

# Utreexo - What is it?

Quick intro

# Agenda for today

## Show & 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 .
```

```
[1] 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
  0K    ./coldcardmk4
104K    ./coldcardmk3
8.2G    .
```

**UTXO set lives in chainstate/**

# The UTXO set

## Quick overview

- TXO - An Output of a TX (transaction)

# Typical TX

[8ad63210d059908229f00ad177469d9f16a5d37a7ea64cede9c92684ff7cf06e](#)

DETAILS



#0 [f758ad929fbefac99ef0d1f8b13990fa75efb517ef6c919178c8af7308f145](#) 0.04852489 BTC  
07:1

#0 [17dSwJytZBszAafyWrK8Pue7rsRh9s5xeJ](#) 0.04033963 BTC

#1 [bc1qwqdg6squ38e46795at95yu9atm8azzmyvckulcc7kytlcckxswvvzej](#) 0.00758526 BTC

1 CONFIRMATION 0.04792489 BTC

# Outputs of a TX

8ad63210d059908229f00ad177469d9f16a5d37a7ea64cede9c92684ff7cf06e

DETAILS +

#0 f758ad929fbefac99ef0d1f8b13990fa75efb517ef6c919178c8af7308f145 0.04852489 BTC  
07:1

#0 17dSwJytZBszAafyWrK8Pue7rsRh9s5xeJ 0.04033963 BTC

#1 bc1qwqdg6squ38e46795at95yu9atm8azzmyvckulcc7kytlcckxswvv 0.00758526 BTC  
zej

1 CONFIRMATION 0.04792489 BTC

# The UTXO set

## Quick overview

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

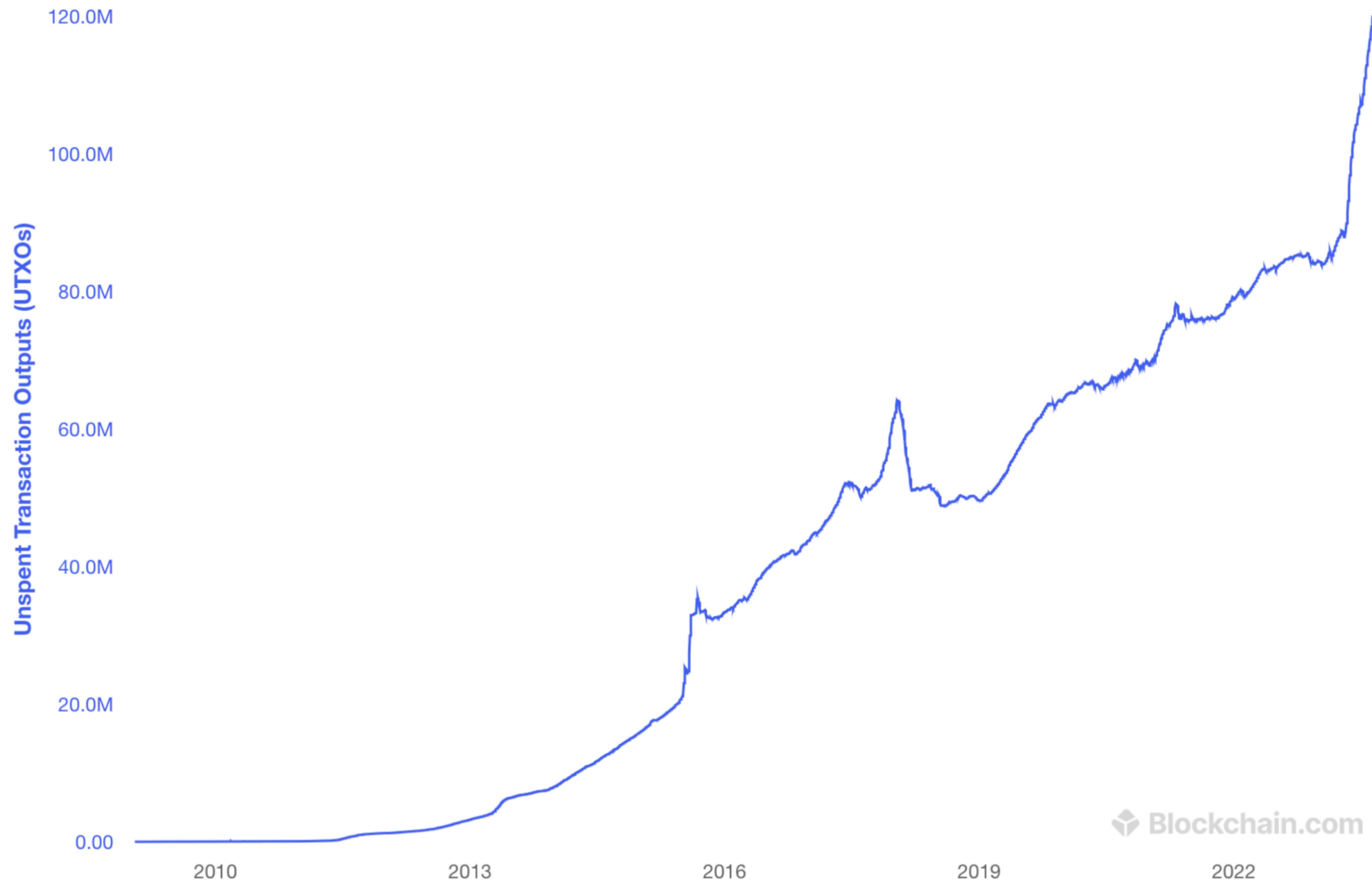
# The UTXO set

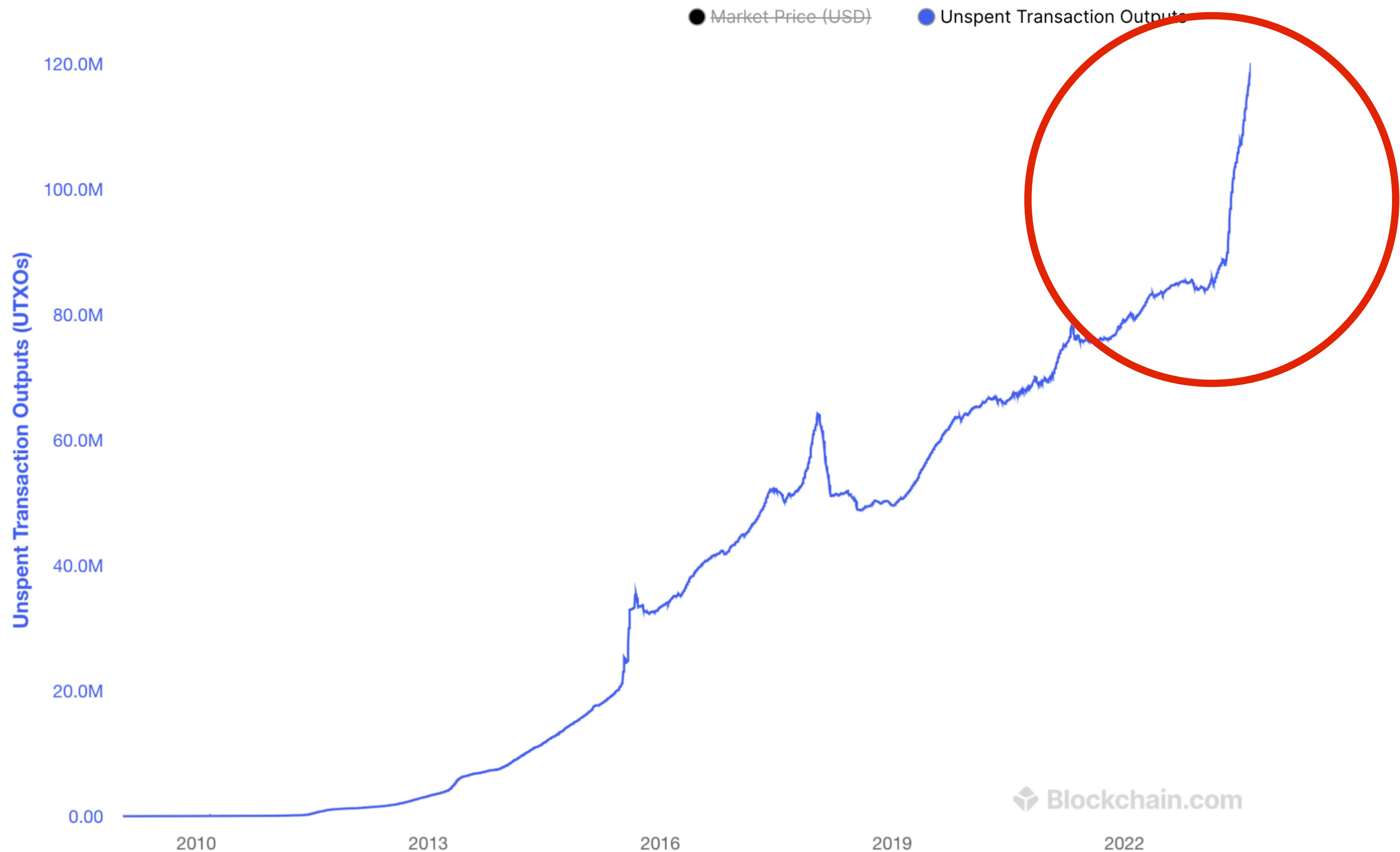
## Quick overview

- 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





**Turn the UTXO set into a  
merkle tree**

Turn the UTXO set into a  
*fancy merkle tree*



# Merkle trees



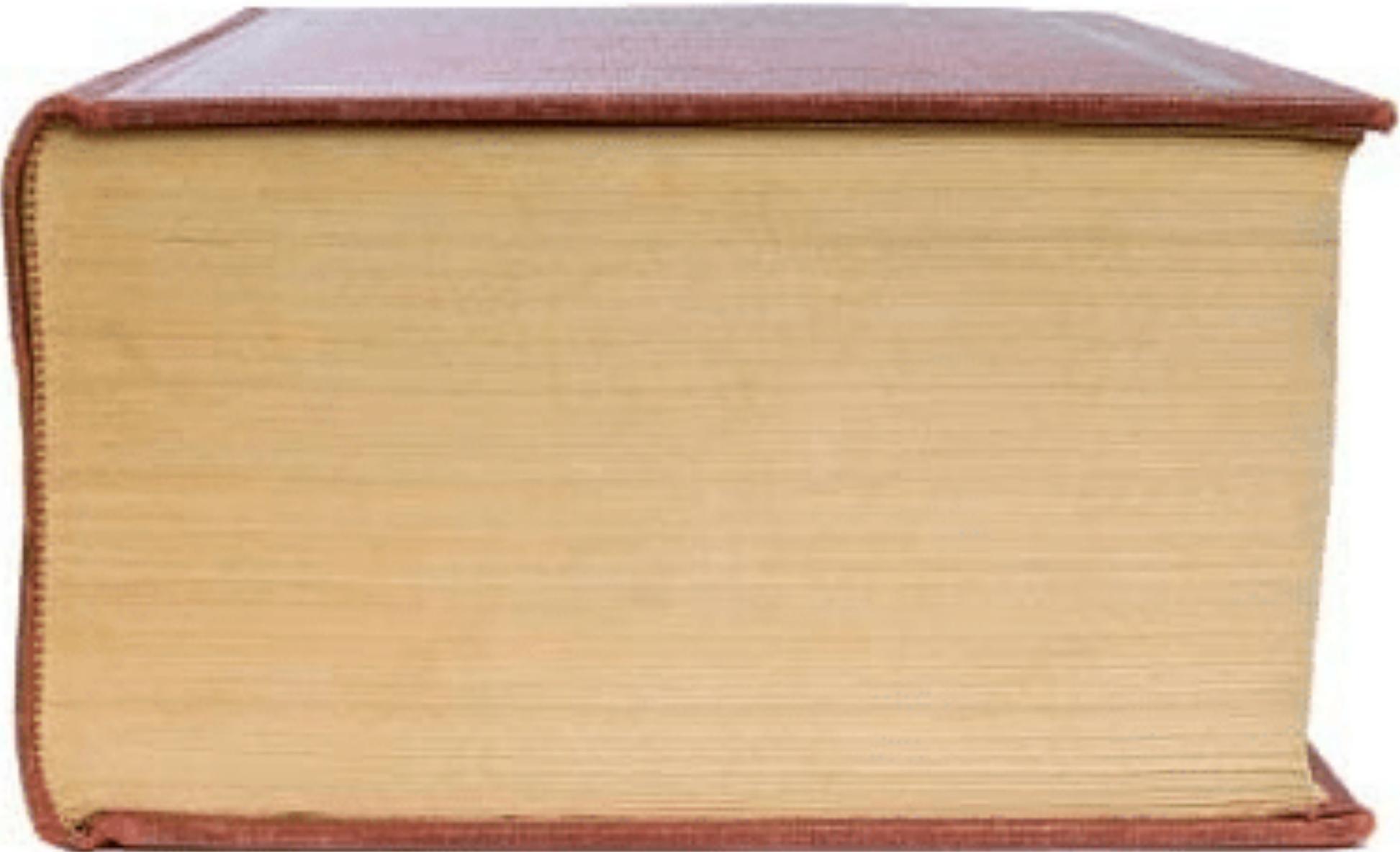
*Utreeexo*

# The UTXO set

Why merkelize it?

- Puts a bound to the UTXO set growth

# Pruned nodes



Without  
Utreeexo



With  
Utreeexo

# Pruned nodes



Without  
Utreeexo



With  
Utreeexo

# Pruned nodes



**O(N) -> O(log2 N)**

**N=8 billion UTXOs, at 59B per entry**

**472GB -> 416B**

# 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**

**3% slower**

**Utreeexo nodes**

# What we flush for utreexo

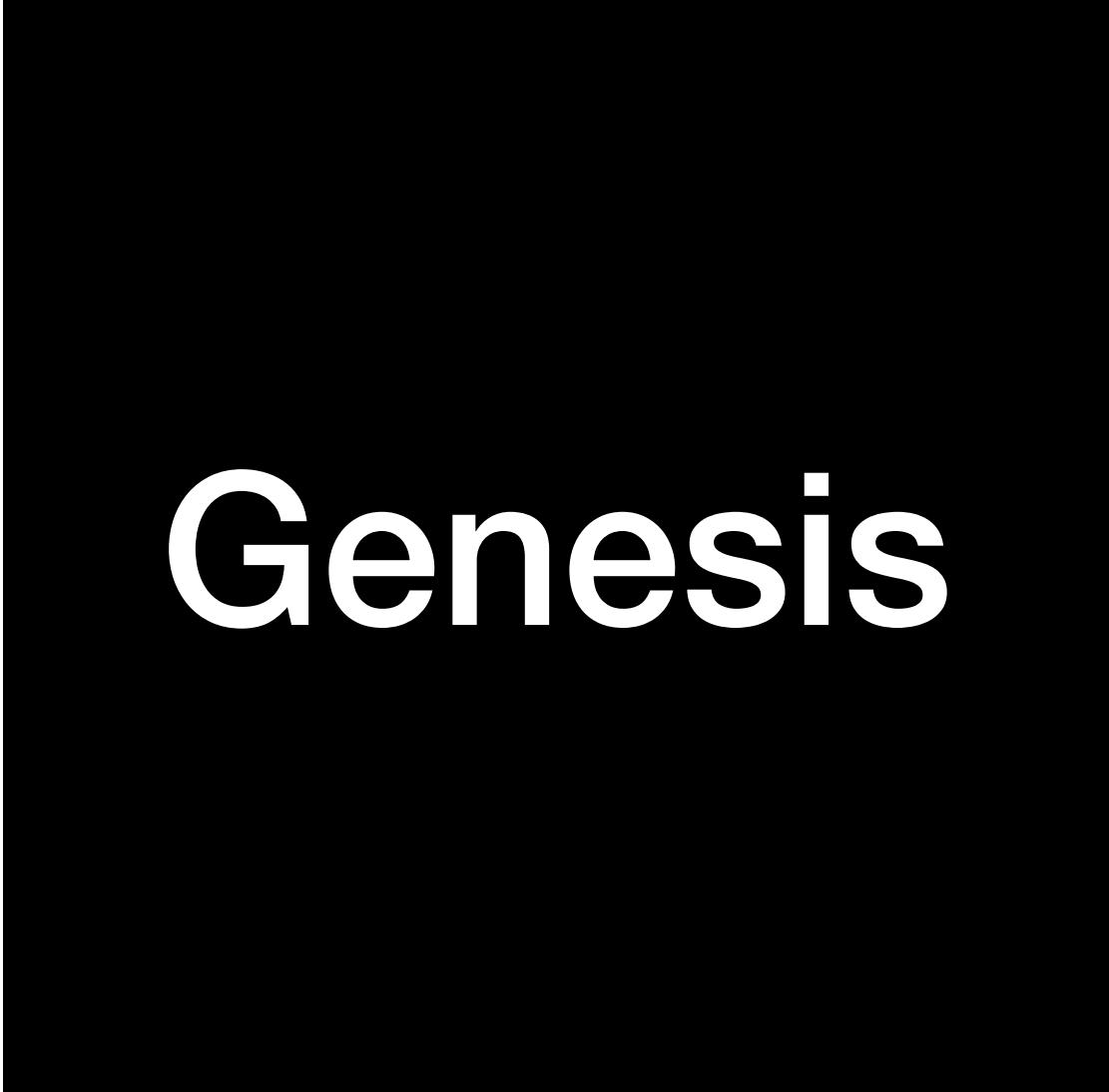
## No random i/o

```
{443000, newHashFromStr("000000000000000047efc3c126c3398cfdef1609682d1492cd2909e3c0a17f"),  
    43243051, []*chainhash.Hash{  
        newLeafHashFromStr("df31c496a54a8021c38c77809da15a8bd3968970315f5a56c07f0fd9763262ff"),  
        newLeafHashFromStr("755f4cfbf5727c3e14aa6dcc6e5ab150c61101cead06fade5a886ee8d1dd377"),  
        newLeafHashFromStr("c572a92c7cd9e14ce5d7c94473ef90bee21856654eed5535314b3c117006bc83"),  
        newLeafHashFromStr("9e85567347ef0ea38c1797d8ed8e399ed0d99a07025388f6e06791ff6b88643c"),  
        newLeafHashFromStr("ab4efc738bd7b8281689e899979d44ed0ca8fe1489efa0f5a09f5b1c276fe792"),  
        newLeafHashFromStr("1c763d8f490baef0b756597dfc2da5a31d0aee324da6314b52f21707351cf51a"),  
        newLeafHashFromStr("29fc9989939a17e5d113fb1f247f3bd05606488bcb48c9cfad55a57006f2248"),  
        newLeafHashFromStr("e79f840fe1bcd6b638a53b637885dda7ea3da2b2b5f1f5447ebf4d54fc0a2e"),  
        newLeafHashFromStr("02f33c3ce0d684e644410fa35fee4c7572454facc710389ea0f69a00e27c1780"),  
        newLeafHashFromStr("4e29ad940088ab42991bd99d7943a68ea38d498f8435c4bea21b8252788b5a7d"),  
        newLeafHashFromStr("feabfc6565e918827df8ade12d0885e7da28bb81410c71a0fc92da28e5538e"),  
        newLeafHashFromStr("42275481edcd150c0891f8f7486941e7ecc9dbec0d1554d1ae3027d8523571b4"),  
