The Past, Present and Future of Indexing on Persistent Memory

Kaisong Huang
kha85@sfu.ca

Yuliang (George) He
georgeh@sfu.ca

Tianzheng Wang
tzwang@sfu.ca

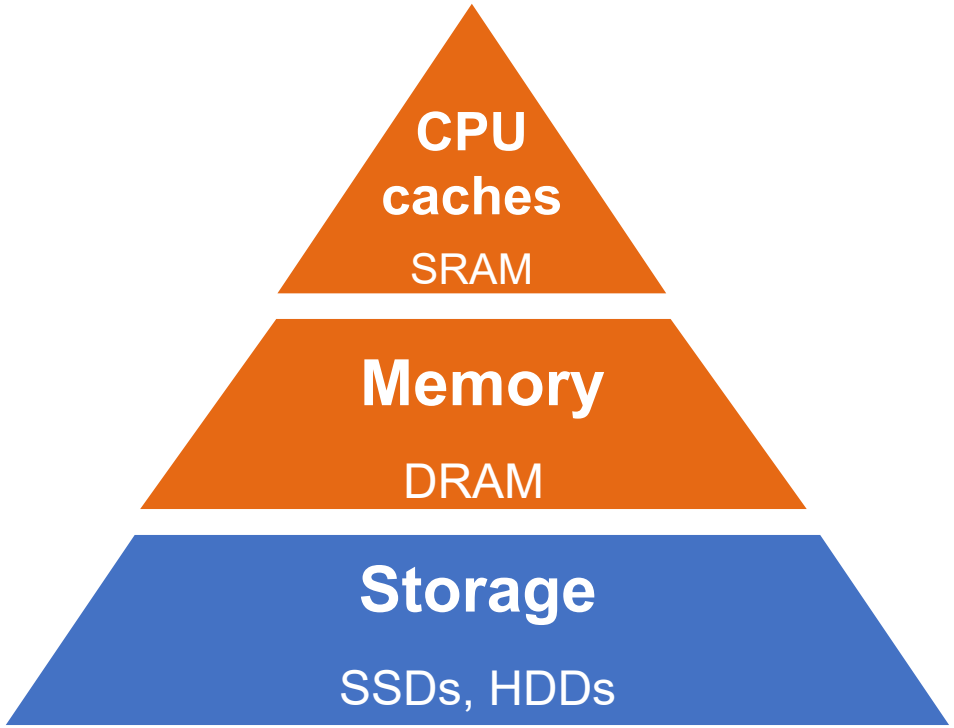
SFU SIMON FRASER UNIVERSITY

Outline

- Part 1 - The memory/storage landscape
  - Why new memory technologies?
  - Persistent memory hardware/software
- Part 2 - Range indexes
- Part 3 - Hash tables
- Part 4 - Implications and outlook
  - Especially, life after Optane

Part 1: PM and Storage Landscape

The (Traditional) Storage Hierarchy



Layers with clear boundaries

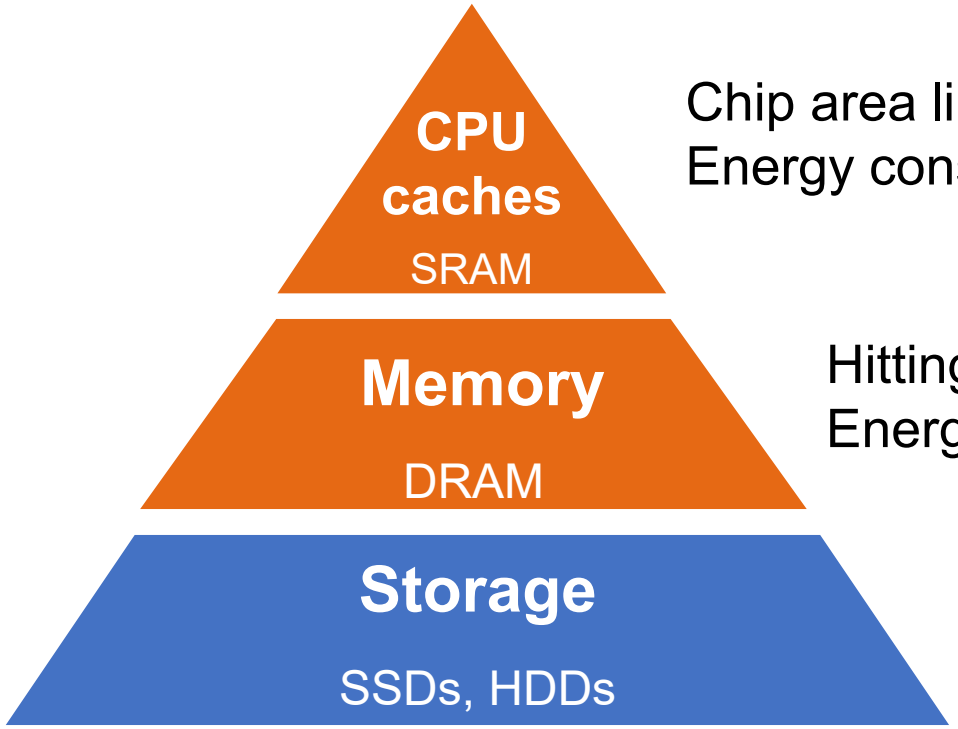
- Memory: fast but volatile
- Storage: slower than memory but persistent

Caching stores hugely successful

- Hot (index) pages in buffer pool (DRAM)
- Persist to SSDs
- Cost-effective

Facing several major challenges

Issues with in (Traditional) Storage Hierarchy



Chip area limits
Energy consumption

Hitting scalability limits (the “scaling wall”) [1]
Energy consumption

High access latency

Need new storage/memory media – mainly for ***better scalability/higher capacity + save energy***

Emerging Memory Techniques to the Rescue

- Phase change memory (PCM) [8]
  - Including Intel's 3D XPoint/Optane
  - Micron (with Intel and initially pre 2015)
- Spin-Transfer Torque Magnetic RAM (STT-RAM) [5]
  - Everspin
- Memristor [2]
  - Notable attempt by HP(E)'s The Machine [3, 4]
- Carbon NanoTube RAM (NRAM, NanoRAM) [6]
  - Nantero
- Ferroelectric RAM (FeRAM) [7]
  - Fujitsu

They just all happen to be non-volatile!

- *Various new flash/DRAM technologies – more on this later*

Persistent Memory

Aside: Terminology

- Non-volatile RAM (NVRAM)
- Non-volatile memory (NVM)*
- Persistent memory

** Except flash memory*

➔ The same thing: durable + byte-addressable

Persistent Memory Properties

- Byte-addressable, durability
  - + Energy efficient
  - + Scales, high density, cheaper

Can be used to build both memory and storage

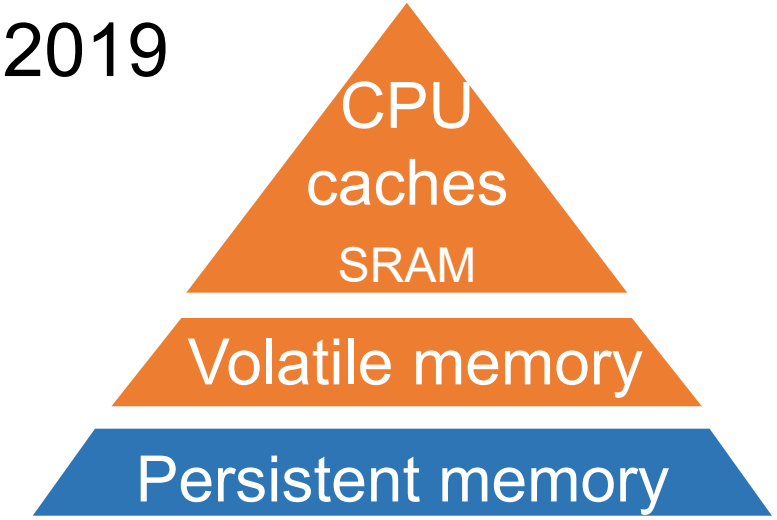
Can be faster or slower than DRAM

- Performance varies depending on particular memory technology
  - E.g., STT-RAM as an alternative to SRAM cache
    - Tradeoffs between persistence/retention/speed/energy profile [9]

- In most cases, biased towards PCM/3D XPoint (Optane) and compare with DRAM:
  - + Energy efficient
  - + Scales, high density, cheaper
  - – Higher read/write latency than DRAM
  - – Read/write asymmetry
  - – Limited lifetime (but not a big concern)

Persistent Memory

- Available today: Intel 3D XPoint (Optane) since 2019
  - But winding down, more on this later
  - (future slides – Optane-specific marked with )
- Other candidates
  - Work-in-progress, or
  - Failed previous attempts, or
  - Do not scale (yet), or
  - Do not scale economically, or



Persistent Memory

- Available today: Intel 3D XPoint (Optane) since 2019
  - But winding down, more on this later
  - (future slides – Optane-specific marked with )
- Other candidates
  - Work-in-progress, or
  - Failed previous attempts, or
  - Do not scale (yet), or
  - Do not scale economically, or

Aside: Non-Volatile DIMMs (NVDIMMs) [9]

- DRAM + flash + supercapacitor
  - Flush data to flash upon power failure, load back when powered on again
  - SNIA standardized: NVDIMM-F, NVDIMM-N, etc.
- **Also "persistent" and available today but:**
  - **Doesn't scale (due to DRAM)**
  - Same speed as DRAM (NVDIMM-N)
  - Expensive

➔ A major research vehicle pre-Optane

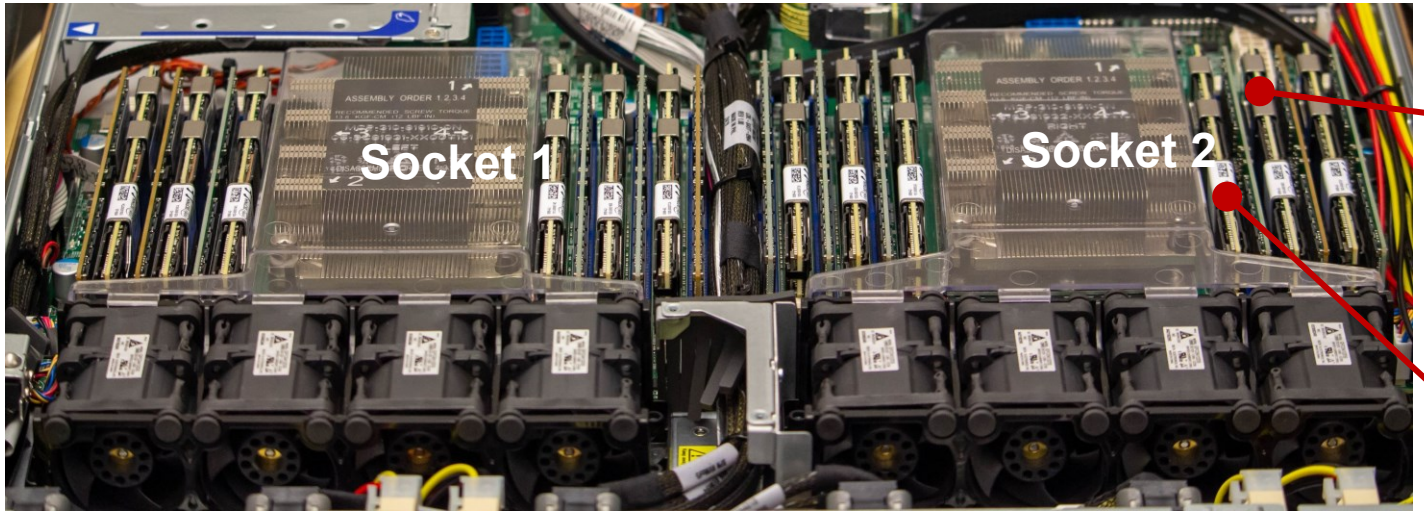
Optane PMem

Aka “Optane DCPMM”

- High capacity – easily TB level
- “Economical” (more later)
  - ~CA$750 / 128GB Optane PMem
  - ~CA$2000 / 128GB DRAM

~July 2019

1	Intel Xeon Gold 6252 24-Core Server	CA$26243.32
	2x Intel Xeon Gold 6252 24-Core 48-Thread 2.10 GHz 35.75M Cache Server Processor	
	384GB (12x 32GB) DDR4-2666 ECC Registered RDIMM	
	1x Intel D3-S4610 240GB SATA SSD - 3D TLC 2.5" 3 DWPD	
	12x Intel 128GB DDR4-2666 Optane Memory - AEP 3DXP DCPMM Optane Memory	

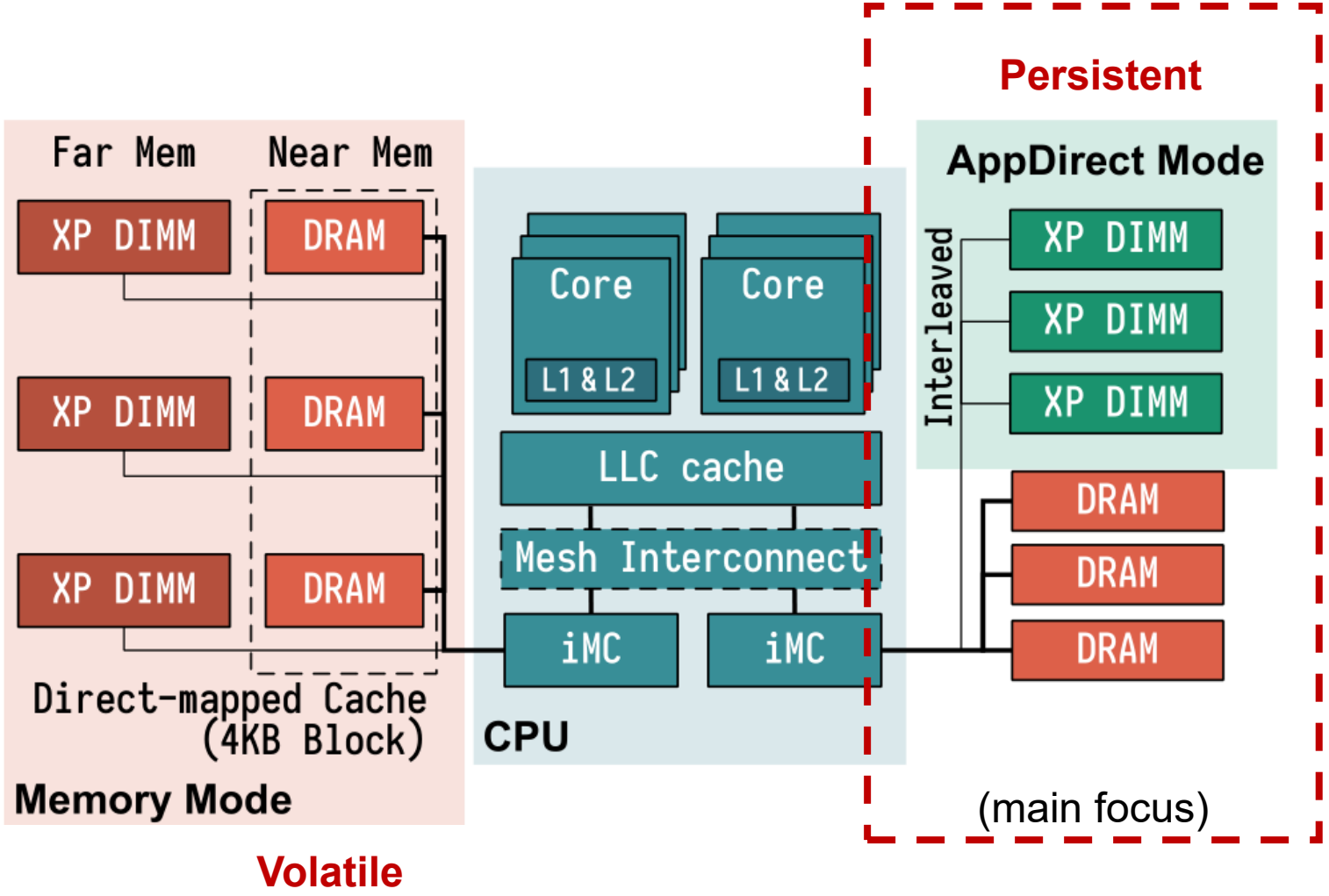


DRAM DIMMs
75ns latency
BW >60-100 GB/s
(16 threads)

Optane PMem 100
300ns read latency
Read BW 7.4-40GB/s
Write BW 5.3-10GB/s
(16 threads)

PMem 200: ~30% higher

System Architecture and Operation Modes



[CEATEC] Rohm Demonstrates Nonvolatile CPU, Power Consumption Cut by 90%

Motoyuki Ooishi, Nikkei Electronics — Oct 4, 2007

Like Share 0 Tweet 0 Print

Rohm Co. Ltd. prototyped a nonvolatile CPU and exhibited it at CEATEC Japan 2007. The company prototyped a 4-bit microcomputer and made it nonvolatile by adding ferroelectric memory chips to all of the about 300 registers.

