Turbo-Geth and Stateless Ethereum

Supported by:

ethereum foundation

ETHEREUM CLASSIC LABS

CONSENSYS GRANTS

Gitcoin grants

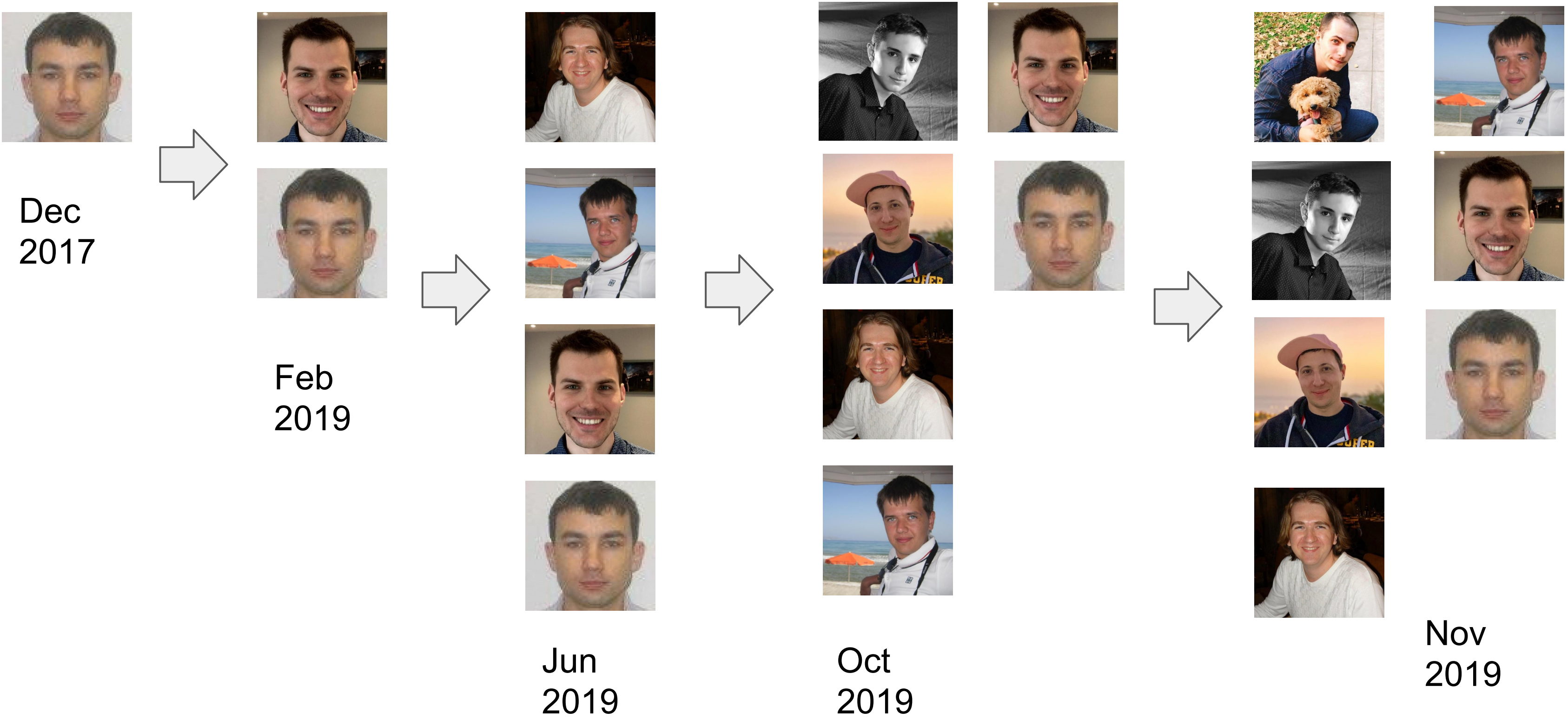
GRANTS

Supported in the past by:

INTERCHAIN FOUNDATION

INFURA

Team



Dec 2017

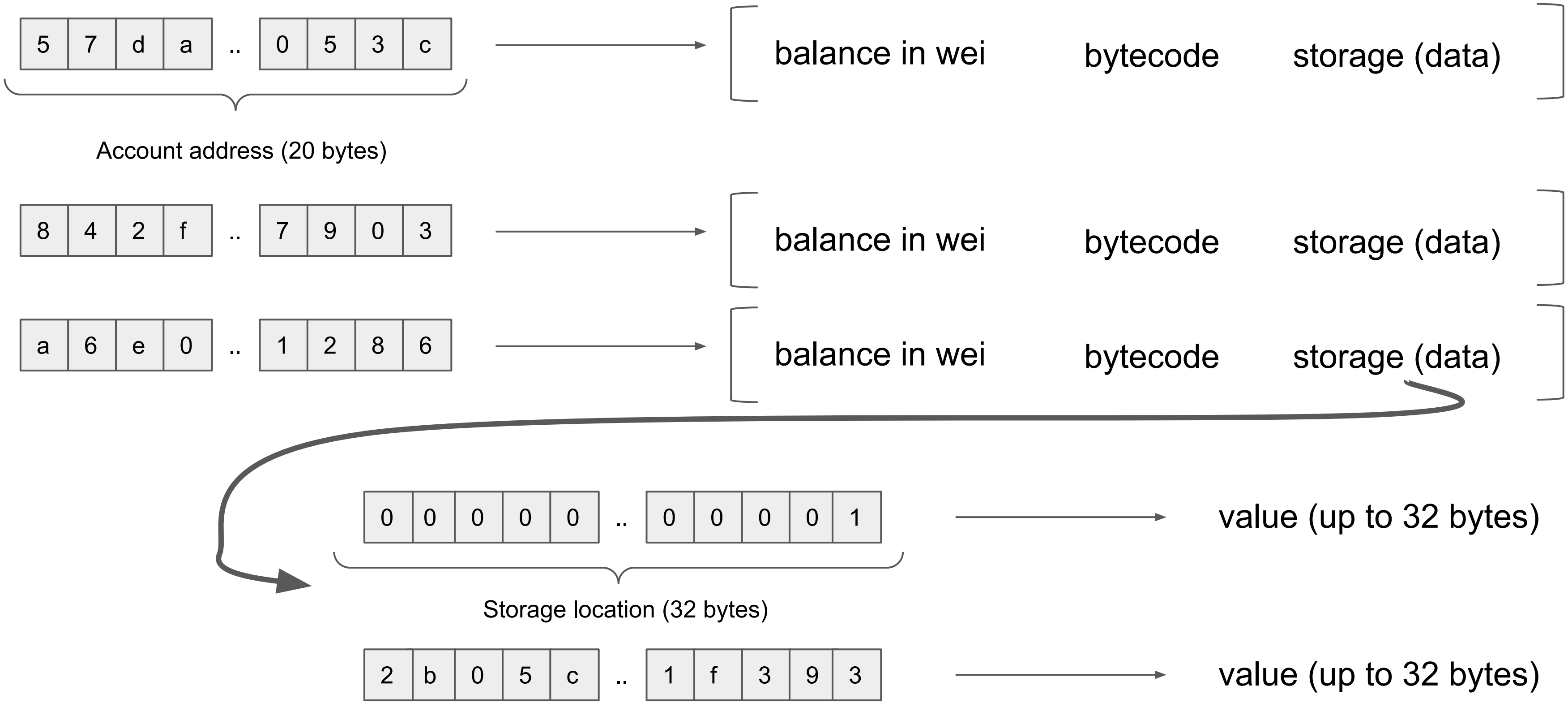
Feb 2019

Jun 2019

Oct 2019

Nov 2019

Ethereum State from EVM’s point of view



5 7 d a .. 0 5 3 c → balance in wei bytecode storage (data)

Account address (20 bytes)

8 4 2 f .. 7 9 0 3 → balance in wei bytecode storage (data)

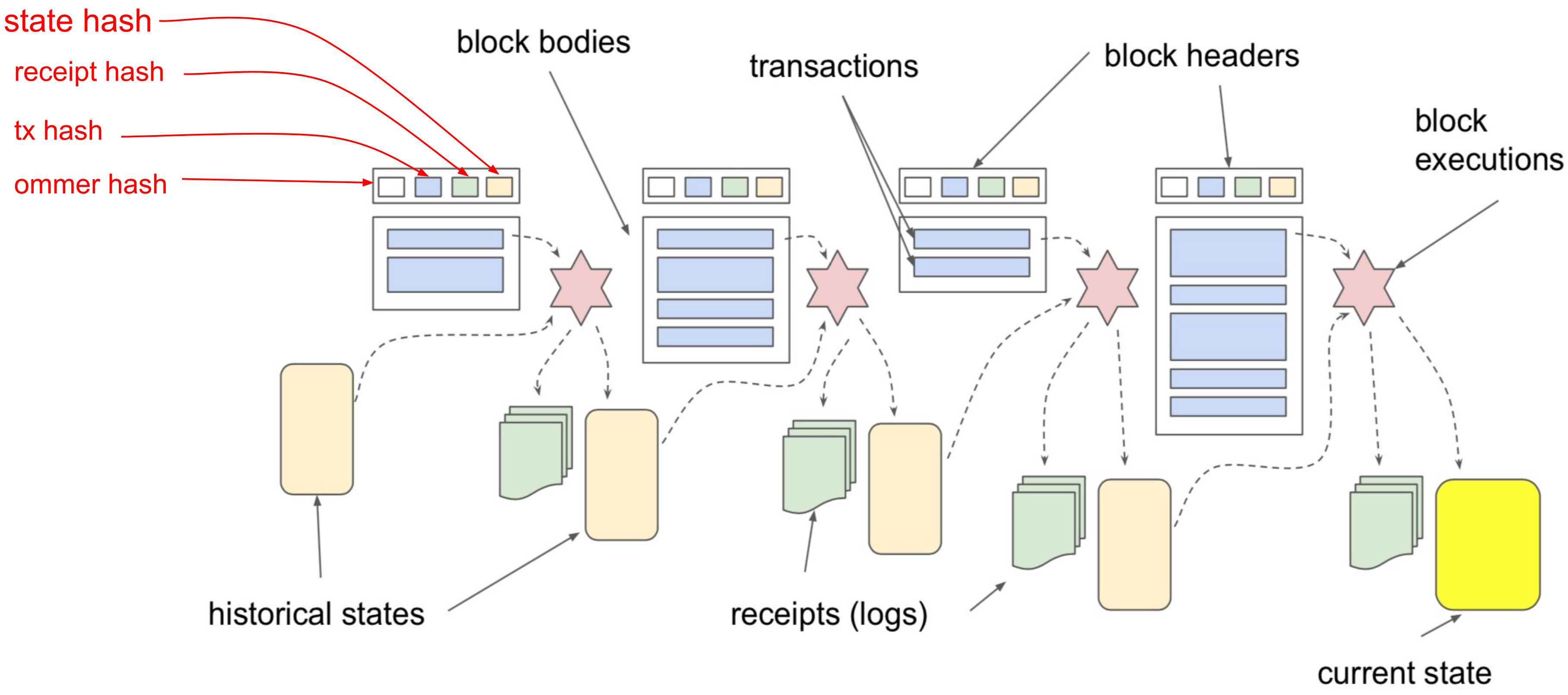
a 6 e 0 .. 1 2 8 6 → balance in wei bytecode storage (data)

0 0 0 0 0 .. 0 0 0 0 1 → value (up to 32 bytes)

