PID-Join: A Fast In-Memory Join Algorithm for Commodity PIM-Enabled DIMMs [SIGMOD '23]

Chaemin Lim[1], Suhyun Lee[1], Jinwoo Choi[1], Jounghoo Lee[1], Seongyeon Park[2], Hanjun Kim[1], Jinho Lee[2], and **Youngsok Kim**[1]

[1]Yonsei University
[2]Seoul National University

Real-World PIM Tutorial @ MICRO '23
October 29, 2023

In-Memory Databases & Join

- In-memory DBs store their tables **in main memory**.
  - CPUs access the main memory, not disks, to access tuples.
- **Join** combines columns from 1+ tables into a new table.
  - A key relational operation of in-memory DBs
  - e.g., `SELECT R.key, R.name, S.quantity FROM R, S`
    `WHERE R.key = S.key`

key	name
0	A
1	B
2	C

R

$\bowtie_{R.key = S.key}$

key	quantity
1	100
2	200
3	300

S

=

R.key	R.name	S.quantity
1	B	100
2	C	200

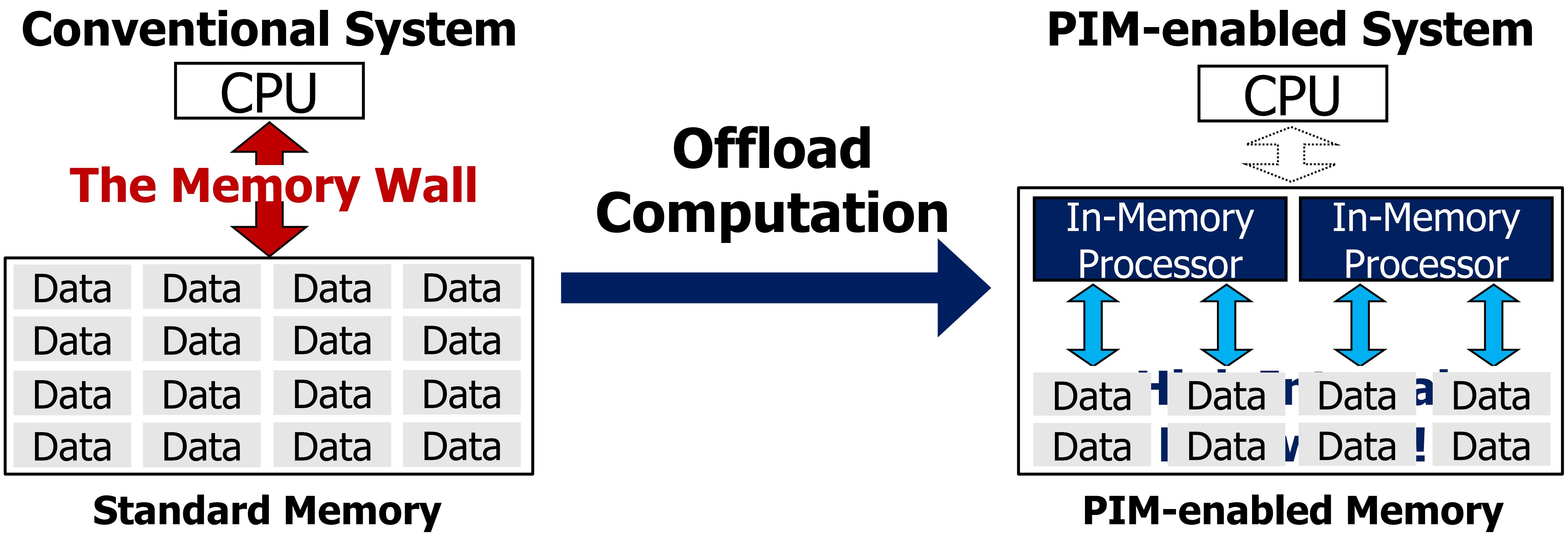
Join is Memory-Intensive!

- Join is a **memory-intensive** operation.
  - Accesses throughout the two tables to join the tuples.
- **High memory access bandwidth** is essential.
  - For accelerating the memory-intensive in-memory join
- Conventional systems **suffer from limited memory B/W.**
  - Hard to achieve higher memory B/W over the memory channels

The **"memory wall"** has become a major **performance bottleneck** for in-memory joins!

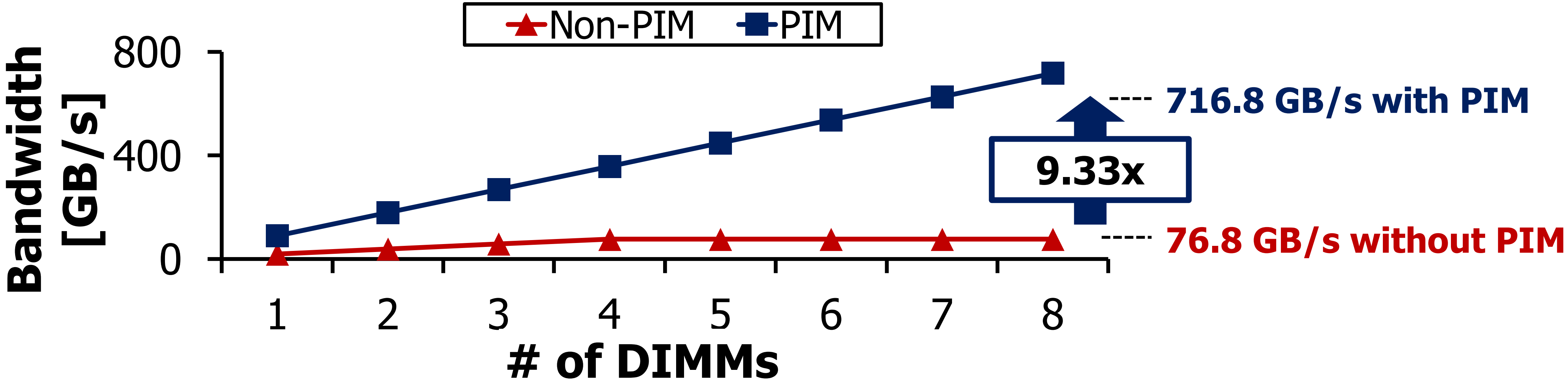
Processing-In-Memory (PIM)

- A **promising solution** to overcome the memory wall
- Achieves **significantly higher memory B/W** over CPUs!
  - By offloading computation to the in-memory processors



PIM Can Greatly Accelerate Joins!

- **PIM** provides **9.33x** higher memory bandwidth!
  - Adding more PIM devices further increases the B/W
  - Not bounded to the count of memory channels



PIM can fully exploit the high internal bandwidth and can greatly accelerate in-memory joins!

Prior Studies on PIM-Assisted Joins

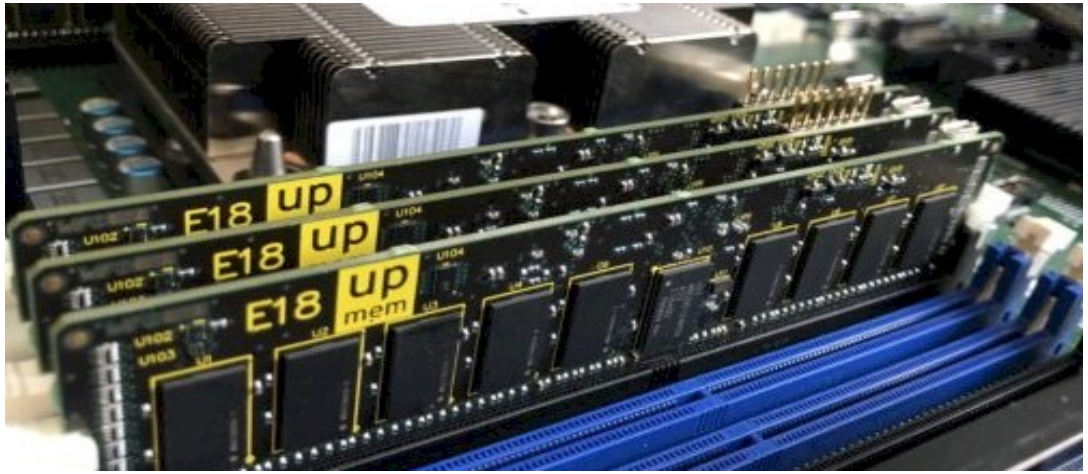
- **Key Limitation**: Incompatible with PIM-enabled DIMMs
- **Limitation #1**: Focused on 3D-stacked memory (HMC)
  - Exhibit architectural characteristics different from those of DIMMs
- **Limitation #2**: Rely on cycle-level timing simulations
  - Inaccurate on real systems due to hardware modeling errors

Prior Study	PIM Architecture	Real System?	Publicly Available?
Mirzadeh et al. [ABDS ’15]	3D-Stacked	✗	✗
Drumond et al. [ISCA ’17]	3D-Stacked	✗	✗
Kepe et al. [VLDB ’19]	3D-Stacked	✗	✗
Boroumand et al. [ICDE ’22]	3D-Stacked	✗	✗
PID-Join [SIGMOD ’23]	**DIMM**	✓	✓

