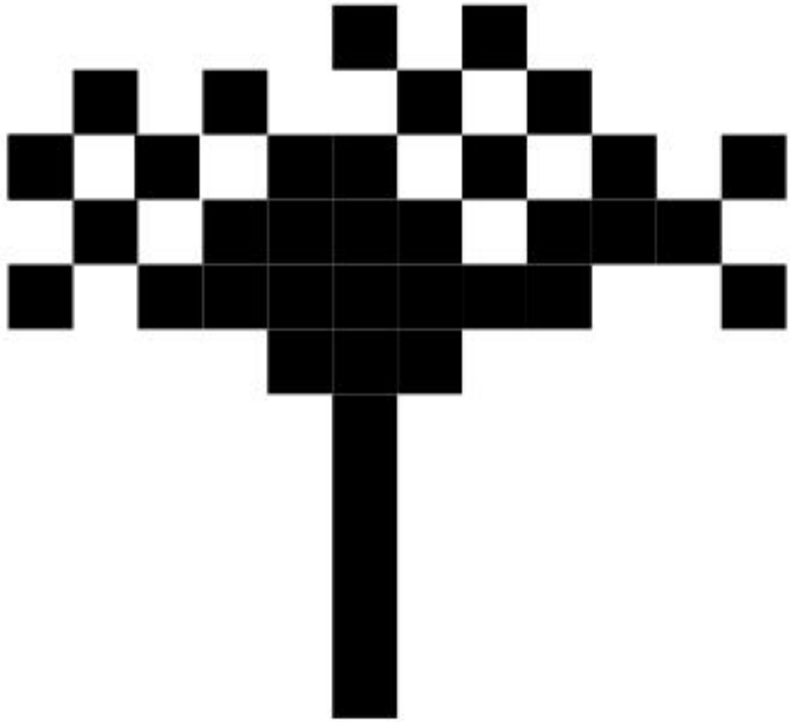


Utreexo - What is it?

What it is and how it works

Utreexo

Agenda for today

What is it?

How does it work?

What is it?

Turn the UTXO set into a Merkle tree

The UTXO set

Quick overview

- TXO - An Output of a TX (transaction)

Typical TX

source: https://blockstream.info/tx/8ad63210d059908229f00ad177469d9f16a5d37a7ea64cede9c92684ff7cf06e

Outputs of a TX

source: https://blockstream.info/tx/8ad63210d059908229f00ad177469d9f16a5d37a7ea64cede9c92684ff7cf06e

The UTXO set

Quick overview

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

The UTXO set

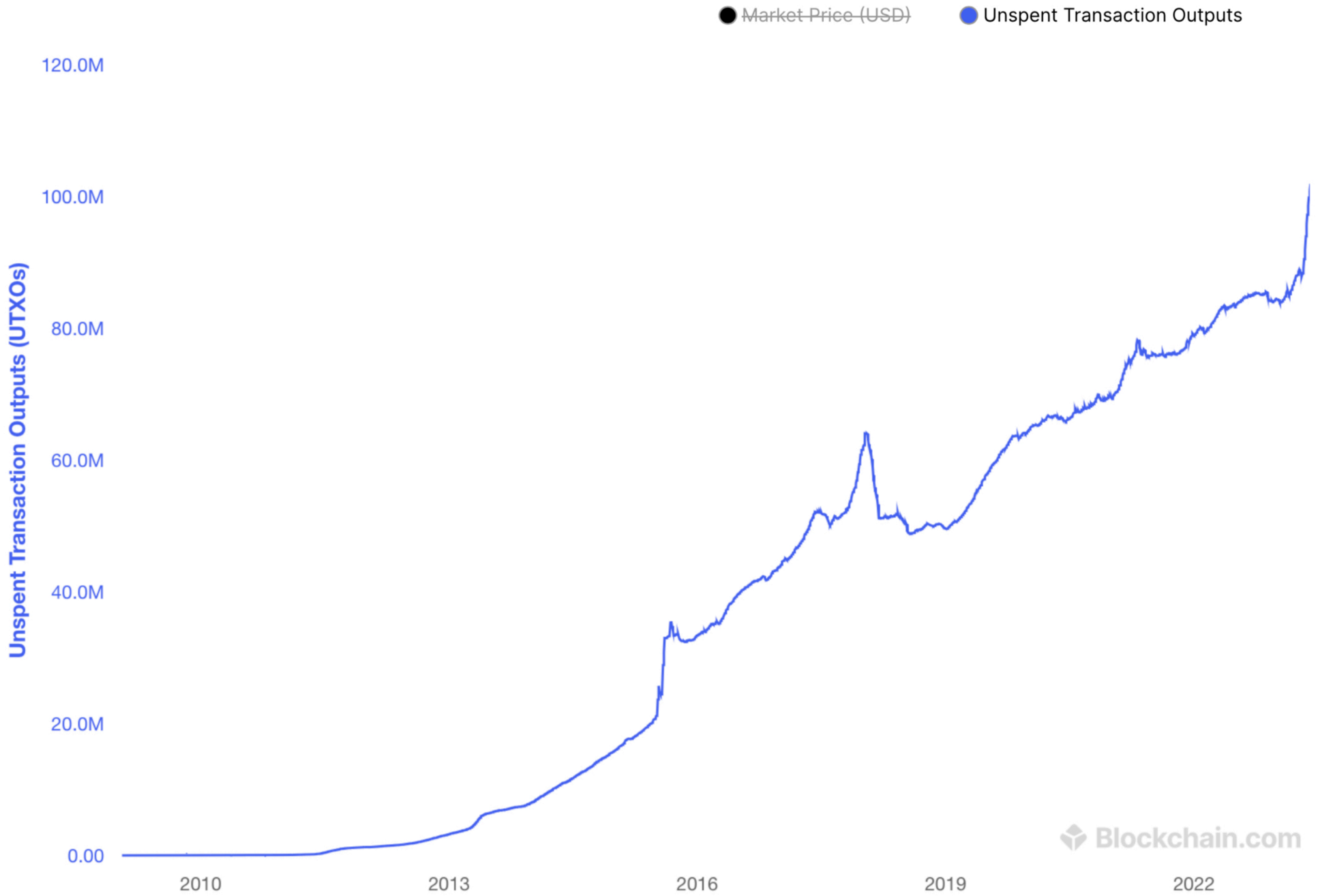
Quick overview

- TXO - An Output of a TX (transaction)
- UTXO - Unspent TXO
- UTXO set - Set of all UTXOs

The UTXO set

Why merklize it?

- Puts a bound to the UTXO set growth



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

O(N) -> O(log2 N)

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

5.8G chainstate/

384B chainstate/

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 ~5GB of UTXO set from torrents

AssumeUTXO

Quick way to sync up

- Receive a UTXO set hash commitment with the binary
- Download the ~5GB 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

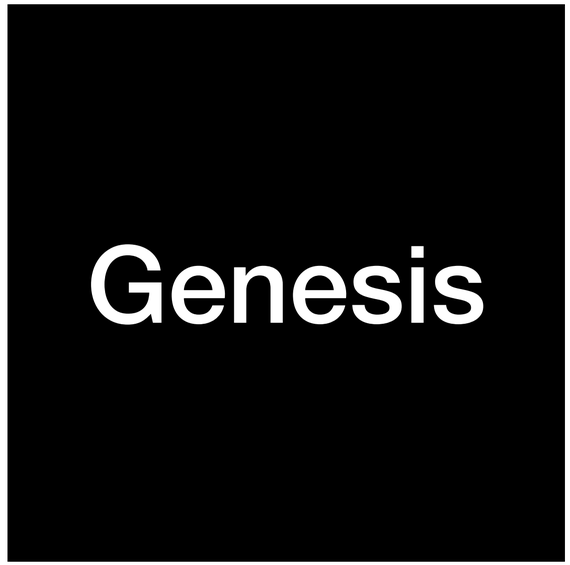
The UTXO set

Why merklize it?

