Daegu Gyeongbuk Institute of Science & Technology

DATA-INTENSIVE COMPUTING SYSTEMS LABORATORY

CSAIL

PinK: High-speed In-storage Key-value Store with Bounded Tails

Junsu Im, Jinwook Bae, Chanwoo Chung*, Arvind*, and Sungjin Lee

Daegu Gyeongbuk Institute of Science & Technology (DGIST)
*Massachusetts Institute of Technology (MIT)

2020 USENIX Annual Technical Conference **(ATC' 20, July 15 ~ 17)**

Key-Value Store is Everywhere!

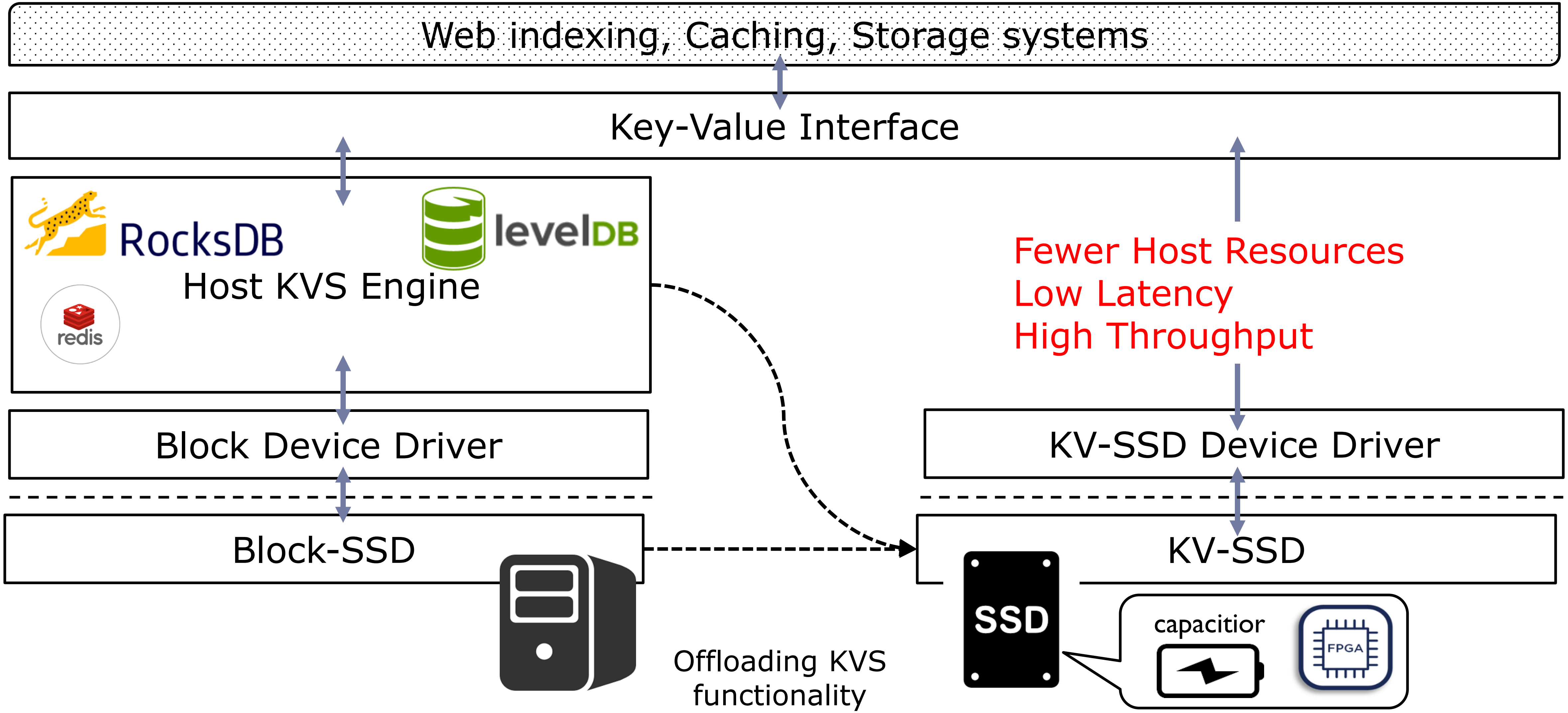
- Key-Value store (KVS) has become a necessary infrastructure

Web indexing, Caching, Storage systems

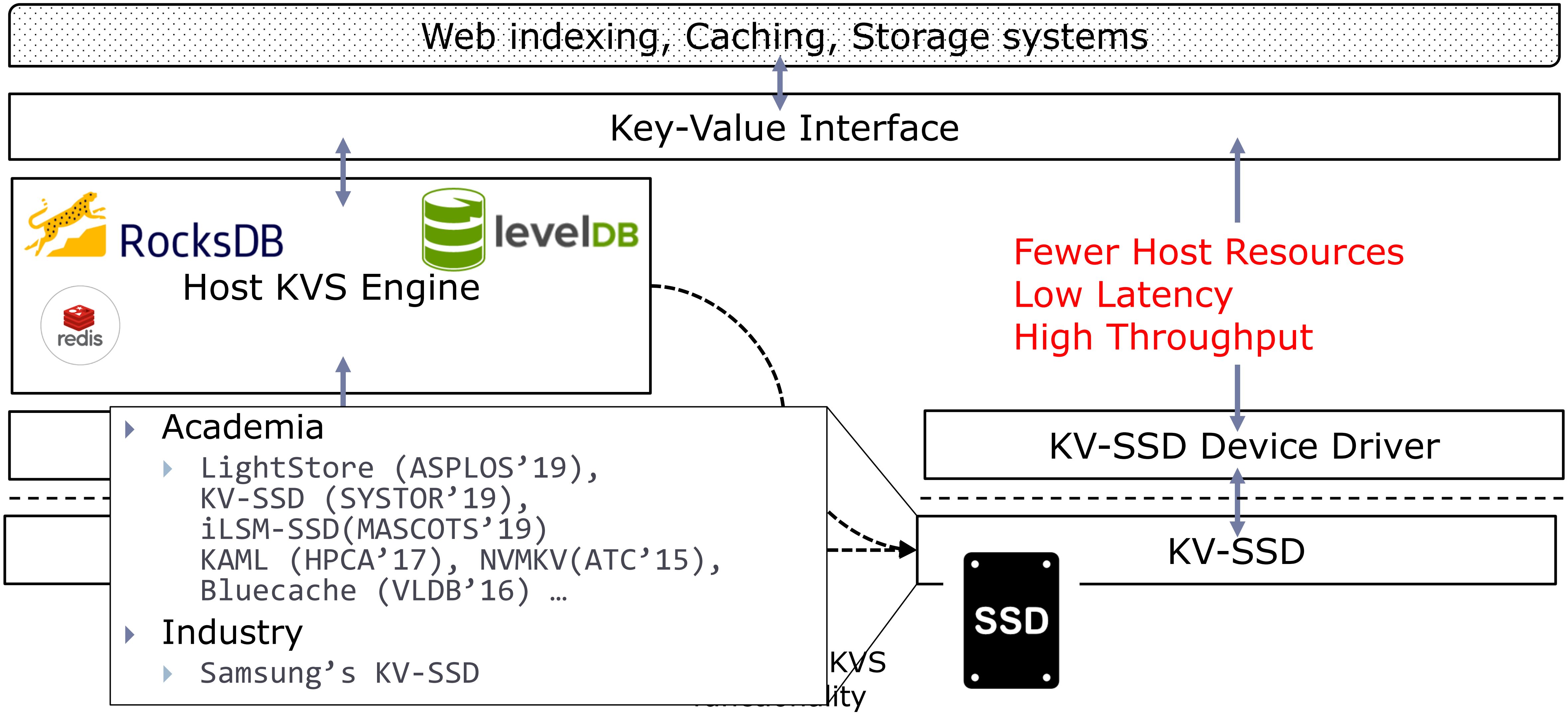
Amazon DynamoDB
mongoDB
RocksDB
levelDB
MEMCACHED
redis

- Algorithm
  - SILK (ATC'19),
  - Dostoevsky (SIGMOD'18)
  - Monkey (SIGMOD'17) …
- System
  - FlashStore (VLDB'10)
  - Wisckey (FAST'16)
  - LOCS (Eurosys'14) …
- Architecture
  - Bluecache (VLDB'16) …

Key-Value (KV) Storage Device

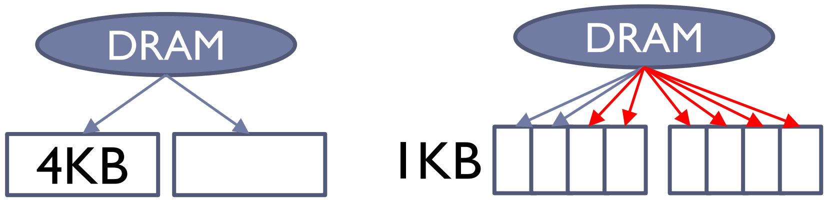


Key-Value (KV) Storage Device

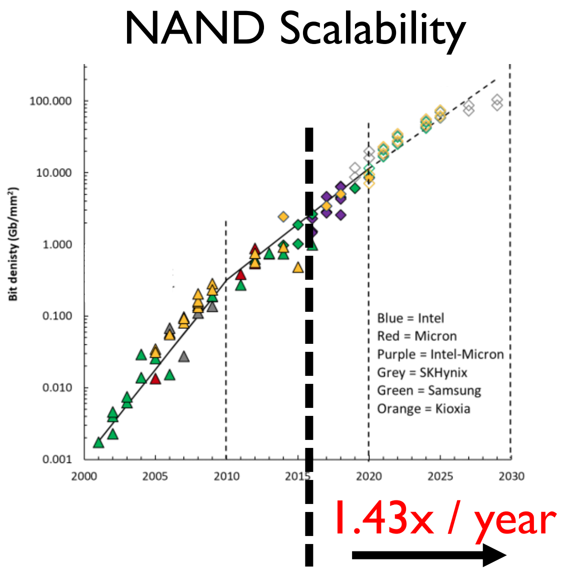


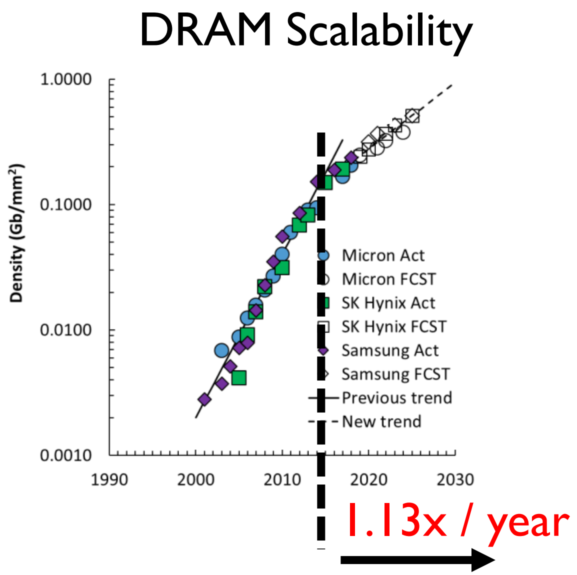
Key Challenges of Designing KV-SSD

- 1. Limited DRAM resource
  - SSDs usually have DRAM as much as 0.1% of NAND for indexing!
    - Logical block: 4KB > KV-pair: 1KB on average



  - DRAM scalability slower than NAND!



