

# Evaluating Persistent Memory Range Indexes: Part Two

Yuliang (George) He

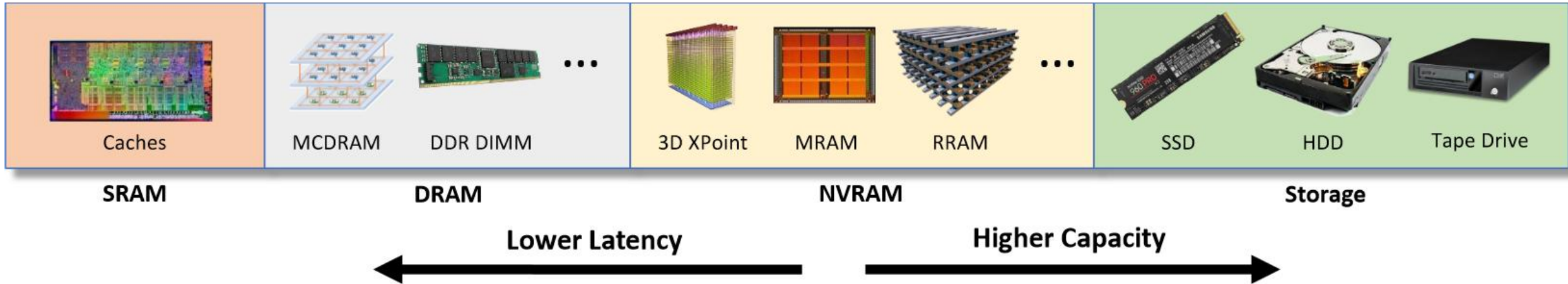
Duo Lu

Kaisong Huang

Tianzheng Wang



# The Persistent Memory (PM) Landscape



\* On the Diversity of Memory and Storage Technologies, I. Oukid, L. Lersch, Datenbank-Spektrum, 2018

Byte-addressable, durability on the memory bus

## WIPs

- STT-MRAM, Carbon NanoTube

## Now on the market

- Intel Optane DCPMM
- NVDIMMs

## Properties (except NVDIMM-N):

- + Energy efficient
- + Scales, high density, cheaper
- Higher read/write latency
- Read/write asymmetry
- Limited lifetime

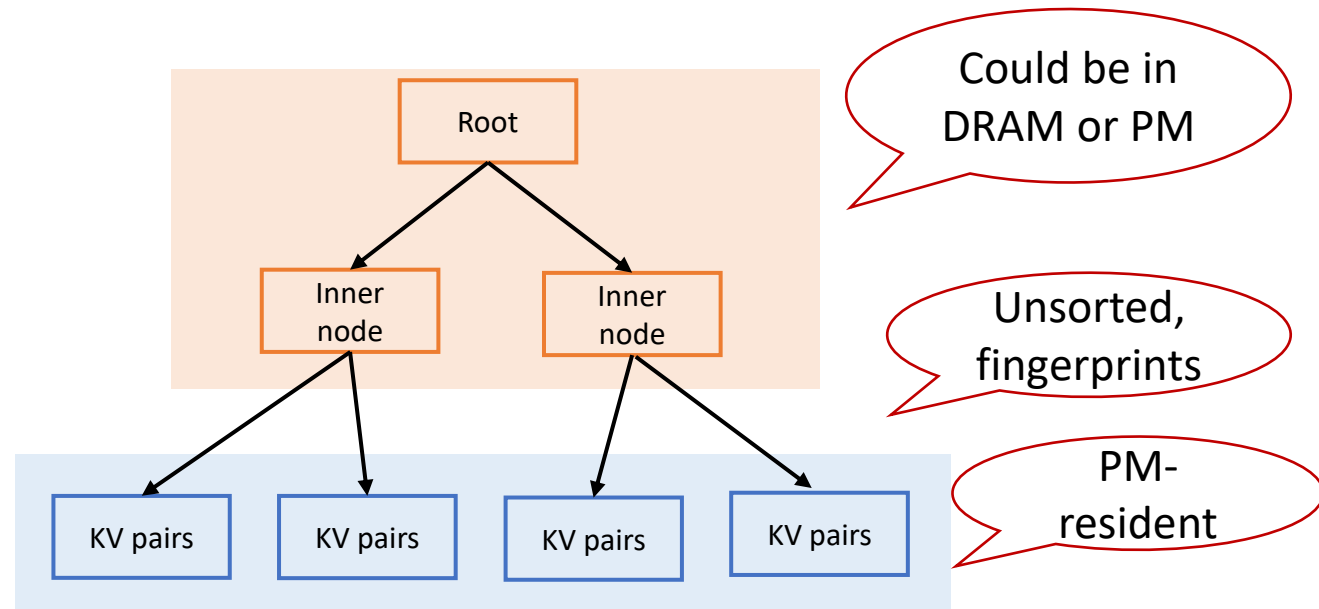
# Range Indexes on Persistent Memory

## Single level storage:

- Persist data on PM without I/O
- No serialization/deserialization cost
- Indexing for larger datasets
- Instant recovery