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

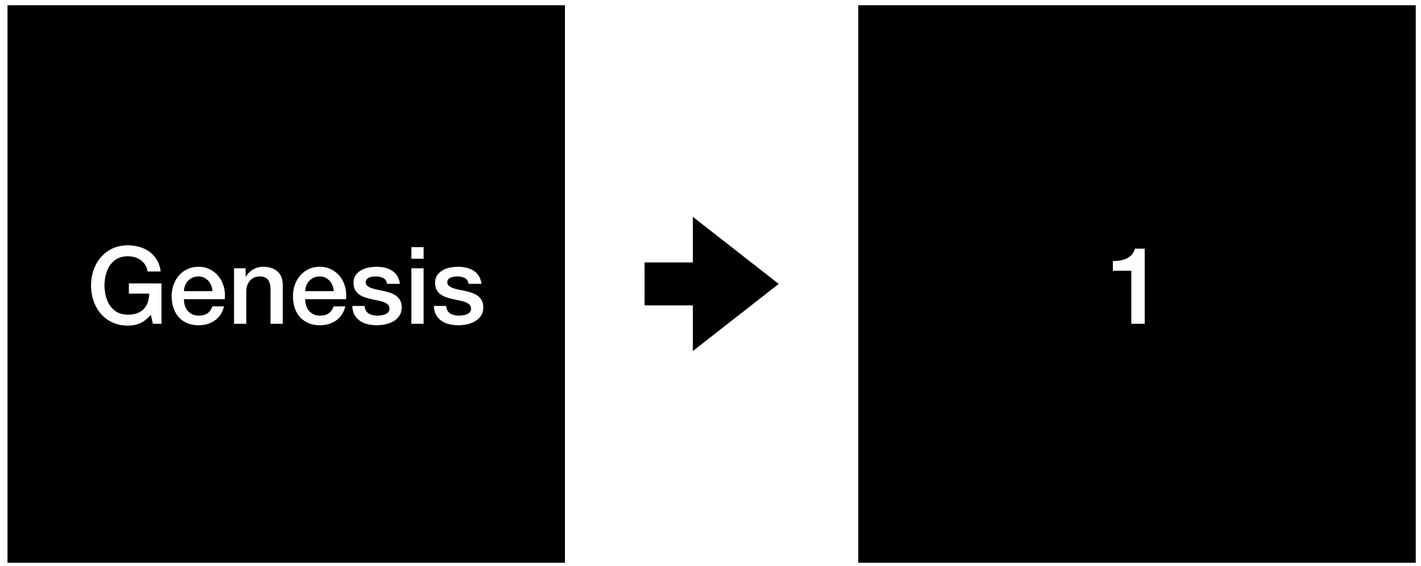
Current block validation

Sequential validation



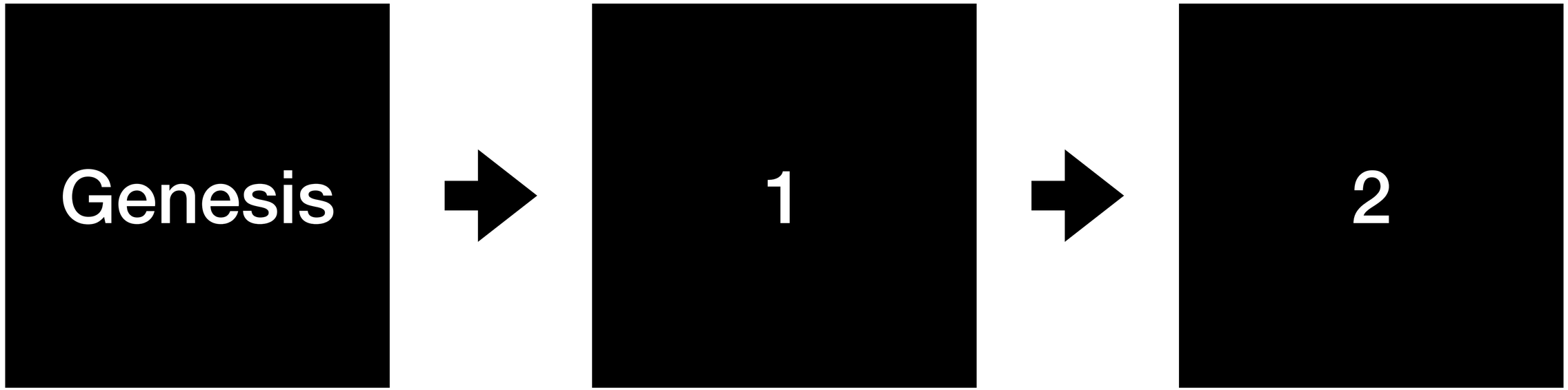
Current block validation

Sequential validation



Current block validation

Sequential validation



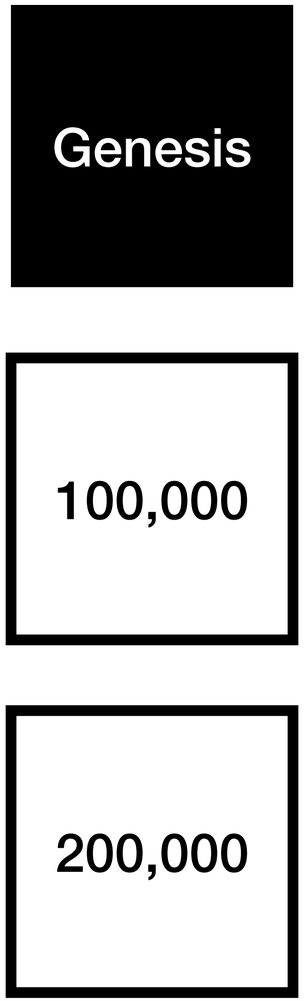
Parallel block validation

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

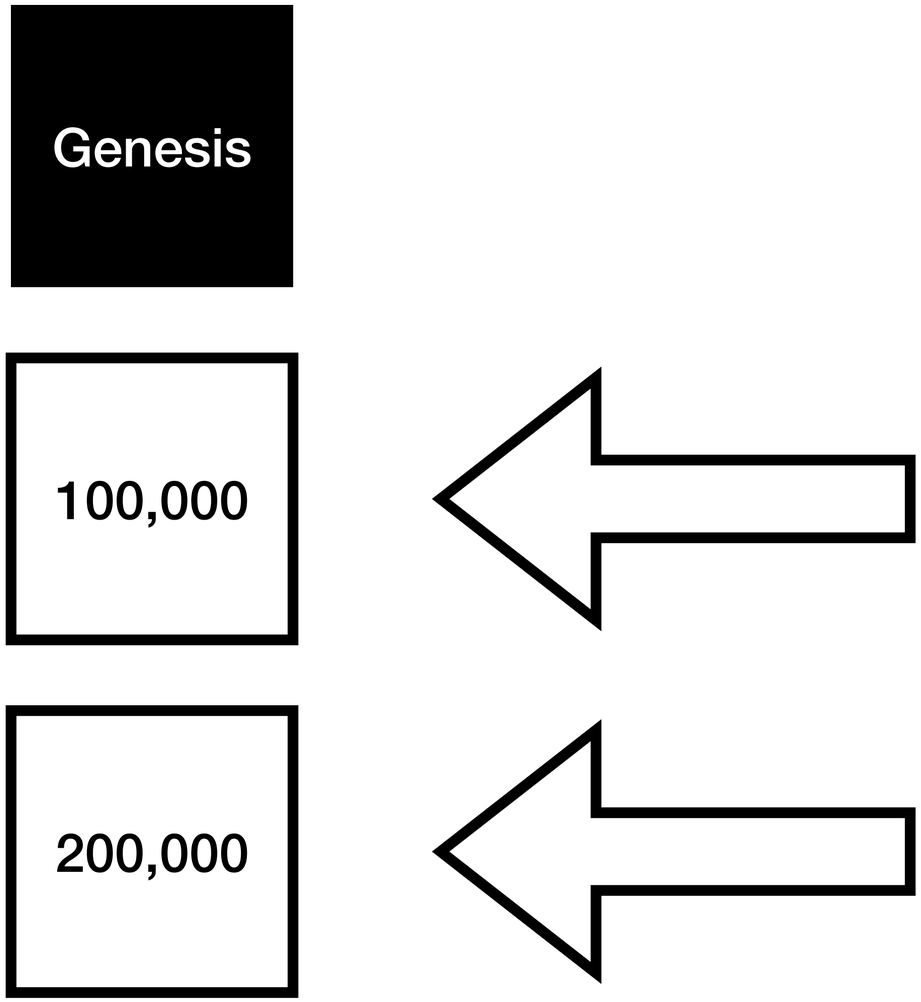
With Utreexo

Efficient parallel validation



With Utreexo

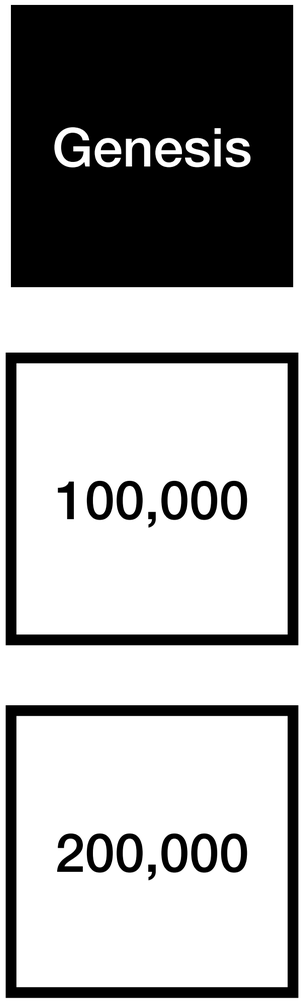
Efficient parallel validation



Untrusted roots at the start

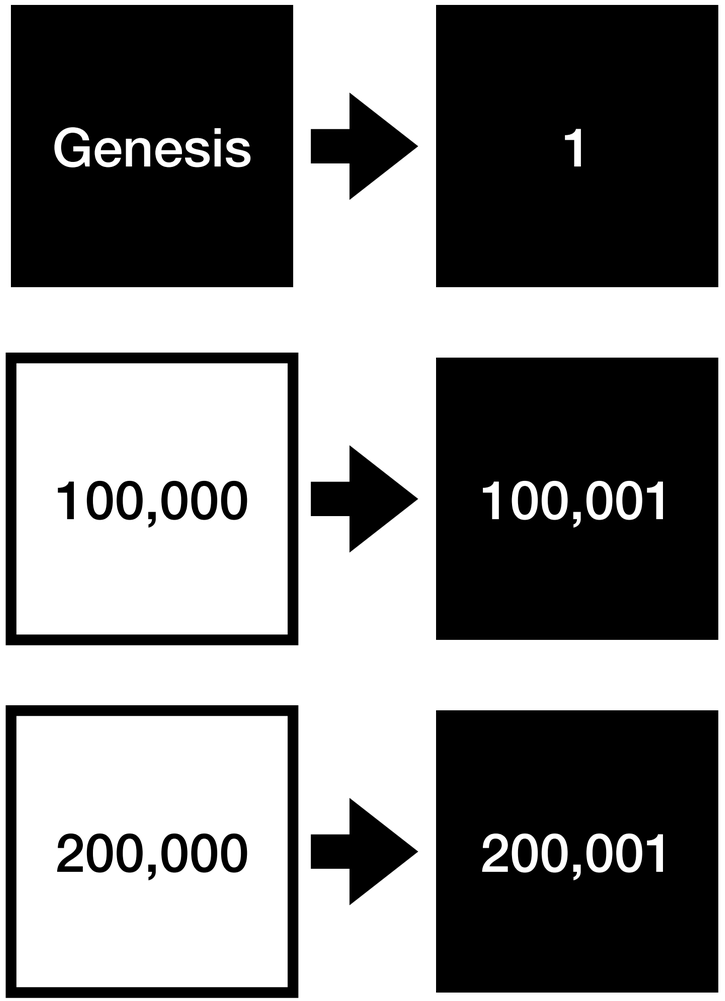
With Utreexo

Efficient parallel validation



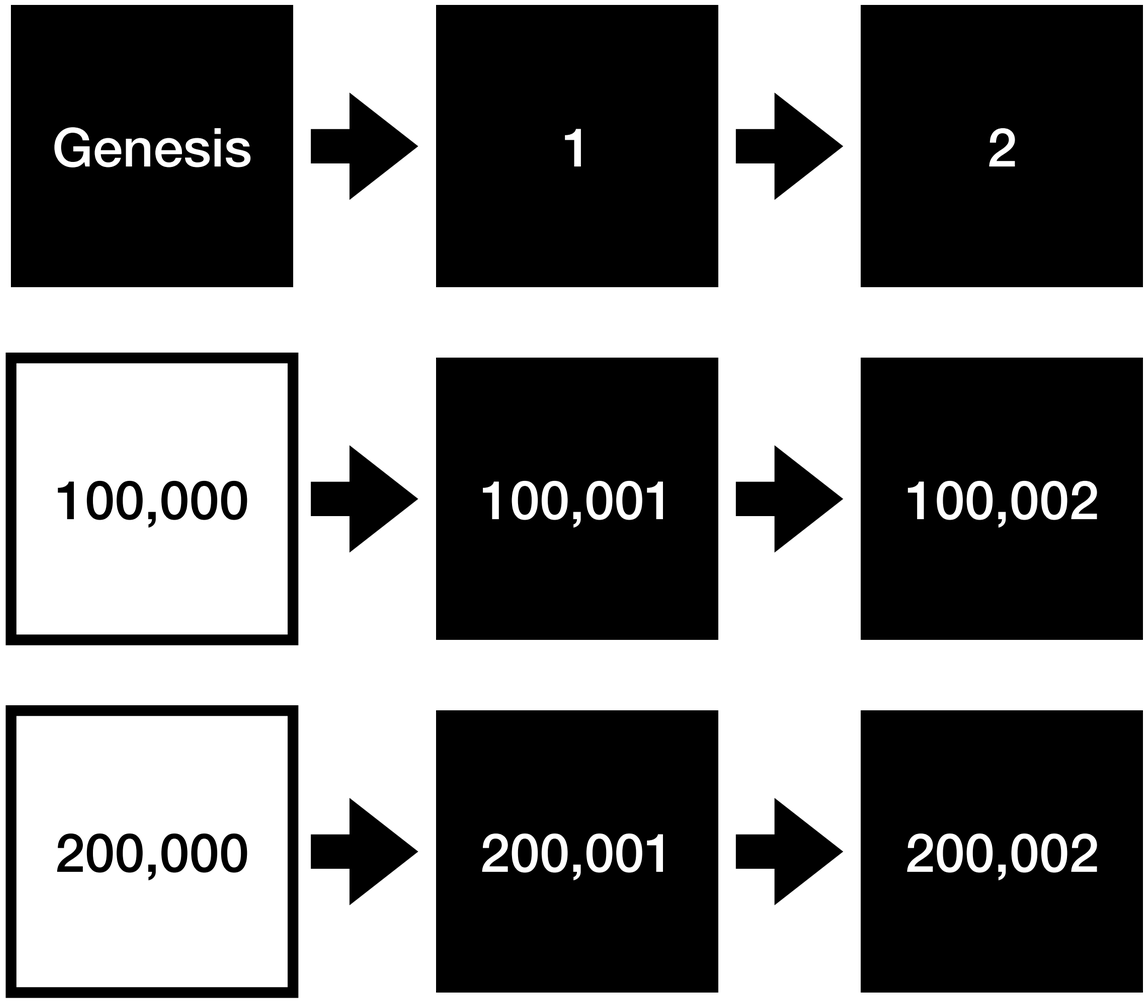
With Utreexo

Efficient parallel validation



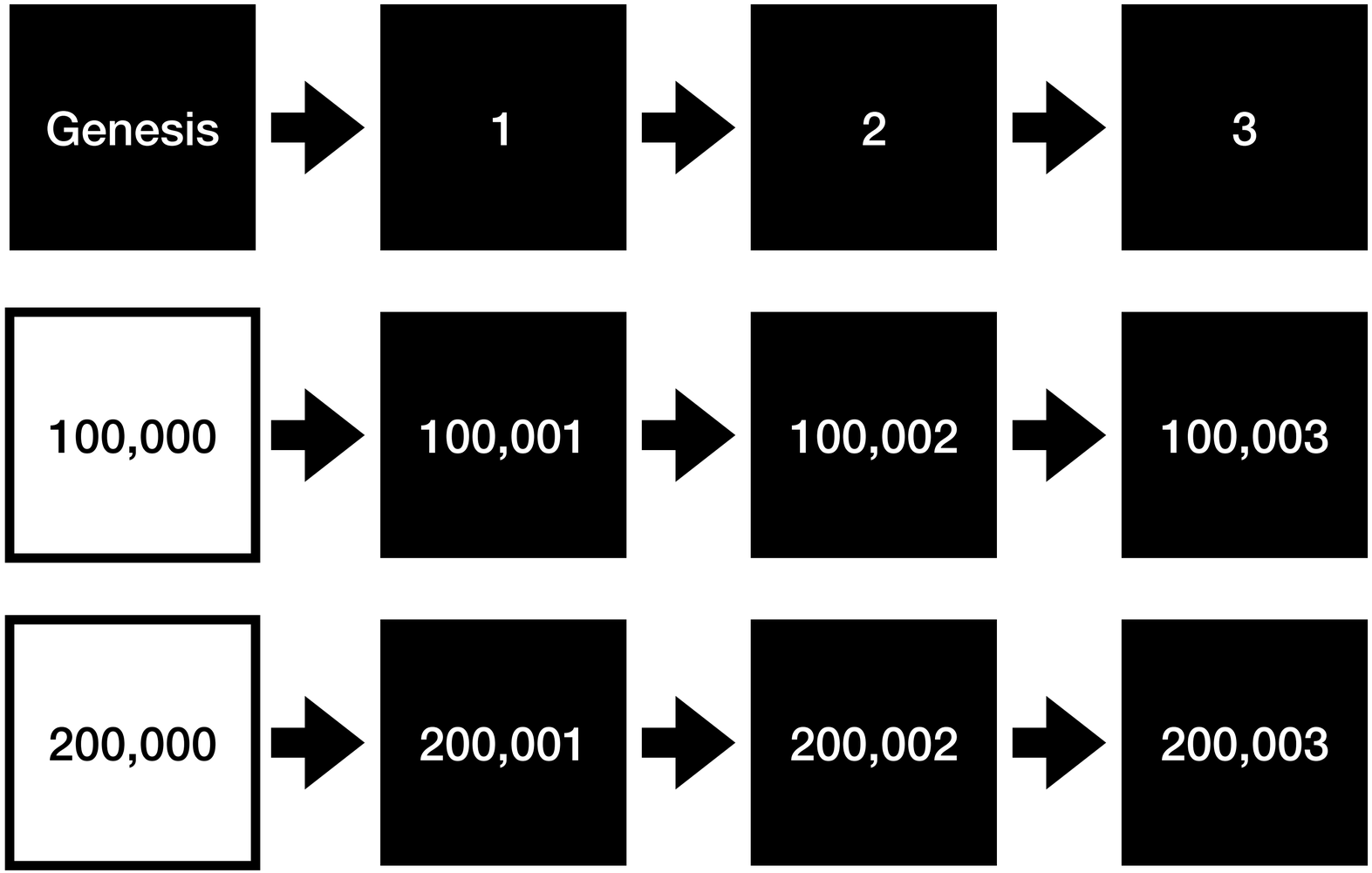
With Utreexo

Efficient parallel validation



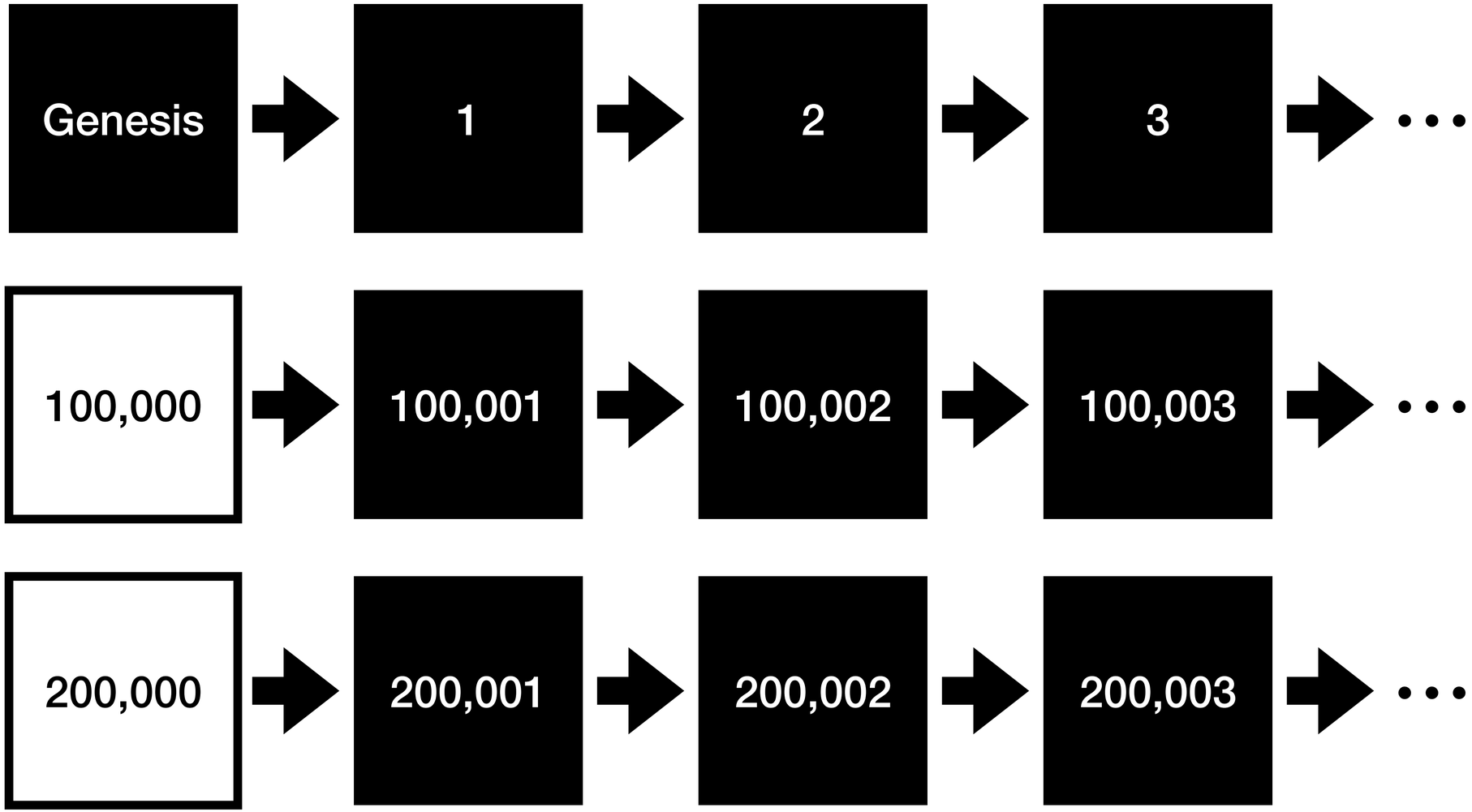
With Utreexo

Efficient parallel validation



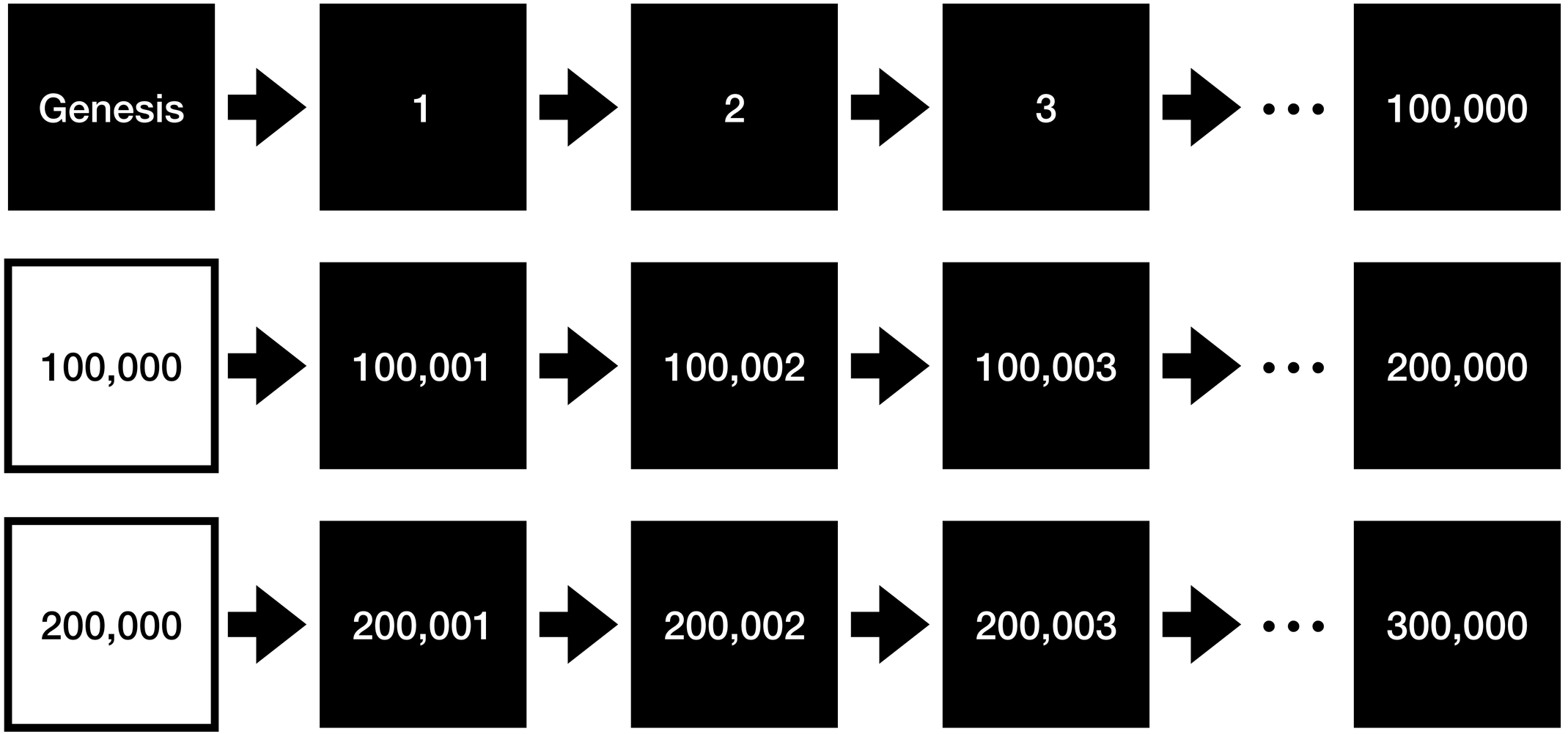
With Utreexo

Efficient parallel validation



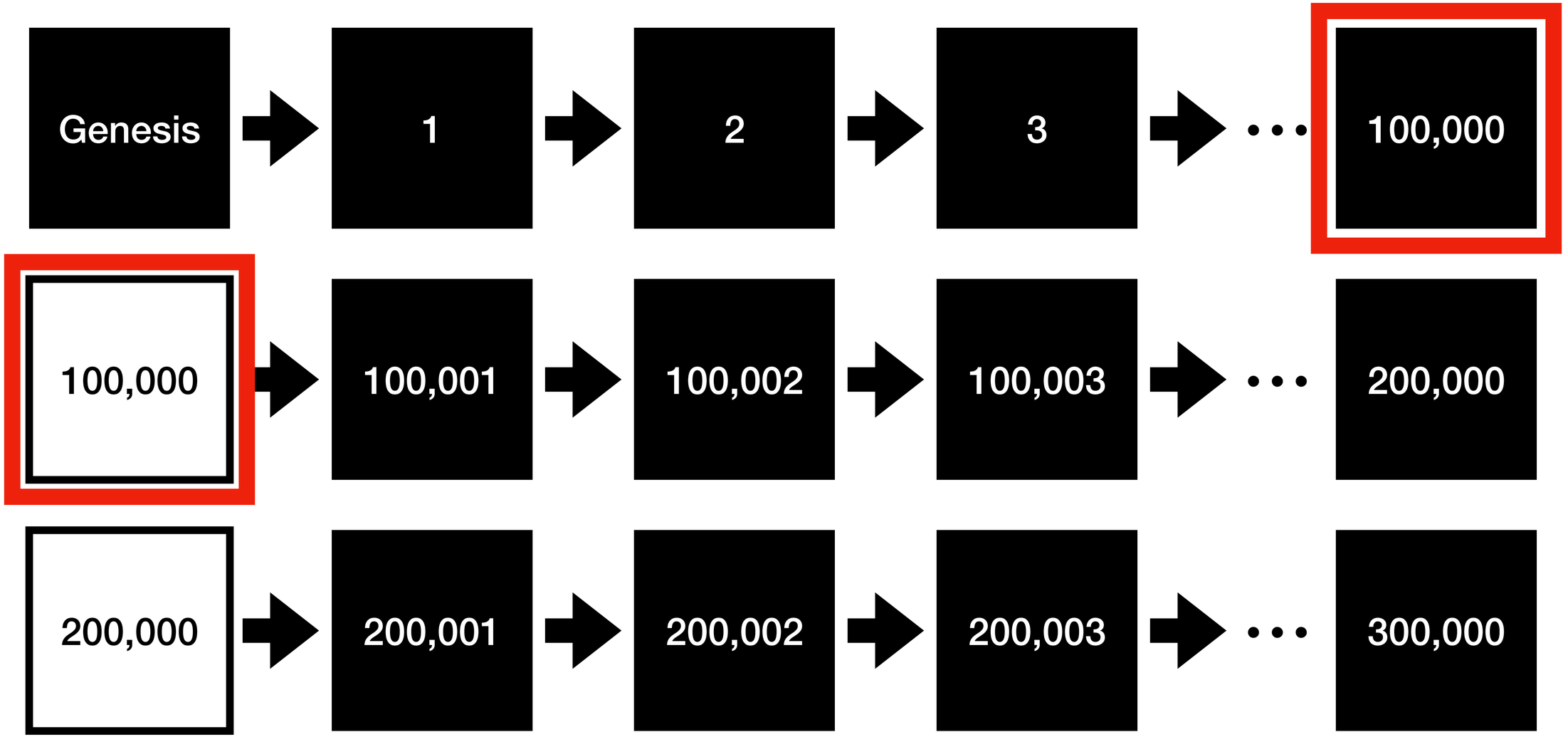
With Utreexo

Efficient parallel validation



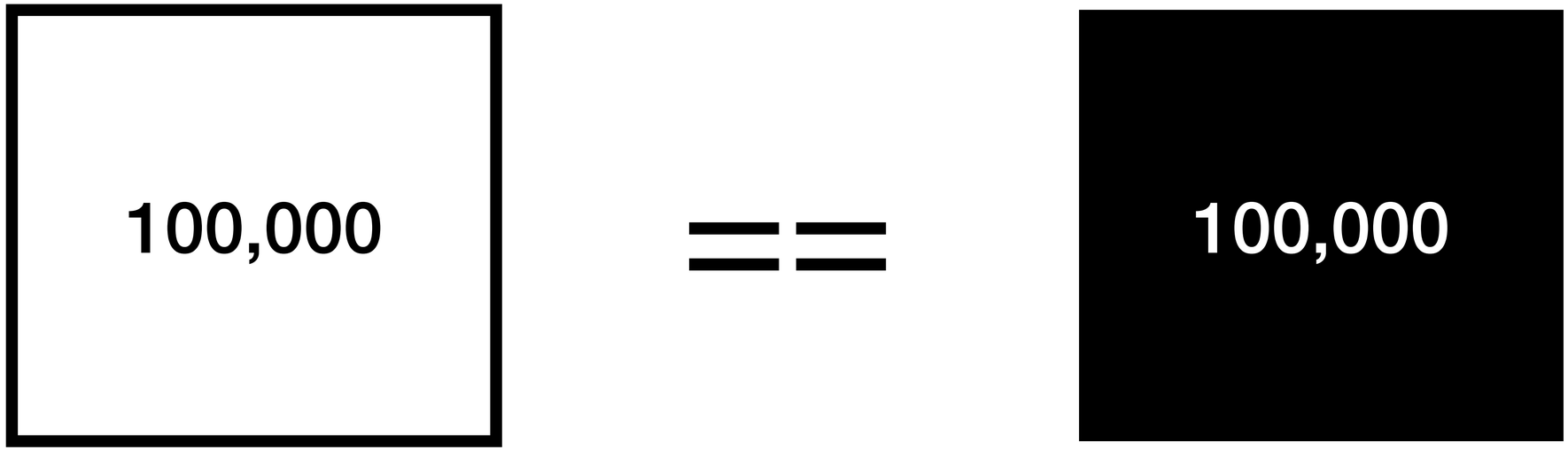
With Utreexo

Efficient parallel validation



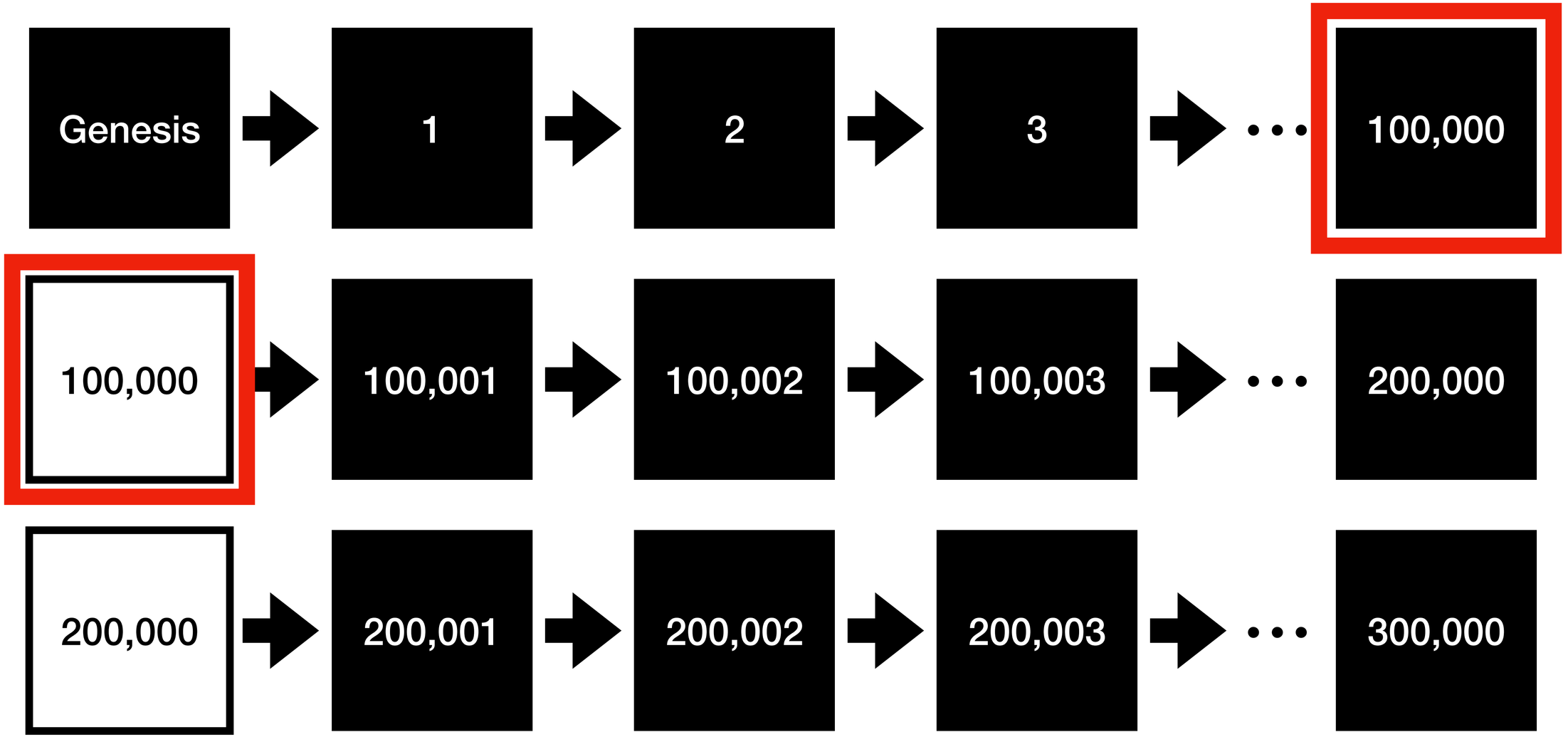
Compare roots

Root at 100,000 becomes trusted if equal



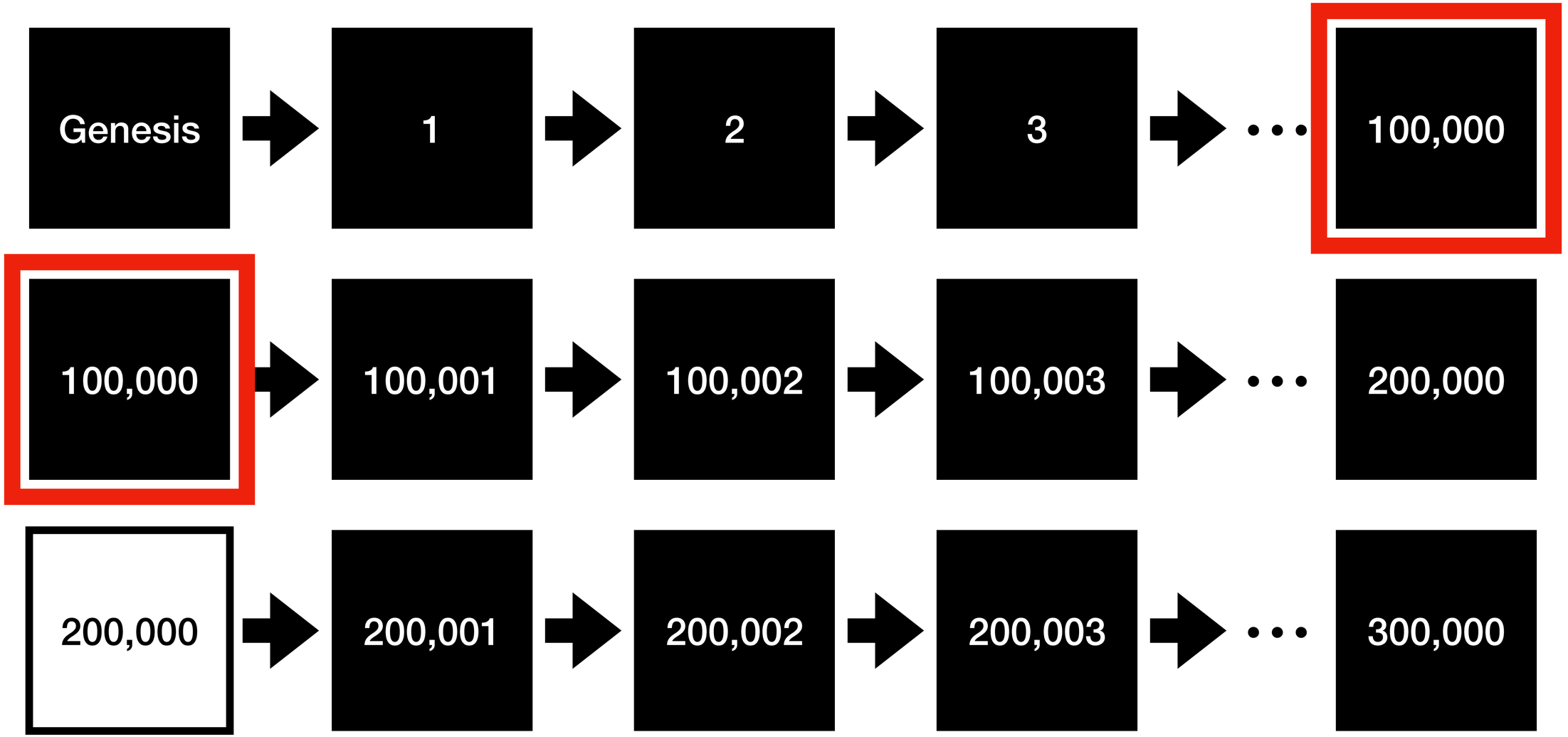
With Utreexo

Efficient parallel validation



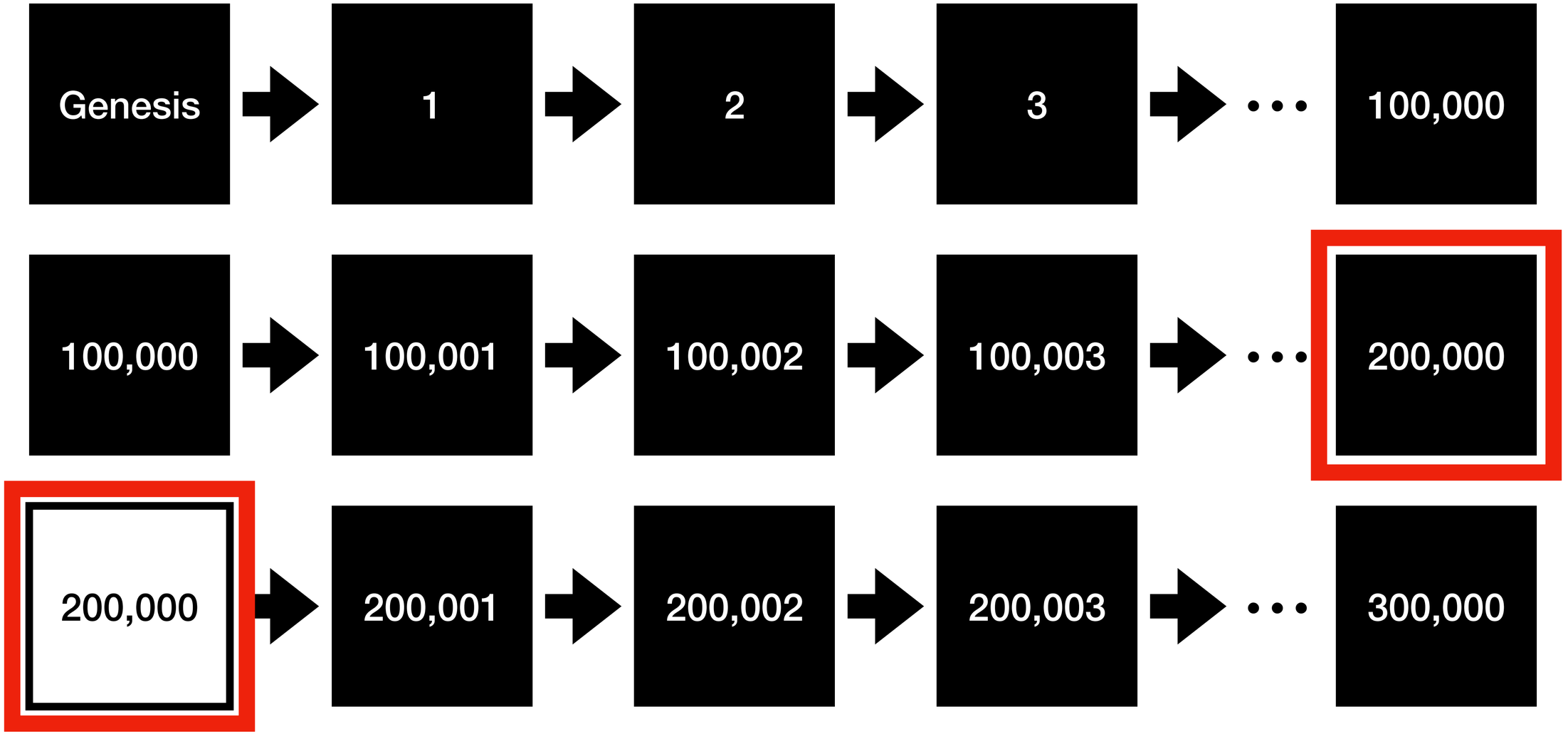
With Utreexo

Efficient parallel validation



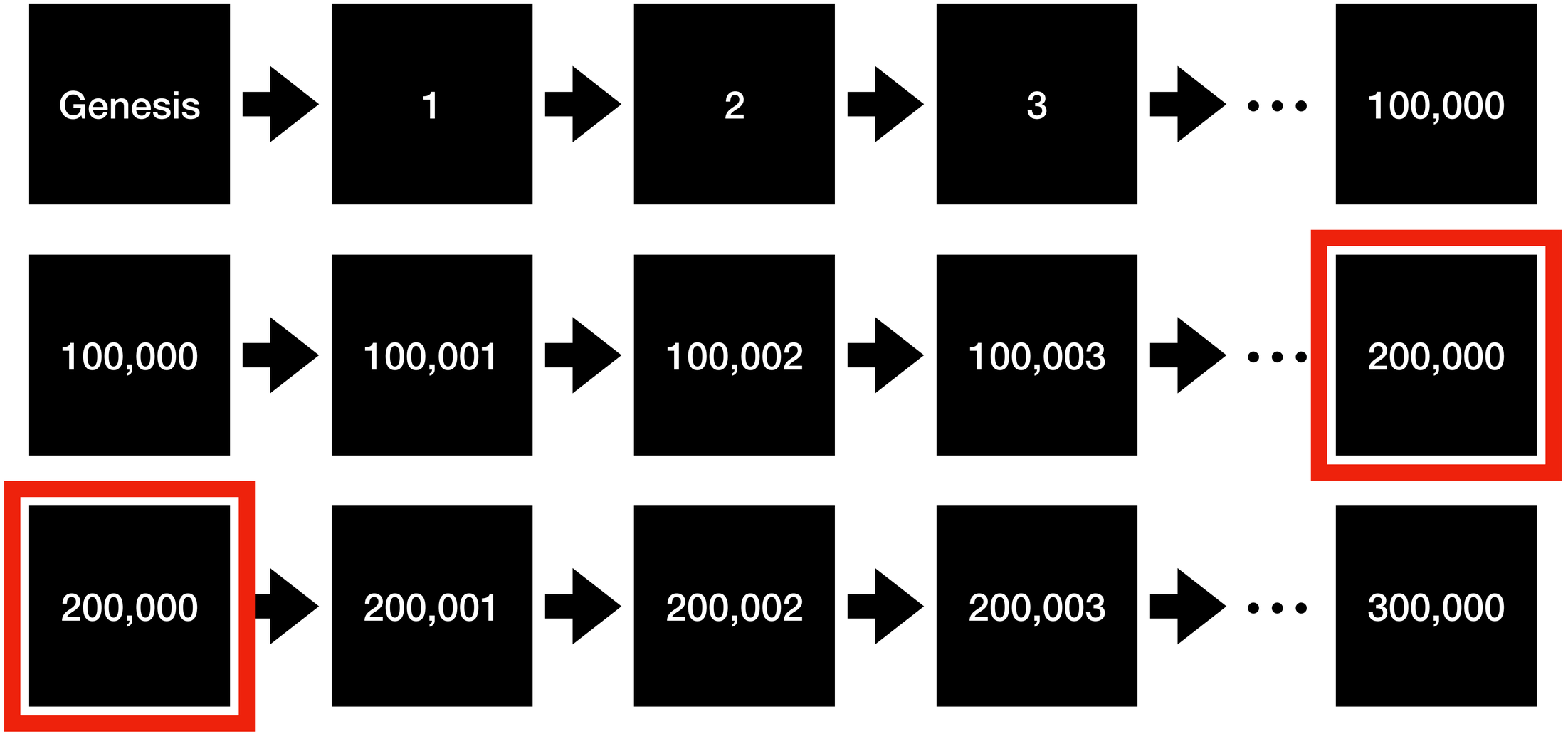
With Utreexo

Efficient parallel validation



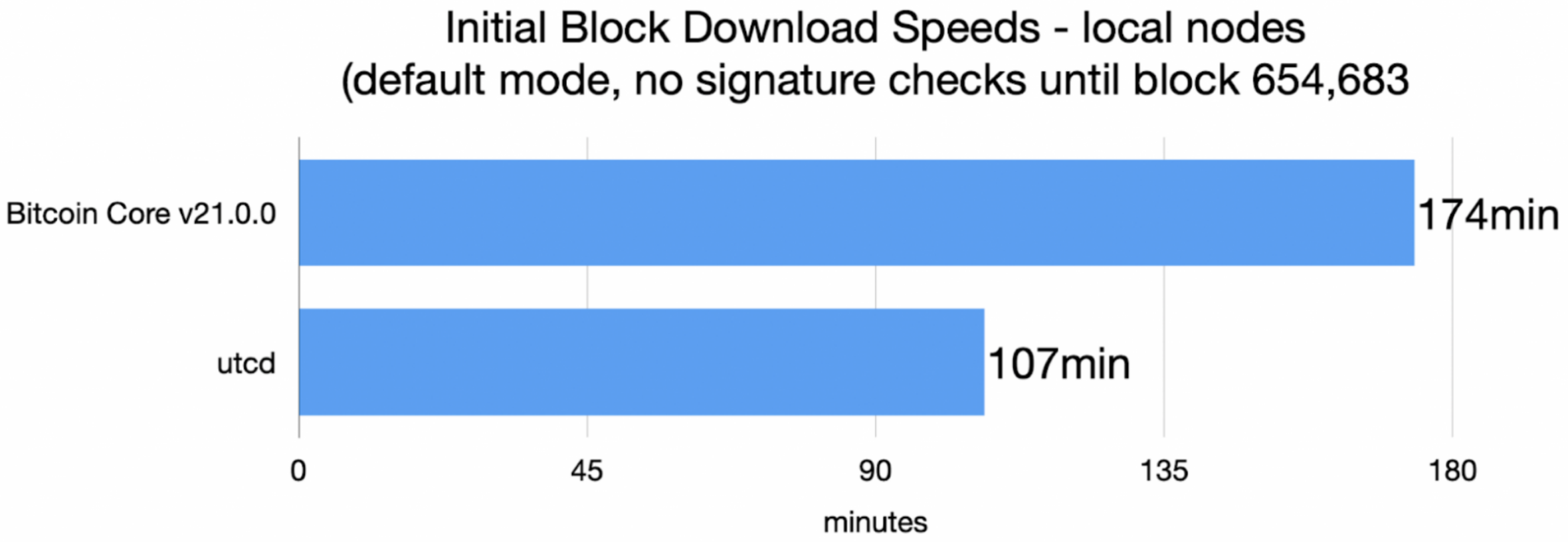
With Utreexo

Efficient parallel validation

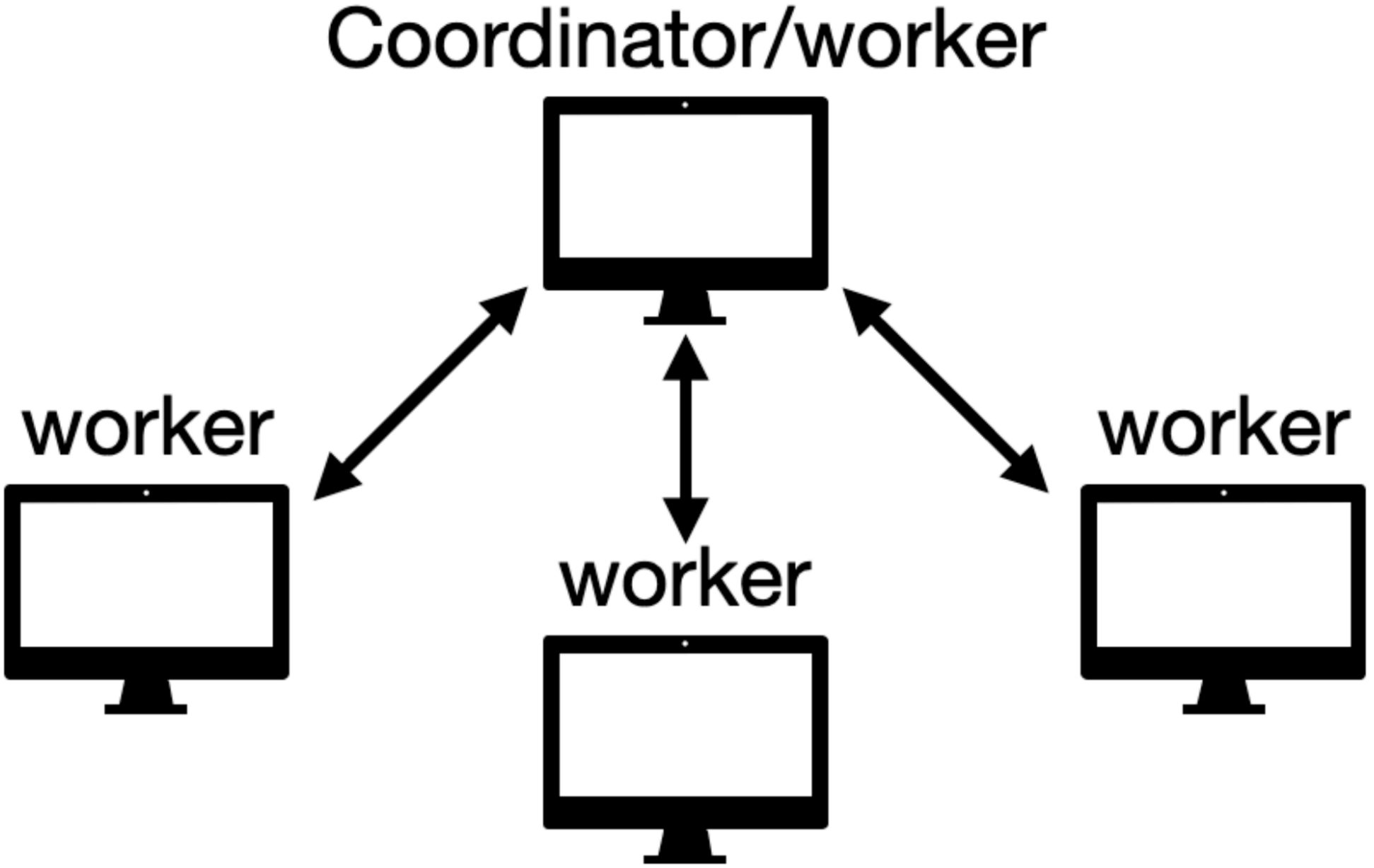


Efficient Parallel Validation

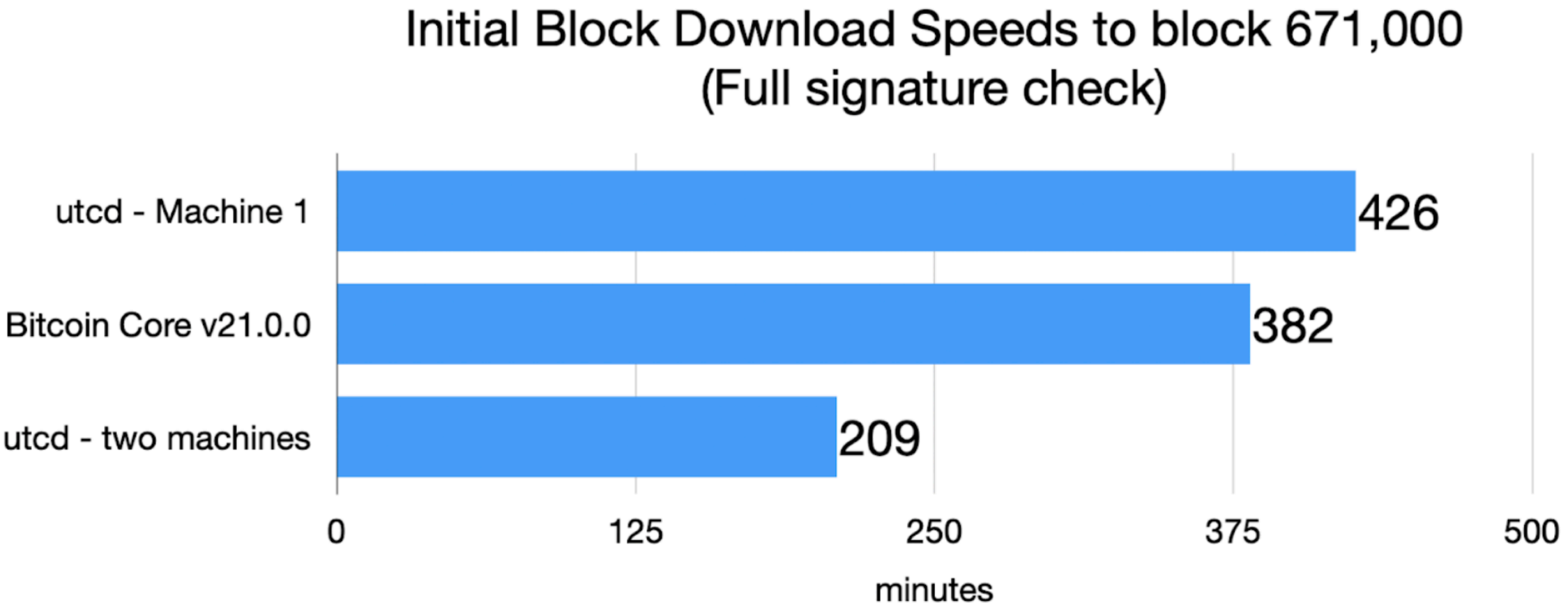
SHA256  > Map access 



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



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



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

The UTXO set

Why merklize it?

- Puts a bound to the UTXO set growth
- Allows for tiny nodes
- Faster block validation
- Define consensus without leveldb

The Block validation

Basic steps

1. Check Header (PoW)

The Block validation

Basic steps

1. Check Header (PoW)
2. Fetch inputs from all TXs (LevelDB)

The Block validation

Basic steps

1. Check Header (PoW)
2. Fetch inputs from all TXs (LevelDB)
3. Verify signatures

The Block validation

Basic steps

1. Check Header (PoW)

The Block validation

Basic steps

1. Check Header (PoW)
2. Verify Utreexo proof

The Block validation

Basic steps

1. Check Header (PoW)
2. Verify Utreexo proof
3. Verify signatures

How does it work?

Replace levelDB

What does levelDB do for bitcoin?

