APEX: A High-Performance Learned Index on Persistent Memory

Baotong Lu[1], Jialin Ding[2], Eric Lo[1], Umar Farooq Minhas[3], Tianzheng Wang[4]

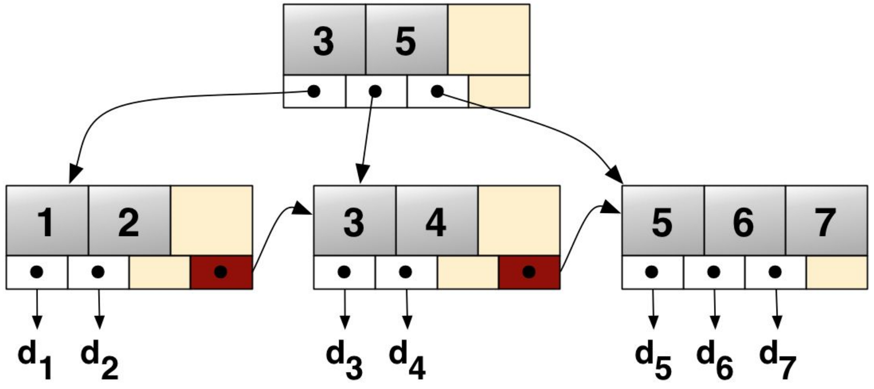
[1] The Chinese University of Hong Kong
[2] MIT
[3] Microsoft Research
[4] Simon Fraser University

Traditional B+-Trees vs. Learned Indexes

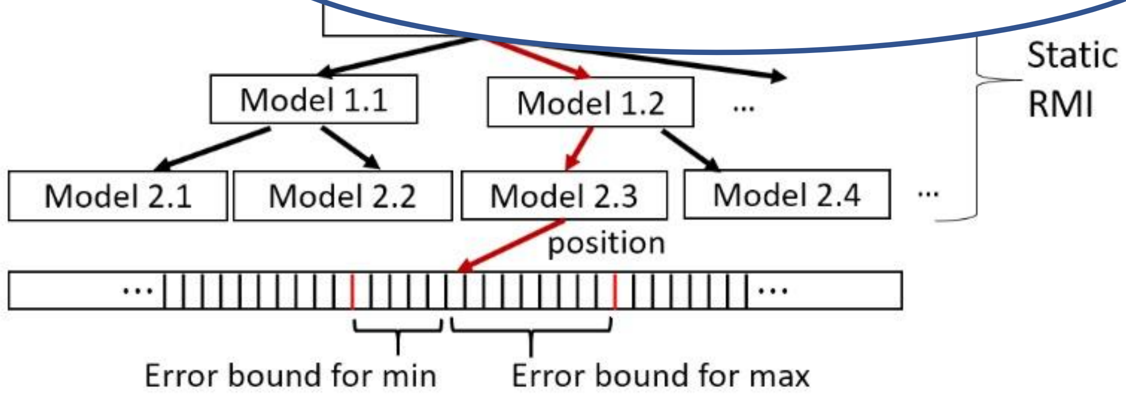
	B+-trees	Learned Indexes
Performance	*O*(logn), n = #items	• Data-distribution aware • They are ready [see Thursday 13:30 PM]
Architecture	In-memory or disk-based	In-memory

