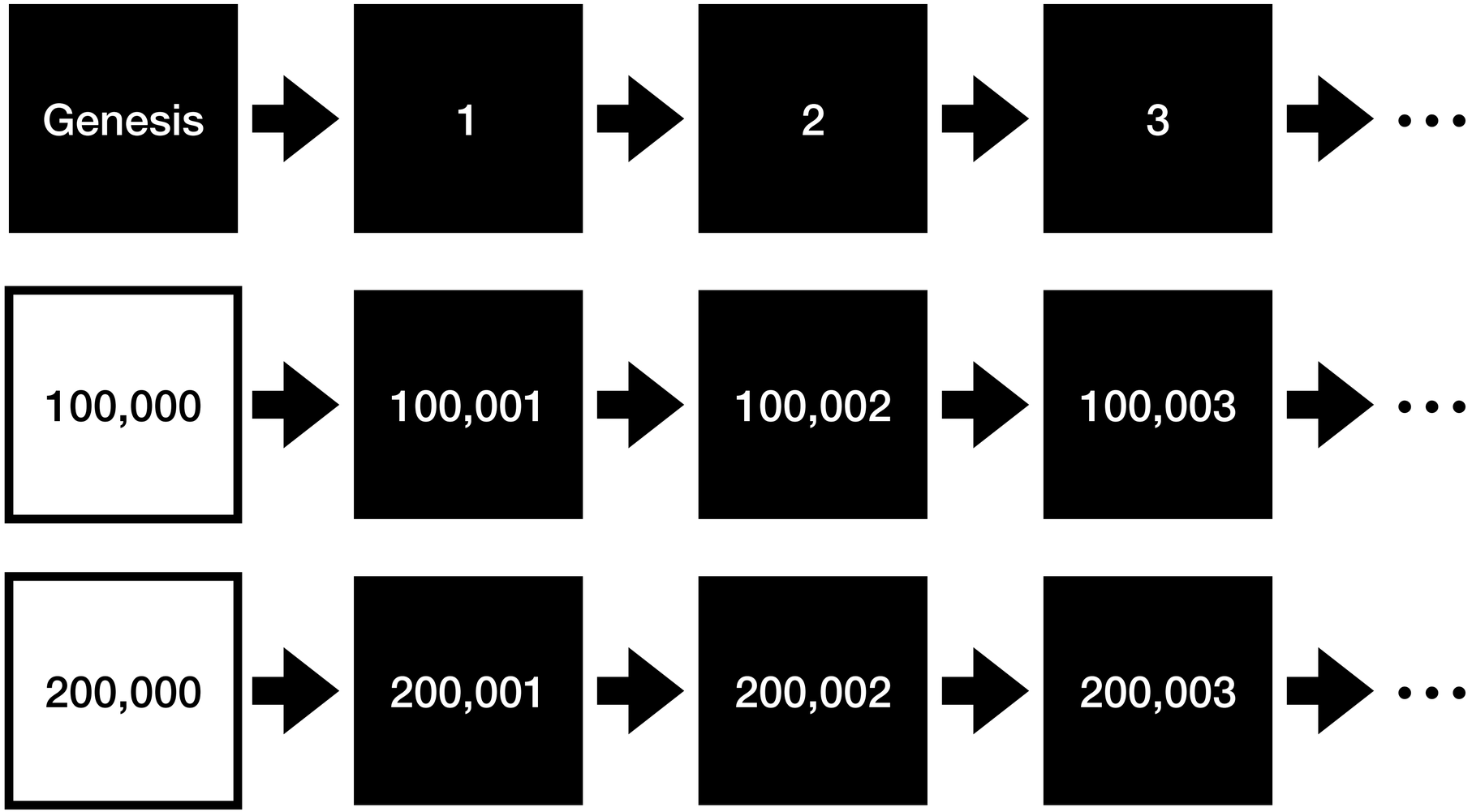
With Utreexo

Efficient parallel validation



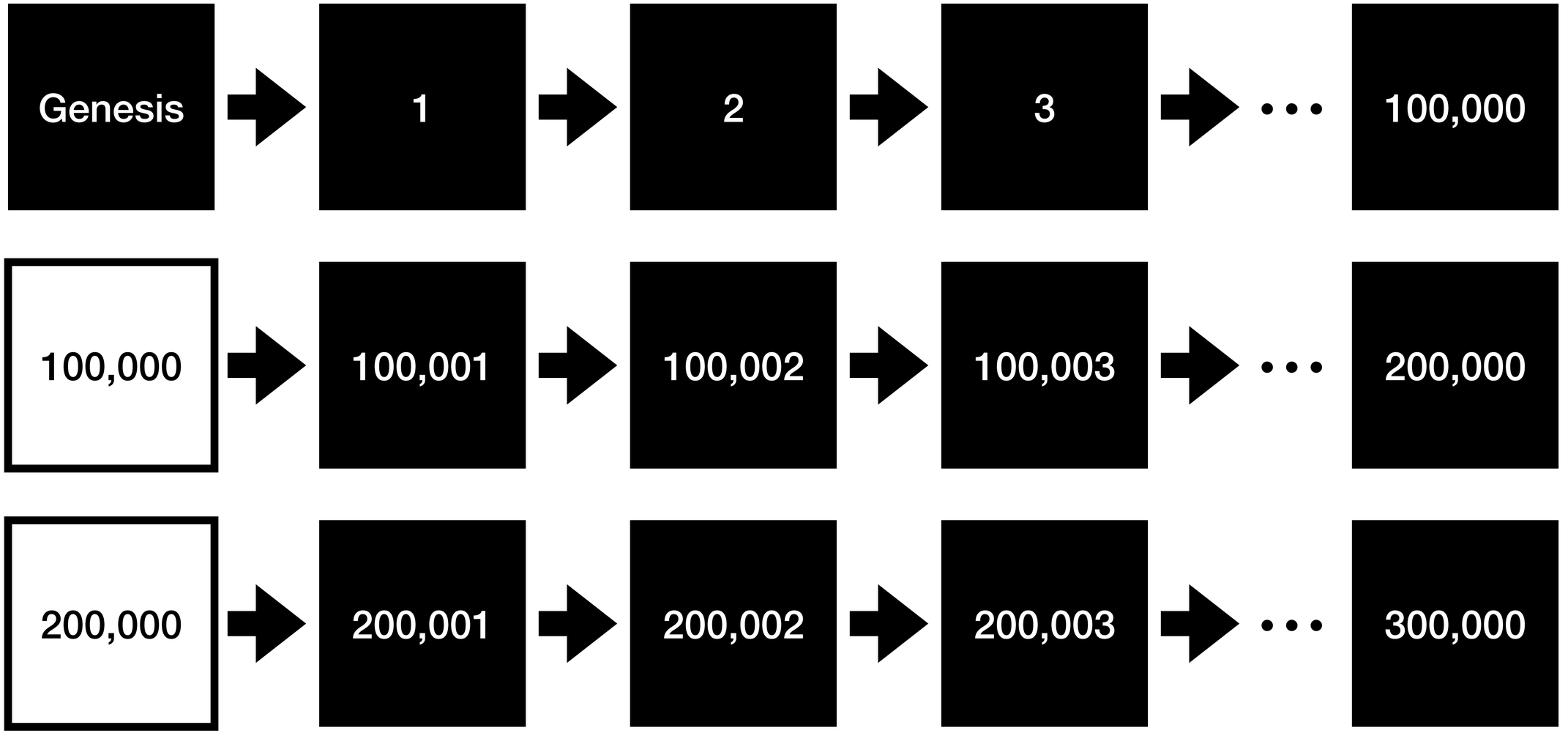
With Utreexo

Efficient parallel validation



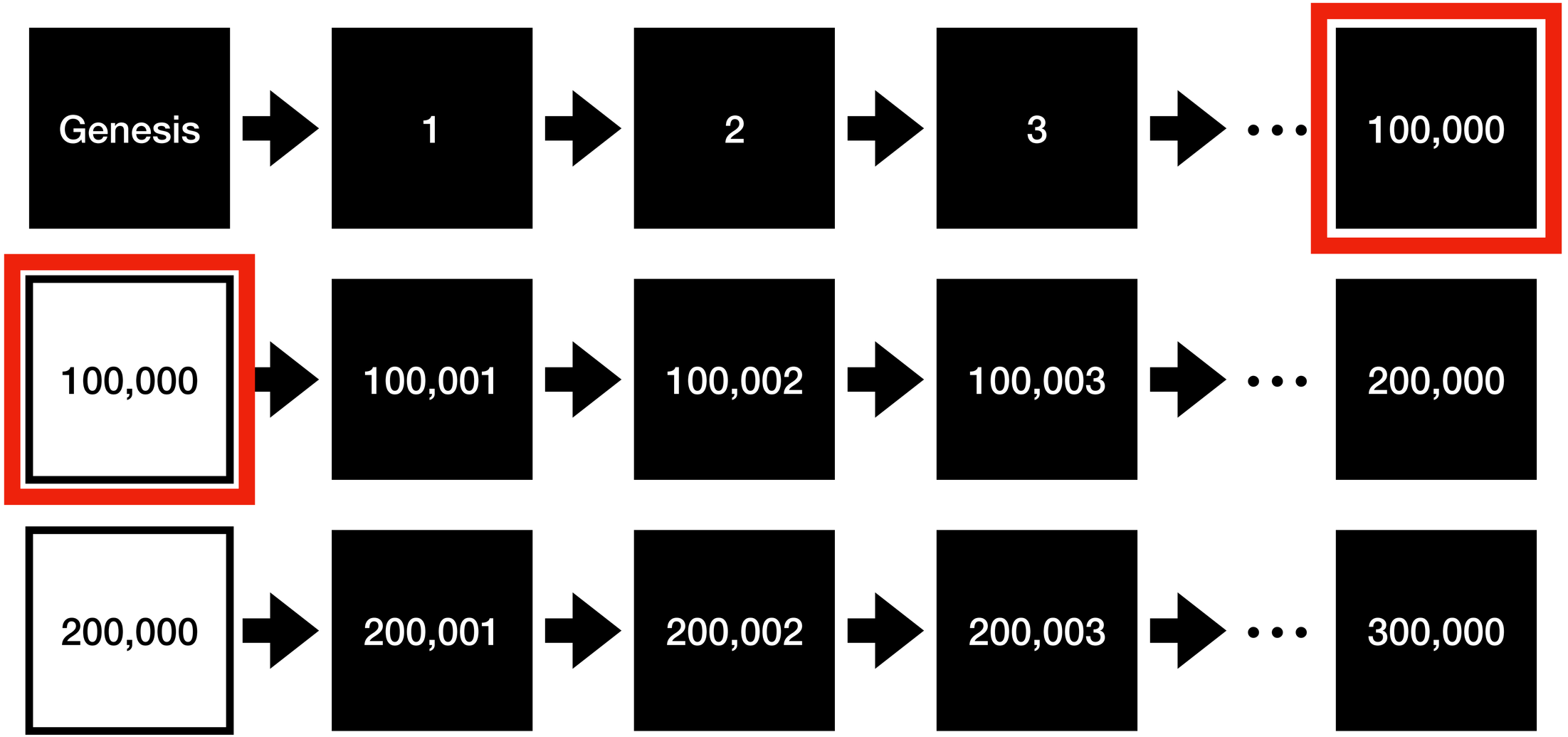
With Utreexo

Efficient parallel validation



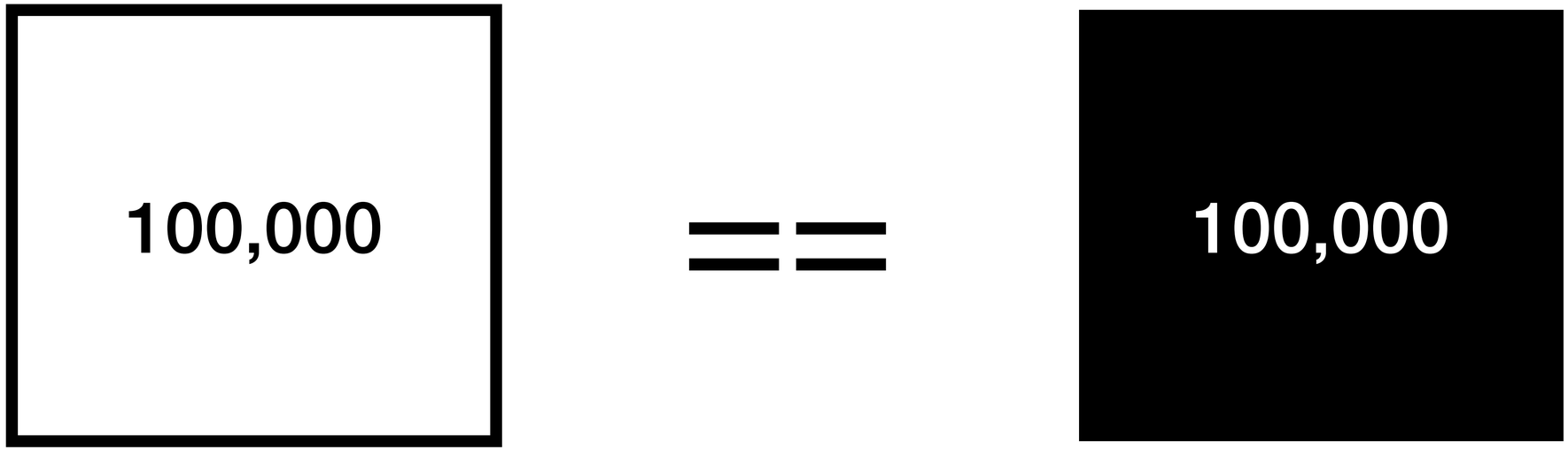
With Utreexo

Efficient parallel validation



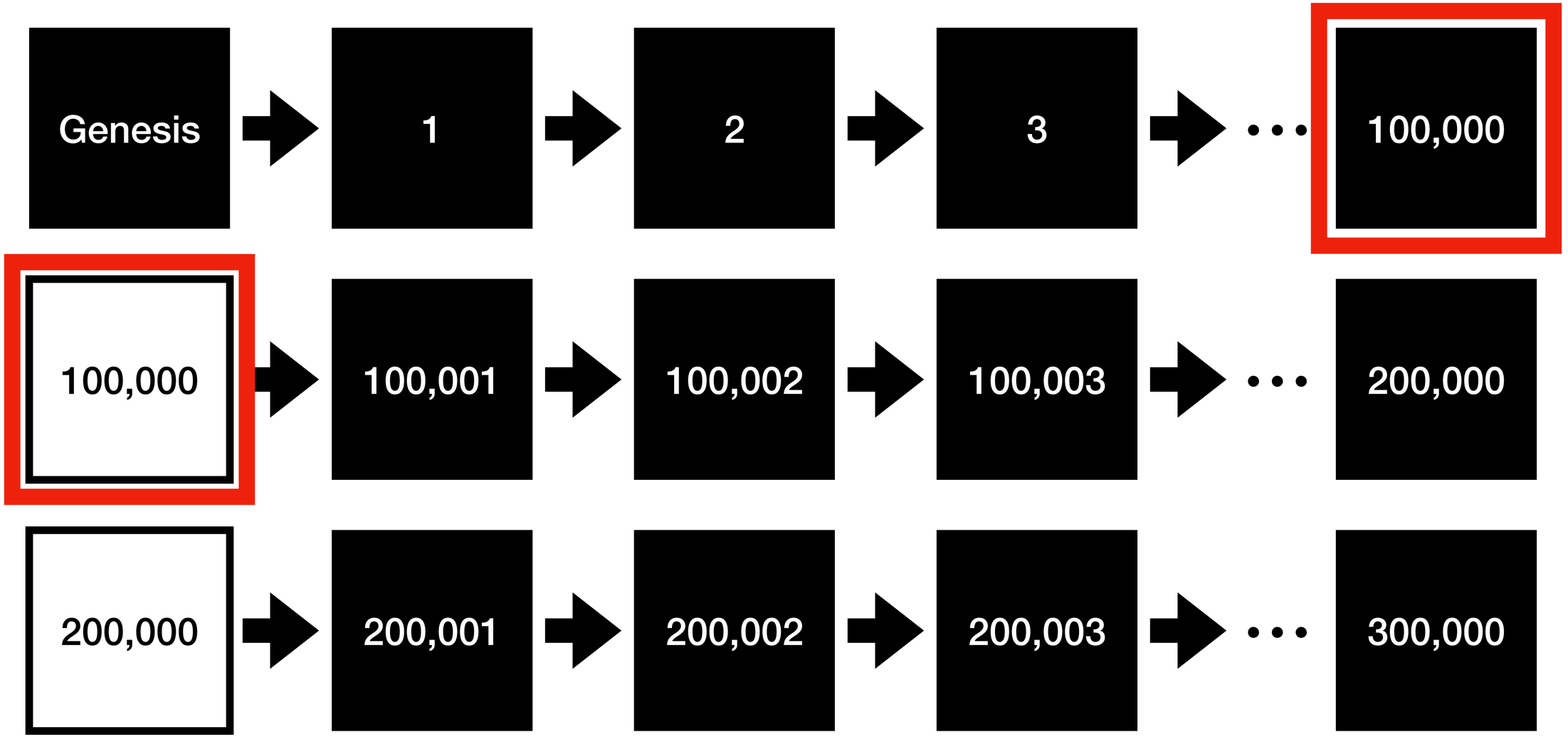
Compare roots

Root at 100,000 becomes trusted if equal



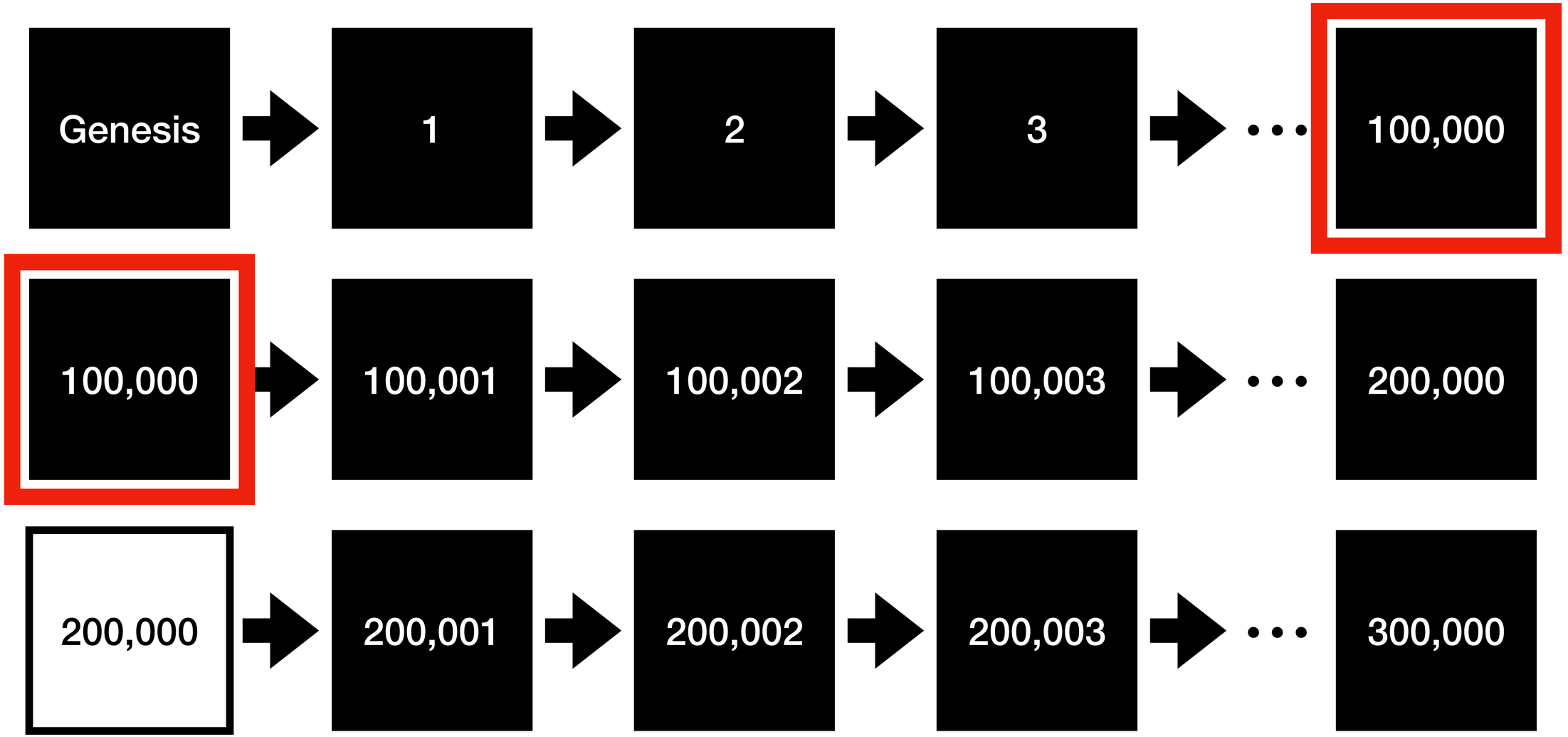
With Utreexo

Efficient parallel validation



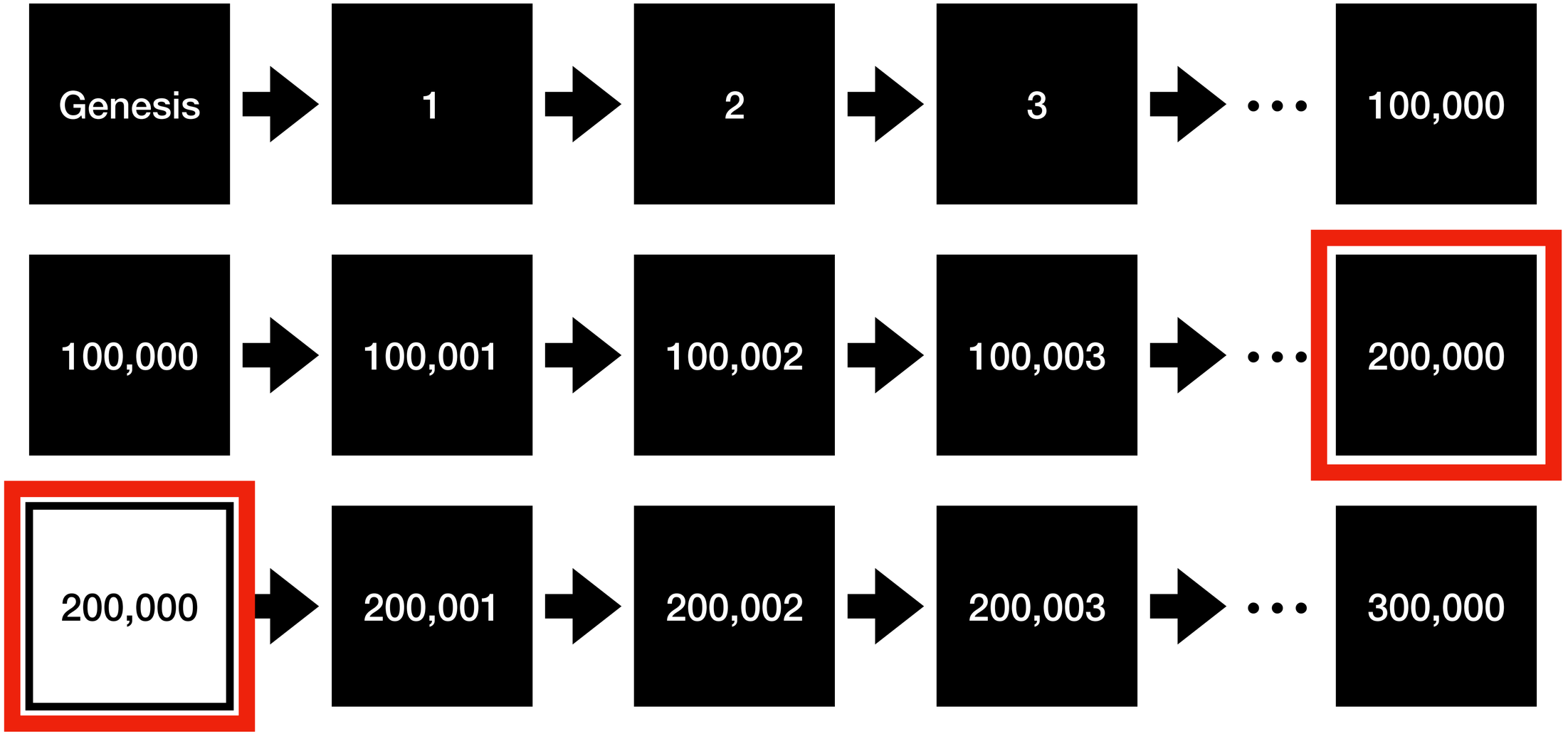
With Utreexo

Efficient parallel validation



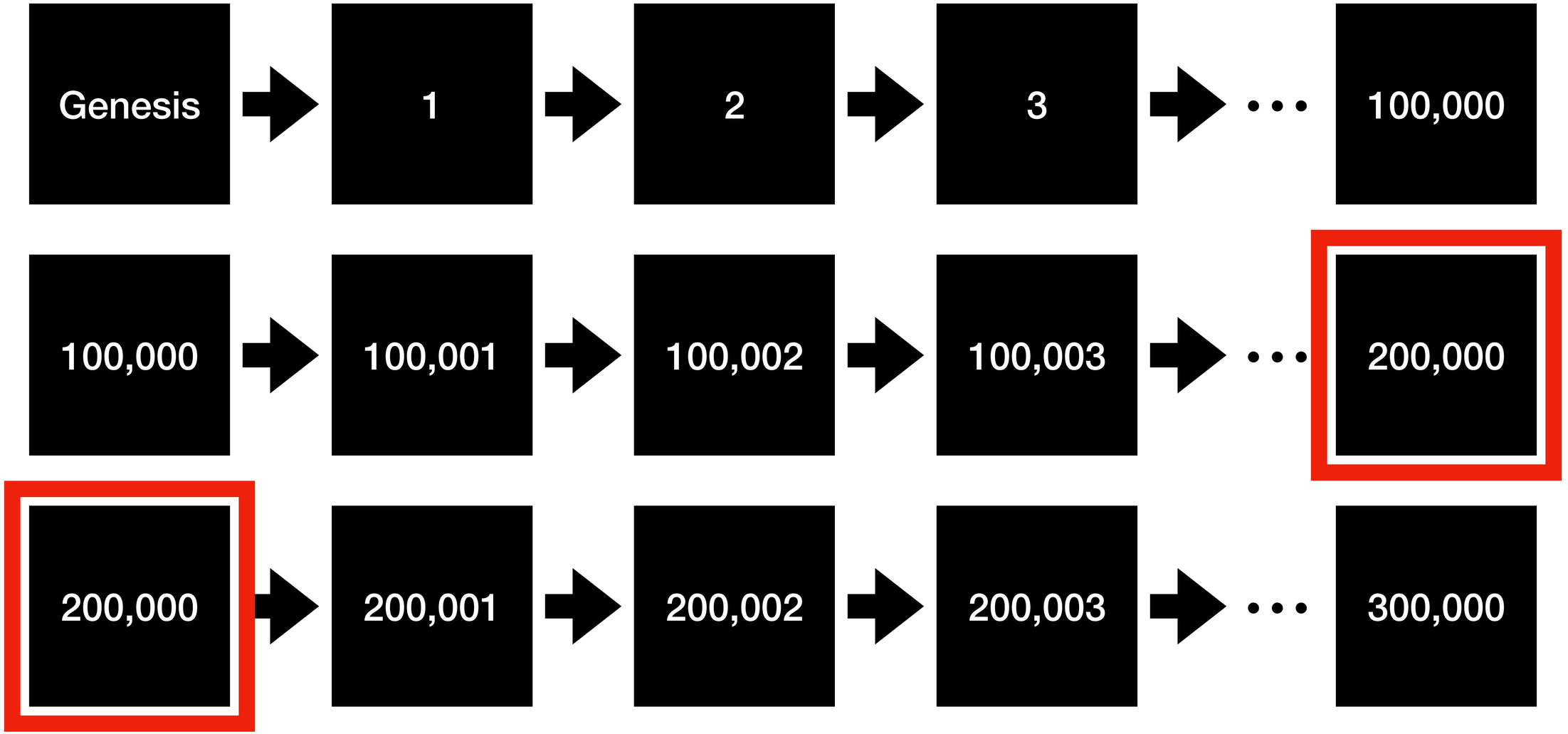
With Utreexo

Efficient parallel validation



With Utreexo

Efficient parallel validation



Efficient Parallel Validation

SHA256 🐇 > Map access 🐢

What’s the catch?

Trade-off between internet usage for less disk usage

More things to download

Need to download Utreexo proofs with blocks & txs

- 1x things to download for ibd

More things to download

Need to download Utreexo proofs with blocks & txs

- 1x things to download for ibd
- 3x things to download after you sync up

More things to download

Need to download Utreexo proofs with blocks & txs

- 1x things to download for ibd
- 3x things to download after you sync up
- We know of ways to reduce this. Implementation in progress…

More things to download

Need to download Utreexo proofs with blocks & txs

- None of this matters because we're in Taipei!
- Everyone benefits from Utreexo in Taipei

Disk usage is very slow

So so slow

- Random reads are slow, even for ssds

Disk usage is very slow

So so slow

- Random reads are slow, even for ssds
- Utreexo nodes only ever do sequential reads

Is Utreexo right for me?

Depends..

- Need to know where your bottleneck is.

Is Utreexo right for me?

Depends..

- Need to know where your bottleneck is.
- Optional for Bitcoin users. Use it only if it benefits you.

Is Utreexo right for me?

Depends..

- Need to know where your bottleneck is.
- Optional for Bitcoin users. Use it only if it benefits you.
- Right for you if you live in Taipei

How much internet would I need?

Depends..

