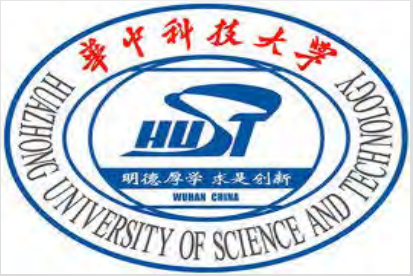
A Write-friendly Hashing Scheme for Non-volatile Memory Systems

Pengfei Zuo and Yu Hua

Huazhong University of Science and Technology, China



Non-volatile Memory

- NVMs are expected to replace DRAM and SRAM

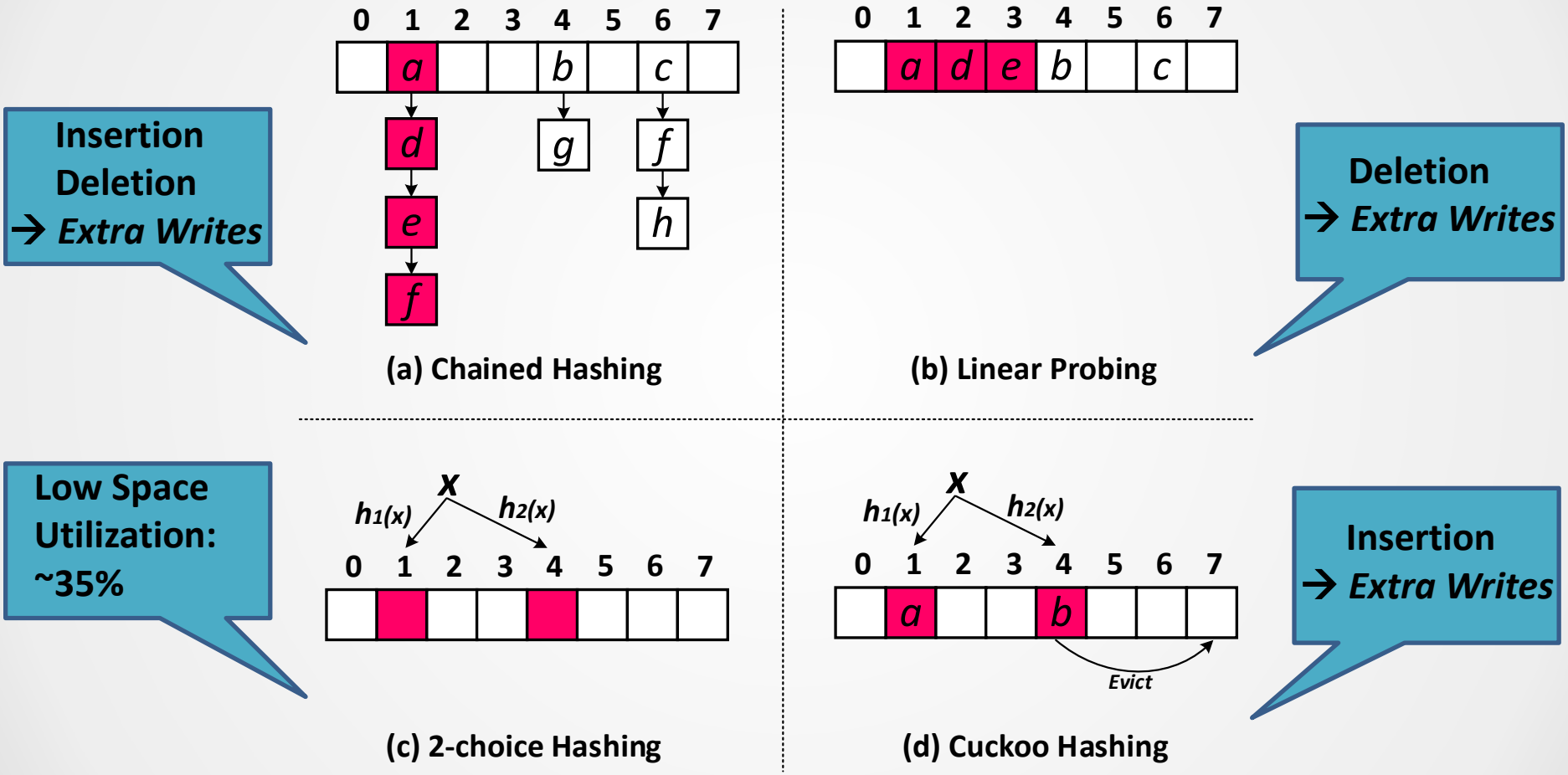
	SRAM	DRAM	PCM	RRAM	STT-RAM
Non-volatile	N	N	Y	Y	Y
Read (ns)	1	10	20~70	10	2~20
Write (ns)	1	10	150~220	50	5~35
Standby Power	High	High	Low	Low	Low
Scalability (nm)	20	20	5	11	32
Endurance (10^N)	> 15	> 15	7~8	8~10	12~15

- NVMs vs. DRAM & SRAM
  - ✓ No-volatile, high scalability, and low standby power
  - X Limited endurance and asymmetric properties

Rethinking Data Structures on NVMs

- *How could in-memory and in-cache data structures be modified to efficiently adapt to NVMs?*
- Previous work mainly focuses on tree-based structures
  - ✓ CDDS-tree (FAST 2011)
  - ✓ NV-tree (FAST 2015)
  - ✓ wB+-tree (VLDB 2015)
  - ✓ FP-tree (SIGMOD 2016)
  - ✓ Write Optical Radix Tree (FAST 2017)
- Hash tables are also widely used in main memory and caches
  - ✓ Main memory database
  - ✓ In-memory key-value store, e.g., Memcached, Redis
  - ✓ In-cache index (ICS 2014, MICRO 2015)

