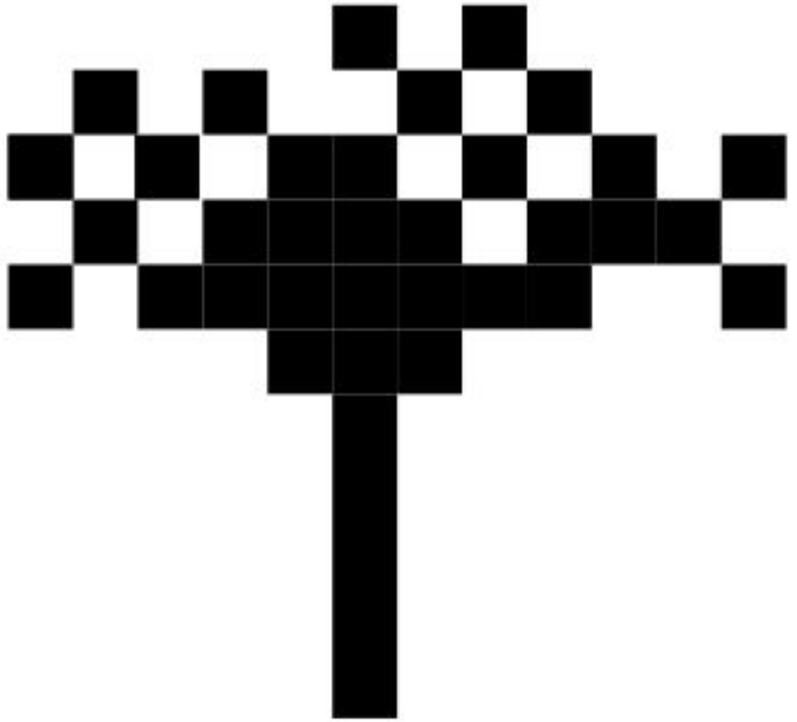


Utreexo - What is it?

Quick intro

Agenda for today

Shill & explain

- Smaller full nodes
- Performance wins
- The catch
- Deployment
- Aligning incentives
- How it works

Smaller full nodes

```
[I] calvin@Calvins-MacBook-Pro ~/L/A/Bitcoin> du -h .
111M    ./blocks/index
744M    ./blocks
7.4G    ./chainstate
 88K    ./coldcardmk4
104K    ./coldcardmk3
8.2G    .
```

```
[I] calvin@Calvins-MacBook-Pro ~/L/A/Bitcoin> du -h .
111M    ./blocks/index
744M    ./blocks
7.4G    ./chainstate
 88K    ./coldcardmk4
104K    ./coldcardmk3
8.2G    .
```

```
[I] calvin@Calvins-MacBook-Pro ~/L/A/Bitcoin> du -h .
111M    ./blocks/index
744M    ./blocks
7.4G    ./chainstate
 88K    ./coldcardmk4
104K    ./coldcardmk3
8.2G    .
```

UTXO set lives in chainstate/

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

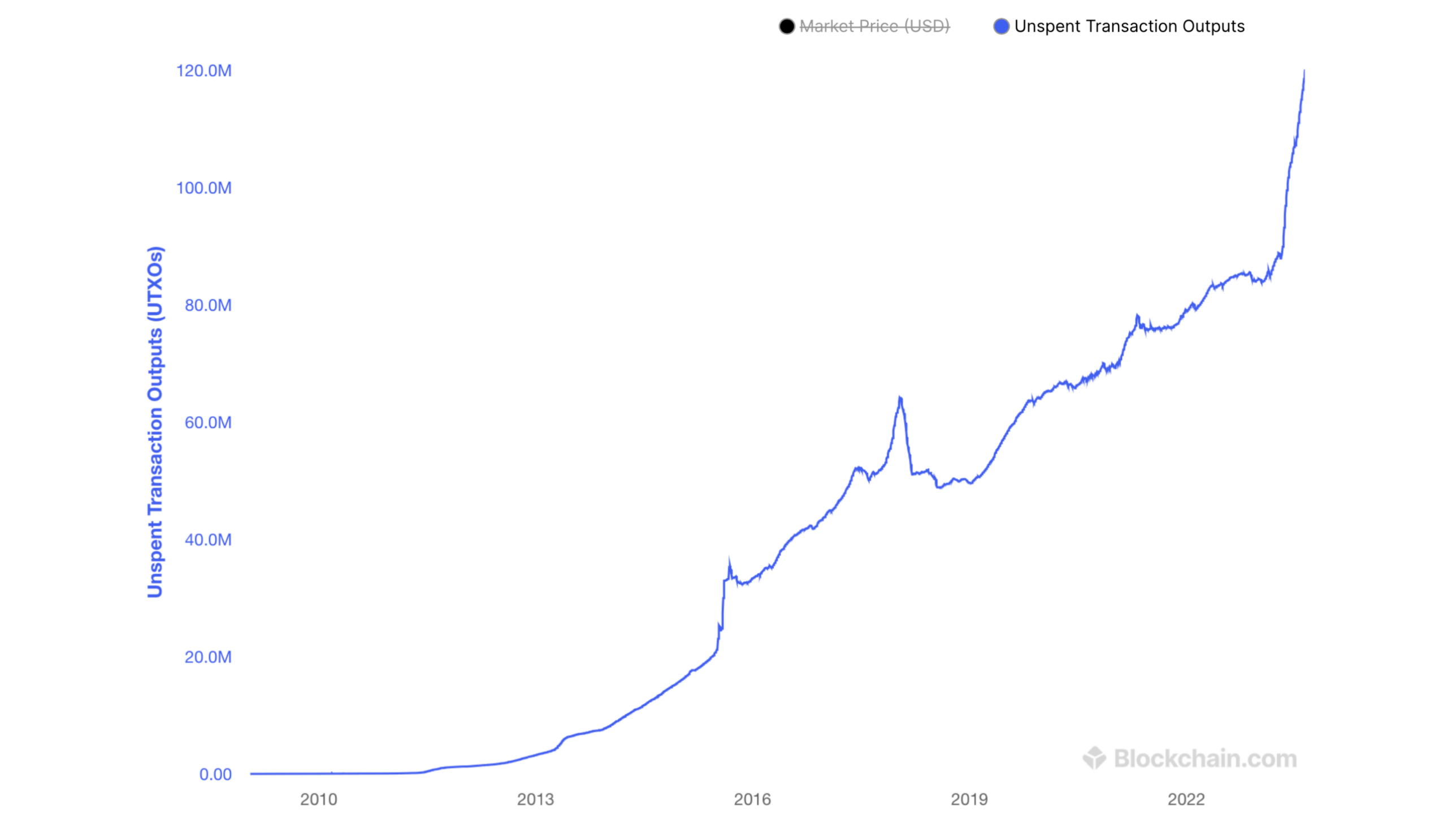
- TXO - An Output of a TX (transaction)
- UTXO - Unspent TXO

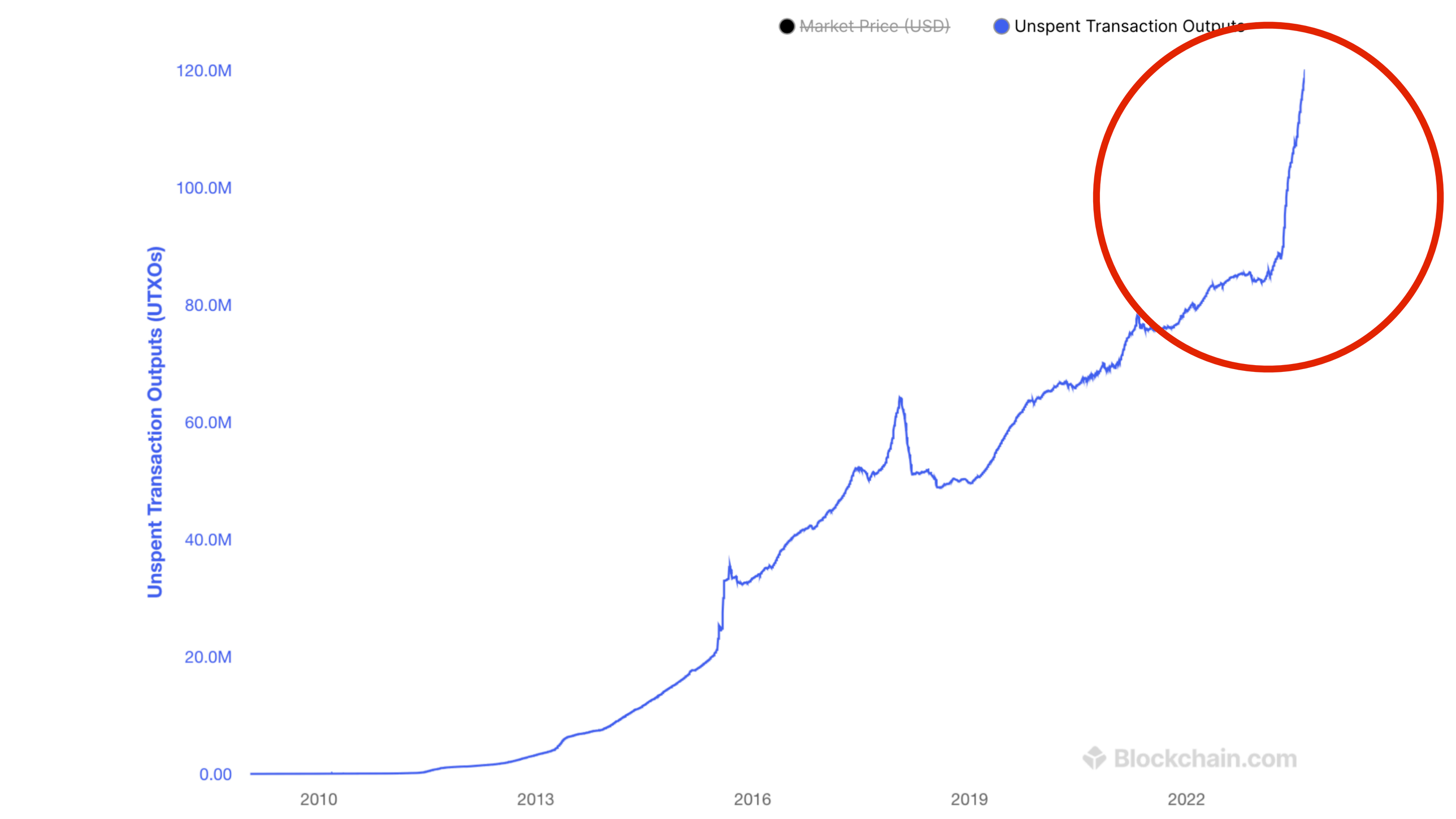
The UTXO set

Quick overview

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

UTXO set is unprunable


Market Price (USD)
Unspent Transaction Outputs
120.0M
100.0M
80.0M
60.0M
40.0M
20.0M
0.00
Unspent Transaction Outputs (UTXOs)
2010
2013
2016
2019
2022
Blockchain.com


Market Price (USD)
Unspent Transaction Outputs
Unspent Transaction Outputs (UTXOs)
120.0M
100.0M
80.0M
60.0M
40.0M
20.0M
0.00
2010
2013
2016
2019
2022
Blockchain.com

Turn the UTXO set into a merkle tree

Turn the UTXO set into a *fancy merkle tree*

Merkle
trees

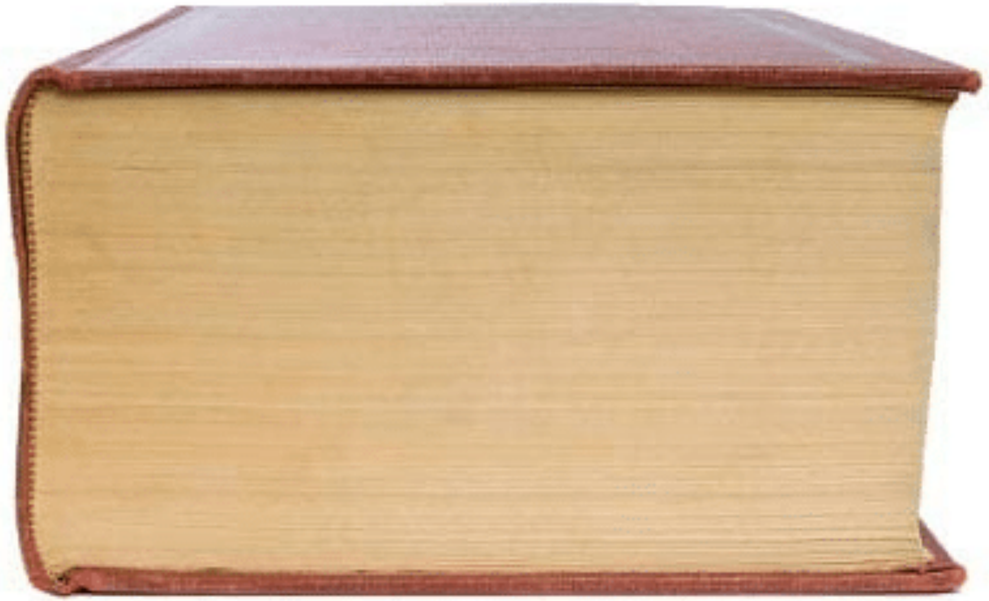
Utreexo

The UTXO set

Why merklize it?

- Puts a bound to the UTXO set growth

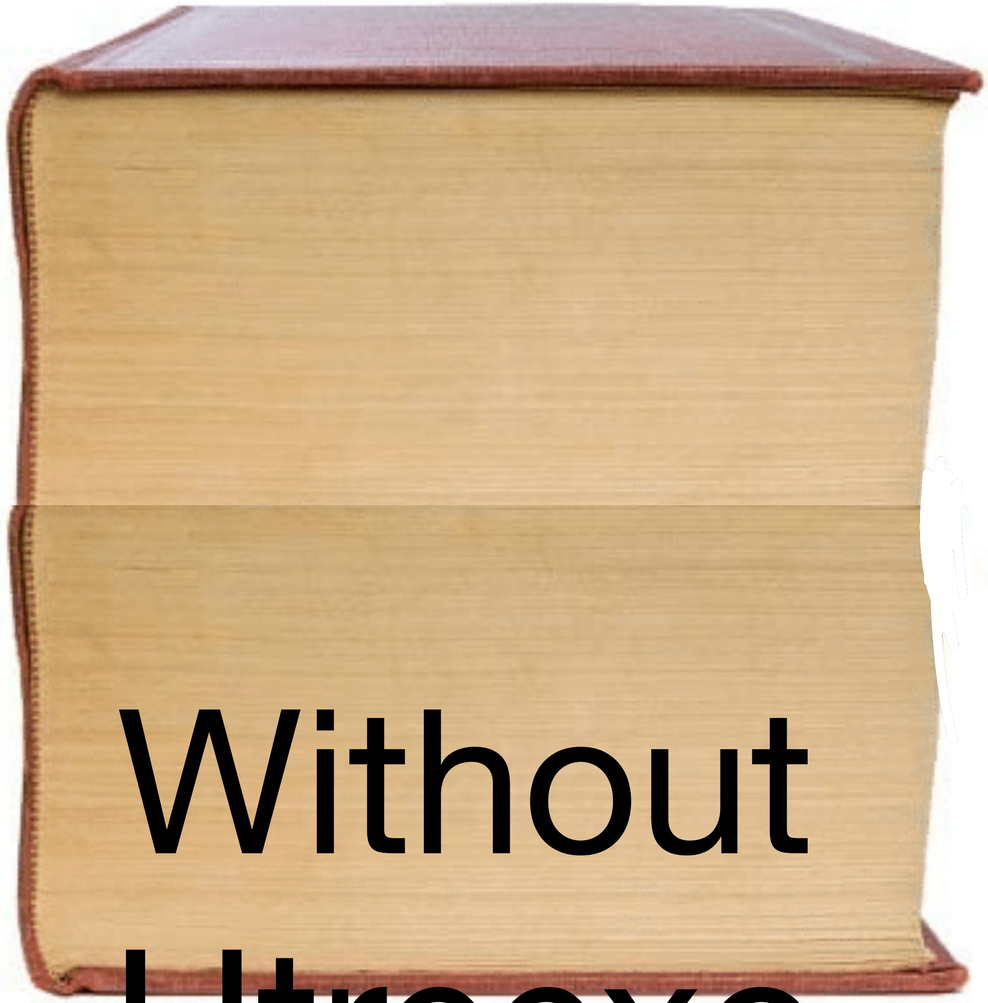
Pruned nodes



Without Utreexo

With Utreexo

Pruned nodes



Without Utreexo

With Utreexo

Pruned nodes

Without Utreexo

With Utreexo

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

Performance wins

~Same Performance on HDD or SSD

Avoid disk i/o

No more random reads

Prove it?

Test methodology

- Compare performance degradation of pruned nodes on HDD vs SSD

36% slower

Current nodes

https://blog.bitmex.com/progress-towards-utreexo-goals/

3% slower

Utreexo nodes

https://blog.bitmex.com/progress-towards-utreexo-goals/

What we flush for utreexo

No random i/o

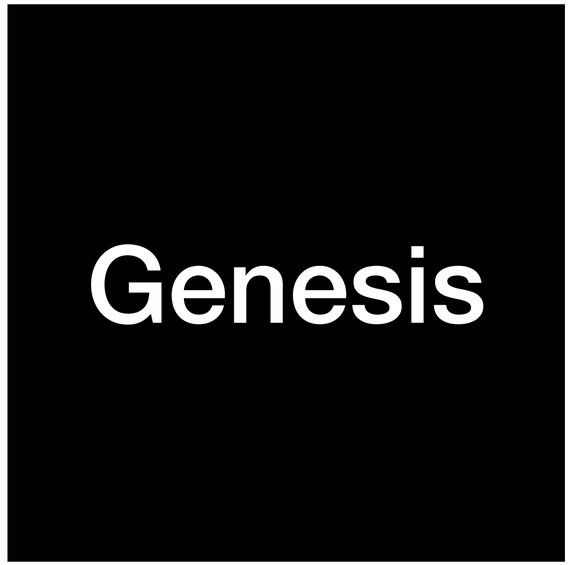
```
{443000, newHashFromStr("0000000000000000047efc3c126c3398cfdef1609682d1492cd2909e3c0a17f"),
        43243051, []*chainhash.Hash{
                newLeafHashFromStr("df31c496a54a8021c38c77809da15a8bd3968970315f5a56c07f0fd9763262ff"),
                newLeafHashFromStr("755f4cfbf5727c3e14aaf6dcc6e5ab150c61101cead06fade5a886ee8d1dd377"),
                newLeafHashFromStr("c572a92c7cd9e14ce5d7c94473ef90bee21856654eed5535314b3c117006bc83"),
                newLeafHashFromStr("9e85567347ef0ea38c1797d8ed8e399ed0d99a07025388f6e06791ff6b88643c"),
                newLeafHashFromStr("ab4efc738bd7b8281689e899979d44ed0ca8fe1489efa0f5a09f5b1c276fe792"),
                newLeafHashFromStr("1c763d8f490baef0b756597dfc2da5a31d0aee324da6314b52f21707351cf51a"),
                newLeafHashFromStr("29fcd9989939a17e5d113fb1f247f3bd05606488bcb48c9cfad55a57006f2248"),
                newLeafHashFromStr("e79f840fe1bcdbb6b638a53b637885dda7ea3da2b2b5f1f5447ebf4d54fc0a2e"),
                newLeafHashFromStr("02f33c3ce0d684e644410fa35fee4c7572454facc710389ea0f69a00e27c1780"),
                newLeafHashFromStr("4e29ad940088ab42991bd99d7943a68ea38d498f8435c4bea21b8252788b5a7d"),
                newLeafHashFromStr("feabfc6565e918827df8ade12d0885e7da28bb81410c71a0fcbd92da28e5538e"),
                newLeafHashFromStr("42275481edcd150c0891f8f7486941e7ecc9dbec0d1554d1ae3027d8523571b4"),
                newLeafHashFromStr("84dd84bbb564ab9e878ea774d66a2cb818c2b74680974e01a07dde68bd064bfd"),
                newLeafHashFromStr("19a64f2f77403d4a1e3482b3a1801aaf360120f6429f5ed47261fc8676920d9d"),
        },
},
```

https://github.com/mit-dci/utcd/blob/7310b99094acbeb11ecda230475d127af064c82e/chaincfg/mainnetroots.go#L6665-L6682