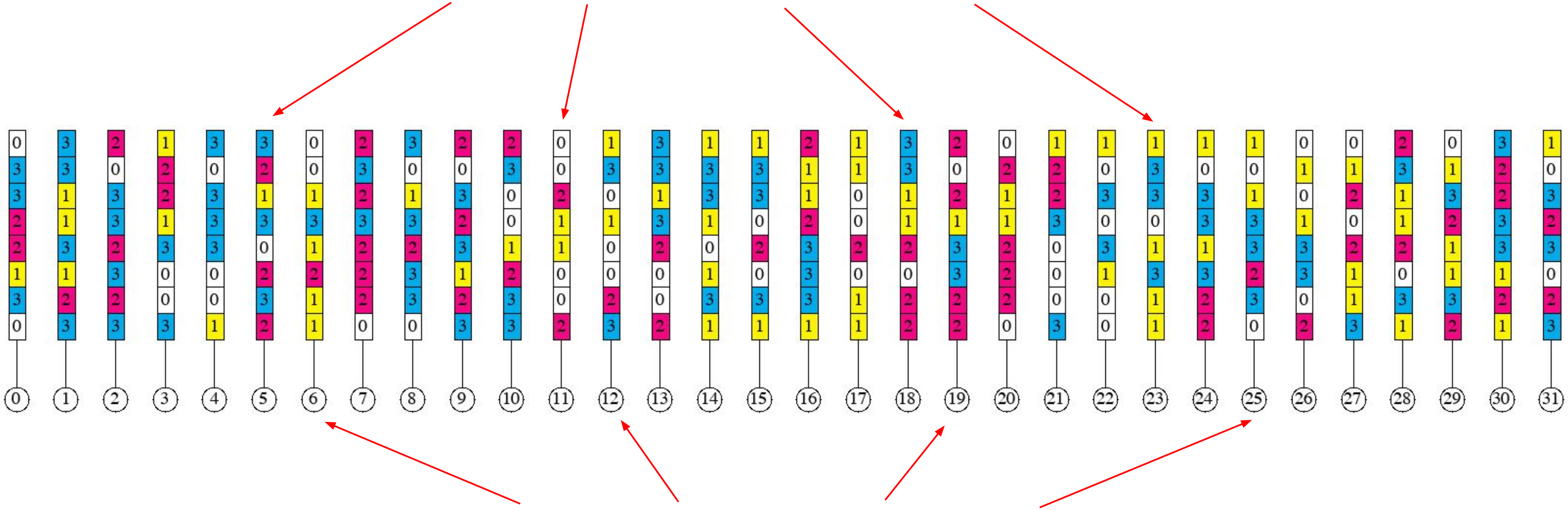
Storage location (32 bytes)

2 b 0 5 c .. 1 f 3 9 3 → value (up to 32 bytes)

Commitments in Ethereum Headers



How state root is constructed (quad instead of hex)

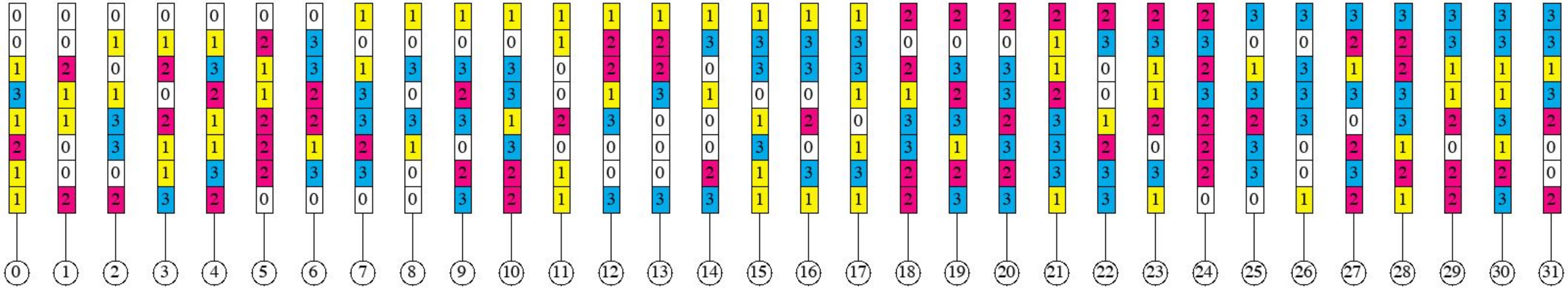


0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	3	2	1	3	3	0	2	3	2	2	0	1	3	1	1	2	1	3	2	0	1	1	1	1	1	0	0	2	0	3	1
3	3	0	2	0	2	0	3	0	0	3	0	3	3	3	3	1	1	3	0	2	2	0	3	0	0	1	1	3	1	2	0
3	1	3	2	3	1	1	2	1	3	0	2	0	1	3	3	1	0	1	2	1	2	3	3	3	1	0	2	1	3	2	3
2	1	3	1	3	3	3	3	3	2	0	1	1	3	1	0	2	0	1	1	1	3	0	0	3	3	1	0	1	2	3	2
2	3	2	3	3	0	1	2	2	3	1	1	0	2	0	2	3	2	2	3	2	0	3	1	1	3	3	2	2	1	3	3
1	1	3	0	0	2	2	2	3	1	2	0	0	0	1	0	3	0	0	3	2	0	1	3	3	2	3	1	0	1	1	0
3	2	2	0	0	3	1	2	3	2	3	0	2	0	3	3	3	1	2	2	2	0	0	1	2	3	0	1	3	3	2	2
0	3	3	3	1	2	1	0	0	3	3	2	3	2	1	1	1	1	2	2	0	3	0	1	2	0	2	3	1	2	1	3

keys (addresses, or storage locations)

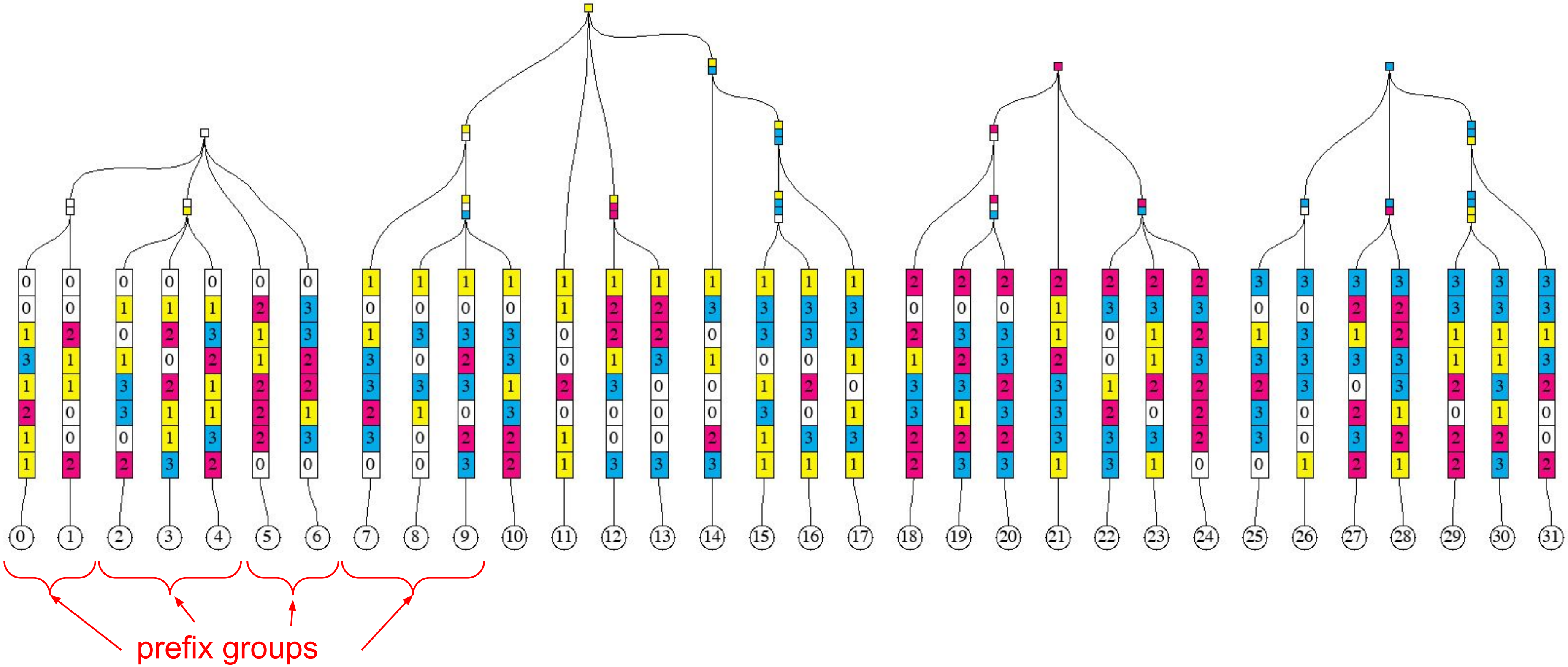
values (account tuples, or storage values)

How state root is constructed (sorted keys)

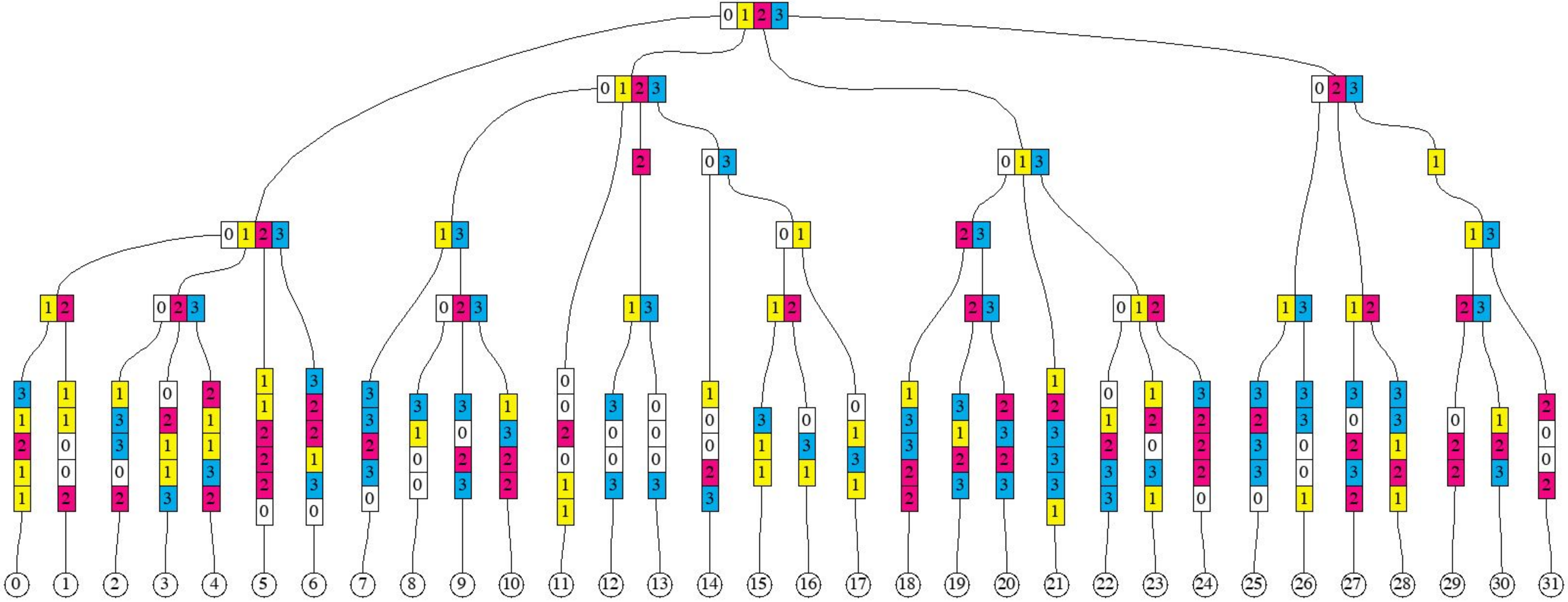


0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	3	3	3	3	3	3	3
0	0	1	1	1	2	3	0	0	0	0	1	2	2	3	3	3	3	0	0	0	1	3	3	3	0	0	2	2	3	3	3
1	2	0	2	3	1	3	1	3	3	3	0	2	2	0	3	3	3	2	3	3	1	0	1	2	1	3	1	2	1	1	1
3	1	1	0	2	1	2	3	0	2	3	0	1	3	1	0	0	1	1	2	3	2	0	1	3	3	3	3	3	1	1	3
1	1	3	2	1	2	2	3	3	3	1	2	3	0	0	1	2	0	3	3	2	3	1	2	2	2	3	0	3	2	3	2
2	0	3	1	1	2	1	2	1	0	3	0	0	0	0	3	0	1	3	1	3	3	2	0	2	3	0	2	1	0	1	0
1	0	0	1	3	2	3	3	0	2	2	1	0	0	2	1	3	3	2	2	2	3	3	3	2	3	0	3	2	2	2	0
1	2	2	3	2	0	0	0	0	3	2	1	3	3	3	1	1	1	2	3	3	1	3	1	0	0	1	2	1	2	3	2
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

