









PIMANN: Turbocharge ANNS on Real Processing-in-Memory by Enabling Fine-Grained Per-PIM-Core Scheduling

Puqing Wu, Minhui Xie, Enrui Zhao, Dafang Zhang, Jing Wang,

Xiao Liang, Kai Ren, Yunpeng Chai

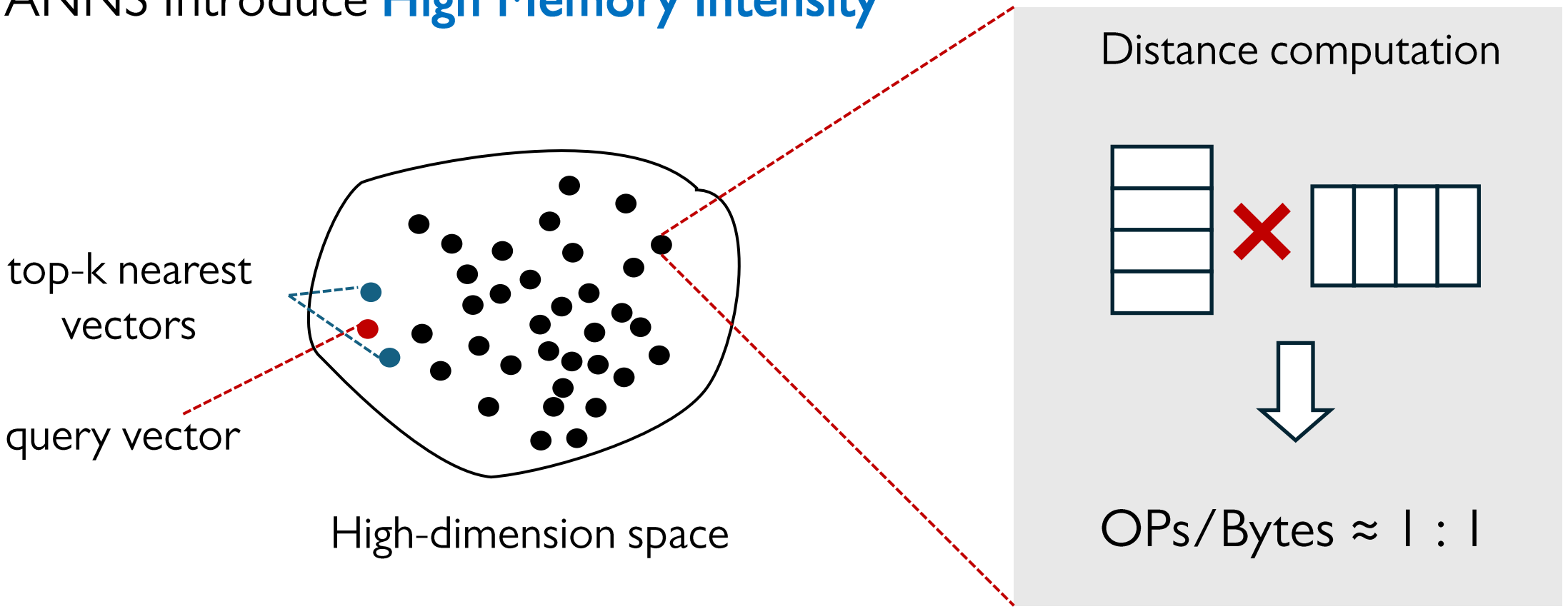
Renmin University of China

Kuaishou Technology

USENIX ATC25

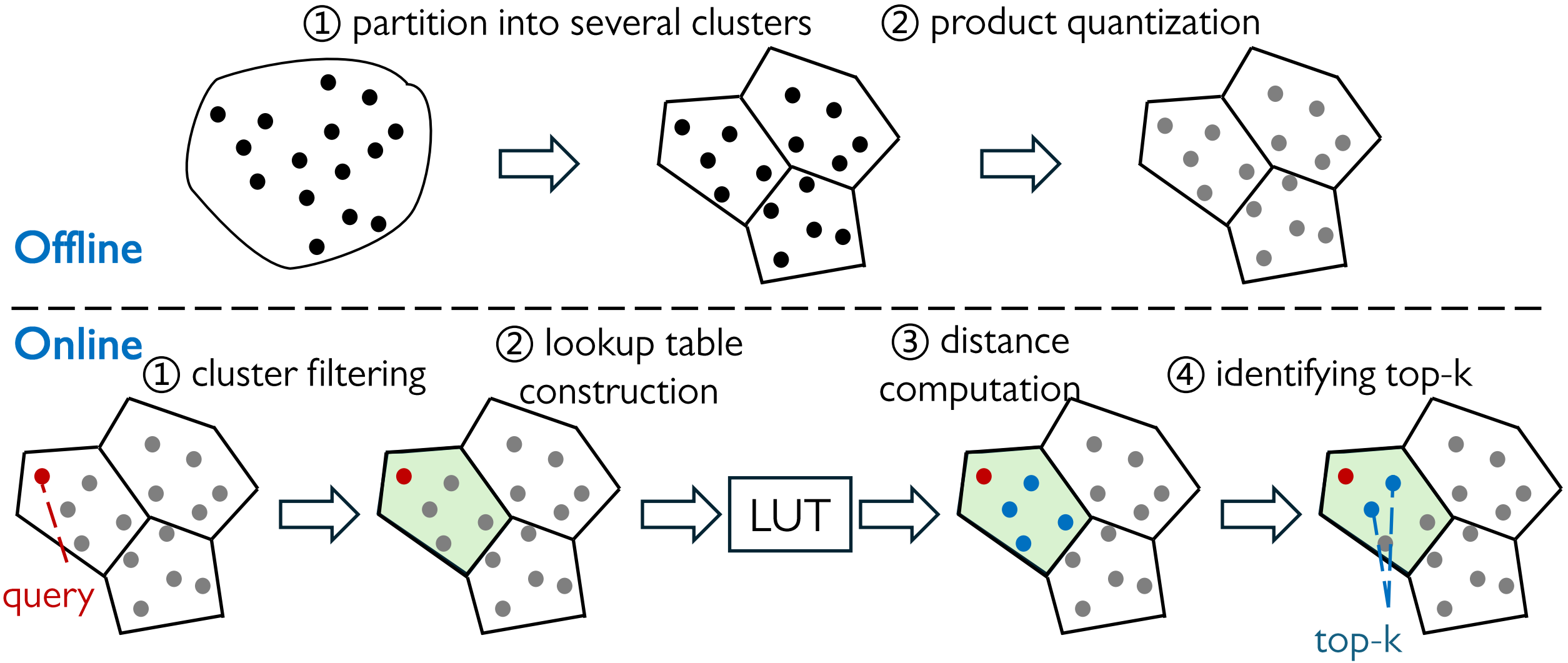
Background: Approximate Nearest Neighbor Search

- Approximate Nearest Neighbor Search (ANNS) aims to return top-k nearest vectors to the query vector
- ANNS Introduce **High Memory Intensity**



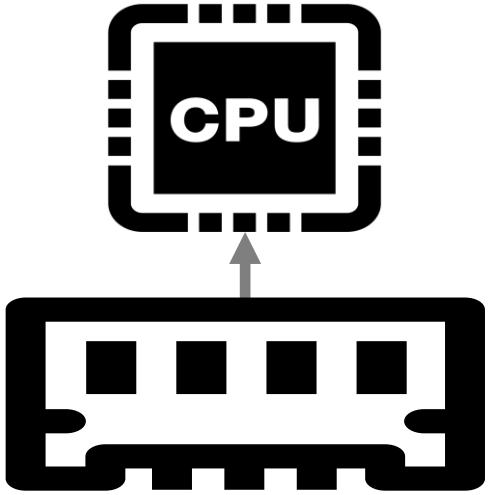
Background: A classic ANNS algorithm (IVFPQ)

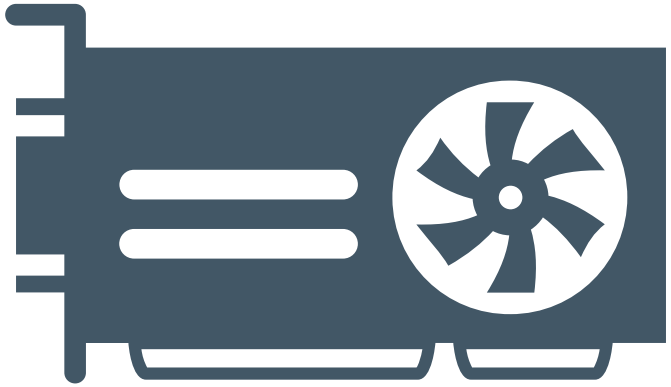
- **IVFPQ**: Inverted File with Product Quantization



Background: Existing ANNS systems

- **CPU-based:** Constrained by memory wall and limited computational throughput (e.g. Faiss-CPU, DiskANN@NIPS19, SPANN@NIPS21)
- **GPU-based**: Constrained by memory wall and limited memory capacity (e.g. Faiss-GPU, Rummy@NSDI24, FusionANNS@FAST25)





