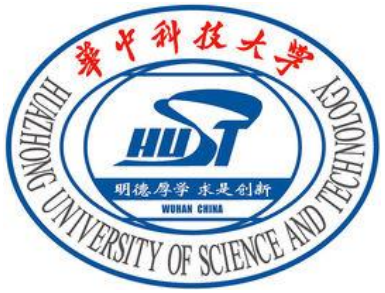


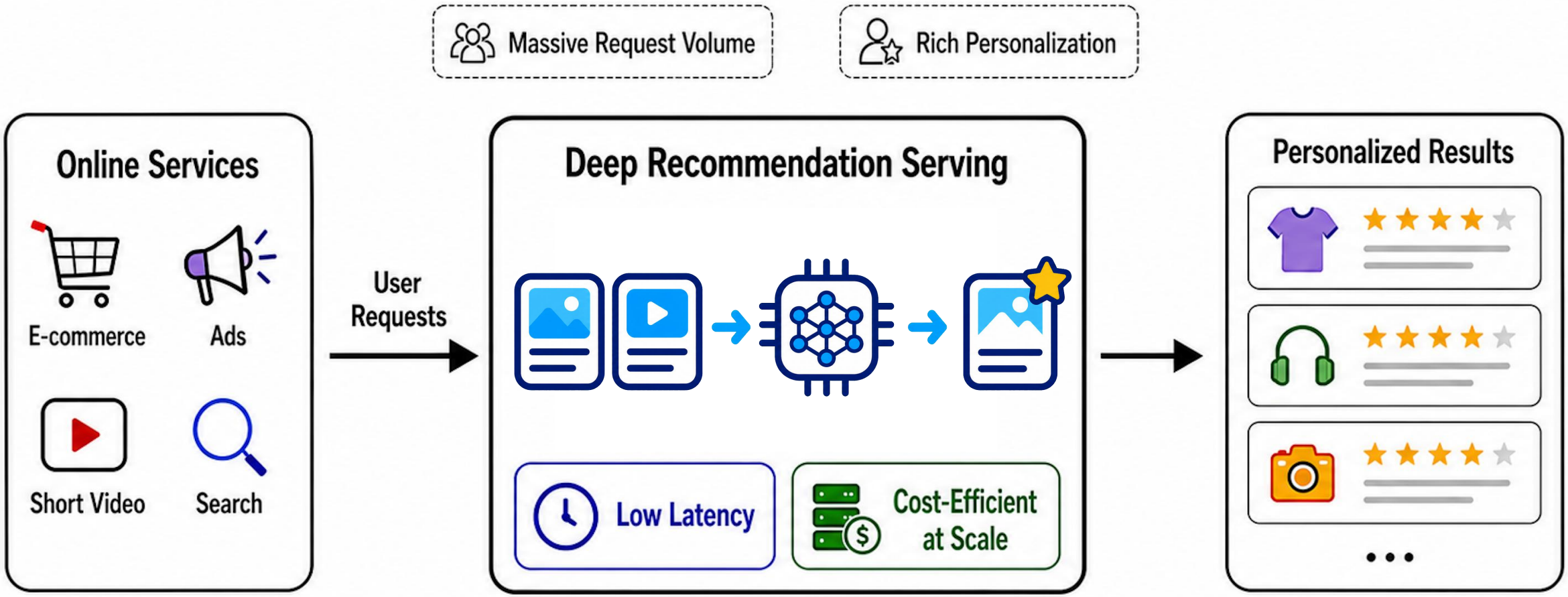
CloverRec: Cost-Efficient Disaggregated Processing-in-Memory Systems for Deep Recommendation Inference

Menglei Chen, Yu Hua, Zhijun Yang, Yixiao Wang, Xumin Chen

Huazhong University of Science and Technology, China

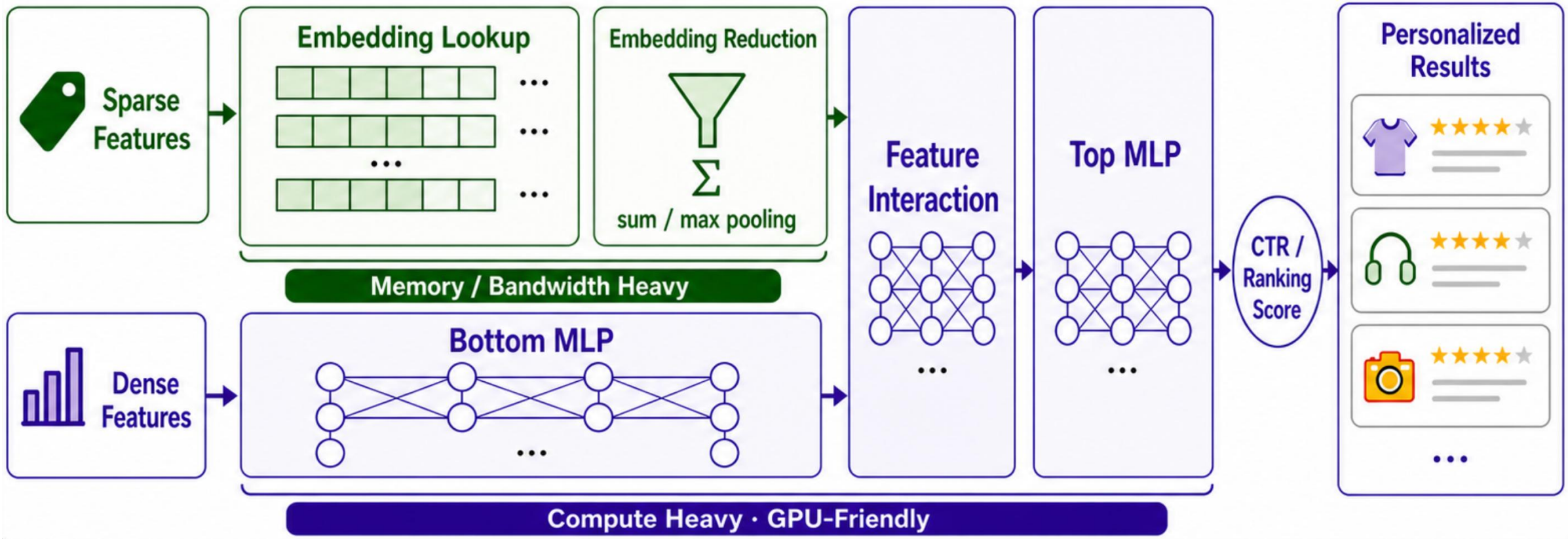
25th USENIX Conference on File and Storage Technologies (**FAST**), 2027

Deep Recommendation Serving



Recommendation serving needs high-performance and cost-efficiency at scale

Workflow of Deep Recommendation Inference



Bandwidth-heavy embeddings and compute-heavy MLPs demand different resources

Embedding Tables Drive Capacity and Cost

Why tables grow

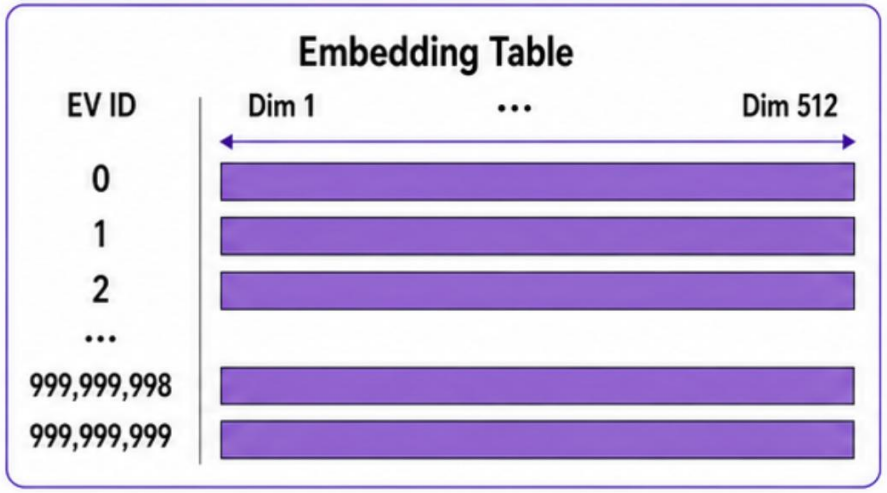
User features
IDs · history · interests

Item features
IDs · categories · products

Context features
time · device · location

More entities + richer sparse features →

A single embedding table can be enormous

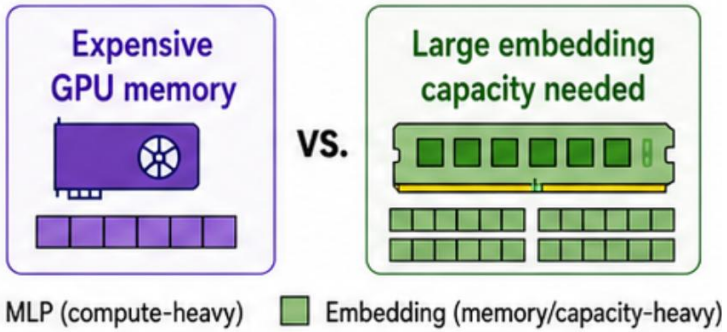


1B EVs × 512 dims × FP32 ≈ 2 TB
for one EV table

GB — TB — dozens of TB

Why this is a systems problem

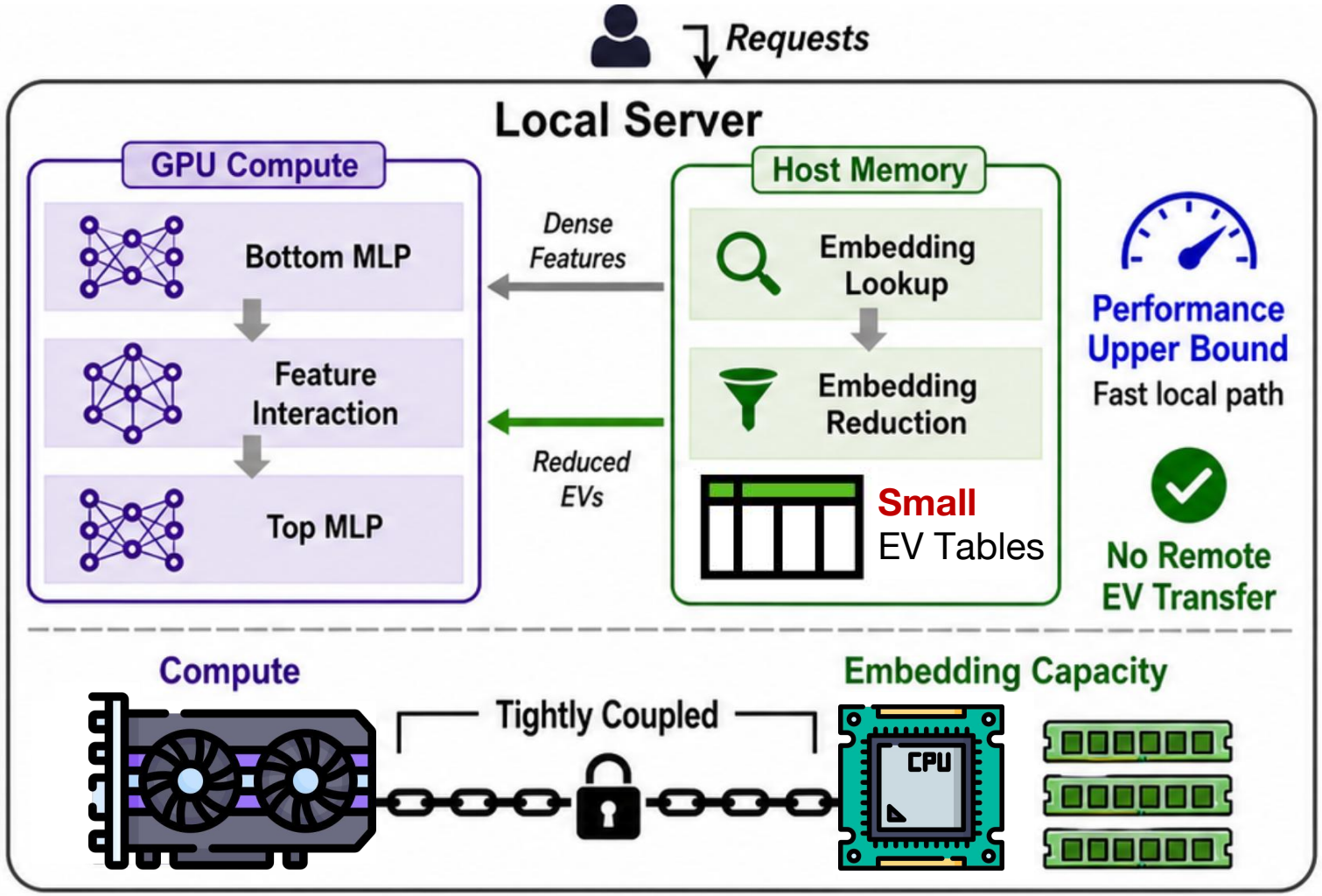
1. Hard to fit on a single machine
2. Using GPU memory for huge EV tables is expensive
3. Capacity demand grows faster than compute demand



Capacity-heavy
Embedding tables dominate memory footprint

Cost-sensitive
Provisioning large tables efficiently is crucial

Local Embedding: Fast But Not Cost-Efficient at Scale



Why It Does Not Scale

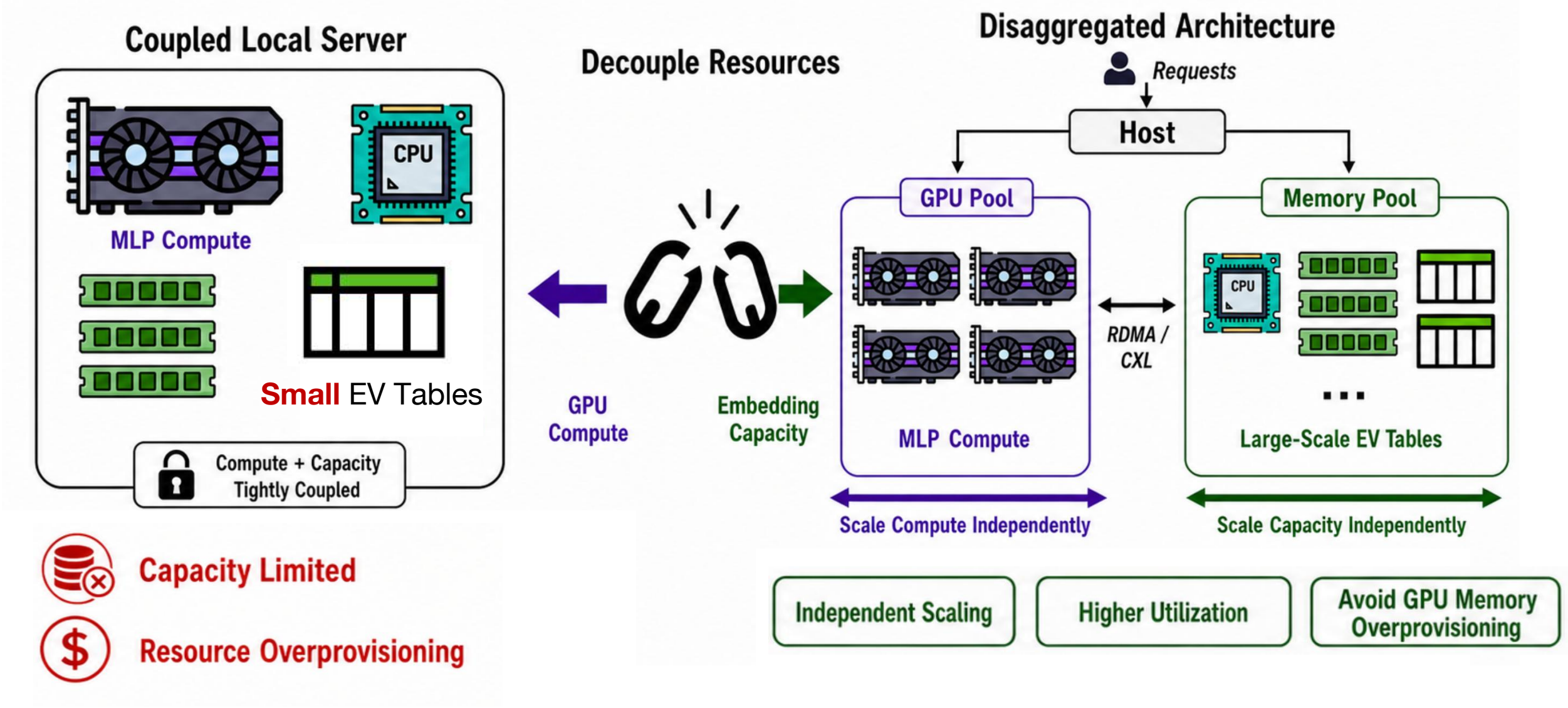
Capacity Limited
Huge tables exceed local memory

