OSDI | 23

17th USENIX Symposium on Operating Systems Design and Implementation

JULY 10–12, 2023

BOSTON, MA, USA

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE

ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL

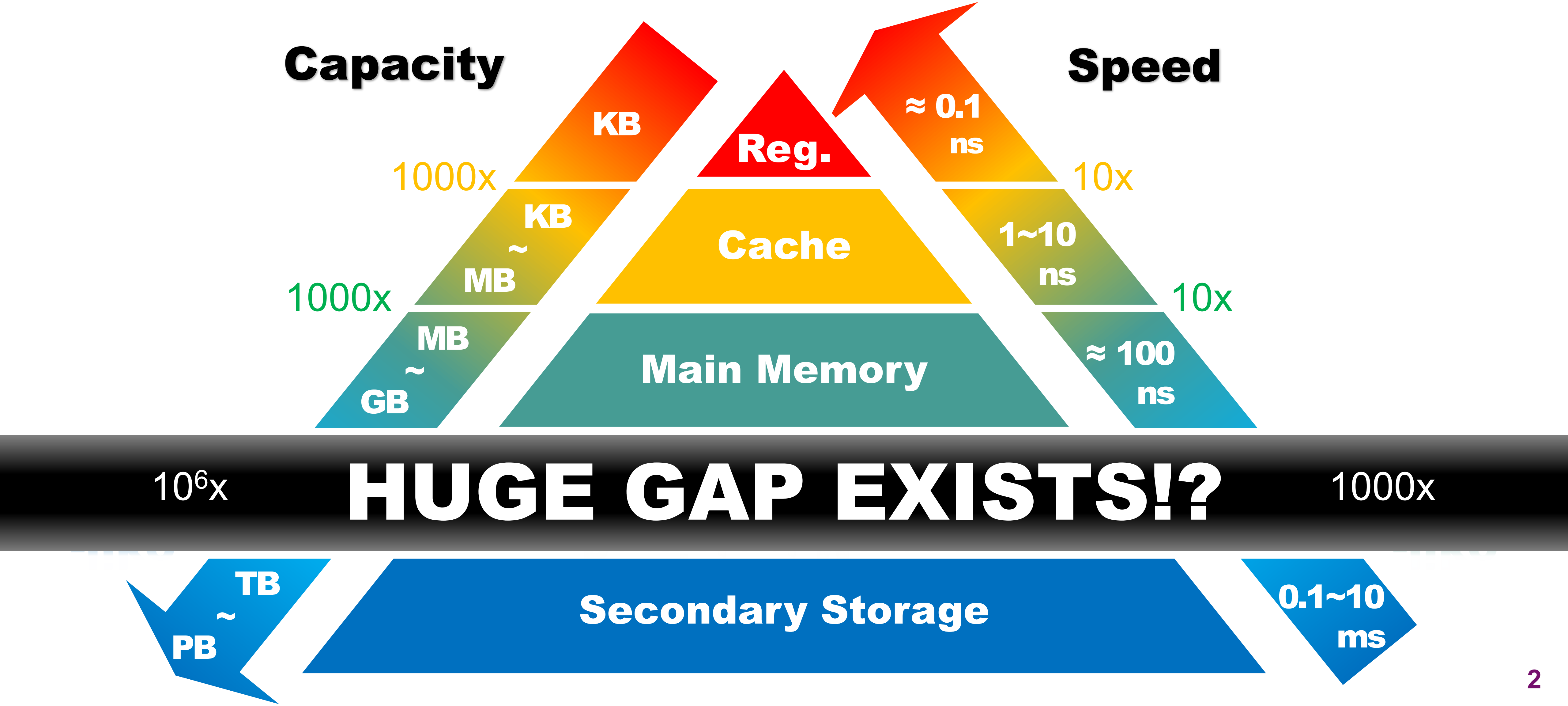
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

SEPH: Scalable, Efficient, and Predictable Hashing on Persistent Memory

Chao Wang, Junliang Hu, Tsun-Yu Yang, Yuhong Liang, and Ming-Chang Yang

The Chinese University of Hong Kong

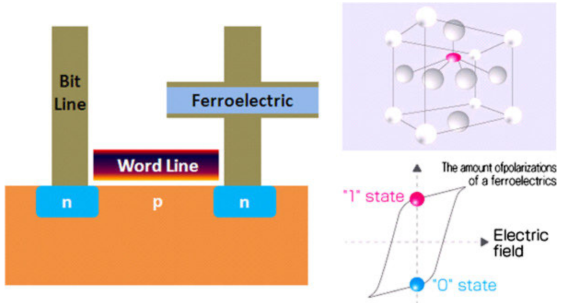
Why Persistent Memory (PM)?



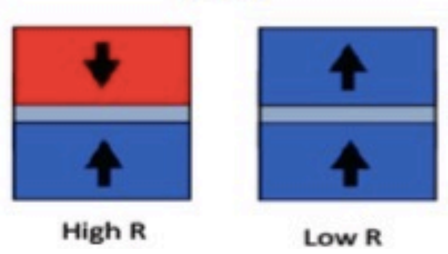
What is Persistent Memory (PM)?

- **PM** is a collective term:

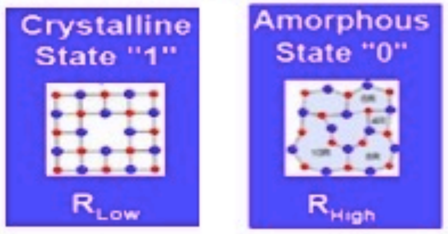
Ferroelectric RAM (FeRAM)



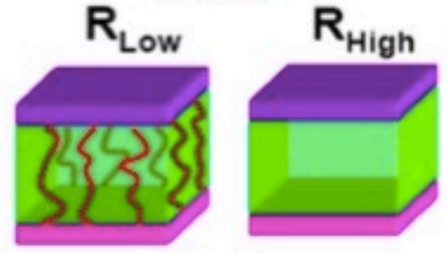
Magnetic RAM (MRAM)



Phase-Change RAM (PCRAM)



Resistive RAM (ReRAM)



- The first PM product is commercially available in 2019.

