COLE: A Column-based Learned Storage for Blockchain Systems

Ce Zhang¹, Cheng Xu¹, Haibo Hu², and Jianliang Xu¹

¹Hong Kong Baptist University, ²Hong Kong Polytechnic University
Blockchain Technology

https://hackr.io/blog/applications-and-use-cases-of-blockchains
Blockchain Technology

- **Distributed Ledger** built upon a set of transactions agreed upon by mutually untrusted nodes
  - Data integrity
  - Provenance queries

![Diagram of blockchain](image)
Blockchain Storage Size

- All blockchain nodes are required to replicate the storage
- Ethereum storage size
  - Reach 16 TB as of Jan 2024
  - Increase around 4 TB per year after 2020

https://etherscan.io/chartsync/chainarchive
Ethereum Index

- Merkle Patricia Trie (MPT)
- Compressed prefix tree

```
<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v1</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v2</td>
</tr>
<tr>
<td>a11e67</td>
<td>v3</td>
</tr>
</tbody>
</table>
```
Ethereum Index

• Merkle Patricia Trie (MPT)
  • Compressed prefix tree

- Extension Node
- Branch Node
- Leaf Node

Search a11e67

```
block_i
  \( H_i \)

\( n_1 \) key: a1

\( n_2 \) 0 1 ... f

\( n_3 \) 0 ... e f

\( n_4 \) key: e67 v_3

\( n_5 \) key: 12 v_1

\( n_6 \) key: 5b v_2
```

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v_1</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v_2</td>
</tr>
<tr>
<td>a11e67</td>
<td>v_3</td>
</tr>
</tbody>
</table>
Ethereum Index

- Merkle Patricia Trie (MPT)
- Compressed prefix tree

Search a11e67

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v₁</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v₂</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃</td>
</tr>
</tbody>
</table>
Ethereum Index

• Merkle Patricia Trie (MPT)
  • Compressed prefix tree

Search a11e67

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v₁</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v₂</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃</td>
</tr>
</tbody>
</table>
Ethereum Index

• Merkle Patricia Trie (MPT)
  • Compressed prefix tree

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v_1</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v_2</td>
</tr>
<tr>
<td>a11e67</td>
<td>v_3</td>
</tr>
</tbody>
</table>

Search a11e67

- Extension Node
- Branch Node
- Leaf Node
Ethereum Index

- Merkle Patricia Trie (MPT)
- Compressed prefix tree

Search a11e67
Ethereum Index

- Merkle Patricia Trie (MPT)
- Compressed prefix tree

Search a11e67

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v₁</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v₂</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃</td>
</tr>
</tbody>
</table>
Ethereum Index

- Merkle Patricia Tree (MPT)
- Merkle Tree: data integrity

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v₁</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v₂</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃</td>
</tr>
</tbody>
</table>
**Ethereum Index**

- Merkle Patricia Tree (MPT)
- Merkle Tree: data integrity

---

**Merkle Patricia Tree (MPT)**

```
 addr     value
 a10012   v₁
 a10e5b   v₂
 a11e67   v₃
```

---

**Diagram**

Block $i$: $H_i$

Node $n_1$: key: $a1$

Node $n_2$: $0 \ 1 \ \ldots \ f$

Node $n_3$: $e \ f$

Node $n_4$: key: $e67 \ v₃$

Node $n_5$: key: $12 \ v₁$

Node $n_6$: key: $5b \ v₂$

---

$h(n₄) = h(e67|v₃)$
Ethereum Index

• Merkle Patricia Tree (MPT)
• Merkle Tree: data integrity

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>$v_1$</td>
</tr>
<tr>
<td>a10e5b</td>
<td>$v_2$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3$</td>
</tr>
</tbody>
</table>

$h(n_2) = h(0|n_3|1|h(n_4))$

$h(n_4) = h(e67|v3)$
Ethereum Index

- Merkle Patricia Tree (MPT)
- Merkle Tree: data integrity

![Diagram of Merkle Patricia Tree](image)

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v₁</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v₂</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃</td>
</tr>
</tbody>
</table>

\[
h(n_1) = h(a1|h(n_2))
\]

\[
h(n_2) = h(0|h(n_3)|1|h(n_4))
\]

\[
h(n_4) = h(e67|v3)
\]
Ethereum Index

- Merkle Patricia Tree (MPT)
- Merkle Tree: data integrity

\[
\begin{align*}
H_i &= h(0|h(n_3)|1|h(n_4)) \\
h(n_4) &= h(e67|v3) \\
h(n_2) &= h(0|h(n_3)|1|h(n_4)) \\
h(n_1) &= h(a1|h(n_2))
\end{align*}
\]

