Maximizing Heterogeneous Memory Bandwidth in Multi-Socket Systems

Rakie Kim, Yunjeong Mun, Honggyu Kim
SK hynix
2026.08.11

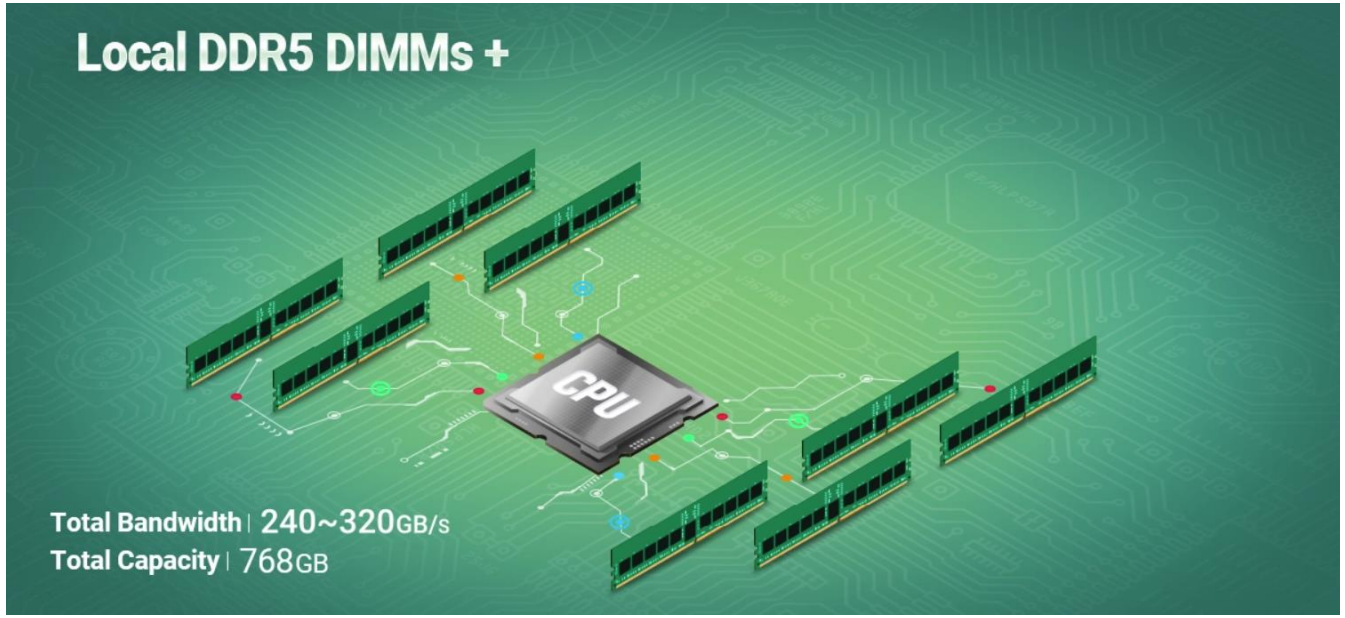
Contents

Contents

Conventional (Homogeneous) Memory System

- Most systems have the same type of memory in their DIMM slots.
  - It just works without software support.
- But it's not easy to expand memory beyond DIMM slot limits.
  - Due to the real-estate issues in the server form factors.

DRAM attached x86 server - Supermicro

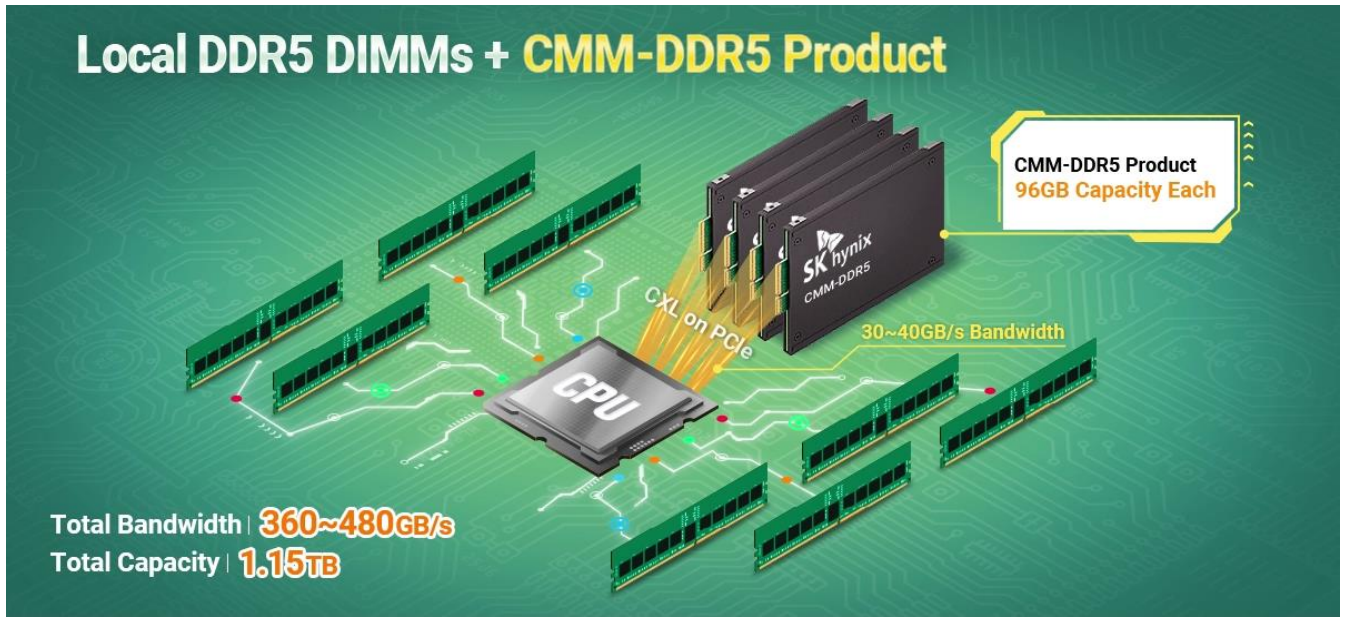


Heterogeneous Memory System

○ CXL memory allows bandwidth/capacity expansion beyond DIMM slot limits.

- But it requires software support for efficient use.

SK hynix CMM(CXL Memory Module)-DDR5

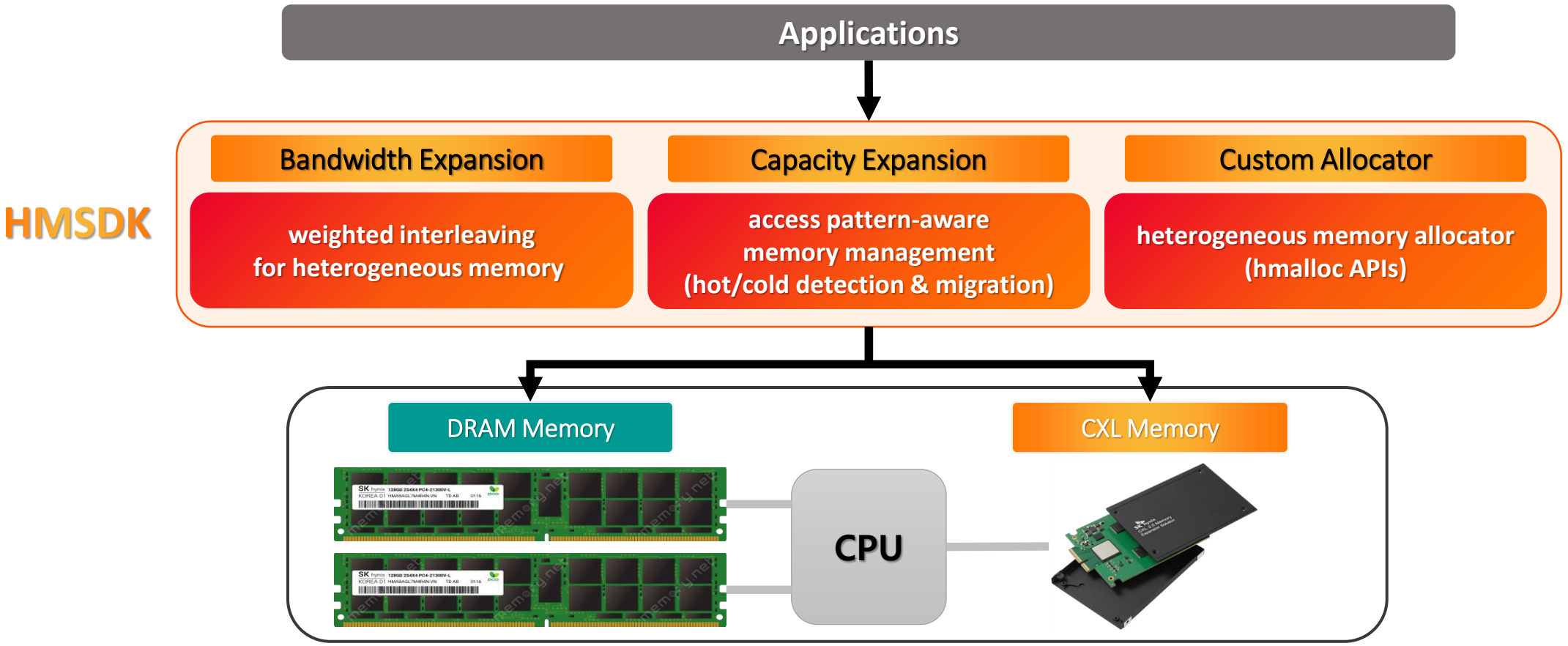


HMSDK Overview

○ HMSDK: Heterogeneous Memory Software Development Kit

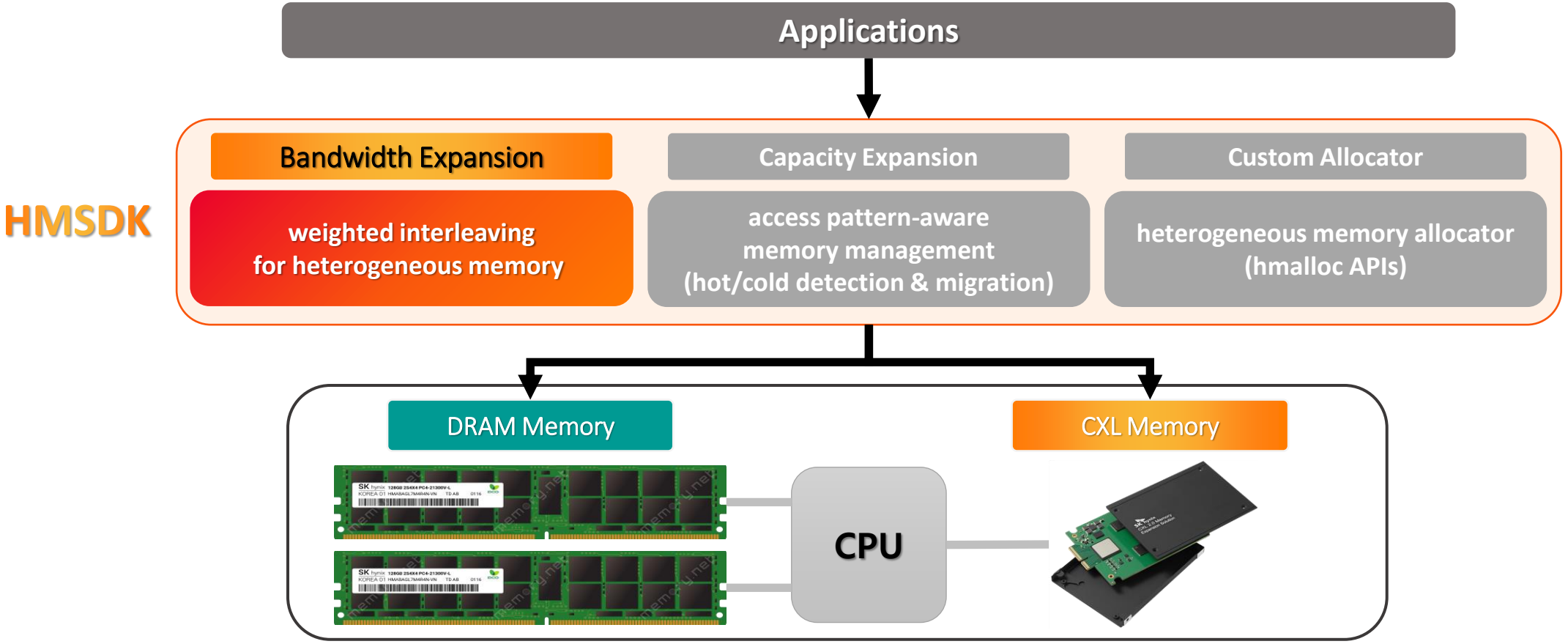
: SK hynix open-source solution for data placement across DRAM and CXL memory.

- https://github.com/skhynix/hmsdk



HMSDK - Bandwidth Expansion

- Bandwidth Expansion : a solution that uses DRAM and CXL together to expand overall memory bandwidth
  → Weighted interleave : distributes pages across the nodes in proportion to their bandwidth
- This talk focuses only on the bandwidth expansion for multi-socket systems



Contents

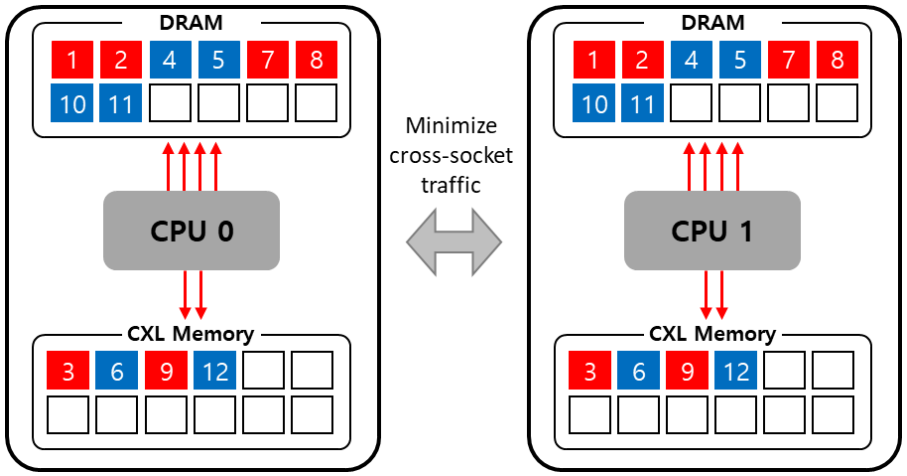
Background & Proposed Approach

○ Research Motivation

- Next-generation datacenter applications demand ever-growing memory bandwidth
  → memory expansion with CXL (Compute Express Link) becomes essential
- To overcome the scaling limit of local DRAM, CXL-based weighted interleave was introduced into the OS (v6.9)
- In multi-socket systems, a structural limitation of the OS causes inefficient remote-socket accesses
  → degrading overall system performance

○ Proposed Approach

- Socket-aware weighted interleave: discovers the hardware topology and confines allocation to the local socket
  → 1.33 TB/s effective bandwidth (+47% vs. Conventional weighted interleave) and 20% lower Loaded Latency

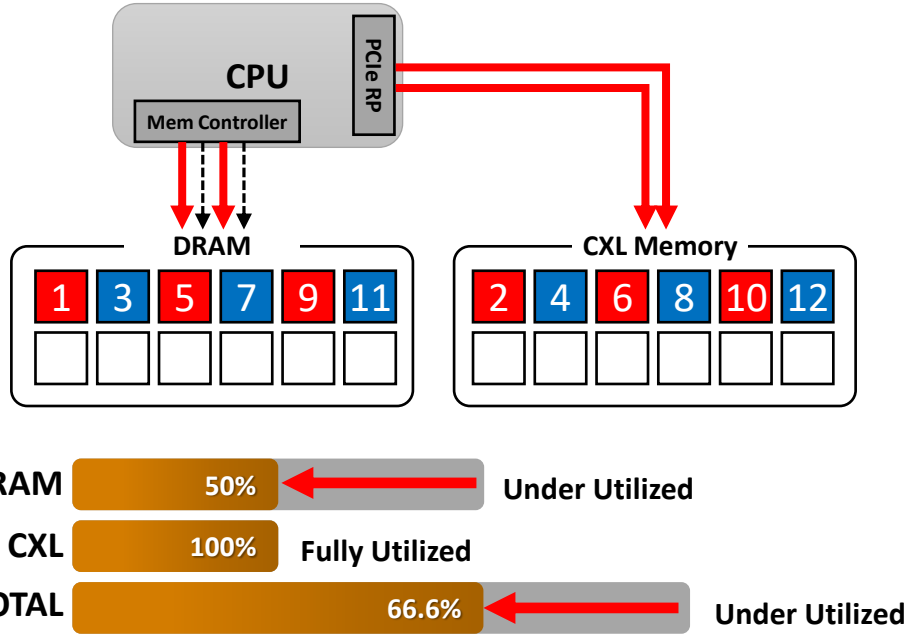


Round robin interleave vs Weighted interleave

- Bandwidth can be expanded with conventional 1:1 interleaving. (MPOL_INTERLEAVE)
- Weighted interleaving is needed to handle the bandwidth differences. (MPOL_WEIGHTED_INTERLEAVE)
  - Officially supported from Linux v6.9 and onwards.

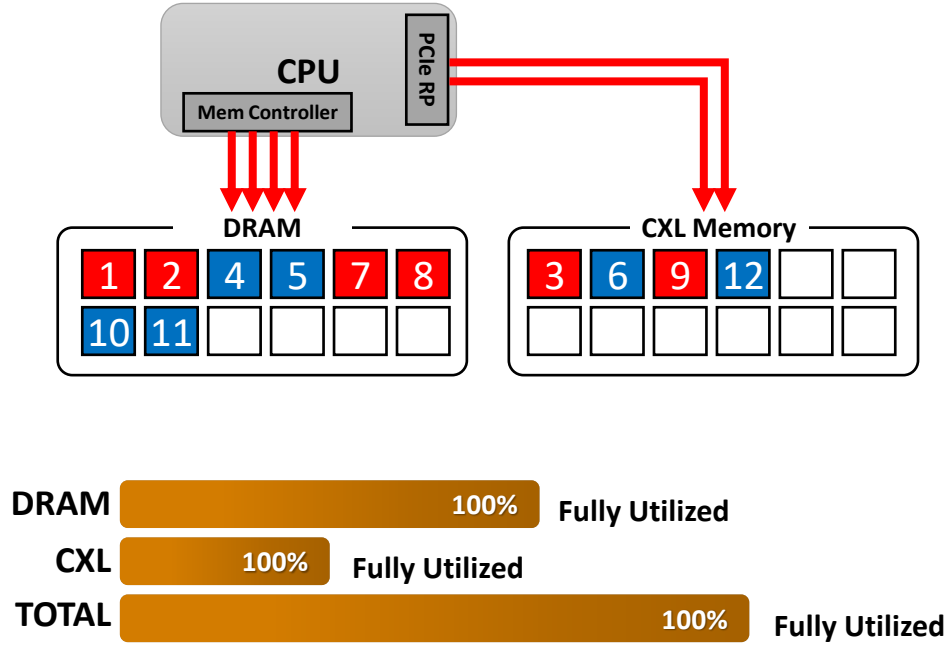
1:1 Round-robin interleave

- Evenly distribute pages regardless of the target media such as DRAM or CXL memory.



