

# Utreeexo - What is it?

## What it is and how it works

# Seoul Bitcoin Meetup

## Where I got started

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



# Seoul Bitcoin Meetup

Where I got started



# **About myself**

**Calvin Kim**

- Joined the meetup in 2017

# About myself

Calvin Kim

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

# About myself

Calvin Kim

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

# About myself

Calvin Kim

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

# About myself

Calvin Kim

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

**Utreexo**

**Agenda for today**

**How does Bitcoin work?**

# **Utreeexo**

## **Agenda for today**

How does Bitcoin work?

What is Utreeexo?

# Utreeexo

## Agenda for today

How does Bitcoin work?

What is Utreeexo?

How does Utreeexo work?

**Utreexo**

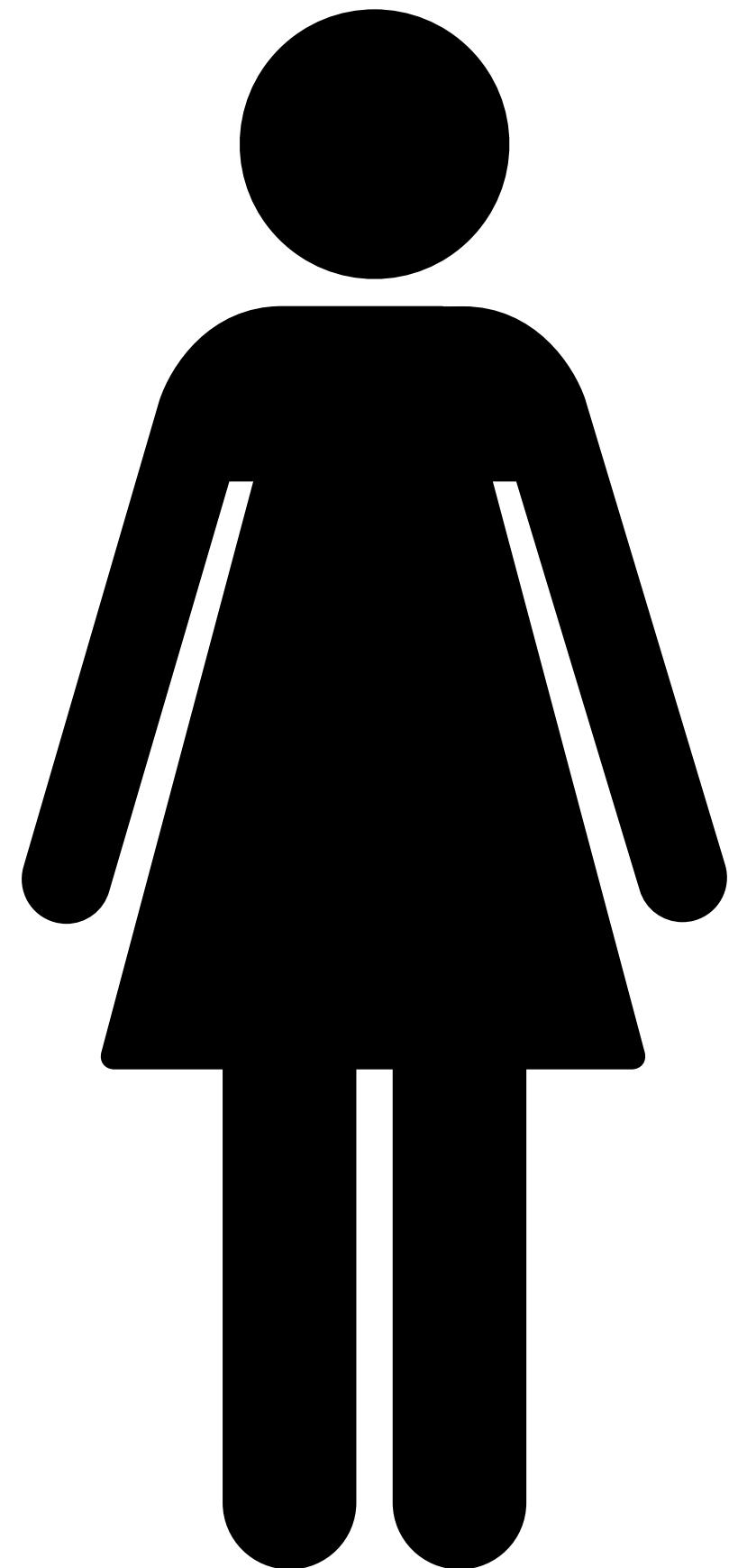
**Agenda for today**

**Questions encouraged!**

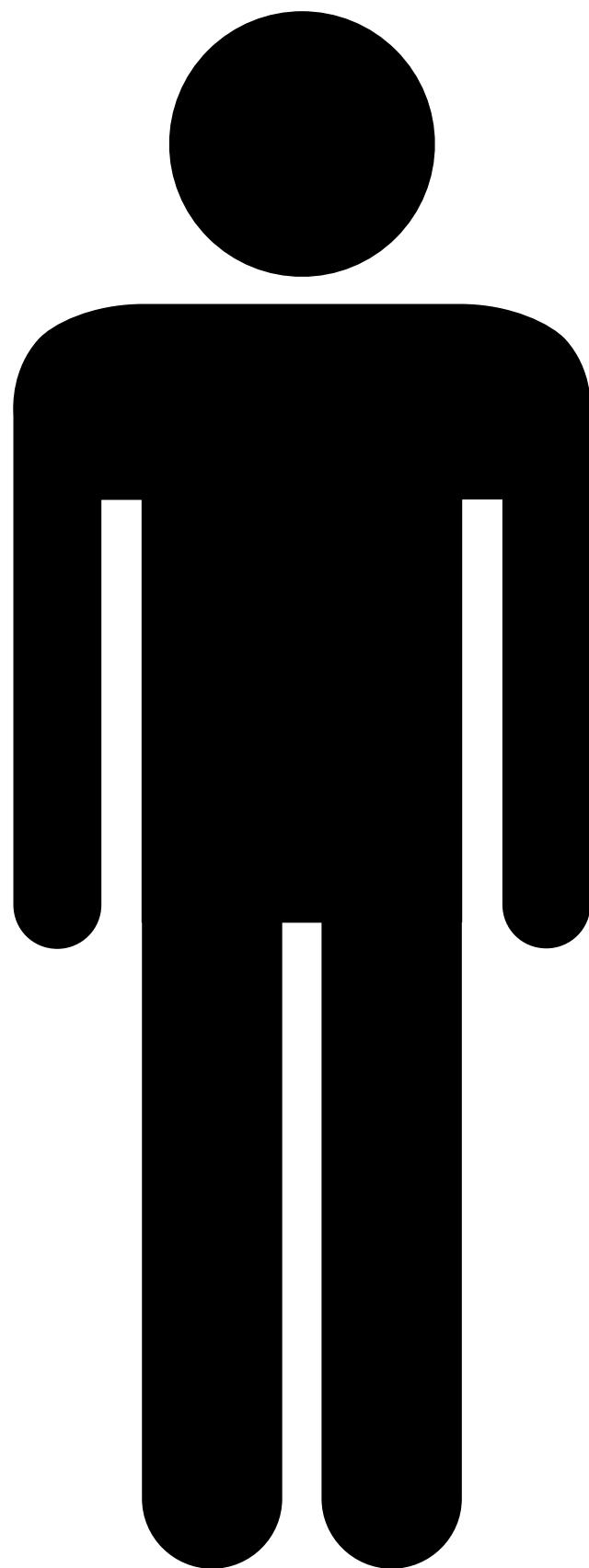
# How does Bitcoin work?

**Bartering with  
cash vs with Bitcoin**

# Barter with cash

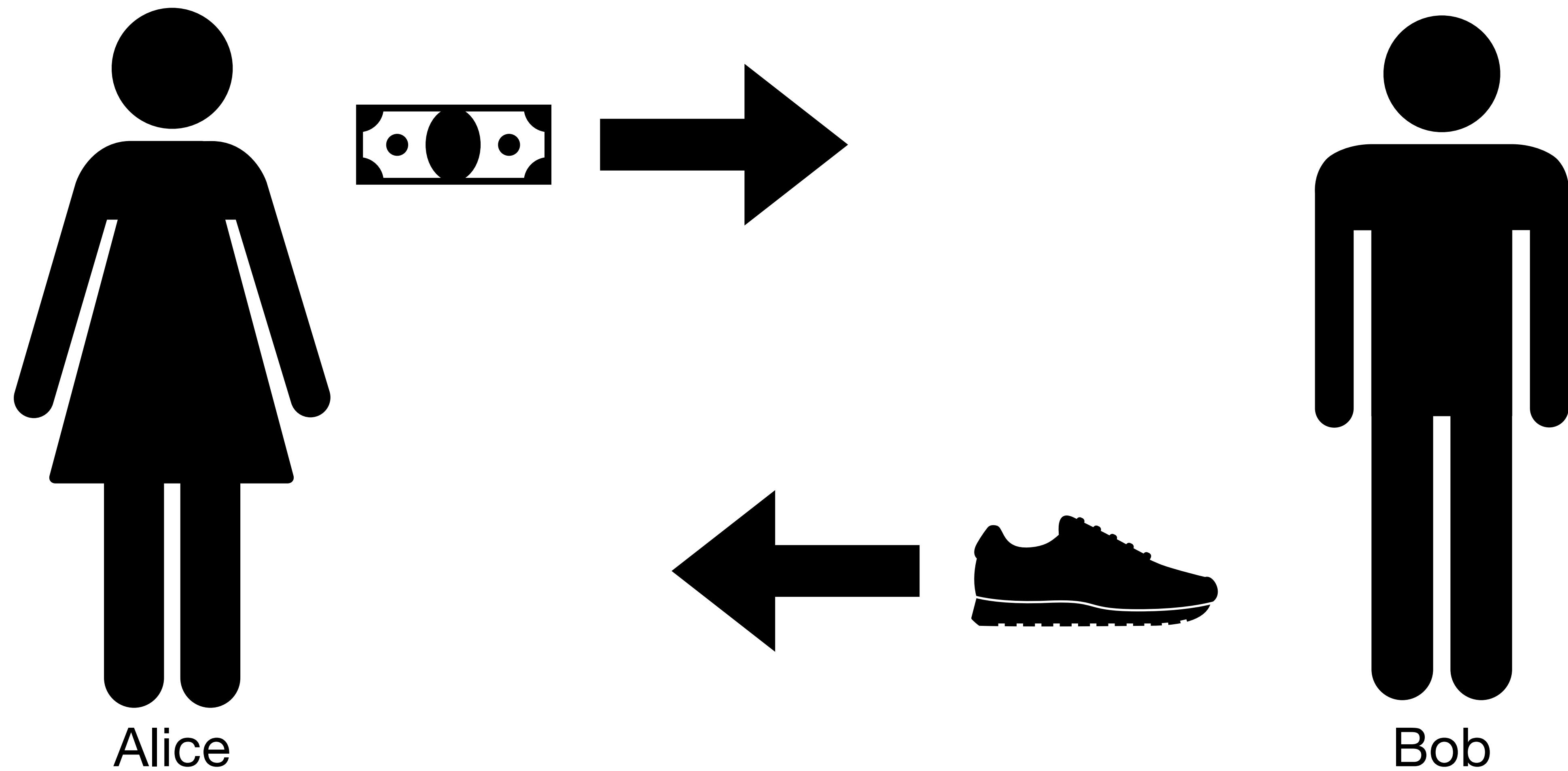


Alice

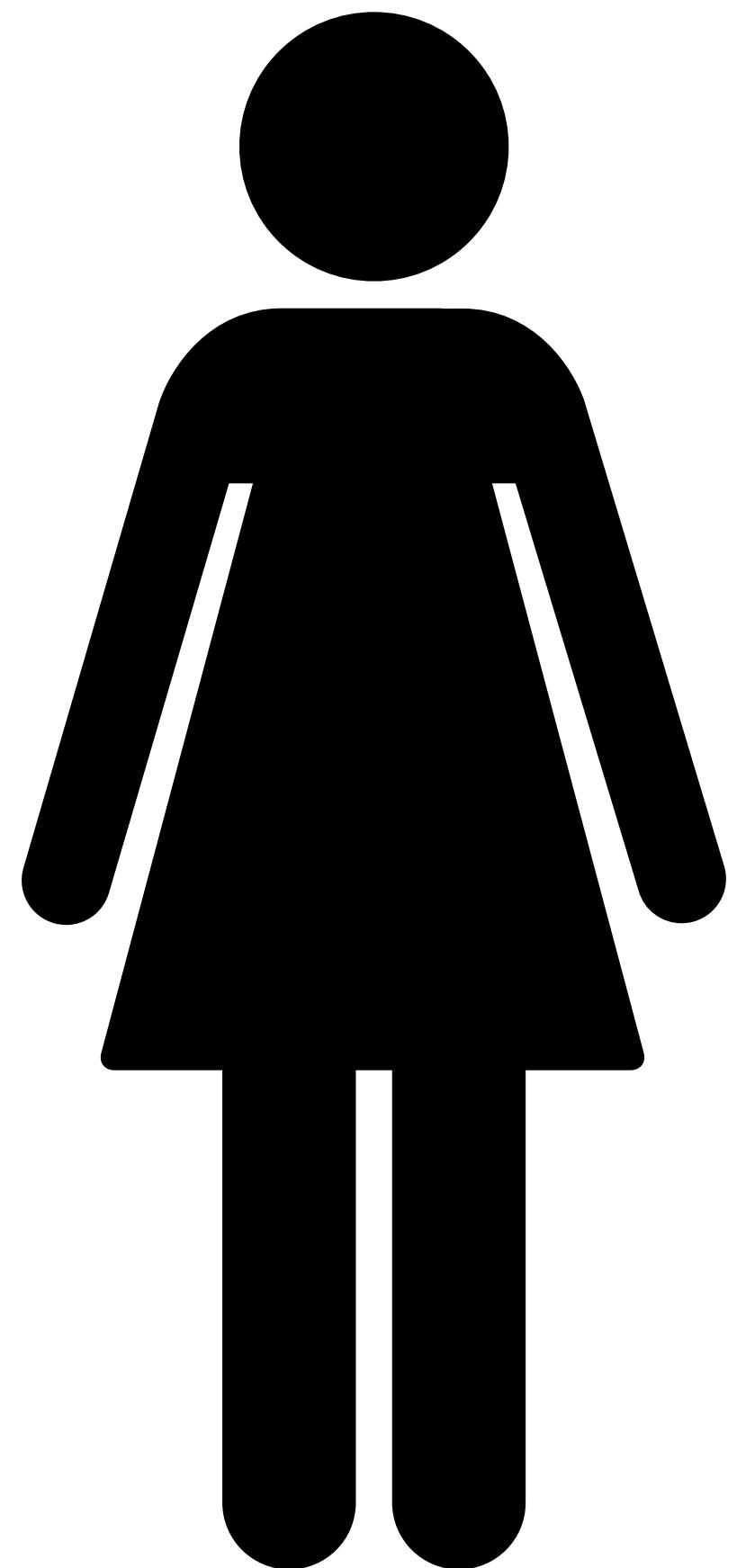


Bob

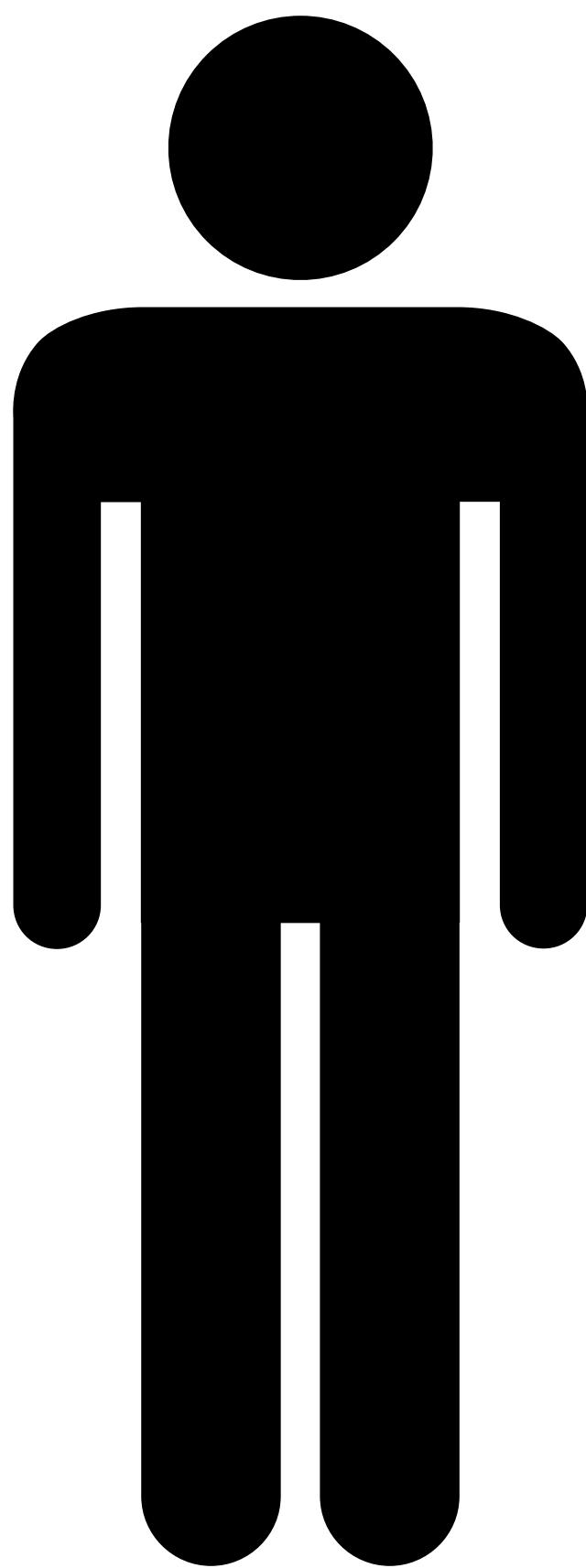
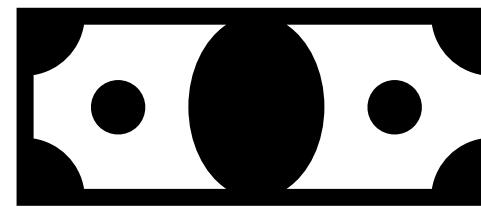
# Barter with cash



# Barter with cash



Alice



Bob

# Barter with cash

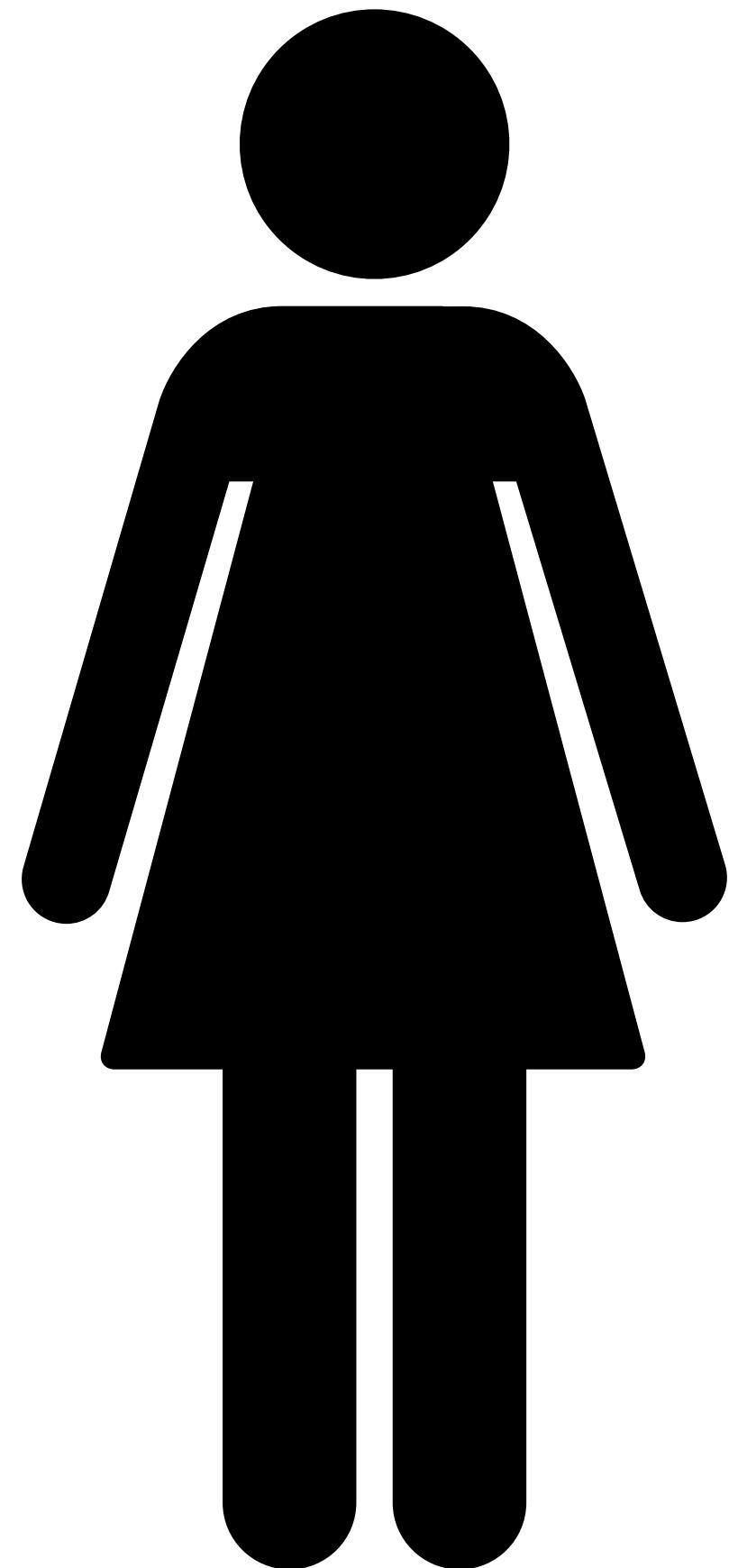
- Alice hands over the dollar
- Bob hands over the shoes

# Barter with cash

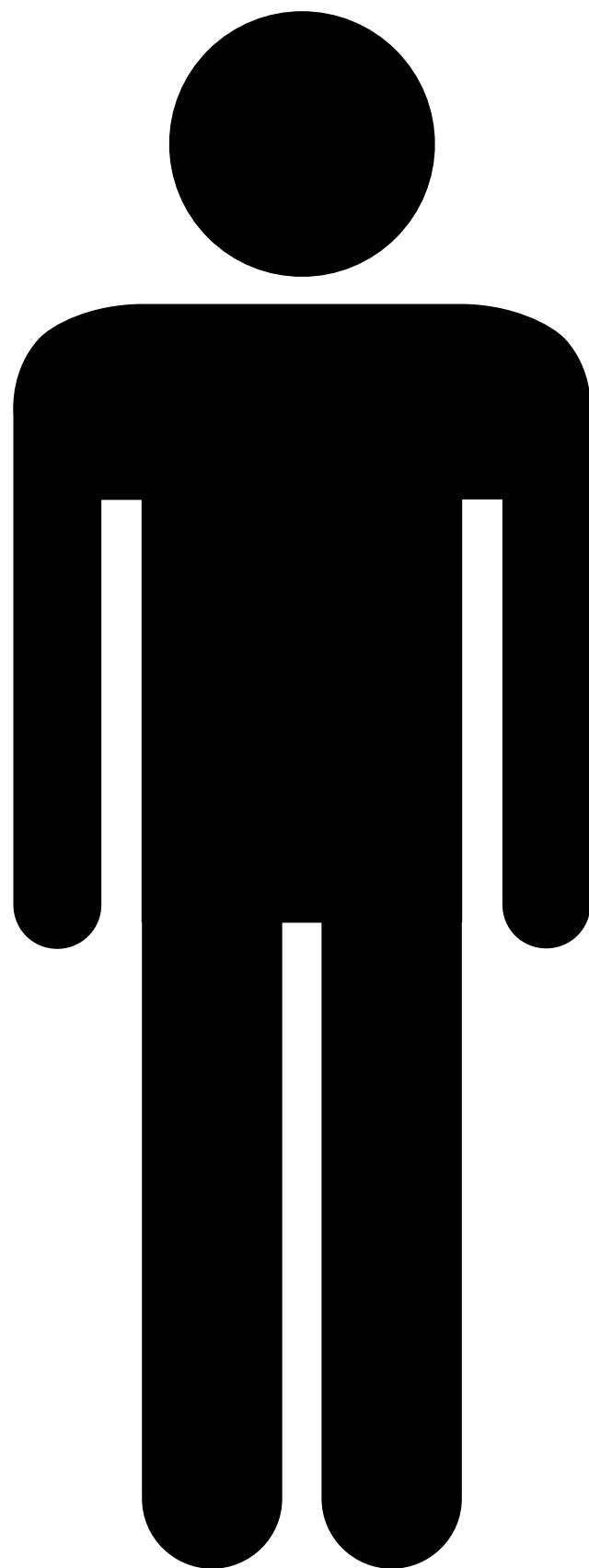
- Alice hands over the dollar
- Bob hands over the shoes
- **Physical exchange has been made**

# Barter with Bitcoin

# Barter with Bitcoin

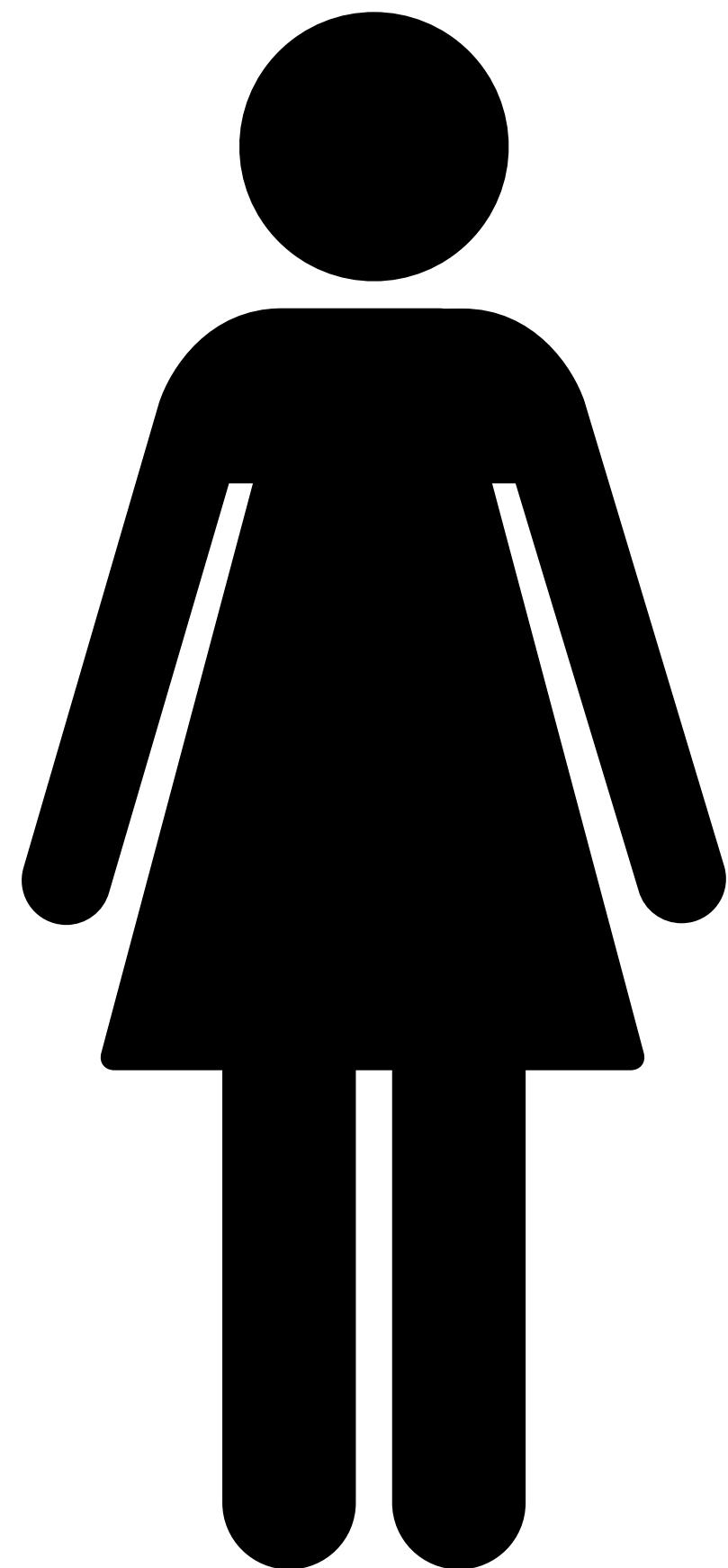


Alice

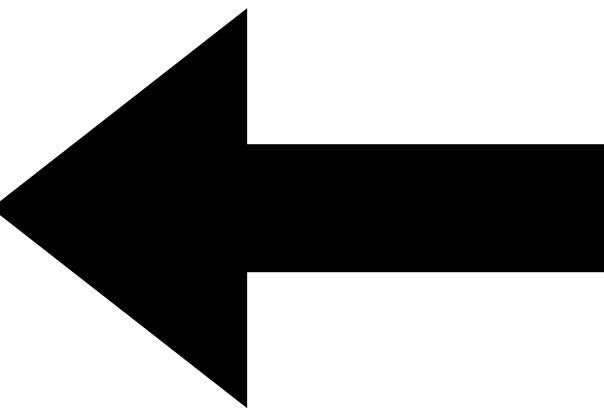


Bob

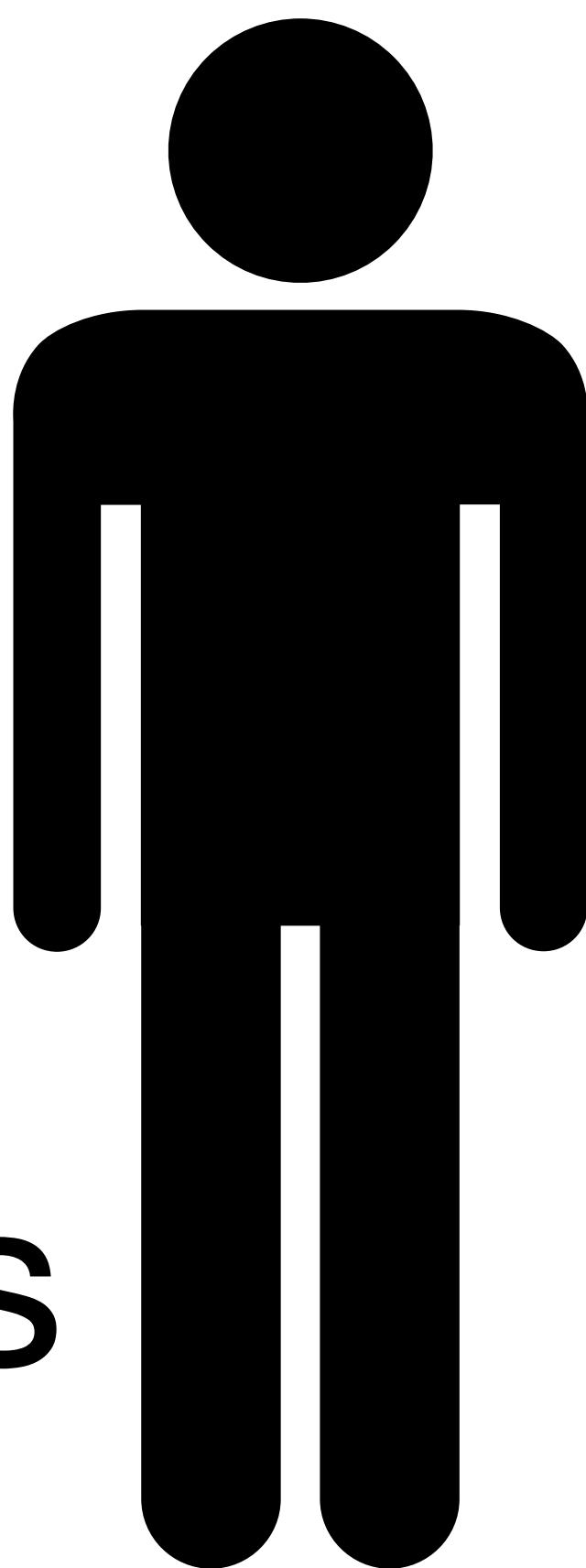
# Barter with Bitcoin



Alice

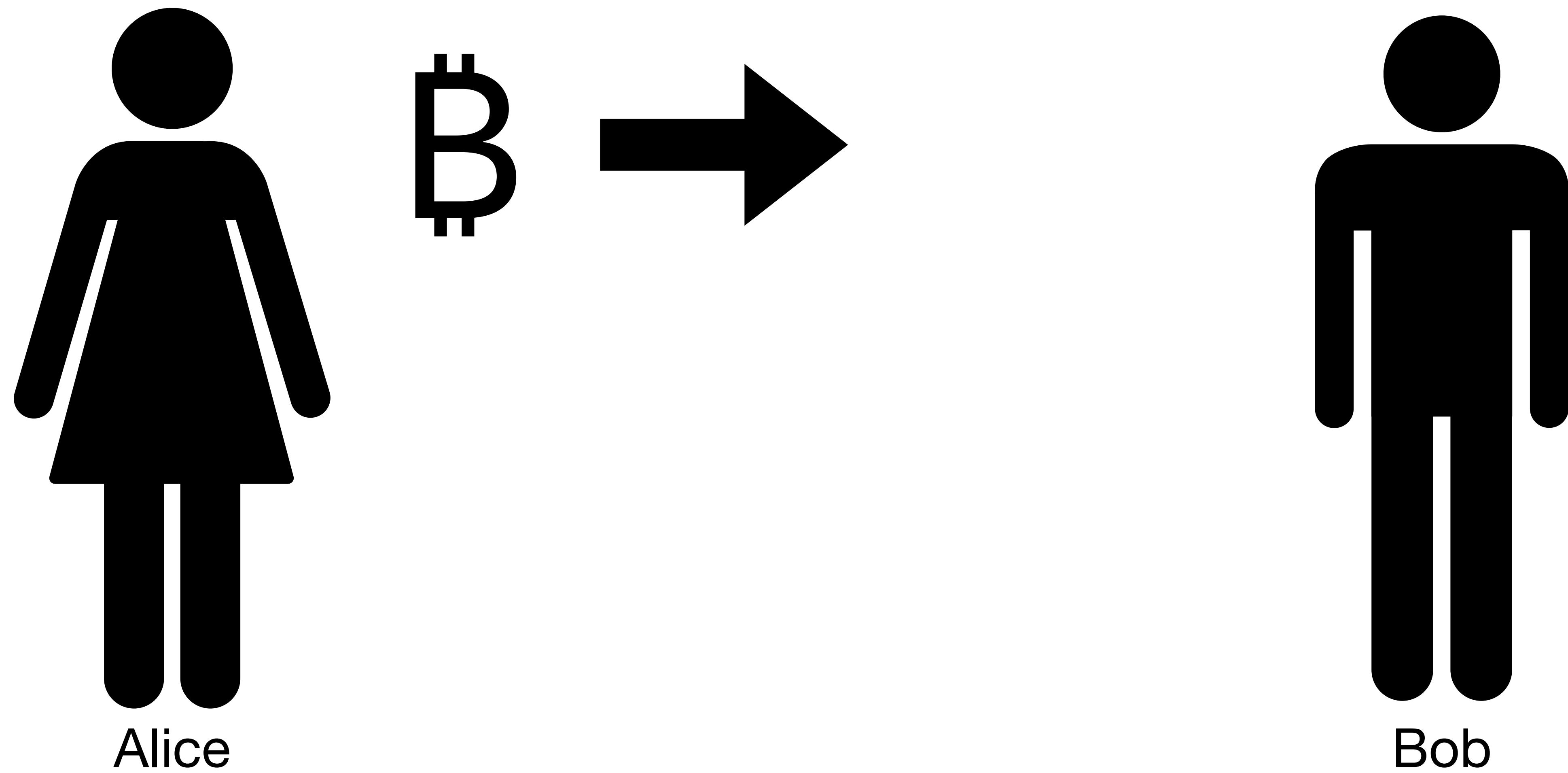


Address

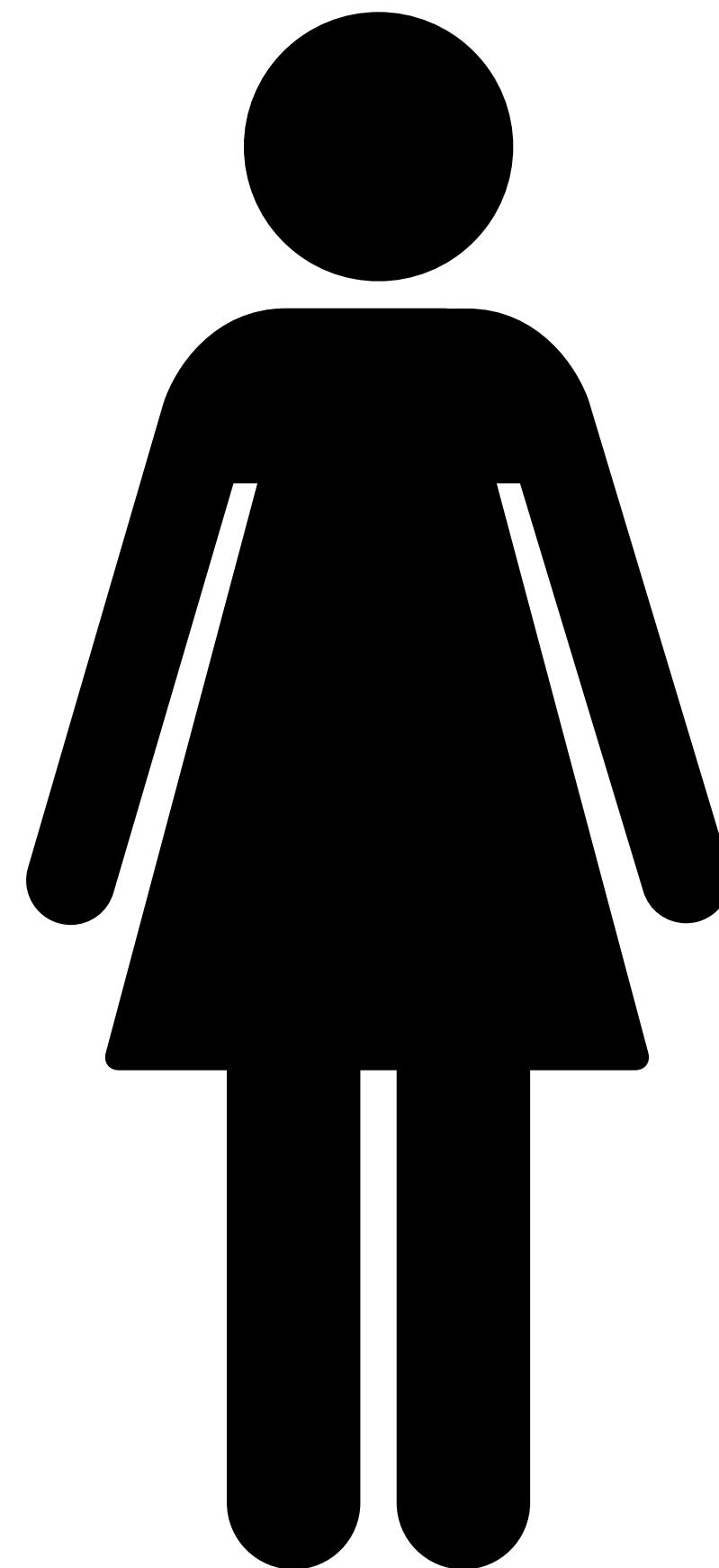


Bob

# Barter with Bitcoin

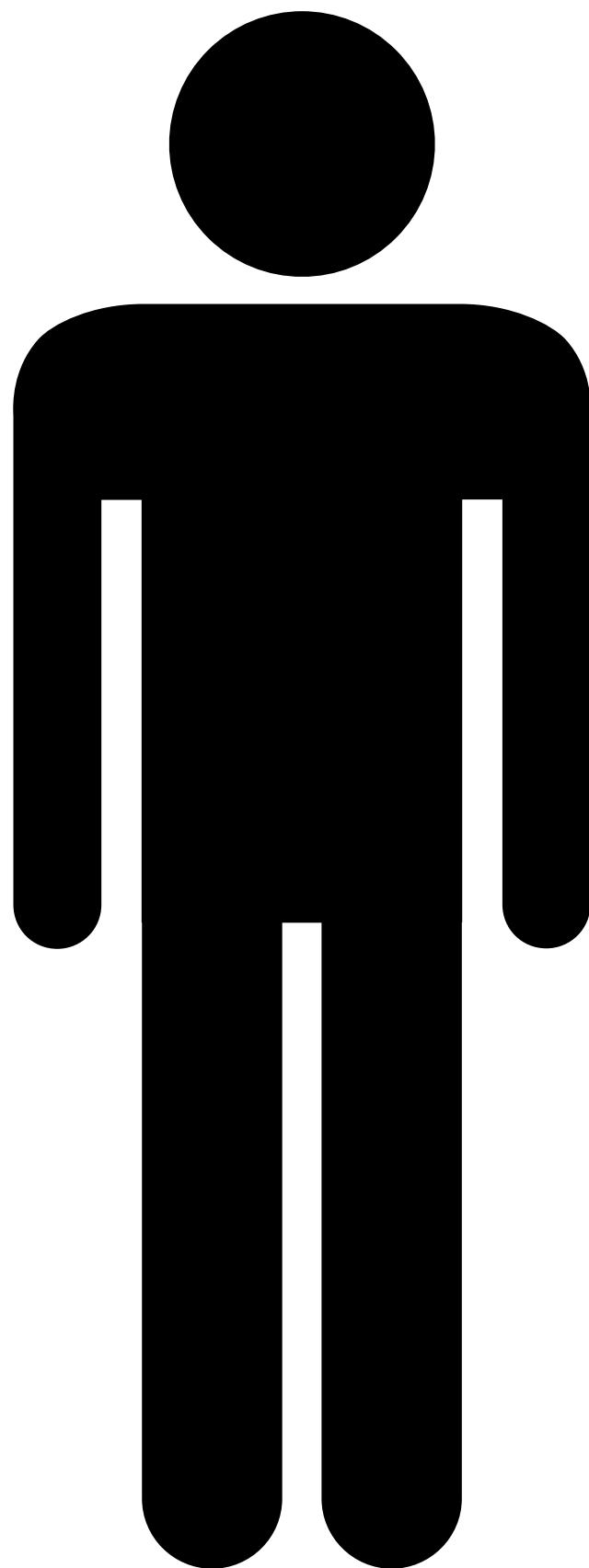


# Barter with Bitcoin



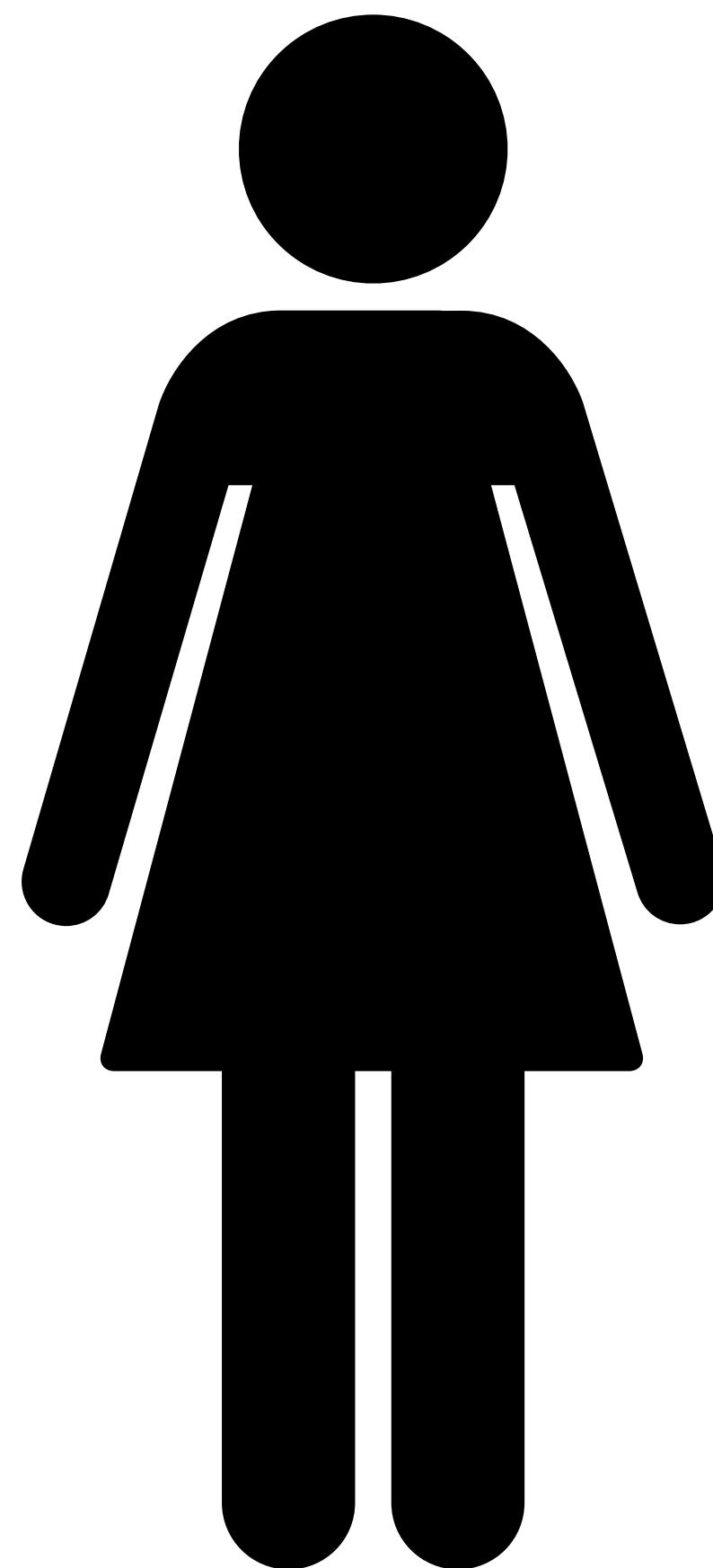
Alice

฿  
1 conf

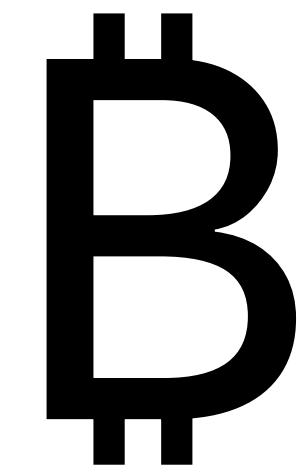
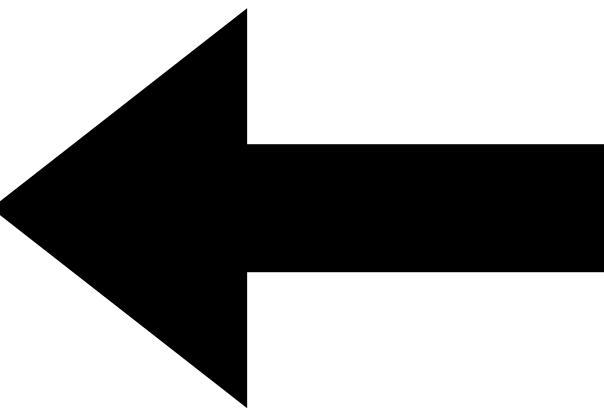


Bob

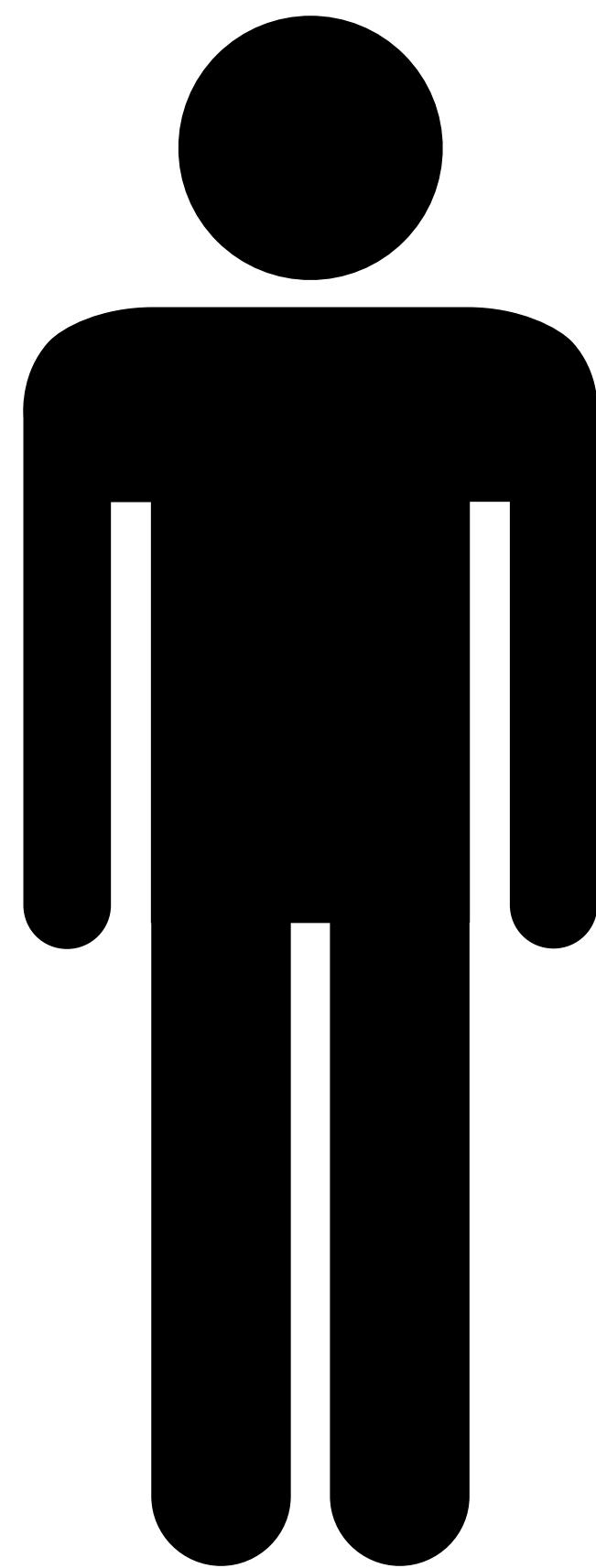
# Barter with Bitcoin



Alice

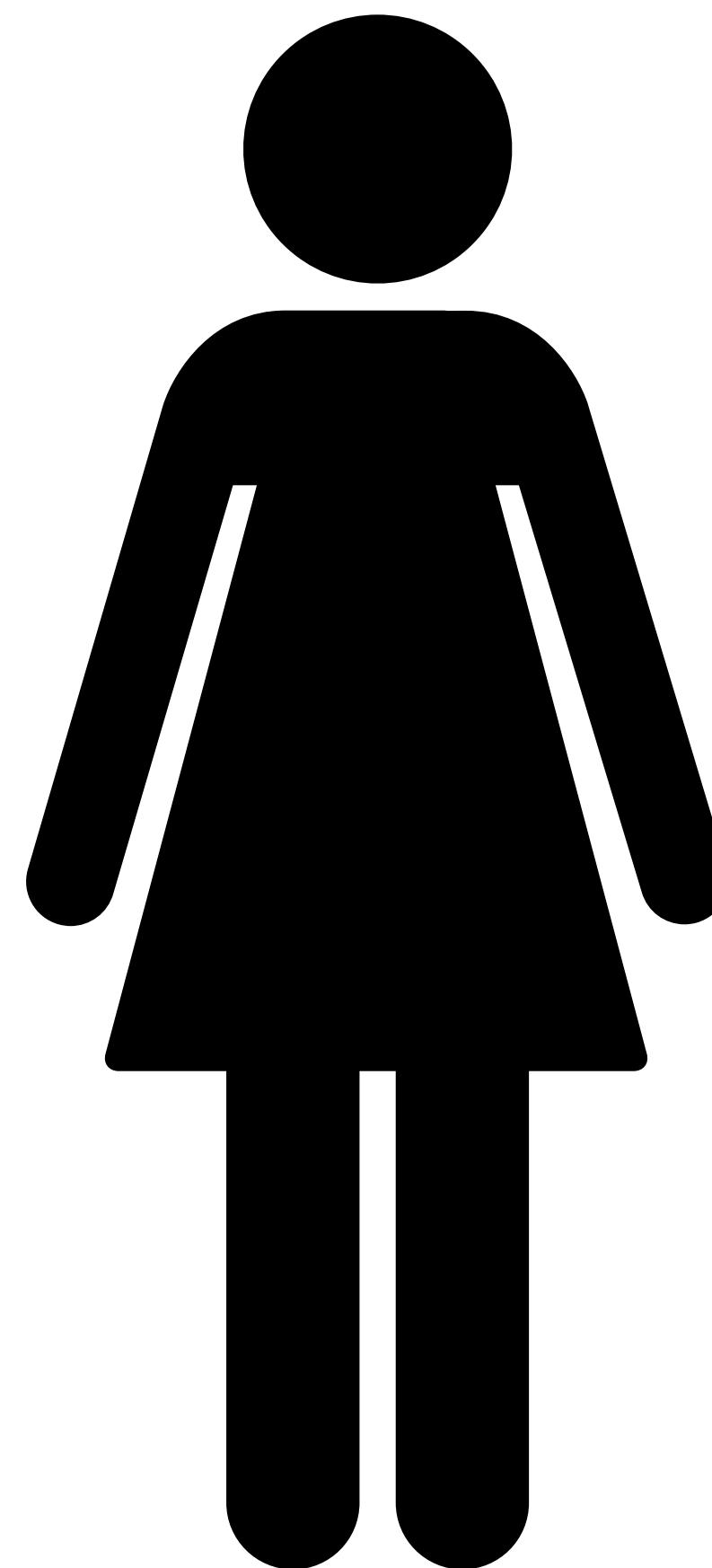


1 conf



Bob

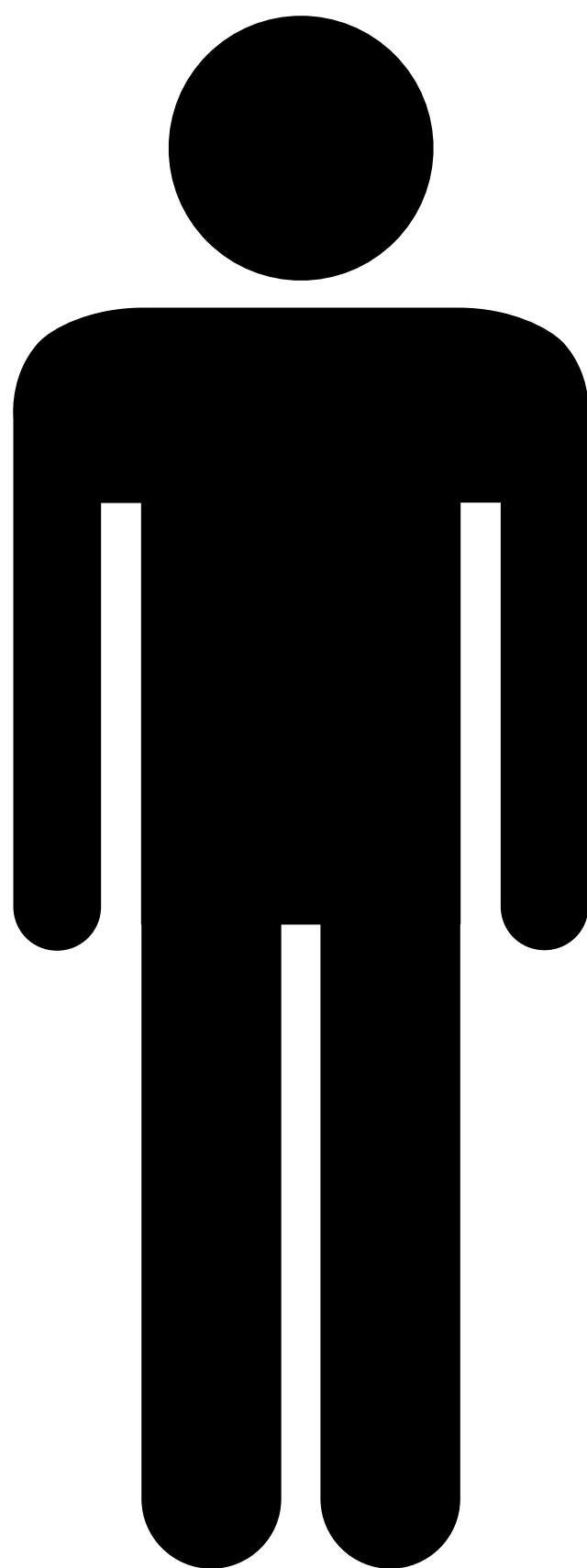
# Barter with Bitcoin



Alice



฿  
1 conf



Bob

# Barter with Bitcoin

- Alice sends Bitcoin to Bob

# Barter with Bitcoin

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

# Barter with Bitcoin

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

## Cash

- Physically hand over the bill

## Bitcoin

- Wait for a confirmation of the tx

# What's a confirmation?

## Bitcoin's rules

- Payment isn't final before a confirmation

# What's a confirmation?

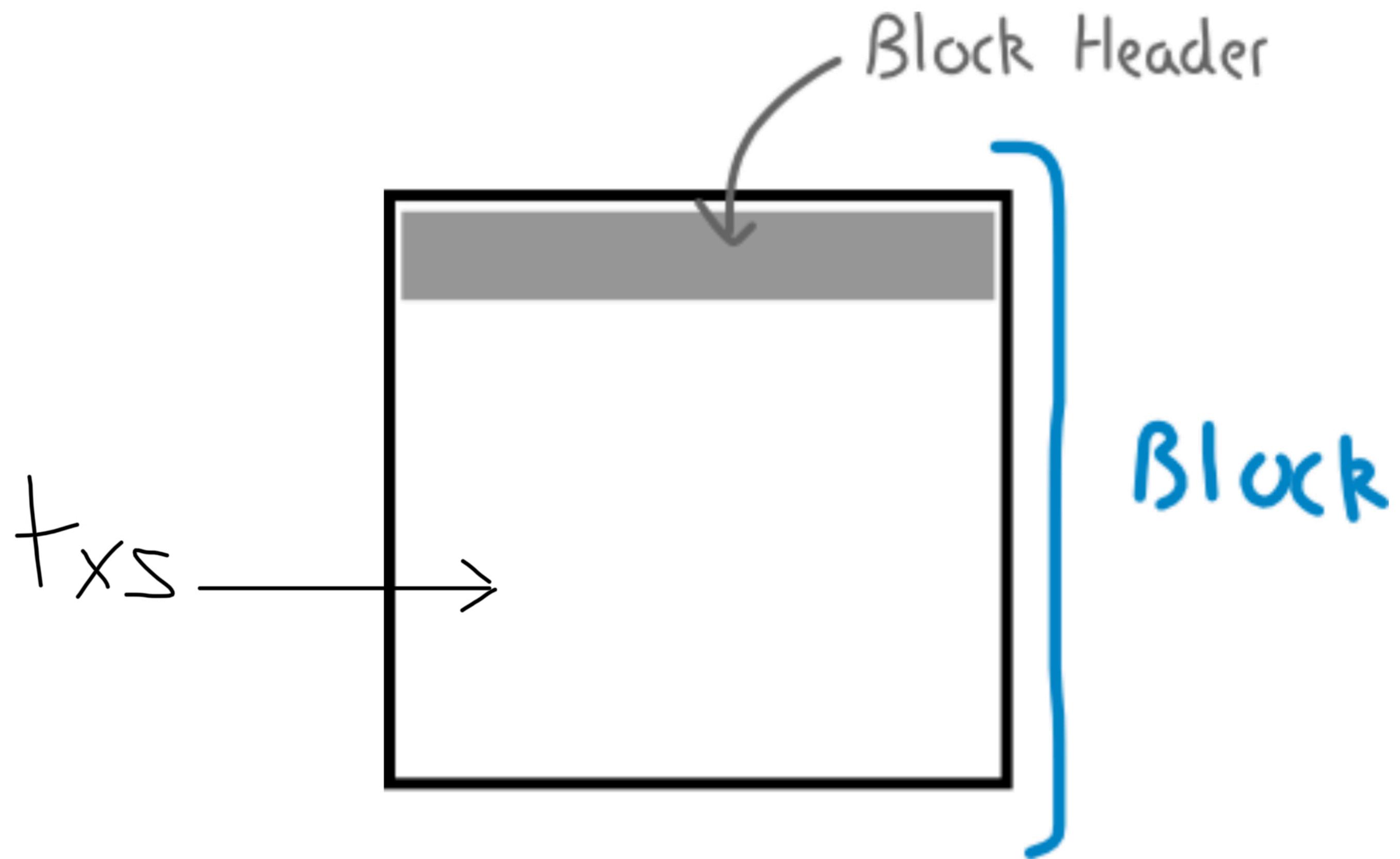
## Bitcoin's rules

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

**What's a block?**

# Bitcoin Block

## Bitcoin Block Internals



# **Bitcoin Block**

## **Bitcoin Block Internals**

- Metadata (Block Header)