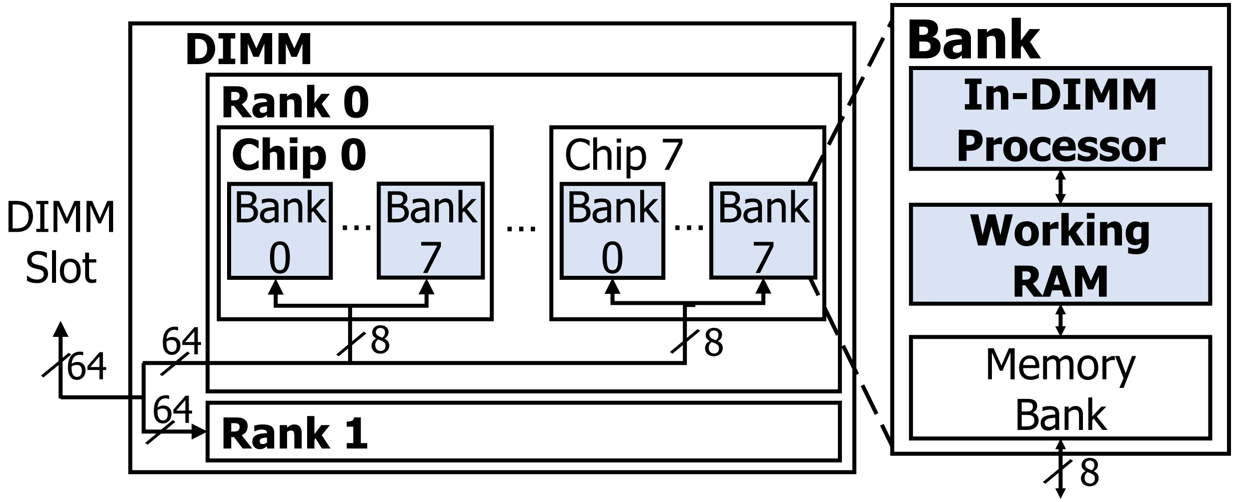
Prior studies on PIM-assisted join & our work

Characteristic #1: Per-Bank Processors

- A DIMM has ranks, chips, and banks in a hierarchical manner.
- **PIM-enabled DIMMs** place **one processor per bank**.
  - In-DIMM processors (IDPs) & working RAMs (WRAMs) to each bank
  - IDPs can **perform computation** & **WRAM-bank data transfers**.
  - e.g., UPMEM DIMMs [HotChips '19], Samsung AxDIMM [HotChips '21]



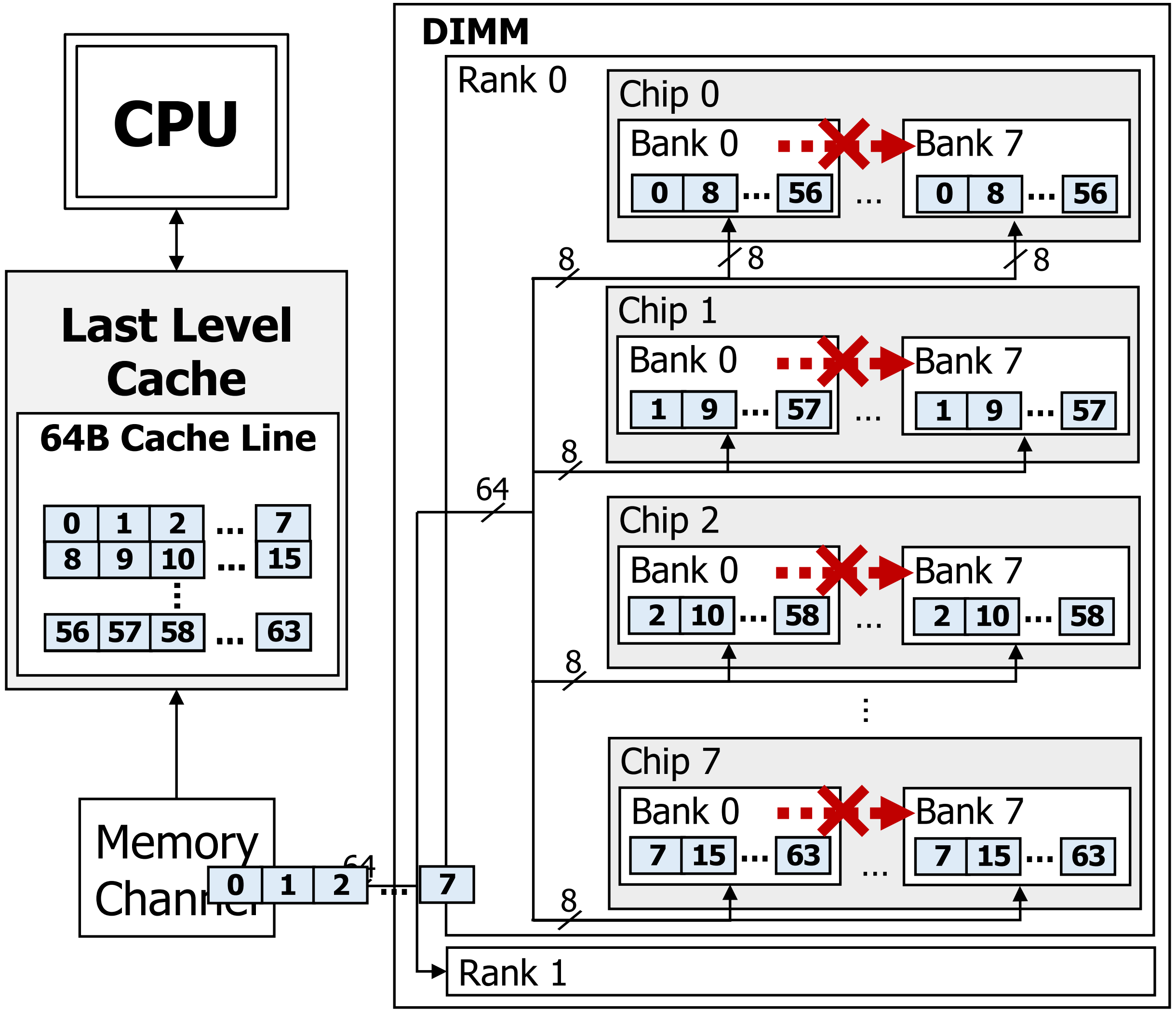
Commercial PIM-enabled UPMEM DIMMs



Internal architecture of an UPMEM DIMM

Characteristic #2: Shared-Nothing Architecture

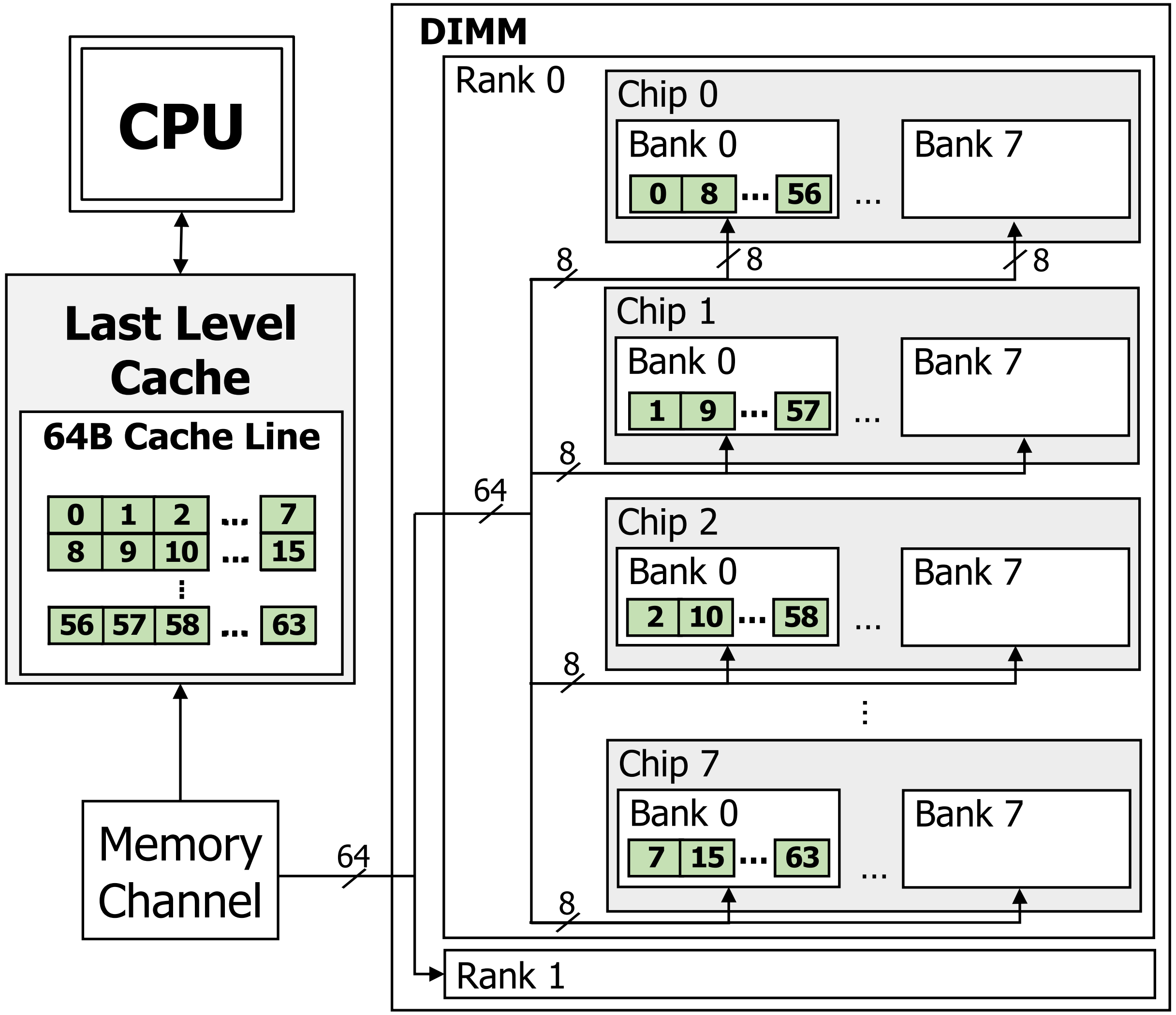
- **No direct inter-bank communication paths** between the banks.
  - CPU must mediate all data transfer through memory channels.
  - CPU needs to **collect** data from the source banks and **distribute** the data to the destination banks.



