A Joint Management Framework for Vector Similarity Search with Near-Memory Processing Systems

M.S. Student : Chun-Chien Liu (劉俊鍵)

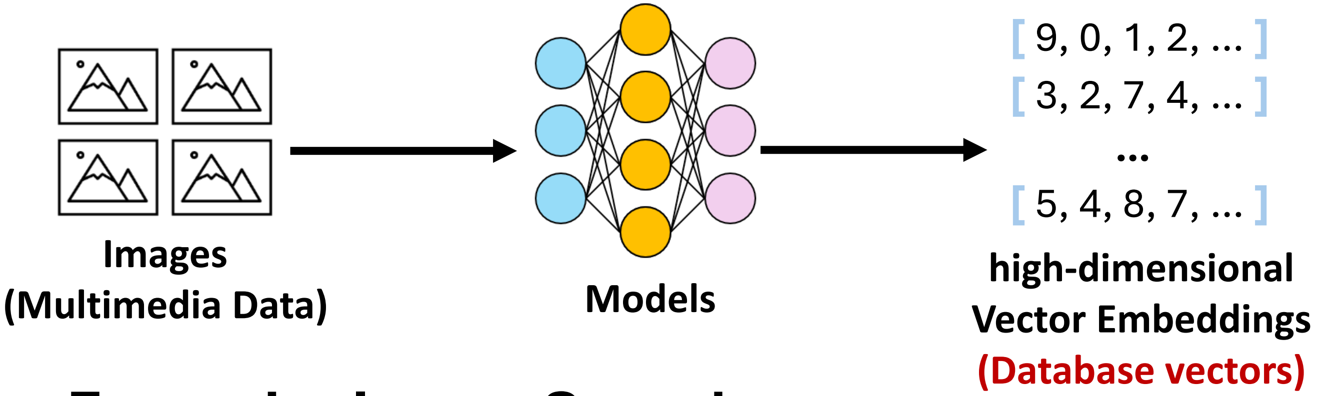
Advisor : Chun-Feng Wu (吳俊峯)

Department of Computer Science

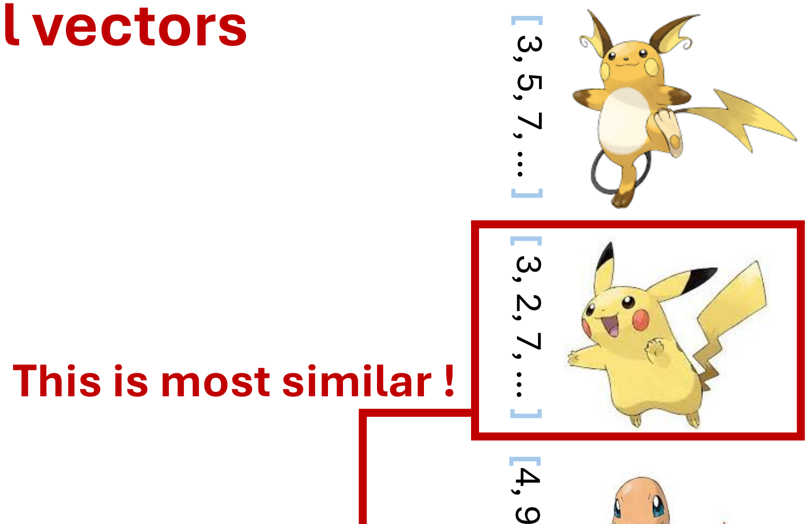
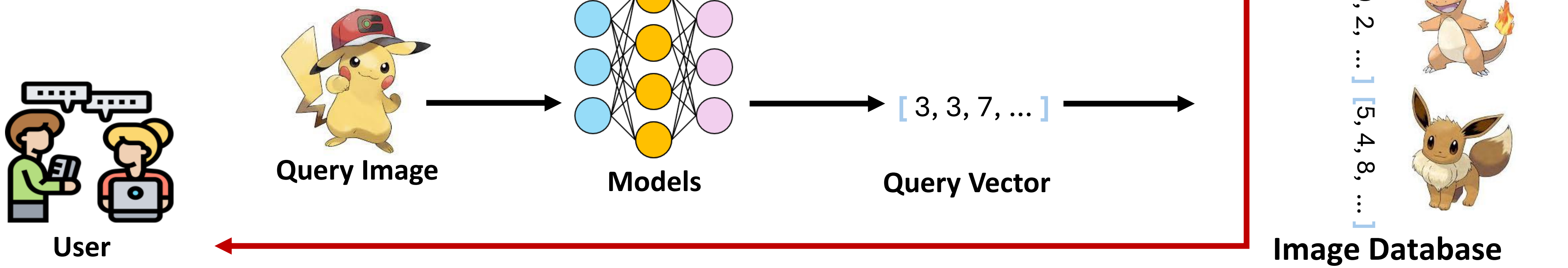
National Yang Ming Chiao Tung University

Introduction – Vector Similarity Search

- **Vector Similarity Search** is a **cornerstone of many modern applications**
  - e.g., LLMs, Recommender systems, information retrieval, and search engines
  - Instead of processing raw data like images or documents, modern systems represent them as **high-dimensional vectors**

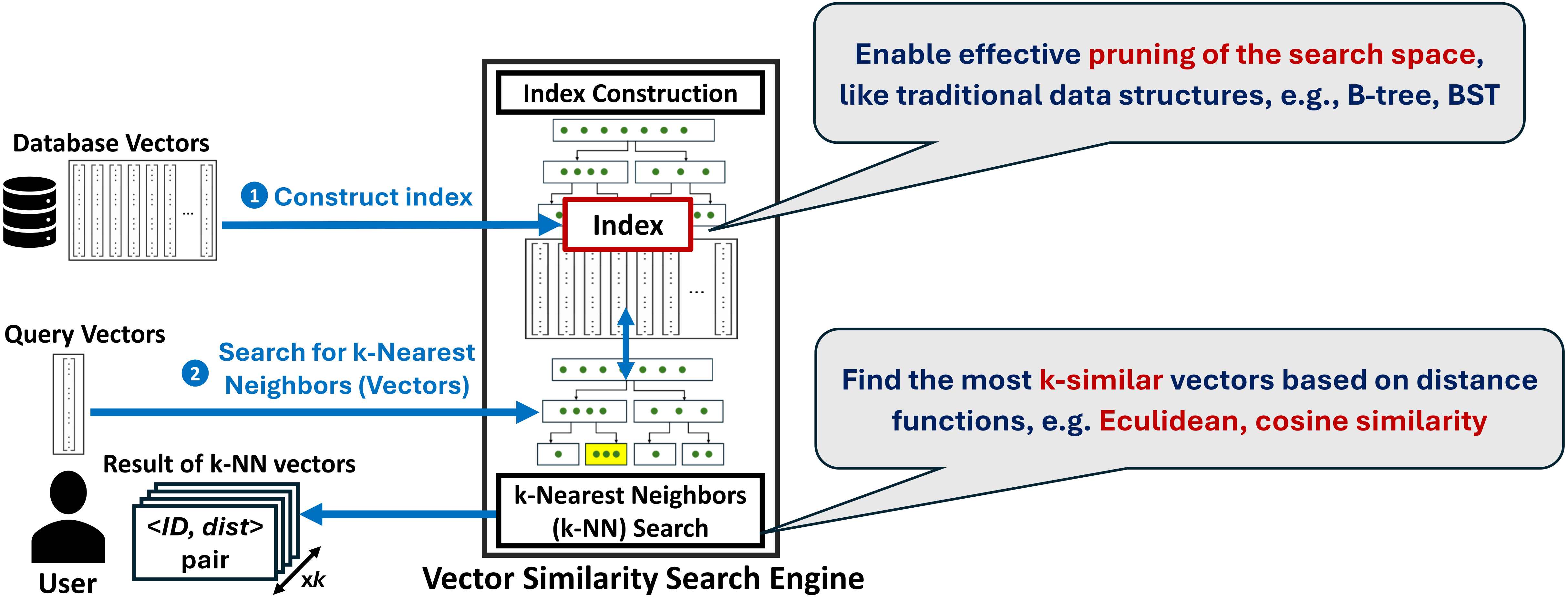


  - **Example: Image Search**

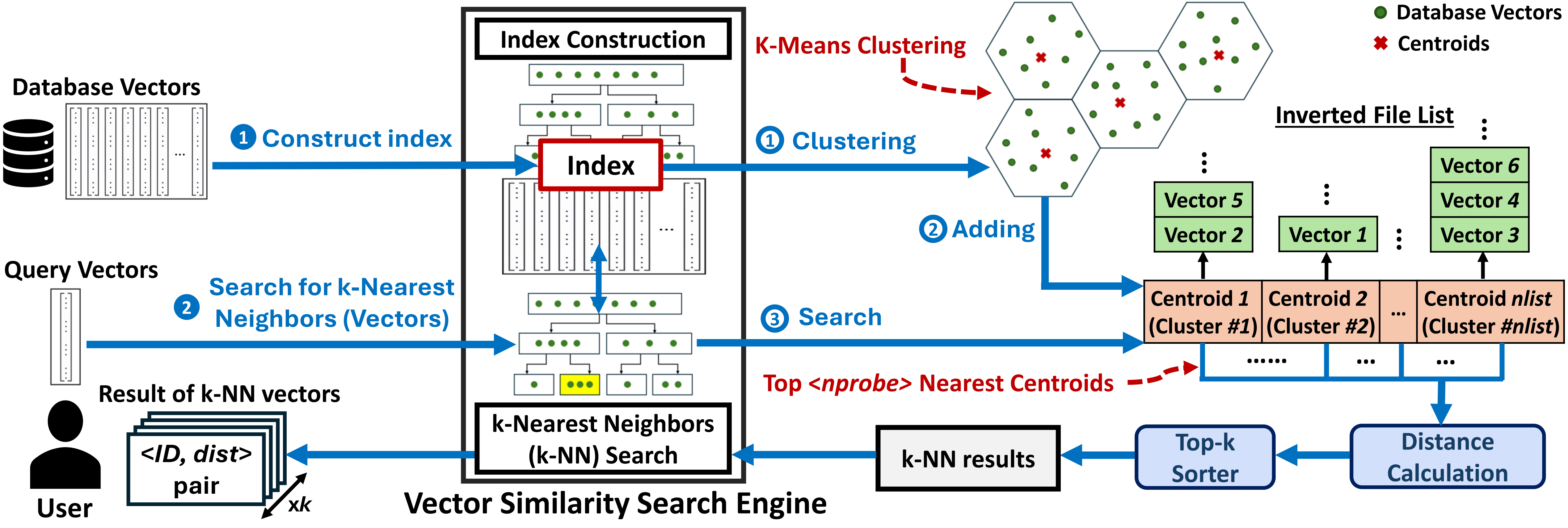
Introduction – Vector Similarity Search

- Workflow of Vector Similarity Search
  - (1) Index Construction → (2) k-Nearest Neighbors (k-NN) Search



Introduction – Vector Similarity Search

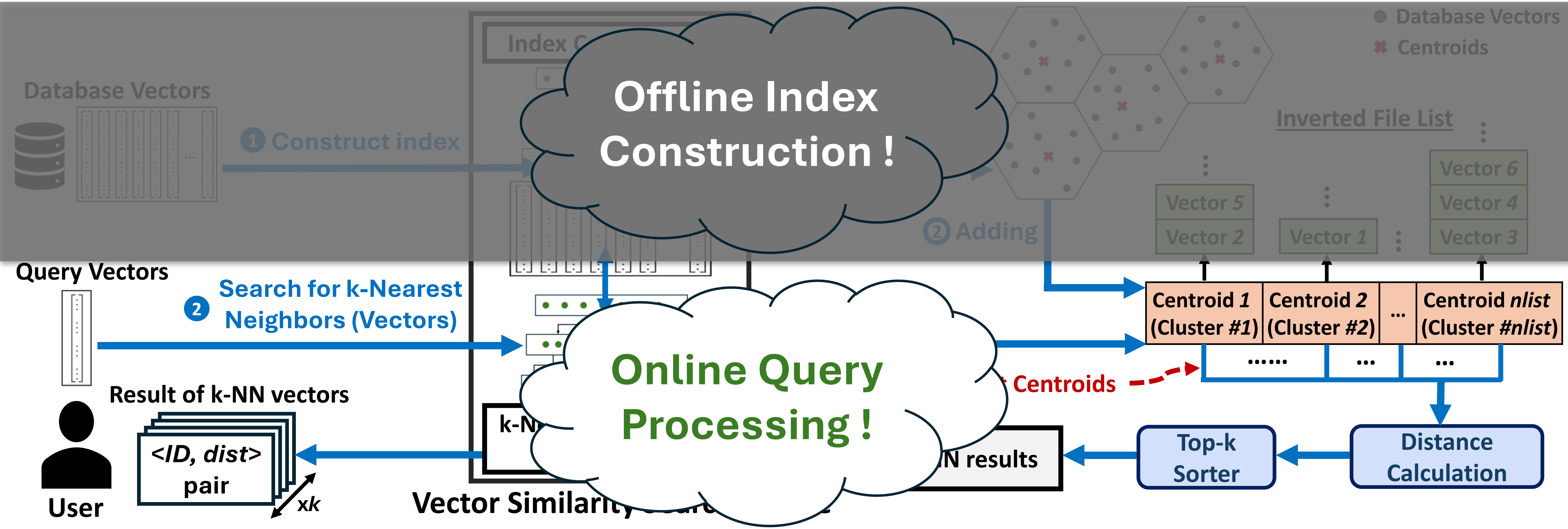
- Workflow of Vector Similarity Search
  - (1) Index Construction → (2) k-Nearest Neighbors (k-NN) Search
  - Exact search is inefficient **[Curse of dimensionality]**
    ***Sol: Heuristic (Approximate) Search* — e.g., Inverted File (IVF)**



