SlimArchive: A Lightweight Architecture for Ethereum Archive Nodes

Hang Feng[1], Yufeng Hu[1], Yinghan Kou[1], Runhuai Li[2],
Jianfeng Zhu[2], Lei Wu[1], and Yajin Zhou[1]

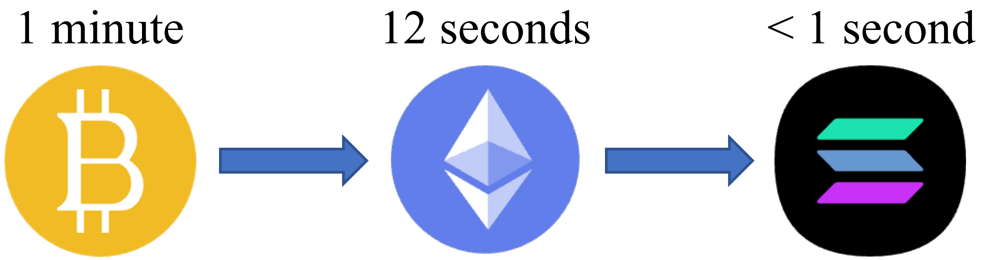
[1]*Zhejiang University*
[2]*BlockSec*

Blockchain Optimization

Performance & Scalability

Consensus Protocol

Faster Consensus



Execution Machine

Parallel Execution

Pipelined Blockchain

Smart Contract JIT

Storage Engine

Sharding / Replica

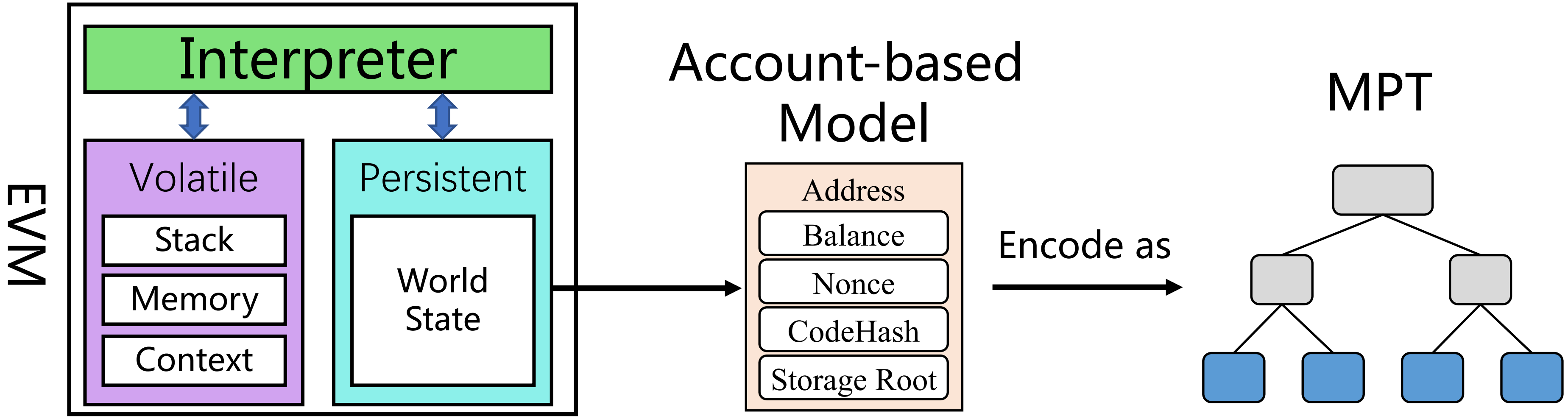
State Model Optimization

This work

Data Compression

Ethereum Storage Layer

- Ethereum: transaction driven **state machine**
- Account-based state model
  - Identified by address (pub-key)
  - Account may have storage, referenced by its storage root
- States are encoded as **Merkle Patricia Tries (MPTs)**, a.k.a. world state trees



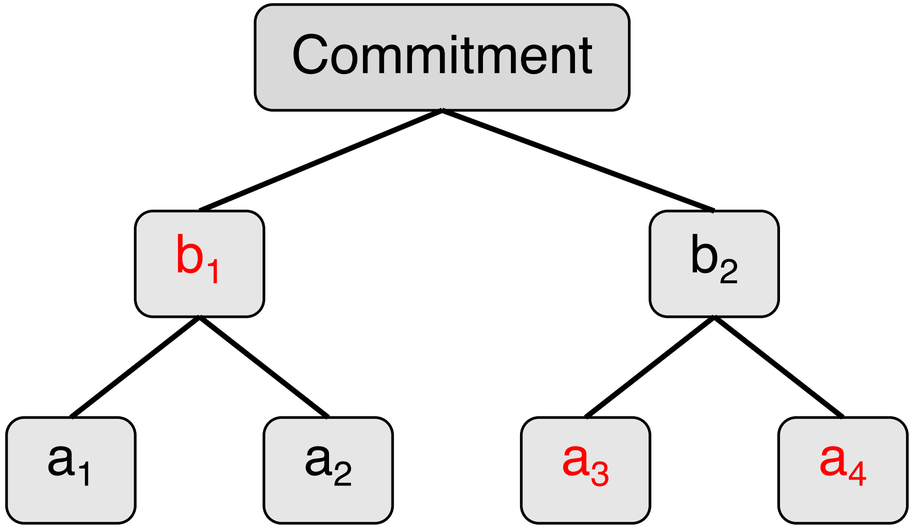
MPT

- Ethereum Merkle Patricia Trie—16-radix Merkle tree

- Merkle tree: a **vector commitment protocol**
- Data are stored in leaf nodes
- Hash pointers link parent and children

