LMPTs:
Eliminating Storage Bottlenecks for Processing Blockchain Transactions

Jemin Andrew Choi [1], *Sidi Mohamed Beillahi* [1], *Peilun Li* [2], *Andreas Veneris* [1], *Fan Long* [1]

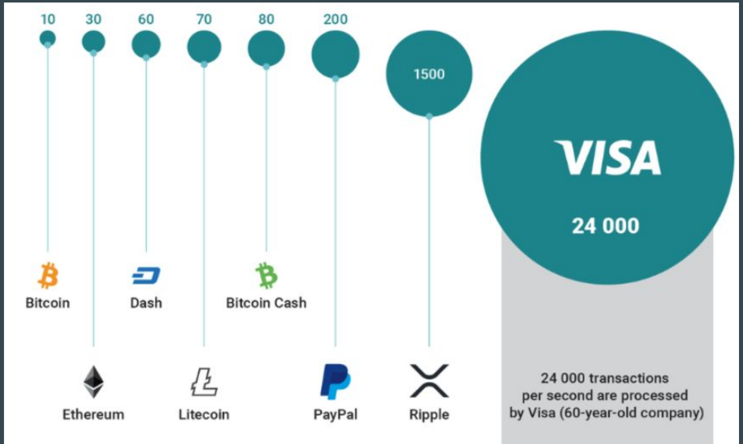
[1] University of Toronto [2] Shanghai Tree-Graph Blockchain Research Institute

ICBC 2022

Contents

Motivation

- Existing blockchain platforms have been (notoriously) slow
    - Lots of progress in consensus layer performance
- What's next?
    - Improve **transaction execution** and **storage layer** performance



https://www.datadriveninvestor.com/2018/10/05/moving-to-an-era-of-highly-scalable-blockchain-networks/

Prior Work

- Previously profiled Ethereum clients to examine biggest bottleneck after consensus

	Storage	Verifier	EVM	Other
ERC20	67.0%	25.9%	3.9%	3.2%
ERC721	73.5%	18.3%	5.7%	2.5%
ERC1202	73.1%	20.5%	3.6%	2.8%

Ao Li, Jemin Andrew Choi, and Fan Long. Securing smart contract with runtime validation. PLDI 2020

Prior Work

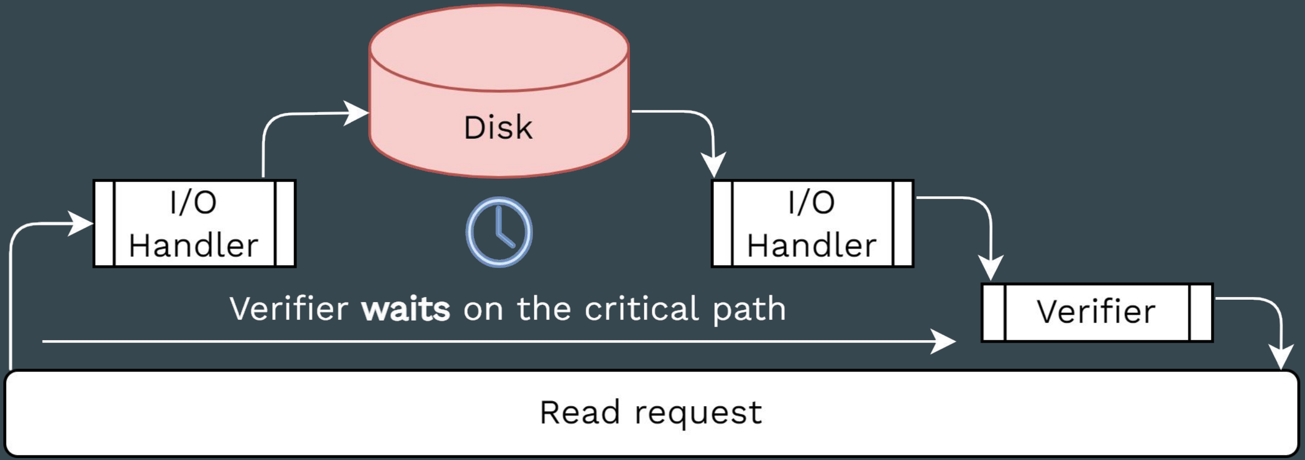
- Previously profiled Ethereum clients to examine biggest bottleneck after consensus
  - State updates on the blockchain are expensive
  - Storage layer is the next performance bottleneck

	Storage	Verifier	EVM	Other
ERC20	**67.0%**	25.9%	3.9%	3.2%
ERC721	**73.5%**	18.3%	5.7%	2.5%
ERC1202	**73.1%**	20.5%	3.6%	2.8%

Ao Li, Jemin Andrew Choi, and Fan Long. Securing smart contract with runtime validation. PLDI 2020

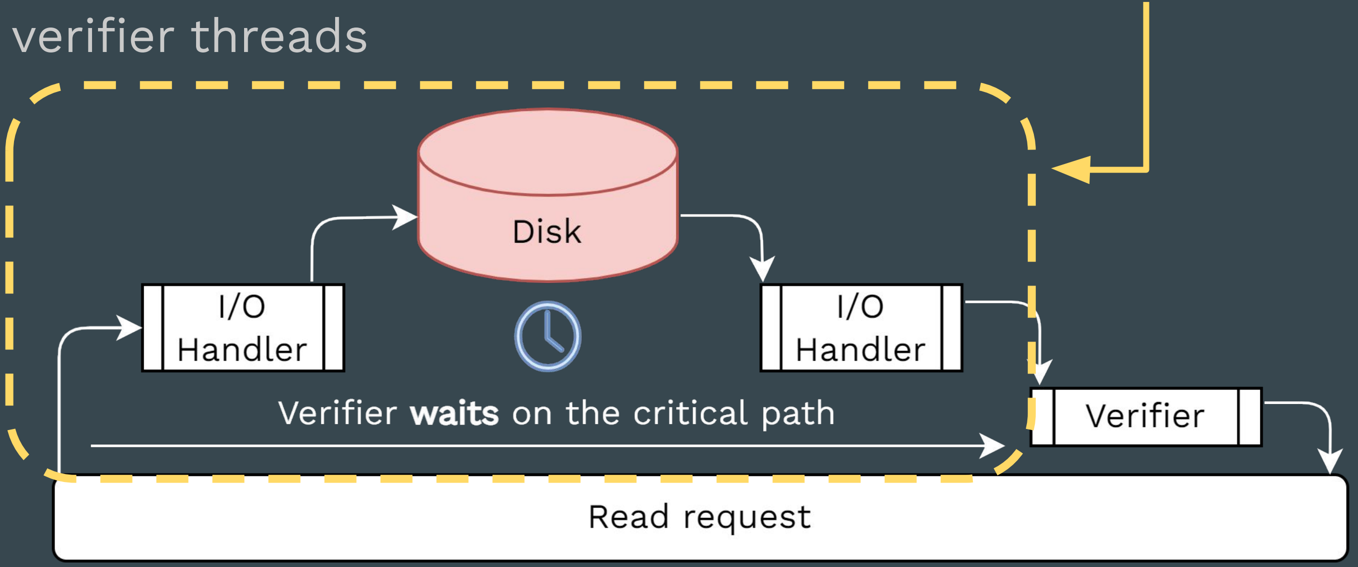
Profiling with **perf**

- Similarly, our findings with **perf** yielded consistent results with prior work
  - Sequential I/O operations are on the **critical path** and blocks verifier threads



