No Cap, This Memory Slaps: Breaking Through the **Memory Wall** of **Transactional** Database Systems with **Processing-in-Memory**

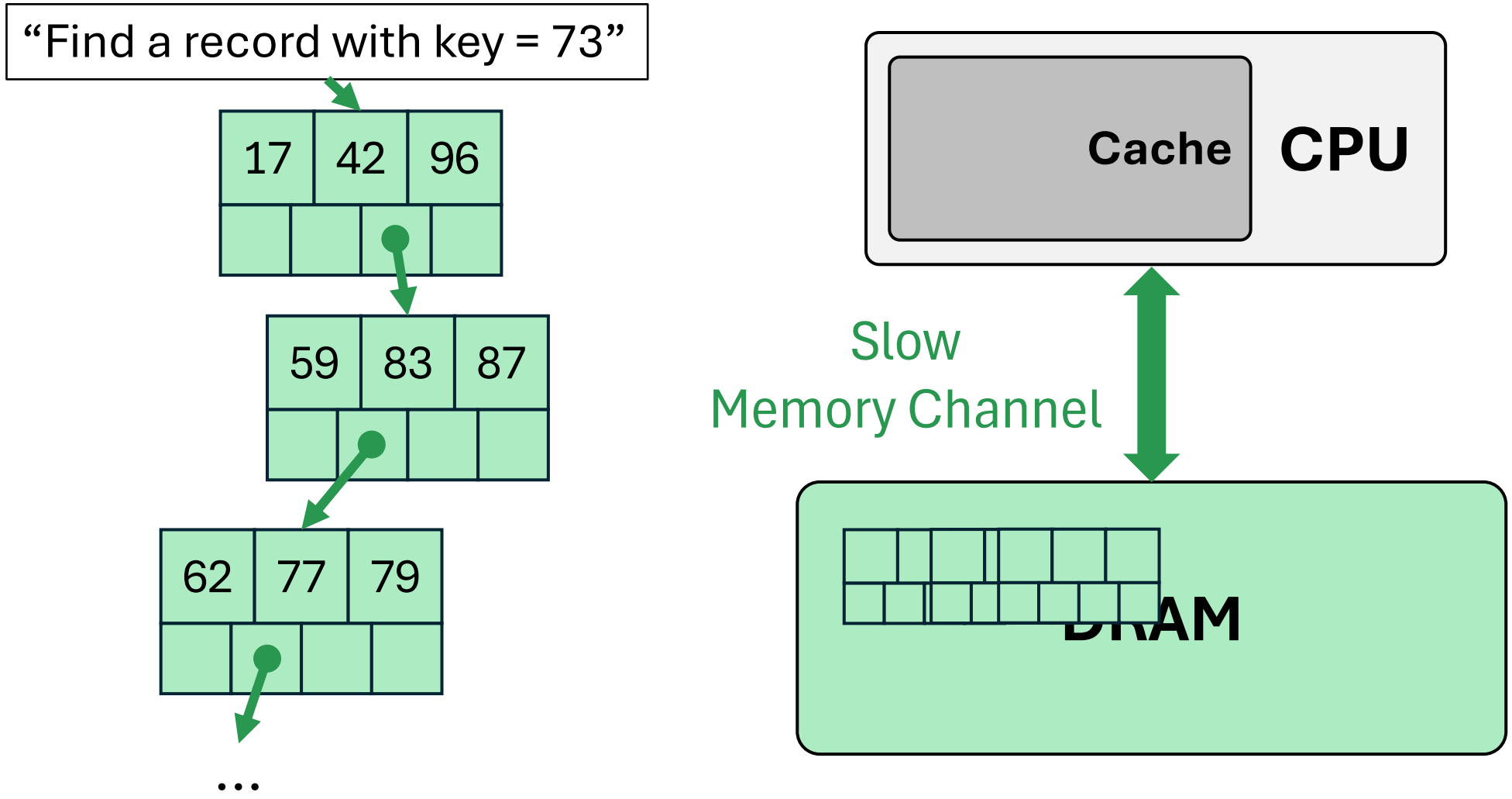
Hyoungjoo Kim, Yiwei Zhao, Andrew Pavlo, Phillip B. Gibbons