1. No persistence

2. Restricted by DRAM capacity



Source: wikipedia.org



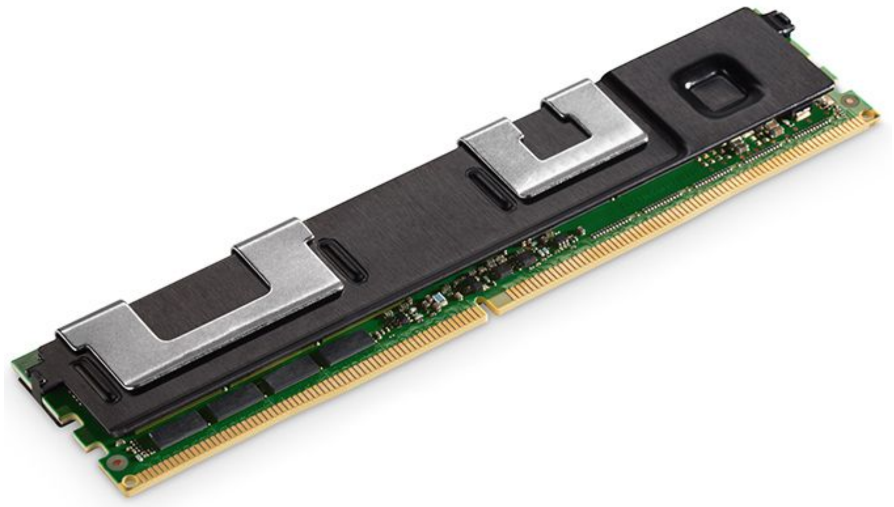
Source: Kraska et al, SIGMOD 2018

Persistent Memory Primer

Persistent Memory (PM)

- Larger capacity (vs. DRAM)
- Lower price (vs. DRAM)
- Byte-addressable
- Persistent

Instant Recovery!



Intel Optane DCPMM

- Properties: **high latency** & **limited bandwidth**
- Existing PM indexes: B+-tree, hash table etc.

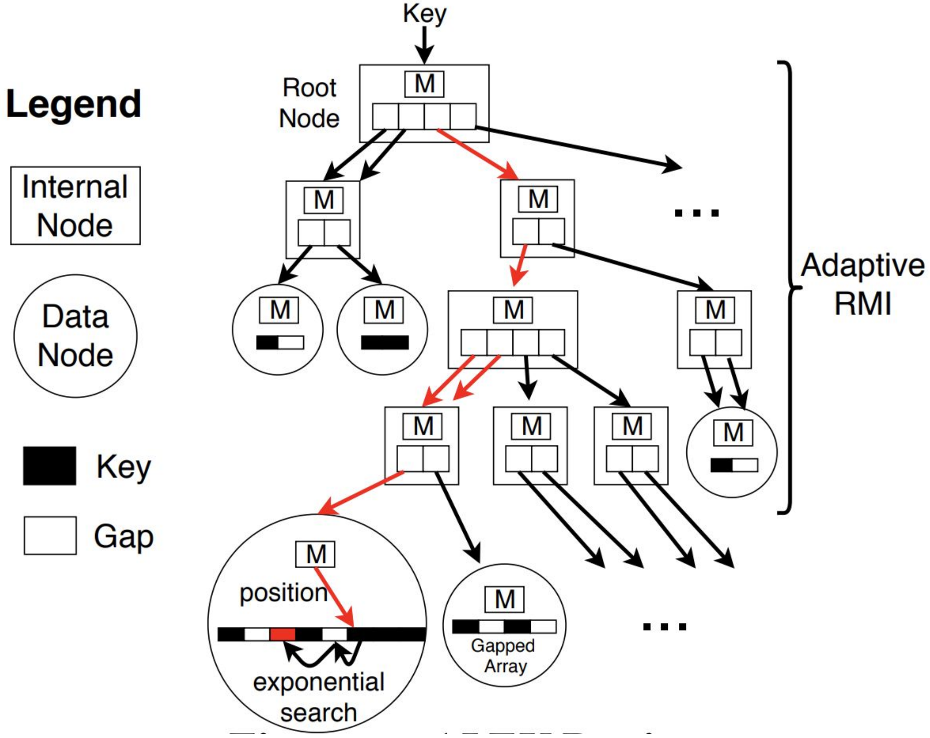
How to build an efficient learned Index on PM?

Challenge 1: Scalability

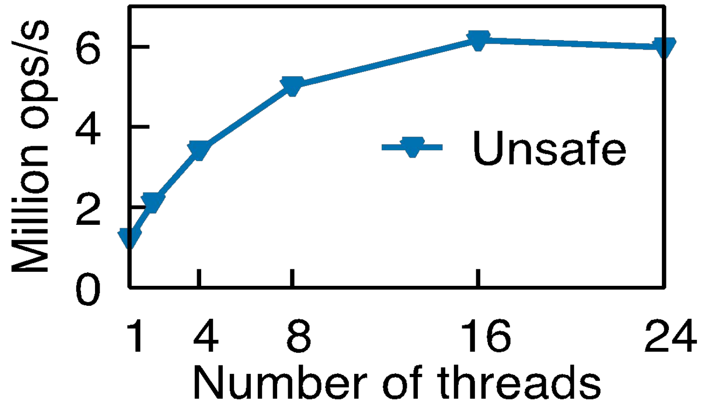
Three aspects inhibit the scalability on PM!

