COLE: A Column-based Learned Storage for Blockchain Systems

Ce Zhang[1], Cheng Xu[1], Haibo Hu[2], and Jianliang Xu[1]

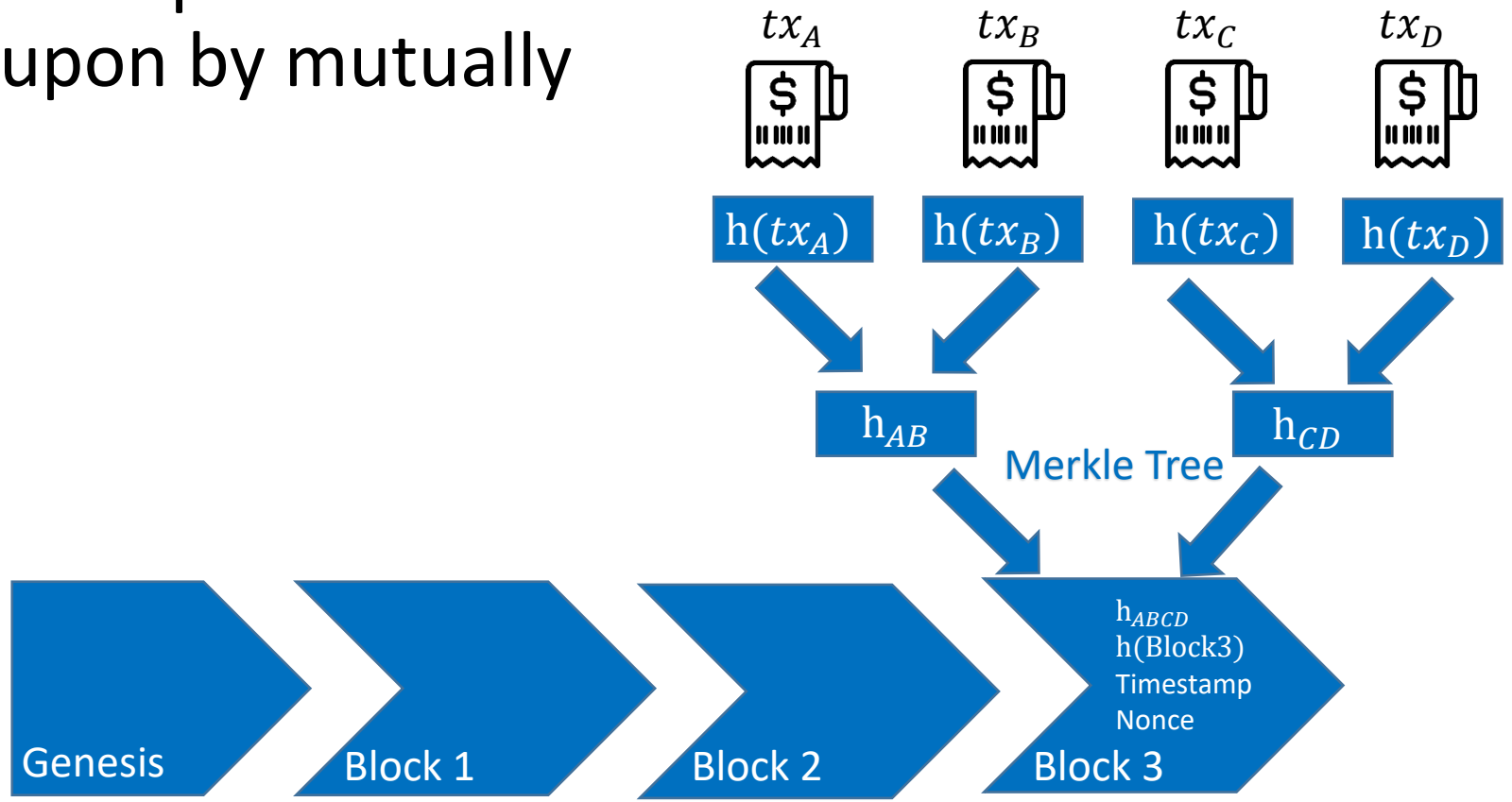
[1]Hong Kong Baptist University, [2]Hong Kong Polytechnic University

Blockchain Technology

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

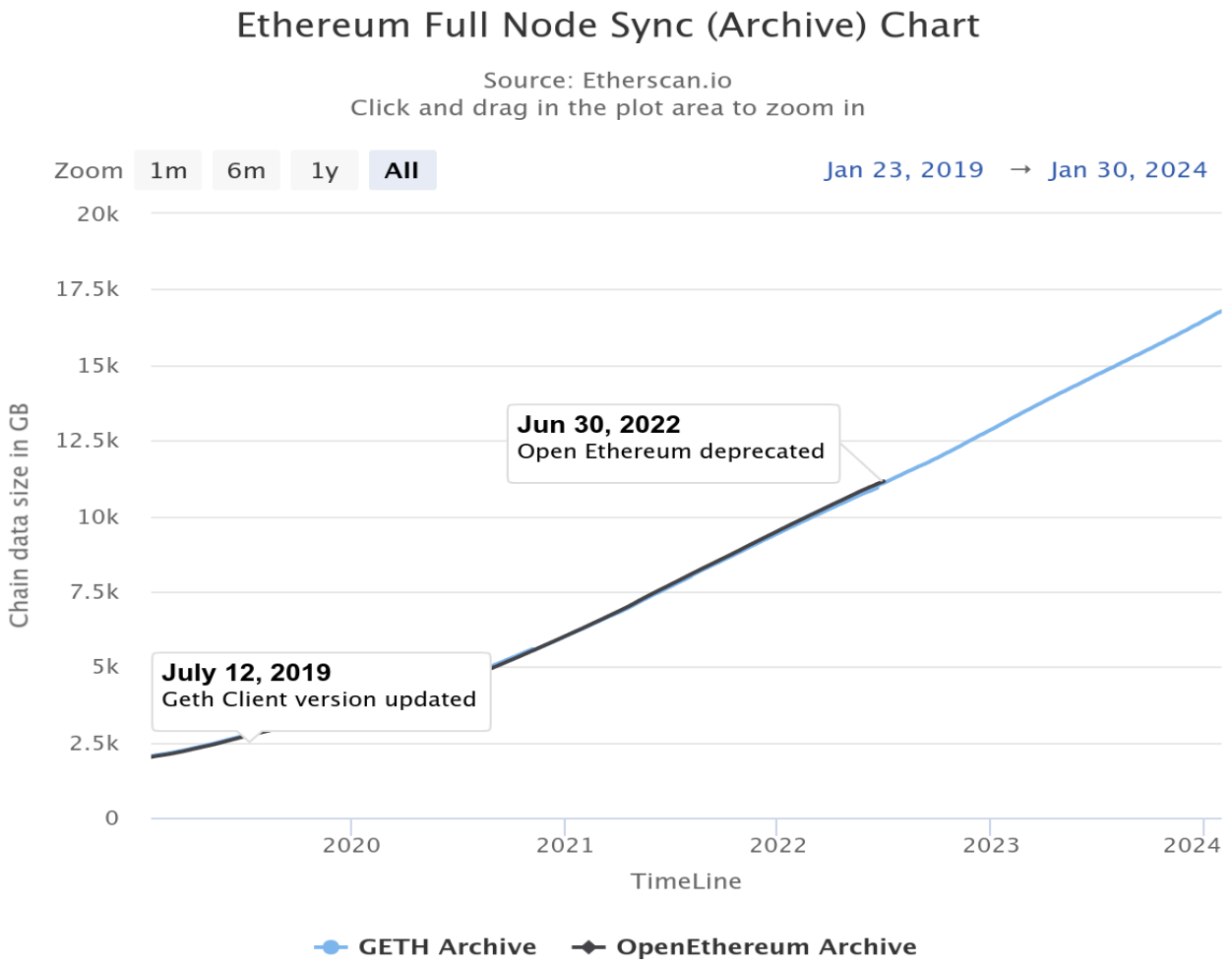
Blockchain Technology

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



Blockchain Storage Size

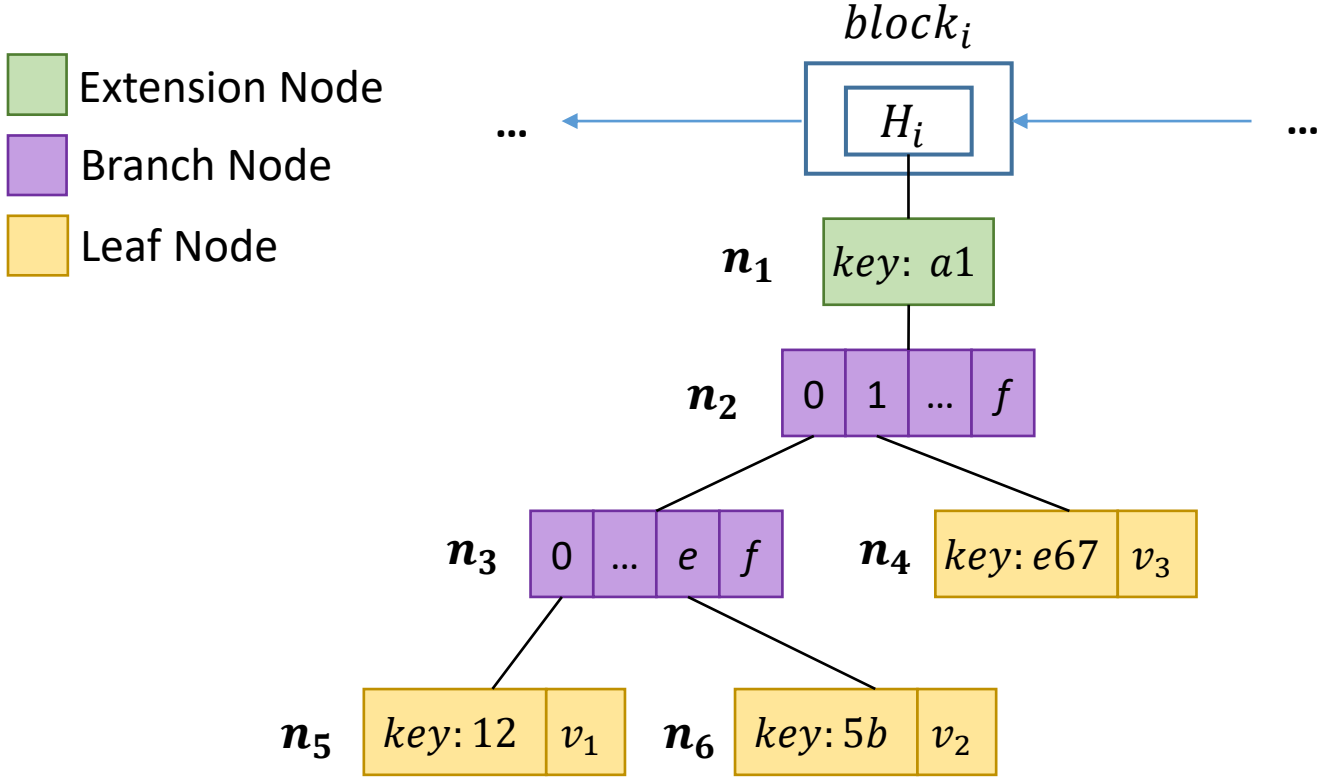
- All blockchain nodes are required to replicate the storage
- Ethereum storage size
  - Reach 16 TB as of Jan 2024
  - Increase around 4 TB per year after 2020



https://etherscan.io/chartsync/chainarchive

Ethereum Index

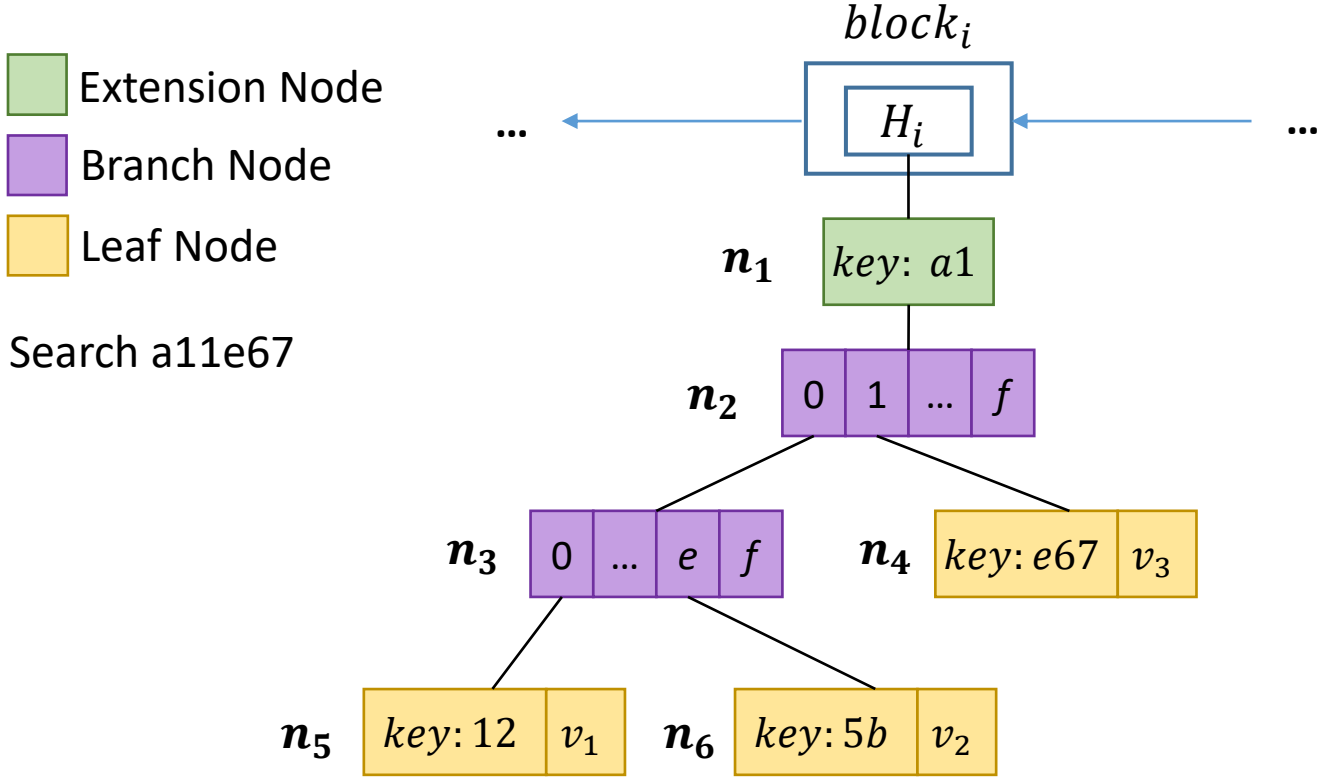
- Merkle Patricia Trie (MPT)
  - Compressed prefix tree



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

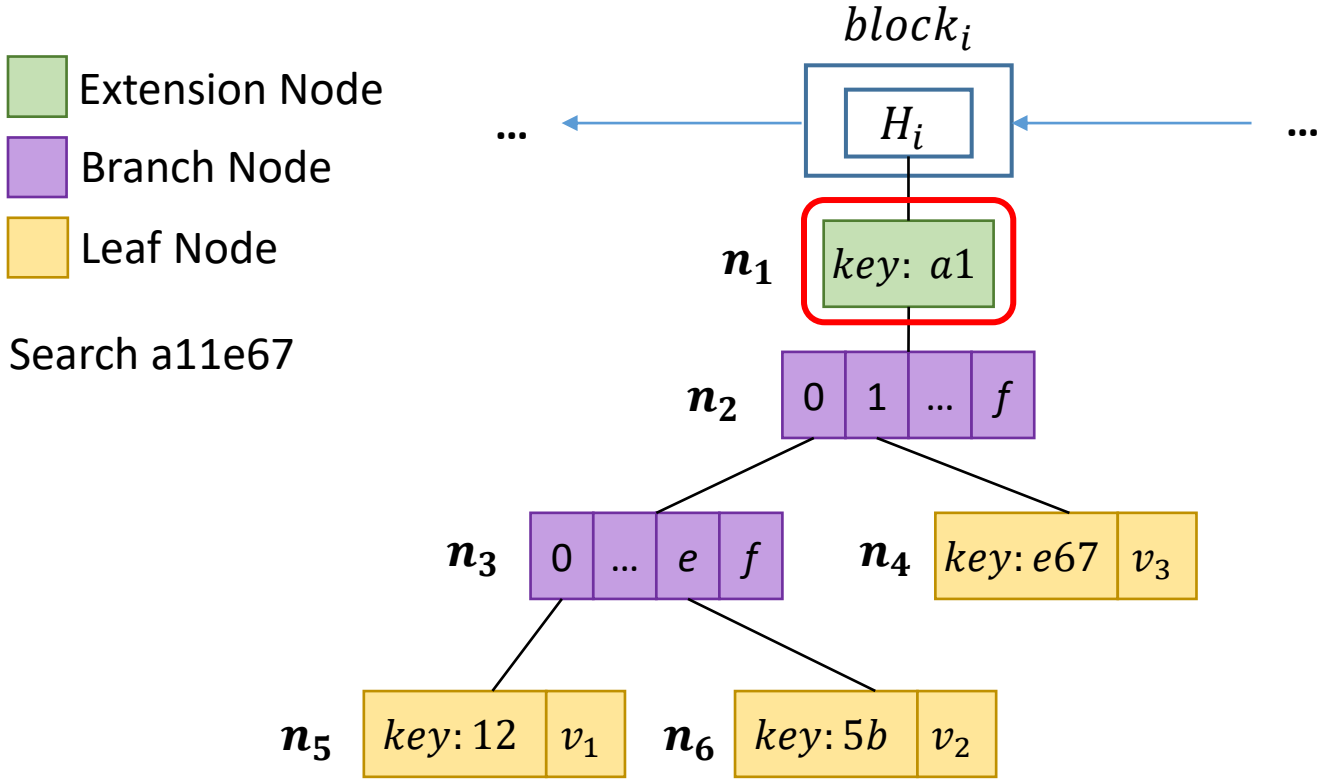
- Merkle Patricia Trie (MPT)
  - Compressed prefix tree



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

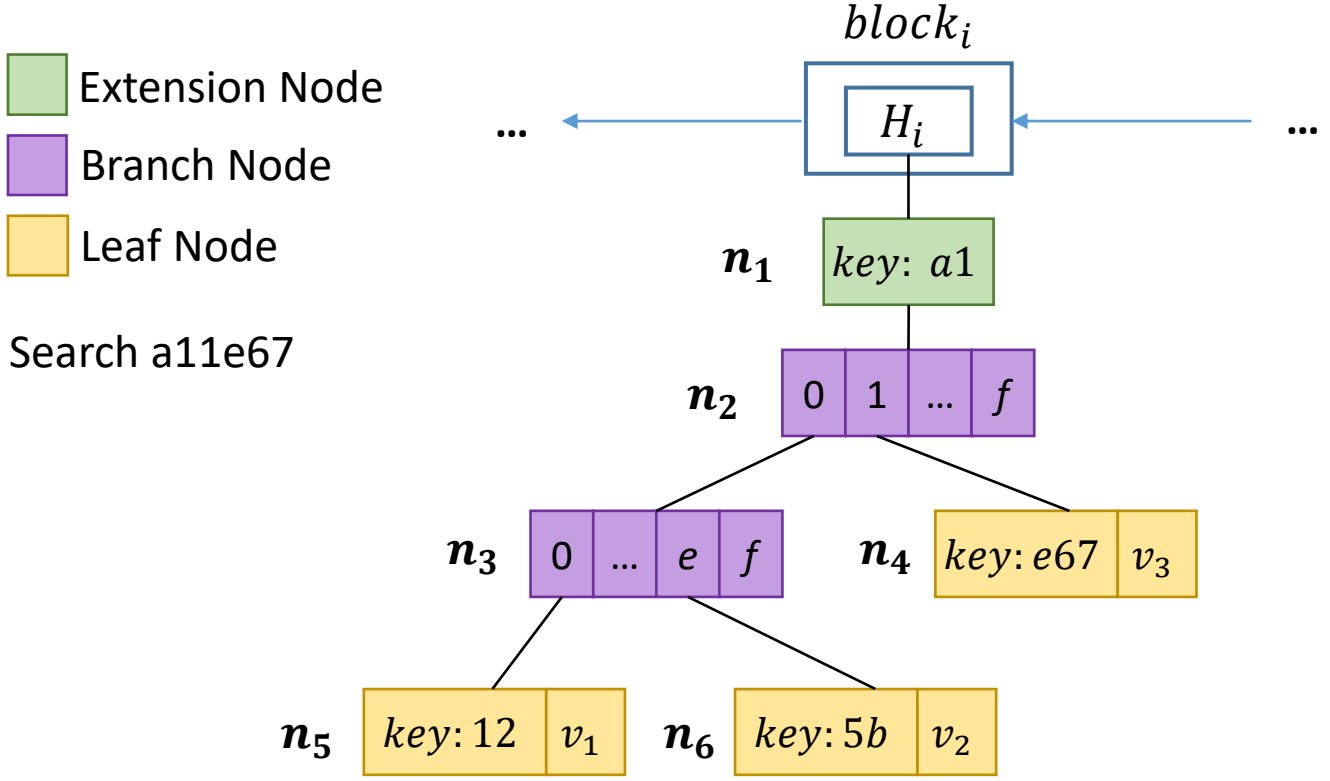
- Merkle Patricia Trie (MPT)
  - Compressed prefix tree



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

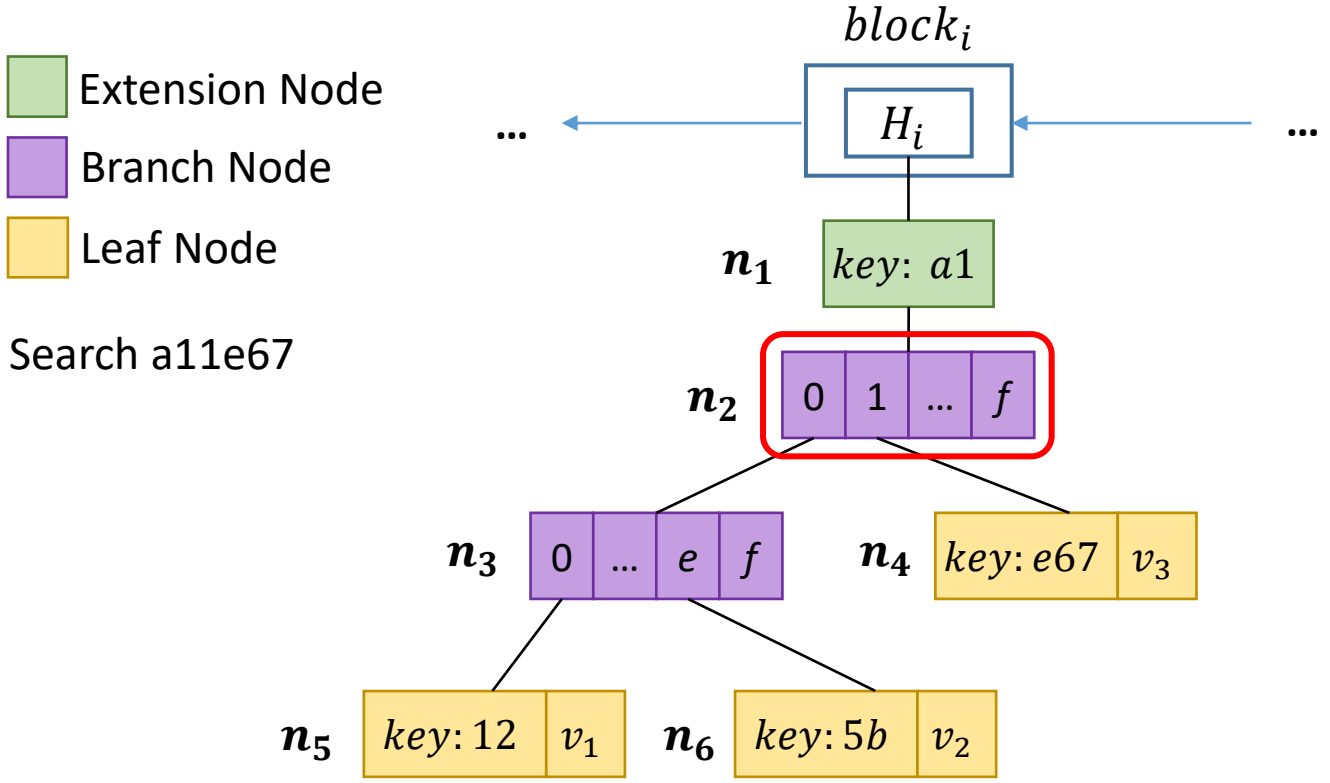
- Merkle Patricia Trie (MPT)
  - Compressed prefix tree



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

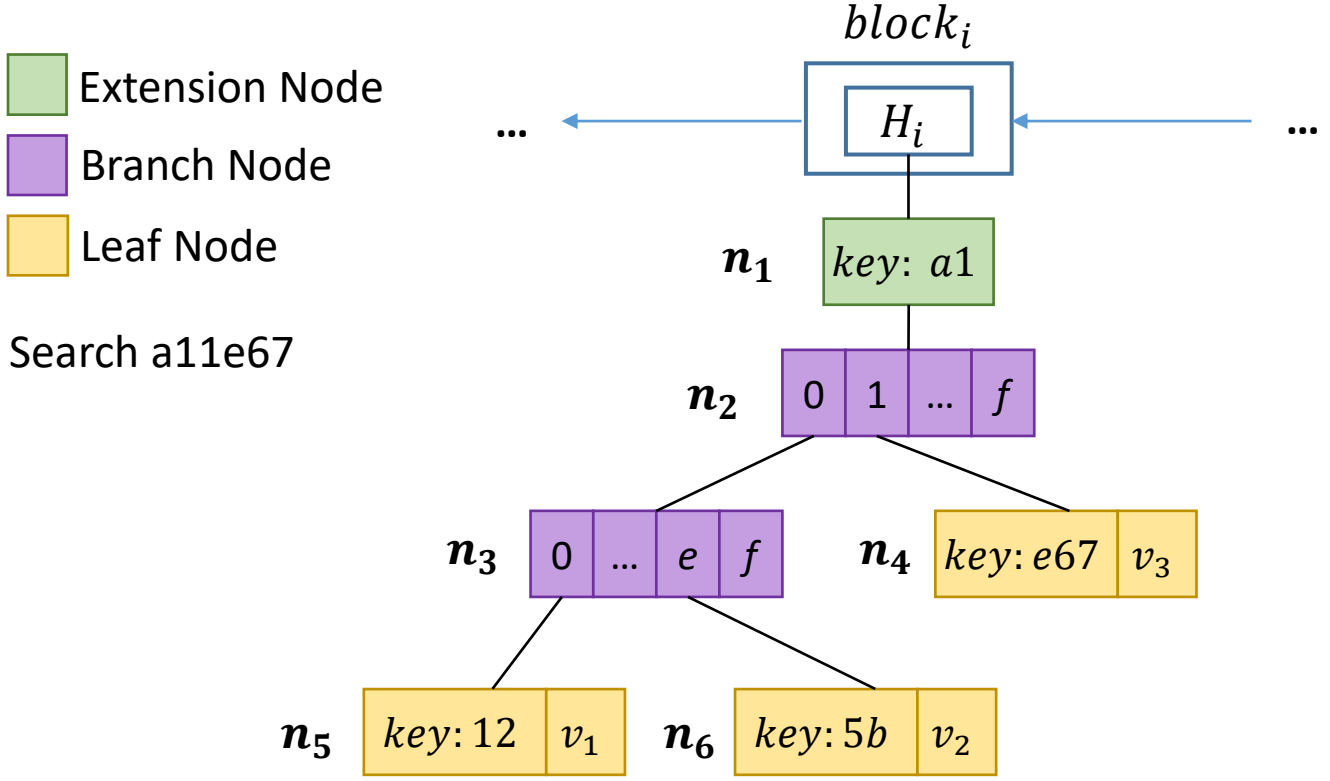
- Merkle Patricia Trie (MPT)
  - Compressed prefix tree



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

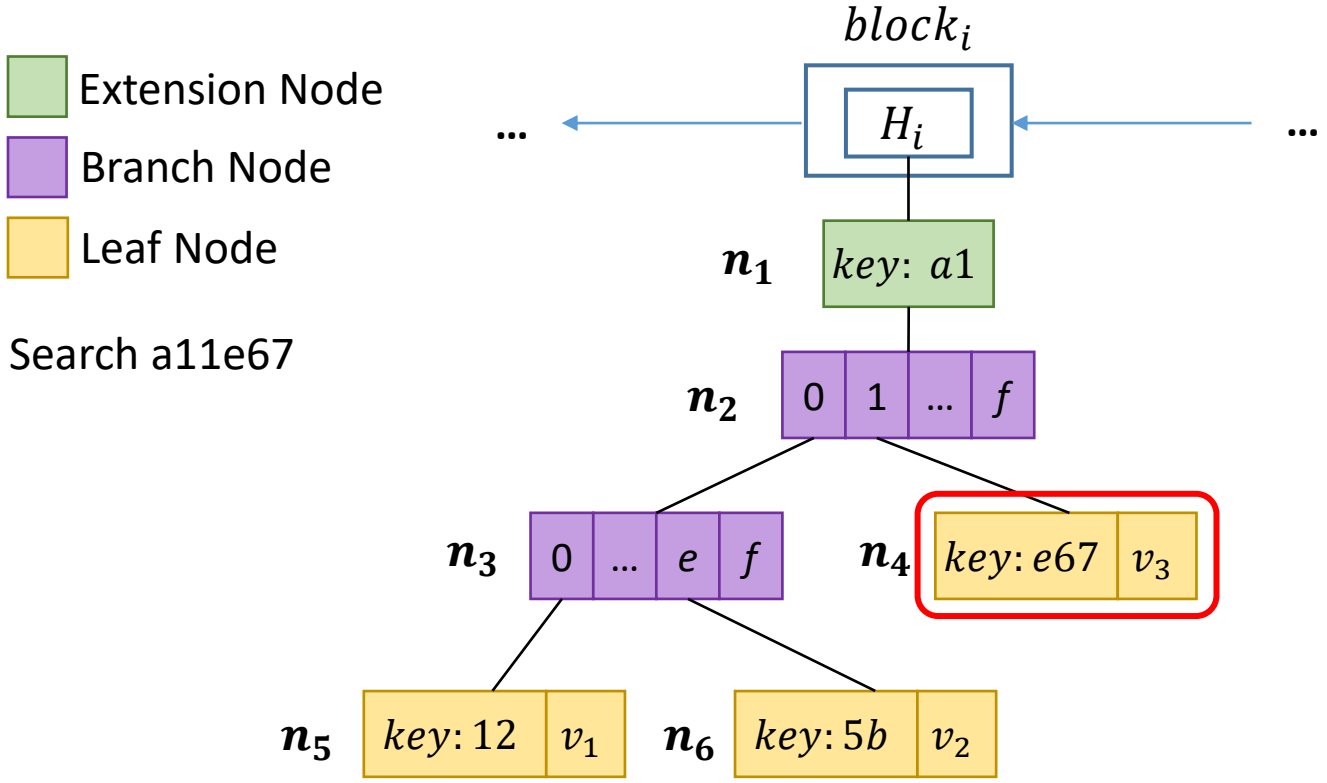
- Merkle Patricia Trie (MPT)
  - Compressed prefix tree



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

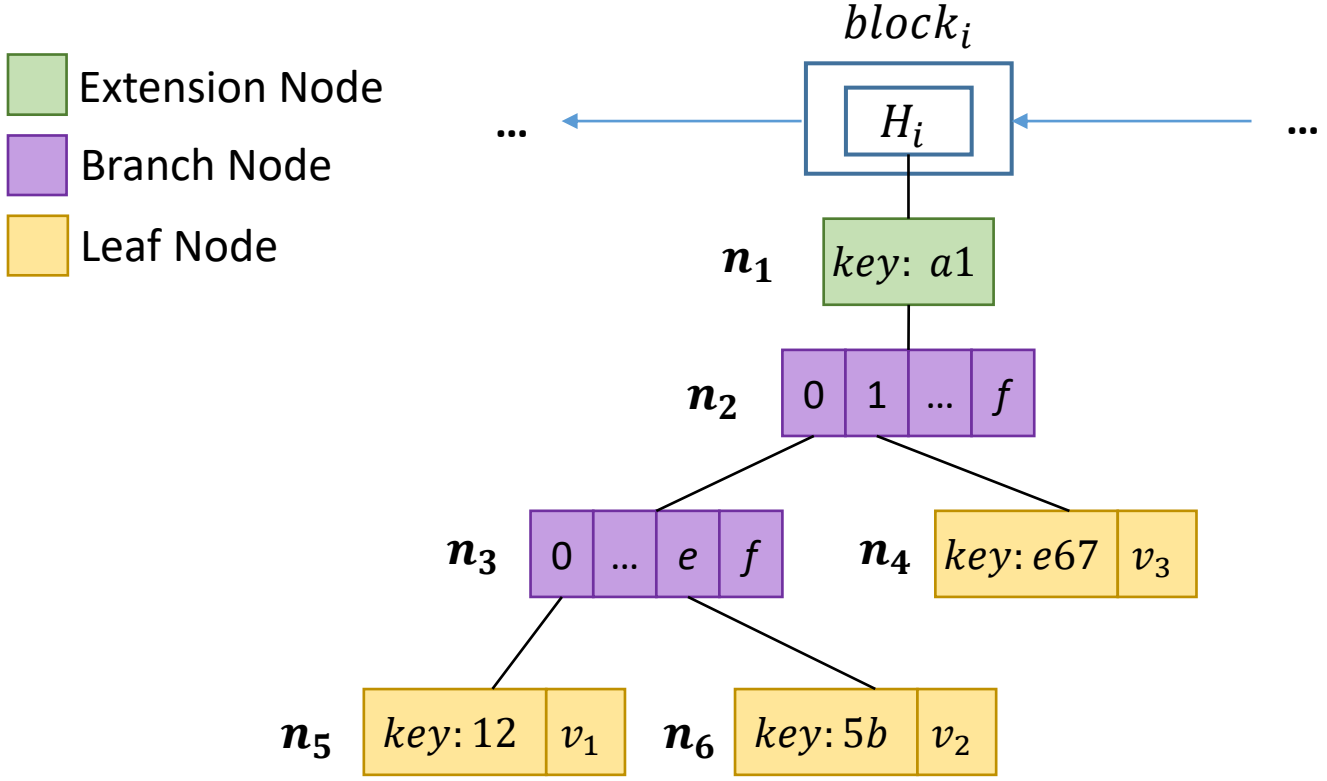
- Merkle Patricia Trie (MPT)
  - Compressed prefix tree