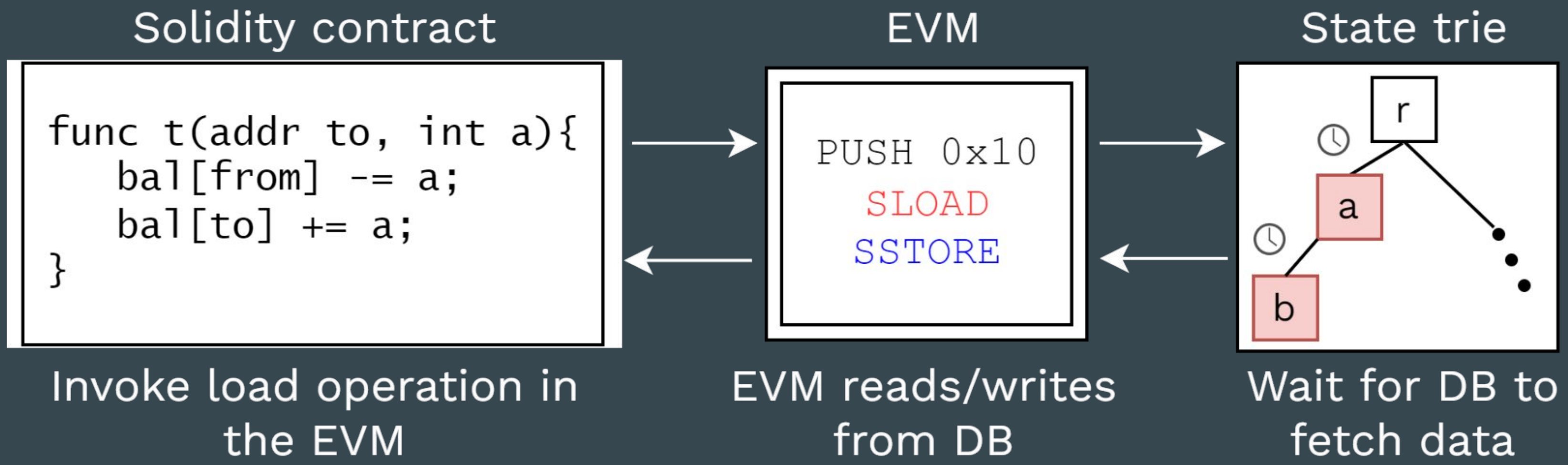
Profiling with **perf**

- Similarly, our findings with **`perf`** yielded consistent results with prior work
  - Sequential I/O operations are on the **critical path** and blocks verifier threads



Profiling with **perf**

- EVM opcodes such as **SSTORE** and **SLOAD** reduce performance
  - Since txn execution thread evaluates EVM sequentially, entire thread is slowed down by state accesses



Preview of LMPTs

- Increasing cache sizes has minimal impact on throughput

- Reading/Writing values from storage faster is imperative in improving transaction throughput

- **Contributions**:
  - **Faster** and **parallelized** authenticated storage structure
  - **x6 throughput** compared to existing EVM clients

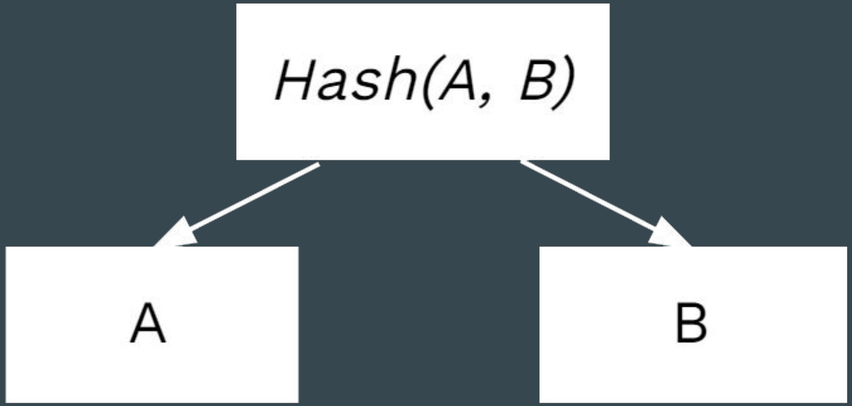
Contents

How Ethereum Stores State

- State trie is represented by a **Merkle-Patricia Tree (MPT)**
  - Using the `keccak256` hash of account address to access account information and state
  - Block headers store the Merkle root to check if state has been tampered
  - Light clients rely on partial Merkle proofs to verify state on chain through authenticated reads

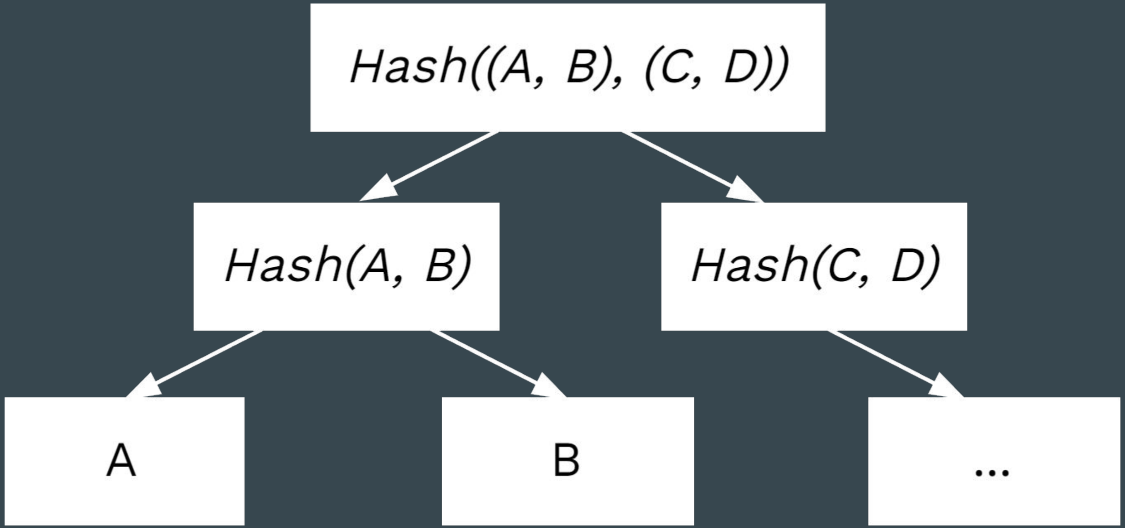
Merkle Trees

- Merkle Trees combine hashes of its children
- Clients can check hash to ensure the reliability of data
  - Authenticated reads in Ethereum