        newLeafHashFromStr("84dd84bbb564ab9e878ea774d66a2cb818c2b74680974e01a07dde68bd064bfd"),  
        newLeafHashFromStr("19a64f2f77403d4a1e3482b3a1801aa360120f6429f5ed47261fc8676920d9d"),  
    },  
},
```

# **Parallel Block validation**

# **Current block validation**

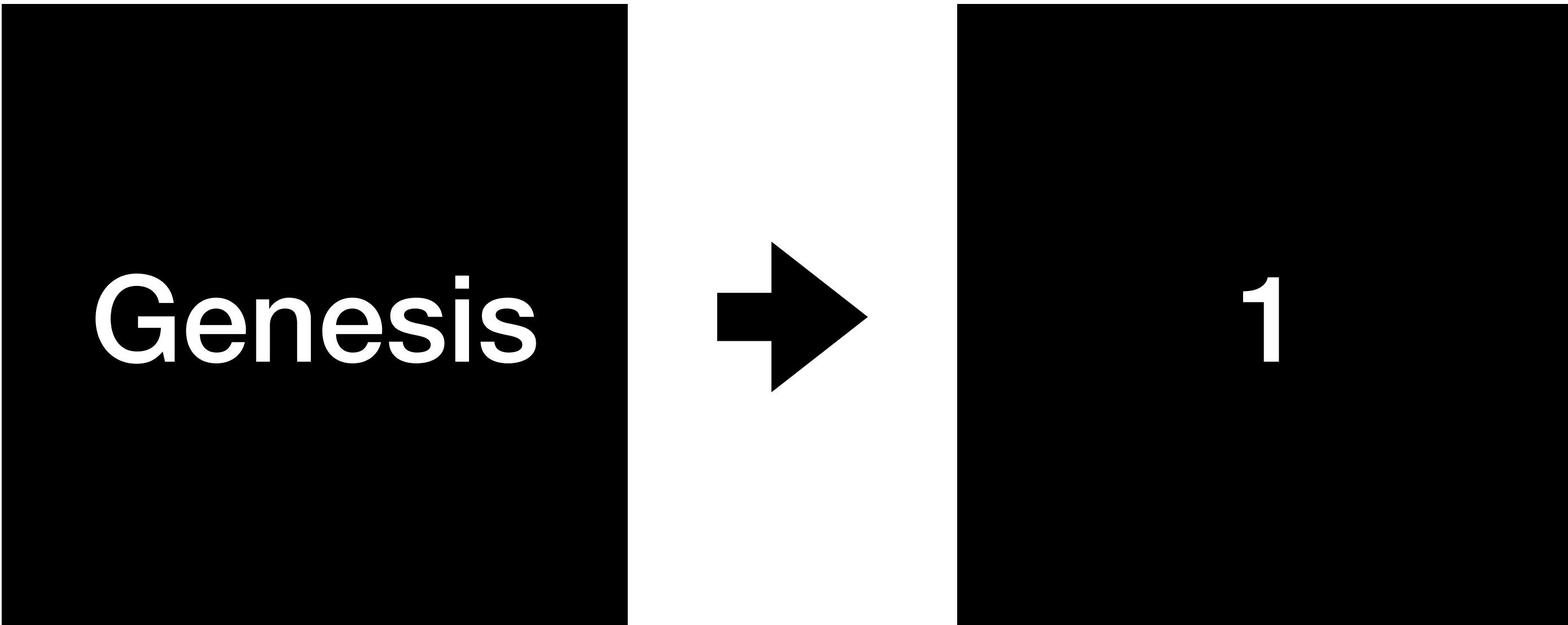
## **Sequential validation**



**Genesis**

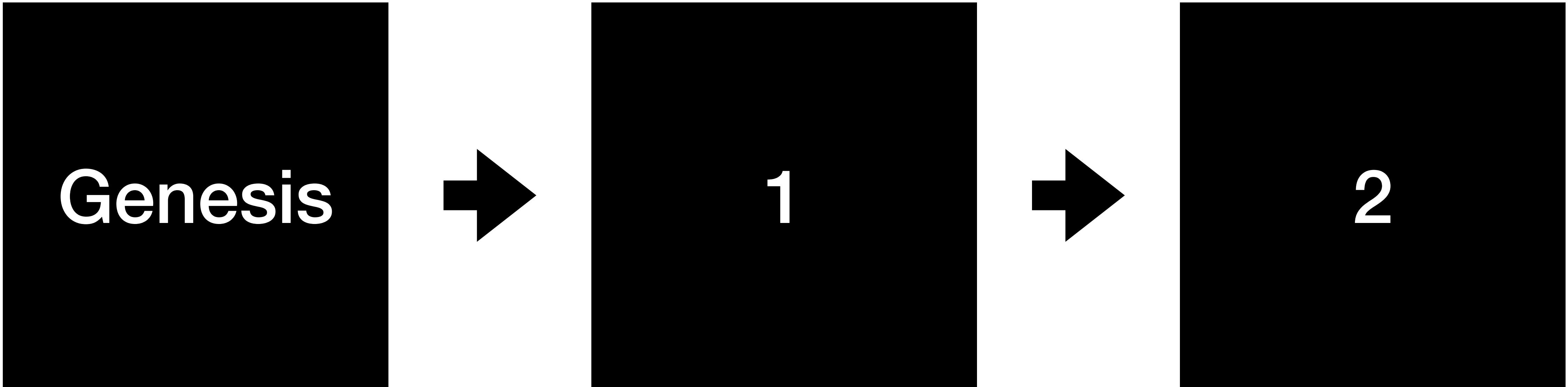
# **Current block validation**

## **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

# With Utreexo

## Efficient parallel validation

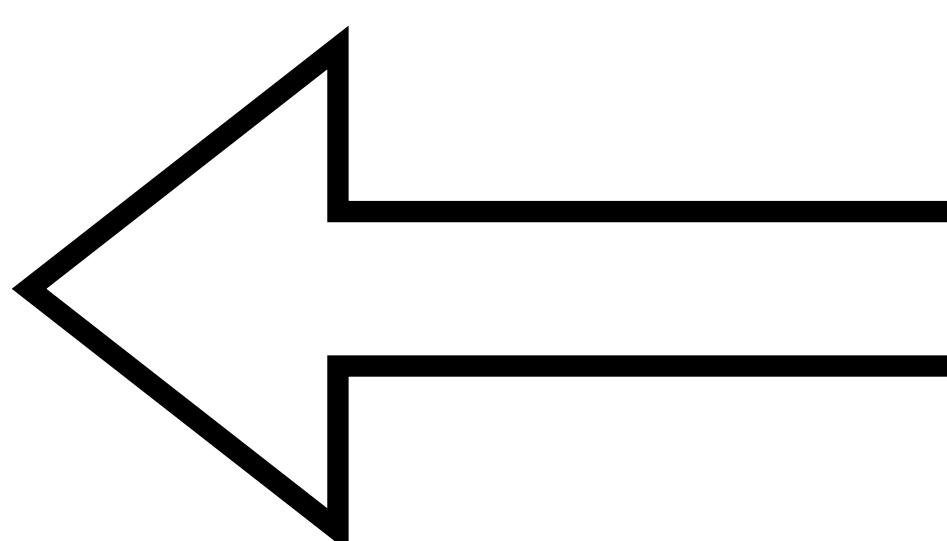
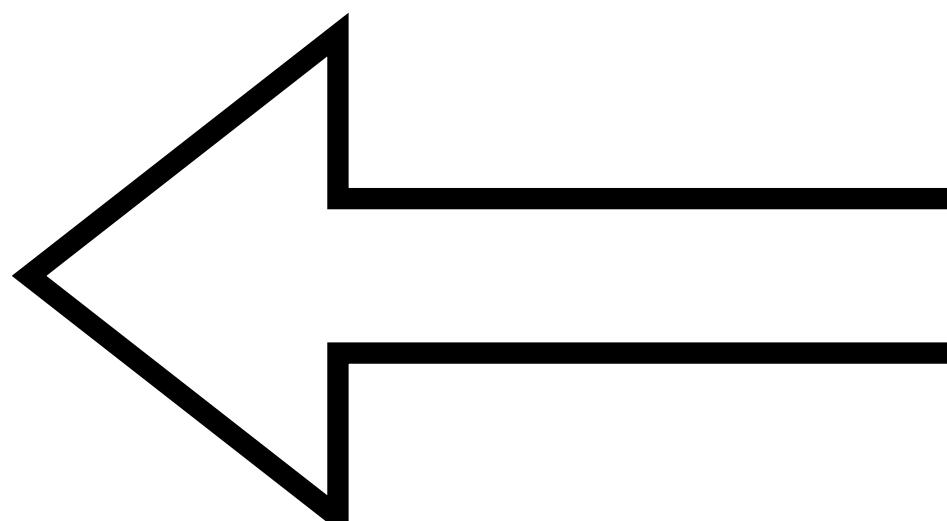
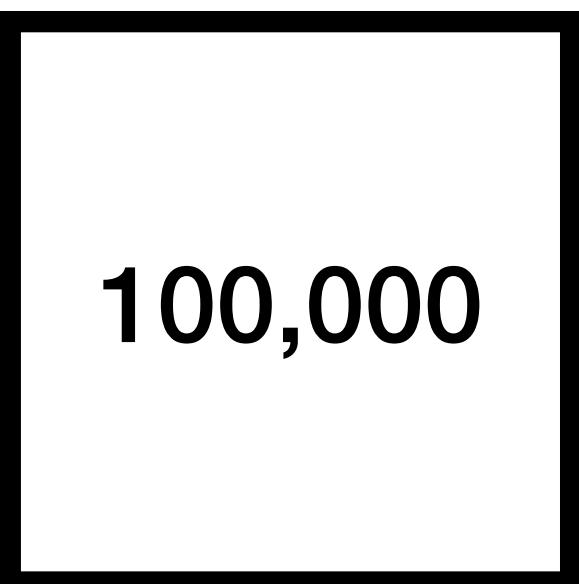
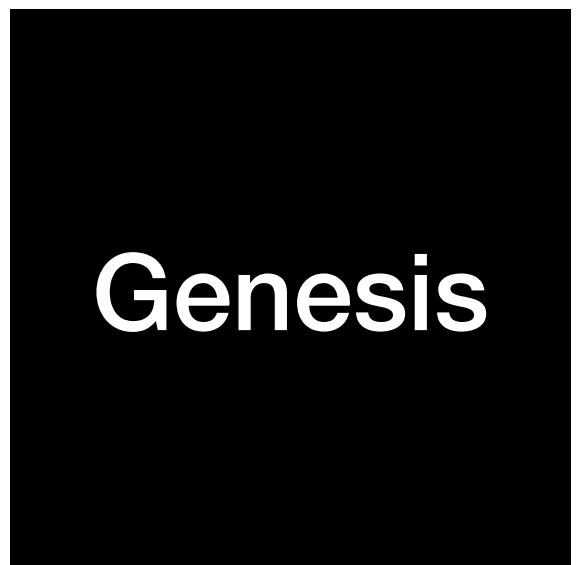
Genesis

100,000

200,000

# With Utreexo

Efficient parallel validation



Untrusted roots at  
the start

# With Utreexo

## Efficient parallel validation

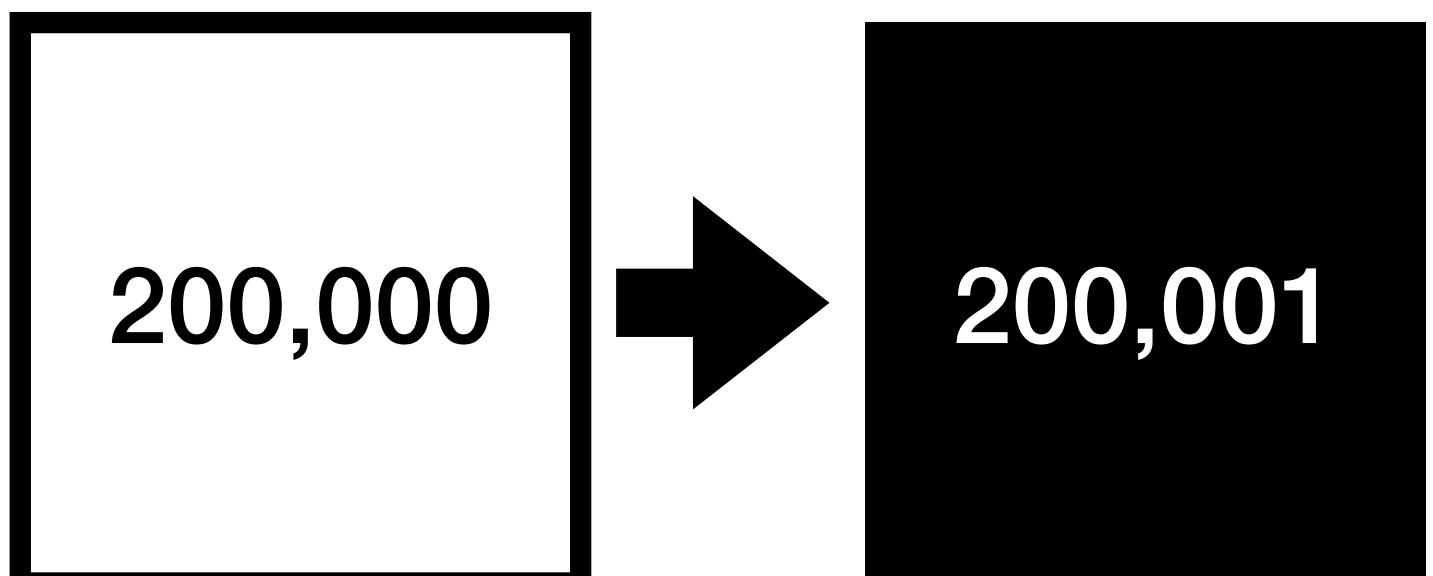
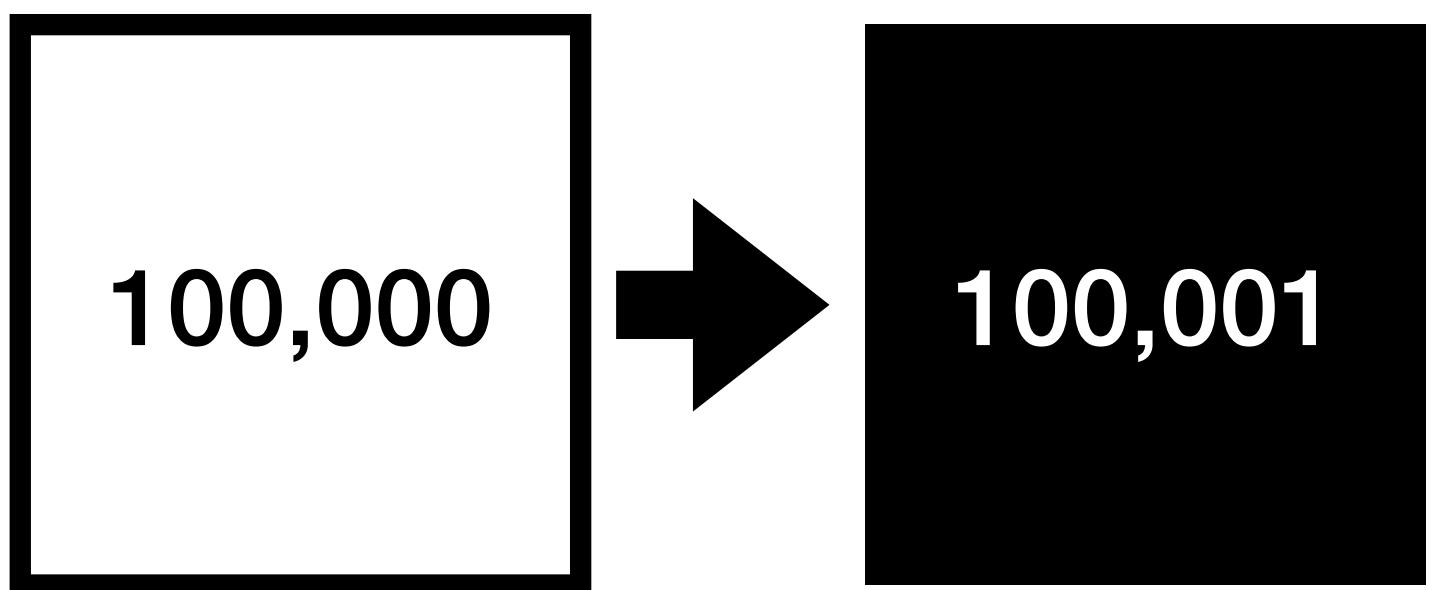
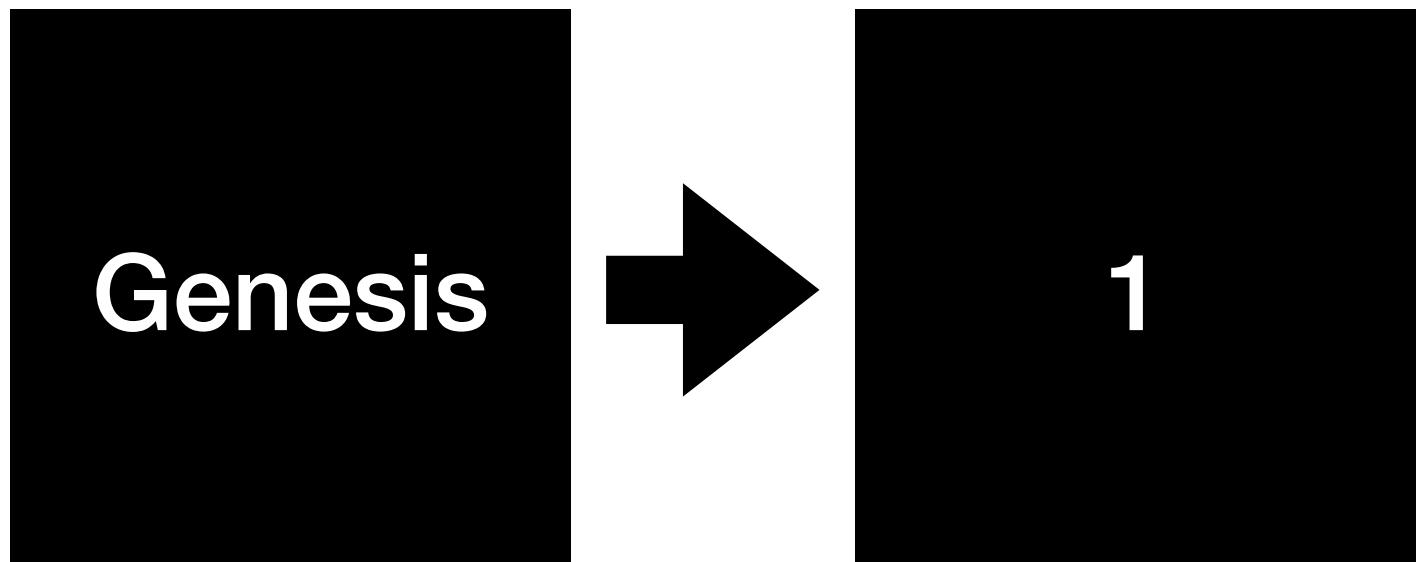
Genesis

100,000

200,000

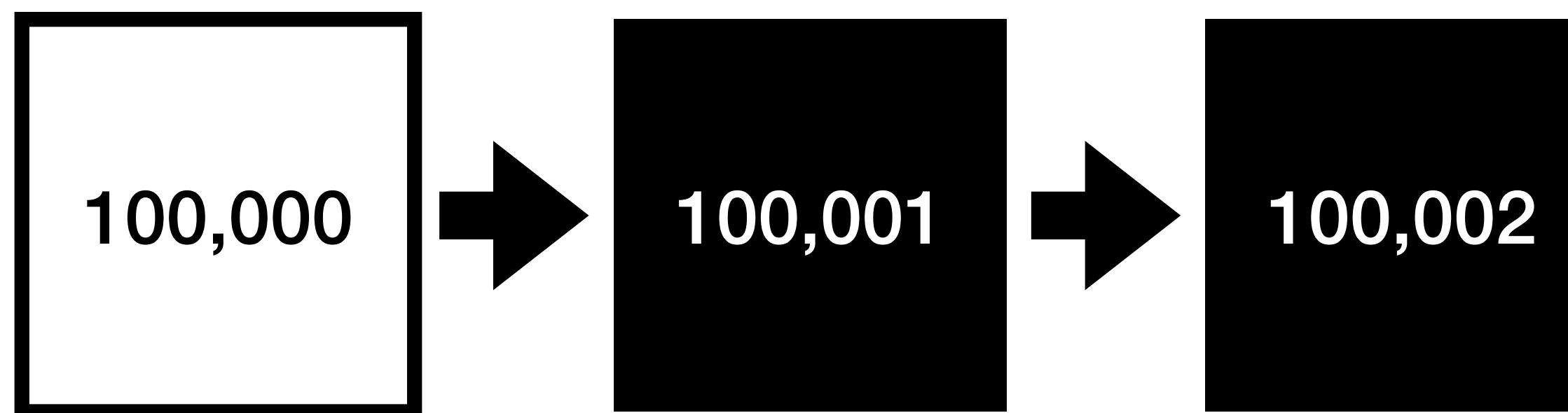
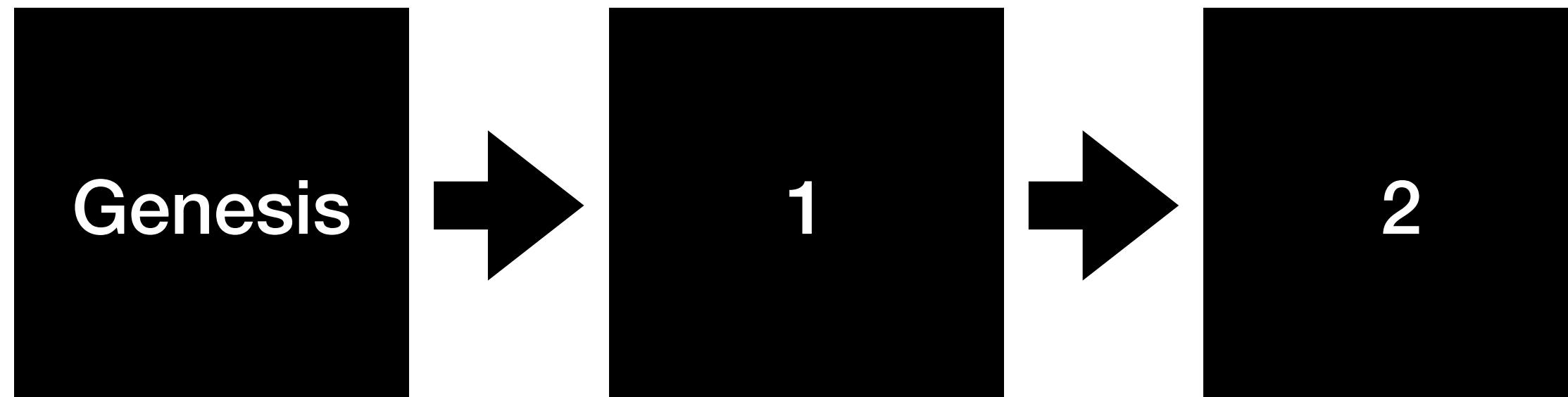
# With Utreexo

## Efficient parallel validation



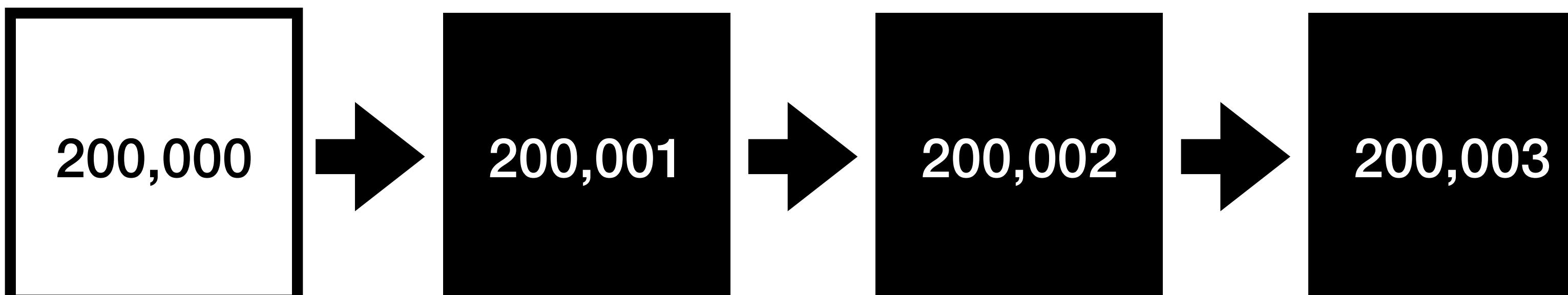
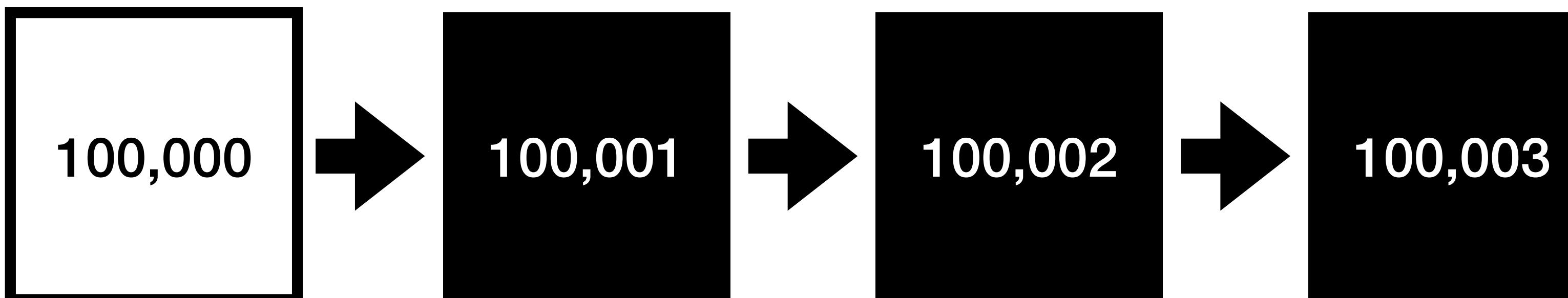
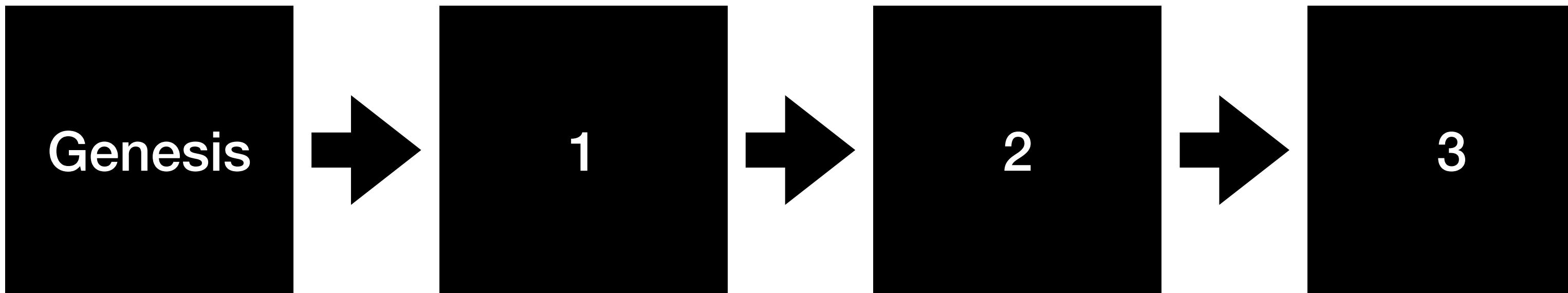
# With Utreexo

## Efficient parallel validation



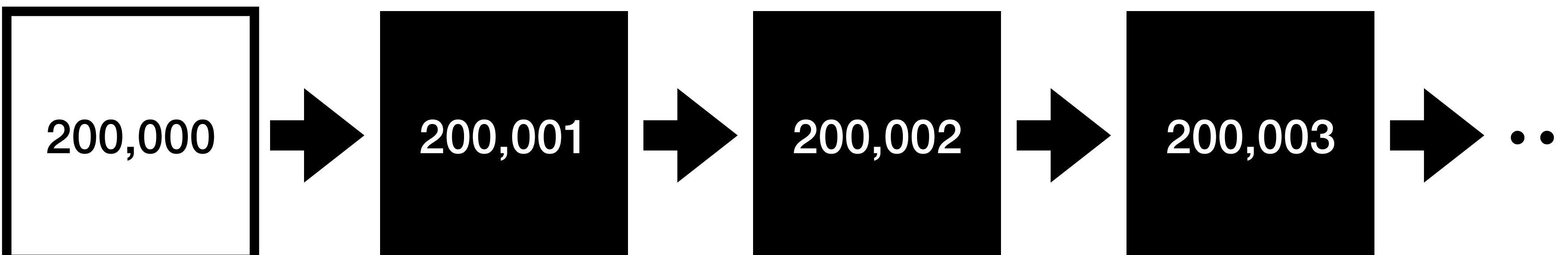
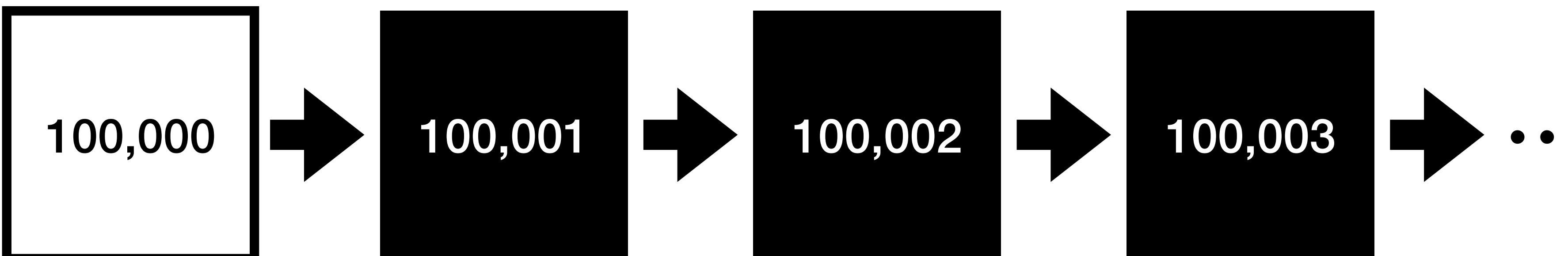
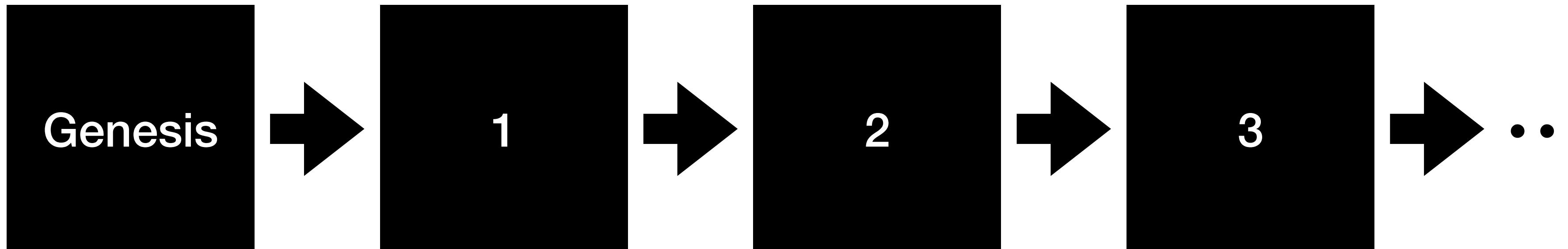
# With Utreexo

## Efficient parallel validation



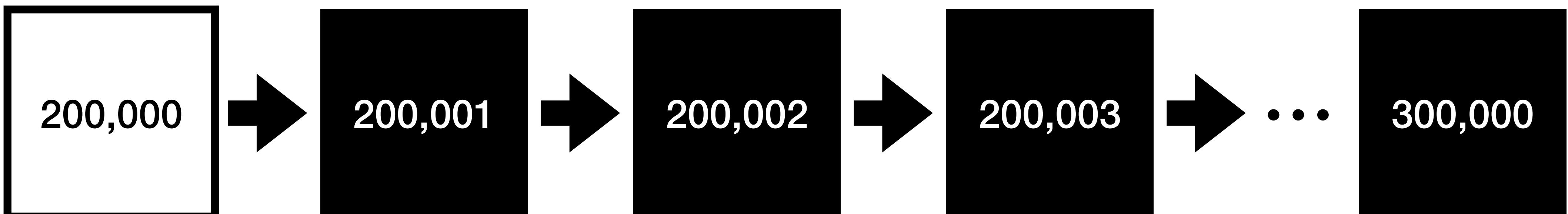
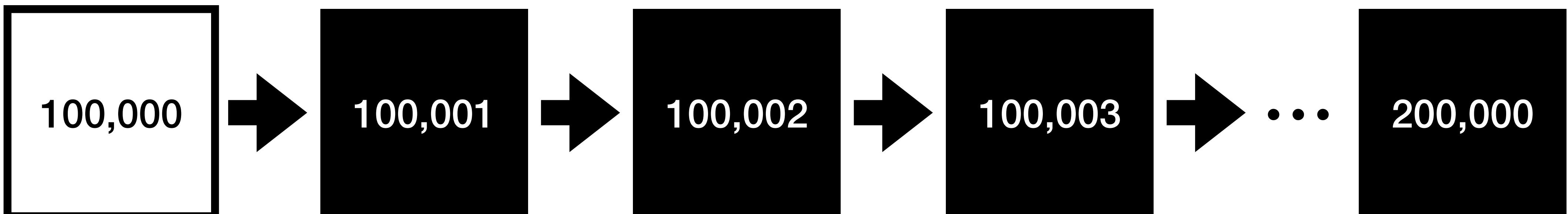
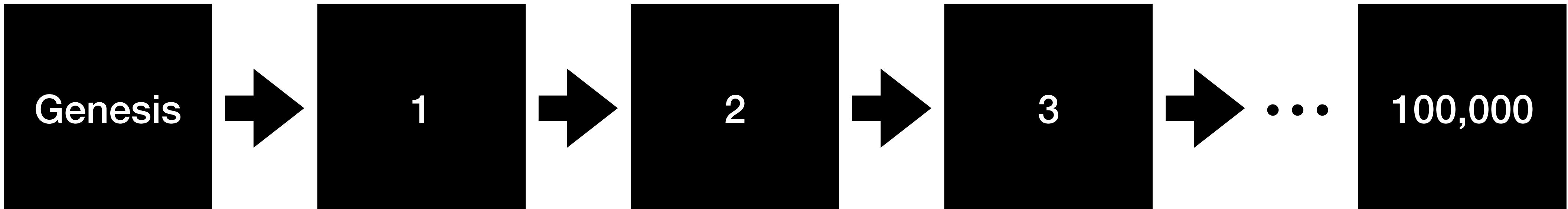
# With Utreexo

## Efficient parallel validation



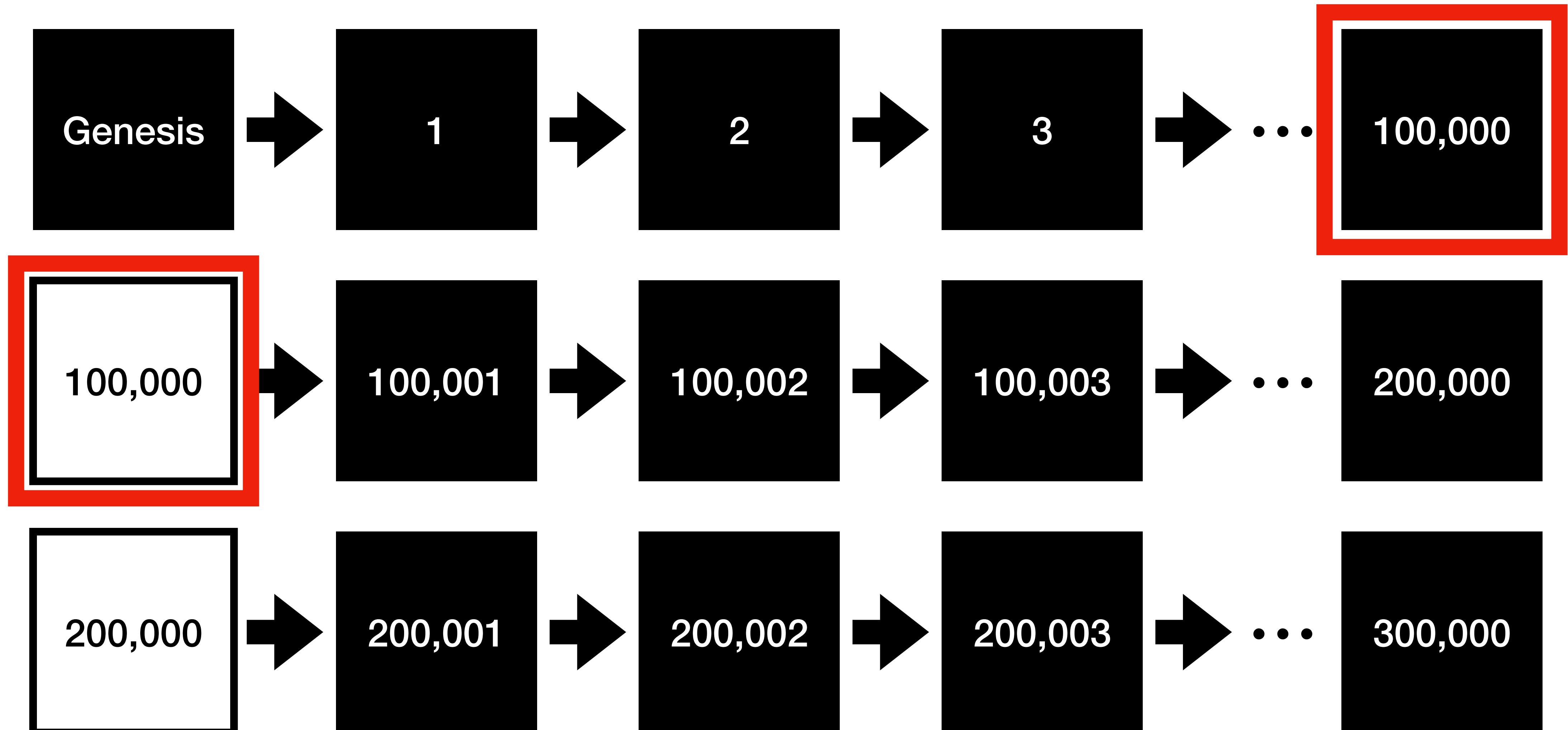
# With Utreexo

## Efficient parallel validation



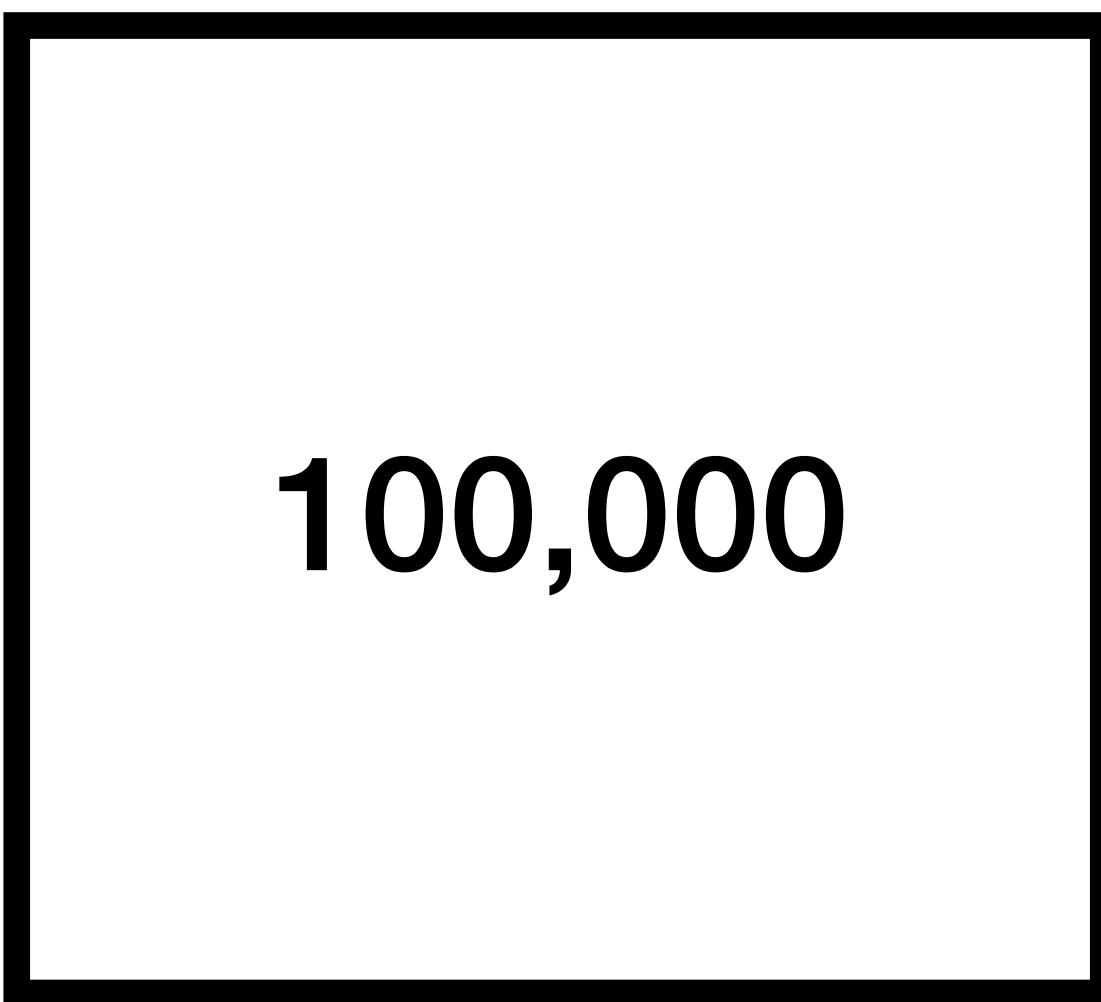
# With Utreexo

## Efficient parallel validation

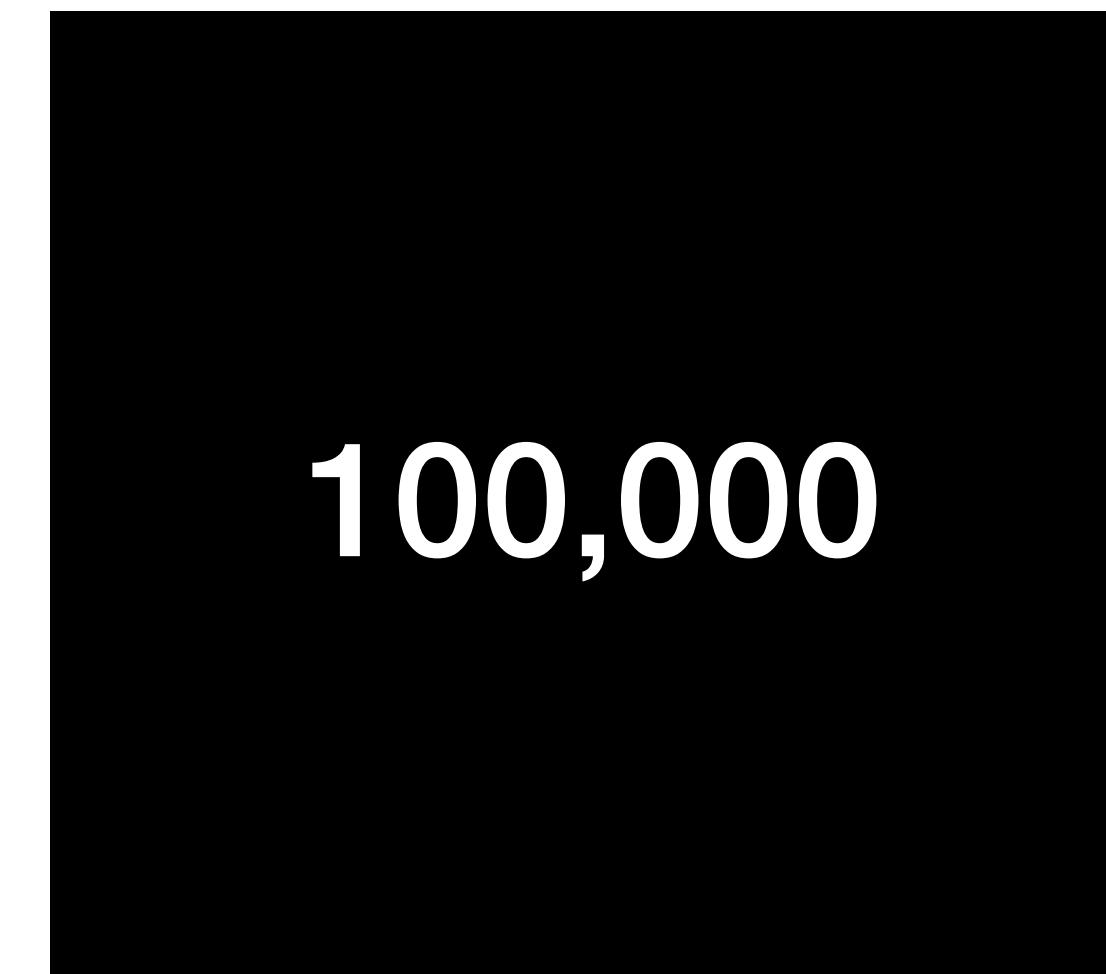


# Compare roots

Root at 100,000 becomes trusted if equal

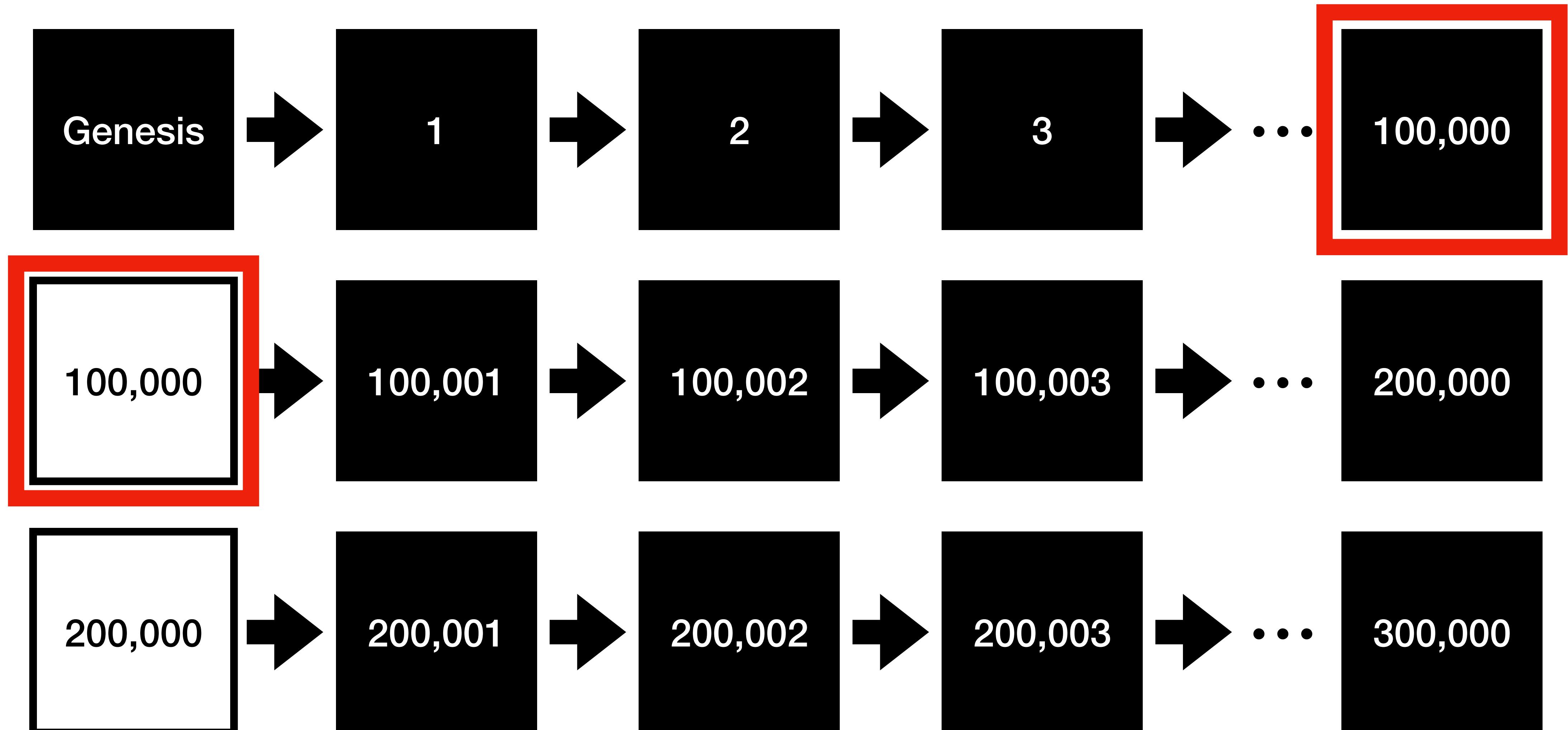


==



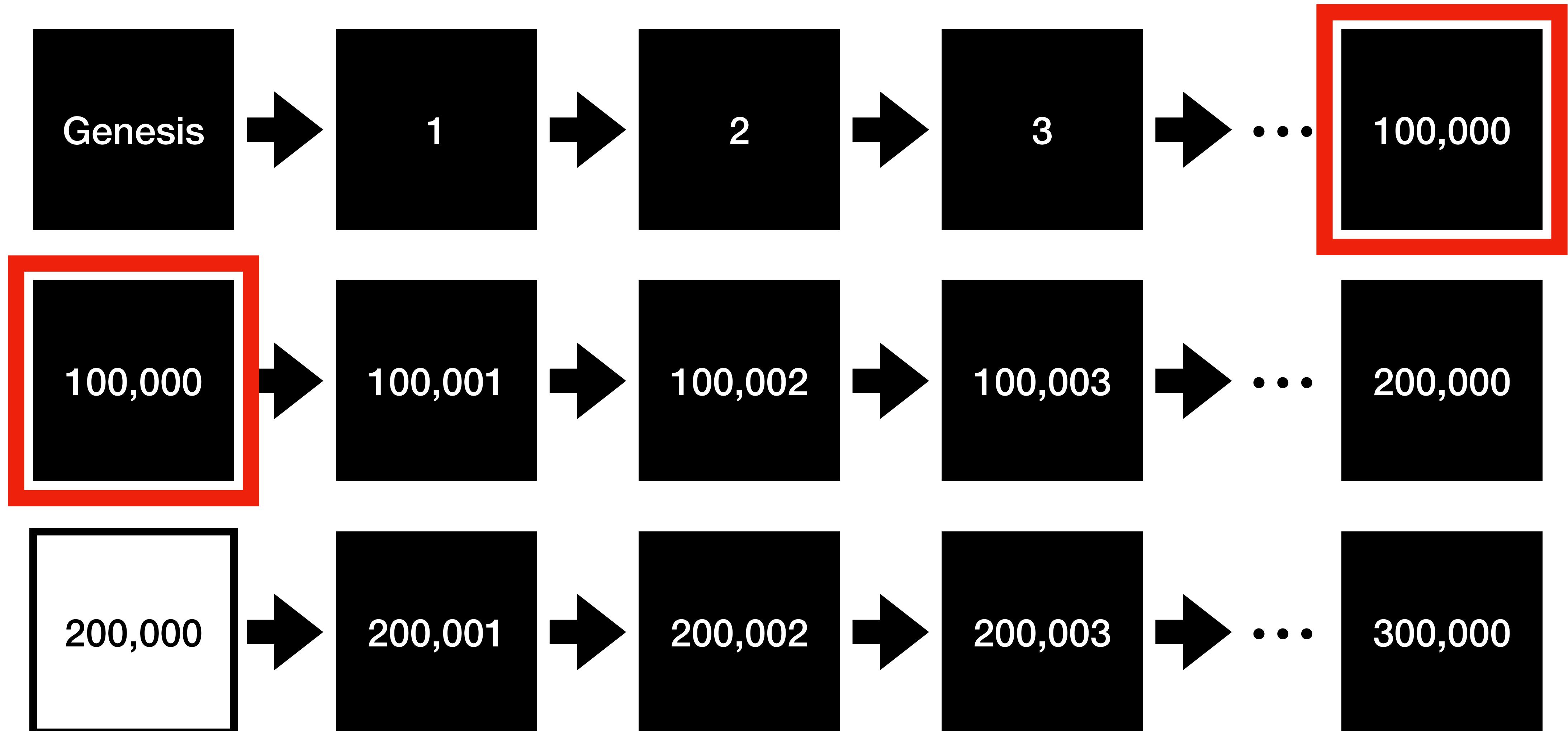
# With Utreexo

## Efficient parallel validation



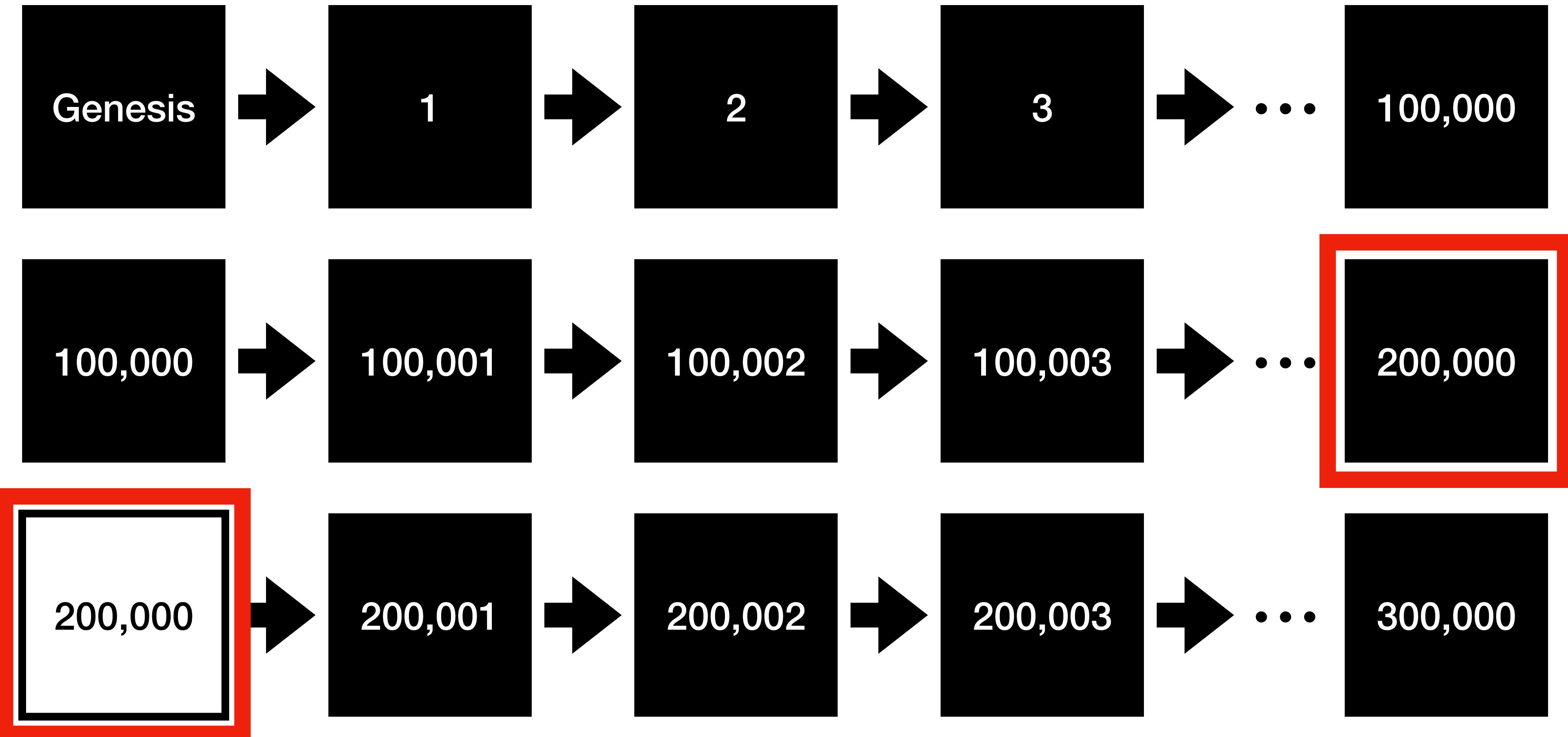
# With Utreexo

## Efficient parallel validation



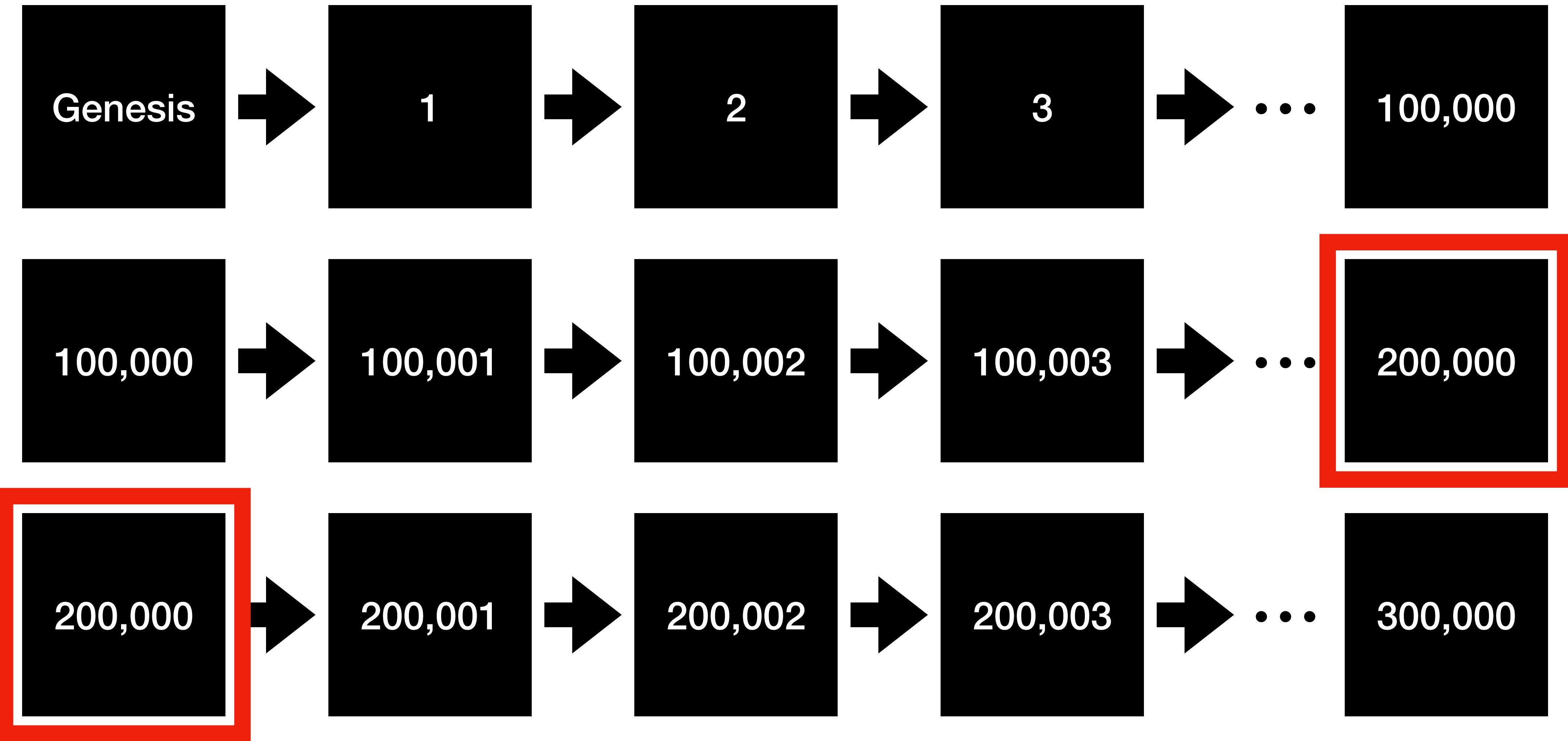
# With Utreexo

## Efficient parallel validation



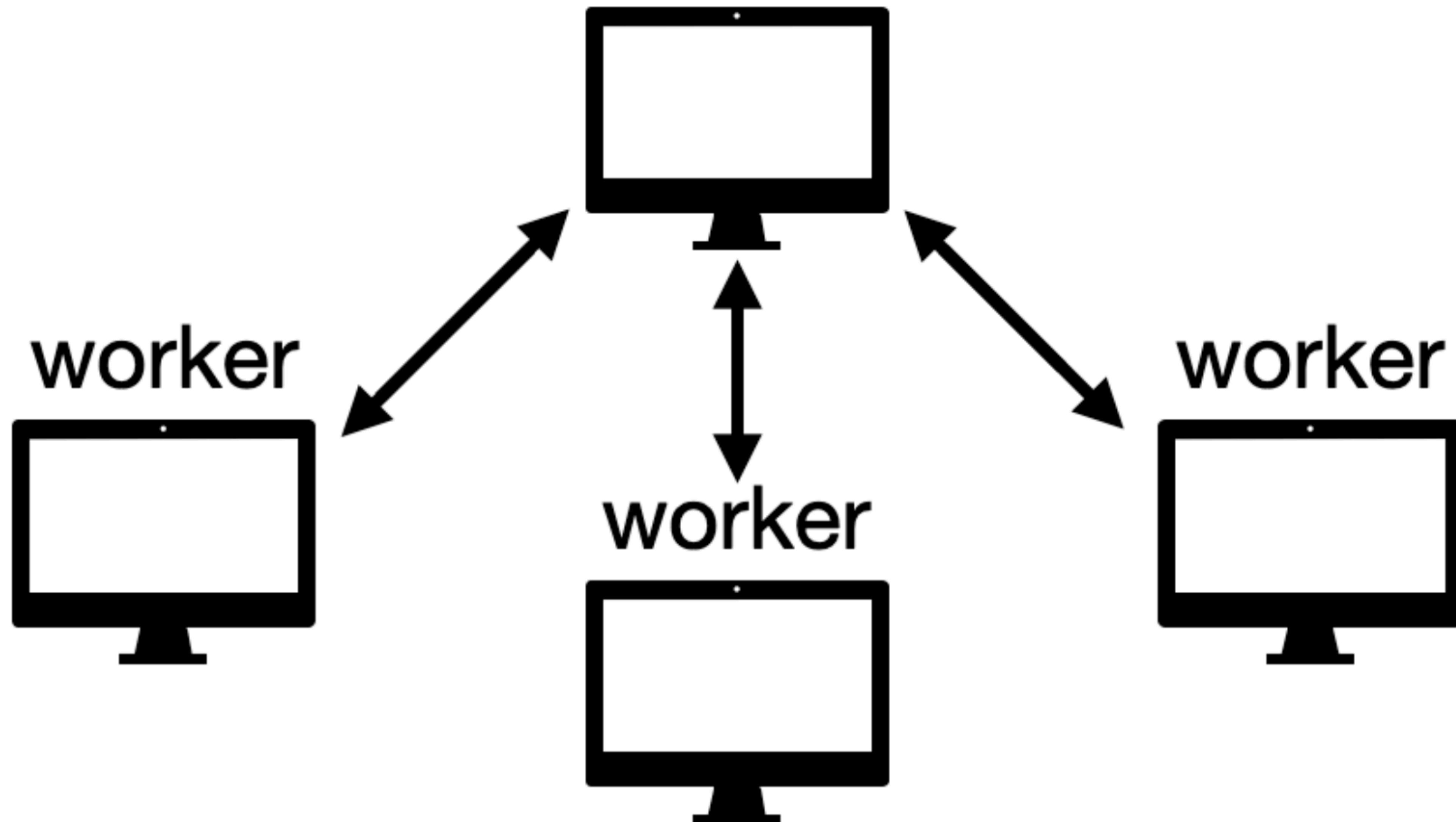
# With Utreexo

## Efficient parallel validation



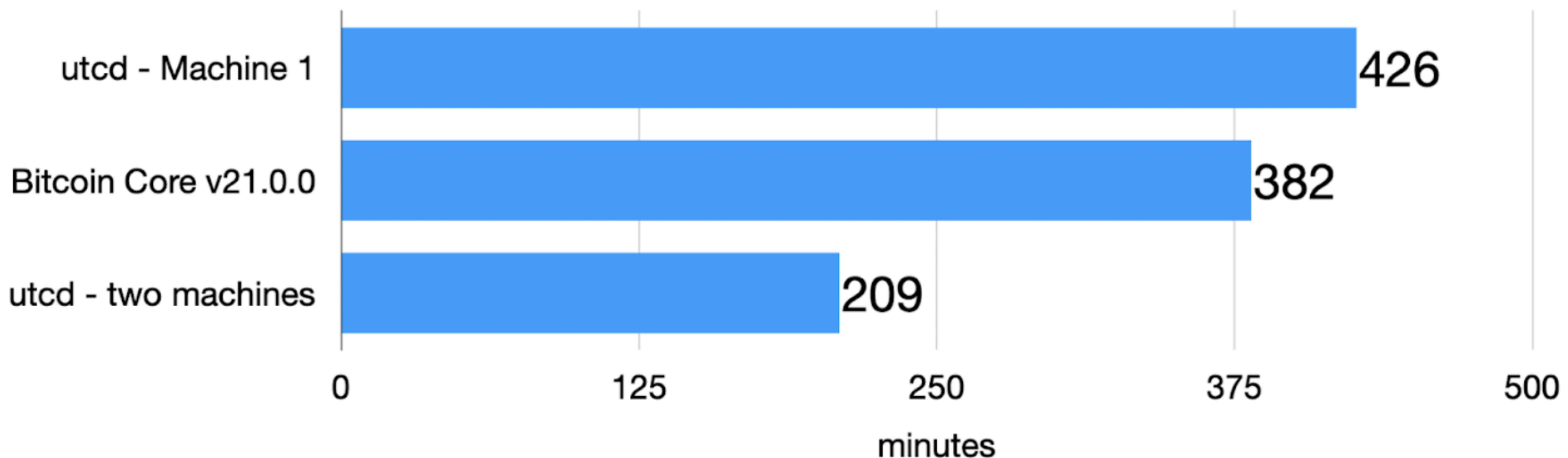
**~62% faster  
IBD**

# Coordinator/worker

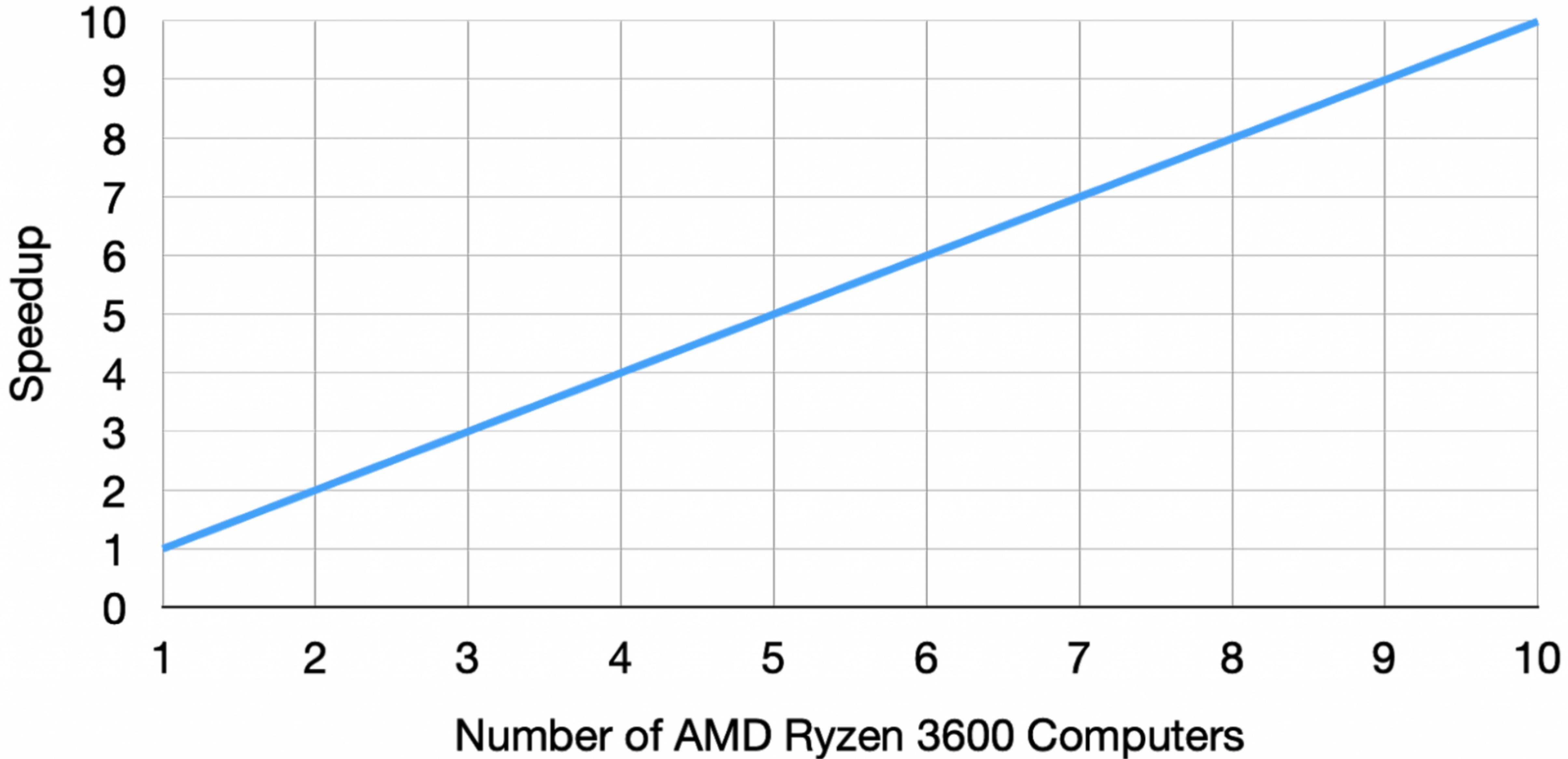


**2x faster IBD**  
**When using 2 computers for IBD**

## Initial Block Download Speeds to block 671,000 (Full signature check)



# Utreexo Parallel Node Speedups (based on Amdahl's Law)



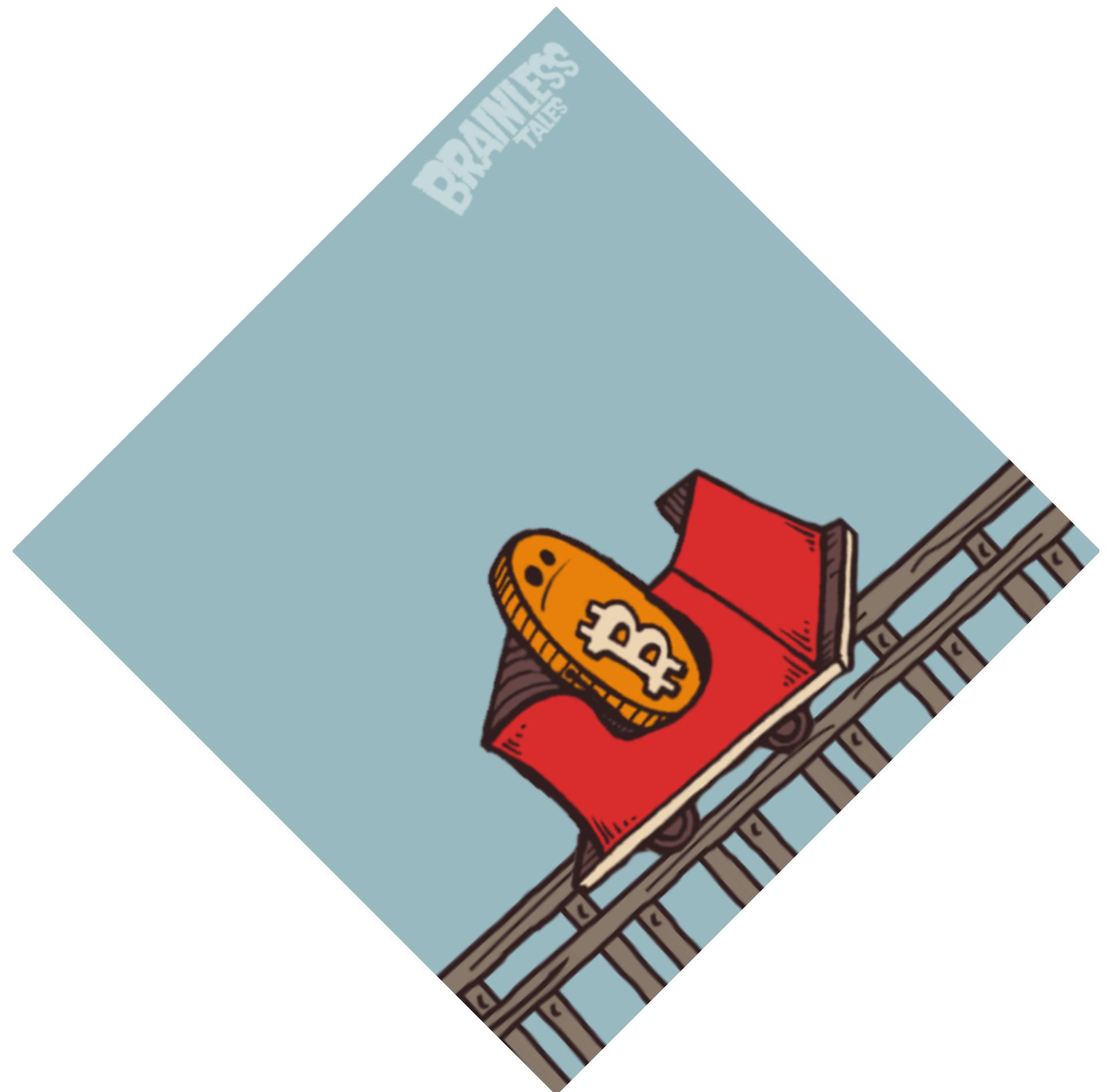
**What's the catch?**

# More data to download

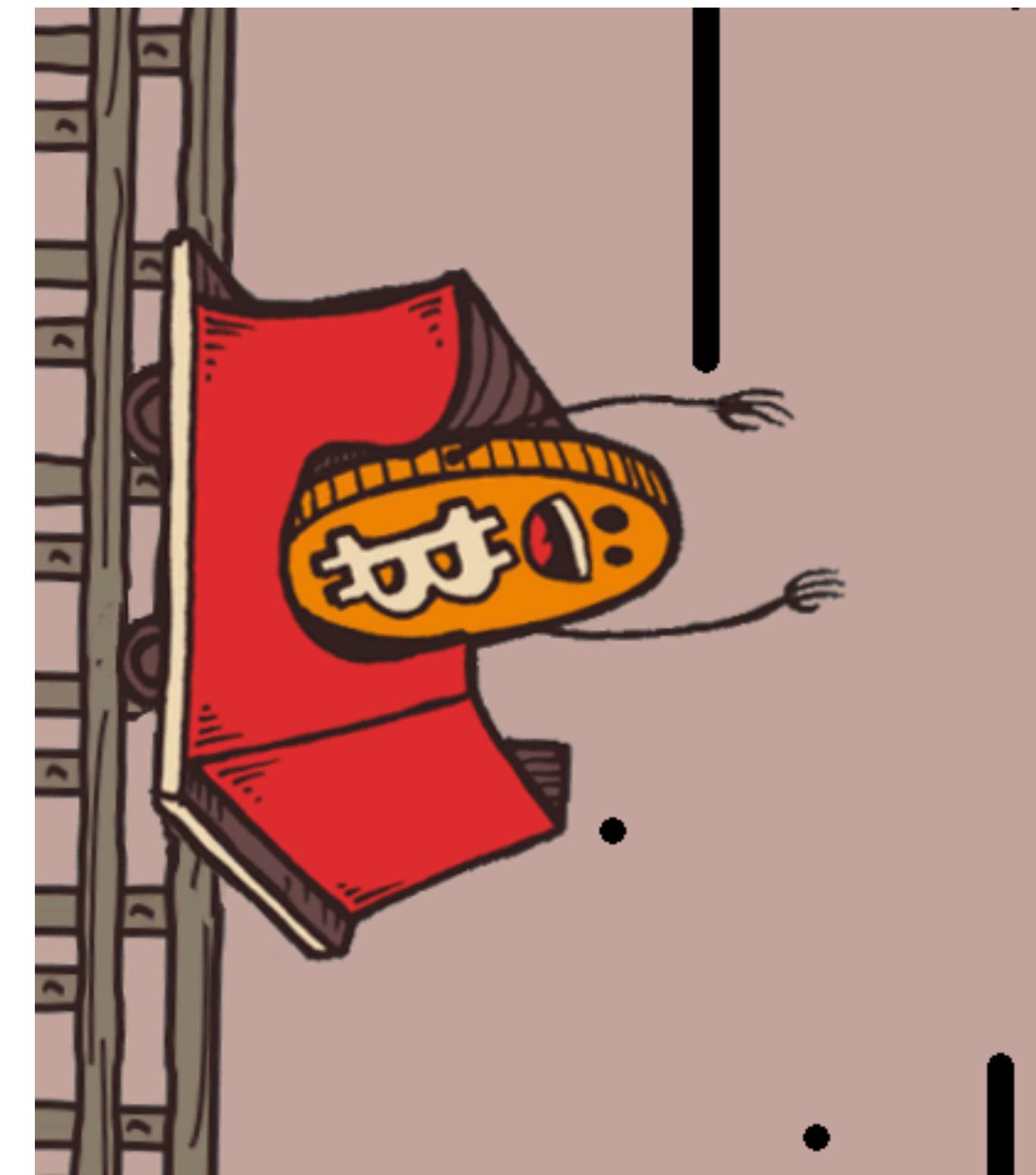
Extra data that a utreexo node will download

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

# Bandwidth CPU



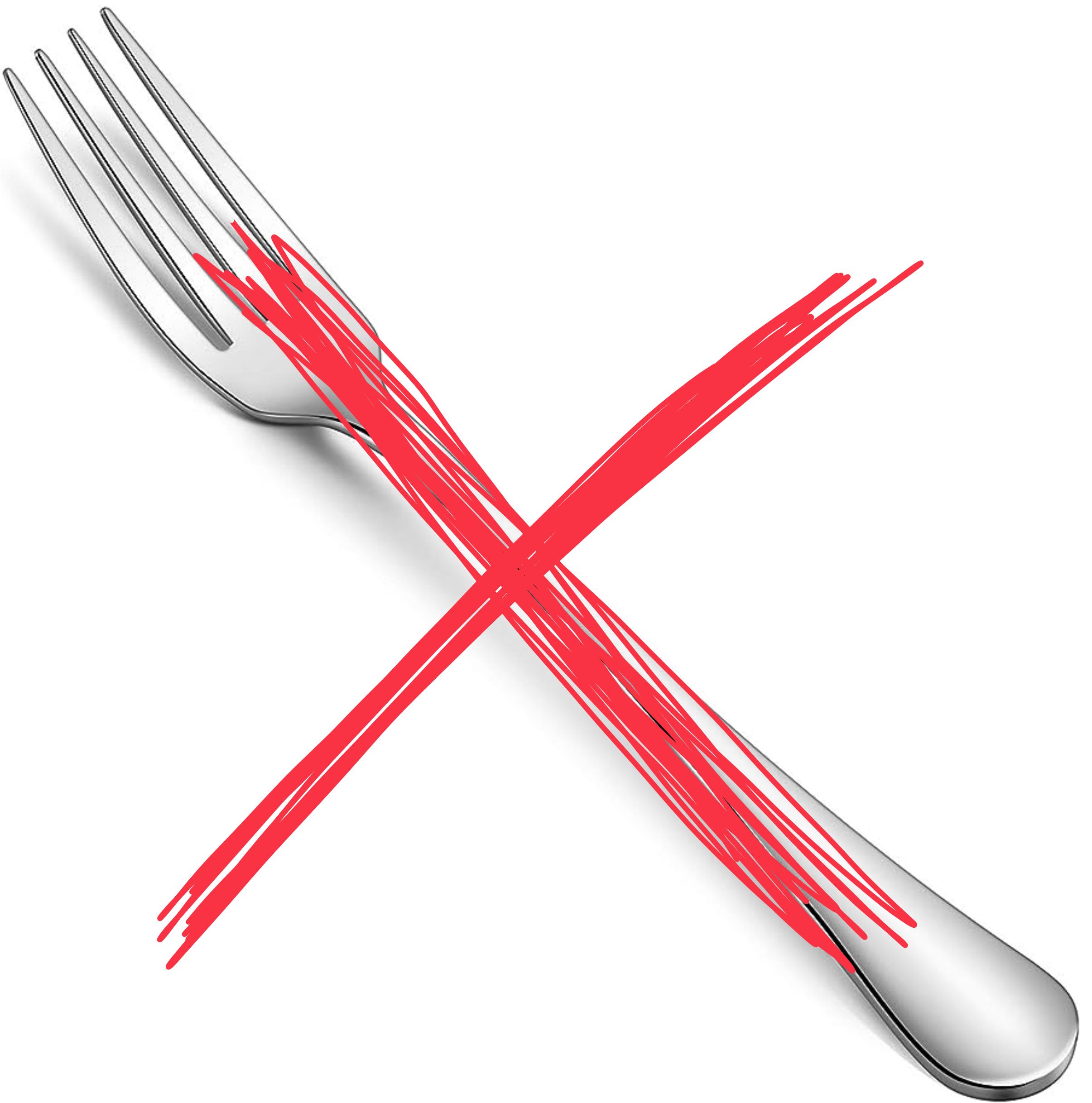
# disk i/o RAM usage



# 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](https://github.com/utreexo/utreexod)

# Aligning incentives

**No burden of  
UTXO storage**

coinbase

coinbase

The Binance logo icon is a yellow diamond shape composed of eight smaller yellow triangles pointing inward toward the center.