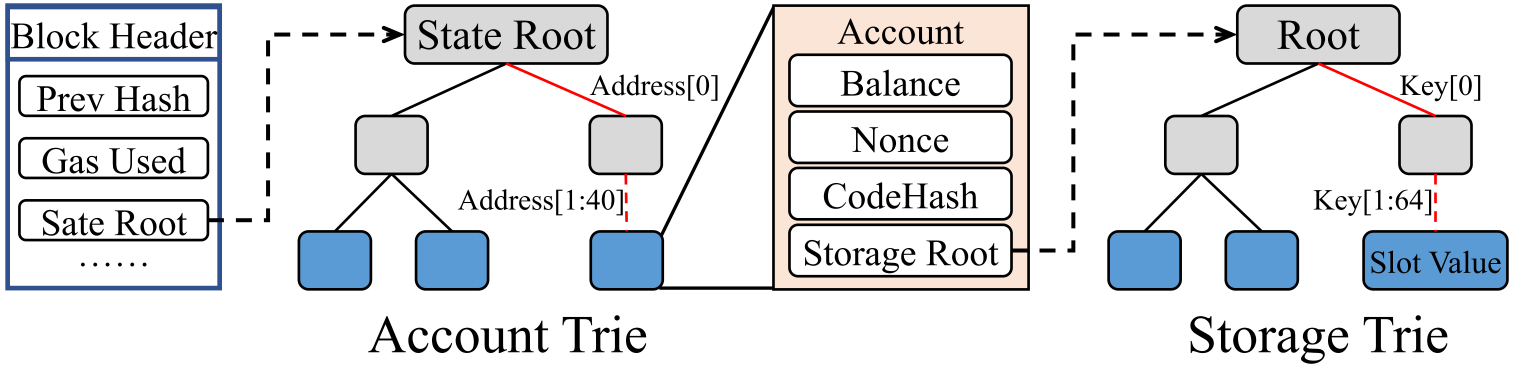
  $b_1 = h(a_1 \| a_2)$ $b_2 = h(a_3 \| a_4)$ $\text{Commitment} = h(b_1 \| b_2)$

- Efficient data authentication, to verify a_3:
  - Prover provides: b_1, a_3, a_4
  - Verifier validates: $\text{Commitment} = h(b_1 \| h(a_3 \| a_4))$



Ethereum World State Tree

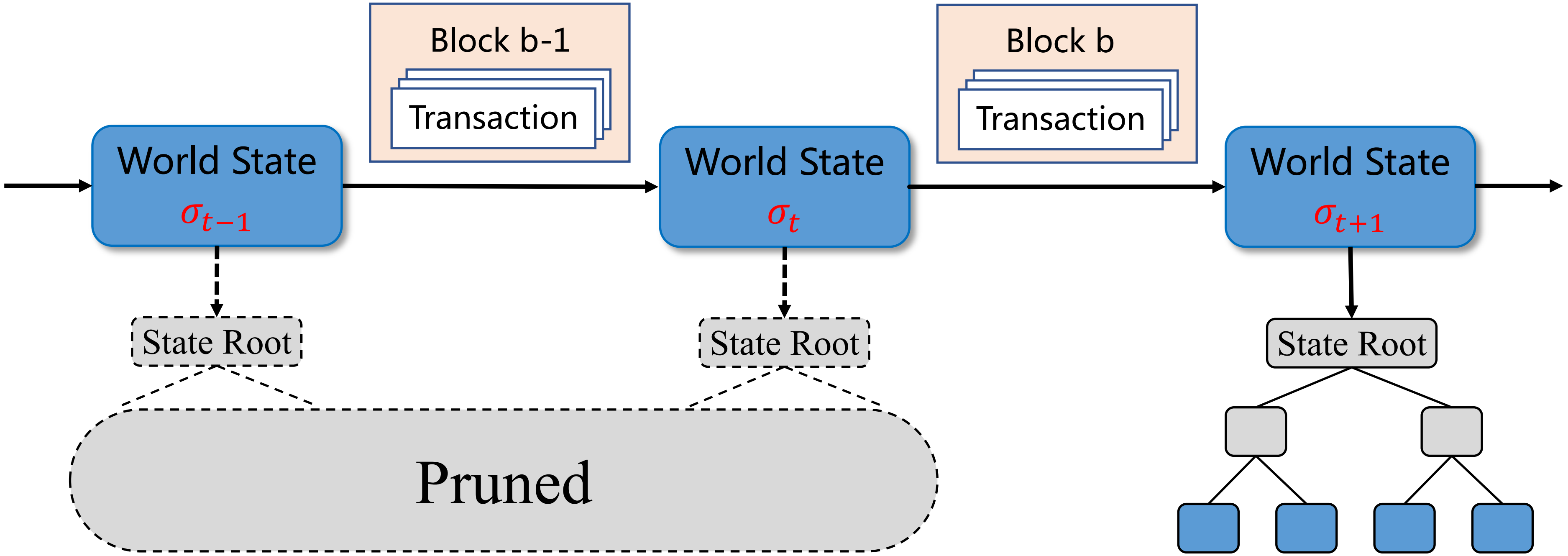
- Ethereum MPT
  - 16-radix account & storage tries
  - State trees are updated **per block**