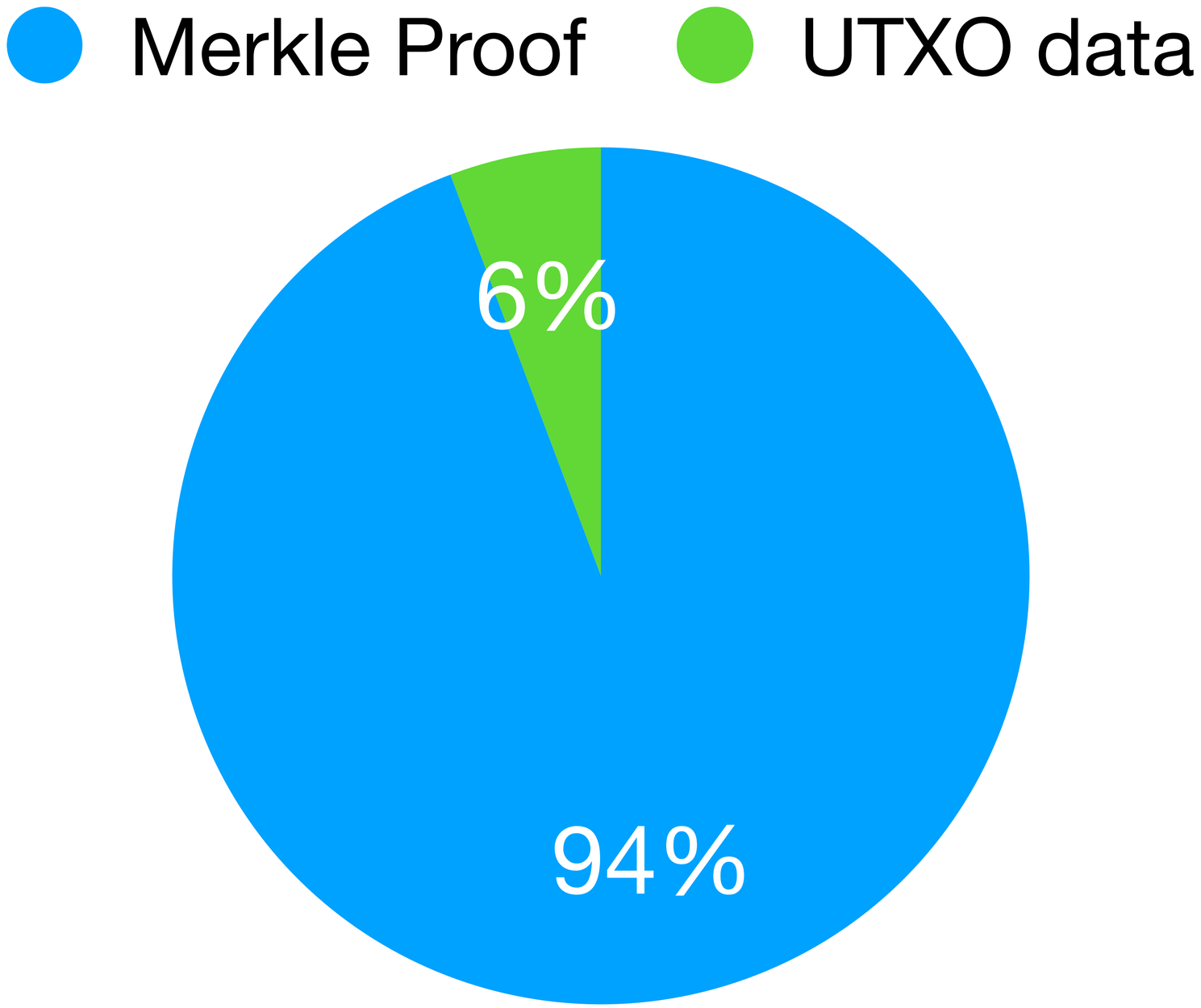
- Utreexo hurts
- Good Computer with terrible internet

- Utreexo is great
- Bad Computer with good internet

Total overhead

Extra data that a utreexo node will download

- 364GB of extra data (as of block 710,000)
- Caching and batch proving multiple blocks is being explored



Measured at block height 710,000. 20GB for LeafData, 343GB for Merkle Proofs

Current Progress

github.com/utreexo

Main github organization

- Accumulator implementation in Go
- Full node with Utreexo
- Full node with electrum-personal-server capability. Full support for protocol v1.4.1.
- Pytreexo

github.com/mit-dci/rustreexo

Accumulator in Rust

- Accumulator implementation in Rust
- Currently missing bridge node capability (2023-05-30)

github.com/Davidson-Souza/Floresta

Full node in Rust

- Full node with Utreexo in Rust
- Supports electrum personal server capabilities

Progress towards end user readiness

Things done & things left to do

- Accumulator design
- Working full node
- Wallet support
- P2P protocol that supports caching
- Efficient mempool with Utreexo

Progress towards end user readiness

Things done & things left to do

- Accumulator design
- Working full node
- Wallet support
- P2P protocol that supports caching
- Efficient mempool with Utreexo

Progress towards end user readiness

Things done & things left to do

- Accumulator design
- Working full node
- Wallet support
- P2P protocol that supports caching
- Efficient mempool with Utreexo

Progress towards end user readiness

Things done & things left to do

- Accumulator design
- Working full node
- Wallet support
- P2P protocol that supports caching
- Efficient mempool with Utreexo

Rust implementation

Things going on the Rust side by Davidson

- Working compact utreexo node
- C bindings
- Python bindings
- Javascript bindings

How does it work?

Replace the UTXO set

What does the UTXO set do for bitcoin?