Carnegie Mellon University

VLDB 2025

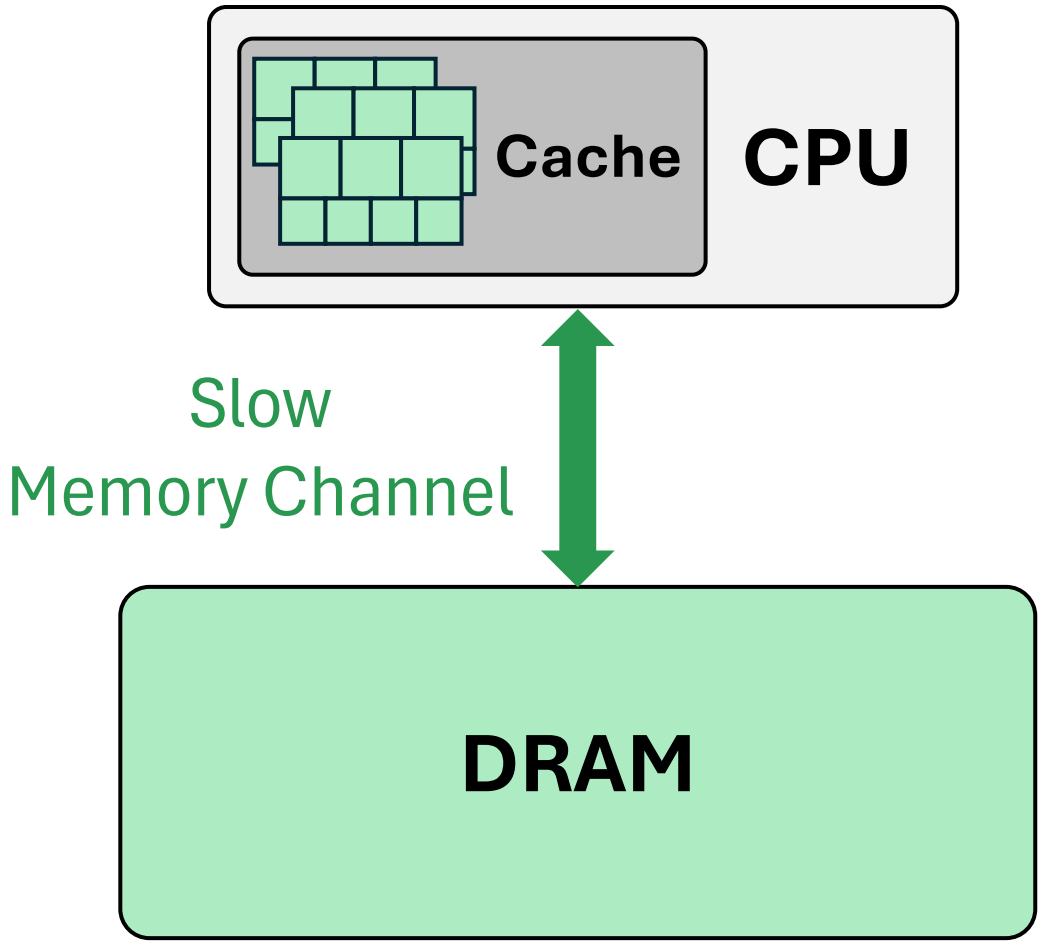
Research 38: Data Management on Novel Hardware