Working model of inter-bank communication

Characteristic #3: Memory Interleaving

- Sequential data blocks get **byte-interleaved** across ranks/chips/banks.
  - Provide high memory band-width by accessing multiple banks in parallel
  - **Transpose is necessary** for transferring sequential data between the banks.



Working model of memory interleaving of a DIMM

Design Goals

Fast in-memory joins on "real PIM-enabled DIMMs"

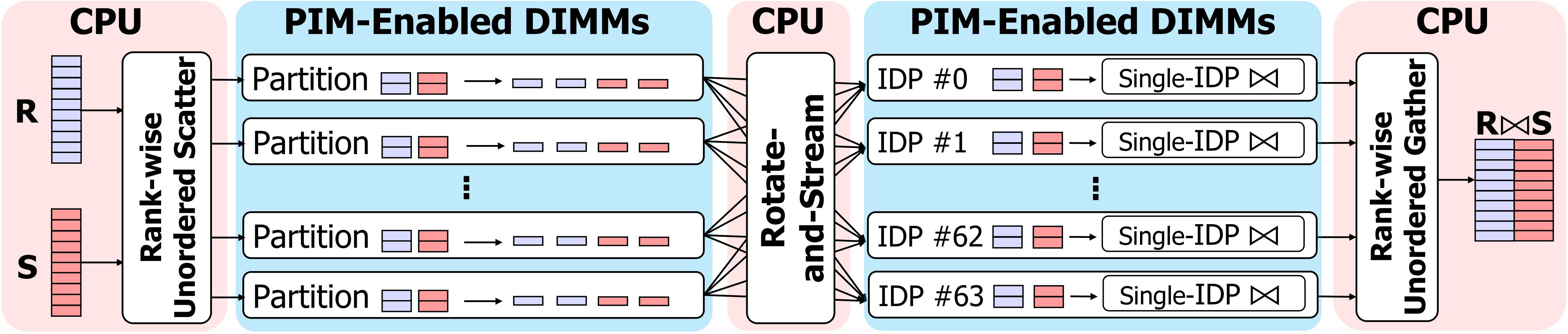
- **Goal #1: Compatible with stock PIM-enabled DIMMs**
  - Prior studies only support 3D-stacked PIM and rely on simulations.
- **Goal #2: Develop fast single-IDP join algorithms**
  - Each IDP can only access the data stored in its associated bank.
- **Goal #3: Maximize inter-bank/host-PIM data xfer B/W**
  - Scaling the single-IDP join algorithm to multiple IDPs/DIMMs requires fast and efficient inter-bank/DIMM data transfers.

Contents

- Background & Motivation
- **PID-Join: P**rocessing-**In**-**D**IMM **Join** Algorithm
  - **Overview**
  - **Challenges & Key Ideas**
- Evaluation
- Conclusion

Processing-In-DIMM Join (PID-Join)

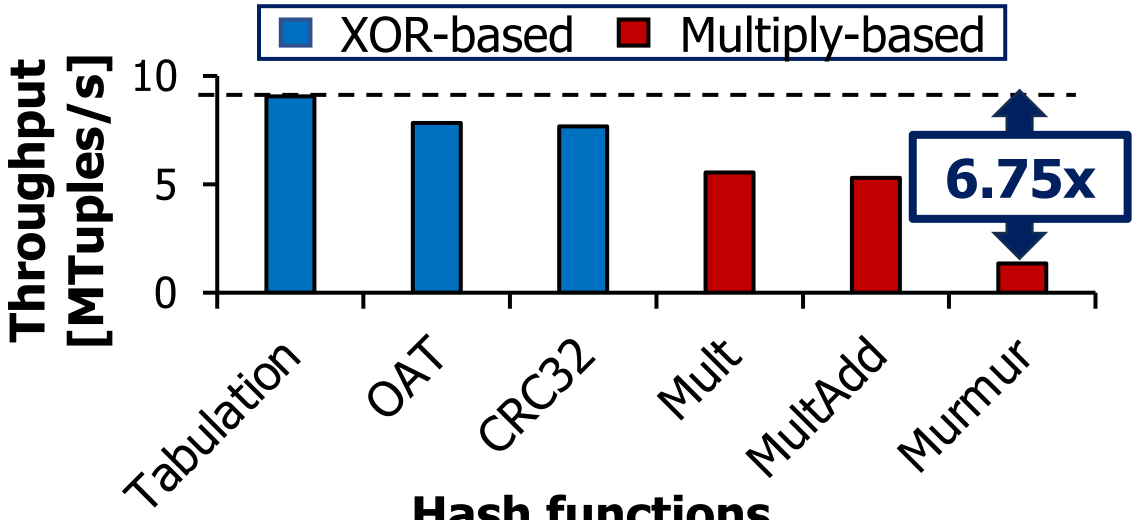
Challenges	Key Ideas
Limited Capabilities of IDPs	Fast Single-IDP Joins
Slow Inter-Bank Communication	Rotate-and-Stream
Slow Host-PIM Data Transfer	Unordered Scatter Gather



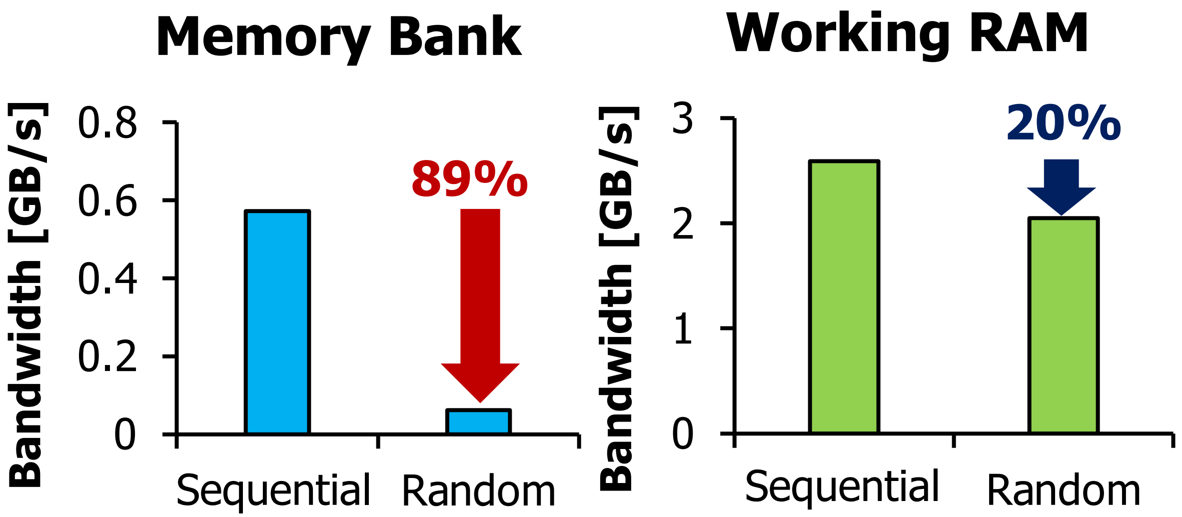
PID-Join's working model

Challenge #1: Limited Capability of IDPs

- **Lack of native hardware support for complex arithmetic**
  - e.g., slow integer multiplication/division, floating-point operation
- **Random bank access leads to significant B/W decrease**
  - 89% lower bandwidth than sequential WRAM-bank access
  - Due to row activation of DRAM banks



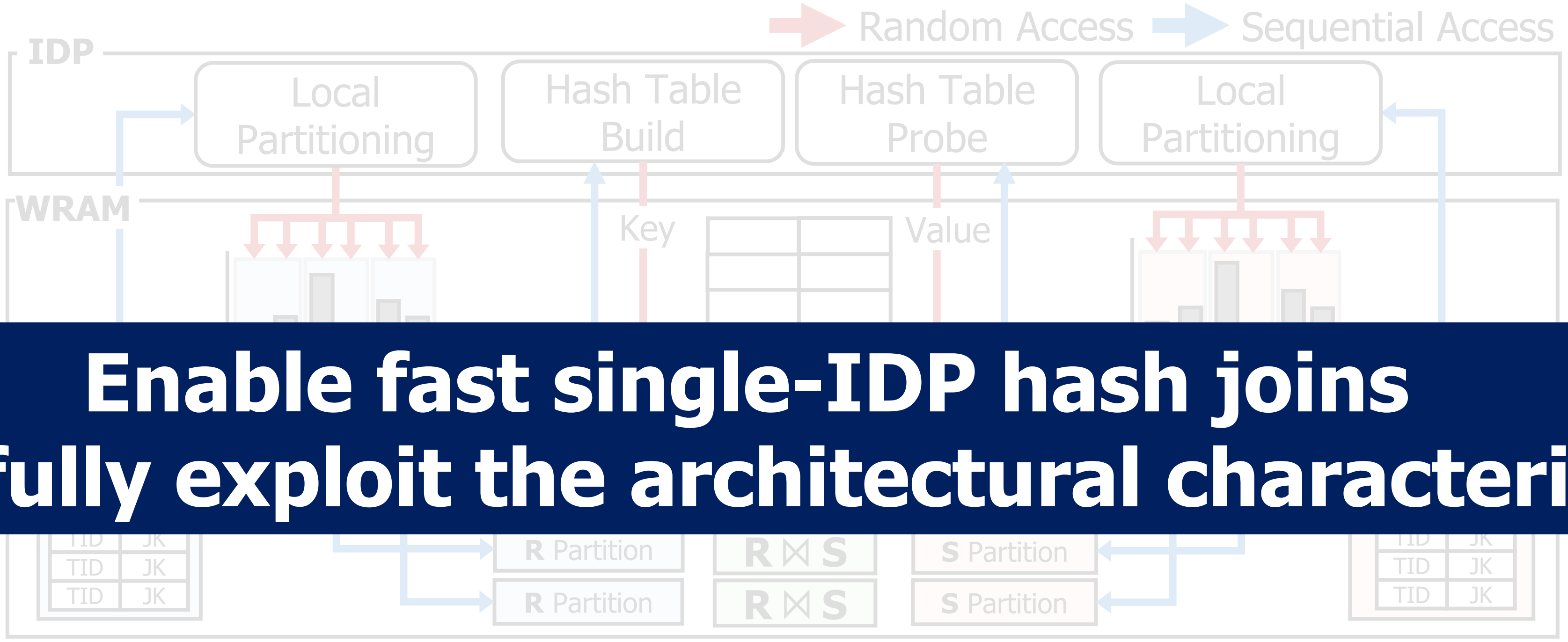
Single-IDP performance of various hash functions



