Turbo-Geth - optimising Ethereum client(s)

Alexey Akhunov

Supported by a grant from Ethereum Foundation

Ledgerwatch

Outline

How it started

Authenticated data structures (trees)

Persistence of state in geth and turbo-geth

Alternatives - sparse merkle trees, self-balancing trees

Block/tx processing architectures: geth, turbo-geth, Ethermint

Latest performance data

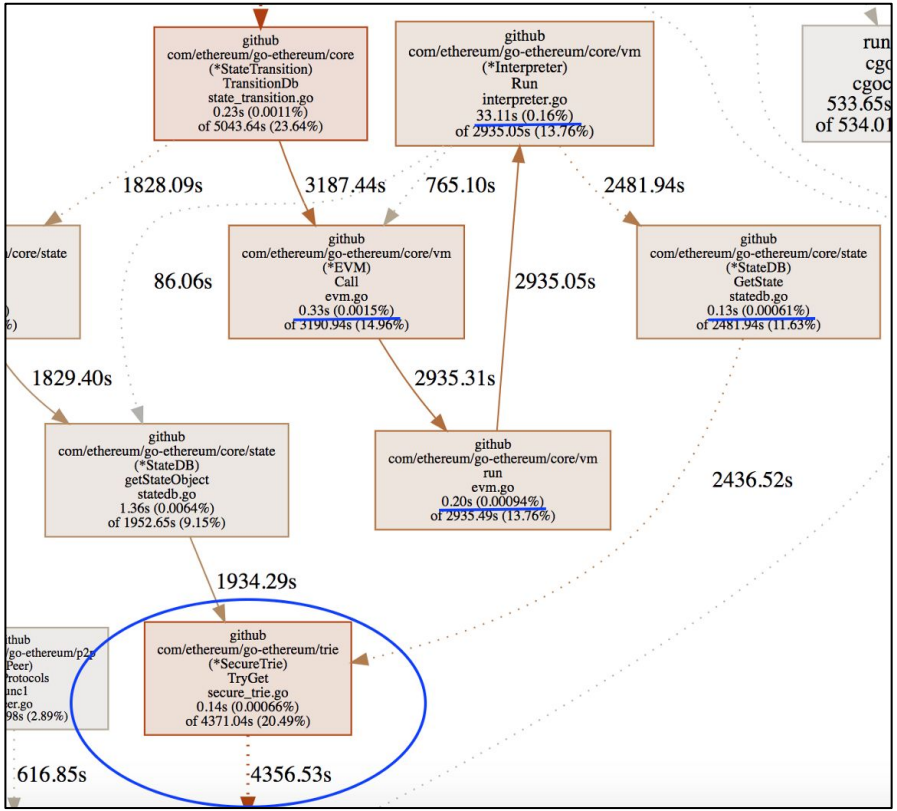
Light clients?

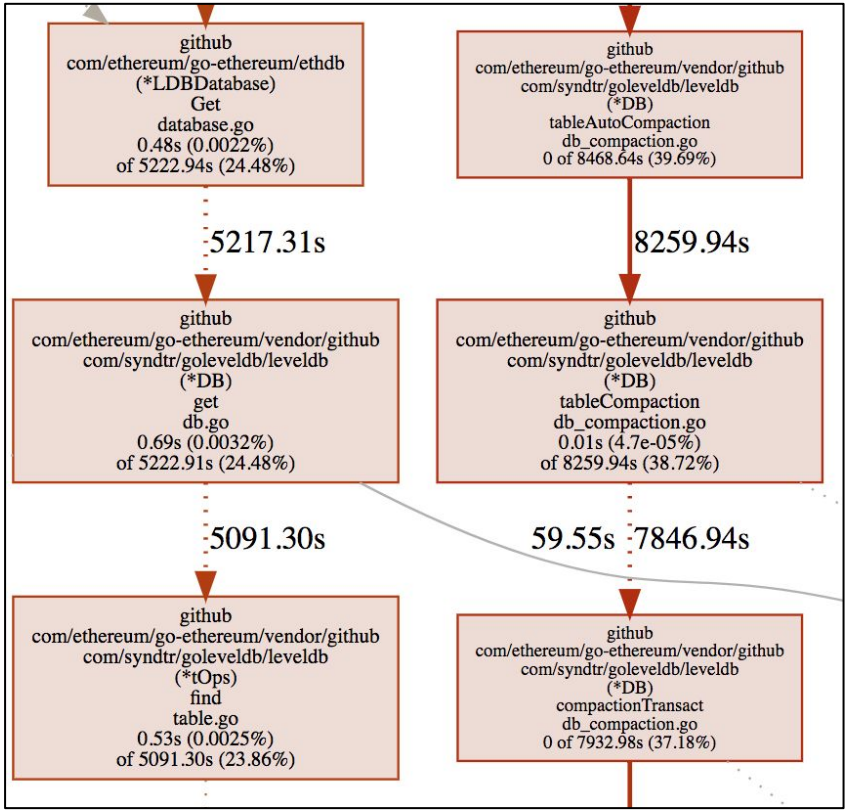
Future experiments (in-memory state, graph DBs)

How it started

```
30 November 2017

> git clone git@github.com:ethereum/go-ethereum.git
> cd go-ethereum
> make
> ./build/bin/geth

… 5 days later

> ./build/bin/geth --cpuprofile cpu.prof

… 3 hours later

> go tool pprof cpu.prof
(pprof)> png
```

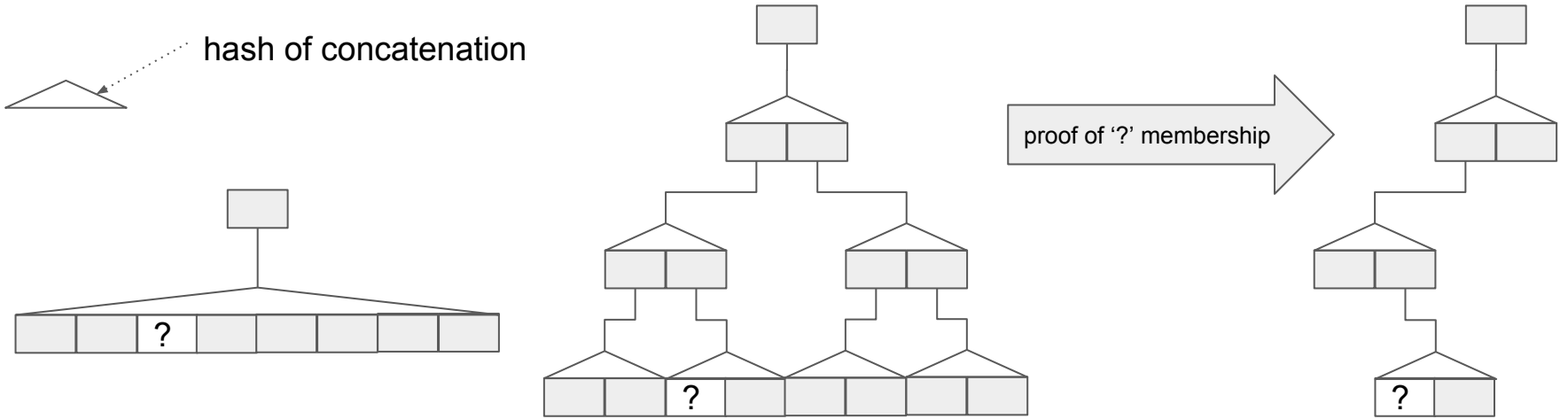
How it started





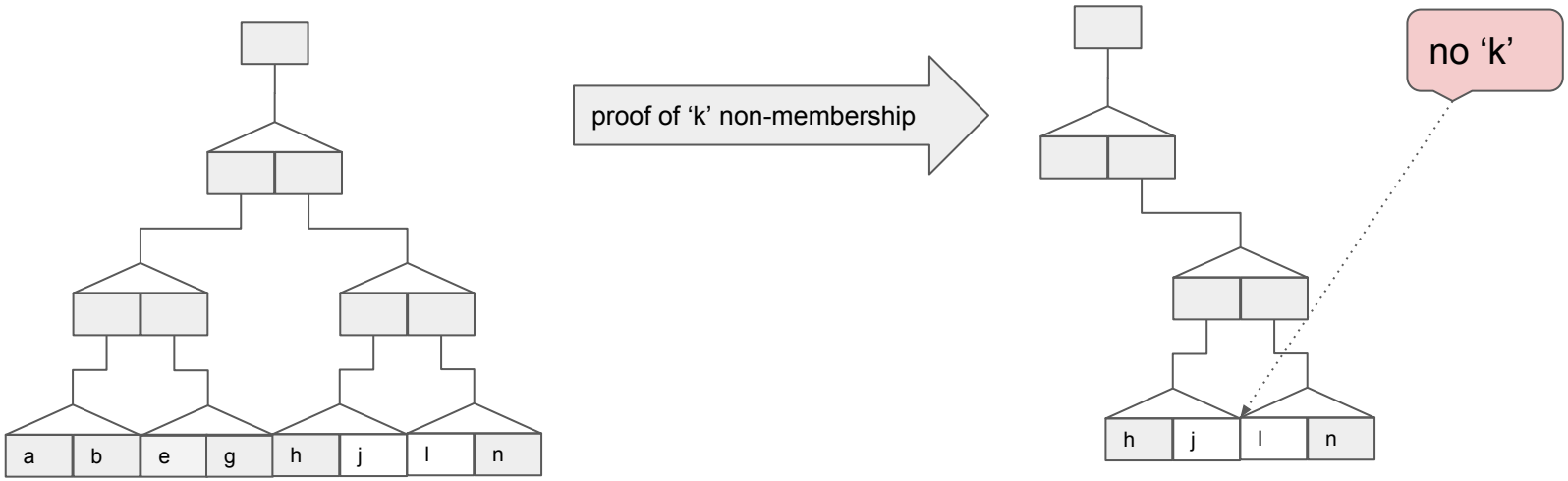
Authenticated data structures

Data on a blockchain has to be stored in a tamper-proof data structures. Merkle tree is an implementation of tamper-proof list. Its main feature is short proof of membership:



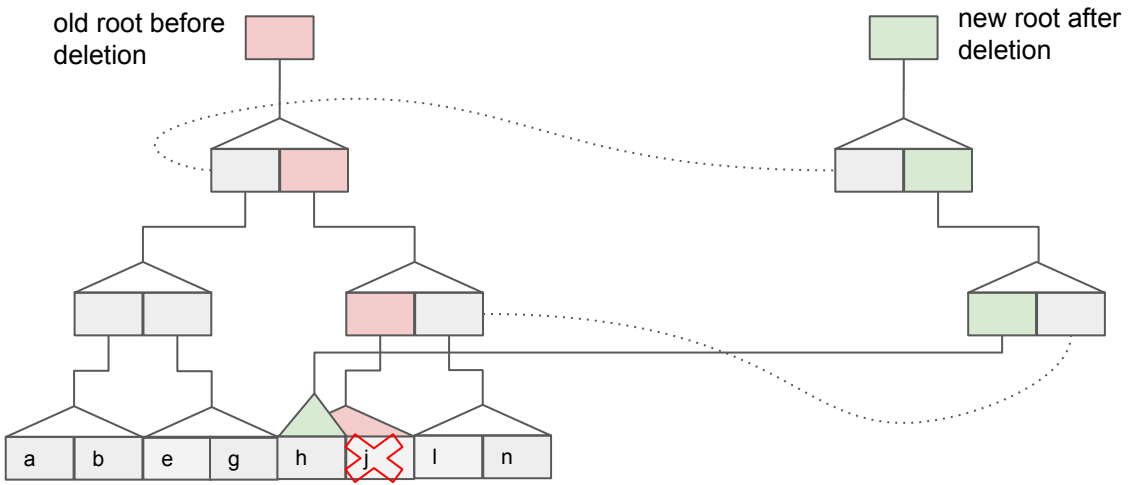
What about proof of non-membership?

How would we prove non-membership? In an ordinary Merkle tree, such proof consists of the entire tree, which is no good. Another alternative would be a sorted Merkle tree:



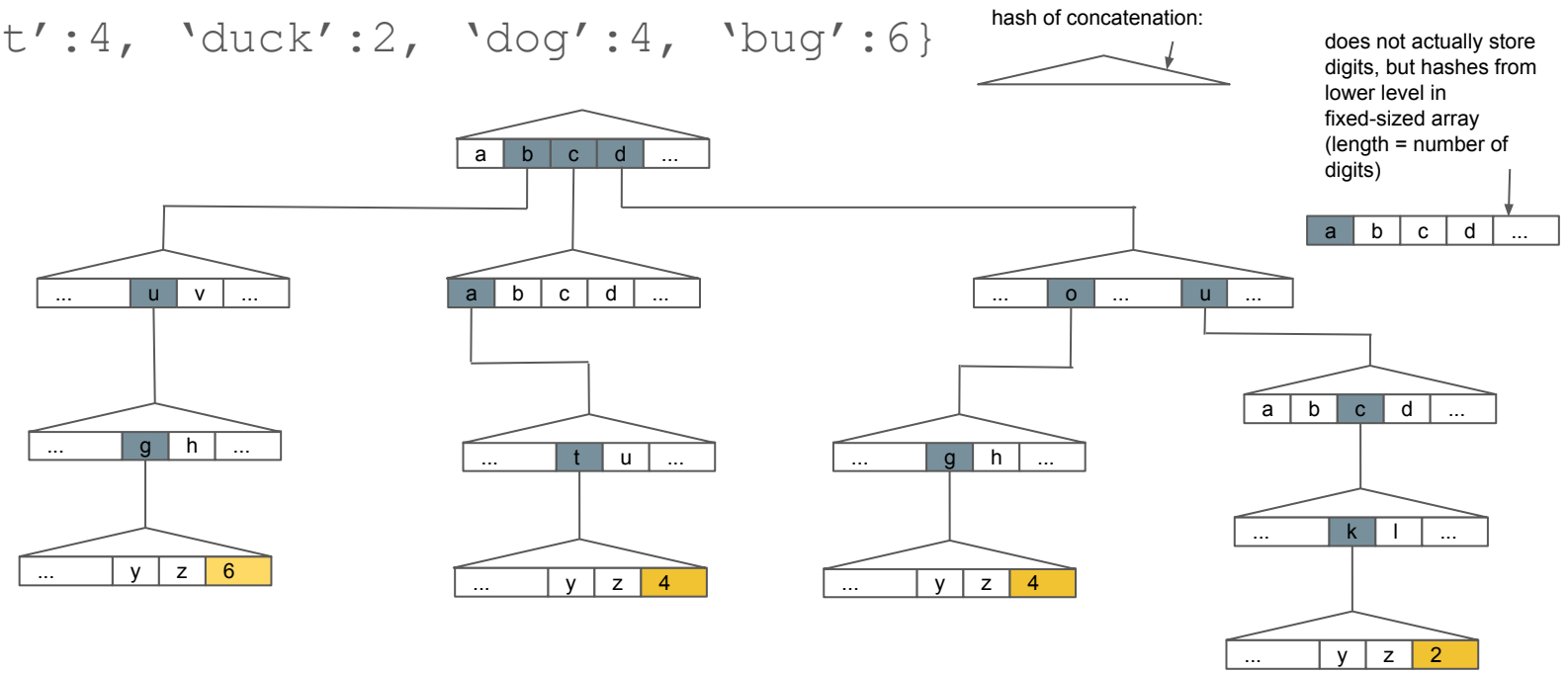
Encoding modifications

Simply sorting the list is not enough to efficiently encode modifications - one needs to preserve the structure of the tree to reuse most of the nodes



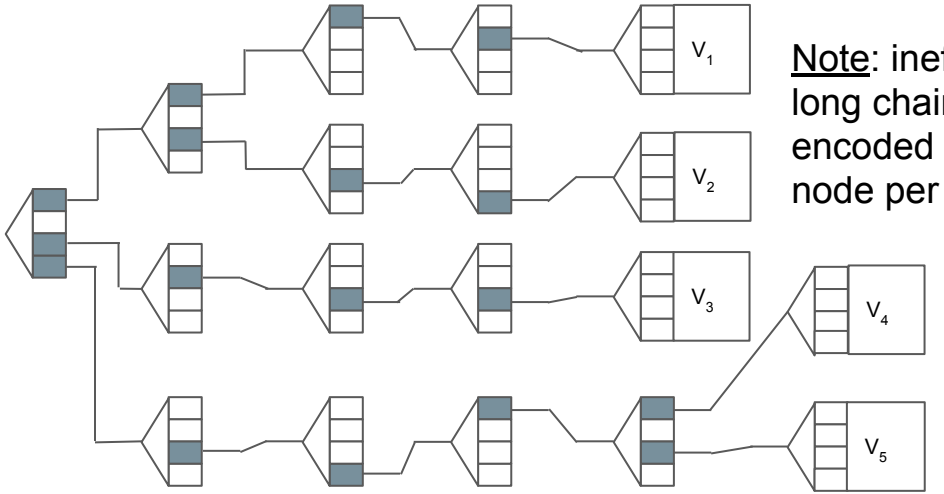
Merkle radix tree (trie)

```
{'cat':4, 'duck':2, 'dog':4, 'bug':6}
```



Merkle radix tree (trie)

Merkle radix tree supports compact encoding of modifications, short proofs of membership and non-memberships. However, if the keys are relatively long, representation tends to be quite inefficient:

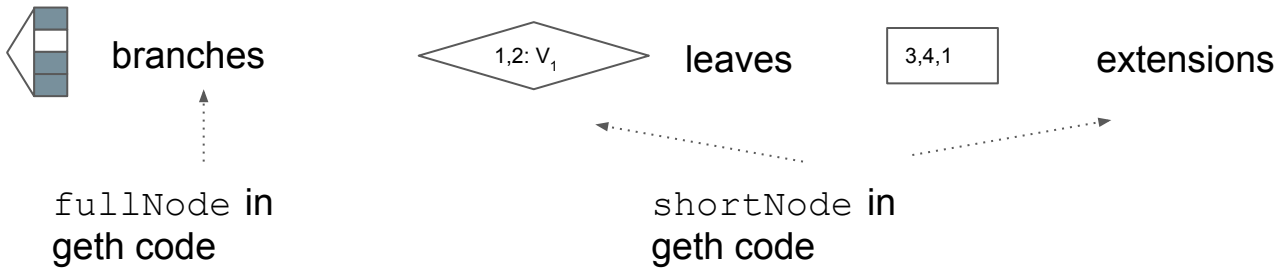


Note: inefficiency comes from the fact that long chains of digits for the keys are encoded by the chains of tree nodes, 1 node per digit