Storage-like Capacity

- 128, 256, 512 GB

- Native persistence

Intel® Optane™ DCPMM

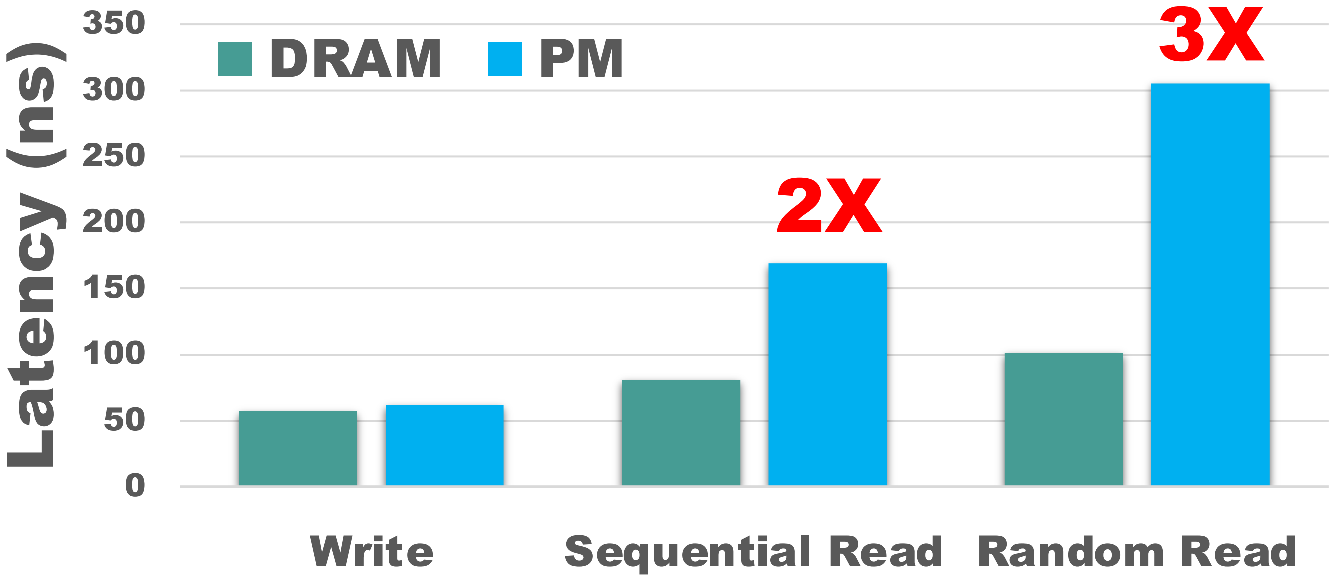
Memory-like Speed

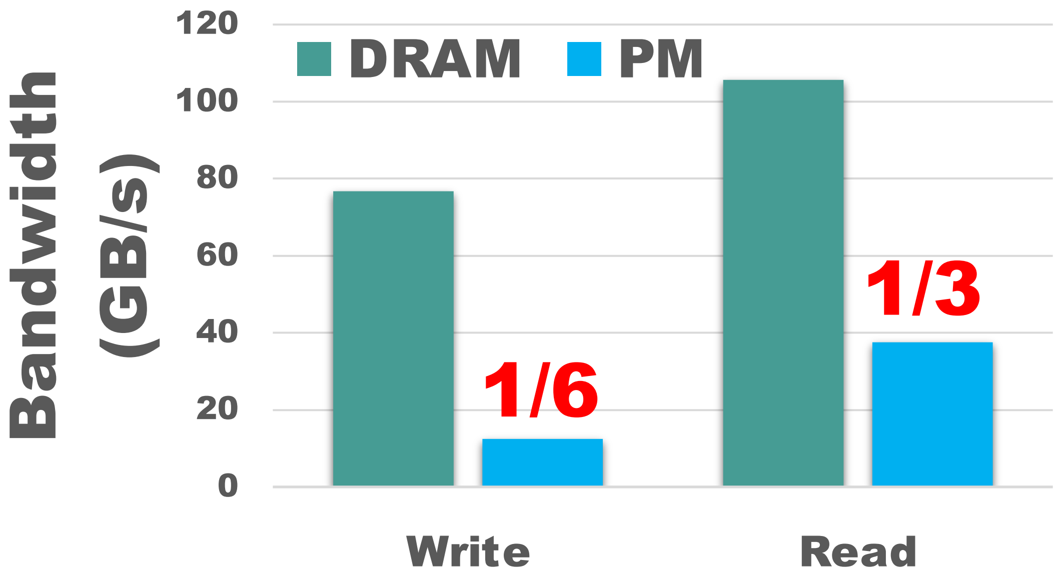
- DRAM-level latency

- DRAM-level bandwidth

- Direct load/store access

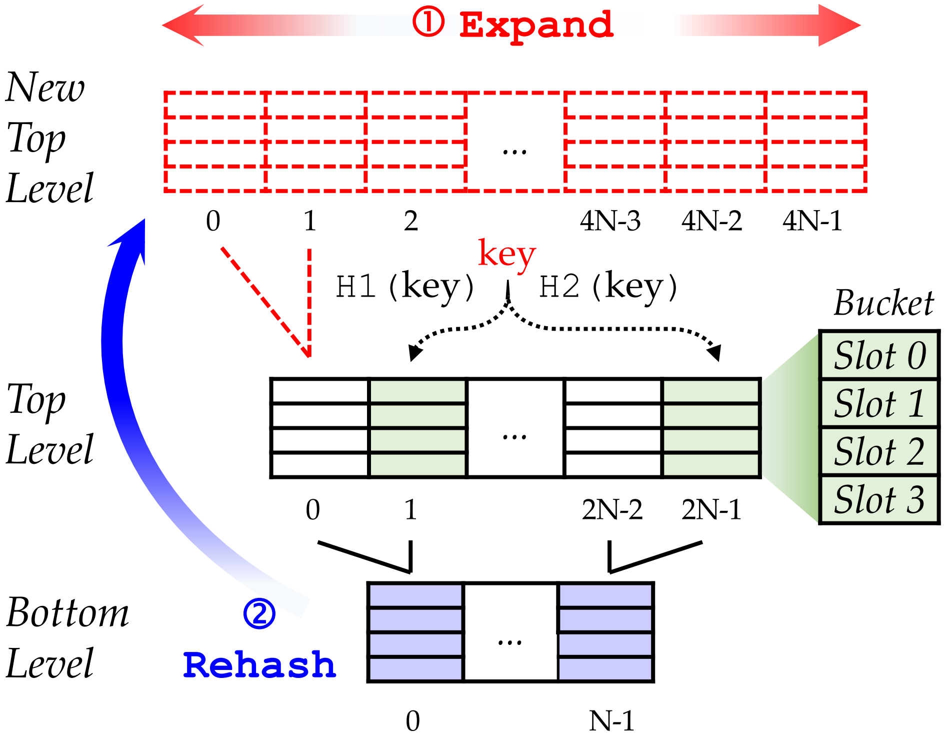
Differences exist between DRAM & PM!