Bandwidth comparison on various access methods

Key Idea #1: Optimized Single-IDP Joins

- Enforce **sequential WRAM-memory bank accesses**
  - Filter random accesses (e.g., hash table lookups) to the WRAM
- Utilize a **fast XOR-based hash function**
  - Maximize the computational throughput of a single IDP

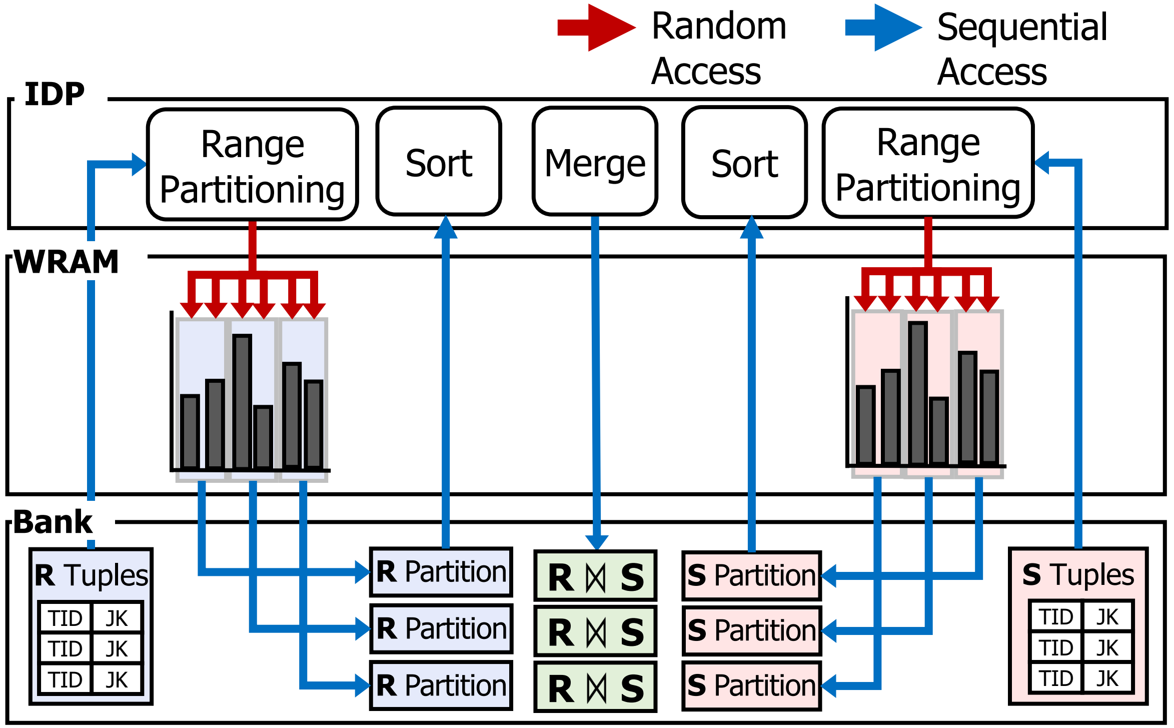


Enable fast single-IDP hash joins which fully exploit the architectural characteristics!

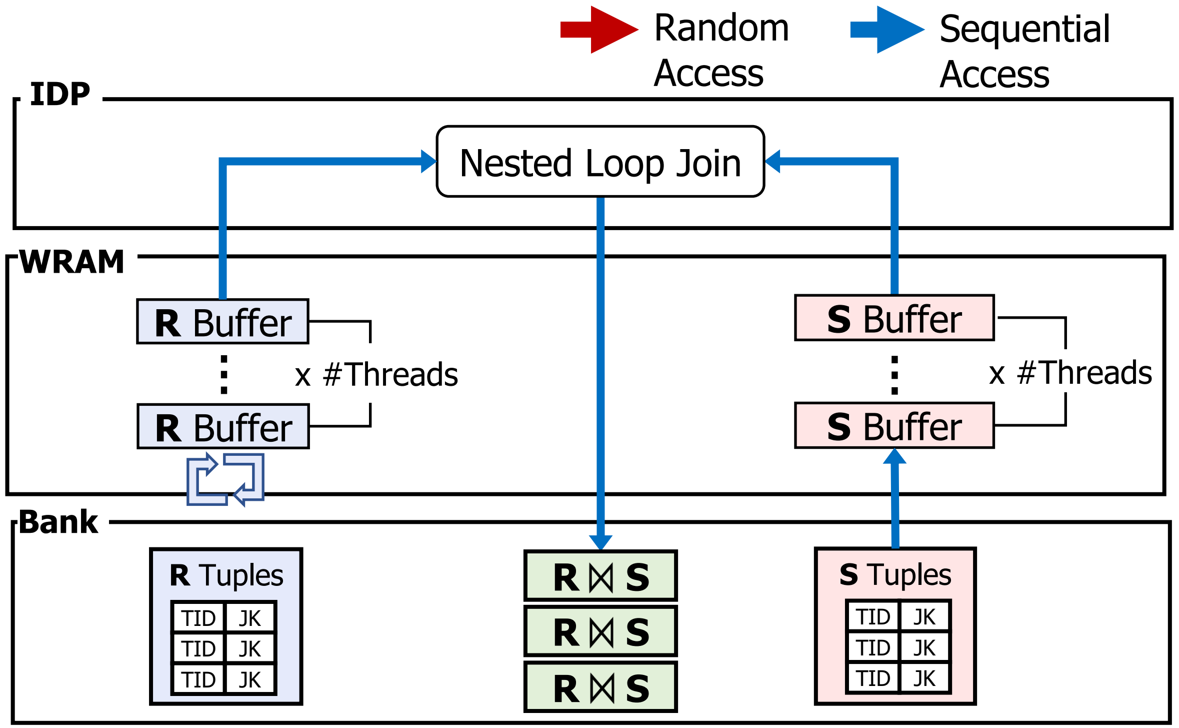
PID-Join's Single-IDP hash join

Optimization of Other Join Algorithms

- **Sort-merge join** exploits range partitioning with the WRAM.
- **Nested-loop join** uses streaming memory access patterns.