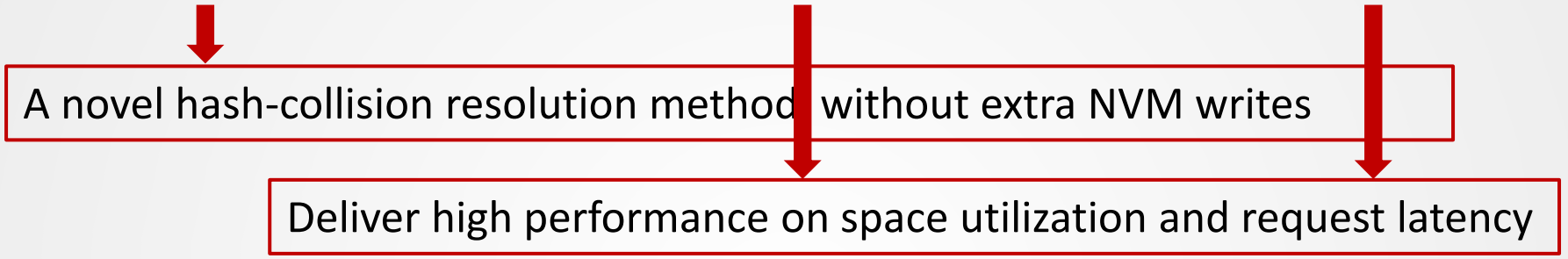
Existing Hashing Schemes on NVMs



- **Our Design Goals**
  - ✓ **Minimize NVM writes while ensuring high performance**