How state root is constructed (prefix groups)

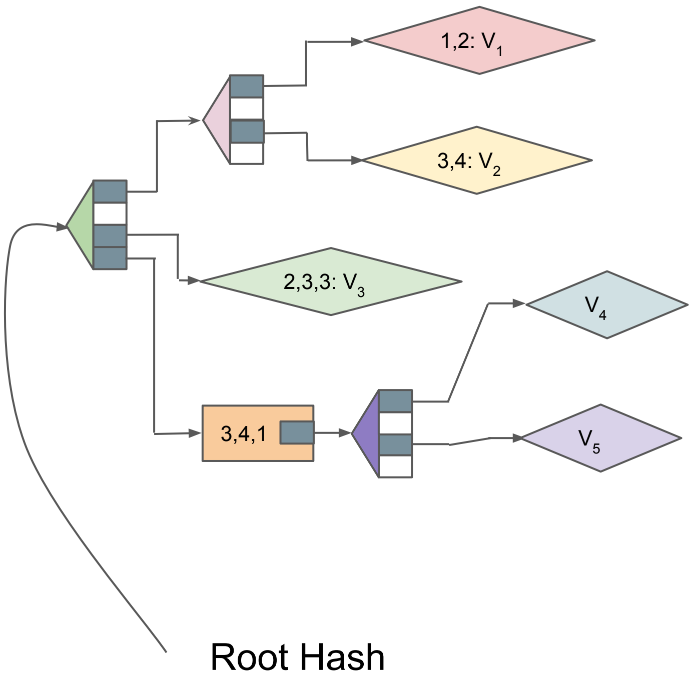


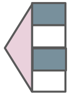
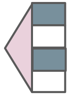
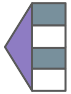
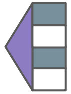
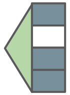
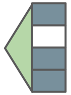
How state root is constructed (leaves and extensions)



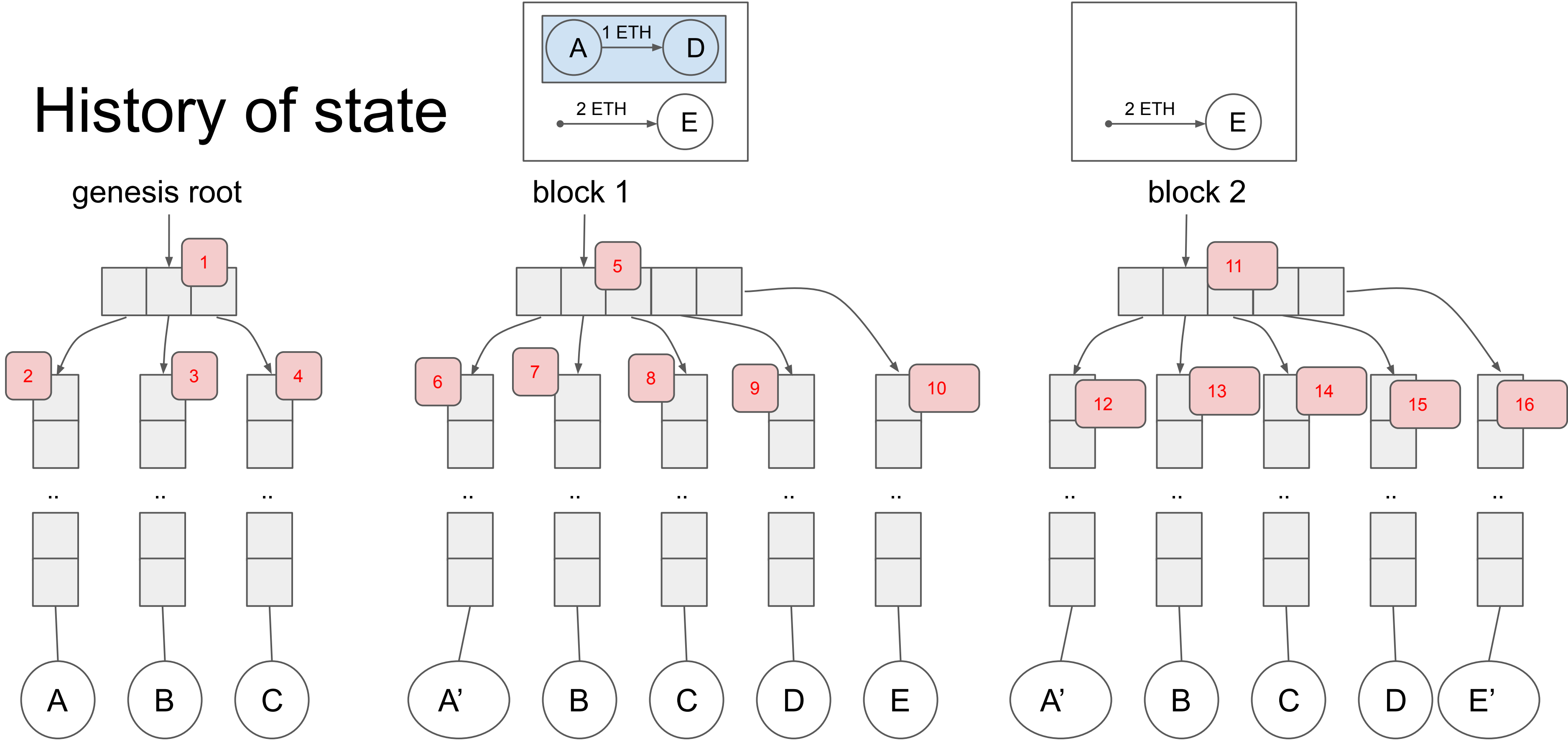
Persistence of Patricia tree in geth (and others)

Number of blocks with arrows on them == number of records in the DB



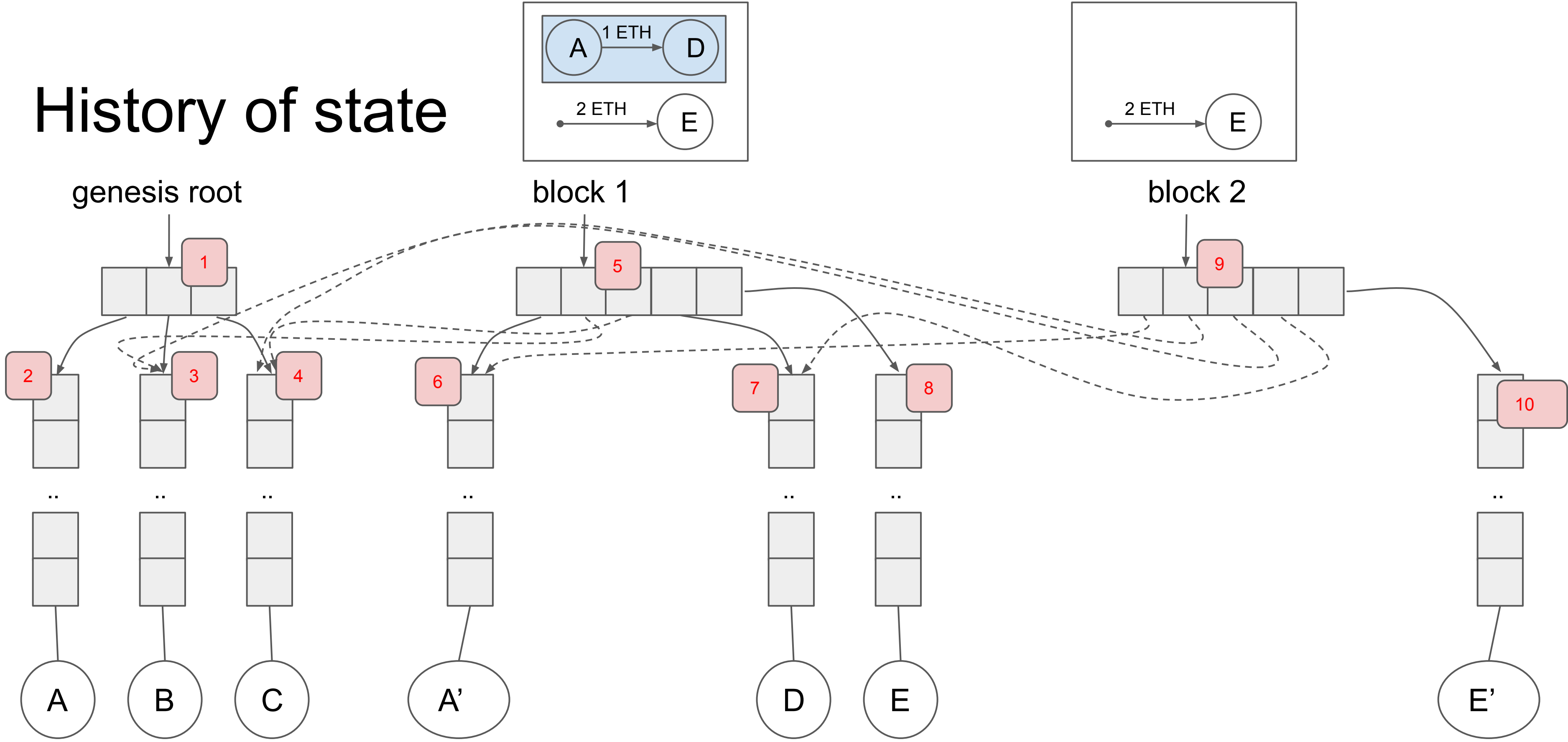
Key	Value
sha3(rlp(1,2: V_1))	rlp(1,2: V_1)
sha3(rlp(3,4: V_2))	rlp(3,4: V_2)
sha3(rlp())	rlp()
sha3(rlp(2,3,3: V_3))	rlp(2,3,3: V_3)
sha3(rlp(V_4))	rlp(V_4)
sha3(rlp(V_5))	rlp(V_5)
sha3(rlp())	rlp()
sha3(rlp(3,4,1))	rlp(3,4,1)
sha3(rlp())	rlp()

History of state



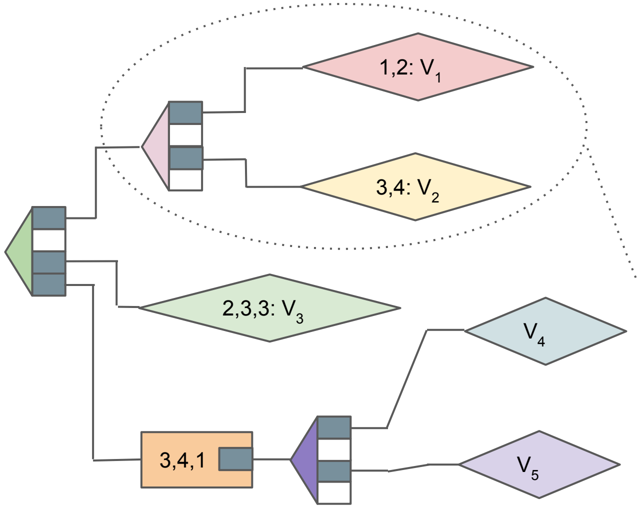
16 records in the DB?