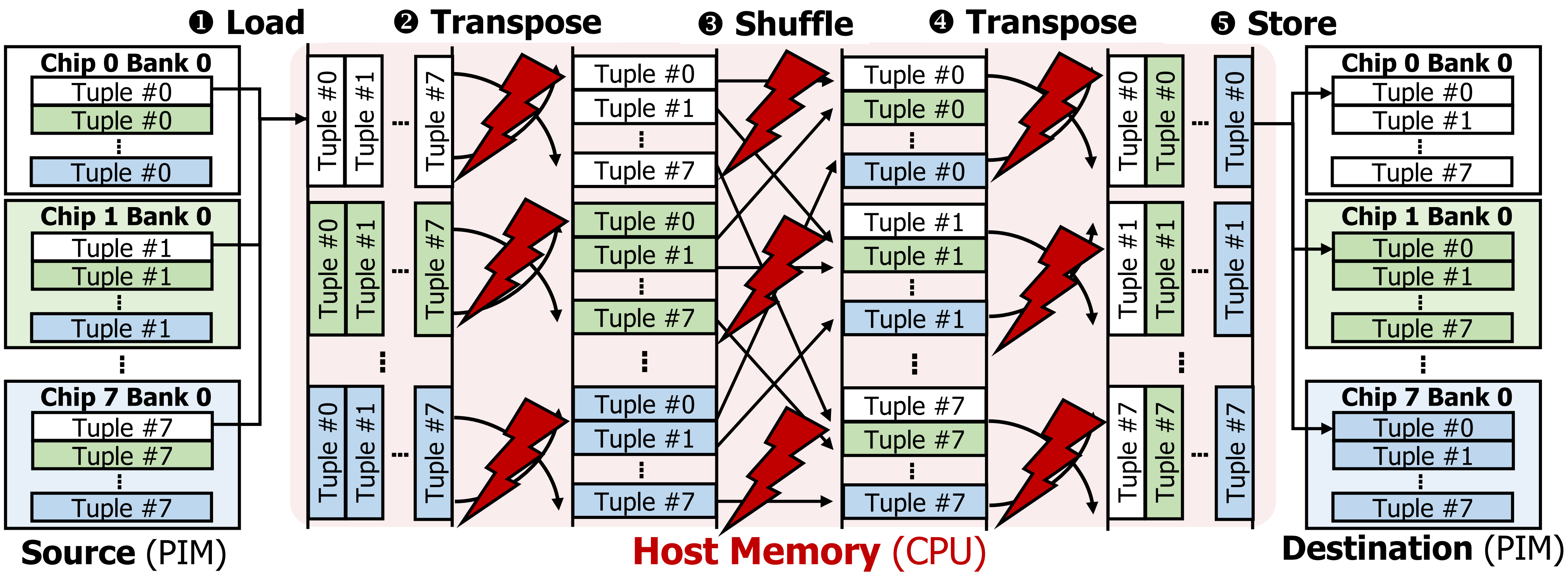
PID-Join's single-IDP sort-merge join



PID-Join's single-IDP nested-loop join

Challenge #2: Slow Inter-Bank Communication

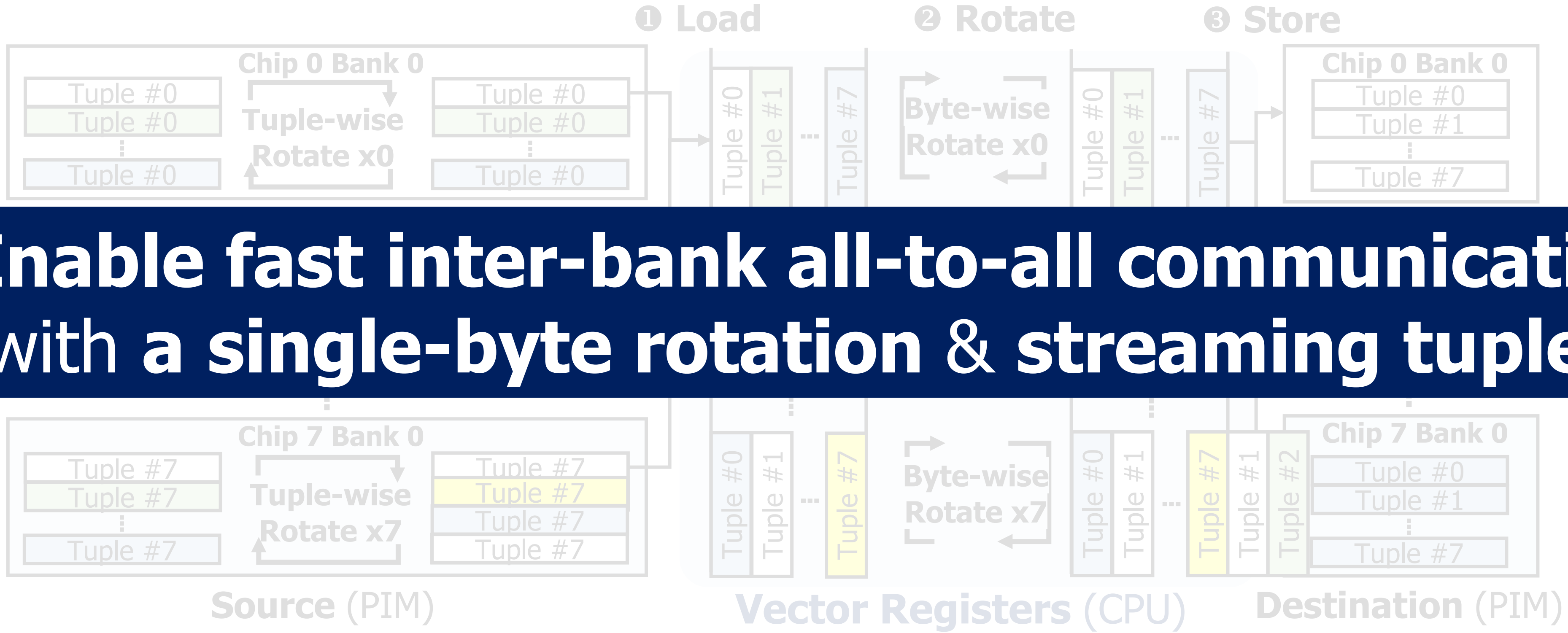
- **Require the CPU to mediate the communication**
  - No direct communication paths between the banks
- **Must transpose per-bank data** using the CPUs
  - Transposing all transferred data incurs high computational overheads.



Working model of inter-bank communication. The colors indicate the tuples' destination banks.

Key Idea #2: Rotate-and-Stream

- Implement an **IDP-CPU cooperative all-to-all shuffle**
  - Minimize the CPU consumption of the data transpose
- Exploit **the identical data layout** of the src. & dst. banks
  - Eliminate adjustment of data layouts & enable streaming tuples

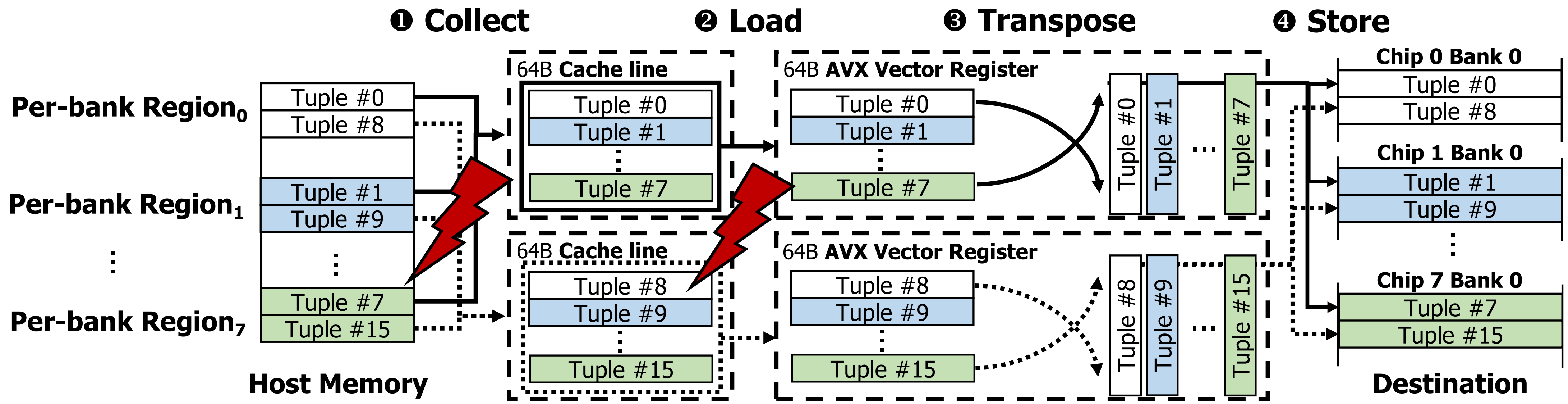


Enable fast inter-bank all-to-all communication with a single-byte rotation & streaming tuples!

The working model of Rotate-and-Stream. The colors indicate the tuples' destination banks.

Challenge #3: Slow Host-PIM Data Transfer

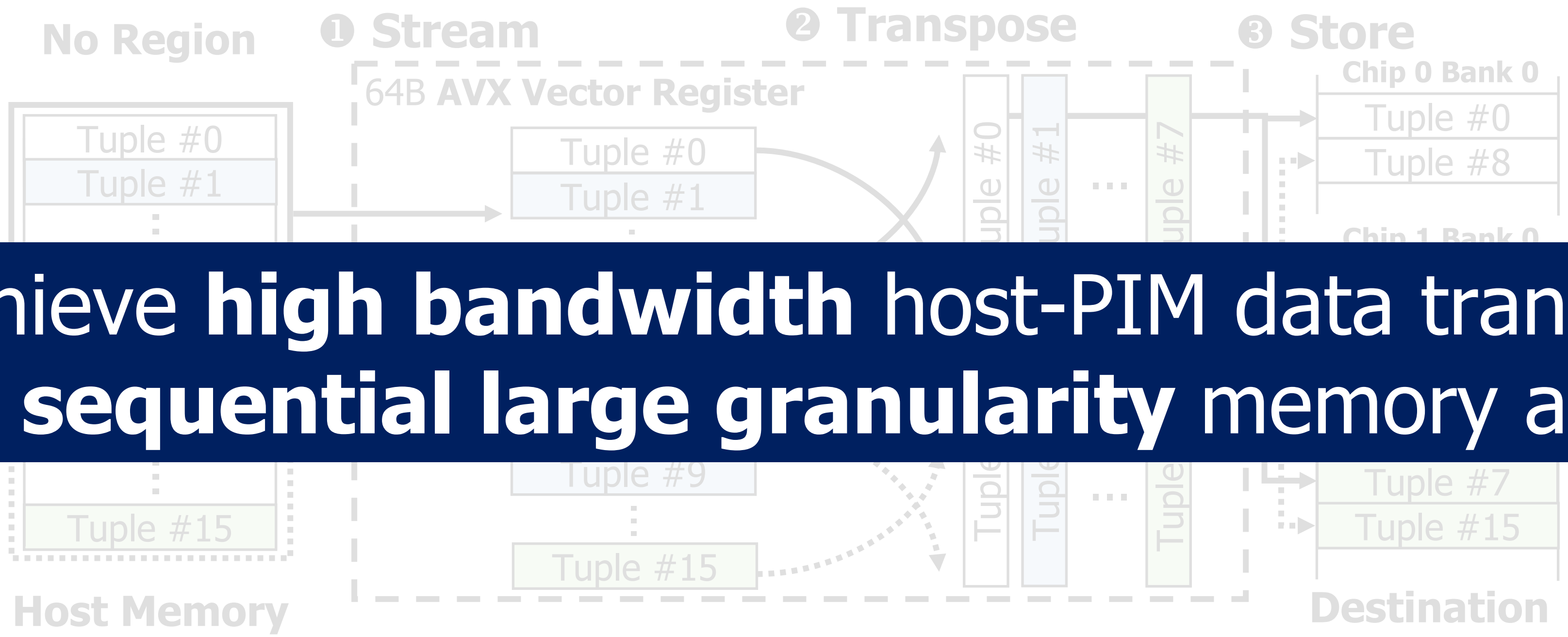
- **Per-bank region splitting** incurs random host mem. Access
  - To identify what data to be stored in each destination bank
  - Collecting the data to construct a cache line
- **Redundant loads** on the CPU side
  - To collect and perform byte-wise transposing