Extension Node
Branch Node
Leaf Node

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>(v_1)</td>
</tr>
<tr>
<td>a10e5b</td>
<td>(v_2)</td>
</tr>
<tr>
<td>a11e67</td>
<td>(v_3)</td>
</tr>
</tbody>
</table>
Ethereum Index

- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries

\[
\begin{array}{c}
\text{Addr} & \text{Value} \\
\hline
a10012 & v_1 \\
a10e5b & v_2 \\
a11e67 & v_3 \\
a11e67 & v_3' \\
\end{array}
\]
Ethereum Index

- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries

```
\\begin{array}{c|c}
\text{addr} & \text{value} \\
\hline
a10012 & v_1 \\
a10e5b & v_2 \\
a11e67 & v_3 \\
a11e67 & v_3' \\
\end{array}
```

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a10012$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$a10e5b$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$a11e67$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$a11e67$</td>
<td>$v_3'$</td>
</tr>
</tbody>
</table>

Diagram:

- Extension Node
- Branch Node
- Leaf Node

```
\begin{array}{c}
block_i \\
\hline
H_i \\
\hline
n_1 \quad \text{key: } a1 \\
\hline
n_2 \quad 0 \quad 1 \quad \ldots \quad f \\
\hline
n_3 \quad 0 \quad \ldots \quad e \quad f \\
\hline
n_4 \quad \text{key: } e67 \quad v_3 \\
\hline
n_5 \quad \text{key: } 12 \quad v_1 \\
\hline
n_6 \quad \text{key: } 5b \quad v_2 \\
\hline
\end{array}
```
Ethereum Index

- **Merkle Patricia Tree (MPT)**
  - Node duplication: provenance queries

---

### Table

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>$v_1$</td>
</tr>
<tr>
<td>a10e5b</td>
<td>$v_2$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3'$</td>
</tr>
</tbody>
</table>
• Merkle Patricia Tree (MPT)
  • Node duplication: provenance queries

Search a11e67 in block i

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v₁</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v₂</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃'</td>
</tr>
</tbody>
</table>
Ethereum Index

- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries

Search a11e67 in block $i$
• Merkle Patricia Tree (MPT)
  • Node duplication: provenance queries

Search a11e67 in block $i$

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>$v_1$</td>
</tr>
<tr>
<td>a10e5b</td>
<td>$v_2$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3'$</td>
</tr>
</tbody>
</table>
Ethereum Index

- Merkle Patricia Tree (MPT)
- Node duplication: provenance queries

Search a11e67 in block $i$
Ethereum Index

- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries

Search `a11e67` in block $i$

```plaintext
addr  value
a10012  v_1
a10e5b  v_2
a11e67  v_3
a11e67  v'_3
```
• Merkle Patricia Tree (MPT)
  • Node duplication: provenance queries

Search a11e67 in block $i$
• Merkle Patricia Tree (MPT)
  • Node duplication: provenance queries

![Diagram of Merkle Patricia Tree]

Search `a11e67` in block $i$

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>$v_1$</td>
</tr>
<tr>
<td>a10e5b</td>
<td>$v_2$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v'_3$</td>
</tr>
</tbody>
</table>
Ethereum Index

• Merkle Patricia Tree (MPT)
  • Node duplication: provenance queries

Search a11e67 in block $i$

```
 addr   value
a10012  $v_1$
a10e5b  $v_2$
a11e67  $v_3$
a11e67  $v_3'$
```
• Merkle Patricia Tree (MPT)
  • High storage overhead

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>$v_1$</td>
</tr>
<tr>
<td>a10e5b</td>
<td>$v_2$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3'$</td>
</tr>
</tbody>
</table>
• Merkle Patricia Tree (MPT)
  • High storage overhead

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>$v_1$</td>
</tr>
<tr>
<td>a10e5b</td>
<td>$v_2$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3'$</td>
</tr>
</tbody>
</table>
• Merkle Patricia Tree (MPT)

• High storage overhead

- Extension Node
- Branch Node
- Leaf Node
Ethereum Index

- Merkle Patricia Tree (MPT)
  - High storage overhead

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v₁</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v₂</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃'</td>
</tr>
</tbody>
</table>
Ethereum Index

- Merkle Patricia Tree (MPT)
- High storage overhead

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>$v_1$</td>
</tr>
<tr>
<td>a10e5b</td>
<td>$v_2$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3$</td>
</tr>
<tr>
<td>a11e67</td>
<td>$v_3'$</td>
</tr>
</tbody>
</table>
• Merkle Patricia Tree (MPT)
  • High storage overhead

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v1</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v2</td>
</tr>
<tr>
<td>a11e67</td>
<td>v3</td>
</tr>
<tr>
<td>a11e67</td>
<td>v3'</td>
</tr>
</tbody>
</table>
Ethereum Index