  - State validation during synchronizing the latest blocks
  - **Data authentication**
    - Used by light nodes (ONLY have state roots) when querying states from untrusted remote nodes

Light/Full Node

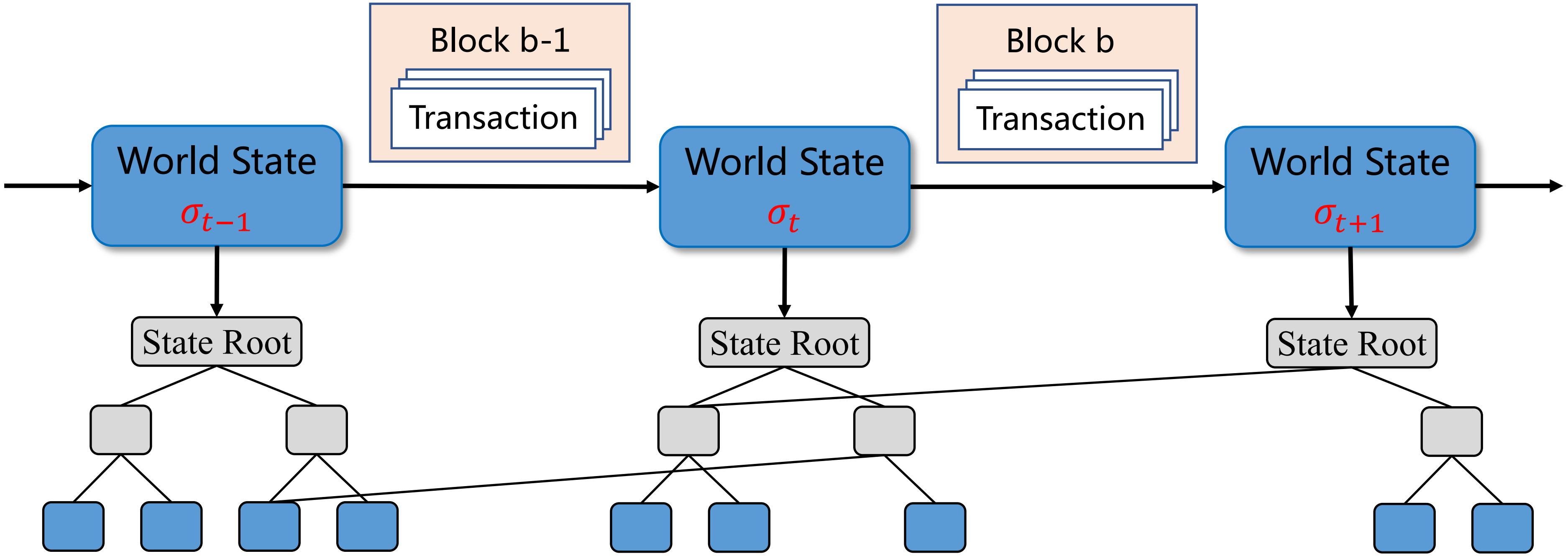
- Light node: does not maintain any states
- Full node: maintains **ONLY** the **latest** world state
  - Historical states are pruned



Archive Node

- Maintains **ALL** historical states
  - The MPT at each block is saved

- Requires more disk resource



The Importance of Archive Node

Q: Why do we use archive nodes?

A: For **testing** and **analyzing** smart contracts and transactions

- ◆ **Abilities of archive node**
  - ✓ **Access to historical states**
  - ✓ **Profiling historical transactions**
    - ● **Data/control flow analysis of a transaction execution**
  - ✓ **Simulating transactions at a historical time point**

The Importance of Archive Node

Q: Why do we use archive nodes?

A: For **testing** and **analyzing** smart contracts and transactions

- **Usage for academia**
  - **Detecting attack transactions and smart contract vulnerabilities**
    - Demystifying defi mev activities in flashbots bundle, *CCS 23*
    - Your exploit is mine: Instantly synthesizing counterattack smart contract, *USENIX Security 23*
  - **Smart contract fuzz testing**
    - Detecting state inconsistency bugs in dapps via on-chain transaction replay and fuzzing, *ISSTA 23*
  - **Quantitative/arbitrage strategies back-testing**
    - Cyclic arbitrage in decentralized exchanges, *WWW 22*
    - A large scale study of the ethereum arbitrage ecosystem, *USENIX Security 23*
  - **Blockchain temporal research**
    - Temporal analysis of the entire ethereum blockchain network, *WWW 21*
  - And more …

The Importance of Archive Node

Q: Why do we use archive nodes?

