SOSP 2024

CHIME: A Cache-Efficient and High-Performance Hybrid Index on Disaggregated Memory

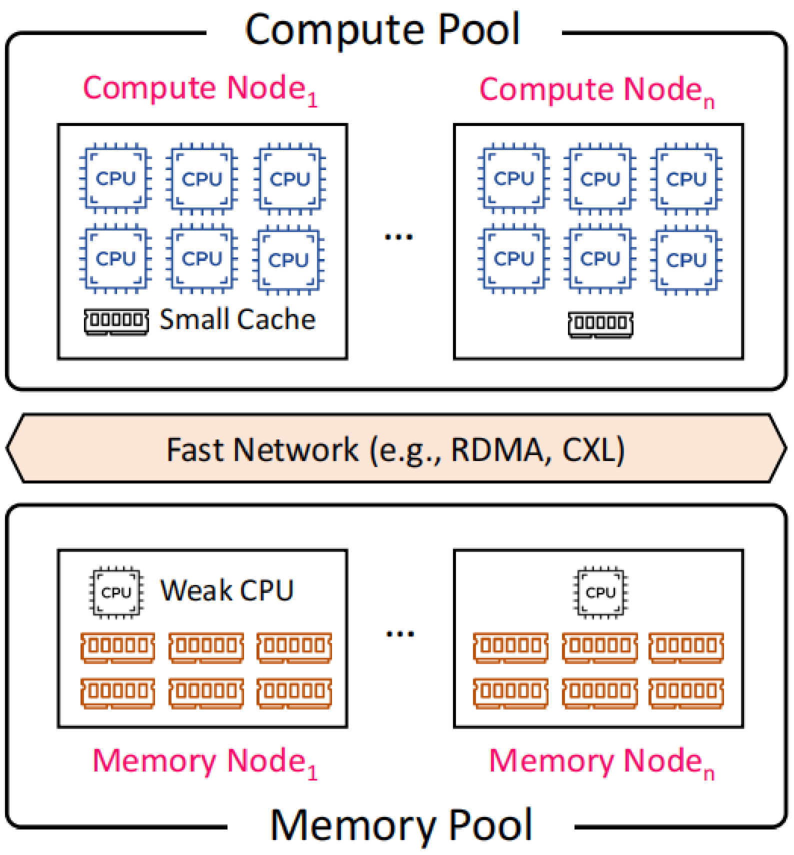
Xuchuan Luo, Jiacheng Shen, Pengfei Zuo, Xin Wang, Micheal R.Lyu, Yangfan Zhou

School of Computer Science, Fudan University
National Key Laboratory of Parallel and Distributed Computing, China
Duke Kunshan University Huawei Cloud The Chinese University of Hong Kong
Shanghai Key Laboratory of Intelligent Information Processing, Shanghai, China

Presented by Sen Han

Background-DM

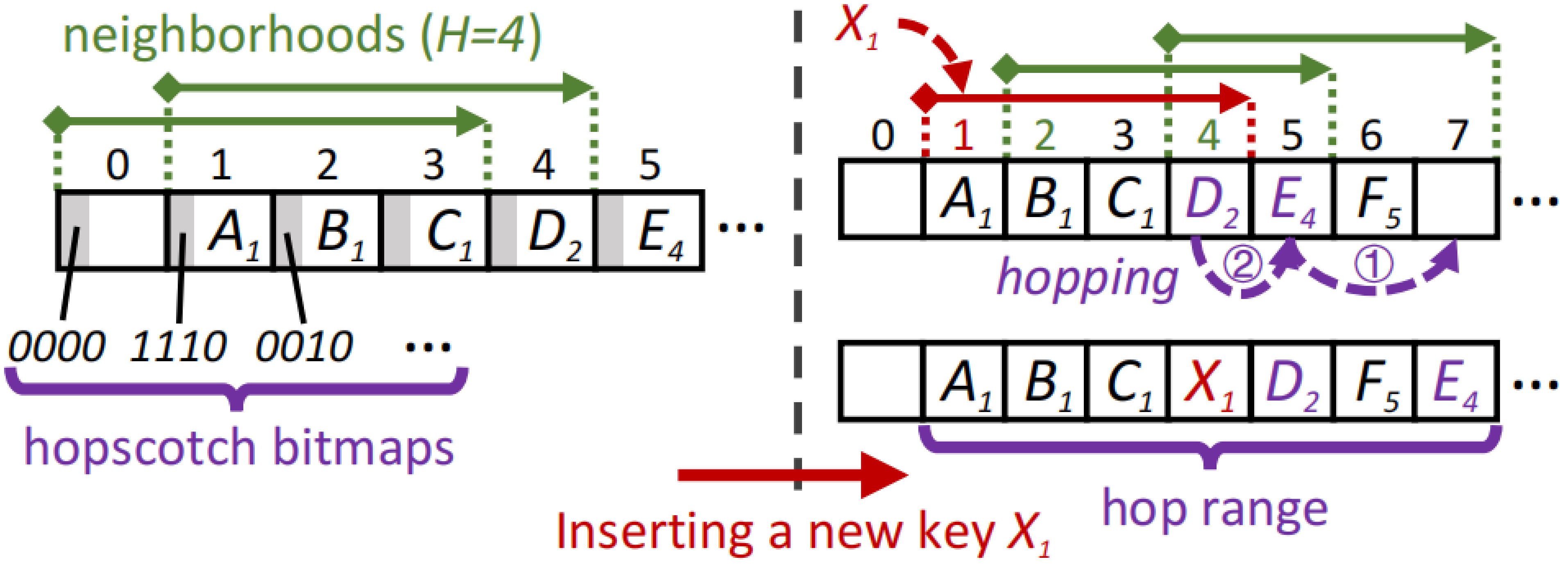
Disaggregated Memory(DM)



☺ **Benefits:**
- ✓ Resource utilization
- ✓ Elasticity

Background-HopScotch Hashing

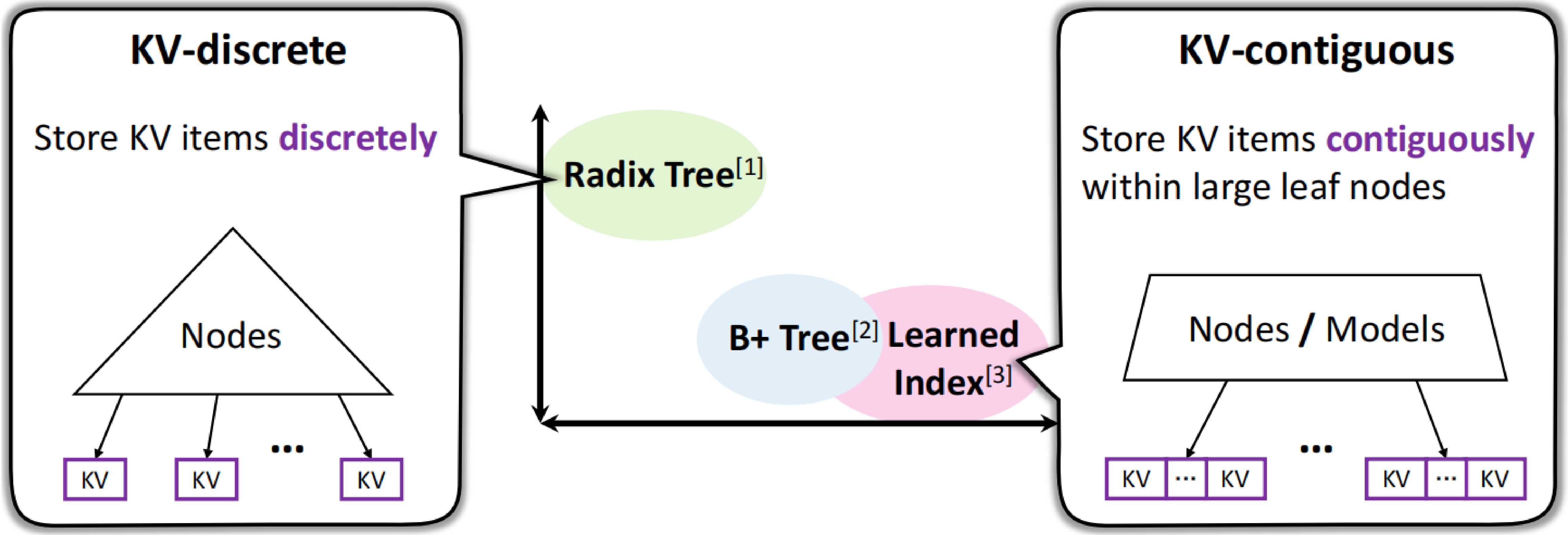
HopScotch Hashing



Range Indexes on Disaggregated Memory

USTC, CHINA ADSLAB

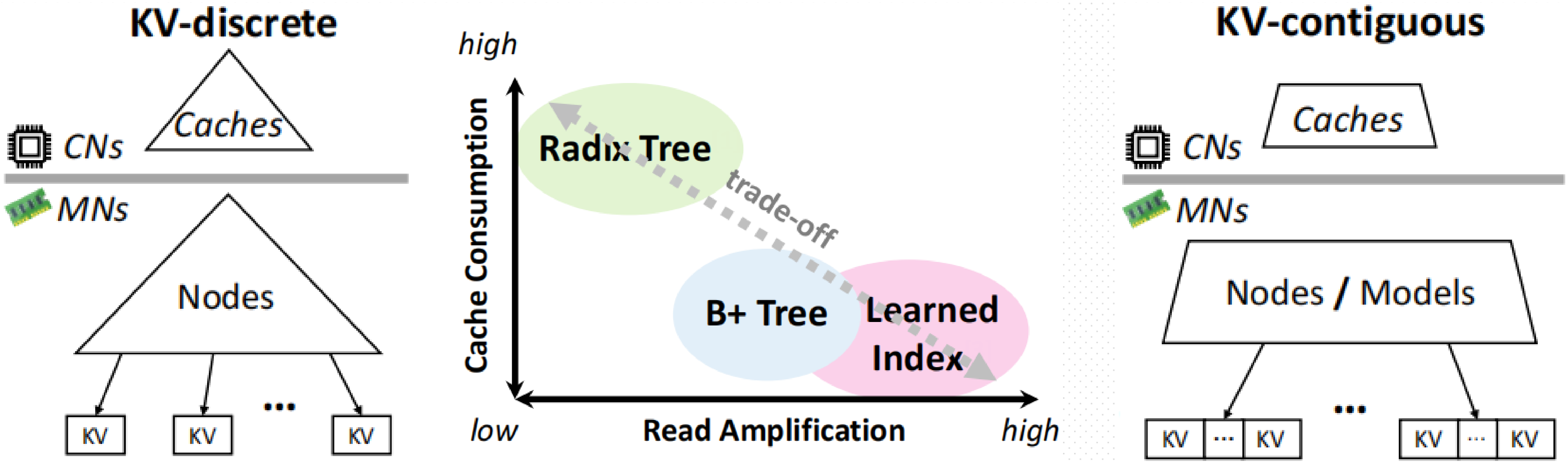
Existing range indexes on DM can be classified into two types:



[1] Xuchuan Luo et al. SMART: A high-performance adaptive radix tree for disaggregated memory. OSDI 2023.
[2] Qing Wang et al. Sherman: A write-optimized distributed B+ tree index on disaggregated memory. SIGMOD 2022.
[3] Pengfei Li et al. ROLEX: A scalable RDMA-oriented learned key-value store for disaggregated memory. FAST 2023.

A Trade-off for Range indexes on DM

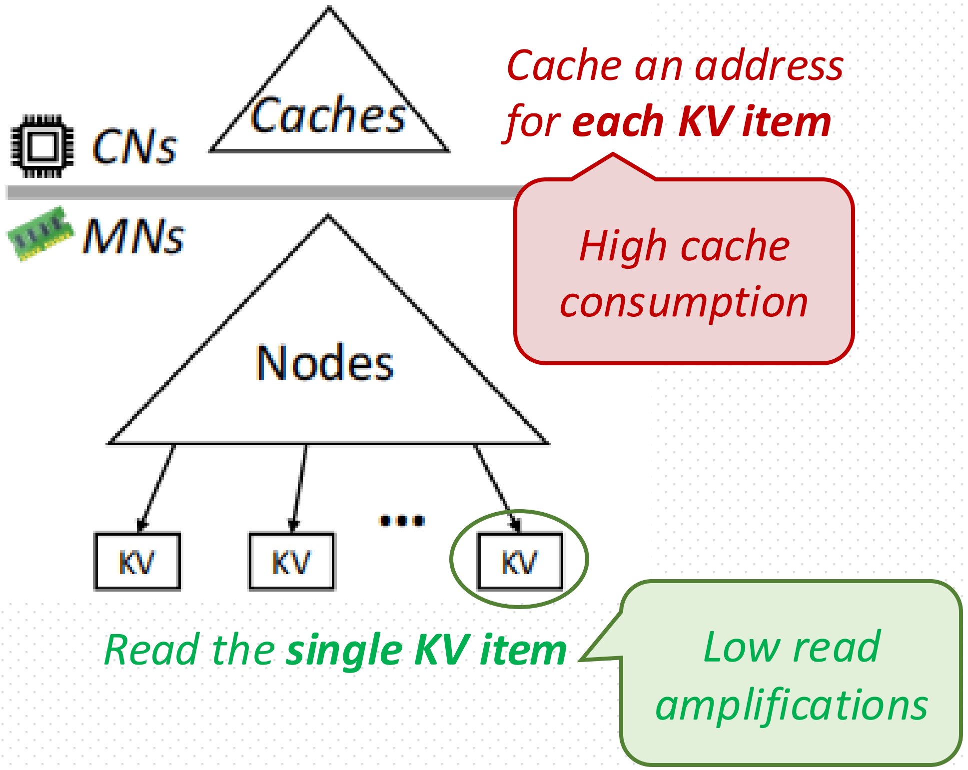
There is a trade-off between read amplifications and cache consumption:



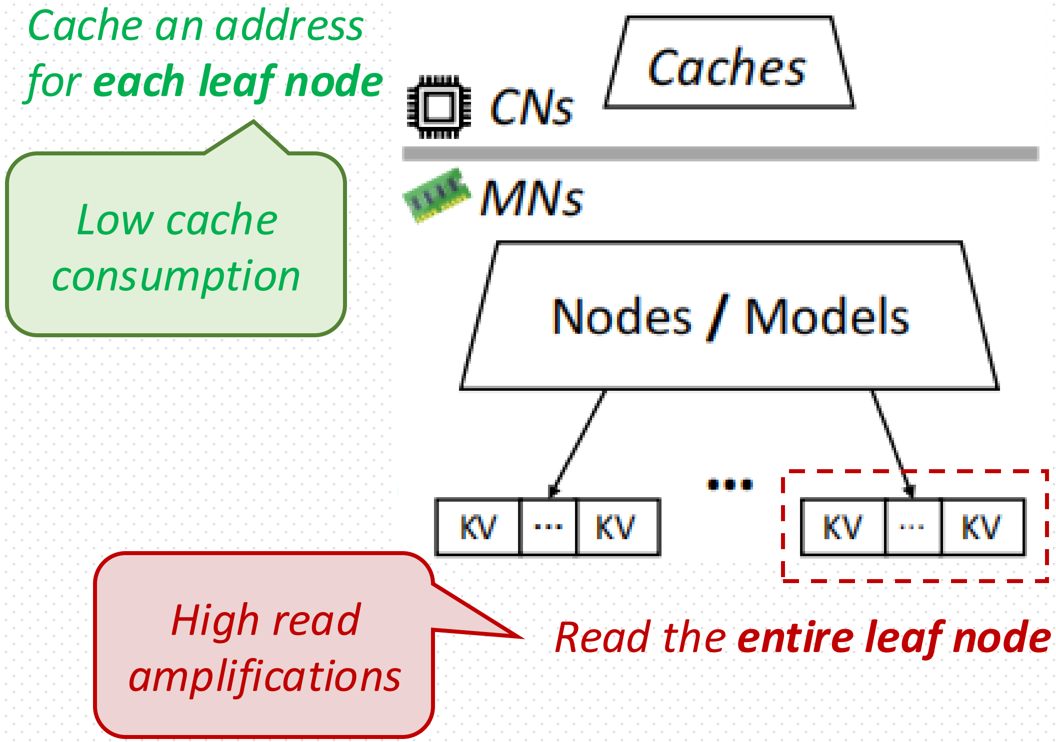
A Trade-off for Range indexes on DM

There is a trade-off between read amplifications and cache consumption:

KV-discrete

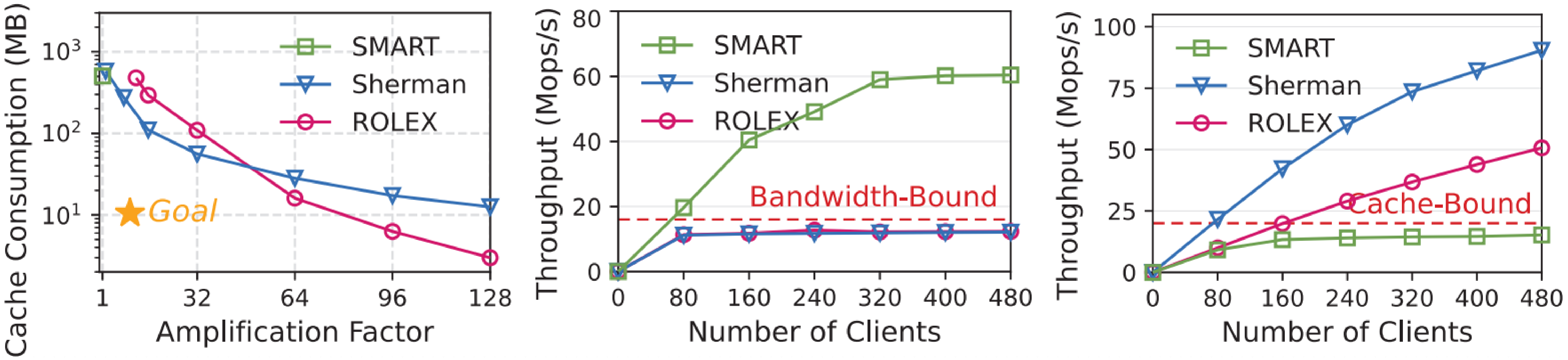


KV-contiguous



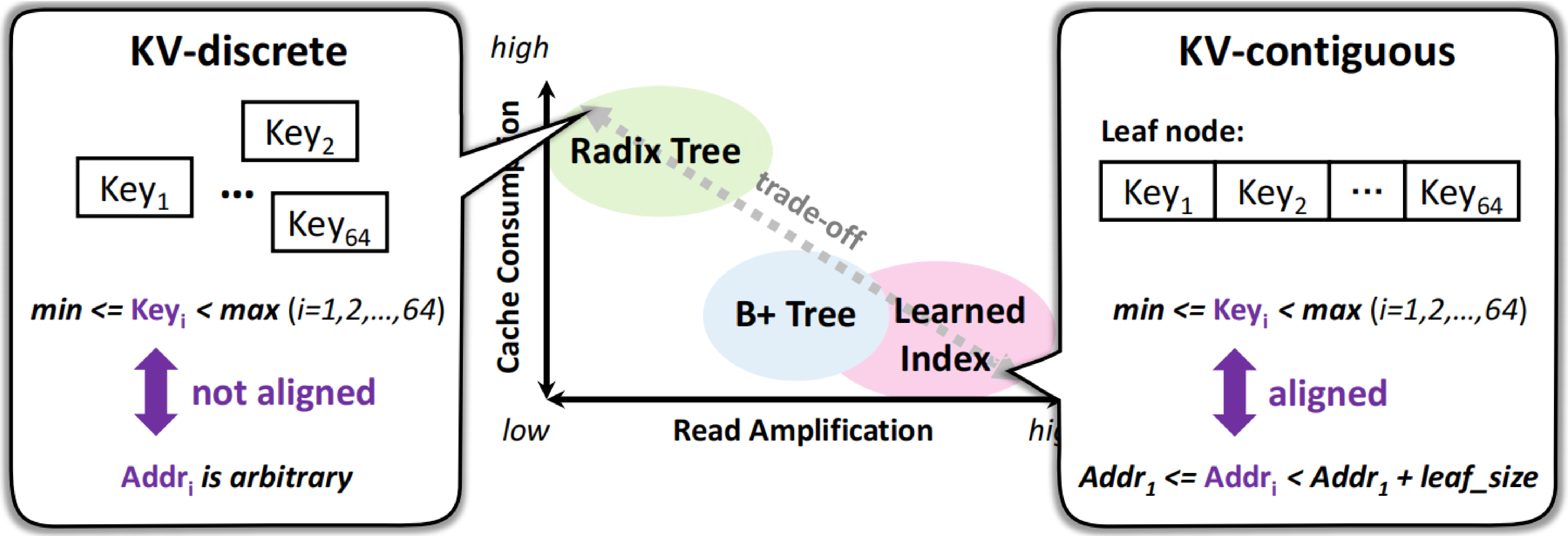
A Trade-off for Range indexes on DM

There is a trade-off between read amplifications and cache consumption:



A Trade-off for Range indexes on DM

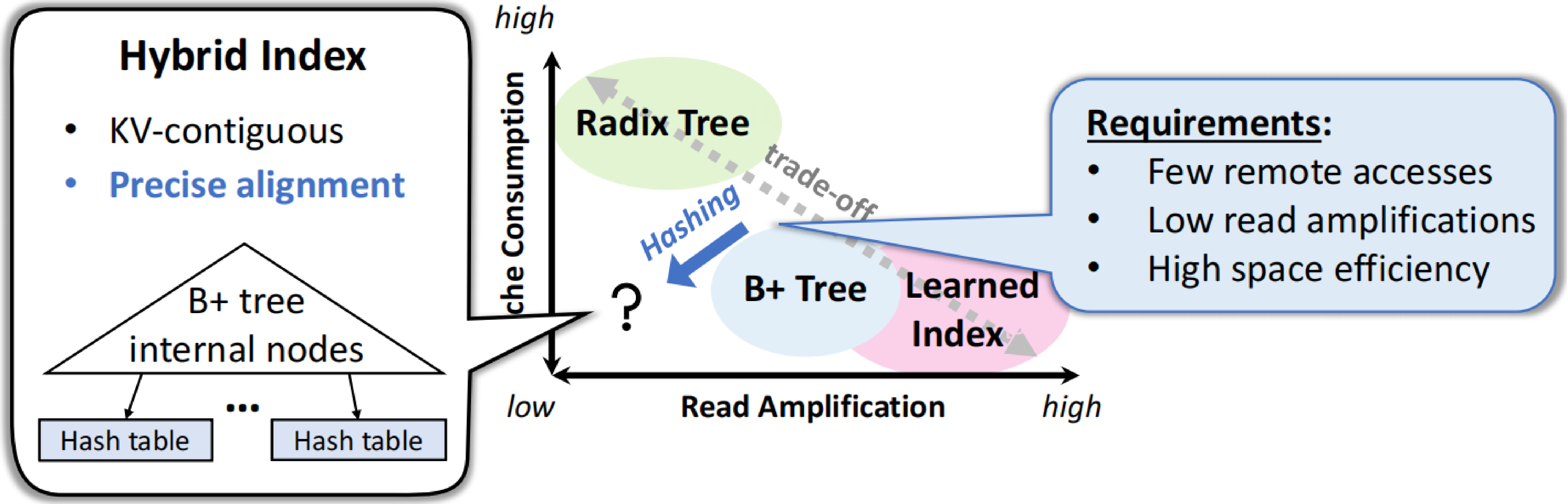
Root Cause: The alignment between keys and memory addresses:



Straightforward Idea

USTC, CHINA ADSLAB

Use a KV-contiguous index (e.g., B+ tree) with hash-table-based leaf nodes



Widely Choose a Suitable Hashing Scheme

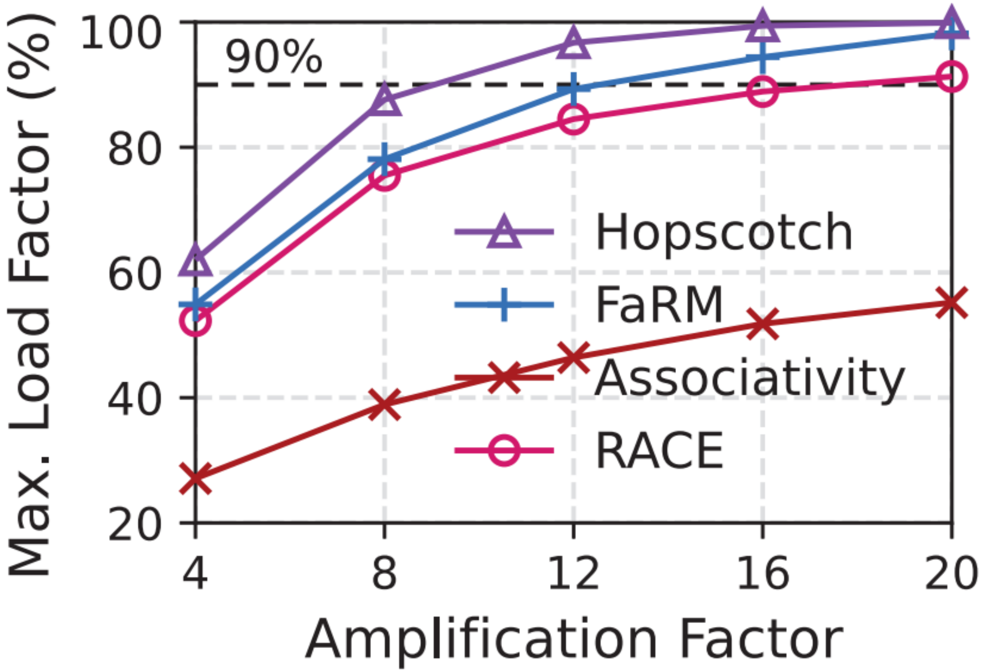
USTC, CHINA ADSLAB

Remote access

Few accesses:

- *Simple associativity*
- *Hopscotch hashing*[1]
- *FaRM*[2]
- *RACE*[3]