BINANCE

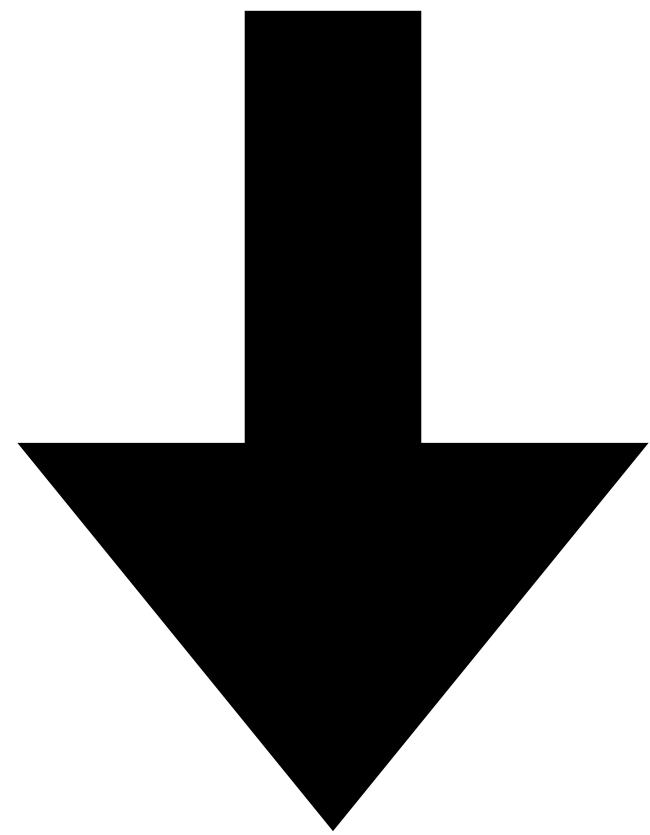
# coinbase



## BITCOIN STAMPS

*Unprunable UTXO Art, Because Sats Don't Exist.*

# **Everyone**



# **Each store their own**

# How does it work?

# Algorithms

Consensus critical stuff

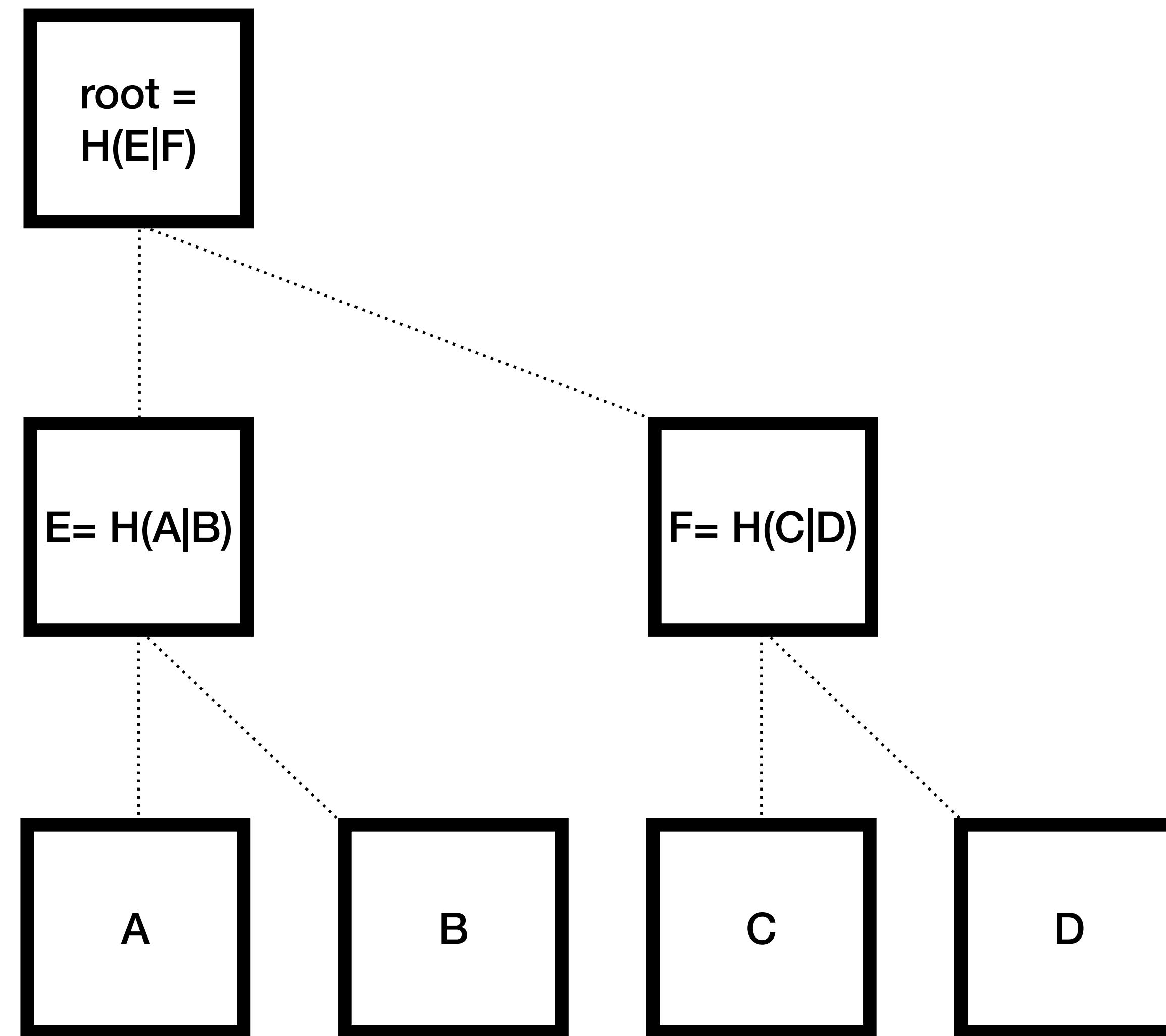
1. Add a UTXO
2. Delete a UTXO
3. Verify the existence

# **Algorithms**

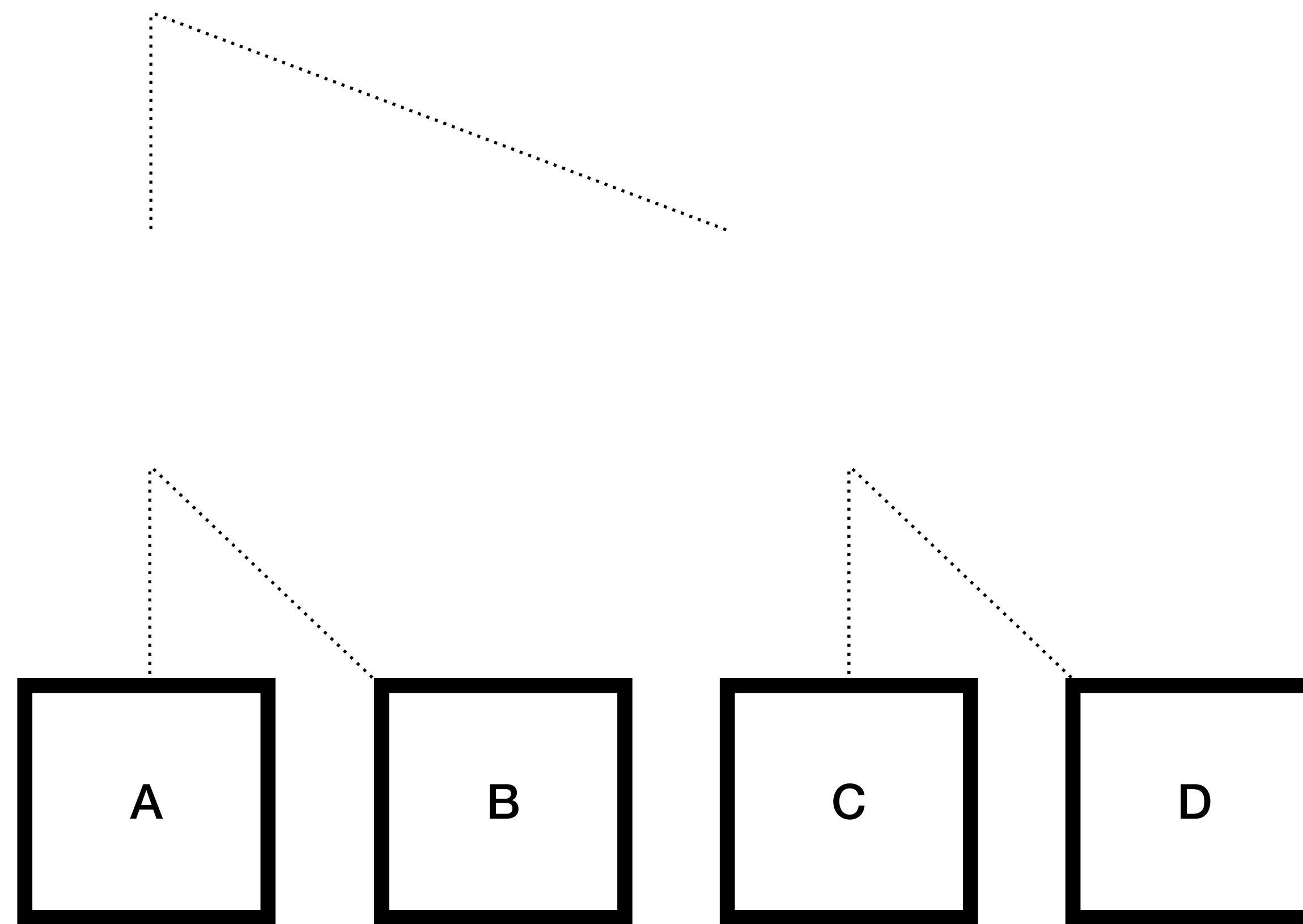
**Consensus critical stuff**

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

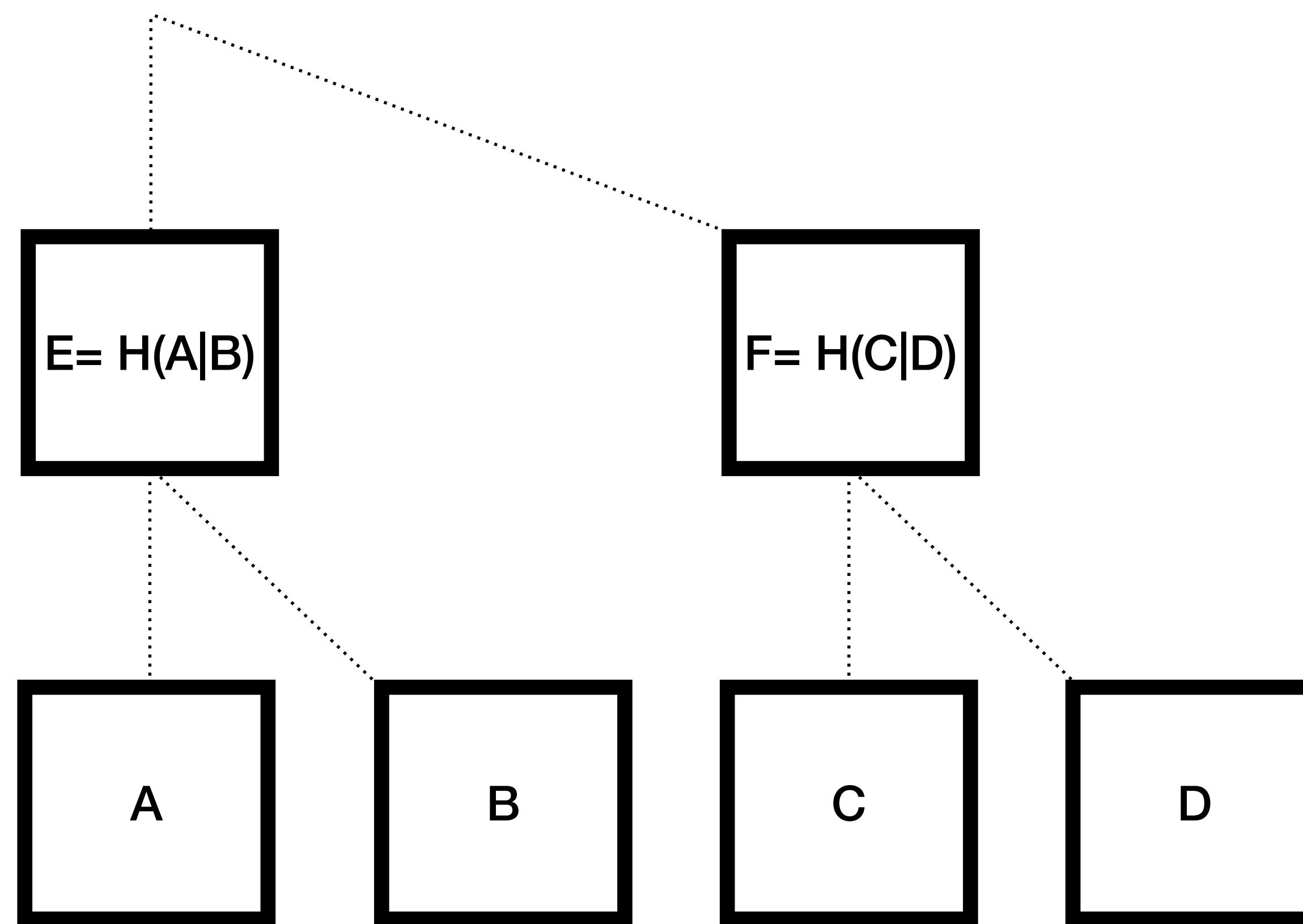
# Quick review of merkle tree creation



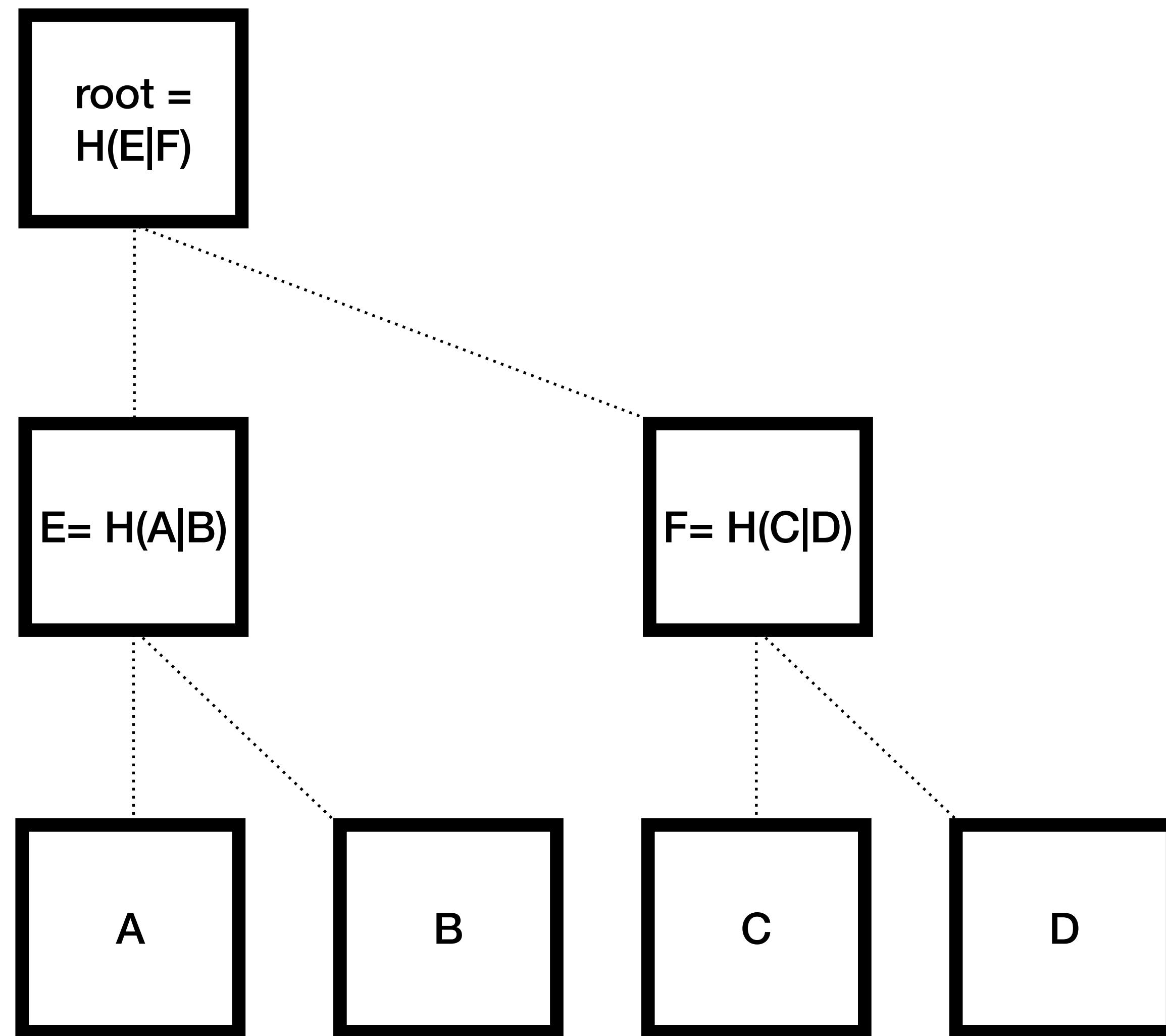
# Start off with the hash of the elements



# Hash each pairs

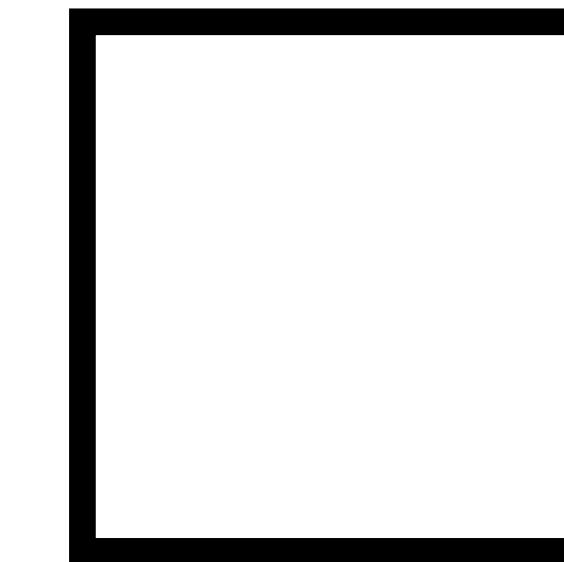


# Hash last pair

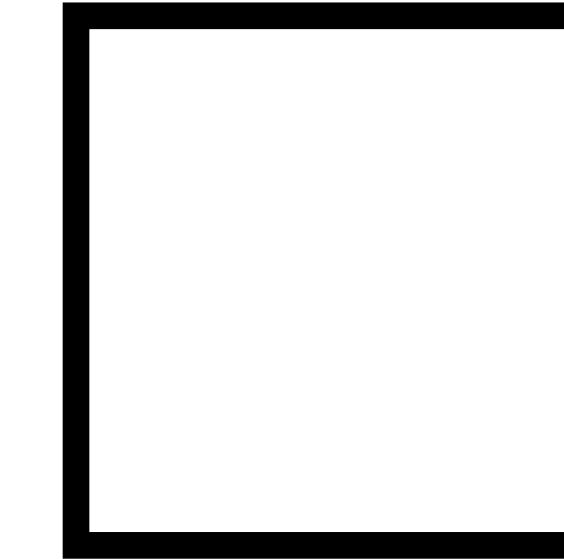
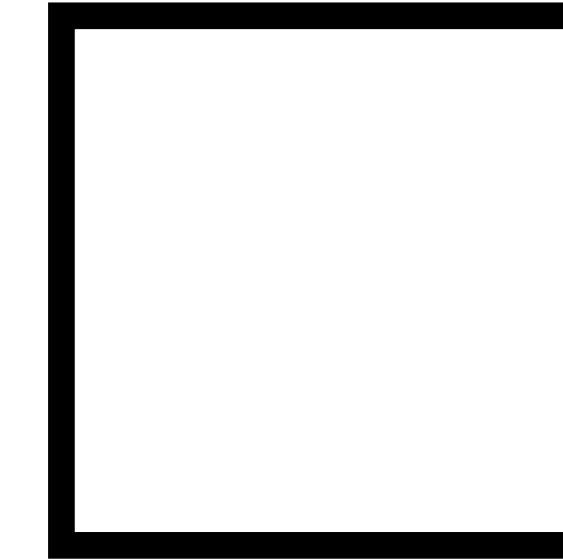


# **1 UTXO**

## **Single root**

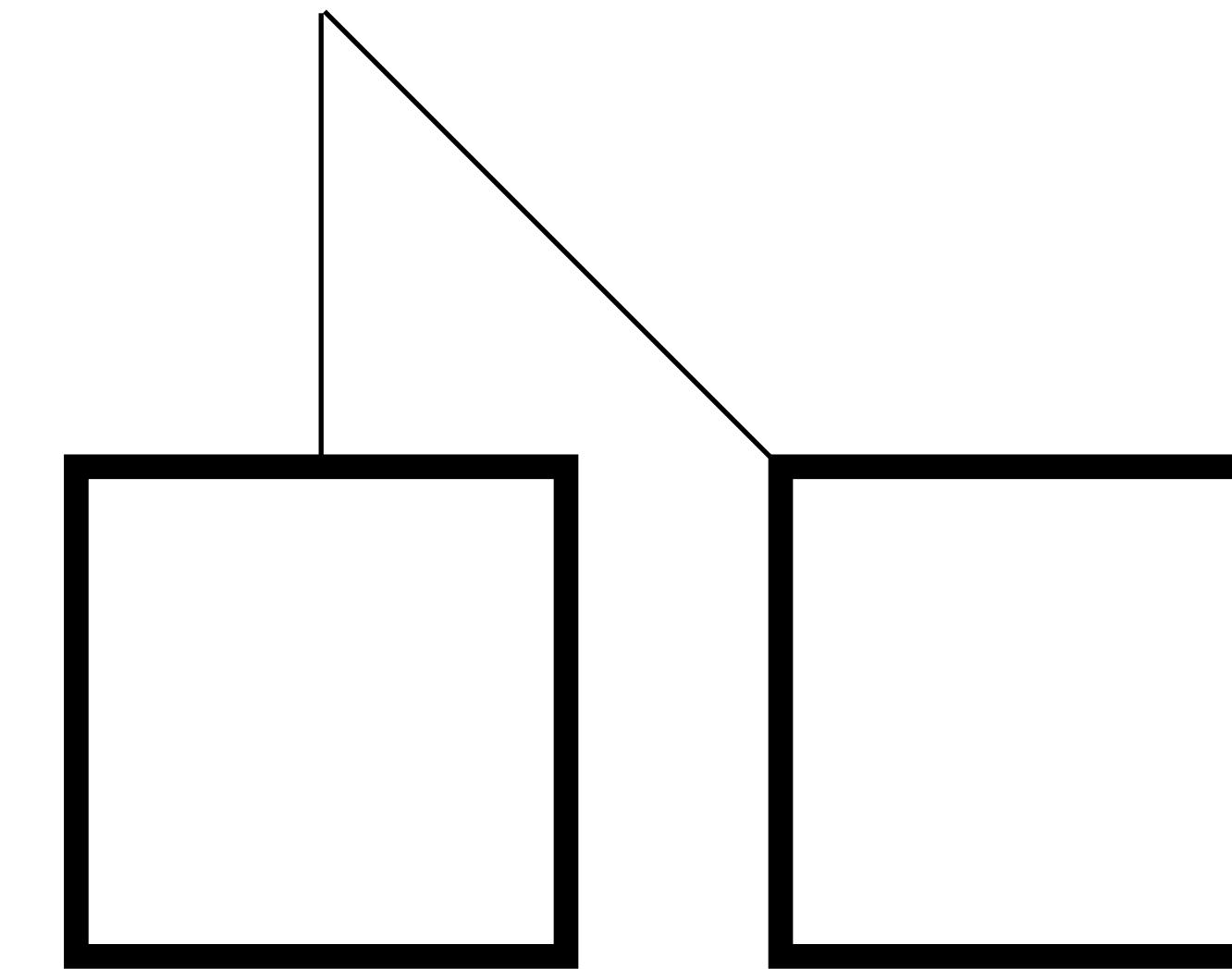


# 2 UTXOs



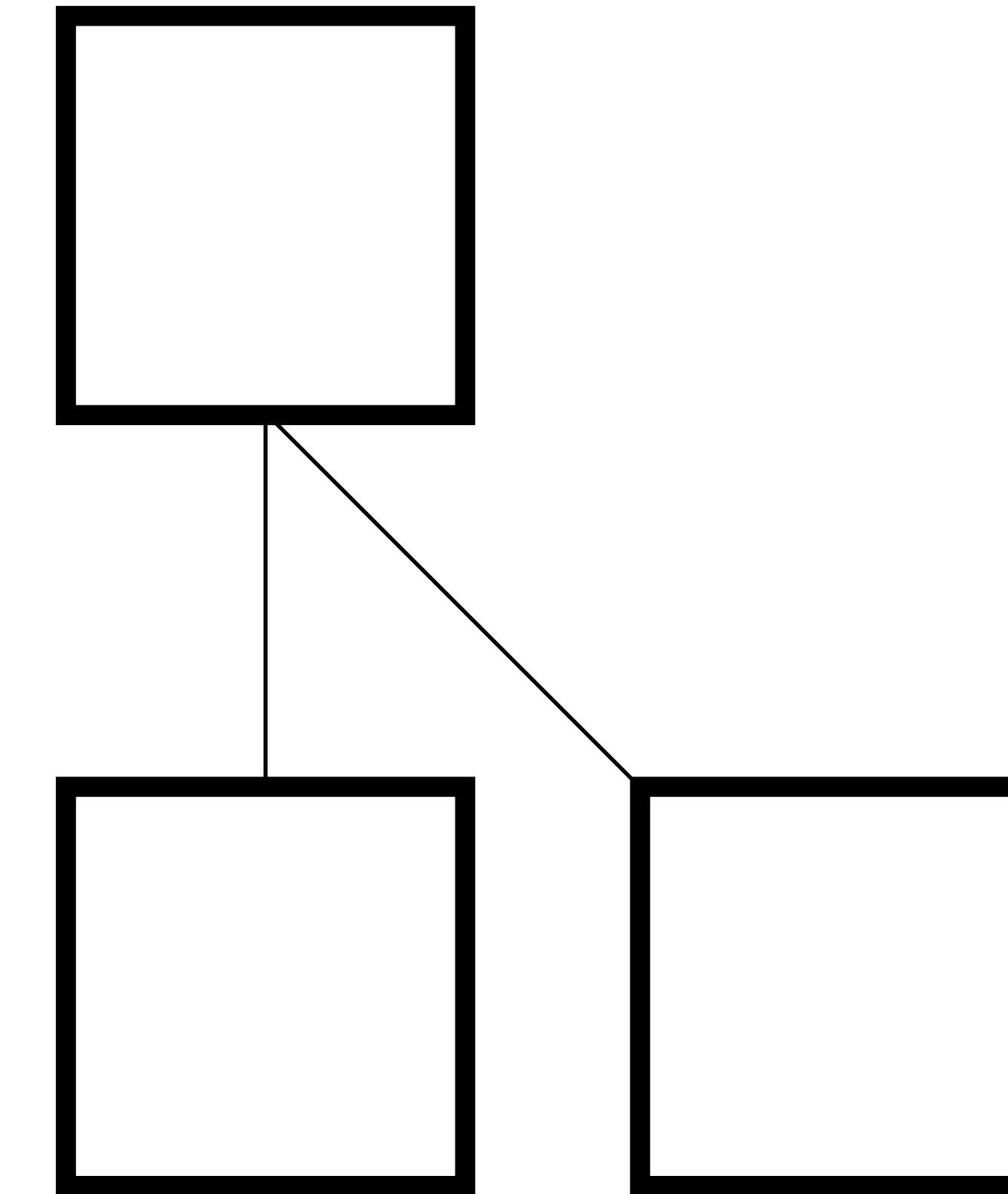
# 2 UTXOs

## Concatenate



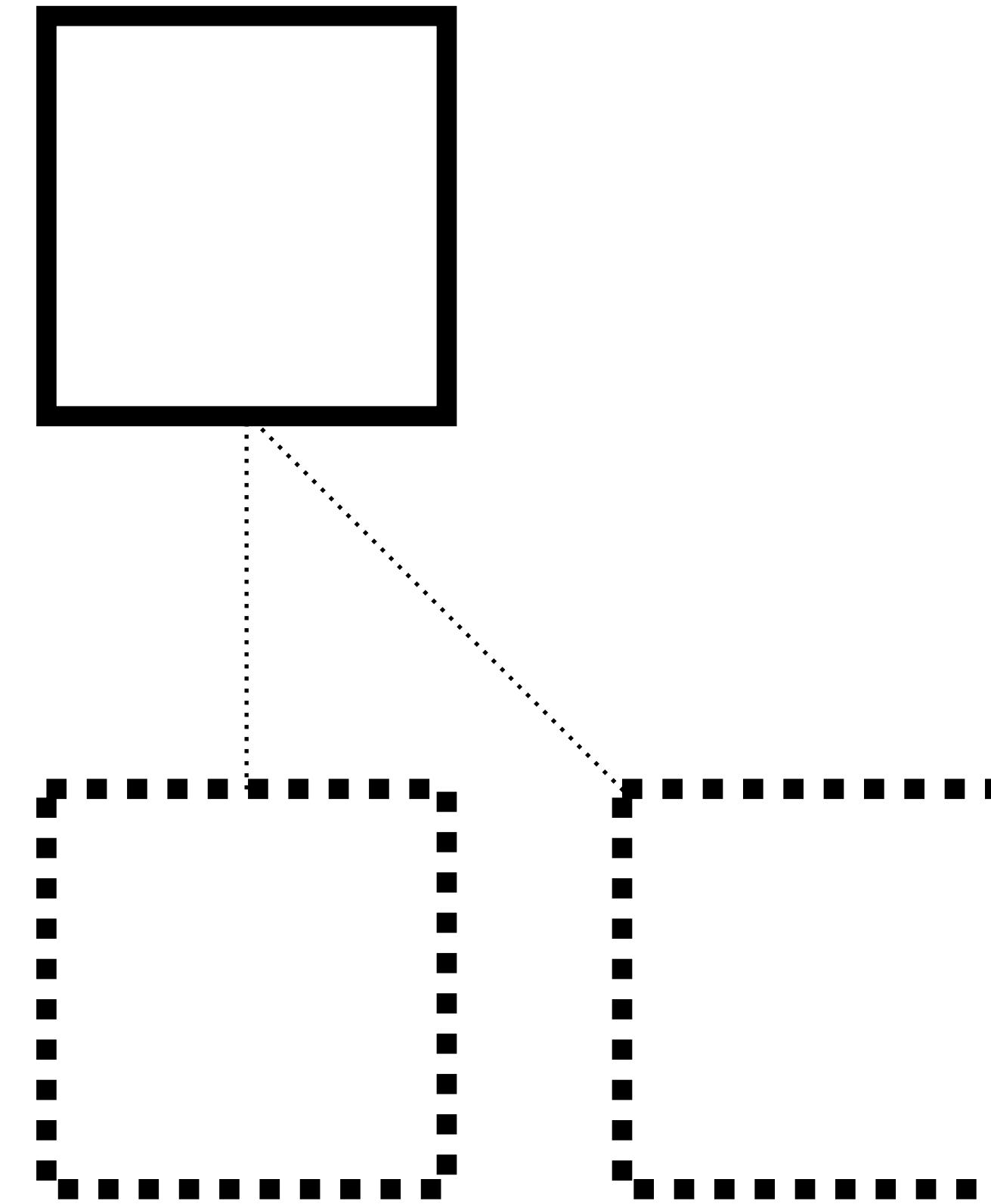
# 2 UTXOs

## Hash to create a new root

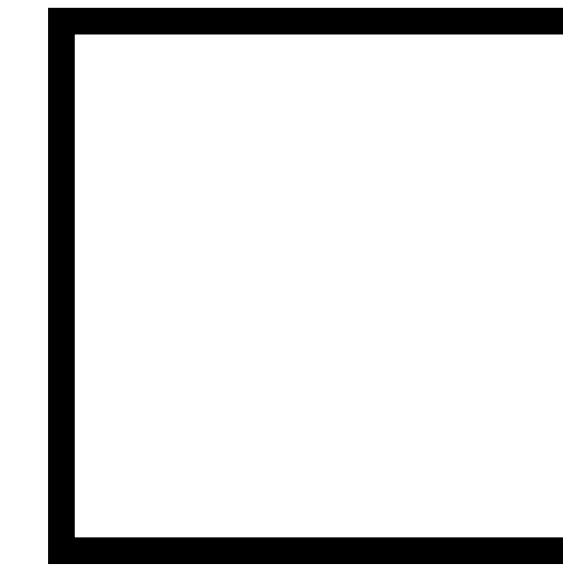


# 2 UTXOs

Can now delete the leaves