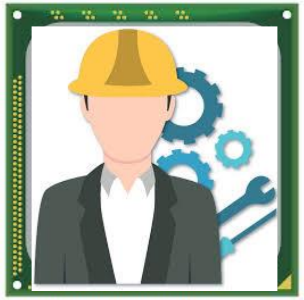


Technology and Cost Trends at Advanced Nodes, 2020, https://semiwiki.com/wp-content/uploads/2020/03/Lithovision-2020.pdf

Key Challenges of Designing KV-SSD (Cont.)

- 2. Limited CPU performance
  - SSDs have low power CPU (ARM based)

x86 CPU



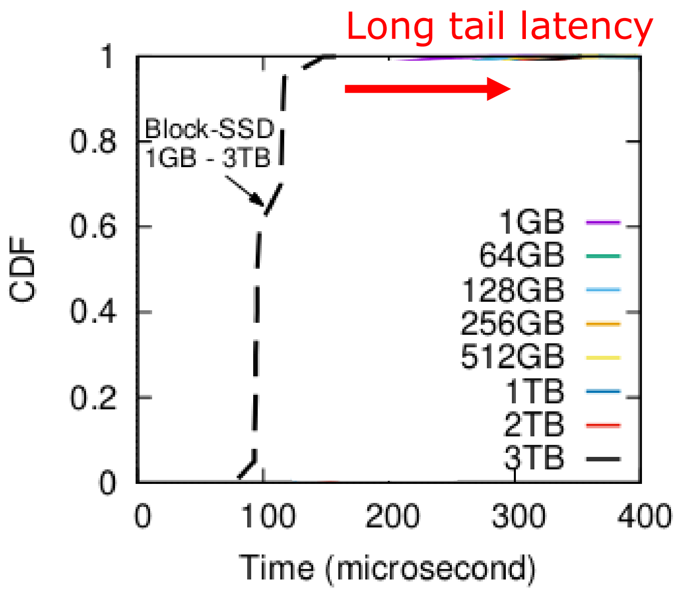
ARM CPU

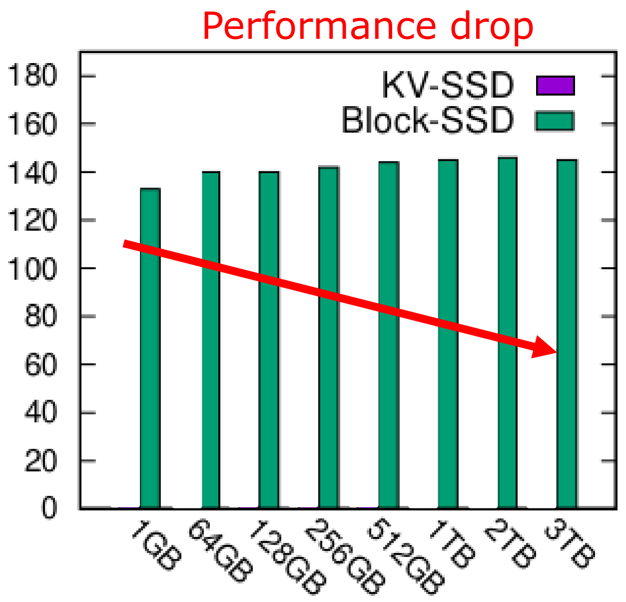
<u>Which algorithm is better for KV-SSD with these limitations,</u>

Hash or Log-structured Merge-tree (LSM-tree)?

Experiments using Hash-based KV-SSD

- Samsung KV-SSD prototype
  - hash-based KV-SSD*
- Benchmark
  - KV-SSD: KVBench**, 32B key and 1KB value read request
  - Block-SSD: FIO, 1KB read request





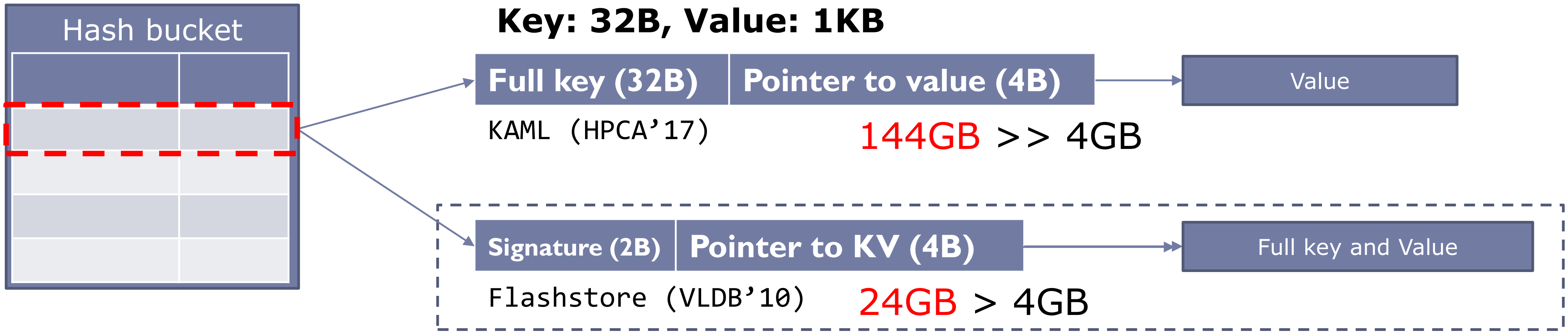
What is the reason?

*KV-PM983, **Samsung KV-SSD benchmark tool

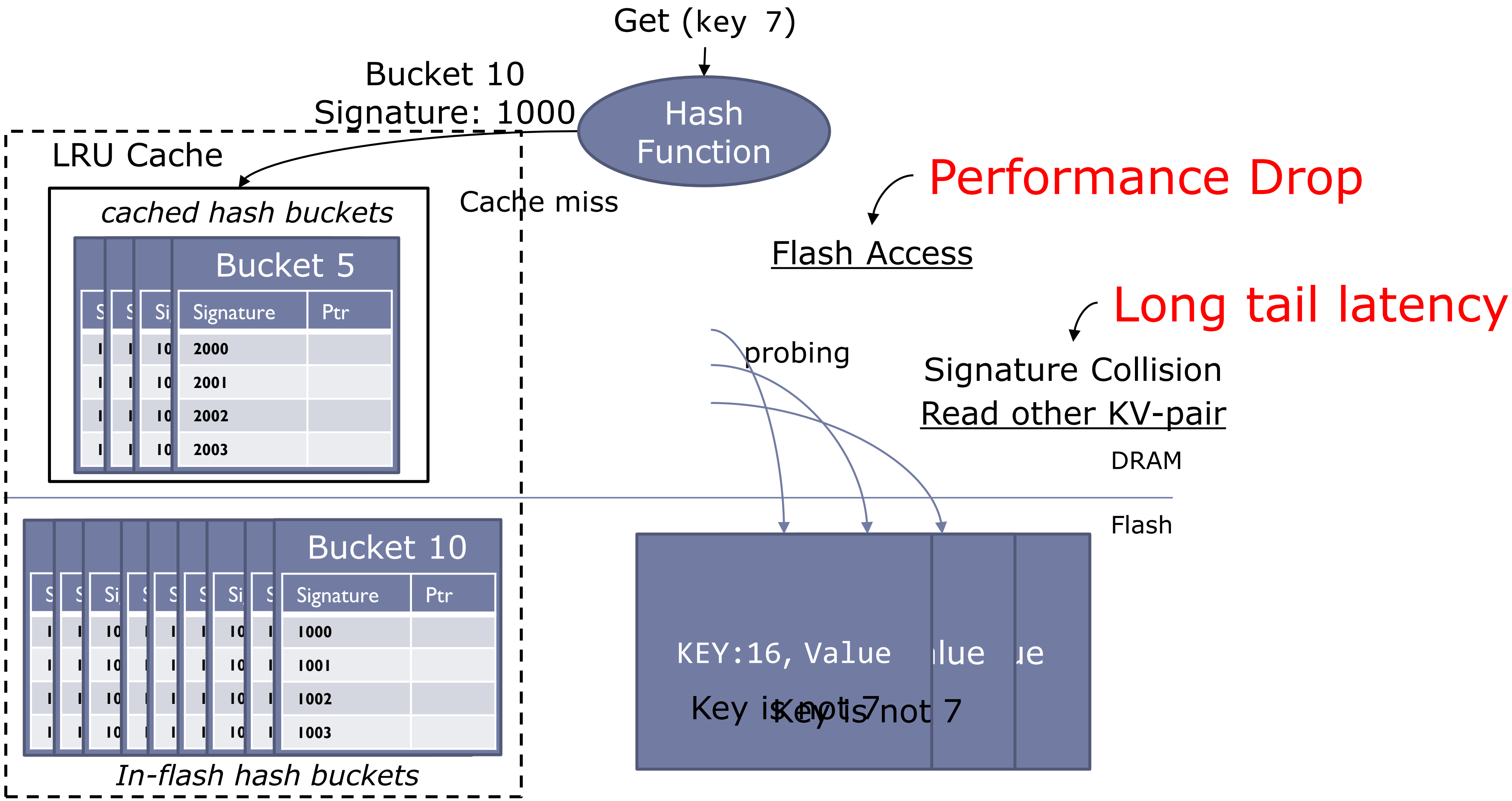
Problem of Hash-based KV-SSD

SSD: 4TB, <u>DRAM:4GB</u>

Key: 32B, Value: 1KB



Problem of Hash-based KV-SSD



LSM-tree?

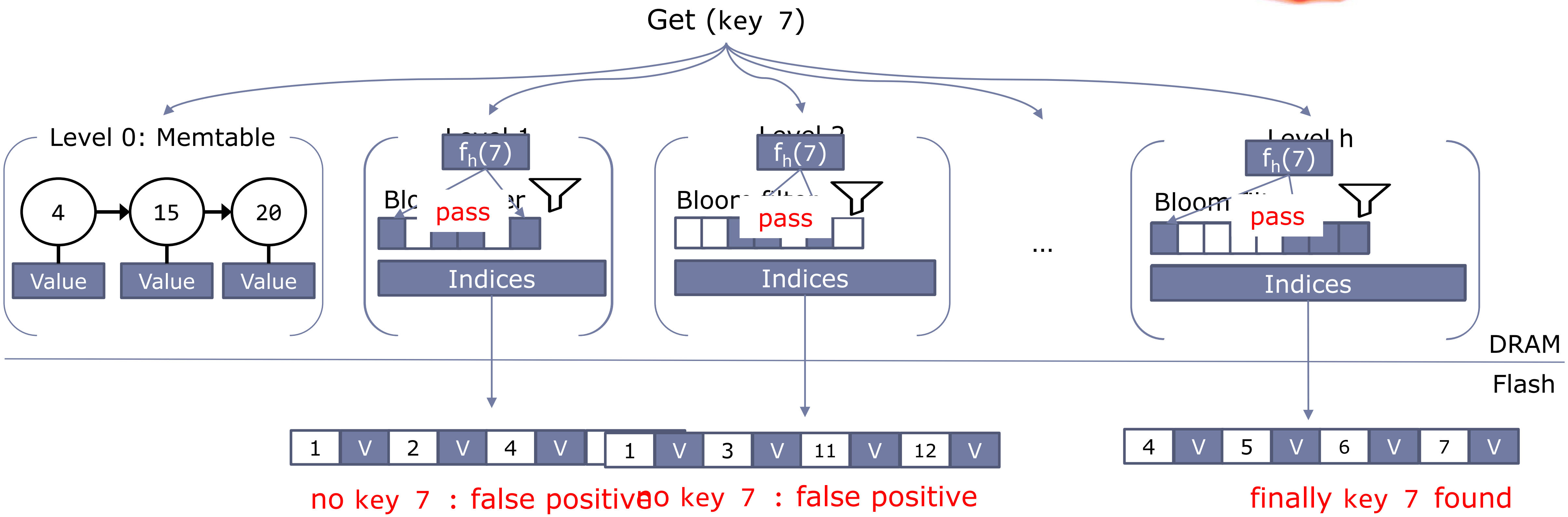
- Another Option “LSM-tree”
  - Low DRAM requirement
  - No collision
  - Easy to serve range query