## Challenges:

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



# Previously on PM Range Indexes\* (Pre-2019)

- 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



*Key takeaways: should save bandwidth + leverage DRAM + fingerprinting*

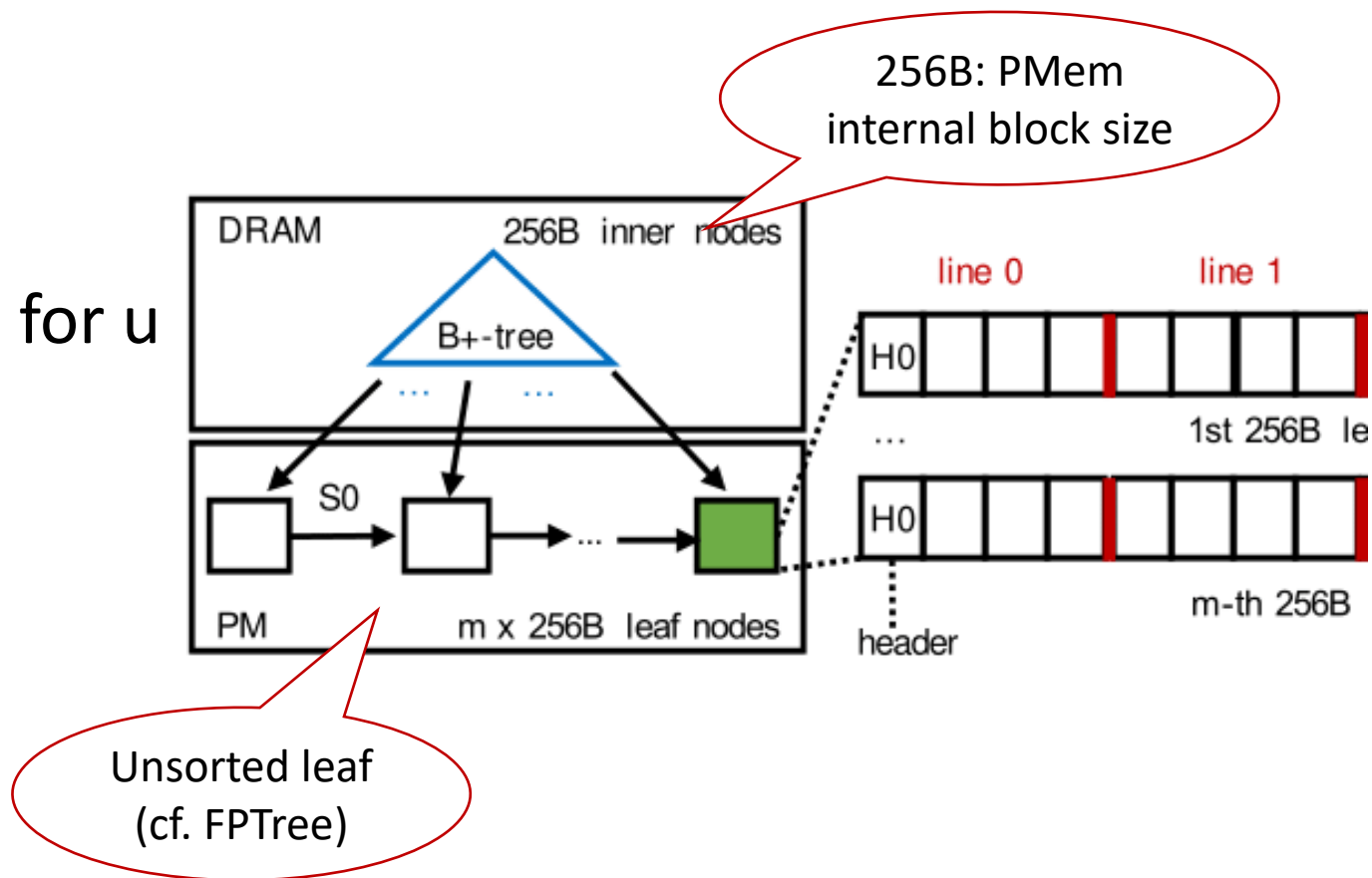
\* Evaluating Persistent Memory Range Indexes, VLDB 2020

# This Episode: PM Range Indexes 2019-2022

- Even more indexes
  - 10s of papers in VLDB/SIGMOD/SOSP, etc.
- How do they compare against each other?
- How are they different/similar from previous work?
  - Are they really better?
- What further challenges and opportunities remain?
- Optane going away - should I still care?