# 2 UTXOs



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

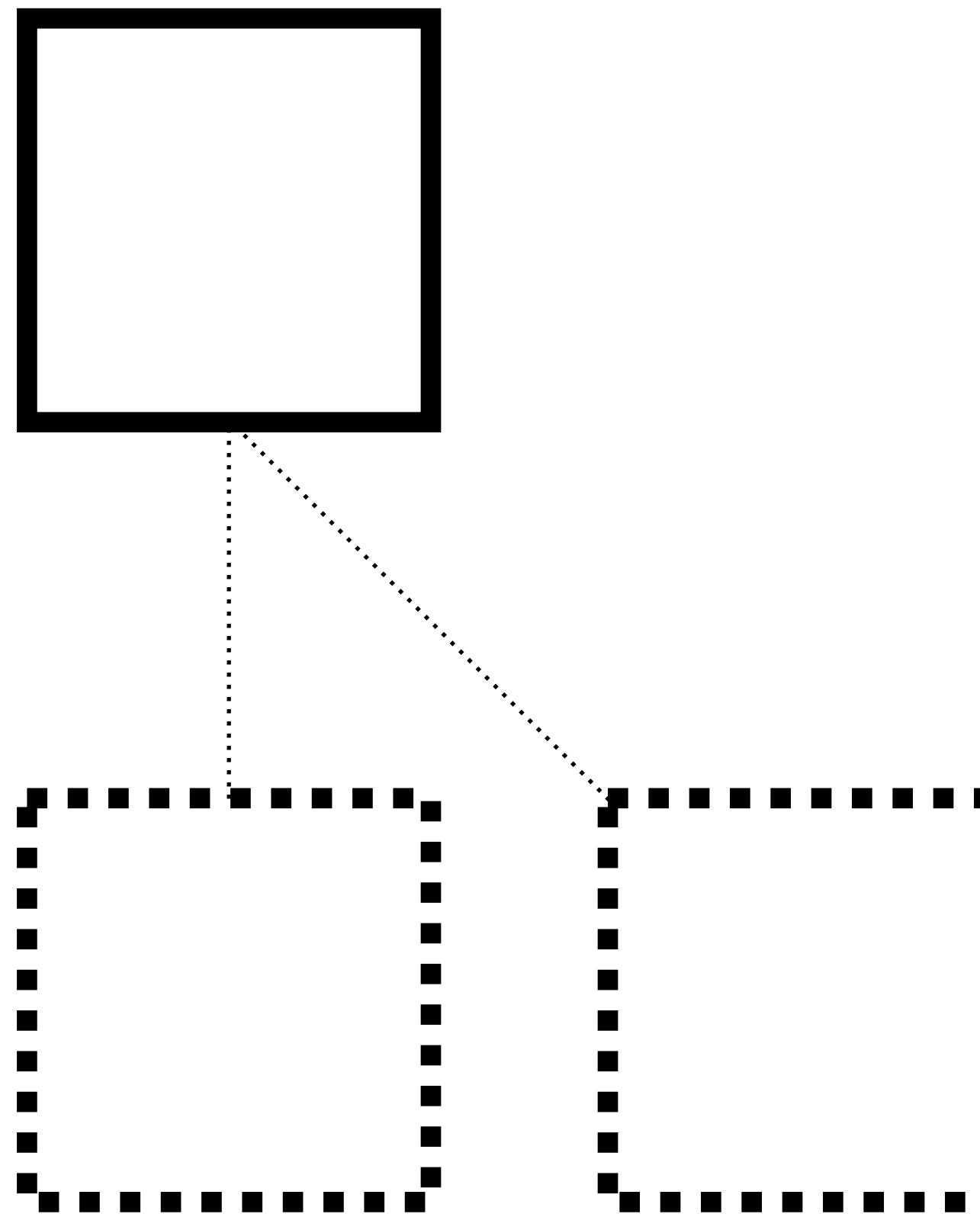
# 3 UTXOs

Add 1 more



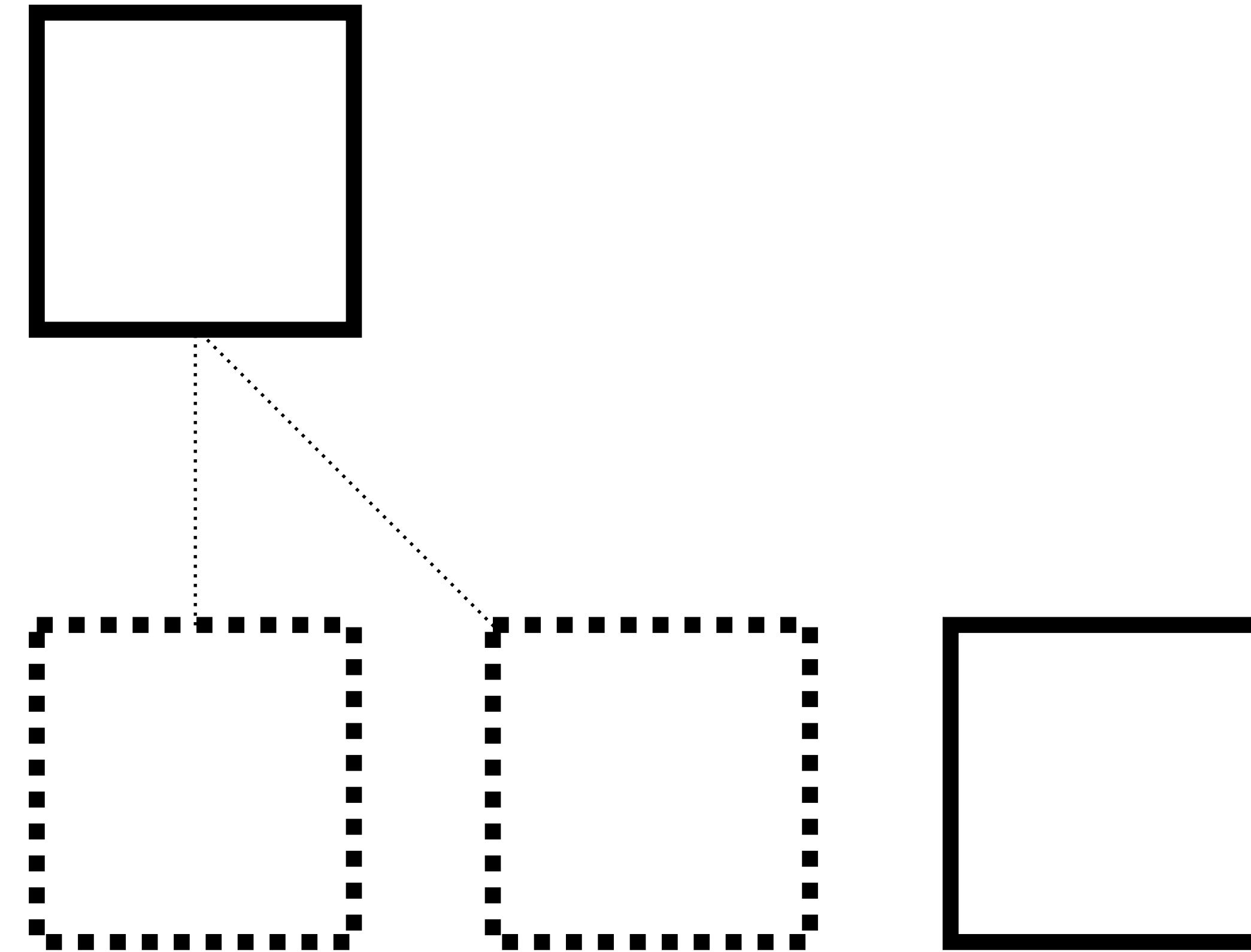
# 3 UTXOs

Add 1 more



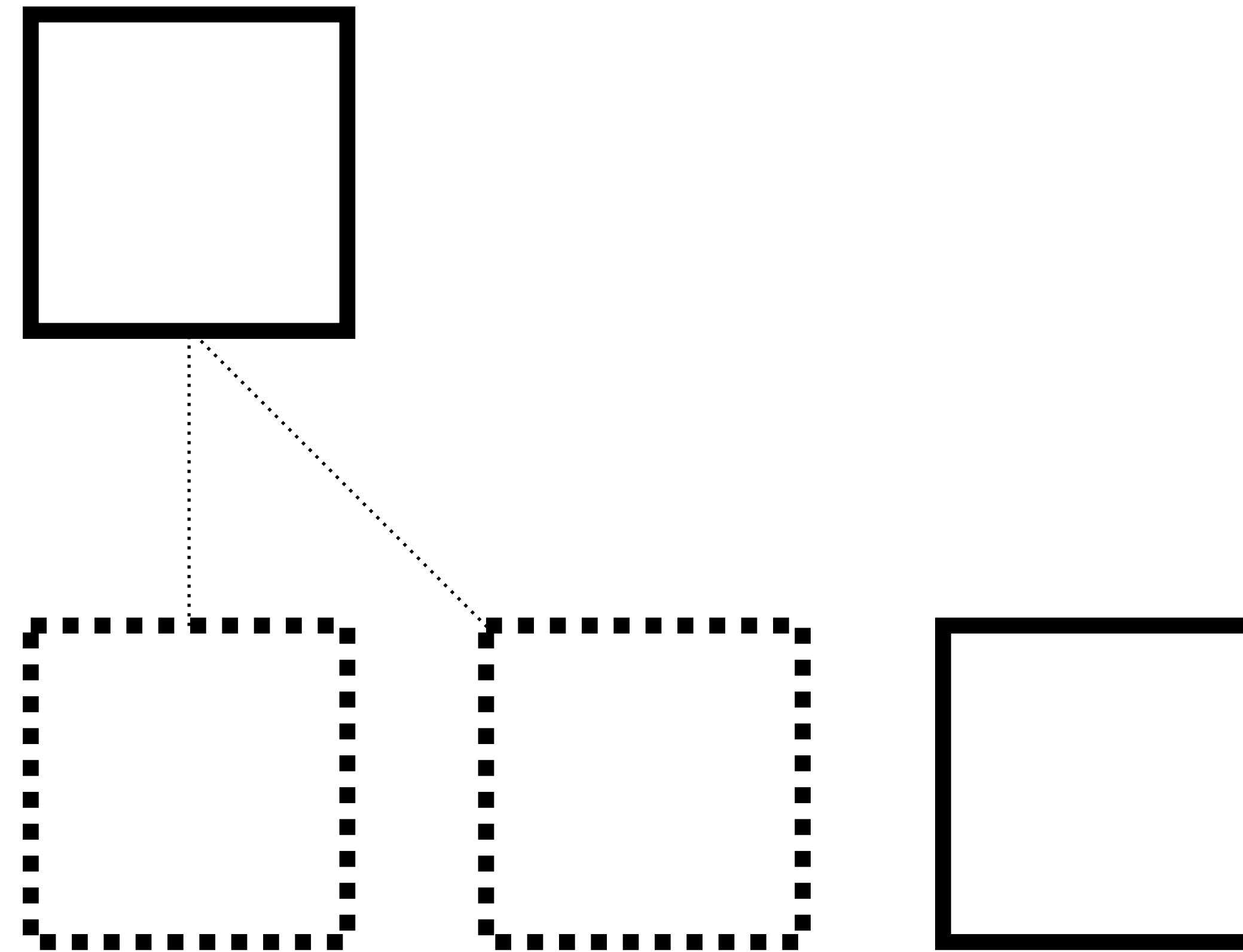
# 3 UTXOs

Add 1 more



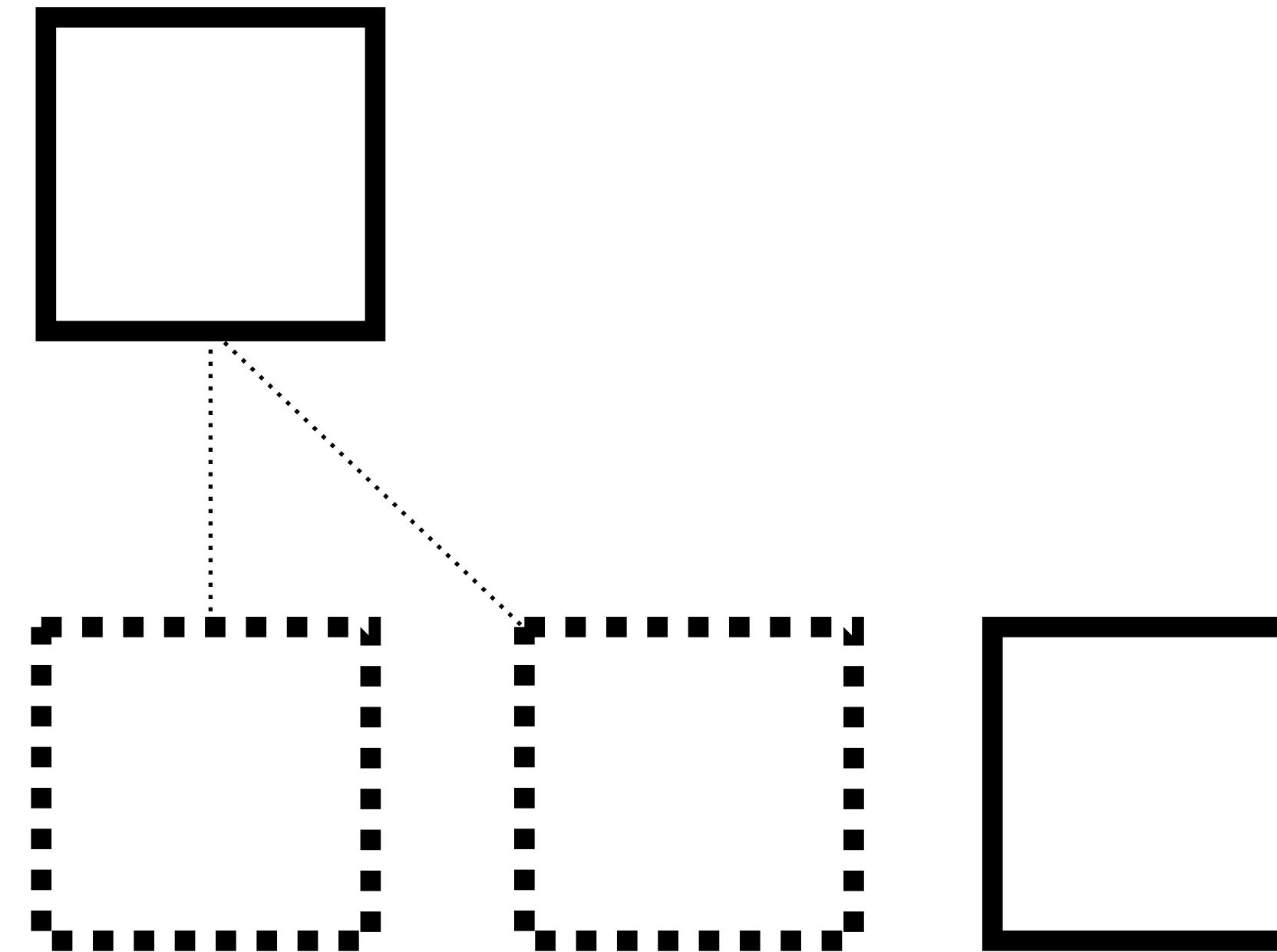
# 3 UTXOs

**Nothing to hash with. Becomes a root**



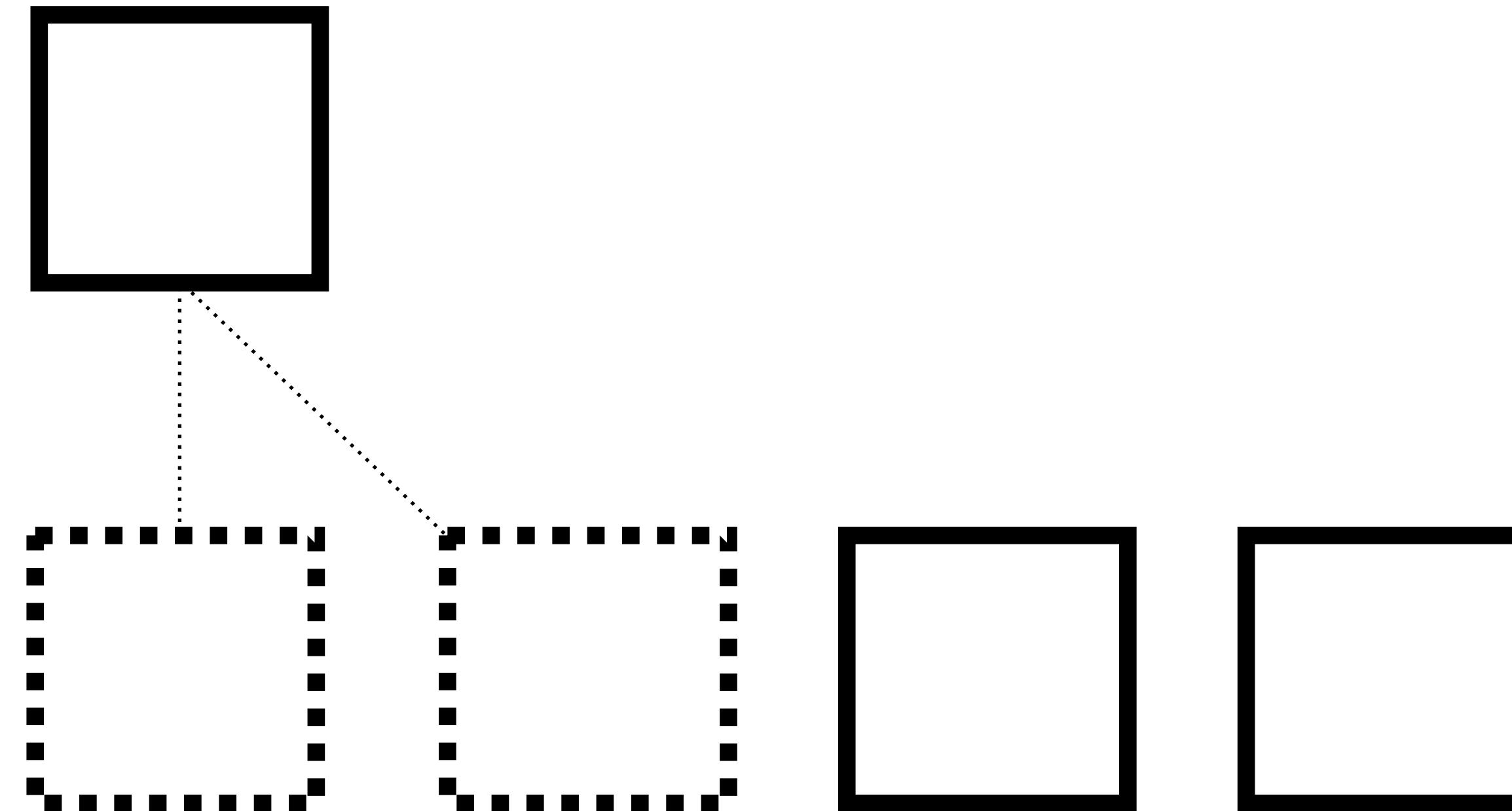
# 4 UTXOs

Another one

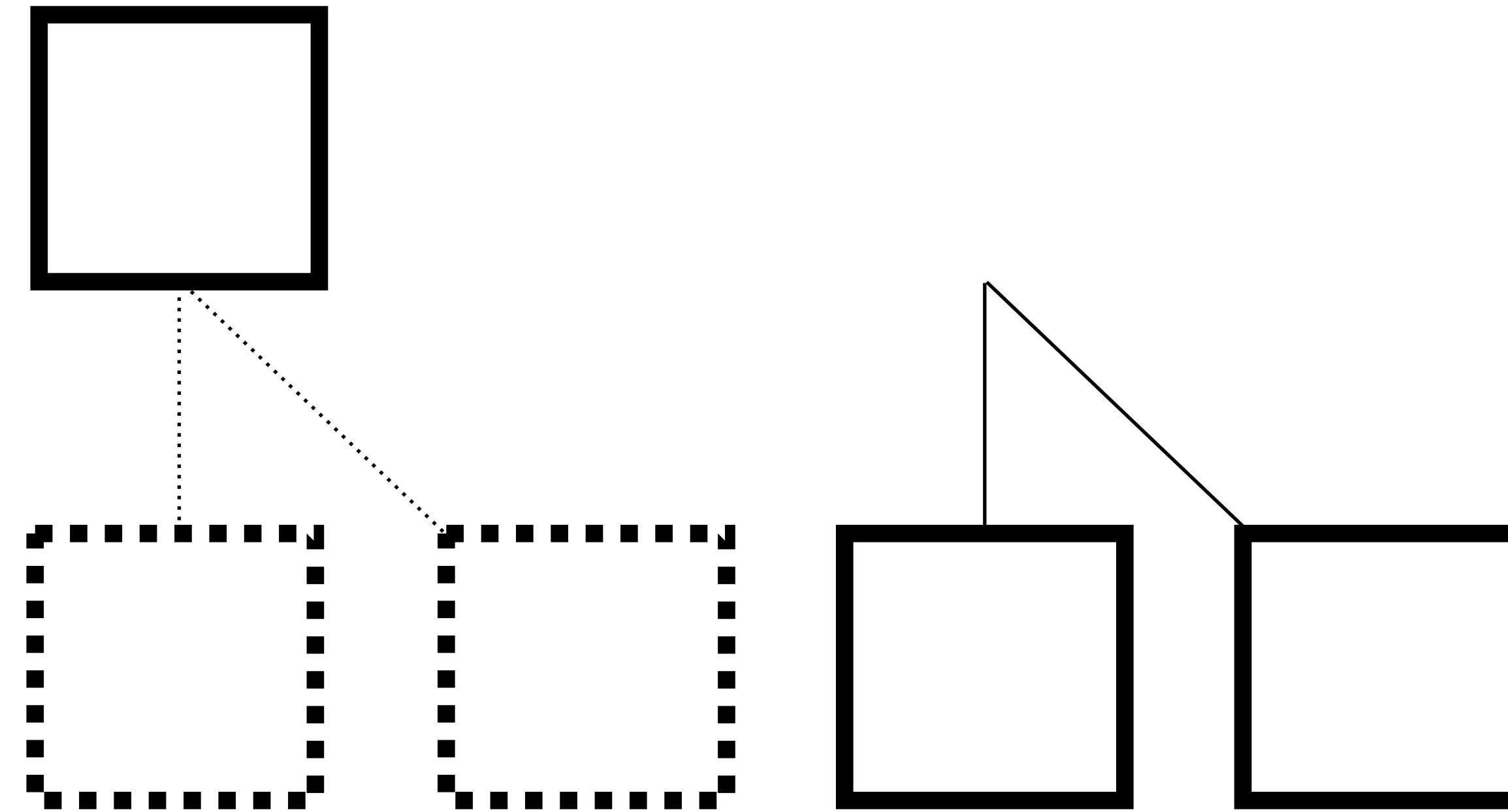


# 4 UTXOs

Another one

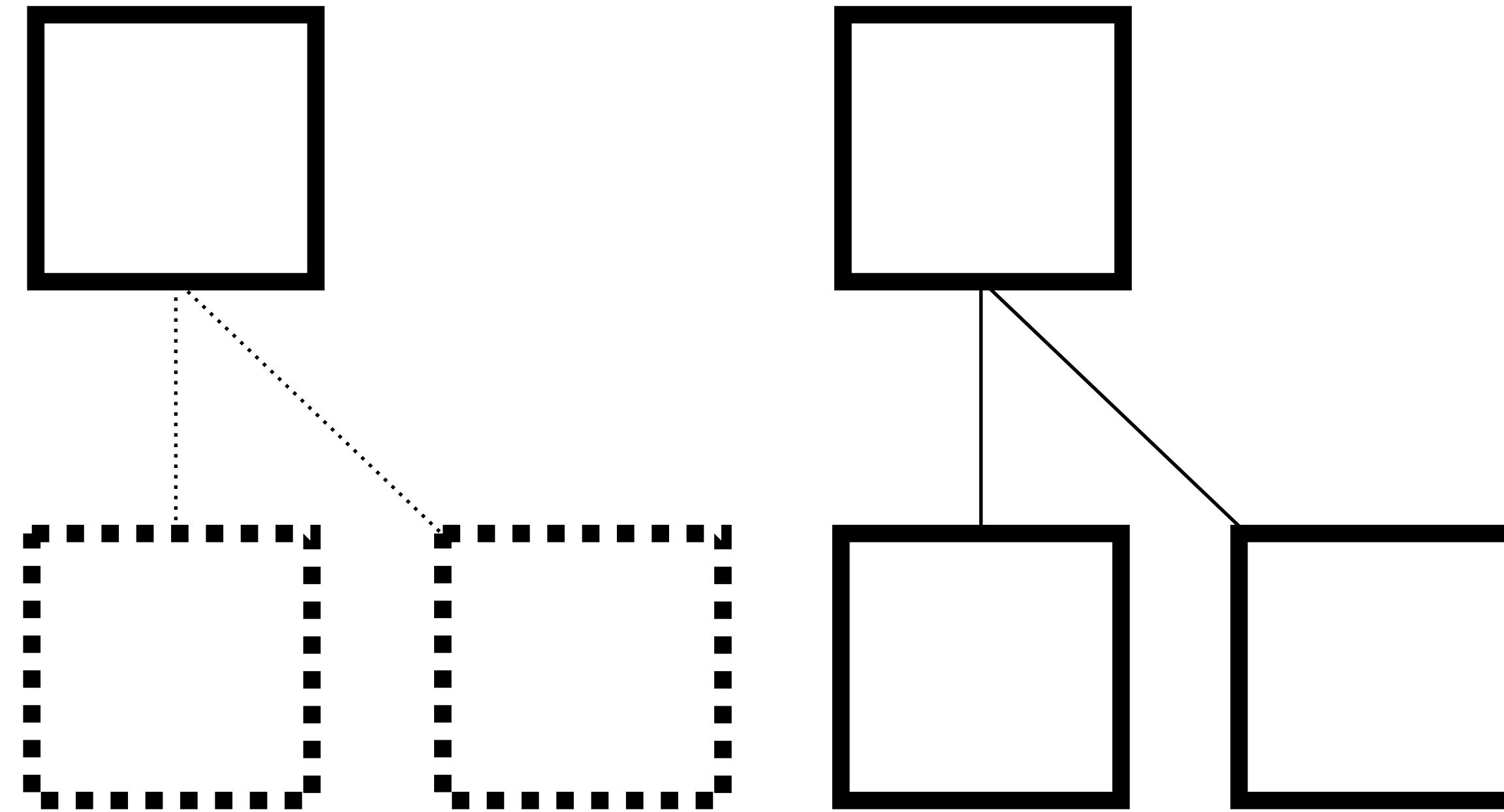


# 4 UTXOs Concatenate



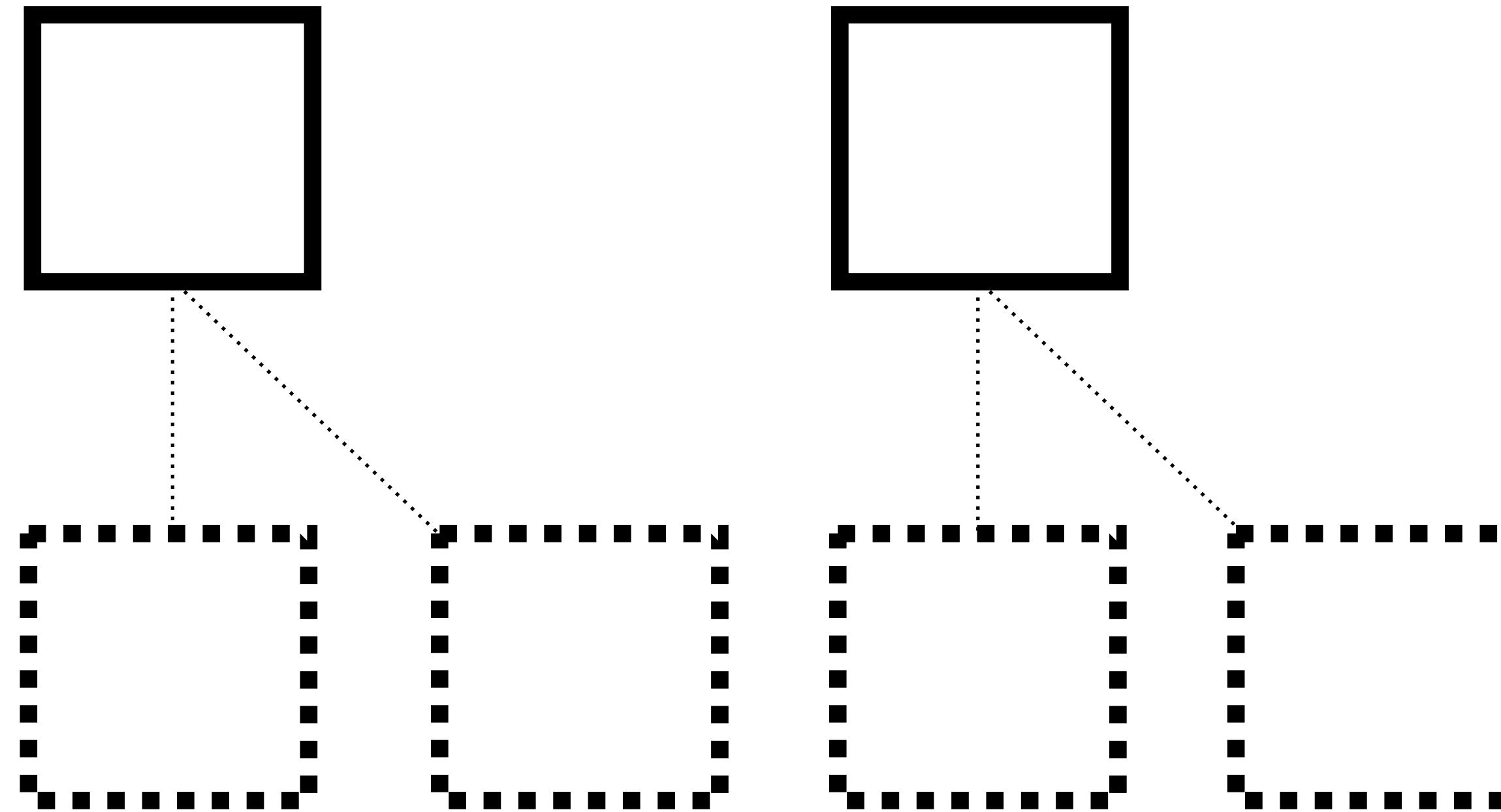
# 4 UTXOs

## Hash

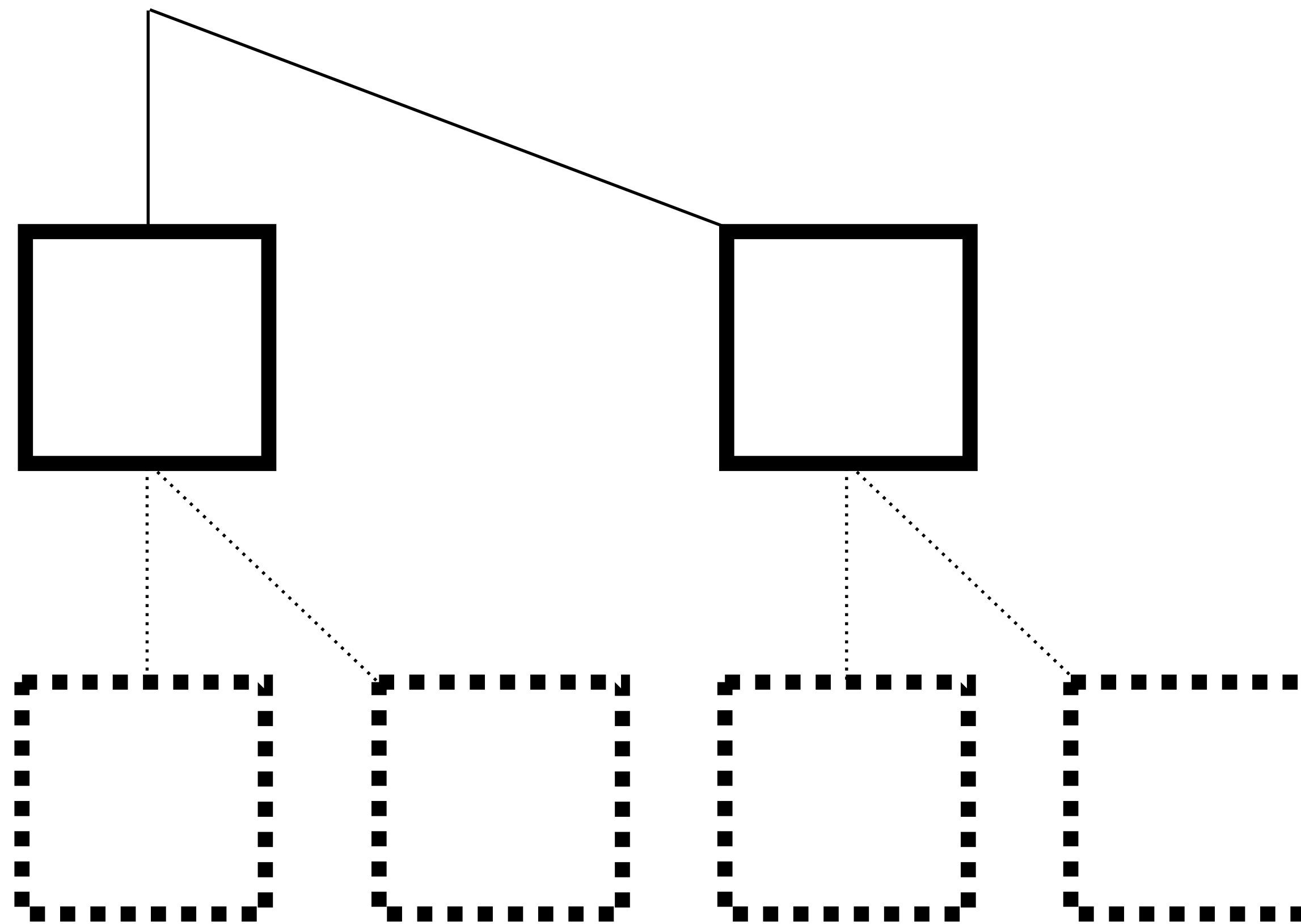


# 4 UTXOs

Can now throw away the leaves

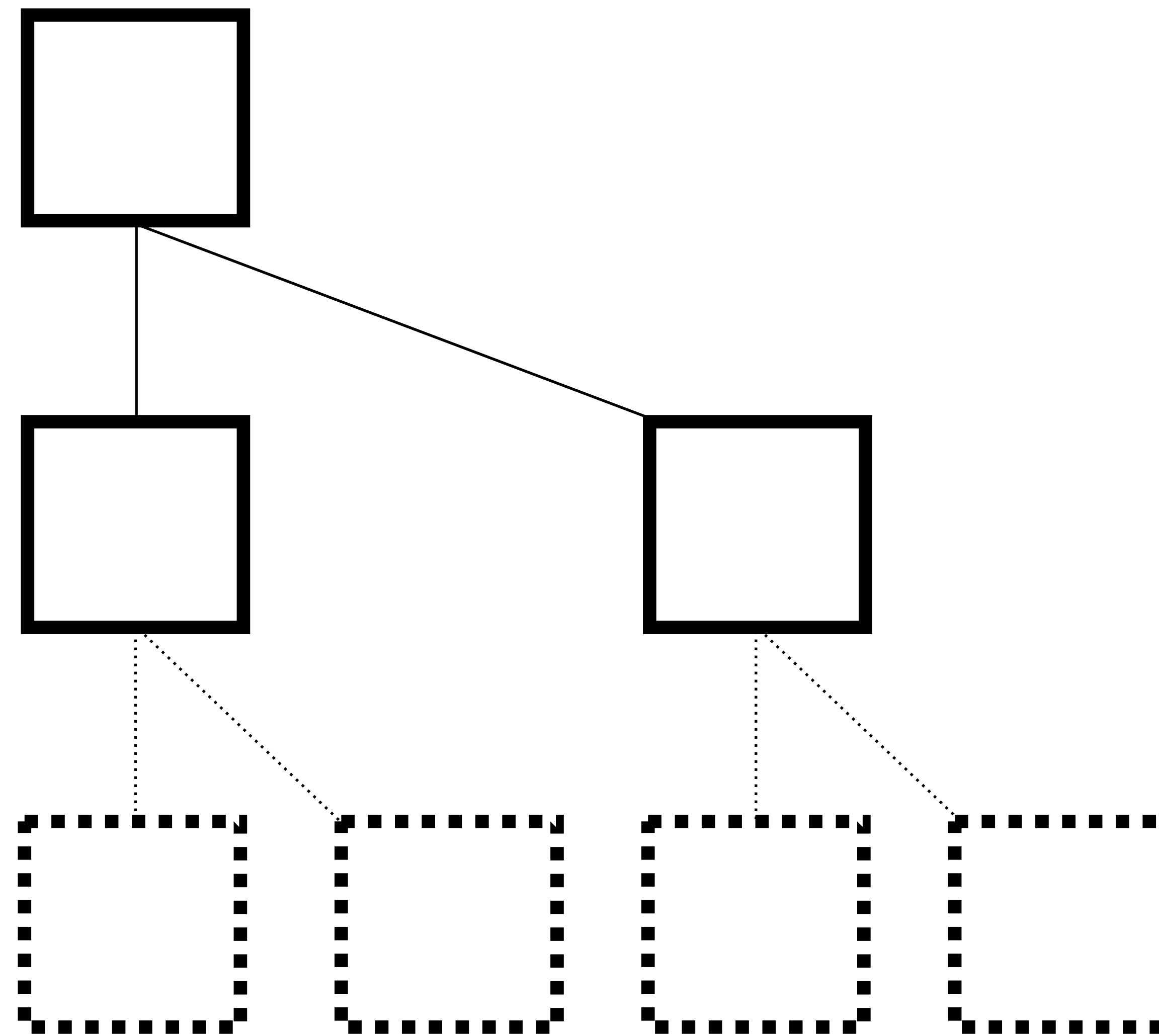


# 4 UTXOs Concatenate



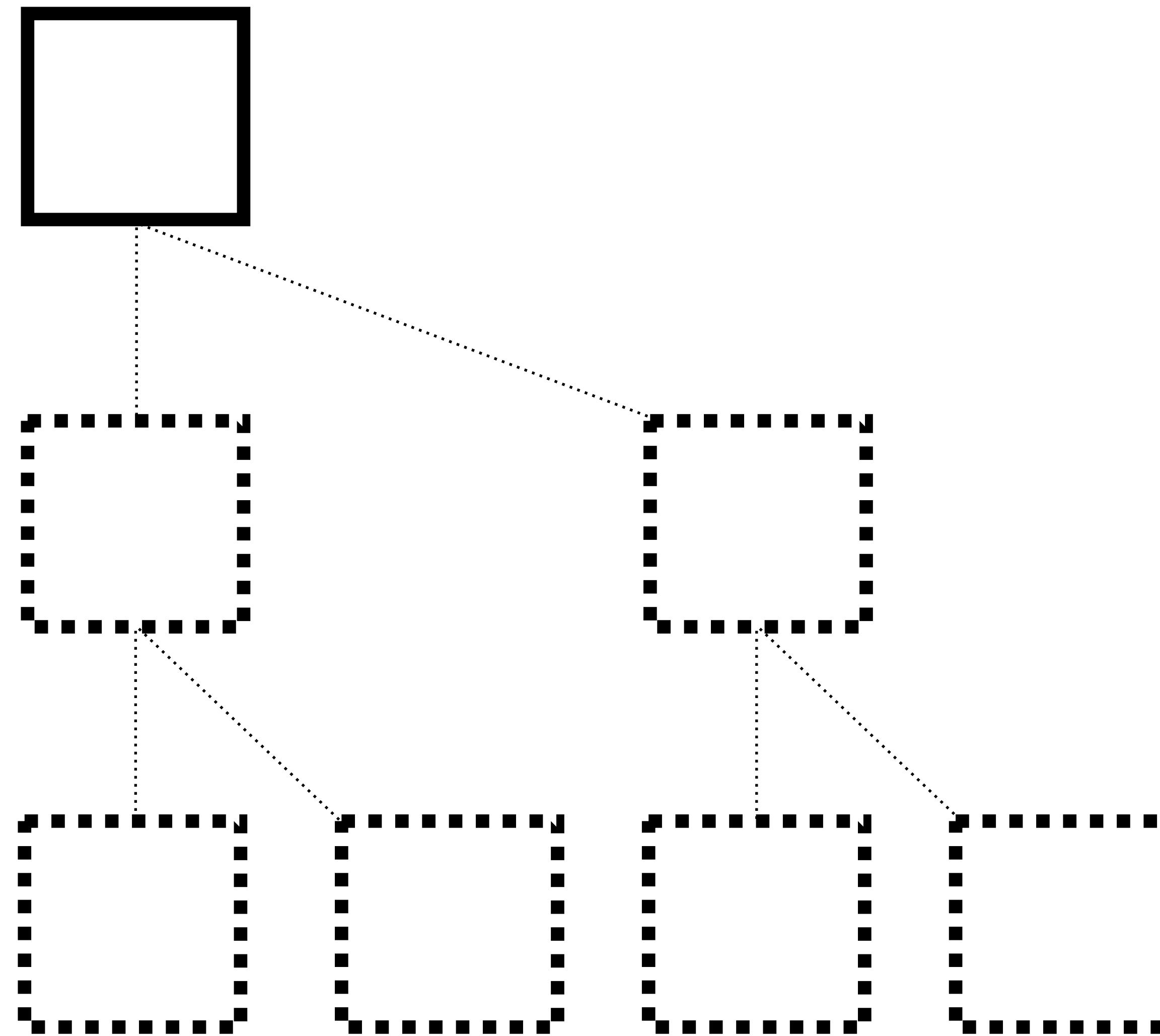
# 4 UTXOs

Hash



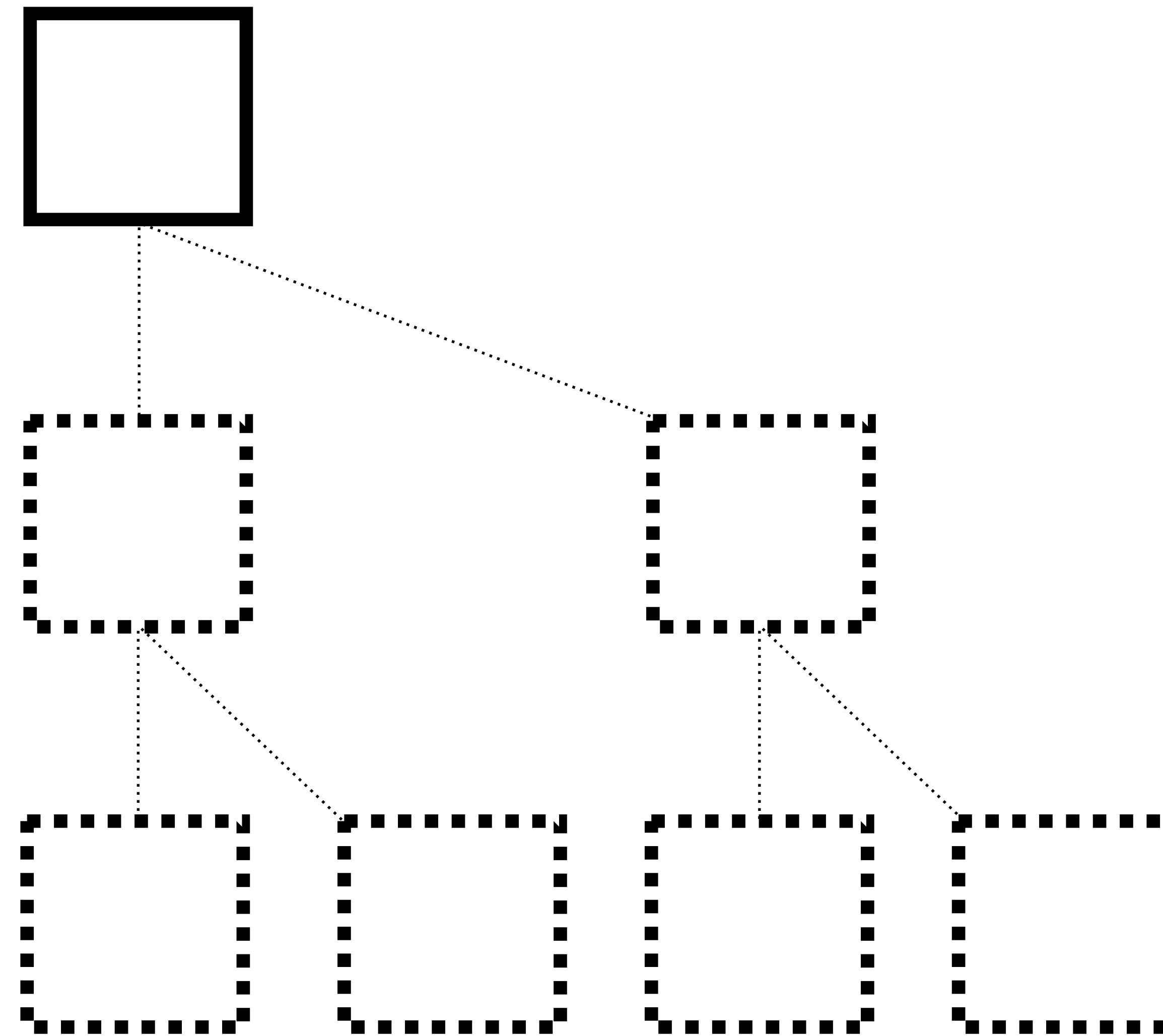
# 4 UTXOs

## Throw away



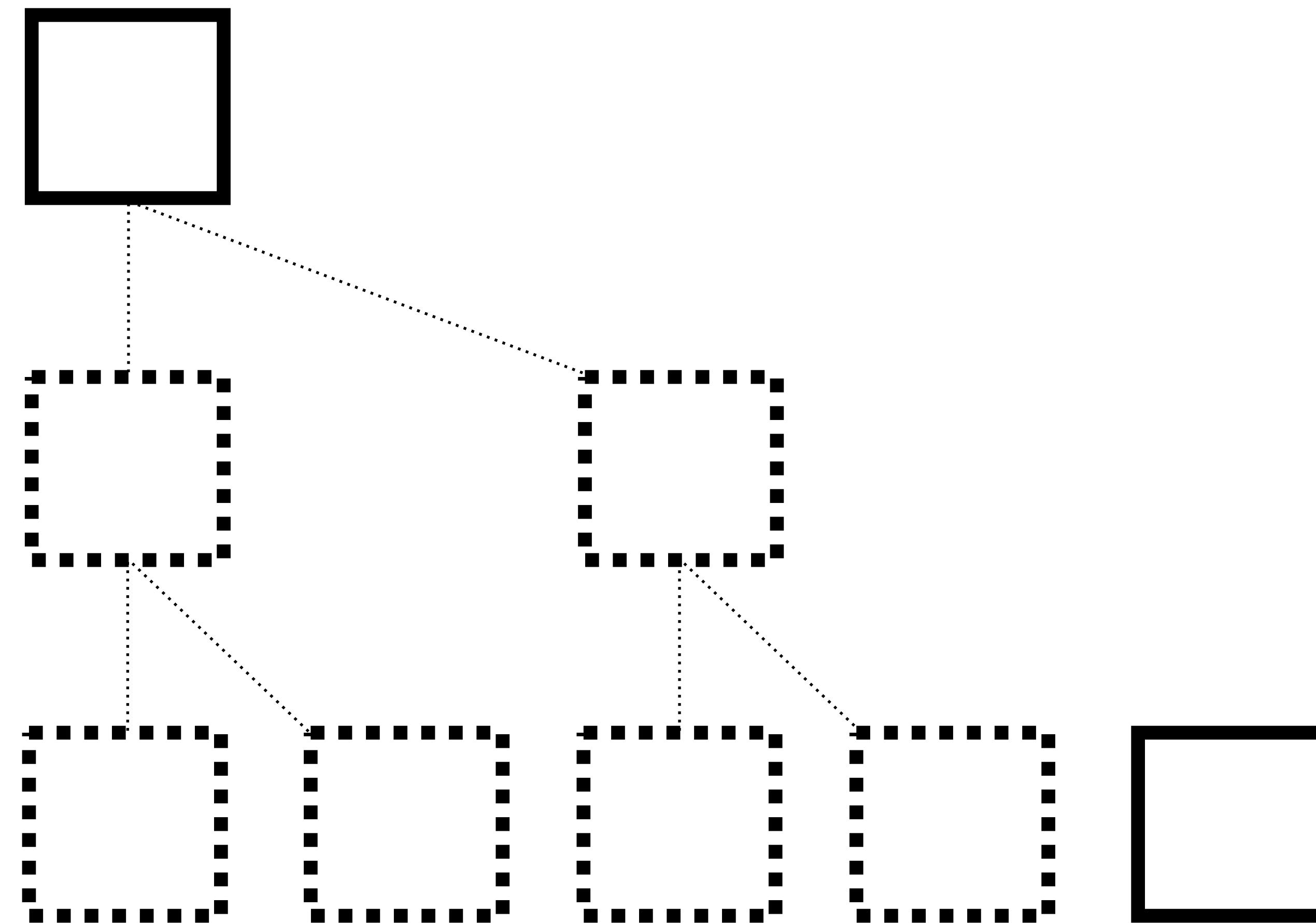
# 4 UTXOs

## Finished tree



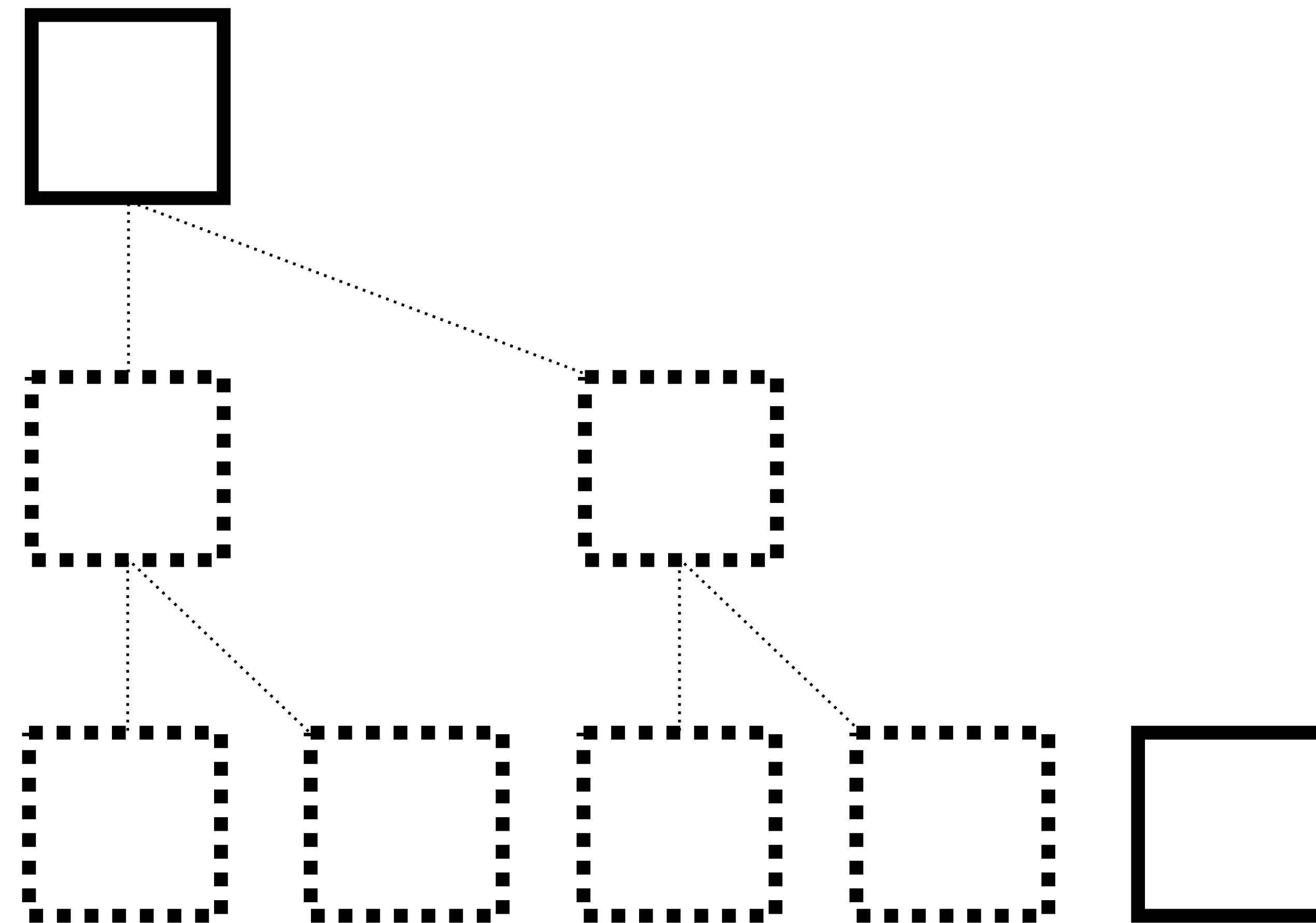
# 5 UTXOs

Add one more



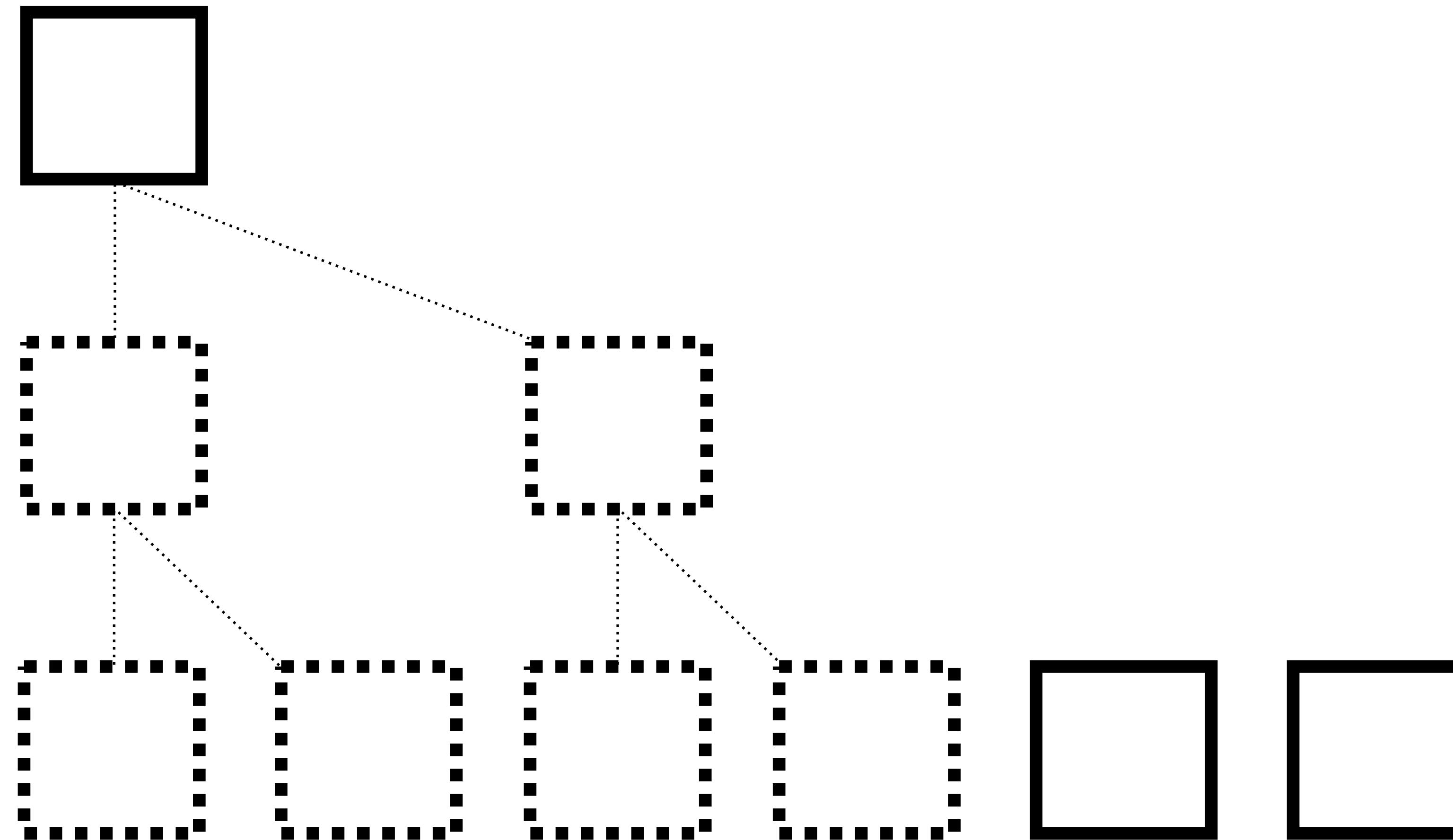
# 5 UTXOs

**Nothing to hash with so it becomes a root**

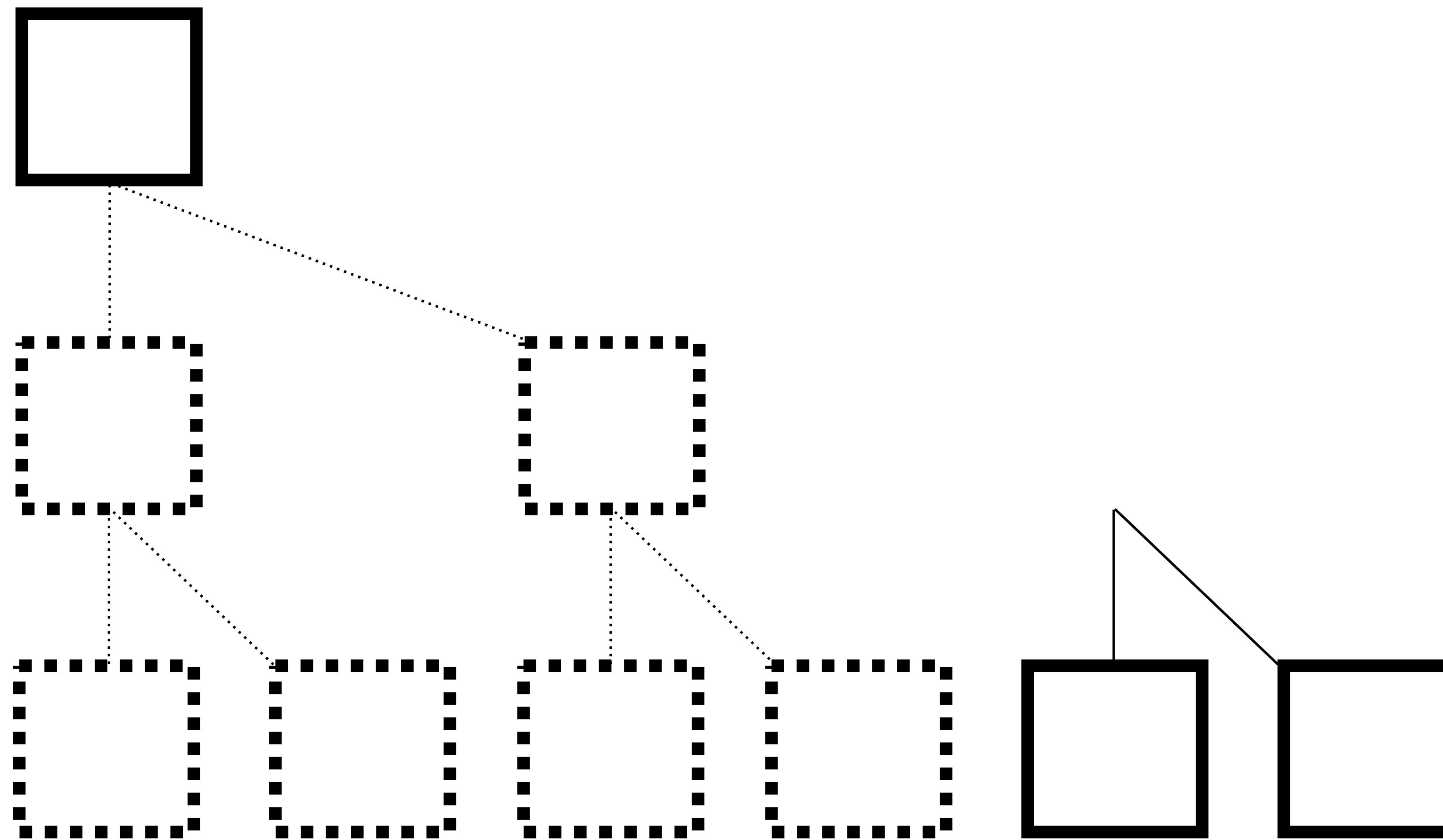