Introduction – Vector Similarity Search

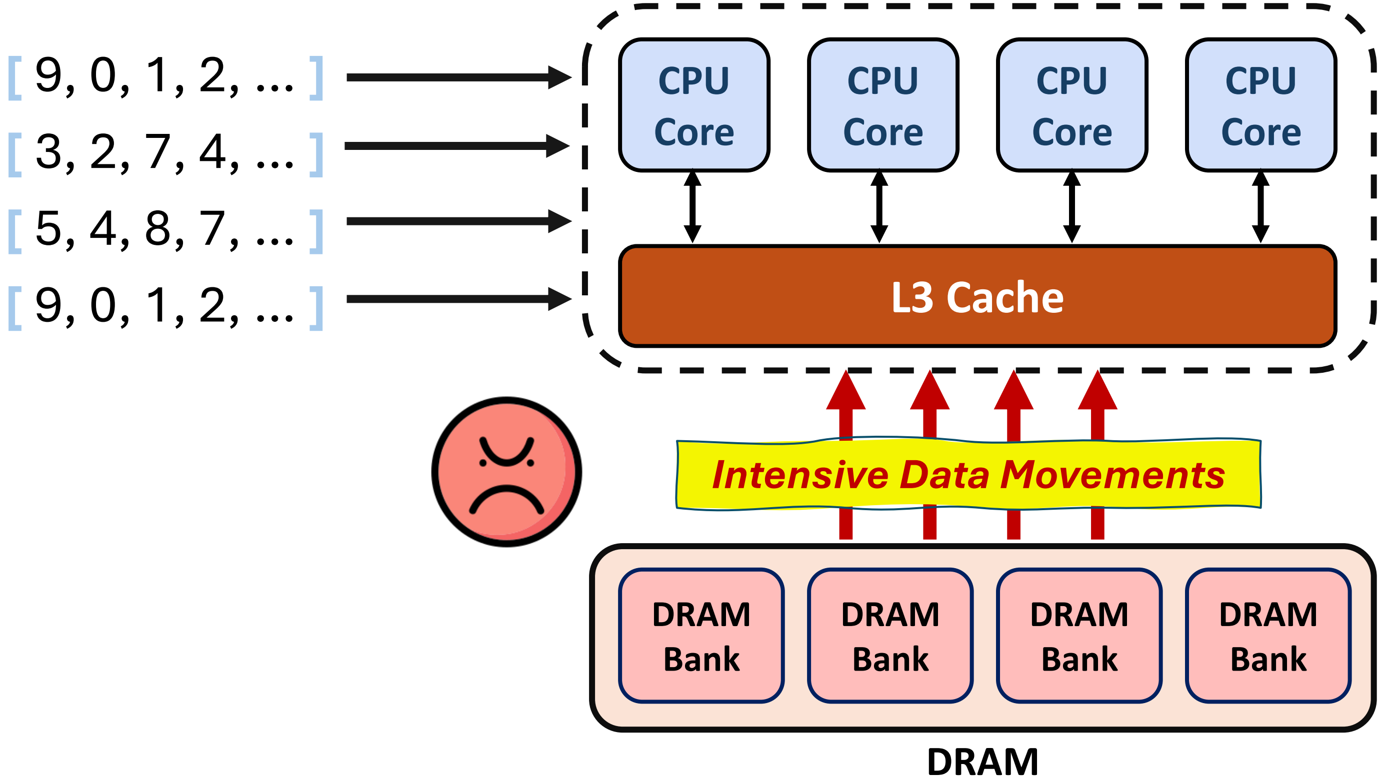
- Workflow of Vector Similarity Search
  - (1) Index Construction → (2) k-Nearest Neighbors (k-NN) Search
  - Exact search is inefficient **[Curse of dimensionality]**

    ***Sol: Heuristic (Approximate) Search* — e.g., Inverted File (IVF)**



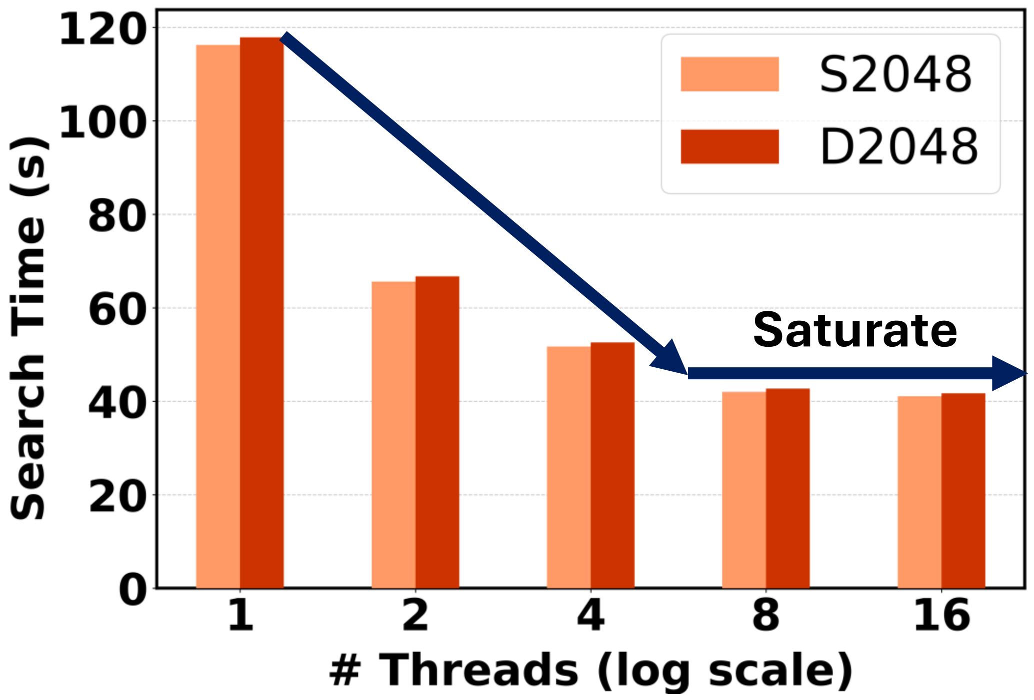
Motivation – Speculation

- **Vector Similarity Search** is a **memory intensive (DRAM-bounded)** application
    - Low CPU cache reuse rate
    - Low computation – I/O ratio
        - SIMD (AVX2, AVX512), BLAS Library



Motivation – Our Findings

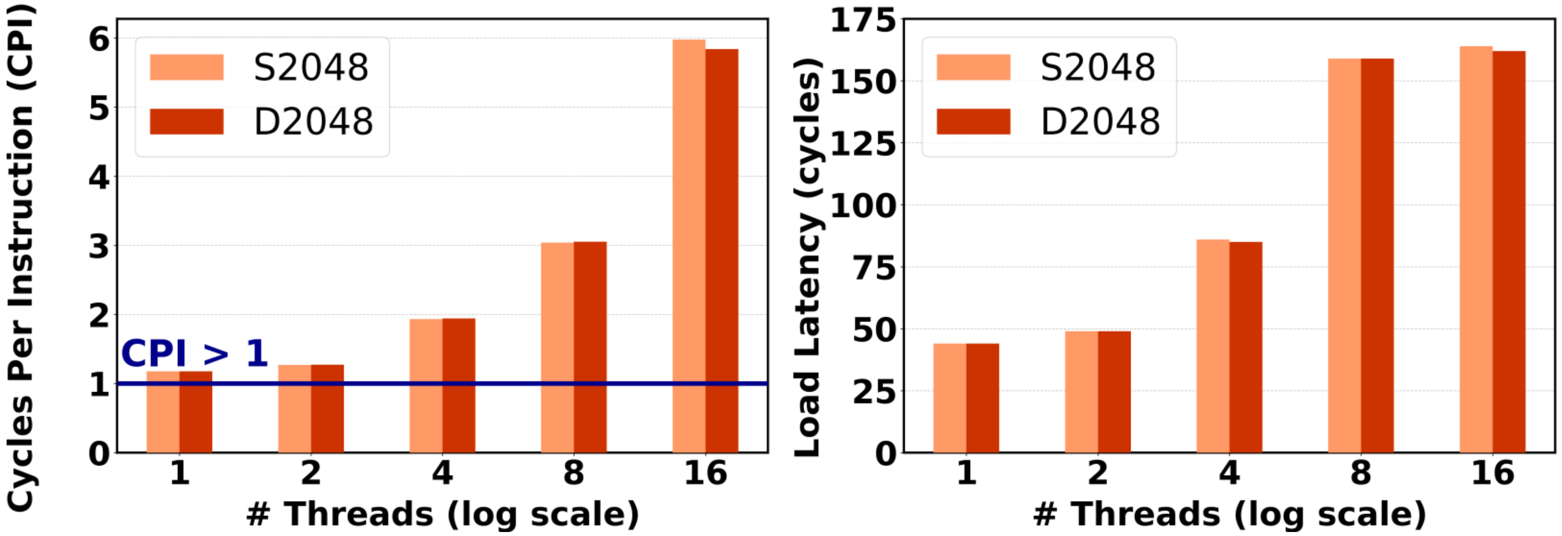
- **Vector Similarity Search** is a **memory intensive (DRAM-bounded)** application
- Scaling of Vector Similarity Search on CPU
  - Profiling META's FAISS-IVF
  - Bottlenecks
    - **Execution time**



Performance tends to saturate as the number of CPU threads grows

Motivation – Our Findings

- **Vector Similarity Search** is a **memory intensive (DRAM-bounded)** application
- Scaling of Vector Similarity Search on CPU
  - Profiling META's FAISS-IVF
  - Bottlenecks
    - Execution time
    - **Cycles Per Instruction**
    - **Load Latency**

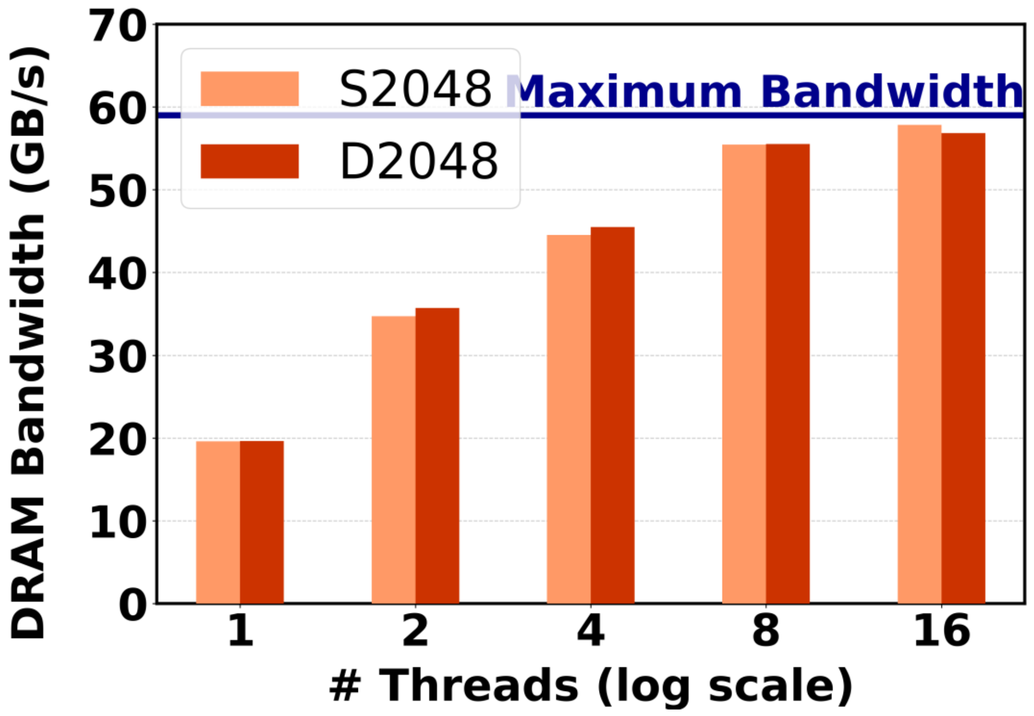


As the number of CPU thread grows, Cycle Per Instruction (CPI) increases. This means that **CPU threads spend more time idle.**

As the number of CPU thread grows, load latency also increases, likely due to **congestion in the shared DRAM bus.**

Motivation – Our Findings

- **Vector Similarity Search** is a **memory intensive (DRAM-bounded)** application
- Scaling of Vector Similarity Search on CPU
  - Profiling META's FAISS-IVF
  - Bottlenecks
    - Execution time
    - Cycles Per Instruction
    - Load Latency
    - **DRAM Bandwidth Utilization**

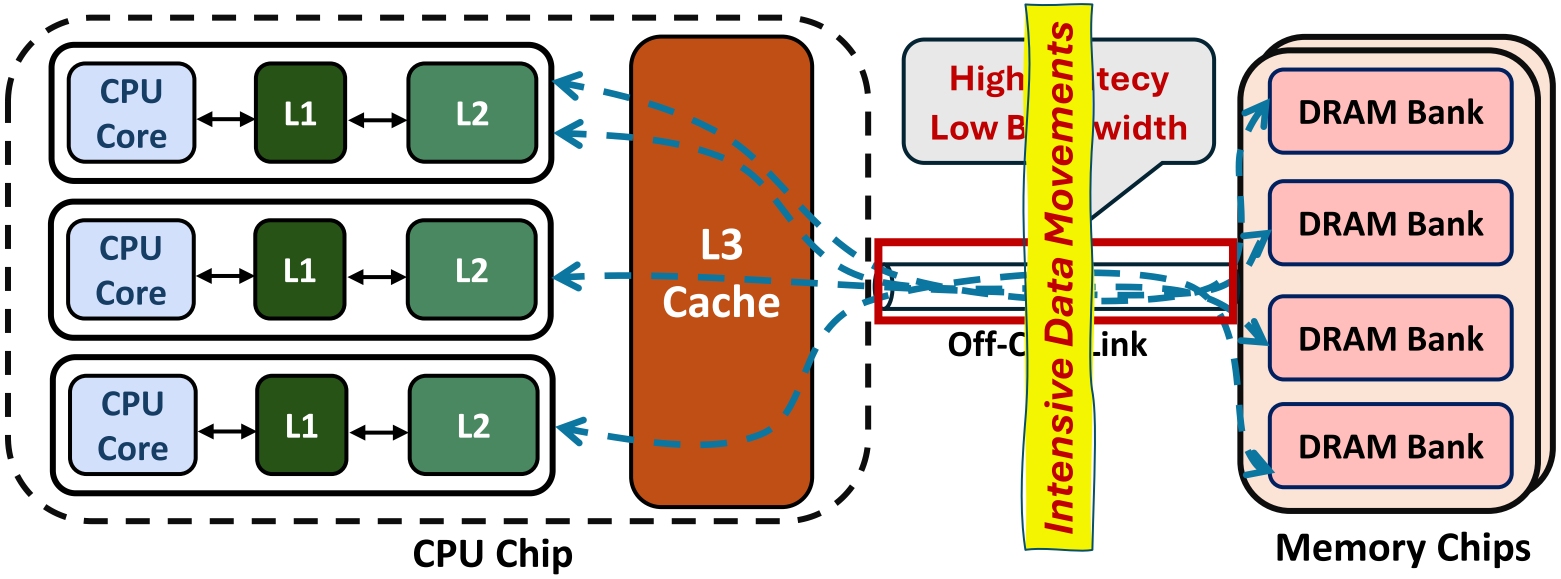


DRAM bandwidth tends to saturate as the number of CPU threads grows.
The shared DRAM bus **failing to meet growing bandwidth demands.**

