utreexo

full nodes in kilobytes

Tadge Dryja

2019-09-09
edge / dev++ / scaling
Tel Aviv University

Current blockchain size: big.

history: 253GB
(only goes up)

current state: 3.5+GB
(mostly goes up)

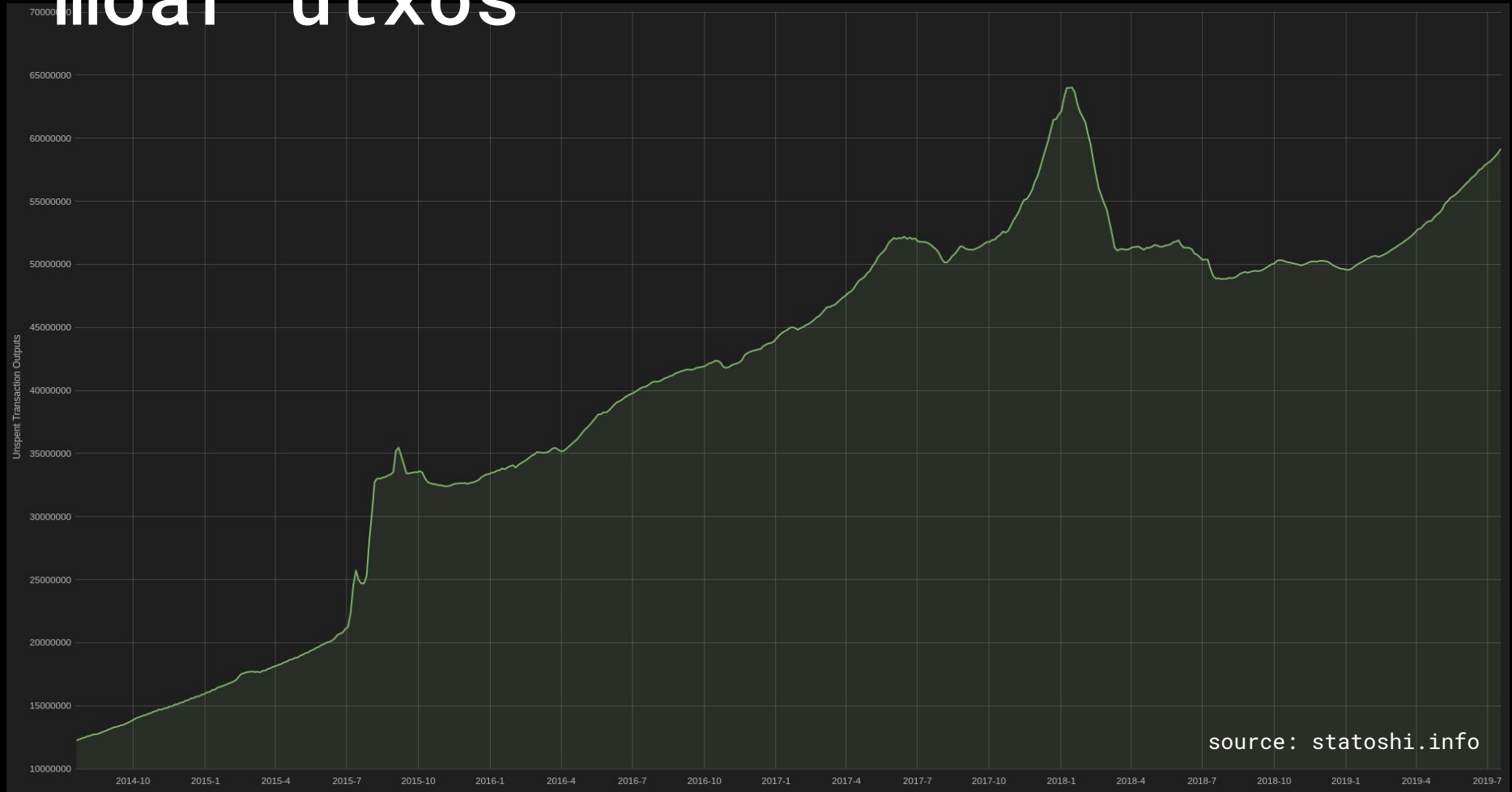
```
~/.bitcoin$ du -h
253G  ./blocks
3.5G  ./chainstate
```

utxos

they're pretty small, less than 64 bytes for everything (script, amount, outpoint...)

small, but lots of em! ~60M now

moar utxos


Unspent Transaction Outputs
70000000
65000000
60000000
55000000
50000000
45000000
40000000
35000000
30000000
25000000
20000000
15000000
10000000
2014-10
2015-1
2015-4
2015-7
2015-10
2016-1
2016-4
2016-7
2016-10
2017-1
2017-4
2017-7
2017-10
2018-1
2018-4
2018-7
2018-10
2019-1
2019-4
2019-7
source: statoshi.info

accumulators

wouldn't it be cool if we didn't store the utxo set at all, but people could prove their coins exist?

accumulators!

wallets

wallets track their own utxos

if you need to update proofs after every add / remove, do so to your utxos.

only 10s of utxos per wallet, so no problem, right...?

bootstrapping

problem: transition

I'm the first accumulator node. I've got proofs for all my utxos.

But nobody gives me proofs for anything! I can't validate

bridge node

The network needs, at least temporarily, a "Bridge Node"

Bridge Nodes maintain proofs for EVERY utxo

problematic for RSA accumulators where proof updates can't be aggregated

accumulators

a Merkle tree is like an accumulator.

...but you can't add to it if you only know the root

keep only the top (root)

prove inclusion of a leaf by giving a branch

utxo accumulator

let's make a hash-based accumulator for UTXOs!

A bridge node would just store the whole tree, and updates to the tree are inherently aggregated

Need to use a bunch of trees - O(log(n)) instead of O(1)

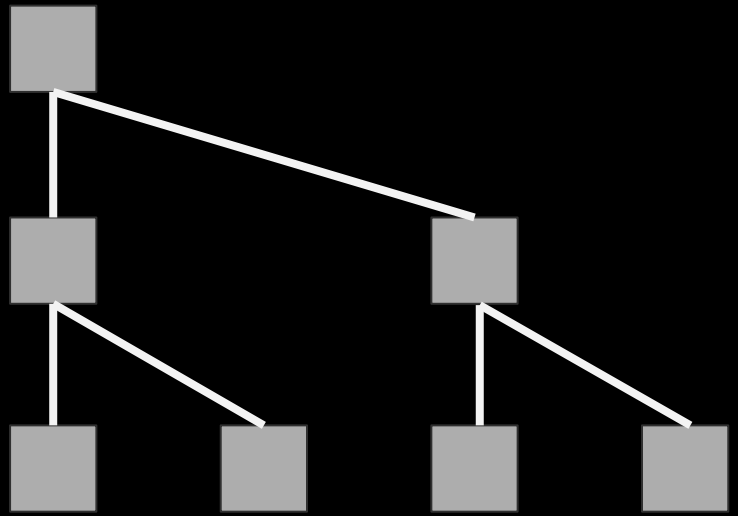
perfect forest

first, how to add leaves

Then how to delete leaves. More complex & novel.

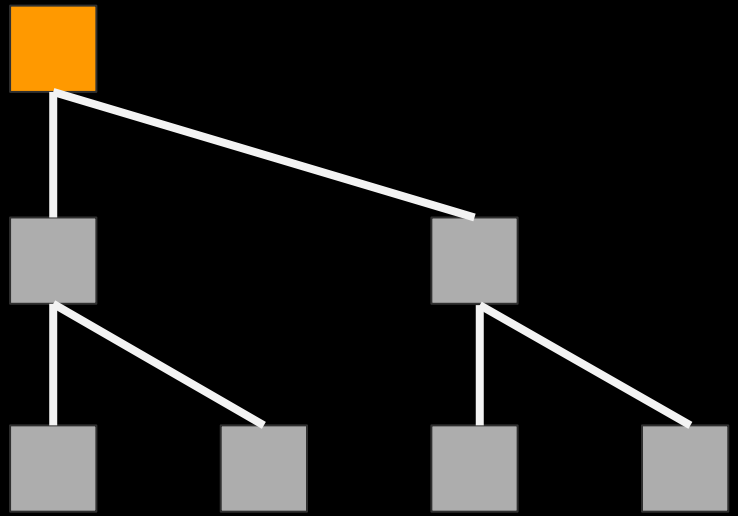
tree

It's got 4 leaves



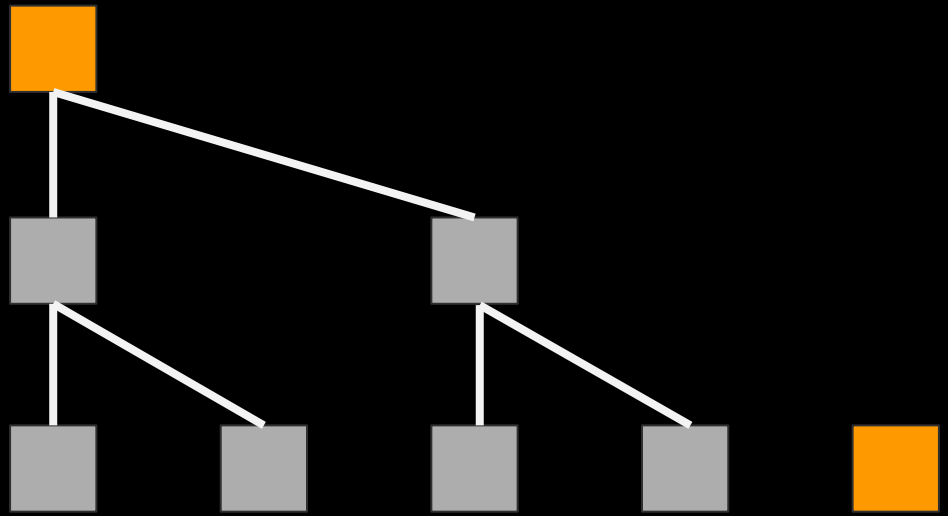
tree

only keep the root (top)



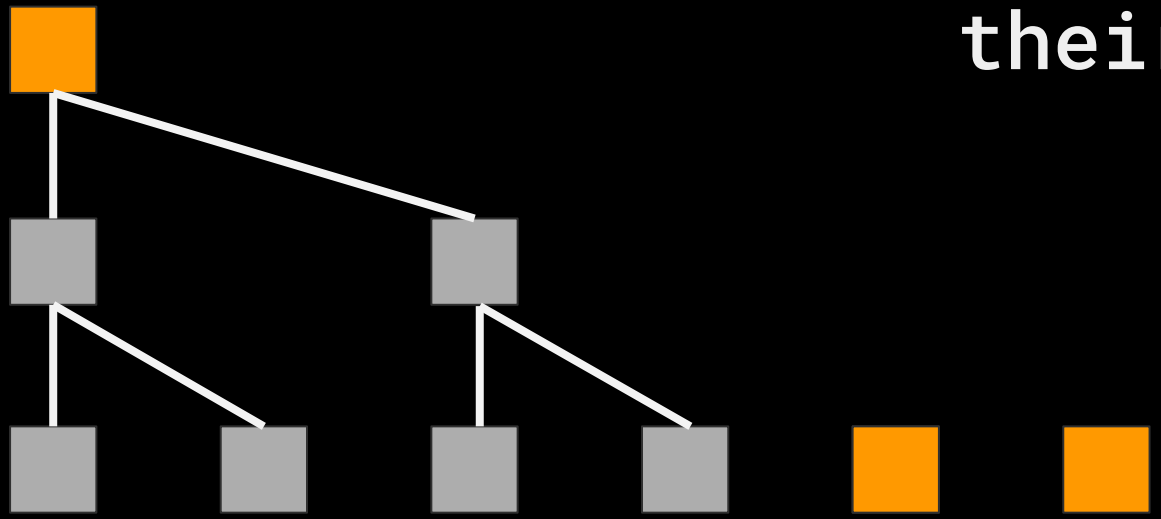
forest

Add a leaf -> 5
Now there are 2 trees.



