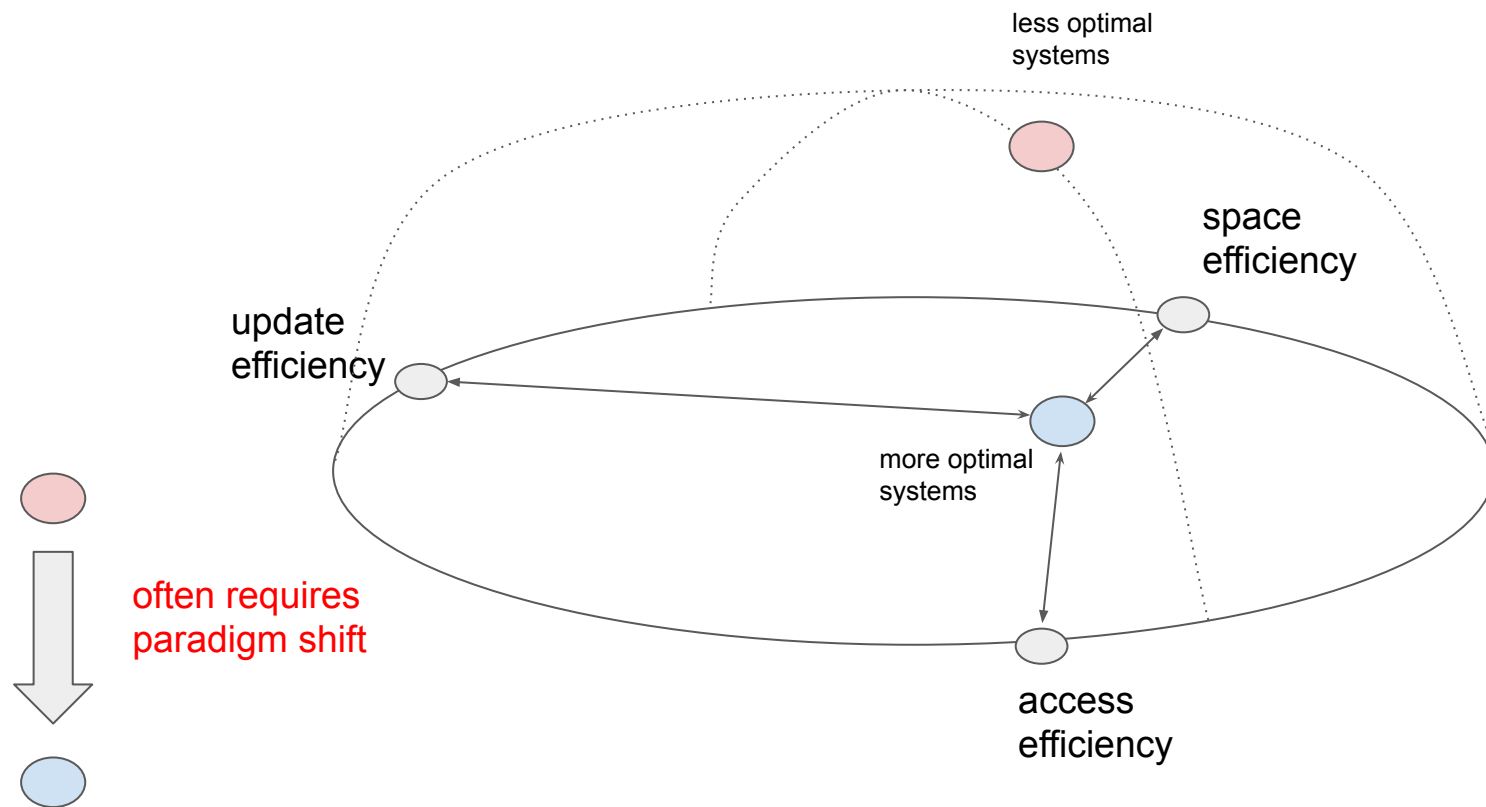


Turbo-Geth - too good to be
true?

Database tradeoffs and efficient frontier



Which paradigms did we shift (a bit)?

DATA MODEL (STATE) - flat vs trees

DATA MODEL (HISTORY) - flat vs trees, reverse diffs vs forward diffs, bitmap indices vs bloom filter-based indices (won't be covered today)

CONCURRENCY - homogeneous (or no concurrency) vs heterogeneous

ARCHITECTURE - modular with separate concerns vs monolith with cross-cutting concerns (done in the name of optimisation)

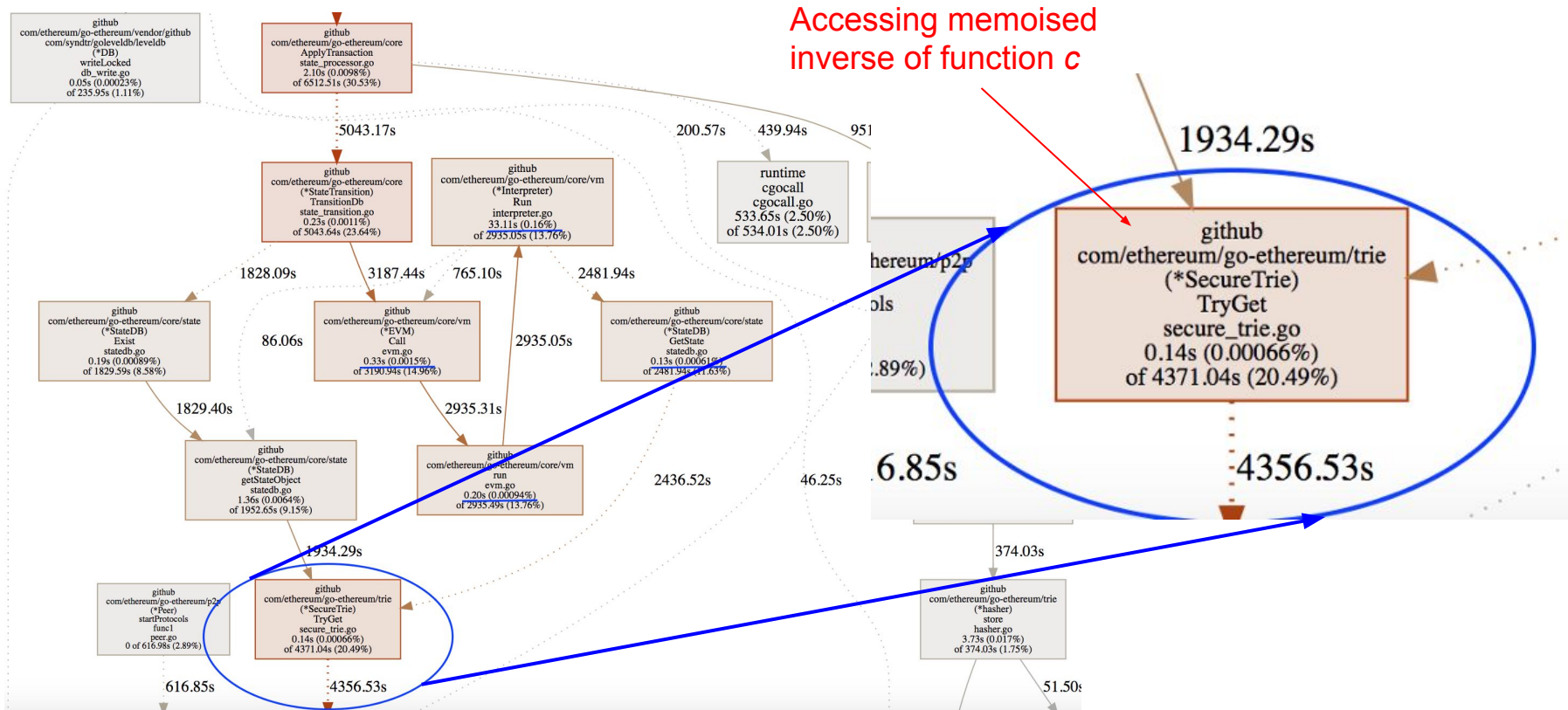
DATA MODEL - It starts with the yellow paper