This time, the company demonstrated a breakout game by using a nonvolatile CPU, comparing with the case in which an existing CPU is used.

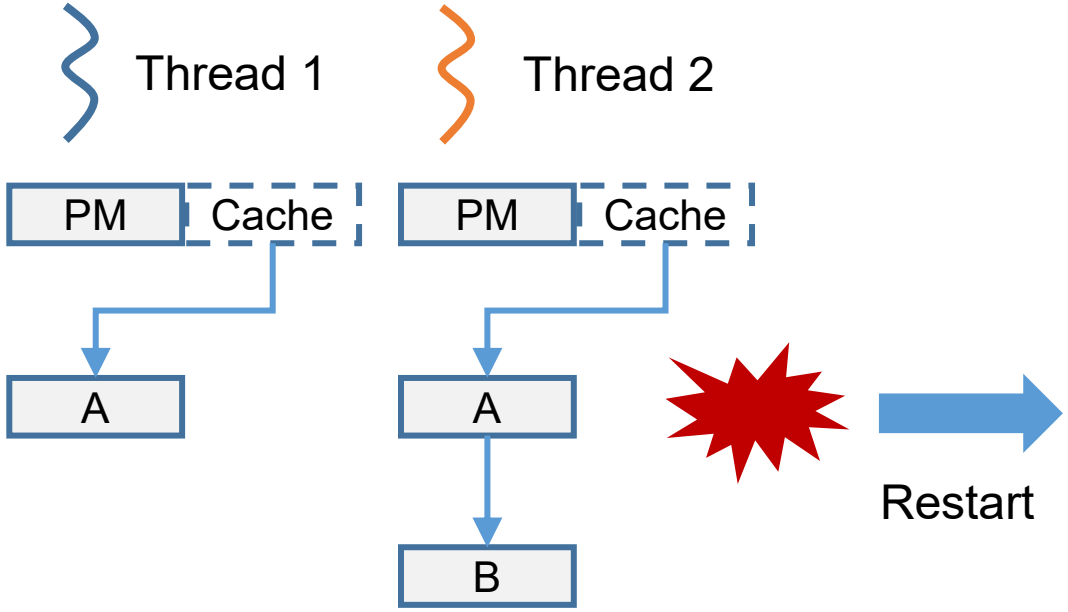
The demonstration of a breakout game

Still volatile CPU caches

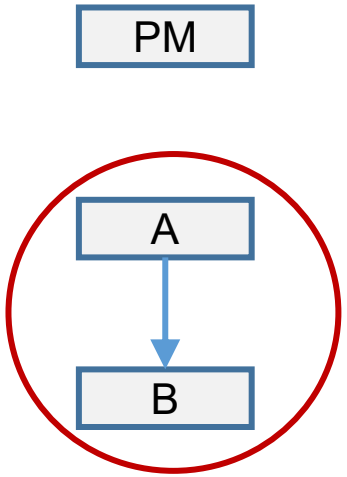
[41] J. Yang et al. An Empirical Guide to the Behavior and Use of Scalable Persistent Memory

Programming Model without eADR

- ADR: Asynchronous DRAM Refresh
  - Includes write buffer and PMem, but not the CPU caches



Case 1: Optane PMem 100, pre- Ice Lake
Actually persisted state:



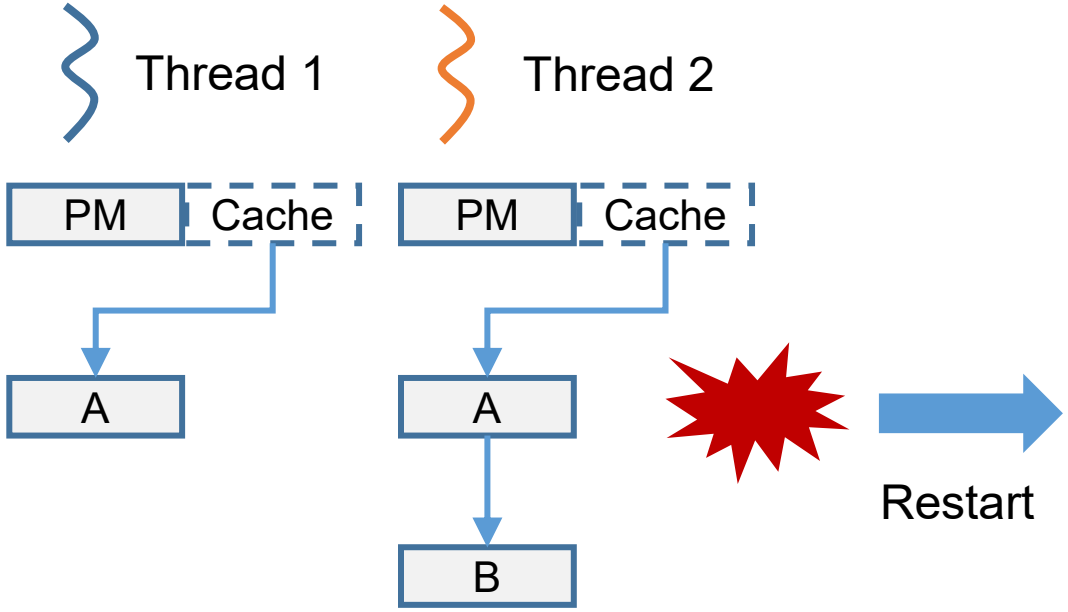
Unreachable

Need explicit cacheline writeback (clflush, clflushopt, clwb)

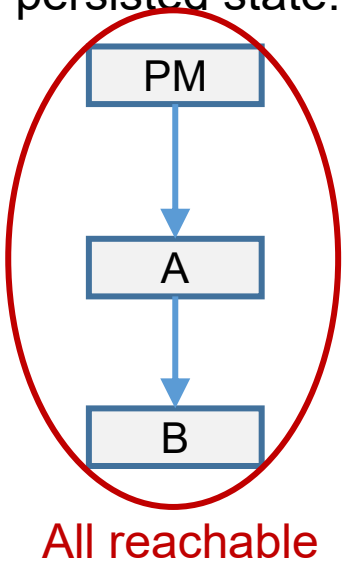
Visible != durable

Programming Model with eADR

- eADR: Enhanced Asynchronous DRAM Refresh
  - Includes write buffer and PMem, *and* the CPU caches



Case 2: Optane PMem 200, pre- Ice Lake
Actually persisted state:



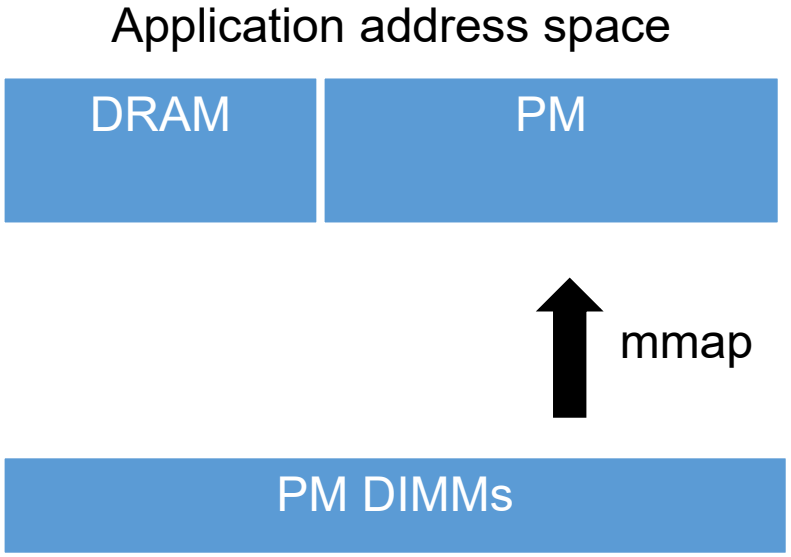
All reachable

Enhanced ADR – no need to flush; fence still needed

Visible == durable

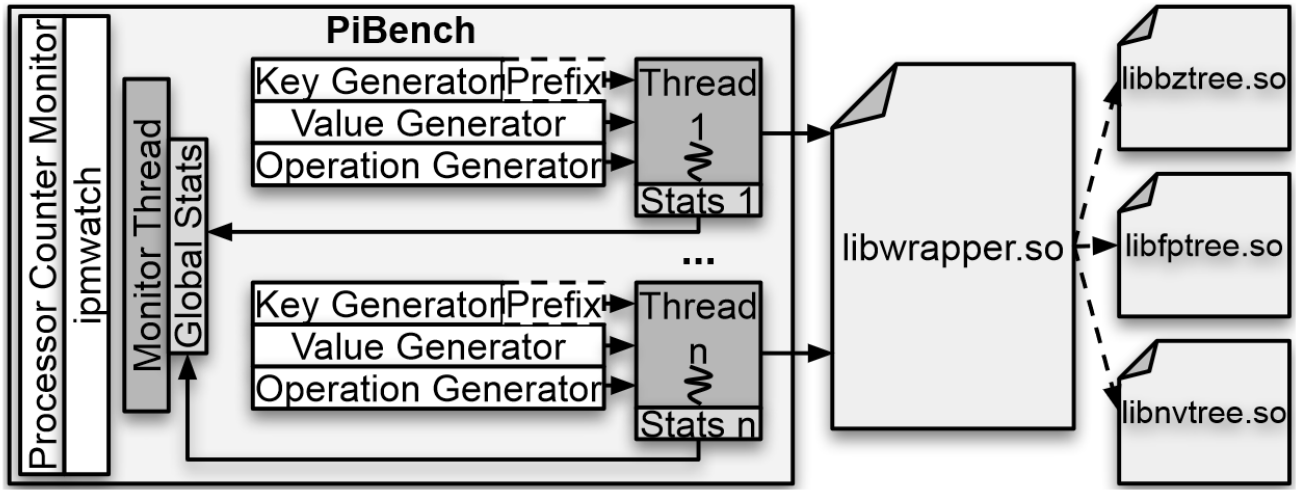
Software Tools: PM Programming

- Access via load/store instructions
- Add fences and flushes (without eADR)
- Guaranteed: 8-byte atomic write
  - Atomics (CAS, XCHG, etc.) also work
- Allocating/deallocating PMem
  - `malloc` doesn't work!
  - Handling (persistent) memory leaks
  - Solutions
    - Ownership transfer protocol: application provides a tracked location for allocator
    - "Transactions" (for durability): use logging
- Guaranteed by PM programming libraries
  - Intel PMDK (https://pmem.io/pmdk)
  - Research: NVHeaps [10], Mnemosyne [11], PMwCAS [12]



Software Tools: Index Evaluation

- PiBench [18]
  - Unified benchmarking framework
  - Pluggable index shared lib
  - Issue synthetic workloads
    - R/W ration, varying core count
  - Stats information
    - Throughput, tail latency, bandwidth



  - Not limited to PM; used by various recent index work
  - Open-source:
    - https://github.com/sfu-dis/pibench

Part 2: PM Range Indexes

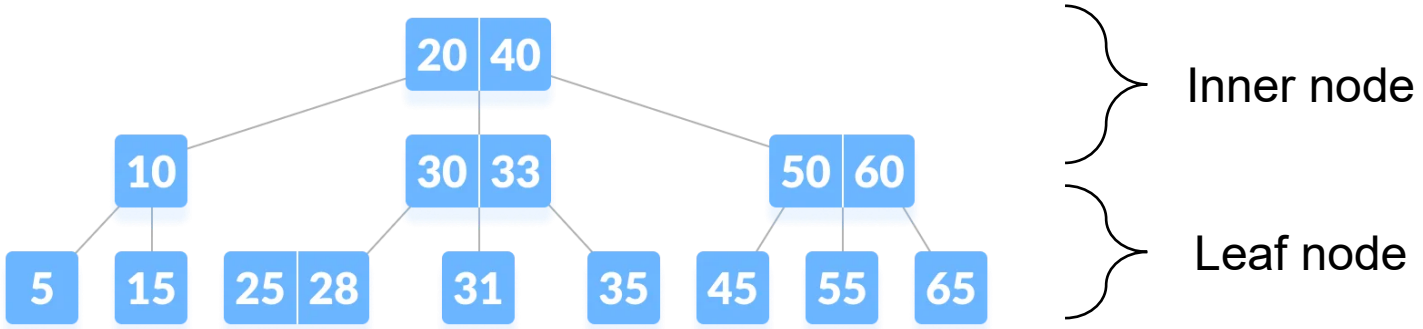
Part Two: Outline

- Common range index choices: B+-Tree vs Trie
- Range indexes on PM
  - Pre-Optane PM indexes
  - State-of-the-art PM indexes

Btree vs Trie

BTree:

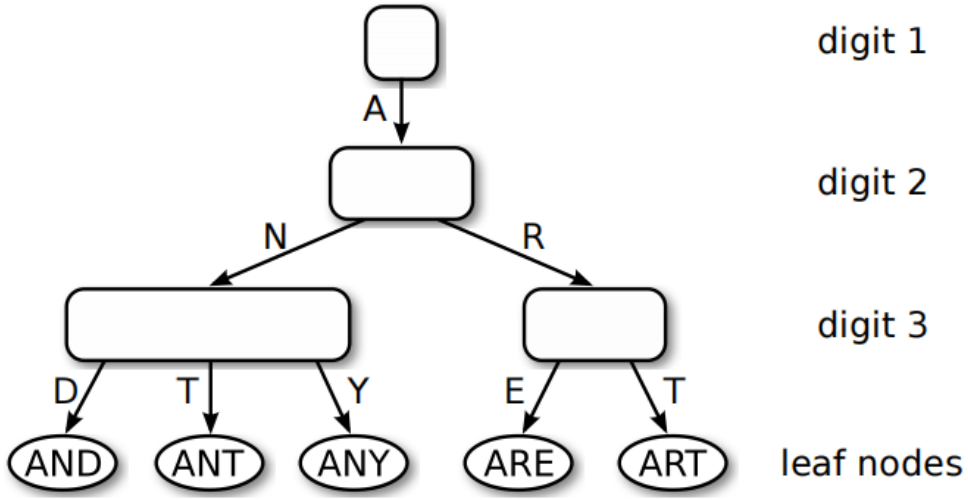
- O(log n) access time
- Range scan locality
- Variable-length key support



https://cdn.programiz.com/sites/tutorial2program/files/b-tree.png

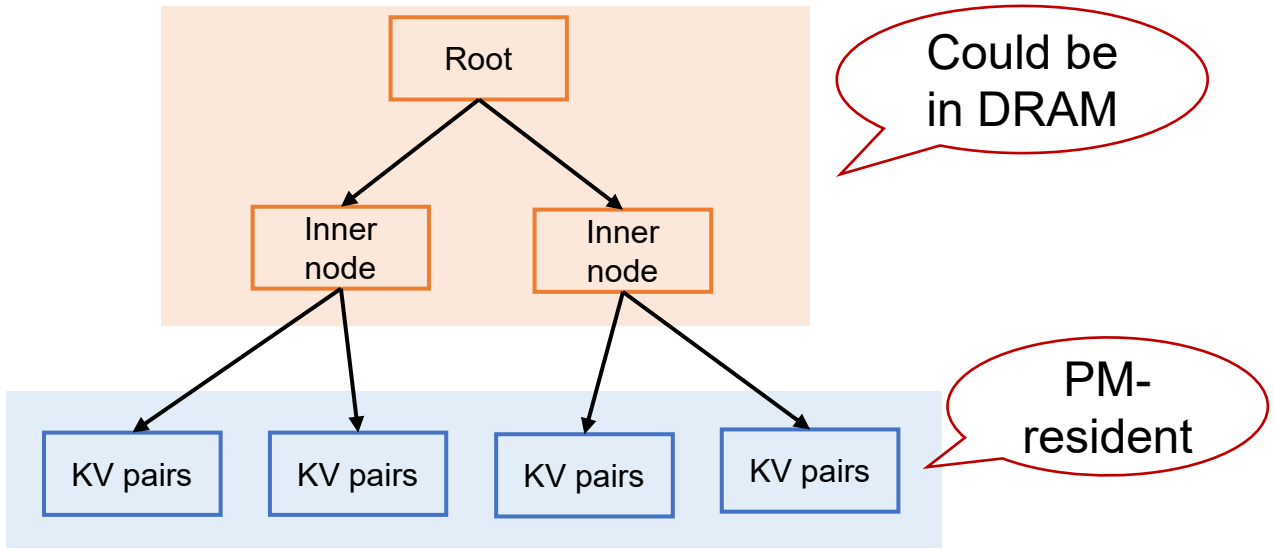
Trie:

- O(L) access time (L = key length)
- Variable-length key support
- Pointer chasing during scan



[13] The Adaptive Radix Tree: ARTful Indexing for Main-Memory Databases

Range Indexes on Persistent Memory



- No serialization/deserialization
- Directly persist on PM
- Tailor-made for Optane DCPMM
- (Near) Instant recovery

Challenges:

- Consistency - 8-byte atomic write
- Performance - scarce write bandwidth
- Recovery - avoid persistent memory leak

Key optimization goal:

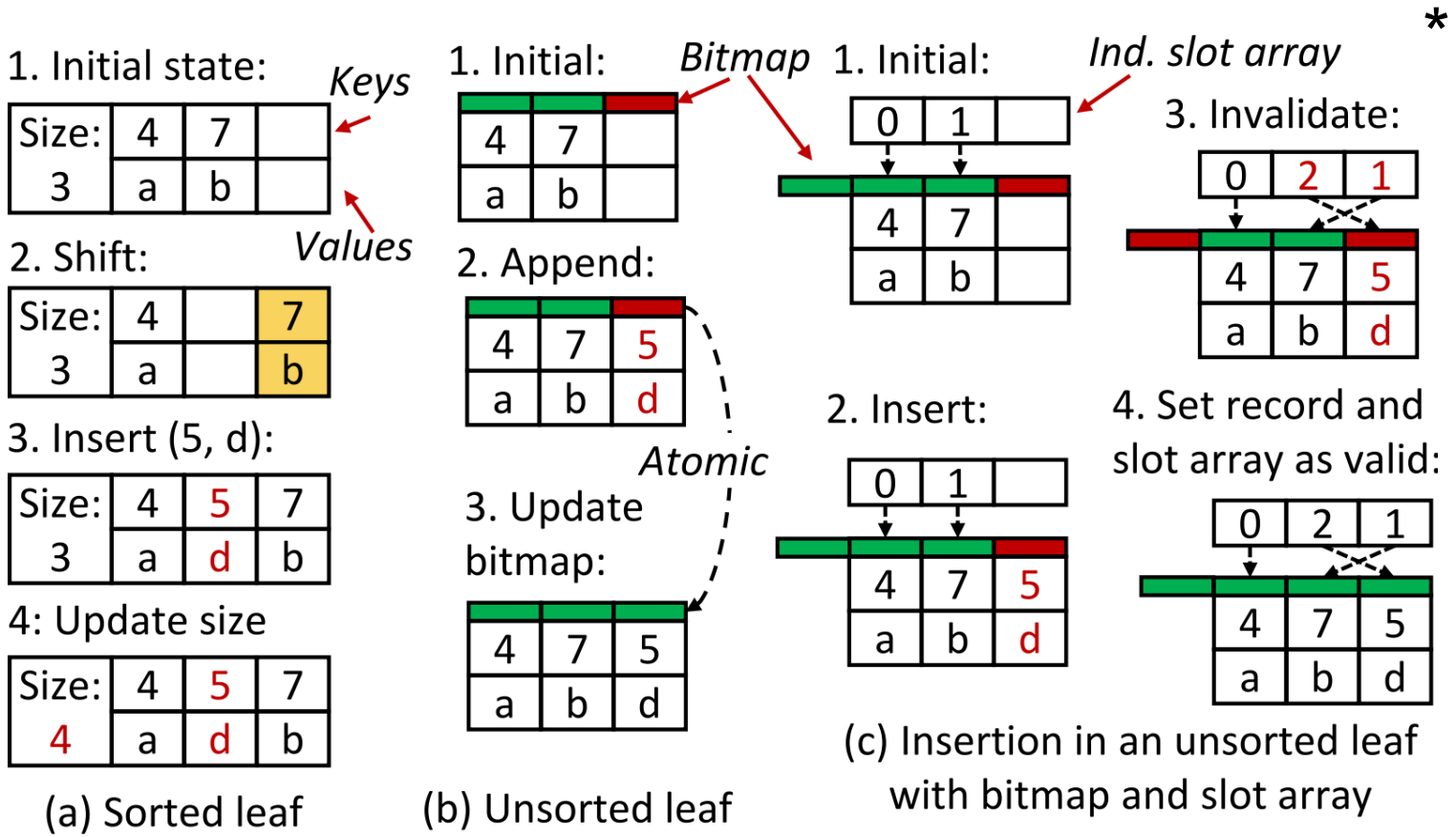
- Reduce PM accesses ➔ higher performance

Perhaps 10s-100s of proposals by now

- Even before real devices appeared
- Even more with real devices

Pre-Optane: wBTree [14]

- Unsorted leaf with atomic update
- Indirection array
  - 1 bit to indicate validity
- Logging during split
- Single-threaded

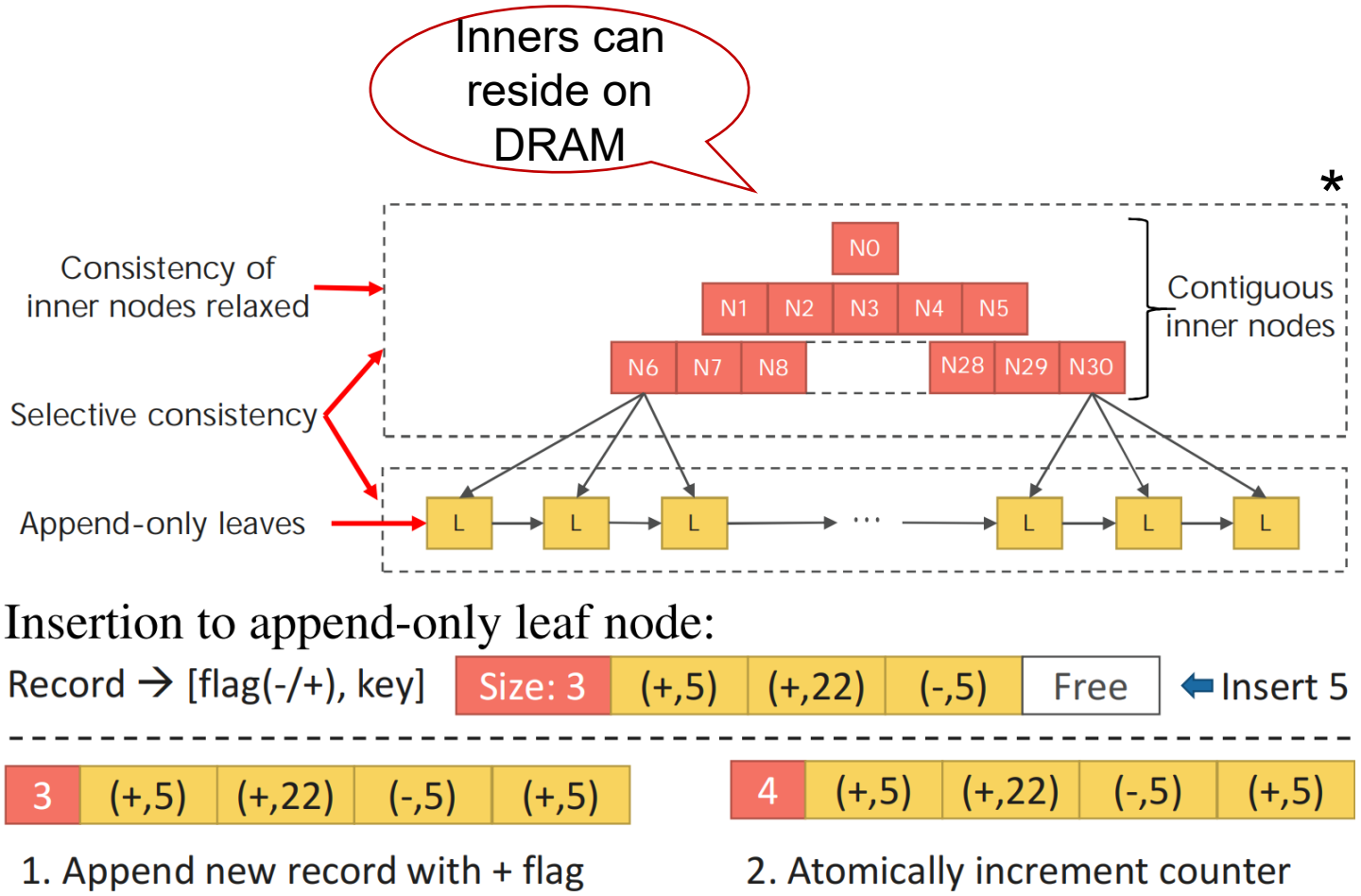


*

[14] Persistent B⁺-trees in non-volatile main memory, *VLDB 2015*

Pre-Optane: NVTree [15]

- Selective consistency
- Contiguous inner nodes
  - Gaped array to absorb split
- Unsorted leaf
  - Append-only strategy
- Scan backwards to find key

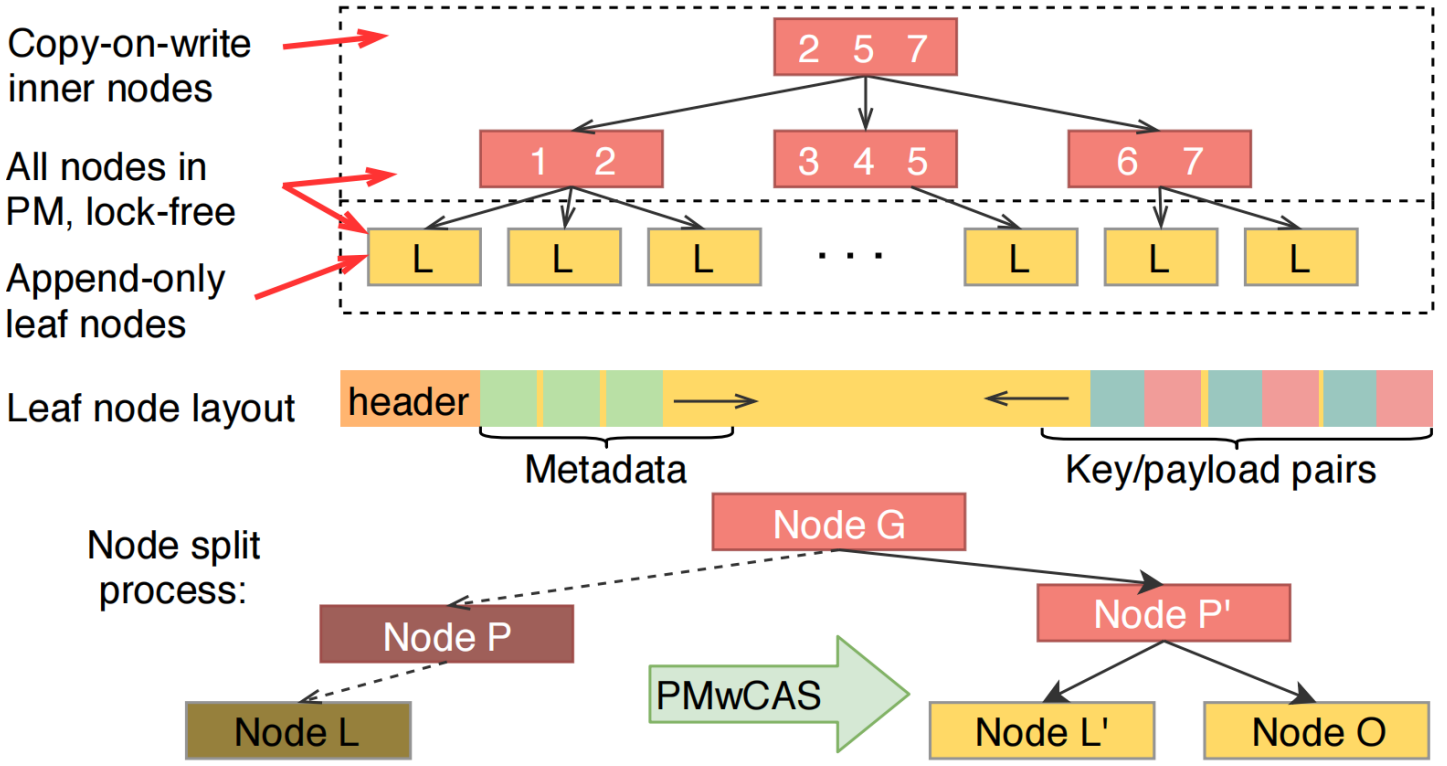


[15] NV-Tree: Reducing Consistency Cost for NVM-based Single Level Systems, *FAST 2015*