• Merkle Patricia Tree (MPT)
  • High storage overhead

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>a10012</td>
<td>v₁</td>
</tr>
<tr>
<td>a10e5b</td>
<td>v₂</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃</td>
</tr>
<tr>
<td>a11e67</td>
<td>v₃'</td>
</tr>
</tbody>
</table>
Ethereum Index

- Merkle Patricia Tree (MPT)
- High storage overhead

Design a blockchain index with less storage size
Learned Index

• Feature: **smaller** size and **faster** speed

• Core idea
  • Substitute directing keys in index nodes with a **learned model**
  • Model’s advantages:
    • Much **smaller size** than directing keys
    • **Faster** speed for locating data

Learned Index

• An example
  • Consider a key-value database with a linear distribution

- Index storage cost: $O(n)$
- Query cost: $O(\log_f n \cdot \log_2 f)$

- Model: $y = x$

- Index storage cost: $O(1)$
- Query cost: $O(1)$
• More general case
  • Hierarchical models
Applying Learned Index to Blockchain Storage

• Support data integrity
  • Merkle Hash Tree

\[
h_{1234} = h(h_{12} | h_{34})
\]

\[
h_{12} = h(h_1 | h_2)
\]

\[
h_{34} = h(h_3 | h_4)
\]
Applying Learned Index to Blockchain Storage

• Support data integrity
  • Provenance queries

\[ H_{block_i} \]

\[ h_{1234} = h(h_{12}|h_{34}) \]

\[ h_{12} = h(h_1|h_2) \]

\[ h_{34} = h(h_3|h_4) \]

\[ h_1 \]

\[ h_2 \]

\[ h_3 \]

\[ h_4 \]

\[ h_{1234} \]

\[ h_{12} \]

\[ h_{34} \]

...
Applying Learned Index to Blockchain Storage

- Support data integrity
- Provenance queries

\[ H_i \rightarrow block_i \rightarrow H_{i+1} \rightarrow block_{i+1} \]

\[ h_{1234} = h(h_{12} | h_{34}) \]

\[ h_{12} = h(h_1 | h_2) \]
\[ h_{34} = h(h_3 | h_4) \]

\[ h_{1234} = h(h_{1234} | h_{1234}) \]
\[ h_{44} = h(h_4 | h_4) \]
Applying Learned Index to Blockchain Storage

- Problem
  - Even larger size than MPT (5x to 31x larger in our experiment) 😞

\[ h_{1234} = h(h_{12}|h_{34}) \]

Model: \[ y = a \cdot x + b \]

\[ [h_0, h_1, ..., h_i, ..., h_f] \]

- \( f \) could be very large

1 hash update -> log # nodes duplication

2/29/2024
Our Design: COLE

• Column-based Learned Storage for Blockchain
  • No node duplication
  • Treat each state as a “column” to support provenance queries
Our Design: COLE

- **Column-based Learned Storage for Blockchain**
- No node duplication
- Treat each state as a "column" to support provenance queries

<table>
<thead>
<tr>
<th>block_{i+1}</th>
<th>State addr (k_1)</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>(v_1^1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(i)</td>
<td>(v_1^i)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr (k_2)</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>(v_2^3)</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>(i)</td>
<td>(v_2^i)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr (k_3)</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>(v_3^2)</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>(i)</td>
<td>(v_3^i)</td>
</tr>
<tr>
<td></td>
<td>(i + 1)</td>
<td>(v_3^{i+1})</td>
</tr>
</tbody>
</table>
Our Design: COLE

- **Column-based** Learned Storage for Blockchain
- No node duplication
- Treat each state as a “column” to support provenance queries

<table>
<thead>
<tr>
<th>$\text{block}_{i+1}$</th>
<th>State addr $k_1$</th>
<th>$\text{blk}$</th>
<th>$\text{value}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>$v^1_1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$i$</td>
<td>$v^i_1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr $k_2$</th>
<th>$\text{blk}$</th>
<th>$\text{value}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>$v^3_2$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$i$</td>
<td>$v^i_2$</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr $k_3$</th>
<th>$\text{blk}$</th>
<th>$\text{value}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$v^2_3$</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$i$</td>
<td>$v^i_3$</td>
<td></td>
</tr>
<tr>
<td>$i + 1$</td>
<td>$v^{i+1}_3$</td>
<td></td>
</tr>
</tbody>
</table>

Store them together:
Our Design: COLE

- **Column-based** Learned Storage for Blockchain
- No node duplication
- Treat each state as a “column” to support provenance queries

<table>
<thead>
<tr>
<th>block_{i+1}</th>
<th>State addr k_1</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>v_1^1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>v_i^1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr k_2</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>v_2^3</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>i</td>
<td>v_i^2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr k_3</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>v_3^2</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>i</td>
<td>v_i^3</td>
</tr>
<tr>
<td></td>
<td>i+1</td>
<td>v_{i+1}</td>
</tr>
</tbody>
</table>