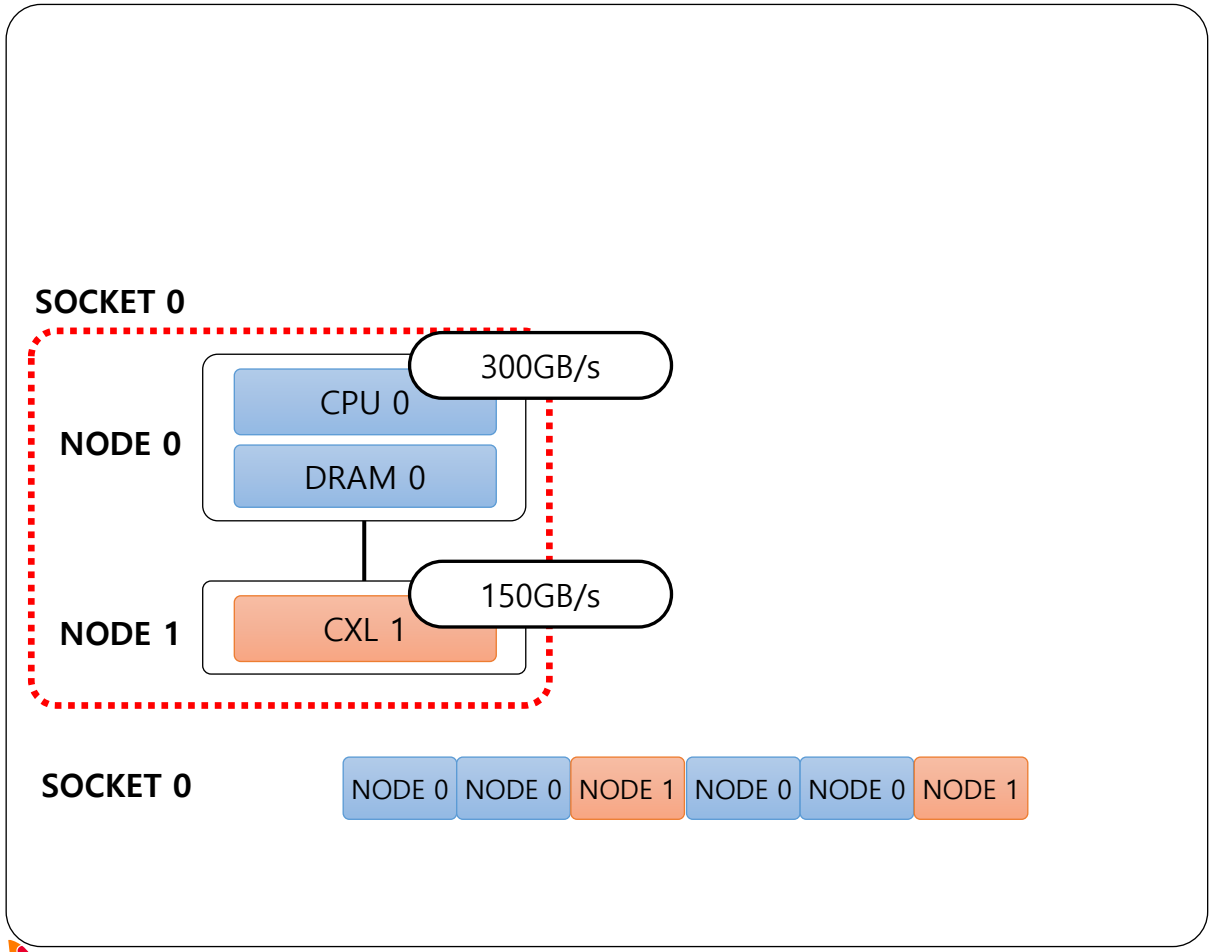
N:M Weighted interleave

- Distribute pages in a weighted manner across different NUMA nodes.



Weighted interleave on a Single Socket

- Global Weight: one system-wide array that sets allocation ratios from each node's bandwidth
- On a single socket every node hangs off one CPU, so the weight is exactly proportional to real effective bandwidth
  → interleaving works as intended and throughput is maximized



CPU-to-Node Bandwidth (GB/s)

	NODE 0	NODE 1
CPU0	300	150

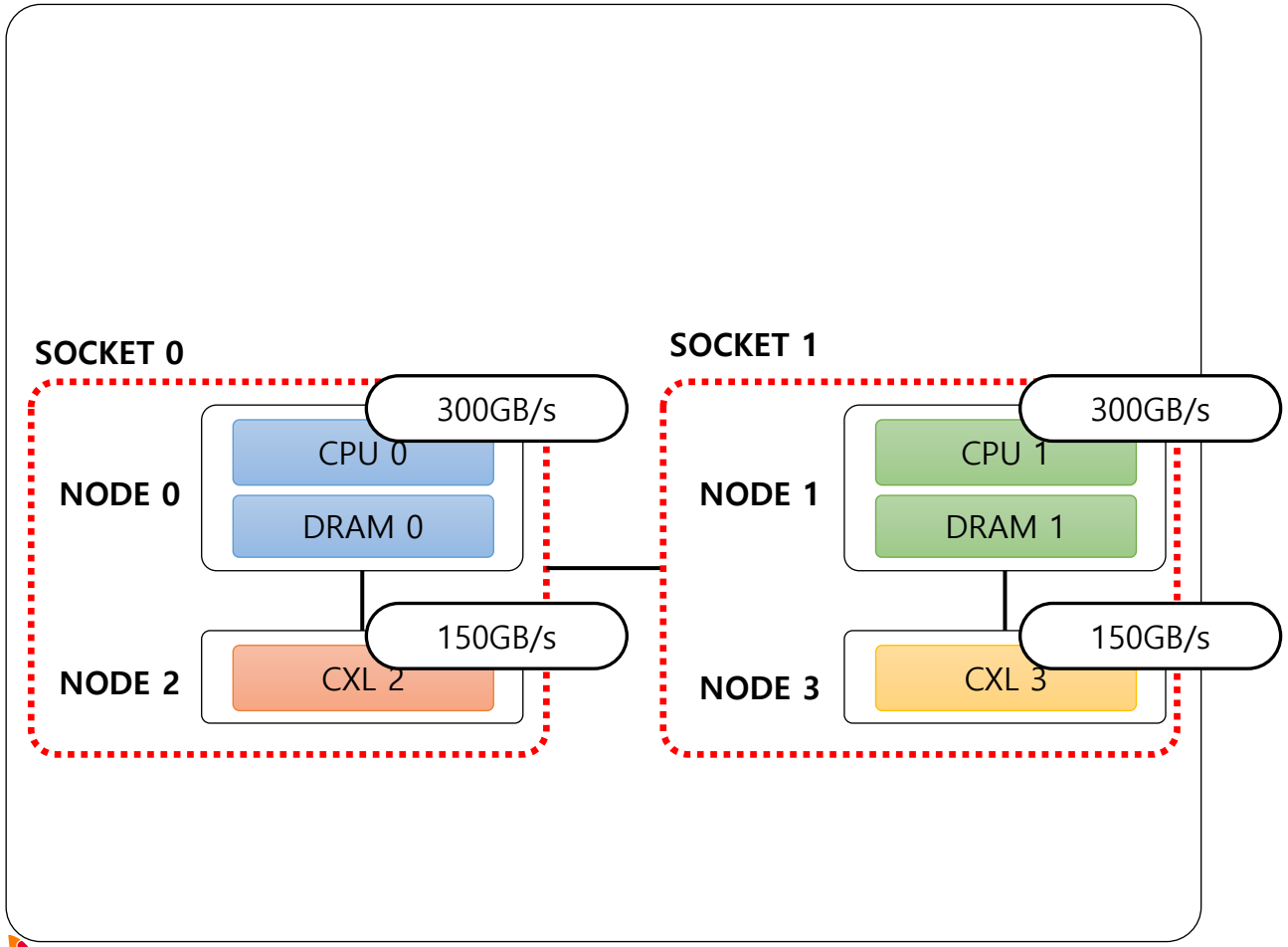
Global Weight

	NODE 0	NODE 1
Weight	2	1

Multi-Socket – One Global Weight

○ Even in a multi-socket system, the Global Weight still reflects only each memory's physical performance

- DRAM 300 GB/s → weight 2, CXL 100 GB/s → weight 1



CPU-to-Node Bandwidth (GB/s)

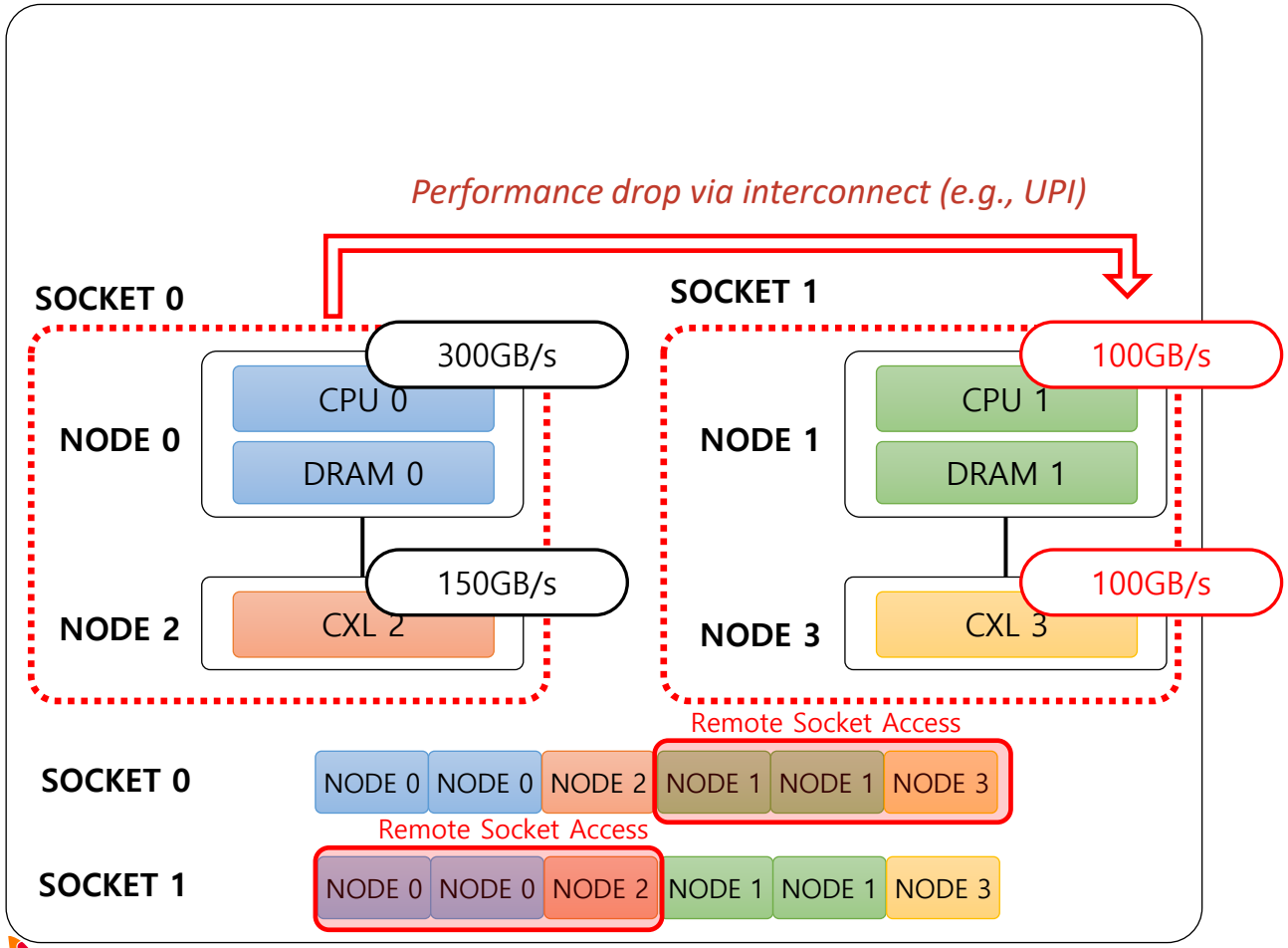
	NODE 0	NODE 1	NODE 2	NODE 3
CPU0	300	300(?)	150	150(?)
CPU1	300(?)	300	150(?)	150

Global Weight

	NODE 0	NODE 1	NODE 2	NODE 3
Weight	2	2	1	1

Multi-Socket – Over-Weighted Remote Memory

- A memory's effective performance depends on the accessing socket → the interconnect reduces it
- Ideally each CPU needs its own weight table, but the OS supports only a single global array
  - Remote memory is granted an excessively high weight relative to its effective bandwidth



CPU-to-Node Bandwidth (GB/s)

	NODE 0	NODE 1	NODE 2	NODE 3
CPU0	300	100	150	100
CPU1	100	300	100	150

Global Weight

	NODE 0	NODE 1	NODE 2	NODE 3
Weight	2	2	1	1

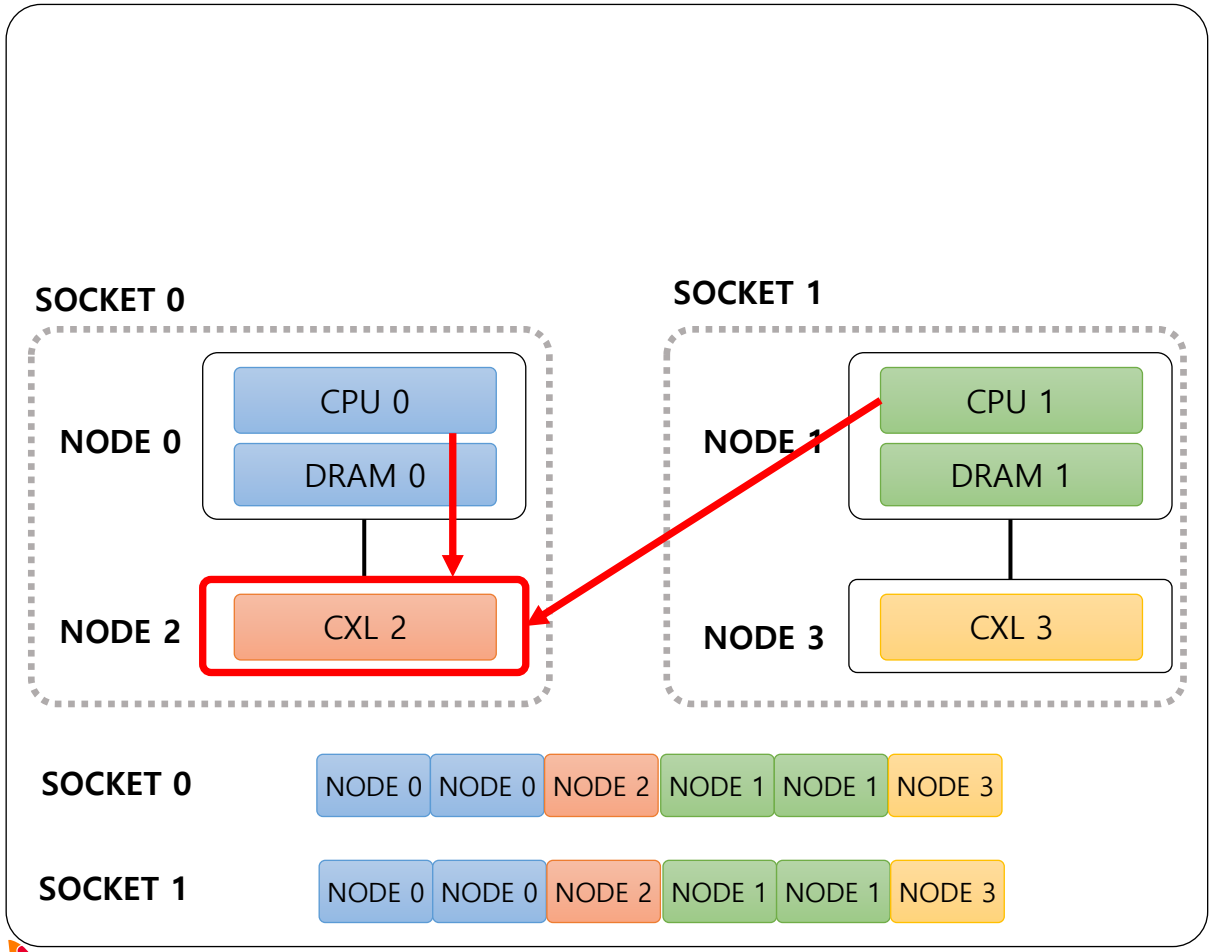
Per-CPU Weight

	NODE 0	NODE 1	NODE 2	NODE 3
CPU0	6	2	3	2
CPU1	2	6	2	3

More Sockets → more Per-CPU weight tables to manage

Multi-Socket – Concurrent Occupancy

- High-bandwidth workloads make multiple sockets hit one memory node at once → 'concurrent occupancy'
- The contention wastes bandwidth beyond any weight, so tuned weights break at runtime
  - Block remote access itself → secure locality at the socket granularity



CPU-to-Node Bandwidth (GB/s)

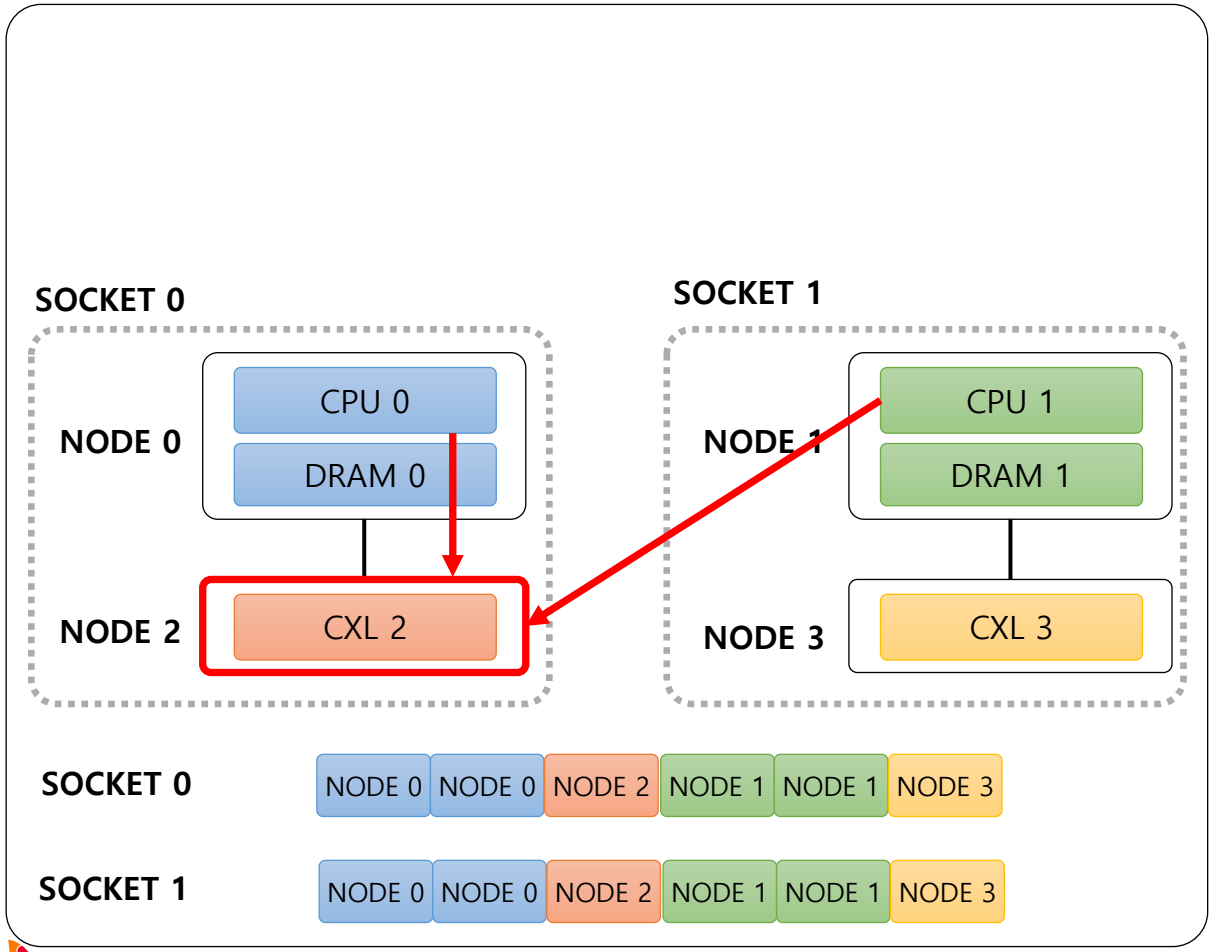
	NODE 0	NODE 1	NODE 2	NODE 3
CPU0	300	100	150	100
CPU1	100	300	100	150