Pre-Optane: BzTree [16]

- Lock-Free (PMwCAS – Persistent Multi-word Compare And Swap)
- Unsorted leaf
  - Periodically sort records
- Search method
  - Binary search sorted area
  - Linear search unsorted area

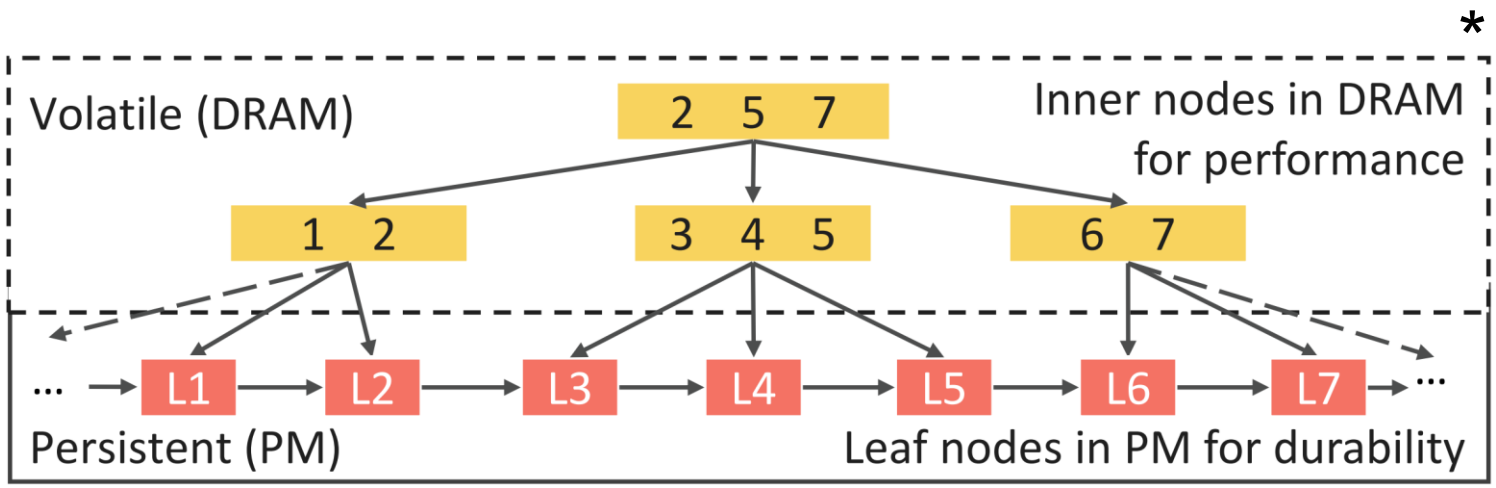
*



[16] Bztree: a high-performance latch-free range index for non-volatile memory, *VLDB 2018*

Pre-Optane: FPTree [17]

- Selective persistence
- Unsorted leaf + fingerprints (one byte hash of key)
- Selective concurrency
  - HTM for inner node update
  - Locks for leaf node update



[17] FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory, *SIGMOD 2016*

Pre-Optane PM Range Indexes (Pre-2019) [18]

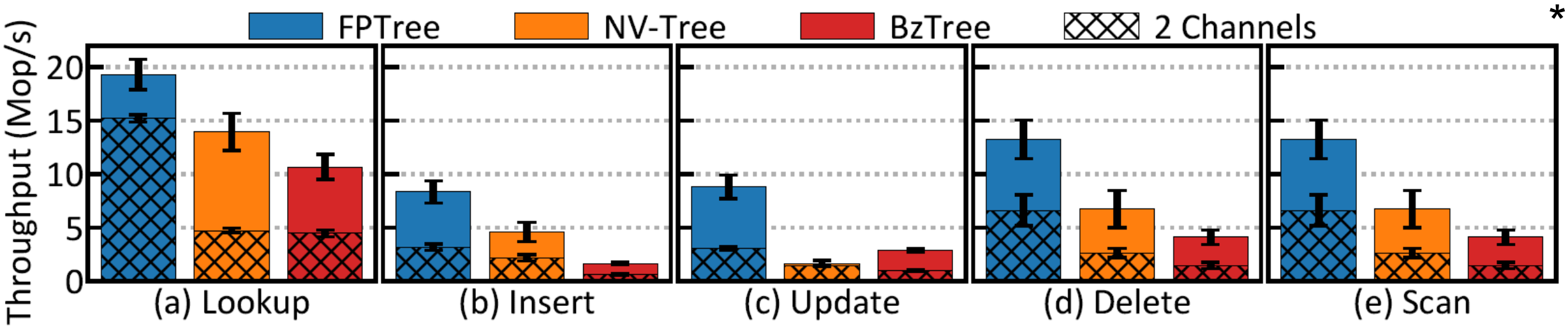
- Proposed under emulation, evaluated under Optane PMem

Index	Architecture	Node Architecture	Concurrency
wBTree [VLDB '15]	PM-only	Unsorted; Indirection array	Single-threaded
NV-Tree [FAST '15]	DRAM + PM	Unsorted leaf; Inconsistent inner node	Locking
FPTree [SIGMOD '16]	DRAM + PM	Unsorted Leaf; Fingerprints	HTM (inner) + Locking (leaf)
BzTree [VLDB '18]	PM-only	Partially unsorted leaf	Lock-free + PMwCAS

[18] Evaluating Persistent Memory Range Indexes, *VLDB 2020*

Pre-Optane PM Range Indexes (Pre-2019) [18]

- 6 channels (solid + shadow) vs. 2 channels (shadow only)
- 23 threads



Key takeaways: save write bandwidth + leverage DRAM + fingerprinting

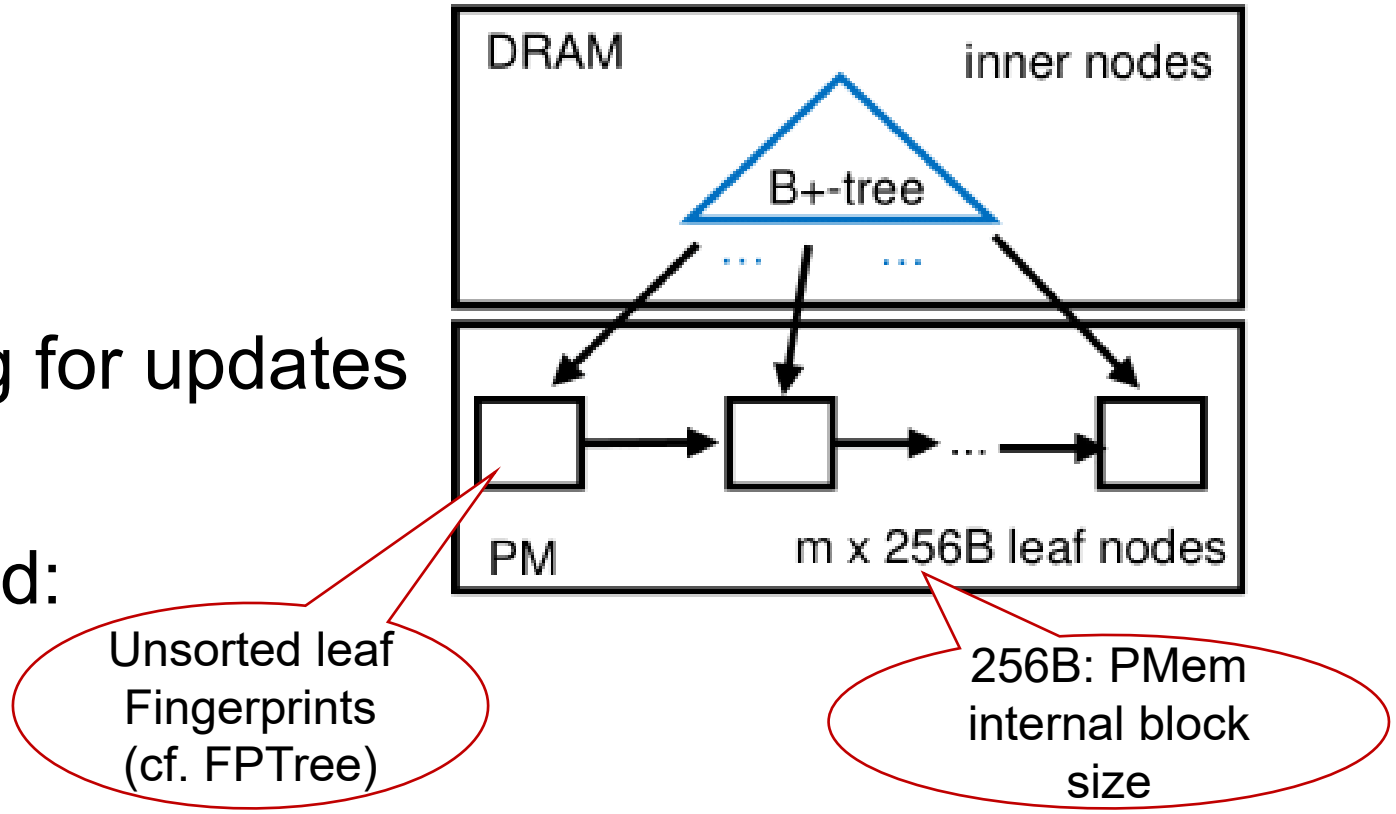
[18] Evaluating Persistent Memory Range Indexes, *VLDB 2020*

Into the Era of Optane 2019-2022

- Even more indexes
  - 10s of papers in VLDB/SIGMOD/SOSP, etc.
- More index structure choices
  - B+-tree, trie, hybrid, learned
- Functionality
  - NUMA-awareness, variable-length key support, etc.

Optane: LB+-Tree [19]

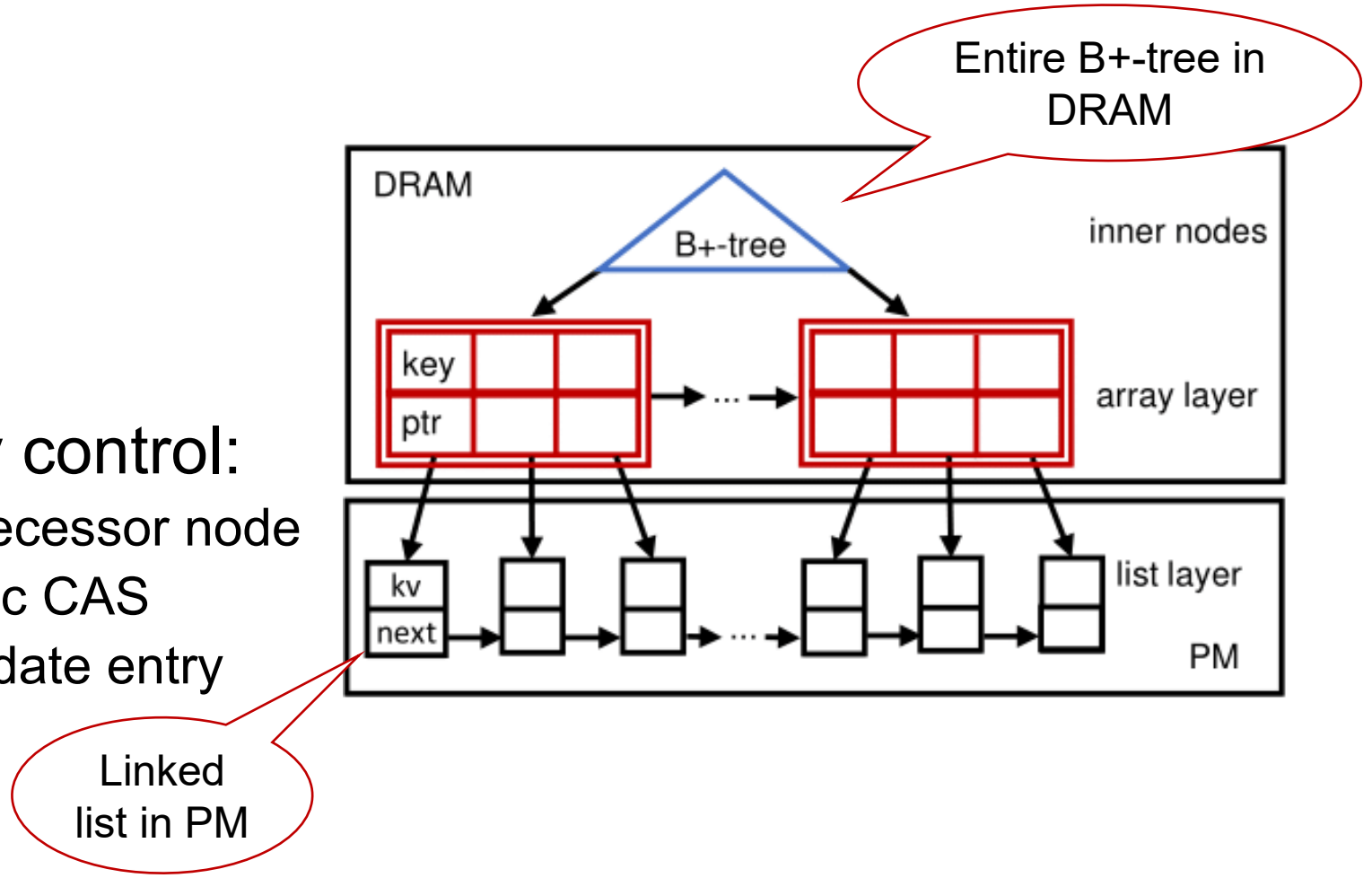
- B+-tree based
  - Inner nodes in DRAM
  - Leaf nodes in PM
- HTM for traversal, locking for updates
- + New techniques to avoid:
  - Excessive PM writes
  - Logging overhead



[19] LB+-Trees: optimizing persistent index performance on 3DXPoint memory, *VLDB 2020*

Optane: µTree [20]

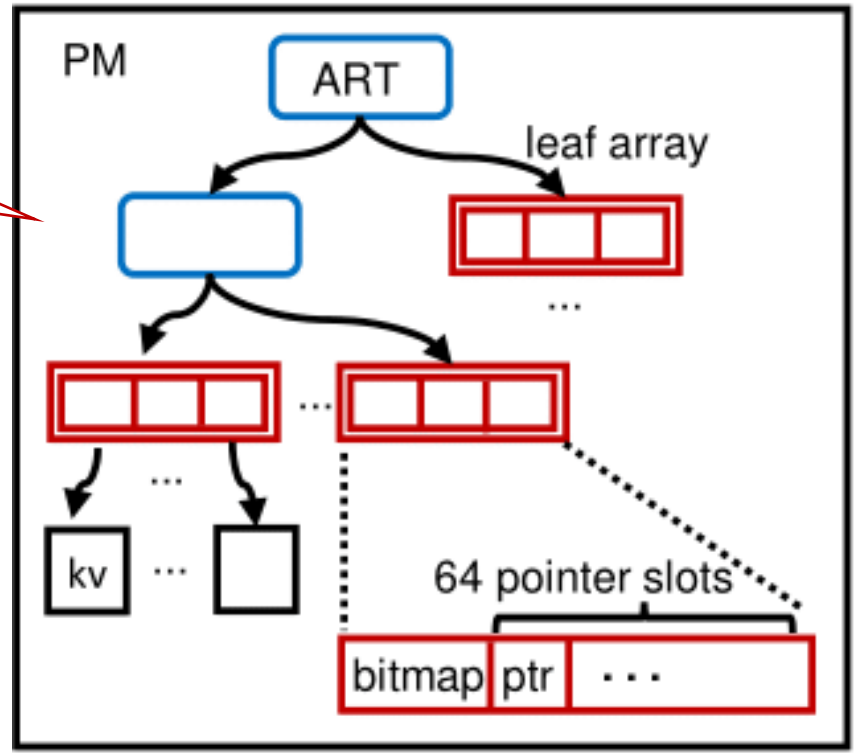
- B+-tree based
- Optimized for tail latency
- Coordinated concurrency control:
  - Traverse B+-Tree, find predecessor node
  - Update list layer using atomic CAS
  - Lock array layer leaf and update entry