Introduction – Data Movement Bottleneck

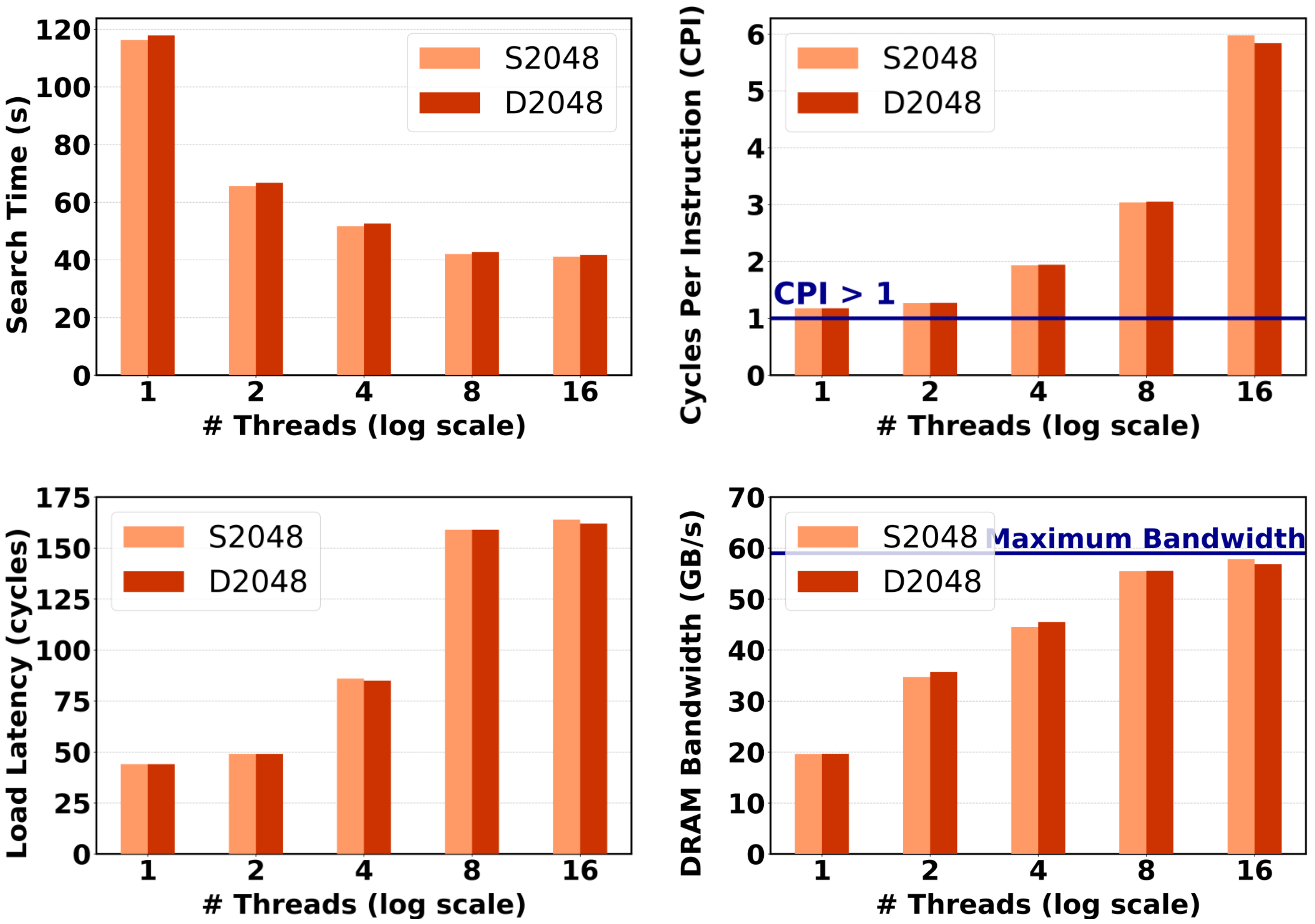
- **Memory Wall —** caused by the growth gap between CPU and Memory bandwidth
- **Data movement** becomes a **bottleneck** in modern **compute-centric systems**
  - **Ineffective use of the cache hierarchy** and **not enough memory bandwidth**
  - Off-chip data movement **over the shared bus**
  - The Data Movement Bottleneck deteriorates in **data-intensive applications**
    - e.g., ML systems, database systems, **Vector Similarity Search**, etc

Compute-Centric Systems



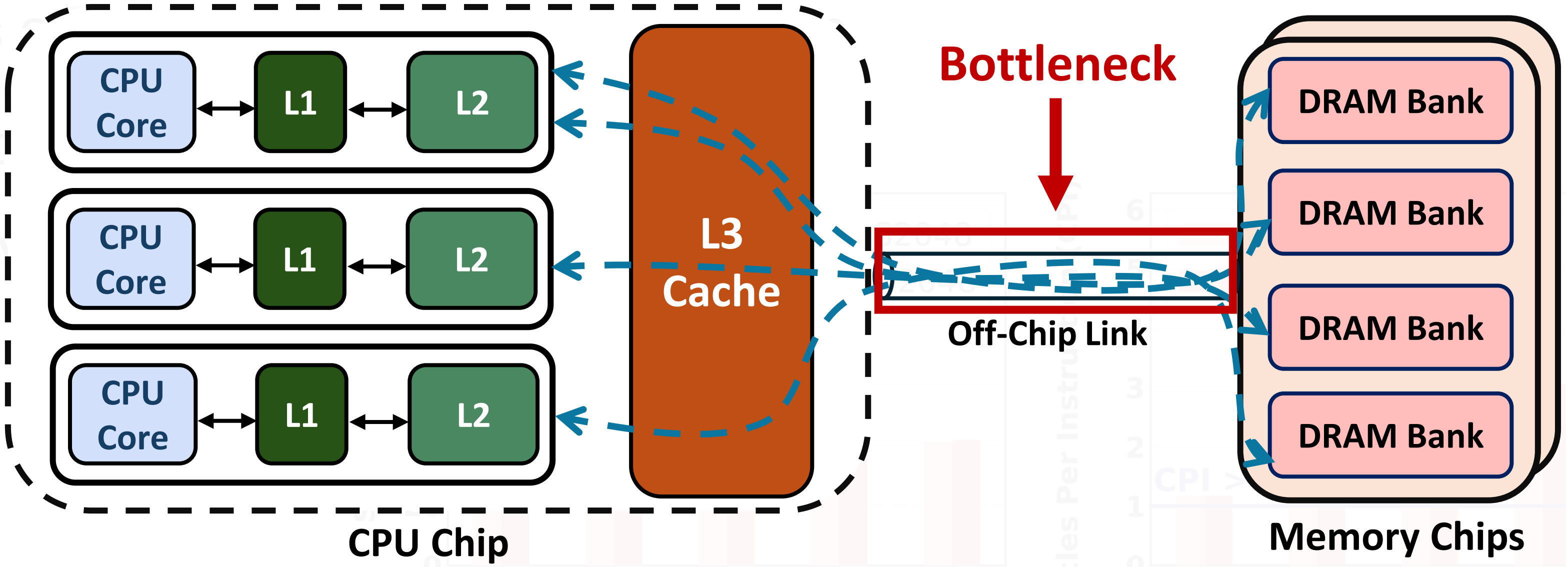
Motivation – Our Findings

- **Vector Similarity Search** is a **memory intensive (DRAM-bounded)** application
- Scaling of Vector Similarity Search on CPU
  - Profiling META's FAISS-IVF
  - Bottlenecks
    - **Execution time**
    - **Cycles Per Instruction**
    - **Load Latency**
    - **DRAM Bandwidth Utilization**

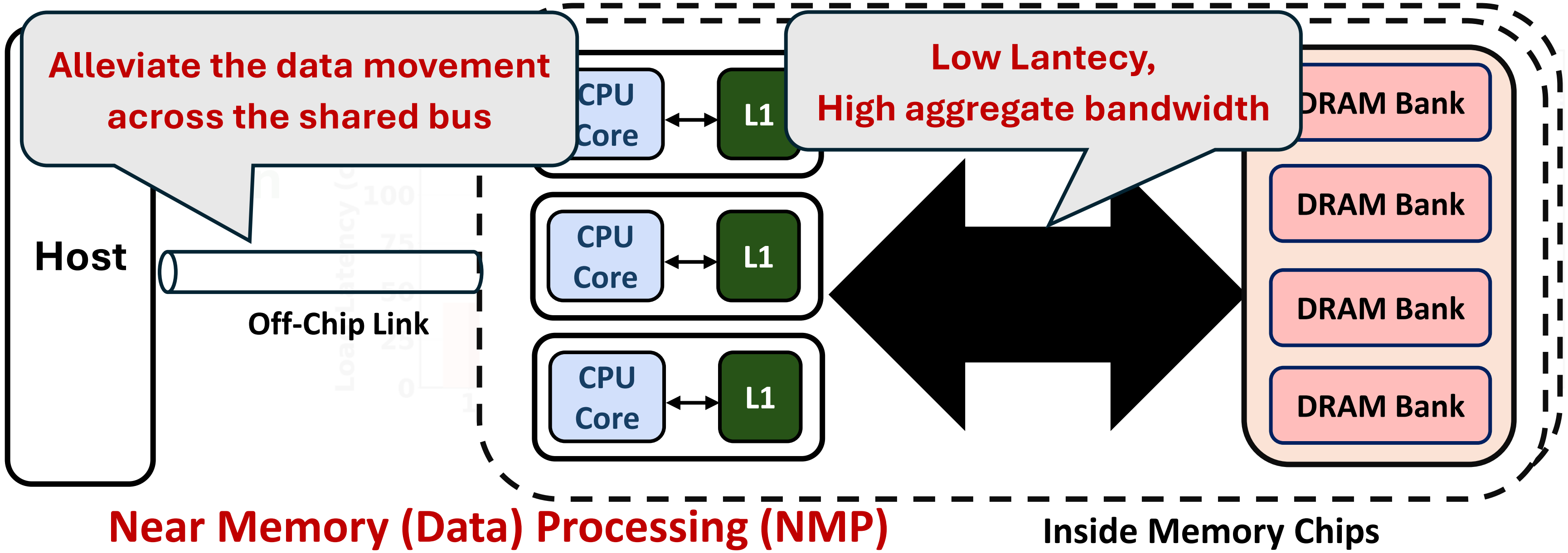


Promising Solution – Near Memory Processing (NMP)

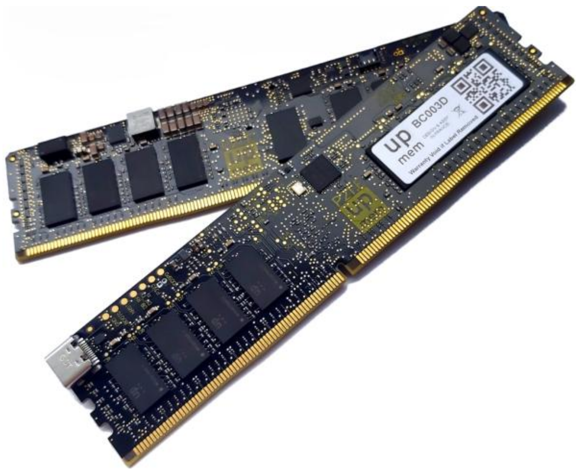
Compute-Centric Systems



Memory-Centric Systems



Real NMP Systems – UPMEM



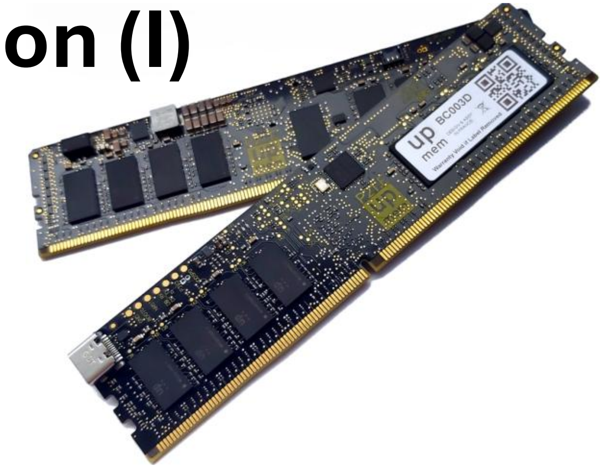
UPMEM DIMM [1]

- Summary
    - Introduce a **real NMP system – UPMEM**
        - The architecture of UPMEM DIMMs
        - How UPMEM works
    - Present the **Joint Management** of <u>**Vector Similarity Search**</u> with <u>**UPMEM**</u>
        - Data Partitioning with load balance
        - Host-NMP coordination
        - Efficient Memory Management

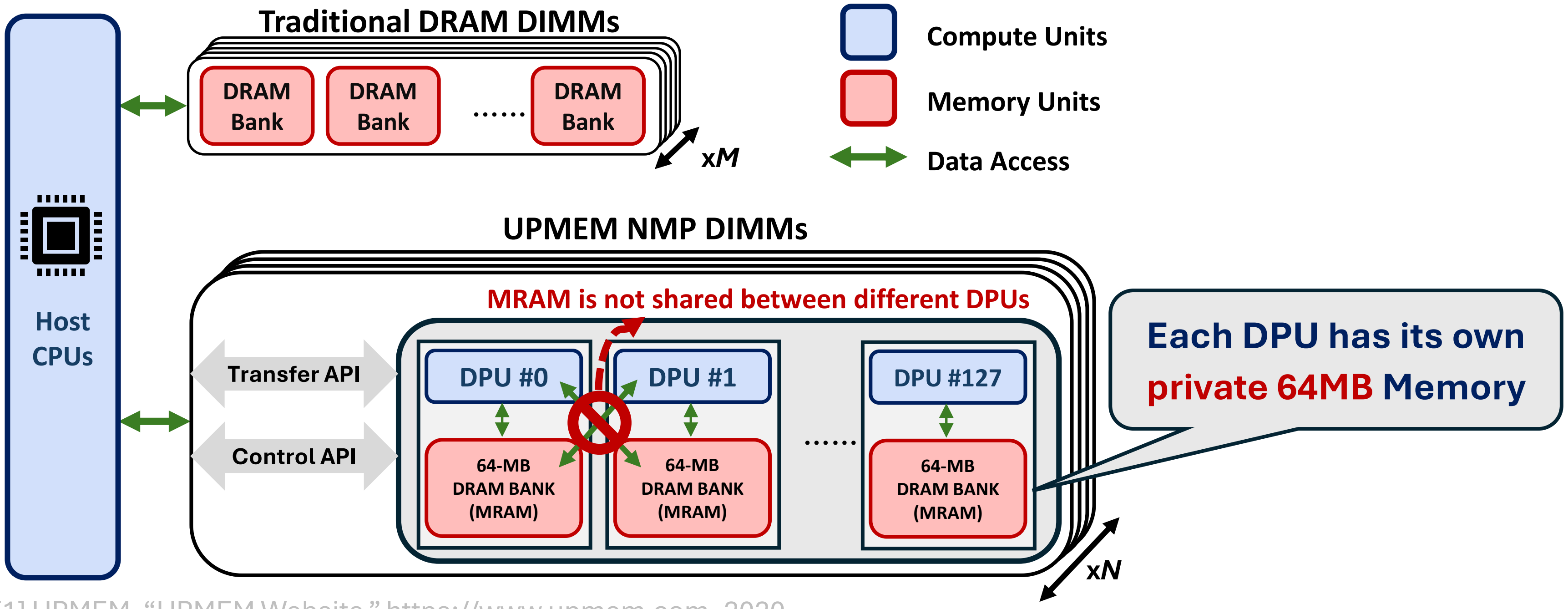
[1] UPMEM, “UPMEM Website,” https://www.upmem.com, 2020.