Parallel Block validation

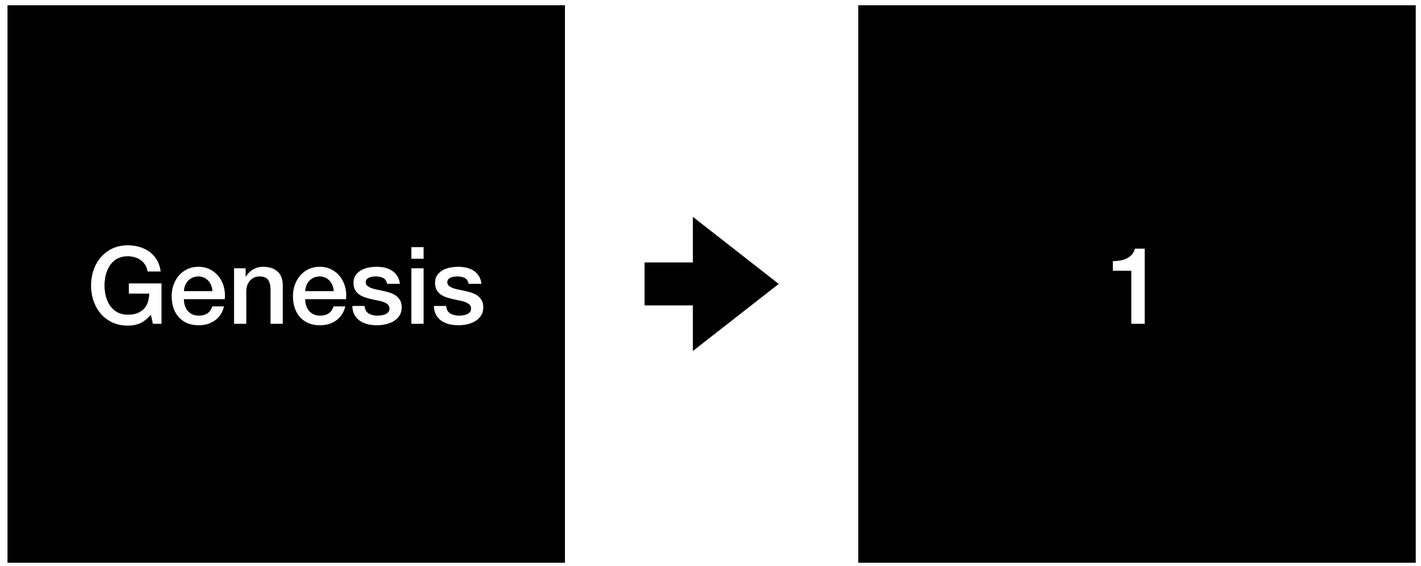
Current block validation

Sequential validation



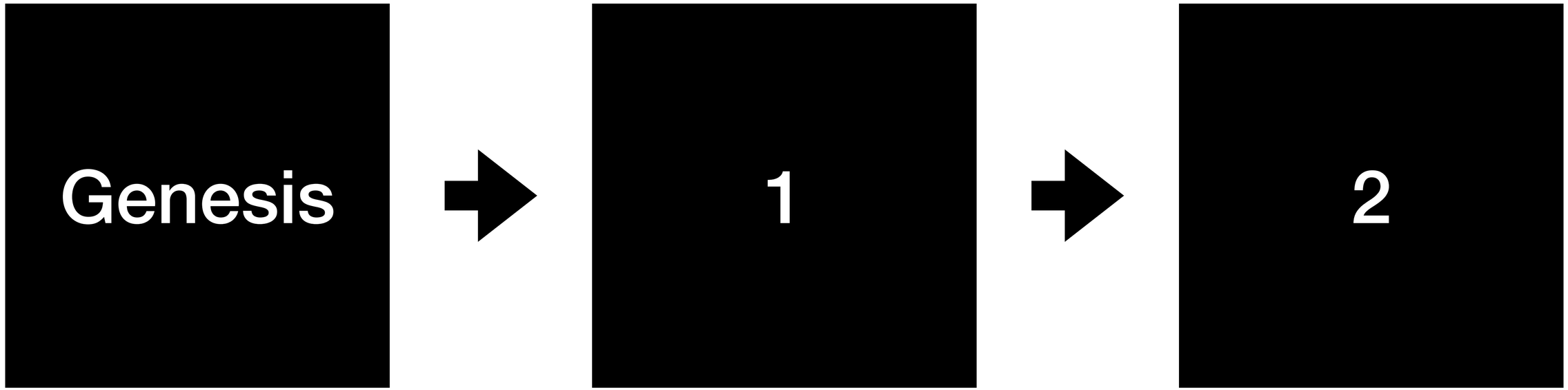
Current block validation

Sequential validation



Current block validation

Sequential validation



Parallel block validation

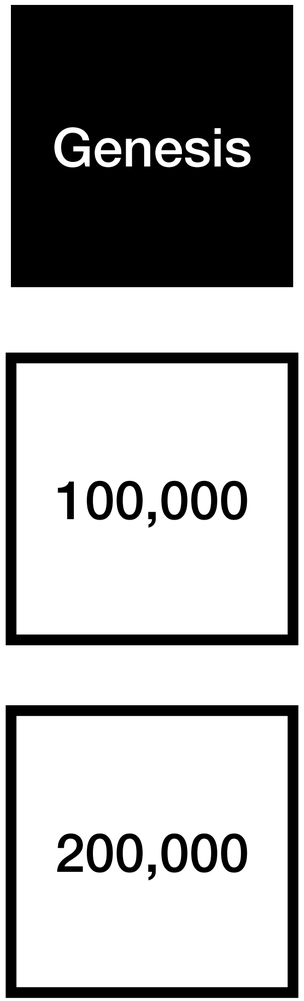
Replay blocks in any order

- 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

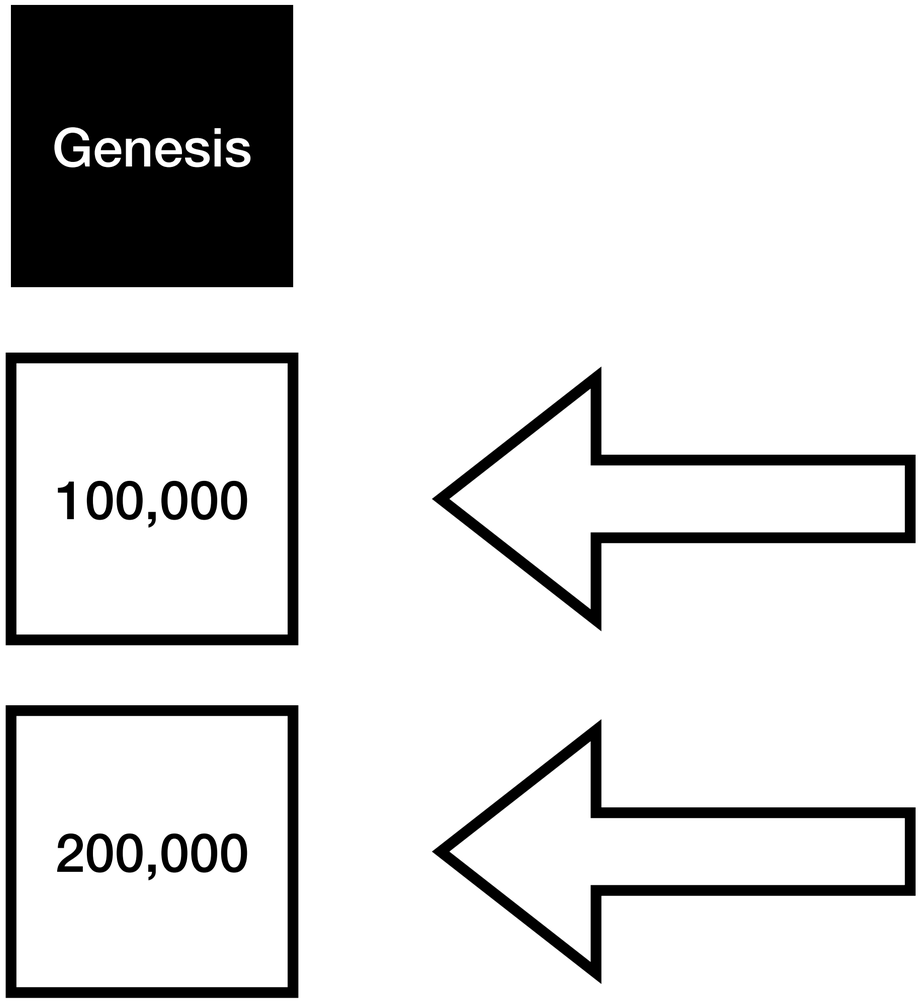
With Utreexo

Efficient parallel validation



With Utreexo

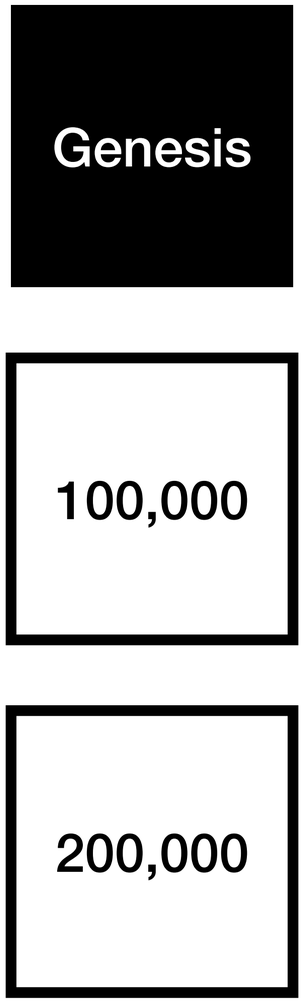
Efficient parallel validation



Untrusted roots at the start

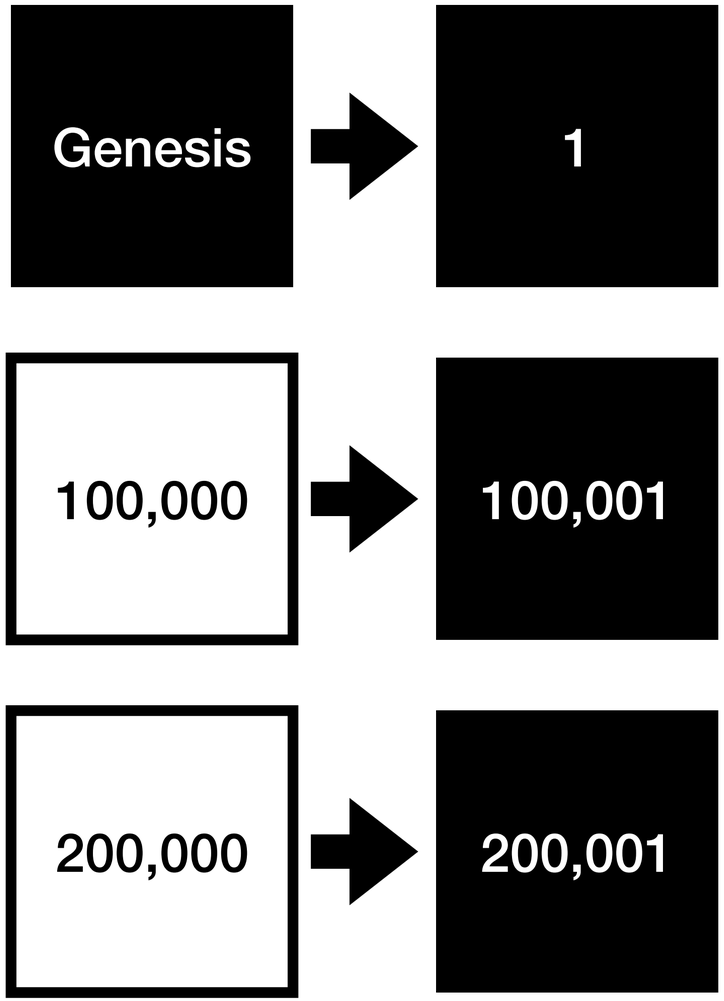
With Utreexo

Efficient parallel validation



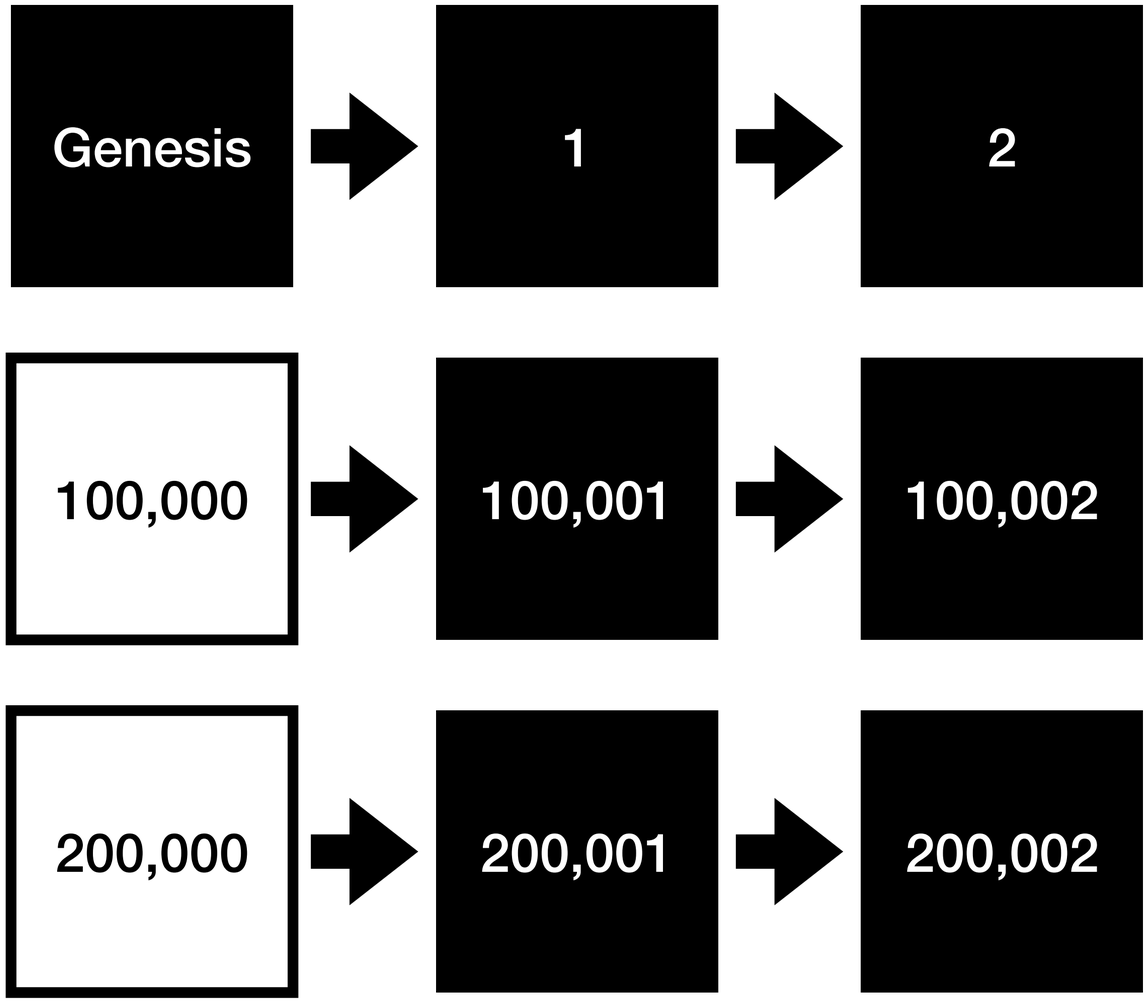
With Utreexo

Efficient parallel validation



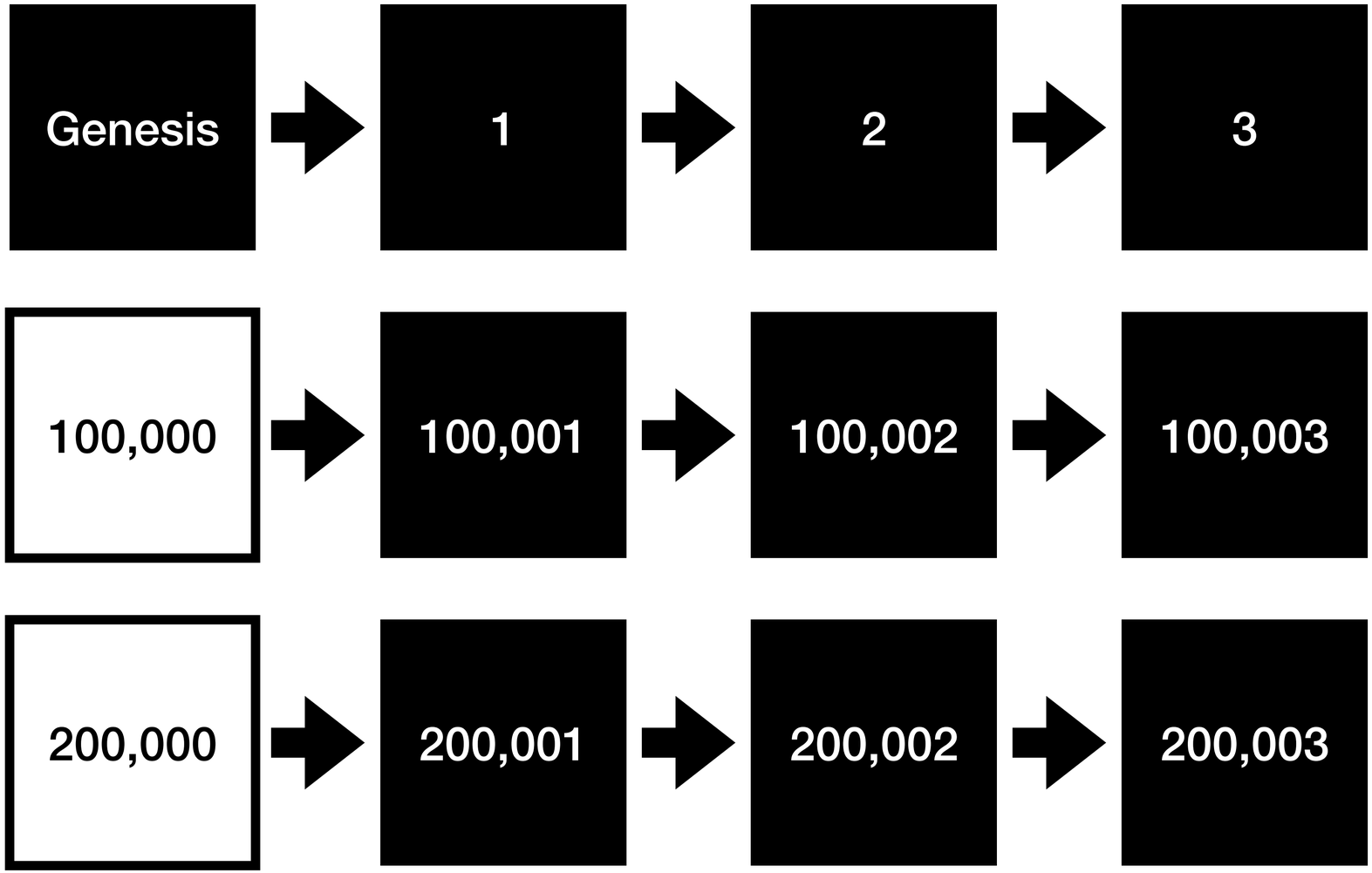
With Utreexo

Efficient parallel validation



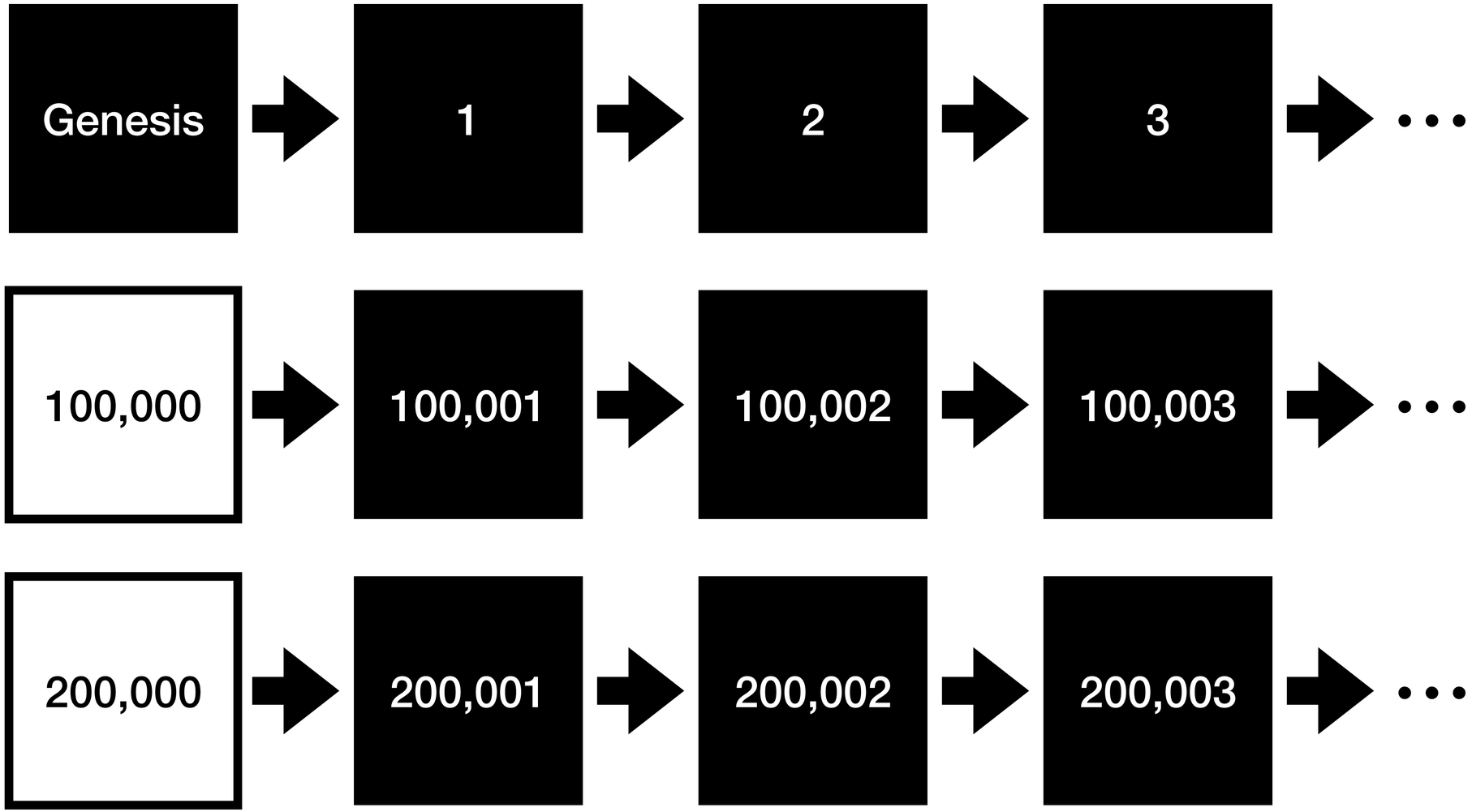
With Utreexo

Efficient parallel validation



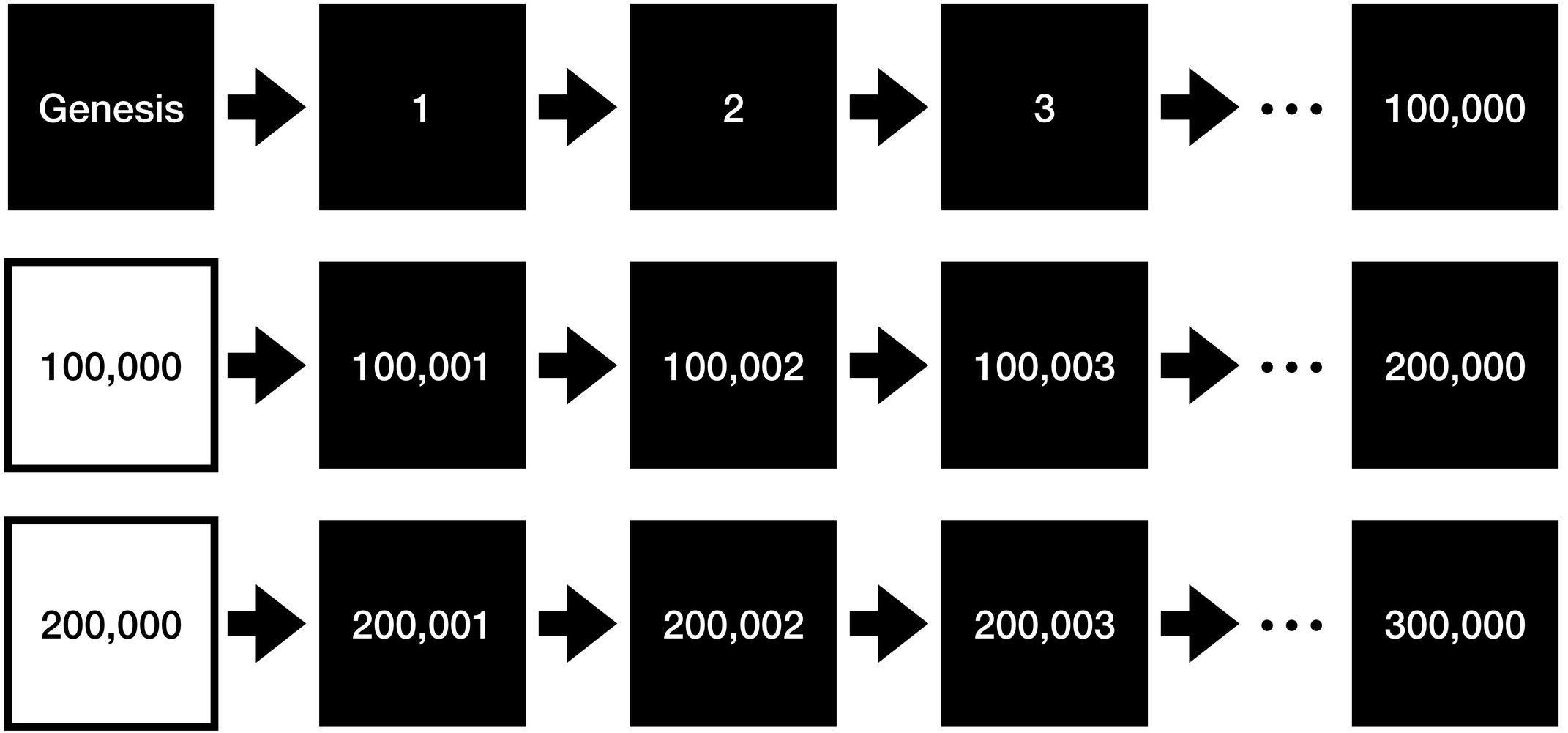
With Utreexo

Efficient parallel validation



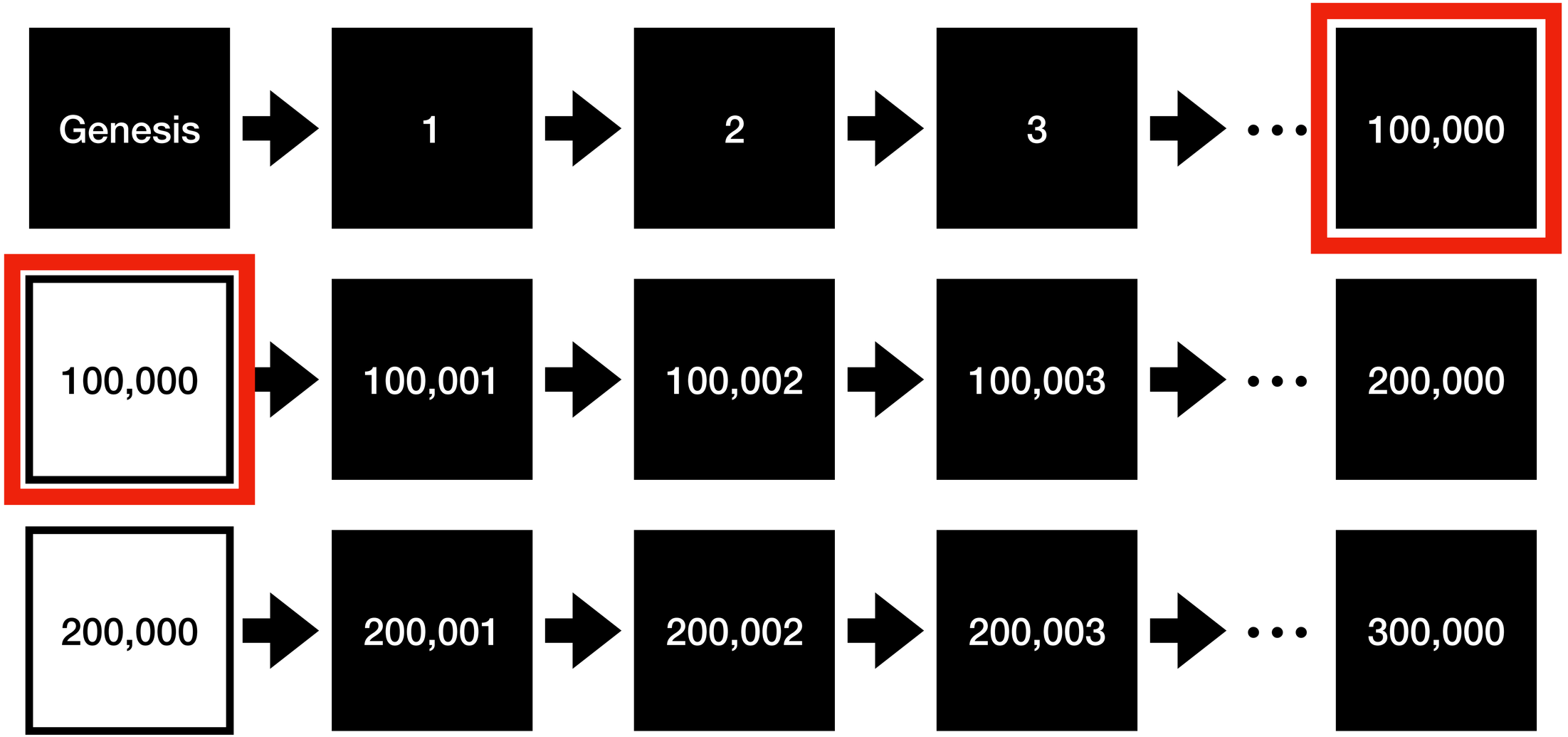
With Utreexo

Efficient parallel validation



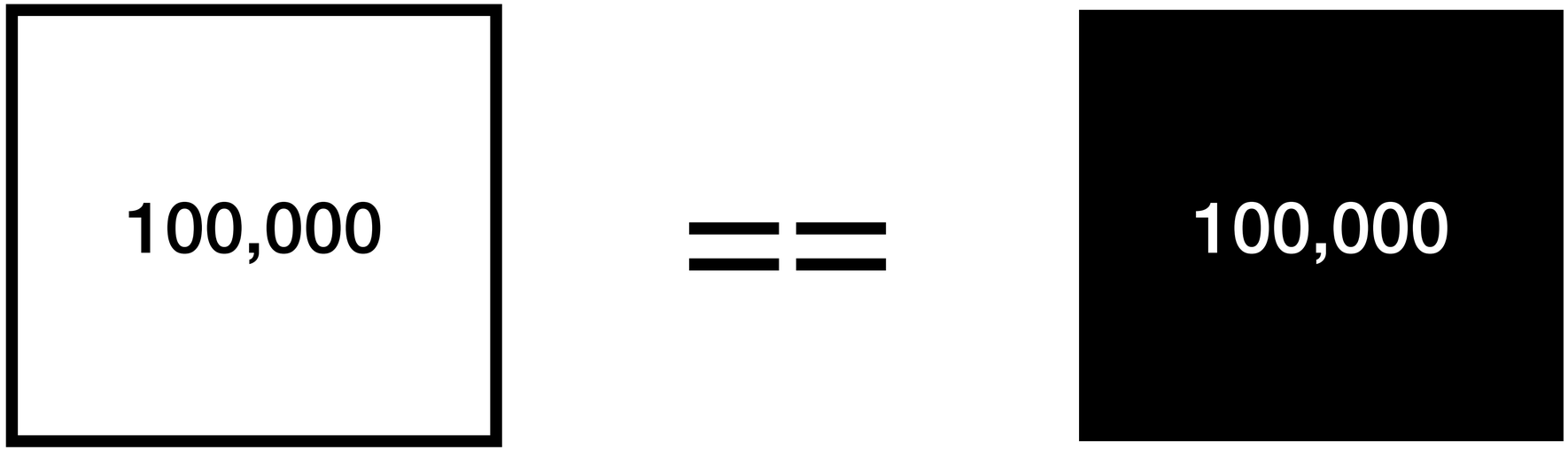
With Utreexo

Efficient parallel validation



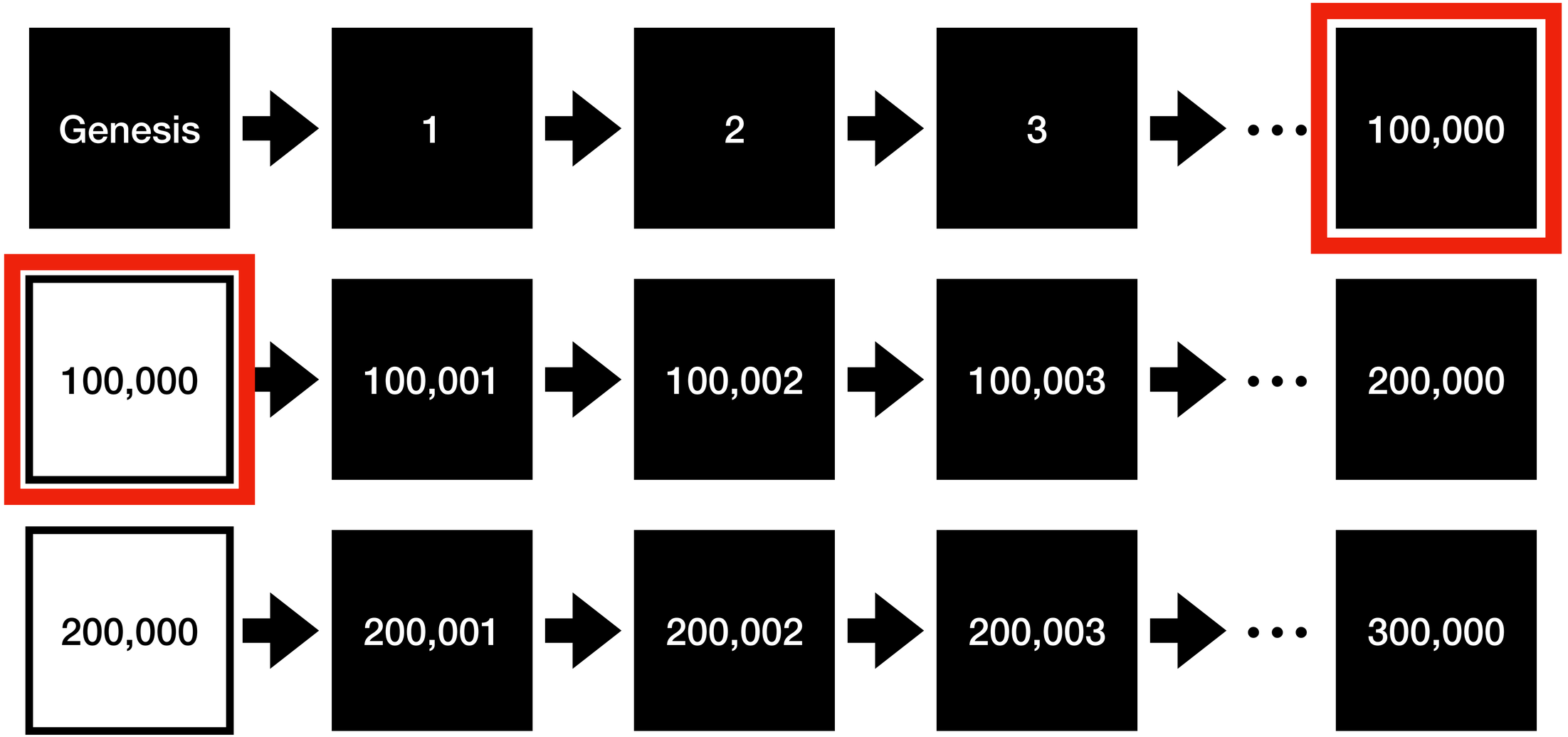
Compare roots

Root at 100,000 becomes trusted if equal



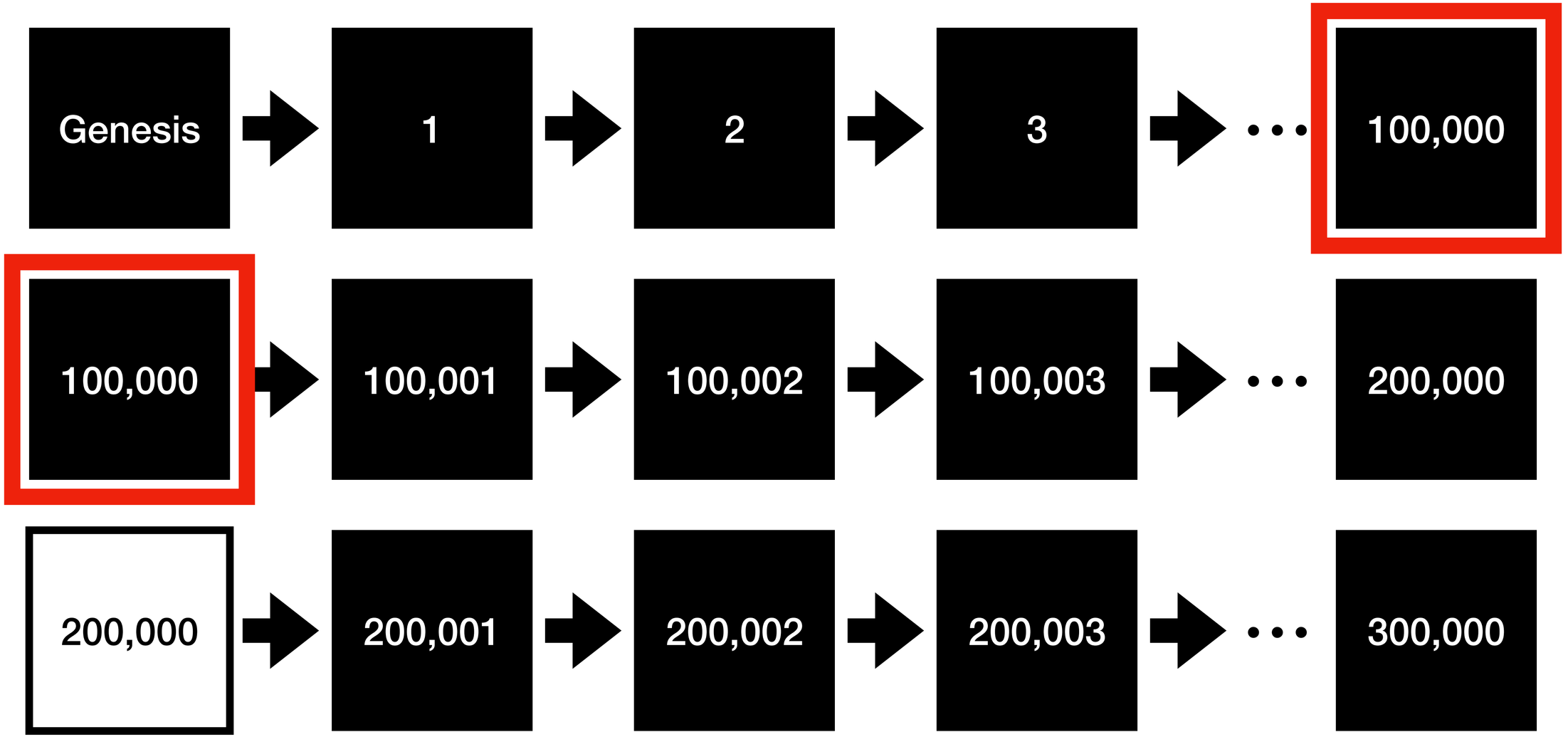
With Utreexo

Efficient parallel validation



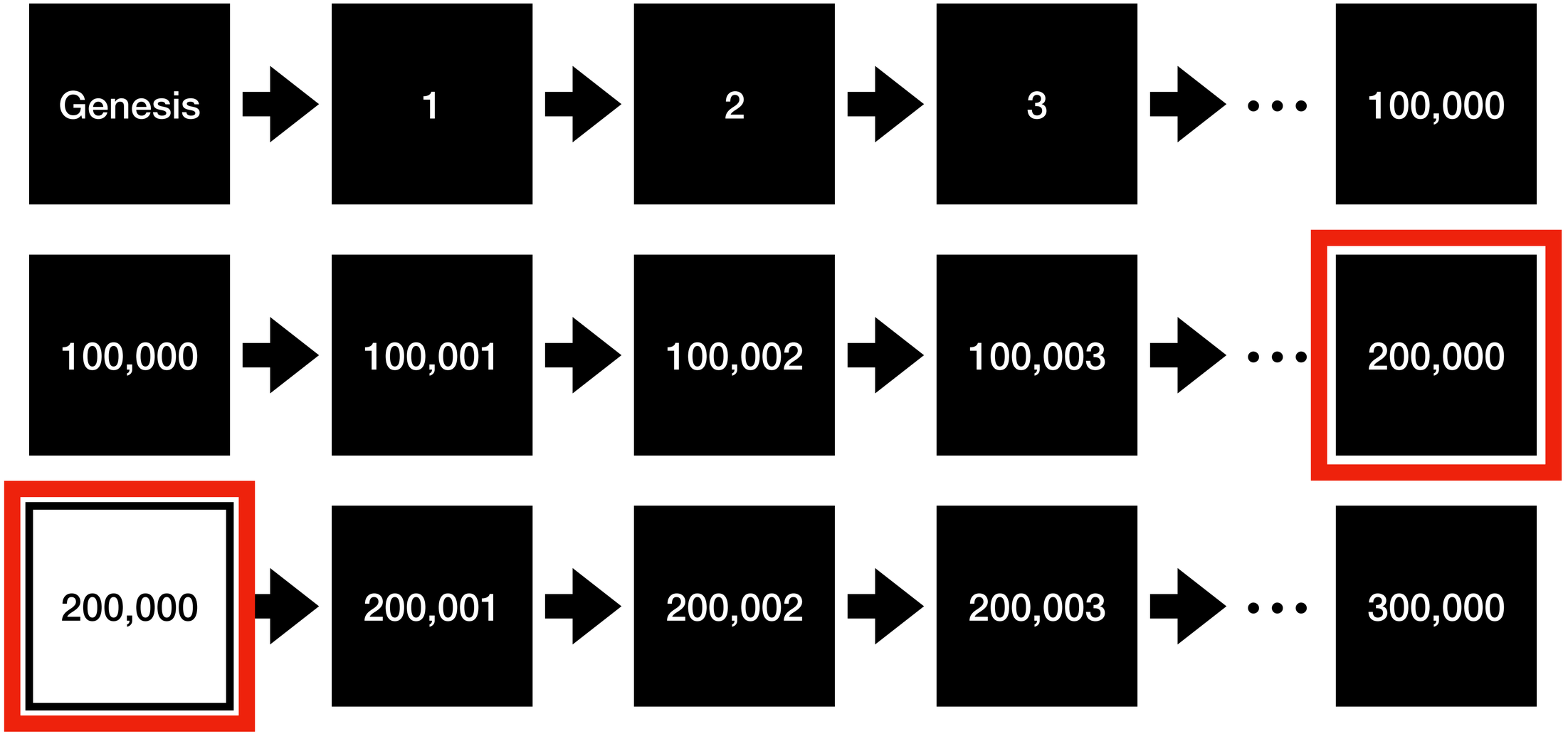
With Utreexo

Efficient parallel validation



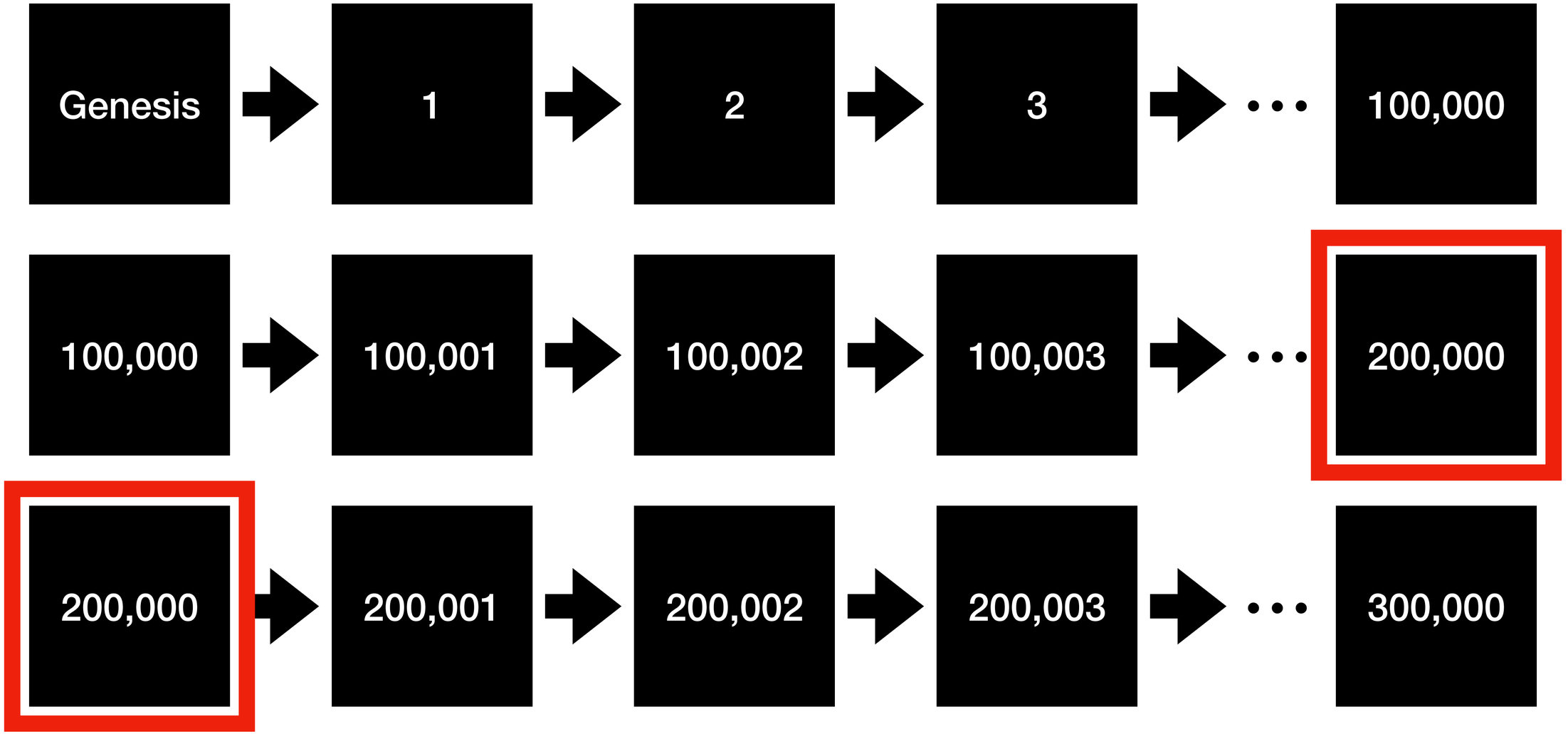
With Utreexo

Efficient parallel validation



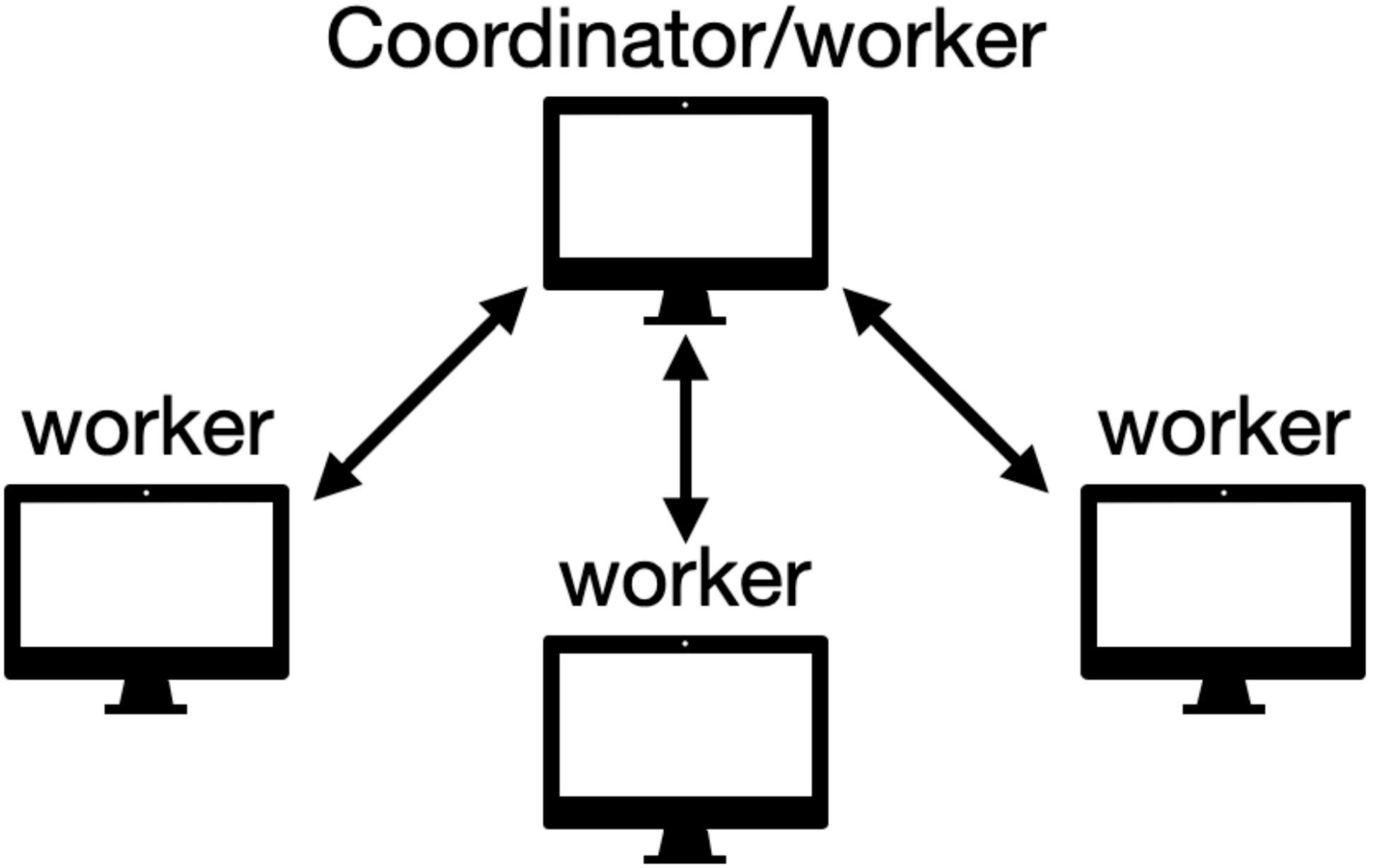
With Utreexo

Efficient parallel validation



~62% faster IBD

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

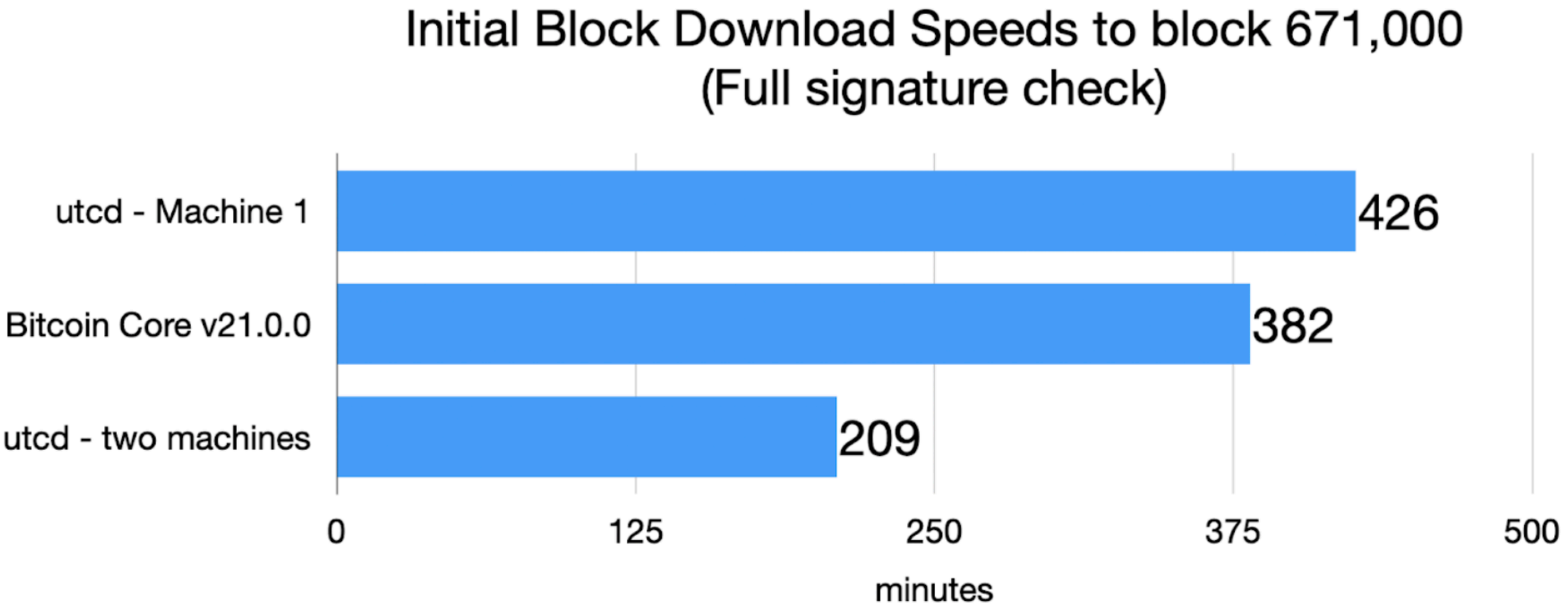


https://blog.bitmex.com/faster-blockchain-validation-with-utreexo-accumulators/

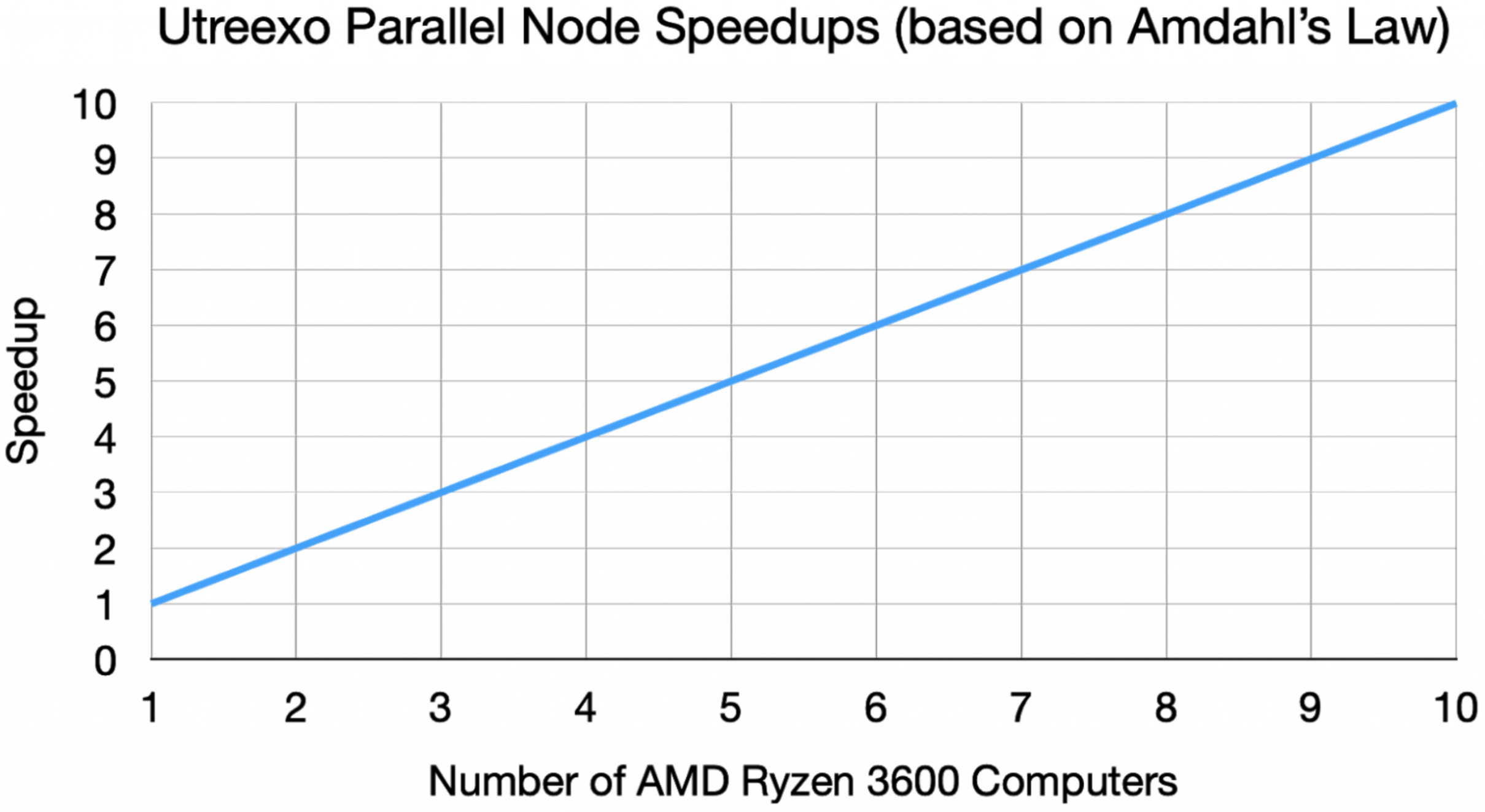
2x faster IBD

When using 2 computers for IBD

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



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


Utreexo Parallel Node Speedups (based on Amdahl's Law)
Speedup
0
1
2
3
4
5
6
7
8
9
10
1
2
3
4
5
6
7
8
9
10
Number of AMD Ryzen 3600 Computers

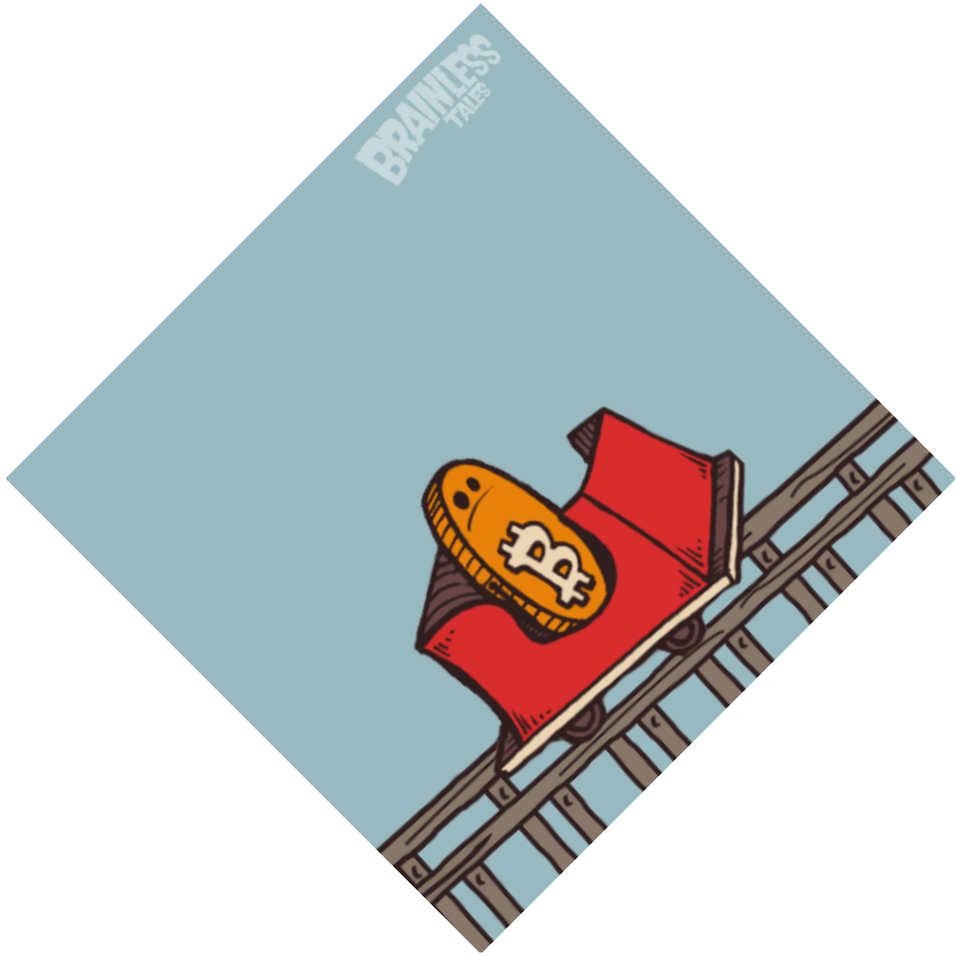
What's the catch?