History of state



10 records in the DB

Persistence ~~of Patricia tree~~ in turbo-geth

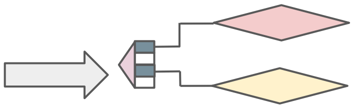


Key	Value
1,1,1,2	V_1
1,3,3,4	V_2
3,2,3,3	V_3
4,3,4,1,1	V_4
4,3,4,1,3	V_5

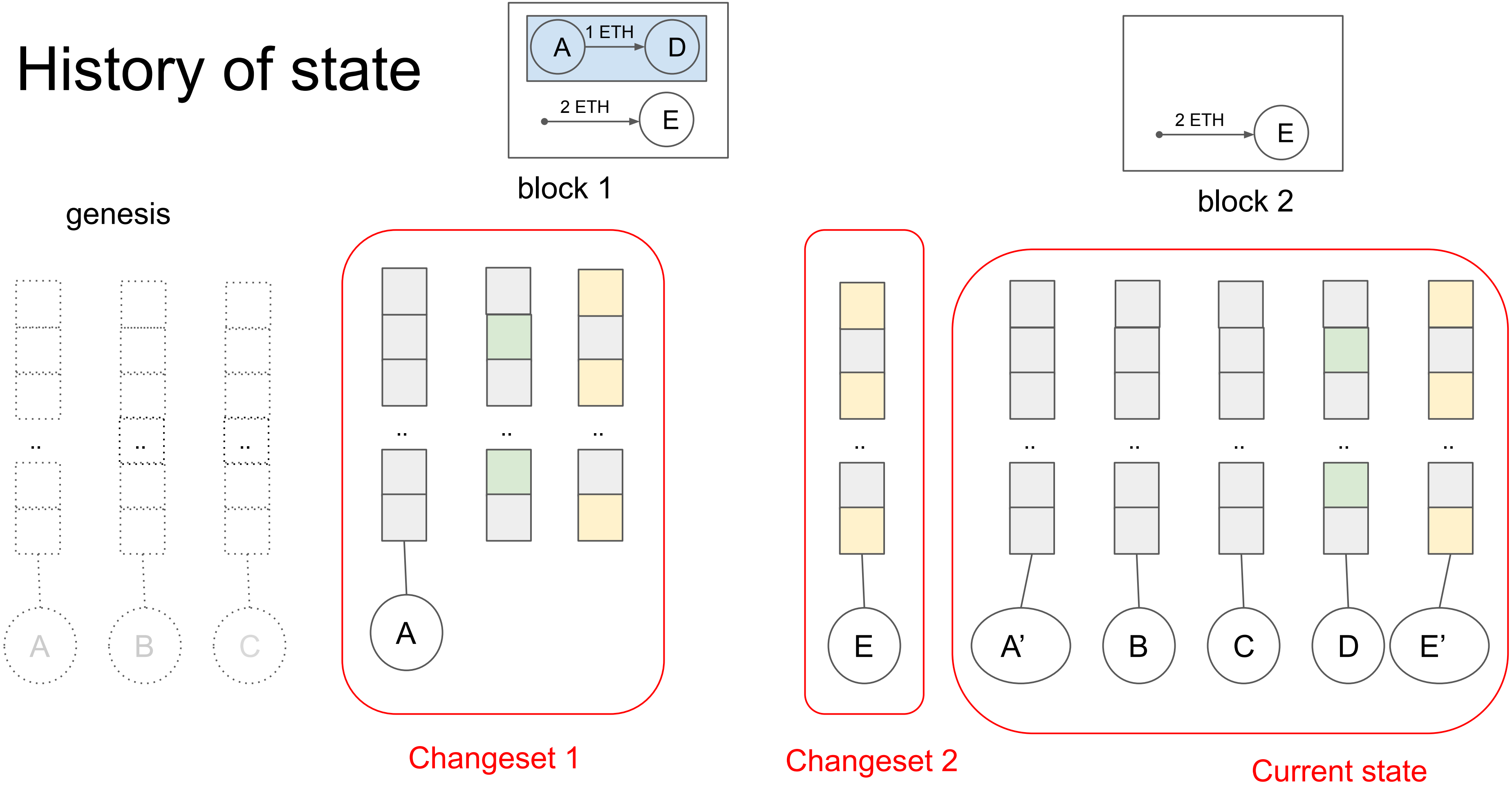
Goes here because it is sorted

Range query: 1,*,*,*

1,1,1,2	V_1
1,3,3,4	V_2



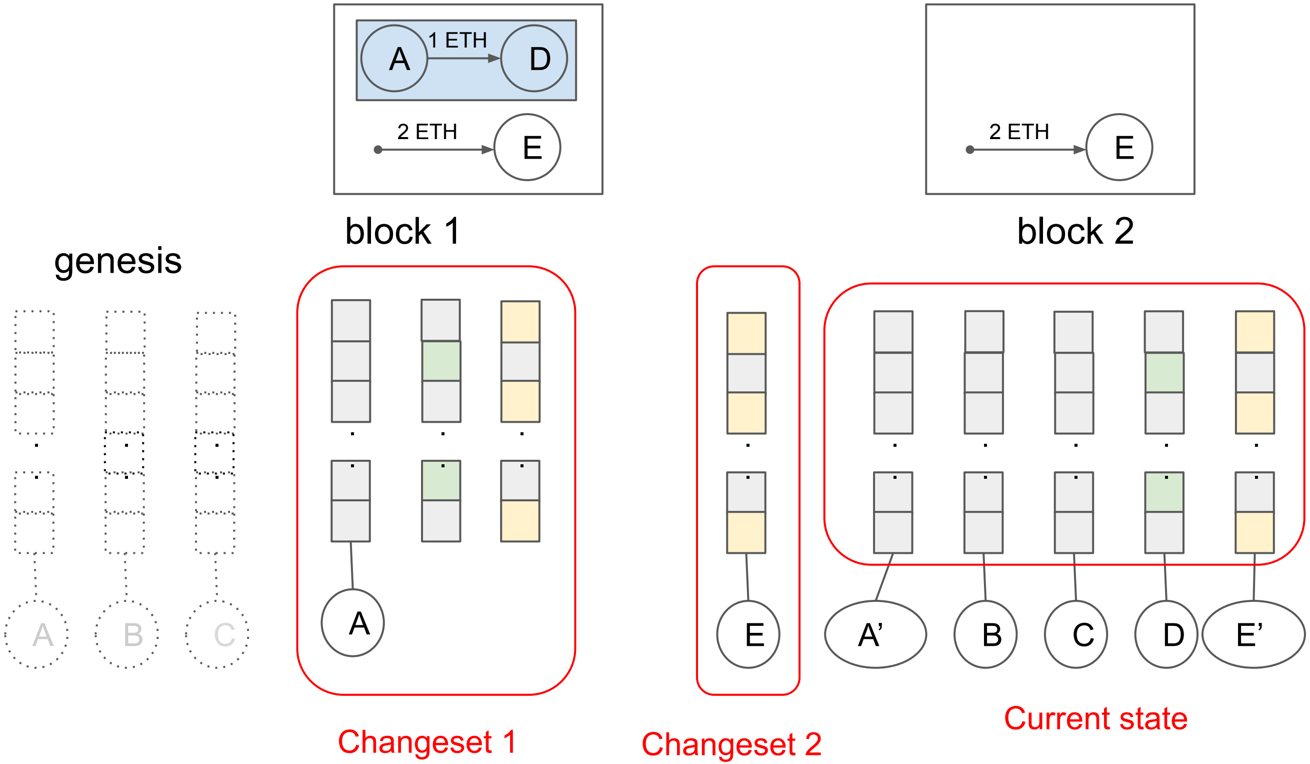
History of state



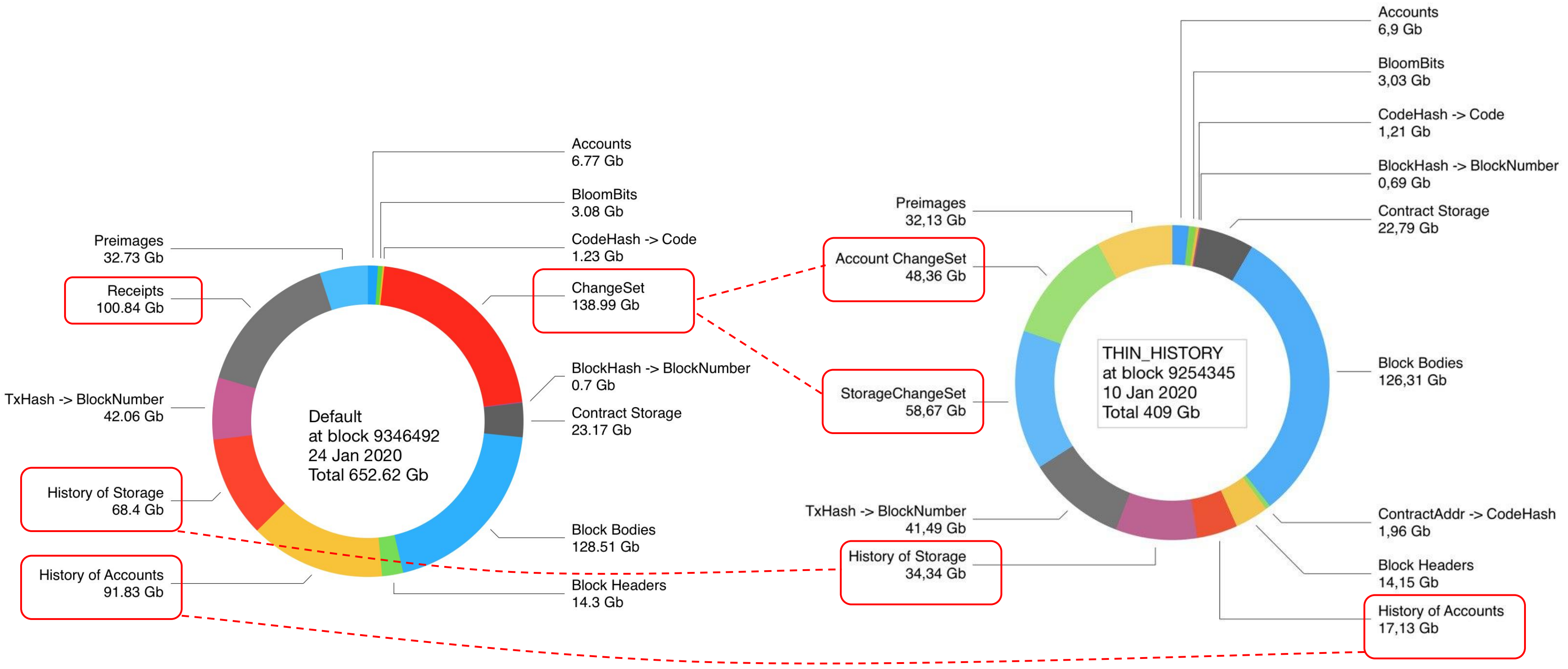
History index (a.k.a. THIN_HISTORY)