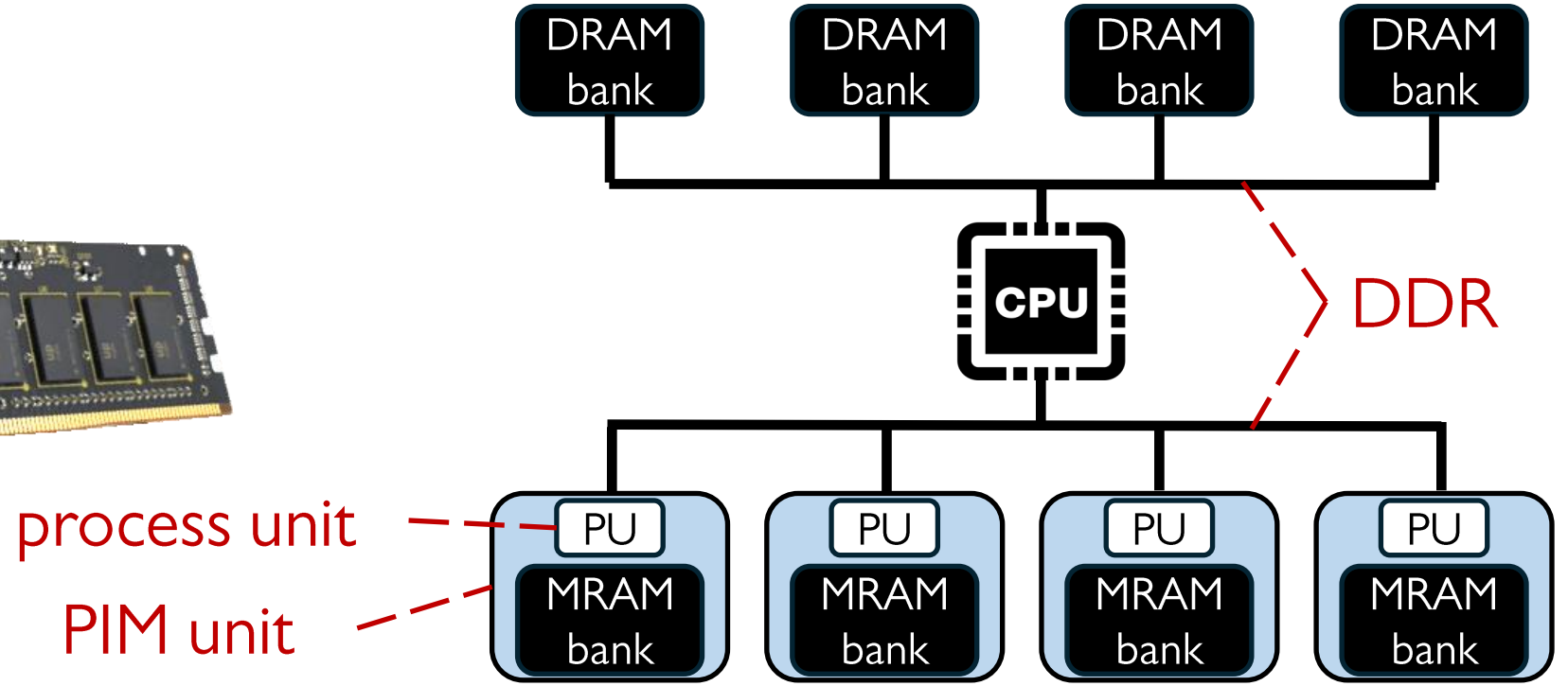
Opportunity: Processing in memory (PIM)

- UPMEM: First commercial PIM hardware

Physical photograph

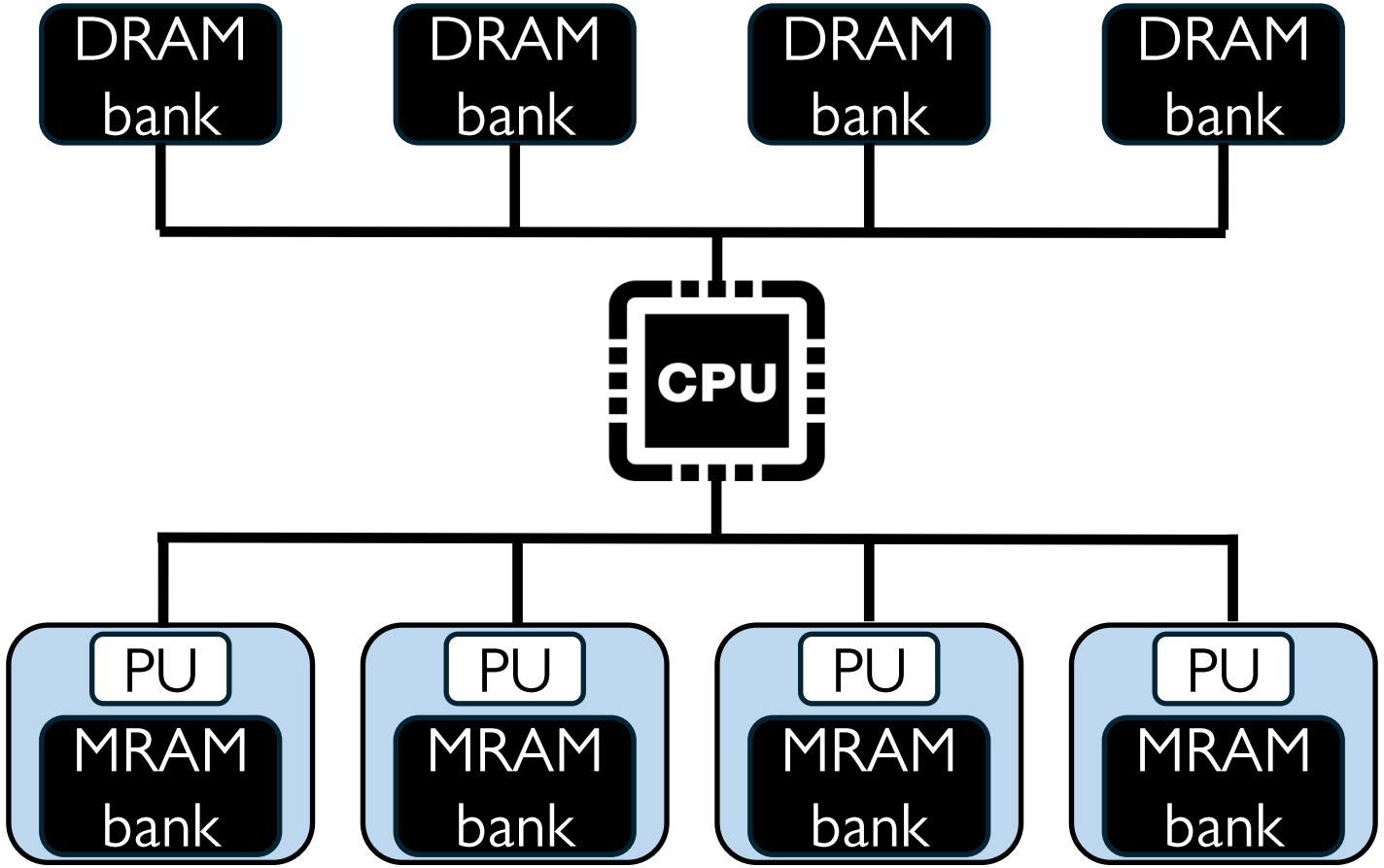


Conceptual diagram



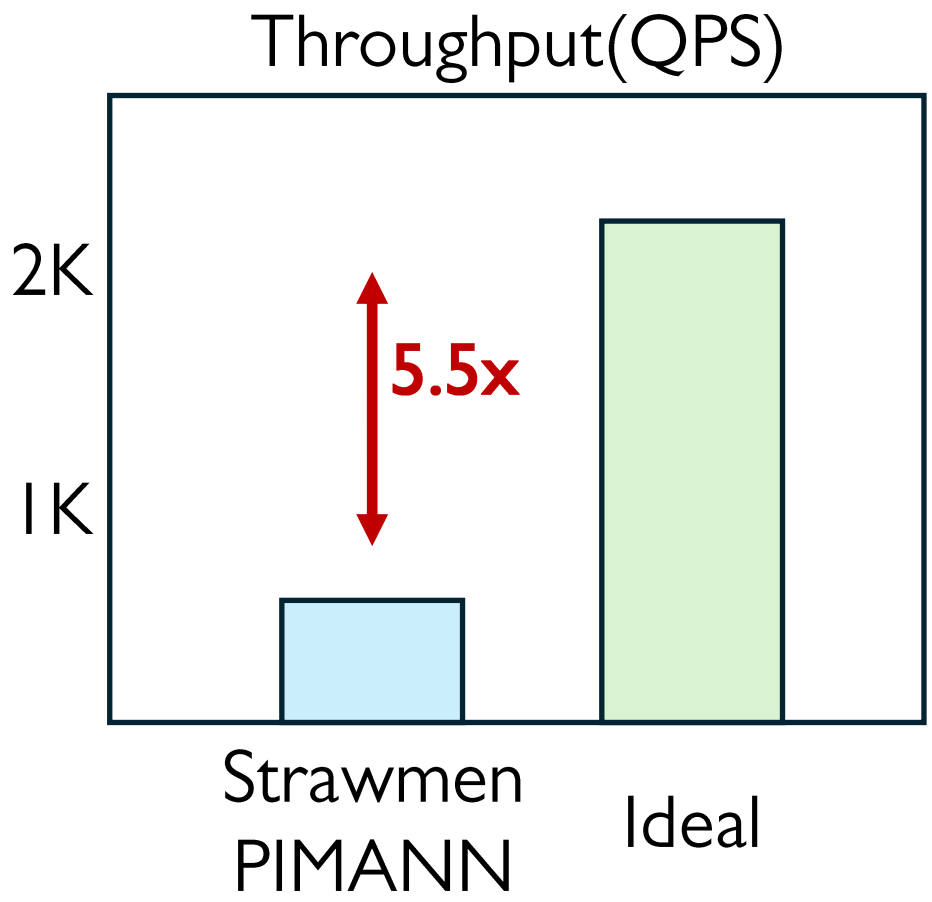
Opportunity: UPMEM

- **High bandwidth**: aggregate bandwidth is 2.56TB/s
- **Highly parallel computing capability**: 2560 PUs, over 40,000 threads

