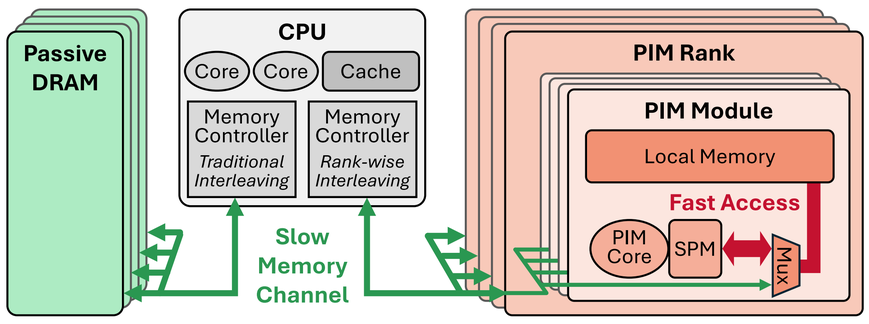
No Cap, This Memory Slaps: Breaking Through the **Memory Wall** of **Transactional** Database Systems with **Processing-in-Memory**

Hyoungjoo Kim, Yiwei Zhao, Andrew Pavlo, Phillip B. Gibbons — Carnegie Mellon University

1. Motivation & Background

- **Data movement is the bottleneck in OLTP**
  - Slow memory channel between CPU & DRAM
  - CPU cache reduces memory traffic, but…
    - Capacity (<100MB) << OLTP working set (>100GB)
  - Multithreading hides memory latency, but…
    - Memory traffic remains the same
    - Throughput is still bounded by memory channel bandwidth
- **Processing-in-Memory (PIM) can overcome this**
  - PIM: small on-chip cores on DRAM
  - Divided into 1000s of isolated modules (banks)



- **Challenges in PIM + OLTP**
  - Place each data on DRAM or PIM?
  - How to partition data across PIM modules?
  - How to minimize the impact of PIM offload latency?

2. OLTPim Design

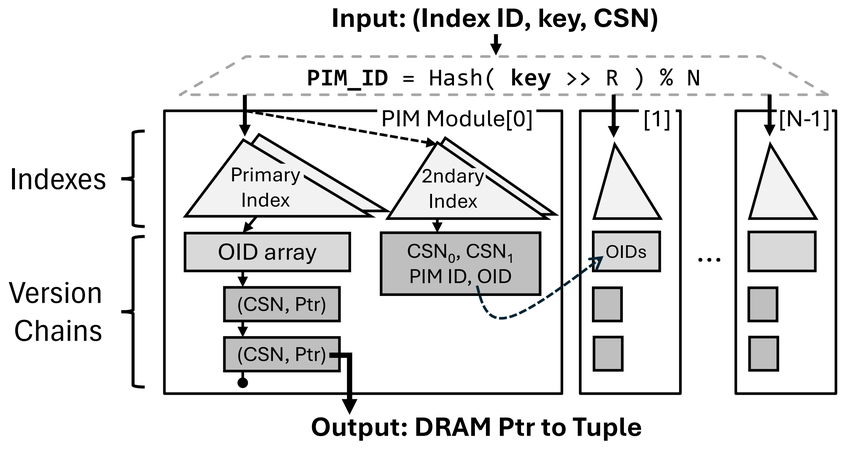
- **PIM vs. DRAM: based on near-memory affinity (NMA)**
  - We define *NMA = (Traffic if DRAM) – (Traffic if PIM)*
  - Ex1) B+ tree, linked list traversal
    - If in DRAM: fetch multiple nodes (high off-chip traffic)
    - If in PIM: on-chip traversal (low off-chip traffic)

    → PIM is better. High NMA

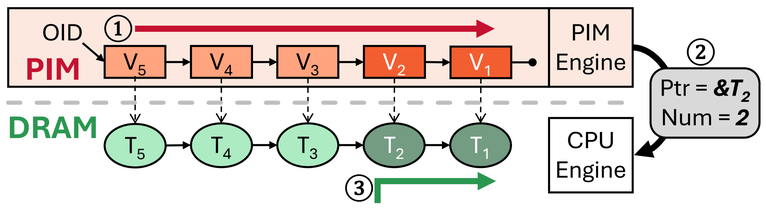
  → Indexes & version chains on PIM
  - Ex2) Tuple fetch
    - If in DRAM: fetch the row
    - If in PIM: fetch the row (same off-chip traffic)

    → No advantage of PIM. Zero NMA

  → Tuple data on DRAM



- **Partitioning within PIM: based on primary key**
  - Hash function maps (2^R) adjacent keys to one module
    - Helpful for range queries
  - 2ndary idx: also uses hash, value points the ver. chain
    - 2 dependent module accesses required
  - Version chains and tuples are managed separately
    - Garbage collection with redundant DRAM-side tuple chain
      - First do GC on PIM-side version chain
      - Then traverse DRAM tuple chain and release