A: For **testing** and **analyzing** smart contracts and transactions

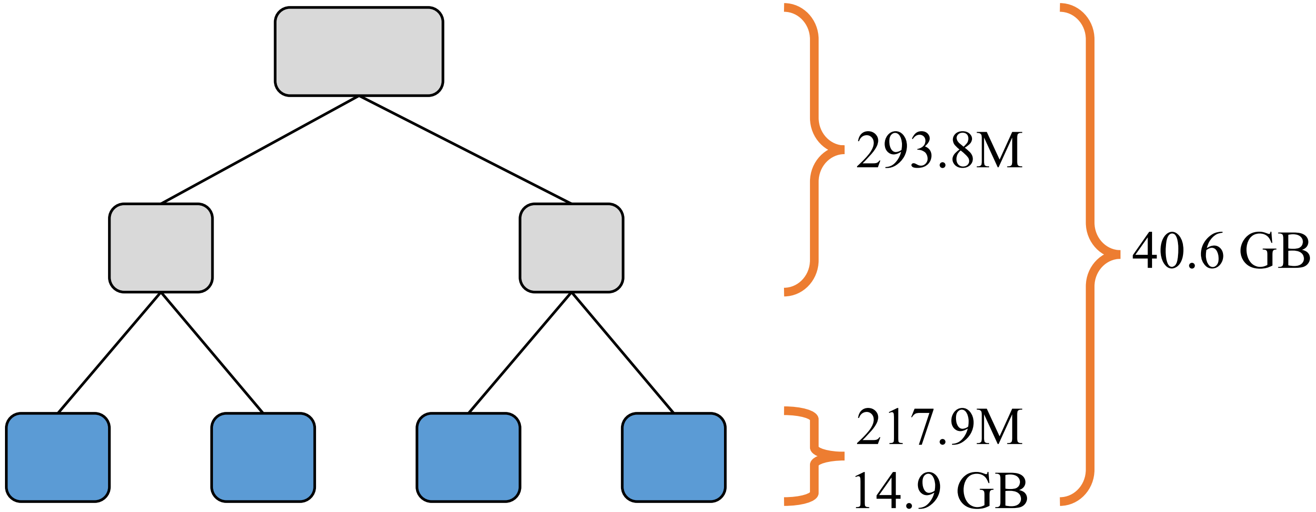
- **Usage in industry**
  - DeFi's developments make **transaction's complexity increasing**
  - Today, users need to **dive into their transactions to better understand the logic**
  - Many infrastructure service providers release products for **debugging and analyzing** historical transactions
    - BlockSec, Tenderly …

Problems

- **Performance and scalability**
  - **Storage exploding**
    - Full node size: ~ 1.1 TB
    - Archive node size: ~ 18.0 TB
  - **Low access throughput**
    - State access consumes the majority of the transaction execution time

Root Cause 1

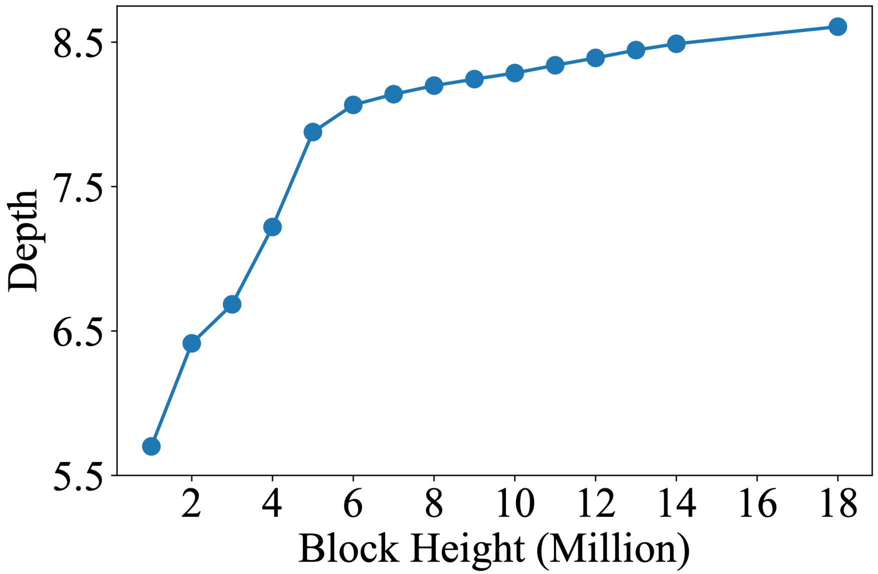
- **Inefficient MPT**
  - **Excessive intermediate data**



The state trie at block height 18M
Storage utilization: 36.7%

Root Cause 1

- **Inefficient MPT**
  - **Read/write amplification**
    - Time complexity: O(log n)
    - **Average depth: 8.6**
    - Each state access is amplified to an average of **8.6 database operations**



The average depth of state tries at different block heights

Solution 1

- **Replace the MPT**
  - **The usage of MPT in Ethereum**
    - State validation & data authentication
  - **For state validation:**

Historical states become immutable after synchronization

↓

Validation of historical states is not required

Solution 1

- **Replace the MPT**
  - **Is data authentication for historical states necessary?**
  - **In most real-world scenarios: No!**