Should probably be
“memoise the inverse of
function c ”

D.1. **Trie Database.** Thus no explicit assumptions are made concerning what data is stored and what is not, since that is an implementation-specific consideration; we simply define the identity function mapping the key-value set \mathcal{J} to a 32-byte hash and assert that only a single such hash exists for any \mathcal{J} , which though not strictly true is accurate within acceptable precision given the Keccak hash’s collision resistance. In reality, a sensible implementation will not fully recompute the trie root hash for each set.

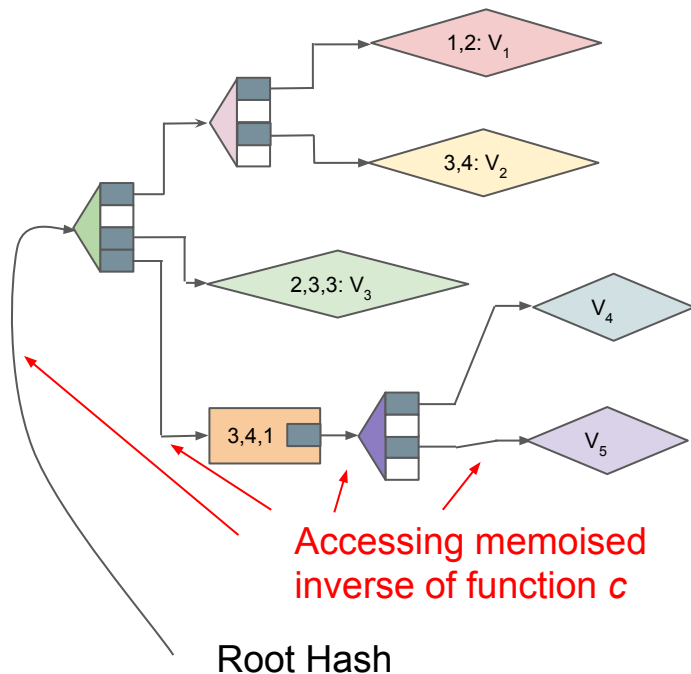
A reasonable implementation will maintain a database of nodes determined from the computation of various tries or, more formally, it will memoise the function c . This strategy uses the nature of the trie to both easily recall the contents of any previous key-value set and to store multiple such sets in a very efficient manner. Due to the dependency relationship, Merkle-proofs may be constructed with an $O(\log N)$ space requirement that can demonstrate a particular leaf must exist within a trie of a given root hash.

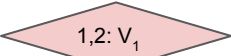
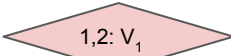
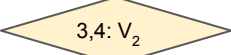
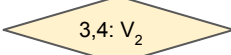


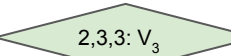
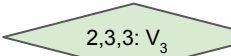






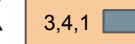
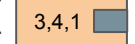


Profiling go-ethereum (5th of December 2017)



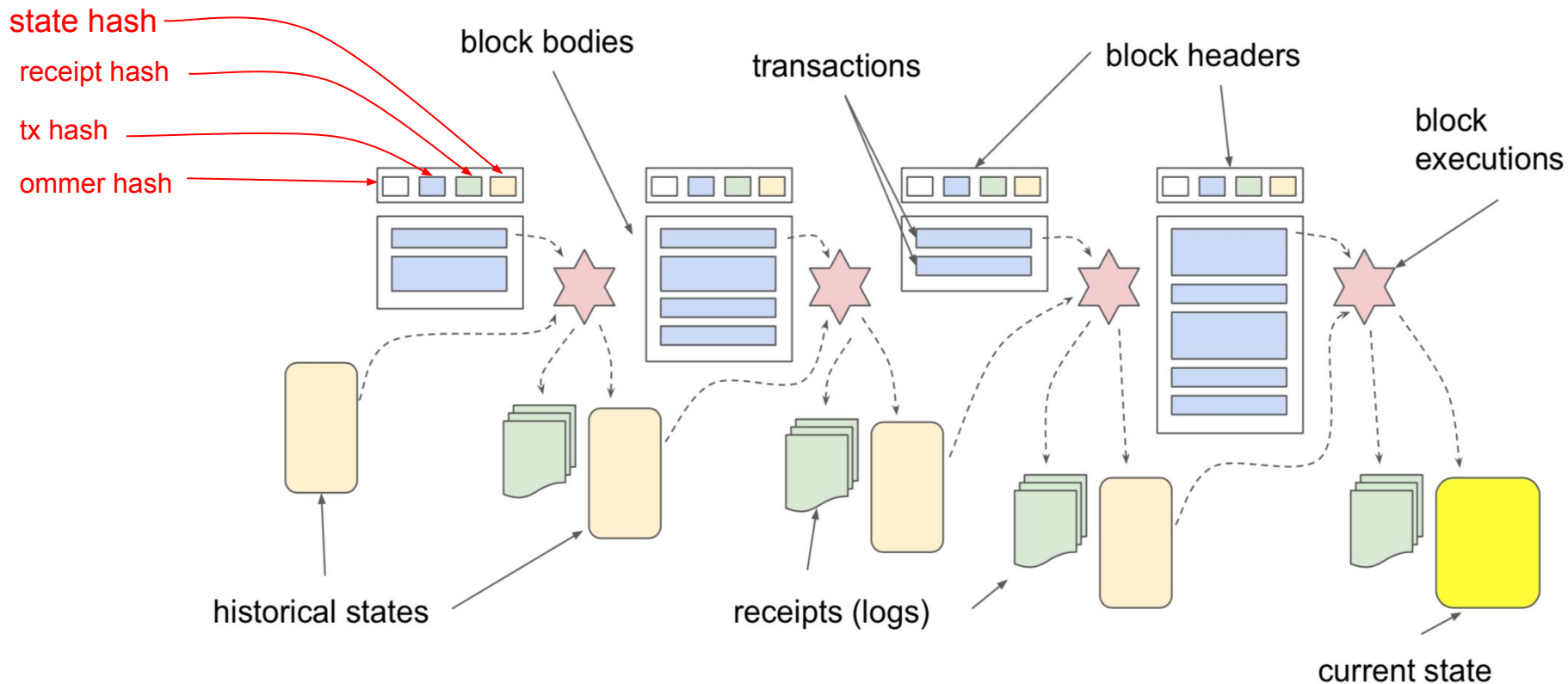
Persistence of Patricia tree in geth (and others)