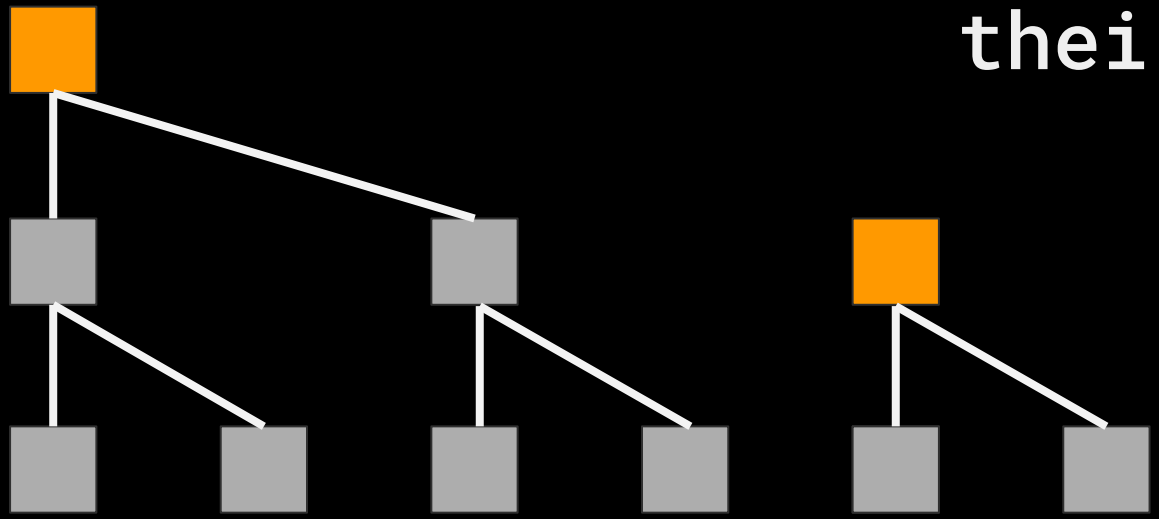
forest

Add another leaf -> 6. those 2 form their own tree.



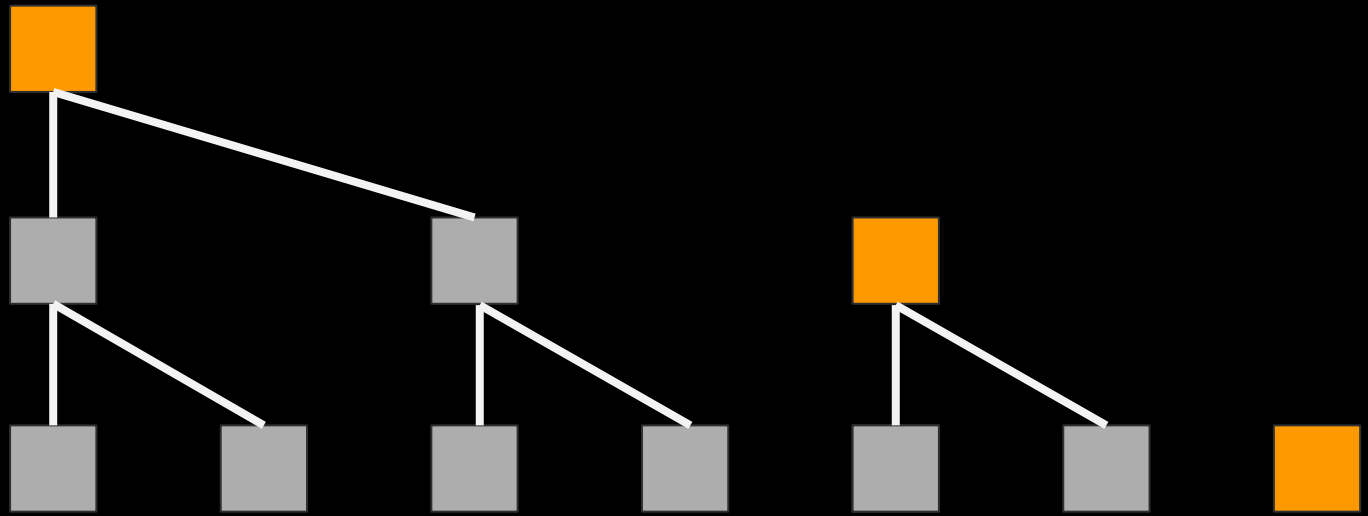
forest

Add another leaf -> 6. those 2 form their own tree.



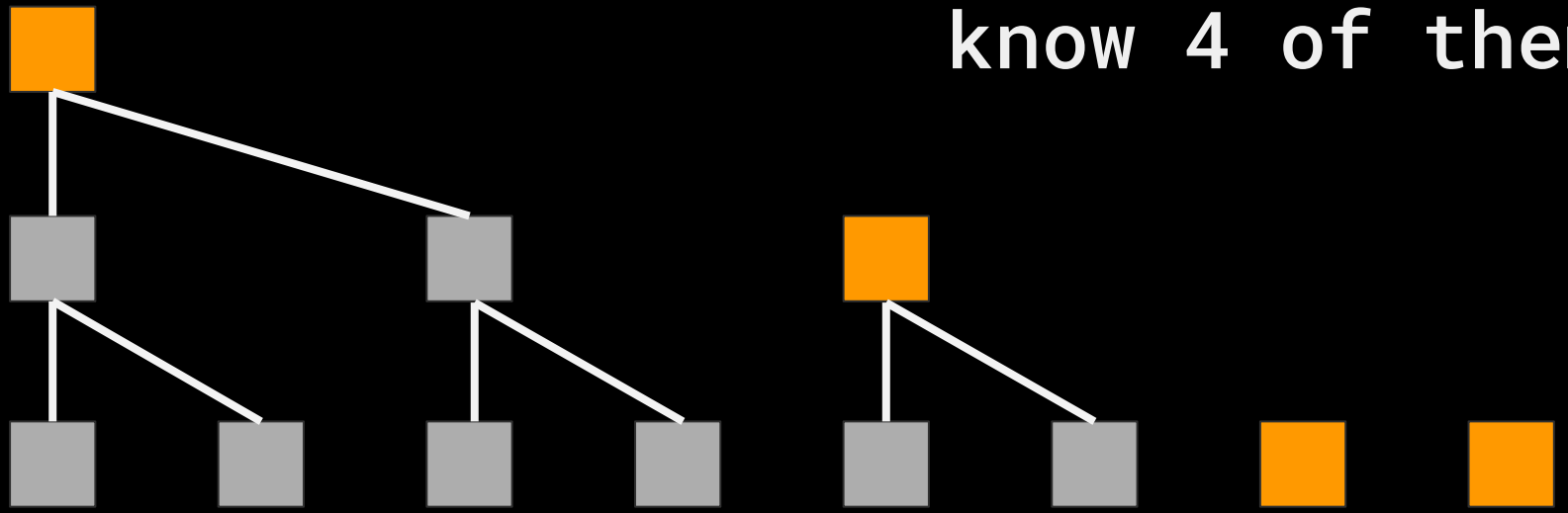
forest

Add again -> 7
3 trees



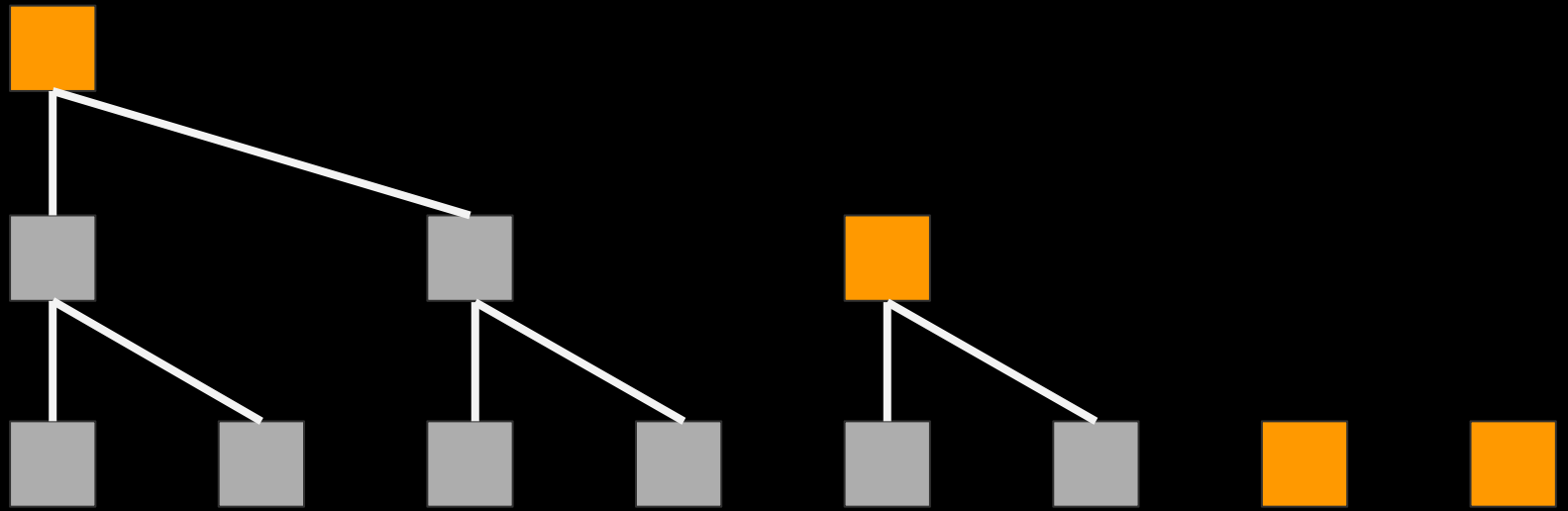
tree

Add another.
Now there are 8 leaves, and we know 4 of them.



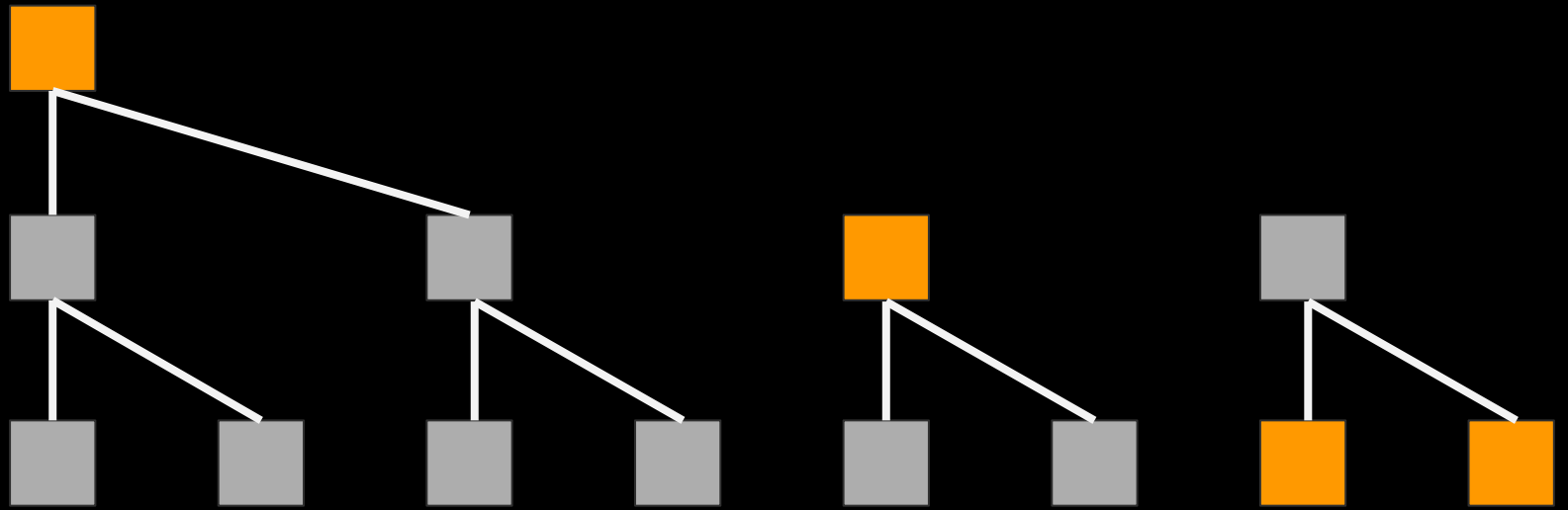
tree

combine...