<u>Is the LSM-tree really good enough?</u>

Problem of LSM-tree-based KV-SSD

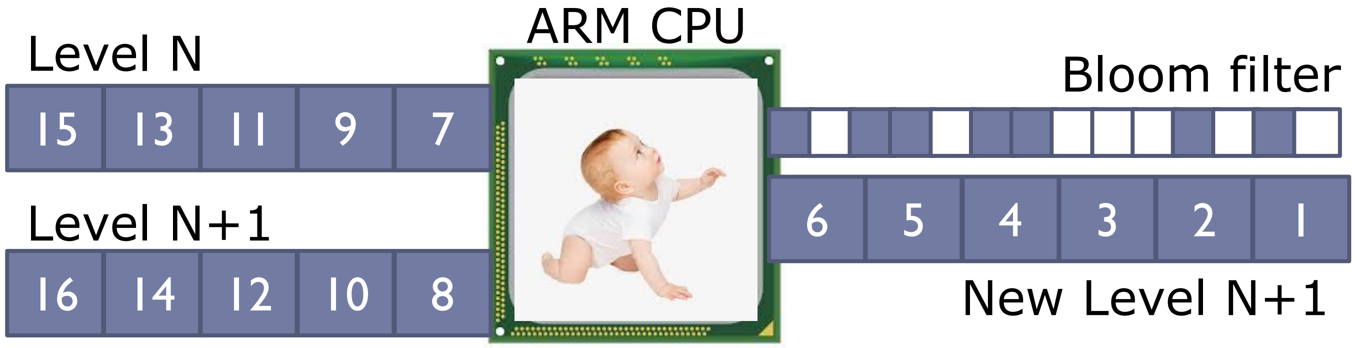
- 1. Long tail latency!

In the worst case,
h-1 flash accesses for 1 KV
(*h* = height of LSM-tree)



Problem of LSM-tree-based KV-SSD

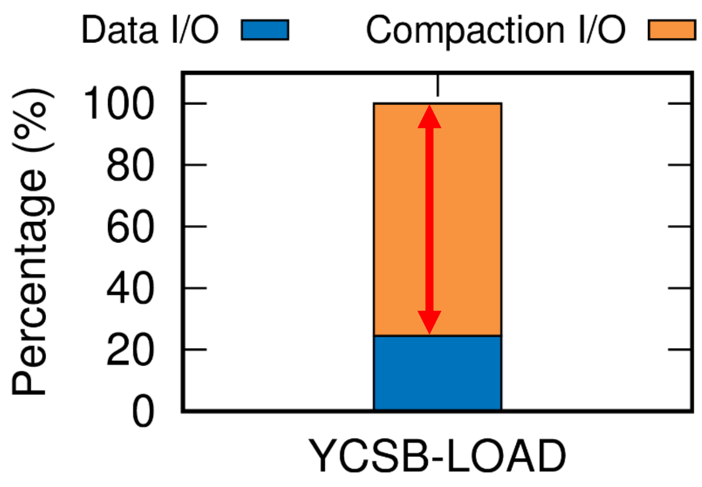
- 2. CPU overhead!
  - Merge sort in compaction
  - Building bloom filters

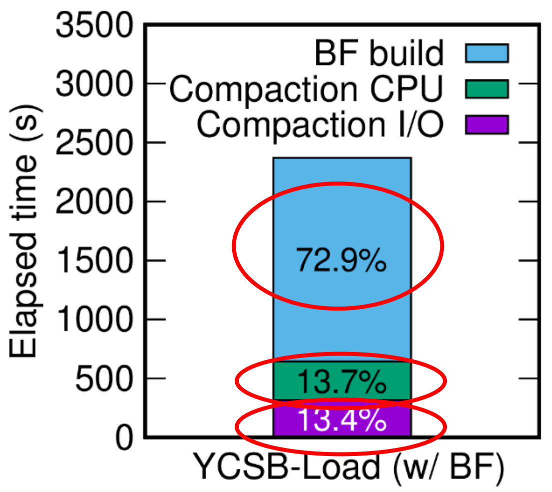


- 3. I/O overhead!
  - Compaction I/O added by LSM-tree

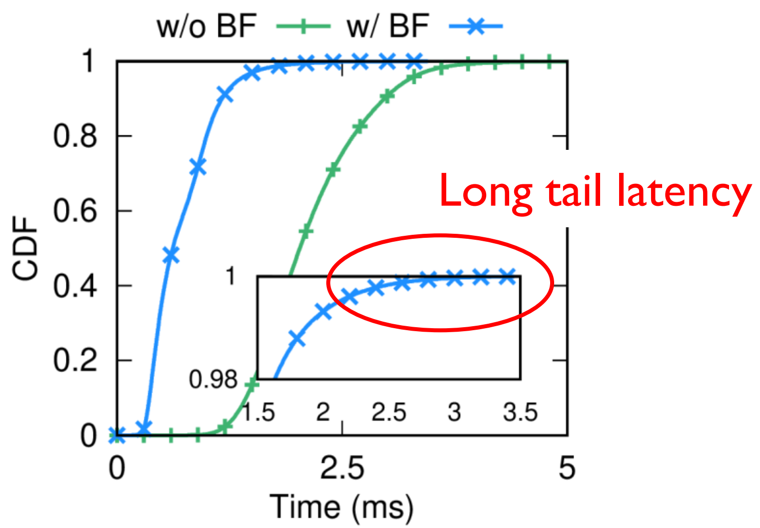
Experiments using LSM-tree-based KV-SSD

- Lightstore*: LSM-tree-based KV-SSD
  - Key-value separation (`Wisckey**`) and Bloom filter (`Monkey***`)
- Benchmark
  - Lightstore: YCSB-LOAD and YCSB-C (Read only), 32B key and 1KB value





Compaction time-breakdown

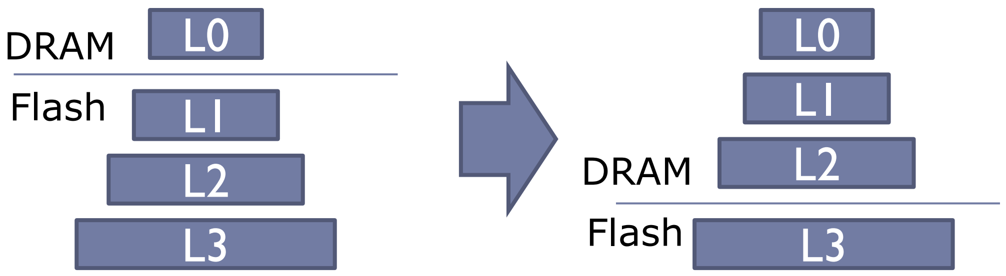


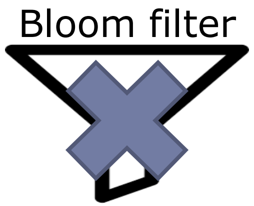
YCSB-C

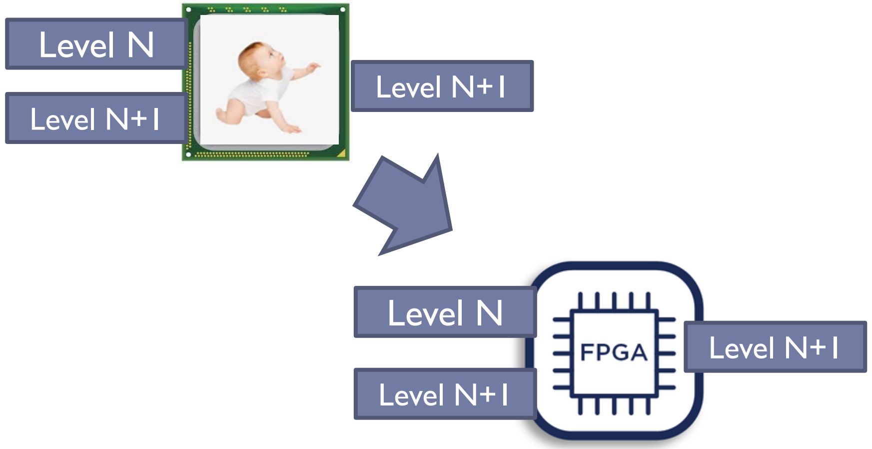
*ASPLOS'19, **FAST'16, ***SIGMOD'17

PinK : New LSM-tree-based KV-SSD

- Long tail latency?
  - Using "Level-pinning"
- CPU overhead?
  - "No Bloom filter"
  - "HW accelerator" for compaction
- I/O overhead?
  - Reducing compaction I/O by level-pinning
  - Optimizing GC by reinserting valid data to LSM-tree