Store them together:

\[ k_1, 1, v_1^1 \quad ... \quad k_1, i, v_i^1 \]
Our Design: COLE

- Column-based Learned Storage for Blockchain
  - No node duplication
  - Treat each state as a “column” to support provenance queries

<table>
<thead>
<tr>
<th>$\text{block}_{i+1}$</th>
<th>State addr $k_1$</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>$v_1^1$</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$i$</td>
<td>$v_i^1$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr $k_2$</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3</td>
<td>$v_2^3$</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>$i$</td>
<td>$v_i^2$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr $k_3$</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2</td>
<td>$v_3^2$</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>$i$</td>
<td>$v_i^3$</td>
</tr>
<tr>
<td></td>
<td>$i + 1$</td>
<td>$v_i^{i+1}$</td>
</tr>
</tbody>
</table>

Store them together:

$k_1, 1, v_1^1$ ... $k_1, i, v_i^1$ $k_2, 3, v_2^3$ ... $k_2, i, v_i^2$
Our Design: COLE

- Column-based Learned Storage for Blockchain
  - No node duplication
  - Treat each state as a "column" to support provenance queries

<table>
<thead>
<tr>
<th>block_{i+1}</th>
<th>State addr k_1</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>v^1_1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>v^i_1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>State addr k_2</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>3</td>
<td>v^3_2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>v^i_2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>State addr k_3</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>2</td>
<td>v^2_3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i</td>
<td>v^i_3</td>
</tr>
<tr>
<td></td>
<td></td>
<td>i + 1</td>
<td>v^{i+1}_3</td>
</tr>
</tbody>
</table>

Store them together:

| k_1, 1, v^1_1 | ... | k_1, i, v^i_1 | k_2, 3, v^3_2 | ... | k_2, i, v^i_2 | k_3, 2, v^2_3 | ... | k_3, i, v^i_3 | k_3, i + 1, v^{i+1}_3 |
Our Design: COLE

- **Column-based Learned Storage for Blockchain**
- **No node duplication**
- **Treat each state as a “column” to support provenance queries**

<table>
<thead>
<tr>
<th>block</th>
<th>State addr</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_{i+1})</td>
<td>1</td>
<td>1</td>
<td>(v^1_1)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(i)</td>
<td>(i)</td>
<td>(v^i_1)</td>
<td>(v^i_1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_1)</td>
<td>3</td>
<td>(v^3_2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(i)</td>
<td>(i)</td>
<td>(v^i_2)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>State addr</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_2)</td>
<td>2</td>
<td>(v^2_3)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>(i)</td>
<td>(i)</td>
<td>(v^i_3)</td>
</tr>
<tr>
<td>(i + 1)</td>
<td>(i + 1)</td>
<td>(v^{i+1}_3)</td>
</tr>
</tbody>
</table>

Store them together:

\[
\begin{array}{cccccccc}
  k_1, 1, v^1_1 & \ldots & k_1, i, v^i_1 & k_2, 3, v^3_2 & \ldots & k_2, i, v^i_2 & k_3, 2, v^2_3 & \ldots & k_3, i, v^i_3 & k_3, i + 1, v^{i+1}_3
\end{array}
\]

Easy to index, easy to search 😊
Our Design: COLE

- Column-based Learned Storage for Blockchain

Search in $block_i$

Provenance queries from the latest block
Our Design: COLE

- **Column-based** Learned Storage for Blockchain
- **LSM-tree-based** maintenance: easy to manage data writes

```
\begin{align*}
L_0 &\quad \langle k_3, i \rangle, v_3 \quad \langle k_3, i + 1 \rangle, v_3' \\
L_1.R_0 &\quad \langle k_1, i - 1 \rangle, v_1 \quad \langle k_1, i \rangle, v_1' \quad \langle k_2, i \rangle, v_2 \\
L_1.R_1 &\quad \ldots \\
\text{Disk level runs} &\quad \ldots
\end{align*}
```

In-Mem:
- MB-tree
- root_hash_list

On-Disk:
- Value File
- Index File
- Merkle File

```
\begin{tabular}{|c|c|c|}
\hline
addr & blk & value \\
\hline
k_1 & i - 1 & v_1 \\
k_1 & i & v_1' \\
k_2 & i & v_2 \\
k_3 & i & v_3 \\
k_3 & i + 1 & v_3' \\
\hline
\end{tabular}
```
Our Design: COLE

- **Column-based Learned Storage for Blockchain**
- **LSM-tree-based** maintenance: easy to manage data writes
Our Design: COLE

- **Column-based Learned Storage for Blockchain**
- **LSM-tree-based** maintenance: easy to manage data writes