Our Scheme: Path Hashing

✓ **Position Sharing** ✓ **Double-path Hashing** ✓ **Path Shortening**

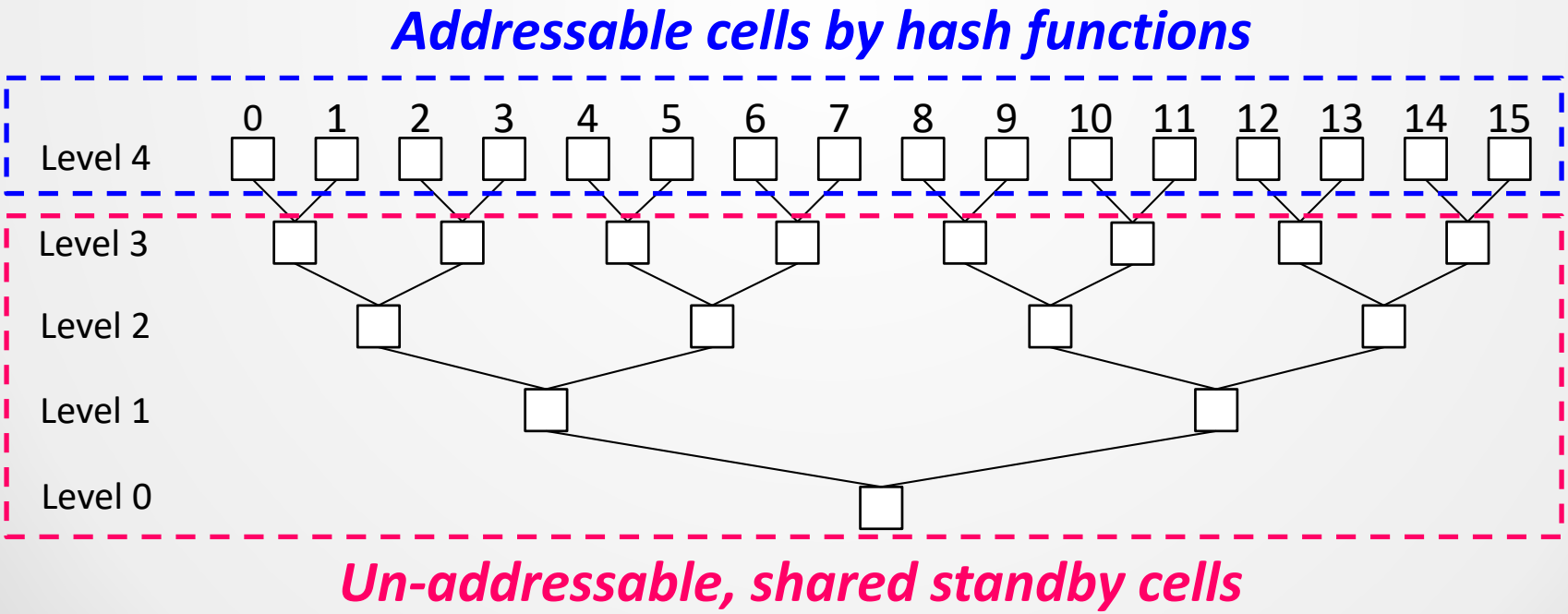


Our Scheme: Path Hashing

✓ **Position Sharing** ✓ Double-path Hashing ✓ Path Shortening

A novel hash-collision resolution method resulting in no extra NVM writes

Deliver high performance on space utilization and request latency



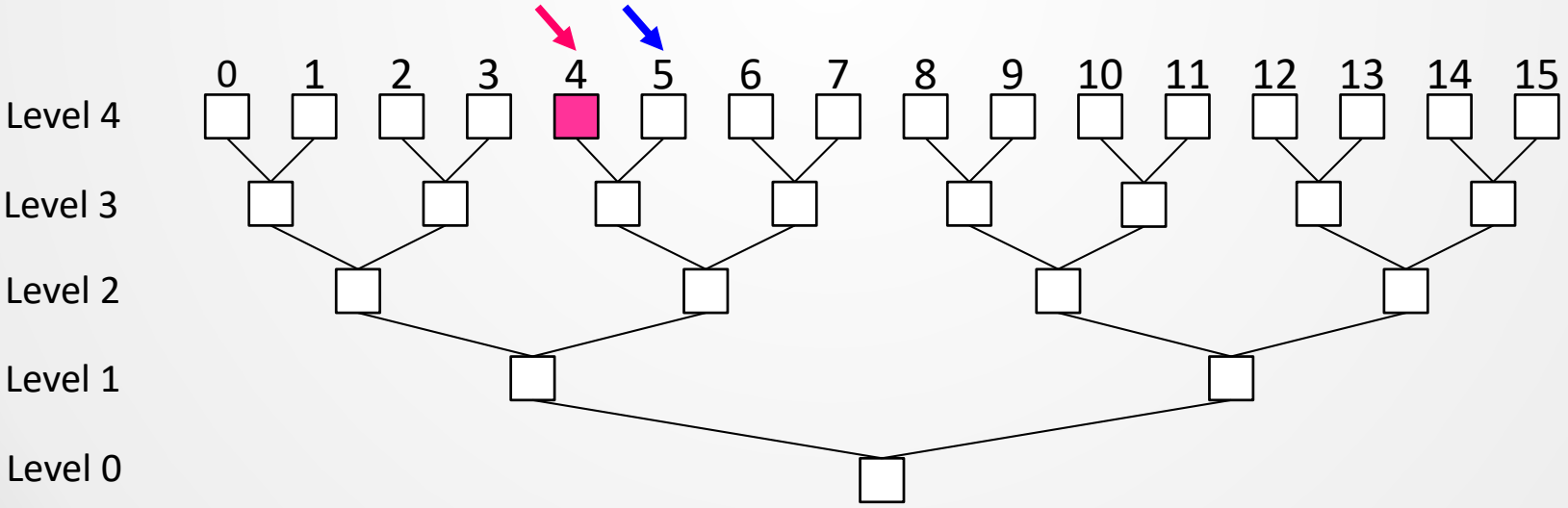
Our Scheme: Path Hashing

✓ **Position Sharing** ✓ Double-path Hashing ✓ Path Shortening

A novel hash-collision resolution method resulting in no extra NVM writes

Deliver high performance on space utilization and request latency

Insertion and deletion without extra modifications and data movements



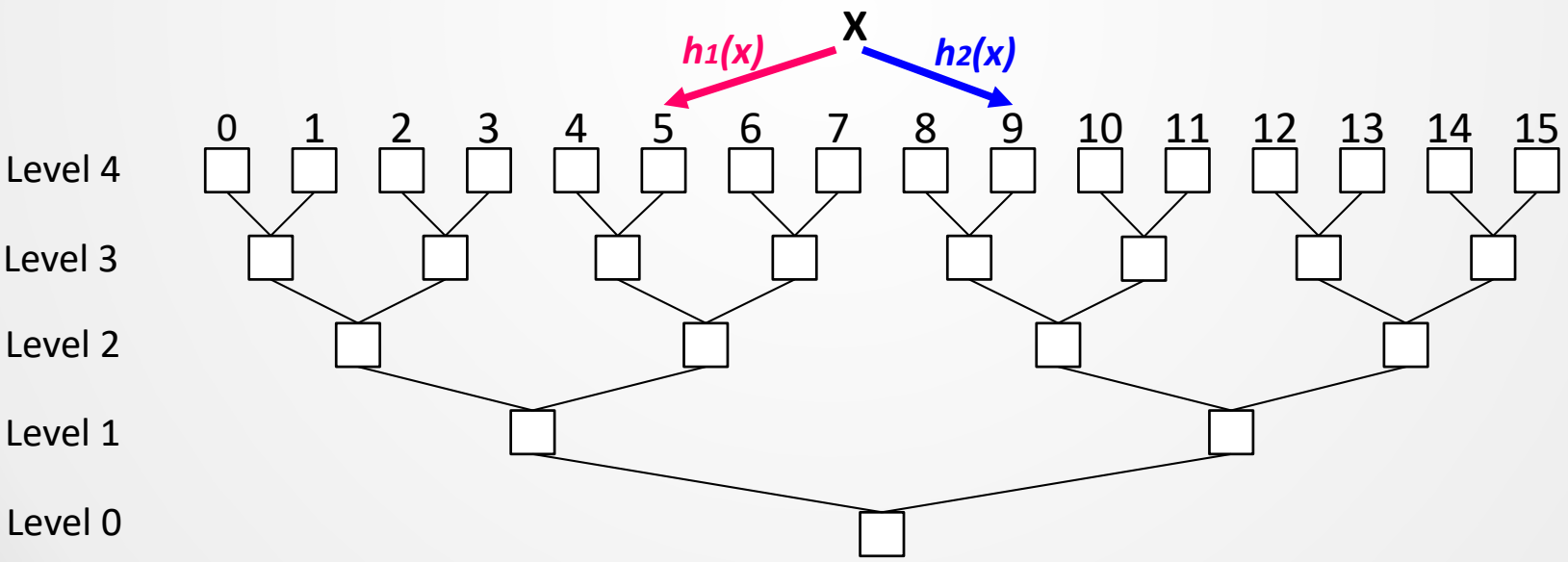
Problem: One path can only deal with at most *L* hash collisions