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

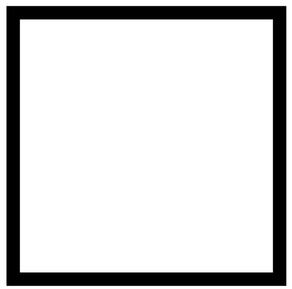
Role of levelDB

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

1 UTXO

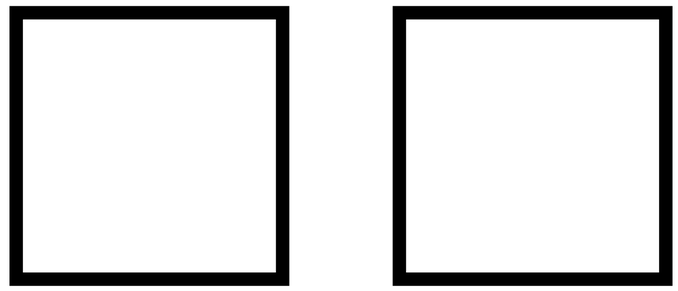
Single root



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

2 UTXOs

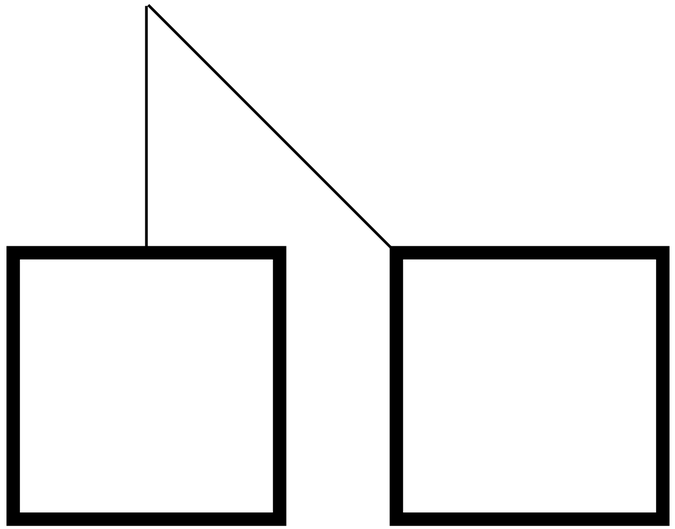
Add 1 more



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

2 UTXOs

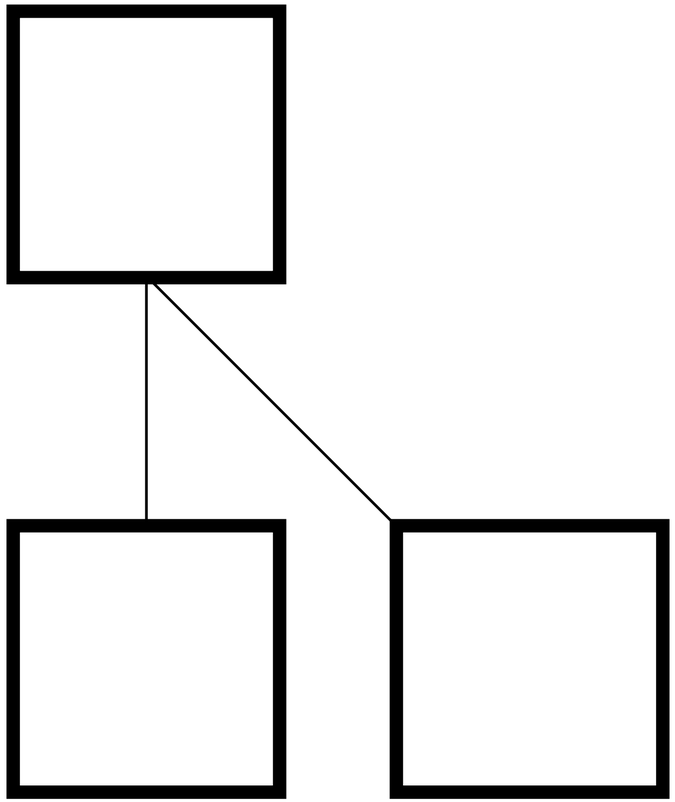
Concatenate



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

2 UTXOs

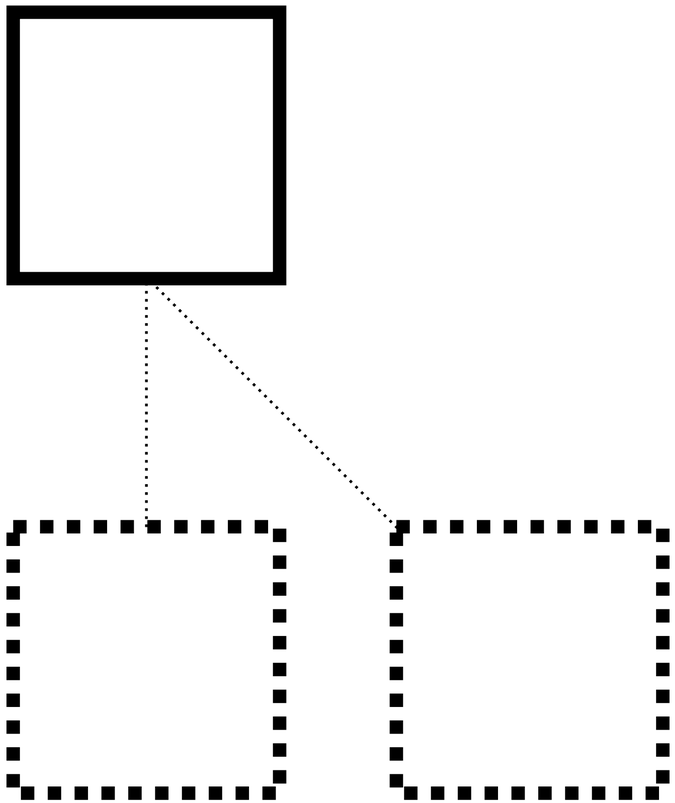
Hash to create a new root



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

2 UTXOs

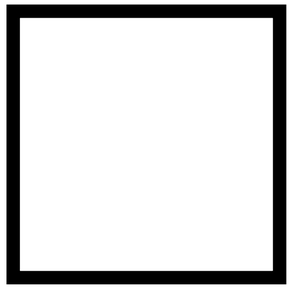
Can now delete the leaves



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

2 UTXOs

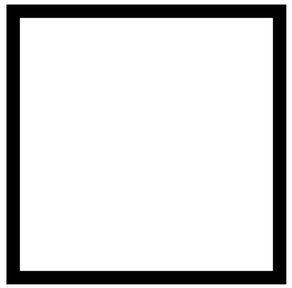
Only keep the roots



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

3 UTXOs

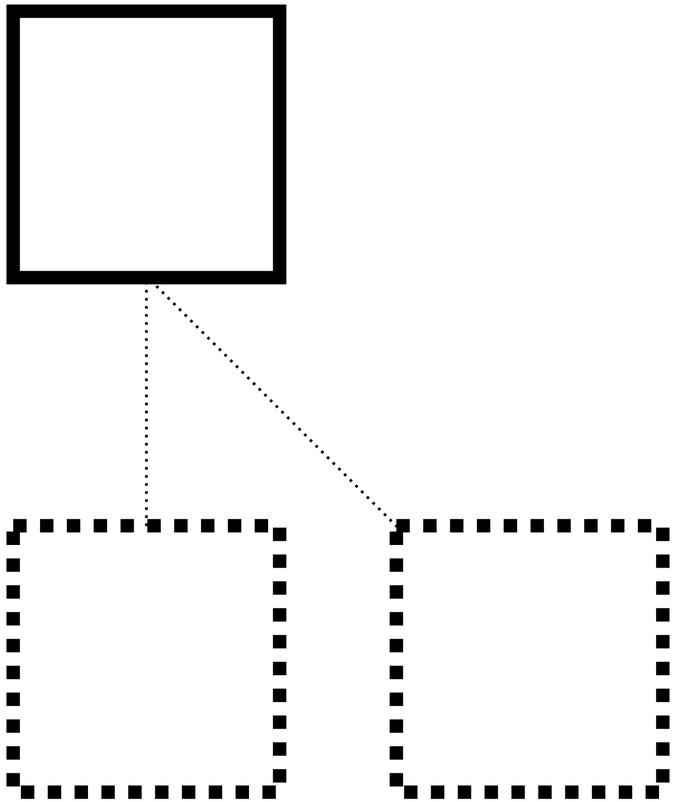
Add 1 more



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

3 UTXOs

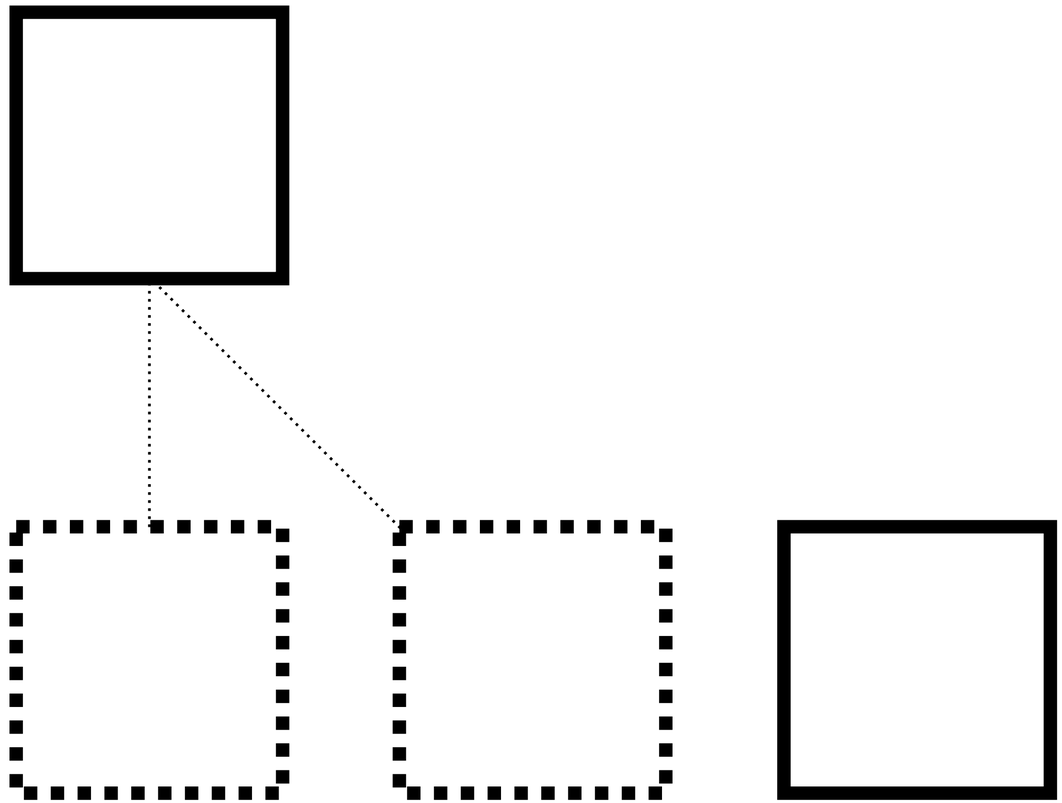
Add 1 more



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

3 UTXOs

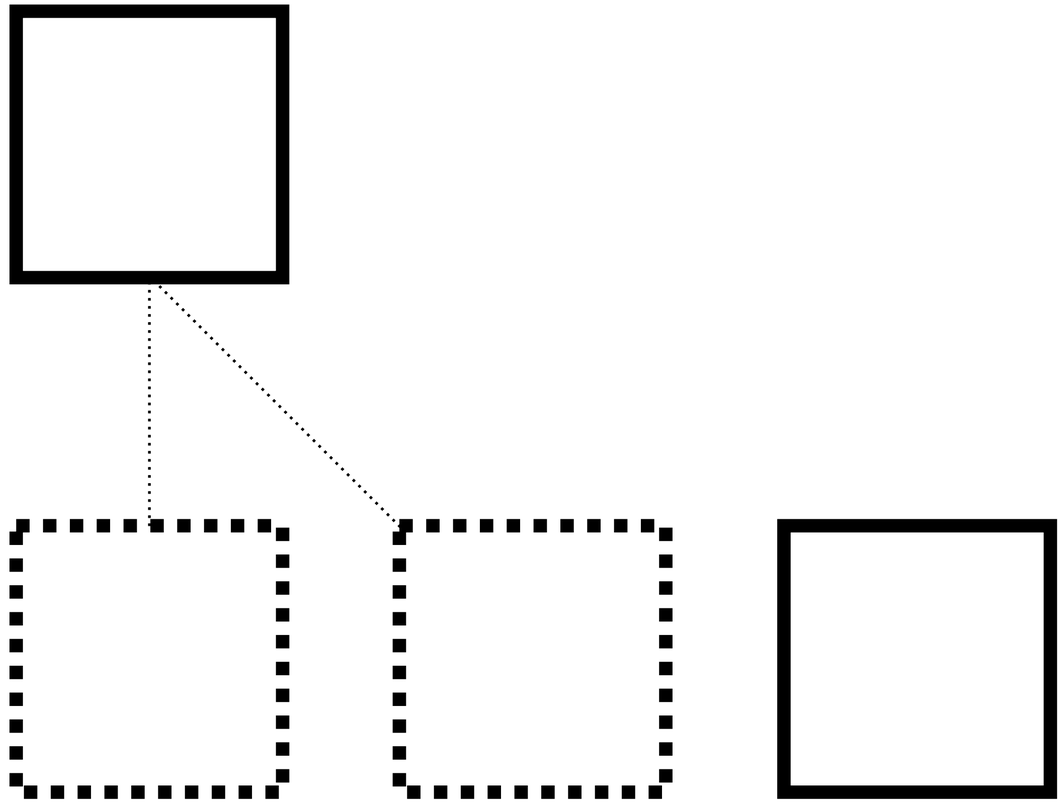
Add 1 more



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

3 UTXOs

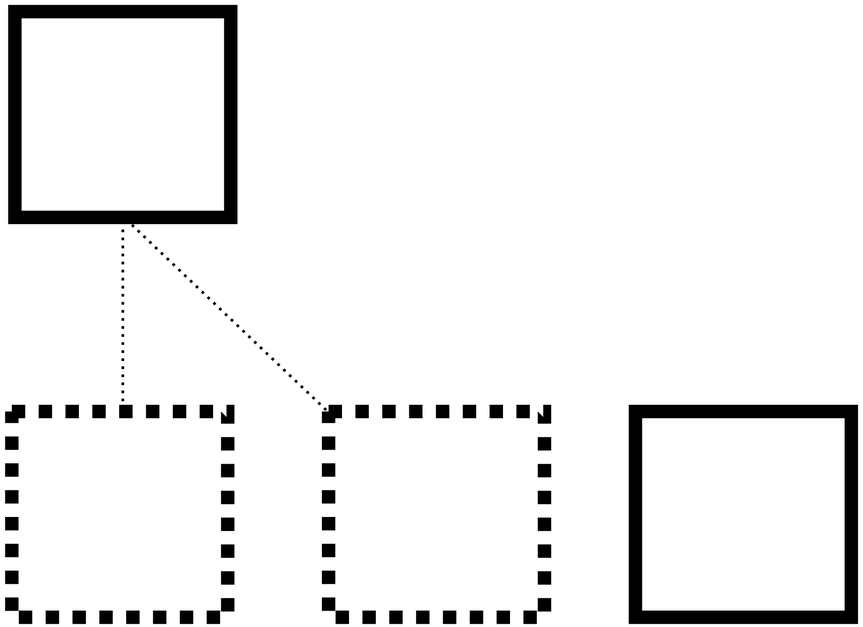
Nothing to hash with. Becomes a root



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

4 UTXOs

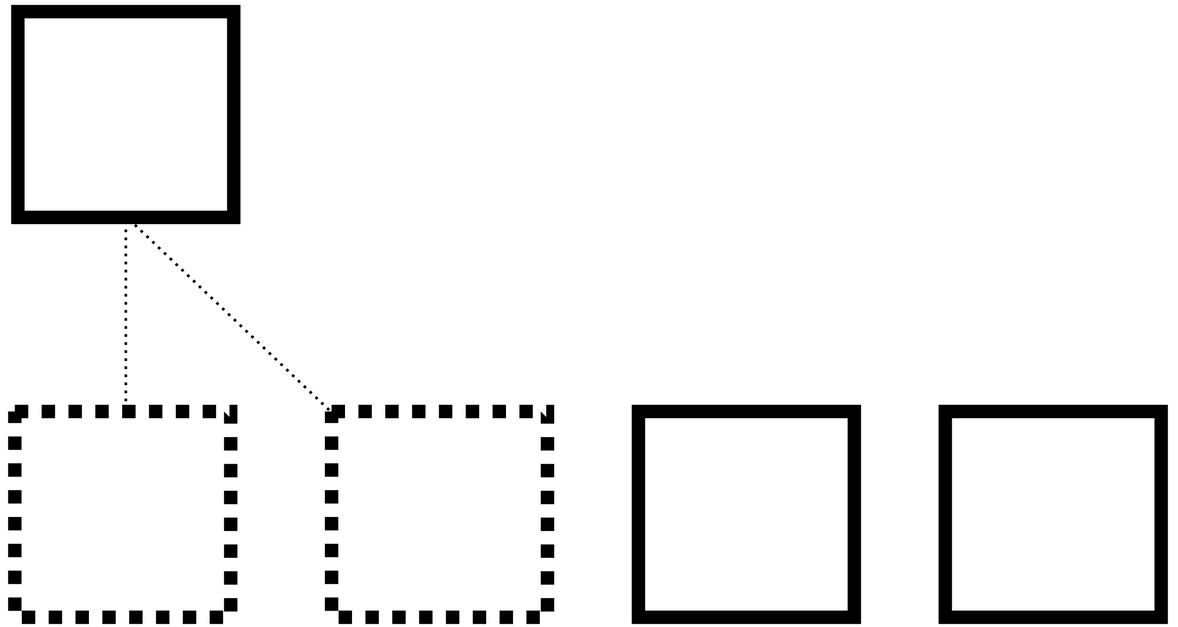
Another one



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

4 UTXOs

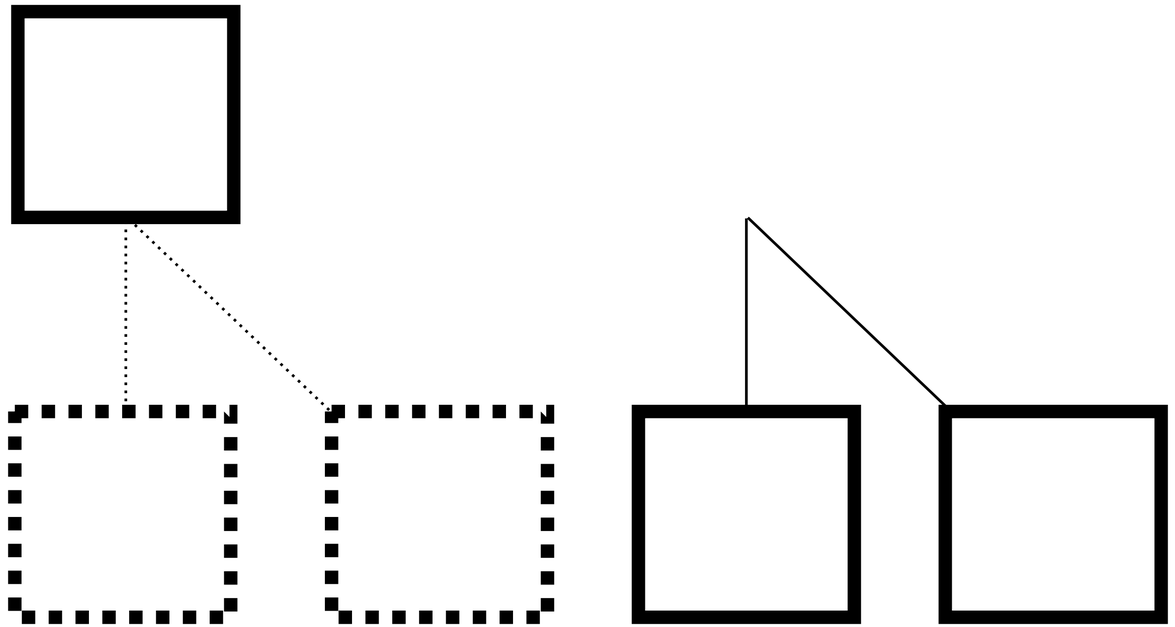
Another one



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

4 UTXOs

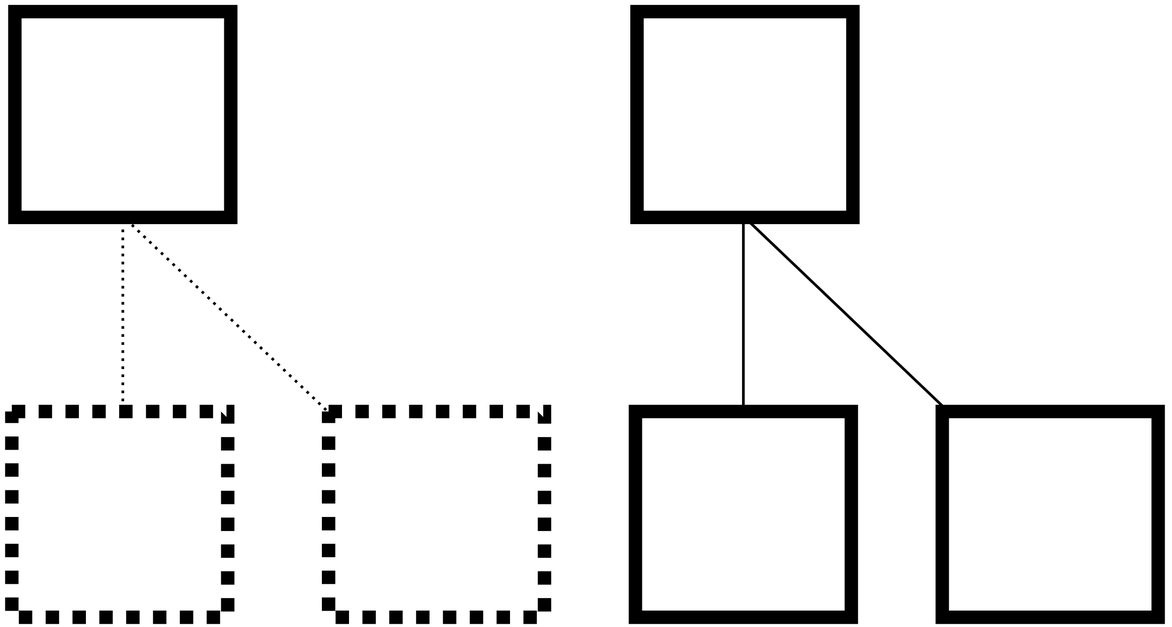
Concatenate