More data to download

Extra data that a utreexo node will download

- 364GB of proof data (as of block 710,000)
- Optimizations being explored

Bandwidth
CPU

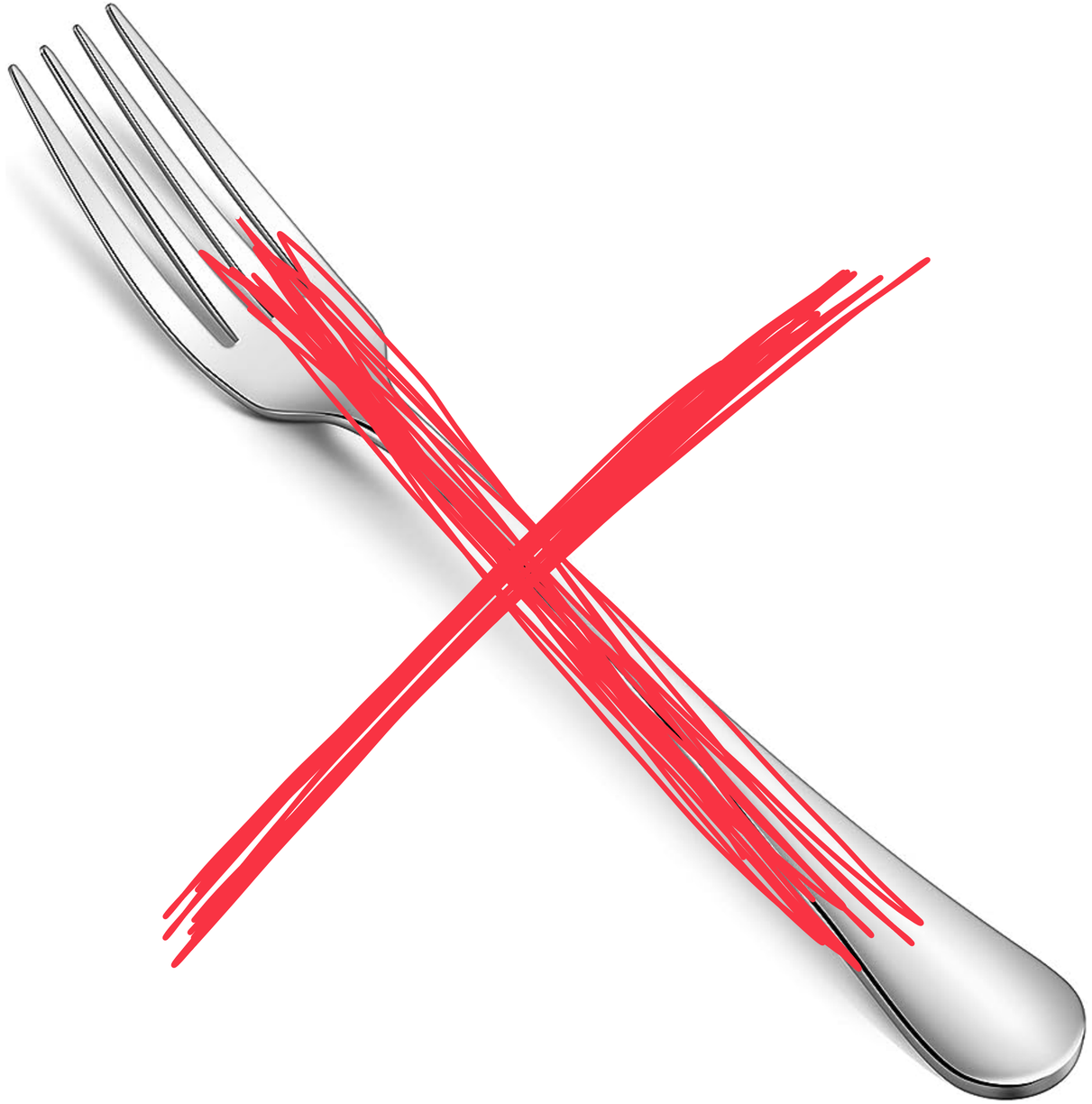


disk i/o
RAM usage

Deployment

100% optional

Ignore it if you don't want it



Need nodes to generate proofs

"bridge nodes"

- Need to generate proofs for new blocks/txs
- These "bridge nodes" keep the entire utxo set and the entire merkle forest

Need nodes to generate proofs

"bridge nodes"

- Can be pruned
- Implemented as an indexer in github.com/utreexo/utreexod

Aligning incentives

No burden of UTXO storage


coinbase


coinbase


BINANCE

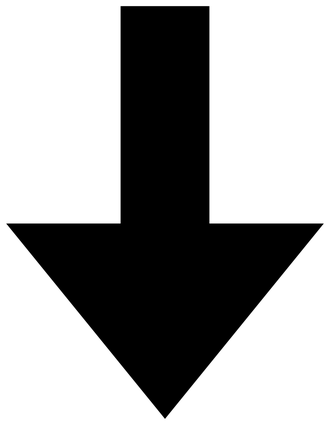
coinbase

BINANCE

BITCOIN STAMPS

Unprunable UTXO Art, Because Sats Don’t Exist.

Everyone



Each store their own

How does it work?

Algorithms

Consensus critical stuff

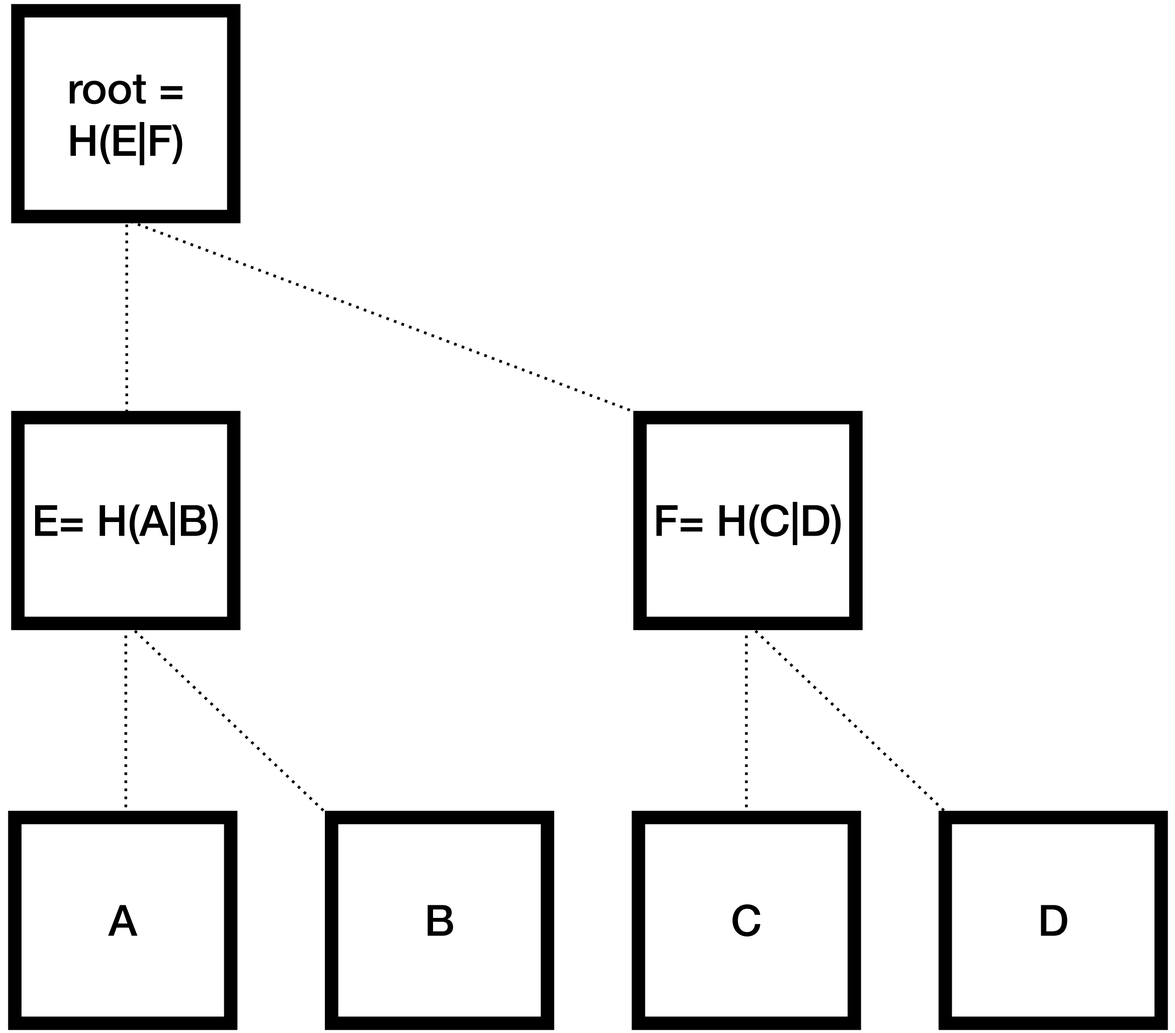
1. Add a UTXO
2. Delete a UTXO
3. Verify the existence