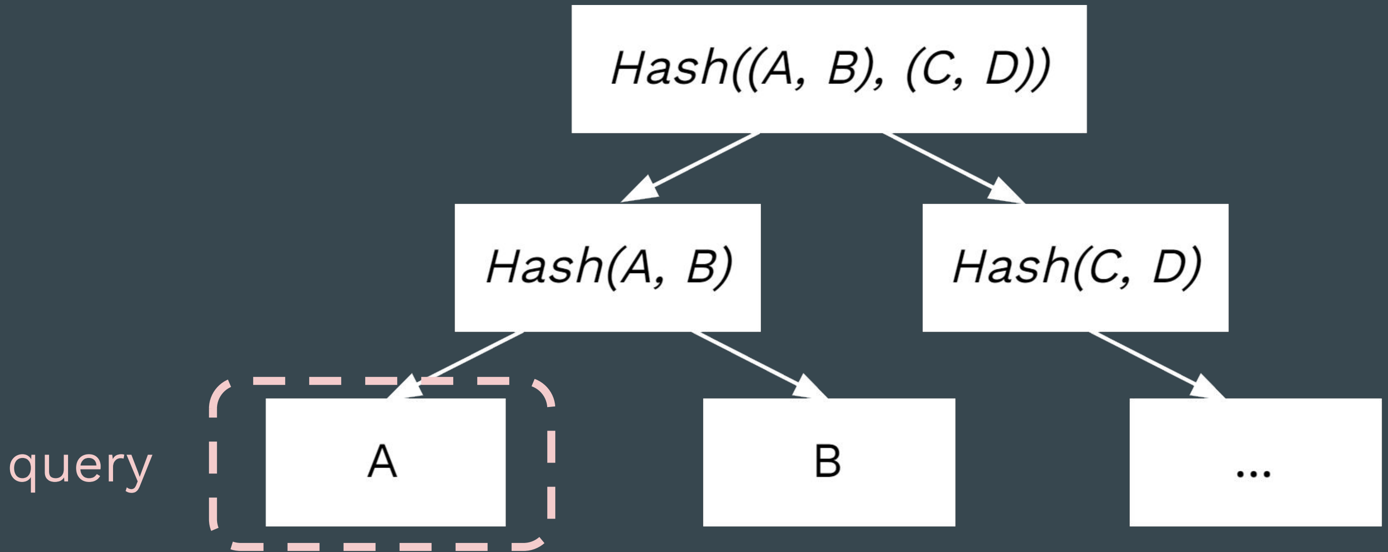
Merkle Trees

- Merkle Trees combine hashes of its children
- Clients can check hash to ensure the reliability of data
  - Authenticated reads in Ethereum



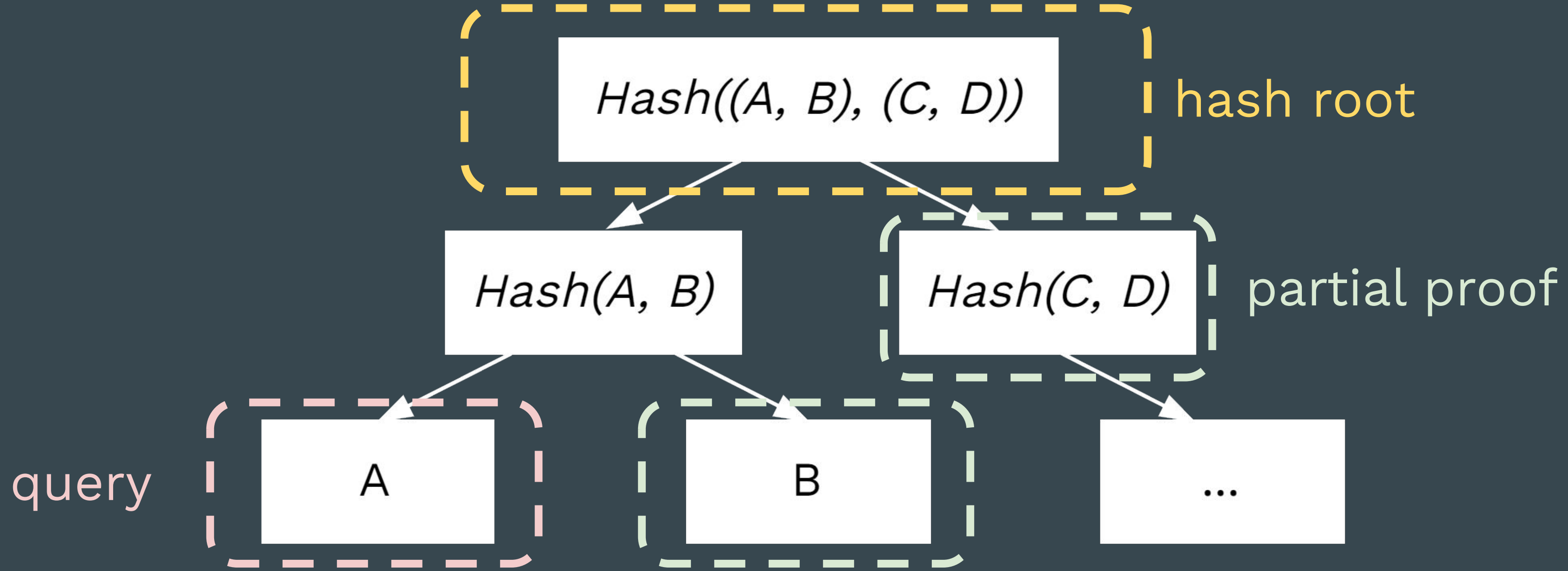
Merkle Trees

- Merkle Trees combine hashes of its children
- Clients can check hash to ensure the reliability of data
  - Authenticated reads in Ethereum



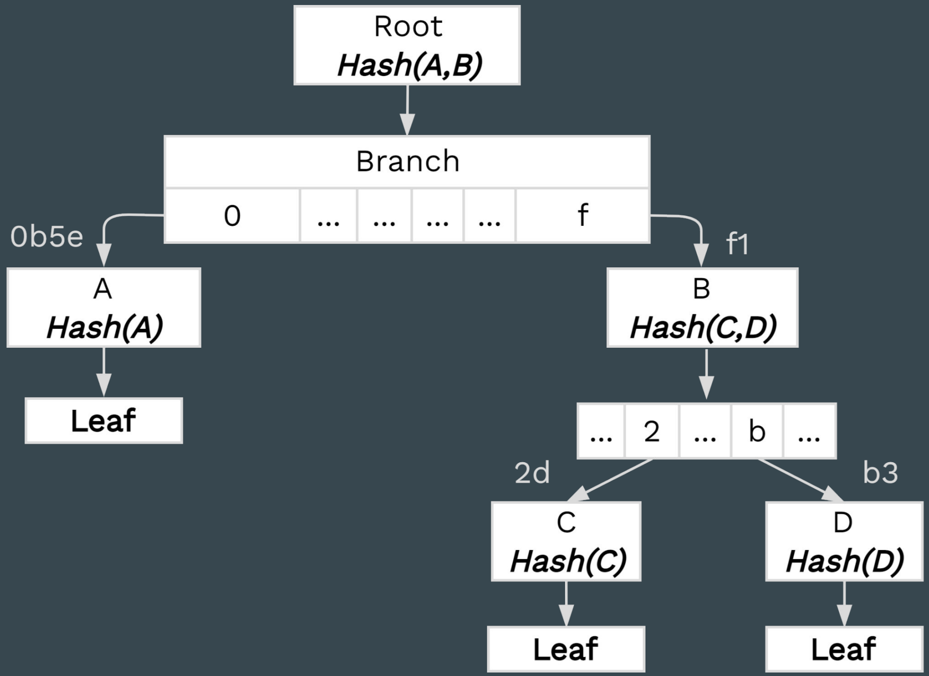
Merkle Trees

- Merkle Trees combine hashes of its children
- Clients can check hash to ensure the reliability of data
  - Authenticated reads in Ethereum



