

SOSP 2024



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

Xuchuan Luo<sup>1,2</sup>, Jiacheng Shen<sup>3</sup>, Pengfei Zuo<sup>4</sup>, Xin Wang<sup>1,6</sup>, Michael R. Lyu<sup>5</sup>, Yangfan Zhou<sup>1,2</sup>

**<sup>1</sup>School of Computer Science, Fudan University**

*<sup>2</sup>National Key Laboratory of Parallel and Distributed Computing, China*

*<sup>3</sup>Duke Kunshan University*

*<sup>4</sup>Huawei Cloud*

*<sup>5</sup>The Chinese University of Hong Kong*

*<sup>6</sup>Shanghai Key Laboratory of Intelligent Information Processing, Shanghai, China*



復旦大學  
FUDAN UNIVERSITY



昆山杜克大學  
DUKE KUNSHAN  
UNIVERSITY



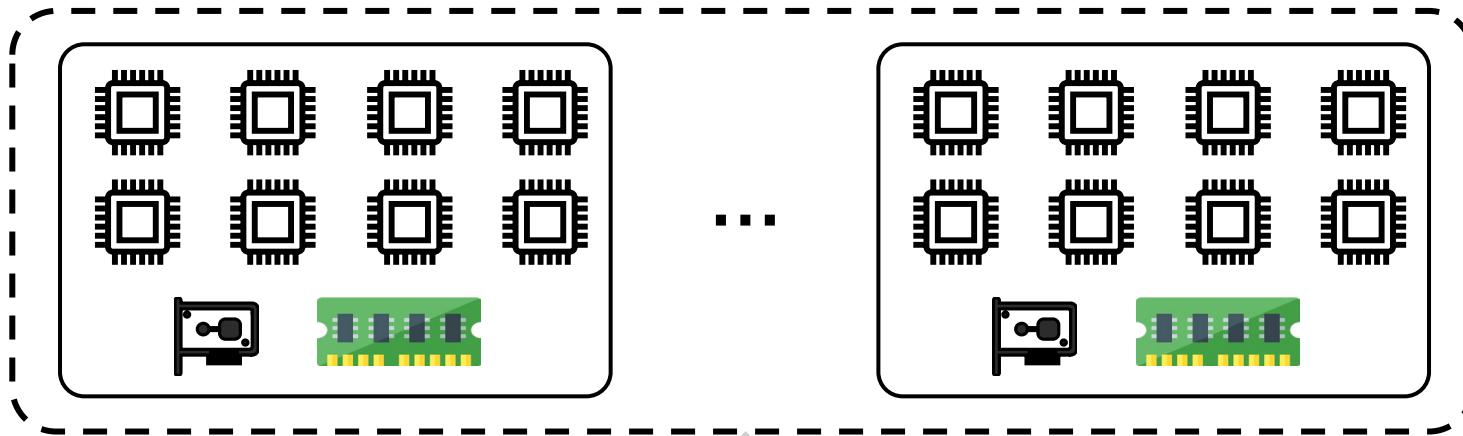
HUAWEI



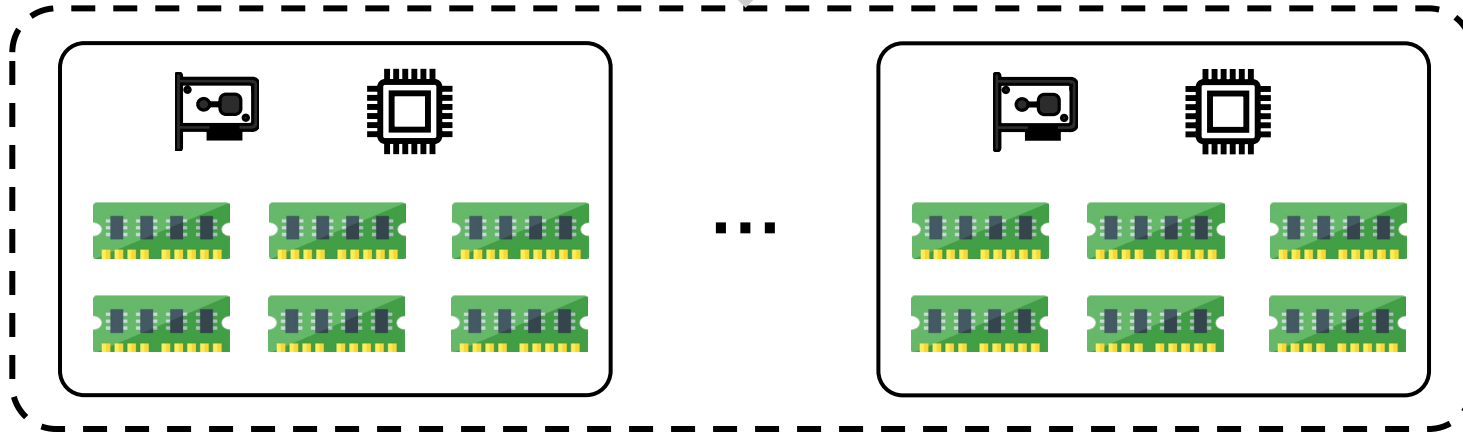
香港中文大學  
The Chinese University of Hong Kong

# Disaggregated Memory (DM)

Compute Nodes (CNs)



Fast Network (e.g., RDMA, CXL)



Memory Nodes (MNs)

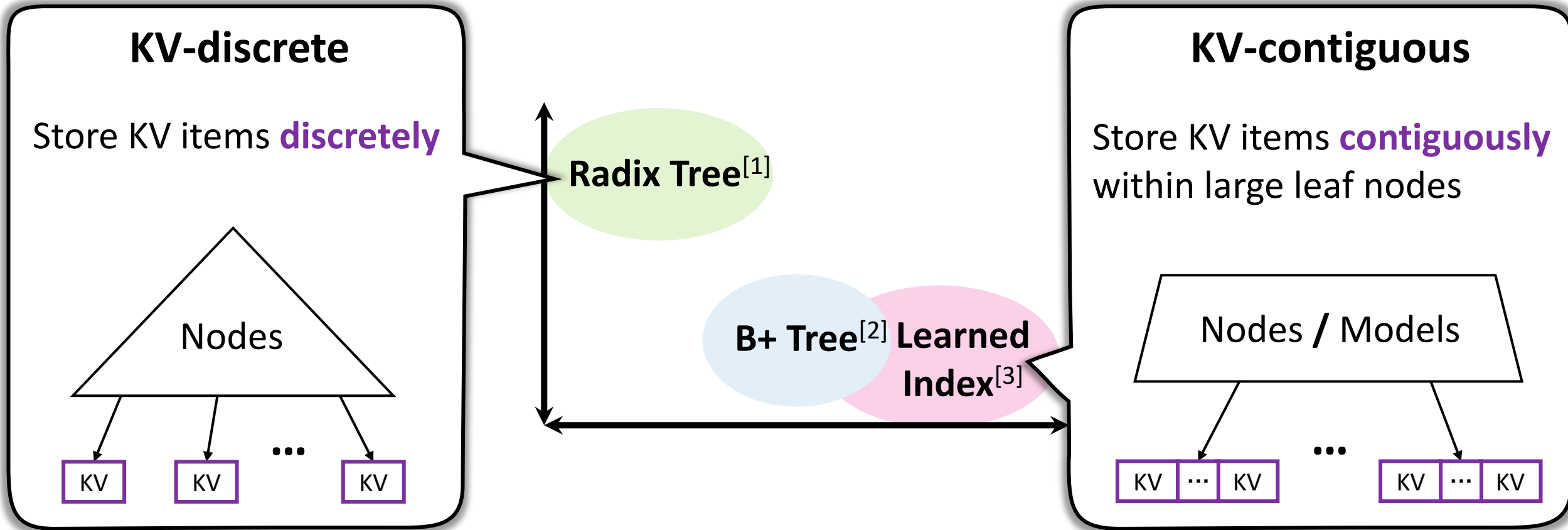


**Benefits:**

- ✓ Resource utilization
- ✓ Elasticity

# Range Indexes on Disaggregated Memory

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.

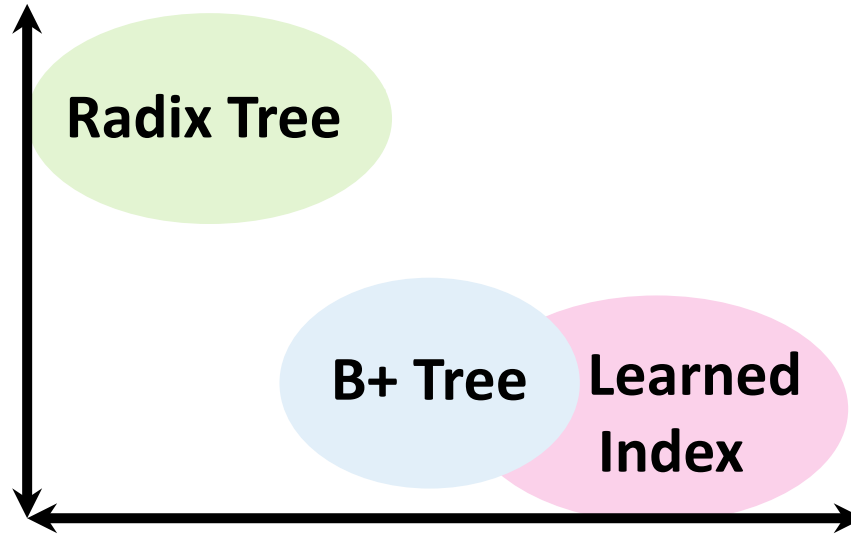
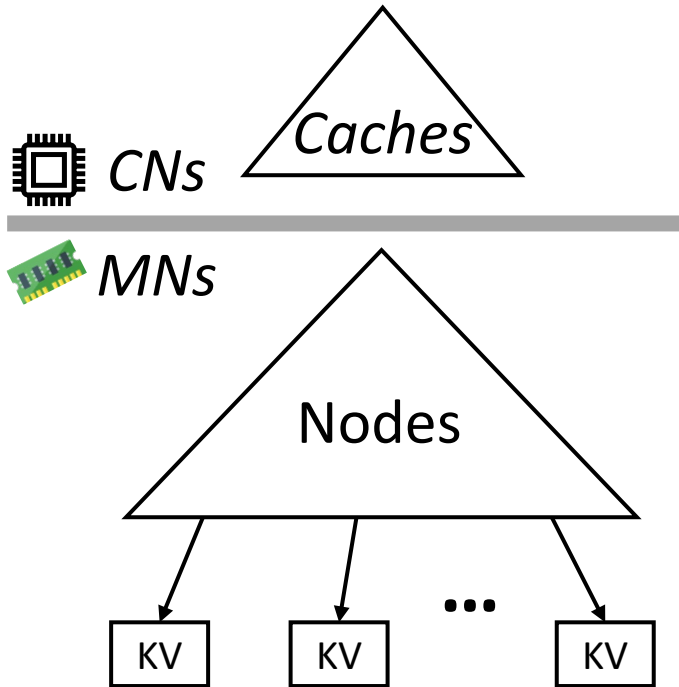
[2] Qing Wang et al. Sherman: A write-optimized distributed B+ tree index on disaggregated memory. SIGMOD 2022.

[3] Pengfei Li et al. ROLEX: A scalable RDMA-oriented learned key-value store for disaggregated memory. FAST 2023.

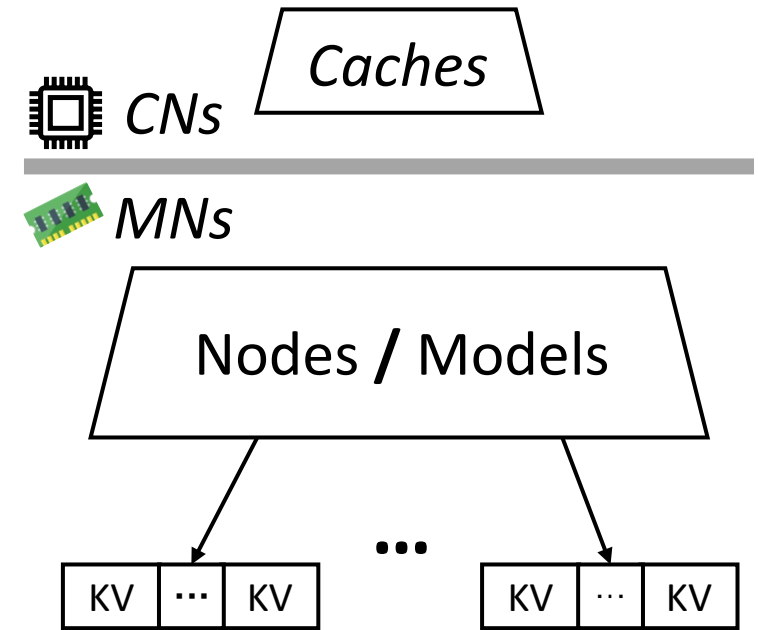
# Range Indexes on Disaggregated Memory

Computing-side caches are adopted to reduce access latency:

## KV-discrete

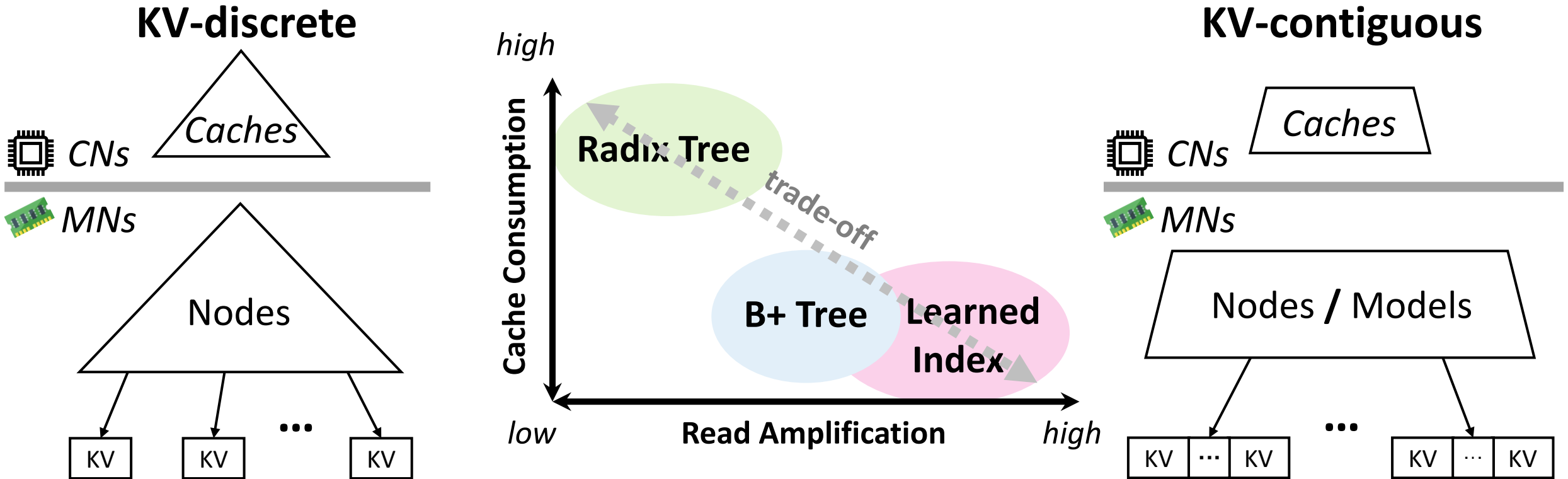


## KV-contiguous



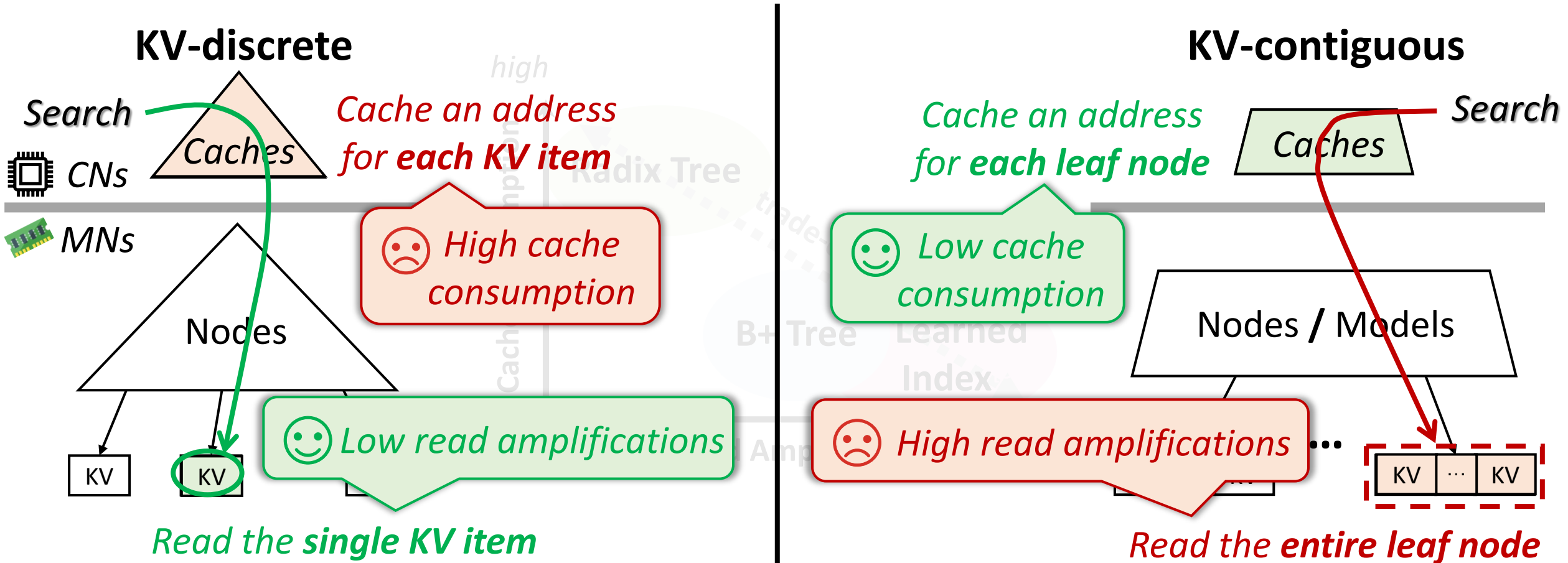
# Motivation: A Trade-off for Range indexes on DM

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



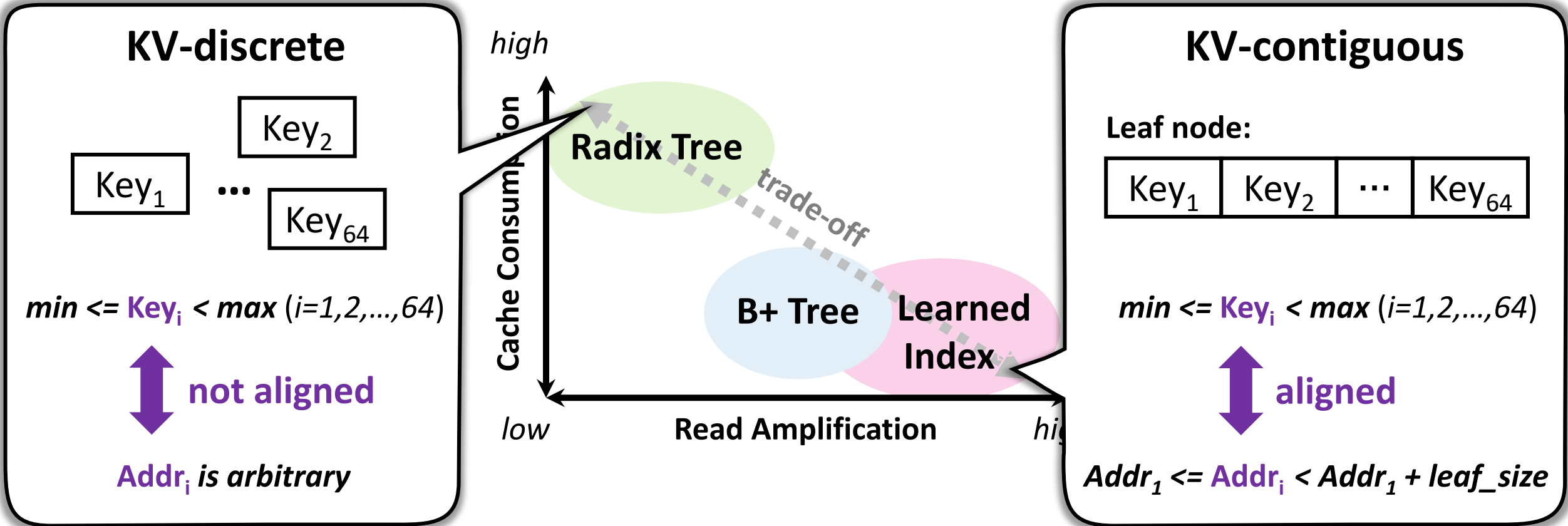
# Motivation: A Trade-off for Range indexes on DM

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



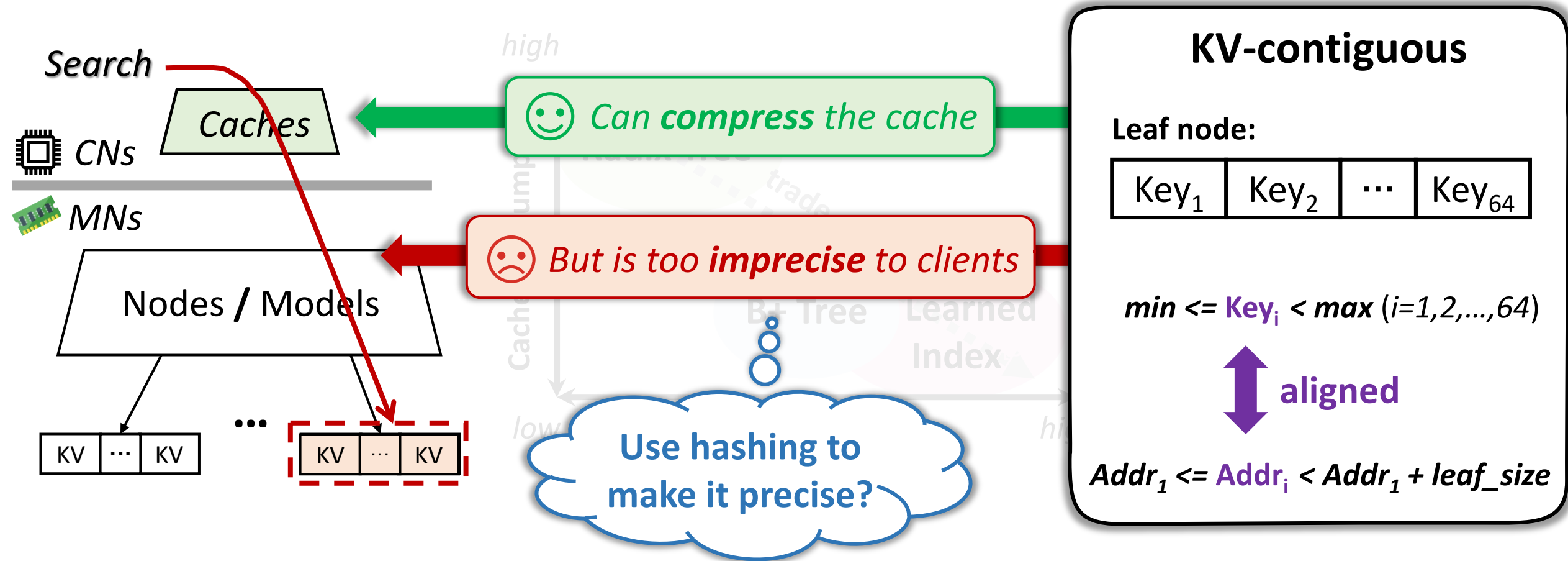
# Motivation: A Trade-off for Range indexes on DM

Root Cause: The **alignment** between keys and memory addresses



# Motivation: A Trade-off for Range indexes on DM

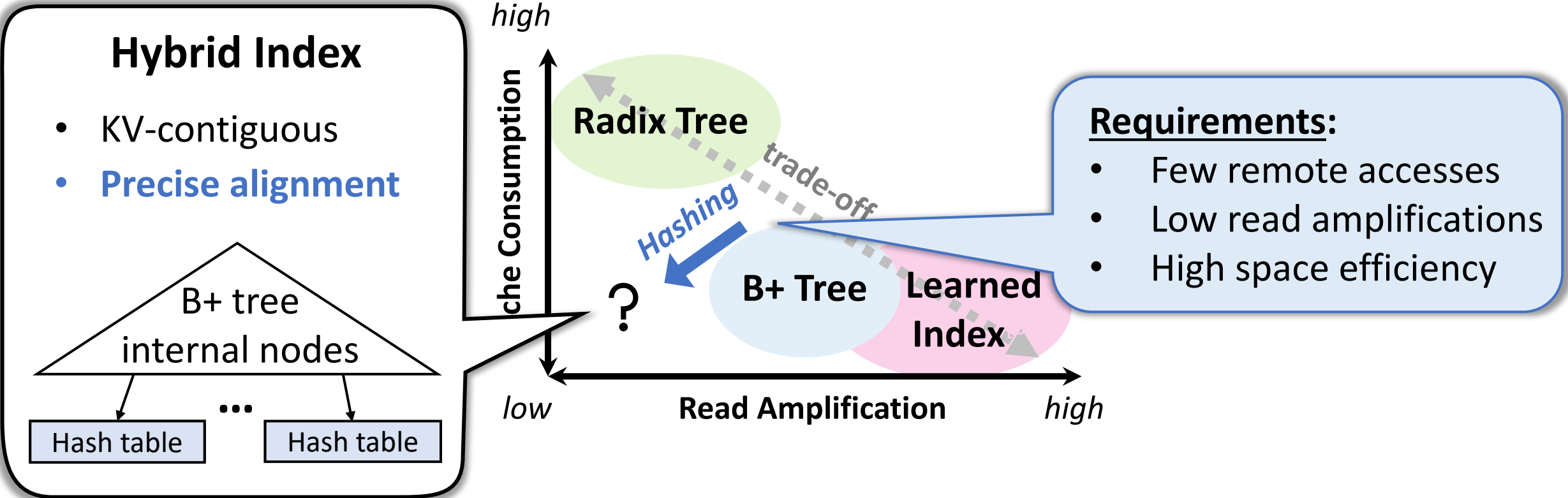
Root Cause: The **alignment** between keys and memory addresses