Merkle Patricia tree (trie)

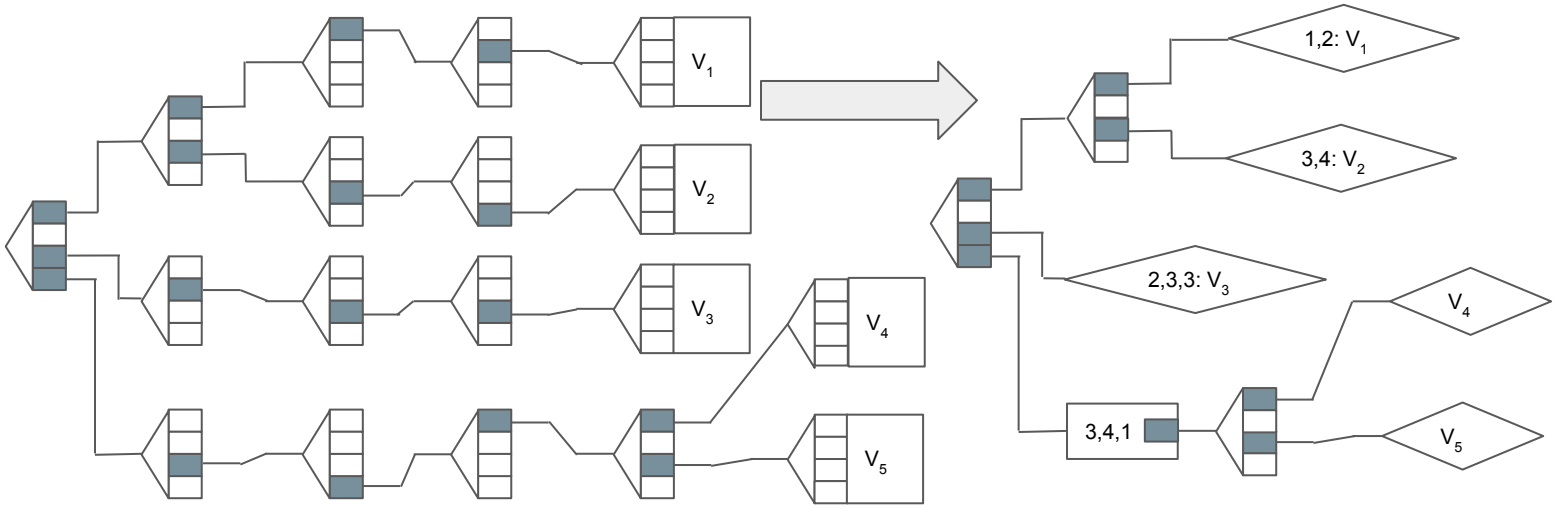
PATRICIA stands for “**P**ractical **A**lgorithm **T**o **R**etrieve **I**nformation **C**oded **I**n **A**lphanumeric” and was invented in 1968 independently by Donald Morrison and Gernot Gwehenberger.

Patricia tree in Ethereum’s version addresses the inefficiency of radix tree by having 3 types of nodes:



Merkle radix tree (trie)

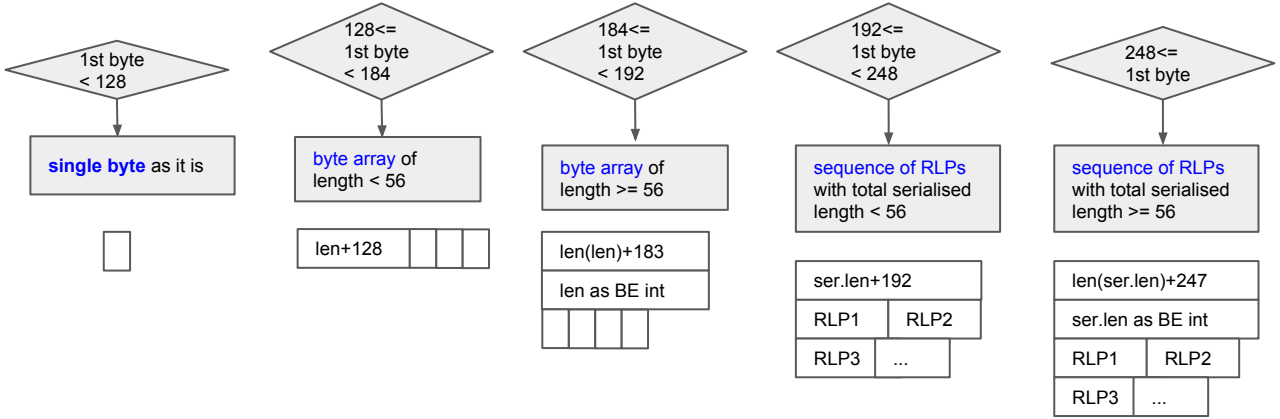
If we transform the radix tree from earlier to Patricia tree, it will look like this:



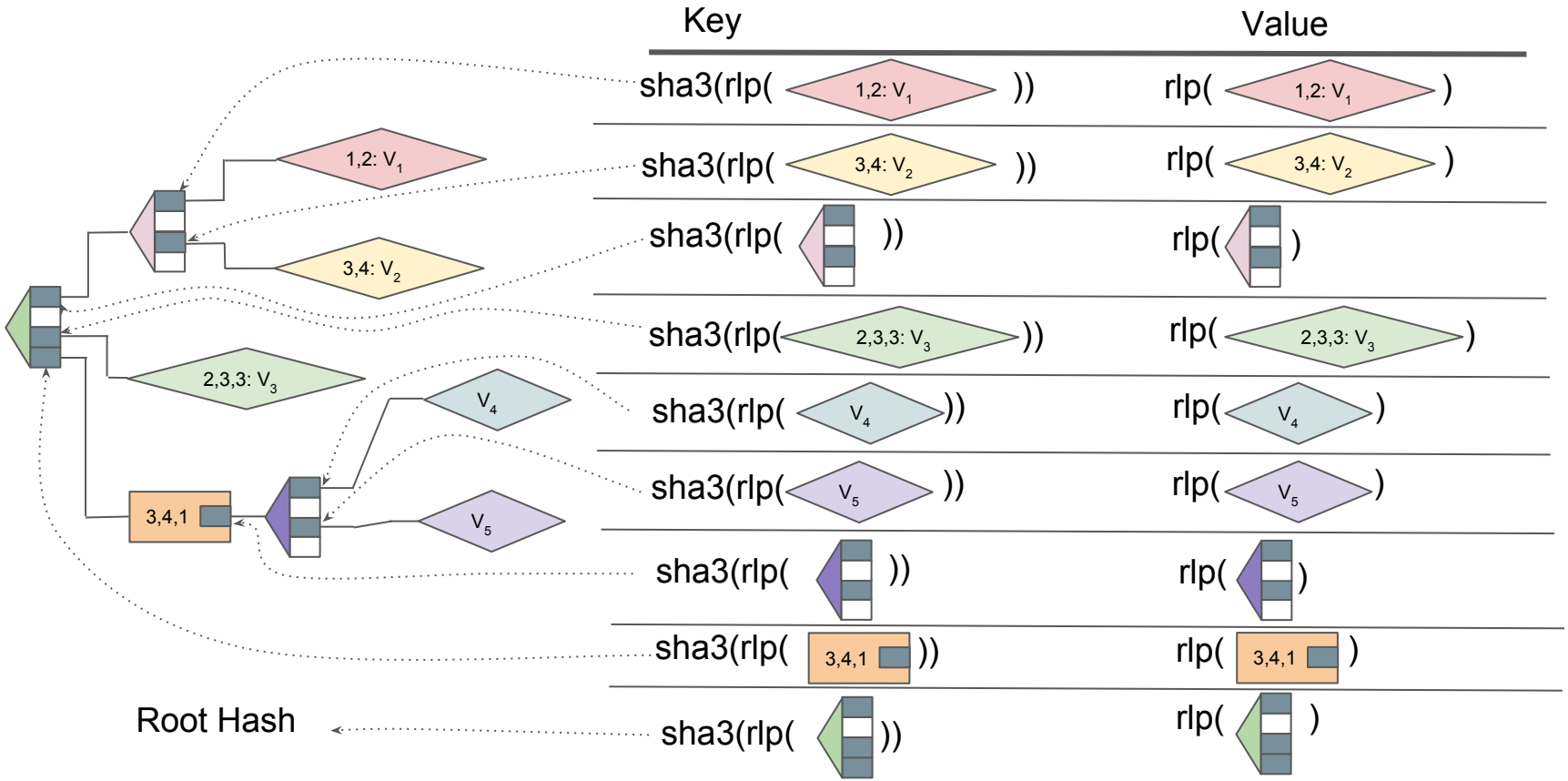
Ethereum: radix 16 (0,1,2,...,b,c,d,e,f), all keys 256 bit == 32 bytes == 64 nibbles