Working model of conventional Host-to-PIM data transfer

Key Idea #3: Unordered Scatter Gather

- Joins **do not impose strict ordering** to input/output tuples.
  - Scatter/gather input/outputs in any order to exploit all IDPs.
- **Rank-wise host-PIM data transfers** rather than bank-wise
  - Remove the need for region splitting and exploits AVX vector registers



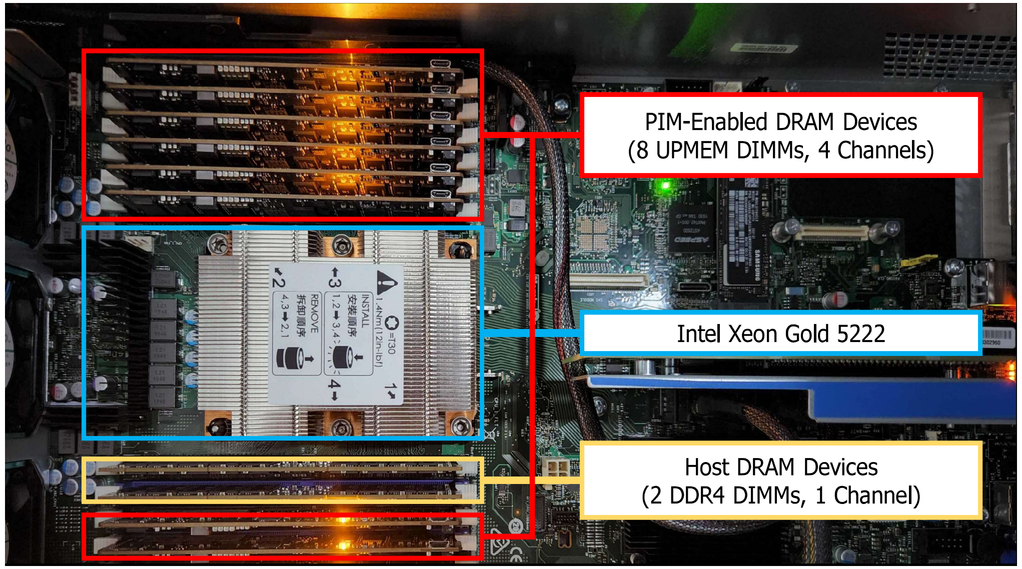
Achieve **high bandwidth** host-PIM data transfer **with sequential large granularity** memory access!

Working model of PID-Join's Unordered Scatter

Contents

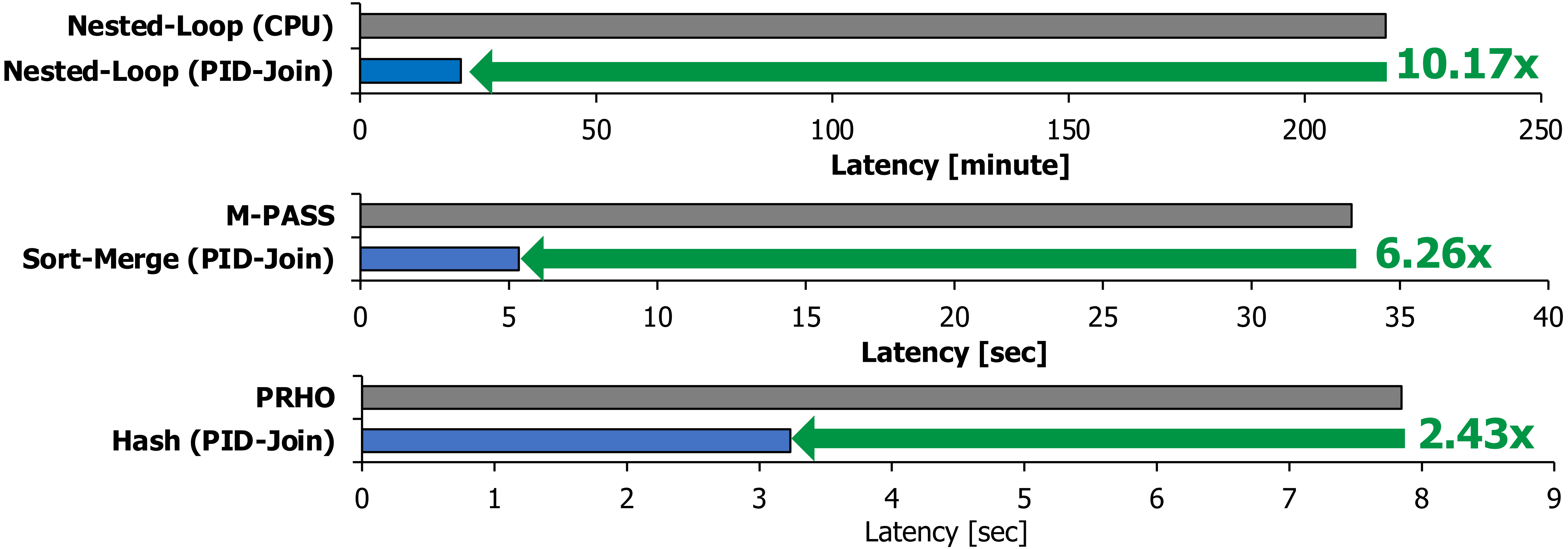
Experimental Setup

- **Baseline & PIM Systems**
  - Intel Xeon Gold 5222 CPU
  - Five DDR4 channels per system
    - Baseline: five standard DDR4 channels
    - PIM: four PIM + one standard channels
- **Workloads**
  - Synthetic benchmarks
    - 32-bit integer join keys & tuple IDs
    - |R|:|S| ratios from 1:1 to 1:32
      - Adjust |S| from 64-M to 512-M tuples
    - Uniform R, uniform/skewed S
  - Four TPC-H queries involving a two-way join



Our PIM system equipped with eight UPMEM DIMMs (a total of 1,024 IDPs)

Fast Join Executions

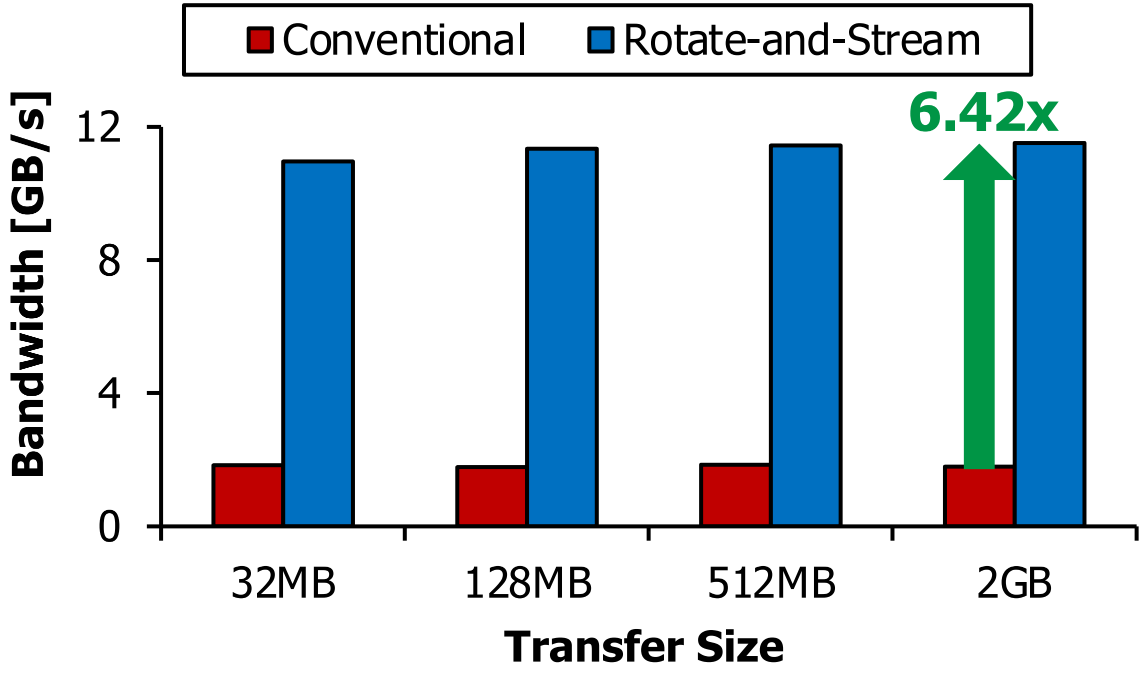


Latency of CPU-based join algorithms and PID-Join with |*R*|:|*S*| ratio of 1:1 with 512-M tuples of |S|