# **Bitcoin Block**

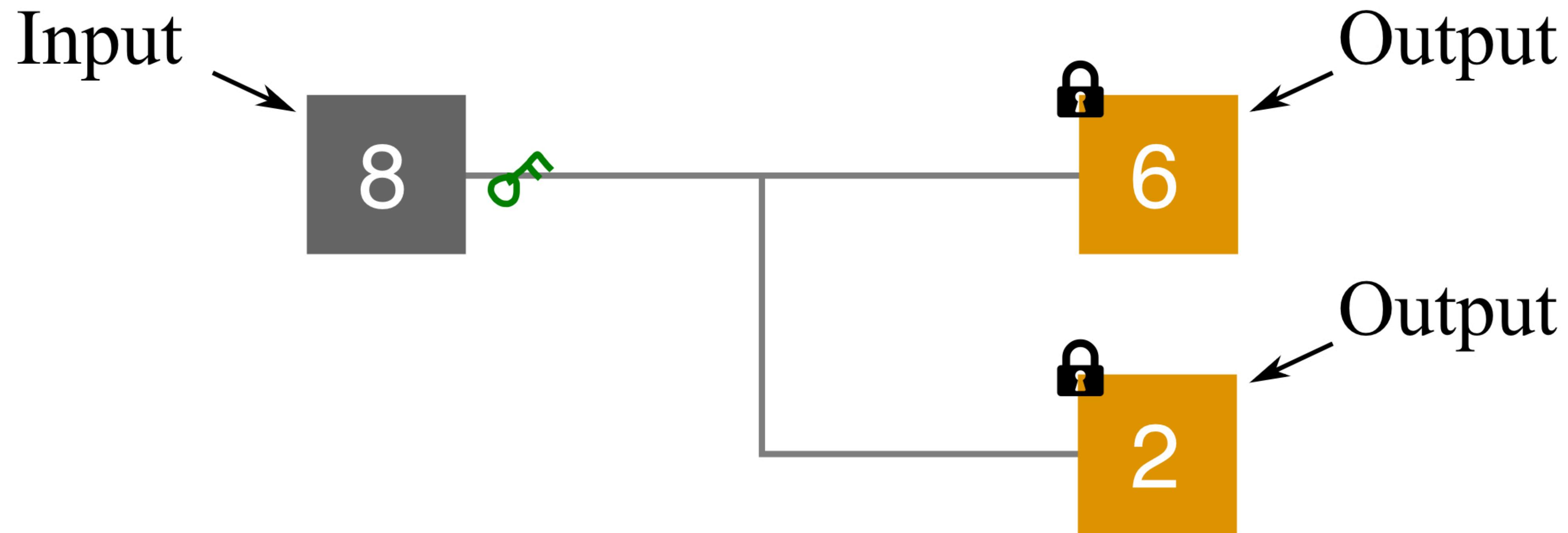
## **Bitcoin Block Internals**

- Metadata (Block Header)
- A collection of transactions

# What's a transaction?

A record of spent Bitcoins  
and newly spendable Bitcoins

# Bitcoin Transaction



# Bitcoin Transaction

- Inputs are spent Bitcoins

# Bitcoin Transaction

- Inputs are spent Bitcoins
- Outputs are newly spendable Bitcoins

# Bitcoin Transaction

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

# Bitcoin Transaction

## Real world example

[1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289](https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289)

DETAILS



#0 6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f9 8.84354951 BTC  
4092b:5



#0 bc1q0qm3wh4g77htmnymkdlzyflupjg5chvcdwju97 0.00300014 BTC

#1 bc1q7cyrfmck2ffu2ud3rn5l5a8yv6f0chkp0zpfemf 8.84018937 BTC

1 CONFIRMATION 8.84318951 BTC

# Bitcoin Transaction

## Real world example

[1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289](https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289)

DETAILS



#0 6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f9  
4092b:5 8.84354951 BTC



#0 bc1q0qm3wh4g77htmnymkdlzyflupjg5chvcdwju97 0.00300014 BTC

#1 bc1q7cyrfmck2ffu2ud3rn5l5a8yv6f0chkp0zpfemf 8.84018937 BTC

1 CONFIRMATION 8.84318951 BTC



# 8.84354951

# Bitcoin Transaction

## Real world example

[1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289](https://blockstream.info/tx/1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289)

DETAILS



#0 6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f9  
4092b:5 8.84354951 BTC



#0 bc1q0qm3wh4g77htmnymkdlzyflupjg5chvcdwju97 0.00300014 BTC

#1 bc1q7cyrfmck2ffu2ud3rn5l5a8yv6f0chkp0zpfemf 8.84018937 BTC

1 CONFIRMATION



8.84354951

8.84318951

# Bitcoin Transaction

Real world example

$$\begin{array}{r} 8.84354951 \leftarrow \text{Input} \\ - 8.84318951 \leftarrow \text{Output} \\ \hline 0.00036000 \end{array}$$

# Bitcoin Transaction

## Real world example

$$\begin{array}{r} 8.84354951 \\ - 8.84318951 \\ \hline 0.00036000 \end{array} \leftarrow \text{fee}$$

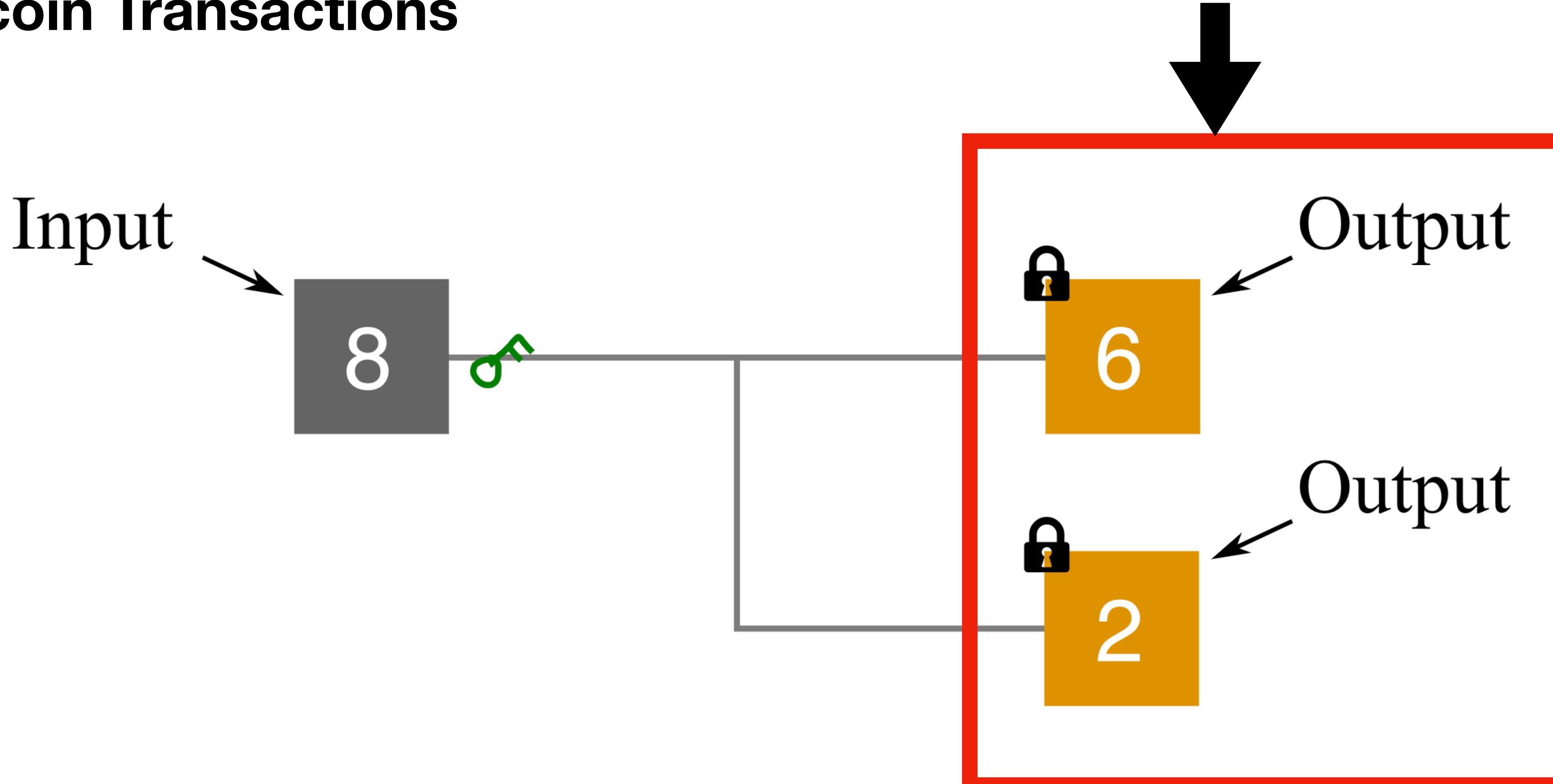
# Vocabulary

## Bitcoin Transactions

- TXO - An Output of a TX (transaction)

# Vocabulary Bitcoin Transactions

# TXOs



# Vocabulary

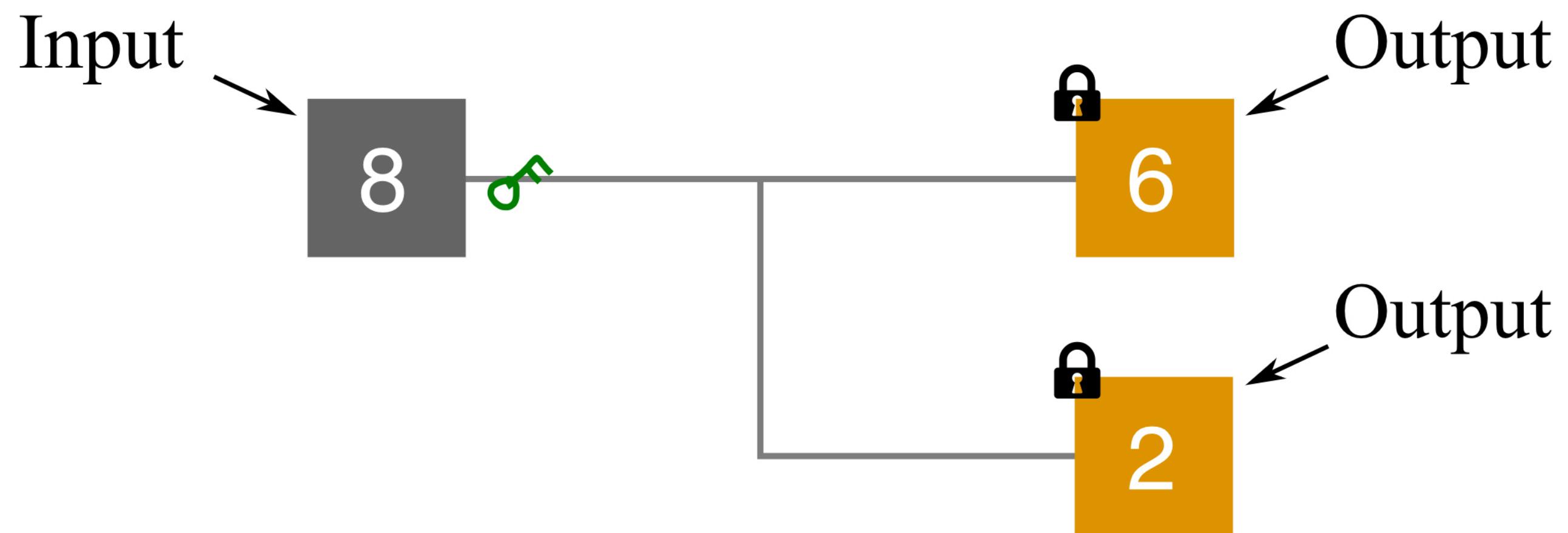
## Bitcoin Transactions

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

# Vocabulary

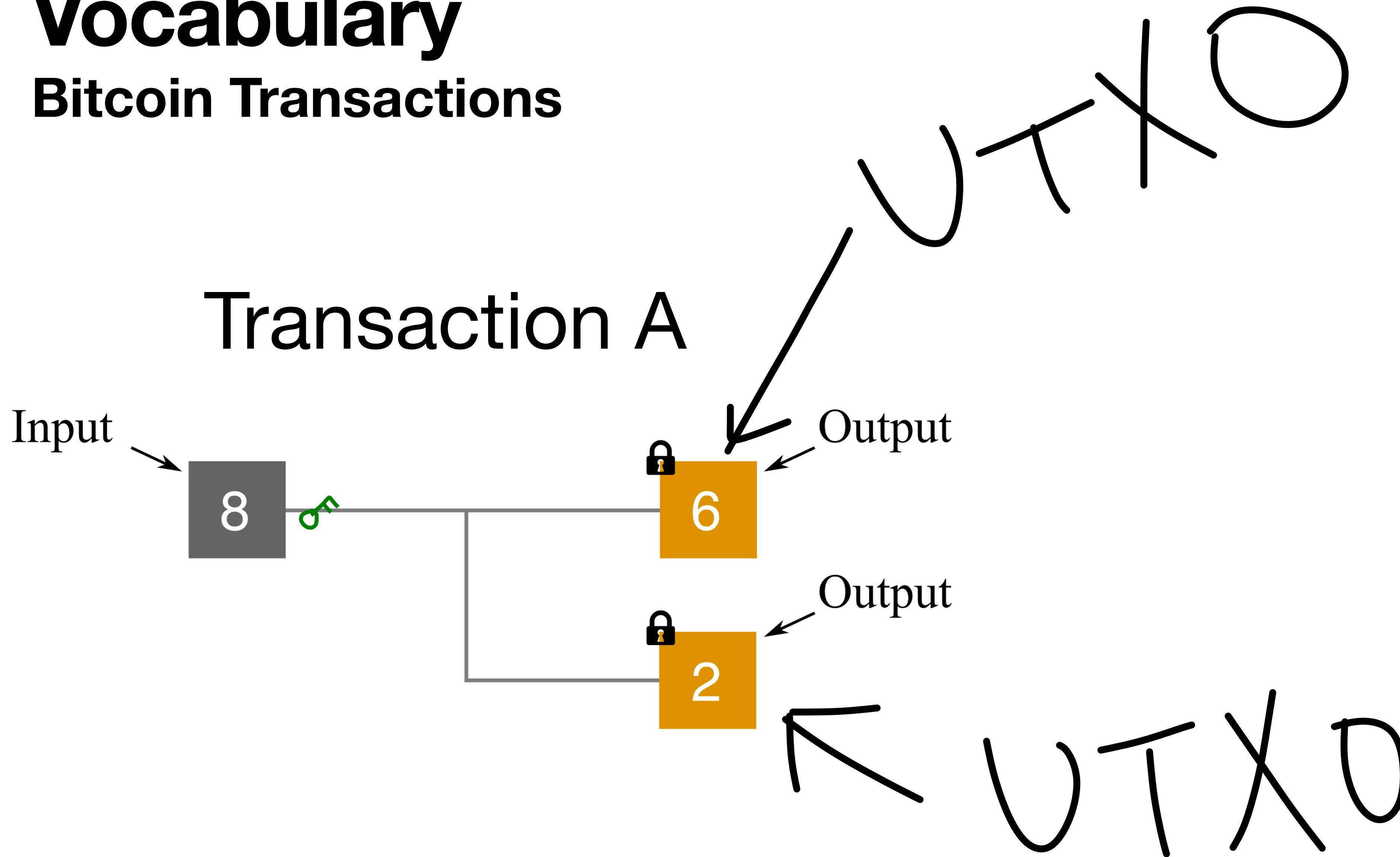
## Bitcoin Transactions

### Transaction A



# Vocabulary

## Bitcoin Transactions



# Vocabulary

## Bitcoin Transactions

#0	bc1q0qm3wh4g77htmnymkdlzyflupjg5chvcdwju97	0.00300014 BTC
TYPE	V0_P2WPKH	
SCRIPTPUBKEY (ASM)	OP_0 OP_PUSHBYTES_20 7837175ea8f7aeb dcc9bb37e2227fc0c914c5d98	
SCRIPTPUBKEY (HEX)	00147837175ea8f7aebdcc9bb37e2227fc0c 914c5d98	
SPENDING TX	Unspent	

# Vocabulary

## Bitcoin Transactions

Details

1FWQiwK27EnGXb6BiBMRLJvunJQZZPMcGd

0.45438206 BTC



bc1qemrfu49savh9cja9qq0hl0u... y0q4xe9n

0.09415525 BTC



0.54853731 BTC

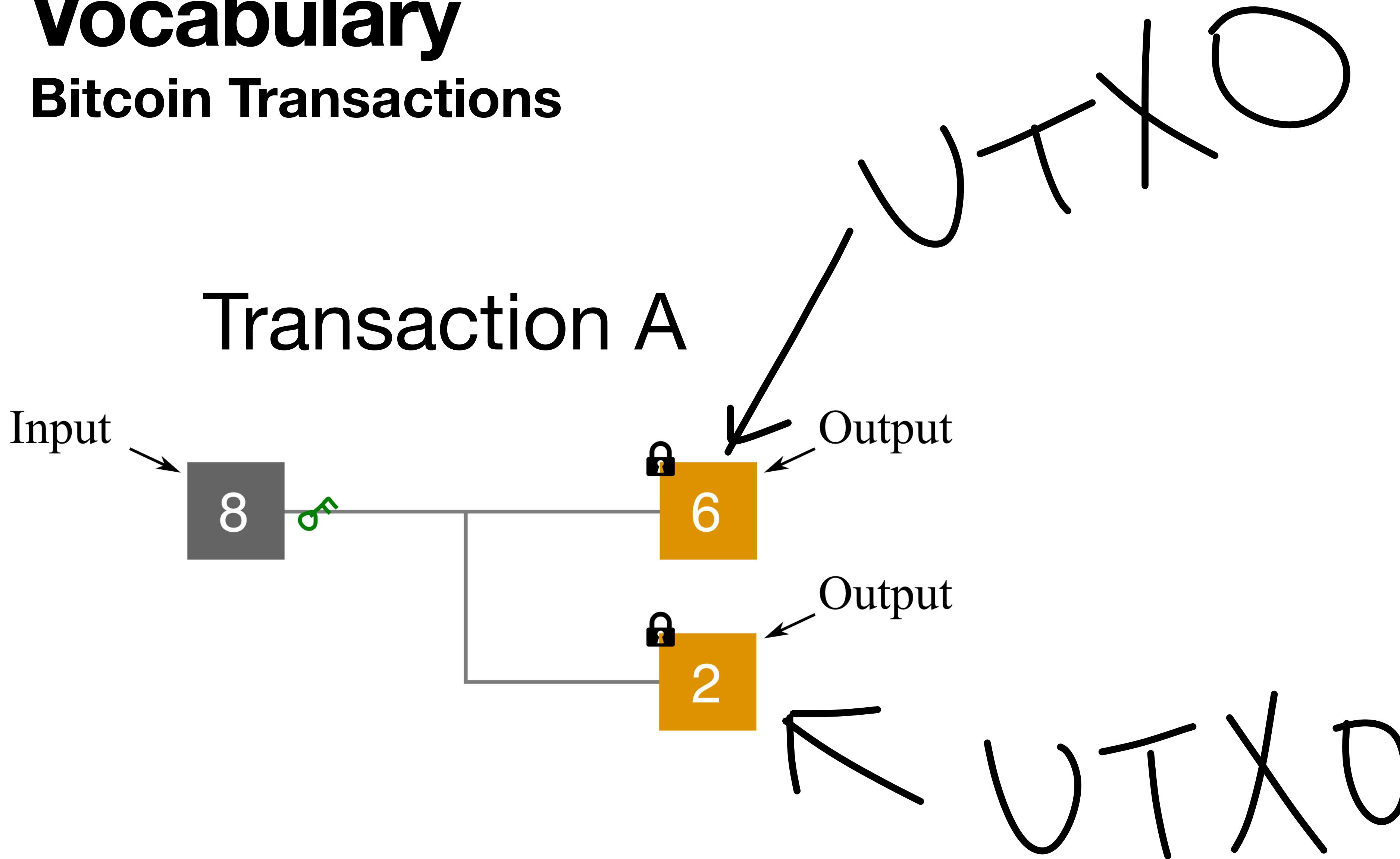
# Vocabulary

## Bitcoin Transactions

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

# Vocabulary

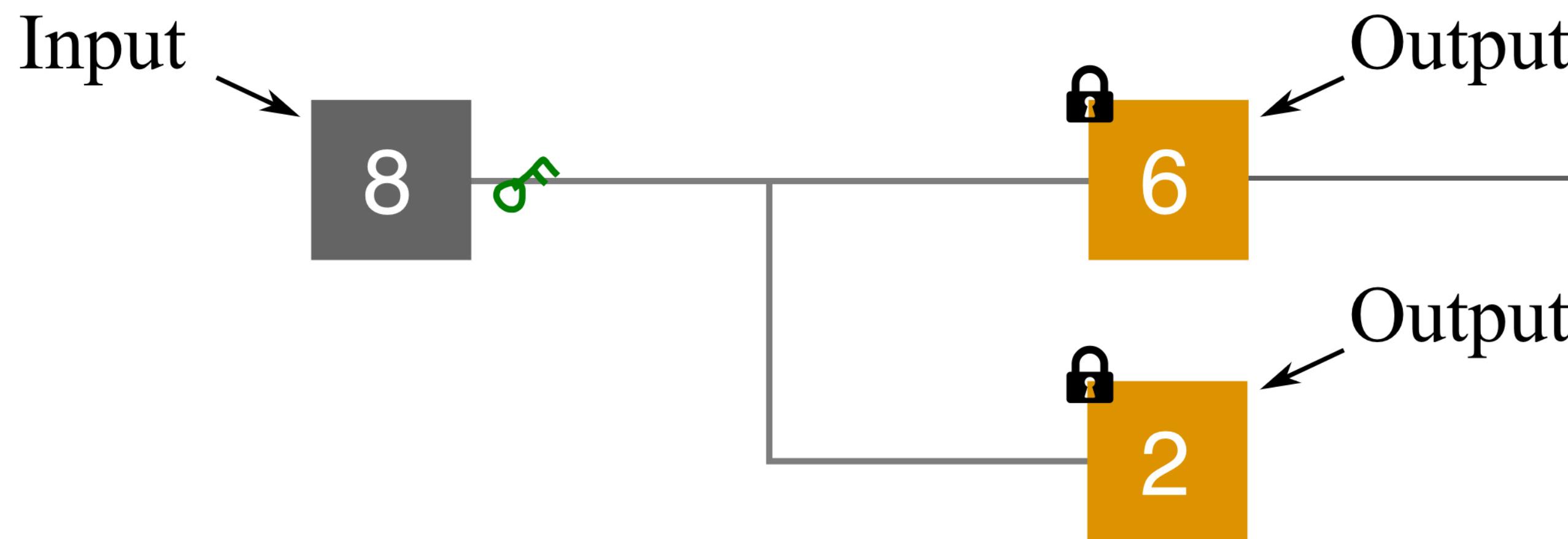
## Bitcoin Transactions



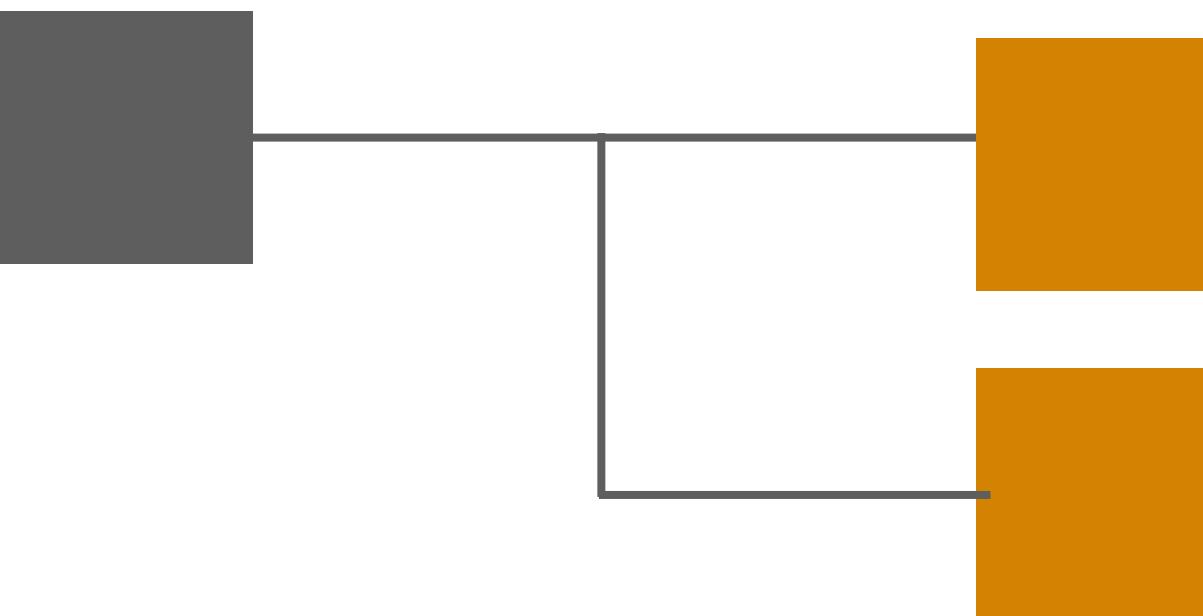
# Vocabulary

## Bitcoin Transactions

Transaction A

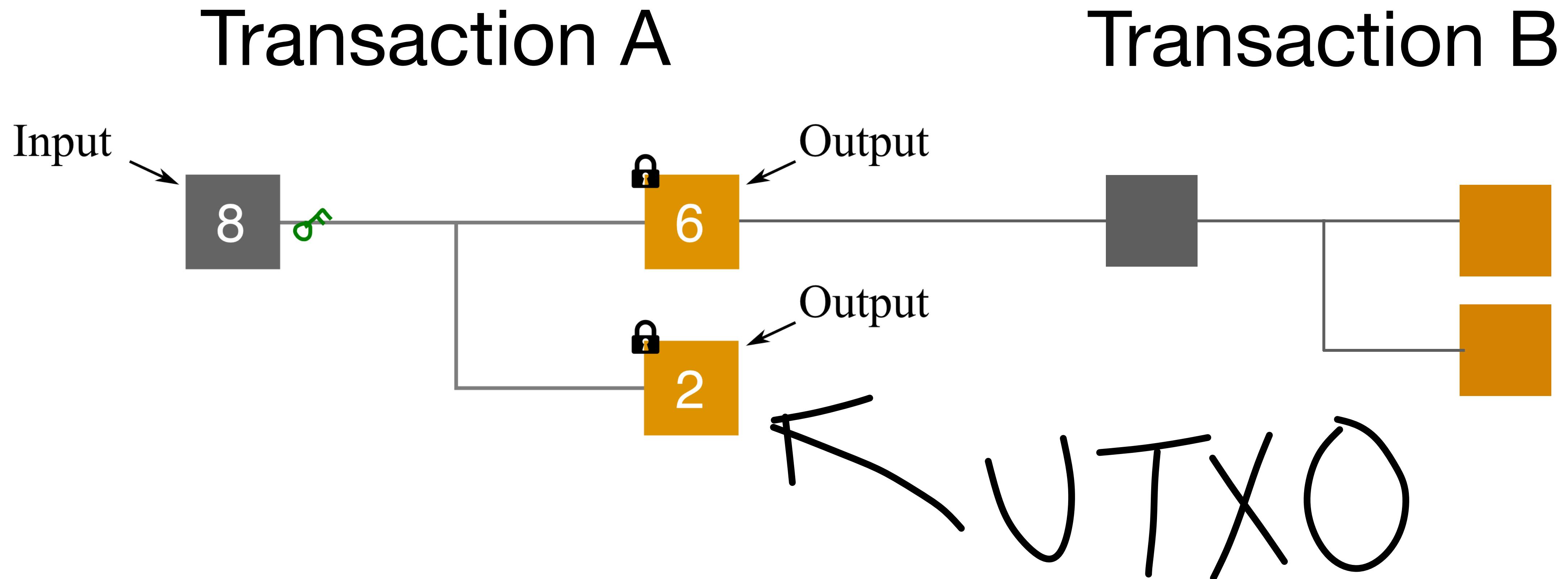


Transaction B



# Vocabulary

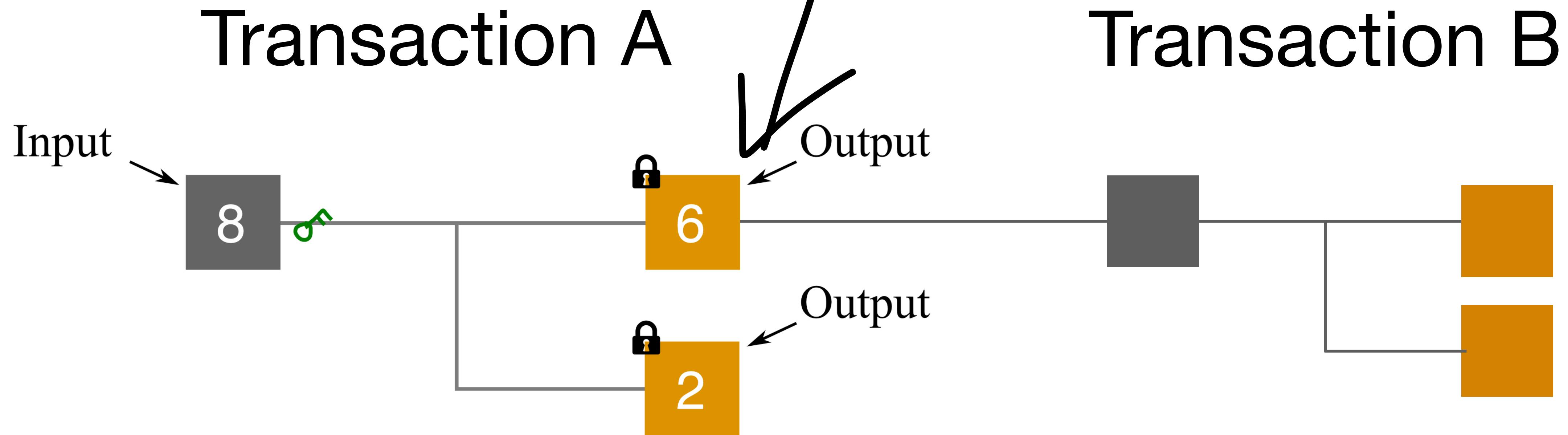
## Bitcoin Transactions



# Vocabulary

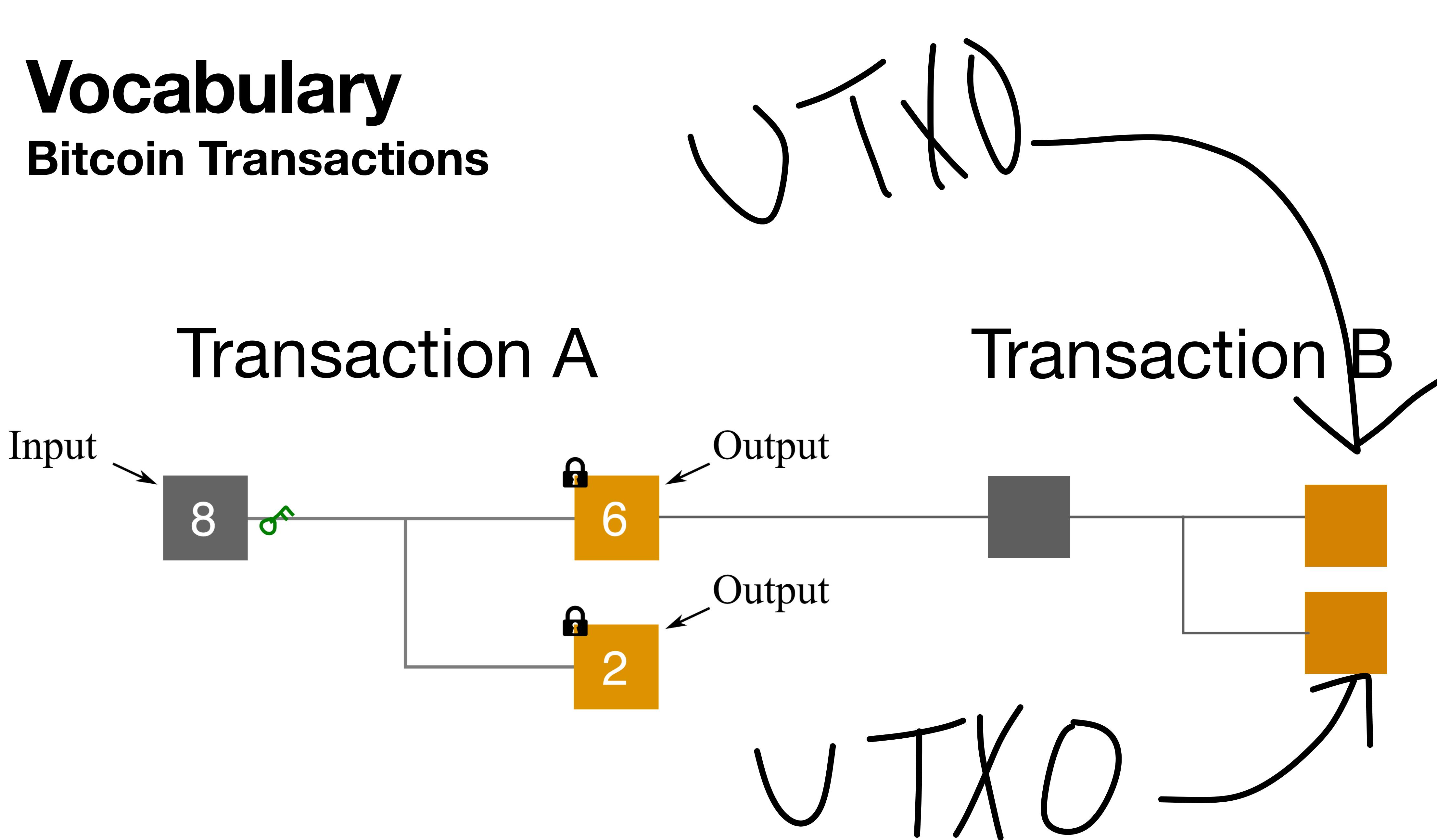
## Bitcoin Transactions

STXO



# Vocabulary

## Bitcoin Transactions



# Whitepaper section 5

## 5. Network

The steps to run the network are as follows:

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

# Whitepaper section 5

## 5. Network

The steps to run the network are as follows:

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

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

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

# Not Already Spent

## Transaction validation

- Inputs must be spending UTXOs not STXOs

# Not Already Spent

## Transaction validation

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

**What's referenced when  
checking for a UTXO?**

# The UTXO set!

# Vocabulary

## Bitcoin Transactions

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

# The UTXO set

All the Bitcoins that are available to be spent



# The UTXO set

All the Bitcoins that are available to be spent



# A single UTXO

What a UTXO is

- Referenced by TXID:Index

# A single UTXO

## What a UTXO is

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

# A single UTXO

## What a UTXO is

The screenshot shows a transaction output highlighted with a red box. The output ID is `1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289`. It contains one input (txid: #0) and two outputs. The first output goes to a wallet address (#0) with 8.84354951 BTC. The second output goes to another wallet address (#1) with 8.84018937 BTC. A blue arrow points from the highlighted output to the spending transaction.

Output	Address	Amount (BTC)
#0	<code>bc1q0qm3wh4g77htmnymkdlzyflupjg5chvcdwju97</code>	0.00300014
#1	<code>bc1q7cyrfmck2ffu2ud3rn5l5a8yv6f0chkp0zpfemf</code>	8.84018937

DETAILS +

1 CONFIRMATION 8.84318951 BTC

# A single UTXO

## What a UTXO is

**1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289**

#0 6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f9 8.84354951 BTC  
4092b:5

# A single UTXO

## What a UTXO is

**1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289**

#0 6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f9  
4092b:5

8.84354951 BTC

# A single UTXO

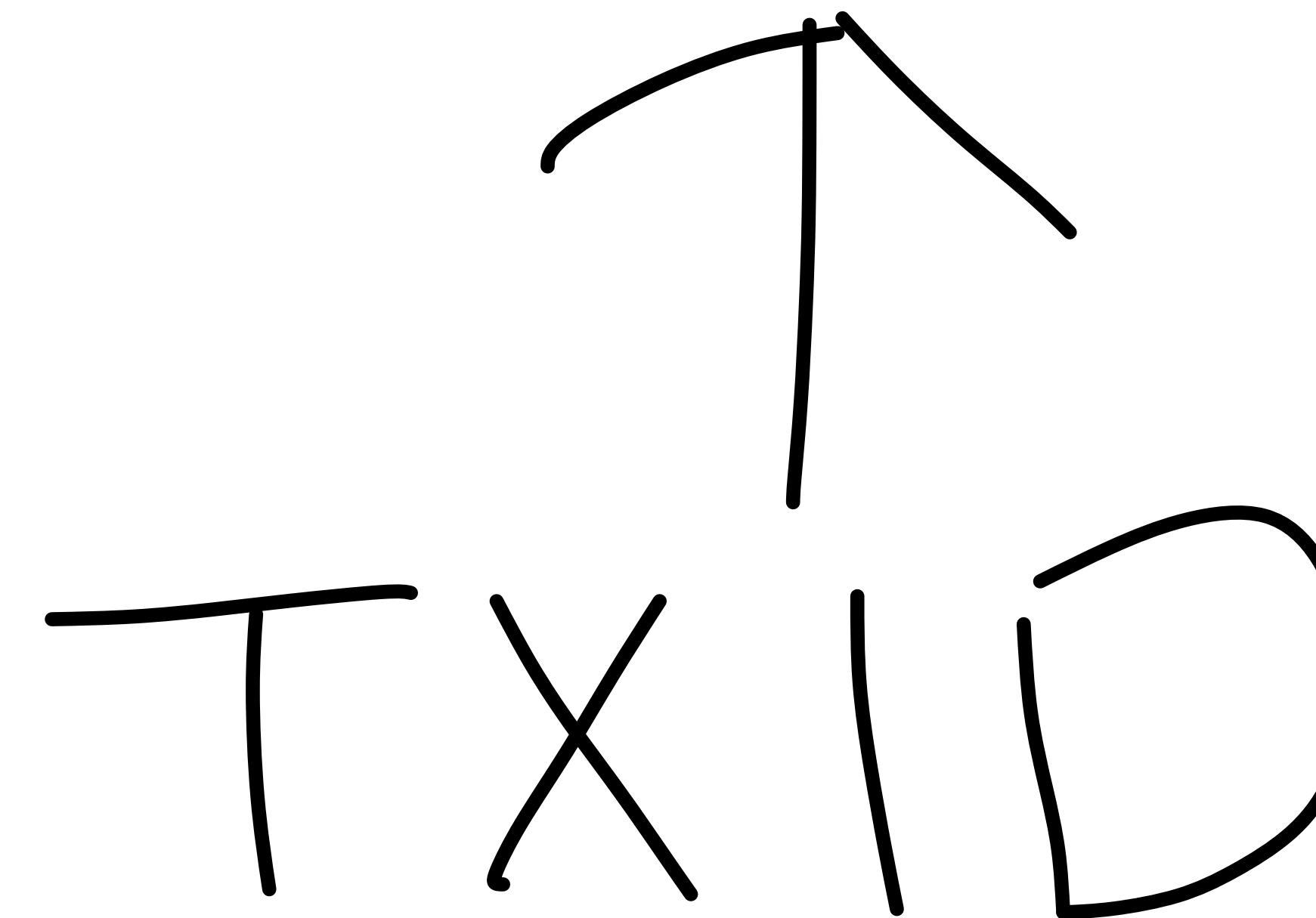
## What a UTXO is

6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f94092b:5

# A single UTXO

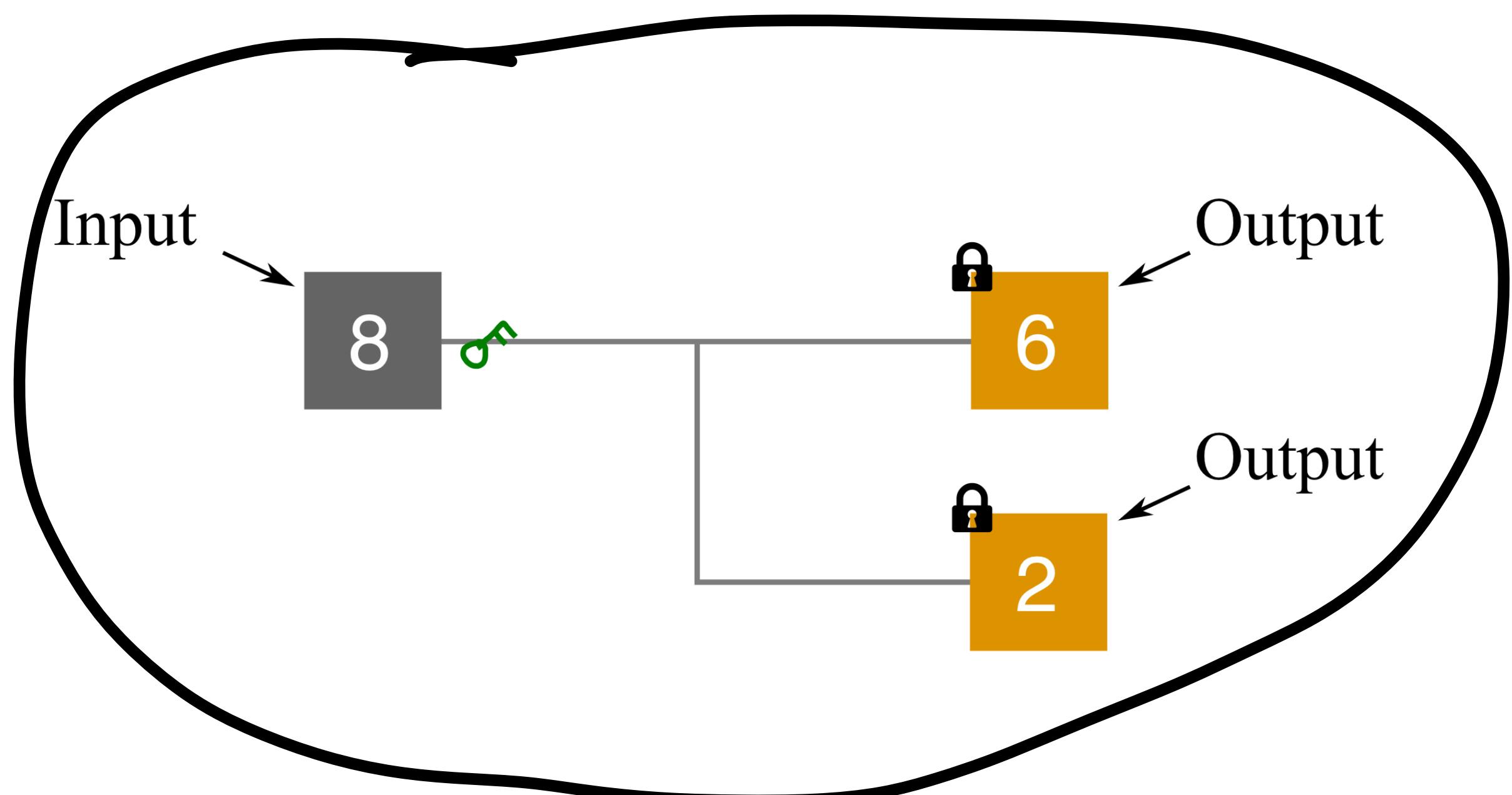
## What a UTXO is

6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f94092b:5



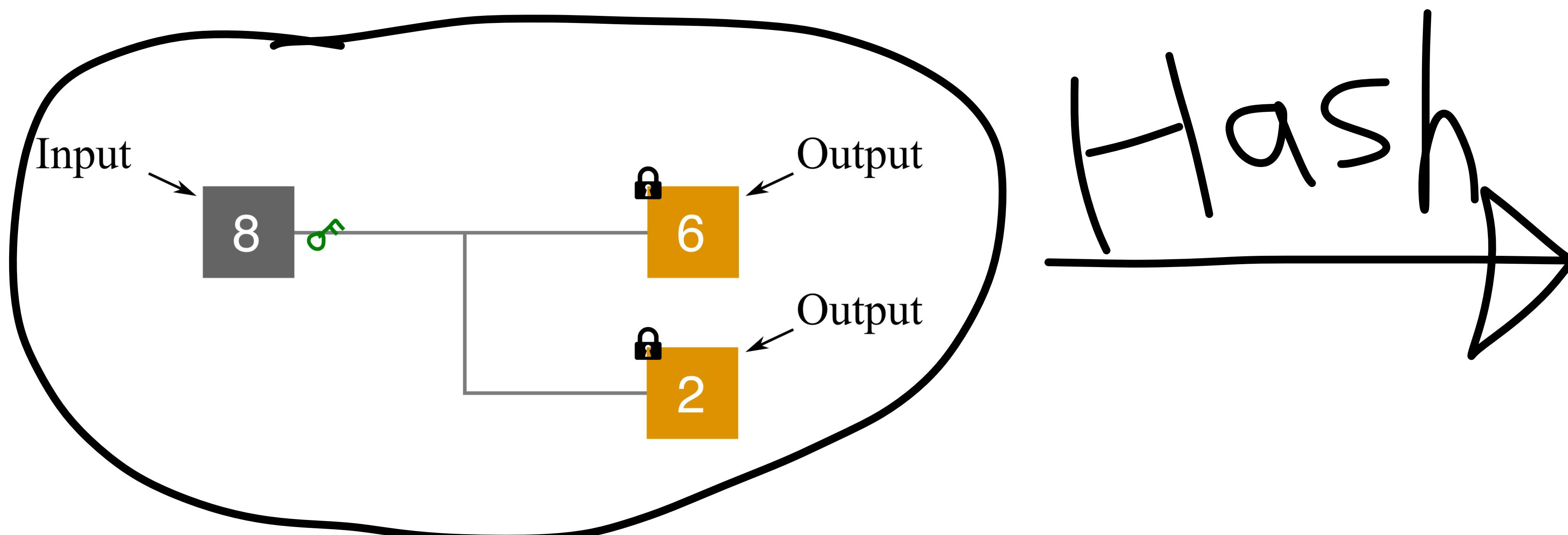
# A single UTXO

What a UTXO is



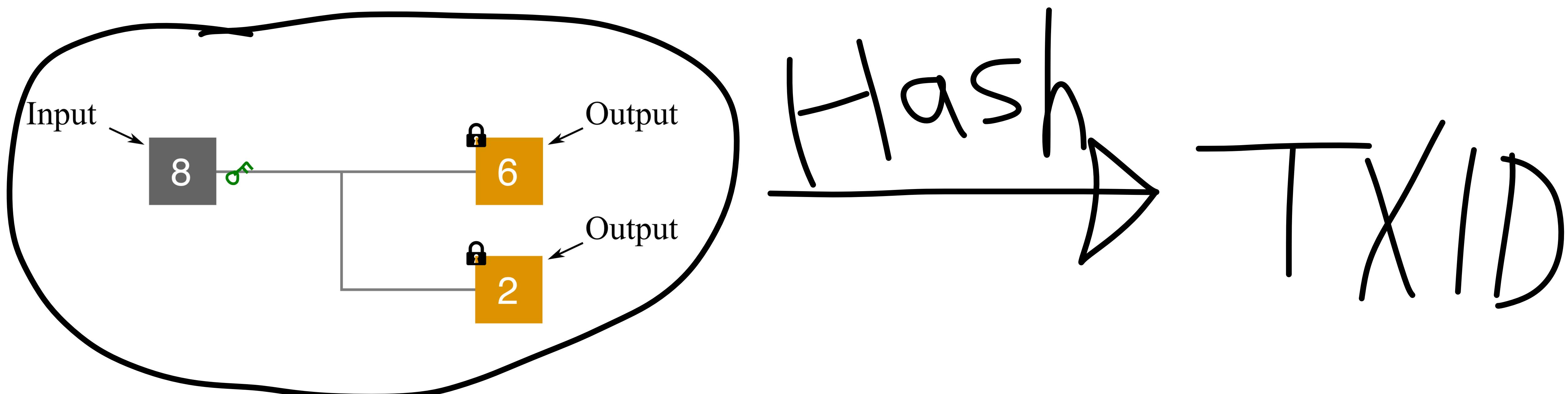
# A single UTXO

What a UTXO is



# A single UTXO

What a UTXO is

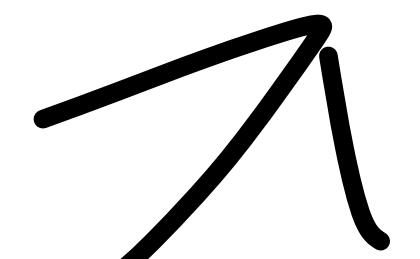


# A single UTXO

What a UTXO is

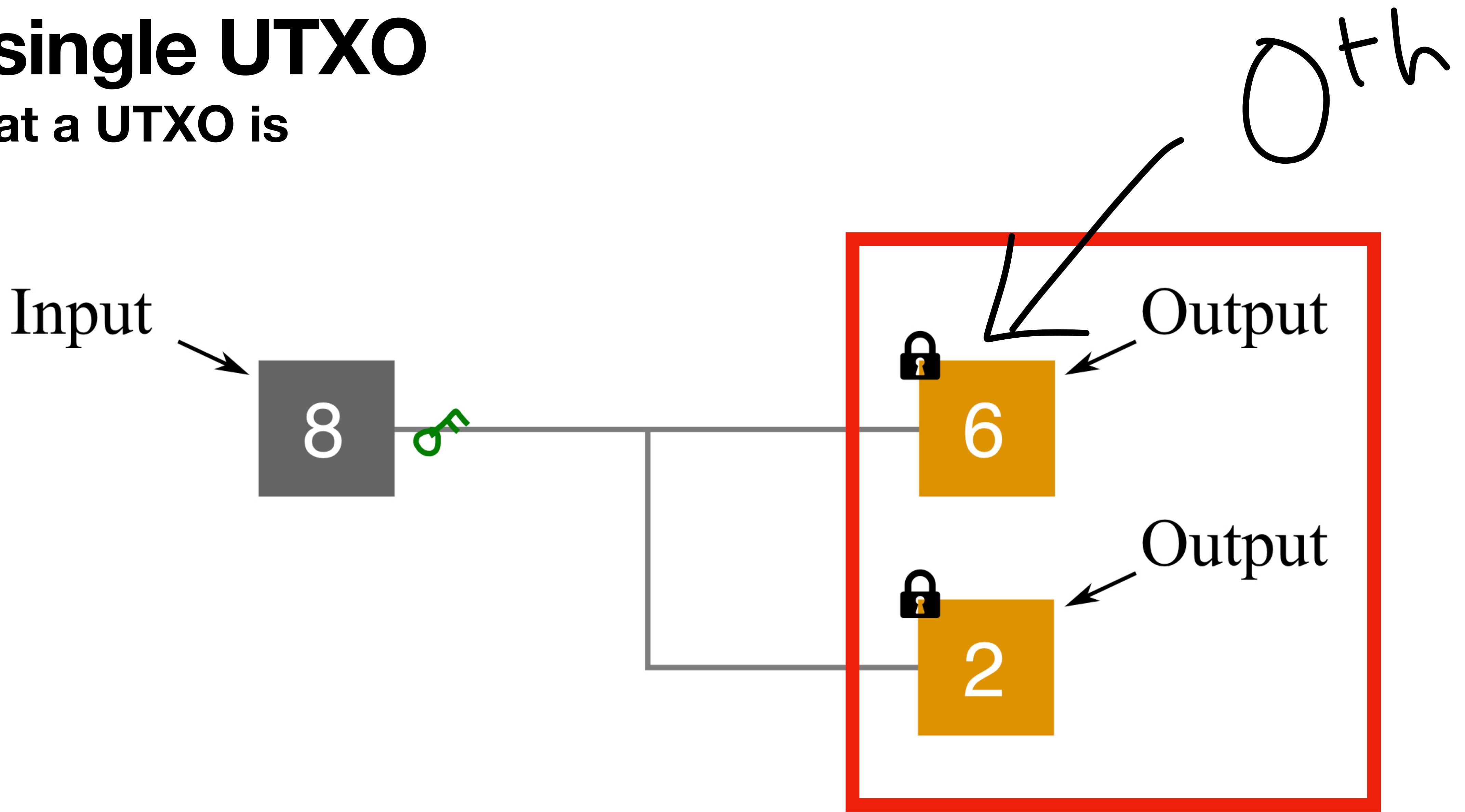
6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f94092b:**5**

Index



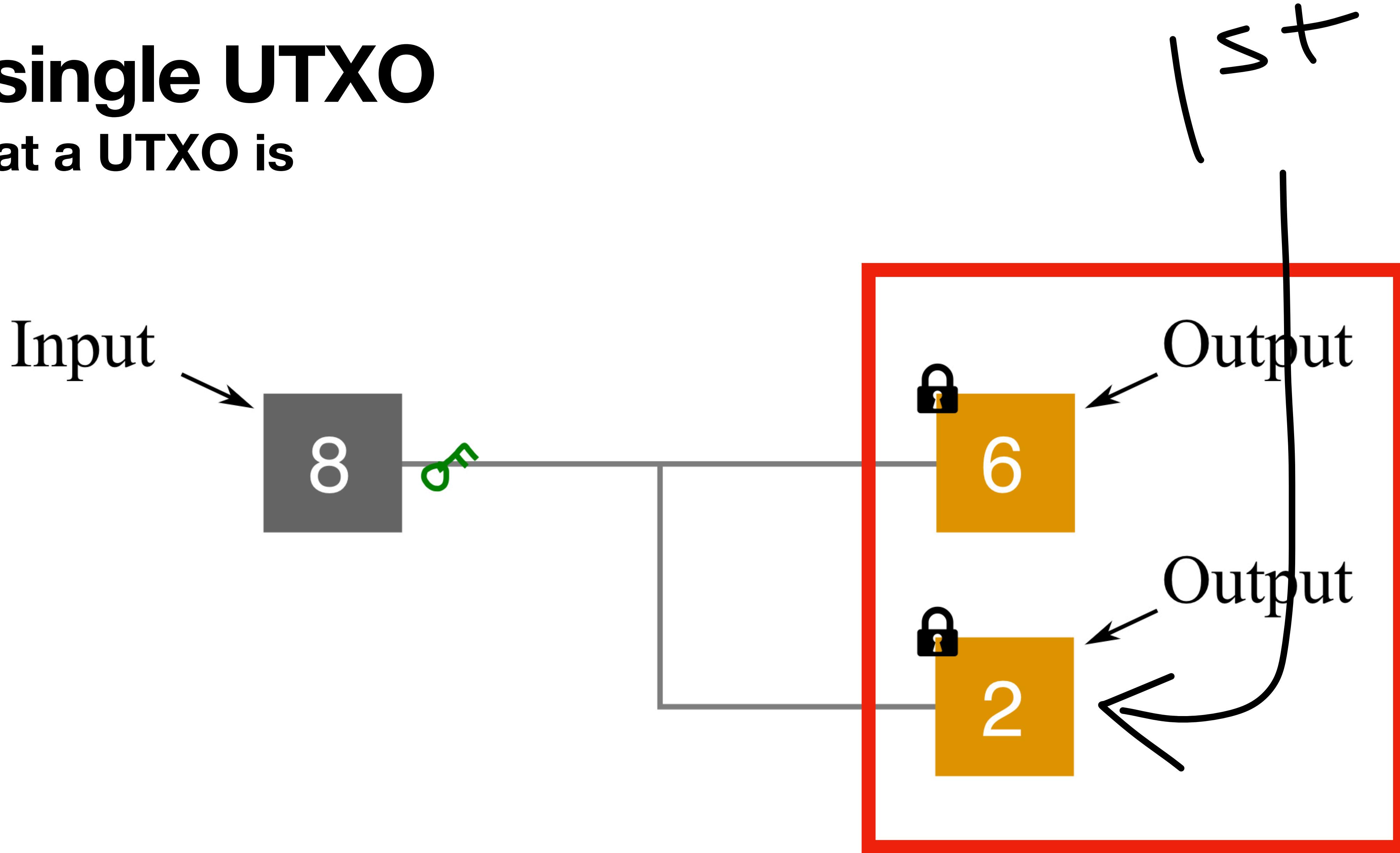
# A single UTXO

What a UTXO is



# A single UTXO

What a UTXO is



# Validating a TX

## Real world example

The screenshot shows a transaction page with a red box highlighting the first input. The transaction ID is `1094680b95cce0f2906fdf7fff6d5f0973254554c28dec95c22c243cbd510289`. There are two outputs:

Output #	Script Hash	Amount (BTC)
#0	<code>6d9a23dec7e2bb6399cdb42741df025c00a89e914c04e1cacf83f1d93f94092b:5</code>	8.84354951 BTC
#1	<code>bc1q7cyrfmck2ffu2ud3rn5l5a8yv6f0chkp0zpemf</code>	8.84018937 BTC

Total amount: 17.68373888 BTC

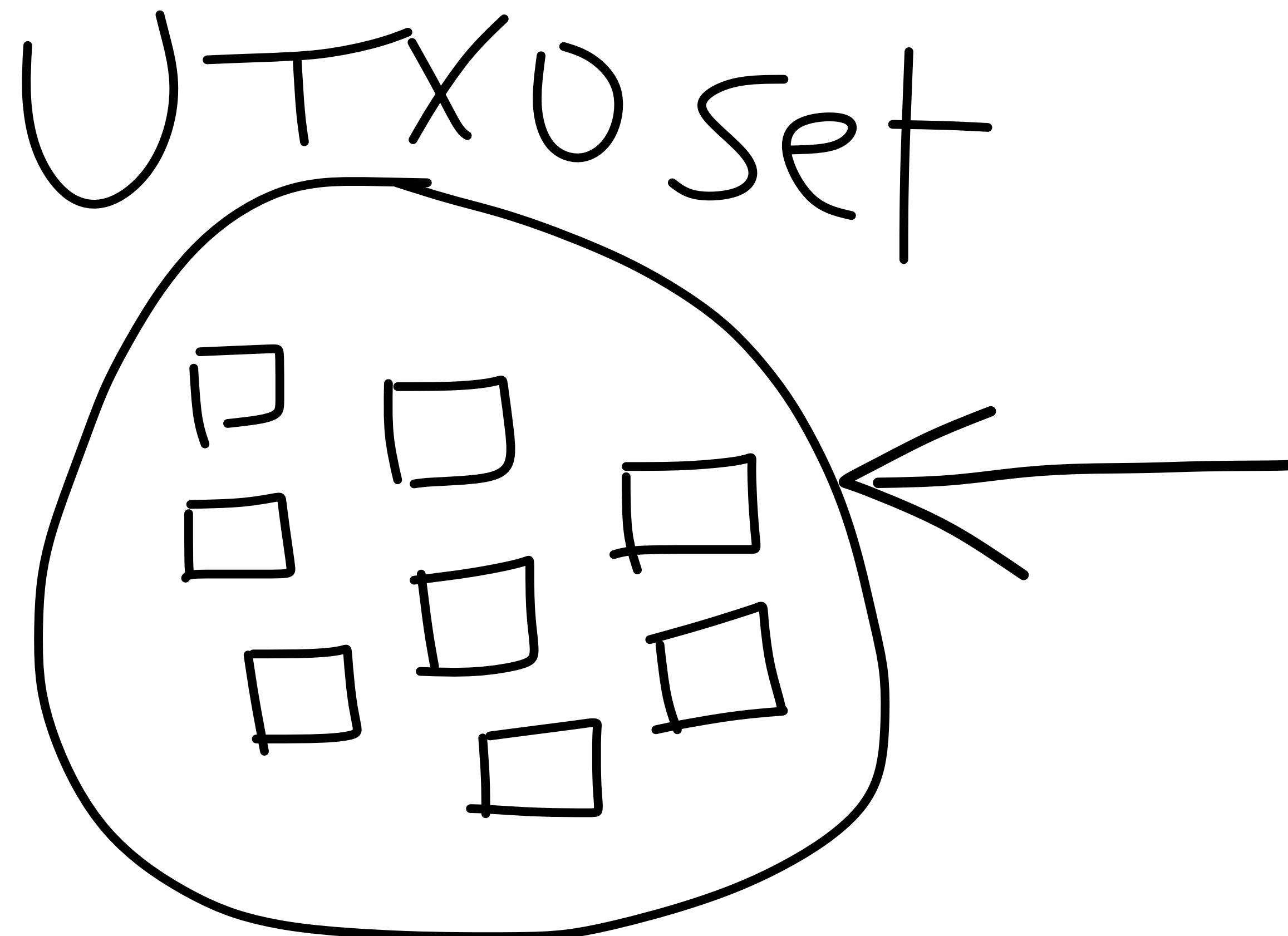
Details button: DETAILS +

Confirmation count: 1 CONFIRMATION

Final amount: 8.84318951 BTC

# Validating a TX

## Real world example



6d9a23dec7e2bb6399cdb  
42741df025c00a89e914c0  
4e1cacf83f1d93f94092b:5

# Validating a TX

Real world example

- Valid if: UTXO exists in the UTXO set

# Validating a TX

## Real world example

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

# **What is Utreexo?**

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



# Utreeexo?

What is it?