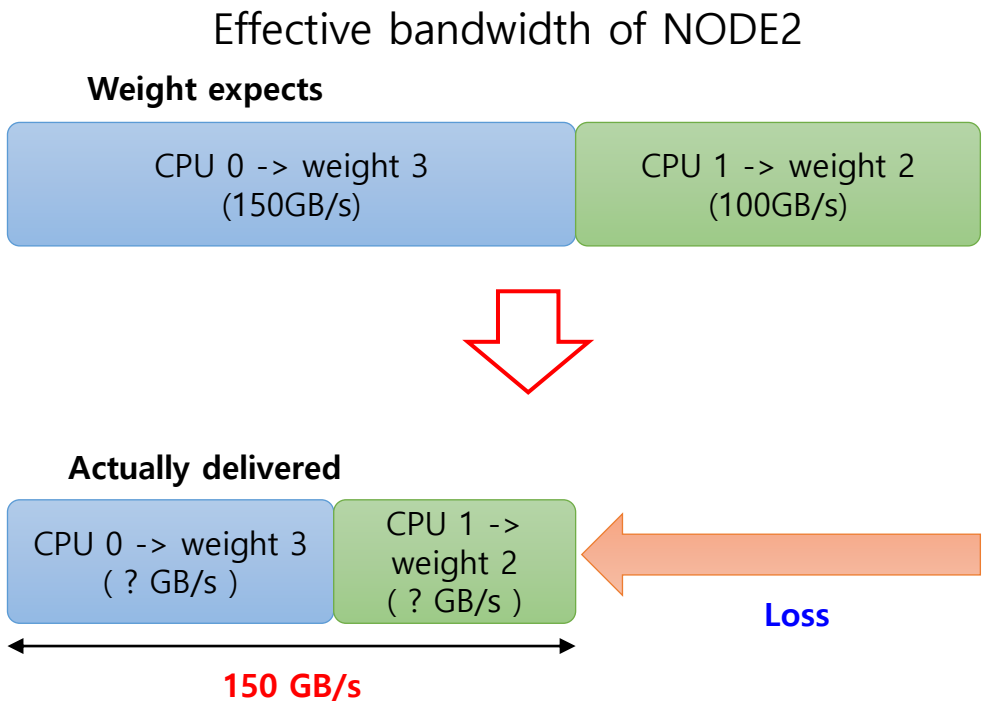
Per-CPU Weight

	NODE 0	NODE 1	NODE 2	NODE 3
CPU0	6	2	3	2
CPU1	2	6	2	3

Multi-Socket – Concurrent Occupancy

- High-bandwidth workloads make multiple sockets hit one memory node at once → 'concurrent occupancy'
- The contention wastes bandwidth beyond any weight, so tuned weights break at runtime
  - Block remote access itself → secure locality at the socket granularity





Socket-aware topology: Concept

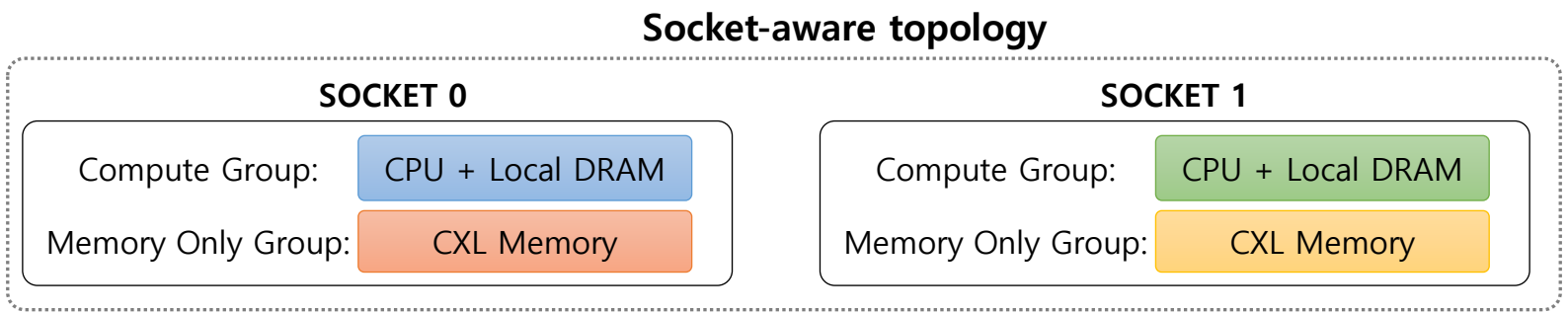
○ Root cause

- 'Topology blindness': the OS cannot represent the physical node<->socket relationship
  - Traditionally memory came paired with a CPU → its socket dependency was evident through that CPU
  - CPU-less CXL (memory-only) nodes cannot be attributed to any socket

    : Treated as global resources shared by all sockets, inviting unnecessary remote access
  - The uncertainty triggers unwanted remote accesses → unpredictable performance degradation
- Existing hints are unreliable
  - NUMA distance is only relative : it does not reflect the physical path or socket structure
  - Firmware tables (HMAT) vary across hardware models and often contain errors

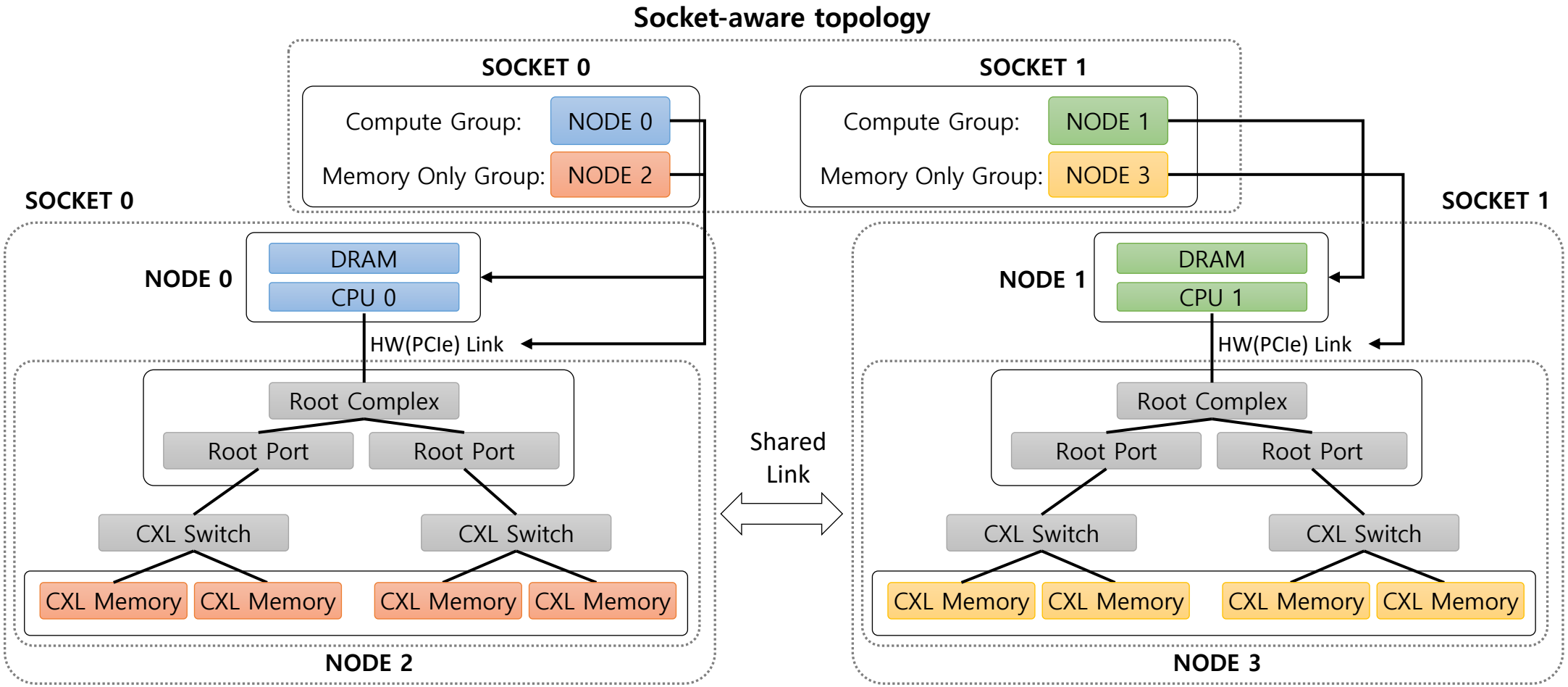
○ Proposed : Socket-aware topology

- Strictly group NUMA nodes by physical socket, derived from the actual connection path
- Within each socket, manage local DRAM and CXL as independent tiers



Socket-aware topology: PCIe-Based Discovery

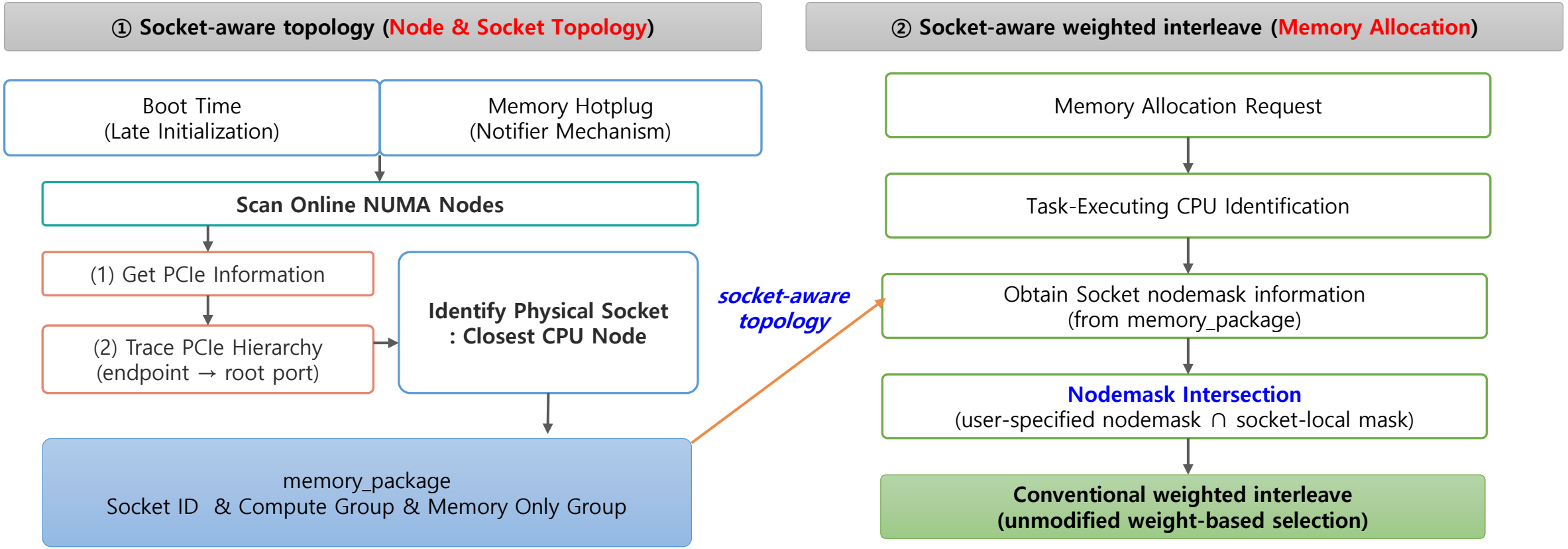
- On CXL device detection, the kernel walks the PCIe tree (Endpoint → Switch → Root Port) to find the CPU controlling it, and thus its physical socket.
- Per socket, nodes are grouped into a Compute Group and a Memory-Only Group



Architecture : Socket-aware topology & Socket-aware weighted interleave

○ Two cooperating components

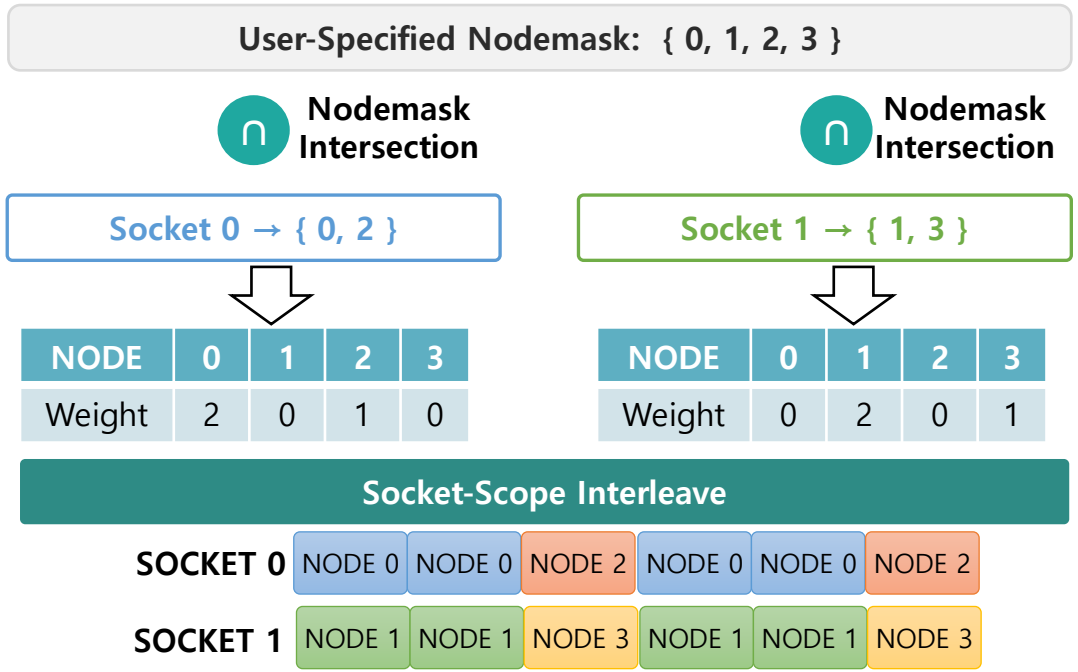
- ① the Socket-aware topology Module builds per-socket domains (memory_package)
- ② the allocation path resolves the final nodemask before Conventional weighted interleave runs

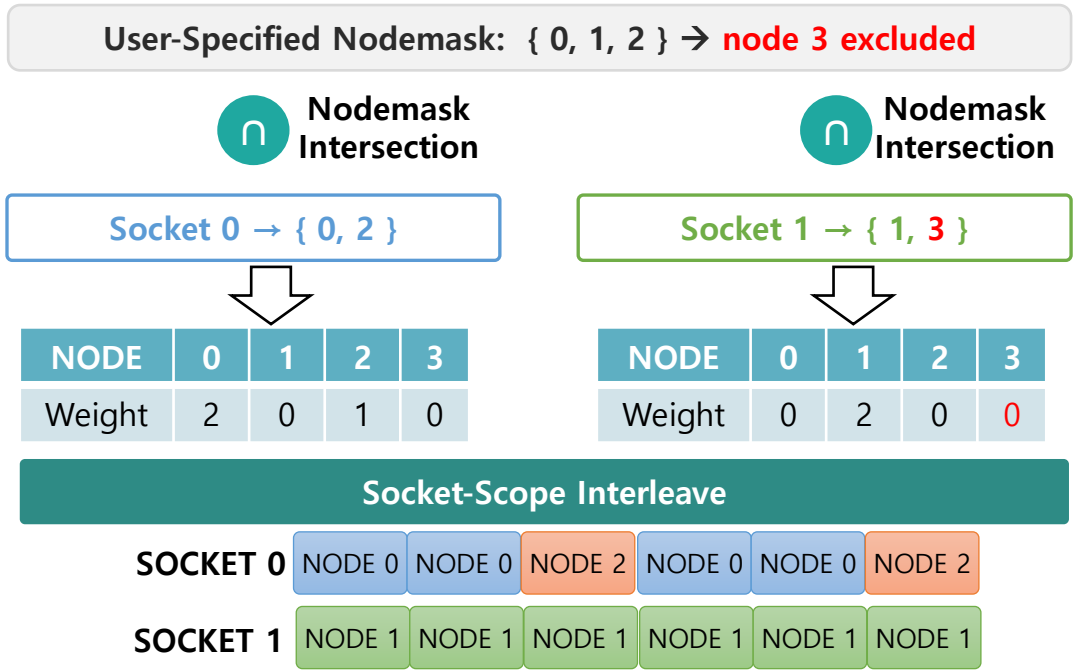


Socket-aware weighted interleave – Nodemask Intersection

- Conventional weighted interleave needs no intersection → the user nodemask is the scope
- Socket-aware takes the user nodemask as the maximum set, intersected with the socket-local mask
  - Excluding node 3 leaves socket 1 with DRAM only, socket 0 unchanged