![Diagram]

- In-Mem:
  - MB-tree
  - root_hash_list

- On-Disk:
  - Value File
  - Index File
  - Merkle File

<table>
<thead>
<tr>
<th>addr</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k_1$</td>
<td>$i - 1$</td>
<td>$v_1$</td>
</tr>
<tr>
<td>$k_1$</td>
<td>$i$</td>
<td>$v_1'$</td>
</tr>
<tr>
<td>$k_2$</td>
<td>$i$</td>
<td>$v_2$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$i$</td>
<td>$v_3$</td>
</tr>
<tr>
<td>$k_3$</td>
<td>$i + 1$</td>
<td>$v_3'$</td>
</tr>
</tbody>
</table>
Our Design: COLE

- **Column-based** Learned Storage for Blockchain
- **LSM-tree-based** maintenance: easy to manage data writes
Our Design: COLE

- **Column-based** Learned Storage for Blockchain
- **LSM-tree-based** maintenance: easy to manage data writes

\[
\begin{align*}
\text{block}_i & \quad \text{block}_{i+1} \\
H_i & \quad H_{i+1}
\end{align*}
\]

In-Mem: MB-tree, root_hash_list
On-Disk: Value File, Index File, Merkle File

<table>
<thead>
<tr>
<th>addr</th>
<th>blk</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>(k_1)</td>
<td>(i - 1)</td>
<td>(v_1)</td>
</tr>
<tr>
<td>(k_1)</td>
<td>(i)</td>
<td>(v_1')</td>
</tr>
<tr>
<td>(k_2)</td>
<td>(i)</td>
<td>(v_2)</td>
</tr>
<tr>
<td>(k_3)</td>
<td>(i)</td>
<td>(v_3)</td>
</tr>
<tr>
<td>(k_3)</td>
<td>(i + 1)</td>
<td>(v_3')</td>
</tr>
</tbody>
</table>

Help to compute the index digest -> integrity and consistency
Learned Model in COLE

- **$\epsilon$-Bounded Piecewise Linear Model**
  - $\mathcal{M} = \langle sl, ic, k_{min} \rangle$, $y = sl \cdot x + ic$
  - $\epsilon$-bound: $|y_{pred} - y_{real}| < \epsilon$, IO-efficient when $\epsilon = \frac{|Page|}{2}$

Search $\mathcal{K}_q$ using $\mathcal{M}$

<table>
<thead>
<tr>
<th>$\mathcal{P}_{i-1}$</th>
<th>$\mathcal{P}_i$</th>
<th>$\mathcal{P}_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{K}_0^{i-1}, \mathcal{K}_1^{i-1}, ...$</td>
<td>$\mathcal{K}_0^i, \mathcal{K}_1^i, ..., \mathcal{K}_m^i$</td>
<td>$\mathcal{K}_0^{i+1}, \mathcal{K}_1^{i+1}, ...$</td>
</tr>
</tbody>
</table>
Learned Model in COLE

• $\epsilon$-Bounded Piecewise Linear Model
  • $\mathcal{M} = \langle sl, ic, k_{min}\rangle$, $y = sl \cdot x + ic$
  • $\epsilon$-bound: $|y_{pred} - y_{real}| < \epsilon$, IO-efficient when $\epsilon = \frac{|Page|}{2}$

Search $\mathcal{K}_q$ using $\mathcal{M}$

\[
\begin{array}{c|c|c}
\mathcal{P}_{i-1} & \mathcal{P}_i & \mathcal{P}_{i+1} \\
\hline
\mathcal{K}_0^{i-1}, \mathcal{K}_1^{i-1}, ... & \mathcal{K}_0^{i}, \mathcal{K}_1^{i}, ..., \mathcal{K}_m^{i} & \mathcal{K}_0^{i+1}, \mathcal{K}_1^{i+1}, ...
\end{array}
\]

Predict $\mathcal{K}_q$ in $\mathcal{P}_i$, read $\mathcal{P}_i$
Learned Model in COLE

• $\epsilon$-Bounded Piecewise Linear Model
  • $\mathcal{M} = \langle sl, ic, k_{min} \rangle$, $y = sl \cdot x + ic$
  • $\epsilon$-bound: $|y_{\text{pred}} - y_{\text{real}}| < \epsilon$, IO-efficient when $\epsilon = \frac{|Page|}{2}$

Search $\mathcal{K}_q$ using $\mathcal{M}$

\[
\begin{array}{ccc}
\mathcal{P}_{i-1} & \mathcal{P}_i & \mathcal{P}_{i+1} \\
\mathcal{K}_0^{i-1}, \mathcal{K}_1^{i-1}, ... & \mathcal{K}_0^i, \mathcal{K}_1^i, ..., \mathcal{K}_m^i & \mathcal{K}_0^{i+1}, \mathcal{K}_1^{i+1}, ...
\end{array}
\]