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

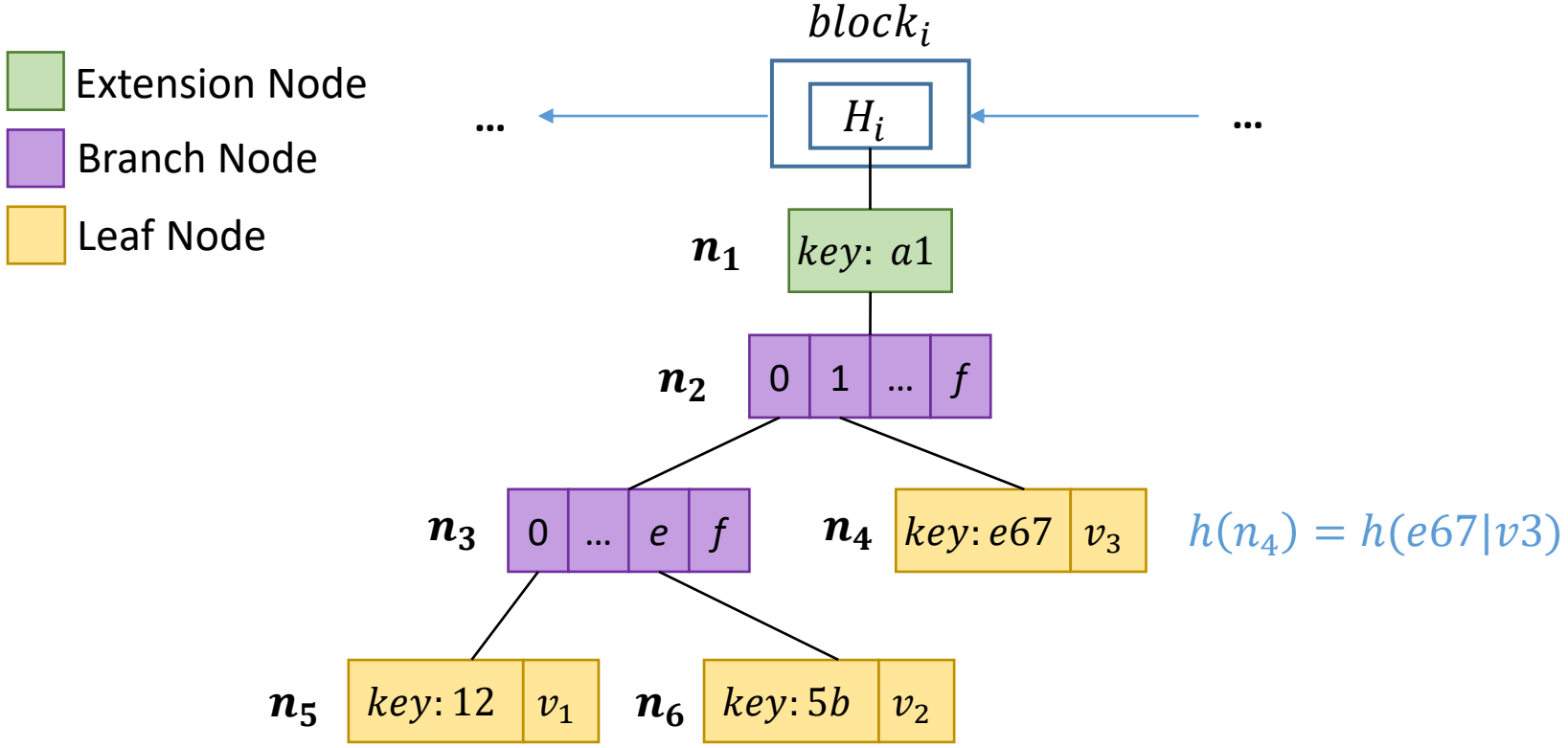
- Merkle Patricia Tree (MPT)
  - Merkle Tree: data integrity



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

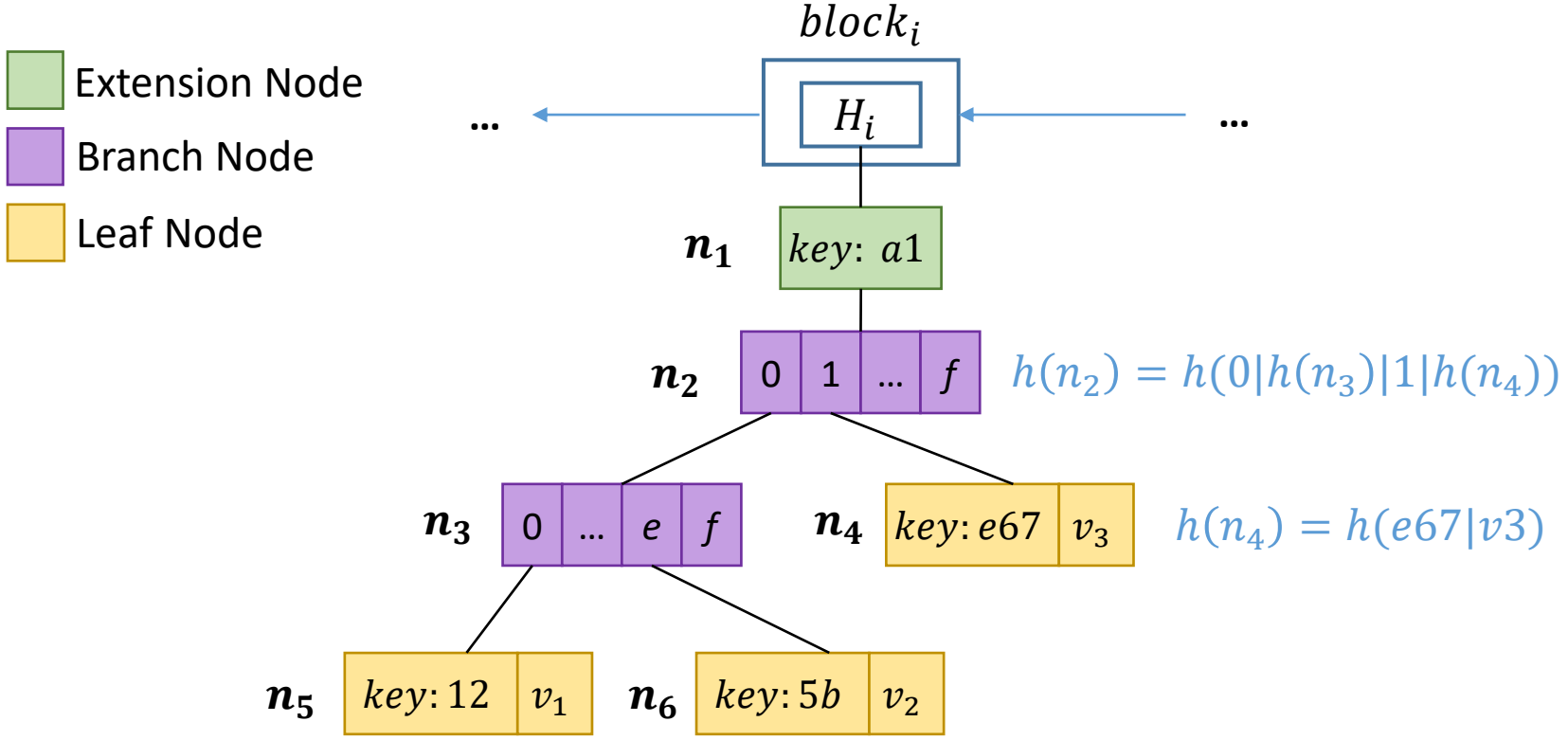
- Merkle Patricia Tree (MPT)
  - Merkle Tree: data integrity



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

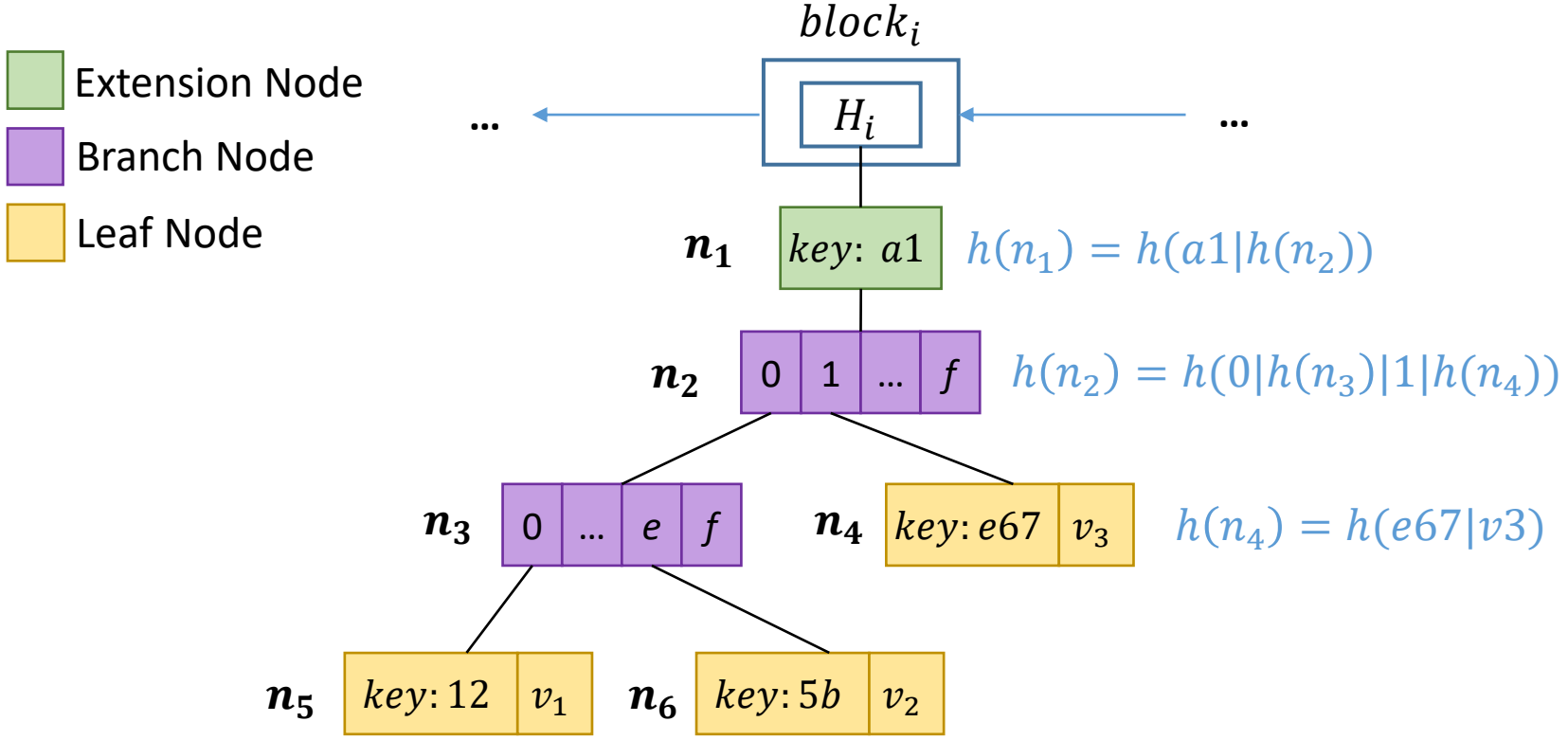
- Merkle Patricia Tree (MPT)
  - Merkle Tree: data integrity



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

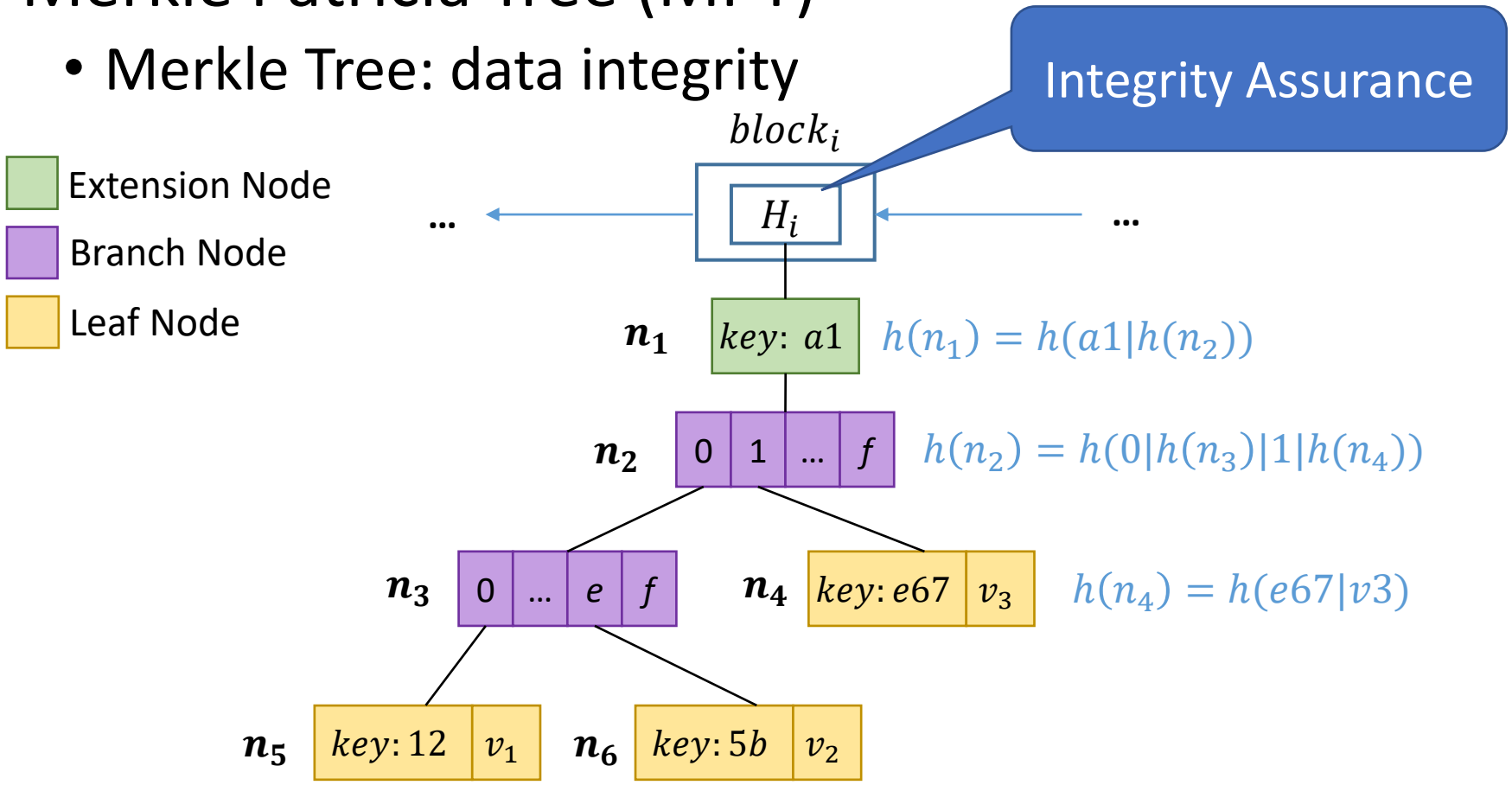
- Merkle Patricia Tree (MPT)
  - Merkle Tree: data integrity



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

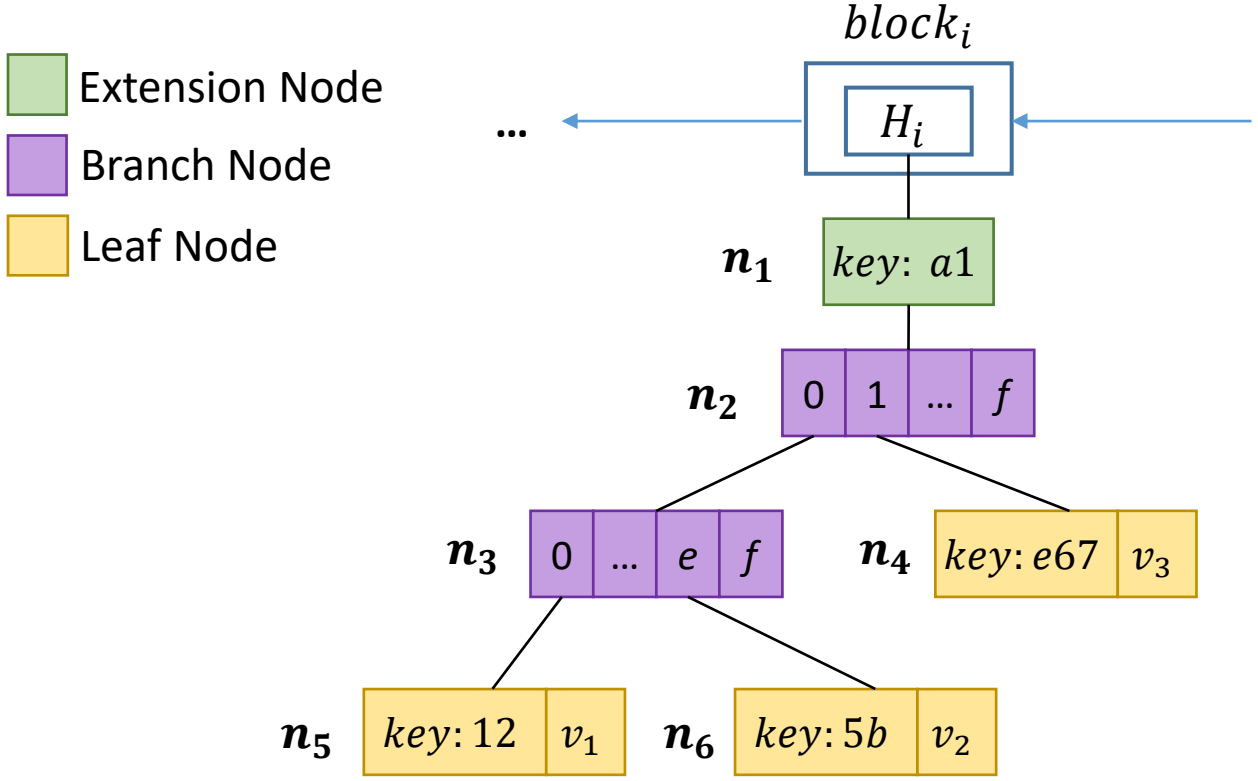
- Merkle Patricia Tree (MPT)
  - Merkle Tree: data integrity



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3

Ethereum Index

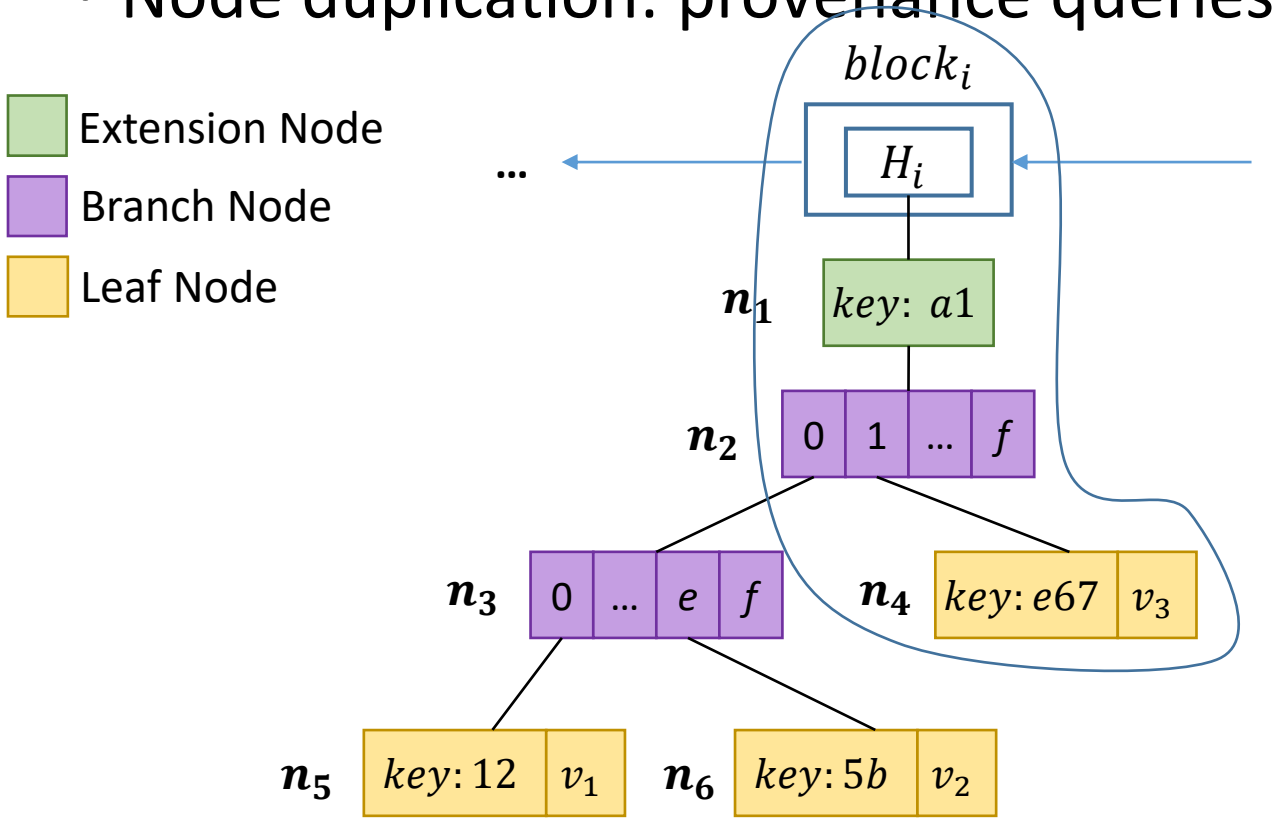
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

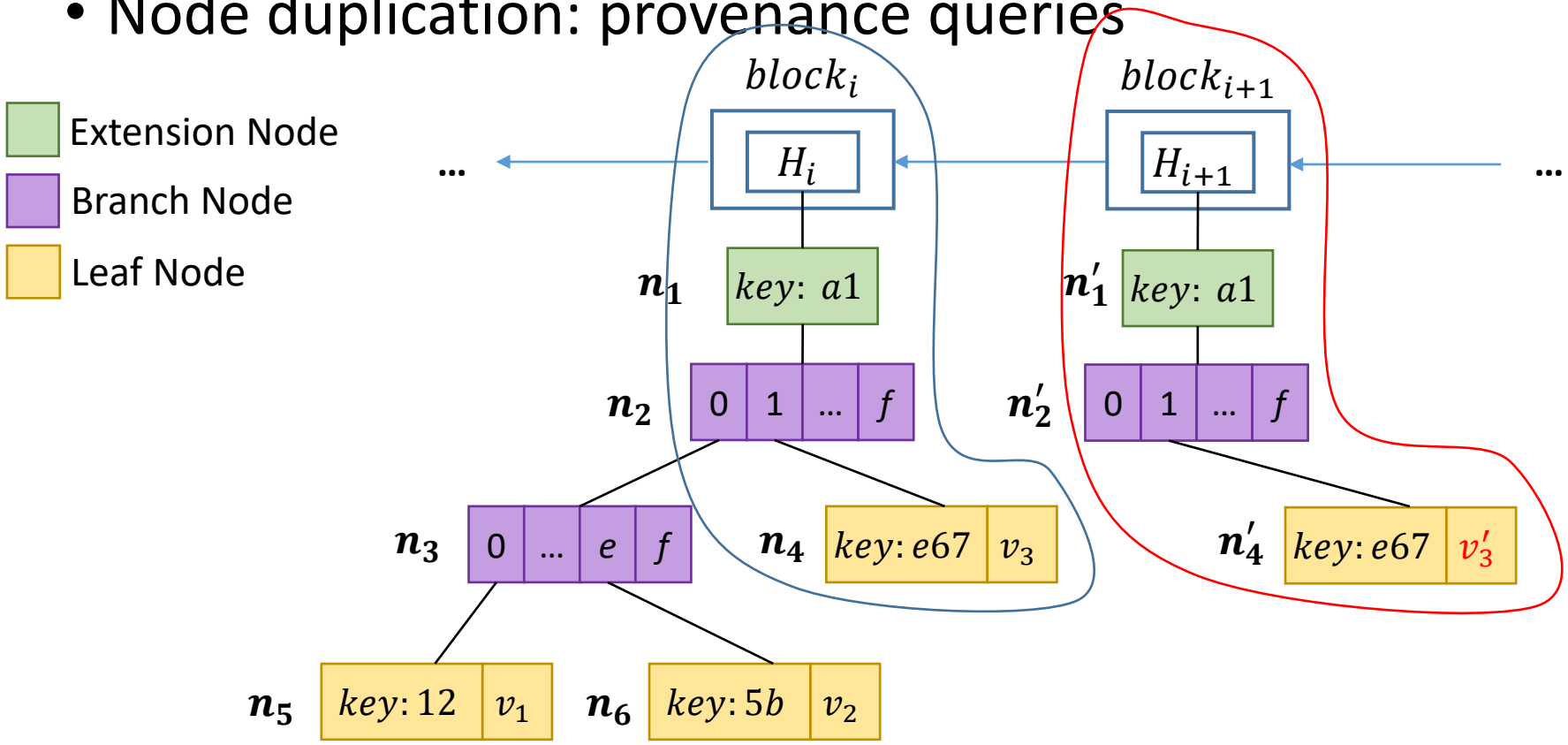
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

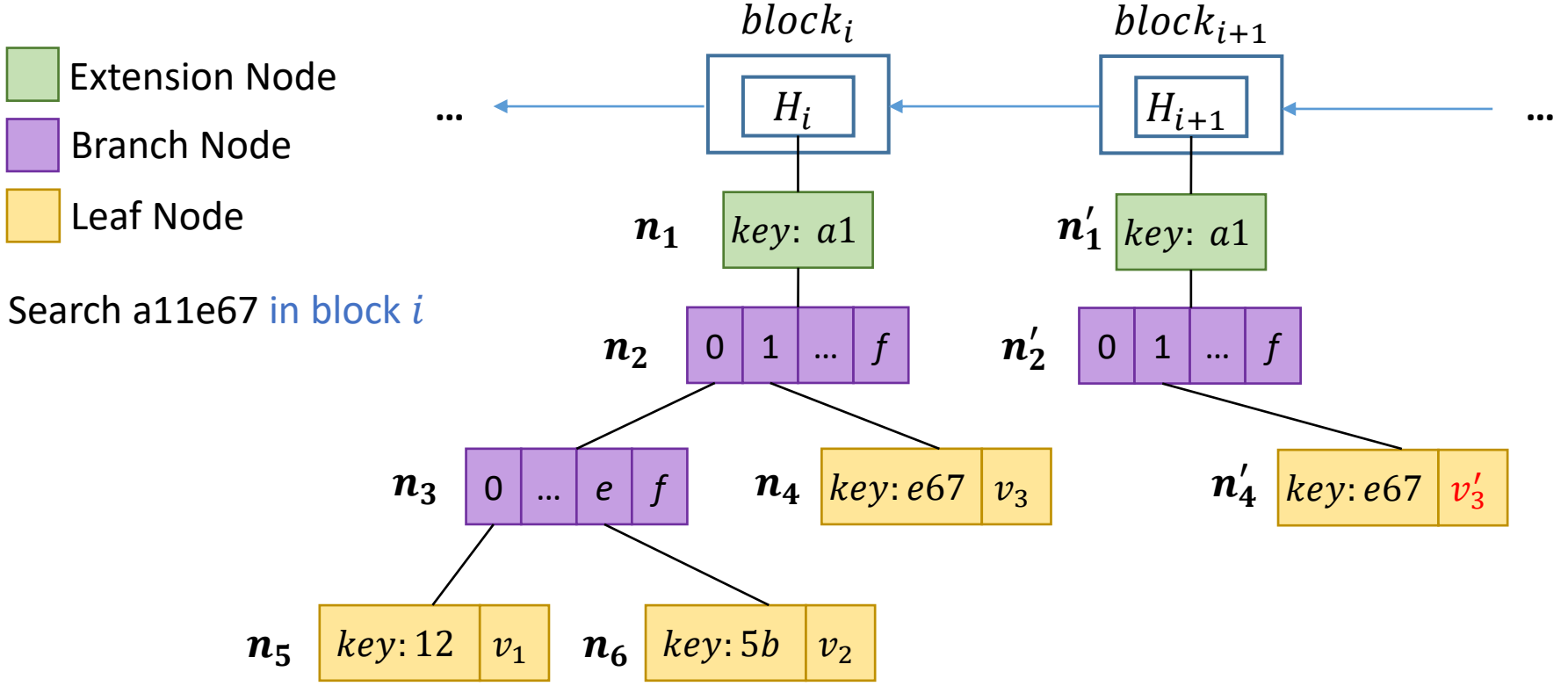
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

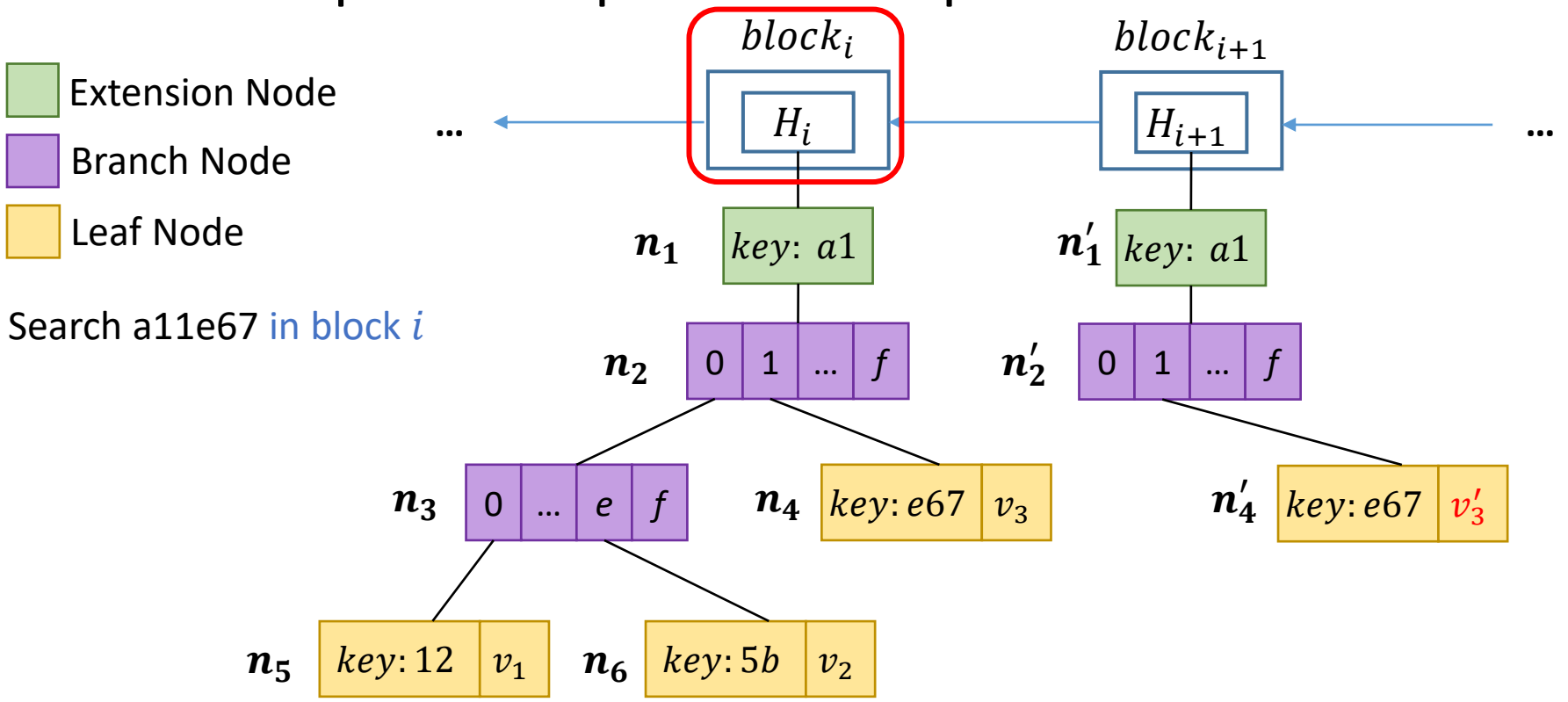
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