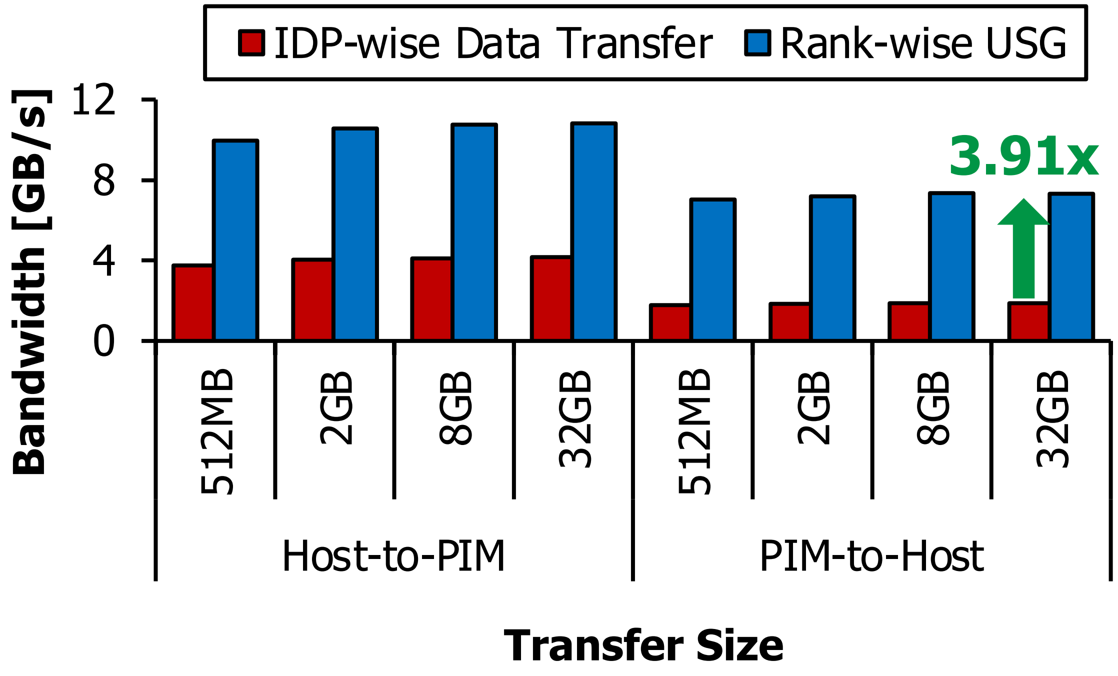
- **Up to 2.43x faster join latency** compared to PRHO*
  - 1.92x (hash), 4.79x (sort-merge), 83.22x (nested-loop) geomean improvements

* Histogram-based Optimized Parallel Radix Join (PRHO) [VLDB '13]

Fast Inter-Bank Communication & Host-PIM Data Transfers



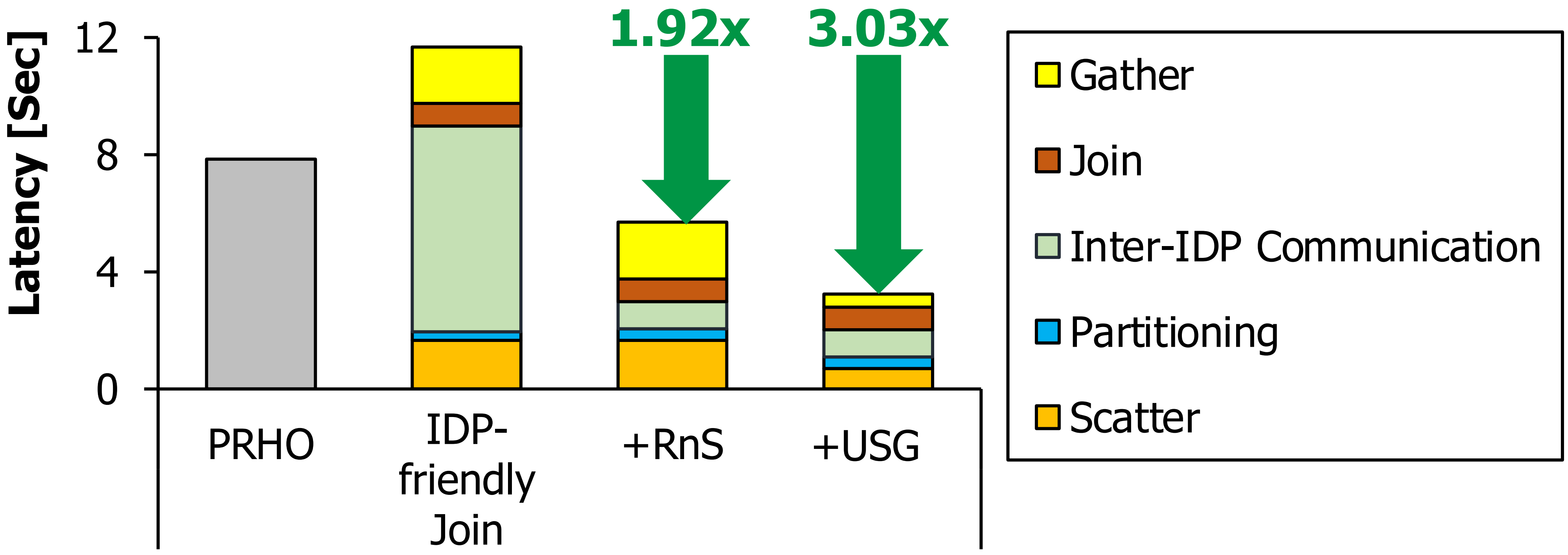
All-to-all communication bandwidth comparison



Host-PIM data transfer bandwidth comparison

- **Can be achieved** by **enabling streaming transposes**
  - **with large access granularity using vector registers**.

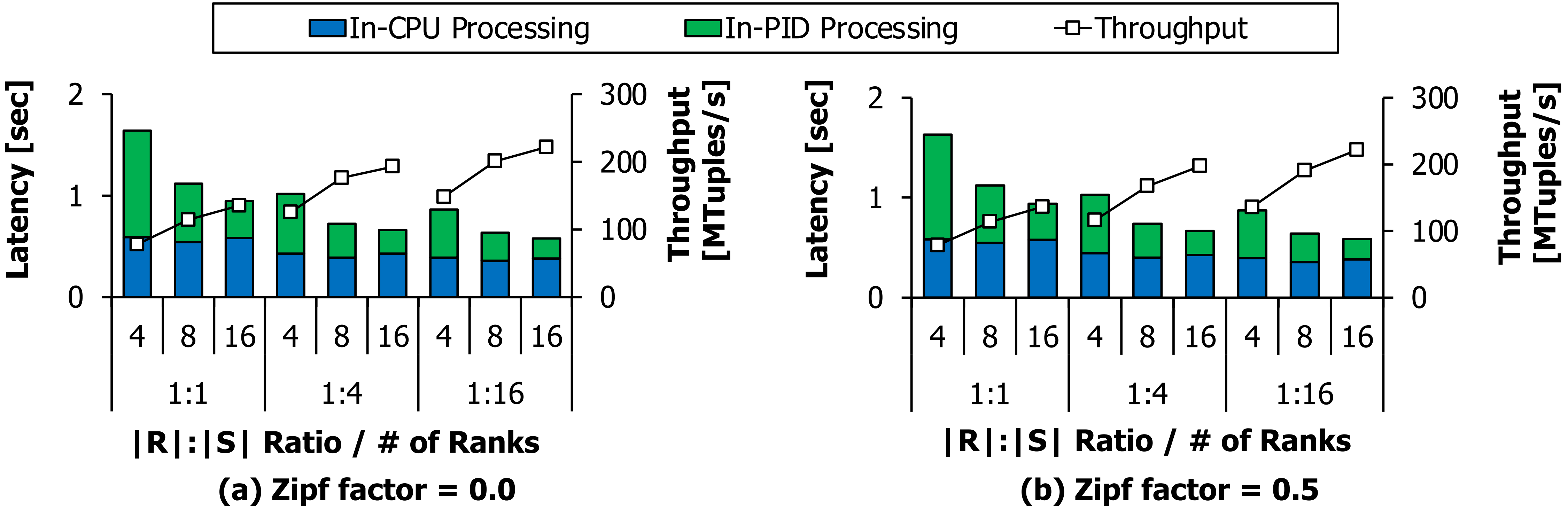
Effect of Data Transfer Optimizations



Join execution latency of PRHO and PID-Join with |R|:|S|=1:1 and |S|=512-M tuples

- **Achieved 1.92x speedup** with rotate-and-stream
- **Achieved 3.03x speedup** RnS + unordered scatter/gather!