Introduction

PinK

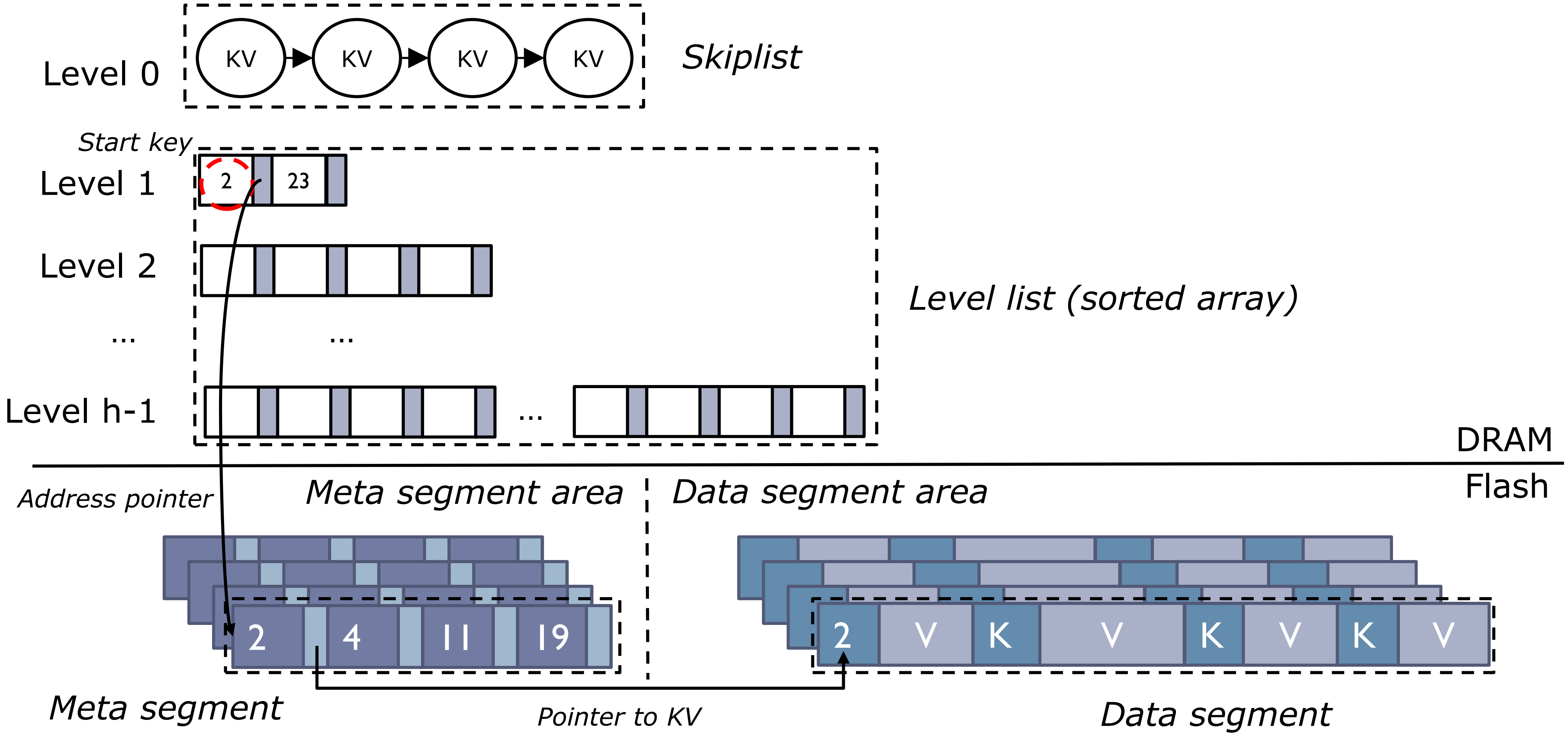
- Overview of LSM-tree in PinK
- Bounding tail latency
- Memory requirement
- Reducing search overhead
- Reducing compaction I/O
- Reducing sorting time