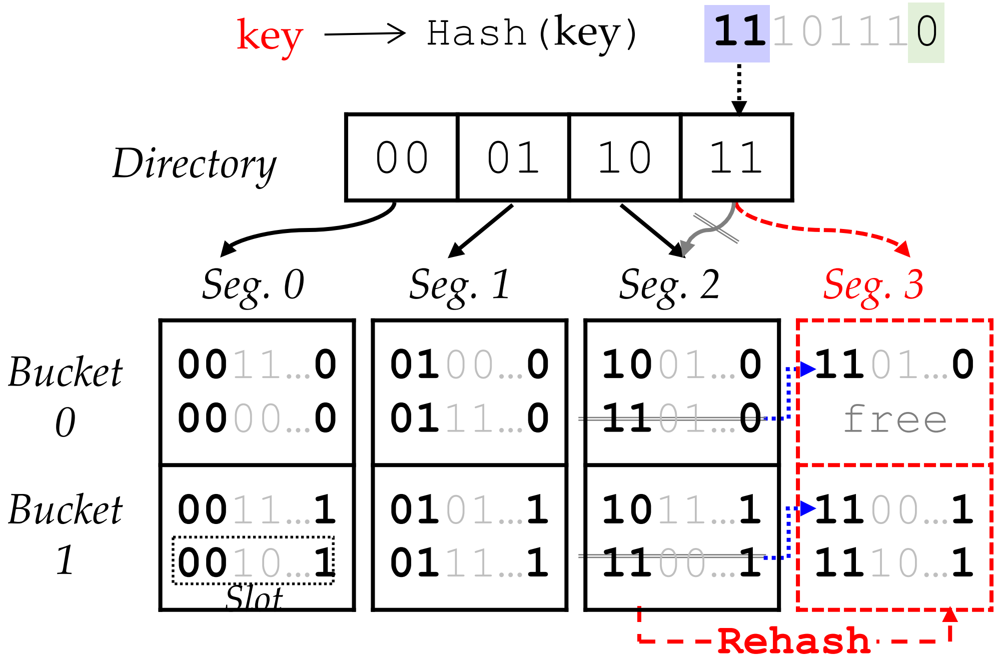




- Indexes/algorithms need to be "re-tailored" for PM!
  - **Tree**: NV-tree [FAST'15], wB+-Tree [VLDB'15], WORT [FAST'16], BzTree [VLDB'18], FAST&FAIR [FAST'18], LB+-Trees [VLDB'20], and ROART [FAST'21], etc.
  - **Hashing**: Level Hashing [OSDI'18], CCEH [FAST'19], Clevel Hashing [ATC'20], Dash [VLDB'20], etc.

Existing PM Hashing Schemes: Two Series





- **Level-based PM Hashing**
  (Level Hashing[OSDI'18], Clevel Hashing[ATC'20])
  - Sharing-based two-level structure
  - Cost-efficient resizing to mitigate the performance degradation

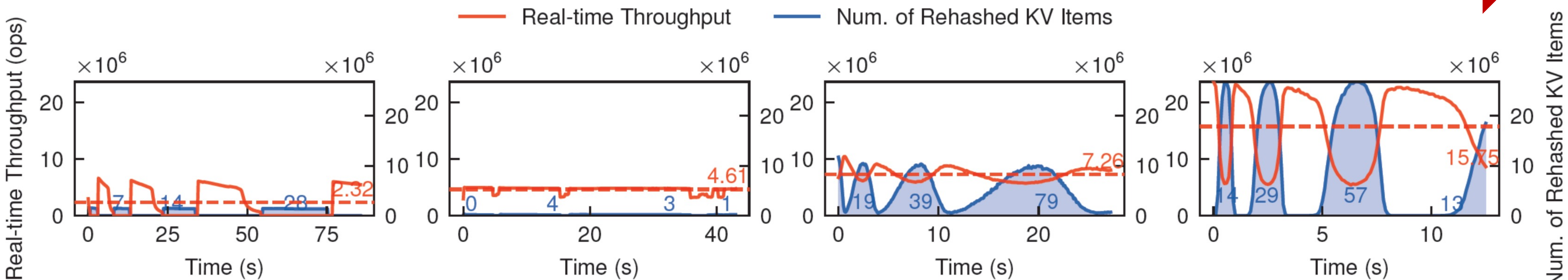
- **EH-based PM Hashing**
  (CCEH [FAST'19], Dash [VLDB'20])
  - Inherited from Extendible Hashing
  - Cacheline-conscious designs for high throughput

Motivation (1/2)

- **Observation 1:** Existing PM hashing schemes face the dilemma between the *performance efficiency* and *predictability*.