Our Scheme: Path Hashing

✓ Position Sharing ✓ **Double-path Hashing** ✓ Path Shortening

A novel hash-collision resolution method resulting in no extra NVM writes

Deliver high performance on space utilization and request latency



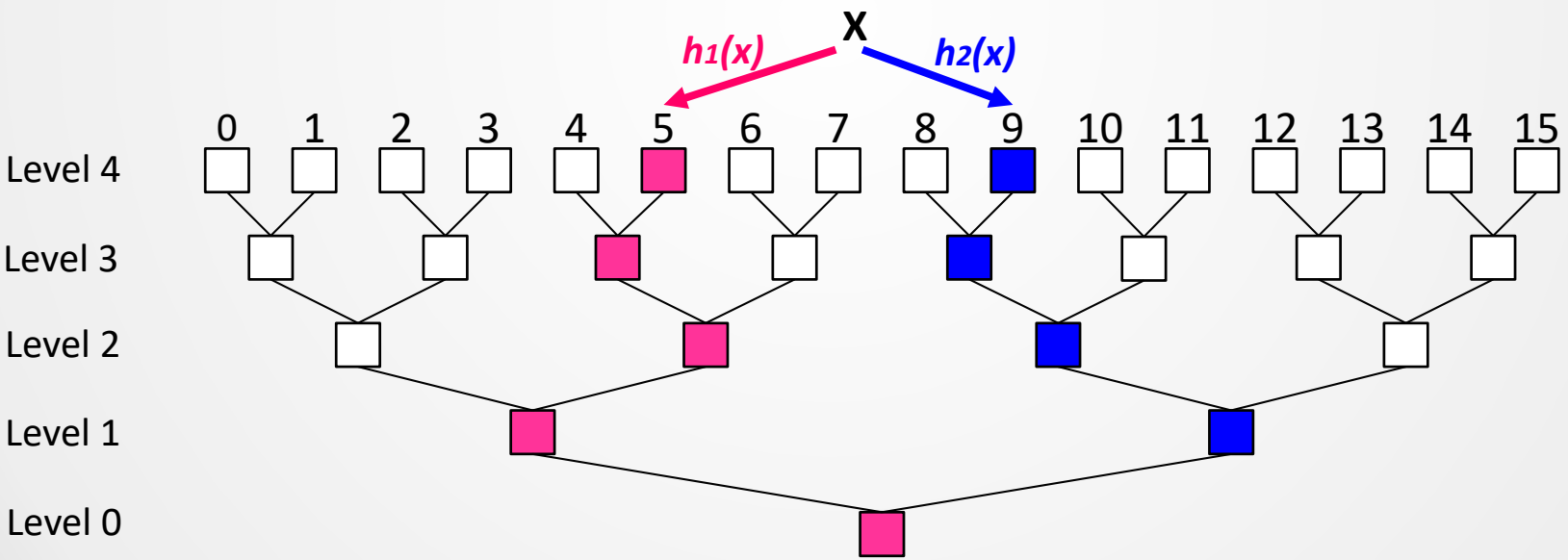
Using two different hash functions to compute two paths → high space utilization

Our Scheme: Path Hashing

✓ Position Sharing ✓ **Double-path Hashing** ✓ Path Shortening

A novel hash-collision resolution method resulting in no extra NVM writes

Deliver high performance on space utilization and request latency



Problem: Each query may probe many nodes in a high tree

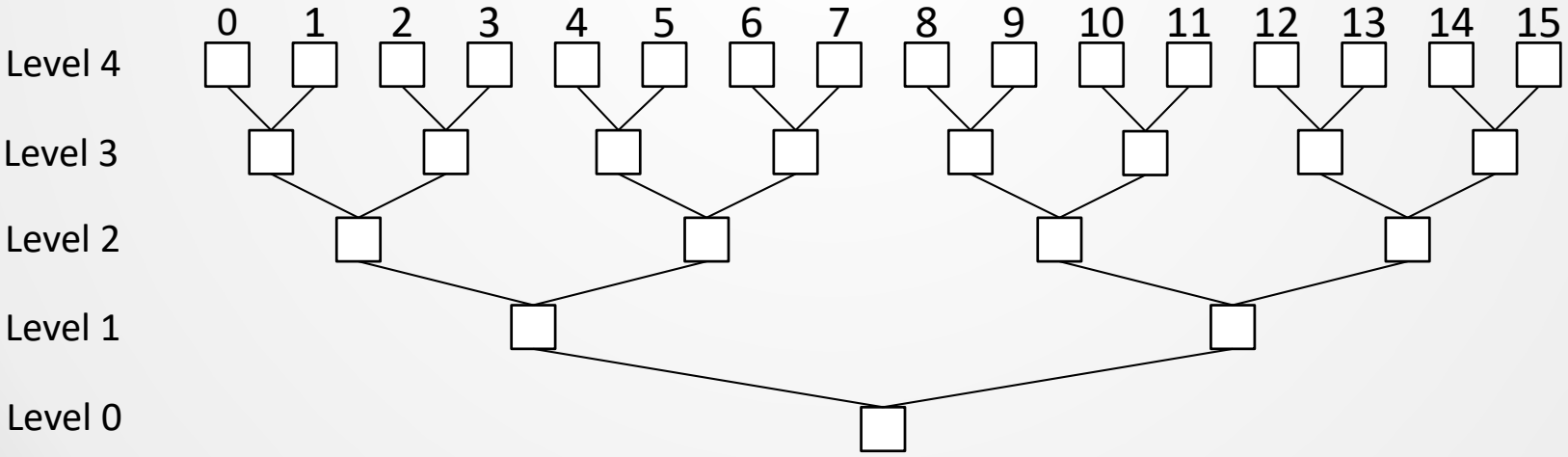
Our Scheme: Path Hashing

✓ Position Sharing ✓ Double-path Hashing ✓ **Path Shortening**

A novel hash-collision resolution method resulting in no extra NVM writes

Deliver high performance on space utilization and request latency

Observation: The bottom levels provide a few standby positions while increasing the length of the read path.



Path Shortening: Removing multiple levels in the bottom.