Resource Overprovisioning
Pay for compute to obtain memory

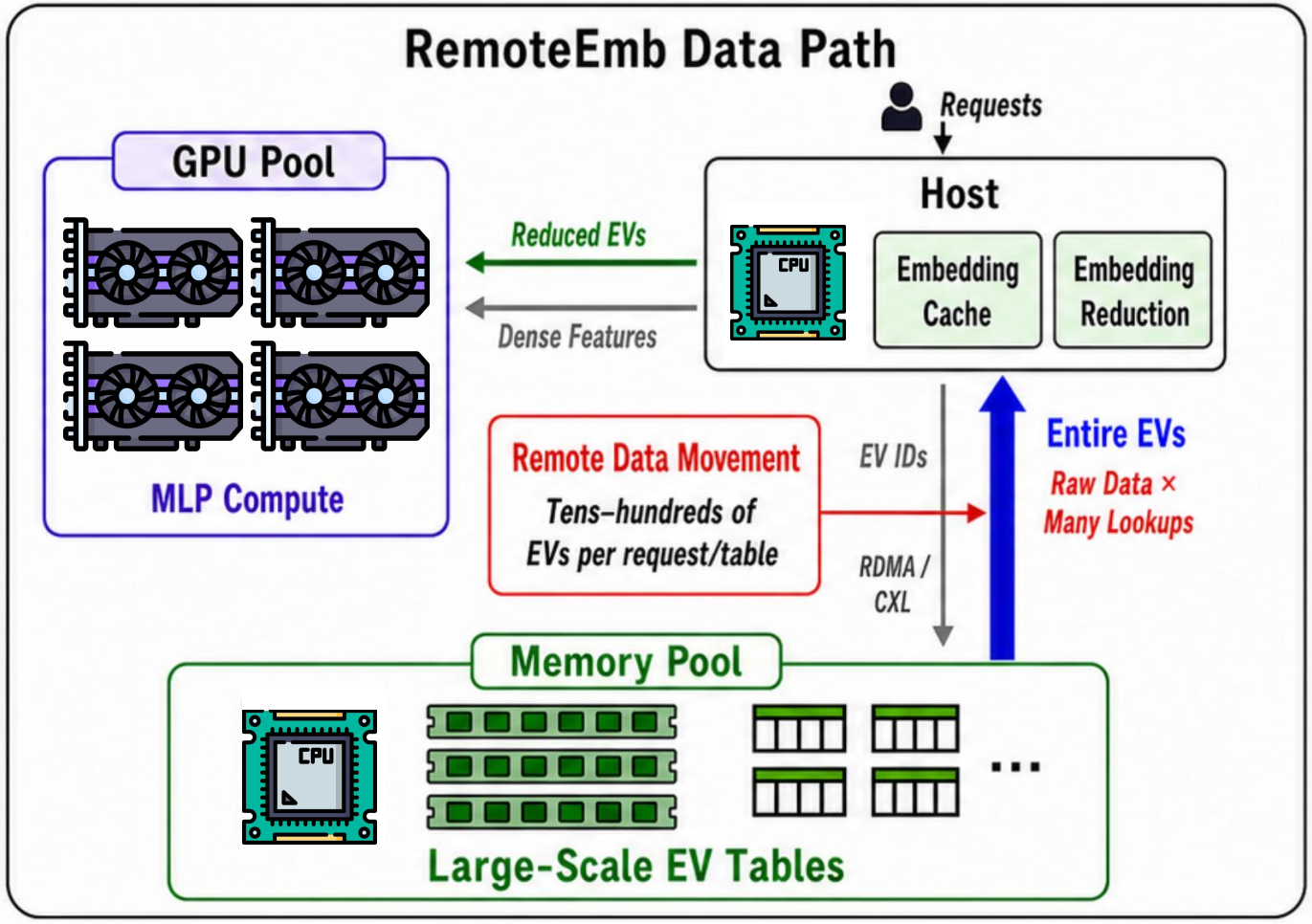
Compute–Memory Mismatch
Cannot scale resources independently

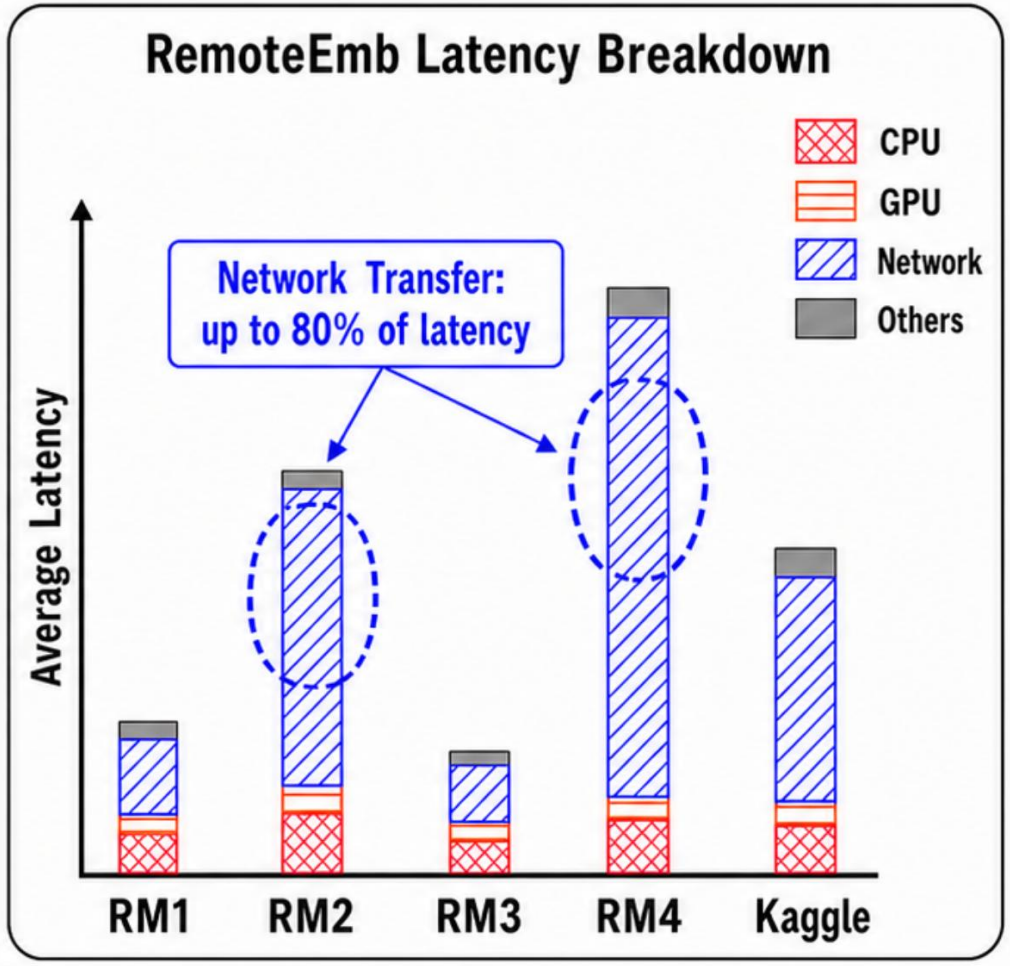
Fast local path, but not a scalable, cost-efficient serving architecture for huge EV tables

Remote Embedding: Scale Compute and Memory Cost-Efficiently



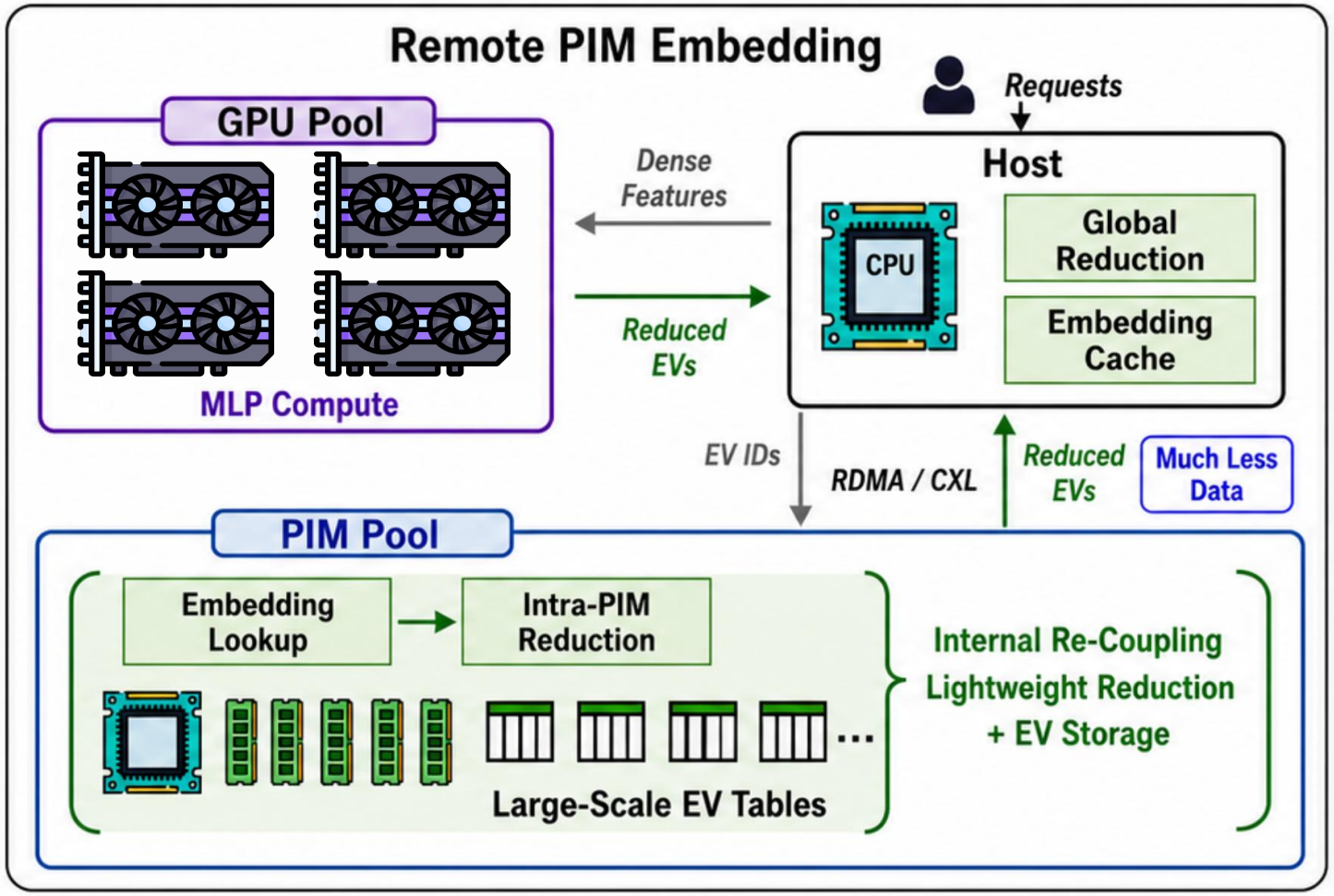
The Price of Disaggregation: Too Many Data Movements