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

4 UTXOs

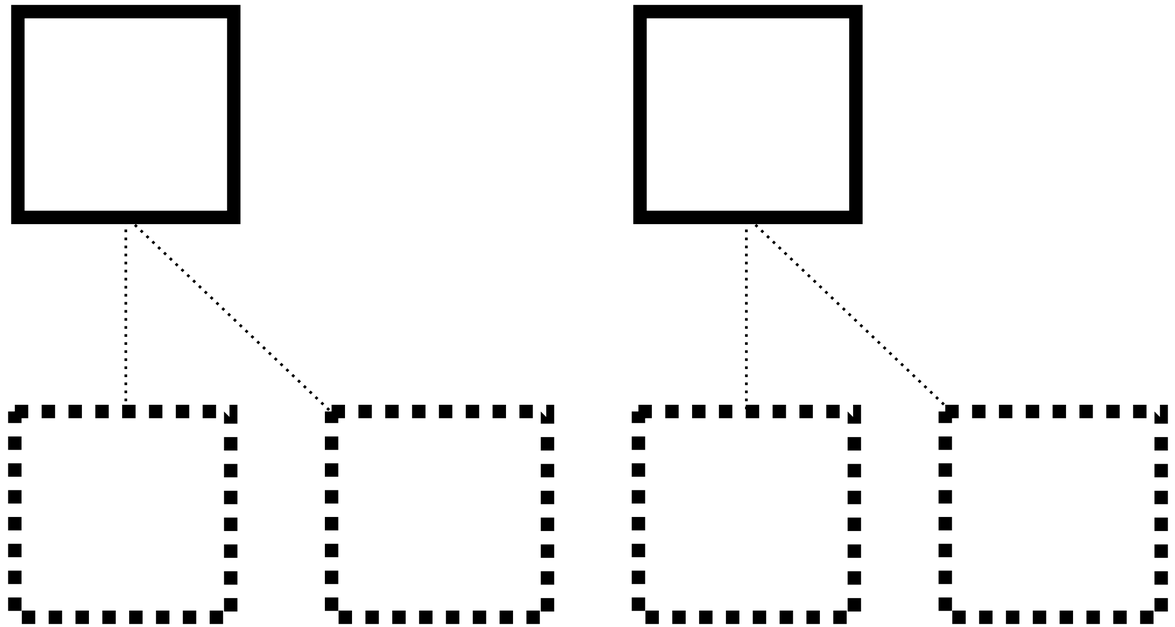
Hash



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

4 UTXOs

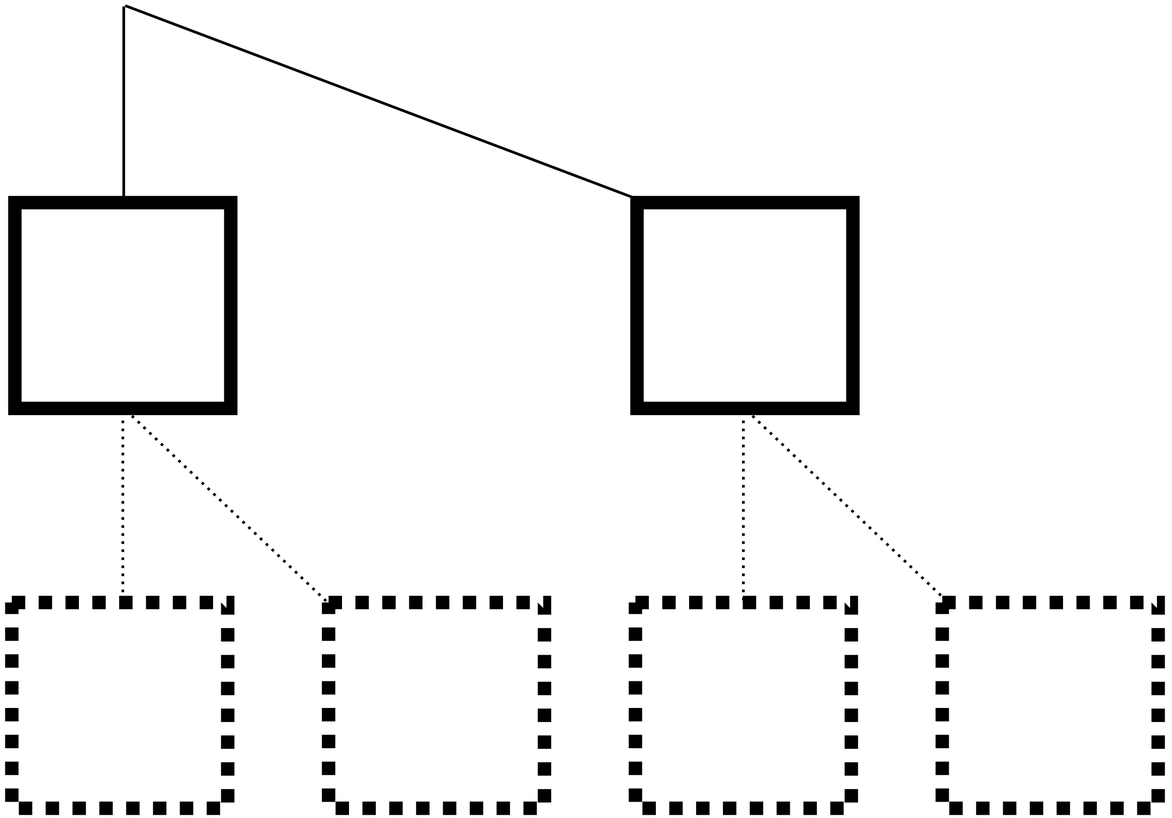
Can now throw away the leaves



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

4 UTXOs

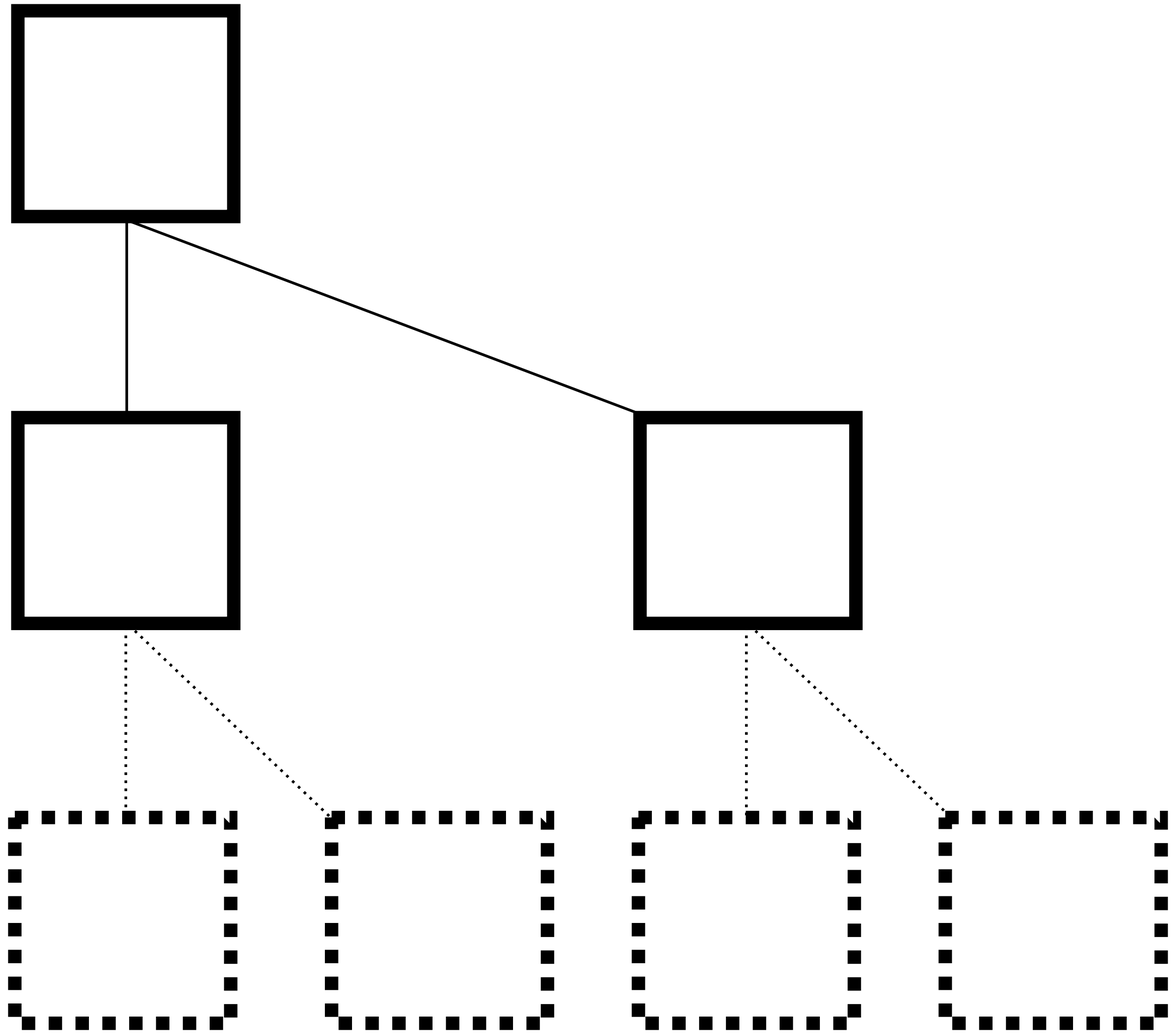
Concatenate



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

4 UTXOs

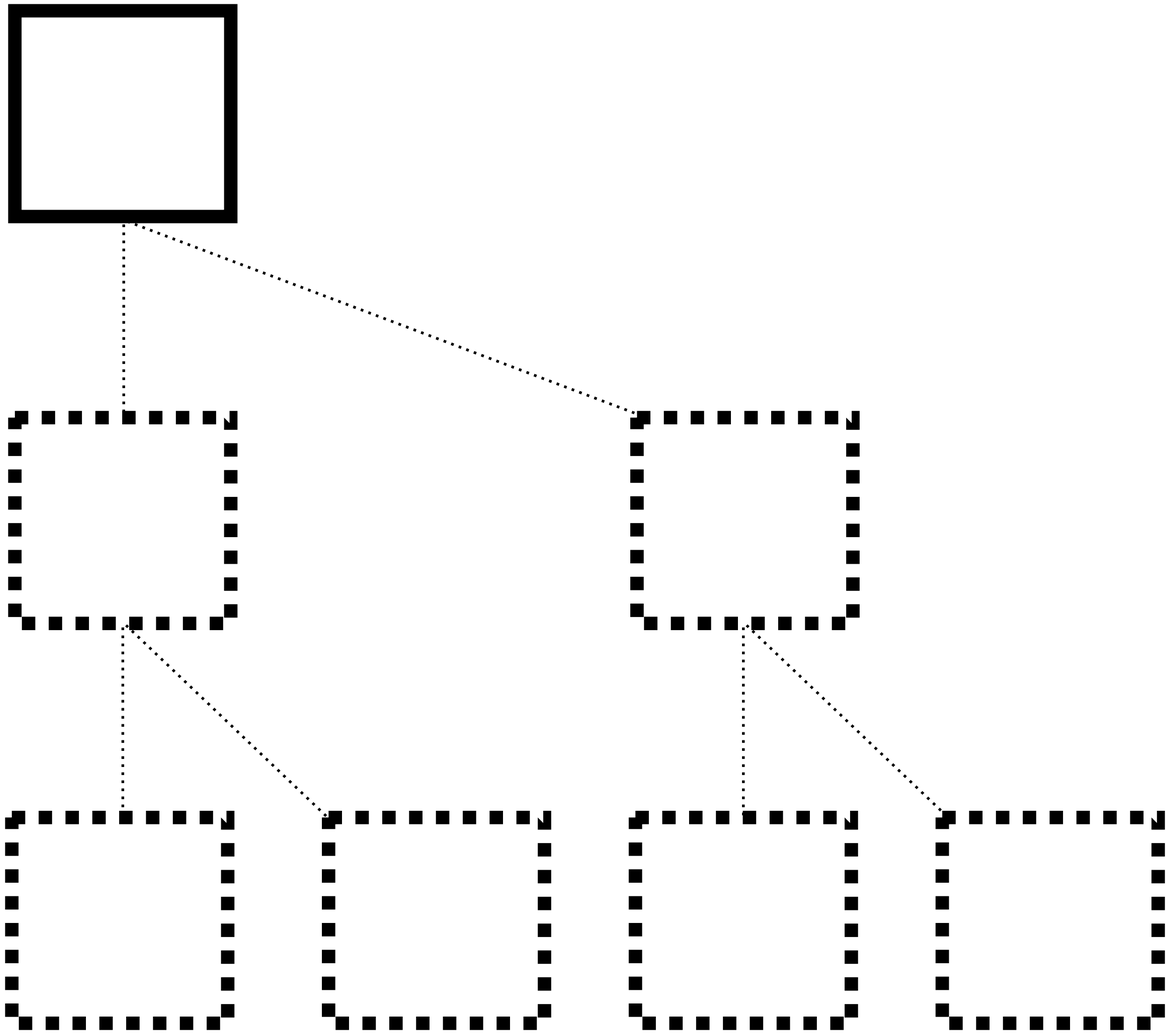
Hash



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

4 UTXOs

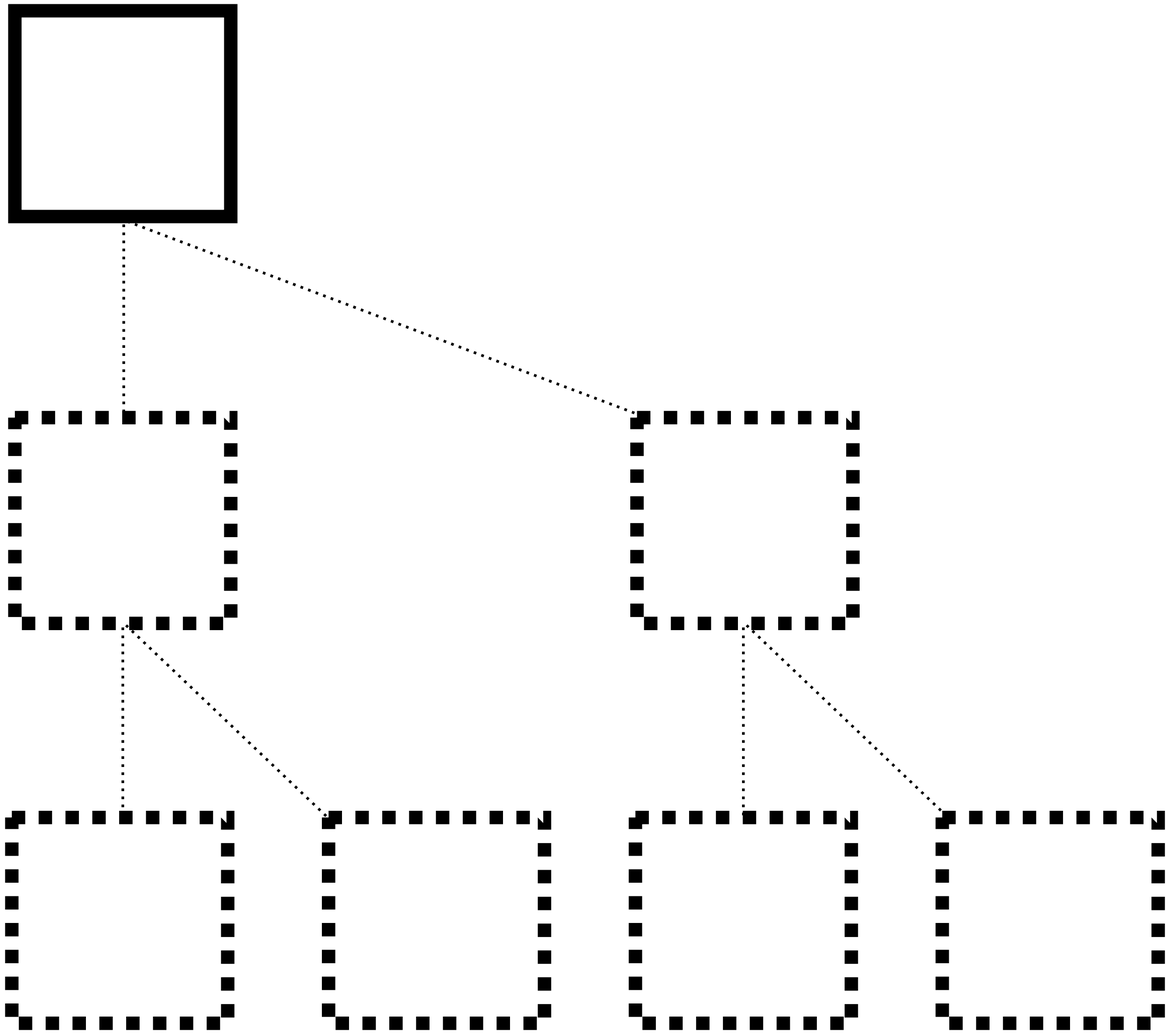
Throw away



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

4 UTXOs

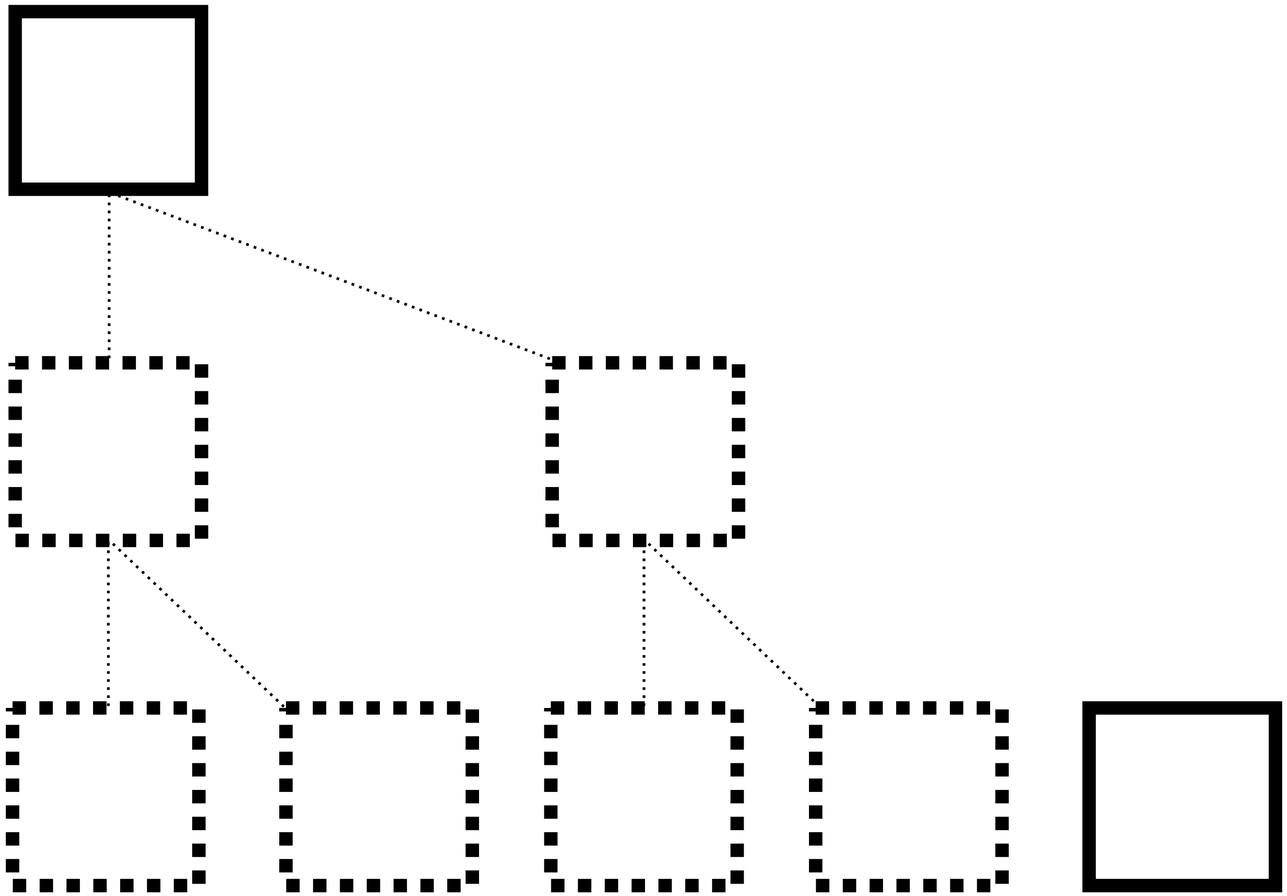
Finished tree



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

5 UTXOs

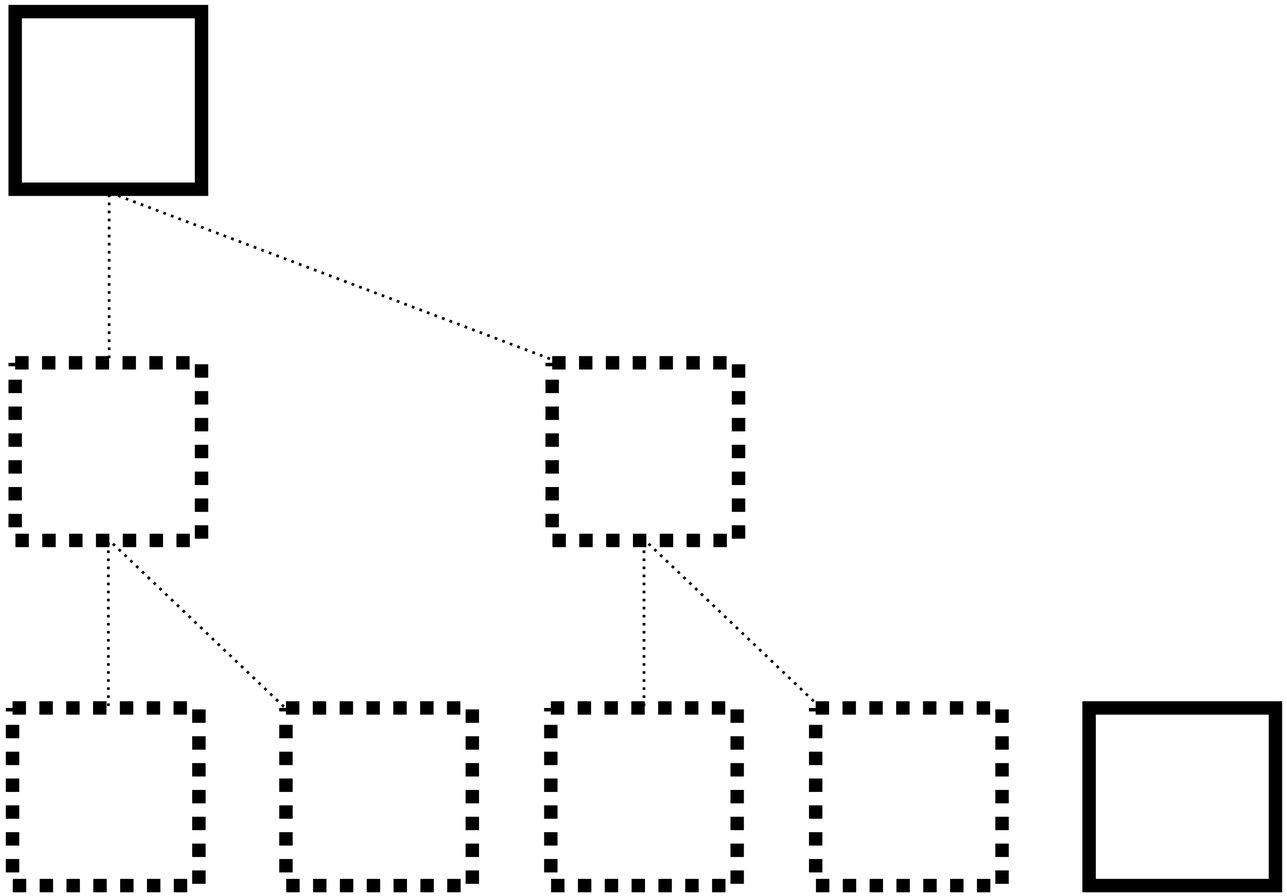
Add one more



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

5 UTXOs

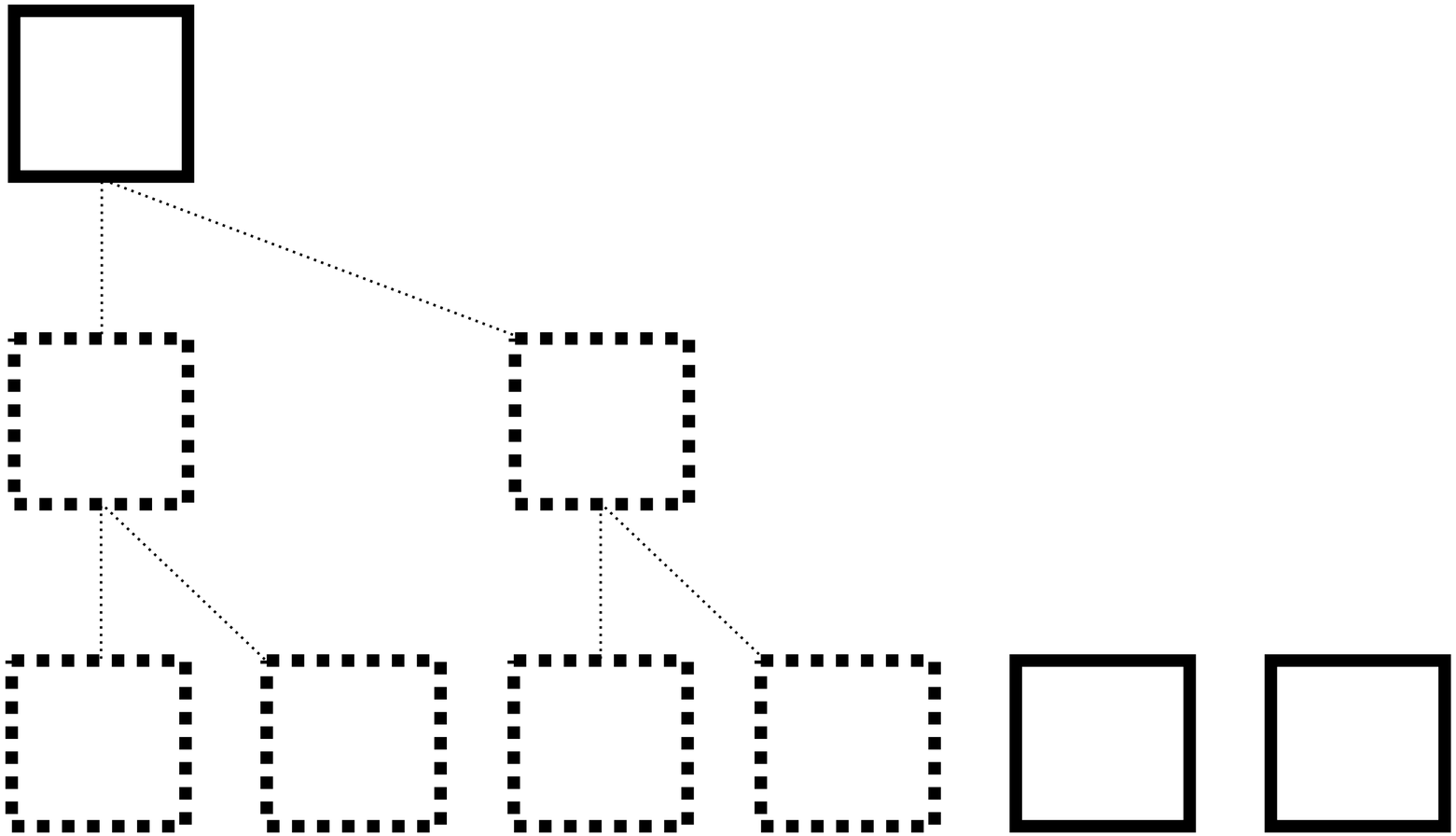
Nothing to hash with so it becomes a root



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

6 UTXOs

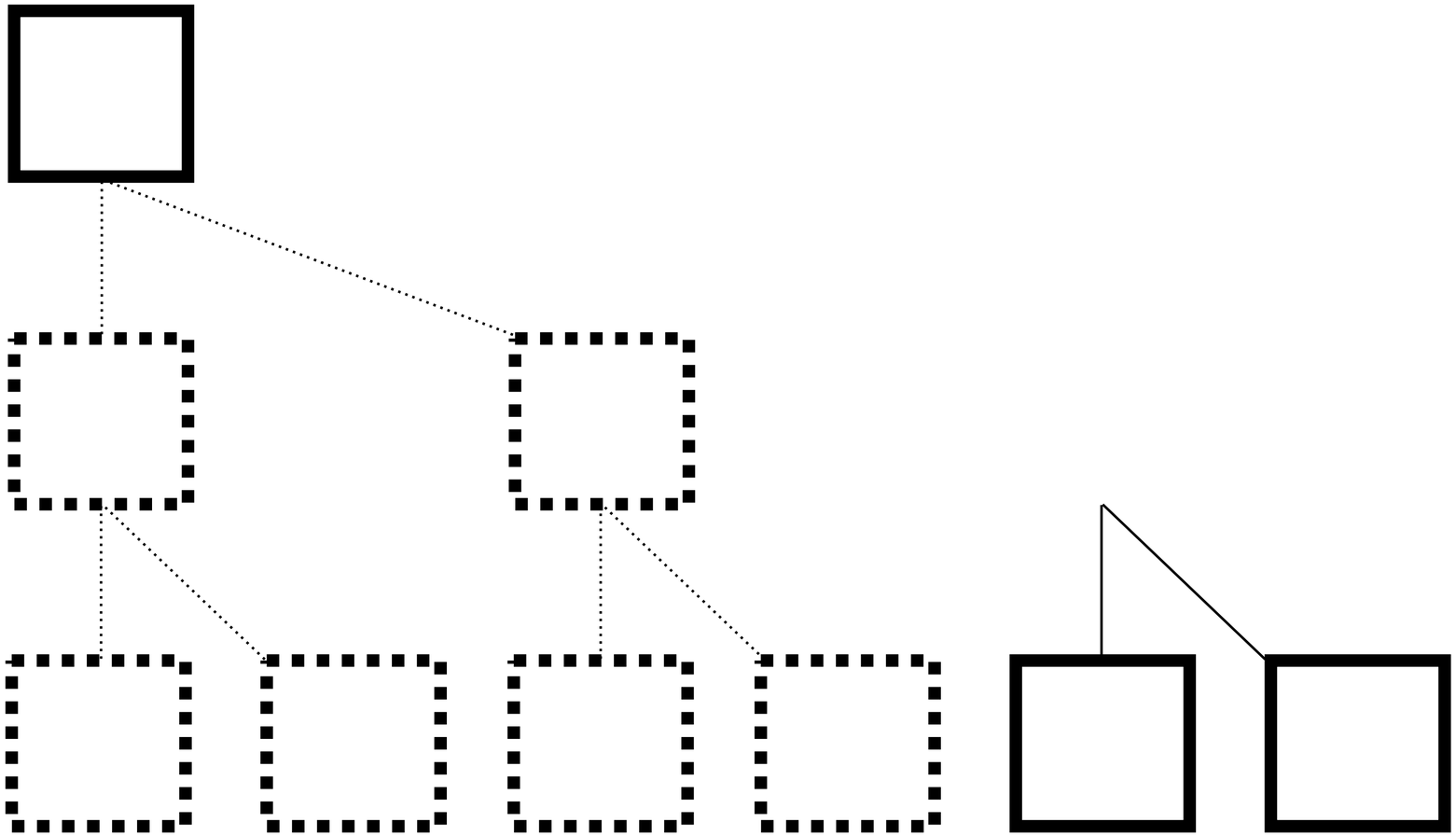
1 more



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

6 UTXOs

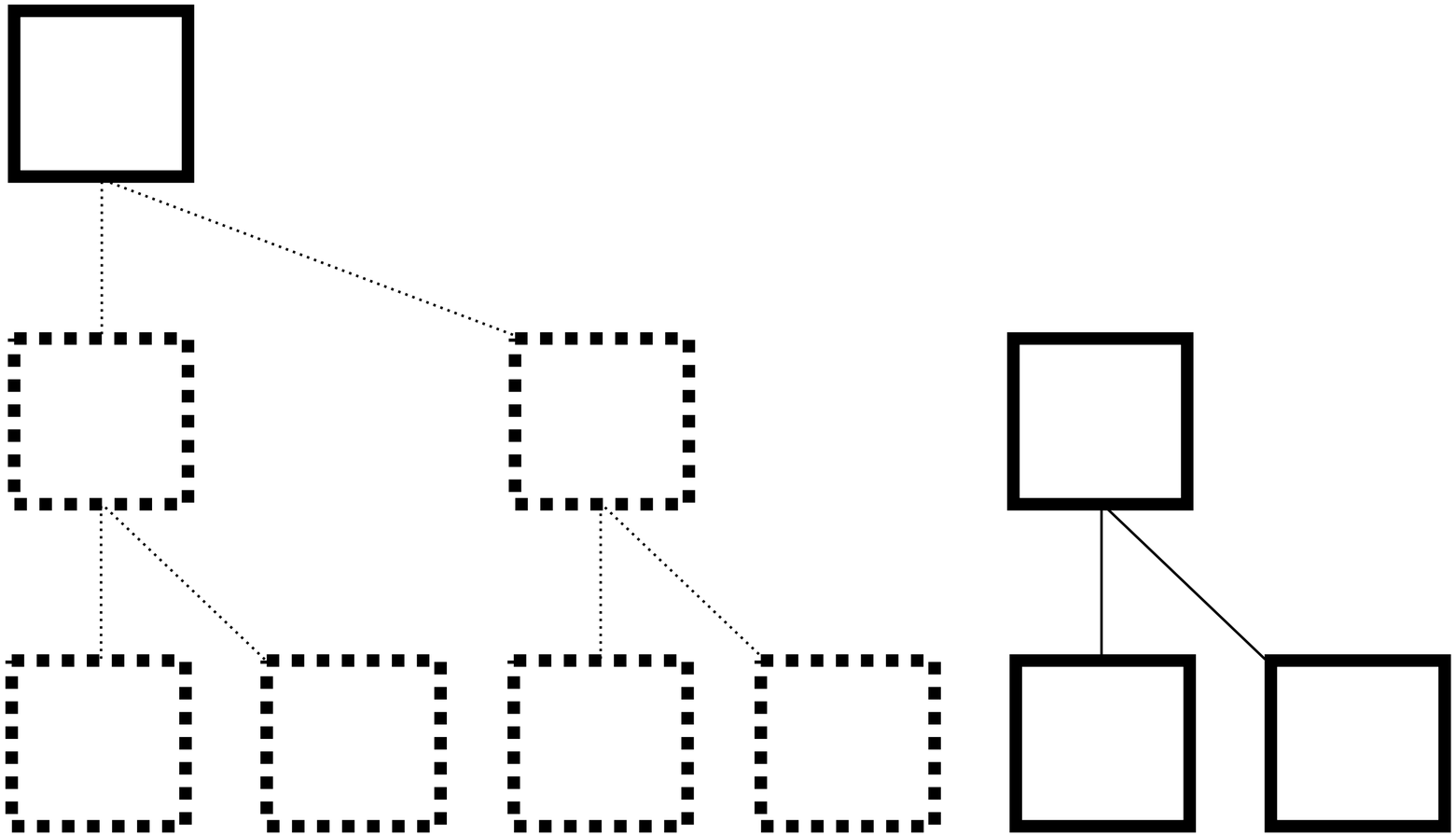
Concatenate



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

6 UTXOs

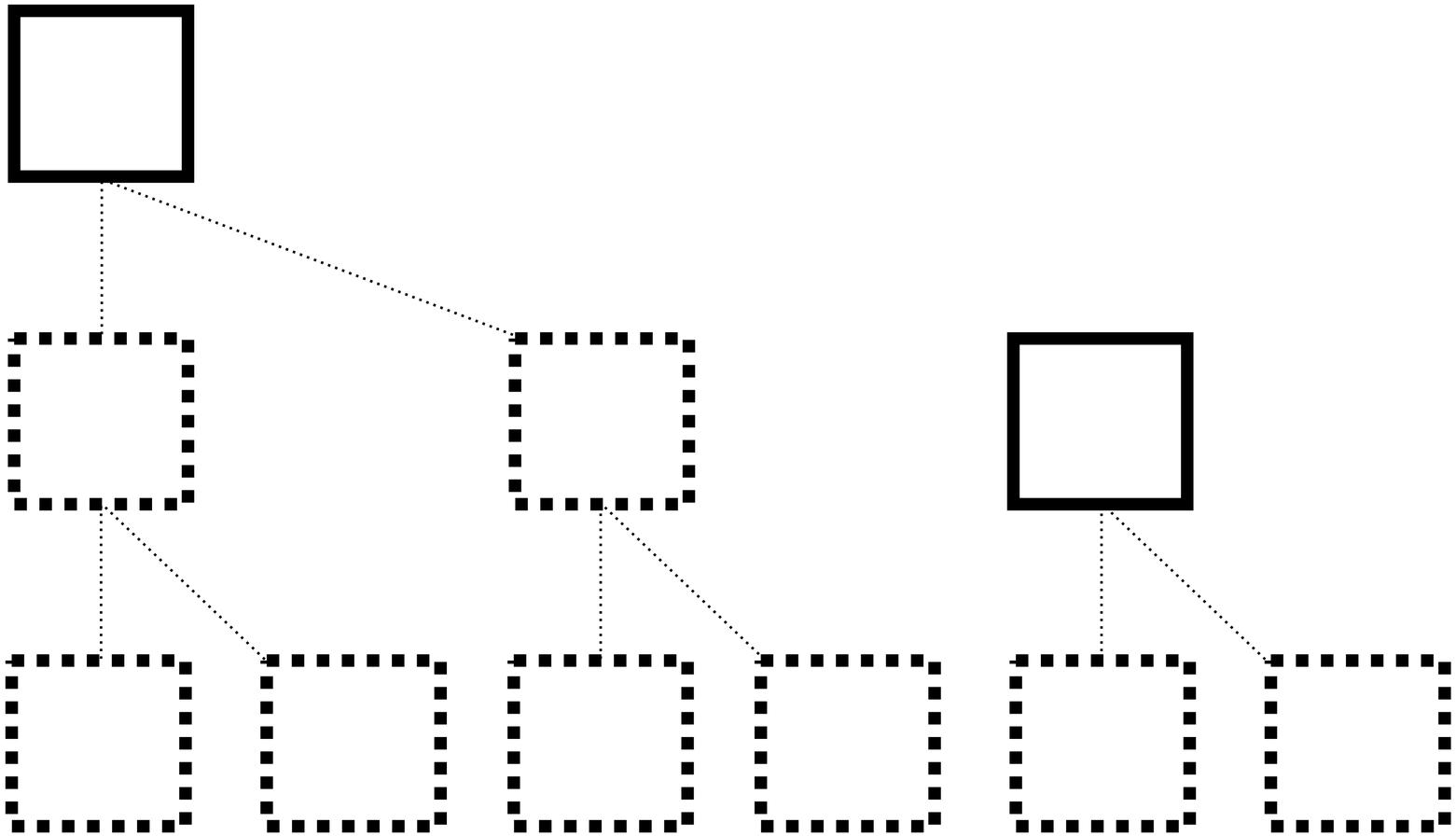
Hash



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

6 UTXOs

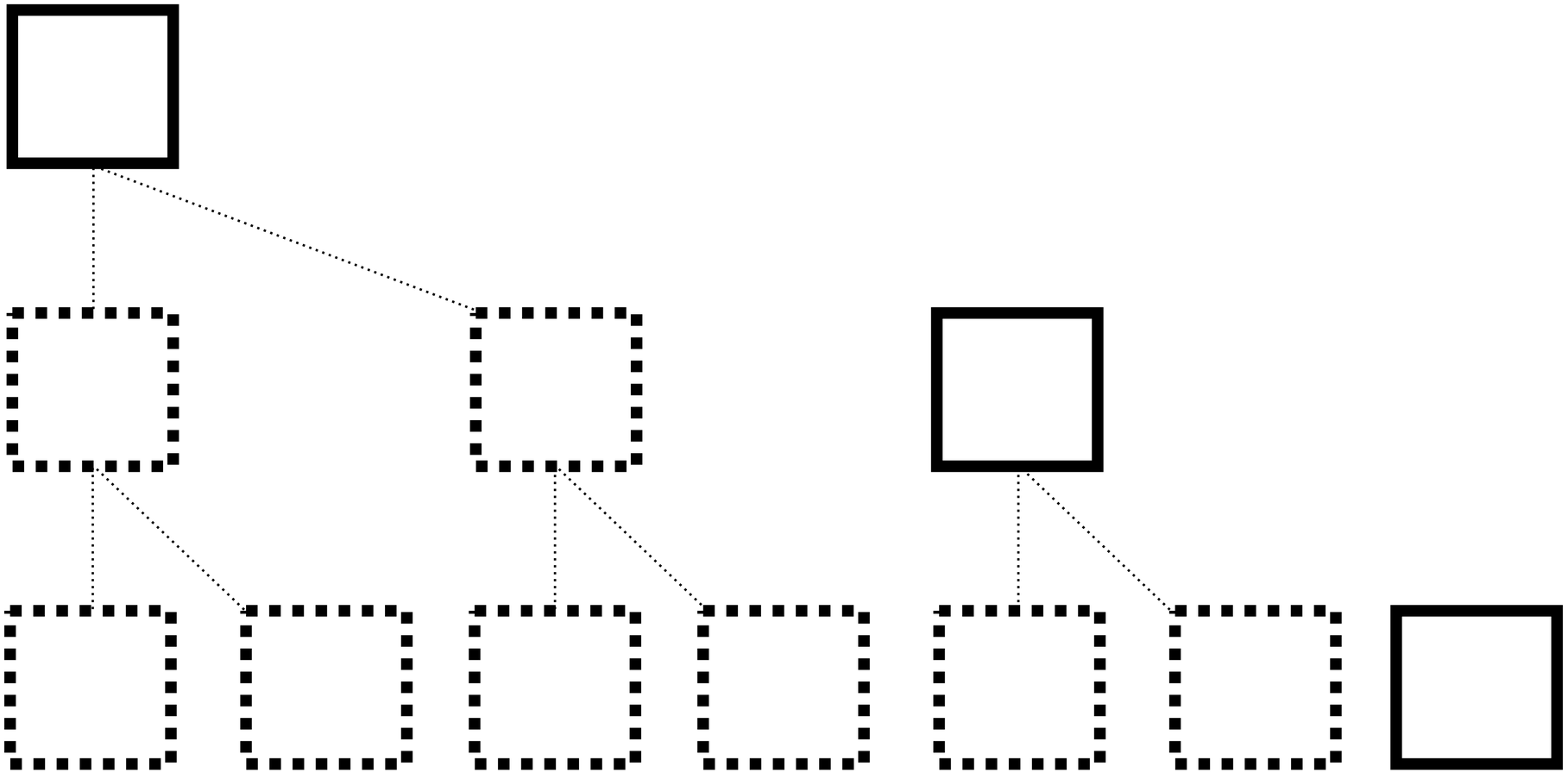
Throw away



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

7 UTXOs

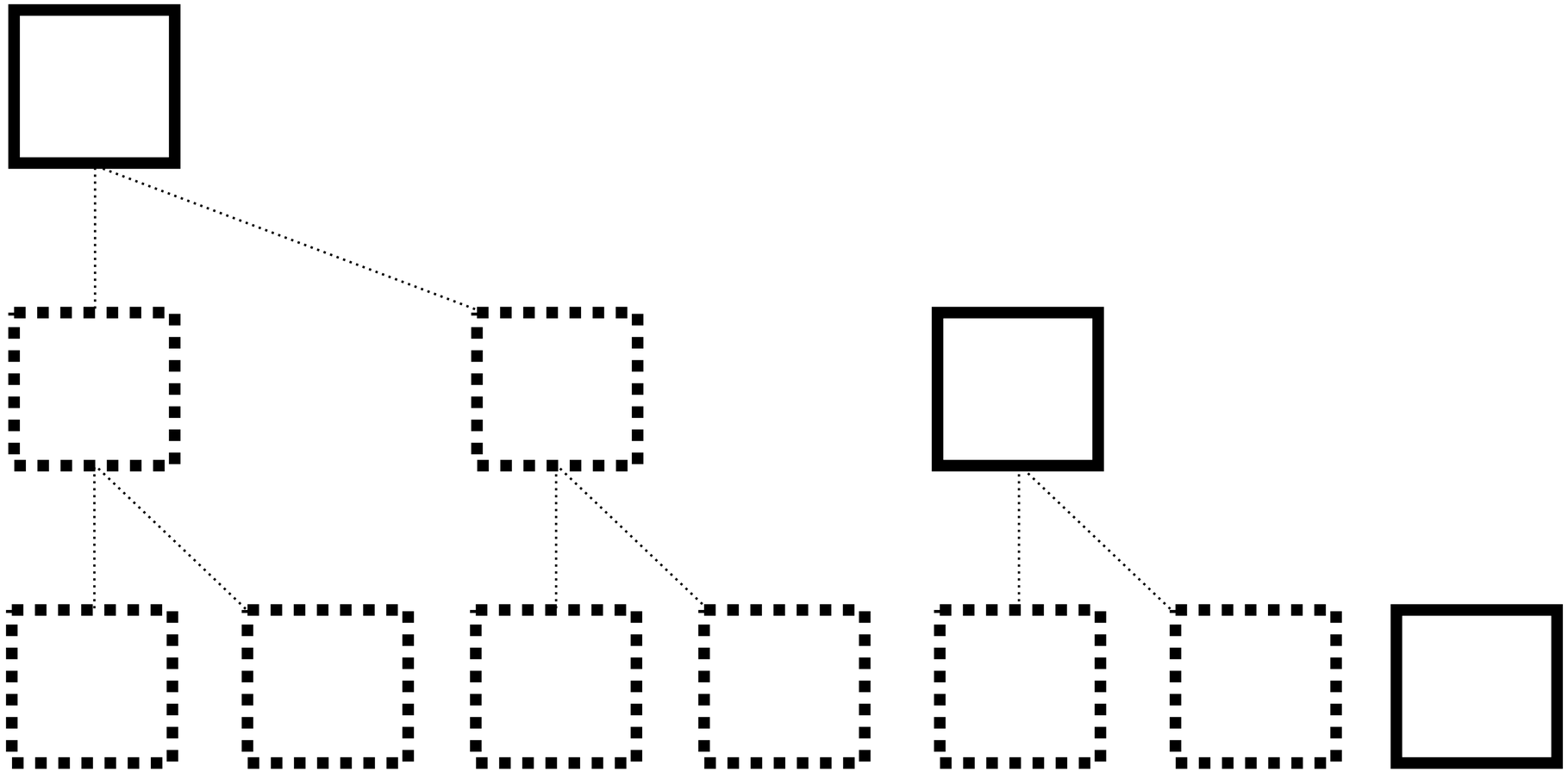
Another one



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

7 UTXOs

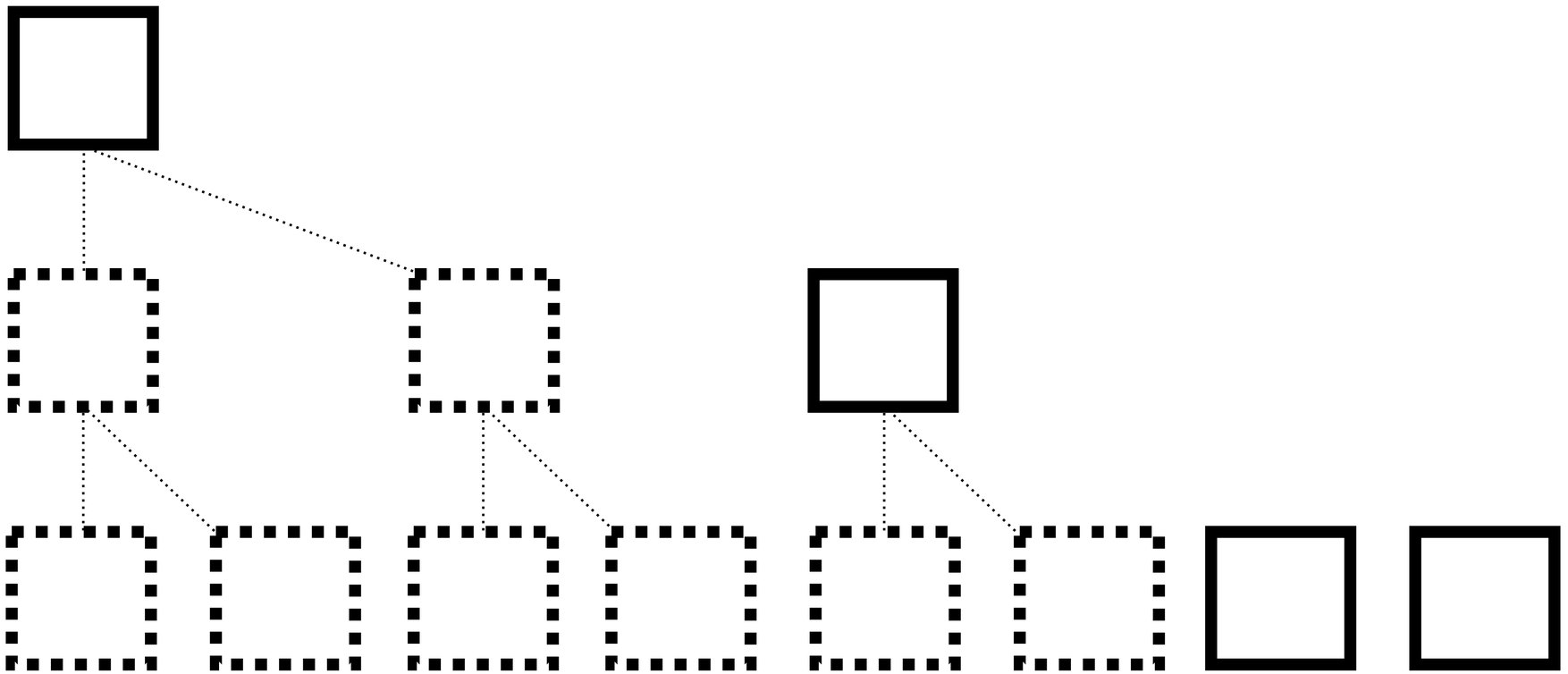
Nothing to hash with so it becomes a root