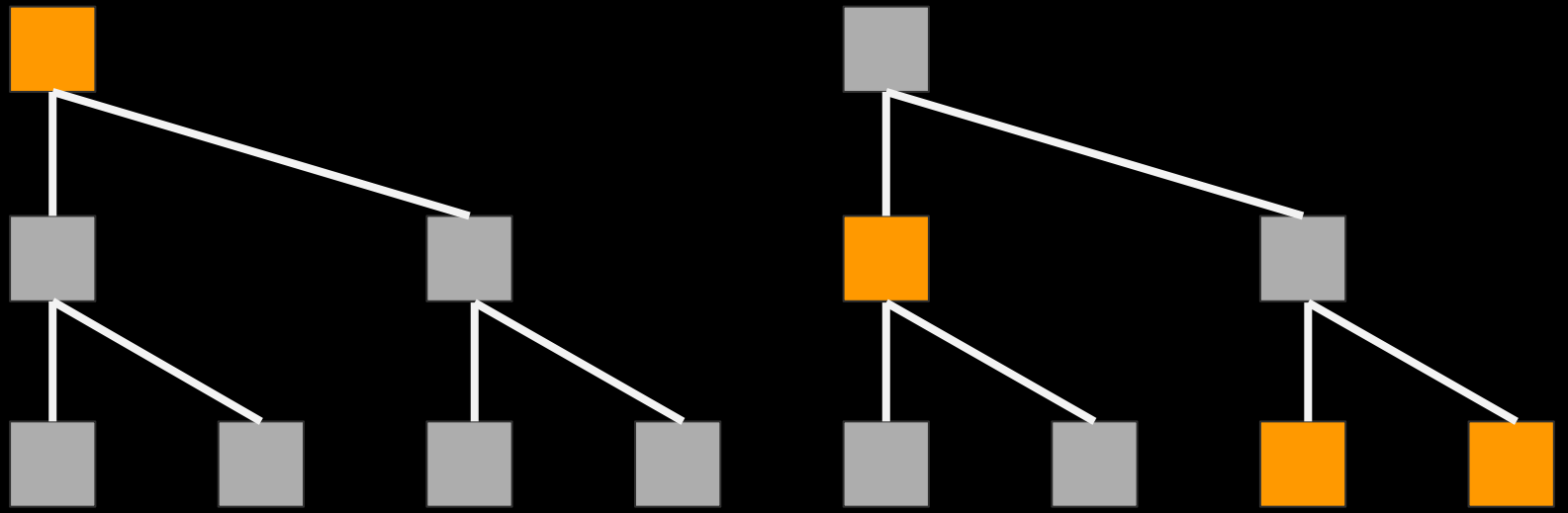
tree

combine..



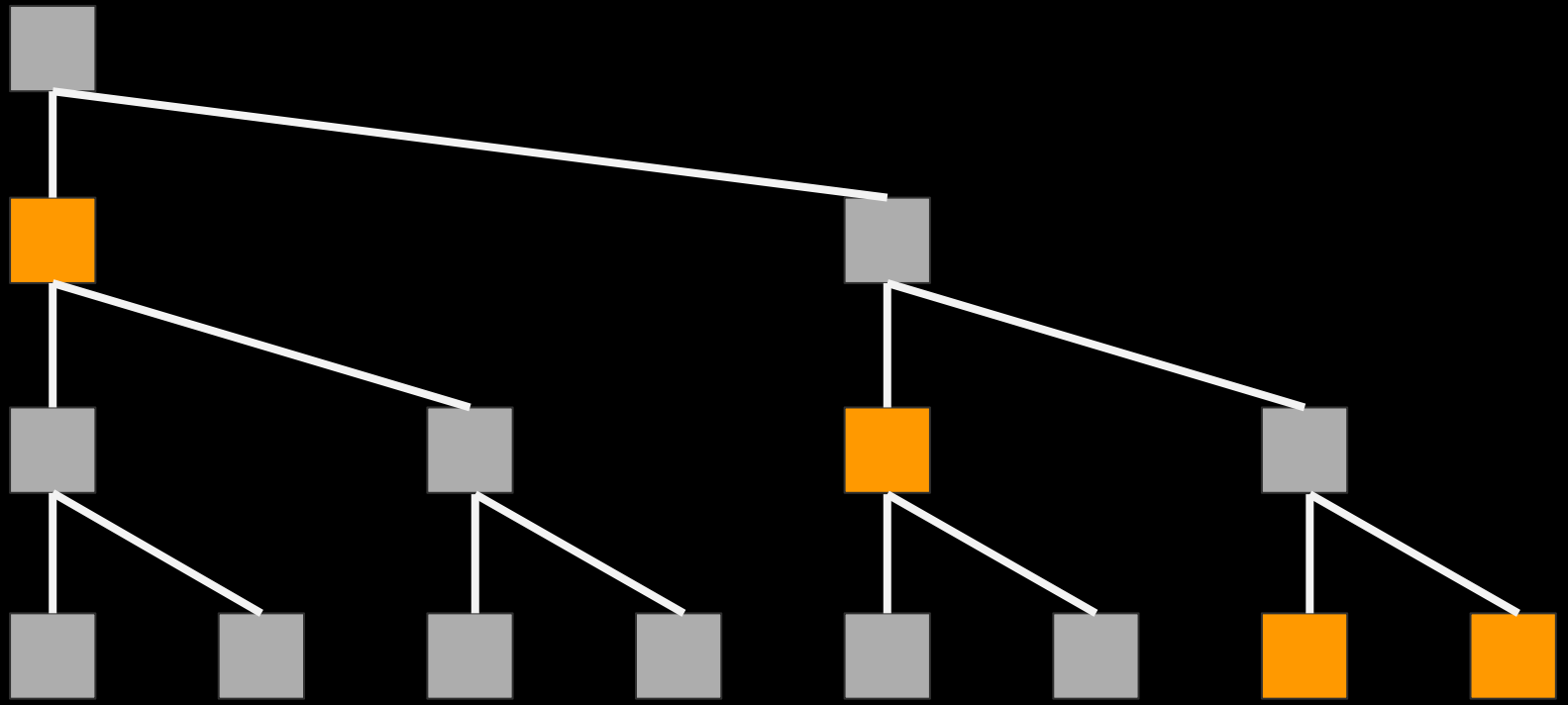
tree

combine..



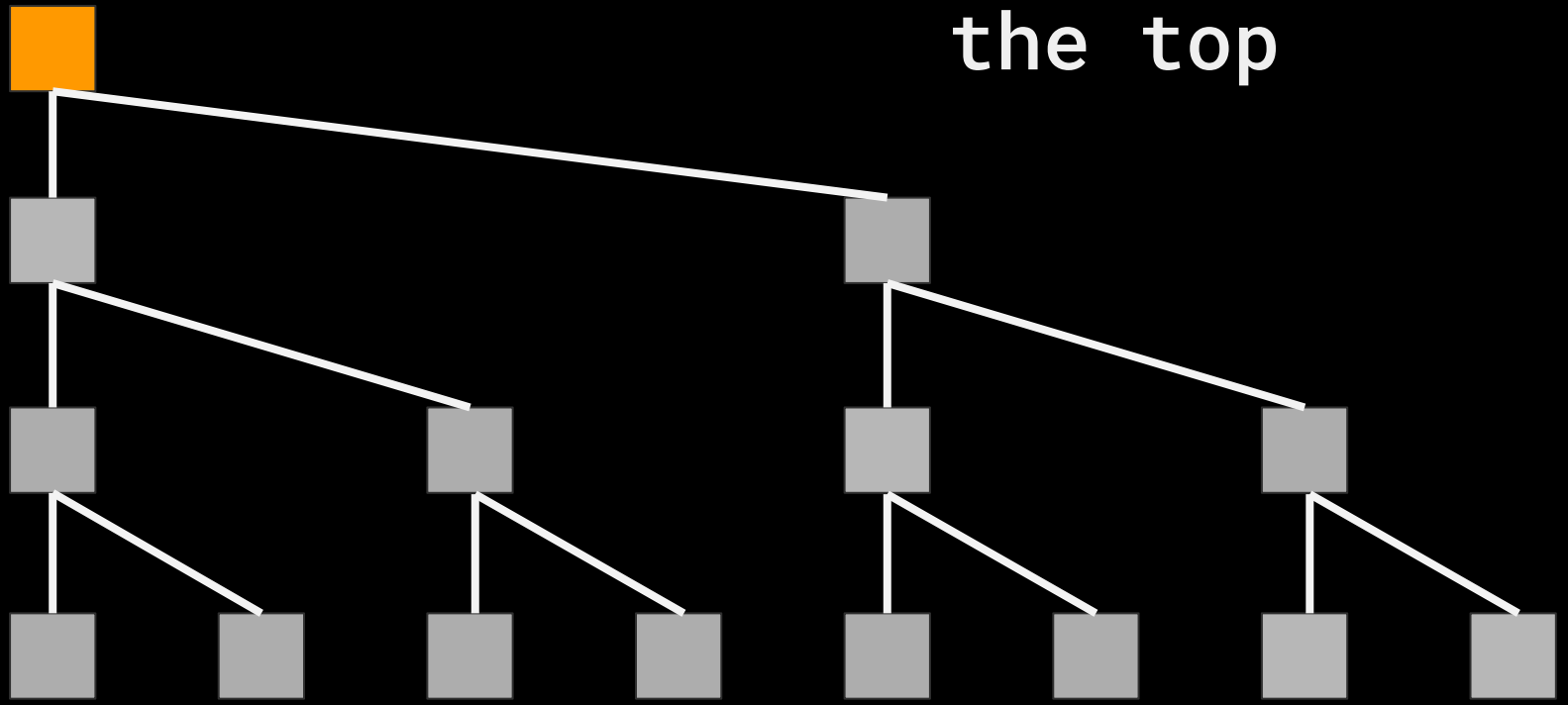
tree

combine..