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

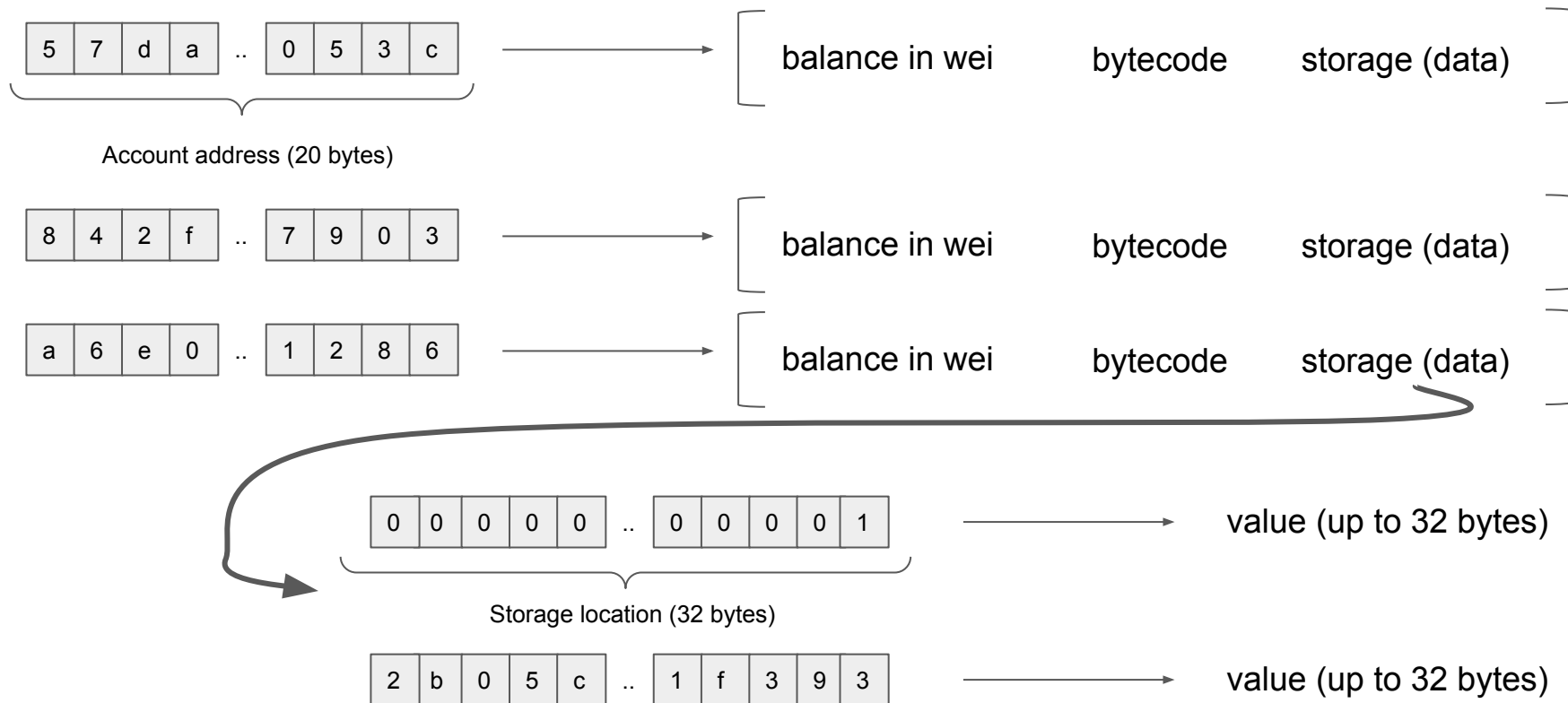


Key	Value
sha3(rlp())	rlp()
sha3(rlp())	rlp()
sha3(rlp())	rlp()
sha3(rlp())	rlp()
sha3(rlp())	rlp()
sha3(rlp())	rlp()
sha3(rlp())	rlp()
sha3(rlp())	rlp()
sha3(rlp())	rlp()

Why is trie the primary data structure?



Ethereum State from EVM's point of view - **no tries!**



Main data model hypothesis of Turbo-Geth

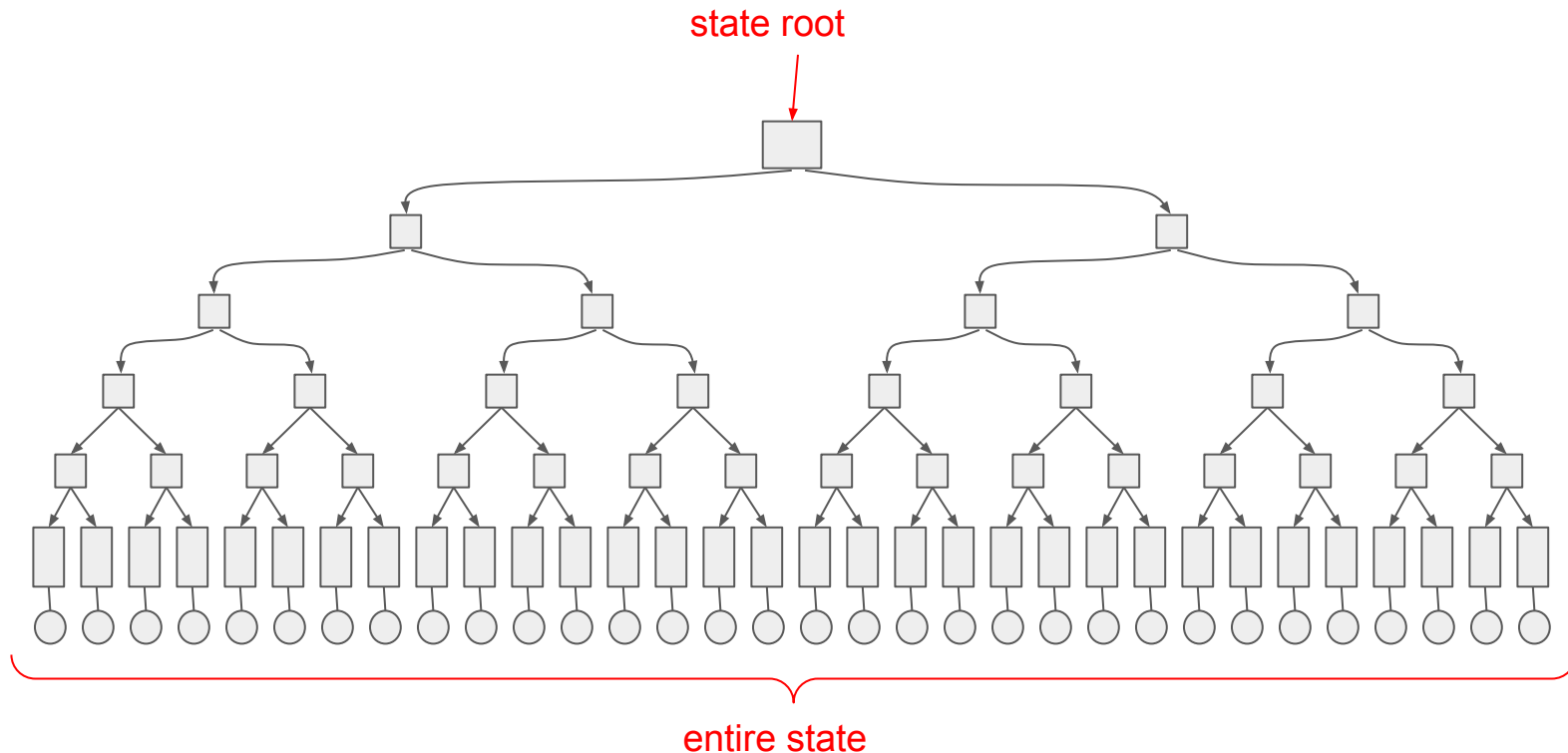
Making the radix tree (trie) the primary structure for storing the state is a “false trade-off”.