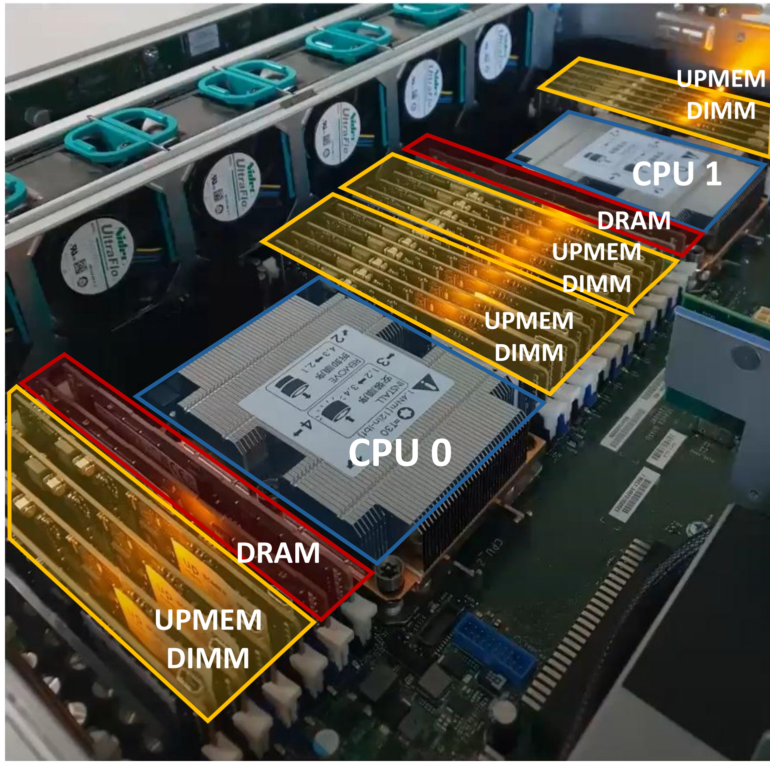
Real NMP Systems – UPMEM : System Organization (I)

- UPMEM NMP System
  - Host CPUs
  - **UPMEM DIMMs** coexist with regular DDR4 DIMMs
    - Combine RISC cores with traditional 2D DRAM arrays in a chip



UPMEM DIMM [1]



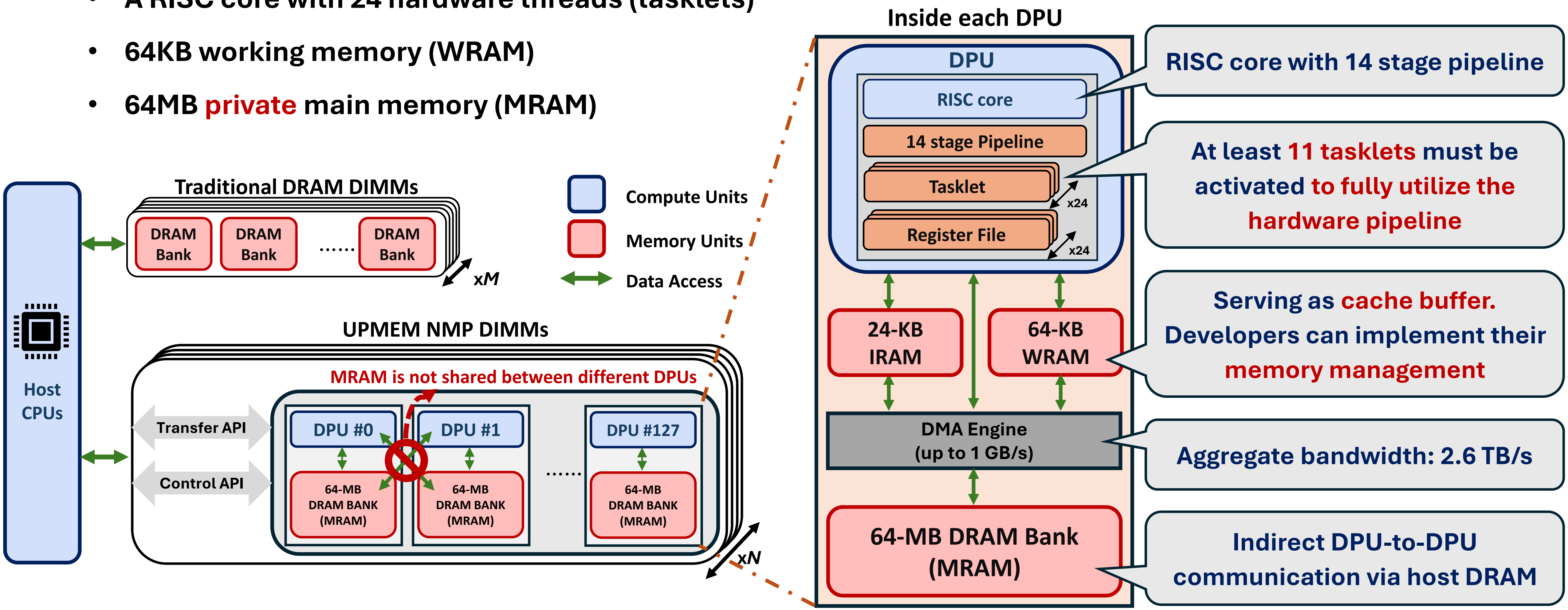


UPMEM NMP Server [2]

[1] UPMEM, "UPMEM Website," https://www.upmem.com, 2020.
[2] Gómez-Luna et al., Benchmarking a real processing-in-memory system, IEEE Access, 2022.

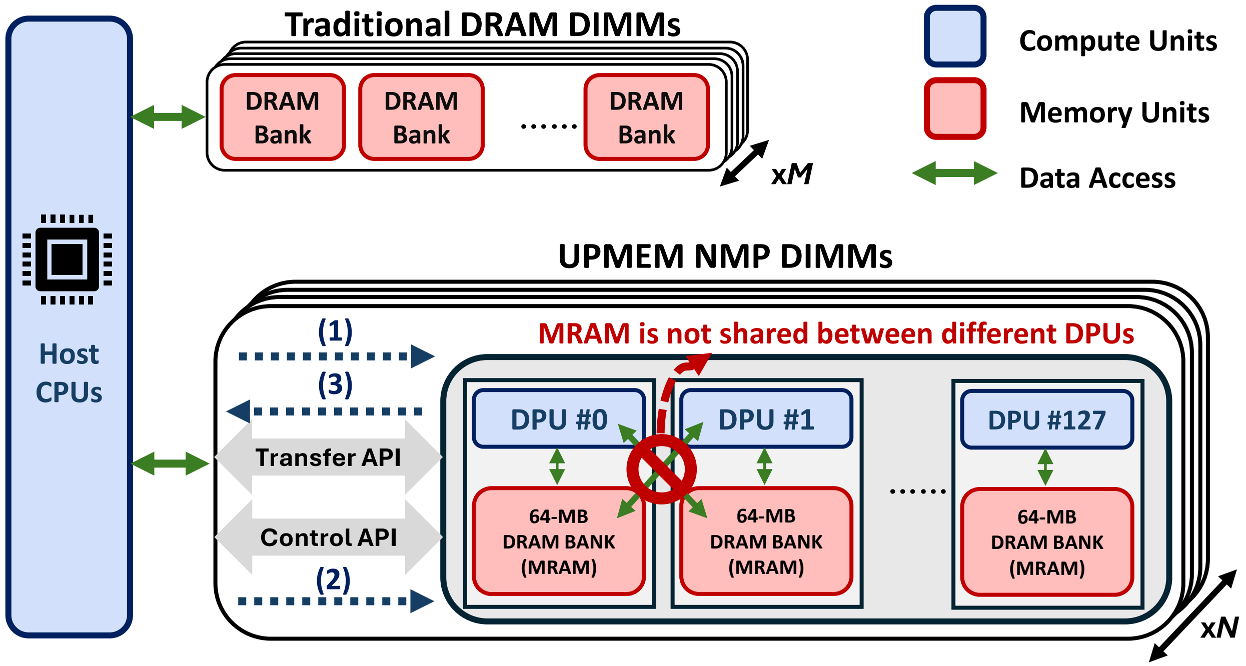
Real NMP Systems – UPMEM : System Organization (II)

- Each UPMEM DIMM contains 128 **DRAM Processing Units (DPUs)**
- **DRAM Processing Unit (DPU)**
  - **A RISC core with 24 hardware threads (tasklets)**
  - **64KB working memory (WRAM)**
  - **64MB private main memory (MRAM)**



Real NMP Systems – How DPU works

- UPMEM SDK
  - Transfer API
  - Control API



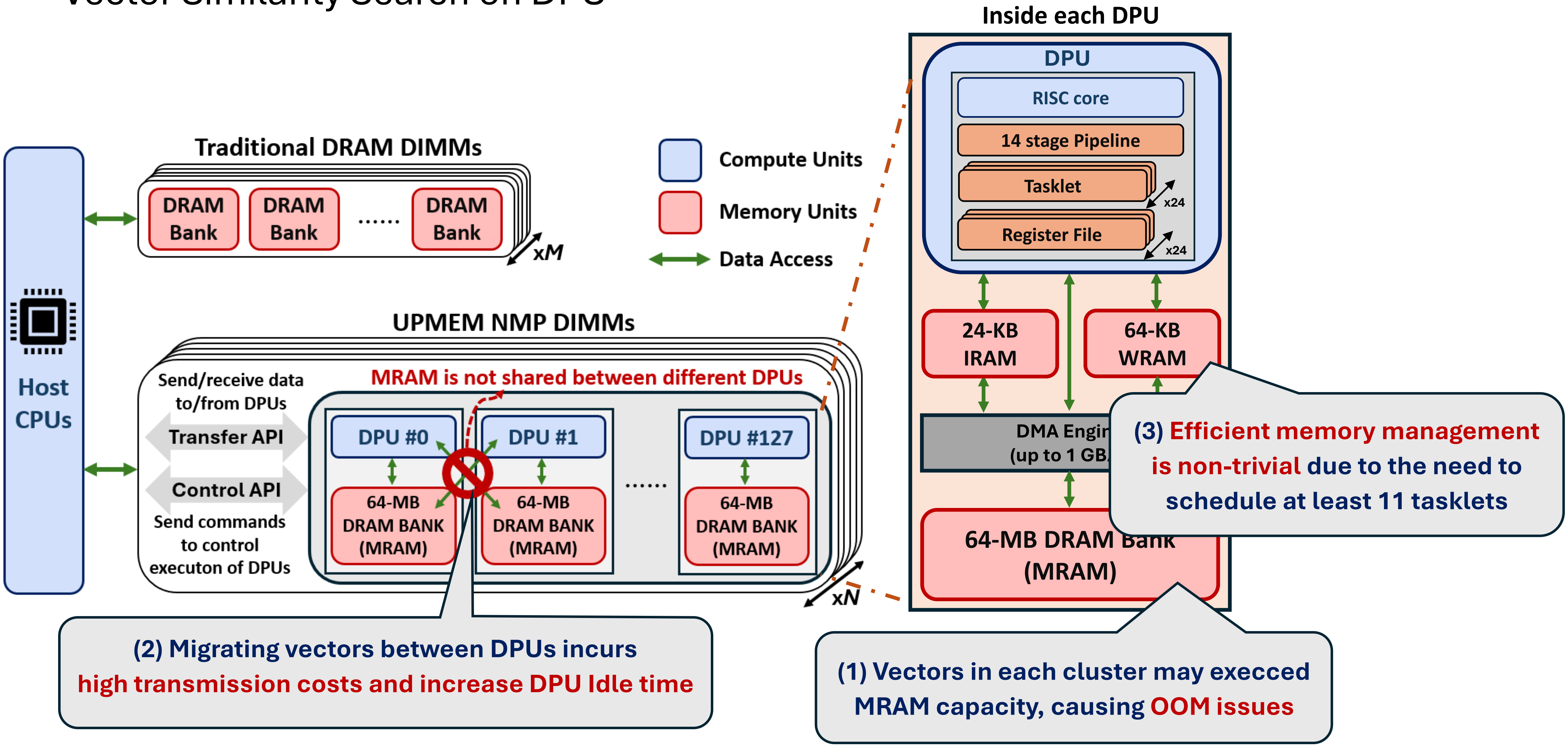
(1) Host sends data to each DPU's MRAM via Transfer API