- A fancy merkle tree proposed by Tadge Dryja



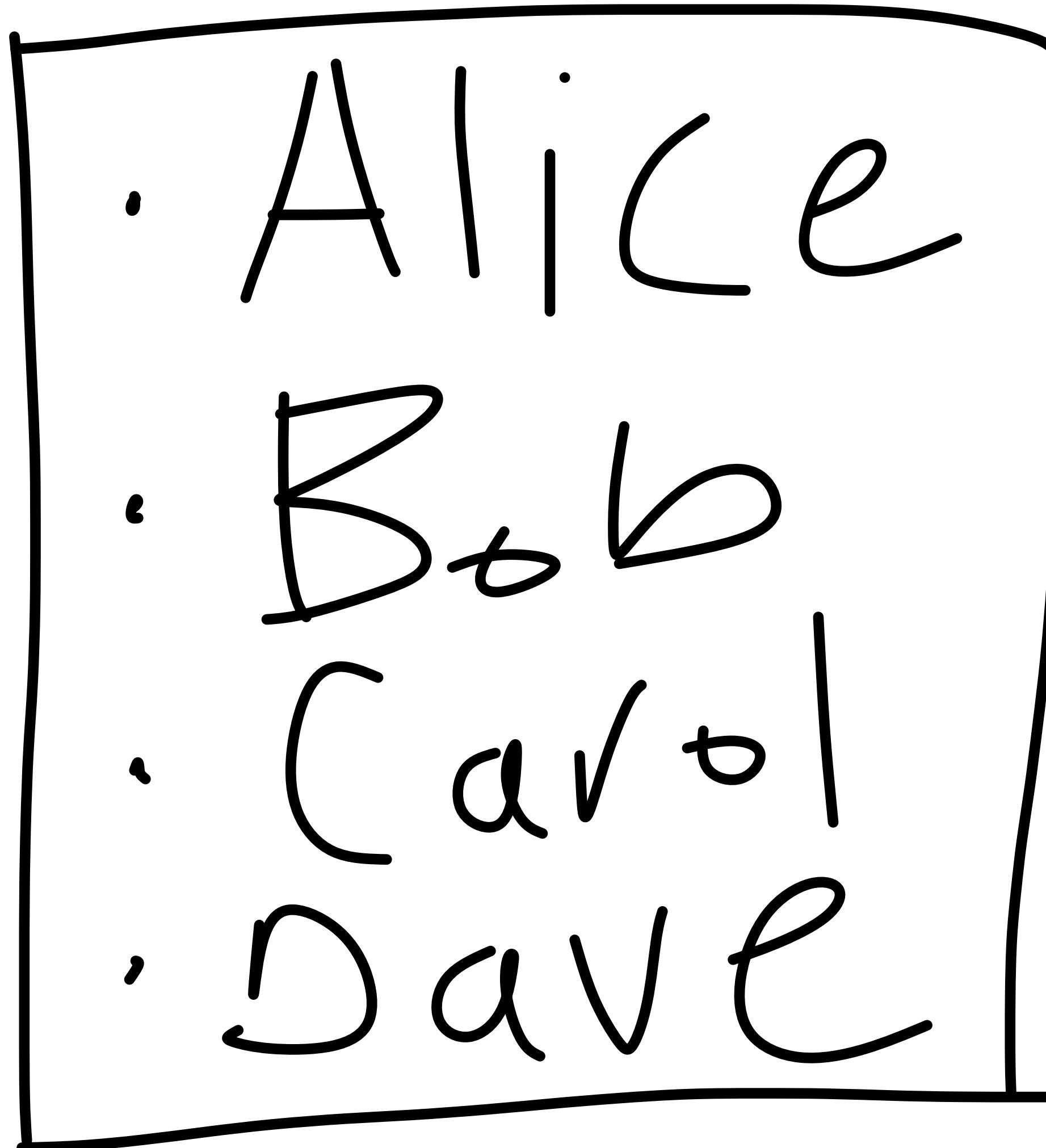
# Merkle Tree

What is it?

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

# Merkle Tree

What is it?



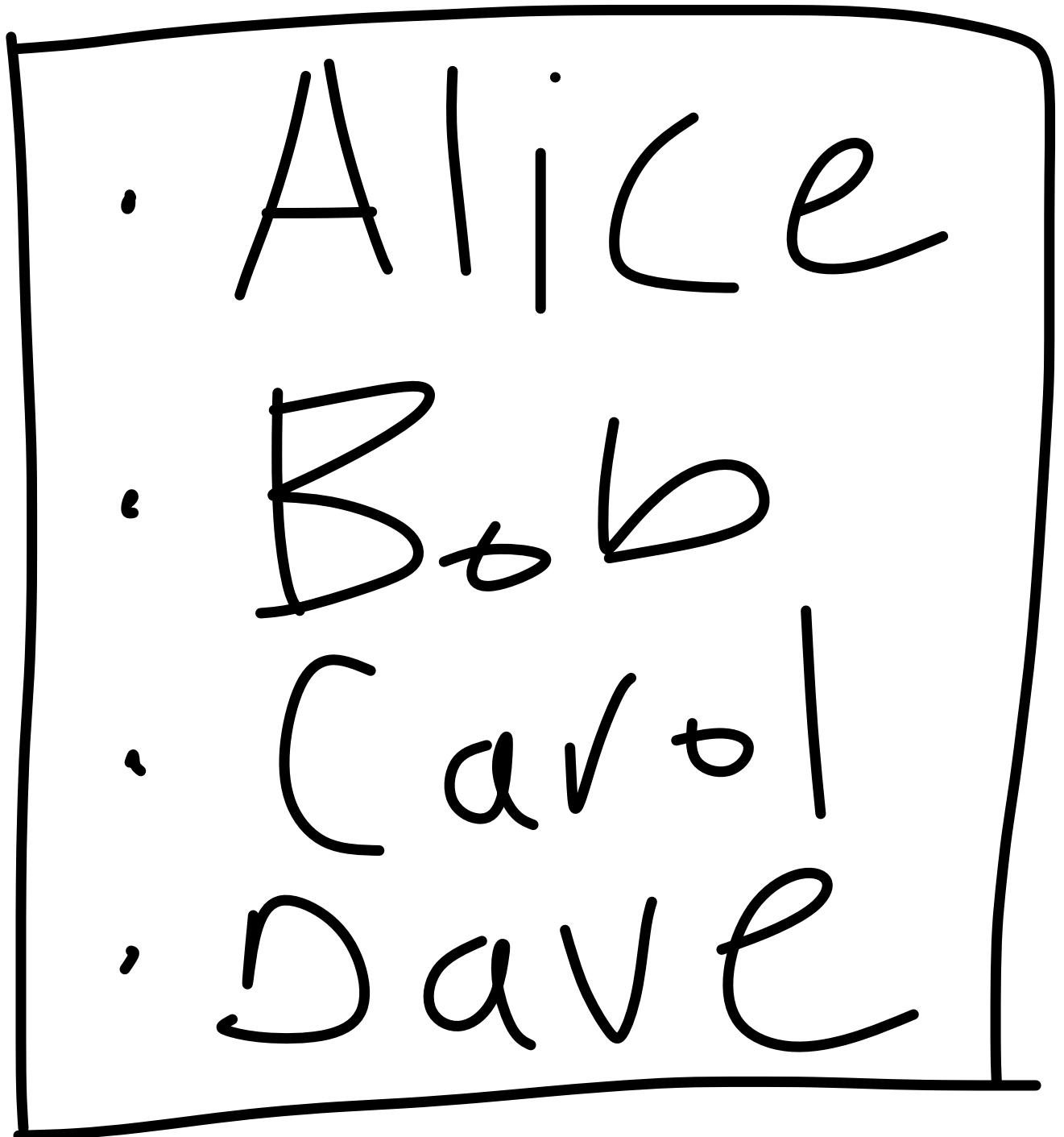
# Merkle Tree

## What is it?

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

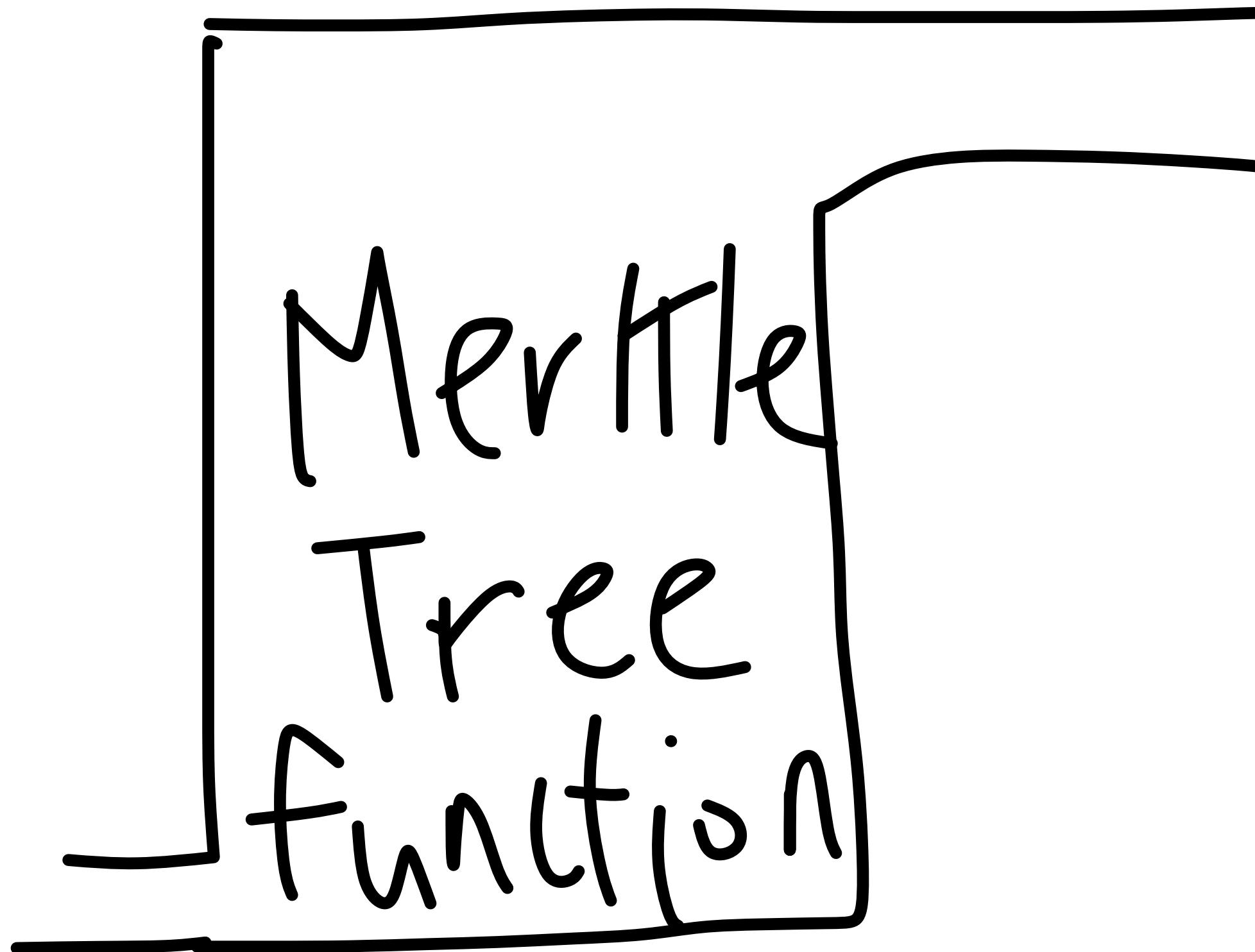
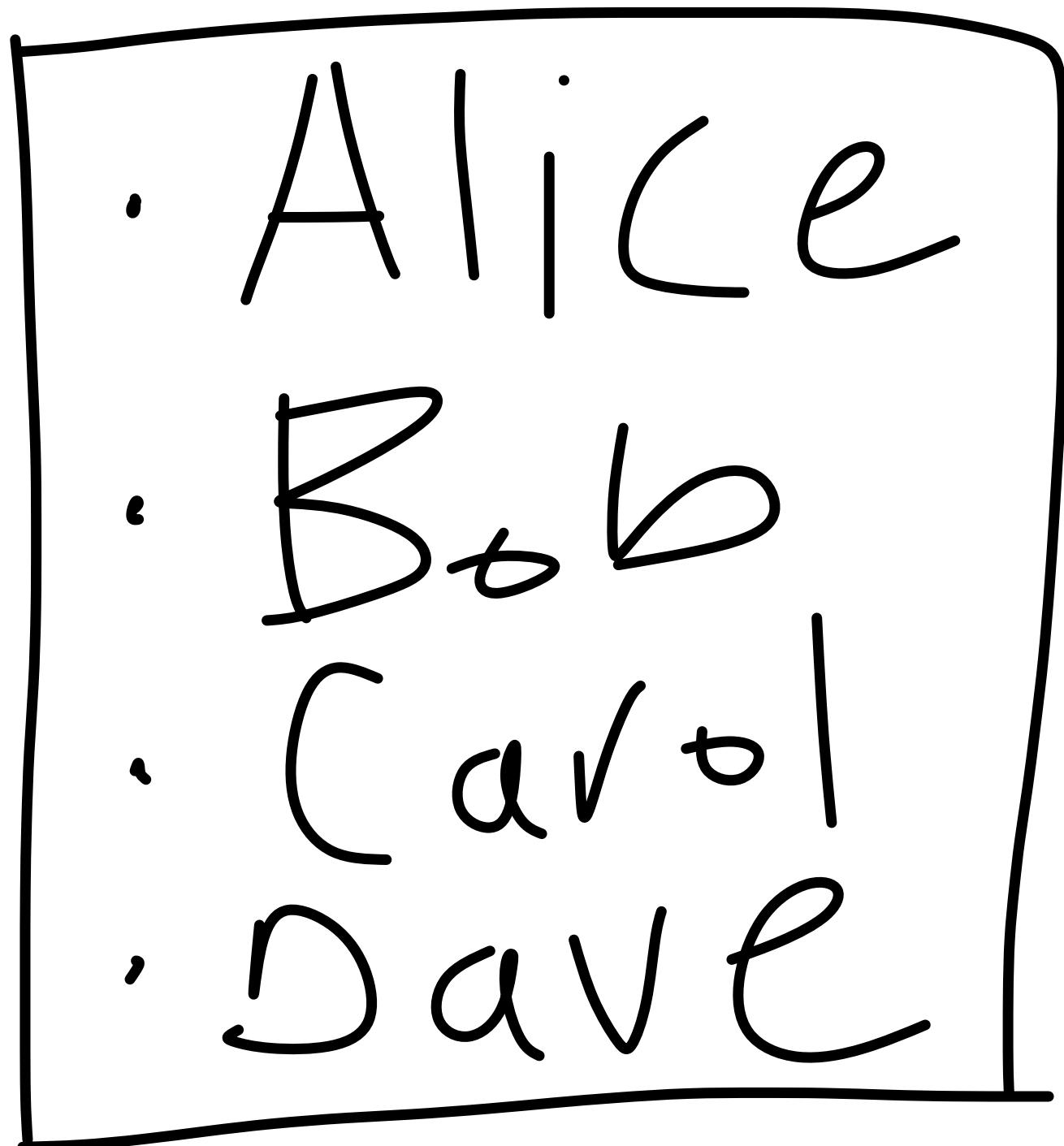
# Merkle Tree

What is it?



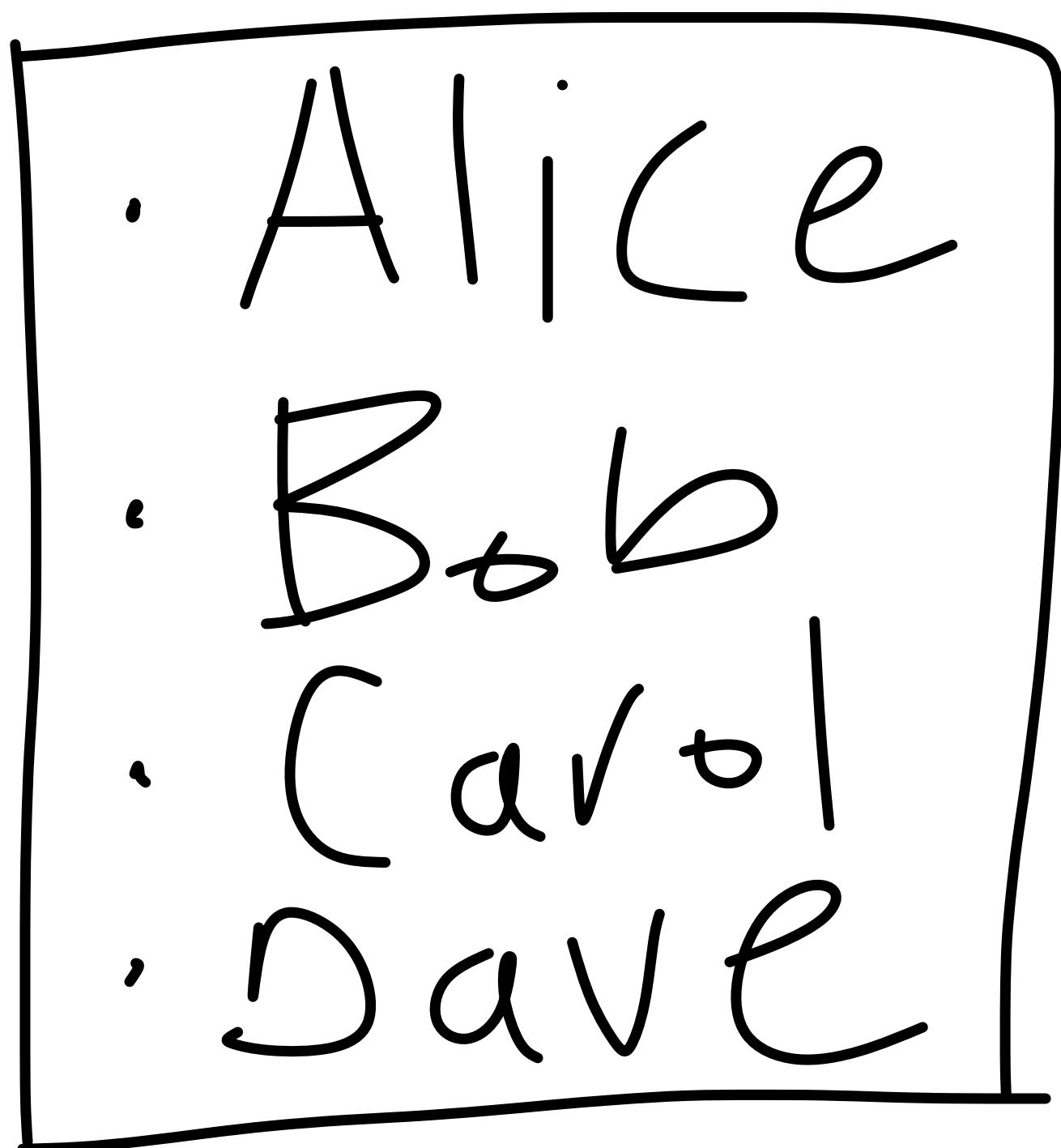
# Merkle Tree

What is it?



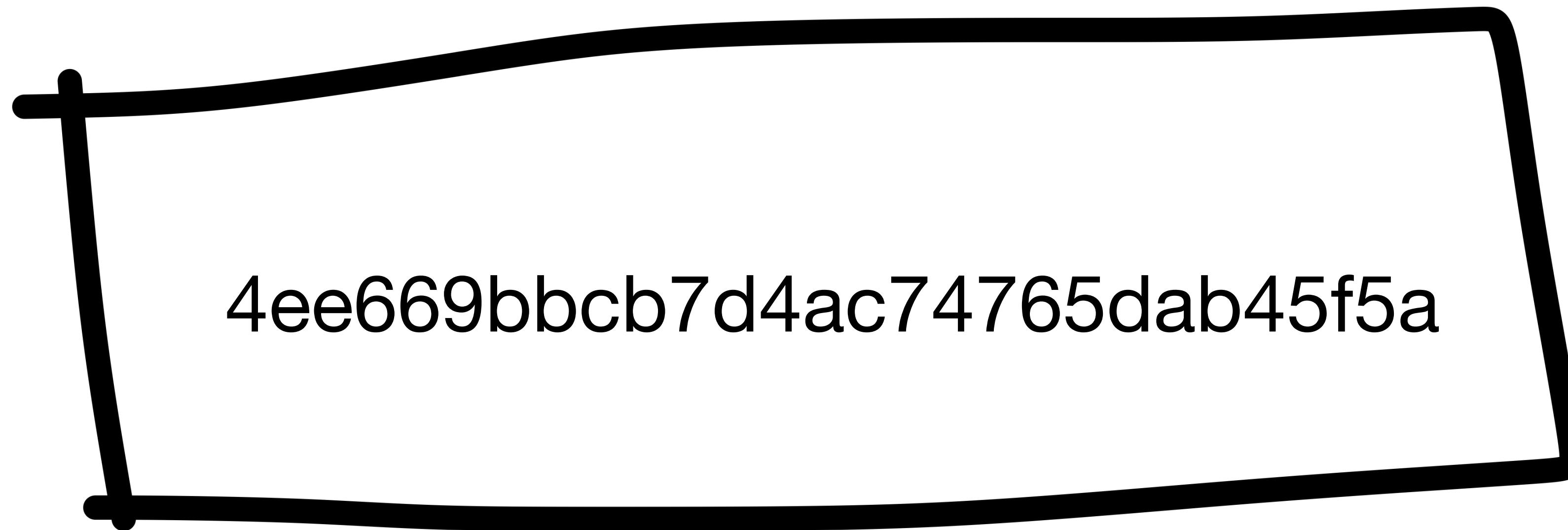
# Merkle Tree

What is it?



# Merkle Tree

What is it?



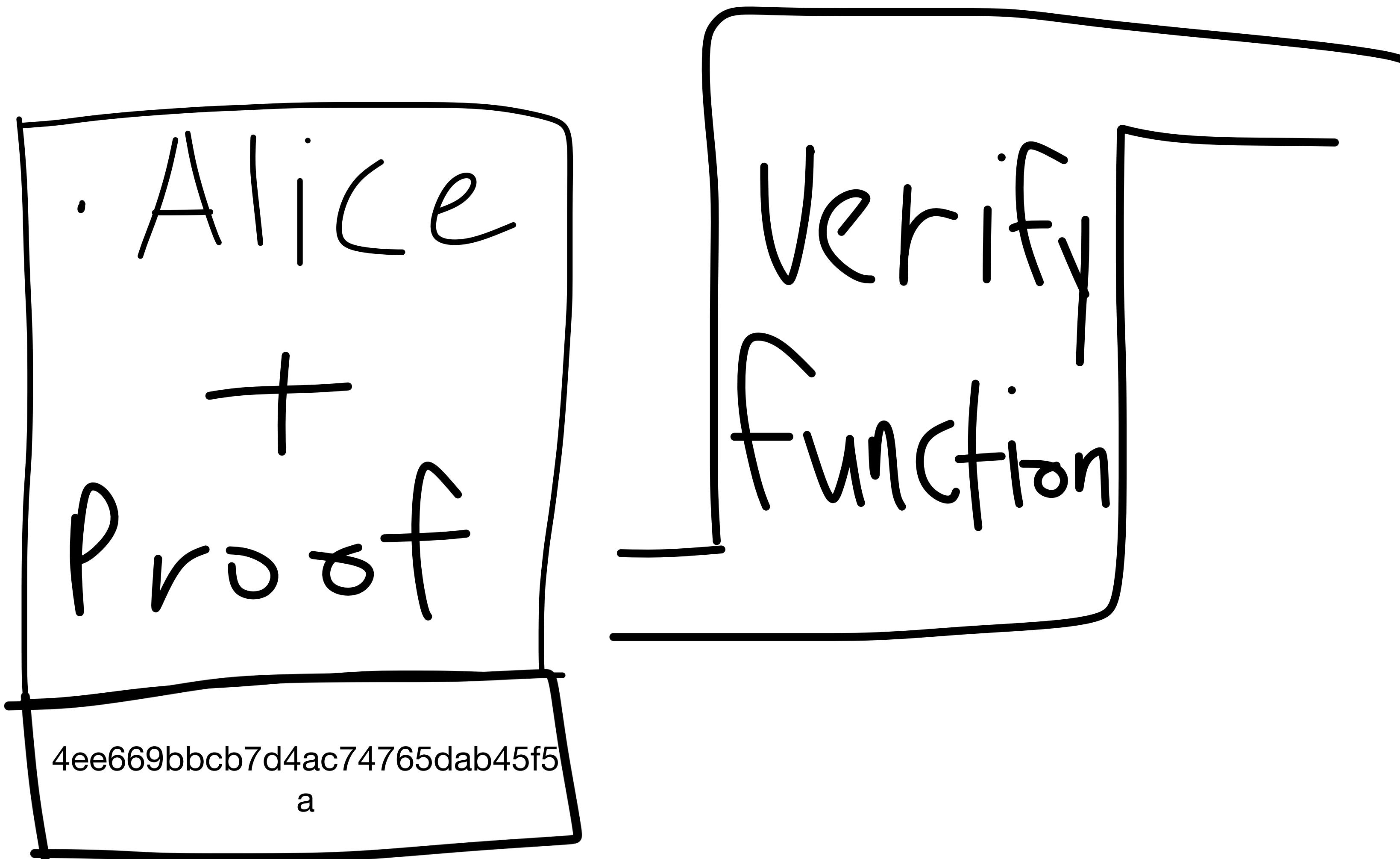
# Merkle Tree

What is it?



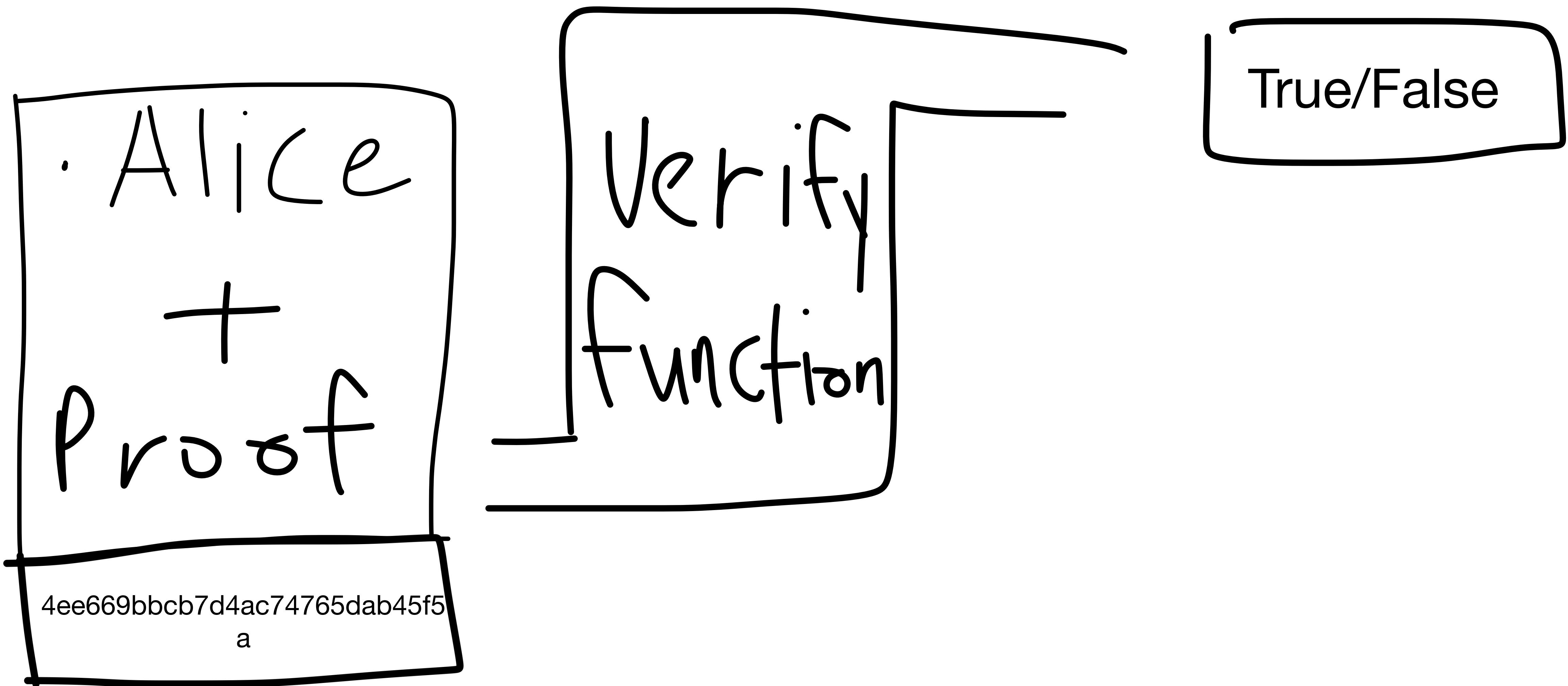
# Merkle Tree

What is it?



# Merkle Tree

What is it?



# Utreeexo?

## What is it?

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



# Pruned nodes?

# What are pruned nodes?

**Small nodes**

- Nodes that only keep the UTXO set

# What are pruned nodes?

## Small nodes

- Nodes that only keep the UTXO set
- Historical block data is removed

# What are pruned nodes?

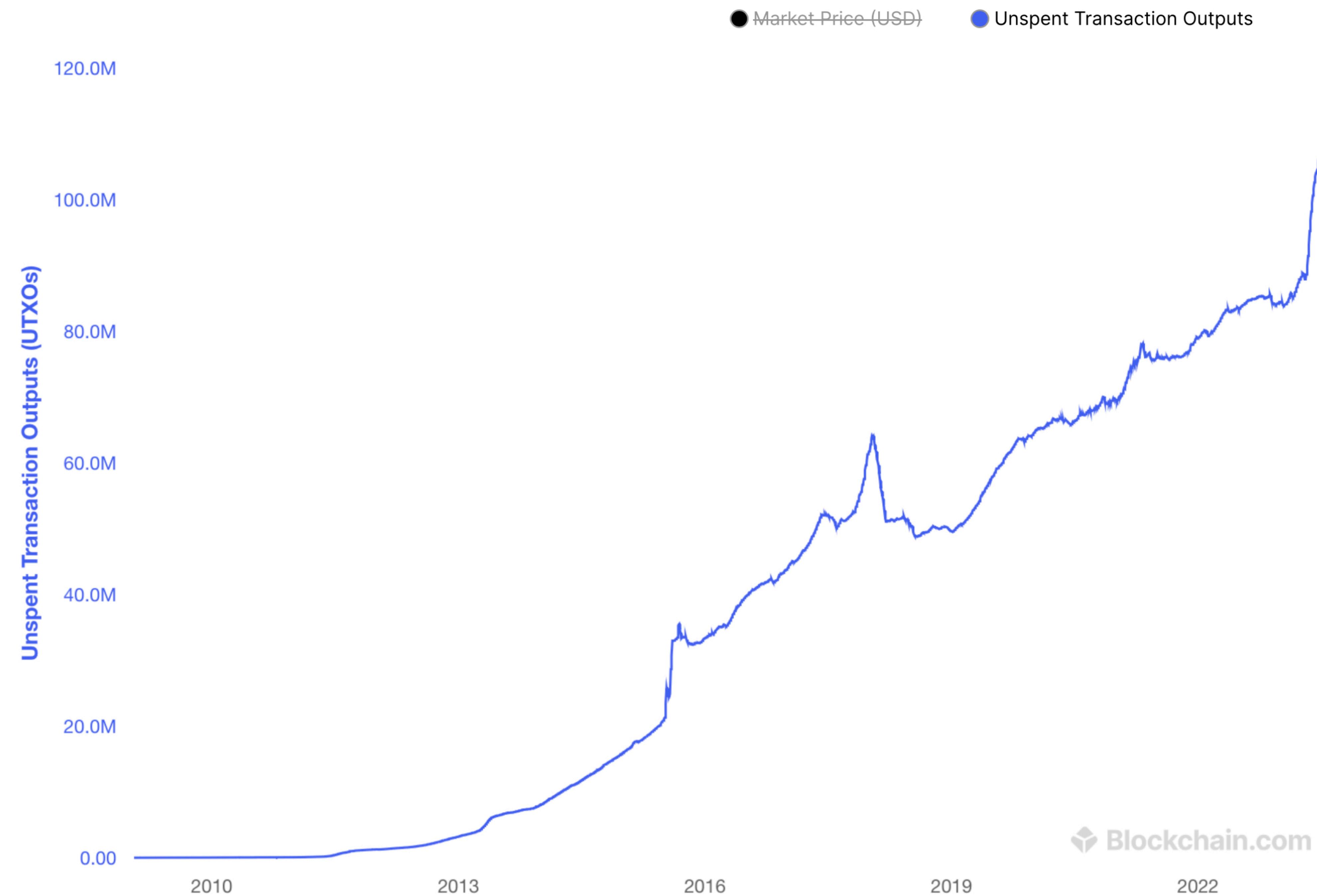
## Small nodes

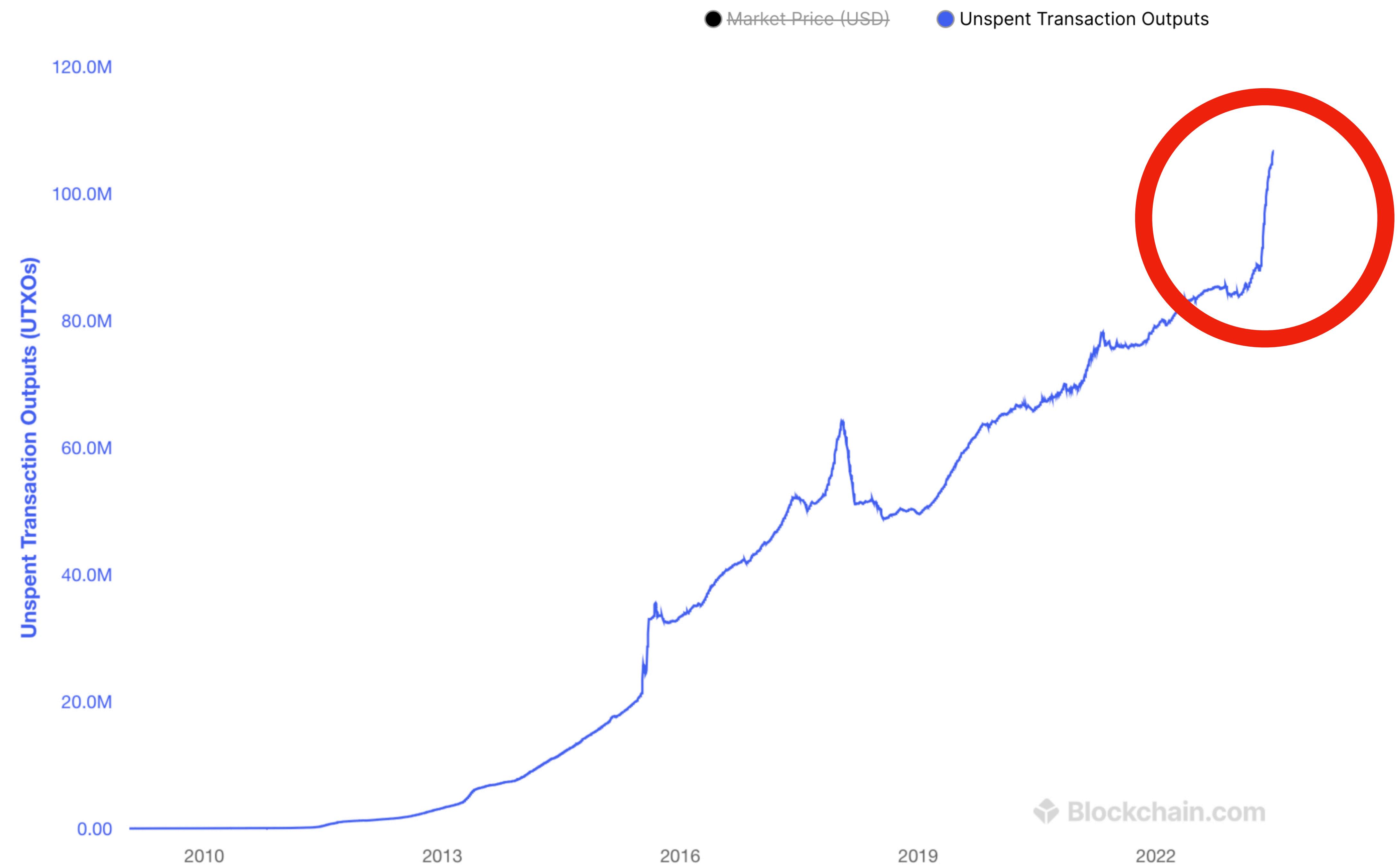
- Nodes that only keep the UTXO set
- Historical block data is removed
- Total size around 6GBs and growing

# The UTXO set

Why merkelize it?

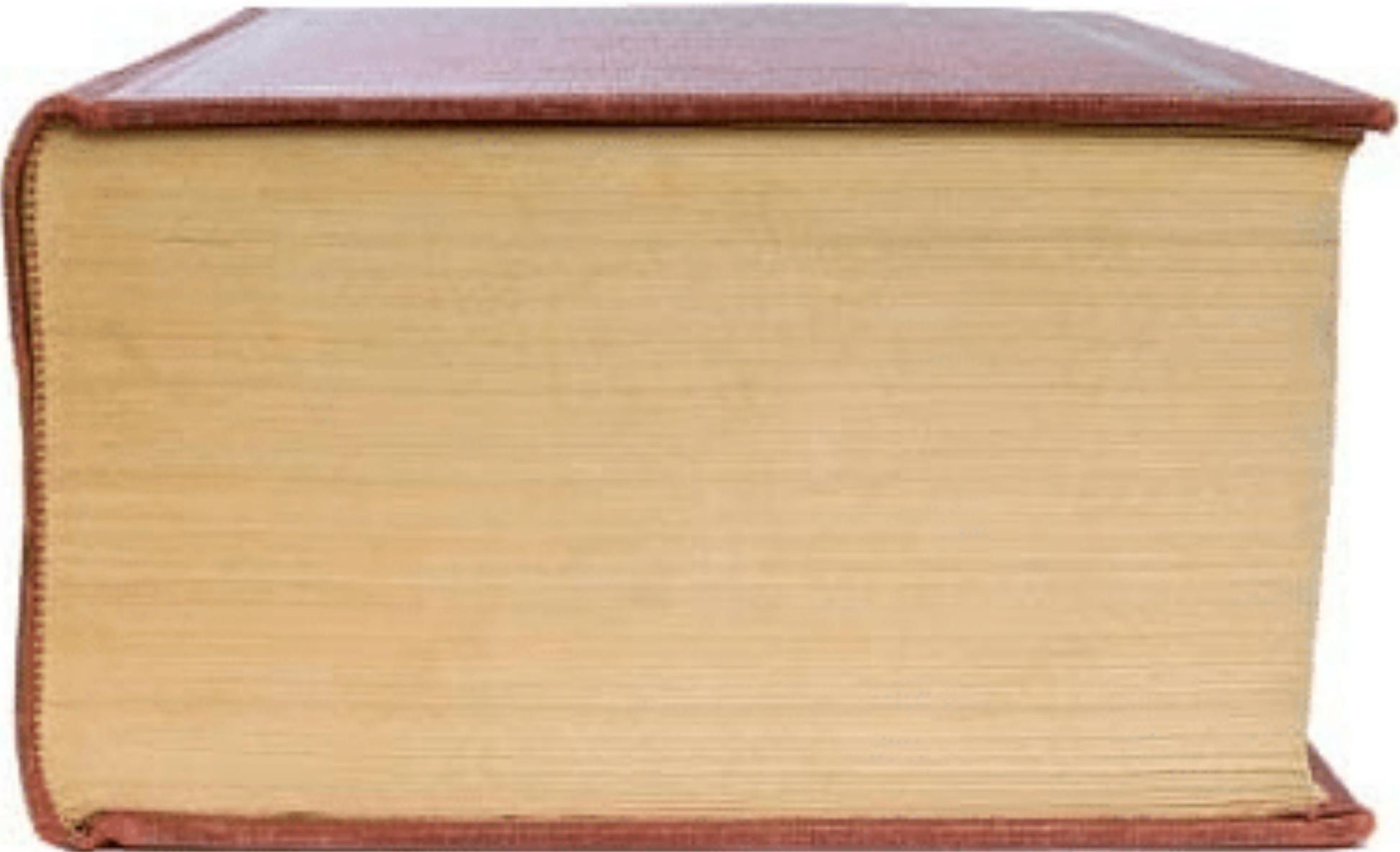
- Puts a bound to the UTXO set growth





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

# Pruned nodes



Without  
Utreeexo



With  
Utreeexo

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

**472GB -> 416B**

# The UTXO set

## Why merkelize it?

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

**6.3G chainstate/**

**384B chainstate/**

# Tiny Utreexo full nodes

We can run them anywhere

- Anywhere with low storage

# Tiny Utreexo full nodes

We can run them anywhere

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

# Tiny Utreexo full nodes

We can run them anywhere

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

# Web Browser full Nodes

Theoretically possible

- Easily accessible. Grandma friendly.

# Web Browser full Nodes

Theoretically possible

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

# Web Browser full Nodes

Theoretically possible

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

# **Router full Nodes**

**Always connected full nodes**

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

# Router full Nodes

## Always connected full nodes

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

Endless possibilities but I'm not  
creative enough

# Immediate node bootstrap

# AssumeUTXO

Quick way to sync up

- Receive a UTXO set hash commitment with the binary

# AssumeUTXO

Quick way to sync up

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

# AssumeUTXO

Quick way to sync up

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

# **AssumeUtreexo**

**Quicker way to sync up**

- Receive the UTXO set with the binary

# AssumeUtreexo

Quicker way to sync up

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



Download Bitcoin Core

# The UTXO set

## Why merkelize it?

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

# Tiny full nodes

Why these can make ibd faster

- Anywhere with slow storage

# Tiny full nodes

## Why these can make ibd faster

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

# Tiny full nodes

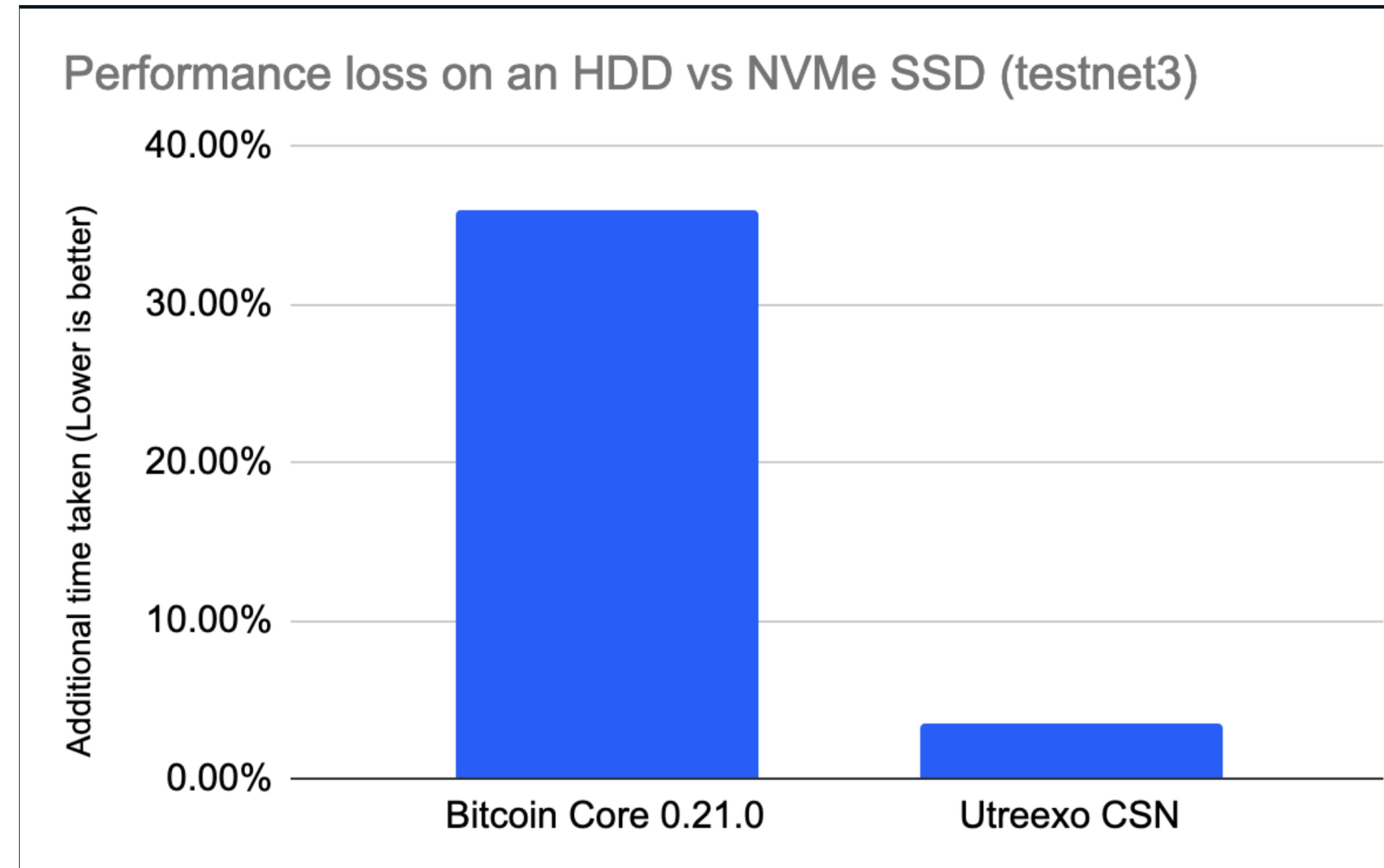
## Why these can make ibd faster

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

# Tiny full nodes

## Why these can make ibd faster

- Less performance degradation for pruned nodes



# Tiny full nodes

Why these can make ibd faster

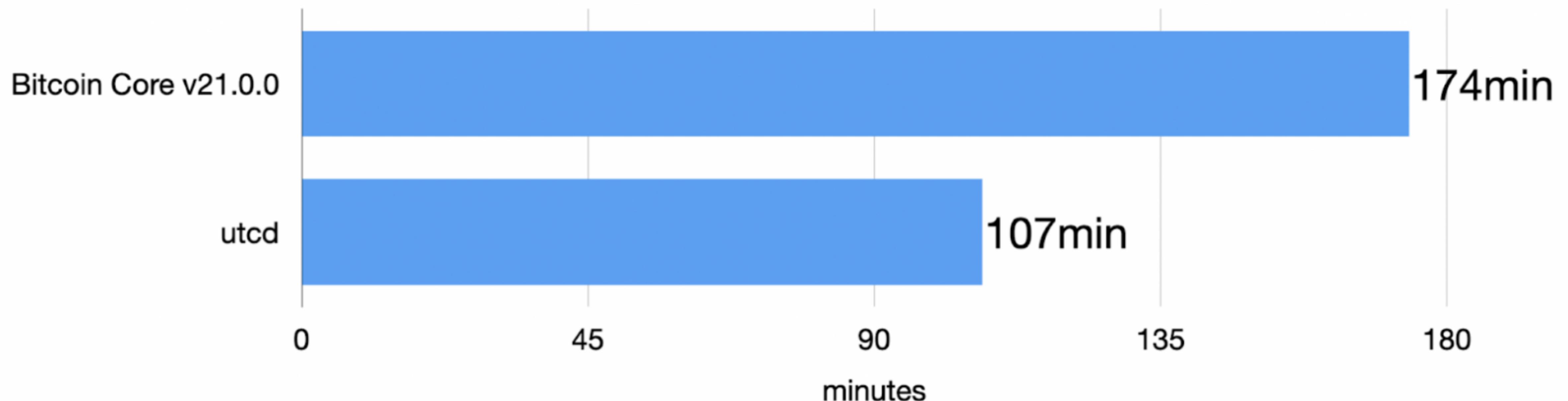
- Tiny UTXO set representation allows for a parallel block validation.