Role of the UTXO set

It let’s you

1. Add a UTXO
2. Delete a UTXO
3. Tell you the existence of a UTXO
4. Provide the data for verification

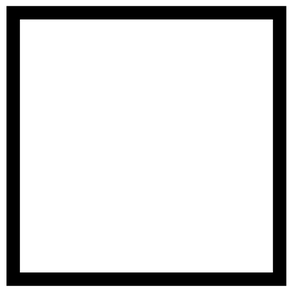
Role of the UTXO set

It let’s you

1. Add a UTXO
2. Delete a UTXO
3. Tell you the existence of a UTXO
4. Provide the data for verification

1 UTXO

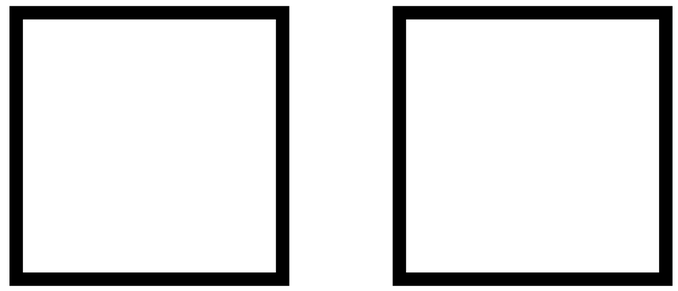
Single root



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

2 UTXOs

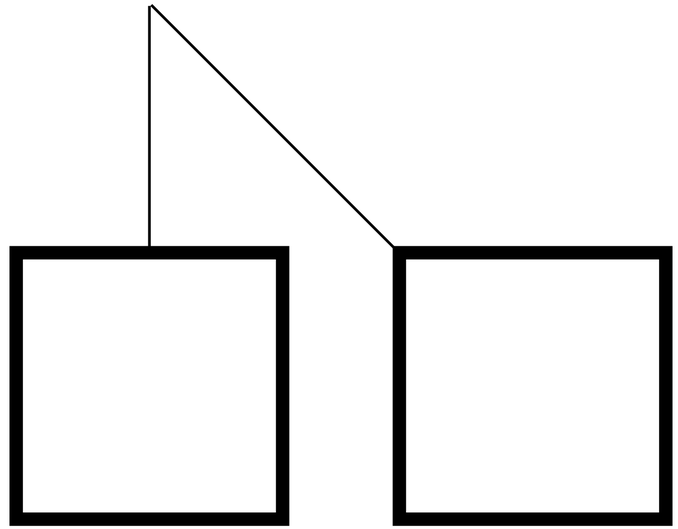
Add 1 more



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

2 UTXOs

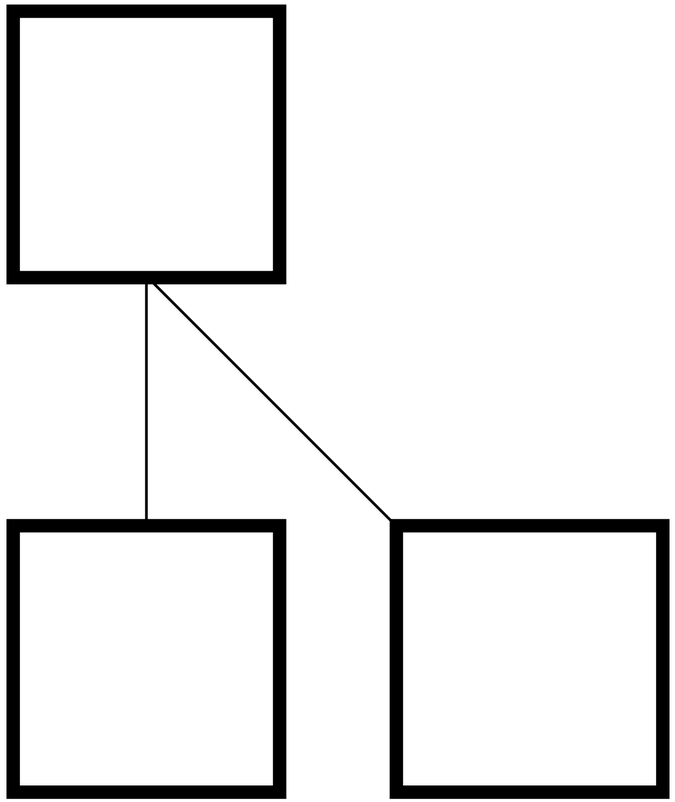
Concatenate



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

2 UTXOs

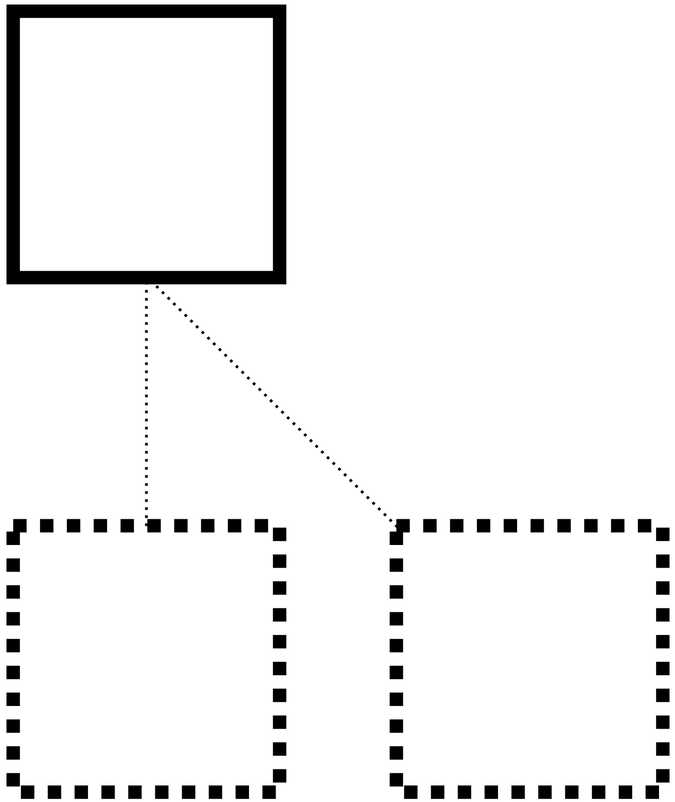
Hash to create a new root



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

2 UTXOs

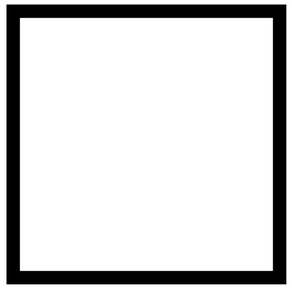
Can now delete the leaves



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

2 UTXOs

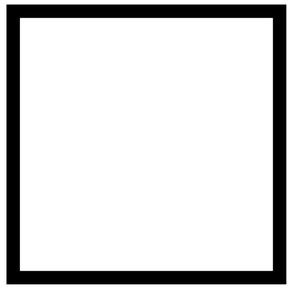
Only keep the roots



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

3 UTXOs

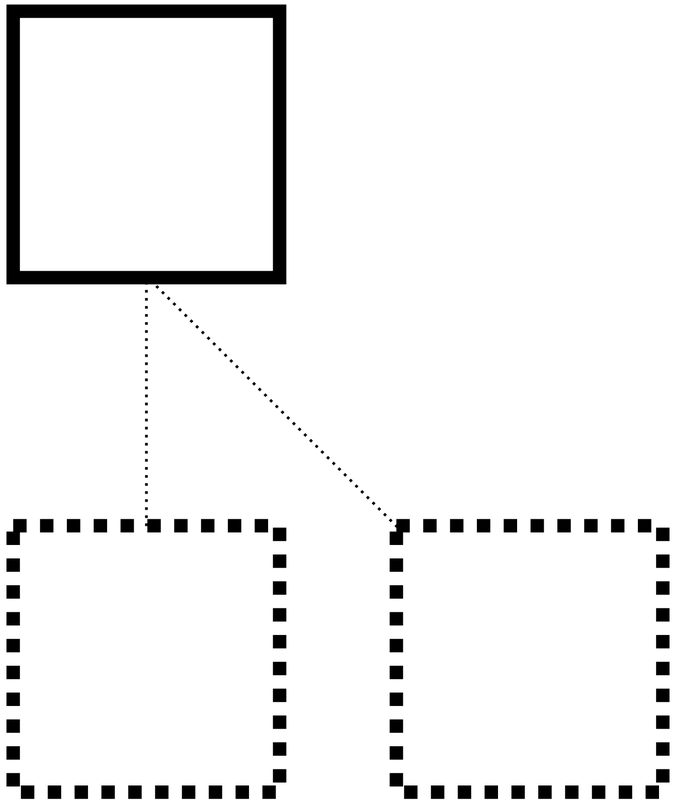
Add 1 more



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

3 UTXOs

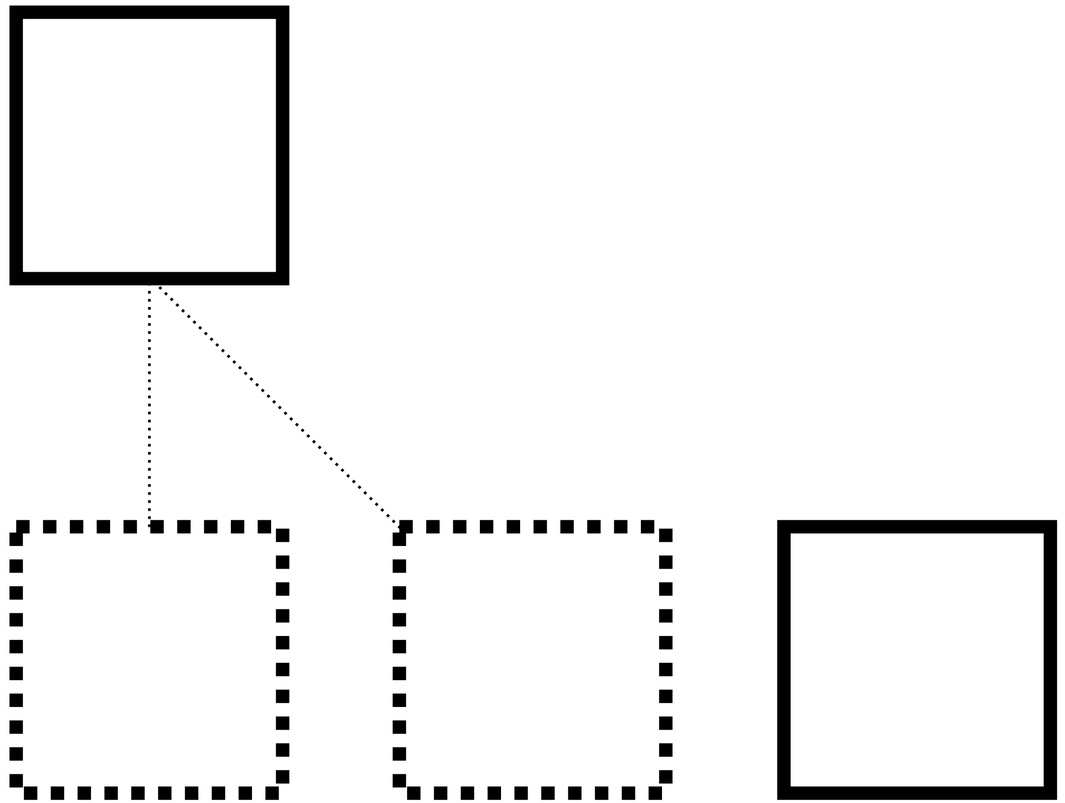
Add 1 more



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

3 UTXOs

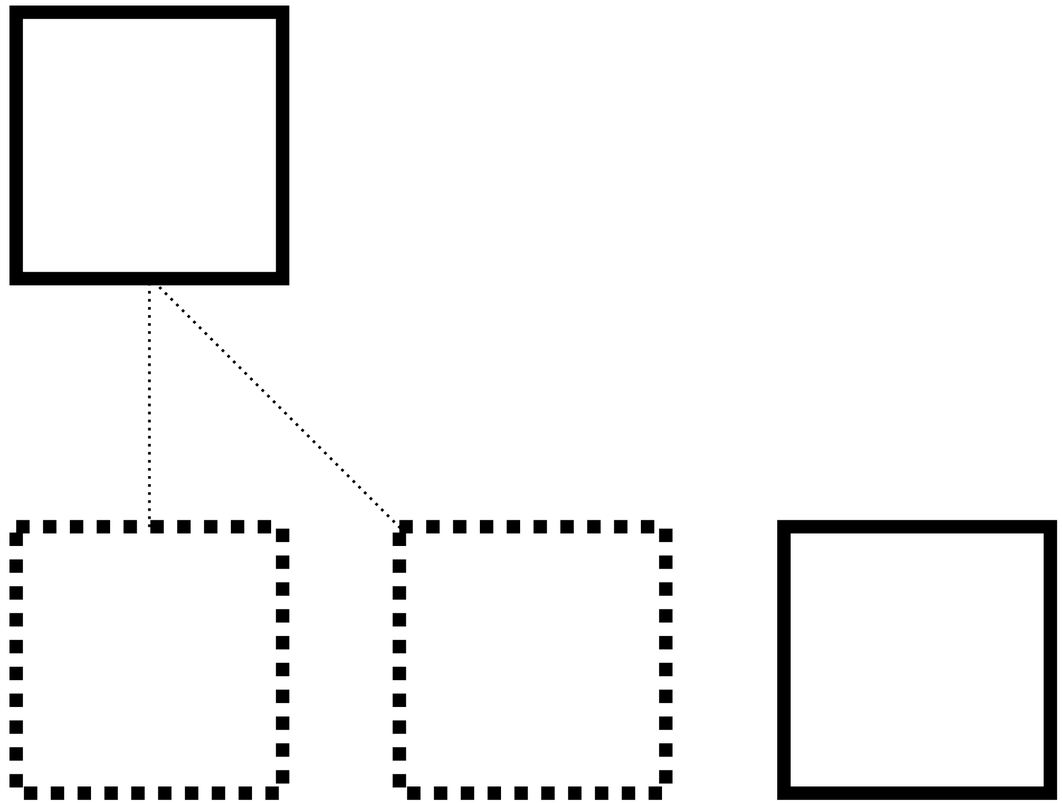
Add 1 more



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

3 UTXOs

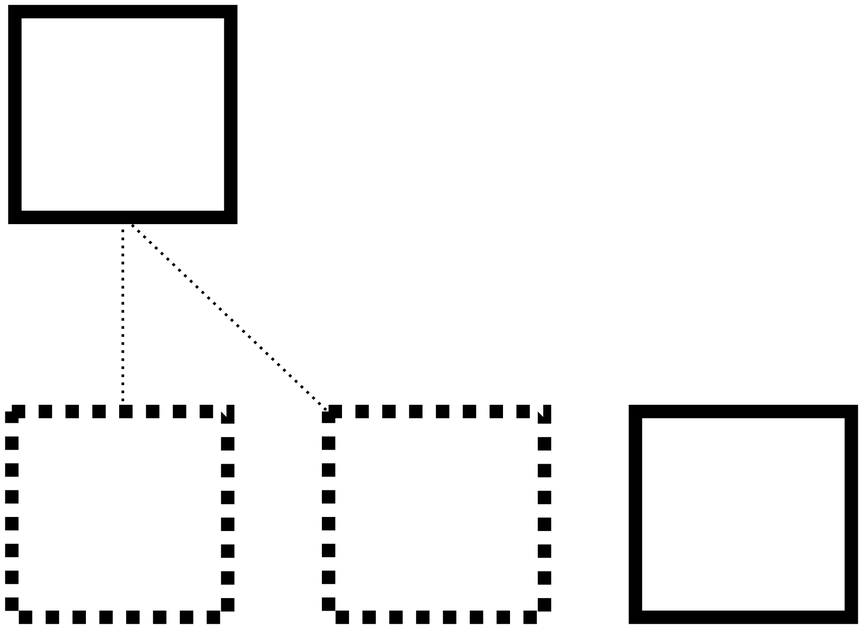
Nothing to hash with. Becomes a root



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

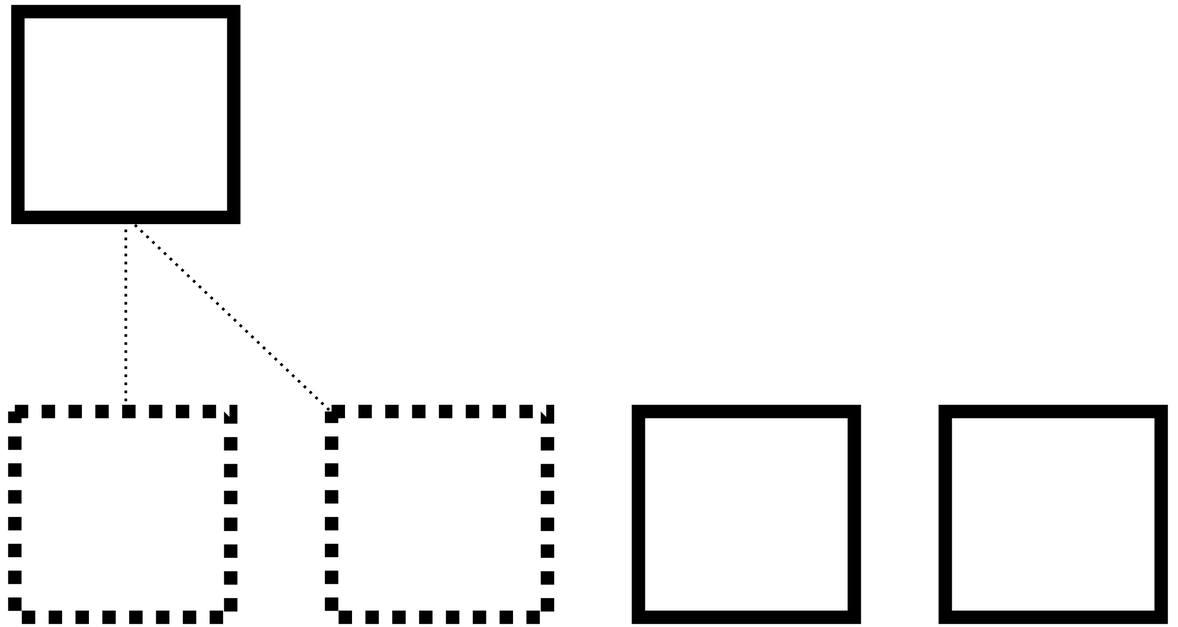
Another one



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

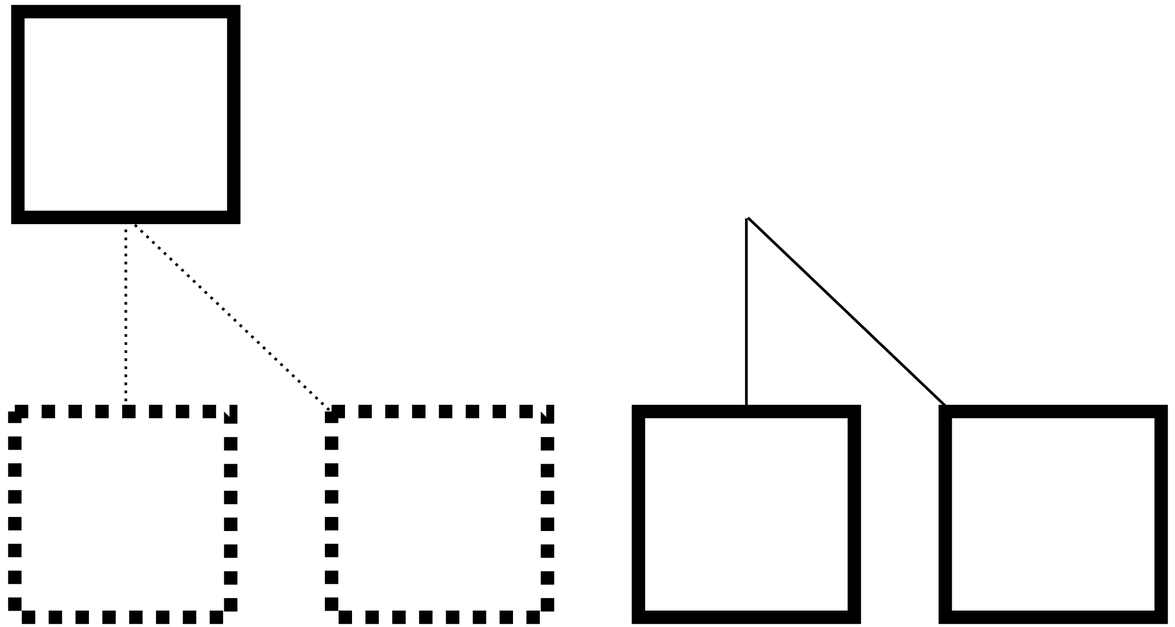
Another one



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

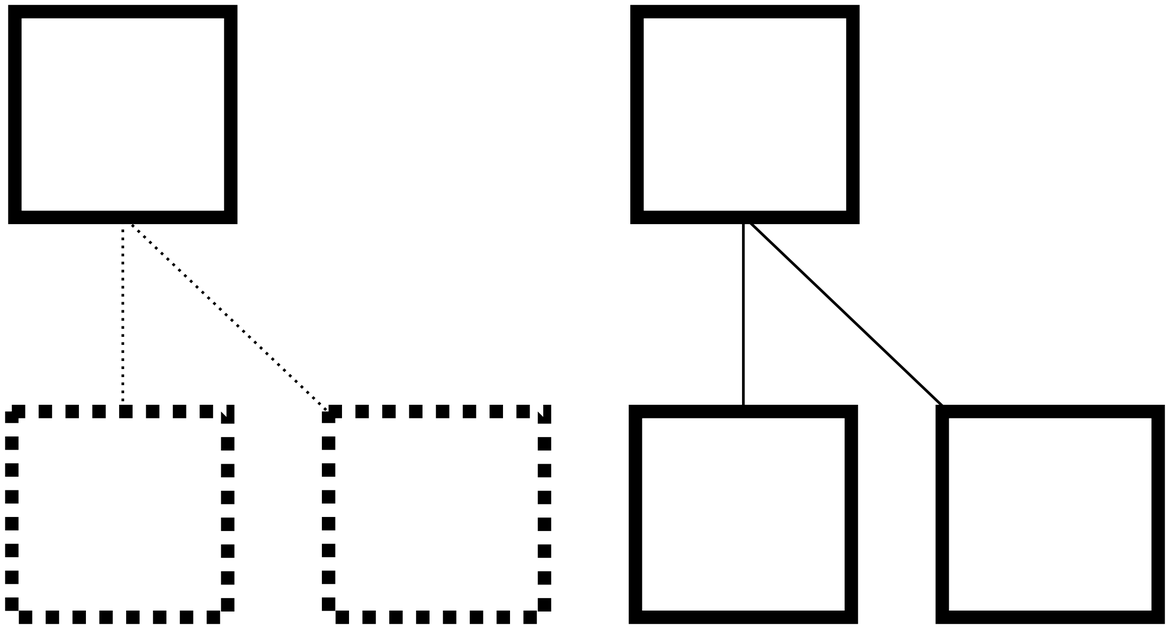
Concatenate



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

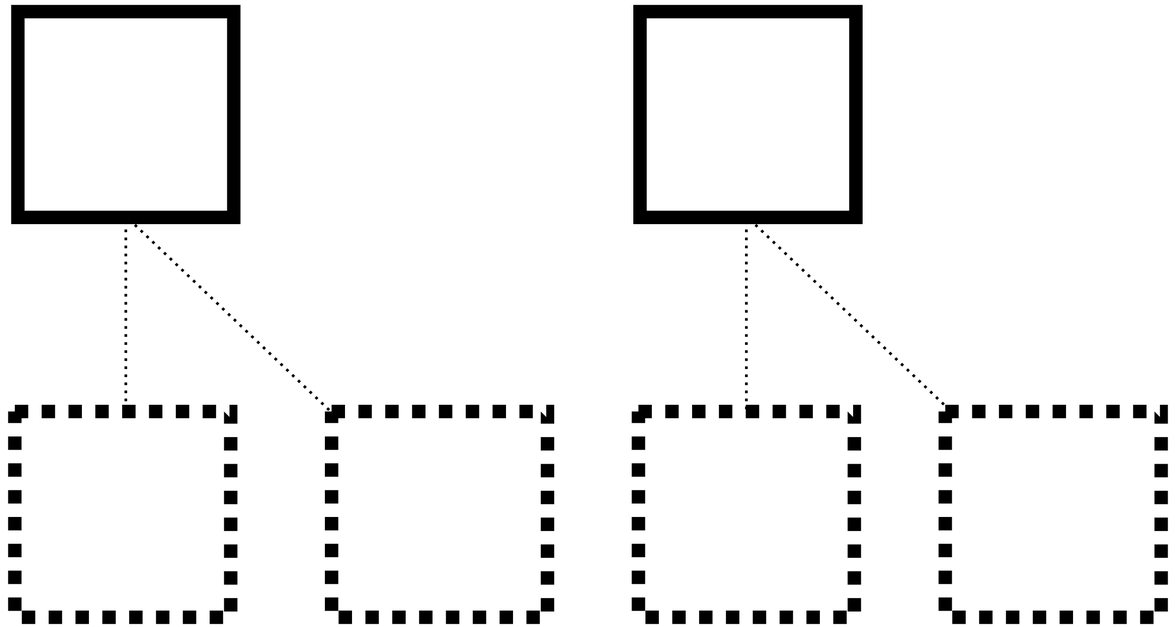
Hash



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

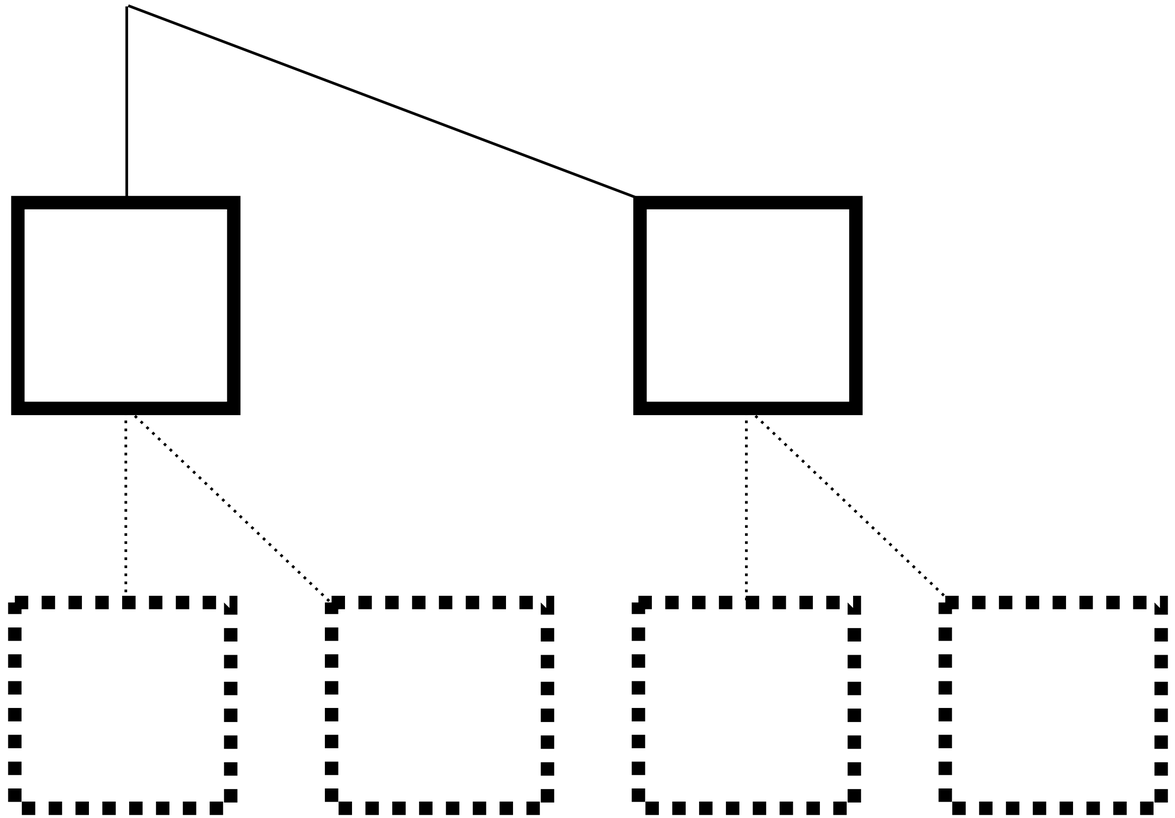
Can now throw away the leaves



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

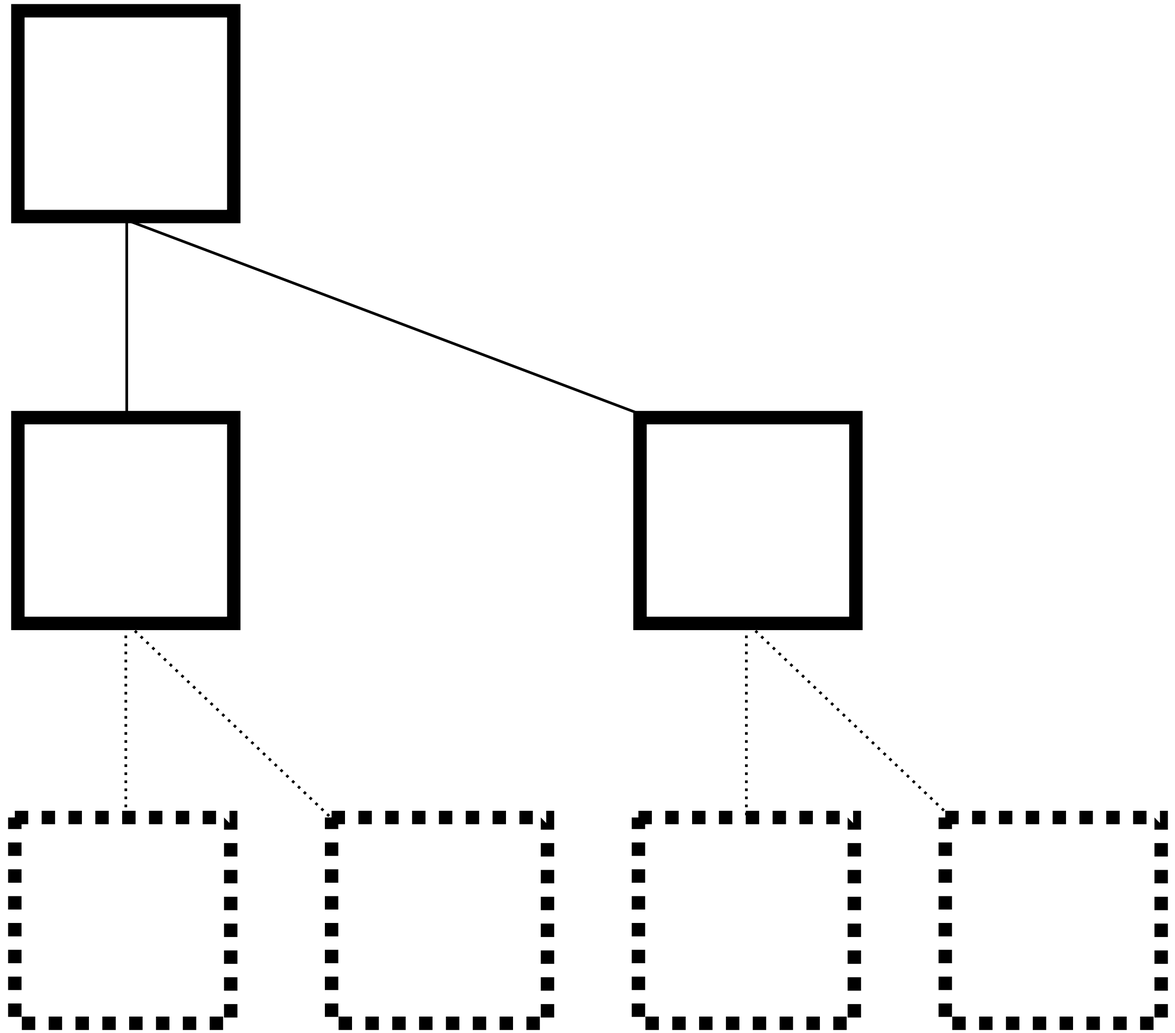
Concatenate



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

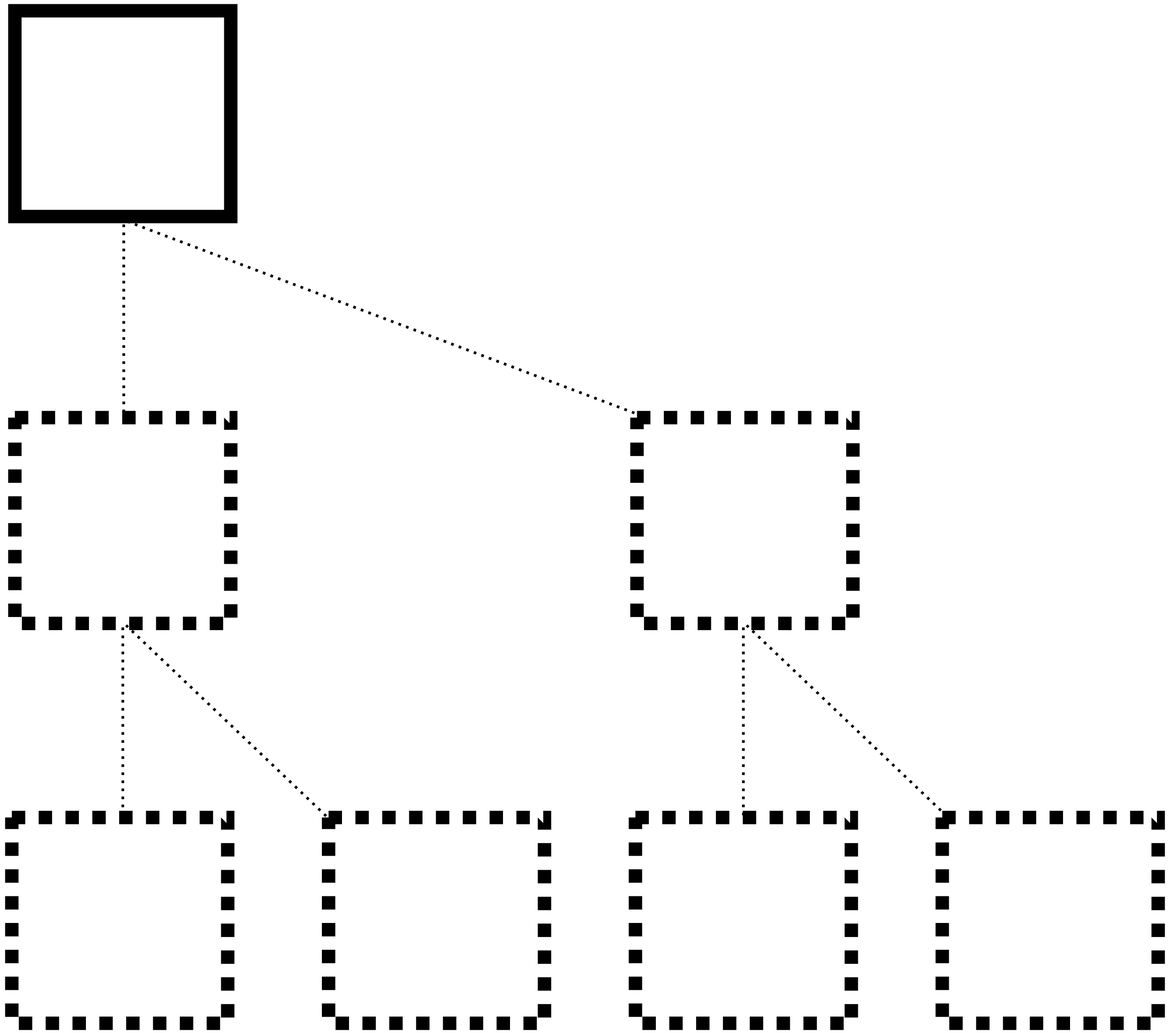
Hash



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

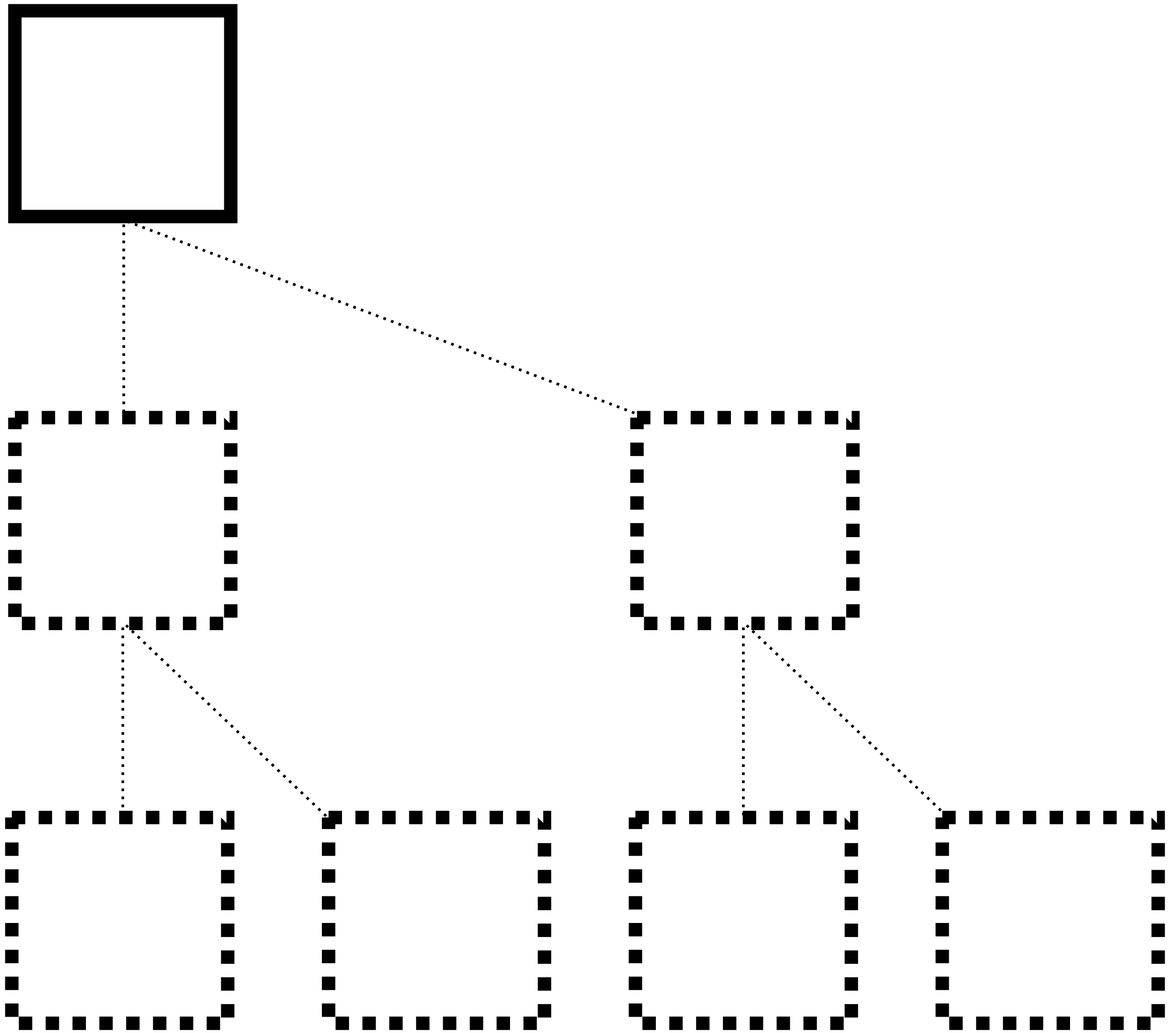
Throw away



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

4 UTXOs

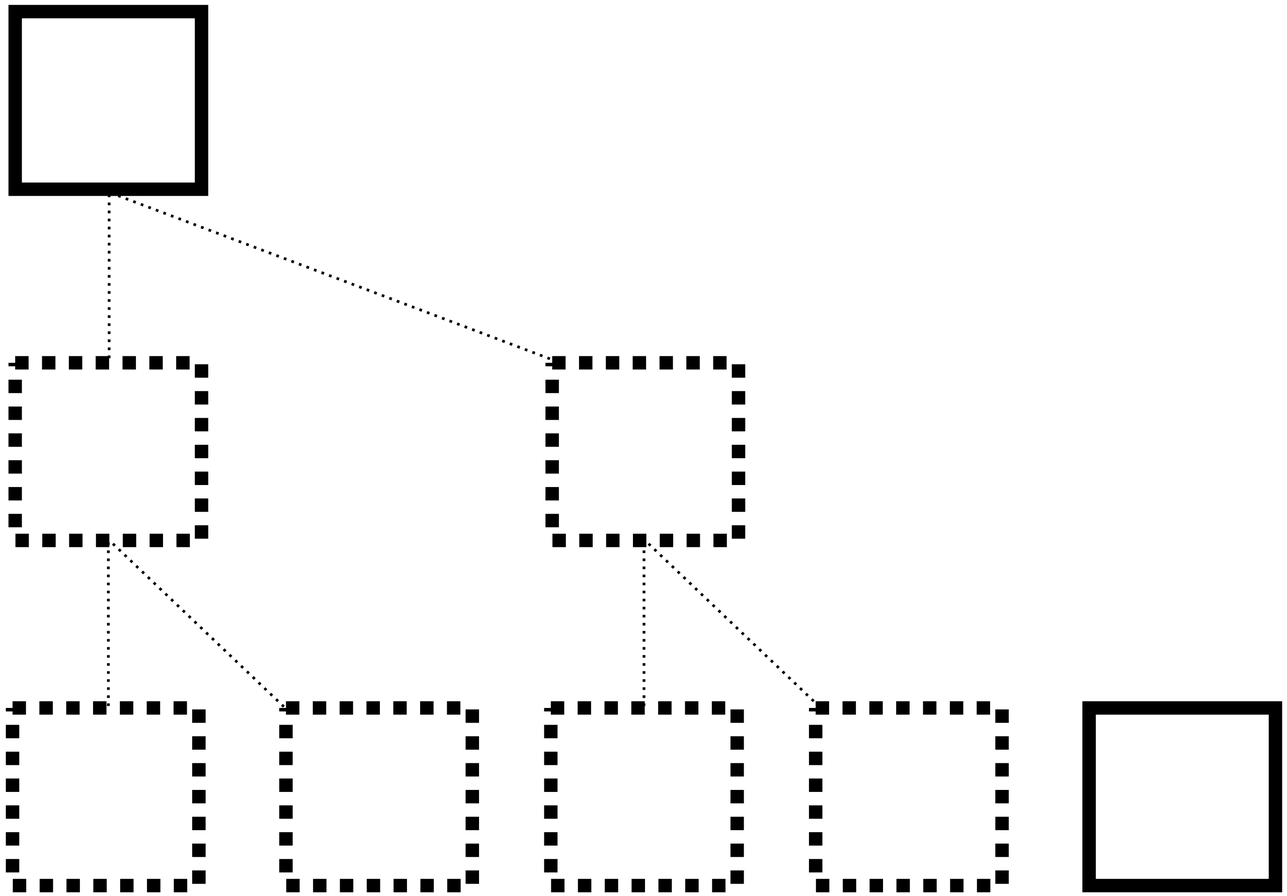
Finished tree



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

5 UTXOs

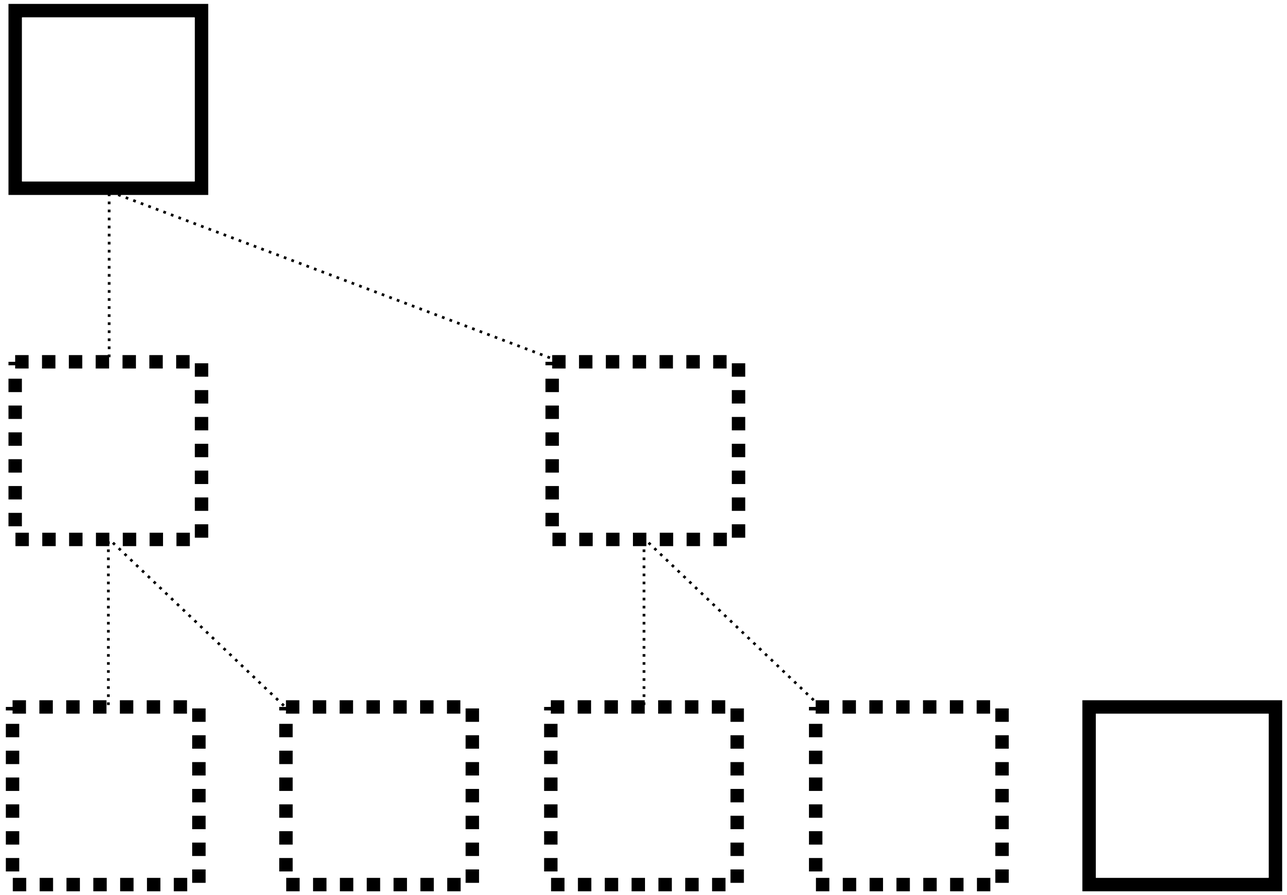
Add one more



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

5 UTXOs

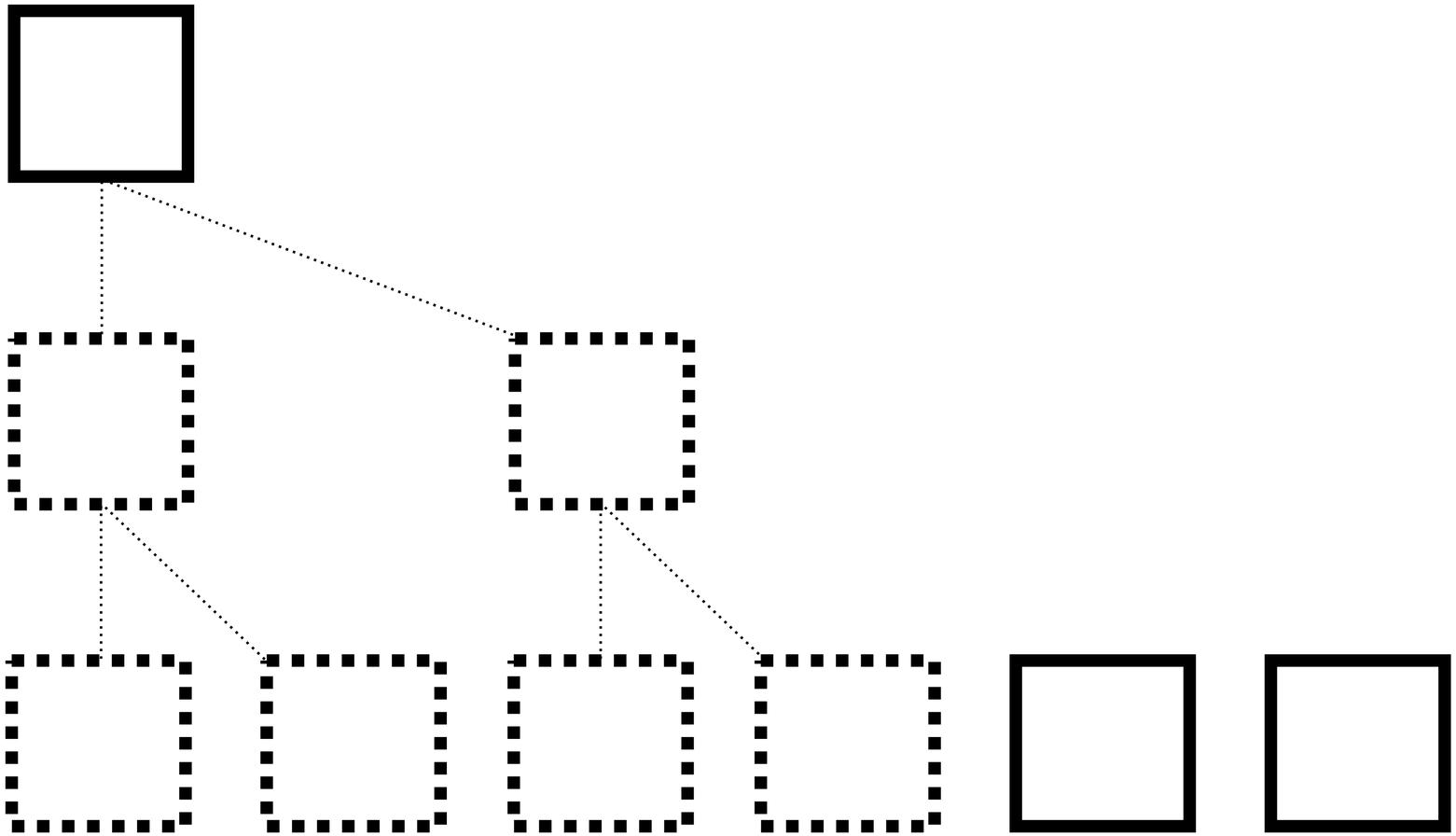
Nothing to hash with so it becomes a root