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

8 UTXOs

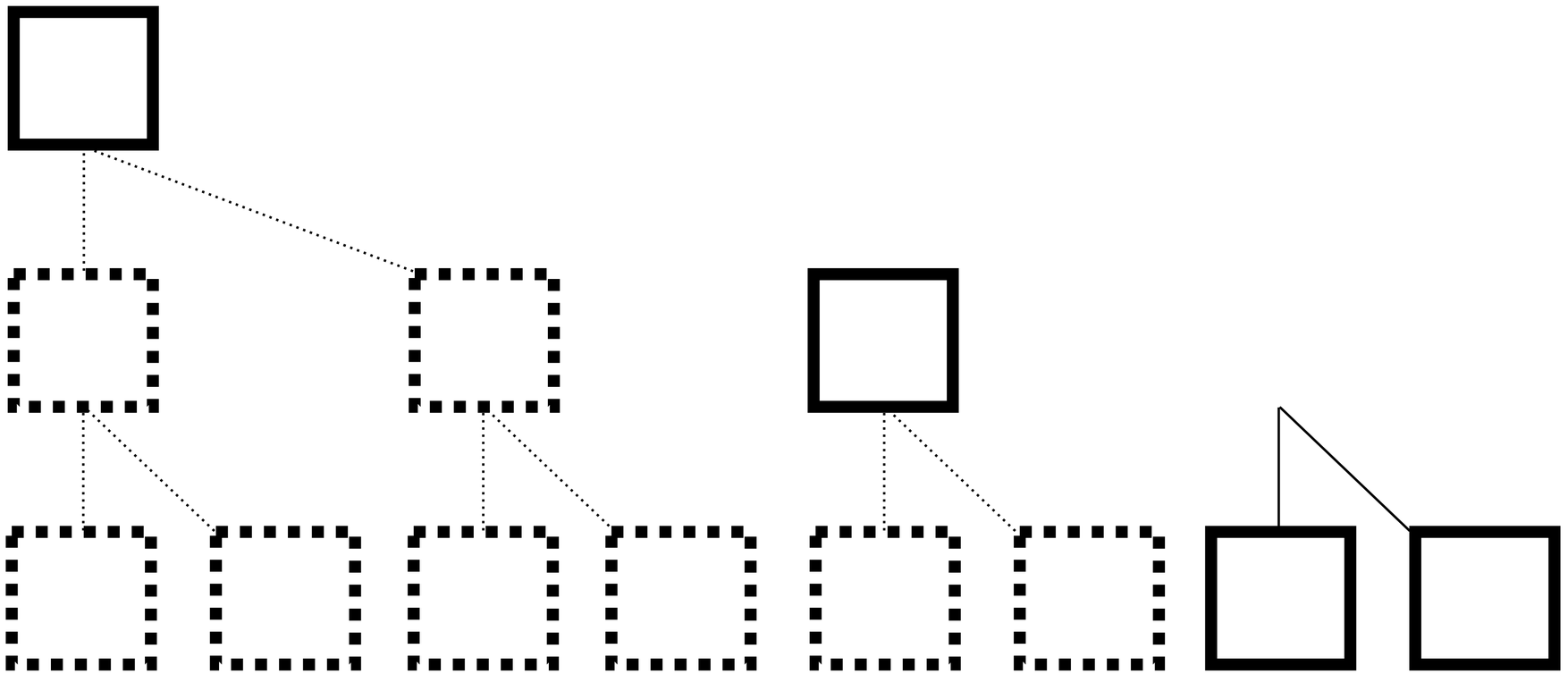
Another one



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

8 UTXOs

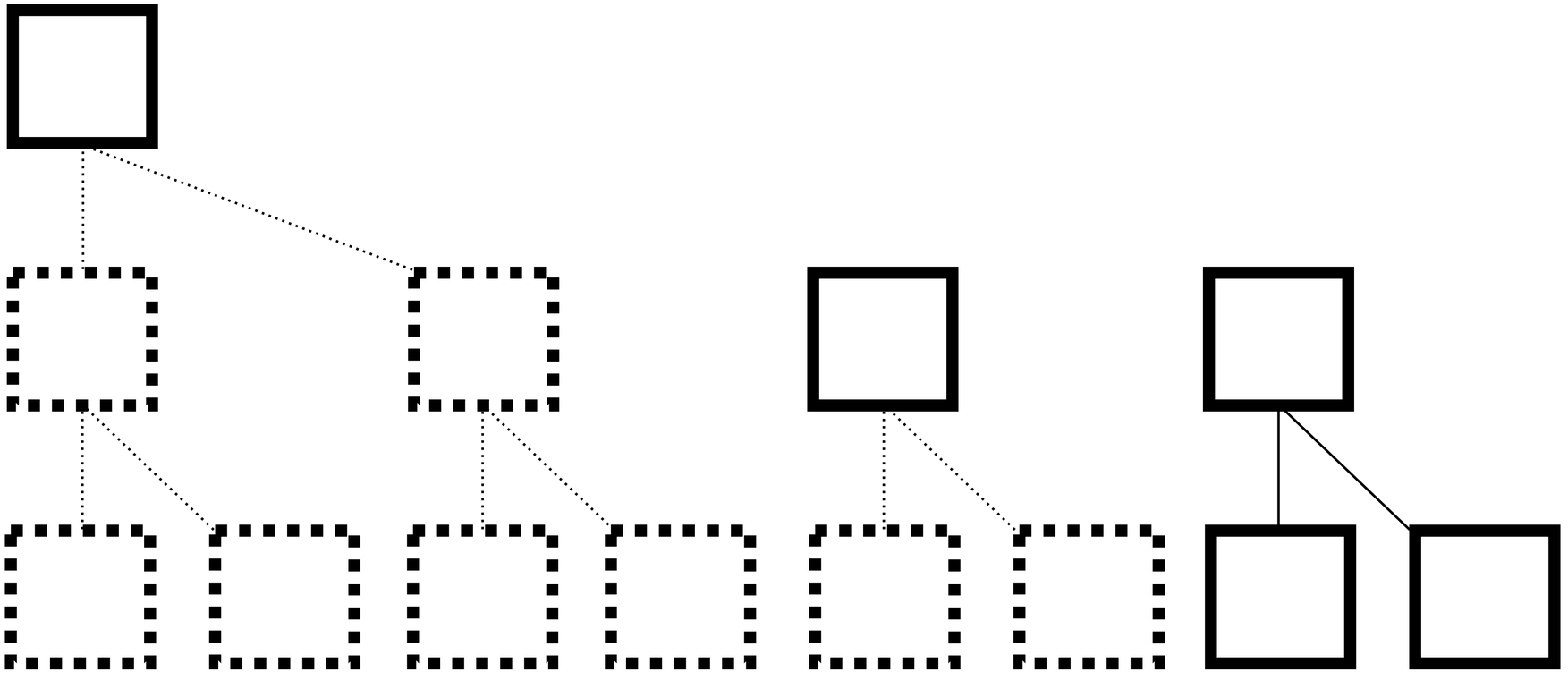
Concatenate



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

8 UTXOs

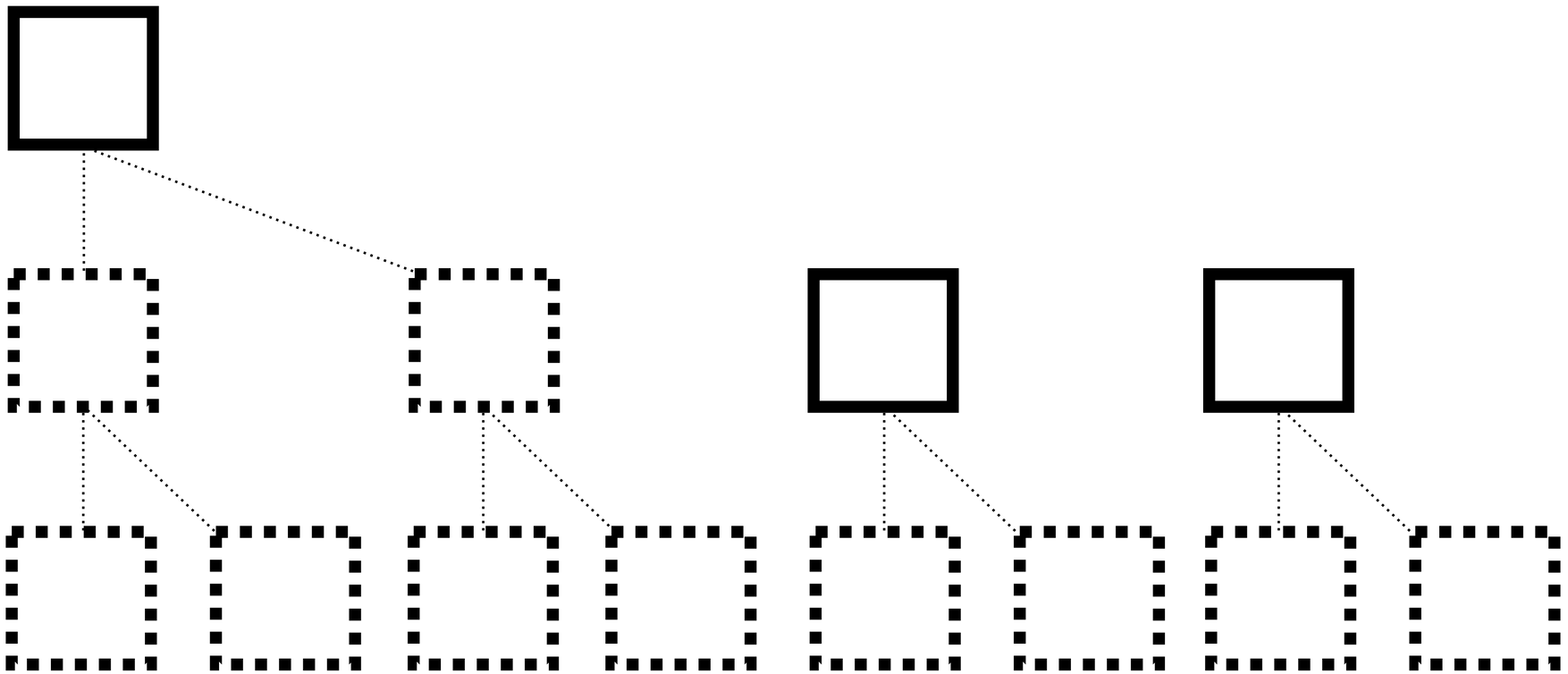
Hash



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

8 UTXOs

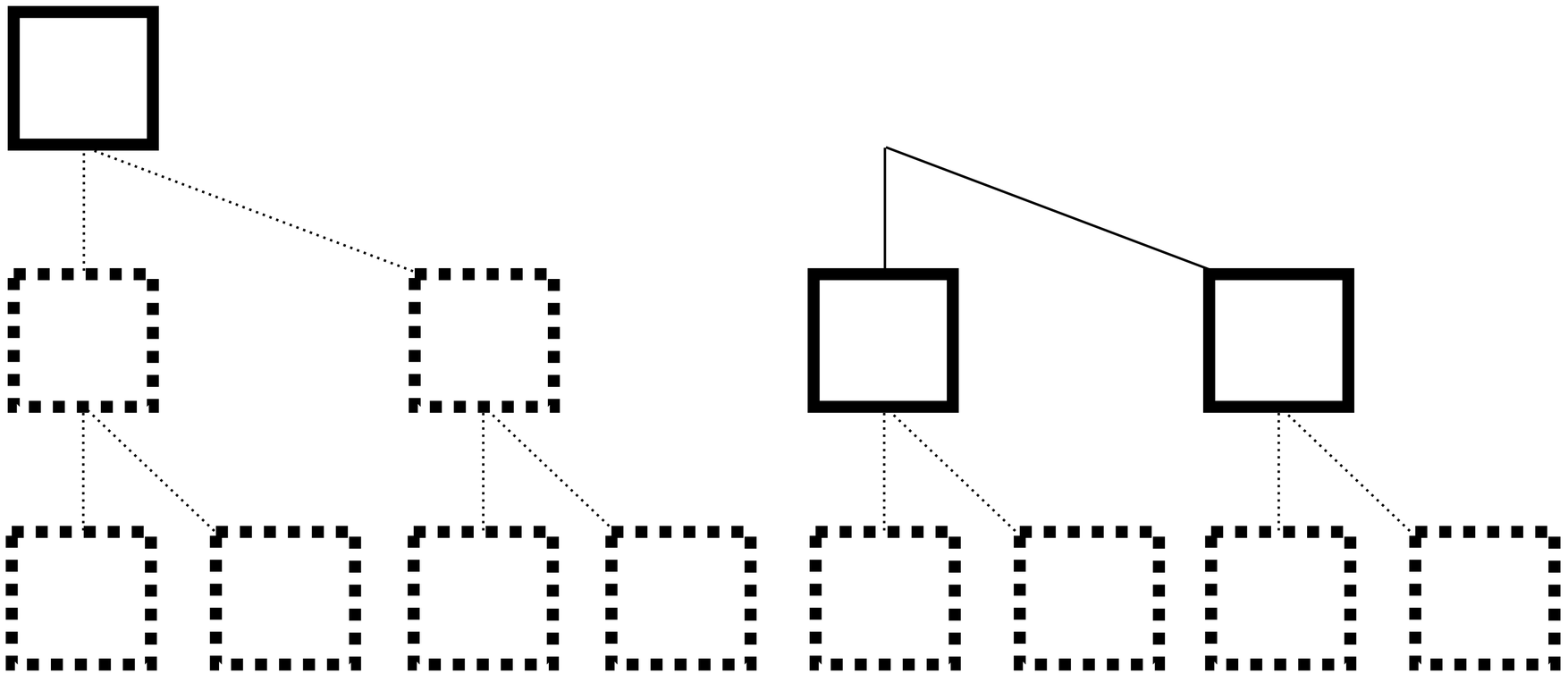
Throw away



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

8 UTXOs

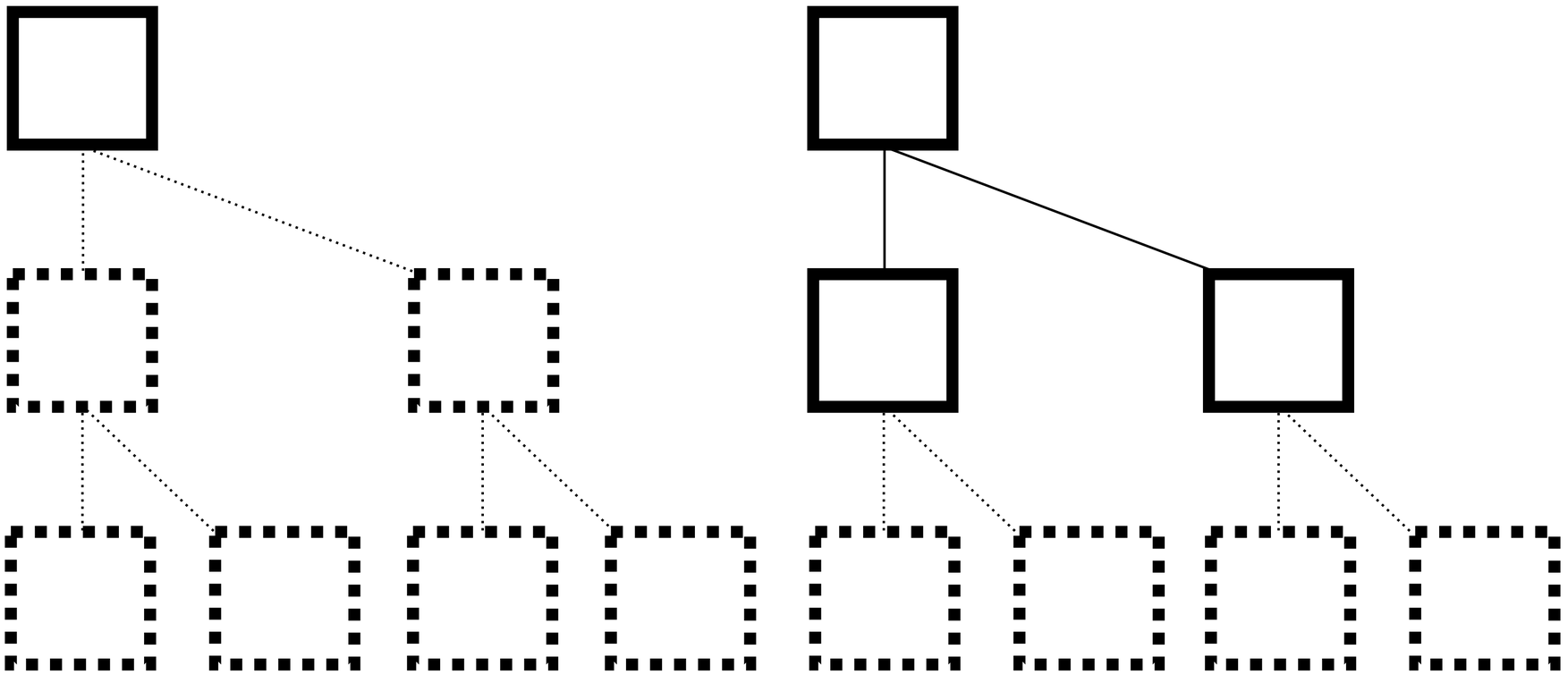
Concatenate



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

8 UTXOs

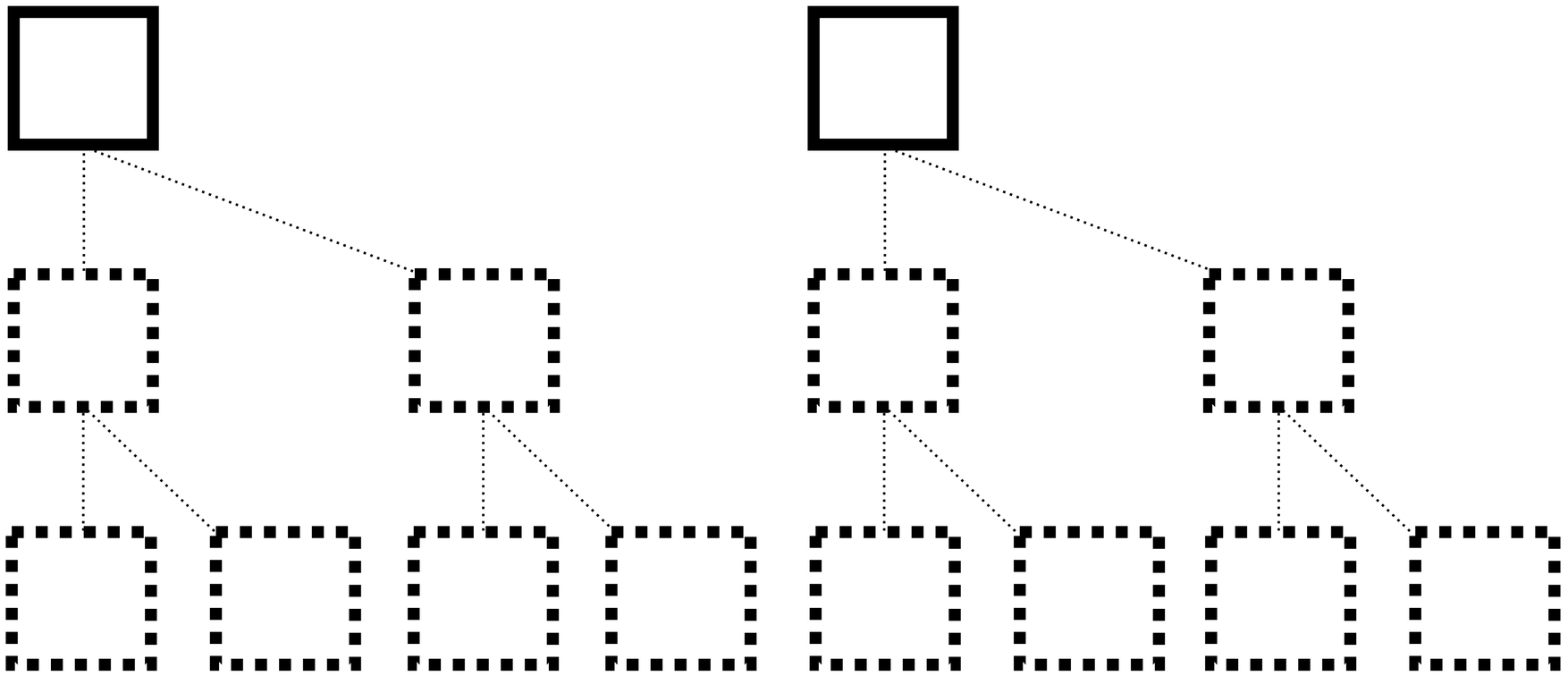
Hash



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

8 UTXOs

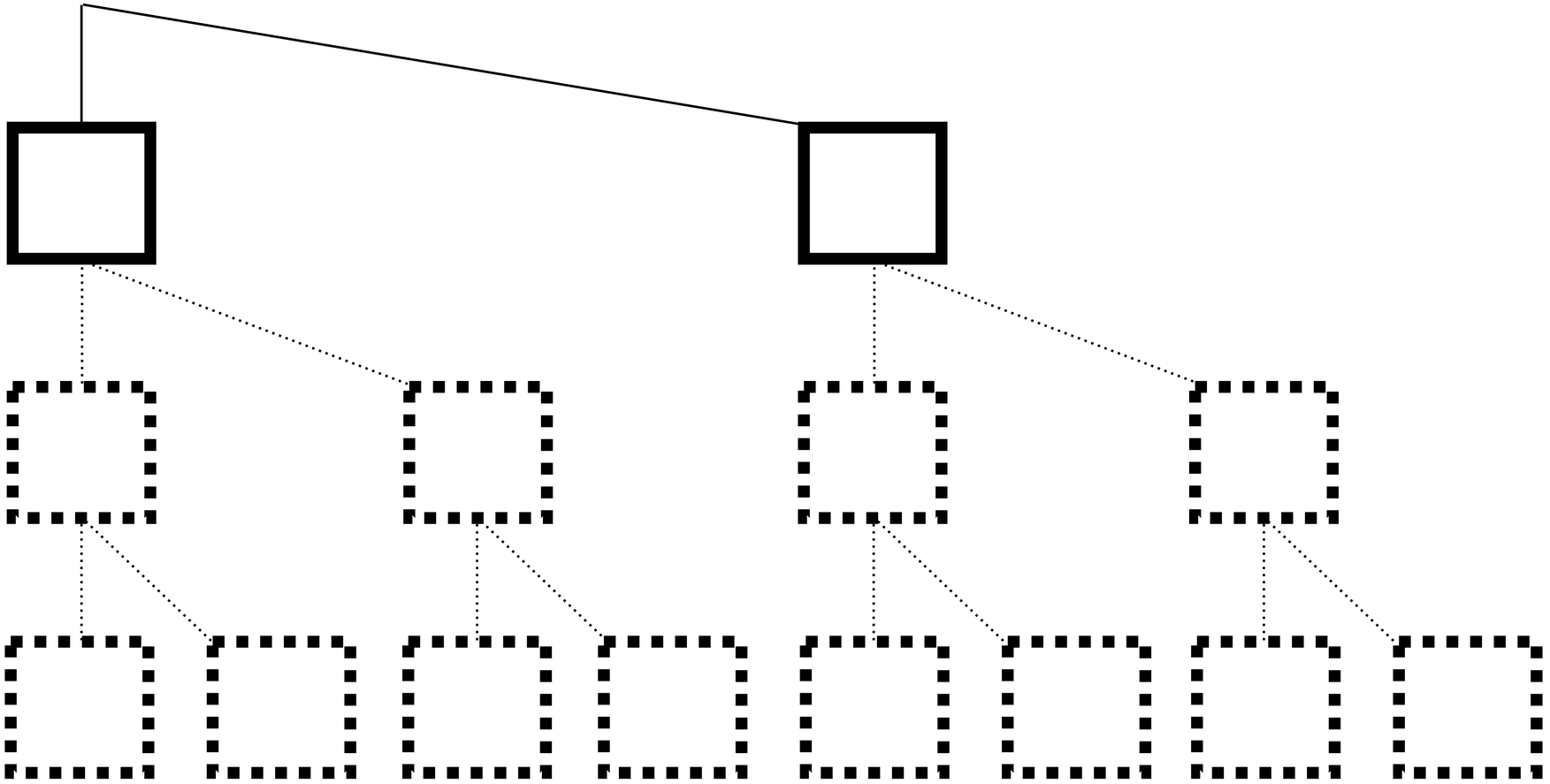
Throw away



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

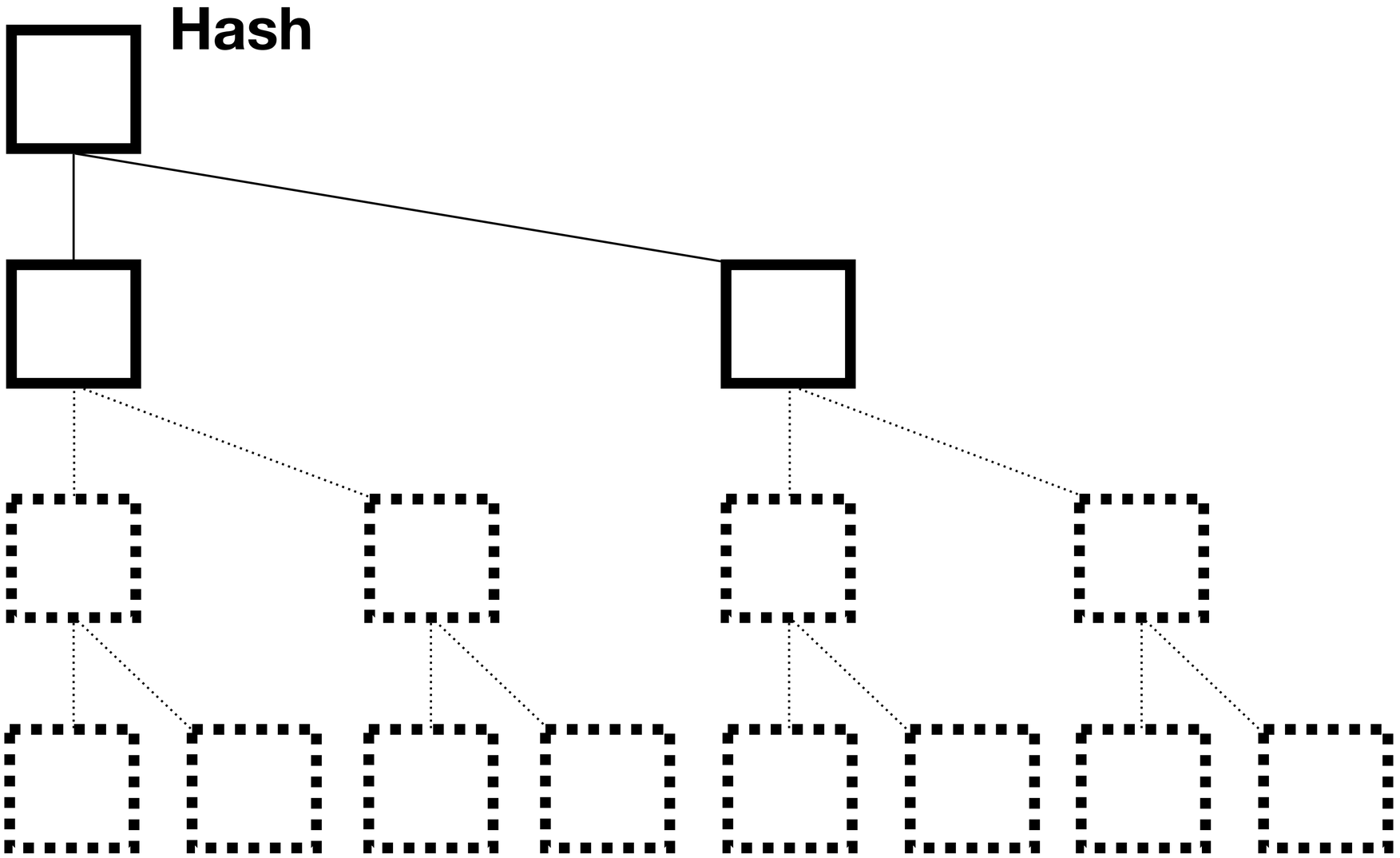
8 UTXOs

Concatenate



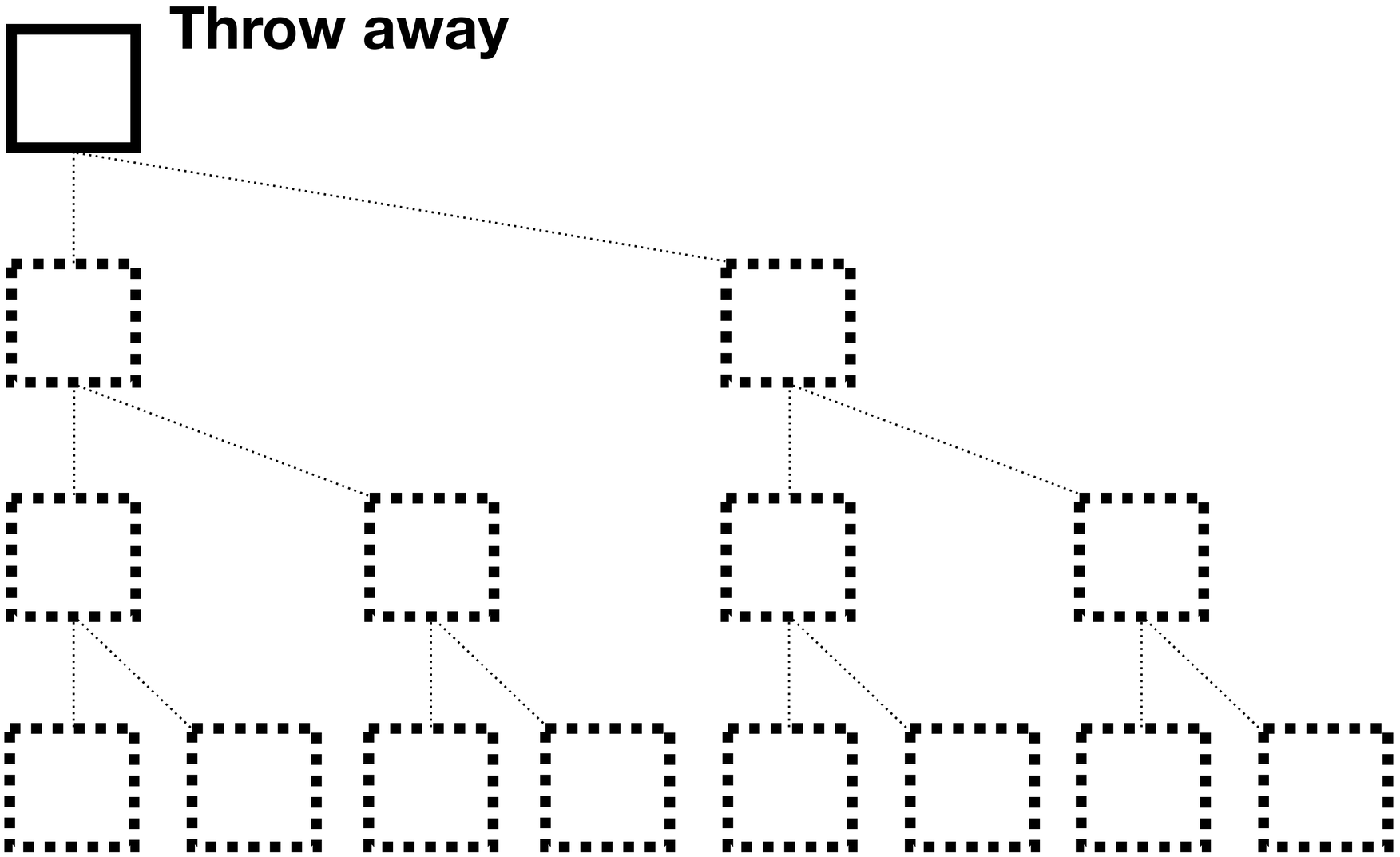
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

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.

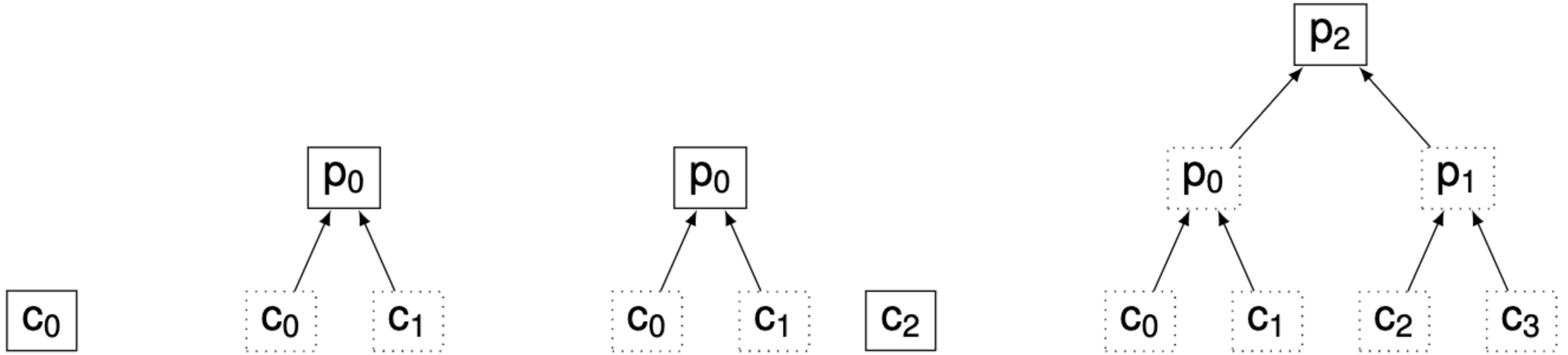


Figure 6: An incomplete tree growing incrementally from 1 to 4 chunks. Dotted boxes represent CVs that no longer need to be stored.