Data Movement is the Bottleneck in OLTP

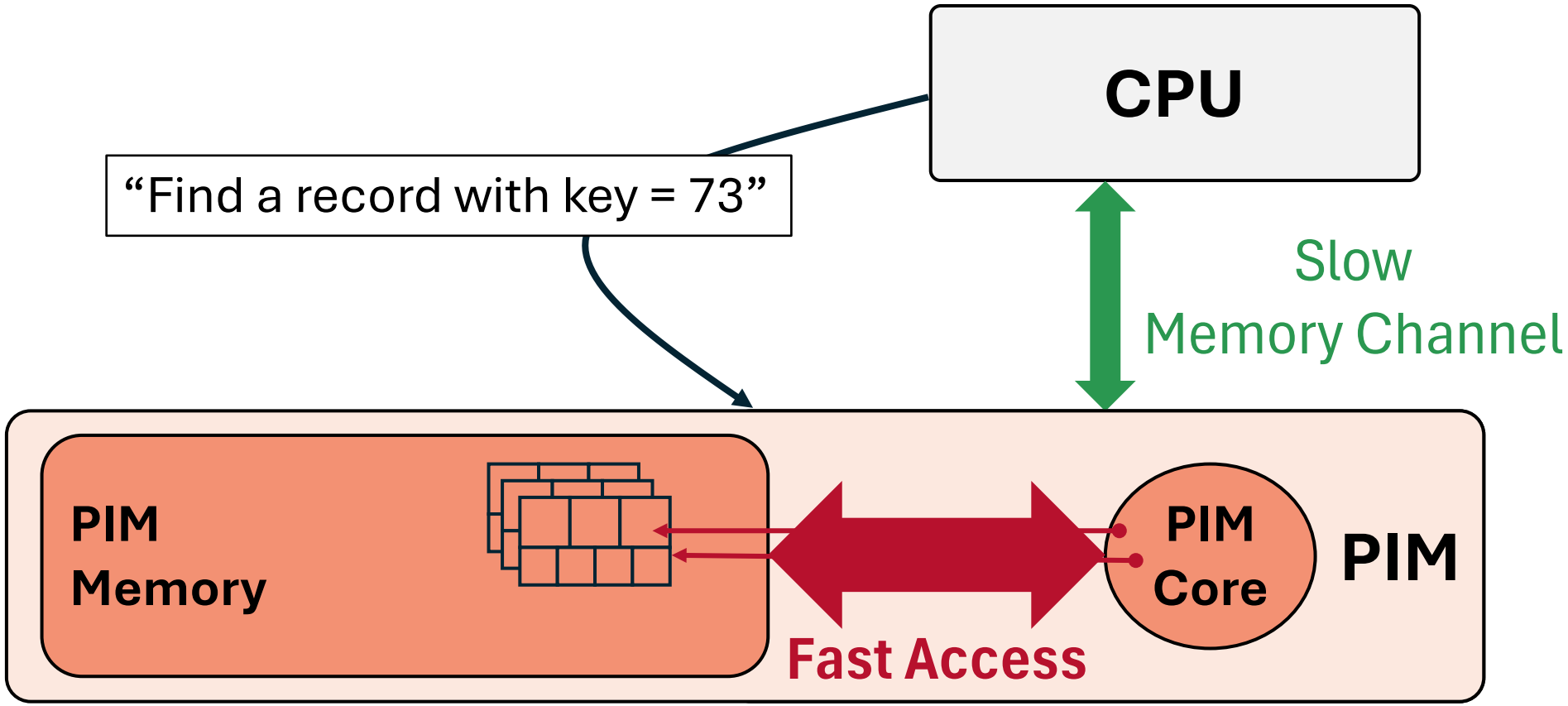


Data Movement is the Bottleneck in OLTP

- CPU Cache?
  - + Good if the working set is small
  - **- Capacity is not scalable**
- Multithreading?
  - + Hides memory latency
  - - **Still bounded by channel BW**



