ROART: Range-query Optimized Persistent ART

Shaonan Ma[1], Kang Chen[1], Shimin Chen[2], Mengxing Liu[1], Jianglang Zhu[1], Hongbo Kang[1], and Yongwei Wu[1]

[1]Tsinghua University

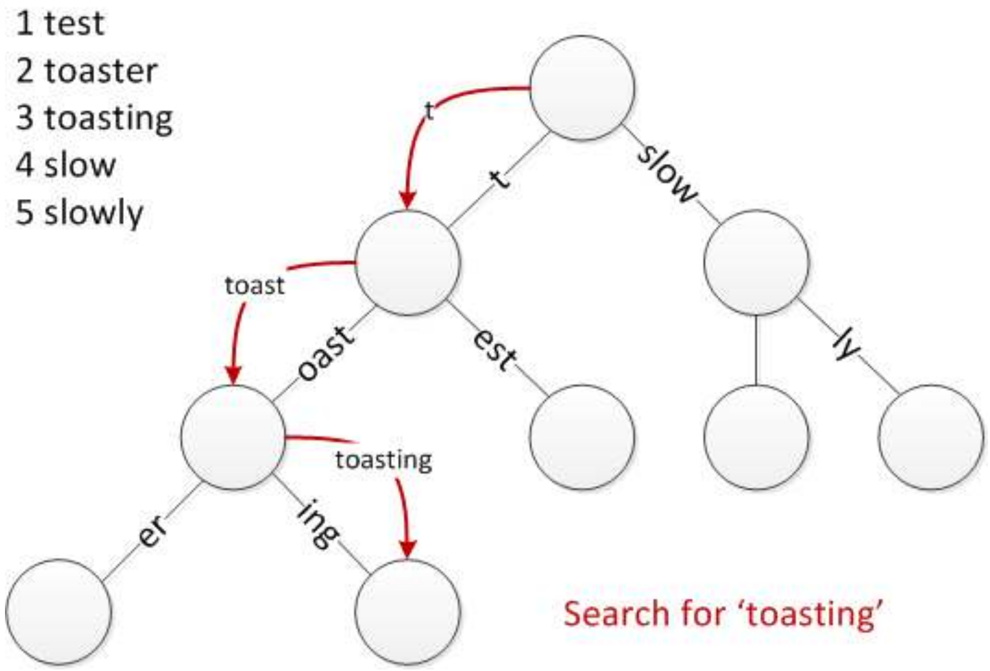
[2]SKL of Computer Architecture, ICT, CAS, and University of Chinese Academy of Sciences

FAST 2021

Background

➢ Radix Tree

- A space-optimized trie (prefix tree)
- Each internal node has **at most $\boldsymbol{r}$** children ($r = 2^n, n \geq 1$)
- Edge: single element or sequences of elements

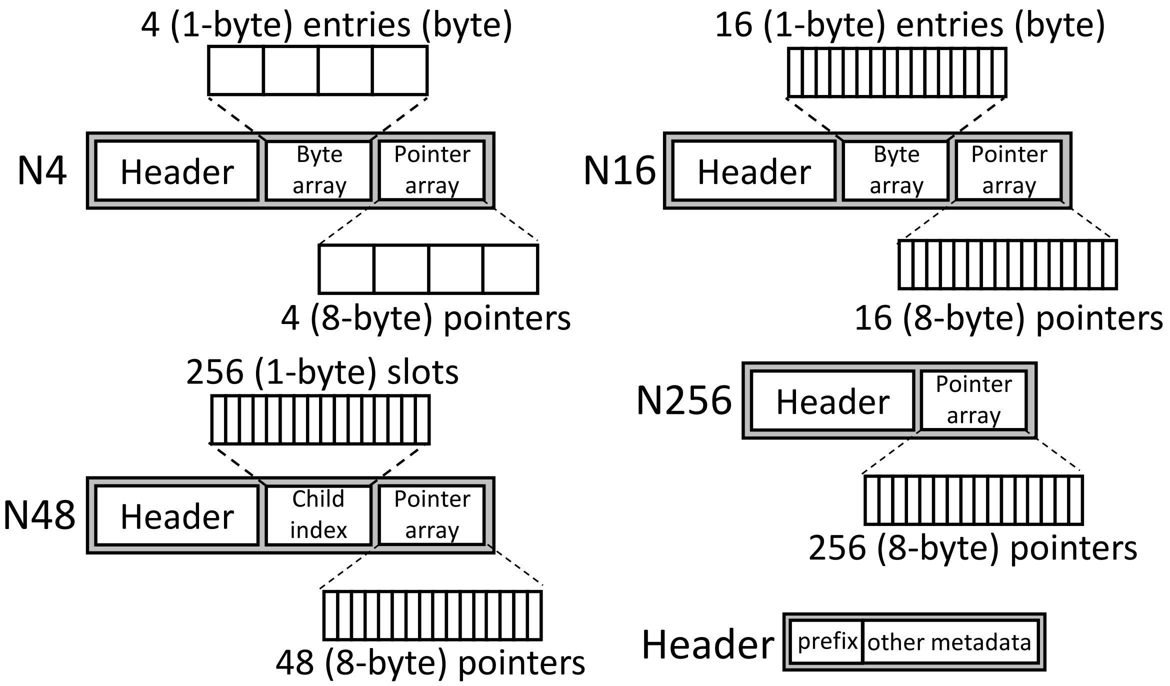


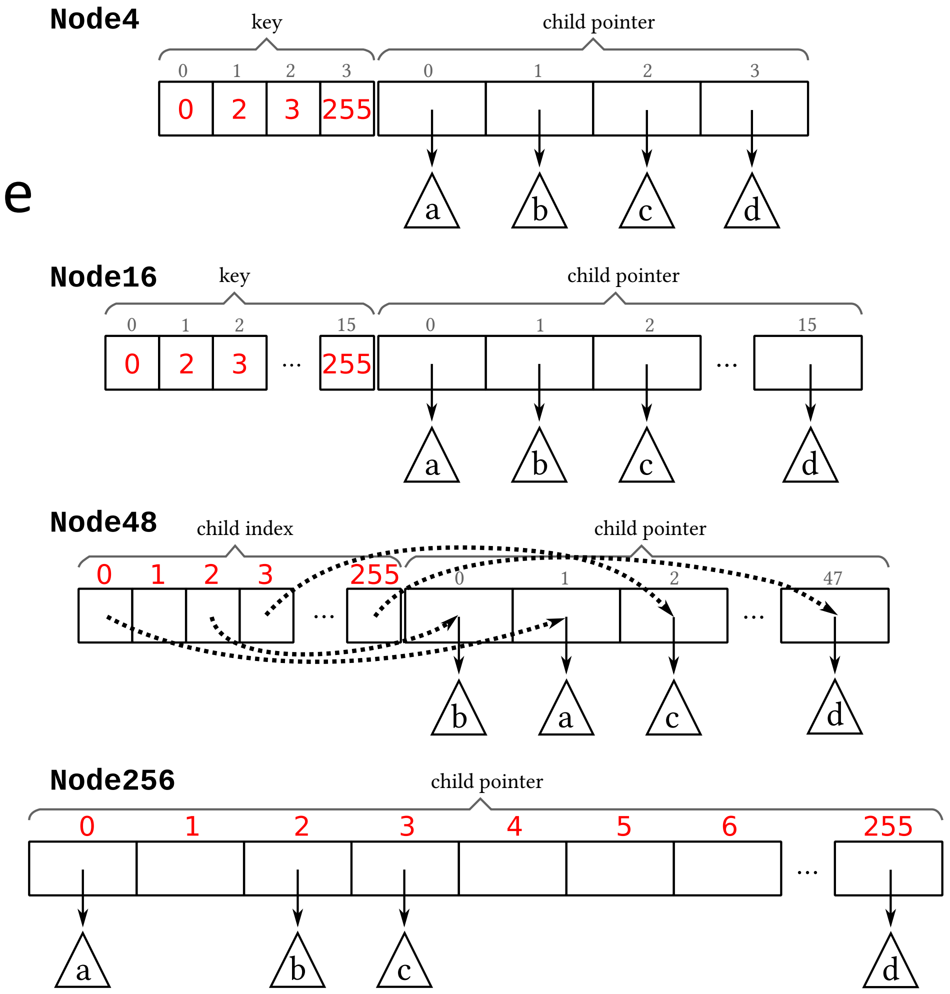
➢ Application

- Efficient for small sets or for sets of strings that share long prefixes
- URL matching (search engine)
- IP routing