[20] µTree: a Persistent B+-Tree with Low Tail Latency, *VLDB 2020*

Optane: ROART [21]

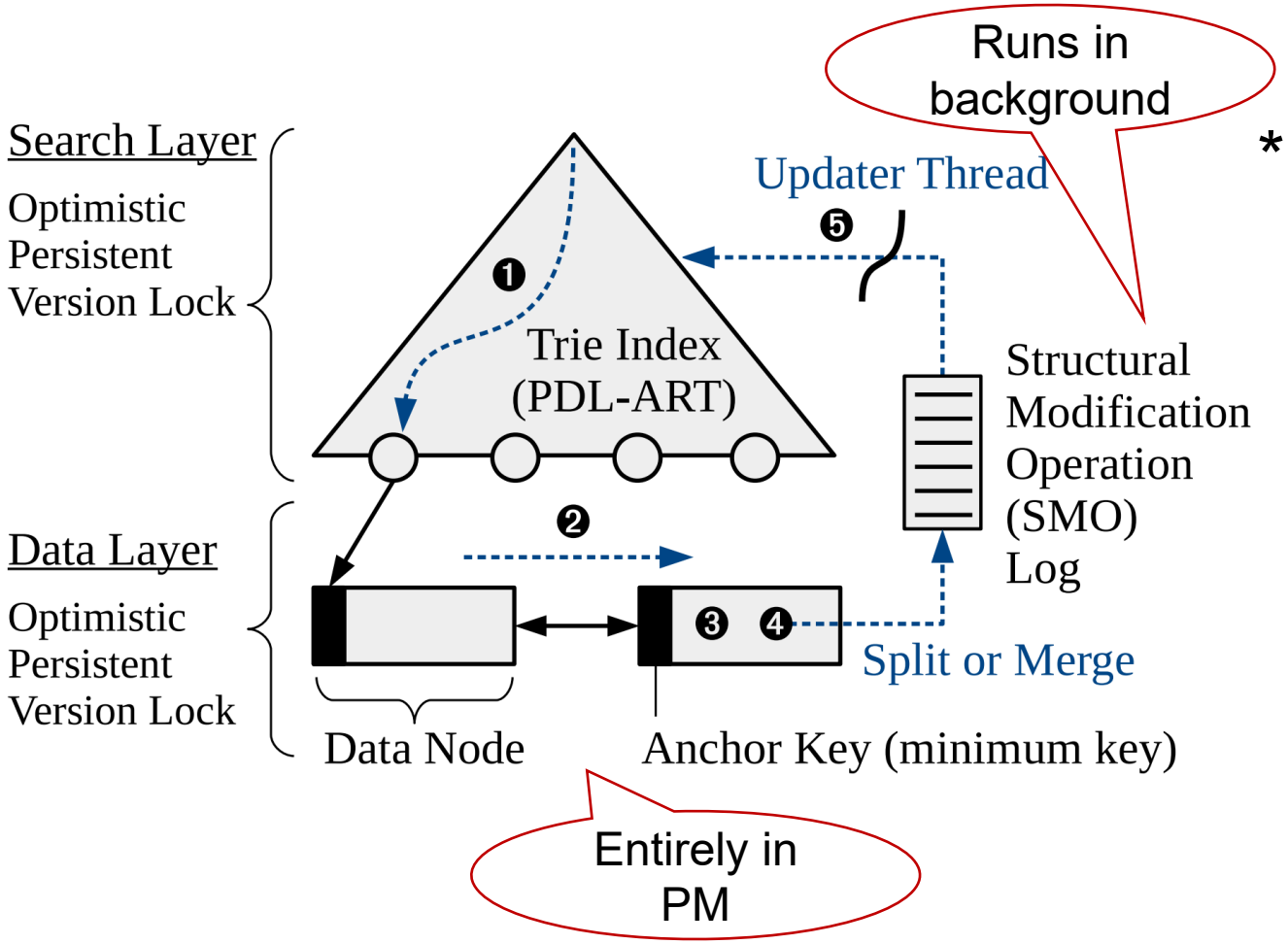
- Based on ART
- Optimized for range scan
- Compact subtrees into leaf arrays
- Delayed Check Memory Management
- Concurrency
  - ART-ROWEX
  - Non-temporal stores



[21] ROART: Range-query Optimized Persistent ART, *FAST 2021*

Optane: PACTree [22]

- Trie-based (ART)
- Search layer: persistent trie
- Data layer: linked list of leaves
- Asynchronous update
  - SMOs by background threads
- NUMA-optimized
  - Per-node PM pool

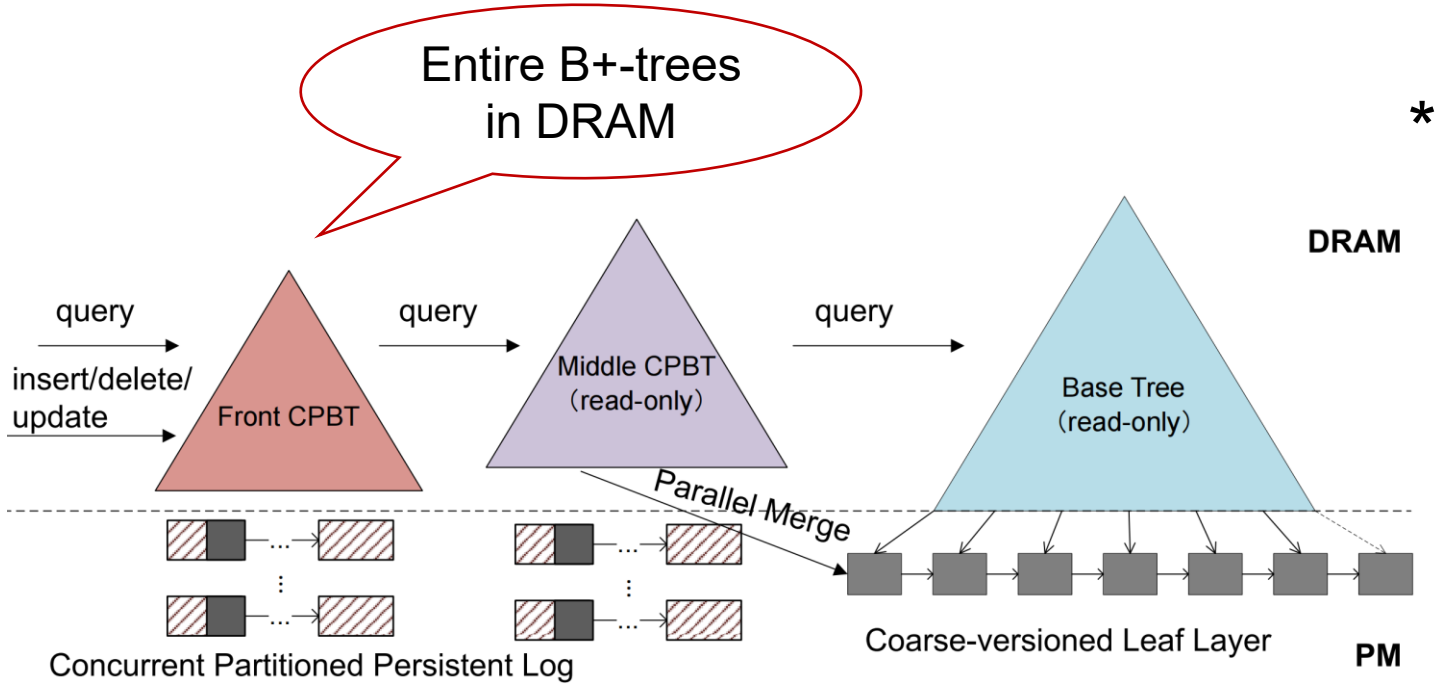


[22] PACTree: A High Performance Persistent Range Index Using PAC Guidelines, *SOSP 2021*

Optane: DPTree [23]

- Hybrid B+-tree and trie

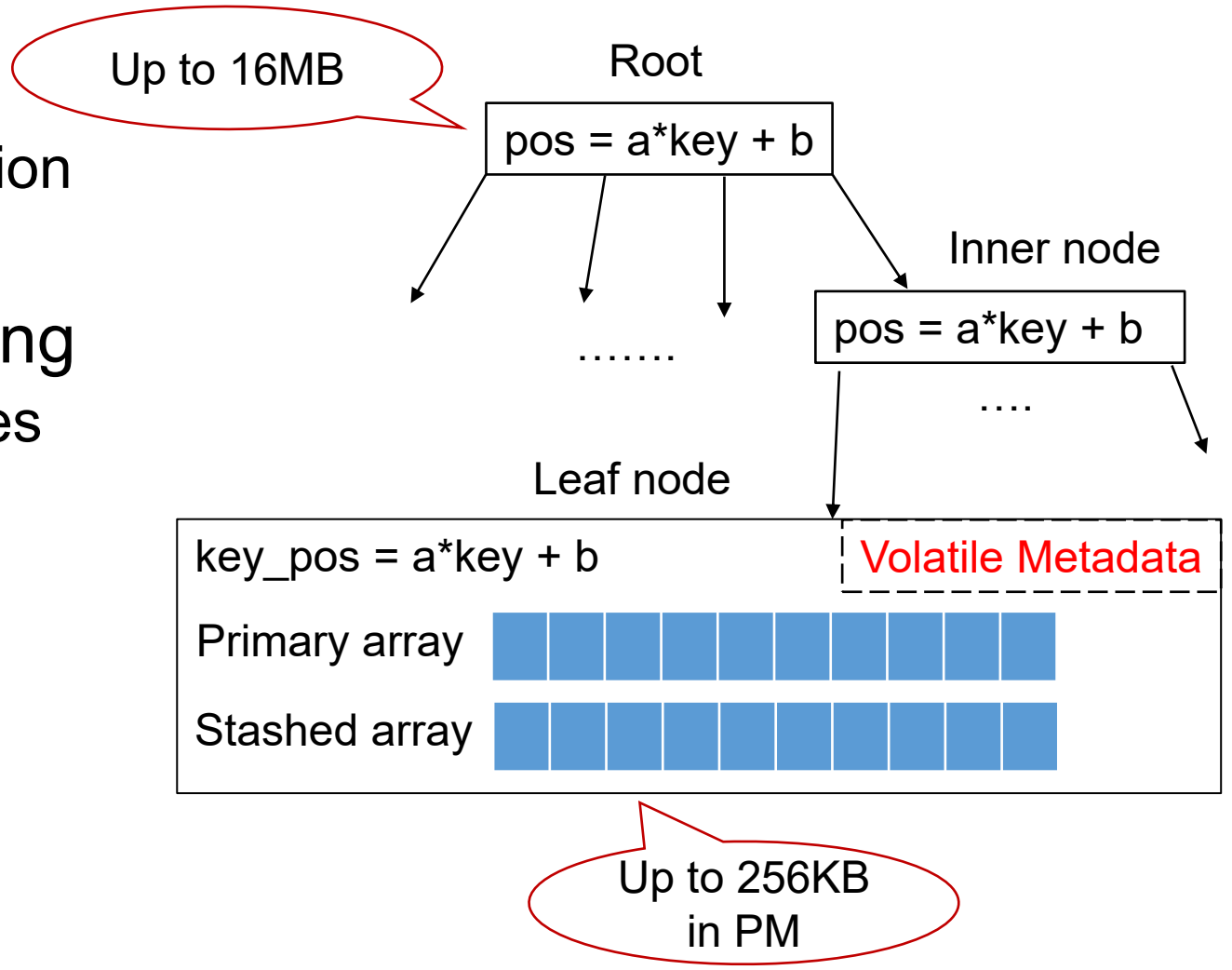
- Front Buffer Tree
  - B+-tree
  - All modifications with logging
- Base Tree
  - Read-only trie for inner nodes
  - B+-Tree style leaf nodes
  - Accumulates front buffer trees
- Lookup will traverse all trees



[23] DPTree: differential indexing for persistent memory, *VLDB 2020*

Optane: APEX [24]

- Learned, based on ALEX [25]
  - Linear function to predict key location
- Probe-and-Stash collision handling
  - Probe primary array up to 16 entries
  - If empty slot found, insert into PA
  - Otherwise insert to stashed array
- Concurrency
  - Inner: Lock-free traversal
  - Leaf: Optimistic locking



[24] APEX: A High-Performance Learned Index on Persistent Memory, VLDB 2022
[25] ALEX: An Updatable Adaptive Learned Index, SIGMOD 2020

Optane-Era PM Range Indexes (2019-2022) [26]

	Architecture	Node structure	Concurrency
LB+-Tree [VLDB 20]	B+-tree; DRAM (inner) + PM (leaf)	Unsorted leaf; fingerprints; extra metadata	HTM (traversal) + locking (update)
uTree [VLDB 20]	B+-tree; DRAM (B+-tree) + PM (leaf)	Sorted	Locking (array layer) + lock-free (list layer)
DPTree [VLDB 20]	Hybrid; DRAM (B+-tree, trie inner) + PM (trie leaf)	Unsorted leaf; fingerprints; indirection; extra metadata	Optimistic lock + async update
ROART [FAST 21]	Trie; PM-only	B+-tree like unsorted leaf; fingerprints	ROWEX
PACTree [SOSP 21]	Trie; PM-only (conceptually DRAM+PM)	Unsorted leaf; fingerprints; indirection	Optimistic lock + async. Update
APEX [VLDB 22]	Learned index; PM-mostly (metadata in DRAM)	Partially unsorted leaf; fingerprints; stashed array	Lock-free traversal + optimistic locking
FPTree [SIGMOD 16]	DRAM (inner nodes) + PM (leaf nodes)	Unsorted leaf nodes; fingerprints	HTM (inner nodes) + locking (leaf nodes)

B+-Trees: extensive use of DRAM

Tries: B+-Tree styled leaf

(mostly) unsorted + Search techniques

(mostly) optimistic

Support var-keys naturally

NUMA-optimized

Learned

Detailed performance comparison: Evaluating Persistent Memory Range Indexes: Part Two, VLDB 2022

Part 3: PM Hash Tables

Part Three: Outline

- Range indexes vs hash tables
- Representative hashing schemes
- New challenges and new proposals
- Design summary