(continued)

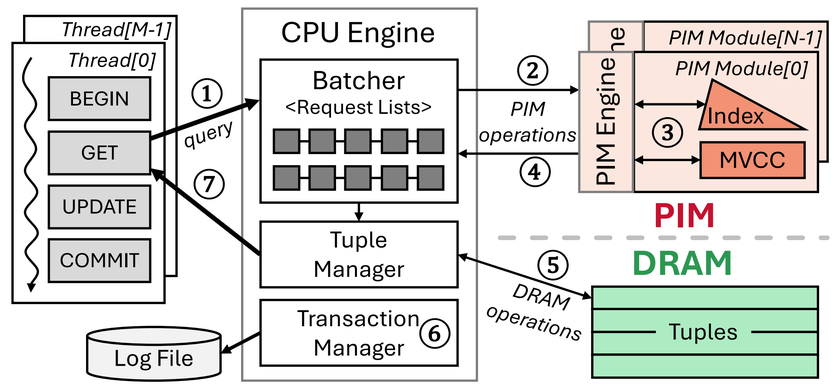
- **Hide PIM offload latency: use lightweight batching**
  - *M* threads : *N* PIM modules multiplexing required

    → Batcher in the CPU engine handles this
  - System-wide batching is expensive

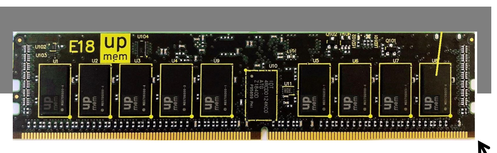
    → (PIM) rank-wise batching
  - CPU cores are already busy, no room for batcher task

    → Flat combining algorithm with coroutines
  - PIM offload has too much overhead in the current impl.

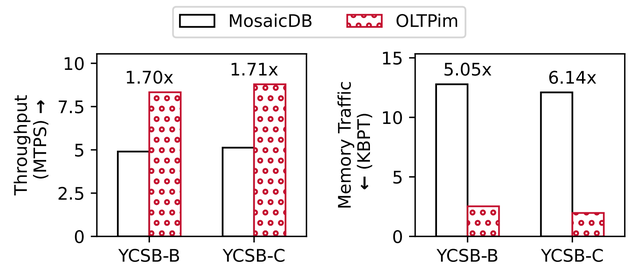
    → New lightweight API for UPMEM



3. Evaluation

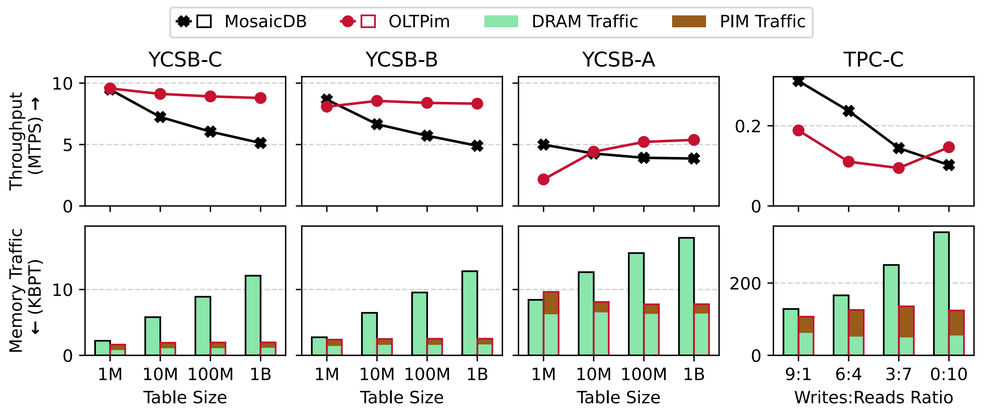


- **Experiment Setup**
  - 2 sockets of Xeon Silver (each 32 thds, 2.1GHz, 22MB LLC)
  - 12 memory channels with DDR4-2400MT/s
    - 8 channels with UPMEM DIMMs
      - Total 2048 PIM modules = 128GB, cores @ 350MHz
    - 4 channels with normal DRAM DIMMs
  - Baseline: MosaicDB[PVLDB 17(3)] (SOTA CPU-only DBMS)
    - Use the same server, but all channels are with DRAMs
  - Workloads: YCSB, TPC-C
    - Read : Update = 100:0 (YCSB-C), 95:5 (B), 50:50 (A)



- **Higher transaction throughput with less mem traffic**
  - CPU-only: becomes worse with increasing workload sizes
  - OLTPim: tput & traffic are same even on large workloads

    * Small table OLTPim tput decrease: due to high-contention GC
  - TPC-C: OLTPim uses less traffic, read-only has higher tput



- **Higher (but tolerable) latency and abort rate**
  - OLTPim requires a larger batch size for optimal throughput
    - b/c PIM offload latency > DRAM access latency
  - Batching makes P99 latency higher, but still <10ms