Read amplifications & Space efficiency

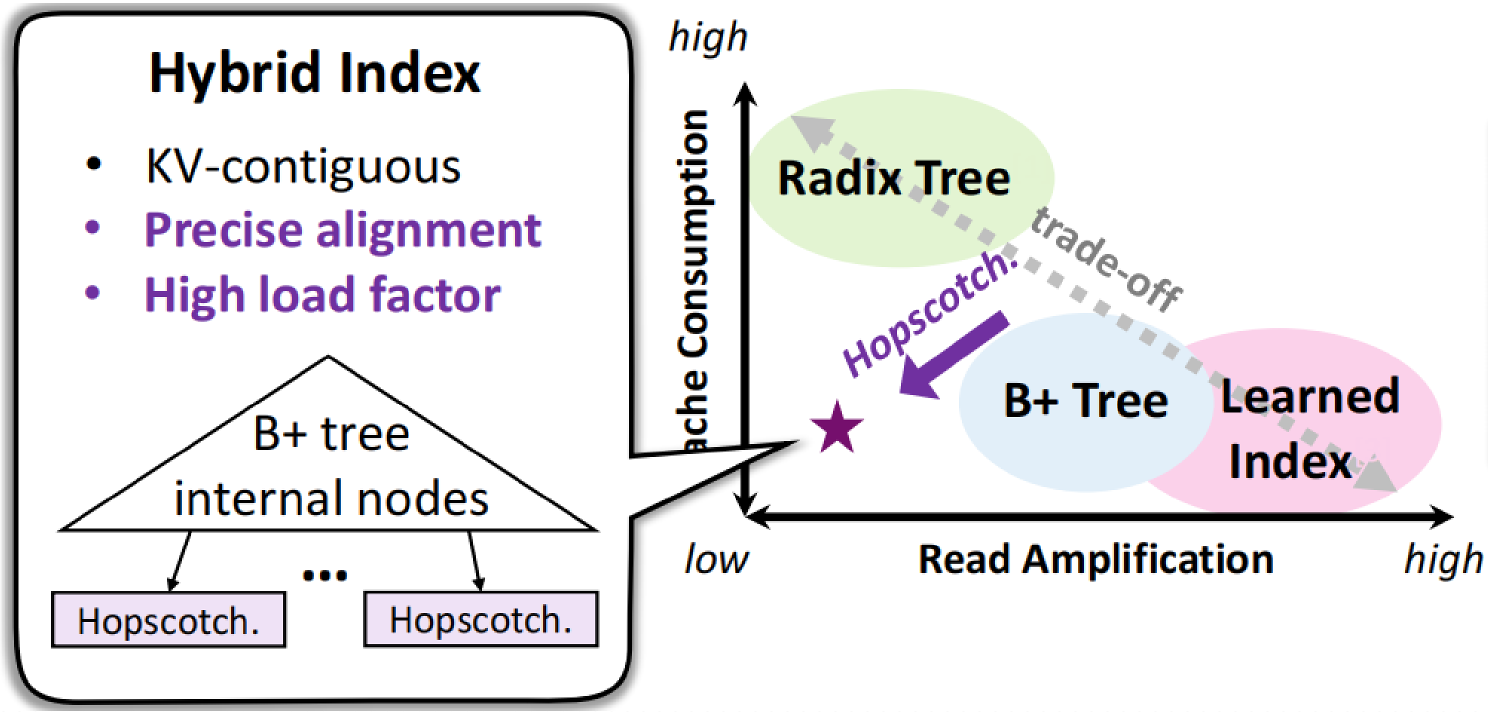


Hopscotch hashing best fits the requirements

[1] Maurice Herlihy et al. Hopscotch hashing. DISC 2008.
[2] Aleksandar Dragojevic et al. FaRM: Fast Remote Memory. NSDI 2014.
[3] Pengfei Zuo et al. One-sided RDMA-conscious extendible hashing for disaggregated memory. ATC 2021.