It is ostensibly optimised for quicker computation of state root for each block, by sacrificing the efficiency of EVM access to the state. Yellow paper calls it “sensible” or “reasonable” implementation.

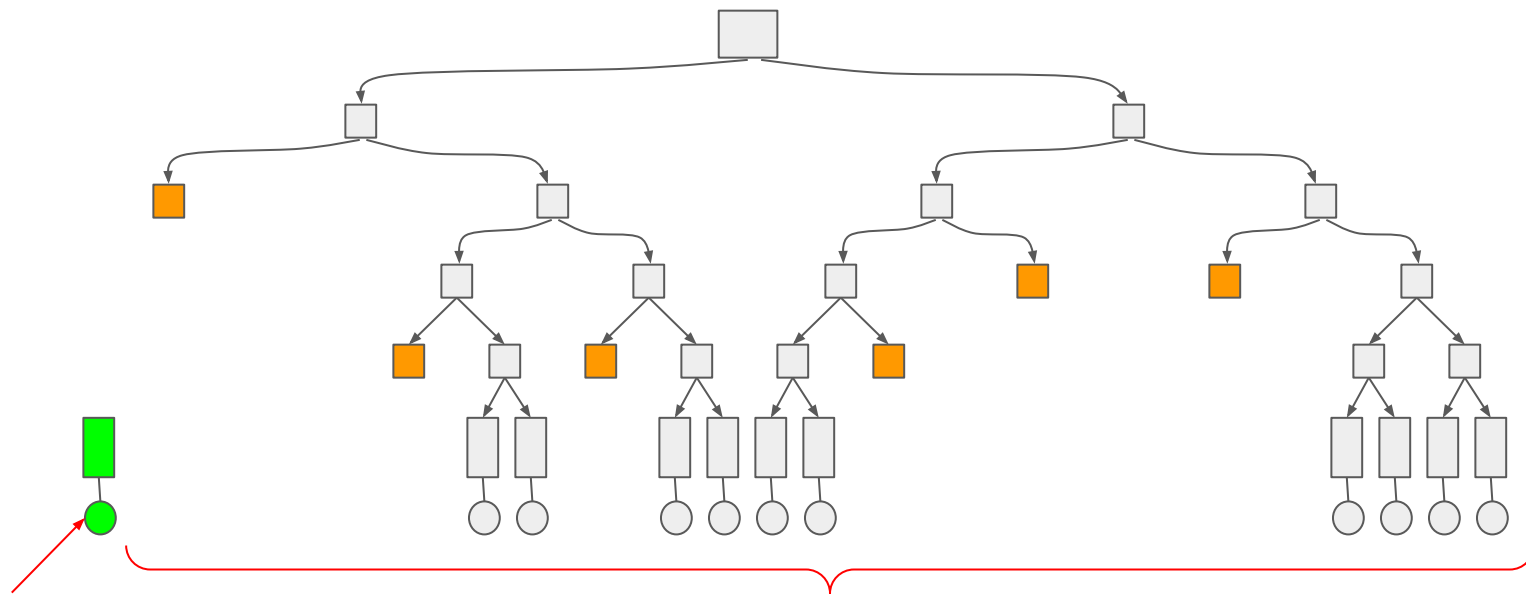
It might be sensible and reasonable, but turns out be not as efficient as it could be.

It is possible to compute state root efficiently AND have efficient EVM access to the state.

How to compute root hash from flat data?



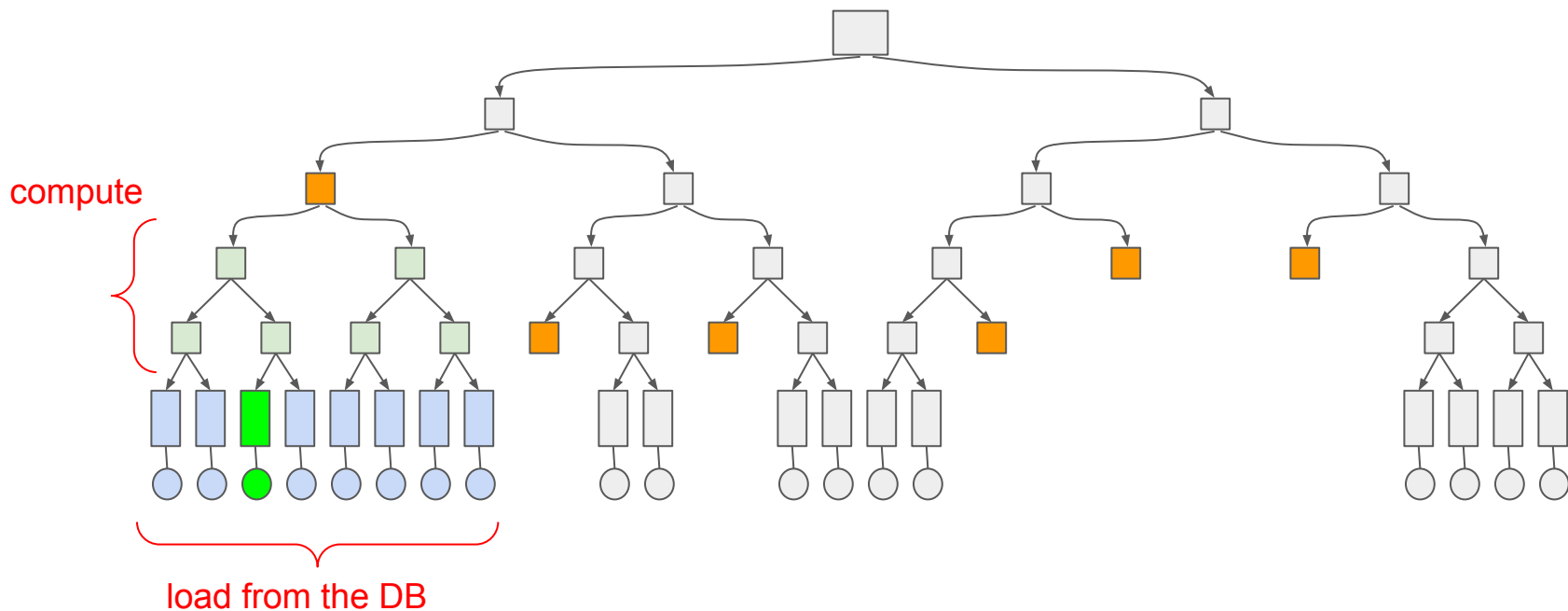
How to compute root hash from flat data?