Experiments

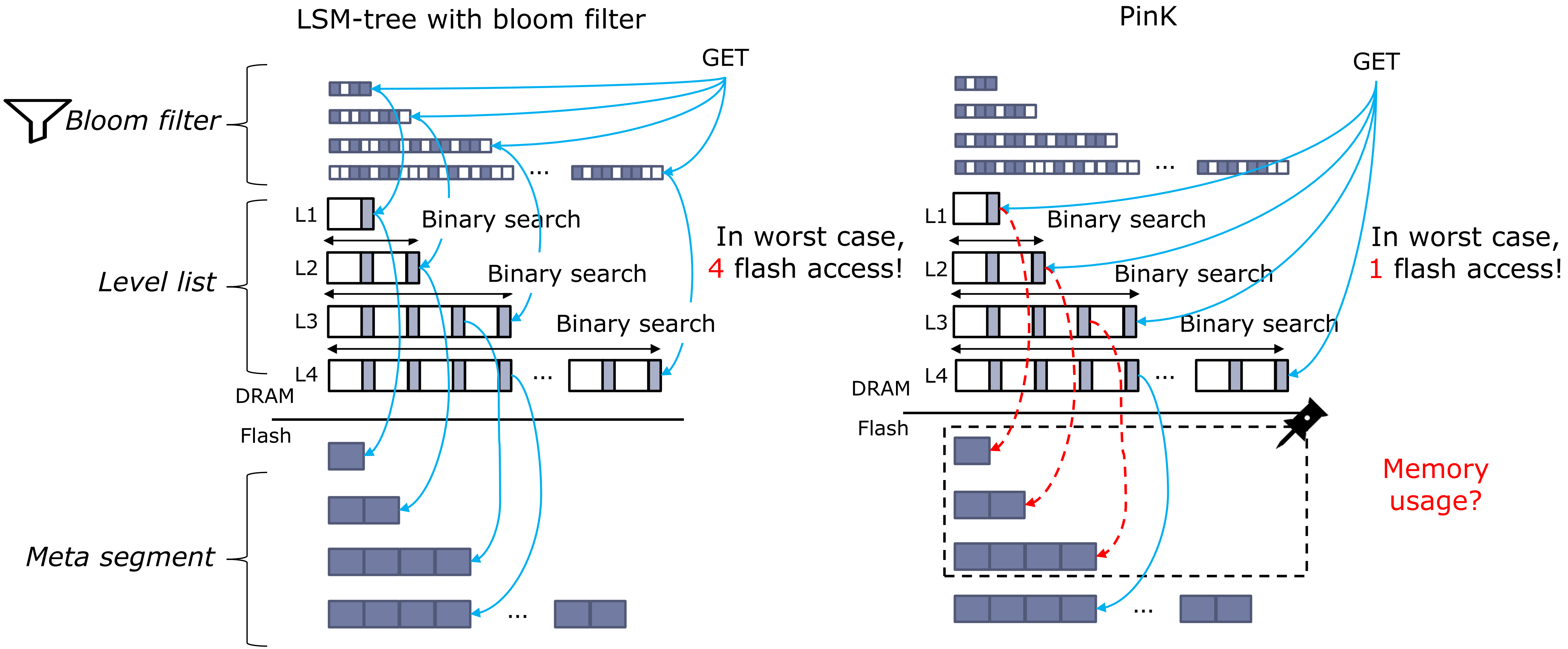
Conclusion

Overview of LSM-tree in PinK

- PinK is based on key-value separated LSM-tree

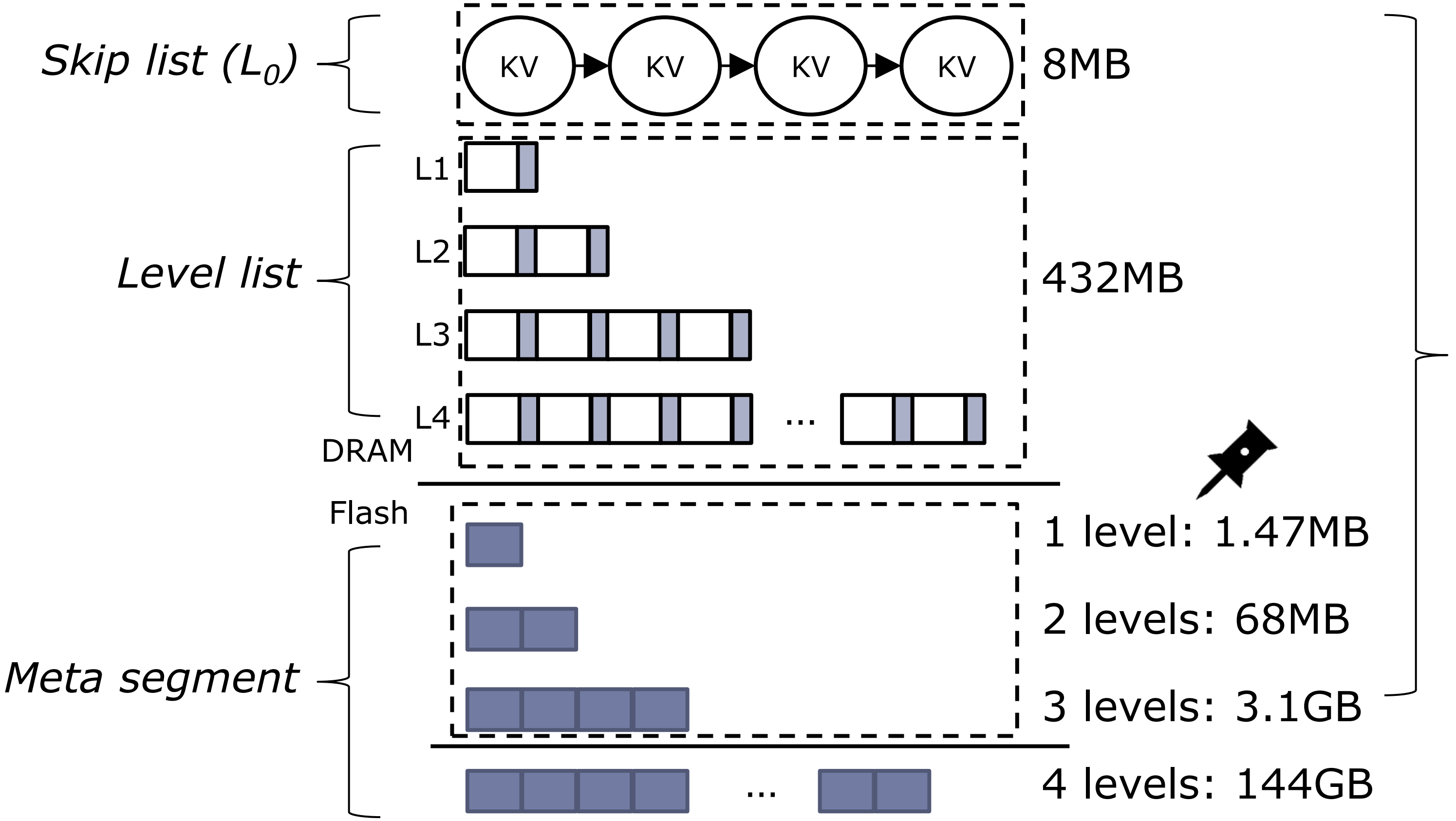


Bounding Tail Latency



Memory Requirement

- 4TB SSD, <u>4GB DRAM</u> (32B key, 1KB value)
  - Total # of levels: 5

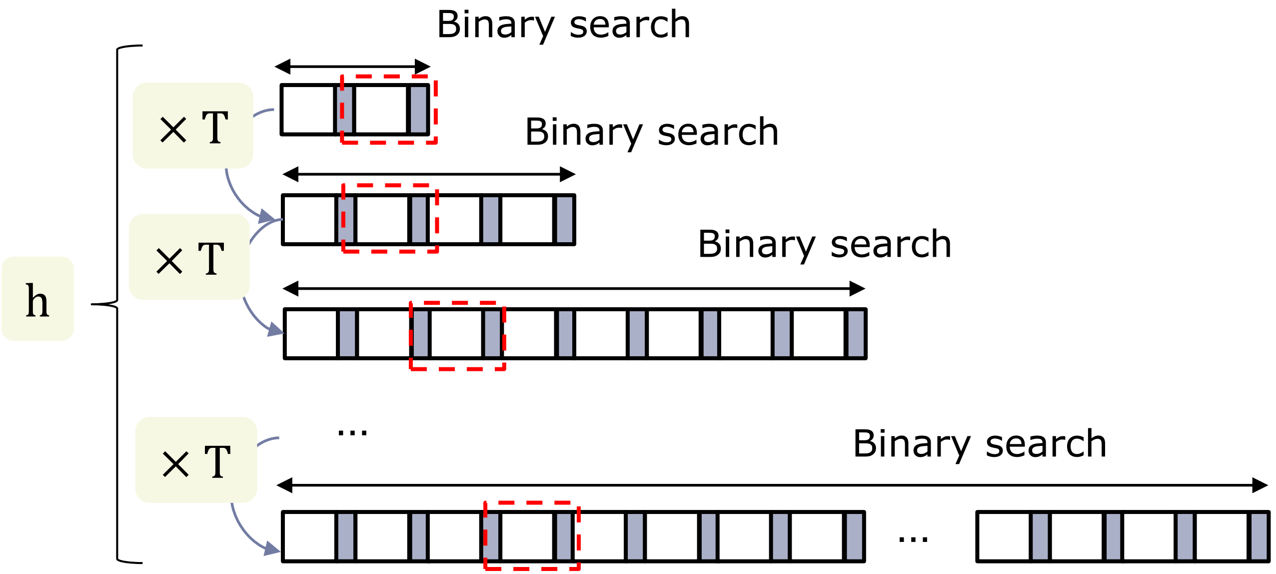


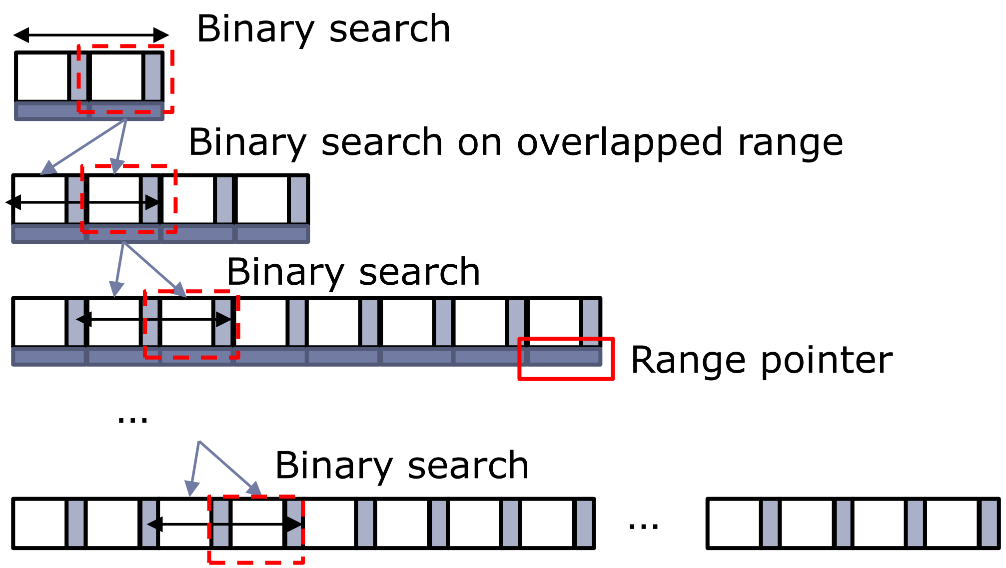
3.5 GB < 4GB

Only one flash access for indexing

Reducing Search Overhead

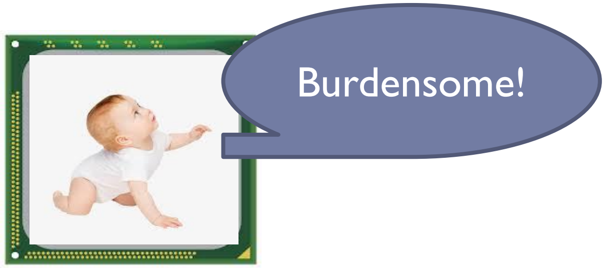
- Fractional cascading





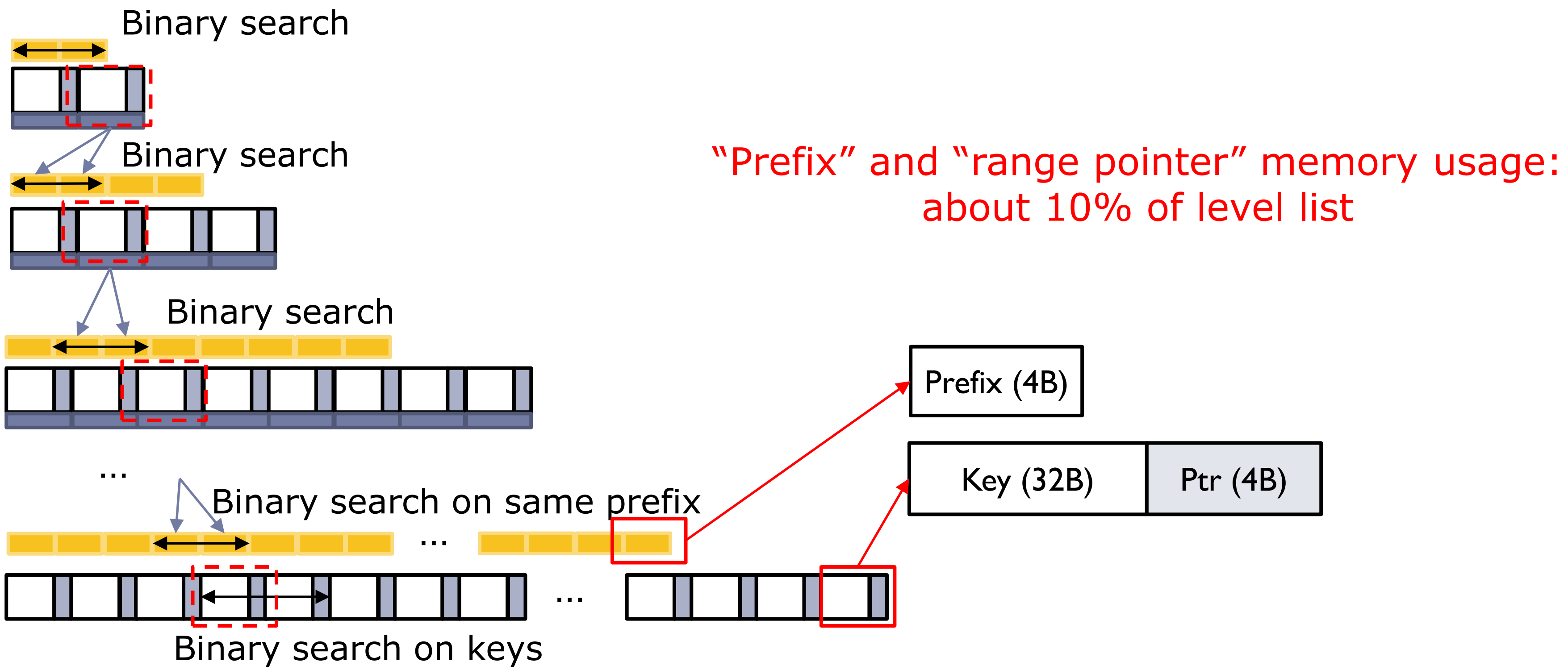
search complexity is $O(h^2 \log(T))$

$O(h \log(T))$



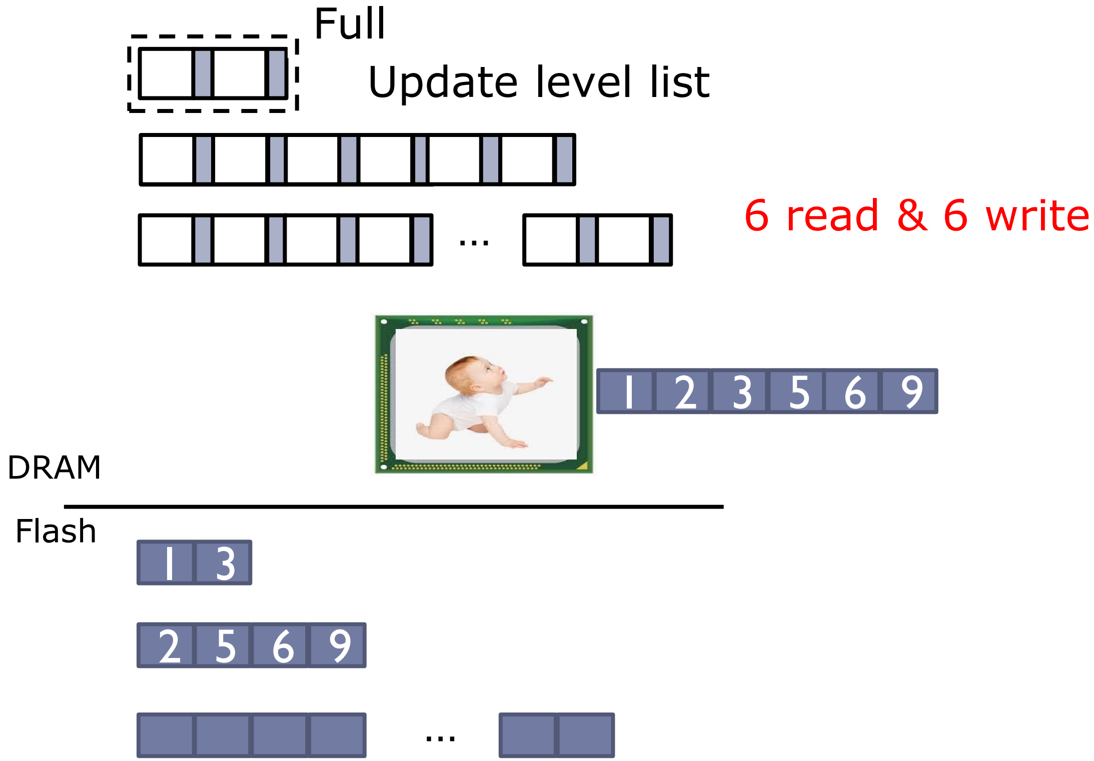
Reducing Search Overhead

- Prefix
  - Less compare overhead
  - Cache efficient search

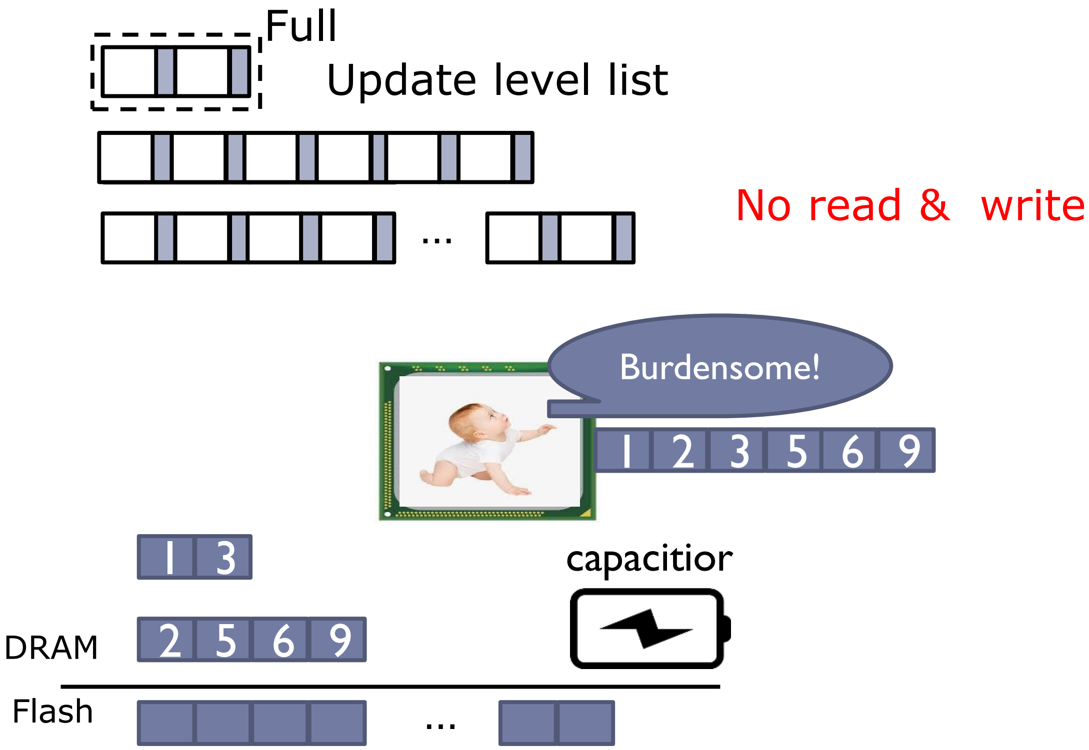


Reducing Compaction I/O

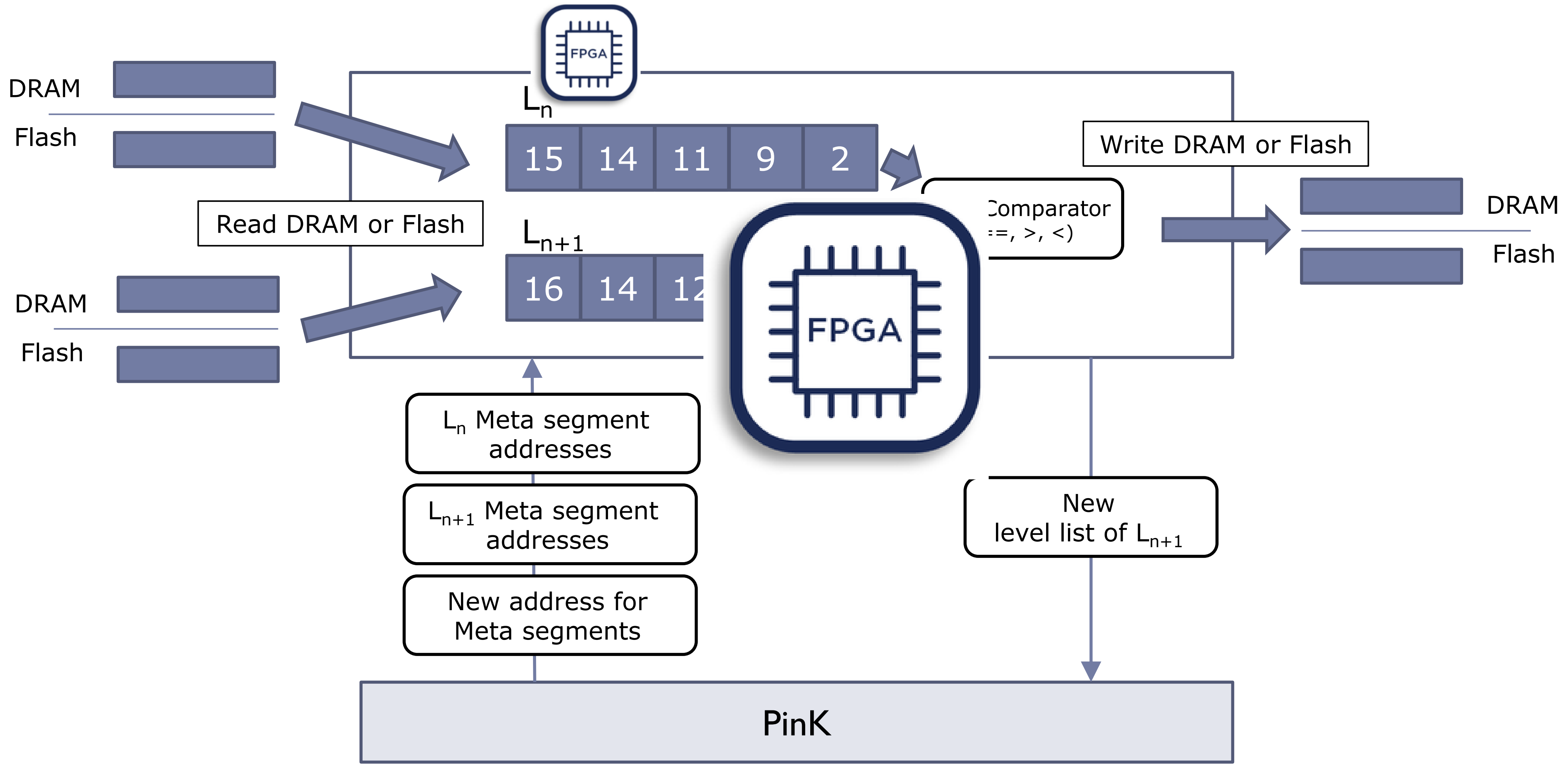
PinK without level-pinning



PinK with level-pinning



Reducing Sorting Time

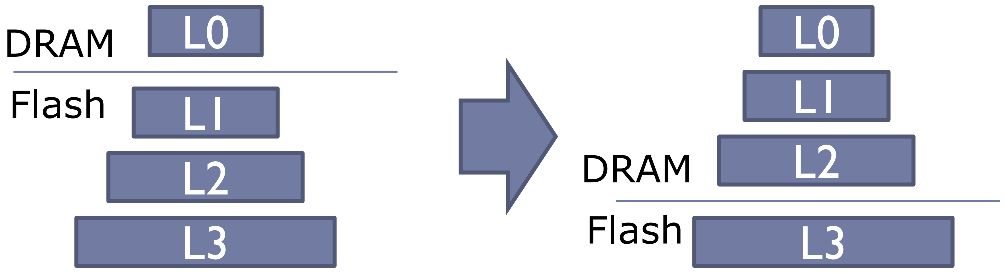