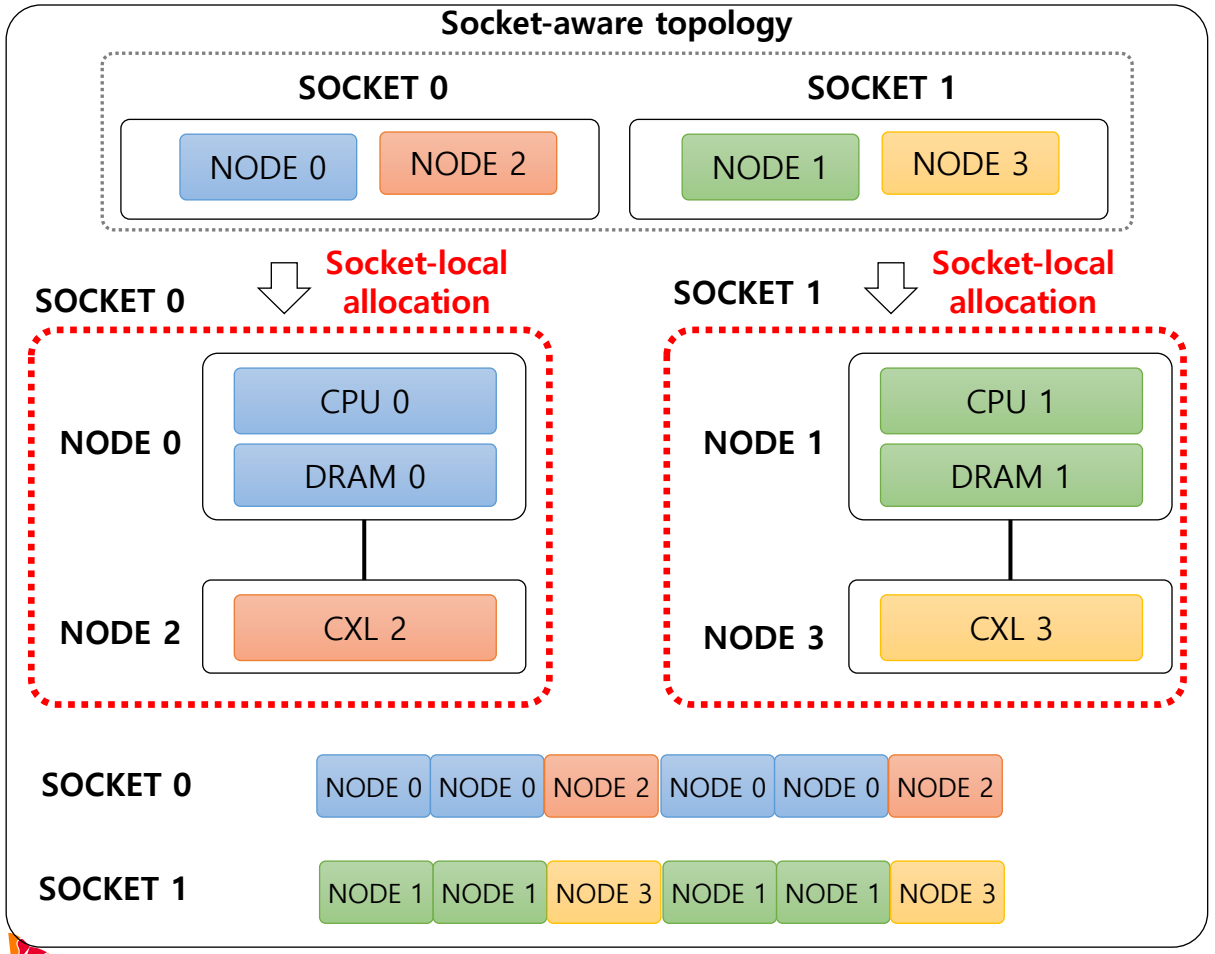
    → intent and locality both hold





Socket-aware weighted interleave – Socket-Local Allocation

- Identify the socket of the CPU running the task and confine allocation to that local socket
- Candidates narrow to local DRAM + local CXL; pages are spread by weight across them
- Remote traffic is blocked at the source, excluding concurrent occupancy → firm locality



CPU-to-Node Bandwidth (GB/s)

	NODE 0	NODE 1	NODE 2	NODE 3
CPU0	300	100	150	100
CPU1	100	300	100	150

Global Weight

	NODE 0	NODE 1	NODE 2	NODE 3
Weight	2	2	1	1

Weight with Socket-aware weighted interleave

	NODE 0	NODE 1	NODE 2	NODE 3
CPU0	2	0	1	0
CPU1	0	2	0	1

Contents

Evaluation - Environment

○ Hardware setup

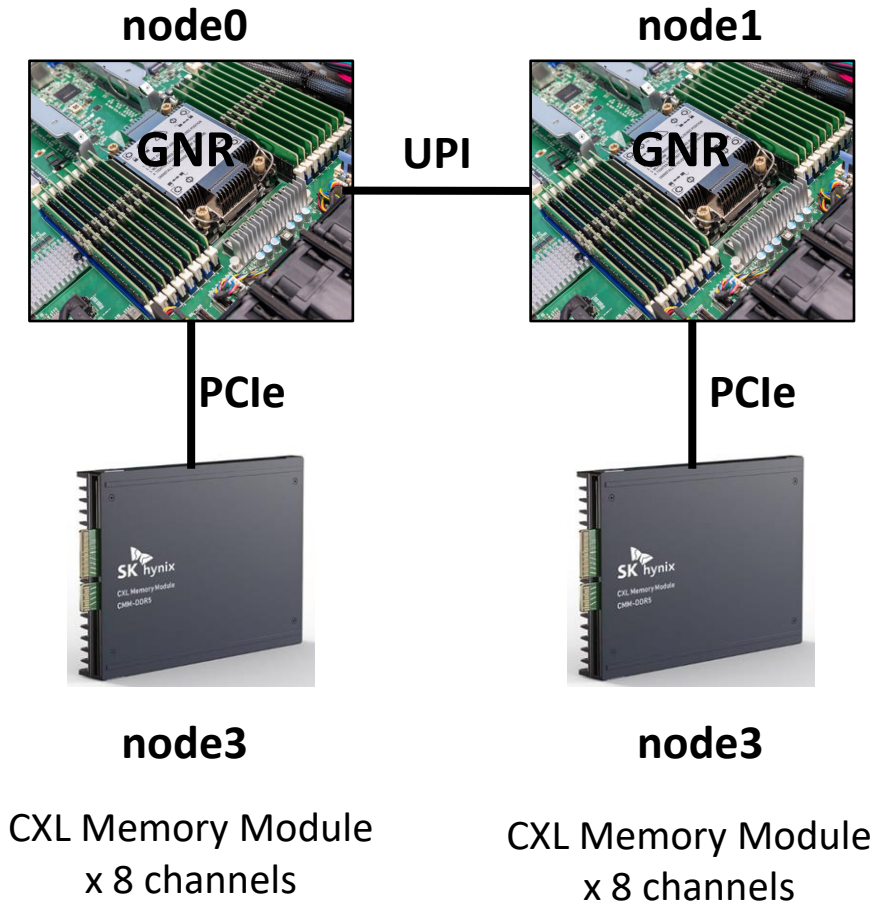
Processor	Dual-Socket Intel Xeon 6980P (Granite Rapids)
Local Memory (Per Socket)	12 Channels, DDR5-6400
CXL Memory (Per Socket)	8 Channels, DDR5-6400

○ Workload

- Intel MLC (Memory Latency Checker)
- VectorDB
- CacheLib

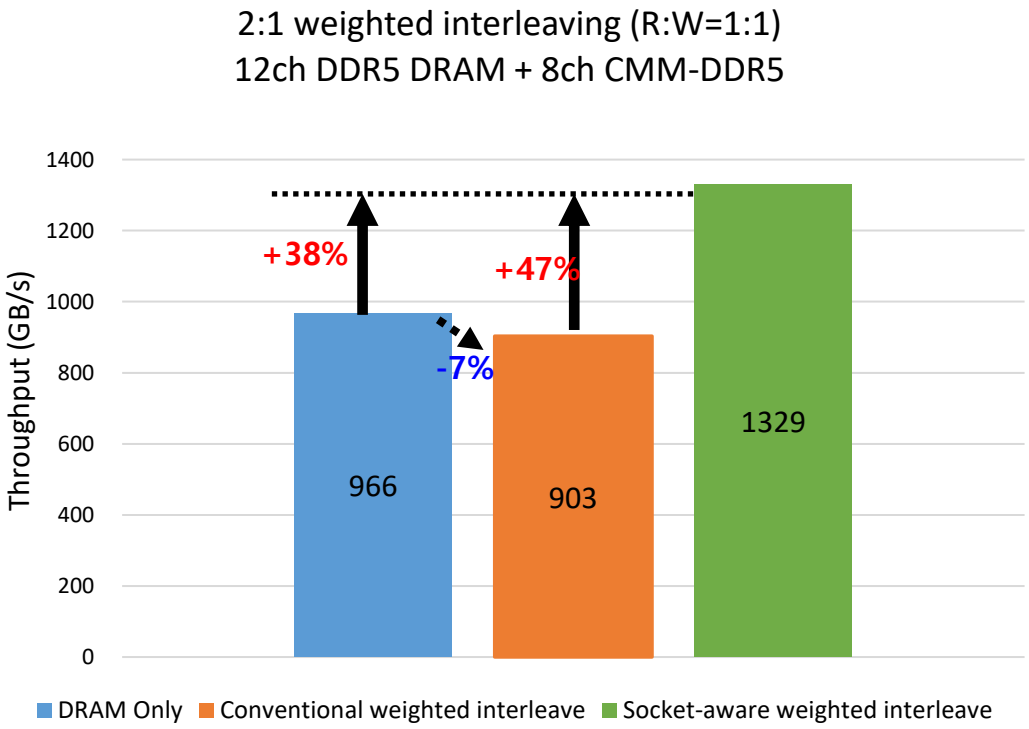
○ Testcases

- DRAM only (Default policy)
- Conventional weighted interleave (Upstream Linux Kernel)
- Socket-aware weighted interleave (HMSDK 4.0)



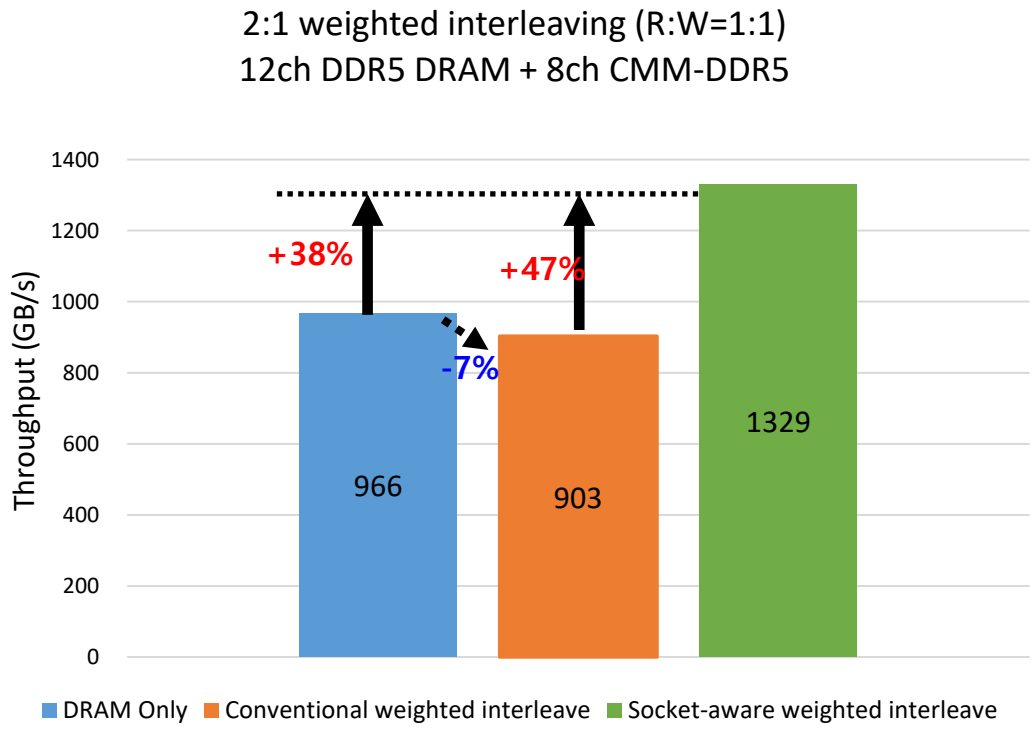
Evaluation - MLC (Throughput)

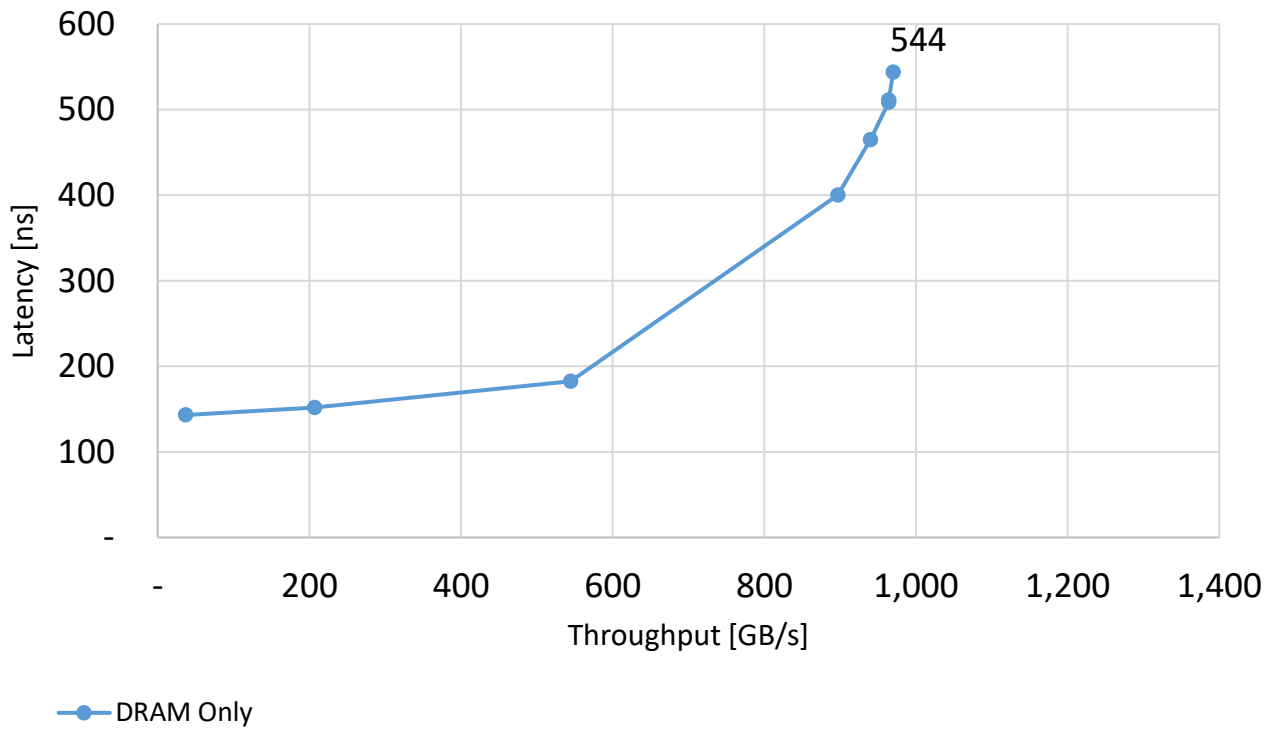
- ○ Maximize system performance by applying CMM hardware alongside Socket-aware Weighted Interleave
- ○ 1.33TB/s industry-leading throughput achieved through effective CMM utilization.
- ○ Enhance Throughput (+38%) by maximizing local resource utilization



Evaluation - MLC (Loaded Latency)

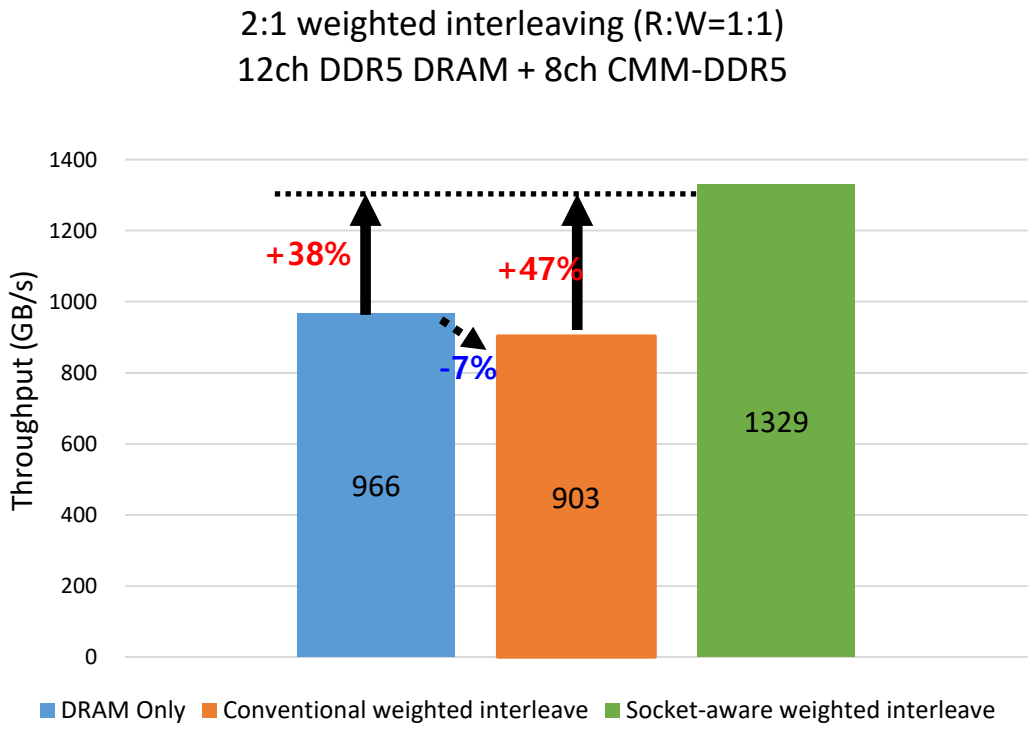
○ Increasing memory pressure leads to higher memory latency

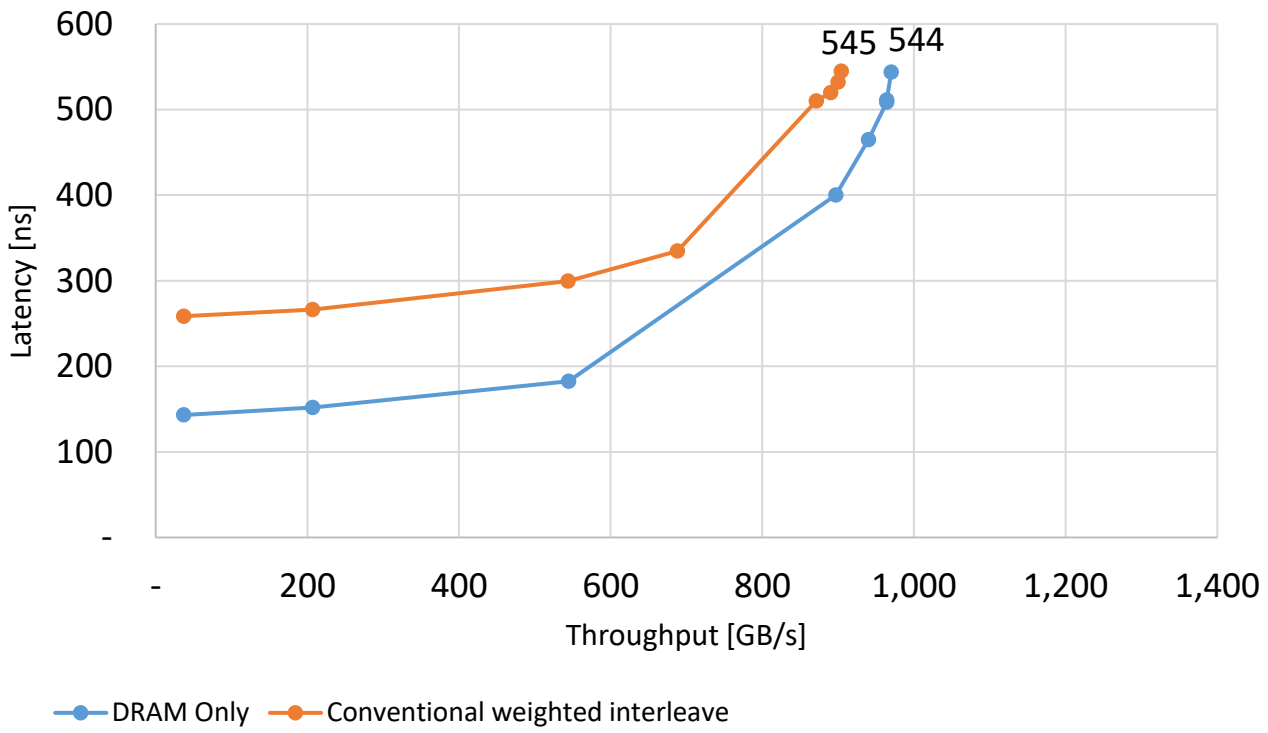




Evaluation - MLC (Loaded Latency)