Algorithms

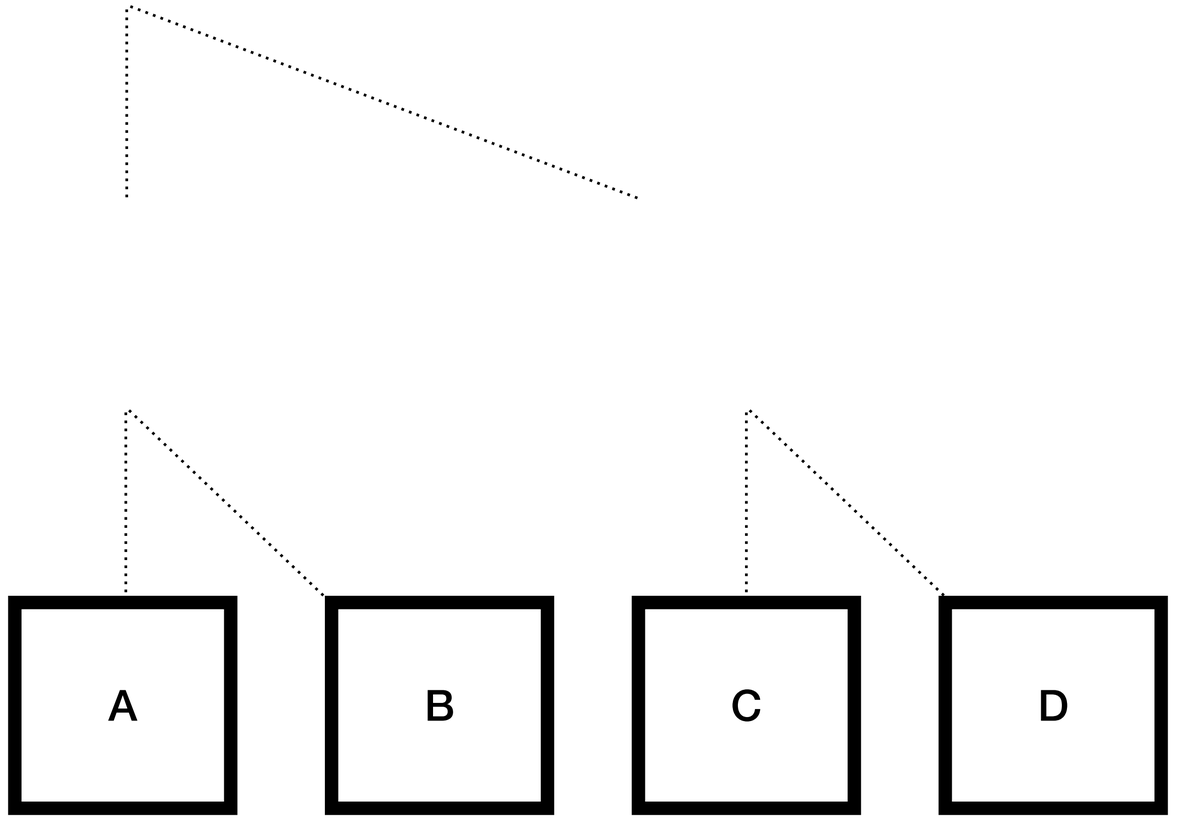
Consensus critical stuff

1. Add a UTXO
2. Delete a UTXO
3. Verify the existence

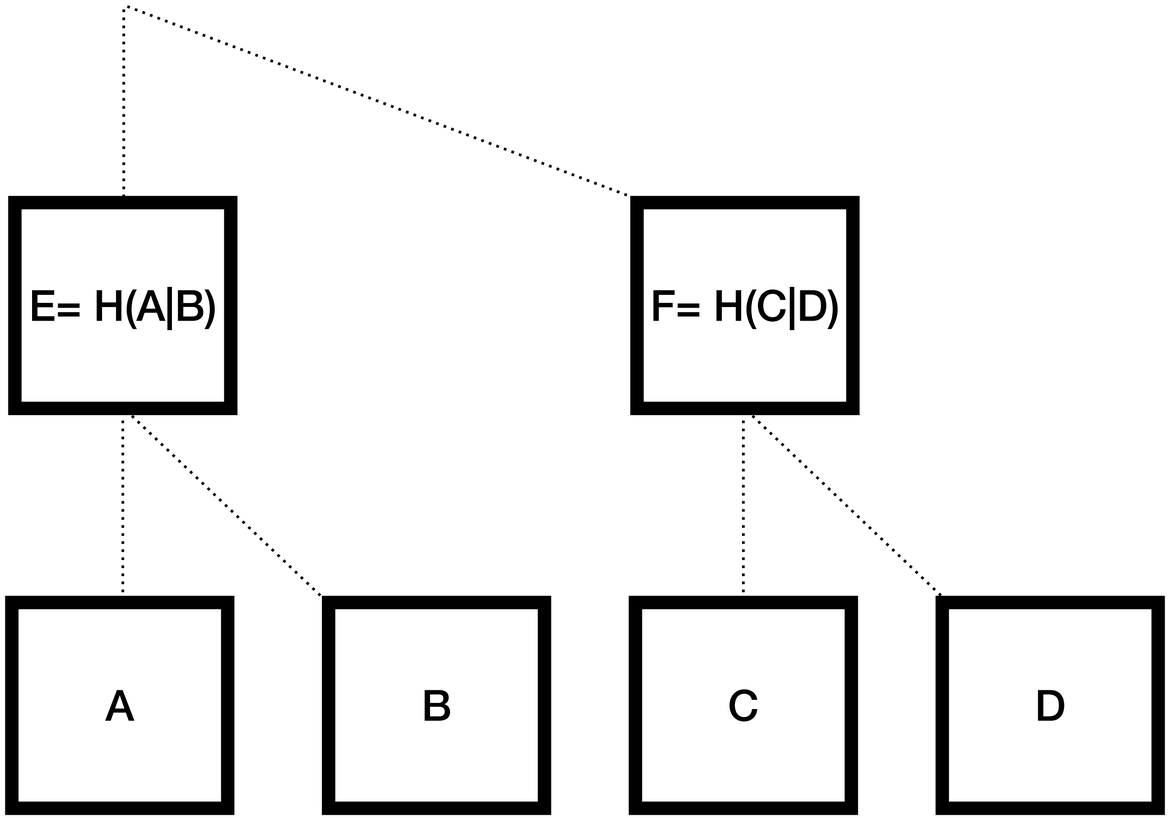
Quick review of merkle tree creation



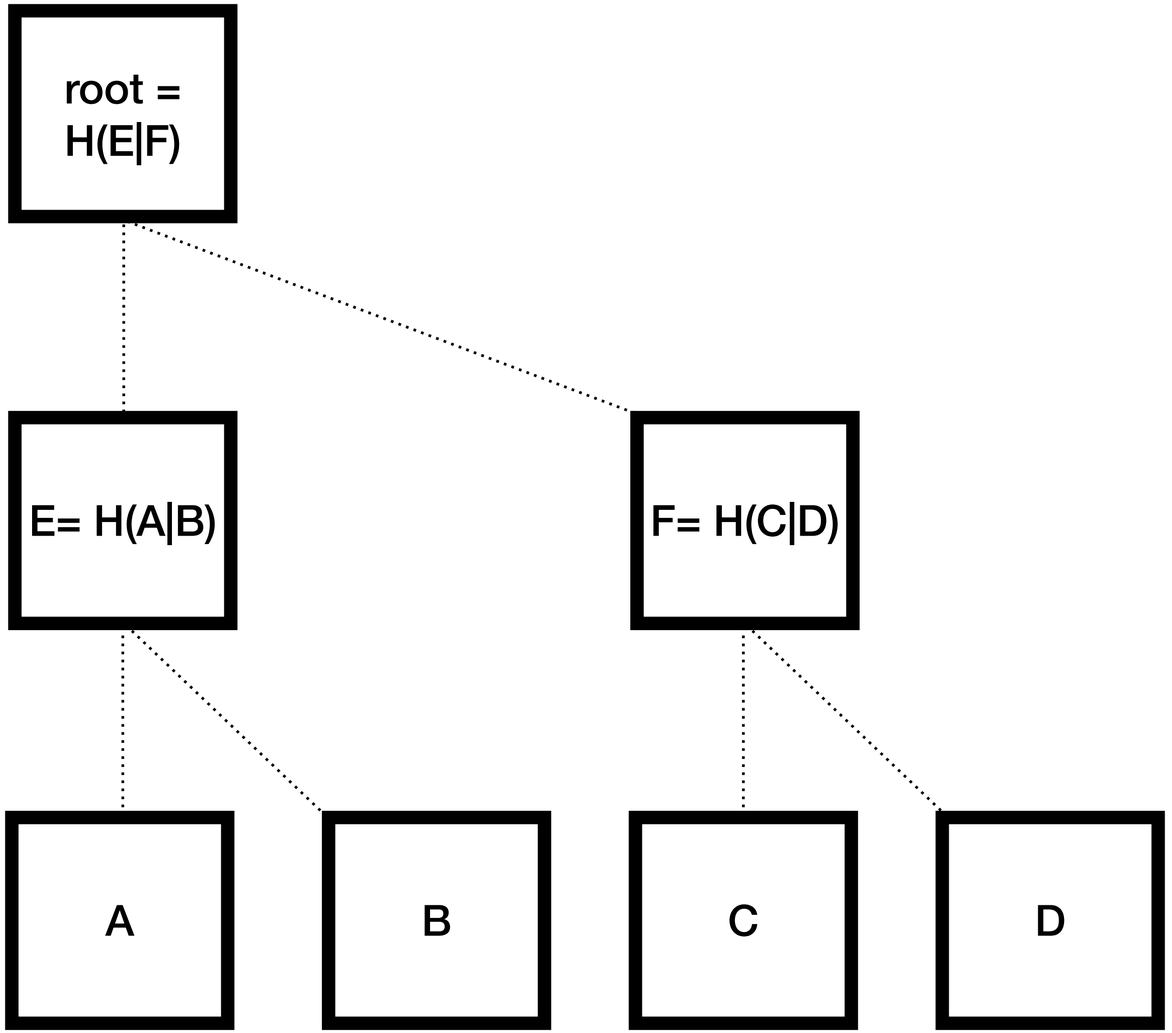
Start off with the hash of the elements



Hash each pairs

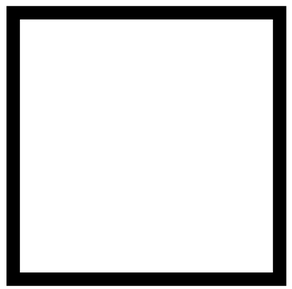


Hash last pair



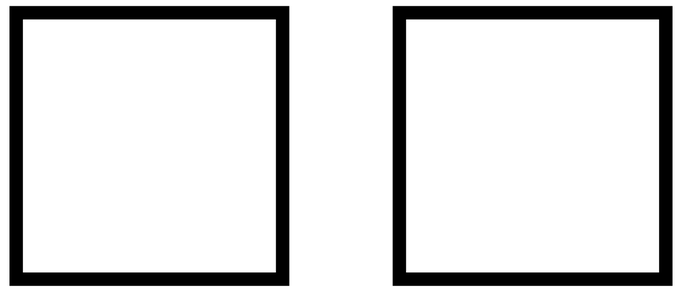
1 UTXO

Single root



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

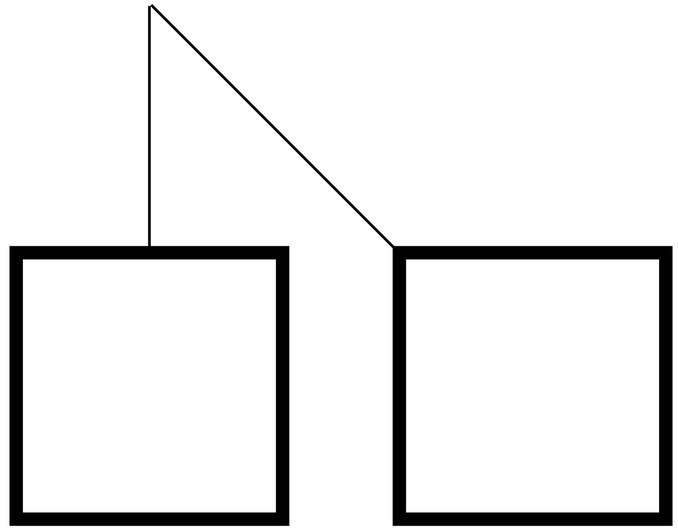
2 UTXOs



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

2 UTXOs

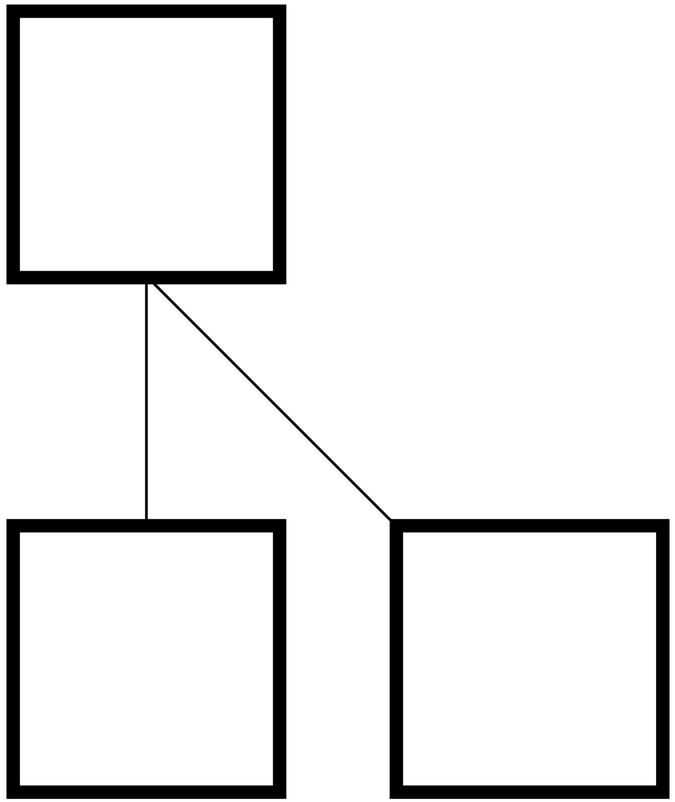
Concatenate



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

2 UTXOs

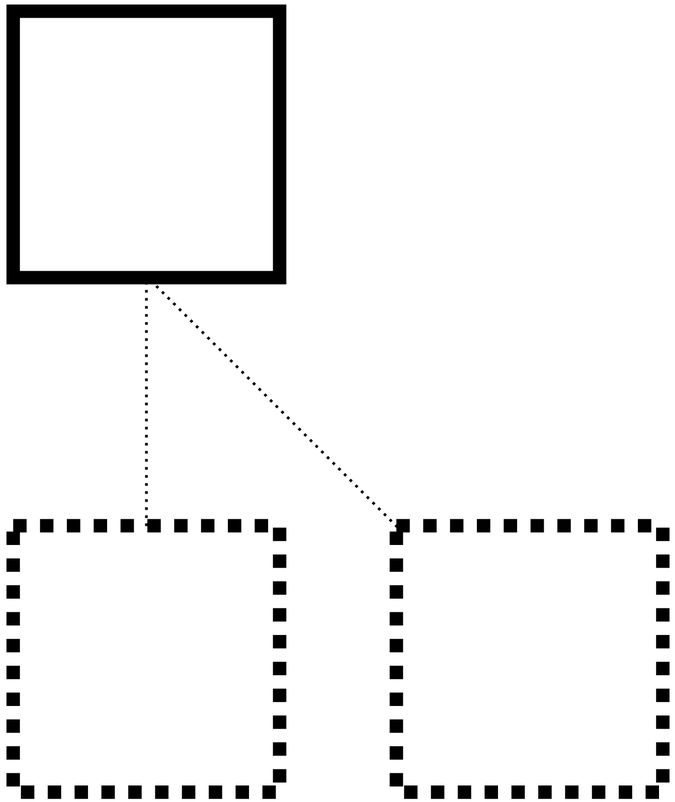
Hash to create a new root



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

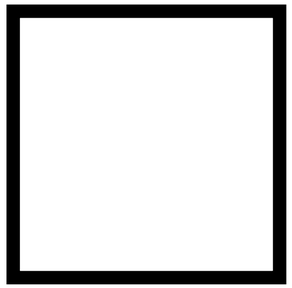
2 UTXOs

Can now delete the leaves



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

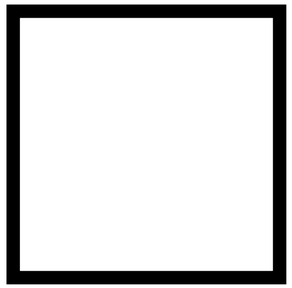
2 UTXOs



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

3 UTXOs

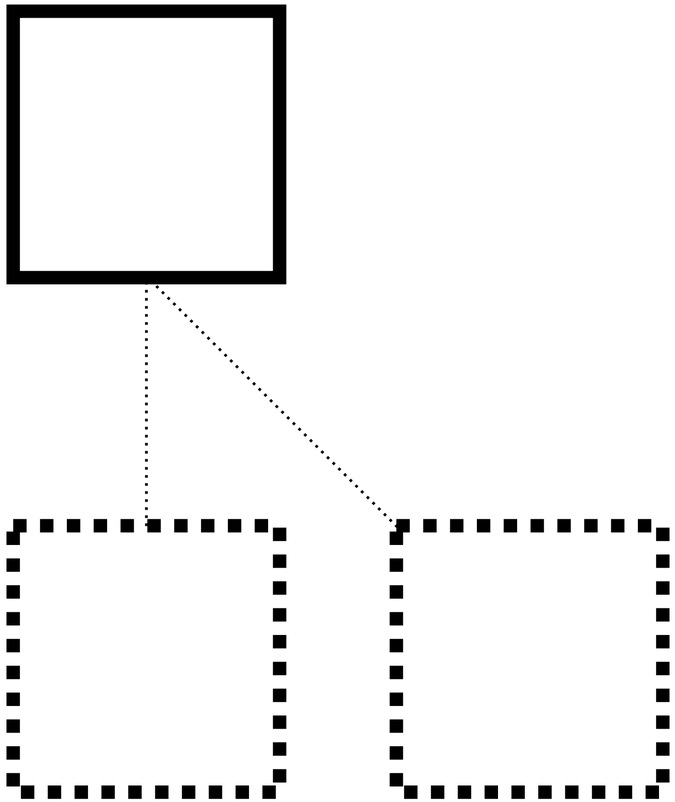
Add 1 more



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

3 UTXOs

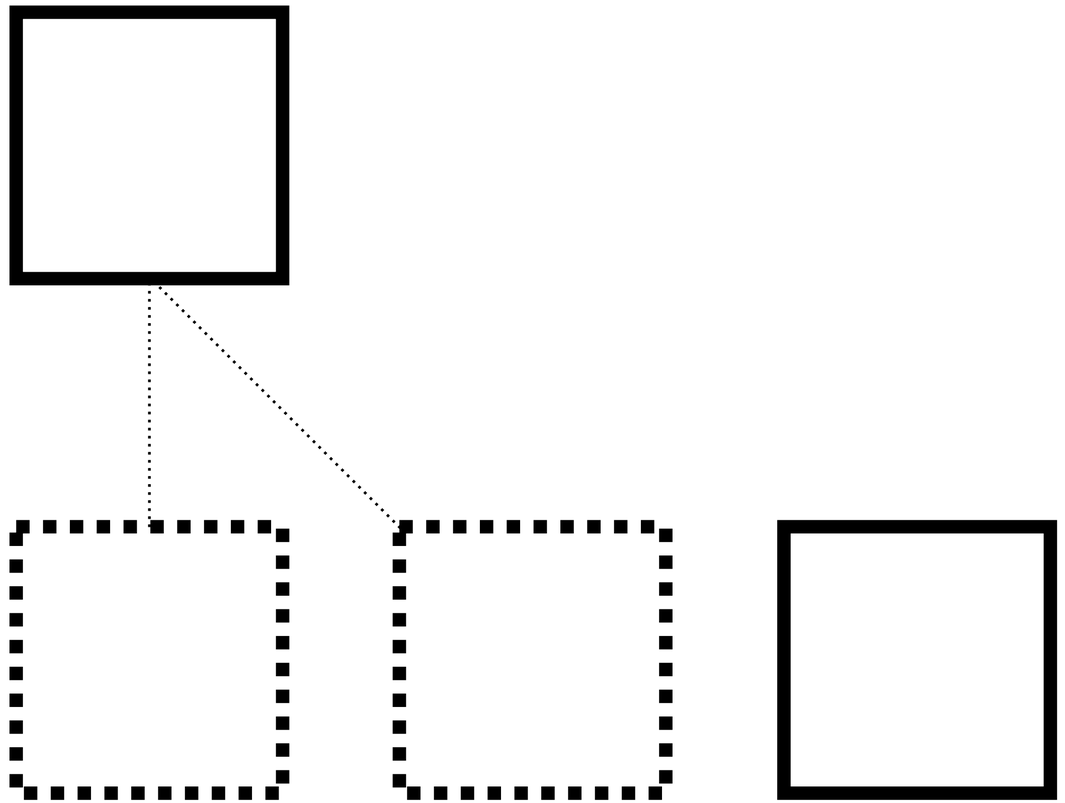
Add 1 more



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

3 UTXOs

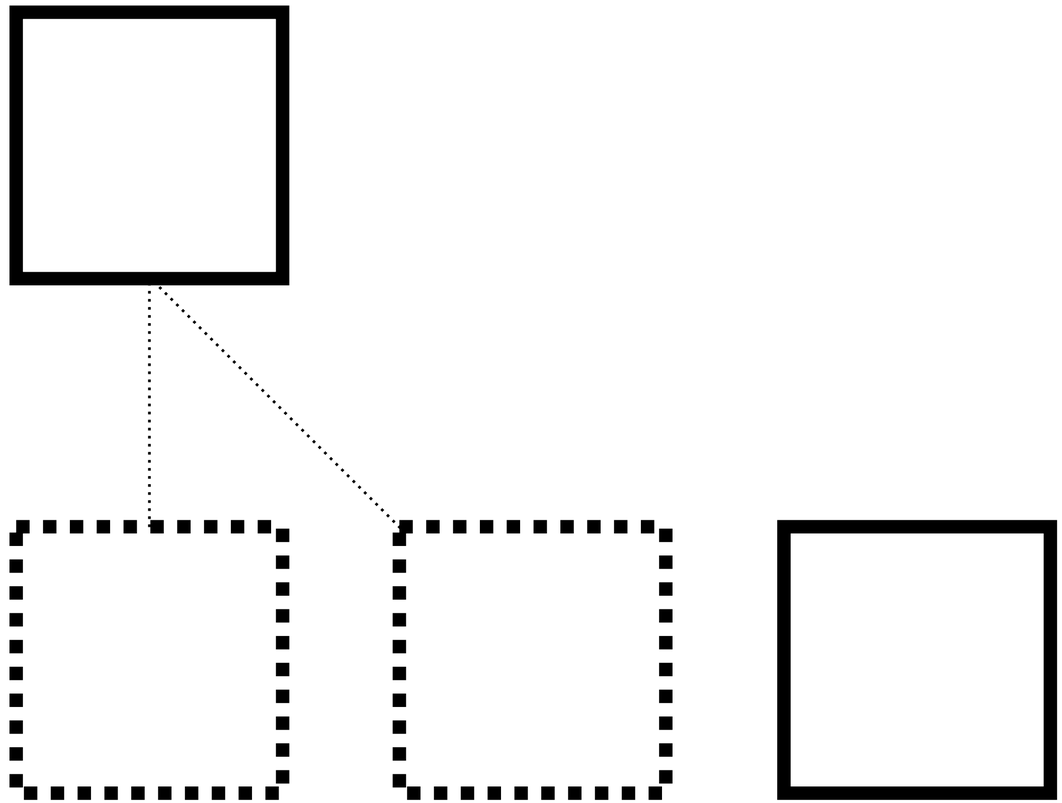
Add 1 more



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

3 UTXOs

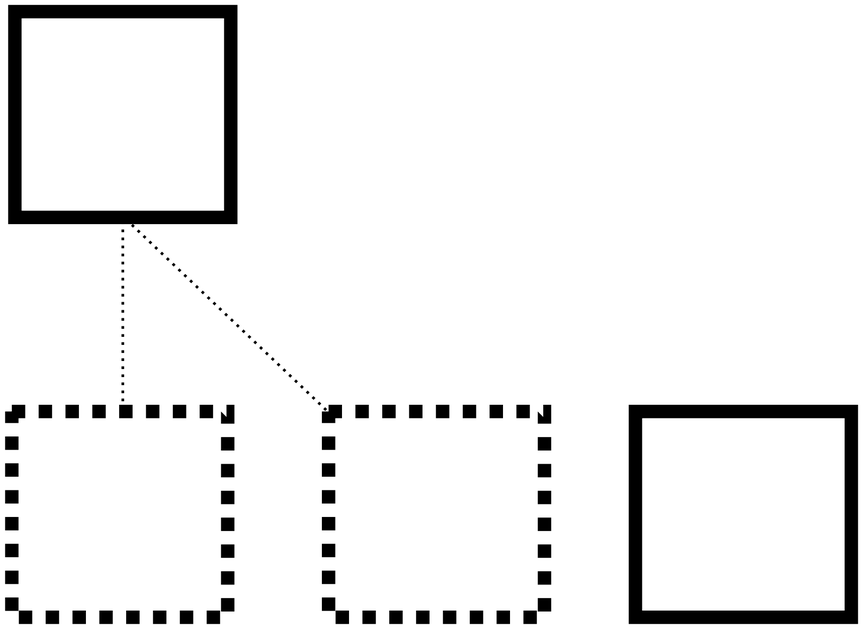
Nothing to hash with. Becomes a root



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

4 UTXOs

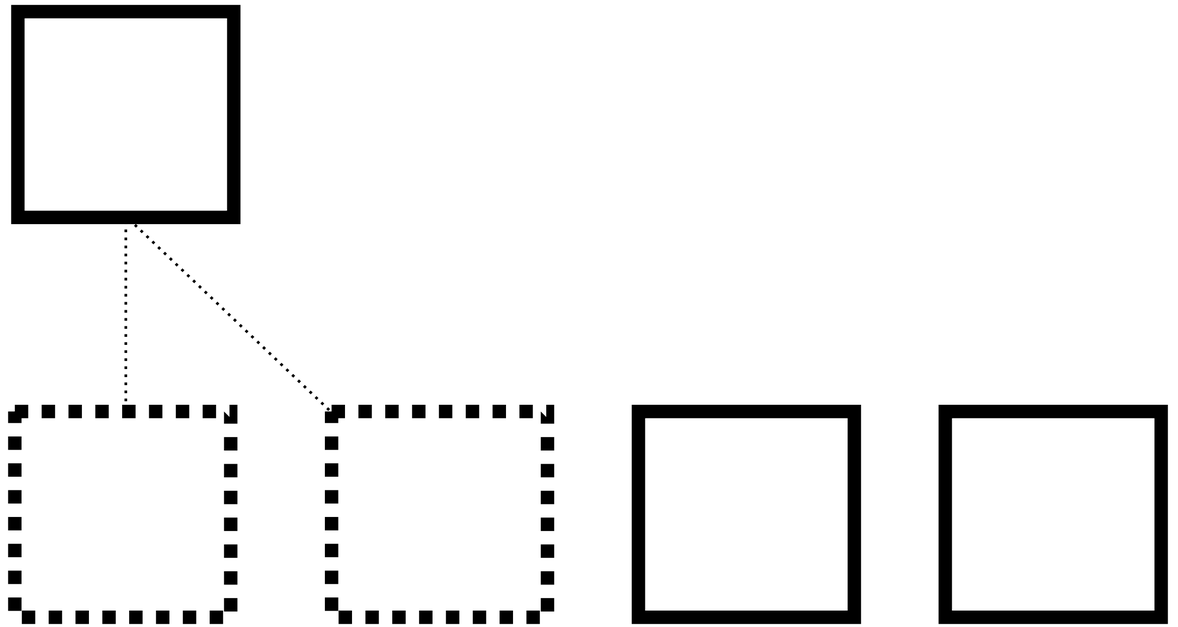
Another one



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

4 UTXOs

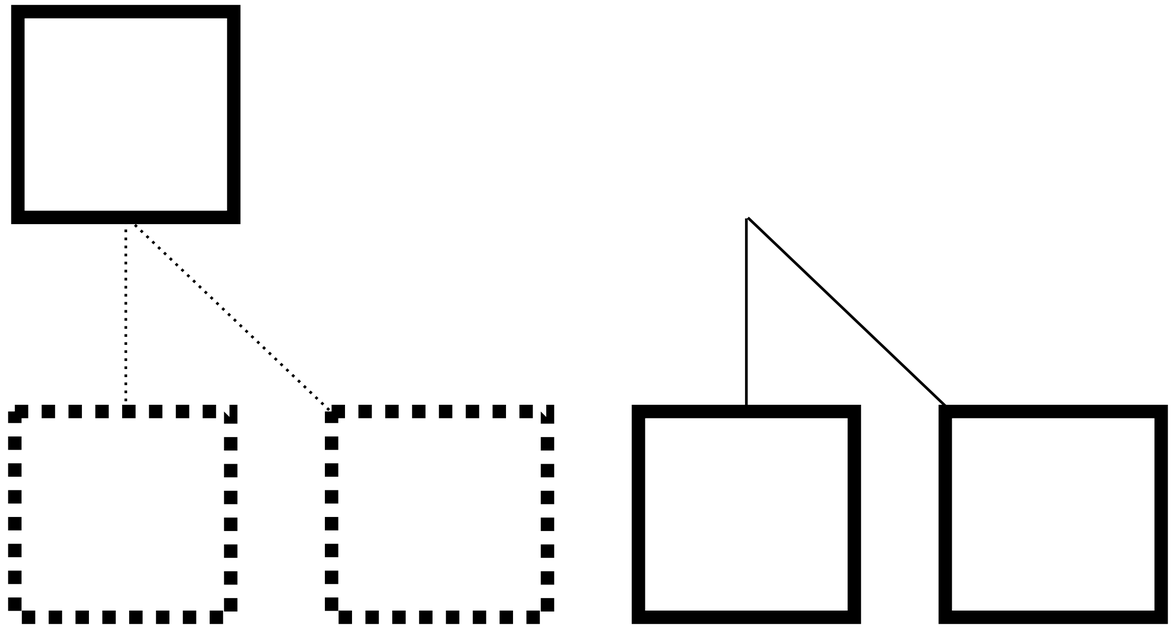
Another one



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

4 UTXOs

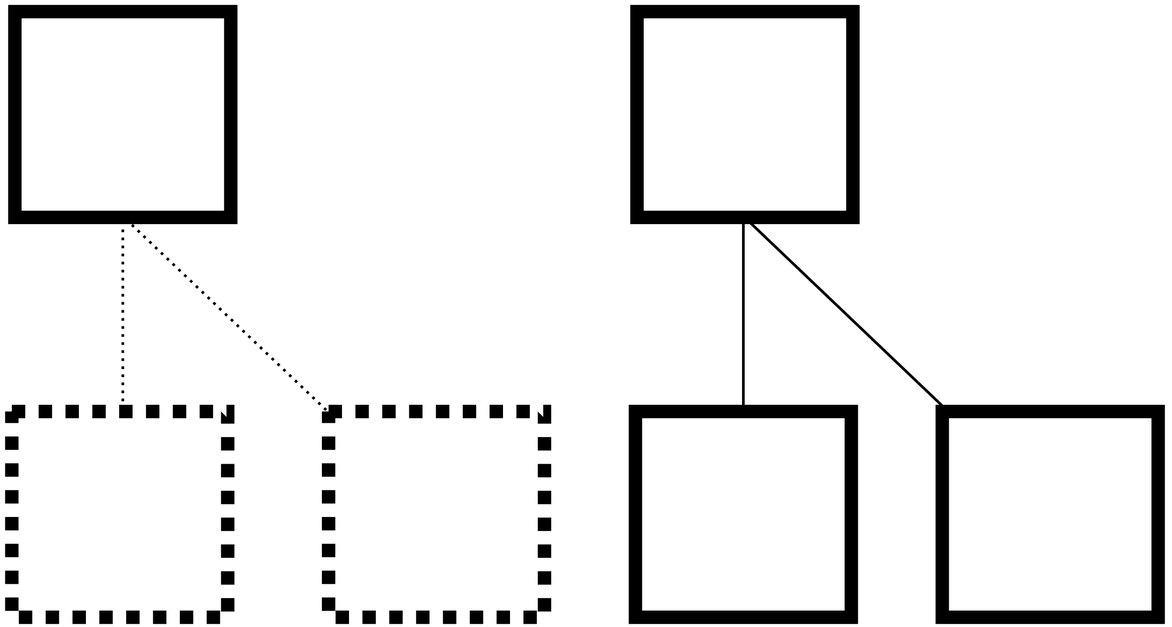
Concatenate



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

4 UTXOs

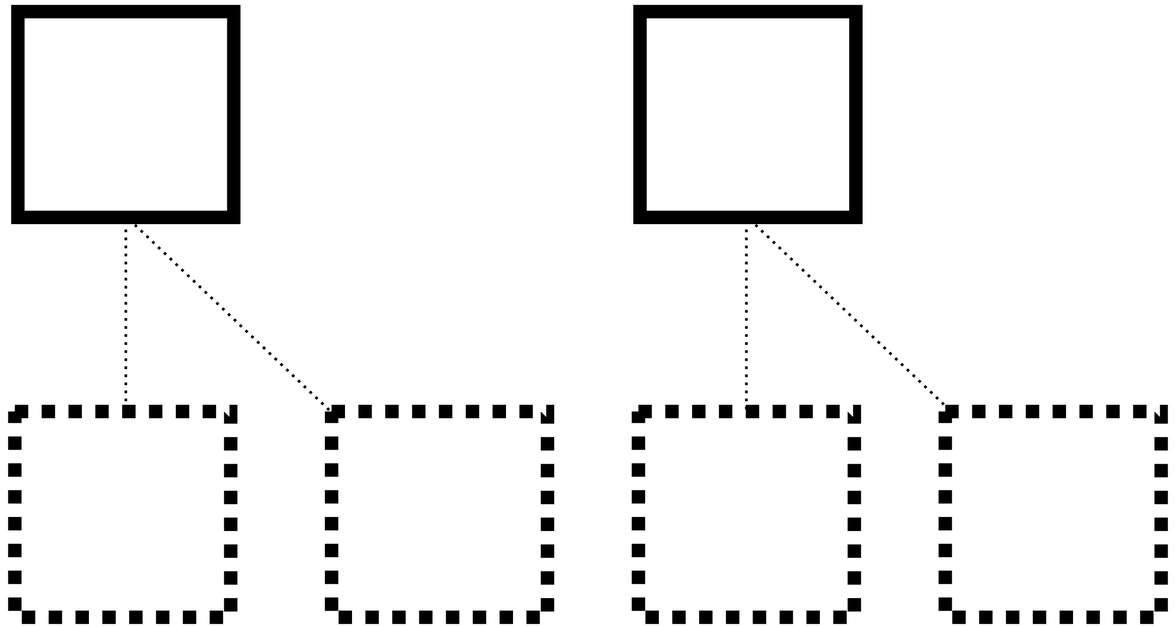
Hash



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

4 UTXOs

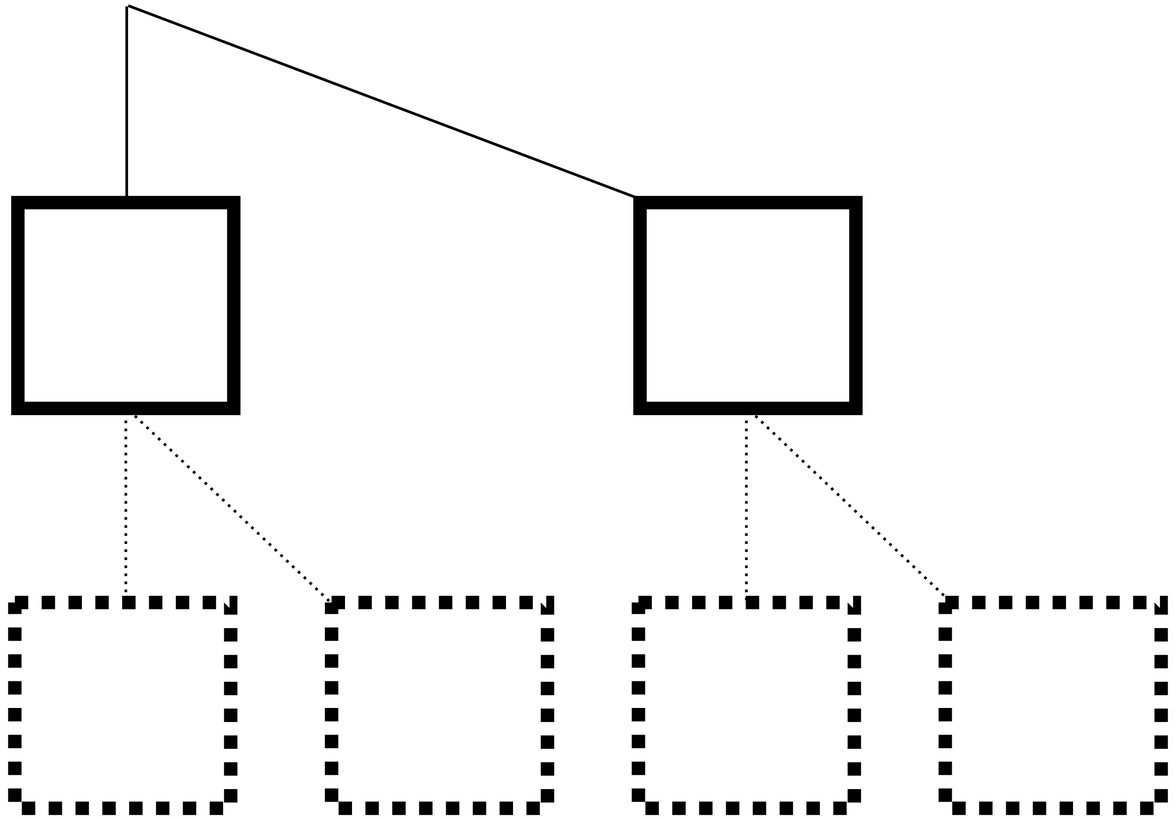
Can now throw away the leaves



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

4 UTXOs

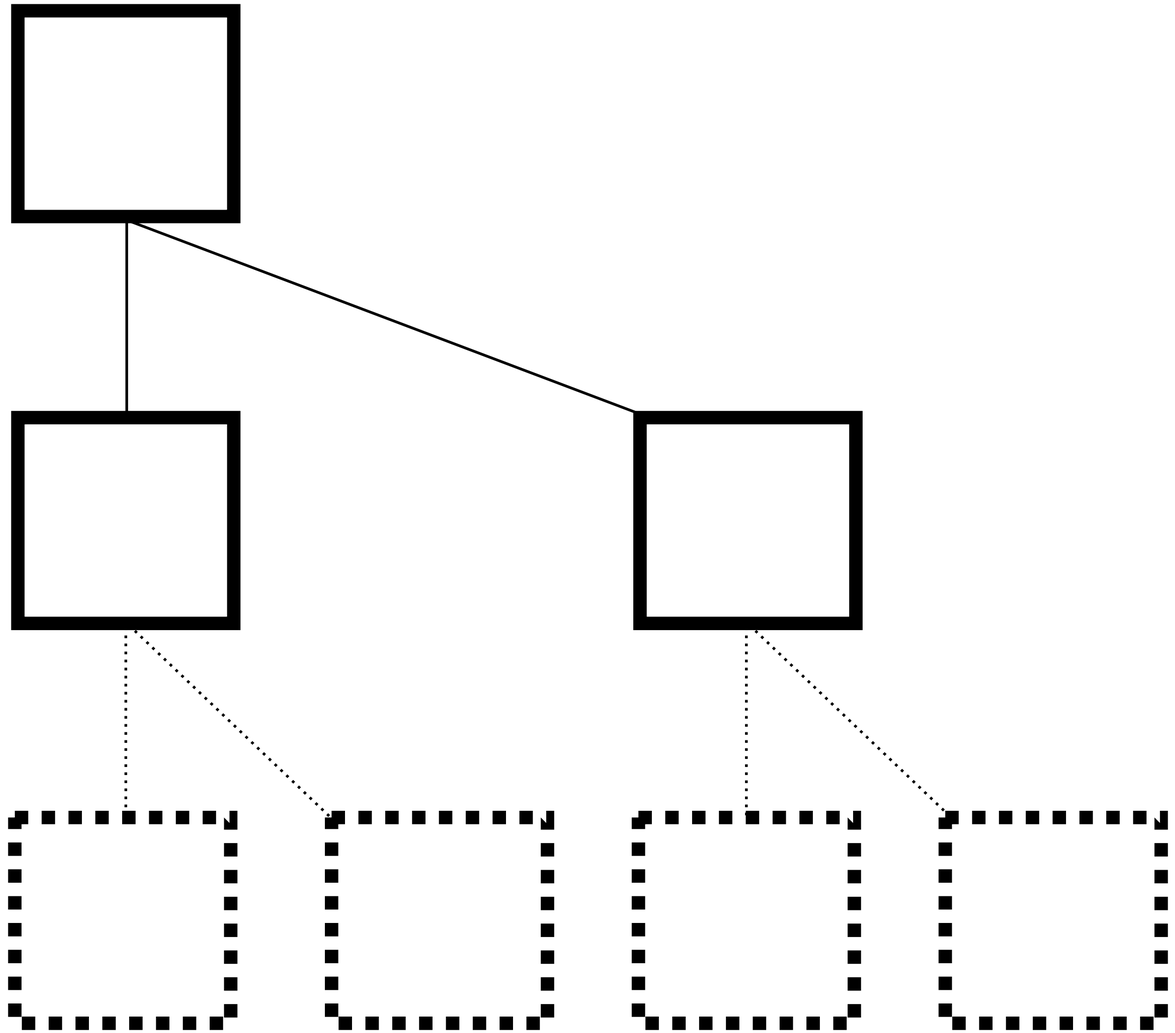
Concatenate



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

4 UTXOs

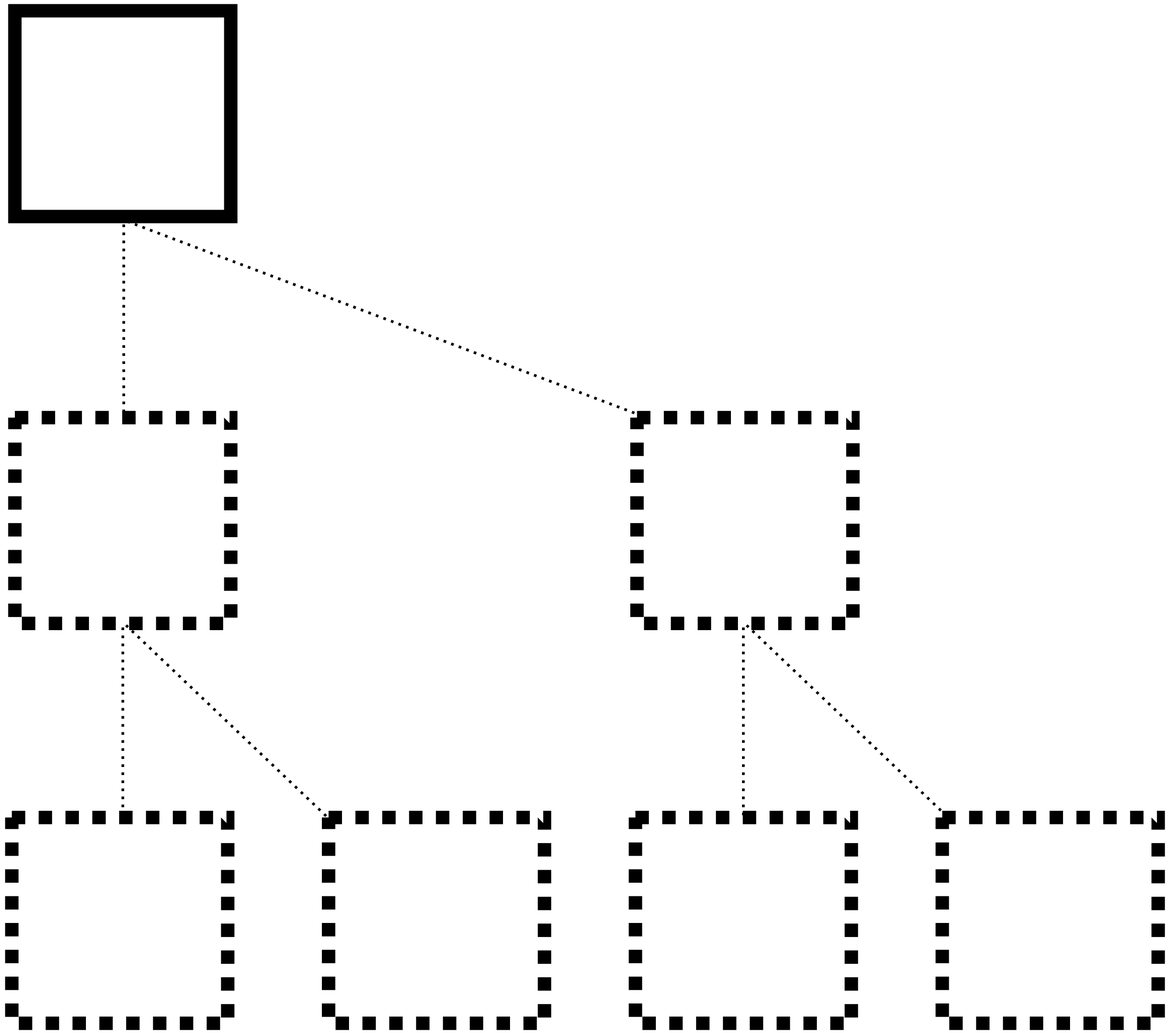
Hash



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

4 UTXOs

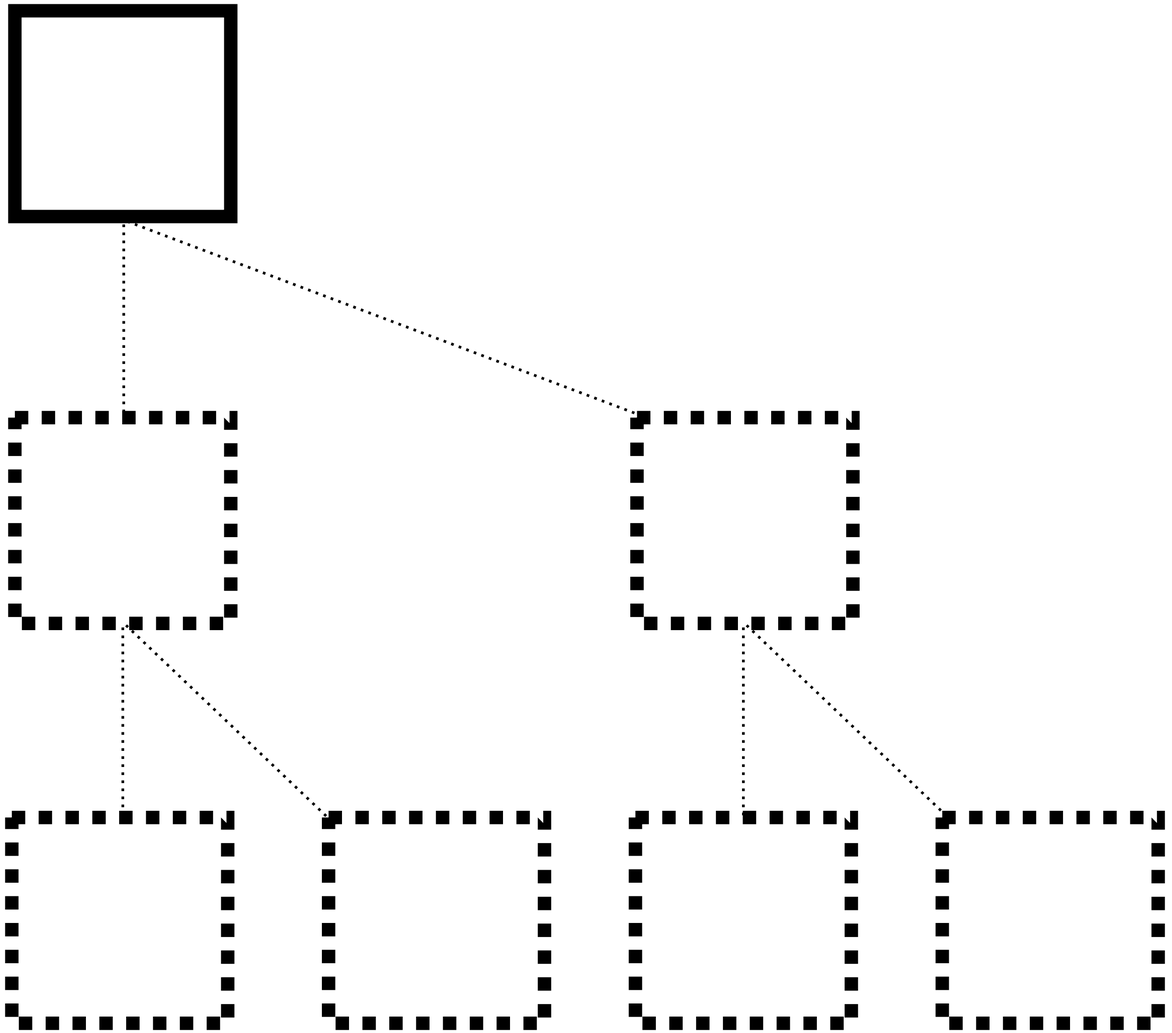
Throw away



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

4 UTXOs

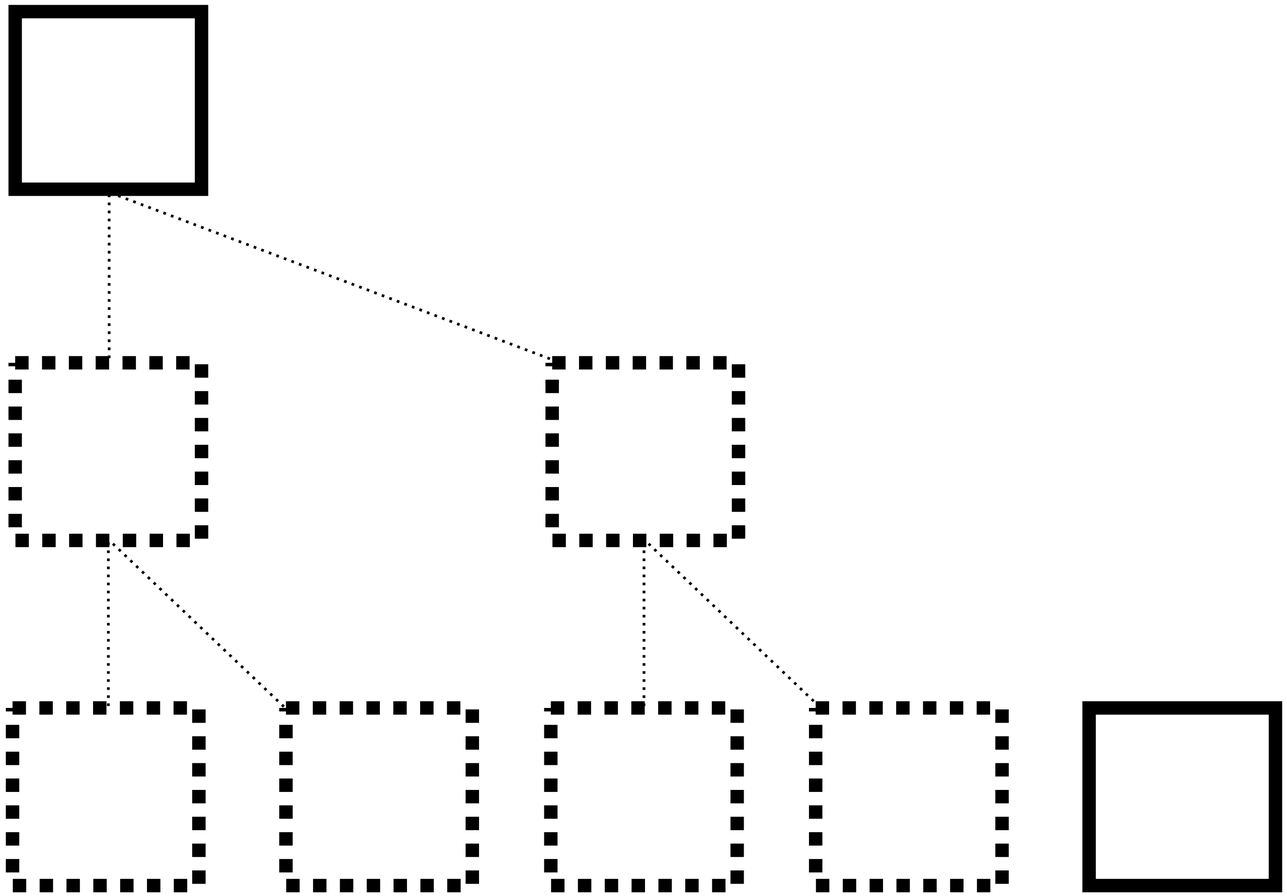
Finished tree



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

5 UTXOs

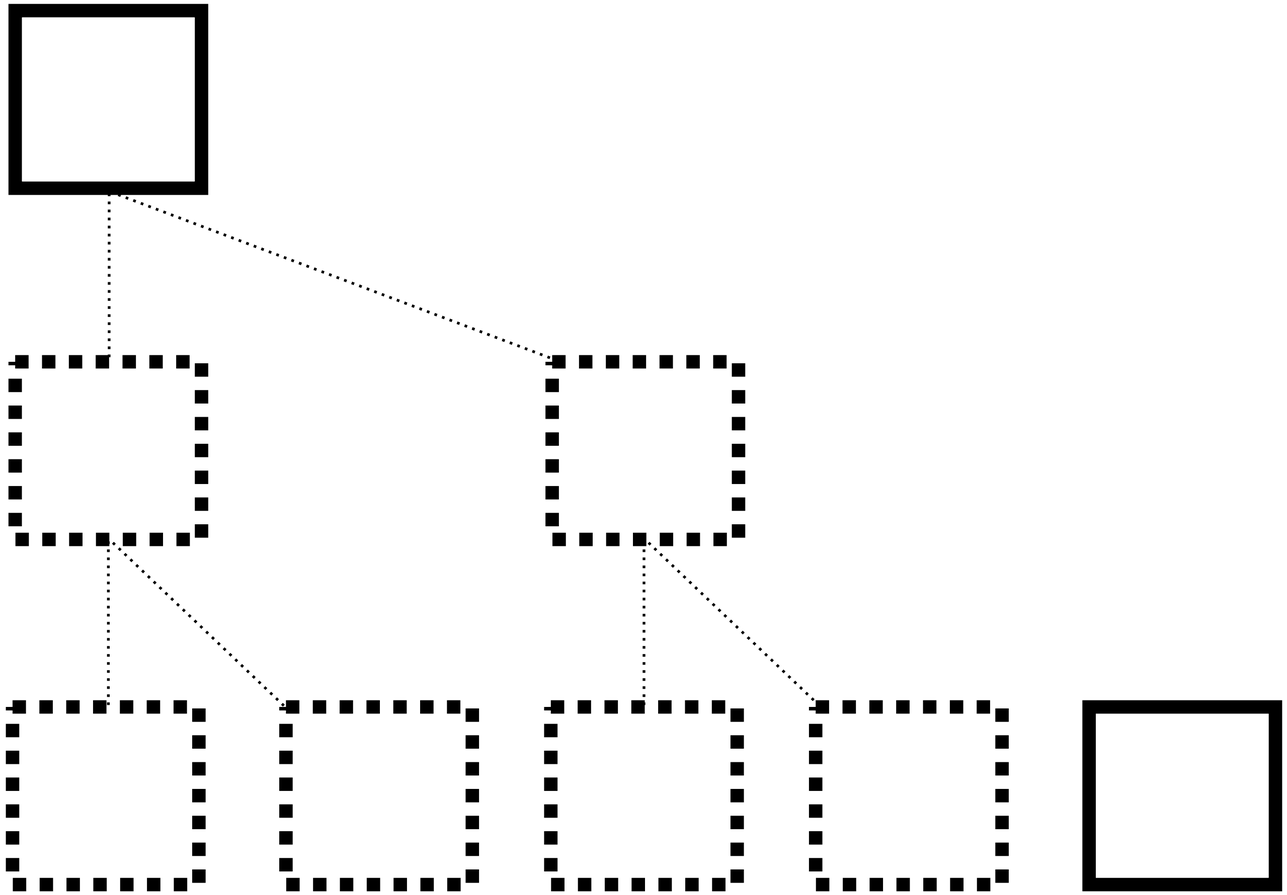
Add one more



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

5 UTXOs

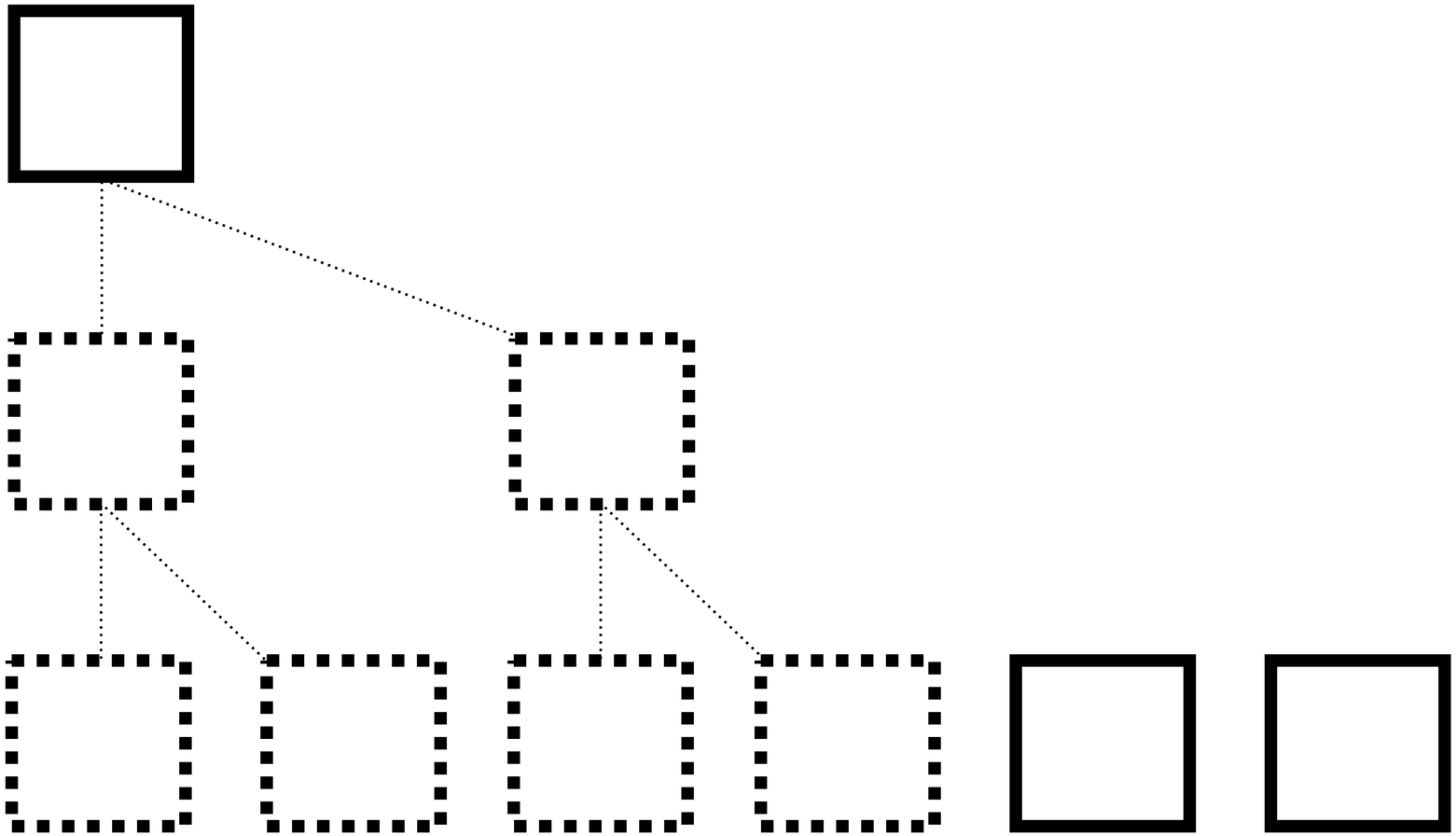
Nothing to hash with so it becomes a root