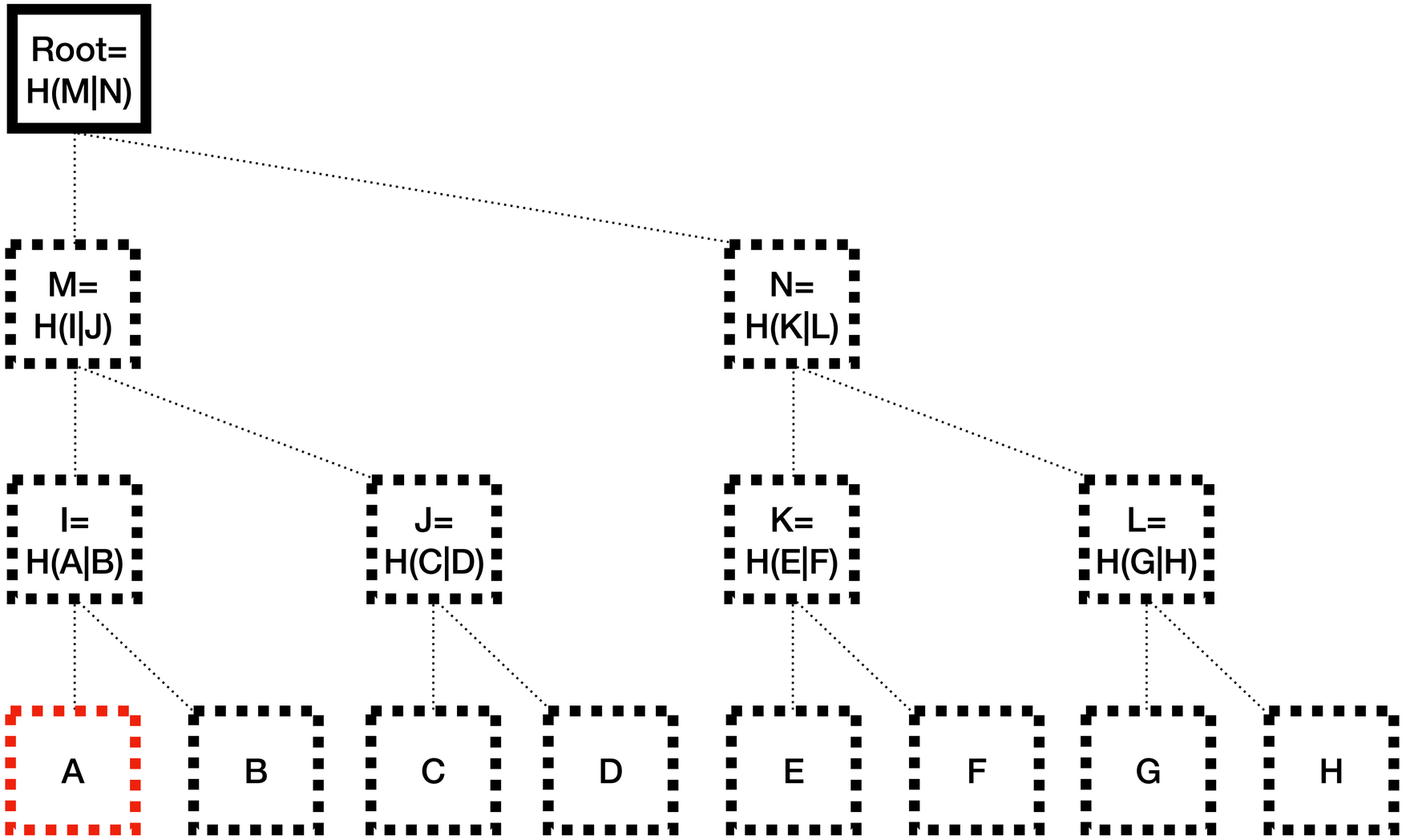
Role of levelDB

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

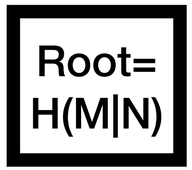
Deletion happens with proving existence

Deleting A



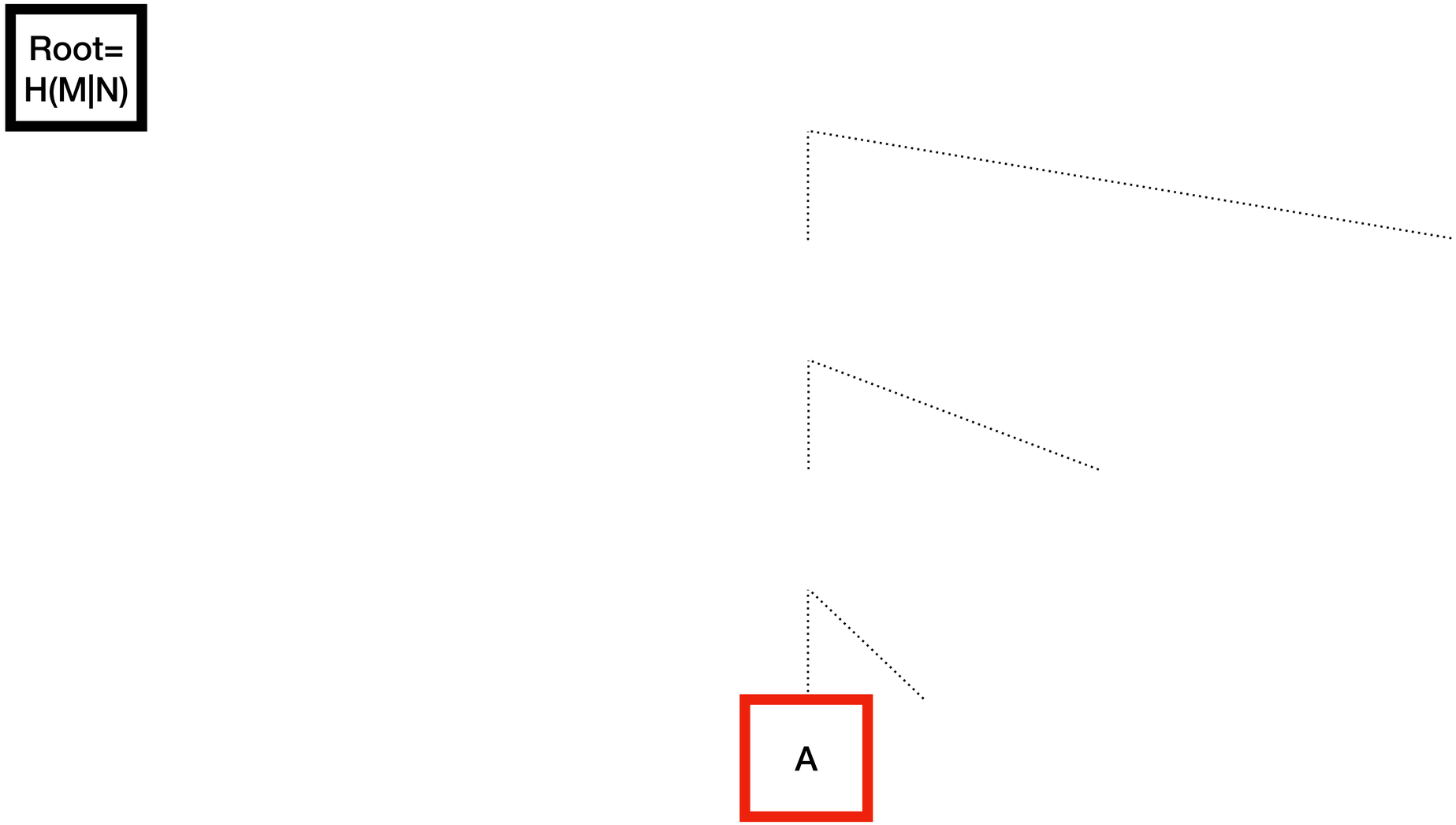
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

The only requirement is the roots



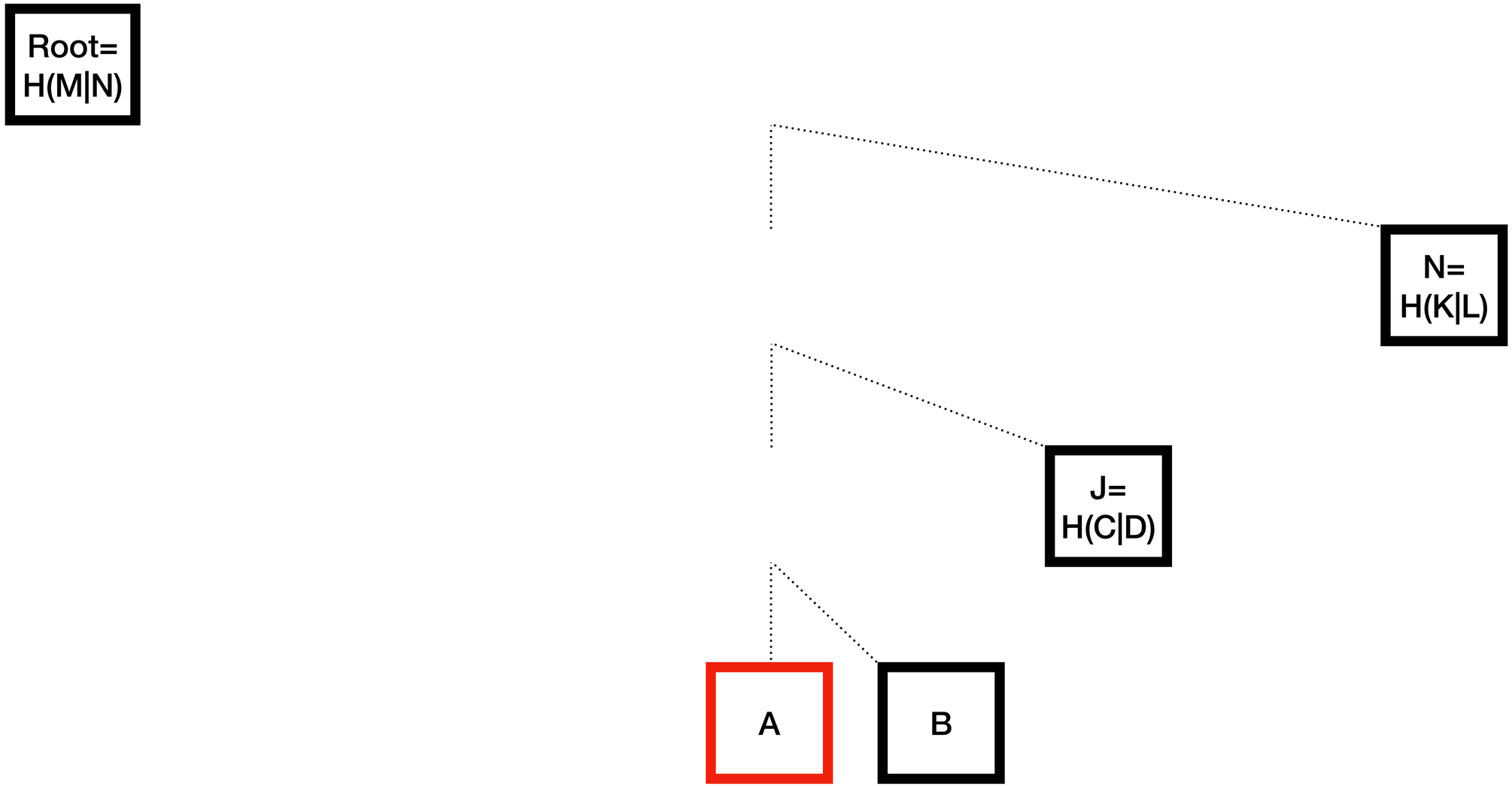
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate A from the Bitcoin Block



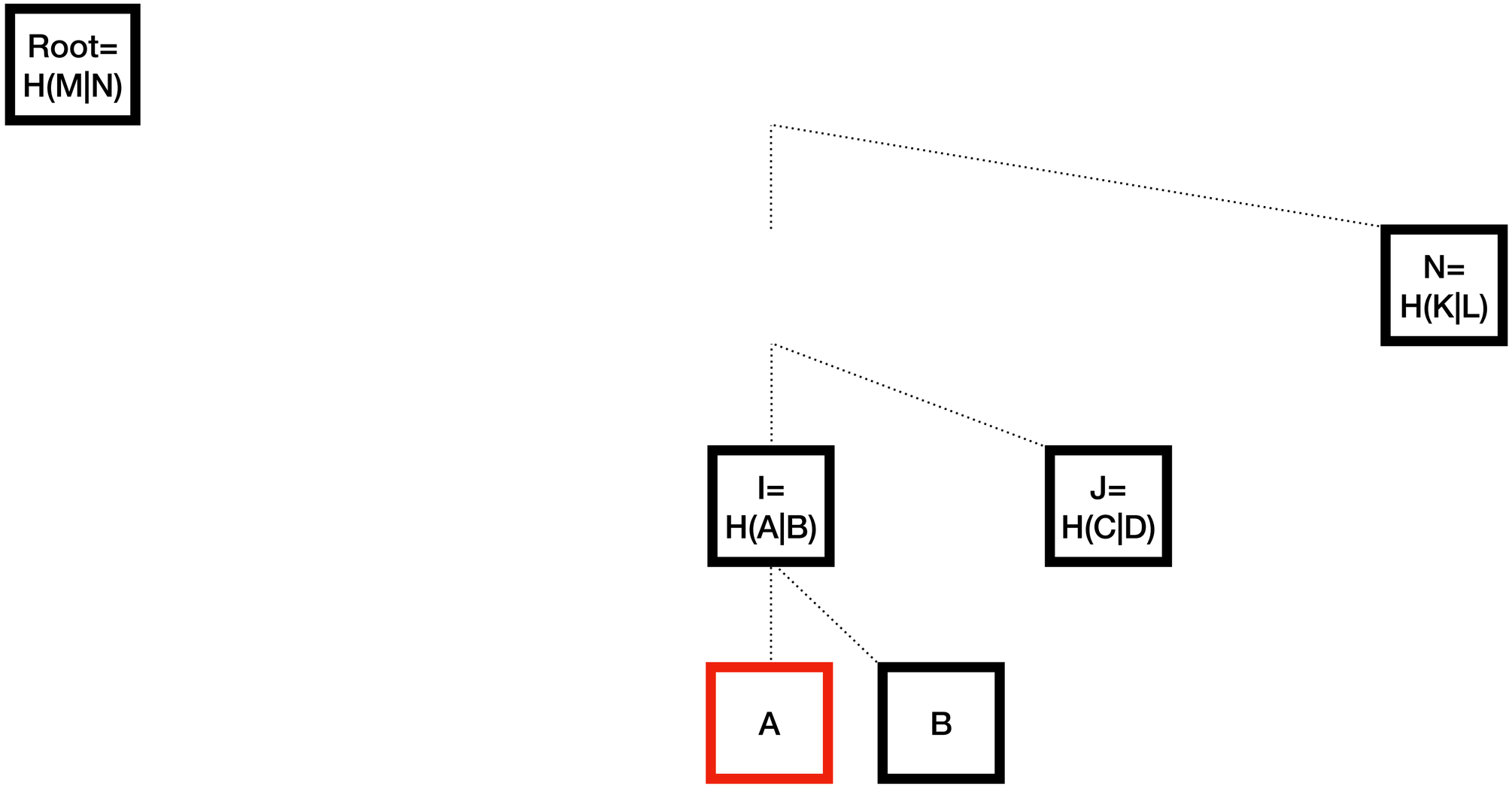
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Receive the proof: B, J, N



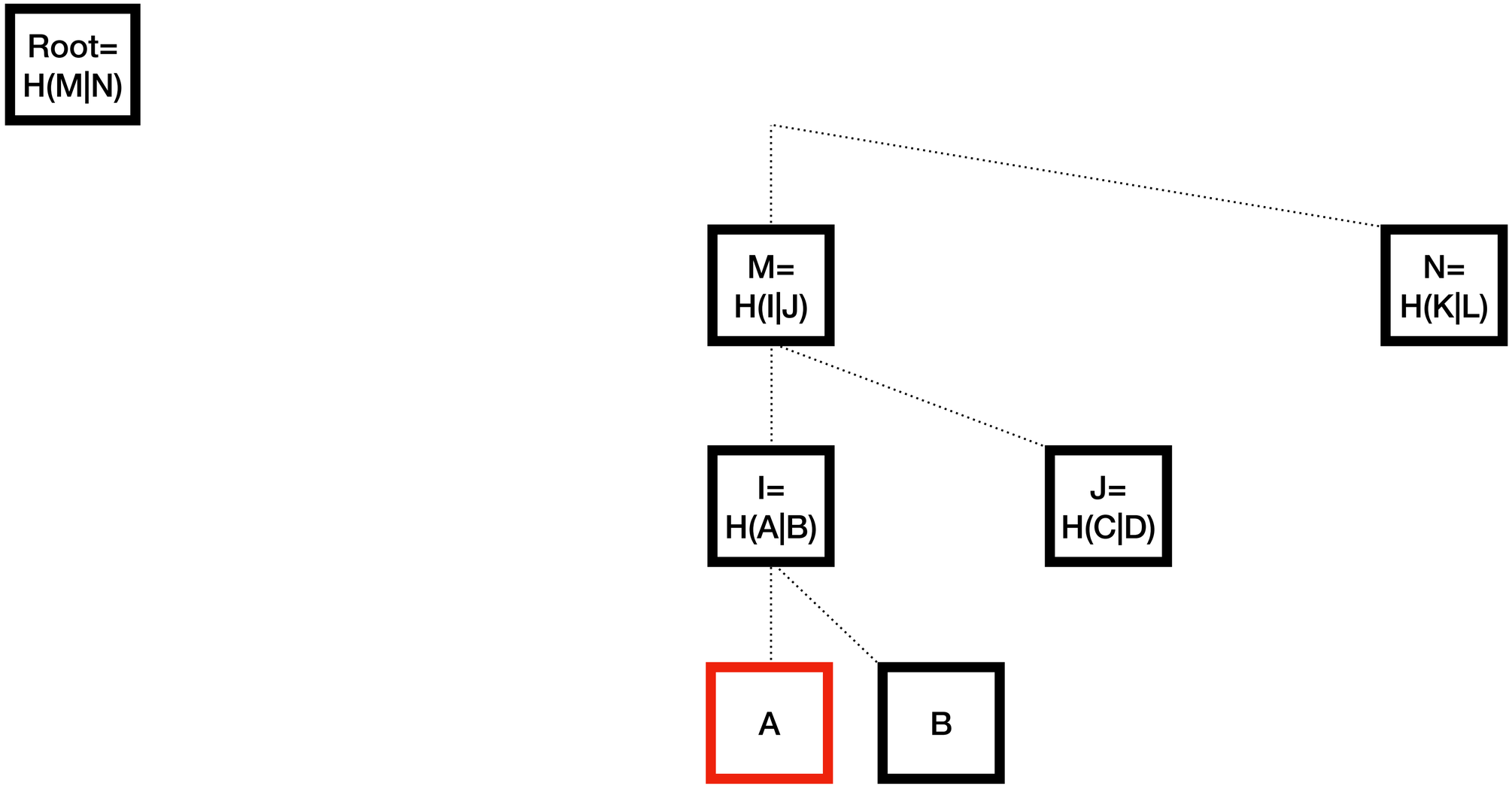
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate the hash for I



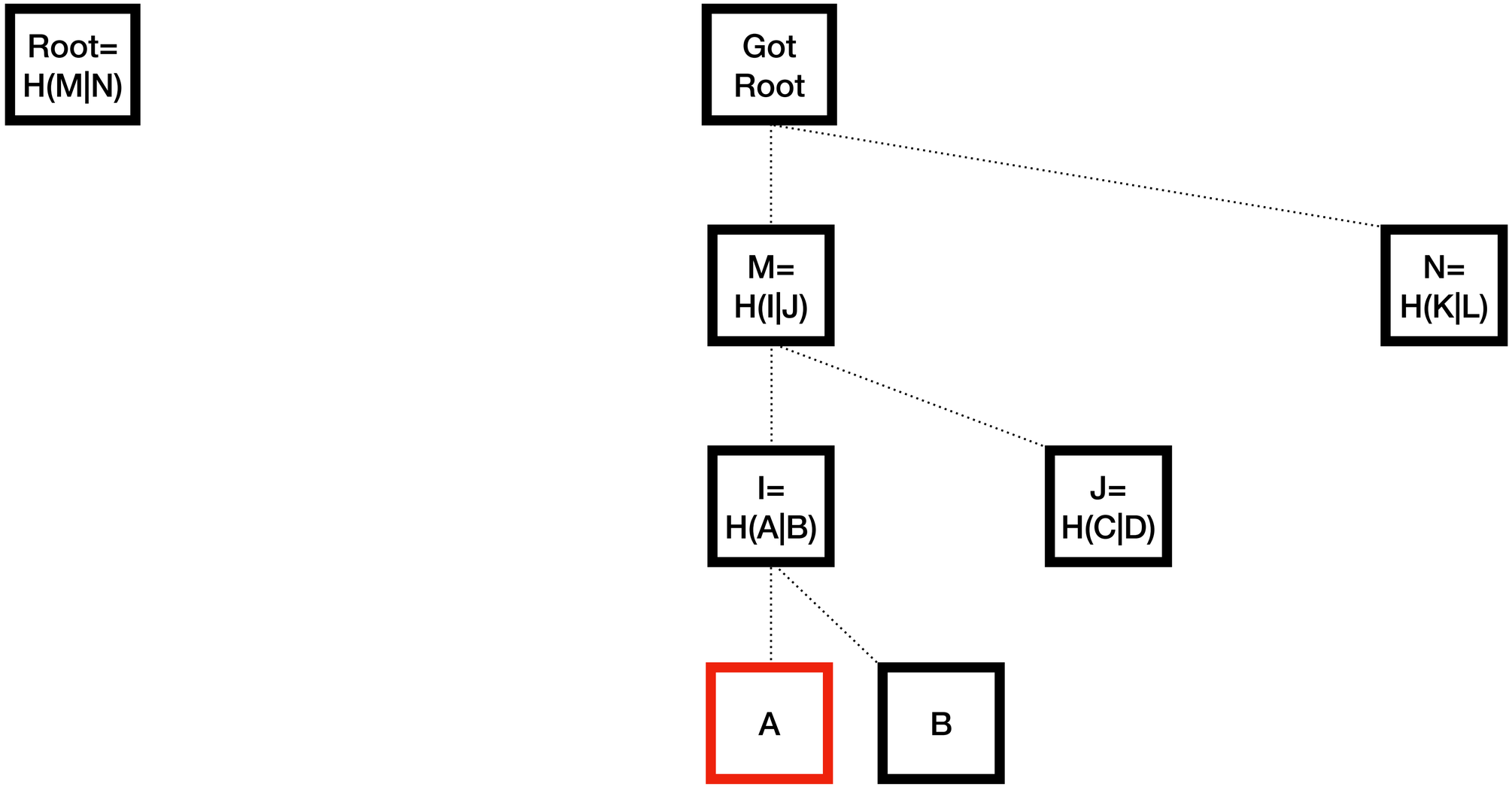
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate the hash for M



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

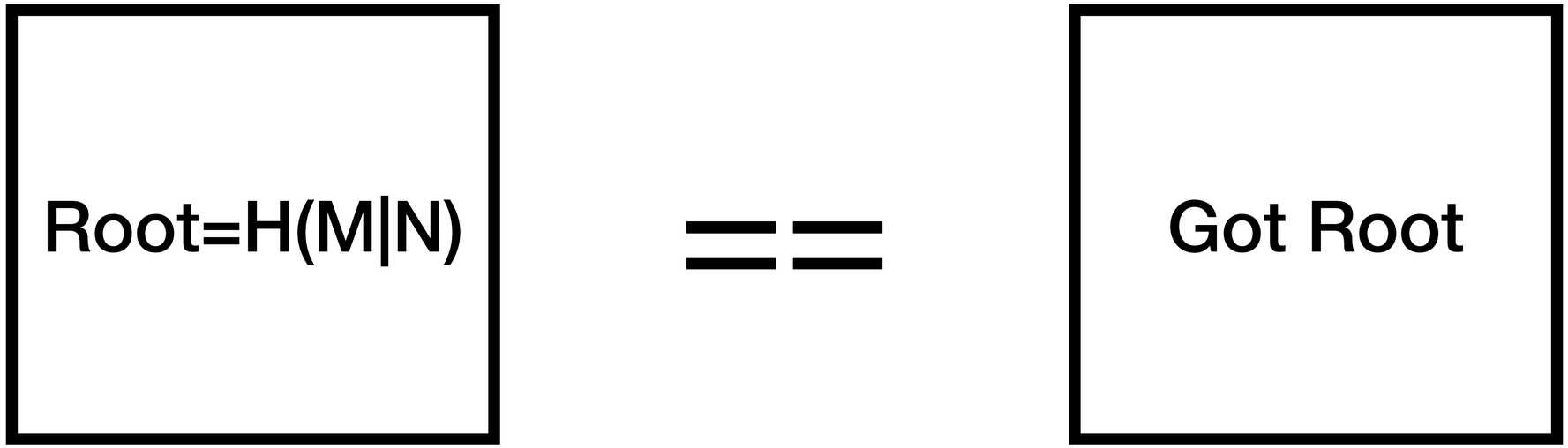
Calculate Root



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

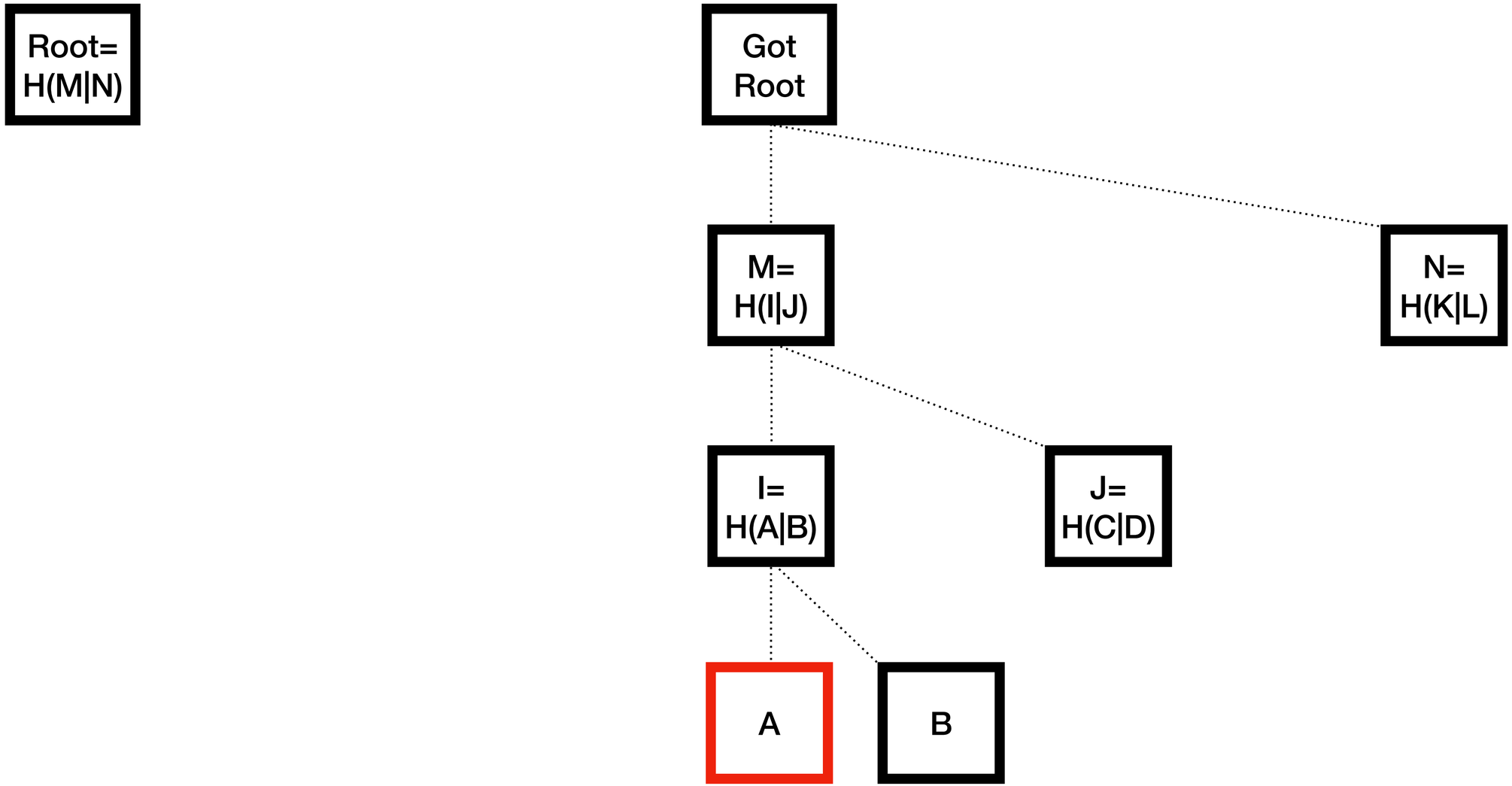
Compare roots

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



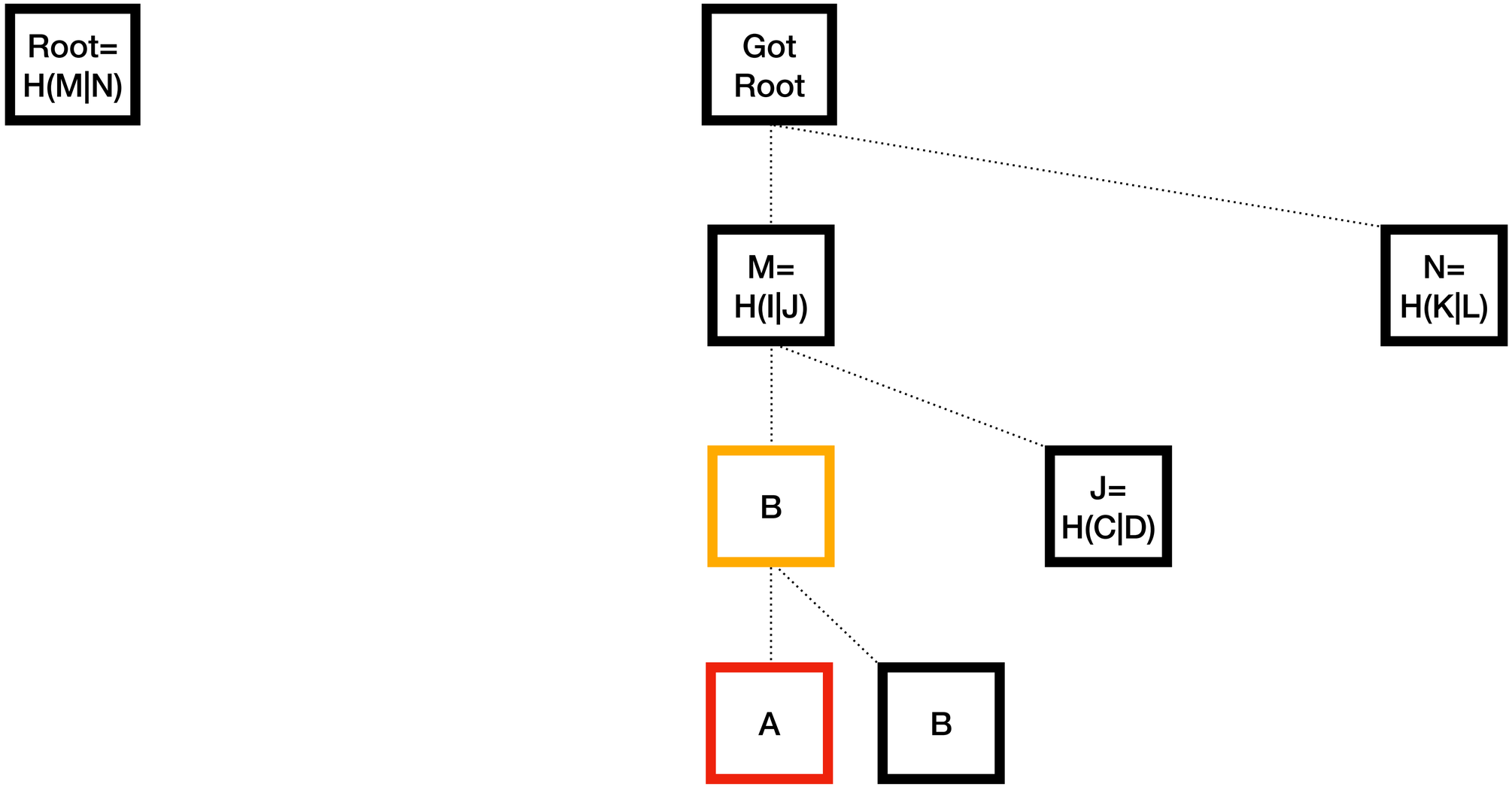
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate root after deletion



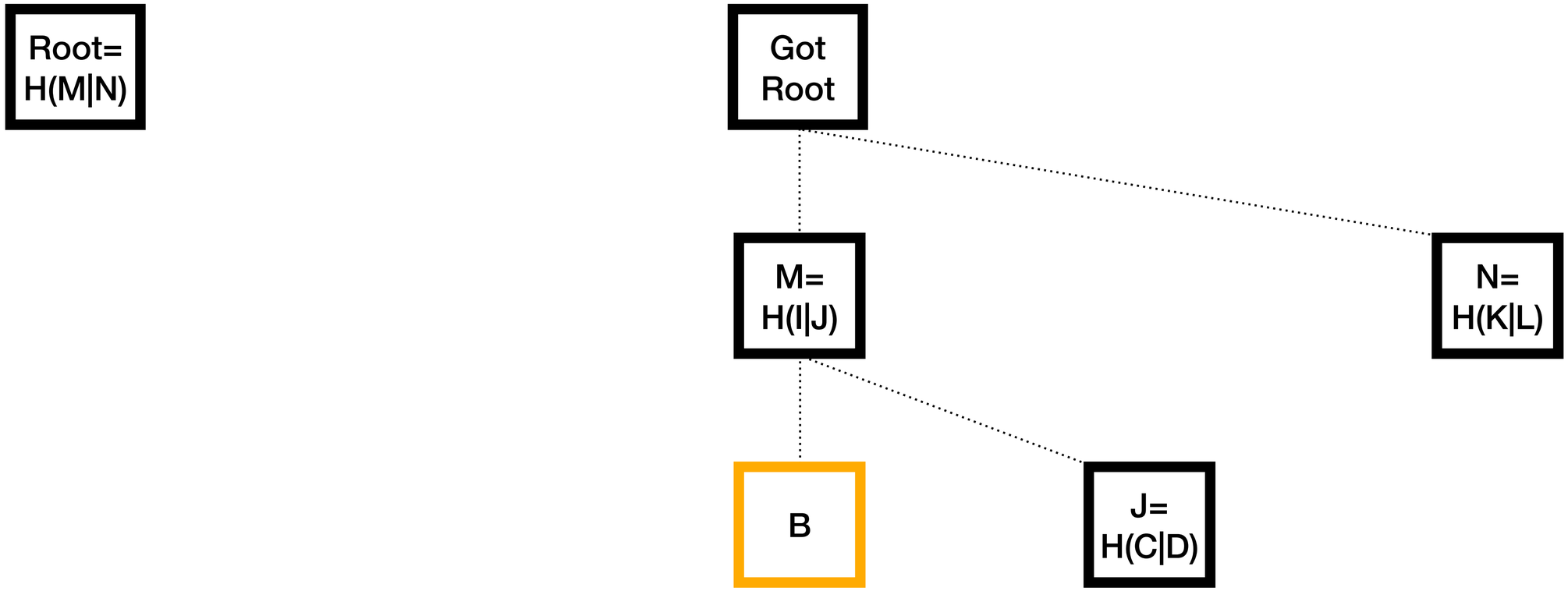
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Move up B to I



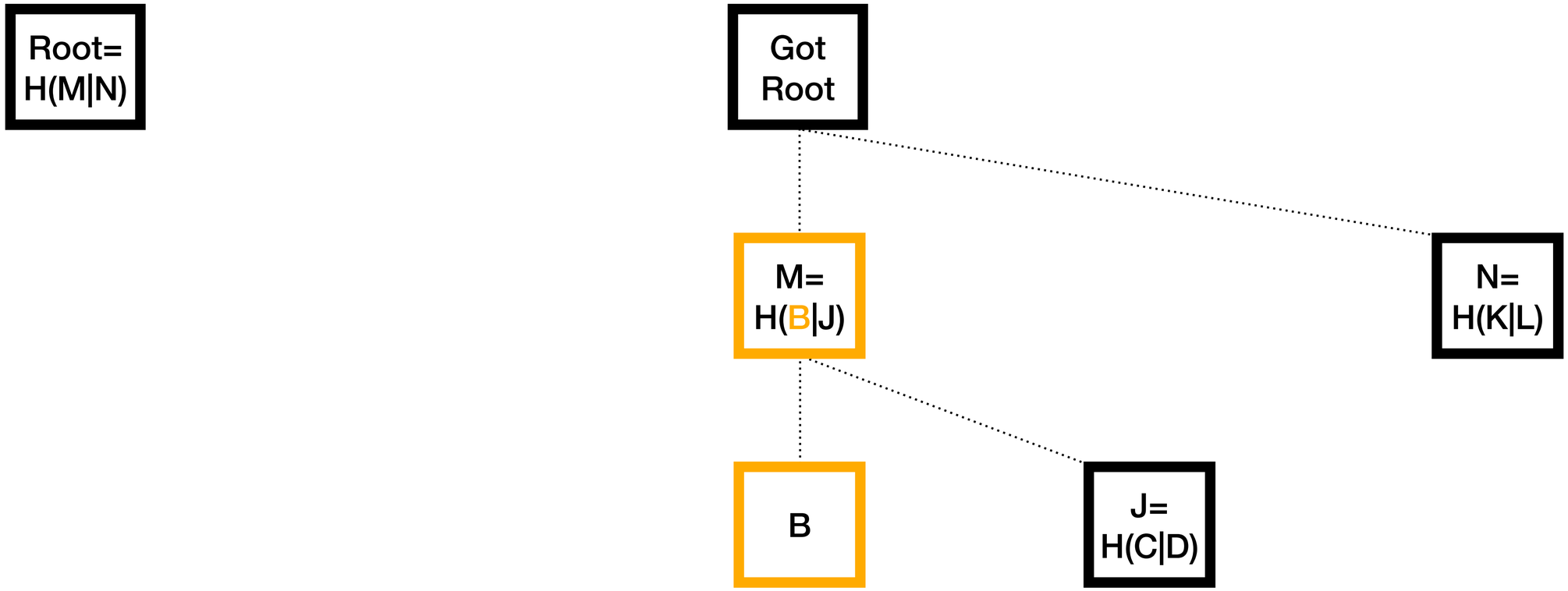
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Remove old nodes for A&B