(2) Host sends commands to trigger each DPU execution via Control API

(3) Host receives data from each DPU's MRAM via Transfer API

Challenges

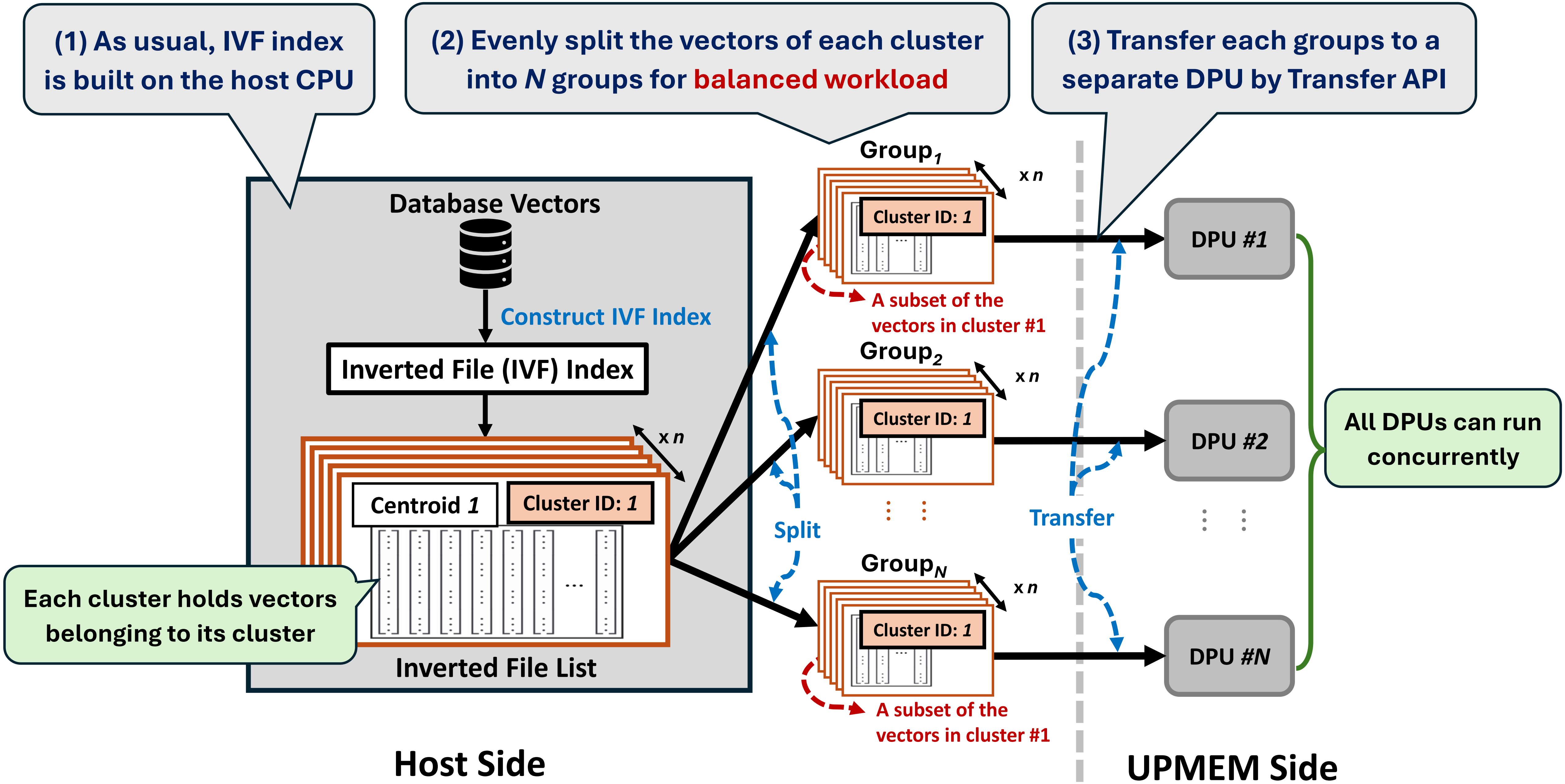
- Vector Similarity Search on DPU



Challenges & Solution – Design Overview

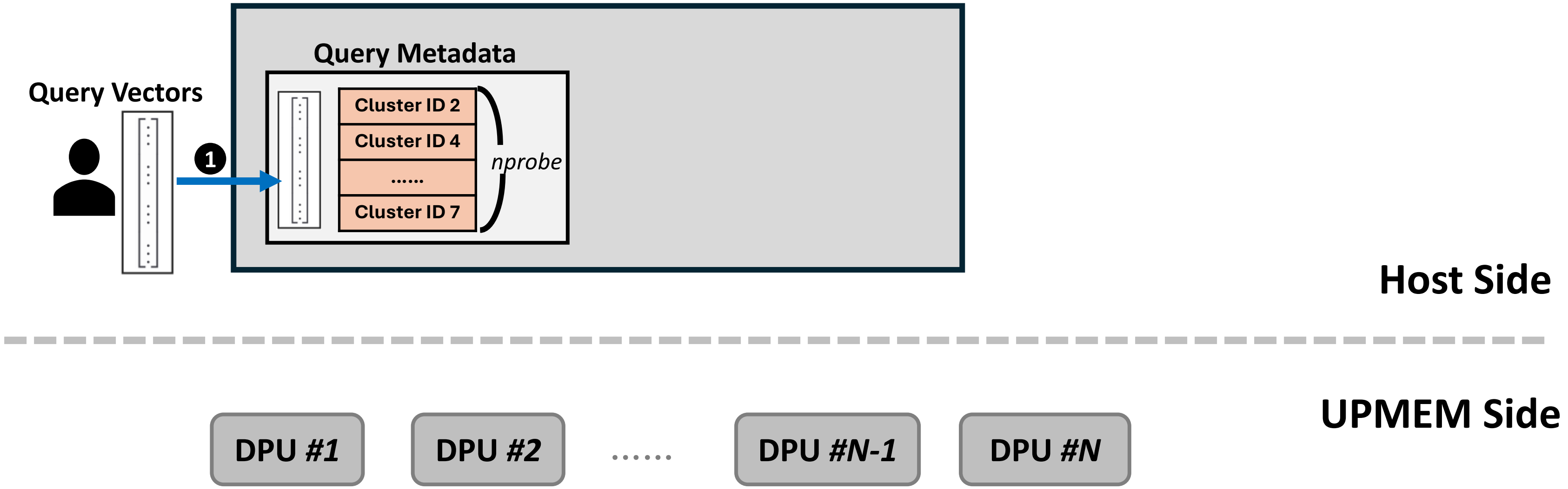
- **UPVSS: UPMEM-aware Vector Similarity Search**
  - **Key idea: Reduce data movement by offloading distance calculation** to DPUs
  - **(1) DPU-aware Clusters Partition**
    - Considering **load balance without synchronization** among DPUs
    - Exploits UPMEM's high parallelism and aggregate bandwidth
  - **(2) DPU-aware Vector Similarity Search Coordinator**
    - **Host-DPU collaboration** for distributed search
    - **Customized WRAM memory management** to maximize DPU hardware pipeline