Viable Idea

Use a hybrid index combining a B+ tree with hopscotch hashing

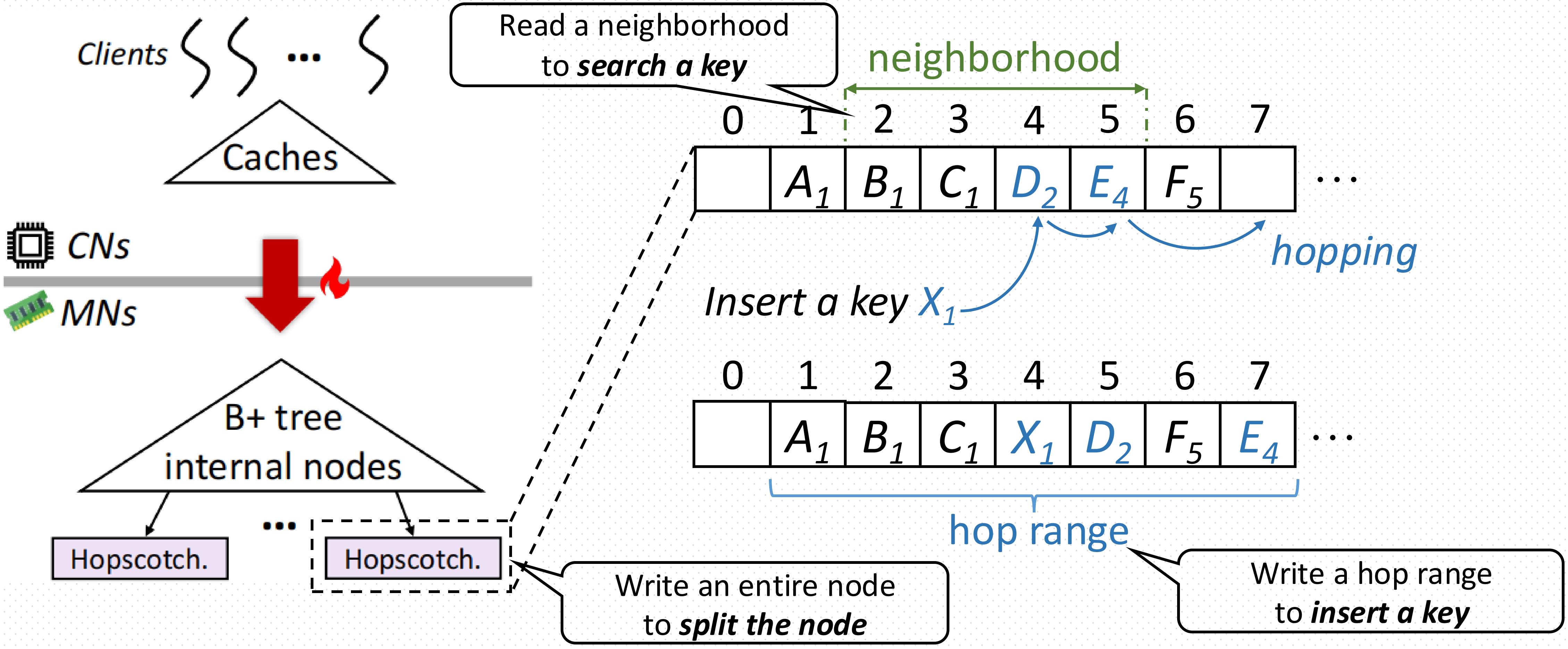


☺ **Benefit:**

- ✓ Cache-efficient
- ✓ Bandwidth-efficient
- ✓ Space-efficient

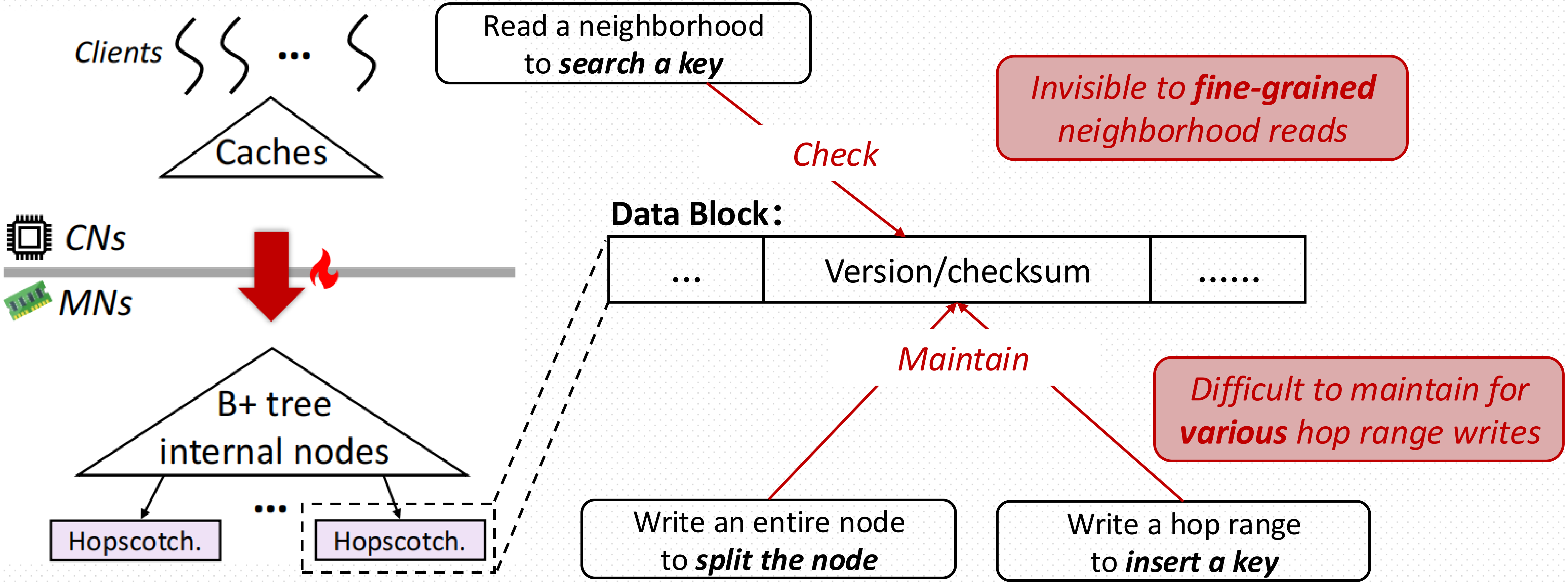
Challenge

Various granularities in reads/writes complicate optimistic synchronization



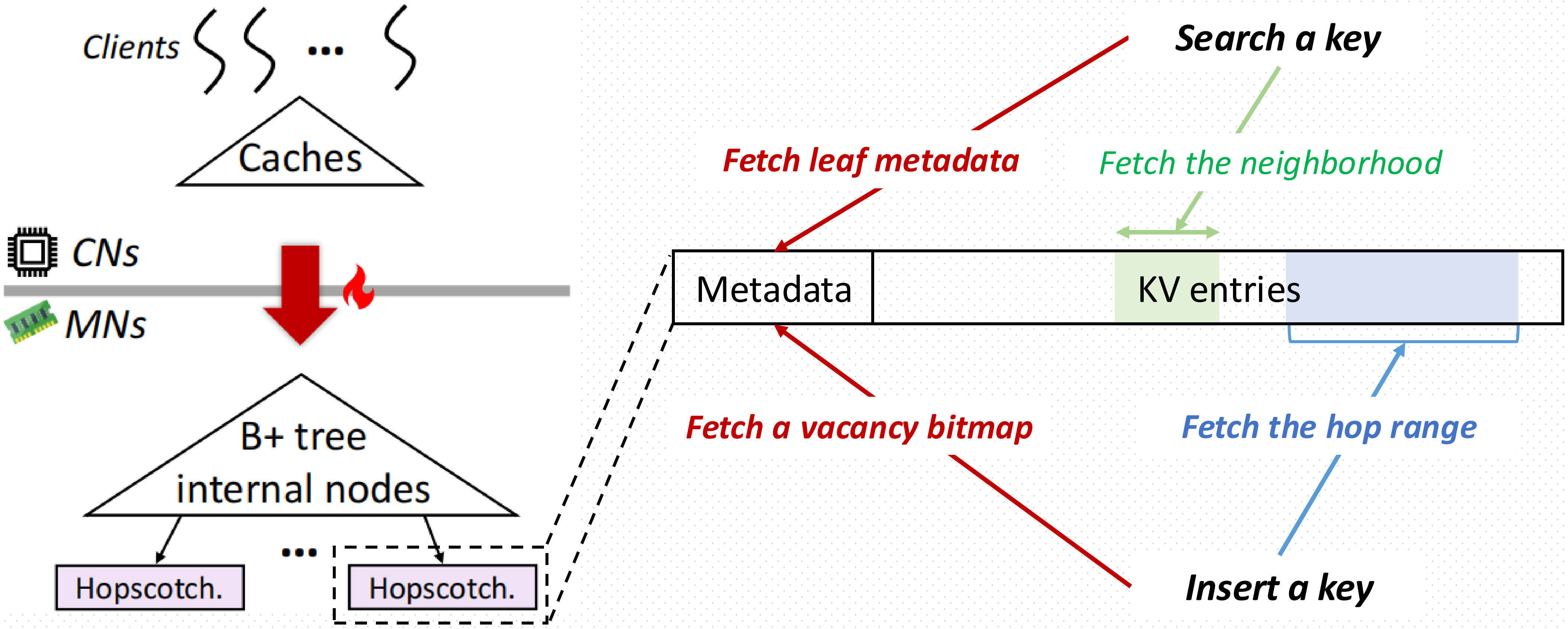
Challenge

Various granularities in reads/writes complicate optimistic synchronization



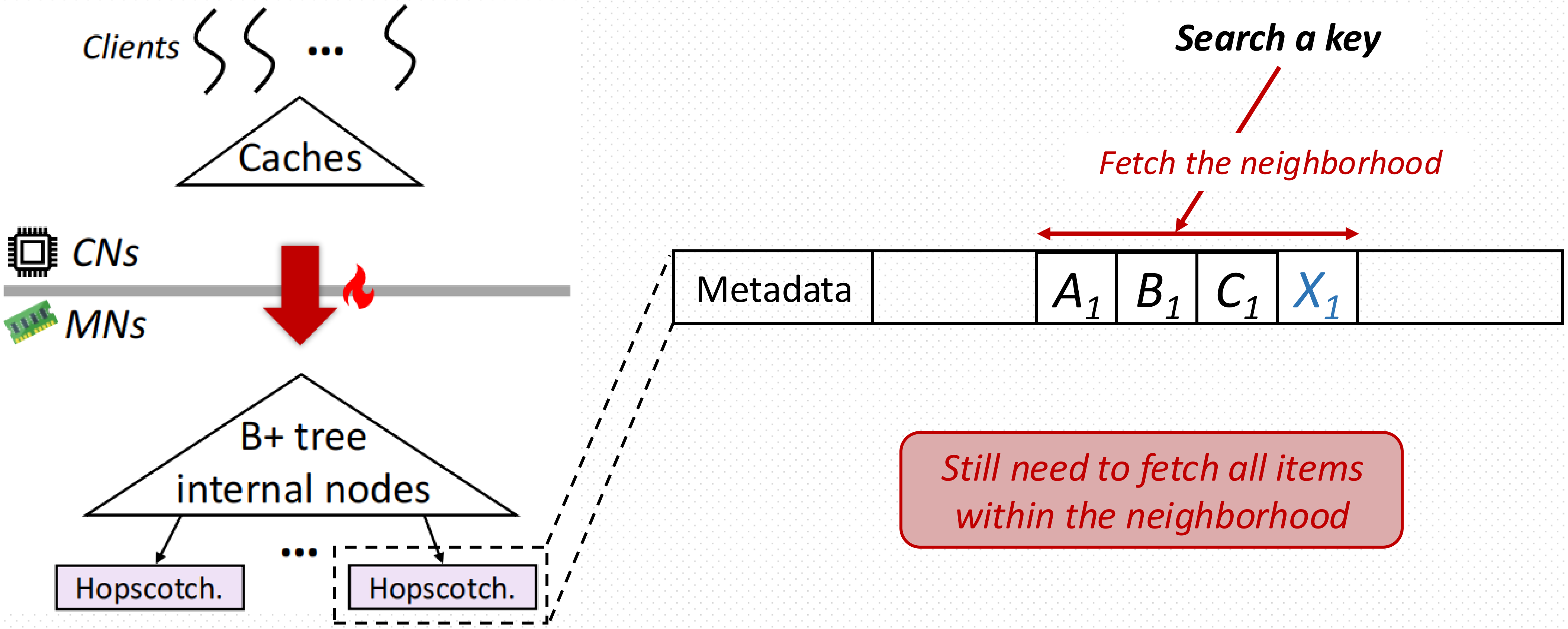
Challenge

Metadata for B+ trees and hopscotch hashing induces extra remote accesses

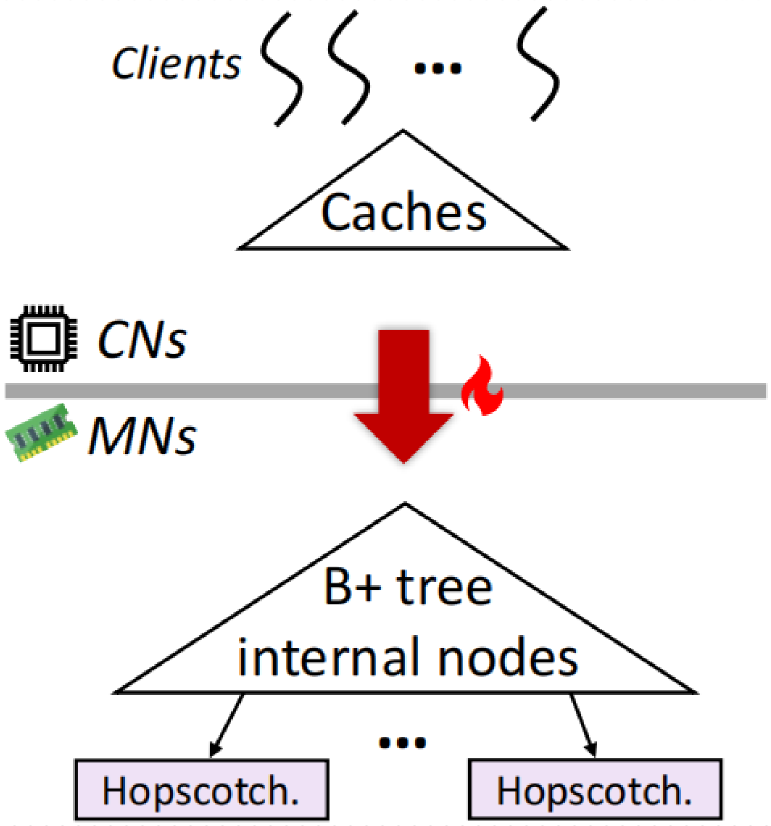


Challenge

Hopscotch hash still incurs read amplifications compared with reading KV



Challenge Summary



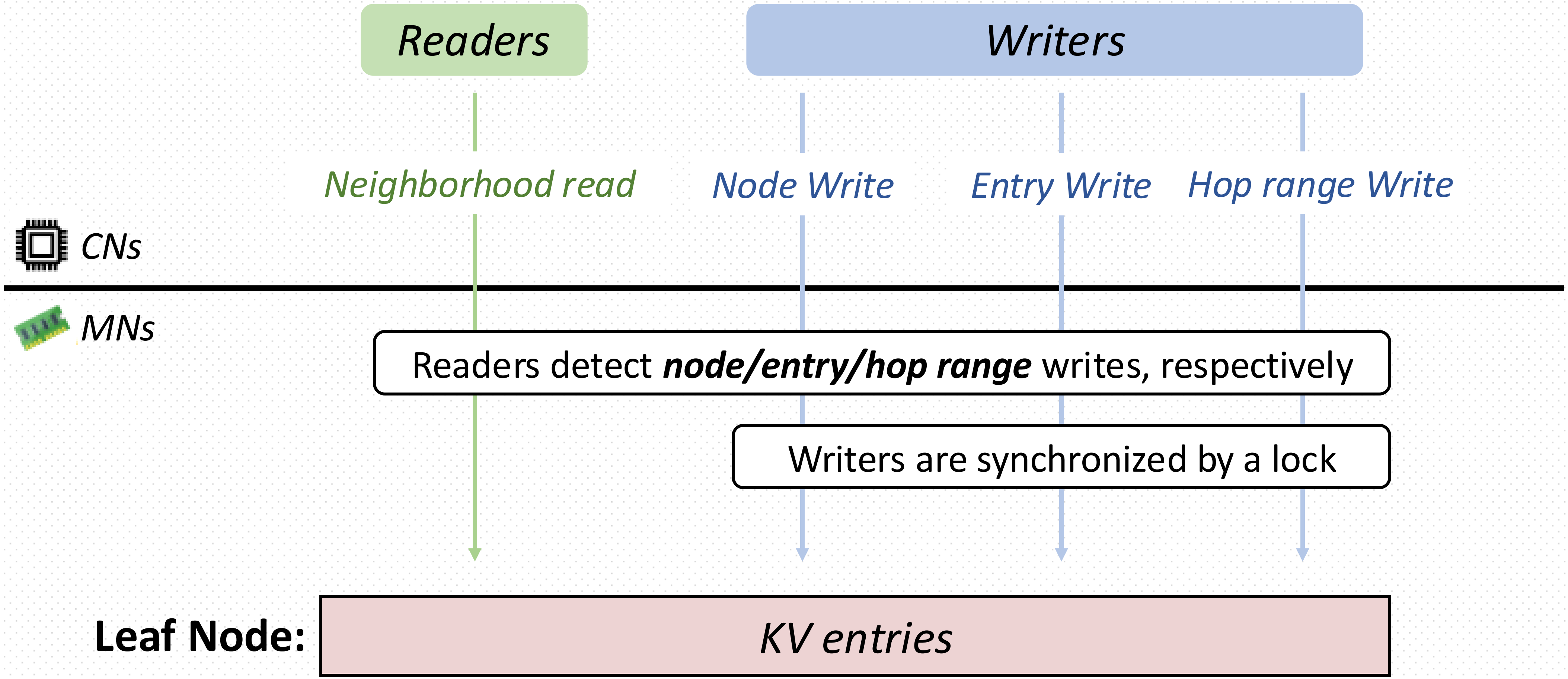
1. Complicated Optimistic Synchronization