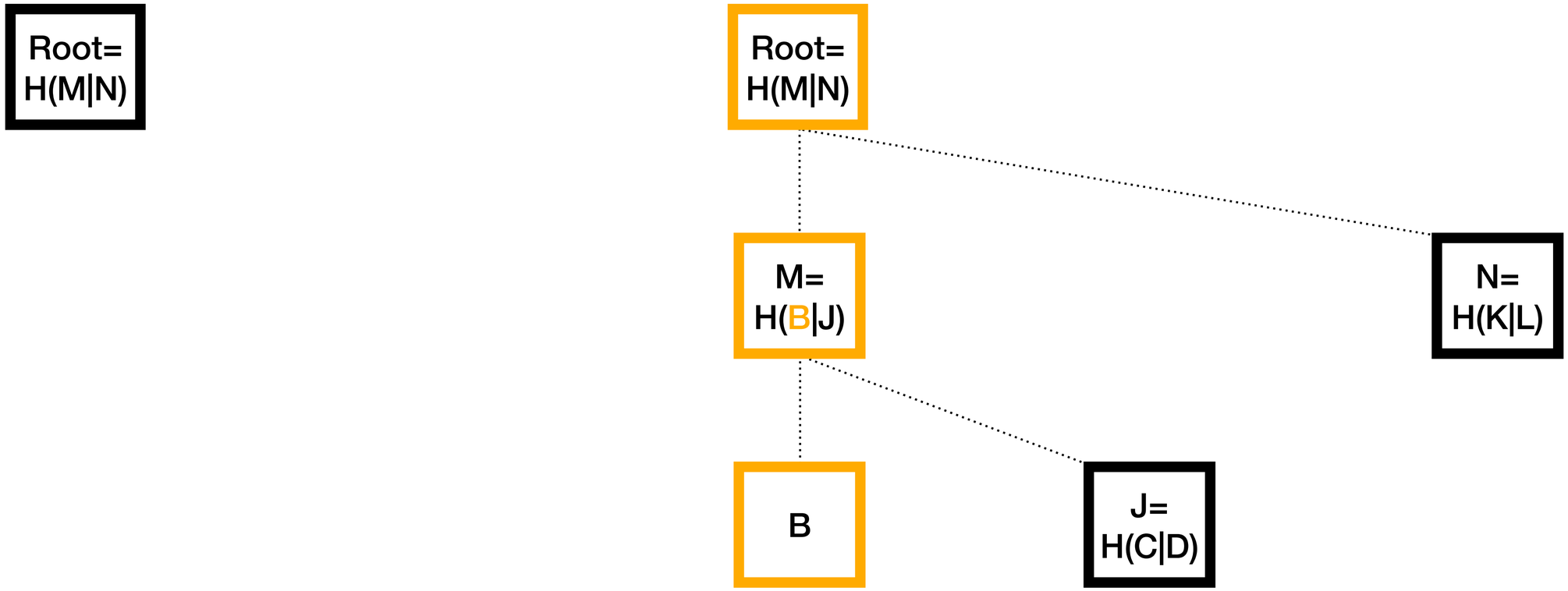
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate H(B|J)



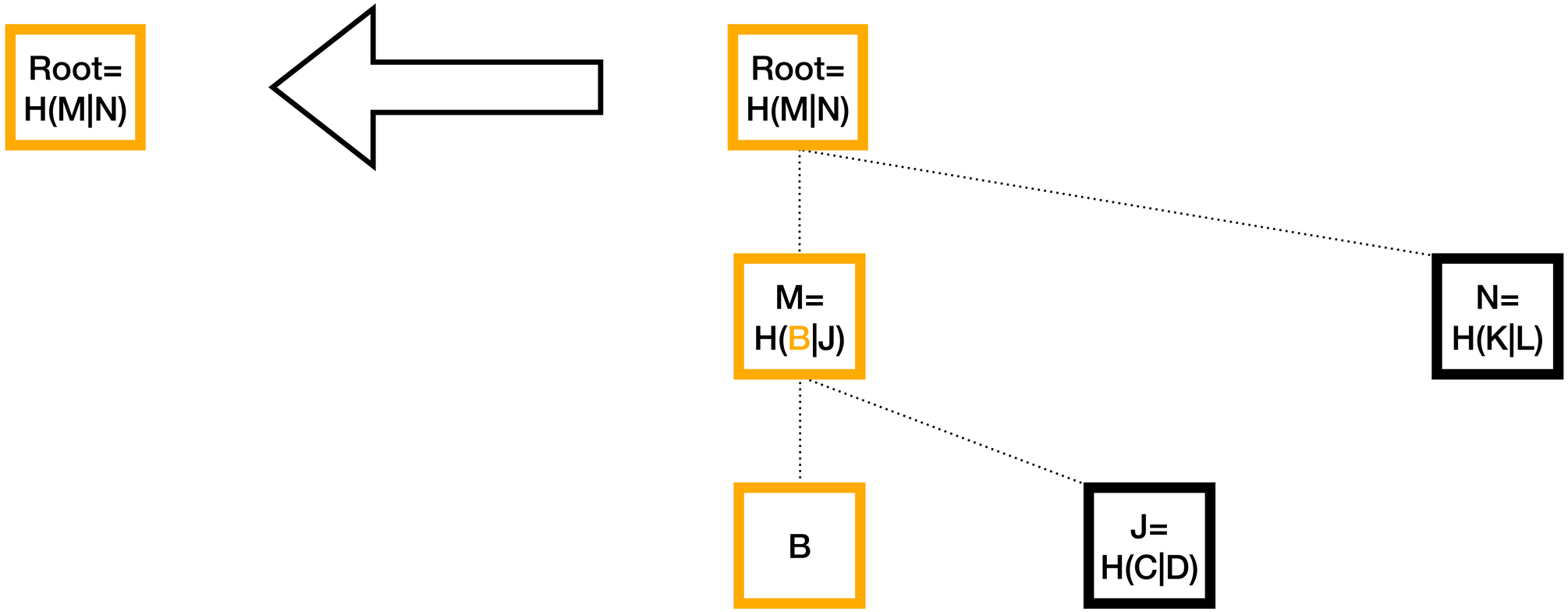
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate new root



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

Copy over the new root to be saved



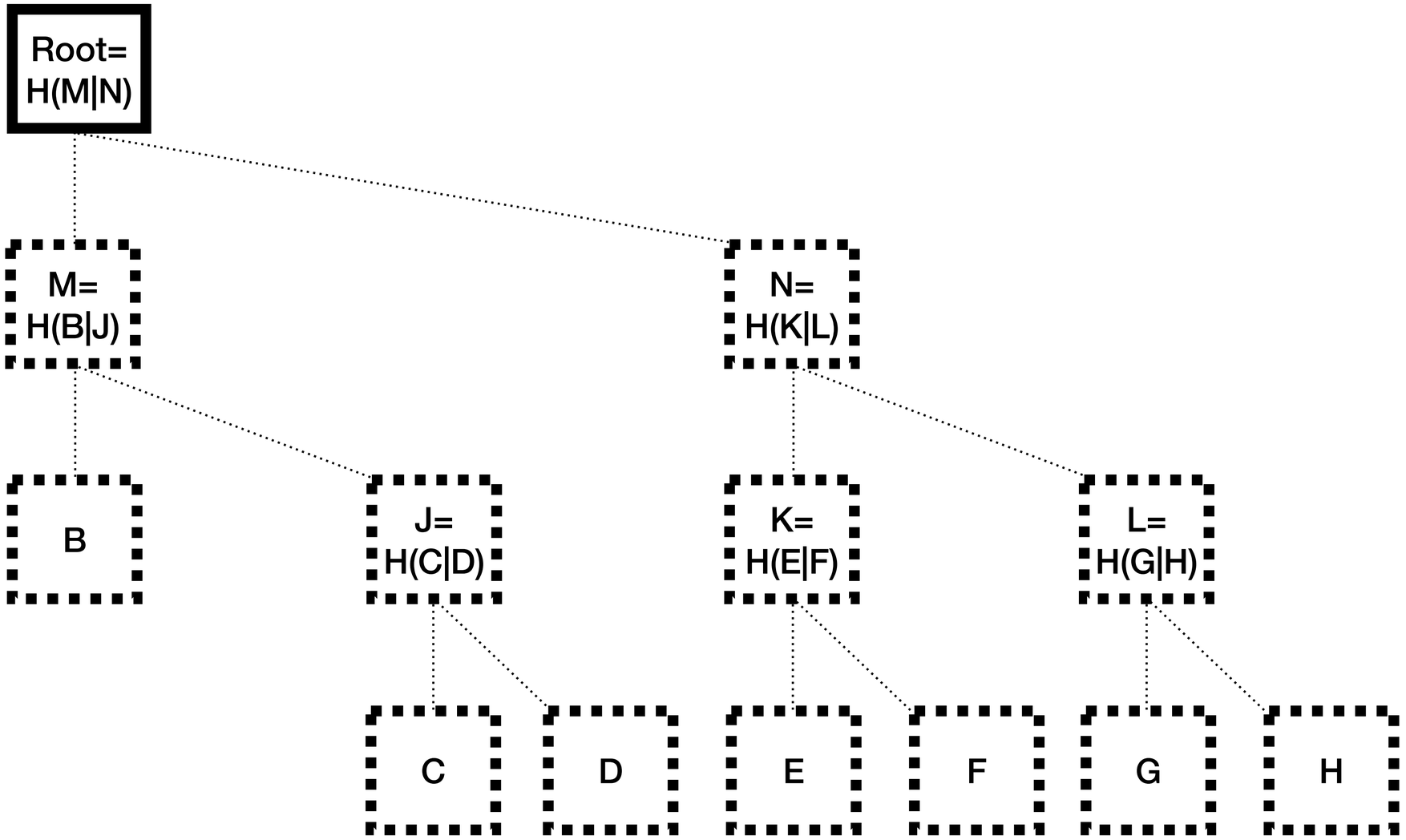
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Done

Root=
H(M|N)

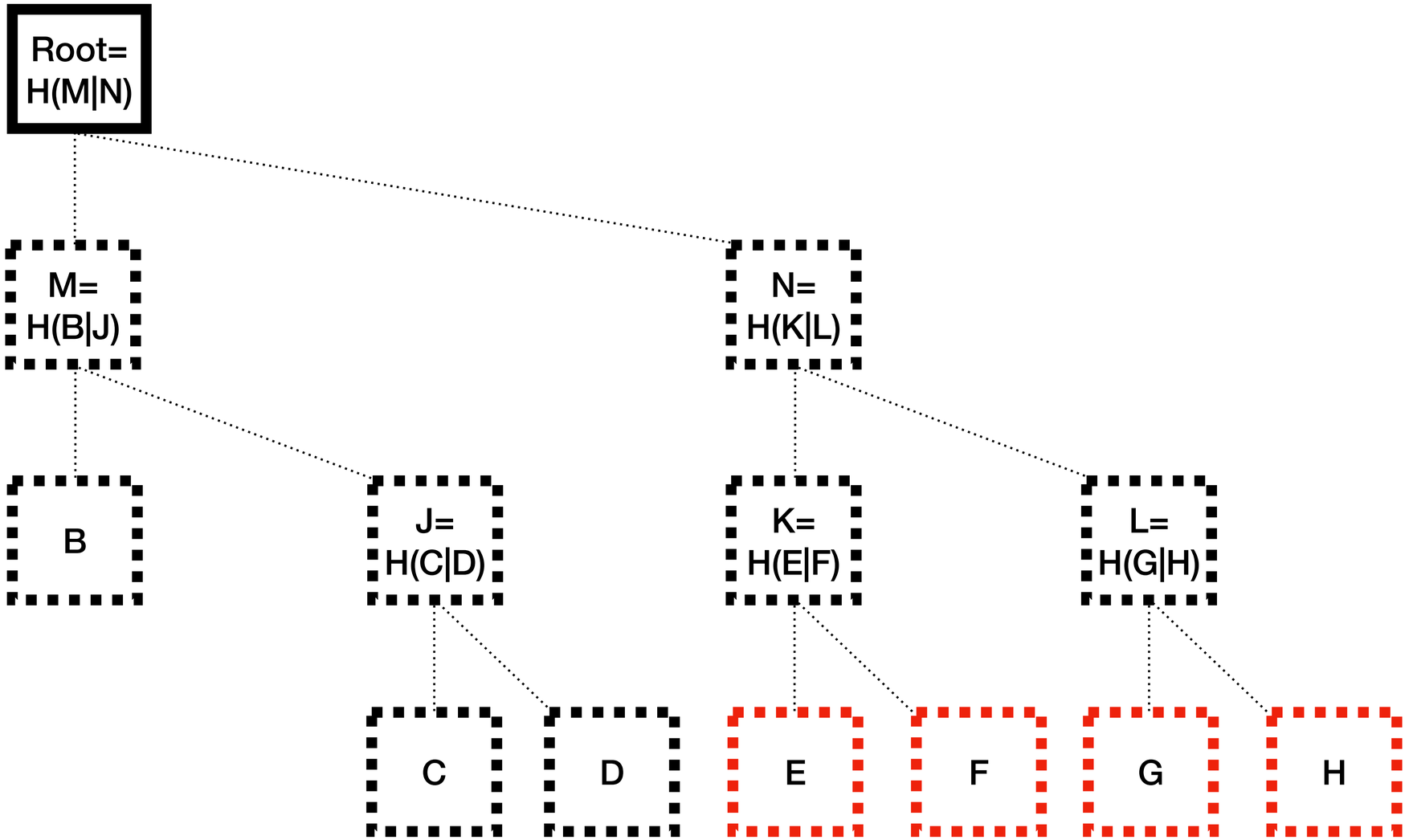
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

After deleting A



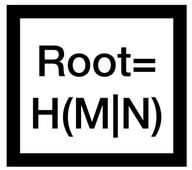
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

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



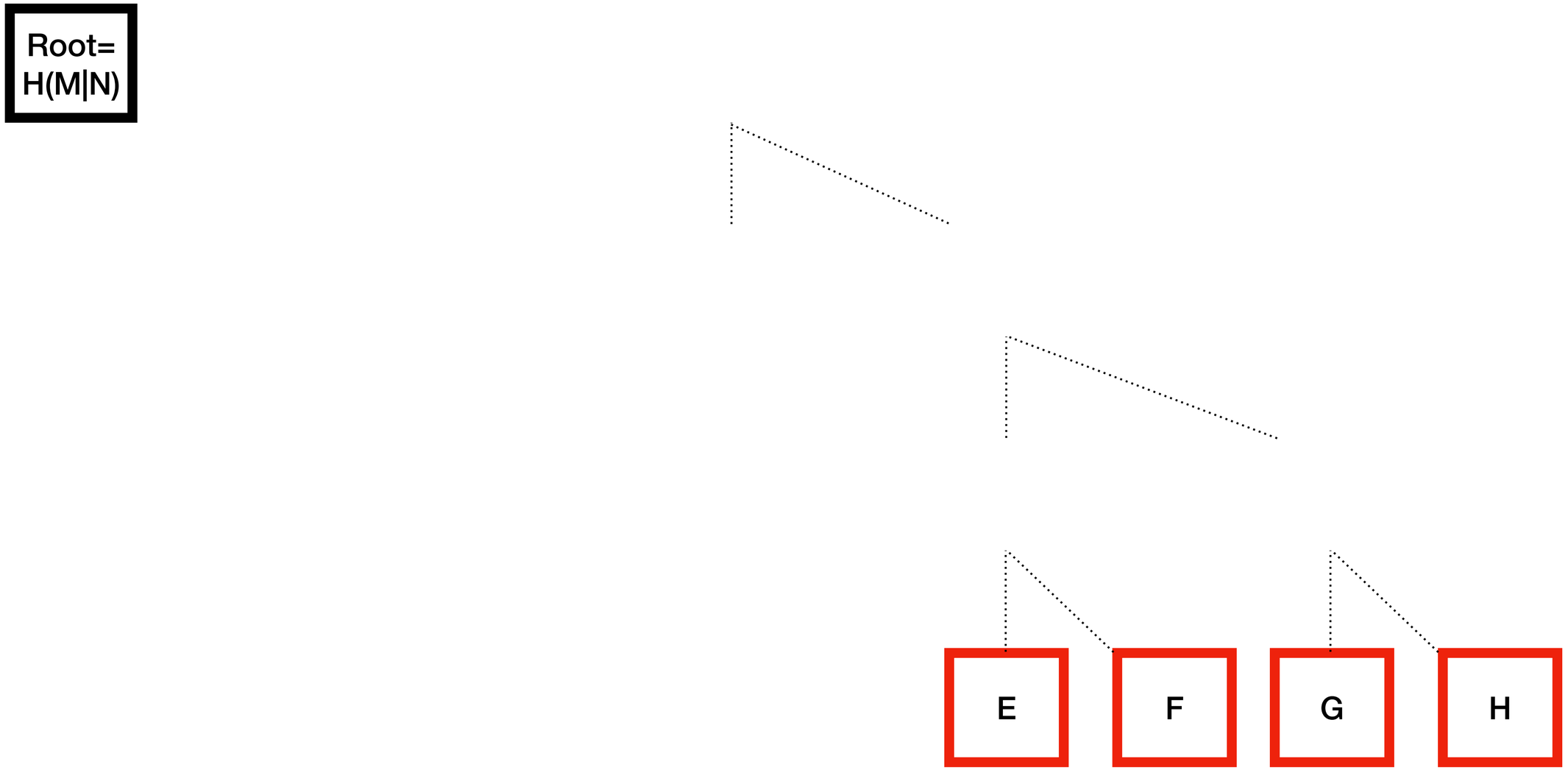
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Start off with only the root



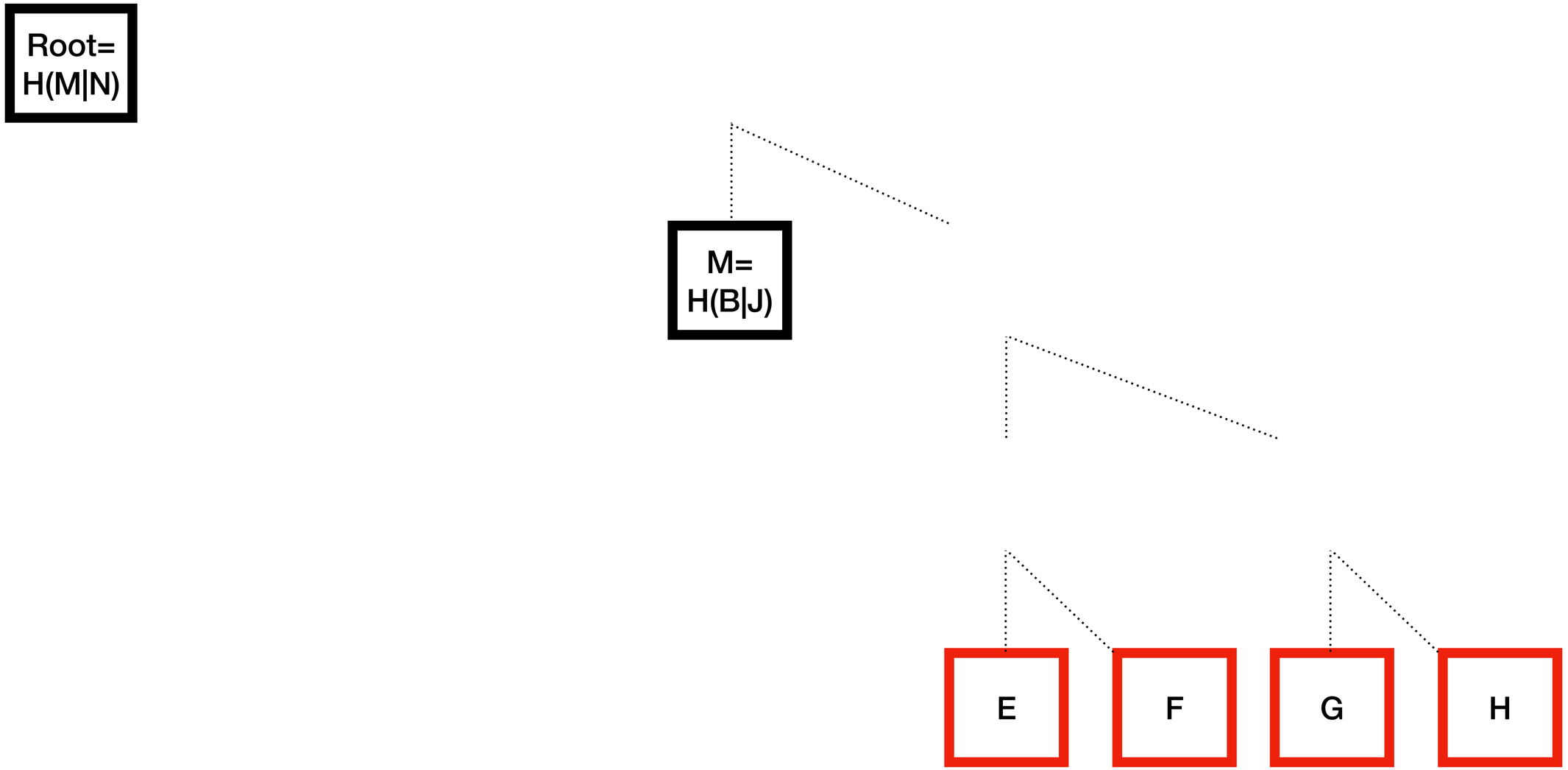
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

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



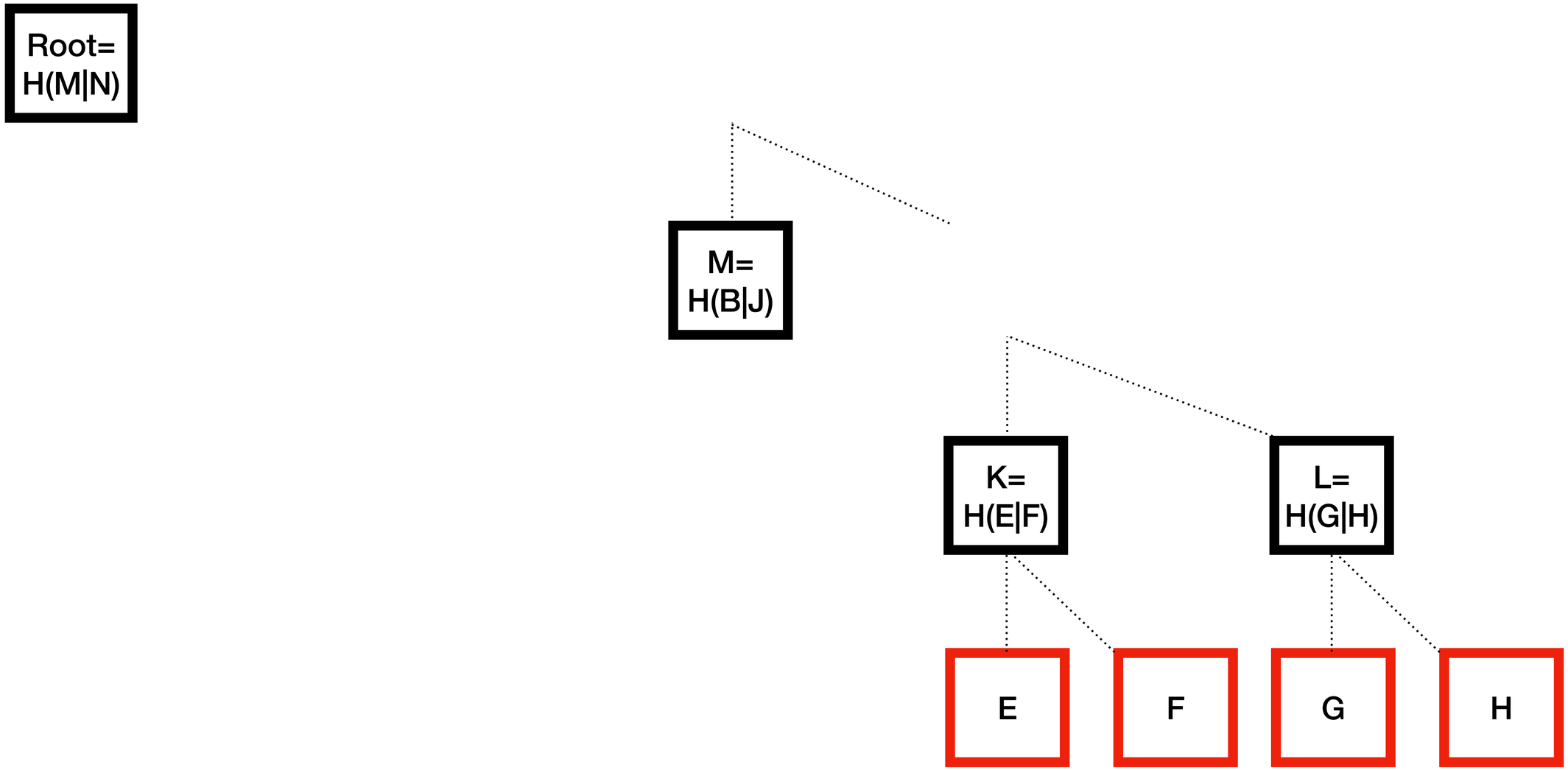
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Receive the proof: M



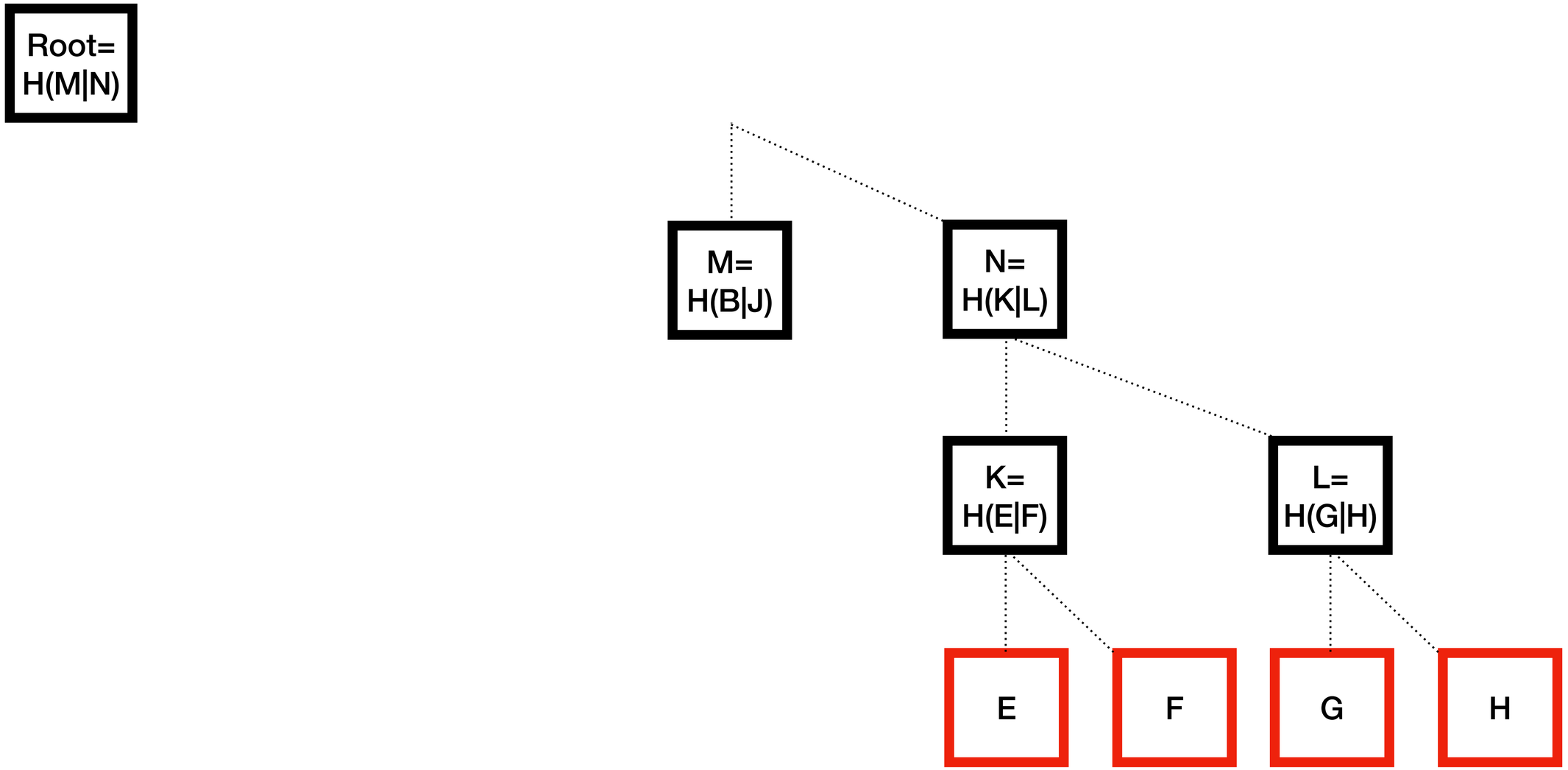
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate K and L



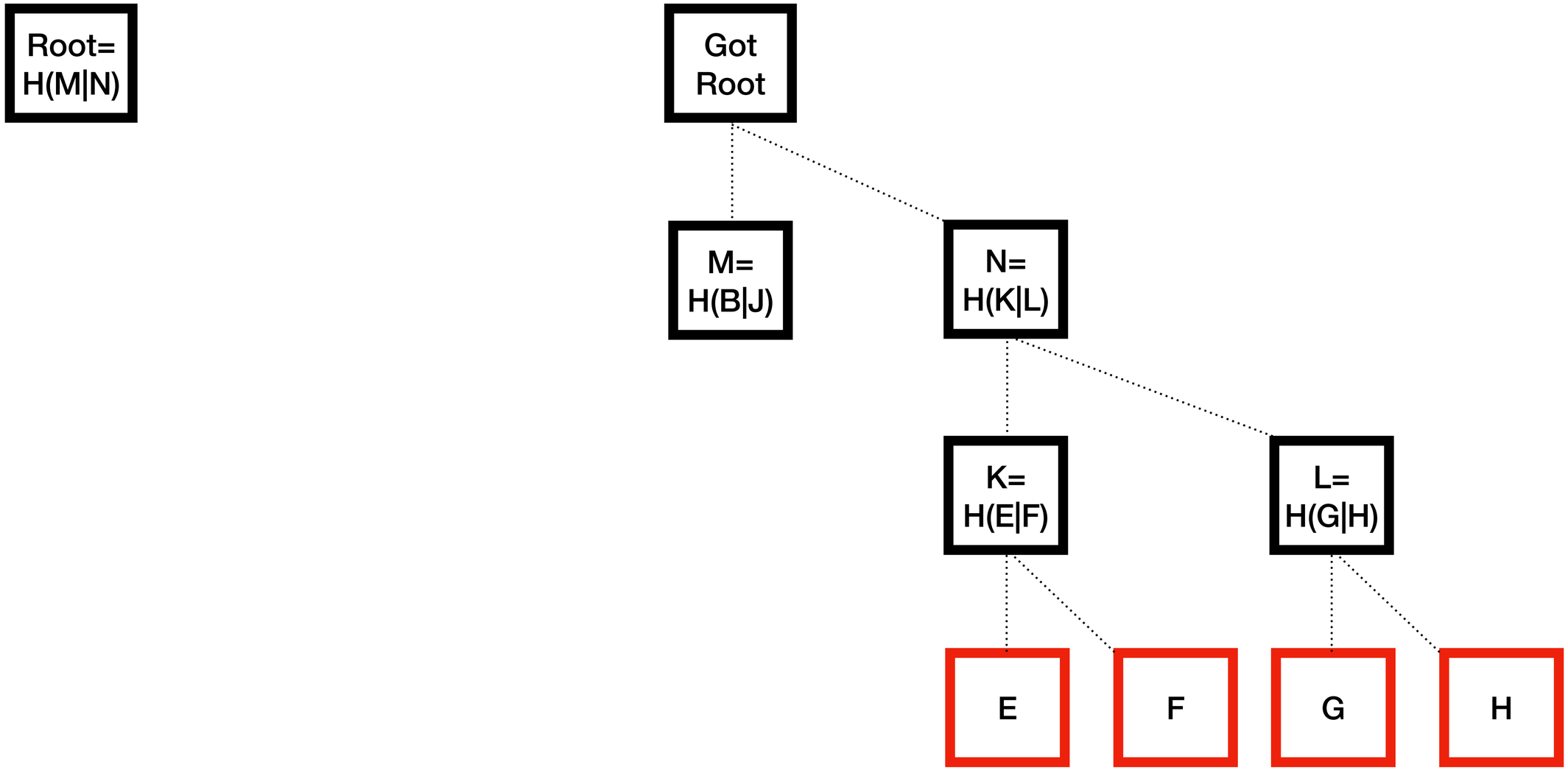
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate N