Start with ALEX [1]

- State-of-the-art learned index



	ALEX	B+-Tree	PM B+-Tree
Node design	Up to 16MB, sort	512B, sort	512B, unsort
1. PM writes	***O(m)***, *m = #KVs in one data node*	*O(m)*	*O(1)*
2. Concurrency	**Not Yet**	Optimistic CC	Optimistic CC
3. SMO Cost	**Large**	Small	Small

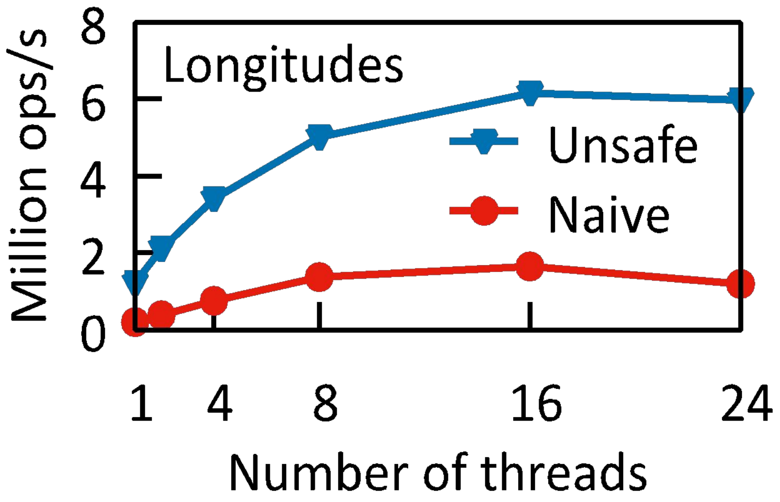


[1] ALEX: An Updatable Adaptive Learned Index ***SIGMOD'20***

Challenge 2: Crash Consistency & Instant Recovery

Old **crash consistency** tricks on PM do not work on learned Index

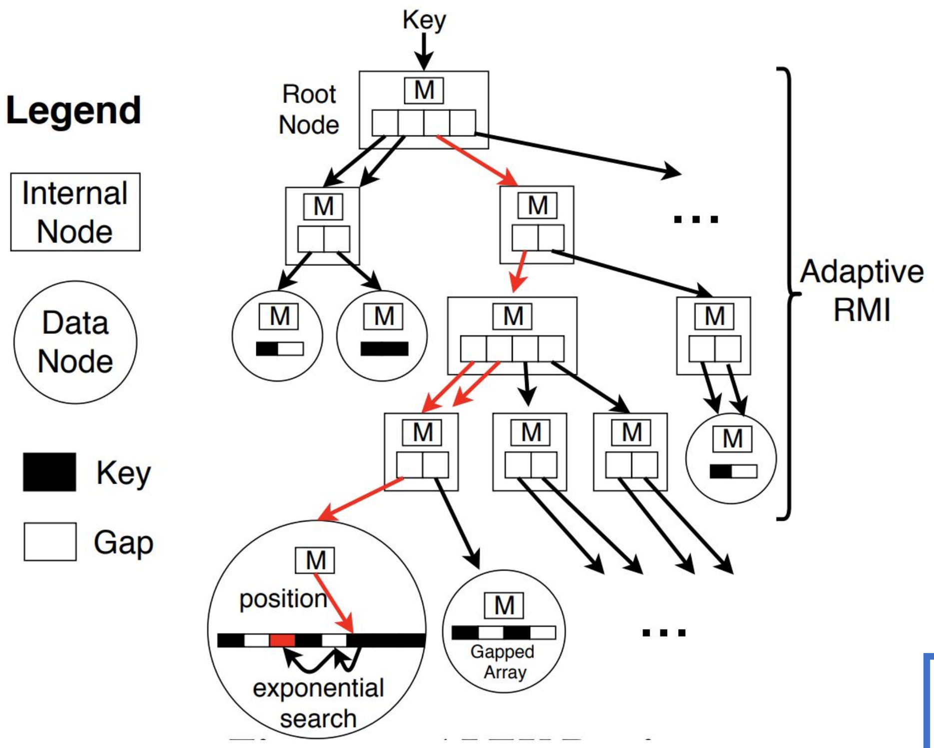
- General approaches (e.g., PMDK TXN) impose high overhead