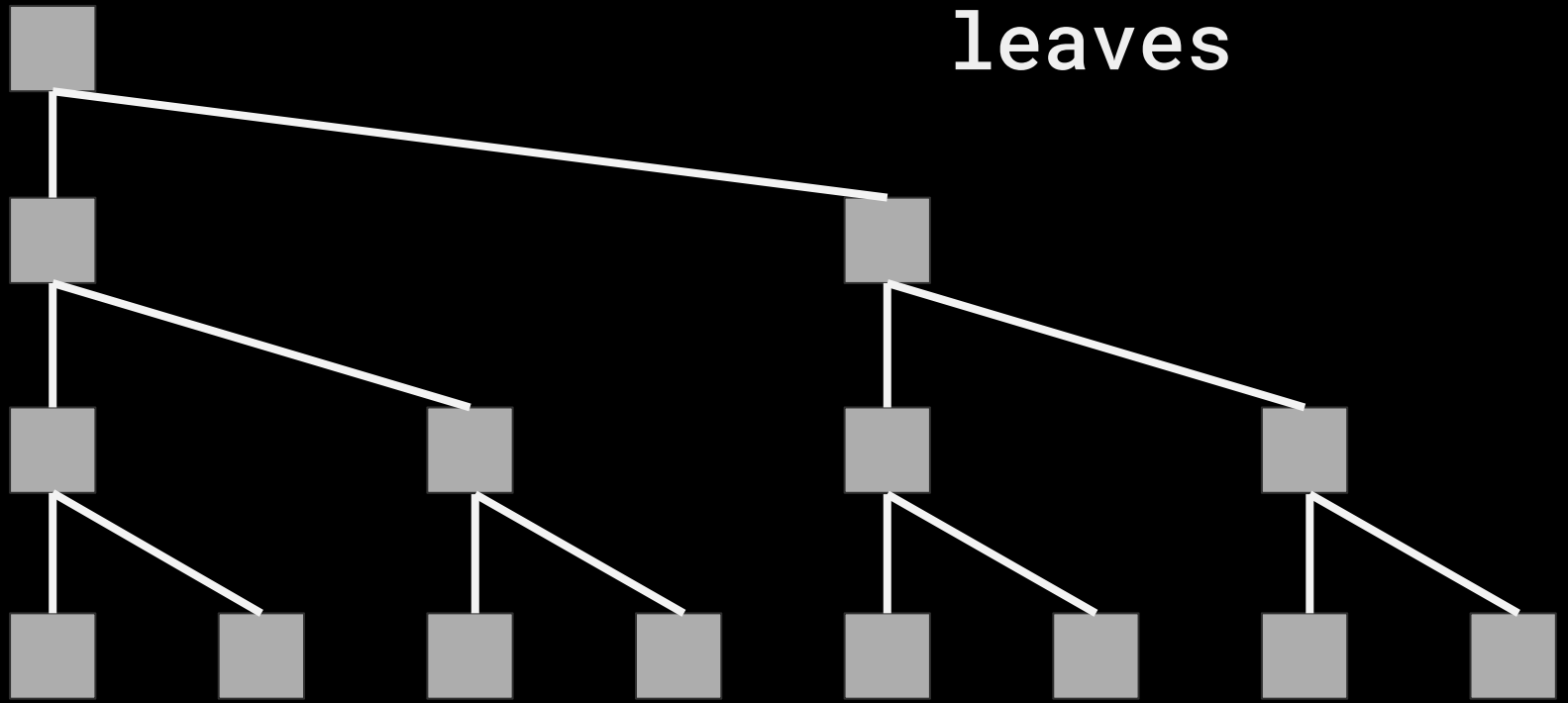
tree

forget all but the top



tree

It's got 8 leaves



perfect forest adding

adding new leaves is pretty cool

we can add on the bottom right, and always have enough data to create a forest of perfect trees (all trees have 2^n leaves)

deleting

delete maintaining perfect trees,
with no empty leaves

Here's how!

First, prove. Then, row by row:
twin / swap / root
then up to the next row

deleting

basic idea (visuals to follow)

twin: skip over two deleted siblings

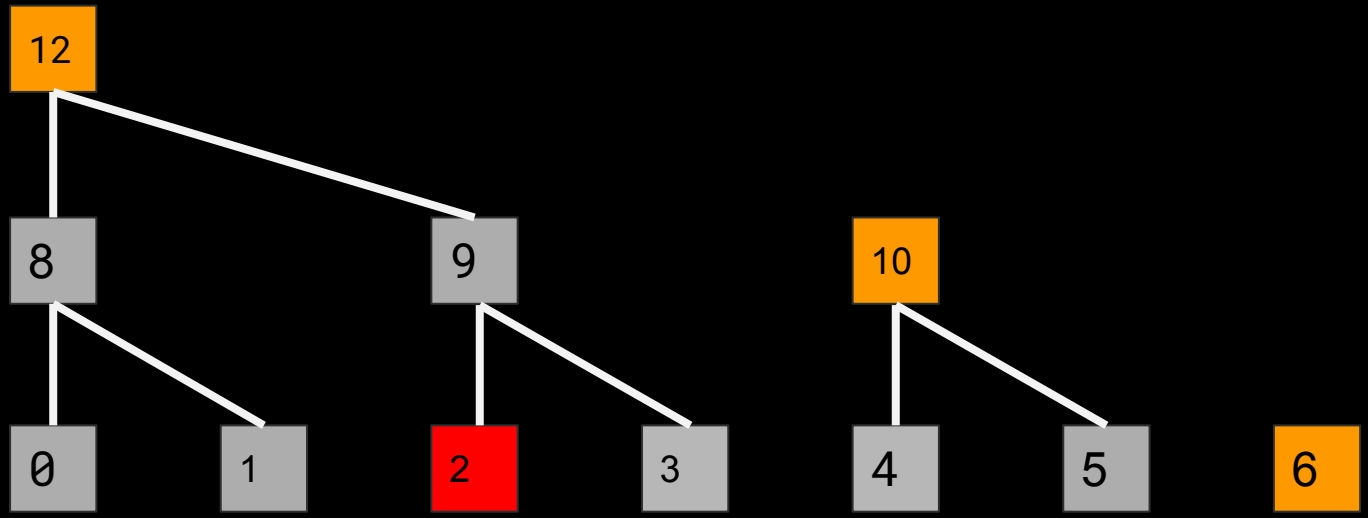
swap: move nodes around to get twin pairs of deletions

root: move to or from the root on that level

(note twin & swap are optimizations, you could do it with just root, 1 at a time)