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

6 UTXOs

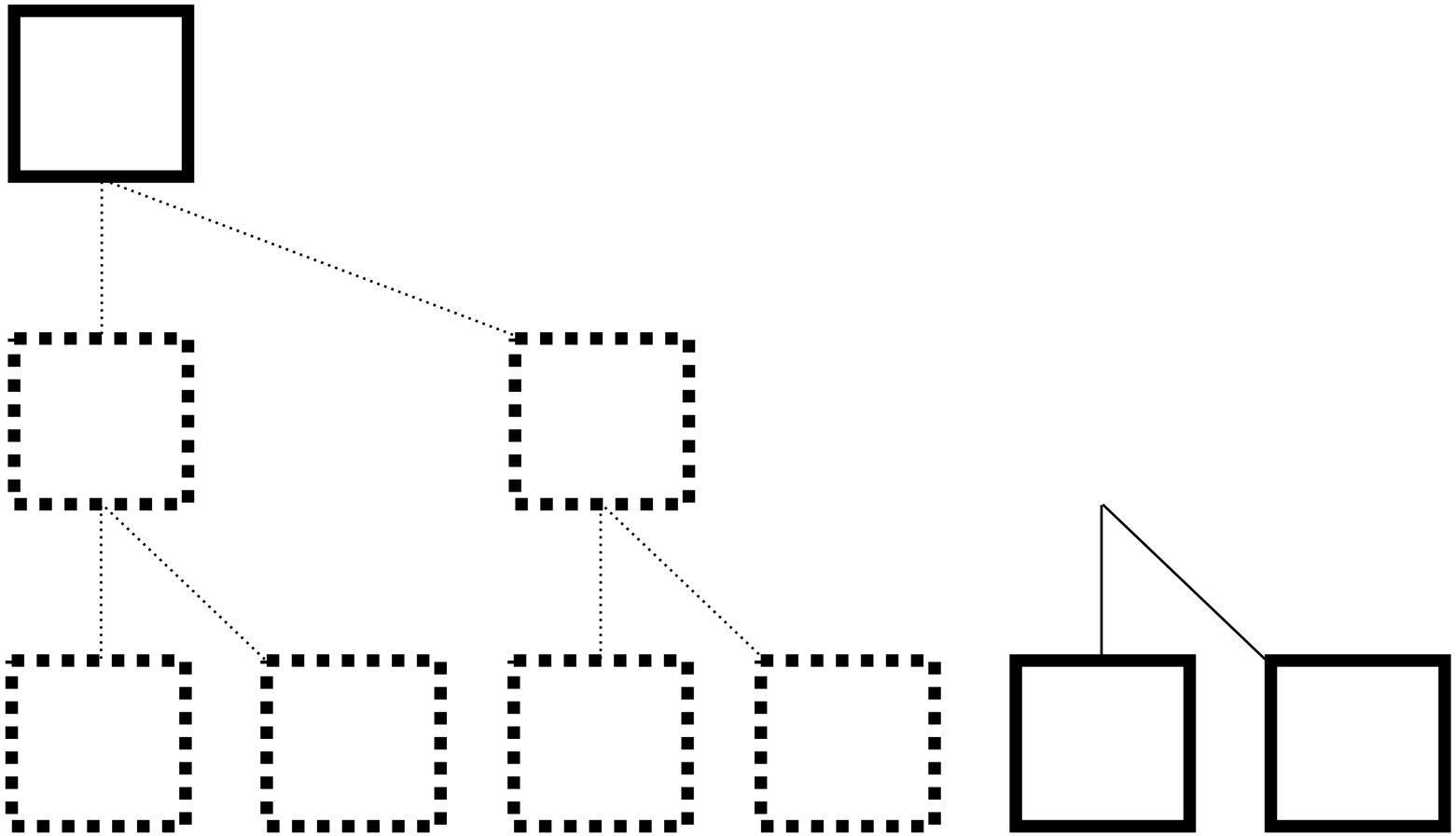
1 more



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

6 UTXOs

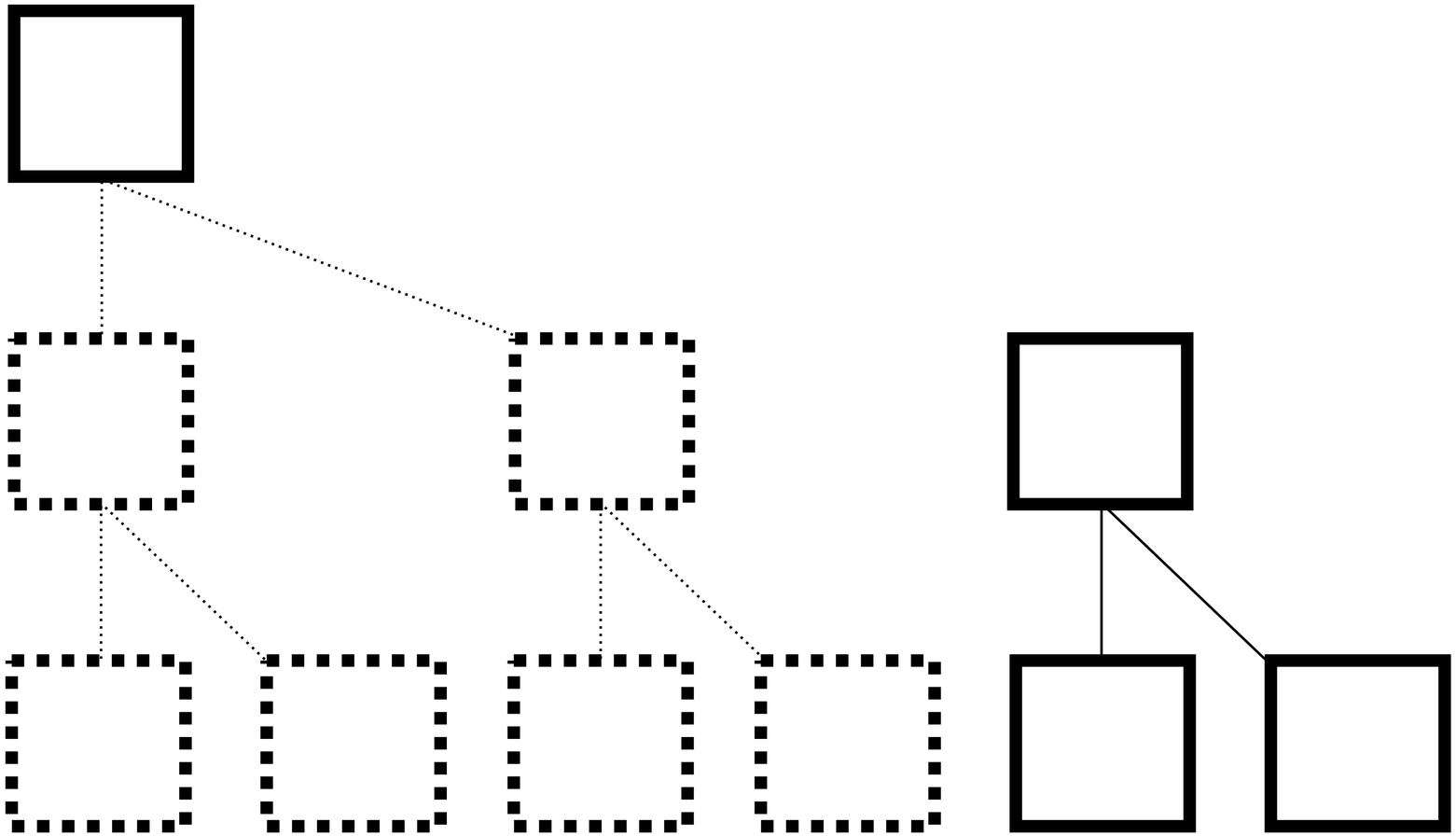
Concatenate



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

6 UTXOs

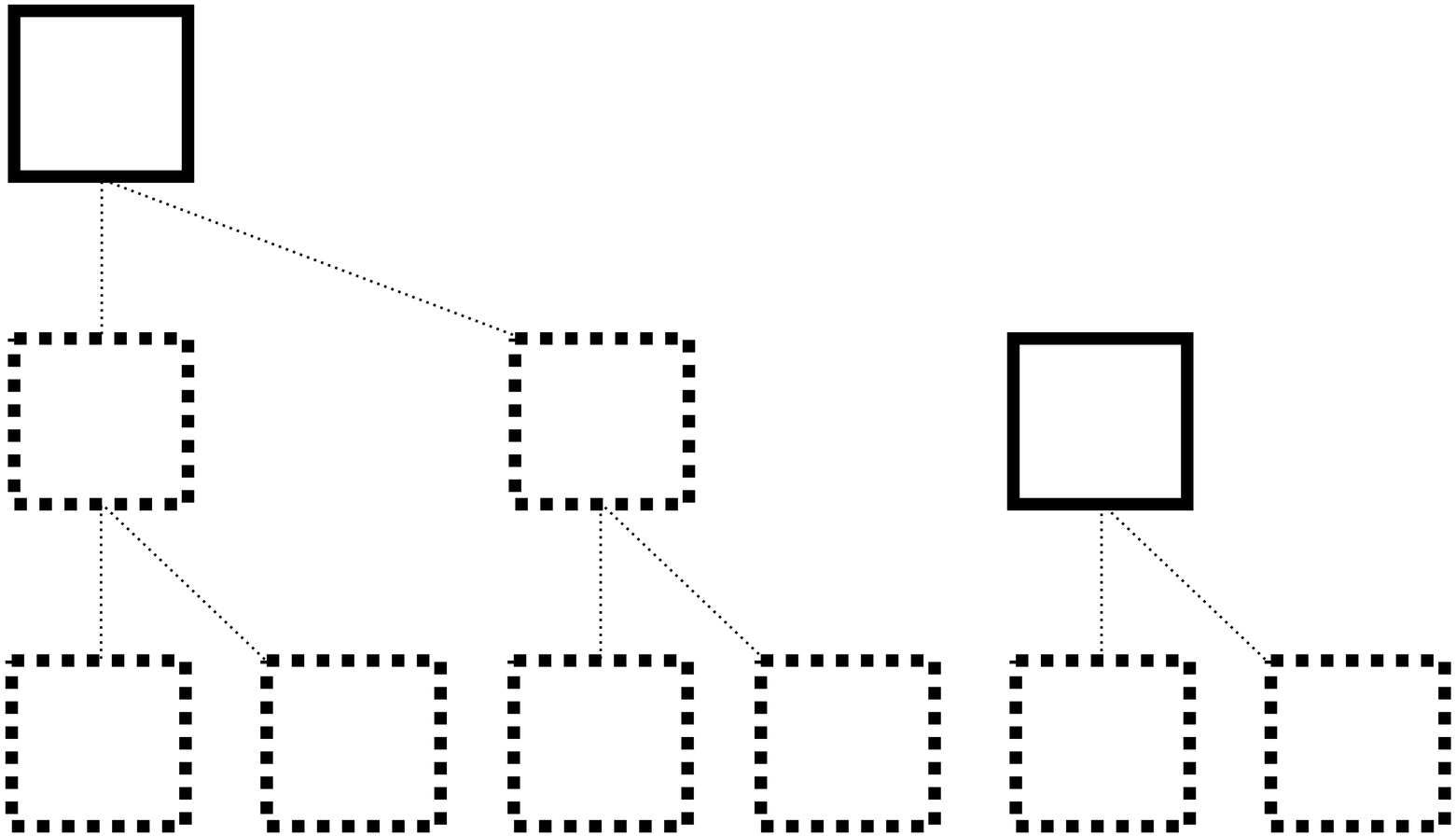
Hash



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

6 UTXOs

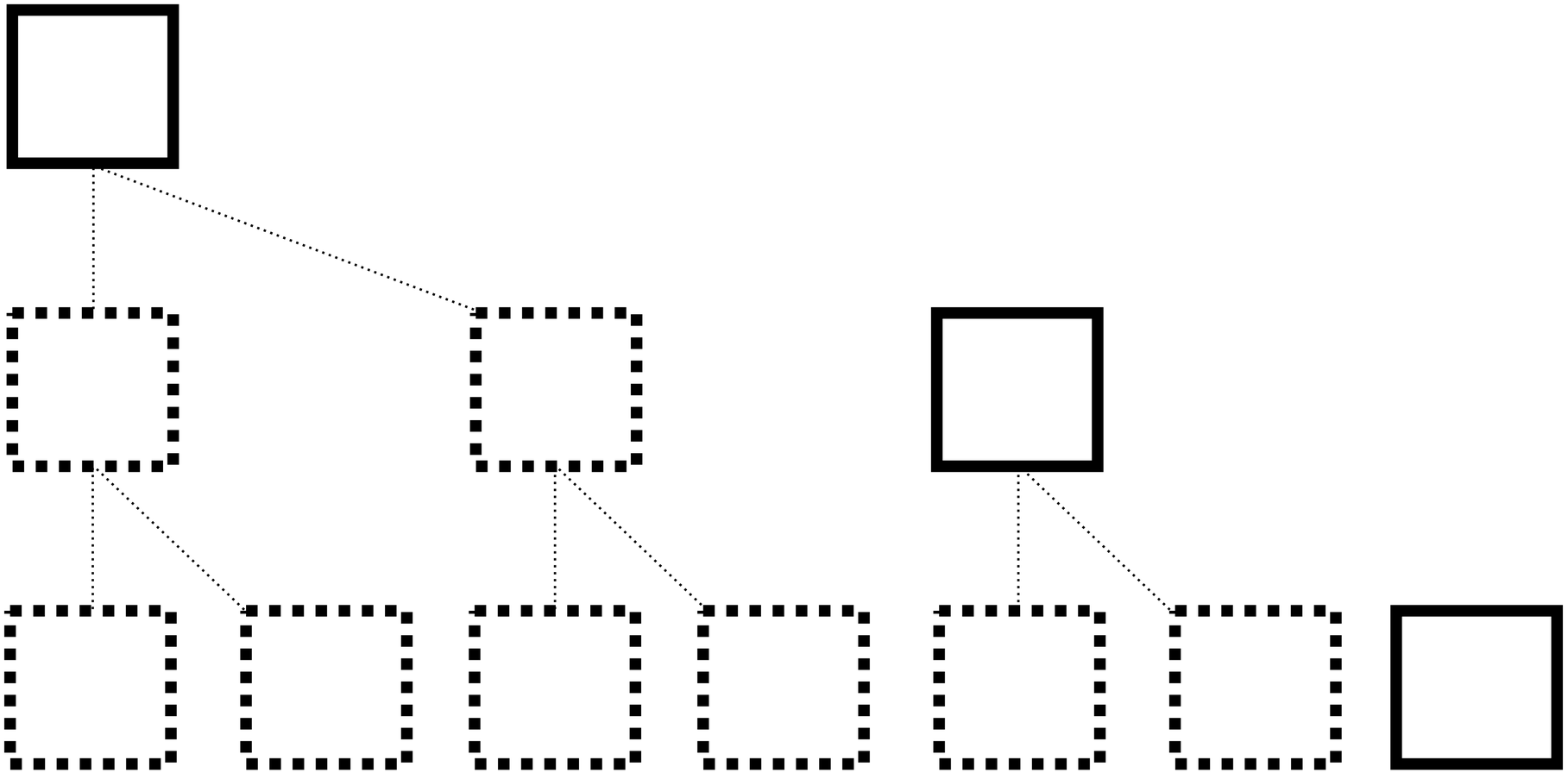
Throw away



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

7 UTXOs

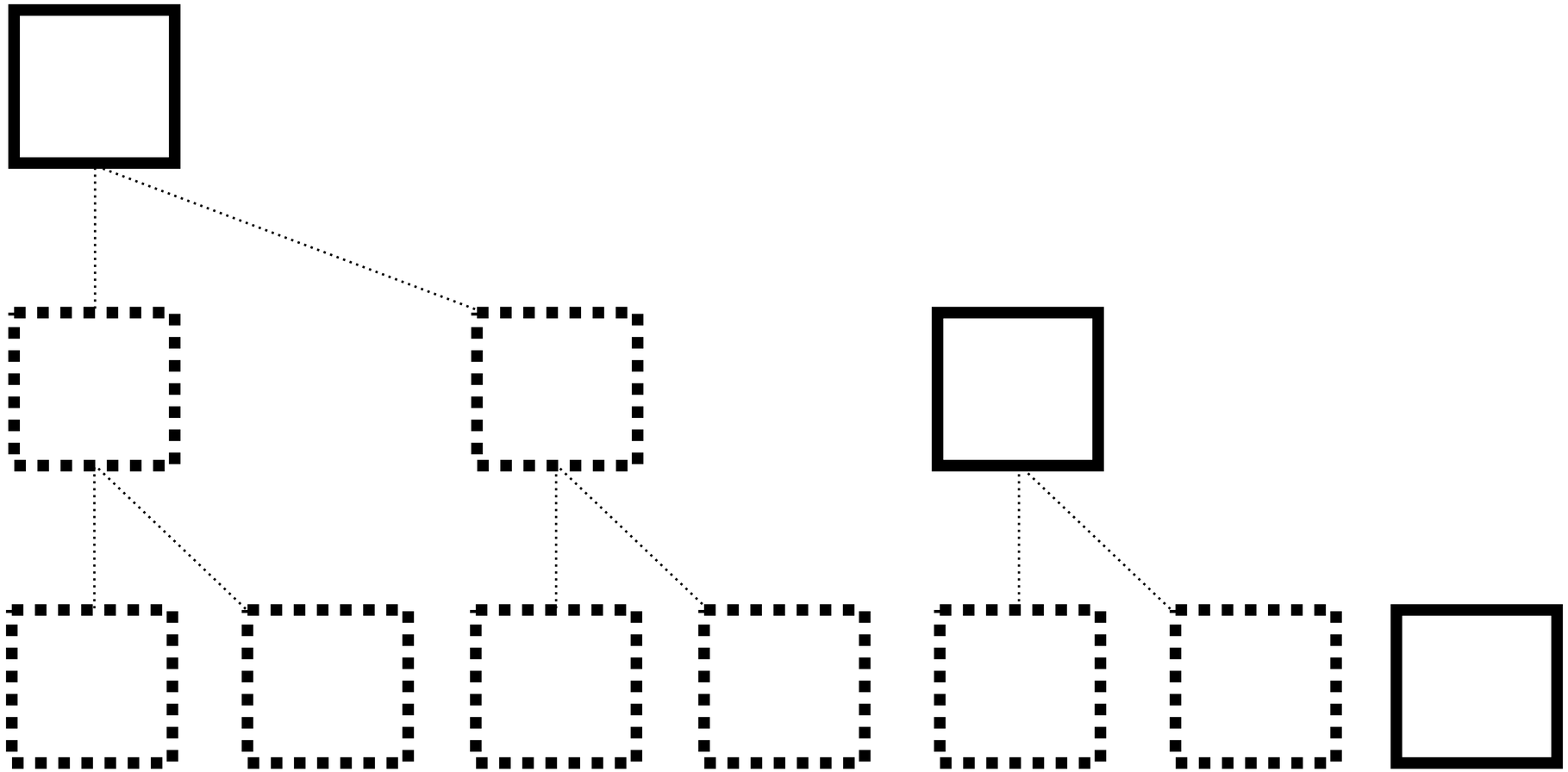
Another one



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

7 UTXOs

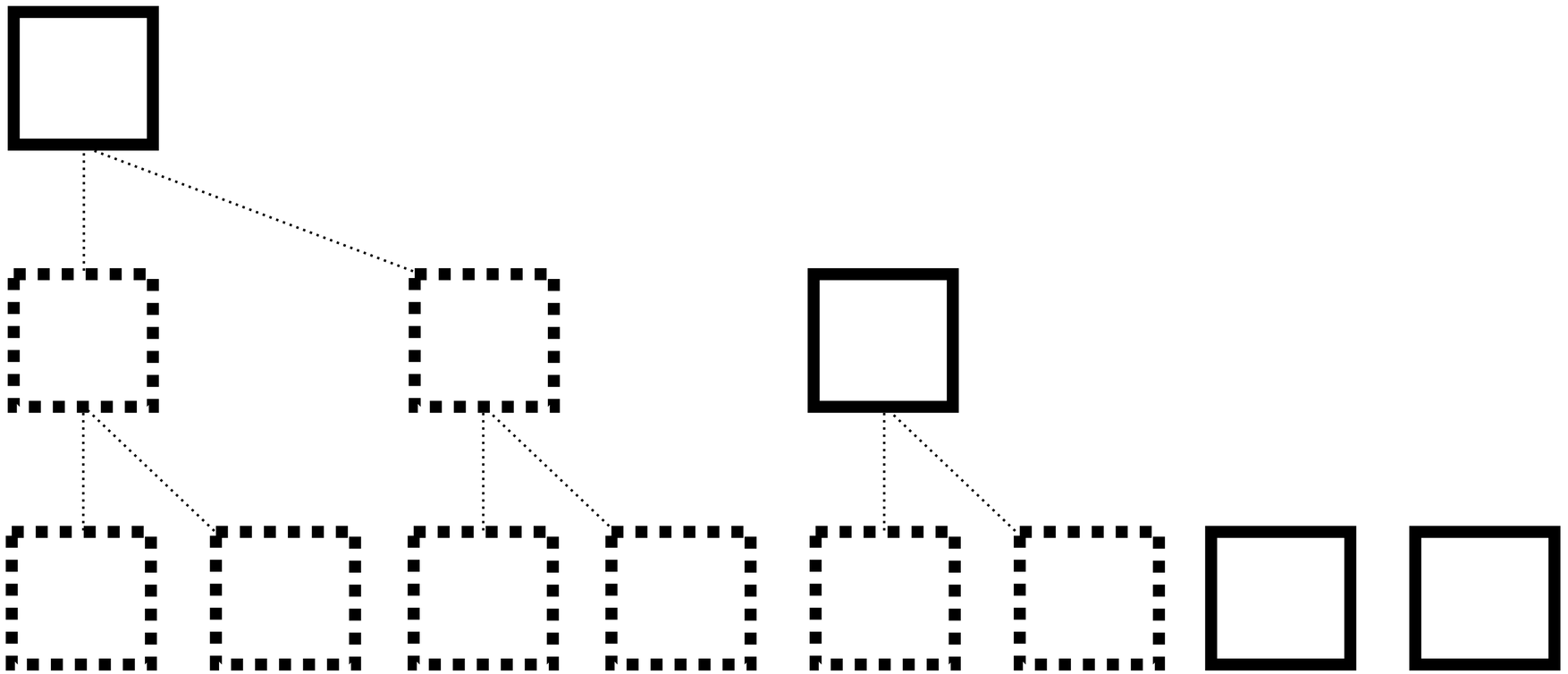
Nothing to hash with so it becomes a root