# Straightforward Idea

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



# Widely Choose a Suitable Hashing Scheme

## Remote accesses

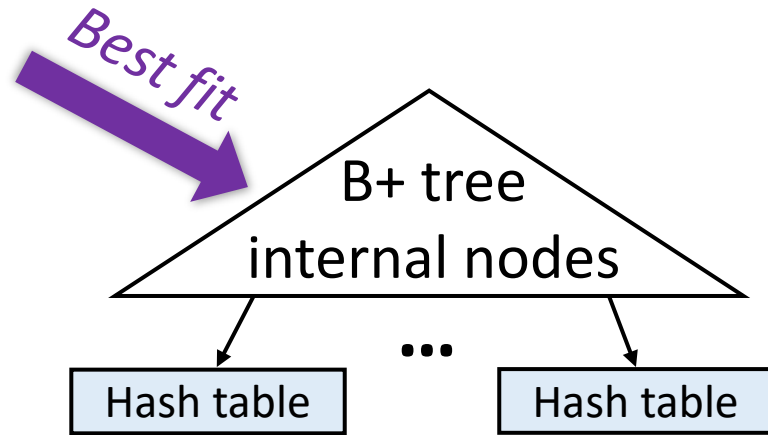
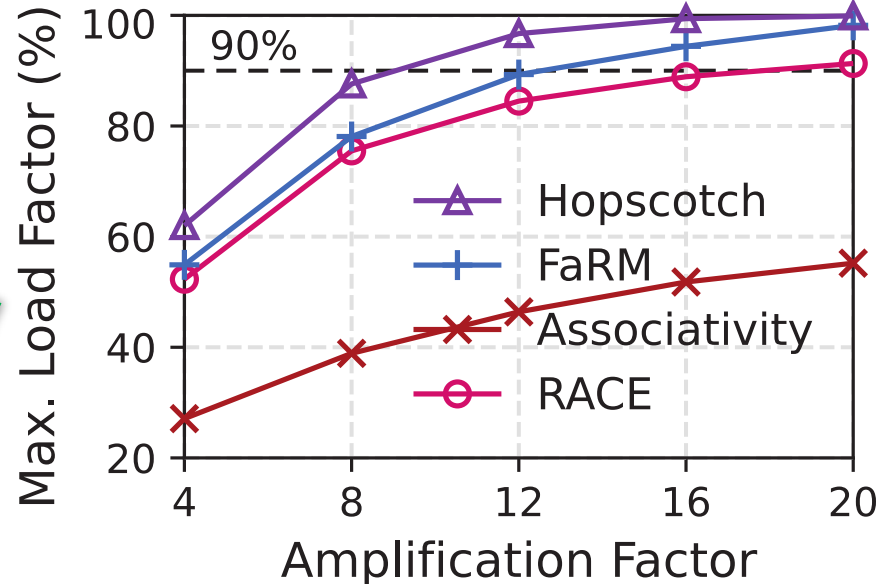
Many accesses: ✗

- Cuckoo hashing
- Clustering hashing
- ...

Few accesses: ✓

- Simple associativity
- Hopscotch hashing<sup>[1]</sup>
- FaRM<sup>[2]</sup>
- RACE<sup>[3]</sup>

## 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.

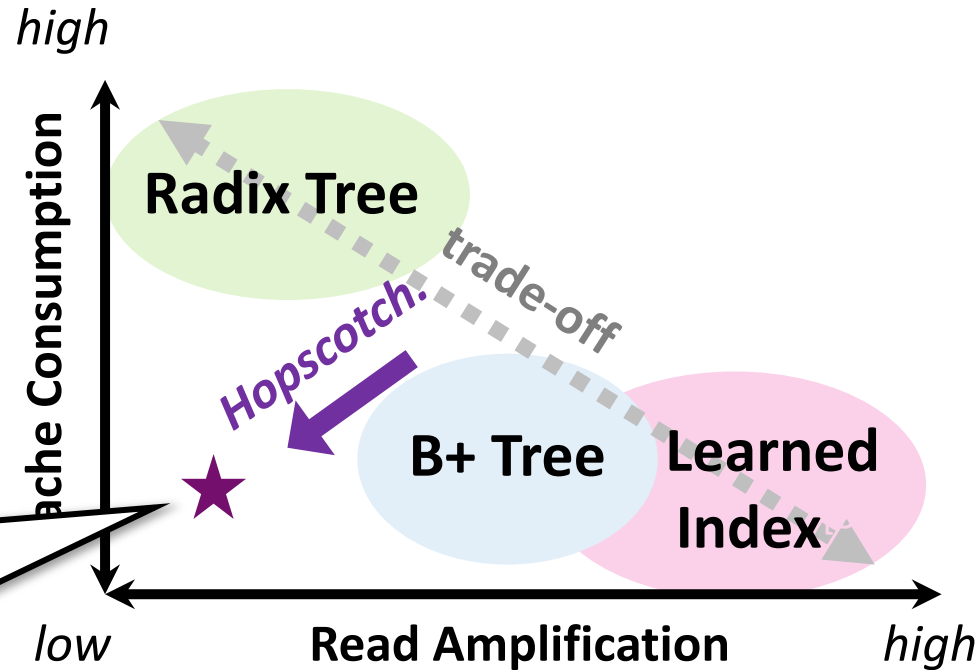
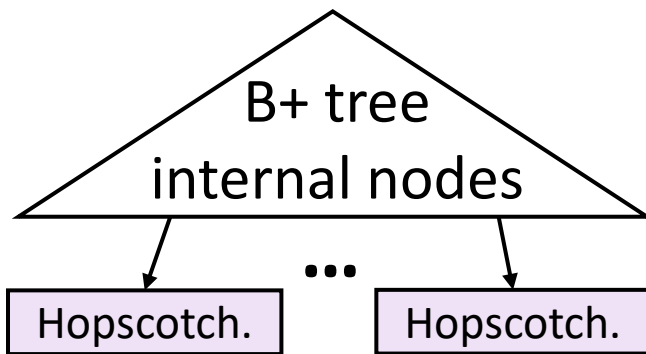


# Viable Idea

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

## Hybrid Index

- KV-contiguous
- **Precise alignment**
- **High load factor**

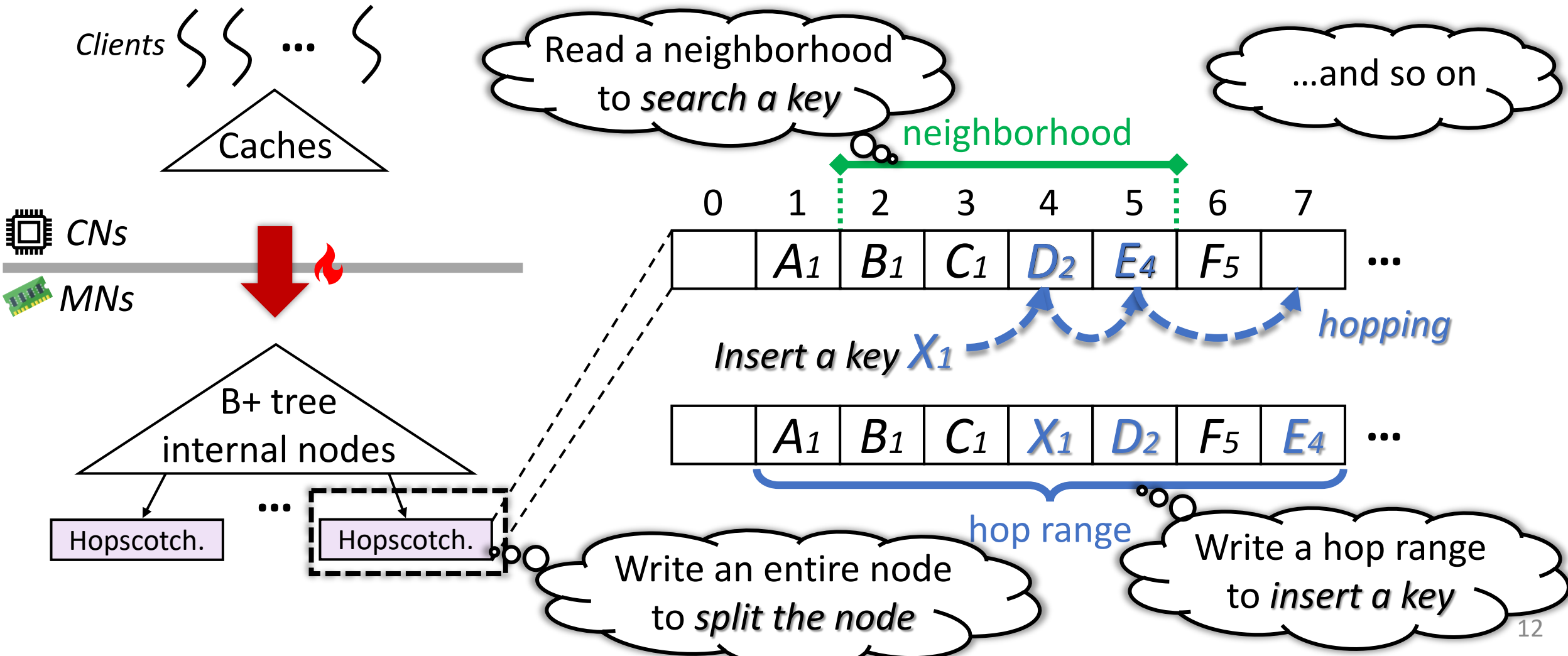


## 😊 Benefit:

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

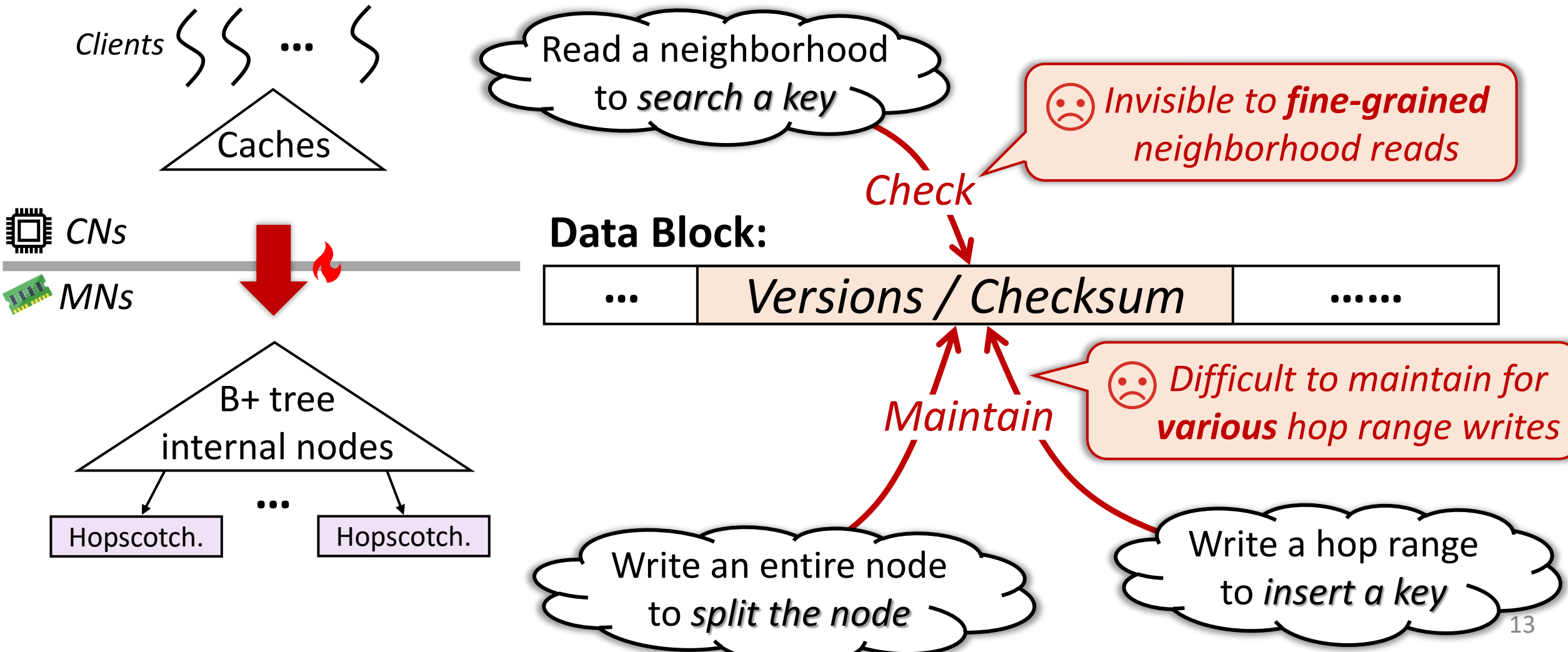
# Challenge 1: Complicated Optimistic Synchronization