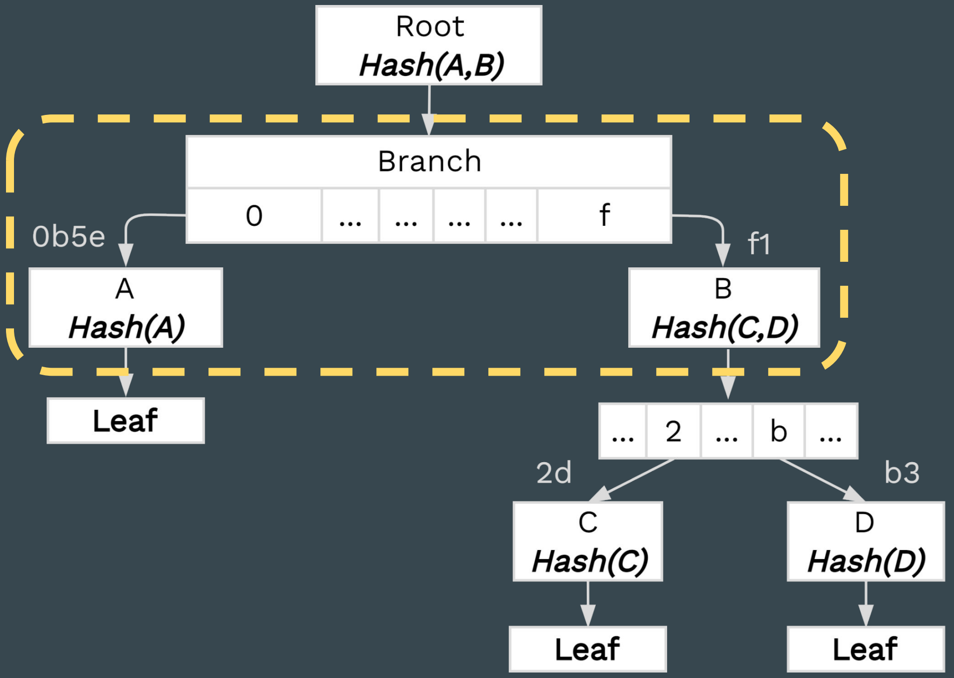
Merkle Patricia Trees (MPT)

- Merkle property & Compress nodes with shared hash sequence
  - Interior node contain hashes of its children
  - Leaf node contain account info (e.g. balance, nonce)
  - Root node contains hashes of all nodes in the tree



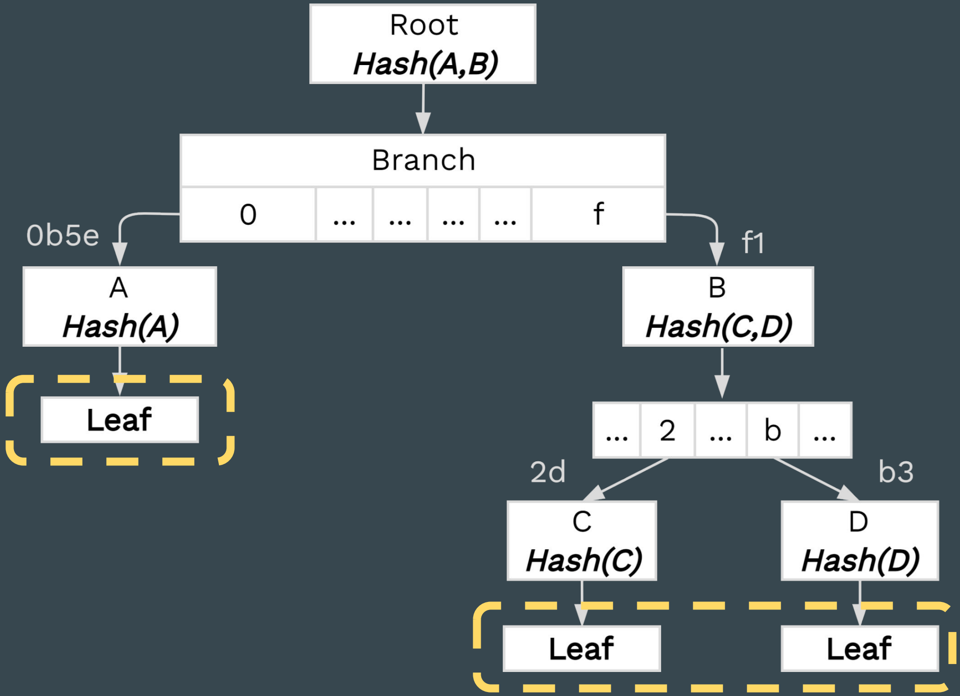
Merkle Patricia Trees (MPT)

- Merkle property & Compress nodes with shared hash sequence
  - Interior node contain hashes of its children
  - Leaf node contain account info (e.g. balance, nonce)
  - Root node contains hashes of all nodes in the tree



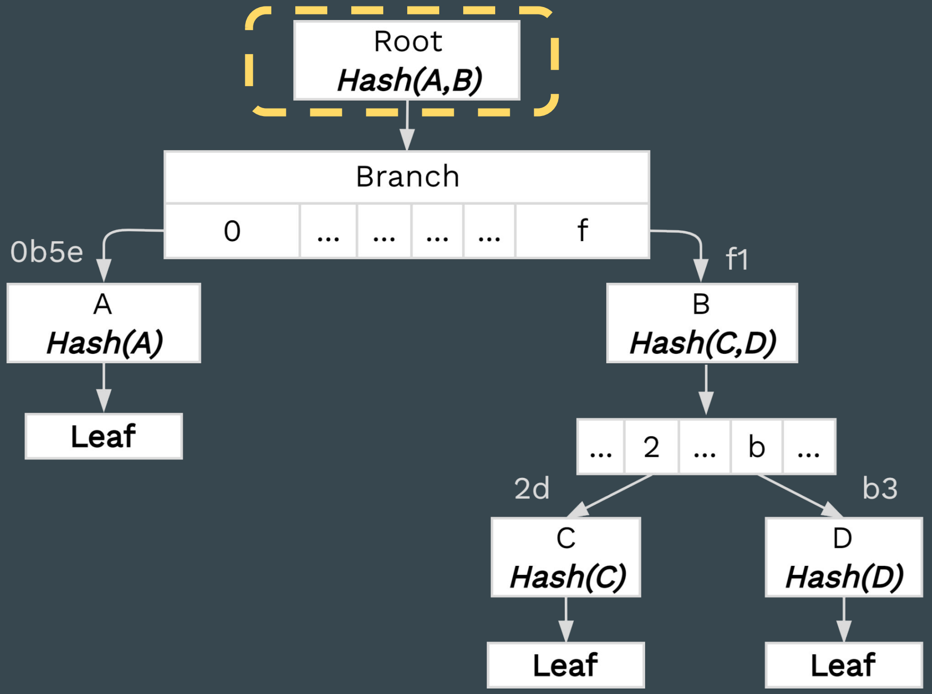
Merkle Patricia Trees (MPT)