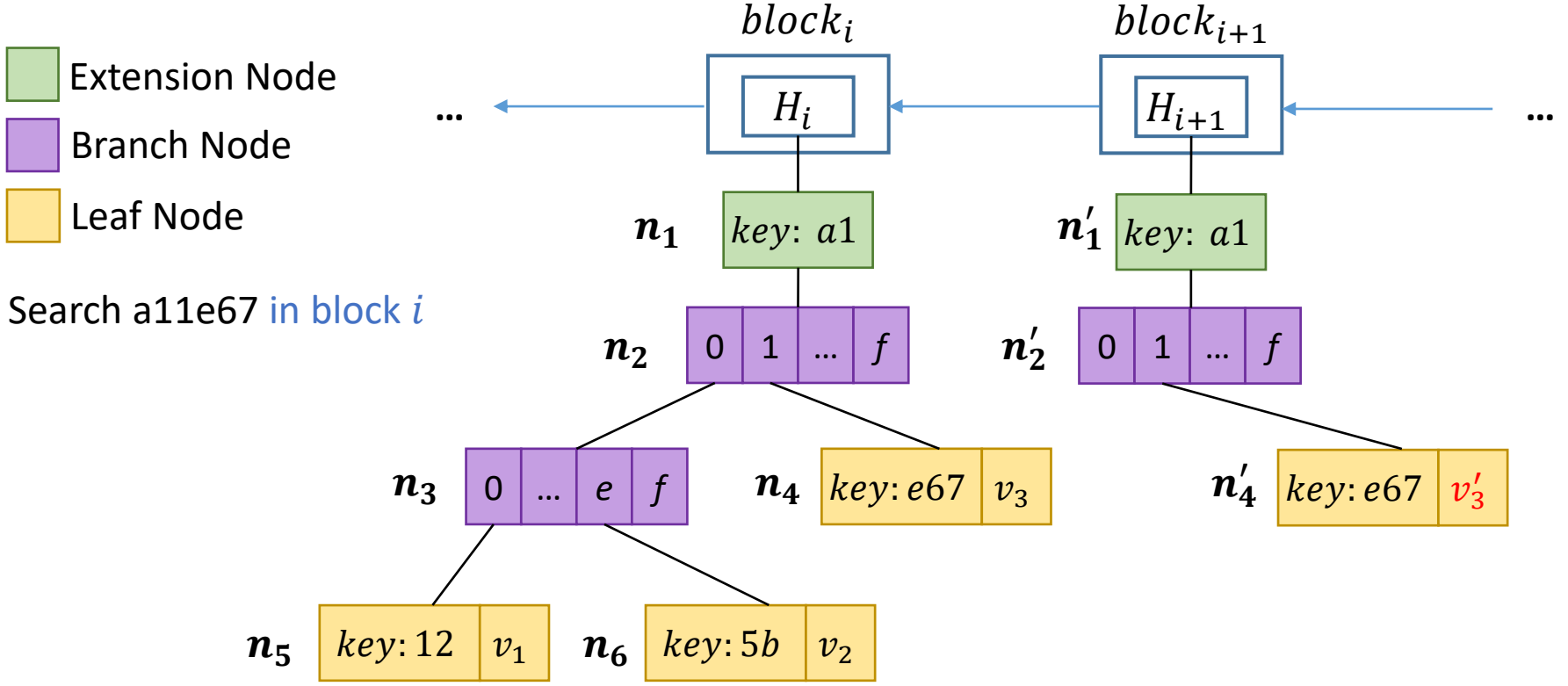
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

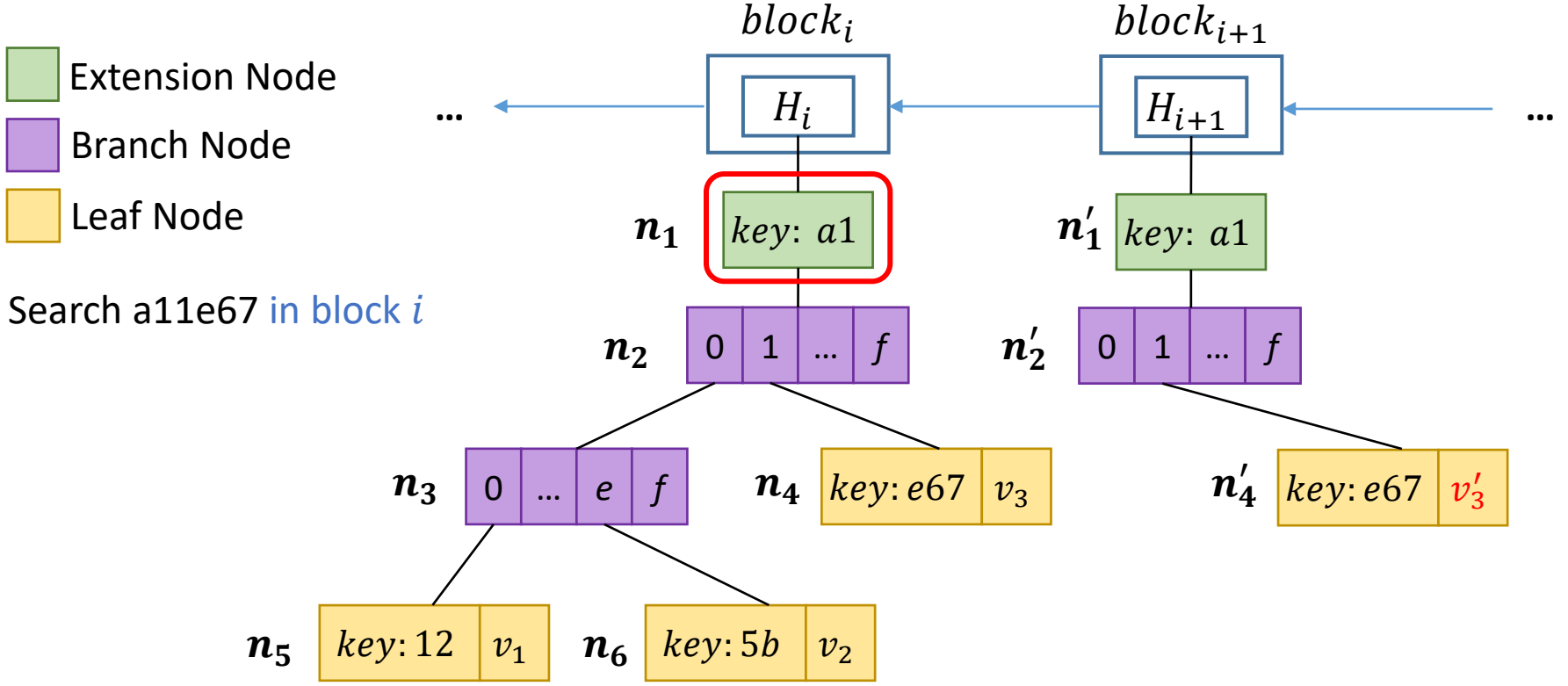
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

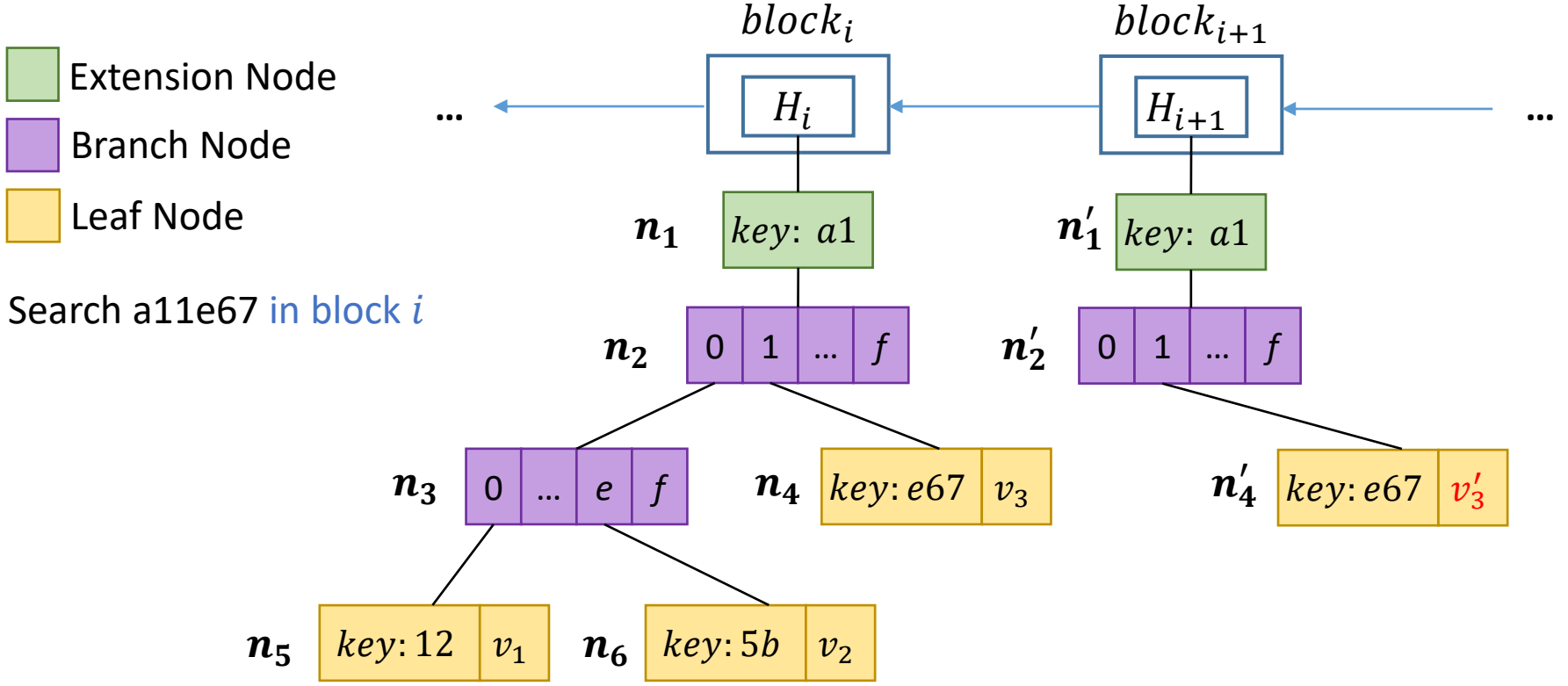
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

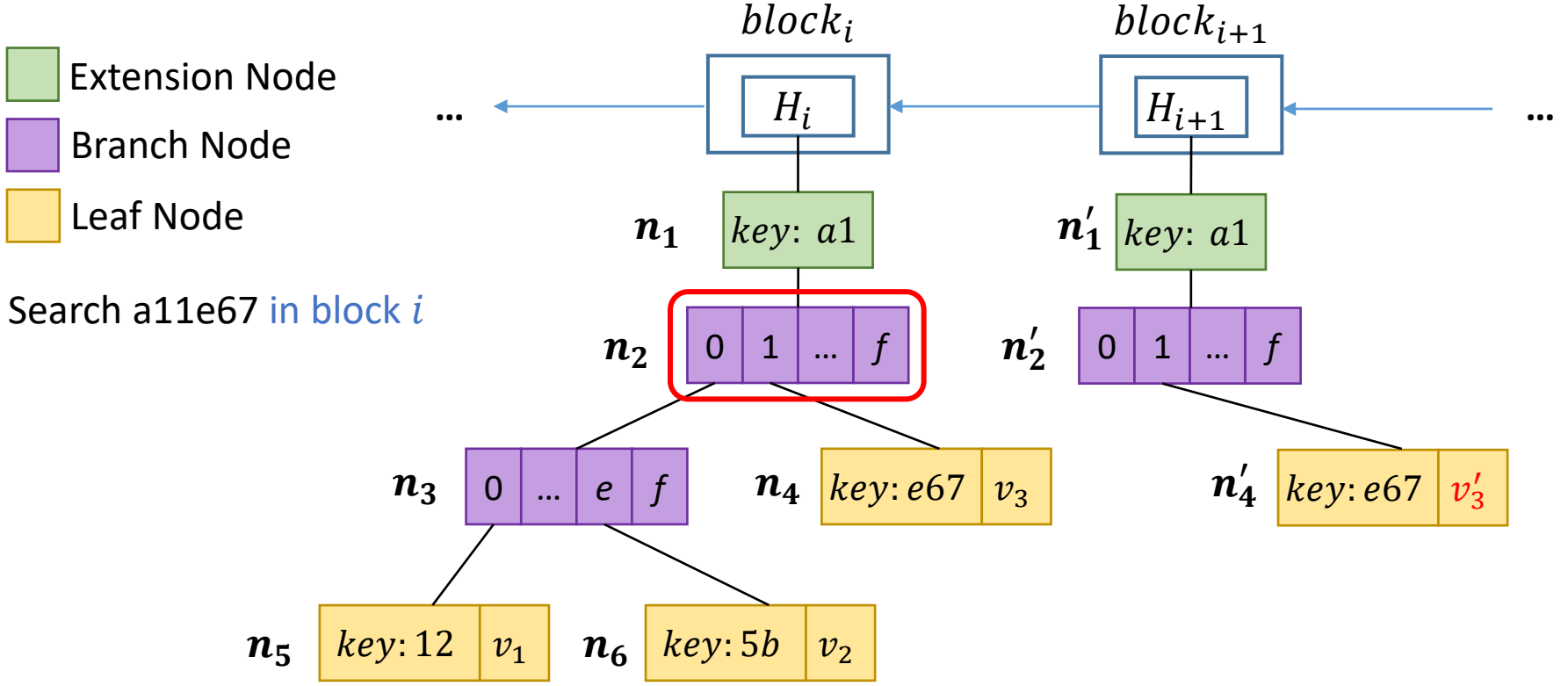
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

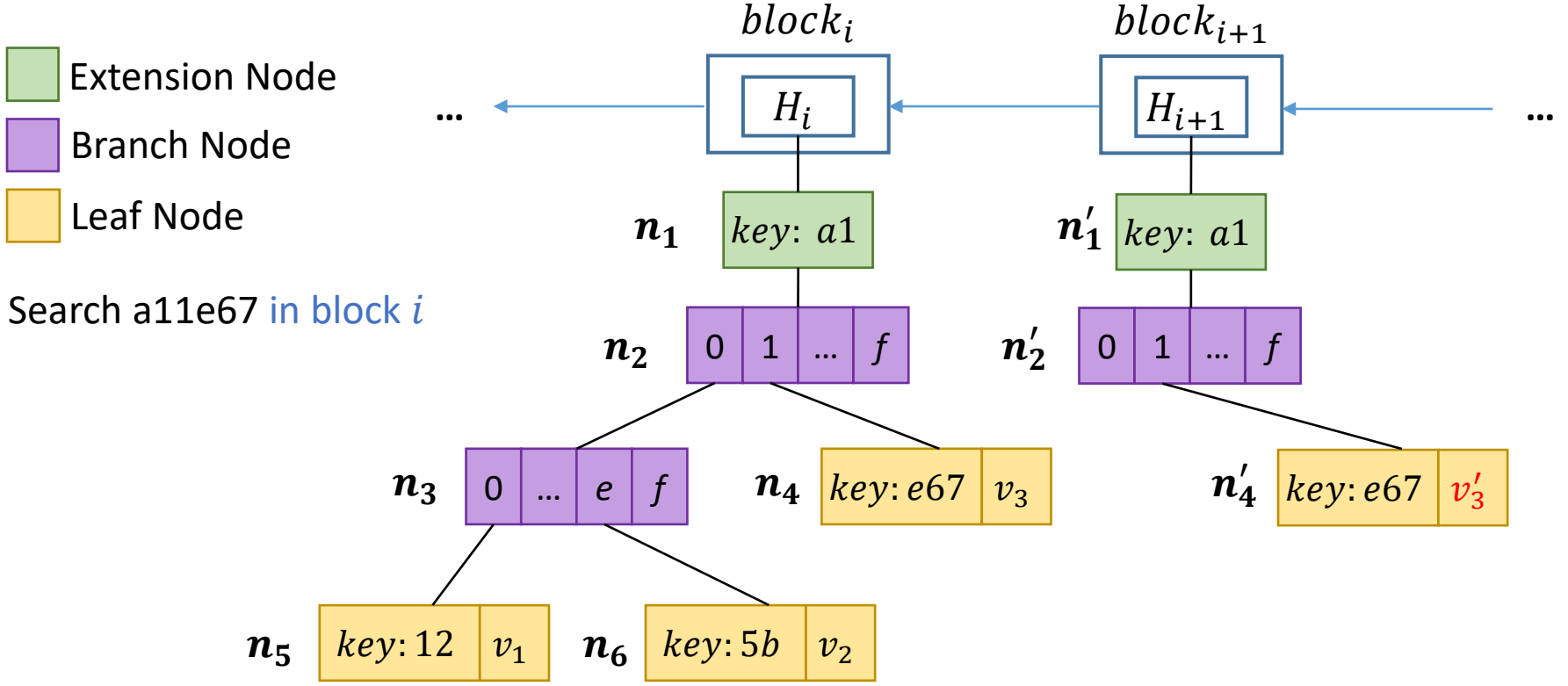
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

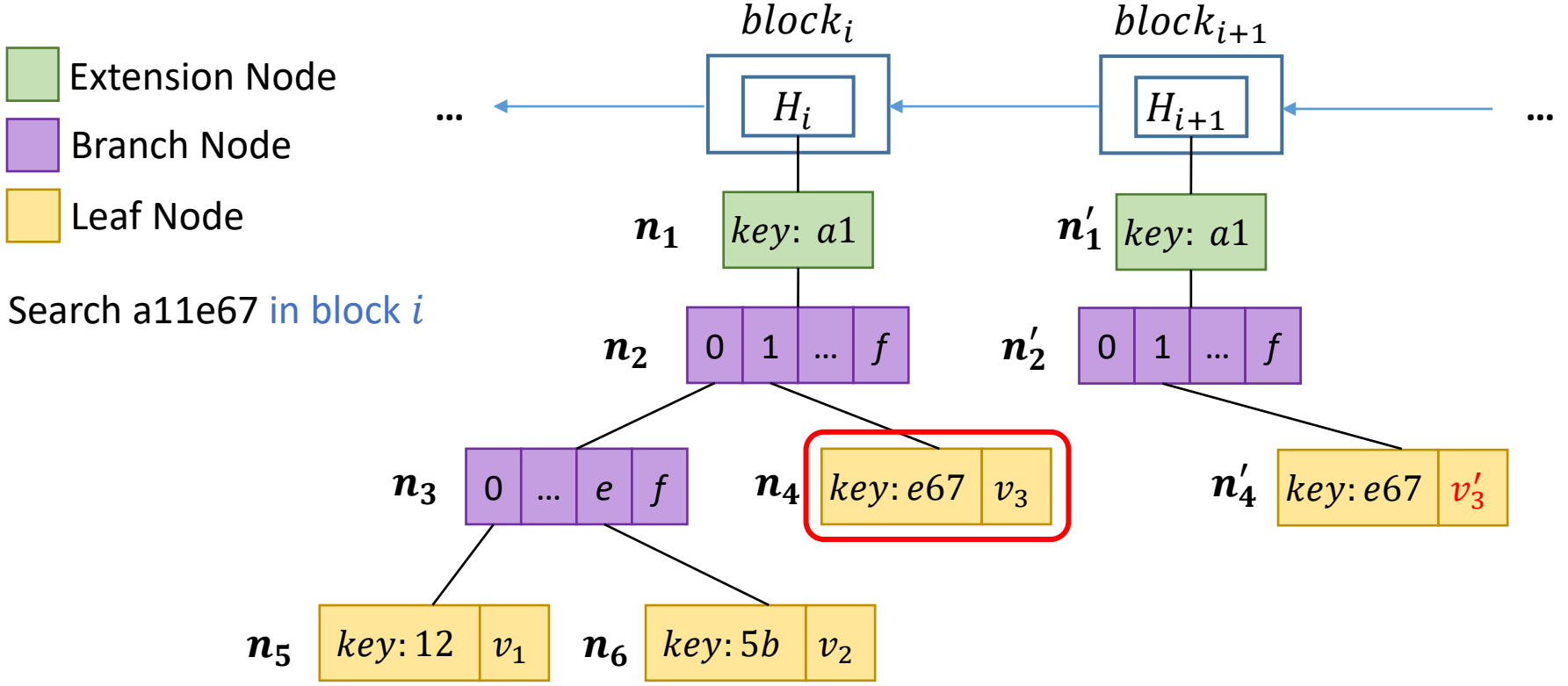
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

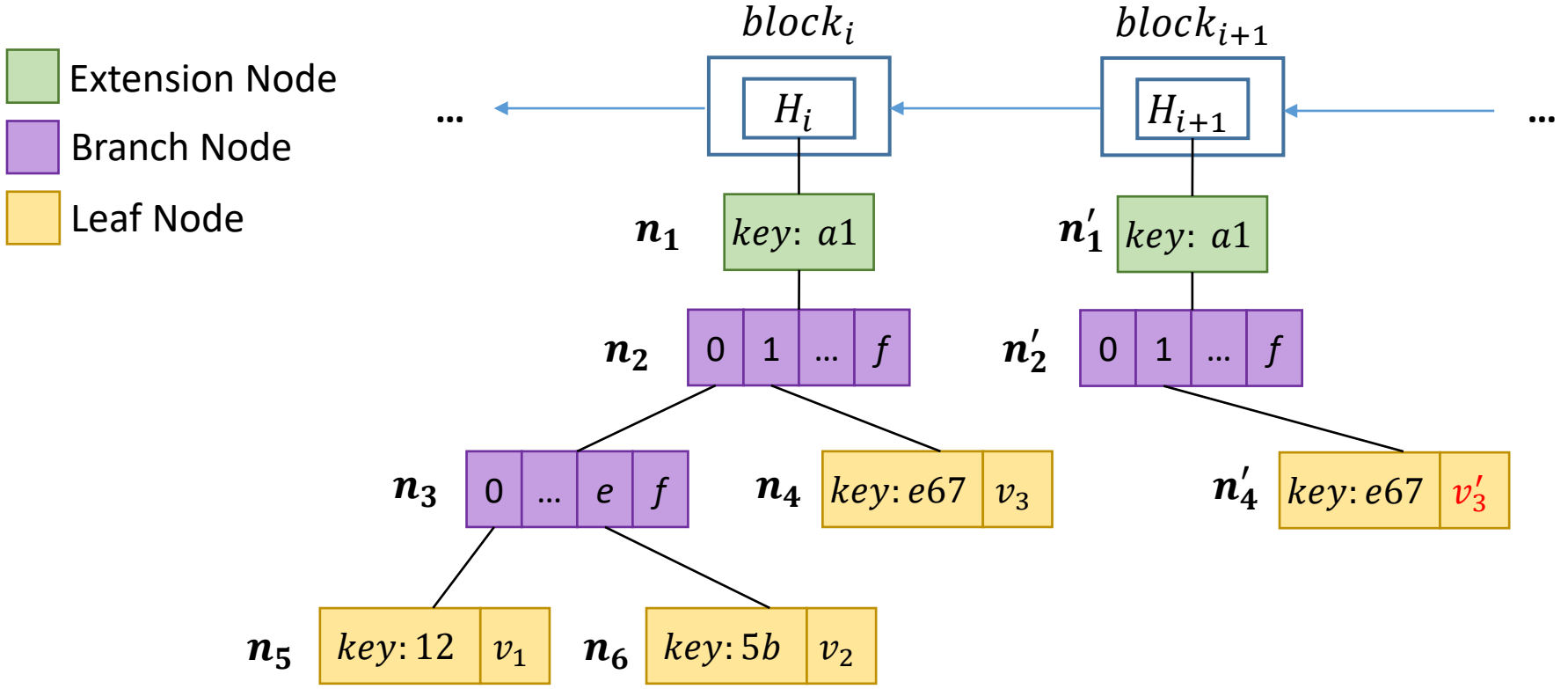
- Merkle Patricia Tree (MPT)
  - Node duplication: provenance queries



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

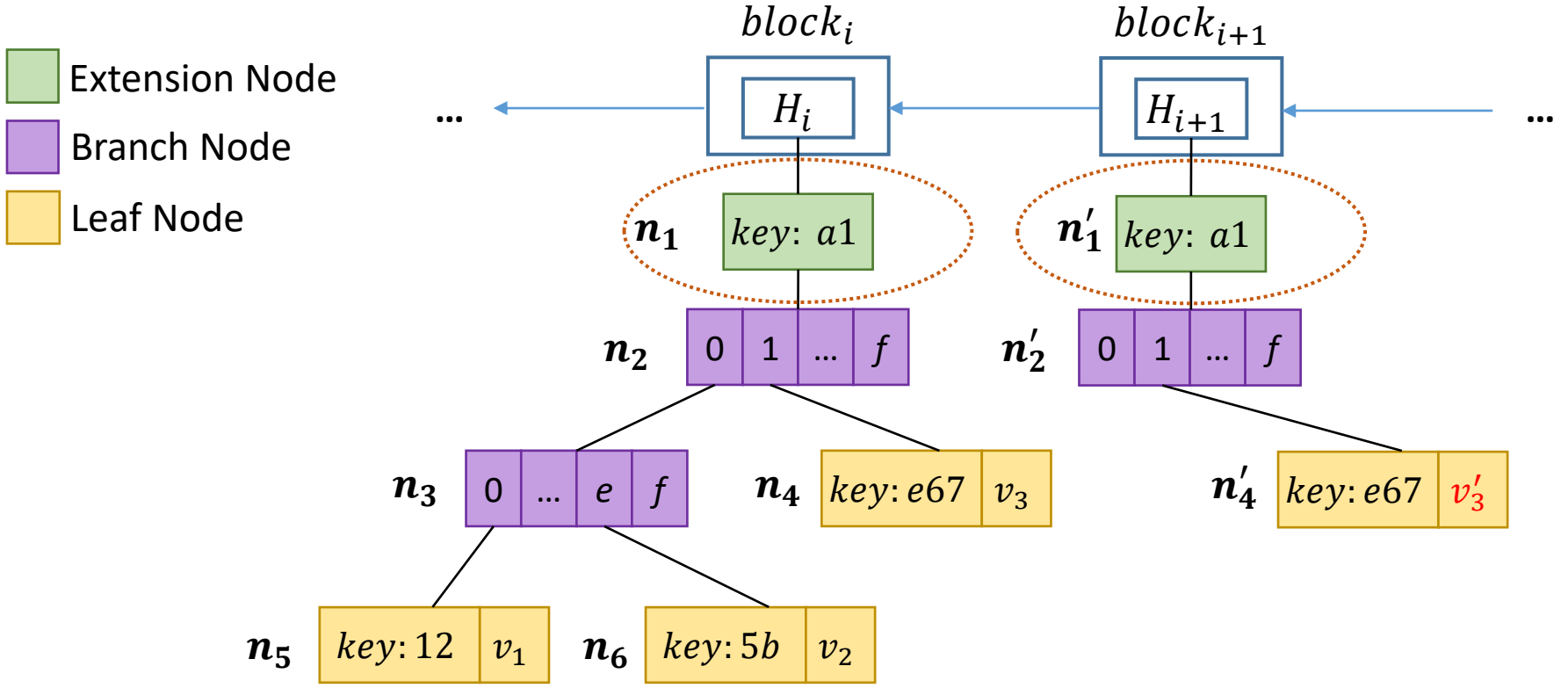
- Merkle Patricia Tree (MPT)
  - High storage overhead



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

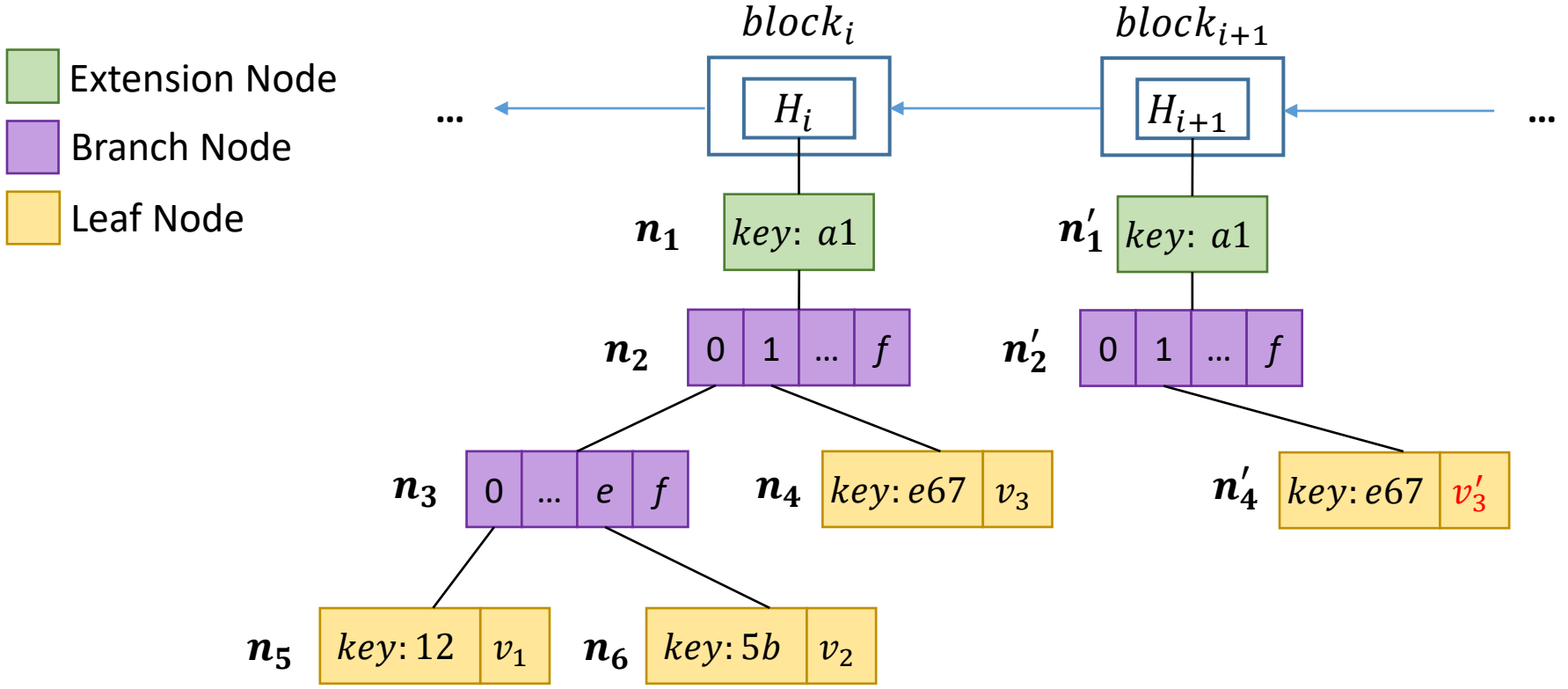
- Merkle Patricia Tree (MPT)
  - High storage overhead



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

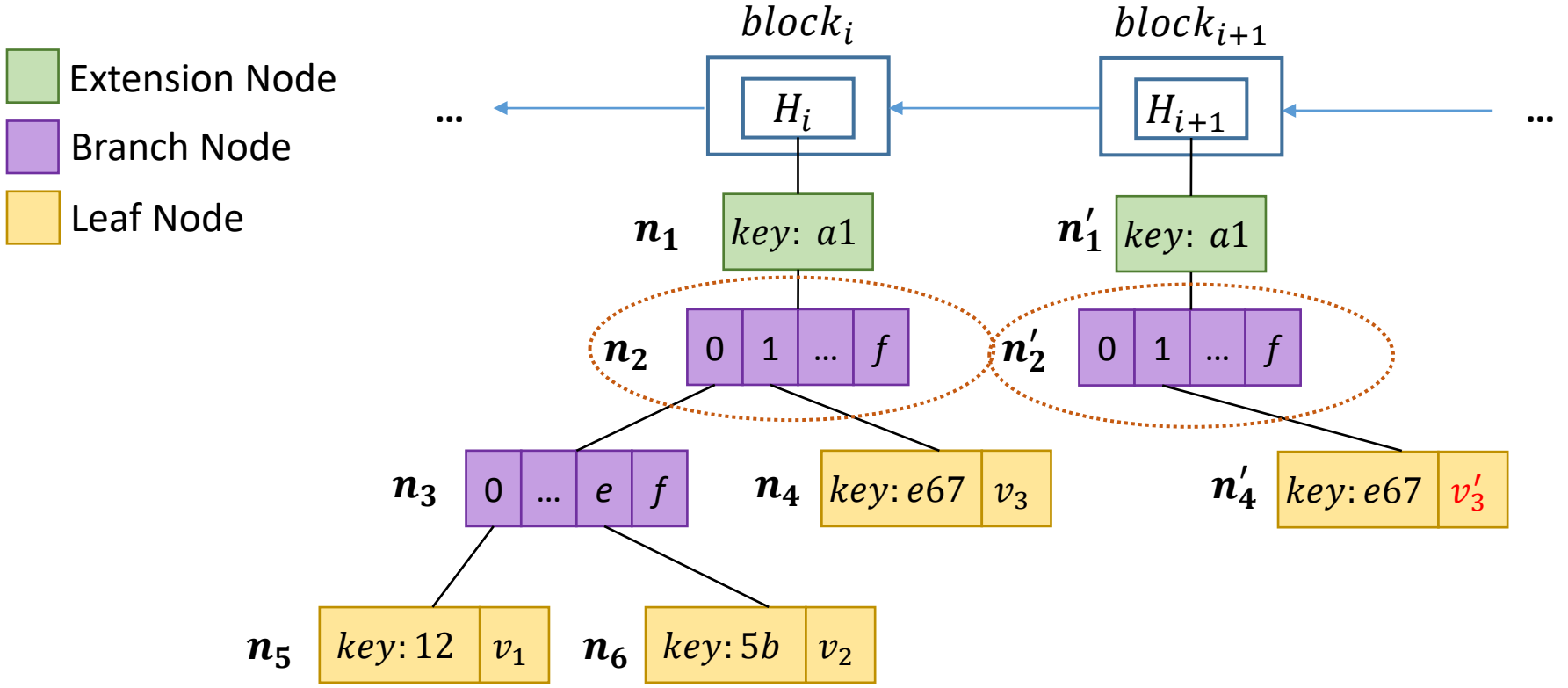
- Merkle Patricia Tree (MPT)
  - High storage overhead



