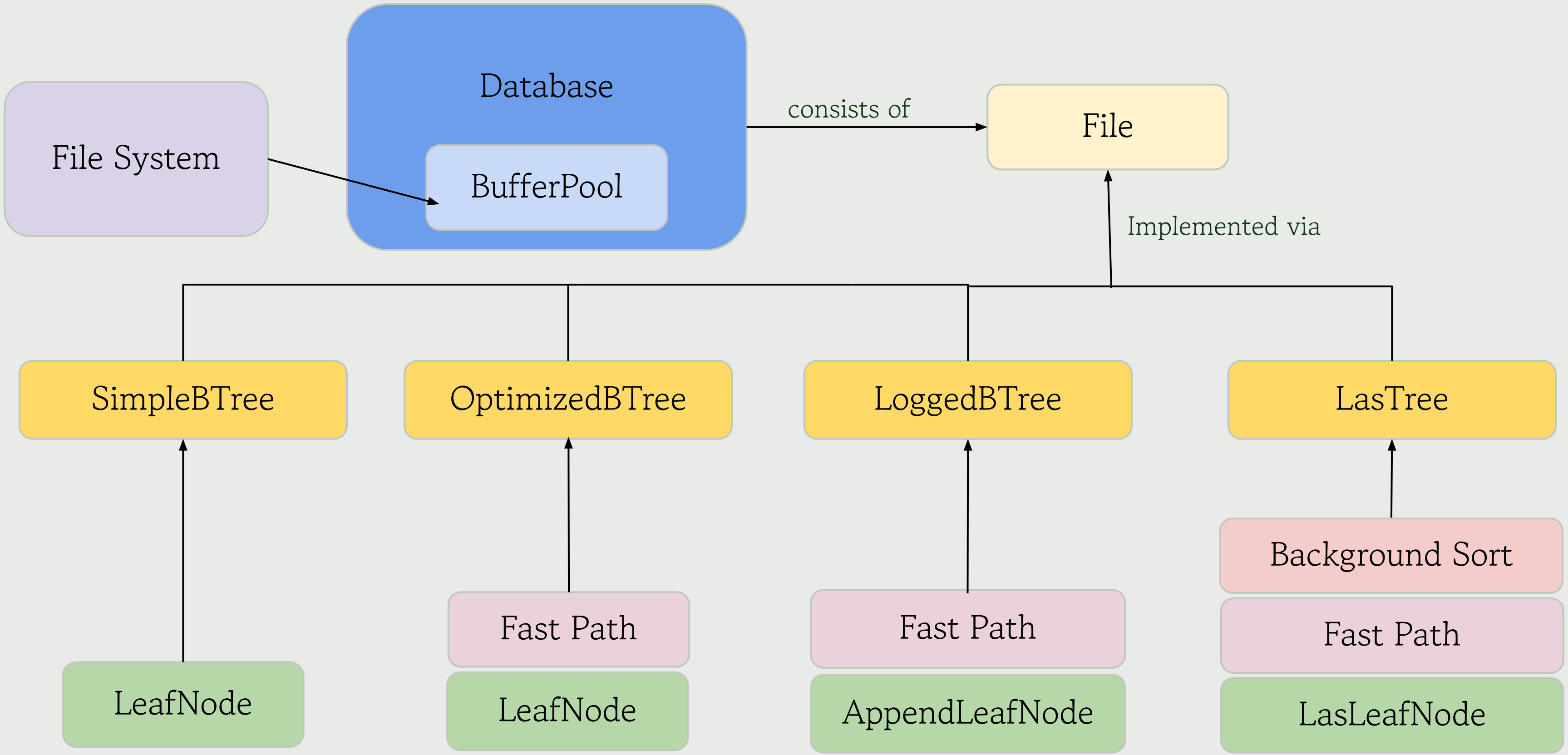
Cache-Awareness for Near-Sorted Indexing: The LaS Tree