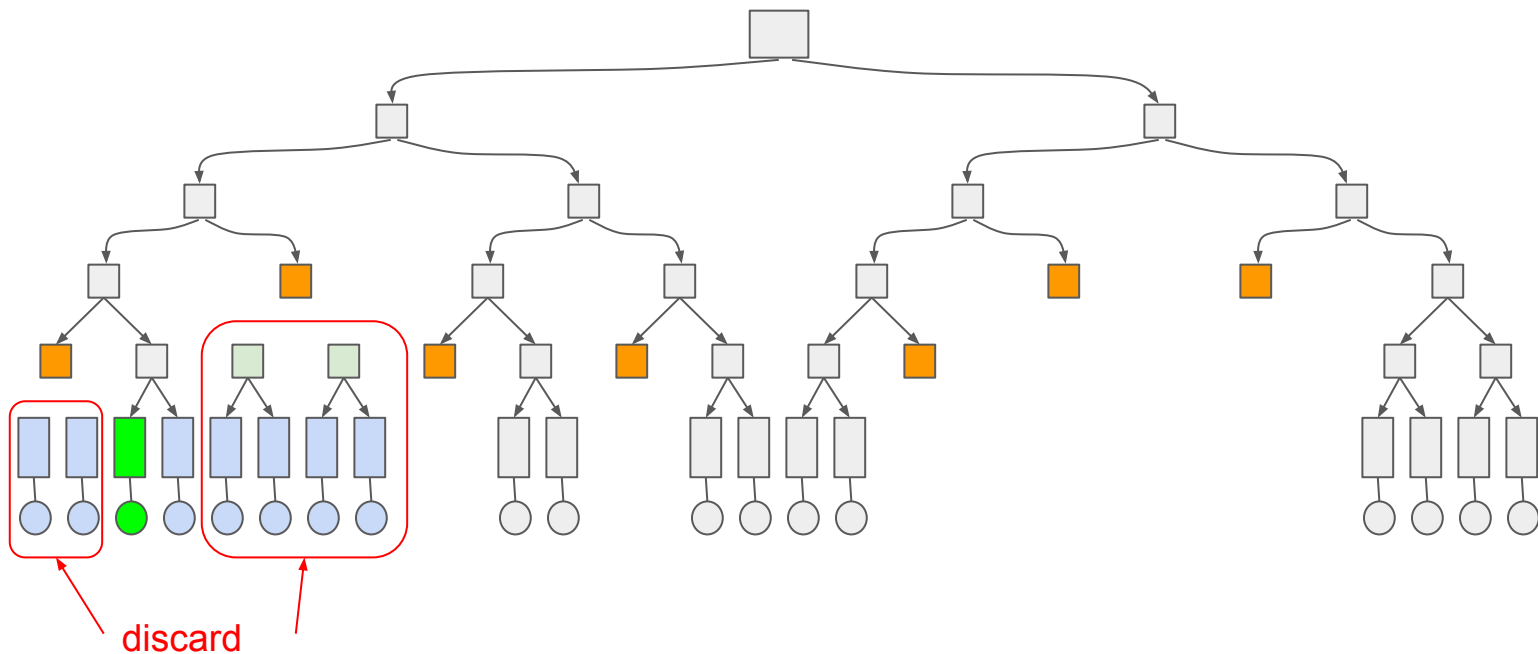
need to modify this

part of state that fits into RAM

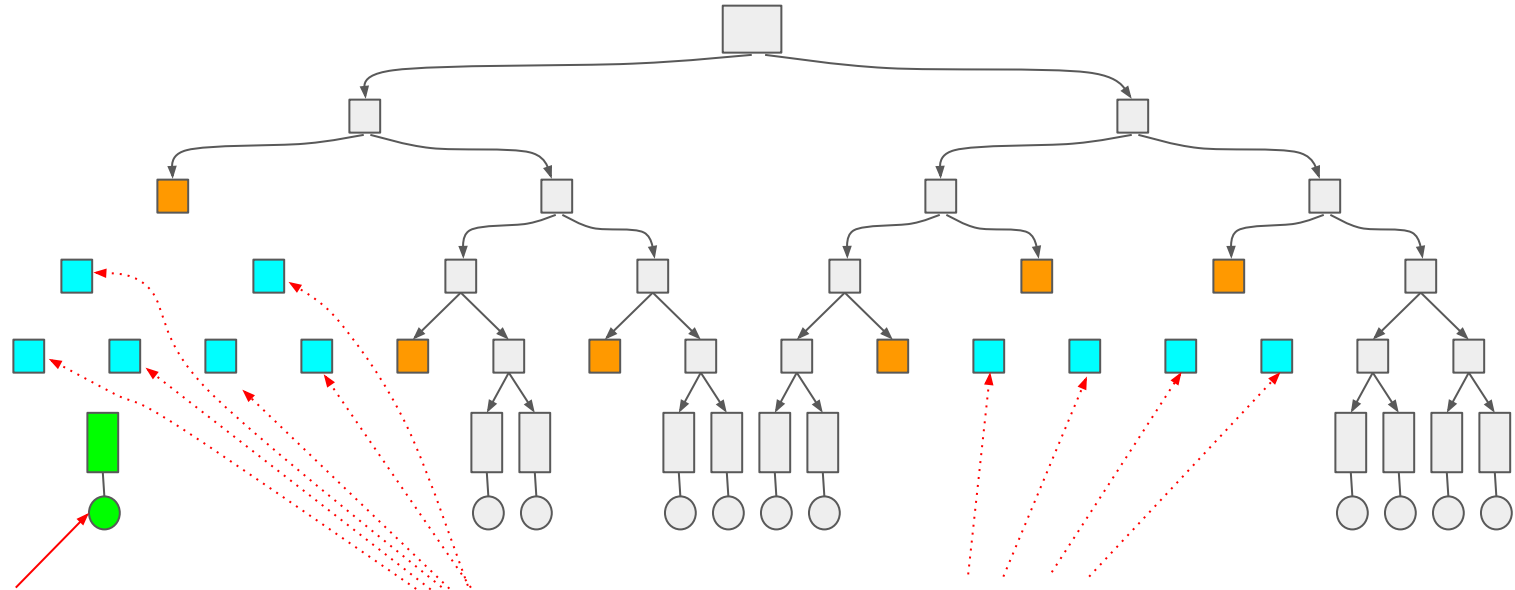
How to compute root hash from flat data?