# Tiny full nodes

## Why these can make ibd faster

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

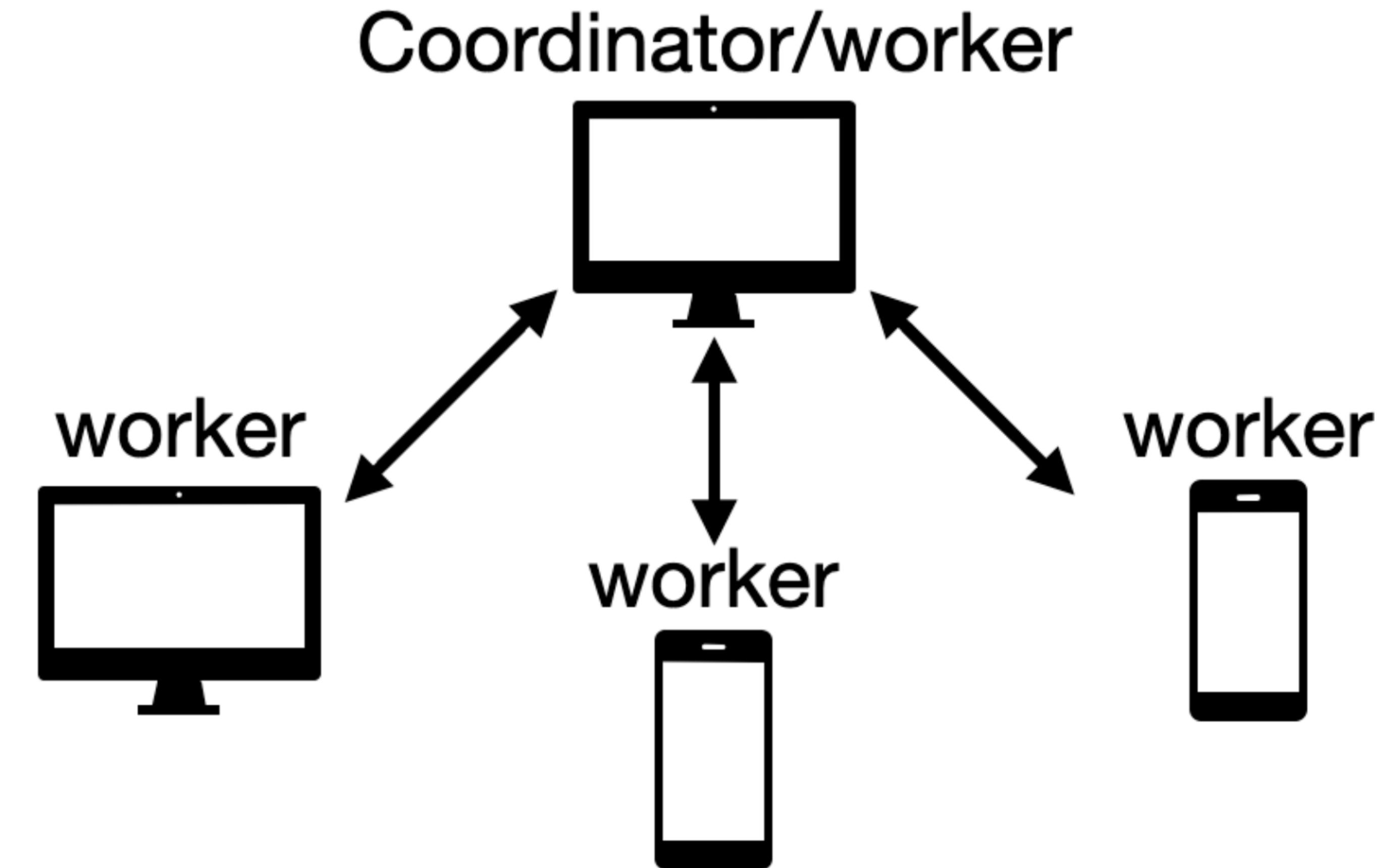
## Initial Block Download Speeds - local nodes (default mode, no signature checks until block 654,683)



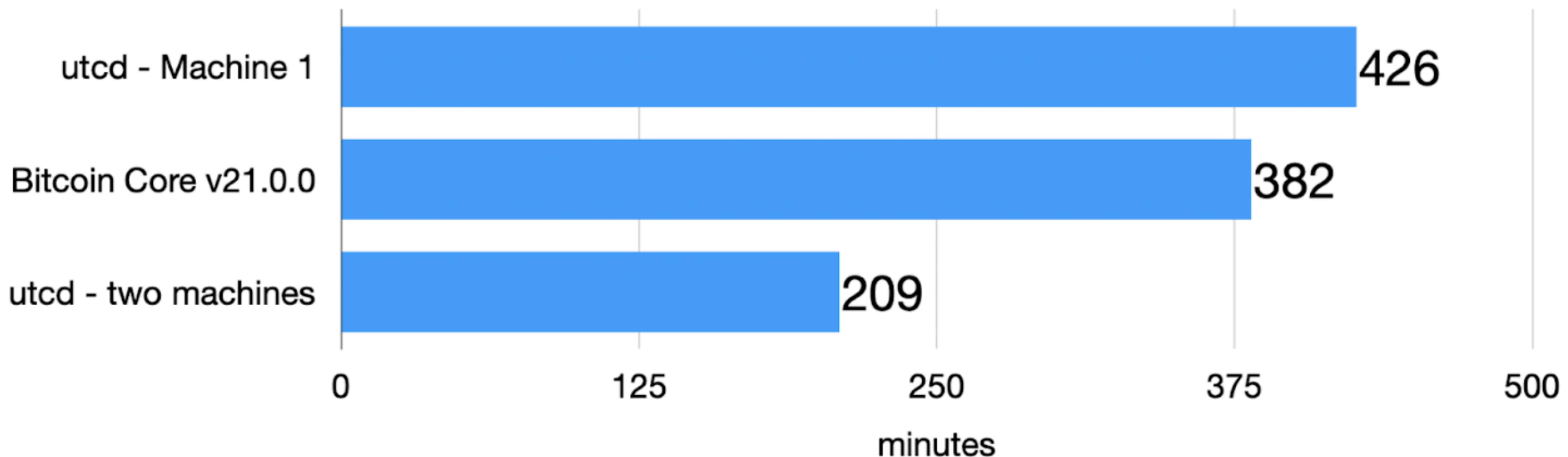
# Multiple device sync

Moar processing power

- Take advantage of all the processing power you have

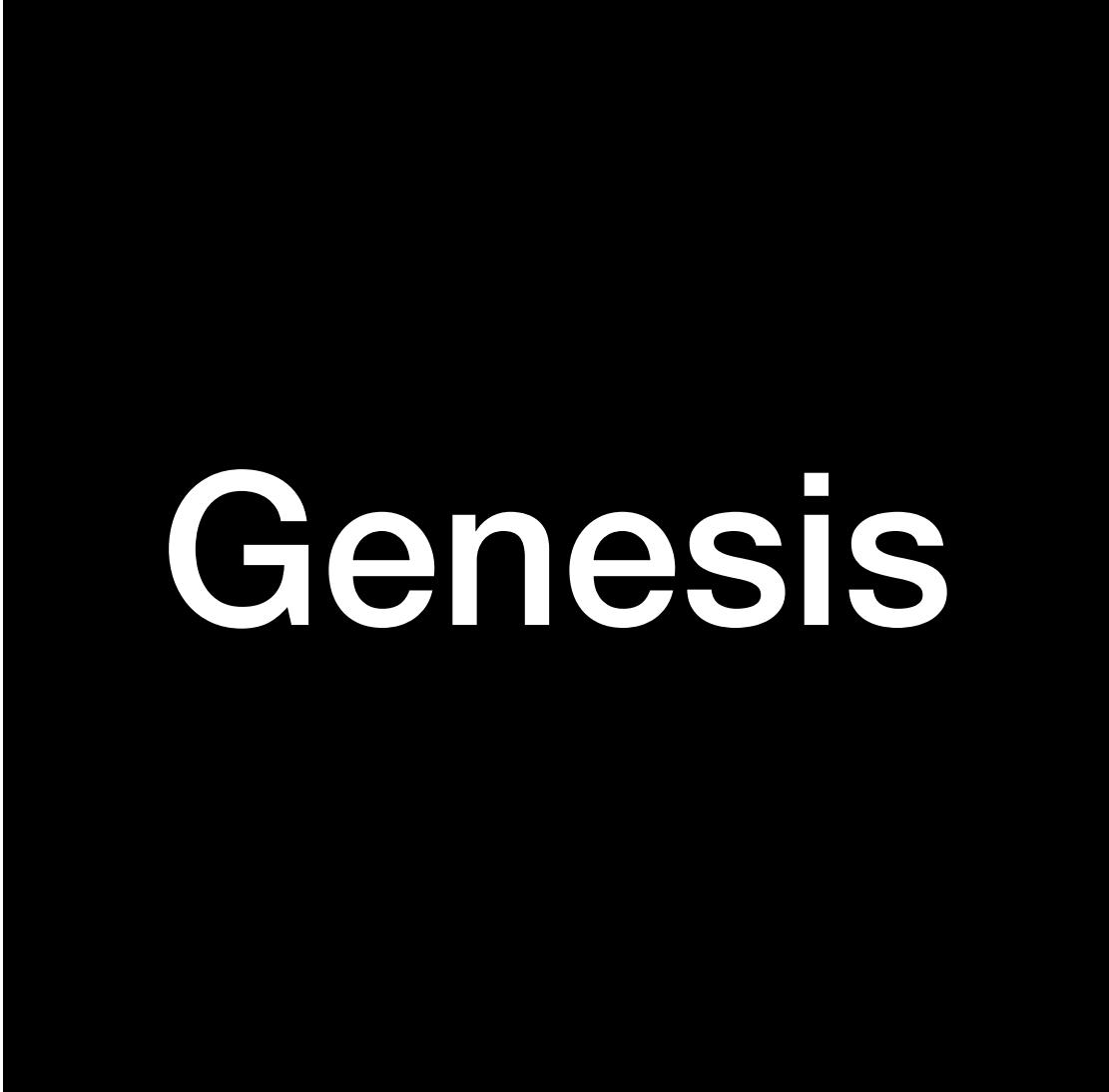


## Initial Block Download Speeds to block 671,000 (Full signature check)



# **Current block validation**

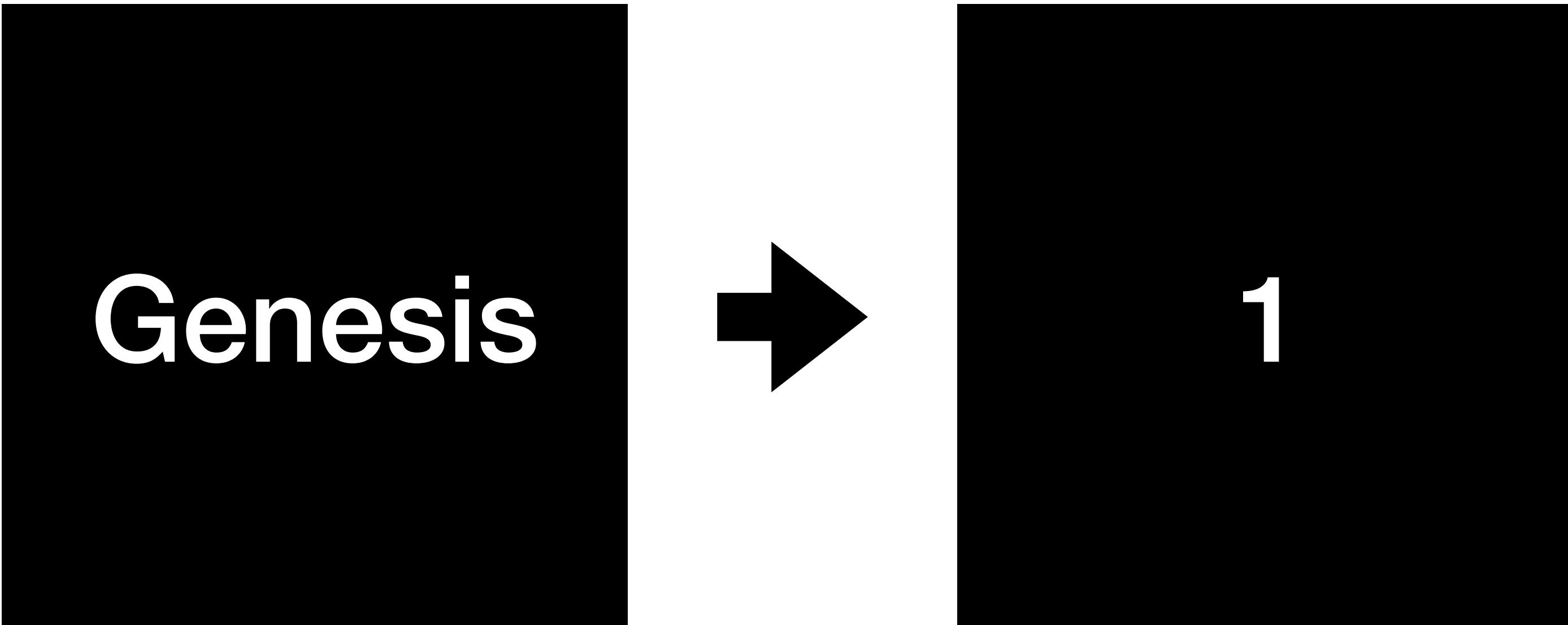
## **Sequential validation**



**Genesis**

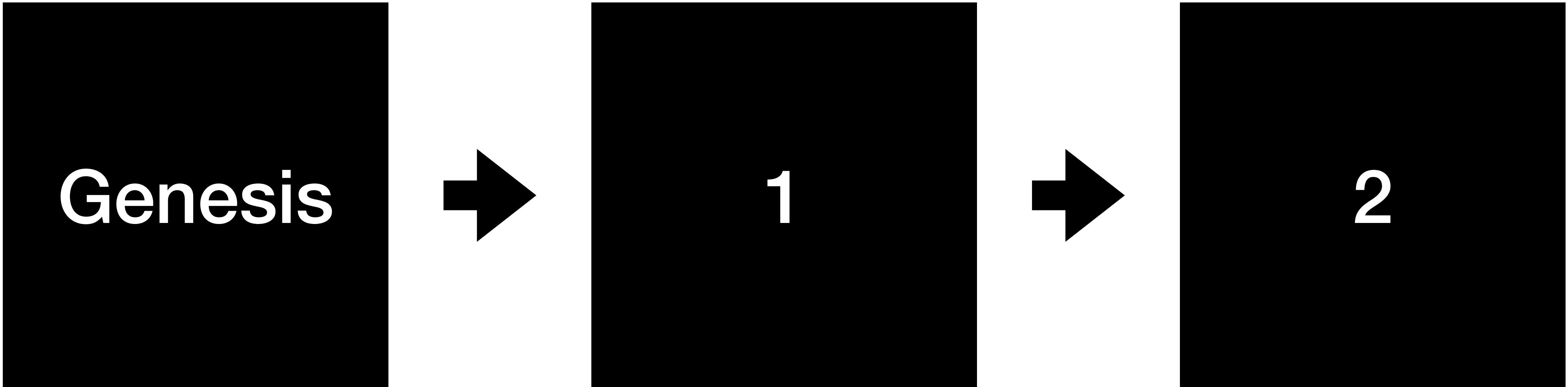
# **Current block validation**

**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
- <https://github.com/mit-dci/utcd/blob/master/chaincfg/mainnetroots.go>

# With Utreexo

## Efficient parallel validation

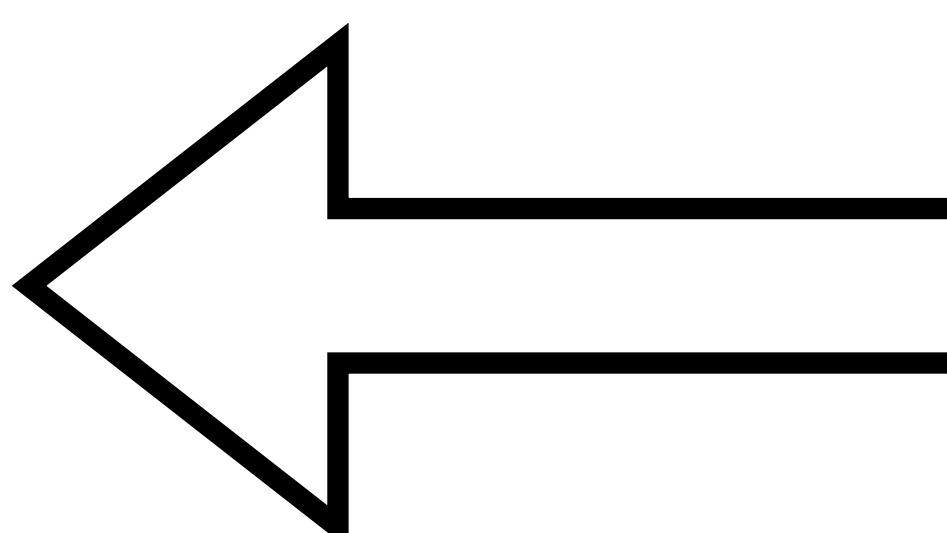
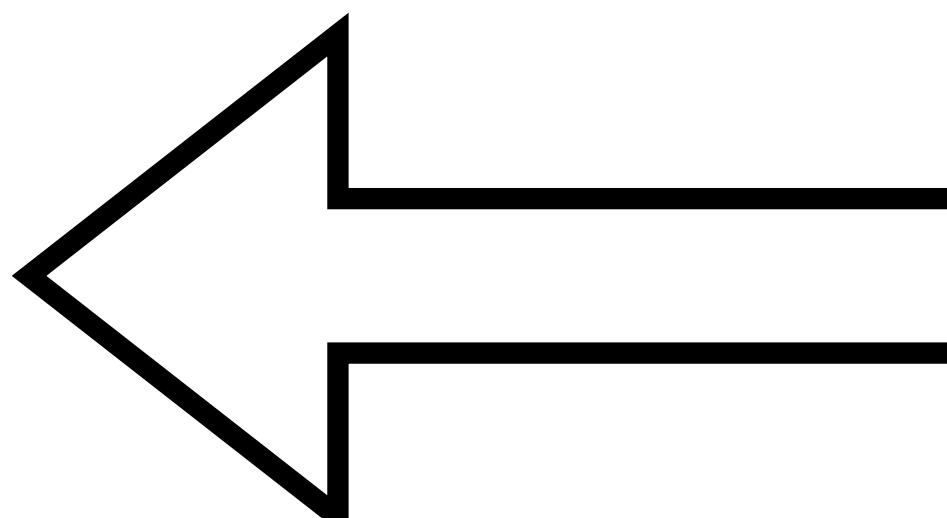
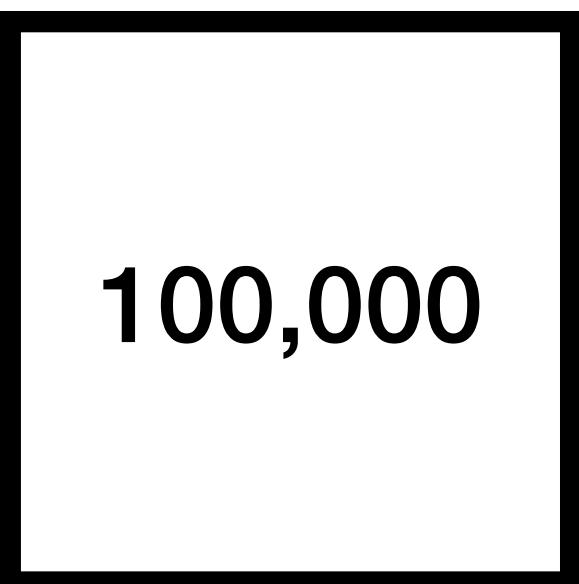
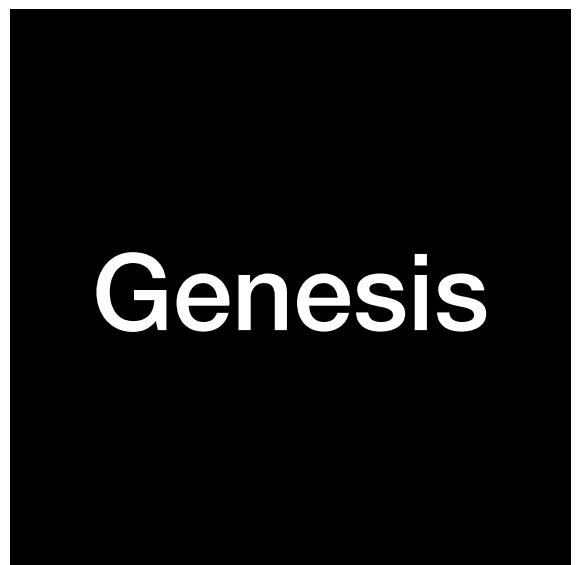
Genesis

100,000

200,000

# With Utreexo

Efficient parallel validation



Untrusted roots at  
the start

# With Utreexo

## Efficient parallel validation

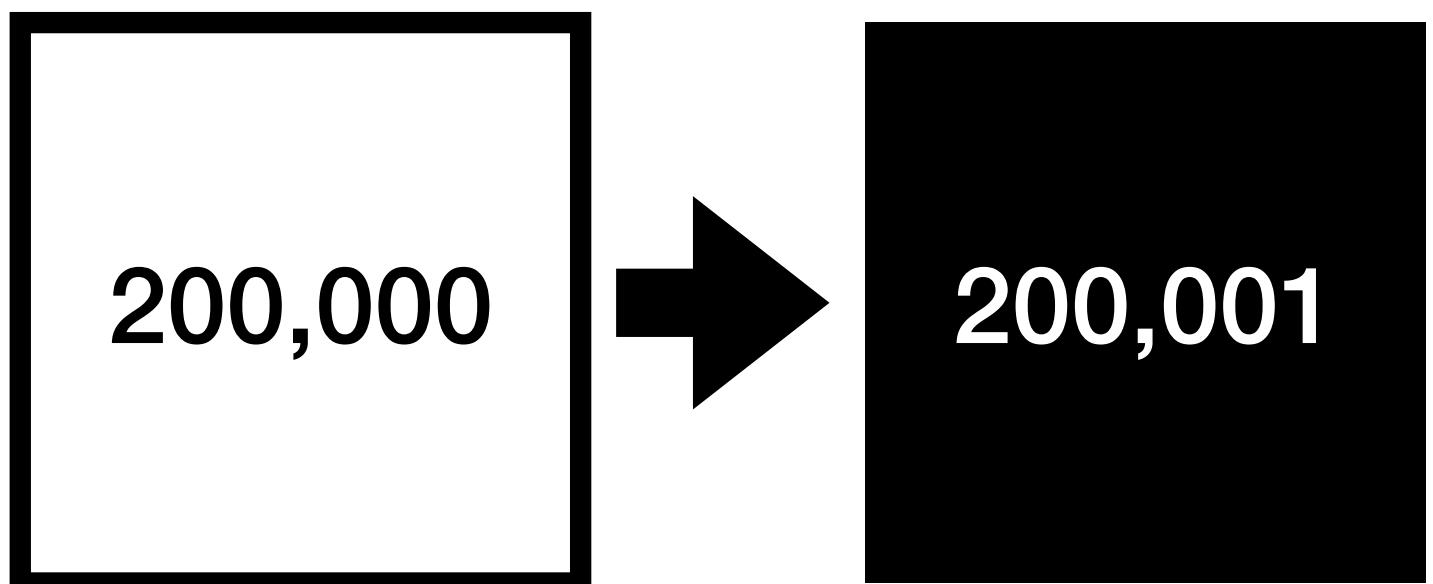
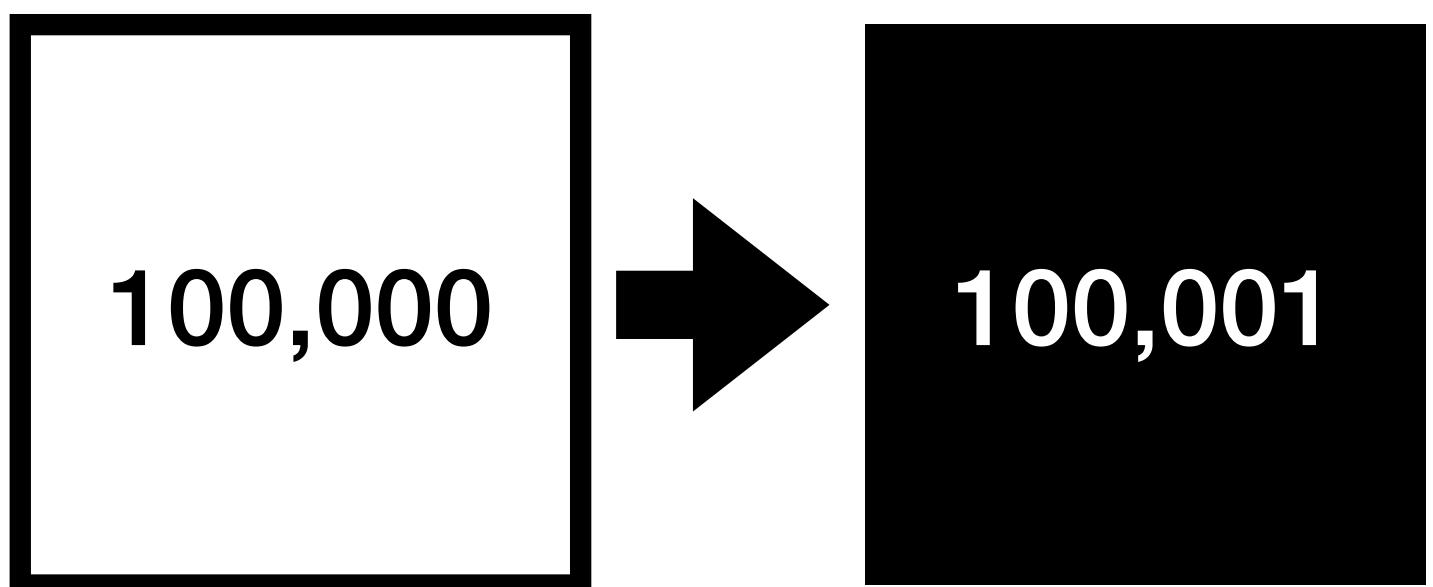
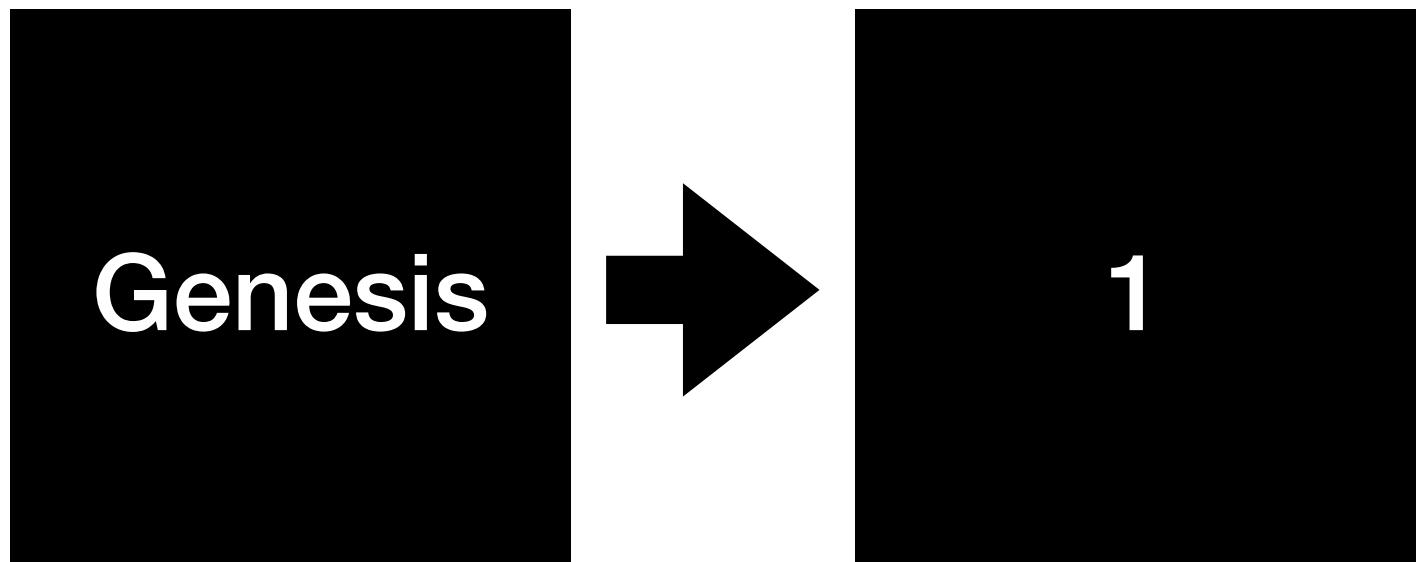
Genesis

100,000

200,000

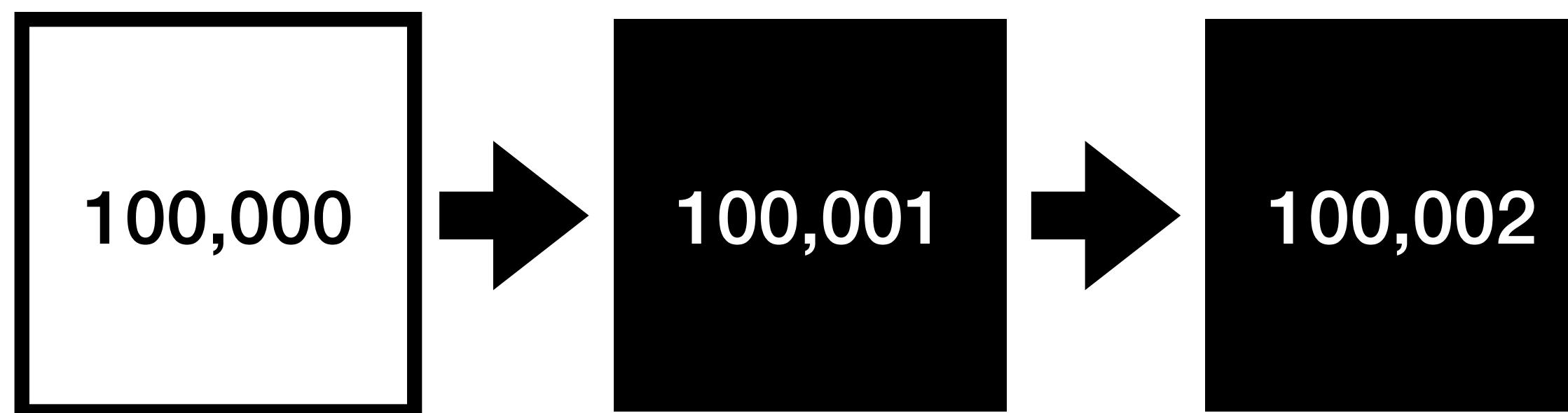
# With Utreexo

## Efficient parallel validation



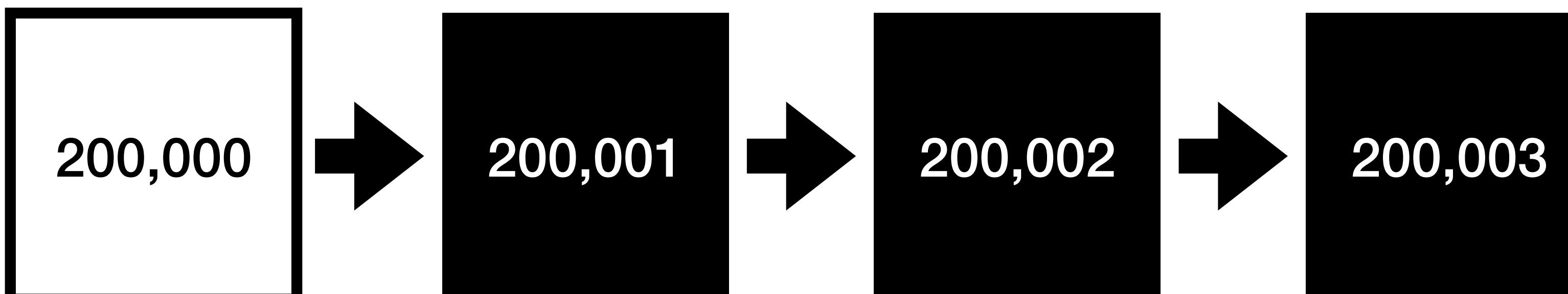
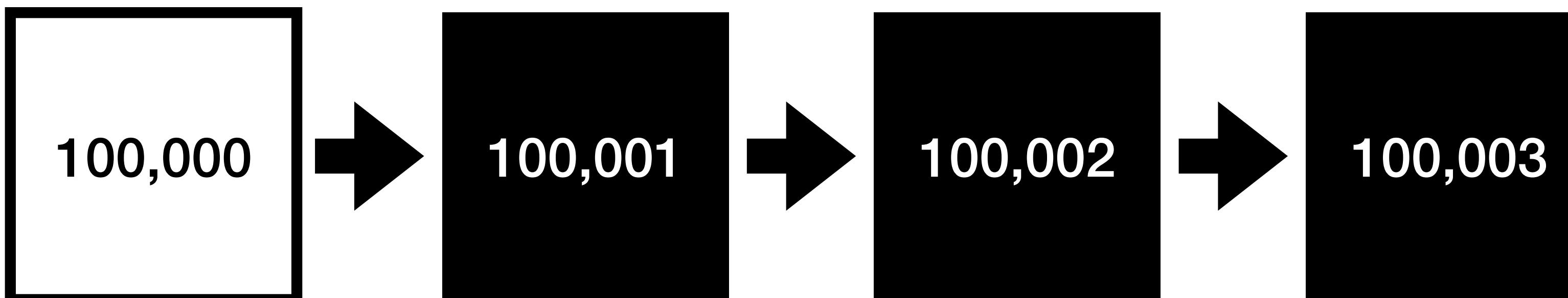
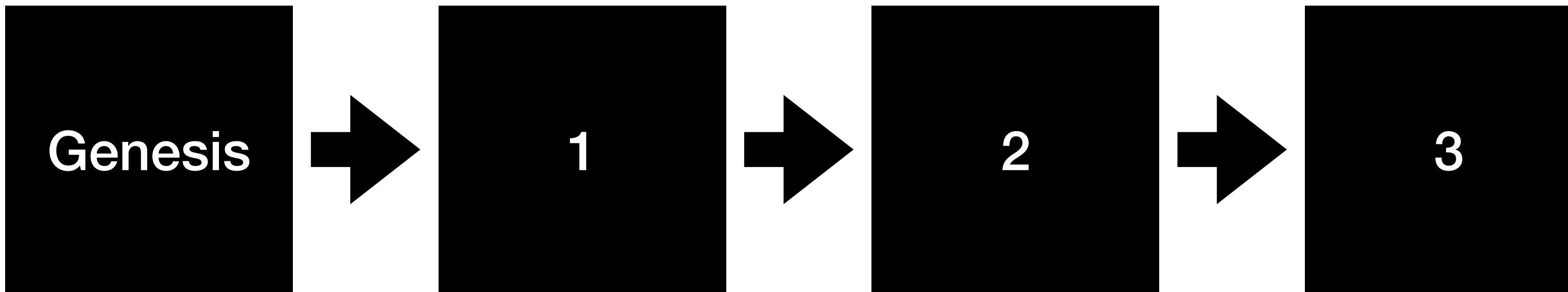
# With Utreexo

## Efficient parallel validation



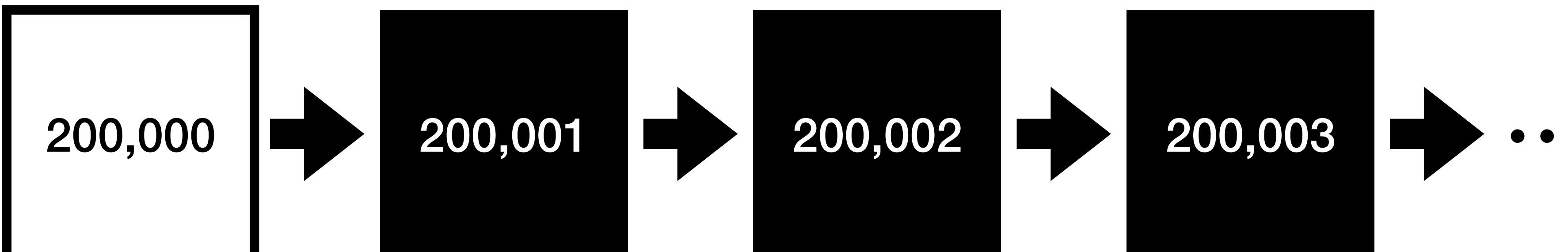
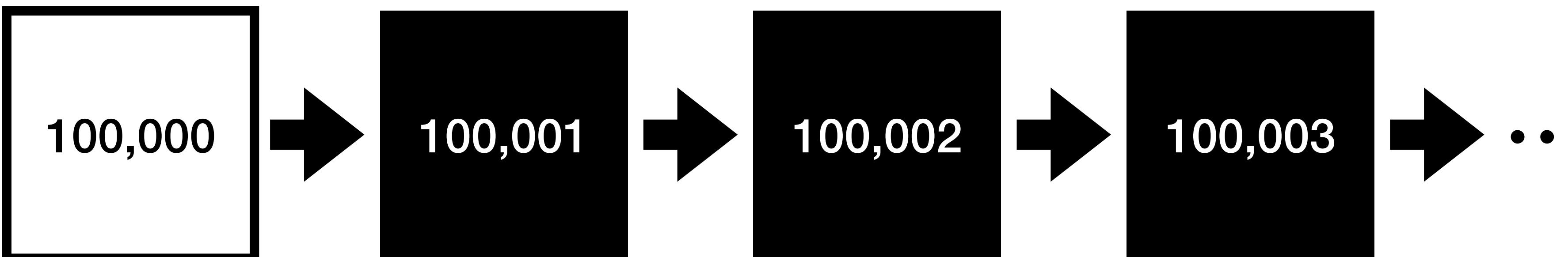
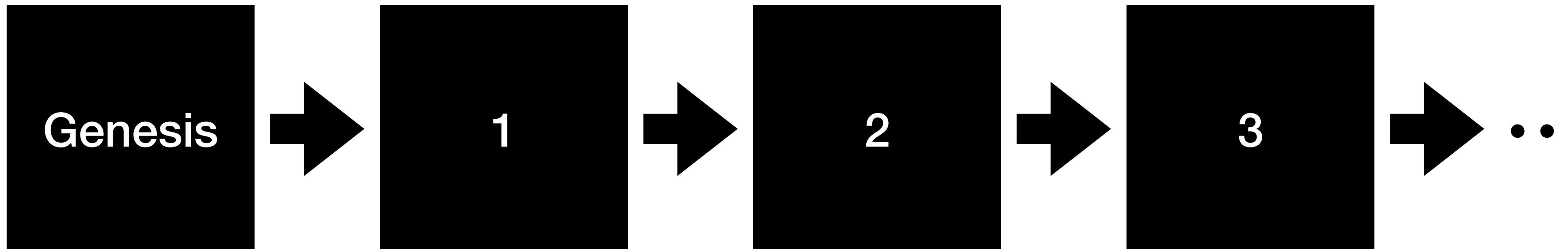
# With Utreexo

## Efficient parallel validation



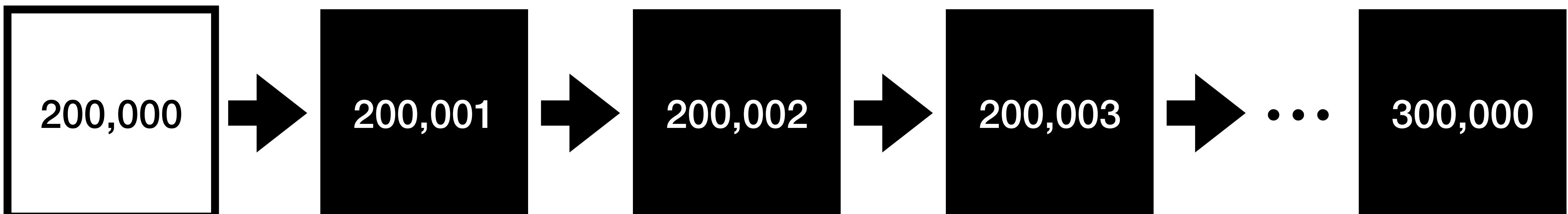
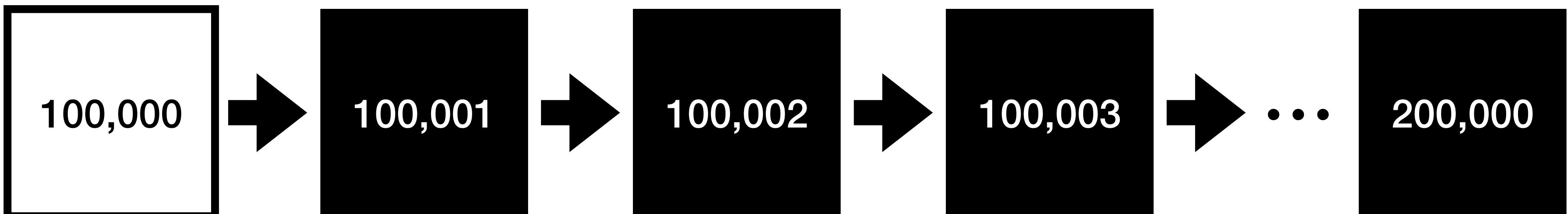
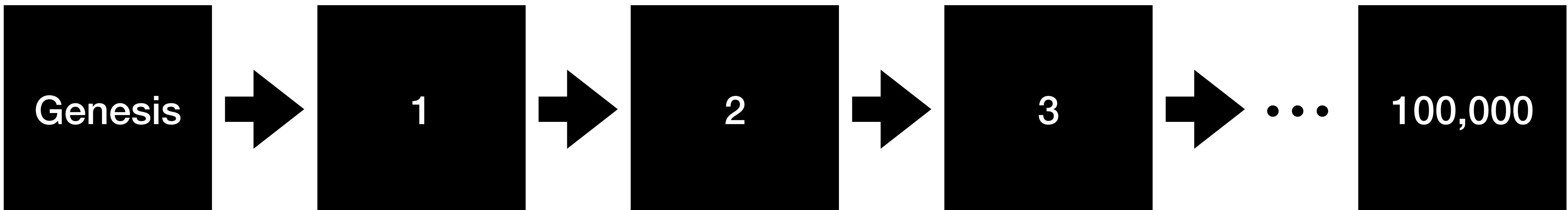
# With Utreexo

## Efficient parallel validation



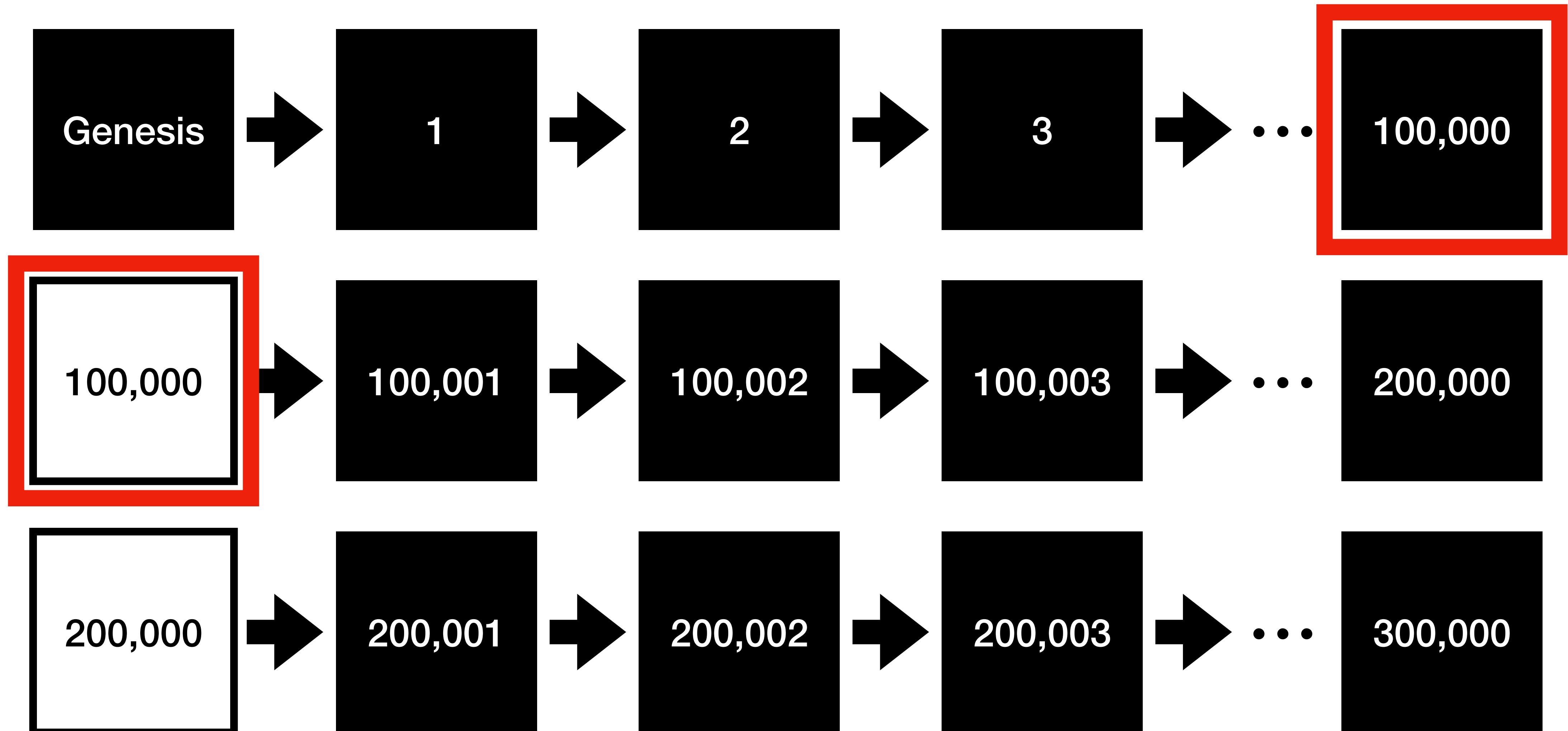
# With Utreexo

## Efficient parallel validation



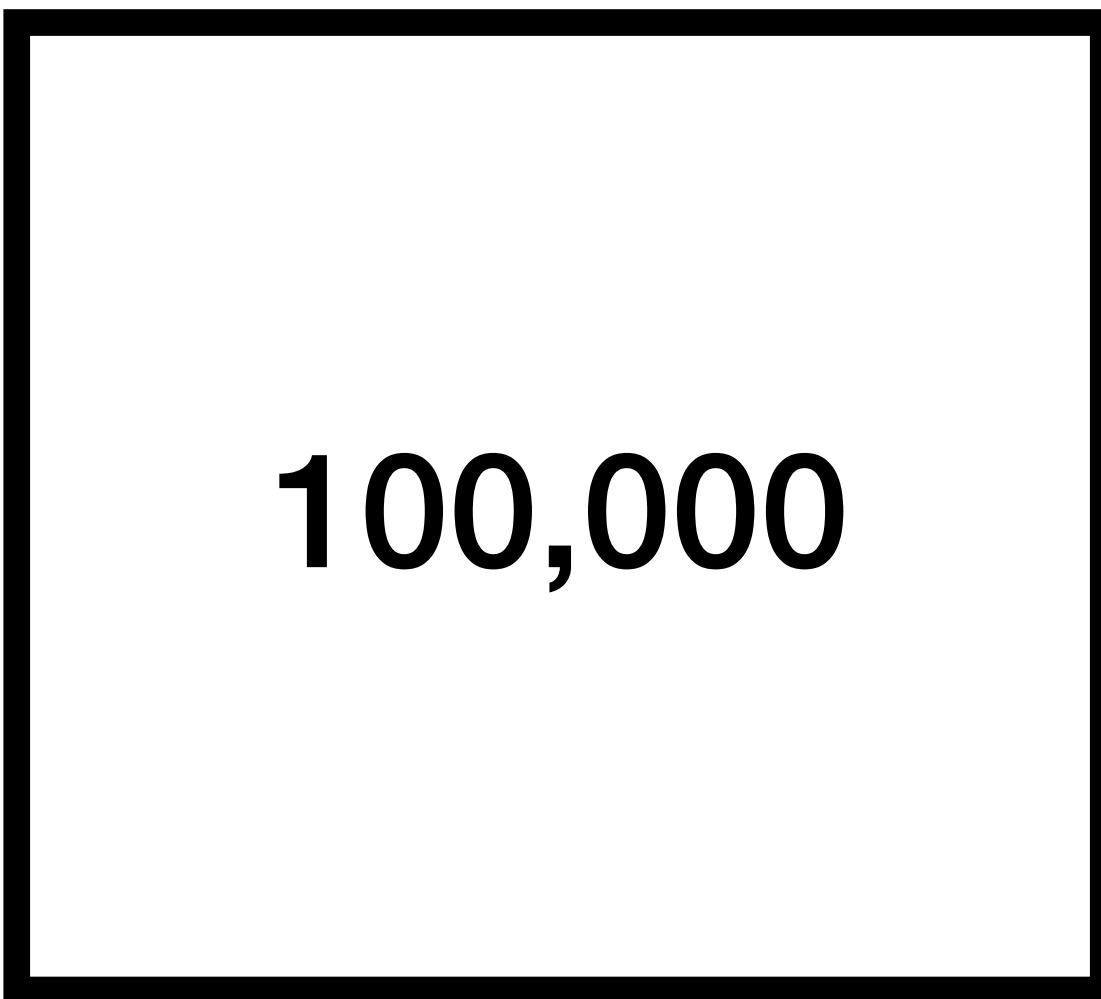
# With Utreexo

## Efficient parallel validation



# Compare roots

Root at 100,000 becomes trusted if equal

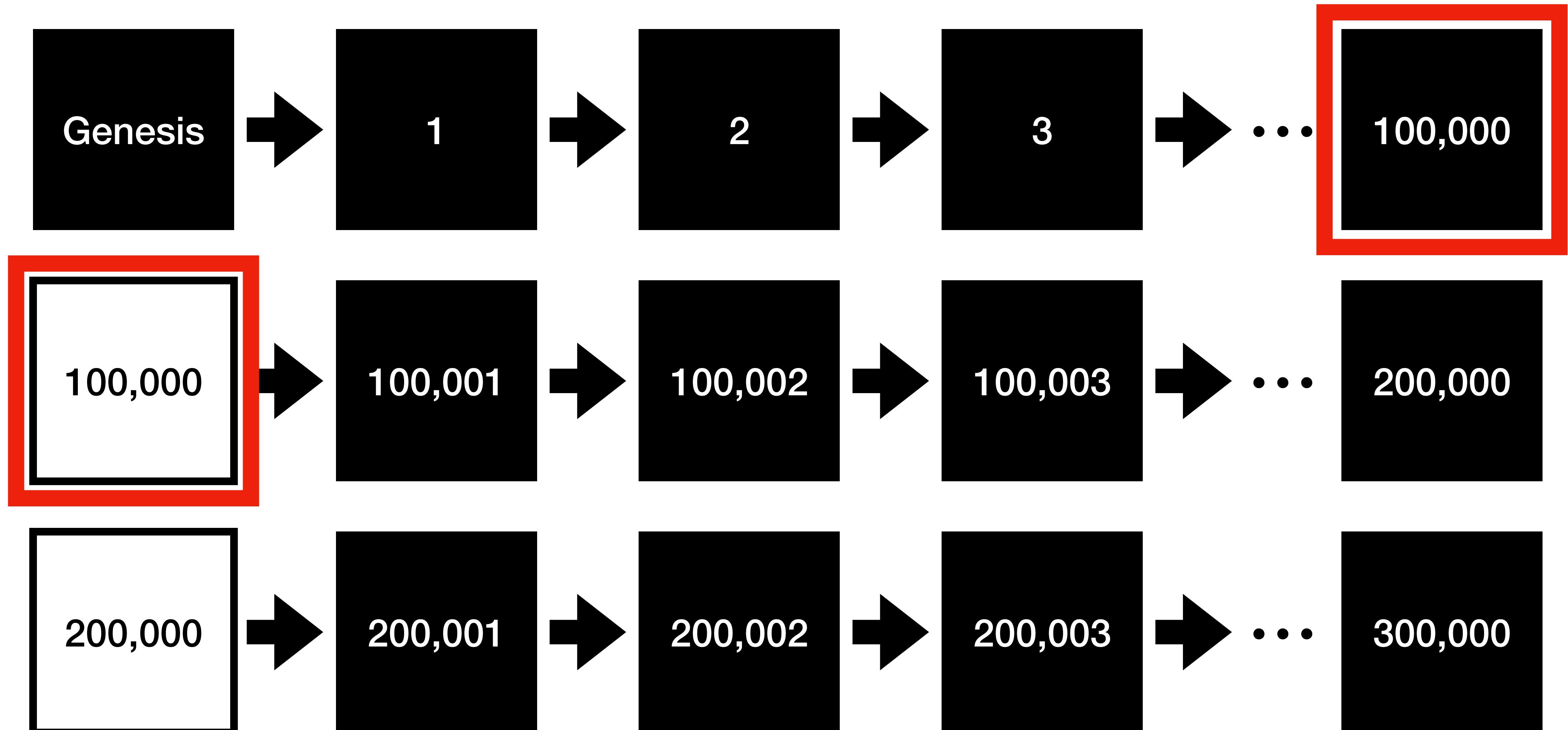


==



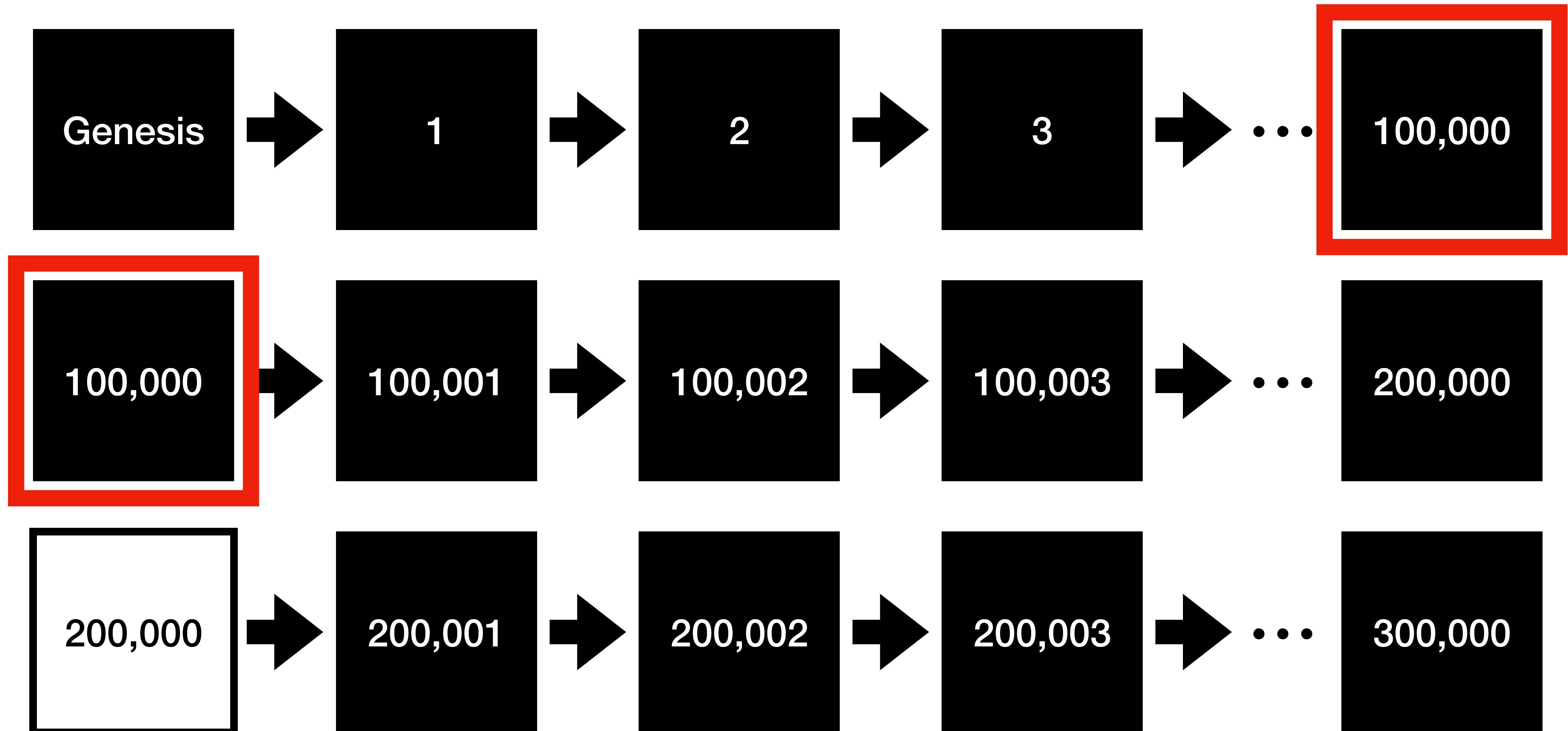
# With Utreexo

## Efficient parallel validation



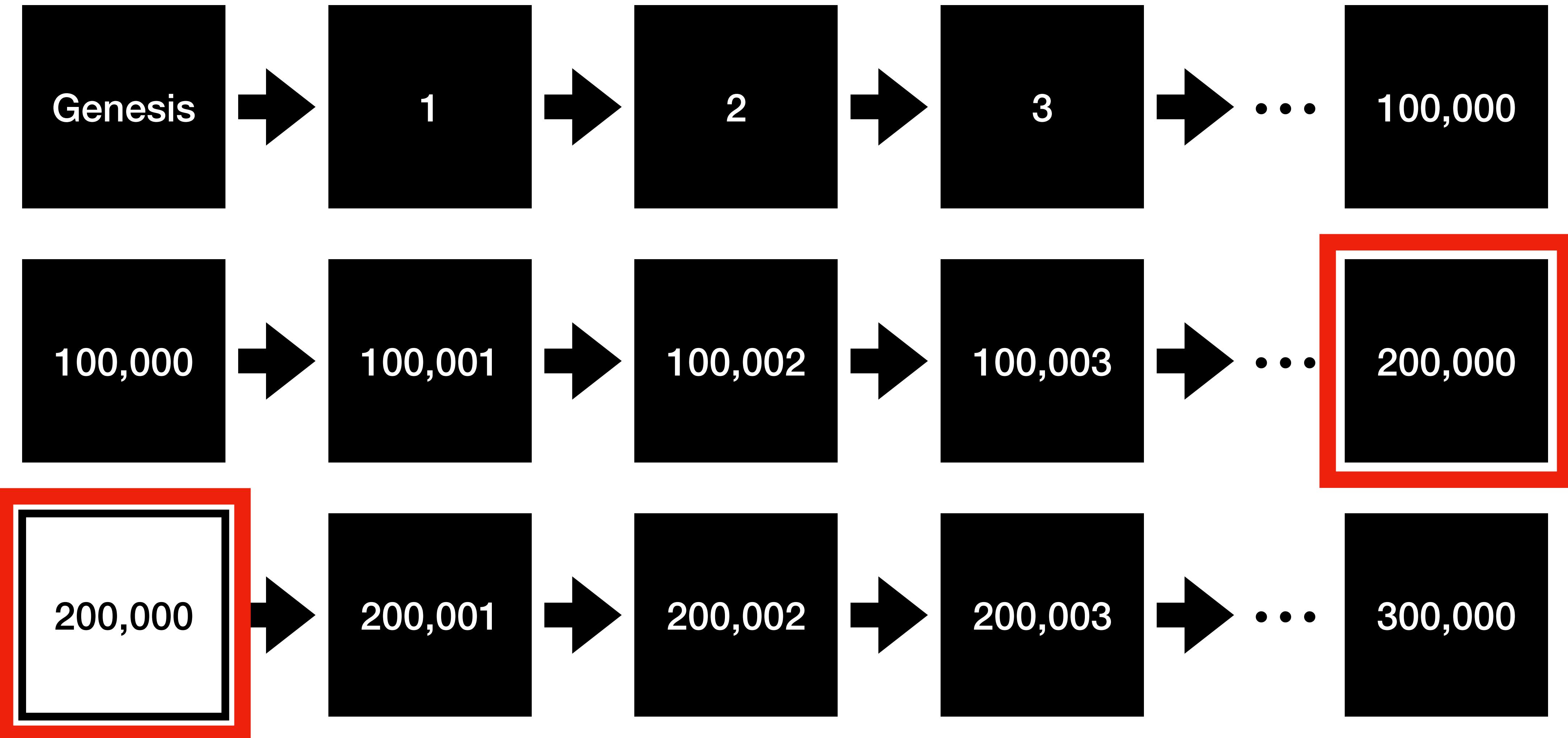
# With Utreexo

## Efficient parallel validation



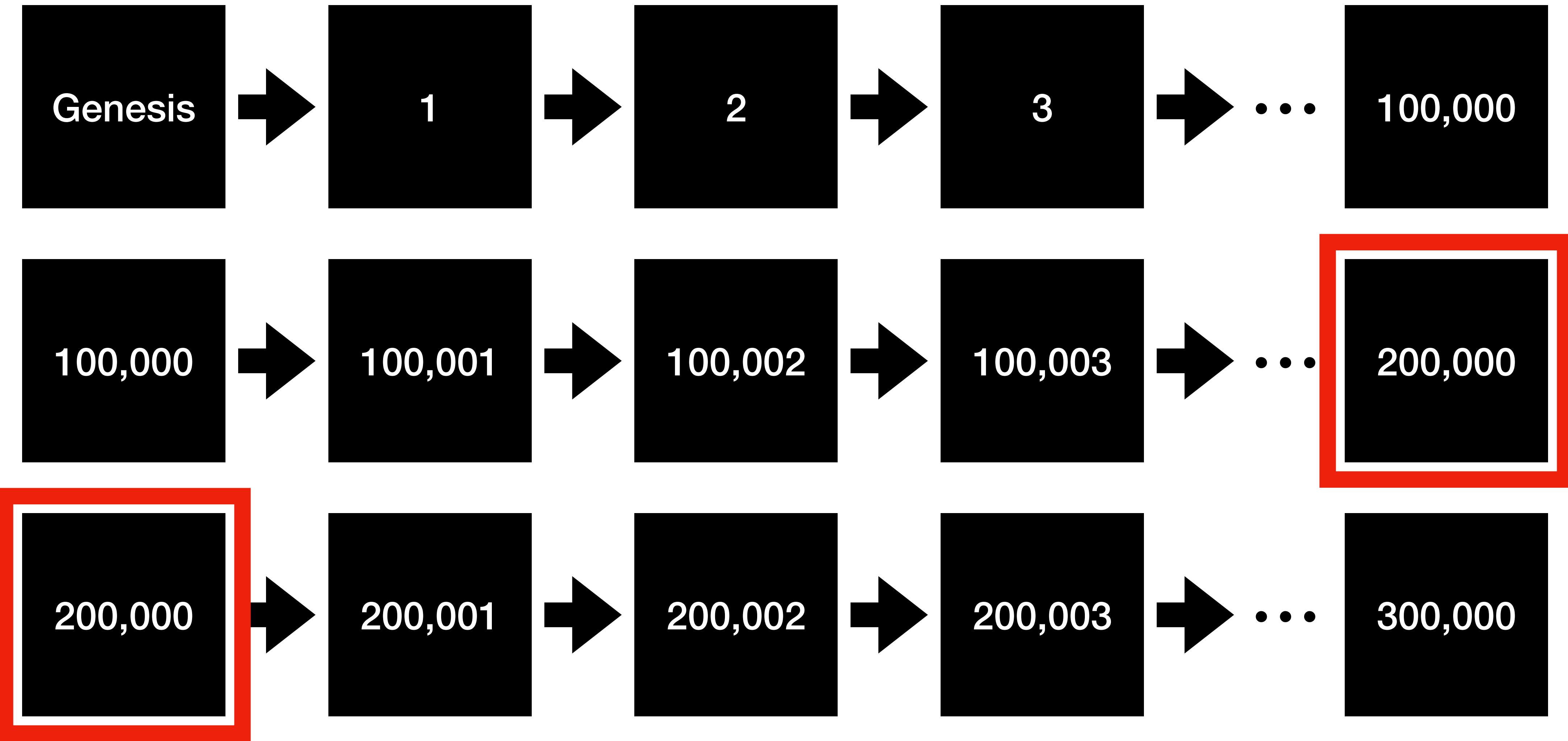
# With Utreexo

## Efficient parallel validation



# With Utreexo

## Efficient parallel validation



# *Efficient* Parallel Validation

**SHA256**  **> Map access** 

# The UTXO set

## Why merkelize 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 (UTXO set)

# The Block validation

## Basic steps

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

# **The Block validation with Utreexo**

## **Basic steps**

- 1. Check Header (PoW)**

# The Block validation with Utreexo

## Basic steps

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

# The Block validation with Utreexo

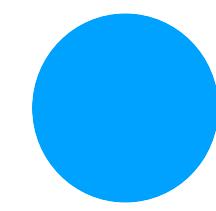
## Basic steps

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

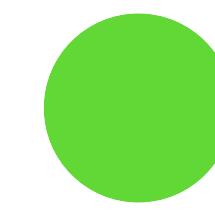
# Total overhead

Extra data that a utreexo node will download

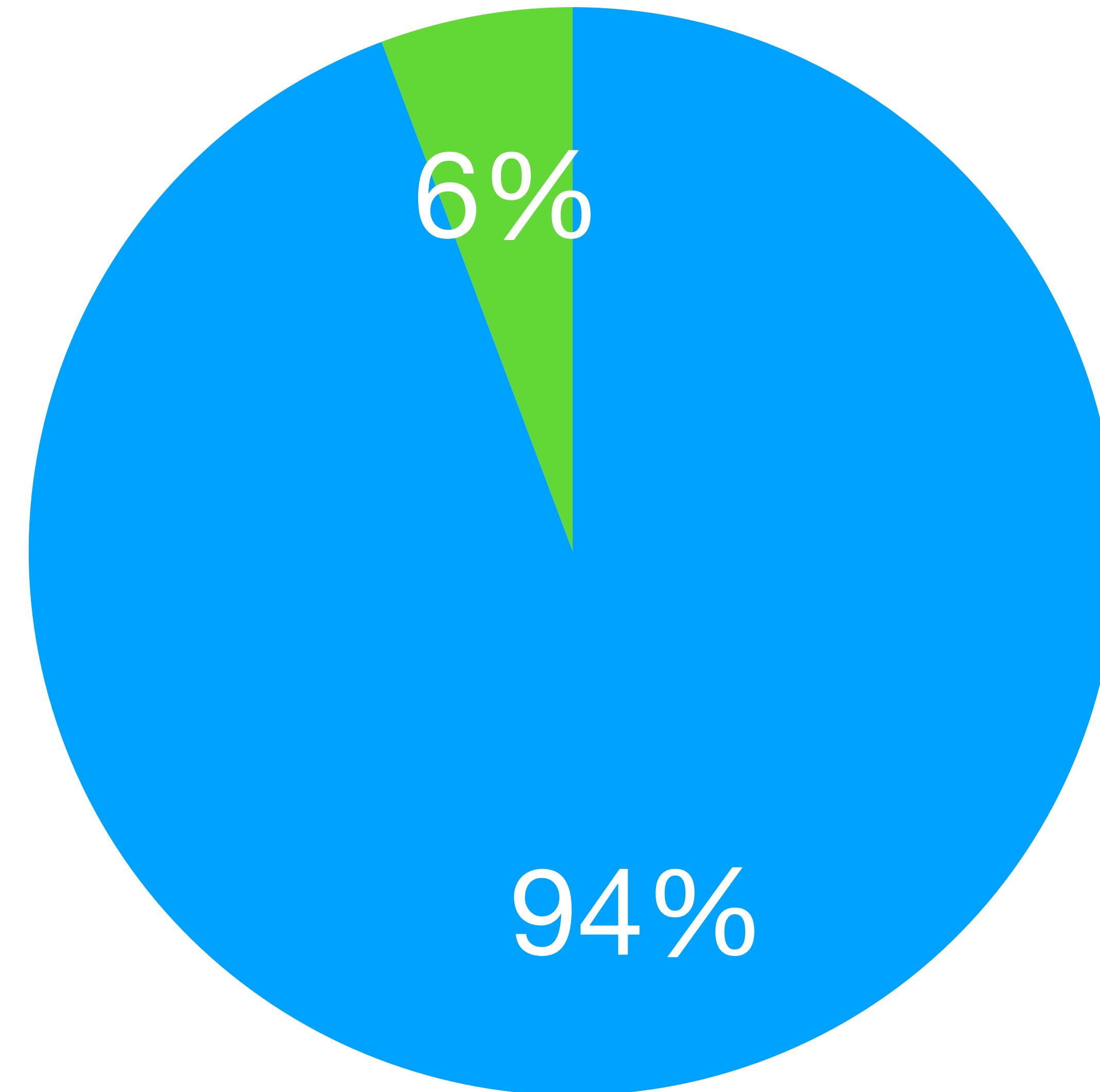
- 364GB of extra data (as of block 710,000)
- Caching and batch proving multiple blocks is being explored