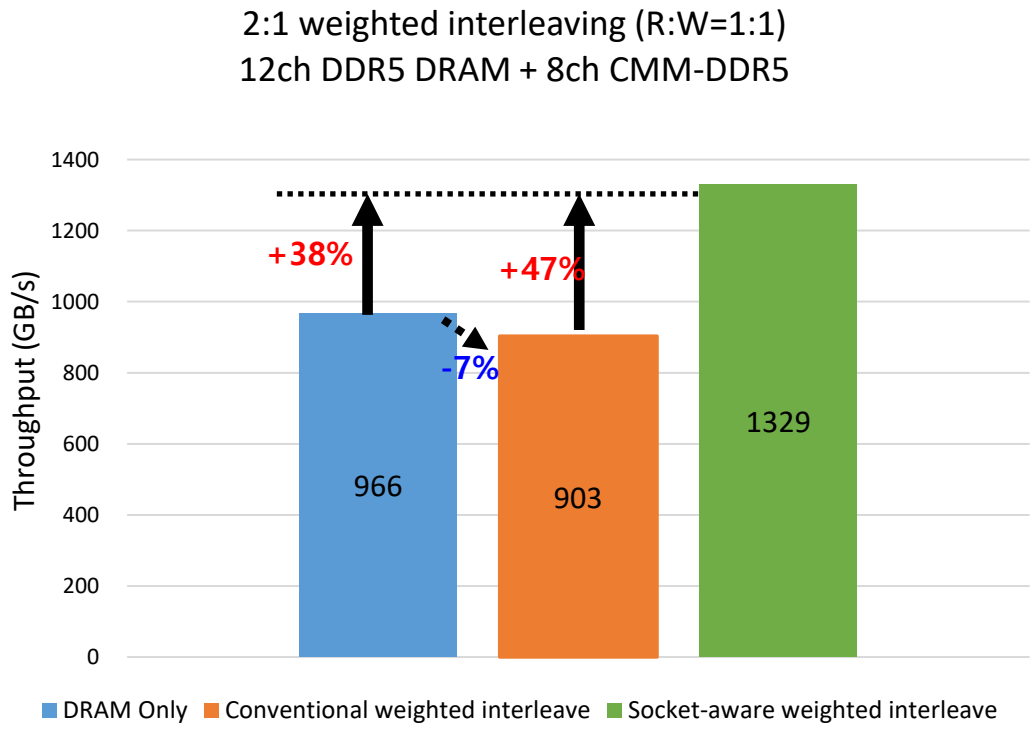
○ Conventional weighted interleave reduces peak throughput and worsen latency under load

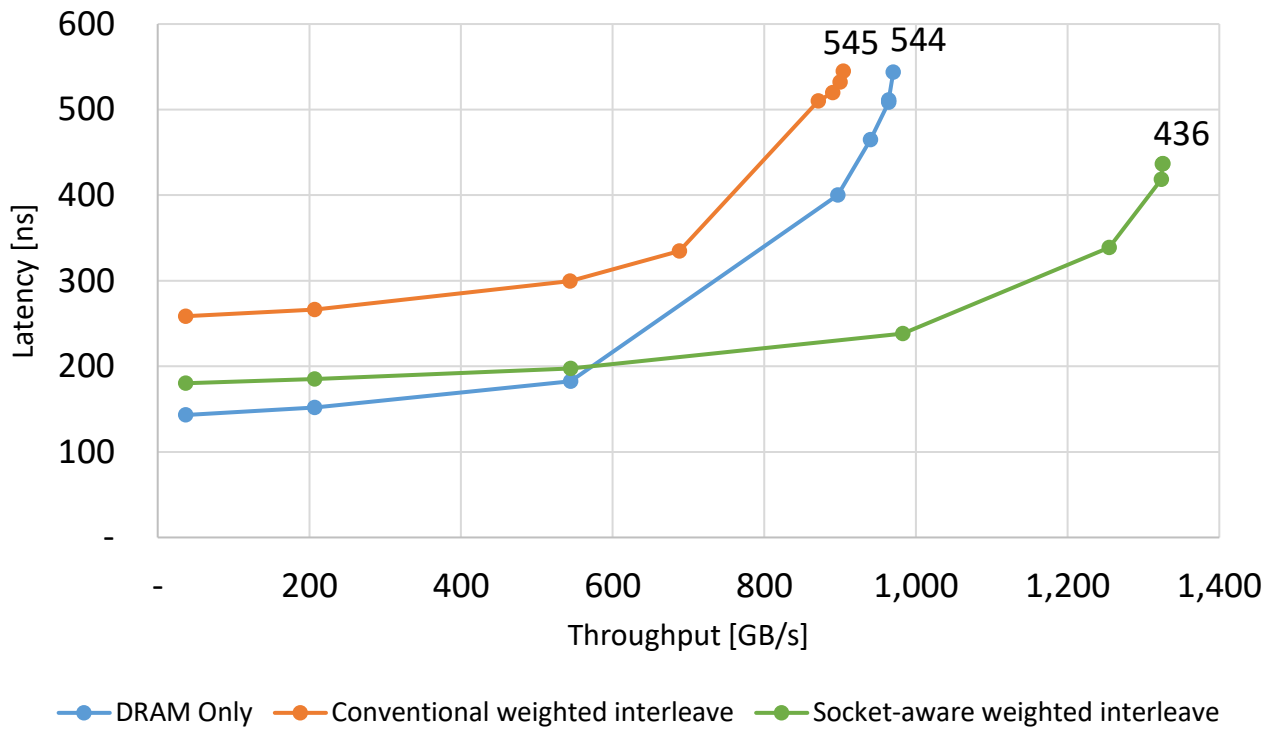




Evaluation - MLC (Loaded Latency)

○ Socket-aware weighted interleave recovers bandwidth while maintaining latency characteristics

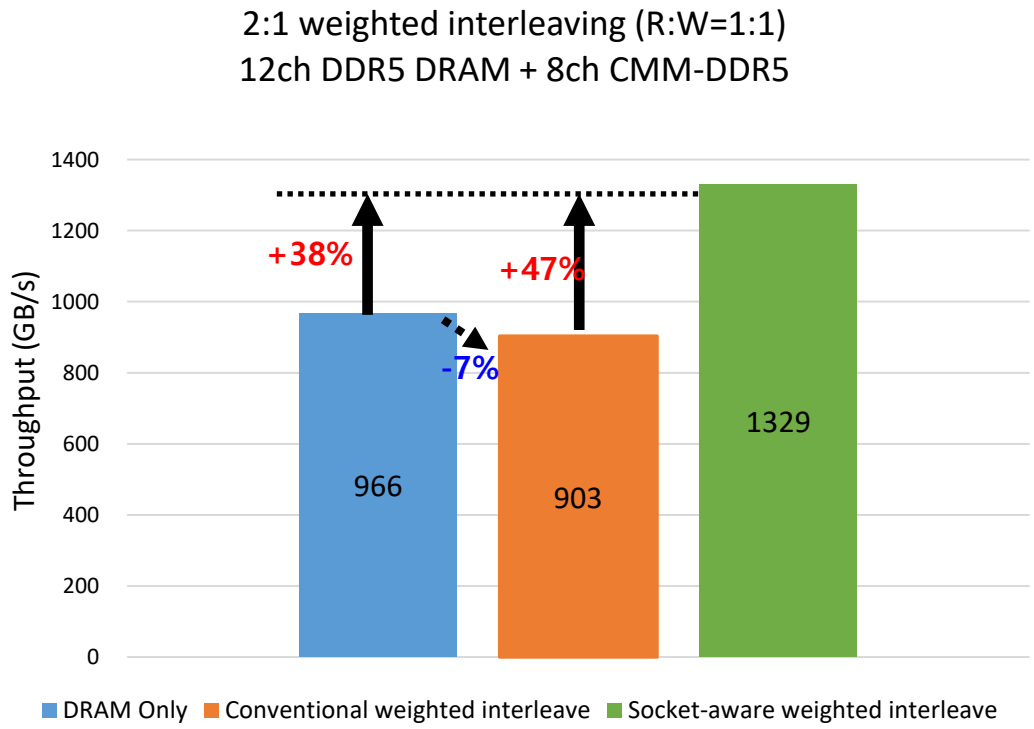


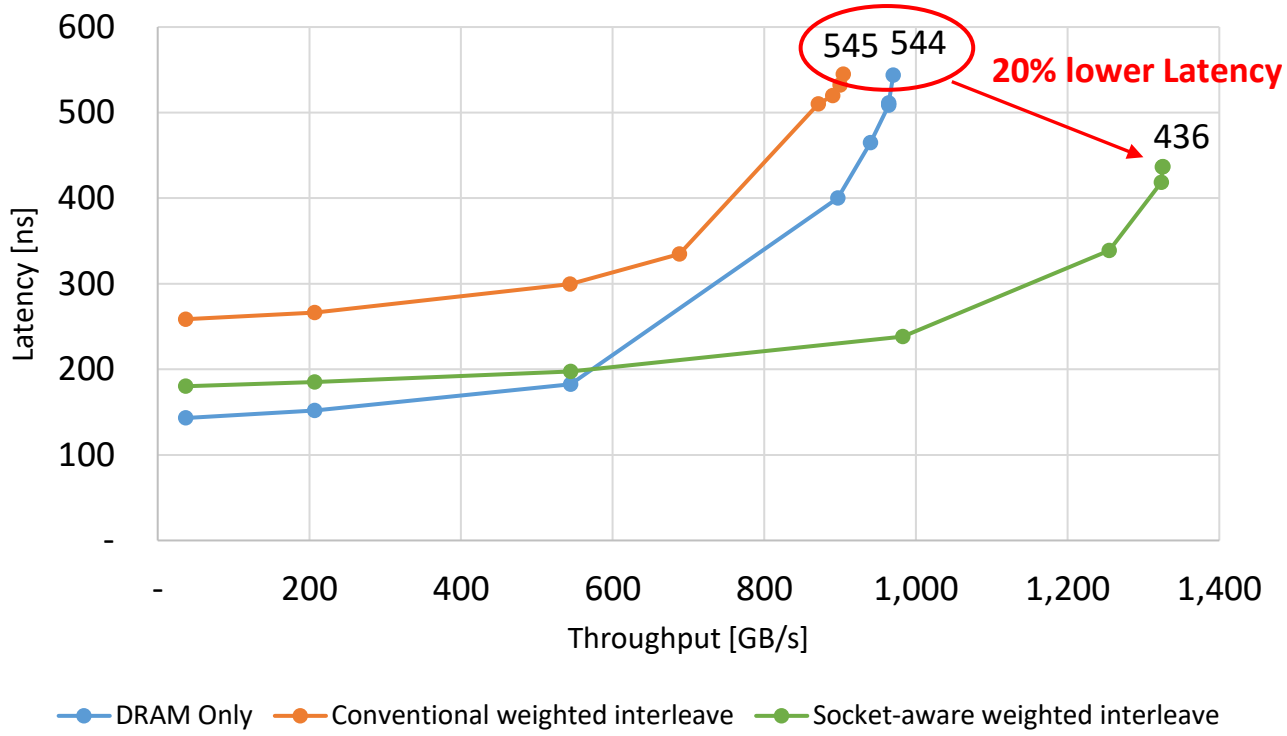


Evaluation - MLC (Loaded Latency)

○ Socket-aware weighted interleave recovers bandwidth while maintaining latency characteristics

- Peak latency reduced by 20% (545ns → 436ns)
- Elbow point shifted right – keeps latency low over wider throughput range





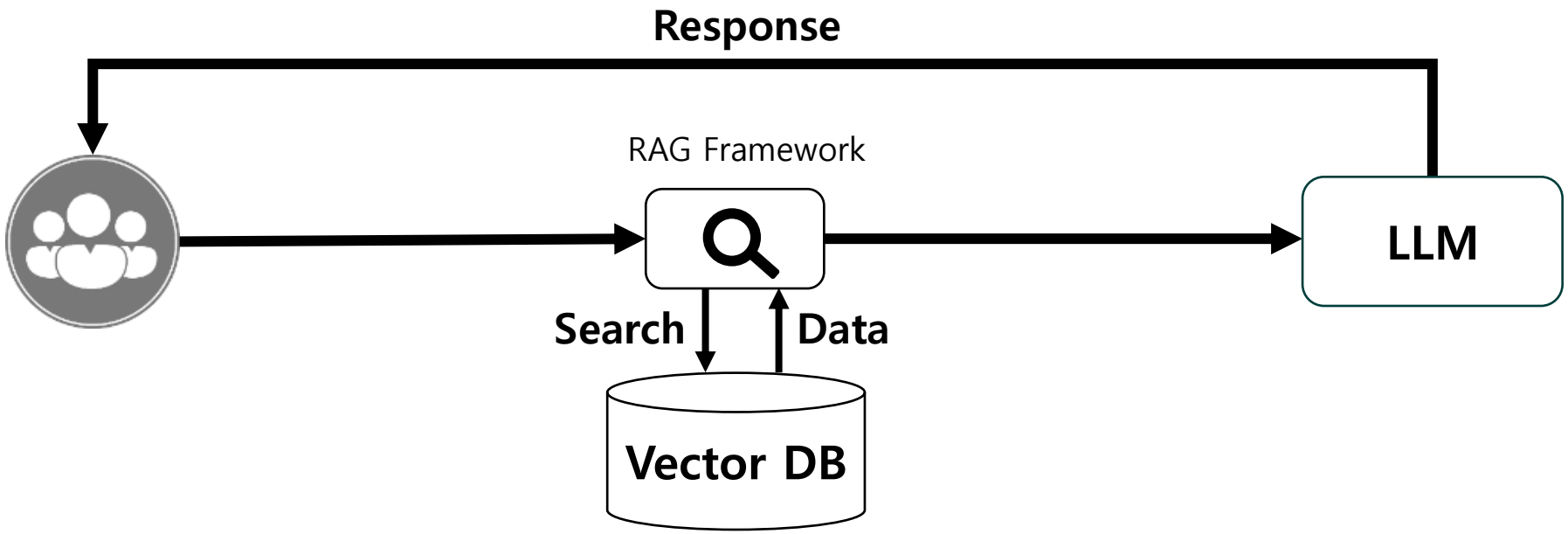
Evaluation - RAG-based LLM Inference (Setup)

○ End-to-end RAG-based LLM inference pipeline : only the VectorDB memory policy varies

- Vector search is the memory-bandwidth-bound stage

  → memory access performance dominates end-to-end latency

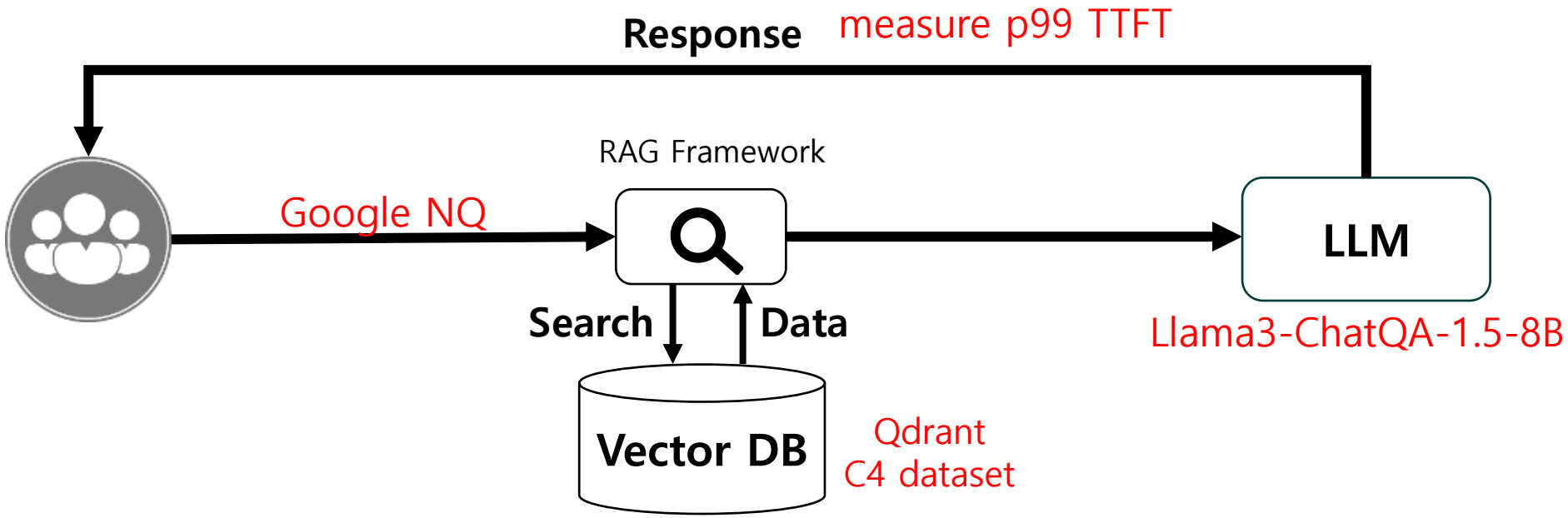
- As load rises, remote-socket access inflates search latency and amplifies TTFT



Evaluation - RAG-based LLM Inference (Setup)

- End-to-end RAG-based LLM inference pipeline : only the VectorDB memory policy varies
  - Vector search is the memory-bandwidth-bound stage