# B+-tree variants: LB+-Tree\* and uTree

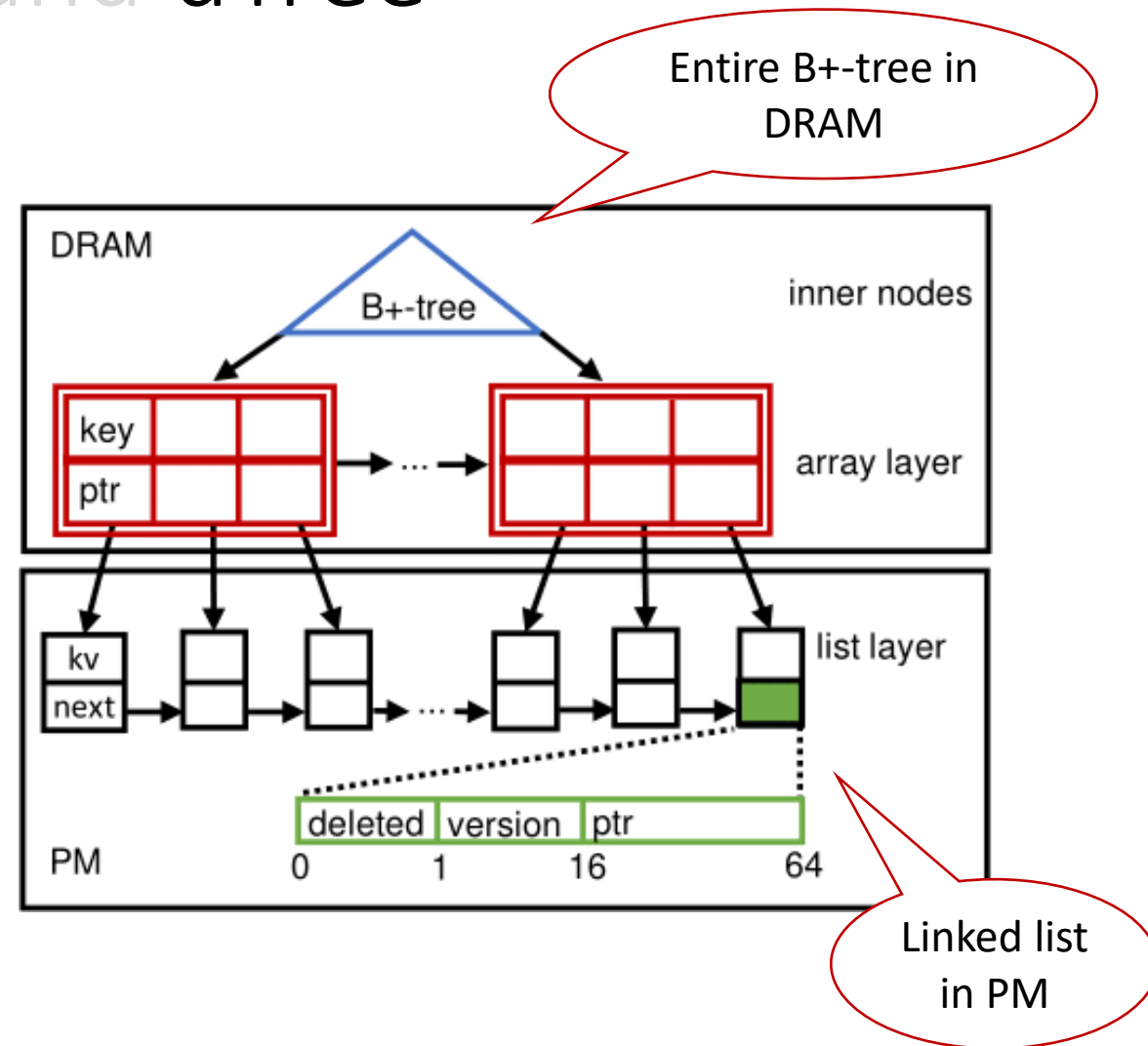
- Inner nodes in DRAM
- Leaf nodes in PM
- HTM for traversal, locking for u
- Techniques to avoid:
  - Cache misses
  - Logging overhead



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

# B+-tree variants: LB+-Tree and uTree\*

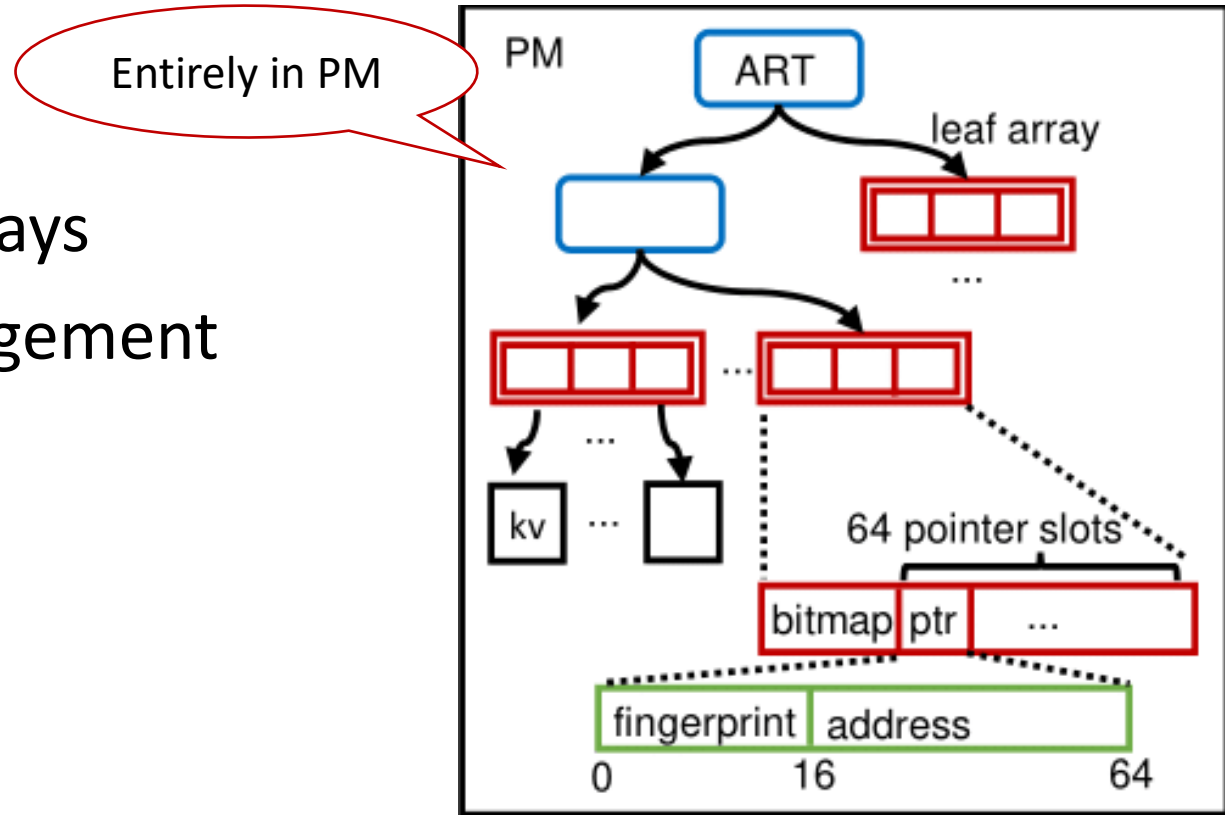
- Optimized for tail latency
- Coordinated concurrency control:
  - Traverse B+-Tree, find predecessor node
  - Update list layer using atomic CAS
  - Lock array layer leaf and update entry