Jingzhi Yan, Zhiyuan Chen, Jinpeng Huang

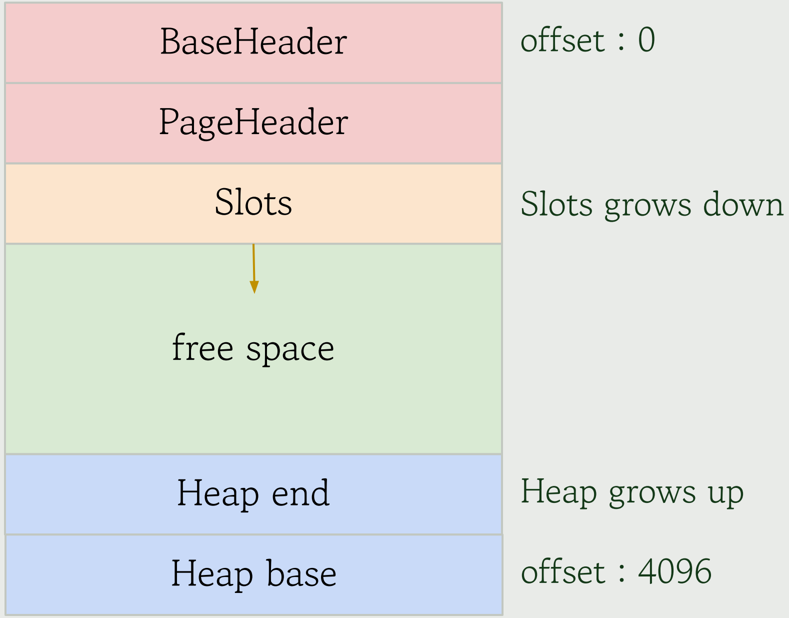
Background & Motivation

- Near-Sorted Data is Common
- Challenge: Fast Insertions
- Cache Awareness
- Still Need Efficient Point & Range Queries


File System
Database
BufferPool
consists of
File
Implemented via
SimpleBTree
OptimizedBTree
LoggedBTree
LasTree
LeafNode
Fast Path
LeafNode
Fast Path
AppendLeafNode
Background Sort
Fast Path
LasLeafNode

LeafNode Page Layout

- Fixed-size 4KB pages
- Slot array grows downward, heap grows upward



Storage Management:Database & BufferPool

- Database: collection of Tree Files + BufferPool
- BaseFile: Reads and writes fixed-size pages from/to disk
- BufferPool
  - caching pages in memory
  - All operations write via bufferpool

Fast Path

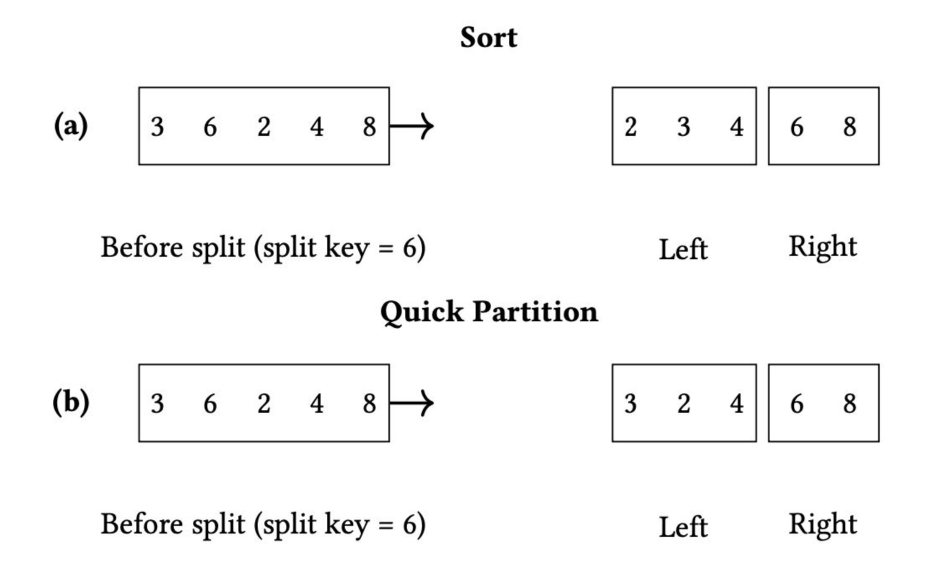
- Key idea : insert to fast path leaf directly without tree traversal
- Fast path hit: Directly insert; no need to traverse tree
- Fast path miss: Fall back to standard root-to-leaf search
- Fast path update:
    - Soft update: Slide fast path to the right neighbor after successful adjacent insert
    - Hard update: Reset fast path to current leaf after multiple failures (e.g., 3 misses)

AppendLeafNode

- Key idea : append to leaf node!
- Benefits:
  - No need to move memory around
  - O(1) insertion
  - Tombstone for deletion
- Sort on split
  - When node is full, compact and fully sort entries
  - Then split at 3/4 position to maintain balance