How to compute root hash from flat data?



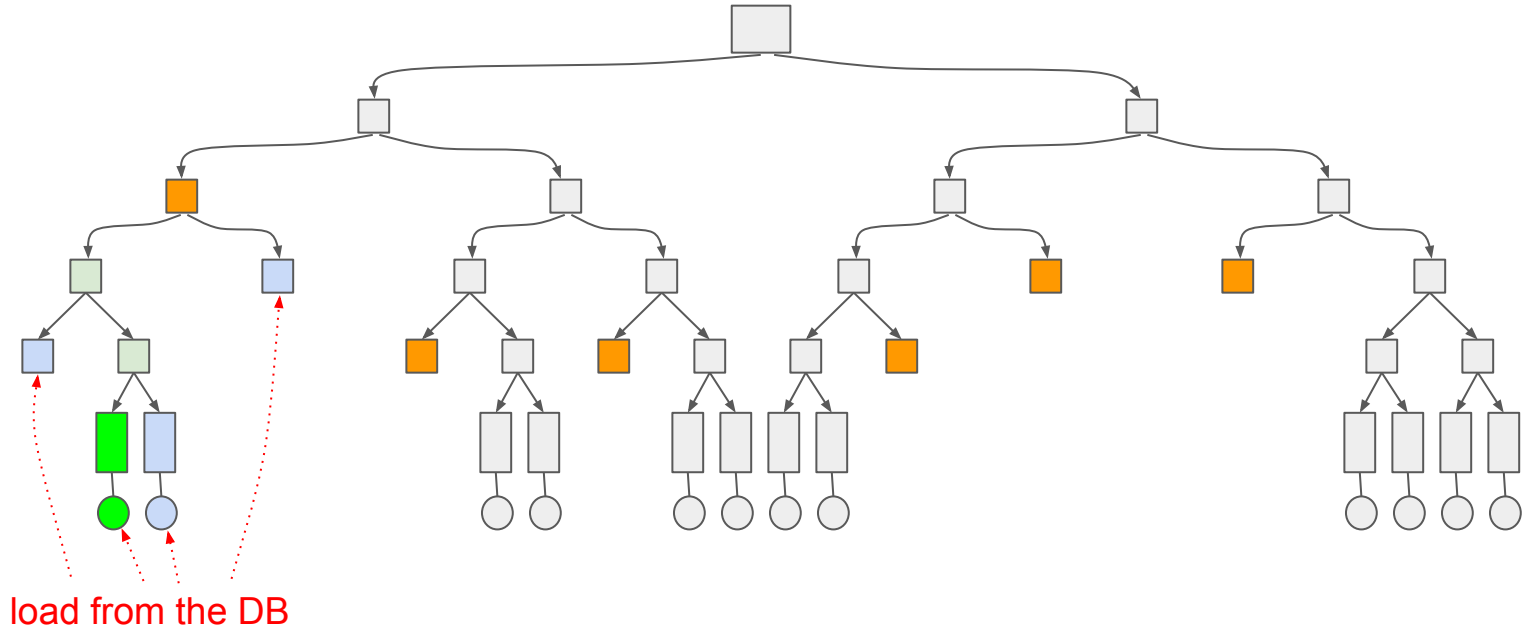
Intermediate hashes - solution (almost) without tradeoffs