index

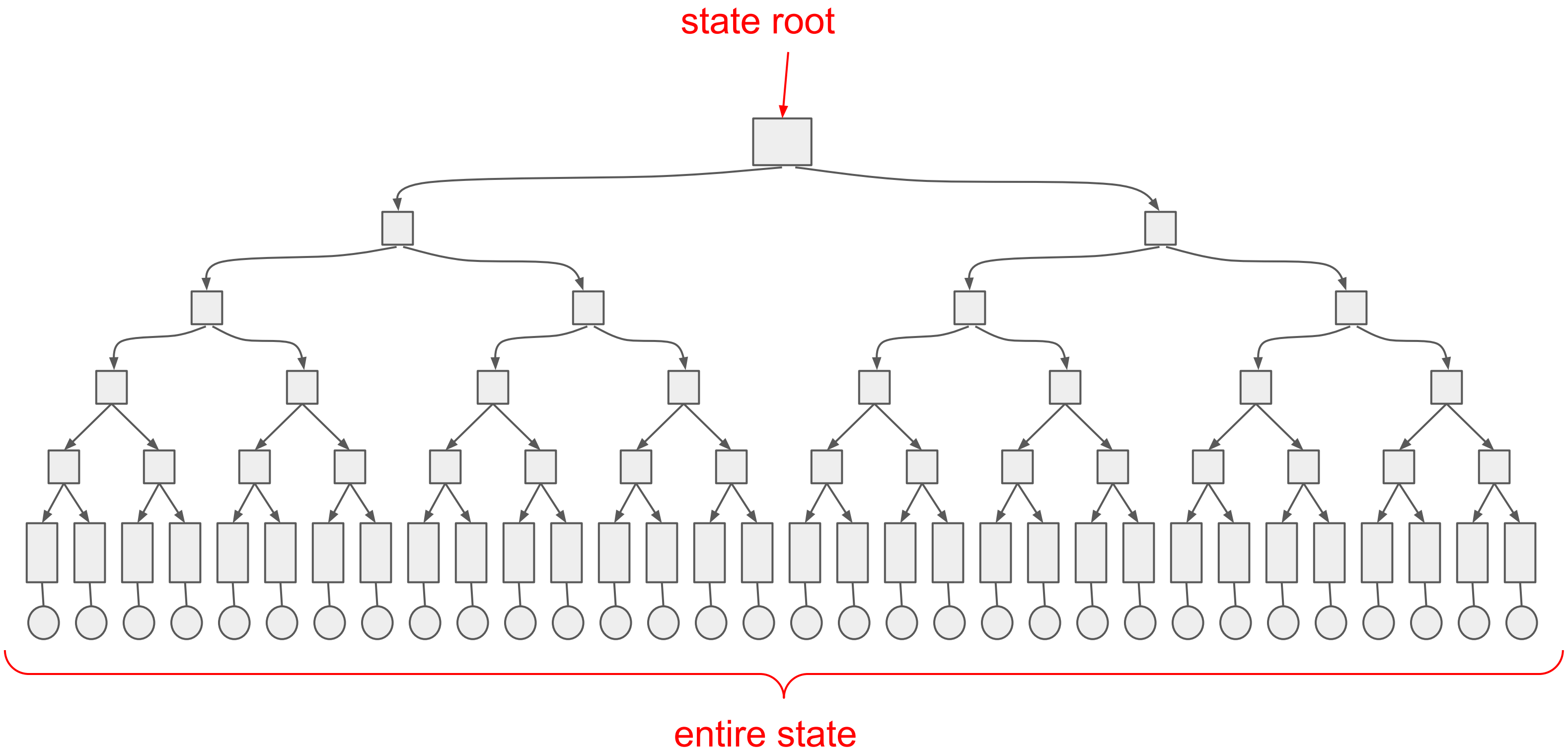
A: 1
D: 1
E: 1, 2



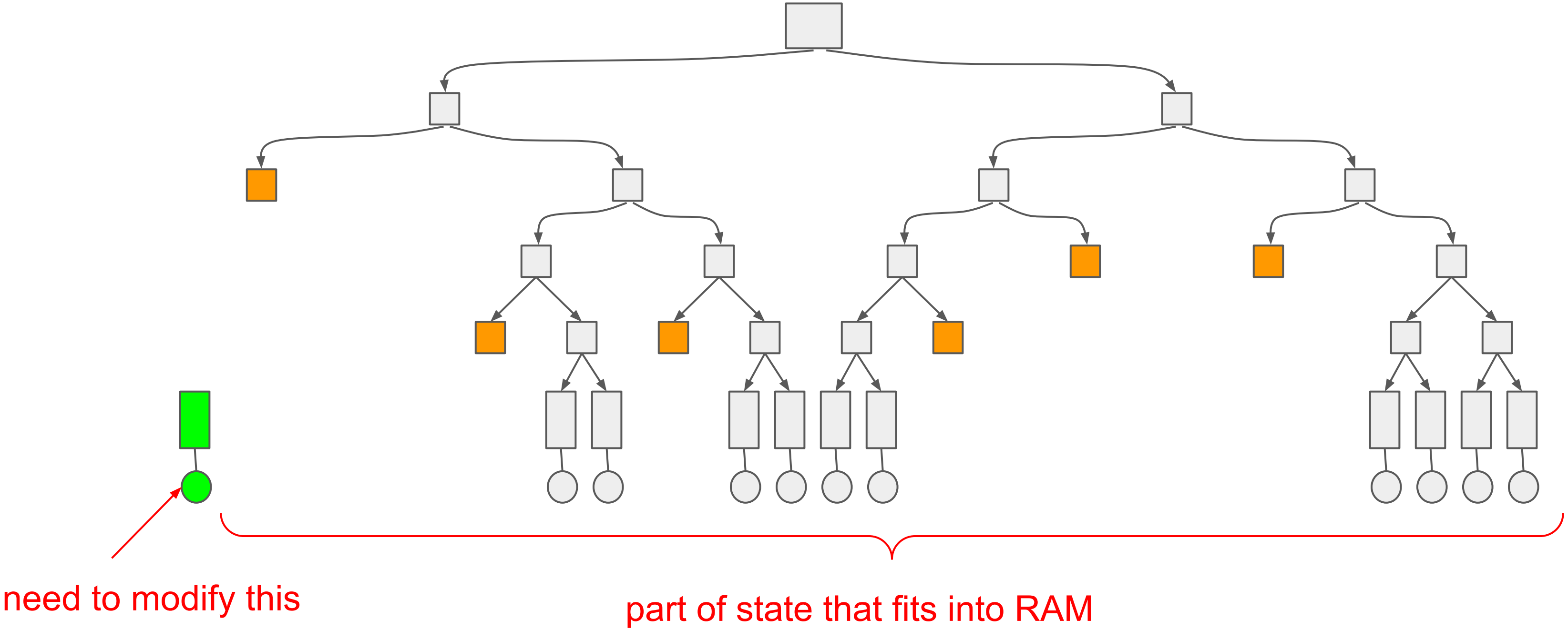
THIN_HISTORY and data on database size



No intermediate hashes - problem



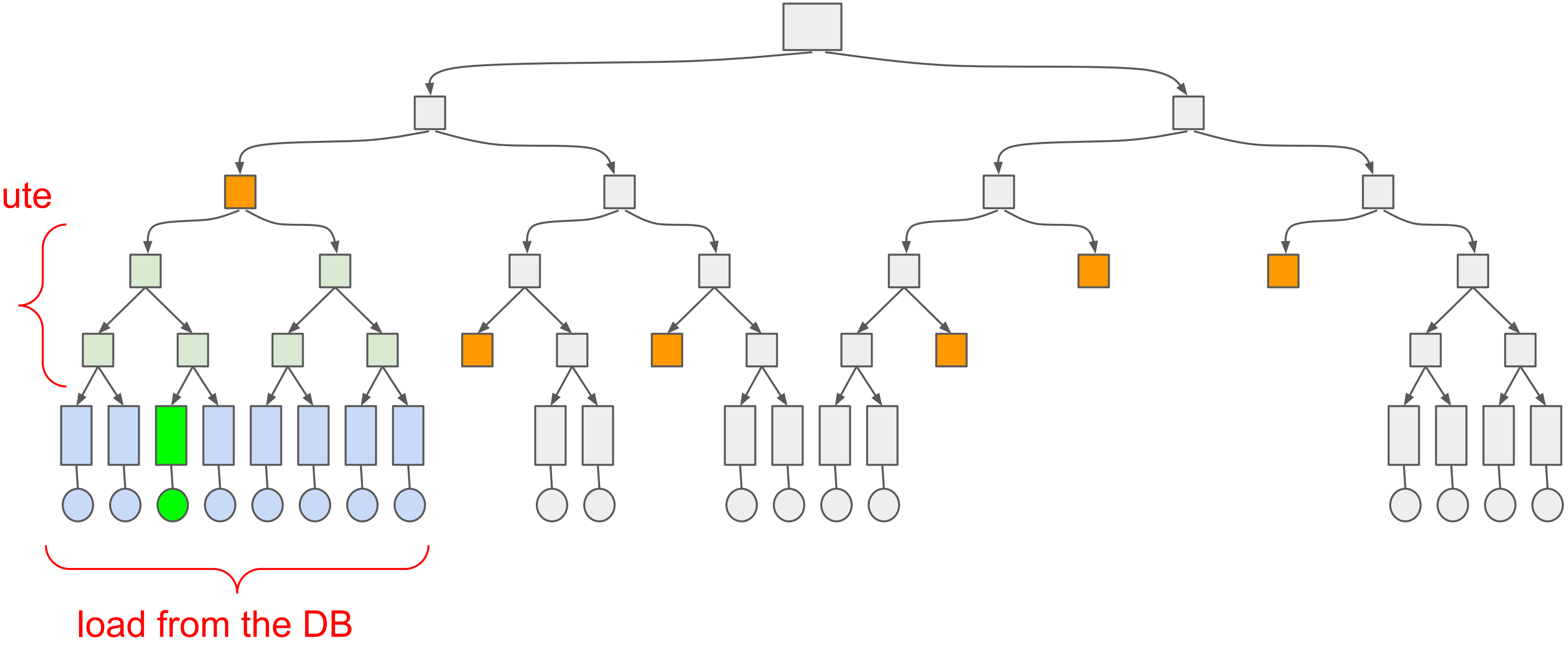
No intermediate hashes - problem



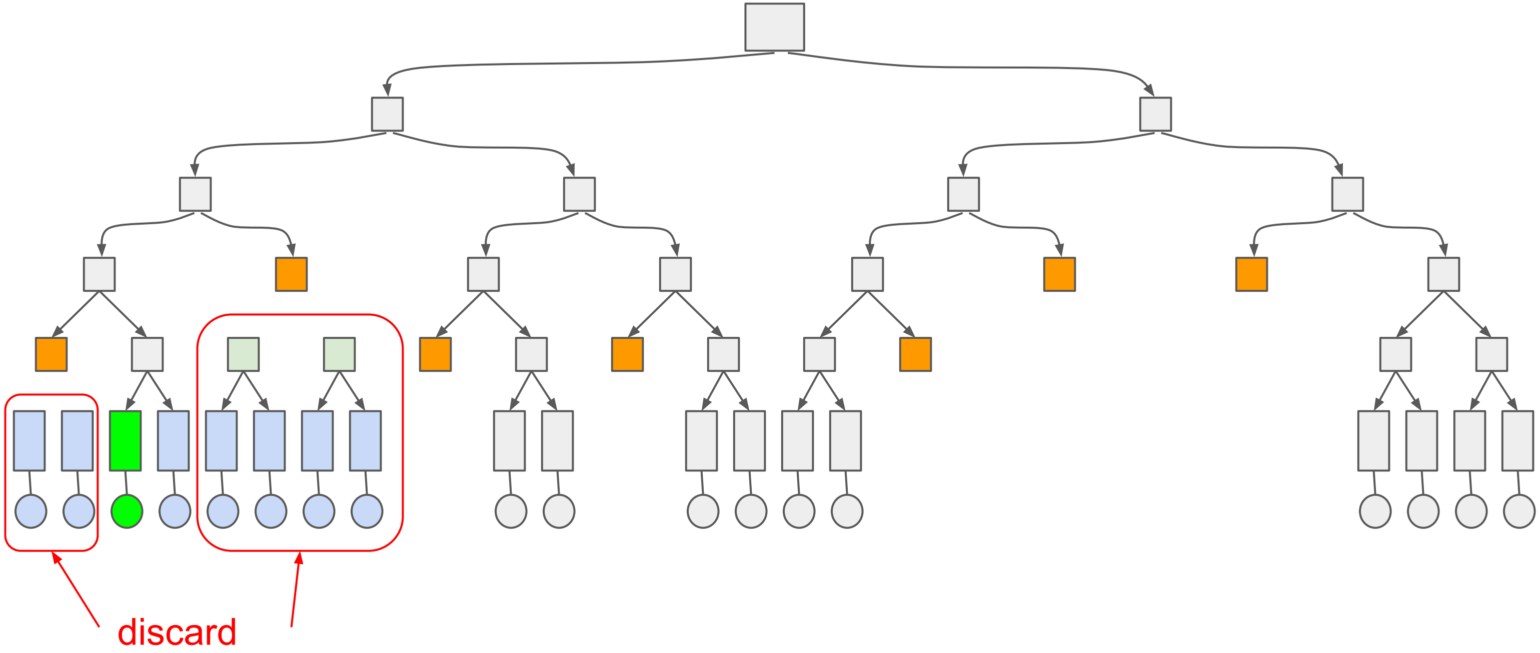
need to modify this

part of state that fits into RAM

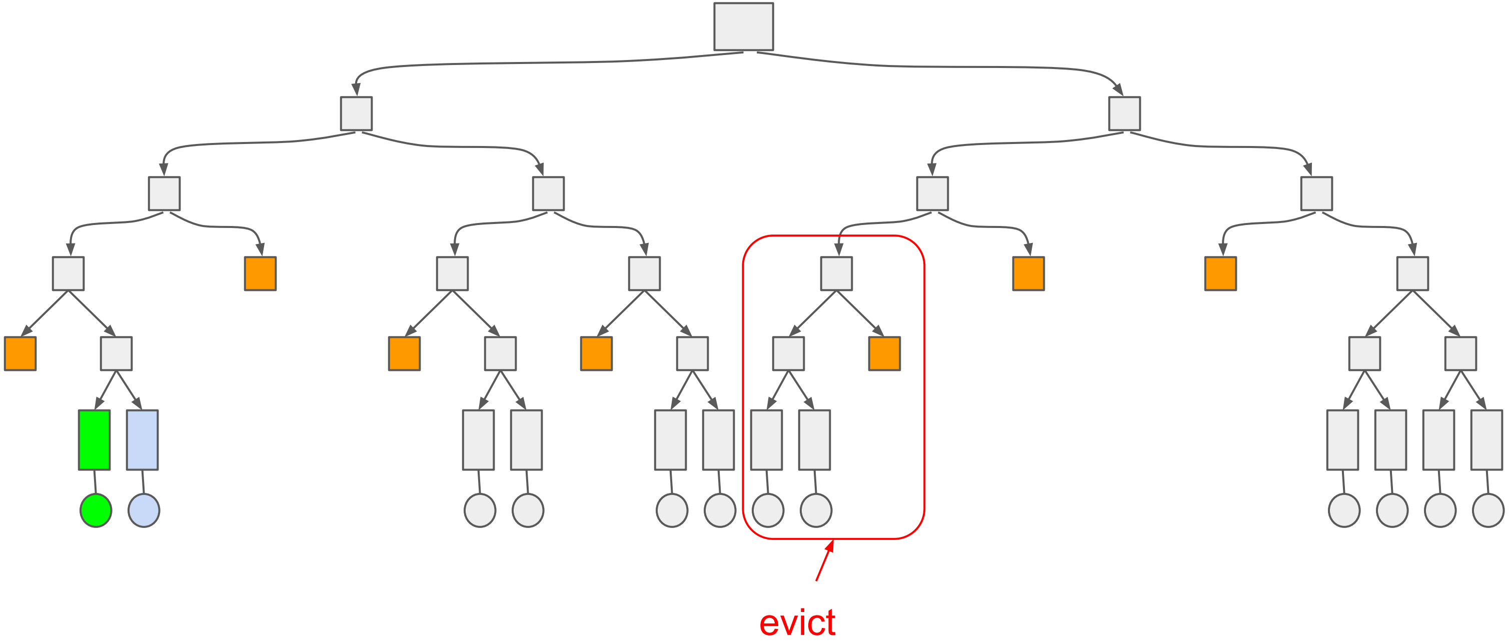
No intermediate hashes - problem



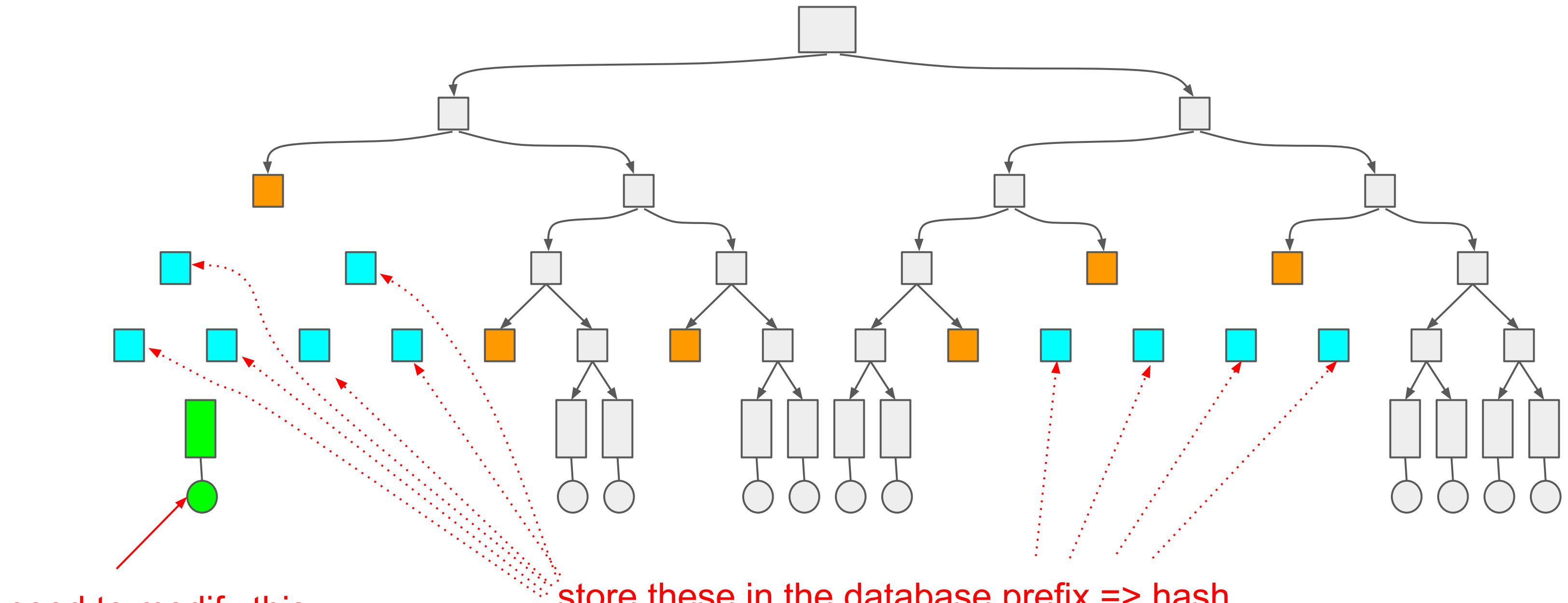