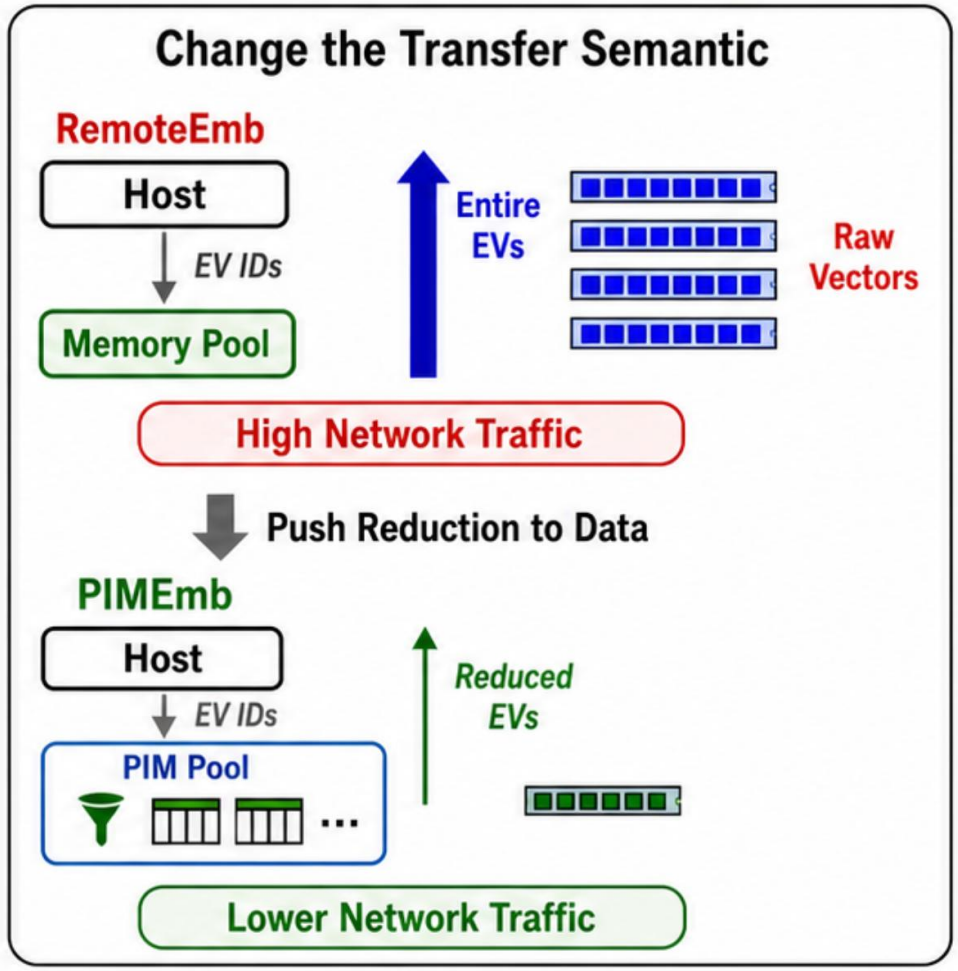




Cost-Efficient Capacity → High Network Tax

PIM Opportunity: High-Performance Disaggregation

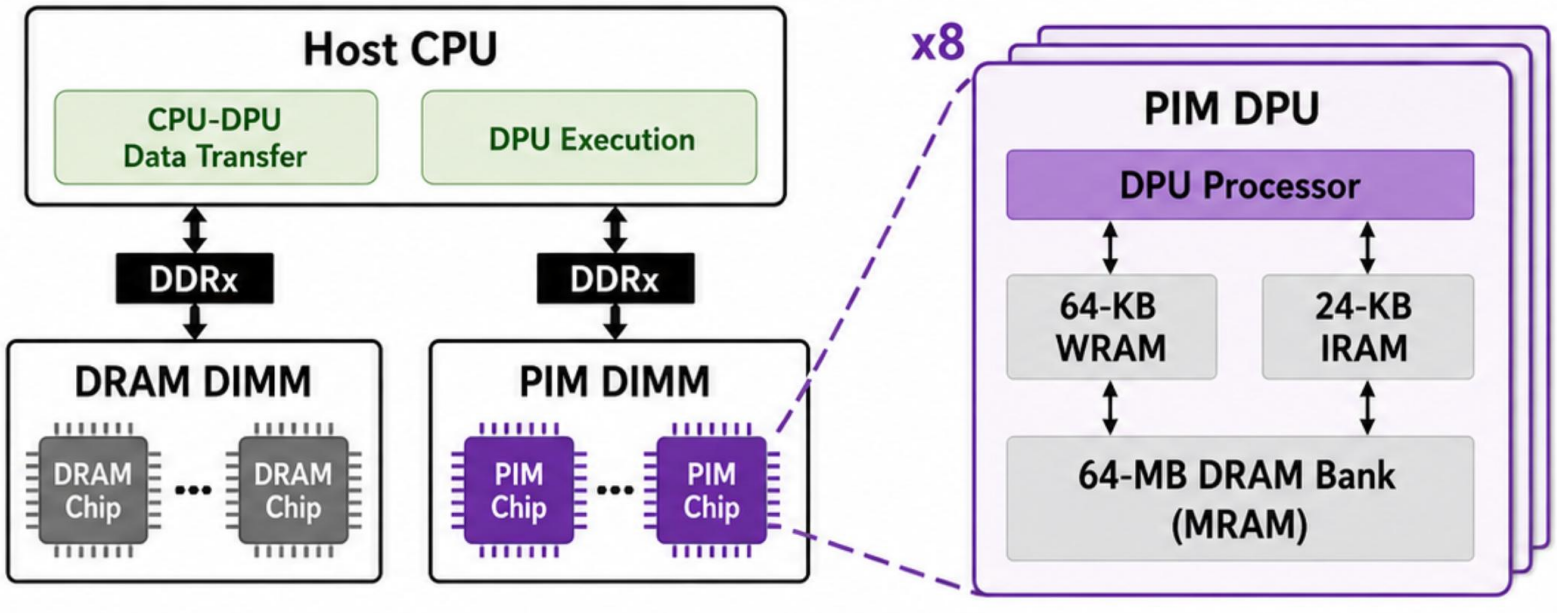




Recover performance while preserving cost-efficient disaggregation

Real PIM Hardware: UPMEM System

CloverRec's design is grounded in the hierarchy and constraints of **real UPMEM PIM hardware**



What Matters for System Design

Lightweight Thread Parallelism

24 tasklets per DPU; costly synchronization

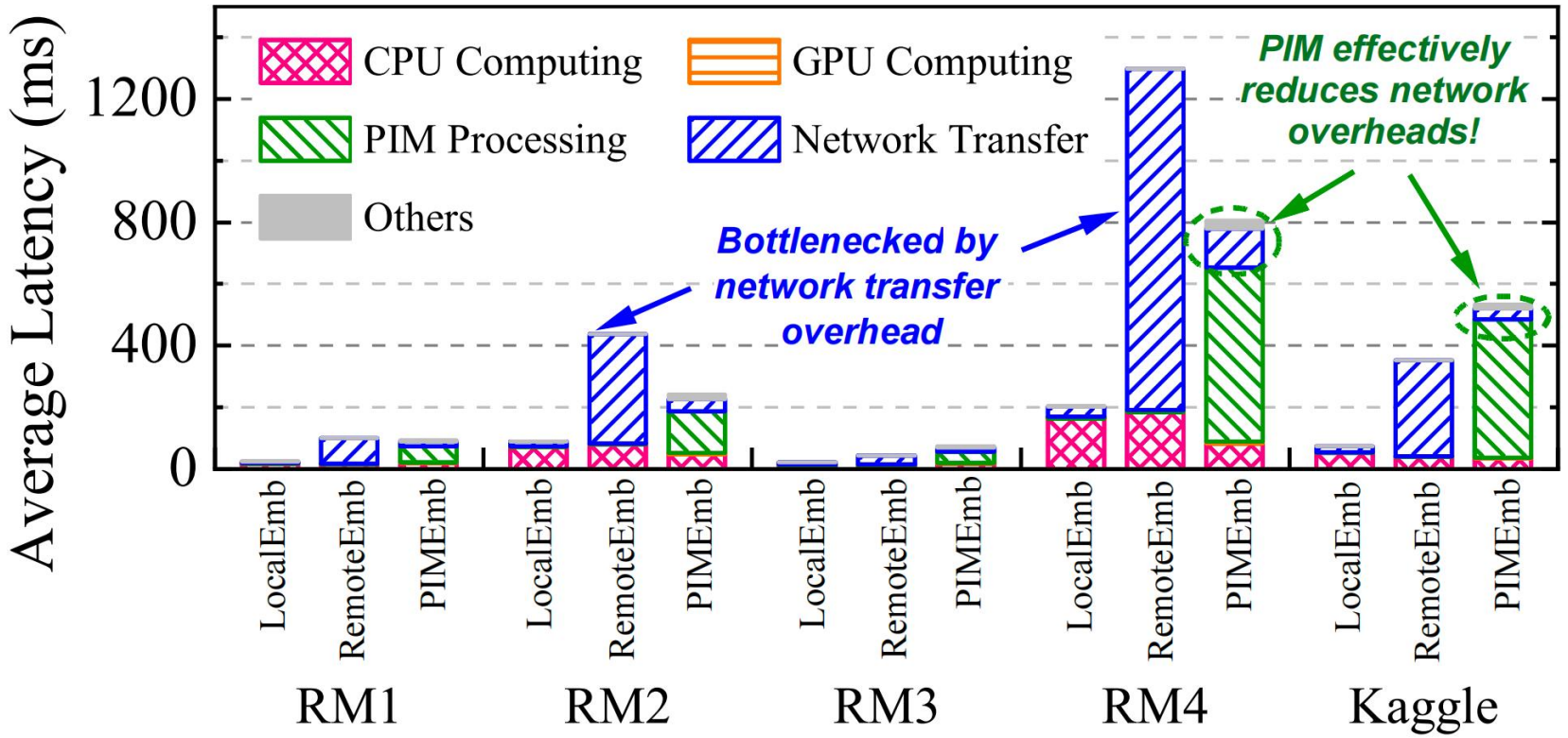
Distributed Data-Local DPUs

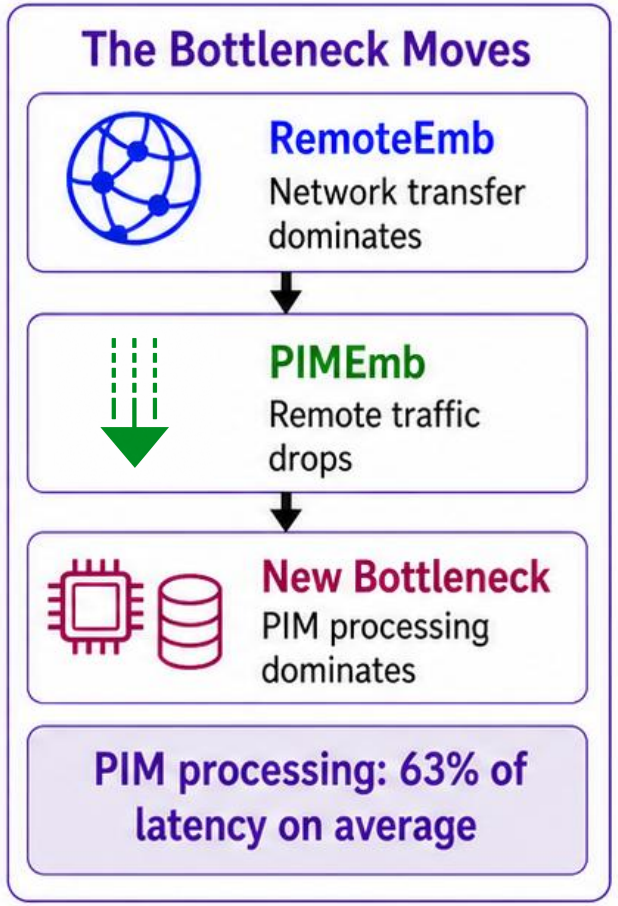
Skewed accesses create hot-DPU bottlenecks

Constrained Bulk Transfers

Equal-size buffers create a bandwidth-padding trade-off

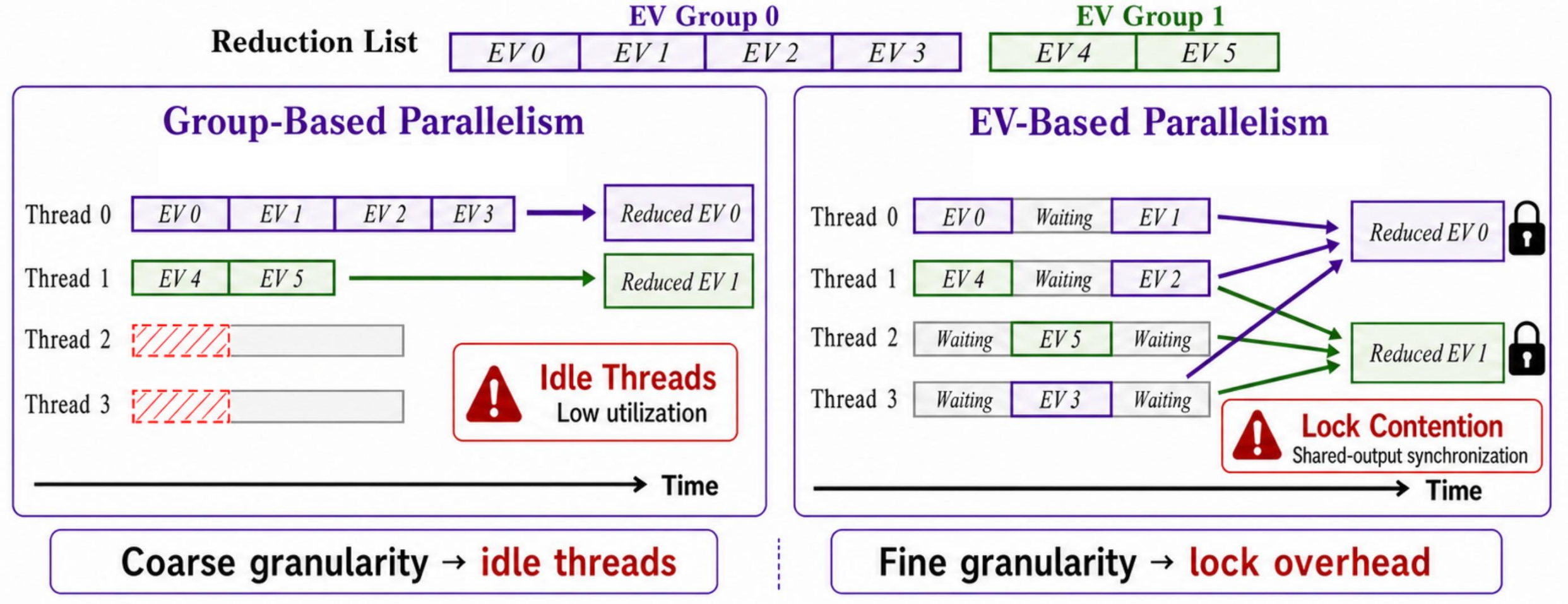
Naive PIM Embedding Is Not Automatically Cost-Efficient and Fast





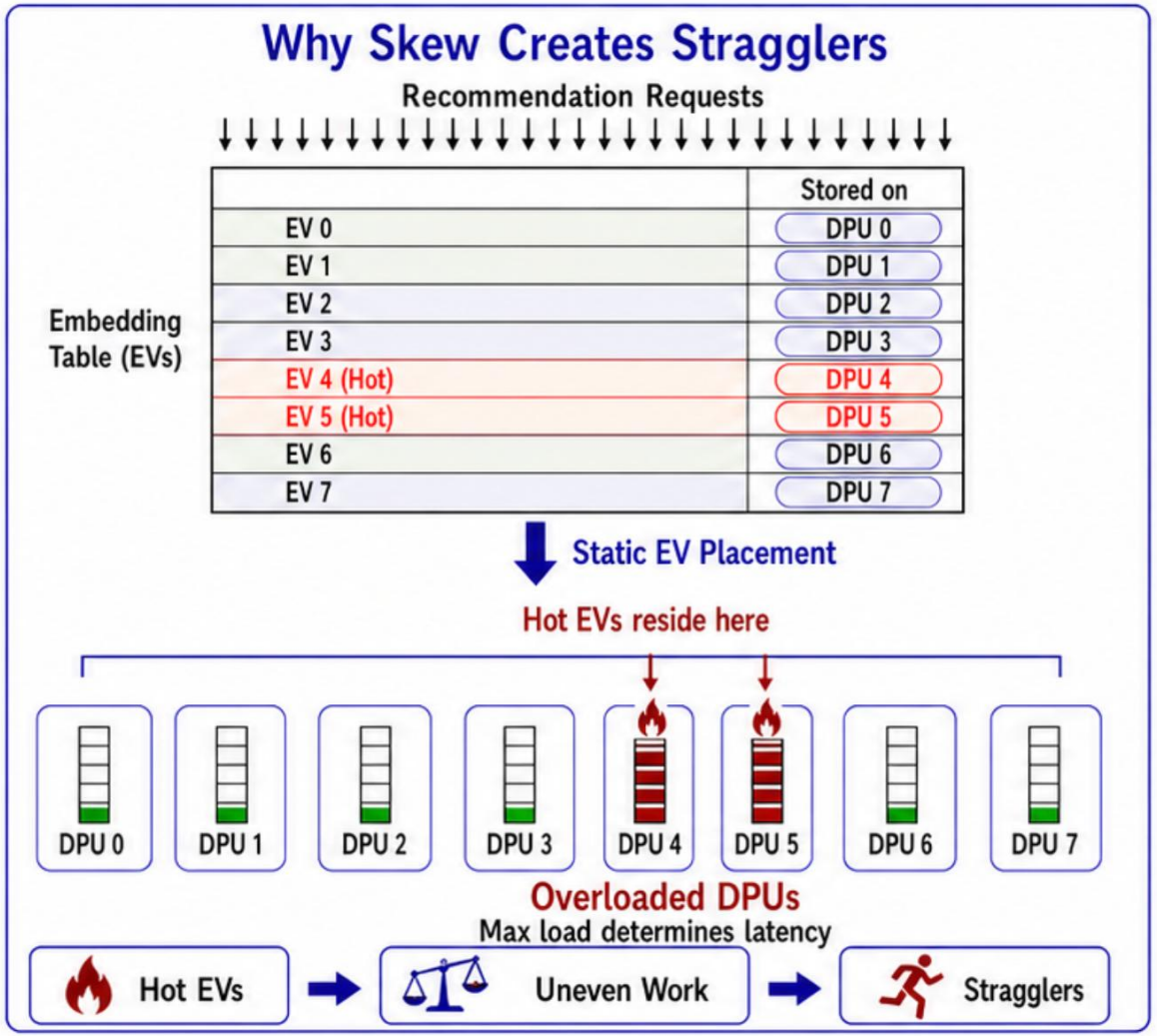
PIM reduces remote data movement, but inefficient PIM execution becomes the next bottleneck