\*  $\mu$ Tree: a Persistent B+-Tree with Low Tail Latency, *VLDB 2020*

# Trie variants: ROART\* and PACTree

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

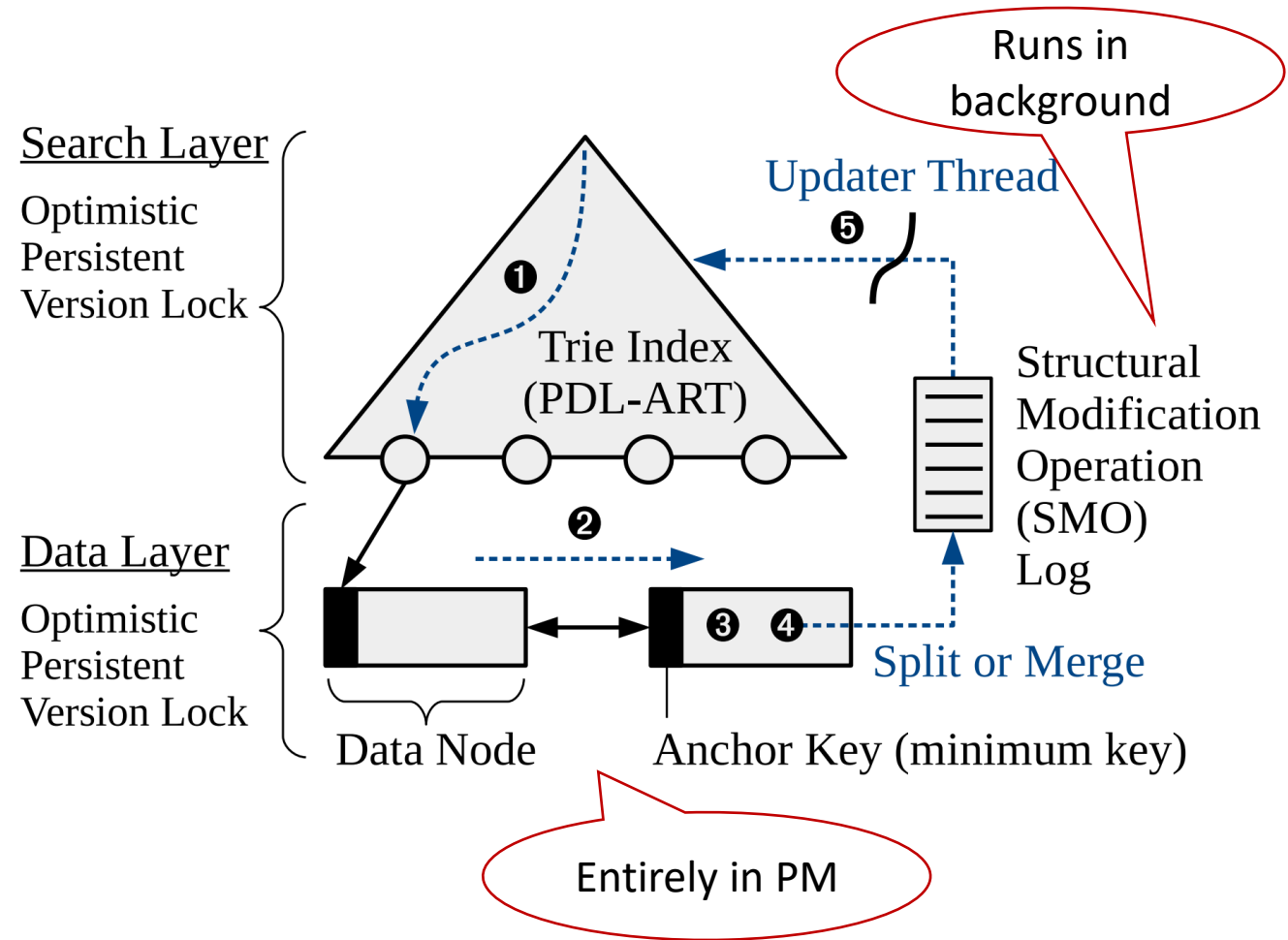


\*ROART: Range-query Optimized Persistent ART, *FAST 2021*



# Trie variants: ROART and PACTree\*

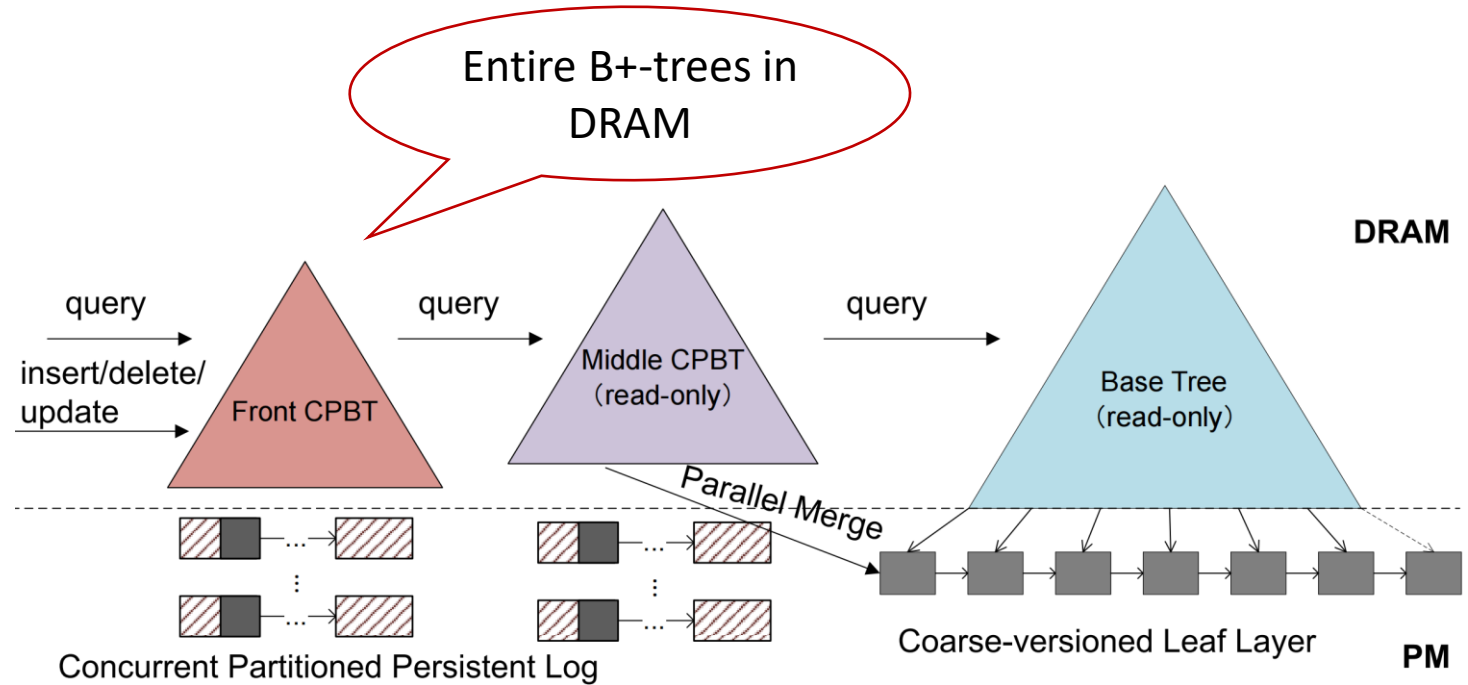
- Search layer: persistent trie
- Data layer: linked list of leaves
- NUMA-aware
  - Per-node PM pool
- Asynchronous update
  - SMOs by background threads
- Concurrency
  - ROWEX for search layer
  - Locking for data layer



\* PACTree: A High Performance Persistent Range Index Using PAC Guidelines, *SOSP 2021*


# Hybrid: DPTree\*

- Front Buffer Tree
  - B+-tree
  - For updates with logging
- Base Tree
  - Trie for inner nodes
  - B+-Tree style leaf nodes
  - Accumulates front buffer trees



\*DPTree: differential indexing for persistent memory, VLDB 2020

# Design Summary

	Architecture	Node structure	Concurrency