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

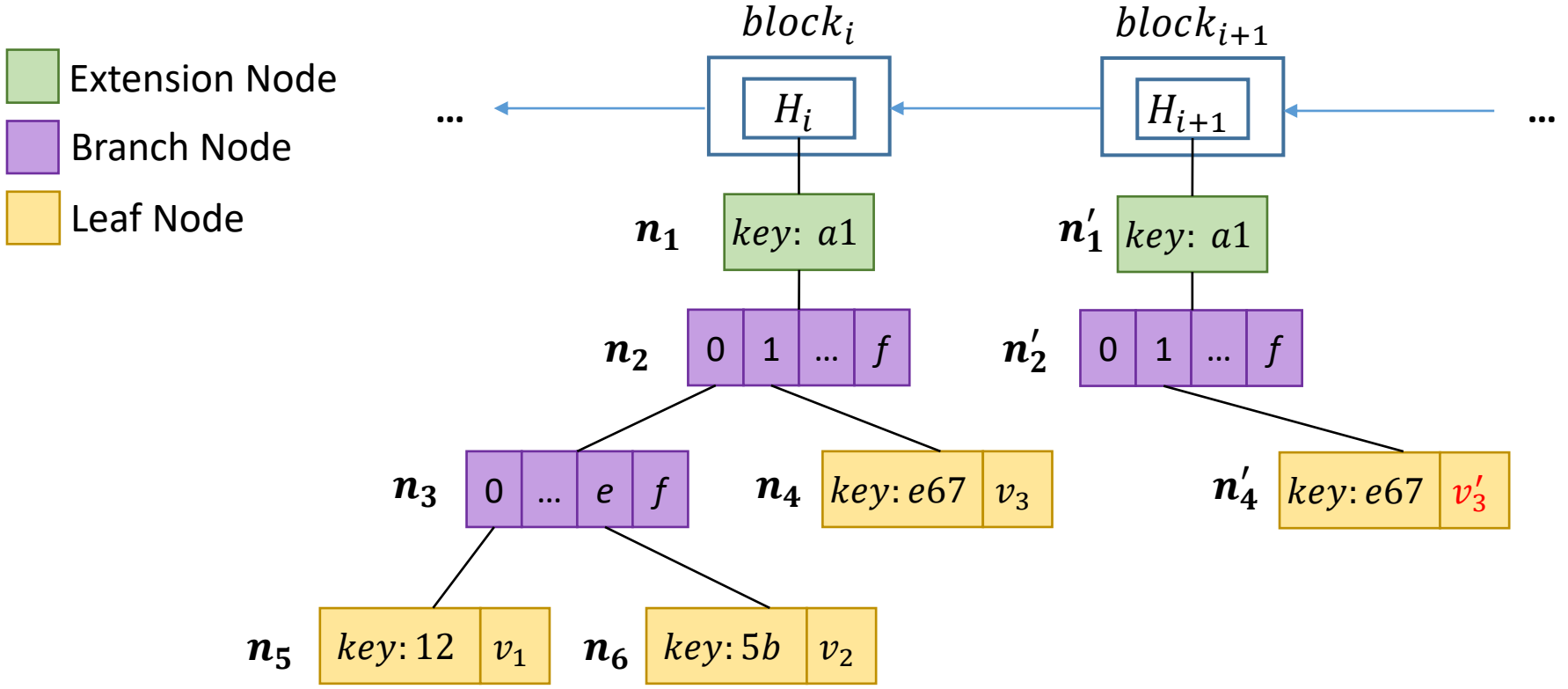
- Merkle Patricia Tree (MPT)
  - High storage overhead



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

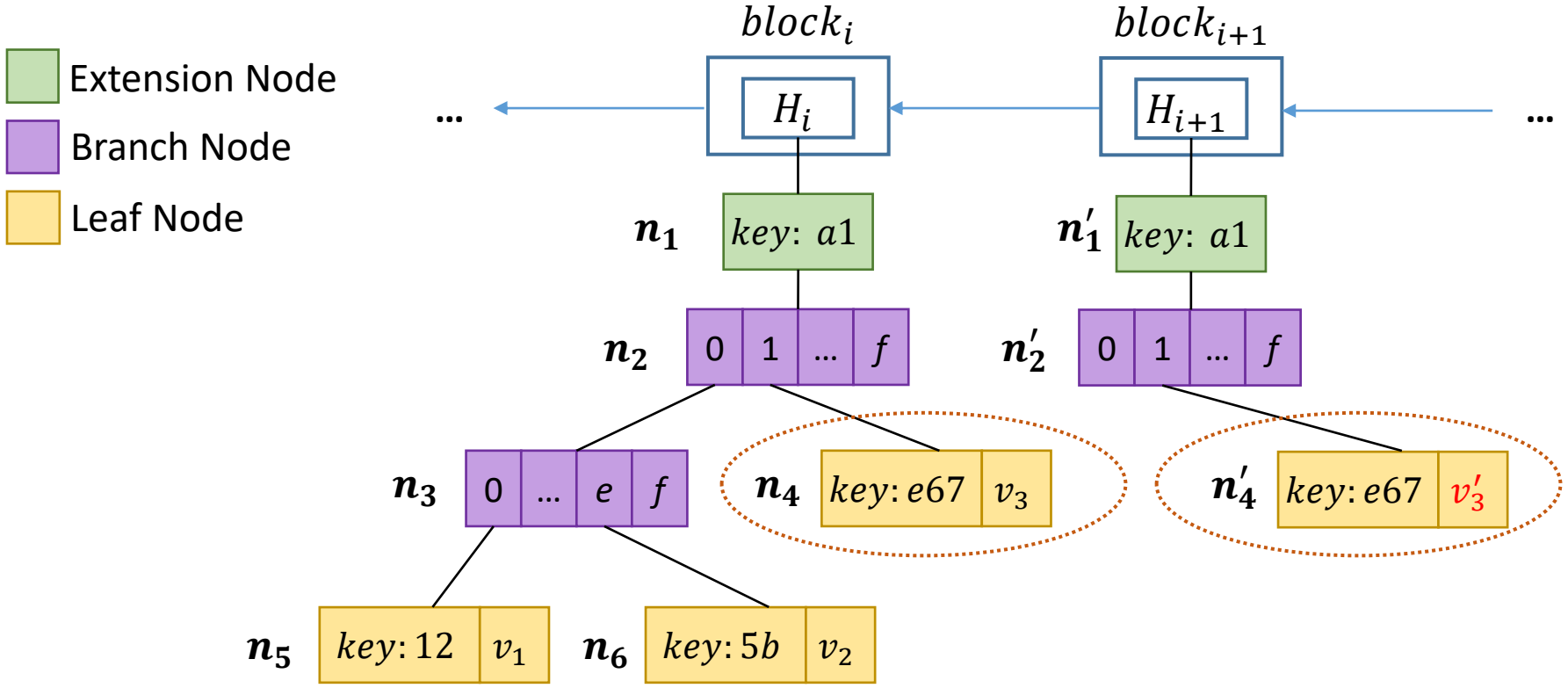
- Merkle Patricia Tree (MPT)
  - High storage overhead



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

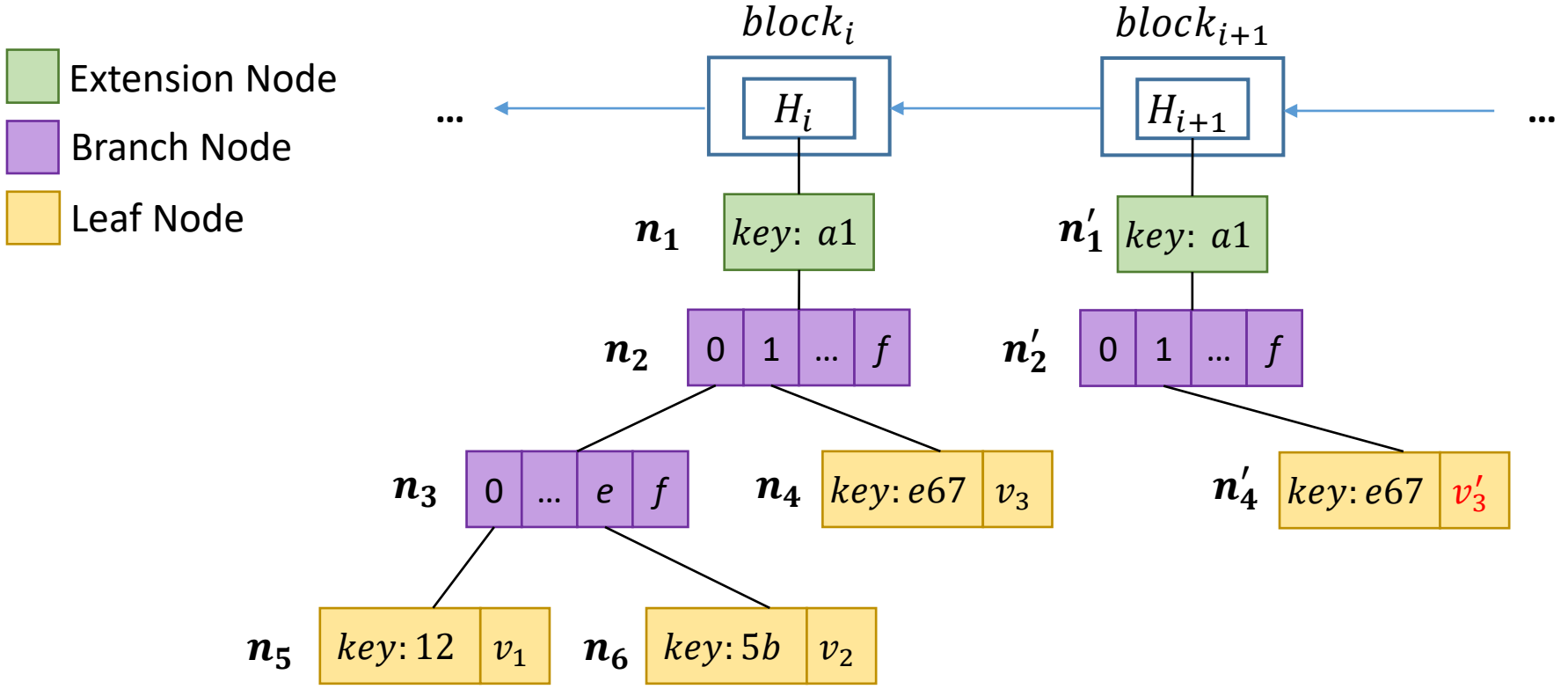
- Merkle Patricia Tree (MPT)
  - High storage overhead



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

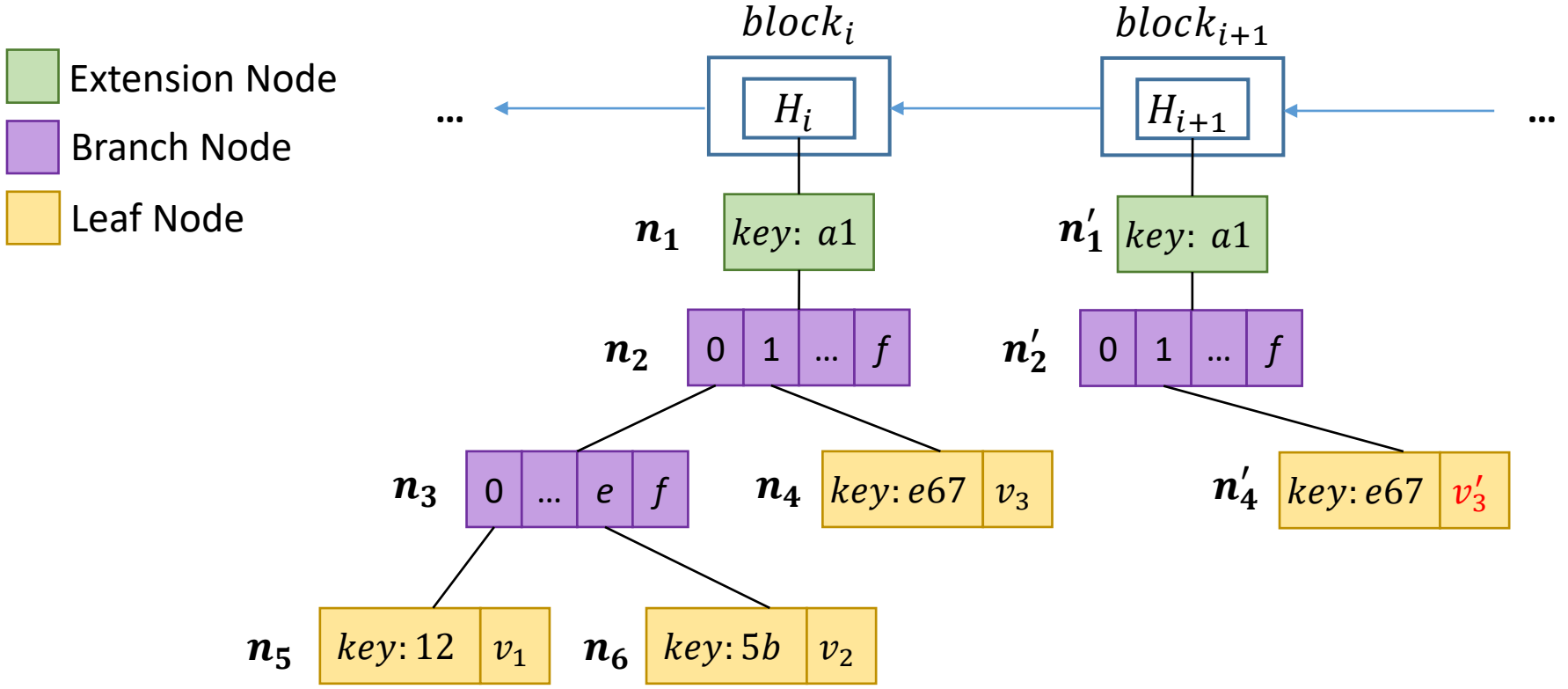
- Merkle Patricia Tree (MPT)
  - High storage overhead



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Ethereum Index

- Merkle Patricia Tree (MPT)
  - High storage overhead



addr	value
$a10012$	v_1
$a10e5b$	v_2
$a11e67$	v_3
$a11e67$	v_3'

Design a blockchain index with less storage size

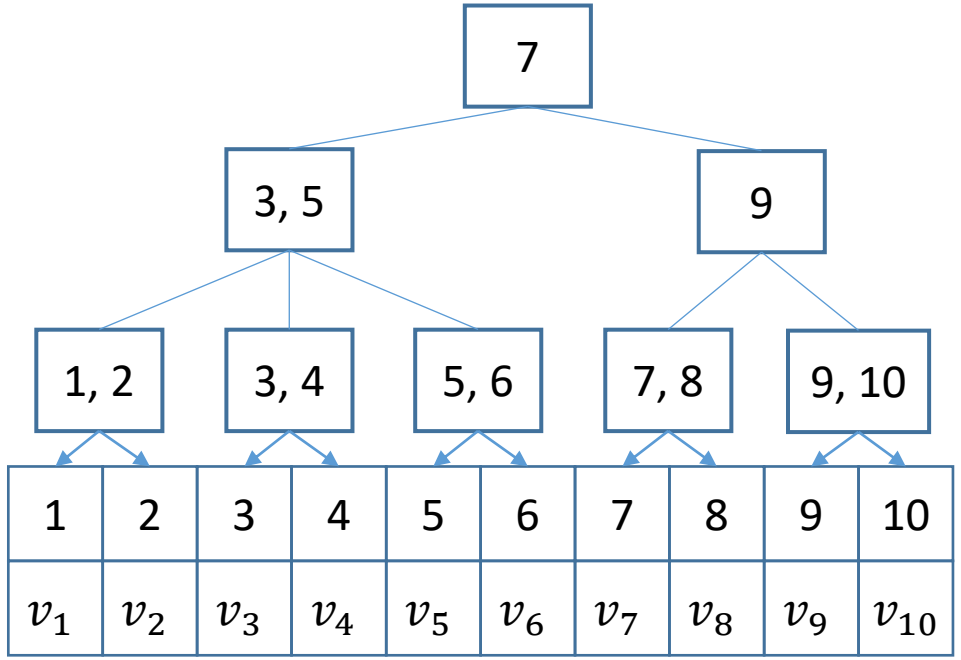
Learned Index

- 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

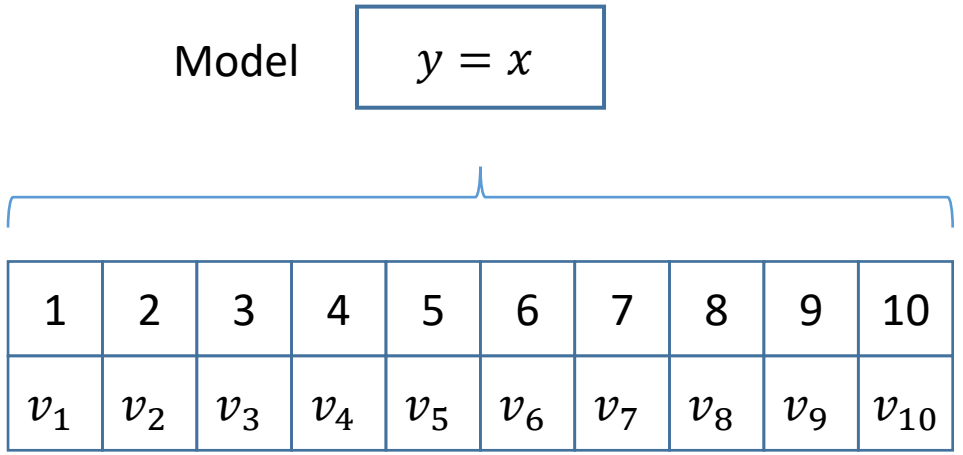
[1] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In ACM SIGMOD. 489–504.
[2] Jiacheng Wu, Yong Zhang, Shimin Chen, Jin Wang, Yu Chen, and Chunxiao Xing. 2021. Updatable learned index with precise positions. PVLDB (2021), 1276–1288.
[3] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, et al. 2020. ALEX: an updatable adaptive learned index. In ACM SIGMOD. 969–984.

Learned Index

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



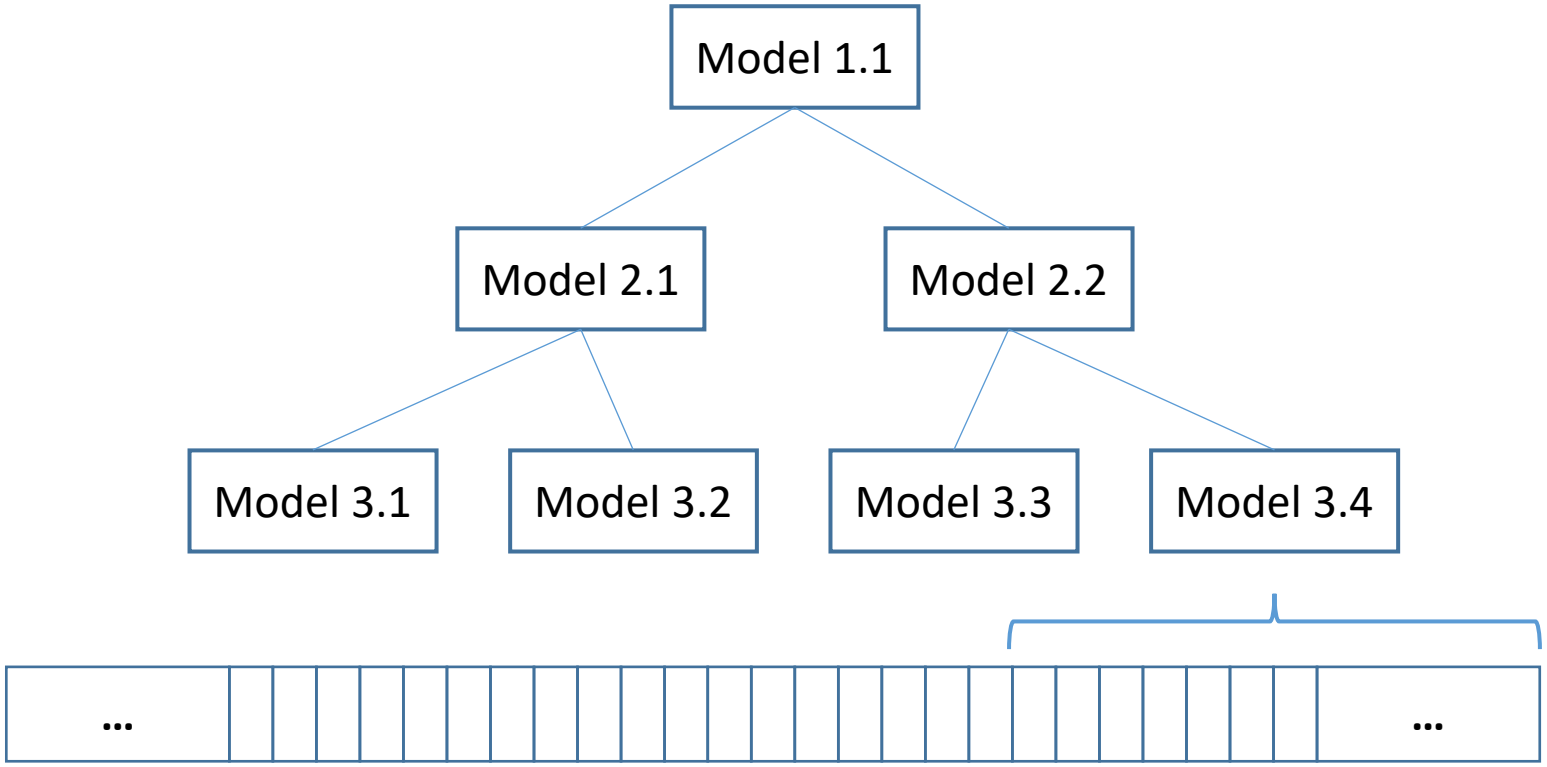
- Index storage cost: $O(n)$
- Query cost: $O(\log_f n \cdot \log_2 f)$



- Index storage cost: $O(1)$
- Query cost: $O(1)$

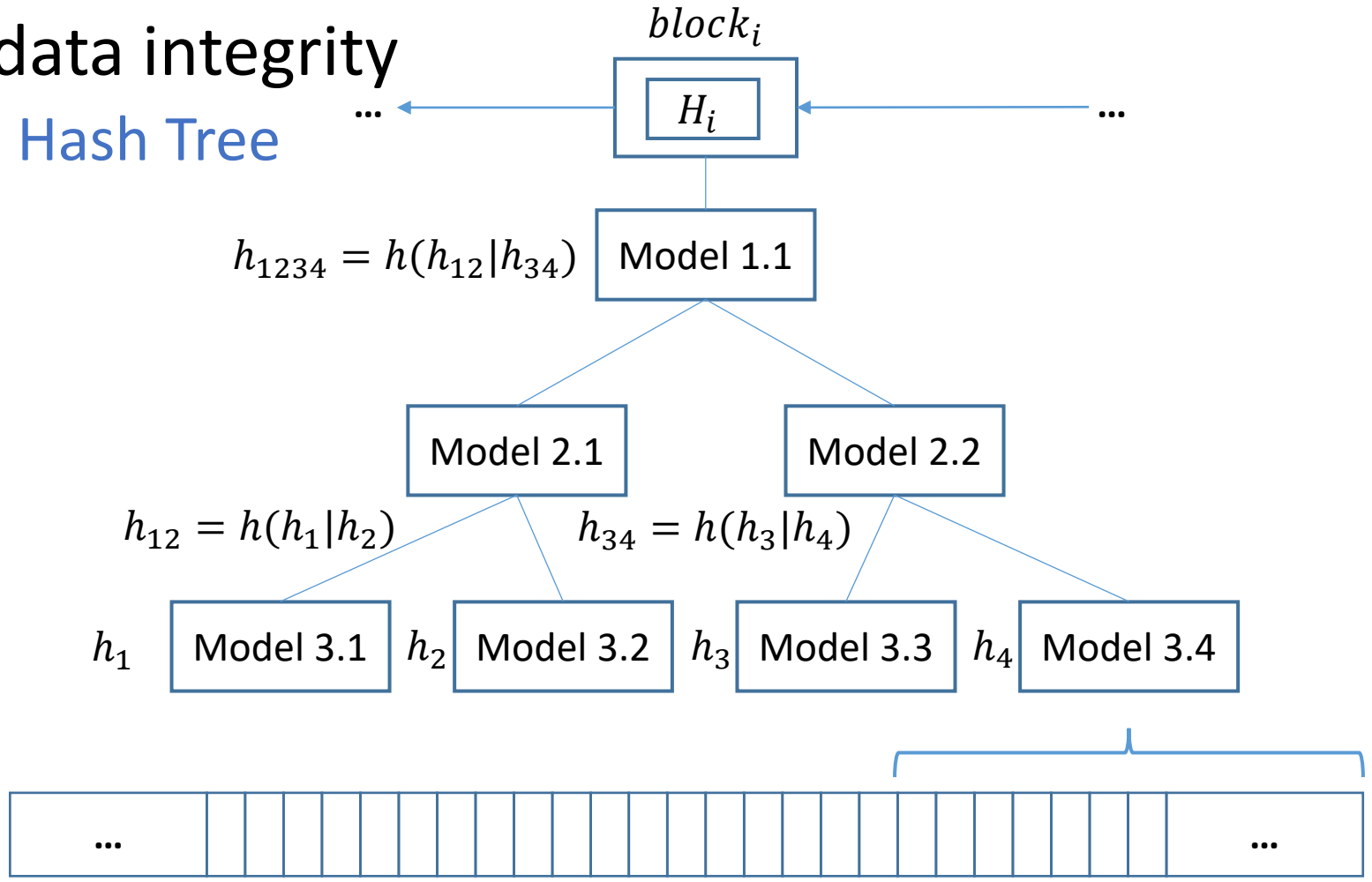
Learned Index

- More general case
  - Hierarchical models



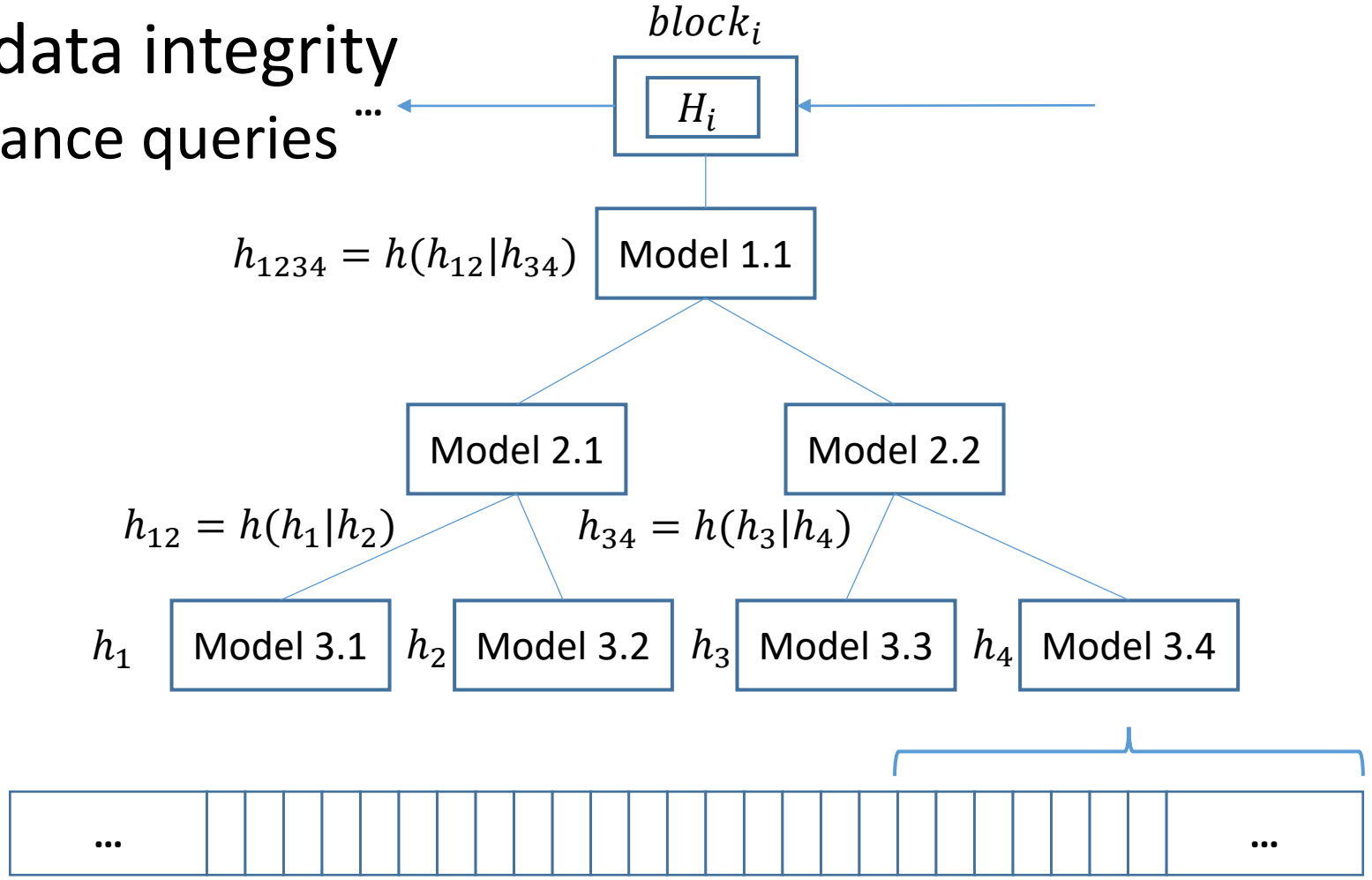
Applying Learned Index to Blockchain Storage

- Support data integrity
  - Merkle Hash Tree



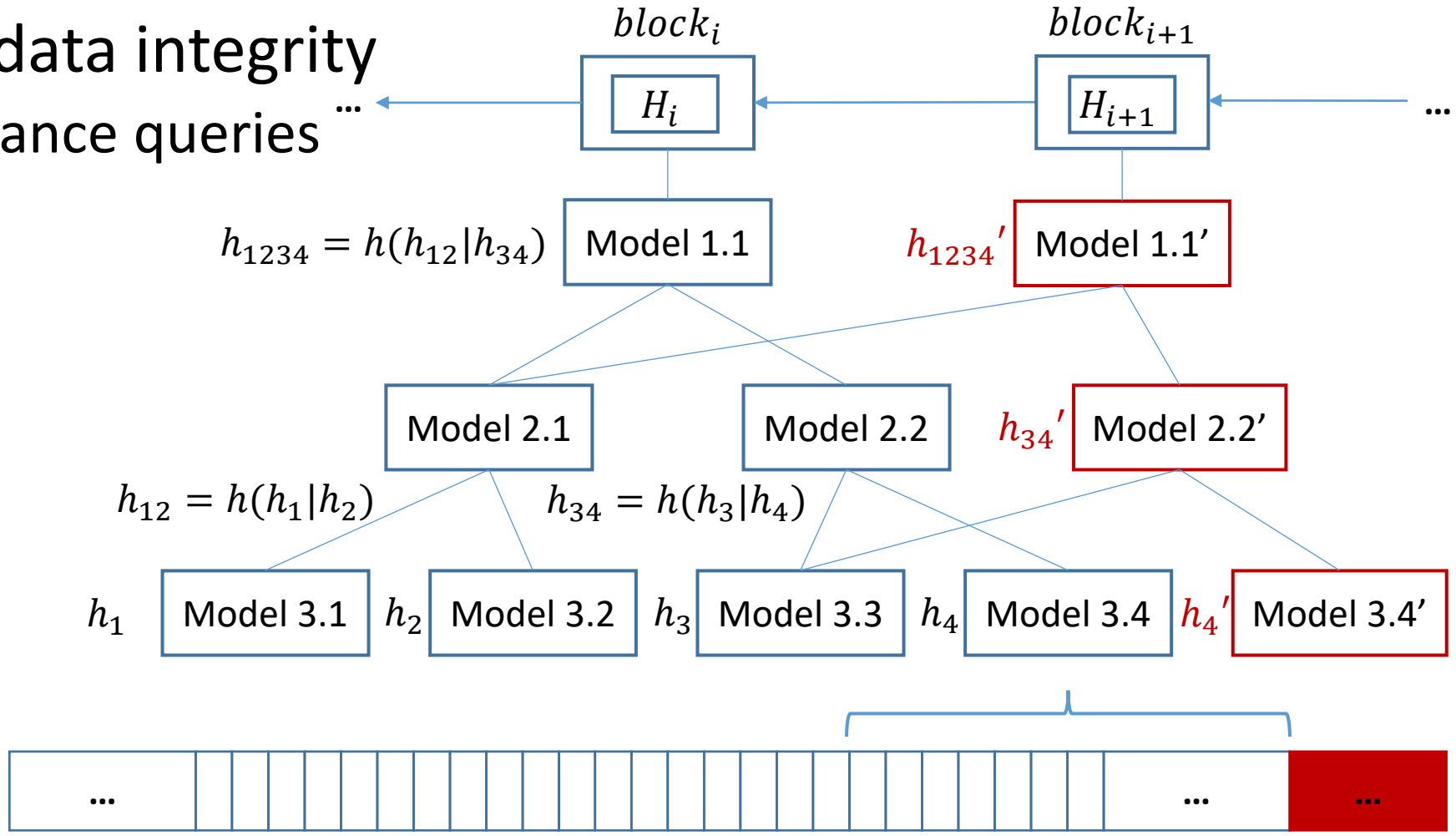
Applying Learned Index to Blockchain Storage