Merkle Proof



UTXO data



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/rustreeexo**

## Accumulator in Rust

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

**github.com/Davidson-Souza/Floresta**

**Full node in Rust**

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

# Progress towards end user readiness

Things done & things left to do

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

# Progress towards end user readiness

Things done & things left to do



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

# Progress towards end user readiness

Things done & things left to do



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

# Progress towards end user readiness

Things done & things left to do

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

# Rust implementation

Things going on the Rust side by Davidson



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

# How does it work?

# **Replace the UTXO set**

**What does the UTXO set  
do for bitcoin?**

# **Role of the UTXO set**

**It let's you**

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

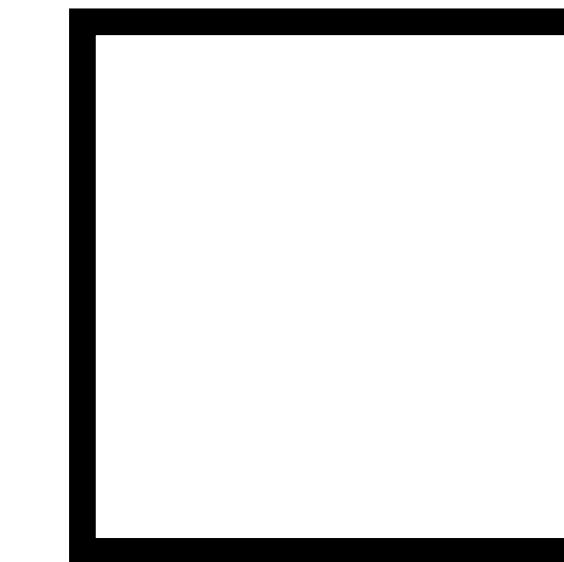
# Role of the UTXO set

It let's you

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

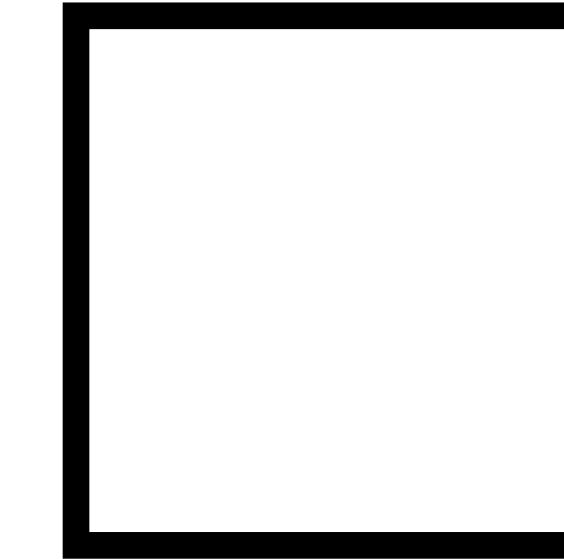
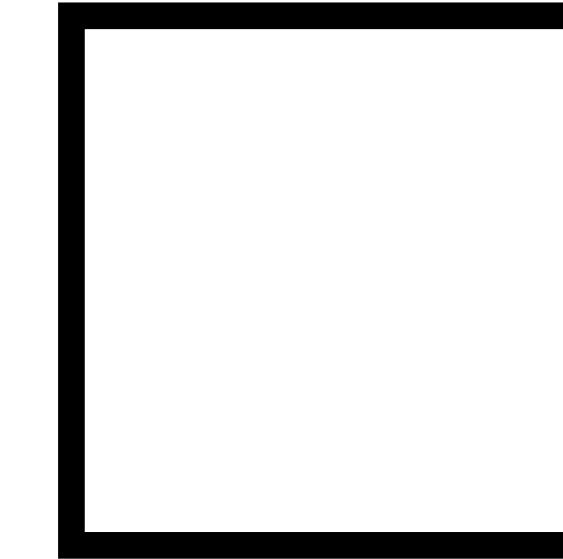
# **1 UTXO**