<b>LB+-Tree [VLDB 20]</b>	B+-tree; DRAM (inner) + PM (leaf)	Unsorted leaf; fingerprints; extra metadata	HTM (inner) + locking (leaf)
<b>uTree [VLDB 20]</b>	B+-tree; DRAM (B+-tree) + PM (linked list)	Sorted	Locking (array layer) + lock-free (list layer)
<b>DPTree [VLDB 20]</b>	Hybrid; DRAM (B+-tree, trie inner)+PM(trieleaf)	Unsorted leaf; fingerprints; indirection; extra metadata	Optimistic + async
<b>ROART [FAST 21]</b>	Trie; PM-only	B+-tree like unsorted leaf; fingerprints	ROWEX
<b>PACTree [SOSP 21]</b>	Trie; PM-only or optionally DRAM+PM	Unsorted leaf; fingerprints; indirection	Optimistic lock + async. Update
 <b>FPTree [SIGMOD 16]</b>	DRAM (inner nodes) + PM (leaf nodes)	Unsorted leaf nodes	Selective (HTM + locking)

**(largely) unsorted + extra metadata**

**(largely) optimistic**

Support var-keys

NUMA-optimized

# Experimental Setup

## Benchmarking Machine:

- 40-core dual-socket Xeon 6242R 3.1Ghz
- 384GB DRAM (12x32GB)
- 1.5TB Optane PMem 100 (12x128GB)
- Linux kernel 5.14.9