# 6 UTXOs

1 more

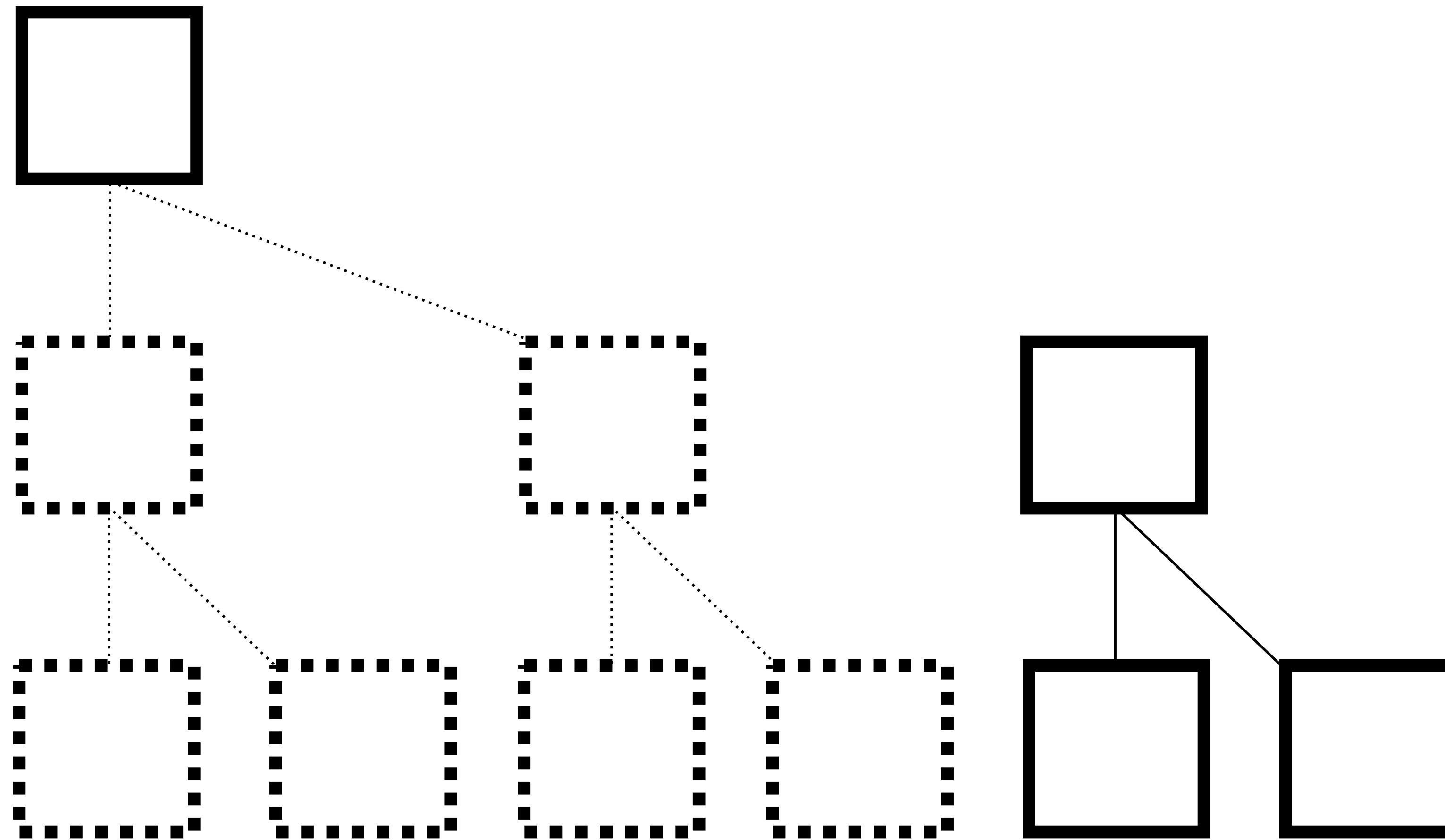


# 6 UTXOs Concatenate



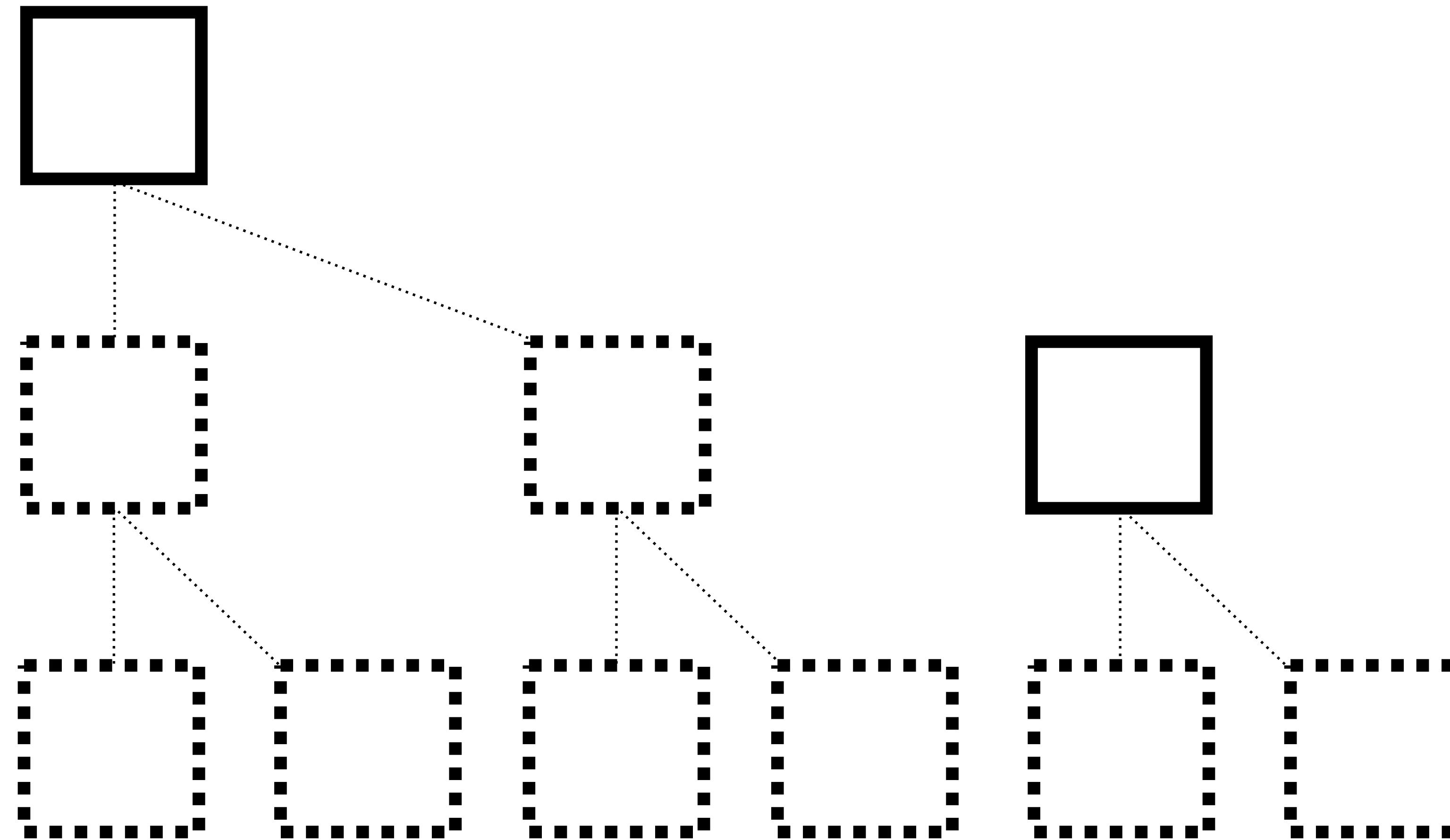
# 6 UTXOs

## Hash



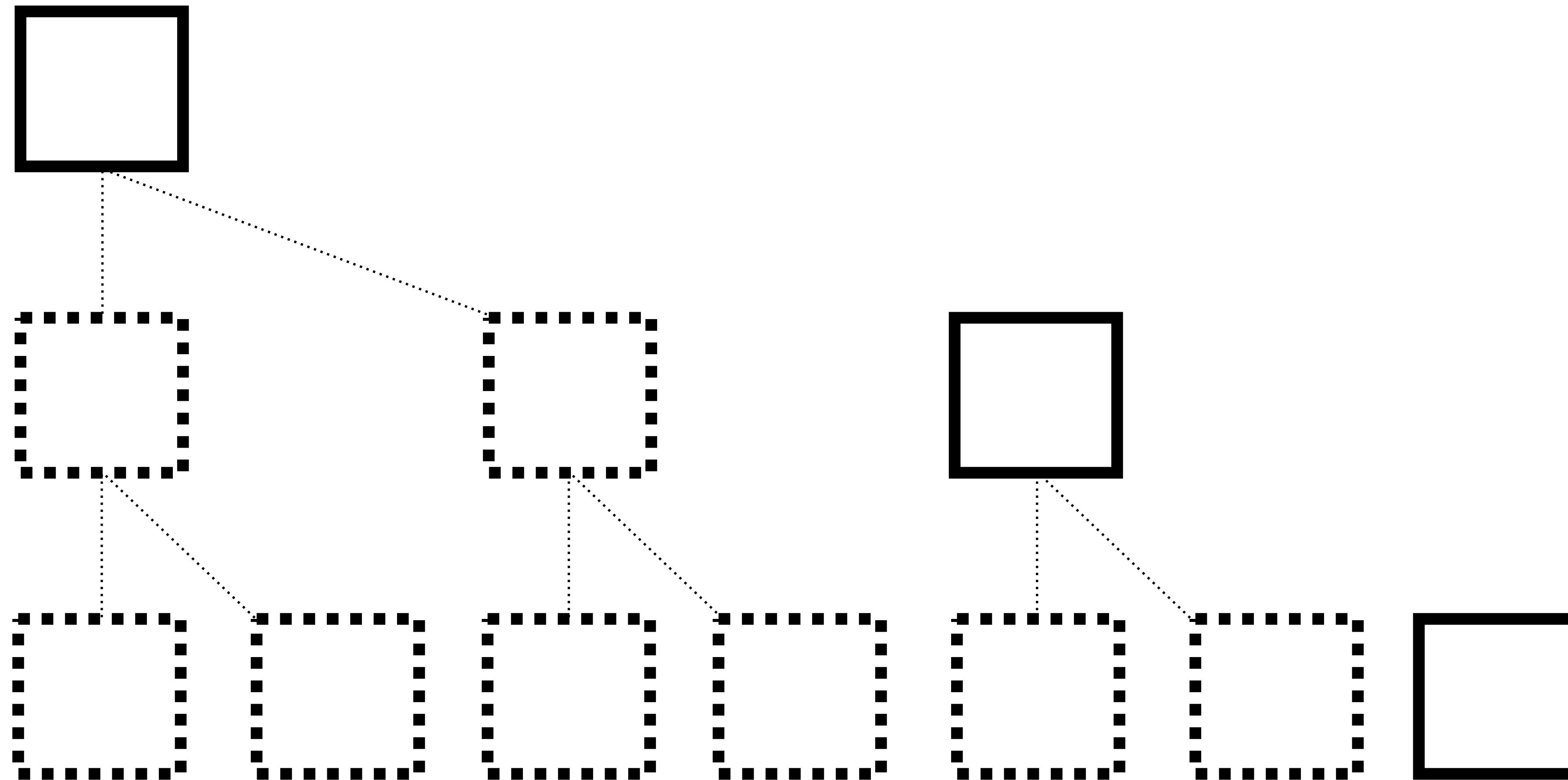
# 6 UTXOs

## Throw away



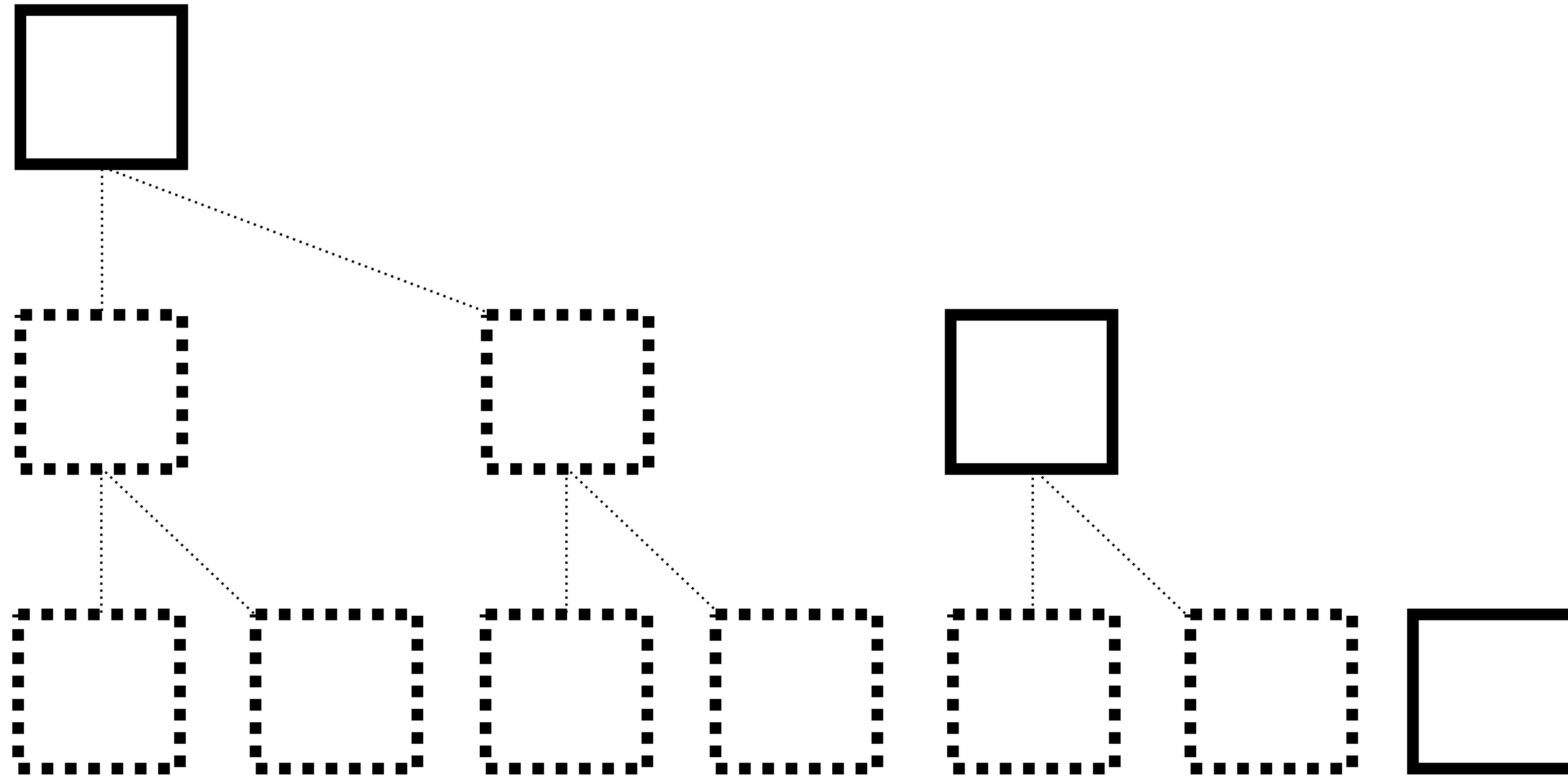
# 7 UTXOs

Another one



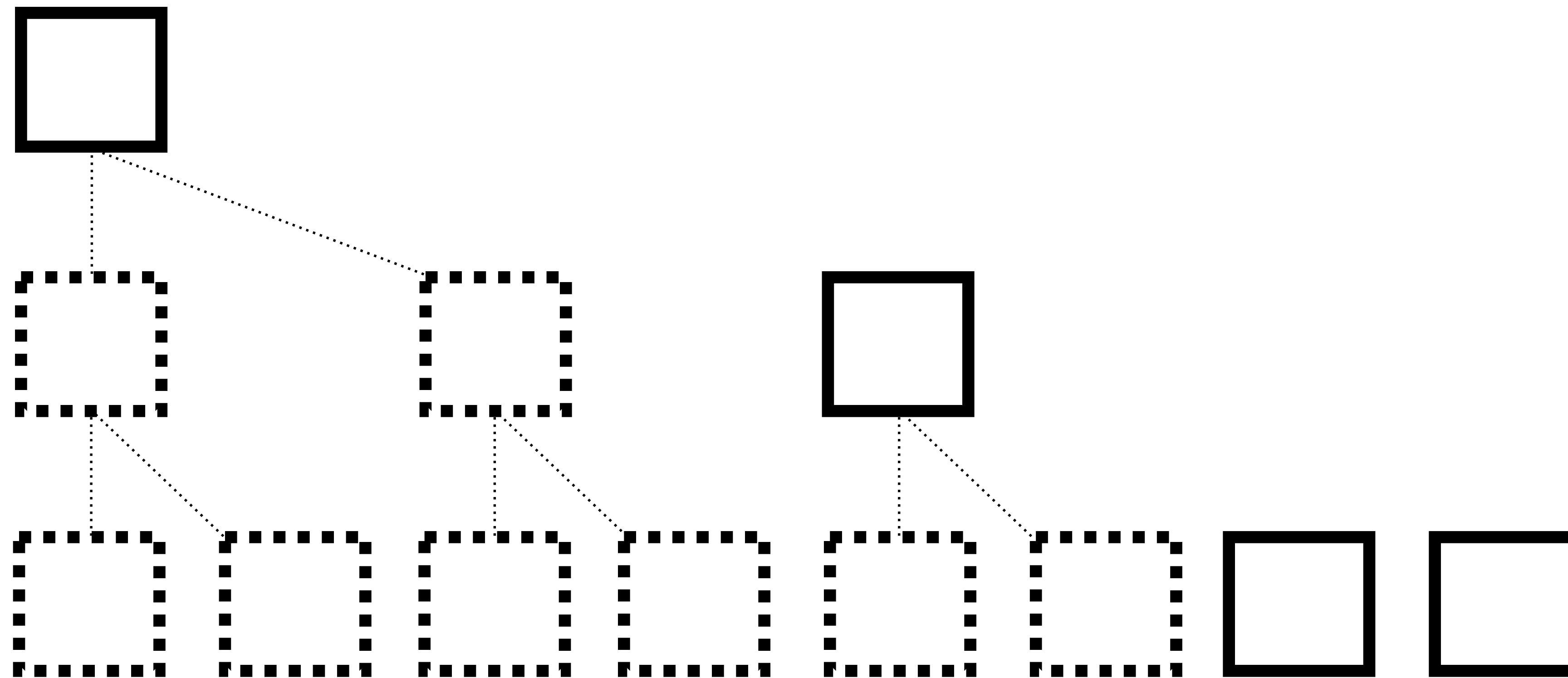
# 7 UTXOs

**Nothing to hash with so it becomes a root**



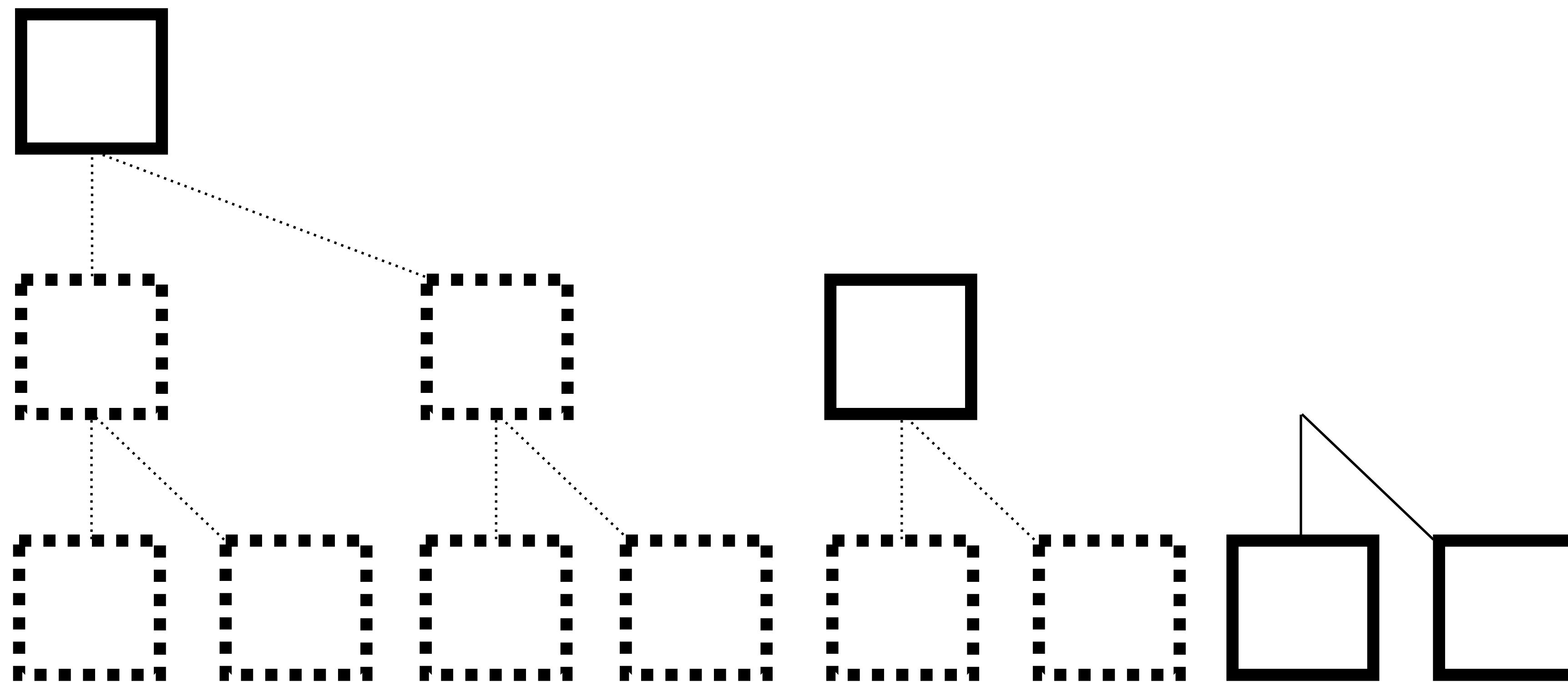
# 8 UTXOs

Another one



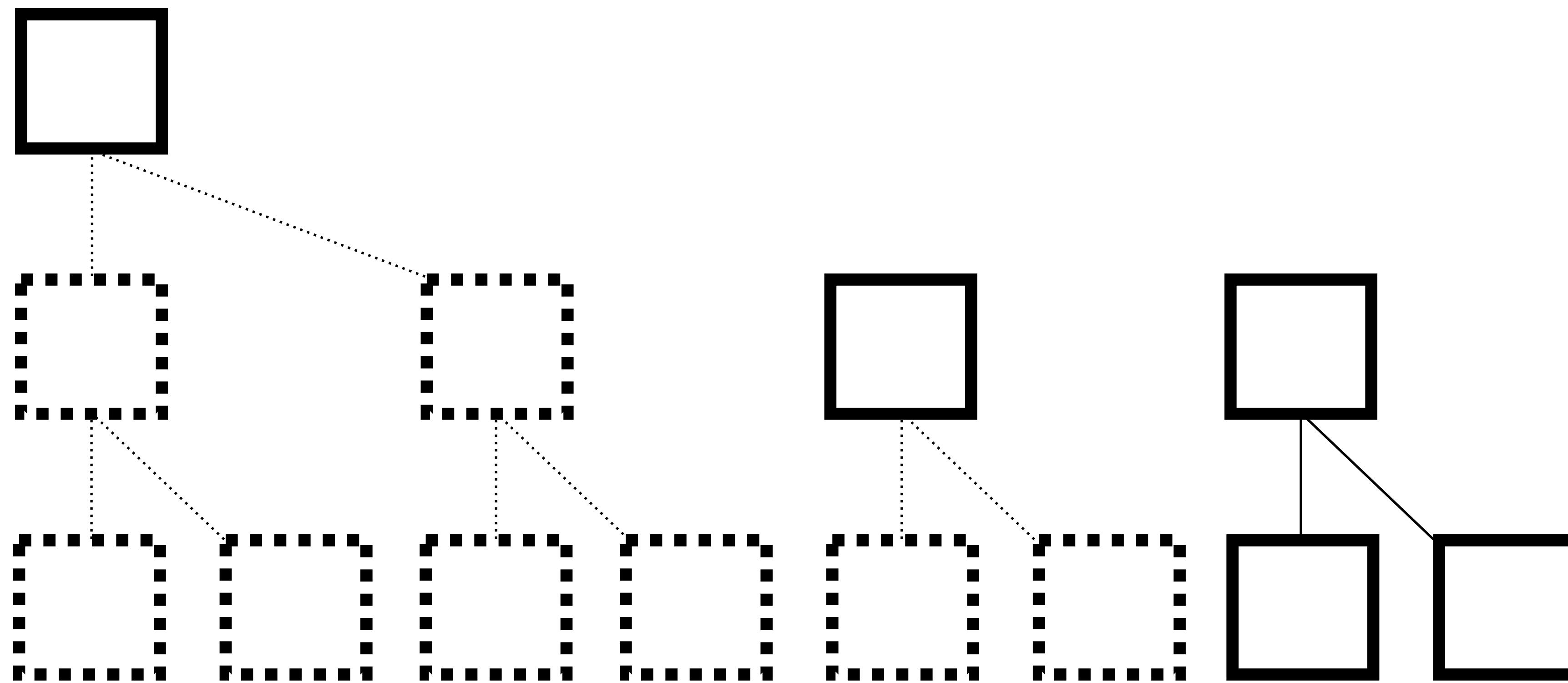
# 8 UTXOs

## Concatenate



# 8 UTXOs

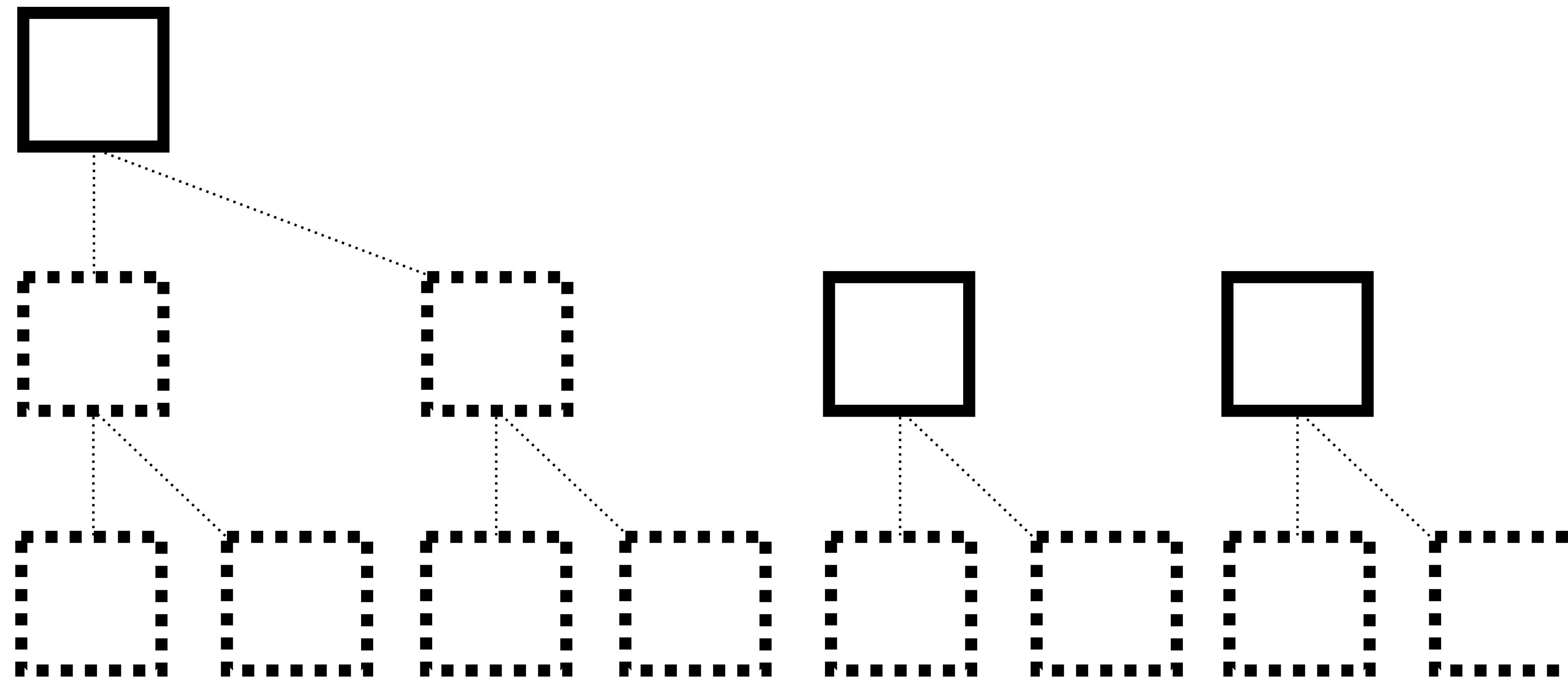
## Hash



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

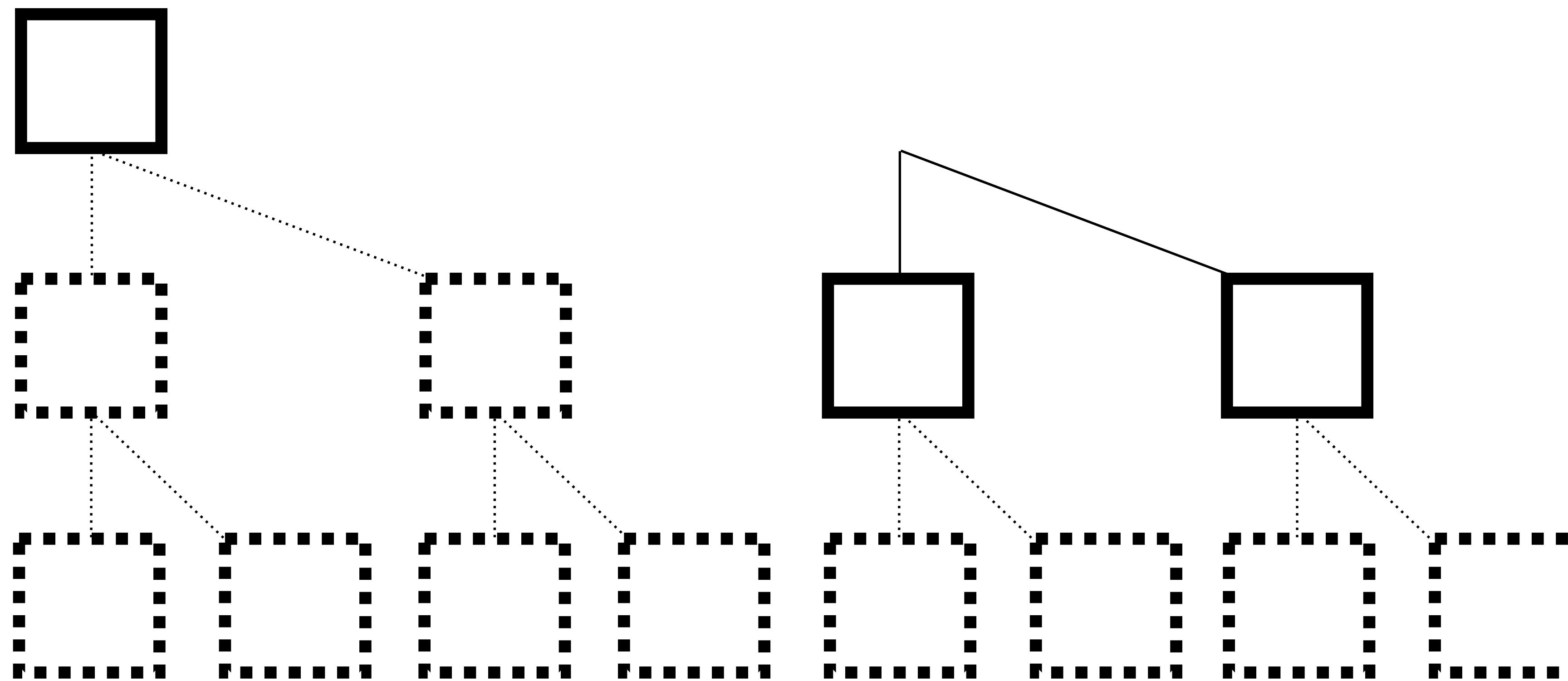
# 8 UTXOs

## Throw away



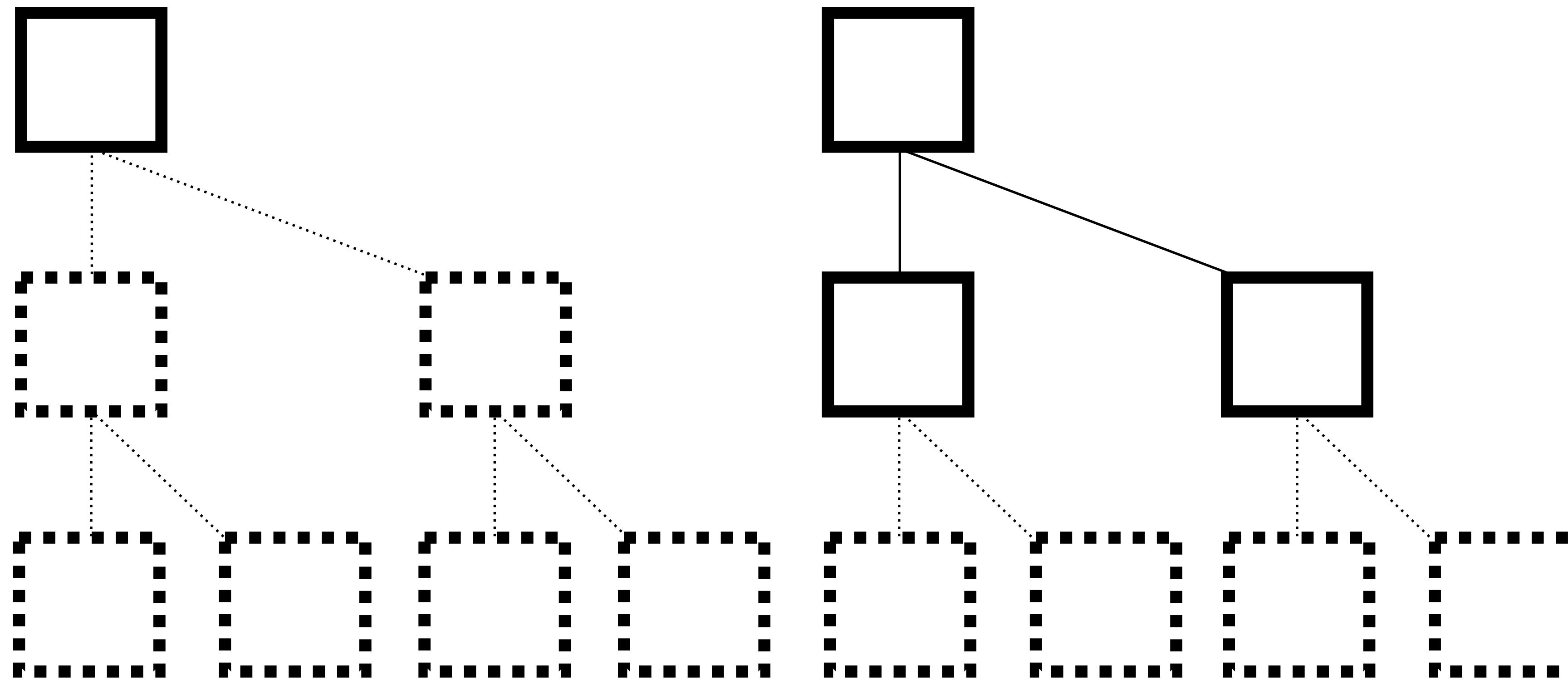
# 8 UTXOs

## Concatenate



# 8 UTXOs

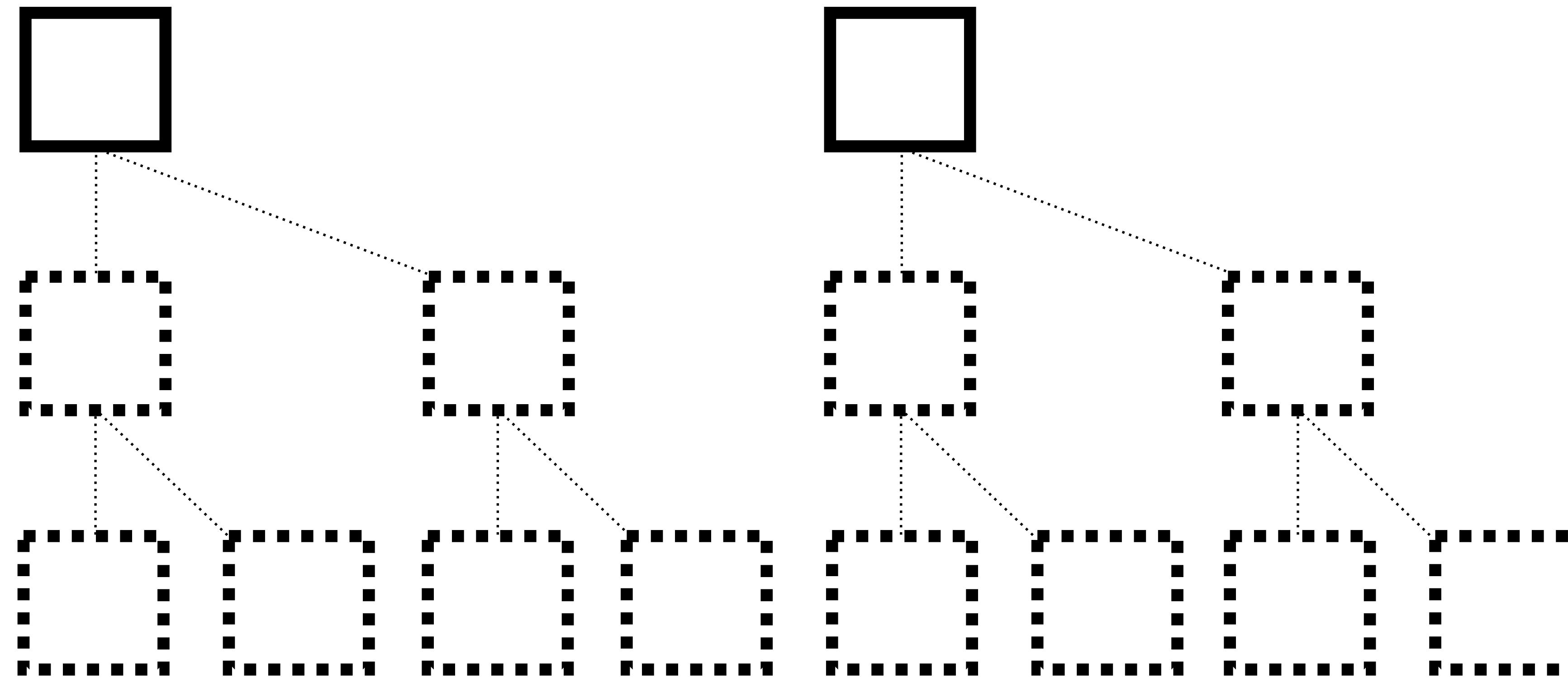
## Hash



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

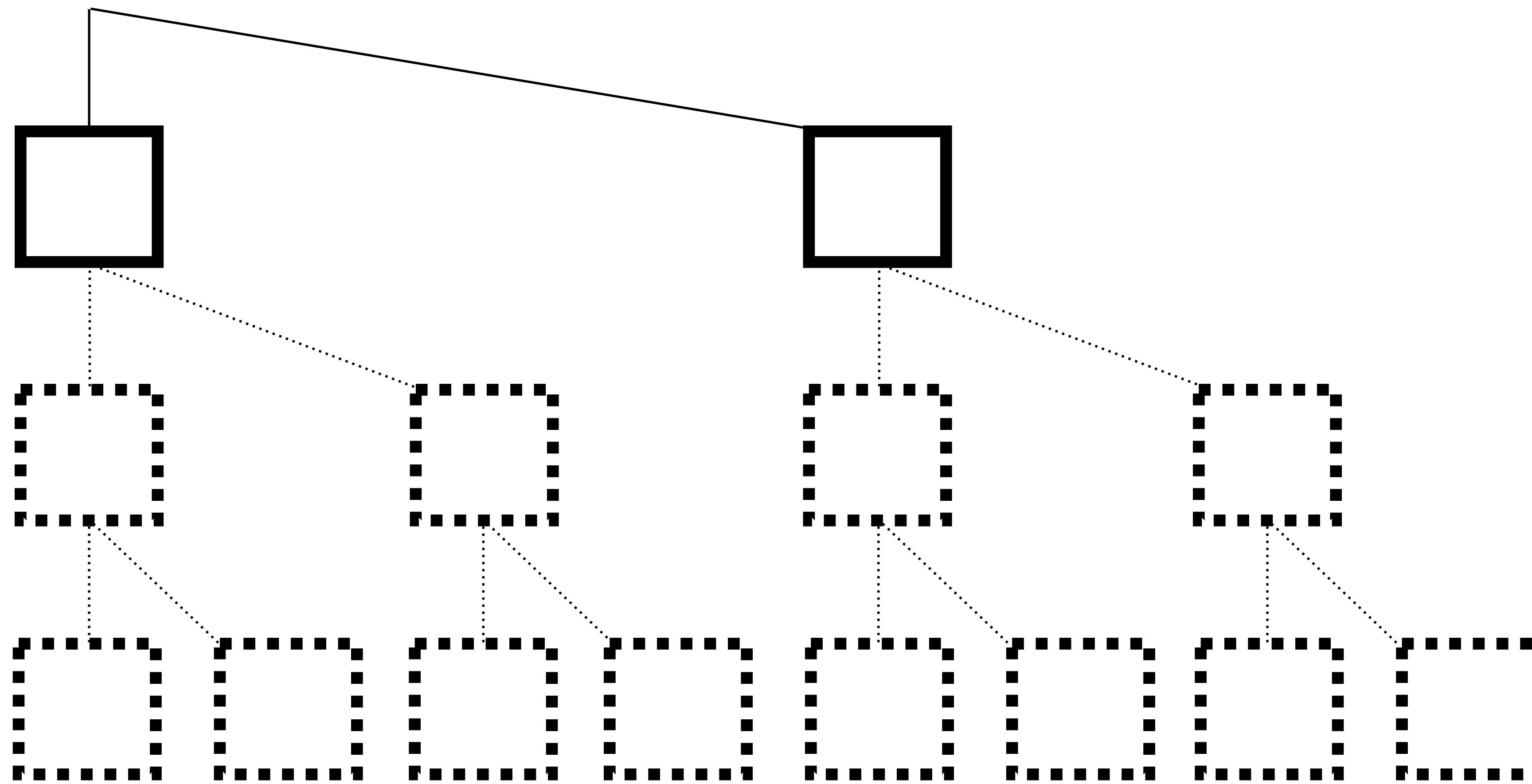
# 8 UTXOs

## Throw away

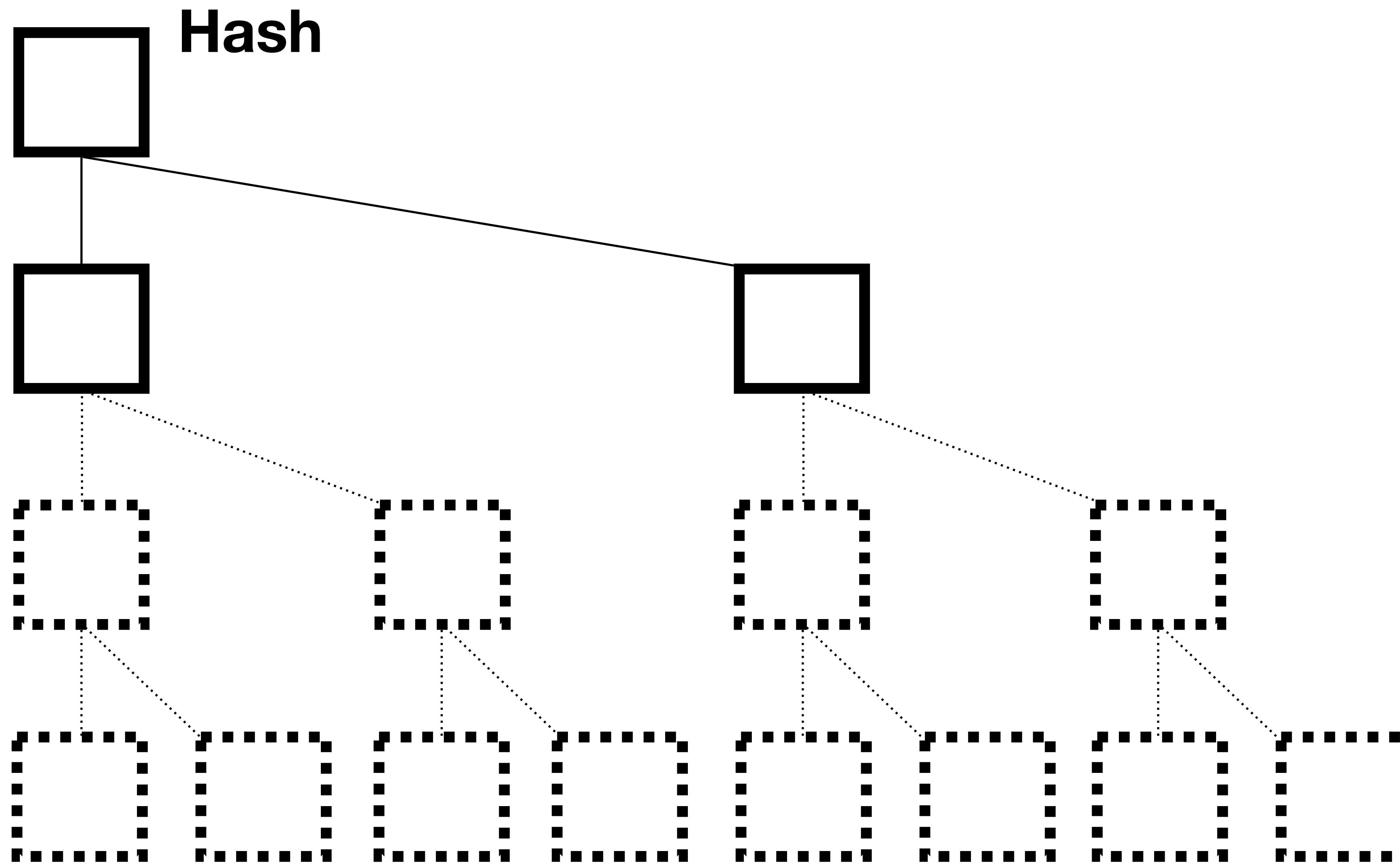


# 8 UTXOs

## Concatenate

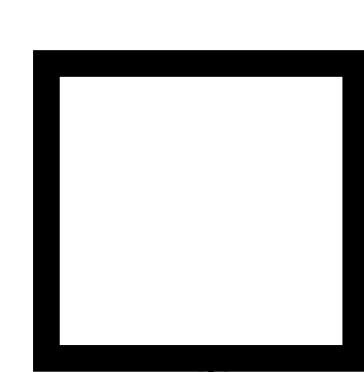