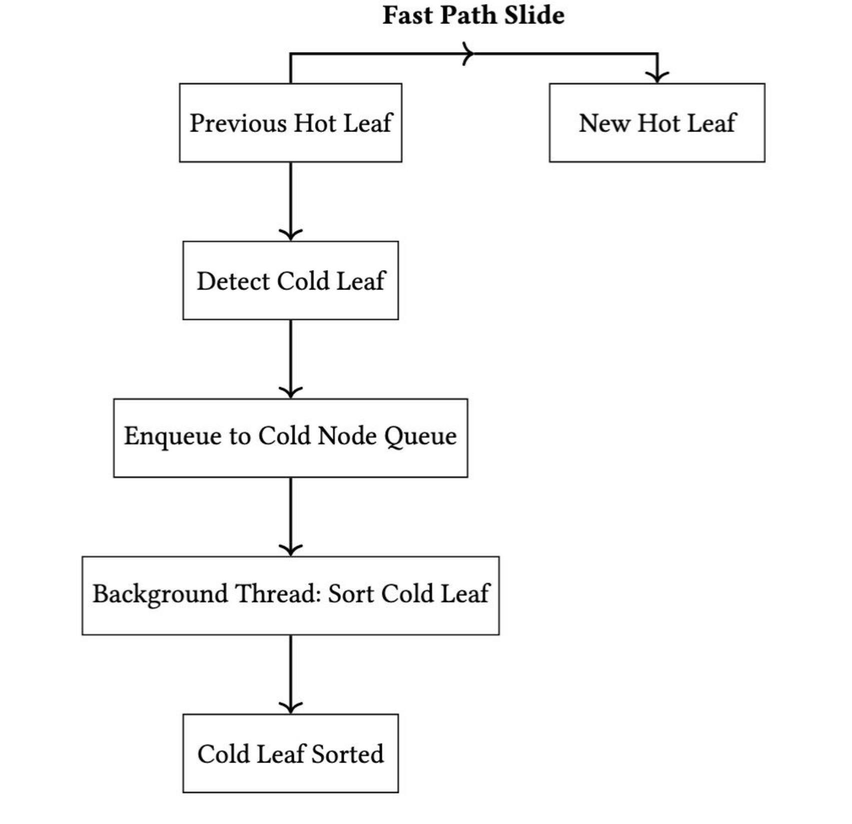
LasLeafNode

- Key idea : lazy sorting!
- Quick partition on Split
    - Compact
    - Partition two leafs by split key, leafs remains unsorted



LasTree = LasLeafNode + Background Sort

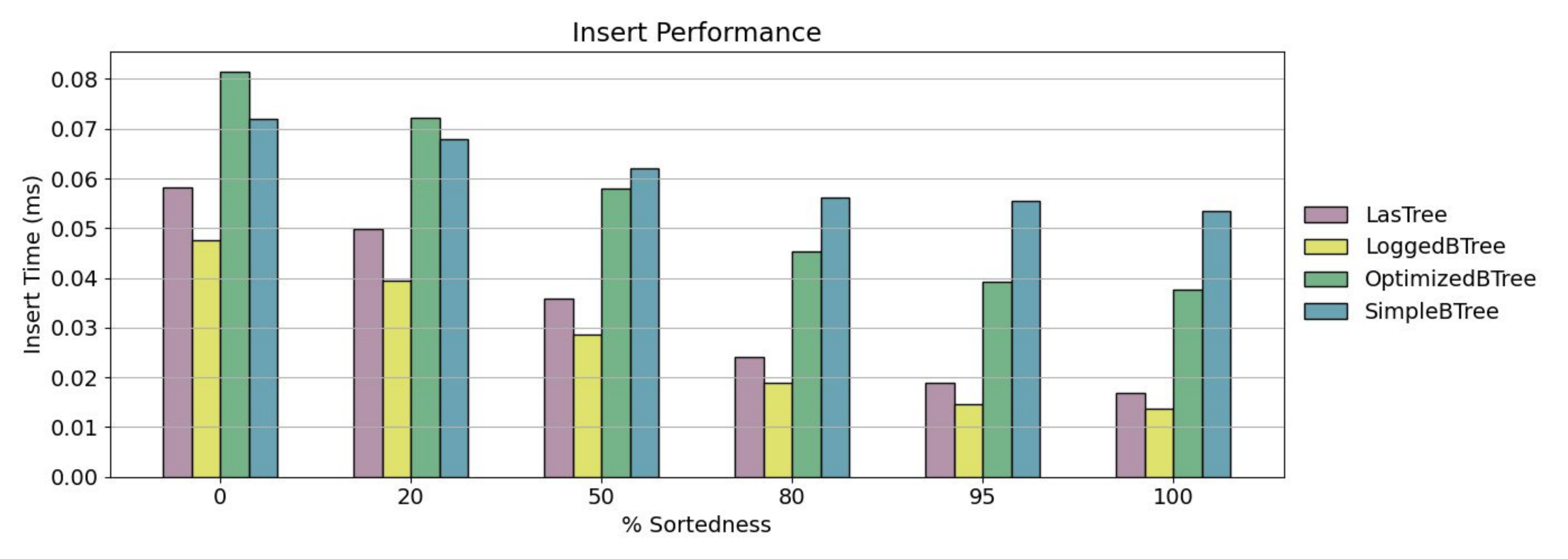
- Background Sorter:
  - Monitor "cold" leaves
  - Sort cold leaves asynchronously
- Benefit:
  - Insertions stay fast
  - Queries benefit from eventual sortedness