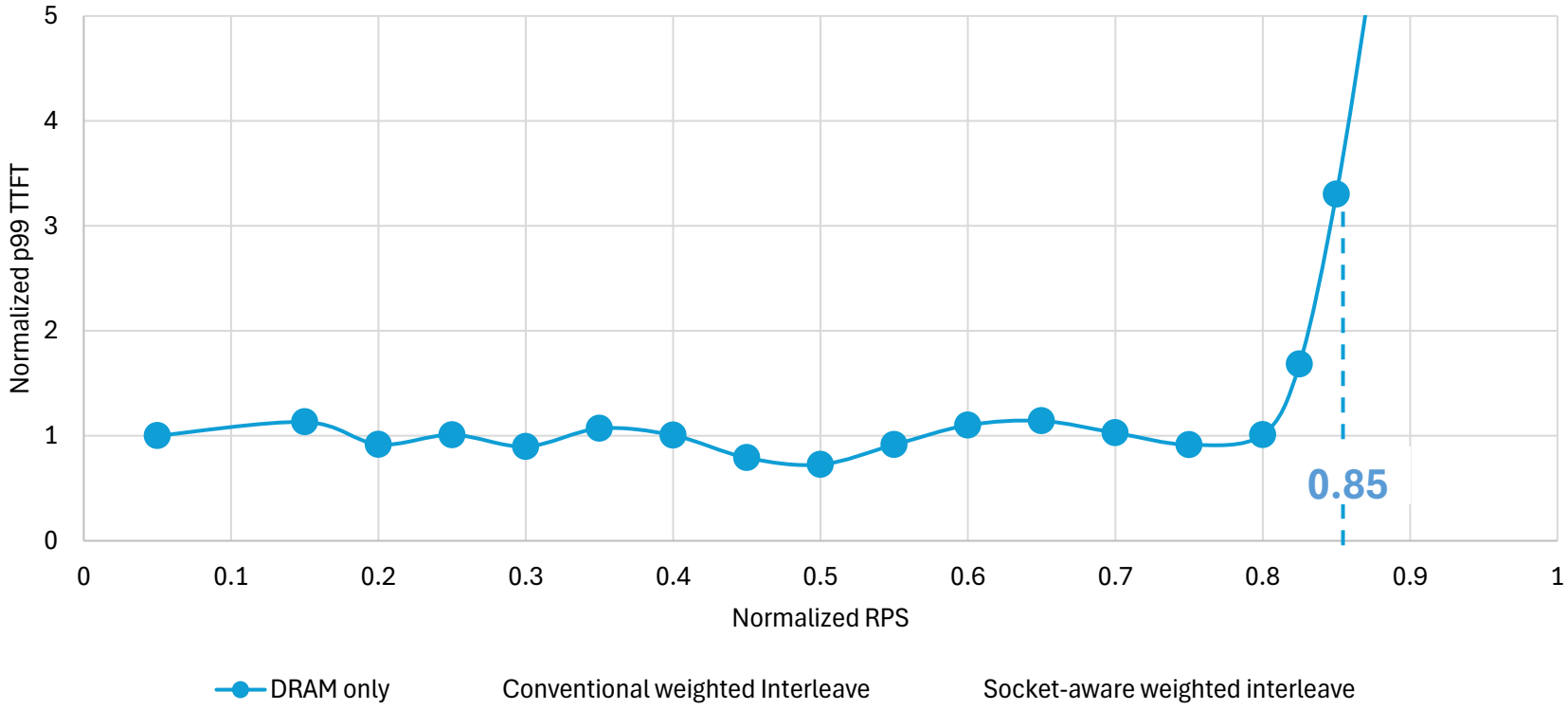
    → memory access performance dominates end-to-end latency
  - As load rises, remote-socket access inflates search latency and amplifies TTFT
- Metric: p99 TTFT (first-token tail latency) under increasing request rate (RPS)
- Criterion: highest RPS within the SLA bound = max sustainable RPS



Evaluation - RAG-based LLM Inference (p99 TTFT)

○ Measure max sustainable RPS within the SLA bound

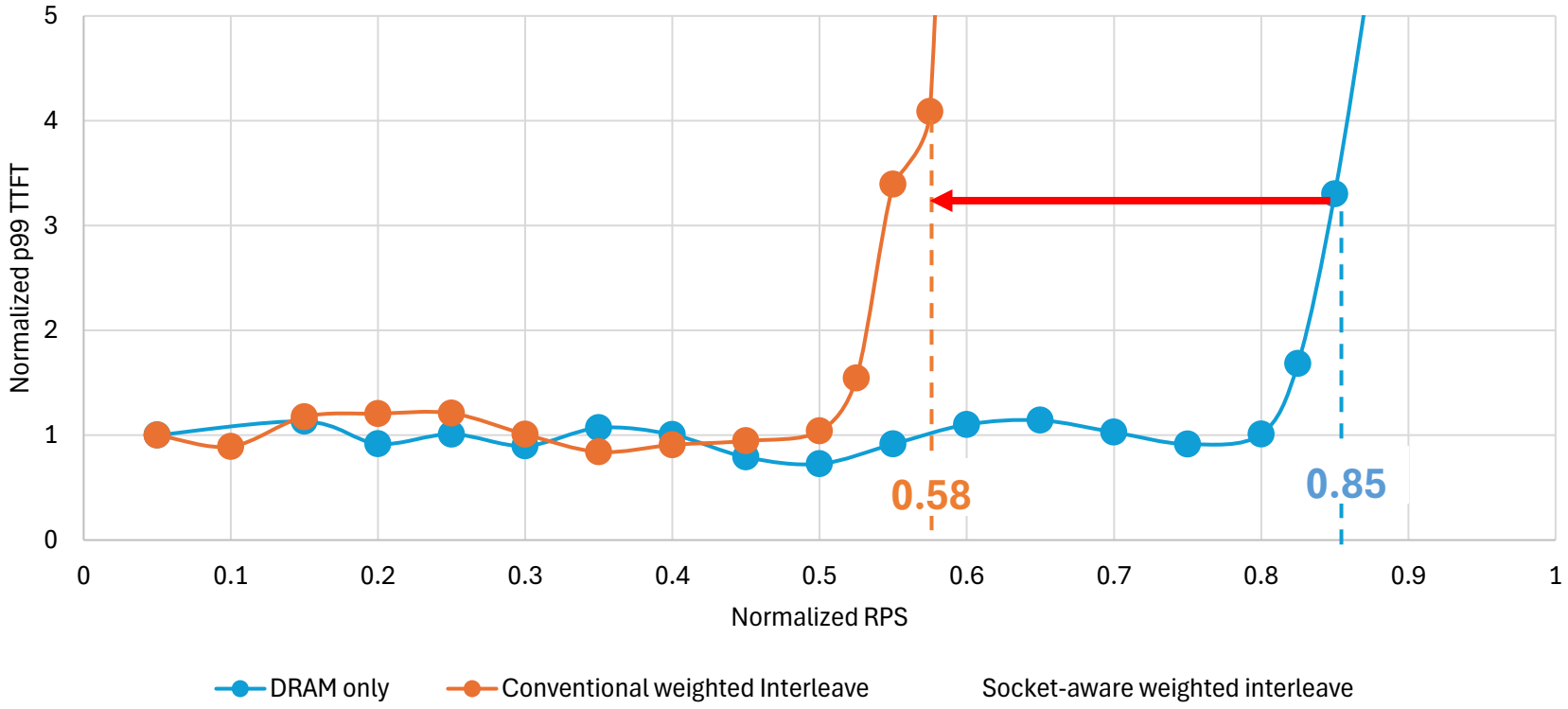
- DRAM only: 0.85



Evaluation - RAG-based LLM Inference (p99 TTFT)

○ Measure max sustainable RPS within the SLA bound

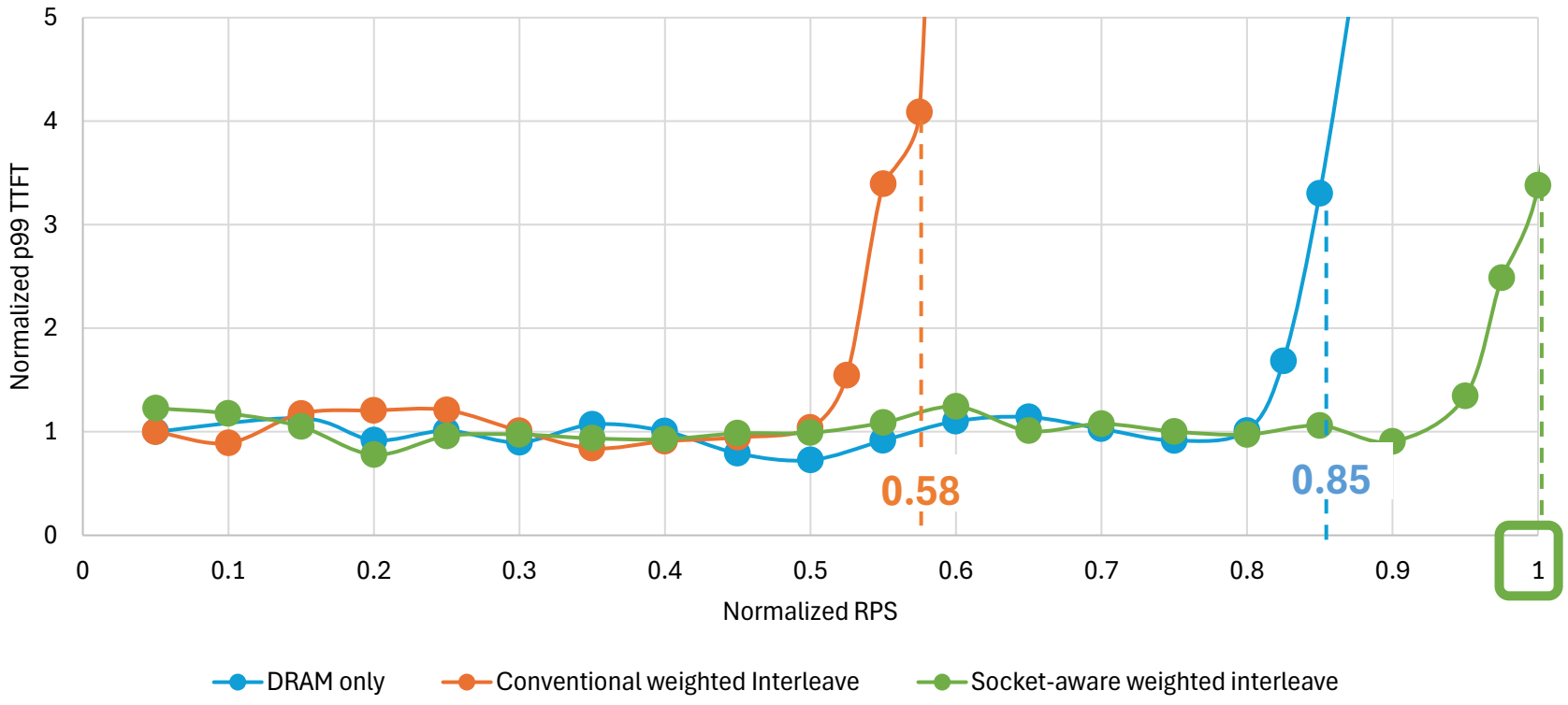
- Conventional weighted interleave: 0.58
  - Exceeds the SLA latency bound at a lower RPS than DRAM only



Evaluation - RAG-based LLM Inference (p99 TTFT)

○ Measure max sustainable RPS within the SLA bound

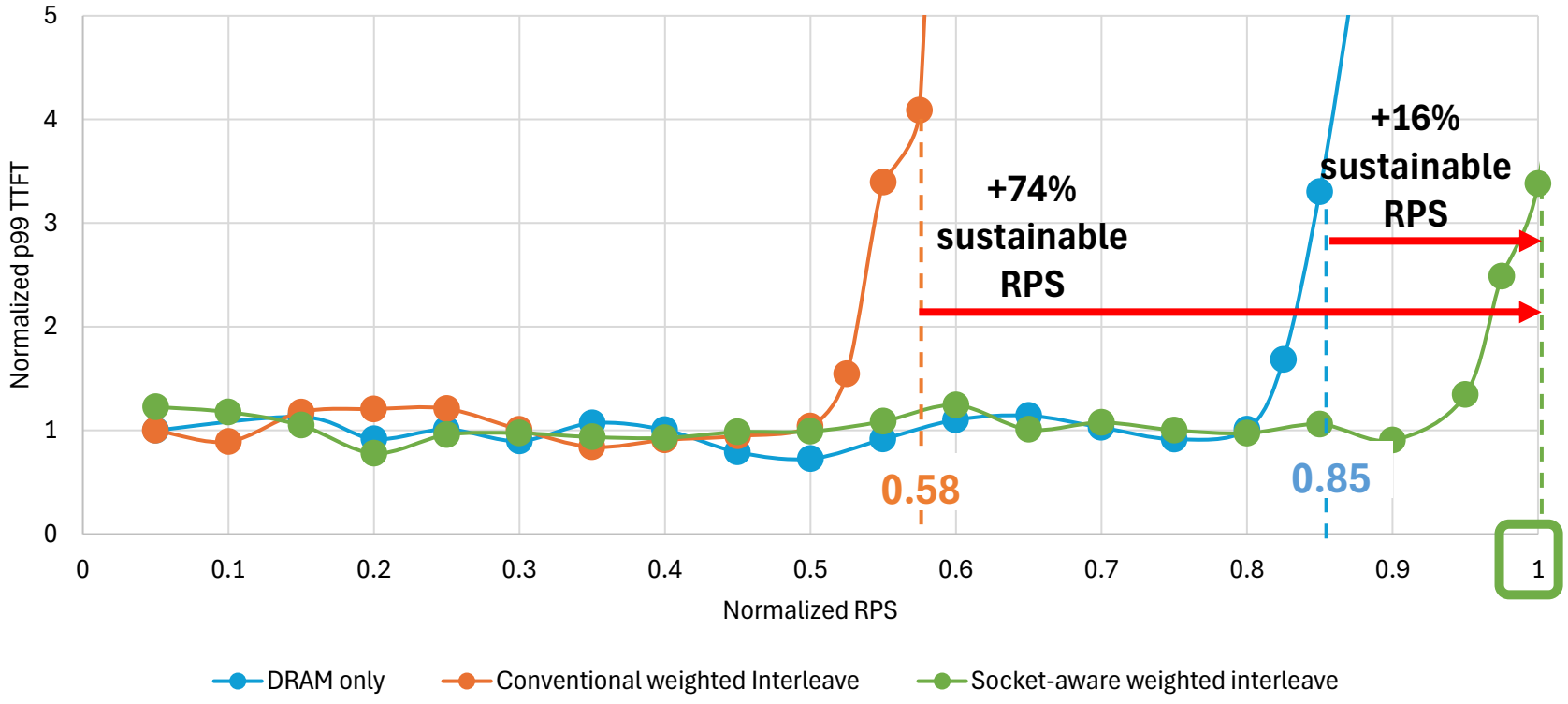
- Socket-aware weighted interleave: 1



Evaluation - RAG-based LLM Inference (p99 TTFT)

○ Measure max sustainable RPS within the SLA bound

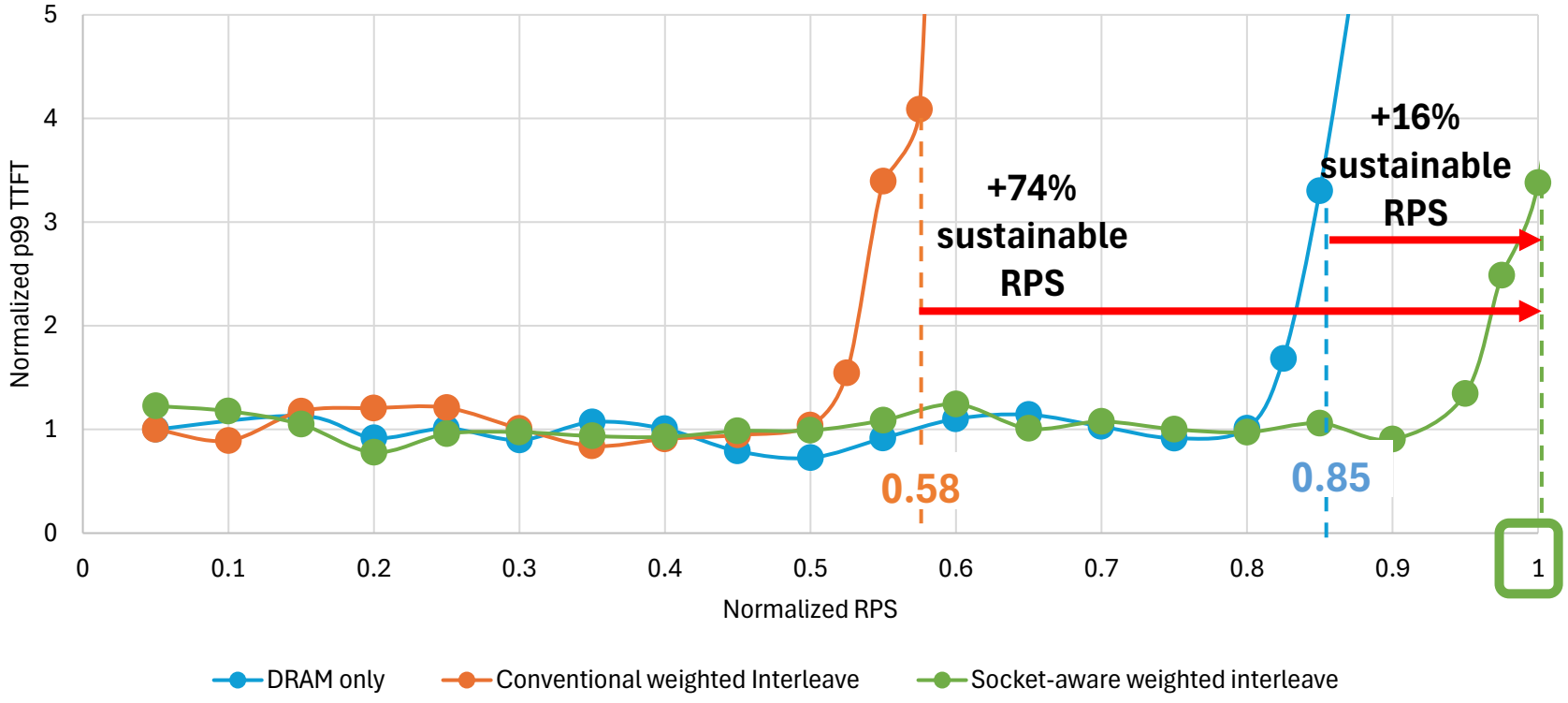
- Socket-aware weighted interleave: 1
  - +16% over DRAM only, +74% Conventional weighted interleave



Evaluation - RAG-based LLM Inference (p99 TTFT)

○ Measure max sustainable RPS within the SLA bound

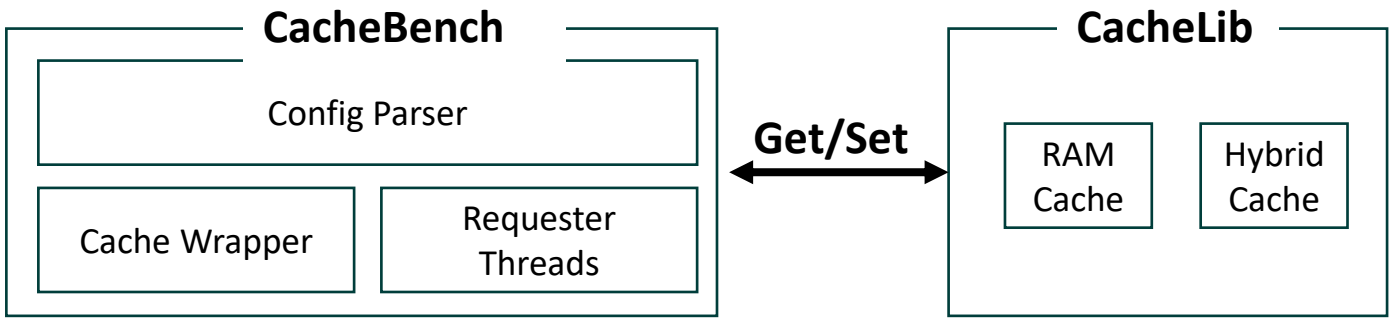
- Socket-aware weighted interleave: 1
  - +16% over DRAM only, +74% Conventional weighted interleave
- **Tail latency stays flat across a far wider load range**



