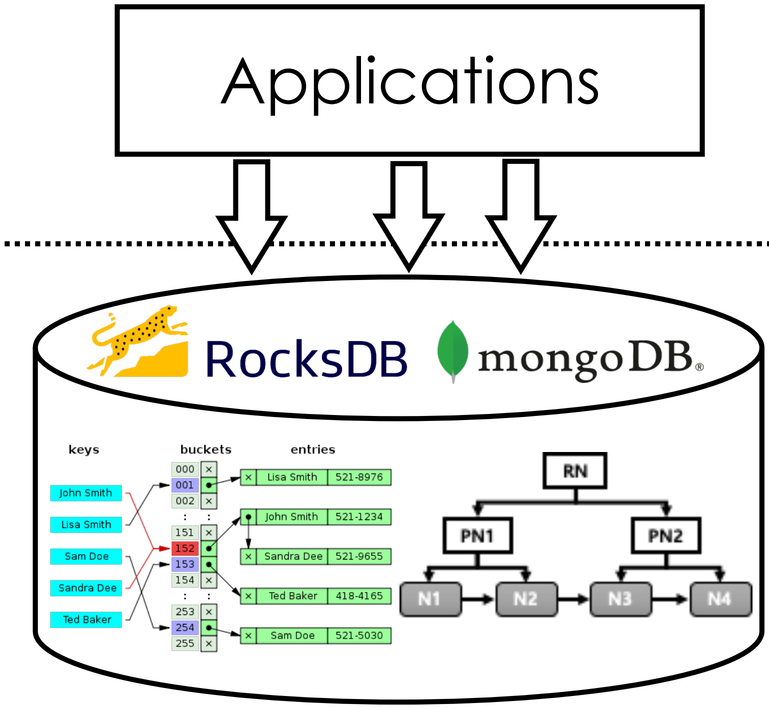
Towards Scalable Manycore-aware Persistent B+-Trees for Efficient Indexing in Cloud Environments

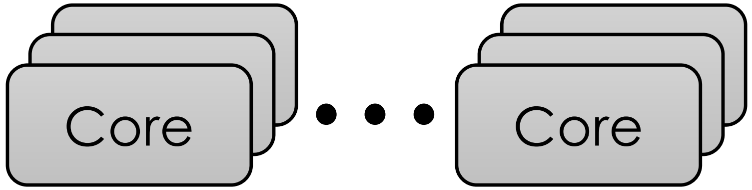
Safdar Jamil, Awais Khan, Bernd Burgstaller, Youngjae Kim

Agenda

- Introduction
- Background
- Motivation & Challenges
- F^3-Tree Design
- Evaluation
- Summary

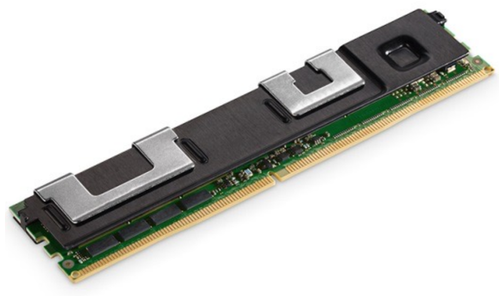
Intel Optane DC Persistent Memory on Manycore machines





100s of CPU cores

DCPM



- Byte-addressable
- Persistent
- Low latency

Intel Optane DC Persistent Memory based manycore machines

- Application at Manycore machines
  - Performance increases
    - With more resources ➔ increased compute and memory resources
  - To achieve performance scalability
    - Efficient data structures are required.

- Target Applications
  - NoSQL database systems
    - Indexing data structures ➔ B+-Trees

B+-Tree for PM

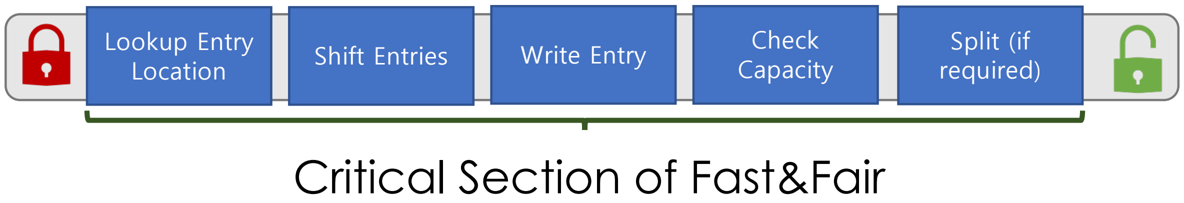
- DB storage engine
- B+-Tree is widely adopted indexing data structure for PM

Existing B+-Tree Studies on Persistent Memory	
Fast&Fair	FAST (2018)
uTree	VLDB Endowment (2020)

Evaluated the performance scalability of Fast&Fair (state-of-the-art B+-tree)