# 8 UTXOs

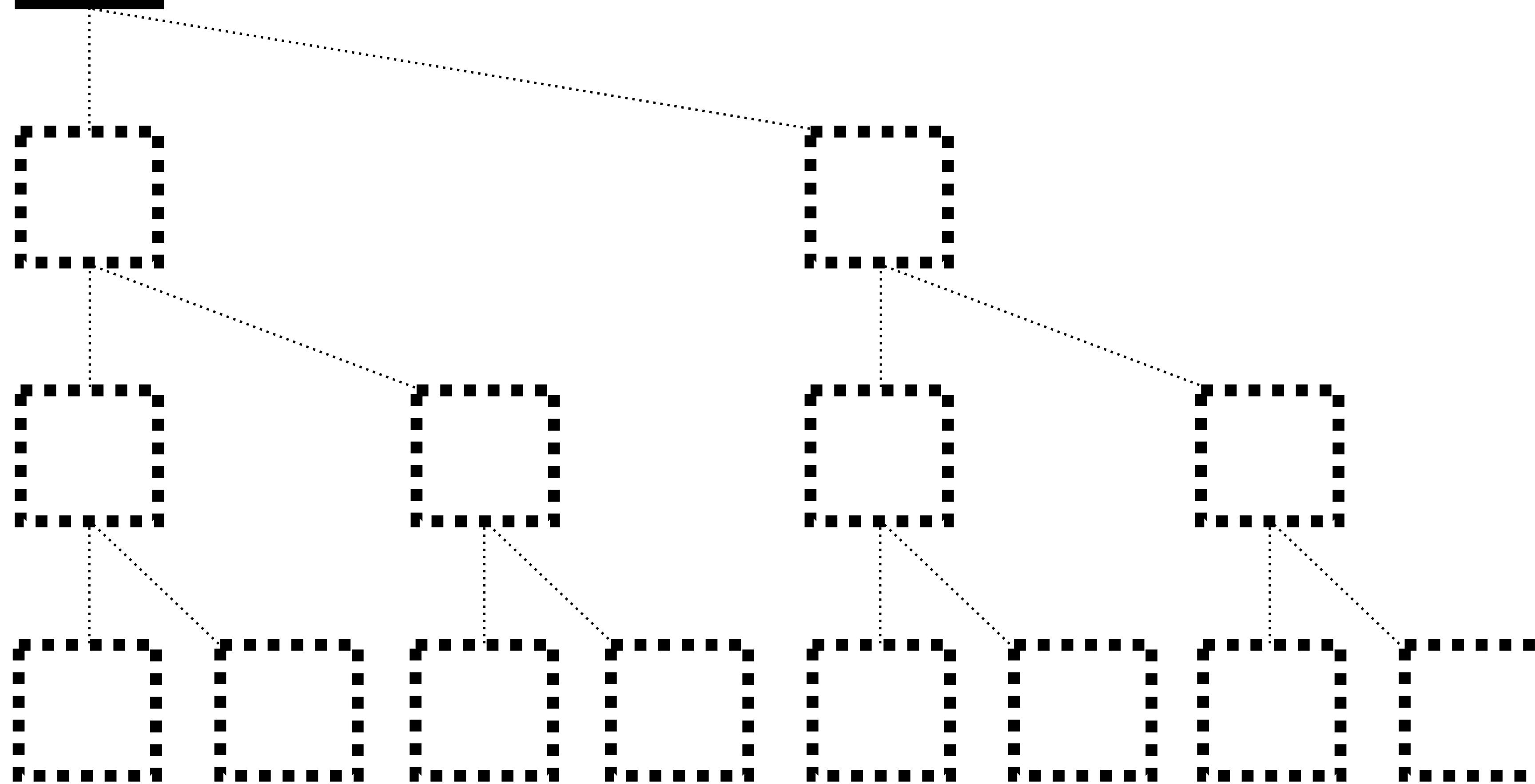


<https://gist.github.com/kcalvin/alvin/a790d524832e1b7f96a70c642315fffc>

# 8 UTXOs



**Throw away**



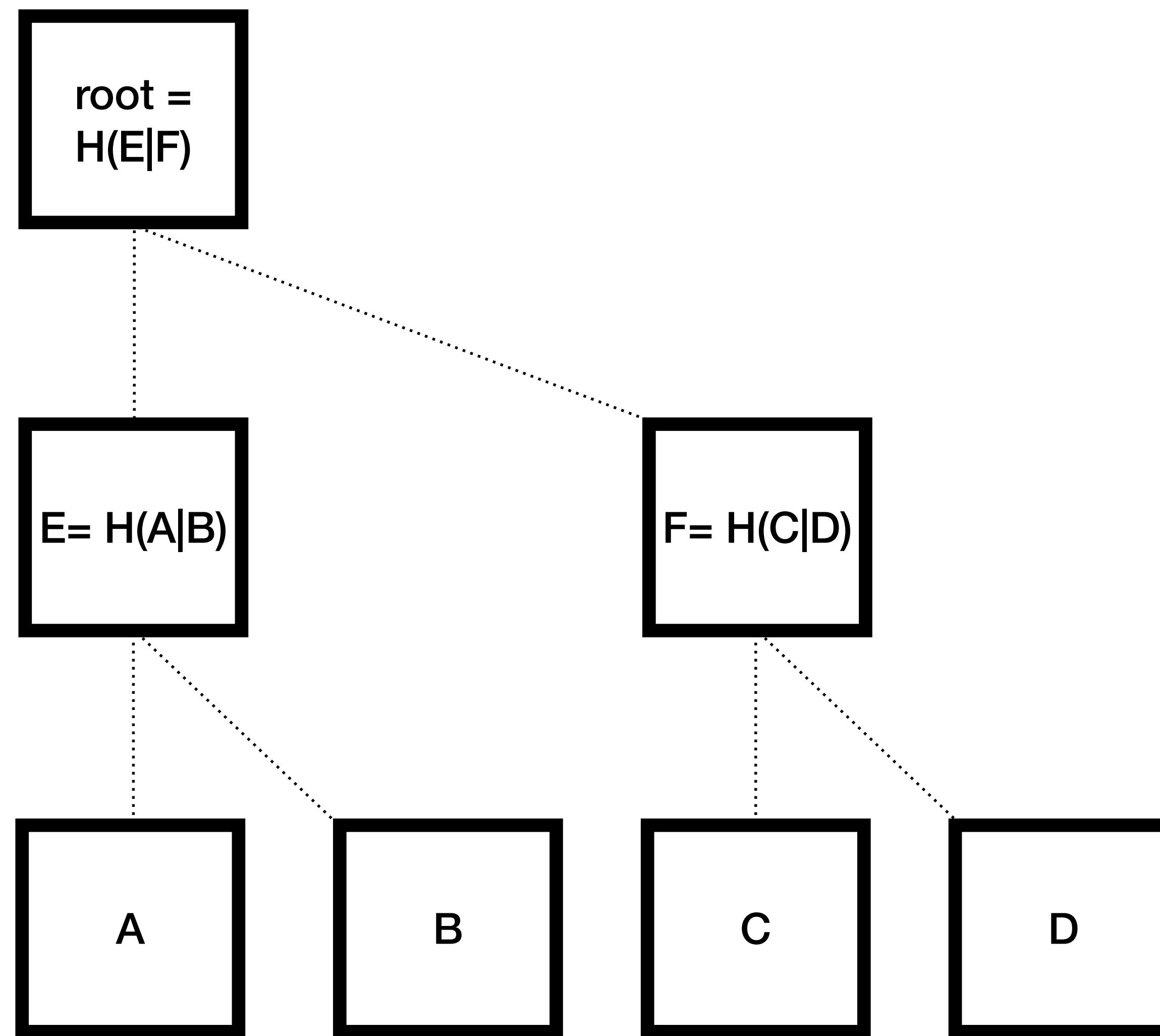
# **Algorithms**

**Consensus critical stuff**

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

**Deletion happens with verifying  
the existence**

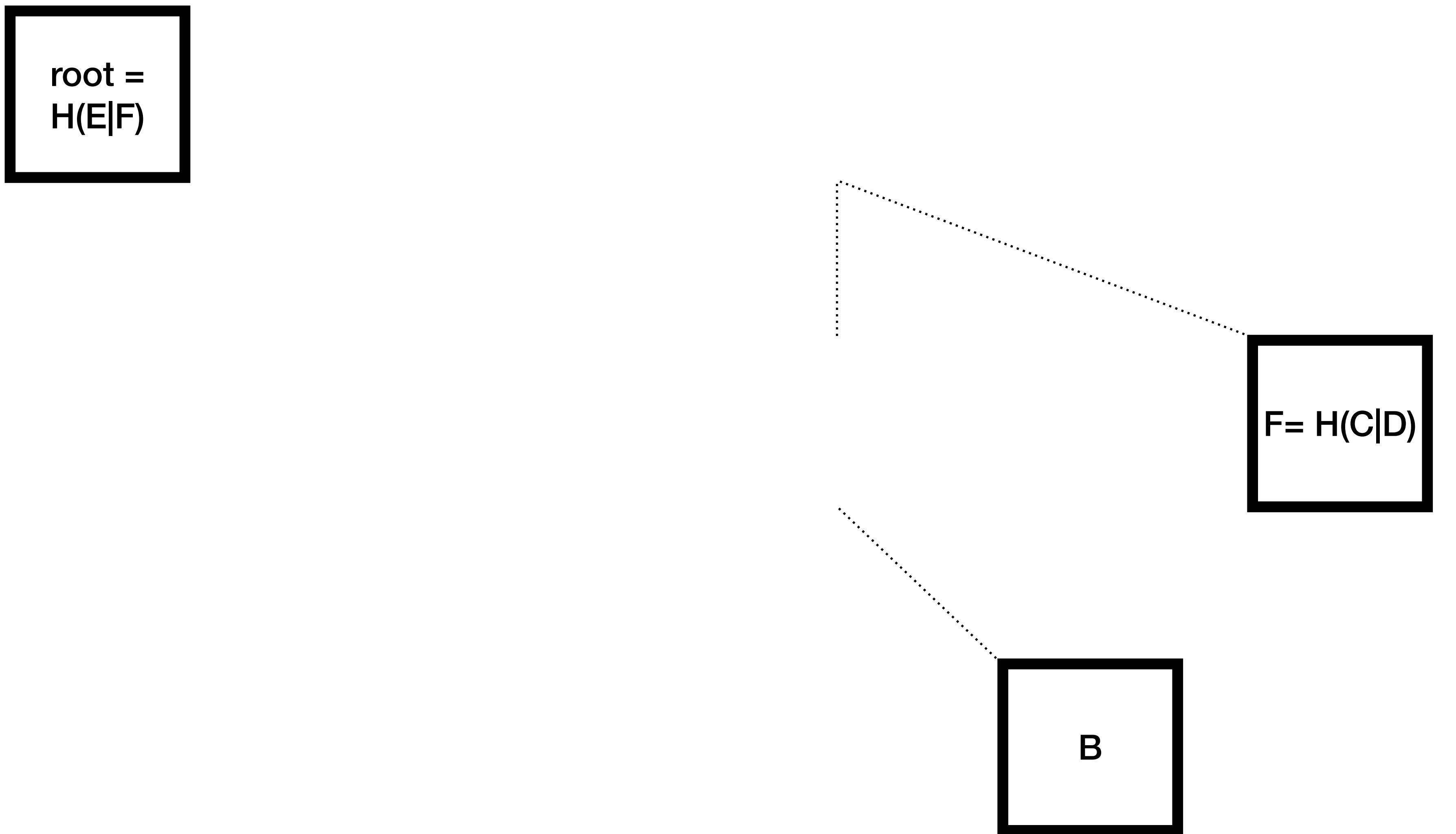
# Quick merkle tree prove review



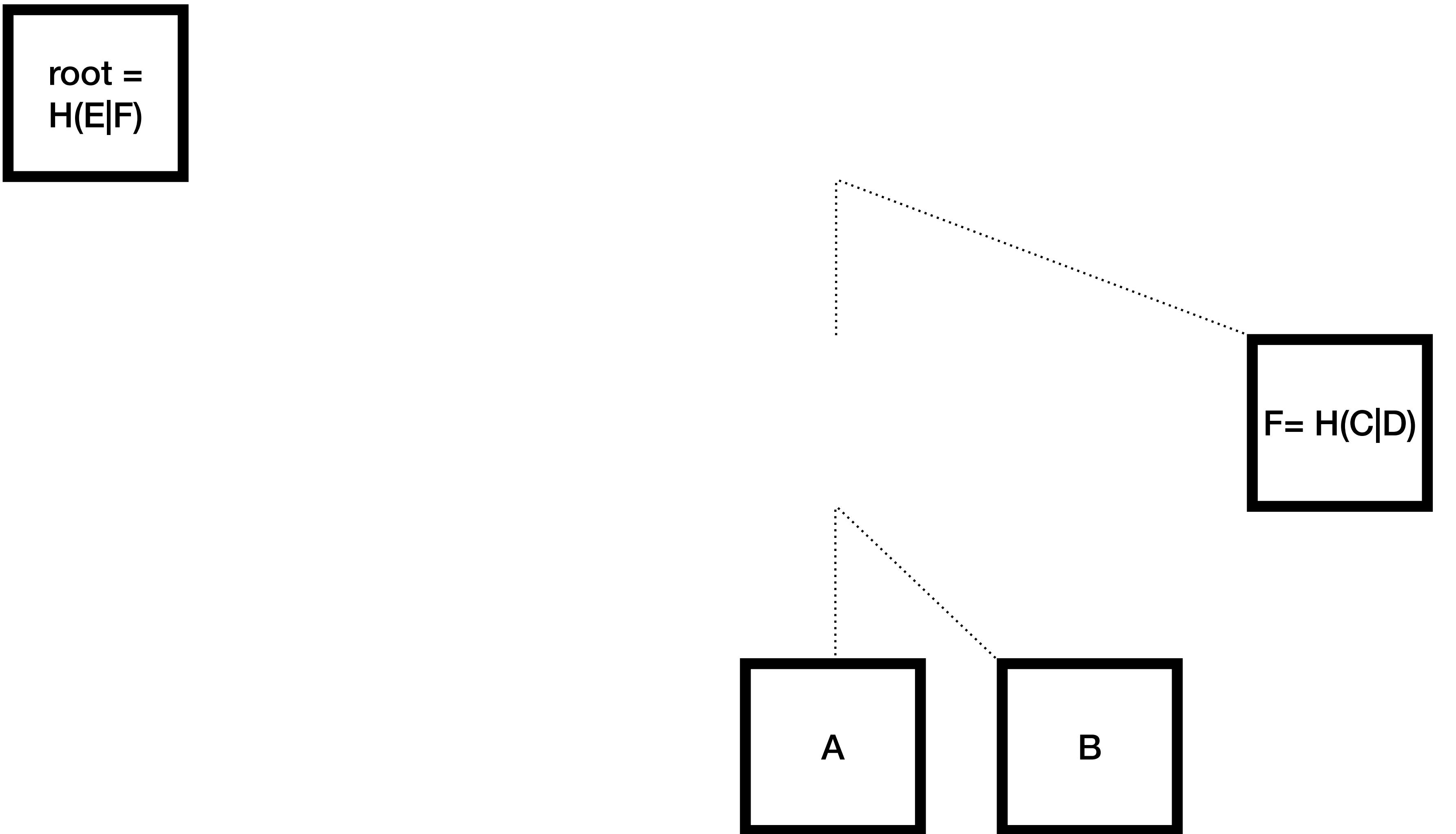
# Proving A

root =  
 $H(E|F)$

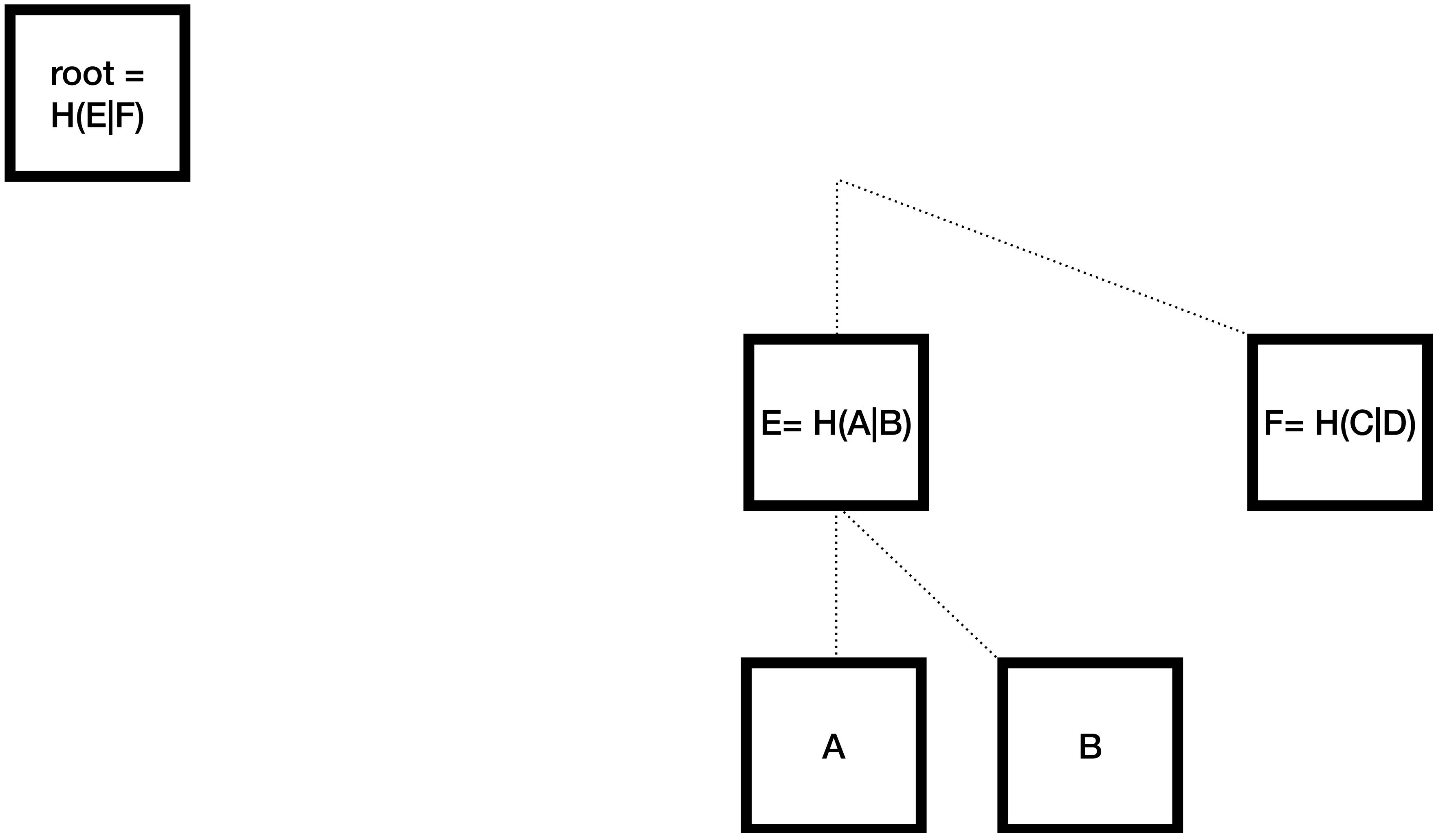
# Receive proof: B, F



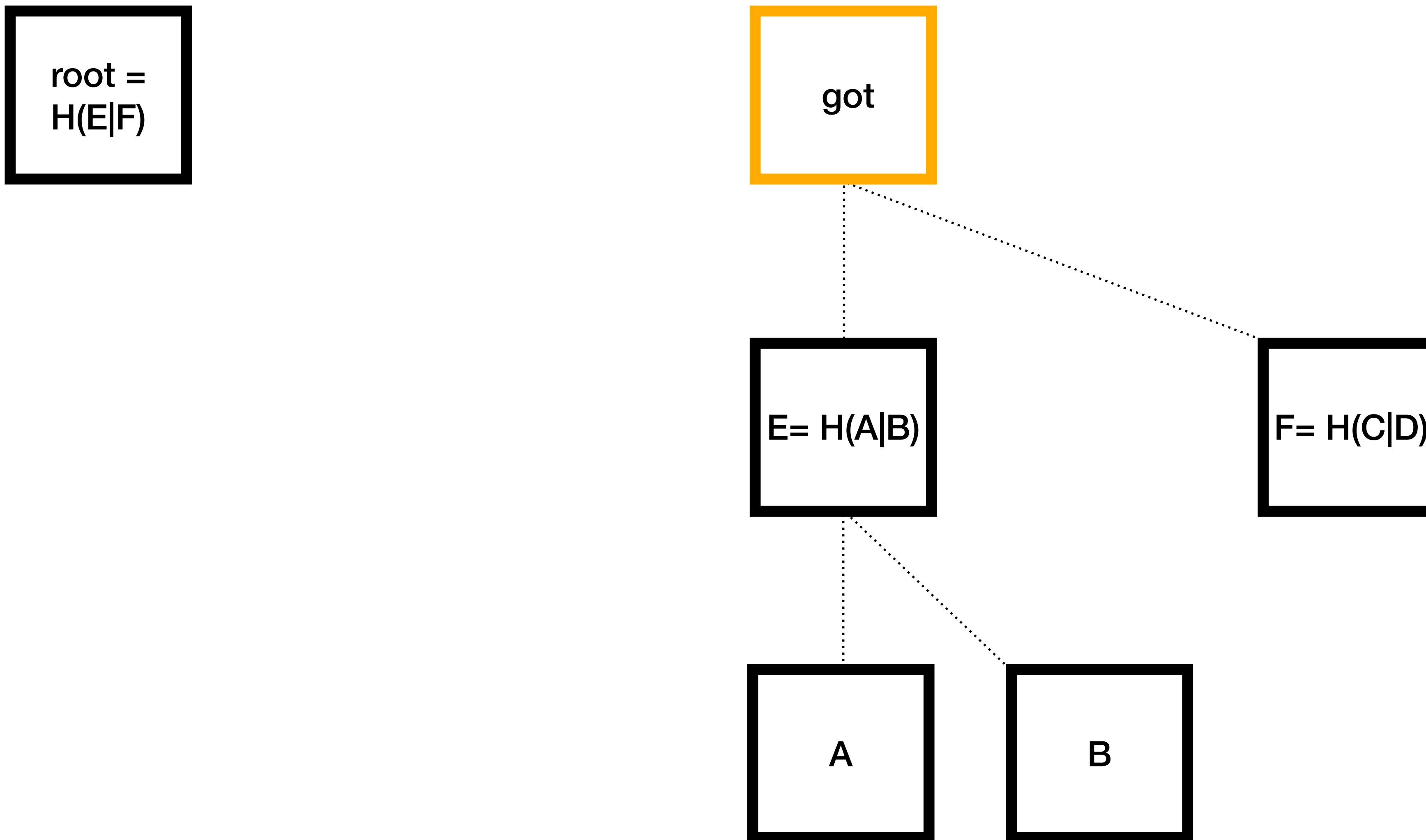
# Fetch A



# Calculate E



# Calculate root



# Compare roots

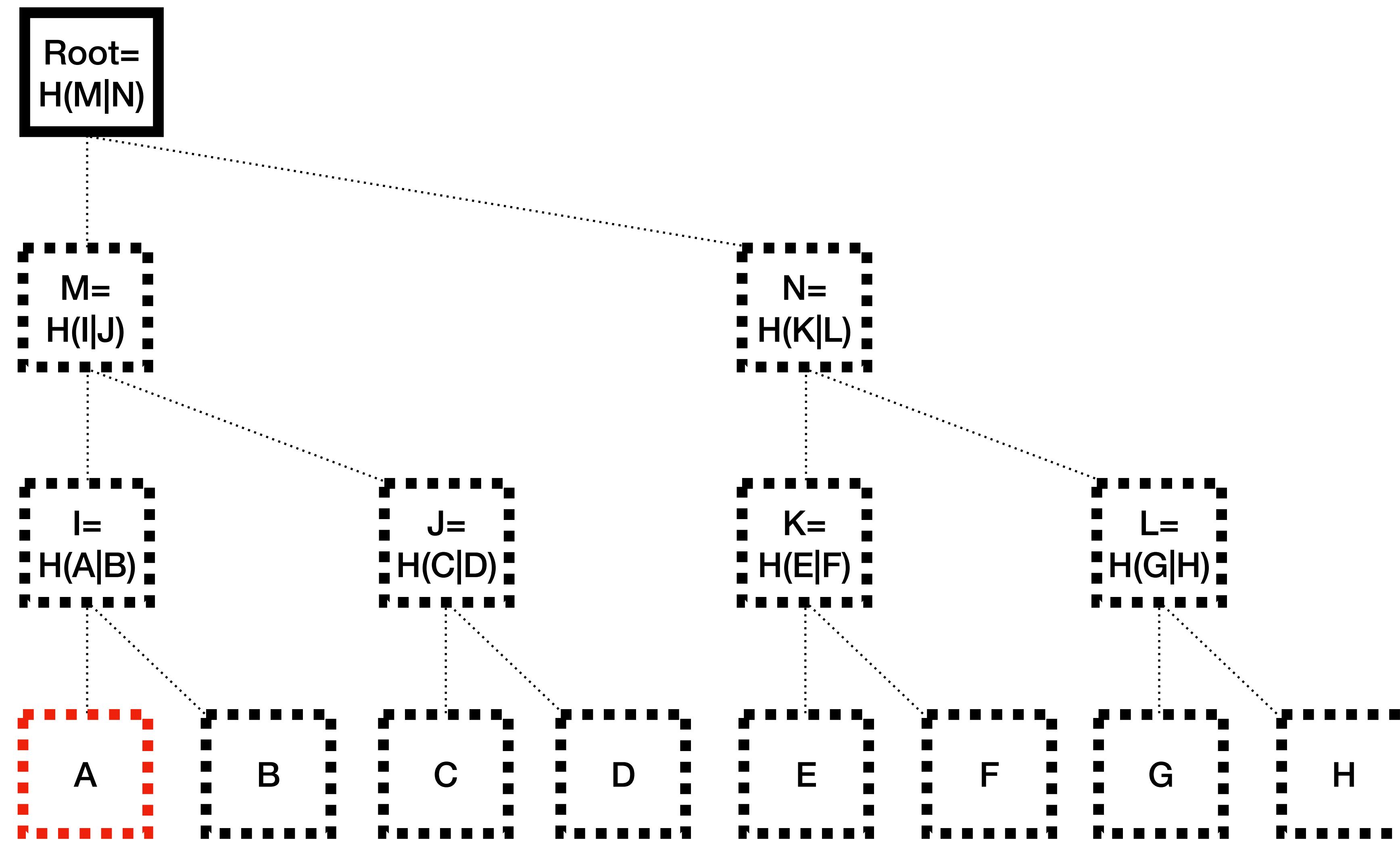
Proof is valid if root is the same

root= $H(E|F)$

==

Got Root

# Deleting A

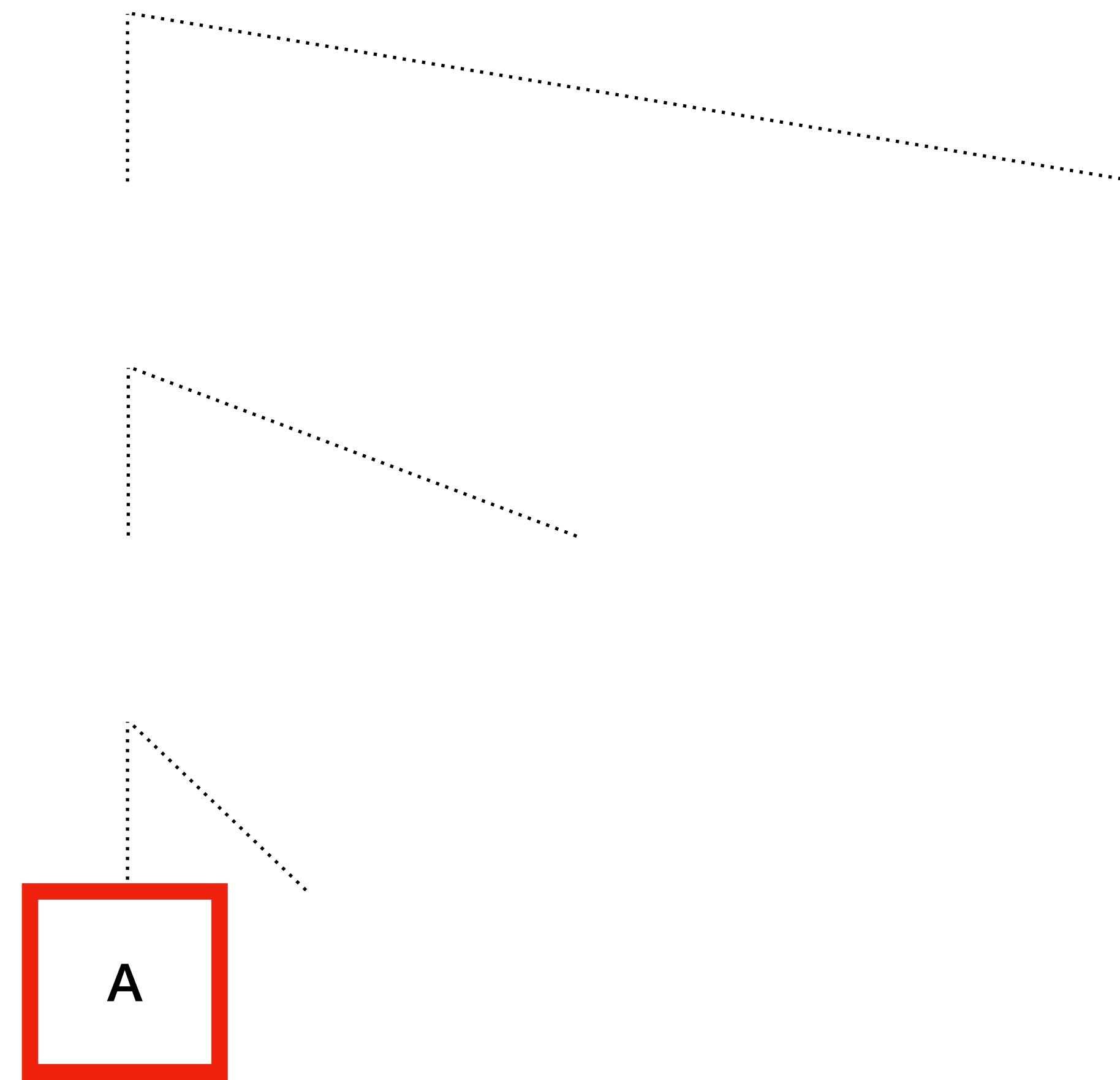


# The only requirement is the roots

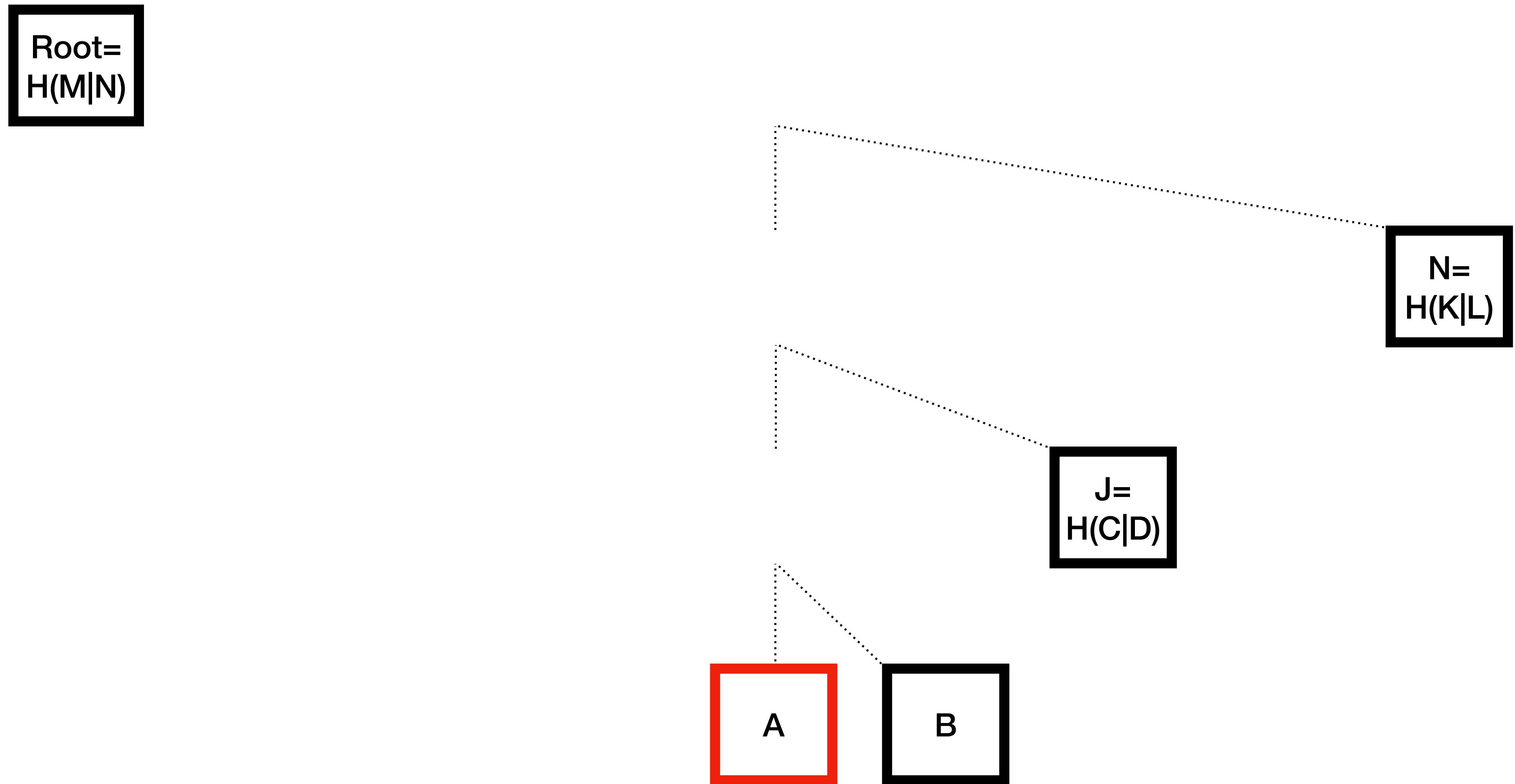
Root=  
 $H(M|N)$

# Calculate A from the Bitcoin Block

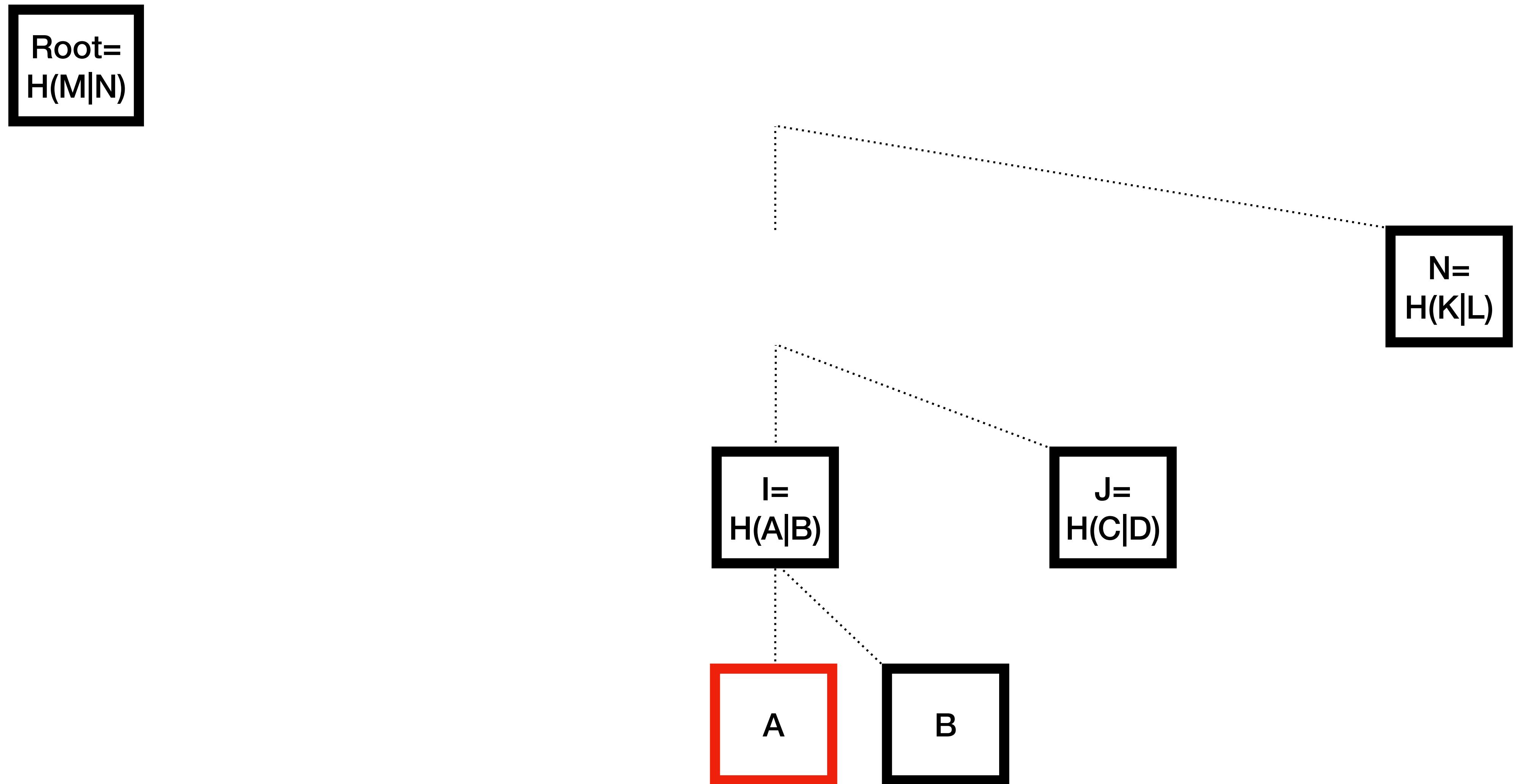
Root=  
 $H(M|N)$



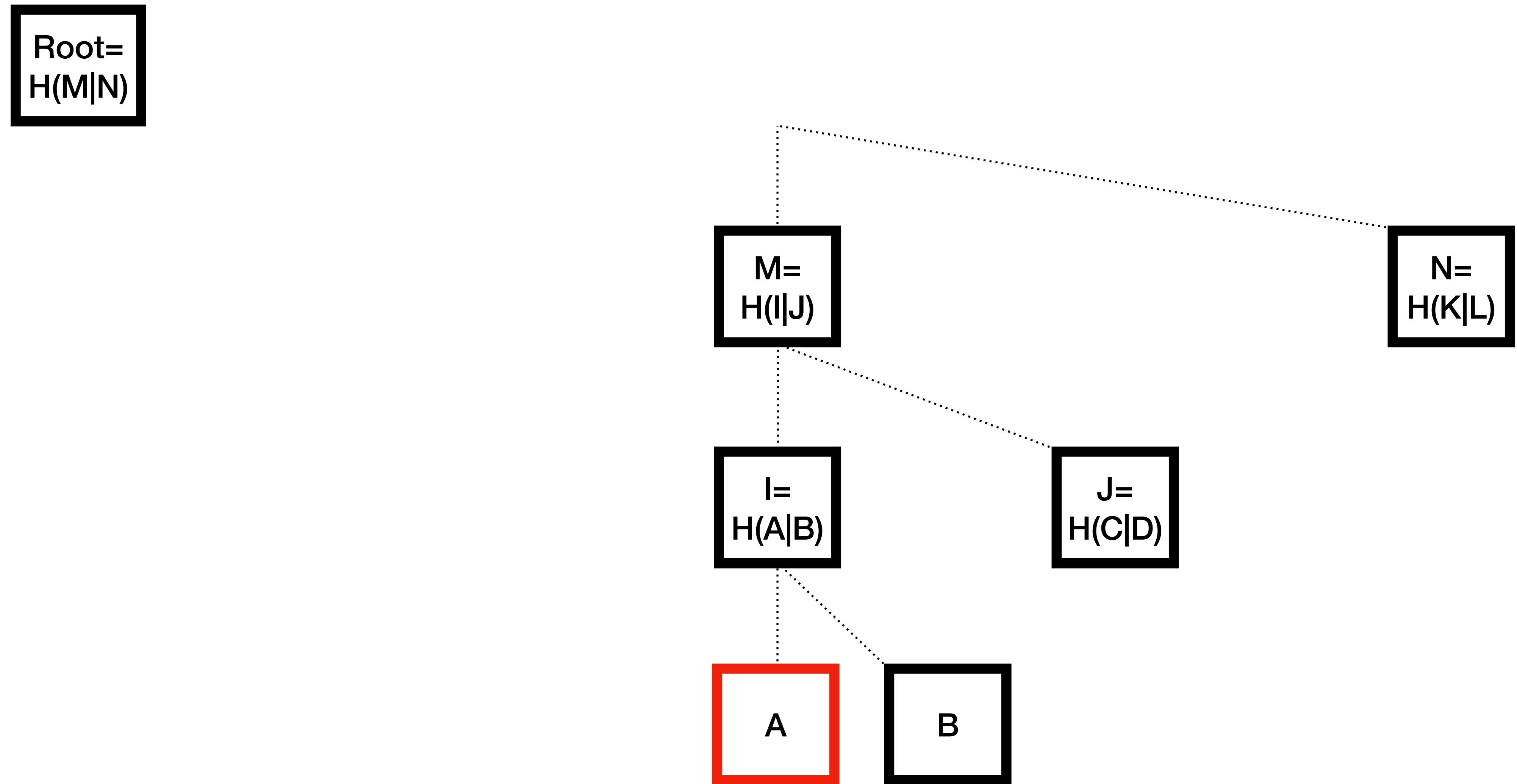
# Receive the proof: B, J, N



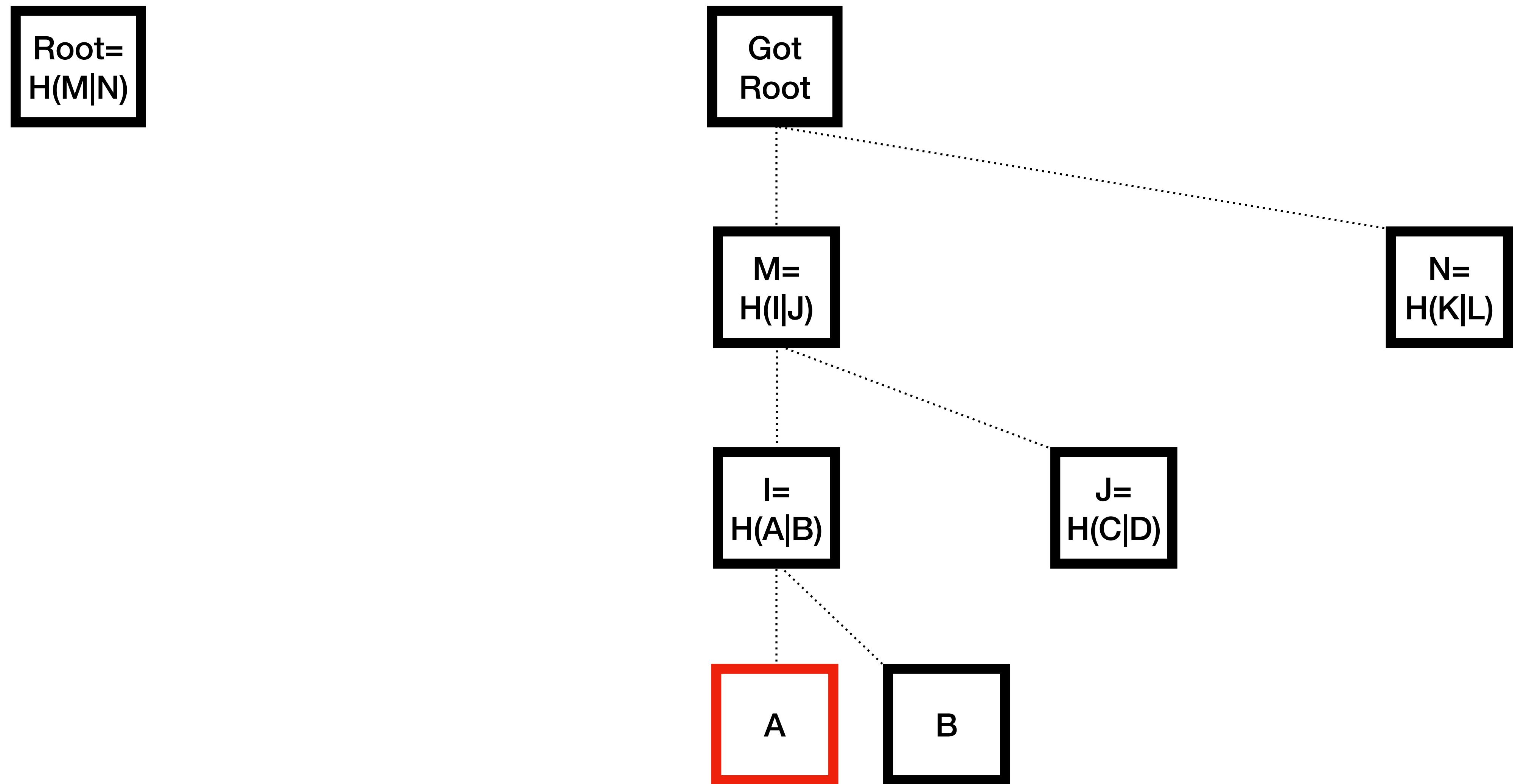