RLP - Recursive Length Prefix encoding

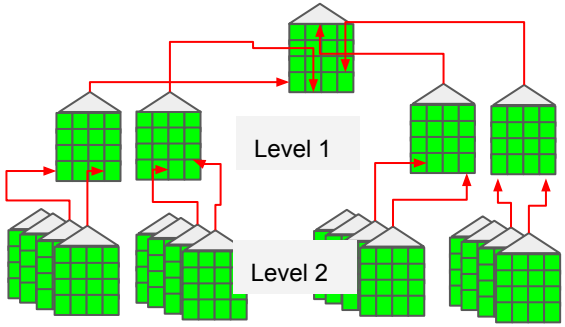
RLP is able to encode two types of structures: byte arrays of arbitrary size and sequences of other RLP-encoded structures. Interpretation of RLP stream:

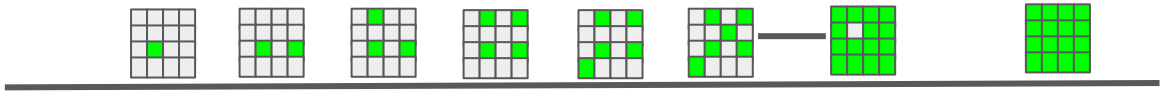


Persistence of Patricia tree in geth



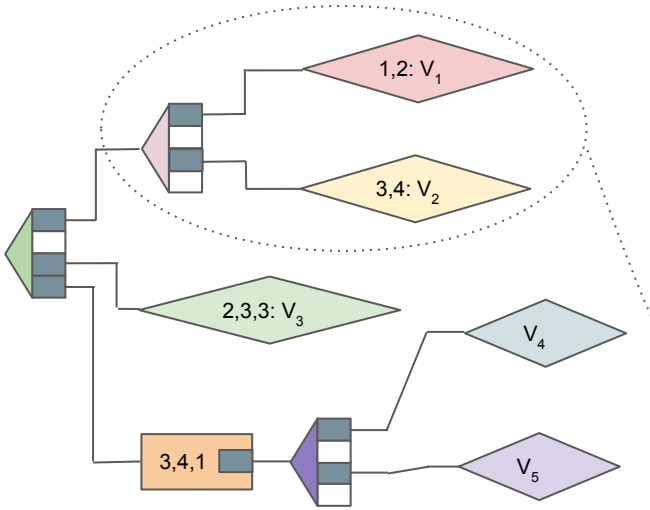
Block 5’032’091





							$1 = 16^0$
							$16 = 16^1$
							$256 = 16^2$
Level 3							$4’096 = 16^3$
Level 4							$65’536 = 16^4$
Level 5				7	41	1’023’387	25’141
Level 6	5.7m	4.4m	2.1m	0.7m	117k	39’123	
Level 7	18m	1.1m	31k	628	11		
Level 8	2.3m	72k	86				
Level 9	144k	1’675					
Level 9, 10, 11, 12	3’484	67					

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

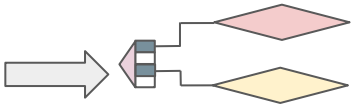
Goes here because it is sorted

DEPTH DOES NOT MATTER

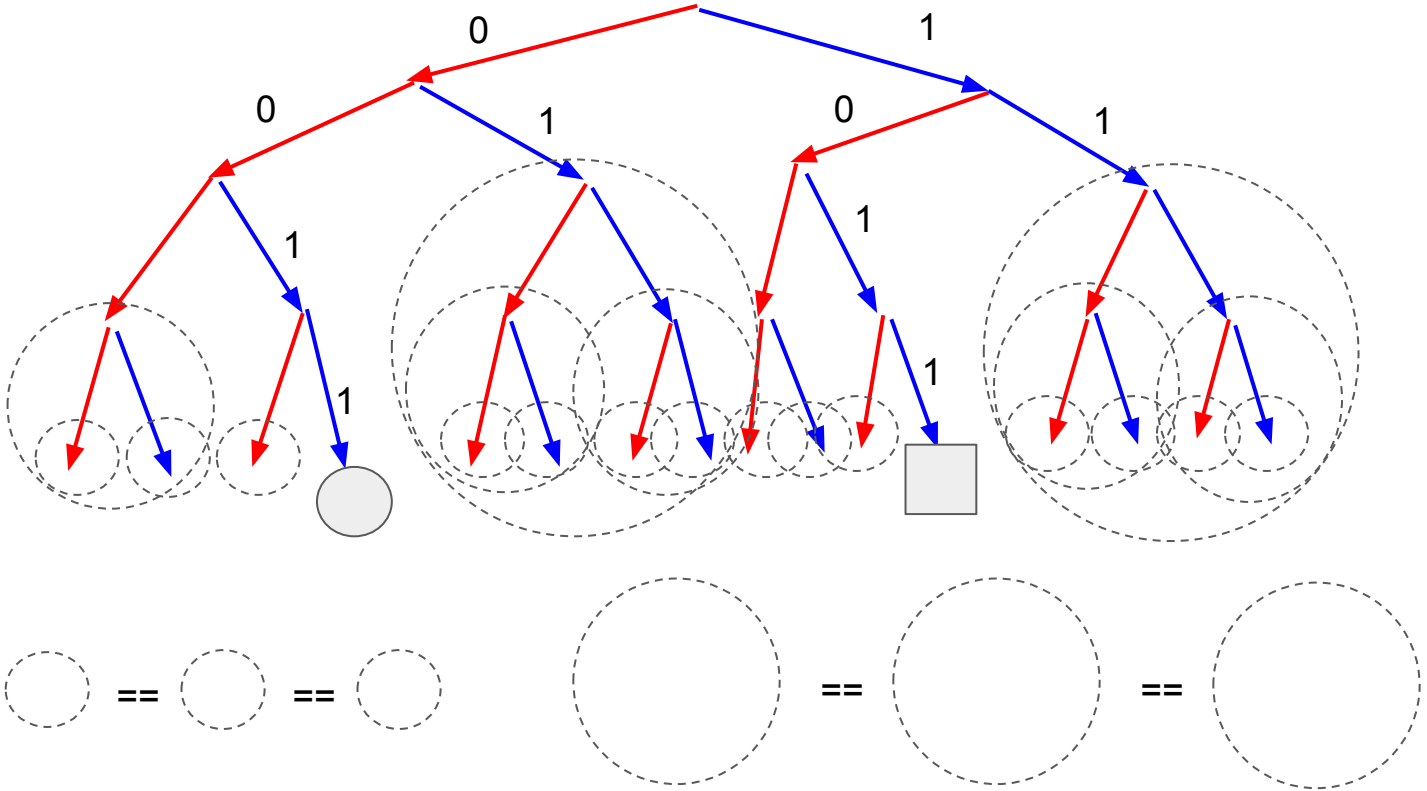
Range query:
1,*,*,*,>=-blockNr

1,1,1,2,-blockNr V_1

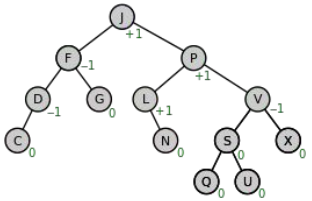
1,3,3,4,-blockNr V_2



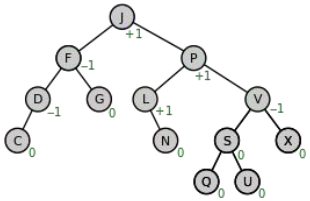
Alternatives - Sparse Merkle tree (binary radix tree)



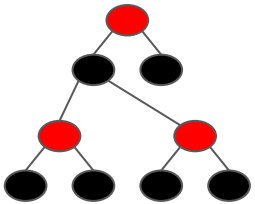
Alternatives - self-balancing trees (AVL, Red black)



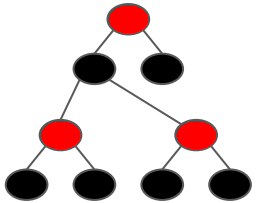
+(kA, vA) + (kB, vB) ≢



+(kB, vB) + (kA, vA)