Instant recovery vs. higher throughput with DRAM

- Being fully PM-resident hurts performance
- B+-Tree: inner nodes in DRAM, leaf nodes in PM

Design Principles



Overall structure of ALEX

ALEX

P1 Avoid excessive PM reads and writes

P2 Model-based operations

P3 Lightweight SMOs

P4 Judicious use of DRAM

P5 Crash consistency

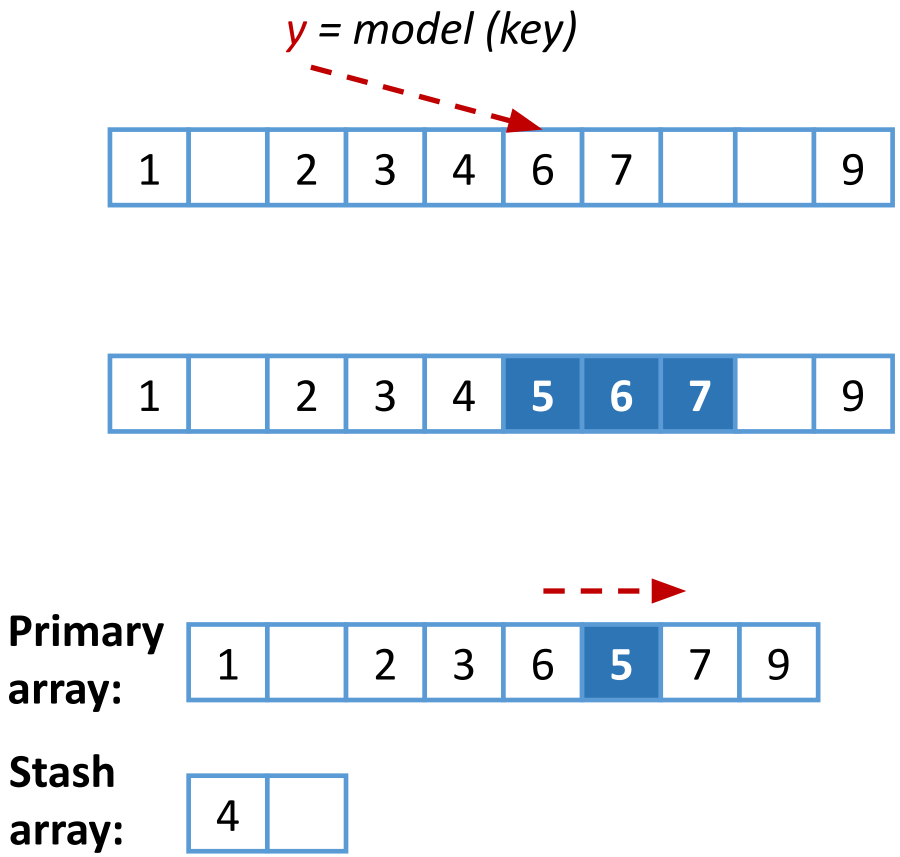
- Ideally, support instant recovery

APEX = Fast + Scalable + Instantly-Recoverable

Data Node Design: PM-Aware Optimizations

Insights

- Data node => hash table
- Linear model => hash function
- Need PM-aware collision-resolving strategy!