Various granularities in reads/writes complicate the optimistic synchronization



# Challenge 1: Complicated Optimistic Synchronization

Various granularities in reads/writes complicate the optimistic synchronization

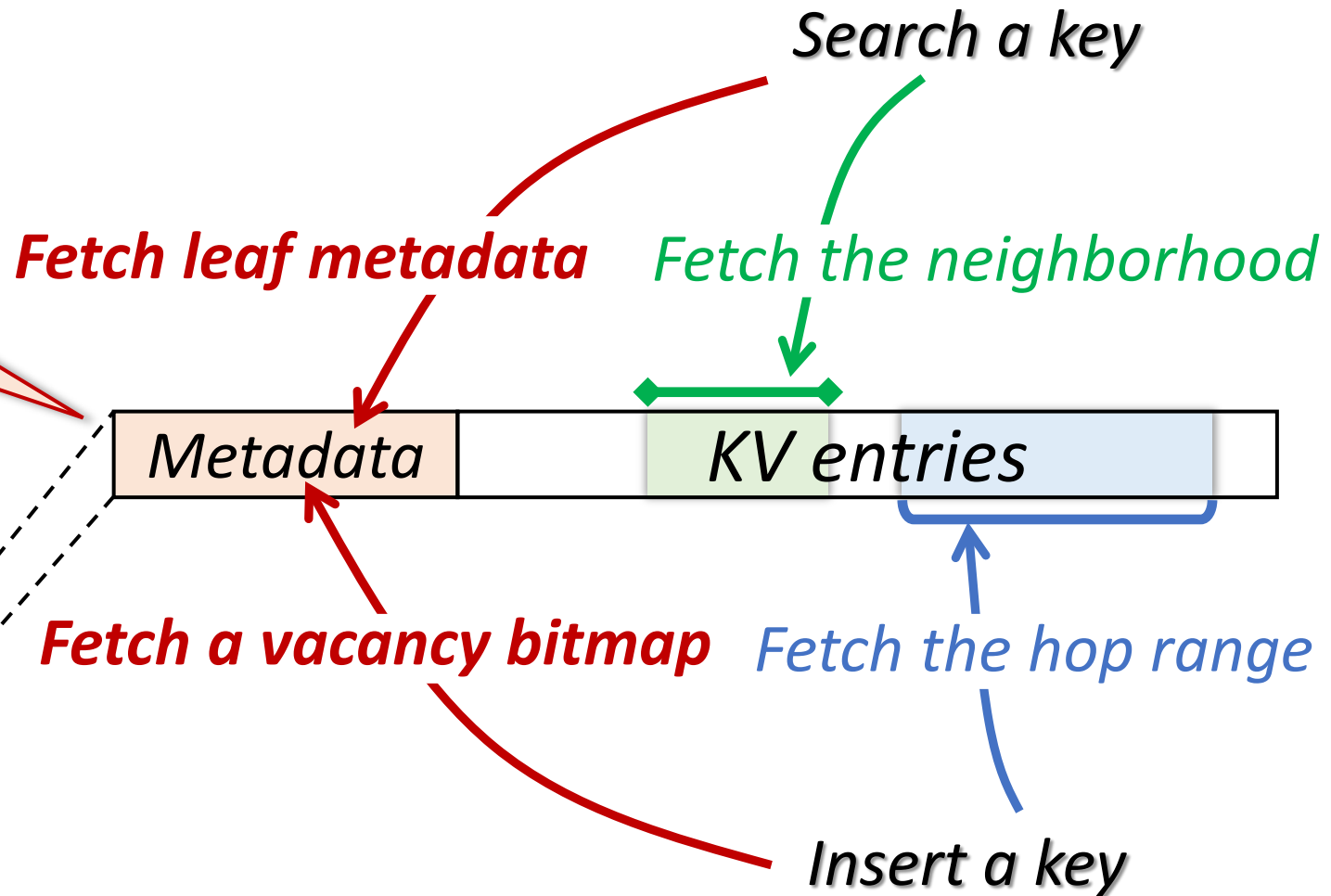
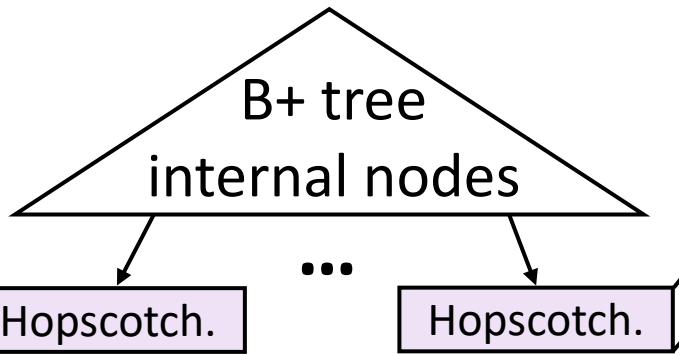


# Challenge 2: Extra Metadata Accesses

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

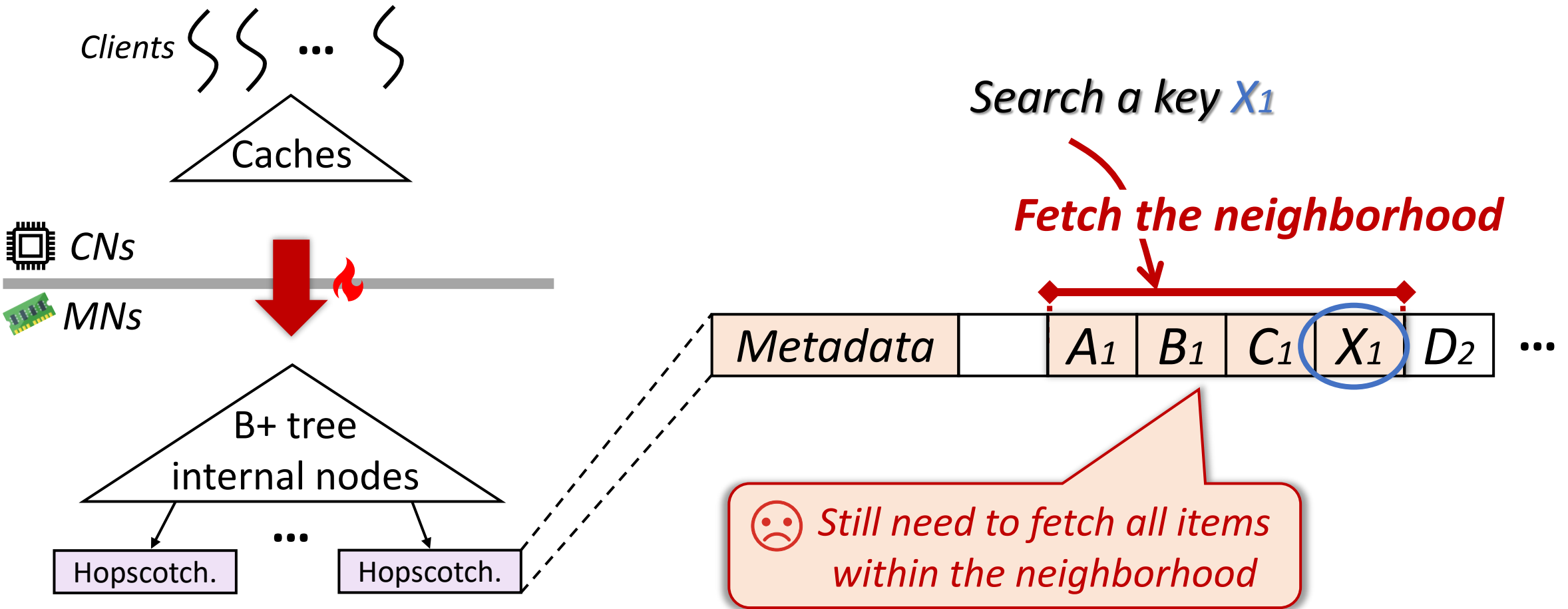
Clients { { ... }

☹️ *Extra remote memory accesses to fetch metadata*

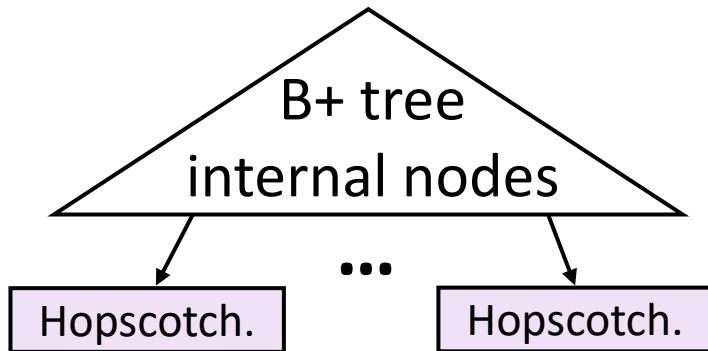
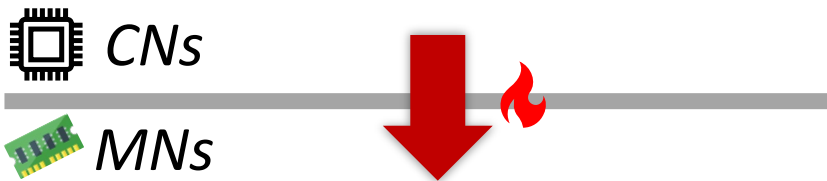
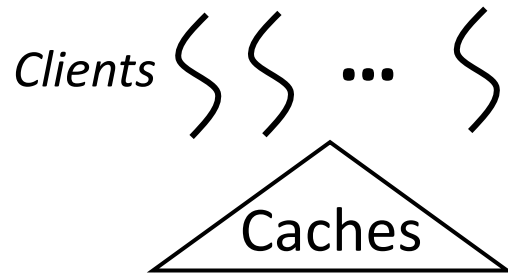


# Challenge 3: Read Amplifications of Hopscotch Hashing

Hopscotch hashing still incurs read amplifications compared with reading a KV



# Challenge Summary



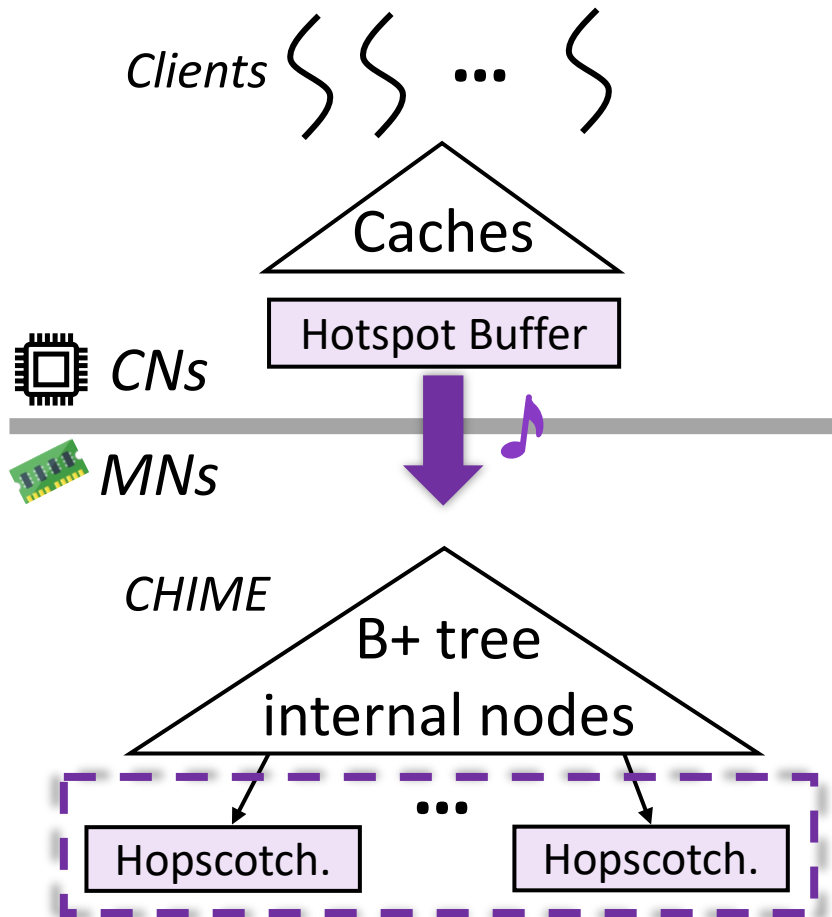
1. Complicated Optimistic Synchronization

2. Extra Metadata Accesses

3. Read Amplifications of Hopscotch Hashing



# The CHIME Design



1. Complicated Optimistic Synchronization

*Solution 1: Three-Level Optimistic Synchronization*

2. Extra Metadata Accesses

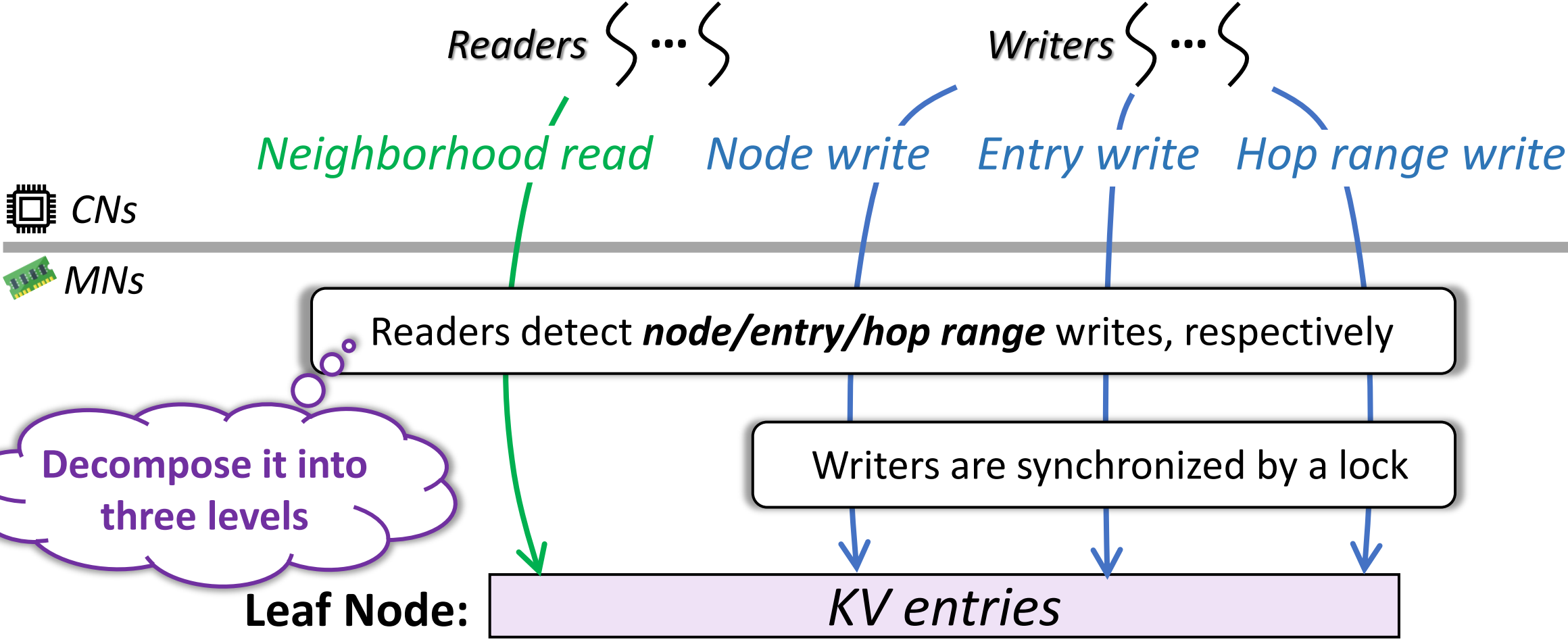
*Solution 2: Access-Aggregated Metadata Management*