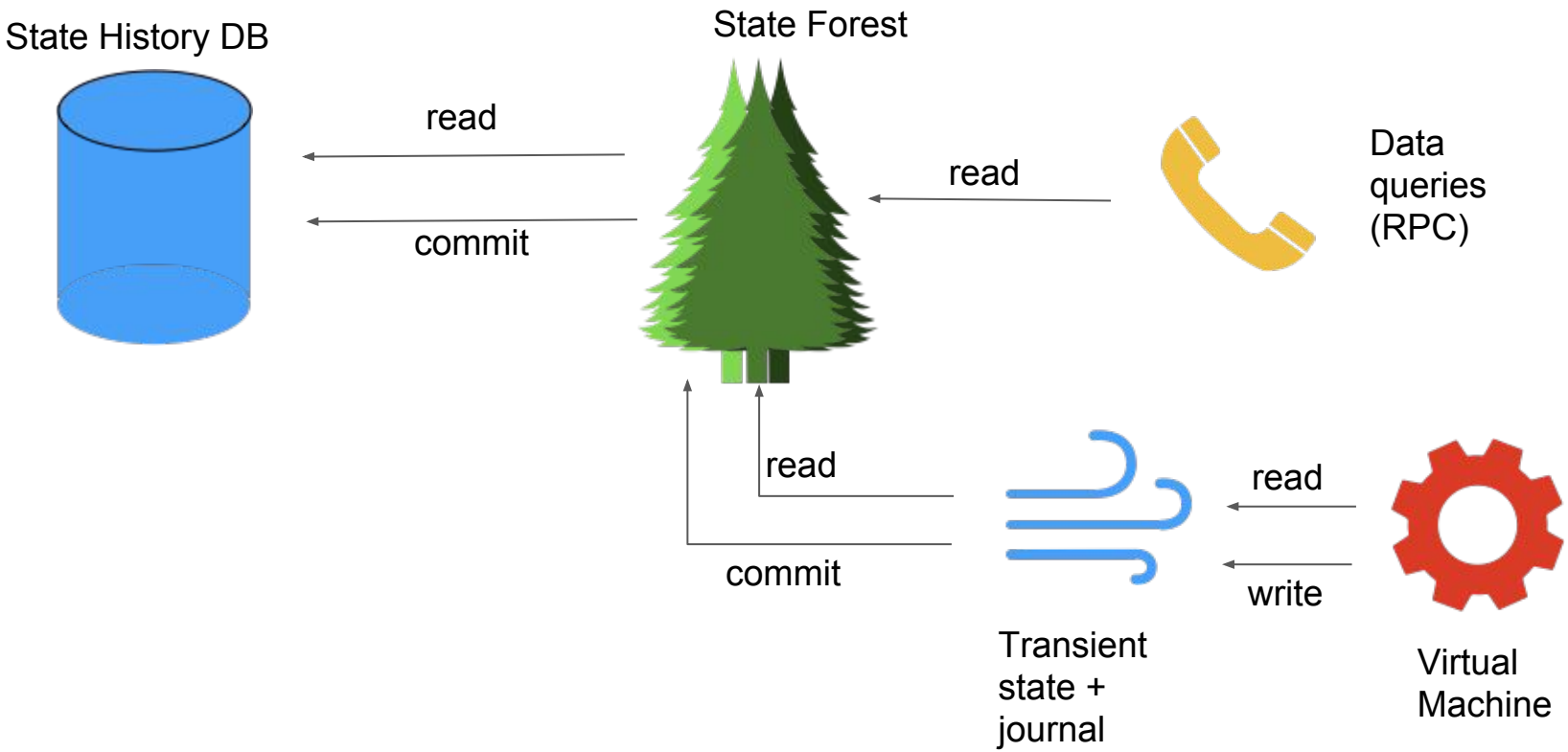


+(kA, vA) + (kB, vB) ≢

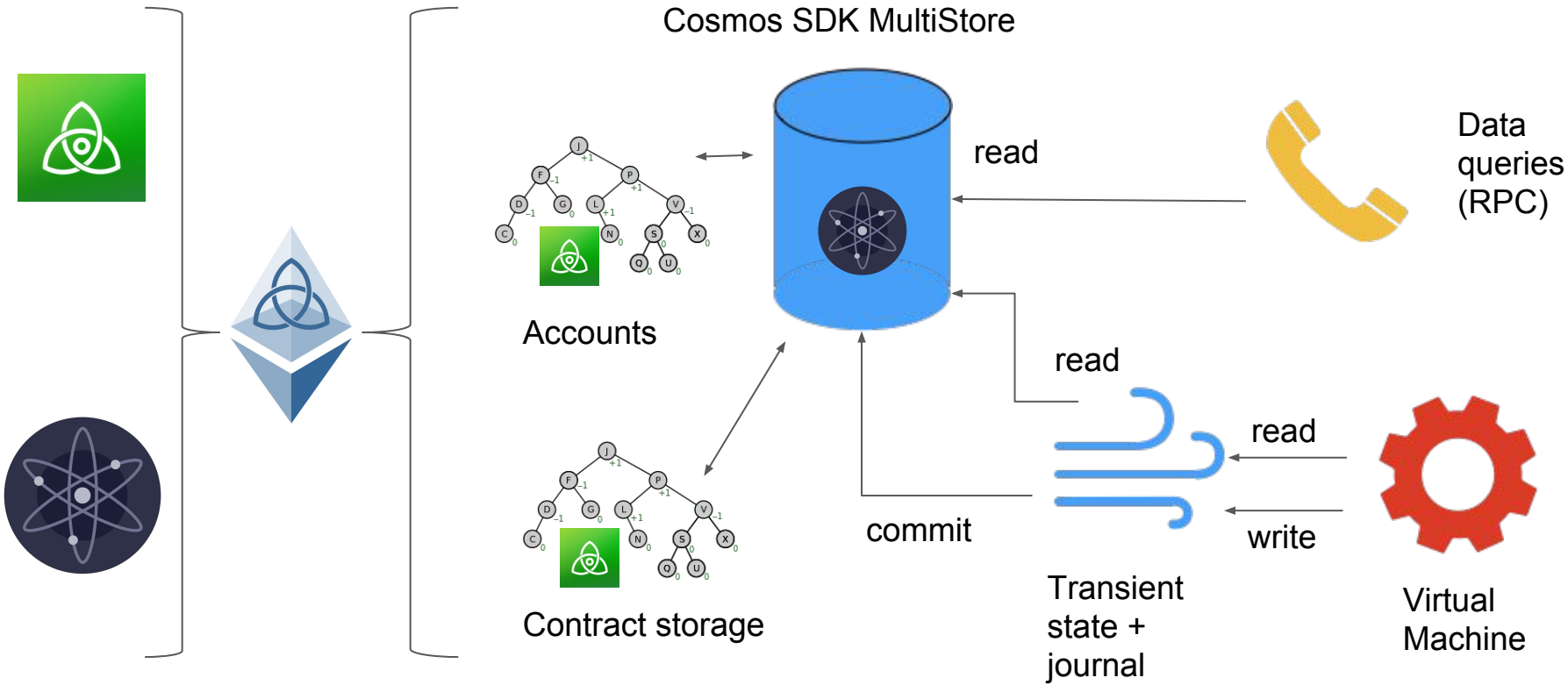


+(kB, vB) + (kA, vA)