Processing-in-Memory (PIM)
can Overcome the Data Mvmt Bottleneck

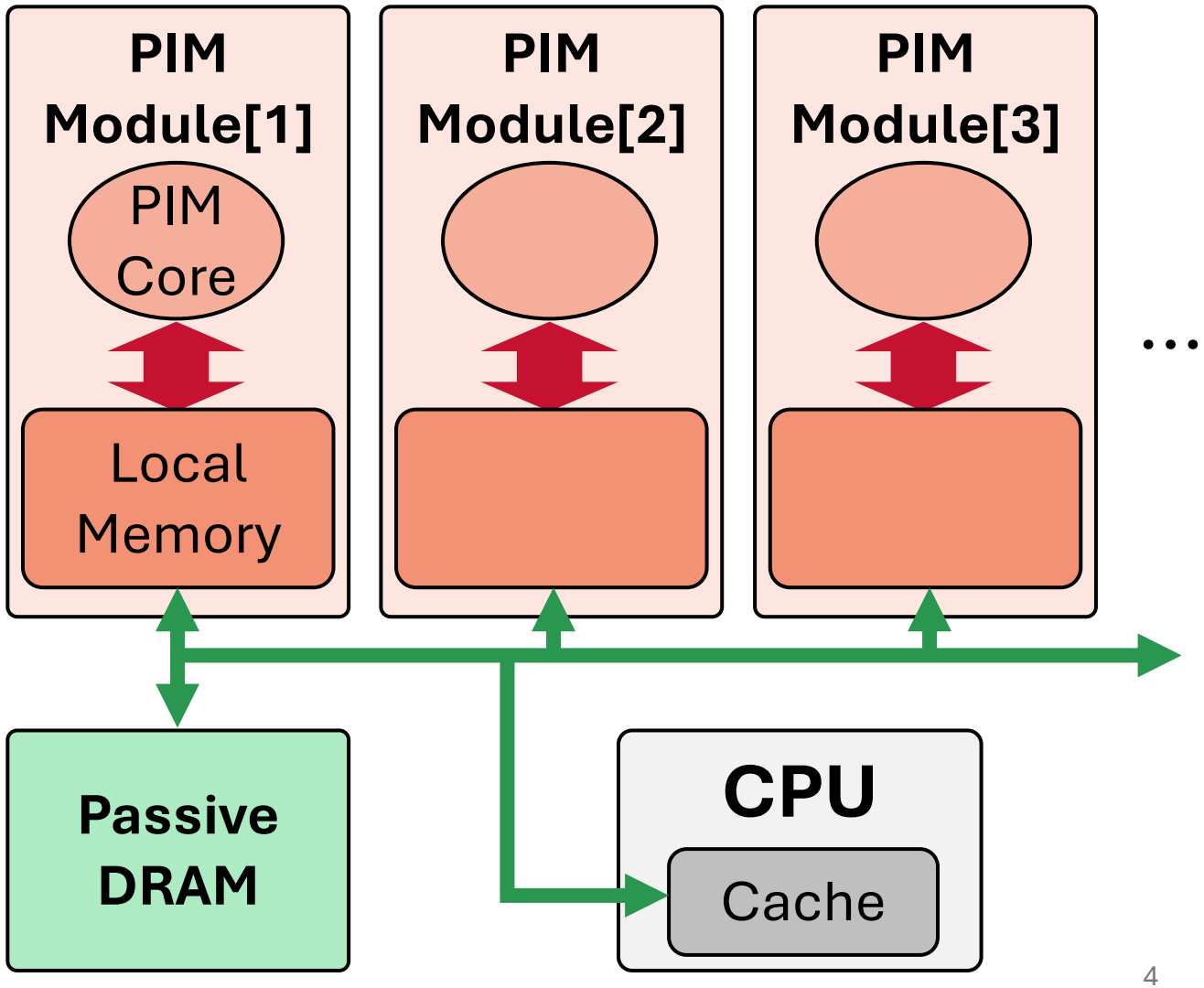


→ PIM overcomes the bottleneck by processing DRAM data in place

PIM is Divided into **1000s of Modules**

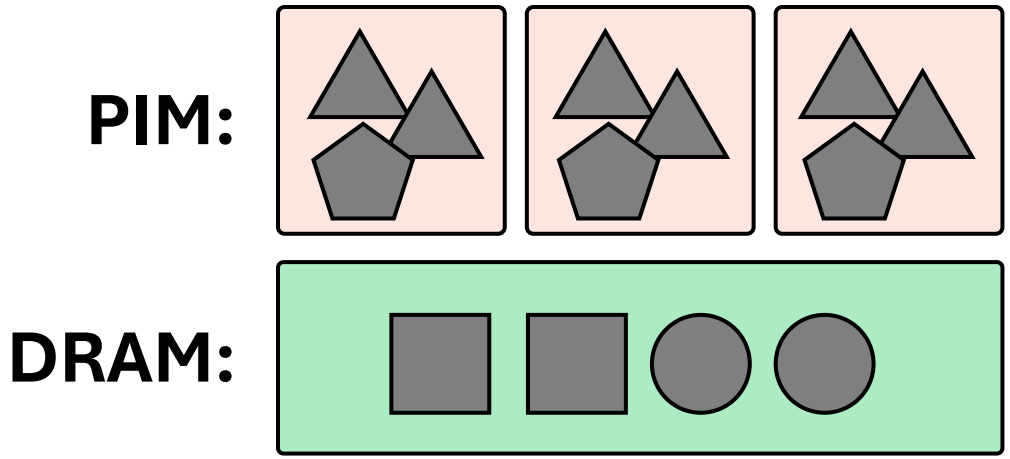
- Shared-nothing DB, ***but*...**

- Very fine-grained
  - Each module has small capacity and weak core