3. Read Amplifications of Hopscotch Hashing

*Solution 3: Hotness-Aware Speculative Read*

# Three-level Optimistic Synchronization

## Synchronization Overview



# Three-level Optimistic Synchronization

**Level 1: Detect the *node write***

**Problem:** Readers cannot perceive the node write

*Search a key*

*Reading a neighborhood*

**Node:**

*KV entries*



*Writing the entire node*

*Split the node*

# Three-level Optimistic Synchronization

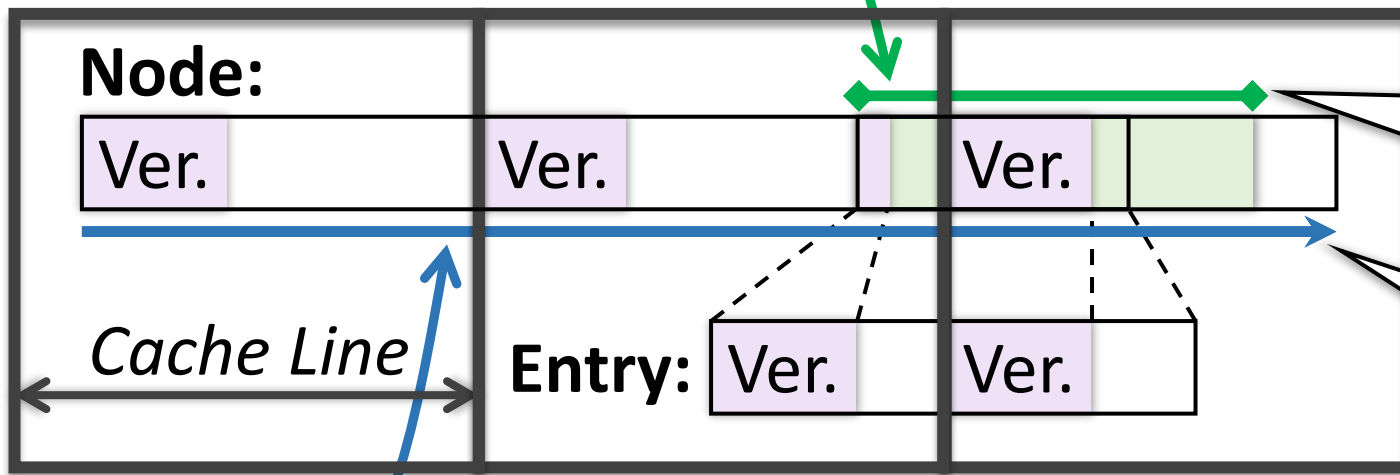
## Level 1: Detect the *node write*

**Problem:** Readers cannot perceive the node write

→ **Solution:** Use cache line versioning<sup>[1]</sup>

Search a key

Reading a neighborhood



Readers:

- Check only **fetch**ed versions
- Retry if they cannot match

Writers:

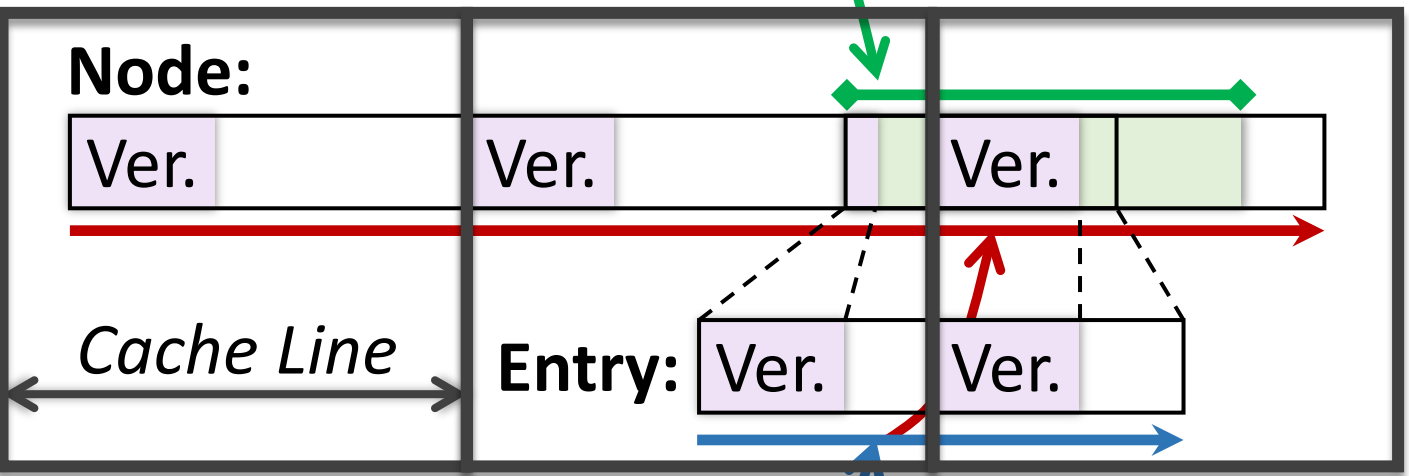
- Increment **all** versions in the node

[1] Aleksandar Dragojevic et al. FaRM: Fast Remote Memory. NSDI 2014.

# Three-level Optimistic Synchronization

**Level 2: Detect the *entry write*** Problem: Writers have to update all versions in the node

*Search a key*  
*Reading a neighborhood*



*Writing an entry*  
*Update a key*

# Three-level Optimistic Synchronization

## Level 2: Detect the *entry write*

**Problem:** Writers have to update all versions in the node

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

Search a key

Reading a neighborhood

**Node:**

Ver.

Ver.

Ver.

Cache Line

**Entry:**

Ver.

Ver.