ART

- **ART: Adaptive Radix Tree**
  - Integrate **adaptive node sizes** to the radix tree
  - Grow while adding new entries
  - Better space of use without reducing speed compared with the radix tree





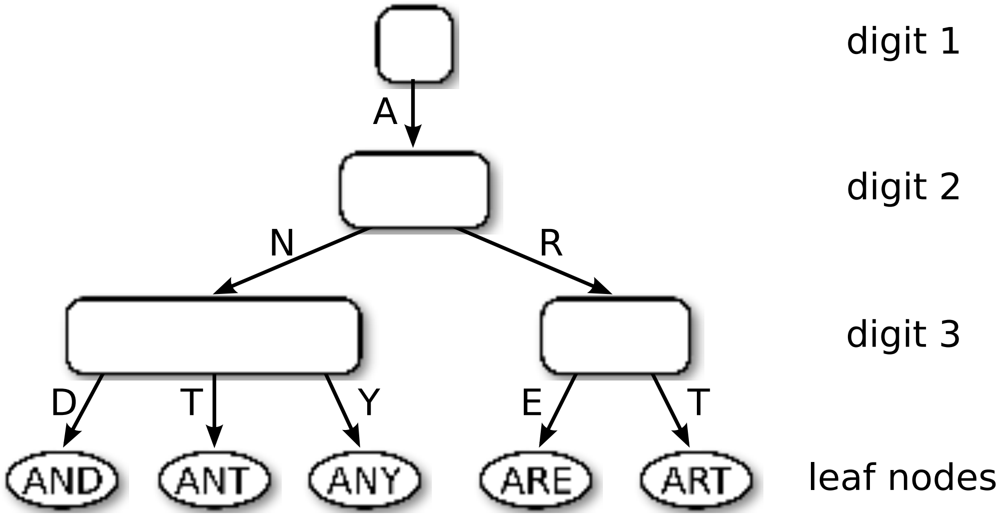
partial keys 0, 2, 3, and 255 are mapped to the subtrees a, b, c, and d

ART

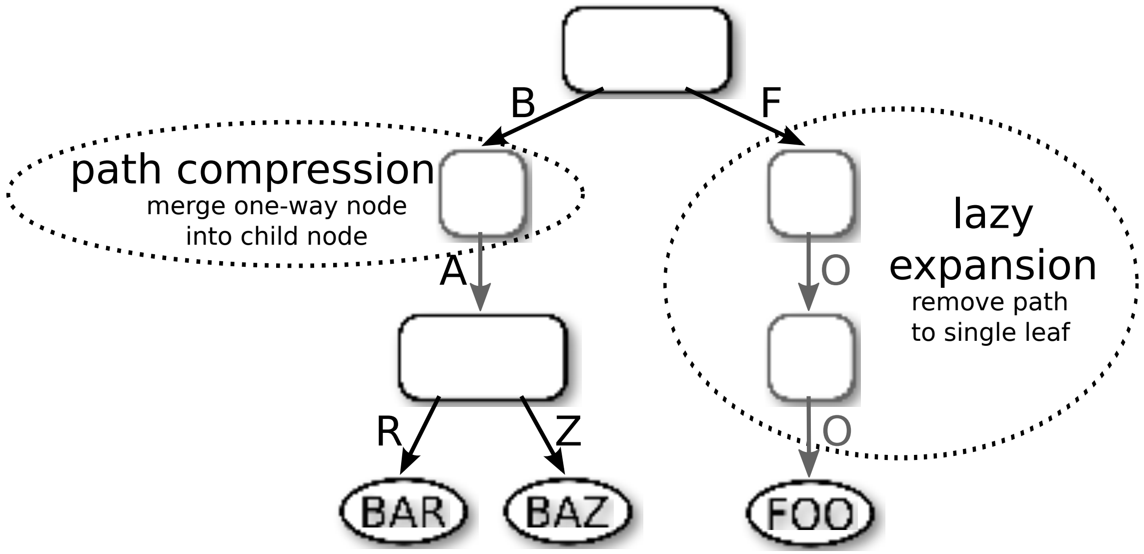
➢ Techniques in ART

- Path compression
- Lazy expansion

Reduce height & Space



Adaptively sized nodes in **ART**



Path Compression: B-A-R → BA-R

Lazy expansion: F-O-O → F-OO

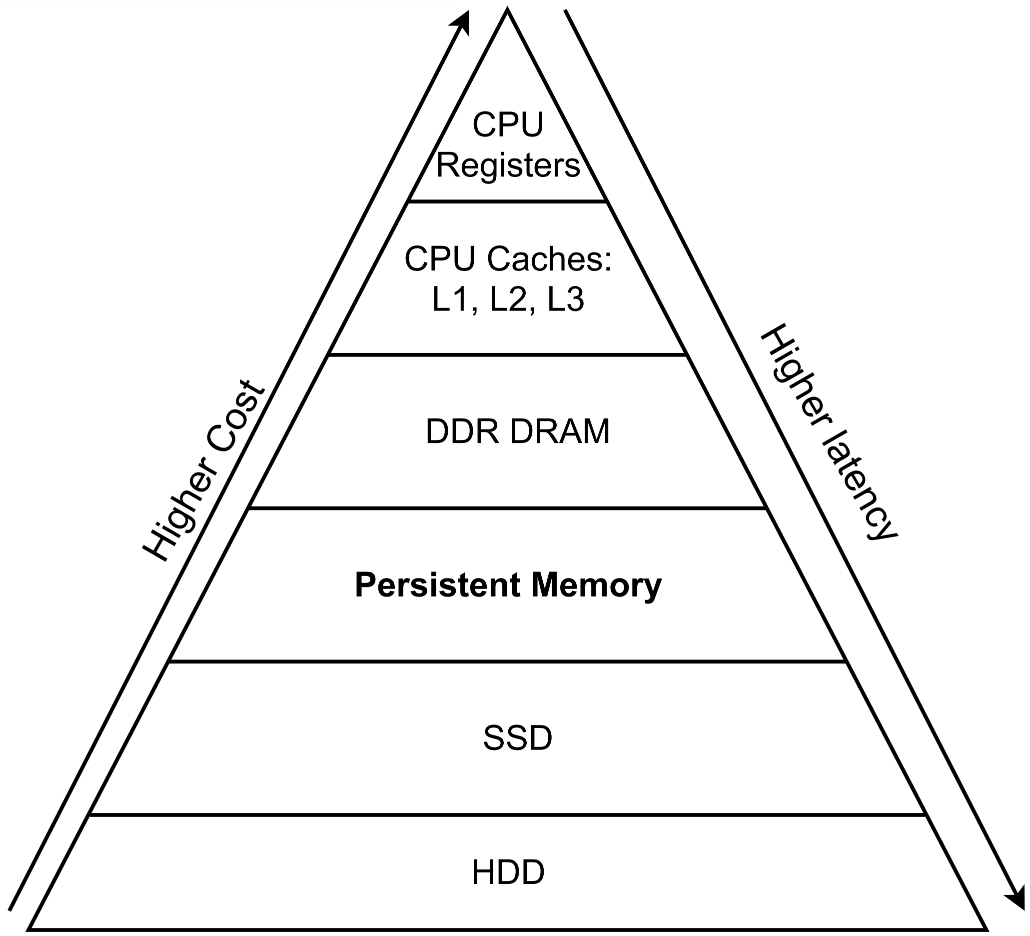
Persistent Memory

➢ **Features**

- Non-Volatile Memory
- Load/Store Instructions

➢ **Advantage**

- Lower access latency than SSDs
- Cheaper than DRAM
- Byte-addressable && Memory-like access