ALEX data node

- Fully sorted
- Element shifts => ***Excessive PM writes***

VS

APEX data node

- Nearly-sorted [1]
- Probe-and-stash => ***Mostly one PM write***

[1] Patience is a Virtue: Revisiting Merge and Sort on Modern Processors ***SIGMOD'14***

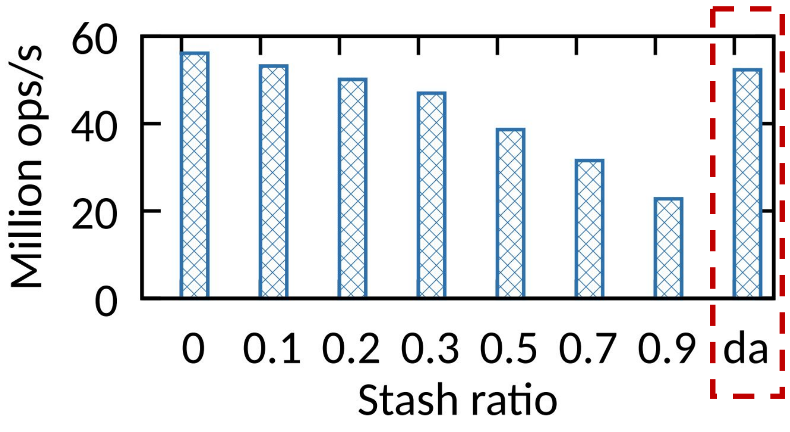
Data Node Design: Data-Aware Optimizations

Primary array (PA) size vs. stash array (SA) size?

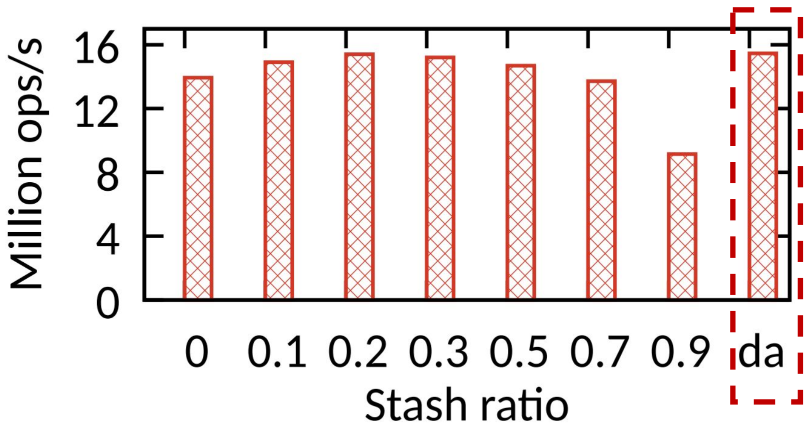
- More PA: facilitate model-based operations
- More SA: efficiently absorb the collisions

Distribution-aware (DA) approach

- "**Customize**" node layout according to the underlying data



Search (Longlat)



Insert (Longlat)

Low-overhead Data Consistency

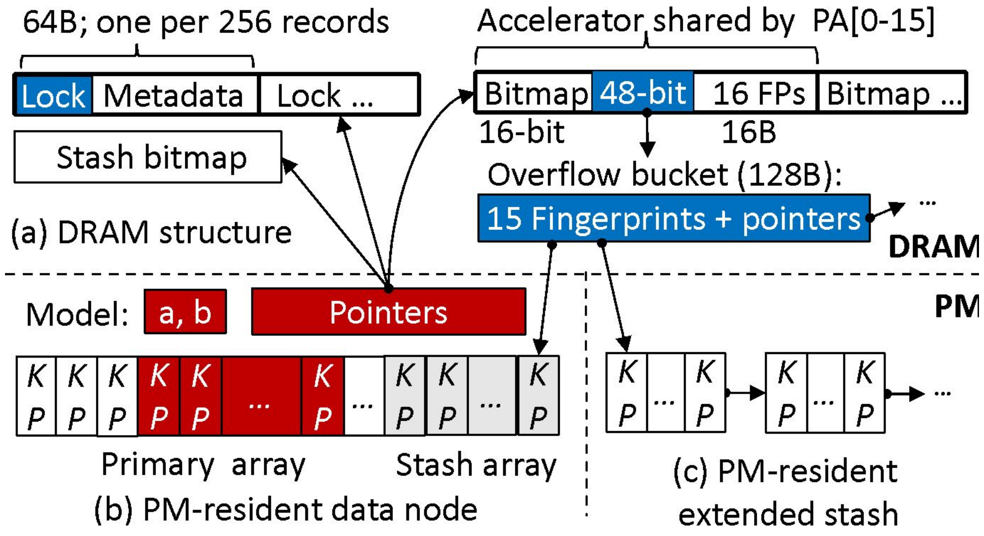
Insight: reduce the amount of data to be persistent as much as possible