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs

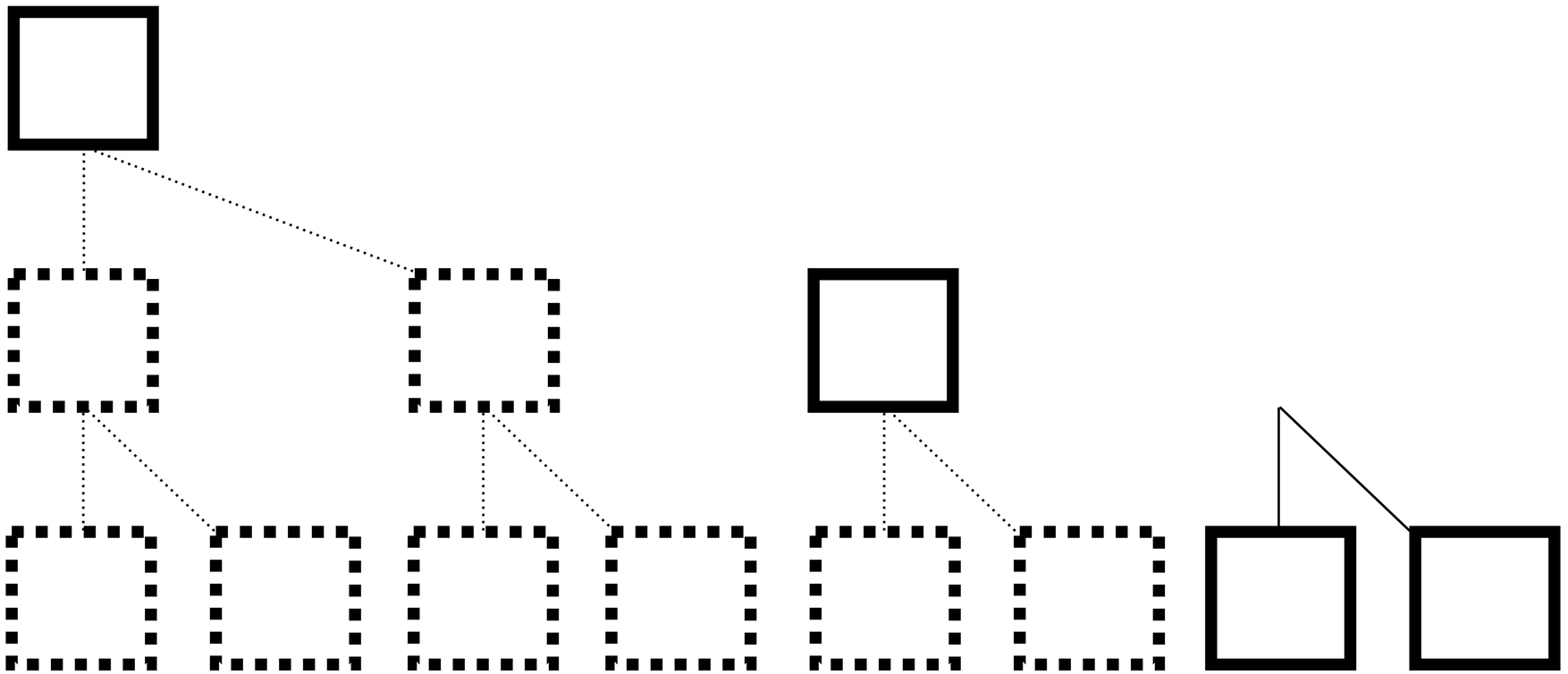
Another one



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs

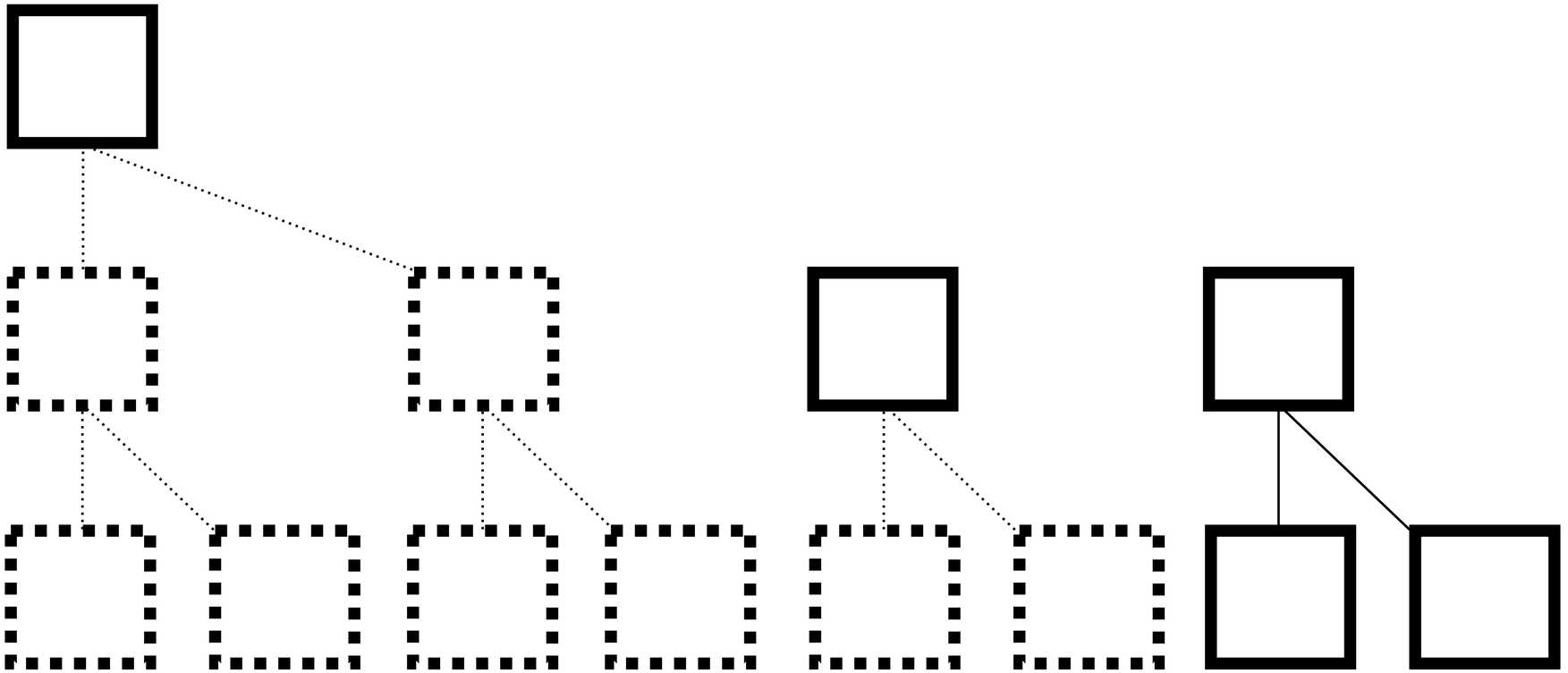
Concatenate



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs

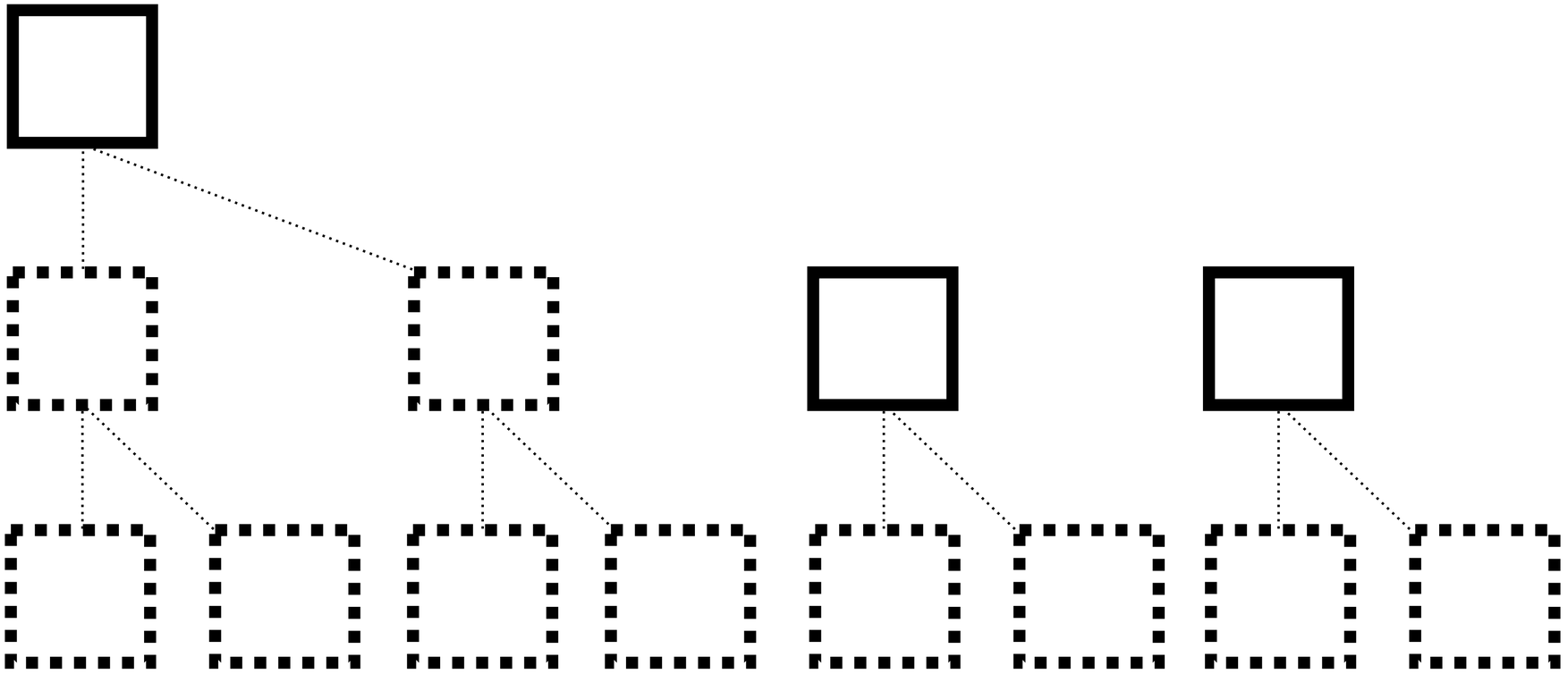
Hash



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs

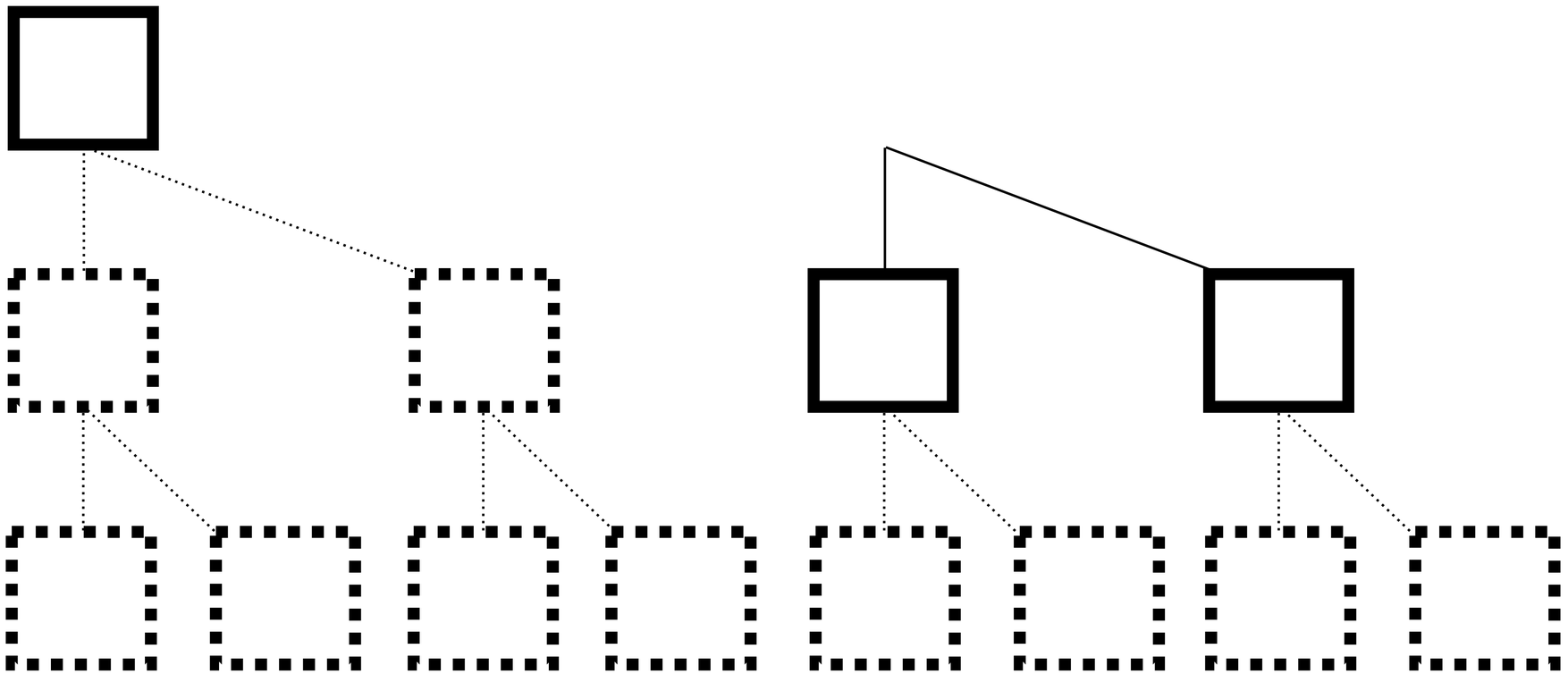
Throw away



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs

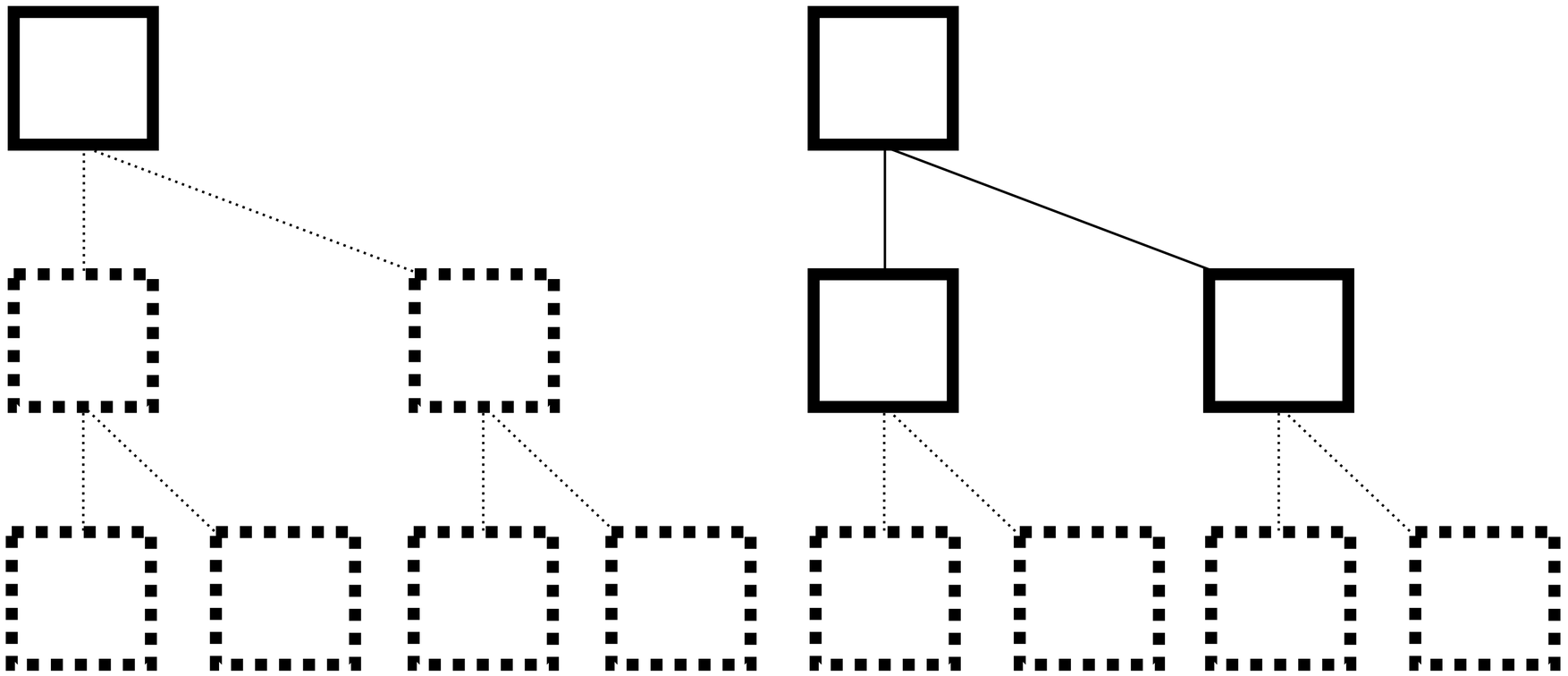
Concatenate



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs

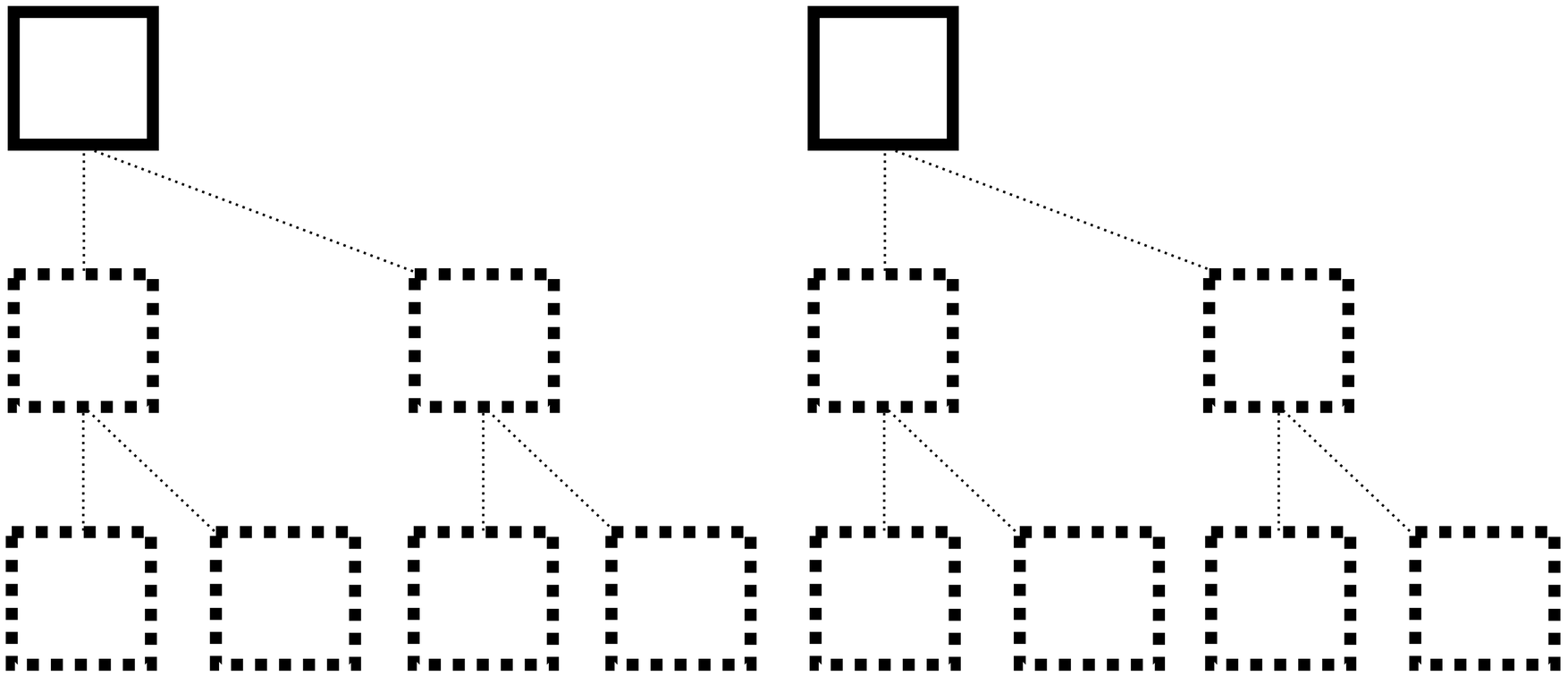
Hash



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs

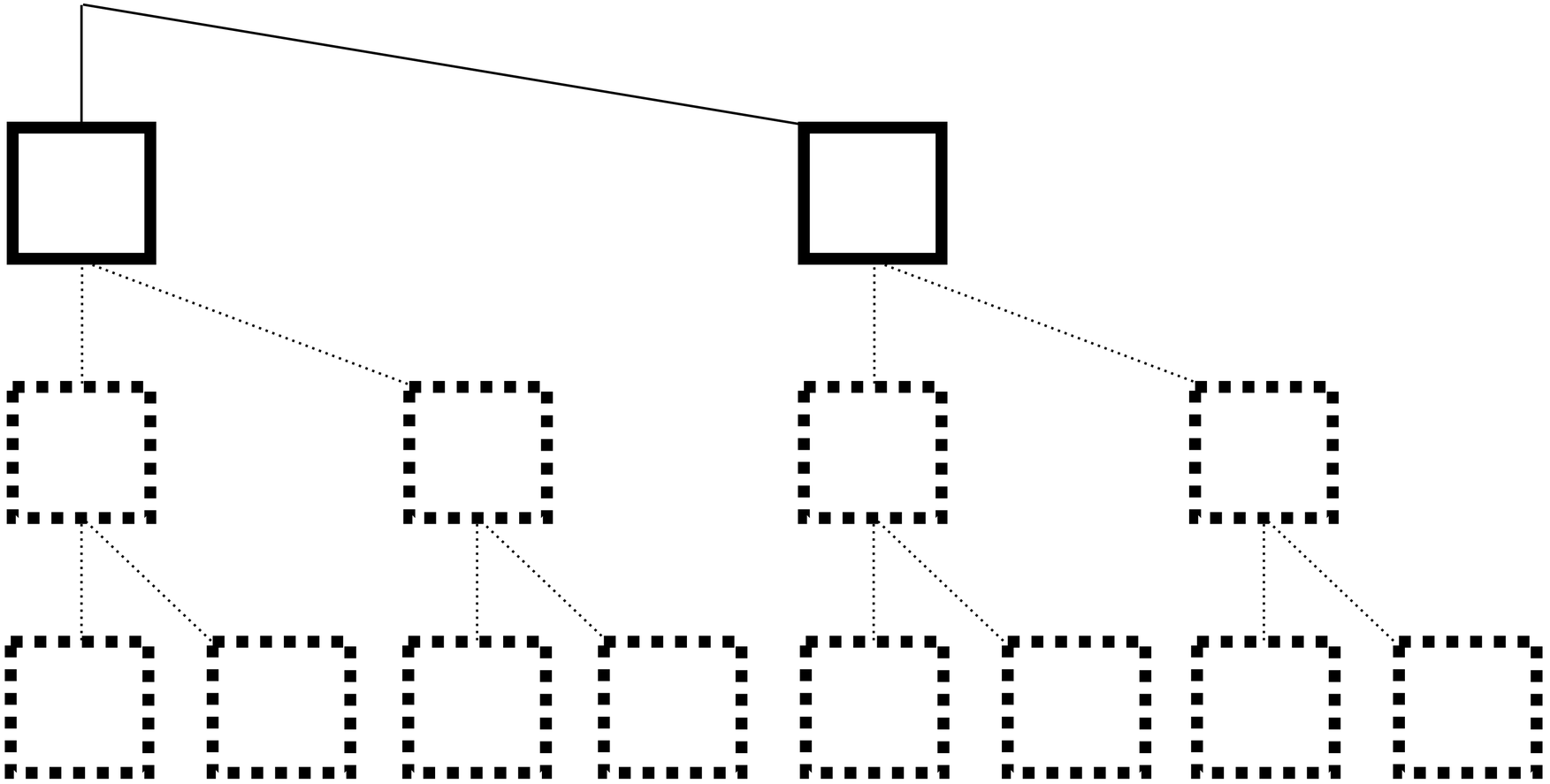
Throw away



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

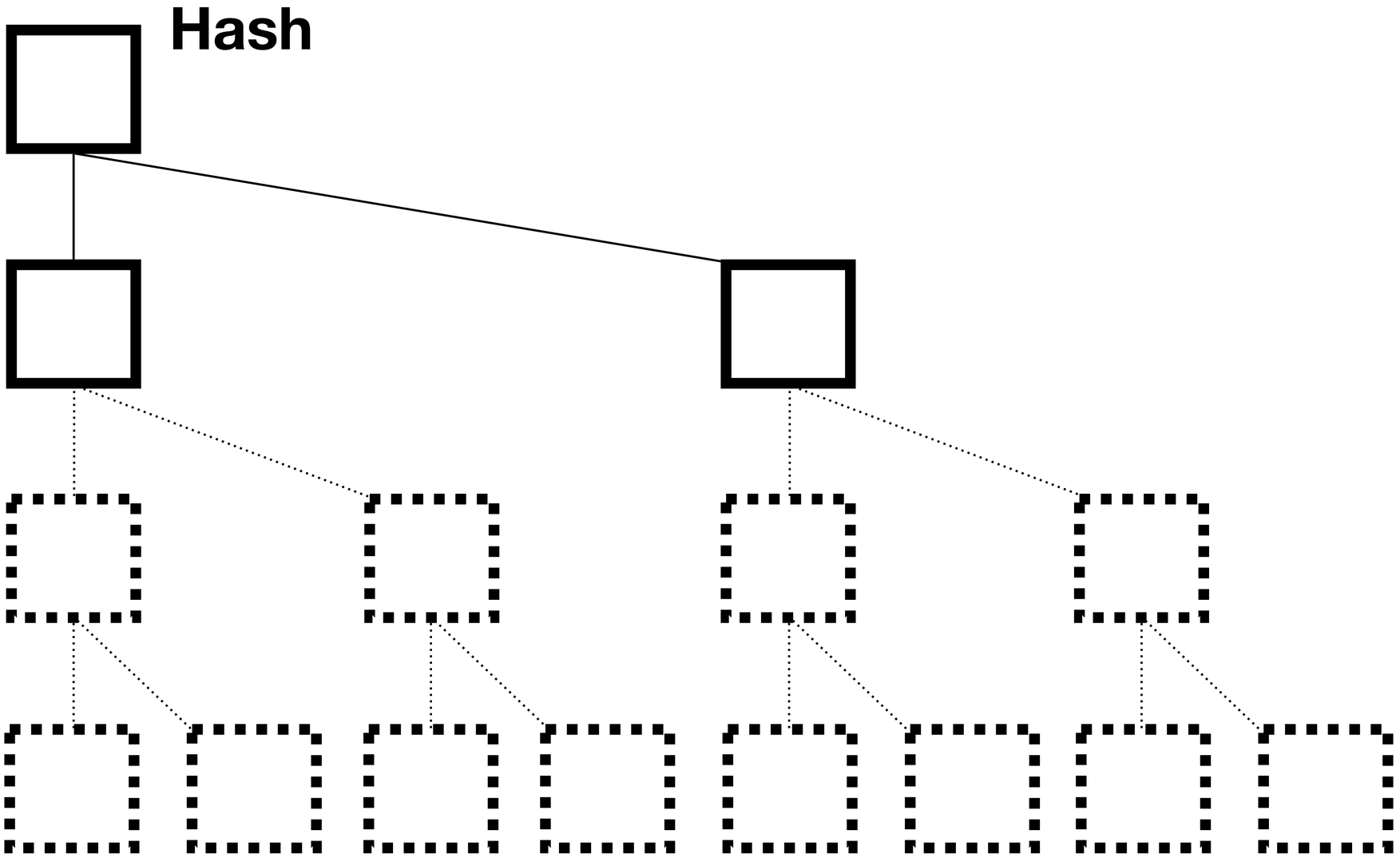
8 UTXOs

Concatenate



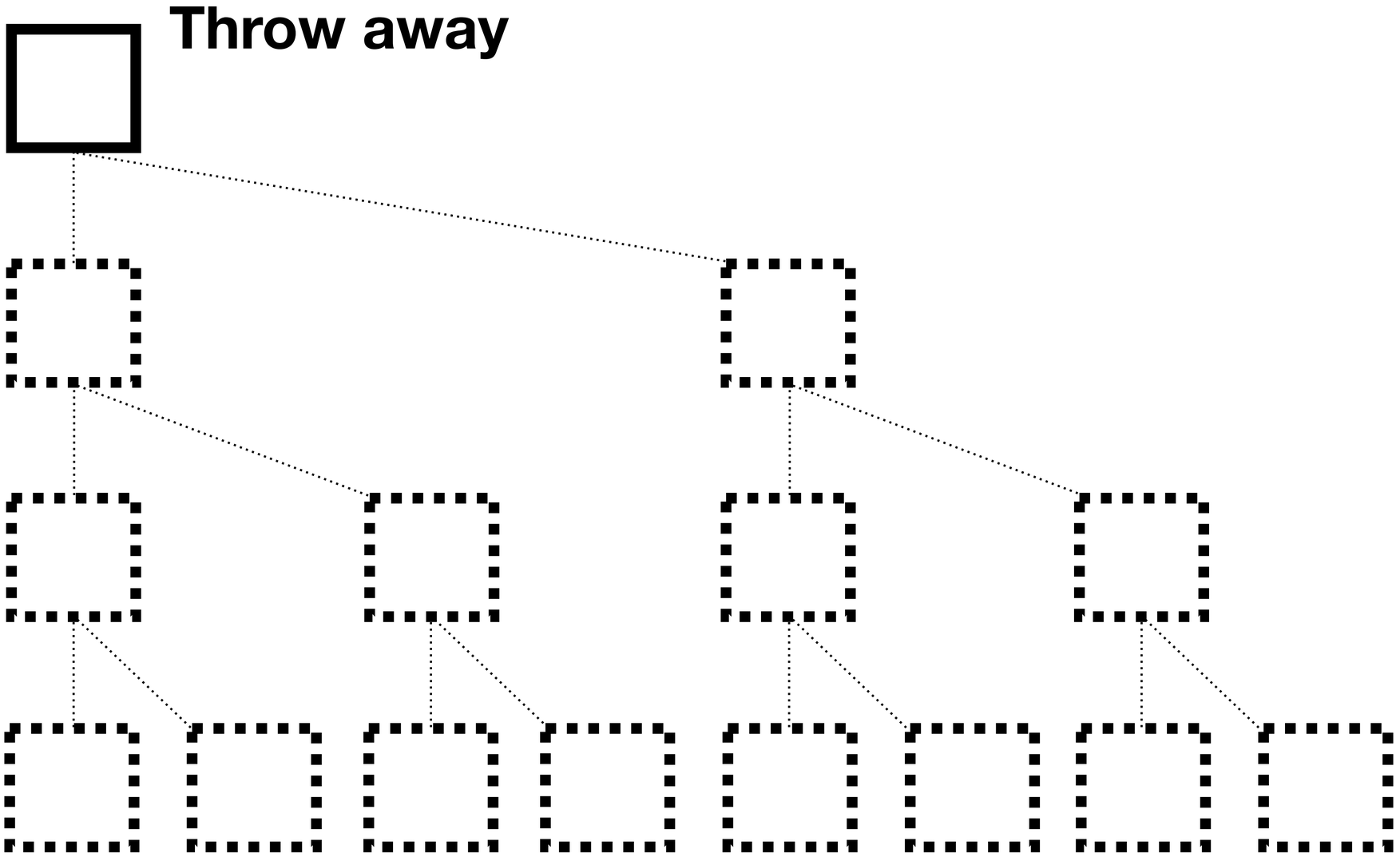
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

5.1.2 Chaining Value Stack

To help picture the role of the CV stack, Figure 6 shows a growing tree as chunk CVs are added incrementally. As just discussed above, chunk CVs are added to this tree only after the caller has supplied at least 1 byte for the following chunk, so we know that none of these chunks or parent nodes is the root of the tree, and we do not need to worry about the `ROOT` flag yet.

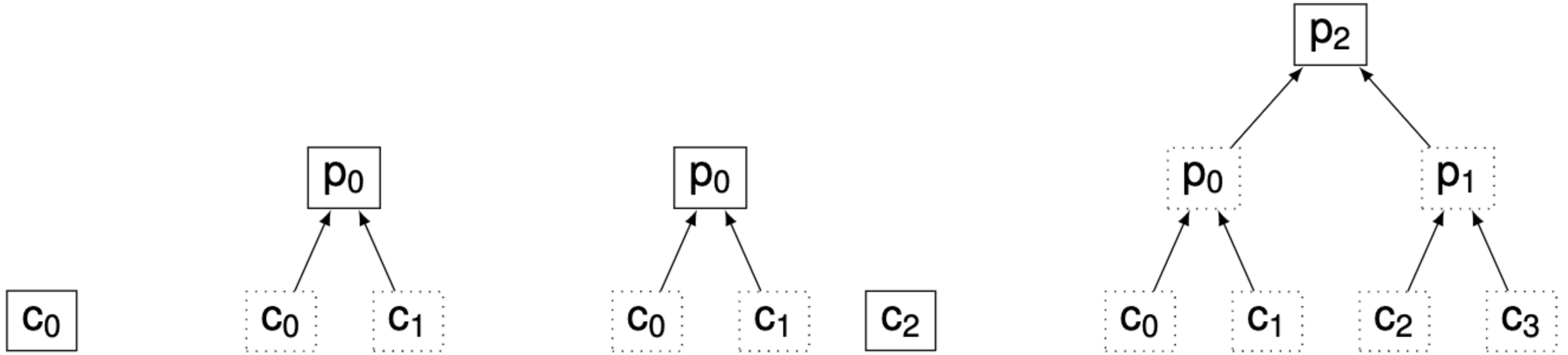


Figure 6: An incomplete tree growing incrementally from 1 to 4 chunks. Dotted boxes represent CVs that no longer need to be stored.

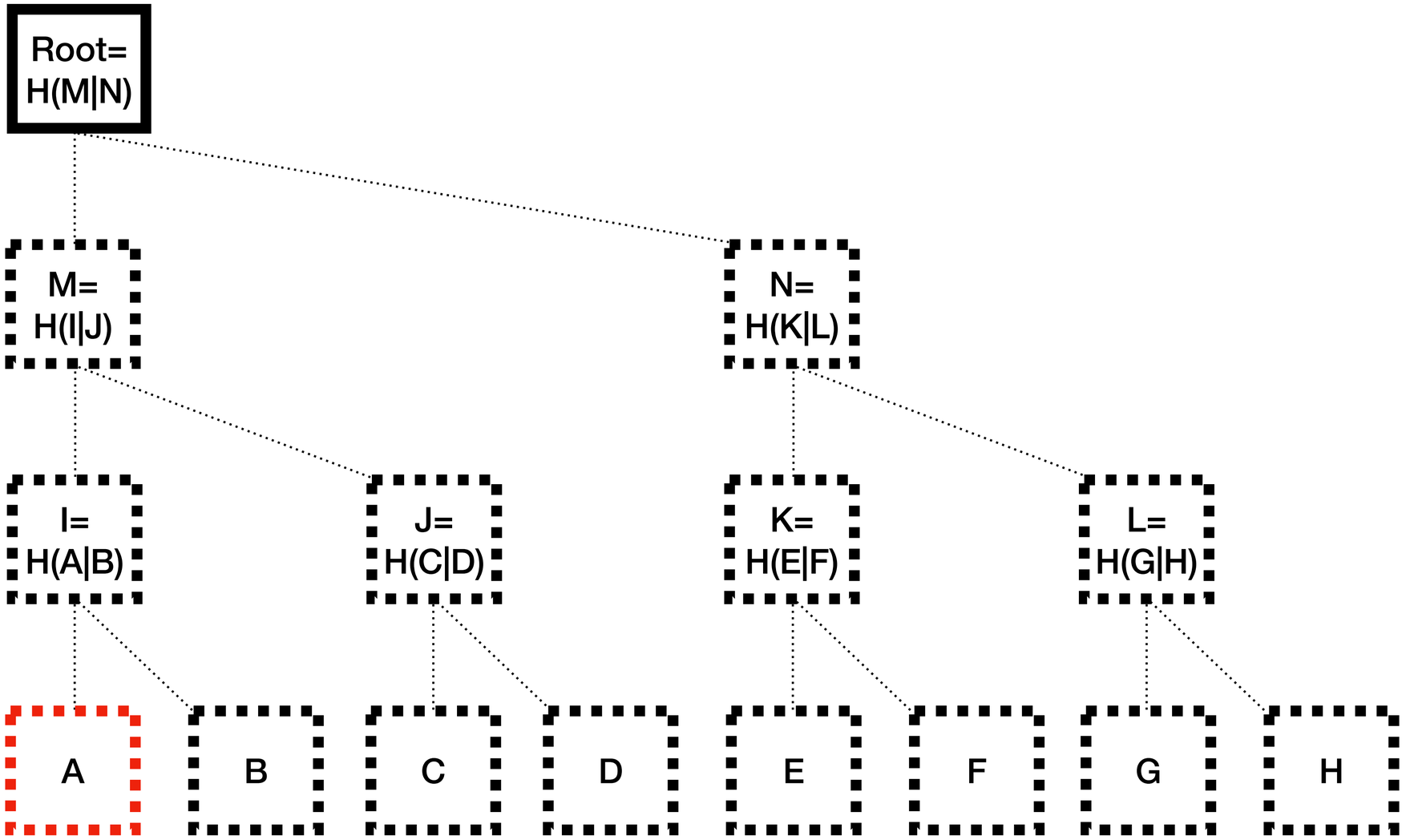
Role of levelDB

It let’s you

1. Add a UTXO
2. Delete a UTXO
3. Tell you the existence of a UTXO
4. Provide the data for verification

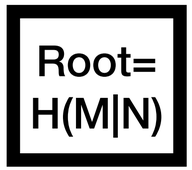
Deletion happens with proving existence

Deleting A



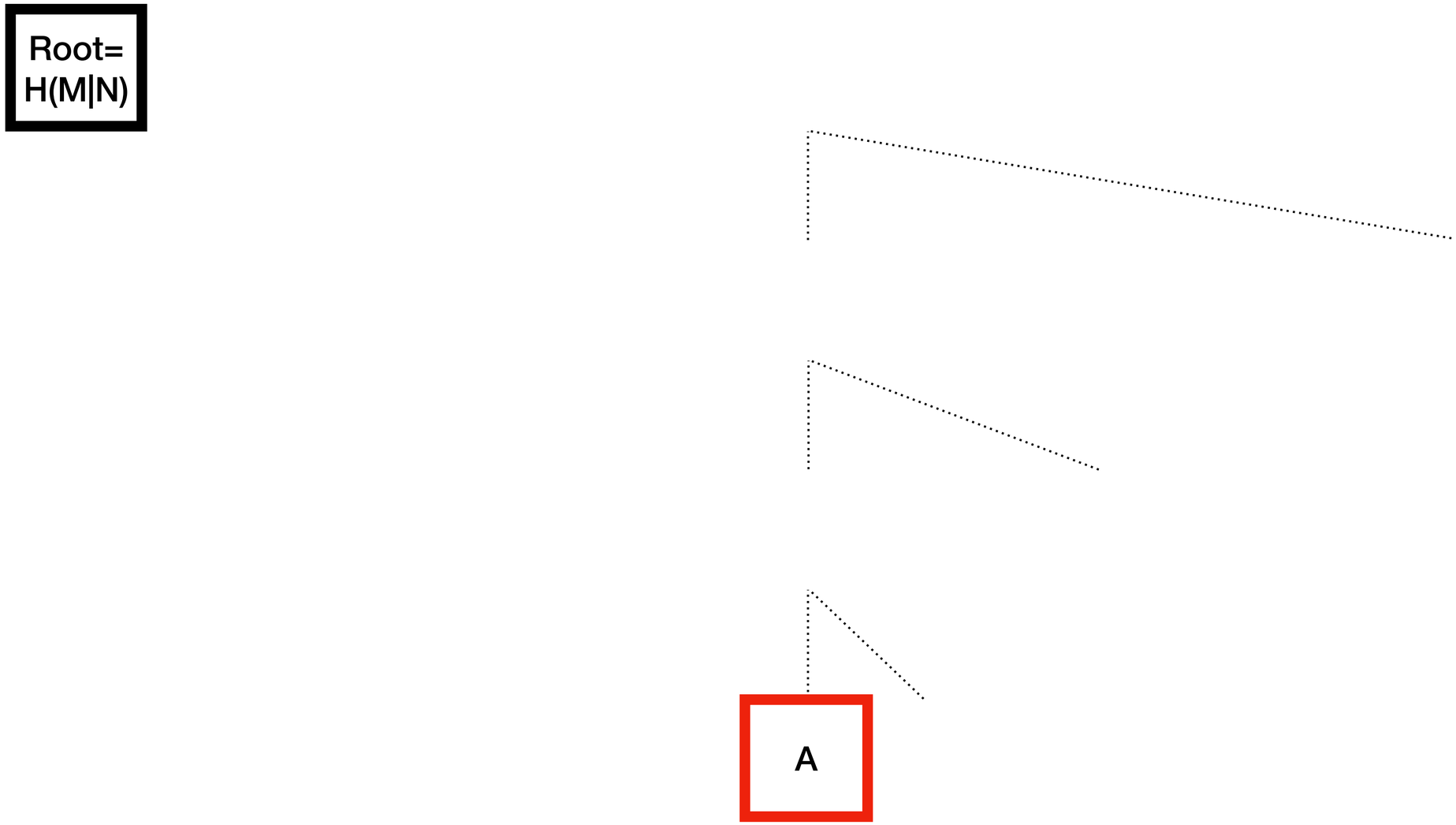
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

The only requirement is the roots



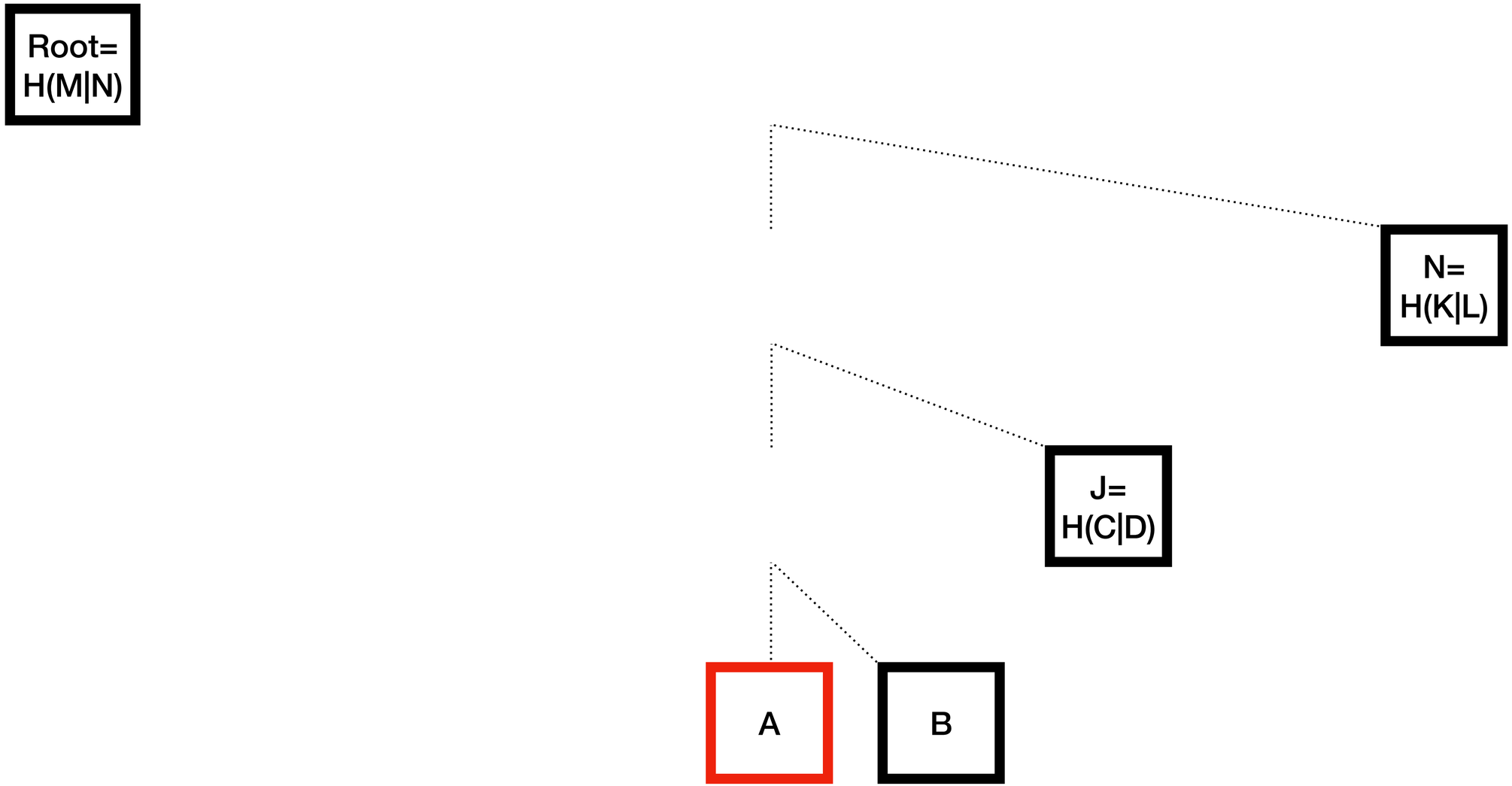
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate A from the Bitcoin Block



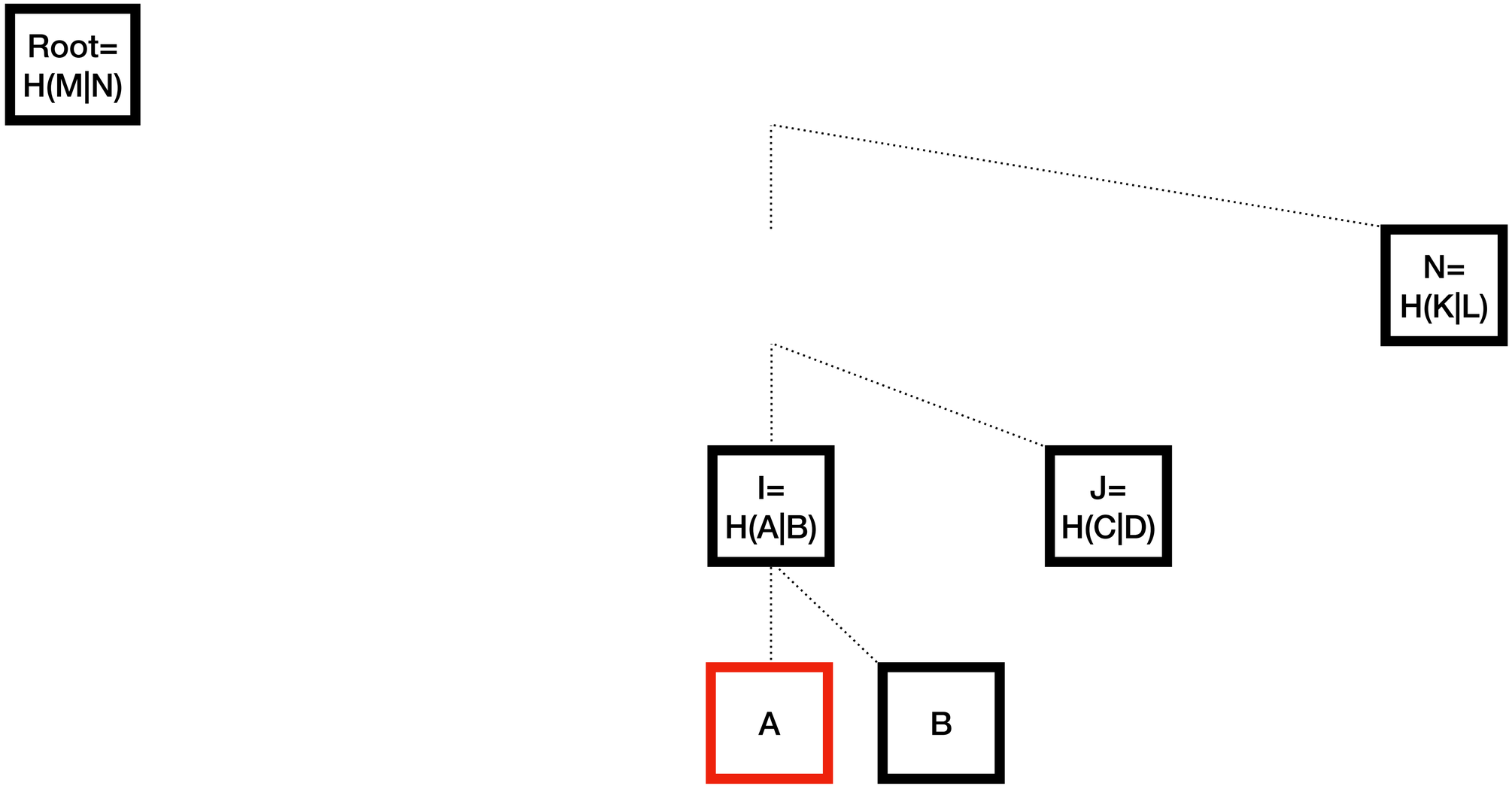
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Receive the proof: B, J, N