- Merkle property & Compress nodes with shared hash sequence
  - Interior node contain hashes of its children
  - Leaf node contain account info (e.g. balance, nonce)
  - Root node contains hashes of all nodes in the tree



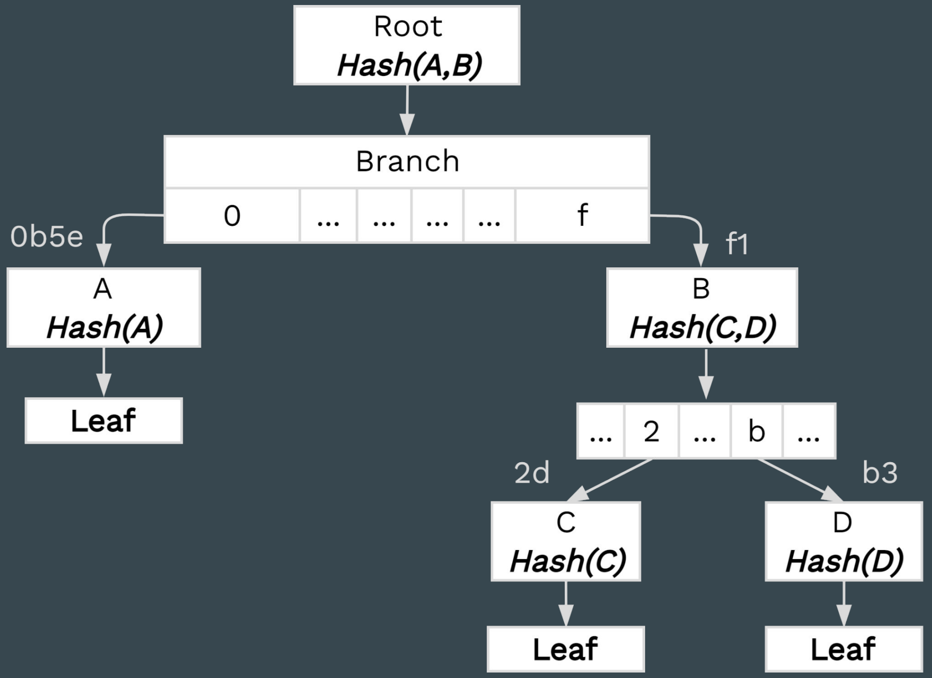
Merkle Patricia Trees (MPT)

- Merkle property & Compress nodes with shared hash sequence
  - Interior node contain hashes of its children
  - Leaf node contain account info (e.g. balance, nonce)
  - Root node contains hashes of all nodes in the tree



Merkle Patricia Trees (MPT)

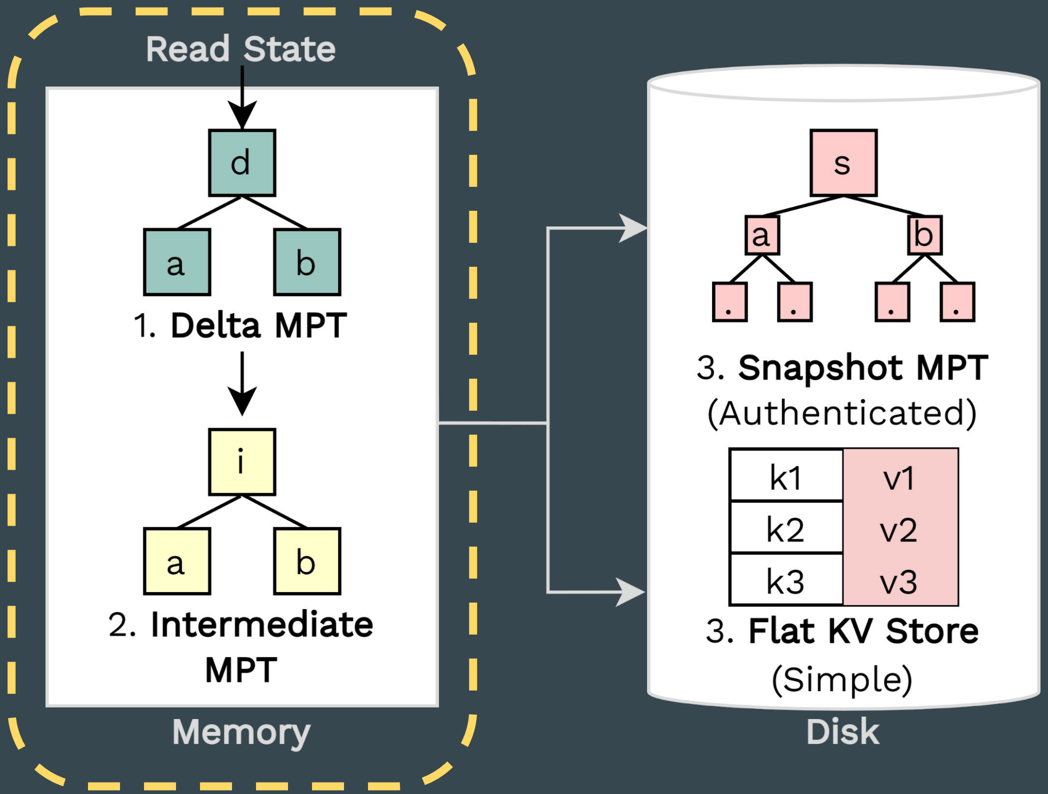
- Merkle property & Compress nodes with shared hash sequence
  - Path: 32-byte hash of addr
  - Each node is 1 db read
  - <u>**x64**</u> read amplification!