No intermediate hashes - problem



No intermediate hashes - problem



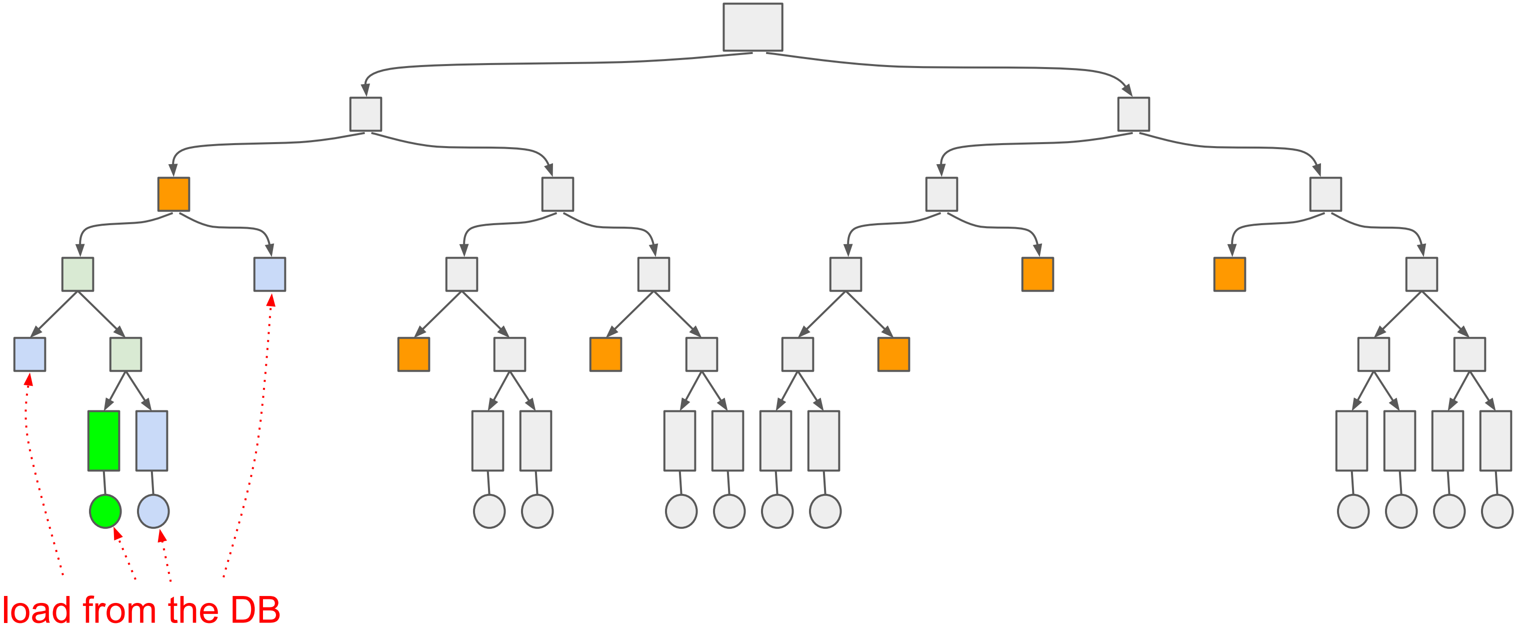
Intermediate hashes - solution



need to modify this

store these in the database prefix => hash

Intermediate hashes - solution



Nuances

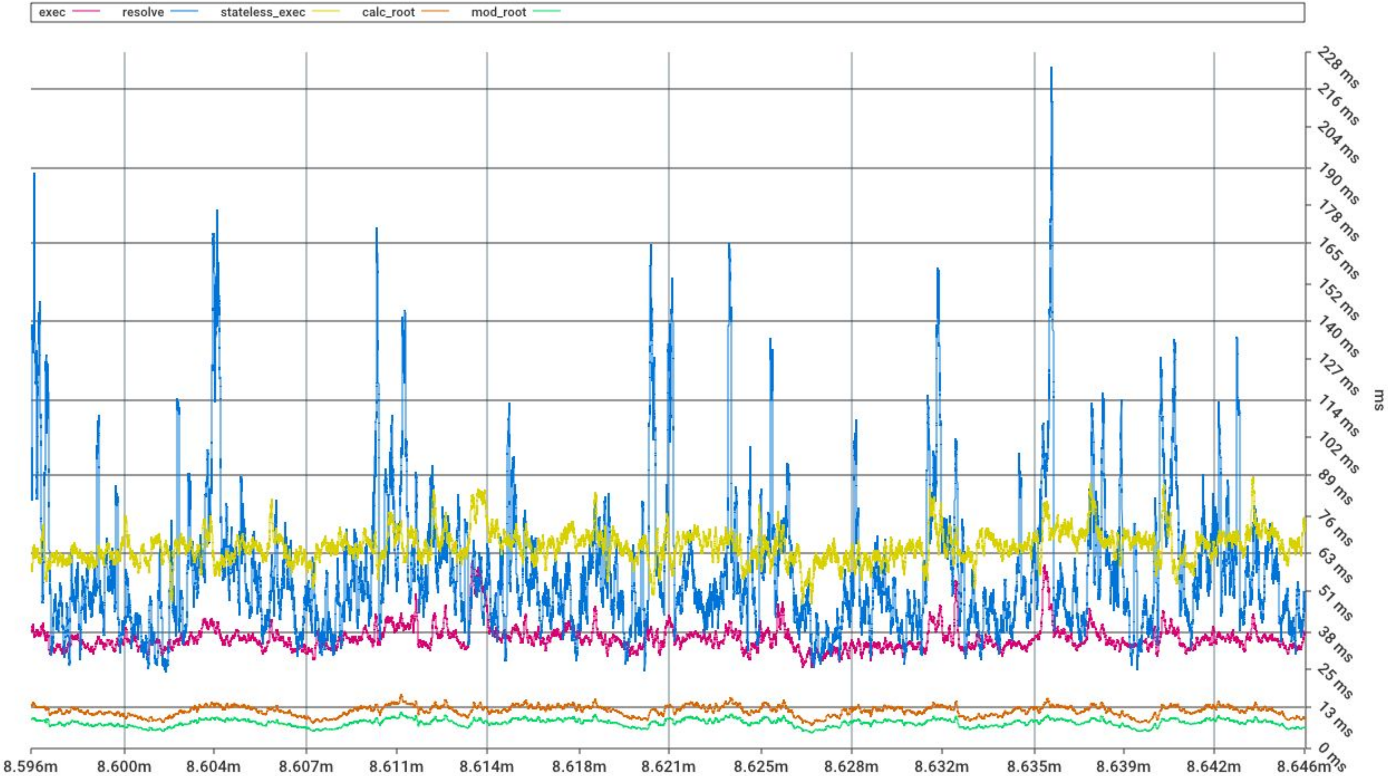
We are not keeping all intermediate hashes, but only those that have prefixes that are “whole” bytes (no odd nibbles)