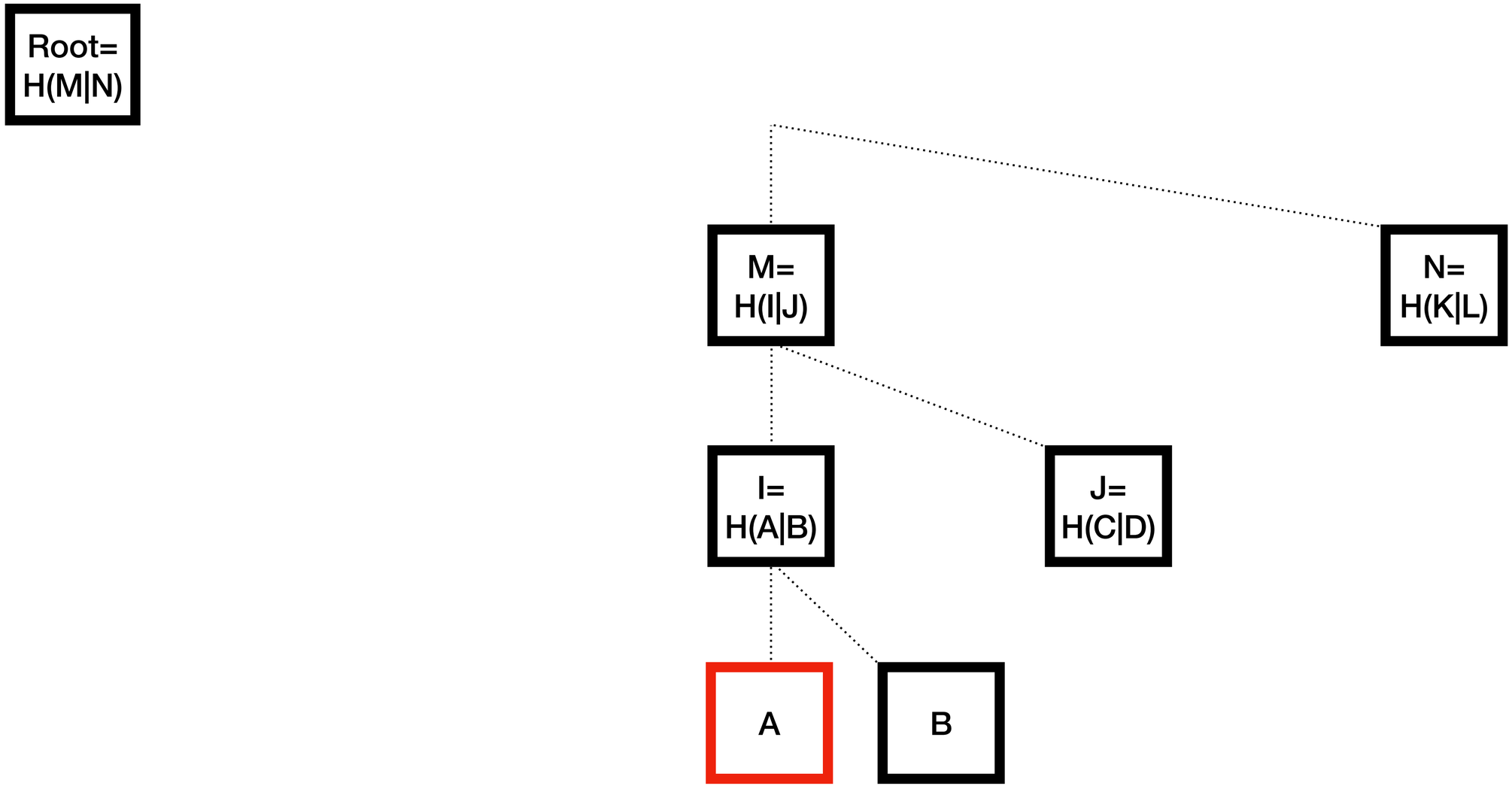
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate the hash for I



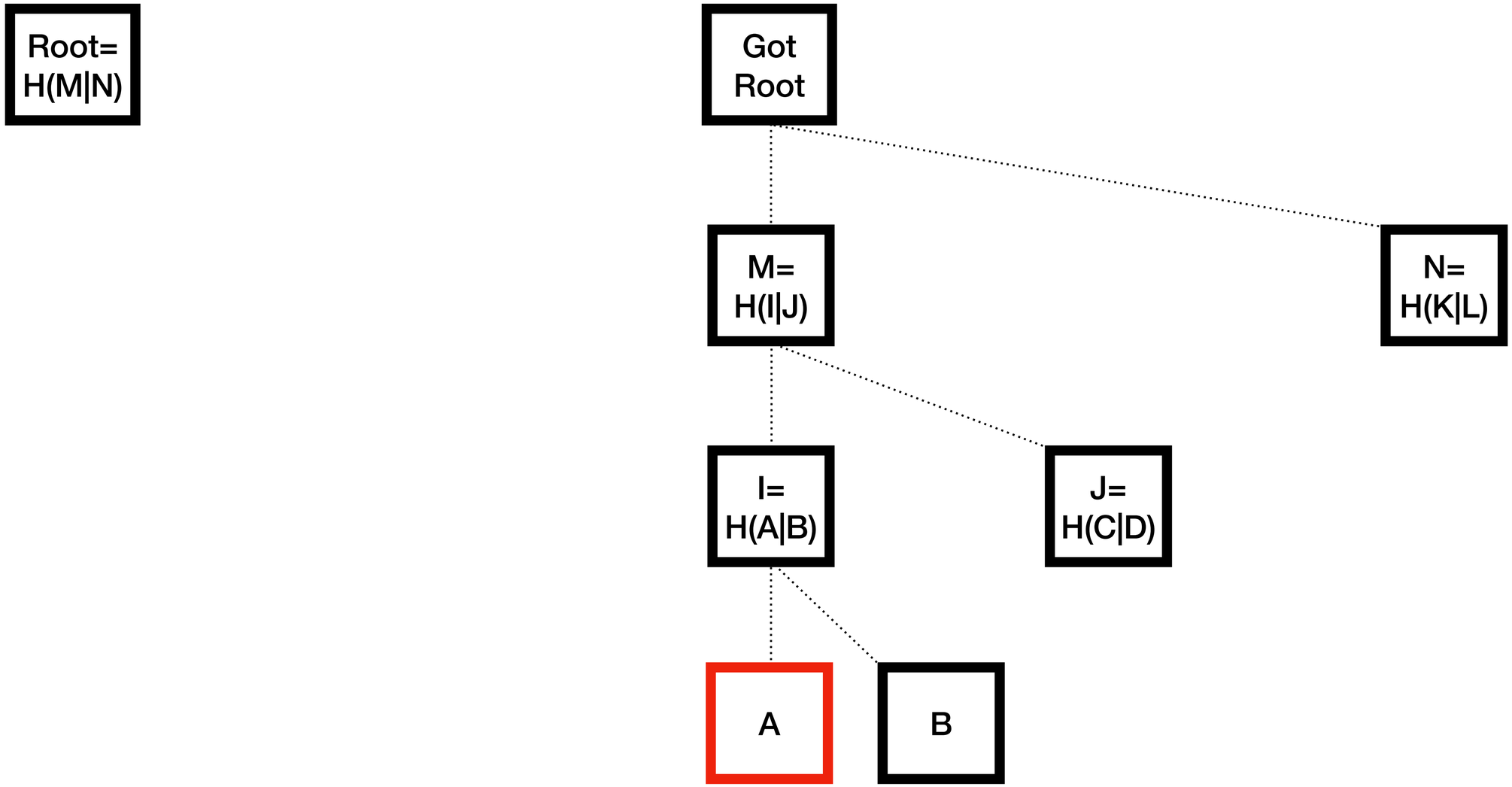
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate the hash for M



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

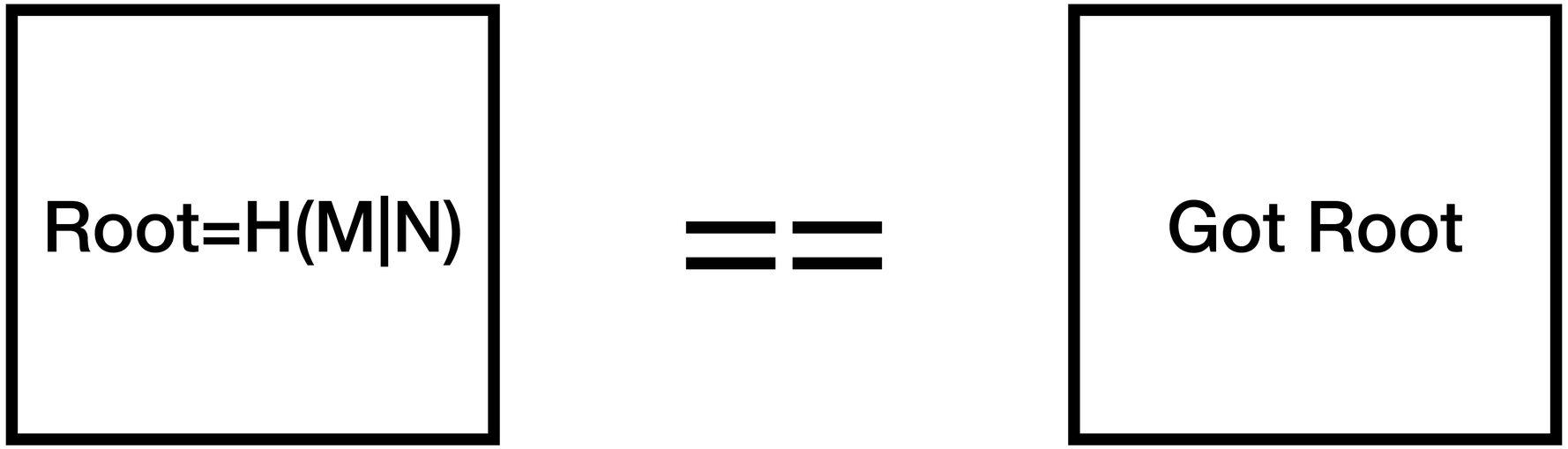
Calculate Root



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

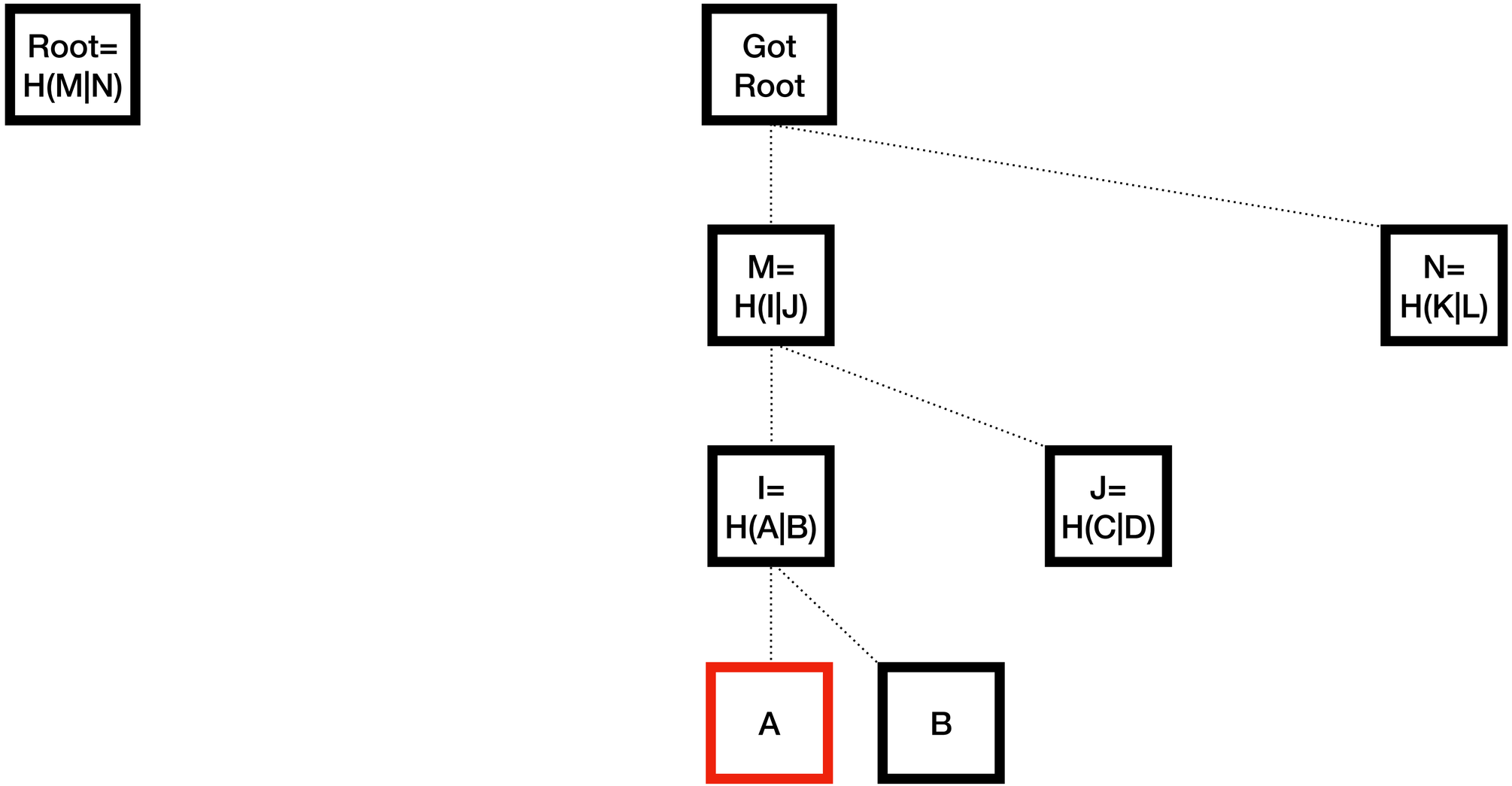
Compare roots

Continue if the roots are equal. Ban peer if not equal



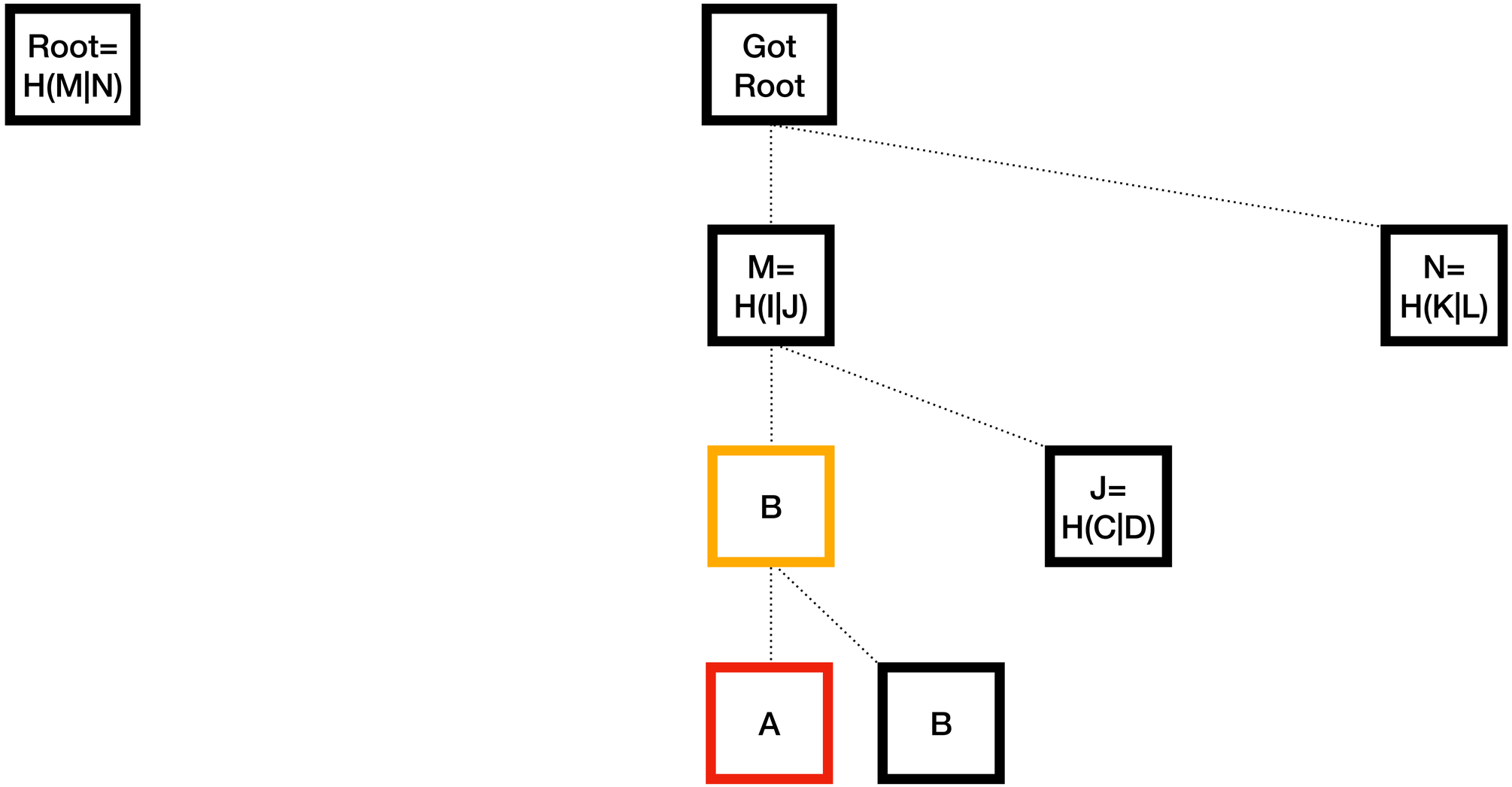
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate root after deletion



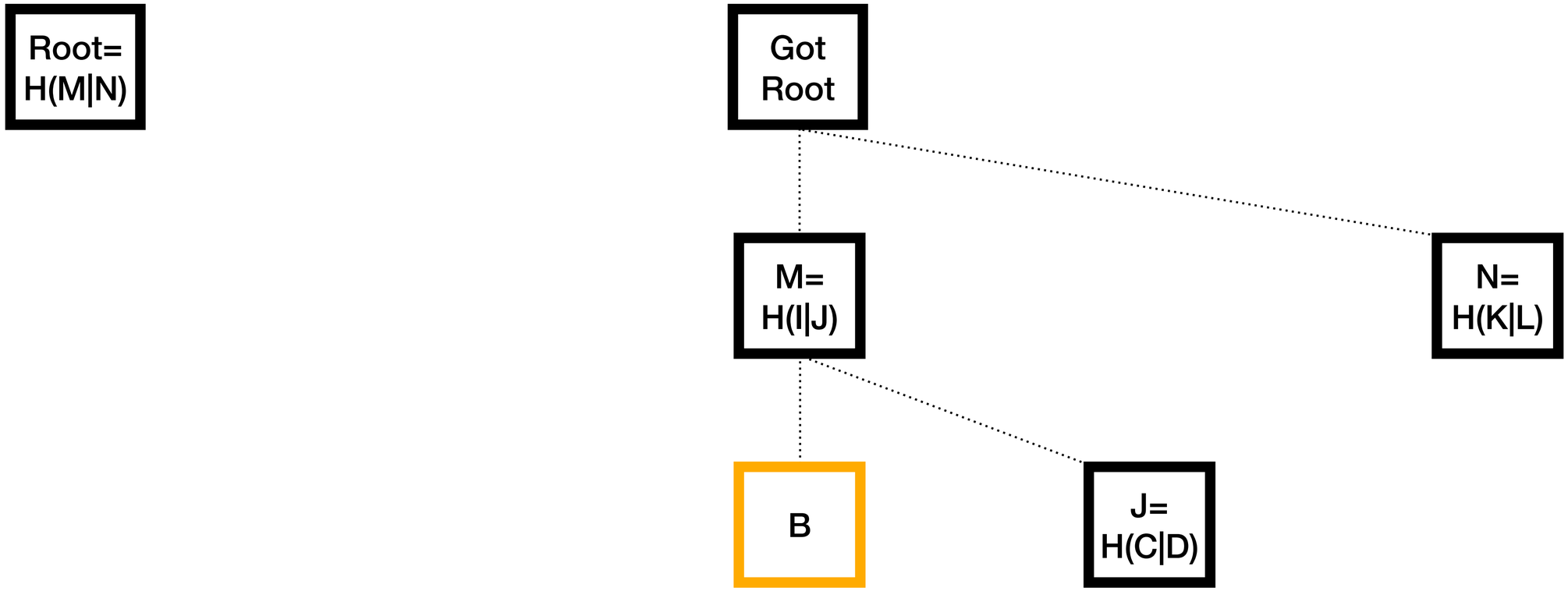
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Move up B to I



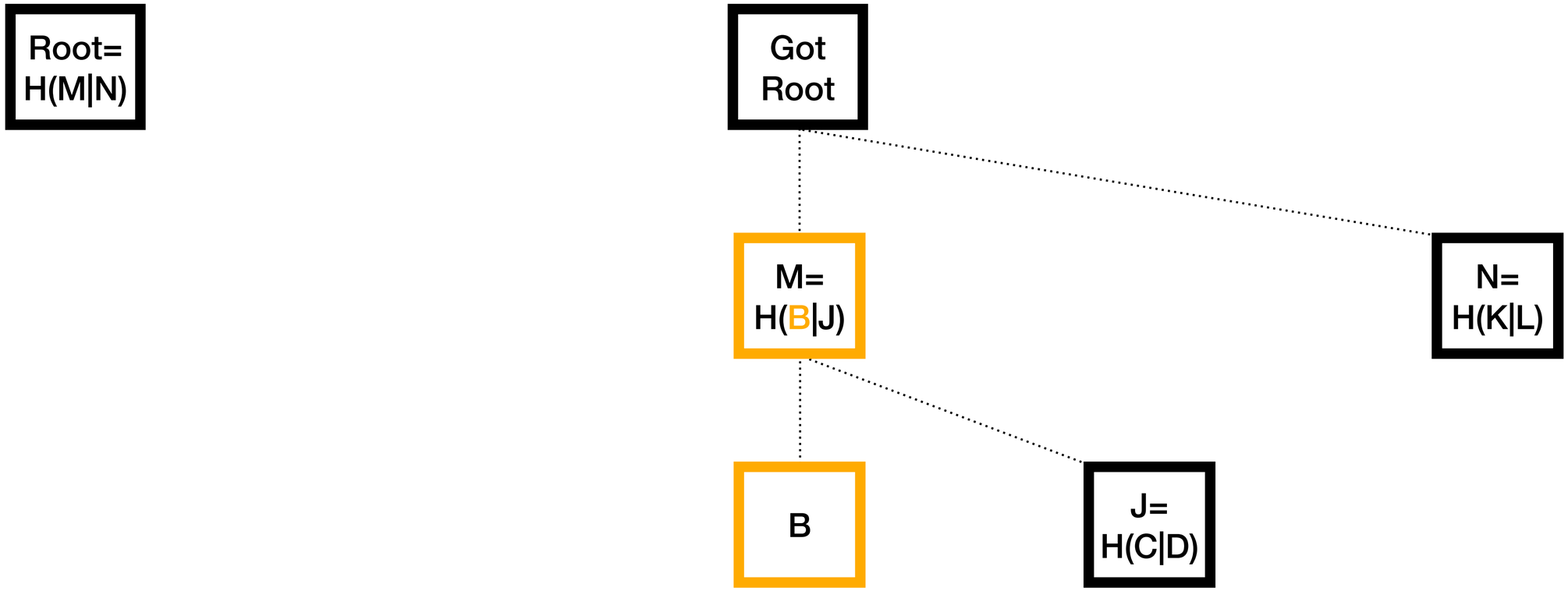
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Remove old nodes for A&B



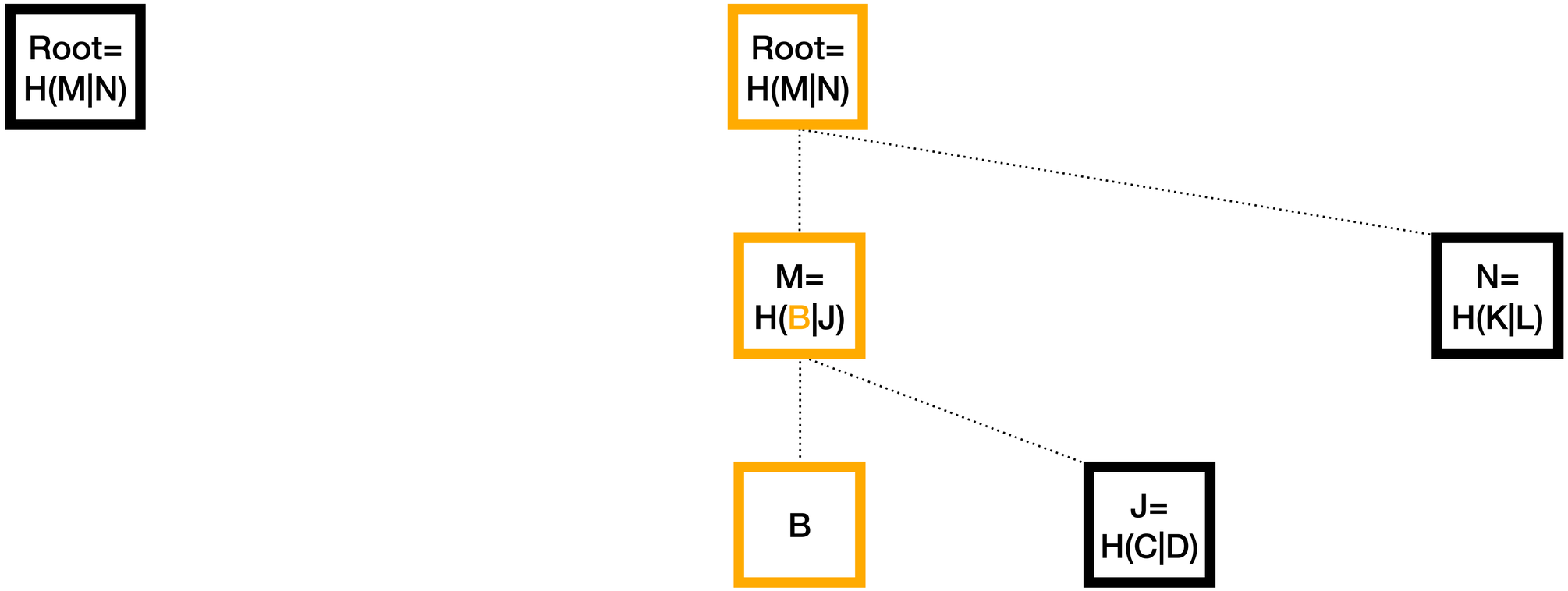
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate H(B|J)



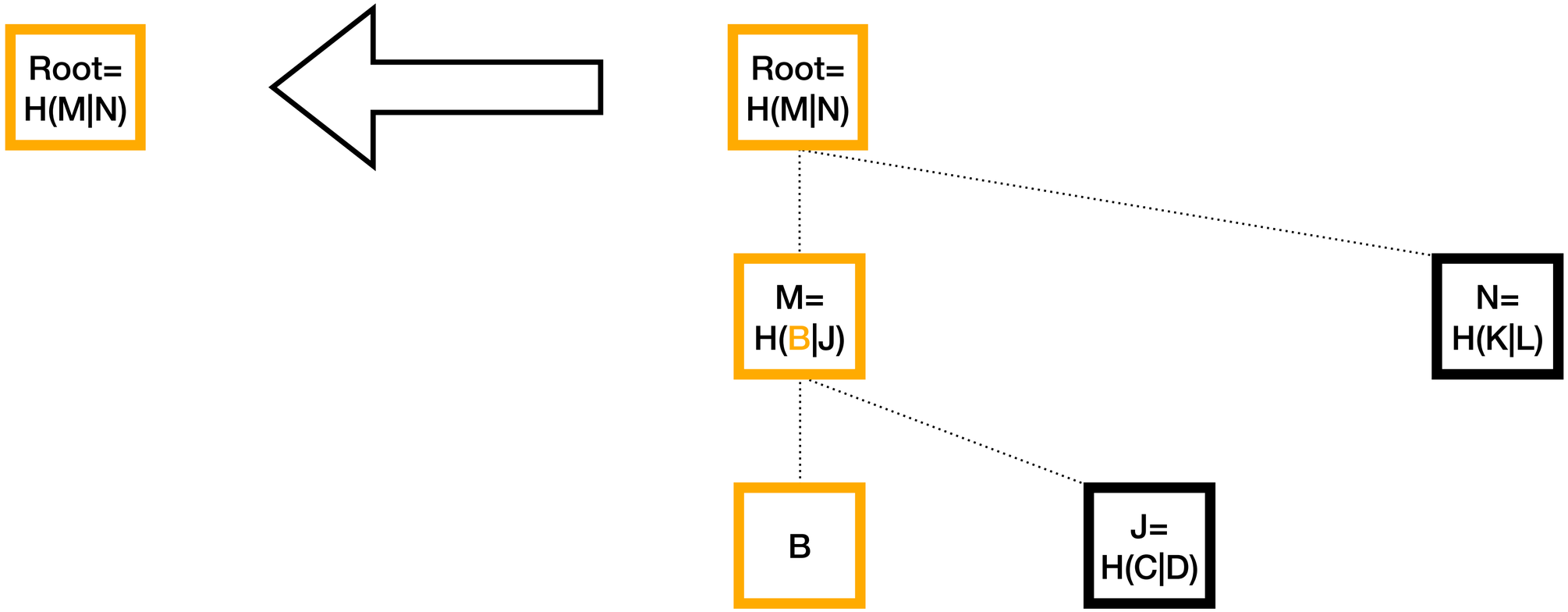
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate new root



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Copy over the new root to be saved