Non-critical data => DRAM-resident

- Including: accelerator etc.
- Rebuilt upon recovery

Critical data => PM-resident

- Models, primary & stash arrays
- How to ensure consistency?
  - 'Key' as the valid indicator



Instant Recovery and Scalable Concurrency

Employ **lazy recovery** [1] to support instant recovery

- Index layer is kept in PM
- Re-construct the DRAM-resident data on demand by accessing threads

Scalable concurrency protocol

- **Hybrid maximum node size** to reduce SMO cost
- Adapt optimistic concurrency to learned index
  - Index traversal is lock-free

[1] Dash: Scalable Hashing on Persistent Memory ***VLDB'20***

Experimental Setup

Setting: 24-core CPU, 128GB X 6 Optane DCPMM (on all six channels)

Competitors

- BzTree, VLDB 2018
- FAST+FAIR, FAST 2018
- FPTree, SIGMOD 2016
- uTree, VLDB 2020
- LB+-tree, VLDB 2020
- DPTree, VLDB 2020

PM-only

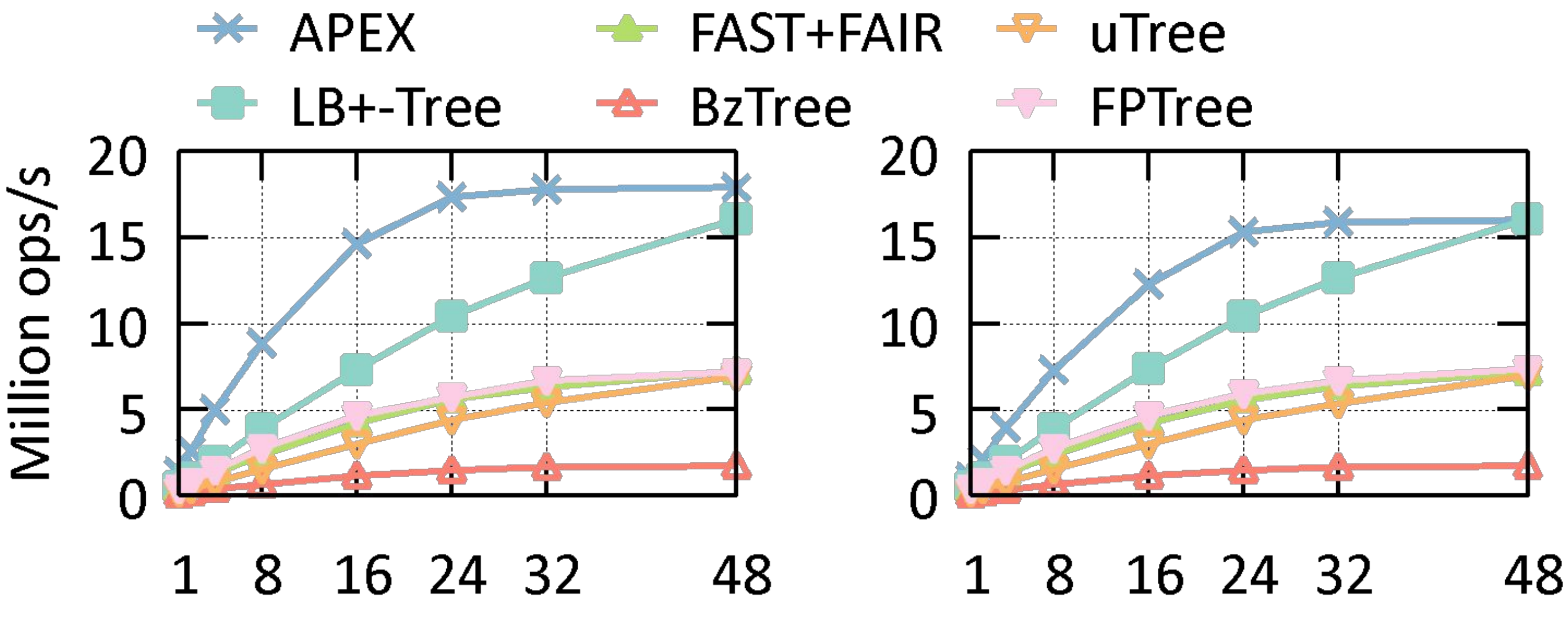
PM-DRAM

Datasets

- Six realistic and synthetic public datasets
  - Easy (Longitudes)/ Hard (Longlat)