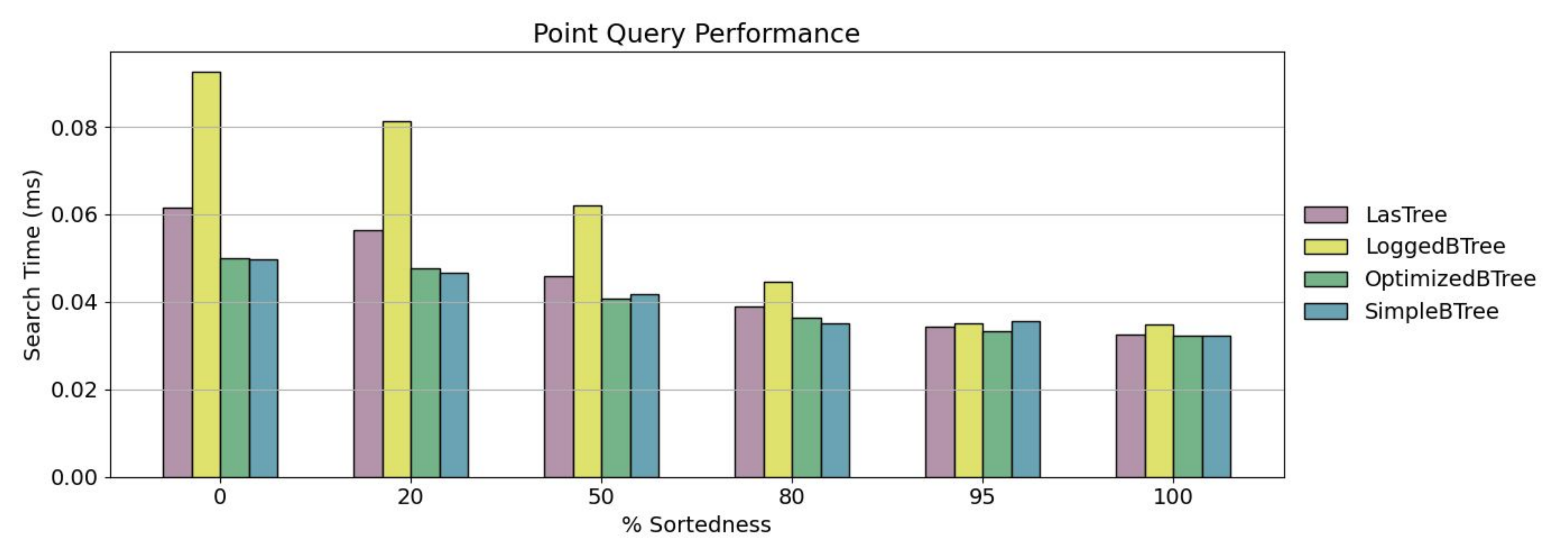
Benchmarking

- Environment: Apple M1 Pro(10-core CPU, 16GB RAM)
- Data Size: 100,000 tuples, average over 3 runs
- Sortedness Control: percentage of unordered entries (k)