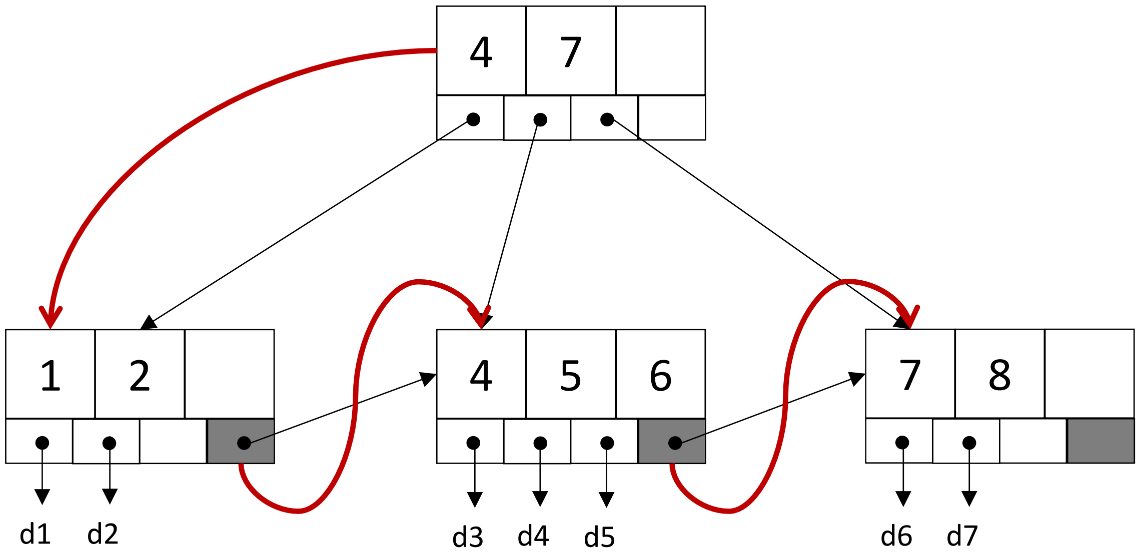


➢ Commercial persistent memory was available in 2019.04

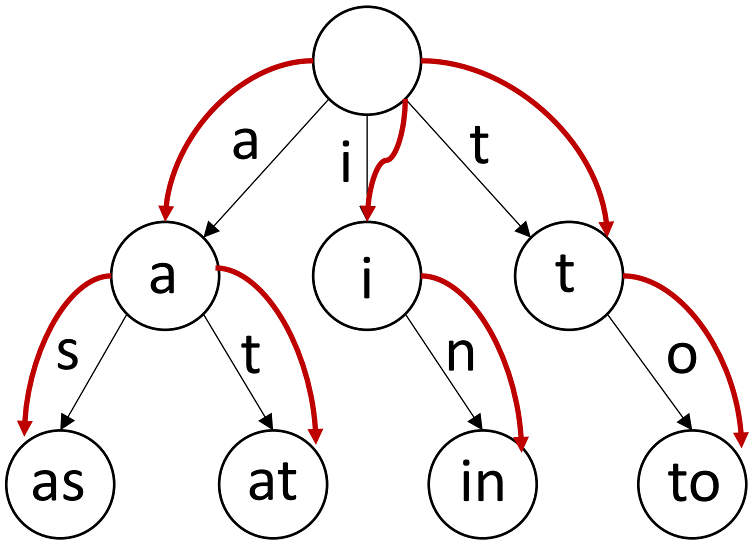
Motivation I: Functionality

➢ Range Queries Support

- Requirement of most KV stores (Redis, Memcached) and databases (MySQL, PostgreSQL)



Pointer chasing in B+-Tree



- ➢ Keys can be in leaf node or non-leaf node → Pointer chasing
- ➢ String Comparison

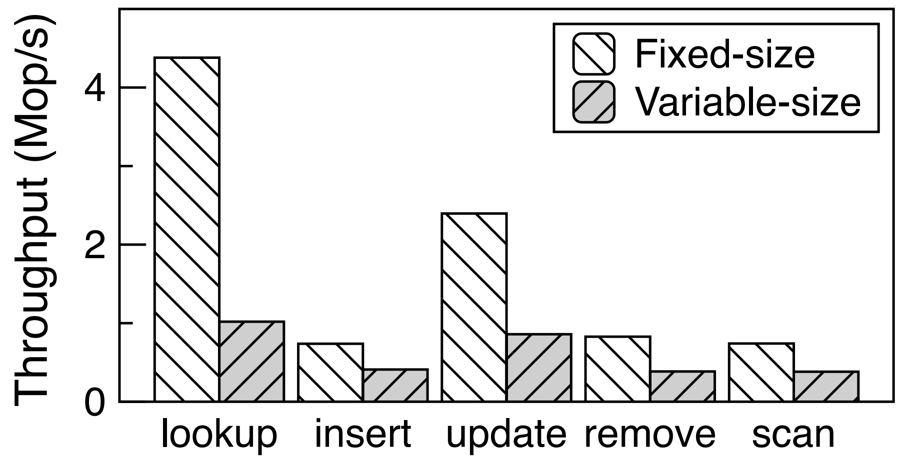
Motivation I: Functionality

➢ Variable-sized Keys Performance

- **1.8-3.9X** Performance degradation in B+-Tree

➢ Existing Solutions

- Fixed-sized B+-Tree → **Optimize 8-byte keys only**
- Variable-sized B+-Tree (store the addresses of keys in indexes) → **Incur more pointer chasing**



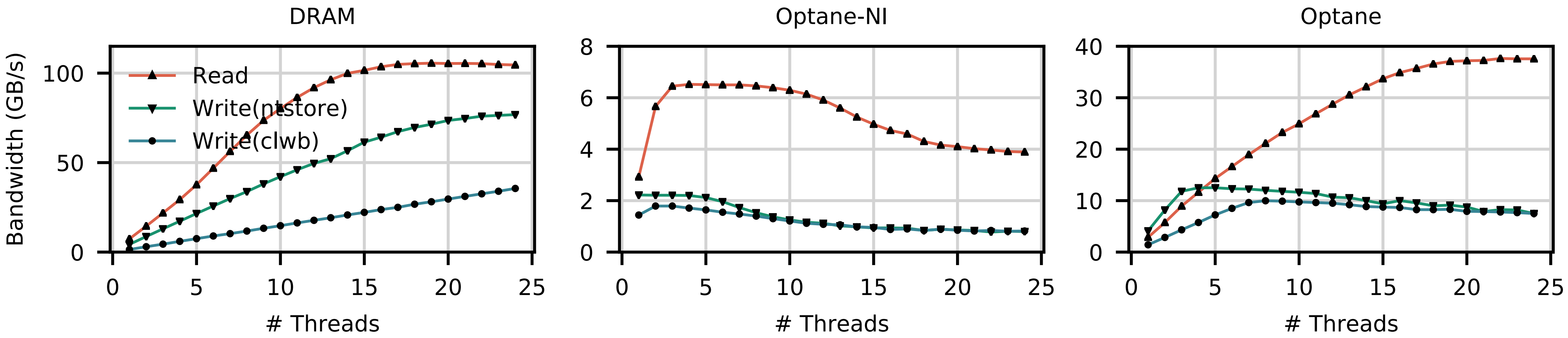
FAST&FAIR (B+-Tree), 8-byte keys, 4-threads