1. Merkle proofs are rarely used in current ecosystem
2. Blockchain nodes are considered trusted by users in most scenarios

  - **Furthermore**
    - Archive nodes are primarily used for **testing** and **analytical** purposes
    - **Performance** is more critical
    - Data authentication carries a **high price (No matter how you optimize the DA)**

Solution 1

- **Replace the MPT**

Data authentication of historical states is not required in most real-world usage scenarios

↓

MPT structure is not necessary

↓

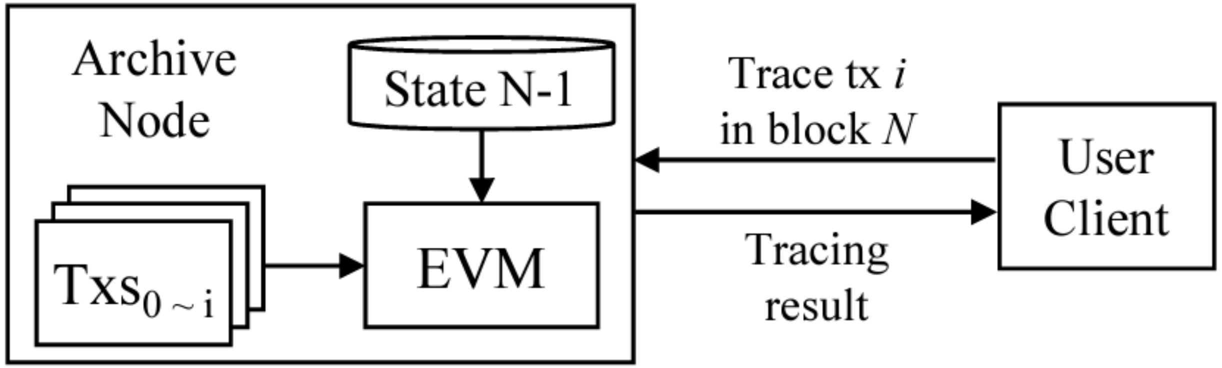
Employ a compacted and flattened data model to minimize intermediate data and simplify state access

✓ Trade off between DA and performance/cost-effectiveness

A solution tailored for most real-world scenarios

Root Cause 2

- **Coarse-grained state granularity**
  - **Block-level world state**
    - The granularity of historical states is **a block**
  - **Intra-block (transaction-level) state fetching**
    - Requires re-executing all txs before the target transaction



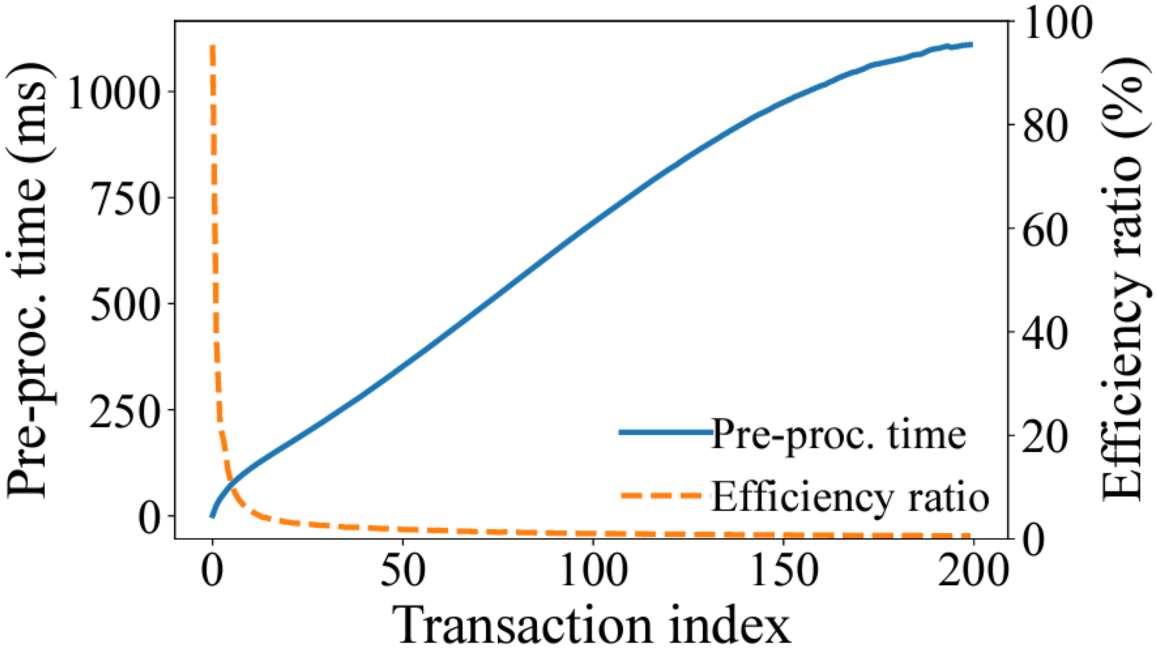
The execution from tx 0 to i−1 is ***pre-processing***
Only the execution of transaction i is ***effective***