UPVSS – DPU-aware Clusters Partition



UPVSS – DPU-aware Vector Similarity Search Coordinator

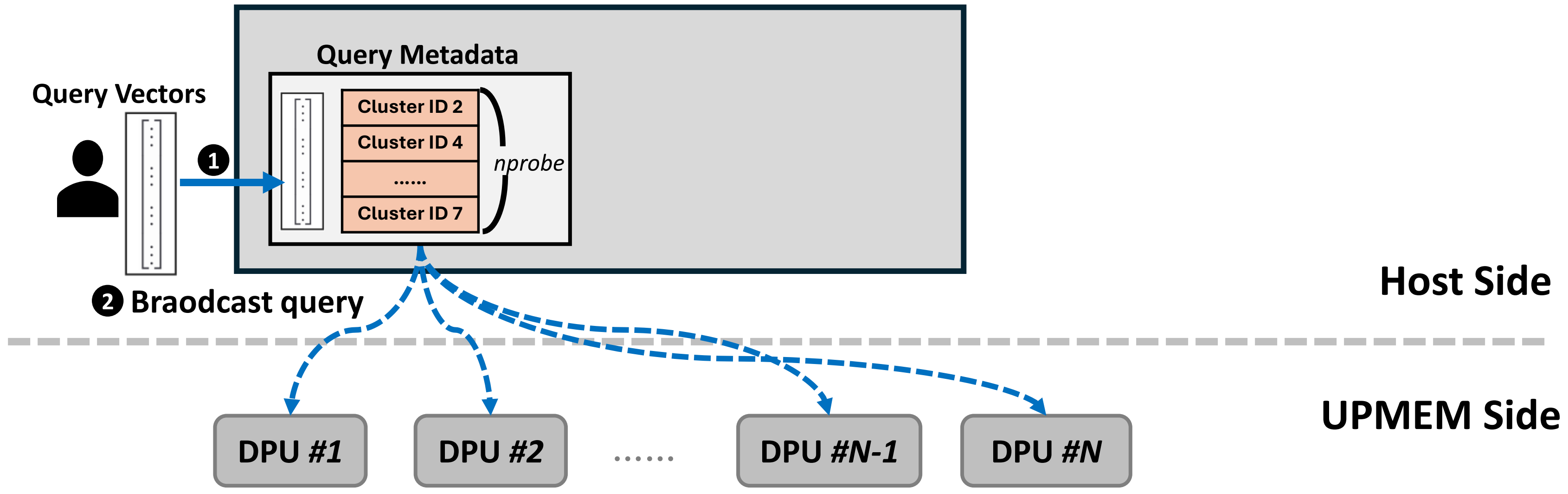
- Host-DPU collaboration
  - **Host**



(1) Host identifies <*nprobe*> centroids and forms *Query Metadata*

UPVSS – DPU-aware Vector Similarity Search Coordinator

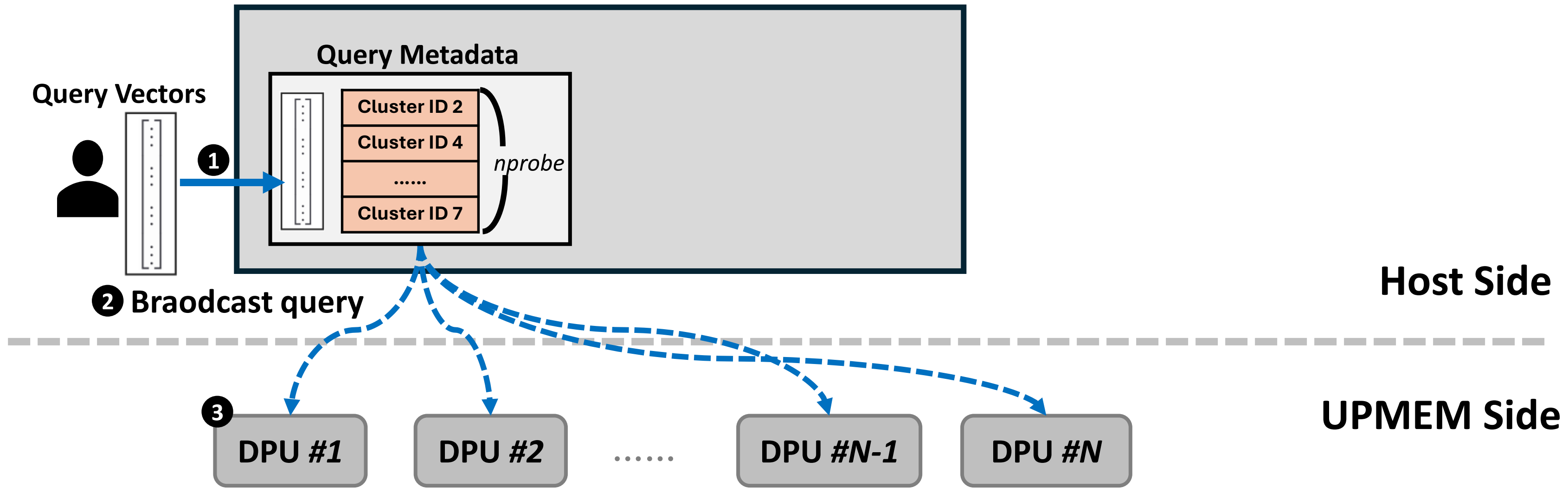
- Host-DPU collaboration
  - **Host**



(2) Broadcast *Query Metadata* to all DPUs by Transfer API

UPVSS – DPU-aware Vector Similarity Search Coordinator

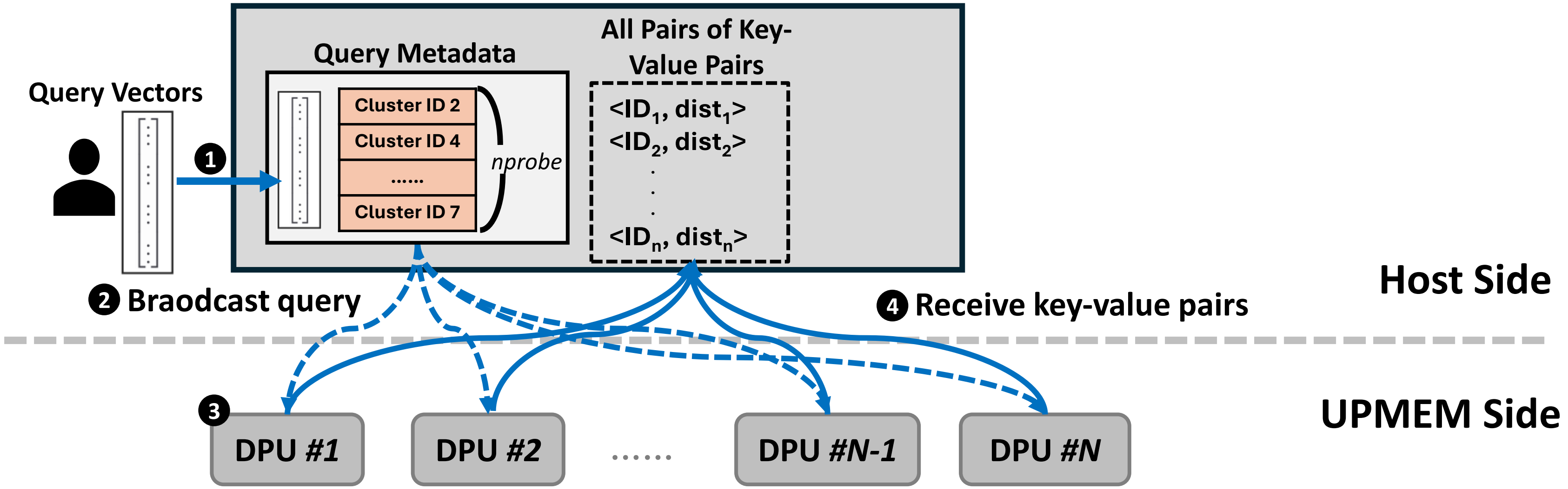
- Host-DPU collaboration
  - **Host**



(3) Host Launches all DPUs to execute distance calculations

UPVSS – DPU-aware Vector Similarity Search Coordinator

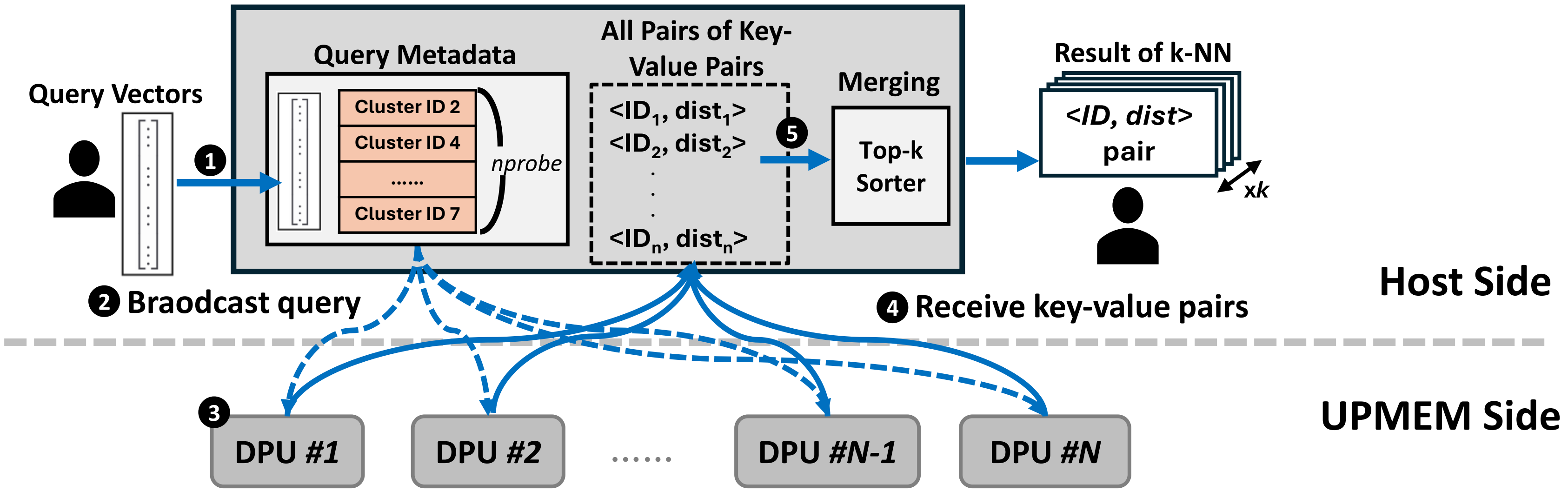
- Host-DPU collaboration
  - **Host**



(4) Each DPU returns <ID, dist> pairs to the host for merging

UPVSS – DPU-aware Vector Similarity Search Coordinator

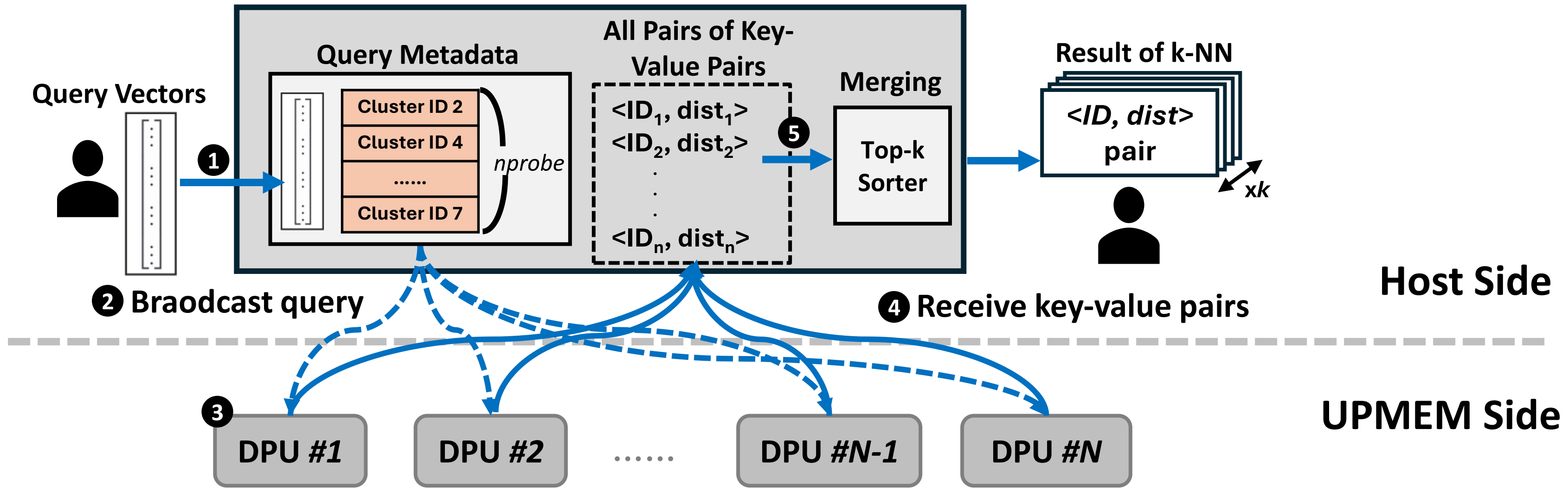
- Host-DPU collaboration
  - **Host**



(5) Host merges all <ID, dist> pairs to get final k-NN results

UPVSS – DPU-aware Vector Similarity Search Coordinator

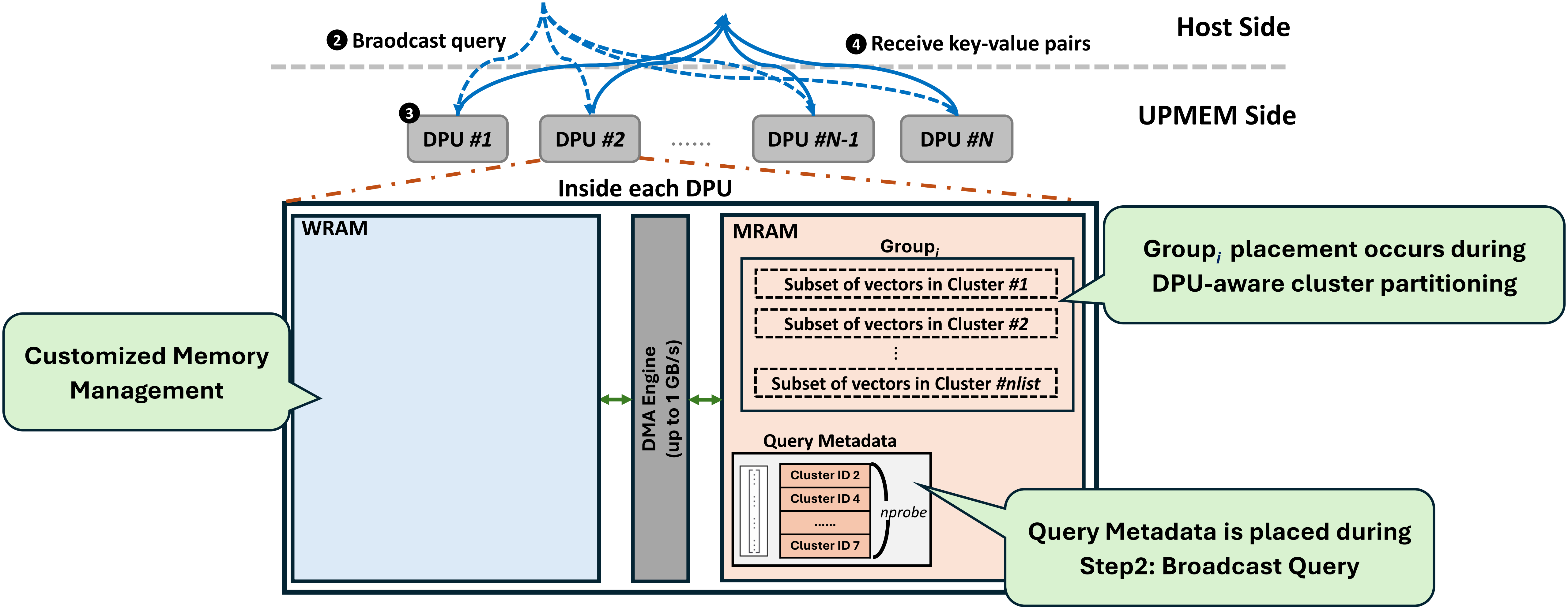
- Host-DPU collaboration
  - **Host**



Step (1) – (5) are iteratively executed for each incoming query

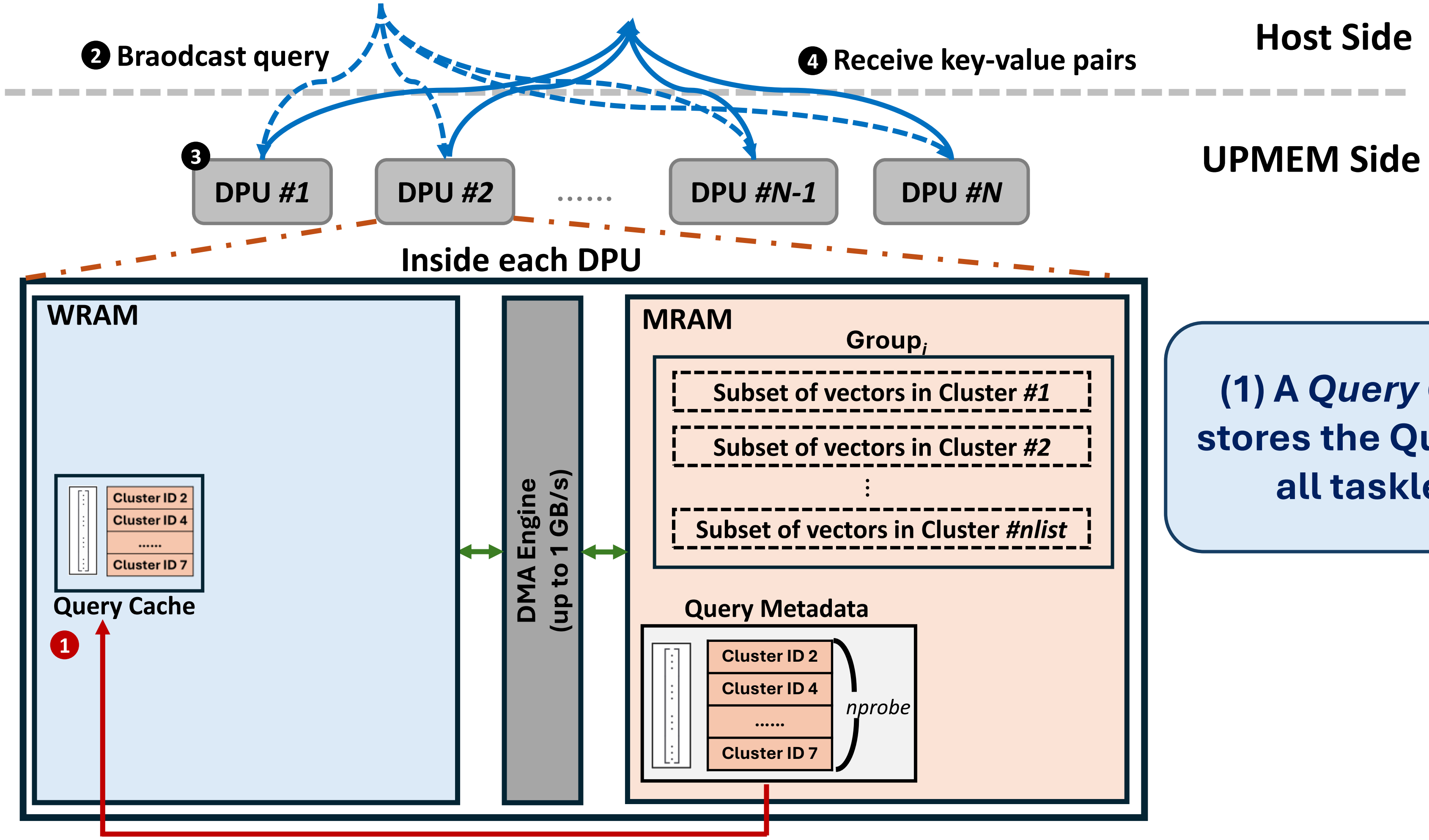
UPVSS – DPU-aware Vector Similarity Search Coordinator

- Host-DPU collaboration
  - **DPU exection**



UPVSS – DPU-aware Vector Similarity Search Coordinator

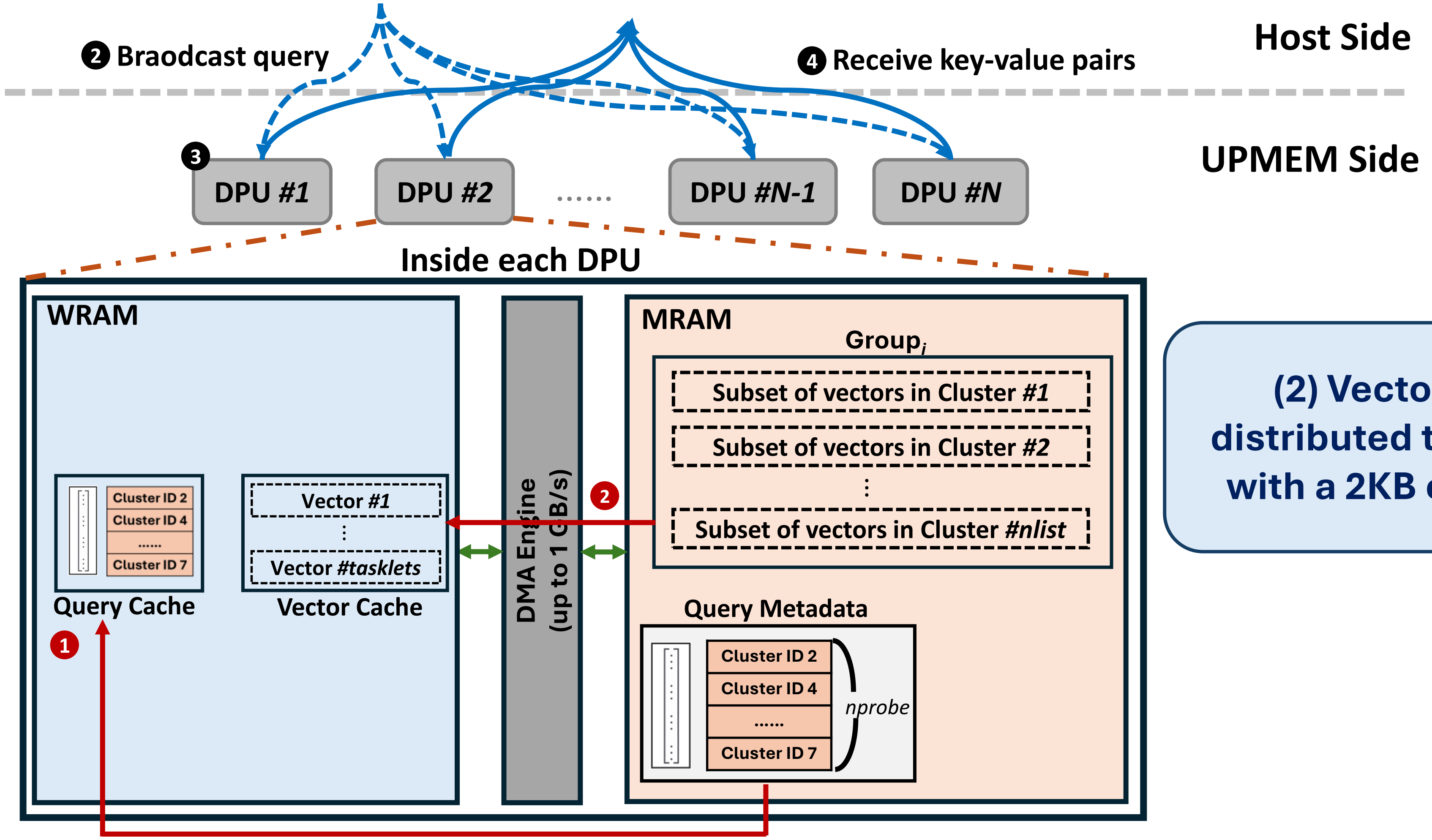
- Host-DPU collaboration
  - **DPU exection**