Performance Degration:
pointer chasing && string comparison during traversal

Motivation II: Performance

➢ Poor write scalability

- Reduce persistence overhead → crucial for performance

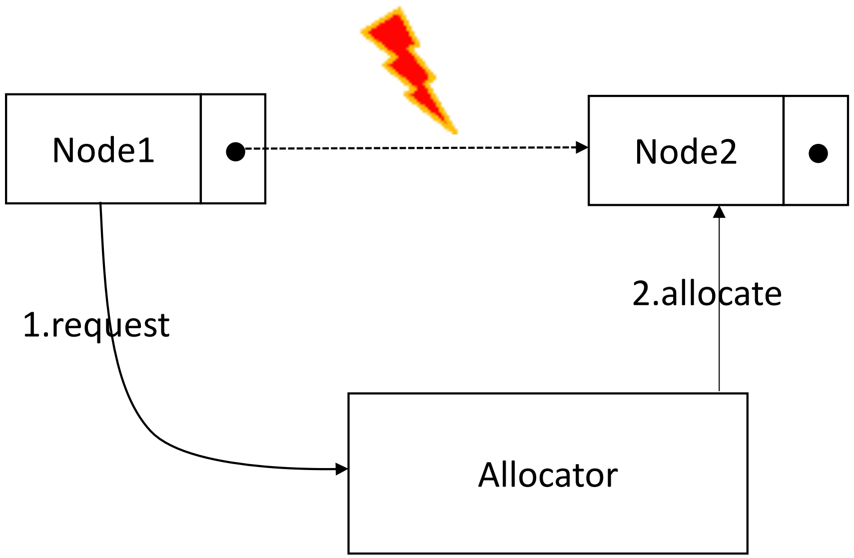


- ➢ Optane-NI: single persistent memory
- ➢ Optane: 6 interleaved persistent memory
- ➢ All threads use a 256 B access size

Persistent Memory has the **poor write scalability**
→ Avoid flush and memory fence as much as possible

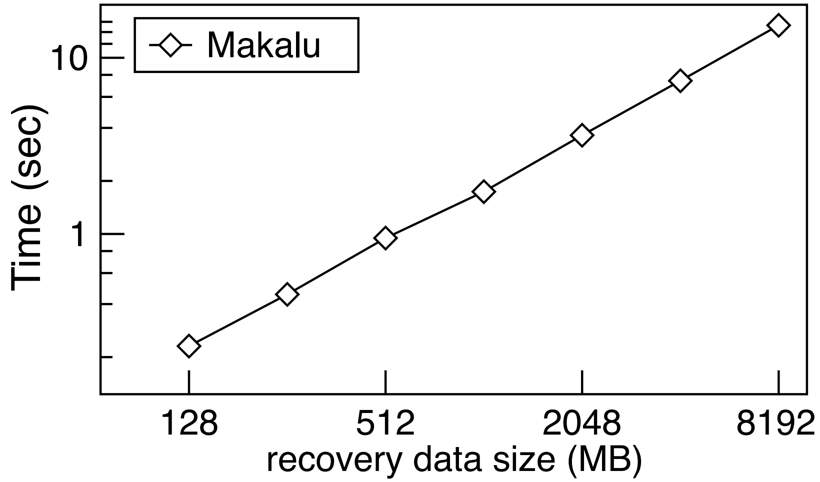
Motivation III: Correctness

- **Memory Safety**
  - **Volatile Allocator** → Memory Leaks
  - **Persistent Allocator**
    - Logging-based
    - Post-crash GC



Volatile Allocator

- Logging-based
  - Constant time recovery
  - Slow allocation/deallocation
- Post-crash GC
  - Fast allocation/deallocation
  - Slow recovery



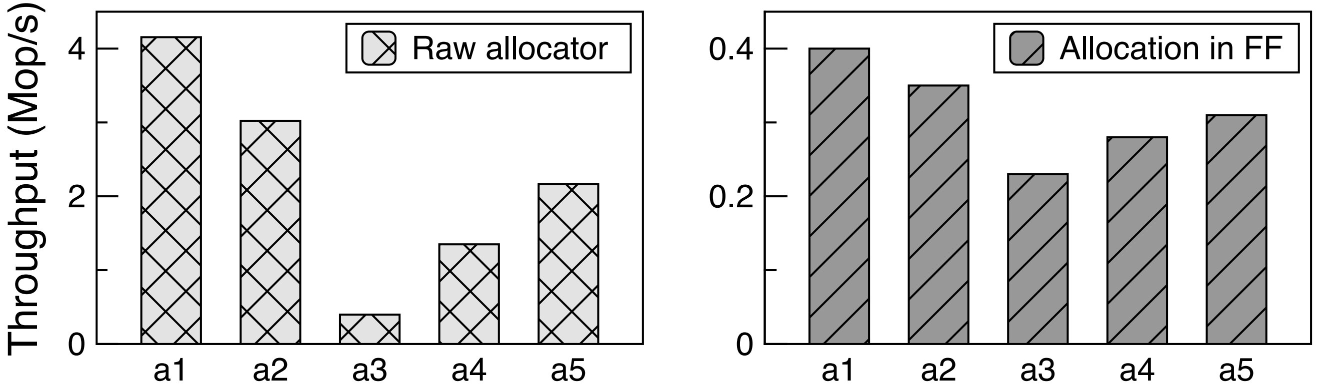
Makalu with post-crash GC

Motivation III: Correctness

➢ Allocator Performance

- a1: ***malloc***, the standard volatile allocator for DRAM
- a2: ***libvmmalloc***, volatile allocator based on `jemalloc`
- a3: ***PMDK***, logging-based persistent allocator
- a4: ***nvm_malloc***, logging-based persistent allocator
- a5: ***Makalu***, persistent allocator with post-crash GC

Allocate 64-byte chunk, then write and persist them in a single thread



(a) Performance of raw allocators (b) Allocators in FAST&FAIR

(a) ***Makalu*** is **50%** and **28%** slower than ***malloc*** and ***libvmmalloc***