Efficiency

(High Average Throughput)



(a) (Level-based) Level Hashing

(b) (Level-based) Clevel Hashing

(c) (EH-based) CCEH

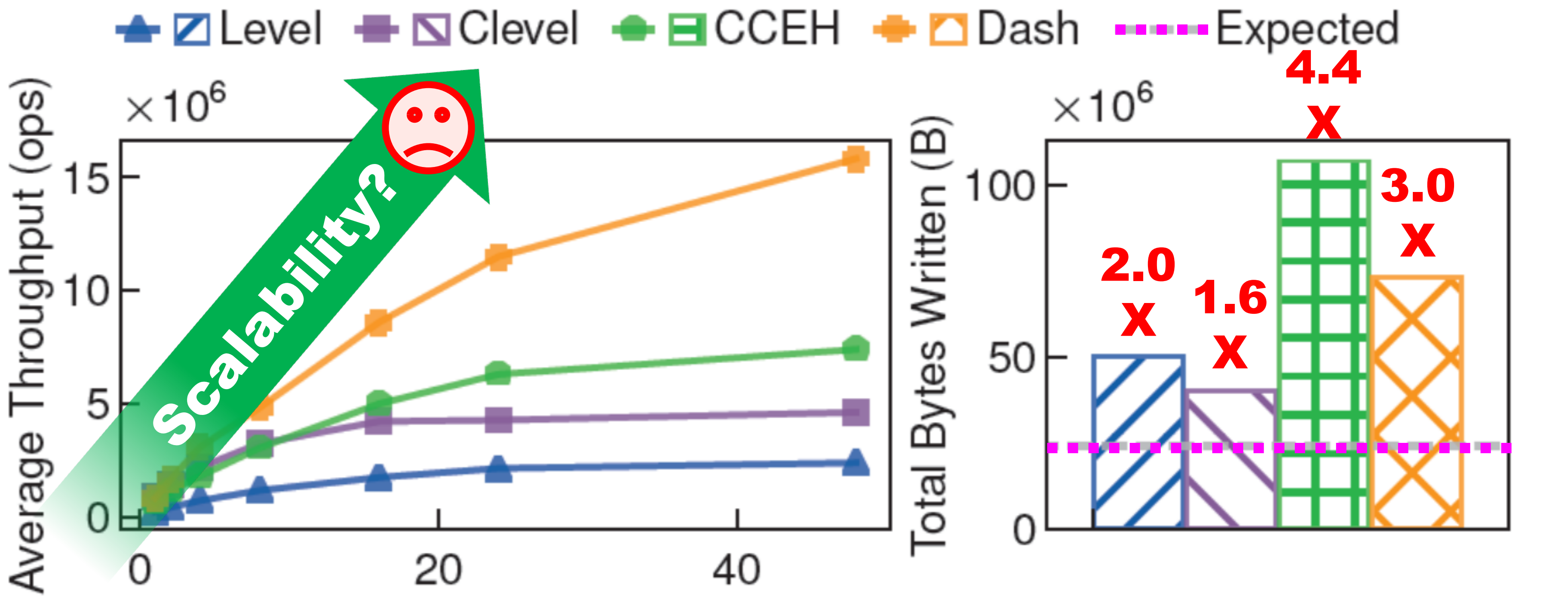
(d) (EH-based) Dash-EH

Predictability

(Low Resizing Overhead)

Motivation (2/2)

- **Observation 2:** *Performance scalability* is limited due to excessive writes in handling concurrency control.



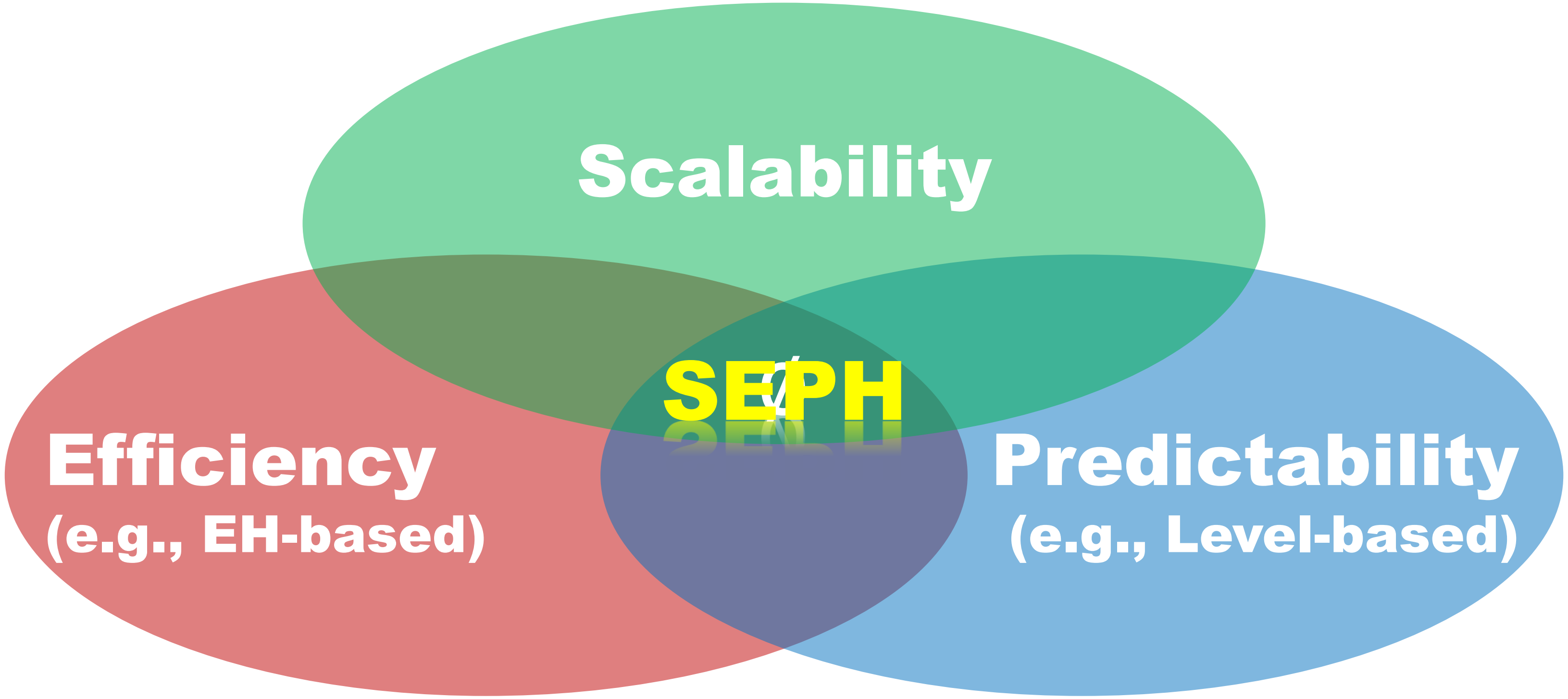
(a) Average Throughput S50% I50%

(b) Total Bytes Written to PM

Our Goal

Limited *scalability*?