We only keeping intermediate hashes of things that are NOT currently in memory

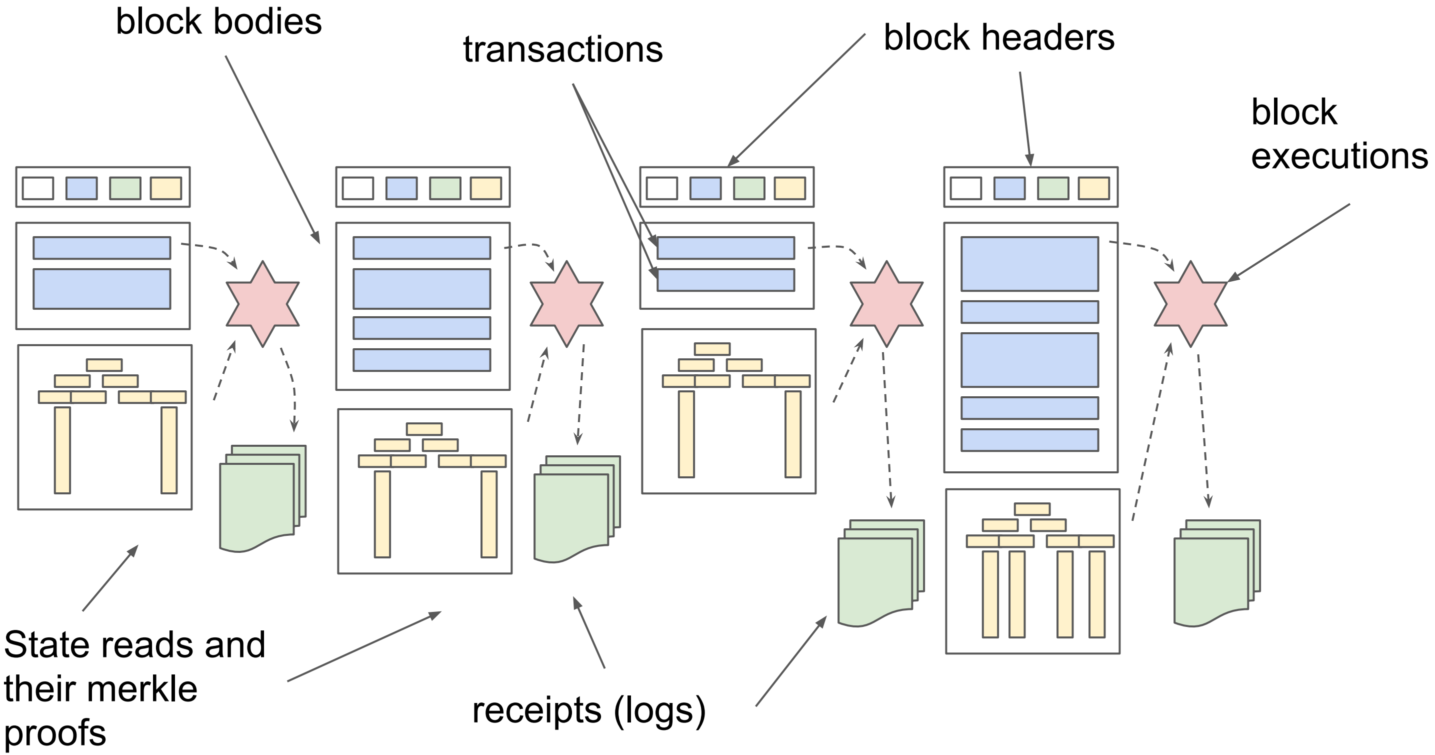
There are complication related to self-destructs and CREATE2 revival (honestly, CREATE2 is the most difficult opcode for Turbo-Geth in terms of impact)

Intermediate hashes still need to be stabilised 😅

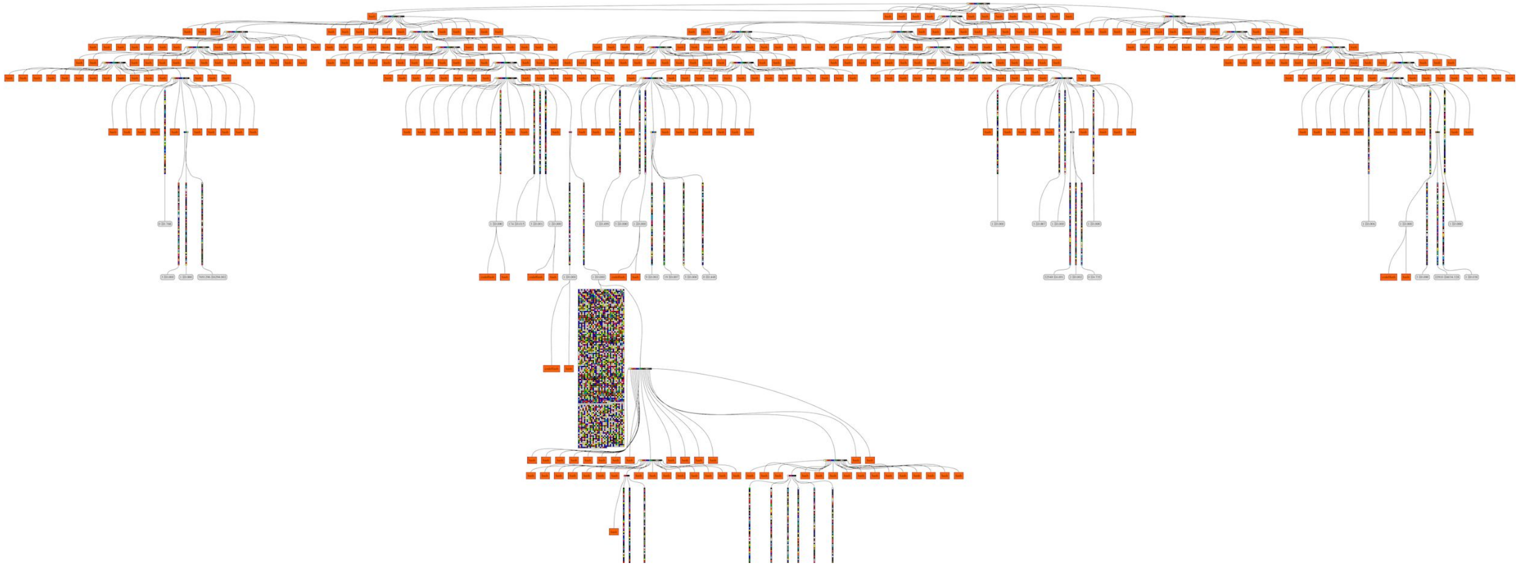
Intermediate hashes - baseline

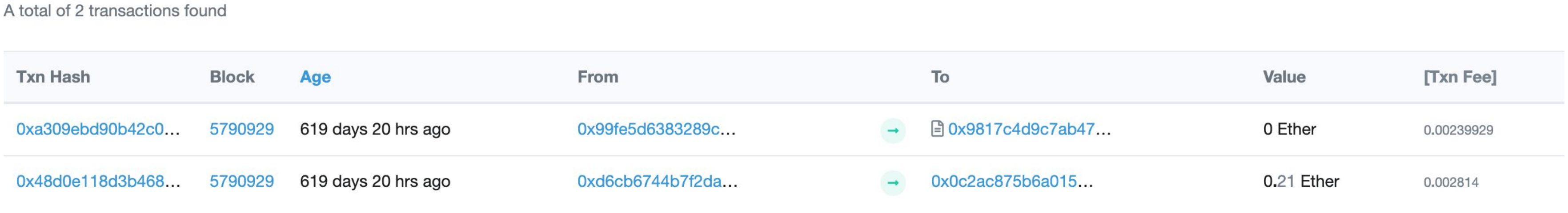


Stateless Ethereum



Block witness (block 57909279)