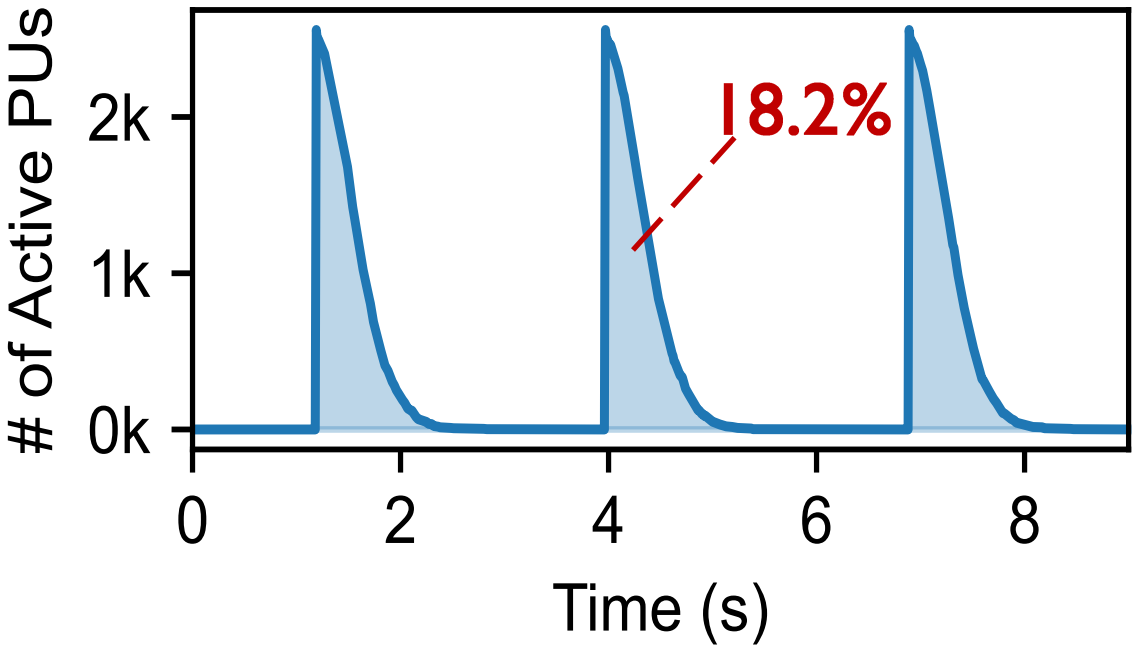


Strawmen PIMANN system: Suboptimal performance

5.5x gap to the ideal performance

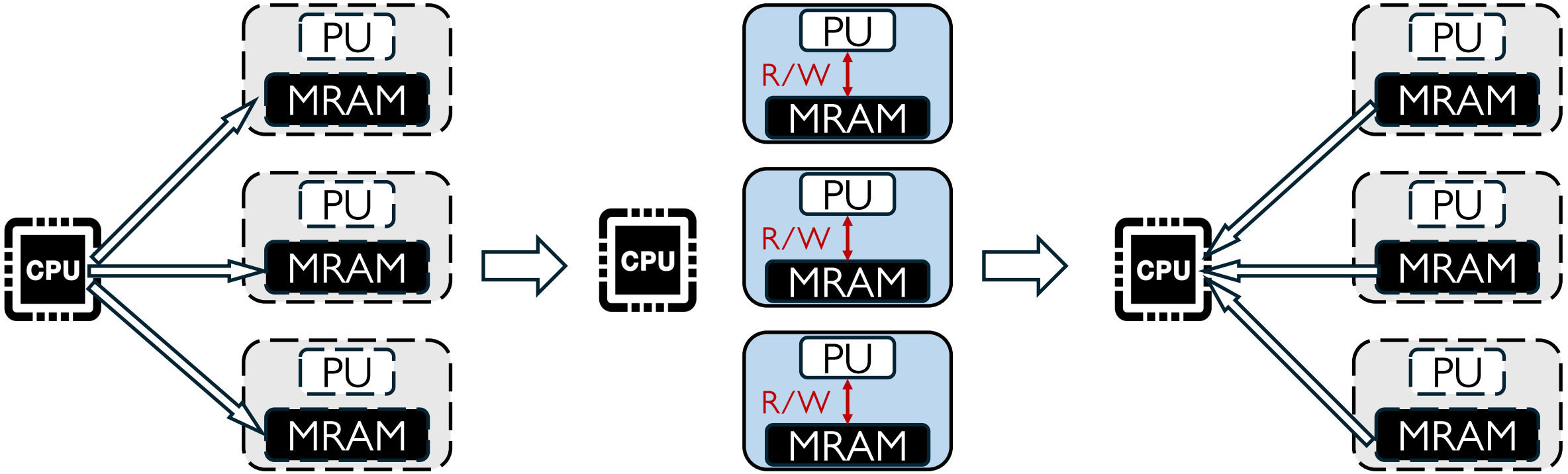


Only 18.2% of PUs are active



Root Cause: Batch scheduling

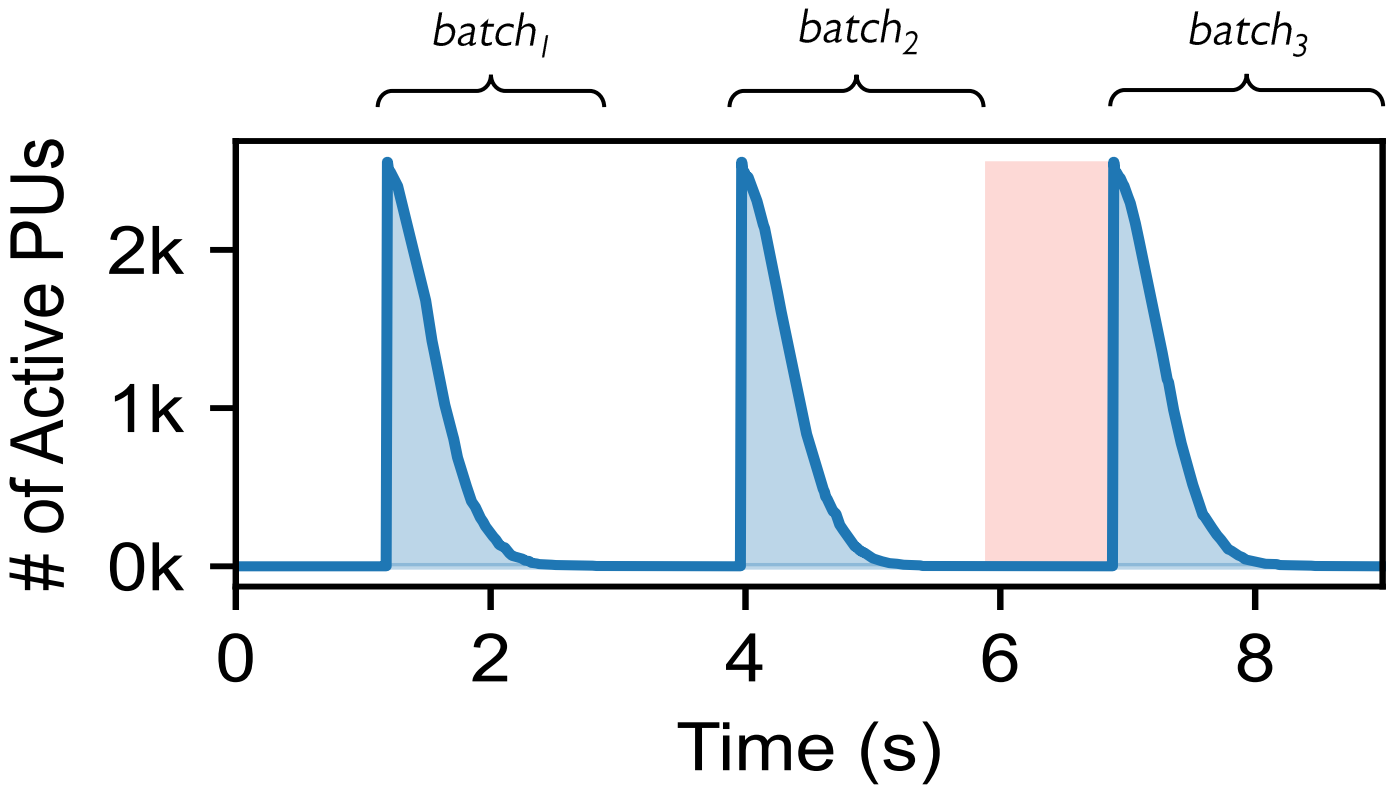
① Send task data ② Launch a PIM kernel ③ Copy back results



Analysis: Why batch scheduling underutilizes PUs

Inter-batch under-utilization:

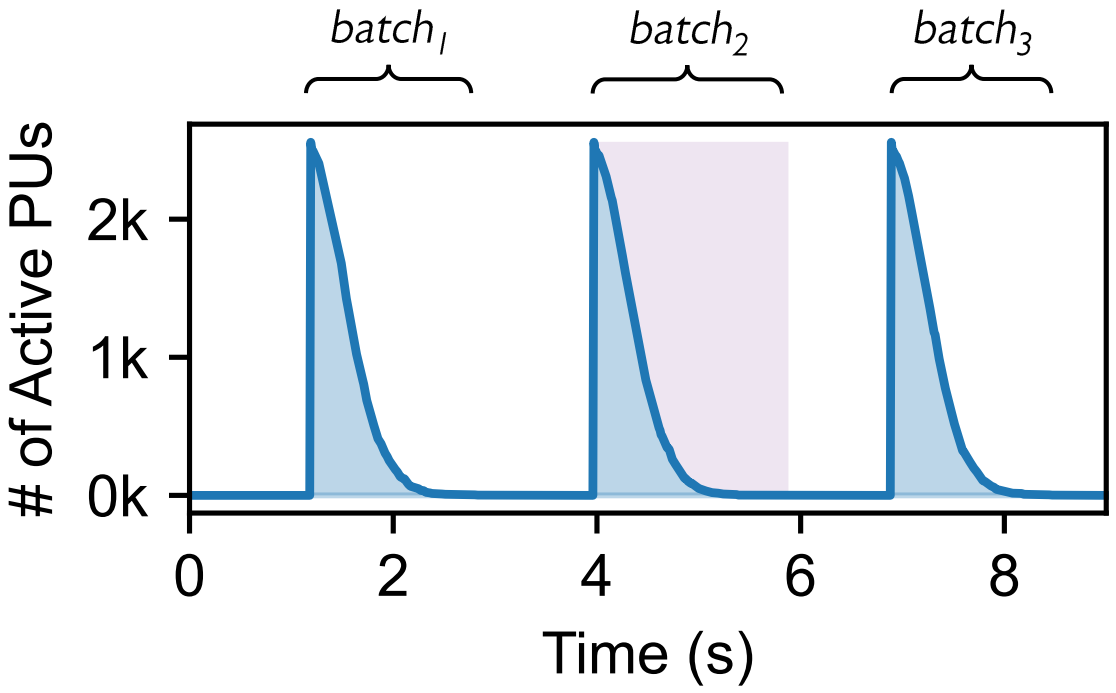
Between two batch, the CPU is copying data, all PUs are idle

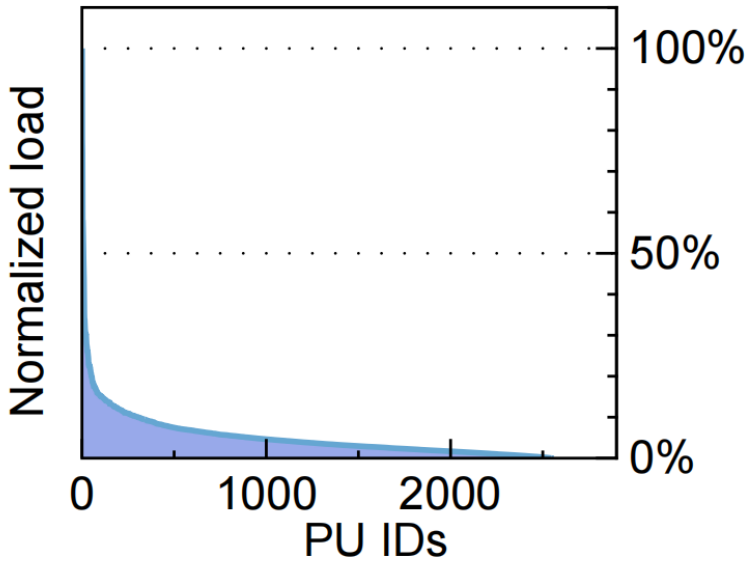


Analysis: Why batch scheduling underutilizes PUs

Intra-batch under-utilization:

Highly-loaded PUs block the entire PIM kernel

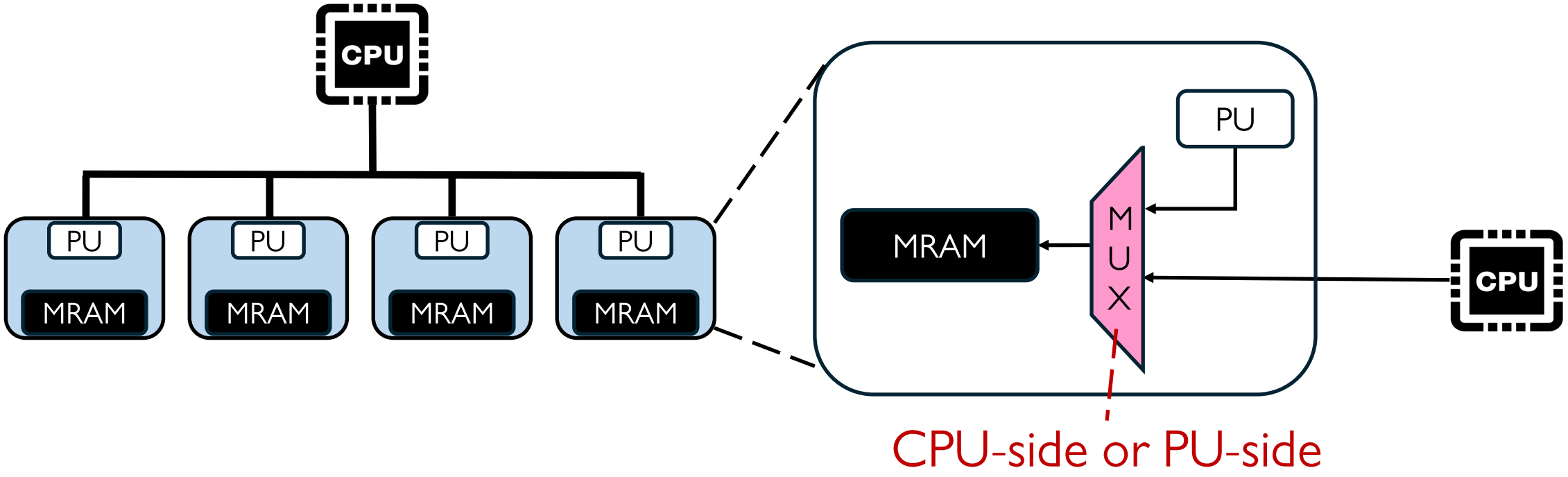




Load of each PU

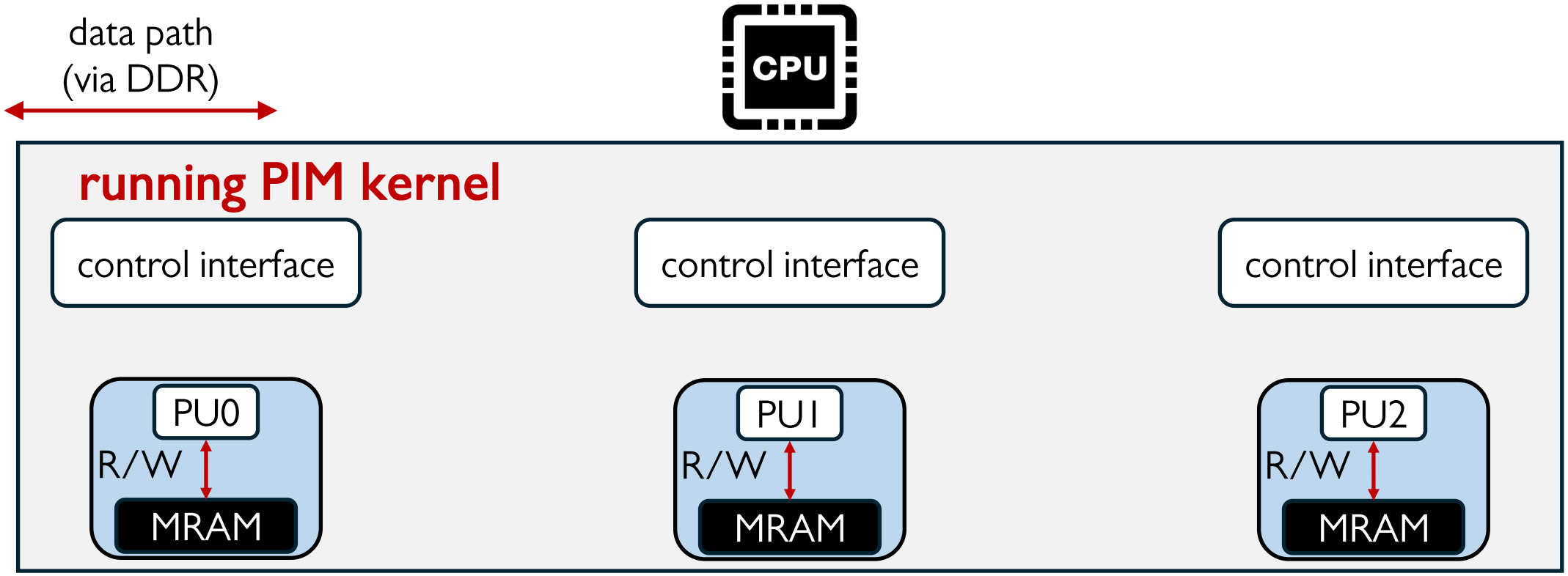
Analysis: Why UPMEM is limited by batch scheduling

- The MRAM bus is shared between CPU and PU, only one processor (CPU or PU) can use the MRAM at any time
- We must prevent CPU and PU from accessing MRAM simultaneously (batch scheduling can do this)



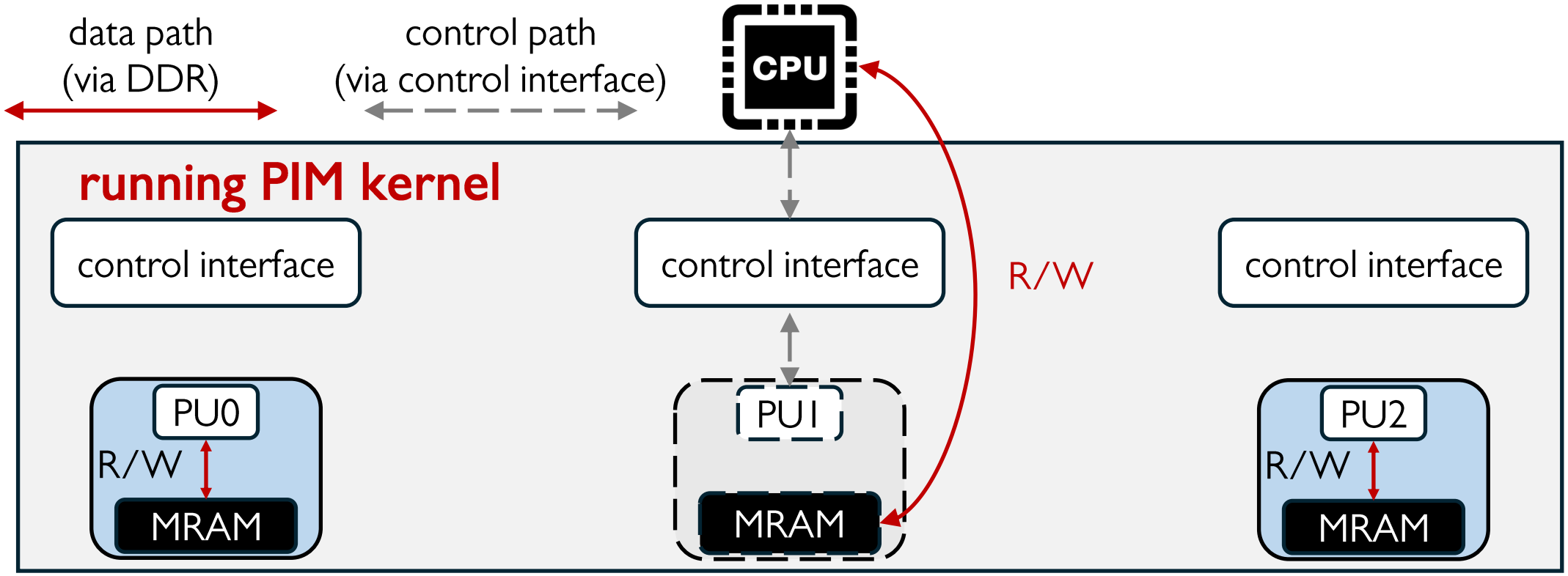
Key idea: Per-PU scheduling with fine-grained arbitration

- **Observation**: Each PU has an independent control interface
- **Key Idea**: Via the control interface, arbitrate MRAM ownership for each PU