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**

Writing an entry

Update a key

4 bit

4 bit

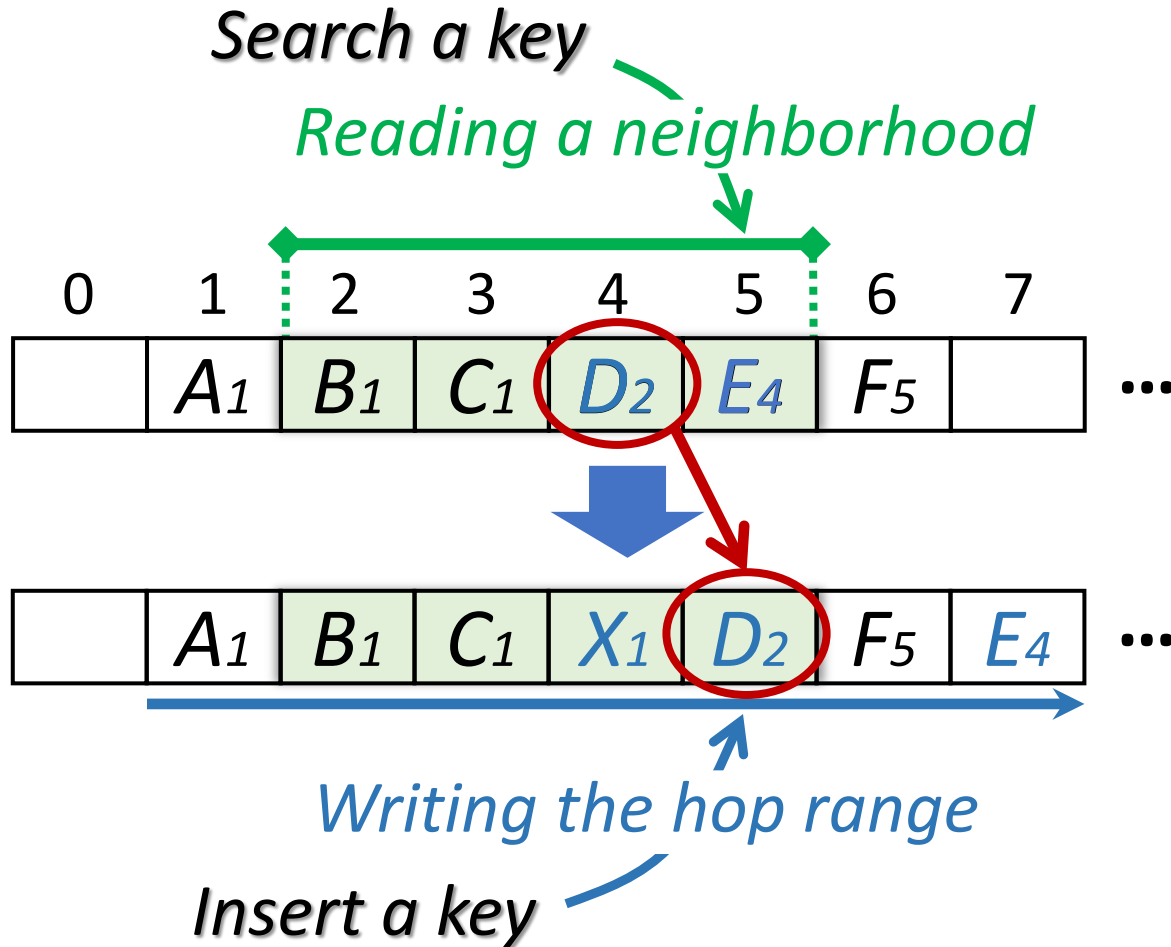
NV

EV

# Three-level Optimistic Synchronization

## Level 3: Detect the *hop range* write

Problem: Location changes of hopped items

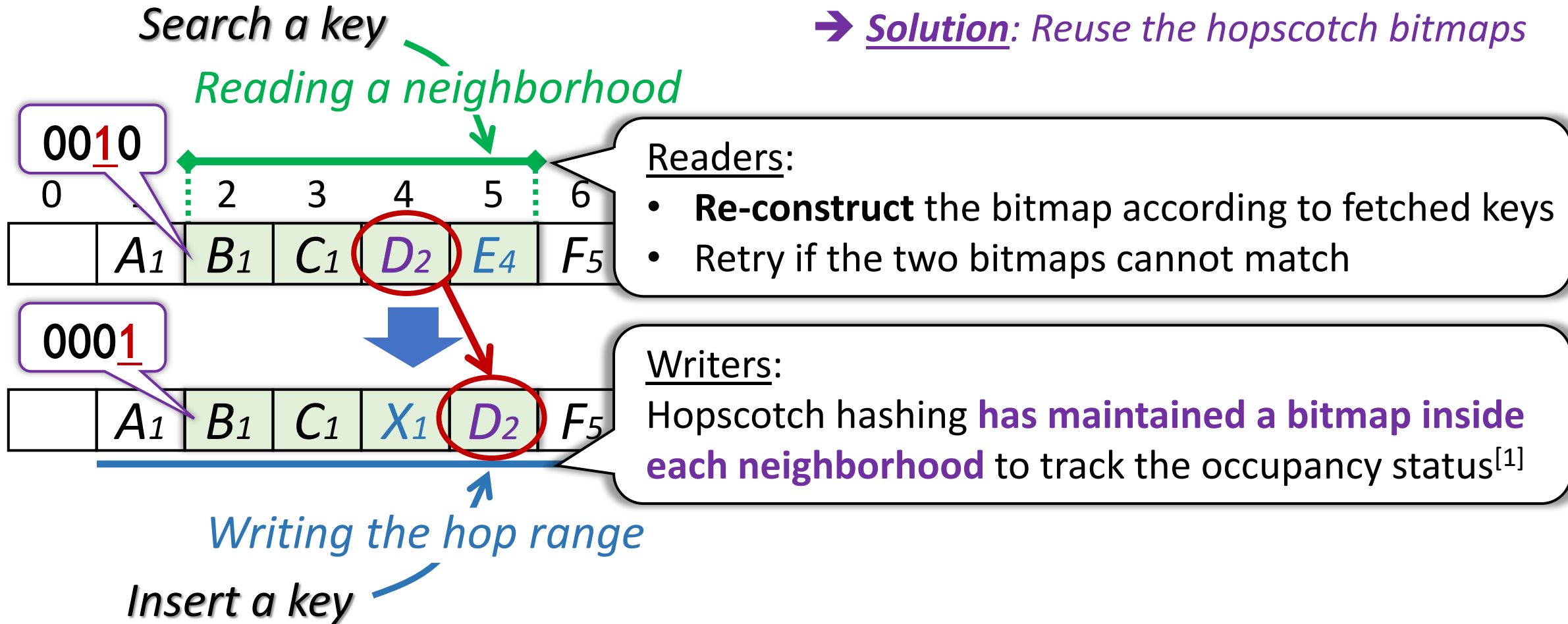


# Three-level Optimistic Synchronization

## Level 3: Detect the *hop range write*

**Problem:** Location changes of hopped items

→ **Solution:** Reuse the hopscotch bitmaps



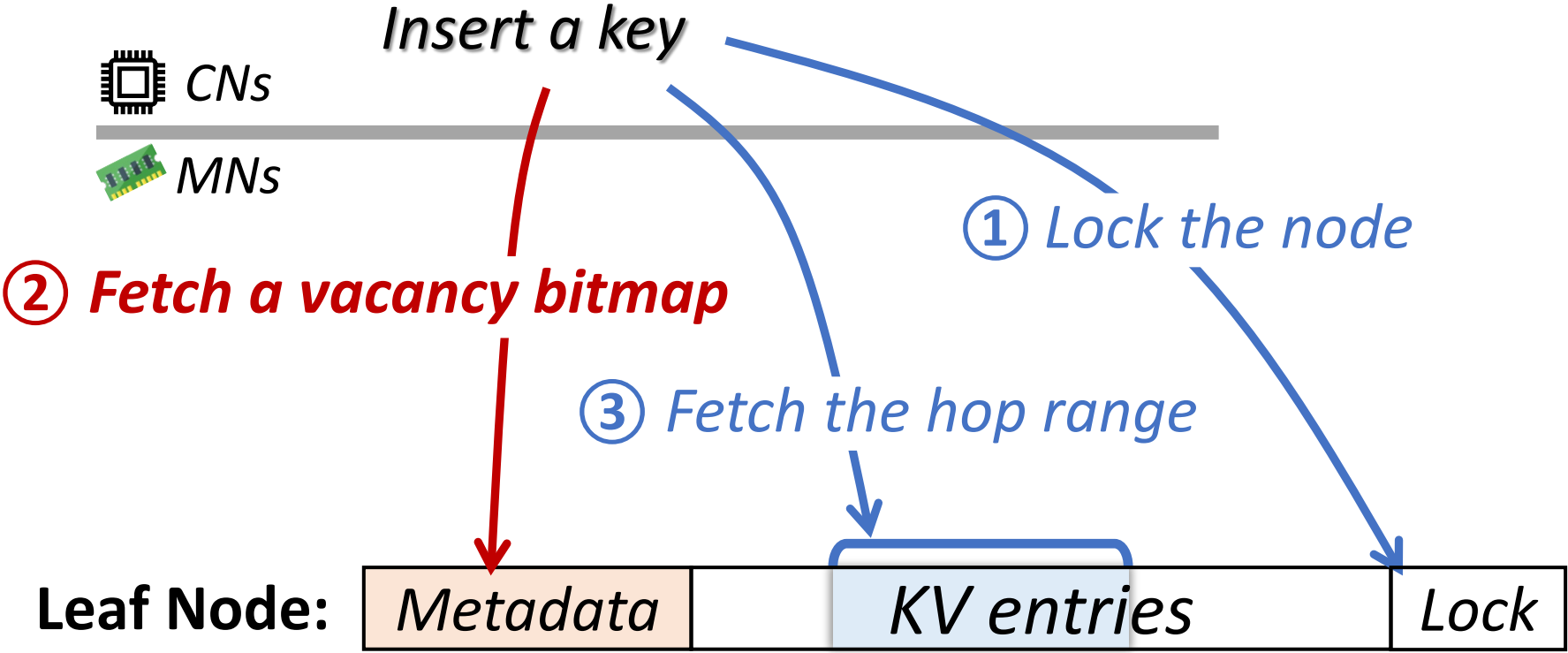
[1] Maurice Herlihy et al. Hopscotch hashing. DISC 2008.



# Access-Aggregated Metadata Management

## Metadata for hopscotch hashing

**Problem:** Vacancy bitmaps induce extra accesses



# Access-Aggregated Metadata Management

## Metadata for hopscotch hashing

**Problem:** Vacancy bitmaps induce extra accesses

→ **Solution:** Piggyback the vacancy bitmap



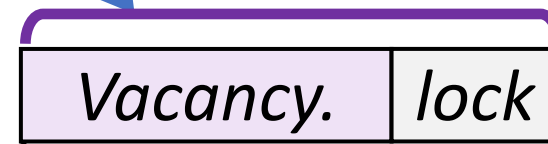
Insert a key

Achieve this via **masked-CAS** [1]:

- **Mask out** the vacancy bitmap during the *compare*
- **Remove the mask** during the *swap*

① Lock the node + get the bitmap

② Fetch the hop range



1 bit

Leaf Node:

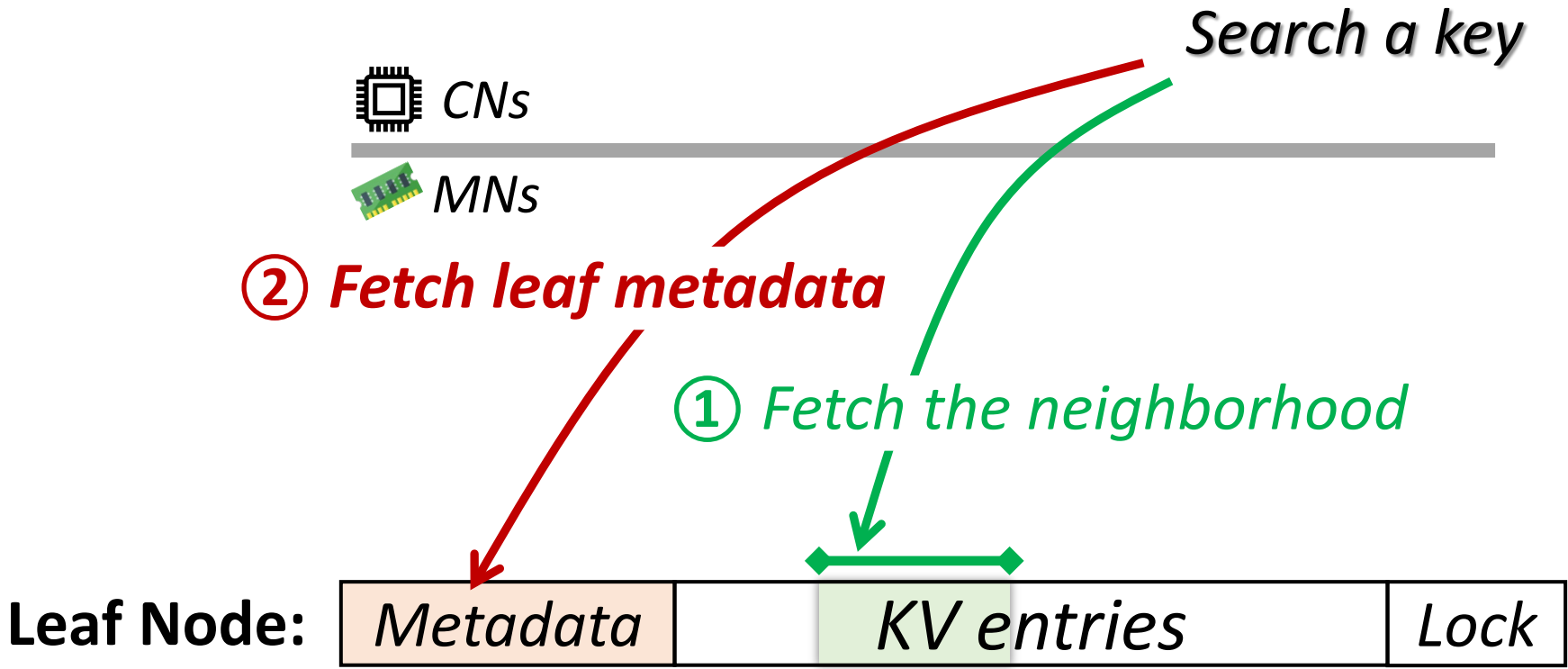


[1] NVIDIA Corporation. RDMA Aware Networks Programming User Manual v1.7.

# Access-Aggregated Metadata Management

## Metadata for the B+ tree

**Problem:** Leaf metadata induce extra accesses

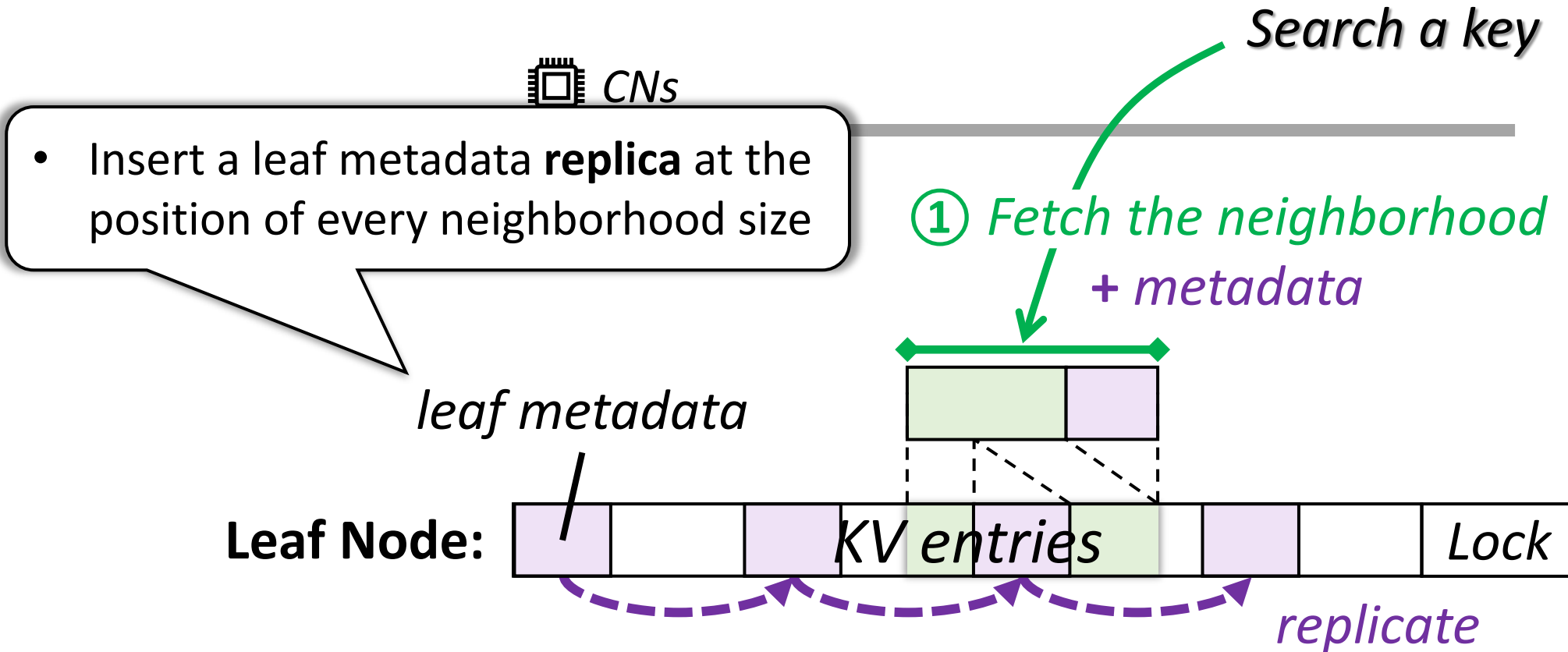


# Access-Aggregated Metadata Management

## Metadata for the B+ tree

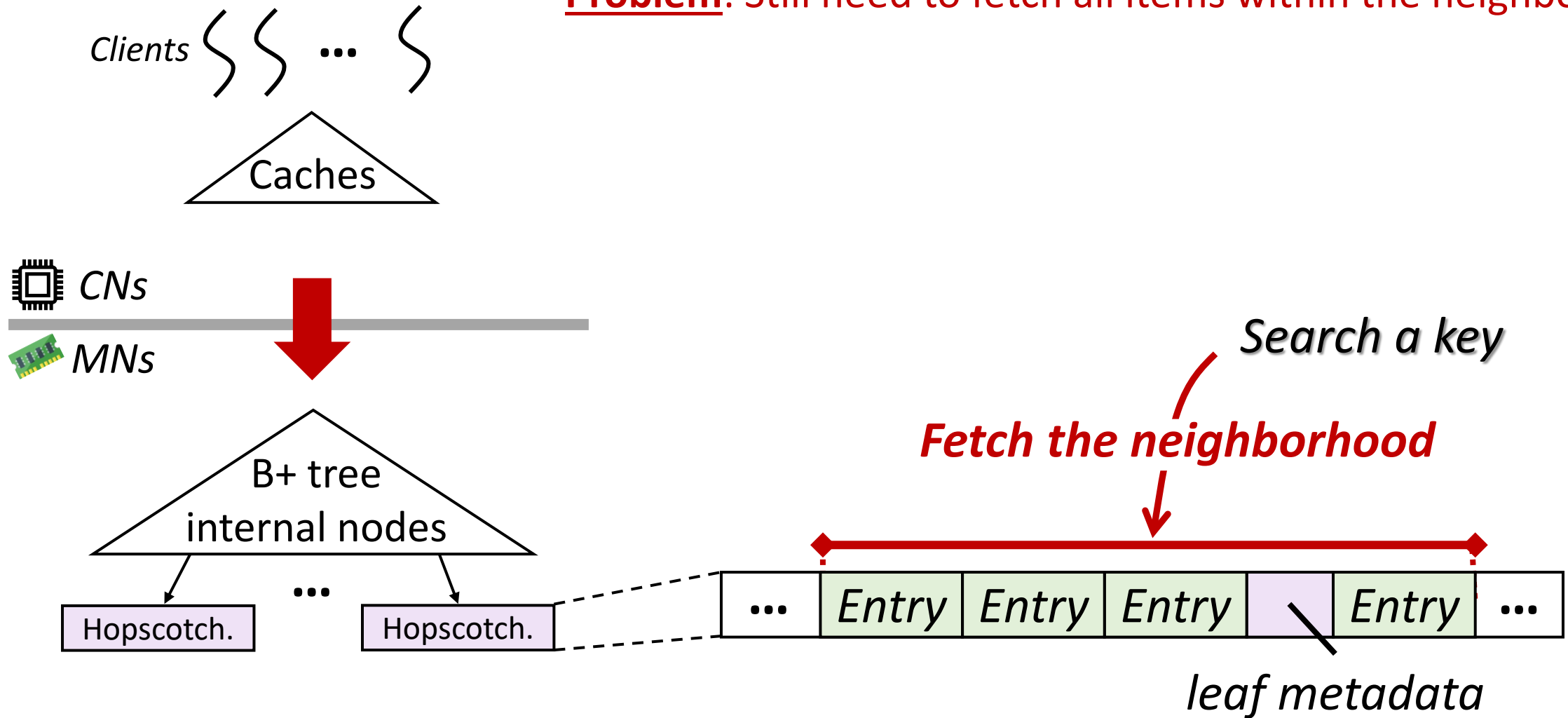
**Problem:** Leaf metadata induce extra accesses