# Calculate the hash for I



# Calculate the hash for M



# Calculate Root



# Compare roots

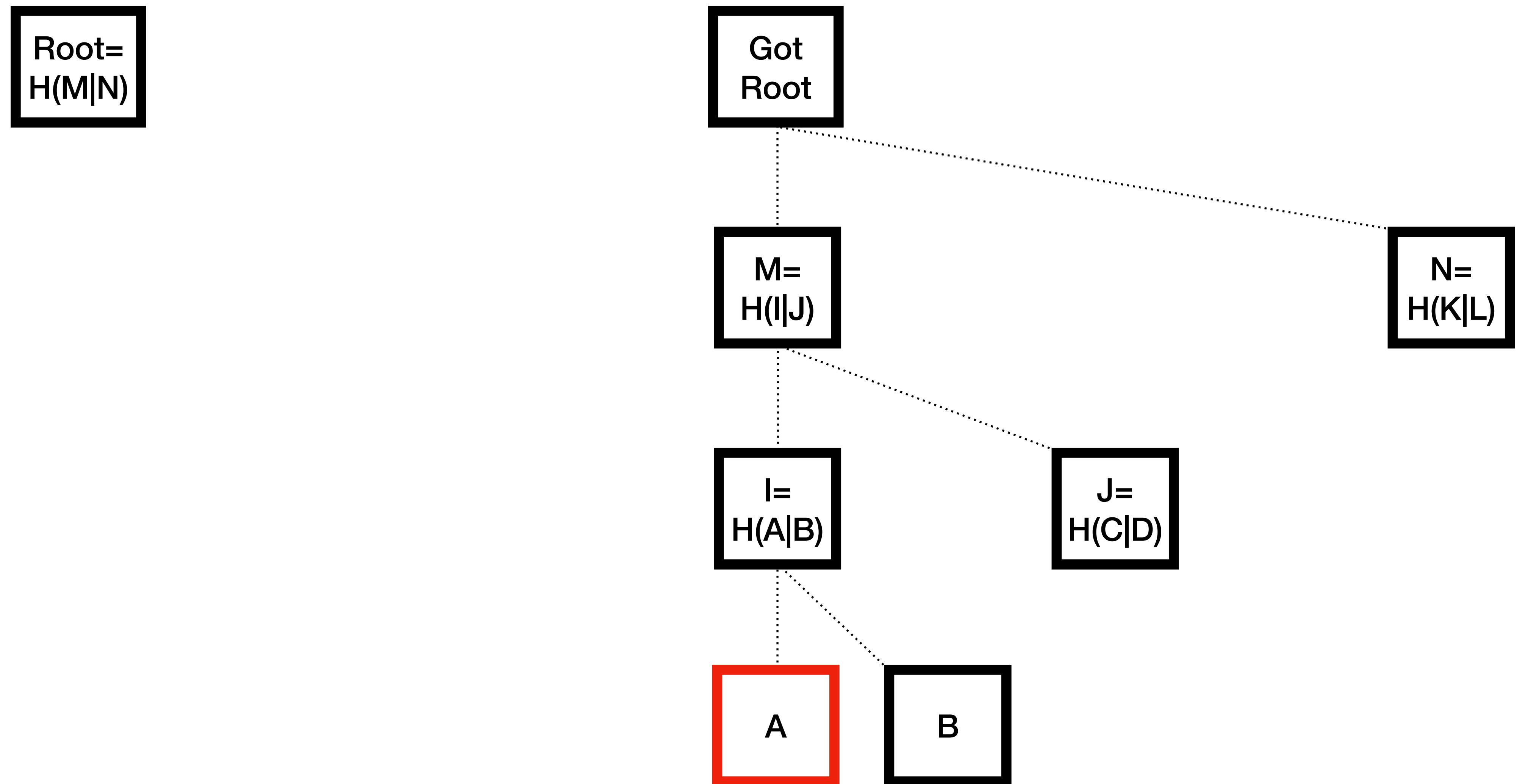
Continue if the roots are equal. Ban peer if not equal

Root= $H(M|N)$

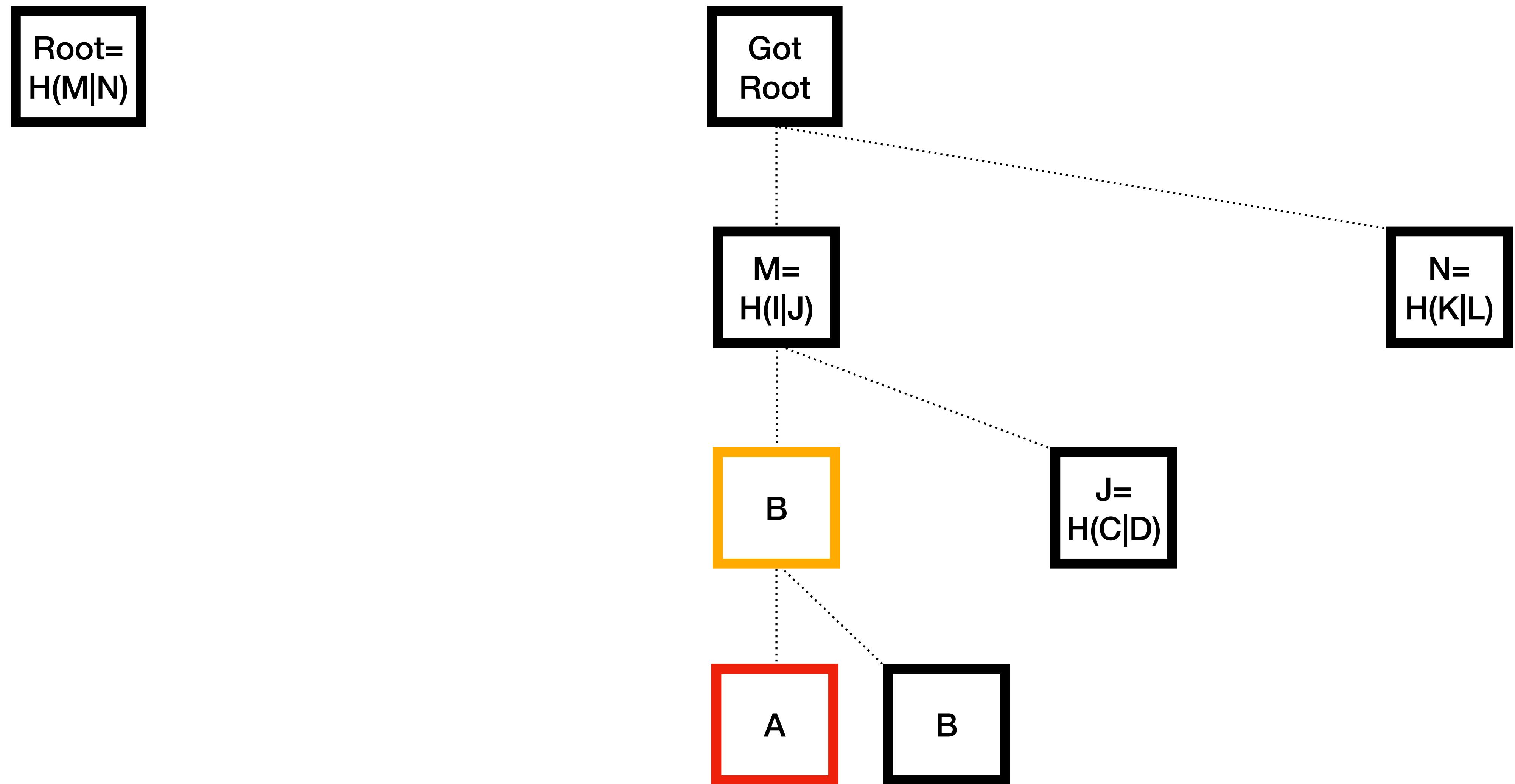
==

Got Root

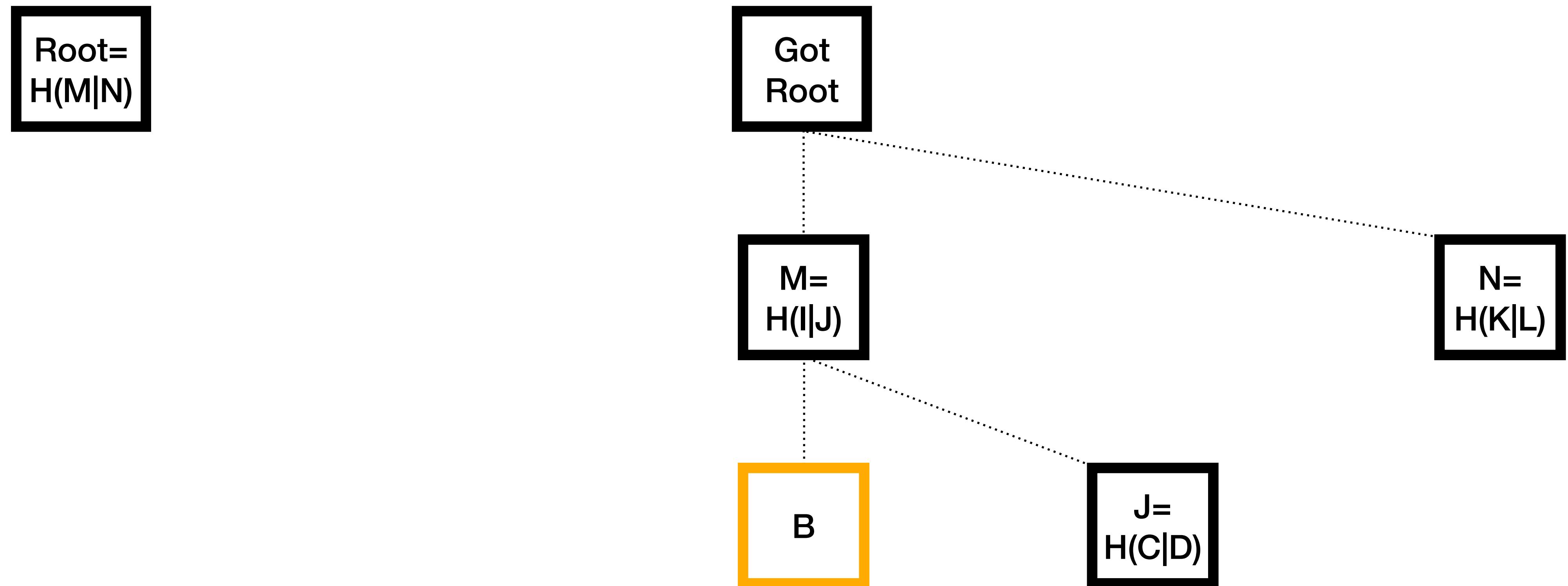
# Calculate root after deletion



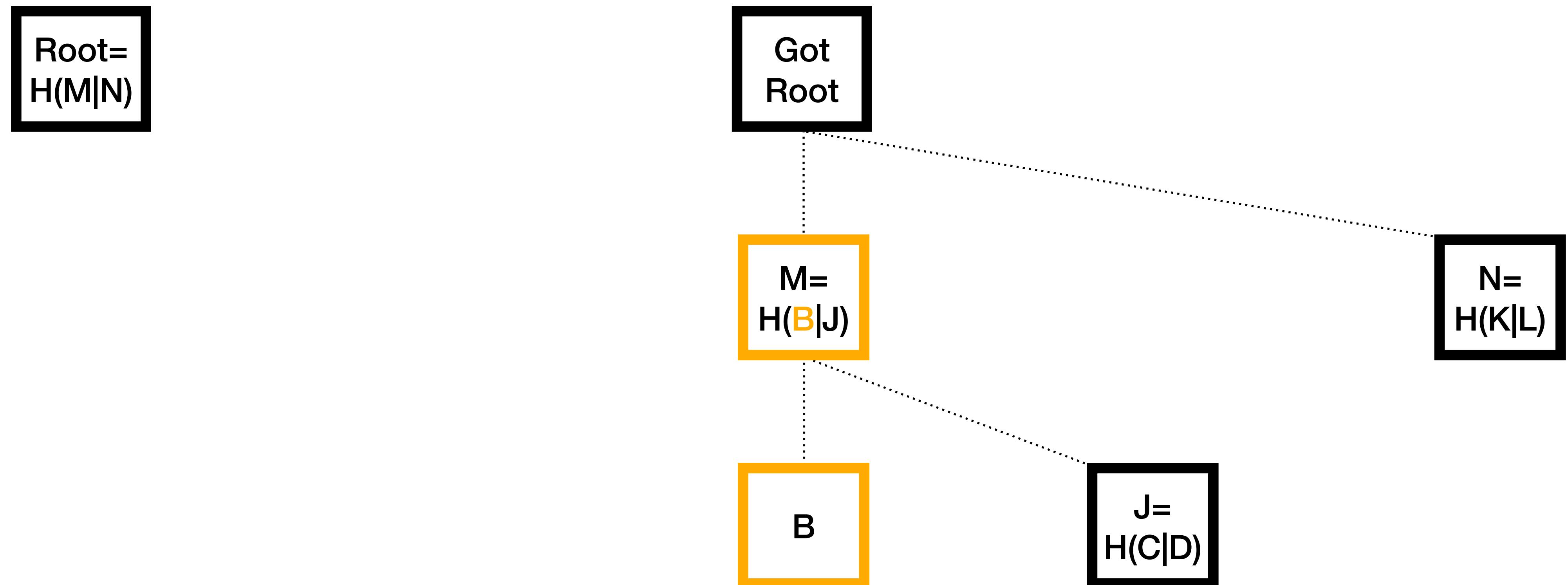
# Move up B to I



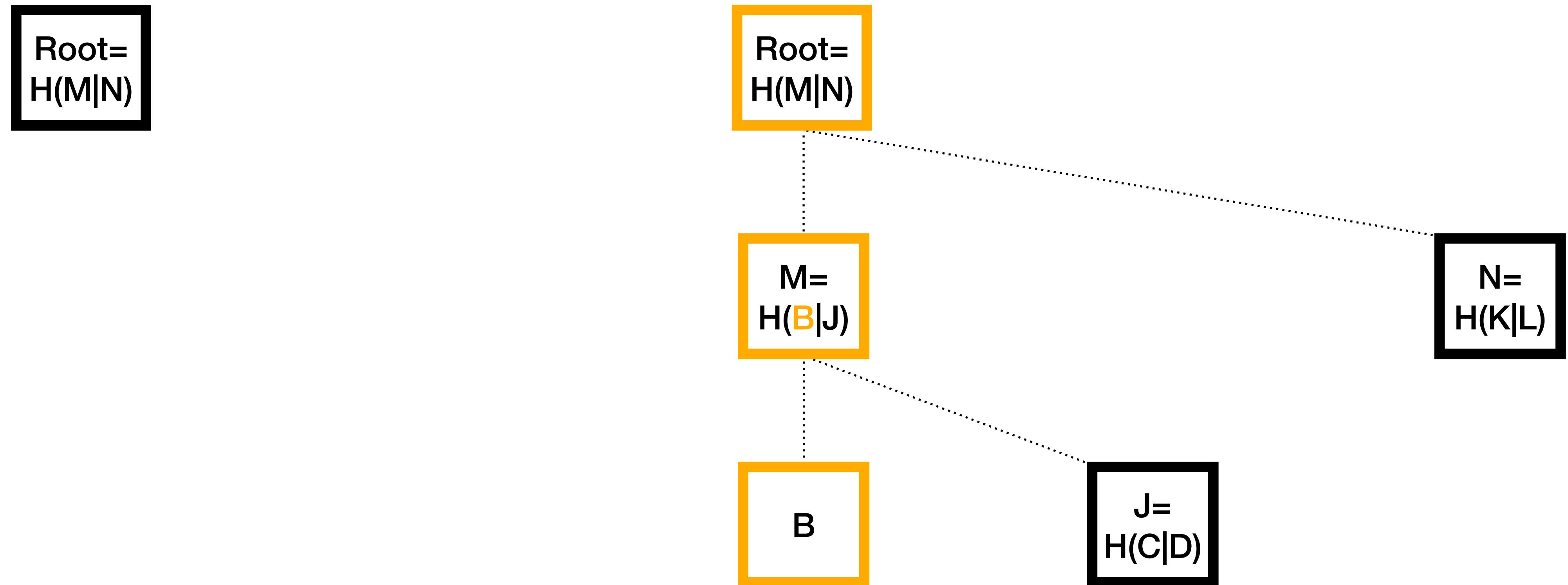
# Remove old nodes for A&B



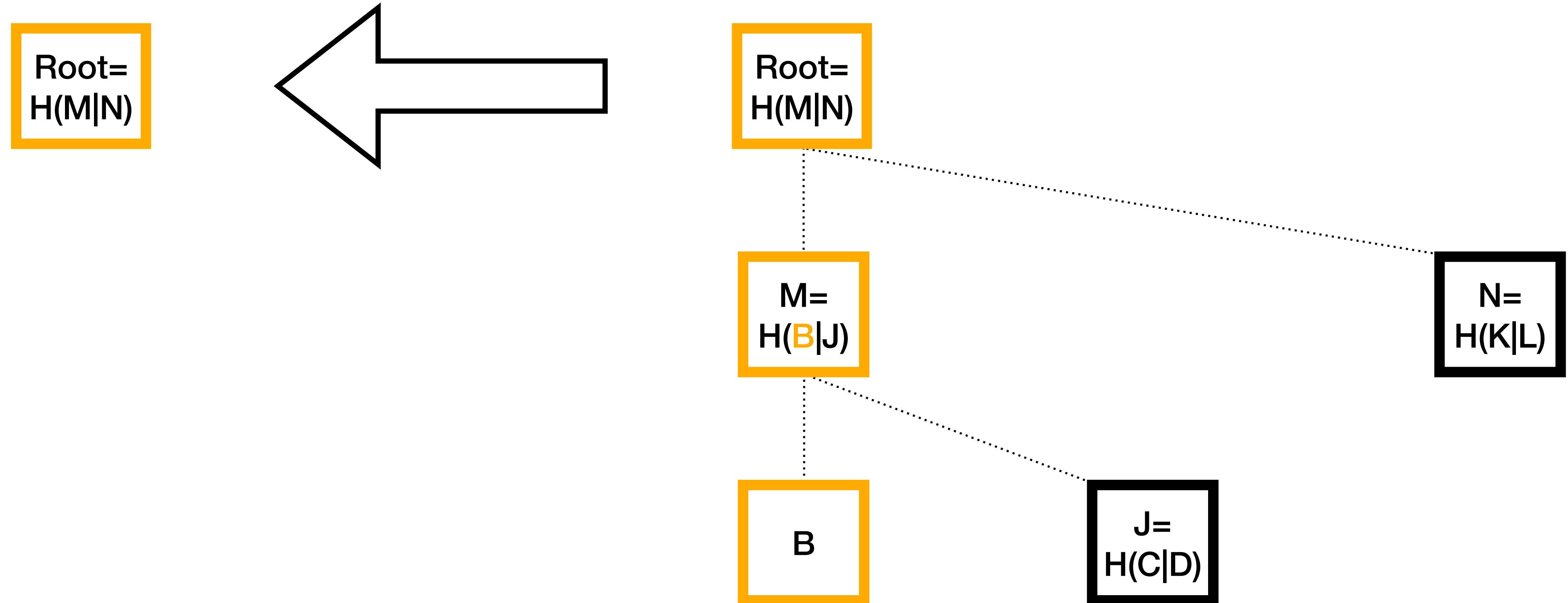
# Calculate $H(B|J)$



# Calculate new root



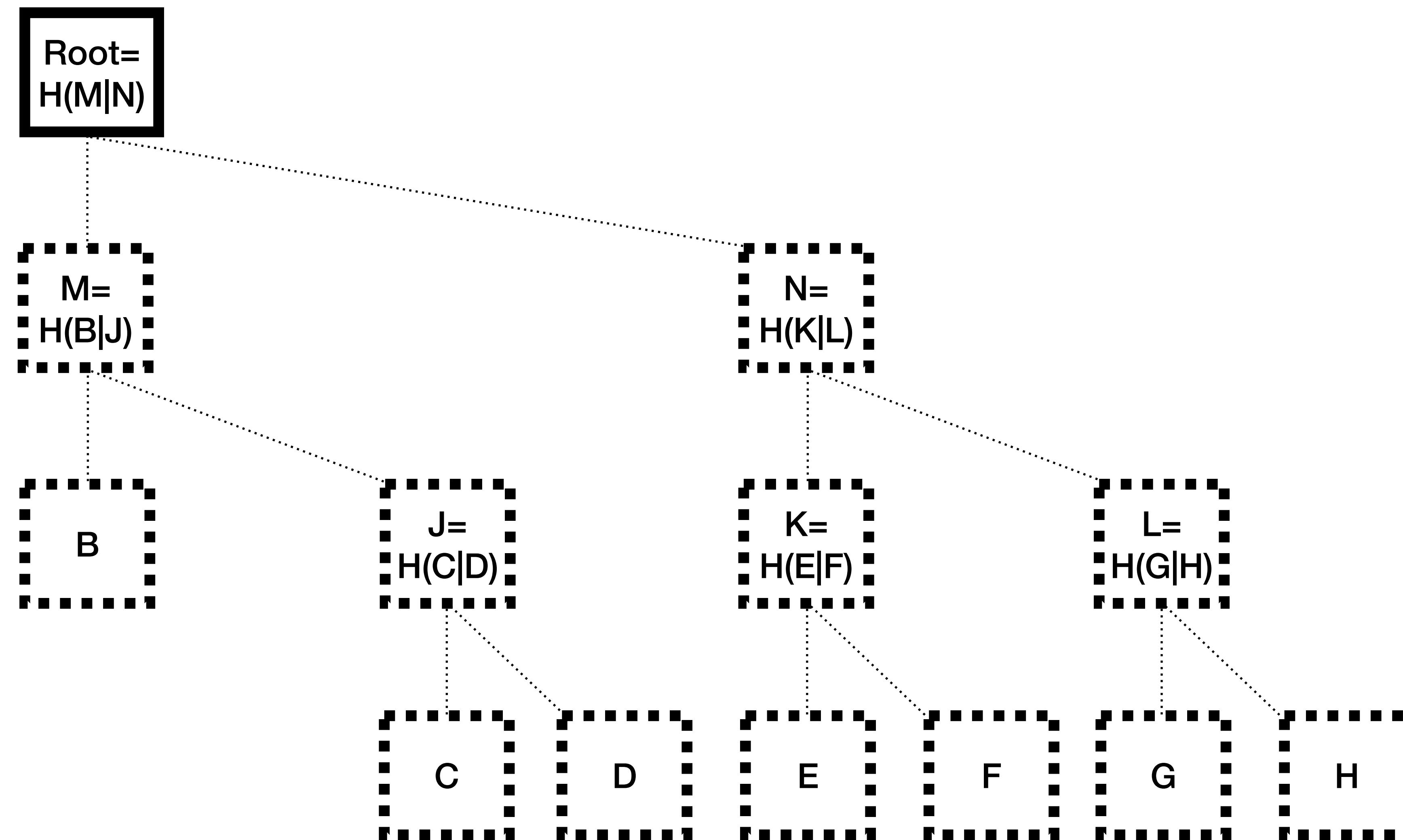
# Copy over the new root to be saved



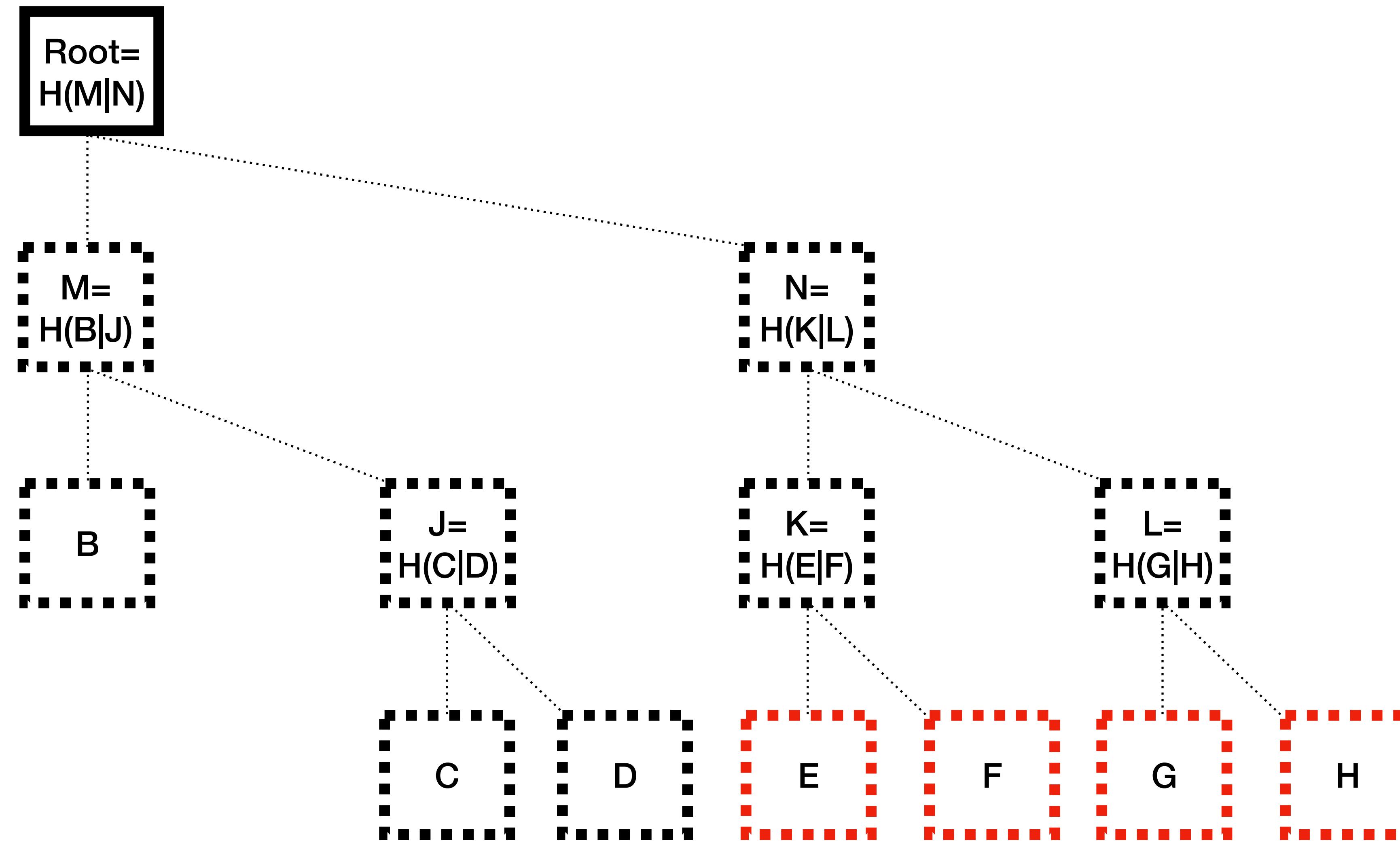
# Done

Root=  
 $H(M|N)$

# After deleting A



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

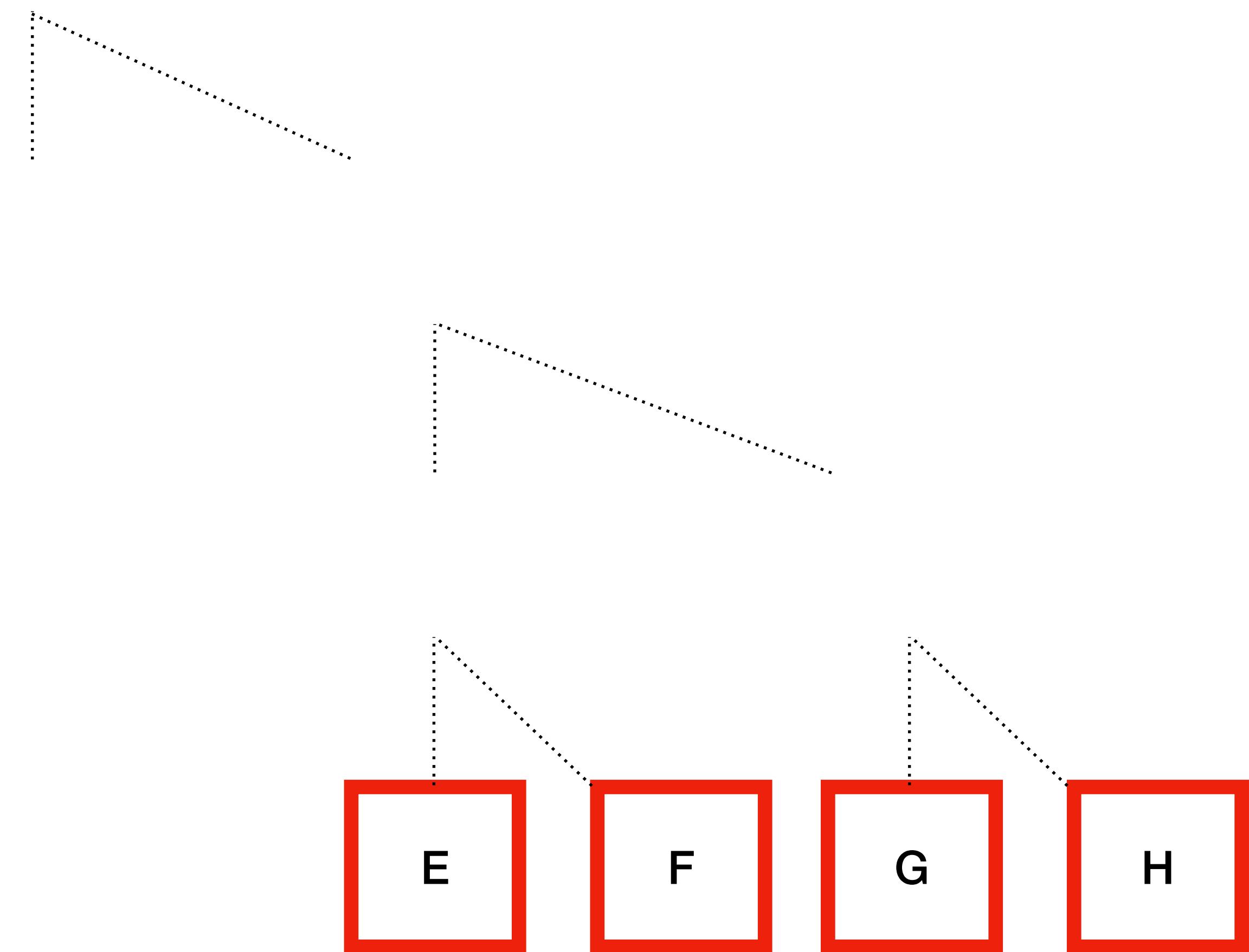


# Start off with only the root

Root=  
 $H(M|N)$

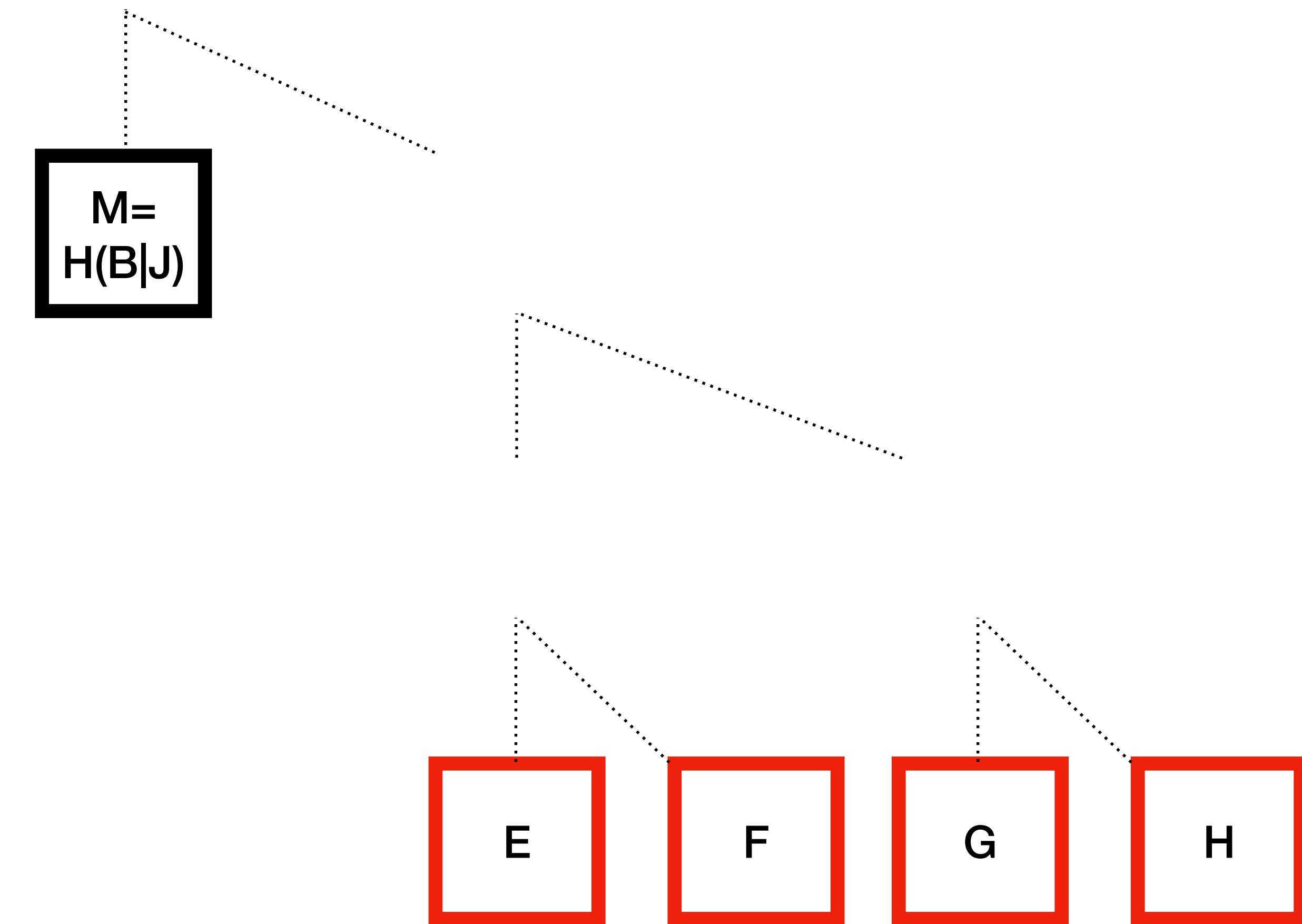
# Calculate E, F, G, H from the Bitcoin Block

Root=  
 $H(M|N)$

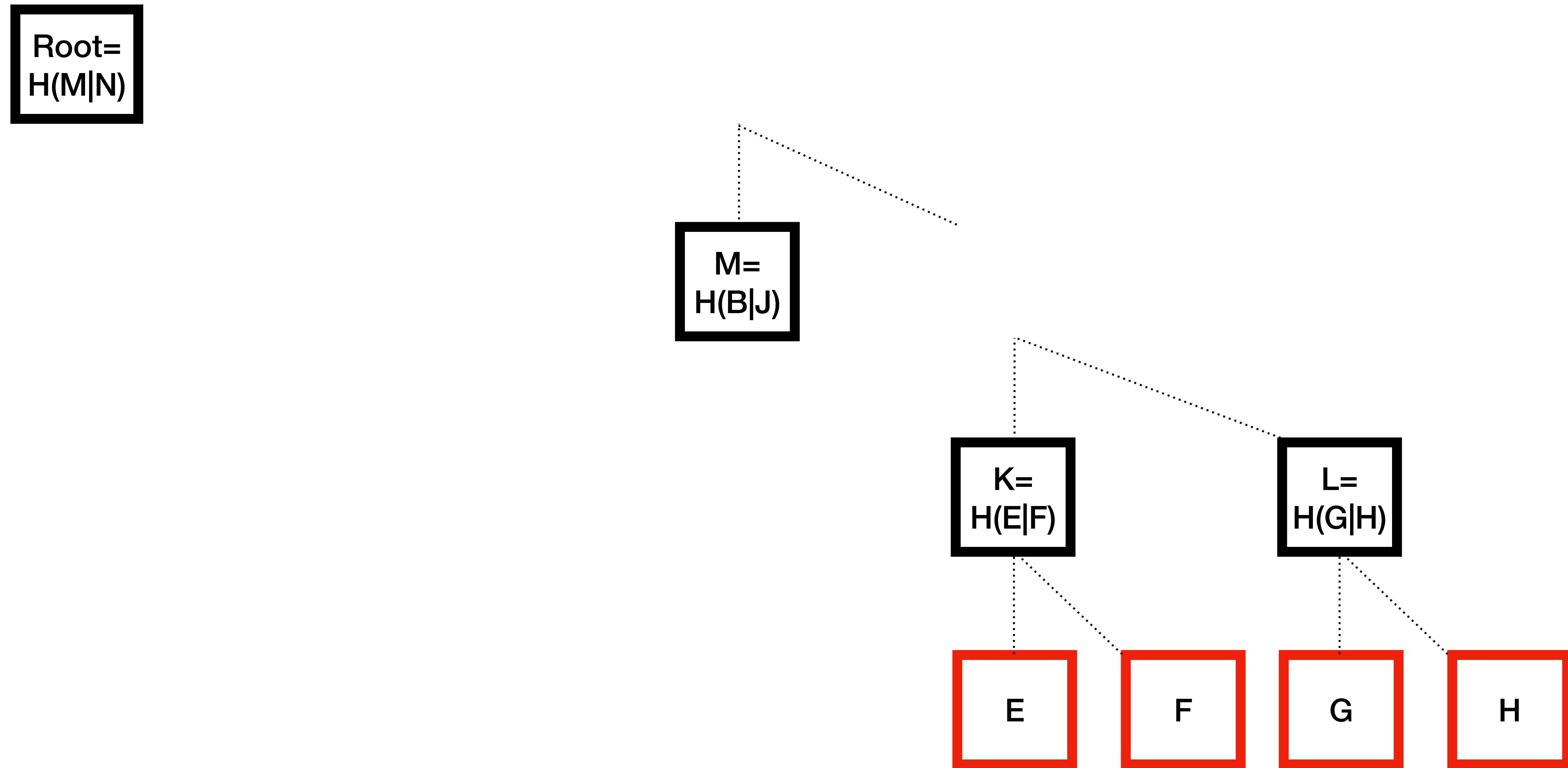


# Receive the proof: M

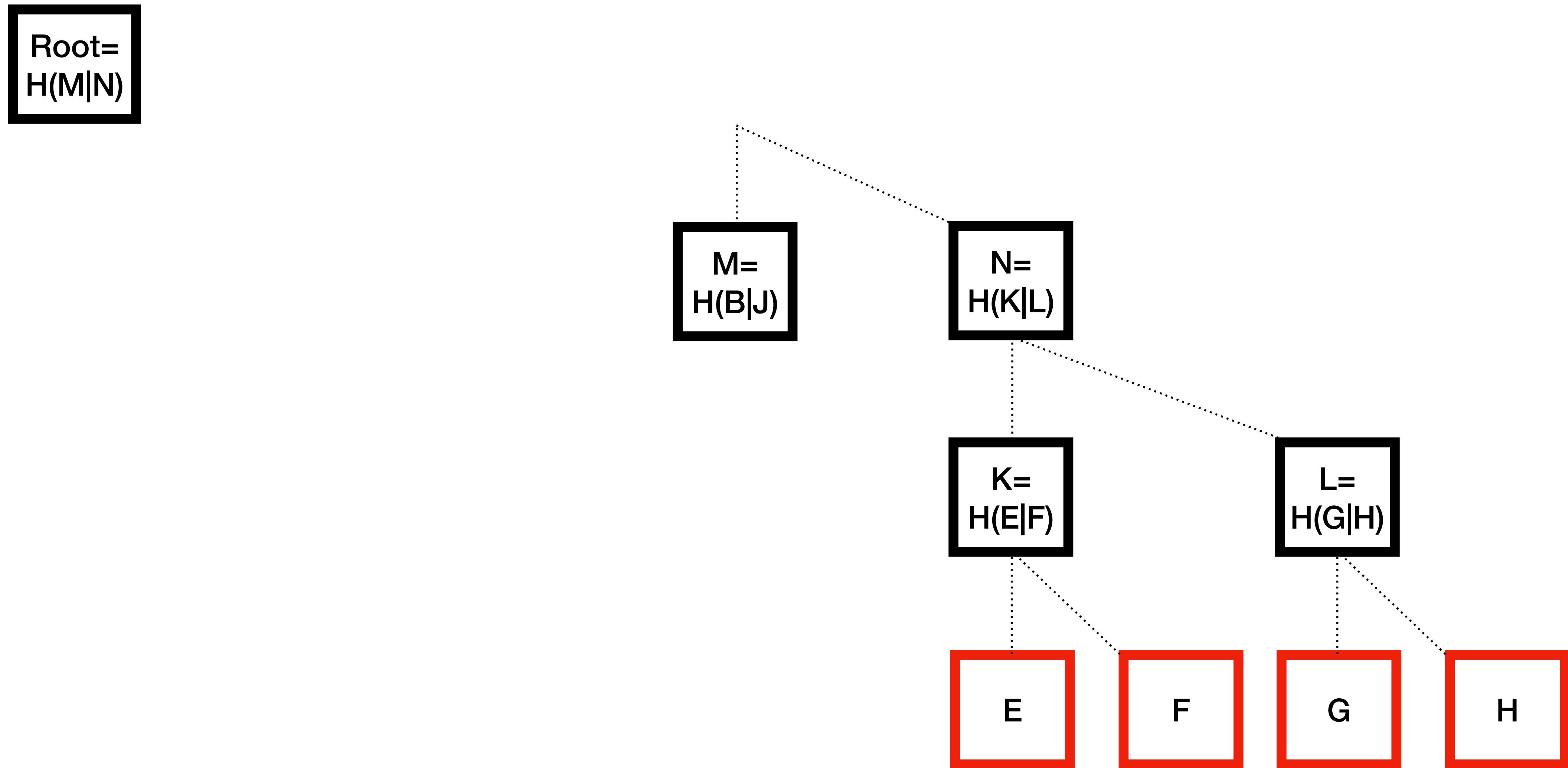
Root=  
 $H(M|N)$



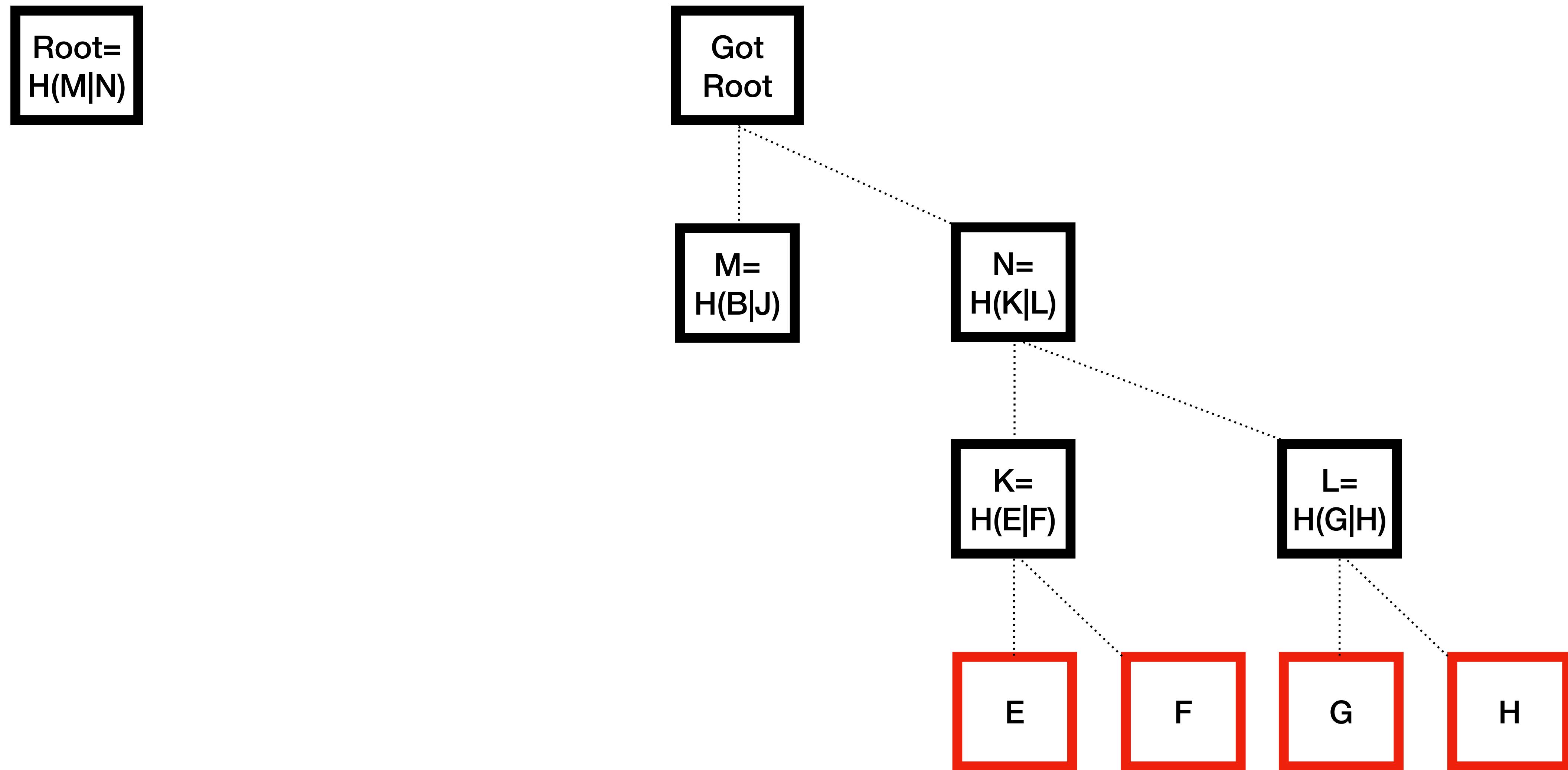