delete example 1

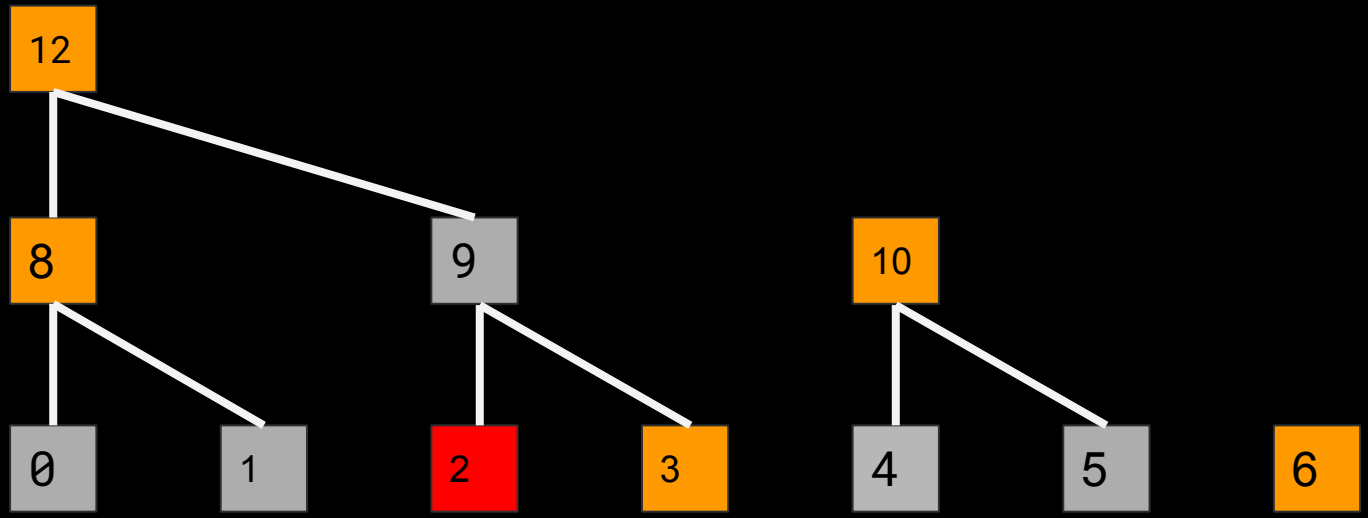
delete 2



delete example 1

delete 2

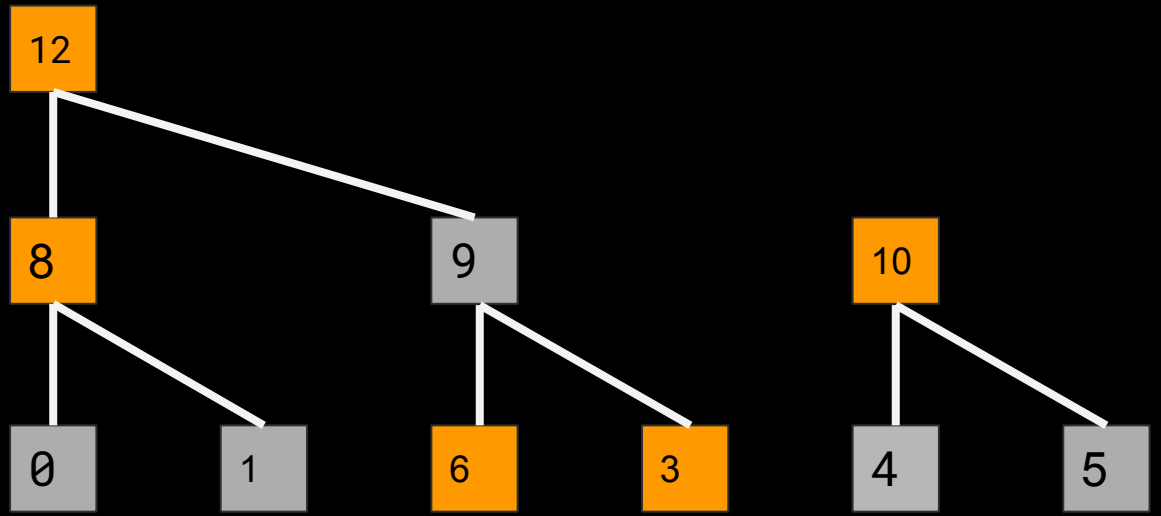
proof is 3, 8



delete example 1

6:root on row 1

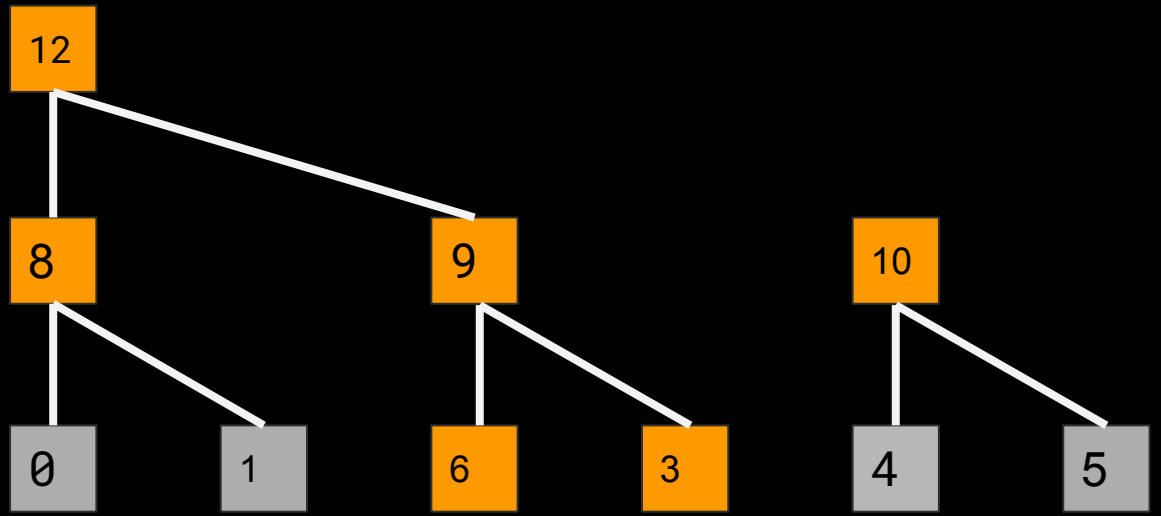
move to 2



delete example 1

compute new 9

compute new 12

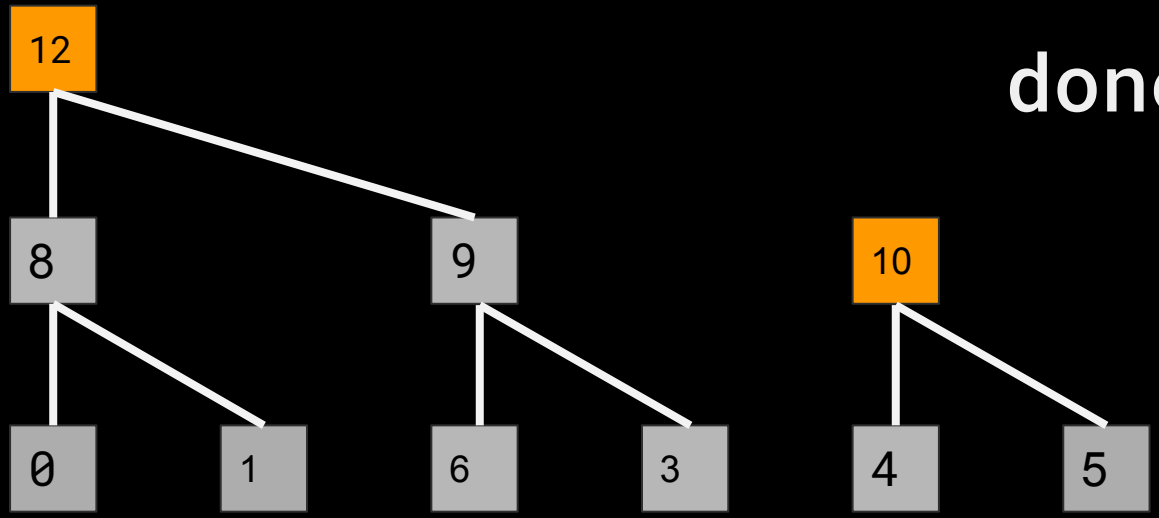


delete example 1

discard 6, 3,

8, 9

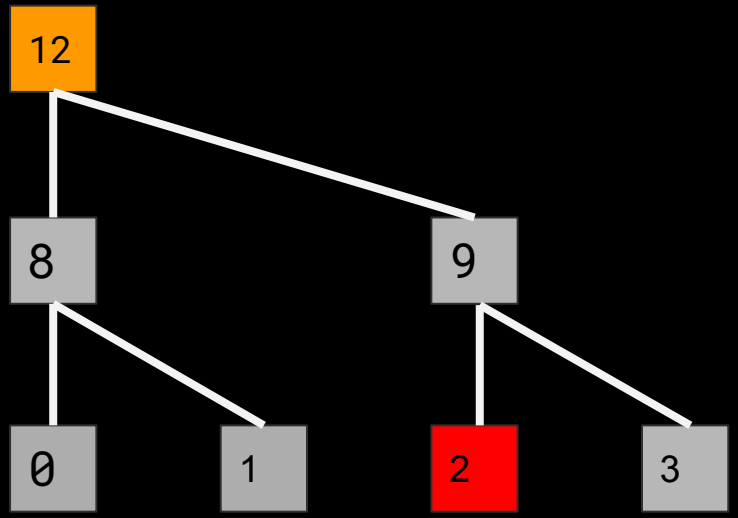
done



delete example 2

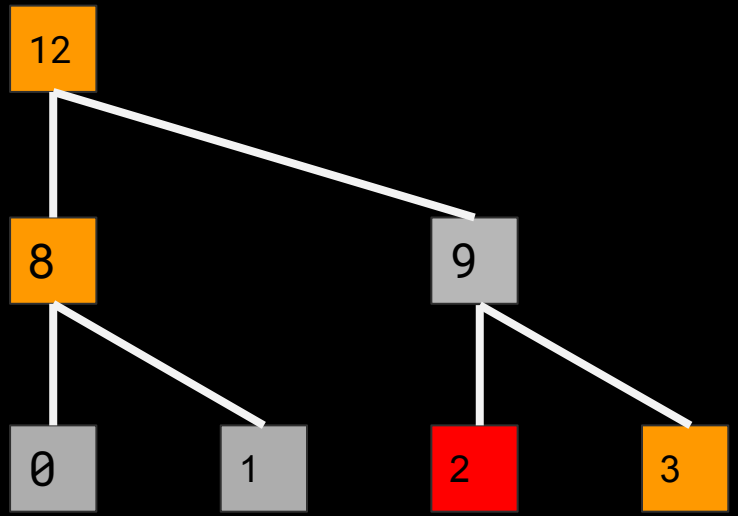
delete 2

(4 leaves)