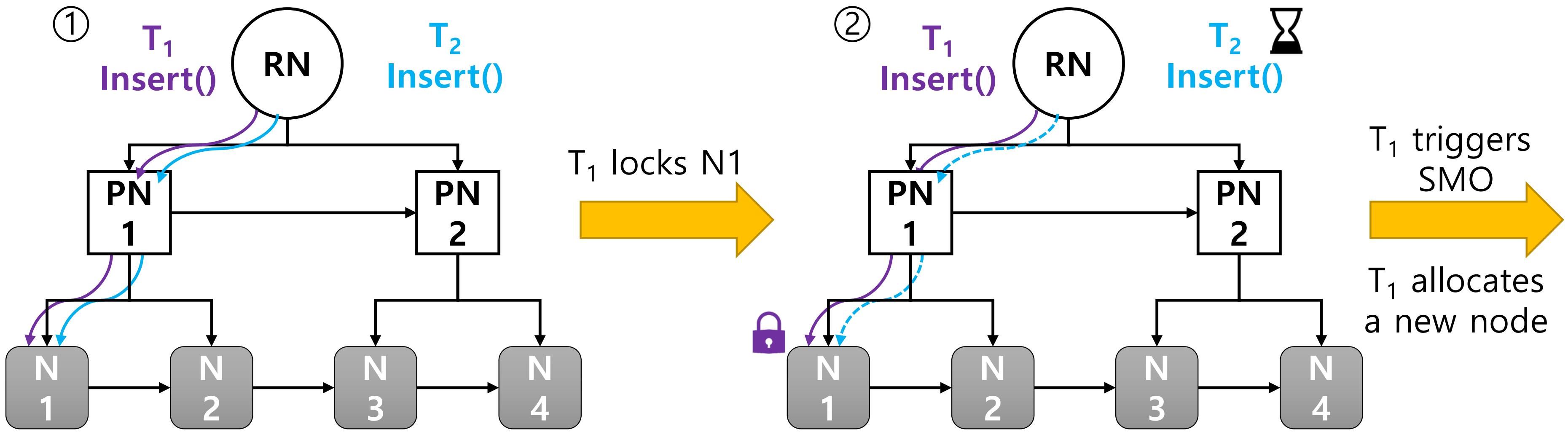
Scalability Analysis of Fast&Fair for Manycore Machines

- Huge critical section

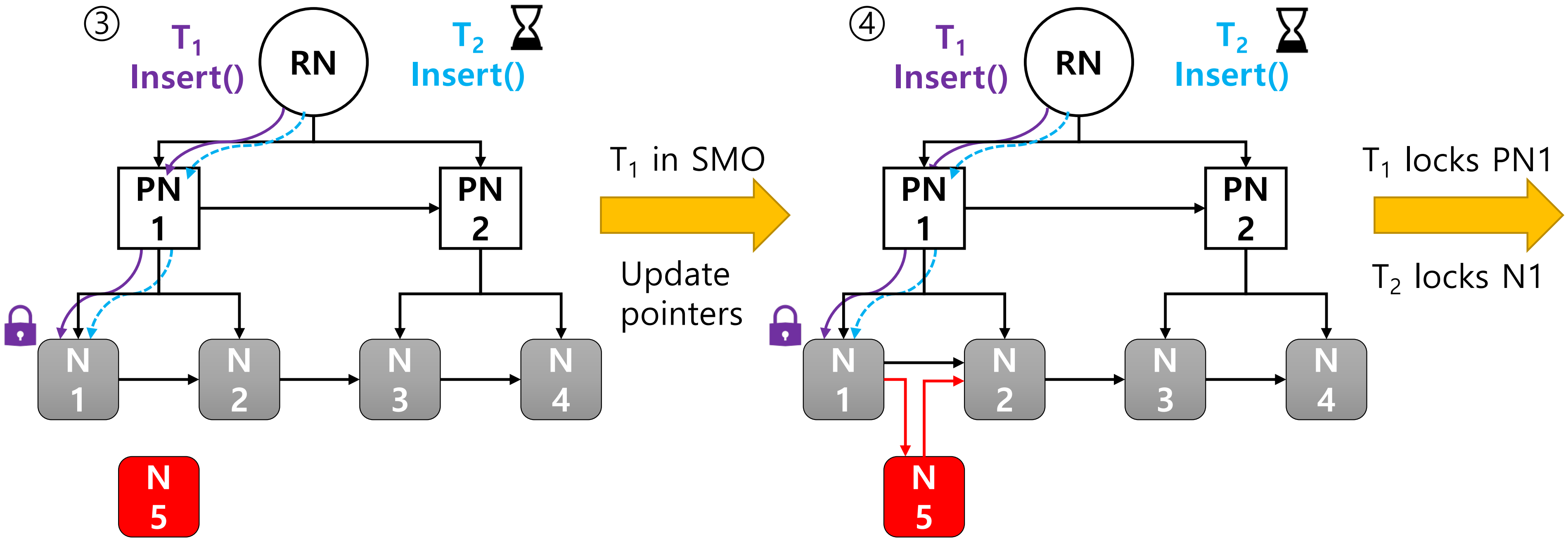


- Per-node MUTEX lock limits the scalability
- SMOs, split & merge, increase contention