(b) Gaps between different cases are narrowed → **Tree's traversal overhead**

Motivation

- **Functionality**
  - Range queries support
  - Reduce pointer chasing

  → Leaf compaction based on **ART**

- **Performance**
  - Reduce PM writes/flush

  → Entry compression && Minimal persistence

- **Correctness**
  - Fast allocation
  - Prevent memory leaks
  - Fast recovery

  → **DCMM** with instant restart

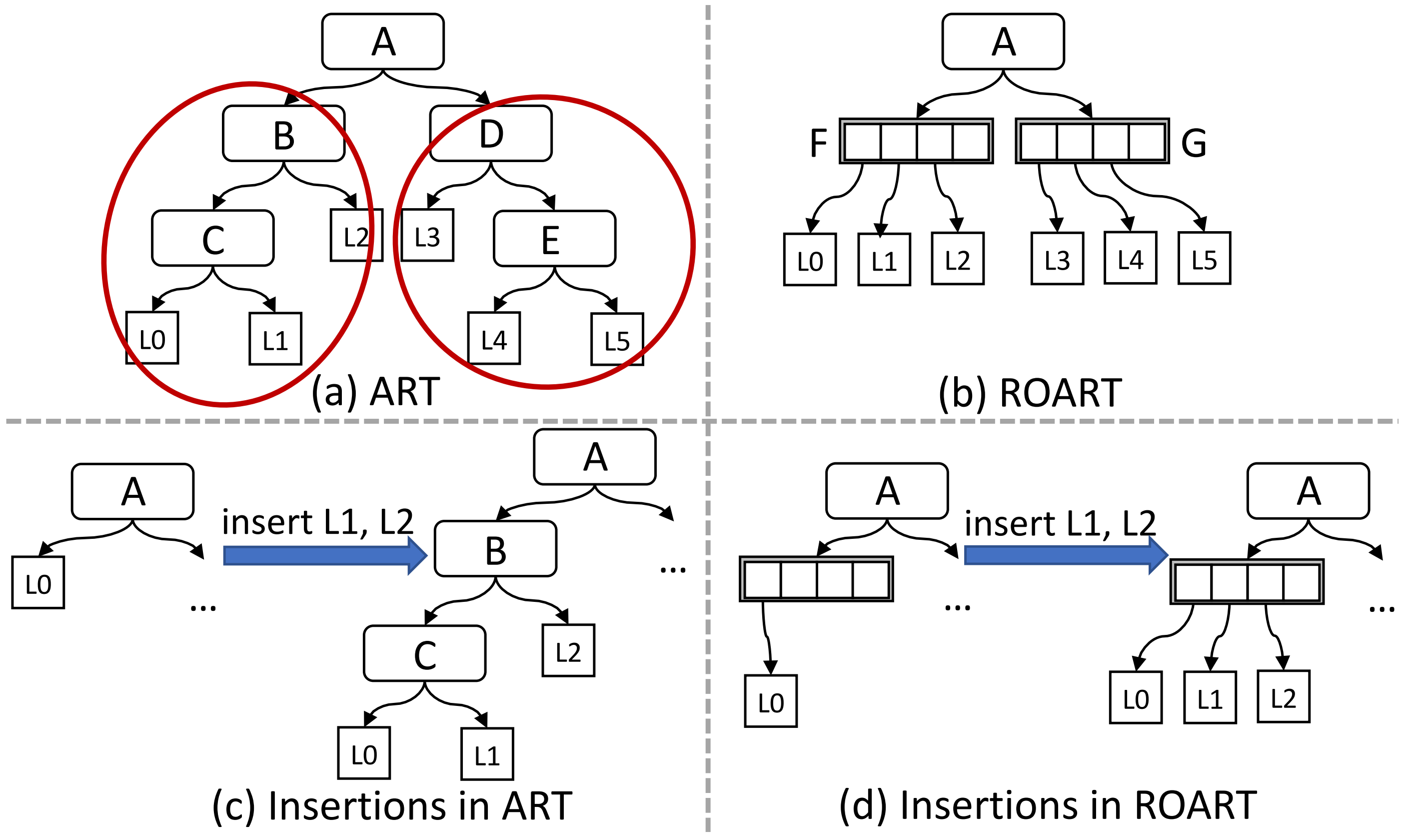
ROART

- ***ROART***, a **R**ange **O**ptimized based on **A**daptative **R**adix **T**ree
  - Reduce pointer chasing
  - Reduce persistent overhead
  - Tend to lower the height of tree
- Techniques
  - Leaf compaction
  - Entry Compression
  - Selective Metadata Persistence
  - Minimally Ordered Split
  - Delayed Check Memory Management(**DCMM**)

Leaf Compaction

➢ Leaf compaction

- Compact the pointers of leaf nodes into a leaf array



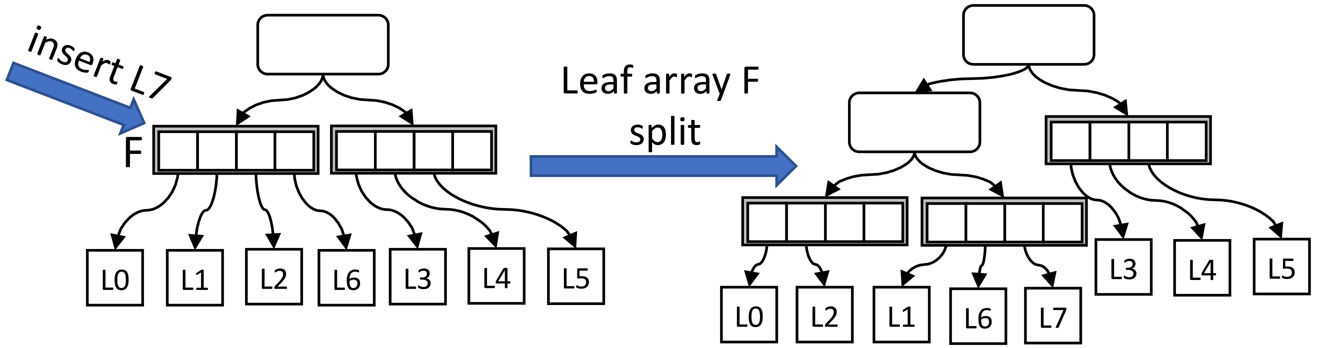
If a subtree of the radix tree has $\leq m$ leaf nodes, the subtree is compacted into a leaf array ($m = 4$ in this case)

➢ **ROART** embeds **fingerprints** (hash) of key in leaf array to avoid comparison for most unnecessary cases

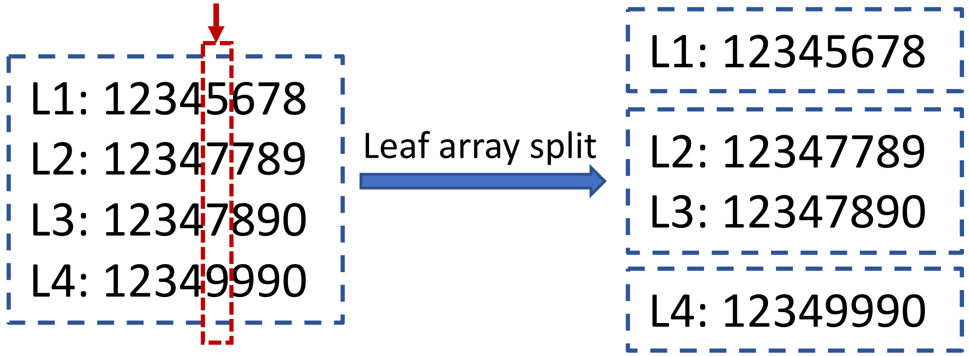
✓ Pointer chasing ↓
✓ Tree height ↓

Leaf Split

- **Leaf Split according to first non-common bytes**
  - Split after the first non-common bytes
  - Split cost is high but rare