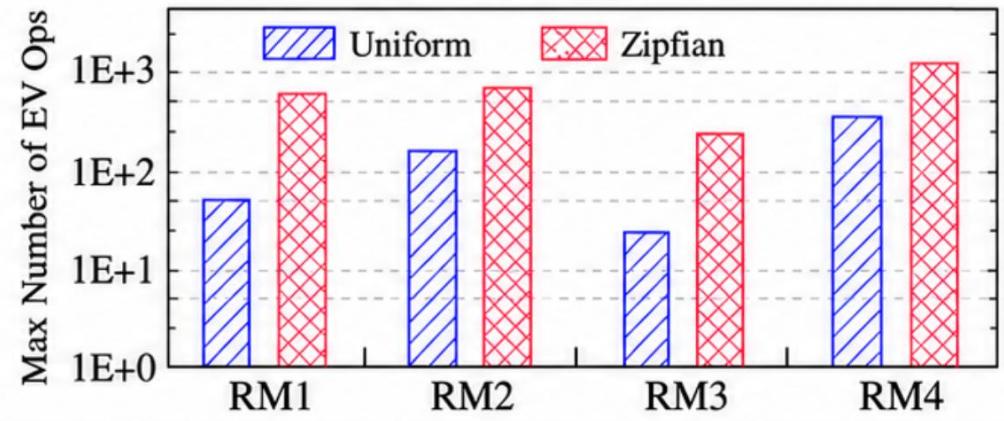
Challenge 1: Inefficient Intra-DPU Parallelism

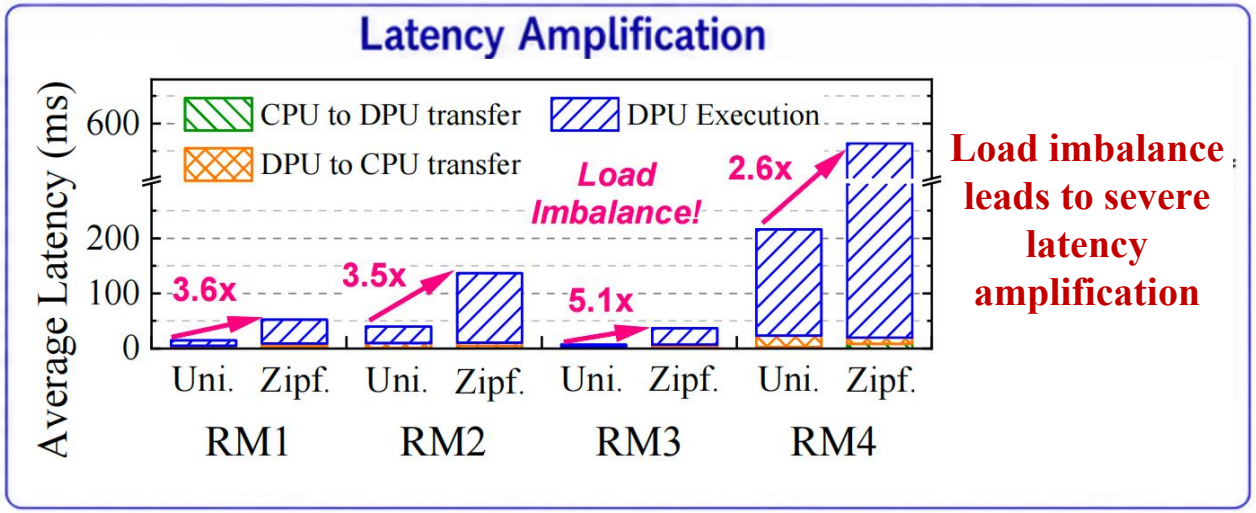


Intra-DPU Parallelism is either underutilized or synchronization-bound

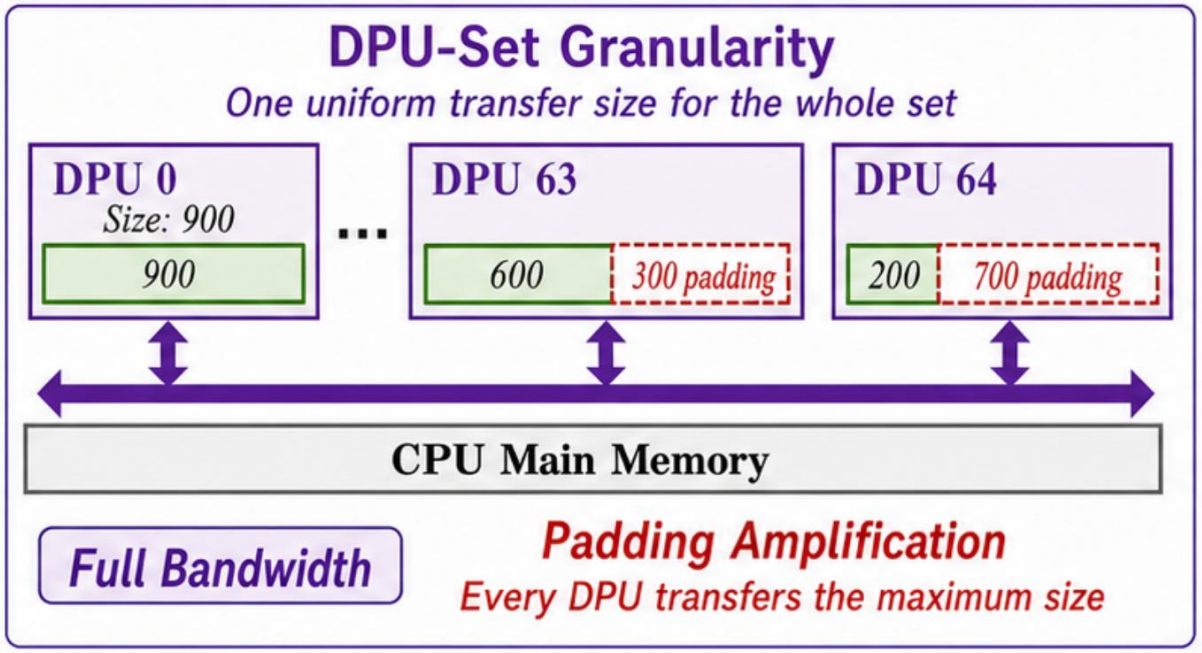
Challenge 2: Severe Inter-DPU Load Imbalance

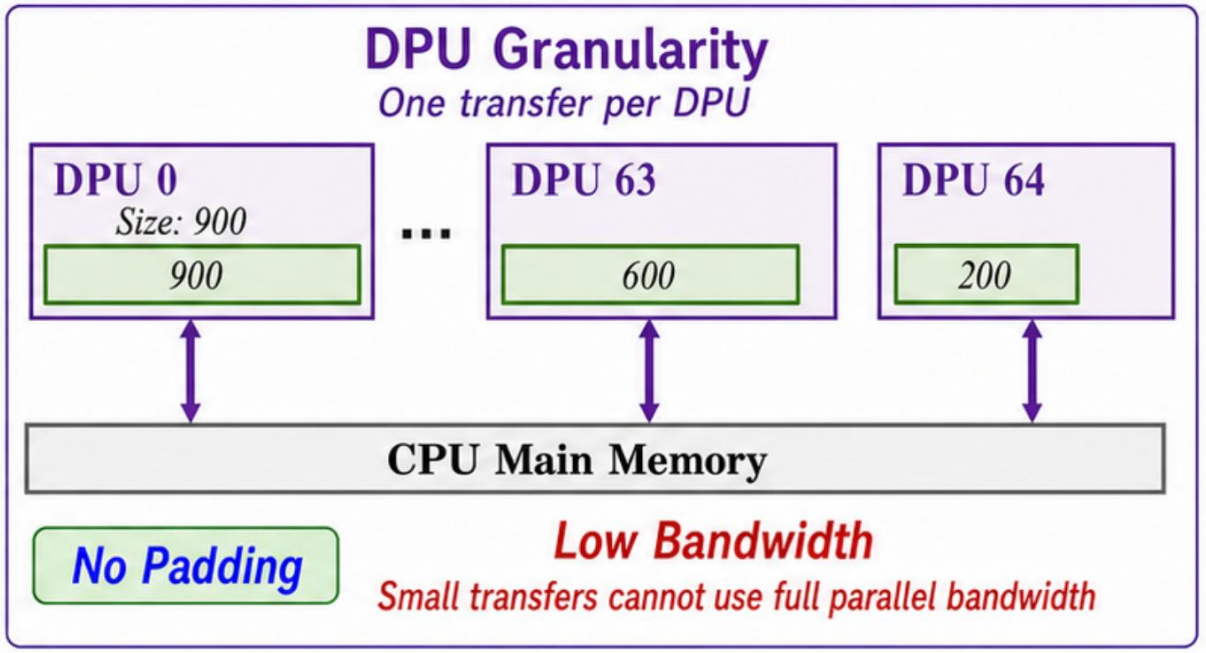


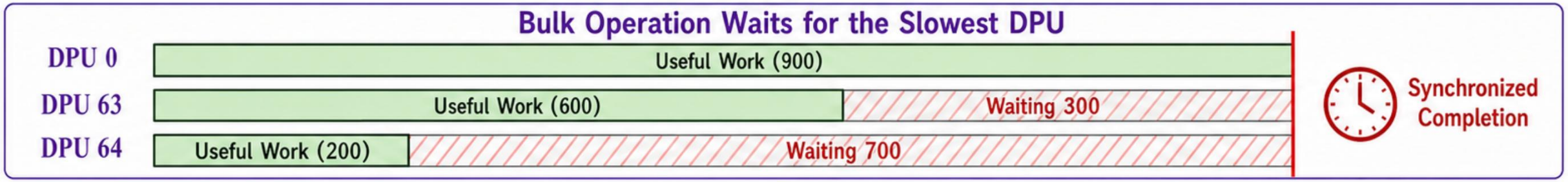




Challenge 3: Granularity Mismatch in Host-DPU Transfer





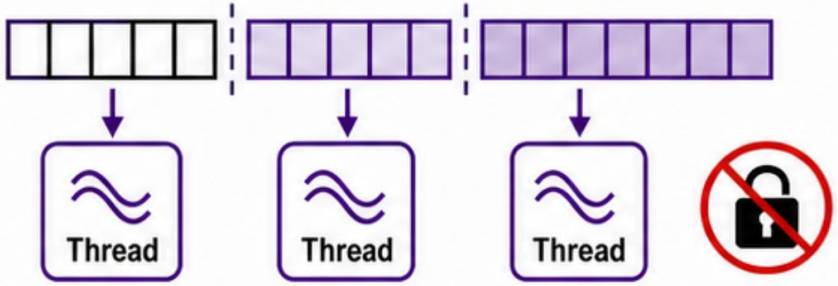


Set-level transfer → *padding waste*

Per-DPU transfer → *bandwidth loss*

Design Principles for Real PIM Devices

1

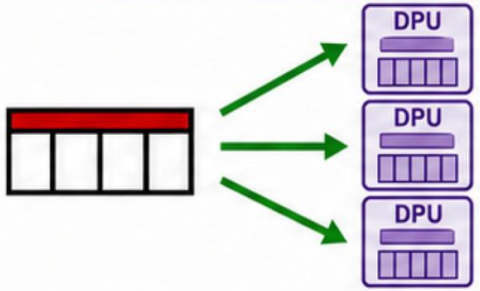


Fine-Grained Partitioning

Split vectors into independent dimension slices

Lock-Free Parallelism

2

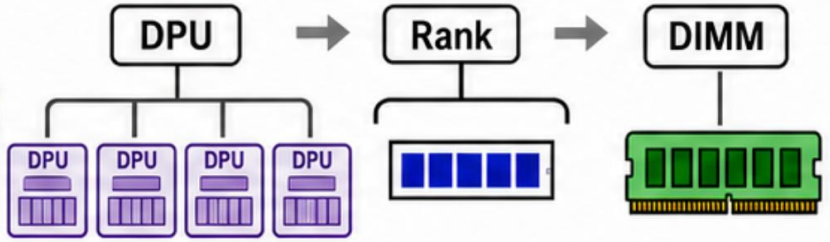


Adaptive Data Placement

Move and replicate hot EVs using observed access frequency

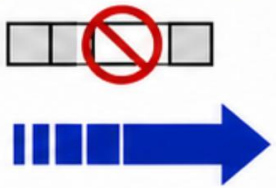
Dynamic Load Balance

3



Hierarchy-Aligned Transfers

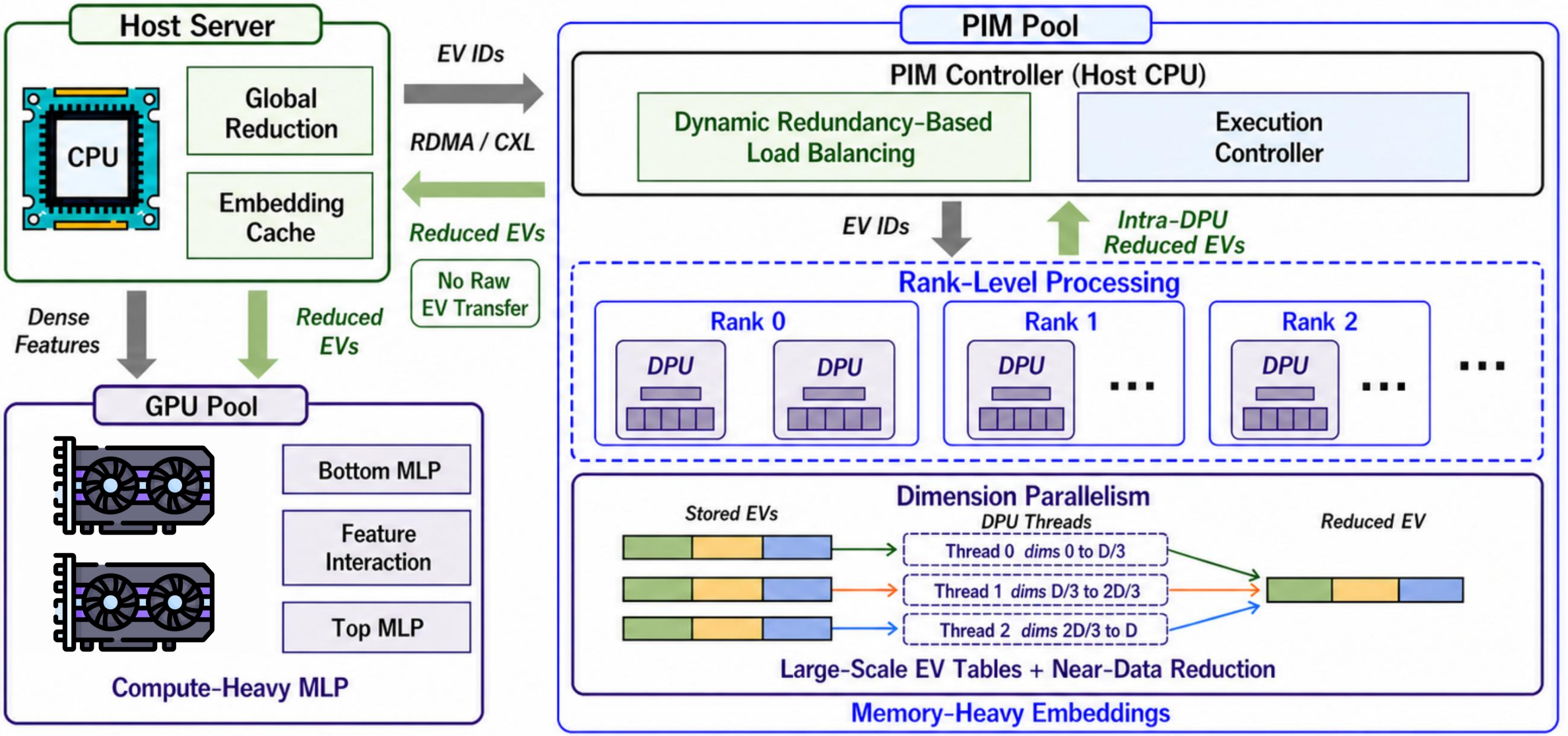
Match transfer granularity to the DPU → Rank → DIMM hierarchy