# Calculate K and L



# Calculate N



# Calculate the Root



# Compare roots

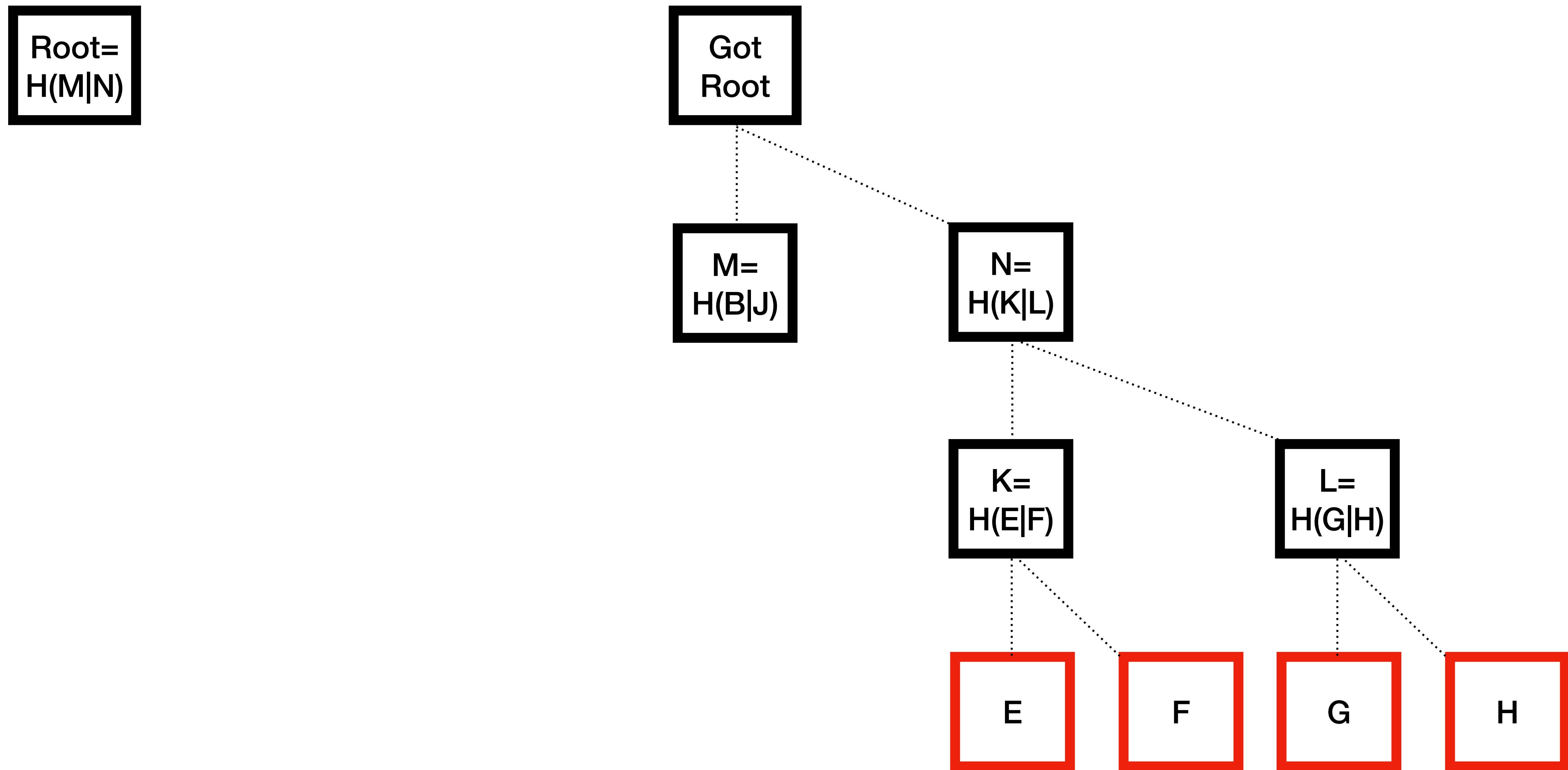
Continue if the roots are equal. Ban peer if not equal

Root= $H(M|N)$

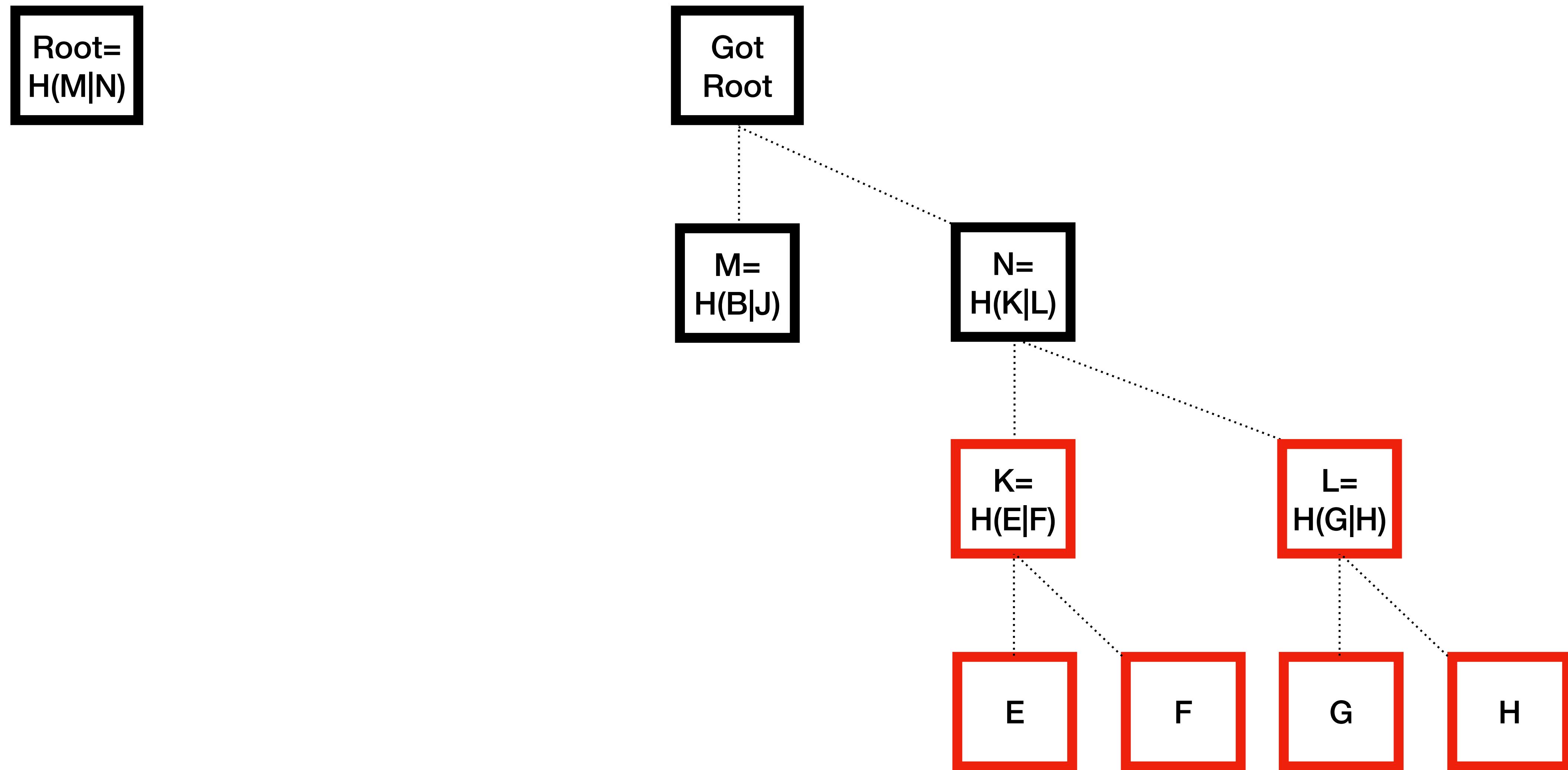
==

Got Root

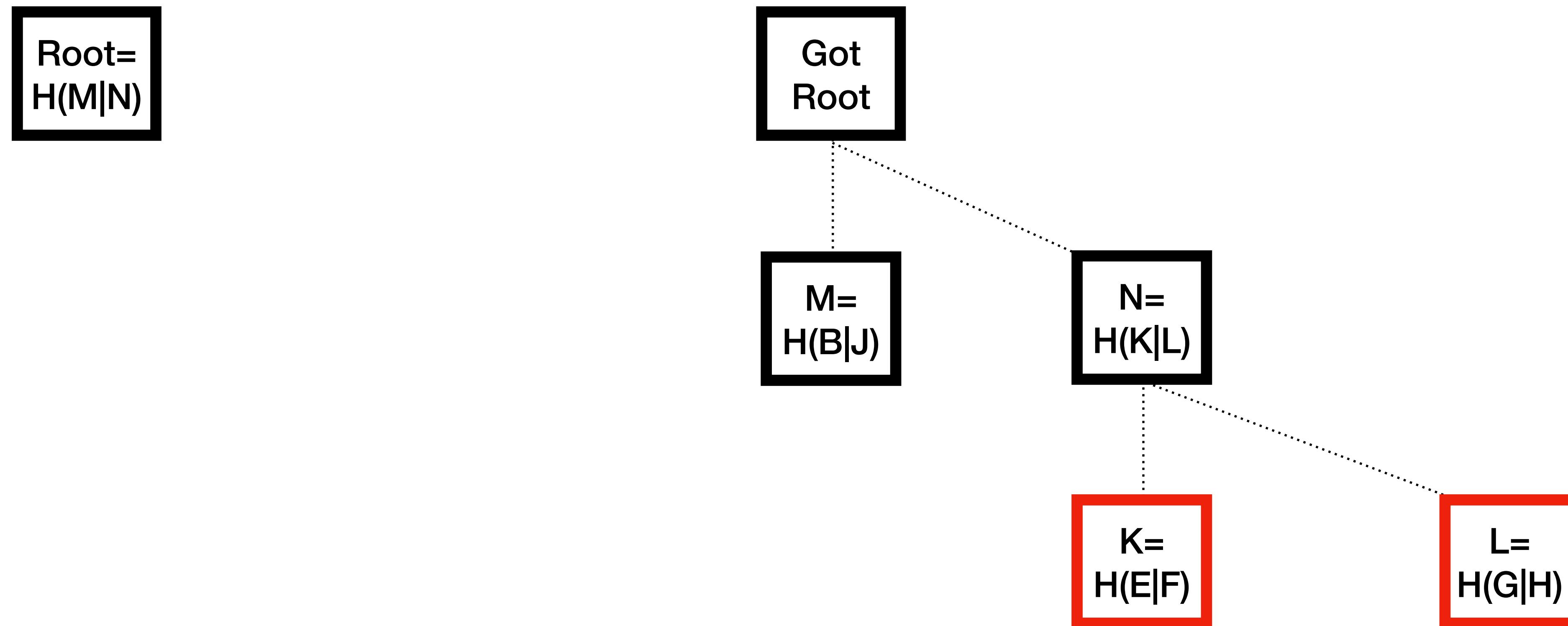
# Calculate new root



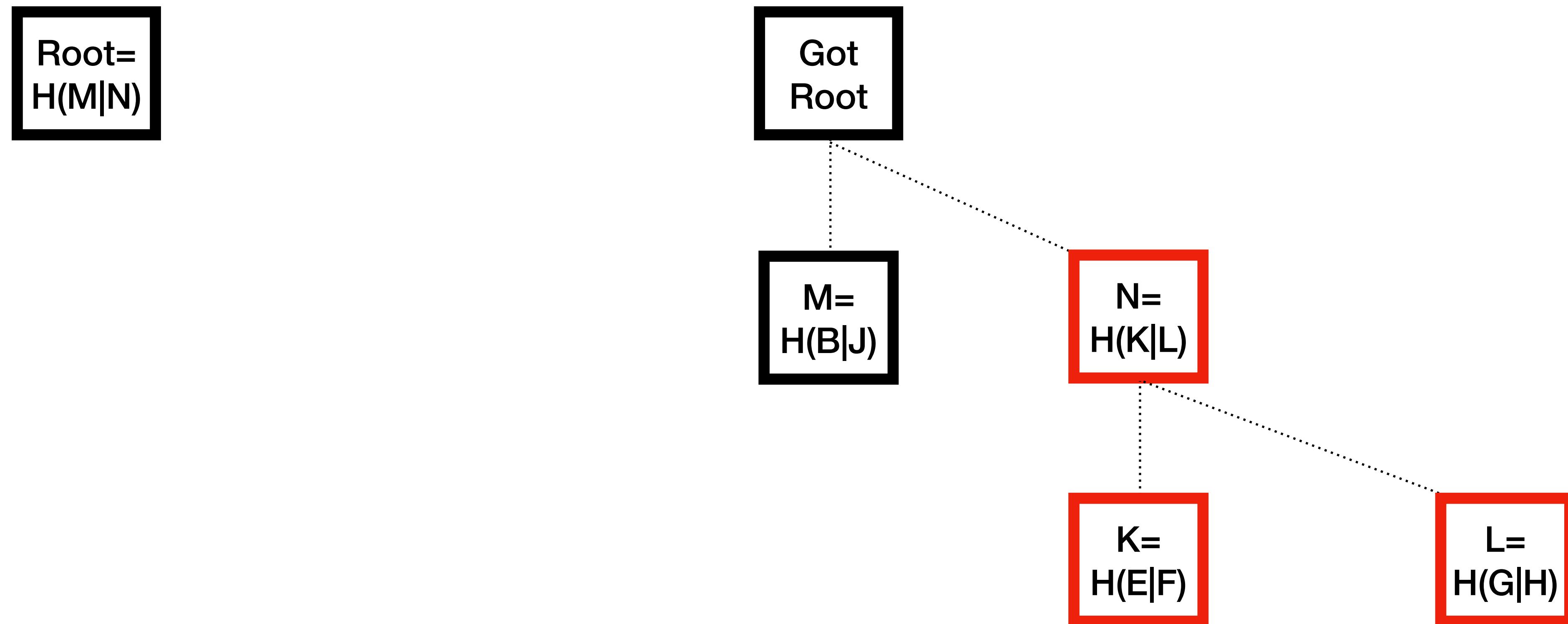
# Mark K & L as node to delete



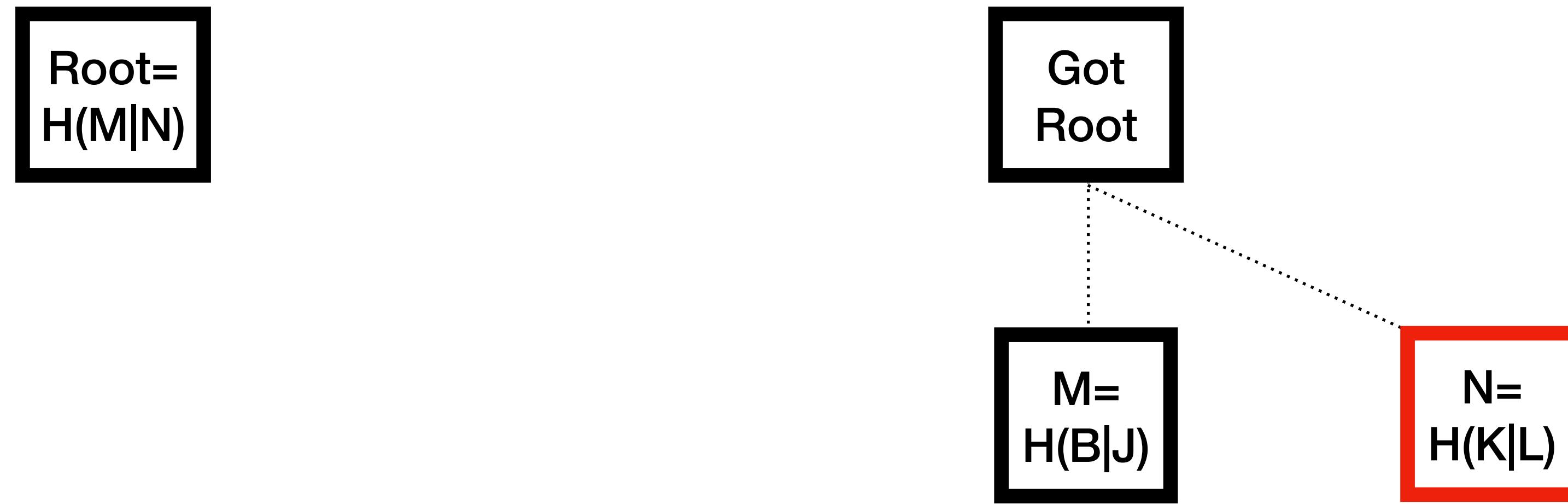
# Remove old nodes for E, F, G, H



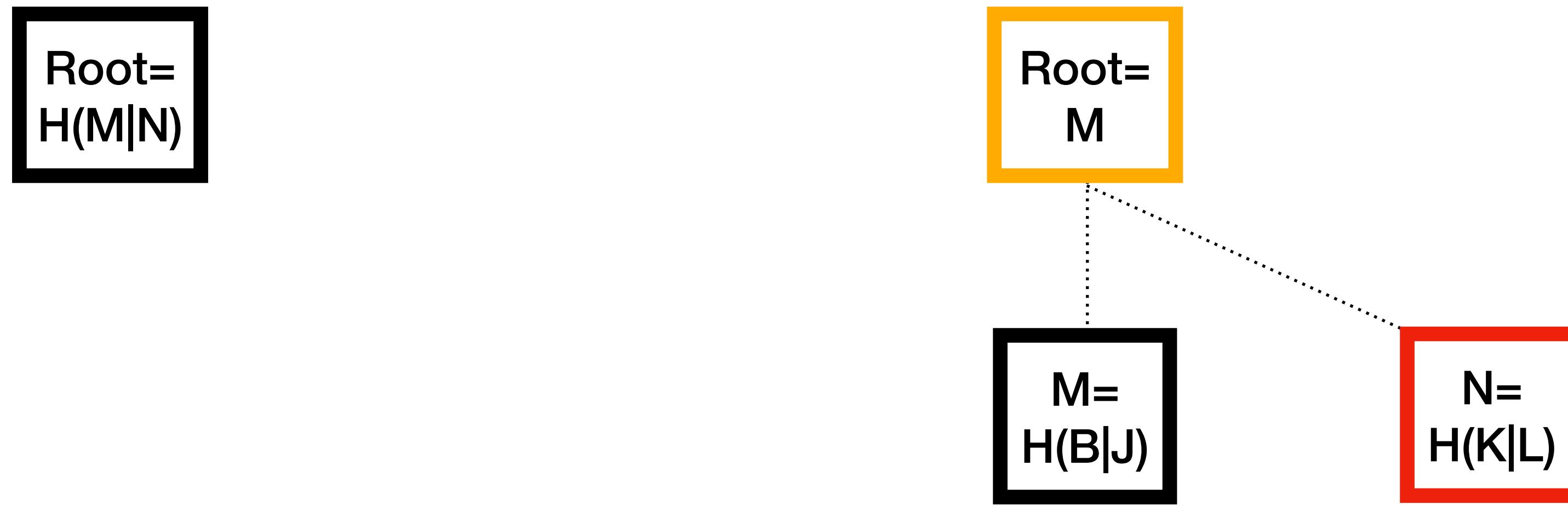
# Mark N as node to delete



# Remove old nodes for K & L



# Move up M

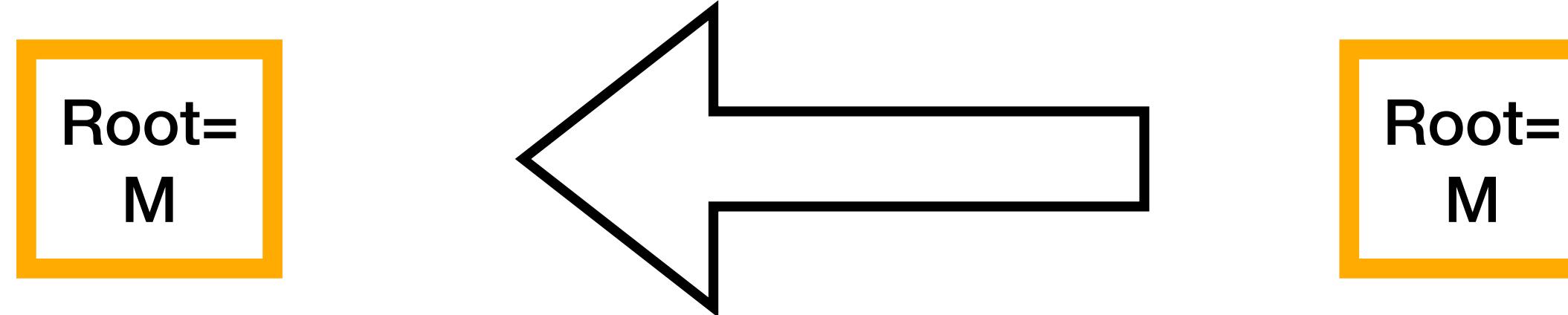


# Remove old nodes for M & N

Root=  
 $H(M|N)$

Root=  
M

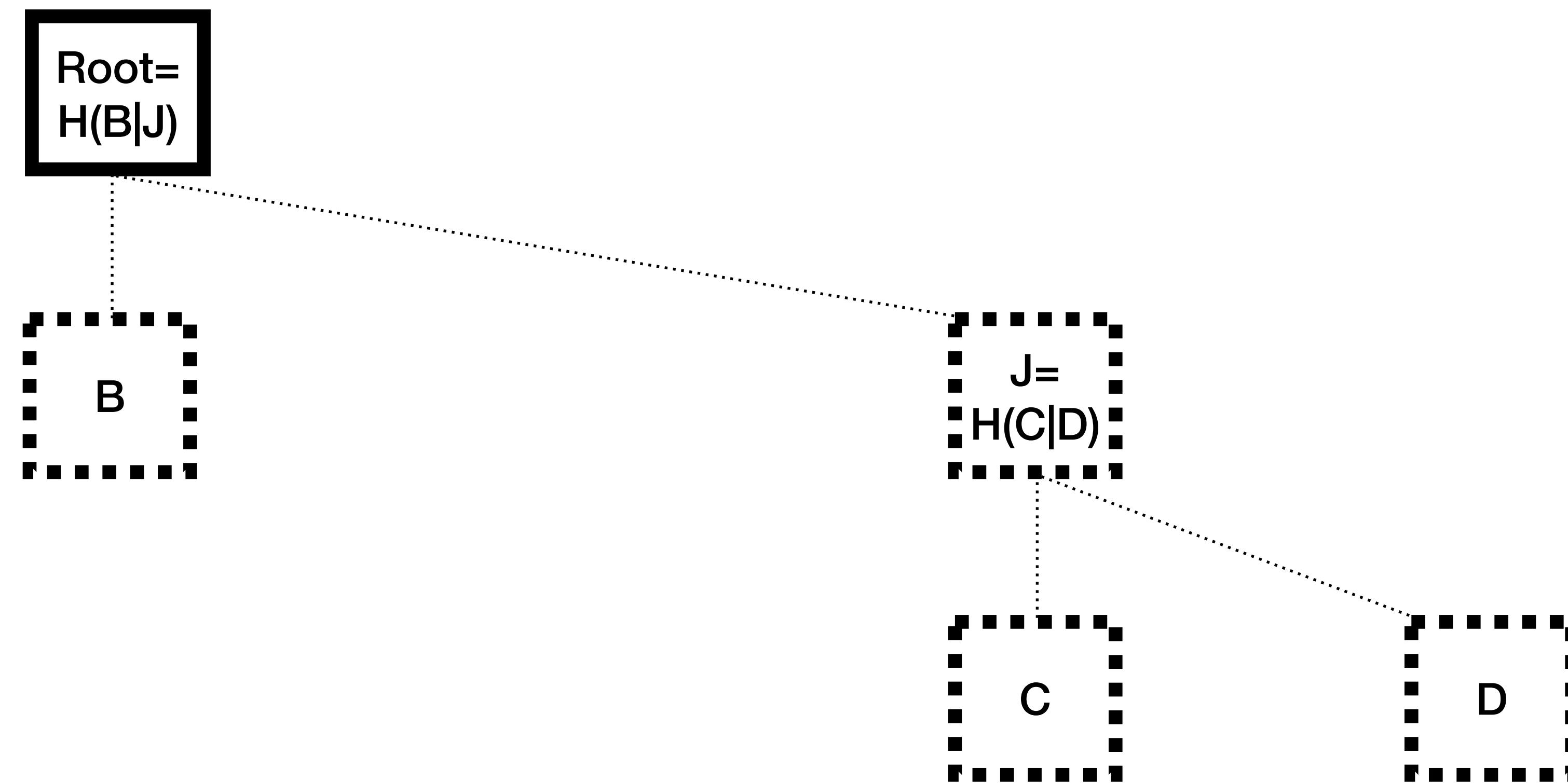
# Copy over the new root to be saved



# Done

Root=  
M

# After deleting E, F, G, H



# Consensus

## Add, Verify, Delete

- That's all the relevant algorithm for consensus
- Implemented in 174 lines of Python:  
[github.com/utreexo/pytreexo](https://github.com/utreexo/pytreexo)



[github.com/kcalvin/alvin/slides-for-tabconf-2023](https://github.com/kcalvin/alvin/slides-for-tabconf-2023)