Contents

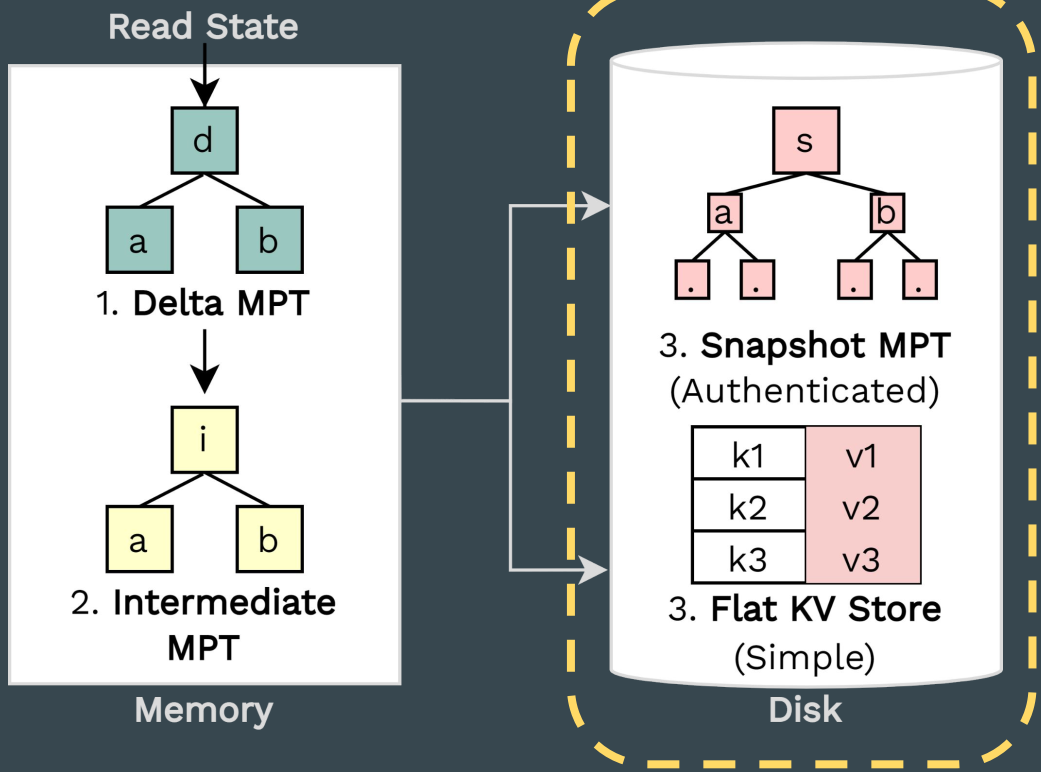
Overview

- Reduce read & write amplification by storing hot data on smaller MPTs that sit in memory ("**Layered**" MPT)



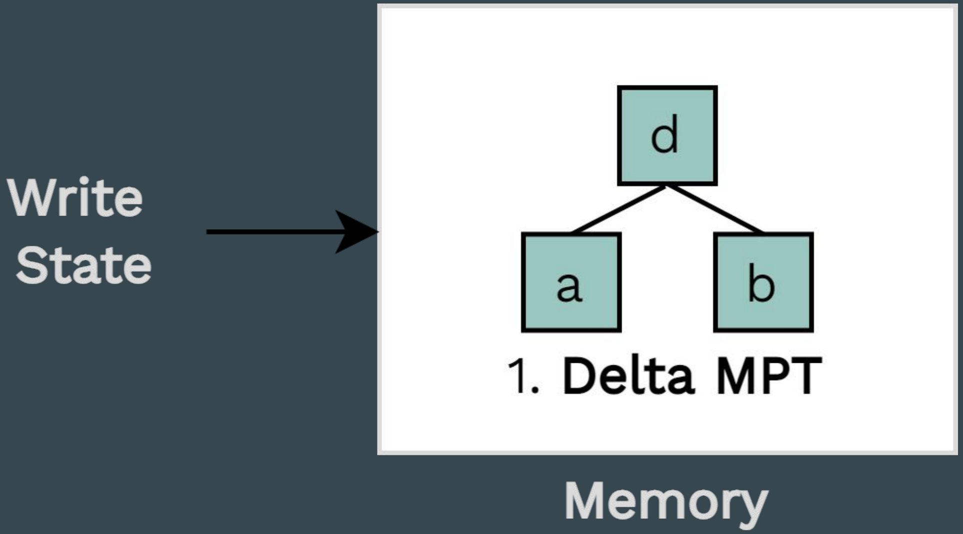
Overview

- Don't query disk unless absolutely necessary
- Have a separate K-V store for non-authenticated reads
- Parallelize execution by flushing updates to disk in background



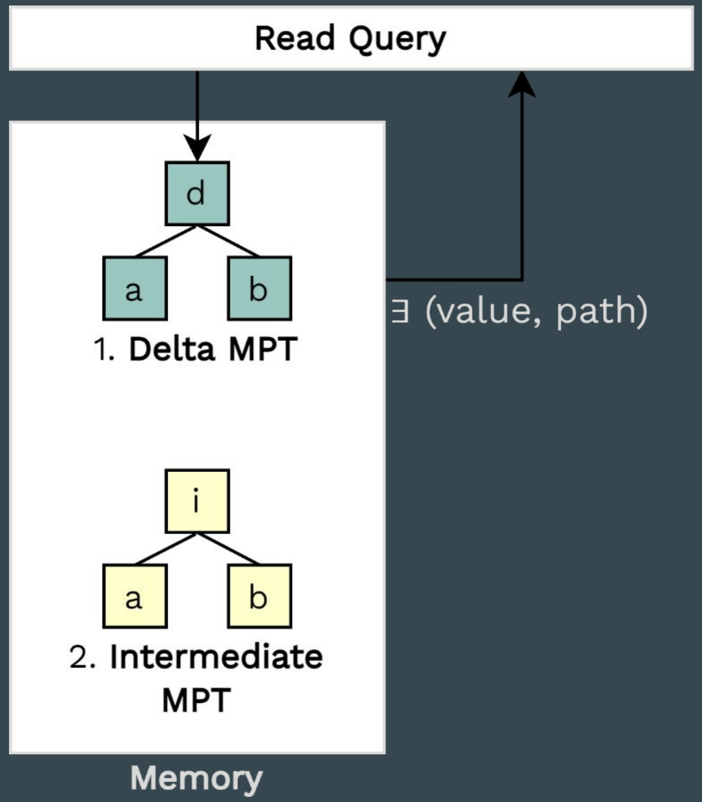
Design: Read and Writes

- Writes
  - Always updated on smallest **delta trie** (fast & frequent data)



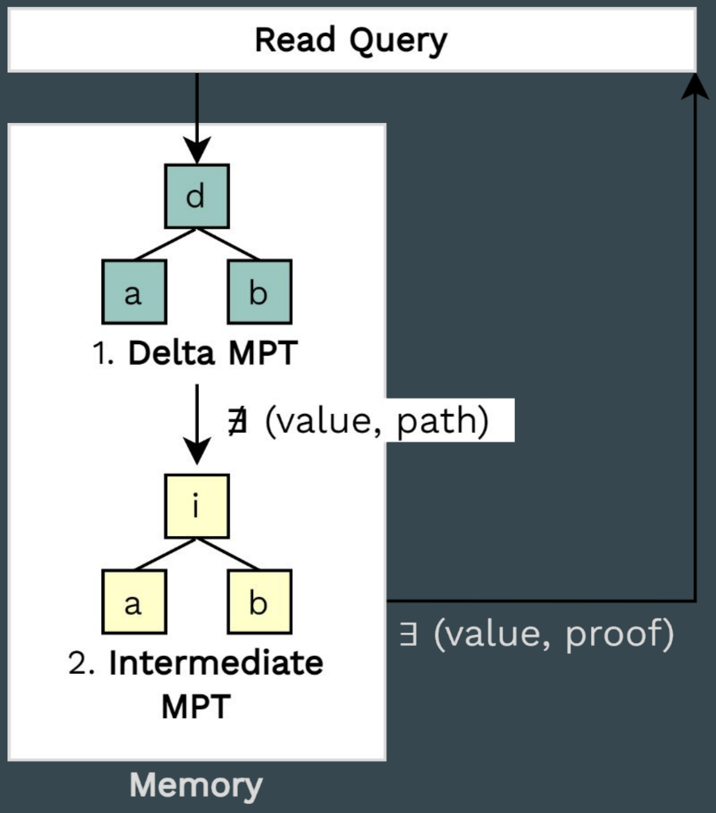
Design: Read and Writes (cont.)