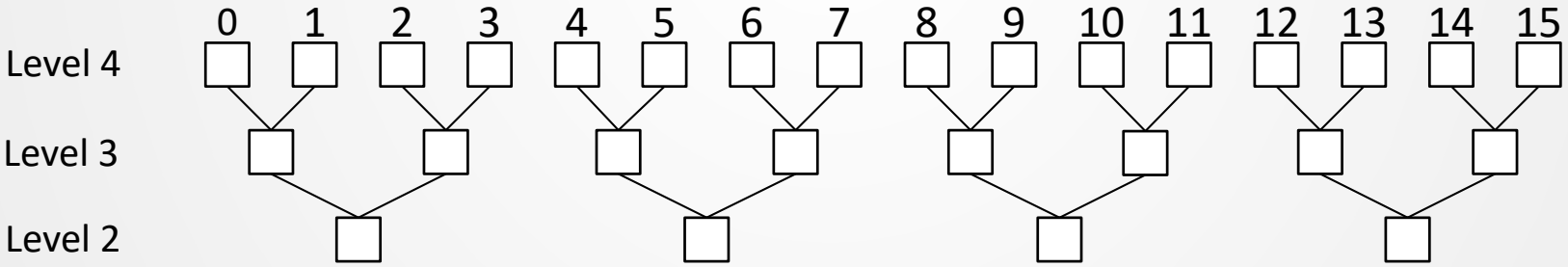
Our Scheme: Path Hashing

✓ Position Sharing ✓ Double-path Hashing ✓ **Path Shortening**

A novel hash-collision resolution method resulting in no extra NVM writes

Deliver high performance on space utilization and request latency

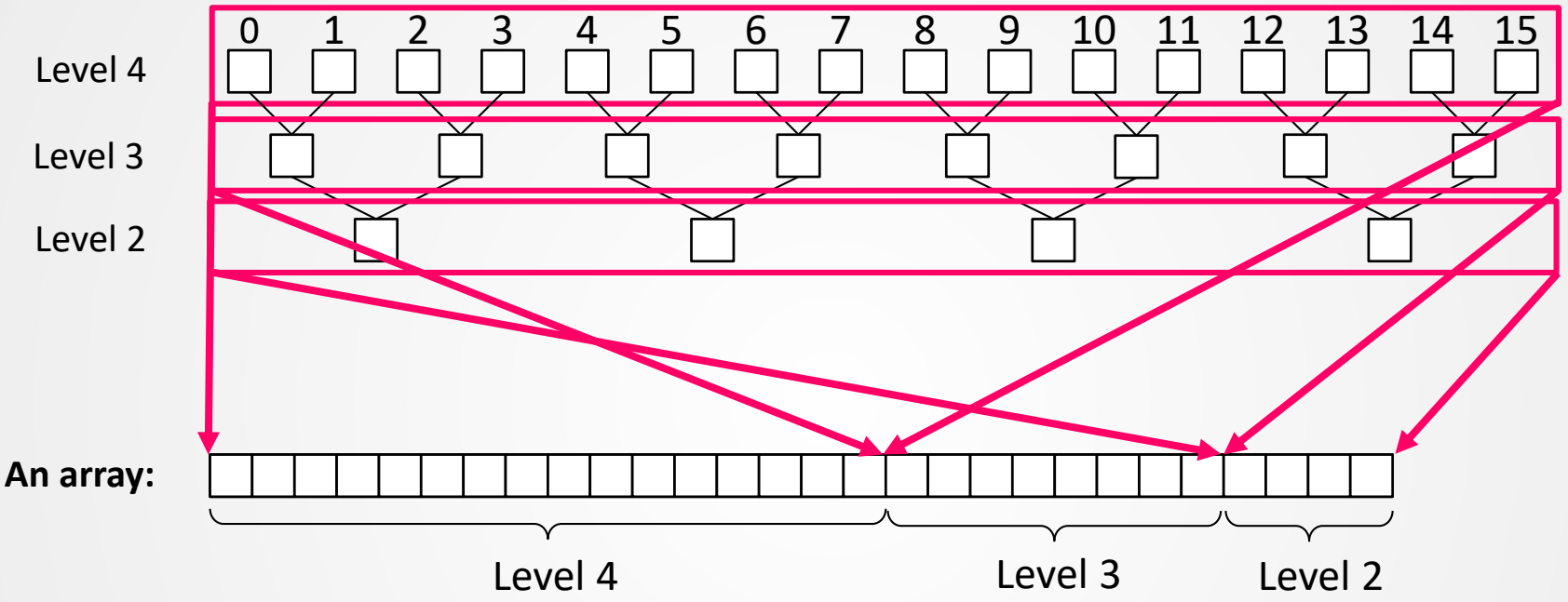
Observation: The bottom levels provide a few standby positions while increasing the length of the read path.