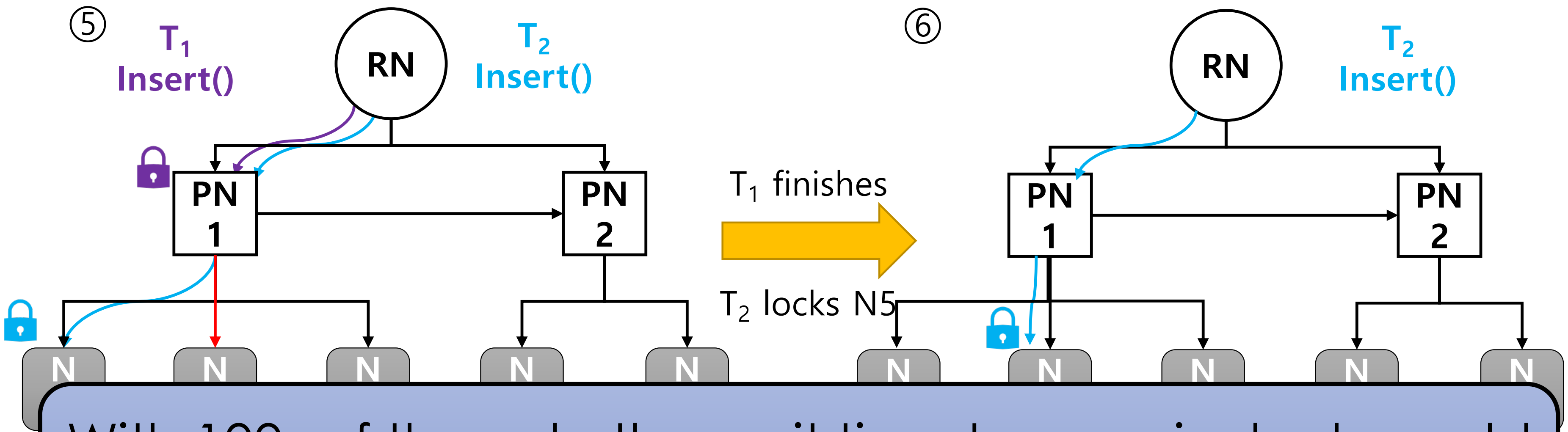
Challenge 1: Point of Contention



Challenge 1: Point of Contention



Challenge 1: Point of Contention



With 100s of threads, the wait time to acquire lock would increase, hence huge performance overhead.