## **Single root**

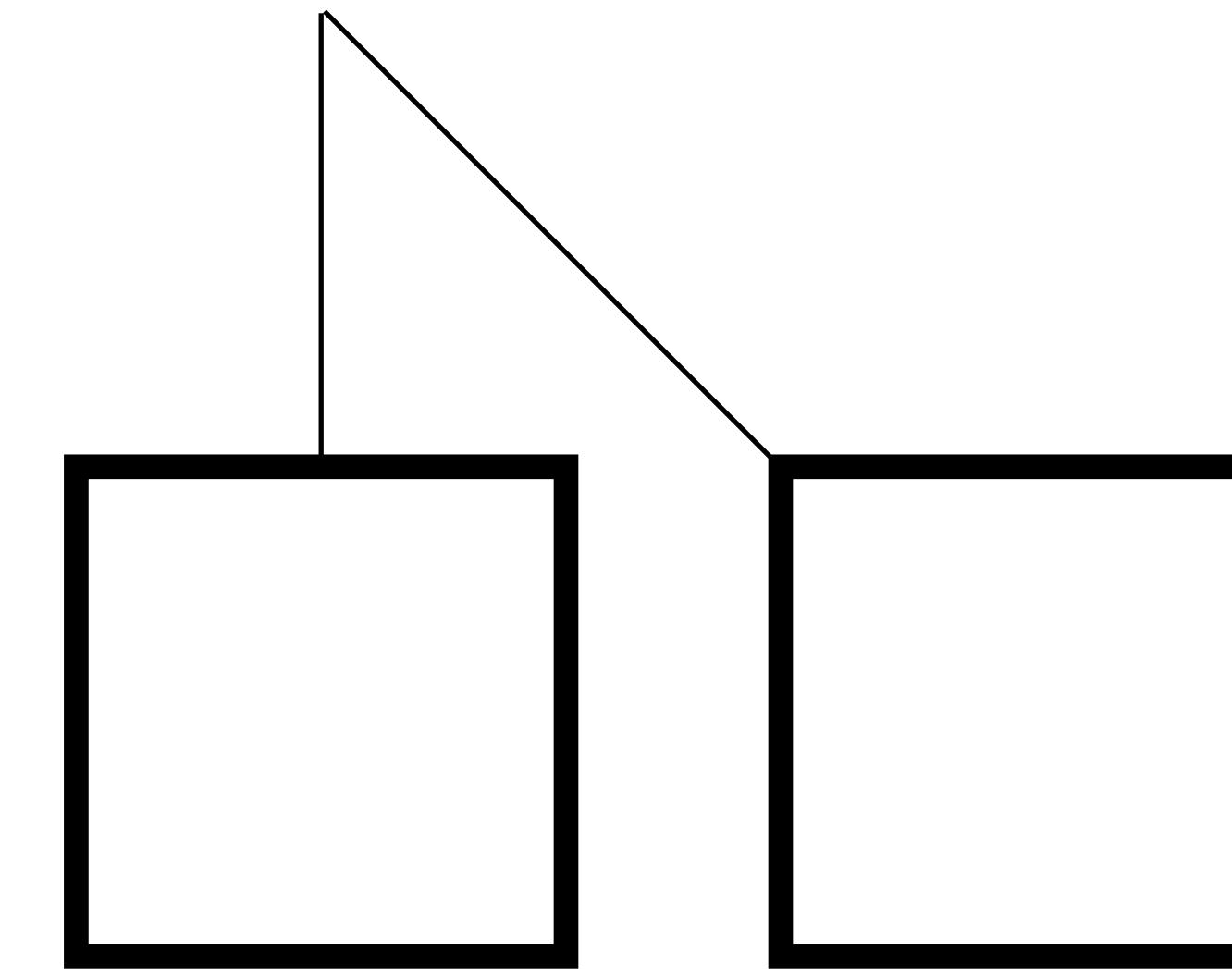


# **2 UTXOs**

**Add 1 more**

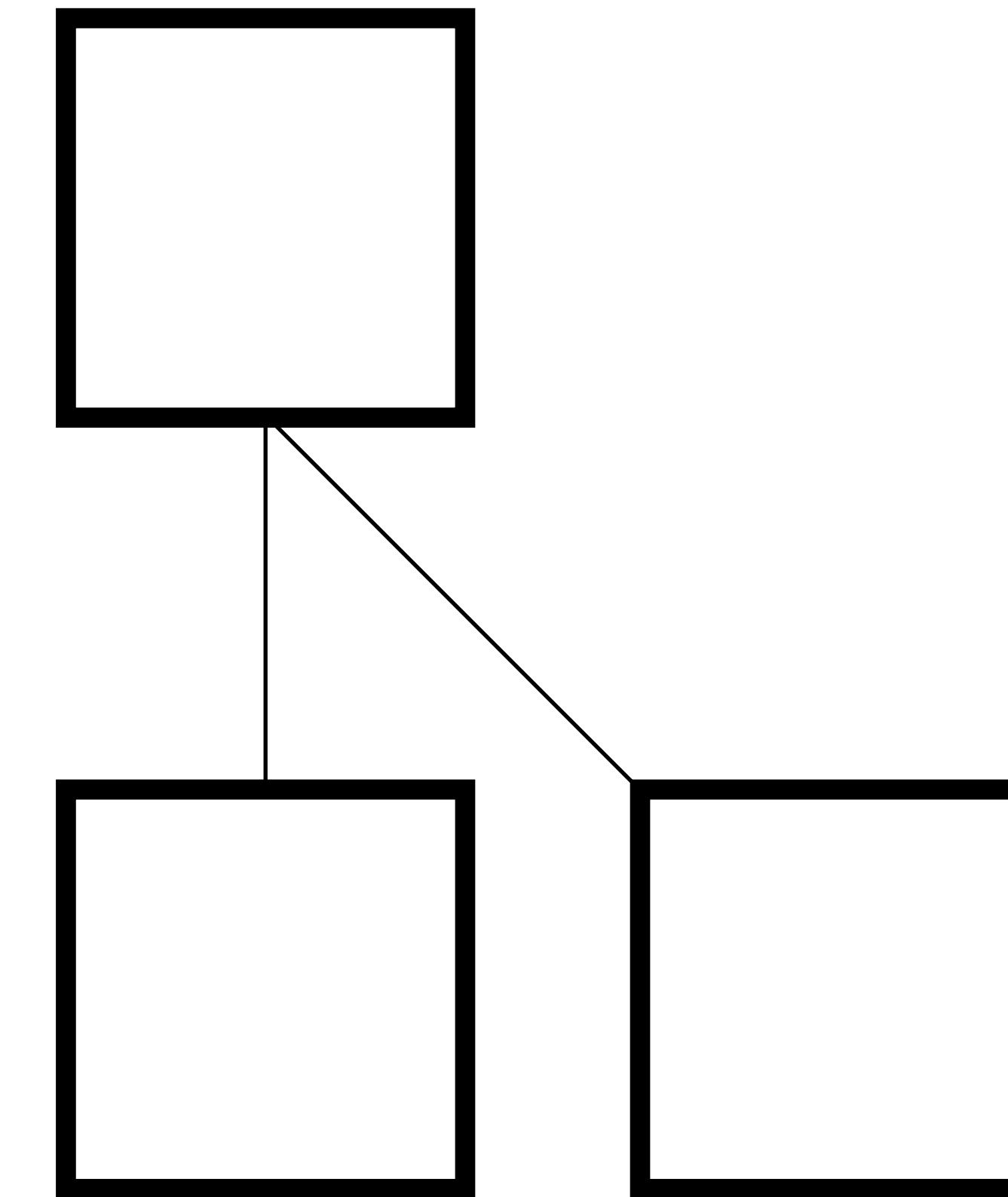


# 2 UTXOs Concatenate



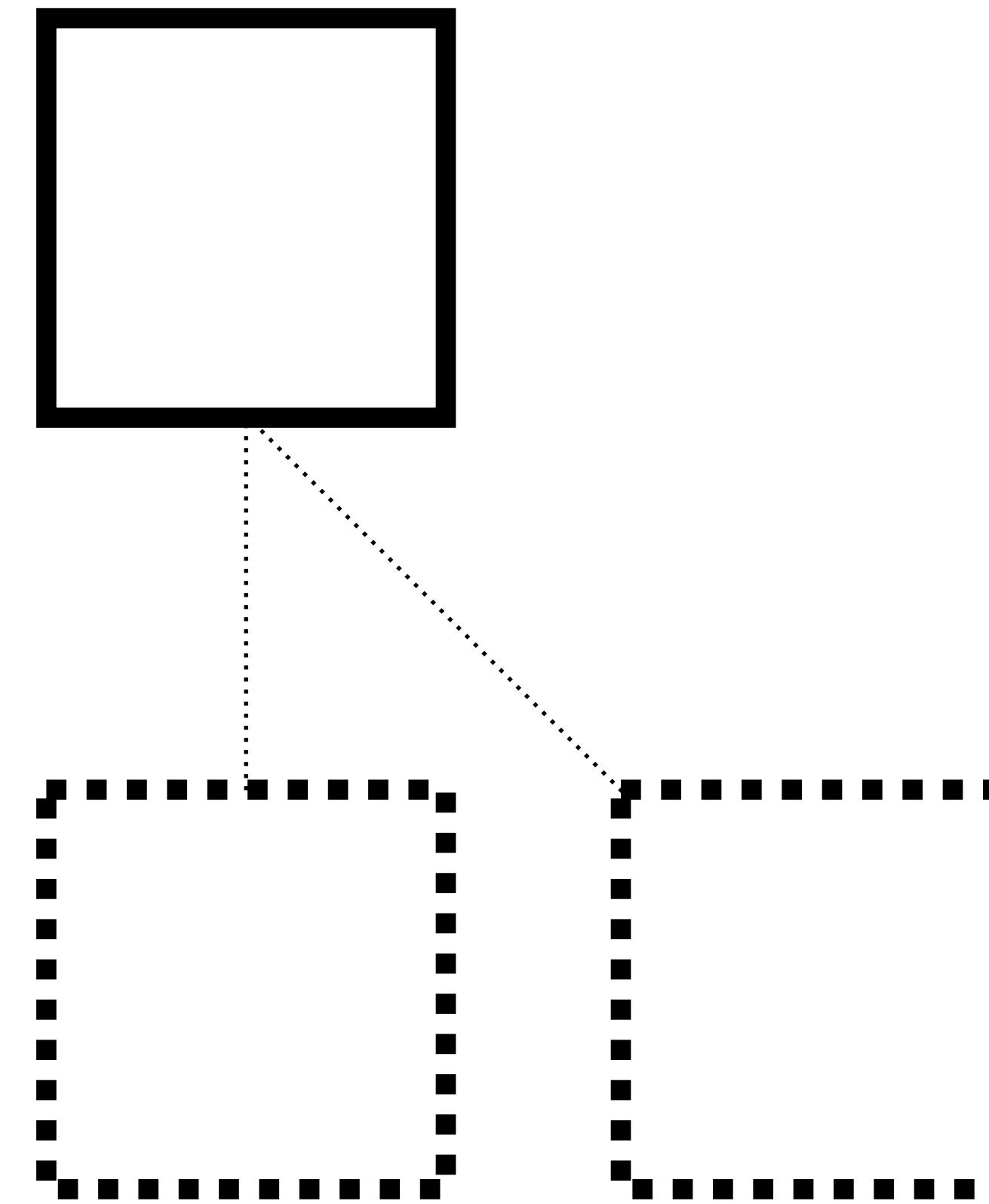
# 2 UTXOs

Hash to create a new root



# 2 UTXOs

Can now delete the leaves



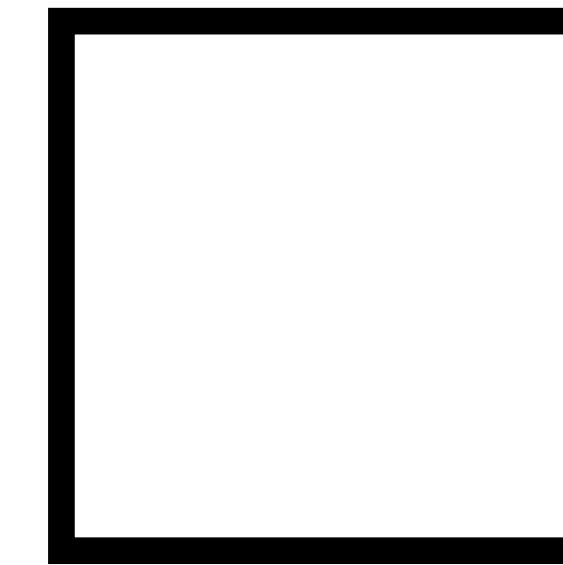
# 2 UTXOs

Only keep the roots



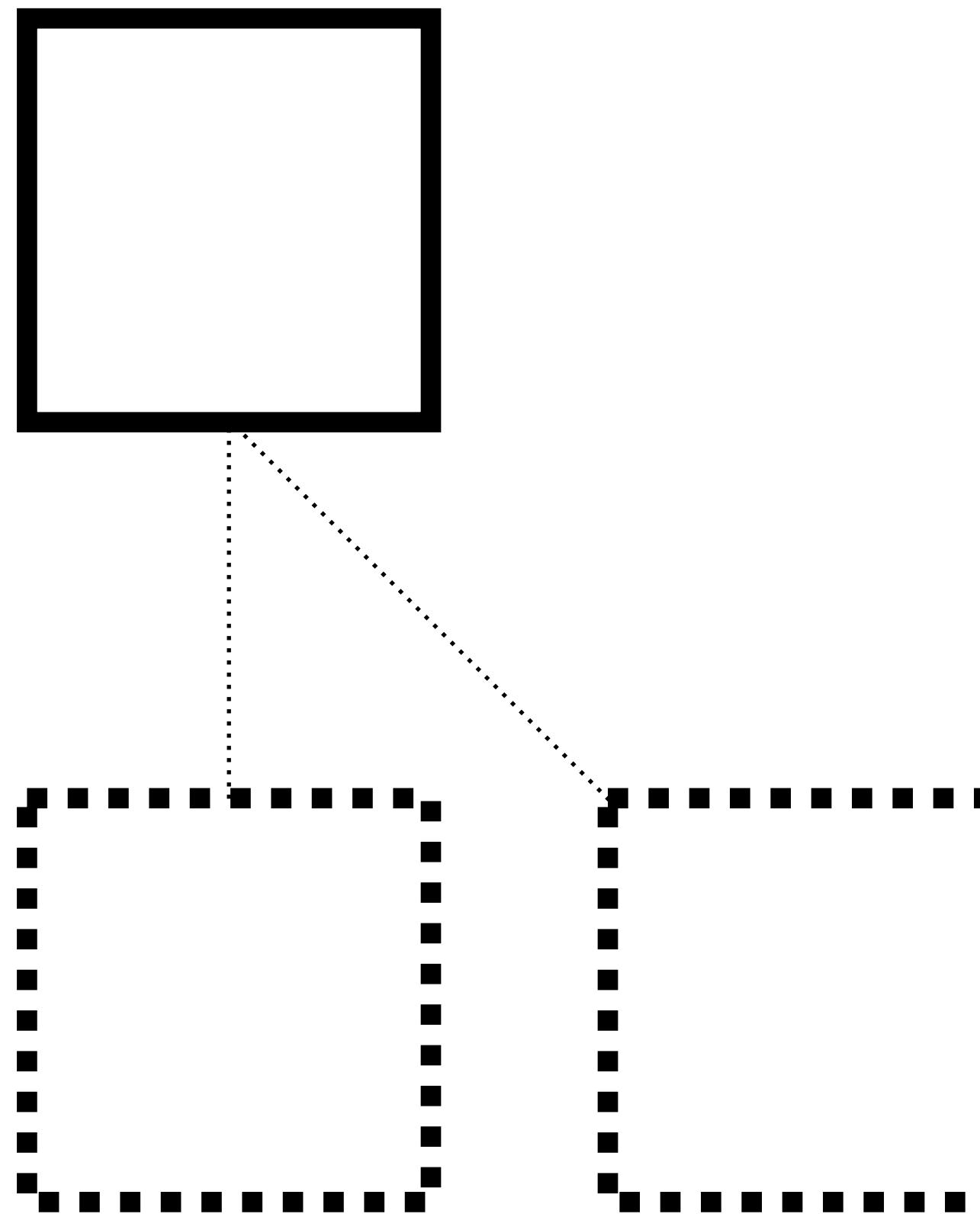
# 3 UTXOs

Add 1 more



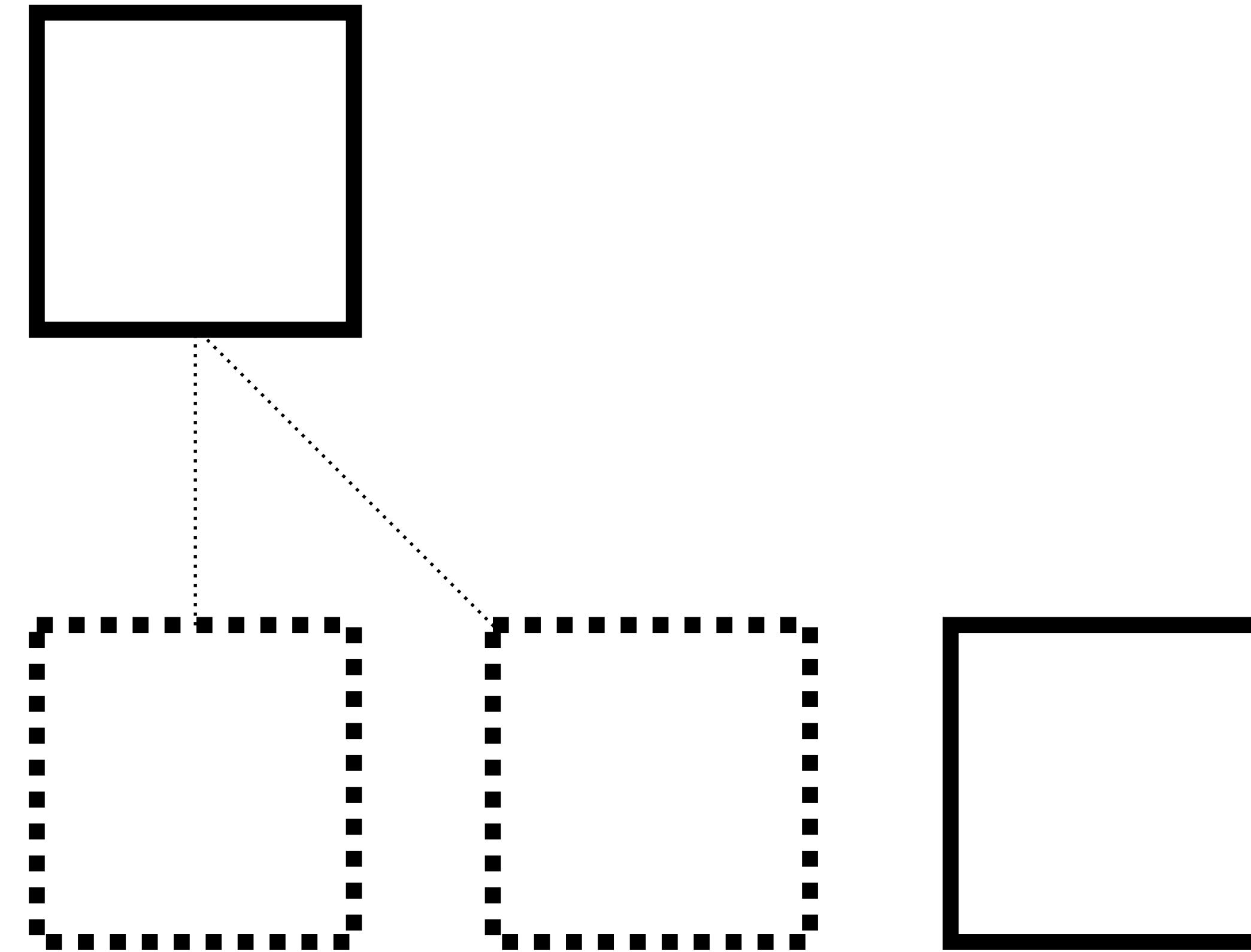
# 3 UTXOs

Add 1 more



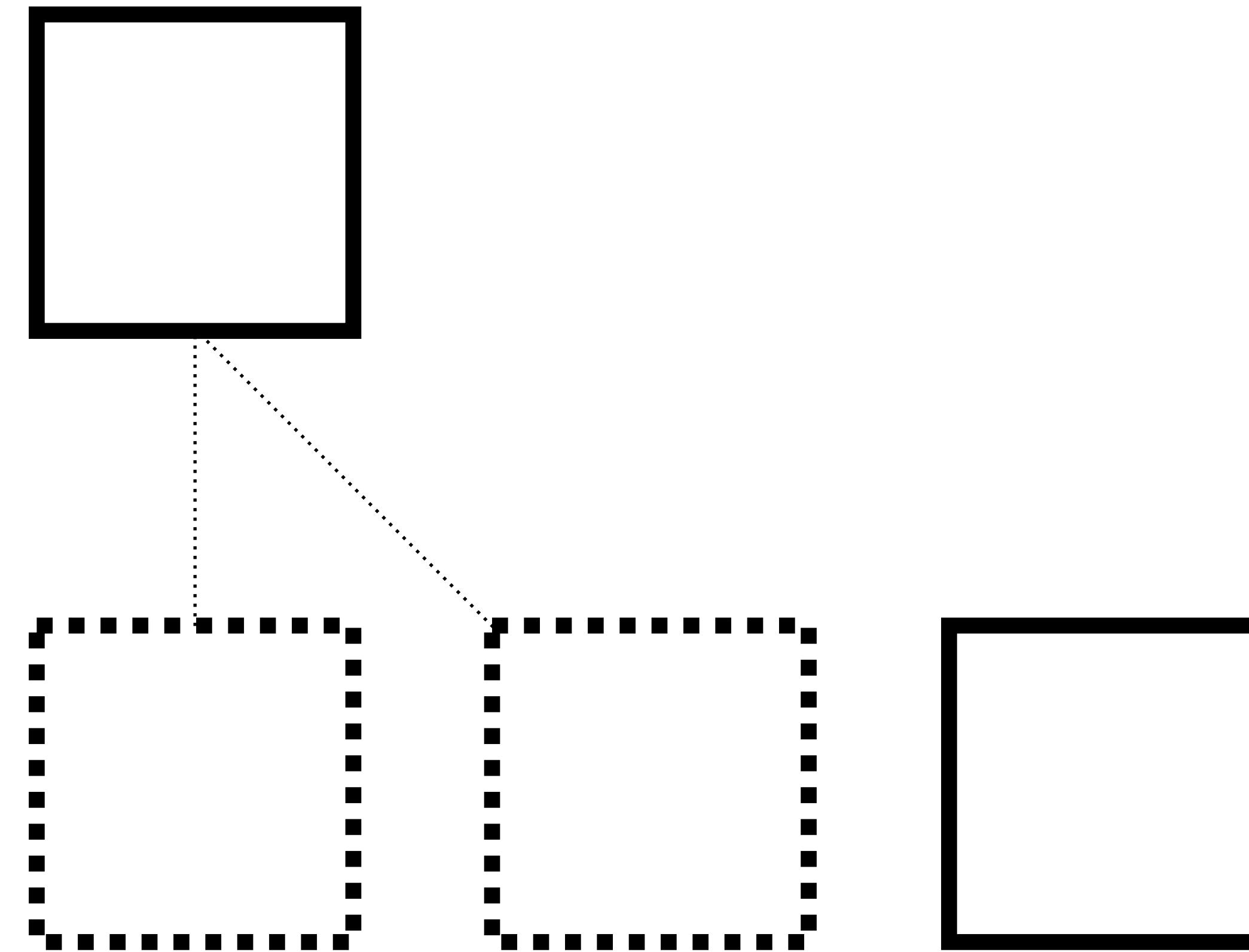
# 3 UTXOs

Add 1 more



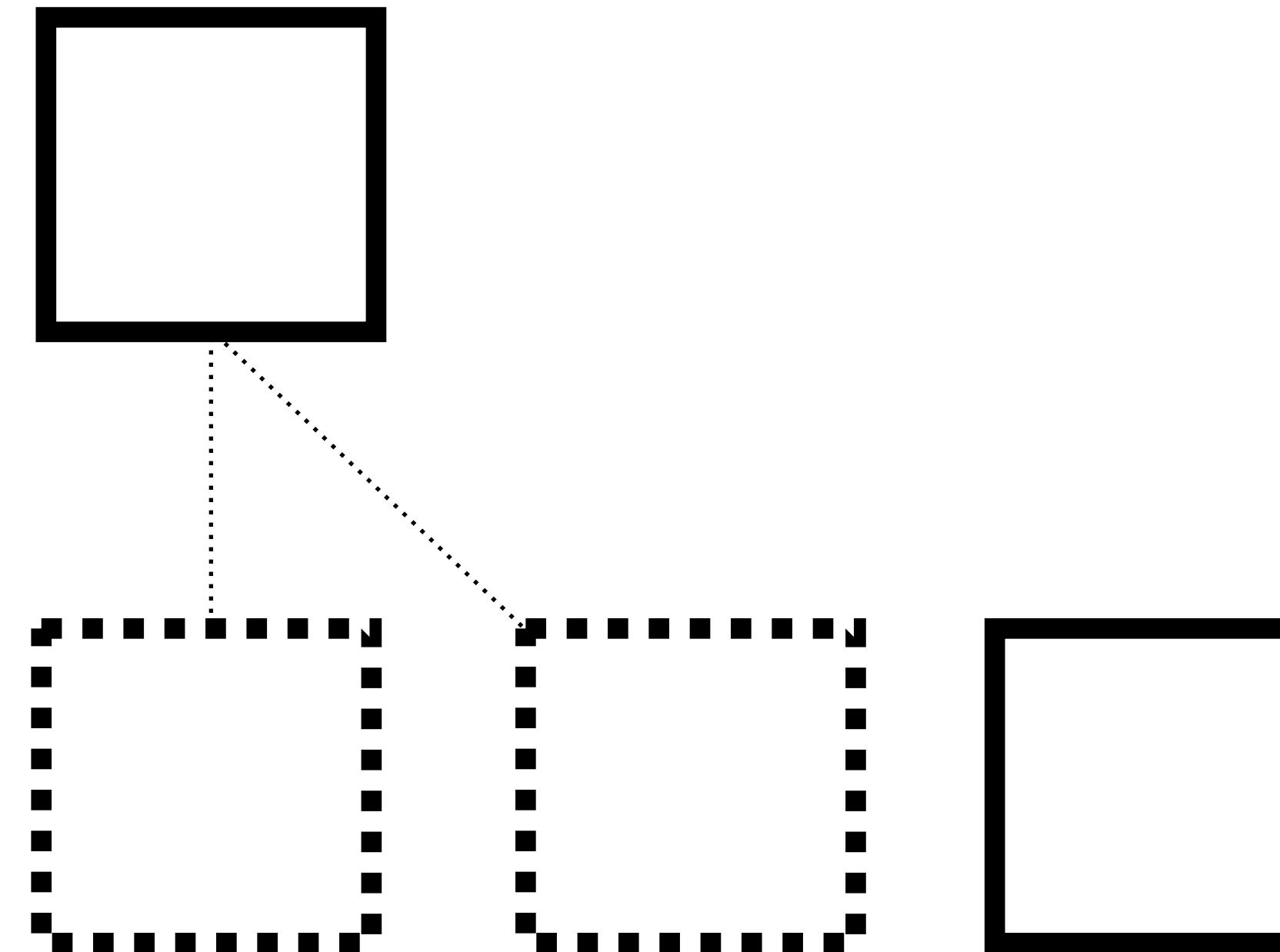
# 3 UTXOs

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



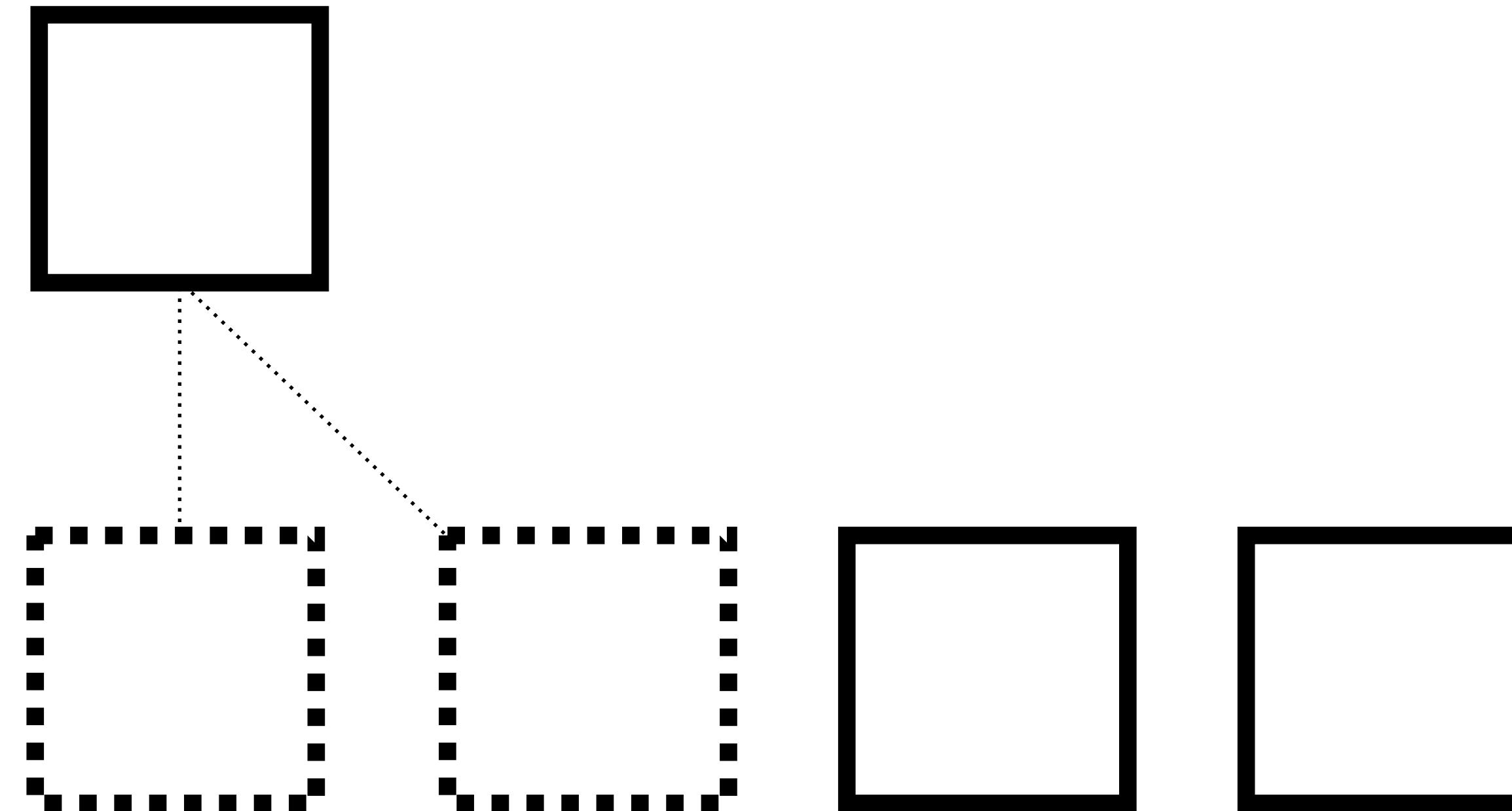
# 4 UTXOs

Another one

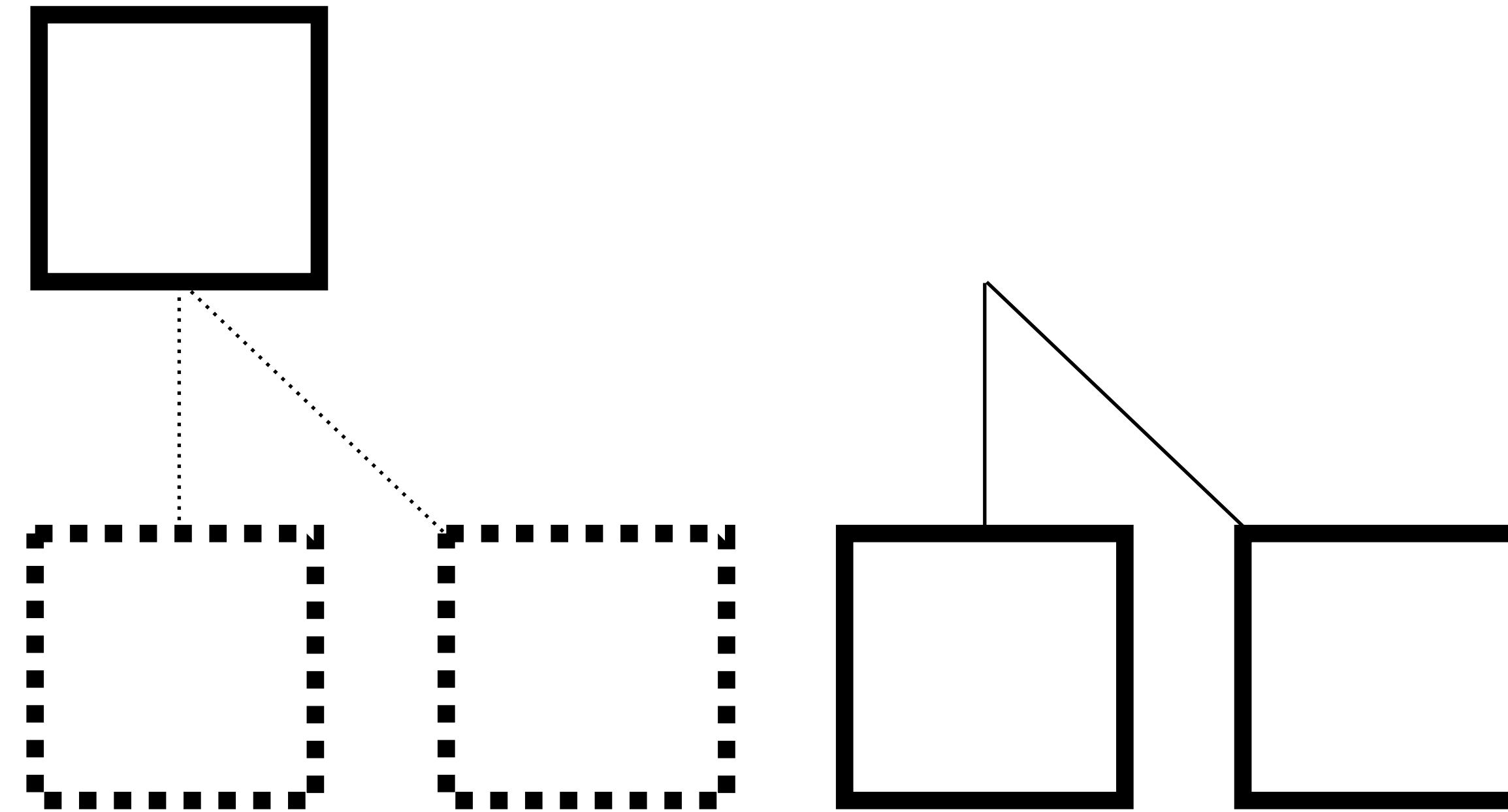


# 4 UTXOs

Another one

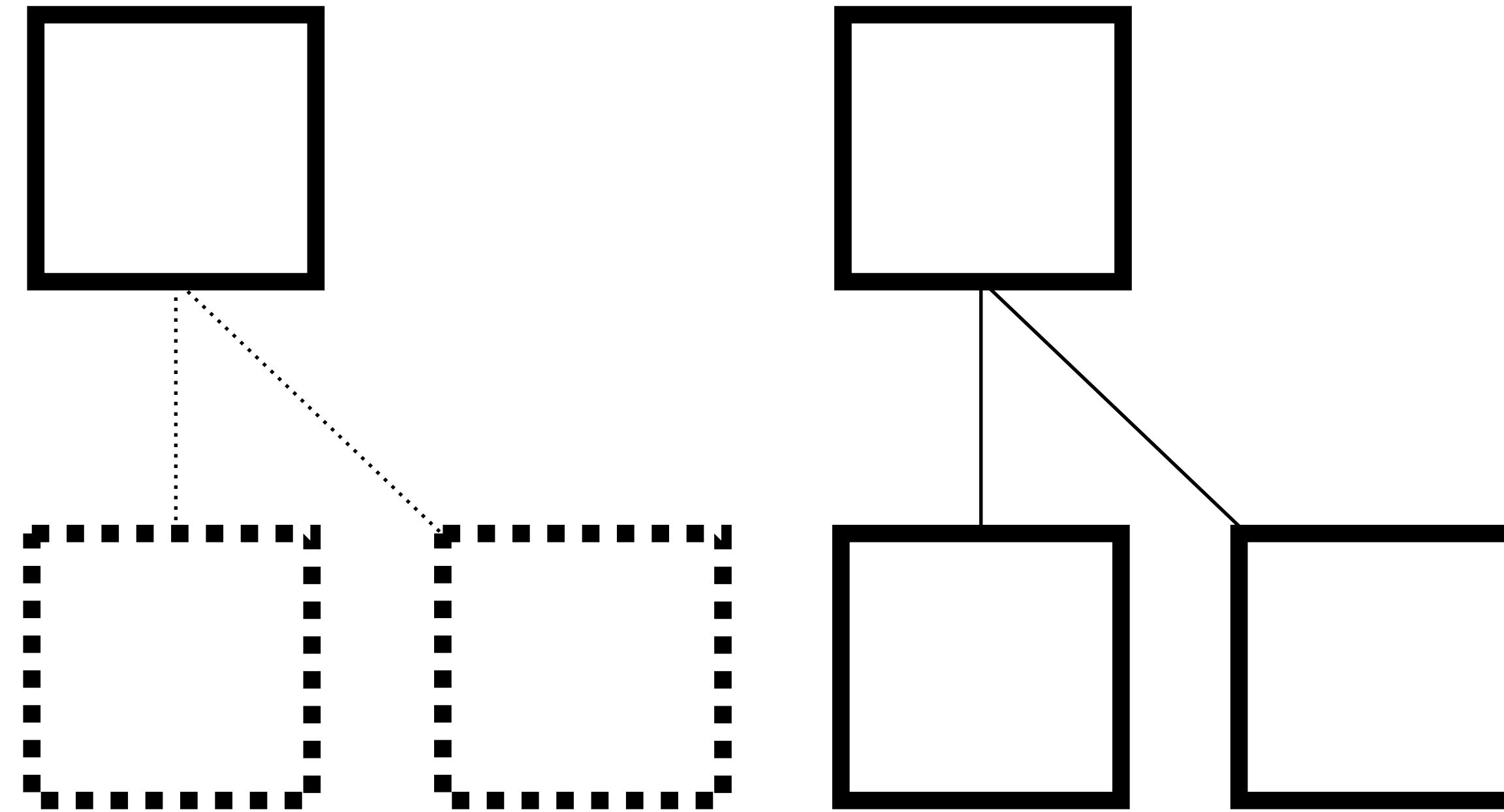


# 4 UTXOs Concatenate



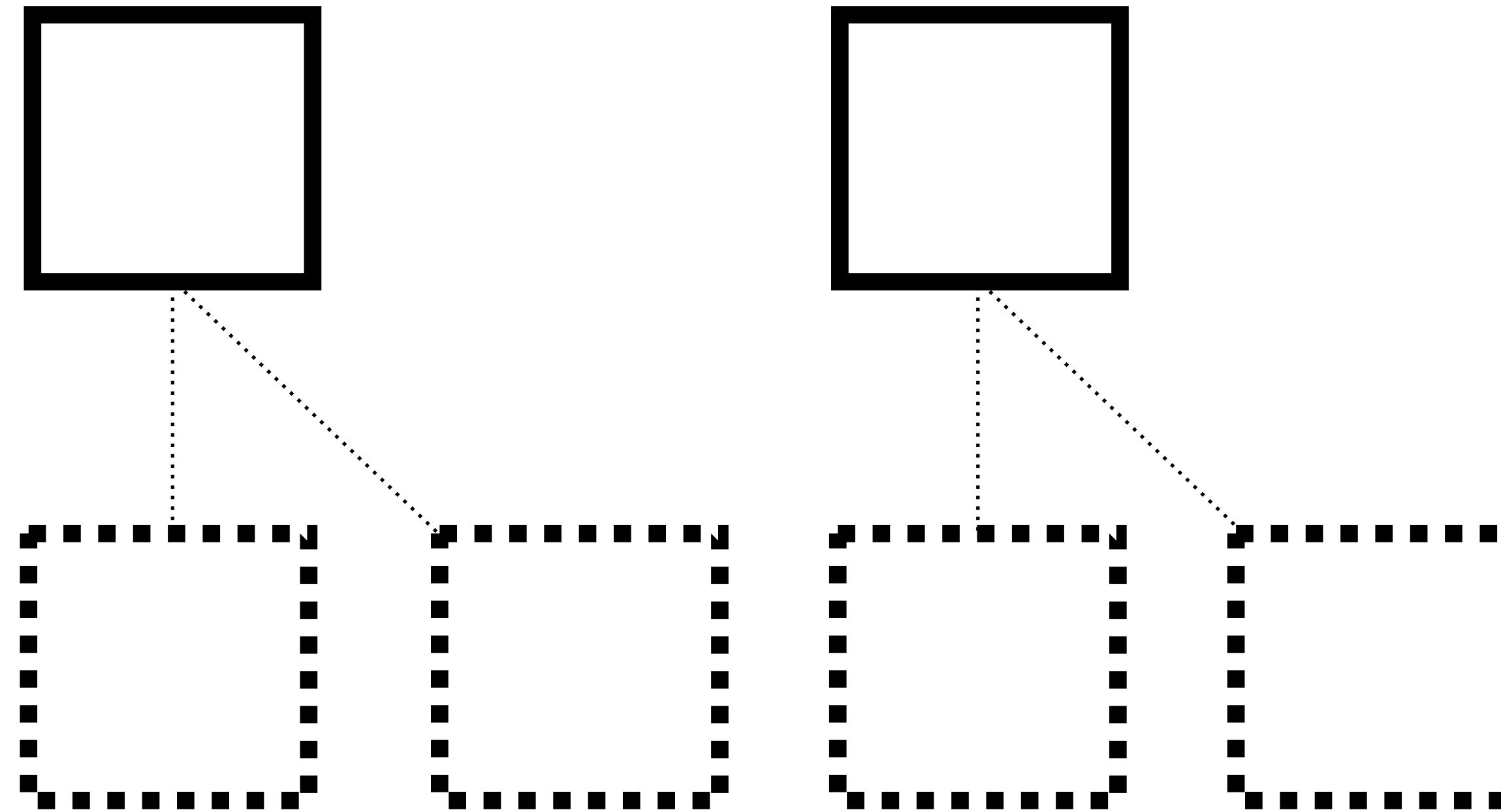
# 4 UTXOs

## Hash

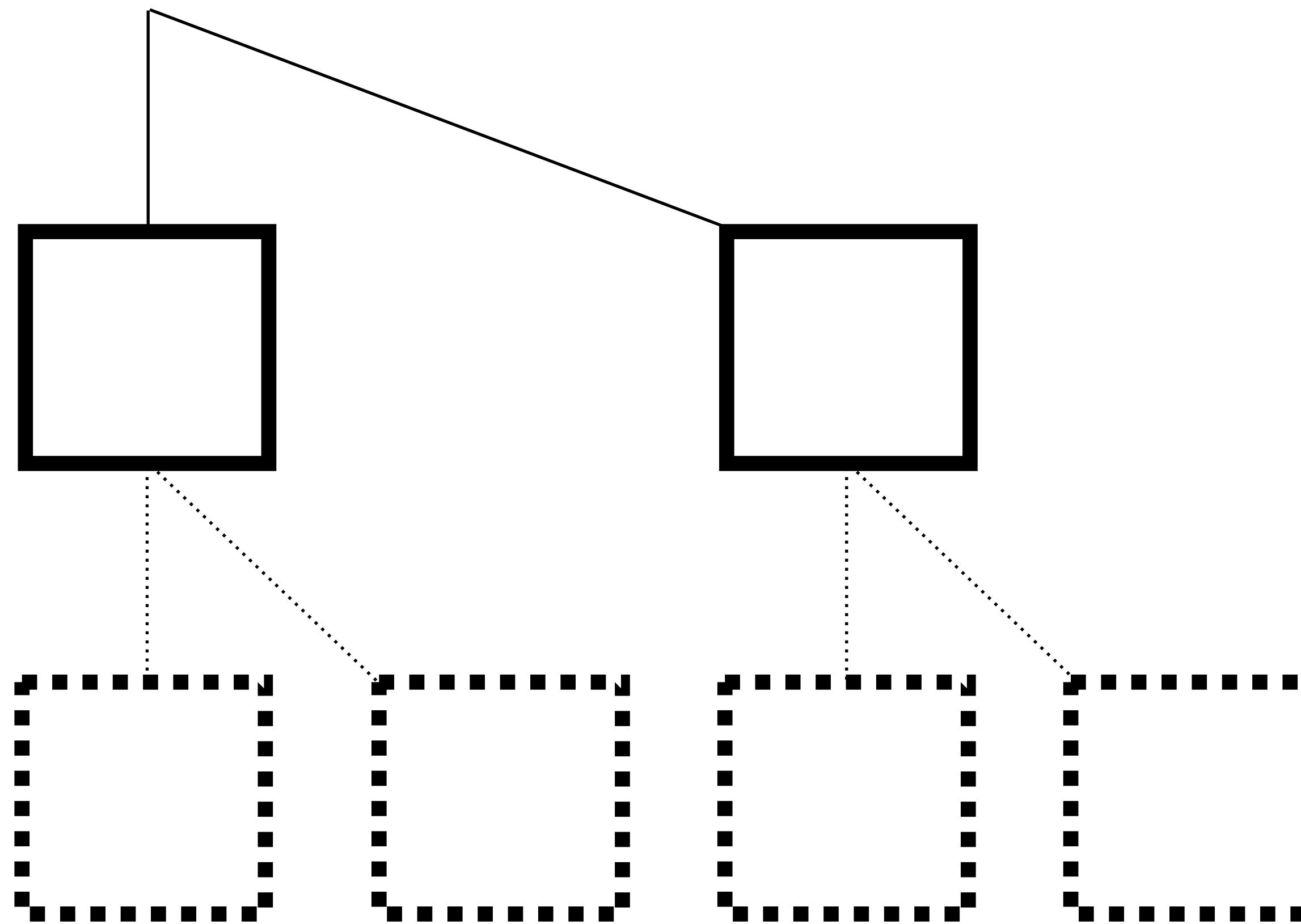


# 4 UTXOs

Can now throw away the leaves

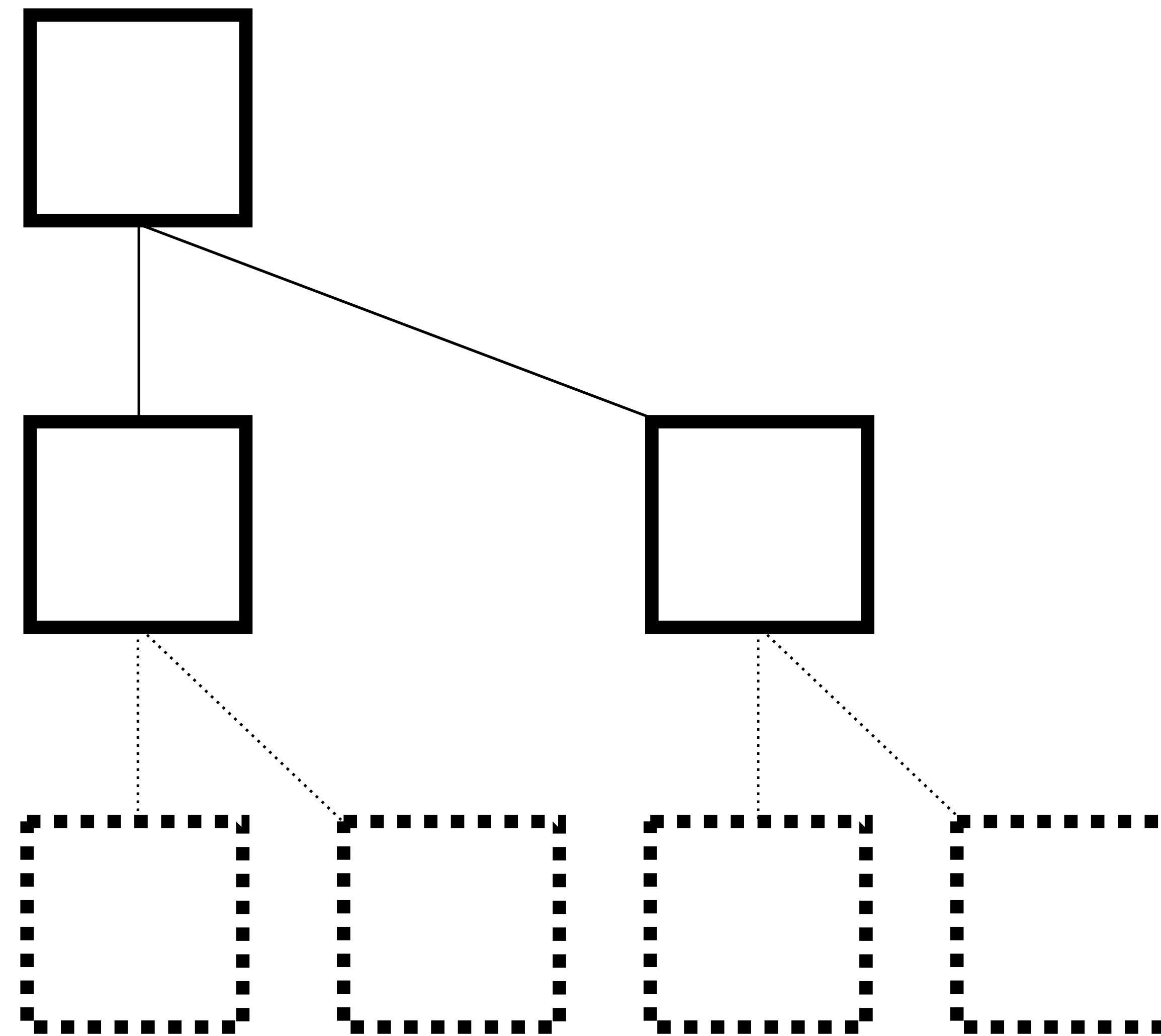


# 4 UTXOs Concatenate



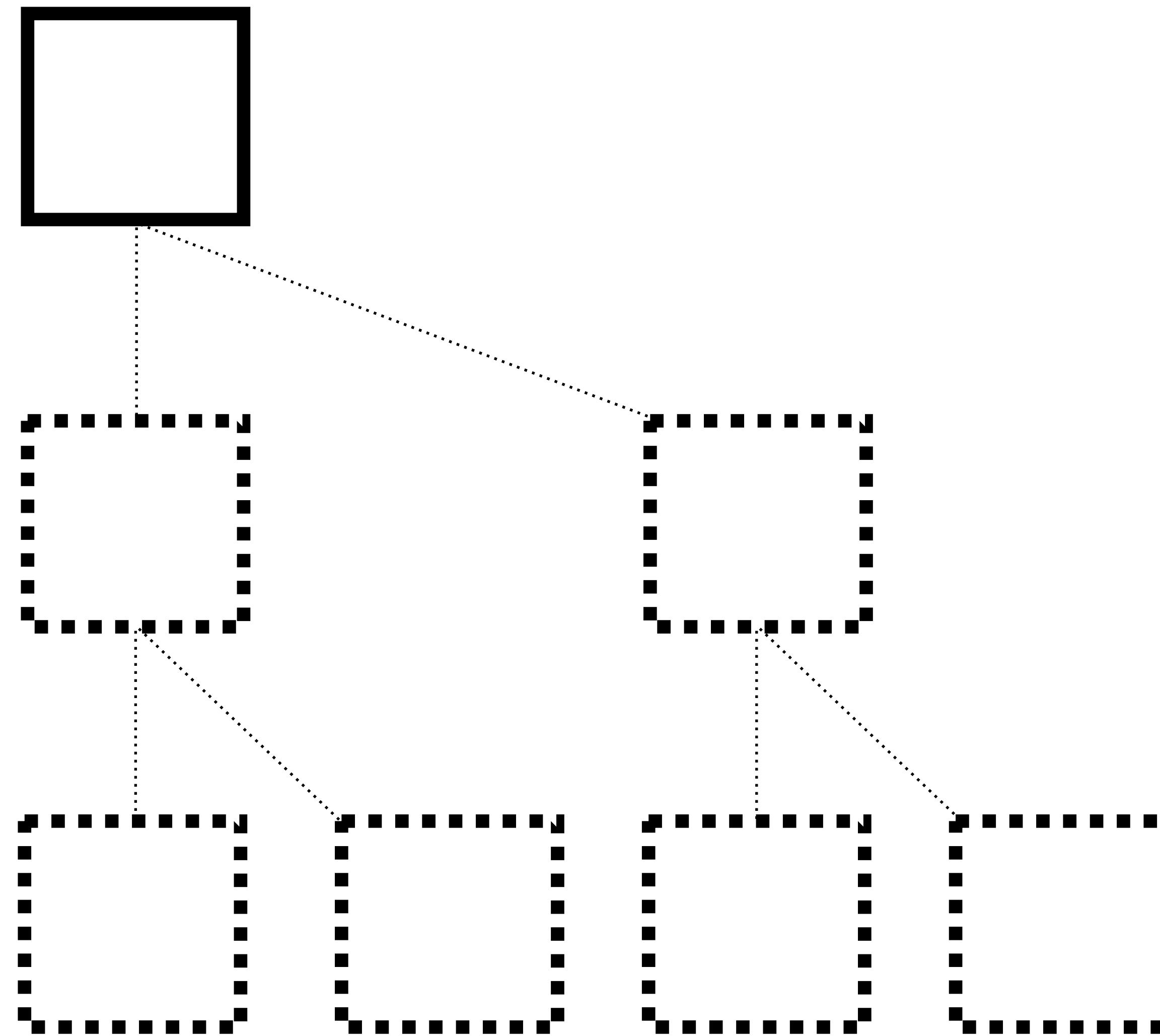
# 4 UTXOs

Hash



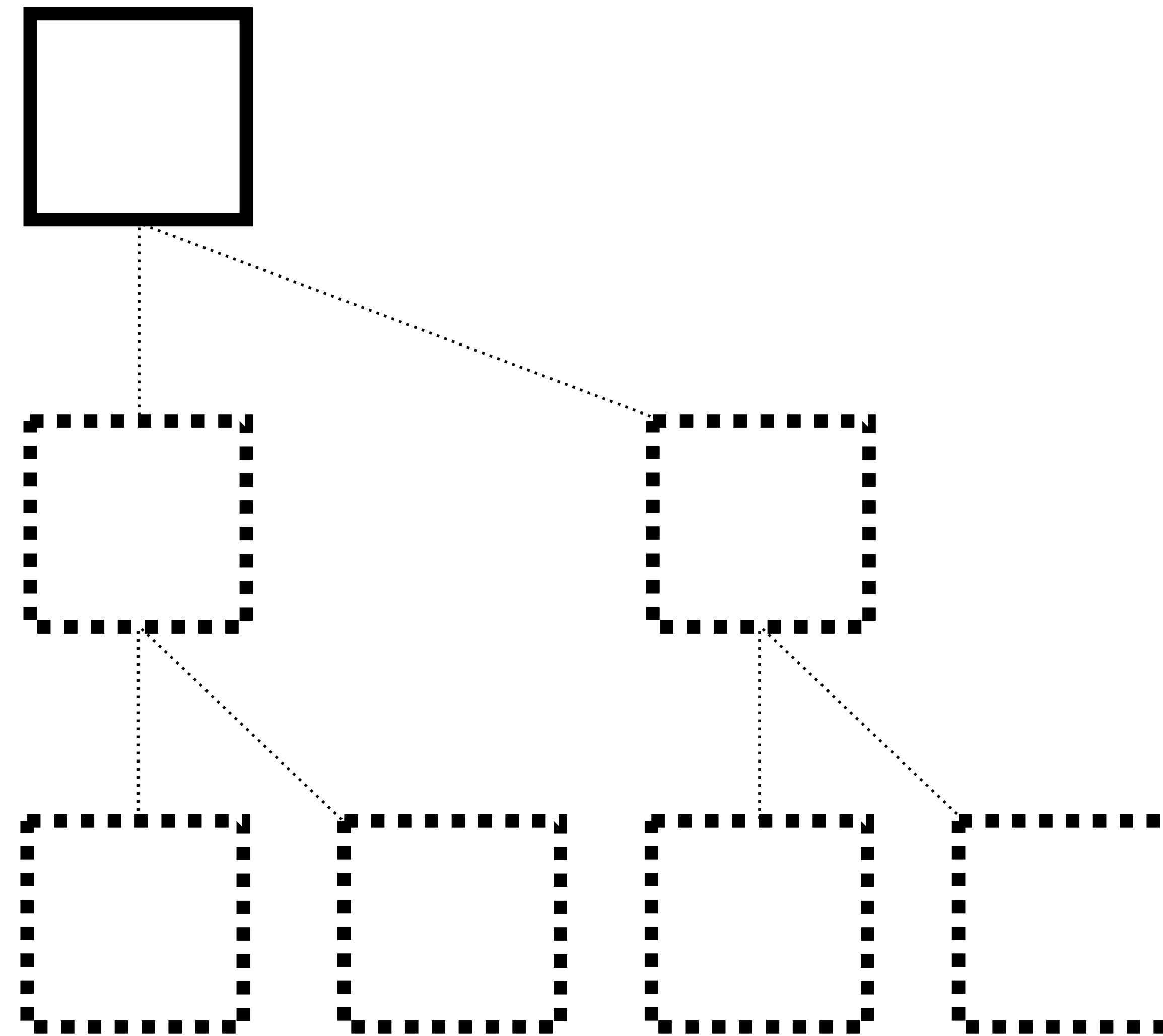
# 4 UTXOs

## Throw away



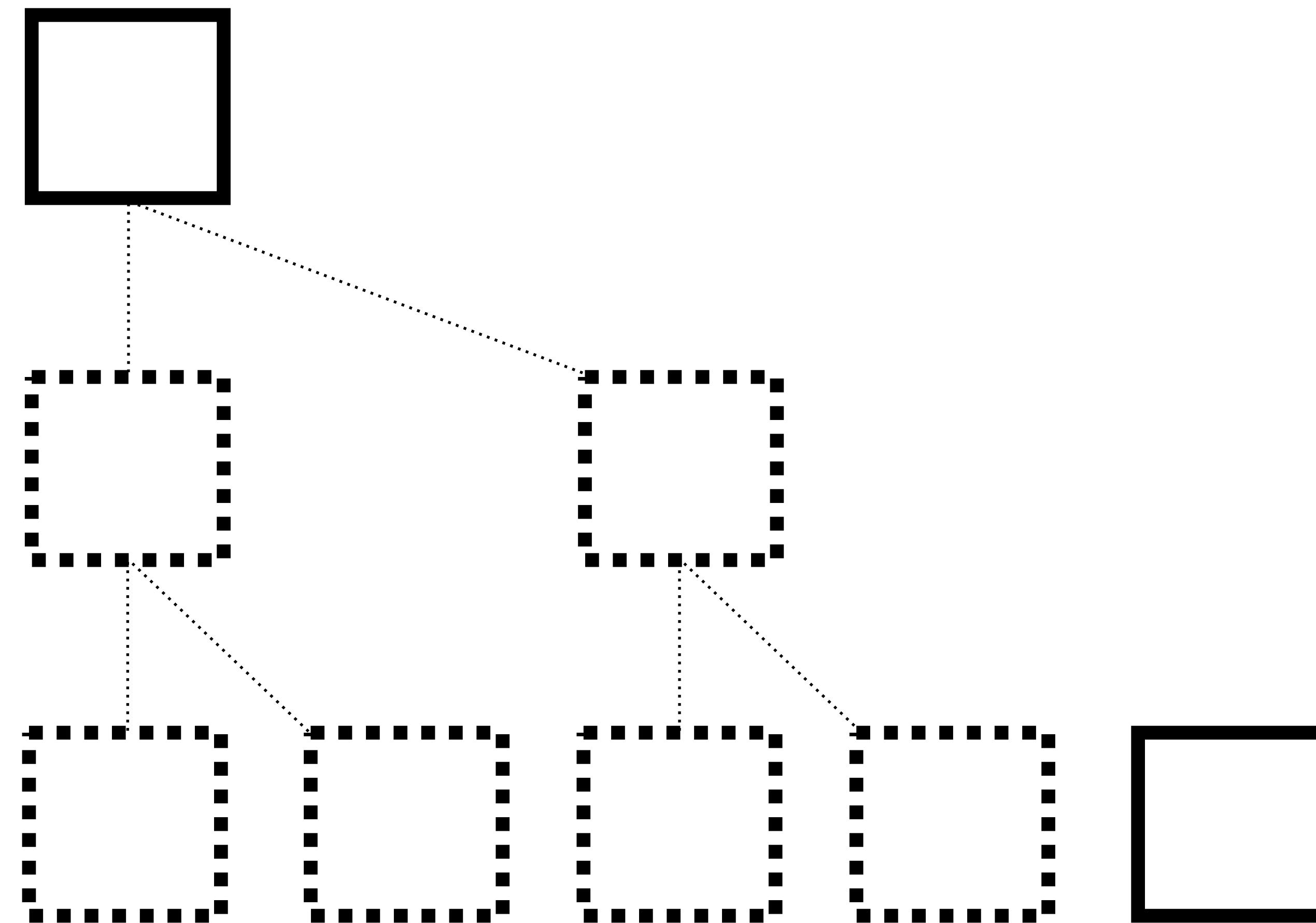
# 4 UTXOs

## Finished tree



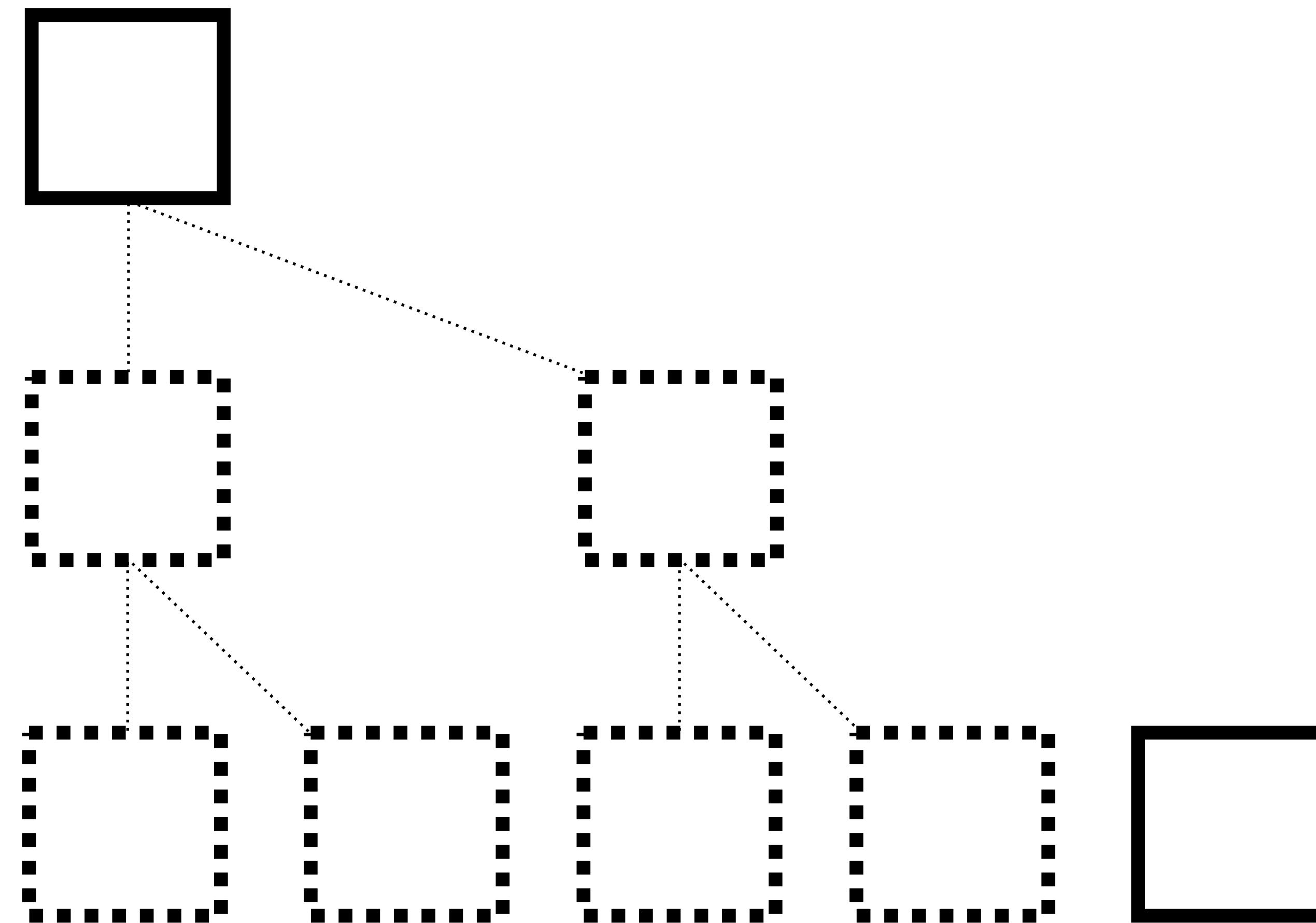
# 5 UTXOs

Add one more



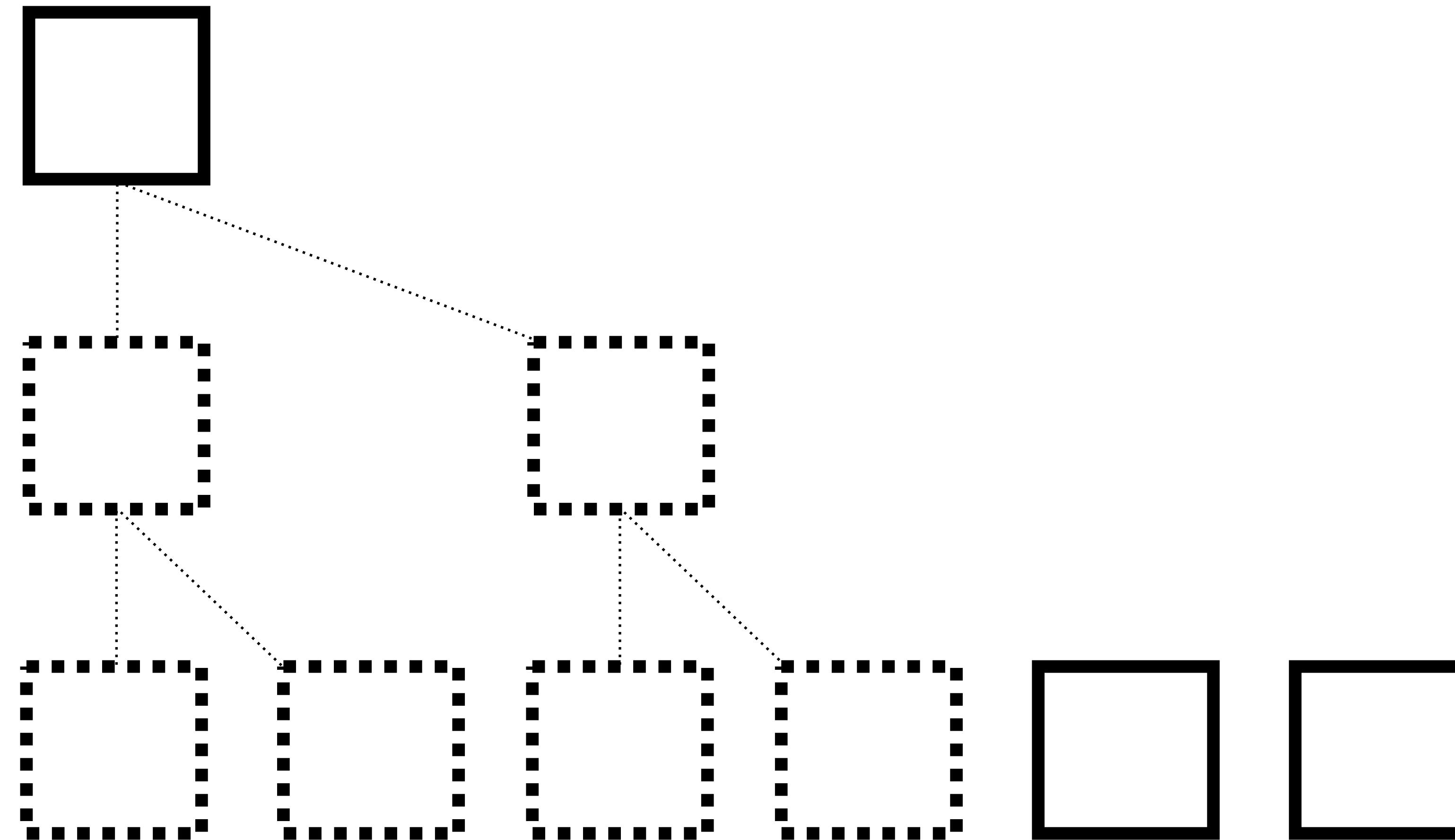
# 5 UTXOs

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

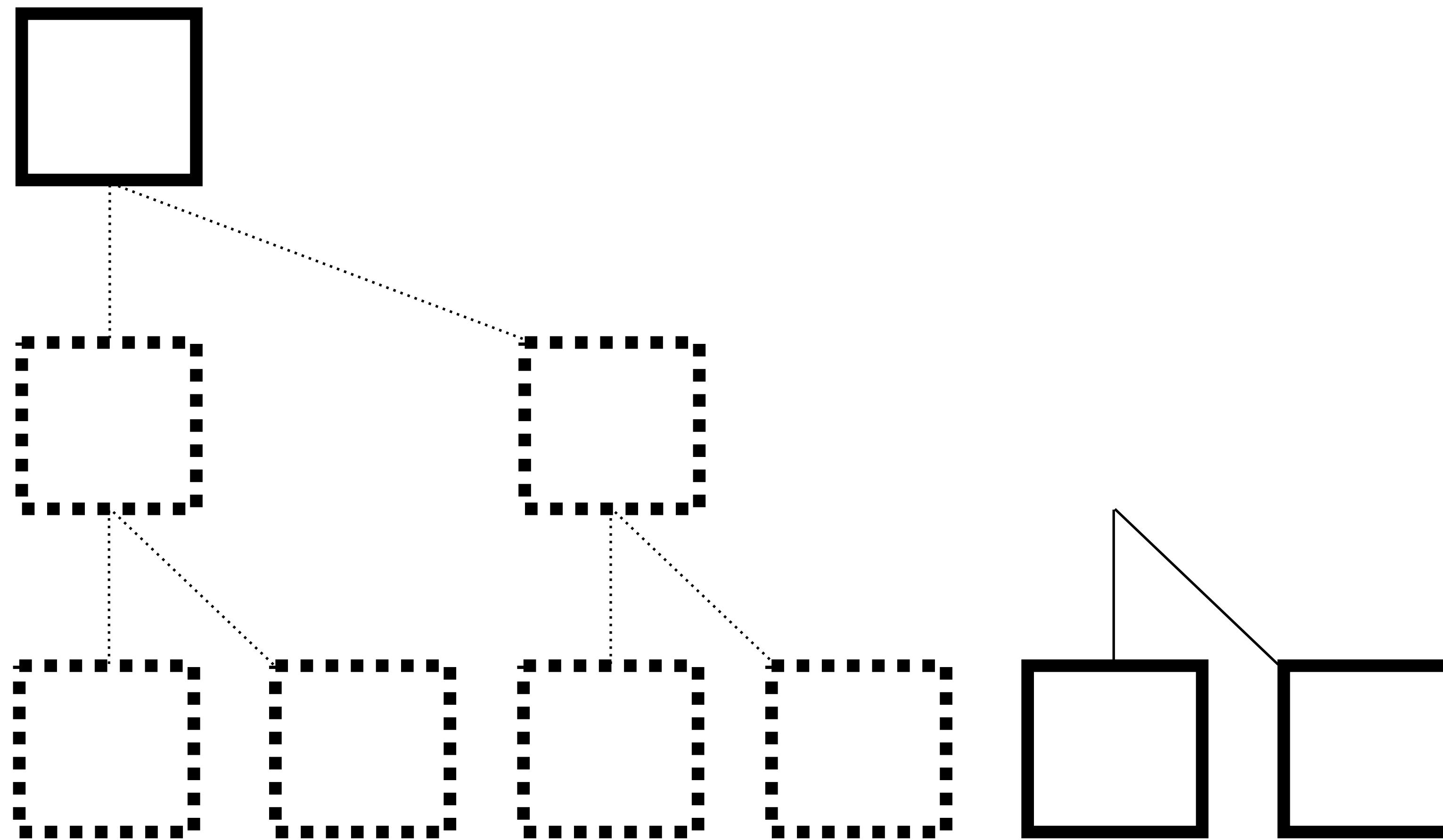


# 6 UTXOs

1 more

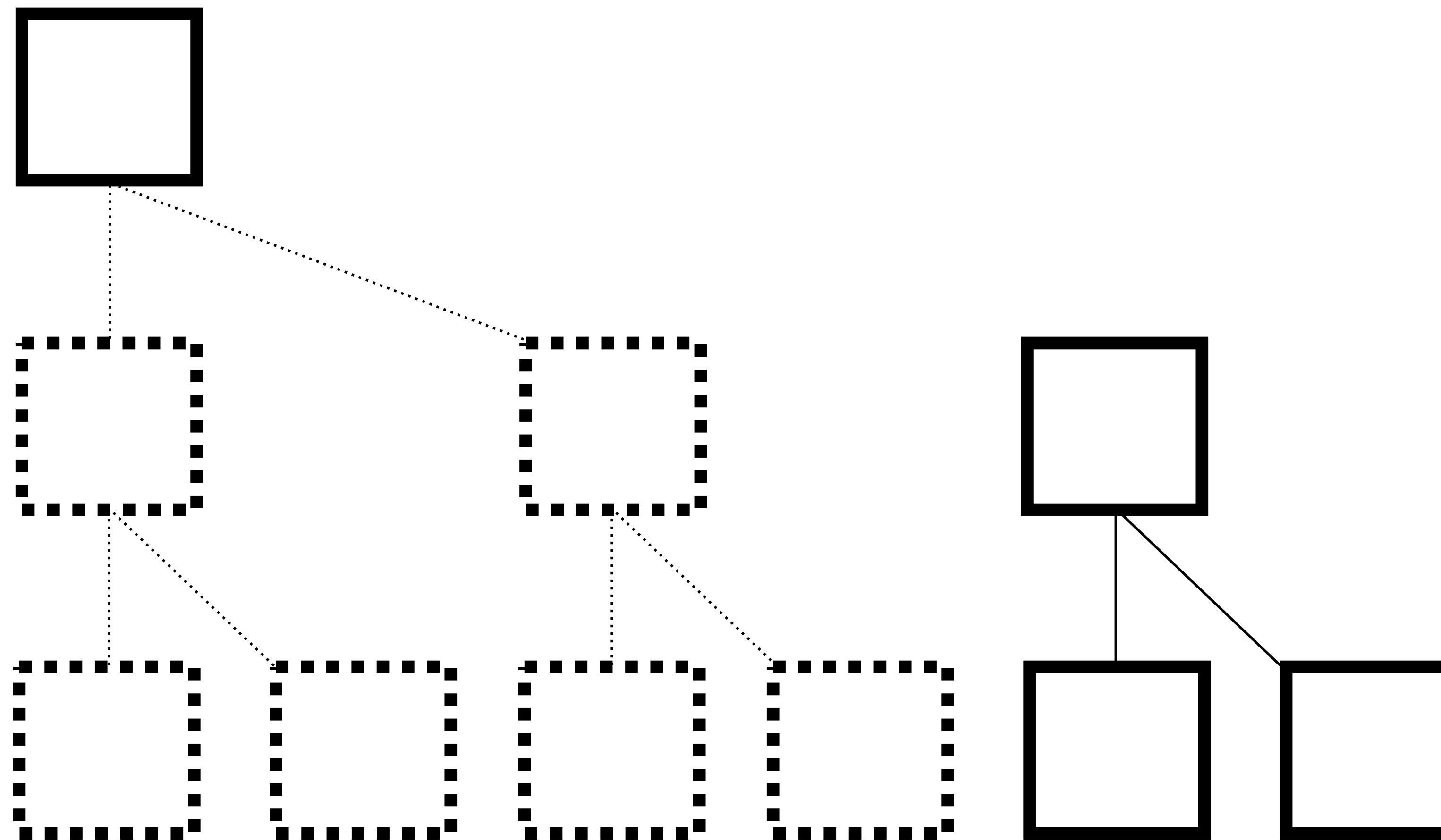


# 6 UTXOs Concatenate



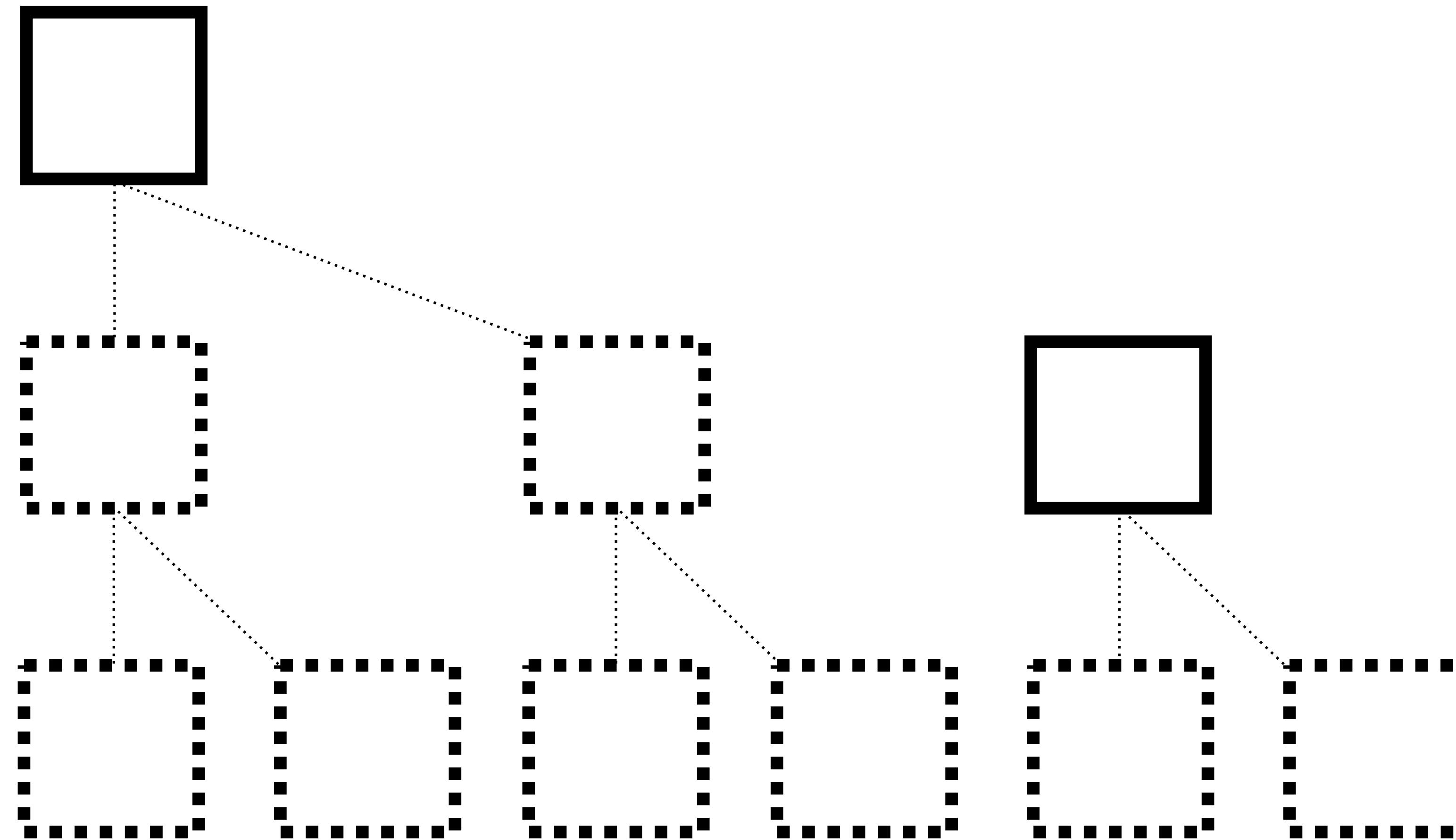
# 6 UTXOs

## Hash



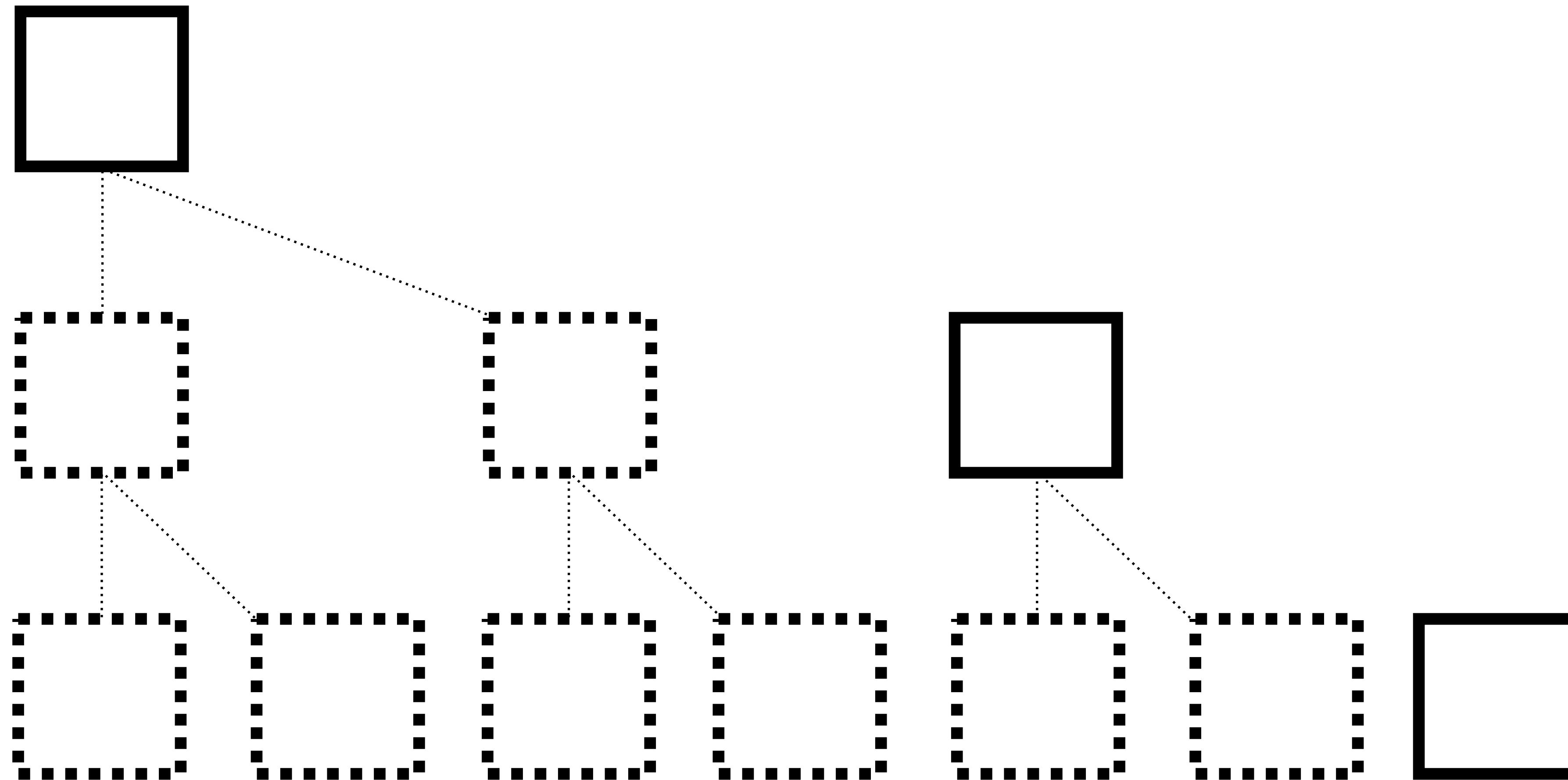
# 6 UTXOs

## Throw away



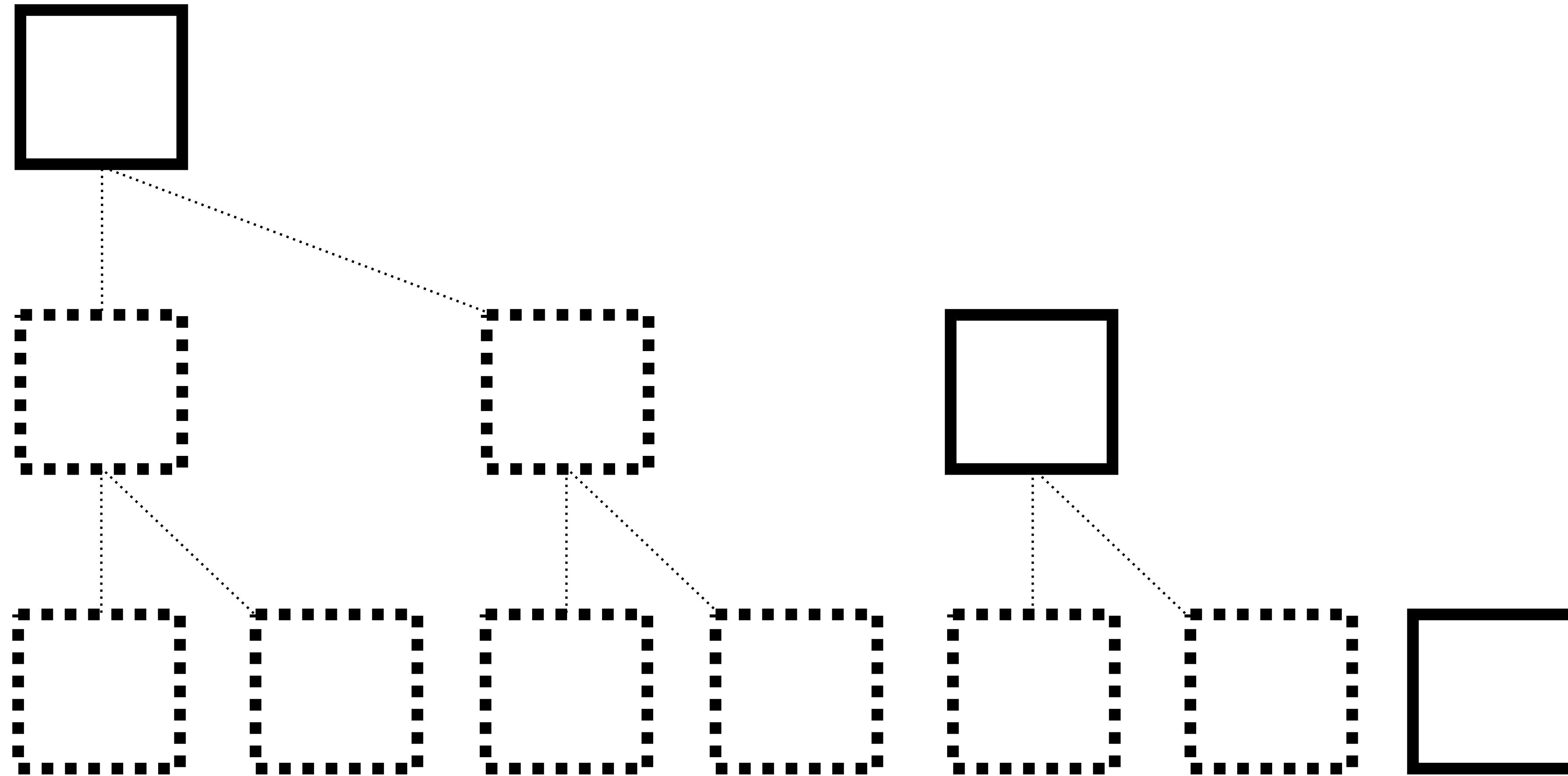