(1) A *Query Cache* in WRAM stores the Query Metadata for all tasklets to access

UPVSS – DPU-aware Vector Similarity Search Coordinator

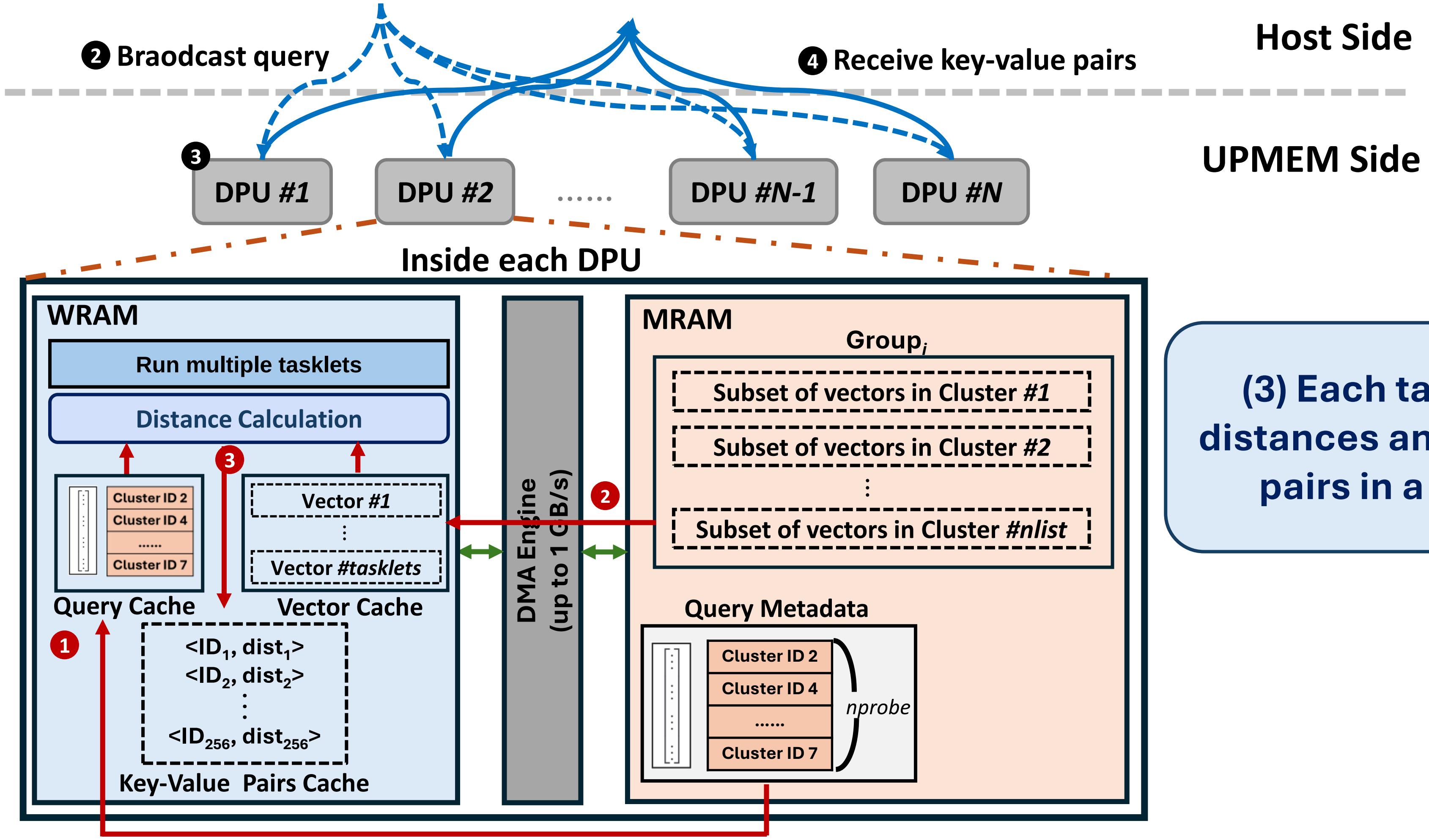
- Host-DPU collaboration
  - **DPU exection**



(2) Vectors are evenly distributed to tasklets, each with a 2KB cache in WRAM

UPVSS – DPU-aware Vector Similarity Search Coordinator

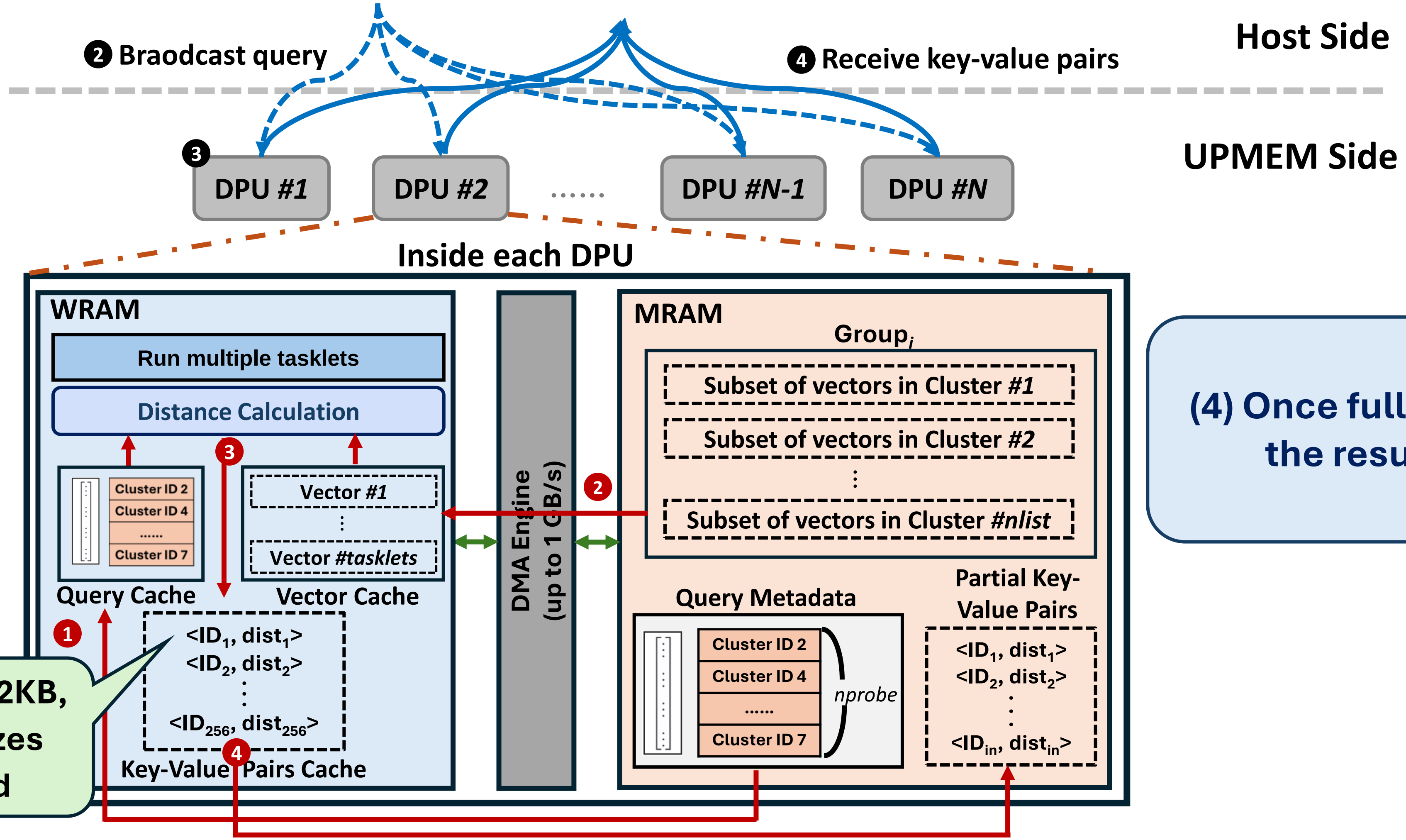
- Host-DPU collaboration
  - **DPU exection**



(3) Each tasklet compute distances and store <ID, dist> pairs in a shared cache

UPVSS – DPU-aware Vector Similarity Search Coordinator

- Host-DPU collaboration
  - **DPU exection**

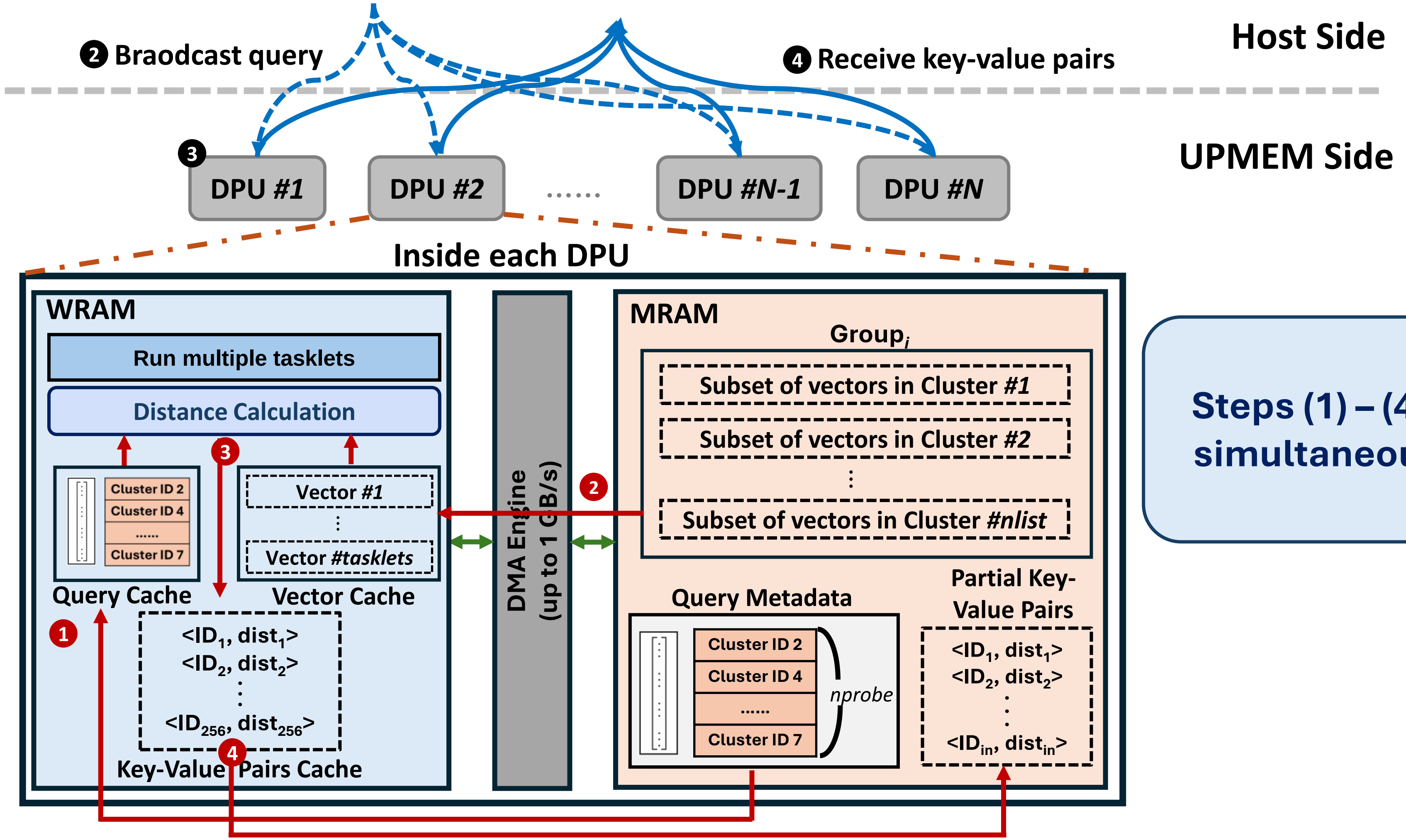


(4) Once full, a tasklet writes the results to MRAM

Each cache is set to 2KB, as this size minimizes transfer overhead

UPVSS – DPU-aware Vector Similarity Search Coordinator

- Host-DPU collaboration
  - **DPU exection**



Steps (1) – (4) are performed simultaneously on all DPUs

Evaluation: Setup

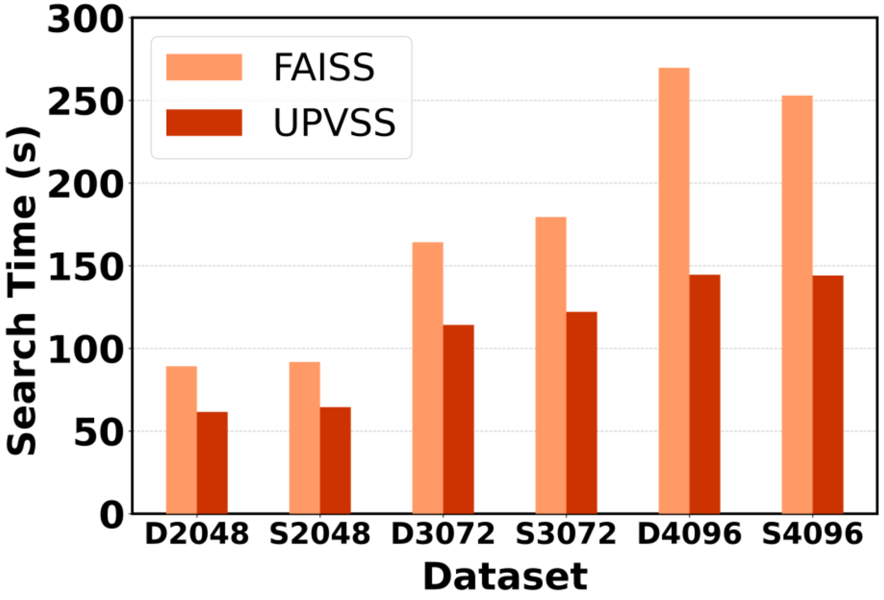
- Synthetic high-dimensional dataset
  - Dimensions: 2048, 3072, 4096
  - # Vectors: 1M
  - All vectors are quantized to unit_8