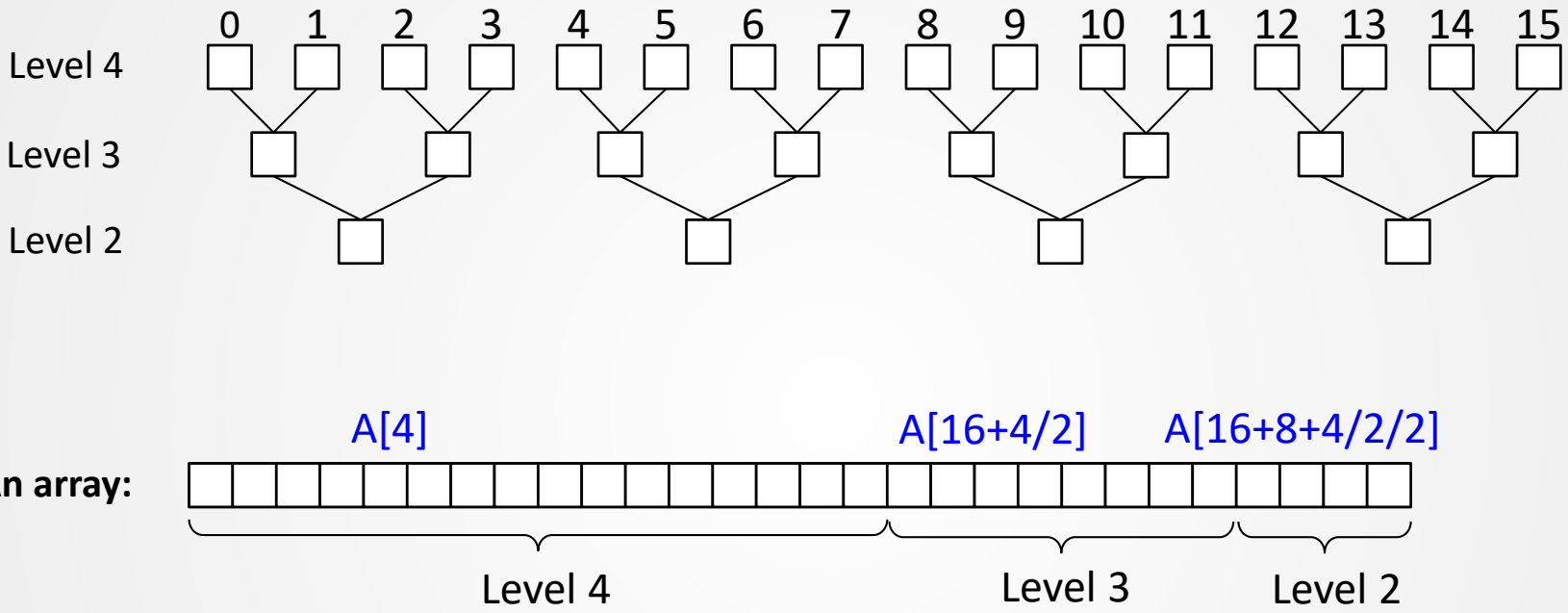
Evaluation: Reserving a small part of levels can also achieve a high space tilization.

Path Shortening: Removing multiple levels in the bottom.

Physical Storage Structure of Path Hashing



Physical Storage Structure of Path Hashing



- No pointers
- The nodes in a path can be accessed in parallel for insertion, query and deletion