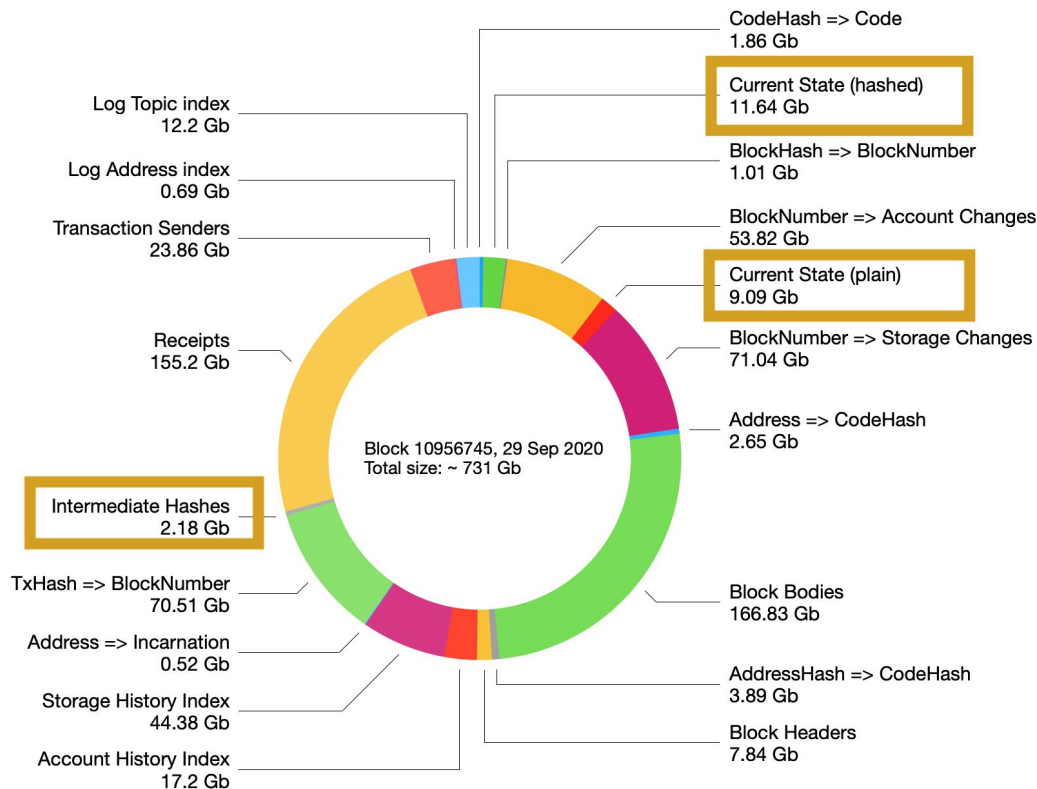
need to modify this

store these in the database prefix => hash

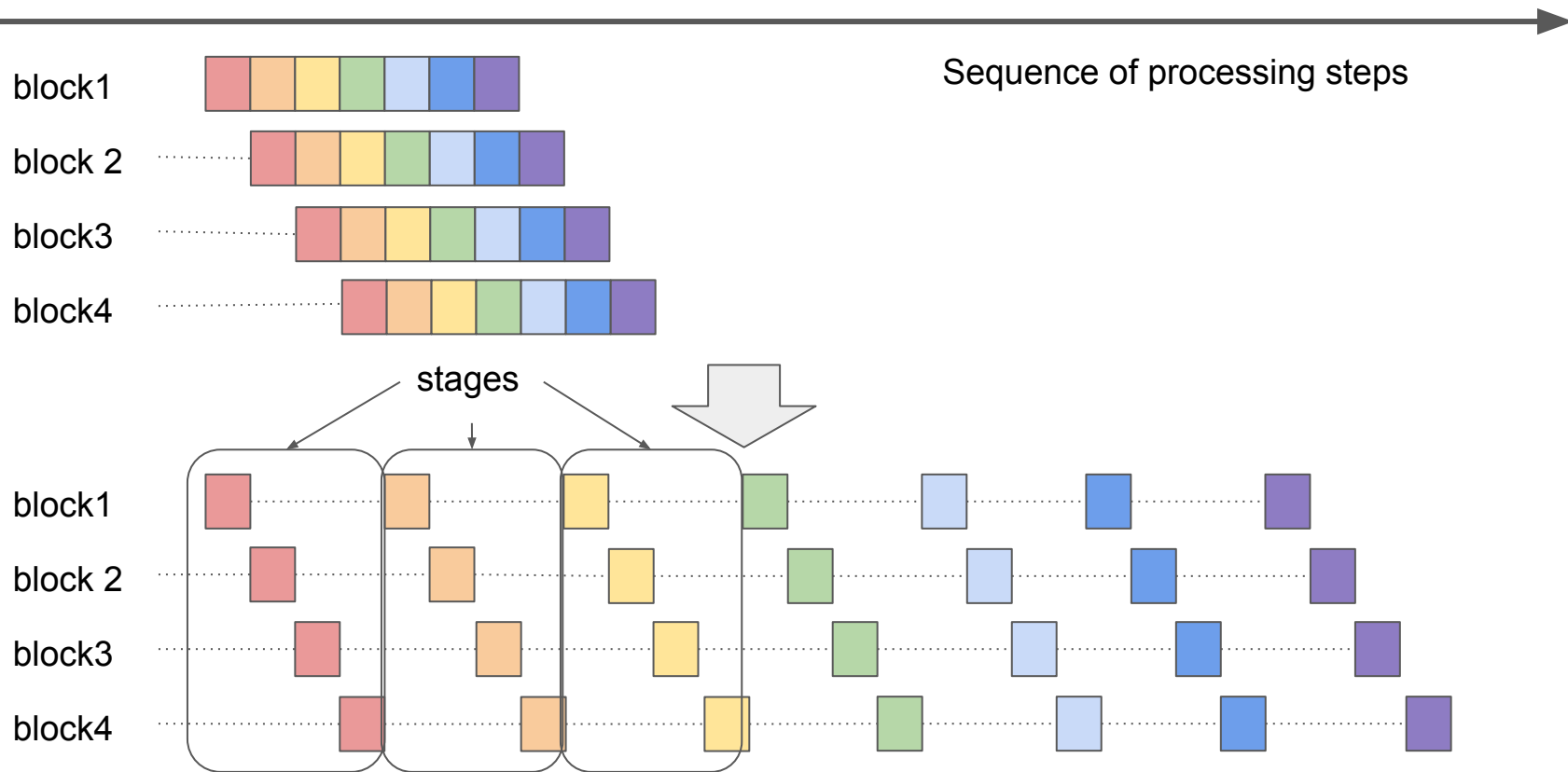
Intermediate hashes - solution (almost) without tradeoffs



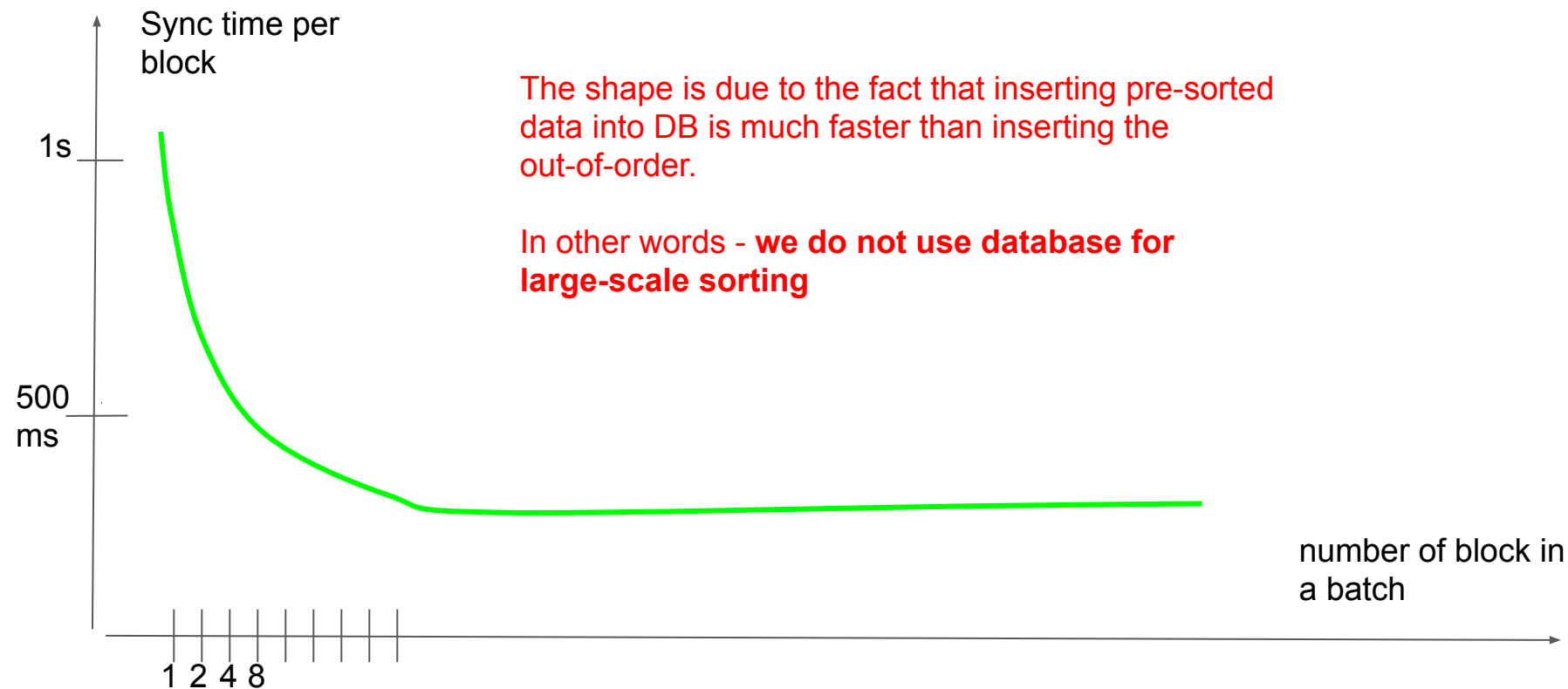
Compact storage (archive node)