## Allocators

- jemalloc for DRAM allocation
- PMDK for PM allocation

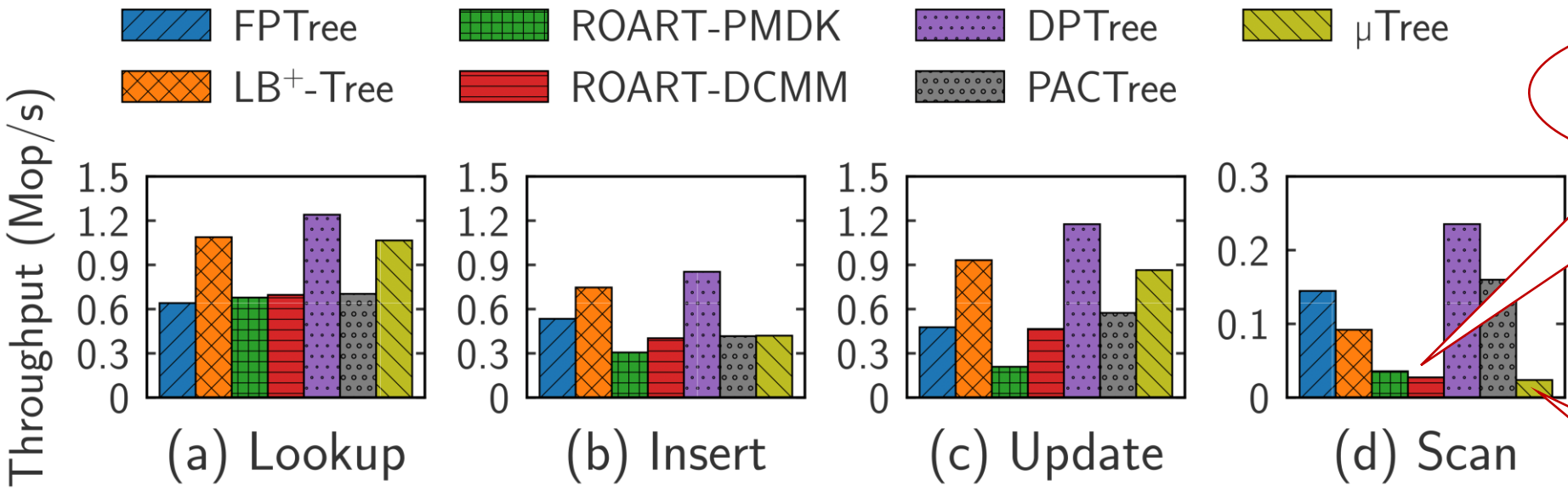
## Benchmarking Framework:

- PiBench [VLDB 2020]
- Metrics: throughput, latency, memory stats...
- Original authors' code as shared libraries