- Support data integrity
  - Provenance queries …



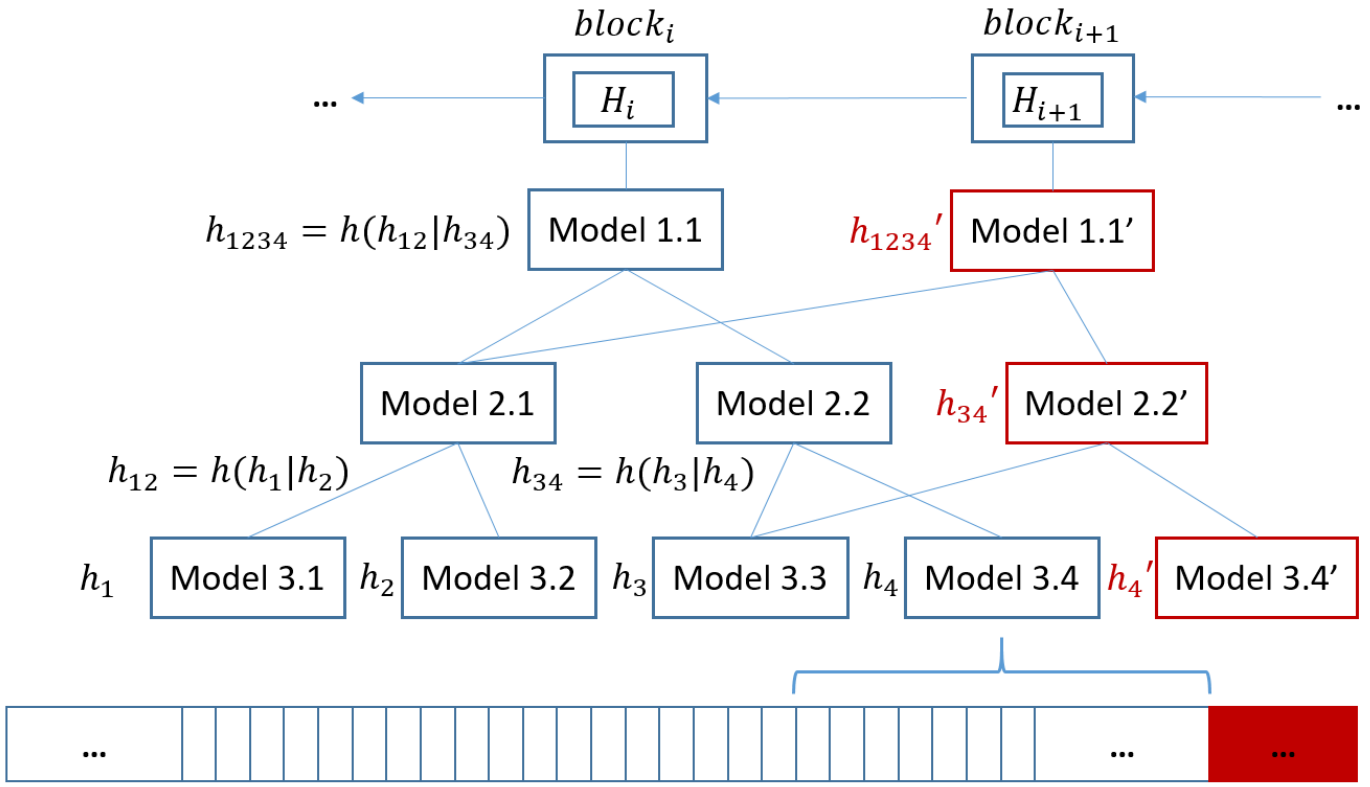
Applying Learned Index to Blockchain Storage

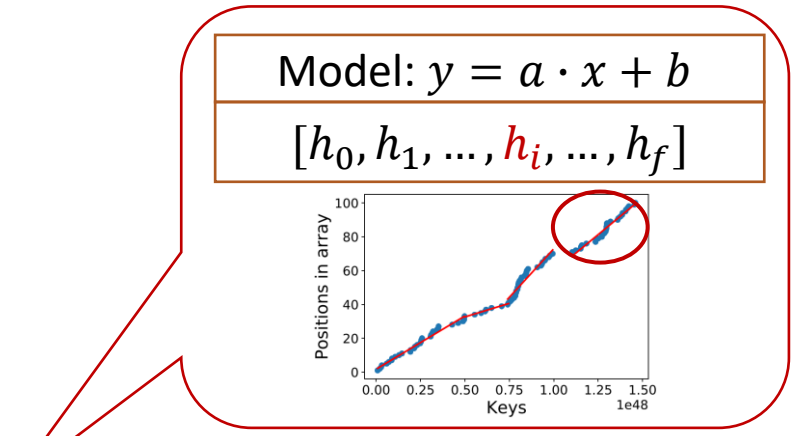
- Support data integrity
  - Provenance queries



Applying Learned Index to Blockchain Storage

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





f could be very large

1 hash update -> log # nodes duplication

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

$\boldsymbol{block_{i+1}}$

State addr k_1

blk	value
1	v_1^1
...	...
i	v_1^i

State addr k_2

blk	value
3	v_2^3
...	...
i	v_2^i

State addr k_3

blk	value
2	v_3^2
...	...
i	v_3^i
$i+1$	v_3^{i+1}

Our Design: COLE

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

$\boldsymbol{block_{i+1}}$

State addr k_1

blk	value
1	v_1^1
...	...
i	v_1^i

State addr k_2

blk	value
3	v_2^3
...	...
i	v_2^i

State addr k_3

blk	value
2	v_3^2
...	...
i	v_3^i
$i+1$	v_3^{i+1}

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

$\boldsymbol{block_{i+1}}$

State addr k_1

blk	value
1	v_1^1
...	...
i	v_1^i

State addr k_2

blk	value
3	v_2^3
...	...
i	v_2^i

State addr k_3

blk	value
2	v_3^2
...	...
i	v_3^i
$i+1$	v_3^{i+1}

Store them together:

$k_1, 1, v_1^1$	...	k_1, i, v_1^i

Our Design: COLE

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

$\boldsymbol{block_{i+1}}$ State addr k_1

blk	value
1	v_1^1
...	...
i	v_1^i

State addr k_2

blk	value
3	v_2^3
...	...
i	v_2^i

State addr k_3

blk	value
2	v_3^2
...	...
i	v_3^i
$i+1$	v_3^{i+1}

Store them together:

$k_1, 1, v_1^1$	...	k_1, i, v_1^i	$k_2, 3, v_2^3$	...	k_2, i, v_2^i

Our Design: COLE

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

$\boldsymbol{block_{i+1}}$

State addr k_1

blk	value
1	v_1^1
...	...
i	v_1^i

State addr k_2

blk	value
3	v_2^3
...	...
i	v_2^i

State addr k_3

blk	value
2	v_3^2
...	...
i	v_3^i
$i+1$	v_3^{i+1}

Store them together:

$k_1, 1, v_1^1$	...	k_1, i, v_1^i	$k_2, 3, v_2^3$	...	k_2, i, v_2^i	$k_3, 2, v_3^2$	...	k_3, i, v_3^i	$k_3, i+1, v_3^{i+1}$

Our Design: COLE

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

$\boldsymbol{block_{i+1}}$

State addr k_1

blk	value
1	v_1^1
...	...
i	v_1^i

State addr k_2

blk	value
3	v_2^3
...	...
i	v_2^i

State addr k_3

blk	value
2	v_3^2
...	...
i	v_3^i
$i+1$	v_3^{i+1}

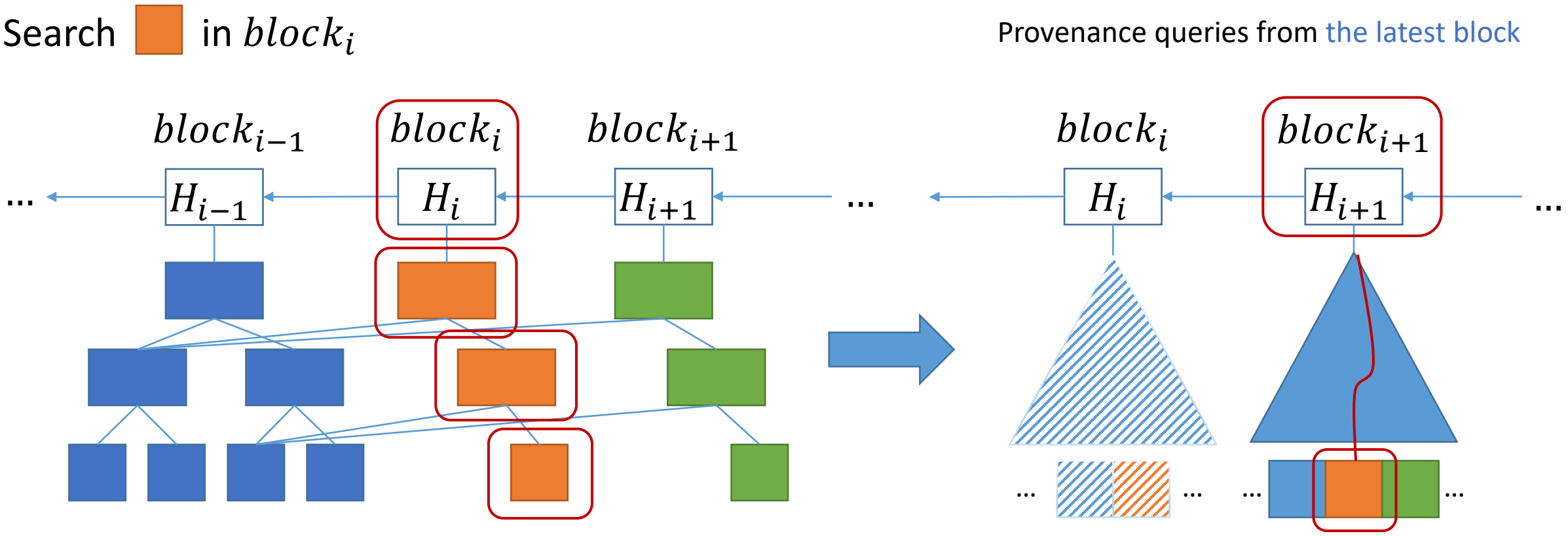
Store them together:

$k_1, 1, v_1^1$	...	k_1, i, v_1^i	$k_2, 3, v_2^3$	...	k_2, i, v_2^i	$k_3, 2, v_3^2$	...	k_3, i, v_3^i	$k_3, i+1, v_3^{i+1}$

Easy to index, easy to search ☺

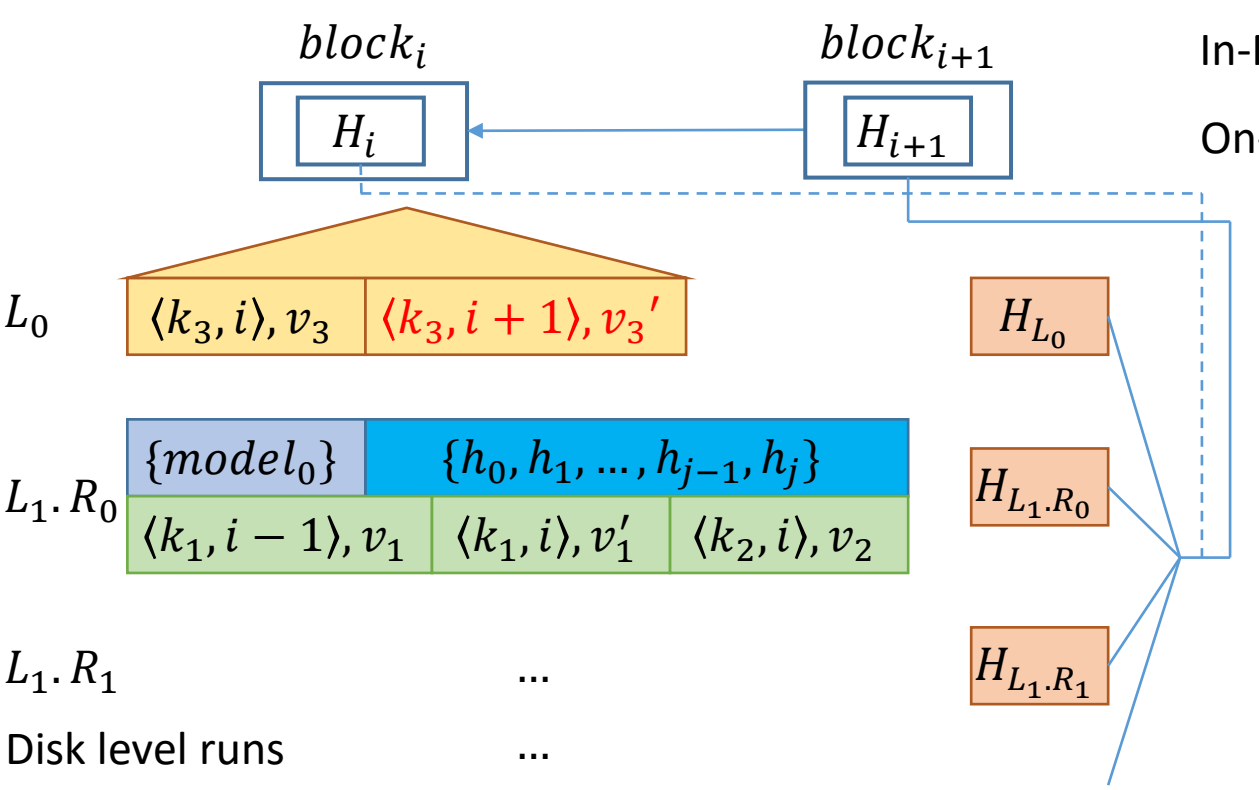
Our Design: COLE

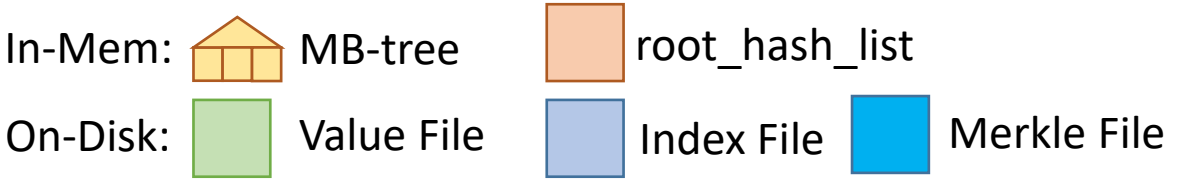
- Column-based Learned Storage for Blockchain



Our Design: COLE

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

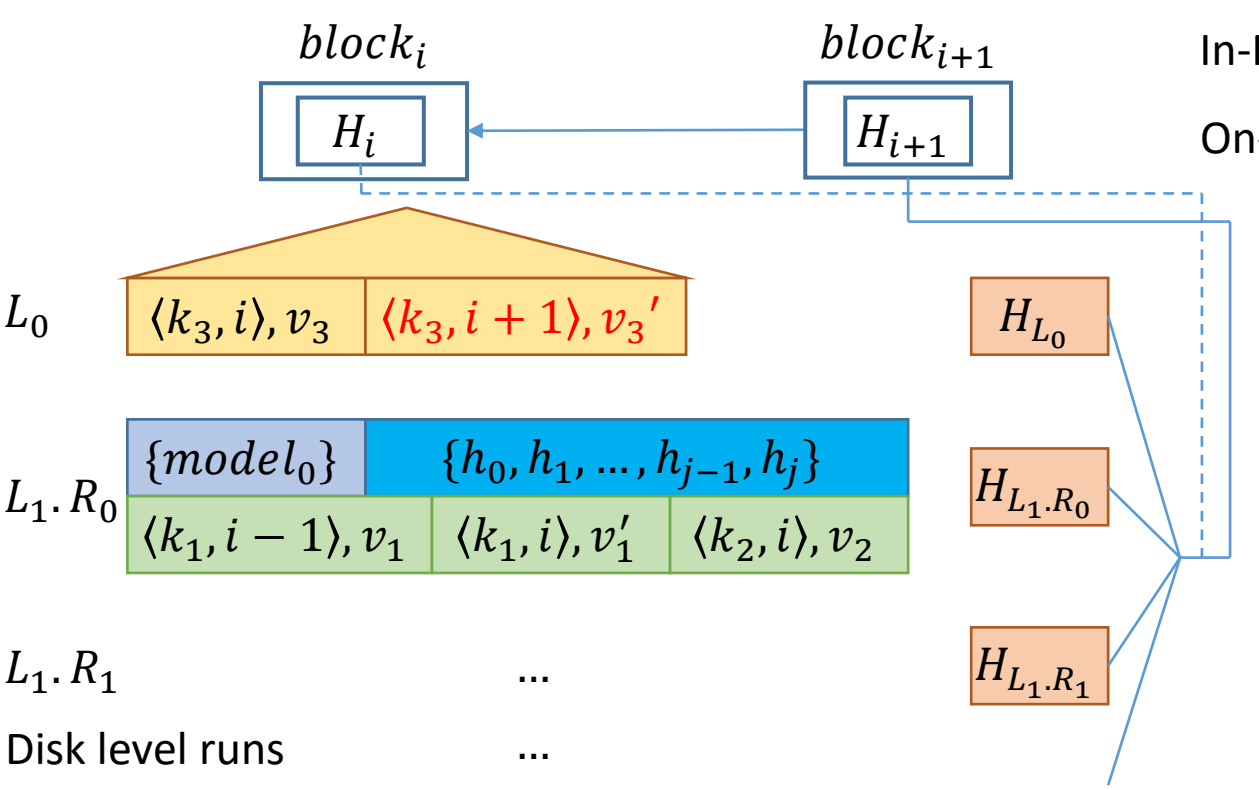


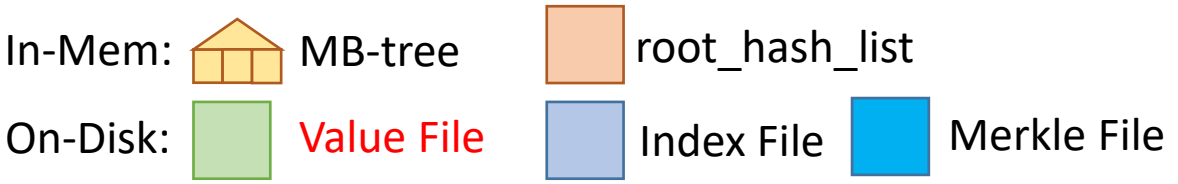


addr	blk	value
k_1	$i-1$	v_1
k_1	i	v_1'
k_2	i	v_2
k_3	i	v_3
k_3	$i+1$	v_3'

Our Design: COLE

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

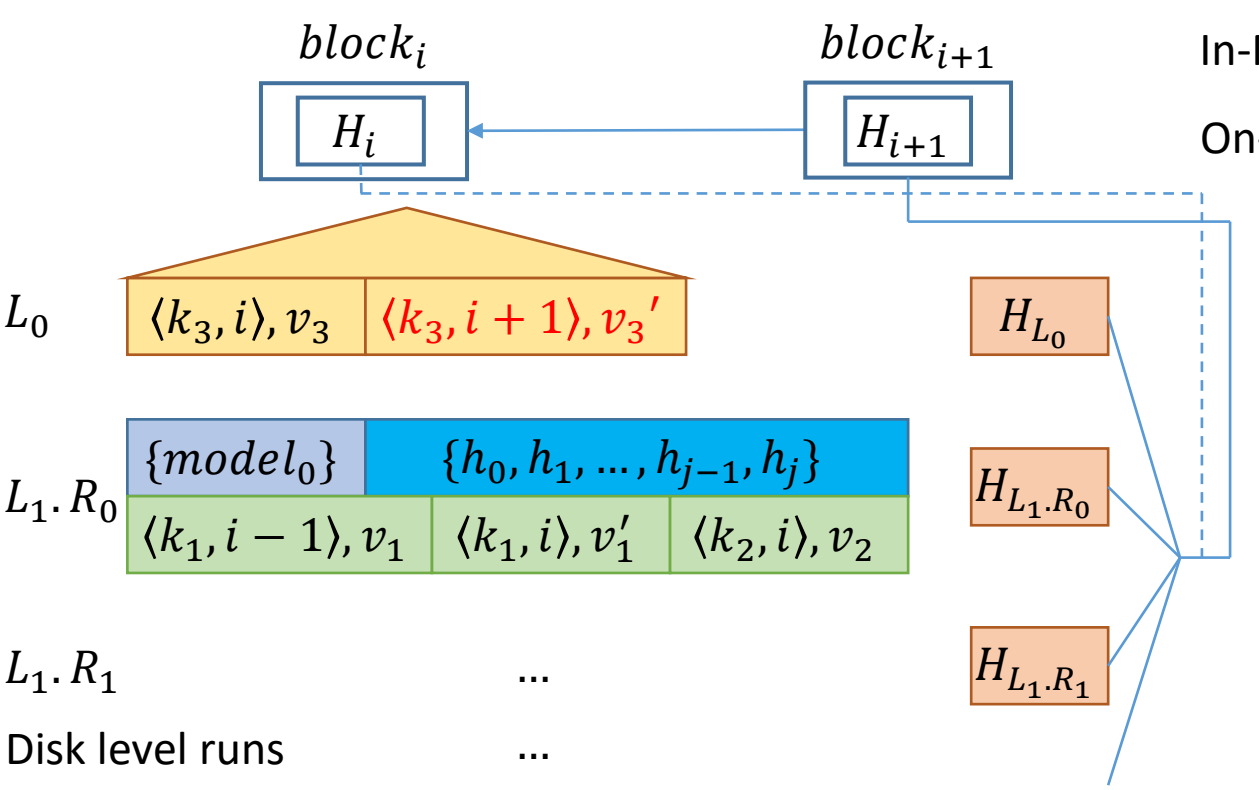


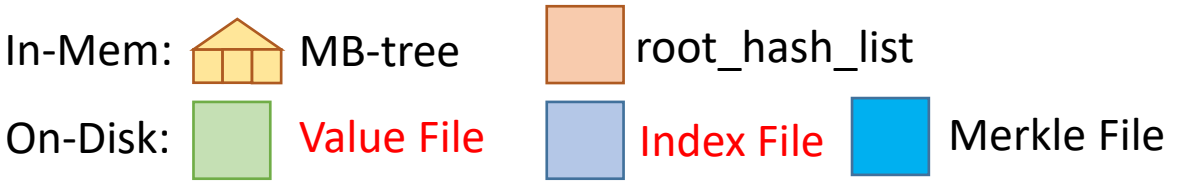


addr	blk	value
k_1	$i-1$	v_1
k_1	i	v_1'
k_2	i	v_2
k_3	i	v_3
k_3	$i+1$	v_3'

Our Design: COLE

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

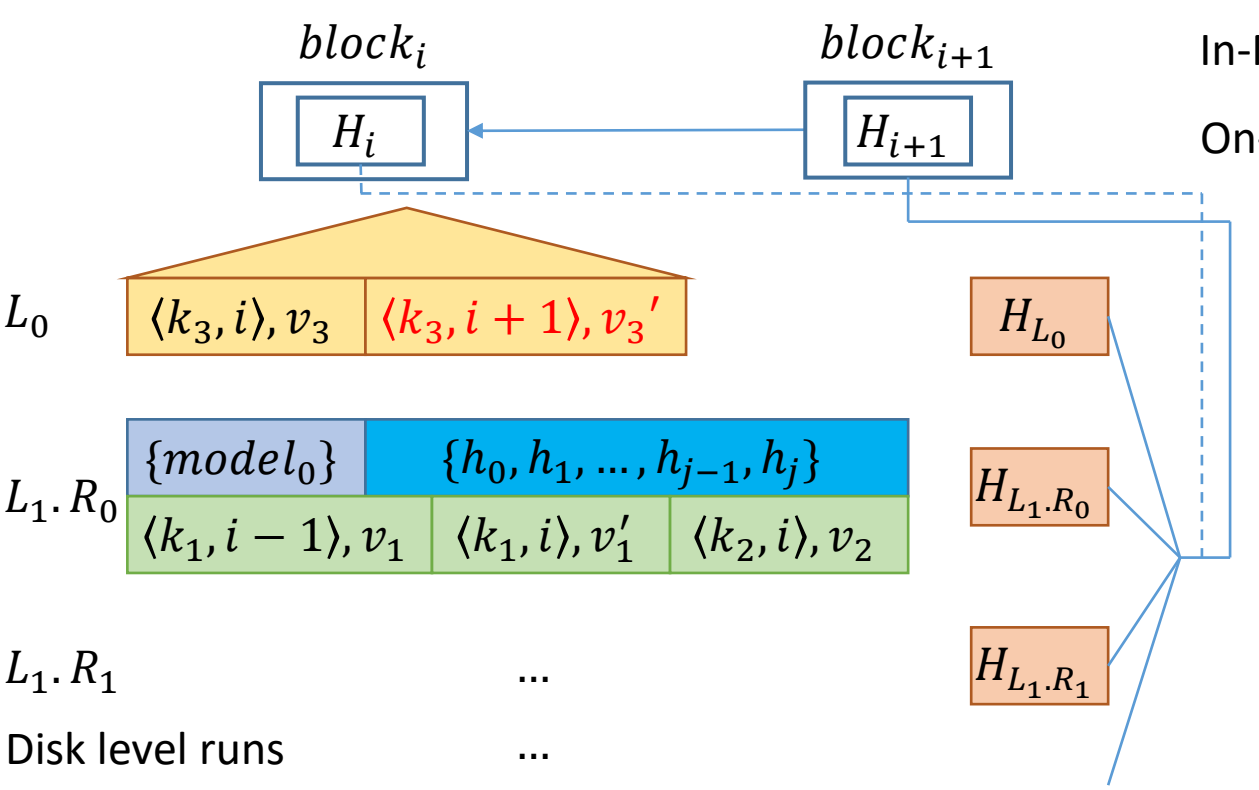


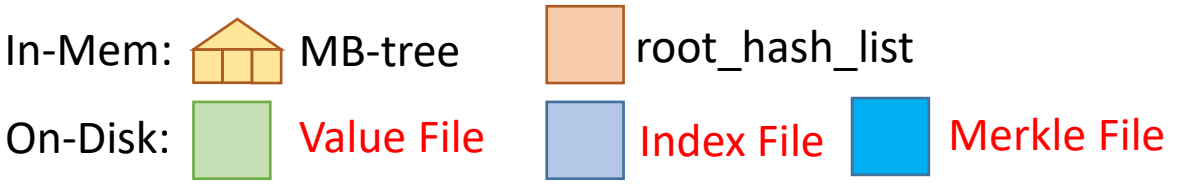


addr	blk	value
k_1	$i-1$	v_1
k_1	i	v_1'
k_2	i	v_2
k_3	i	v_3
k_3	$i+1$	v_3'

Our Design: COLE

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

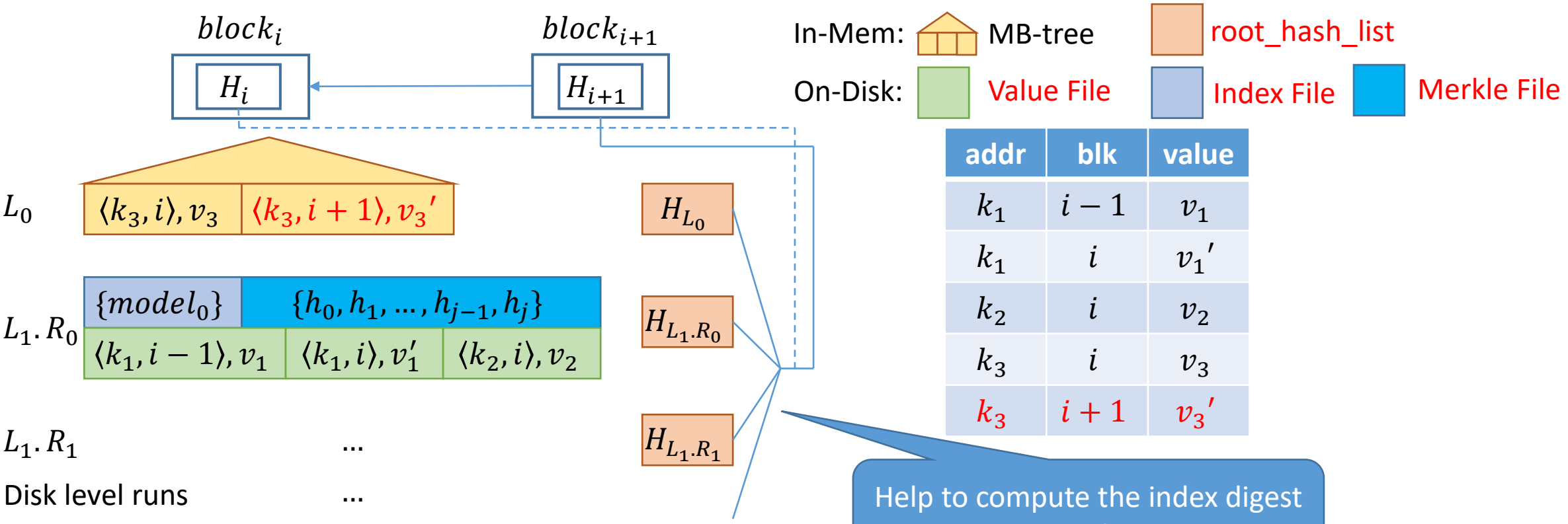




addr	blk	value
k_1	$i-1$	v_1
k_1	i	v_1'
k_2	i	v_2
k_3	i	v_3
k_3	$i+1$	v_3'

Our Design: COLE

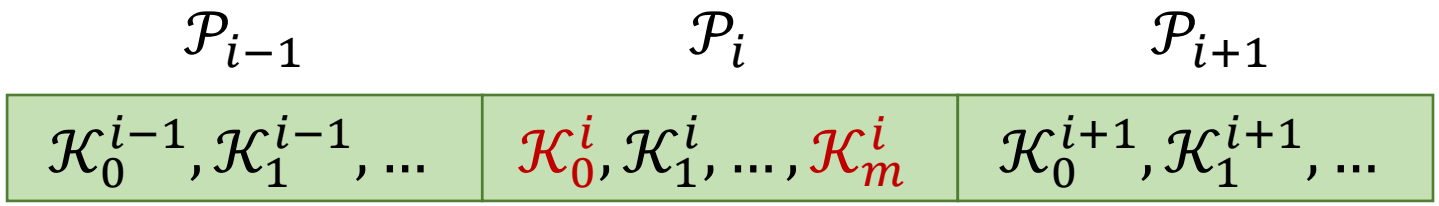
- Column-based Learned Storage for Blockchain
  - LSM-tree-based maintenance: easy to manage data writes



Learned Model in COLE

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

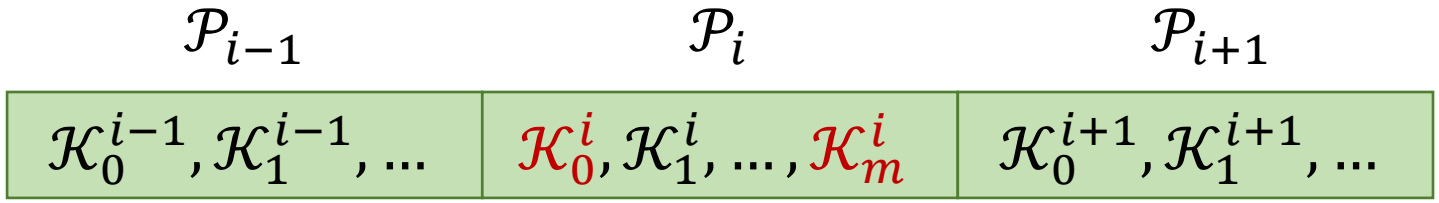
Search $\mathcal{K}_q$ using $\mathcal{M}$