$\mathcal{K}_q < \mathcal{K}_0^i$, search $\mathcal{P}_{i-1}$
Learned Model in COLE

- $\varepsilon$-Bounded Piecewise Linear Model
  - $\mathcal{M} = \langle s_l, i_c, k_{\min} \rangle$, $y = s_l \cdot x + i_c$
  - $\varepsilon$-bound: $|y_{\text{pred}} - y_{\text{real}}| < \varepsilon$, IO-efficient when $\varepsilon = \frac{|\text{Page}|}{2}$

Search $\mathcal{K}_q$ using $\mathcal{M}$

<table>
<thead>
<tr>
<th>$\mathcal{P}_{i-1}$</th>
<th>$\mathcal{P}_i$</th>
<th>$\mathcal{P}_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{K}<em>{i-1}^0, \mathcal{K}</em>{i-1}^1, \ldots$</td>
<td>$\mathcal{K}_i^0, \mathcal{K}_i^1, \ldots, \mathcal{K}_m^i$</td>
<td>$\mathcal{K}_0^{i+1}, \mathcal{K}_1^{i+1}, \ldots$</td>
</tr>
</tbody>
</table>

$\mathcal{K}_q \succ \mathcal{K}_m^i$, search $\mathcal{P}_{i+1}$
Learned Model in COLE

• \( \epsilon \)-Bounded Piecewise Linear Model
  • \( \mathcal{M} = \langle sl, ic, k_{min} \rangle \), \( y = sl \cdot x + ic \)
  • \( \epsilon \)-bound: \( |y_{pred} - y_{real}| < \epsilon \), IO-efficient when \( \epsilon = \frac{|Page|}{2} \)

Search \( \mathcal{K}_q \) using \( \mathcal{M} \)

\[
\begin{align*}
\mathcal{P}_{i-1} & \quad \mathcal{P}_i & \quad \mathcal{P}_{i+1} \\
\mathcal{K}_0^{i-1}, \mathcal{K}_1^{i-1}, & \quad \mathcal{K}_0^i, \mathcal{K}_1^i, & \quad \mathcal{K}_0^{i+1}, \mathcal{K}_1^{i+1}, \\
& \mathcal{K}_m^i & \mathcal{K}_m^i \\
\end{align*}
\]

Else, search \( \mathcal{P}_i \)
Learned Model in COLE

• $\epsilon$-Bounded Piecewise Linear Model
  • $\mathcal{M} = \langle sl, ic, k_{min} \rangle$, $y = sl \cdot x + ic$
  • $\epsilon$-bound: $|y_{pred} - y_{real}| < \epsilon$, IO-efficient when $\epsilon = \frac{|Page|}{2}$

Search $\mathcal{K}_q$ using $\mathcal{M}$

<table>
<thead>
<tr>
<th>$\mathcal{P}_{i-1}$</th>
<th>$\mathcal{P}_i$</th>
<th>$\mathcal{P}_{i+1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{K}_0^{i-1}, \mathcal{K}_1^{i-1}, ...$</td>
<td>$\mathcal{K}_0^i, \mathcal{K}_1^i, ..., \mathcal{K}_m^i$</td>
<td>$\mathcal{K}_0^{i+1}, \mathcal{K}_1^{i+1}, ...$</td>
</tr>
</tbody>
</table>

Else, search $\mathcal{P}_i$

2 Page IO
Learned Model in COLE

• \(\varepsilon\)-Bounded Piecewise Linear Model
  • Fast to learn: \(O(1)\) cost to add each point

\[
\begin{align*}
\text{Position} & \\
\text{Key} & \\
\text{Pos} & \\
& \mathcal{K}
\end{align*}
\]
Learned Model in COLE

- $\epsilon$-Bounded Piecewise Linear Model
- Fast to learn: $O(1)$ cost to add each point

\[ \begin{array}{c}
\text{Position} \\
\text{Key} \quad \mathcal{K}_1 \quad \mathcal{K}_2 \\
\text{Pos} \quad \text{pos}_1 \quad \text{pos}_2
\end{array} \]
Learned Model in COLE

• $\varepsilon$-Bounded Piecewise Linear Model
  • Fast to learn: $O(1)$ cost to add each point
Learned Model in COLE

• $\epsilon$-Bounded Piecewise Linear Model
  • Fast to learn: $O(1)$ cost to add each point

\[ \text{Position} \]

$K_1$, $K_2$, $K_3$

$pos_1$, $pos_2$, $pos_3$
Learned Model in COLE

- $\epsilon$-Bounded Piecewise Linear Model
  - Fast to learn: $O(1)$ cost to add each point
Learned Model in COLE