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

6 UTXOs

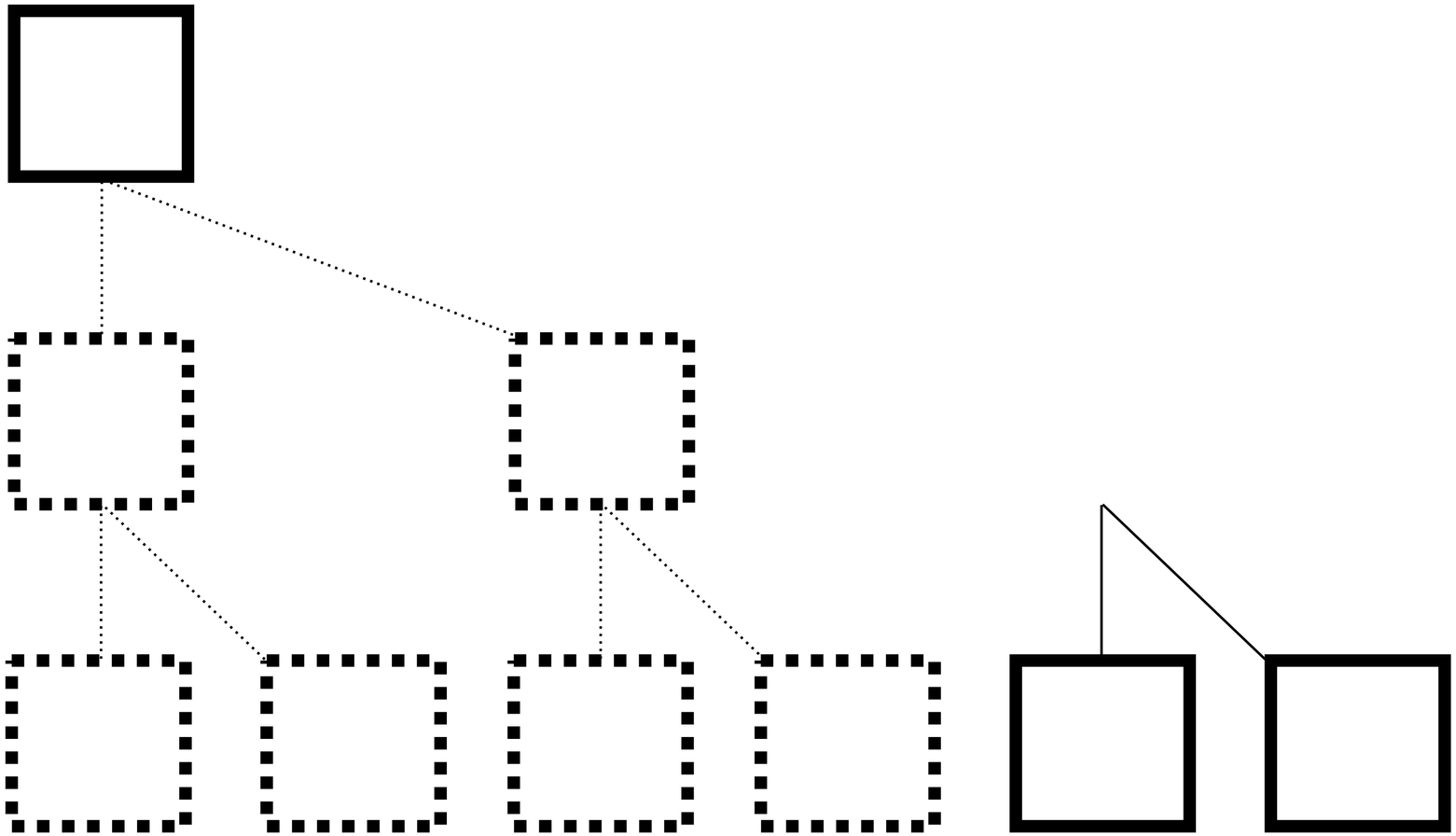
1 more



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

6 UTXOs

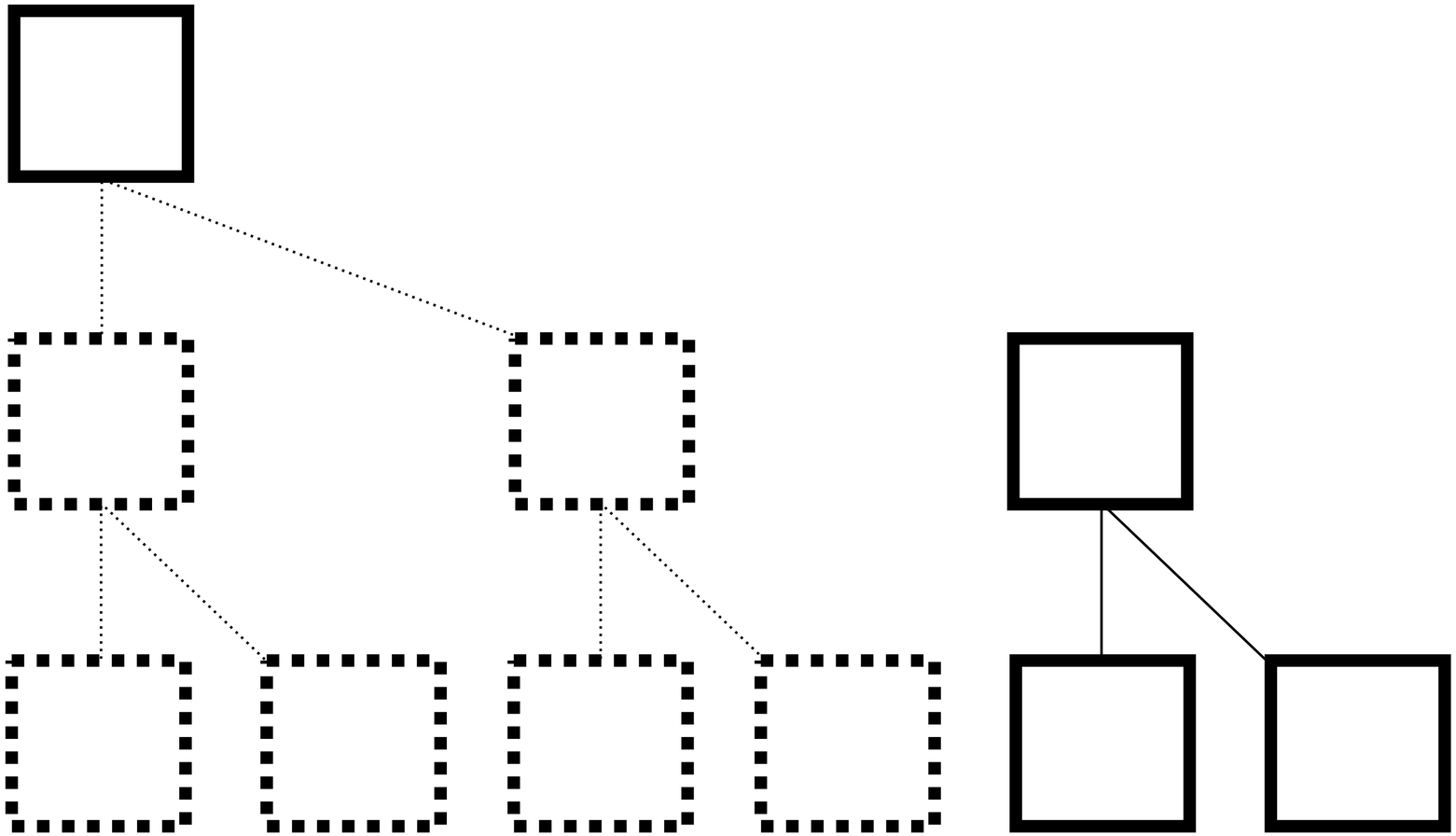
Concatenate



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

6 UTXOs

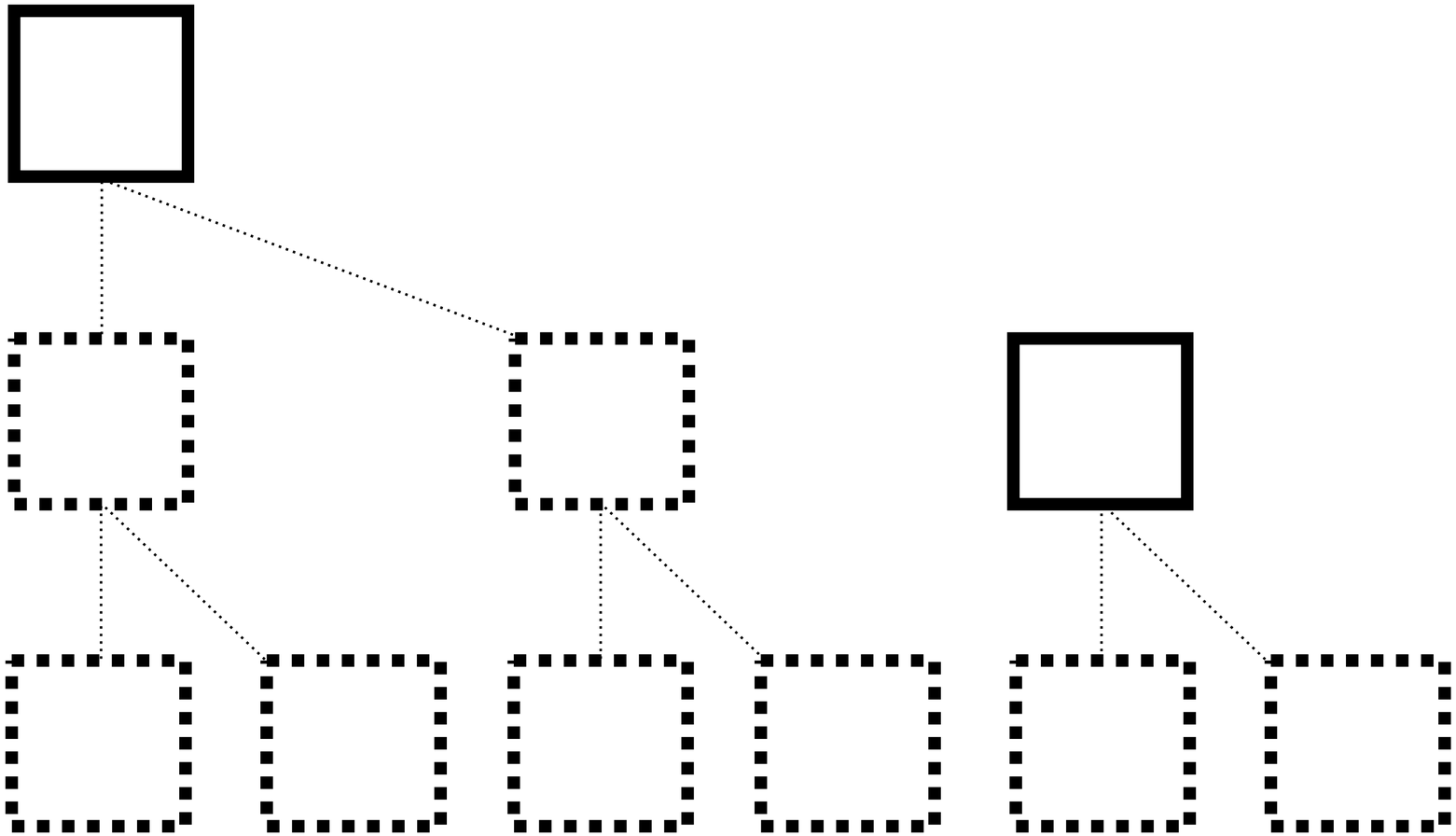
Hash



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

6 UTXOs

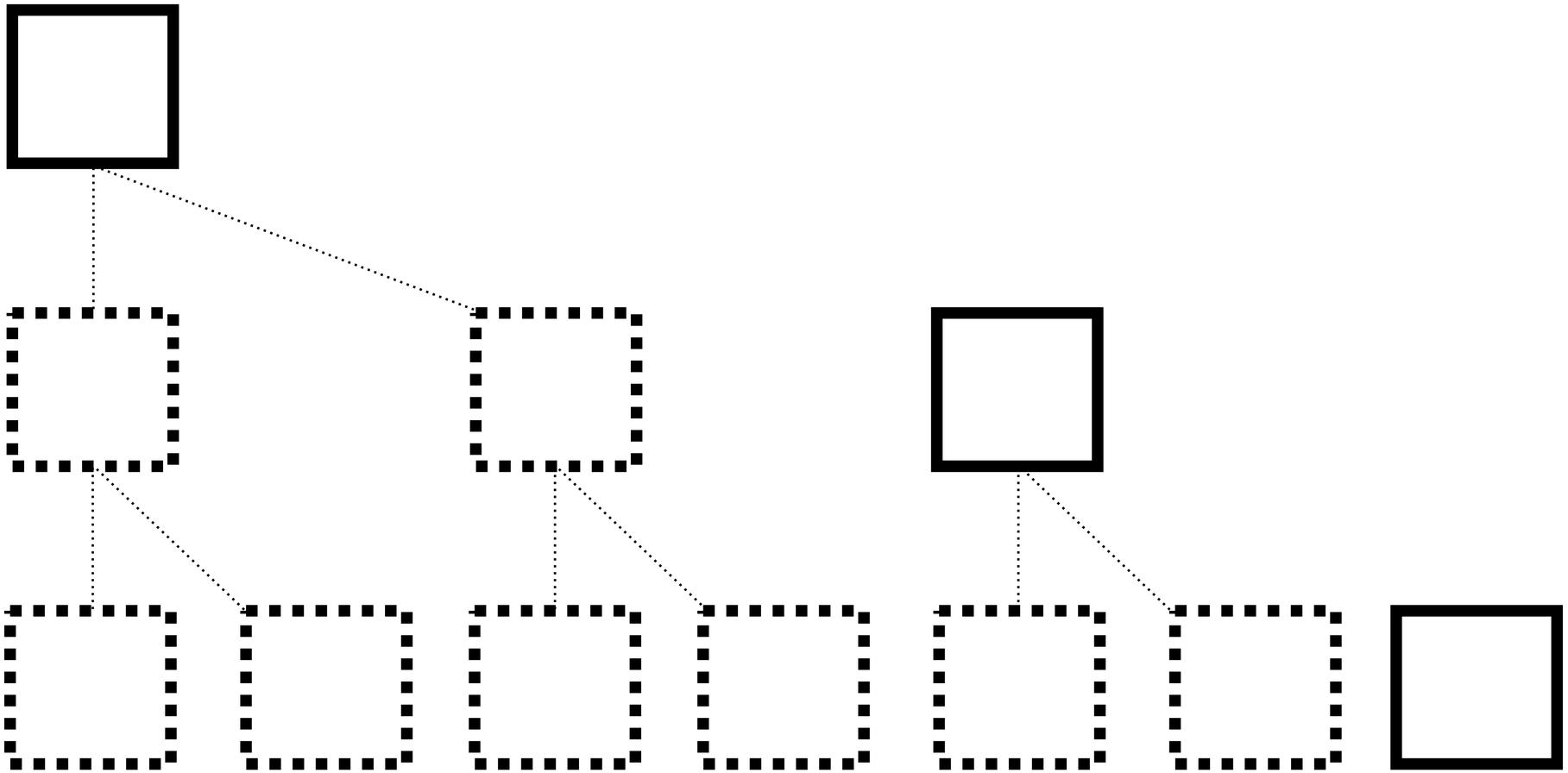
Throw away



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

7 UTXOs

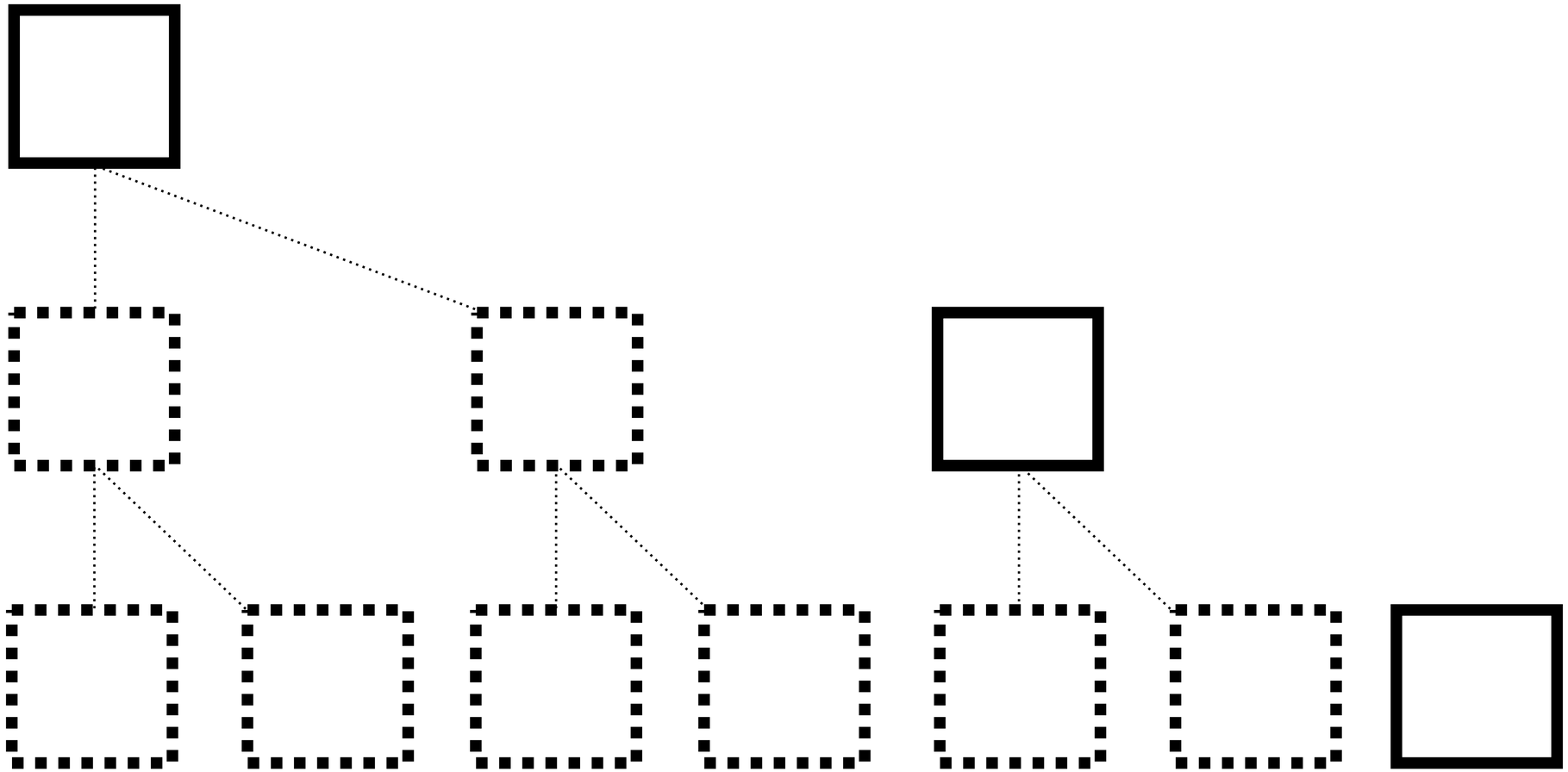
Another one



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

7 UTXOs

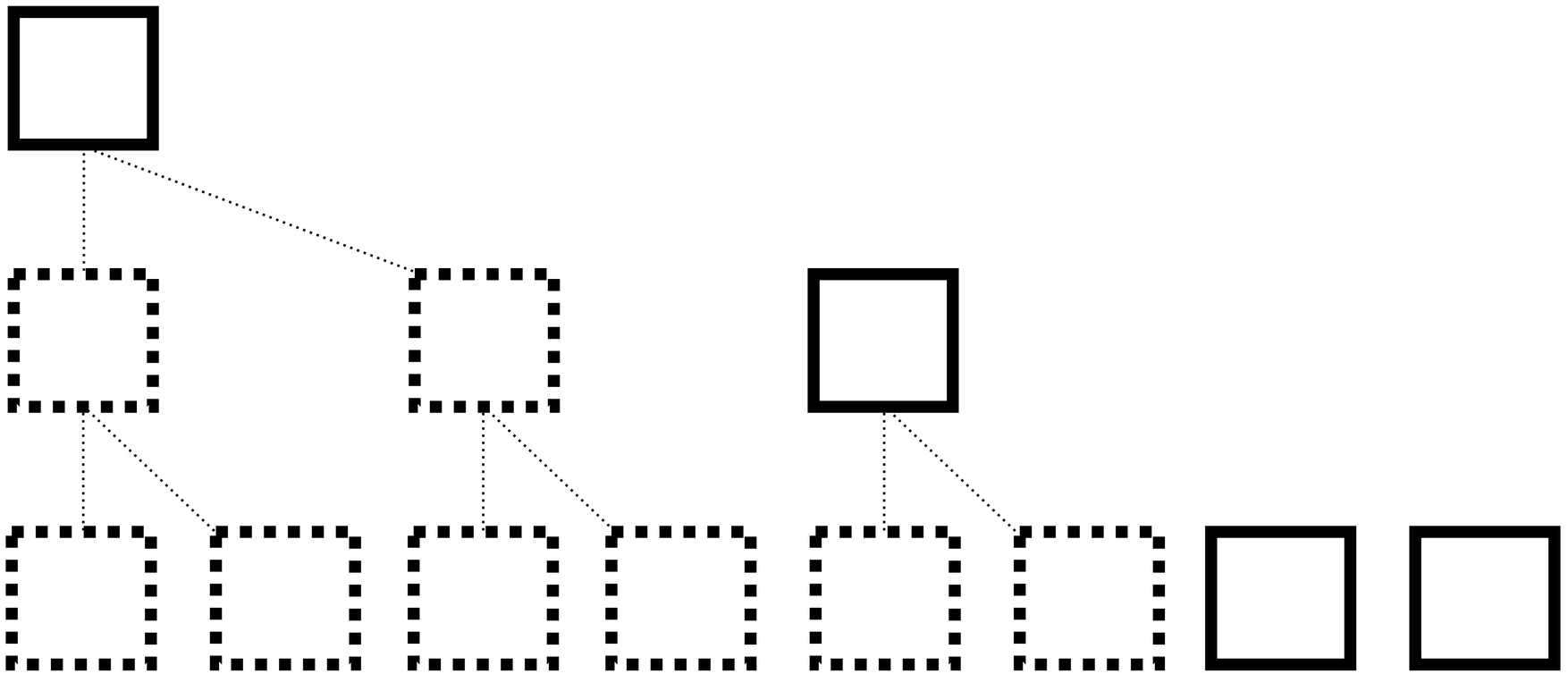
Nothing to hash with so it becomes a root



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

8 UTXOs

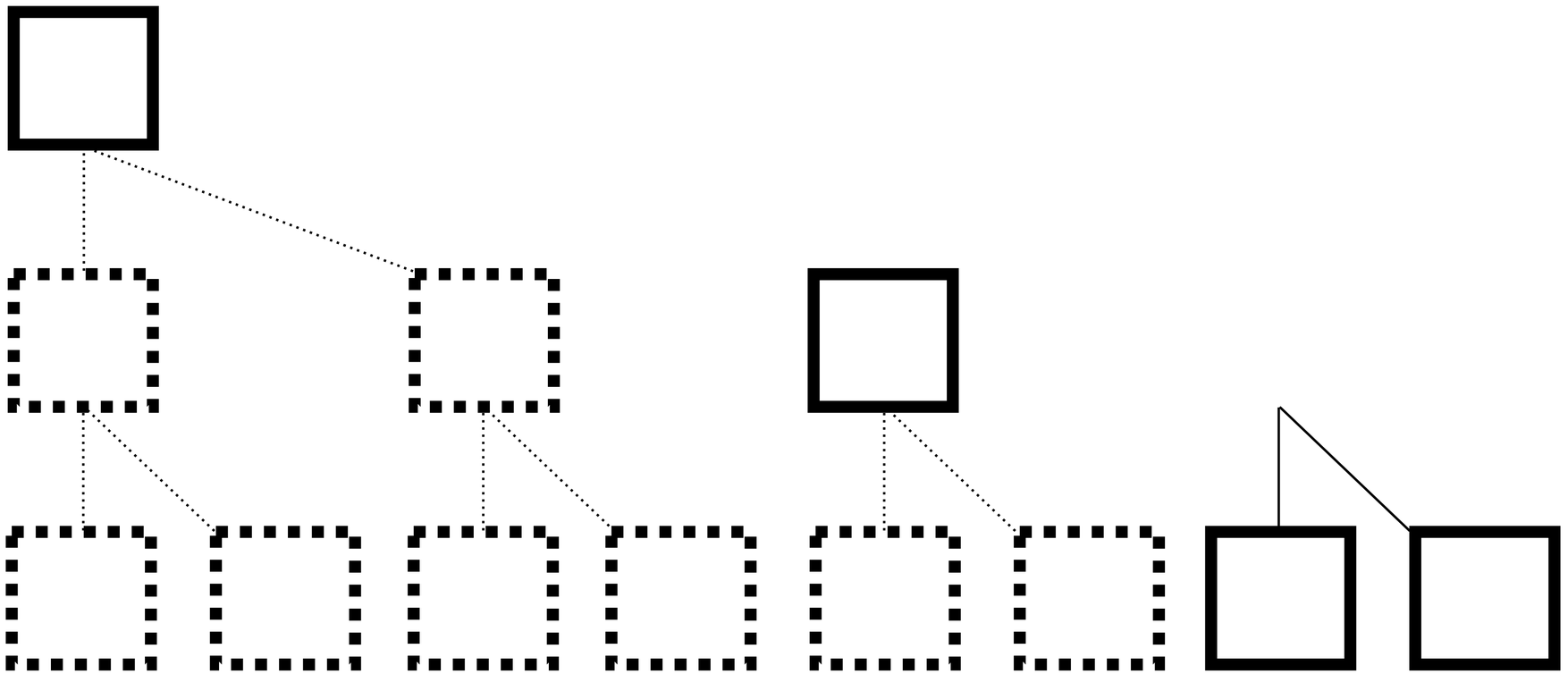
Another one



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

8 UTXOs

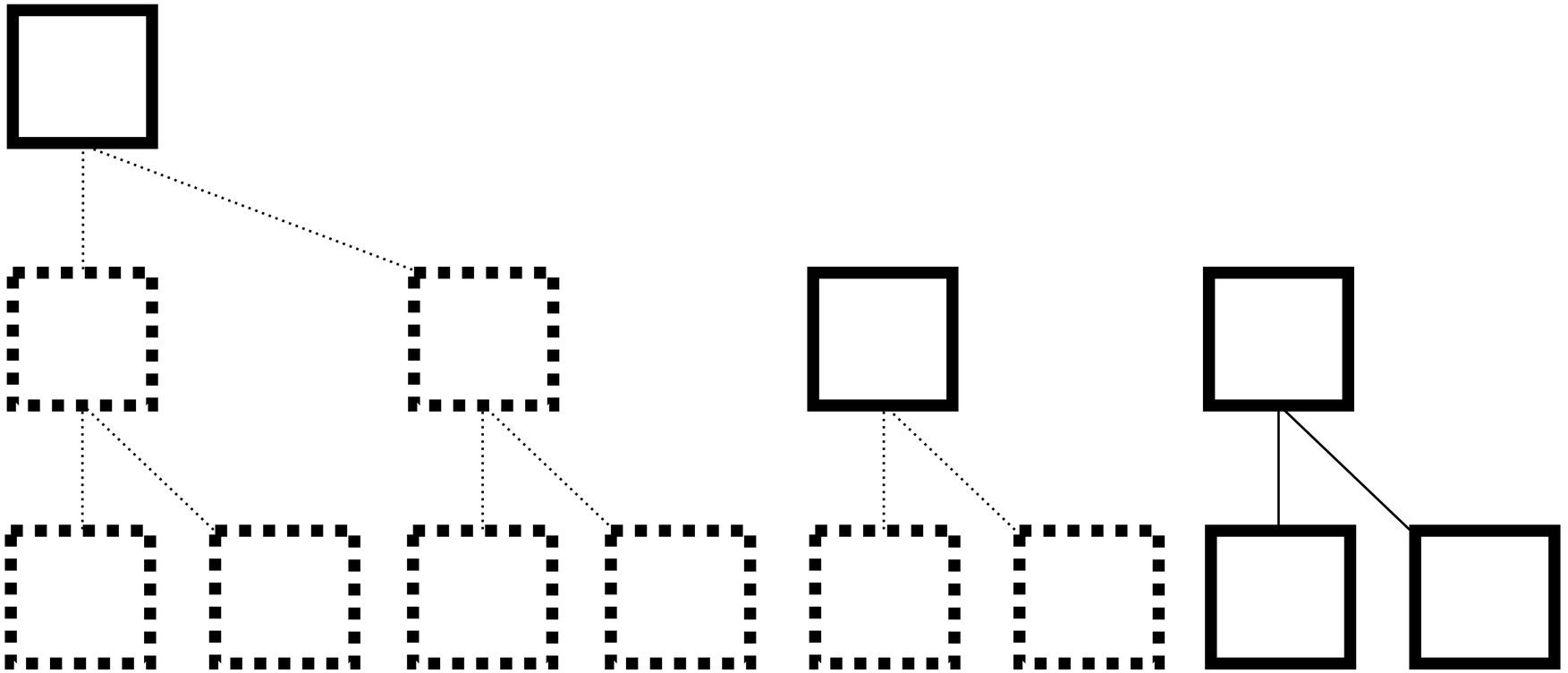
Concatenate



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

8 UTXOs

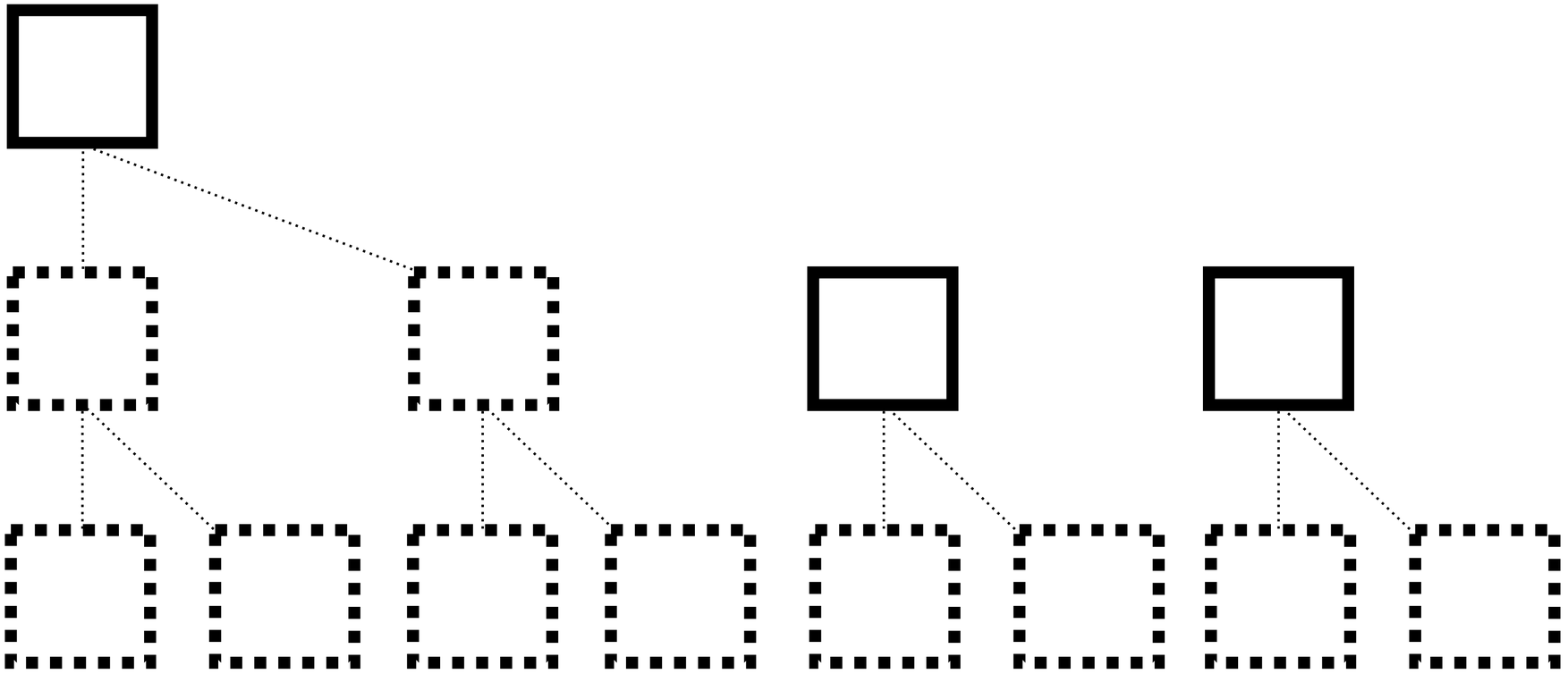
Hash



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

8 UTXOs

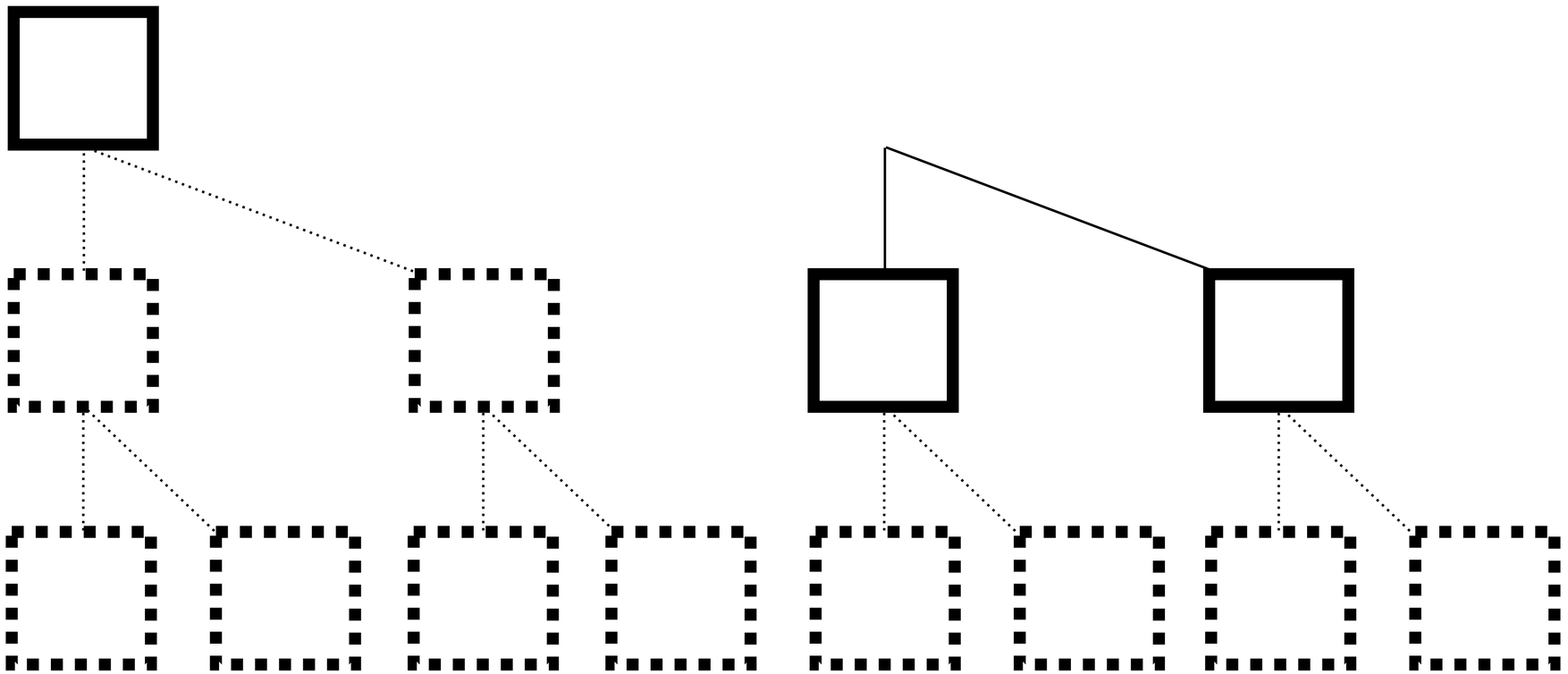
Throw away



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

8 UTXOs

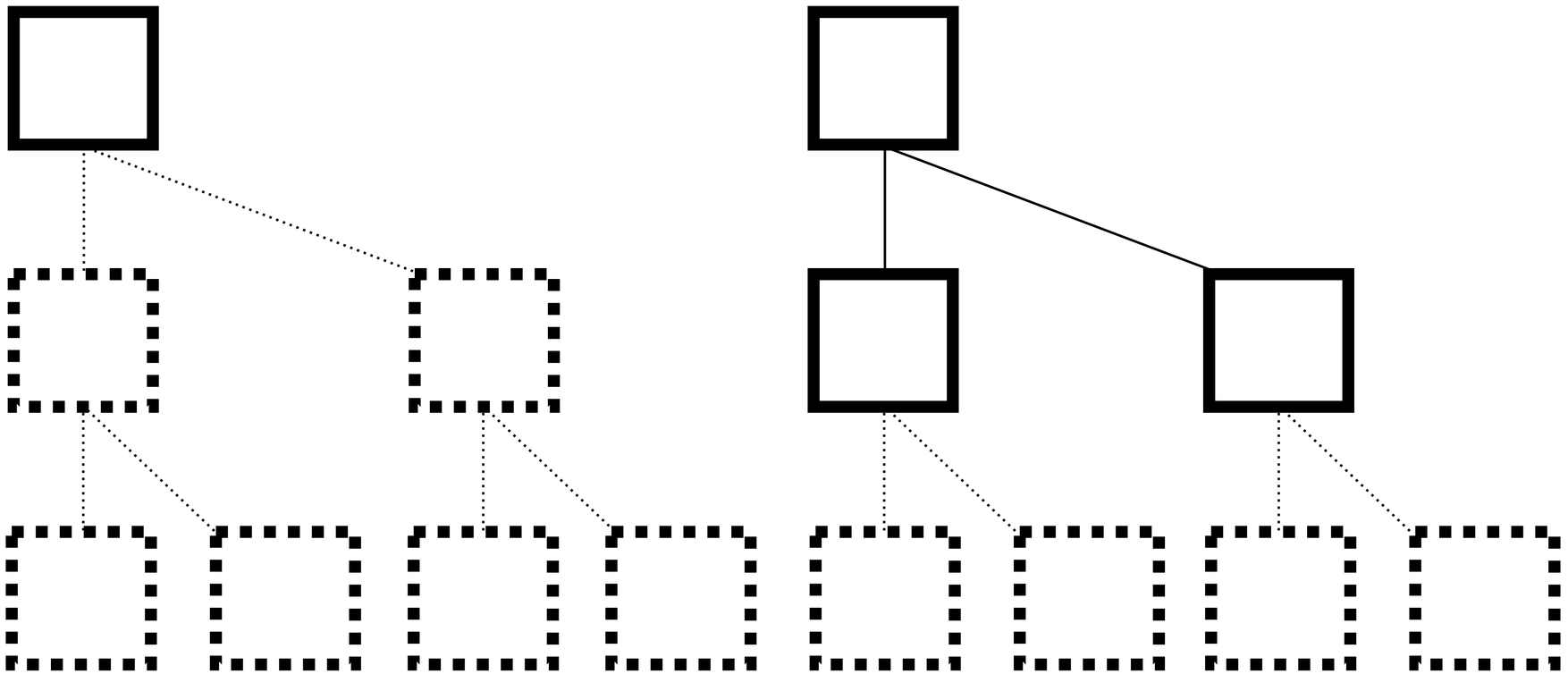
Concatenate



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

8 UTXOs

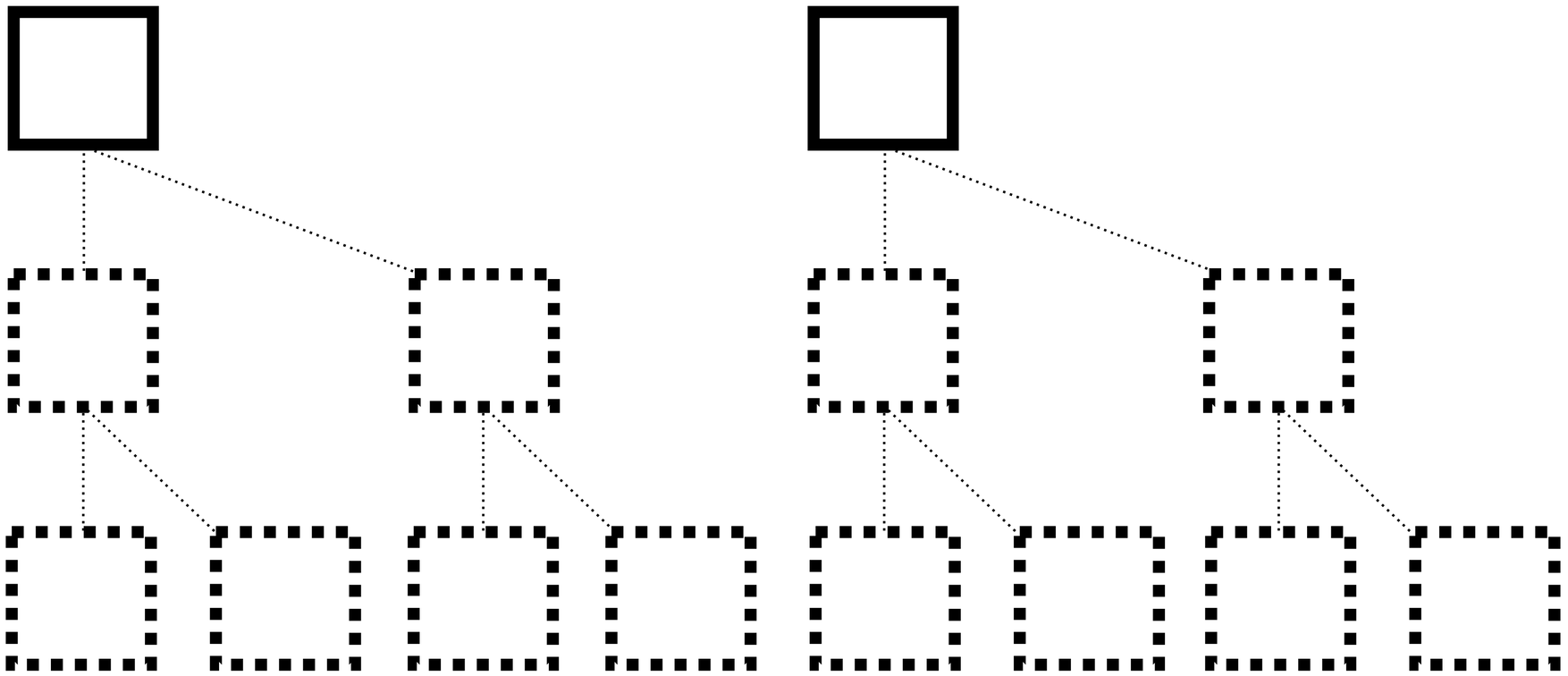
Hash



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

8 UTXOs

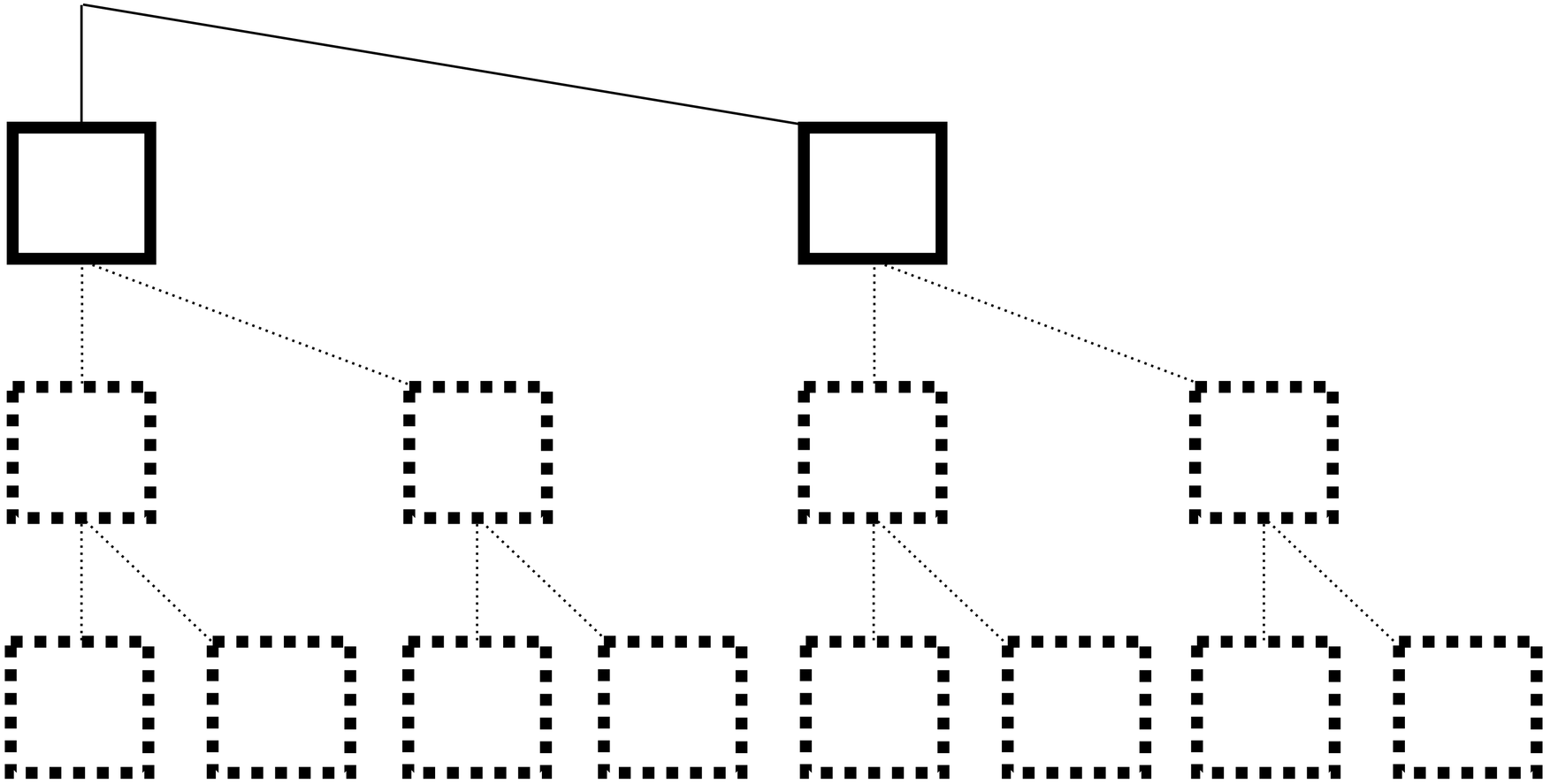
Throw away



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

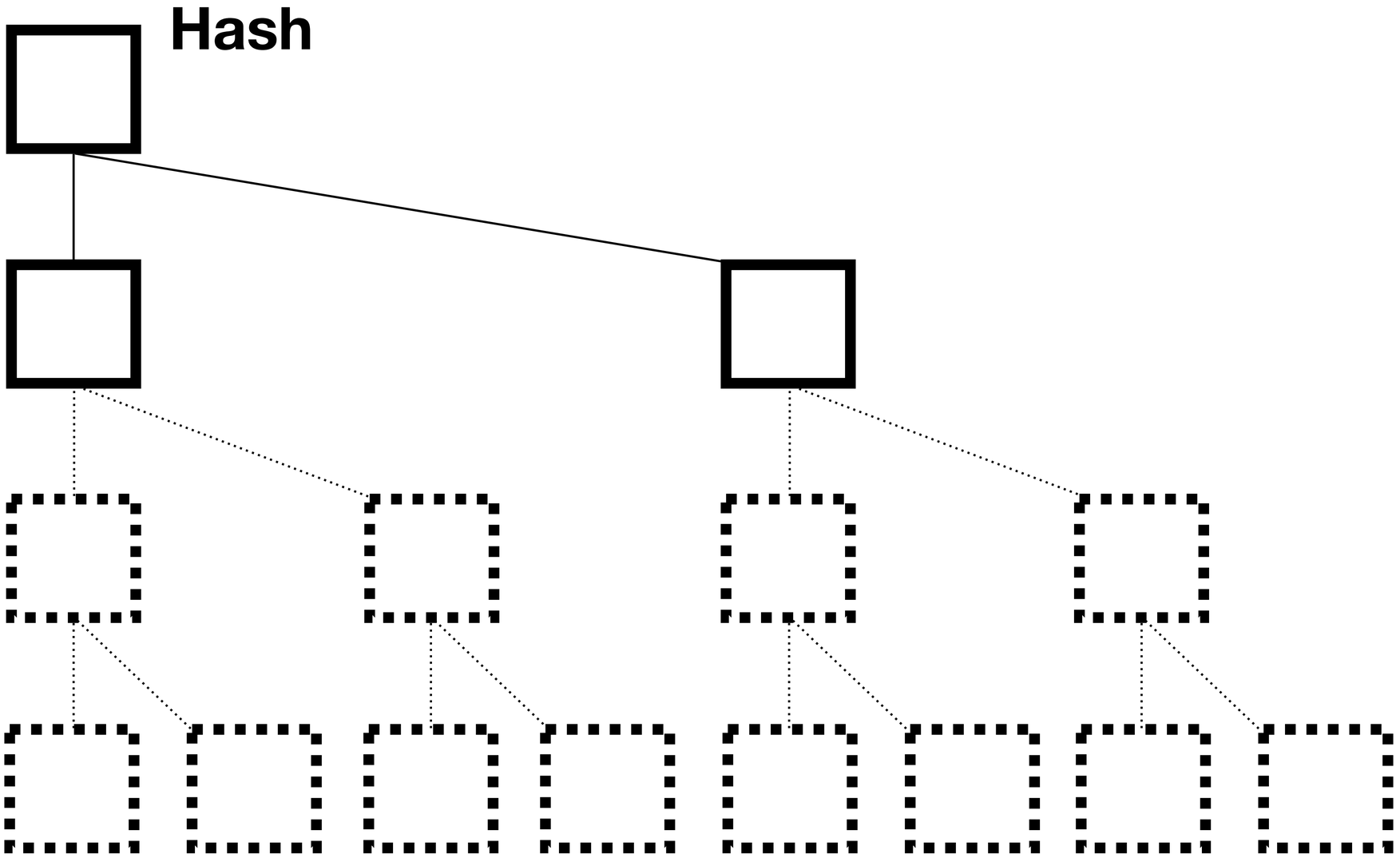
8 UTXOs

Concatenate



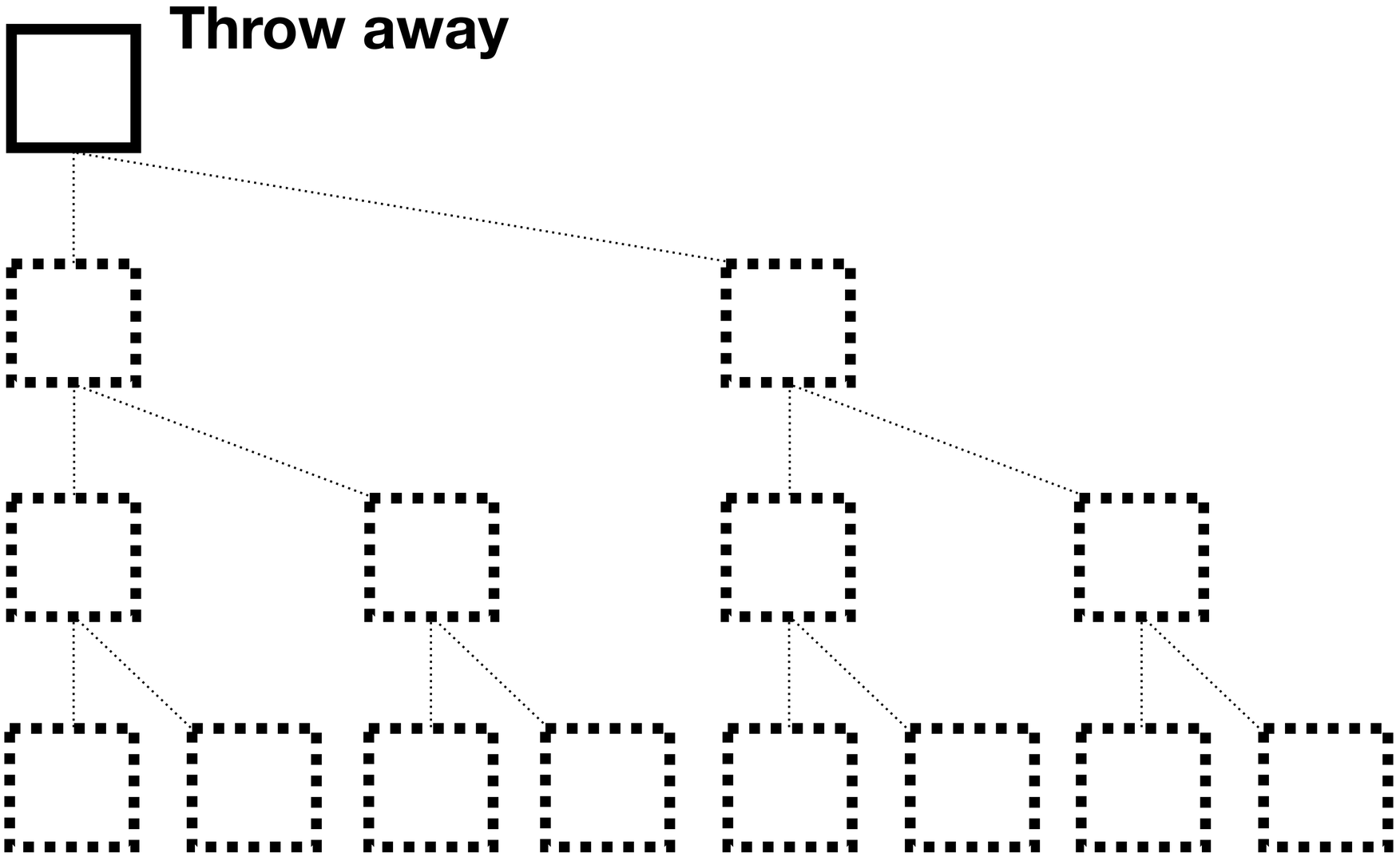
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