Range Indexes vs. Hash Tables

	Range Indexes	Hash Tables
Pros	• Good at range queries • Smooth growth (collision-free)	• Good at point queries • Average O(1) time complexity for insertion/deletion/search
Cons	• Average O(logN) time complexity for insertion/deletion/search	• Lack support for range queries • Unavoidable collisions

PM Range Indexes vs. PM Hash Tables

	Range Indexes	Hash Tables
Pros	Common frenemy: PM access!	
Cons		

PM Range Indexes vs. PM Hash Tables

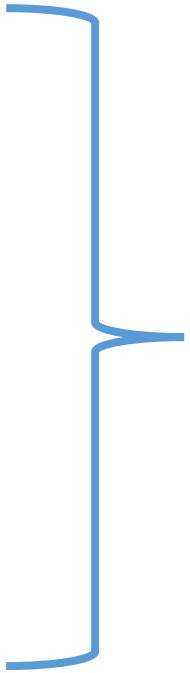
	Range Indexes	Hash Tables
Pros		
Cons	**Common challenges:** #1 Consistency – 8-byte atomic write #2 Performance – scarce write bandwidth #3 Recovery – avoid persistent memory leak	

Recap

A hash table implementation = hashing scheme + hash function

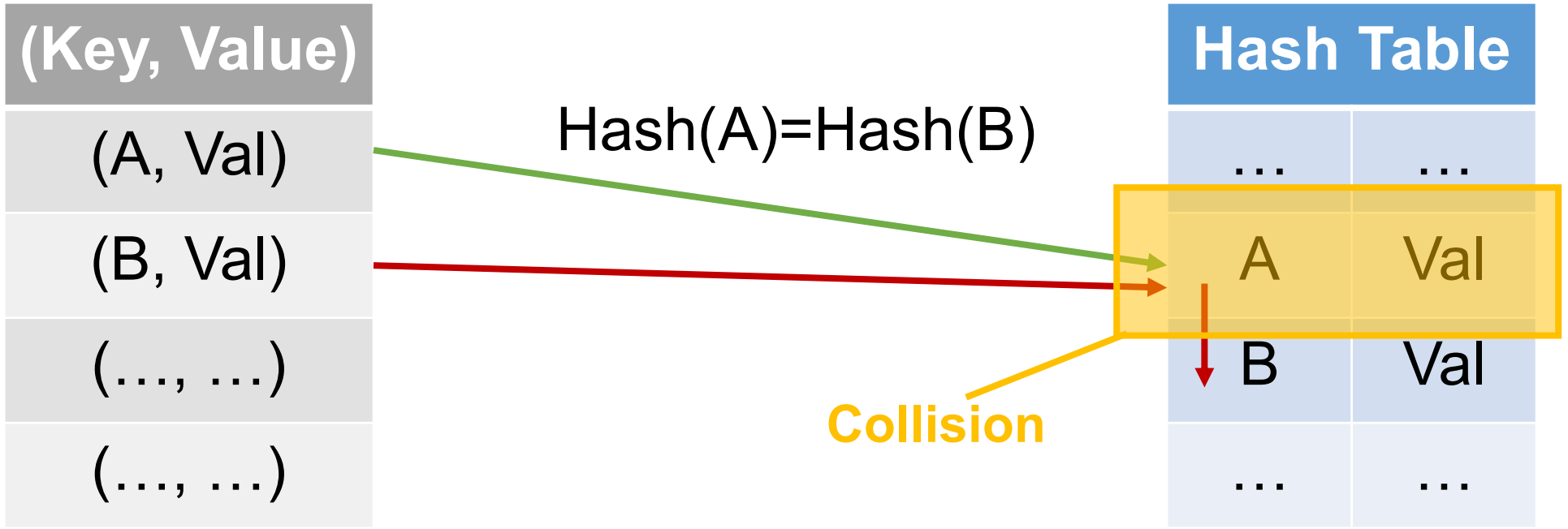
Static hashing schemes
- Linear probing
- Cuckoo hashing
- …

Dynamic hashing schemes
- Extendible hashing
- Linear hashing
- …

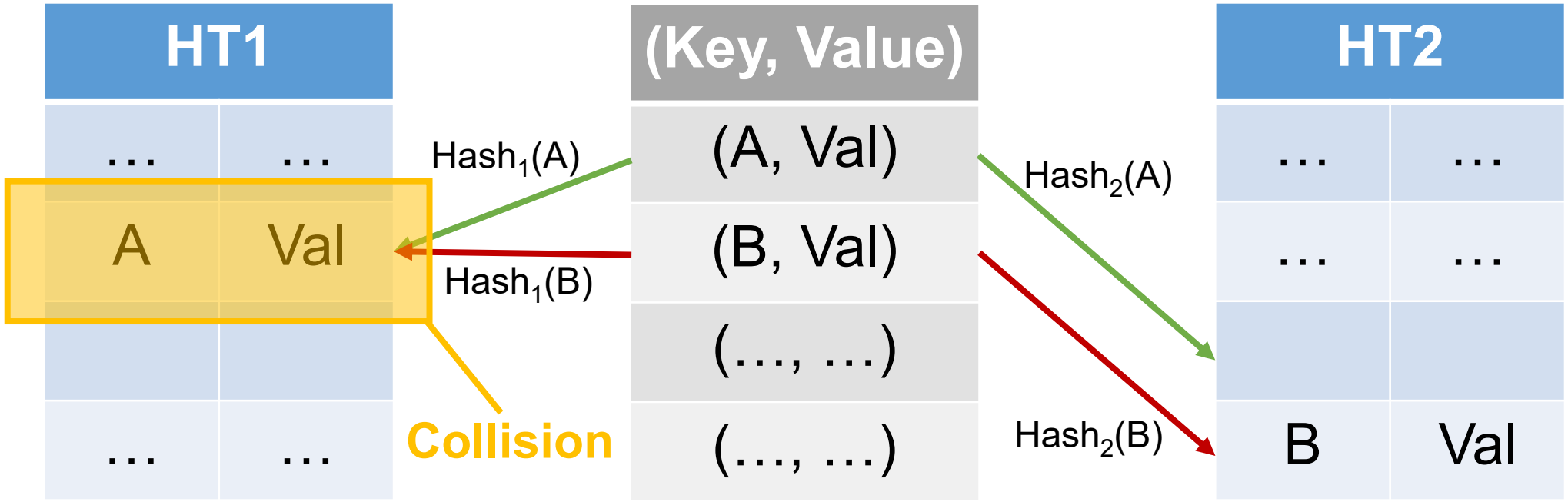


Most PM hash tables are also based on

Static Scheme – Linear Probing



Static Scheme – Cuckoo Hashing



From Static to Dynamic

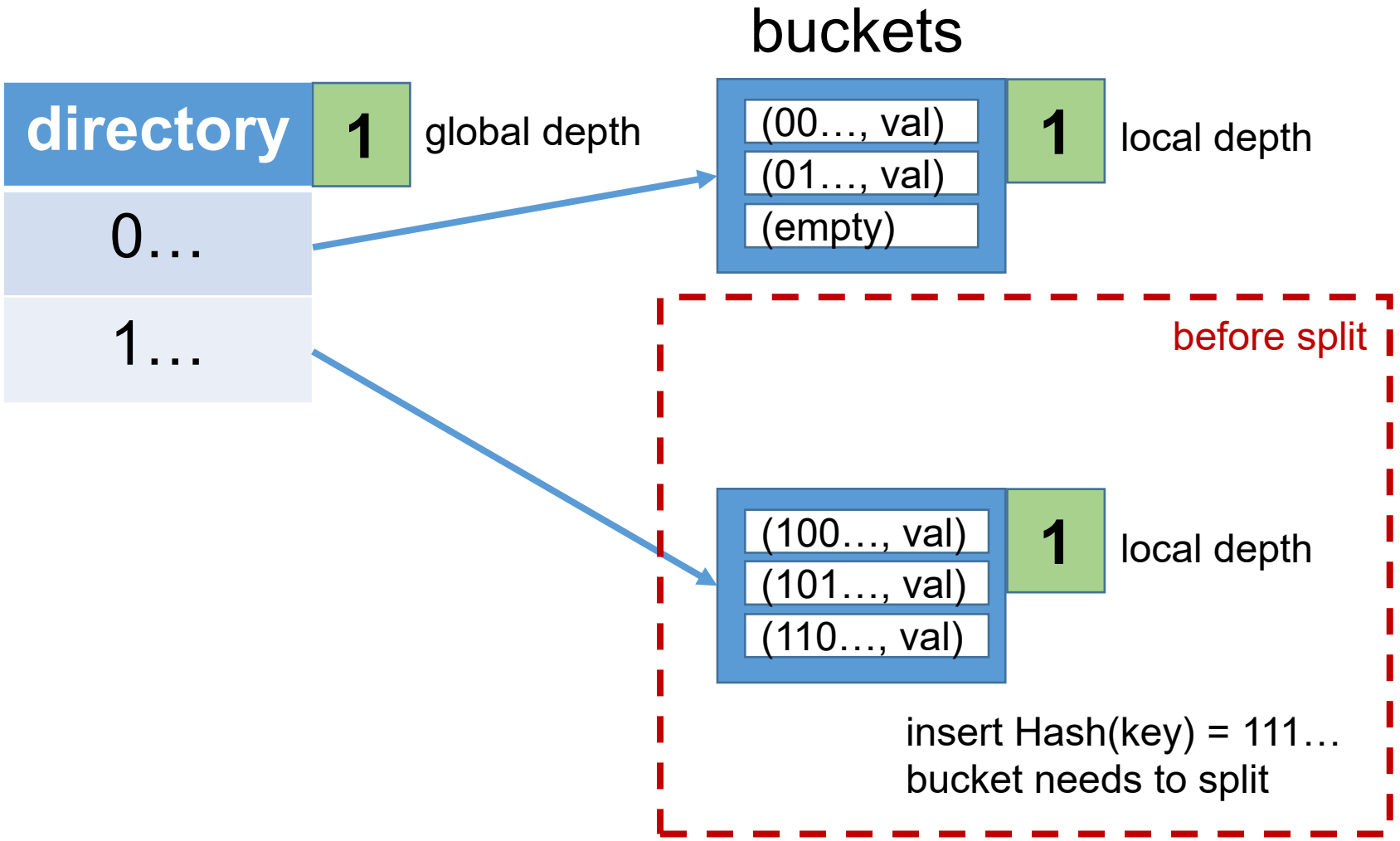
So, we need to rebuild the entire hash table when it is full.

But, rebuilding a hash table is very expensive even for DRAM.

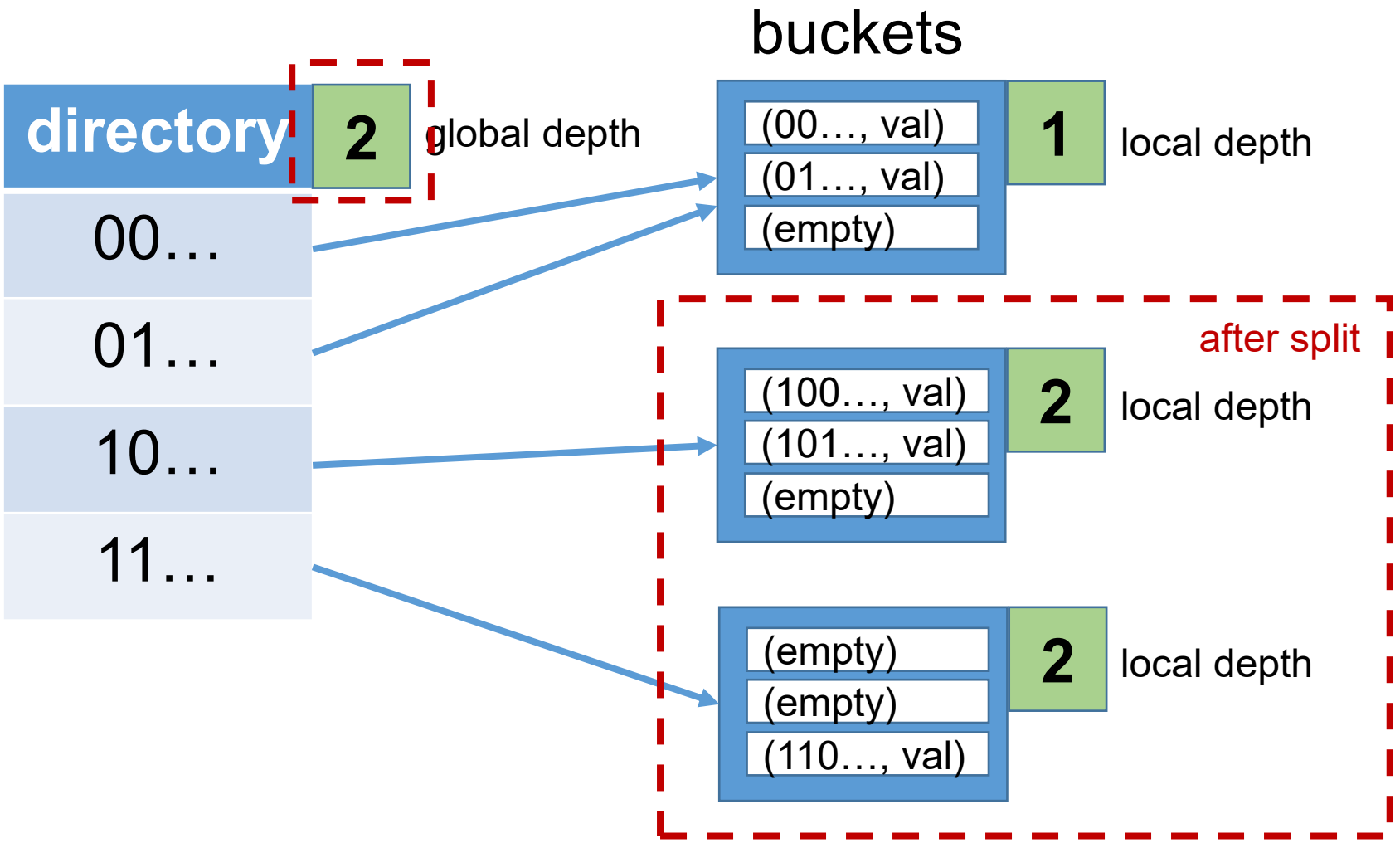
How to smooth out the process?

Dynamic hashing schemes.

Dynamic Scheme – Extendible Hashing



Dynamic Scheme – Extendible Hashing



Dynamic Scheme – From Extendible to Linear

So, we need to double the size of the directory when a bucket splits.

Oof, can we make the growth even smoother?