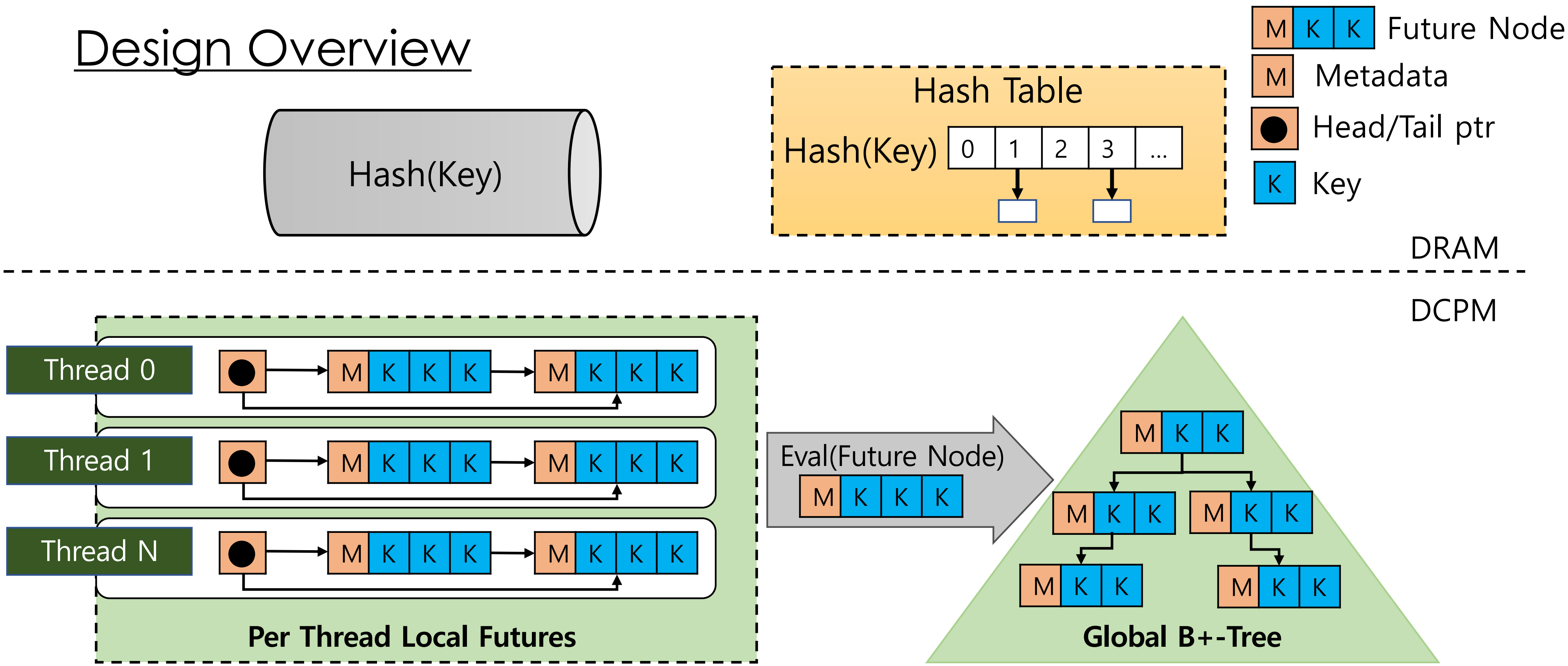
F^3-Tree for PM-based Manycore machines

- Future-based Fast&Fair B+-Tree
  - Guarantees high performance and scalability
  - Maintains read performance
- Employed future objects (FOs) at per-thread level
- Dedicated async threads to evaluate FOs to global B+-Tree
- Adopted hash table to overcome read performance

Proposed Idea 1: Future Objects (FOs)

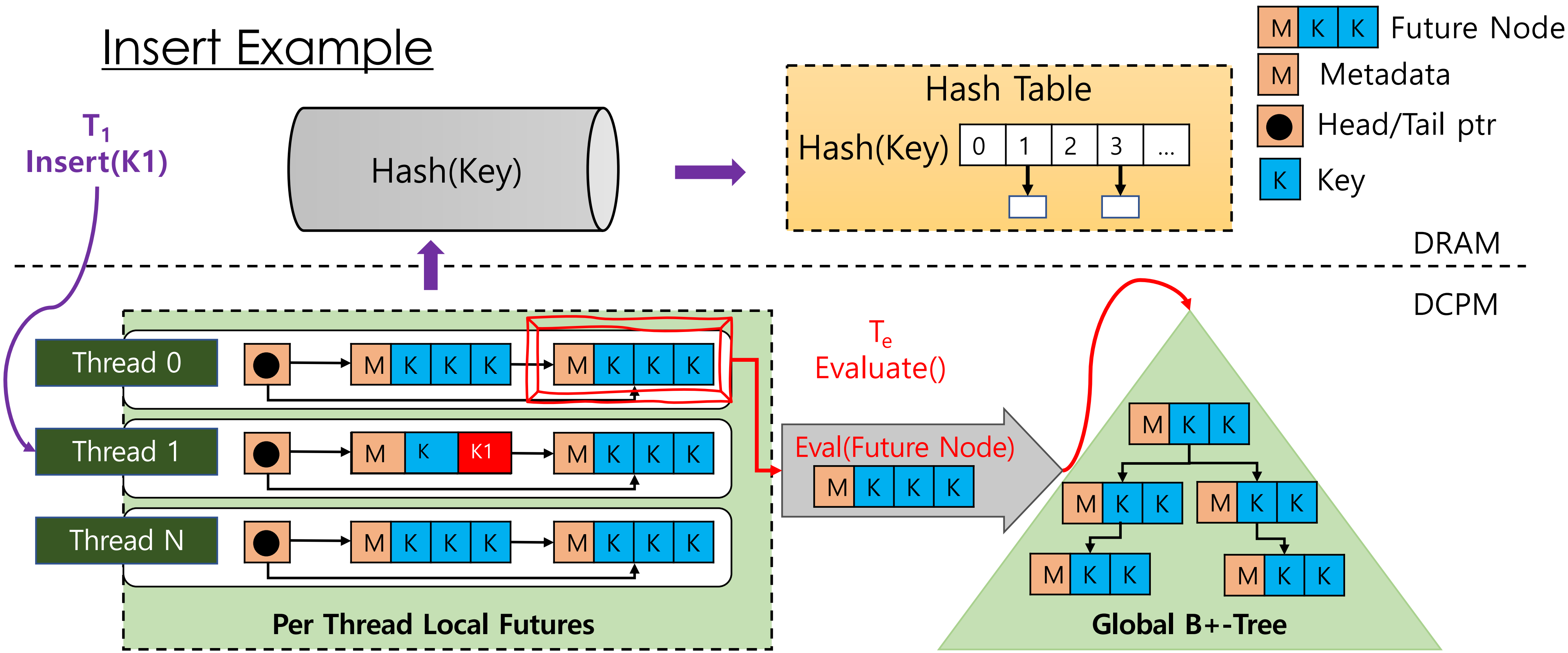
- Promises to deliver the results once evaluated
- Performance efficient for shared data structures
- Per-thread local future objects (PTFO)
  - A per-thread local LinkedList
  - Lock-free
  - Rely on durable linearizability for correctness