→ **Solution:** Replicate the leaf metadata

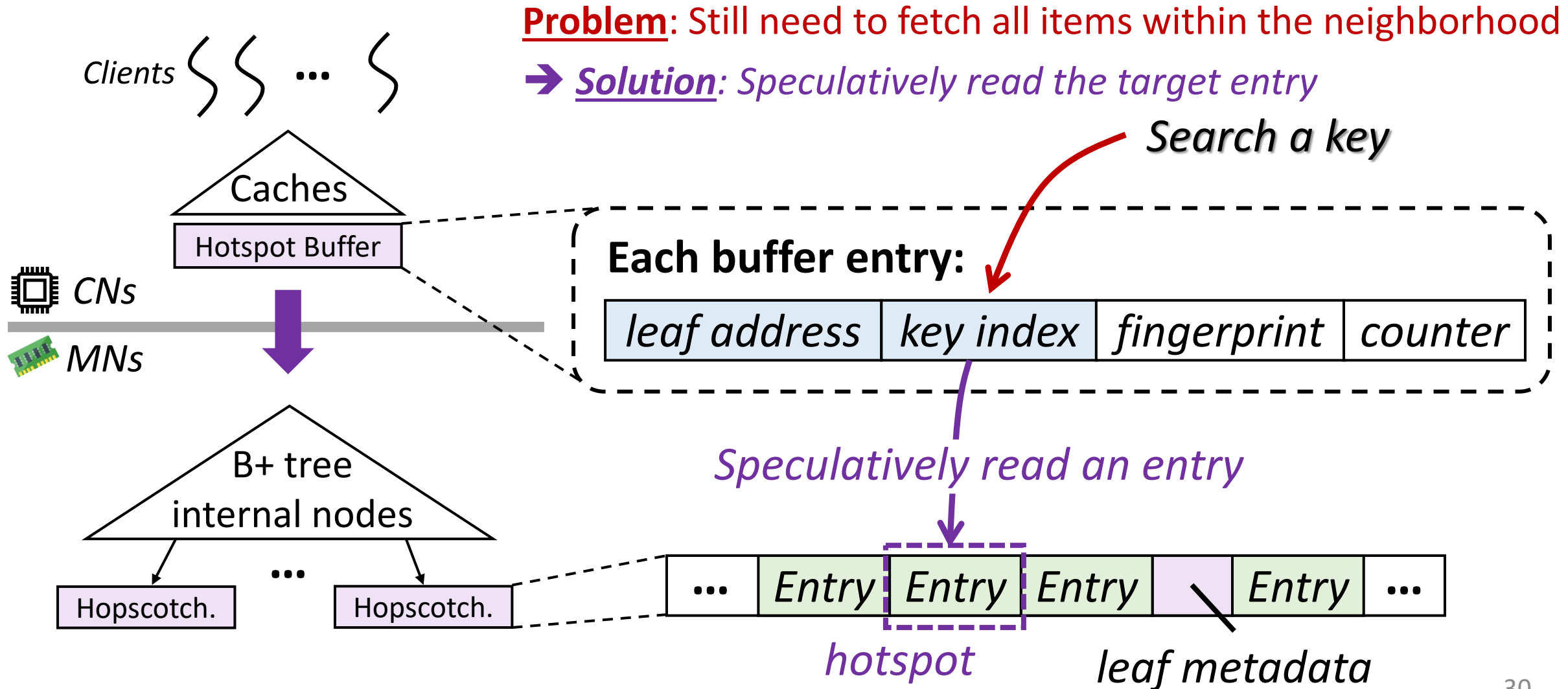


# Hotness-Aware Speculative Read

**Problem:** Still need to fetch all items within the neighborhood

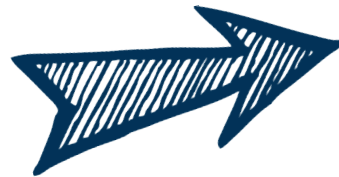


# Hotness-Aware Speculative Read



# More Details

- ✓ Sibling-based validation
- ✓ Support for variable-sized keys and values
- ✓ Applicability to the learned Index
- ✓ Detailed operations
- ✓ .....



**Figure 6.** The node structures of CHIME. "NV" and "EV" are the node-level and entry-level versions, respectively. For brevity, the cache line versions are omitted. The op...

**Figure 7.** Two-level cache line versions for detecting node...

**Figure 5.** The hopscotch bitmap inside...

**Figure 1.** The trade-off between cache consumption and read amplifications for existing DM range indexes [32, 36, 56].

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

Xuchuan Luo<sup>1,2</sup>, Jiacheng Shen<sup>1</sup>, Pengfei Zuo<sup>3</sup>, Xin Wang<sup>4,5</sup>, Michael R. Lyu<sup>6</sup>, Yangfan Zhou<sup>1,2</sup>

<sup>1</sup>School of Computer Science, Fudan University  
<sup>2</sup>National Key Laboratory of Parallel and Distributed Computing, China  
<sup>3</sup>Duke Kunshan University  
<sup>4</sup>Huawei Cloud  
<sup>5</sup>The Chinese University of Hong Kong  
<sup>6</sup>Shanghai Key Laboratory of Intelligent Information Processing, Shanghai, China

**Abstract**  
Disaggregated memory (DM) is a widely discussed data-center architecture in academia and industry. It decouples computing and memory resources from monolithic servers into two network-connected resource pools. Range indexes are widely adopted by storage systems on DM to efficiently locate and query remote data. However, existing range indexes on DM suffer from either high computing-side cache consumption or high memory-side read amplifications. In this paper, we propose CHIME, a hybrid index combining B-trees with hopscotch hashing, to achieve low cache consumption and low read amplifications simultaneously. There are three challenges in constructing CHIME on DM, i.e., the access and the read amplifications introduced by hopscotch hashing. CHIME leverages 1) a three-level optimistic synchronization scheme to synchronize read and write operations with various granularities, 2) an access-aggregated metadata management technique to eliminate extra metadata accesses by piggybacking and replicating metadata, and 3) an effective read amplification reduction mechanism to mitigate the read amplifications of hopscotch hashing. Experimental results show that CHIME outperforms the state-of-the-art range indexes on DM by up to 5.1x with the same cache size and achieves similar performance with up to 8.7x lower cache consumption.

**CCS Concepts:** Information systems → Distributed storage; Data structures

**Keywords:** Disaggregated Memory, RDMA, Hybrid Index, B+ Tree, Hopscotch Hashing

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org. SCSP '24, November 4–6, 2024, Austin, TX, USA. ACM ISBN 978-0-400-1231-7/24/11 https://doi.org/10.1145/3694715.3695959

**1 Introduction**  
The disaggregated memory (DM) architecture is being widely discussed due to its potential to improve datacenter resource utilization [2, 21, 28, 43, 48, 56]. It decouples computing and memory resources into independent computing and memory pools and interconnects them with fast networks, e.g., InfiniBand [6] and compute express link (CXL) [11]. Range indexes [32, 36, 56] are cornerstones for storage systems on DM, e.g., databases and computing-side range queries. For a practical range index on DM, computing-side cache consumption [34, 56] and memory-side read and write amplifications [36, 56] are two critical aspects. First, read and write amplifications waste network bandwidth between computing and memory pools. As network bandwidth between computing and memory pools is limited, existing approaches cache part of the index structure and addresses of KV items in the computing pool to reduce the overhead of remote index traversals [5, 32, 36, 56]. Range indexes should also reduce cache consumption due to the limited computing-side memory. Unfortunately, existing approaches cannot simultaneously achieve both low computing-side cache consumption and low memory-side read amplifications because there exists a trade-off between these two aspects, as shown in Figure 1. Range indexes on DM can be classified into two categories, i.e., those that store KV items contiguously [32, 36] and discretely [56]. Discretely storing KV items, e.g., SMART [6], reduces read amplifications since each KV item is mapped to a unique memory address and accessed individually. However, they suffer from high computing-side cache consumption since they need to cache an address for each KV item. In

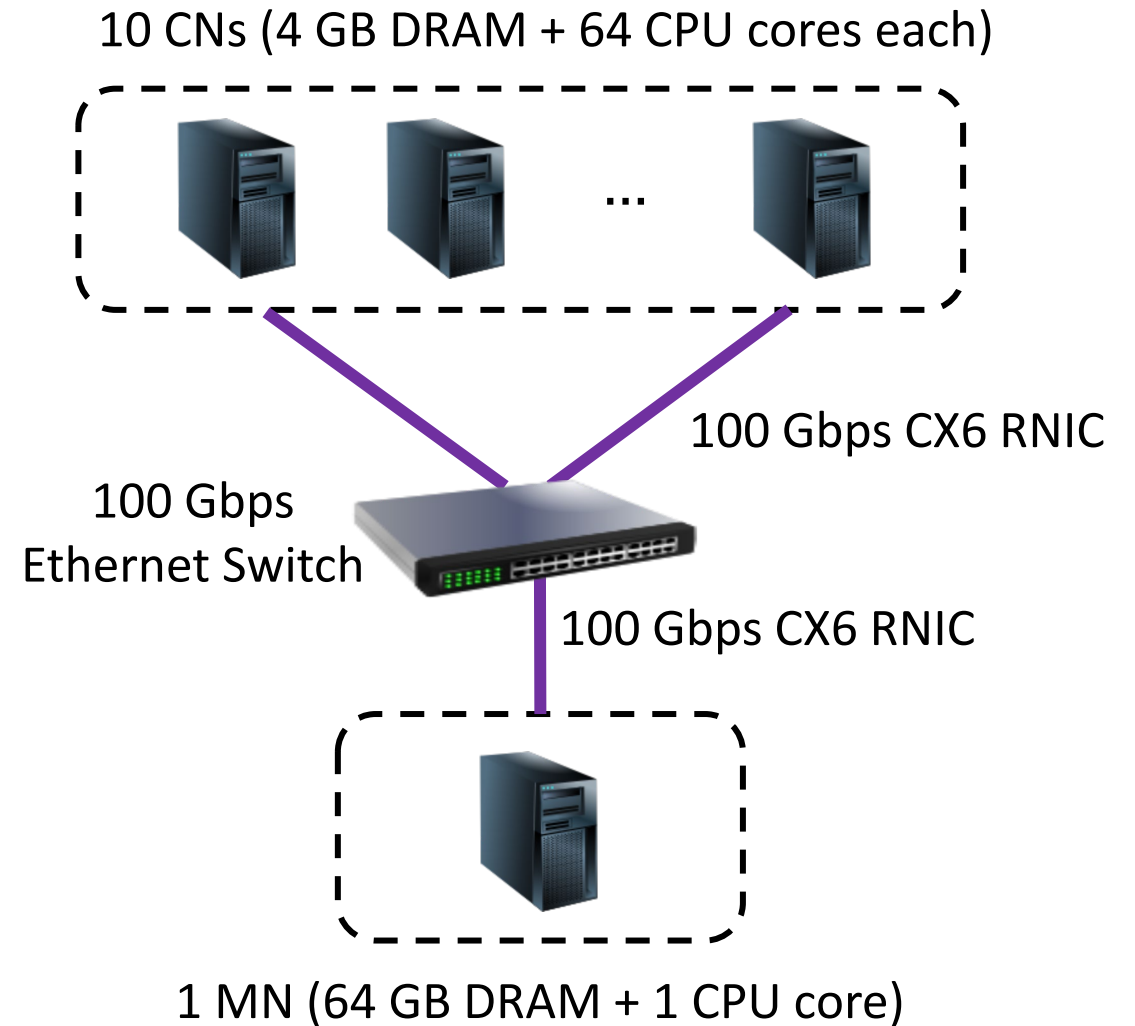
# Evaluation

## Workloads and Parameters

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

## Comparisons

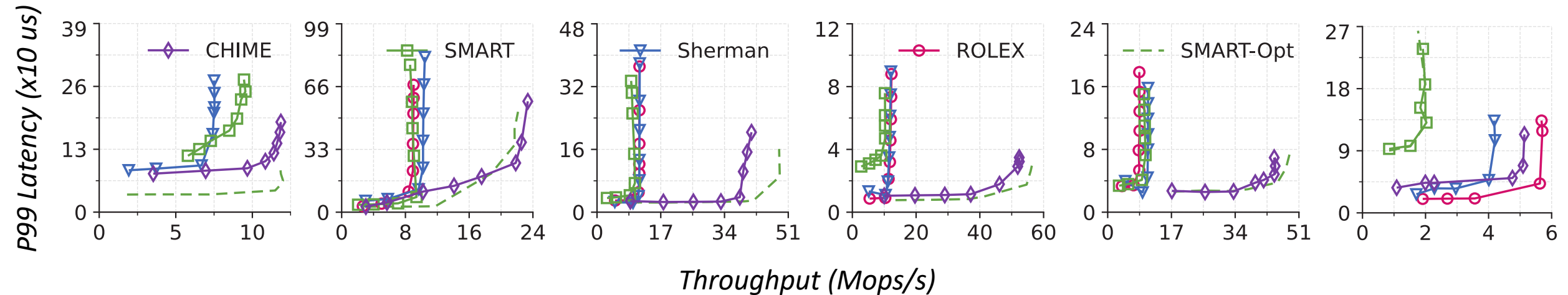
- SMART [OSDI'23]
  - The latest radix tree design on DM
- Sherman [SIGMOD'22]
  - The classic B+ tree design on DM
- ROLEX [FAST'23]
  - The latest learned index on DM
- SMART-Opt (*Optimal case*)
  - SMART with sufficient caches