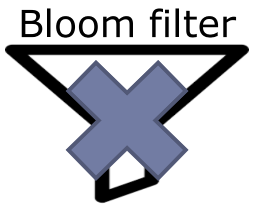
PinK Summary

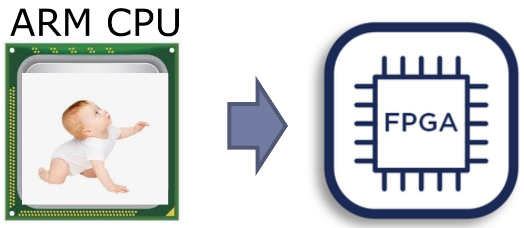
- Long tail latency?
  Using level-pinning
- CPU overhead?
  Removing Bloom filter
  Optimizing binary search
  Adopting HW accelerator
- I/O overhead?
  Reducing compaction I/O

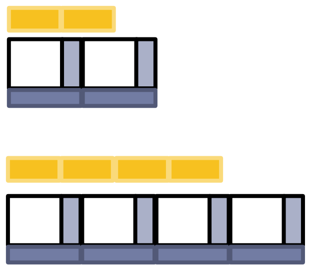
  Optimizing GC by
  reinserting valid data to LSM-tree

Please refer to the paper!









Introduction

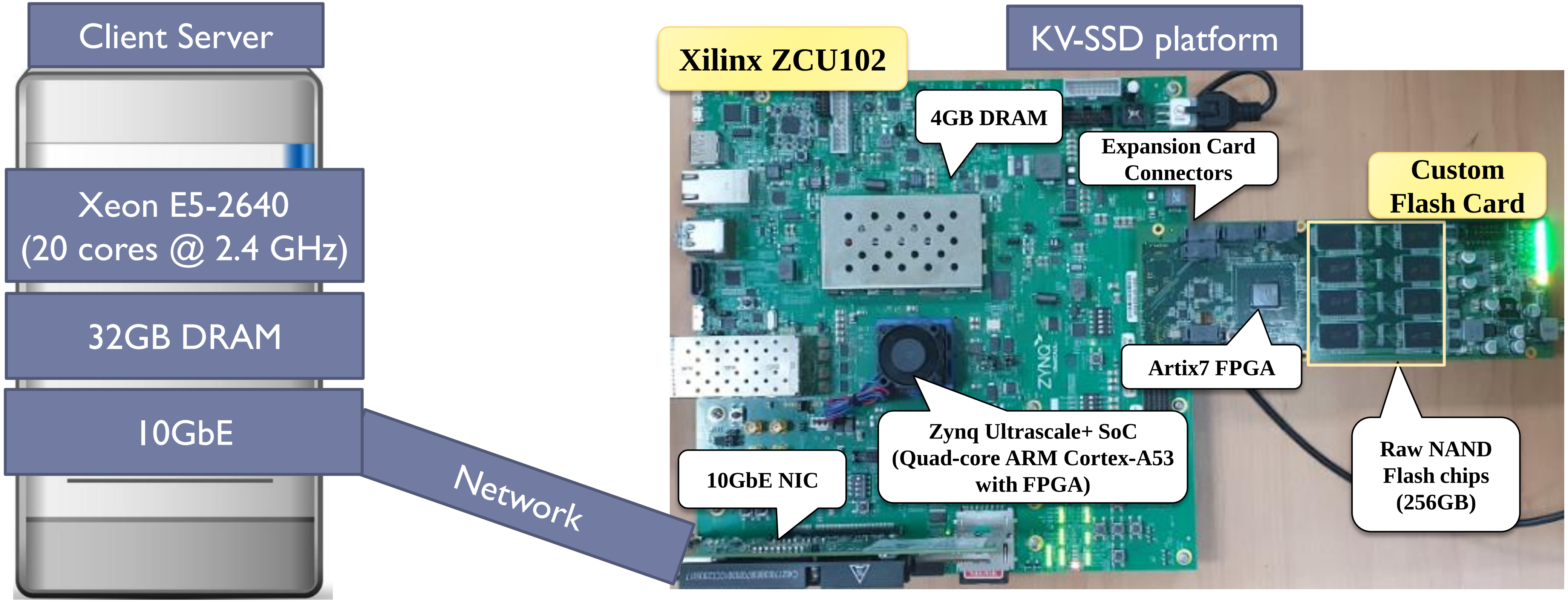
PinK

Experiments

Conclusion

Custom KV-SSD Prototype and Setup

- All algorithms for KV-SSD were implemented on ZCU102 board
  - For fast experiments: 64GB SSD, 64 MB DRAM (0.1% of NAND capacity)



Benchmark Setup

- YCSB: 32B key, 1KB value

	Load	A	B	C	D	E	F
R:W ratio	0:100	50:50	95:5	100:0	95:5	95:5	50:50(RMW)
Query type	Point					Range read	Point
Request distribution	Uniform	Zipfian			Latest (Highest locality)	Zipfian	

- Two phases
  - Load: issue unique 44M KV pairs (44GB, 70% of total SSD)
  - Run: issue 44M KV pairs following workload description