- Powerful central CPU
- Also use passive DRAM

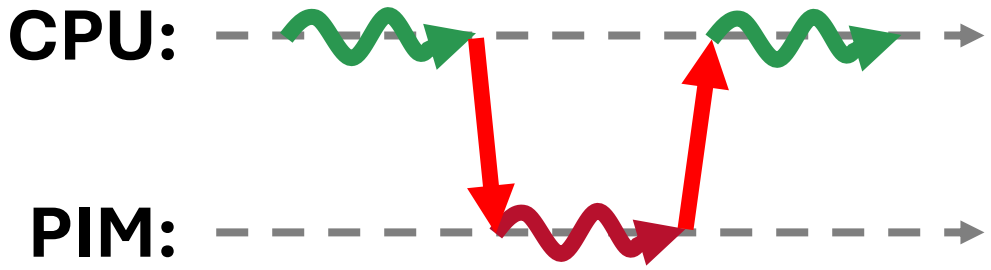


Naive Shared-Nothing Design is **not Enough**

- Where to place each data?
  - PIM vs. Passive DRAM
  - Partitioning across PIM modules



- How to execute transactions?
  - PIM offload latency

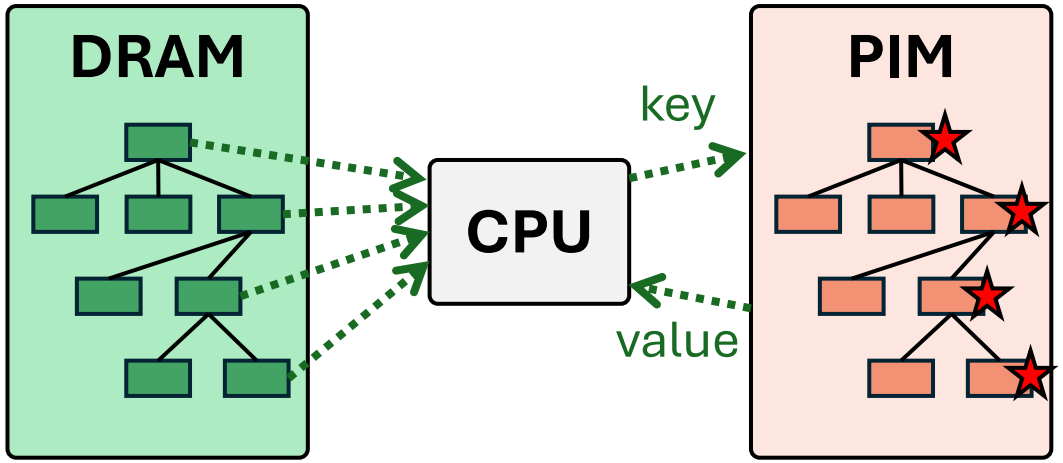


→ We present **OLTPim**!