2. Extra Metadata Accesses

3. Read Amplifications of Hopscotch Hashing

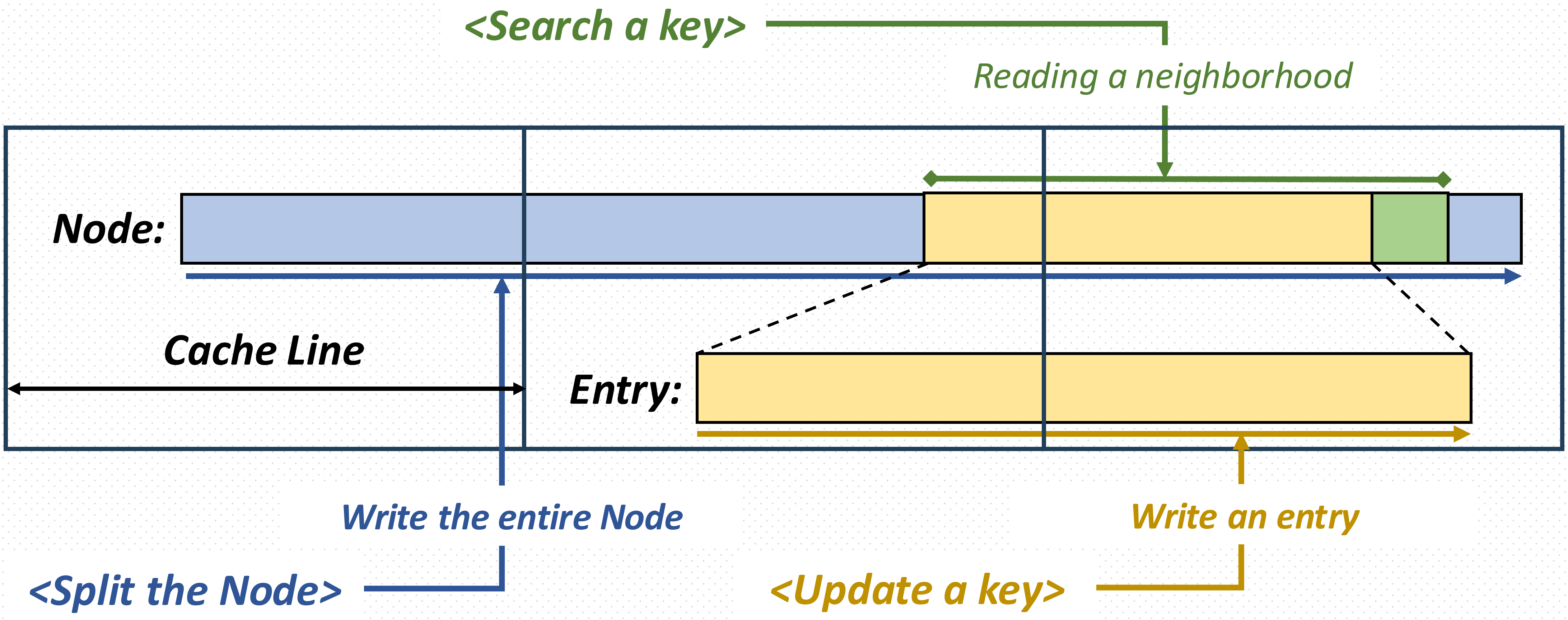
Three-level Optimistic Synchronization

Synchronization Overview



Three-level Optimistic Synchronization

Level 1 & Level 2: Detect the Node Write & Entry Write

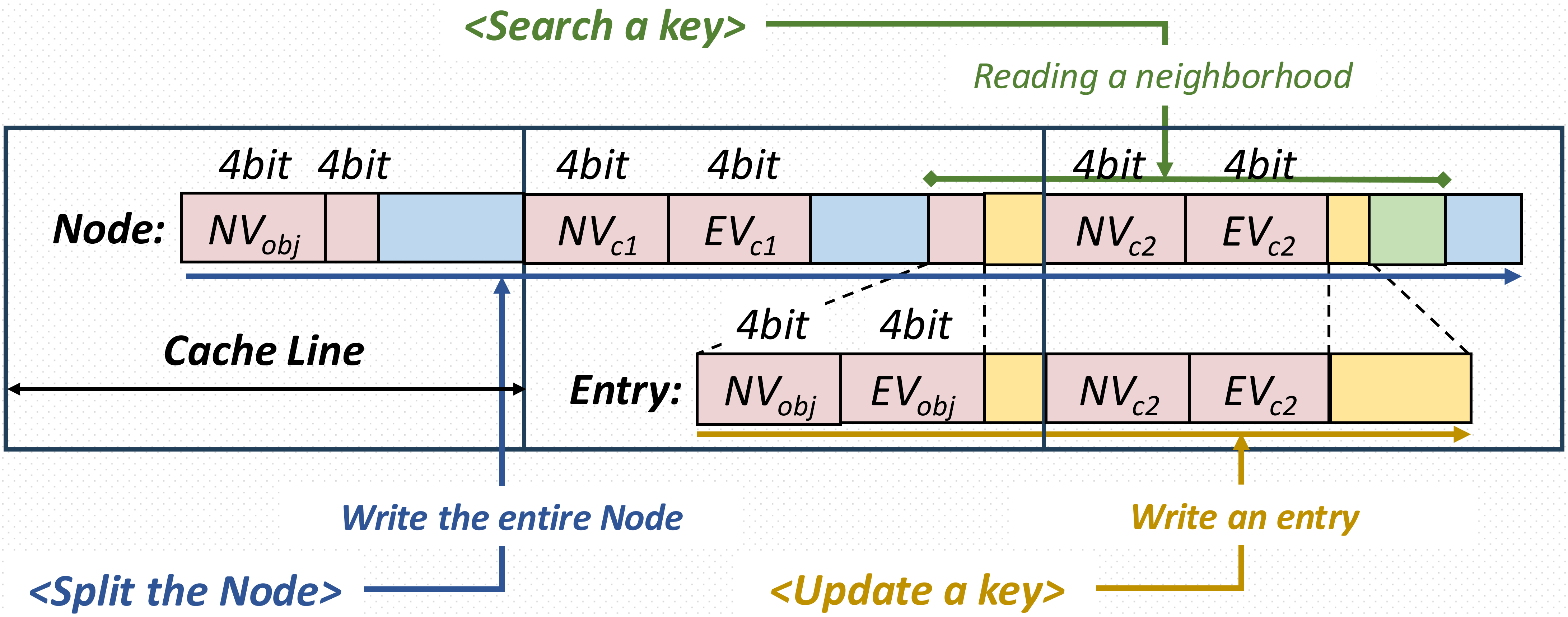


Three-level Optimistic Synchronization

USTC, CHINA ADSLAB

Level 1 & Level 2: Detect the Node Write & Entry Write

→ ***Solution:*** *Use two-level cache line versioning*

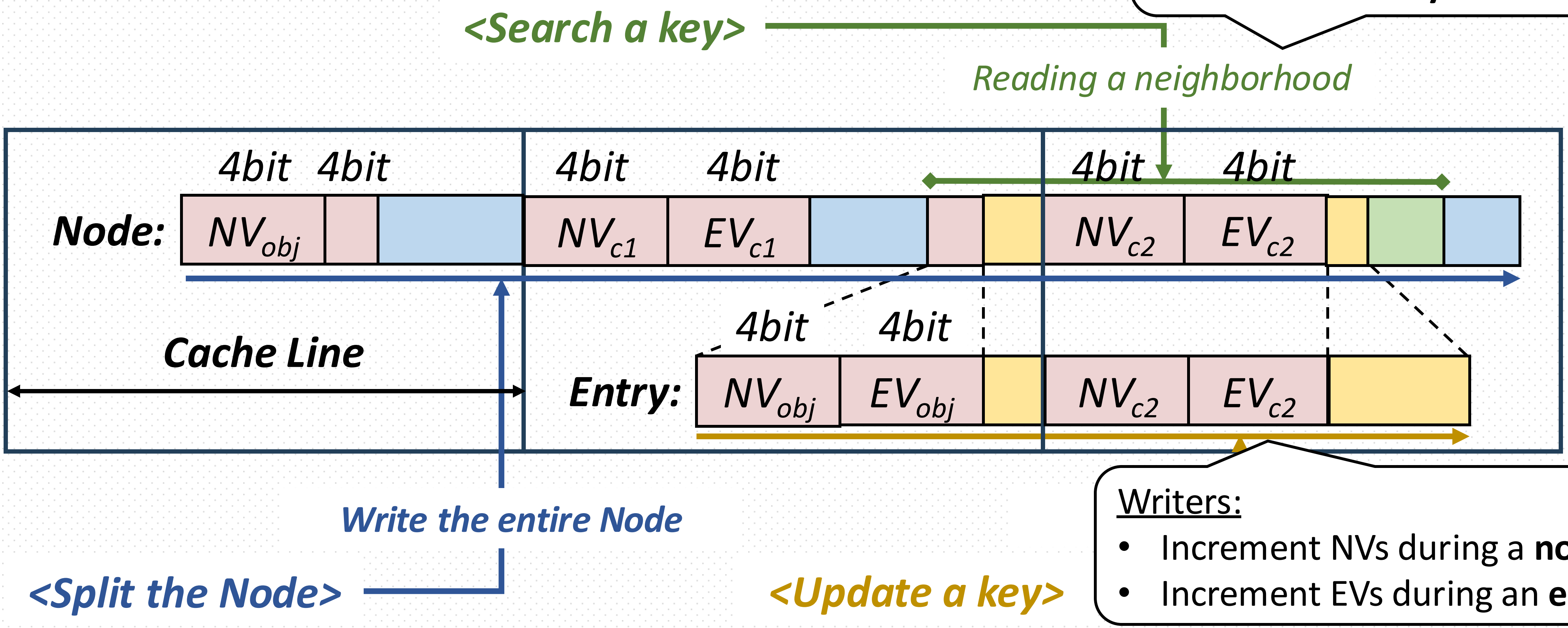


Three-level Optimistic Synchronization

Level 1 & Level 2: Detect the Node Write & Ent

→ ***Solution:*** *Use two-level cache line versioning*

Readers:
- Check the **node write** via NVs
- Check the **entry write** via EVs

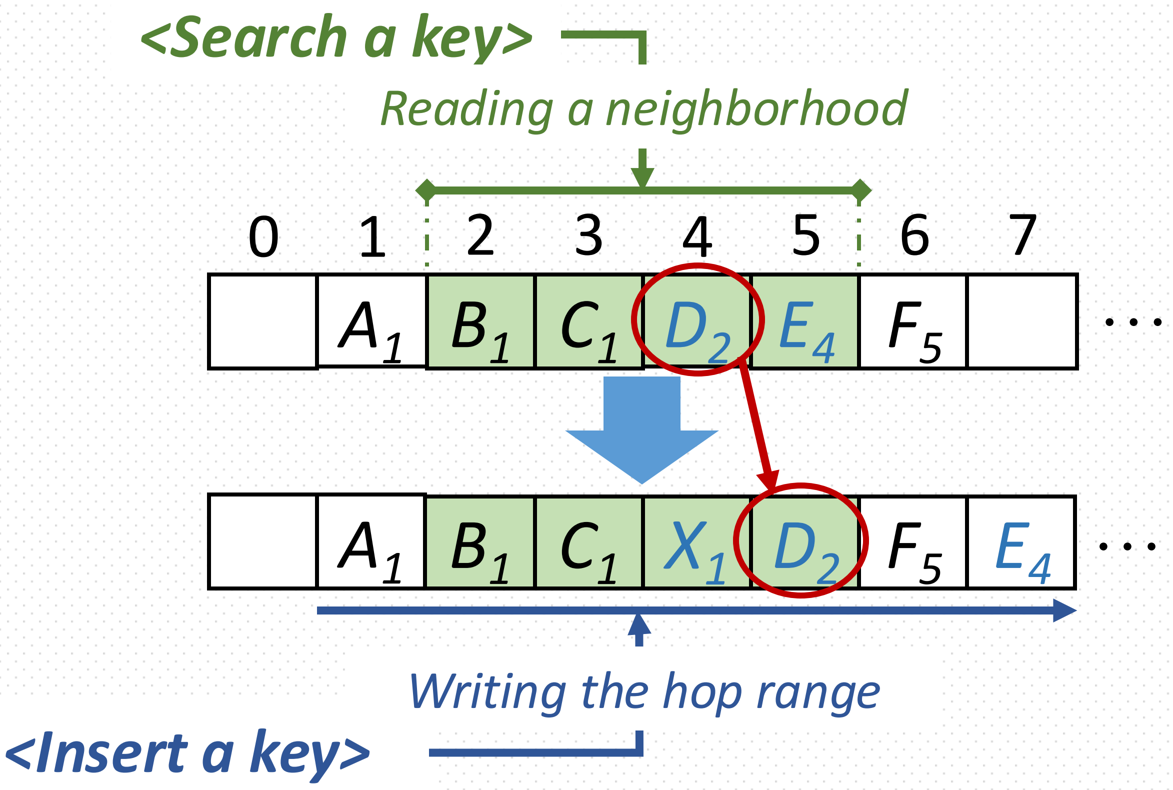


Writers:
- Increment NVs during a **node write**
- Increment EVs during an **entry write**

Three-level Optimistic Synchronization

Level 3: Detect the hop range write

Problem: *Location changes of hooped items*



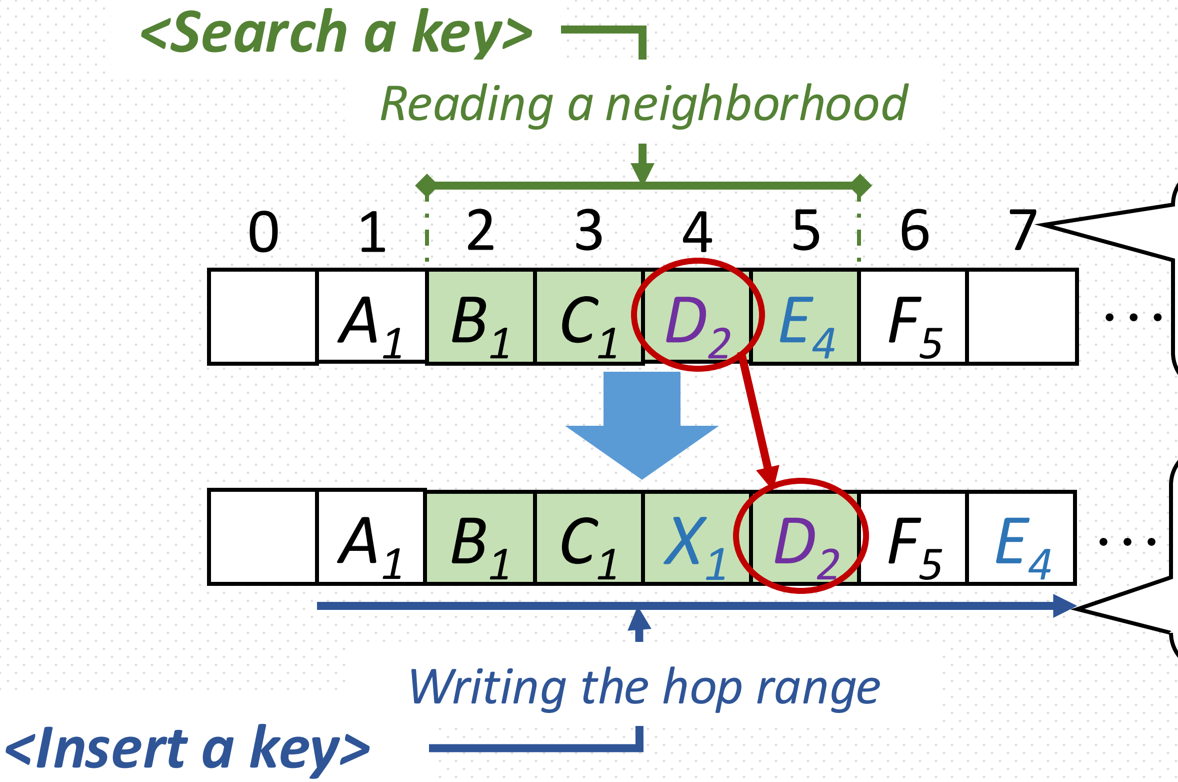
Three-level Optimistic Synchronization

USTC, CHINA ADSLAB

Level 3: Detect the hop range write

Problem: *Location changes of hoped items*

➡ ***Solution***: Reuse the hopscotch bitmaps



Readers:

- **Re-construct** the bitmap according to fetched keys
- Retry if the two bitmaps cannot match

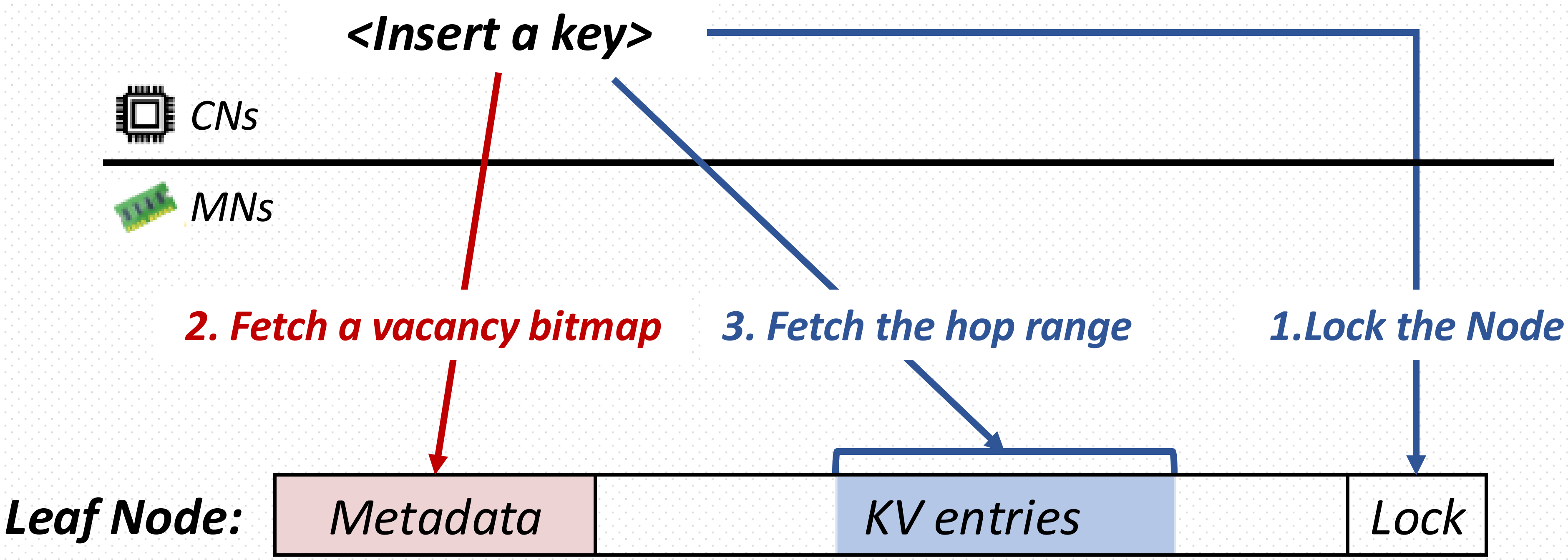
Writers:

Hopscotch hashing **has maintained a bitmap inside each neighborhood** to track the occupancy status

Access-Aggregated Metadata Management

Metadata for hopscotch hashing

Problem: Vacancy bitmaps induce extra accesses



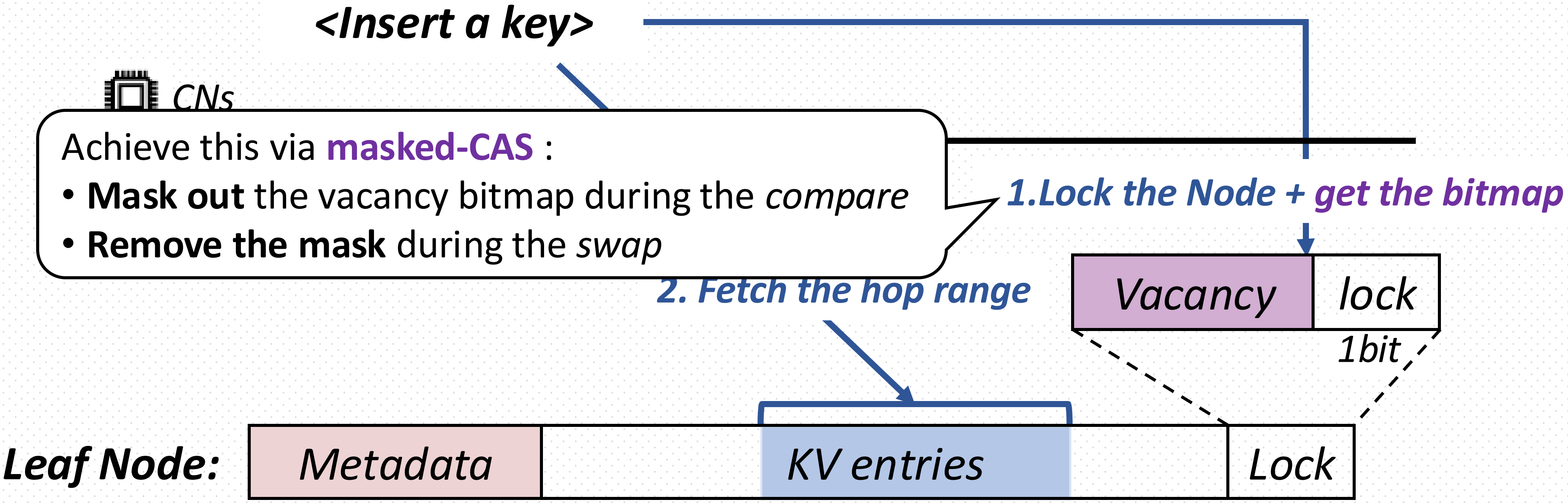
Access-Aggregated Metadata Management

USTC, CHINA ADSLAB

Metadata for hopscotch hashing

Problem: Vacancy bitmaps induce extra accesses

➡ ***Solution:*** *Piggyback the vacancy bitmap*

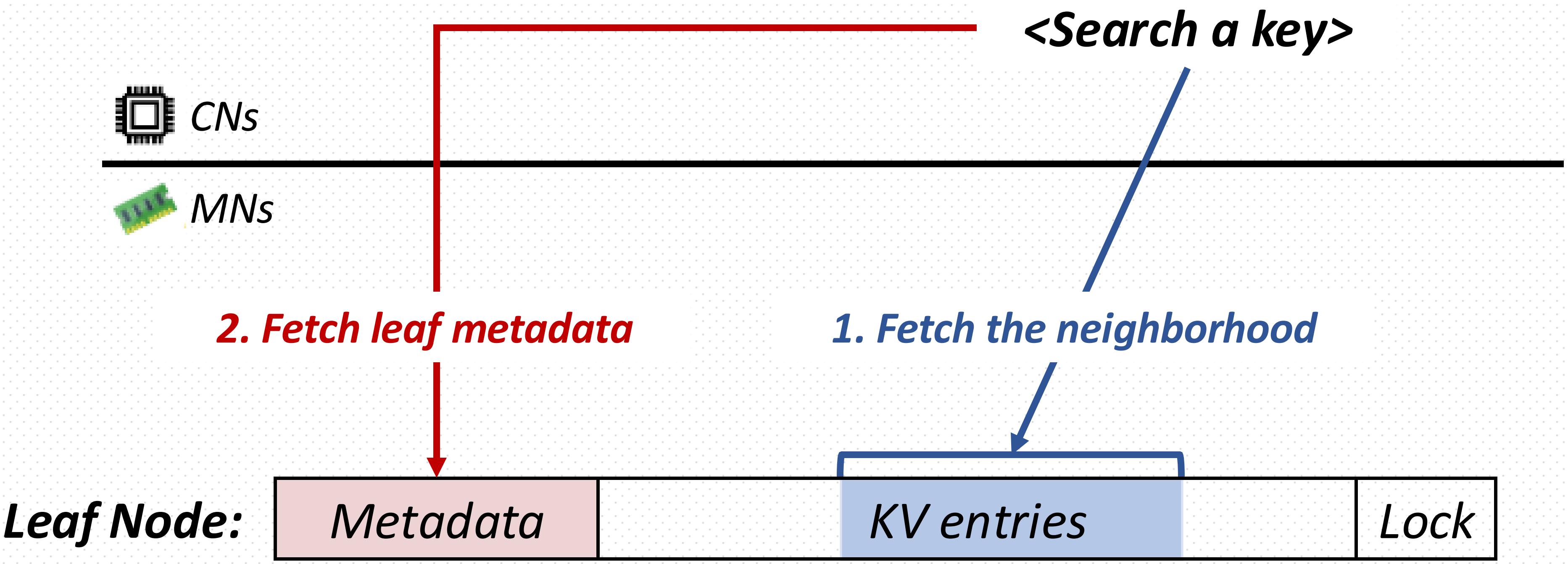


Achieve this via **masked-CAS** :
- **Mask out** the vacancy bitmap during the *compare*
- **Remove the mask** during the *swap*