Evaluation: Insert

- APEX: winner and bandwidth-bound

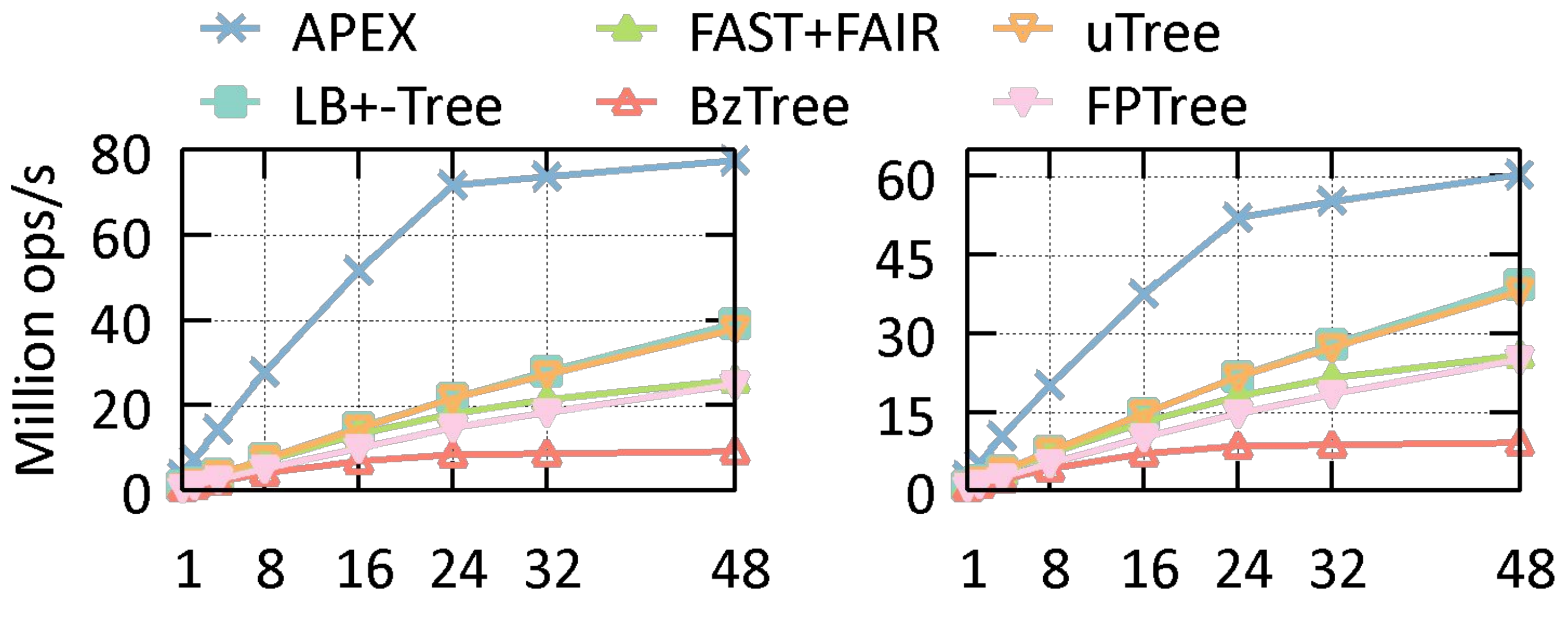


(a) Insert (easy).

(b) Insert (hard).

Evaluation: Search

- APEX: winner and near-Linear scalable (bandwidth-bound)



(a) Search (easy). (b) Search (hard).

Conclusions

- Designing learned indexes on PM is challenging both for performance and persistence.
- APEX: Five design principles for PM-optimized learned indexes
  - Retains learned indexes' advantages
  - Adapts proven PM indexes techniques
  - Instantly-recoverable
  - Combine the best of PM and machine learning

Open source at: https://github.com/baotonglu/apex

Thank you!