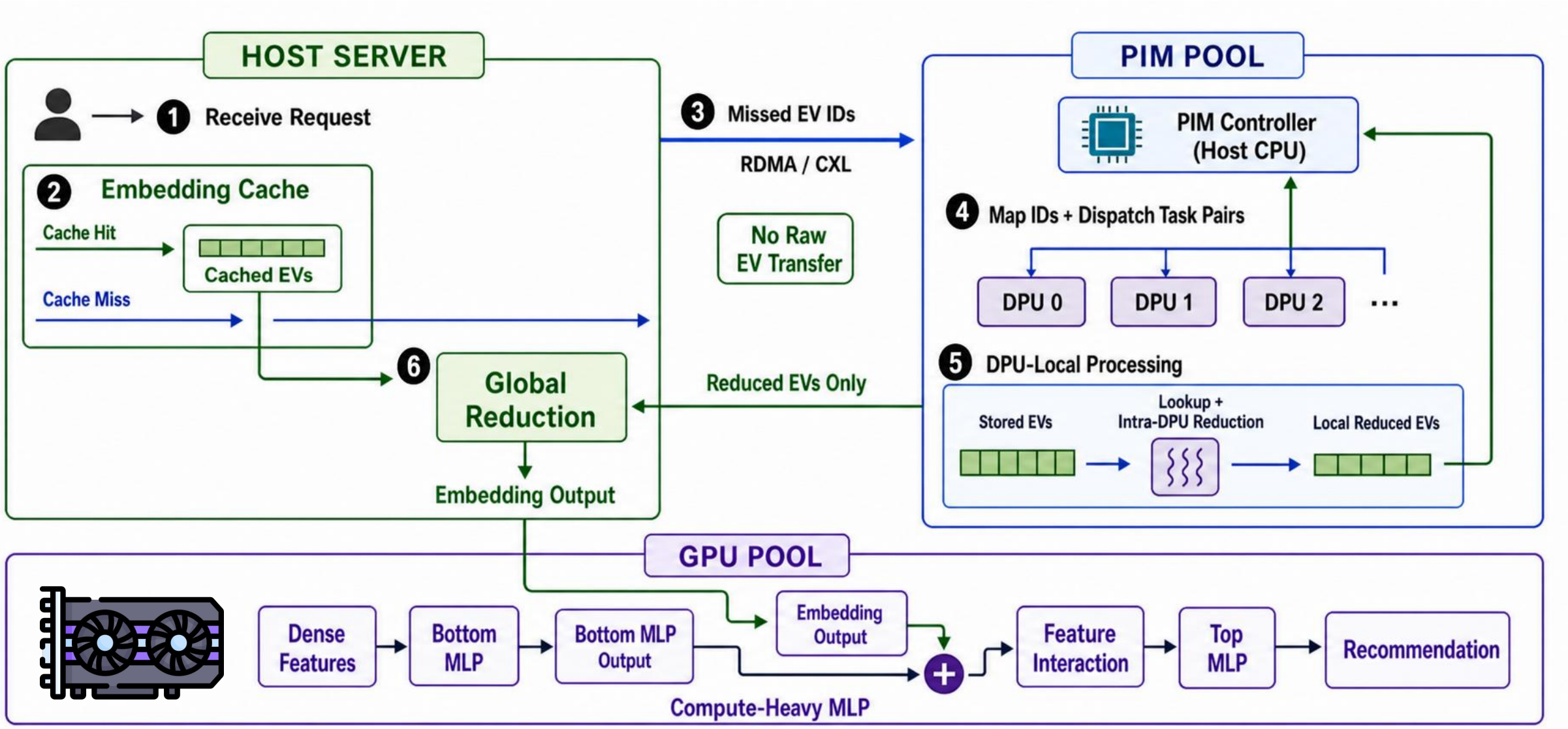
Less Padding

Full Bandwidth

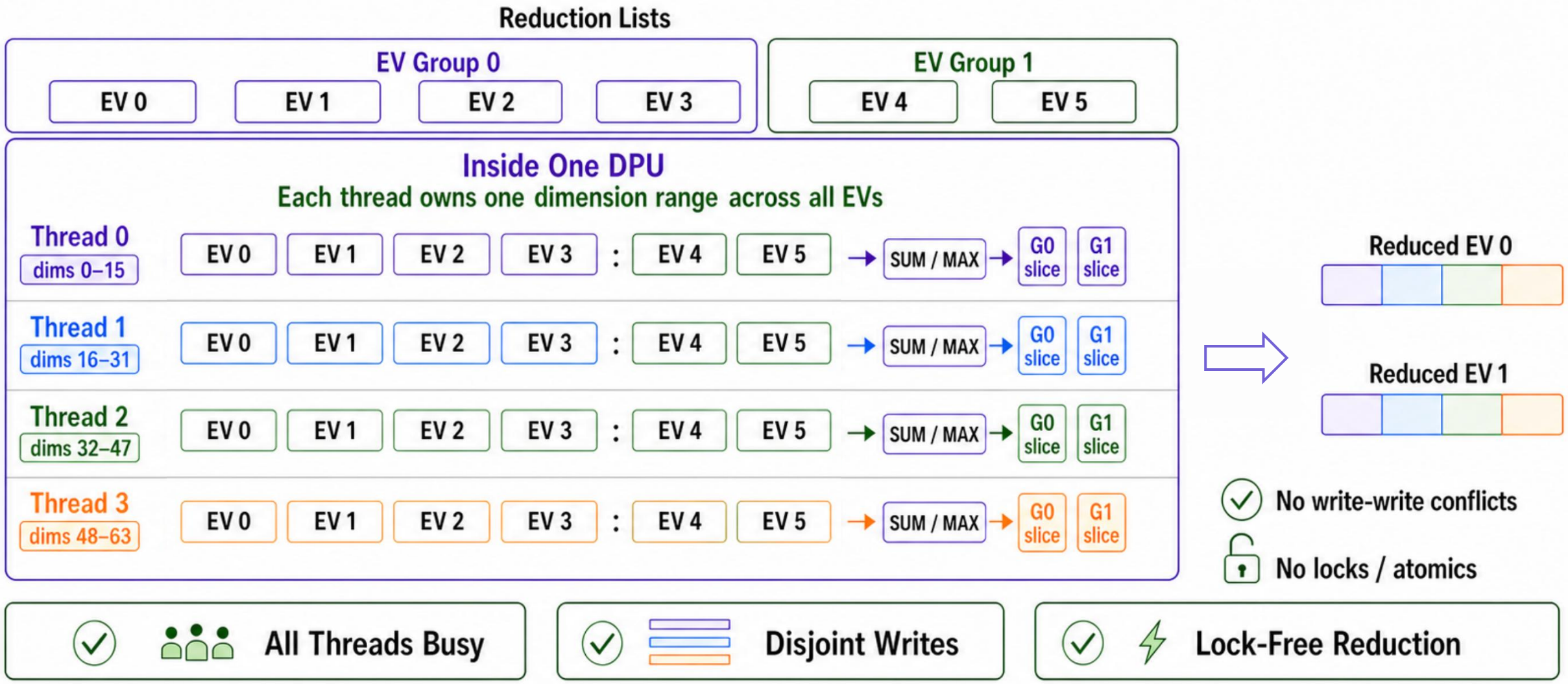
CloverRec Overview



CloverRec Inference Flow



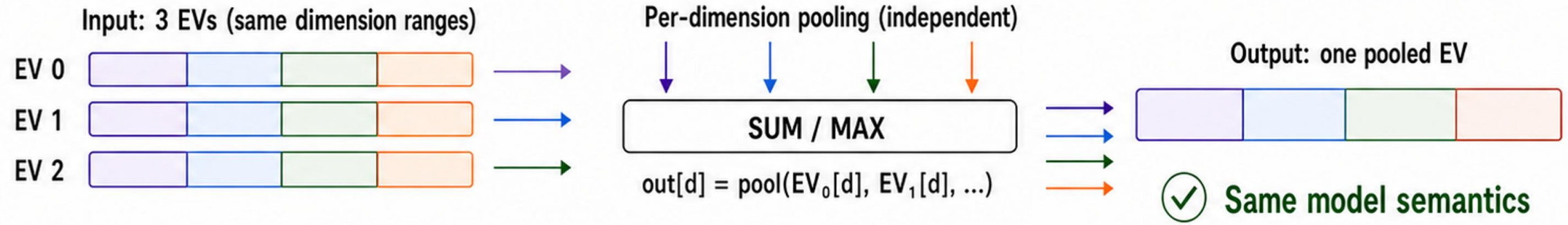
Design 1: Dimension Parallelism



Dimension Parallelism Is Broadly Applicable

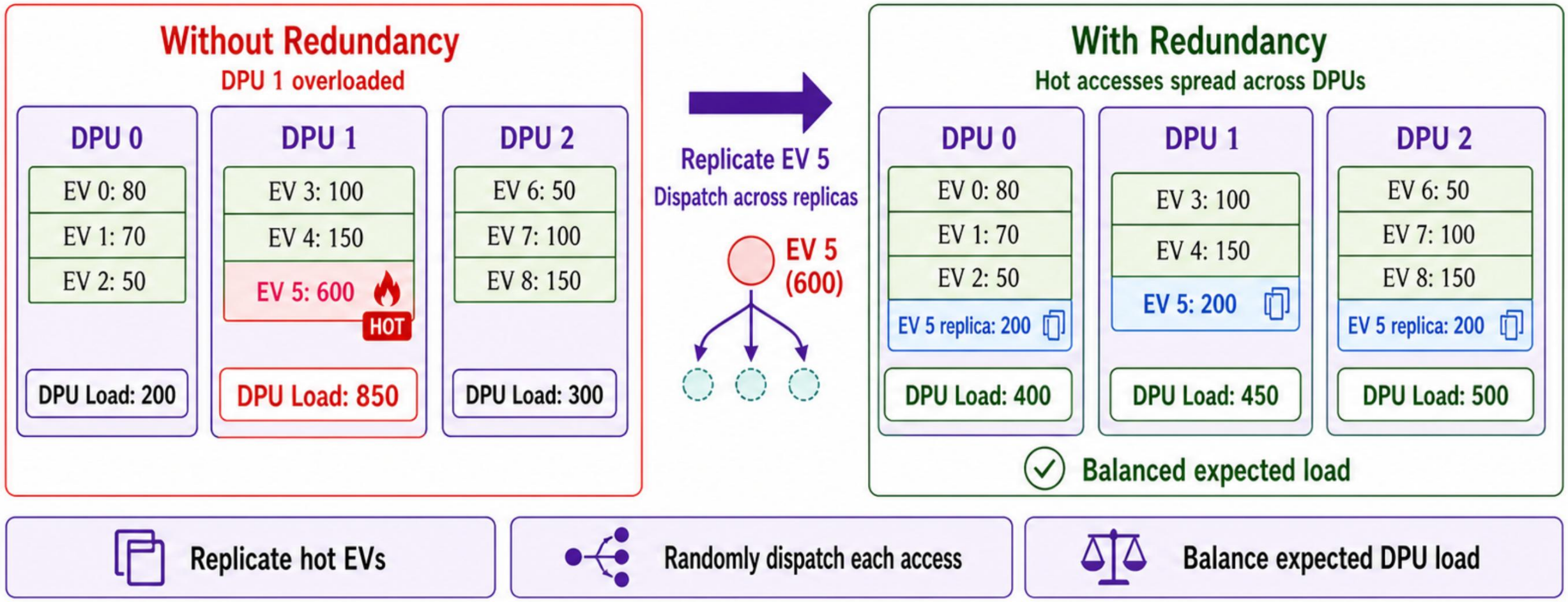
1 Enough Dimensions for Parallel Threads