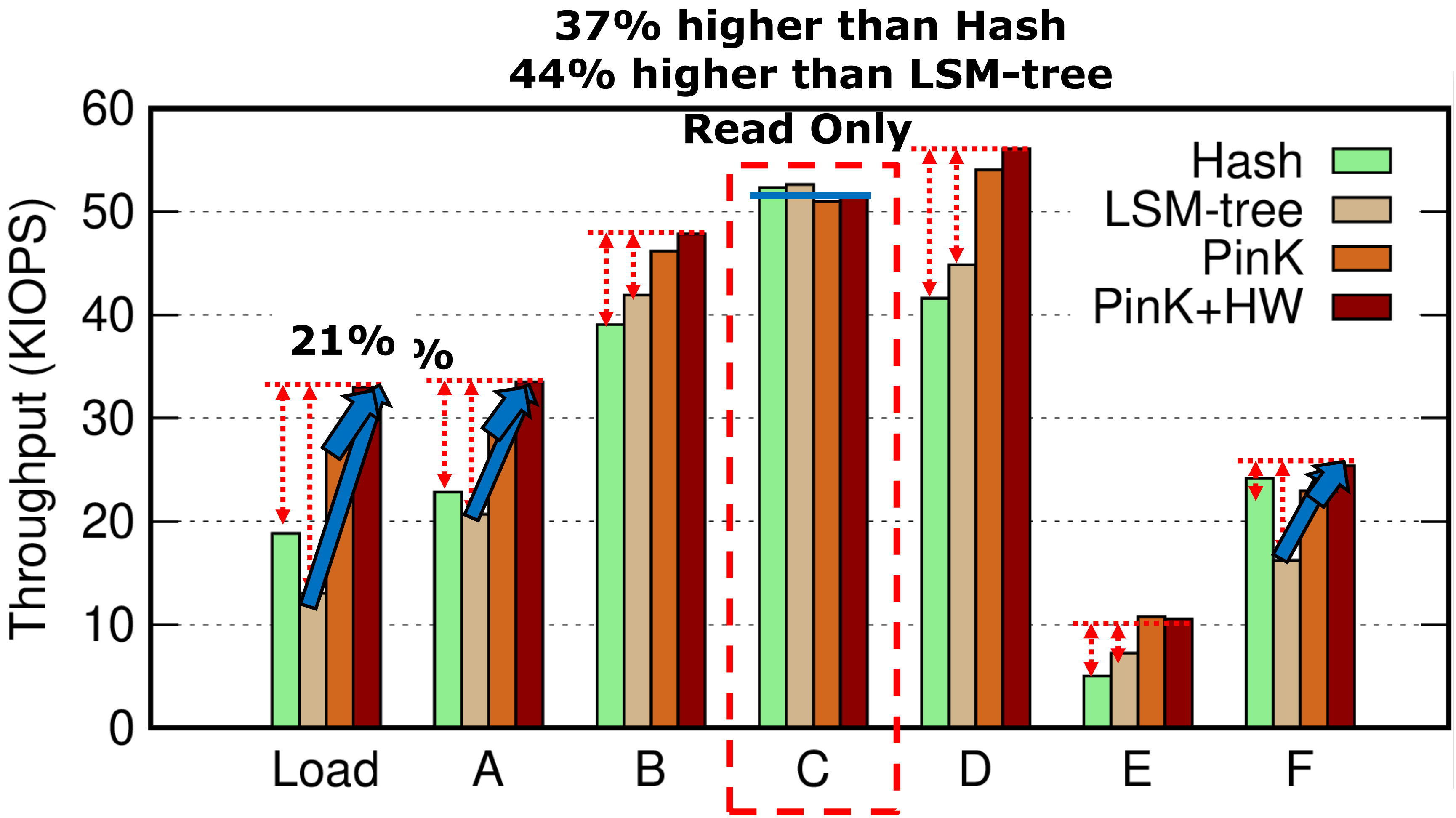
Testing Algorithms

- Hash
  - 8-bit signature: total 320MB buckets
- LSM-tree
  - The conventional LSM-tree implementation based on Lightstore*
  - Total 5 levels (1~4 level in flash)
- PinK
  - Total 5 levels (pinning top 3 levels, one level in flash)
- PinK+HW
  - Using HW accelerator for compaction based on PinK

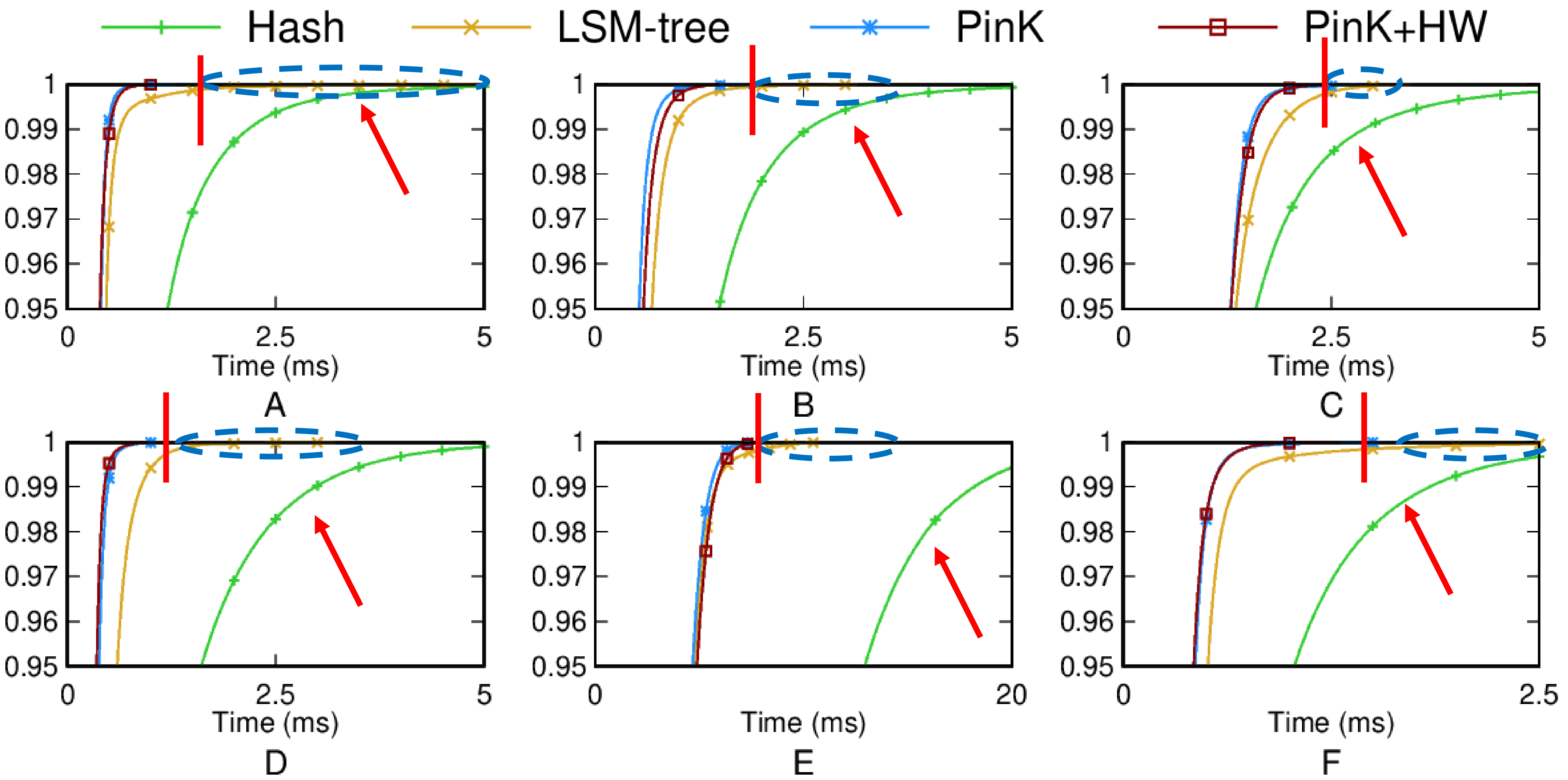
	Hash	LSM-tree	PinK, PinK+HW
64MB DRAM	LRU bucket caching (64MB)	Level list (9MB) Bloom filter (55MB)	Level list + prefix, range (10MB) Level-pinning (54MB)