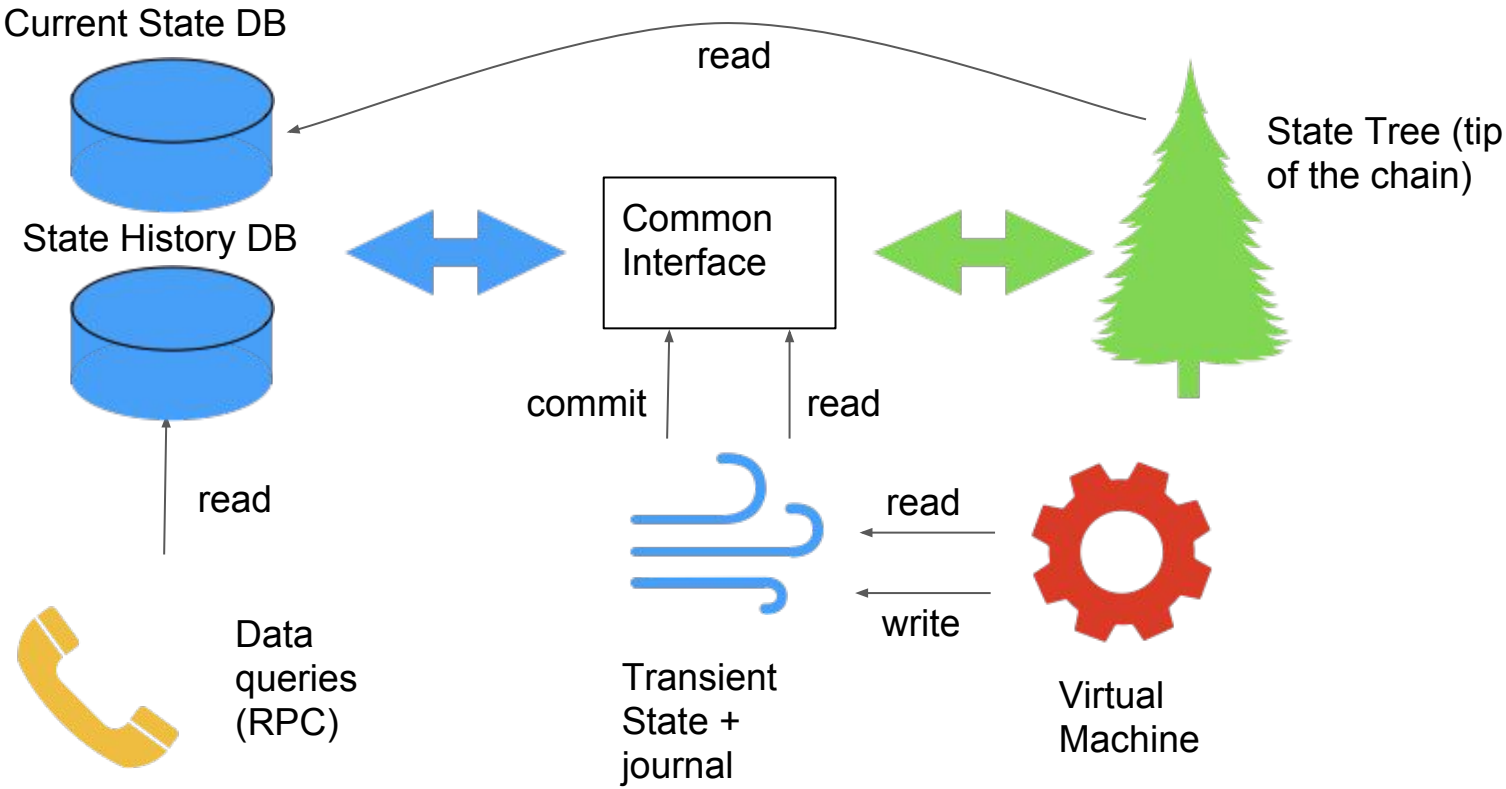
State and Virtual Machine in geth



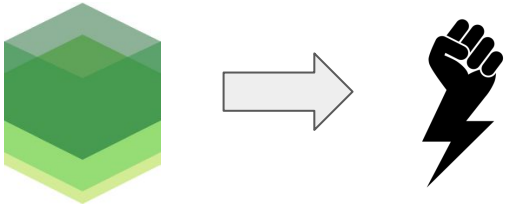
Go-ethereum and Ethermint 2.0 (PoC)



State and Virtual Machine in turbo-geth

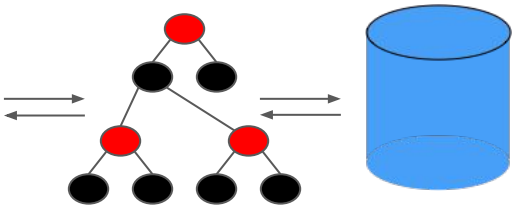


Other differences

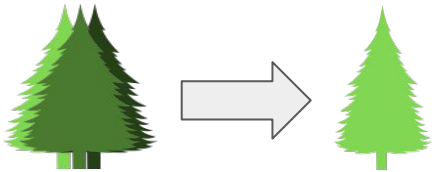


BoltDB (B+Tree) instead of LevelDB
(Log-Structured Merge)

B+tree traversal instead of
random read is predominant
DB workload

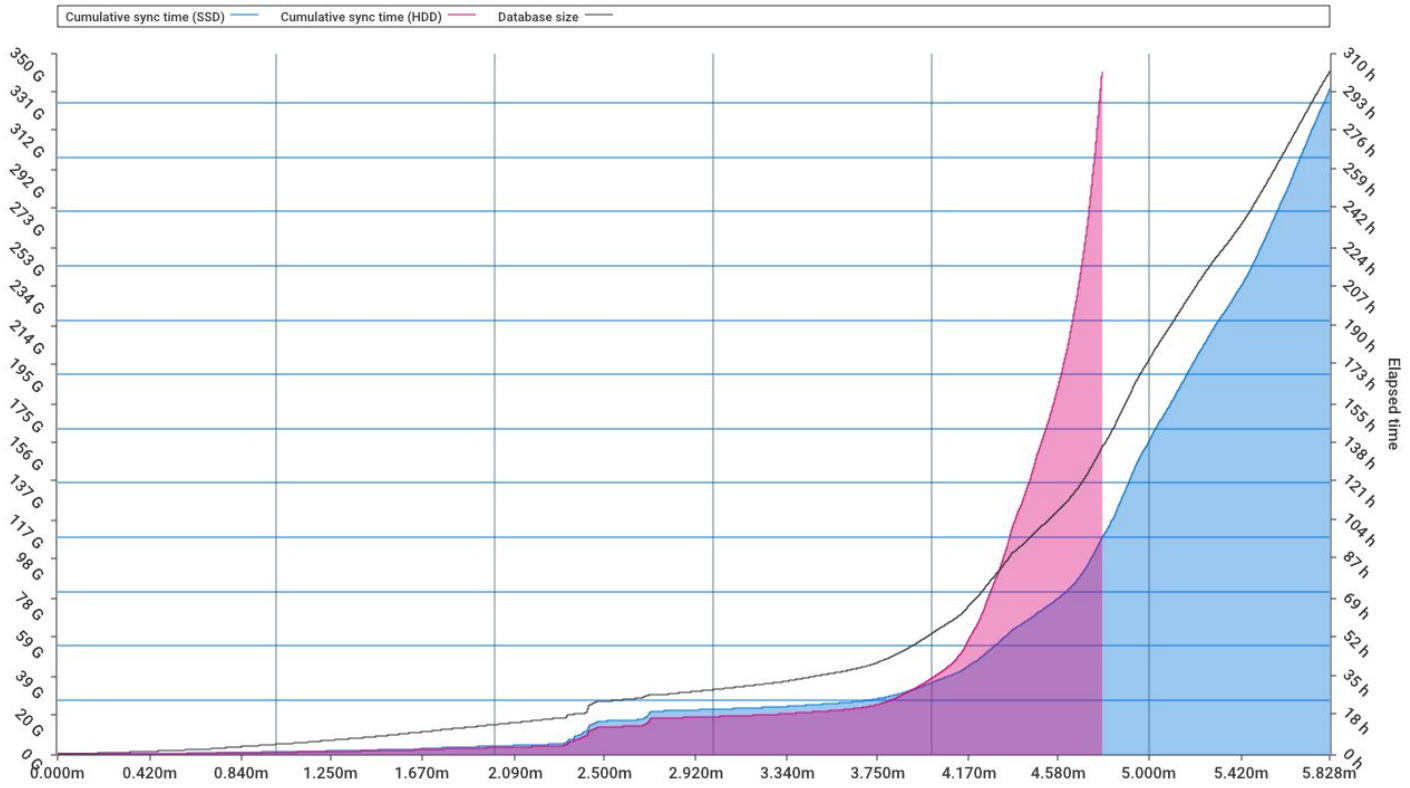


Large DB transaction buffer (Red-Black trees)

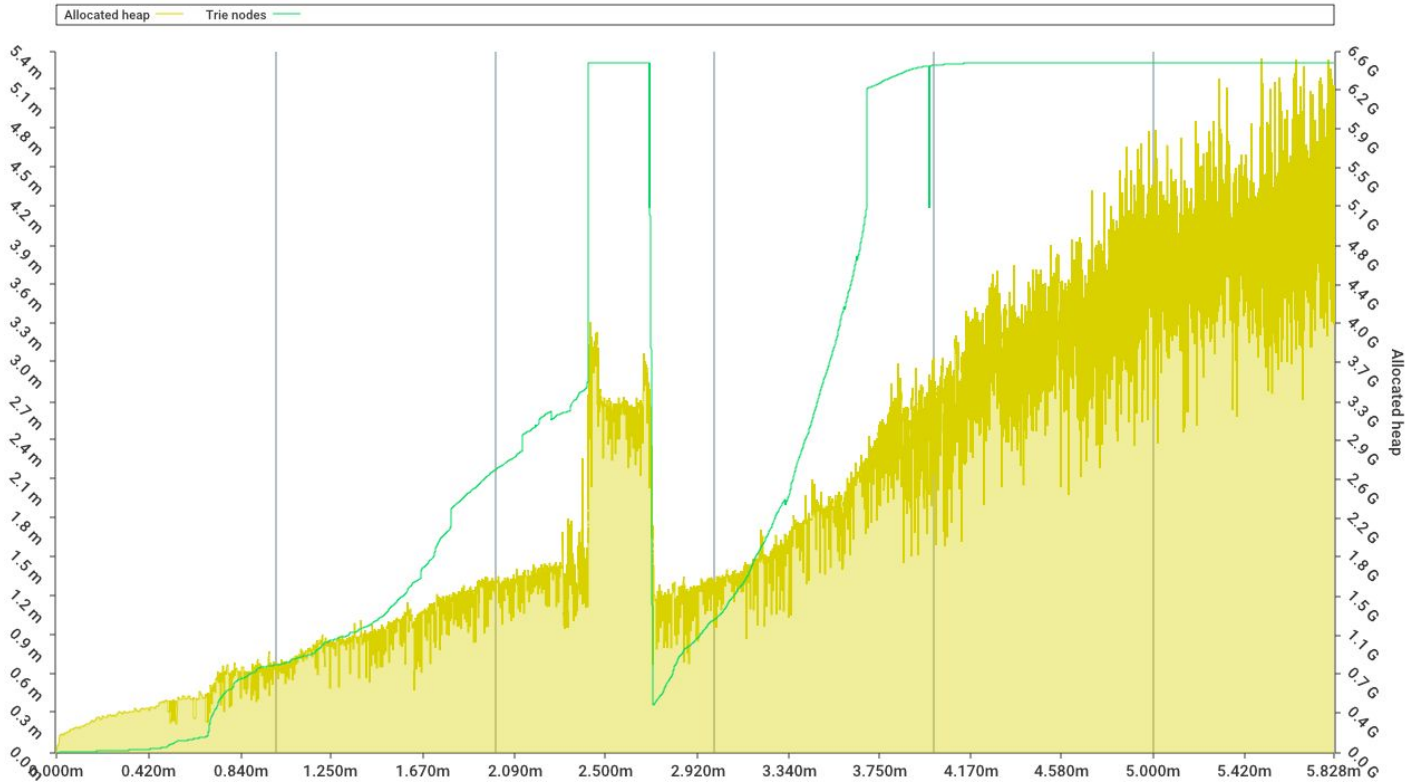


Cannot maintain multiple tips of the chain -> reorgs
have to rewind history and the trie