Design Overview



F3-Tree Design

Insert Example



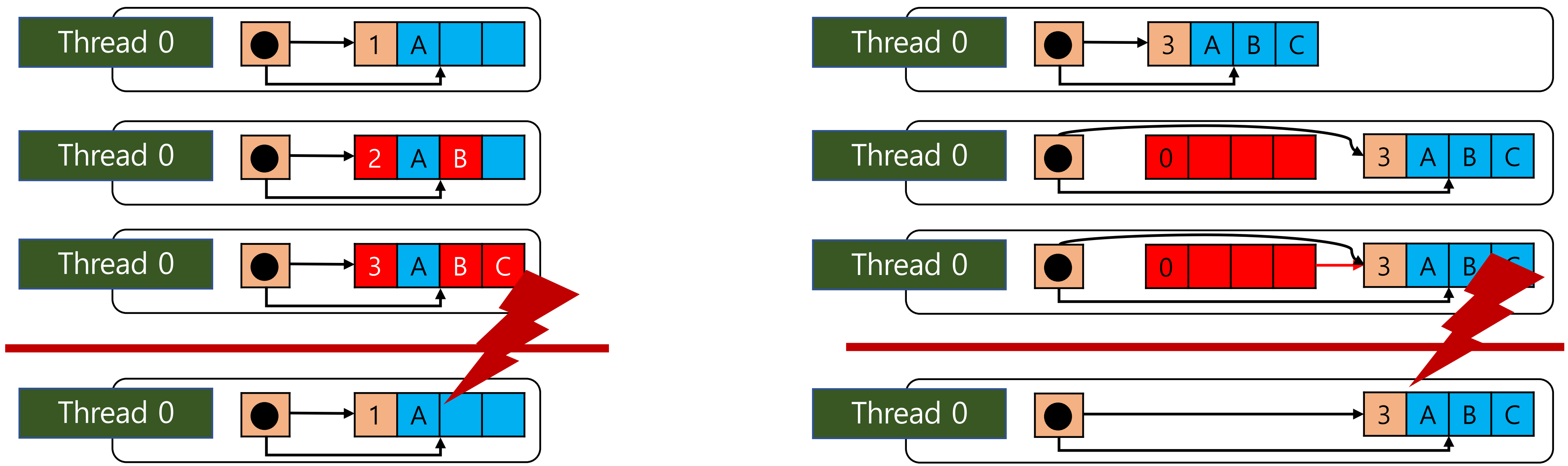
Challenge 2: Consistent View

- Inconsistent view:
  - Per-thread local future objects
  - Global B+-Tree

Challenge 2: Consistent View

Single future object case:

Multiple future objects case:



Red color represent non-persistent state

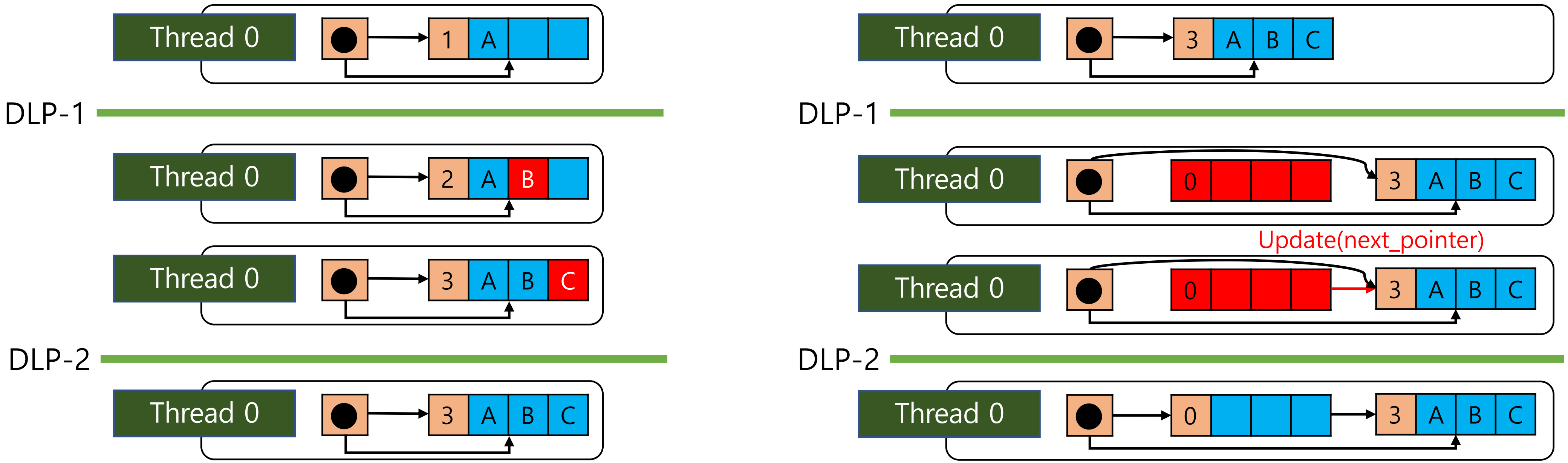
Proposed Idea 2: Durable Linearizability

- Common practice for correctness condition
- Operations take effect durably in between its invocation and response
- Promises the consistent view of the data structure
  - By reverting to previous DL Point