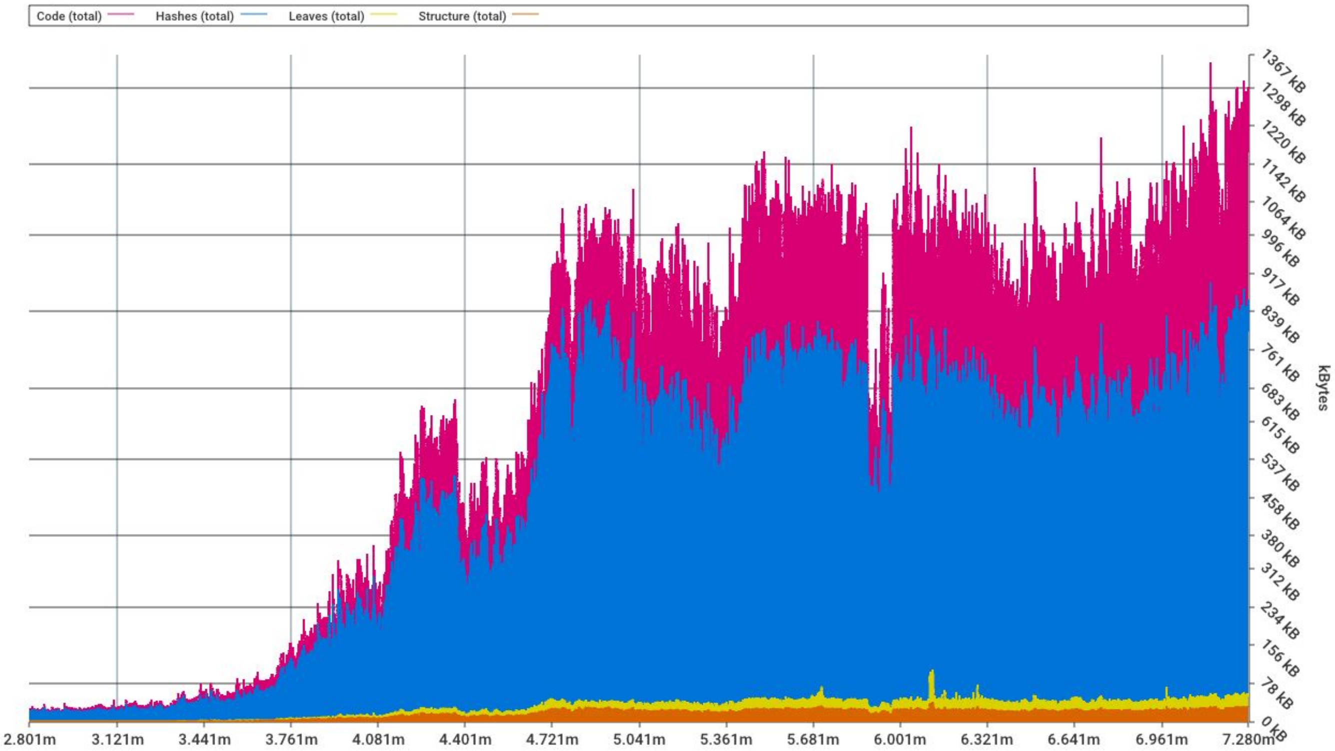




A total of 2 transactions found

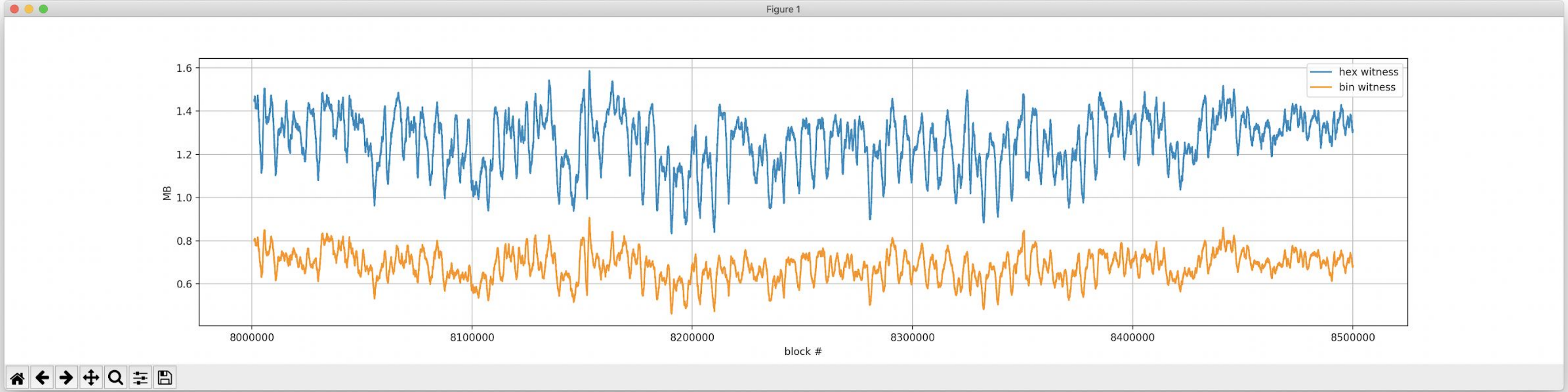
Txn Hash	Block	Age	From		To	Value	[Txn Fee]
0xa309ebd90b42c0…	5790929	619 days 20 hrs ago	0x99fe5d6383289c…	→	0x9817c4d9c7ab47…	0 Ether	0.00239929
0x48d0e118d3b468…	5790929	619 days 20 hrs ago	0xd6cb6744b7f2da…	→	0x0c2ac875b6a015…	0.21 Ether	0.002814

Block witness sizes



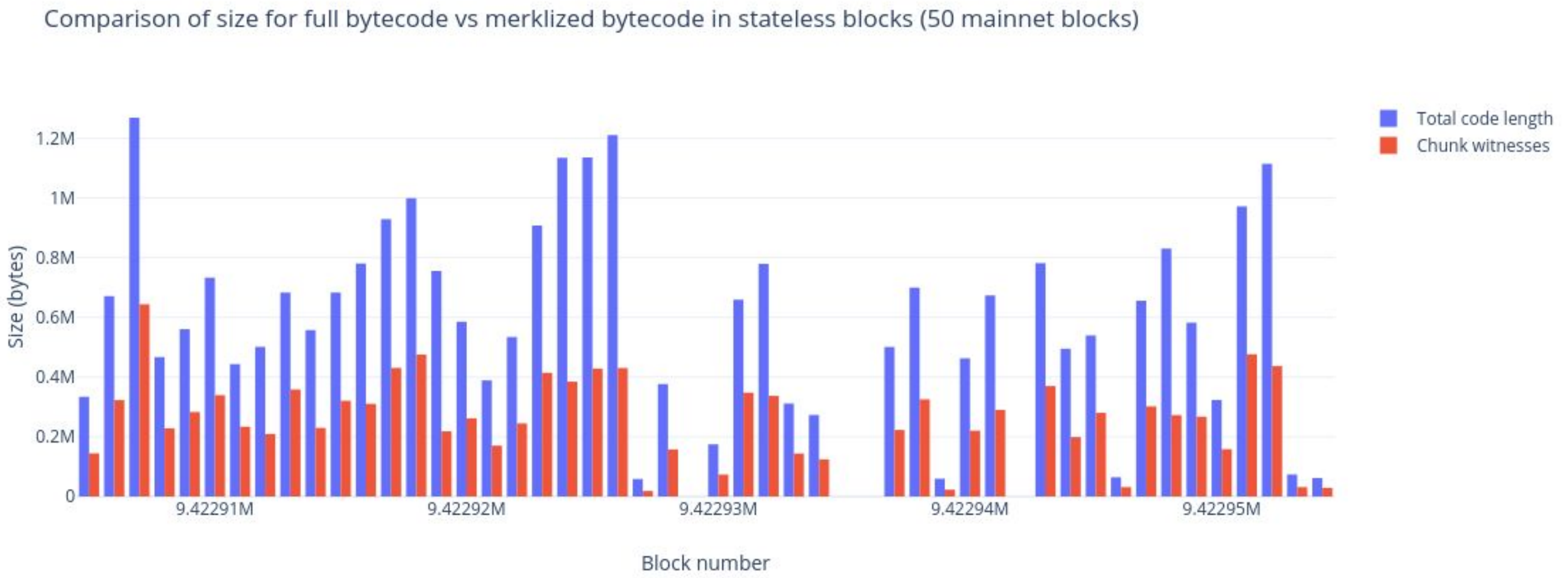
Reduction strategies

Hexary tries => binary trees



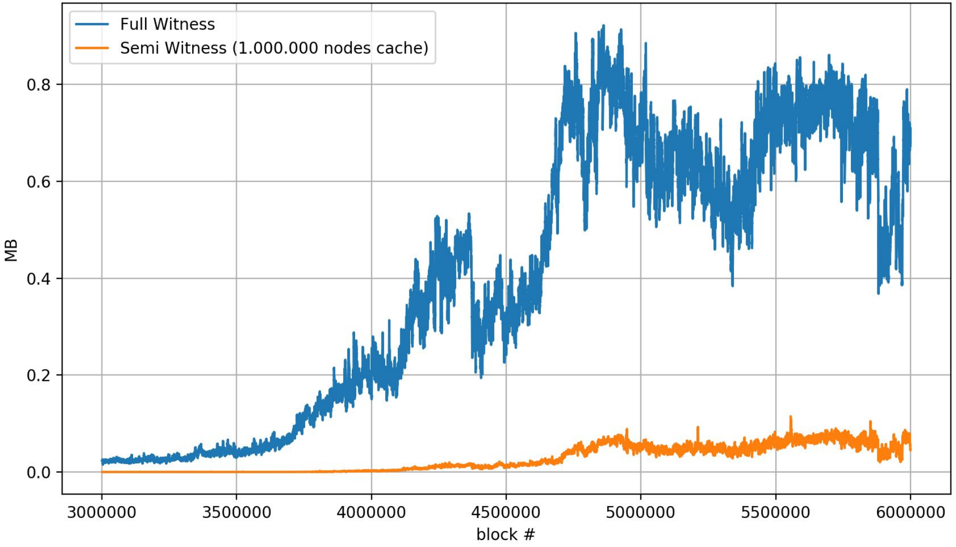
Reduction strategies

Code merkelisation (by Sina Mahmoodi)



Reduction strategies

Semi-statelessness (partial witnesses) - only as an average case optimisation



Reduction strategies

STARK/SNARK proofs to remove intermediate hashes

This is possible, but likely a lot of work. So we are not pursuing it currently.

Added challenges compared to the baseline approach:

- Need to have 2 proofs for block: pre-execution and post-execution
- Need to include special proof of non-existence for accounts/storage with attempted accessed but missing from the state
- Splitting witness into pieces (for more optimal relay) may not be possible