## Methodology:

1. Preload index with 100M 8B key/value pairs
2. Execute 10 seconds of operations

# The Old : Not over the hill yet



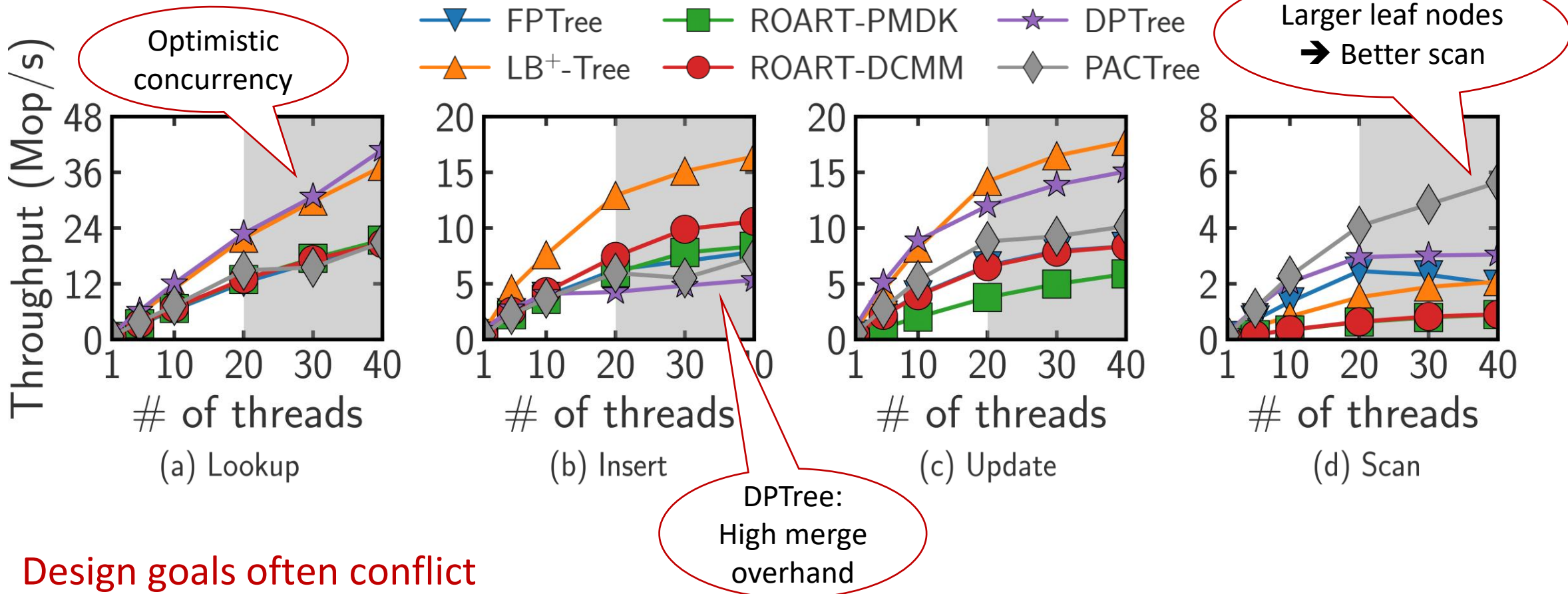
ROART: Under-utilized leaves

uTree: Pointer chasing overhead

FPTree still very competitive; new != better

New techniques + using DRAM (more aggressively) help a lot

# Scalability (Single-Socket)



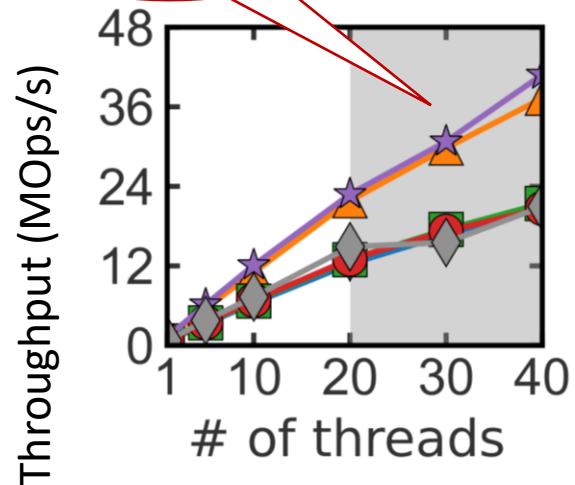
Design goals often conflict

E.g., LB+-Tree tops for all ops other than scan

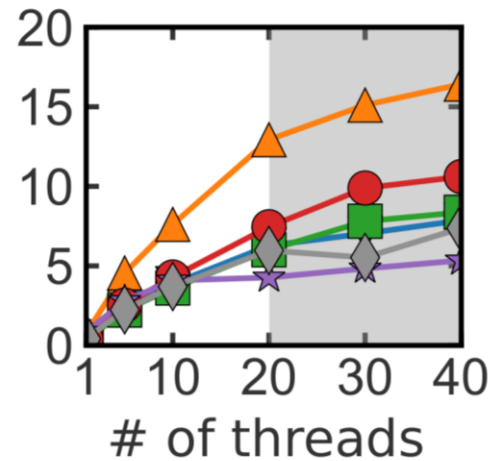
# Variable-length keys: Same 8B keys, opposite trends

8-byte integer keys

DPTree  
LB+-Tree



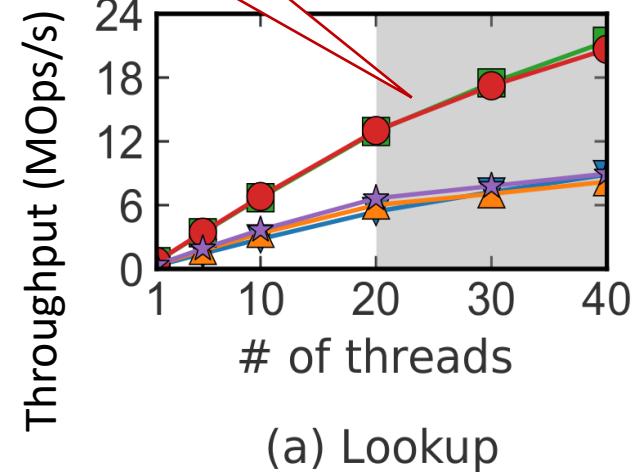
(a) Uniform Lookup



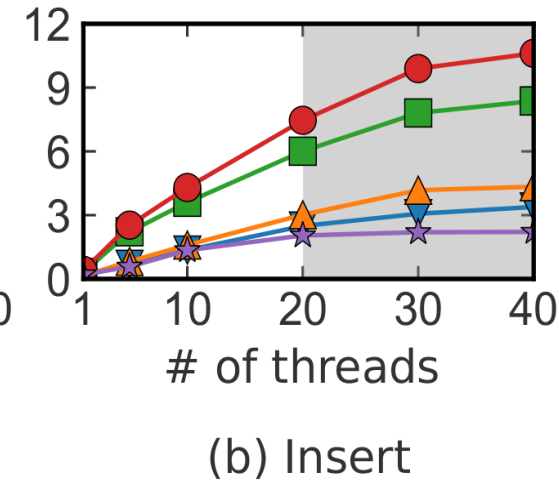
(b) Uniform Insert

8-byte string keys

ROART  
(trie)