Semi Lock-Free Concurrency Control

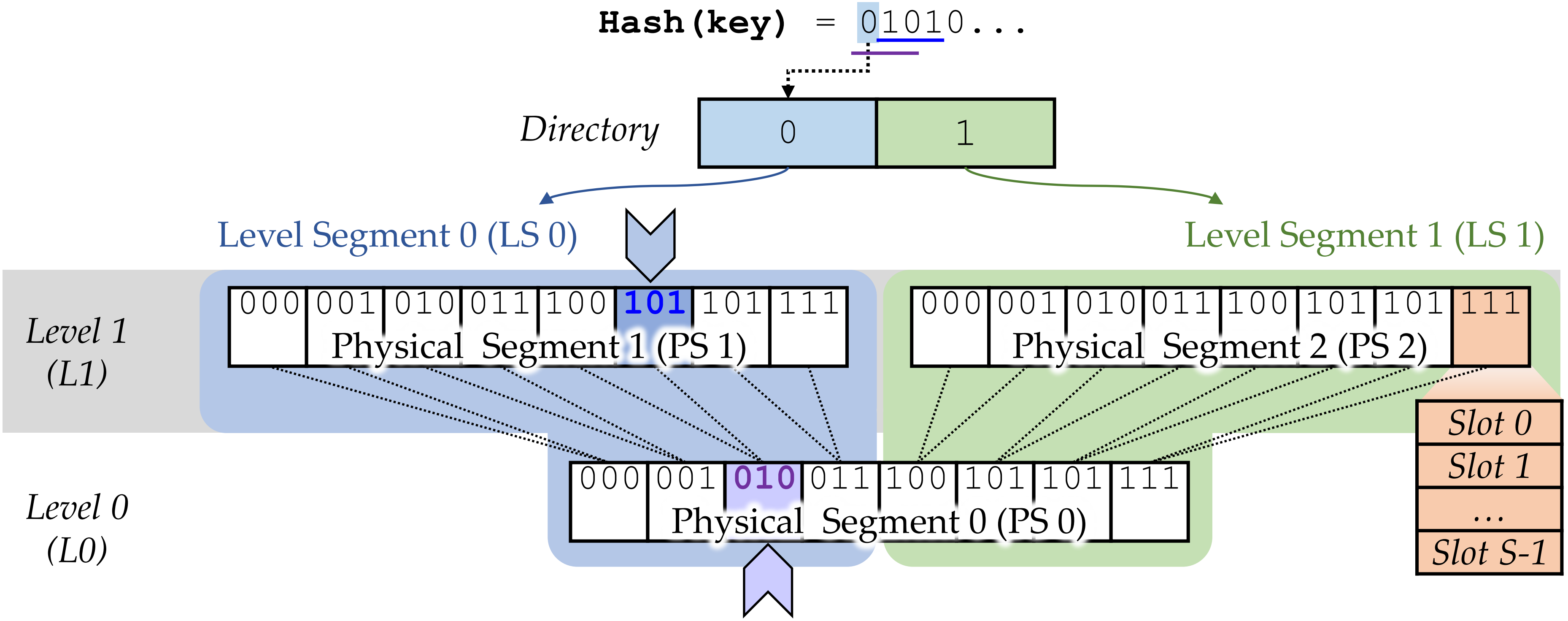


Dilemma between *efficiency* and *predictability*?