- $\varepsilon$-Bounded Piecewise Linear Model
  - Fast to learn: $O(1)$ cost to add each point

\[
\begin{align*}
K_1 & \quad K_2 \\
K_3 & \quad K_4
\end{align*}
\]

\[
\begin{align*}
pos_1 & \quad pos_2 \\
pos_3 & \quad pos_4
\end{align*}
\]
Learned Model in COLE

- \( \varepsilon \)-Bounded Piecewise Linear Model
  - Fast to learn: \( O(1) \) cost to add each point
Learned Model in COLE

- $\epsilon$-Bounded Piecewise Linear Model
- Fast to learn: $O(1)$ cost to add each point

\[ s_1, s_2, s_3, s_4, s_5 \]

\[ \mathcal{K}_1, \mathcal{K}_2, \mathcal{K}_3, \mathcal{K}_4, \mathcal{K}_5 \]

\[ \text{Pos: } pos_1, pos_2, pos_3, pos_4, pos_5 \]
Learned Model in COLE

- $\epsilon$-Bounded Piecewise Linear Model
  - Fast to learn: $O(1)$ cost to add each point

\[ \mathcal{K} \]

\[ pos_1 pos_2 pos_3 pos_4 pos_5 \]

Key

\[ \mathcal{K}_1 \mathcal{K}_2 \mathcal{K}_3 \mathcal{K}_4 \mathcal{K}_5 \]

Yield model $\mathcal{M}$

\[ p_{max} = 4, \ k_{min} = \mathcal{K}_1 \]
Learned Model in COLE

• $\epsilon$-Bounded Piecewise Linear Model
• Fast to learn: $O(1)$ cost to add each point
Learned Model in COLE

• Index File
  • Build multiple layers of models and write them to a file

Layer 1

\[
S_1 \ S_2 \ S_3 \ S_4 \ S_5 \ S_6 \ S_7 \quad \ldots \quad S_i \ S_{i+1}
\]
Learned Model in COLE

- Index File
  - Build **multiple layers** of models and write them to a file
Learned Model in COLE

- Index File
  - Build **multiple layers** of models and write them to a file
Learned Model in COLE

• Index File
  • Build multiple layers of models and write them to a file

Layer 1

\[ \mathcal{M}_1 \quad \mathcal{M}_2 \quad \mathcal{M}_3 \quad \mathcal{M}_4 \quad \ldots \quad \mathcal{M}_m \]

\[ \mathcal{K}_1 \quad \mathcal{K}_2 \quad \mathcal{K}_3 \quad \mathcal{K}_4 \quad \mathcal{K}_5 \quad \mathcal{K}_6 \quad \mathcal{K}_7 \quad \ldots \quad \mathcal{K}_i \quad \mathcal{K}_{i+1} \]

\[ s_1 \quad s_2 \quad s_3 \quad s_4 \quad s_5 \quad s_6 \quad s_7 \quad \ldots \quad s_i \quad s_{i+1} \]
Learned Model in COLE

- Index File
  - Build *multiple layers* of models and write them to a file
Learned Model in COLE

- Index File
  - Build **multiple layers** of models and write them to a file

Layer 1

Layer 2

Learner
Learned Model in COLE

- Index File
  - Build **multiple layers** of models and write them to a file
Learned Model in COLE

- Index File
  - Build **multiple layers** of models and write them to a file

Index File Layout
Merkle File

- $m$-ary complete MHT to ensure data integrity
- When the number of states $n$ and fanout $m$ are given, MHT’s structure is fixed
Merkle File

• Construct $m$-ary complete MHT streamingly
  • Maintain MHT’s height number of buffers
• Construct $m$-ary complete MHT streamingly
  • Maintain MHT’s height number of buffers
Merkle File

• Construct $m$-ary complete MHT streamingly
  • Maintain MHT’s height number of buffers
• Construct $m$-ary complete MHT streamingly
  • Maintain MHT’s height number of buffers
Merkle File

- Construct $m$-ary complete MHT streamingly
  - Maintain MHT’s height number of buffers
• Construct $m$-ary complete MHT streamingly
  • Maintain MHT’s height number of buffers
• Construct $m$-ary complete MHT streamingly
  • Maintain MHT’s height number of buffers
Read Operations

• Get query
  • Search for the latest value of $addr_q$: $\mathcal{K}_q = \langle addr_q, \text{max\_int} \rangle$
  • Retrieve the state with the largest $\mathcal{K}_r$ and $\mathcal{K}_r < \mathcal{K}_q$

For each run in LSM-tree:

```
Layer 1        Layer 2        Layer 3

Index File

$\mathcal{M}_1^1$  ...  $\mathcal{M}_m^1$  $\mathcal{M}_1^2$  ...  $\mathcal{M}_3^2$  $\mathcal{M}_1^4$

Value File

$P_{i-1}$  $P_i$  $P_{i+1}$

...  $\mathcal{K}_0^i, \mathcal{K}_1^i, ..., \mathcal{K}_m^i$  ...
```
Read Operations