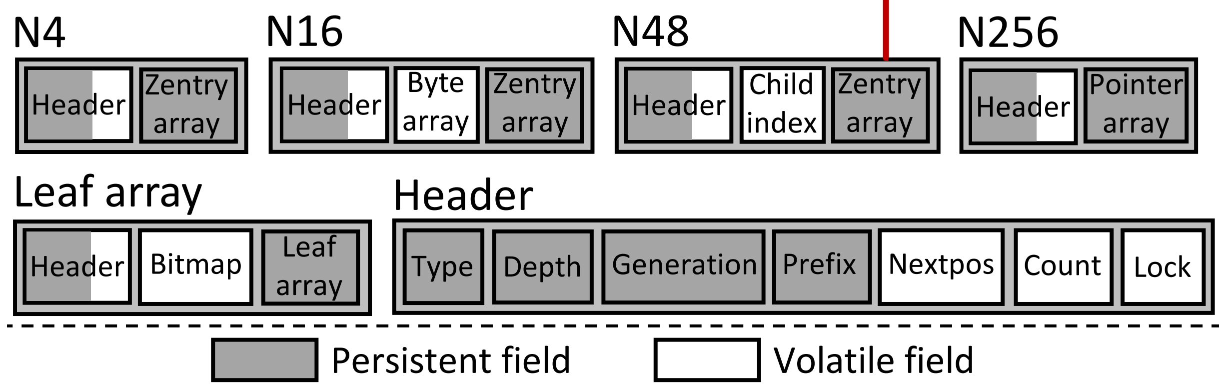
Insert L7; L1, L6 and L7 share the same prefix



Example of leaf array split

Optimize Persistent Overhead

- **Entry Compression**
  - Compact key byte into the pointer in N4, N16, N48
- **Selective Metadata Persistence**
  - Do not persist data which can be computed by scanning Zentry or pointer array
  - Generation is used for lazy recovery (Global generation number vs Node generation number)
    - Global generation number += 1 when restarted
    - Restore metadata on demand



Node structures in **ROART**

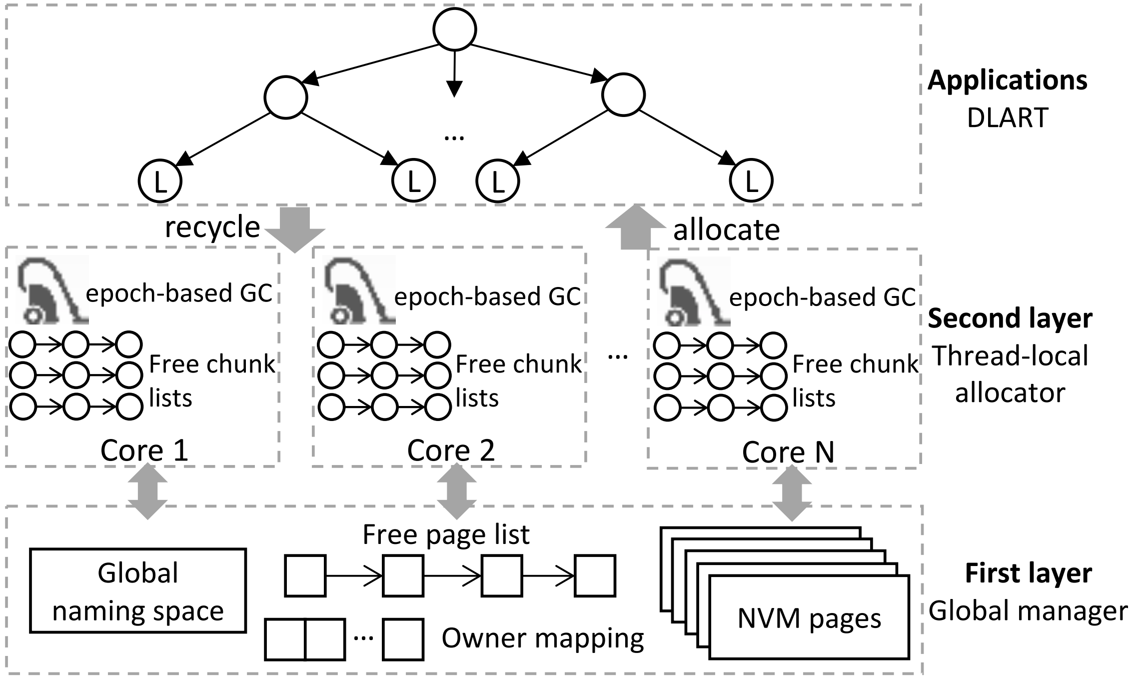
Per element in Zentry array:

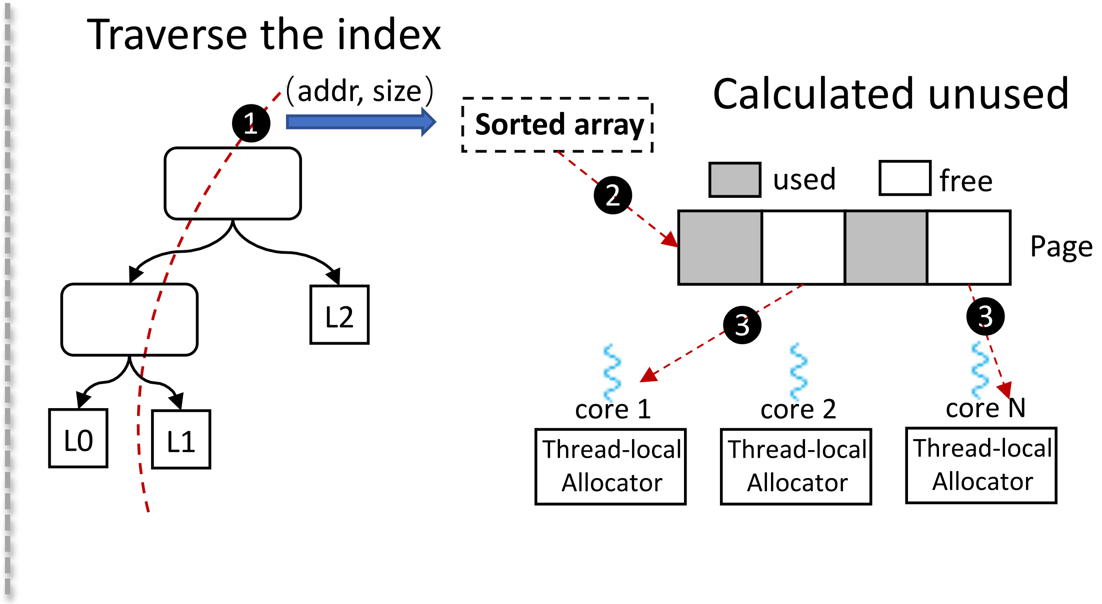
empty: 8-bit	key: 8-bit	pointer: 48-bit

Additional optimization: Minimally Ordered Split
→ Concurrent split operations & Reduce `sfence`

Allocator: DCMM

- **Delayed Check Memory Management (DCMM) based on post-crash GC**
  - Global naming space: Contain the roots of indexes and an ***`offset`*** field indicating the offset of last allocated page
  - Obtain offset && increase ***`offset`*** (persistence required) to request a new page