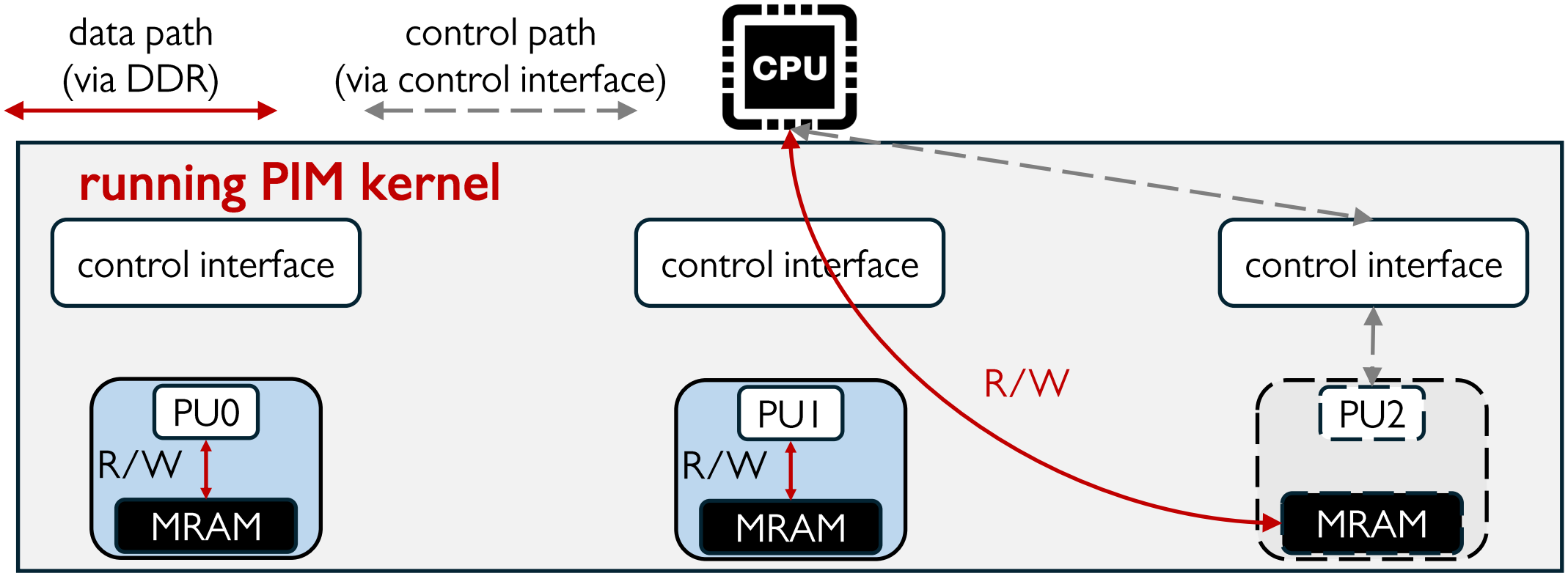
Key idea: Per-PU scheduling with fine-grained arbitration

- **Observation**: Each PU has an independent control interface
- **Key Idea**: Via the control interface, arbitrate MRAM ownership for each PU



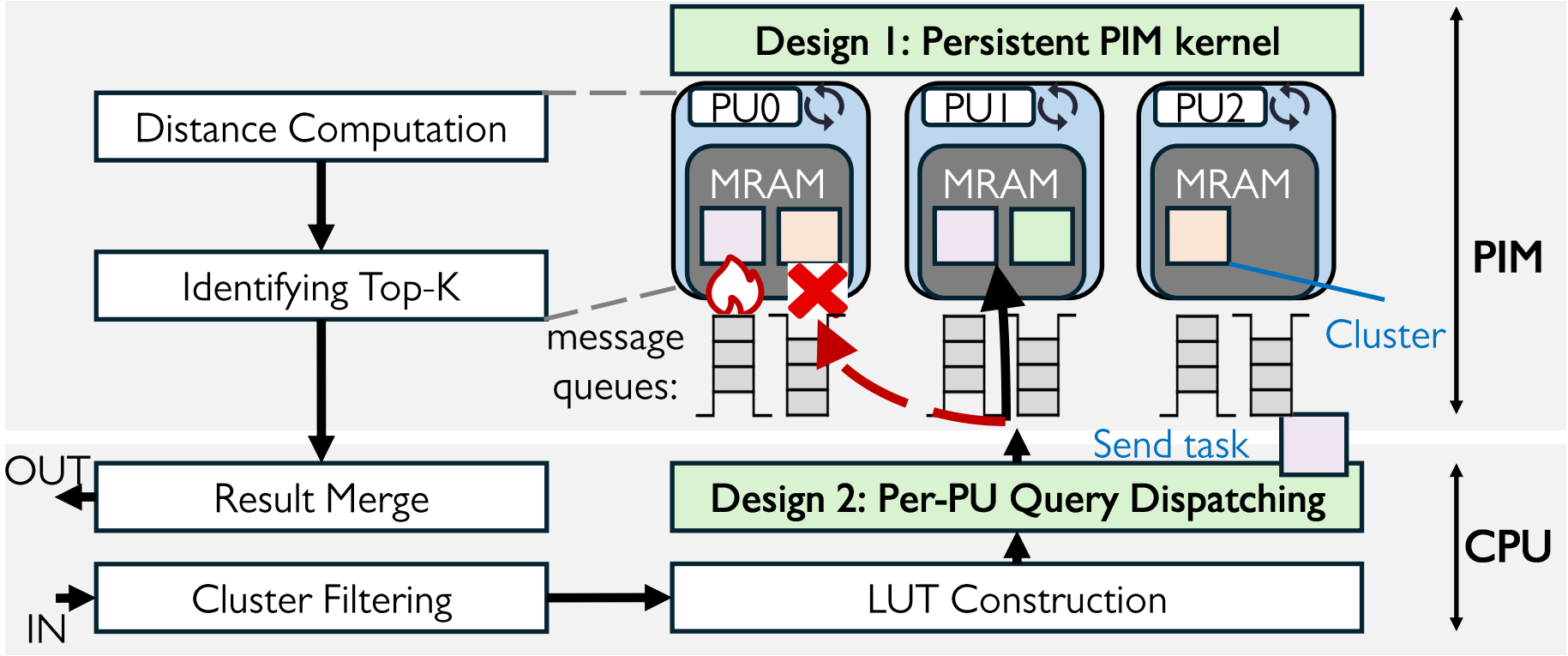
Key idea: Per-PU scheduling with fine-grained arbitration

- **Observation**: Each PU has an independent control interface
- **Key Idea**: Via the control interface, arbitrate MRAM ownership for each PU



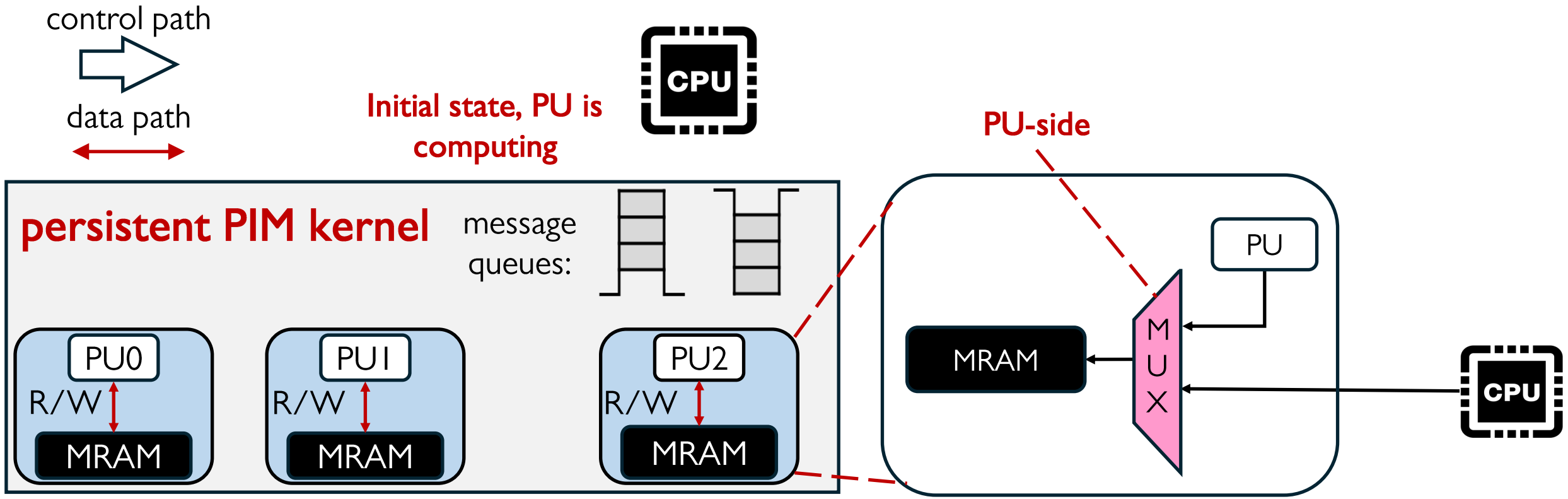
PIMANN Architecture

- **Persistent PIM kernel**: Each PU can continuously and independently receive tasks
- **Per-PU query dispatching**: Send tasks based on PU's real-time load



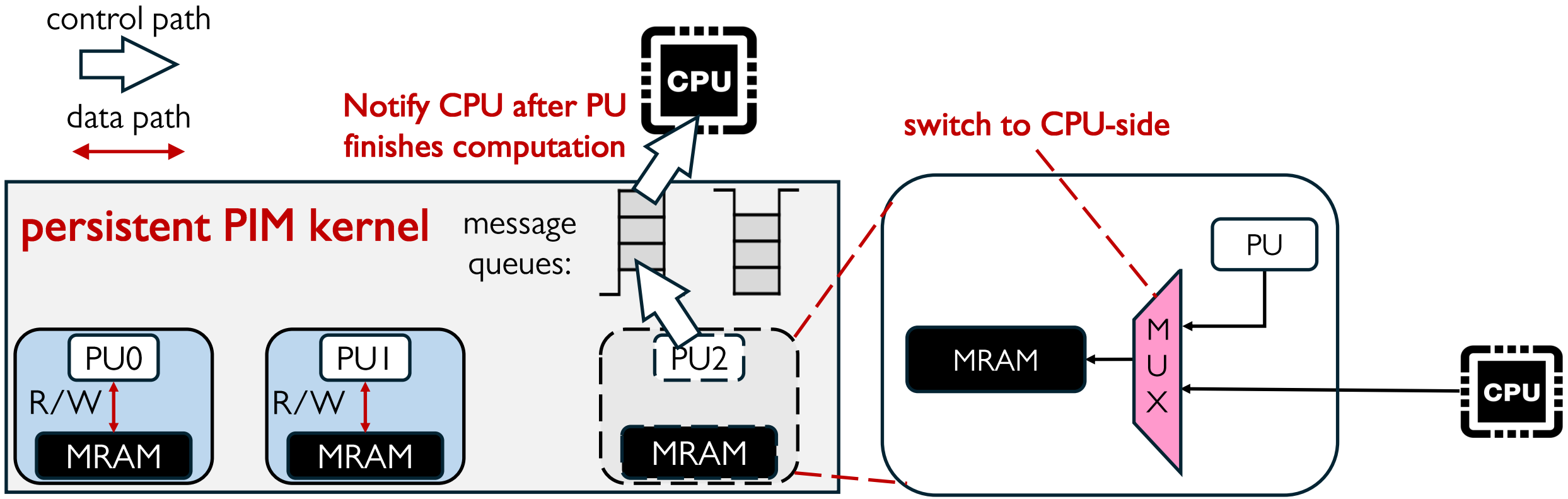
Design 1: Persistent PIM kernel

- Transfer data between CPU and PU when PIM is running
  - **Control path**: Ensure that CPU and PU will not access MRAM simultaneously
  - **Data path**: Modify UPMEM driver, expand data path for running PIM kernel