Level Segment Structure & Low-Overhead Split

SEPH: Level Segment based Hash Table

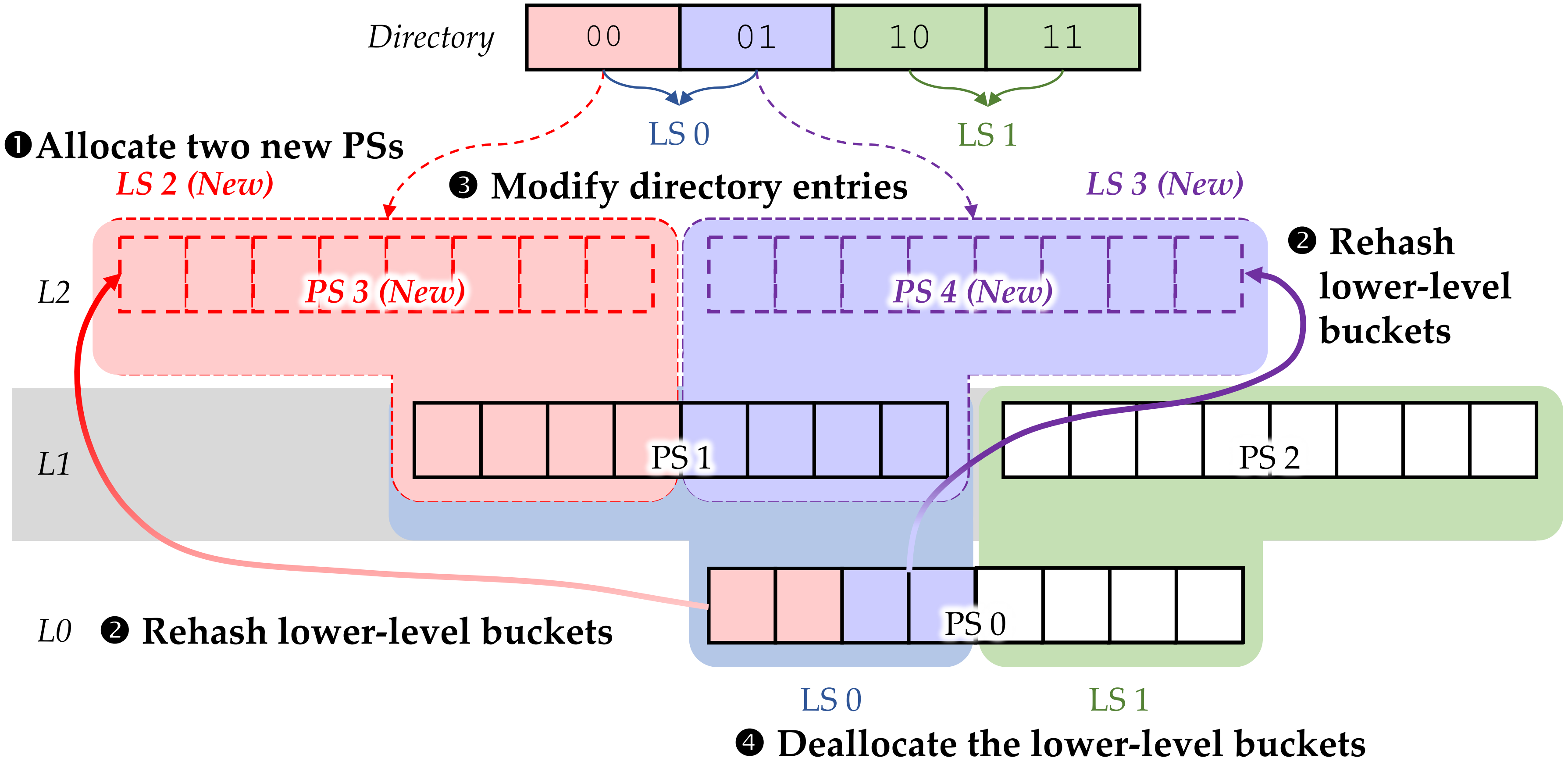
- **Level segment (LS)**, a novel structure proposed to combine the respective strengths of the two series of PM hashing.



- Physical Segment ⇒ cacheline-conscious designs (for efficiency)
- Level Segment ⇒ cost-efficient resizing (for predictability)

SEPH: Low-Overhead Split (1/2)

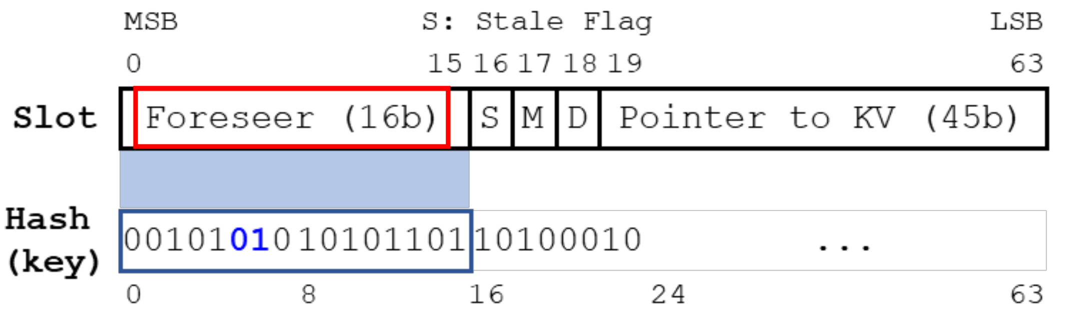
- **One-third Split**
  - Splits one LS into two, but only rehashes "1/3" of the KV items


Directory
00
01
10
11
LS 0
LS 1
❶ Allocate two new PSs
LS 2 (New)
❸ Modify directory entries
LS 3 (New)
❷ Rehash lower-level buckets
L2
PS 3 (New)
PS 4 (New)
L1
PS 1
PS 2
L0
❷ Rehash lower-level buckets
PS 0
LS 0
LS 1
❹ Deallocate the lower-level buckets

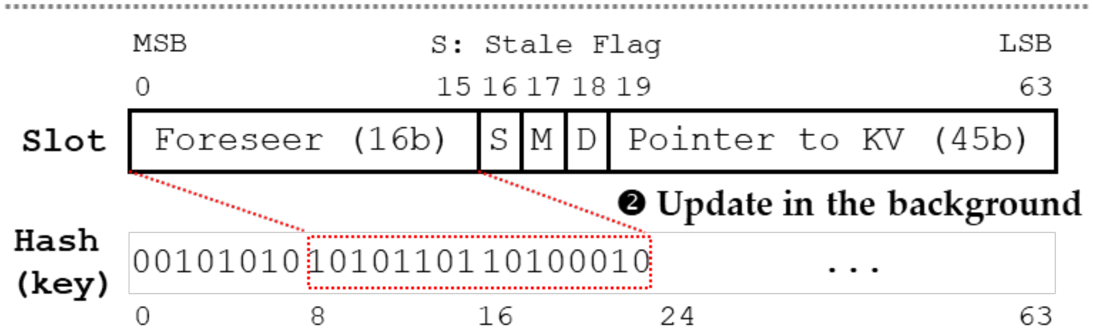
SEPH: Low-Overhead Split (2/2)

- Common Practice: Variable-length key ⇒ store KV pointers
- Potential Problem: ❷Rehashing requires ~~pointer dereferences~~ to calculate *Hash(key)* ↪ PM random read

- **Dereference-Free Rehashing**
  - Only two bits of *Hash(key)* are needed for ❷Rehashing.
  - We stores these bits in advance, as a foreseer of KV's future position.