Root Cause 2

- **Coarse-grained state granularity**
  - **Transaction execution efficiency**

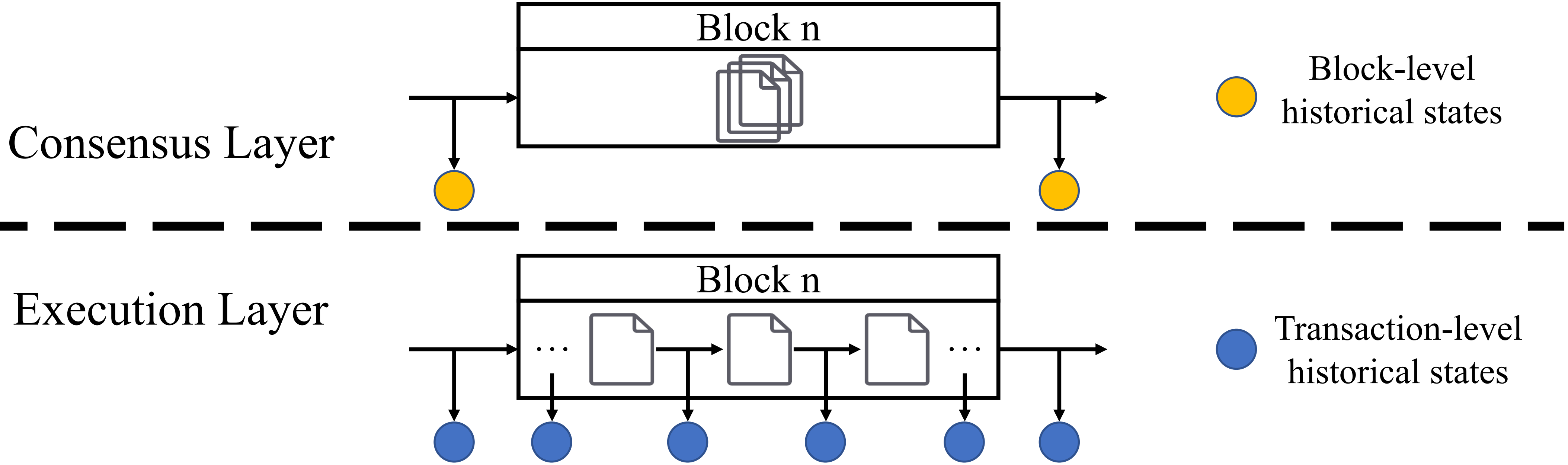
$$efficiency\_ratio = \frac{effective\_execution}{pre\_processing + effective\_execution}$$

- Pre-processing cost exceeds 1s
- Efficiency ratio is near zero



Solution 2

- **Refining the granularity**
  - **Decoupled** state transition granularity
    - Consensus layer: block
    - **Execution layer: transaction**



Block n

Consensus Layer

● Block-level historical states

Block n

Execution Layer

● Transaction-level historical states

Solution 2

◆ **Transaction-level historical states**

The granularity of state transition at the low-level execution layer is a transaction

↓

Refine the granularity of historical states to a transaction to eliminate the overhead caused by the pre-processing

SlimArchive Design

Objectives

- ✓ Lightweight
- ✓ Flexible
- ✓ High-performance

Properties

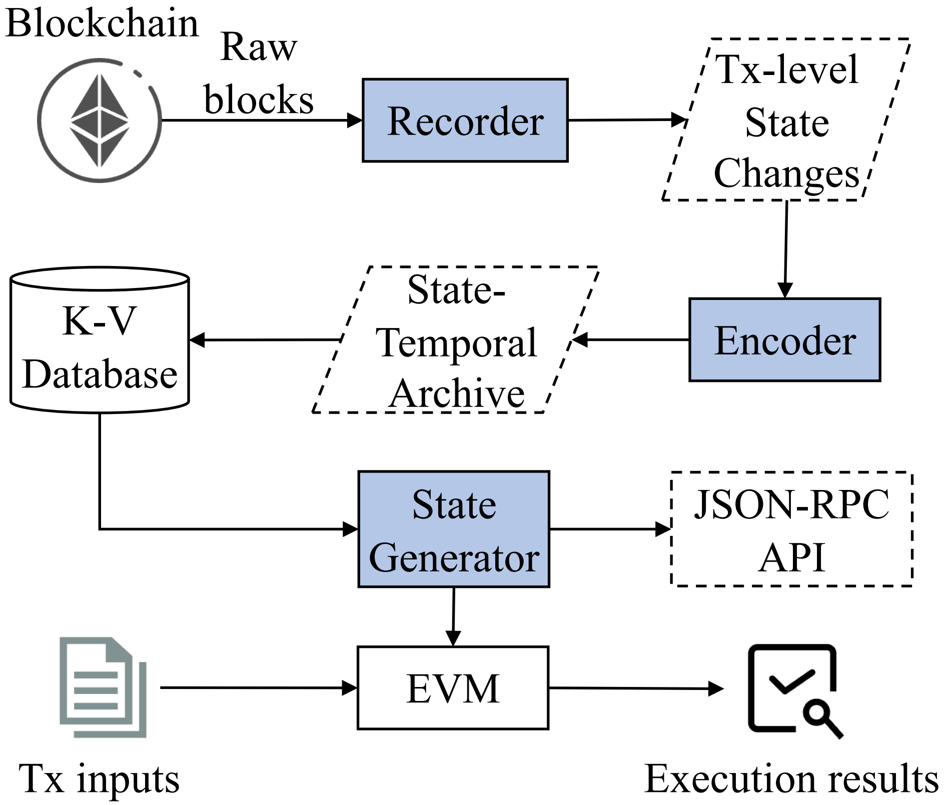
- ➢ **Flattened** state model that simplifies state access
- ➢ **Compacted** data storage that reduces intermediate data
- ➢ **Fine-grained** state granularity that eliminates computation overhead