Evaluation - CacheLib Overview

- CacheLib is an open-source C++ library developed by Meta
  - High-performance, in-process key-value caching framework
  - https://cachelib.org

- CacheBench is a benchmark and stress-testing tool designed to evaluate the performance of CacheLib
  - Represents real-world Meta caching workloads
    - Pre-defined workloads: social graphs, CDN, key-value storage



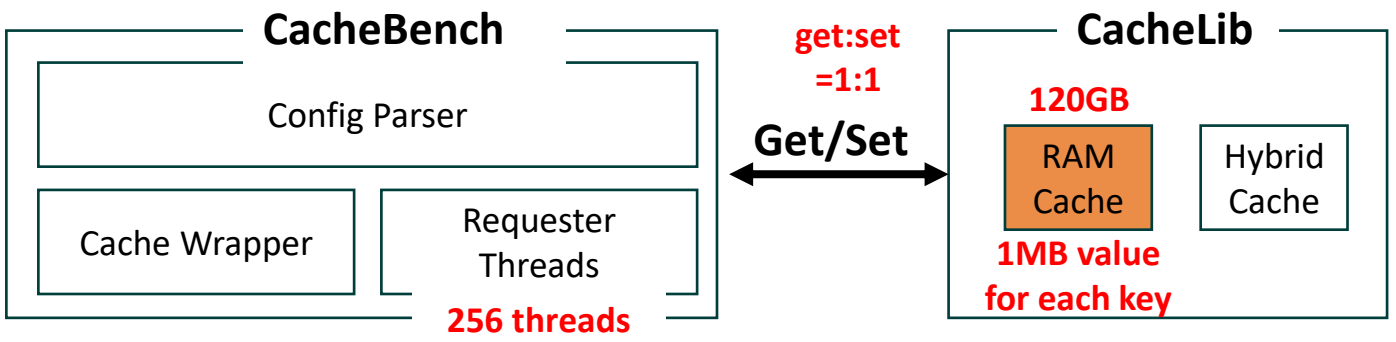
Evaluation - CacheLib Overview

○ CacheLib is an open-source C++ library developed by Meta

- High-performance, in-process key-value caching framework
- https://cachelib.org

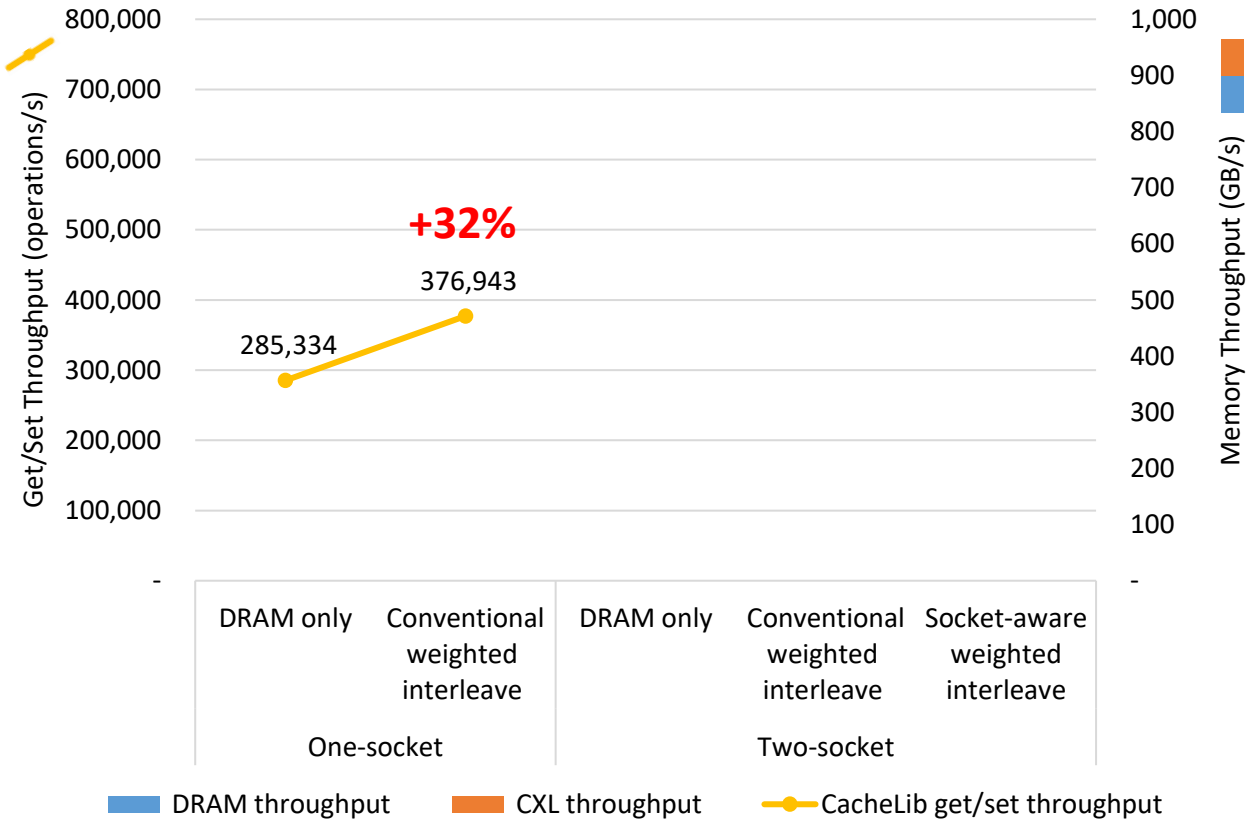
○ CacheBench is a benchmark and stress-testing tool designed to evaluate the performance of CacheLib

- Represents real-world Meta caching workloads
  - Pre-defined workloads: social graphs, CDN, key-value storage
- Large working sets and heavy concurrent cache request require high memory bandwidth



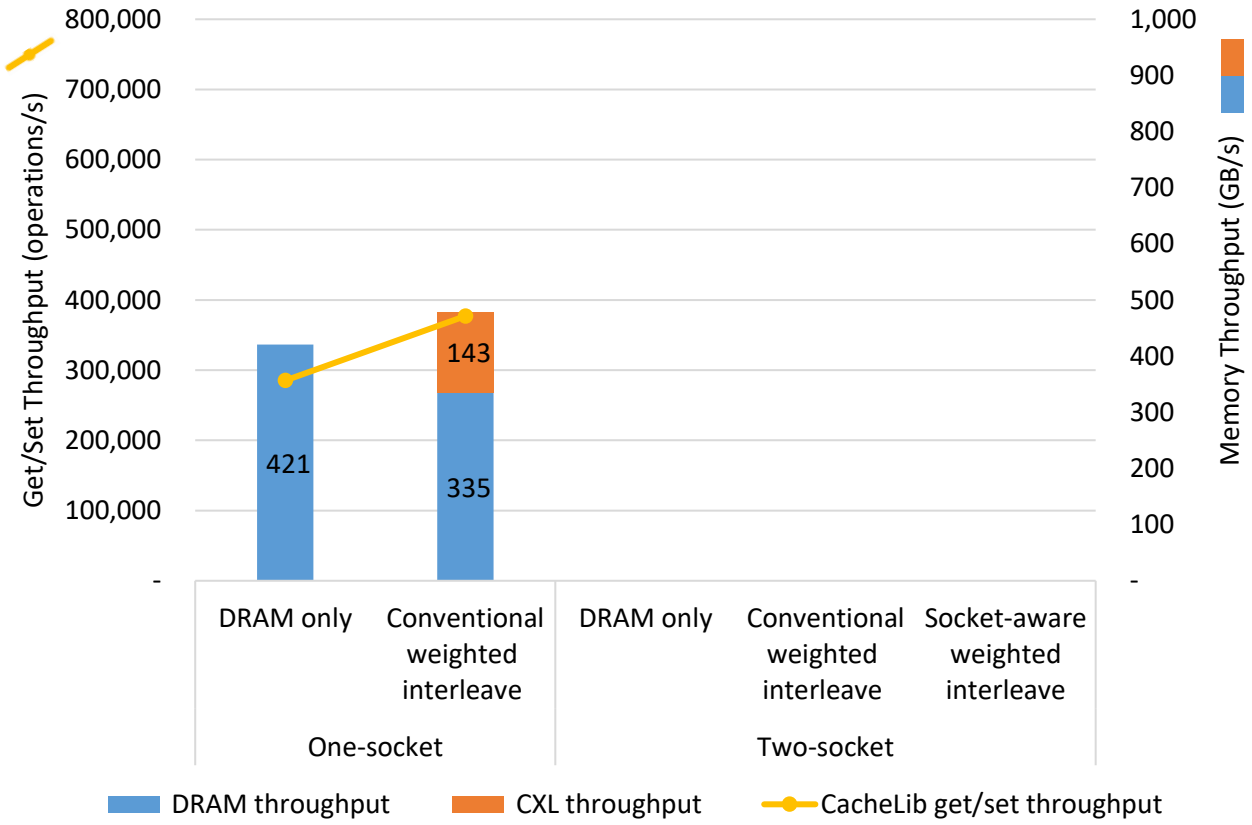
Evaluation - CacheLib (Single-Socket Throughput)

○ With single-socket weighted interleave, CacheLib throughput improves by 32%.



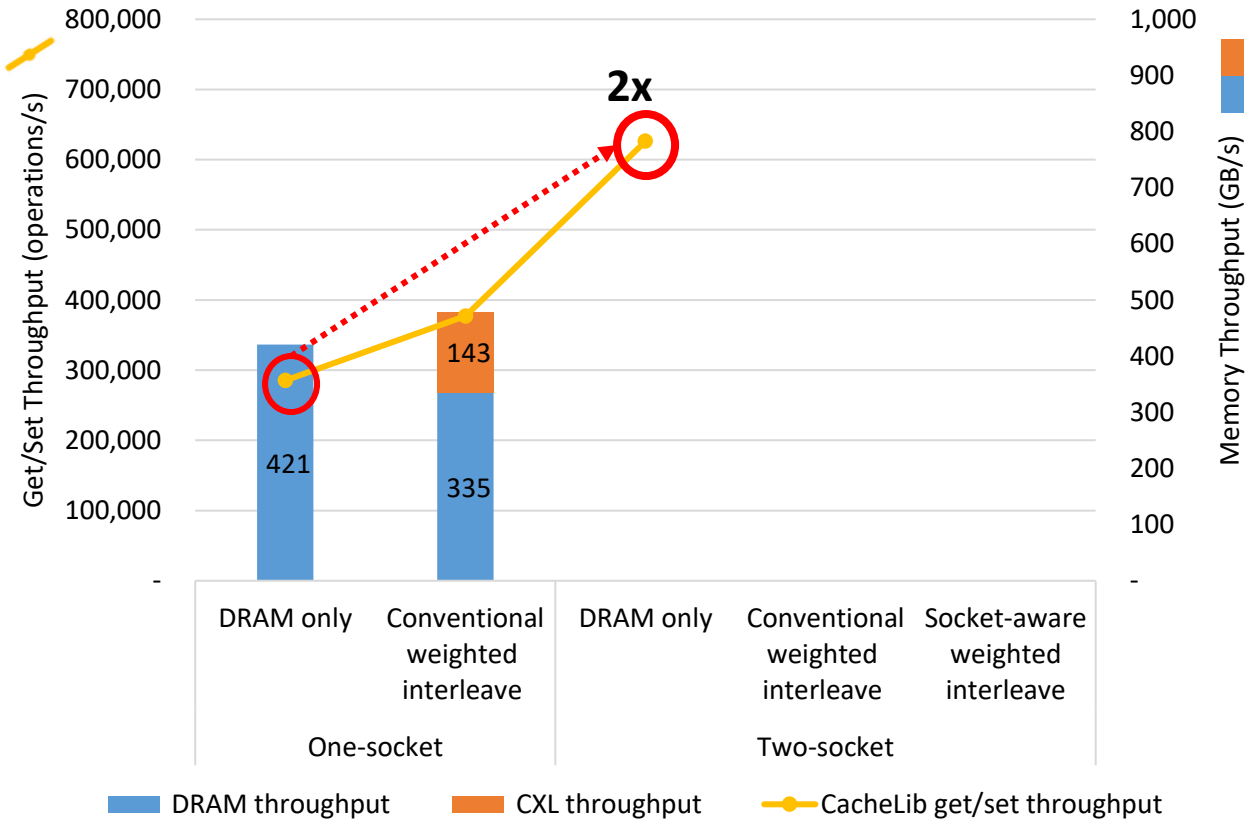
Evaluation - CacheLib (Single-Socket Throughput)

- With single-socket weighted interleave, CacheLib throughput improves by 32%.
- Memory throughput increases from 421GB/s to 478GB/s



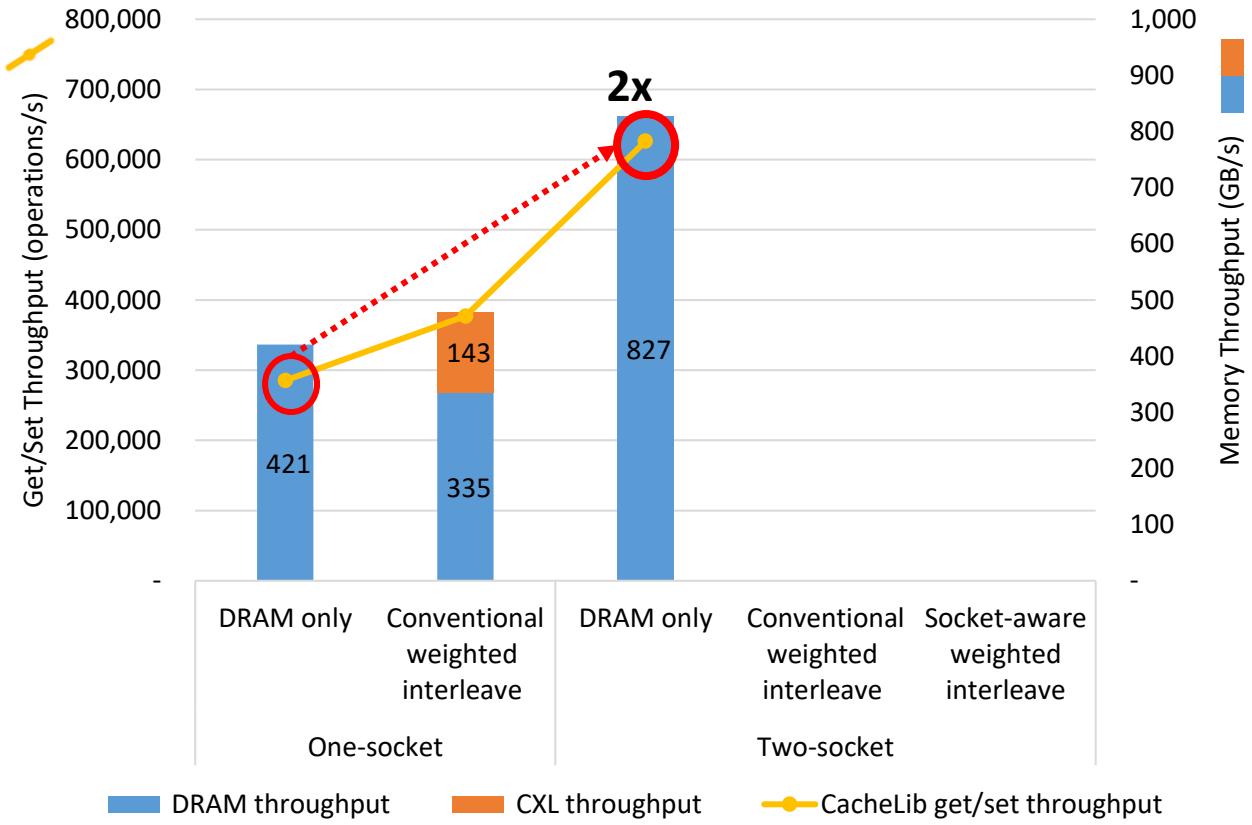
Evaluation - CacheLib (Dual-Socket Throughput)

○ In dual-socket, CacheLib throughput doubles compared to single-socket



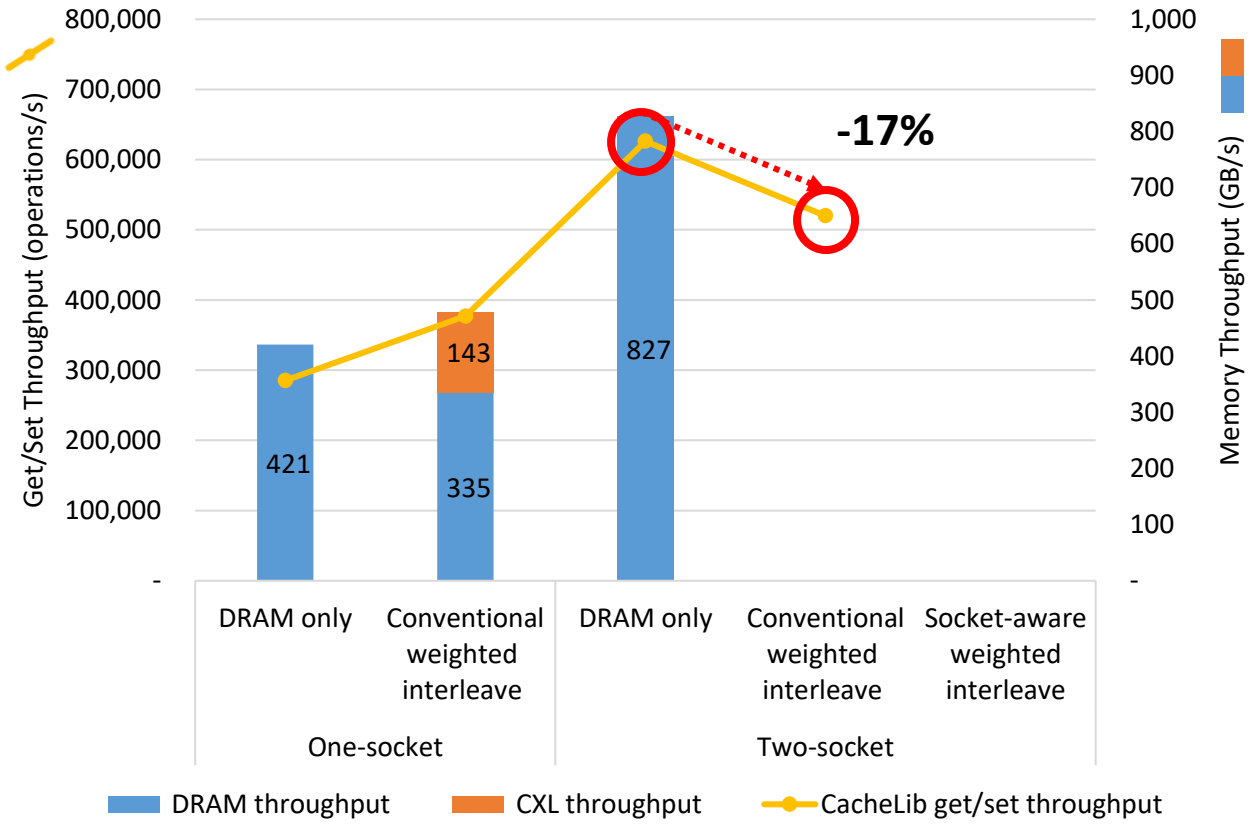
Evaluation - CacheLib (Dual-Socket Throughput)