Linearizability - Examples

DLP within a Future Node

DLP in-between Future Node



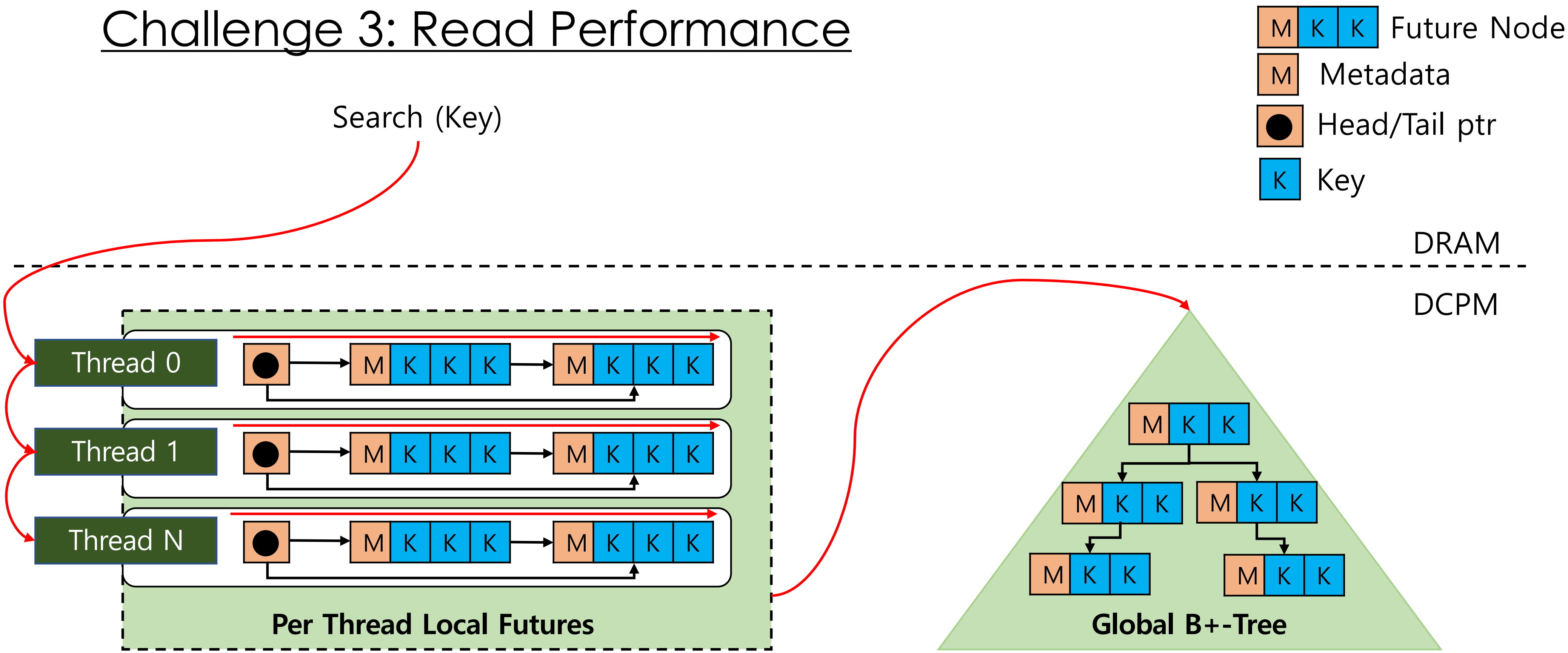
DLP – Durable Linearizability point

Red color represent non-persistent state

Challenge 3: Read Performance

- Key Lookup:
  - Huge performance overhead
  - Linear search at per-thread local future objects
  - Binary search at the global B+-tree
  - O(M)+O(log N)
    - Where M is no. of future objects at PTFO
    - N is the no. of global B+-tree nodes