8 UTXOs



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

8 UTXOs



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

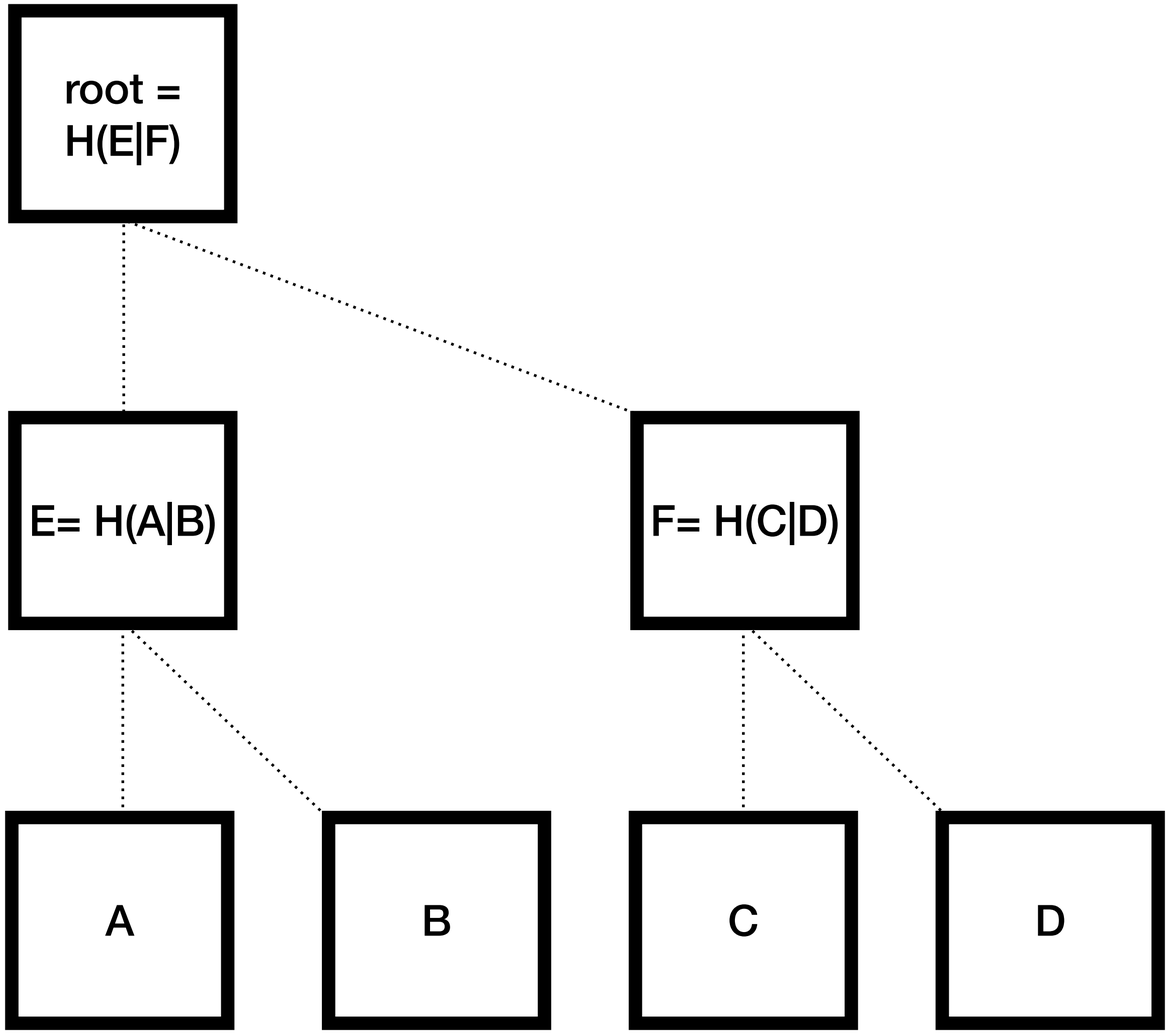
Algorithms

Consensus critical stuff

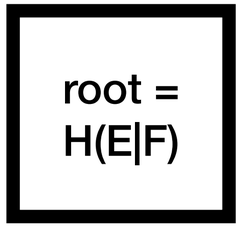
1. Add a UTXO
2. Delete a UTXO
3. Verify the existence

Deletion happens with verifying the existence

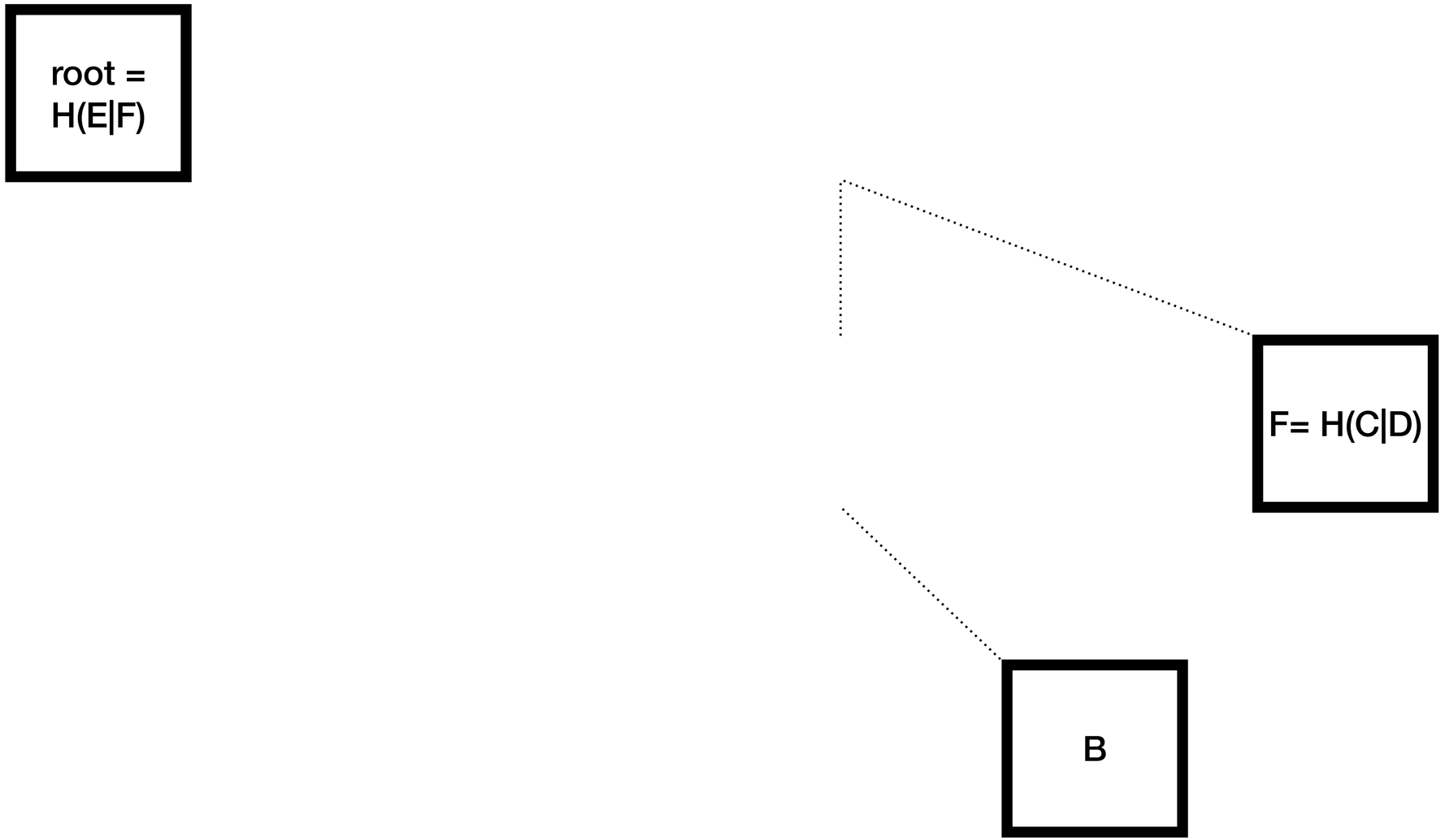
Quick merkle tree prove review



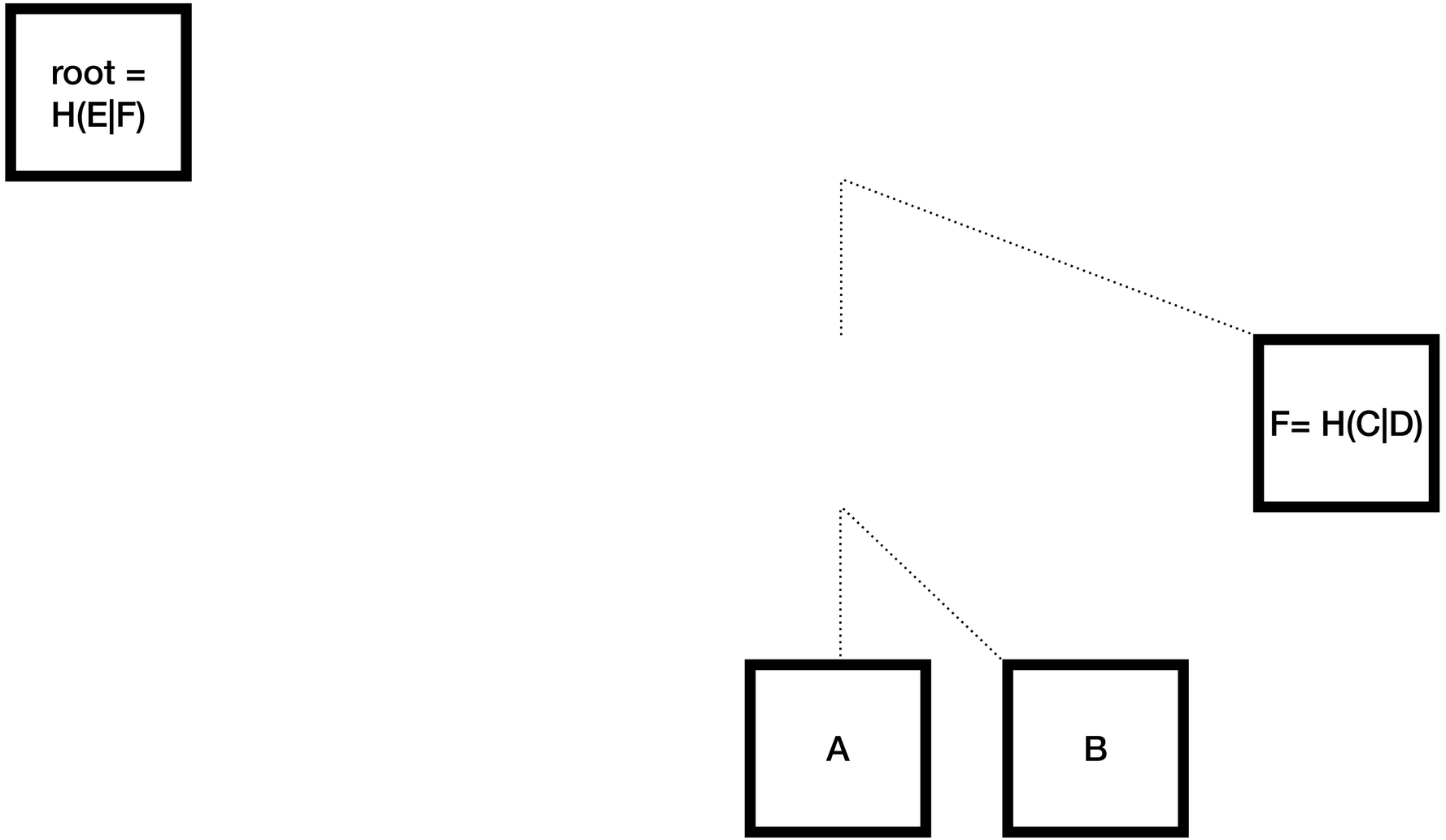
Proving A



Receive proof: B, F

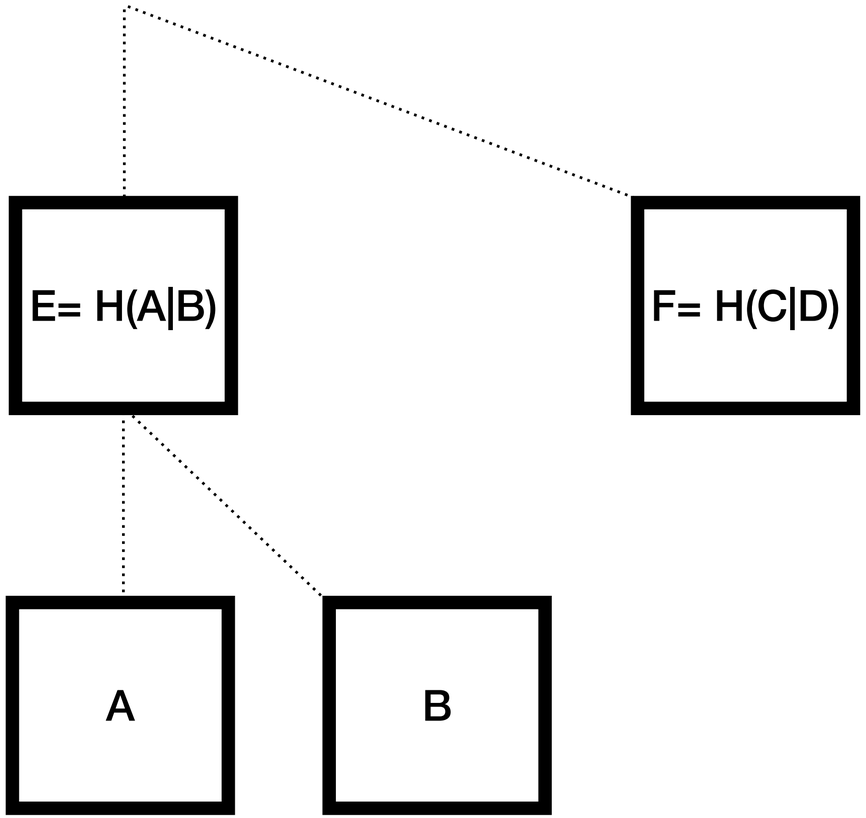


Fetch A



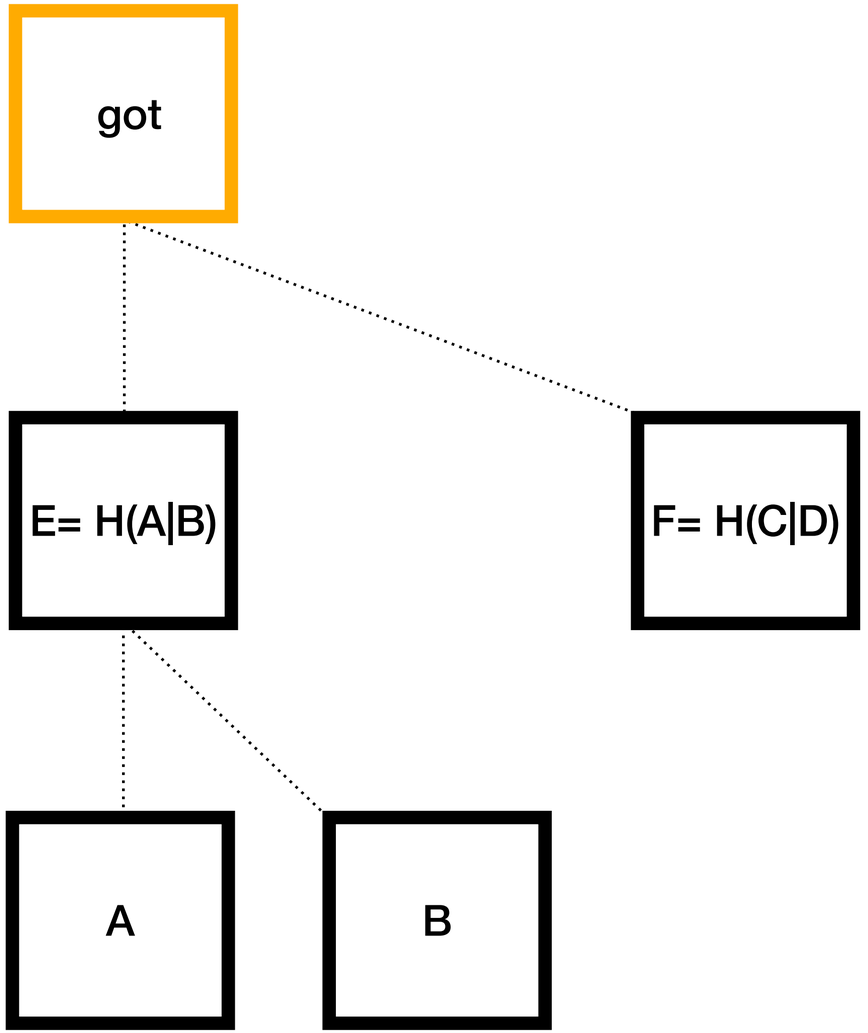
Calculate E

root =
H(E|F)



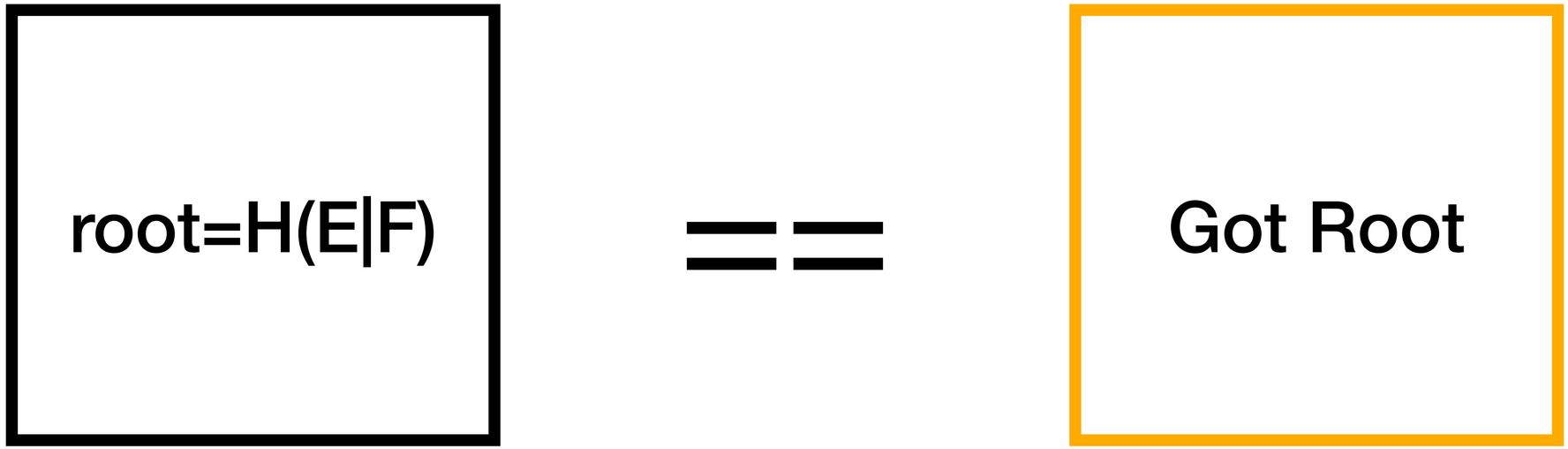
Calculate root

root =
H(E|F)



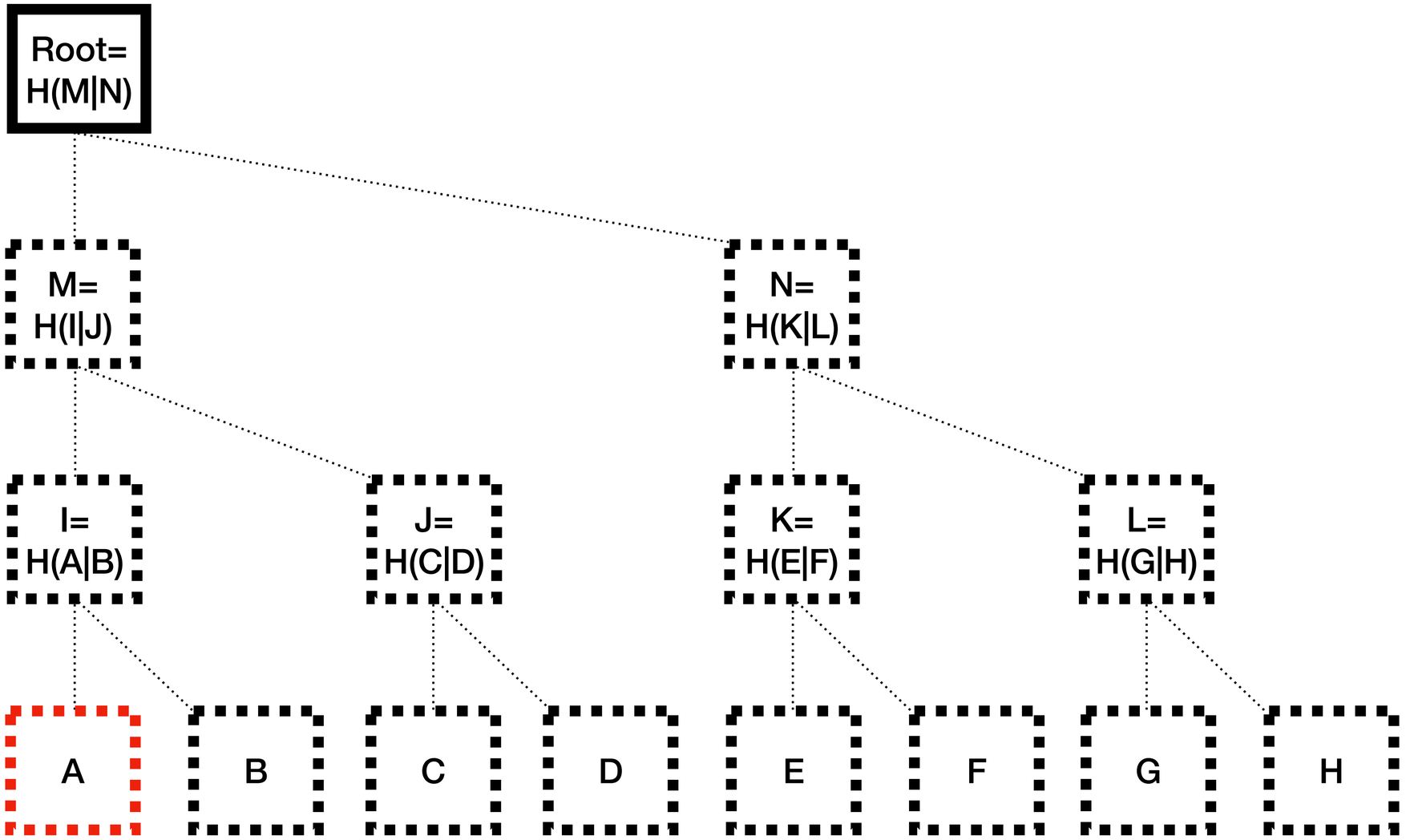
Compare roots

Proof is valid if root is the same



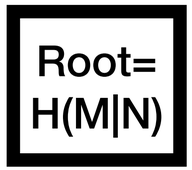
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Deleting A



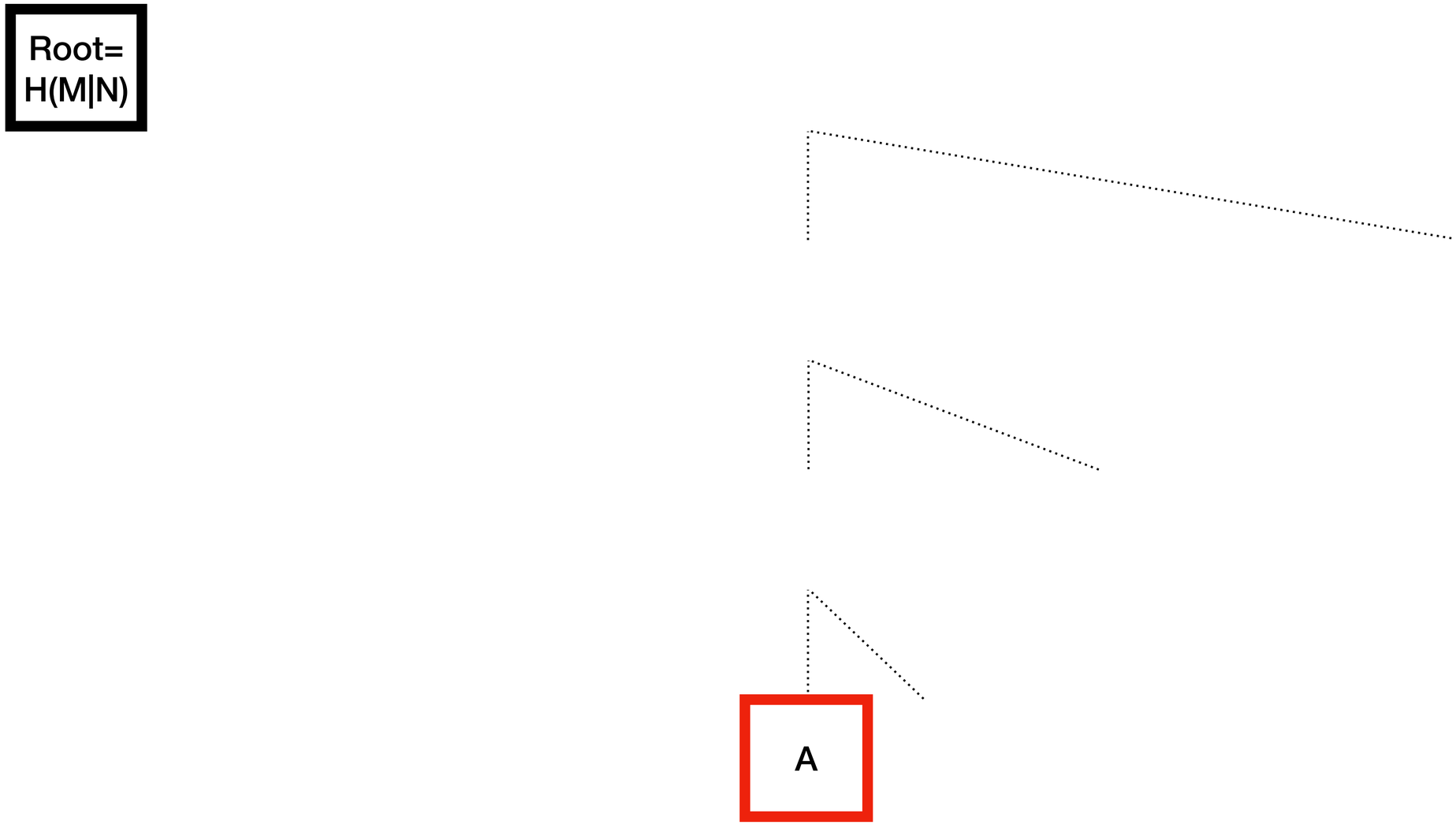
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

The only requirement is the roots



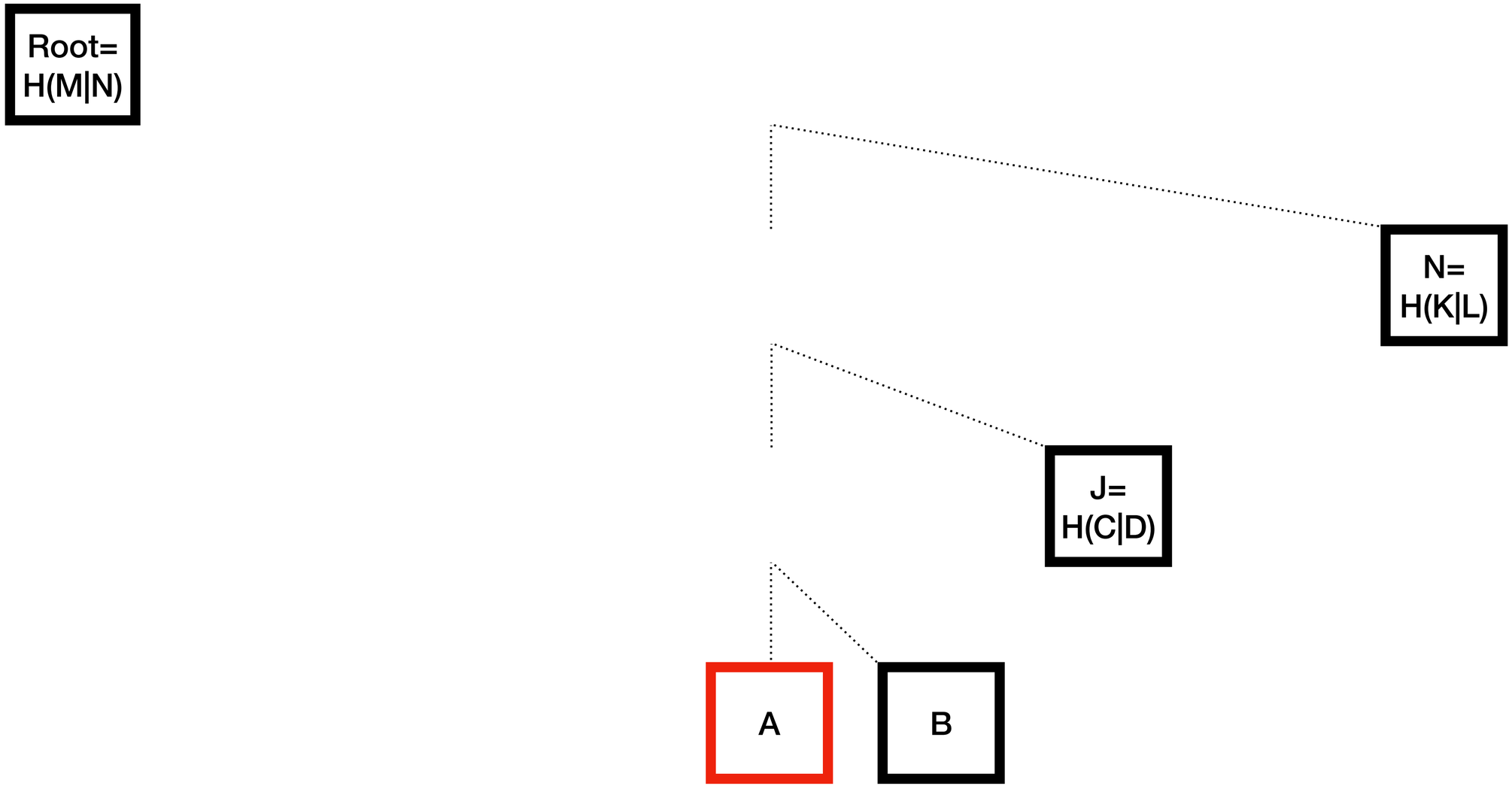
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate A from the Bitcoin Block



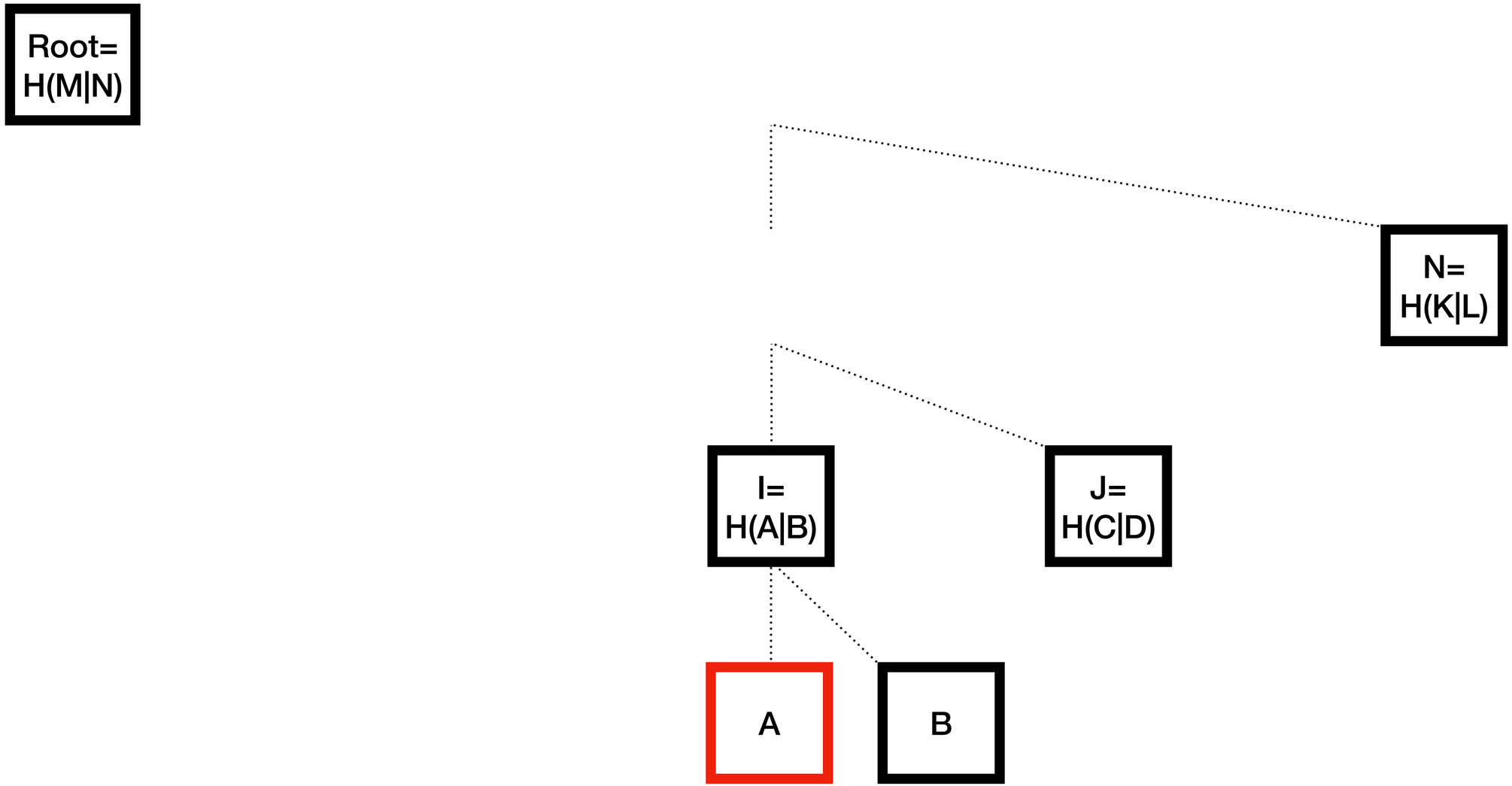
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Receive the proof: B, J, N