Linear hashing scheme. (Similar challenges in practice)

Hash Tables on PM

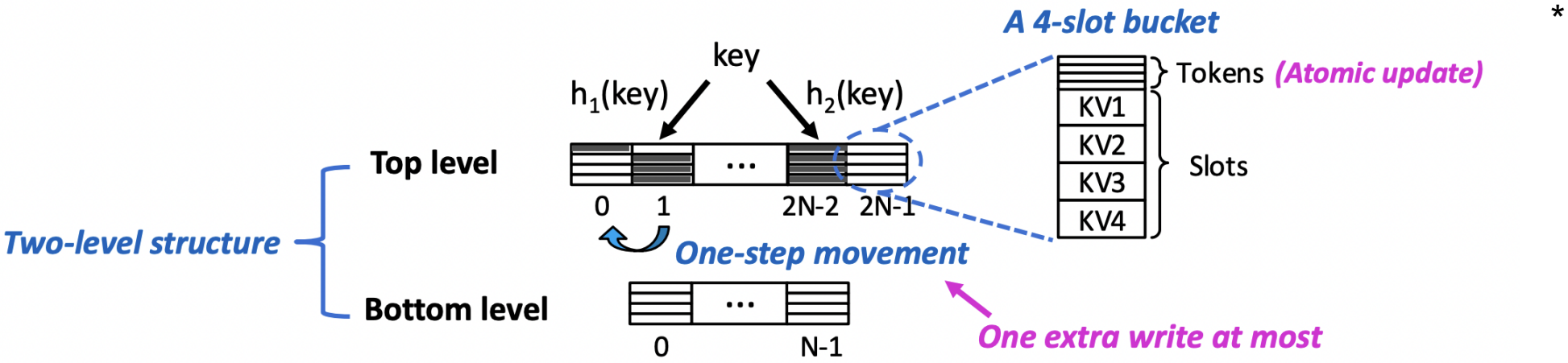
- Static hashing variants
  - Level hashing [27]
  - Clevel [28]

- Dynamic hashing variants
  - Cacheline-conscious extendible hashing (CCEH) [29]
  - Dash [30]

[27] Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory, OSDI ’18
[28] Lock-free Concurrent Level Hashing for Persistent Memory, ATC ’20
[29] Write-Optimized Dynamic Hashing for Persistent Memory, FAST ’19
[30] Dash: scalable hashing on persistent memory, VLDB ’20

Pre-Optane, Static: Level Hashing [27]

- Emulation-based bucketized cuckoo hashing
- Challenge #1: heavyweight consistency guarantee
  - Overcome by atomic token update
- Challenge #2: excessive PM write
  - Overcome by two-level bucketized hash table & in-place resizing



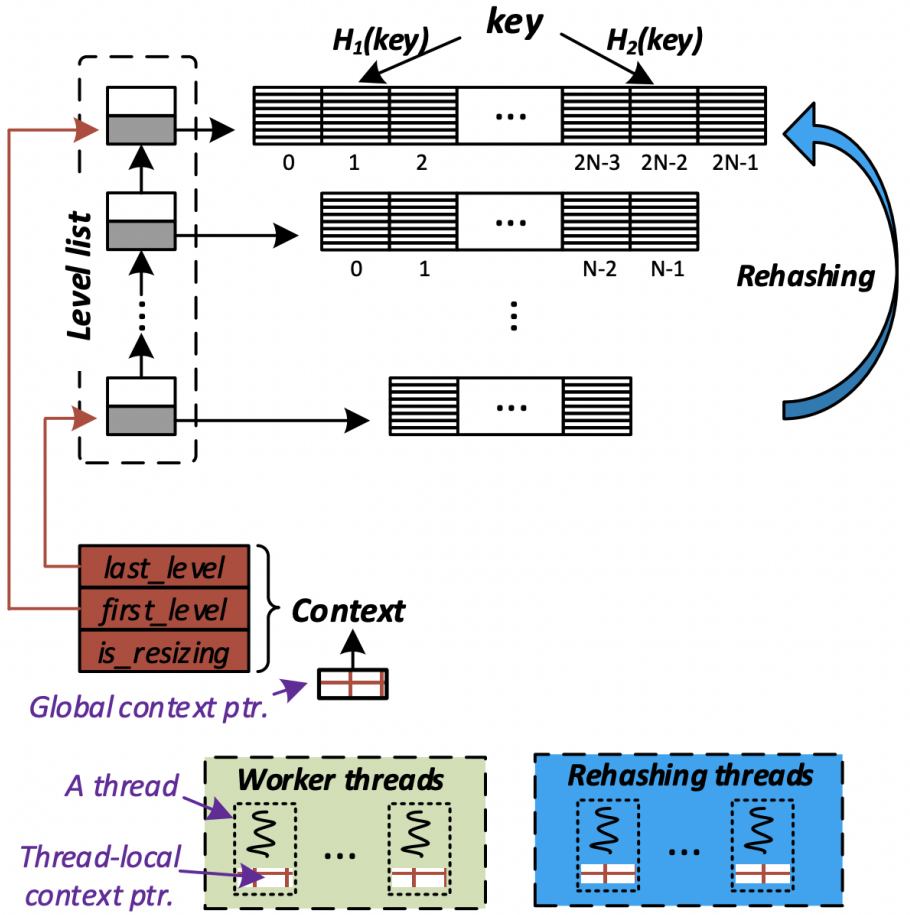
*

[27] Write-Optimized and High-Performance Hashing Index Scheme for Persistent Memory, OSDI '18

**https://www.usenix.org/system/files/atc20-paper227-slides-chen.pdf*

Optane, Static + Resizing: Clevel [28]

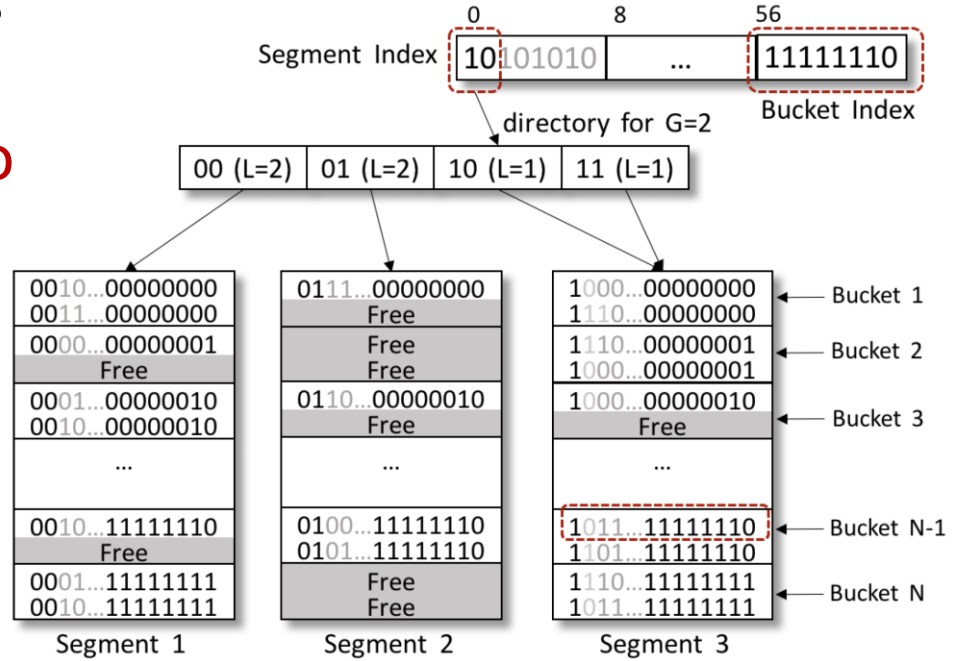
- Lock-free concurrent level hashing implemented on Optane PMem
- **Based** on level hashing with new challenges!
- Challenge #1: performance degradation during resizing
  - Overcome by replacing coarse-grained locks in level hashing with async rehashing
- Challenge #2: poor scalability of level hashing
  - Overcome by lock-free search/insertion/update/deletion



[28] Lock-free Concurrent Level Hashing for Persistent Memory, ATC '20

Pre-Optane, Extendible: CCEH [29]

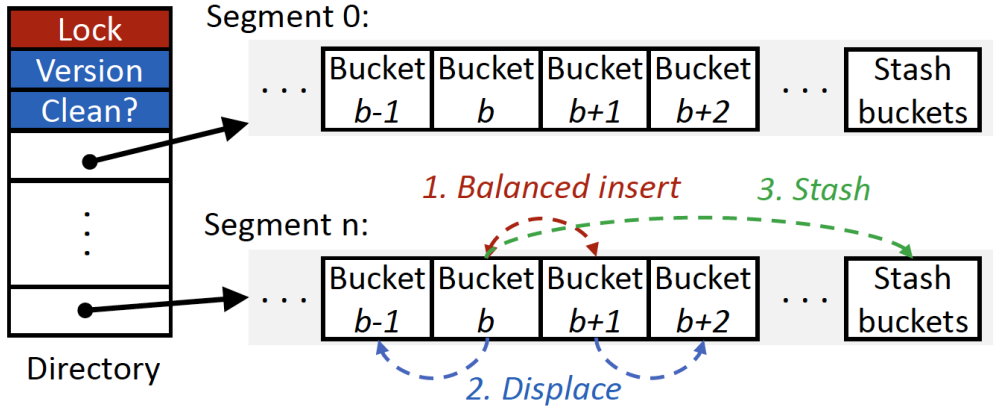
- Emulation-based extendible hashing
- Challenge #1: reducing cacheline accesses
  - Overcome by the three-level structure which makes sure that record can be found within two cacheline accesses
- Challenge #2: crash consistency
  - Overcome by keeping track of split history in the split buddy tree and reducing dirty writes through lazy deletion



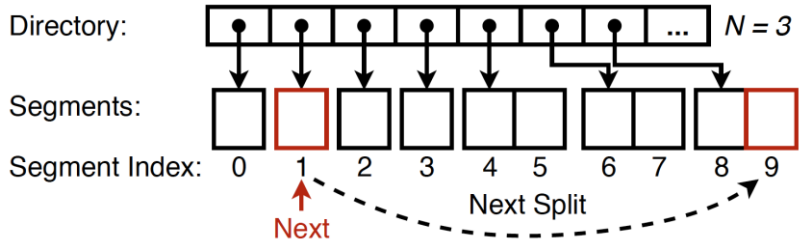
[29] Write-Optimized Dynamic Hashing for Persistent Memory, FAST '19

Optane, Extendible/Linear: Dash [30]

- Optane-based extendible/linear hashing
- Challenge #1: excessive PM read
  - For Optane, read latency > write latency
  - Overcome by fingerprint
- Challenge #2: heavyweight concurrency control (read-write lock)
  - Overcome by optimistic lock
- Challenge #3: load factor optimization
  - Overcome by bucket load balancing
- Challenge #4: instant recovery
  - Overcome by lazy recovery



(a) Overview of Dash-EH



(b) Overview of Dash-LH

[30] Dash: Scalable Hashing on Persistent Memory, VLDB 2020

PM Hash Tables: Design Summary

	Level Hashing	Clevel	CCEH	Dash
Reduce PM write	☑	☑	☑	☑
Reduce PM read				☑
Lightweight concurrency control		☑		☑
Lightweight consistency guarantee	☑	☑	☑	☑
Load factor optimization	☑	☑		☑
Resizing optimization	☑	☑		
NUMA optimization				
Instant recovery	☑	☑		☑
Variable-length key support				☑

Part 4: Implications and Outlook

Other Related/More-Recent Work

- This tutorial by no means exhaustive
  - Still fast evolving

- Some more recent PM indexes
  - Tree leveraging eADR: NB-Tree [34]
  - Hash table: Plush [35]

- PM key-value stores
  - Viper [38], FlatStore [39], Halo [40], etc.

- PMem-based full systems
  - Tair [36], OpenMLDB [37]

NEWS

Intel pulls the plug on Optane

Intel will wind down its storage class
elsewhere, while still supporting curr

By **Adam Armstrong,** News Writer

Home > Storage

Intel To Wind Down Optane Memory Business - 3D XPoint Storage Tech Reaches Its End

by Ryan Smith on July 28, 2022 5:00 PM EST

Posted in Storage | Intel | 3D XPoint | Optane | Optane Memory

Intel Kills Optane Memory Business, Pays $559 Million Inventory Write-Off

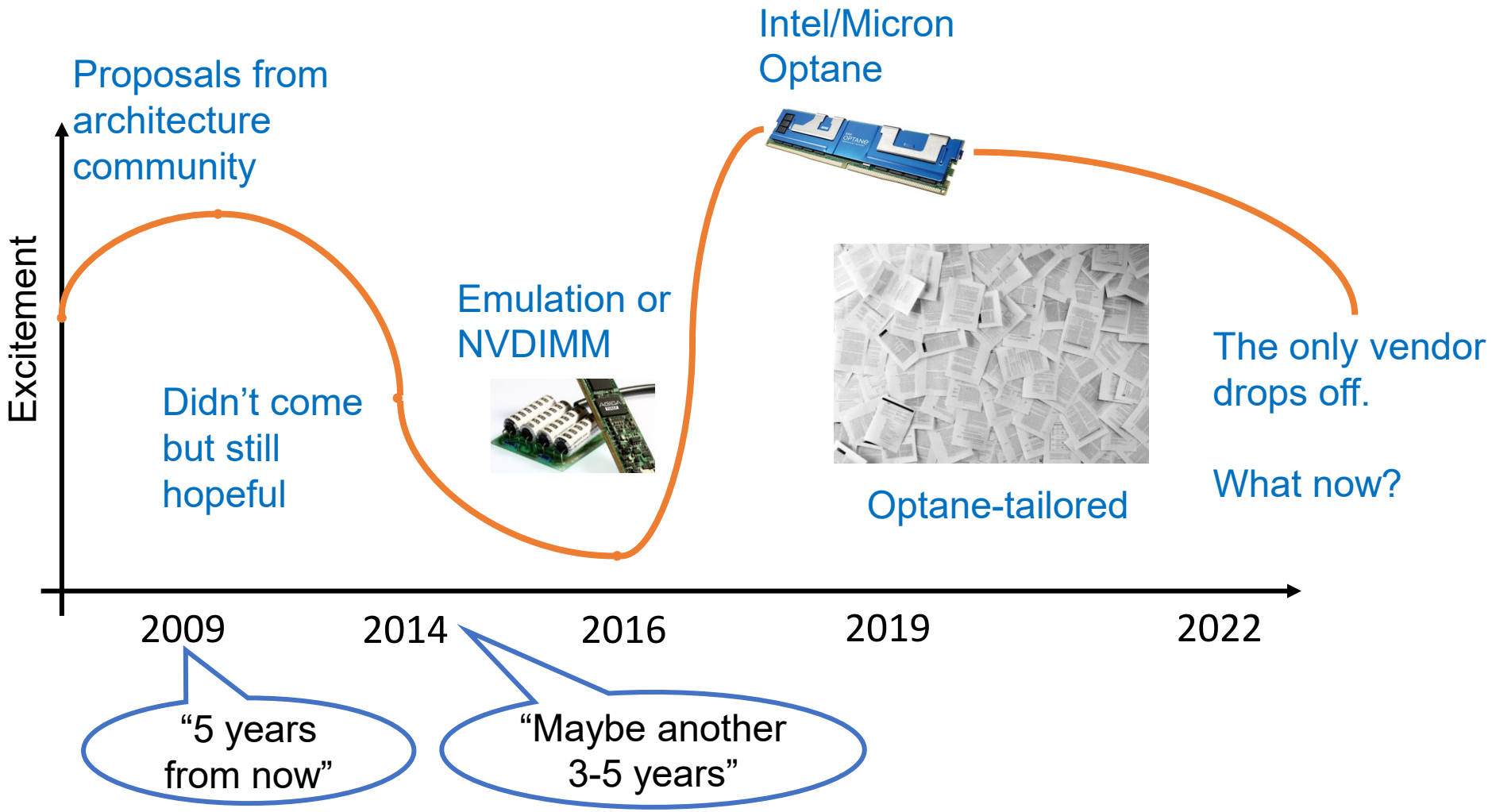
By Paul Alcorn published July 28, 2022

3D XPoint at the last crossroad.