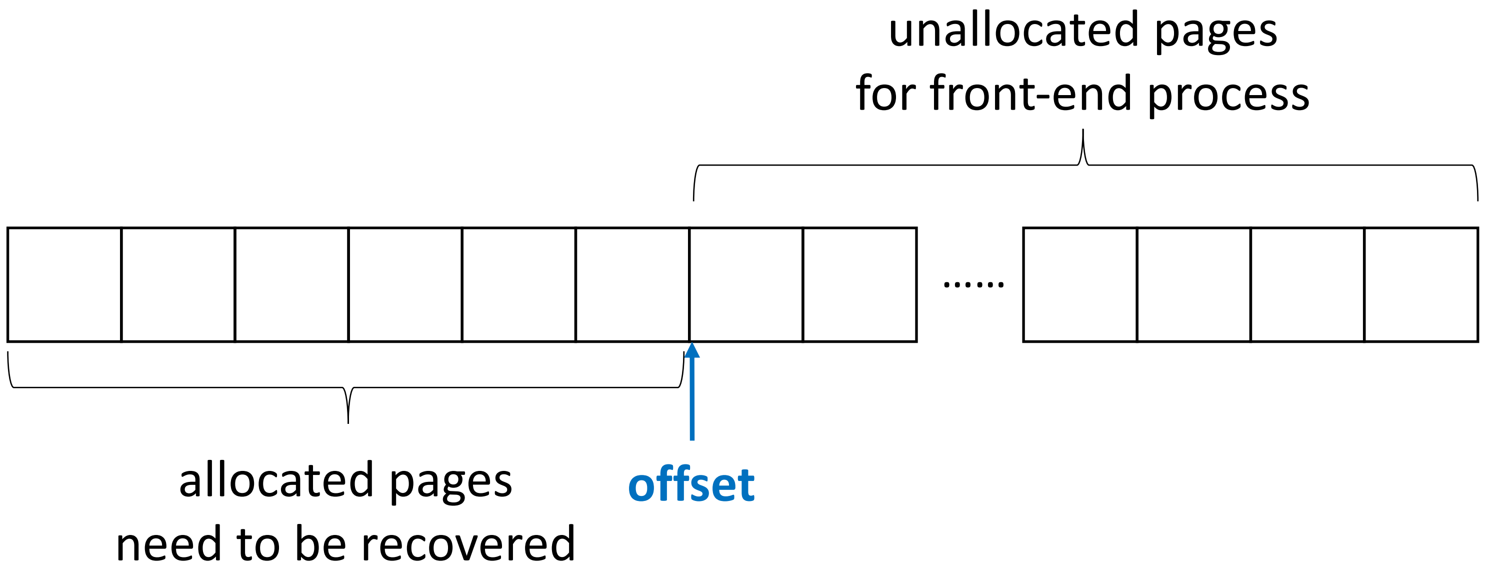




`ower_mapping`: a map between each page and its owner thread, $[0, \text{offset})$

Instant Restart

- Allocate immediately new pages after offset **without waiting** for other metadata recovery to complete
- Additional: Lazy recovery
  - Global generation number (+=1 when restarted) ↔ Node generation number



Start from offset when recovery

Experimental Setup

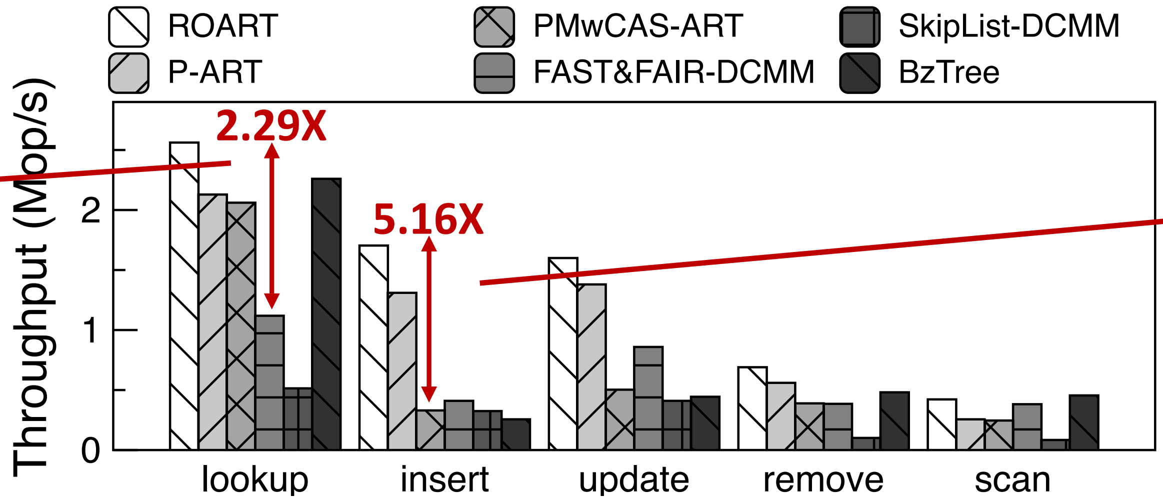
- Evaluation Setup
  - Dell PowerEdge R740 server
  - Four Intel(R) Xeon(R) Gold 5220 processors
  - 6×128GB Optane DC PMM per socket
  - 32KB L1-cache, 1MB L2-cache, and 25MB L3-cache
  - Persistent memory is managed by a DAX file system
- Comparison
  - **P-ART** [SOSP’19]
  - **PMwCAS-ART**(based on PMwCAS [ICDE’18])
  - **FAST&FAIR** [FAST’18]
  - **Lock-free SkipList** [ATC’18]
  - **BzTree** [VLDB’18]

- Modify **P-ART**, **FAST&FAIR** and **SkipList** with **DCMM** allocator for a fair comparison.
- **PMwCAS** and **BzTree** use their own persistent allocators based on **PMDK**

Overall Performance

- **Methodology**
  - 4 threads to generate randomly
  - **Keys**: randomly generated with sizes: 4-128 bytes
  - **Values**: fixed as 8 bytes

- FAST&FAIR: binary search ✗
- SkipList: cache locality ✗

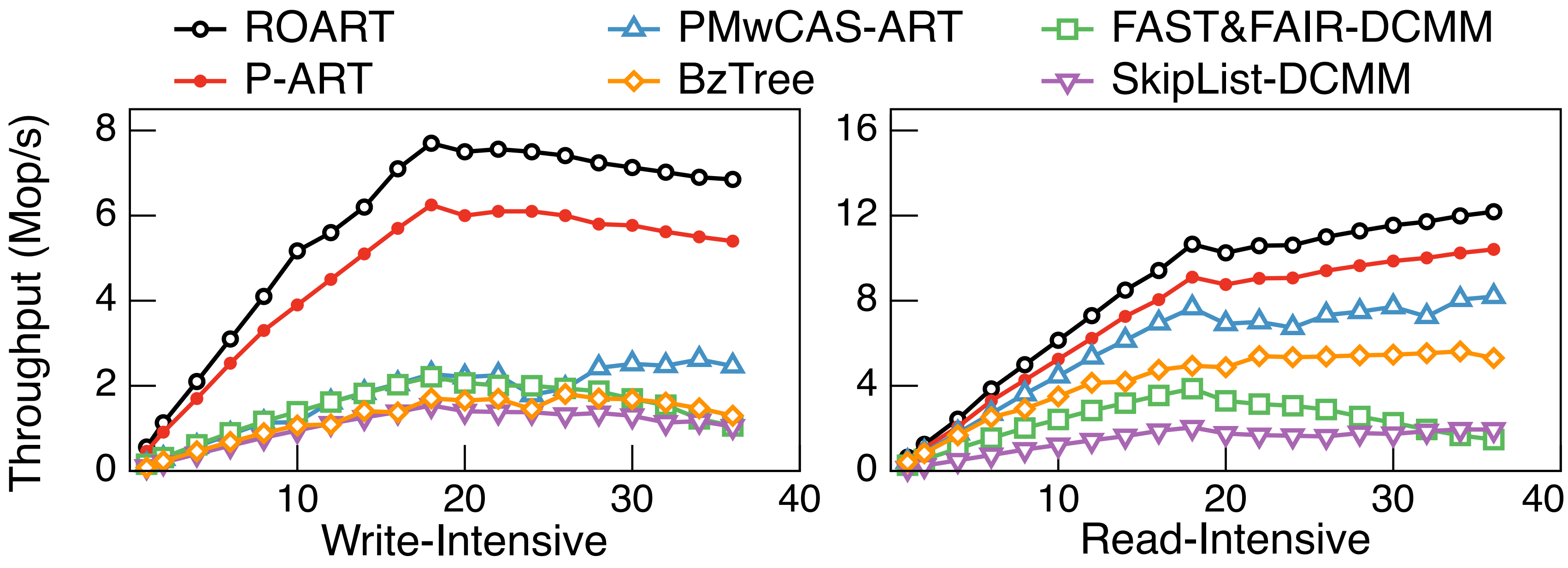


- Higher allocation performance compared with **PMwCAS (PMDK)**
- Less persistent related