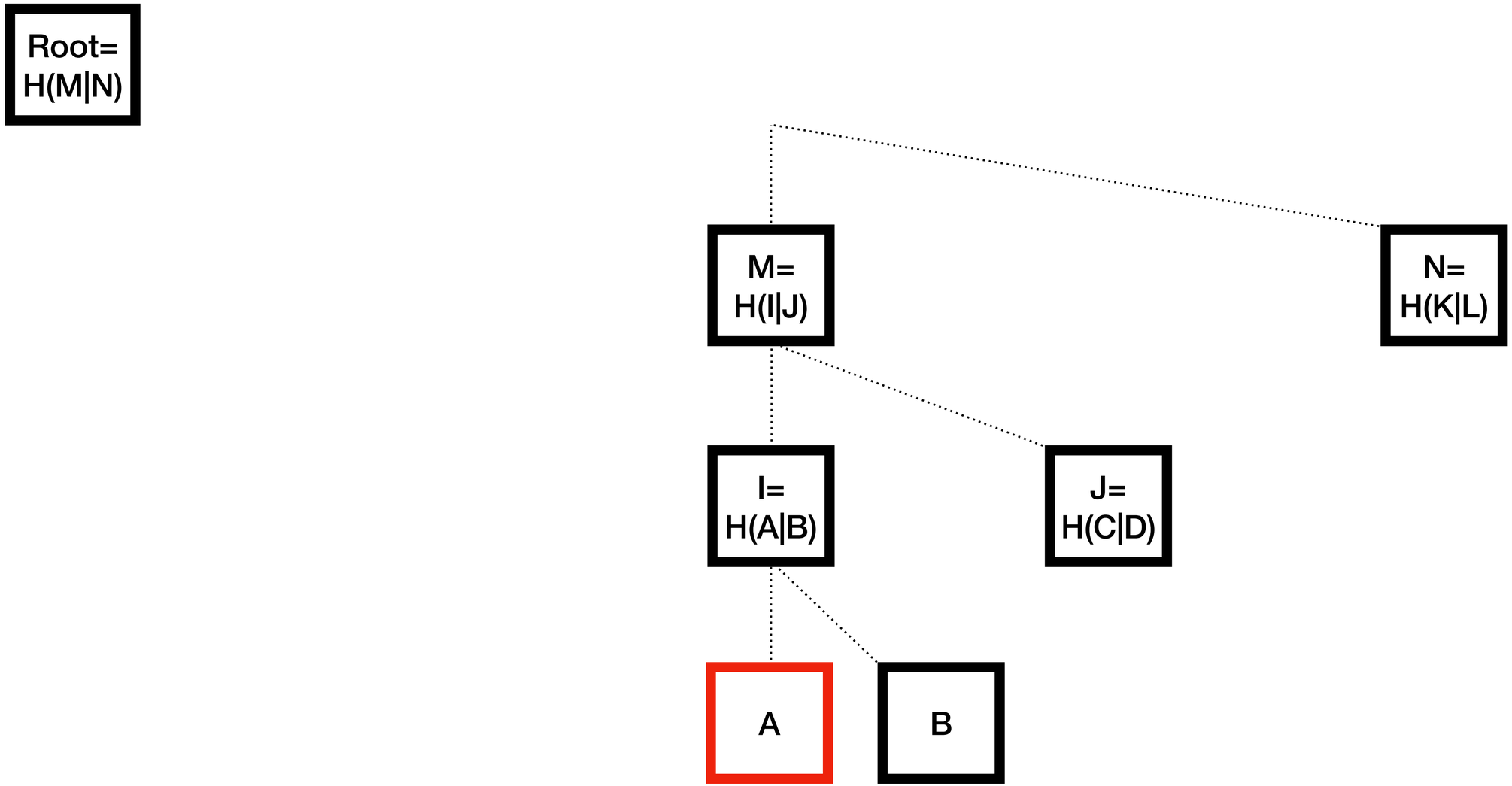
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate the hash for I



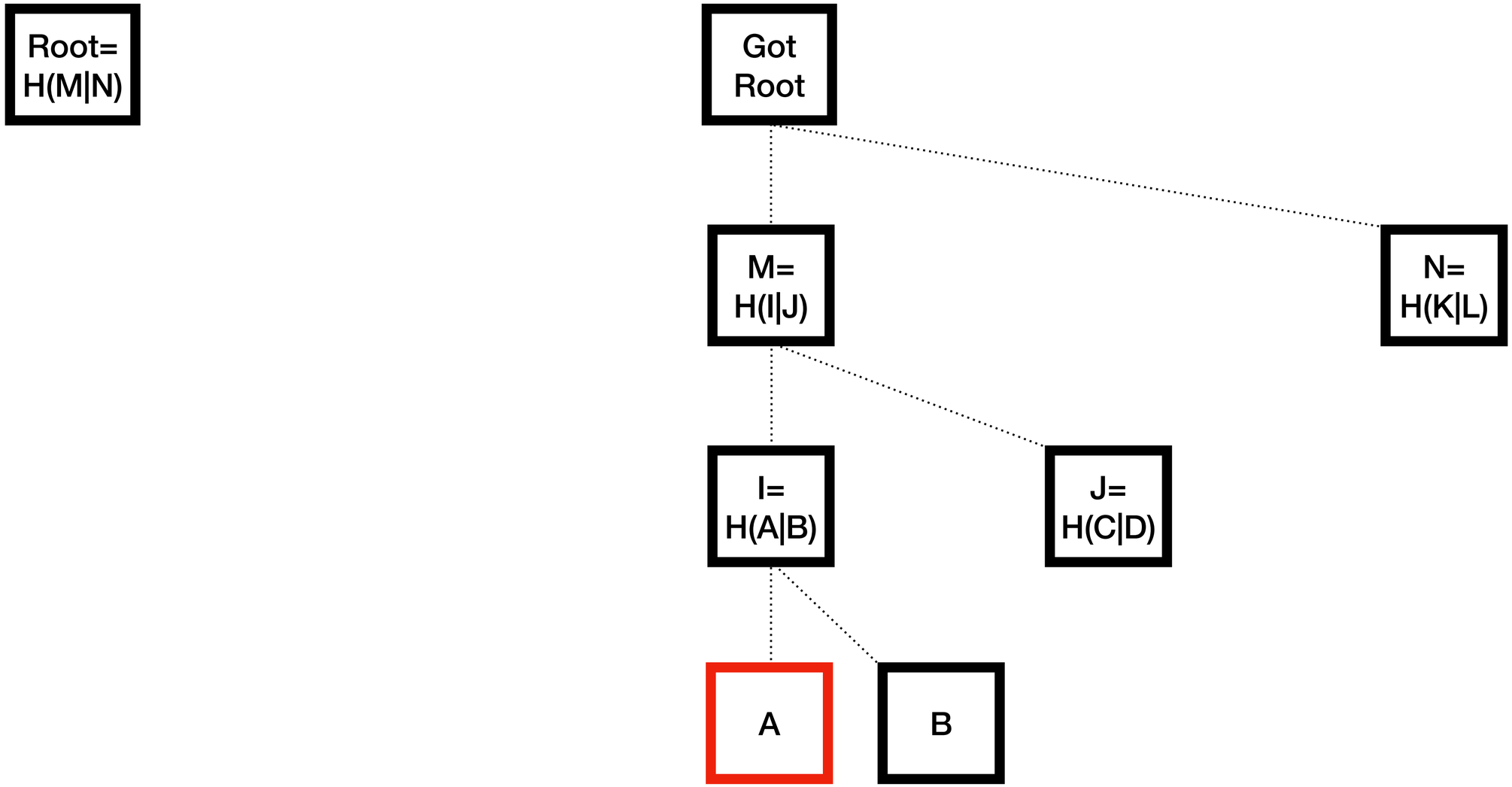
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate the hash for M



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

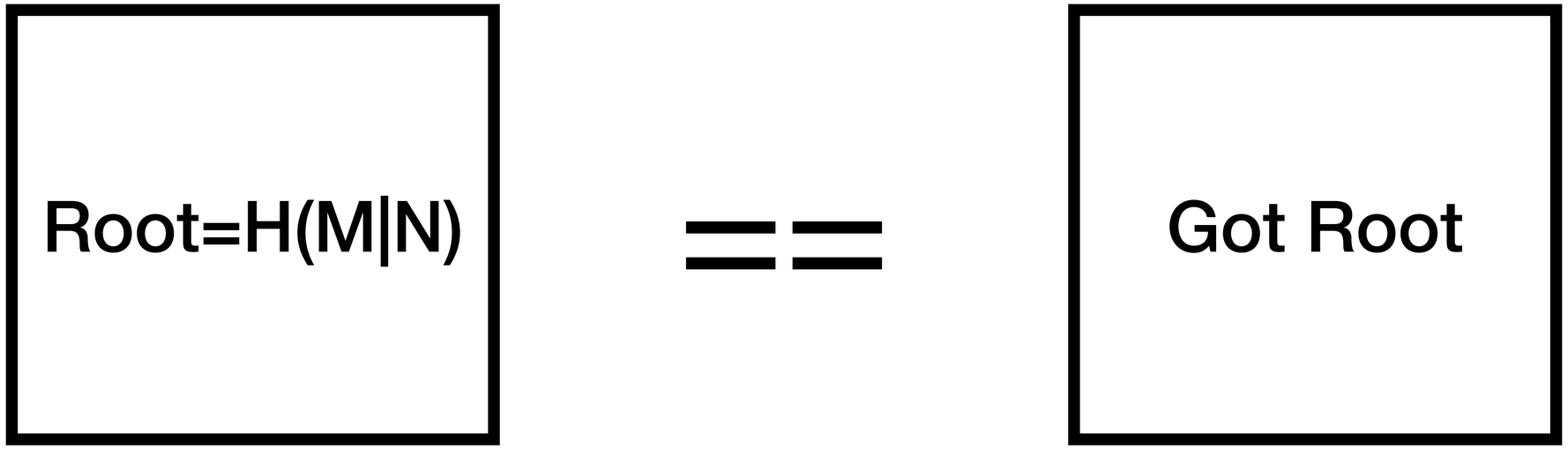
Calculate Root



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

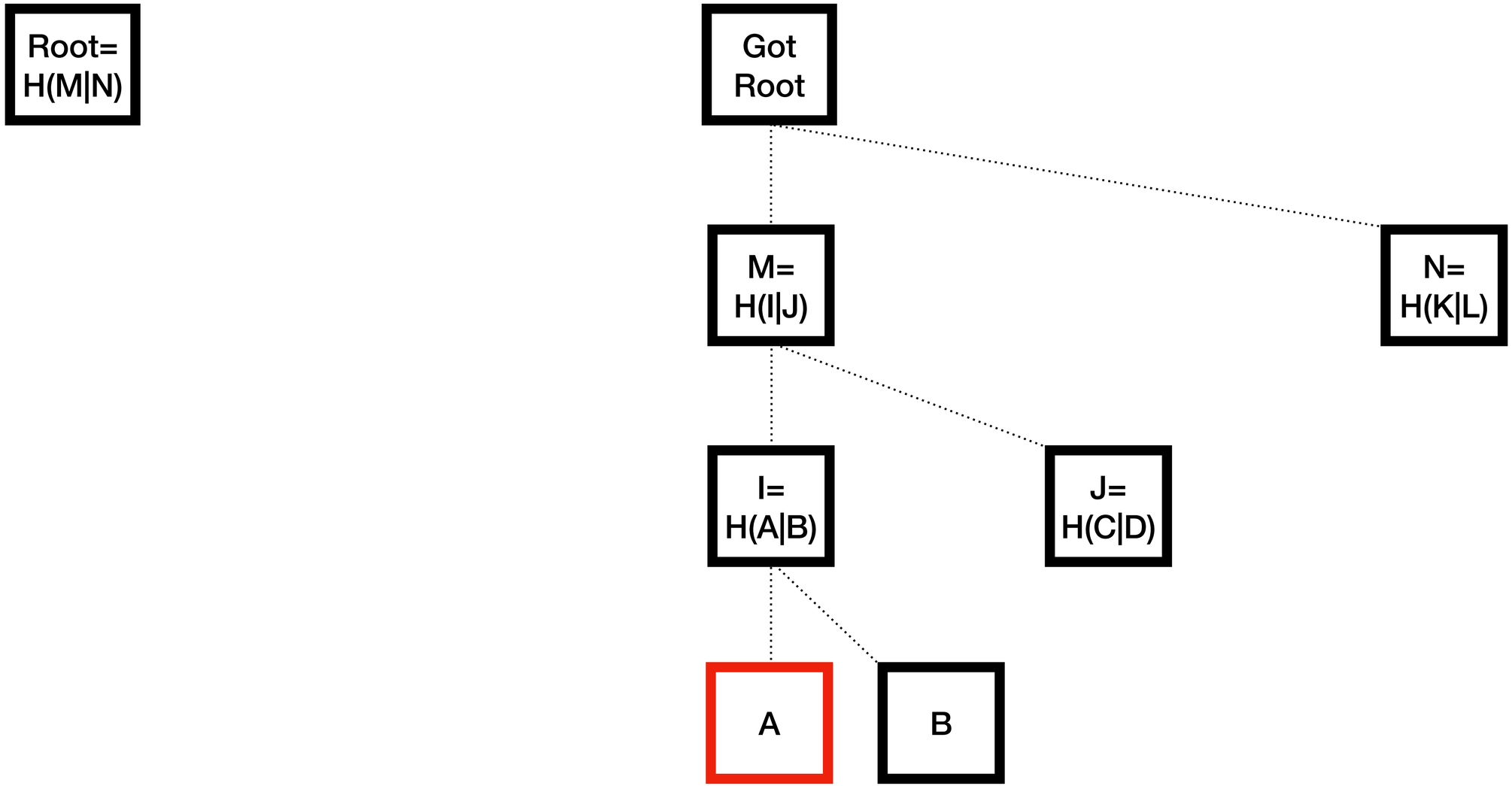
Compare roots

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



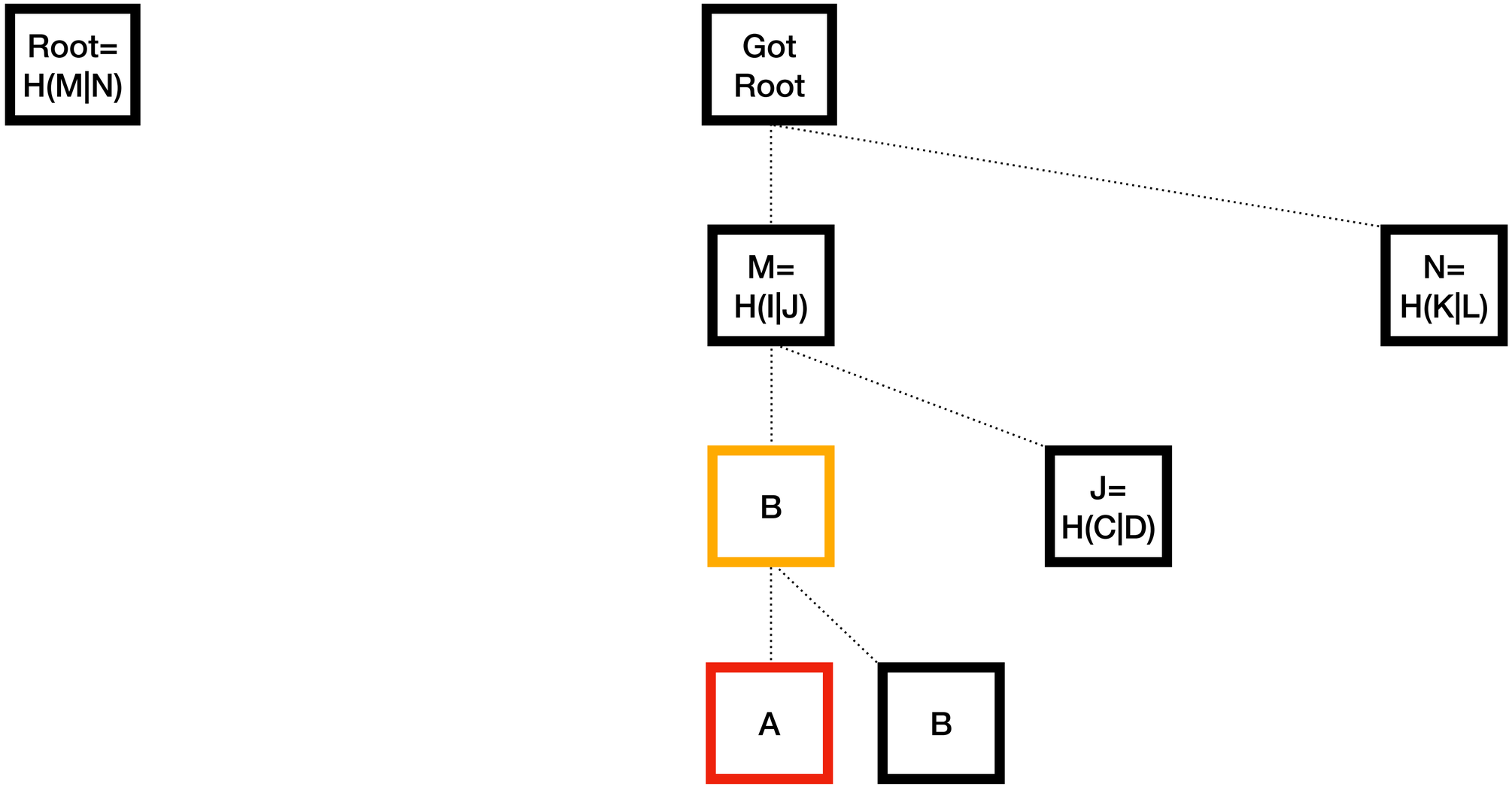
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate root after deletion



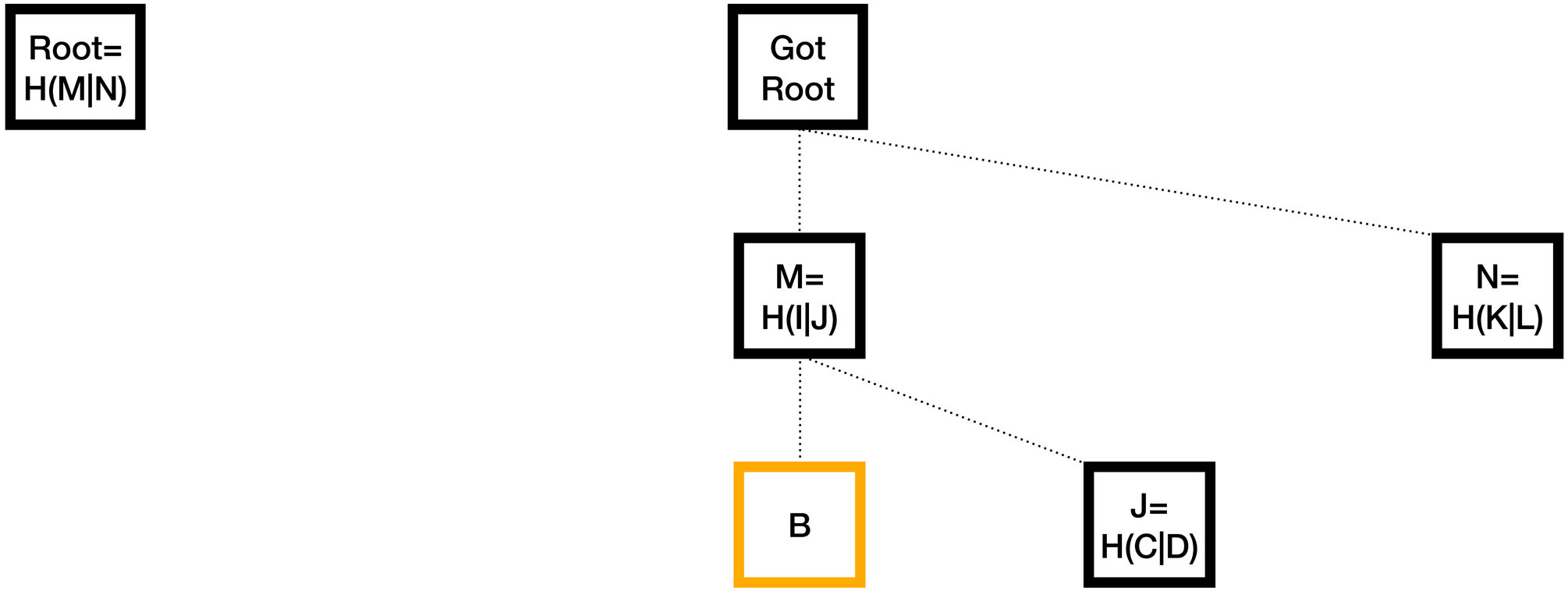
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Move up B to I



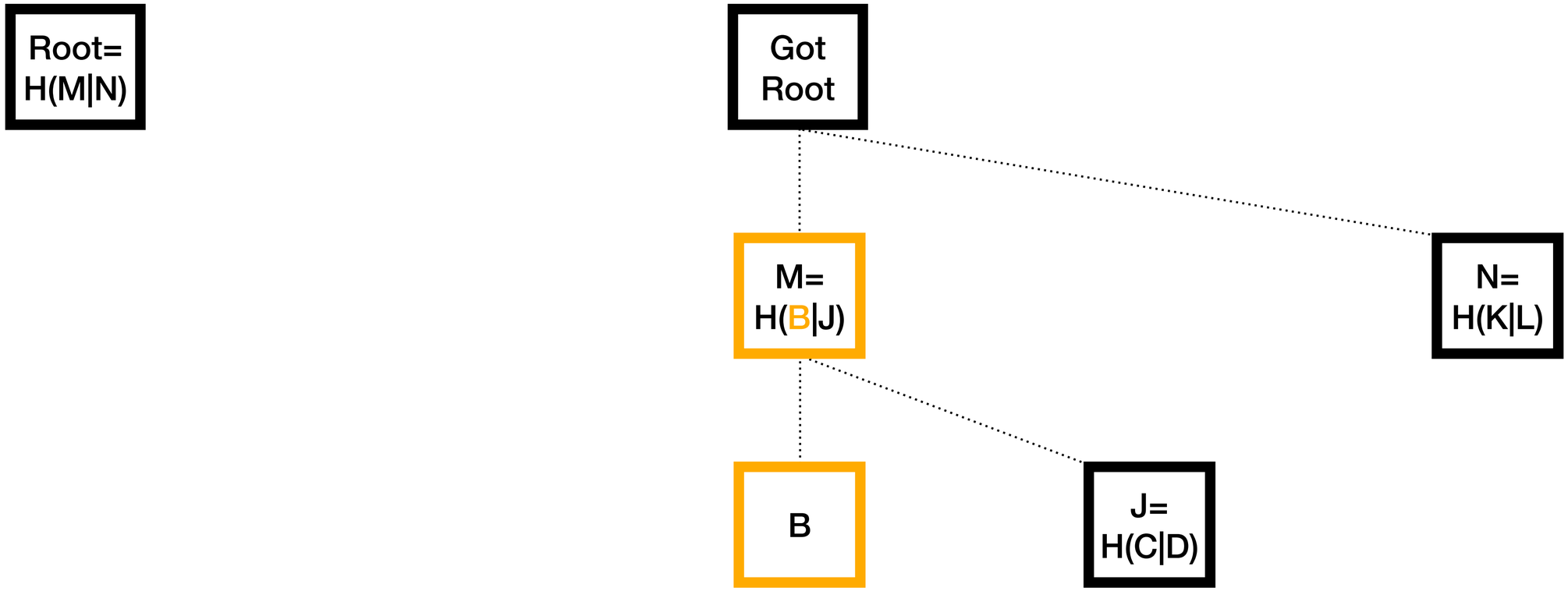
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Remove old nodes for A&B



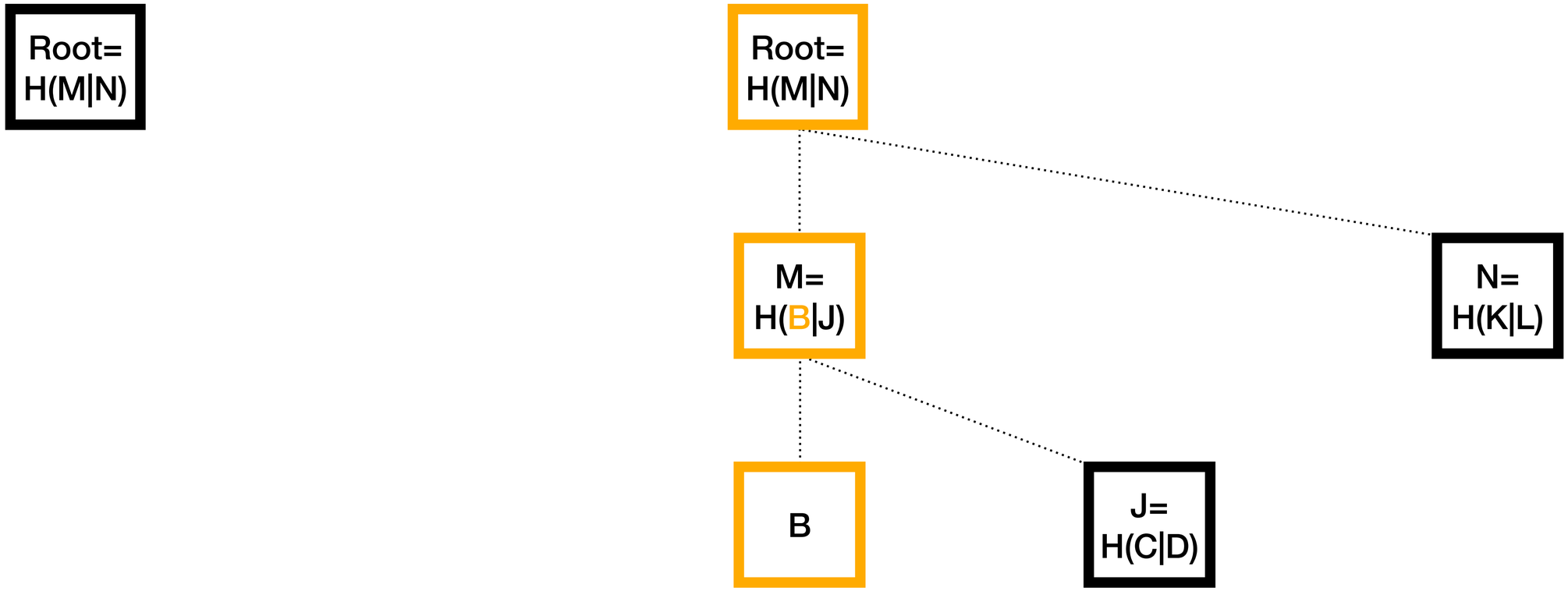
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate H(B|J)



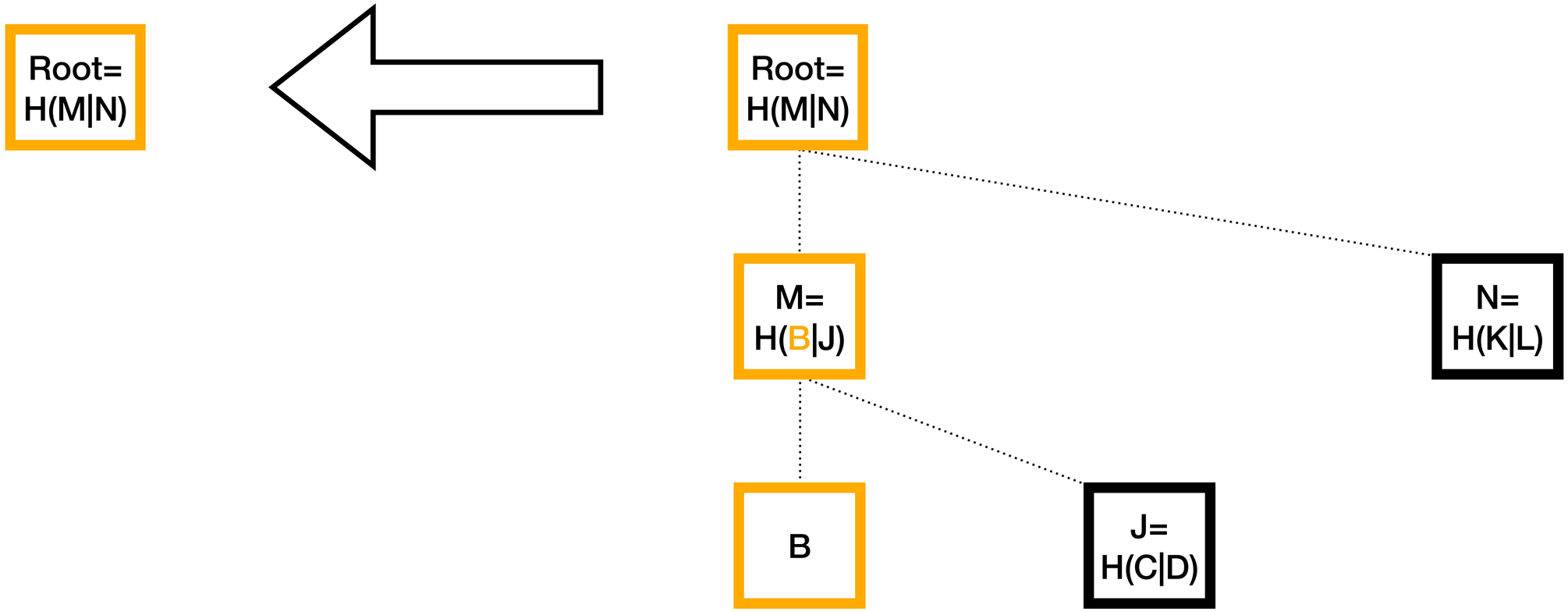
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate new root



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

Copy over the new root to be saved



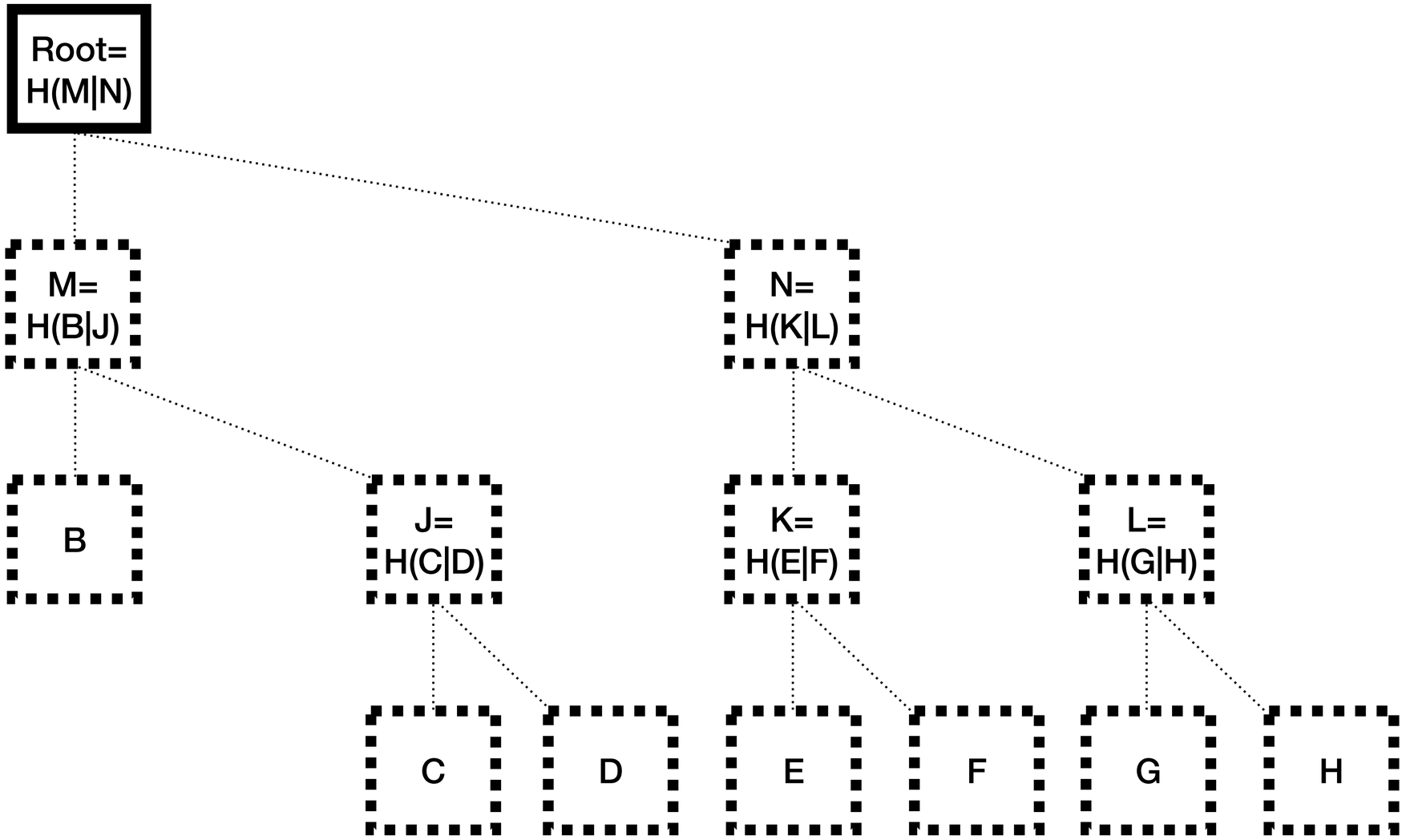
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Done

Root=
H(M|N)

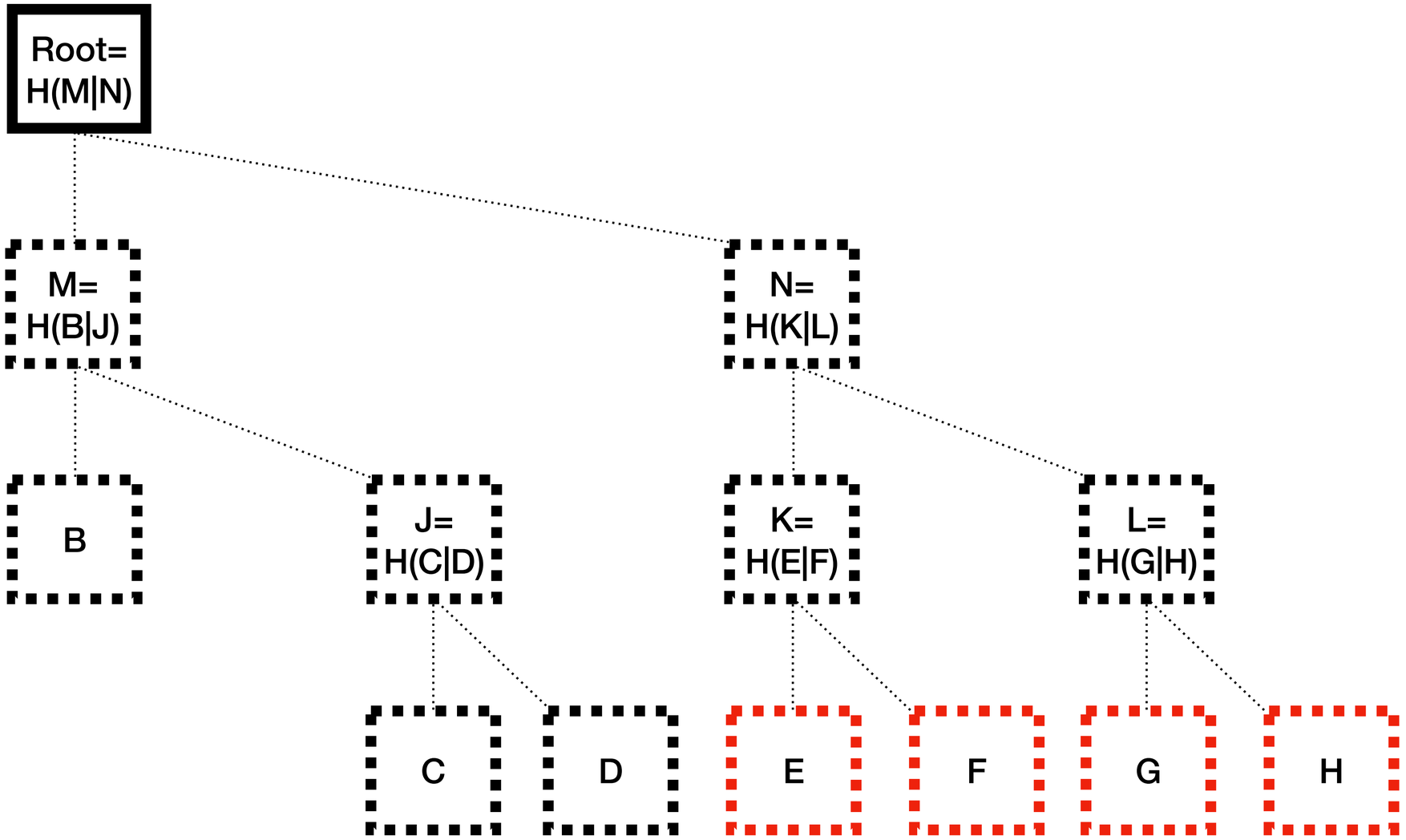
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

After deleting A



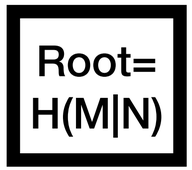
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

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



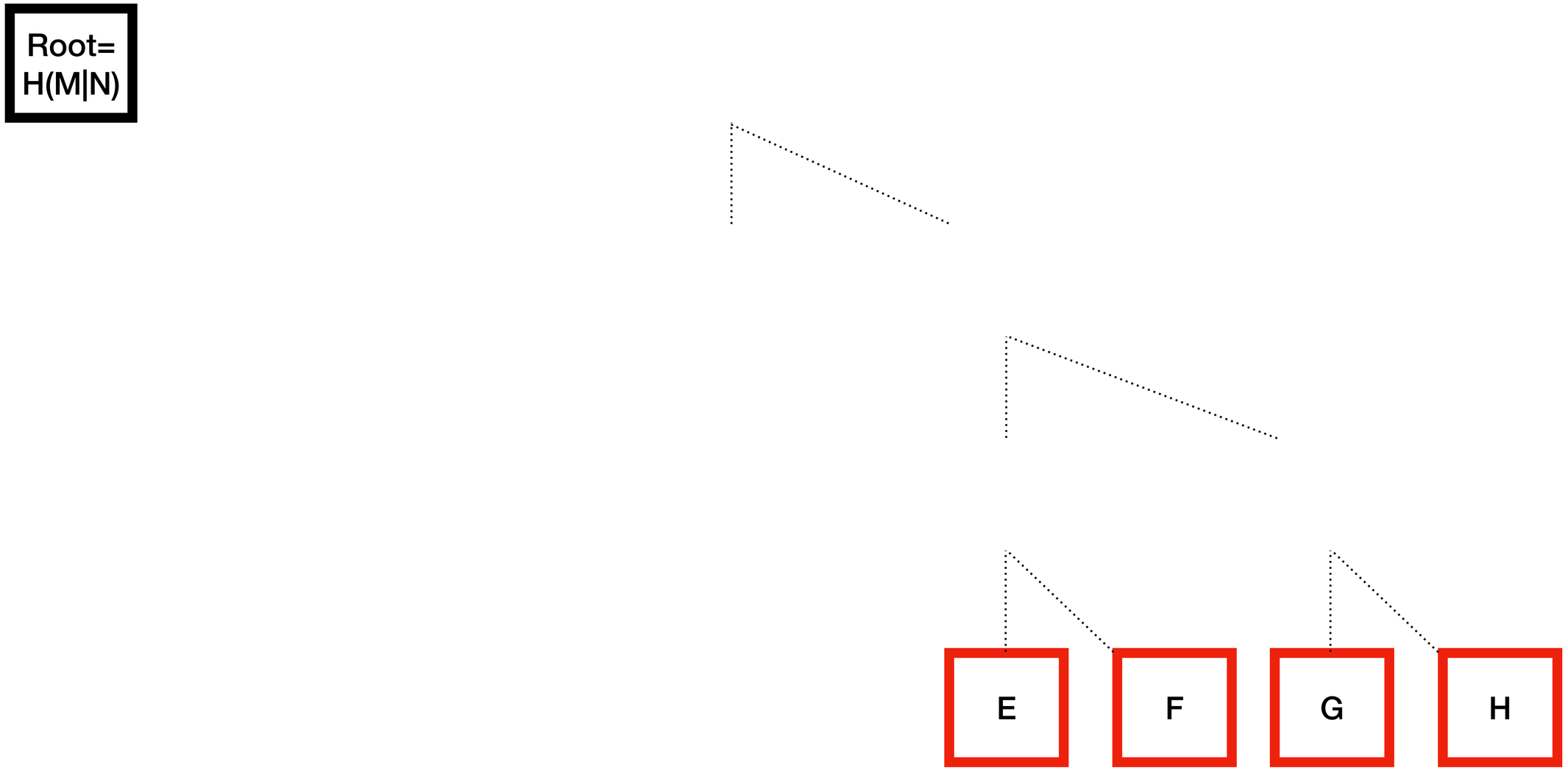
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Start off with only the root