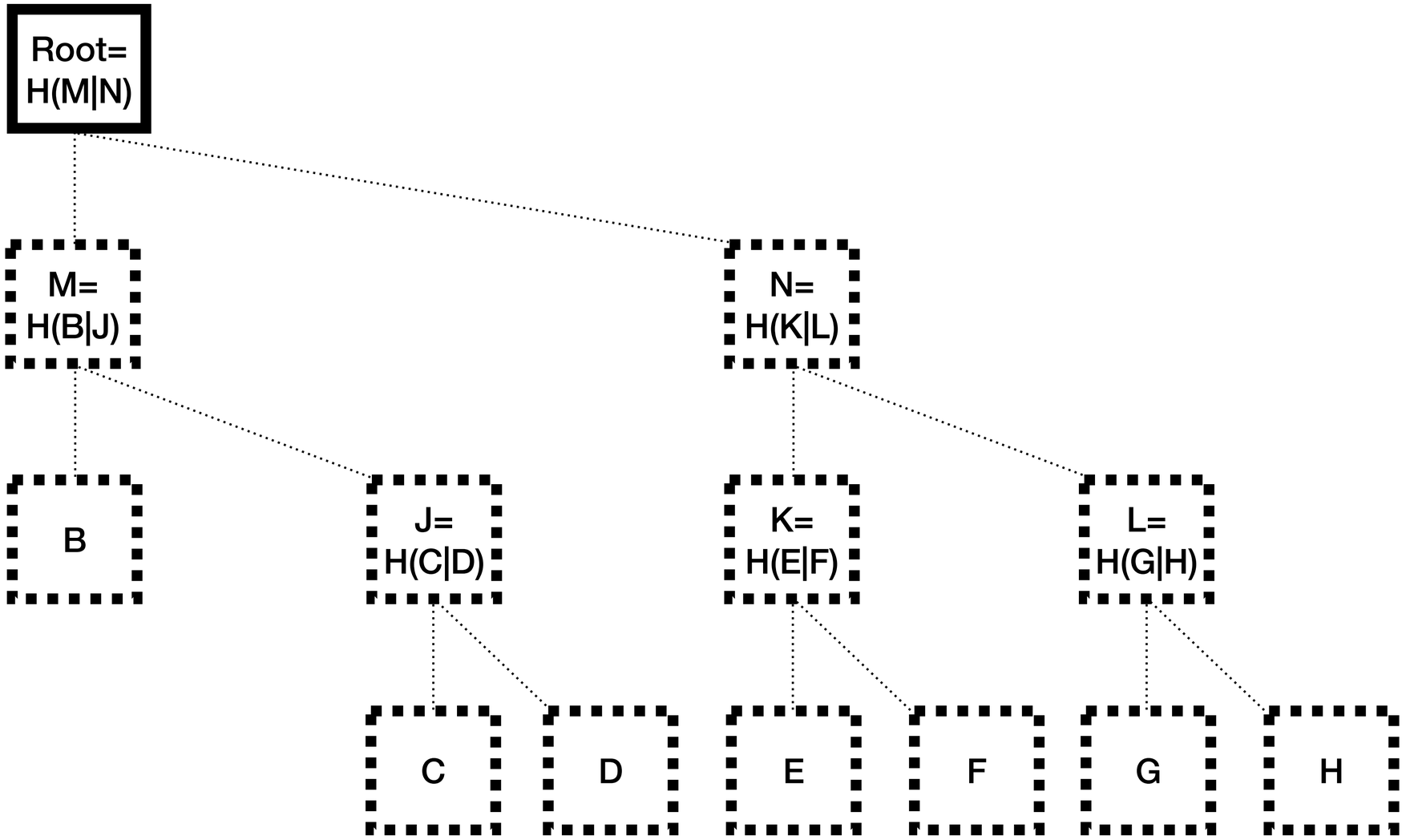
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Done

Root=
H(M|N)

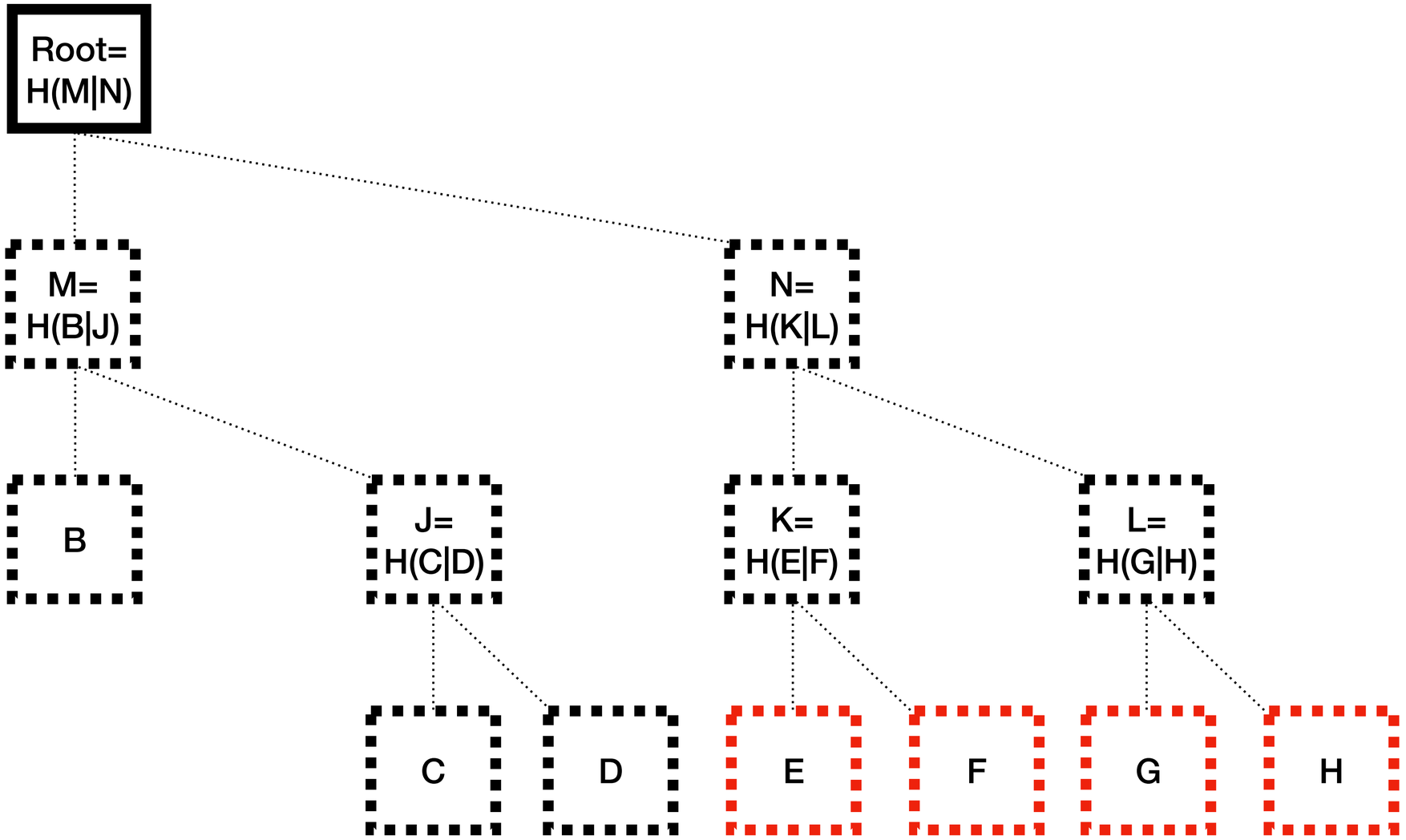
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

After deleting A



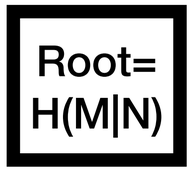
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Delete E, F, G, H. (Batched deletions)



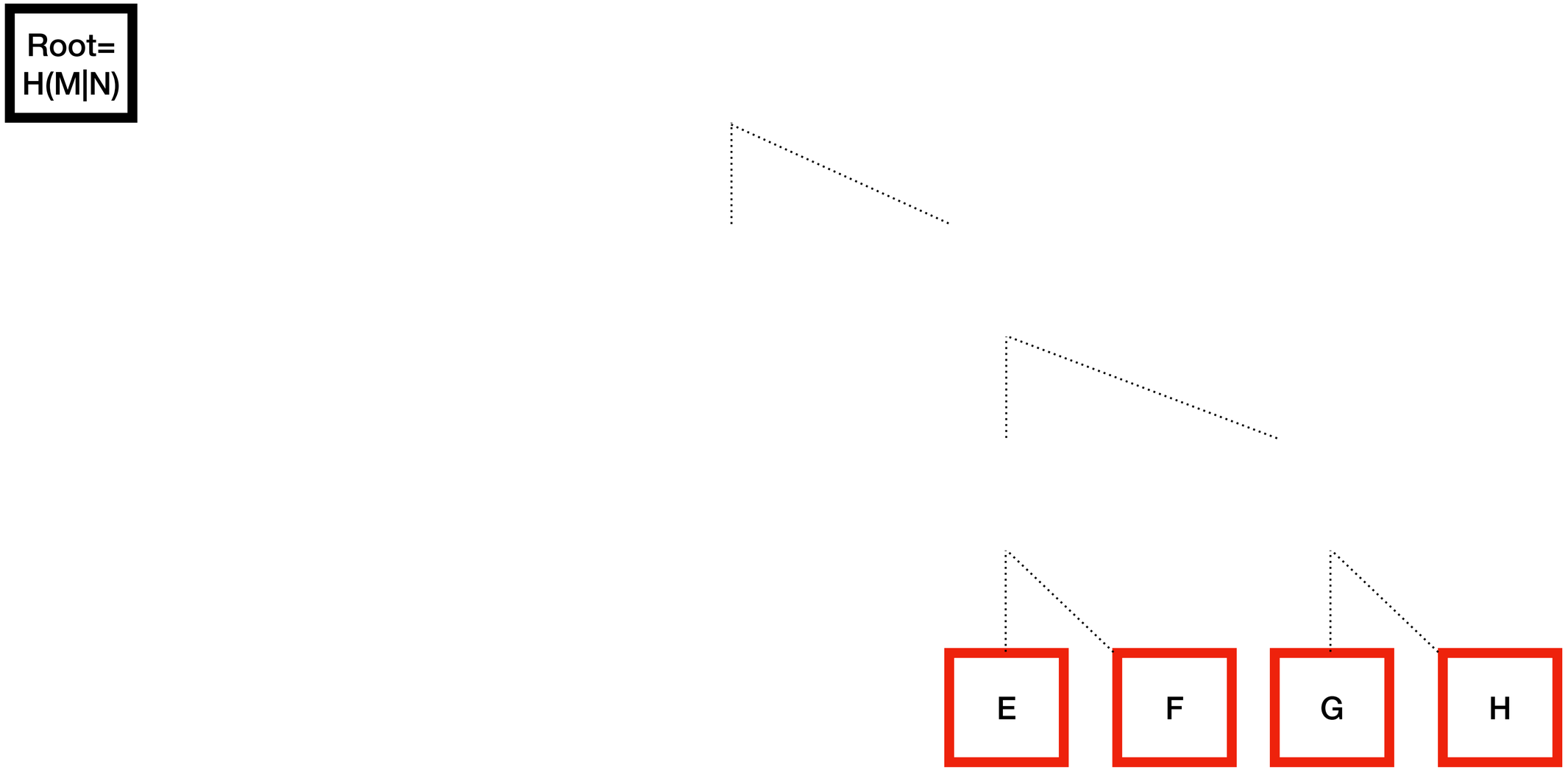
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Start off with only the root



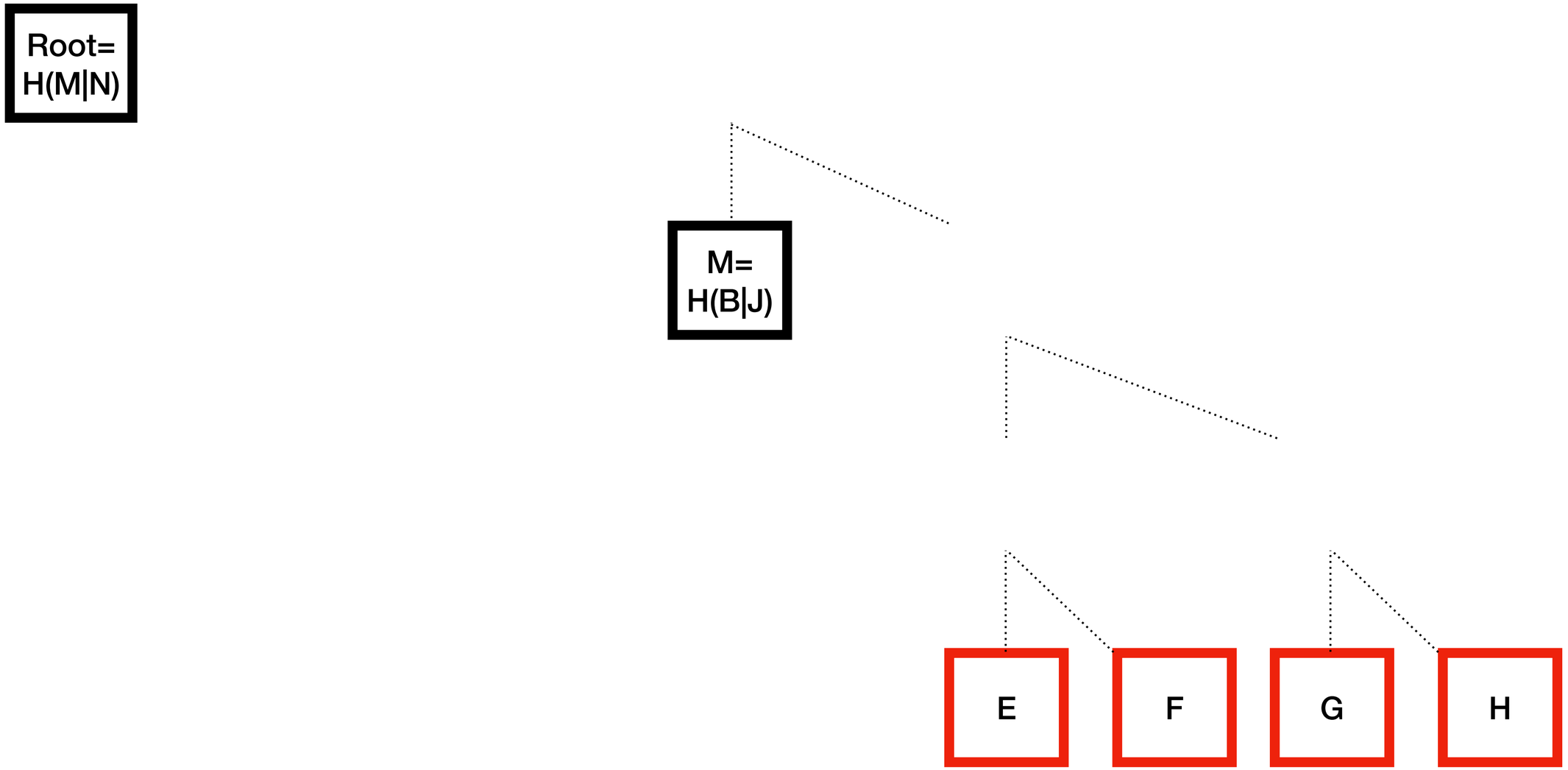
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate E, F, G, H from the Bitcoin Block



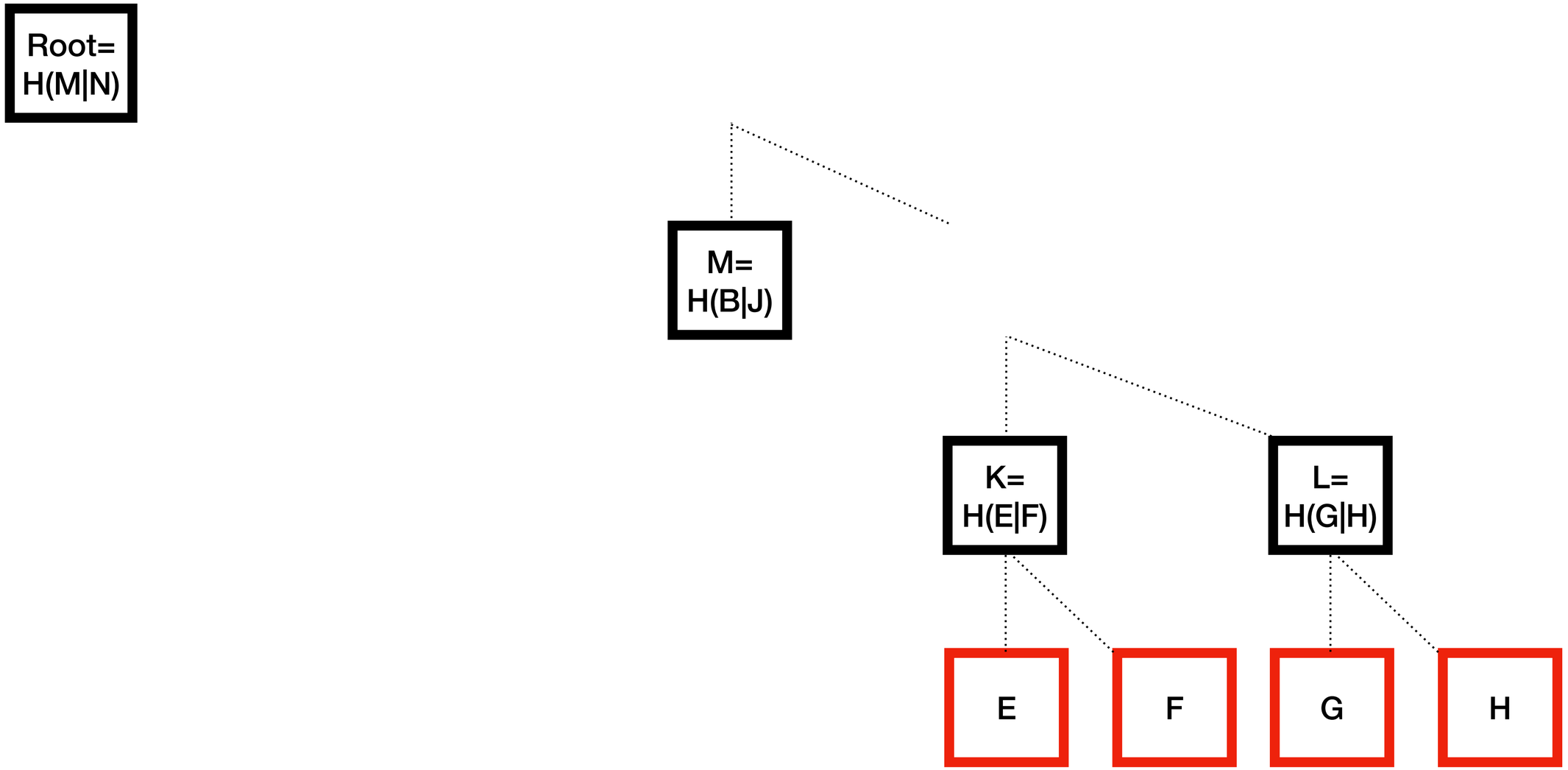
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Receive the proof: M



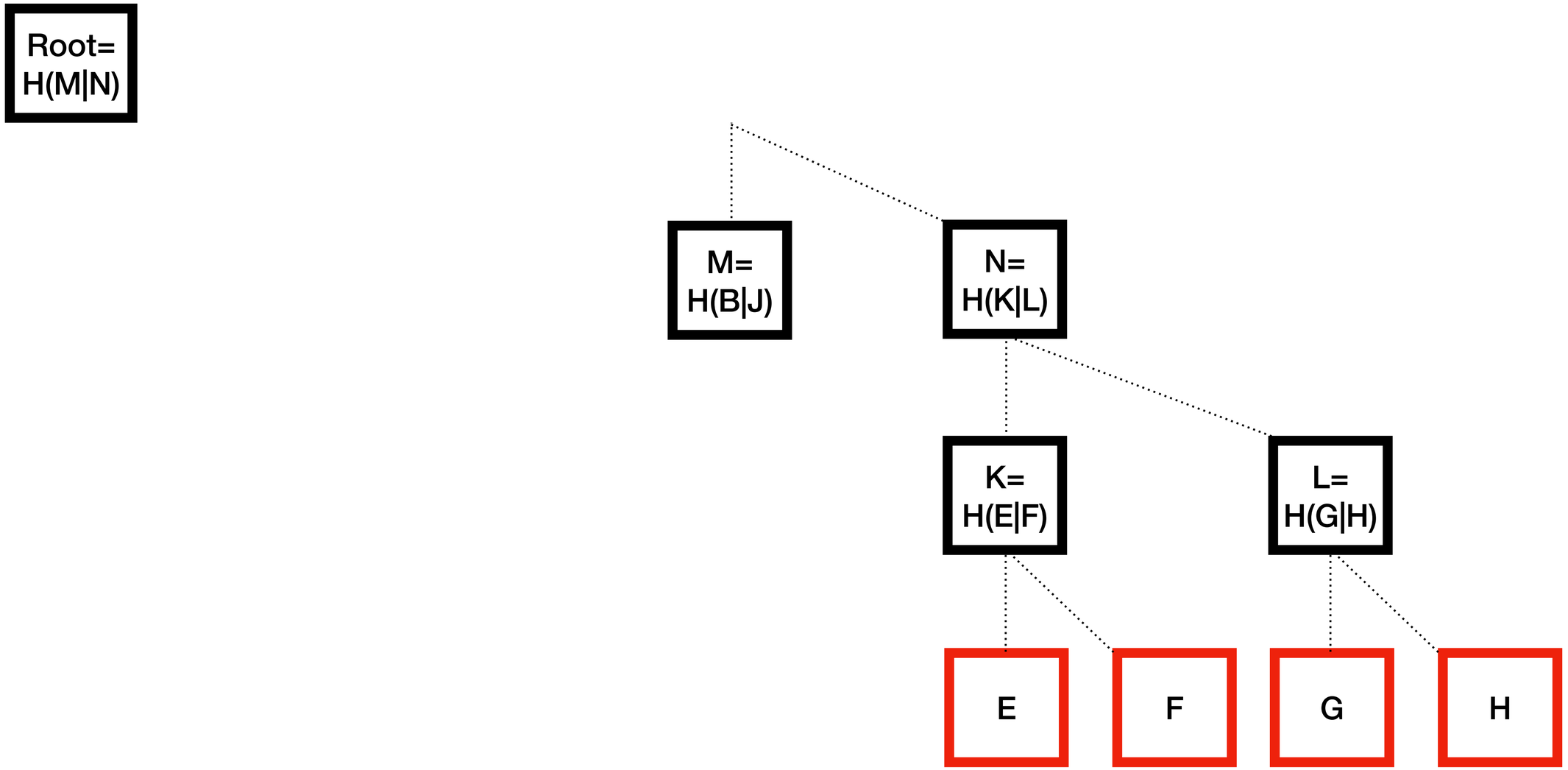
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate K and L



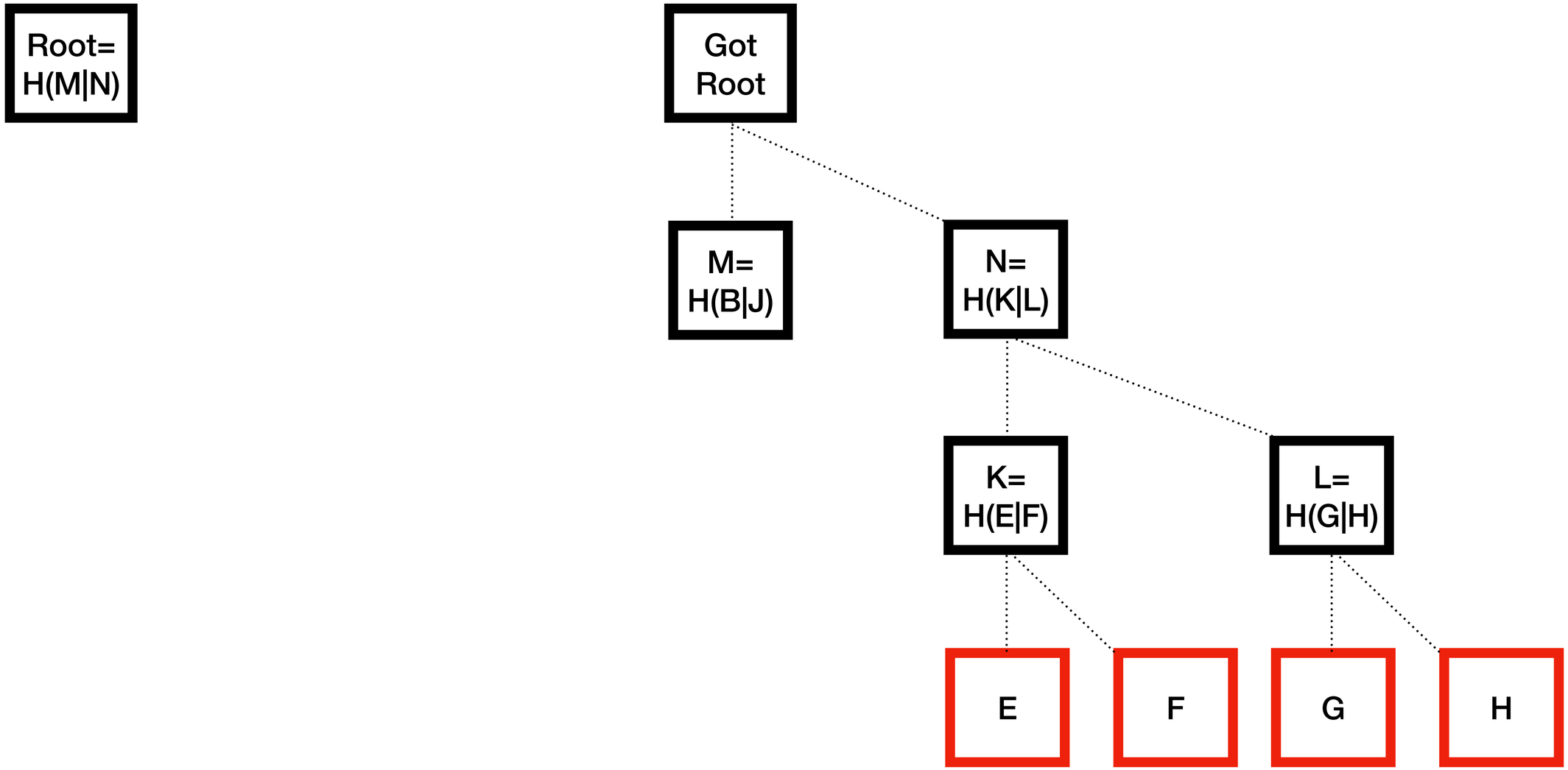
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate N



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

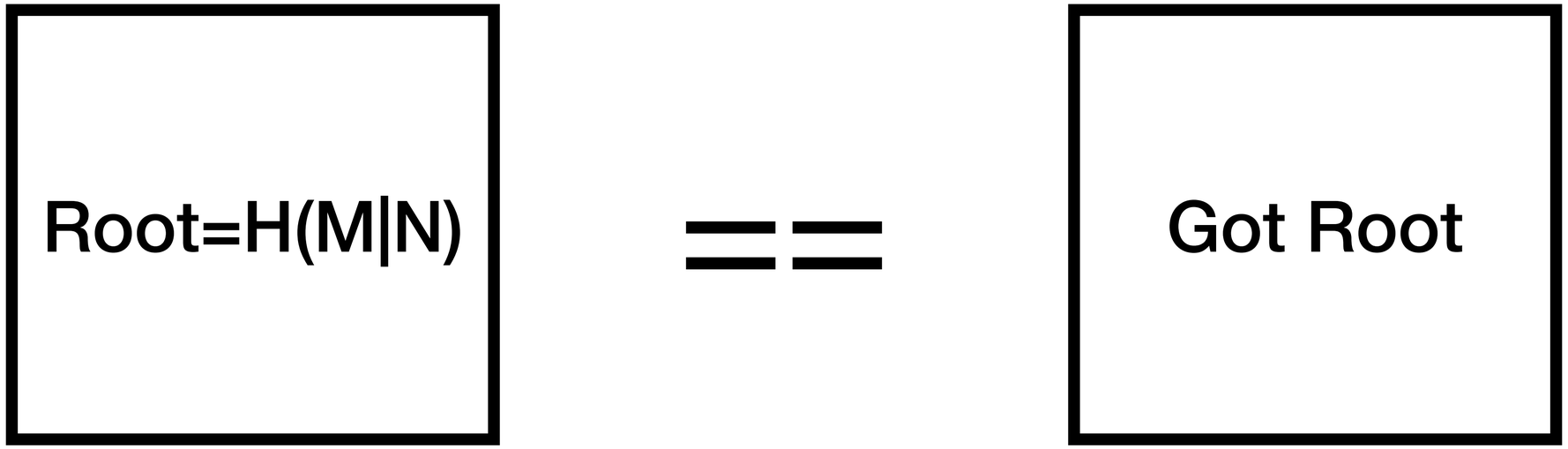
Calculate the Root



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

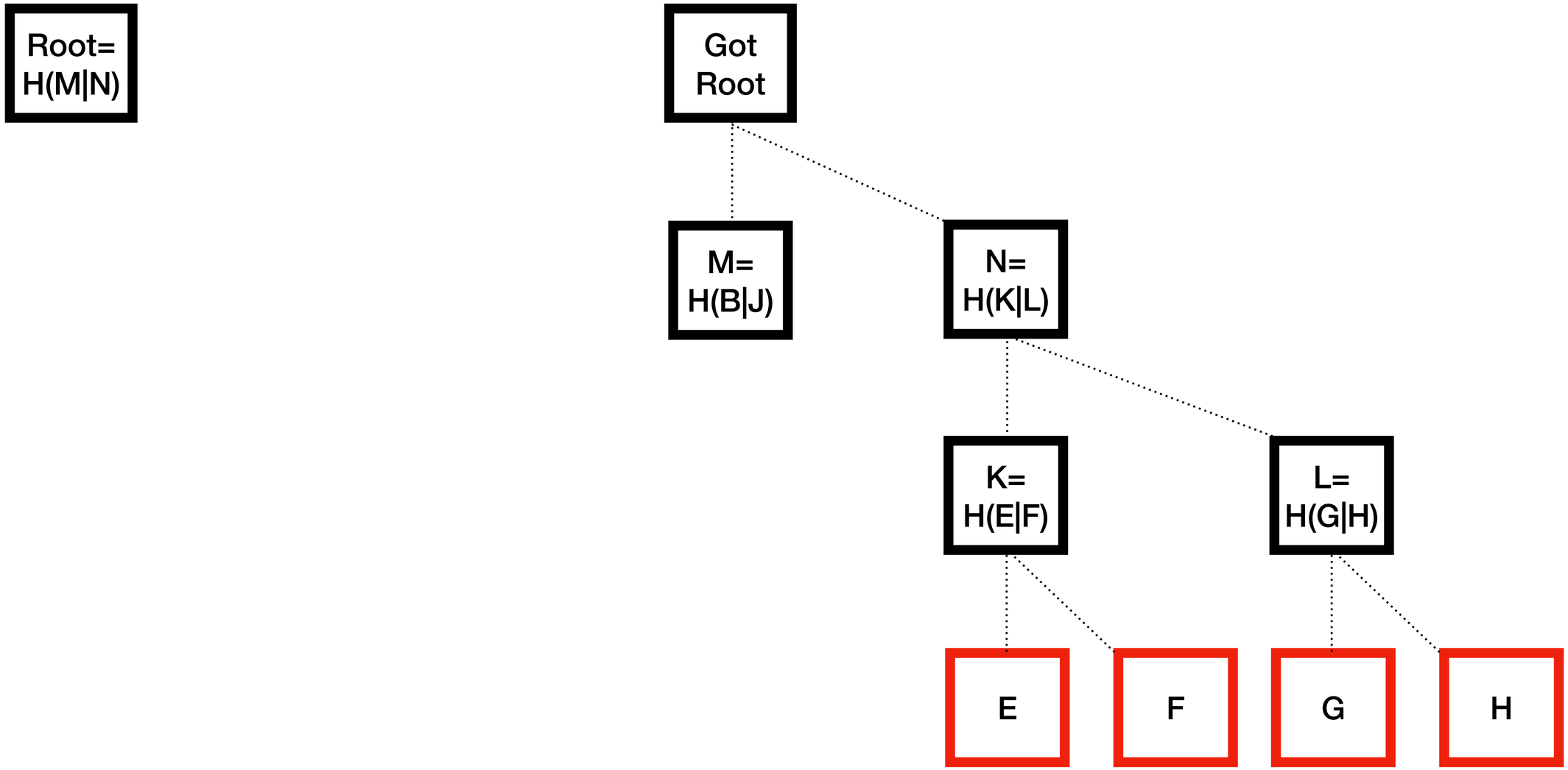
Compare roots

Continue if the roots are equal. Ban peer if not equal



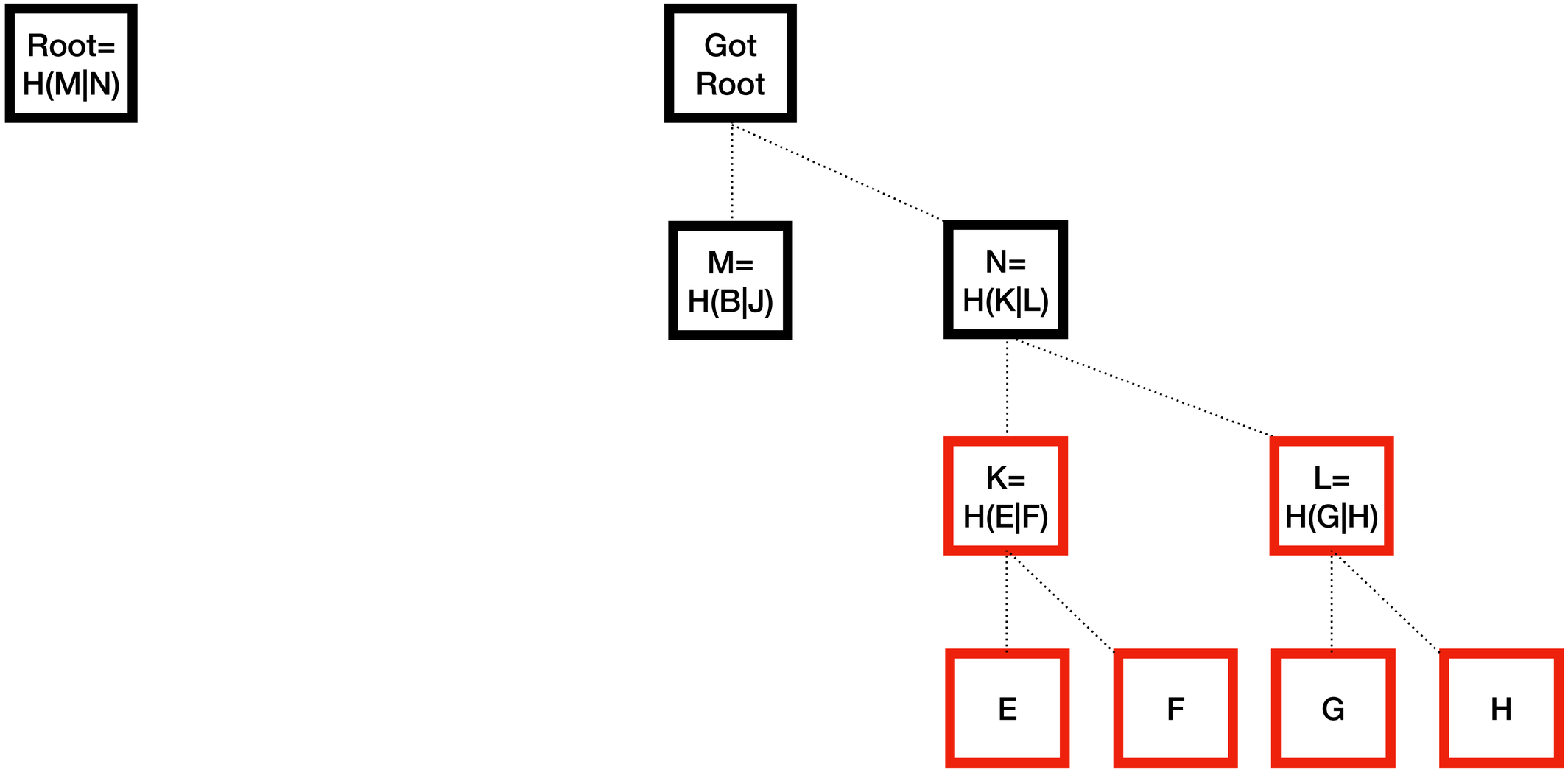
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate new root