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

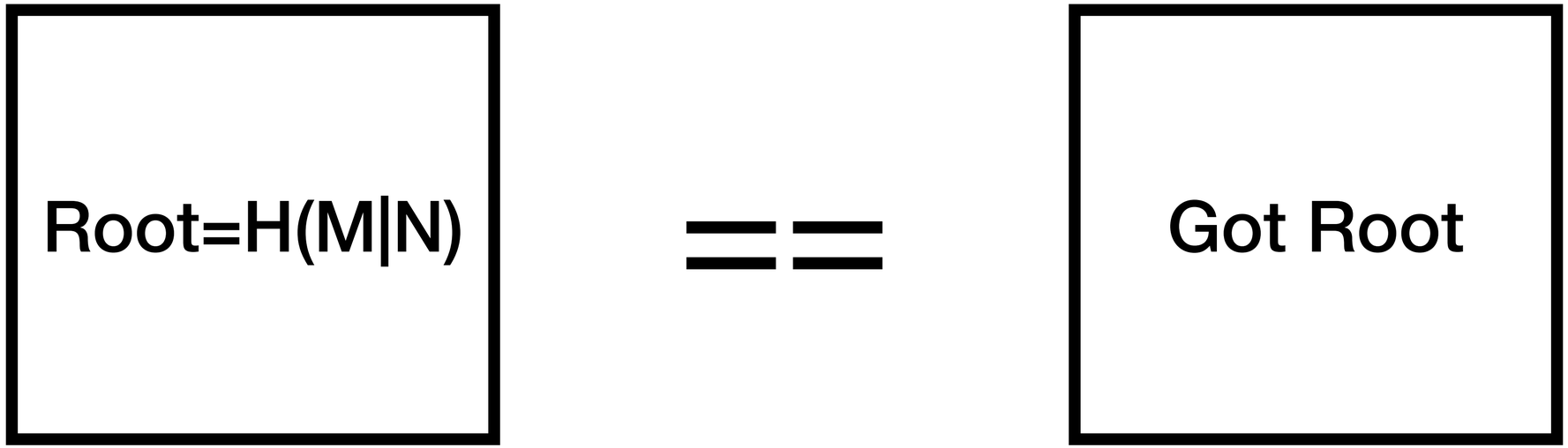
Calculate the Root



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

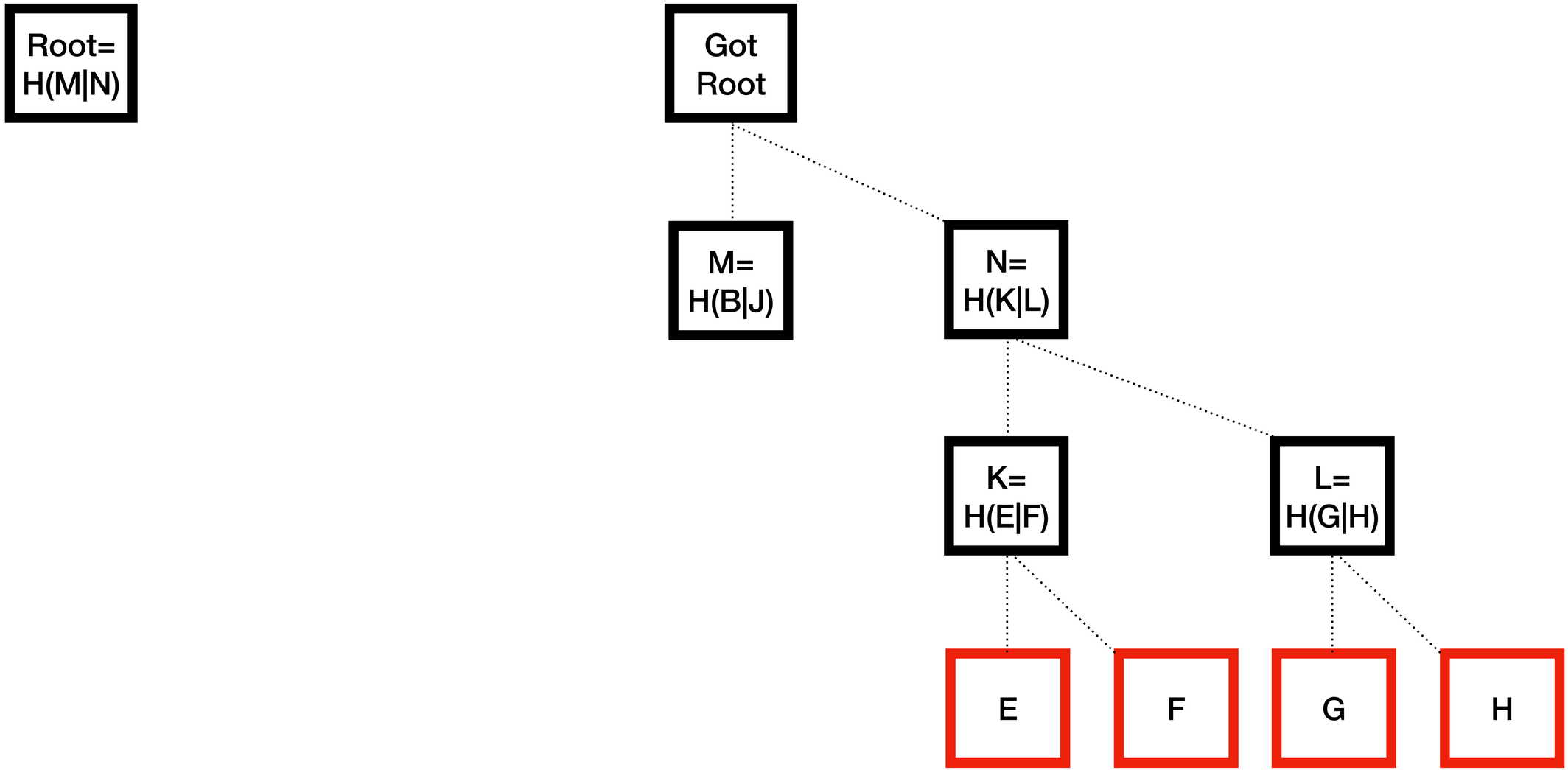
Compare roots

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



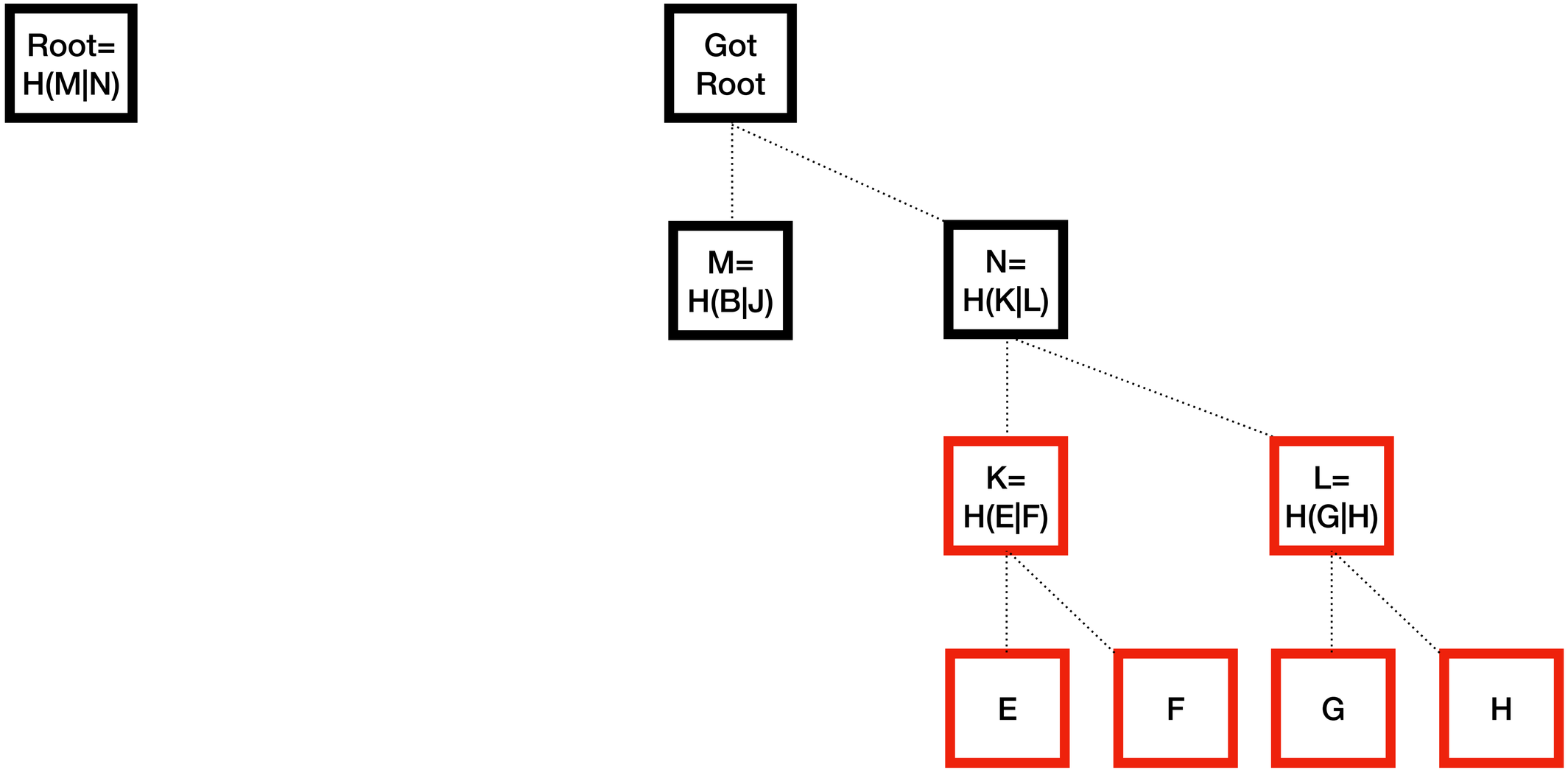
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate new root



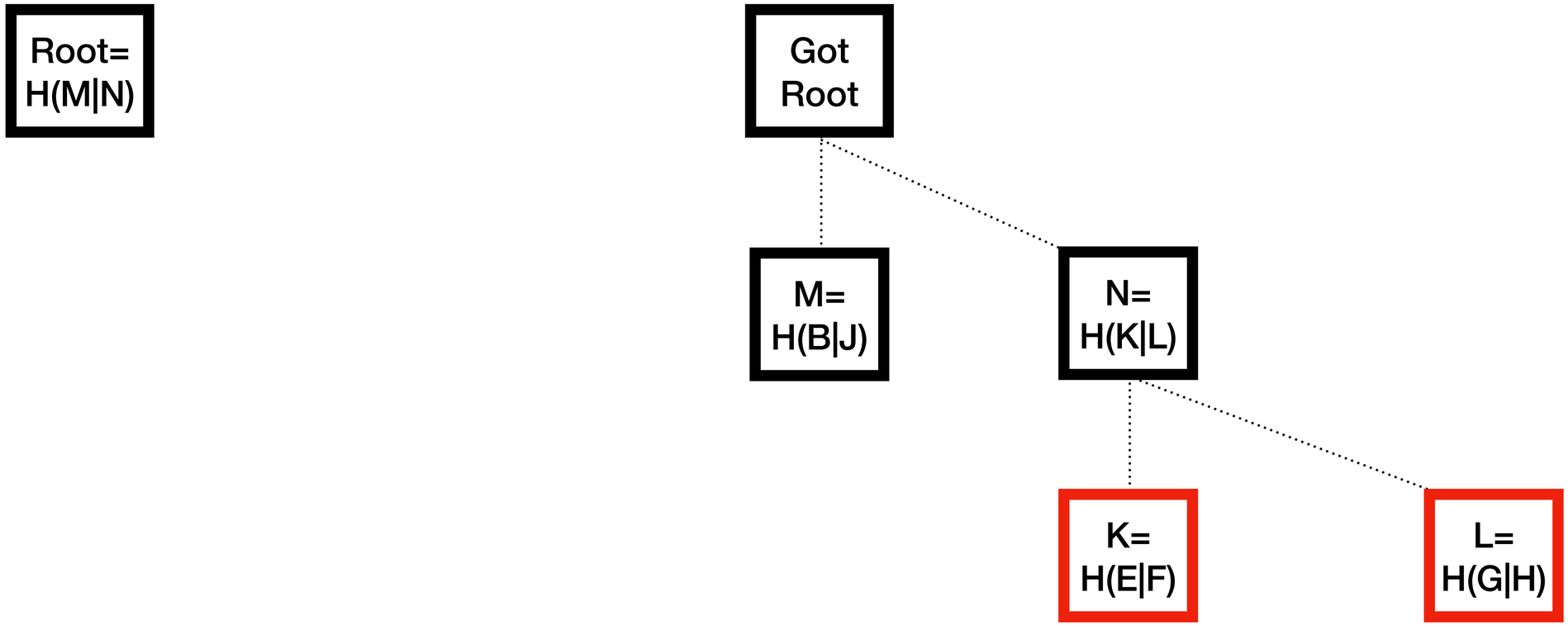
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Mark K & L as node to delete



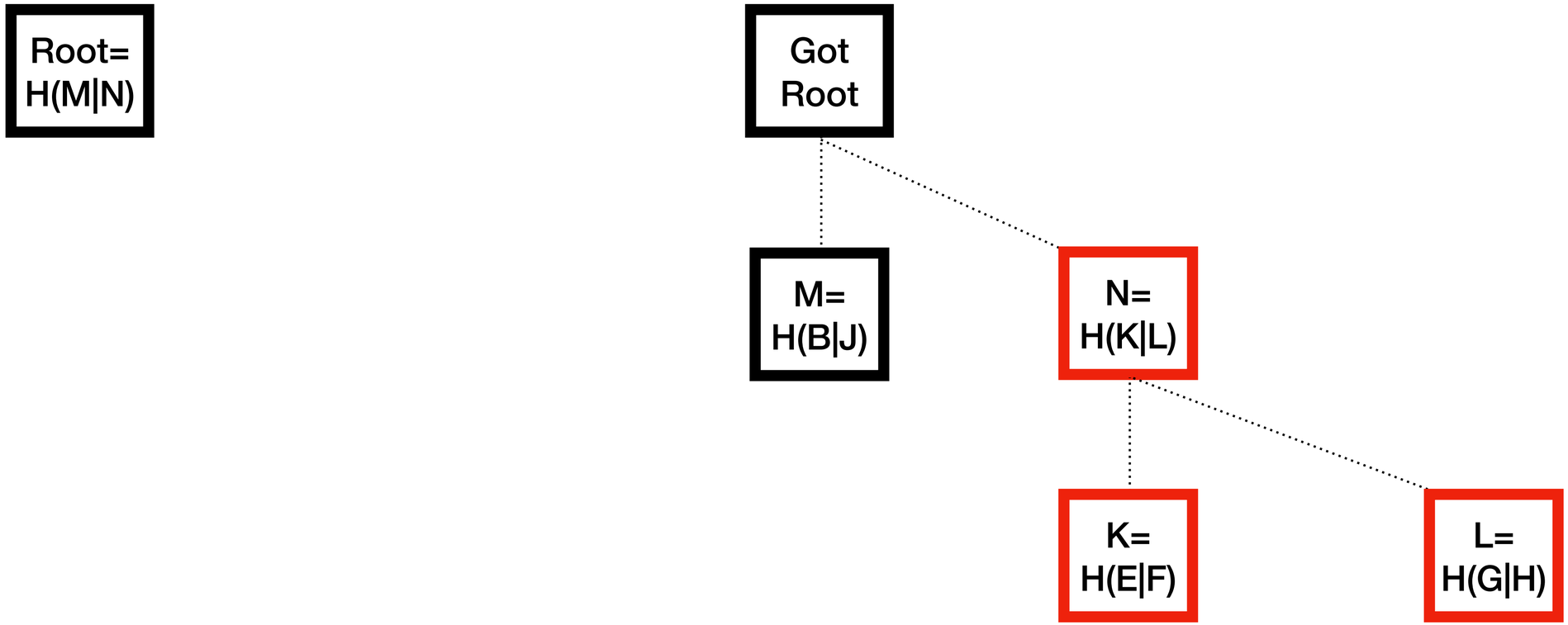
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Remove old nodes for E, F, G, H



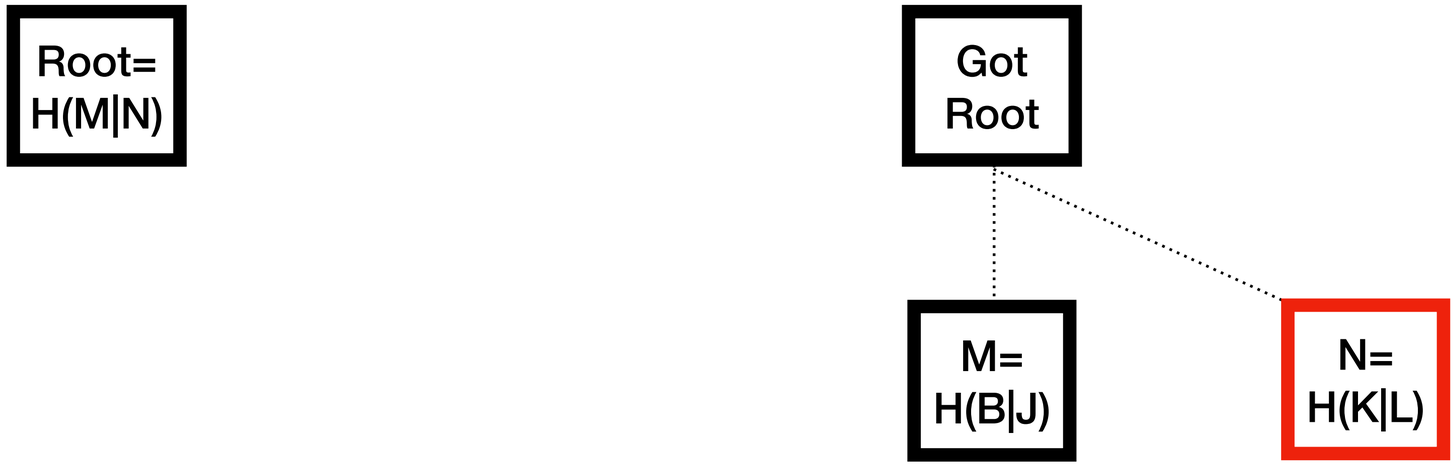
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Mark N as node to delete



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

Remove old nodes for K & L



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

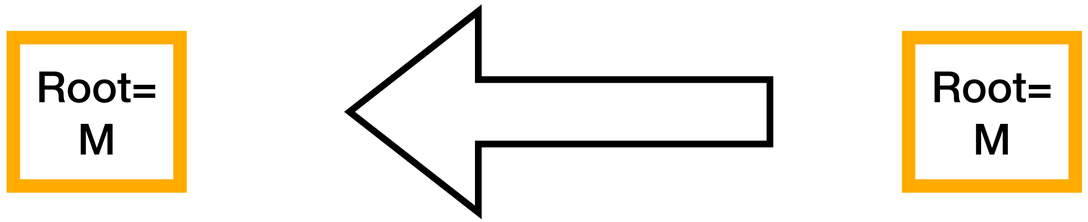
Move up M

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

Remove old nodes for M & N

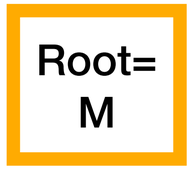
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Copy over the new root to be saved



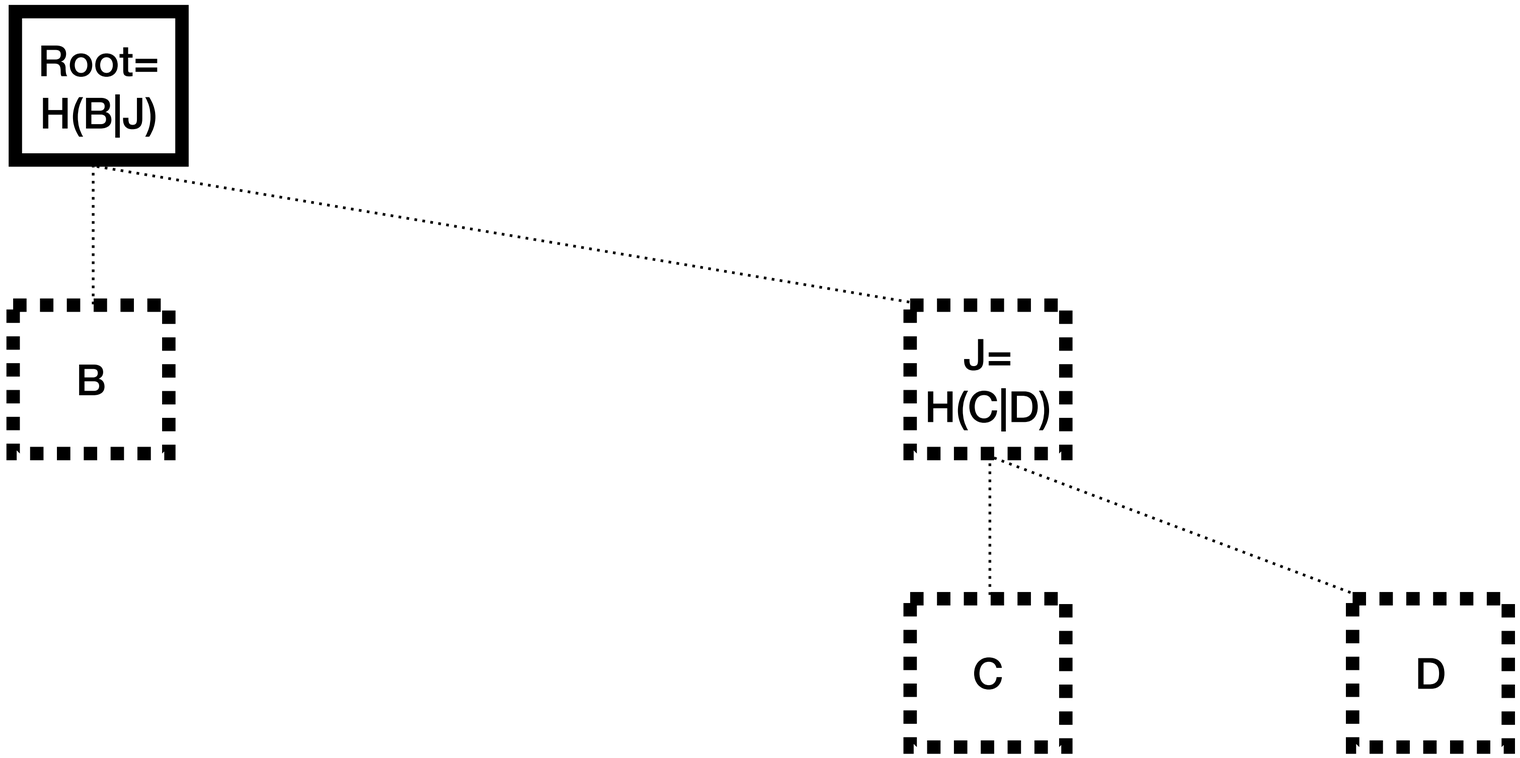
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

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

LevelDB

Fetching a UTXO

091dd74c2abc5dc5a656450288
0657037d09f56545b5f08365b6
5d21dcc19d2f:1

LevelDB

Fetching a UTXO

Amount: 291312

BlockHeight: 791794

IsCoinbase: False

PkScript: 0014703a38f47197ede65d38c3f79f60093f956d6e43

No LevelDB 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!

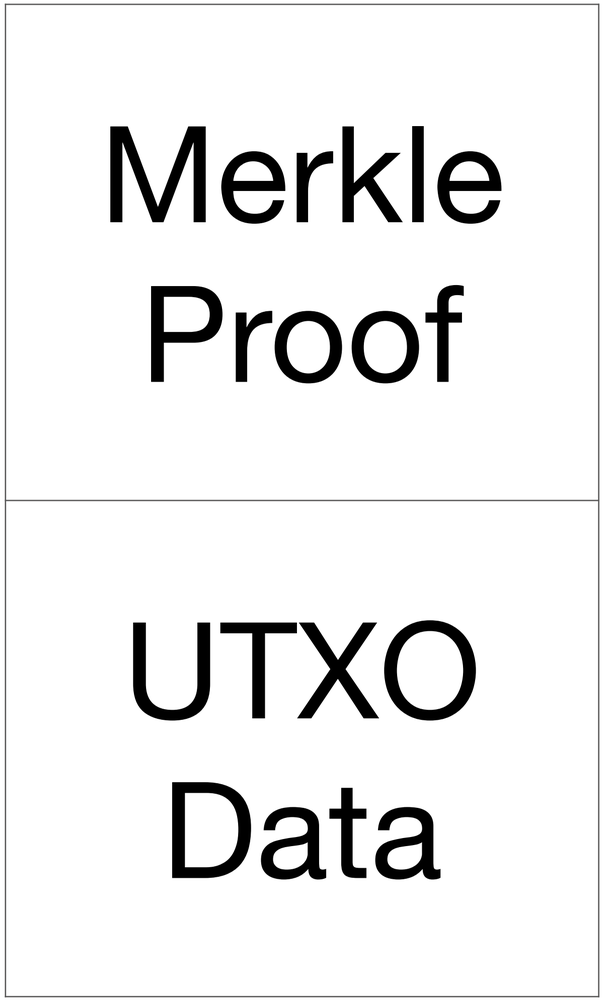
Version
Flag
TxIn Count
TxIns
TxOut Count
TxOuts
Witness
Locktime
Utreexo Proof Data

TX serialization for Utreexo nodes

Not finalized!

Utreexo Proof Data

Not finalized!



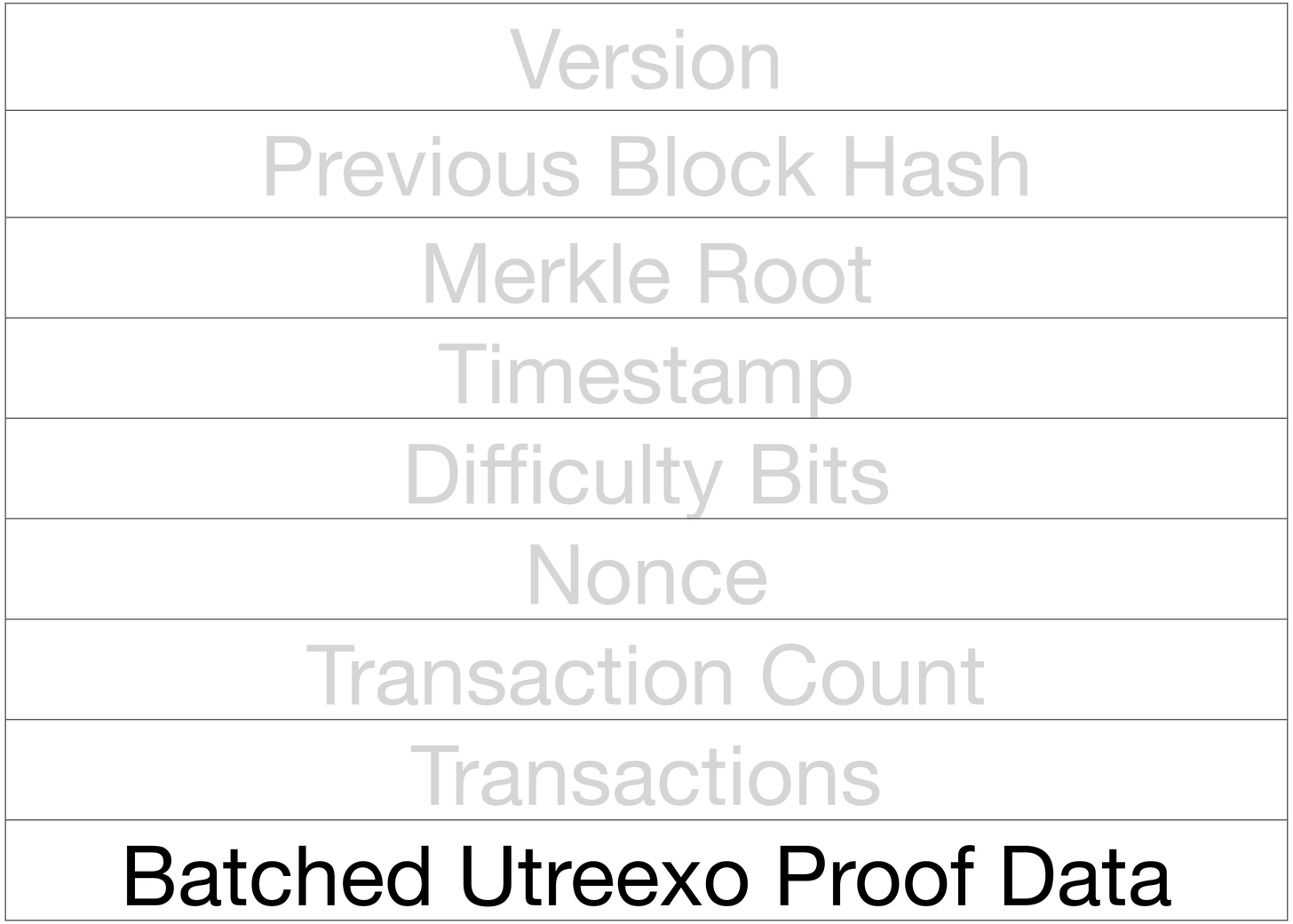
Block serialization for Utreexo nodes

Not finalized!

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

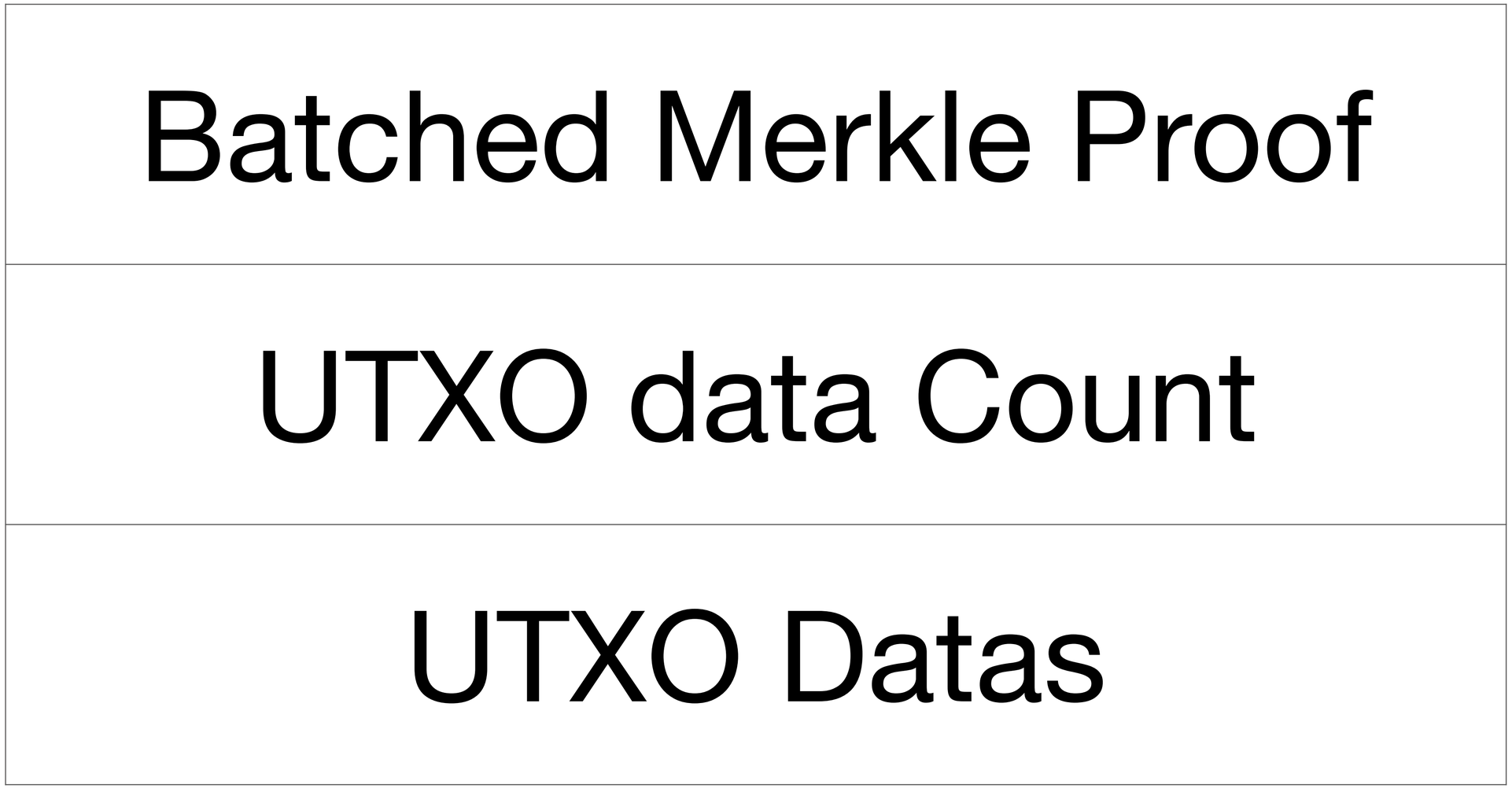
Block serialization for Utreexo nodes

Not finalized!



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

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

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

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

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

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

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

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

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

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

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

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

UTXO Data serialization

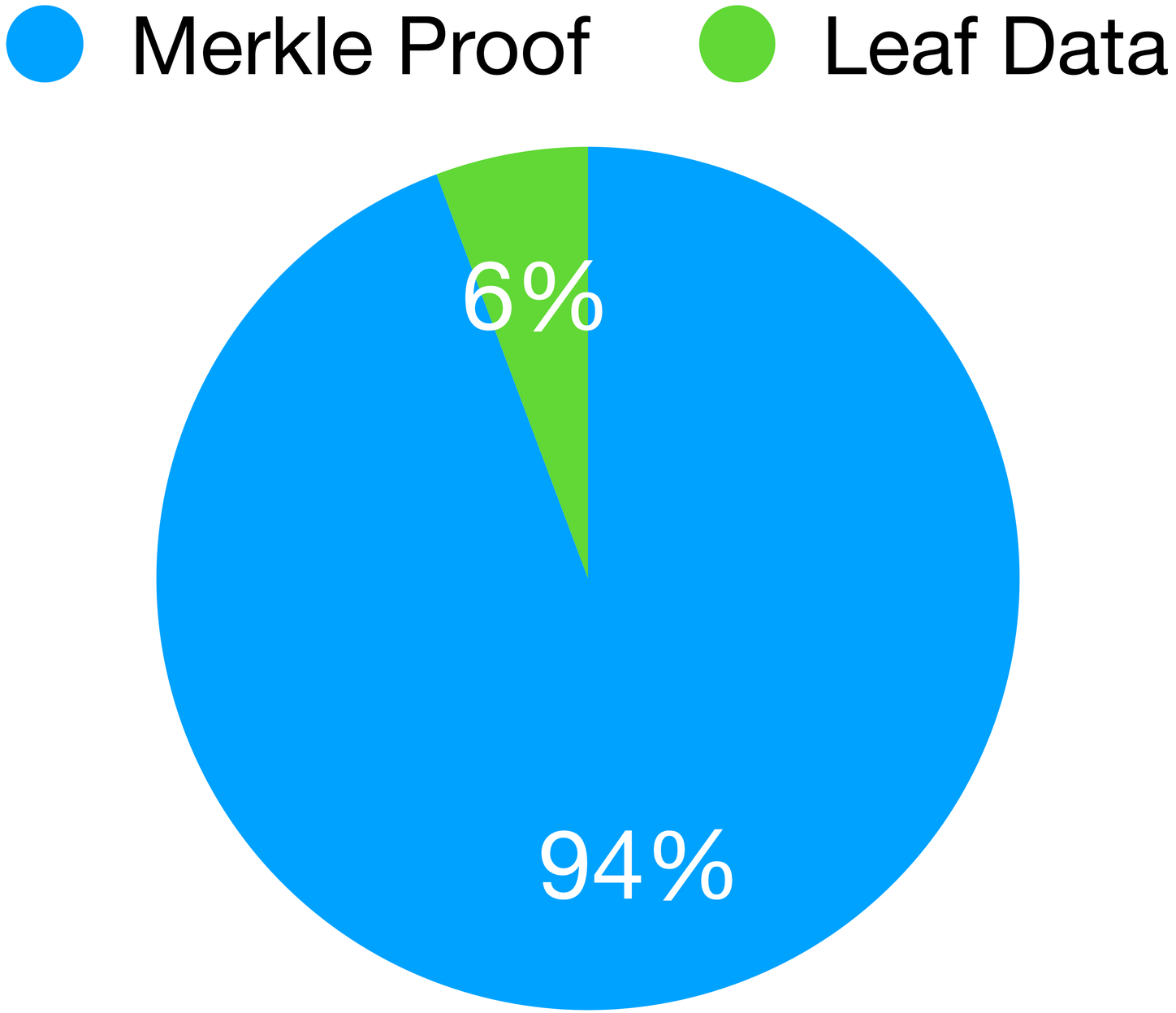
For disk storage/p2p transfer

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

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

Twitter:
@kcalvinalvinn

Slides:
github.com/kcalvinalvin/
slides-for-btcprague