Access-Aggregated Metadata Management

Metadata for the B+ tree

Problem: Leaf metadata induce extra accesses

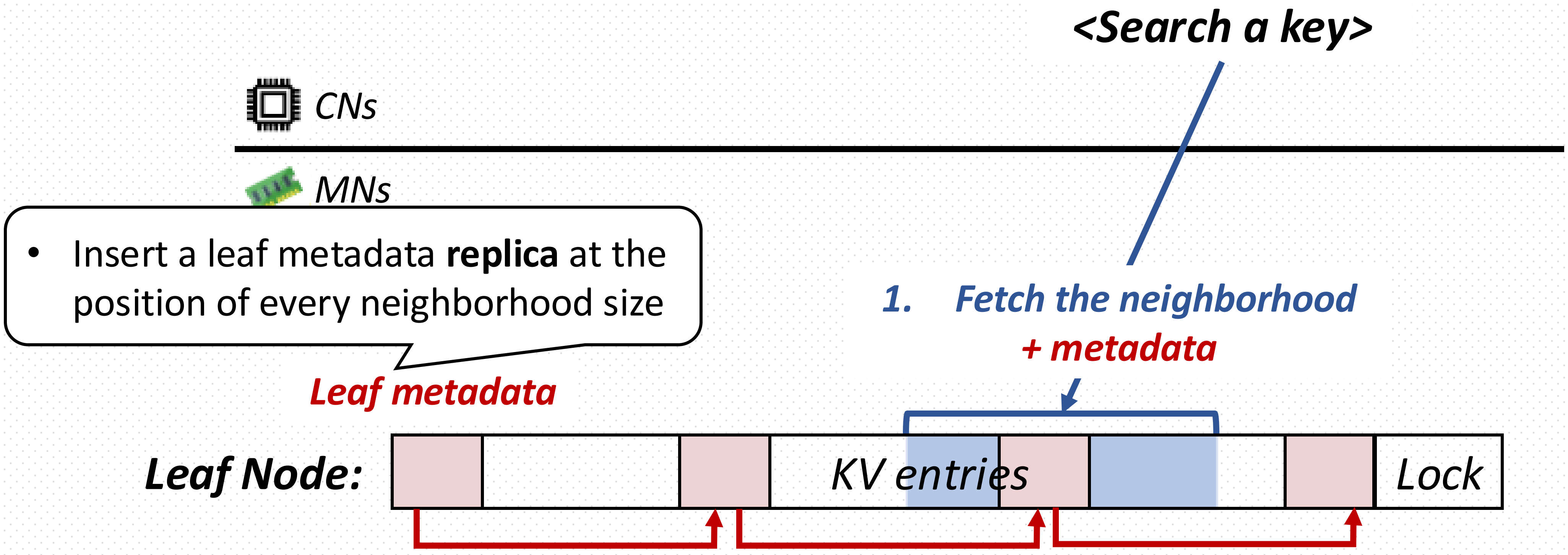


Access-Aggregated Metadata Management

Metadata for the B+ tree

Problem: Leaf metadata induce extra accesses

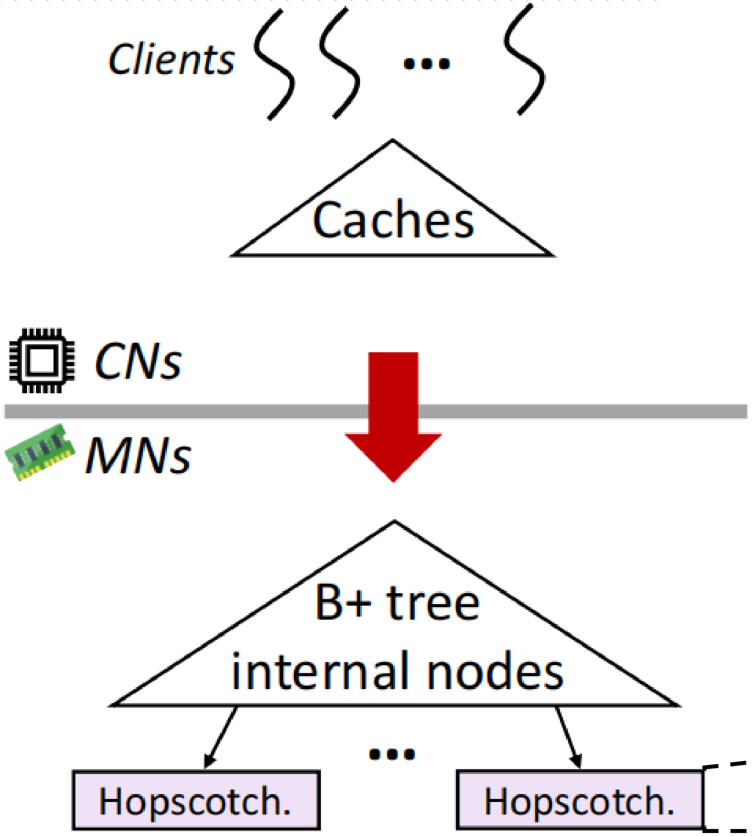
Solution: Replicate the leaf metadata

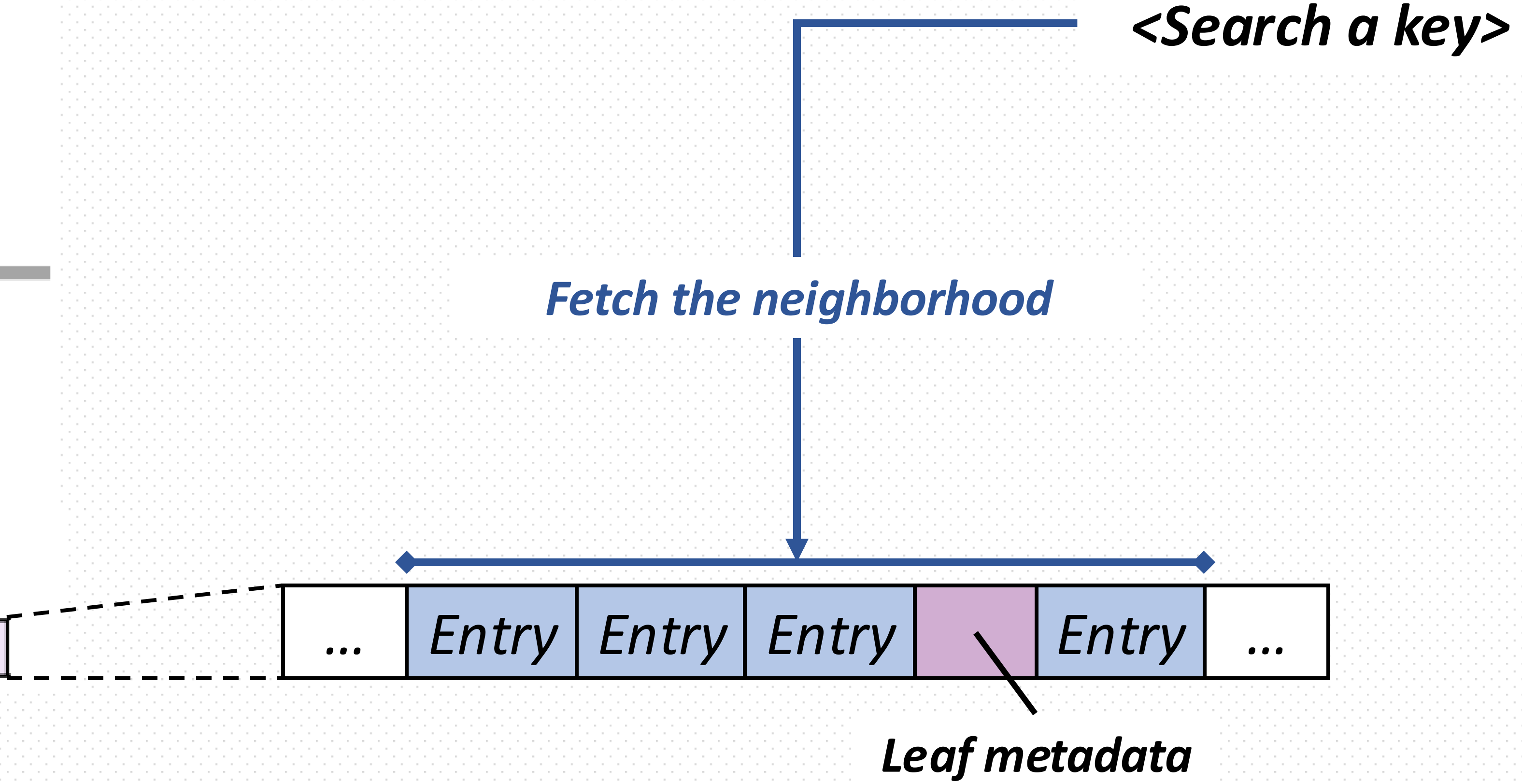


Hotness-Aware Speculative Read

Metadata for the B+ tree

Problem: Still need to fetch all items within the neighborhood





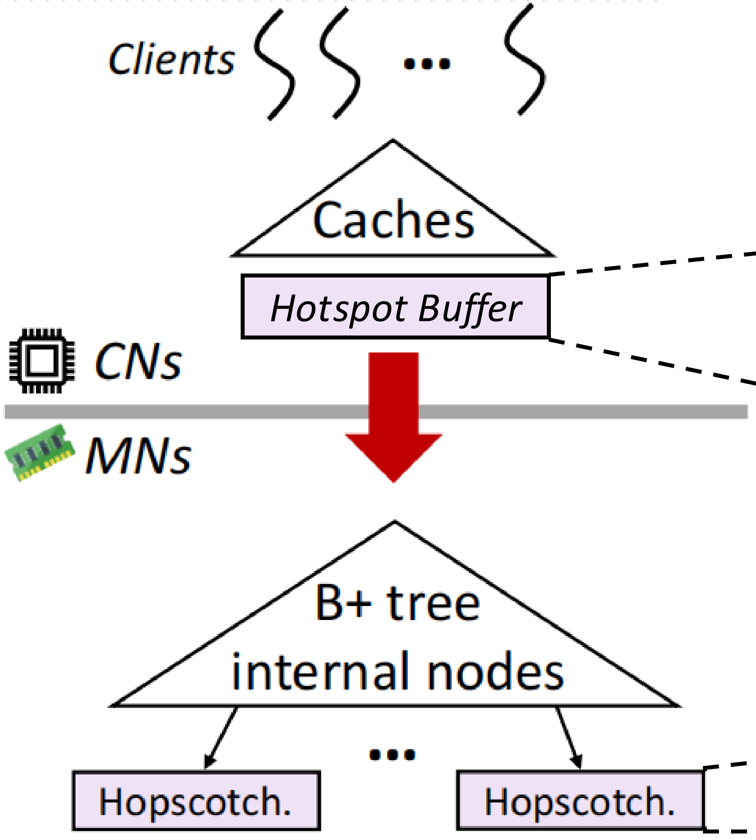
Hotness-Aware Speculative Read

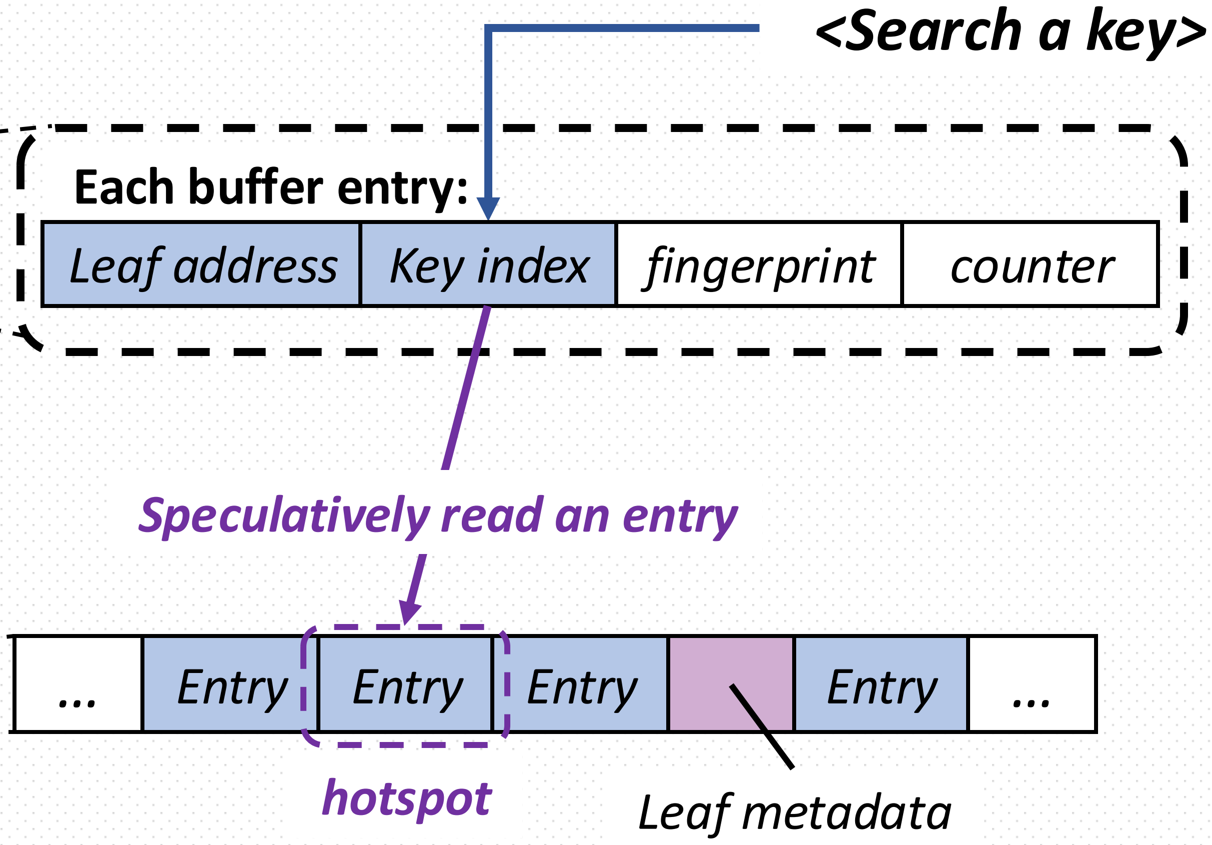
USTC, CHINA ADSLAB

Metadata for the B+ tree

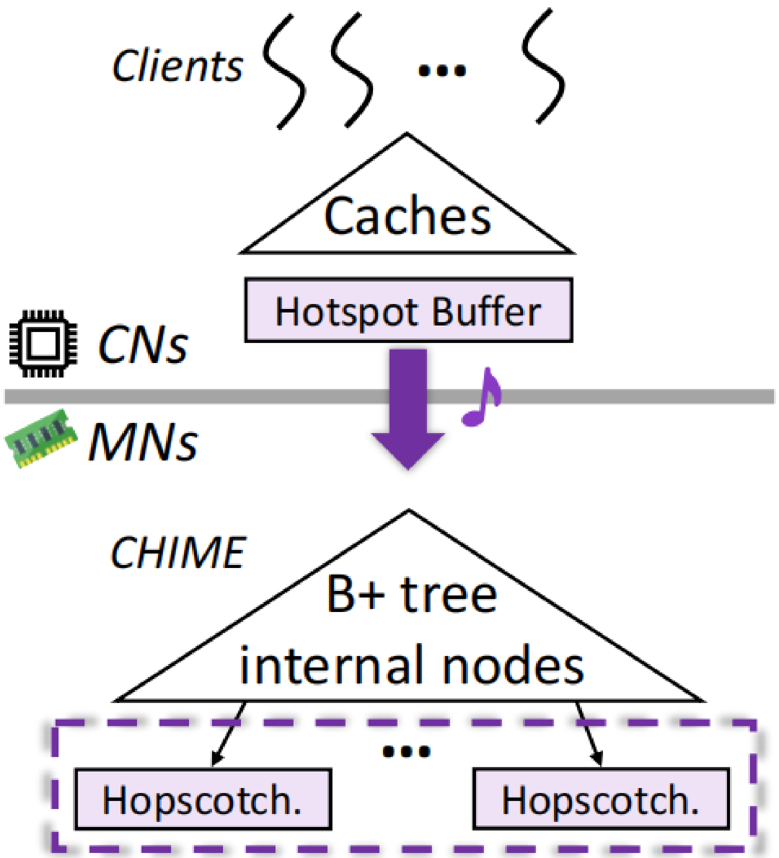
Problem: Still need to fetch all items within the neighborhood

→ ***Solution:*** *Speculatively read the target entry*





Optimization Summary



1. Complicated Optimistic Synchronization

Solution 1: *Three-Level Optimistic Synchronization*

2. Extra Metadata Accesses

Solution 2: *Access-Aggregated Metadata Management*

3. Read Amplifications of Hopscotch Hashing

Solution 3: *Hotness-Aware Speculative Read*

Evaluation

Workloads and Parameters

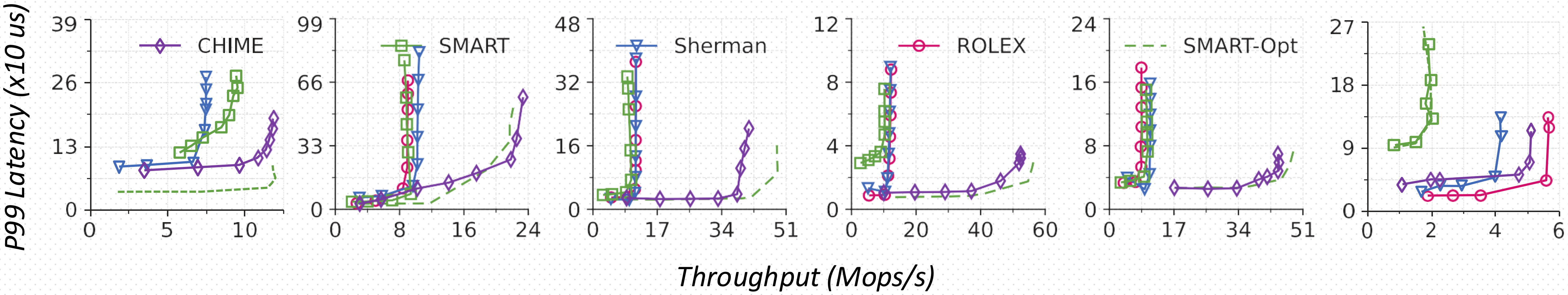
- YCSB workloads
- 8-byte keys and 8-byte values
- Limit the cache size to 100 MB per CN

Comparisons

- SMART[*OSDI' 23*]
  - The latest radix tree design on DM
- Sherman[*SIGMOD' 22*]
  - The classic B+ tree design on DM
- ROLEX[*FAST' 23*]
  - The latest learned index on DM
- SMART-Opt[*Optimal case*]
  - SMART with sufficient caches

Performance Comparison

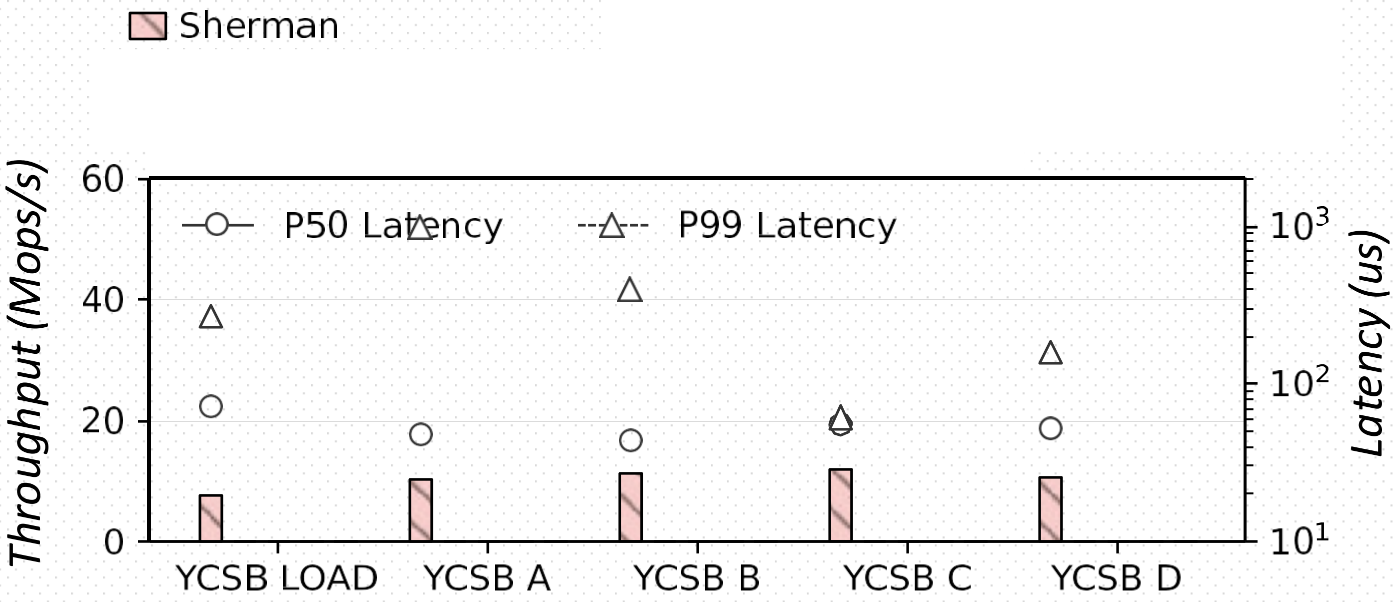
YCSB LOAD	YCSB A	YCSB B	YCSB C	YCSB D	YCSB E
100% insert	50% read 50% update	95% read 5% update	100% read	95% read 5% insert	95% scan 5% insert