Methodology

Flattening the minimum state changes of each transaction required for the world state

SlimArchive Overview

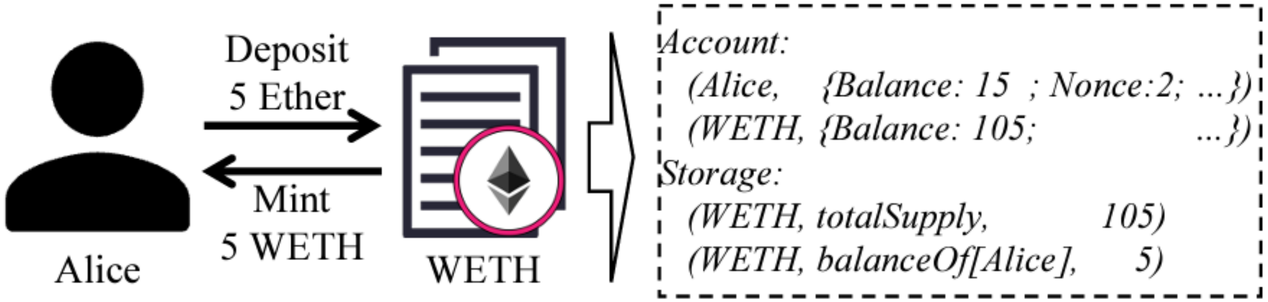
- **Recorder**
  - An instrumented EVM
  - Collects state changes of each transaction
- **Encoder**
  - Encodes state changes as *state-temporal archive,* a *flattened* representation of *transaction-level* historical states
- **State Generator**
  - Recovers historical states
  - Provides query interfaces for EVM and users



SlimArchive workflow

Recorder

- **Transaction-level state change collection**
  - **What** to collect
    - Temporal data, and post account/storage states
    - **Ignore** authentication data and runtime data
  - **Where** to collect
    - Normal/virtual transaction
  - **How** to collect
    - R/W set tracking
    - De-duplication



Account:
(Alice, {Balance: 15 ; Nonce:2; ...})
(WETH, {Balance: 105; ...})
Storage:
(WETH, totalSupply, 105)
(WETH, balanceOf[Alice], 5)

Example: state changes of a WETH deposit transaction

Encoder

◆ State-temporal archive

- Each state changed is encoded as a k-v pair, with three parts:

$$\boxed{StateKey}\boxed{TemporalKey} \rightarrow \boxed{StateValue}$$

- State Key: **which** state is changed

$$StateKey = Append(StateFlag, StateID)$$

- Temporal Key: **when** the state change occurred

$$TemporalKey = Append(BlockNumber, TransactionIndex)$$

- State Value: **what** the state is after the transaction

◆ Flattened historical states

- Key aligned
- **Partially chronological order: each entity's state changes are placed chronologically**

Encoder

◆ State-temporal archive

State Type	Key					Value
	State Key			Temporal Key		
Account	Account Flag	Address		Block Number	Tx Index	RLP(SlimAccount)
Deleted	Deleted Flag					Empty String
Storage	Storage Flag	Address	Slot Key			Slot Value
Code	Code Flag	Code Hash		N/A		Contract Bytecode

SlimAccount

- Balance
- Nonce
- CodeHash

State Generator

- **Fetching historical states**

Querying the state at a specific time point

Seeking the last state change before that time

Lower Bound: *StateKey*
Upper Bound: *Append(StateKey,TemporalKey)*
Seek the last change with key in [lower, upper)