❶ Foresee the sliding bucket index for dereference-free rehashing



SEPH: Semi Lock-Free Concurrency Control

- Scalability ☹ ⇐ excessive PM writes for concurrency control

	Lock-based Designs (e.g. Level Hashing, CCEH, Dash)	Lock-free Designs (e.g. Clevel Hashing)
PM writes are to	Manage Locks	Guarantee Correctness

- SEPH solves it by

Frequent Operations (e.g. insert, search, update, delete)	Infrequent operatoins (e.g. split)
Be Lock-free (to save PM writes)	Be Lock-based (to ease correctness guarantee)

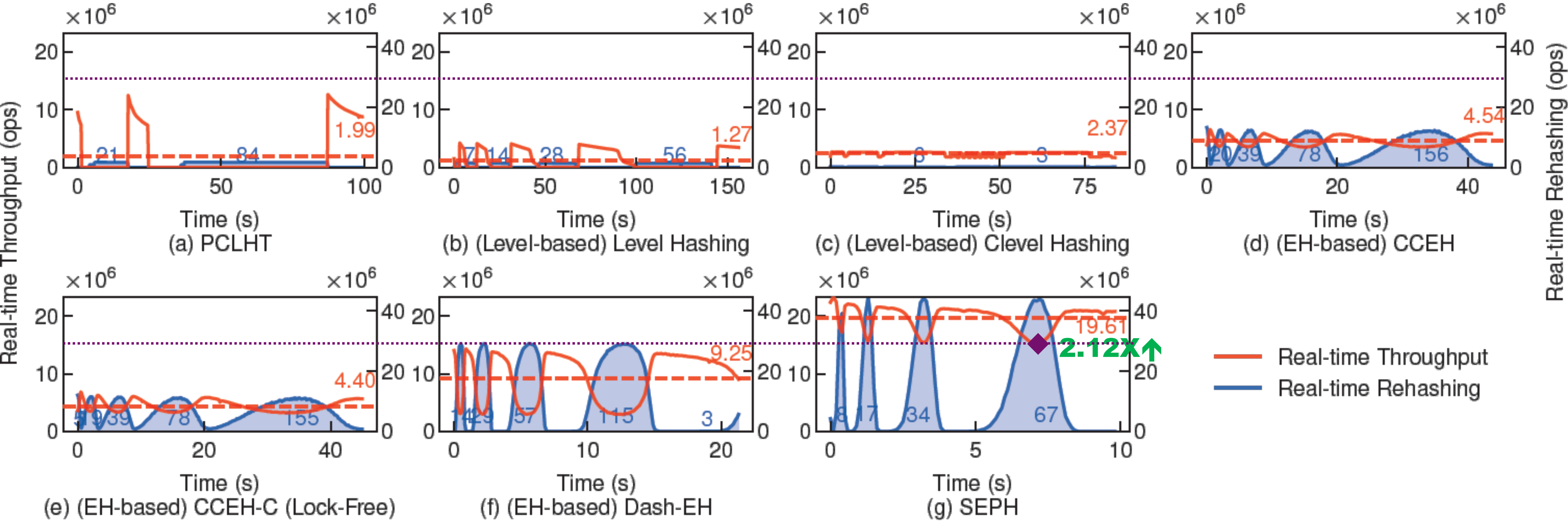
- Thus, SEPH achieves nearly minimal PM writes and scales well.

Evaluation Setup

- The following hashing schemes are compared with **`SEPH`**:
  - **`DRAM-converted`**: PCLHT [SOSP'19]
  - **`Level-based`**: Level Hashing [OSDI'18], Clevel Hashing [ATC'20],
  - **`EH-based`**: CCEH/CCEH-C [FAST'19], Dash [VLDB'20]

- All experiments are conducted on
  - Intel Xeon Platinum 8260 CPU
  - Six 128 GB Intel® Optane™ DCPMM 100 series in *App Direct* mode.

Evaluation Results (1/3): Efficiency & Predictability



(a) PCLHT

(b) (Level-based) Level Hashing

(c) (Level-based) Clevel Hashing

(d) (EH-based) CCEH

(e) (EH-based) CCEH-C (Lock-Free)