- **ROART** achieves 20%-24% throughput compared with **P-ART** and **PMwCAS-ART** because of leaf compaction mainly → Lower the height of tree and benefit traversal
- **BzTree** has slotted-page node layout → Good cache locality for variable-sized KVs && Binary search

Overall Performance

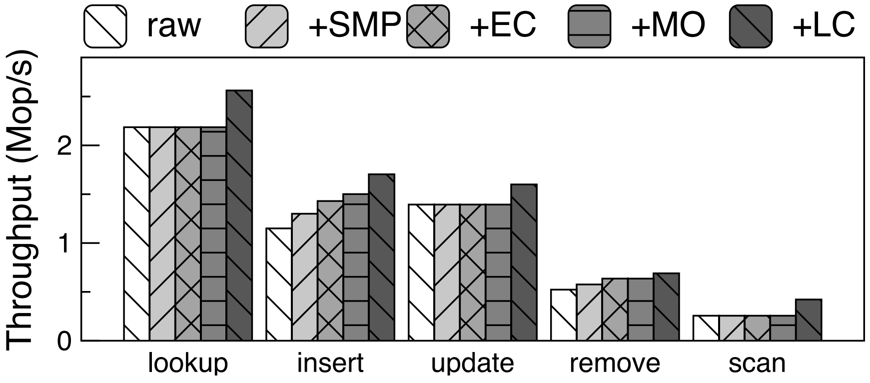
- **Methodology**
  - **YCSB** benchmark



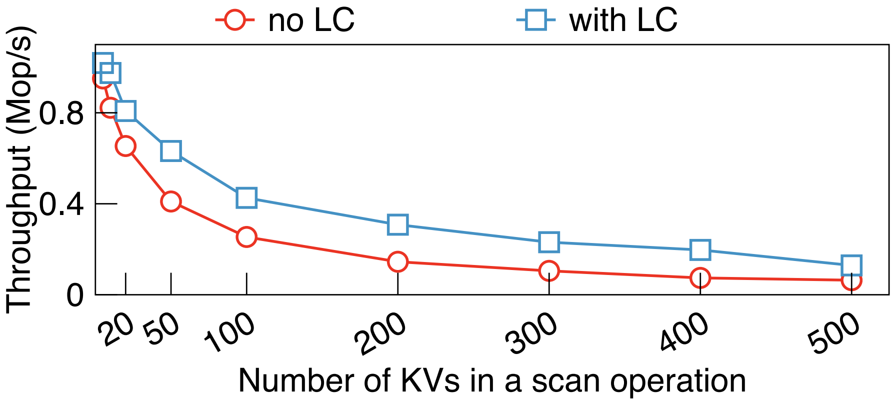
- **ROART** achieves **2.78-6.57X** using 36 threads → Less traversal and persistence

Microbenchmark

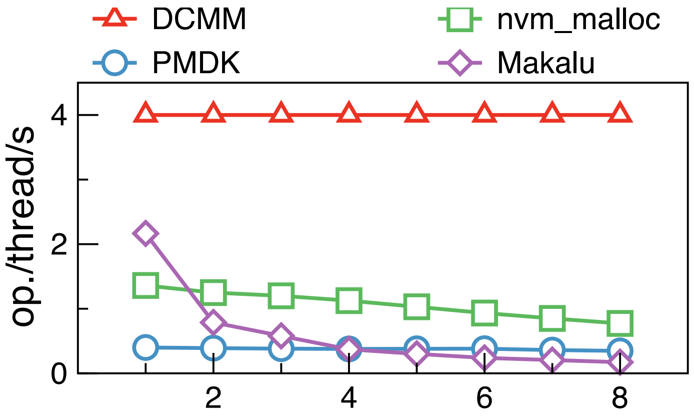
- **Each optimization**
  - **SMP**: Selective Metadata Persistence && **EC**: Entry Compression
  - **MO**: Minimally Ordered split && **LC**: Leaf Compaction



Performance improvement of each optimization



Range queries with different key numbers

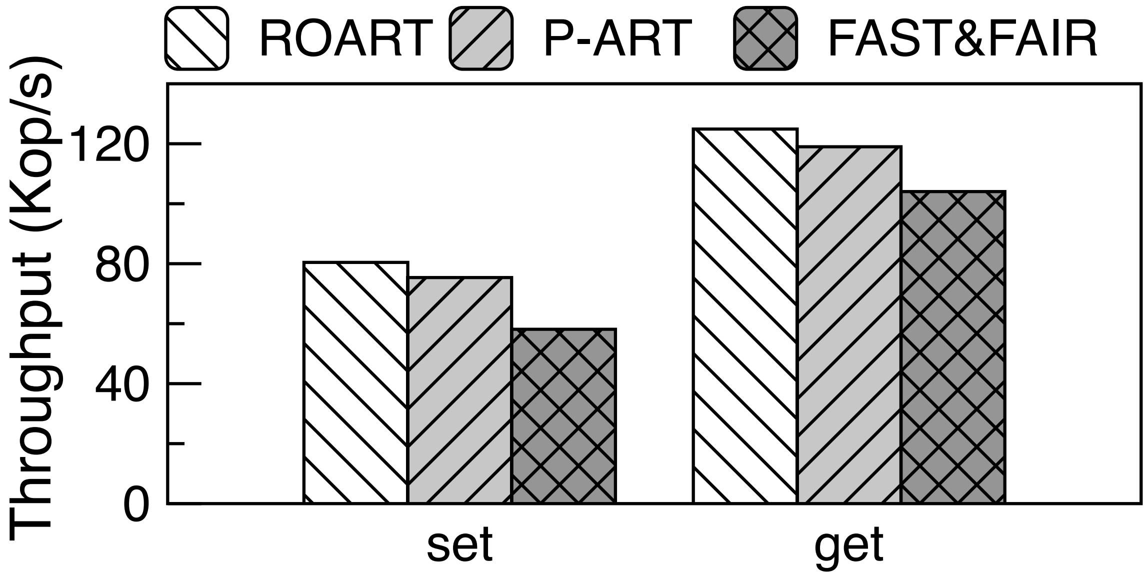


Performance with different allocators

- **ROART** with **LC** outperforms the version without **LC** by **1.07~2.01X**
- **DCMM:** larger pages in a lock-free manner, page size is only 4K in **Makalu** (lock-based)

Real-world Application Performance

- Test in **Memcached**
  - Replace its hash index to three persistent indexes: **ROART**, **P-ART**, **FAST&FAIR**
  - Single thread



- **ROART** outperforms **P-ART** and **FAST&FAIR** by up to **1.07X** and **1.38X** in set operations, **1.06X** and **1.19X** in get operations

Conclusion

- **ROART**, a **R**ange **O**ptimized based on **A**daptative **R**adix **T**ree
  - Leaf compaction
  - Entry Compression
  - Selective Metadata Persistence
  - Minimally Ordered Split
  - Delayed Check Memory Management(**DCMM**)