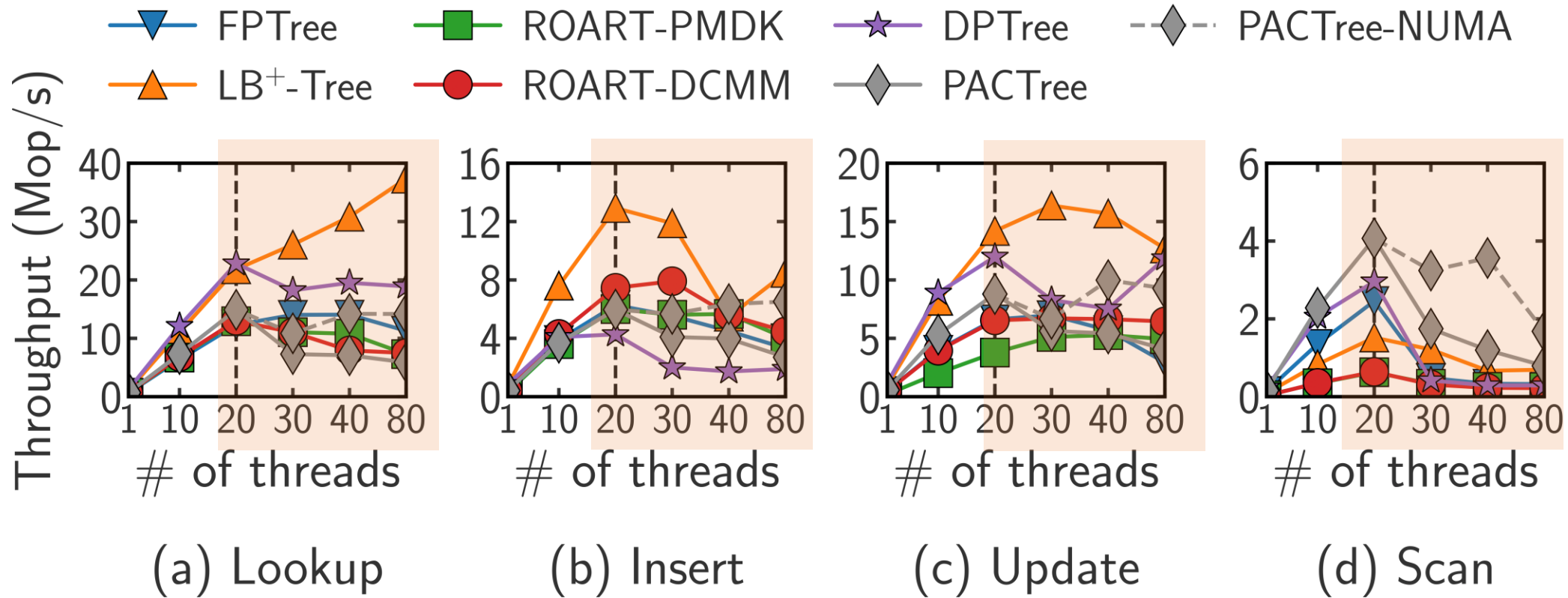
(a) Lookup



(b) Insert

May need to combine best of B+-trees and tries

# Handling NUMA: No Ideal Candidate

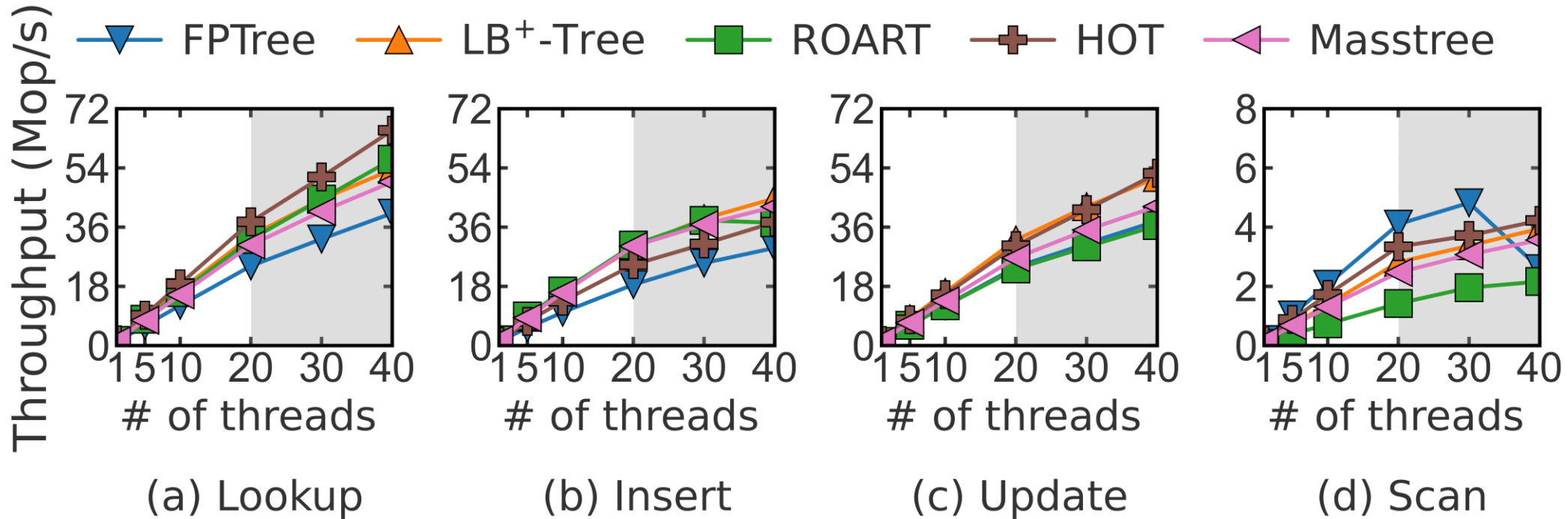


**Beyond one socket (20 threads), coherence traffic dominates**



# Life without Optane: 2022+

- Running PM range indexes on DRAM
  - Sans flushes and fences, using DRAM allocator



PM index technique also effective for DRAM

# Conclusion

- New/more complex designs != Better
  - Time to rethink and focus on simpler designs, if possible
- Still lacking in full functionality
  - NUMA effect
  - Variable length key support
- Why care even without Optane?
  - Using the techniques as DRAM optimizations
  - WIP PM media coming up

*More in our paper and code repo:*  
[github.com/sfu-dis/pibench-ep2](https://github.com/sfu-dis/pibench-ep2)

*Thank you!*