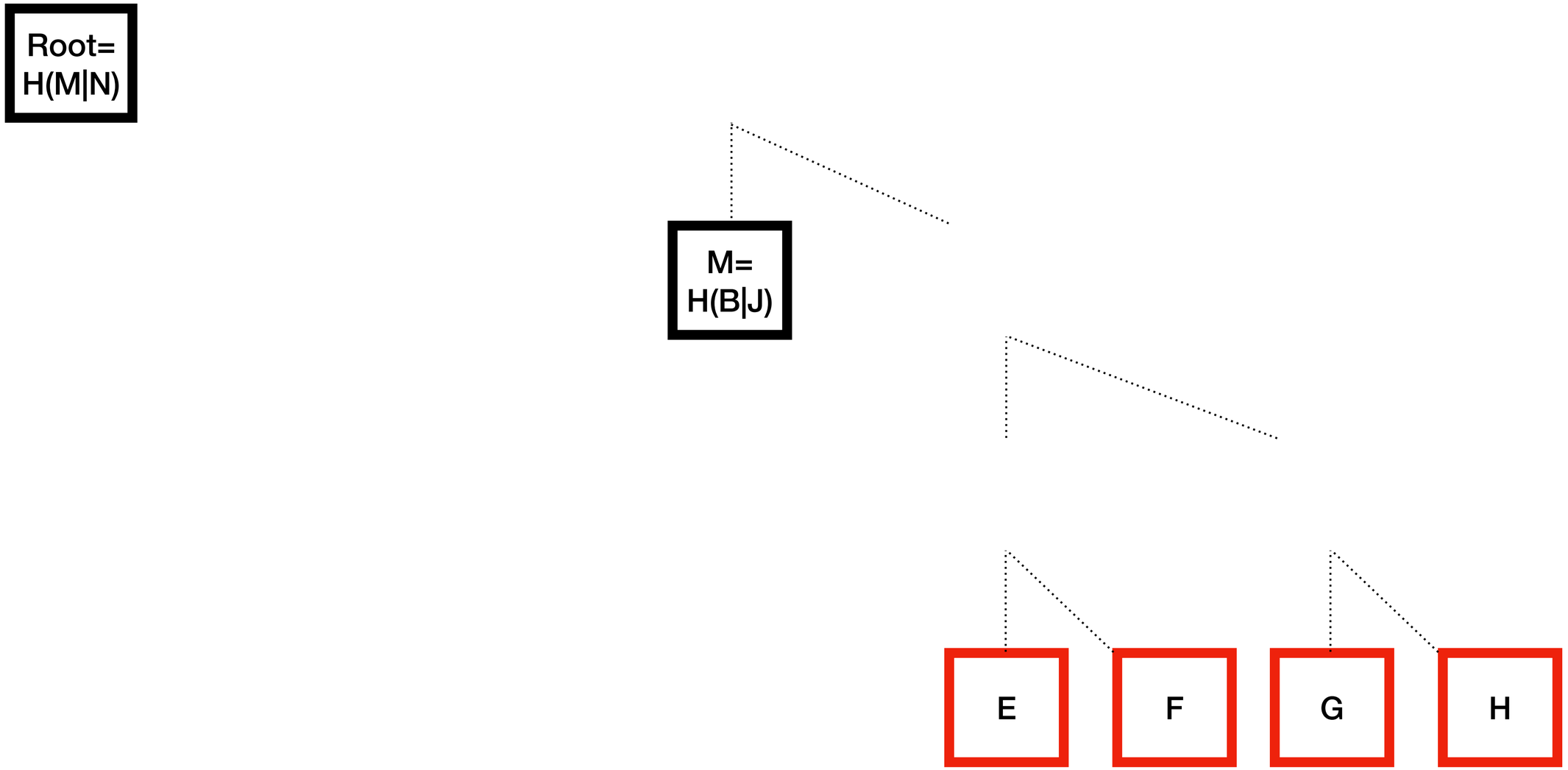
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

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



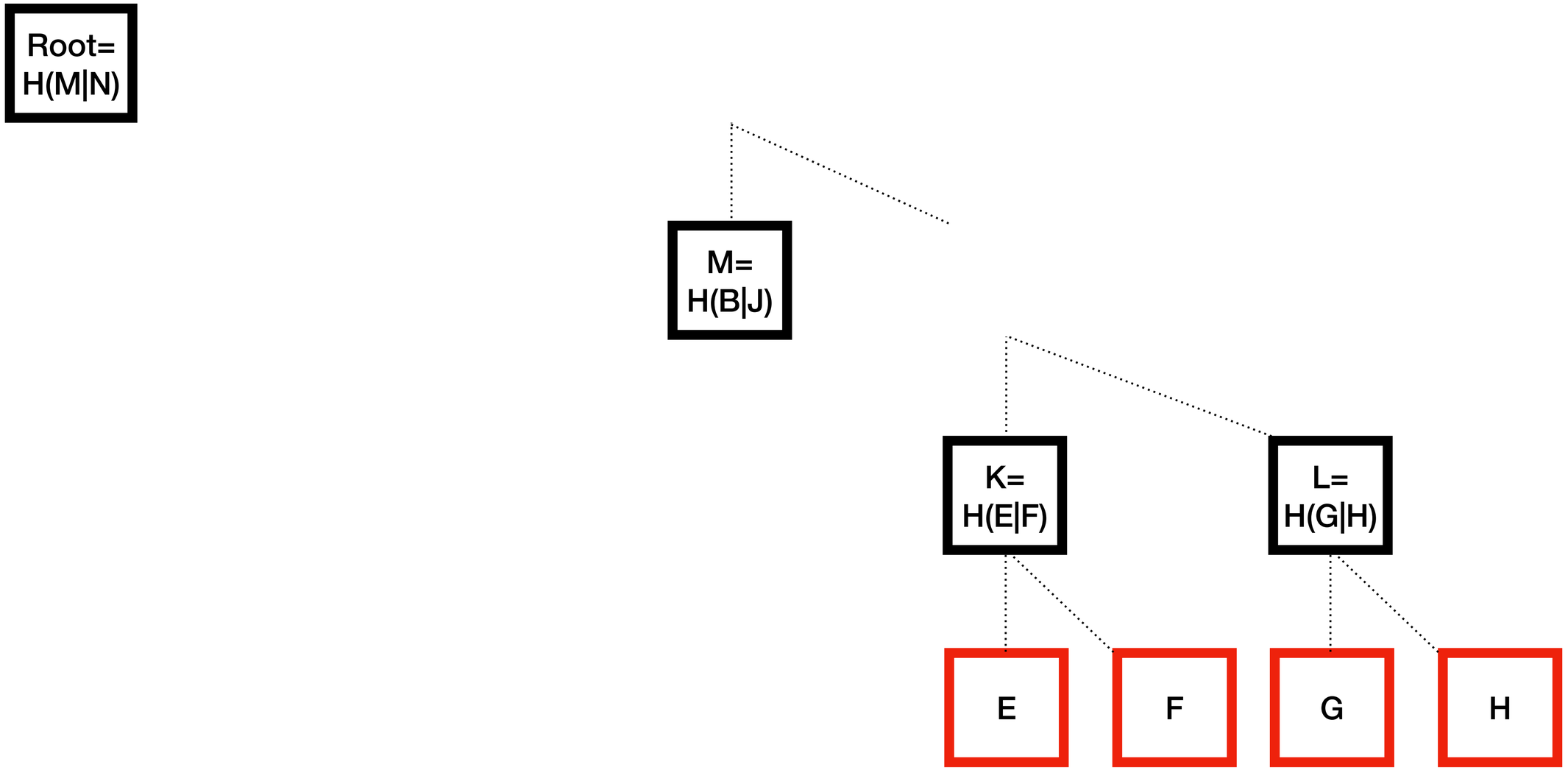
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Receive the proof: M



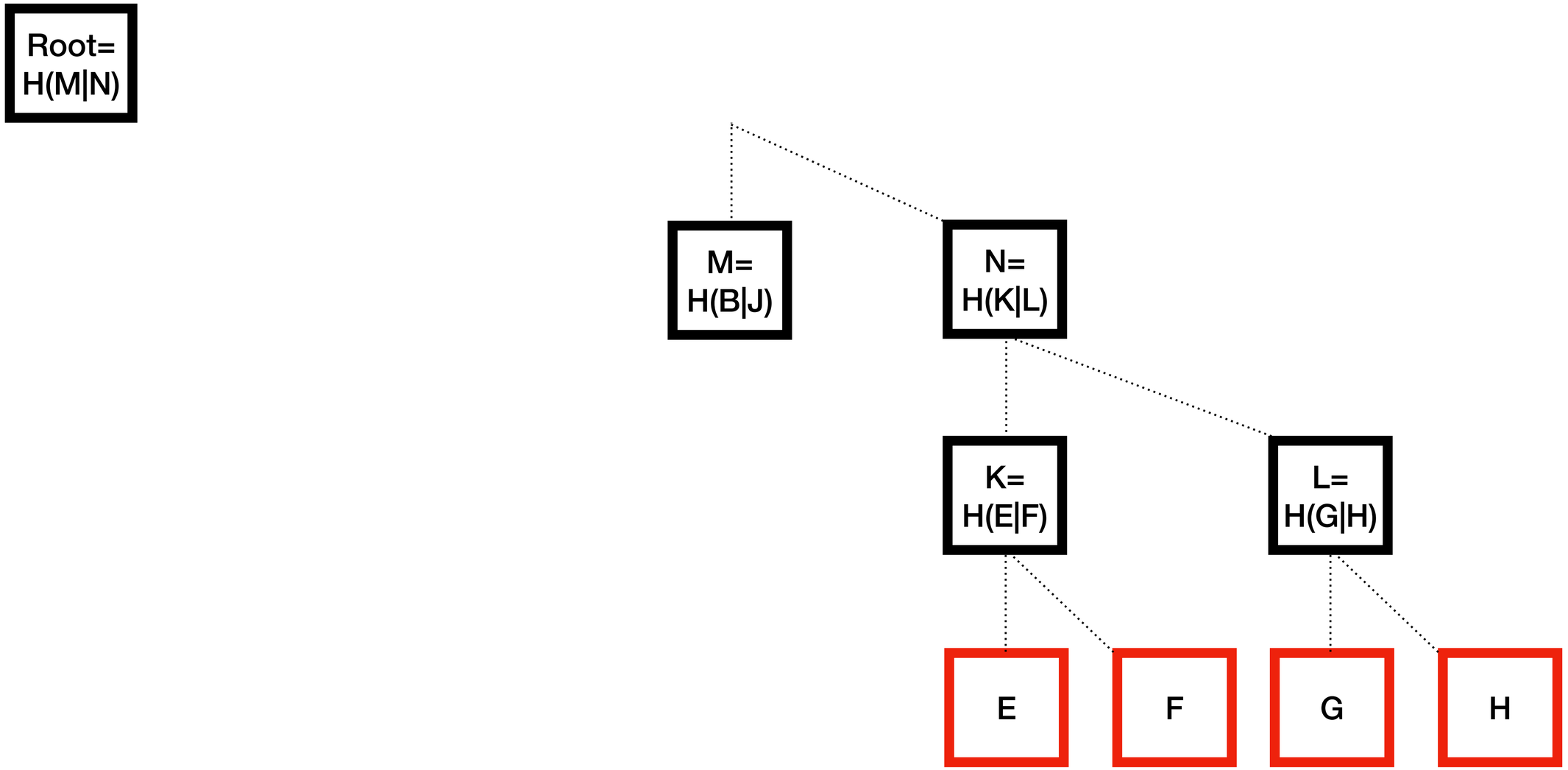
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate K and L



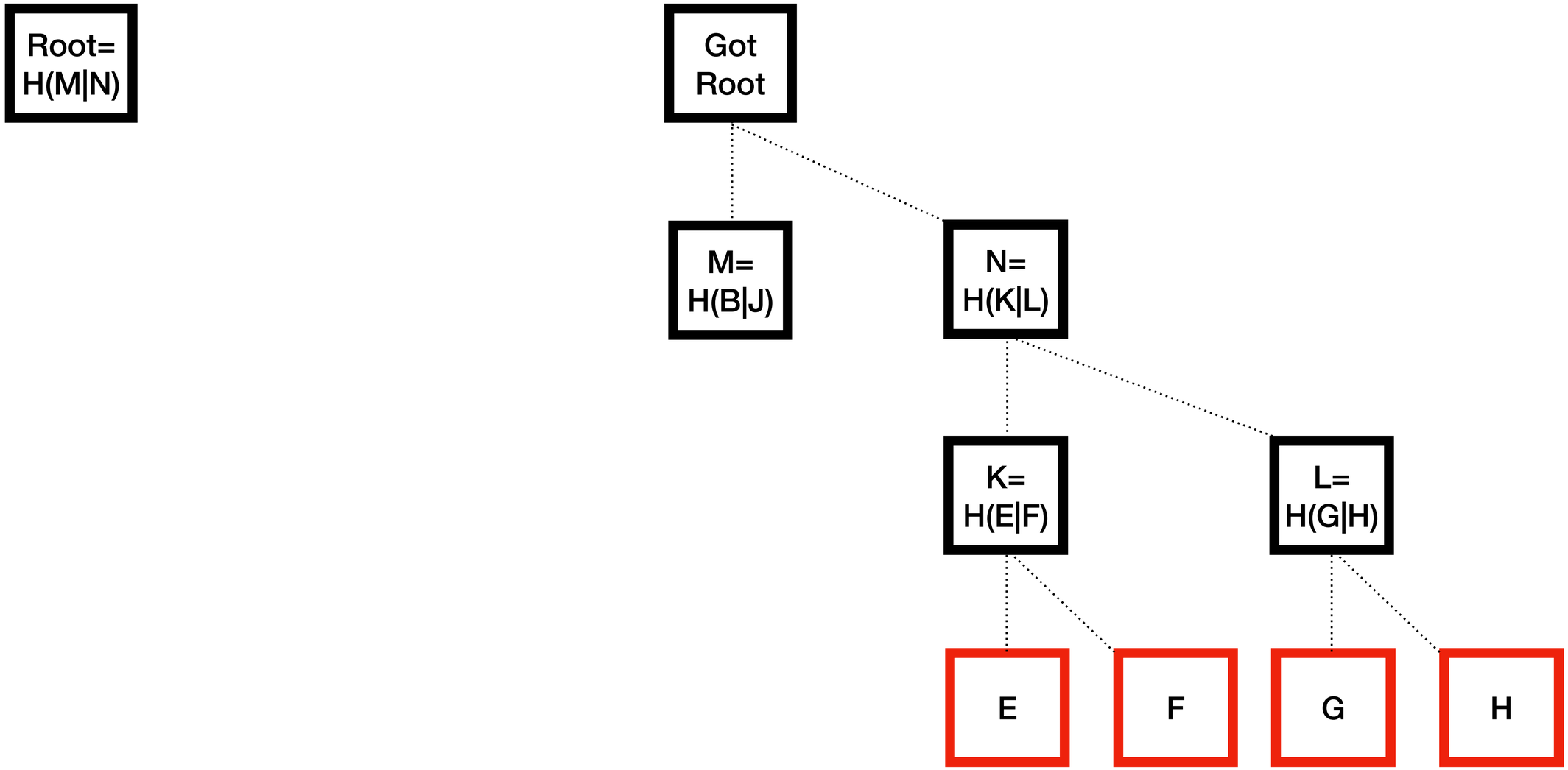
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate N



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

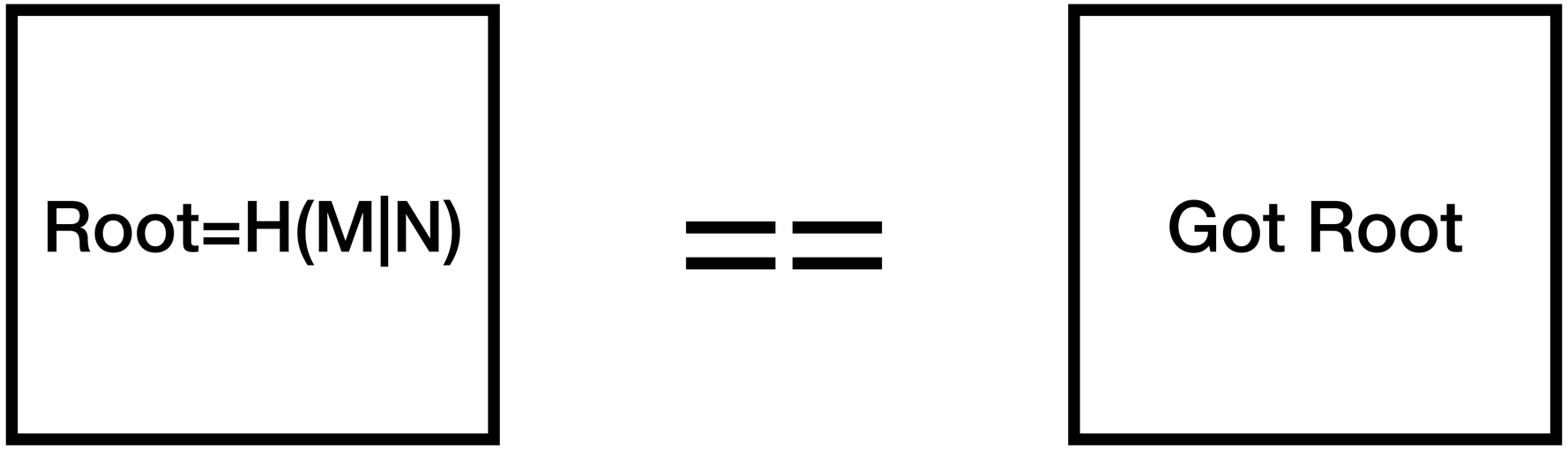
Calculate the Root



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

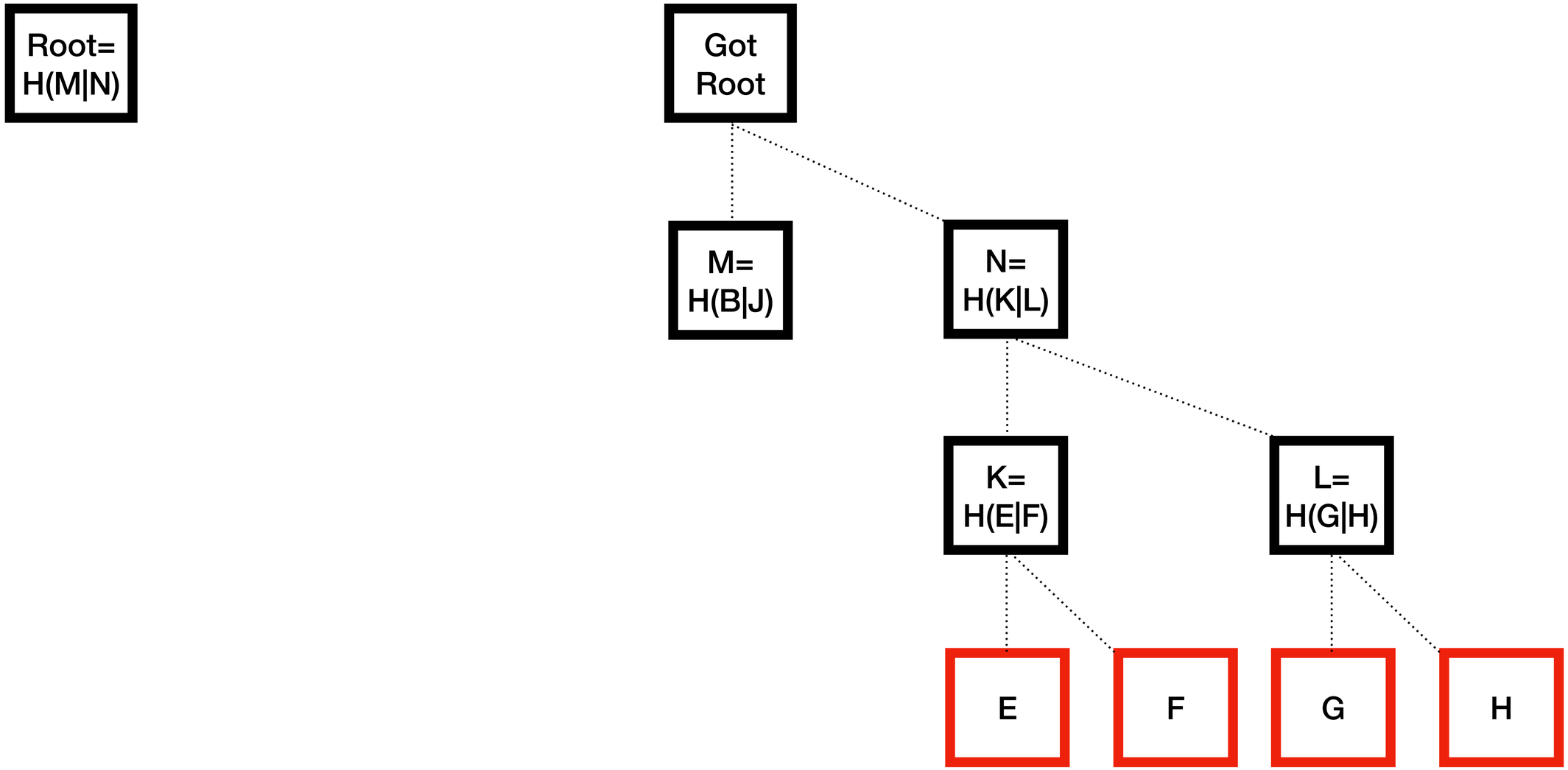
Compare roots

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



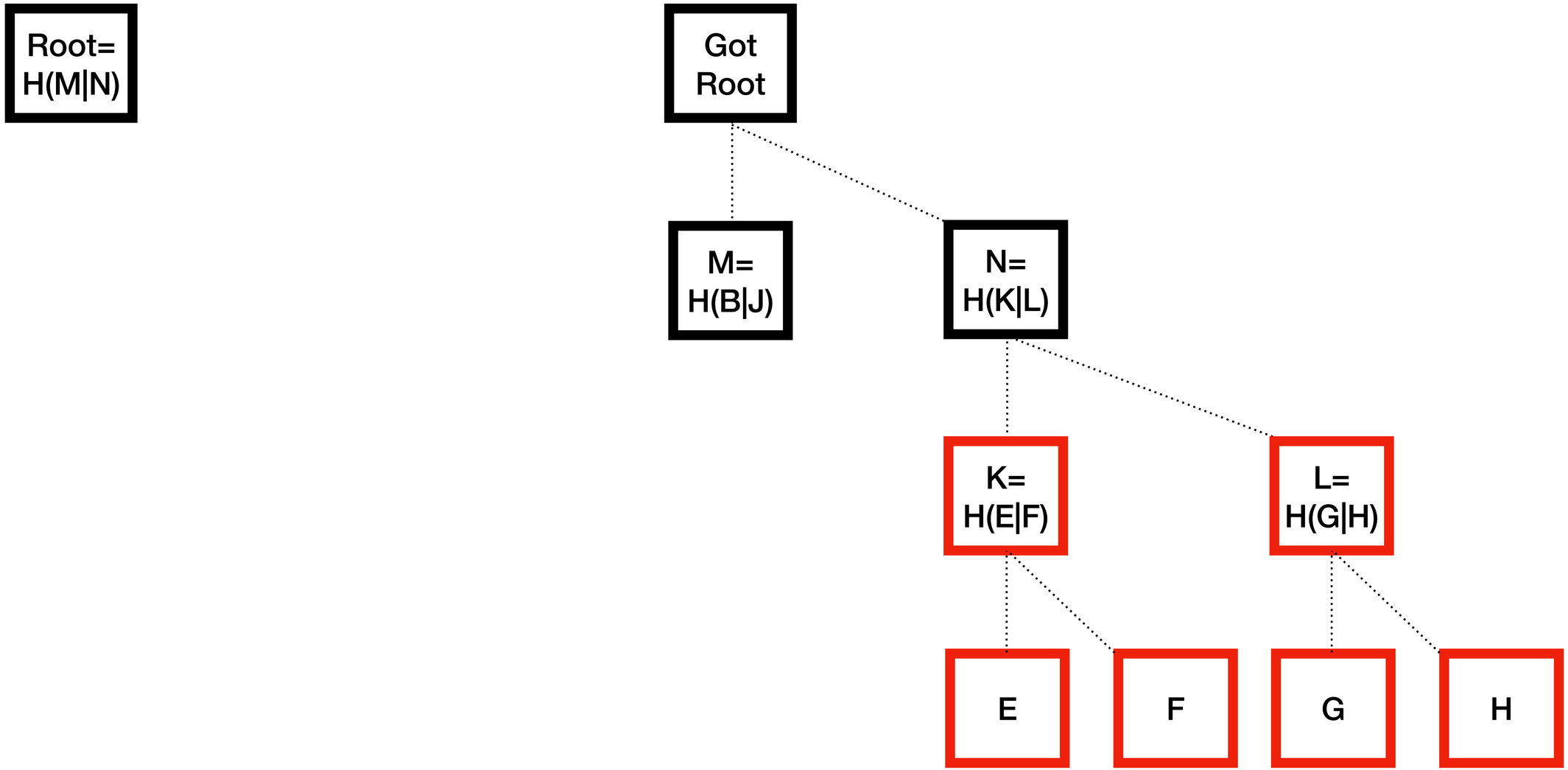
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Calculate new root



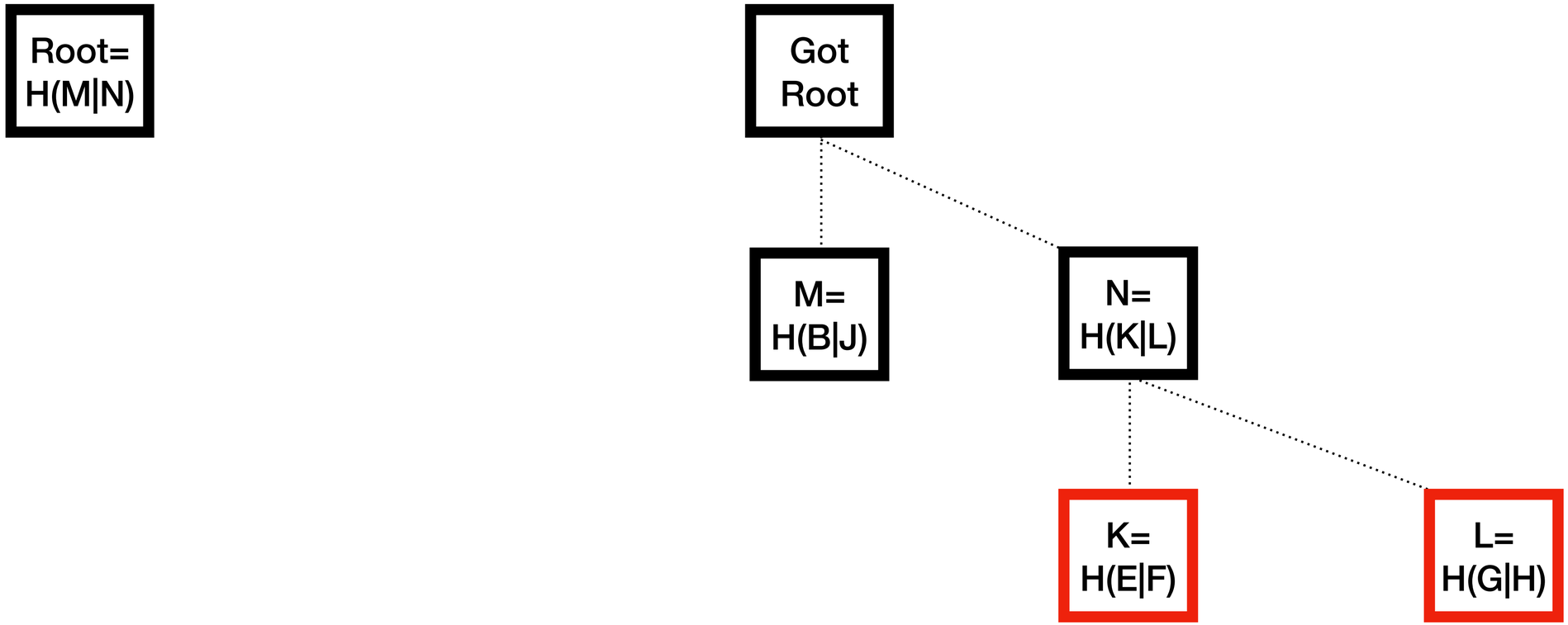
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Mark K & L as node to delete



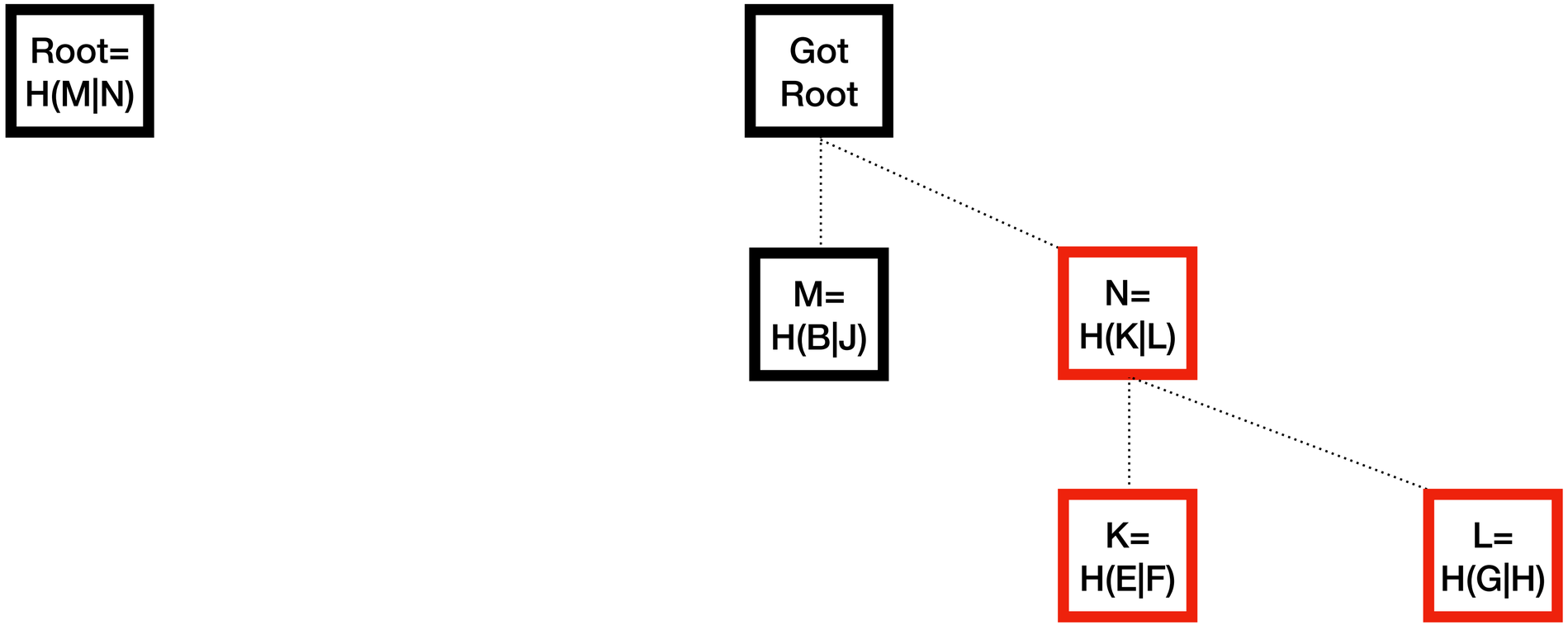
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Remove old nodes for E, F, G, H



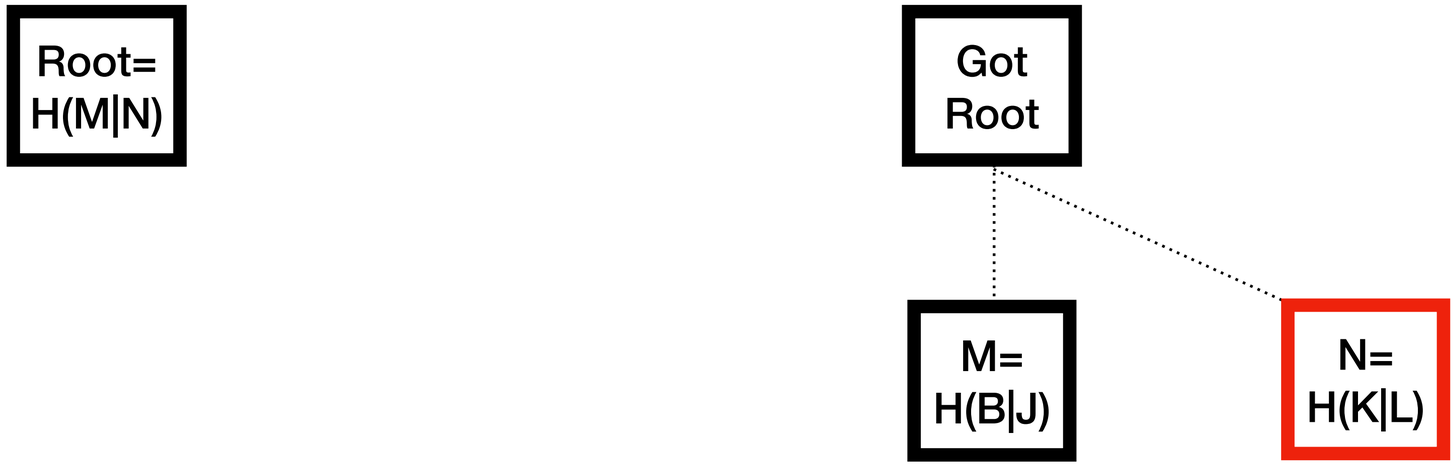
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Mark N as node to delete



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

Remove old nodes for K & L



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

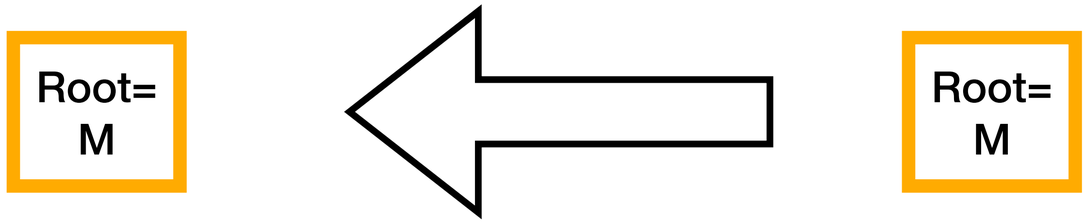
Move up M

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

Remove old nodes for M & N

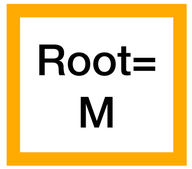
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

Copy over the new root to be saved



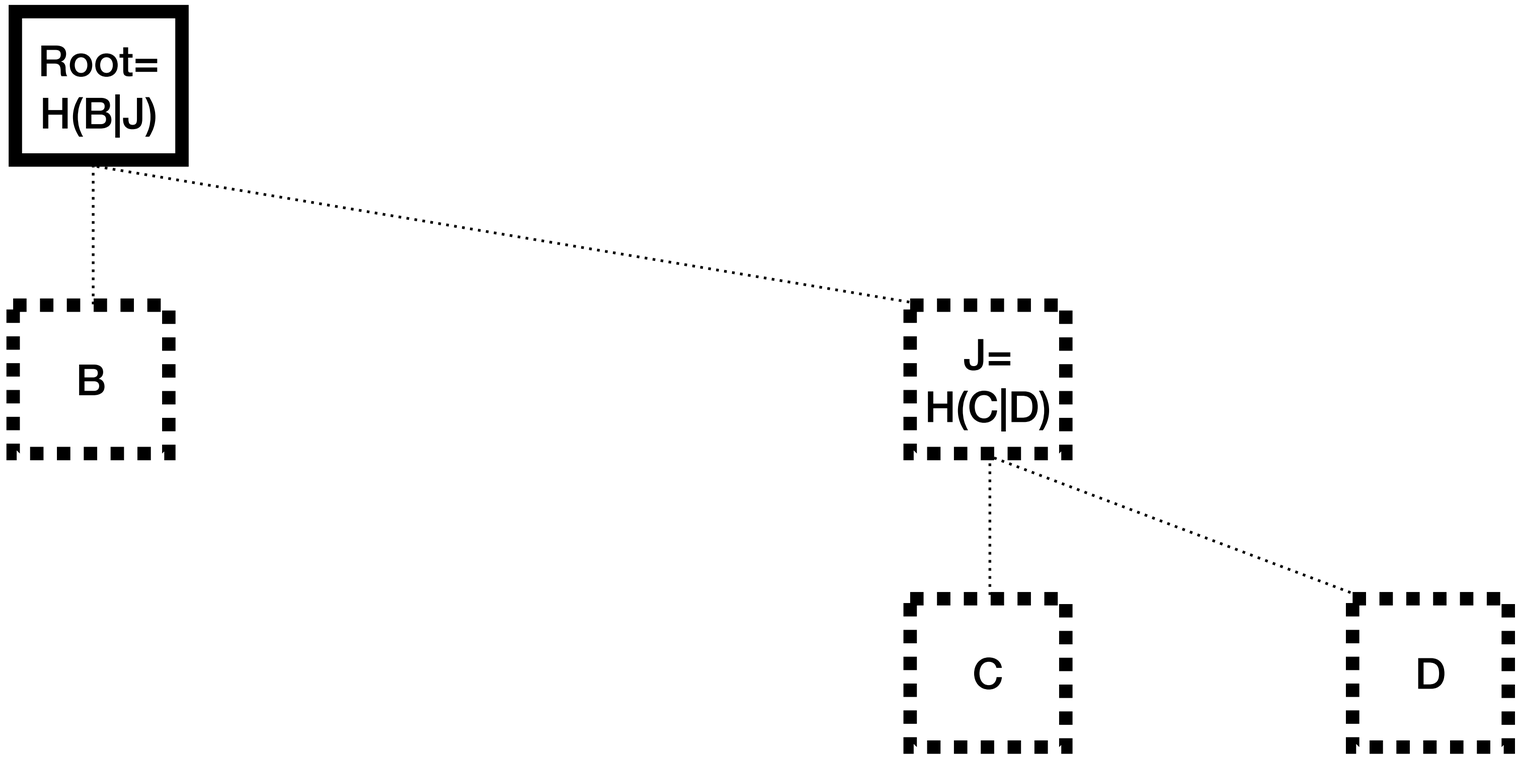
https://gist.github.com/kcalvinalvin/a790d524832e1b7f96a70c642315fffc

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

github.com/kcalvinalvin/slides-for-tabconf-2023