OLTPim Places Data using **Near-Mem Affinity**

ex_1) Traversing a B+ tree

(Similar for version chains)

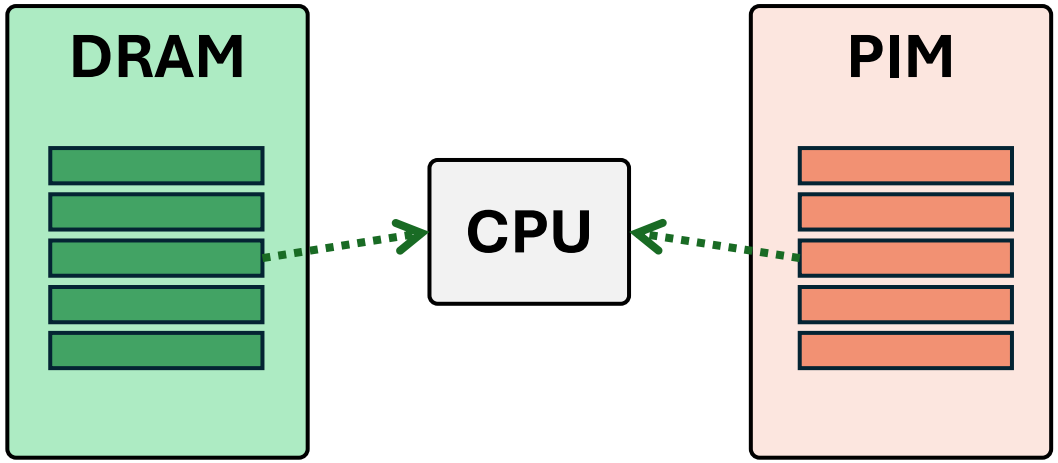


(Height * NodeSize) **>** **(Key + Value)Size**

→ Large advantage of using PIM

→ Place Indexes & Version chains in PIM

ex_2) Fetching a tuple

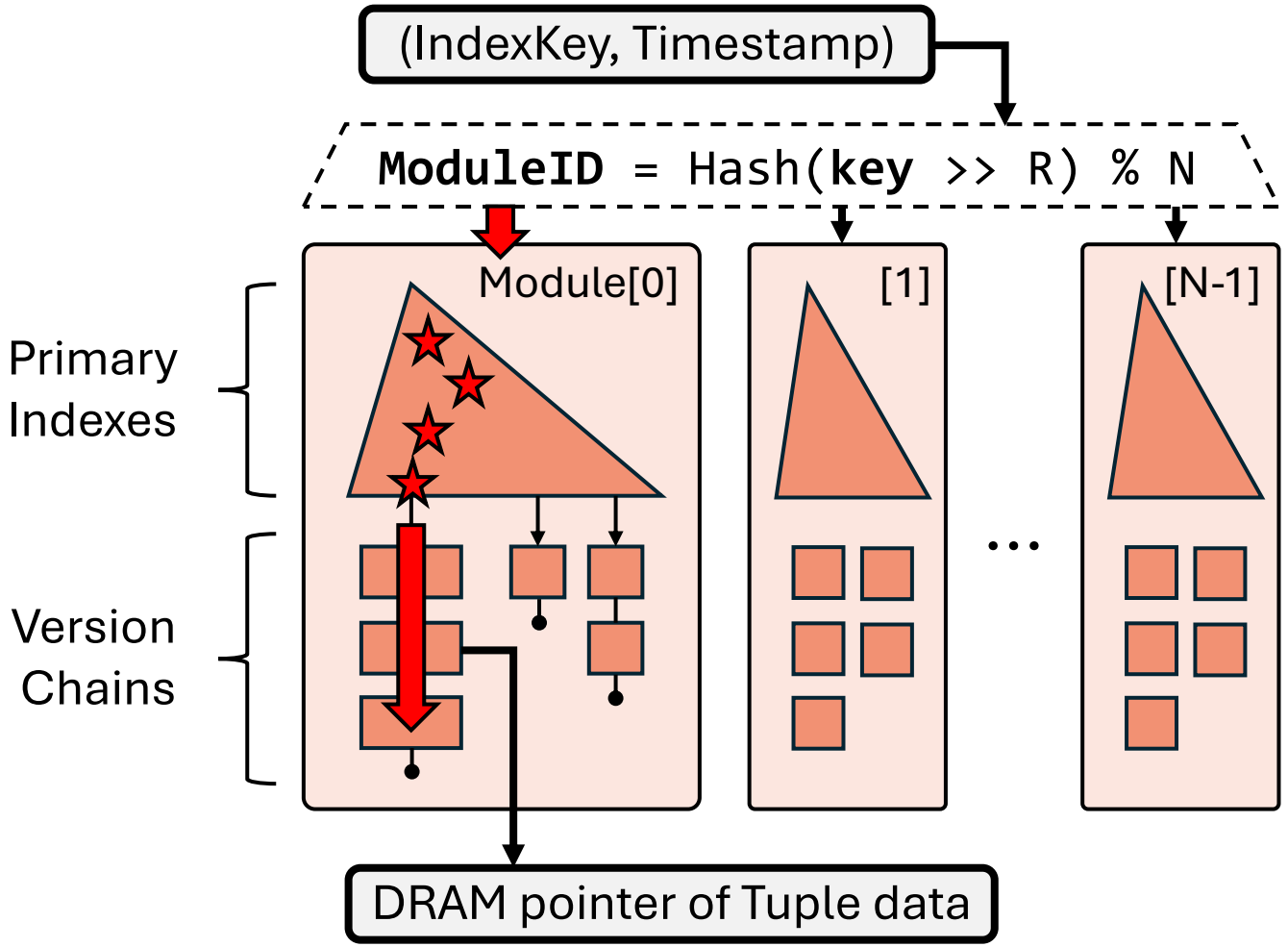


(TupleSize) **=** **(TupleSize)**

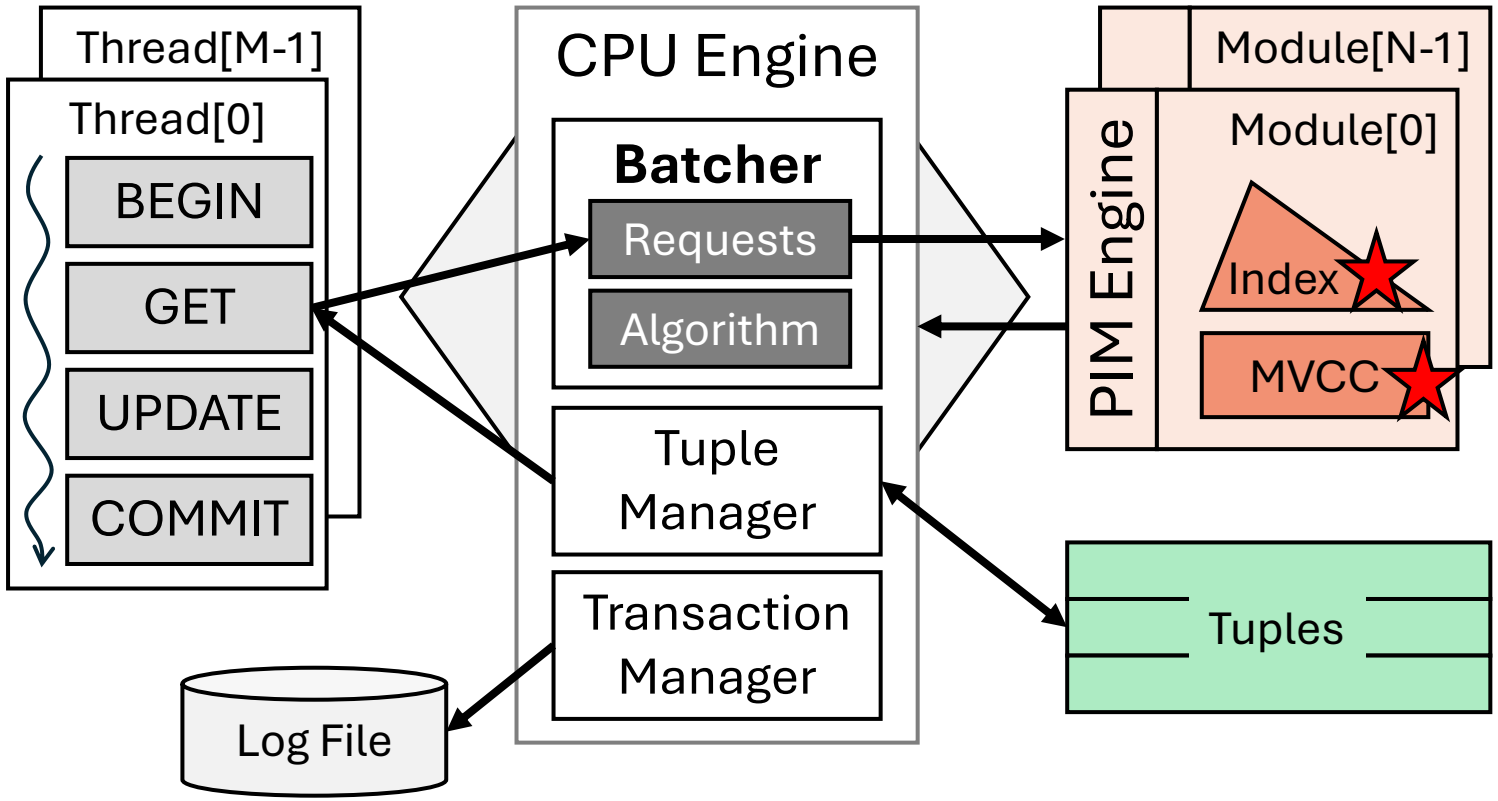
→ No advantage of using PIM

→ Place Tuple data in DRAM

PIM-side is **Partitioned** Based on Primary Key

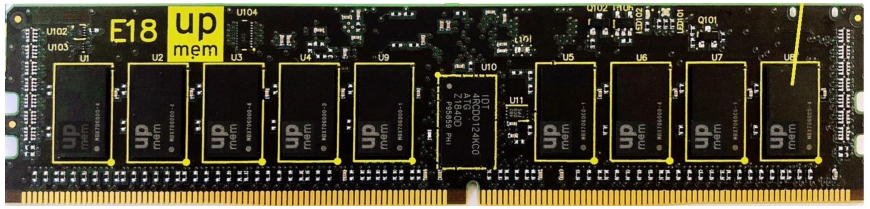


Requests are **Batched** to Hide PIM Latency



We Evaluated OLTPim on Real PIM System

- Dual-socket server with 2x Xeon Silver 4216
  - Each CPU has 32 threads, 2.1GHz, 22MB LLC
  - Total 12 memory channels
    - 8 channels: **UPMEM PIMs** (Total 2048 modules = 128GB, each core at 350MHz)
    - 4 channels: Normal DRAMs
    - Each with DDR4-2400MT/s



- Baseline: MosaicDB[1]
  - State-of-the-art research OLTP DBMS; Inherits code of Silo, ERMIA, CoroBase
  - Evaluated on the same server, except:
    - All 12 memory channels are equipped with Normal DRAMs to achieve maximum aggregated memory channel bandwidth

[1] Huang K., et. al., “The art of latency hiding in modern database engines”, PVLDB 17(3)