Not the first time

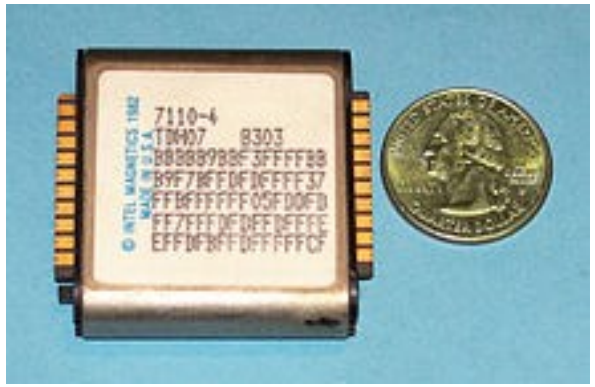
* https://www.techtarget.com/searchstorage/news/252523339/Intel-pulls-the-plug-on-Optane
* https://www.anandtech.com/show/17515/intel-to-wind-down-optane-memory-business
* https://www.tomshardware.com/news/intel-kills-optane-memory-business-for-good

Recent Timeline



Another Bubble Memory [32] (1960-1981)?

- Was the hope, like today's PM [31]
- Did not make it due to
  - Scalability/price
  - Complex to make
  - Require memory controller help
  - DRAM and magnetic disks caught up
- Optane with similar issues
  - Performance per $: lower than SSDs [33]
  - SSDs getting faster and faster
  - More complex memory controller
  - Single vendor



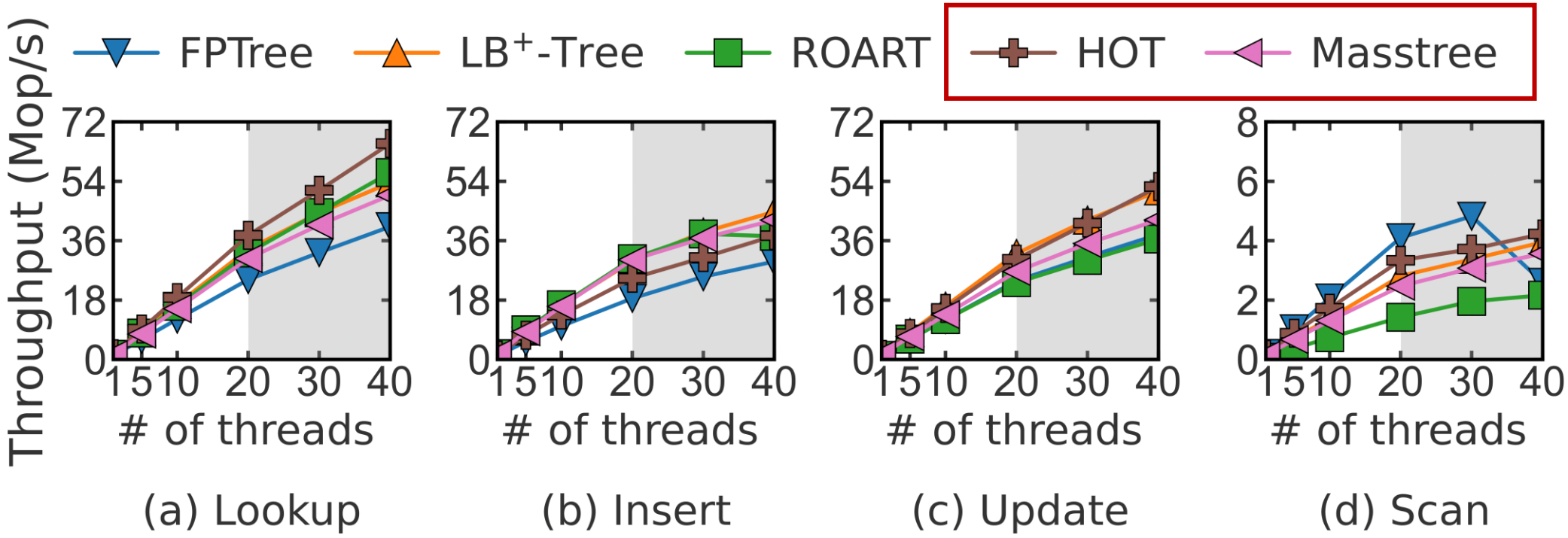
Intel (!) 7110 bubble memory*

Relevance of today's software techniques for PM?

* https://en.wikipedia.org/wiki/Bubble_memory#/media/File:Bubble_memory_module.jpg

Potential: PM Techniques on DRAM [26]

- Running PM range indexes on DRAM
  - No extra flushes/fences, using DRAM allocator



1. PM index techniques also effective for DRAM
2. Should focus more on full functionality [26]

Future: a Storage Jungle [33]

- NVDIMMs
  - Always been there
- Carbon Nanotubes: WIP
- Optane PMem
- Storage
  - Faster flash memory/SSDs
  - SSDs with memory semantics
  - 3D-stack DRAM
  - CXL enabling pooled memory

Thank you! + Q&A

References

[1] Memory Scaling is Dead,
Long Live Memory Scaling, Yale's "Mid Career" Celebration at University of Texas at Austin, Sept 19 2014
https://hps.ece.utexas.edu/yale75/qureshi_slides.pdf

[2] Strukov, D., Snider, G., Stewart, D. *et al.* The missing memristor found. *Nature* 453, 80–83 (2008).

[3] https://venturebeat.com/business/hp-enterprise-unveils-single-memory-160-terabyte-computer-the-machine/

[4] https://www.nextbigfuture.com/2020/11/memristors-and-hpe-machine-focused-computer-research-seems-almost-dead.html

[5] A. K. Mishra, X. Dong, G. Sun, Y. Xie, N. Vijaykrishnan, and C. R. Das. Architecting on-chip interconnects for stacked 3D STT-RAM caches in CMPs. In ISCA, 2011.

[6] B. Gervasi, "Will Carbon Nanotube Memory Replace DRAM?," in IEEE Micro, vol. 39, no. 2, pp. 45-51, 1 March-April 2019, doi: 10.1109/MM.2019.2897560.

[7] Ali Sheikholeslami and P. G. Gulak: A survey of circuit innovations in Ferroelectric random-access memories, *Proceedings of the IEEE,* Vol. 88, No. 3, pp. 667-689, May 2000

[8] H. . -S. P. Wong et al., "Phase Change Memory," in Proceedings of the IEEE, vol. 98, no. 12, pp. 2201-2227, Dec. 2010, doi: 10.1109/JPROC.2010.2070050.

[9] Viking Technology, Non-Volatile Memory and Its Use in Enterprise Applications, https://snia.org/sites/default/files/Non-Volatile%20Memory%20&%20Its%20Use%20in%20Enterprise%20Applications.pdf

[10] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. 2011. NV-Heaps: making persistent objects fast and safe with next-generation, non-volatile memories. In Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS XVI).

References

[11] Haris Volos, Andres Jaan Tack, and Michael M. Swift. 2011. Mnemosyne: lightweight persistent memory. In Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS XVI).

[12] Wang, Tianzheng & Levandoski, Justin & Larson, Per-Åke. (2018). Easy Lock-Free Indexing in Non-Volatile Memory. 461-472. 10.1109/ICDE.2018.00049.

[13] Viktor Leis, Alfons Kemper, and Thomas Neumann. 2013. The adaptive radix tree: ARTful indexing for main-memory databases. In Proceedings of the 2013 IEEE International Conference on Data Engineering (ICDE 2013) (ICDE '13). IEEE Computer Society, USA, 38–49. https://doi.org/10.1109/ICDE.2013.6544812

[14] Shimin Chen and Qin Jin. 2015. Persistent B+-trees in non-volatile main memory. Proc. VLDB Endow. 8, 7 (February 2015), 786–797. https://doi.org/10.14778/2752939.2752947

[15] Yang, Jun & Wei, Qingsong & Chen, Cheng & Wang, Chundong & Leong, Khai & He, Bingsheng. (2015). NV-Tree: Reducing Consistency Cost for NVM-based Single Level Systems.

[16] Joy Arulraj, Justin Levandoski, Umar Farooq Minhas, and Per-Ake Larson. 2018. Bztree: a high-performance latch-free range index for non-volatile memory. Proc. VLDB Endow. 11, 5 (January 2018), 553–565. https://doi.org/10.1145/3164135.3164147

[17] Ismail Oukid, Johan Lasperas, Anisoara Nica, Thomas Willhalm, and Wolfgang Lehner. 2016. FPTree: A Hybrid SCM-DRAM Persistent and Concurrent B-Tree for Storage Class Memory. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD '16). Association for Computing Machinery, New York, NY, USA, 371–386. https://doi.org/10.1145/2882903.2915251

[18] Lucas Lersch, Xiangpeng Hao, Ismail Oukid, Tianzheng Wang, and Thomas Willhalm. 2019. Evaluating persistent memory range indexes. Proc. VLDB Endow. 13, 4 (December 2019), 574–587. https://doi.org/10.14778/3372716.3372728

References

[19] Jihang Liu, Shimin Chen, and Lujun Wang. 2020. LB+Trees: optimizing persistent index performance on 3DXPoint memory. Proc. VLDB Endow. 13, 7 (March 2020), 1078–1090. https://doi.org/10.14778/3384345.3384355

[20] Youmin Chen, Youyou Lu, Kedong Fang, Qing Wang, and Jiwu Shu. 2020. UTree: a persistent B+-tree with low tail latency. Proc. VLDB Endow. 13, 12 (August 2020), 2634–2648. https://doi.org/10.14778/3407790.3407850

[21] Ma, Shaonan, Kang Chen, Shimin Chen, Mengxing Liu, Jianglang Zhu, Hong Kyu Kang and Yongwei Wu. “ROART: Range-query Optimized Persistent ART.” FAST (2021).

[22] Wook-Hee Kim, R. Madhava Krishnan, Xinwei Fu, Sanidhya Kashyap, and Changwoo Min. 2021. PACTree: A High Performance Persistent Range Index Using PAC Guidelines. In Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP '21). Association for Computing Machinery, New York, NY, USA, 424–439. https://doi.org/10.1145/3477132.3483589

[23] Xinjing Zhou, Lidan Shou, Ke Chen, Wei Hu, and Gang Chen. 2019. DPTree: differential indexing for persistent memory. Proc. VLDB Endow. 13, 4 (December 2019), 421–434. https://doi.org/10.14778/3372716.3372717

[24] Baotong Lu, Jialin Ding, Eric Lo, Umar Farooq Minhas, and Tianzheng Wang. 2021. APEX: a high-performance learned index on persistent memory. Proc. VLDB Endow. 15, 3 (November 2021), 597–610. https://doi.org/10.14778/3494124.3494141

[25] Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David Lomet, and Tim Kraska. 2020. ALEX: An Updatable Adaptive Learned Index. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 969–984. https://doi.org/10.1145/3318464.3389711

[26] Yuliang He, Duo Lu, Kaisong Huang and Tianzheng Wang, Evaluating Persistent Memory Range Indexes: Part Two. VLDB 2022

References

[27] Pengfei Zuo, Yu Hua, and Jie Wu. 2018. Write-optimized and high-performance hashing index scheme for persistent memory. In Proceedings of the 13th USENIX conference on Operating Systems Design and Implementation (OSDI'18). USENIX Association, USA, 461–476.

[28] Zhangyu Chen, Yu Hua, Bo Ding, and Pengfei Zuo. 2020. Lock-free concurrent level hashing for persistent memory. In Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC'20). USENIX Association, USA, Article 55, 799–812.

[29] Moohyeon Nam, Hokeun Cha, Young-Ri Choi, Sam H. Noh, and Beomseok Nam. 2019. Write-optimized dynamic hashing for persistent memory. In Proceedings of the 17th USENIX Conference on File and Storage Technologies (FAST'19). USENIX Association, USA, 31–44.

[30] Baotong Lu, Xiangpeng Hao, Tianzheng Wang, and Eric Lo. 2020. Dash: scalable hashing on persistent memory. Proc. VLDB Endow. 13, 8 (April 2020), 1147–1161. https://doi.org/10.14778/3389133.3389134

[31] W. Myers, "Key Developments in Computer Technology: A Survey," in Computer, vol. 9, no. 11, pp. 48-77, Nov. 1976, doi: 10.1109/C-M.1976.218441.

[32] Baker, K.F.: 'A review of magnetic bubble memories and their applications', Radio and Electronic Engineer, 1981, 51, (3), p. 105-115, DOI: 10.1049/ree.1981.0014IET Digital Library, https://digital-library.theiet.org/content/journals/10.1049/ree.1981.0014

[33] Kaisong Huang, Darien Imai, Tianzheng Wang and Dong Xie, SSDs Striking Back: The Storage Jungle and Its Implications on Persistent Indexes.12th Annual Conference on Innovative Data Systems Research (CIDR ’22). January 9-12, 2022, Chaminade, USA.

[34] Bowen Zhang, Shengan Zheng, Zhenlin Qi, and Linpeng Huang. 2022. NBTree: a lock-free PM-friendly persistent B+-tree for eADR-enabled PM systems. Proc. VLDB Endow. 15, 6 (February 2022), 1187–1200. https://doi.org/10.14778/3514061.3514066

[35] Plush: A Write-Optimized Persistent Log-Structured Hash-Table

References

[36] Tair-PMem: A Fully Durable Non-Volatile Memory Database

[37] Cheng Chen, Jun Yang, Mian Lu, Taize Wang, Zhao Zheng, Yuqiang Chen, Wenyuan Dai, Bingsheng He, Weng-Fai Wong, Guoan Wu, Yuping Zhao, and Andy Rudoff. 2021. Optimizing in-memory database engine for AI-powered on-line decision augmentation using persistent memory. Proc. VLDB Endow. 14, 5 (January 2021), 799–812. https://doi.org/10.14778/3446095.3446102

[38] Lawrence Benson, Hendrik Makait, and Tilmann Rabl. 2021. Viper: an efficient hybrid PMem-DRAM key-value store. Proc. VLDB Endow. 14, 9 (May 2021), 1544–1556. https://doi.org/10.14778/3461535.3461543

[39] Youmin Chen, Youyou Lu, Fan Yang, Qing Wang, Yang Wang, and Jiwu Shu. 2020. FlatStore: An Efficient Log-Structured Key-Value Storage Engine for Persistent Memory. In Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '20). Association for Computing Machinery, New York, NY, USA, 1077–1091. https://doi.org/10.1145/3373376.3378515

[40] Daokun Hu, Zhiwen Chen, Wenkui Che, Jianhua Sun, and Hao Chen. 2022. Halo: A Hybrid PMem-DRAM Persistent Hash Index with Fast Recovery. In Proceedings of the 2022 International Conference on Management of Data (SIGMOD '22). Association for Computing Machinery, New York, NY, USA, 1049–1063. https://doi.org/10.1145/3514221.3517884

[41] Jian Yang, Juno Kim, Morteza Hoseinzadeh, Joseph Izraelevitz, and Steven Swanson. 2020. An empirical guide to the behavior and use of scalable persistent memory. In Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST'20). USENIX Association, USA, 169–182.