Learned Model in COLE

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

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

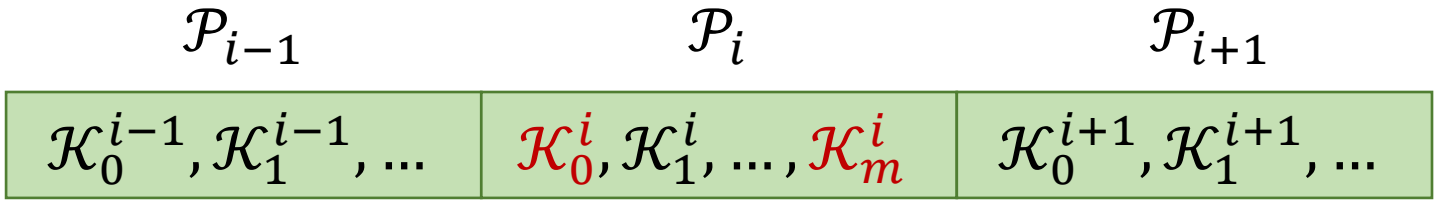


Predict $\mathcal{K}_q$ in $\mathcal{P}_i$, read $\mathcal{P}_i$

Learned Model in COLE

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

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

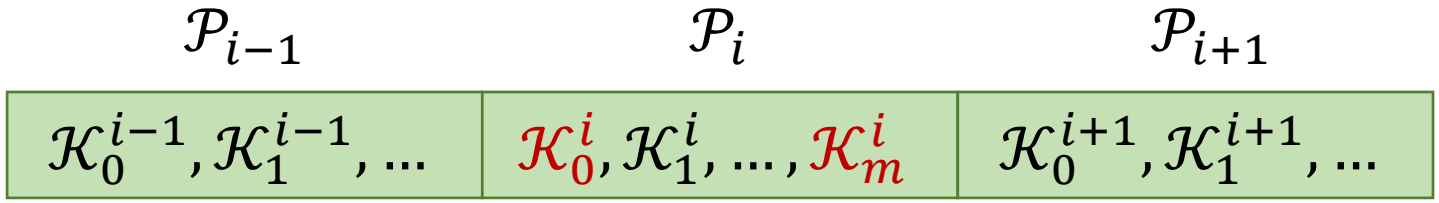


$\mathcal{K}_q < \mathcal{K}_0^i$, search $\mathcal{P}_{i-1}$

Learned Model in COLE

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

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

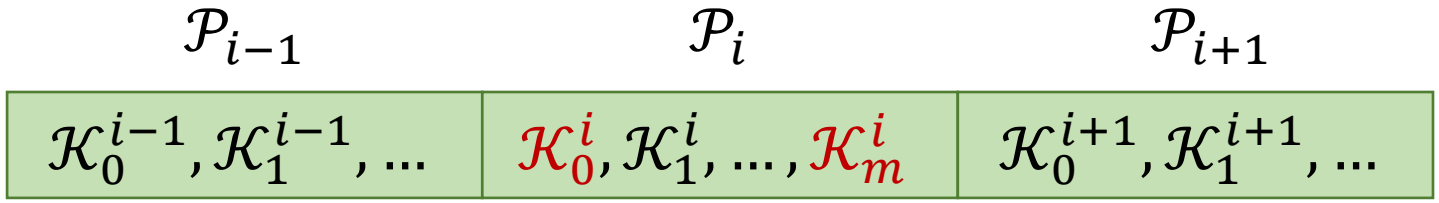


$\mathcal{K}_q > \mathcal{K}_m^i$, search $\mathcal{P}_{i+1}$

Learned Model in COLE

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

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

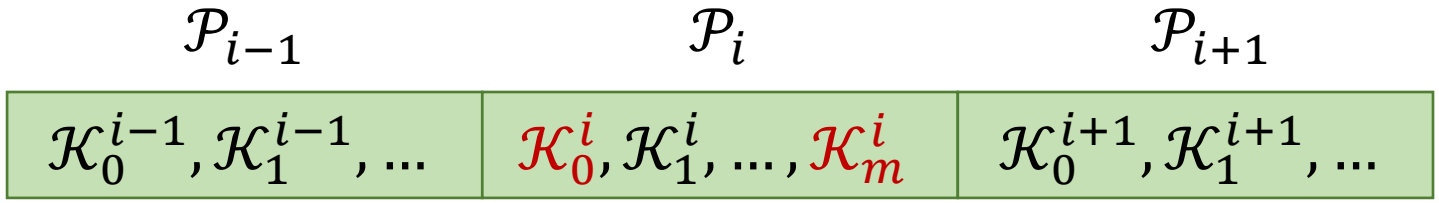


Else, search $\mathcal{P}_i$

Learned Model in COLE

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

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



Else, search $\mathcal{P}_i$

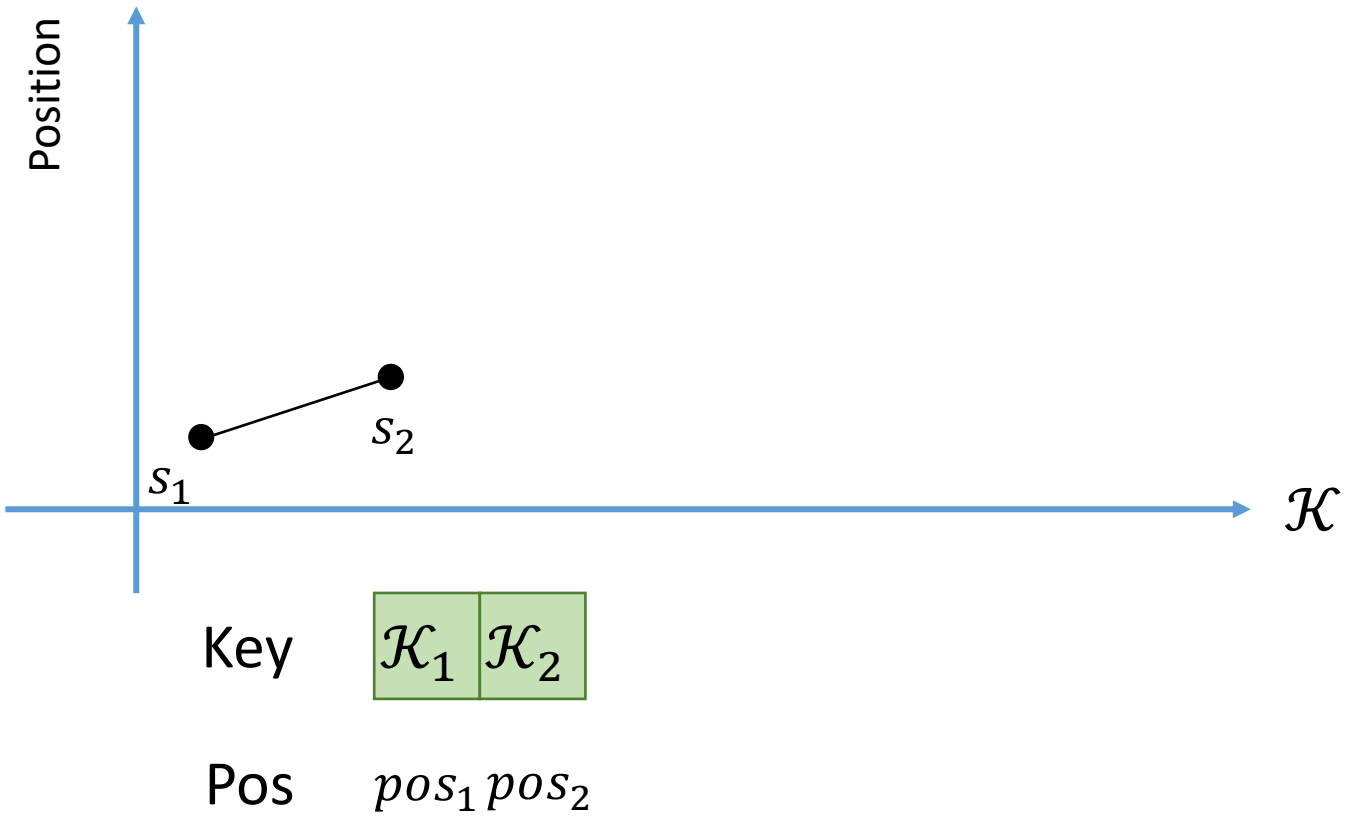
2 Page IO

Learned Model in COLE

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

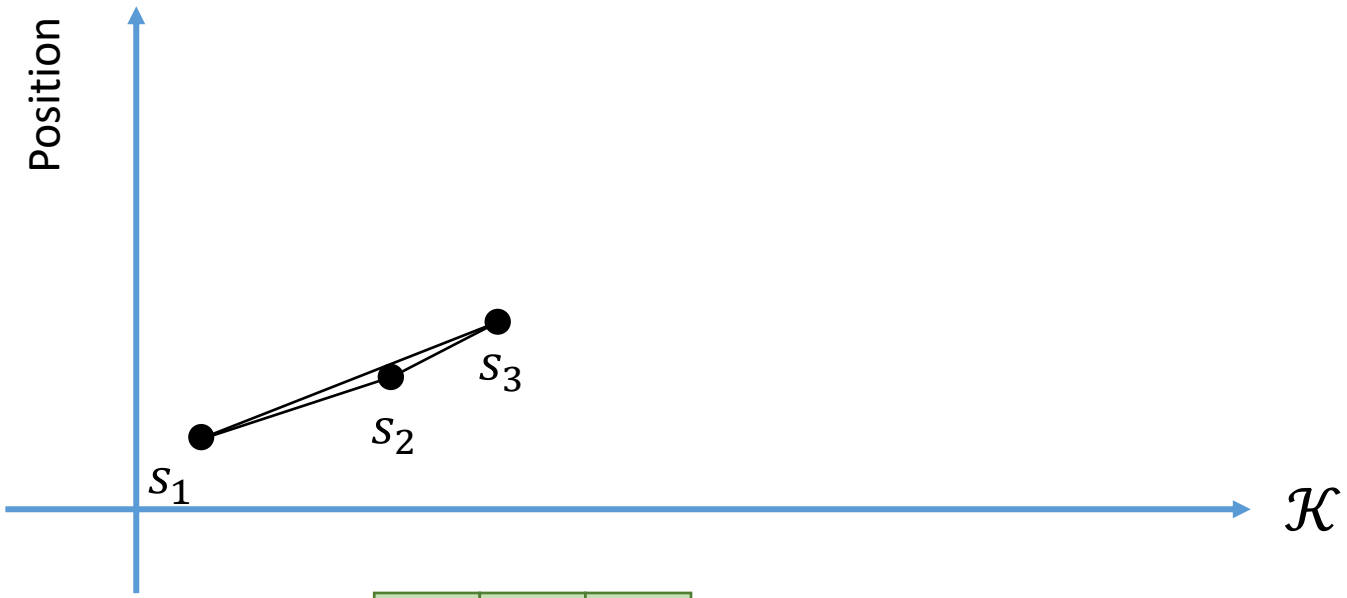
Learned Model in COLE

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



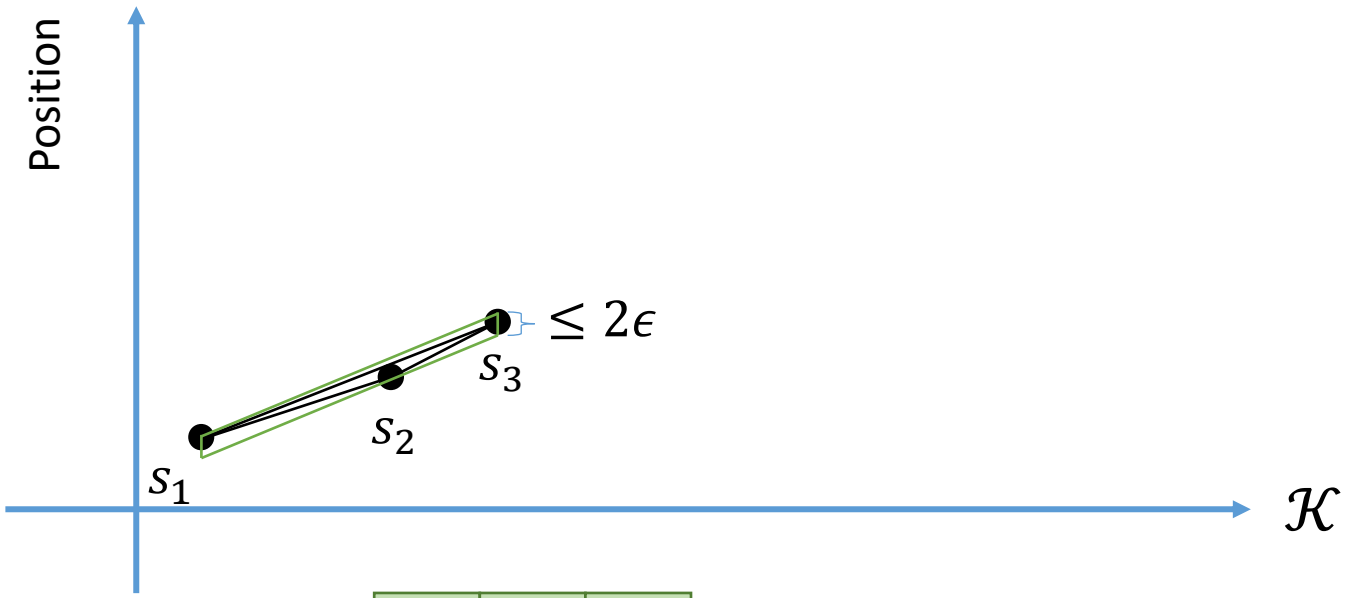
Learned Model in COLE

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



Learned Model in COLE

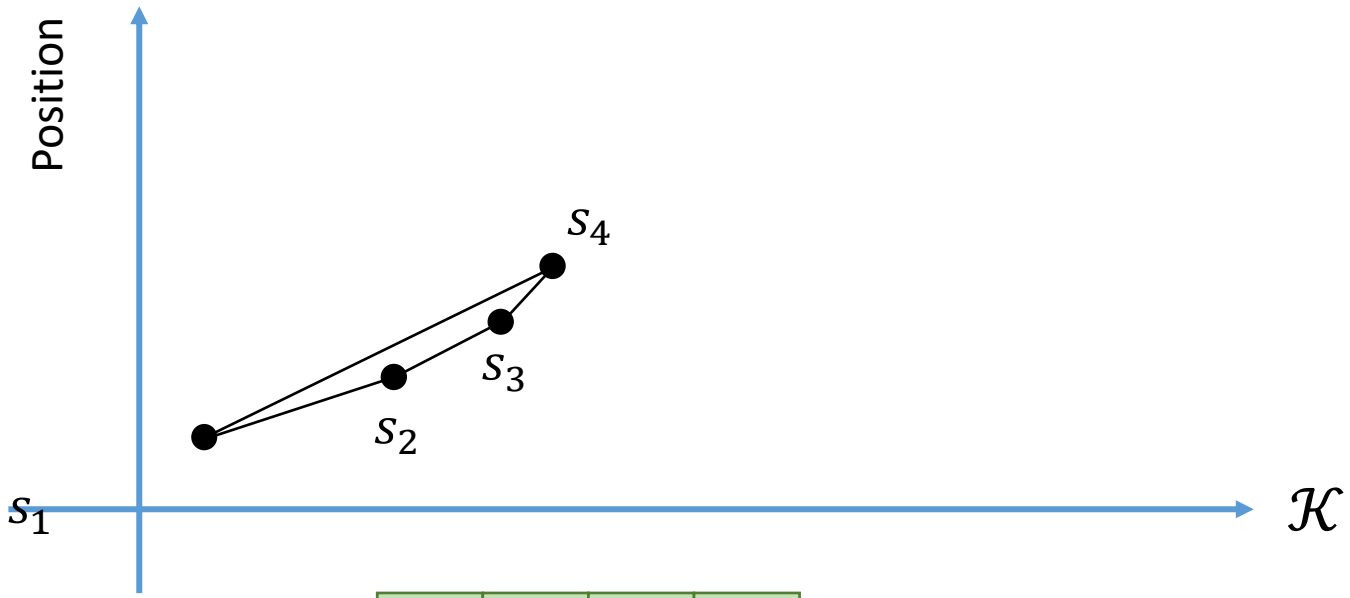
- ϵ-Bounded Piecewise Linear Model
  - Fast to learn: O(1) cost to add each point



Key	$\mathcal{K}_1$	$\mathcal{K}_2$	$\mathcal{K}_3$
Pos	pos_1	pos_2	pos_3

Learned Model in COLE

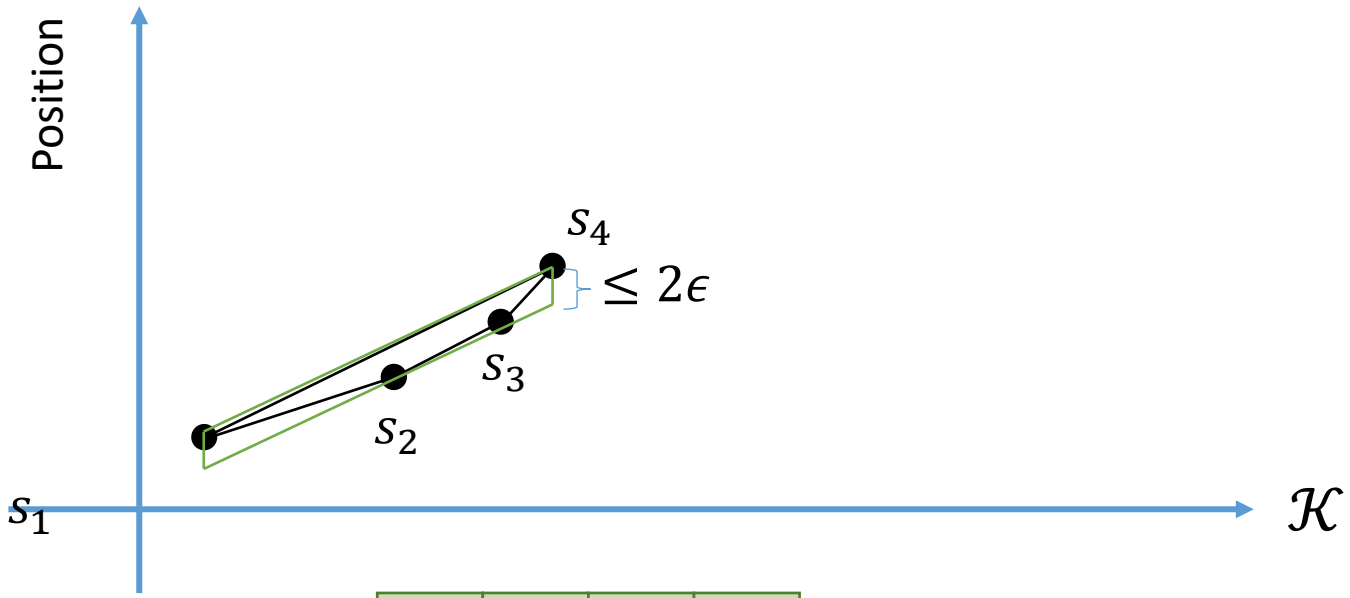
- ϵ-Bounded Piecewise Linear Model
  - Fast to learn: O(1) cost to add each point



Key	$\mathcal{K}_1$	$\mathcal{K}_2$	$\mathcal{K}_3$	$\mathcal{K}_4$
Pos	pos_1	pos_2	pos_3	pos_4

Learned Model in COLE

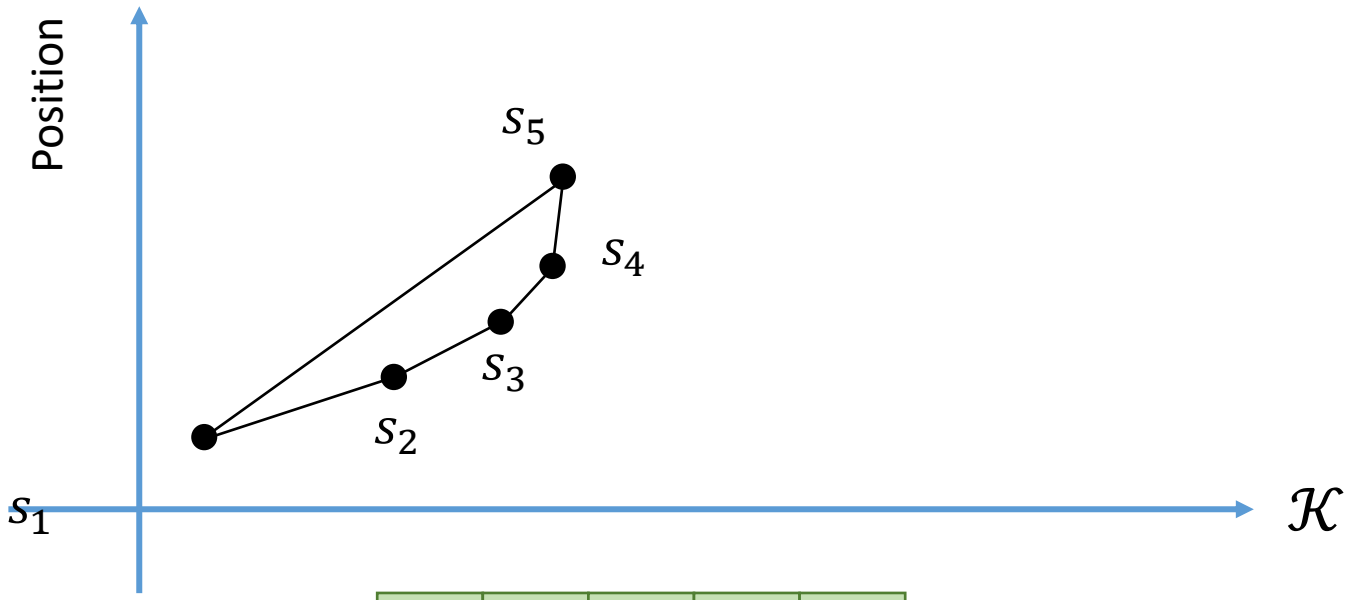
- ϵ-Bounded Piecewise Linear Model
  - Fast to learn: O(1) cost to add each point



Key	$\mathcal{K}_1$	$\mathcal{K}_2$	$\mathcal{K}_3$	$\mathcal{K}_4$
Pos	pos_1	pos_2	pos_3	pos_4

Learned Model in COLE

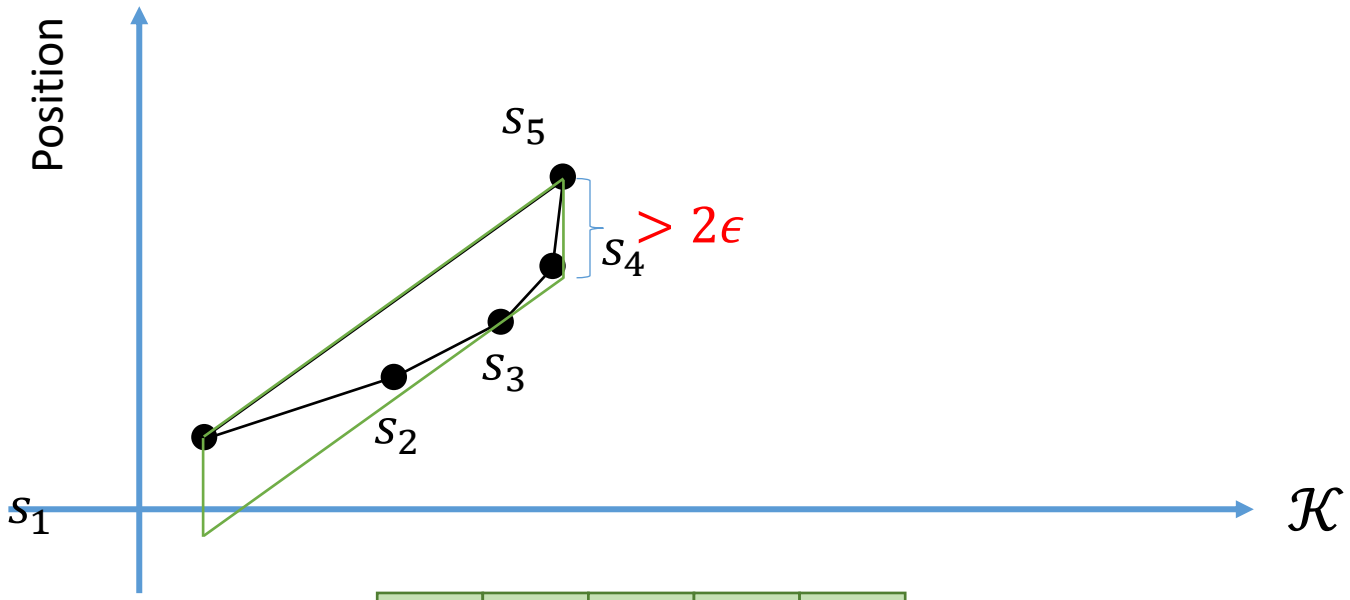
- ϵ-Bounded Piecewise Linear Model
  - Fast to learn: O(1) cost to add each point



Key	$\mathcal{K}_1$	$\mathcal{K}_2$	$\mathcal{K}_3$	$\mathcal{K}_4$	$\mathcal{K}_5$
Pos	pos_1	pos_2	pos_3	pos_4	pos_5

Learned Model in COLE

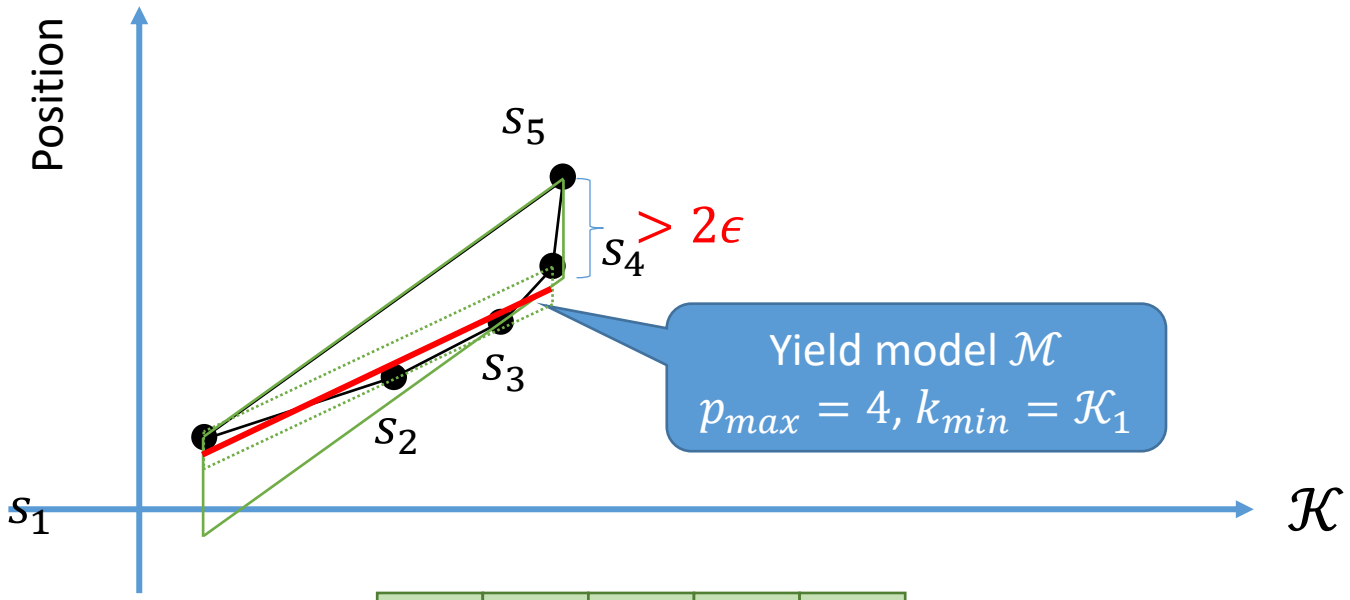
- ϵ-Bounded Piecewise Linear Model
  - Fast to learn: O(1) cost to add each point



Key	$\mathcal{K}_1$	$\mathcal{K}_2$	$\mathcal{K}_3$	$\mathcal{K}_4$	$\mathcal{K}_5$
Pos	pos_1	pos_2	pos_3	pos_4	pos_5

Learned Model in COLE

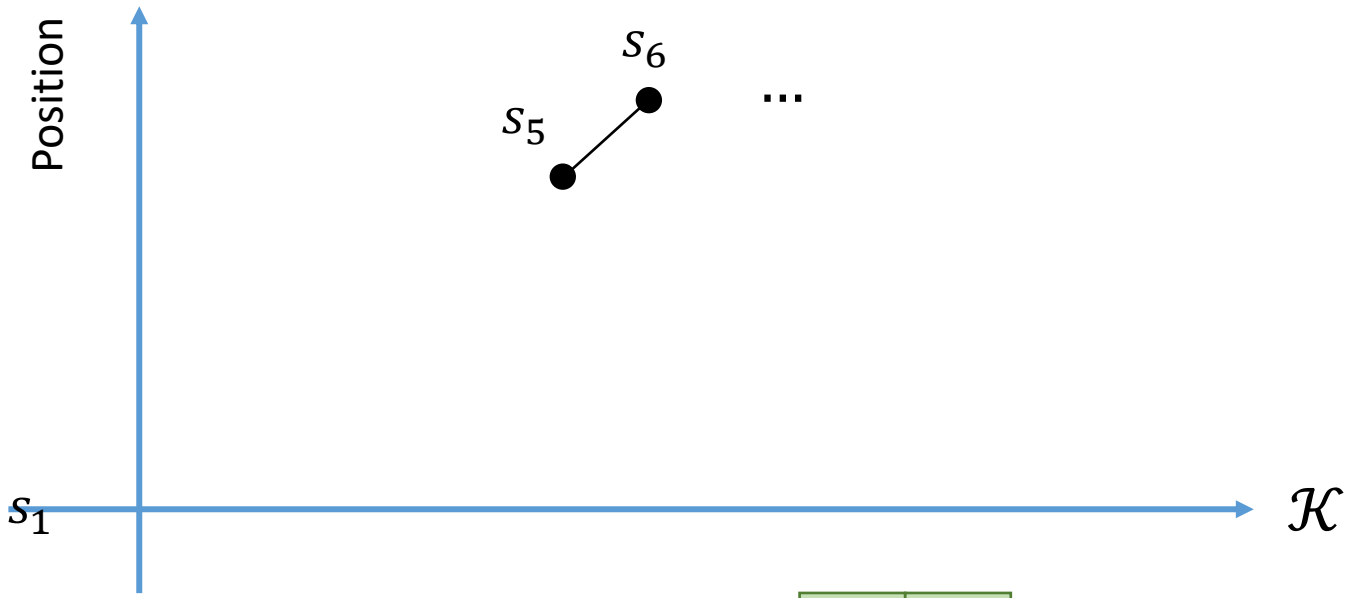
- ϵ-Bounded Piecewise Linear Model
  - Fast to learn: O(1) cost to add each point



Key	$\mathcal{K}_1$	$\mathcal{K}_2$	$\mathcal{K}_3$	$\mathcal{K}_4$	$\mathcal{K}_5$
Pos	pos_1	pos_2	pos_3	pos_4	pos_5

Learned Model in COLE

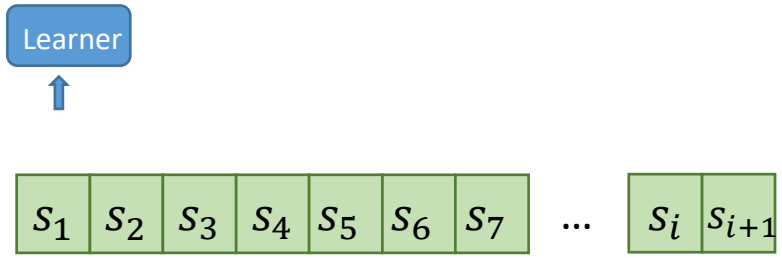
- ϵ-Bounded Piecewise Linear Model
  - Fast to learn: O(1) cost to add each point



Learned Model in COLE

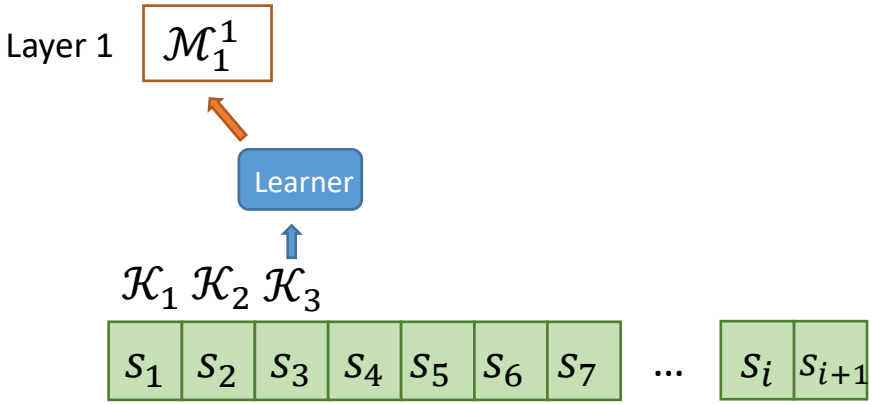
- Index File
  - Build multiple layers of models and write them to a file