• Get query
  • Search for the latest value of $addr_q$: $\mathcal{K}_q = \langle addr_q, \text{max\_int} \rangle$
  • Retrieve the state with the largest $\mathcal{K}_r$ and $\mathcal{K}_r < \mathcal{K}_q$

For each run in LSM-tree:
Read Operations

• Get query
  • Search for the latest value of $addr_q$: $\mathcal{K}_q = \langle addr_q, \text{max\_int} \rangle$
  • Retrieve the state with the largest $\mathcal{K}_r$ and $\mathcal{K}_r < \mathcal{K}_q$

For each run in LSM-tree:
Read Operations

• Get query
  • Search for the latest value of $addr_q$: $\mathcal{K}_q = \langle addr_q, \text{max\_int} \rangle$
  • Retrieve the state with the largest $\mathcal{K}_r$ and $\mathcal{K}_r < \mathcal{K}_q$

For each run in LSM-tree:

\[
\begin{array}{cccc}
\text{Index File} & \text{Layer 1} & \text{Layer 2} & \text{Layer 3} \\
\mathcal{M}^1_1 & \ldots & \mathcal{M}^1_m & \ldots \\
\mathcal{M}^2_1 & \ldots & \mathcal{M}^2_m & \ldots \\
\mathcal{M}^3_1 & \mathcal{M}^4_1 & \ldots & \ldots \\
\end{array}
\]

\[
\begin{array}{cccc}
\text{Value File} & \mathcal{P}_{i-1} & \mathcal{P}_i & \mathcal{P}_{i+1} \\
\ldots & \mathcal{K}^i_0, \mathcal{K}^i_1, \ldots, \mathcal{K}^i_m & \ldots & \ldots \\
\end{array}
\]
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)

\[ L_0 \]

\[ L_1 \]

\[ L_2 \]

\[ L_3 \]
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)

```
\[ L_0 \]
\[ L_1 \]
\[ L_2 \]
\[ L_3 \]
```

- \( R_0 \)
- \( R_1 \)
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)
Read Operations

• Get query
  • For each LSM-tree level, search in a top-down fashion
  • For each level, search each run by freshness (new -> old)

$L_0$

$L_1$
  $R_0$  $R_1$

$L_2$
  $R_0$  $R_1$

$L_3$
  $R_0$

Find the result, early stop
Read Operations

- Provenance query
  - (i) range query, (ii) Merkle proof generation
  - $\mathcal{K}_l = \langle addr_q, blk_l \rangle$; $\mathcal{K}_u = \langle addr_q, blk_u \rangle$
  - Search level by level from top to bottom and combine results and proofs

Search range

$s_1$ and $s_4$ are boundary objects
Read Operations

• Provenance query
  • (i) range query, (ii) Merkle proof generation
  • $\mathcal{K}_l = \langle addr_q, blk_l \rangle$; $\mathcal{K}_u = \langle addr_q, blk_u \rangle$
  • Search level by level from top to bottom and combine results and proofs

Search range

$s_1$ and $s_4$ are boundary objects
Optimization

- Write asynchronous merge operation
  - Alleviate the long-tail latency (known as *write stall*)
  - Make merge operation *asynchronous*
  - Commit the storage in a *consistent way*
    - In blockchain systems, honest nodes’ storage should be consistent

Figure source: Liang J, Chai Y. CruiseDB: An LSM-Tree Key-Value Store with Both Better Tail Throughput and Tail Latency. IEEE ICDE 21
Evaluation

• Baselines
  • MPT
  • LIPP + node-persist-MHT
  • Column-based Merkle Index (column-based design with traditional indexes)

• Underlying storage
  • Baselines: RocksDB
  • COLE: simple files

• Workloads
  • SmallBank
  • KVStore + YCSB (read-write, read-only, write-only)
• Compare with MPT
  • Storage size is reduced by 94% and 93% for two workloads, respectively
  • Throughput is improved by 1.4x – 5.4x
  • COLE* (with async merge) is slightly worse than COLE
• With more write: MPT degrades by up to 93% while that of COLE and COLE* degrades by up to 87%
• COLE* decreases the tail latency by 1-2 orders
• Provenance query
  • MPT’s CPU time and proof size grow **linearly** with the block height range while COLE and COLE* grow **sub-linearly**
• Designed COLE, an efficient column-based learned storage for blockchain systems

• Developed a disk-optimized learned index to facilitate efficient data retrieval

• Experiments demonstrate that COLE significantly outperforms MPT and other baselines in terms of both storage size and throughput
Thank You