# 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

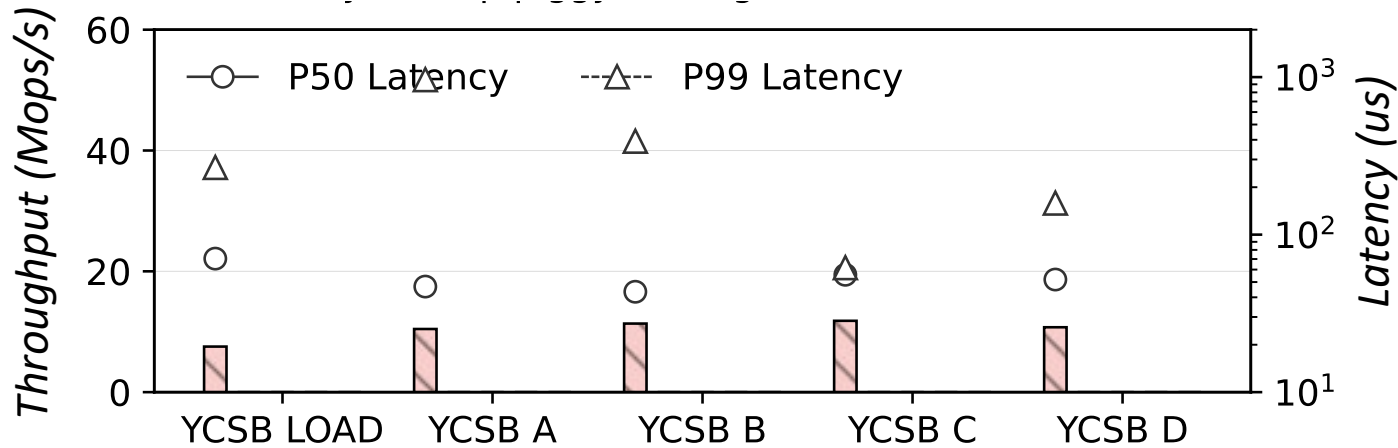


- CHIME achieves:
  - Up to **4.3x higher** throughput than Sherman and ROLEX
  - Up to **5.1x higher** throughput than SMART
  - A **close performance** to the optimal case, with up to **8.7x lower** cache consumption (**57.6 MB vs. 503.6 MB**)

# 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

■ Sherman

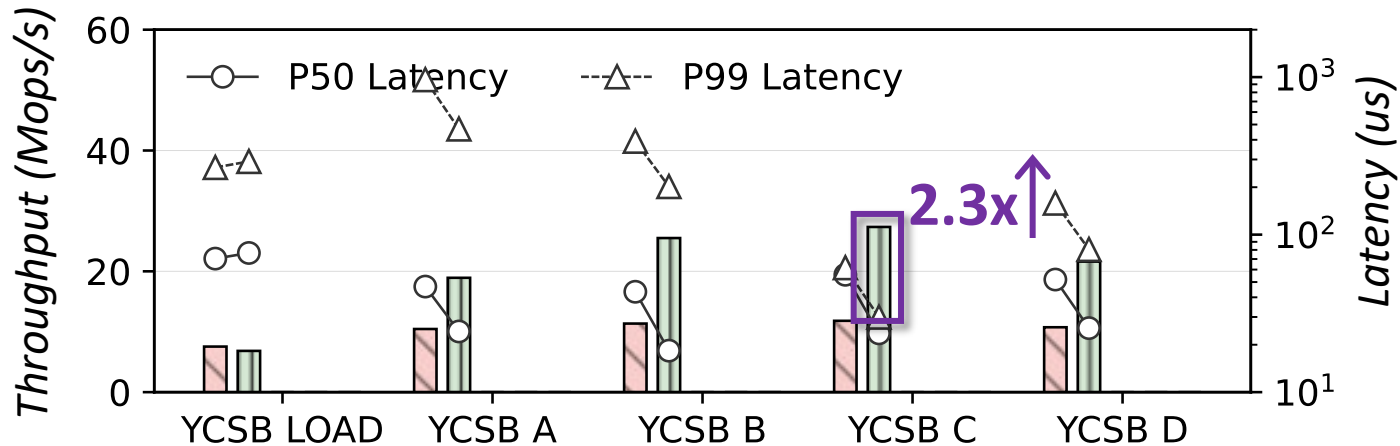


- Start with Sherman and apply each proposed technique one by one

# 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

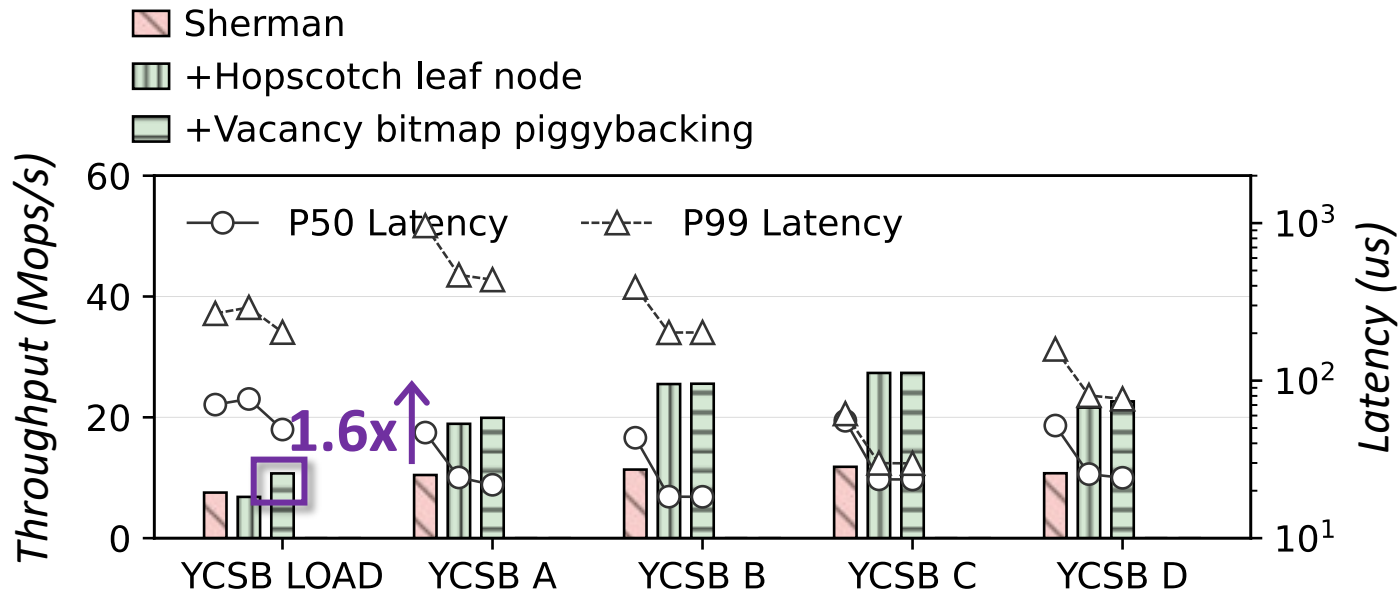
■ Sherman  
■ +Hopscotch leaf node



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

# Factor Analysis

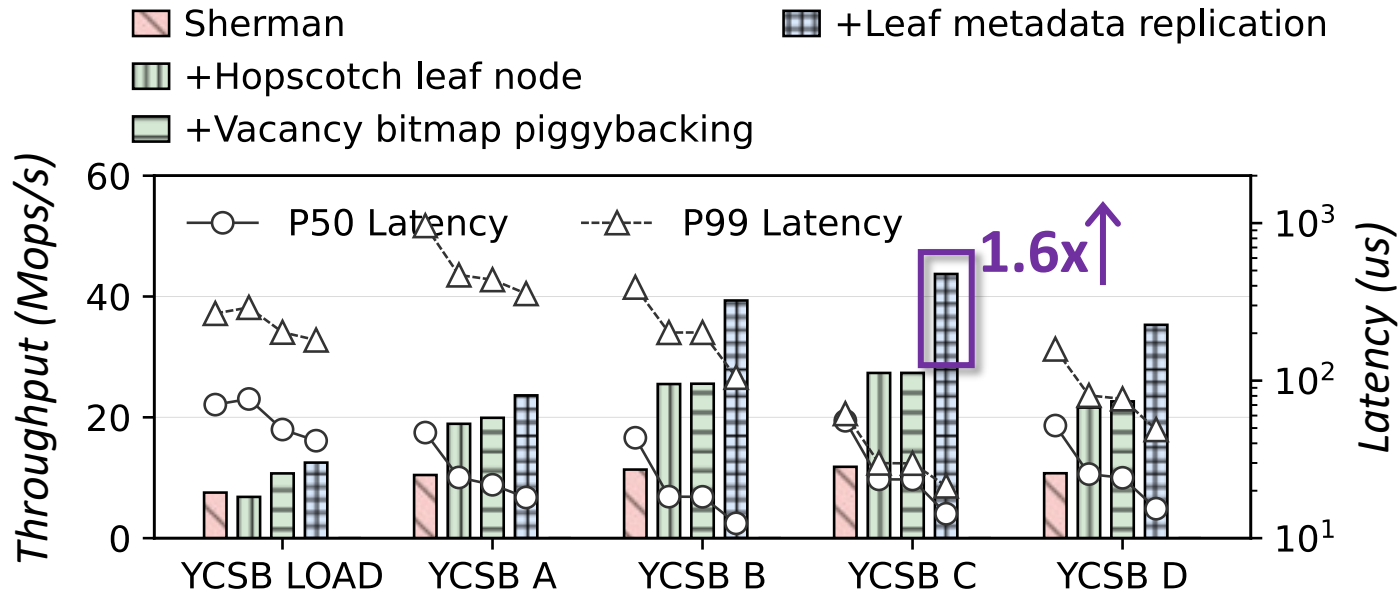
YCSB LOAD	YCSB A	YCSB B	YCSB C	YCSB D
100% insert	50% read 50% update	95% read 5% update	100% read	95% read 5% insert



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

# Factor Analysis

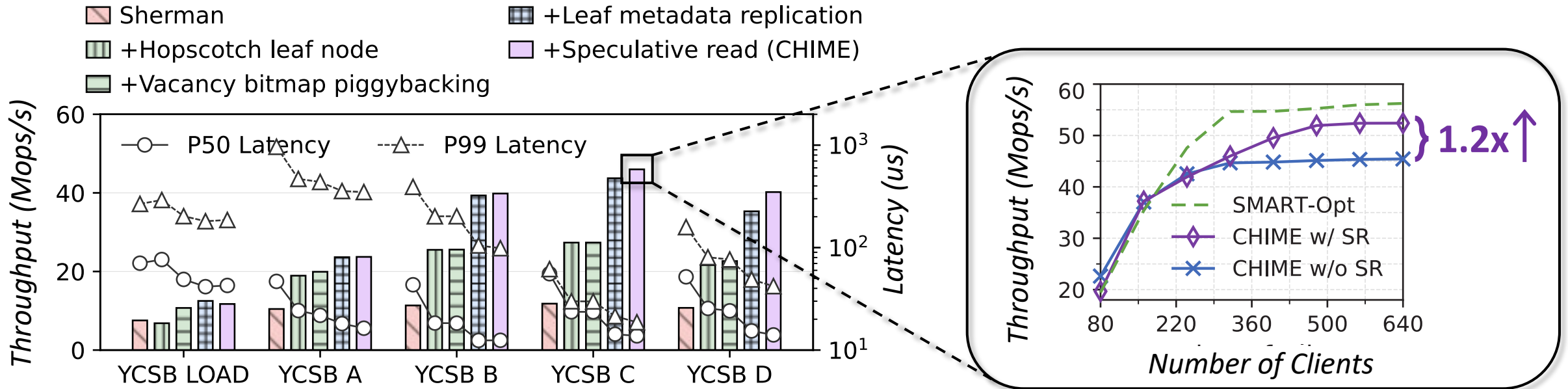
YCSB LOAD	YCSB A	YCSB B	YCSB C	YCSB D
100% insert	50% read 50% update	95% read 5% update	100% read	95% read 5% insert



- 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

# Thank you! Q&A



<https://github.com/dmemsys/CHIME>



復旦大學  
FUDAN UNIVERSITY



昆山杜克大學  
DUKE KUNSHAN  
UNIVERSITY



HUAWEI



香港中文大學  
The Chinese University of Hong Kong