Challenge 3: Read Performance

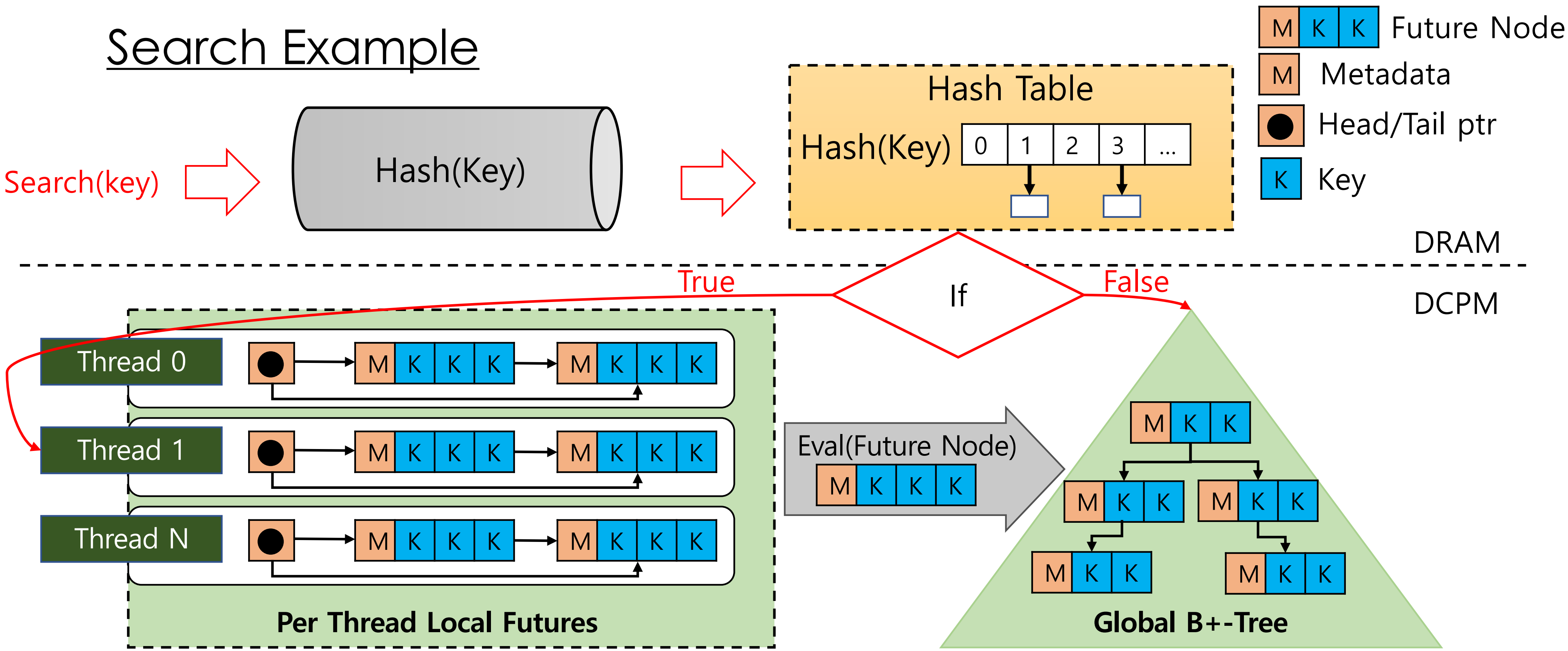


Proposed Idea 3: Hash Tables

- Directly access the thread's local future objects
- O(1) + O(Log N)

F3-Tree Design

Search Example

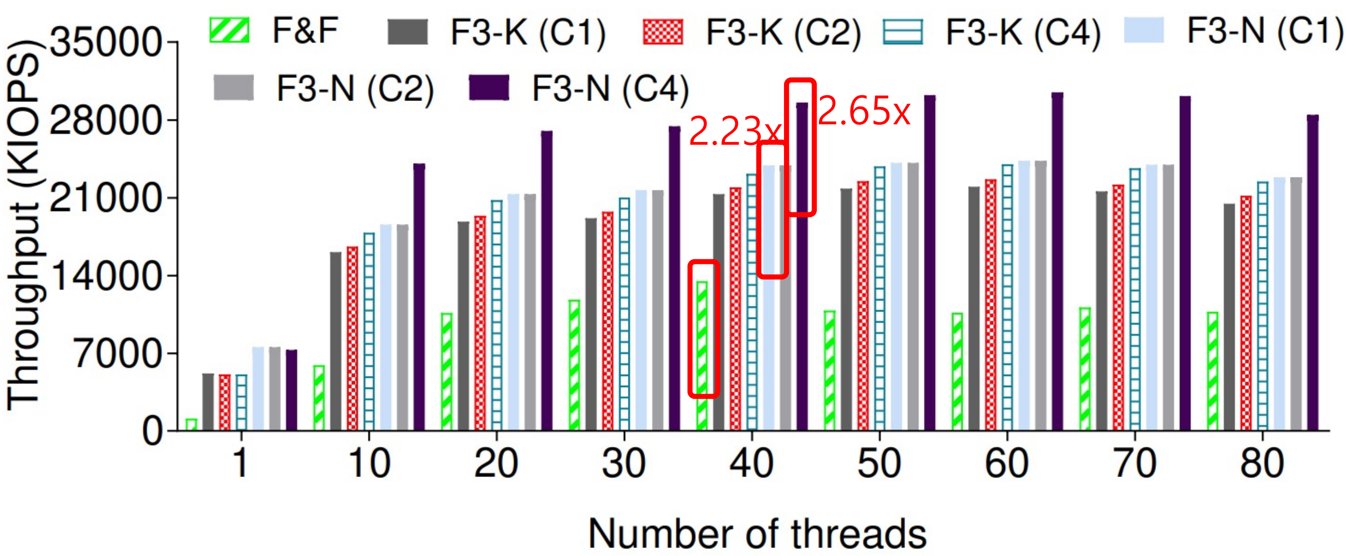


Experiment Setup

- Xeon(R) E5-4640 v2 CPUs@ 2.20 GHz
  - **10 physical cores per node**
- 256 GB DDR3 DRAM
- Linux kernel v5.4.0
- Synthetic Workload
  - **Sequential and Random**

- Compared against:
  - **Fast&Fair:** existing state-of-art PM-based B+-Tree
  - **F3-K**: F3-tree with key-based evaluation method
  - **F3-N**: F3-tree with future node-based evaluation method