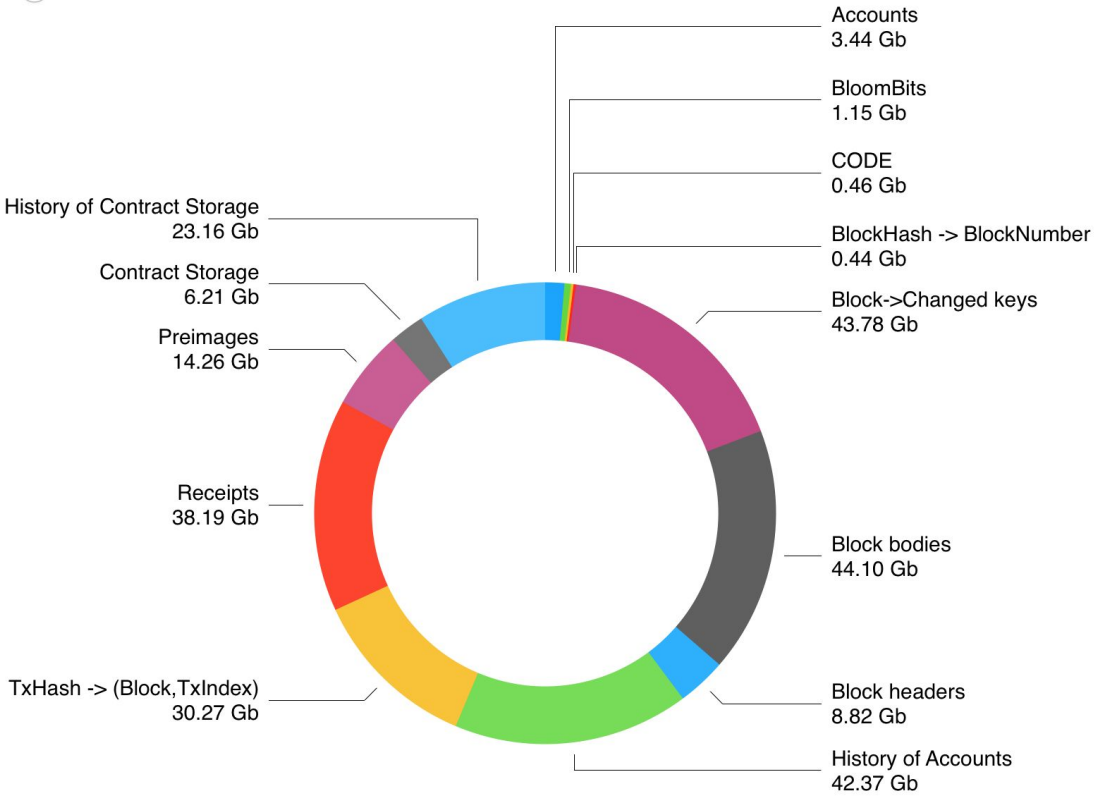
Full sync (fast sync not supported) time



Full sync space



Disk usage (block 5.83m) - total 260 Gb



Roadmap for the first release

Reorg testing (eth tester, hive?)

Fix RPC APIs

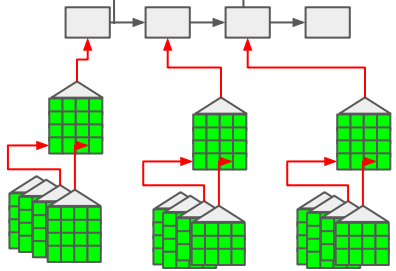
Support for retesteth

Database layout is expected to change even after that

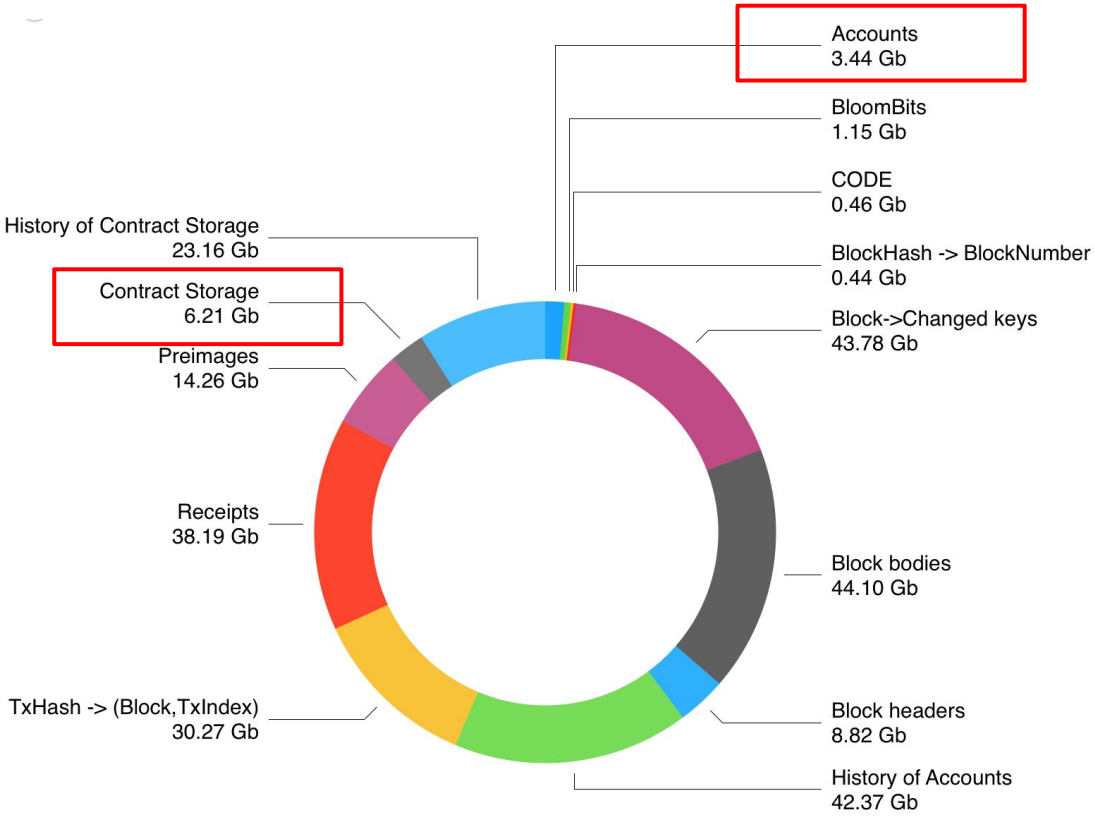
Light clients?

	eth/63	les/2	
Status	✓	✓	Handshake - negotiate version, network_id, genesis
[Get]Block(Headers|Bodies)	✓	✓	Get/Send headers/blocks by number or by hash
[Get]NodeData	✓	✓	Get/Send nodes of the patricia tree by hash
[Get]Receipts	✓	✓	Get/Send receipts by transaction hash
NewBlock[Hashes]	✓		Announce new block/block hash
Announce		✓	Announce new chain head
[Get]Proofs		✓	Get/Send merkle proof for given part of trie and block hash
[Get]ContractCode		✓	Get/Send code of given contract at block hash
SendTx	✓	✓	Add new transaction to the pool and relay
[Get]HelperTreeProofs		✓	Get/Send merkle proof of block hash/bloom filters