2 Pooling Is Independent Across Dimensions

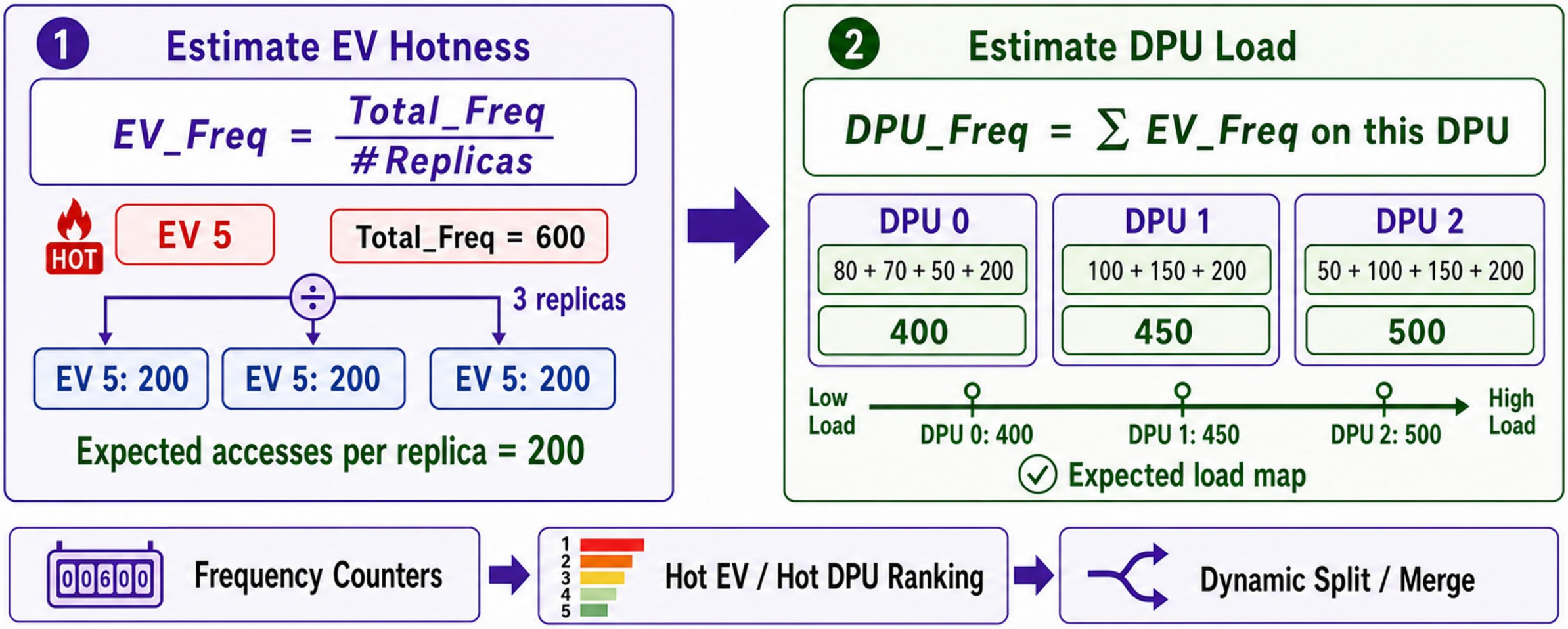


High-dimensional EVs • Per-dimension pooling • No model change

Design 2: Dynamic Redundancy-Based Load Balancing

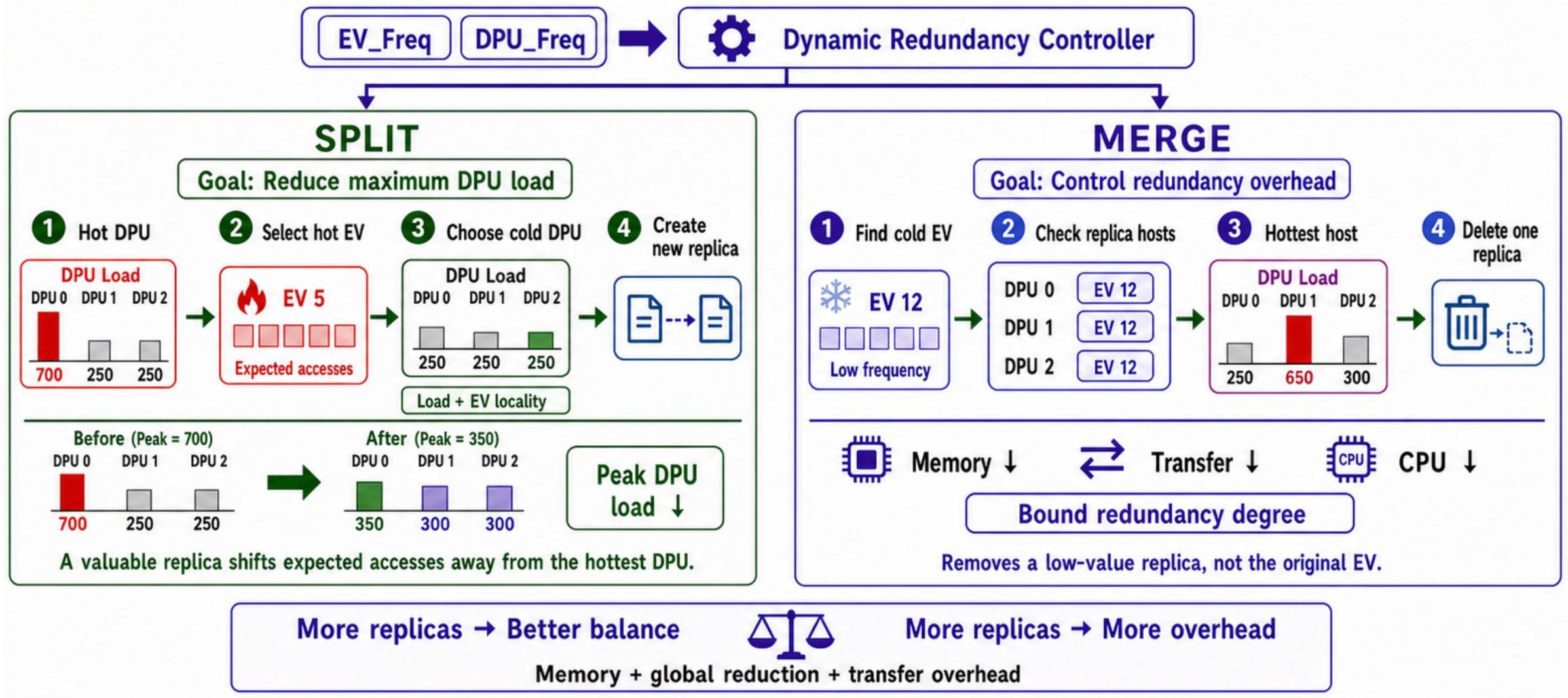


Tracking Hotness and Expected DPU Load

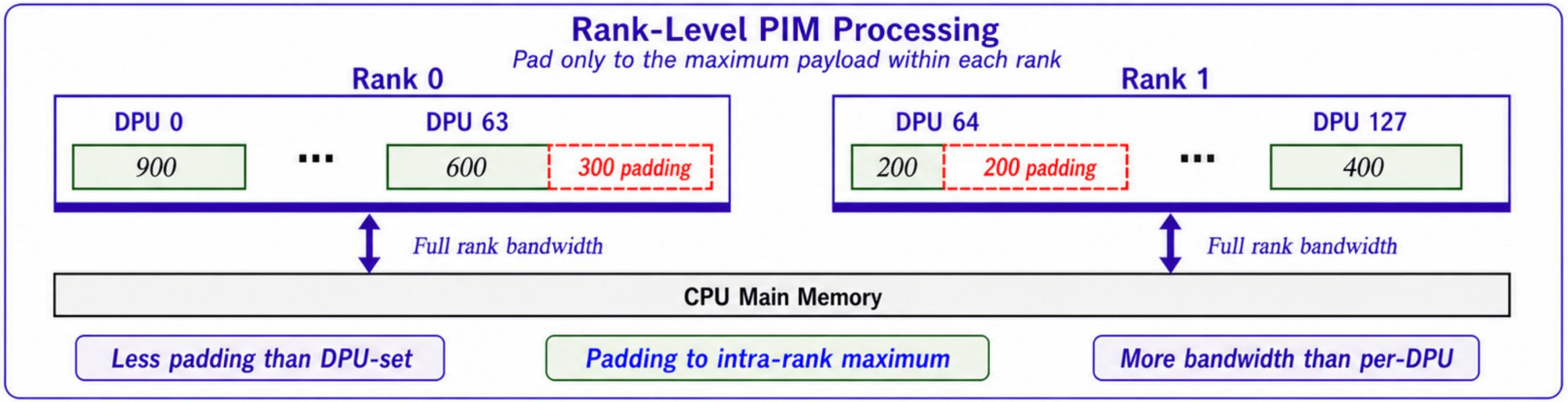


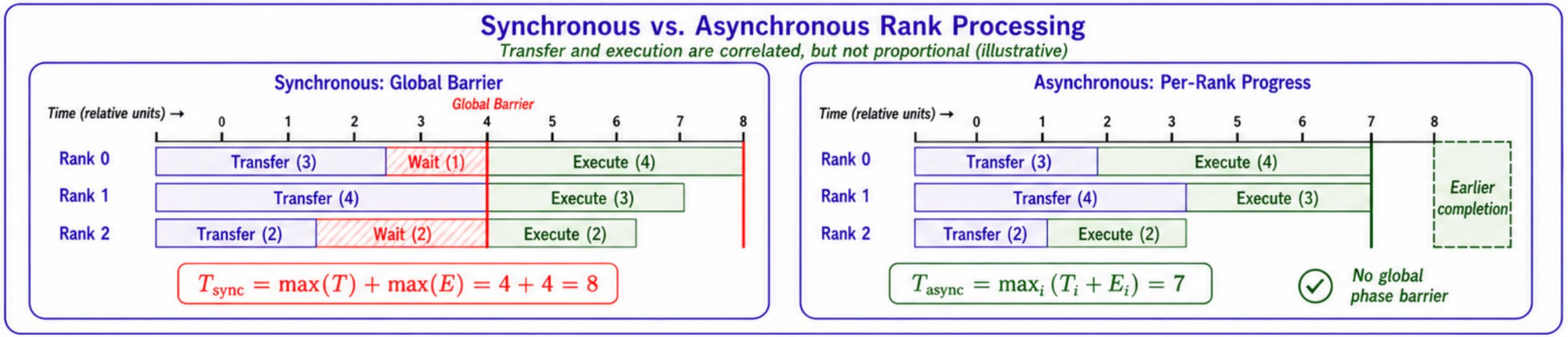
Lightweight: 4-byte counter per EV (<1% for 128-dim EV)

Lightweight Split/Merge Operation



Design 3: Rank-Level PIM Processing





More Details

- Detailed Structure of the DPU Buffer
- Aging Mechanism
- Adaptive Parameter Changing

...

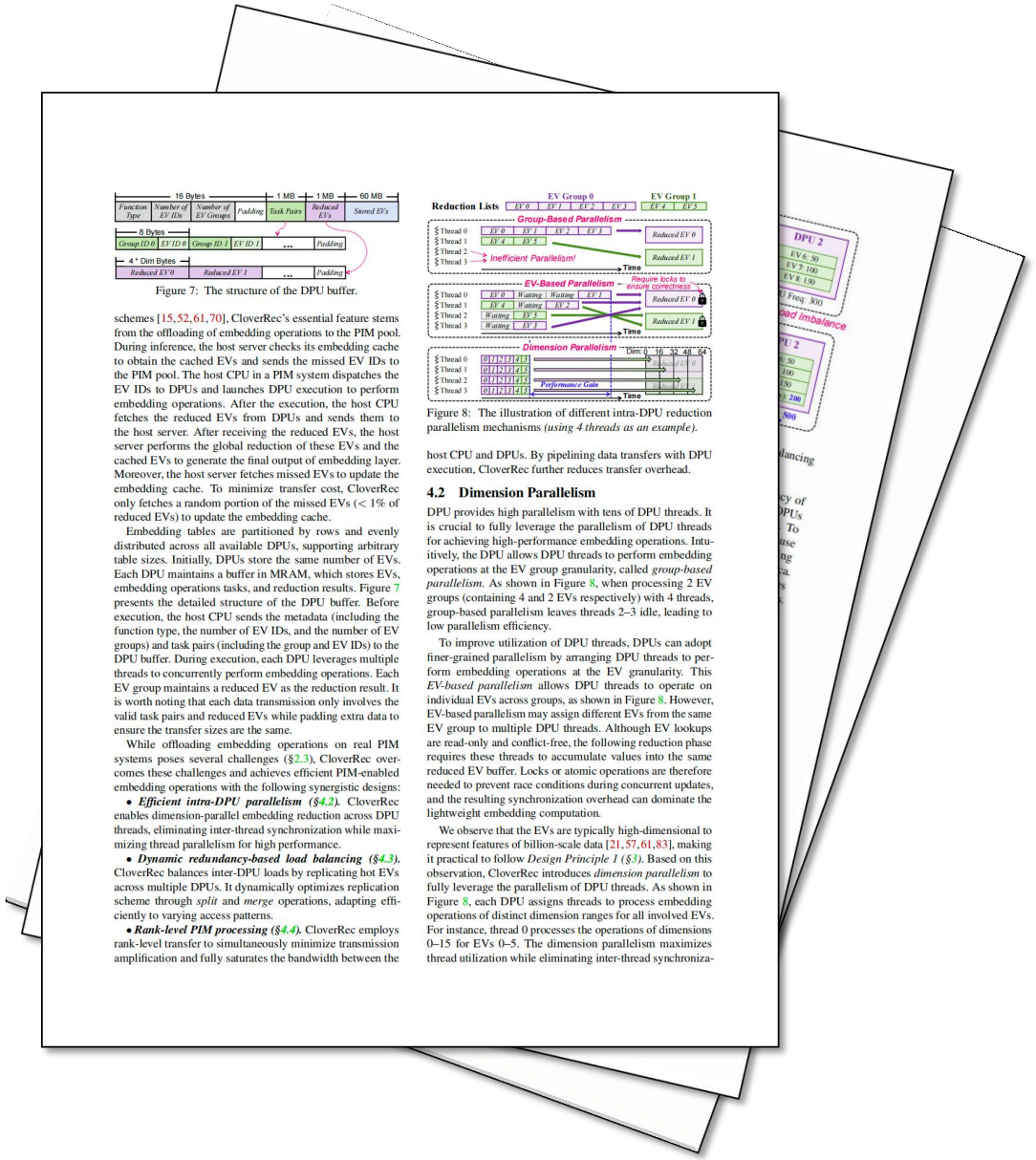


Figure 7: The structure of the DPU buffer.

schemes [15,52,61,70], CloverRec's essential feature stems from the offloading of embedding operations to the PIM pool. During inference, the host server checks its embedding cache to obtain the cached EVs and sends the missed EV IDs to the PIM pool. The host CPU in a PIM system dispatches the EV IDs to DPUs and launches DPU execution to perform embedding operations. After the execution, the host CPU fetches the reduced EVs from DPUs and sends them to the host server. After receiving the reduced EVs, the host server performs the global reduction of these EVs and the cached EVs to generate the final output of embedding layer. Moreover, the host server fetches missed EVs to update the embedding cache. To minimize transfer cost, CloverRec only fetches a random portion of the missed EVs (< 1% of reduced EVs) to update the embedding cache.

Embedding tables are partitioned by rows and evenly distributed across all available DPUs, supporting arbitrary table sizes. Initially, DPUs store the same number of EVs. Each DPU maintains a buffer in MRAM, which stores EVs, embedding operations tasks, and reduction results. Figure 7 presents the detailed structure of the DPU buffer. Before execution, the host CPU sends the metadata (including the function type, the number of EV IDs, and the number of EV groups) and task pairs (including the group and EV IDs) to the DPU buffer. During execution, each DPU leverages multiple threads to concurrently perform embedding operations. Each EV group maintains a reduced EV as the reduction result. It is worth noting that each data transmission only involves the valid task pairs and reduced EVs while padding extra data to ensure the transfer sizes are the same.

While offloading embedding operations on real PIM systems poses several challenges (§2.3), CloverRec overcomes these challenges and achieves efficient PIM-enabled embedding operations with the following synergistic designs:

- ***Efficient intra-DPU parallelism (§4.2).*** CloverRec enables dimension-parallel embedding reduction across DPU threads, eliminating inter-thread synchronization while maximizing thread parallelism for high performance.
- ***Dynamic redundancy-based load balancing (§4.3).*** CloverRec balances inter-DPU loads by replicating hot EVs across multiple DPUs. It dynamically optimizes replication scheme through *split* and *merge* operations, adapting efficiently to varying access patterns.
- ***Rank-level PIM processing (§4.4).*** CloverRec employs rank-level transfer to simultaneously minimize transmission amplification and fully saturates the bandwidth between the host CPU and DPUs. By pipelining data transfers with DPU execution, CloverRec further reduces transfer overhead.

Figure 8: The illustration of different intra-DPU reduction parallelism mechanisms *(using 4 threads as an example)*.

4.2 Dimension Parallelism

DPU provides high parallelism with tens of DPU threads. It is crucial to fully leverage the parallelism of DPU threads for achieving high-performance embedding operations. Intuitively, the DPU allows DPU threads to perform embedding operations at the EV group granularity, called *group-based parallelism*. As shown in Figure 8, when processing 2 EV groups (containing 4 and 2 EVs respectively) with 4 threads, group-based parallelism leaves threads 2–3 idle, leading to low parallelism efficiency.