- Reads
  - First, query **delta trie** and if it exists, return the value and path



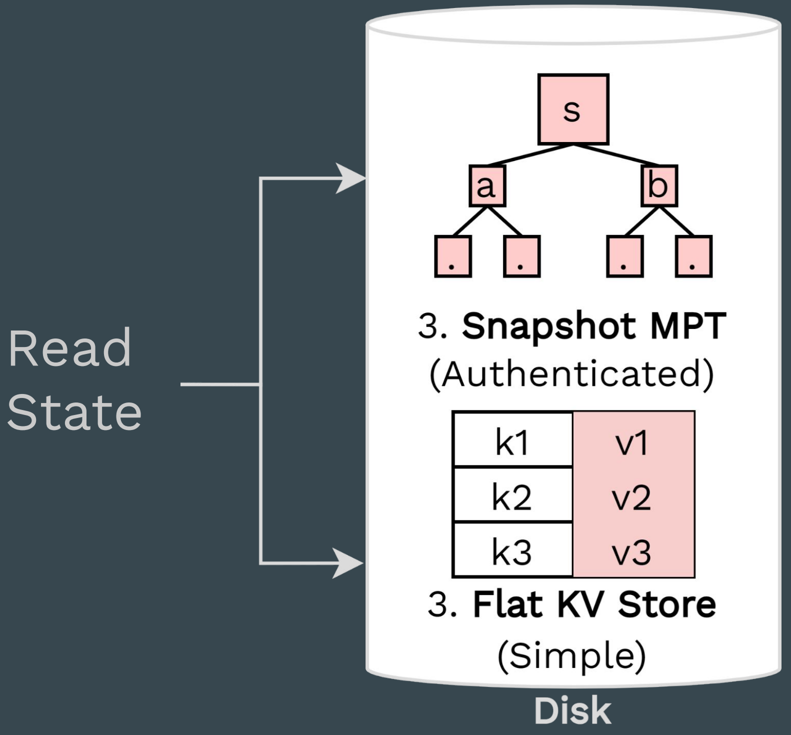
Design: Read and Writes (cont.)

- Reads
  - If miss, need to return proof that two immediately adjacent paths exist in the tree instead.
  - Then, query the **intermediate trie**, and if it exists, return the value and combined proof of the **delta** + **intermediate trie**



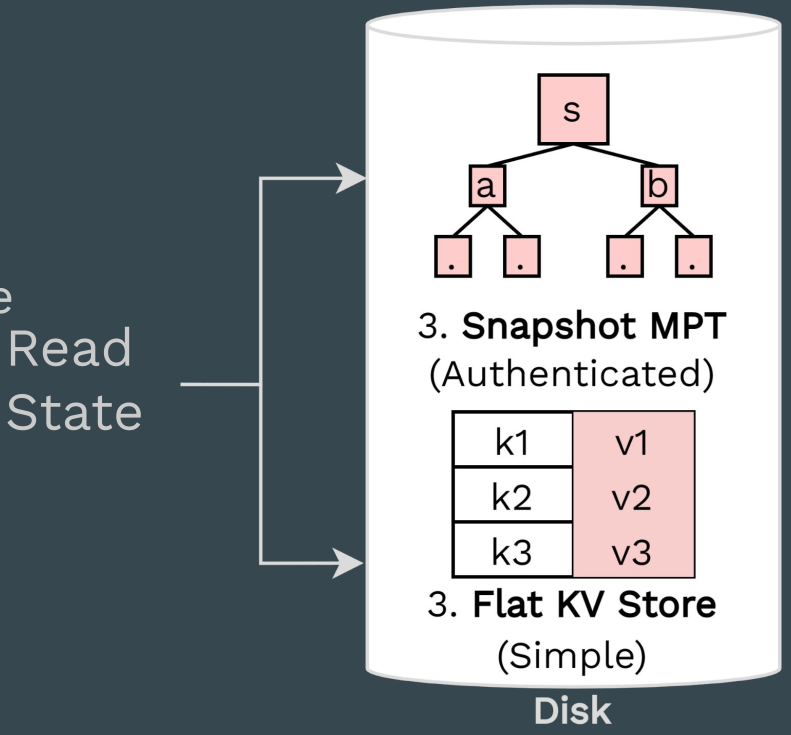
Design: Read and Writes (cont.)

- Reads
  - If the key is not in the **delta** or **intermediate** trie, query **disk**



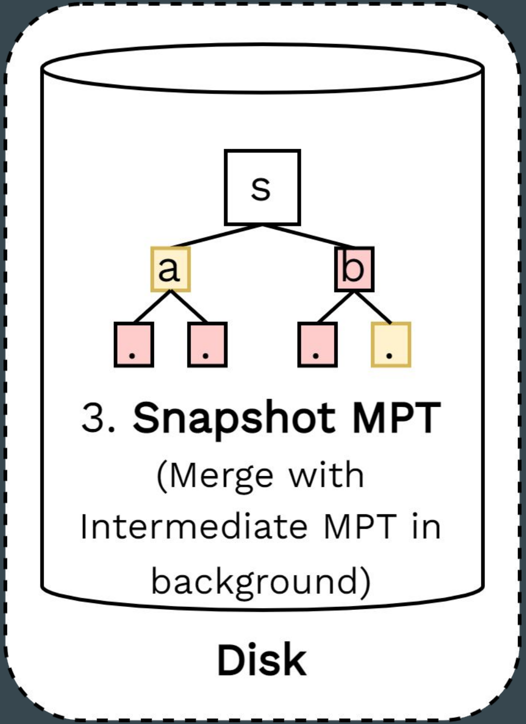
Design: Read and Writes (cont.)

- For authenticated reads, read the **snapshot trie**
- If client trust the authenticity of data (e.g. reading from its own disk), then look up the value in the **flat k-v map**
- Delay costly disk accesses and reduce read amplification on bigger tries



Design: Trie Merging Operations

- `merge_compute()`
  - Periodically, **intermediate trie** changes are merged to **snapshot trie** and **flat k-v store** on disk
  - Reduce write amplification by updating smaller portion of trie



Background Thread

Design: Trie Merging Operations (cont.)