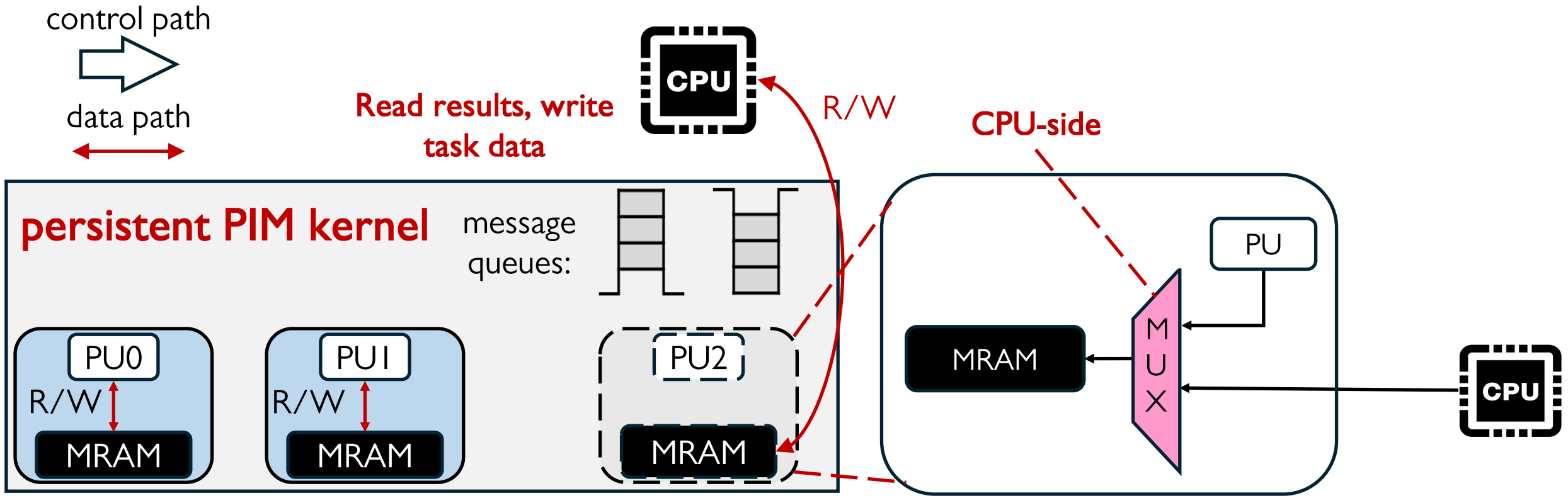
Design 1: Persistent PIM kernel

- Transfer data between CPU and PU when PIM is running
  - **Control path**: Ensure that CPU and PU will not access MRAM simultaneously
  - **Data path**: Modify UPMEM driver, expand data path for running PIM kernel



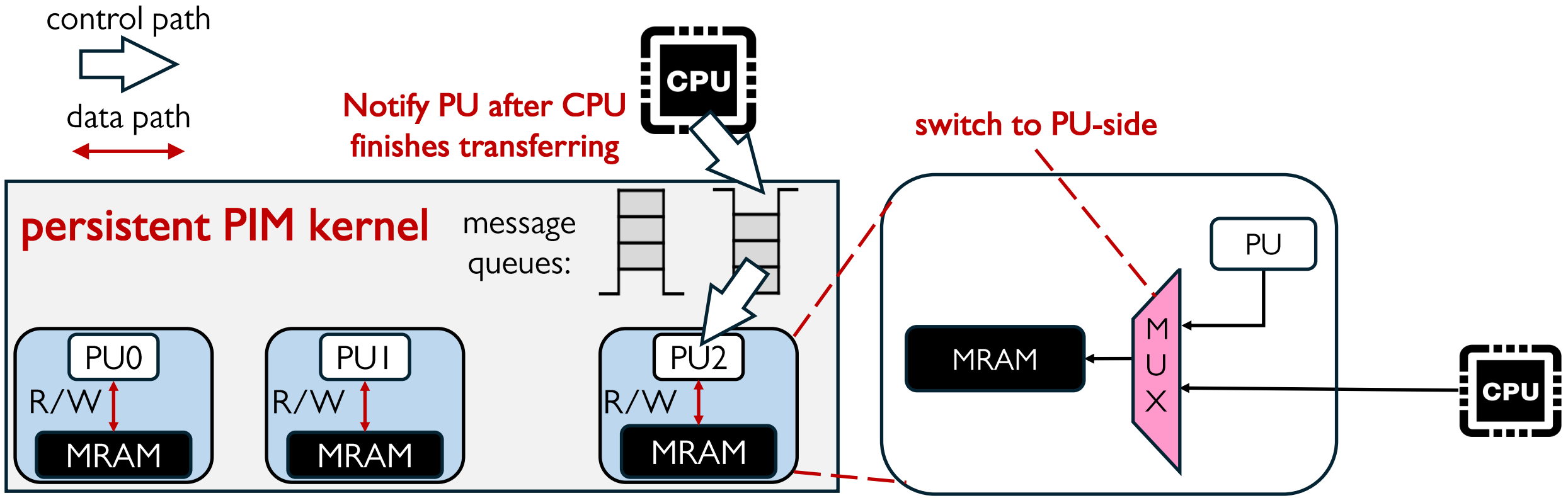
Design 1: Persistent PIM kernel

- Transfer data between CPU and PU when PIM is running
  - **Control path**: Ensure that CPU and PU will not access MRAM simultaneously
  - **Data path**: Modify UPMEM driver, expand data path for running PIM kernel



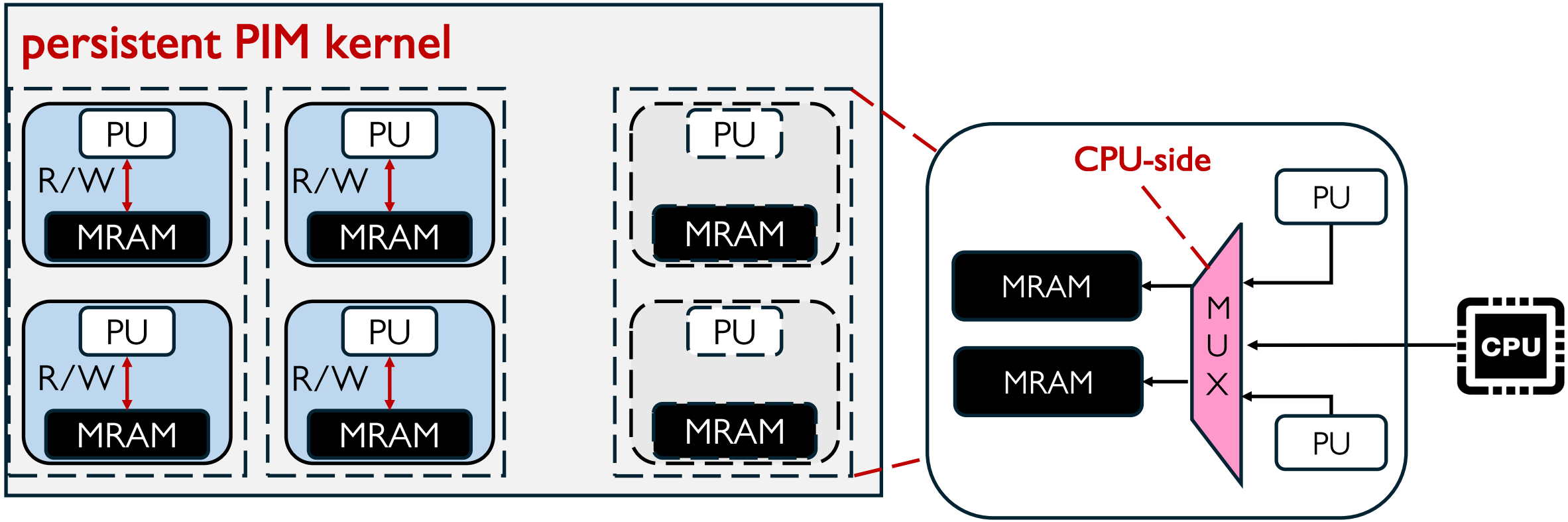
Design 1: Persistent PIM kernel

- Transfer data between CPU and PU when PIM is running
  - **Control path**: Ensure that CPU and PU will not access MRAM simultaneously
  - **Data path**: Modify UPMEM driver, expand data path for running PIM kernel



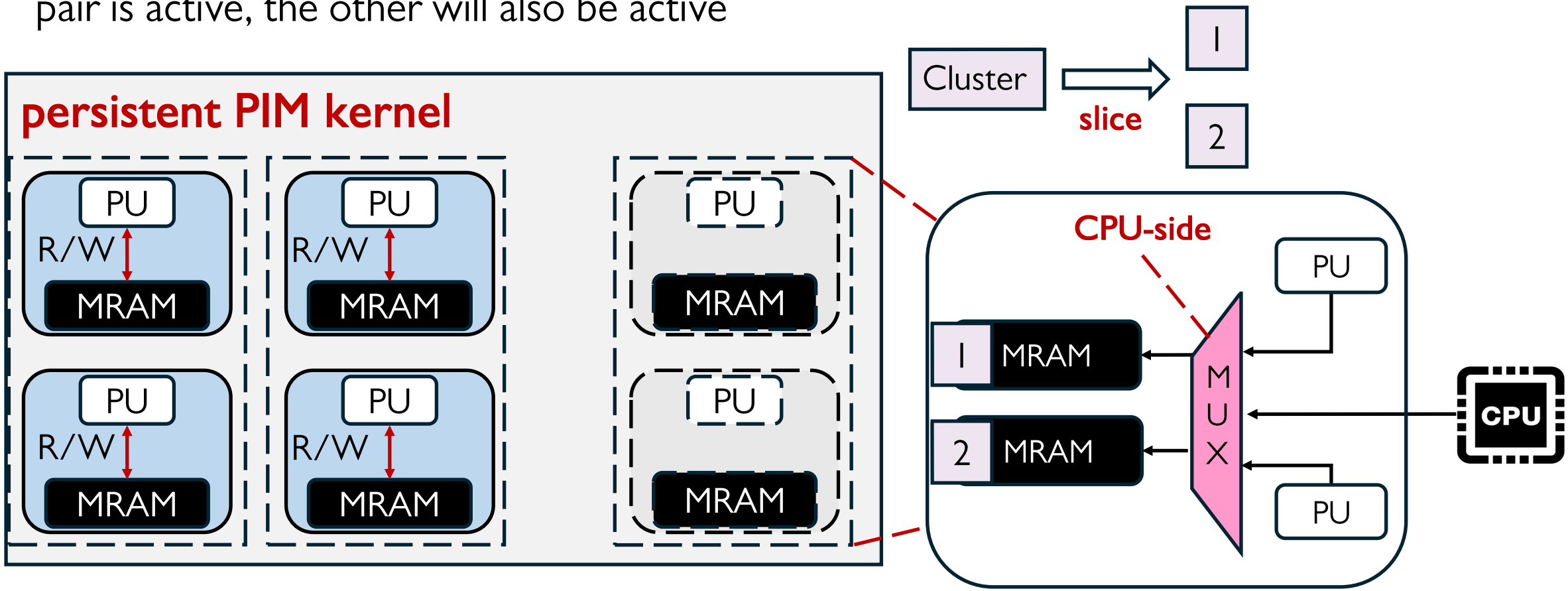
Design 1: Persistent PIM kernel

- Two adjacent PUs share a physical MUX, it is necessary to simultaneously switch the ownership of two PUs