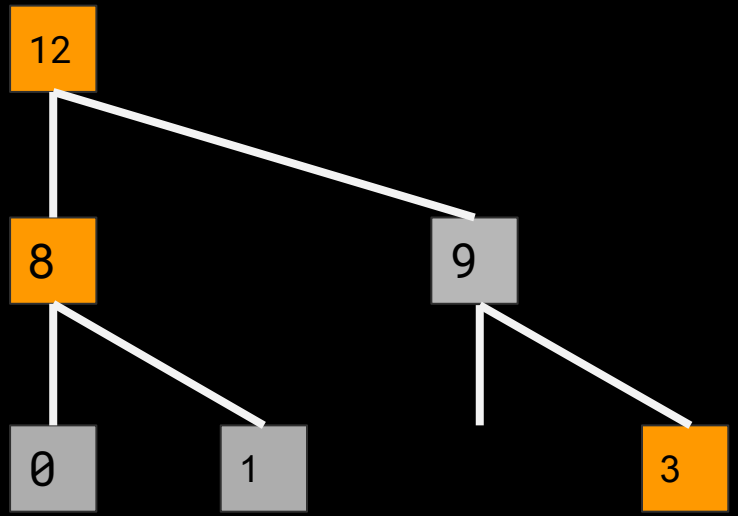
delete example 2

proof is 3, 8



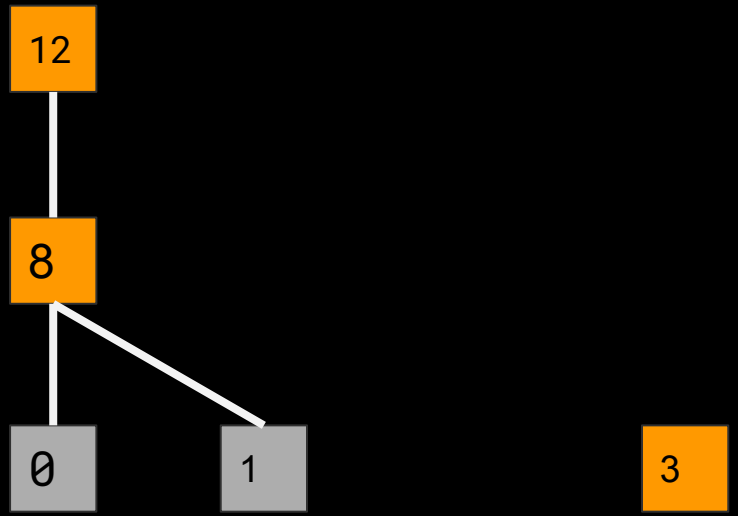
delete example 2

3 becomes root



delete example 2

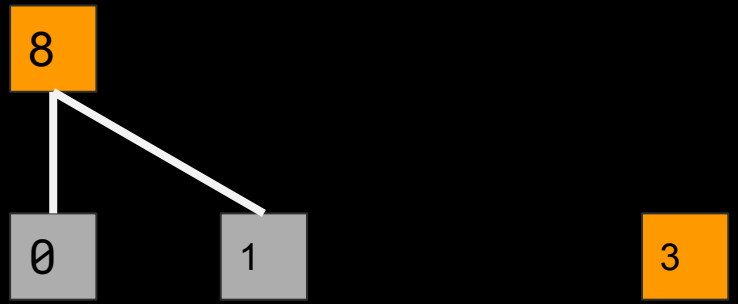
8 becomes root



delete example 2

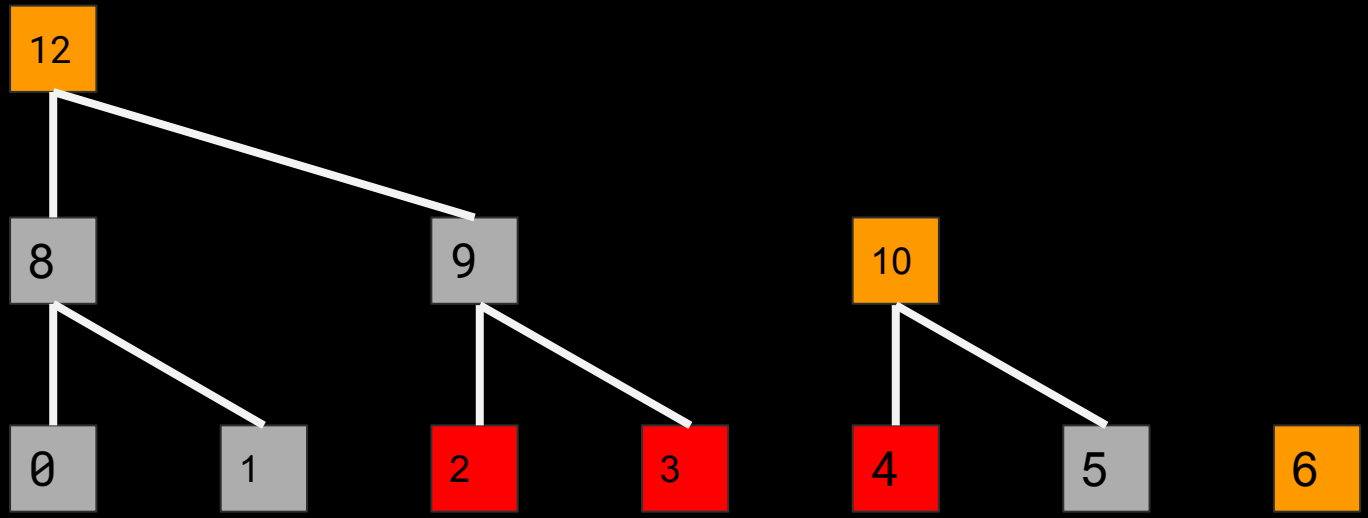
12 deleted

done



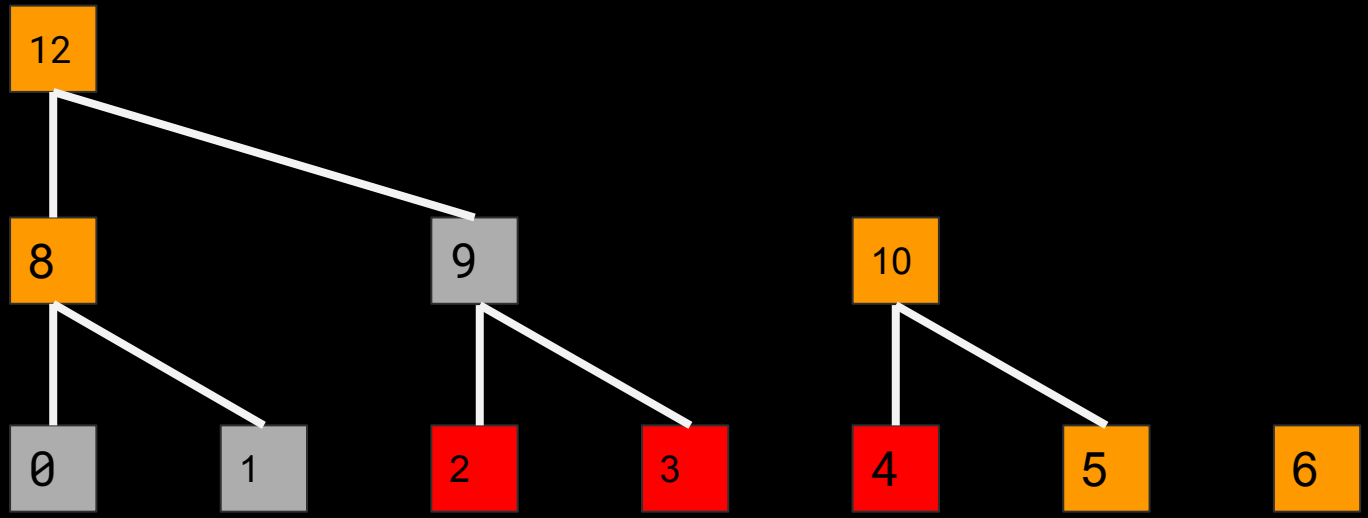
delete example 3

delete 2, 3, 4



delete example 3

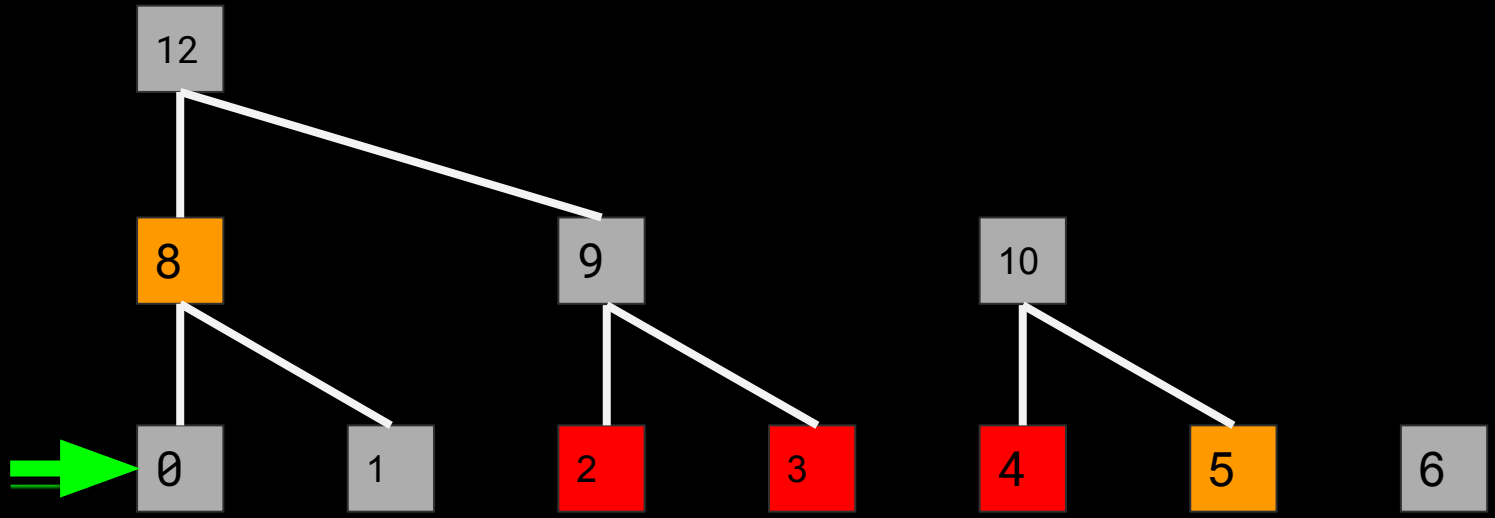
proof is 5, 8



row 0: twin

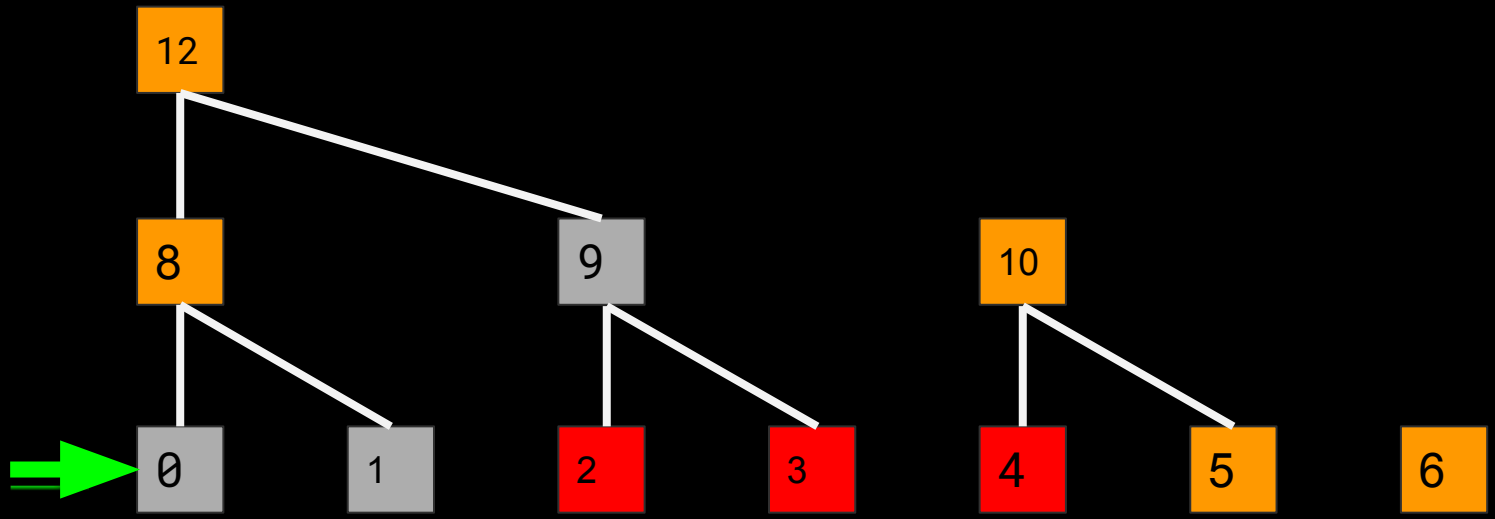
2, 3 are twins,

OK



row 0: swap

nothing to swap

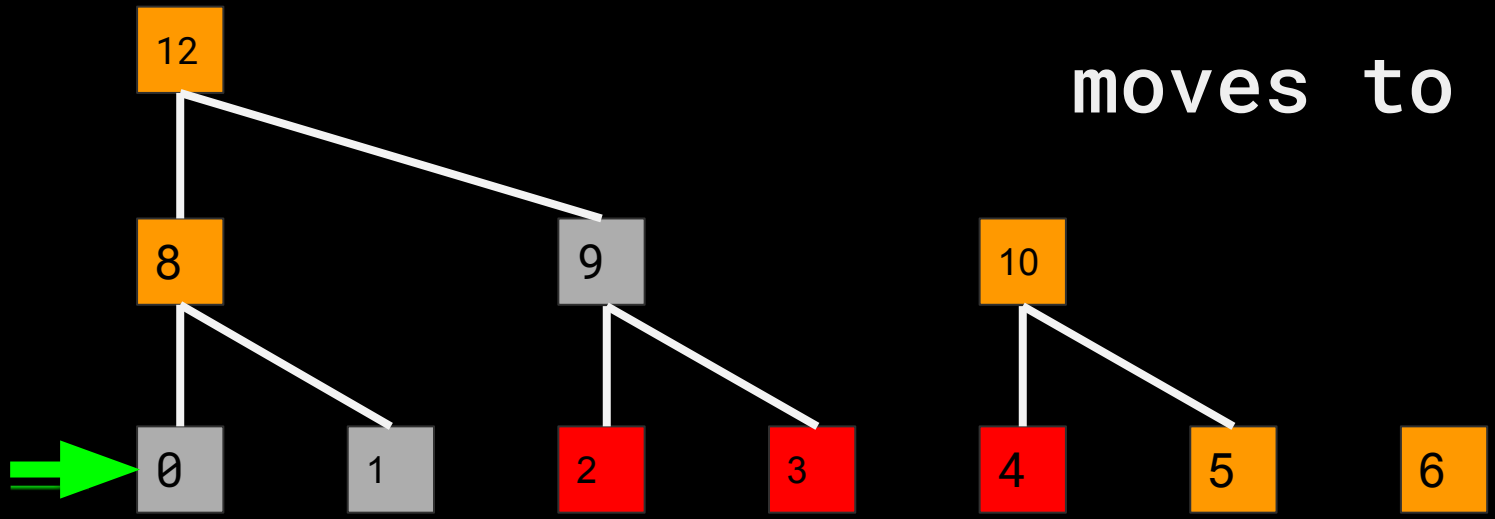


row 0: root

4 last deletion,

6 is root. 6

moves to 4

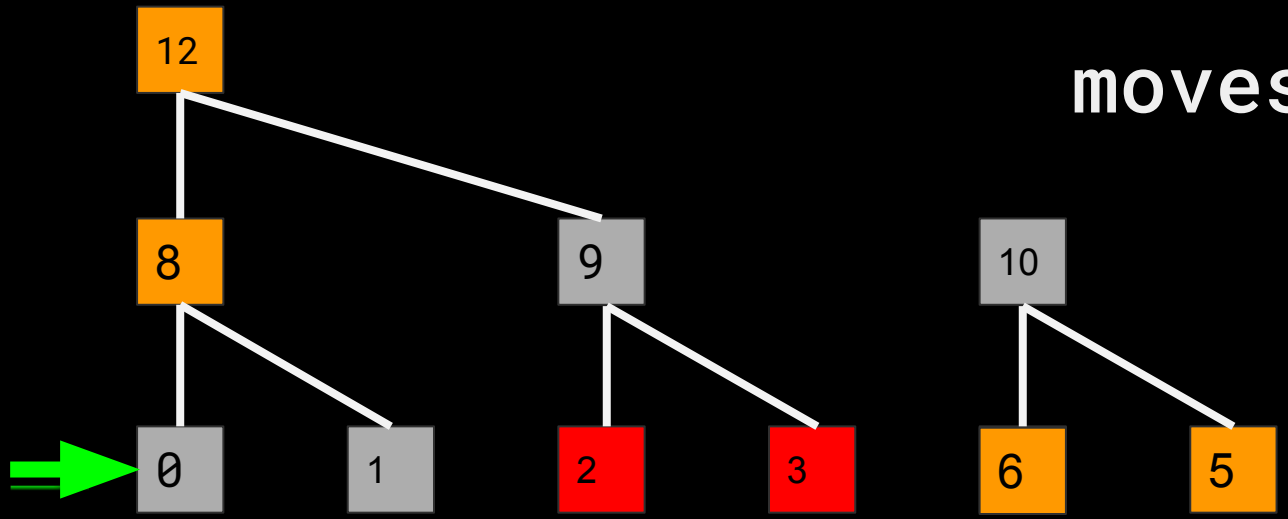