Evaluation

- **Baselines:**
  - Geth
  - Erigon

- **Workloads:**
  - Real-world Ethereum transactions and states

Evaluation

- **Synchronization**
  - **Time spent on generating historical states**

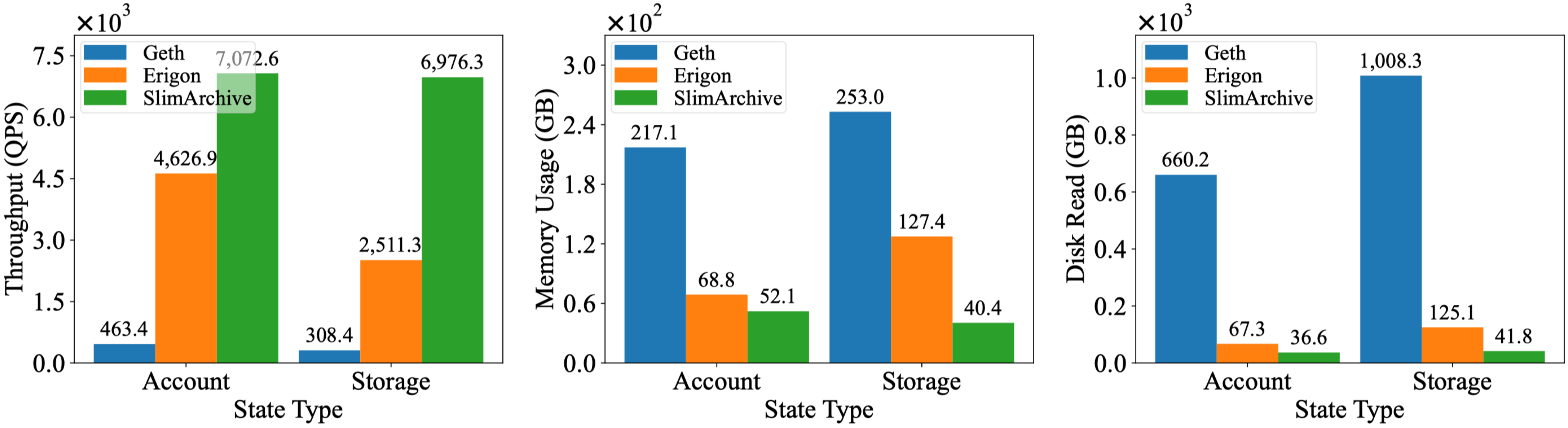
	Blocks	Agg.	Persisting	Total
Geth	0-14M	N/A	961.9	961.9
Erigon	0-18M	15.7	29.9	45.6
SLIMARCHIVE	**0-18M**	**11.0**	**3.7**	**14.7**

  - **Disk usage for historical states of 18M blocks**

	Geth	Erigon	SLIMARCHIVE
Storage (GB)	14,041.5	791.5	**267.6**

Evaluation

◆ **State access**



(a) Access throughput. **(b) Memory overhead.** **(c) Disk overhead.**

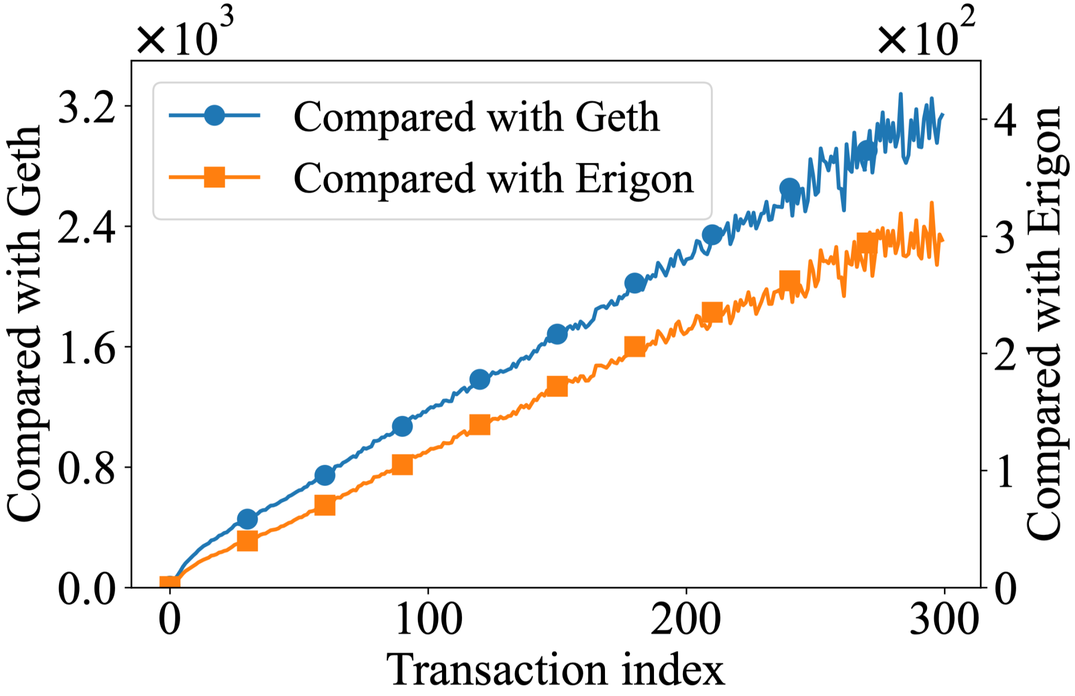
Evaluation

◆ Transaction execution

Overall speedup

Speedup over	Mean	Median
Geth	1112.5	672.2
Erigon	109.4	60.9

The positive correlation between transaction index and execution speedup



Summary & Takeaways

- The limitations of current Ethereum archive nodes
  - Inefficient MPT
  - Coarse-grained state granularity

- Our solution
  - Replace MPT with a compacted and flattened data model
  - Refine the granularity with transaction level

- Evaluation
  - Saves disk by 98.1%
  - Improves access throughput by 19.0×
  - Speeds up transaction execution by 1112.5×

Thank You!

Any questions? Contact me:

Hang Feng
Zhejiang University
Email: h_feng@zju.edu.cn