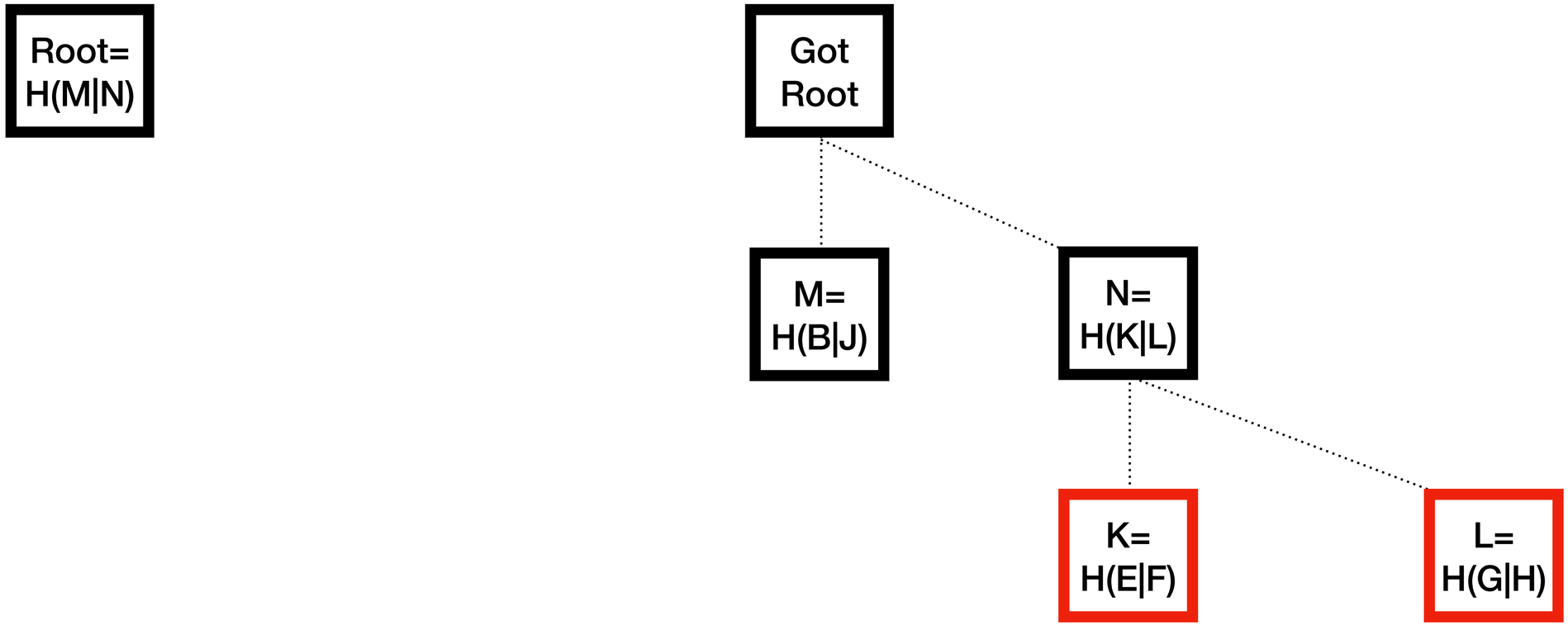
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Mark K & L as node to delete



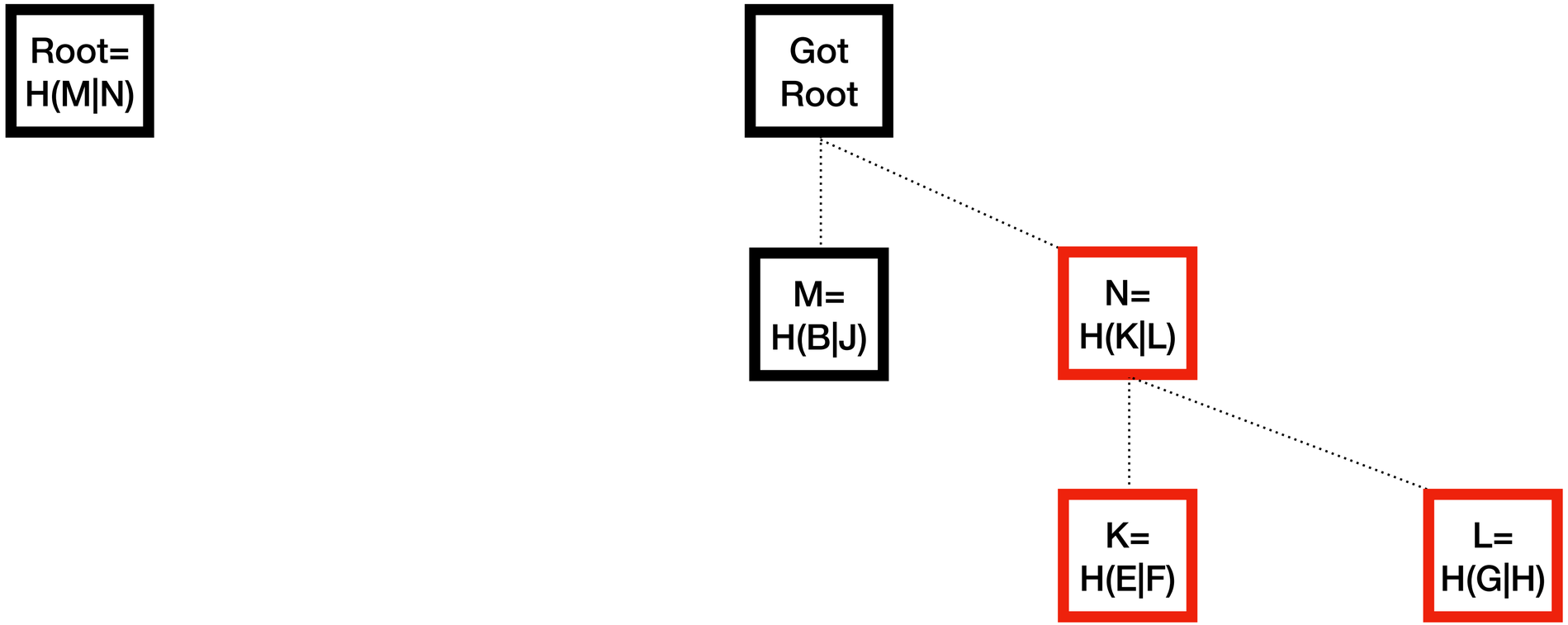
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Remove old nodes for E, F, G, H



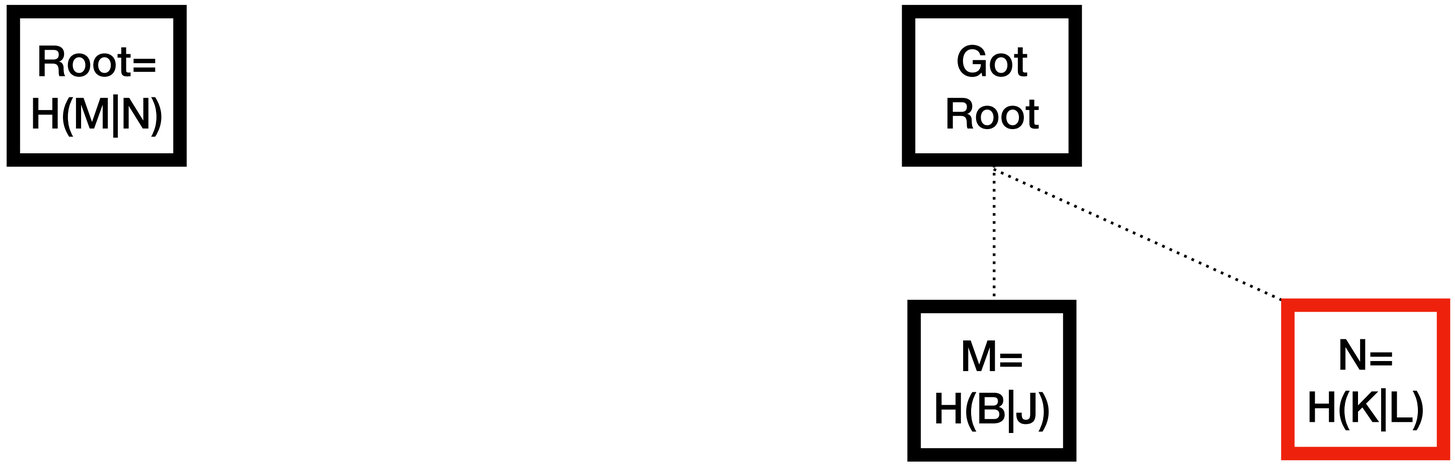
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Mark N as node to delete



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Remove old nodes for K & L



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

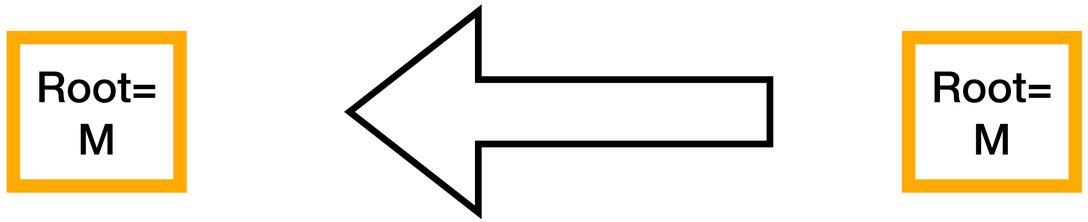
Move up M

https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Remove old nodes for M & N

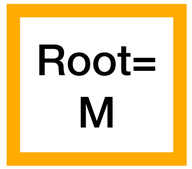
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Copy over the new root to be saved



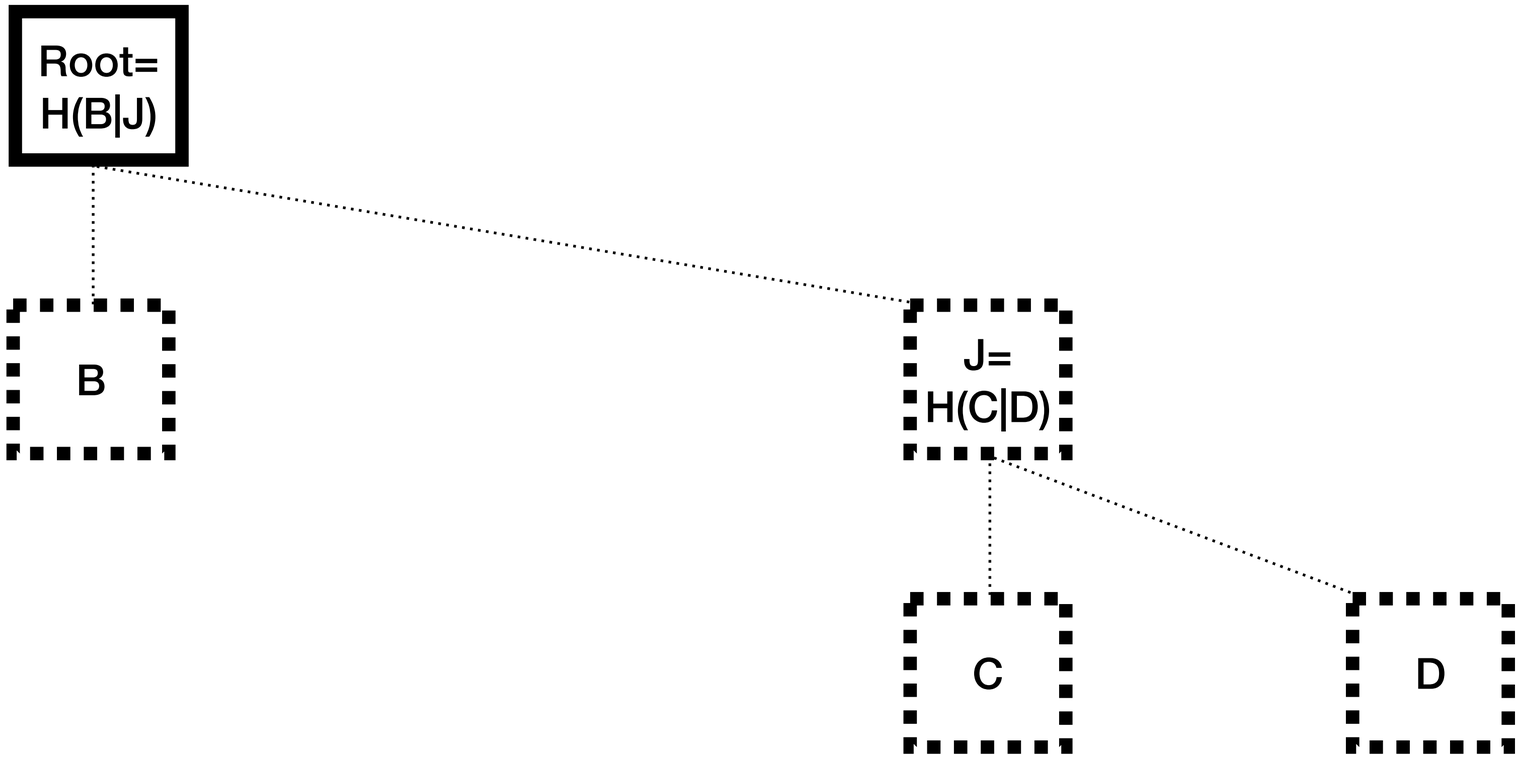
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Done



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

After deleting E, F, G, H



https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Consensus

Add, Verify, Delete

- That's all the relevant algorithm for consensus
- Implemented in 174 lines of Python: github.com/utreexo/pytreexo
- Thanks theStack! (Sebastian Falbesoner)

Role of levelDB

It let’s you

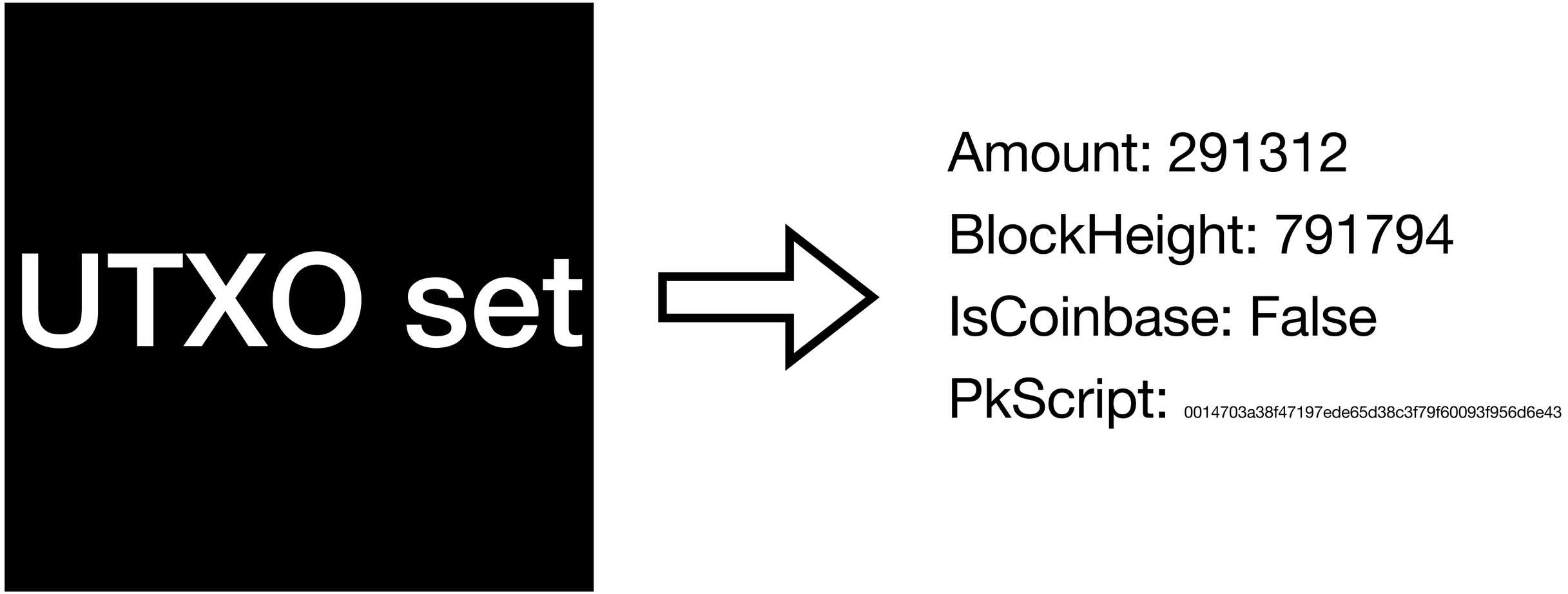
1. Add a UTXO
2. Delete a UTXO
3. Tell you the existence of a UTXO
4. Provide the data for verification

UTXO set

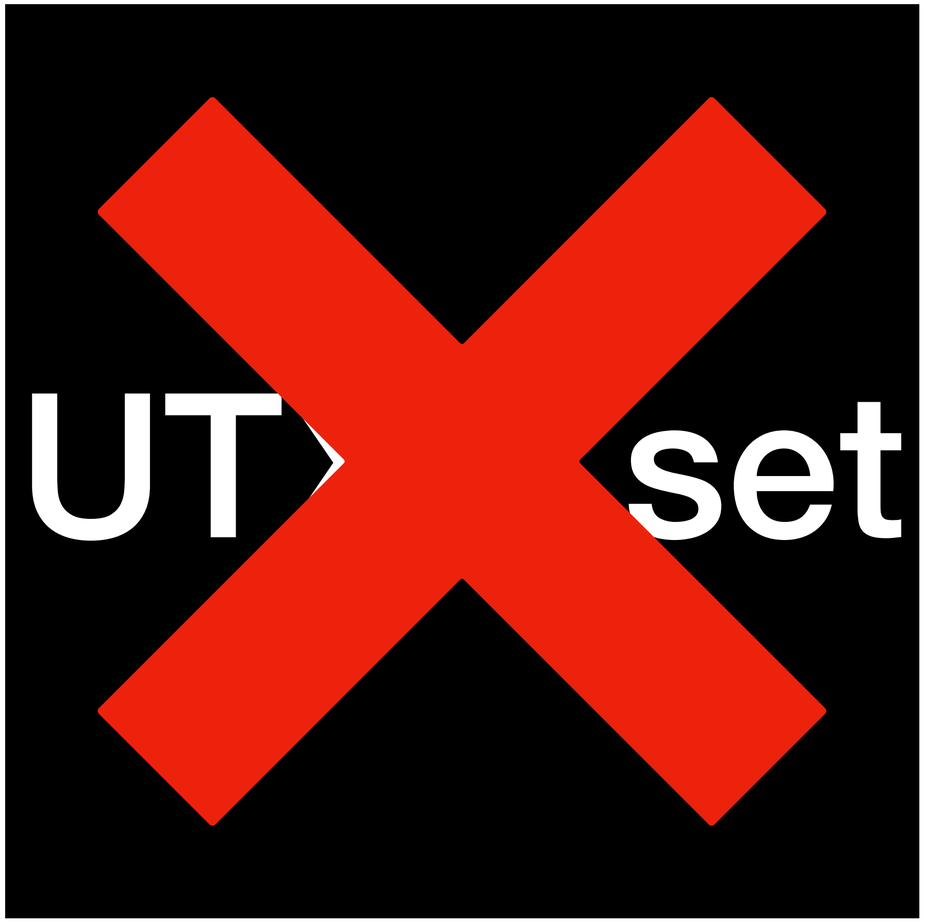
Fetching a UTXO

UTXO set

Fetching a UTXO



No UTXO set with Utreexo



How the data is provided

What Uteexo nodes download

- Data is provided by the peer
- Sent along with every block or transaction

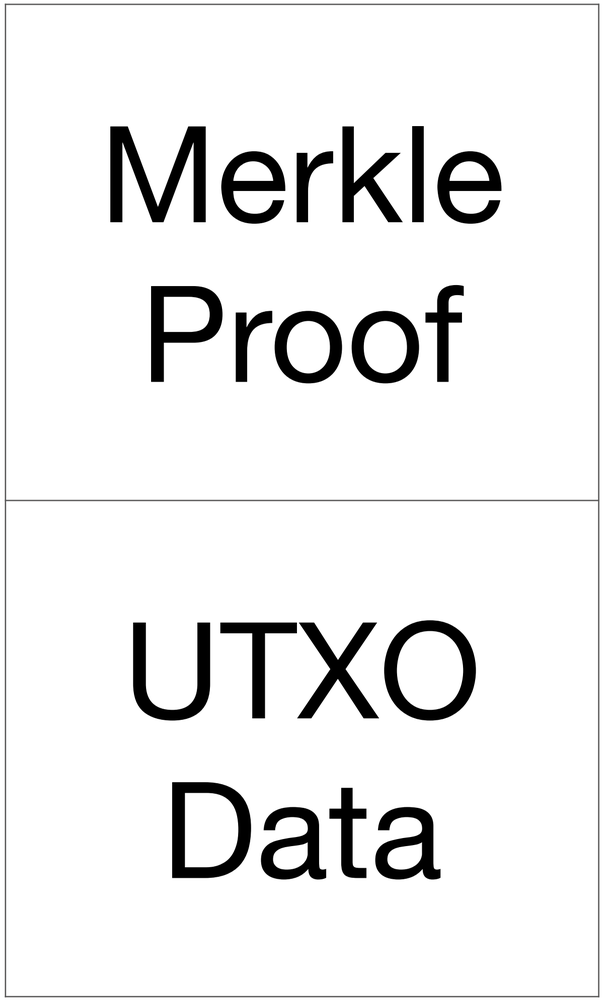
TX serialization for Utreexo nodes

Not finalized!

Version
Flag
TxIn Count
TxIns
TxOut Count
TxOuts
Witness
Locktime
Utreexo Proof Data

Utreexo Proof Data

Not finalized!



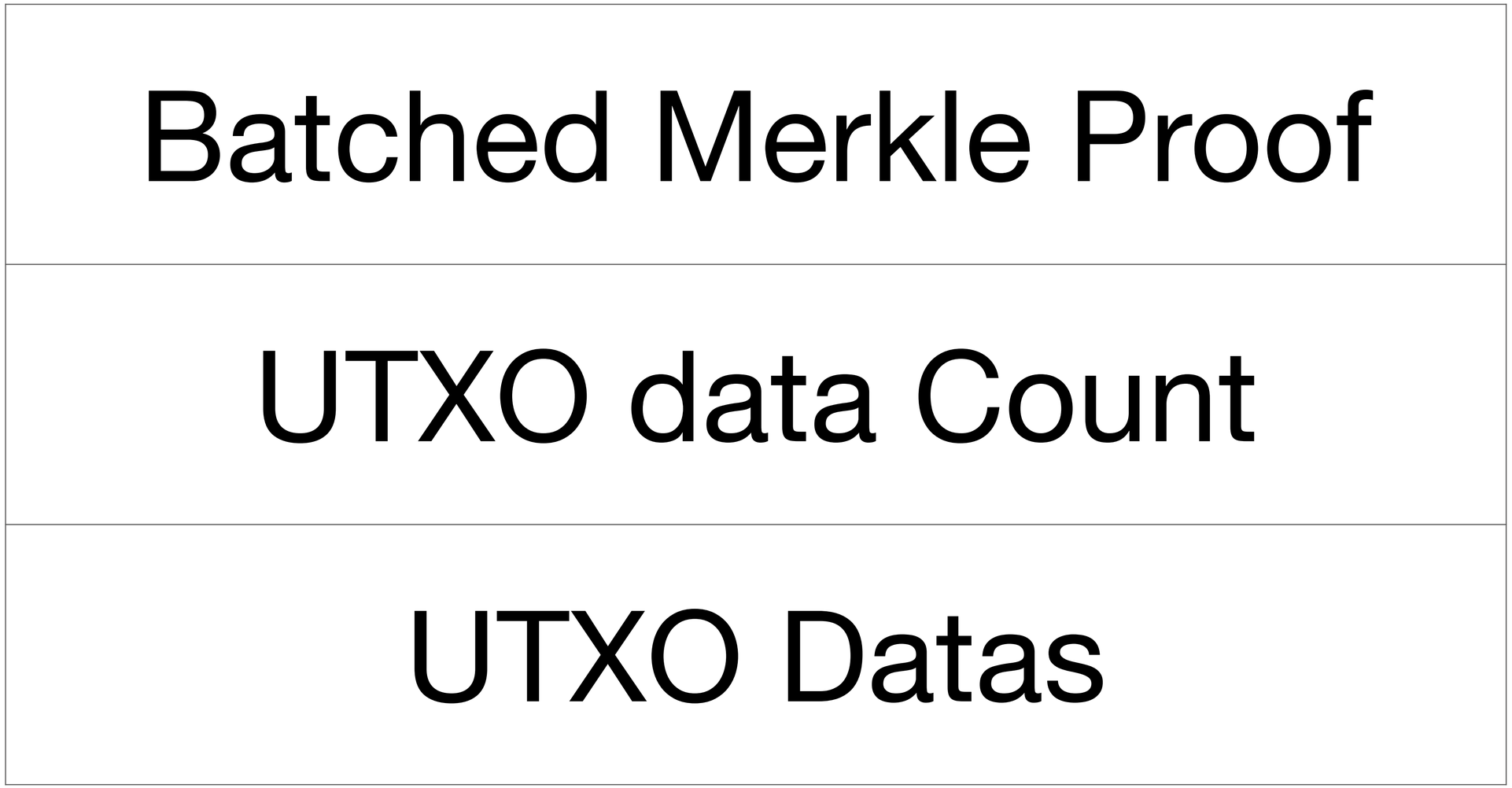
Block serialization for Utreexo nodes

Not finalized!

Version
Previous Block Hash
Merkle Root
Timestamp
Difficulty Bits
Nonce
Transaction Count
Transactions
Batched Utreexo Proof Data

Batched Utreexo Proof Data

Not finalized!



UTXO Data serialization

2 serialization methods

- For calculating the hash to be committed into the accumulator

UTXO Data serialization

For hash calculation

Block Hash	The block hash where the tx was included
TxHash	The transaction hash for the UTXO
Index (Vout)	The index within the TX for this UTXO
Block Height + IsCoinBase (blockHeight << 1)&IsCoinBase	Block height and Coinbase indicator
Amount	Amount in satoshis
PkScript length	Length of the pkscript in varint
PkScript	PkScript itself

UTXO Data serialization

2 serialization methods

- For calculating the hash to be committed into the accumulator
- For sending data to peers/storage on disk

UTXO Data serialization

For disk storage/p2p transfer

Block Height + IsCoinBase (blockHeight << 1)&IsCoinBase	Block height and Coinbase indicator
Amount	Amount in satoshis
IsReconstructablePkScript	True if pkscript can be fetched from the spending TxIn. Nil pkscript for p2pkh&p2sh
PkScript length	Length of the pkscript in varint
PkScript	PkScript itself

Twitter:
@kcalvinalvinn

Slides:
github.com/kcalvinalvin/
slides-for-taiwan-bitdevs