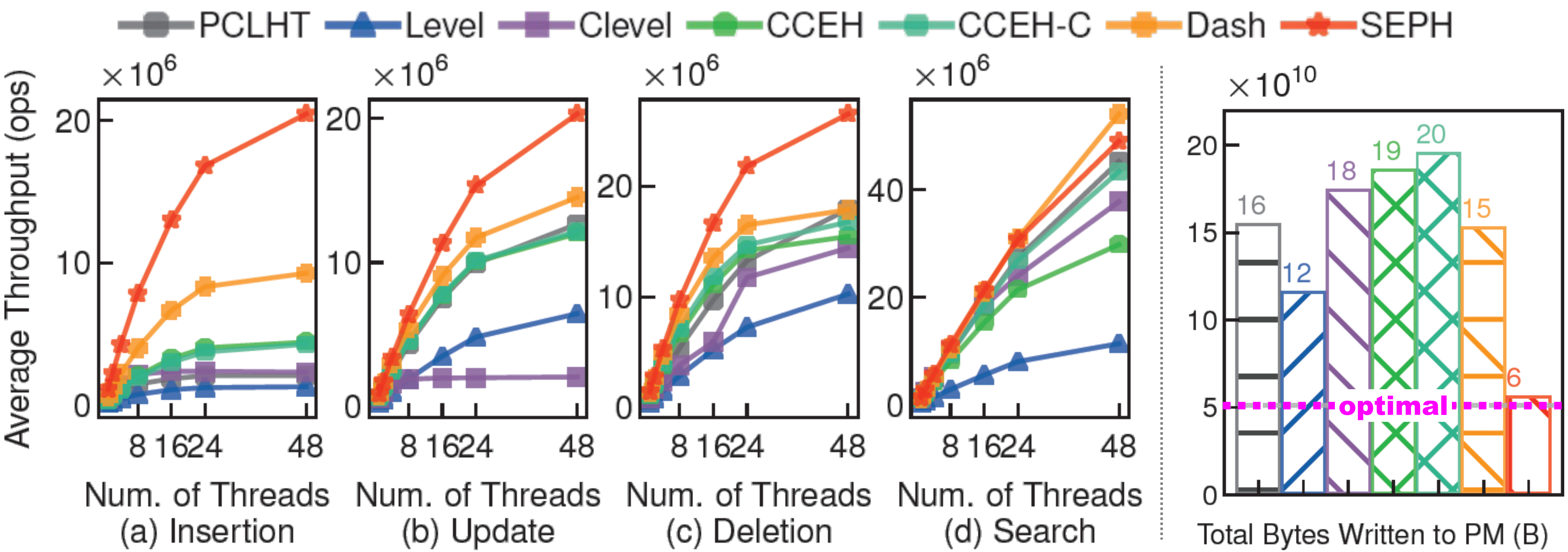
(f) (EH-based) Dash-EH

(g) SEPH

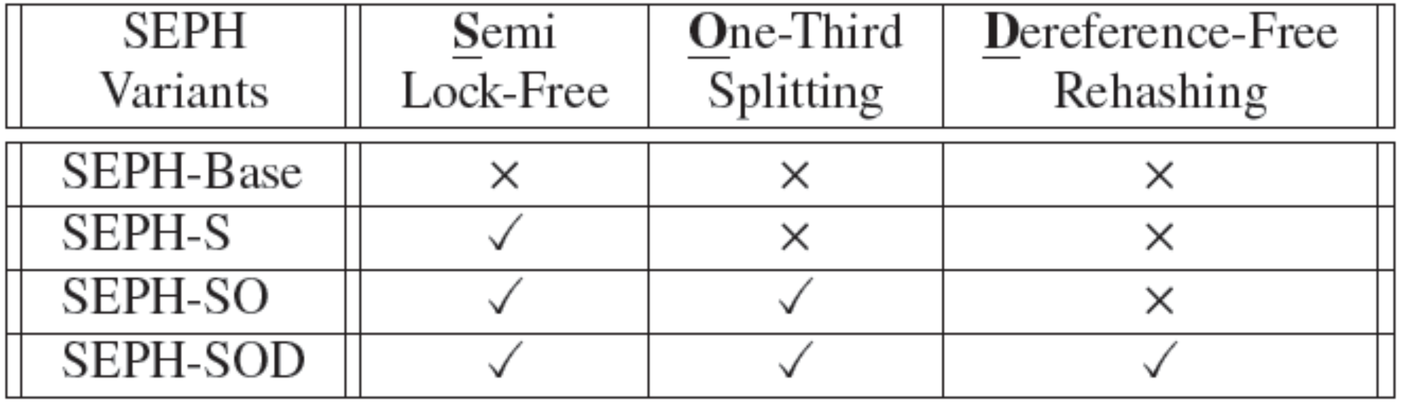
- *Efficiency*: 2.12X better average throughput.
- *Predictability*: best worst-case throughput

even > peak throughput of other designs

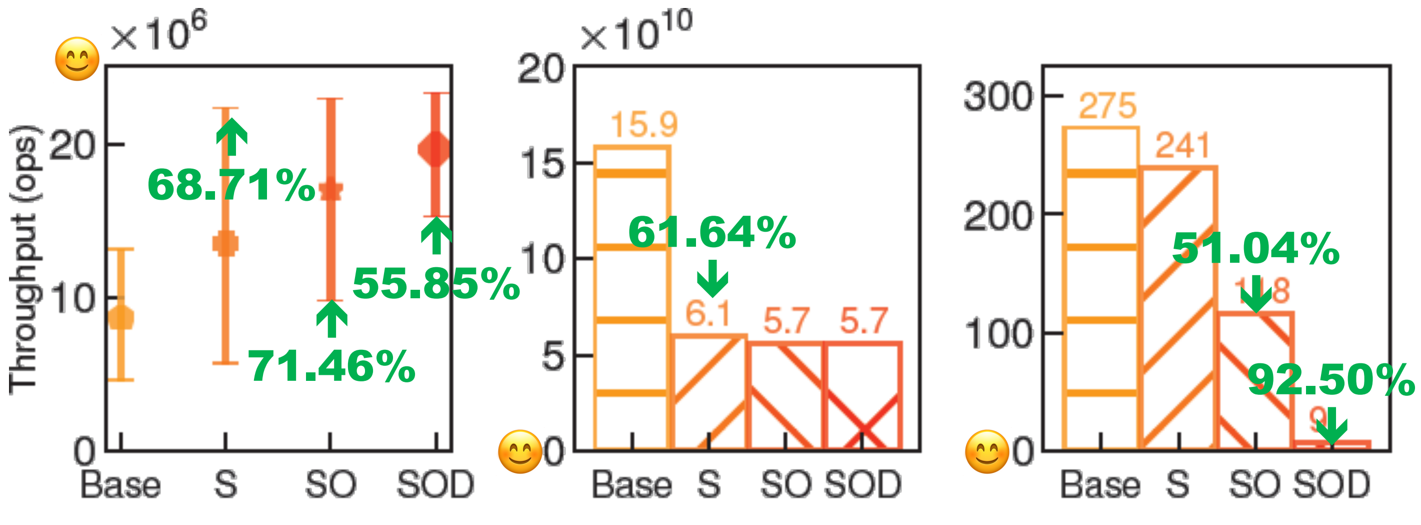
Evaluation Results (2/3): Scalability



Evaluation Results (3/3): Performance Breakdown



SEPH Variants	Semi Lock-Free	One-Third Splitting	Dereference-Free Rehashing
SEPH-Base	×	×	×
SEPH-S	✓	×	×
SEPH-SO	✓	✓	×
SEPH-SOD	✓	✓	✓



(a) Throughput Profile

(b) Reason: PM Writes

(c) Reason: Resizing Time

Summary

- **SEPH**: scalable, efficient, and predictable hashing for PM
  - *Efficiency* vs. *Predictability*
    - Level segment structure & low-overhead split algorithm.
    - To combine the strengths of two series of PM hashing.
  - *Scalability*
    - Semi lock-free concurrency control
    - To minimizing the PM writes for concurrency control
- SEPH is rigorously validated on Intel Optane and demonstrates its potential value to the time-sensitive applications.