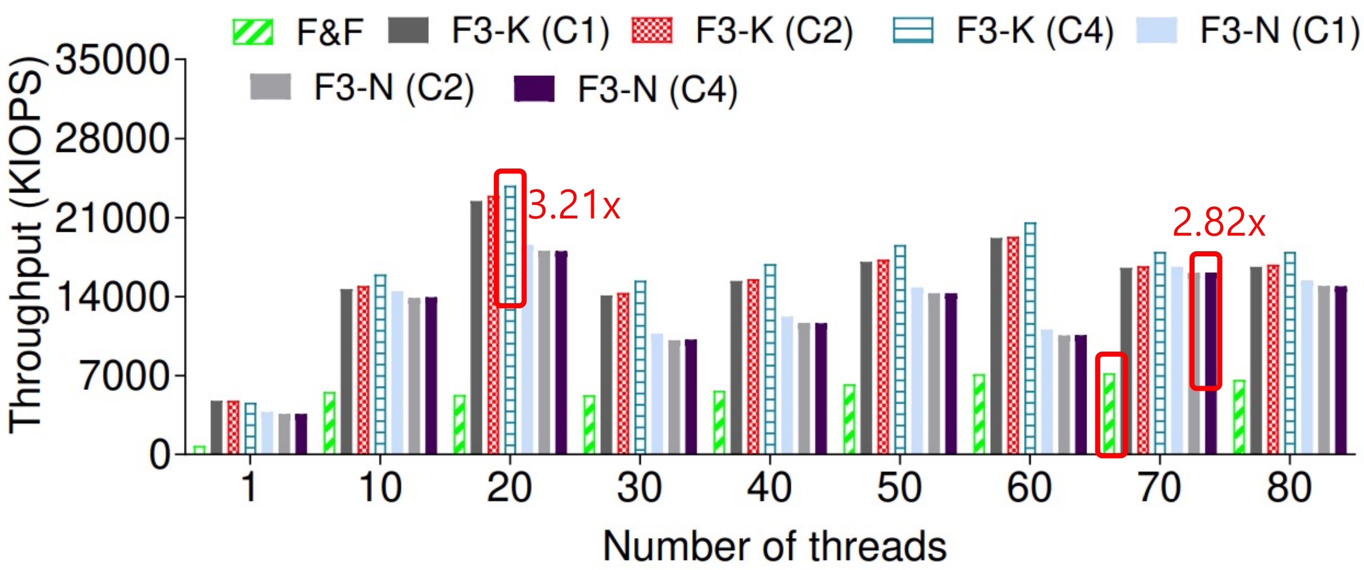
Sequential Workload Analysis



Sequential workload

Cx – x number of evaluate threads

Random Workload Analysis

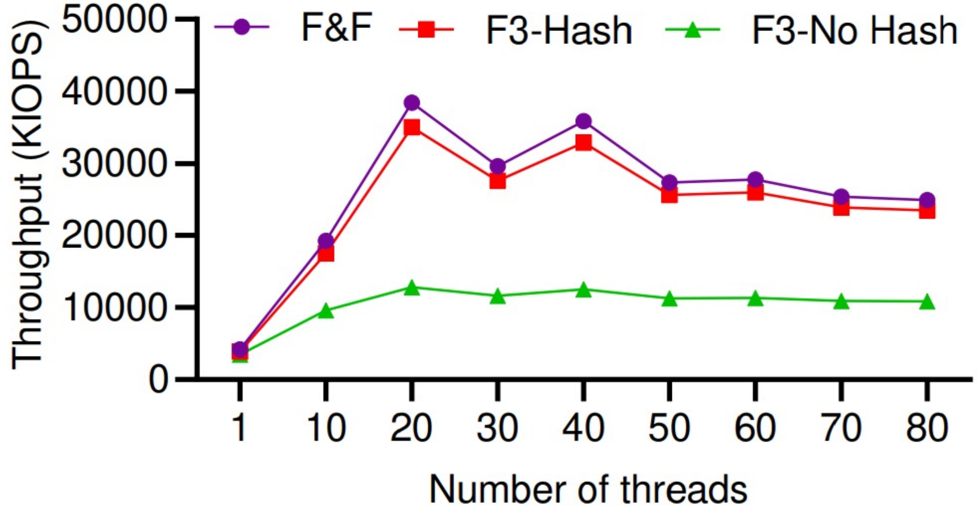


Random workload

Cx – x number of evaluate threads

Search performance & realistic workload

- F3-No Hash linear search the PTFO of a key
- F3-Hash first lookup a key in hash table
- 20% data in PTFO (Hash table) and 80% in global B+-tree



Search Performance

Conclusion

Summary

- ❑ Adoption of PM-based manycore machines in cloud computing
- ❑ Scalability problems for indexing data structures
- ❑ F3-Tree targets manycore machines
  - ❑ Per-thread local future objects, global Fast&Fair B+-Tree, and hash table
  - ❑ Achieves higher performance than counter parts

Thank you

Safdar Jamil

Email: safdarjamil95@gmail.com

Web: https://sites.google.com/view/safdarjamil95/