OLTPim Achieves **Higher Throughput** with **Less Memory Channel Traffic**

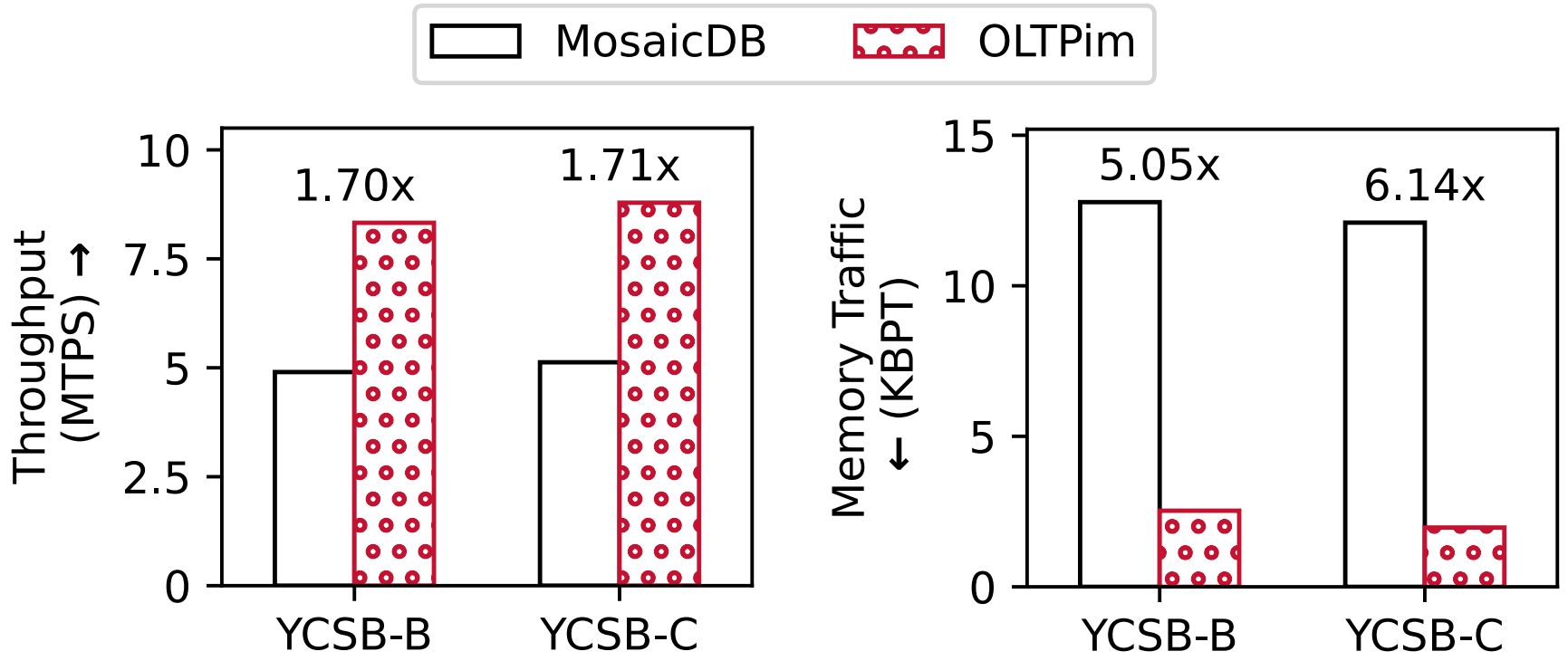
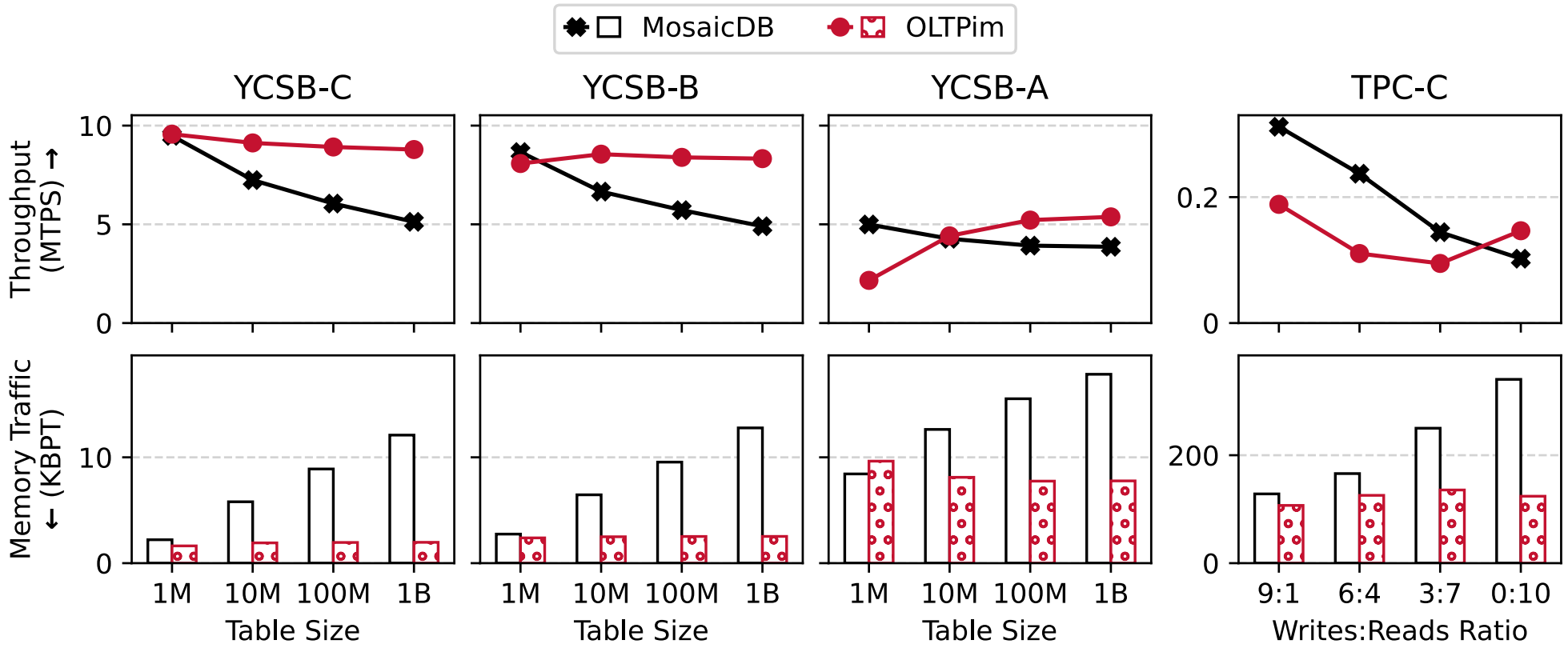


Table with 1B rows, 10 rows per transaction. YCSB-B (Read:Update = 95:5), YCSB-C (100% Read)

OLTPim Reduces Memory Traffic also on Other Workloads



→ Memory traffic remains the same on larger datasets

→ Performs better on the read-only workload

More in our paper!

- Formal definition of near-memory affinity
- Range Queries, Secondary Index, Garbage Collection
- Write Sets, Logging, Recovery
- Methods to Avoid OS Overheads on Batching
- Real-World Implementation Challenges
- Discussions on Future PIM Hardware

Key Takeaways

- **PIM** can reduce the **memory channel traffic** on OLTP workloads
  - Improves throughput and saves energy
  - But naive OLTP+PIM is not enough

- **OLTPim** shows up to 1.7x higher throughput with 6.1x less memory traffic
  - Principled **partitioning** of DBMS components
  - Optimized **batching** algorithm