row 0: root

4 last deletion,

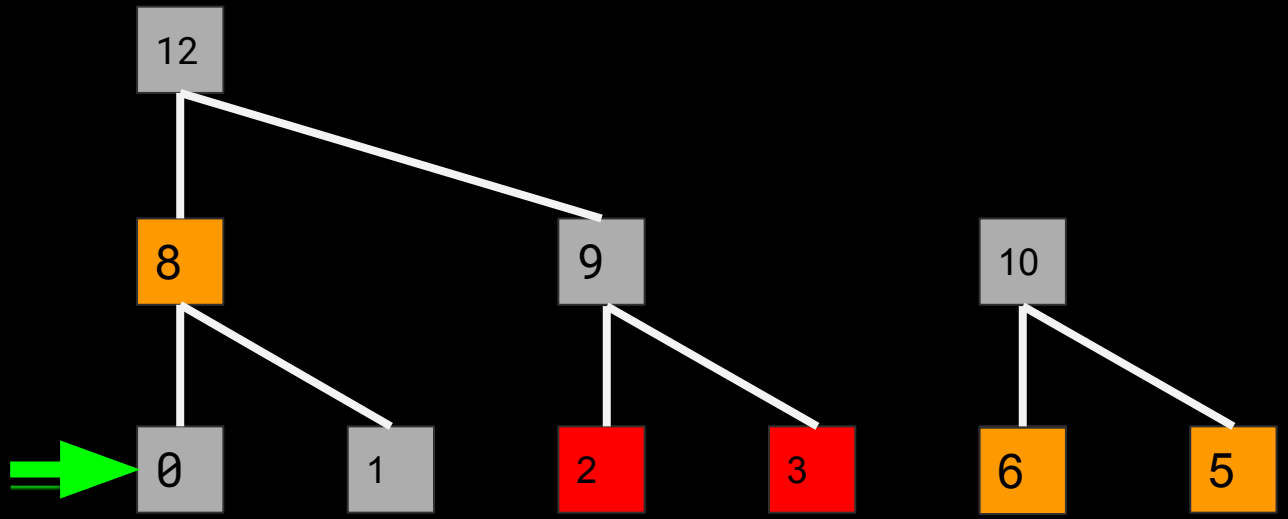
6 is root. 6

moves to 4



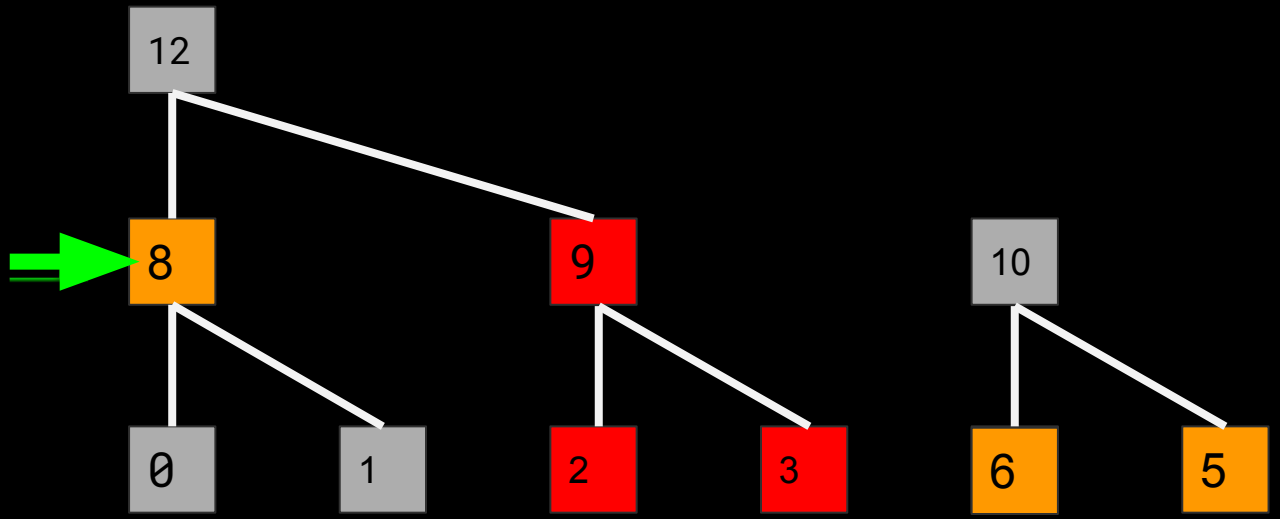
row 0 -> row 1

delete 9



row 0 -> row 1

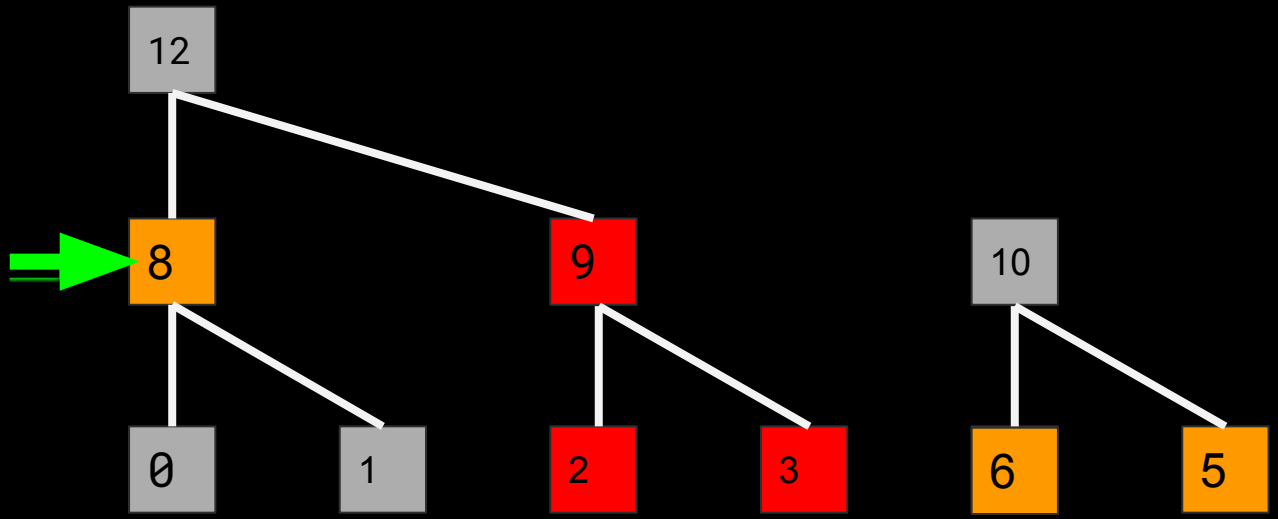
delete 9



row 1: no twin / swap

only 1 deletion,

go to root phase

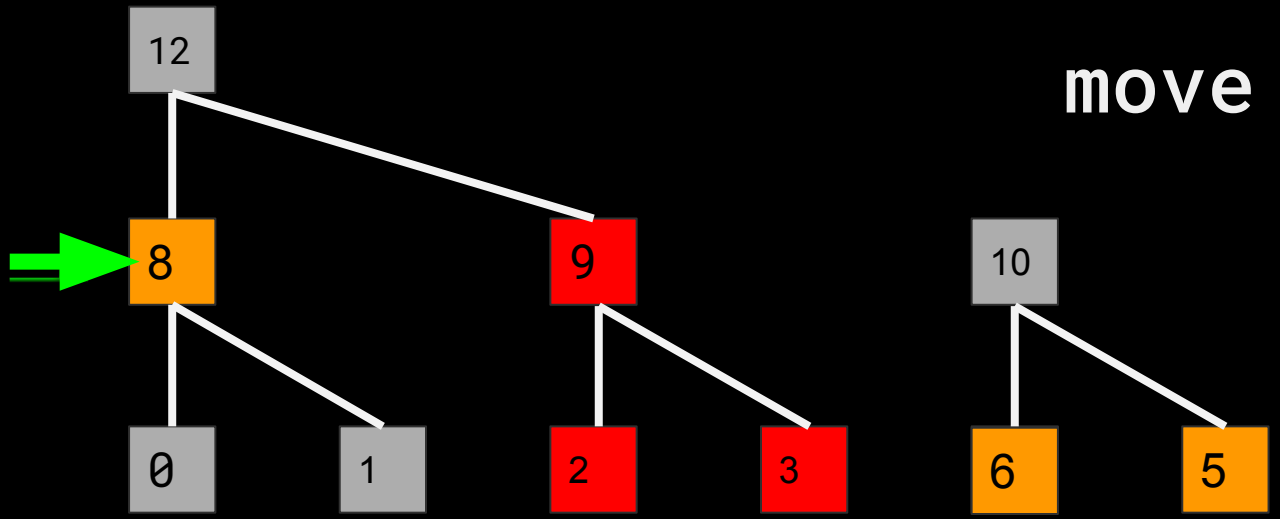


row 1: root

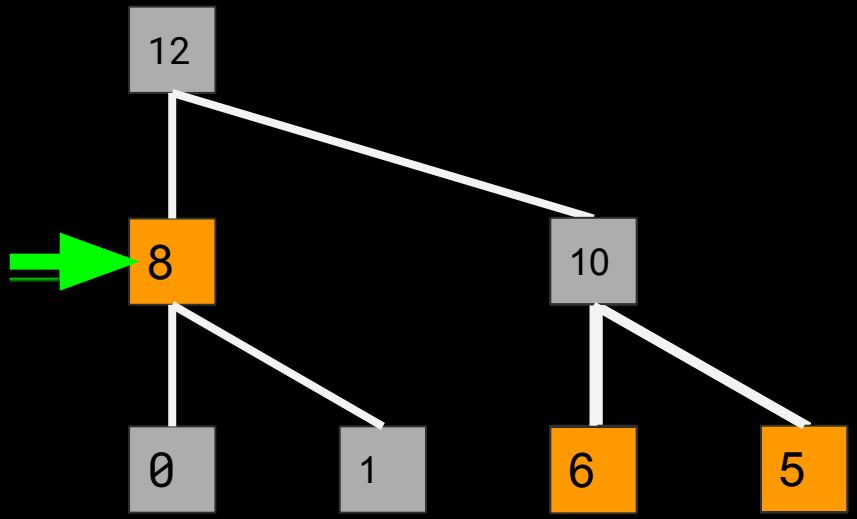
there is a root,

10

move 10 to 9



row 1: root

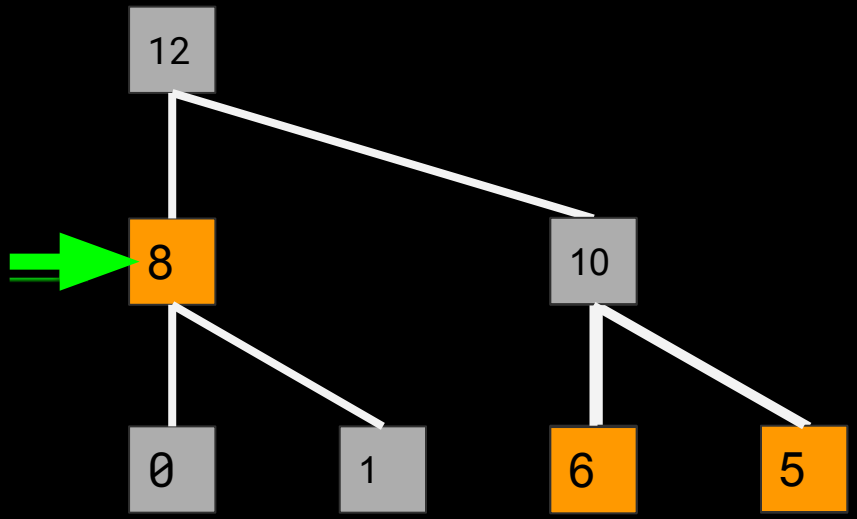


there is a root,

10

move 10 to 9

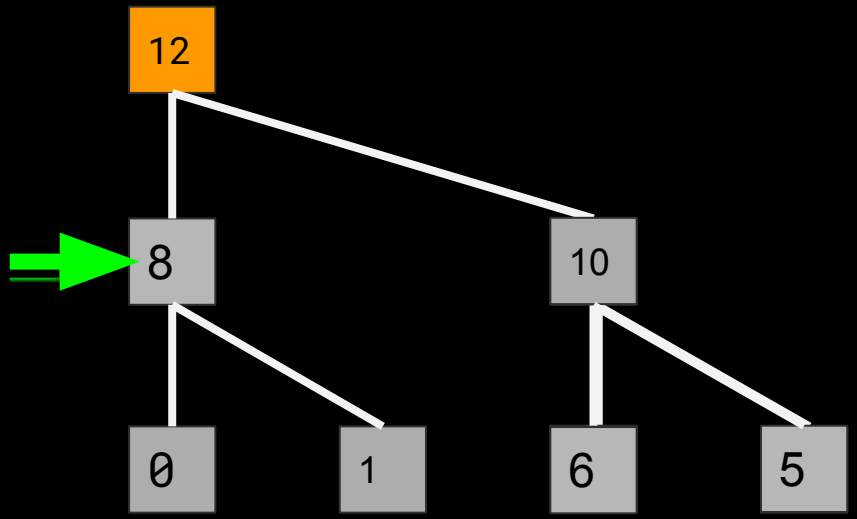
row 1: done



no more deletions; we're done!

compute new root at 12

row 1: done



no more deletions; we're done!