*ASPLOS'19

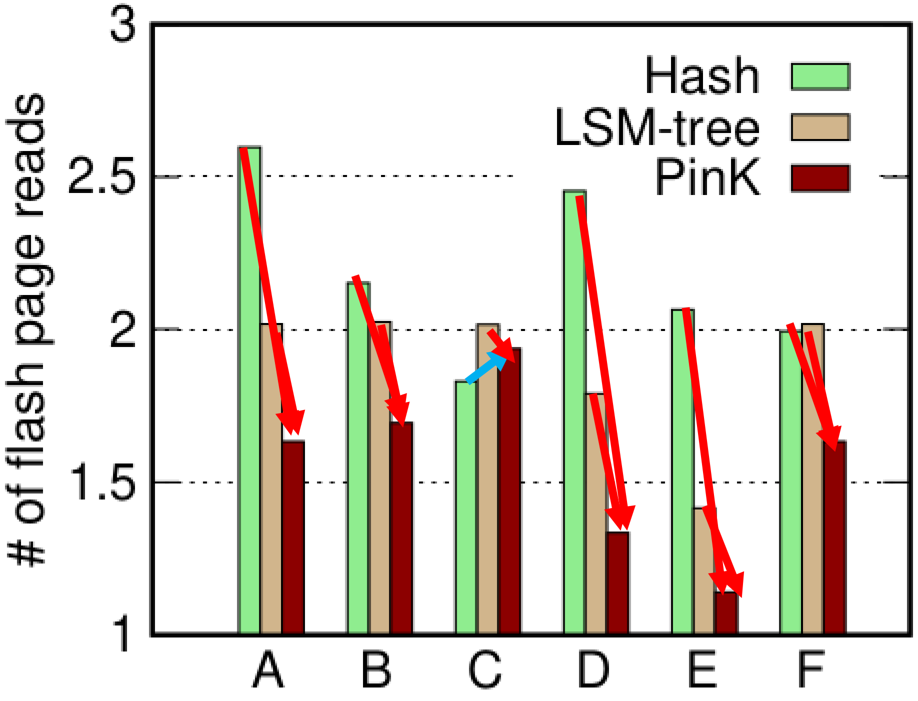
Experiment: Throughput



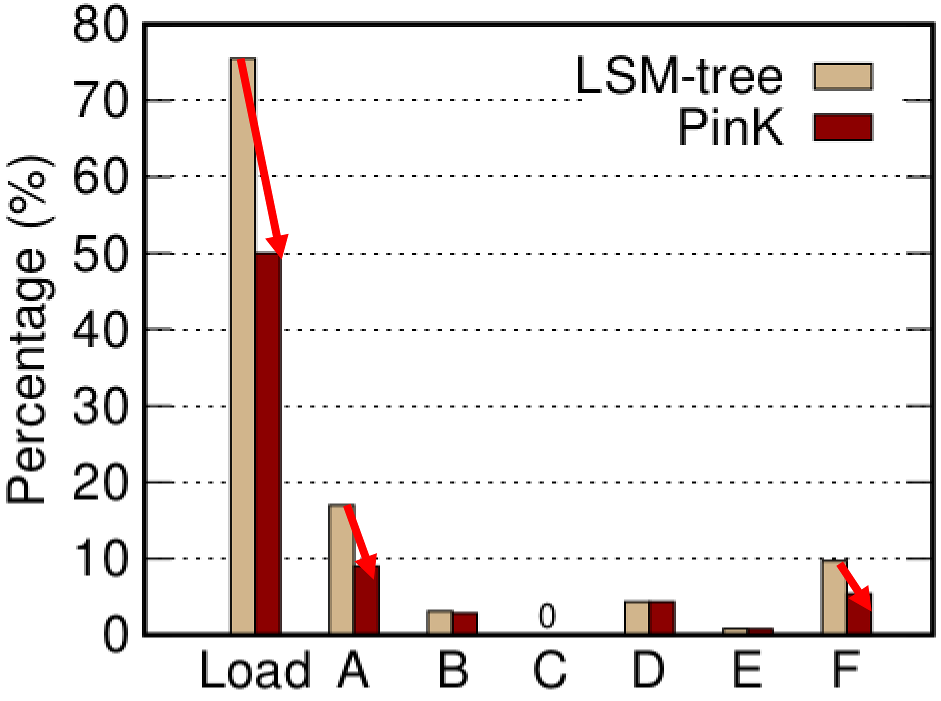
Experiment: Latency



Experiment: Impact of Level-pinning



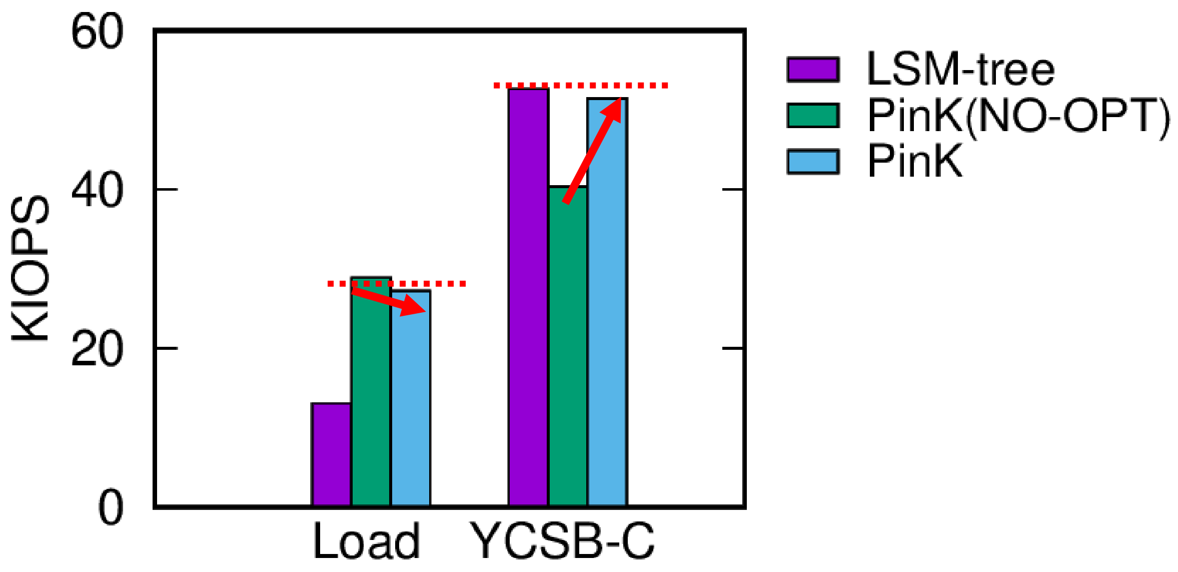
(a) Flash page reads per query

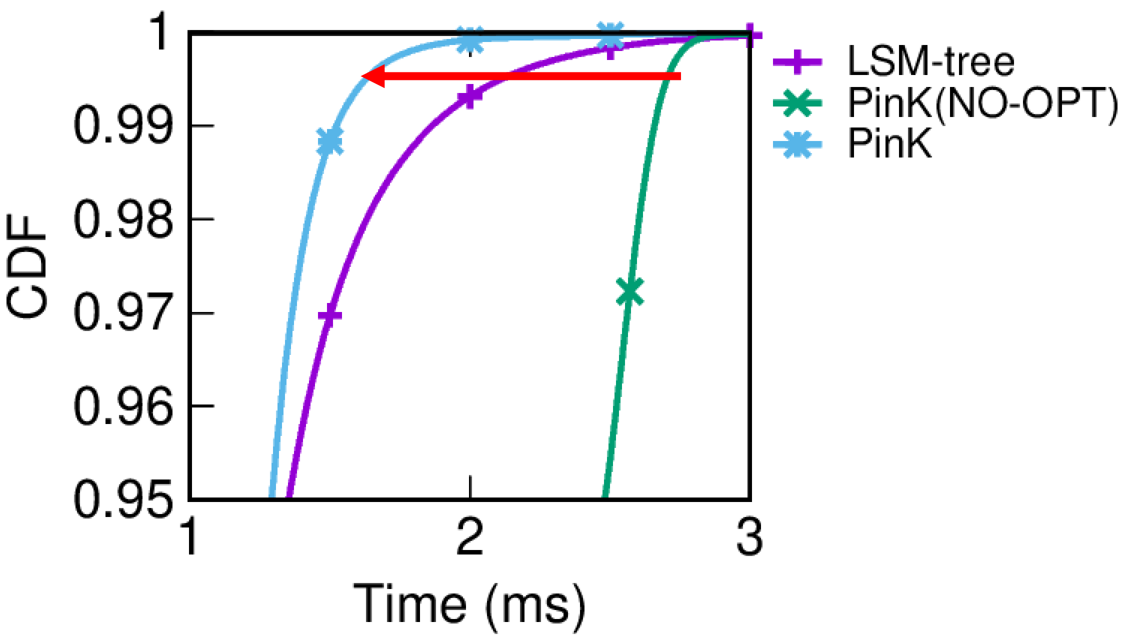


(b) Compaction I/Os

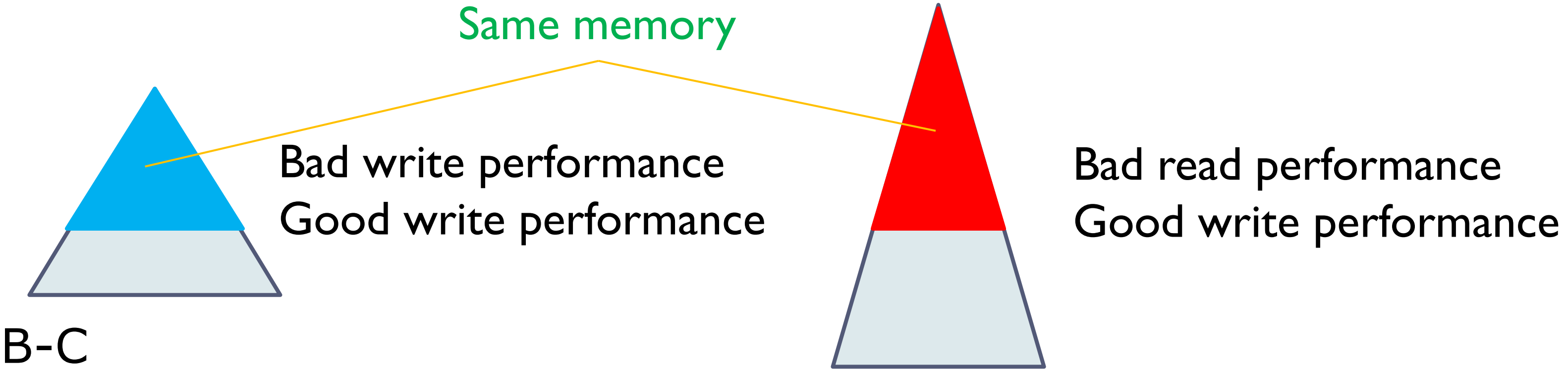
Experiment: Search Optimization

- Settings
  - PinK (NO-OPT): PinK without prefix and range pointer
  - Benchmark: YCSB-Load and YCSB-C





Experiment: Level-pinning on Higher LSM-tree



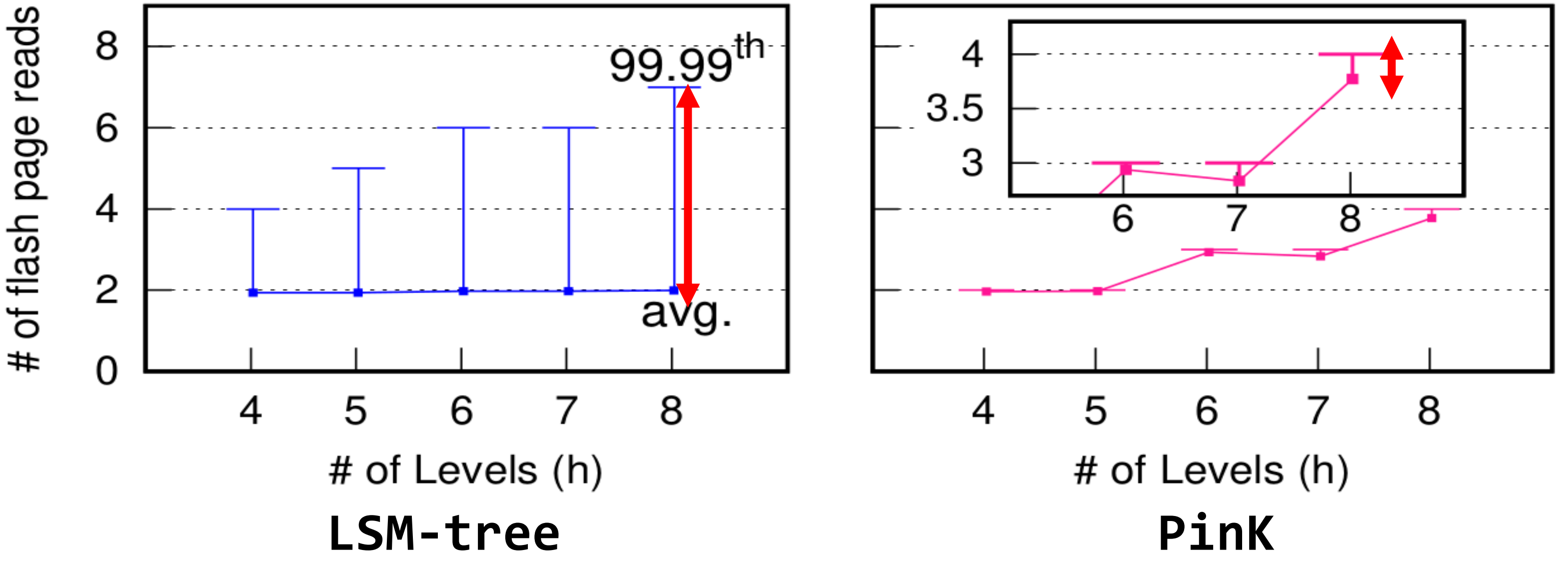
- Benchmark: YCSB-C
  - # of total levels: 4 ~ 8

- PinK

  # of levels: 4, 5
  - Unpinned-level: 1

  # of levels: 6, 7
  - Unpinned-level: 2

  # of levels:8
  - Unpinned-level: 3



Introduction

PinK

Experiments

Conclusion

Conclusion

- Since the conventional KV-SSD's algorithms did not consider the embedded system's limitations well, they have suffered from long tail latency and throughput degradation
- PinK
  - Pinning KV indices of top levels of LSM-tree to DRAM to reduce latency
  - Using HW accelerator for compaction sorting
- Benefits
  - 99 percentile tail latency: 73%
  - Average latency: 42%
  - Throughput : 37%

Thank You !

Junsu Im (junsu_im@dgist.ac.kr)