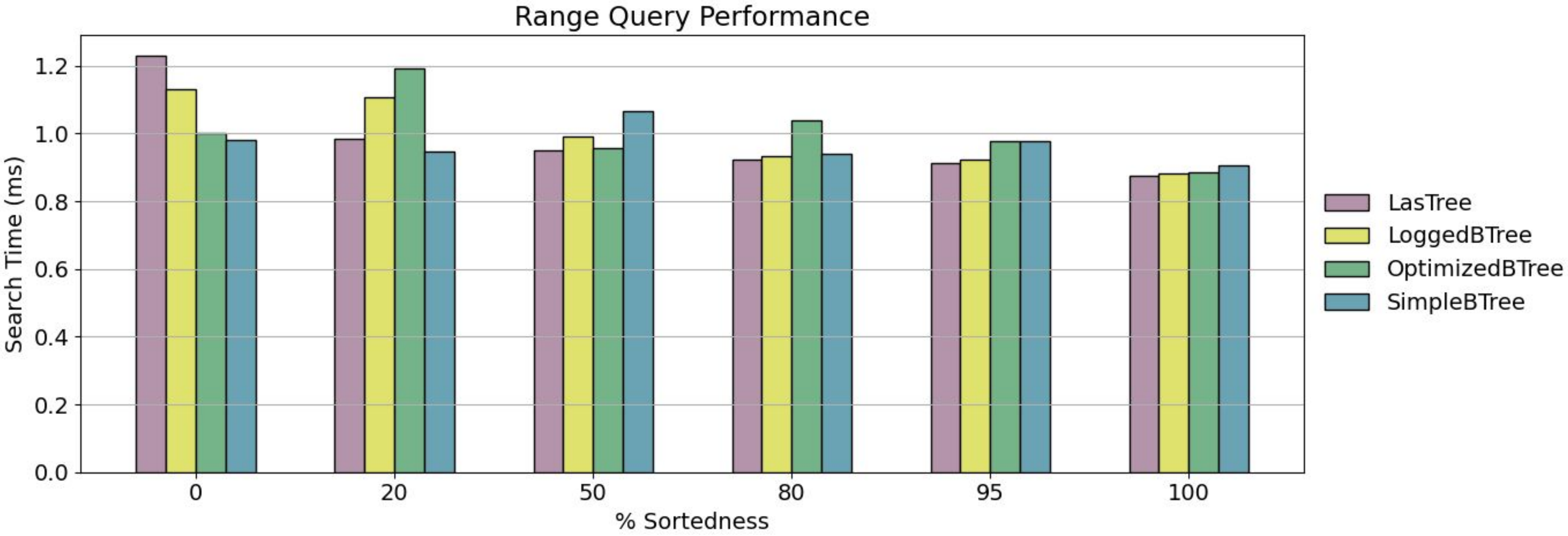
Insert Performance



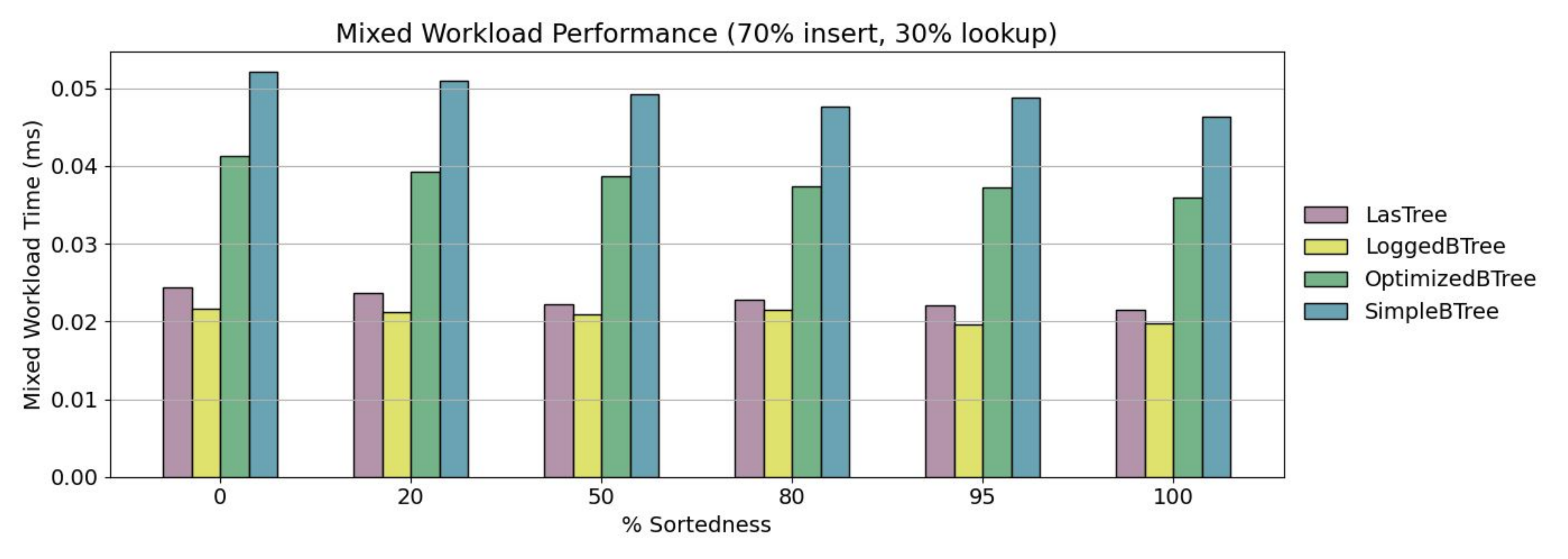
Point Query Performance



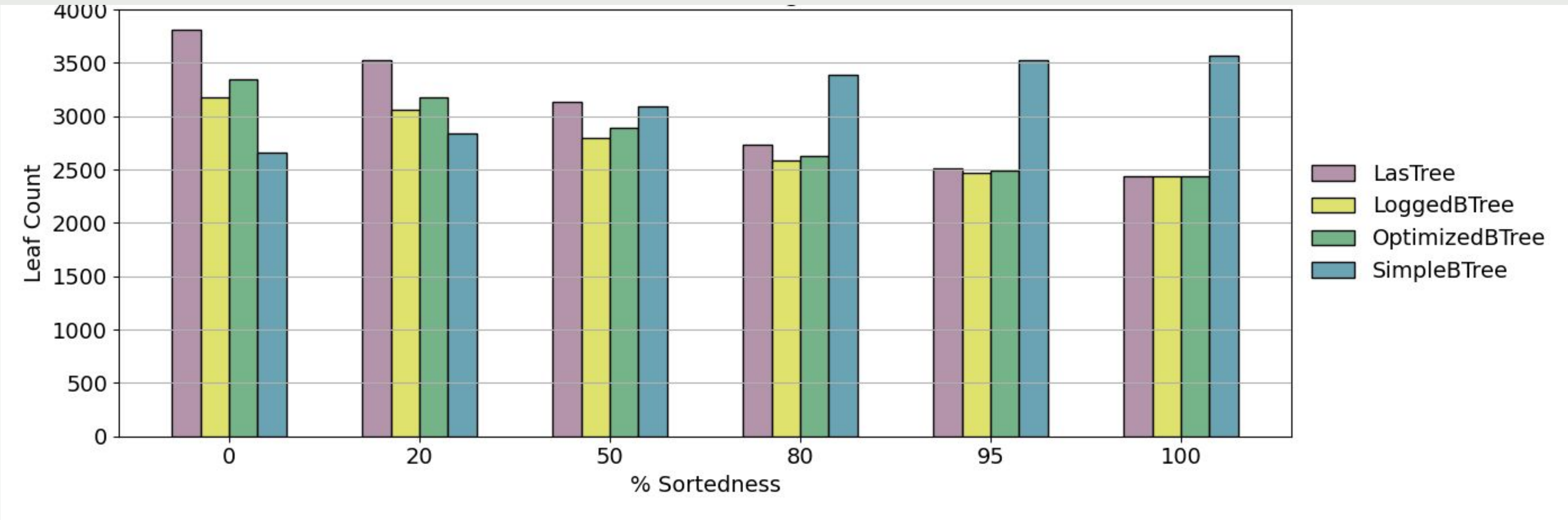
Range Query Performance



Mixed Workload



Leaf Count



4000
3500
3000
2500
2000
1500
1000
500
0
Leaf Count
0
20
50
80
95
100
% Sortedness
LasTree
LoggedBTree
OptimizedBTree
SimpleBTree

LasTree does not Perform as Expected!

OverHead!

1. **Compaction Cost**
   - Removing duplicates and tombstones is a must
   - What if simply erase records(add tombstones) and no compaction?

     -> proves more overhead
2. **Split key selection instability**
   - Imbalanced splits when data is not fully sorted
3. **Lock Contention**
   - Two threads competing for locks on same leaf, blocking insertion

Division of Labor

Design

———————————————

We decided the high-level design together.

Implementation

———————————————

Jingzhi - LeafNode, Debugging

Zhiyuan - Fast path

Jinpeng - Database + BufferPool

Experiments

———————————————

Jingzhi - benchmark metrics

Zhiyuan, Jinpeng - python file plotting data

Future Work

- Concurrency control
- Better cold leaf detection algorithm
- Larger scale evaluation

Thank You!