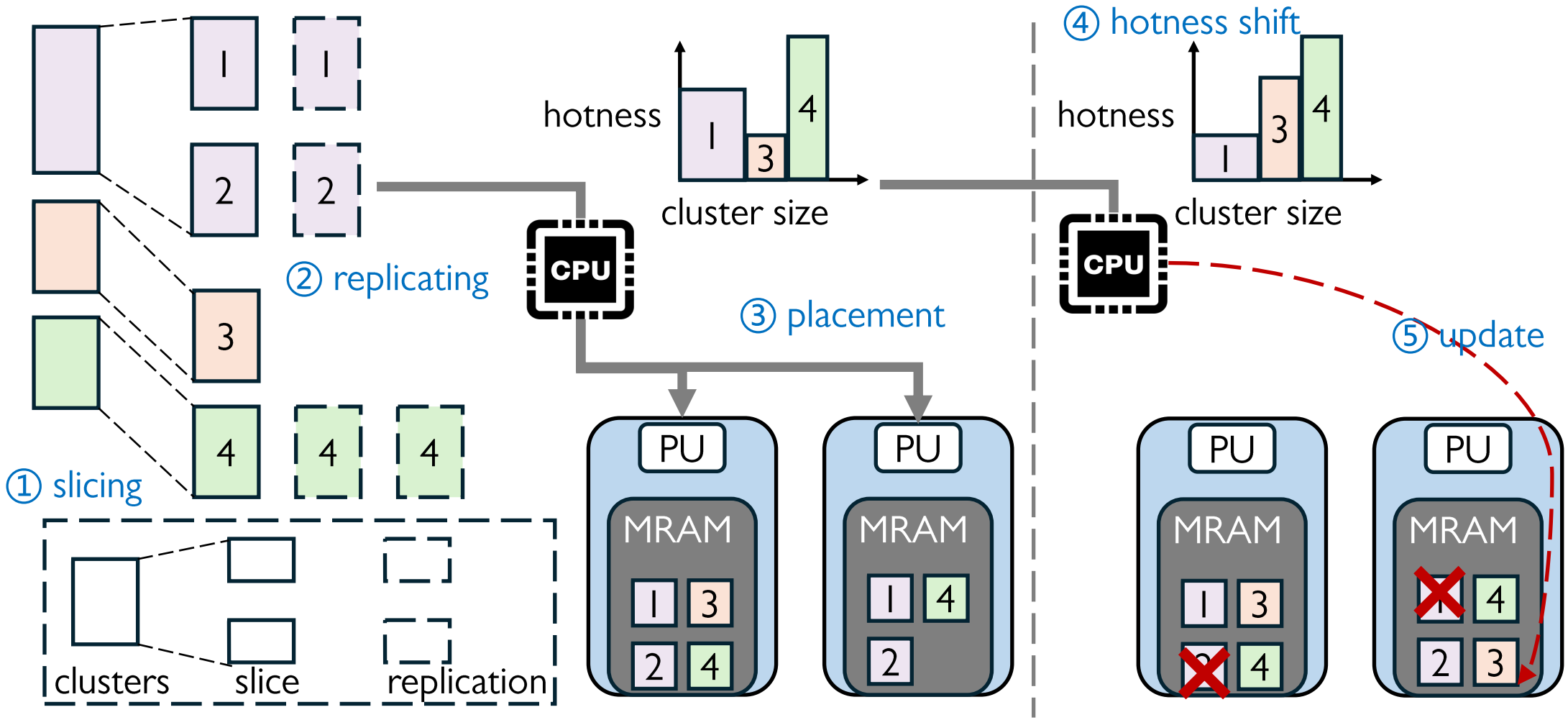


Design 1: Persistent PIM kernel

- Two adjacent PUs share a physical MUX, it is necessary to simultaneously switch the ownership of two PUs
- Each cluster is sliced into two smaller cluster slices, ensures that when one PU in a pair is active, the other will also be active



Design 2: Per-PU query dispatching



For details, please refer to the paper

Evaluation: Competitors && Hardware Platform

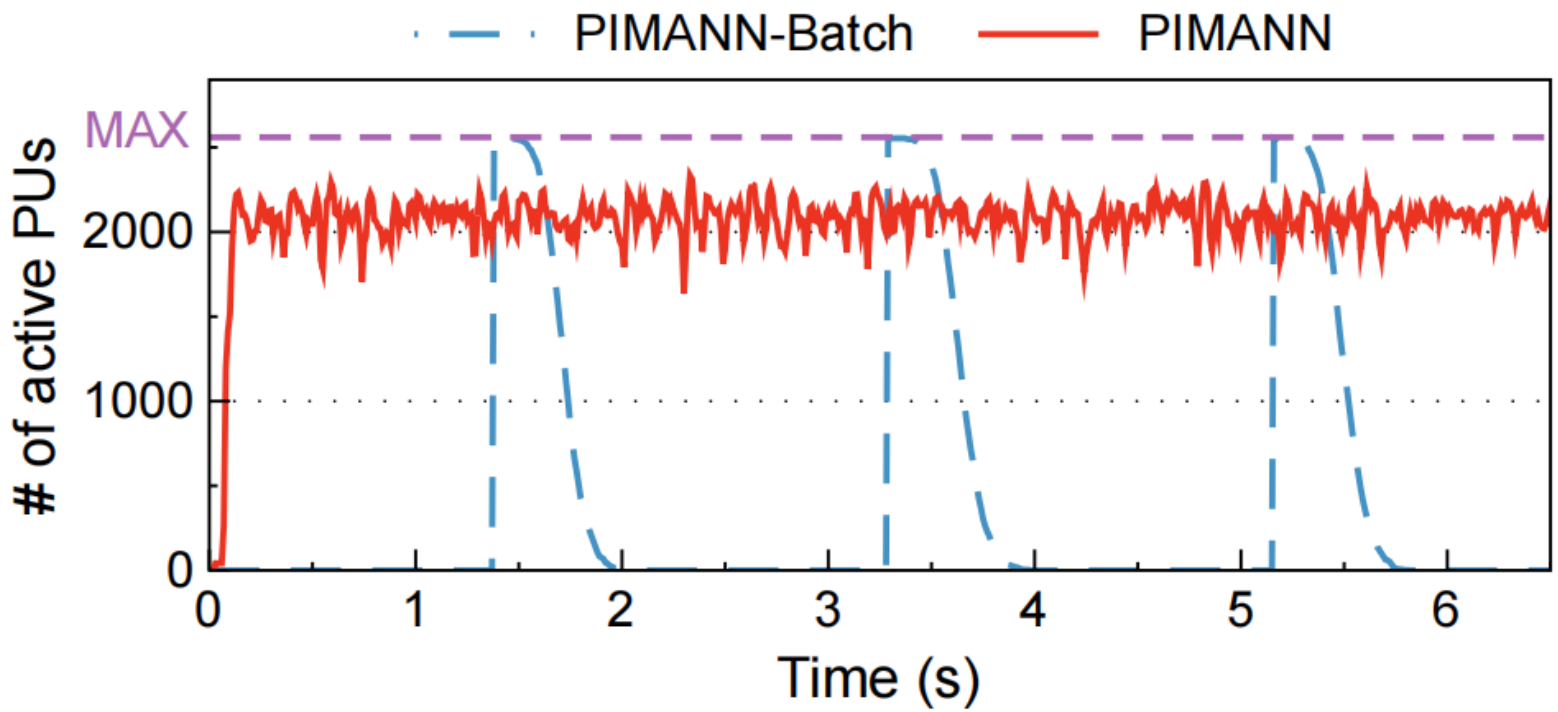
Faiss-CPU	2 * Intel Xeon Silver 4210 CPUs
Faiss-GPU	NVIDIA RTX A6000
PIMANN-Batch	UPMEM (Batch scheduling)
PIMANN	**UPMEM (Per-PU scheduling)**

Evaluation: Datasets

	Size	Dim	Data Type	Parameter (IVFPQ)
SIFT-1B	1 billion	128	uint8	Cluster=4096, M=32
SPACE-1B	1 billion	100	int8	Cluster=4096, M=20