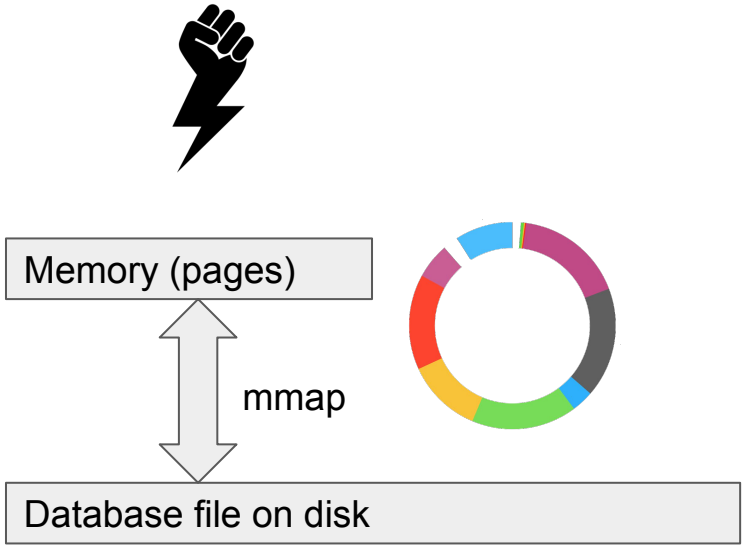
Require materialised
Patricia tree

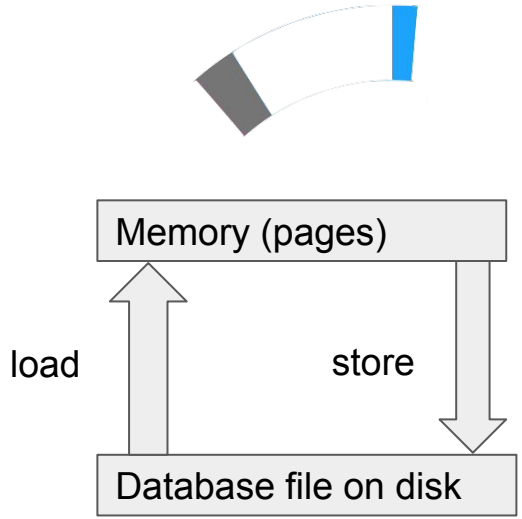


Future experiments - current state in memory

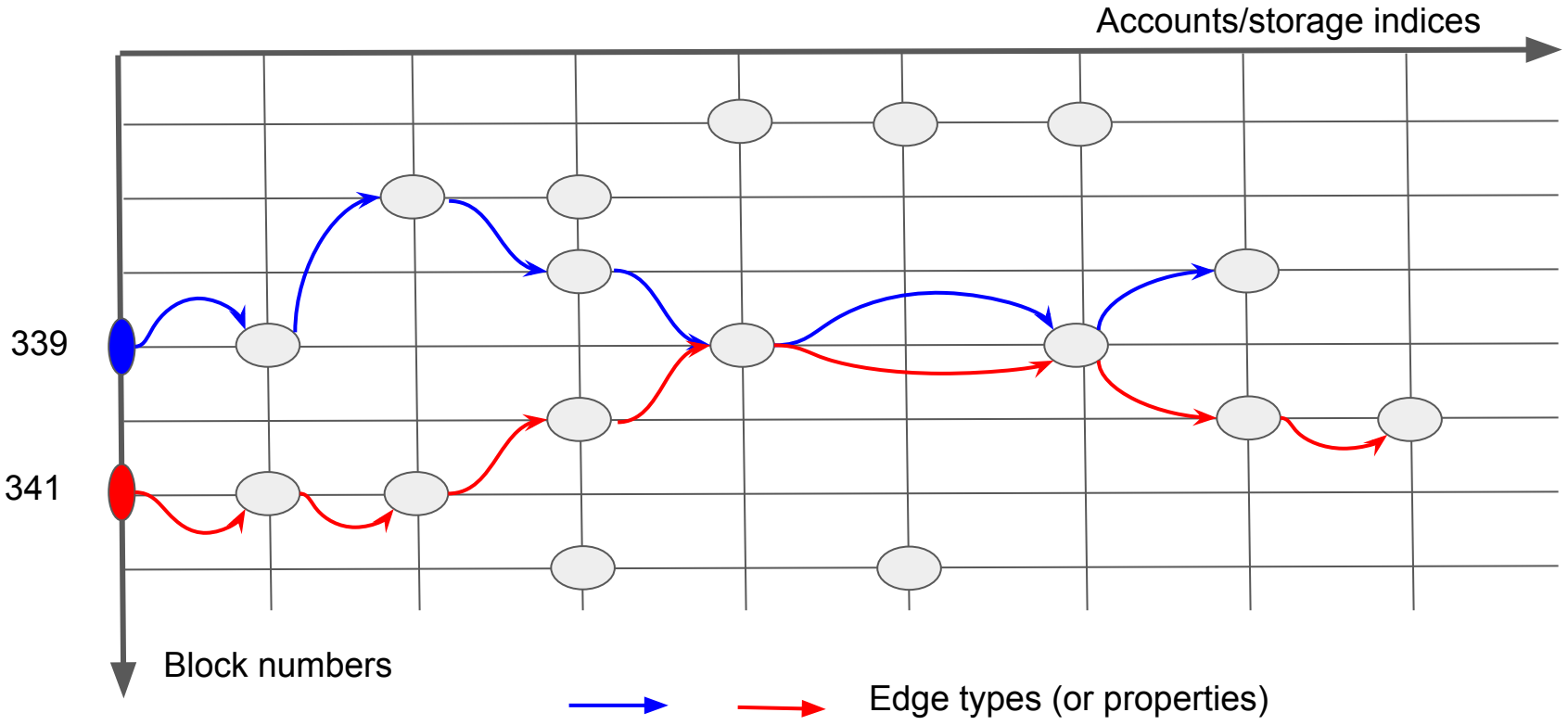


Future experiments - current state in memory

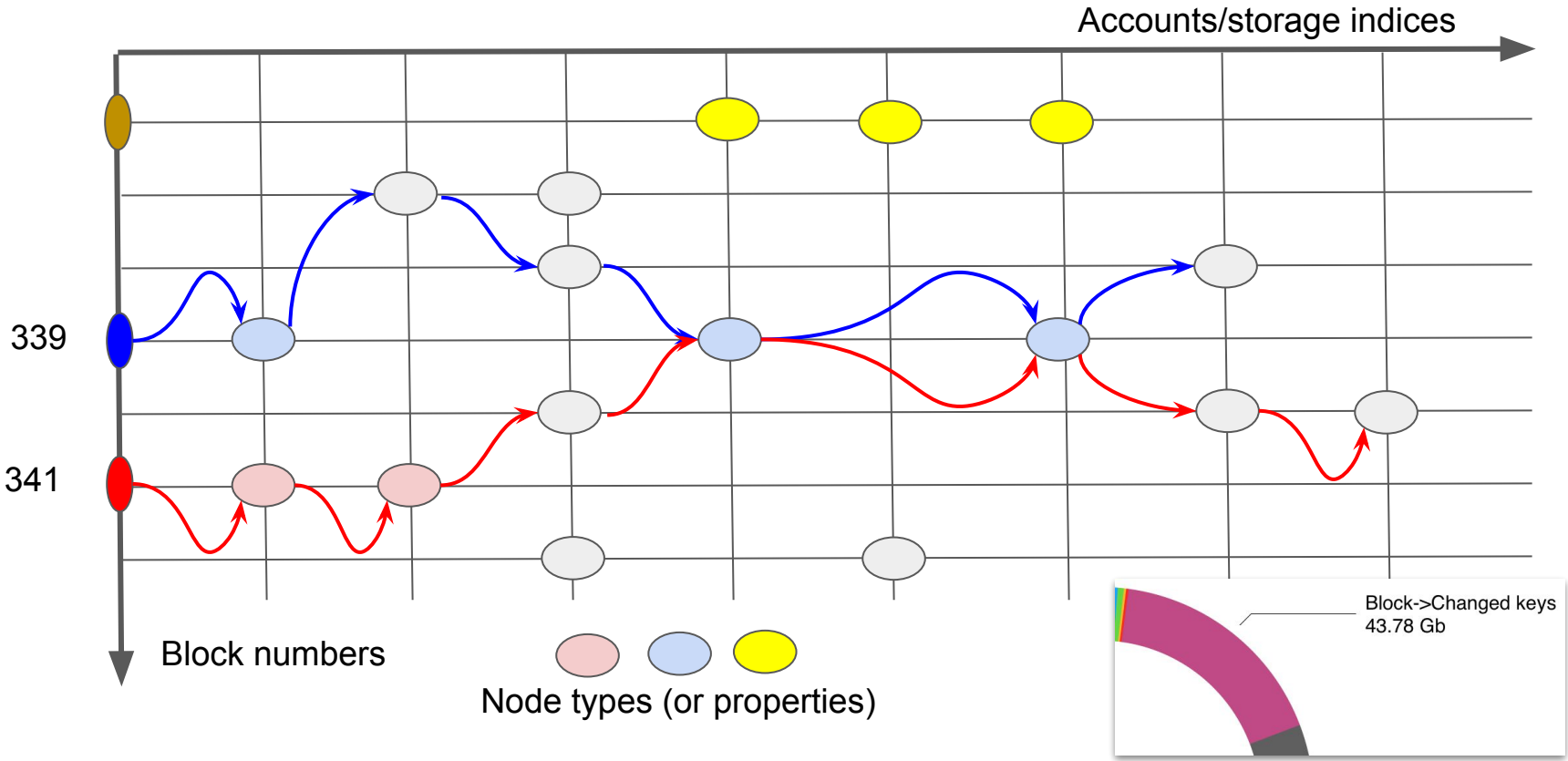




Future experiments - state history as a graph

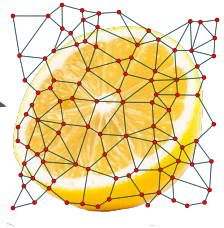


Future experiments - state history as a graph

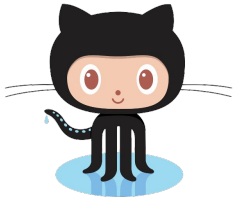


Lemongraph

Not the project logo!



Uses as underlying store



log **txnlog**

node_idx

edge_idx

prop_idx

srcnode_idx

tgtnode_idx

kv

scalar

scalarID → scalar (byte seq)