- CHIME achieves:
  - Up to 4.3x higher throughput than Sherman and ROLEX
  - Up to 5.1x higher throughput than SMART
  - A close performance to the optimal case, with up to 8.7x lower cache consumption ***(57.6 MB vs. 503.6 MB)***

Factor Analysis

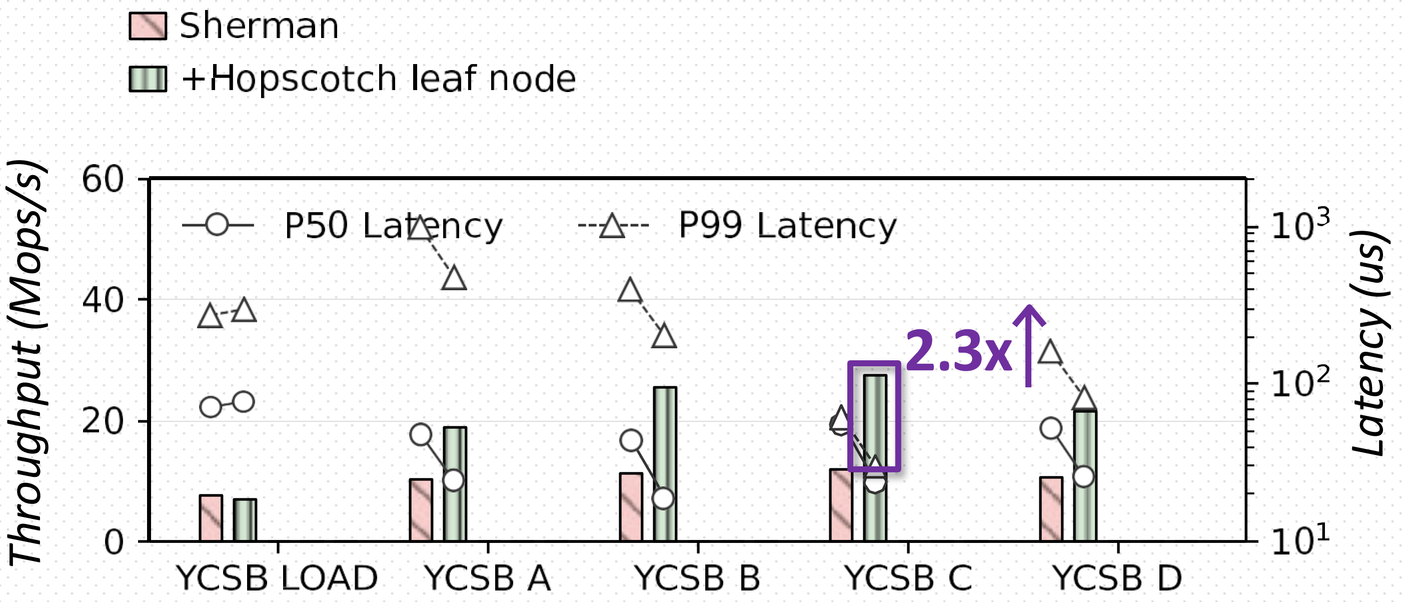
YCSB LOAD	YCSB A	YCSB B	YCSB C	YCSB D
100% insert	50% read 50% update	95% read 5% update	100% read	95% read 5% insert



- Start with Sherman and apply each proposed technique one by one

Factor Analysis

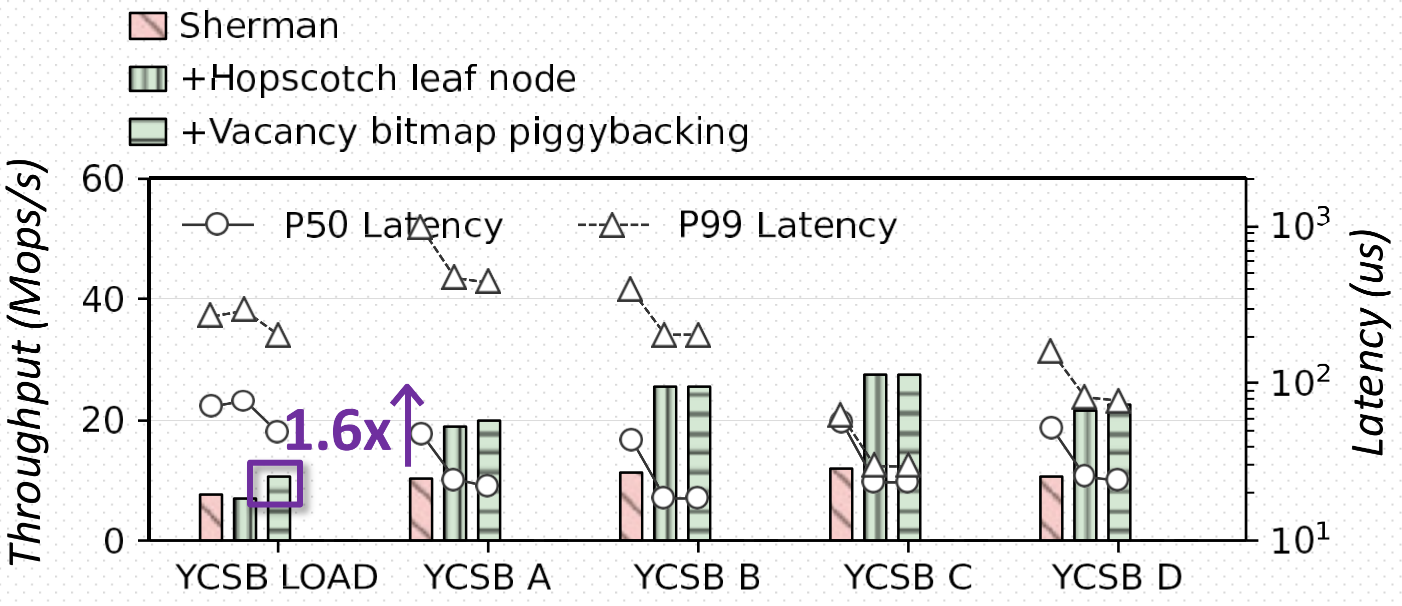
YCSB LOAD	YCSB A	YCSB B	YCSB C	YCSB D
100% insert	50% read 50% update	95% read 5% update	100% read	95% read 5% insert



- The *hopscotch leaf node* enables fetching the neighborhood rather than the entire leaf node

Factor Analysis

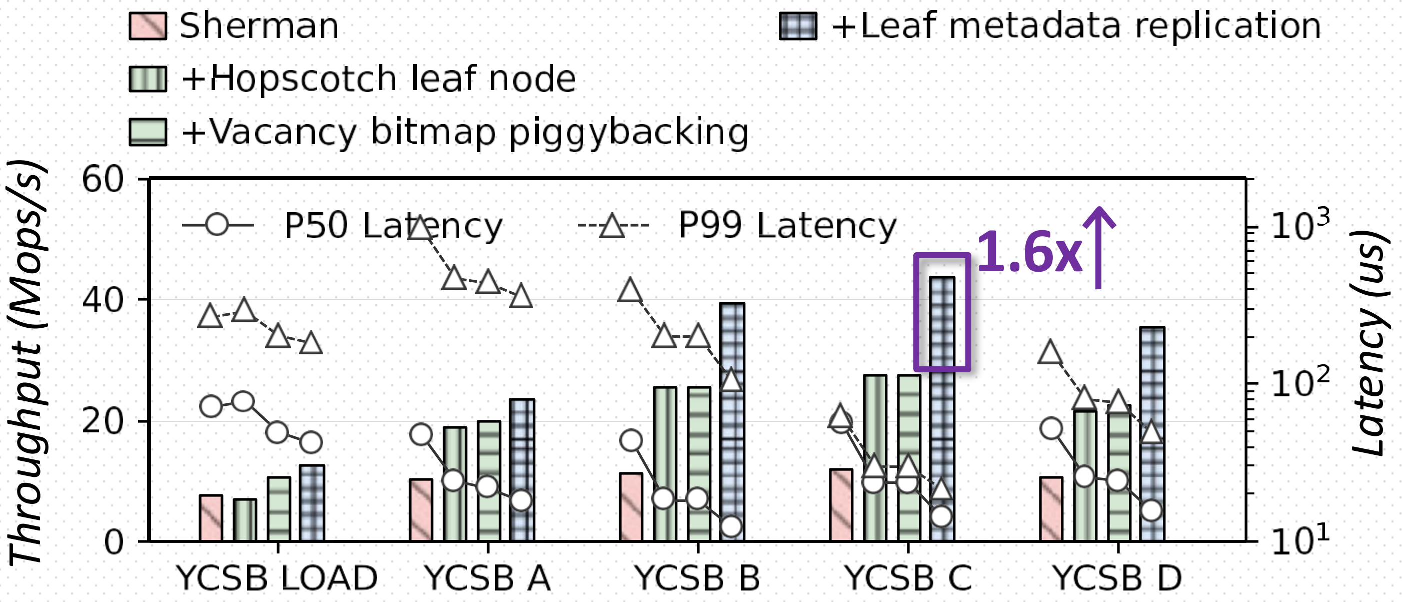
YCSB LOAD	YCSB A	YCSB B	YCSB C	YCSB D
100% insert	50% read 50% update	95% read 5% update	100% read	95% read 5% insert



- The *vacancy bitmap piggybacking* enables fetching the hop range rather than the entire leaf node, without inducing extra remote accesses

Factor Analysis

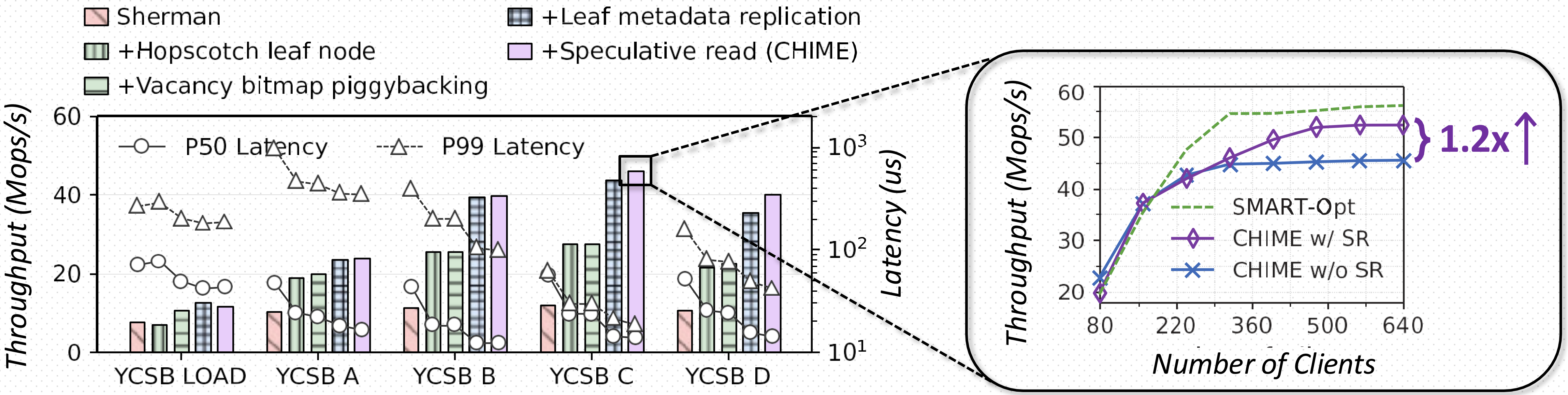
YCSB LOAD	YCSB A	YCSB B	YCSB C	YCSB D
100% insert	50% read 50% update	95% read 5% update	100% read	95% read 5% insert



- The *leaf metadata replication* avoids the extra remote accesses of fetching in-header leaf metadata

Factor Analysis

YCSB LOAD	YCSB A	YCSB B	YCSB C	YCSB D
100% insert	50% read 50% update	95% read 5% update	100% read	95% read 5% insert



- The *speculative read* enables greedily fetching the target entry rather than the entire neighborhood

Conclusion

- This paper identifies the **trade-off** between read amplifications and cache consumption for range indexes on DM

- We propose **CHIME**, a hybrid index combining the B+ tree with hopscotch hashing to break the trade-off:
  - Three-level optimistic synchronization
  - Access-aggregated metadata management
  - Hotness-aware speculative read

- CHIME outperforms the state-of-the-art range indexes on DM by up to **5.1x in throughput** with the same cache size and achieves similar performance with up to **8.7x lower cache consumption**

SOSP 2024

CHIME: A Cache-Efficient and High-Performance Hybrid Index on Disaggregated Memory

Xuchuan Luo, Jiacheng Shen, Pengfei Zuo, Xin Wang, Micheal R.Lyu, Yangfan Zhou

School of Computer Science, Fudan University
National Key Laboratory of Parallel and Distributed Computing, China
Duke Kunshan University Huawei Cloud The Chinese University of Hong Kong
Shanghai Key Laboratory of Intelligent Information Processing, Shanghai, China

Thanks you for your attention