To improve utilization of DPU threads, DPUs can adopt finer-grained parallelism by arranging DPU threads to perform embedding operations at the EV granularity. This *EV-based parallelism* allows DPU threads to operate on individual EVs across groups, as shown in Figure 8. However, EV-based parallelism may assign different EVs from the same EV group to multiple DPU threads. Although EV lookups are read-only and conflict-free, the following reduction phase requires these threads to accumulate values into the same reduced EV buffer. Locks or atomic operations are therefore needed to prevent race conditions during concurrent updates, and the resulting synchronization overhead can dominate the lightweight embedding computation.

We observe that the EVs are typically high-dimensional to represent features of billion-scale data [21,57,61,83], making it practical to follow *Design Principle 1 (§3)*. Based on this observation, CloverRec introduces *dimension parallelism* to fully leverage the parallelism of DPU threads. As shown in Figure 8, each DPU assigns threads to process embedding operations of distinct dimension ranges for all involved EVs. For instance, thread 0 processes the operations of dimensions 0–15 for EVs 0–5. The dimension parallelism maximizes thread utilization while eliminating inter-thread synchroniza-

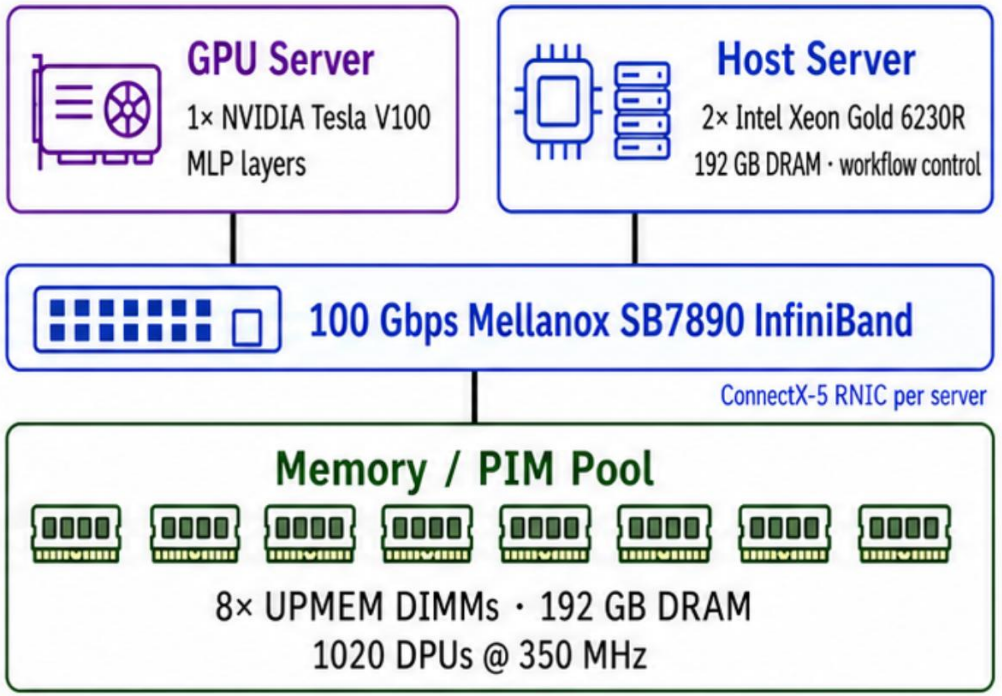
Evaluation Setup

Workload Configurations

Model	EV Tables	Rows / Table	Emb. Dim	Lookups / Table	Profile
RM1	10	1M	64	80	Embedding-heavy
RM2	40	1M	64	80	Embedding-heavy
RM3	10	1M	64	20	Compute-heavy
RM4	80	1M	128	160	Embedding-heavy
Kaggle	26	Variable (≤10M)	64	80	Embedding-heavy

RM1–RM4: synthetic, Zipfian skew = 0.99 · Kaggle: real-world · RM3 Bottom MLP: 2560-512-64

Real Disaggregated PIM Testbed



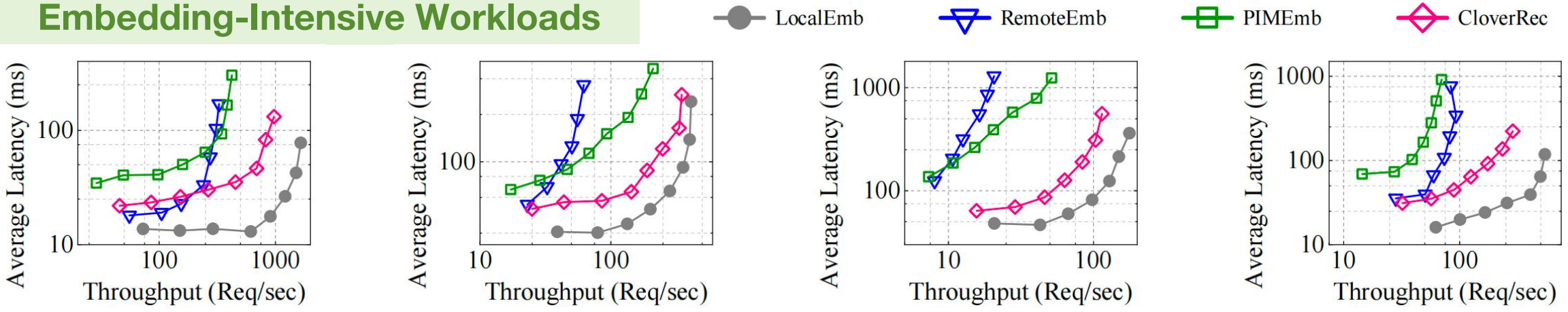
Compared Systems

- **LocalEmb** · upper bound
- **RemoteEmb** · one-sided RDMA
- **PIMEmb** · naive PIM
- **CloverRec** · optimized PIM

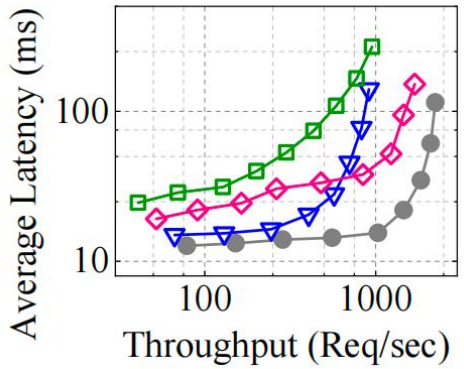
Default: batch size 32 · 1020 DPUs · 16 threads / DPU · LRU cache = 0.1% of total EV size

End-to-End Performance

Embedding-Intensive Workloads



Compute-Intensive Workloads



CloverRec recovers the disaggregated penalty

up to **8.18×** vs. RemoteEmb

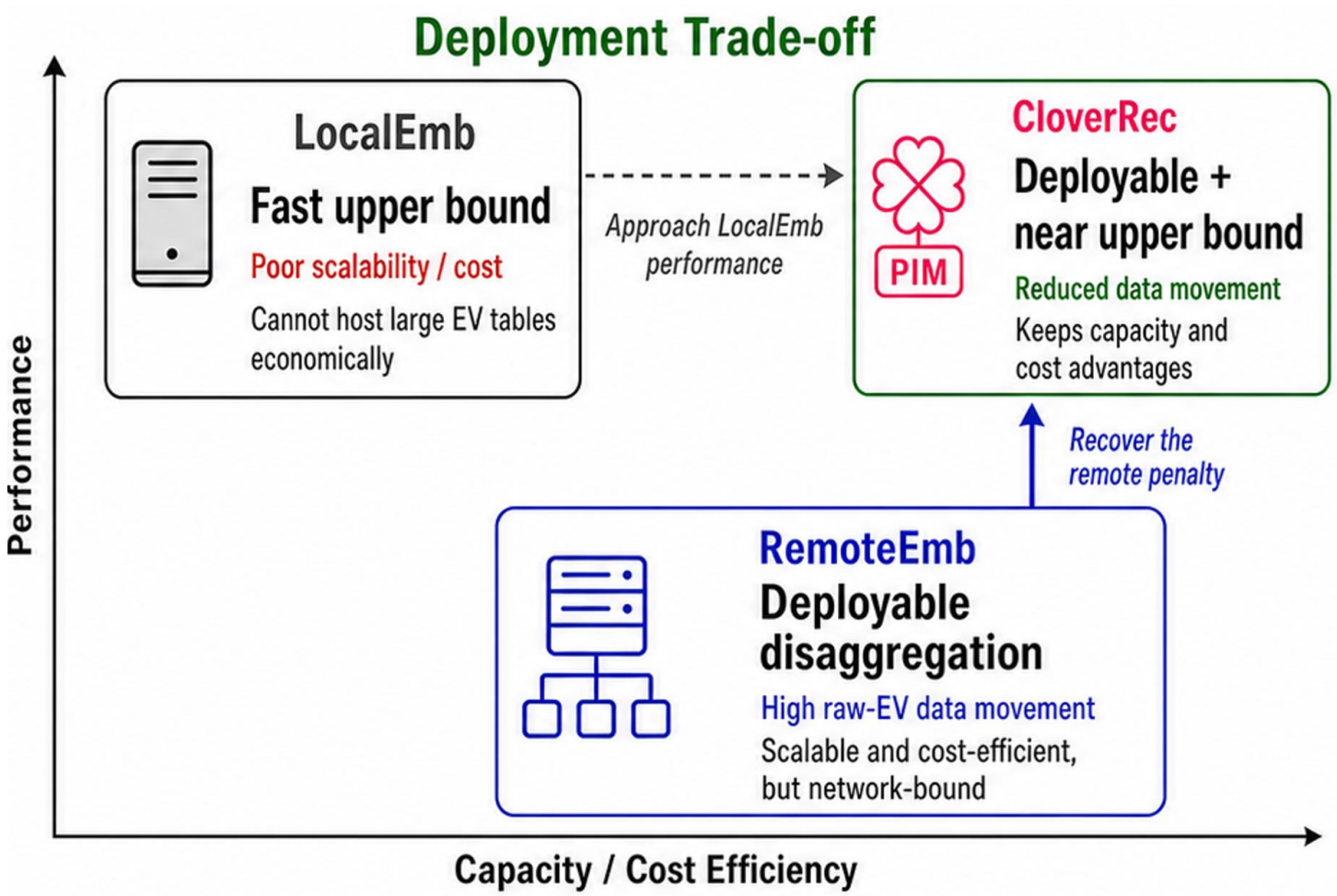
up to **5.05×** vs. PIMEmb

Largest gains on embedding-intensive workloads; RM3 also improves.

CloverRec approaches the LocalEmb upper bound.

Less network movement + lower PIM-side overhead

Approaching Local Performance Without Sacrificing Scalability



Workload Type Matters

Embedding-intensive

RM1 · RM2 · RM4 · Kaggle

Embedding movement and PIM processing dominate

CloverRec delivers the largest gains

Compute-intensive

RM3

Larger MLP reduces the relative embedding share

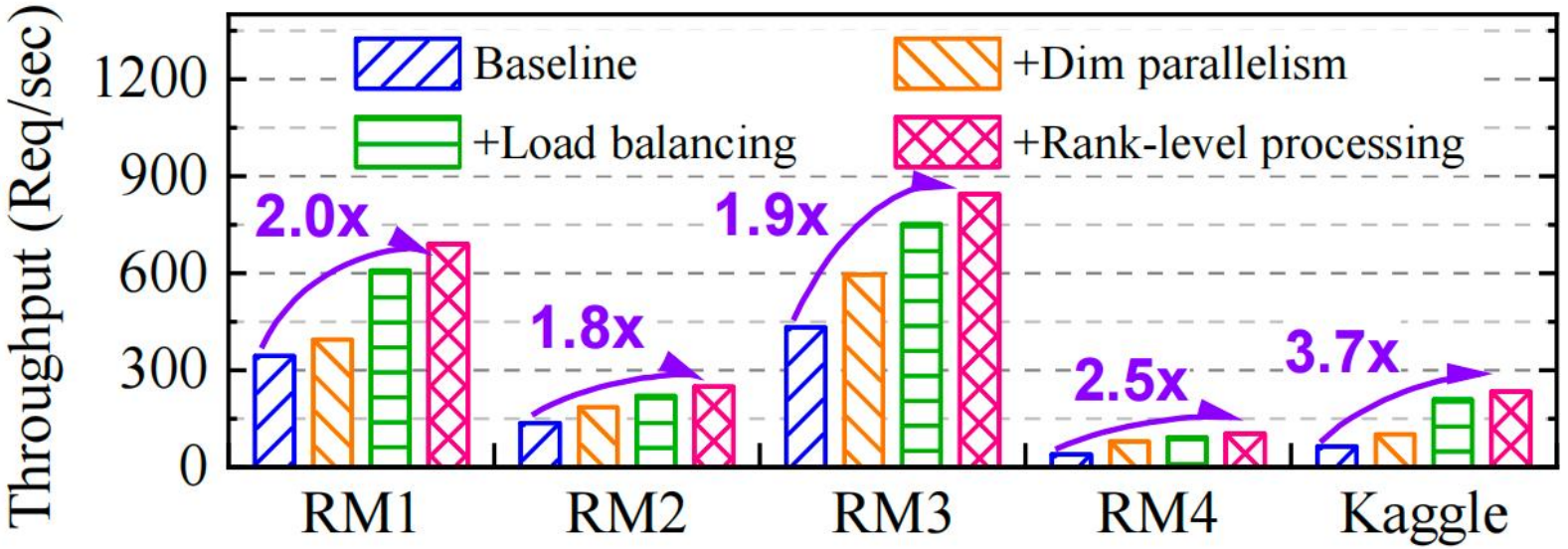
Embedding still affects end-to-end performance

up to 1.49× vs. RemoteEmb	up to 2.09× vs. PIMEmb

Goal: Near-LocalEmb performance with capacity and cost efficiency

Ablation Study

End-to-End Performance Contributions



- *Starting from a Baseline without any proposed designs, we progressively apply each feature*

What Each Design Protects

1. **Dimension Parallelism**
   1.14×–1.93×
   Lock-free thread utilization
   Efficient intra-DPU processing

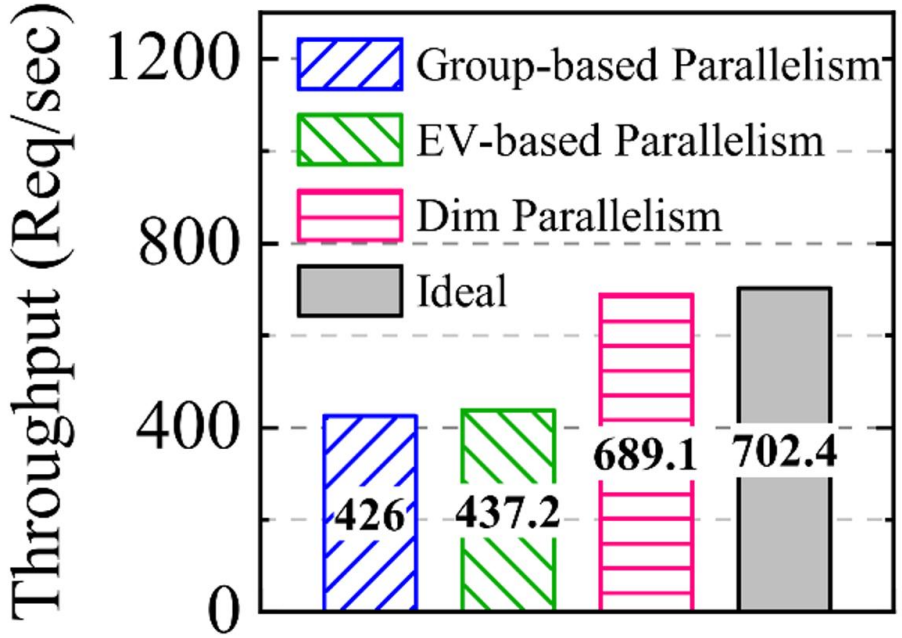
2. **Dynamic Load Balancing**
   1.45× avg.
   Lower maximum DPU load
   Protects parallel efficiency under skew