Layer 1



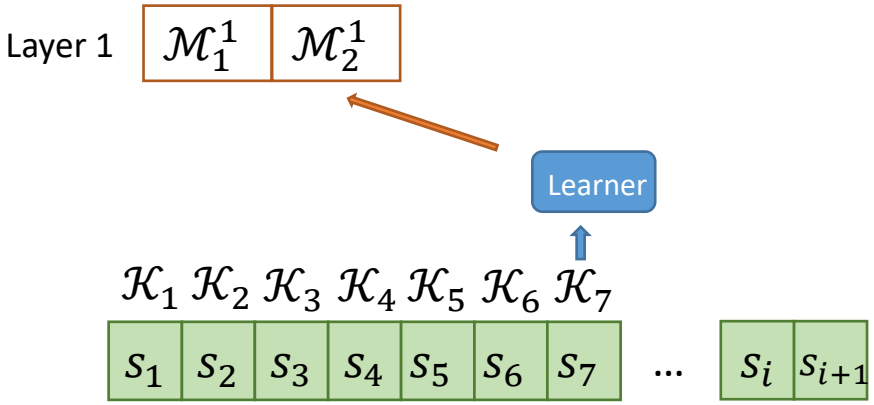
Learned Model in COLE

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



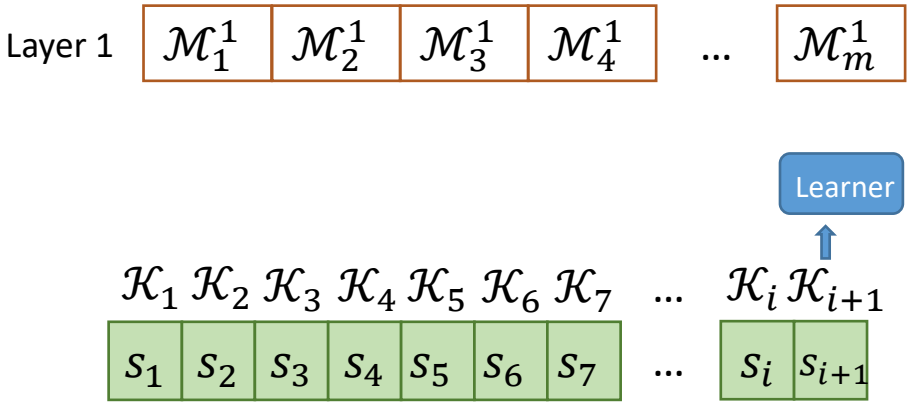
Learned Model in COLE

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



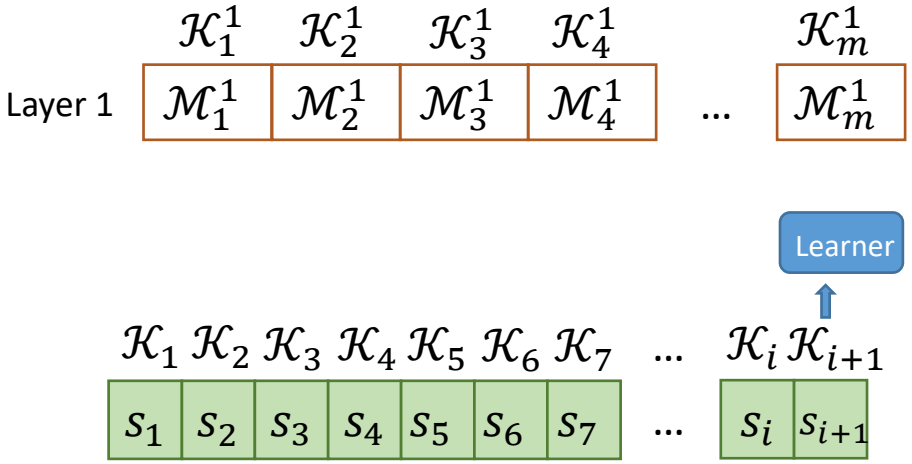
Learned Model in COLE

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



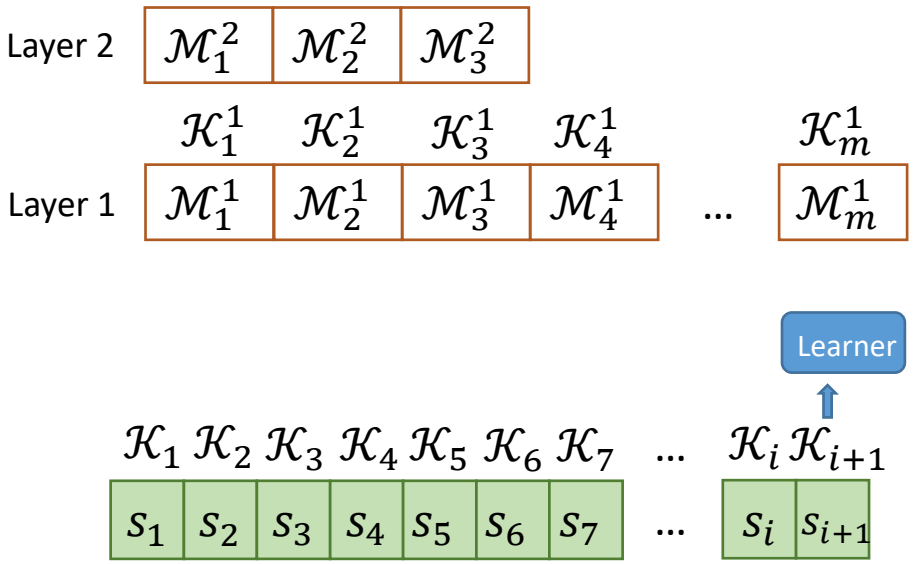
Learned Model in COLE

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



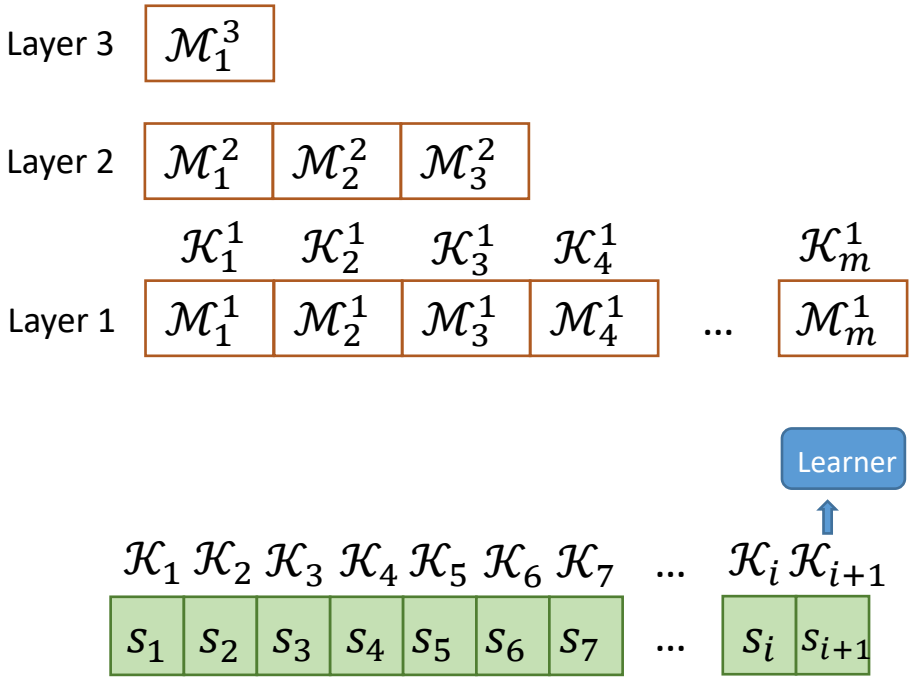
Learned Model in COLE

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



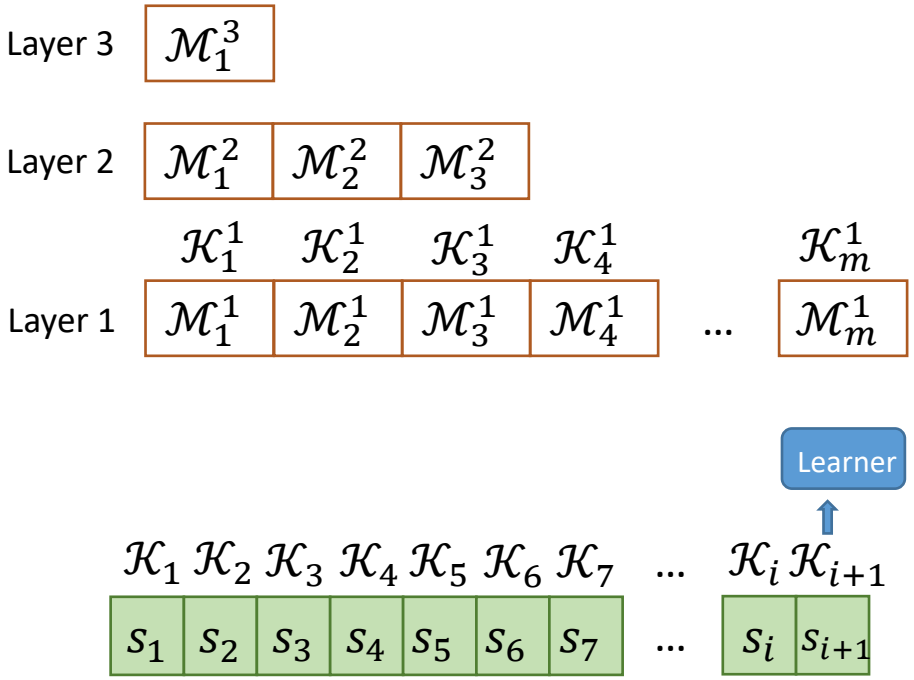
Learned Model in COLE

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

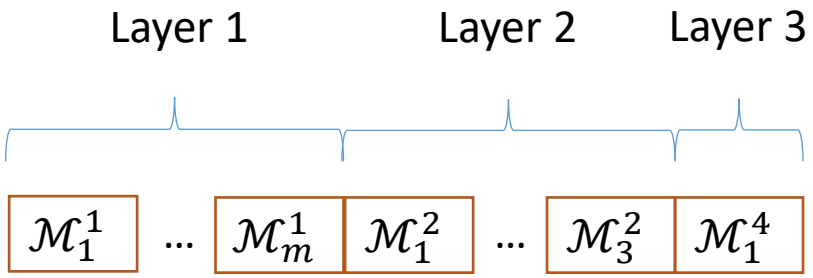


Learned Model in COLE

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

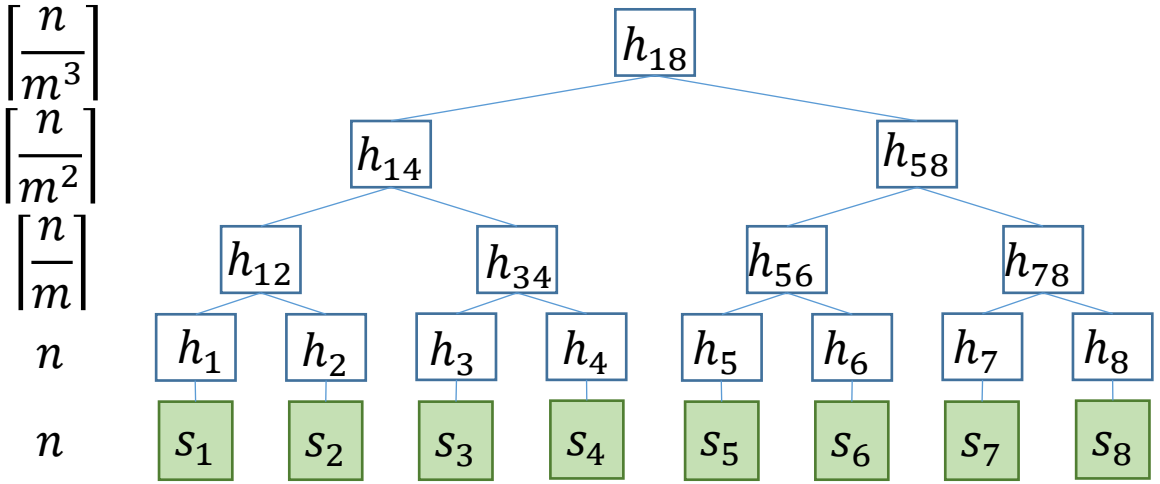


Index File Layout

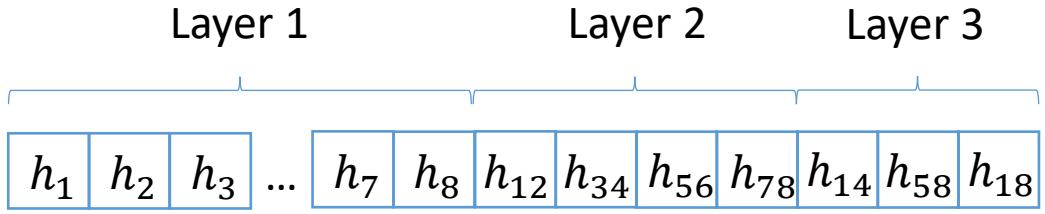


Merkle File

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

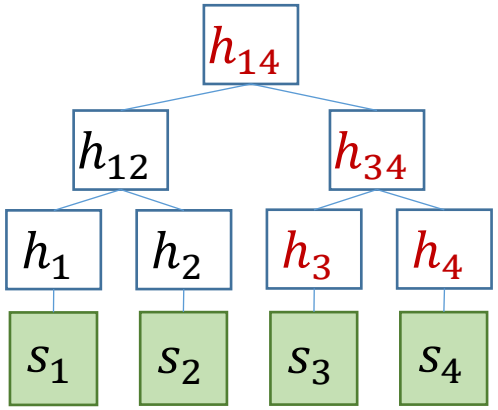


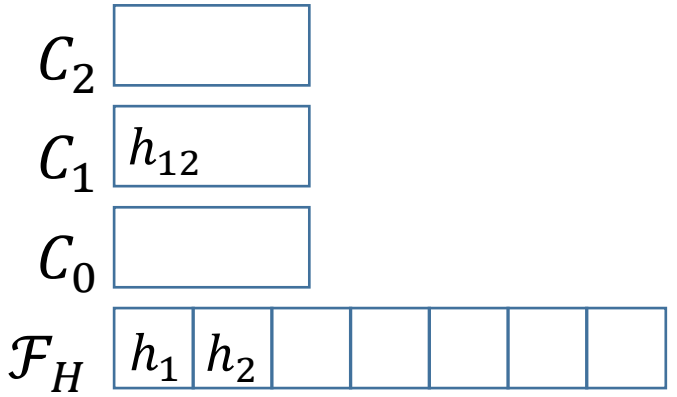
Merkle File Layout



Merkle File

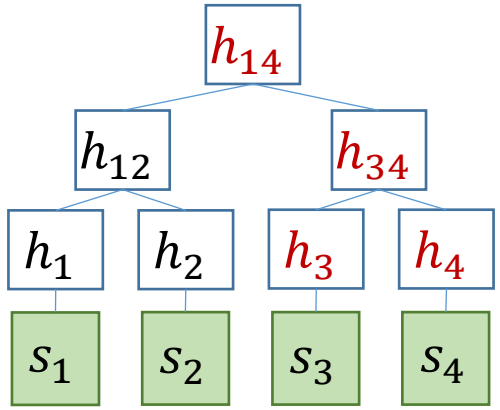
- Construct m-ary complete MHT streamingly
  - Maintain MHT's height number of buffers

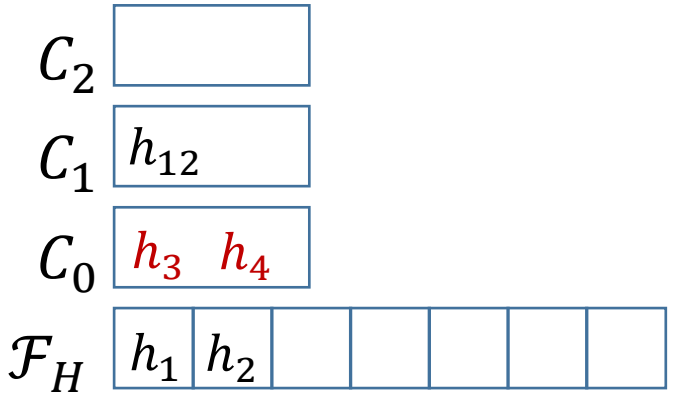




Merkle File

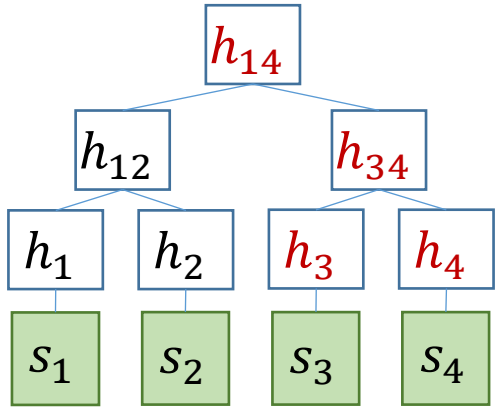
- Construct m-ary complete MHT streamingly
  - Maintain MHT's height number of buffers

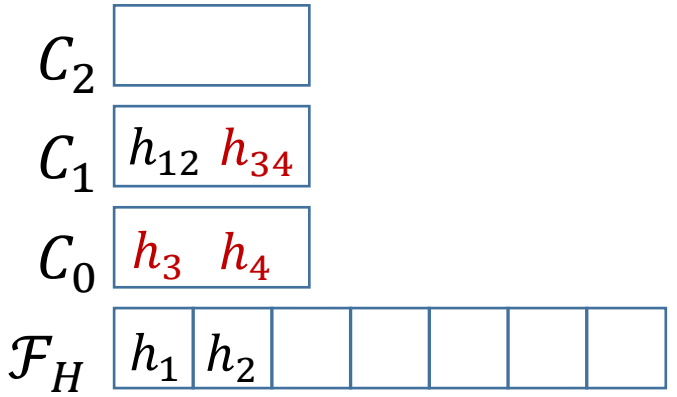




Merkle File

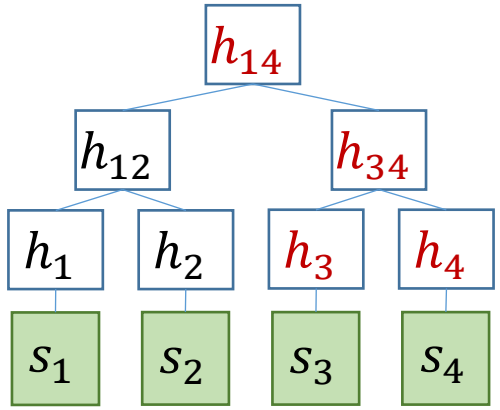
- Construct m-ary complete MHT streamingly
  - Maintain MHT's height number of buffers

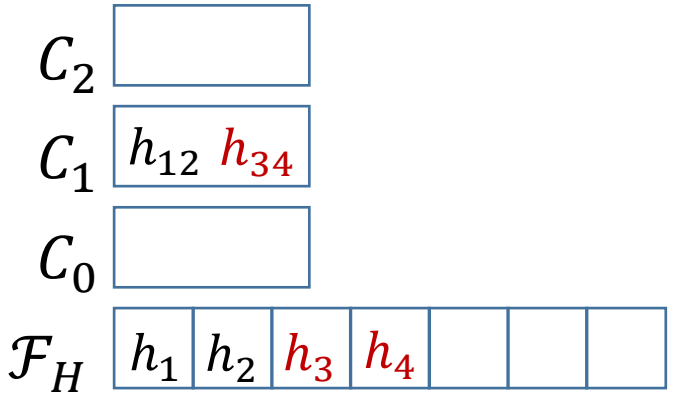




Merkle File

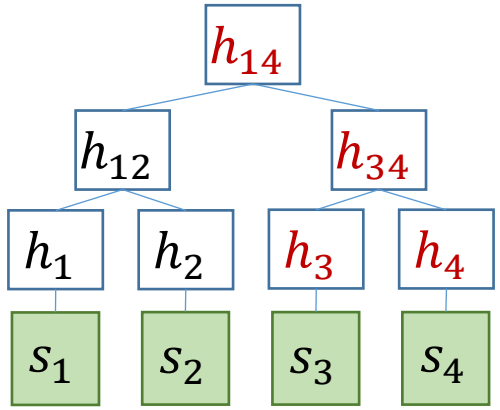
- Construct m-ary complete MHT streamingly
  - Maintain MHT's height number of buffers

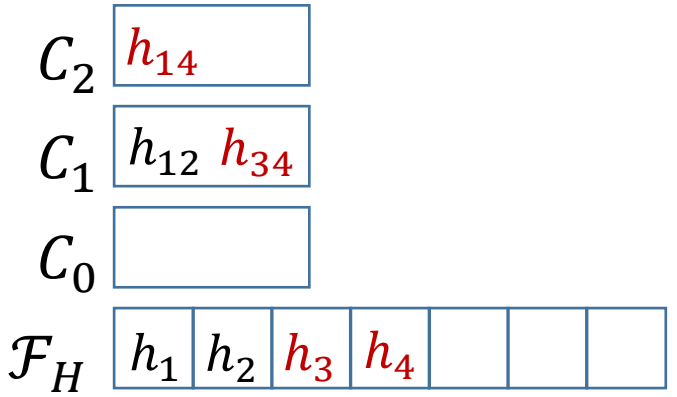




Merkle File

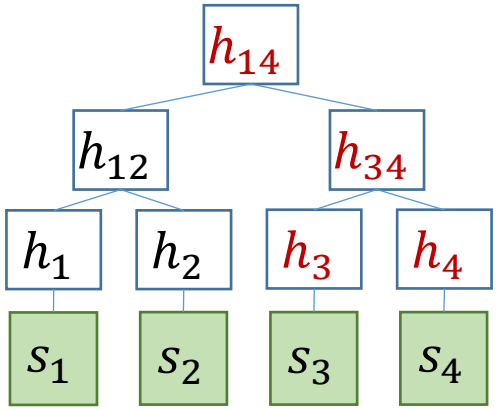
- Construct m-ary complete MHT streamingly
  - Maintain MHT's height number of buffers

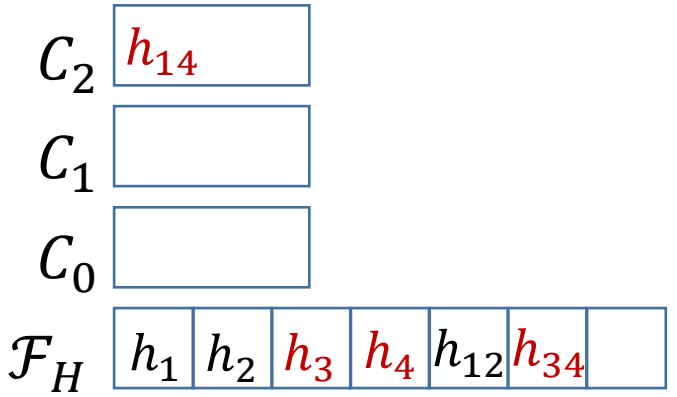




Merkle File

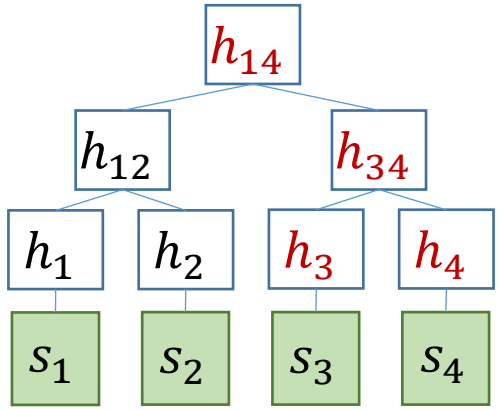
- Construct m-ary complete MHT streamingly
  - Maintain MHT's height number of buffers

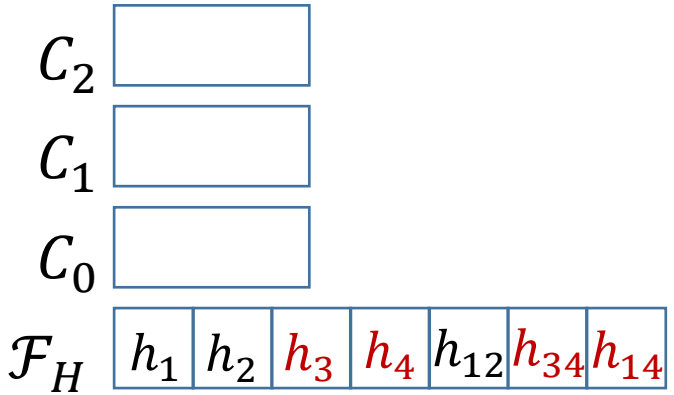




Merkle File

- Construct m-ary complete MHT streamingly
  - Maintain MHT's height number of buffers

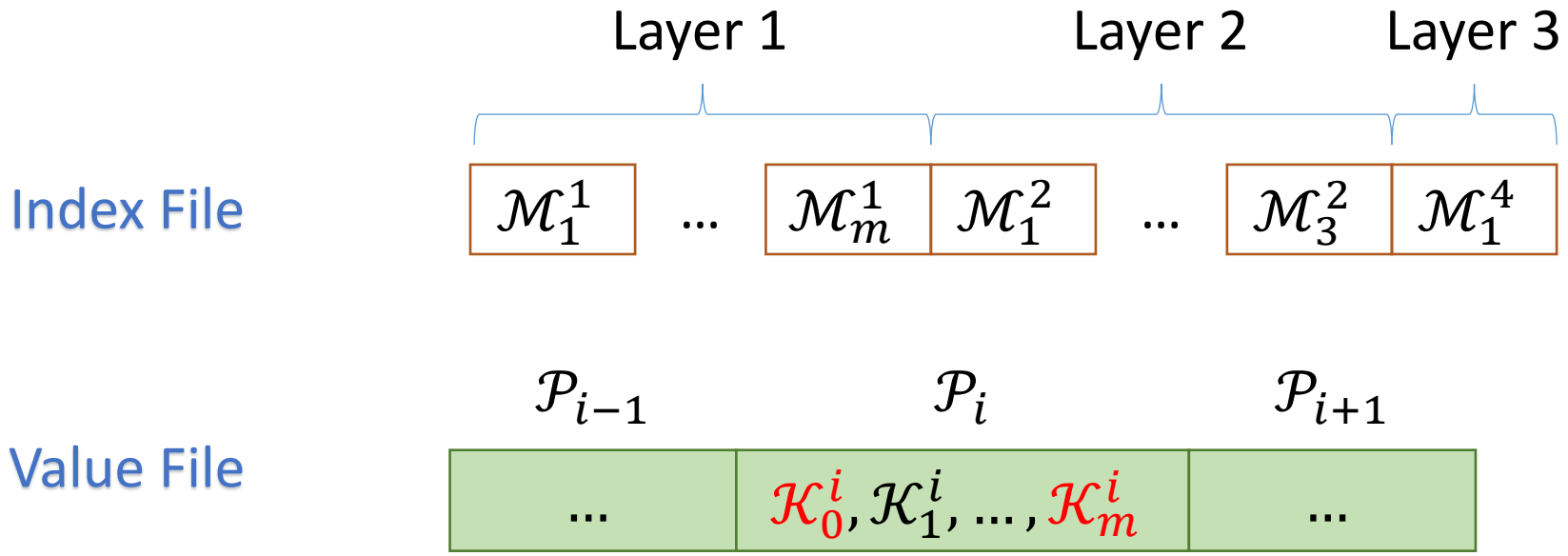




Read Operations

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

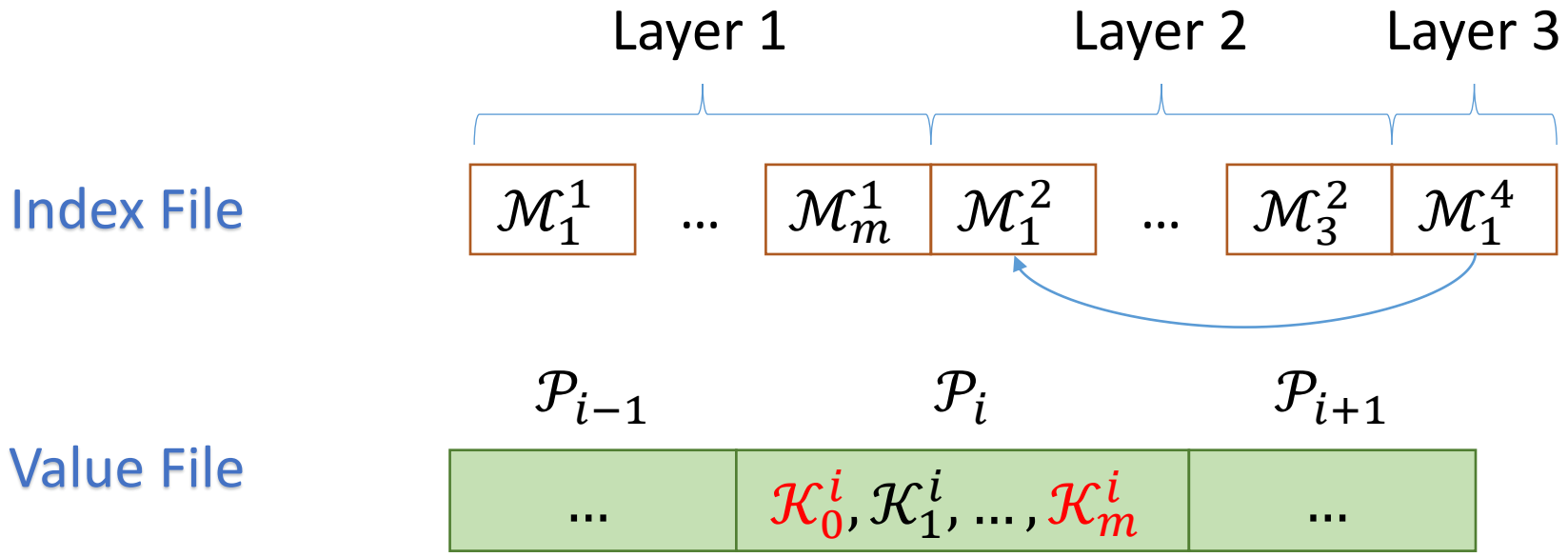
  For each run in LSM-tree:



Read Operations

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

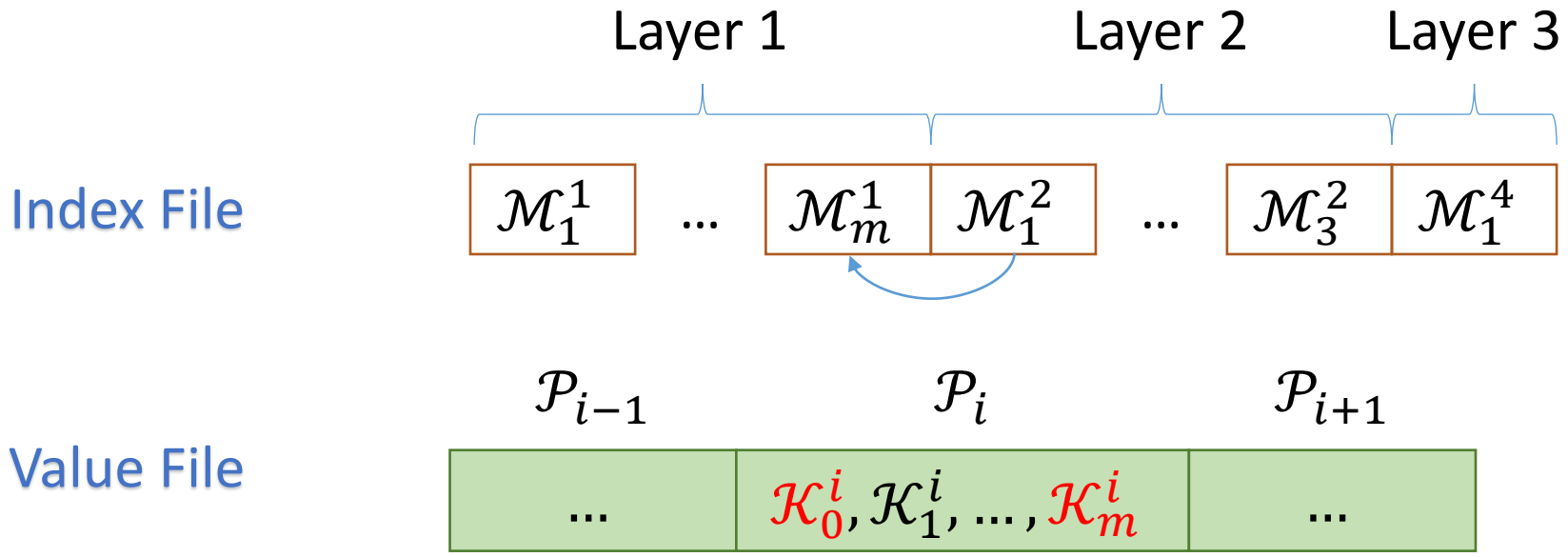
  For each run in LSM-tree:



Read Operations

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

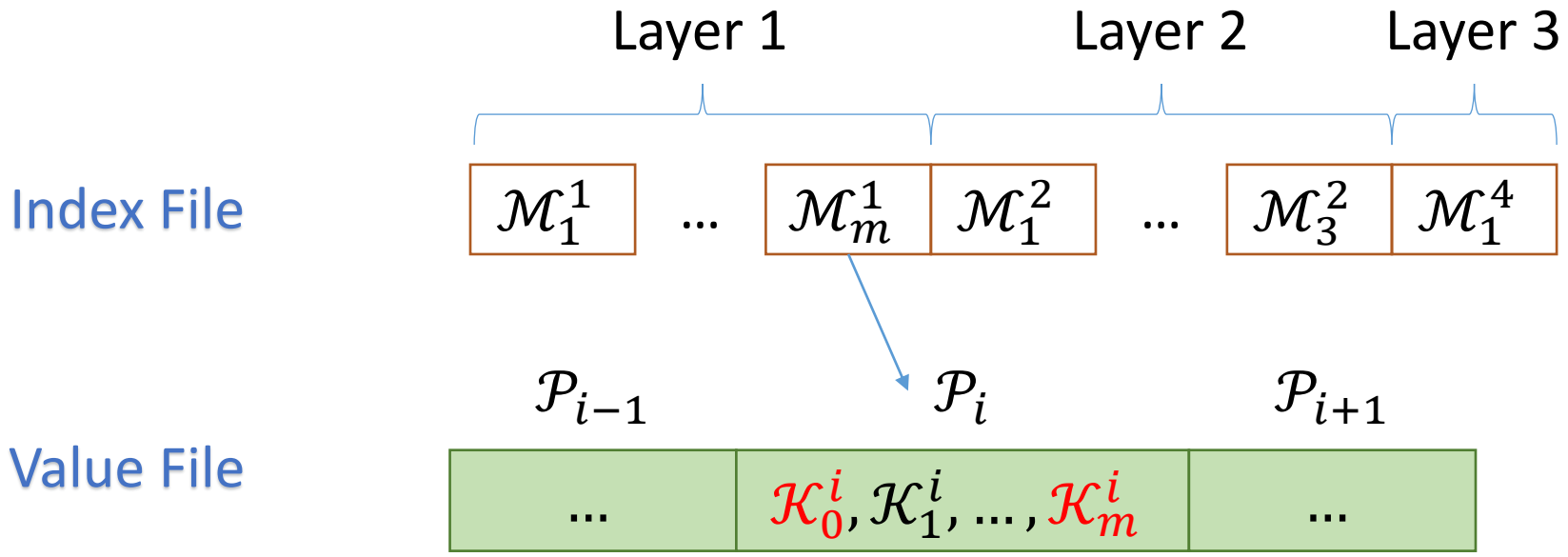
  For each run in LSM-tree:



Read Operations

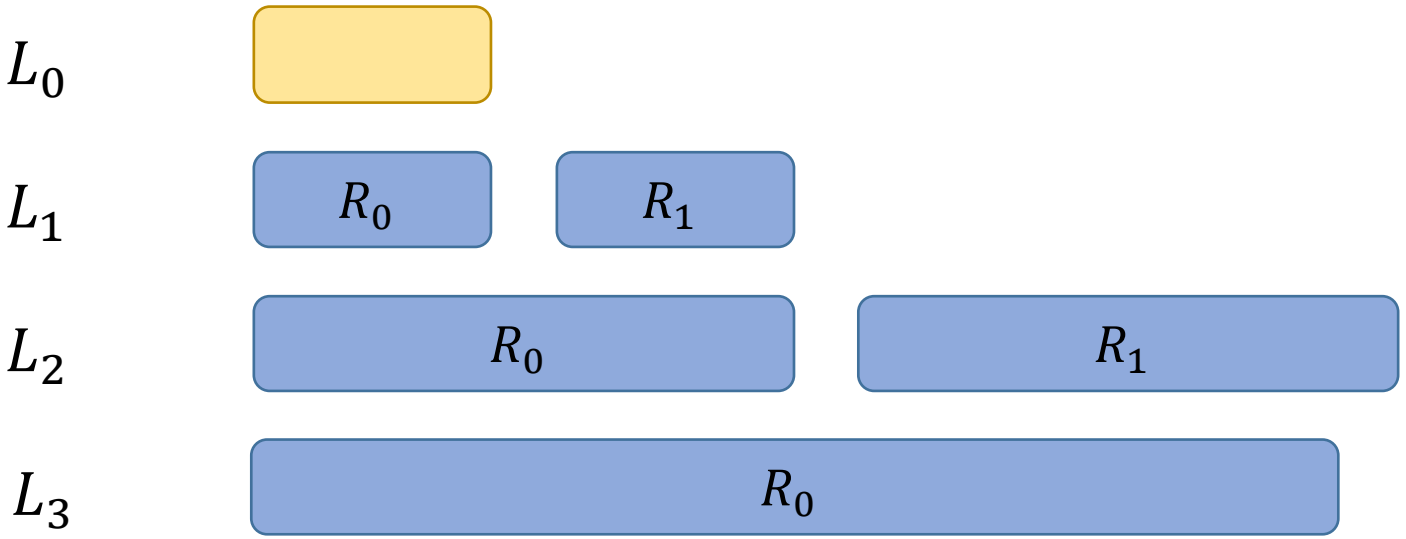
- Get query
  - Search for the latest value of $addr_q$: $\mathcal{K}_q = \langle addr_q, \max\_int \rangle$
  - Retrieve the state with the largest $\mathcal{K}_r$ and $\mathcal{K}_r < \mathcal{K}_q$

  For each run in LSM-tree:



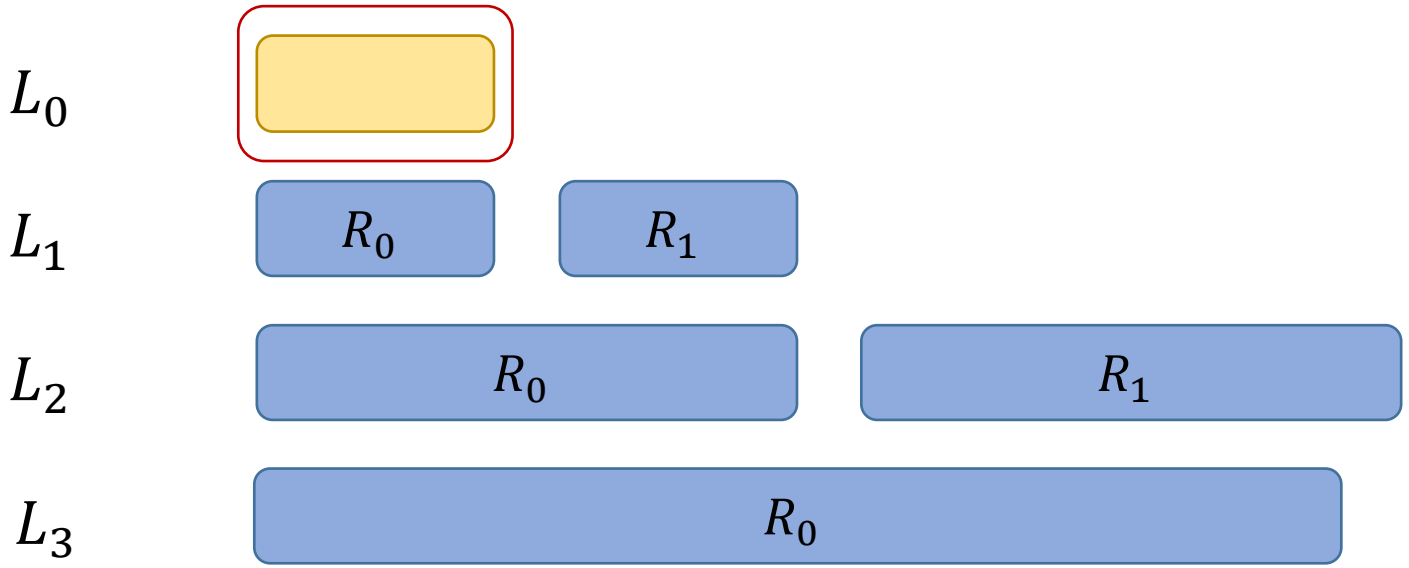
Read Operations

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



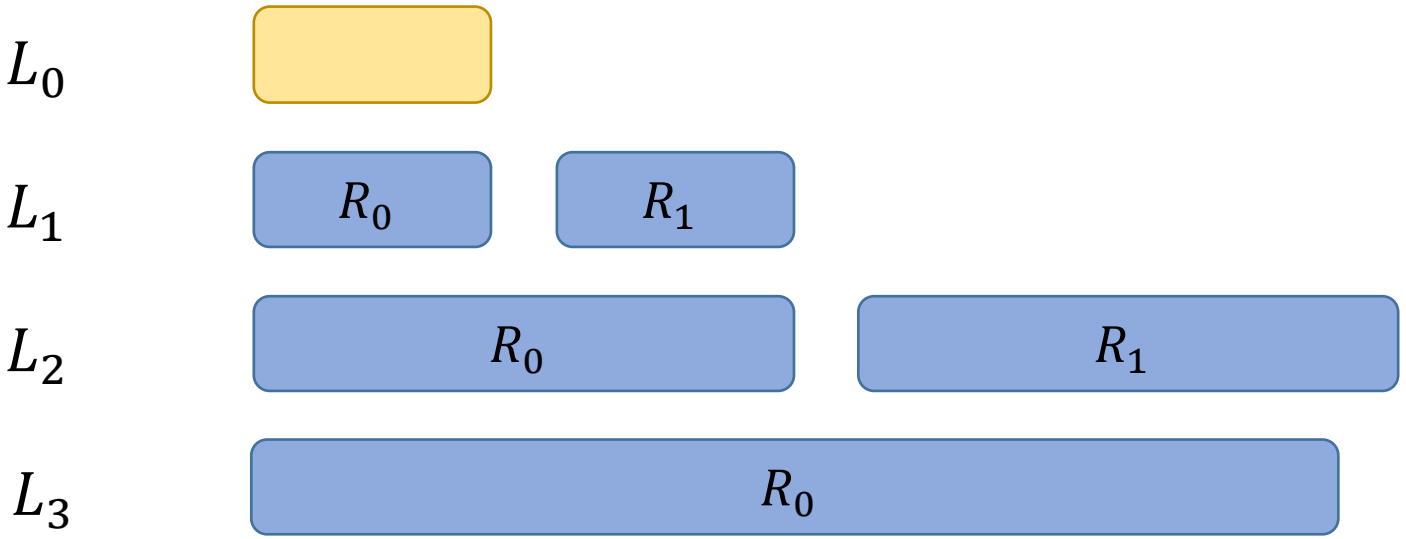
Read Operations

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



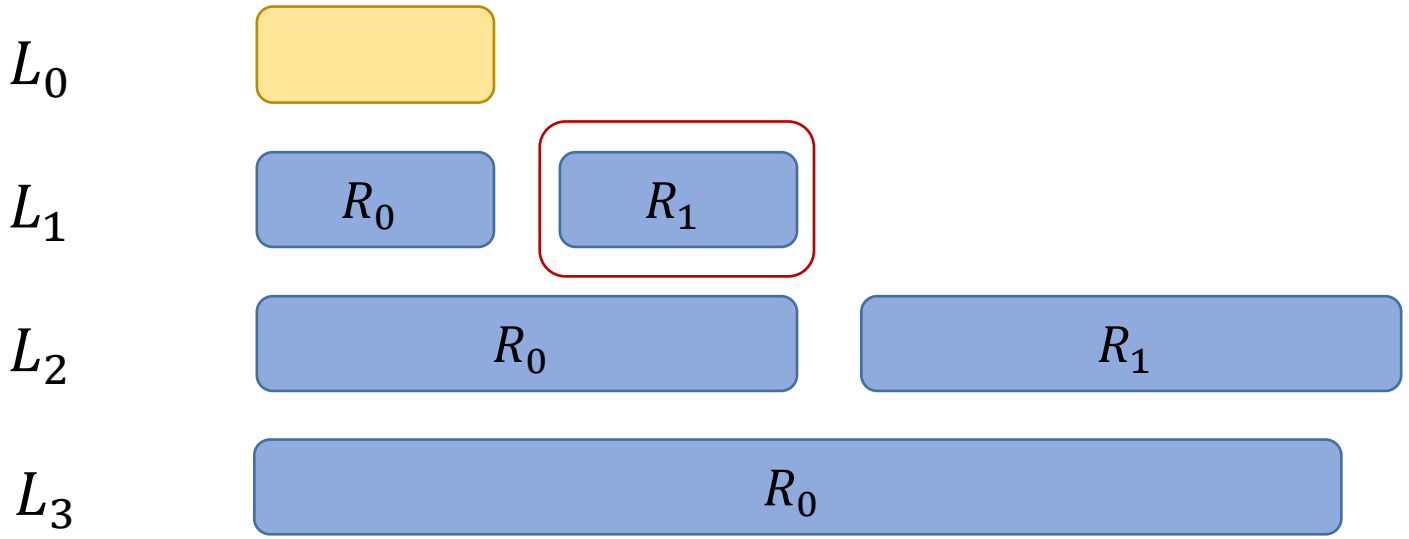
Read Operations

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



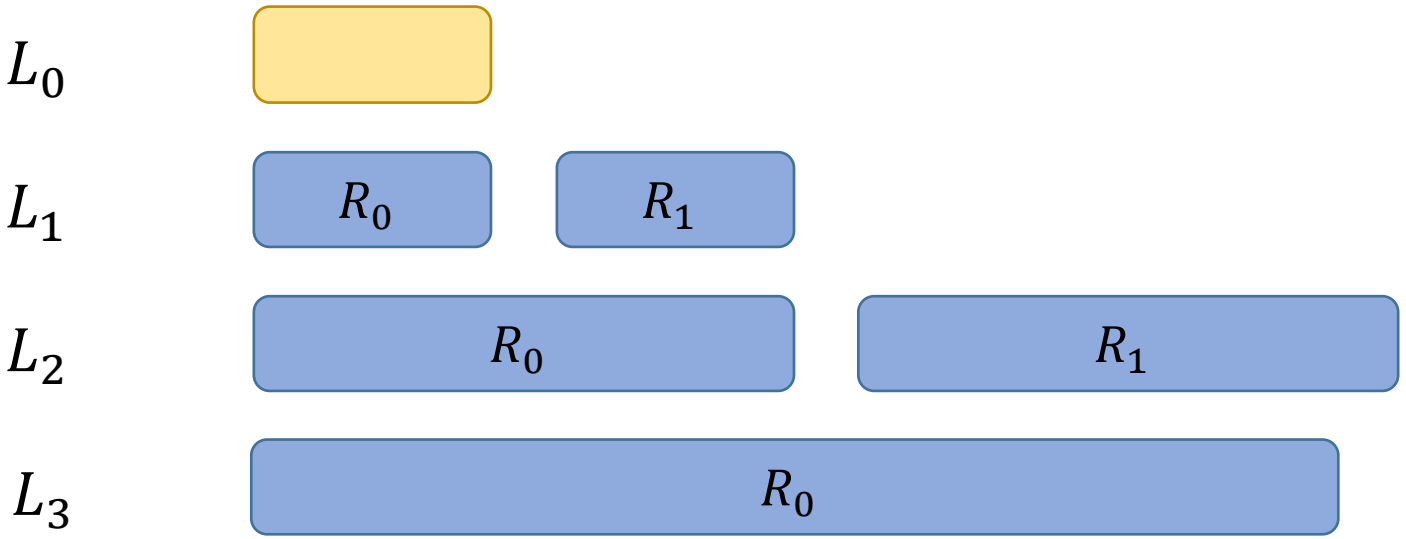
Read Operations

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



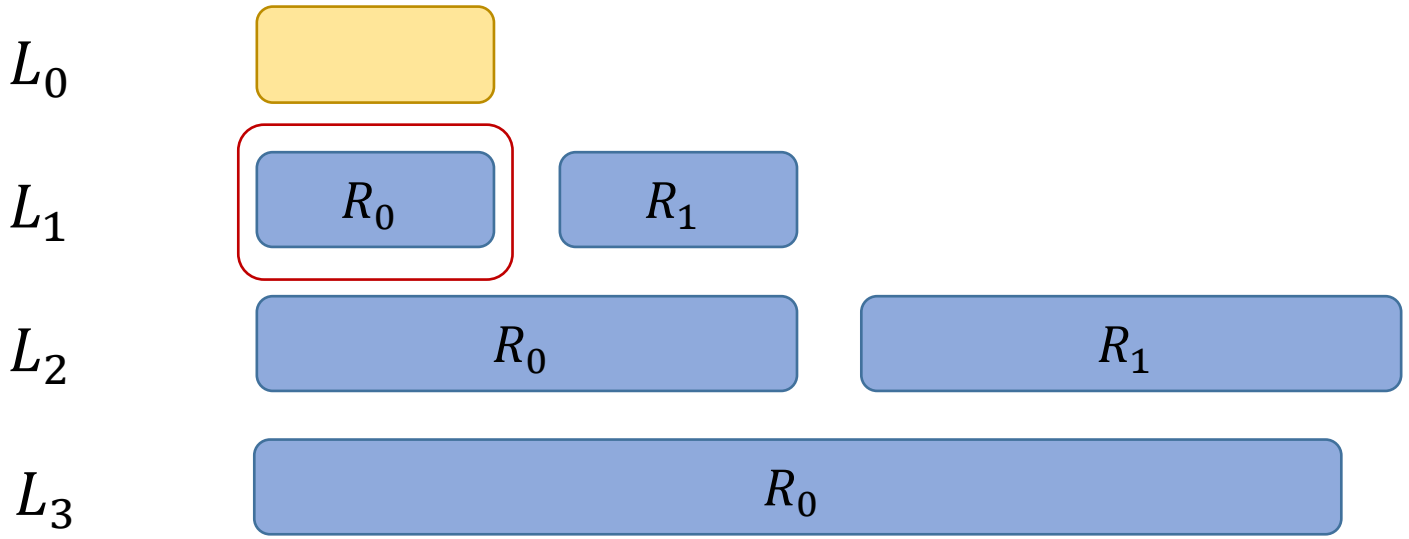
Read Operations

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



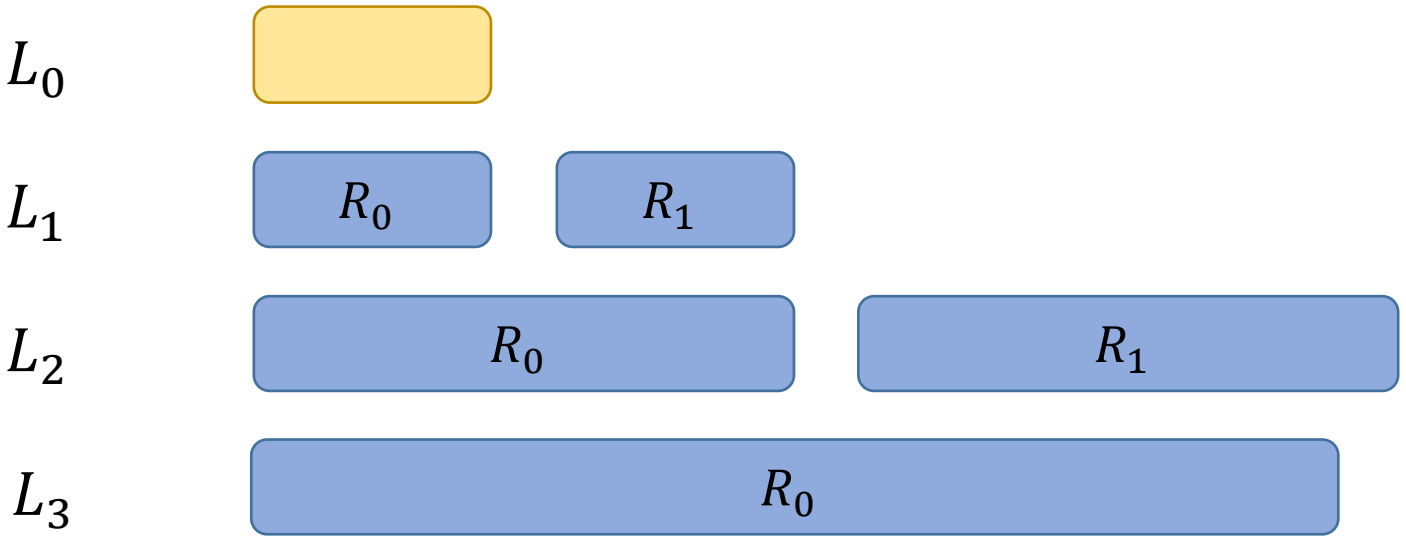
Read Operations

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



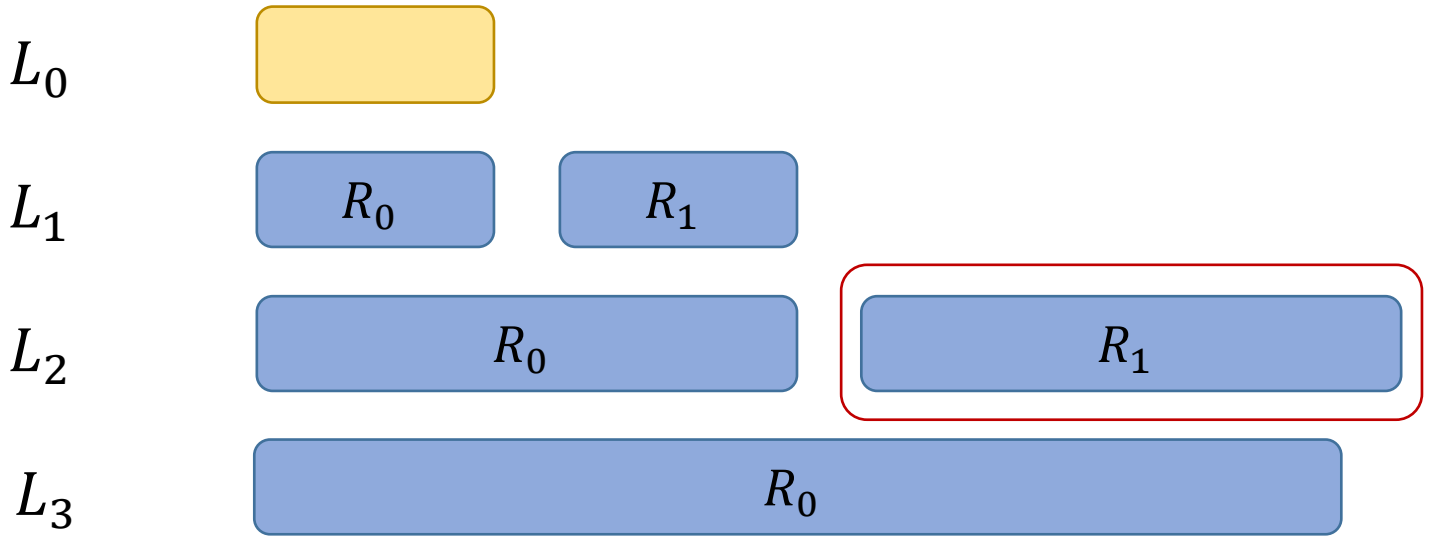
Read Operations

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



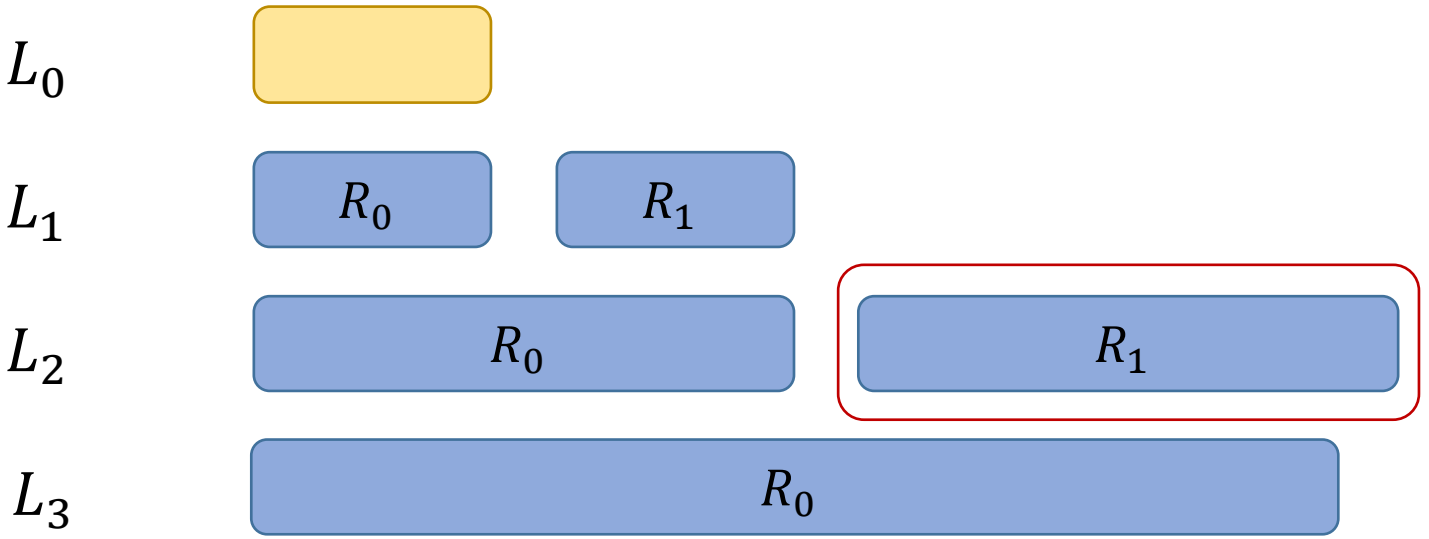
Read Operations

- 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

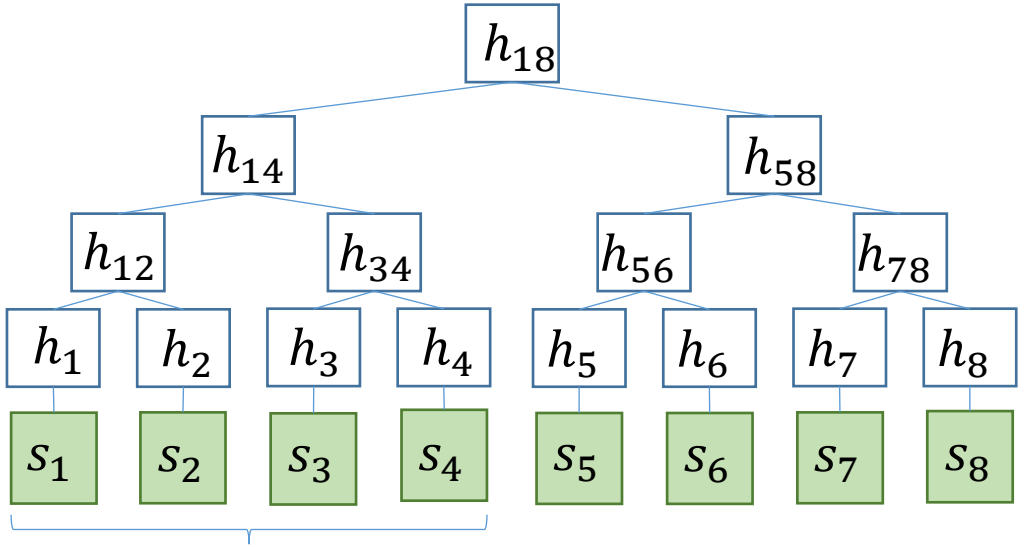
- Get query
  - For each LSM-tree level, search in a top-down fashion
  - For each level, search each run by freshness (new -> old)



Find the result, early stop

Read Operations

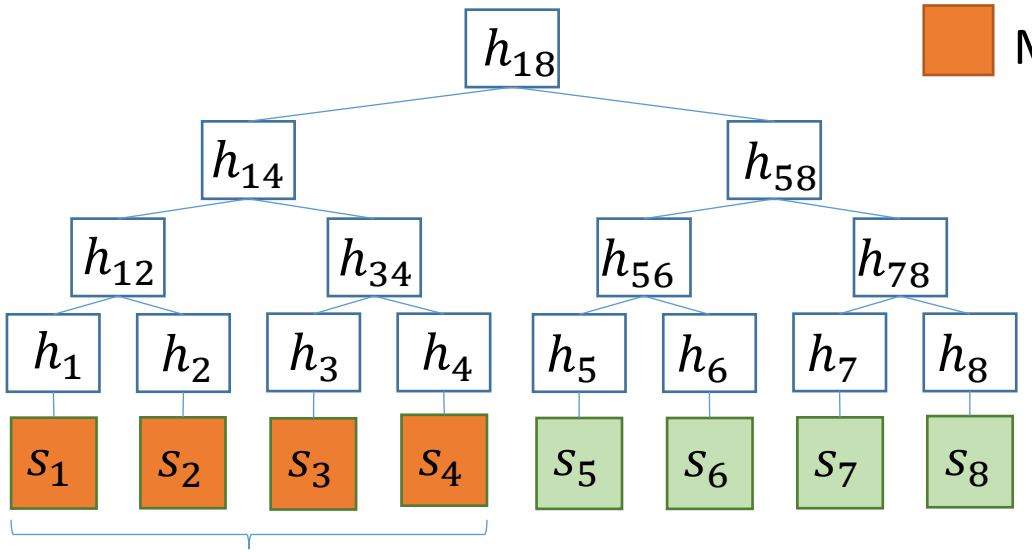
- Provenance query
  - (i) range query, (ii) Merkle proof generation
  - $\mathcal{K}_l = \langle addr_q, blk_l \rangle; \mathcal{K}_u = \langle addr_q, blk_u \rangle$
  - Search level by level from top to bottom and combine results and proofs



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

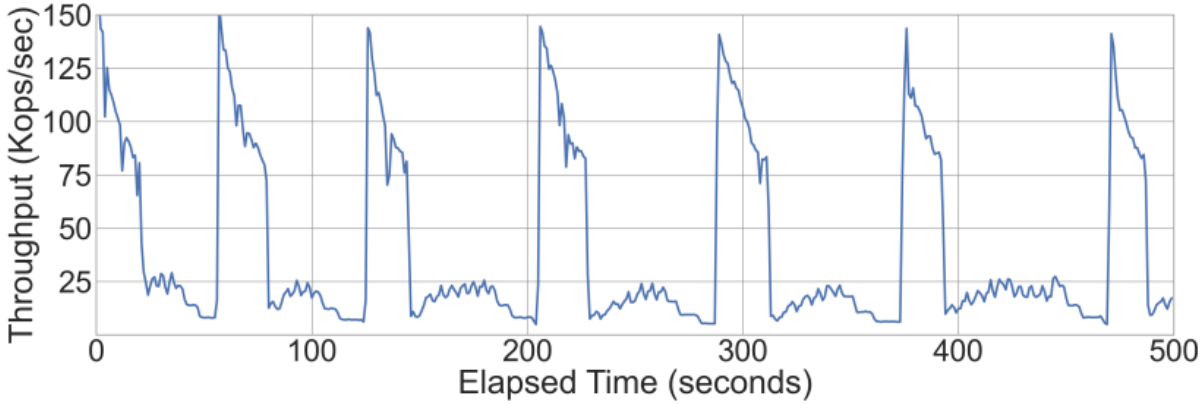


Merkle Proof

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



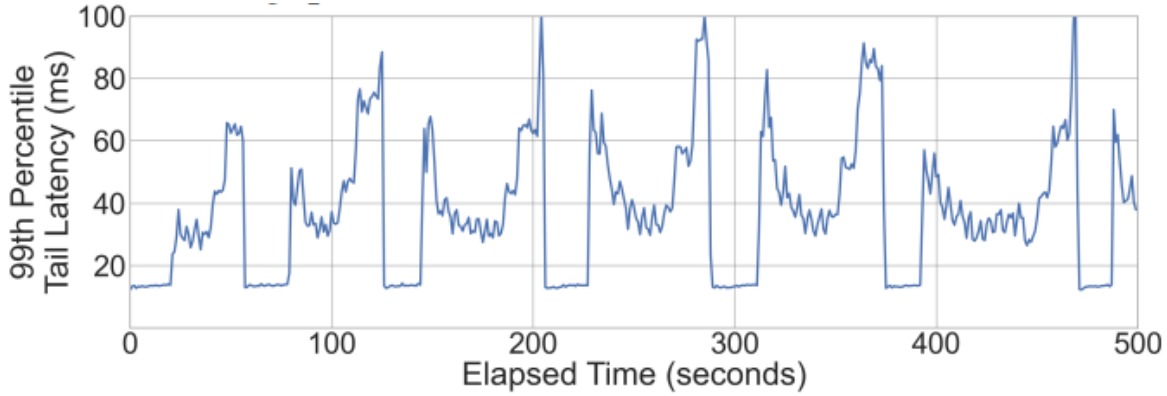


Figure source: Liang J, Chai Y. CruiseDB: An LSM-Tree Key-Value Store with Both Better Tail Throughput and Tail Latency. IEEE ICDE 21

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)

Evaluation

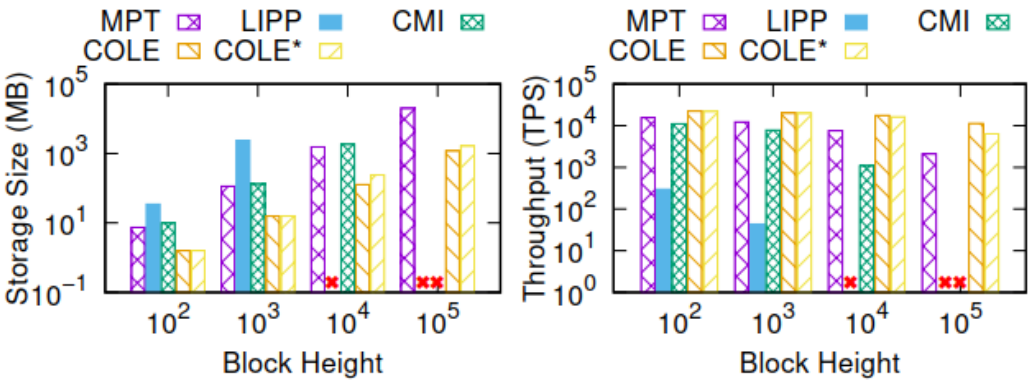


Figure 10: Performance vs. Block Height (SmallBank)

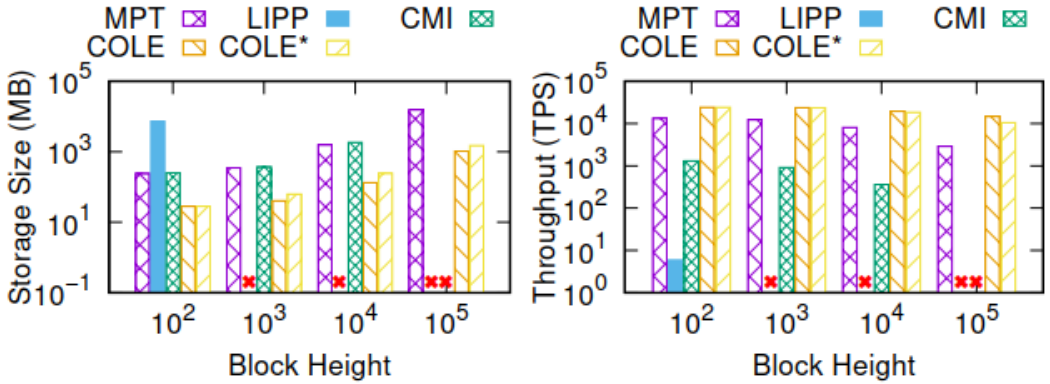


Figure 11: Performance vs. Block Height (KVStore)

- 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

Evaluation

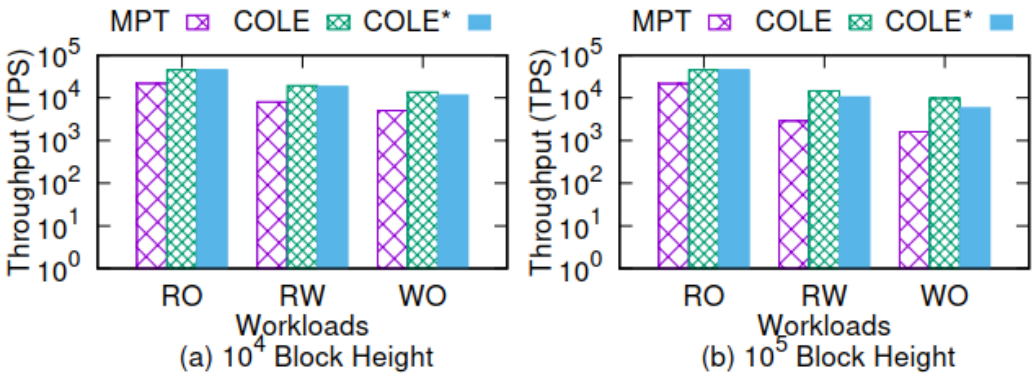


Figure 12: Throughput vs. Workloads (KVStore)

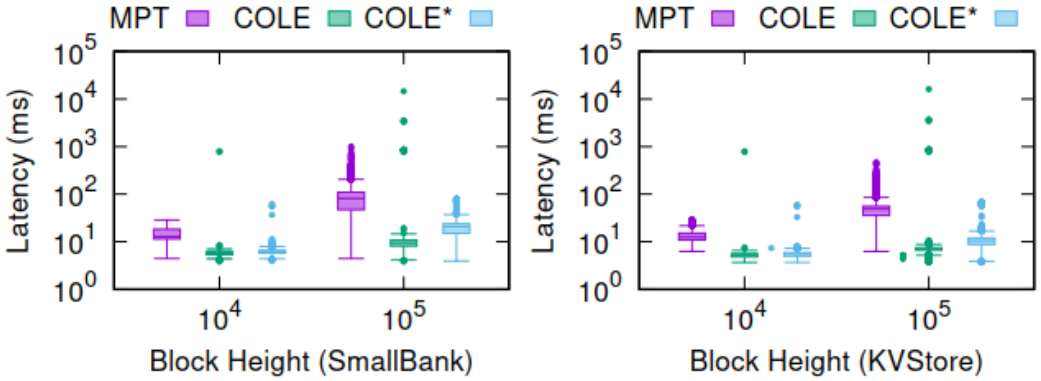


Figure 13: Latency Box plot

- 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

Evaluation

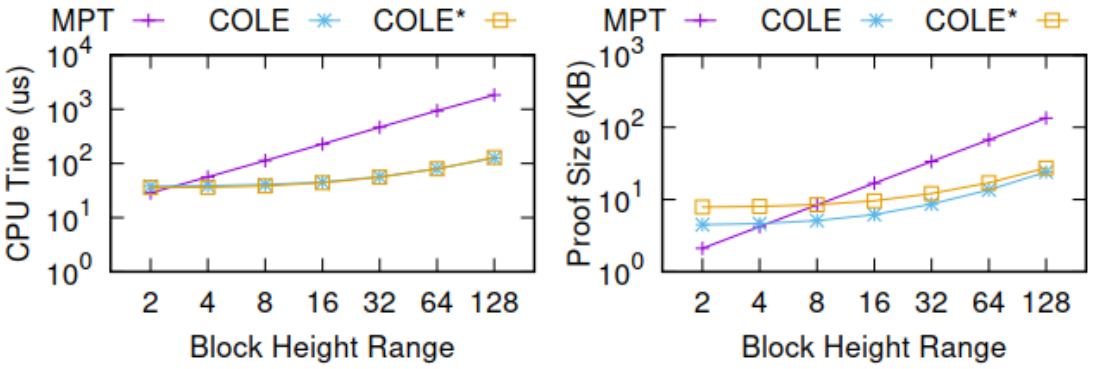


Figure 15: Prov-Query Performance vs. Query Range

- Provenance query
  - MPT's CPU time and proof size grow linearly with the block height range while COLE and COLE* grow sub-linearly

Summary

- 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