- In dual-socket, CacheLib throughput doubles compared to single-socket
- Memory throughput doubles as well



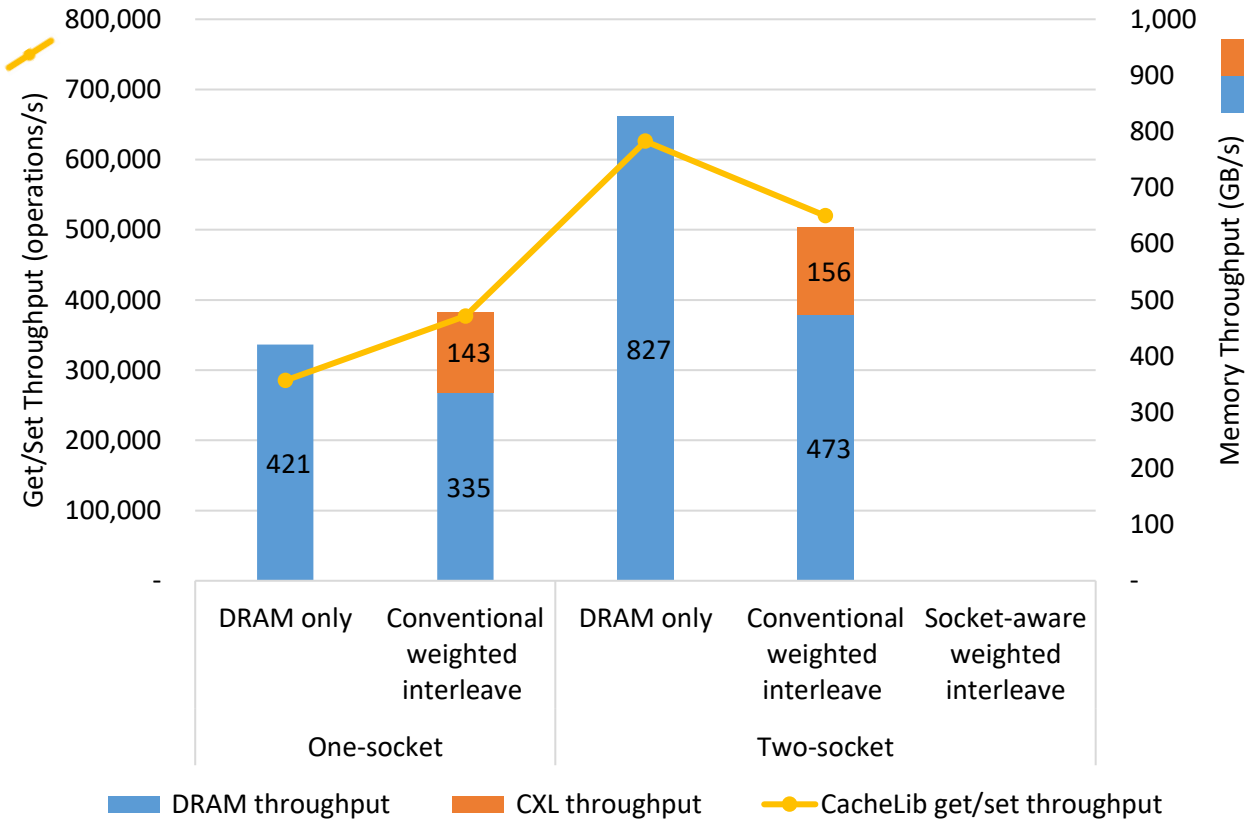
Evaluation - CacheLib (Dual-Socket Bottleneck)

○ However, weighted interleave degrades performance by 17%



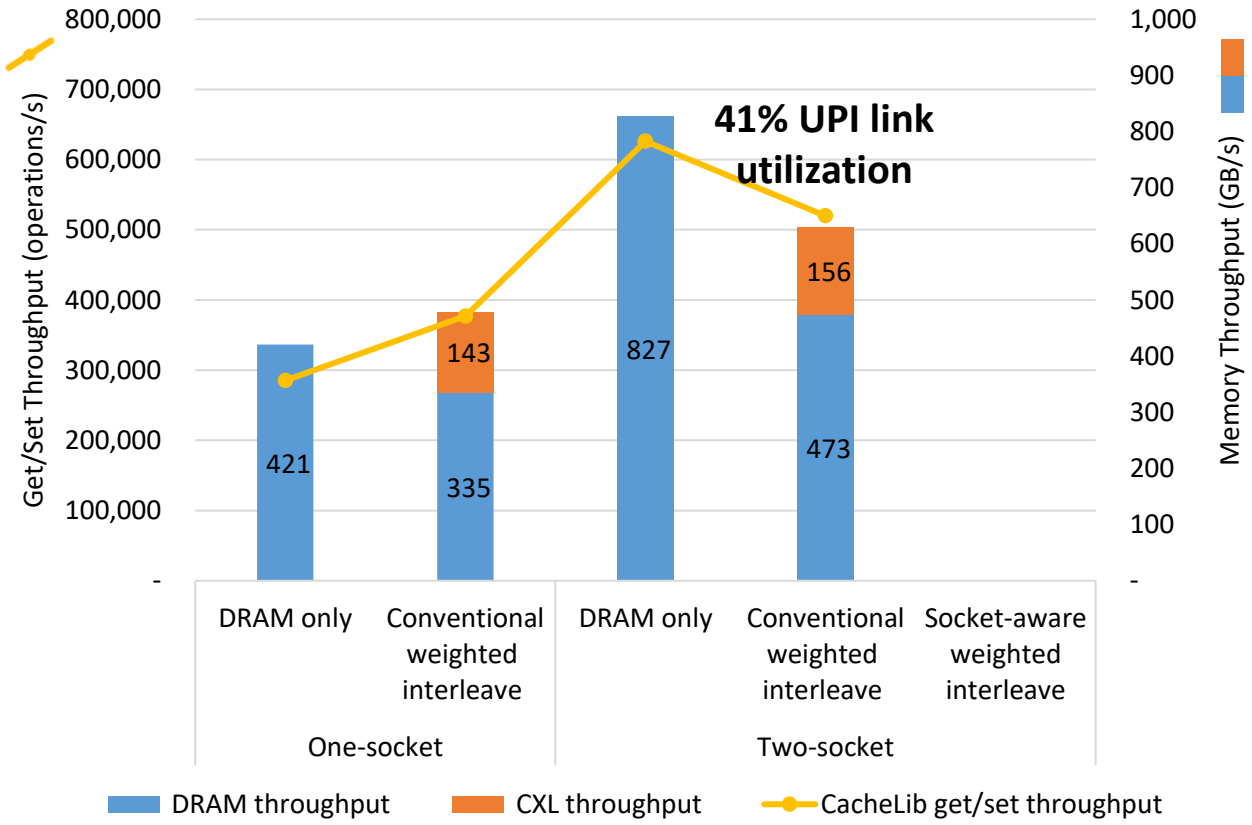
Evaluation - CacheLib (Dual-Socket Bottleneck)

○ Memory throughput drops 24%: 827GB/s ➔ 629GB/s



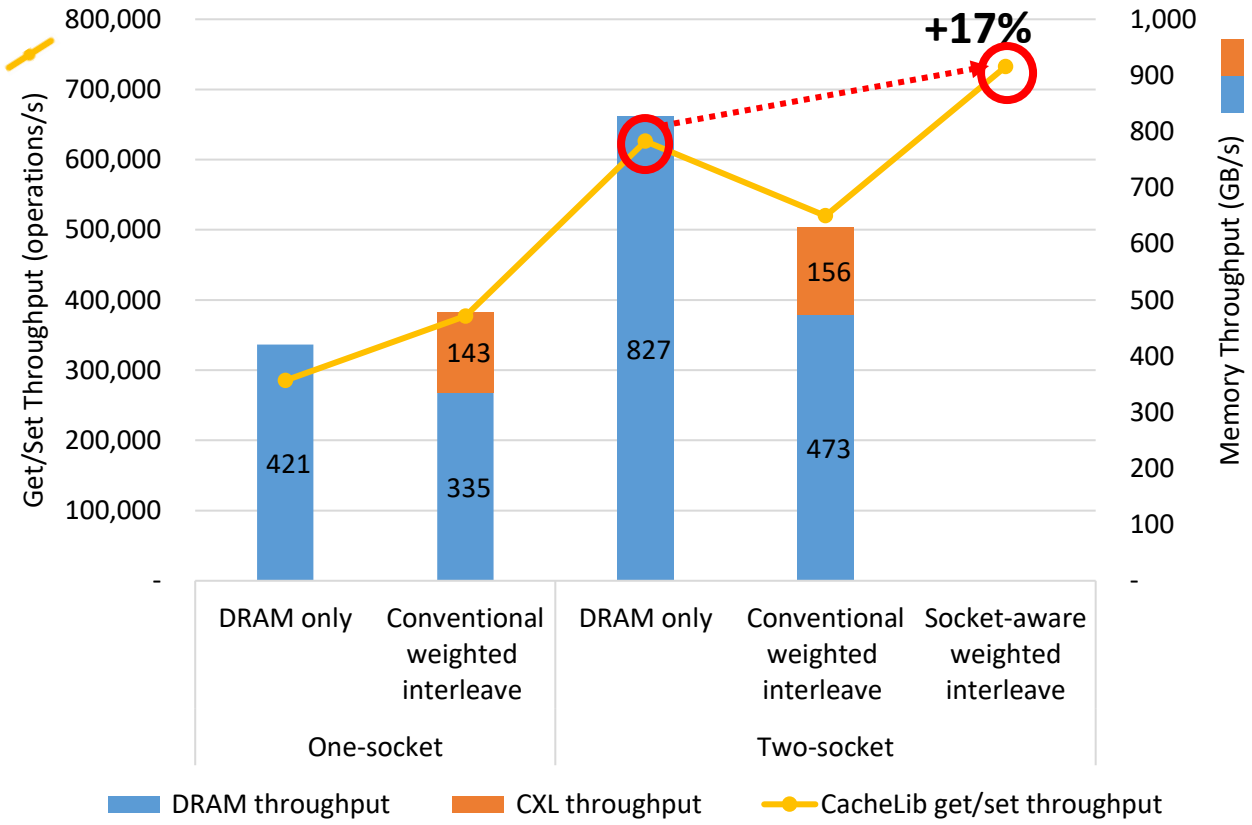
Evaluation - CacheLib (Dual-Socket Bottleneck)

- Memory throughput drops 24%: 827GB/s ➔ 629GB/s
- Root cause: 41% UPI link utilization from remote memory access
  - Remote memory access accounts for 67% of local memory access



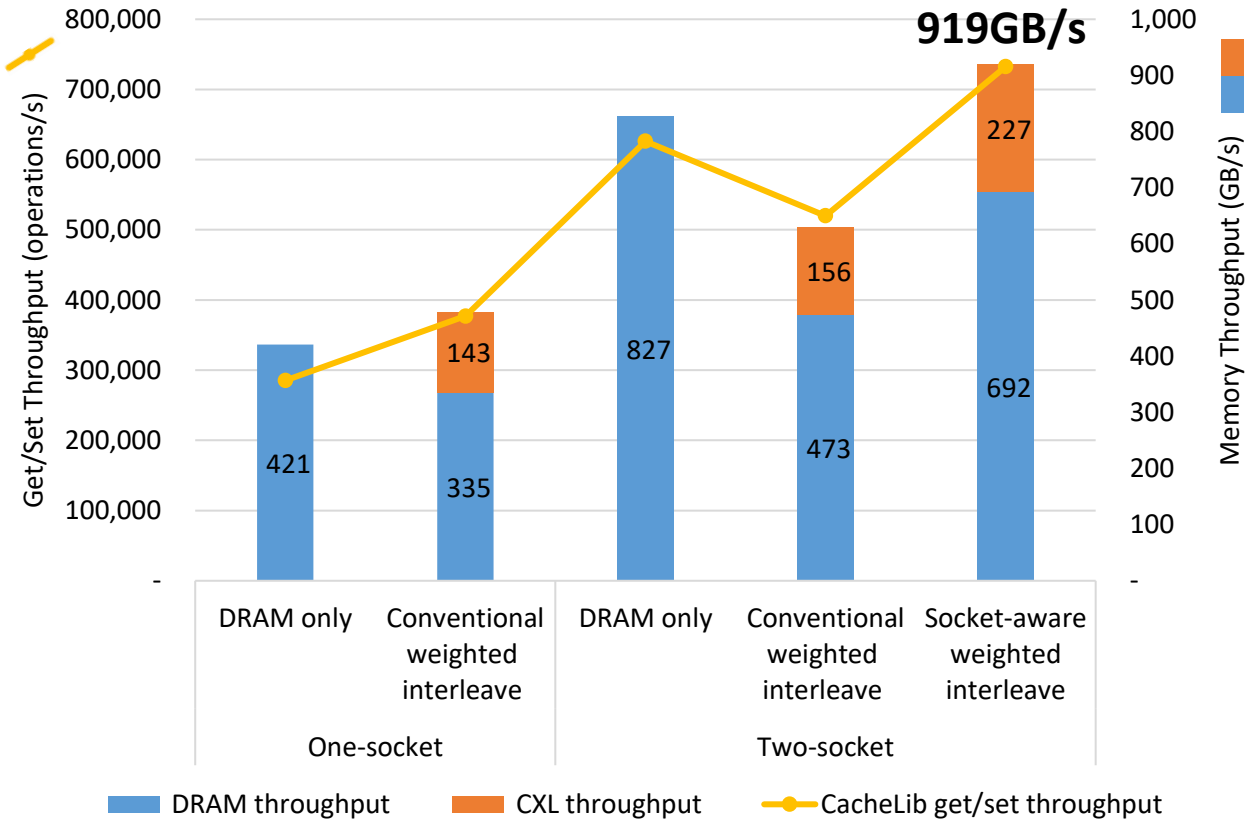
Evaluation - CacheLib (Socket-aware weighted interleave)

○ Socket-aware weighted interleave outperforms conventional weighted interleave



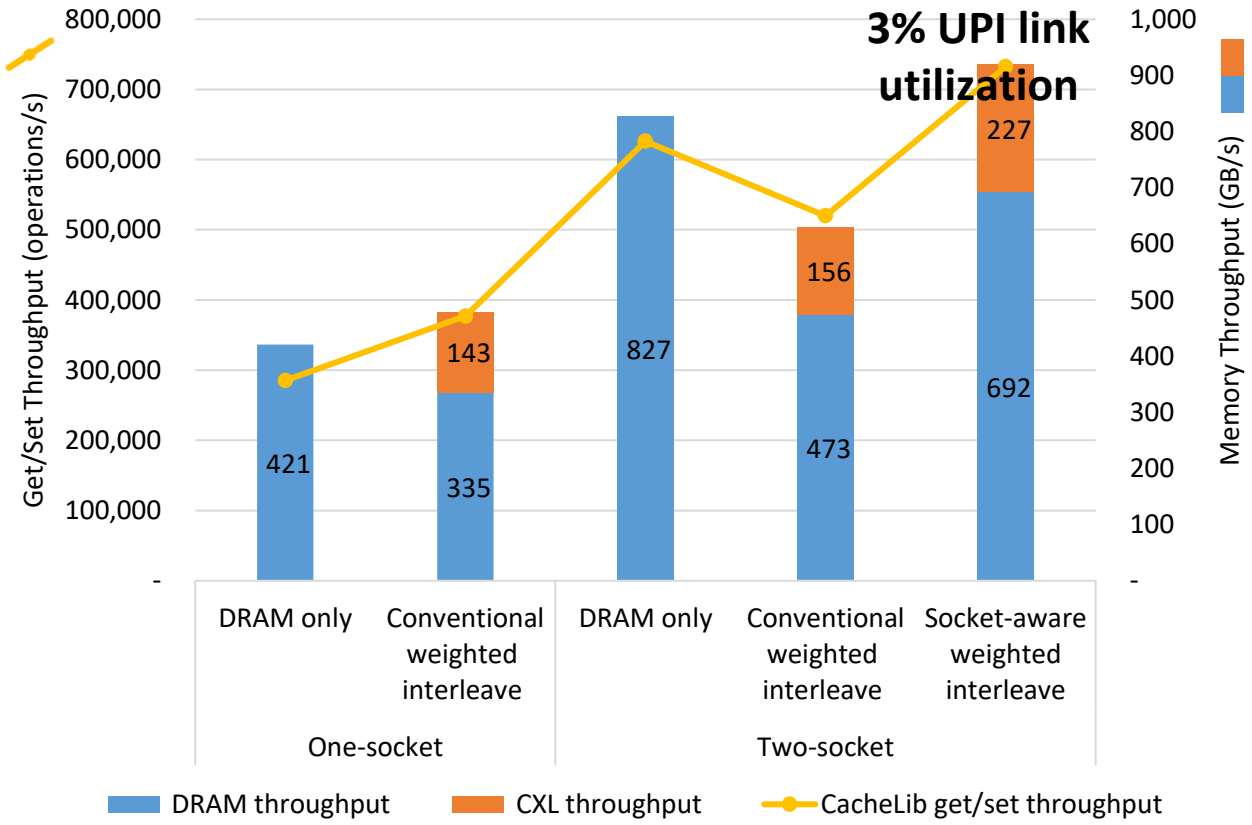
Evaluation - CacheLib (Socket-aware weighted interleave)

- Socket-aware weighted interleave outperforms conventional weighted interleave
- Memory throughput increases to 919GB/s



Evaluation - CacheLib (Socket-aware weighted interleave)

- Socket-aware weighted interleave outperforms conventional weighted interleave
- Memory throughput increases to 919GB/s
- UPI link utilization drops from 41% to 3%



Contents

Conclusion

○ **The weighted interleave in upstream Linux is not designed for multi-socket systems**

- A single global weight table inefficiently distribute pages making UPI traffics across multiple sockets.

○ **Socket-aware weighted interleave maximizes system bandwidth in multi-socket systems**

- by binding allocations to socket-local nodes, minimizing remote traffic across different sockets.
- It empowers multi-socket systems with CXL memory by fully utilizing its system bandwidth

○ **It shows a terabyte scale system bandwidth result with CXL memory devices**

- Achieved **1.33 TB/s effective bandwidth** with Intel MLC measurement
  - by fully utilizing 40 channels of memory modules (24ch DRAM and 16ch CXL memory)
- Beneficial in real world bandwidth-hungry workloads such as RAG-based LLM inference and CacheLib.

○ **Socket-aware weighted interleave is posted on LKML and under community review**

- https://lore.kernel.org/linux-cxl/20260806080936.421-1-rakie.kim@sk.com

THANK YOU