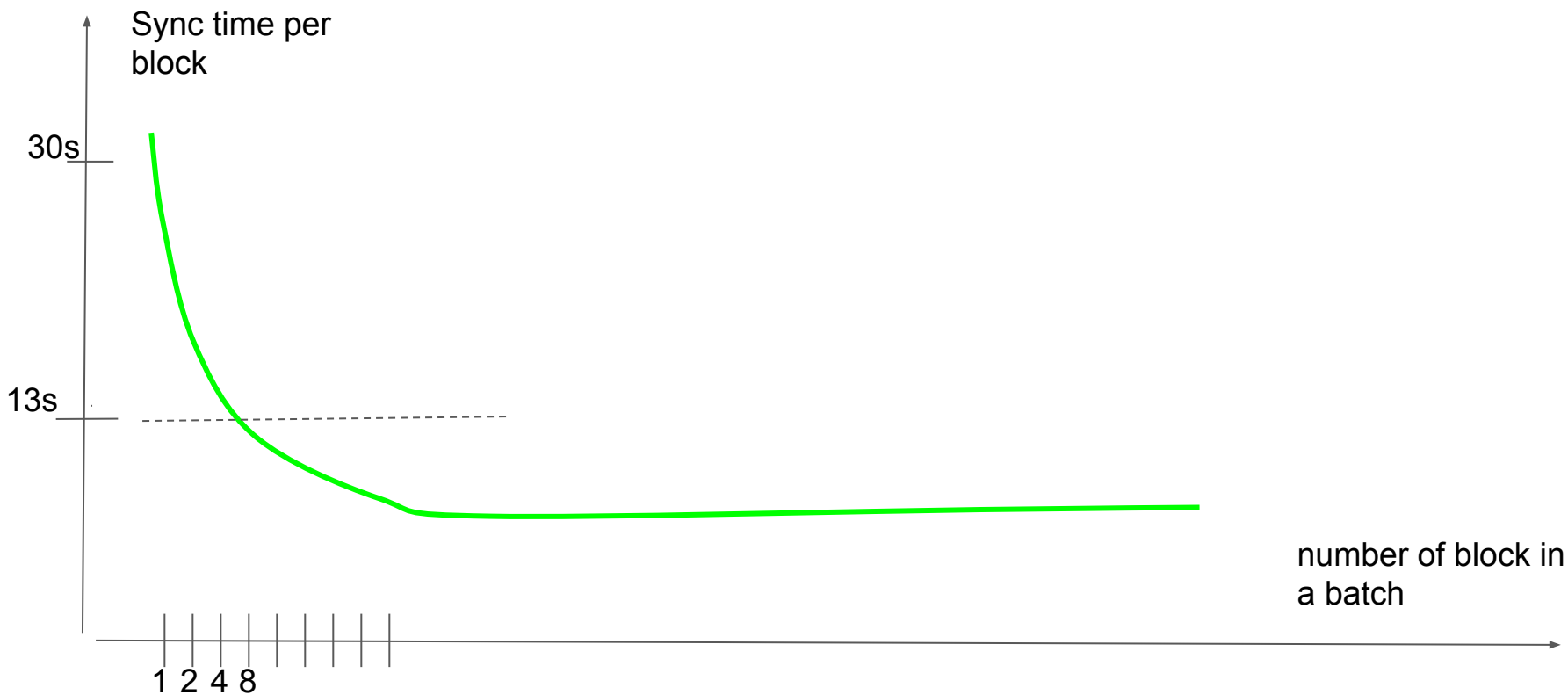
CONCURRENCY - Staged sync



Staged sync



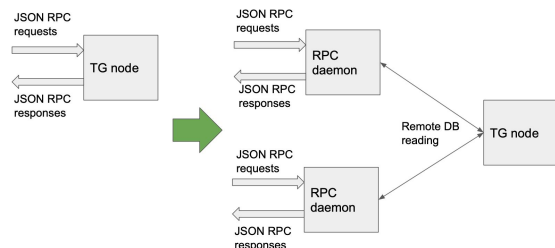
Staged sync on HDD (for academic purposes?)



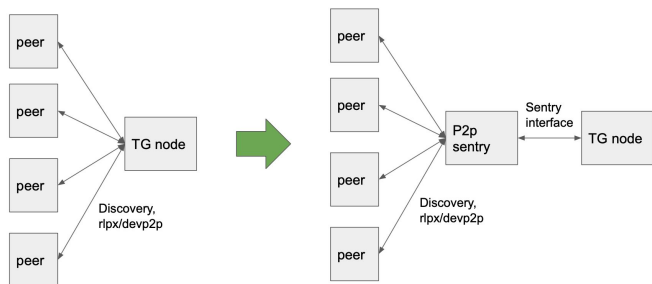
ARCHITECTURE - components, components

Core devs call in July 2020

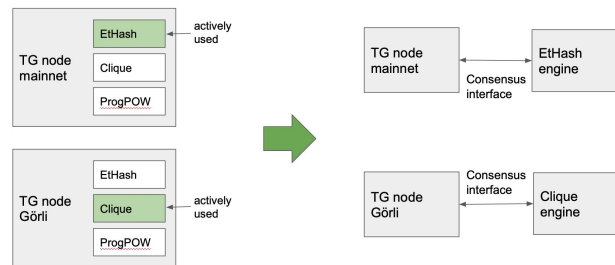
RPC daemon split (done)



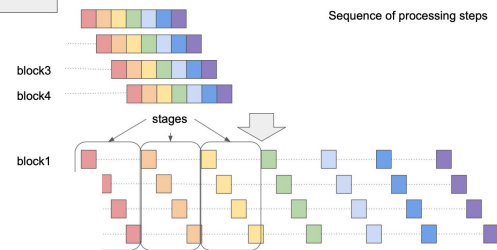
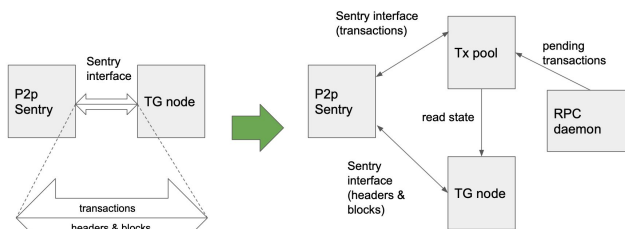
P2p sentry split (in progress)



Consensus engine split (in progress)



Transaction pool split (not started)



What about eth1x => eth10x (or any x)?

There are 3 main challenges:

1. Execution time for worst-case block (aka DOS attacks)
 - a. Access to absent items in the state (can be protected against by filters - bloom, cuckoo)
 - b. Access to present items in the state (less potent - protection is via efficient state access or gas penalties for access, e.g. access quotas per transaction, or EIP-2929)
 - c. Computations - gas cost of SHA3 and precompiles were/are being calibrated to 20 Mgas/sec.
Protection - recalibration or more efficient implementation (**and no rush in making things cheaper please!**)
2. Propagation of large blocks (potential solution via “thin blocks” of transaction hashes)
3. Storage of large blocks (off-loading storage to multitude of other networks, efficient download).