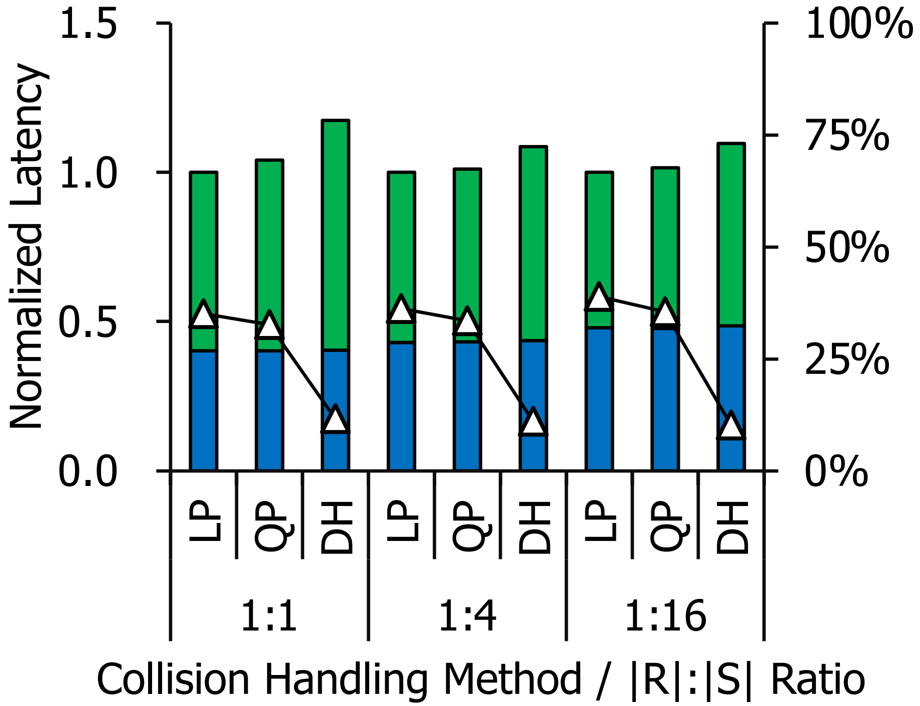
Varying Rank Counts

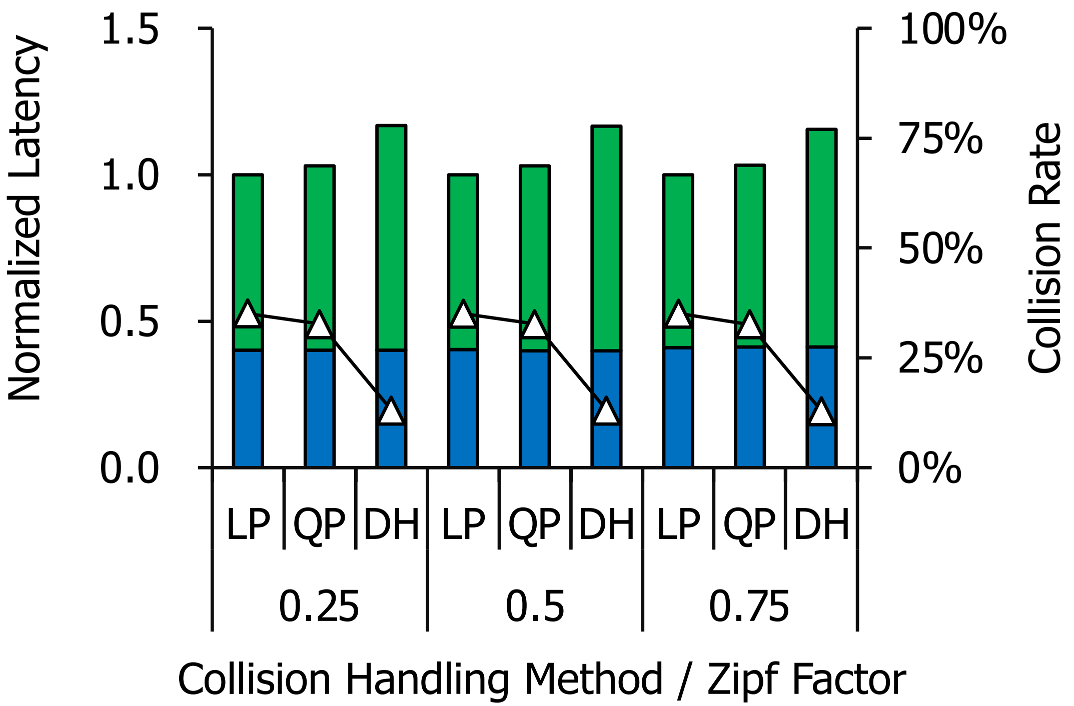


Join execution latency of PID-Join with varying rank counts and |R|:|S| ratios

- **Easily scales-out** with varying rank counts.
  - The number of IDPs increases accordingly
  - Some memory padding incurs little latency increase

Varying Collision Handling Methods

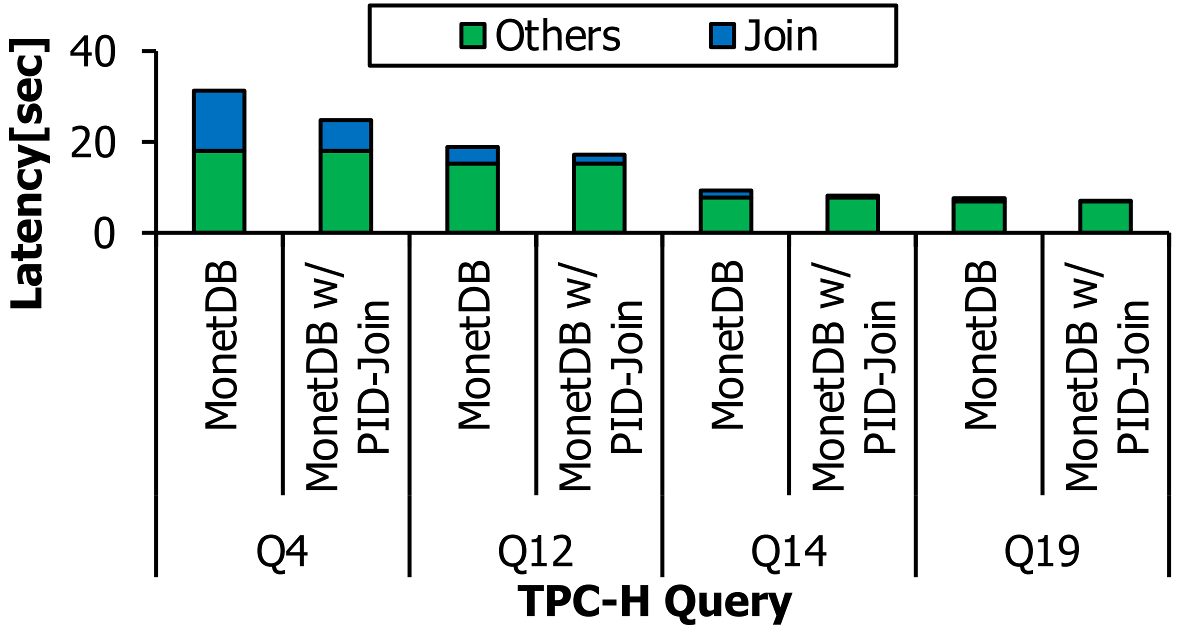




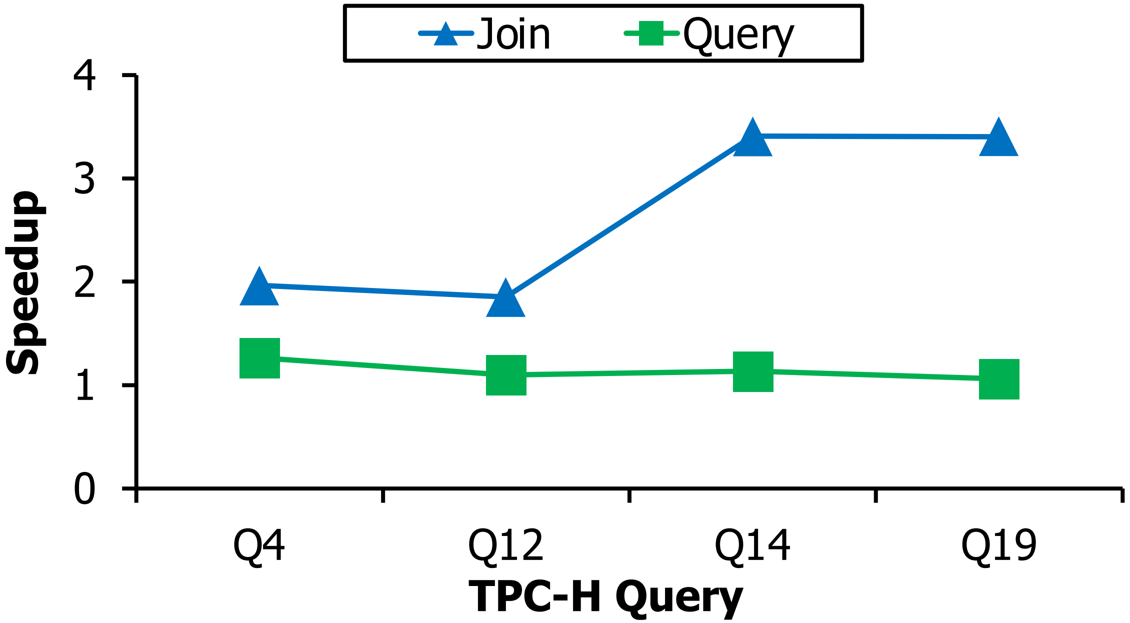
Single-IDP performance of PID-Join with varying collision handling methods

- **Linear probe** shows the **highest hash performance**.
  - Due to its low computational overhead.

Potential Speedup with TPC-H Queries



Breakdown of the TPC-H query execution latency of MonetDB



The potential speedup of PID-Join

- **Up to 3.41x faster** two-way join for TPC-H queries
  - Geomean speedup of 2.55x with joins of the queries
- **Up to 1.26x faster** TPC-H queries
  - Geomean estimated speedup of 1.14x with four two-way join queries.

Conclusion

- **PIM is highly promising** for accelerating in-memory joins.
  - Achieve significantly higher memory bandwidth over the CPU
- **First study** which accelerates join on PIM-enabled DIMMs
  - The existing PIM-assisted joins focus on 3D-stacked PIM devices.
  - The existing PIM-assisted joins rely on cycle-level simulations.
- **PID-Join**, a fast PIM-assisted in-memory join algorithm
  - First work that implements PIM-assisted join on the real system
  - Optimized single-IDP joins, Rotate-and-Stream, and Unordered Scatter-Gather
  - **Achieve geometric-mean speedup of 1.92x vs. PRHO**

Thank You!

- Any questions?
  - Please refer to our paper for more details & experimental results!
  - **`https://github.com/yonsei-hpcp/pid-join`**

Youngsok Kim

Assistant Professor
Department of Computer Science
College of Computing
Yonsei University

Email: `youngsok@yonsei.ac.kr`
Website: `https://hpcp.yonsei.ac.kr/~youngsok`