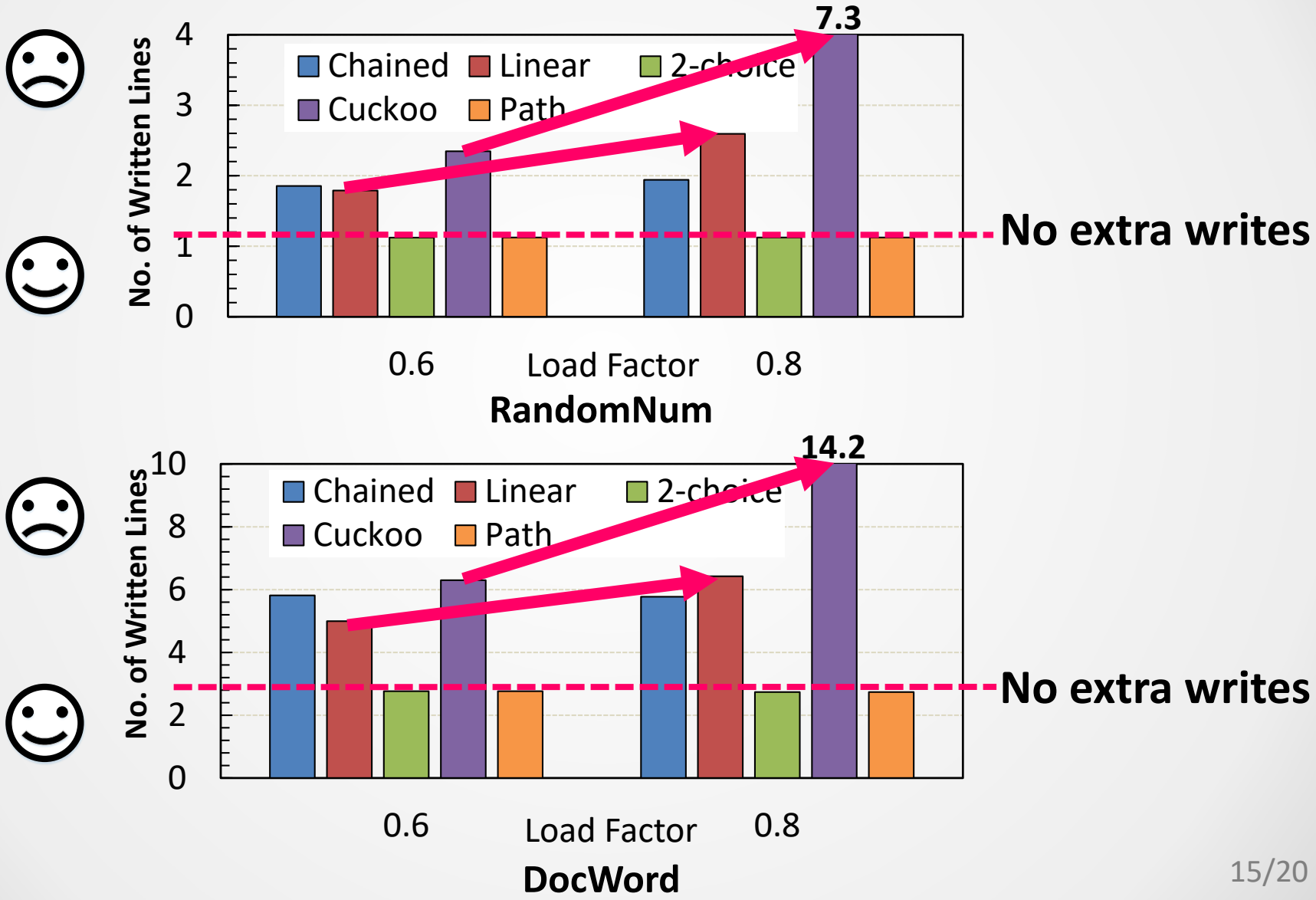
Experimental Configurations

- Gem5: a full system simulator
- NVMain: a main memory simulator for NVMs

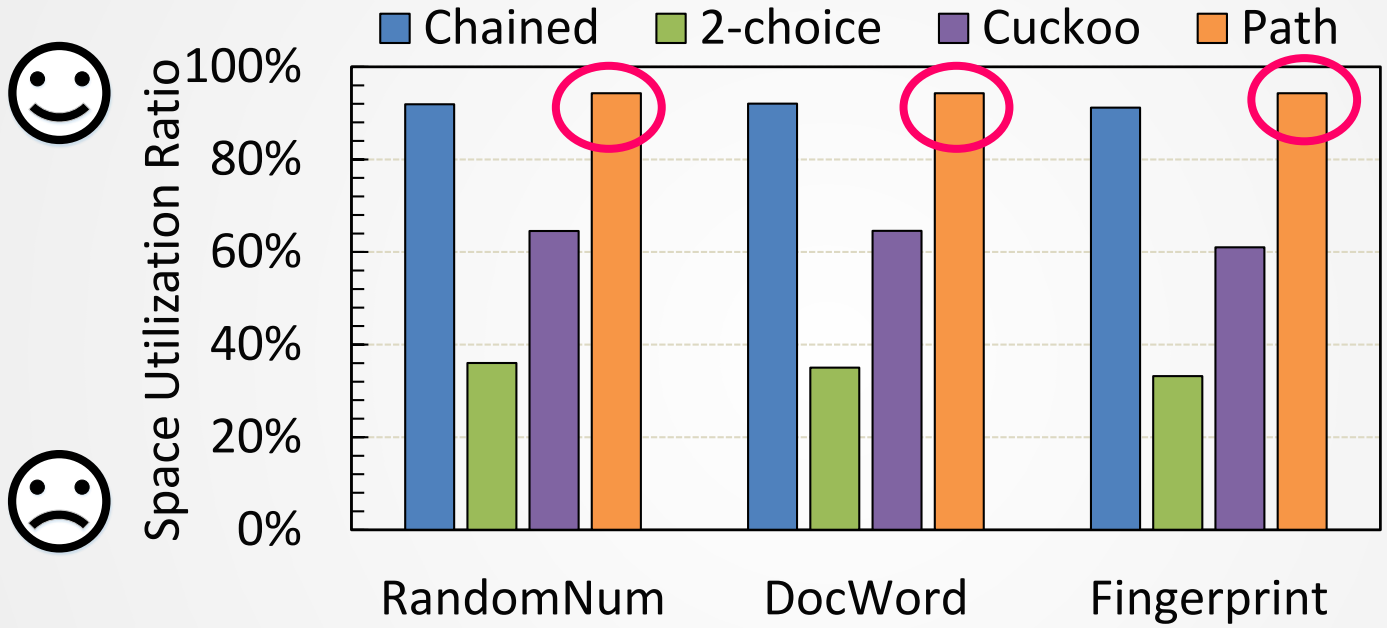
Processor and Cache	
CPU	4 cores, X86-64 processor, 2 GHz
Private L1 cache	32 KB (each core), 2-way, LRU, 2-cycle latency
Shared L2 cache	4 MB, 8-way, LRU, 20-cycle latency
Shared L3 cache	32 MB, 8-way, LRU, 50-cycle latency
Memory Controller	FCFRFS
Main Memory using PCM	
Capacity	16 GB
Read latency	75 ns
Write latency	150 ns

- Datasets: Random Number, Document Word, Fingerprint

NVM Writes

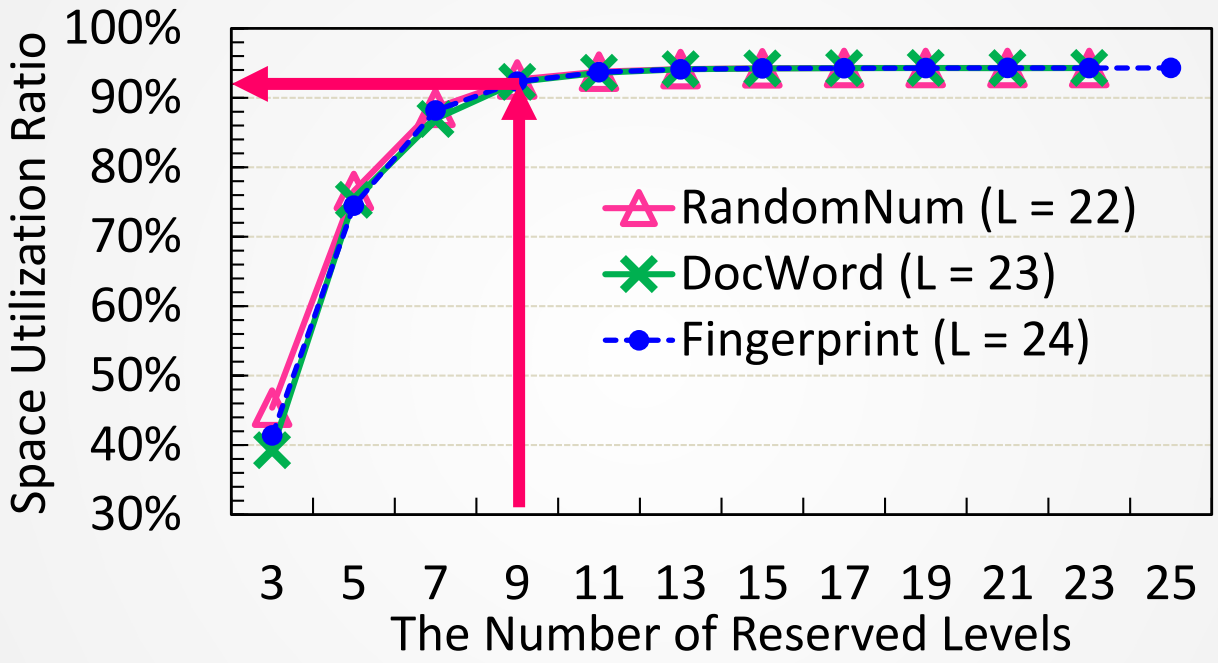

No. of Written Lines
4
3
2
1
0
Chained
Linear
2-choice
Cuckoo
Path
7.3
No extra writes
0.6
Load Factor
0.8
RandomNum
No. of Written Lines
10
8
6
4
2
0
Chained
Linear
2-choice
Cuckoo
Path
14.2
No extra writes
0.6
Load Factor
0.8
DocWord