# 7 UTXOs

Another one



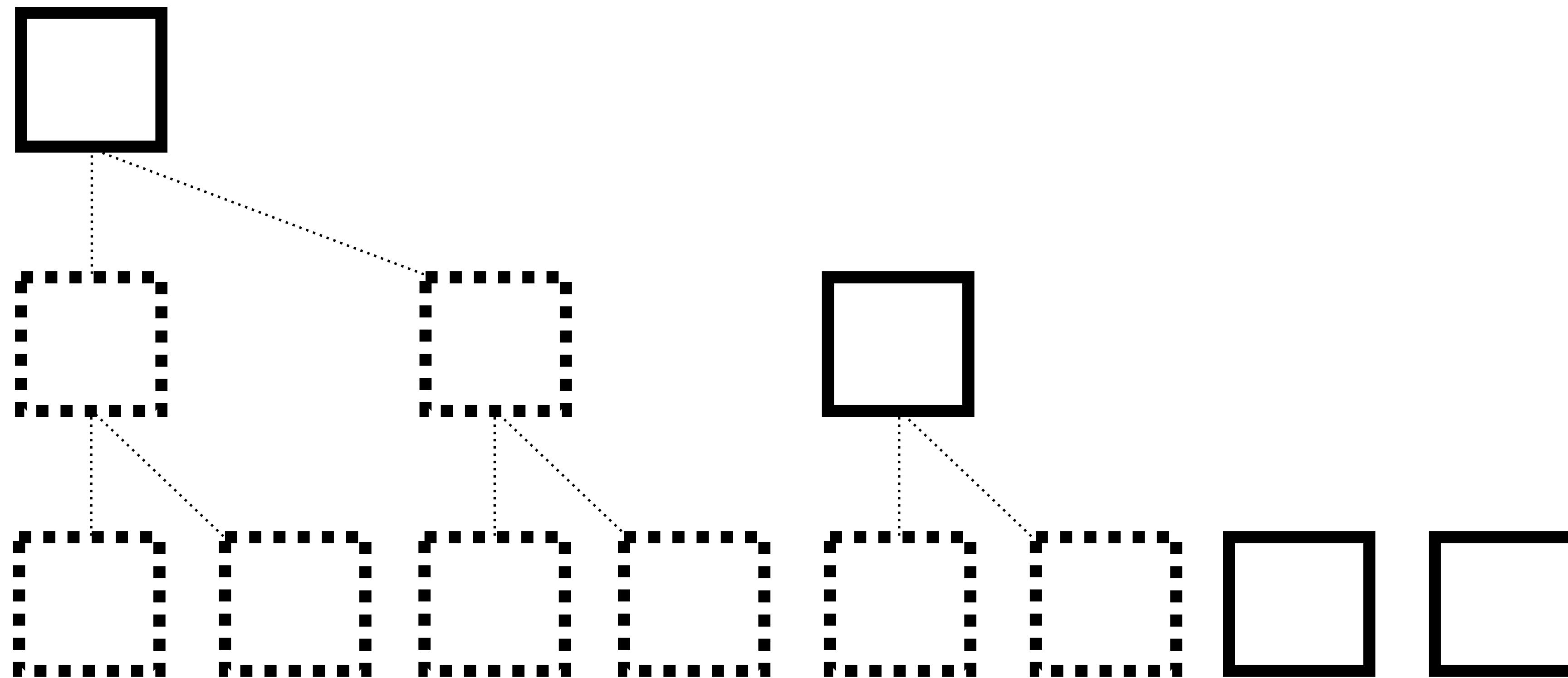
# 7 UTXOs

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



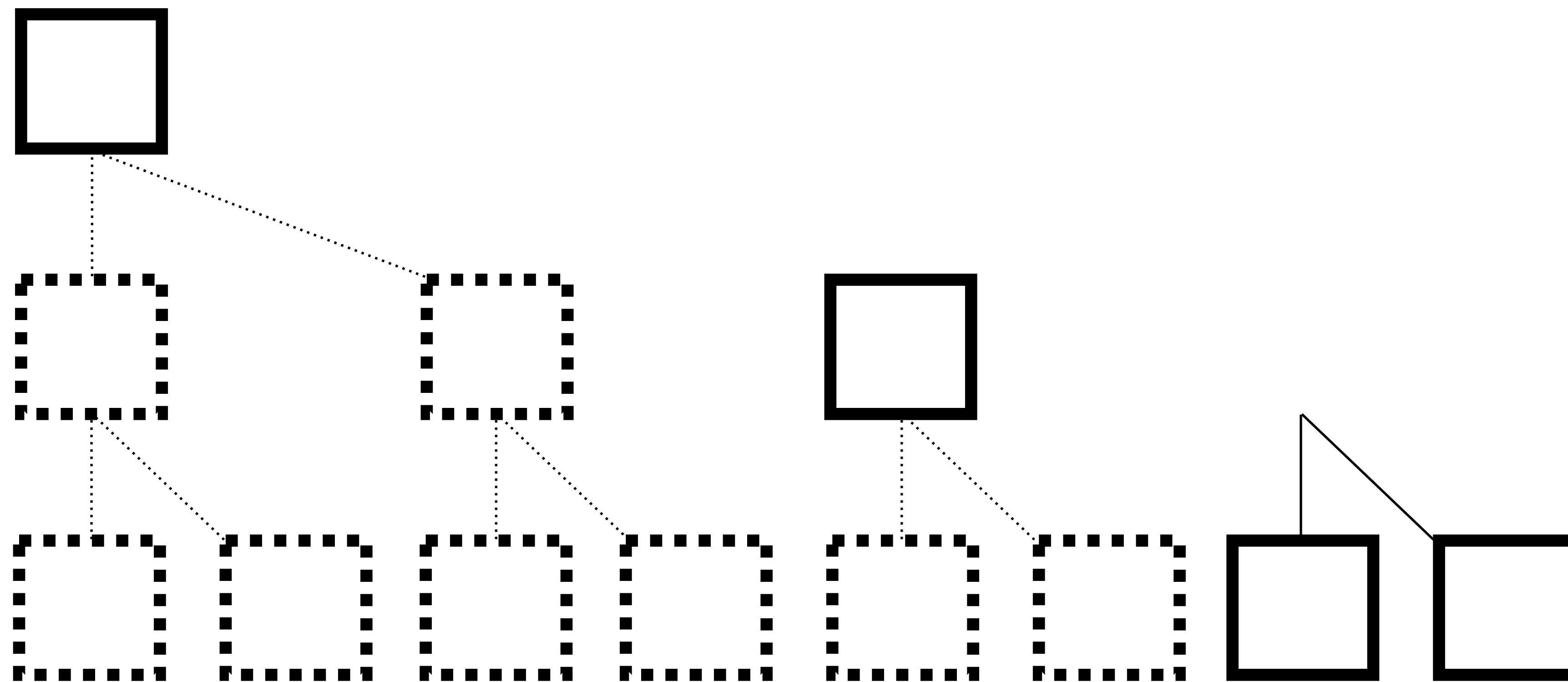
# 8 UTXOs

Another one



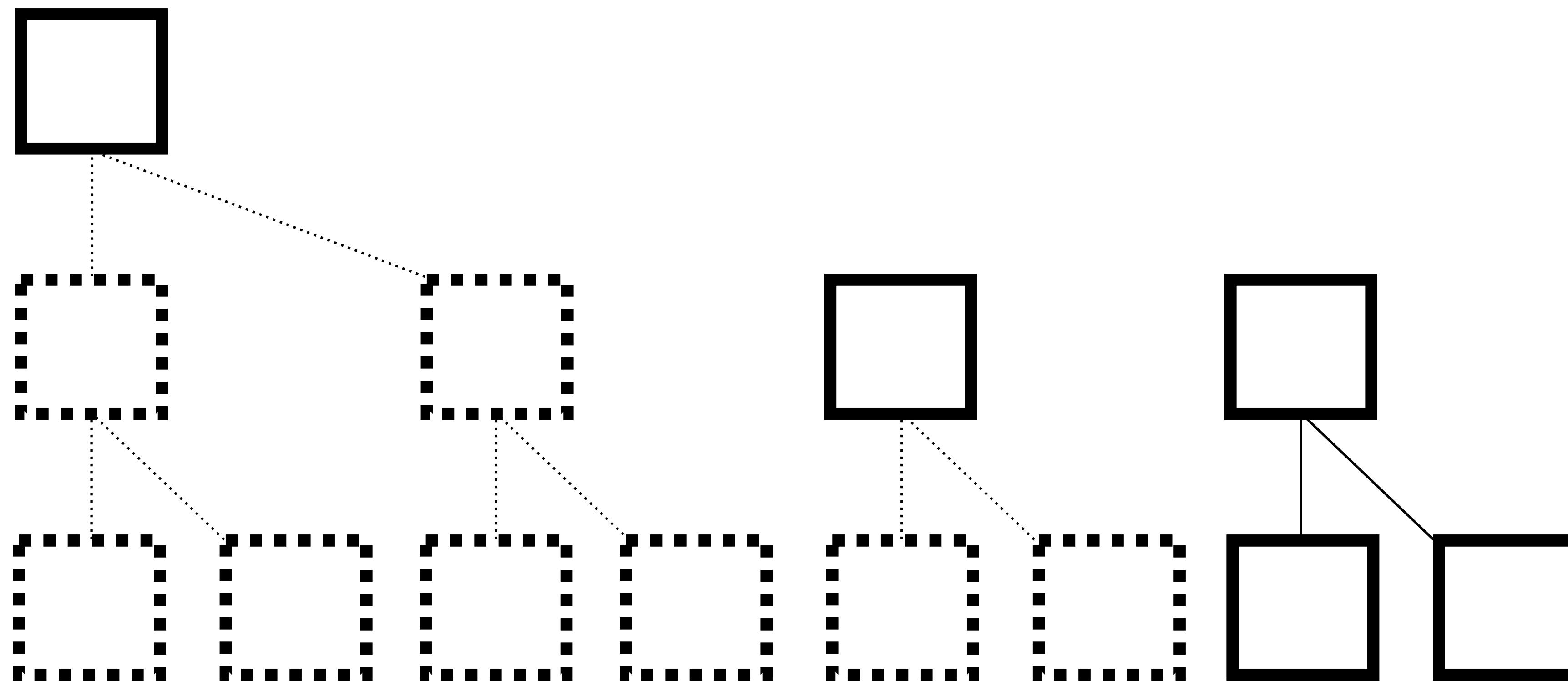
# 8 UTXOs

## Concatenate



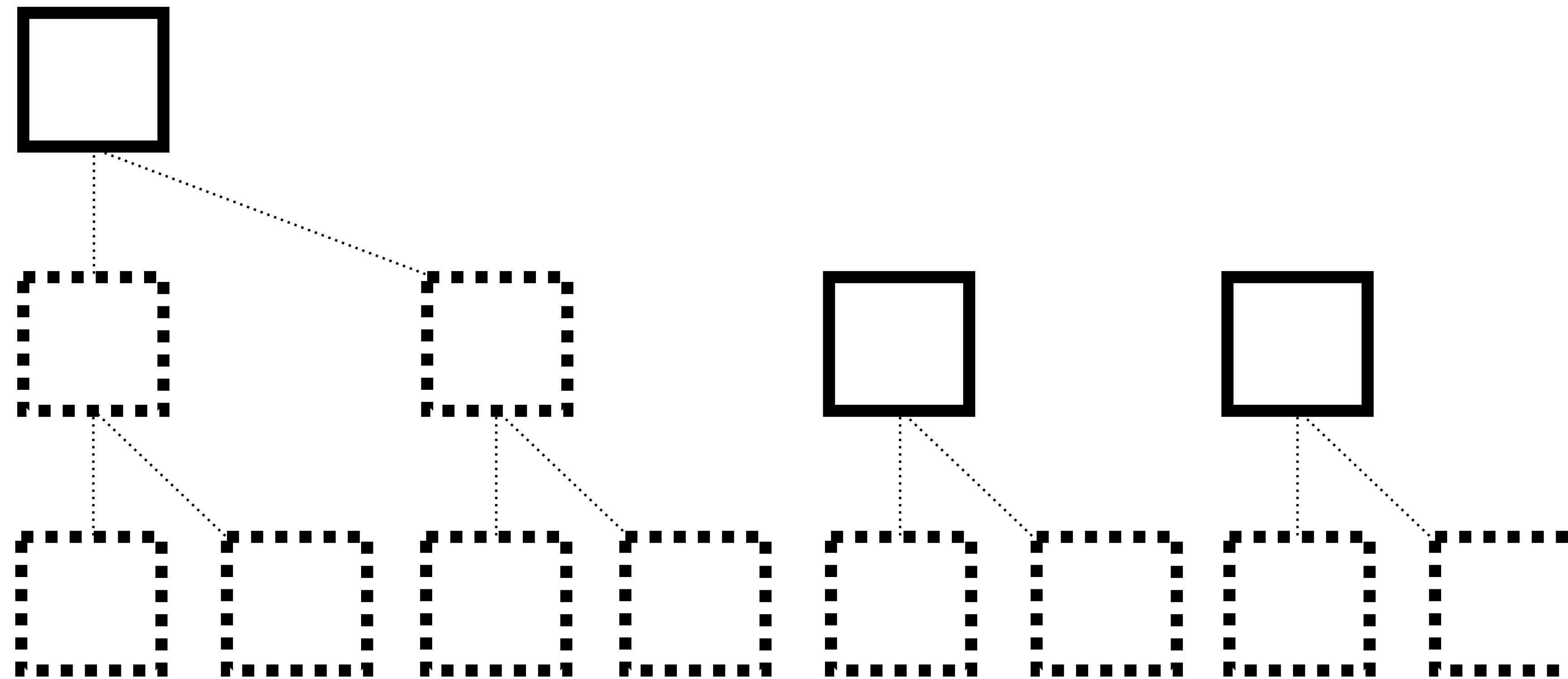
# 8 UTXOs

## Hash



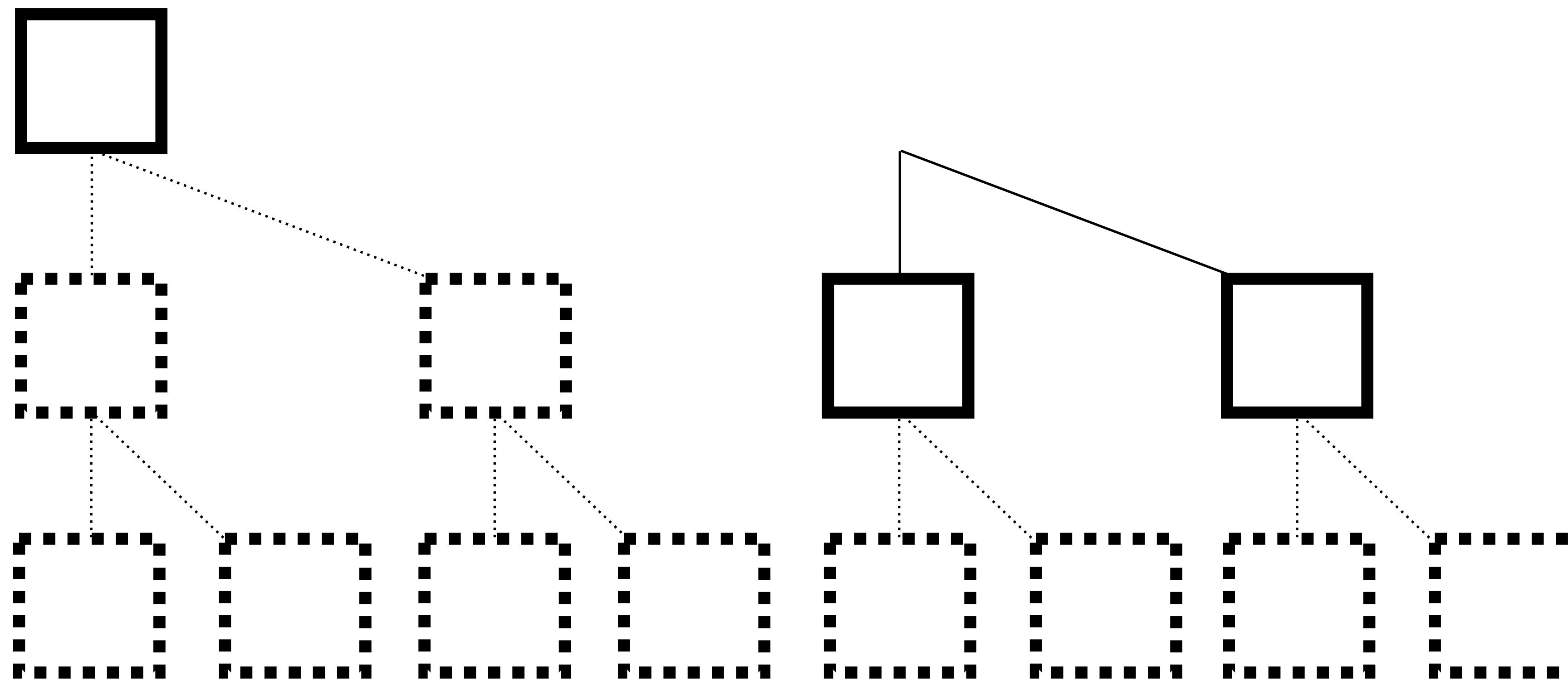
# 8 UTXOs

## Throw away



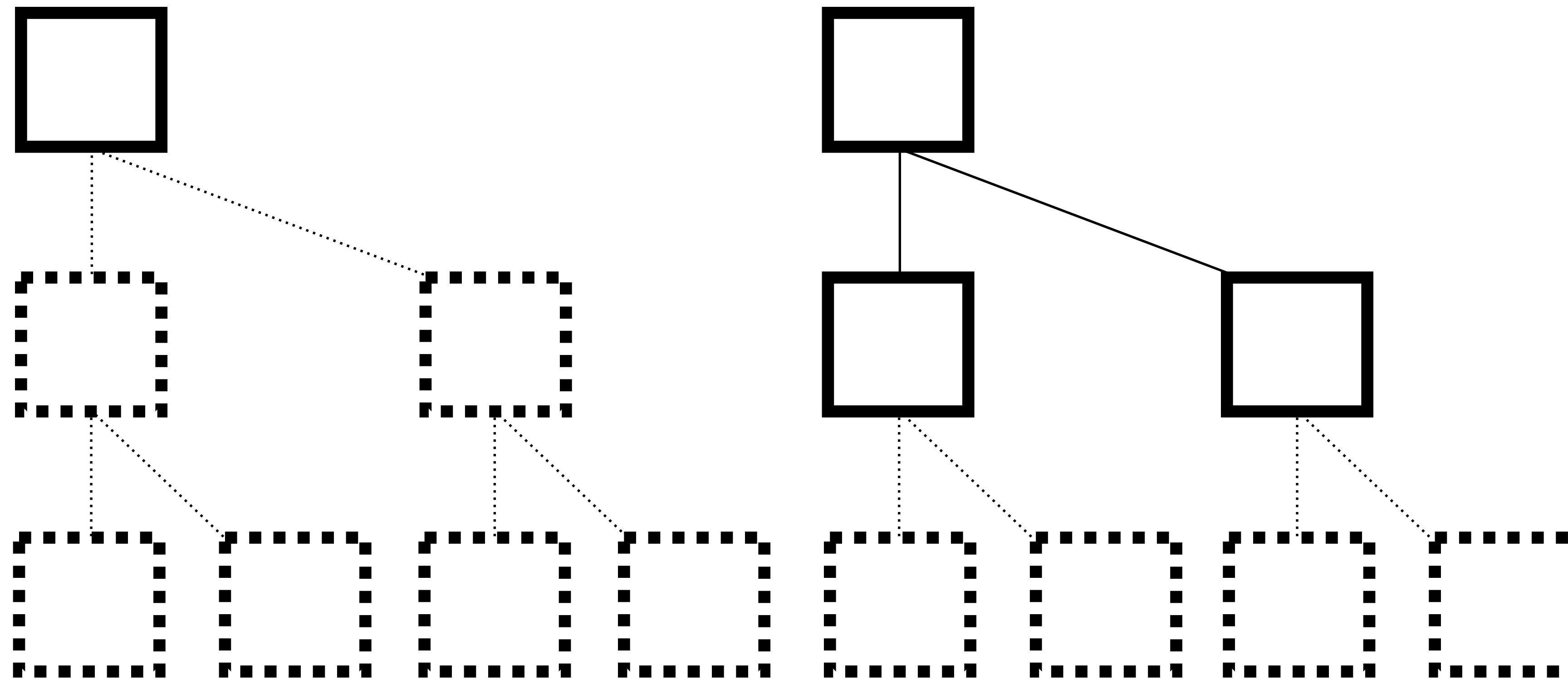
# 8 UTXOs

## Concatenate



# 8 UTXOs

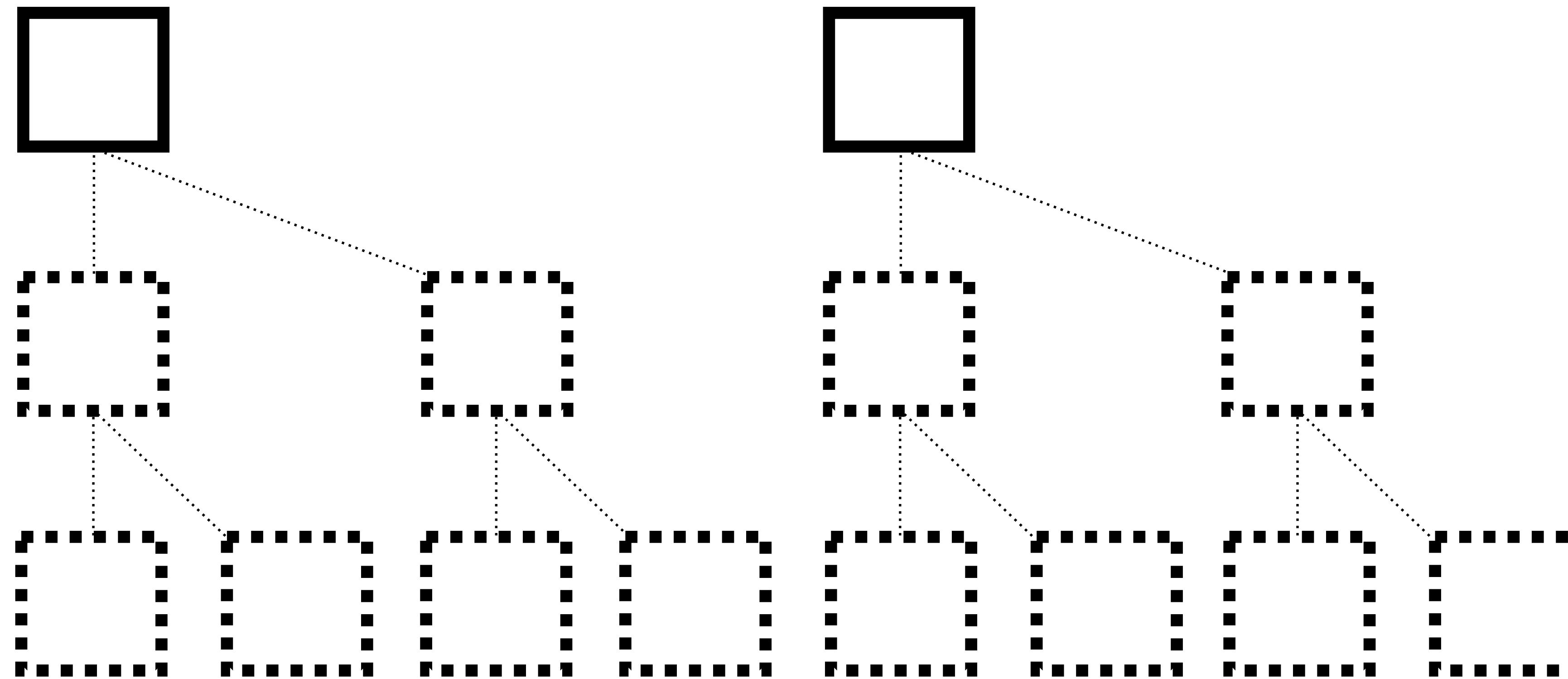
## Hash



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

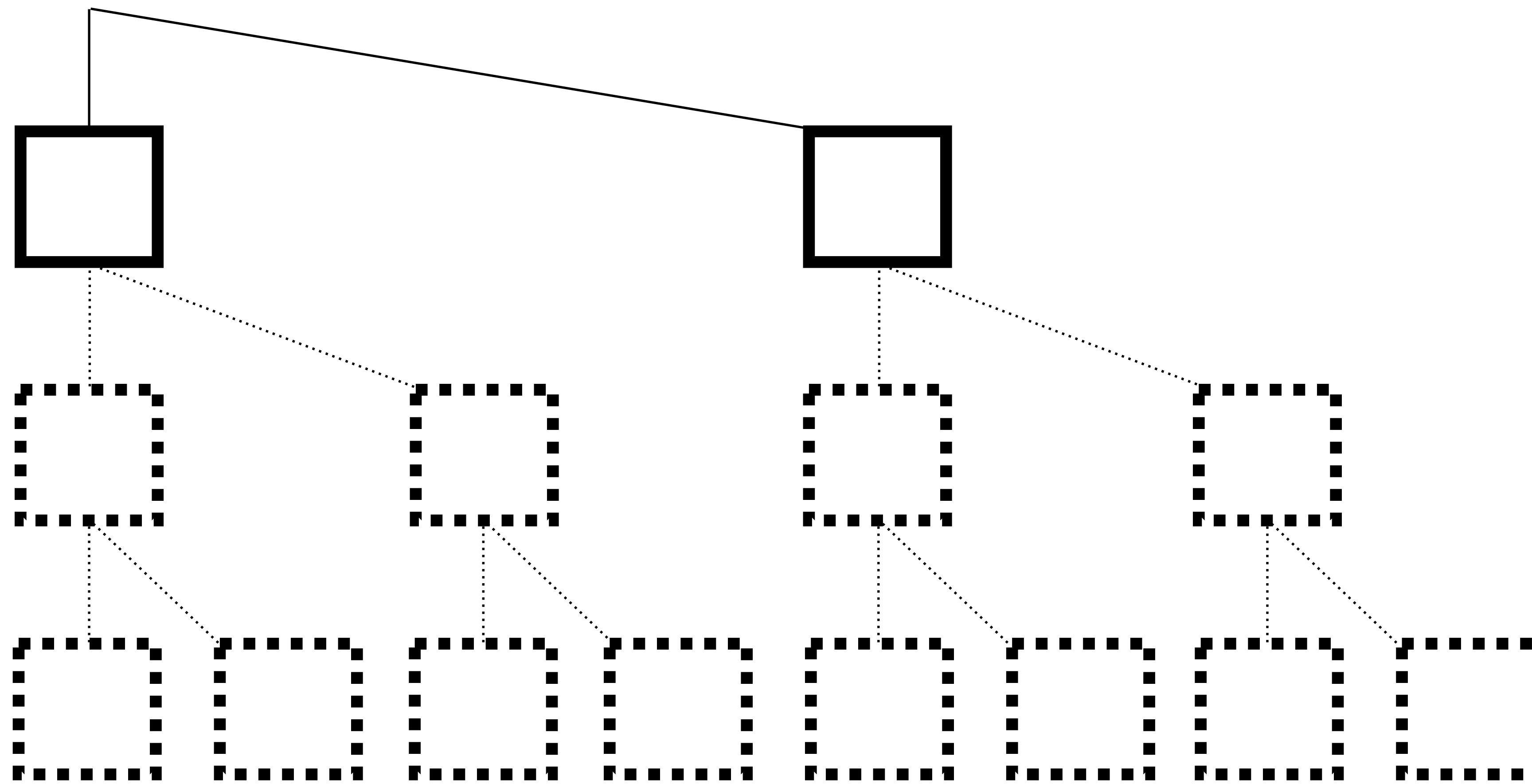
# 8 UTXOs

## Throw away

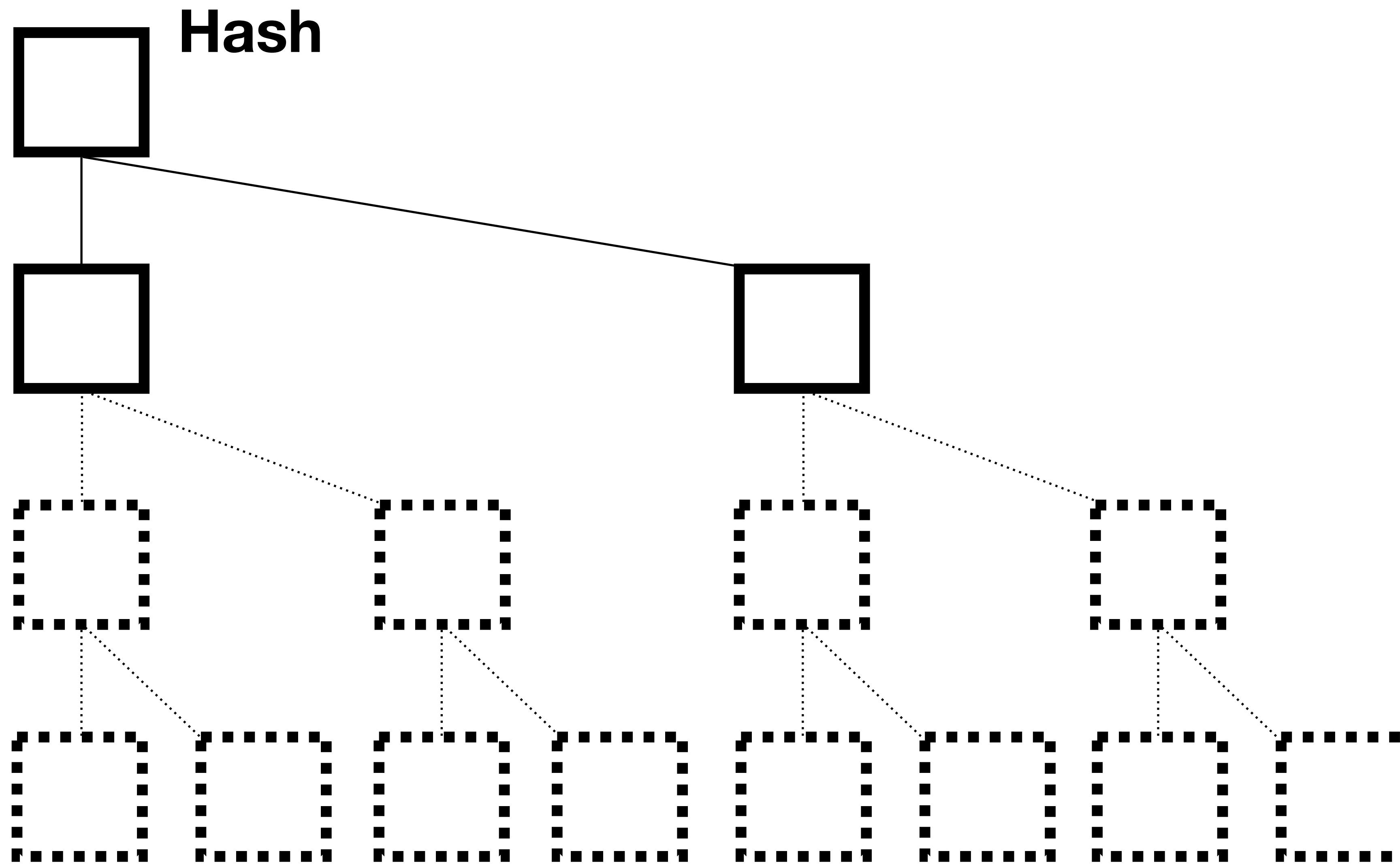


# 8 UTXOs

## Concatenate

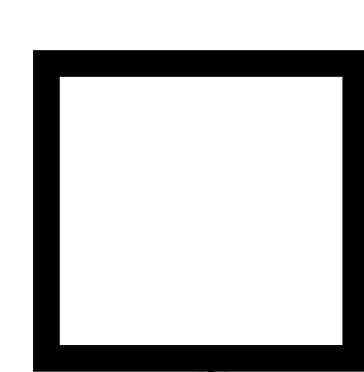


# 8 UTXOs

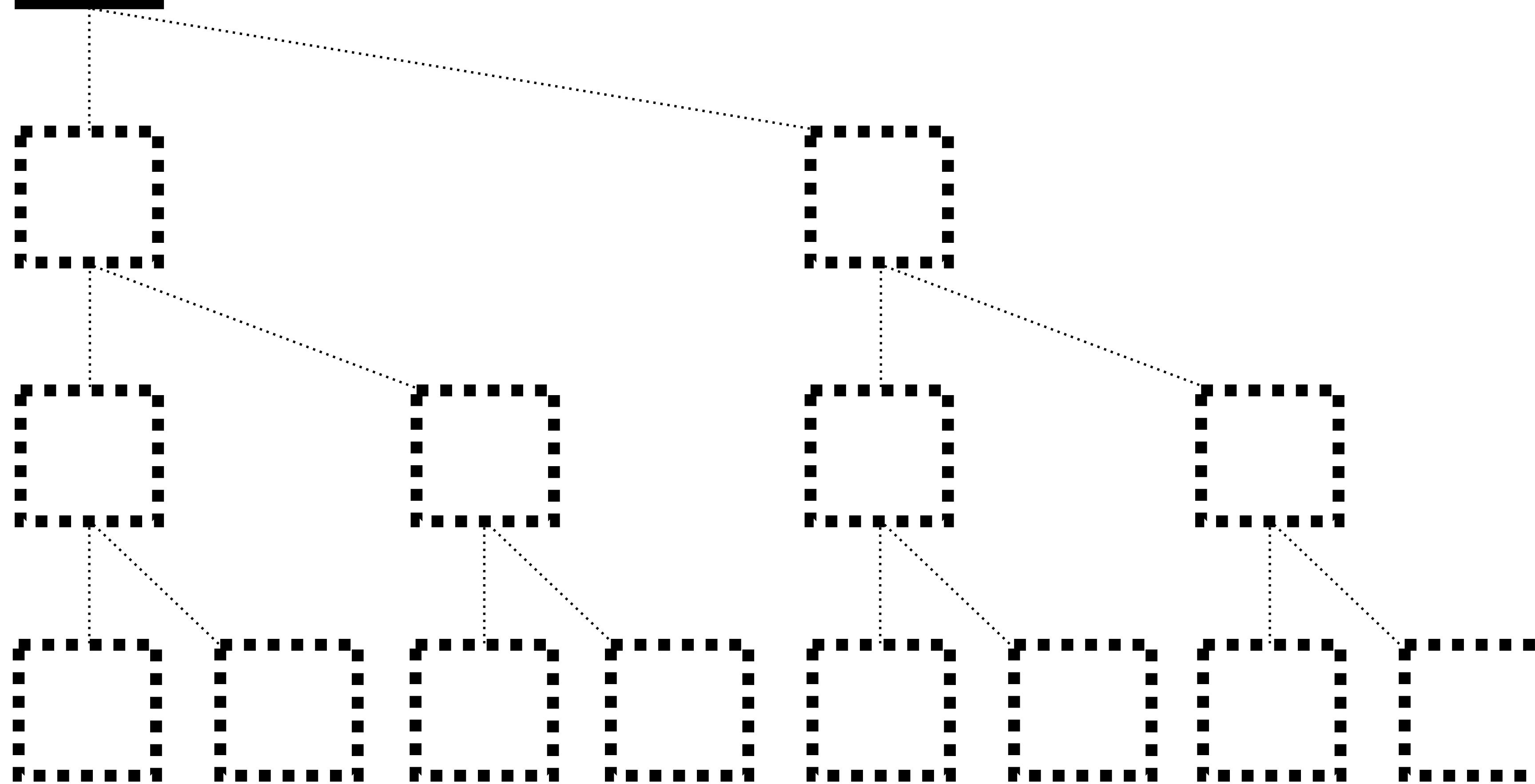


<https://gist.github.com/kcalvin/alvin/a790d524832e1b7f96a70c642315fffc>

# 8 UTXOs

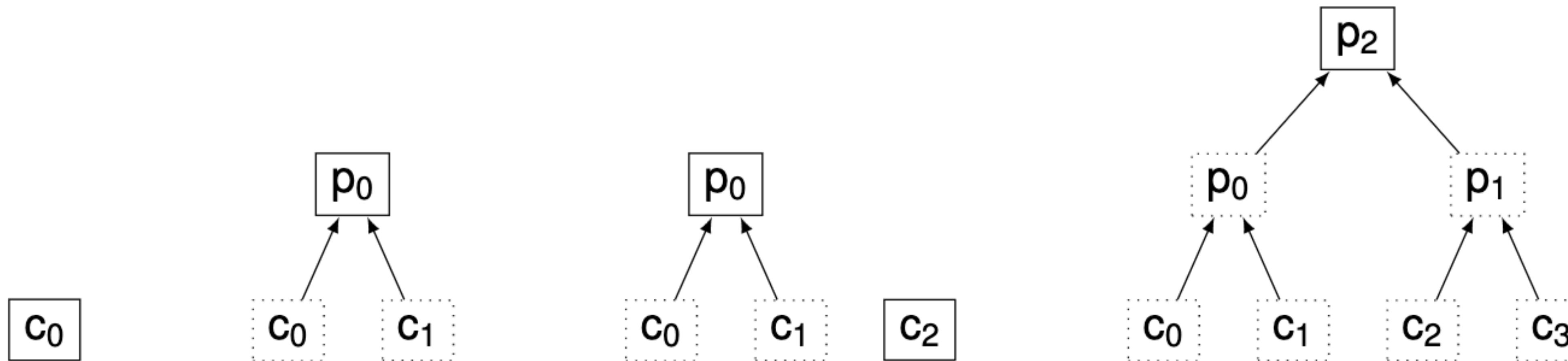


**Throw away**



## 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.

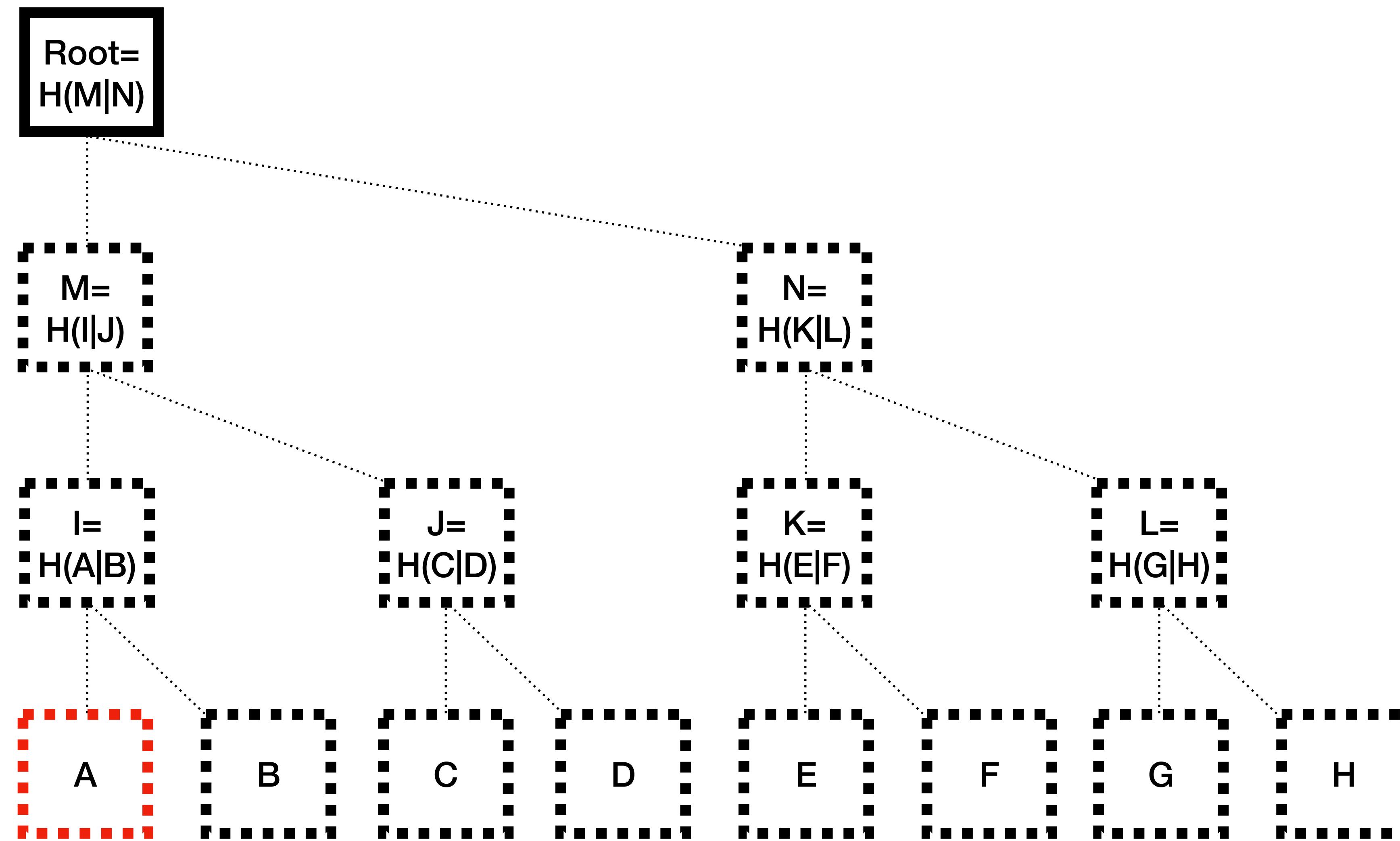
# **Role of levelDB**

**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

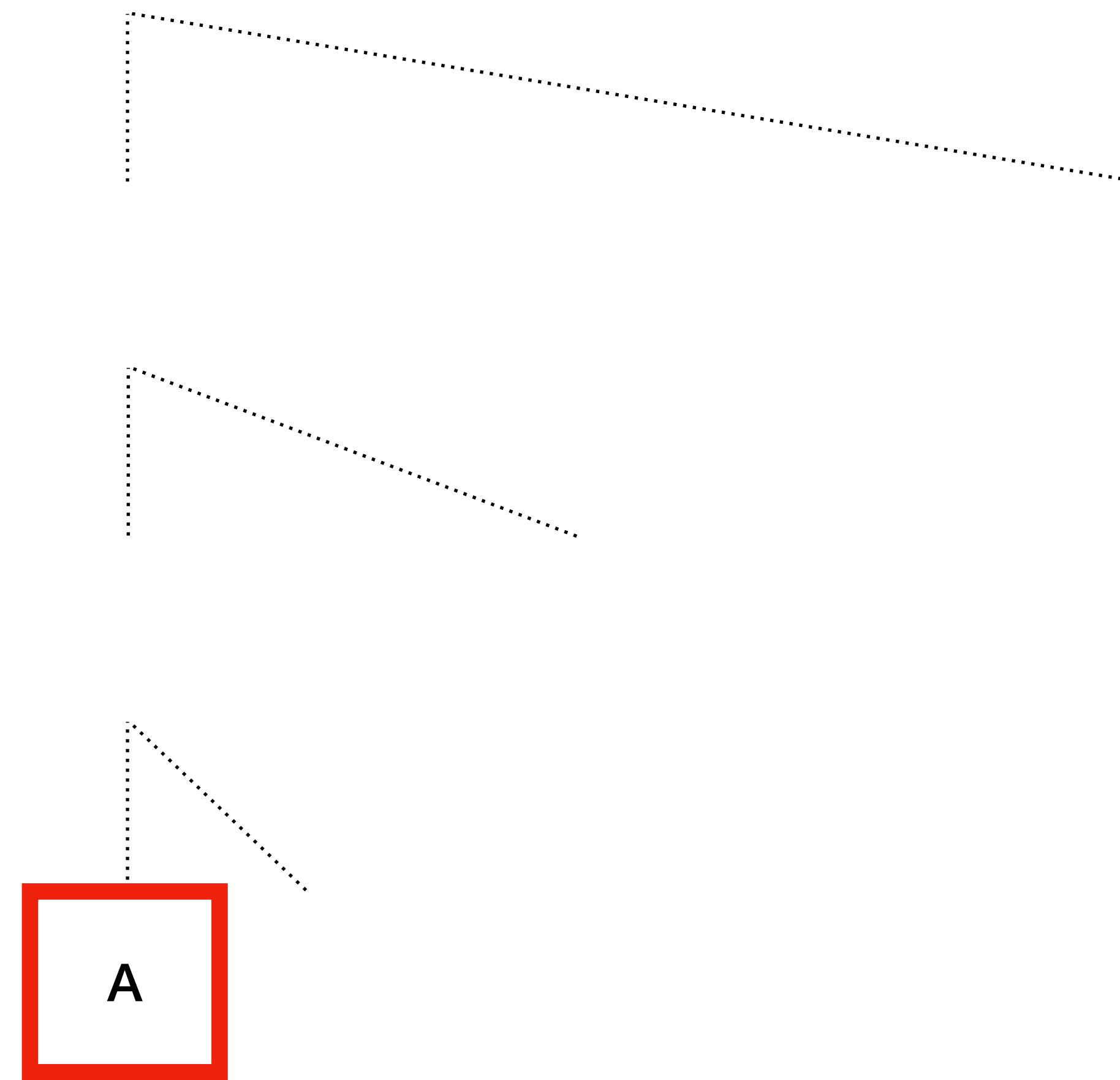


# The only requirement is the roots

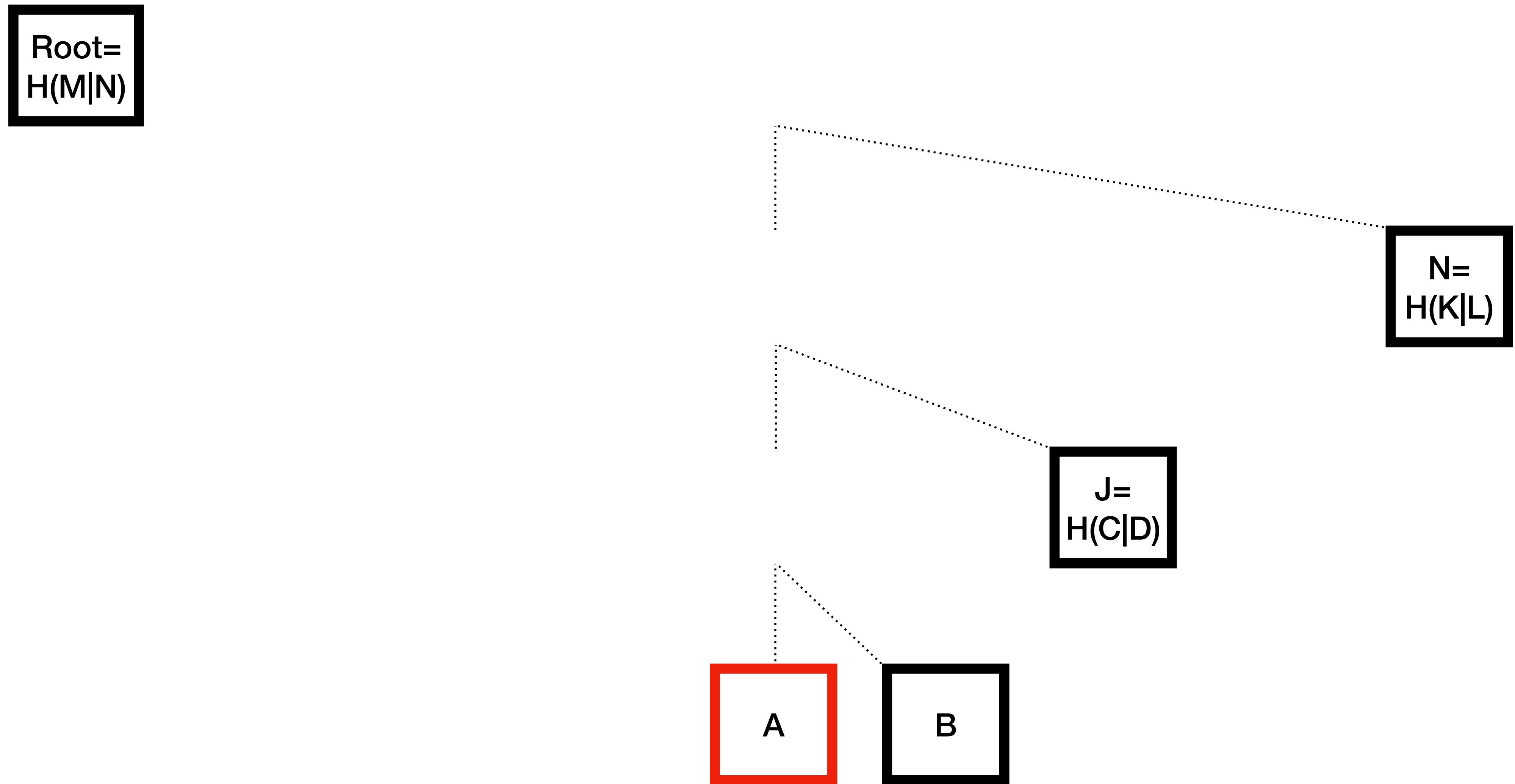
Root=  
 $H(M|N)$

# Calculate A from the Bitcoin Block

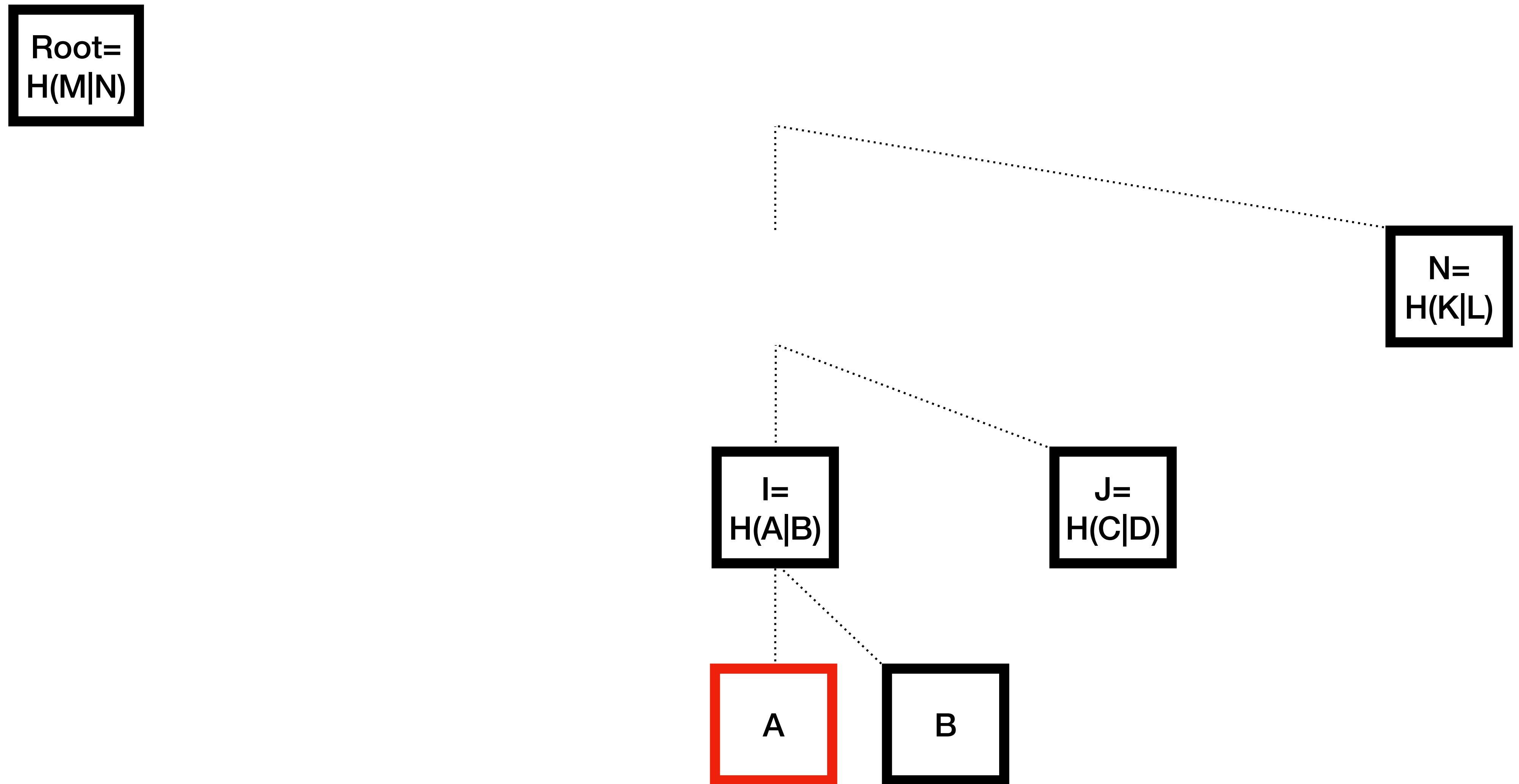
Root=  
 $H(M|N)$



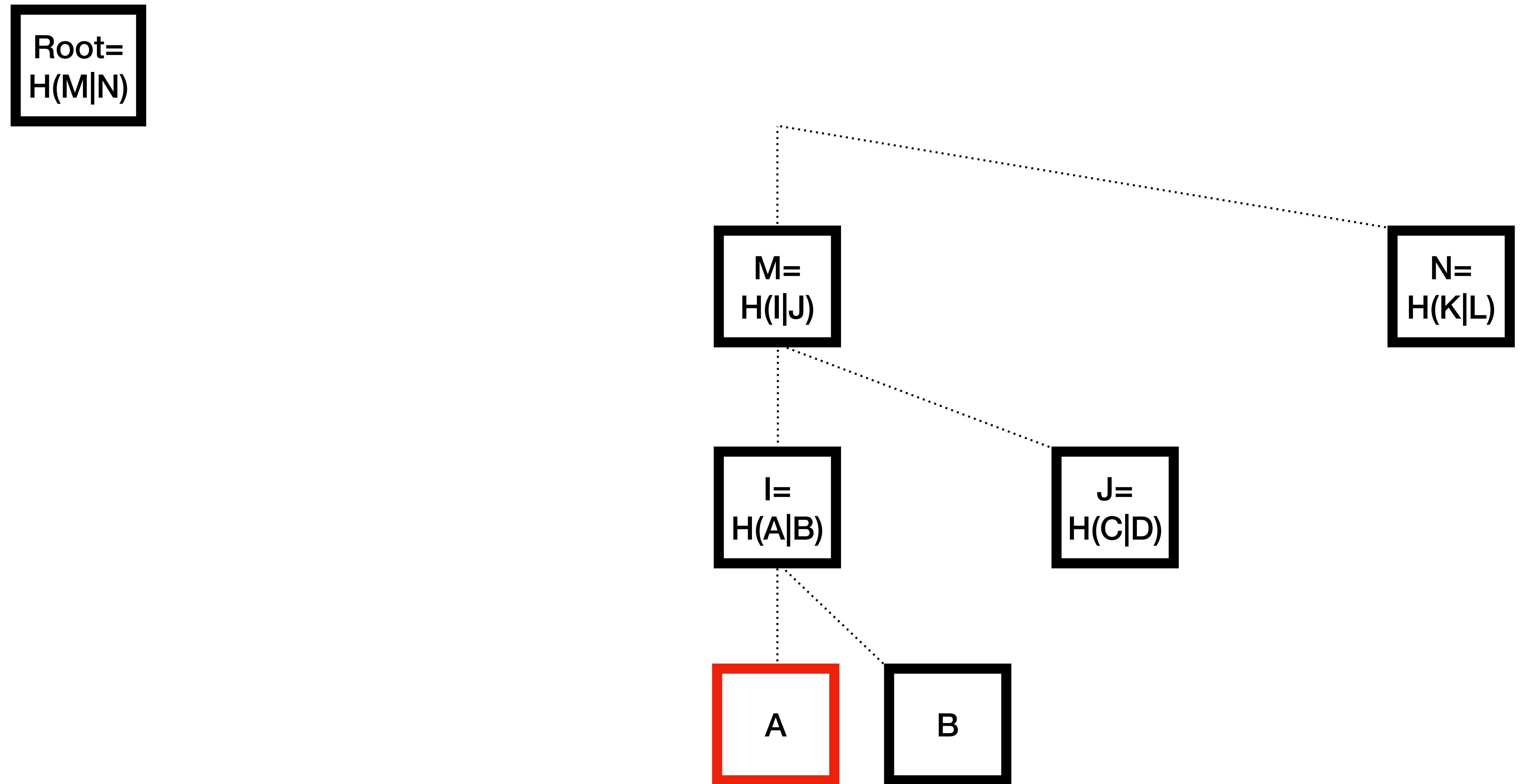
# Receive the proof: B, J, N



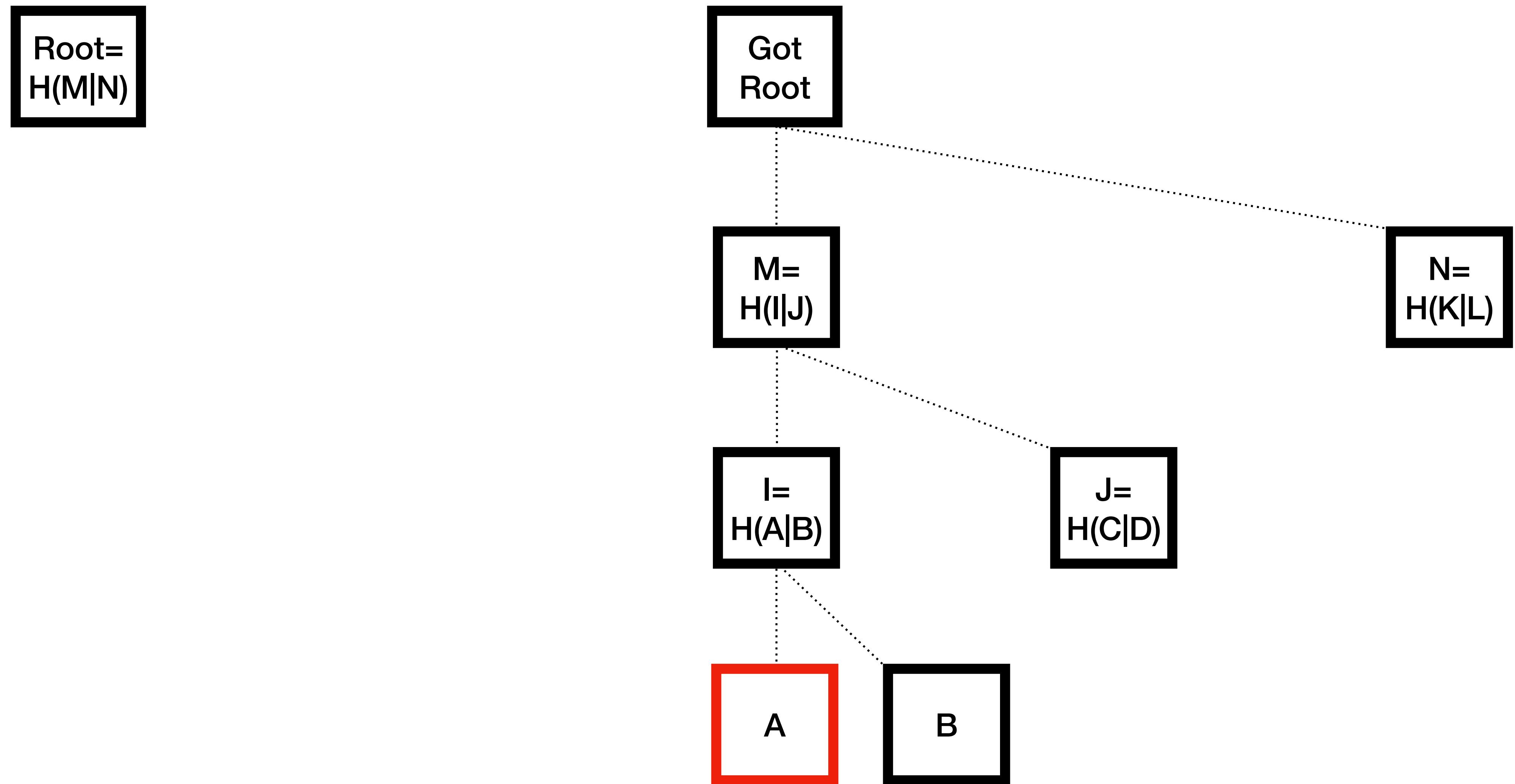
# Calculate the hash for I



# Calculate the hash for M



# Calculate Root



# Compare roots

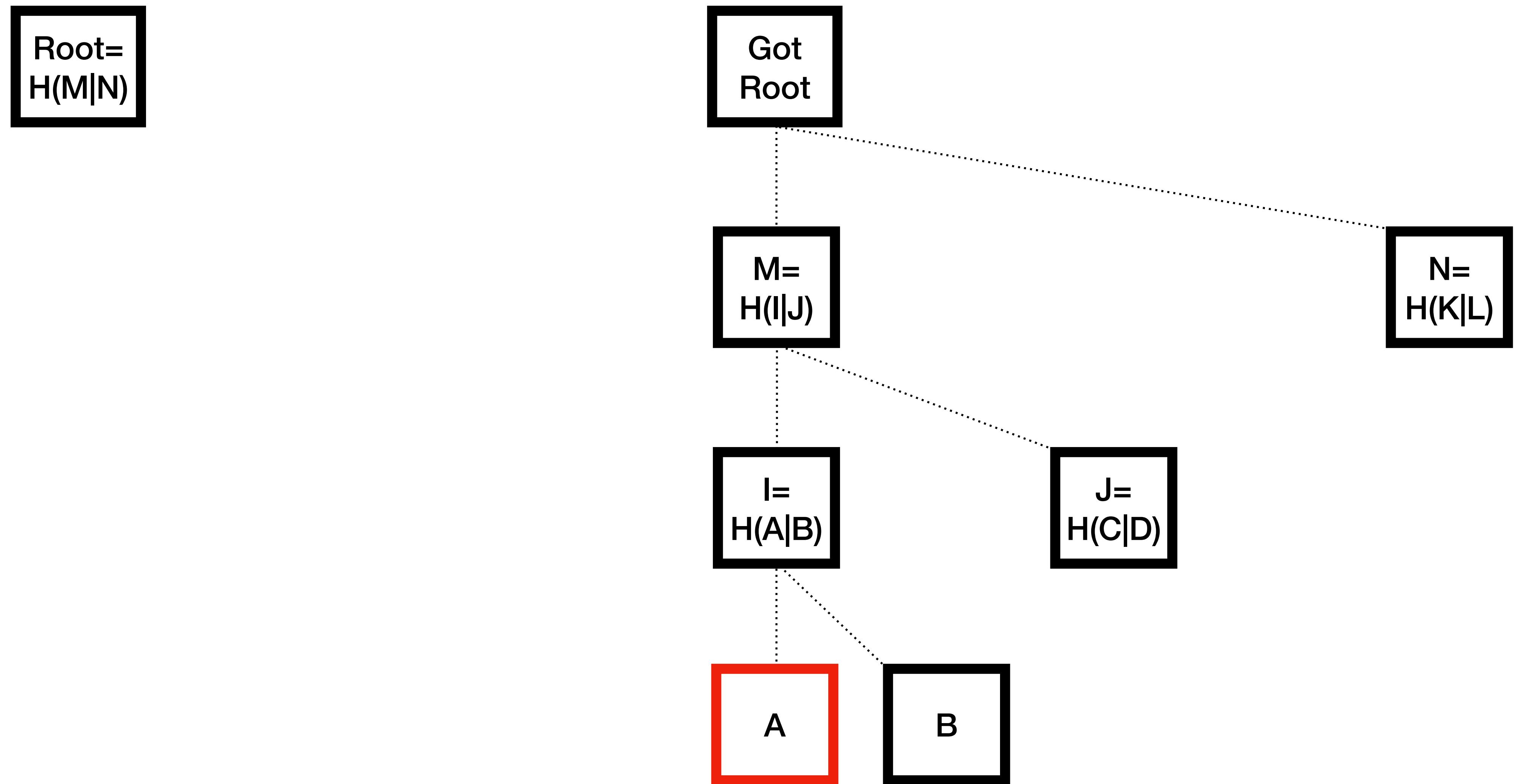
Continue if the roots are equal. Ban peer if not equal