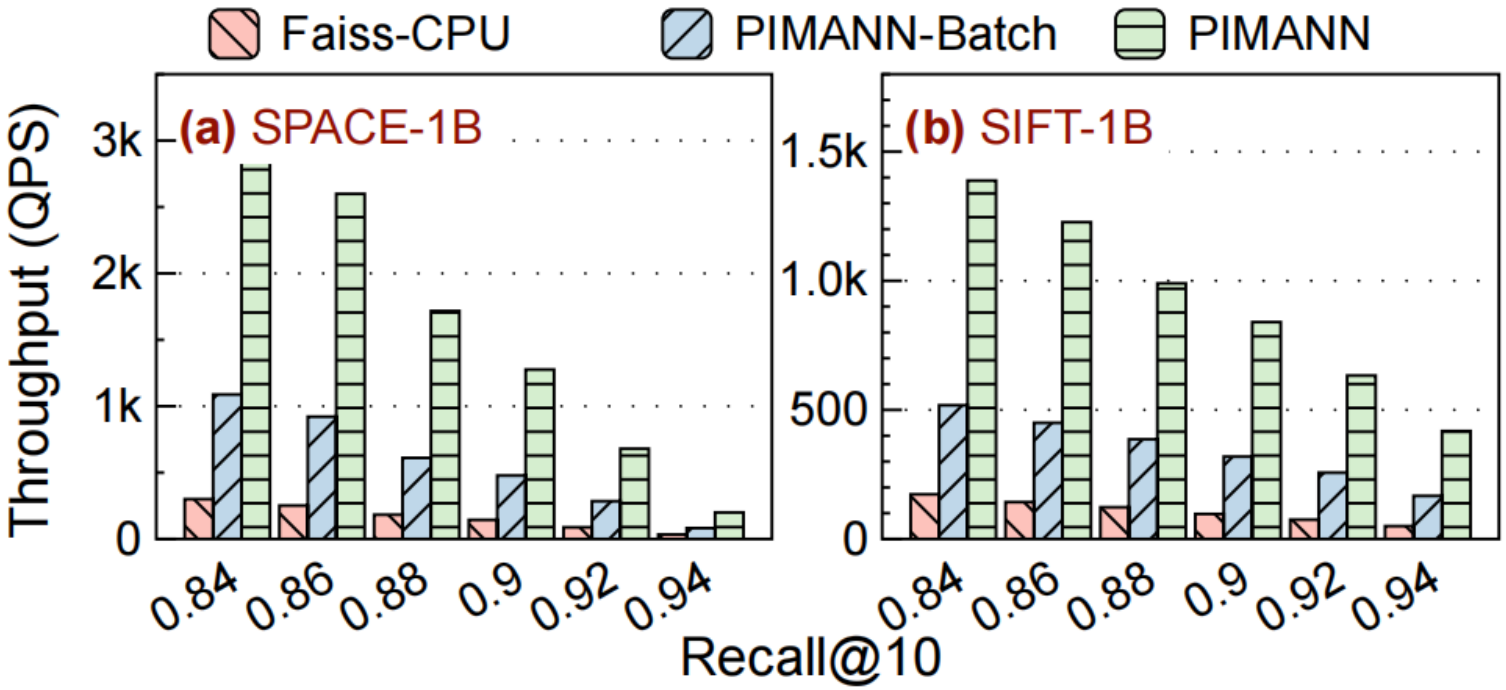
Evaluation: PIM utilization

- PIMANN achieves **~80%** utilization vs. **18.2%** for PIMANN-Batch
- Sustained high utilization from:
  - Persistent PIM kernels (eliminates inter-batch underutilization)
  - Per-PU query dispatching (eliminates intra-batch underutilization)"



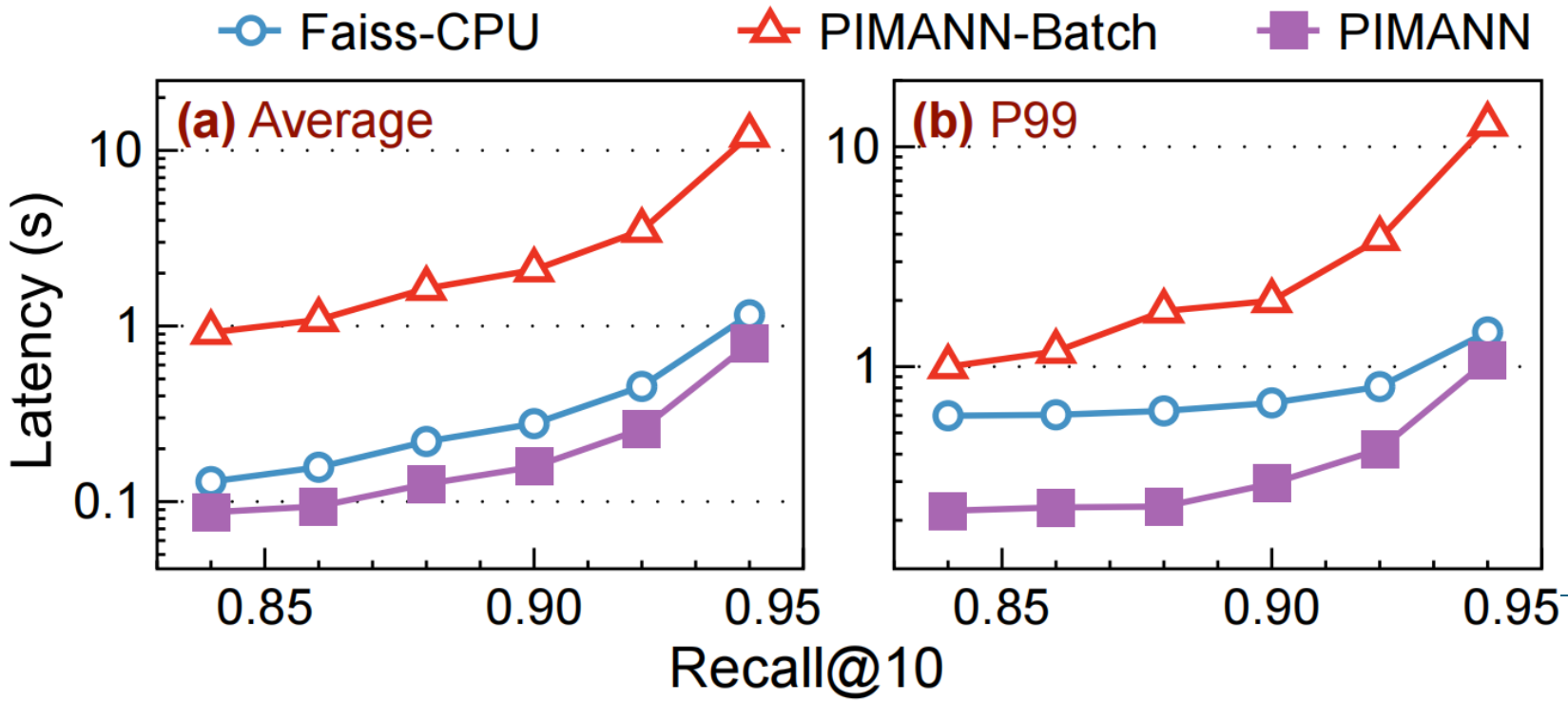
Evaluation: Throughput

- PIMANN achieves:
  - **2.4×–2.9×** higher throughput vs. PIMANN-Batch
  - **5.9×–10.4×** higher throughput vs. Faiss-CPU



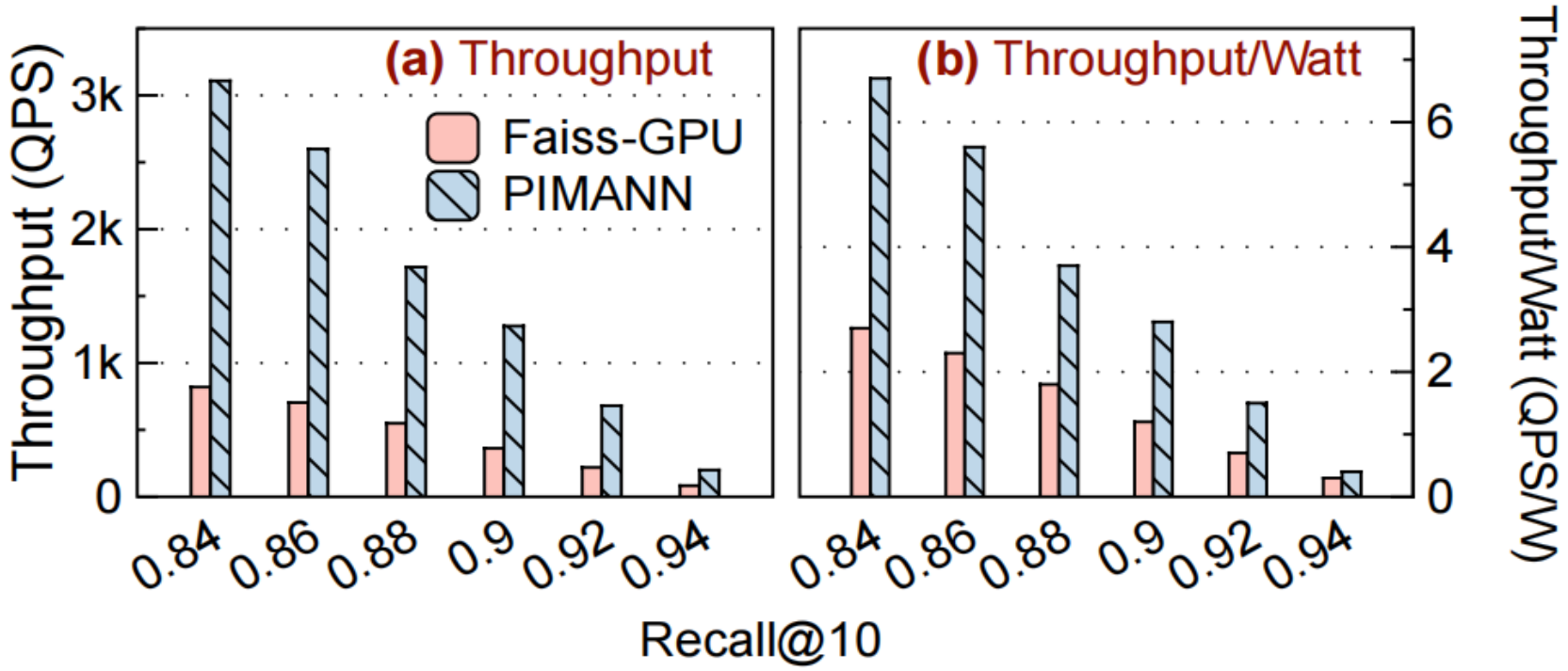
Evaluation: Latency

- PIMANN reduces latency vs. Faiss-CPU:
  - Avg latency: **reduces 32–43%**
  - Tail latency: **reduces 26–63%**
  - (PIMANN-Batch incurs higher latency)



Evaluation: Comparison with GPU-based systems

- PIMANN outperforms Faiss-GPU:
  - Throughput: **improves 2.4×–3.7×**
  - Throughput/Watt: **improves 1.6×–2.5×**



Conclusion

PIMANN is an ANNS system designed to fully exploit the potential of UPMEM,
Key Idea is Per-PU scheduling.

Code: https://github.com/cds-ruc/PIM-ANNS







wupuqing2022@ruc.edu.cn

Thanks!