Space Utilization



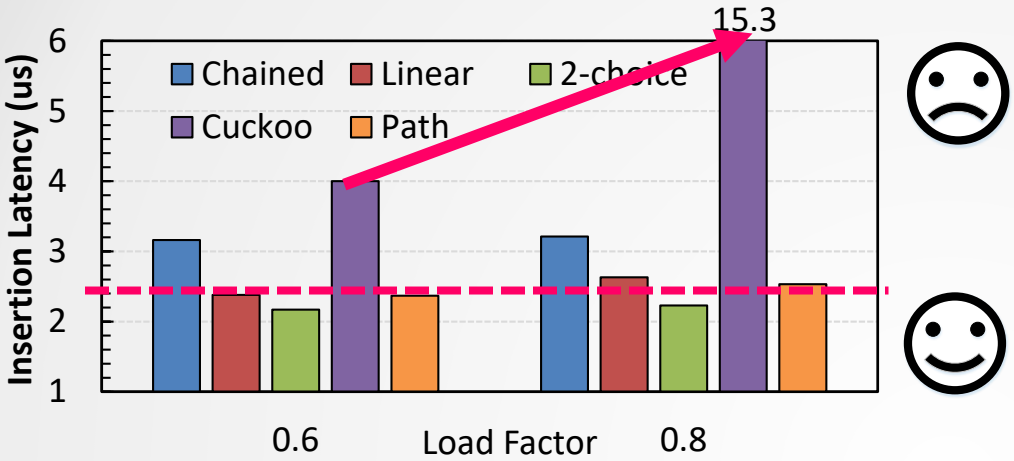
- Path hashing achieves up to 95% space utilization ratio

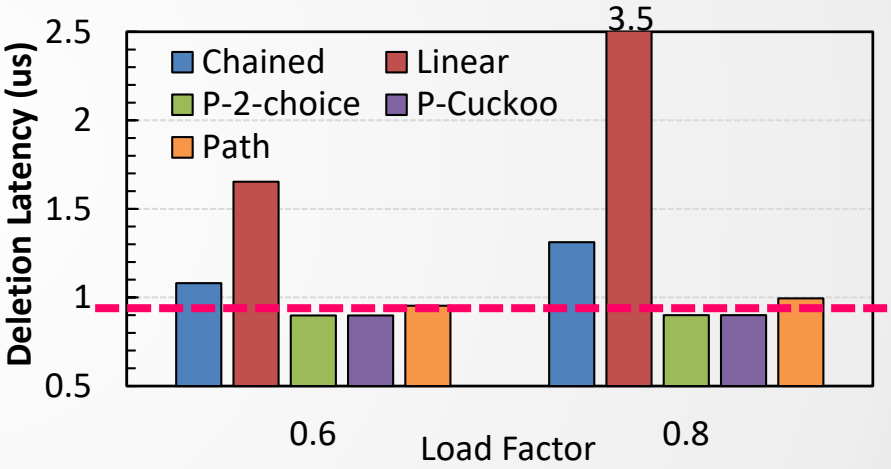
Reserved Levels vs. Space Utilization

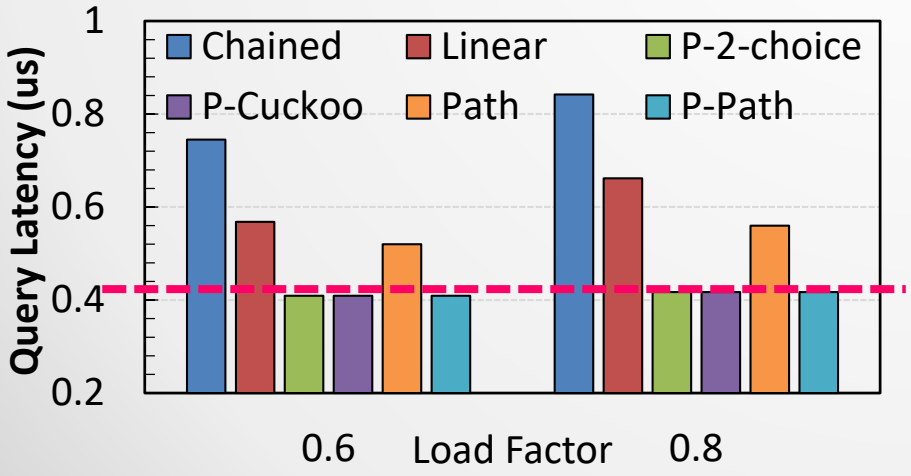


- Reserving a small part of levels can also achieve a high space utilization ratio

Request Latency







Conclusion

- Existing main hashing schemes usually cause many extra writes to NVMs
- We propose a write-friendly hashing scheme, path hashing, without extra writes while having high performance
  - ✓ Position sharing
  - ✓ Double-path hashing
  - ✓ Path shortening
- Experimental results on gem5 with NVMain
  - ✓ No extra writes
  - ✓ Up to 95% space utilization ratio
  - ✓ Low request latency

Thanks! Q&A

Open-source Code: https://github.com/Pfzuo/Path-Hashing
E-mail: pfzuo@hust.edu.cn
Homepage: http://pfzuo.github.io/about/