- `merge_update()`
  - Flush changes in smaller tries to bigger tries
  - Set **snapshot trie** to output of `merge_compute()`
  - Set **intermediate trie** & **root** to **delta trie**
  - Initialize new trie & root for **delta trie**

Design: Trie Merging Operations (cont.)

- At predefined “**merge intervals**”, call merge_update() to flush changes to tries
- In a background thread, call merge_compute() to batch disk operations in parallel to the main execution thread

```
if block_cnt % merge_interval == 0:
    Wait for last spawned thread to end
    merge_update(Trie, root, flat)
    spawn_thread(root, flat = merge_compute(Trie))
```

Design: Integration with EVM

- **LMPT** has the advantage of being easily integrated with EVM-based systems with the following modifications
  - Use hash functions like `keccak256` to combine root hashes of the smaller tries to generate a single 32-byte root hash of state
  - Since authentication proof requires a combination of proofs from tries, modify the **verifier module** to accept a combination proof

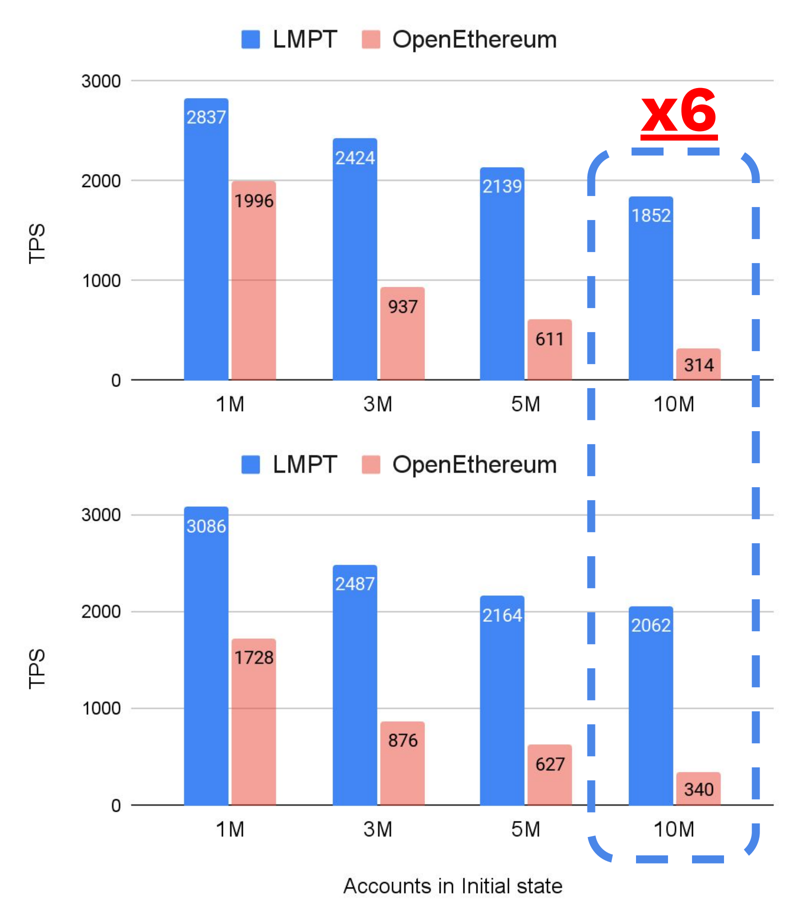
Contents

LMPT Evaluation

- Modified `OpenEthereum` client to integrate **LMPT**
  - Turn off the consensus engine to compare storage layer
- Ran experiments on 500,000 transactions on AWS EC2
  - Tested on Simple payment and ERC20 transfer transactions
  - Sample trace from real Ethereum network
  - Randomly send transactions with uniform distribution
- Set initial states with increasing number of accounts in genesis block
  - Increase storage performance workload for larger initial state

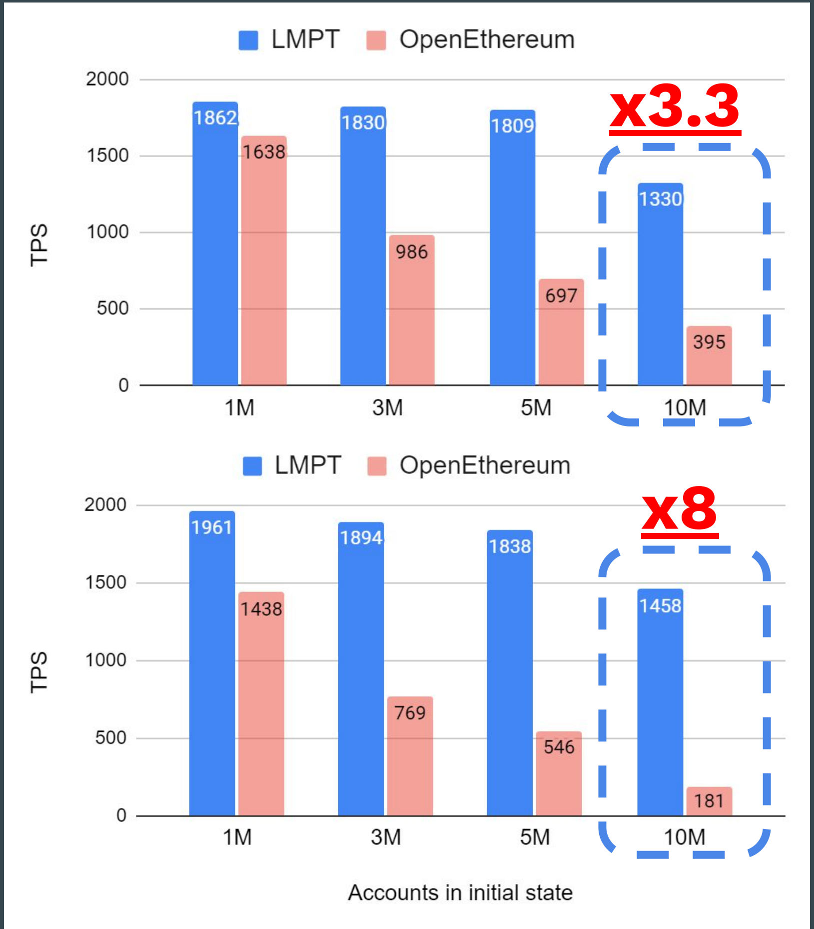
LMPT Evaluation

- Simple payments
  - Ethereum Trace on top
  - Random senders on bottom
- Upto **x6** throughput on LMPT client vs regular client with MPT
  - After 20M, regular client fails to make much progress, whereas LMPT can handle larger state



LMPT Evaluation

- ERC20 transfers
- **x3~8** throughput on LMPT client vs regular client
- LMPT is effective in smart contract data accesses



Contents

Conclusion

- As consensus bottleneck is removed and blockchain state grows, storage layer performance will be critical in high-throughput ledgers

- **LMPT**s allow faster access to hot data and enable I/O operations to be decoupled from the critical path

- Easy to integrate with existing EVM systems