Root= $H(M|N)$

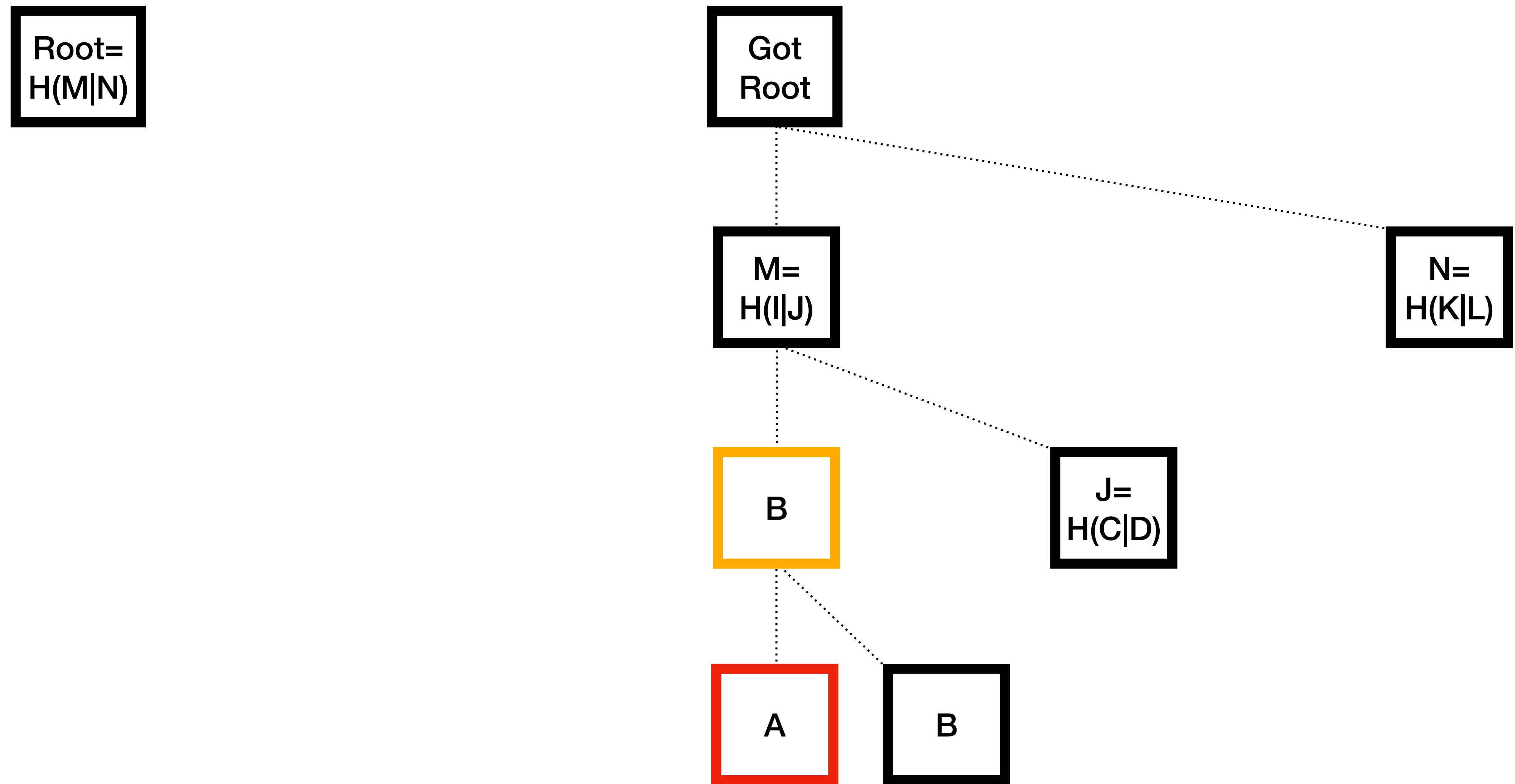
==

Got Root

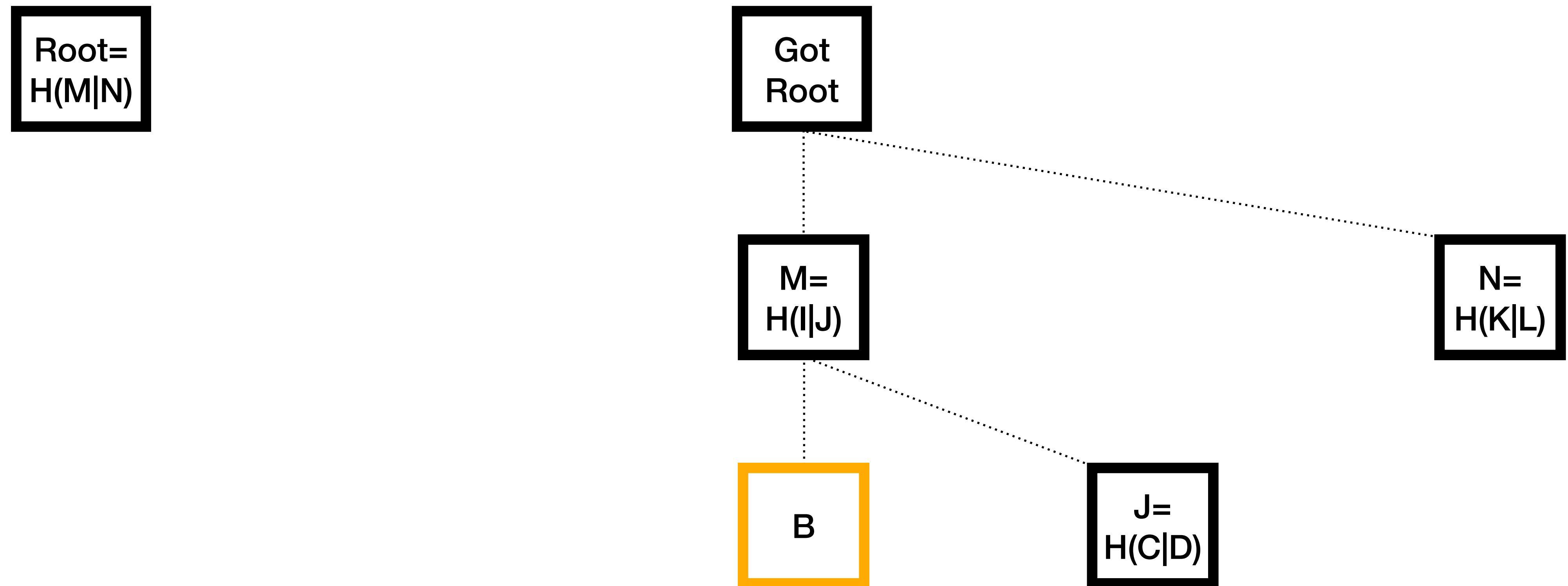
# Calculate root after deletion



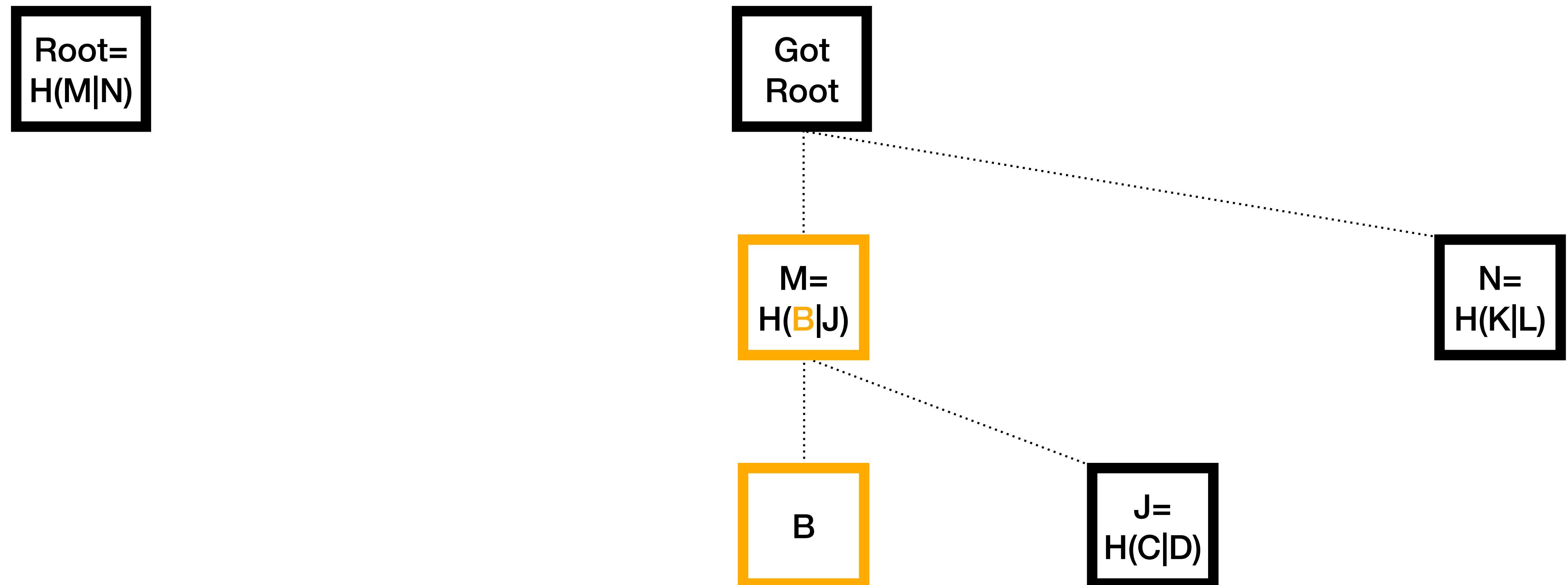
# Move up B to I



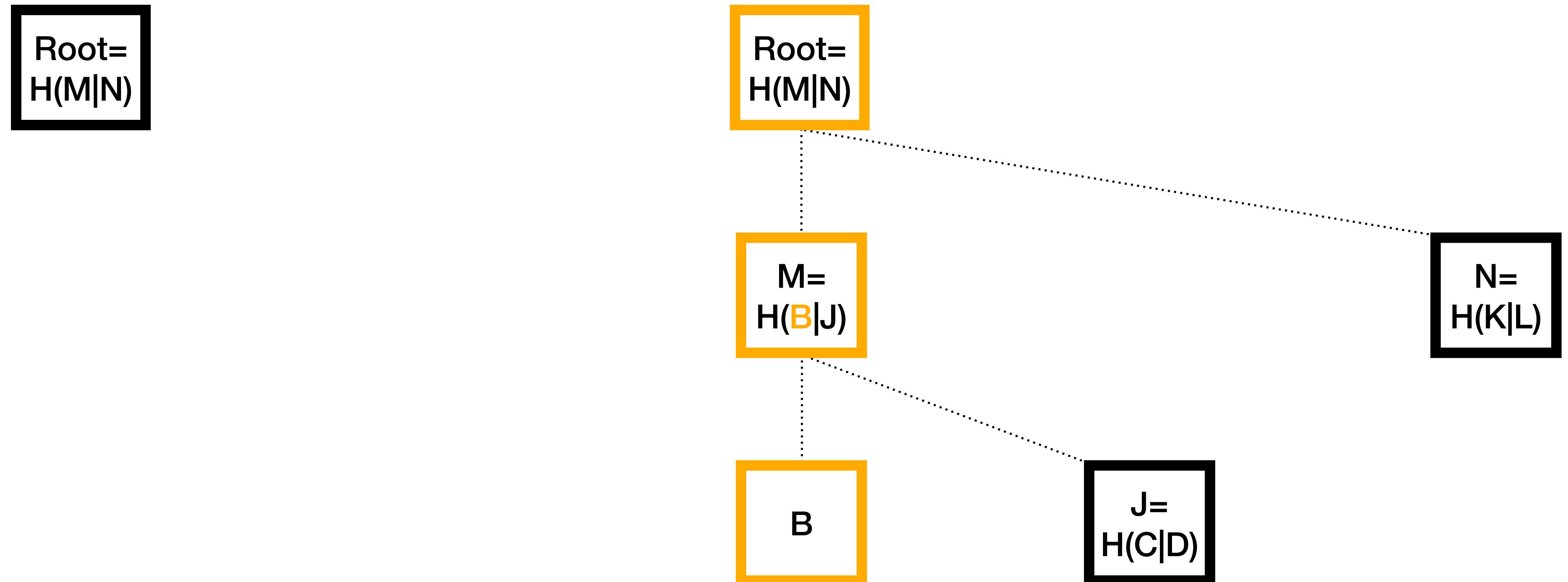
# Remove old nodes for A&B



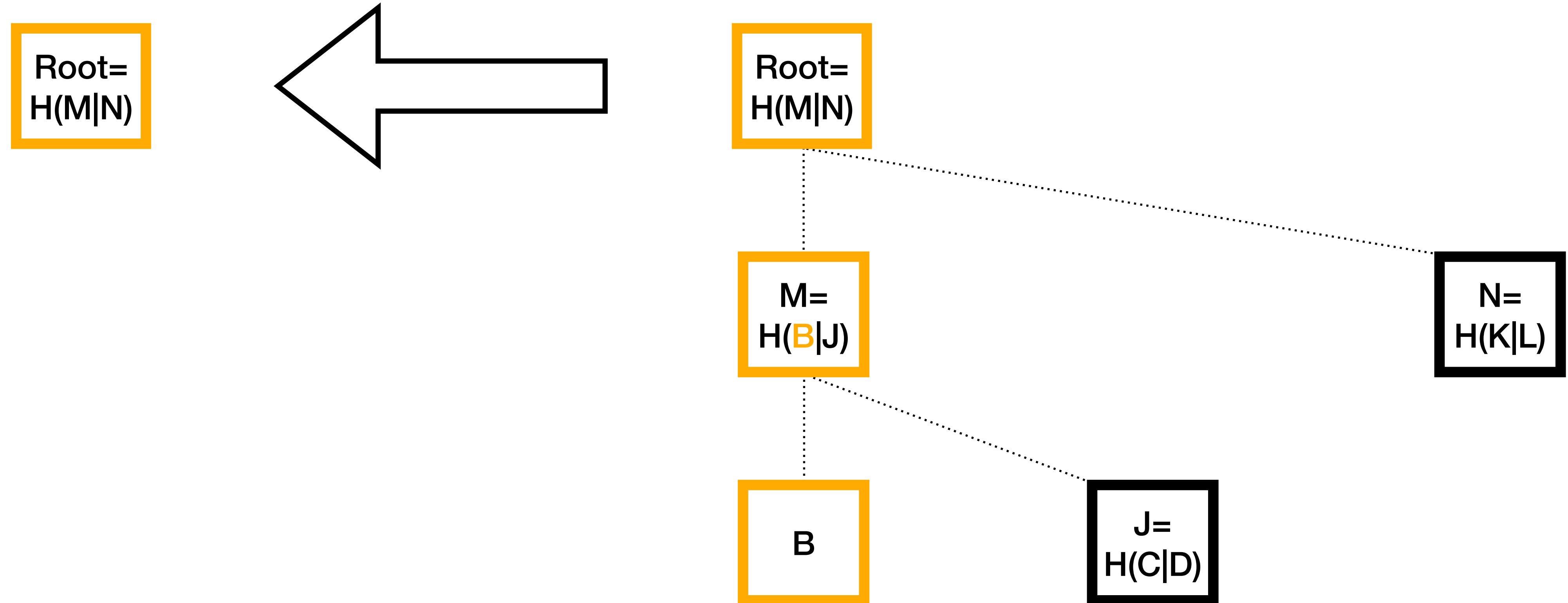
# Calculate $H(B|J)$



# Calculate new root



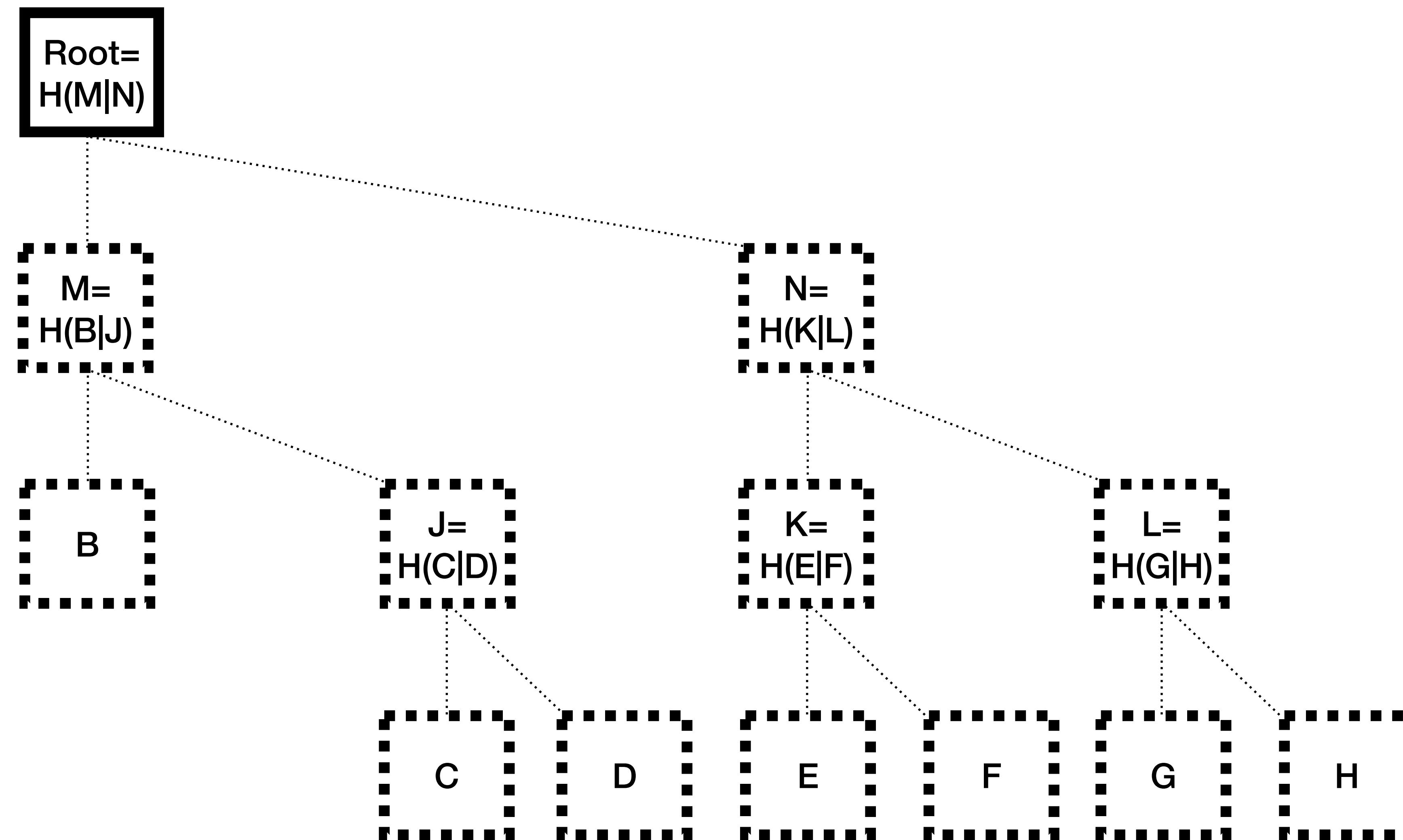
# Copy over the new root to be saved



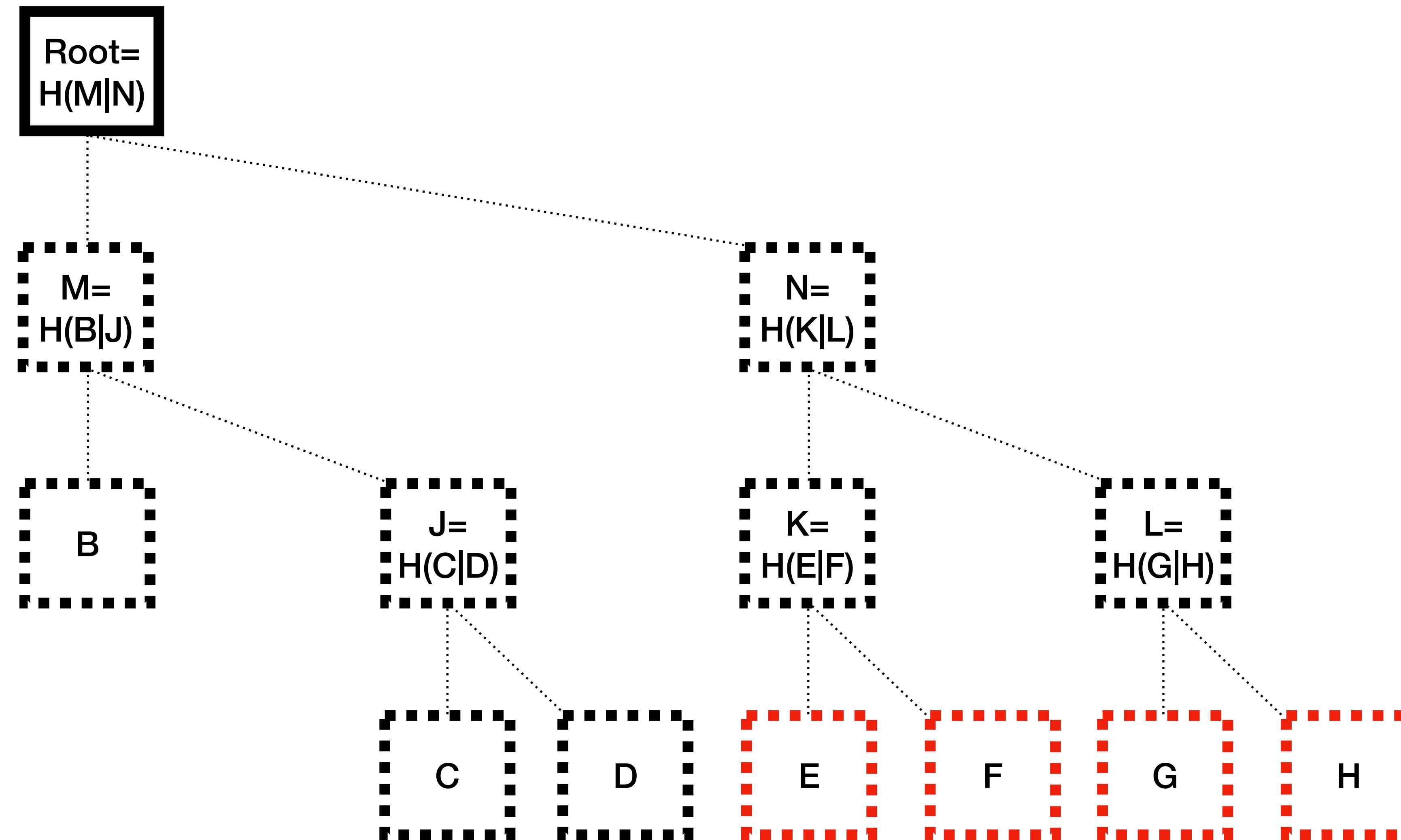
# Done

Root=  
 $H(M|N)$

# After deleting A



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

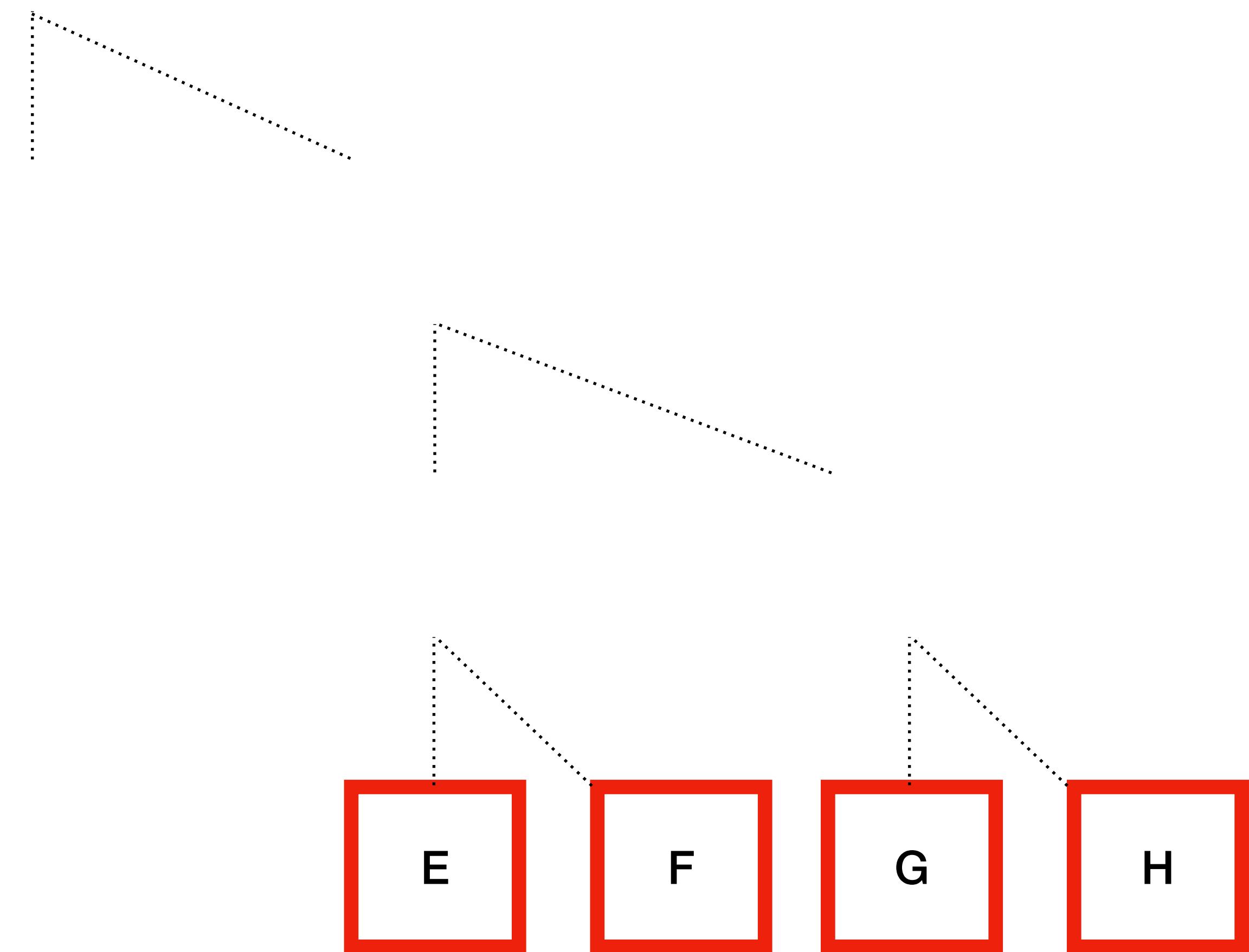


# Start off with only the root

Root=  
 $H(M|N)$

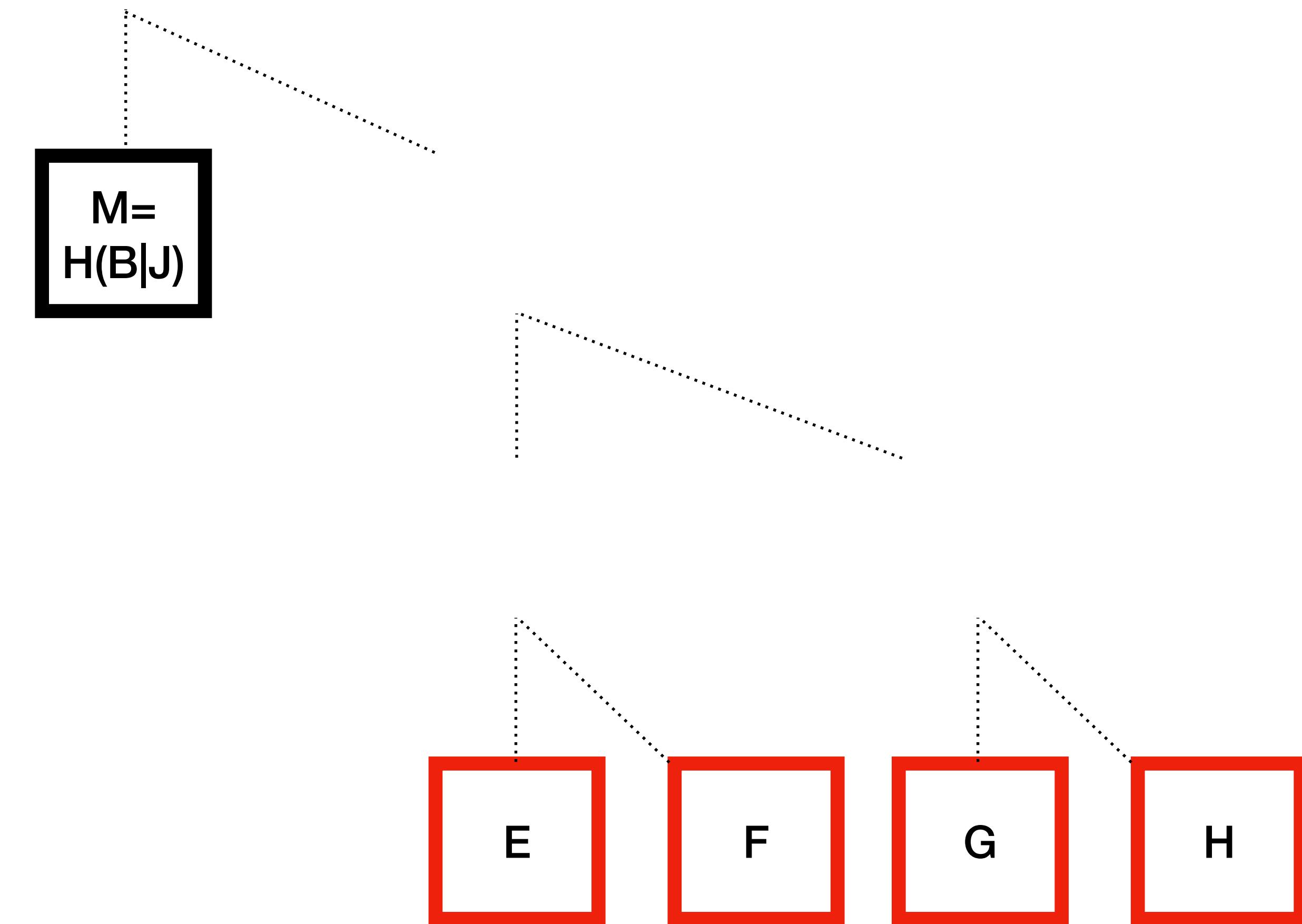
# Calculate E, F, G, H from the Bitcoin Block

Root=  
 $H(M|N)$

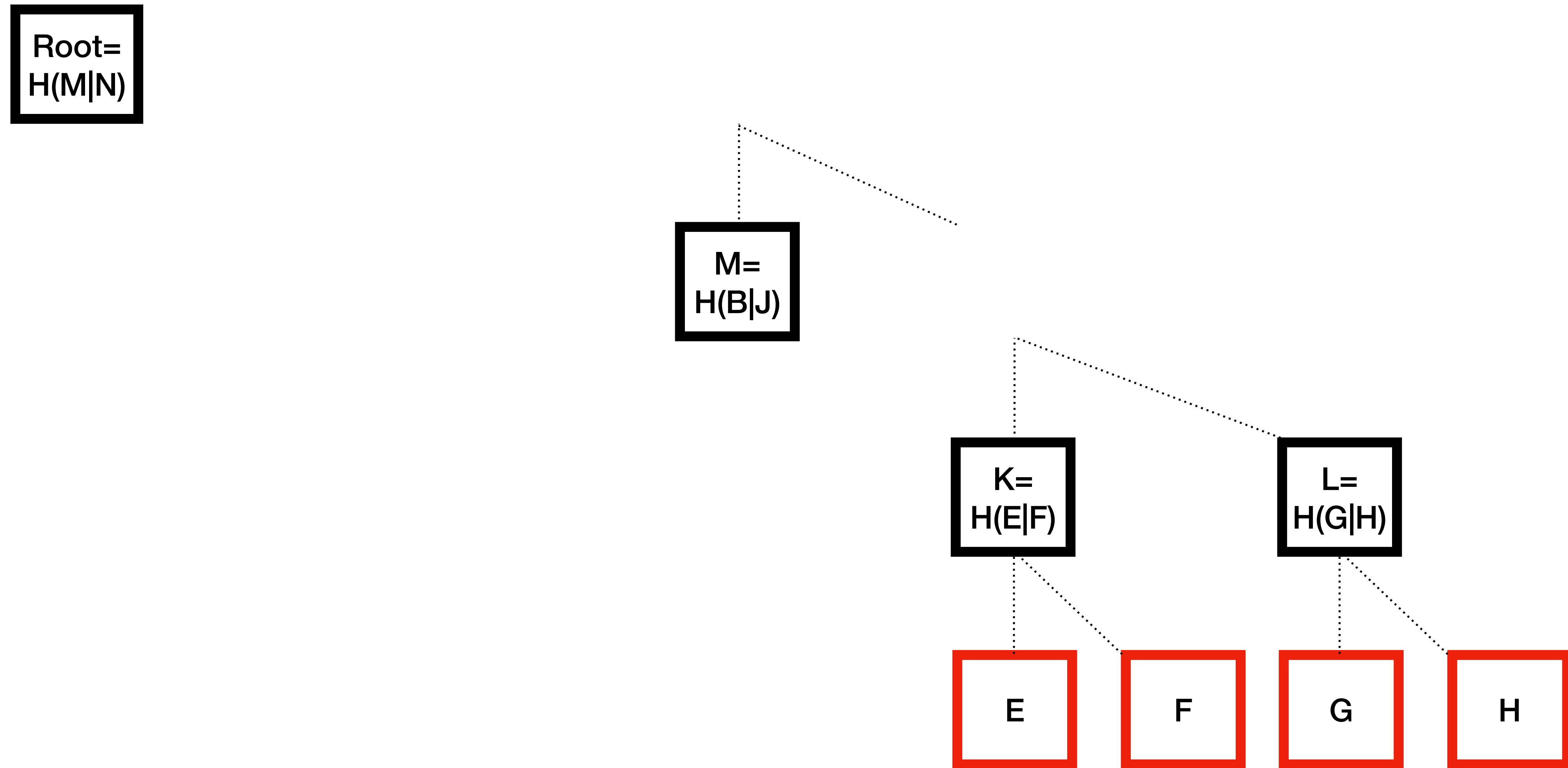


# Receive the proof: M

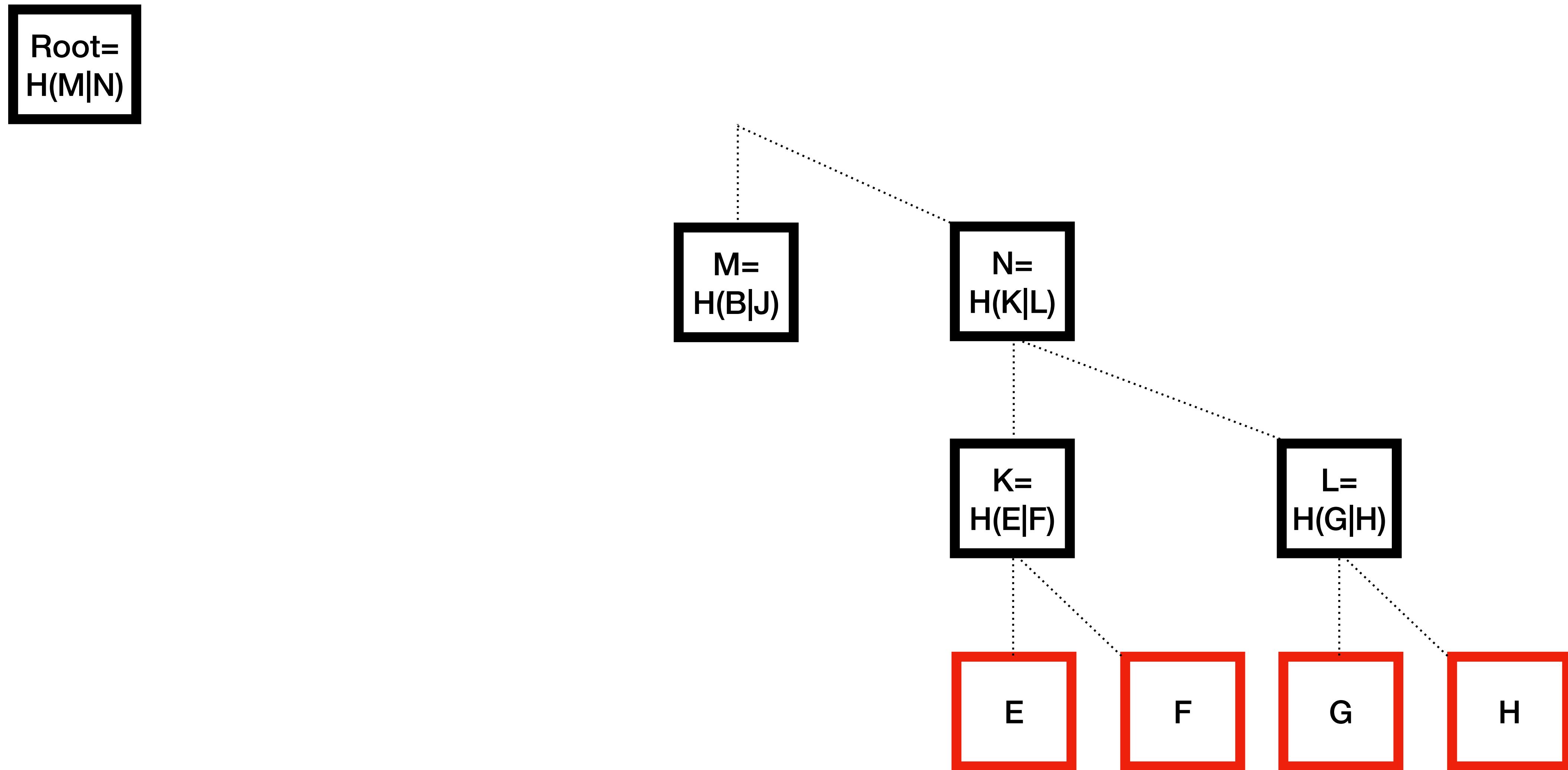
Root=  
 $H(M|N)$



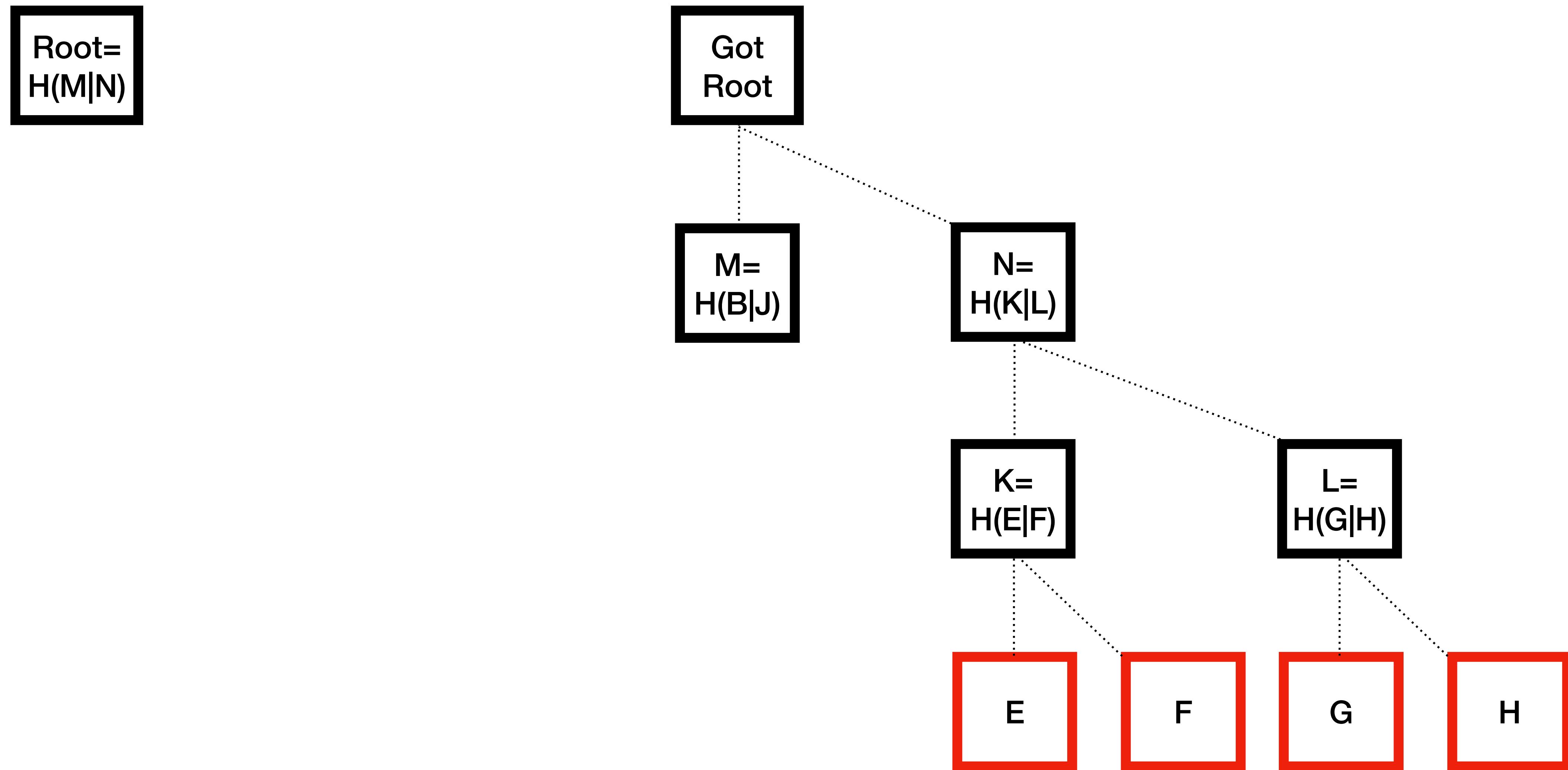
# Calculate K and L



# Calculate N



# Calculate the Root



# Compare roots

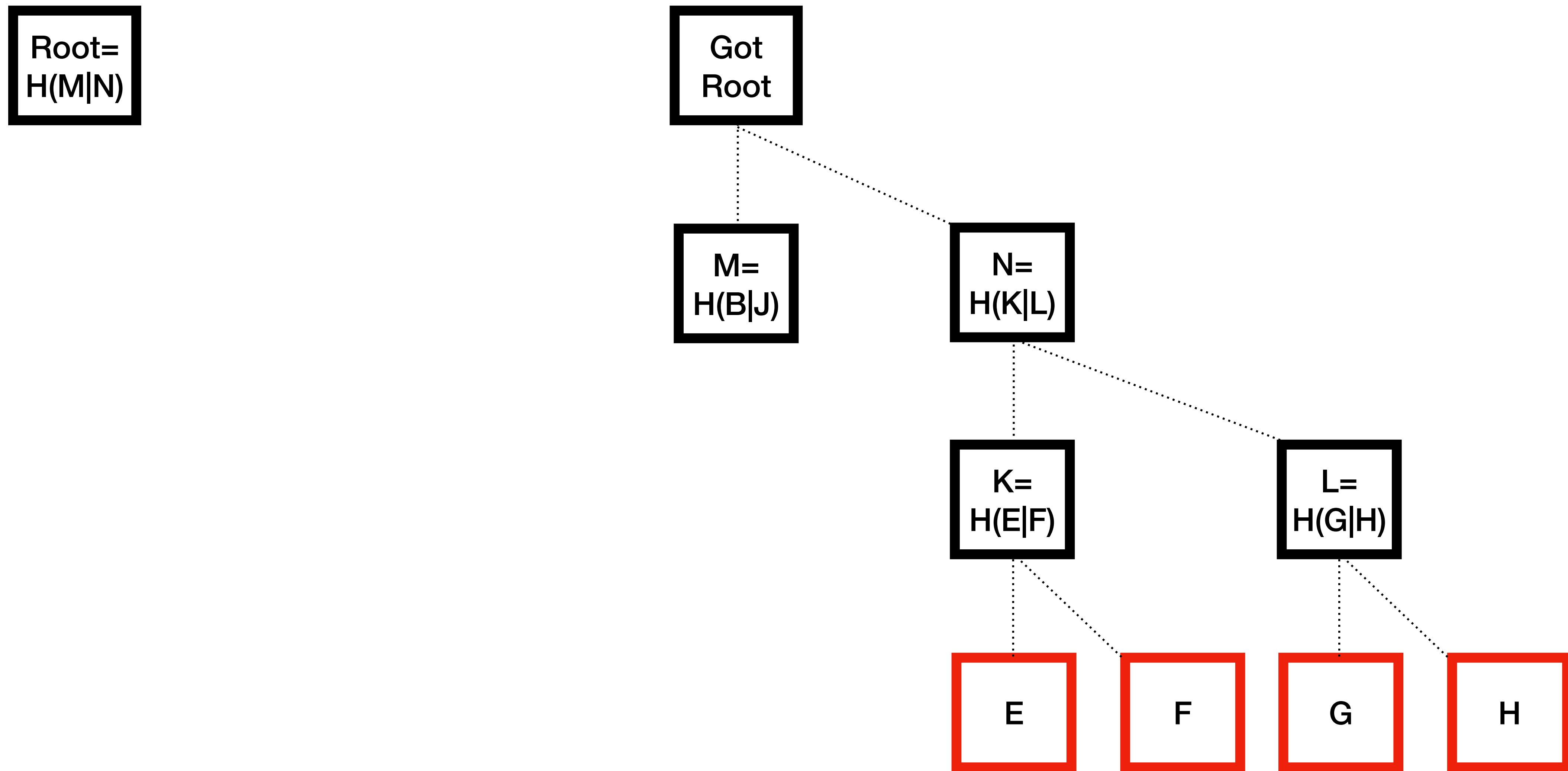
Continue if the roots are equal. Ban peer if not equal

Root= $H(M|N)$

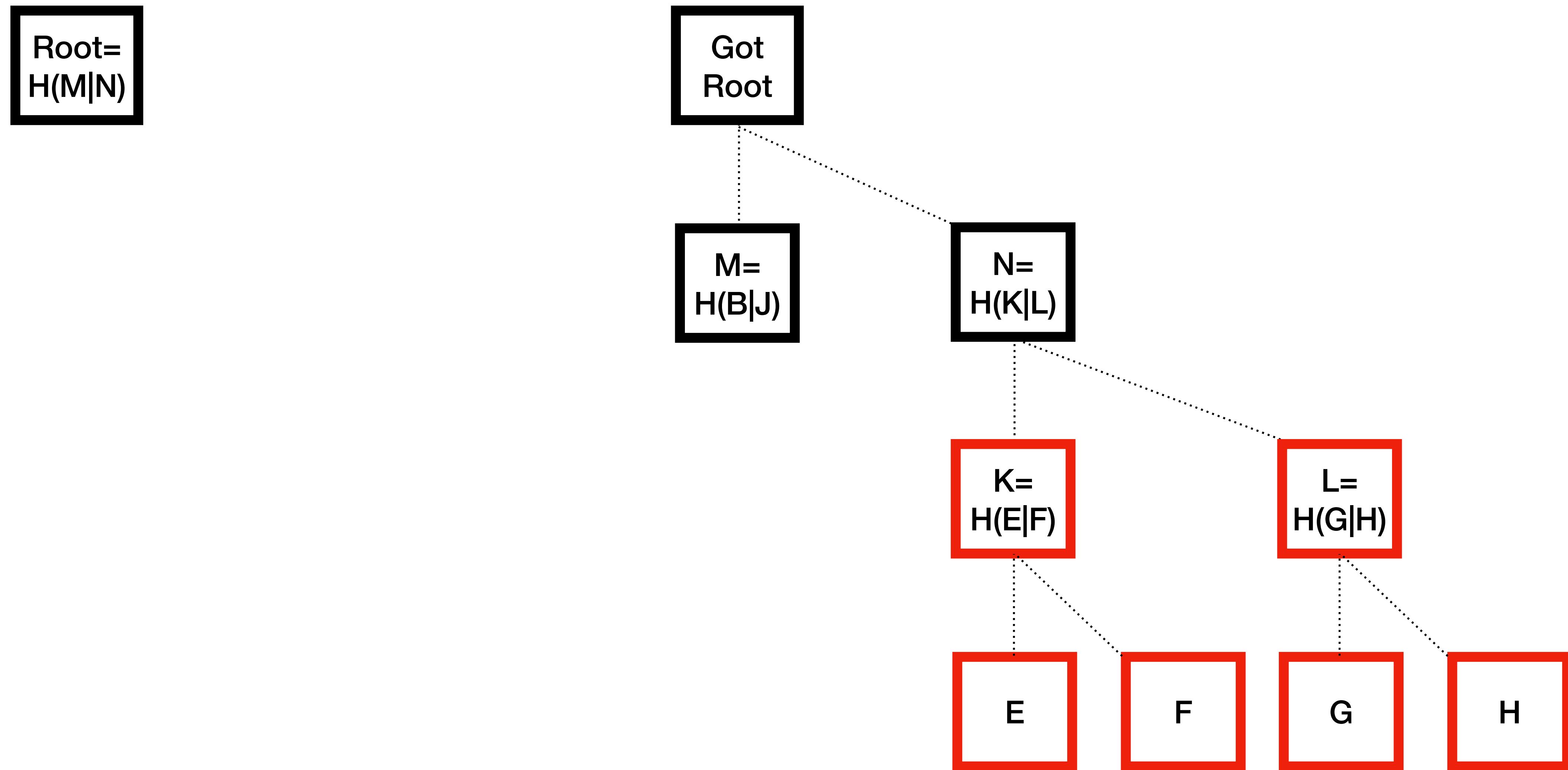
==

Got Root

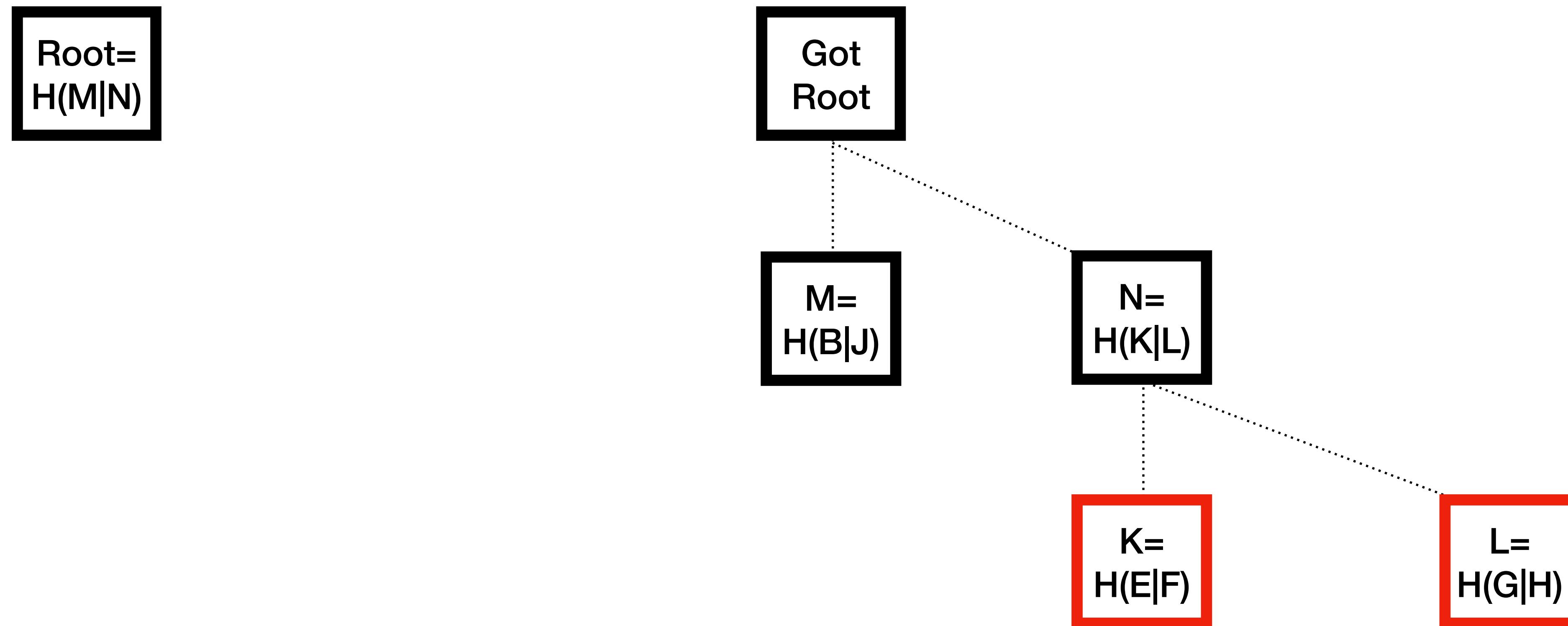
# Calculate new root



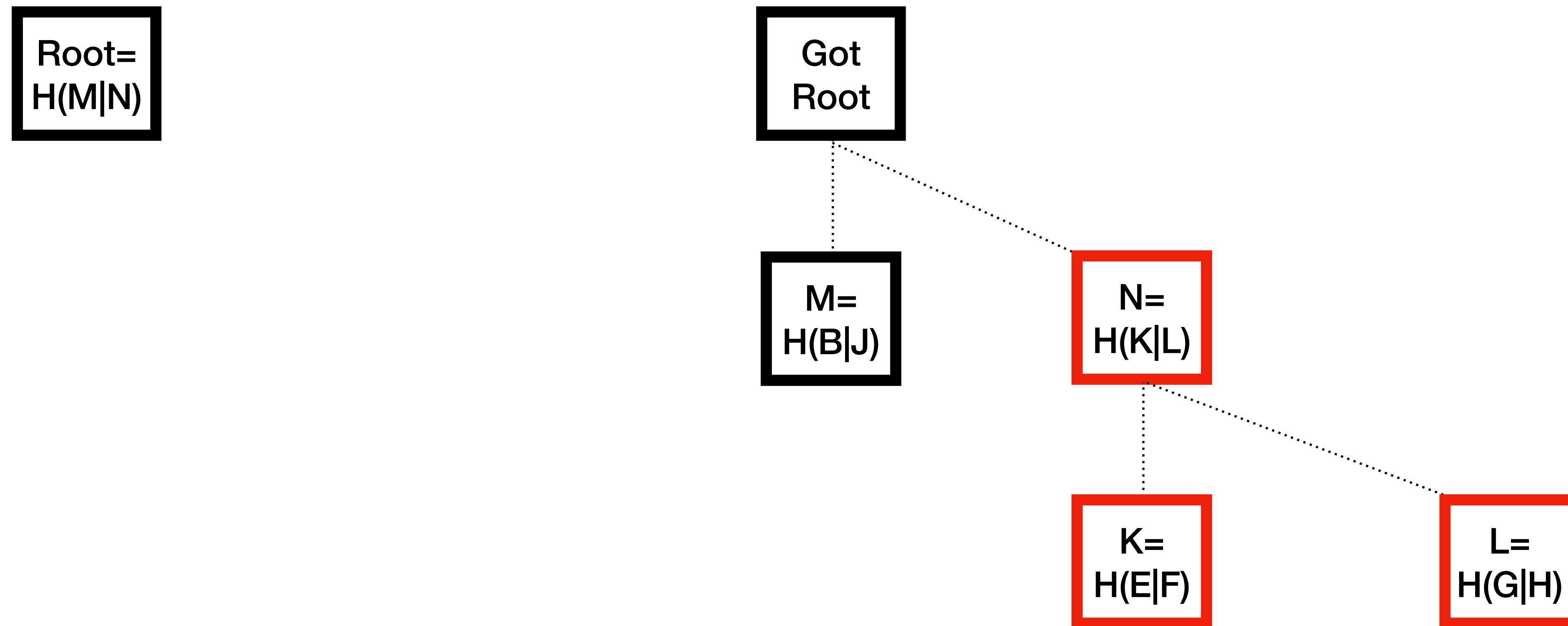
# Mark K & L as node to delete



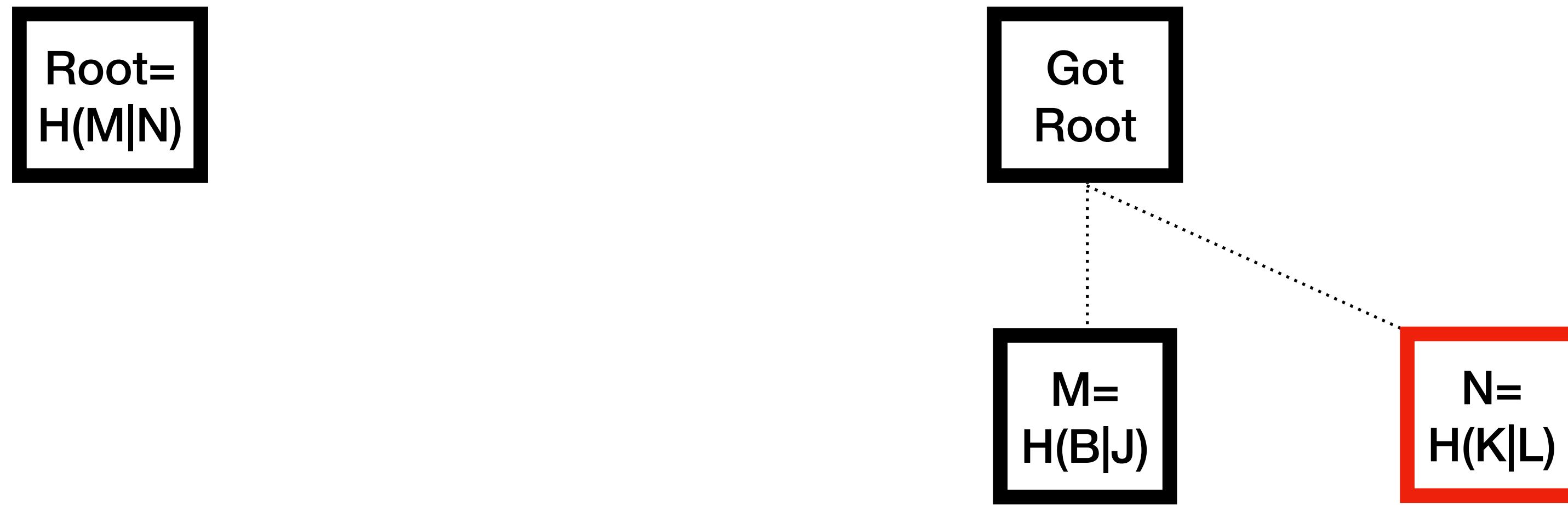
# Remove old nodes for E, F, G, H



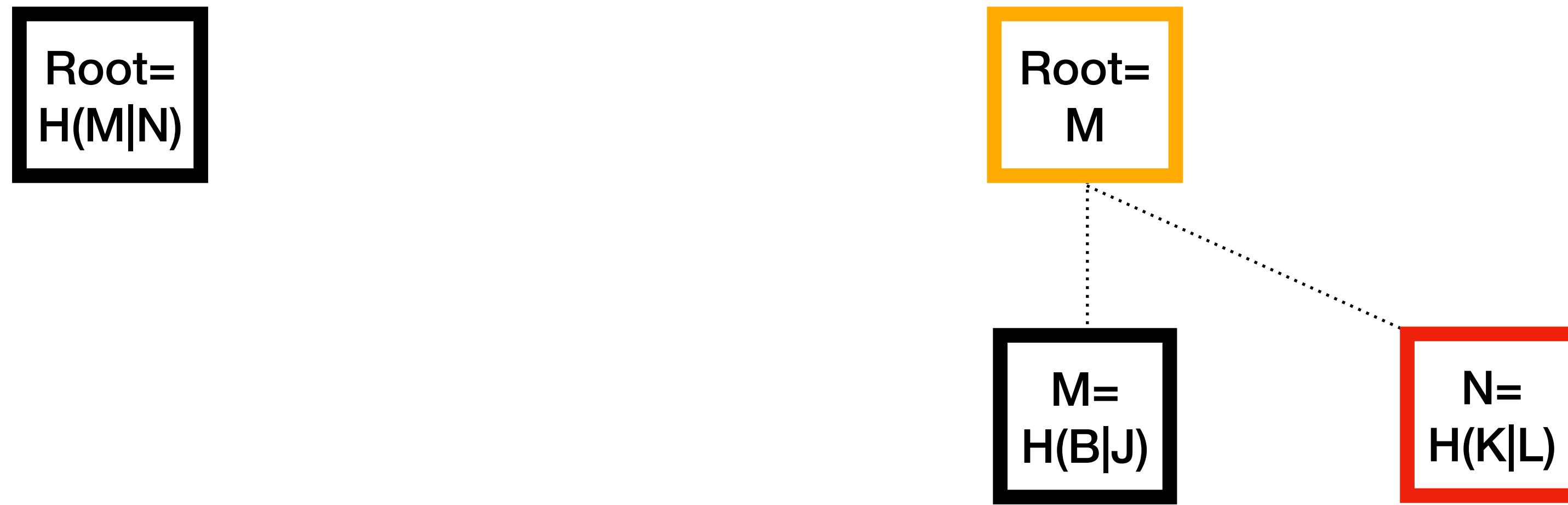
# Mark N as node to delete



# Remove old nodes for K & L



# Move up M

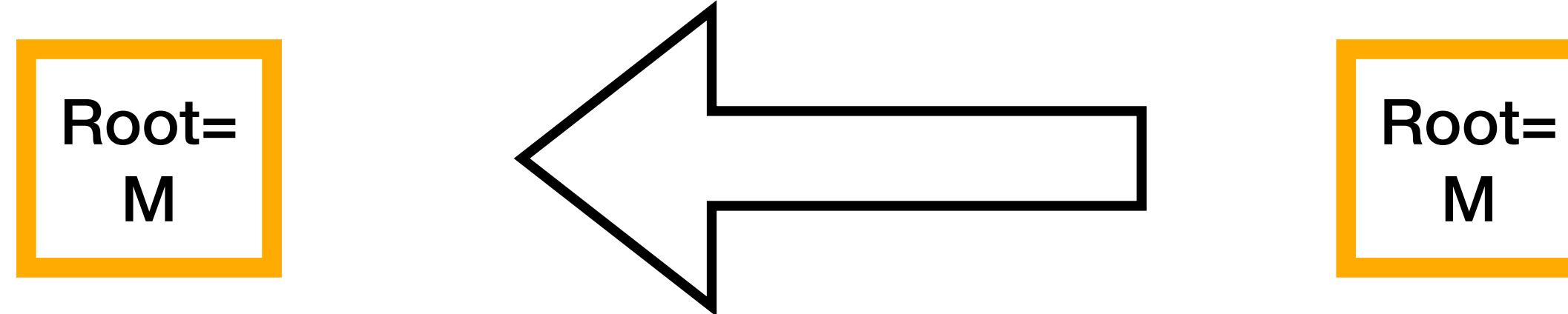


# Remove old nodes for M & N

Root=  
 $H(M|N)$

Root=  
M

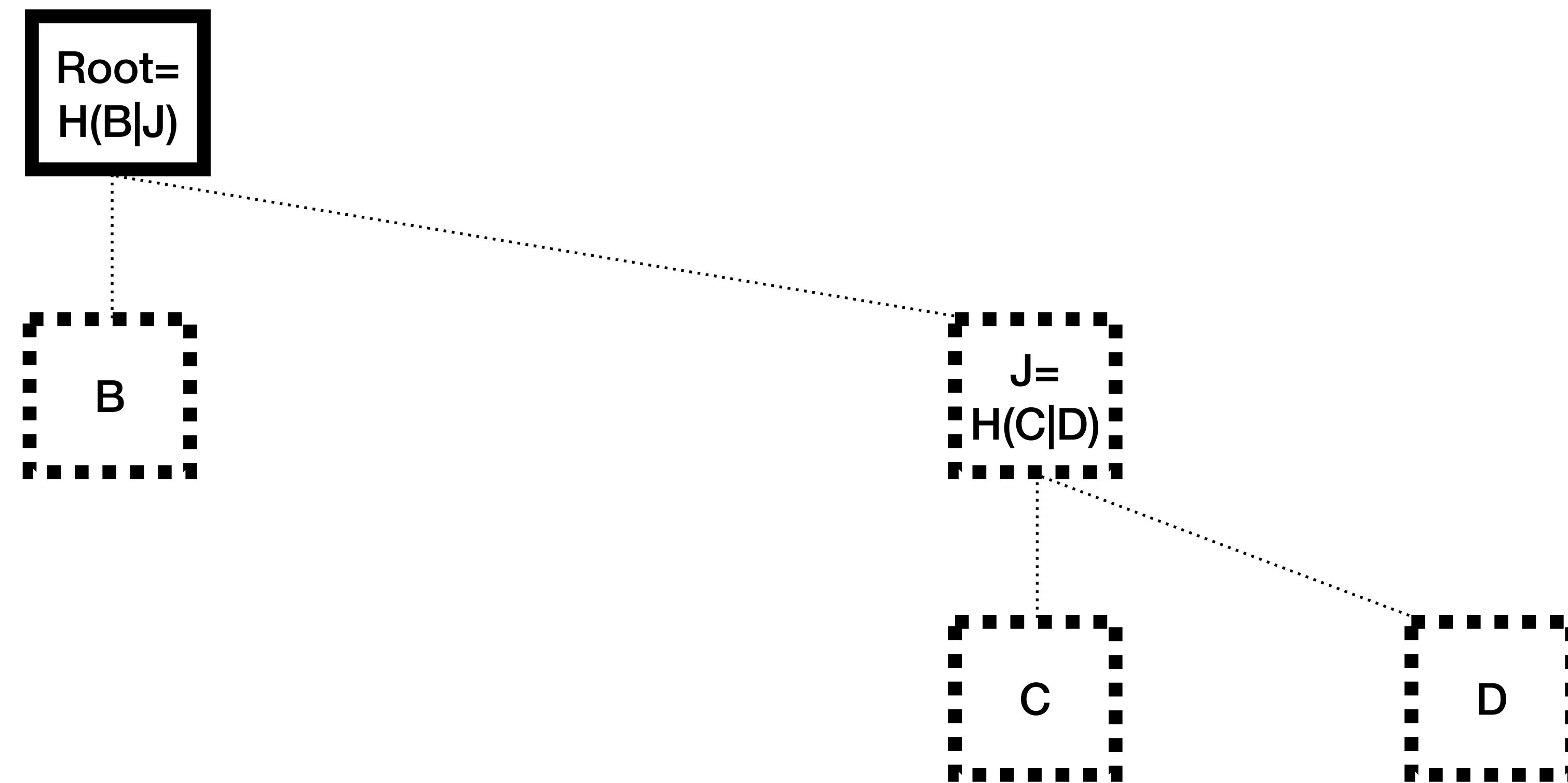
# Copy over the new root to be saved



# 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)
- Thanks theStack! (Sebastian Falbesoner)

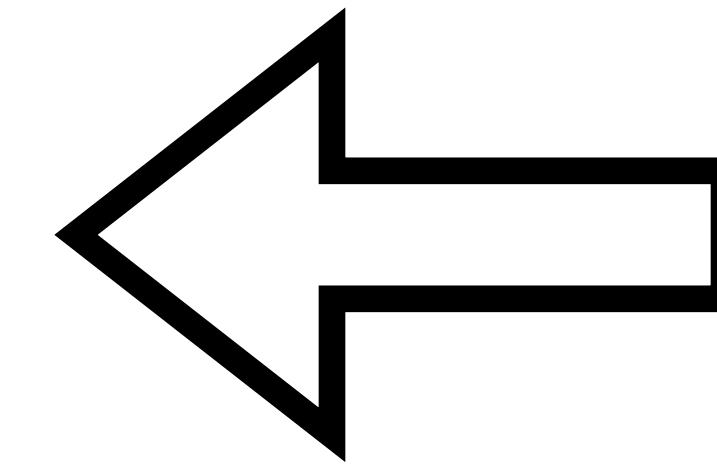
# Role of levelDB

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

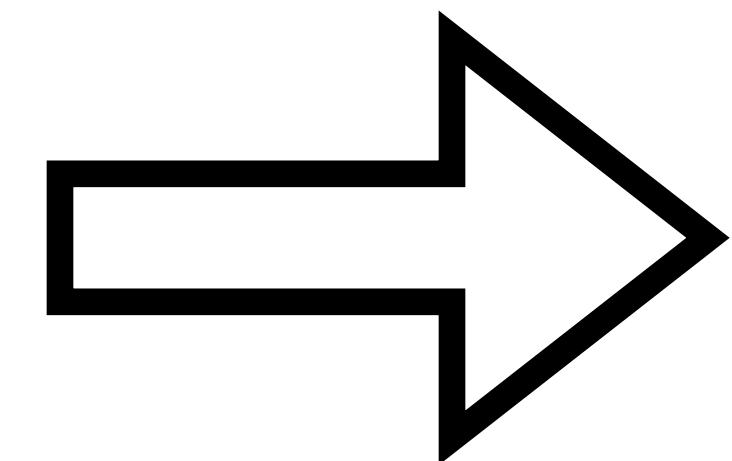
## Fetching a UTXO



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

# Utreeexo Proof Data

Not finalized!

Merkle  
Proof

UTXO  
Data

# Block serialization for Utreexo nodes

Not finalized!

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

# **Batched Utreexo Proof Data**

**Not finalized!**

**Batched Merkle Proof**

**UTXO data Count**

**UTXO Datas**

# UTXO Data serialization

## 2 serialization methods

- For calculating the hash to be committed into the accumulator

# UTXO Data serialization

## For hash calculation

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

# UTXO Data serialization

## 2 serialization methods

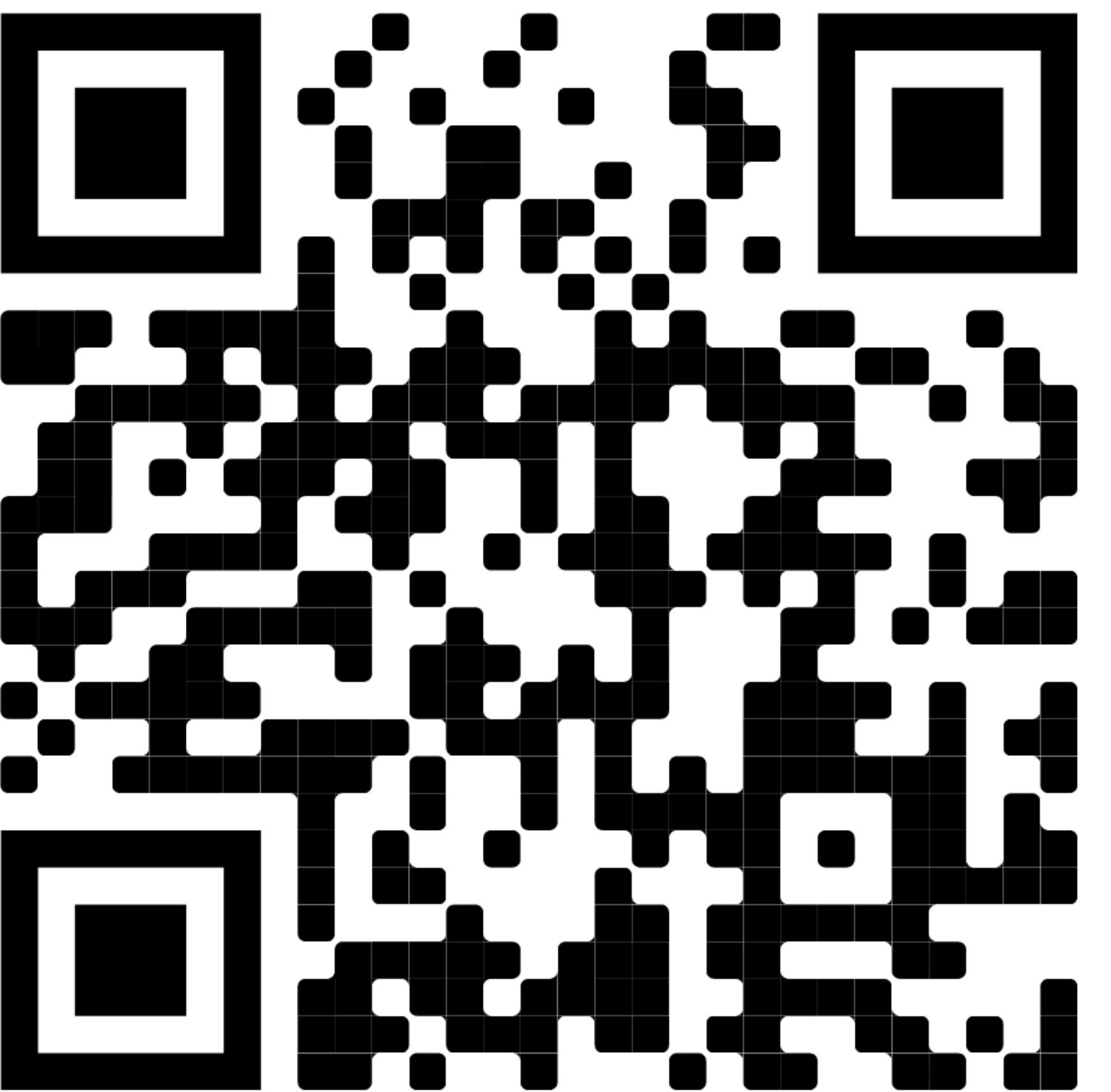
- For calculating the hash to be committed into the accumulator
- For sending data to peers/storage on disk

# UTXO Data serialization

For disk storage/p2p transfer

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

Twitter:  
@kcalvinalvinn



Slides:  
[github.com/kcalvinalvin/  
slides-for-taiwan-bitdevs](https://github.com/kcalvinalvin/slides-for-taiwan-bitdevs)