3. **Rank-Level Processing**
   1.11×–1.13×
   Less transfer amplification
   Keeps host–DPU transfer efficient

Right Granularity Unlocks PIM Efficiency

1 Dimension Parallelism

DPU computation: balanced work + independent outputs

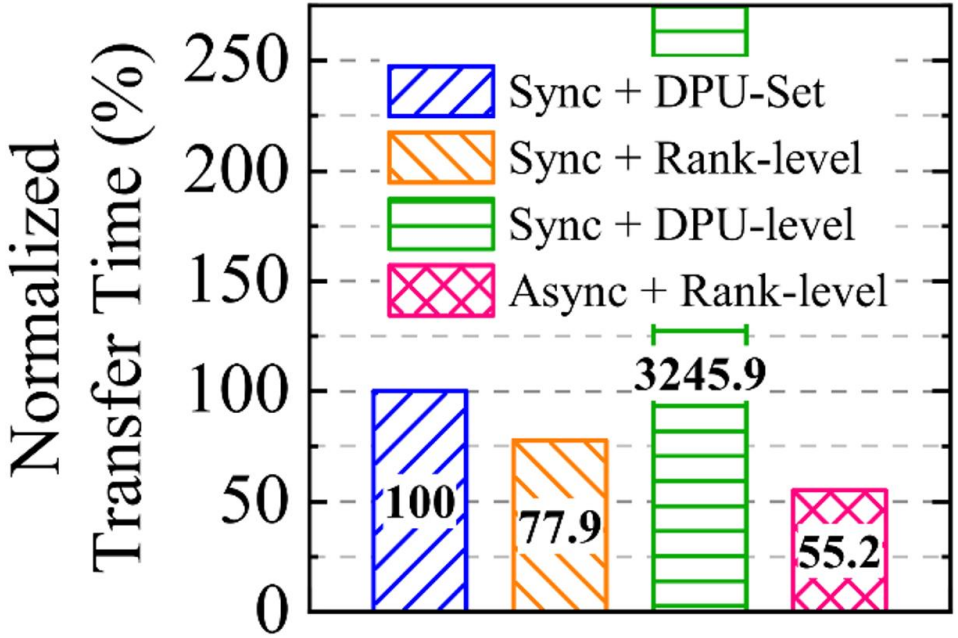


1.62× higher throughput

Balanced threads, no shared-output locks

2 Rank-Level Transfer

Host-DPU path: less padding + sufficient bandwidth

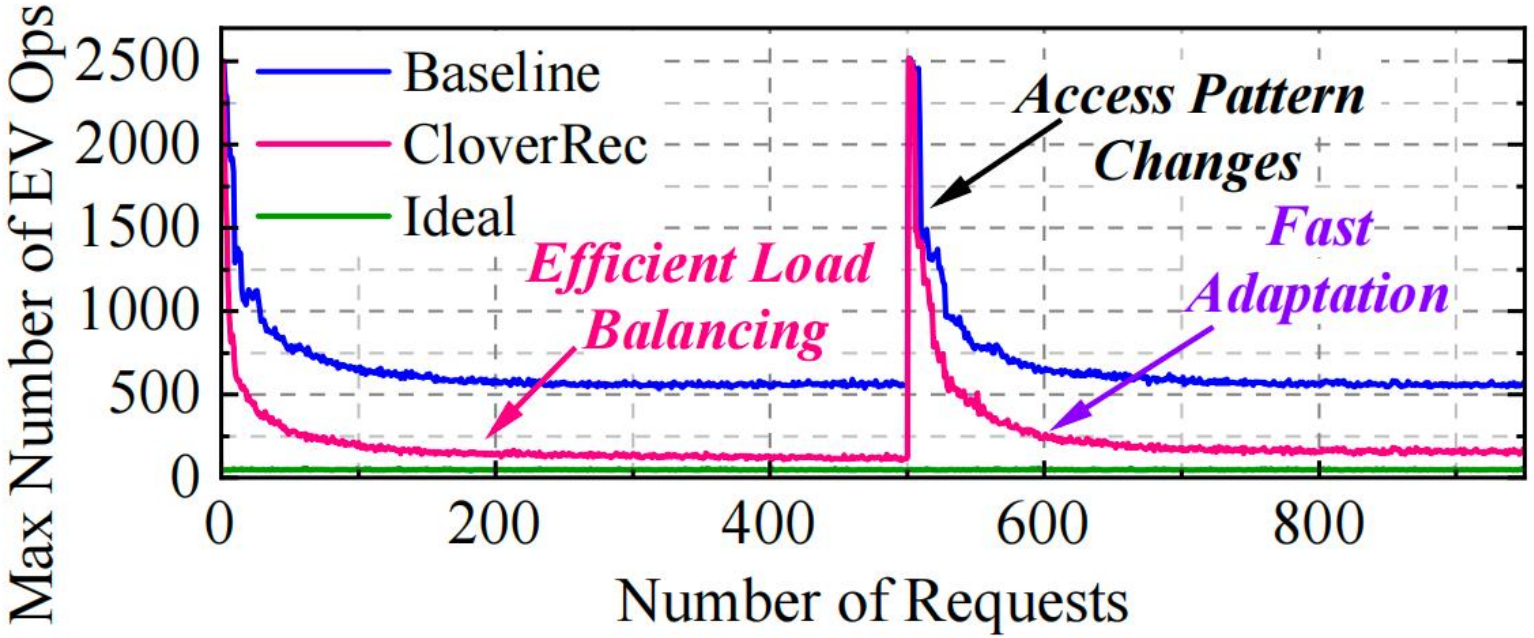


−22.1% vs. DPU-Set

−22.7 pts further with async

Load-Balancing Adaptation

Maximum DPU Load Under Moving Hotspots



How CloverRec Responds

1. Hot EVs move
Access pattern changes

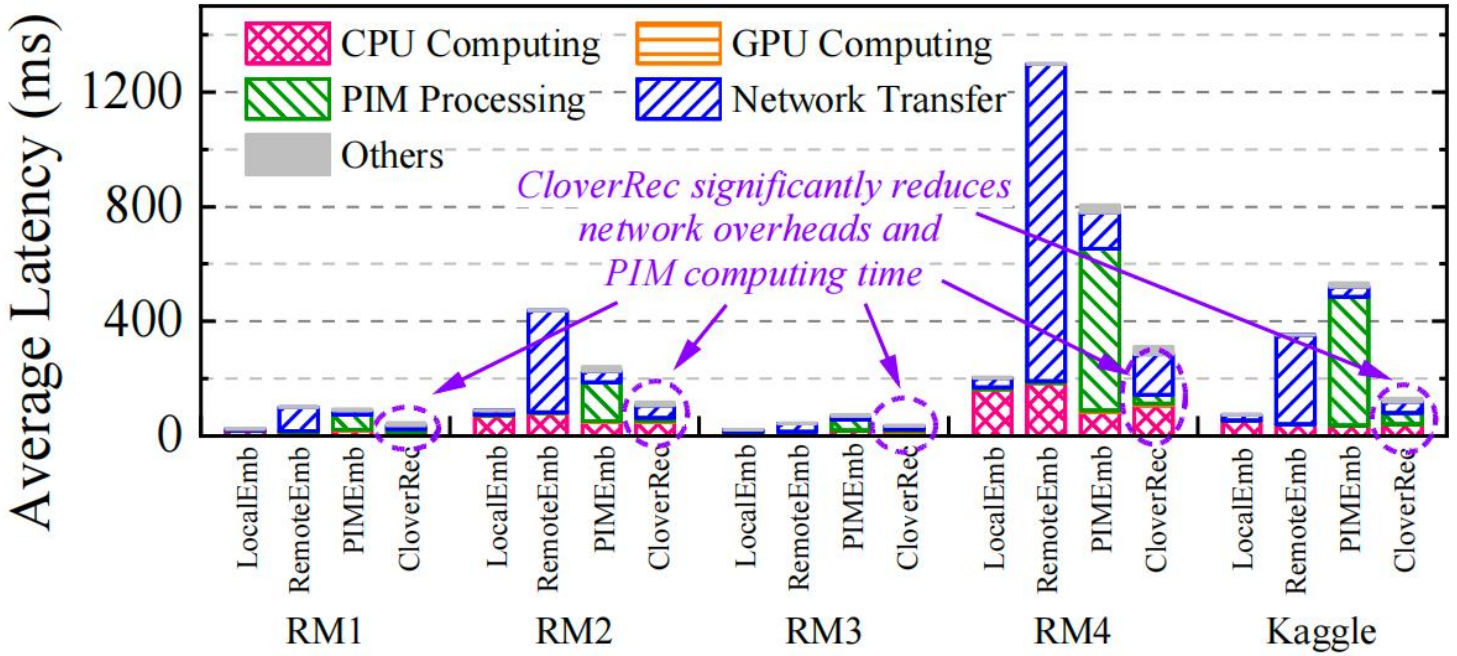
2. Replicas split
Hot objects spread across DPUs

3. Load rebalances
Maximum DPU load approaches Ideal

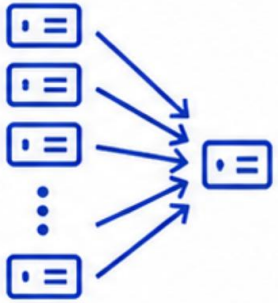
4.28× lower
maximum DPU load
Fast adaptation after each hotspot shift

Latency Breakdown

End-to-End Latency Breakdown



Two Bottlenecks Removed

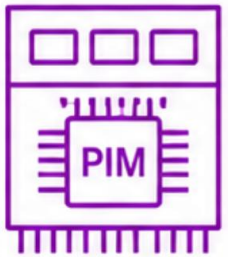


vs. RemoteEmb

7.92× lower

network cost

PIM-side reduction returns reduced EVs



vs. PIMEmb

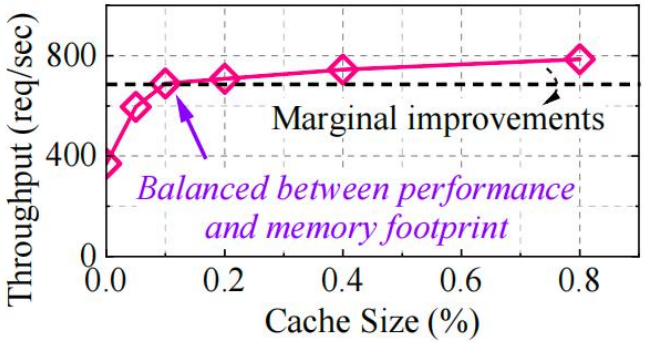
18.68× faster

PIM processing

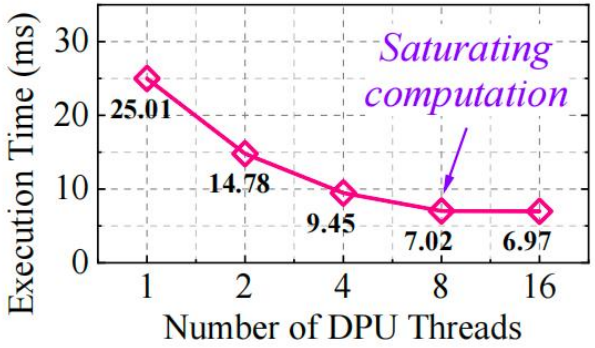
Parallelism + balancing + rank-level processing

Robustness and Overheads: Low Extra Cost

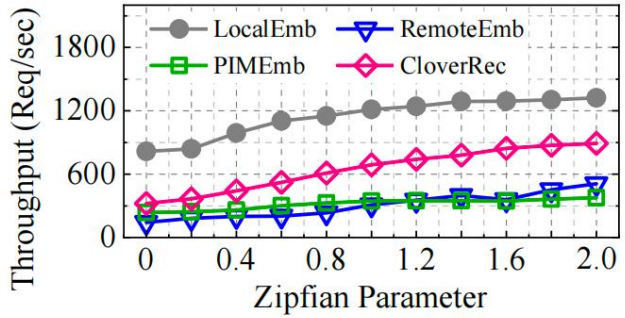
Host Cache



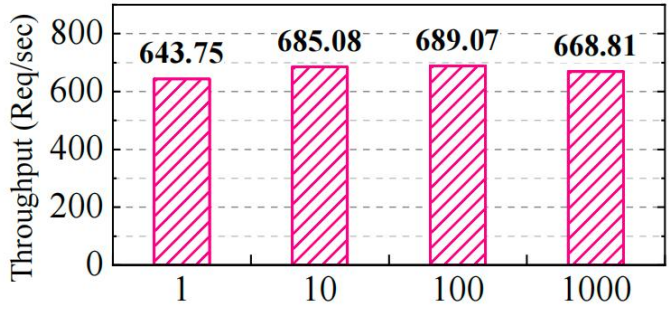
DPU Threads



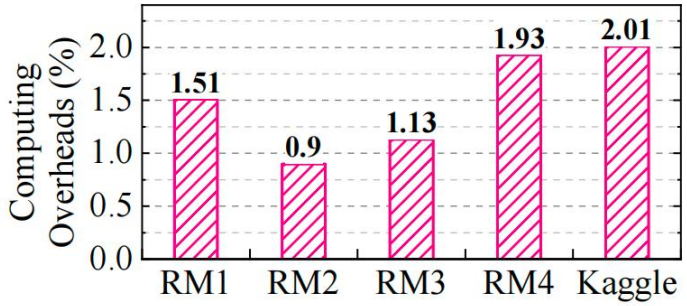
Skewness



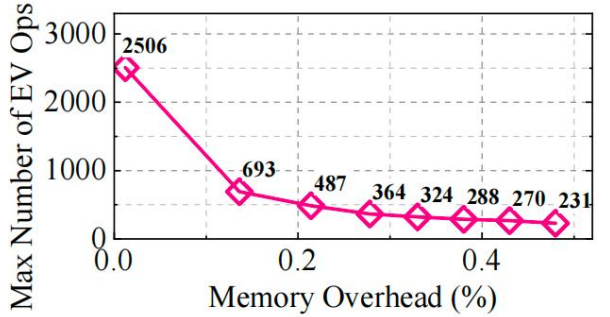
Aging



Split/Merge Compute



Replica Memory



1.85× throughput @ 0.1% cache

3.56× lower DPU time @ 8 threads

7.1% aging fluctuation

0.82%-2.01% split/merge compute

<0.5% replica memory

Low overhead preserves the advantage of PIM-based disaggregation

Conclusion

- Large embedding tables make fully local serving **cost-inefficient**.

  Disaggregation scales capacity but incurs a **remote data-movement penalty**.

- **CloverRec** makes PIM-based disaggregation practical with **PIM-aware designs**:
  - Dimension parallelism
  - Dynamic redundancy
  - Rank-level processing

- On **real UPMEM hardware**, CloverRec outperforms state-of-the-artembedding disaggregation by up to **8.18x** with **low overhead**.

Thank you! Q&A