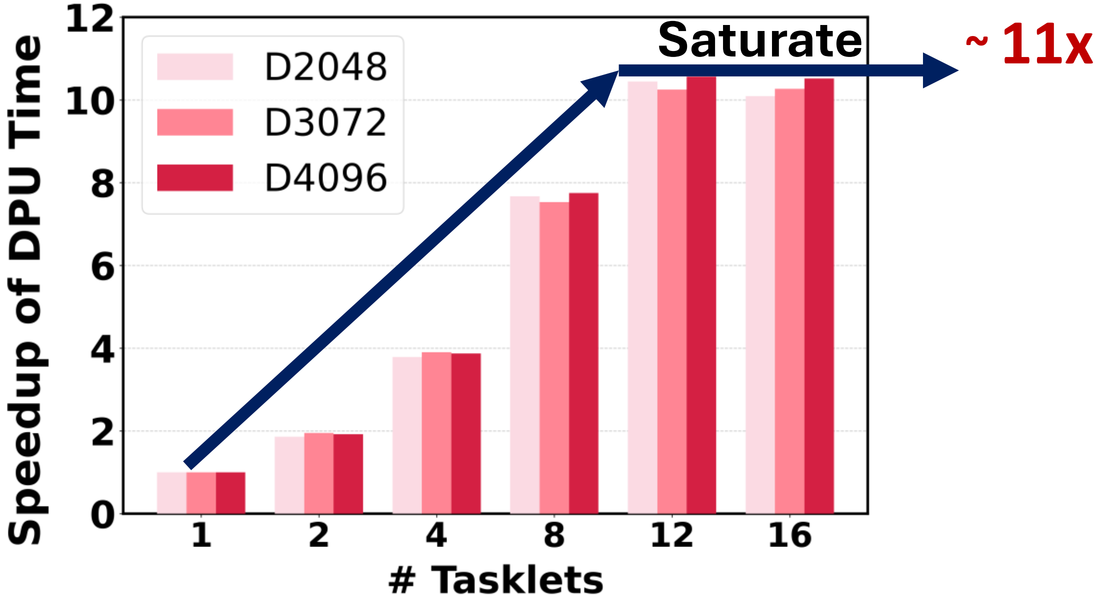
Dataset	# Dims	# Vectors	Dataset	# Dims	# Vectors
S2048	2,048	1,000,000	D2048	2,048	1,000,000
S3072	3,072	1,000,000	D3072	3,072	1,000,000
S4096	4,096	1,000,000	D4096	4,096	1,000,000

- We compare UPVSS against Meta's FAISS-IVF
  - FAISS optimized with multi-threading (OpenMP), AVX2, and BLAS library
    - Using 64 CPU threads
- Evaluated System
  - 2 host CPUs with 256GB host DRAM
  - 8 UPMEM DIMMs with 1024 DPUs (64GB)

Evaluation: Performance & Effectiveness

- Performance comparison over State-Of-The-Art
  - Performance comparison with FAISS
  - Effectiveness of our WRAM memory management



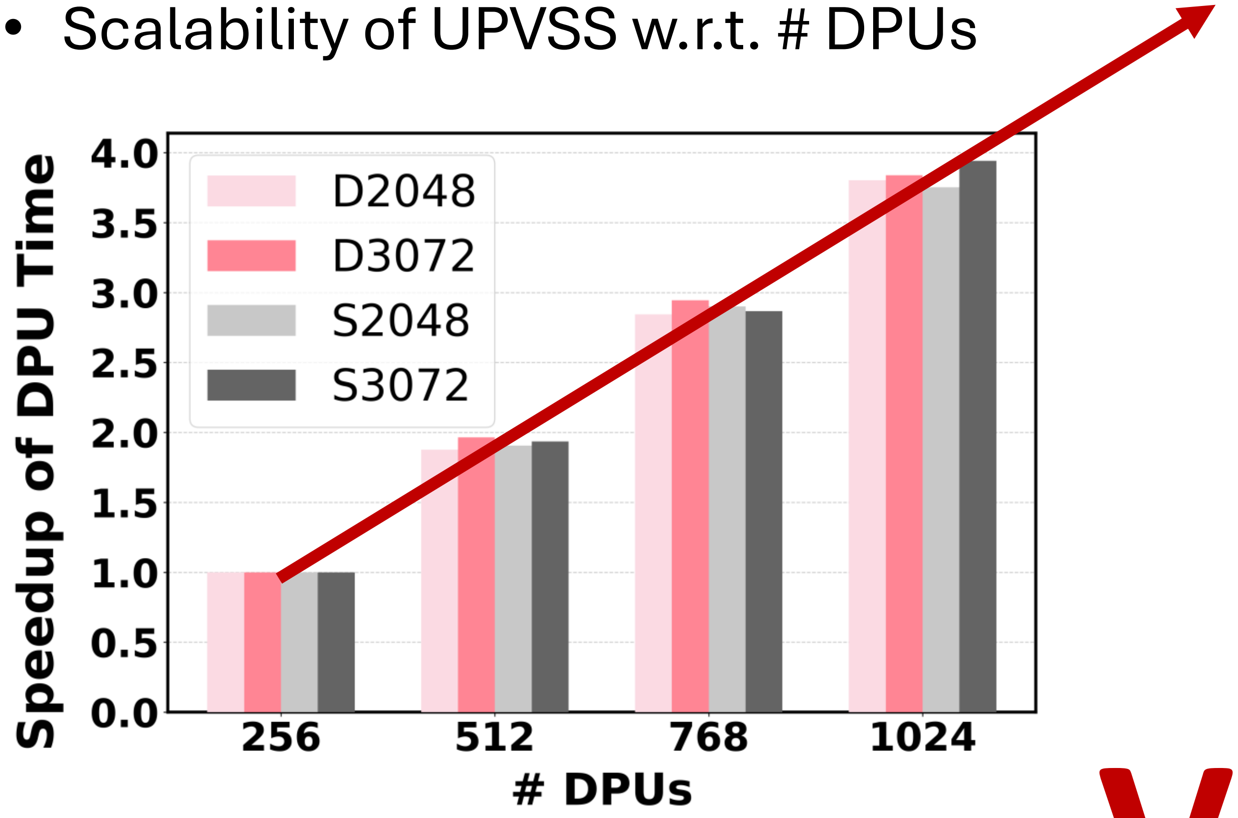


UPVSS achieves **1.42x – 1.86x speedup** over FAISS

Our WRAM management **efficiently maximizes pipeline utilization**

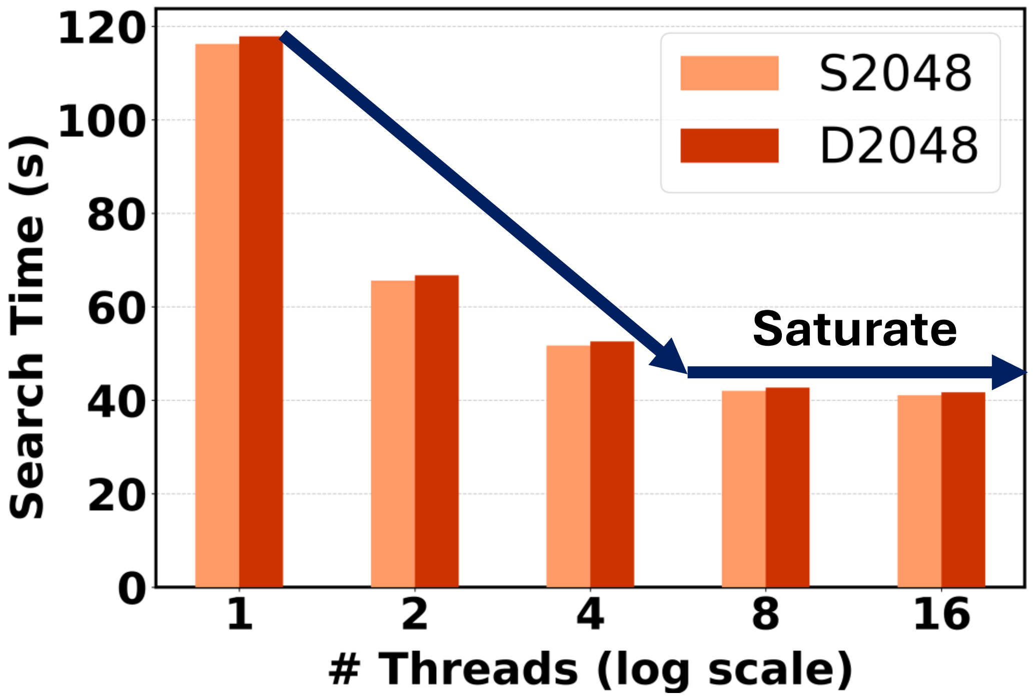
Evaluation: Scalability

- Scalability of UPVSS w.r.t. # DPUs



UPVSS using UPMEM

VS.

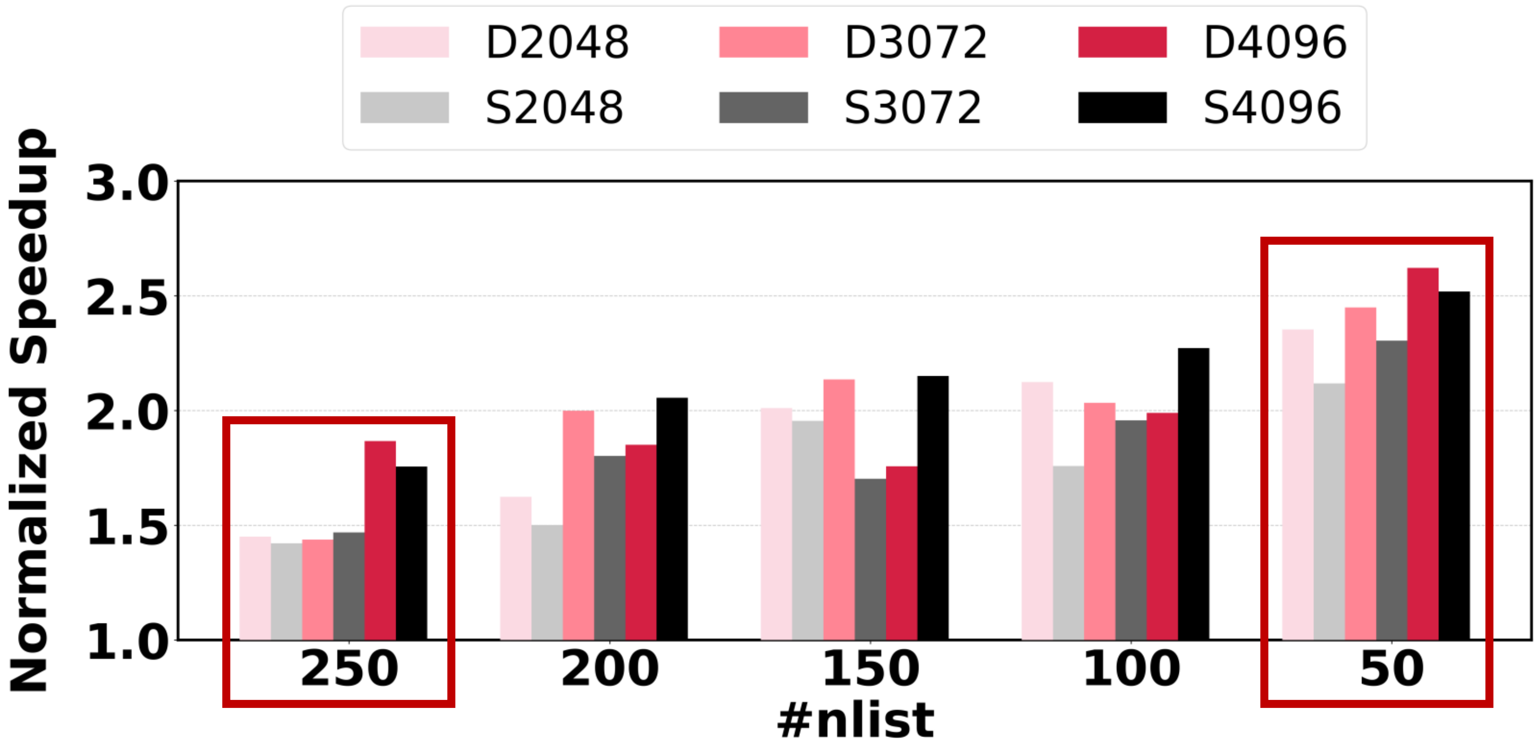


FAISS on CPU

UPVSS scales linearly with more DPUs and is not bottlenecked by bandwidth

Experiment Results

- Speedup w.r.t. to # nlist values



UPVSS achieves **average 1.95x speedup** compared to FAISS

Smaller nlist value means more vectors per cluster,
more computation per DPU, and greater speedup

Conclusion

- We reveal that **Vector Similarity Search (VSS)** is **DRAM-bounded**
  - Saturated performance and DRAM bandwidth
  - increased CPI and load latency
- **Near Memory (Data) Processing** is a promising solution
  - Alleviate the data movement bottleneck by integrating compute elements directly into memory modules
- We propose **UPVSS, a joint management framework** for VSS with real NMP – UPMEM
  - **Load-balanced** cluster partitioning
  - **Scalability** with respect to the number of DPUs and tasklets
  - **Efficient** WRAM **memory management**
- **Results:** UPVSS achieves **up to 1.95x speedup** compared to FAISS

Our Works

- Accepted by **Design Automation Conference (DAC)**, 2025

UPVSS: Jointly Managing Vector Similarity Search with Near-Memory Processing Systems

Chun-Chien Liu*, Chun-Feng Wu*, Yunho Jin†
*Department of Computer Science, National Yang Ming Chiao Tung University, Taiwan
†Department of Computer Science, Harvard University, USA
Corresponding Author: Chun-Feng Wu
E-mail: ccliu.cs12@nycu.edu.tw, cfwu417@cs.nycu.edu.tw, yjin@g.harvard.edu

Thanks for your attention

Question & Answer