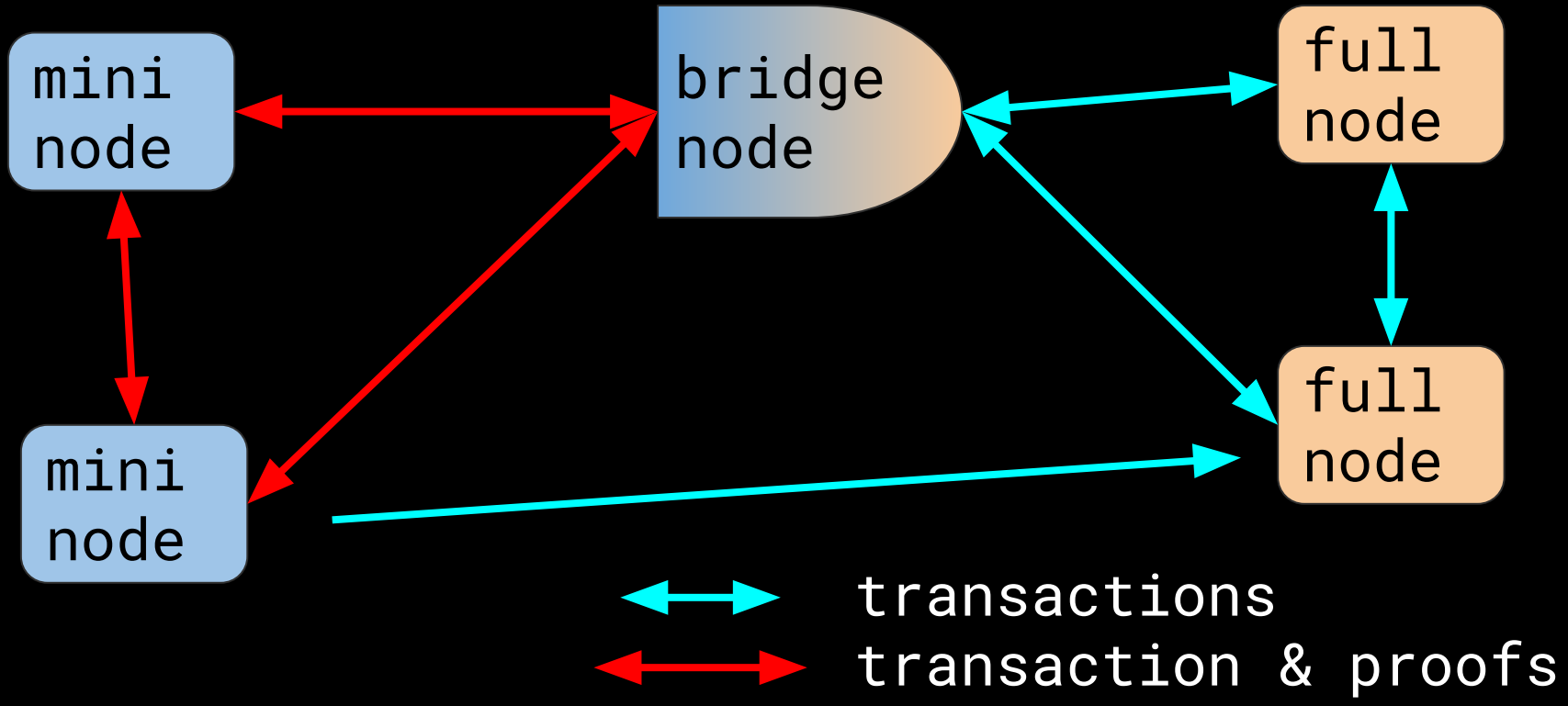
compute new root at 12

full node

Can run a node that validates every transaction and signature, while storing very little.

Every transaction now needs to prove that the coins it spends exist, because we don't save them to disk.

bridge network



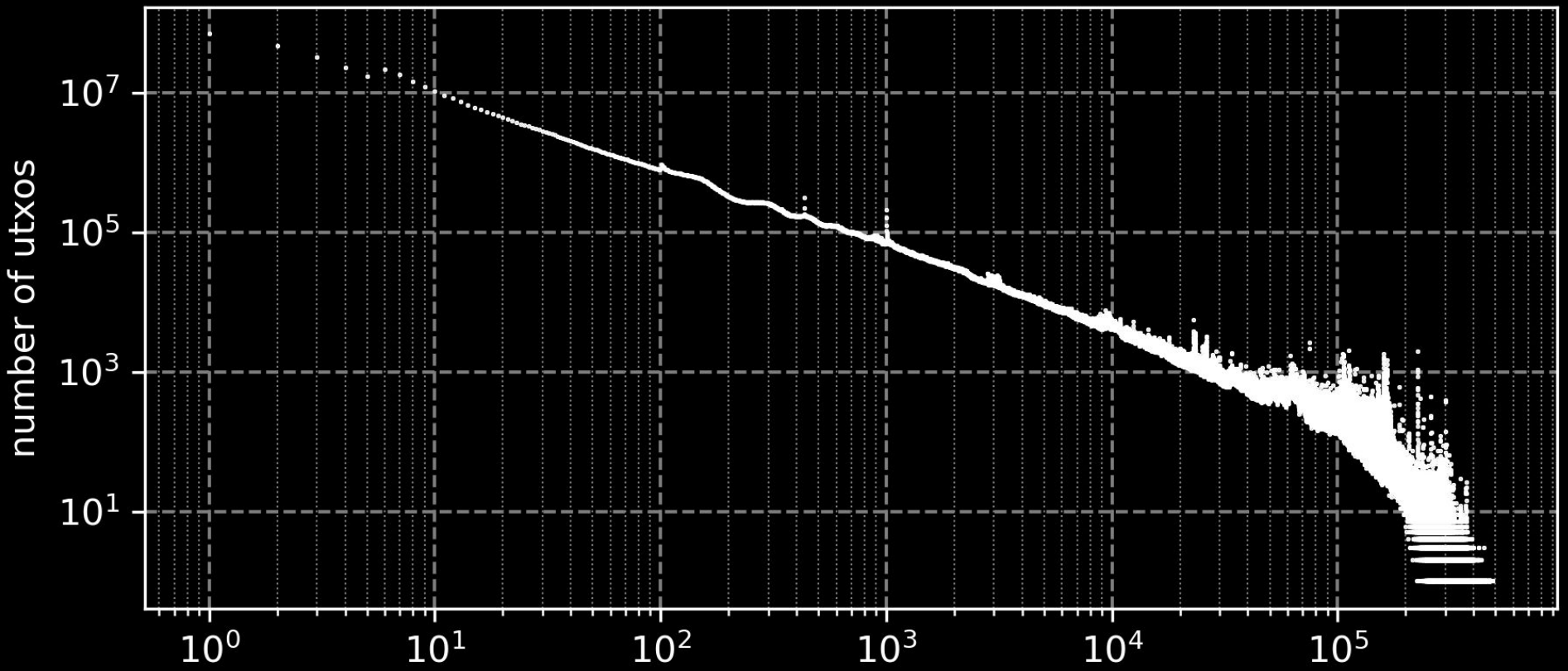
proof sizes

biggest downside: now there are all these proofs! How big are they?

1 proof is around 20 hashes, with 5000 inputs in a block, that's 3.2MB! 4X retroactive block size increase!

Need ways to cut that down:

proof sizes: utxo lifespan



proof size

Naively, proofs are several times the transactions. IBD would be ~600GB of proofs (+250GB of tx data!)

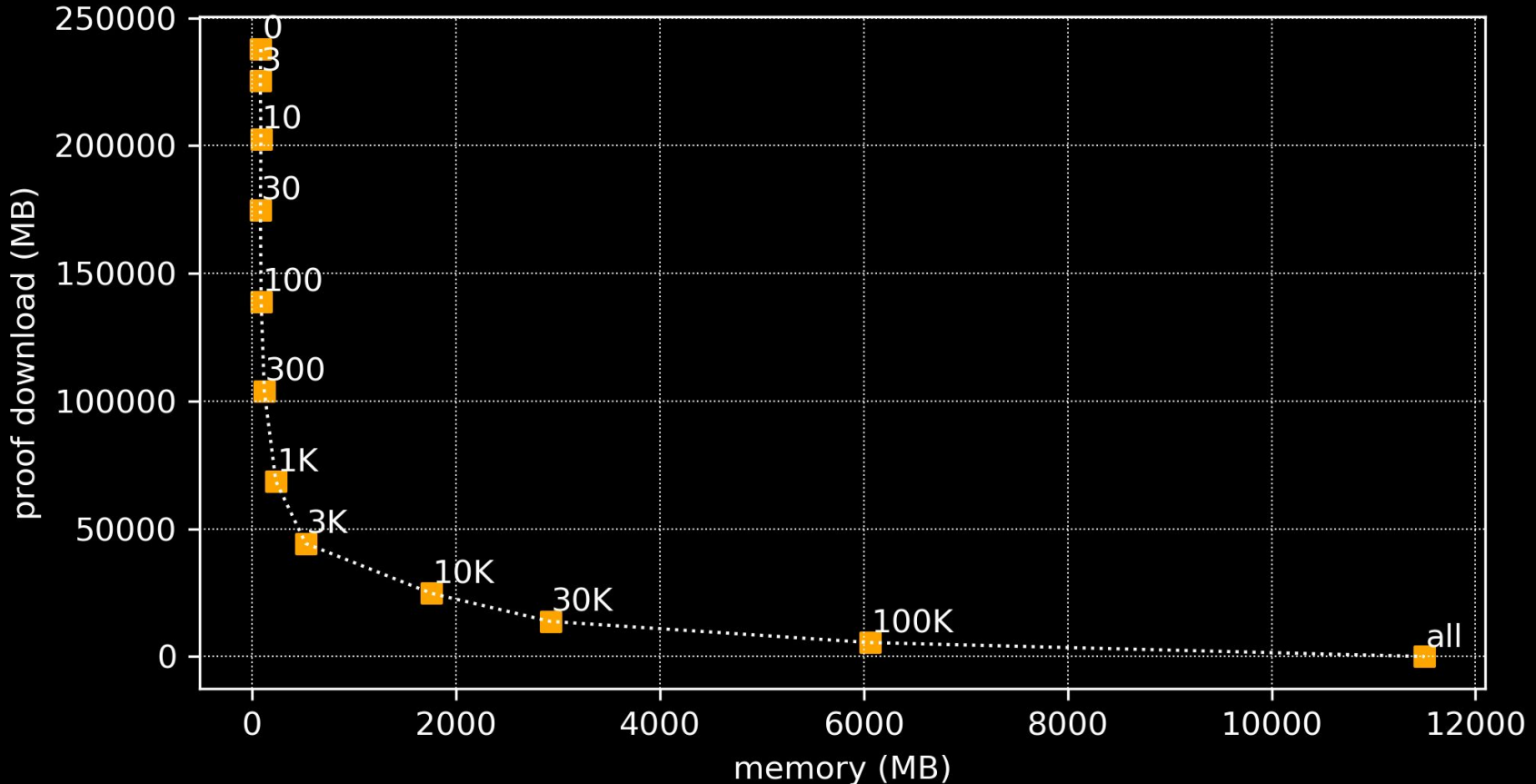
But proofs aggregate in a block, as we saw. That brings IBD down to 7.5G hashes (~250GB)

IBD hints

The IBD server "knows the future"; the client is downloading block 50, but the server has up to block 9000.

The server can give hints about what happens next. Which leaves get deleted soon, and thus which to remember.

Results: IBD to block 546000



no consensus? no problem

Not a fork. Permission not required!

Need to start with a bridge node, and archive nodes which send block proofs

on github! many things to optimize!
github.com / mit-dci / utreexo

issues! PRs! IRC #utreexo