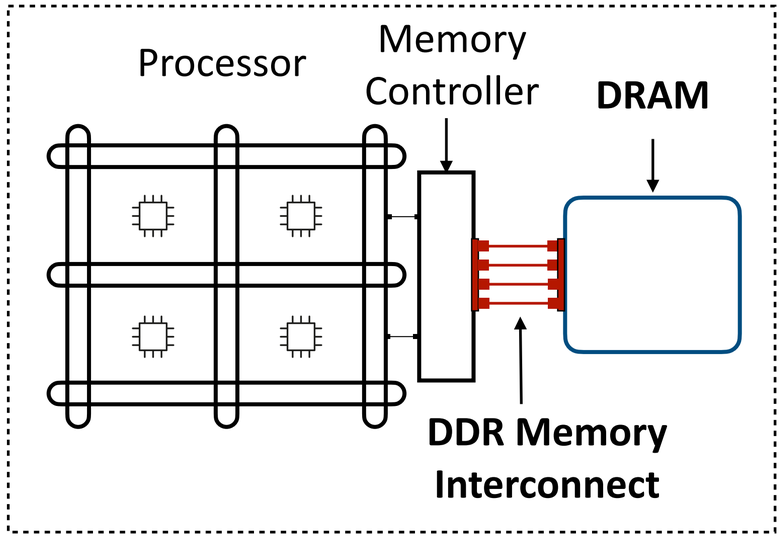
Tiered Memory Management: Access Latency is the Key!

Midhul Vuppalapati

Rachit Agarwal

Cornell University

Classical memory architecture in servers



Modern applications demand larger memory capacity and bandwidth

In-memory caches, graph processing engines, ML frameworks, …….

Memory contributes to large fractions of datacenter cost

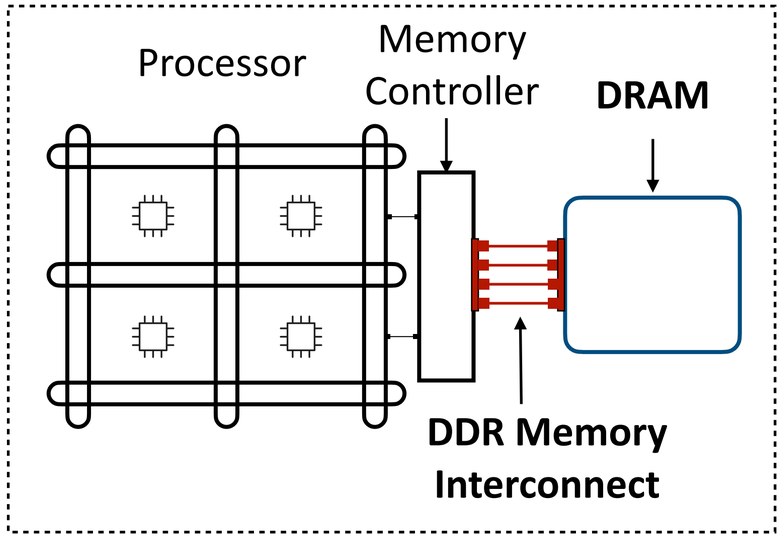
37% of Meta's server costs

50% of Microsoft Azure server costs

Classical memory architecture: DRAM connected via DDR memory interconnect

Classical memory architecture has reached scaling limits

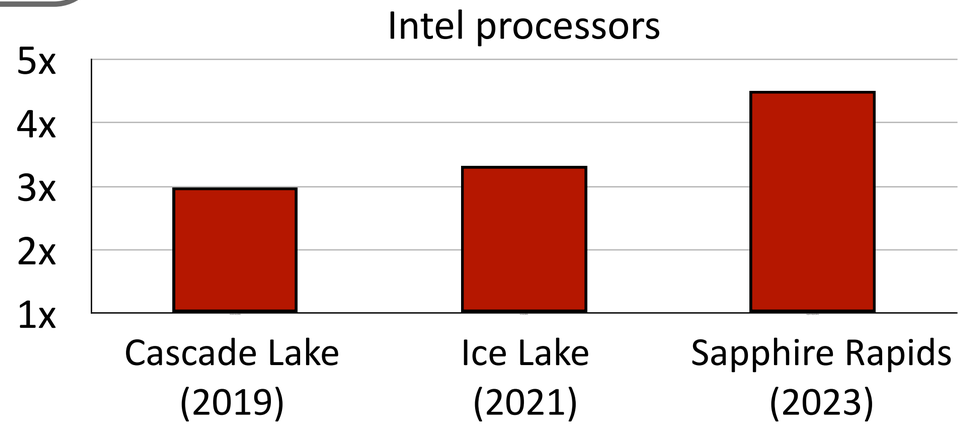
Memory interconnect is increasingly oversubscribed



Maximum per-core load × # of cores

Total memory load processor can generate

Memory bandwidth



AMD Genoa (2023): **9.37x**

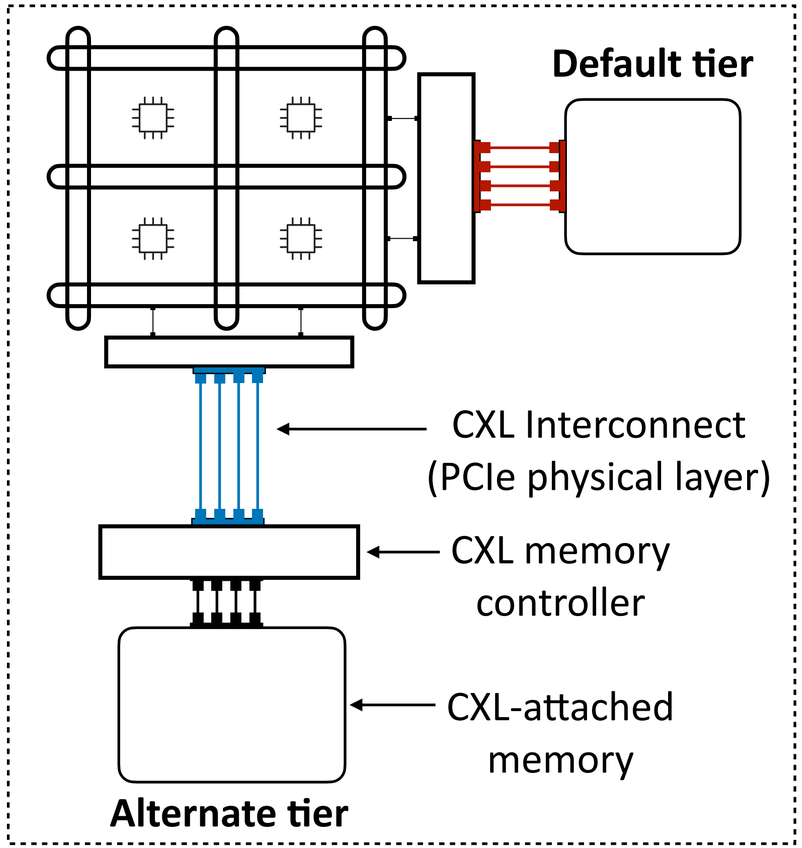
Processor core counts, concurrency-per-core are increasing

DDR memory interconnect bandwidth is difficult to scale

Processor pin and signaling limitations

Emergence of tiered memory architectures

New memory tiers via alternate interconnects



Example alternate interconnect: Compute Express Link (CXL)

Transparent, cache-coherent access to memory (via standard load/store)

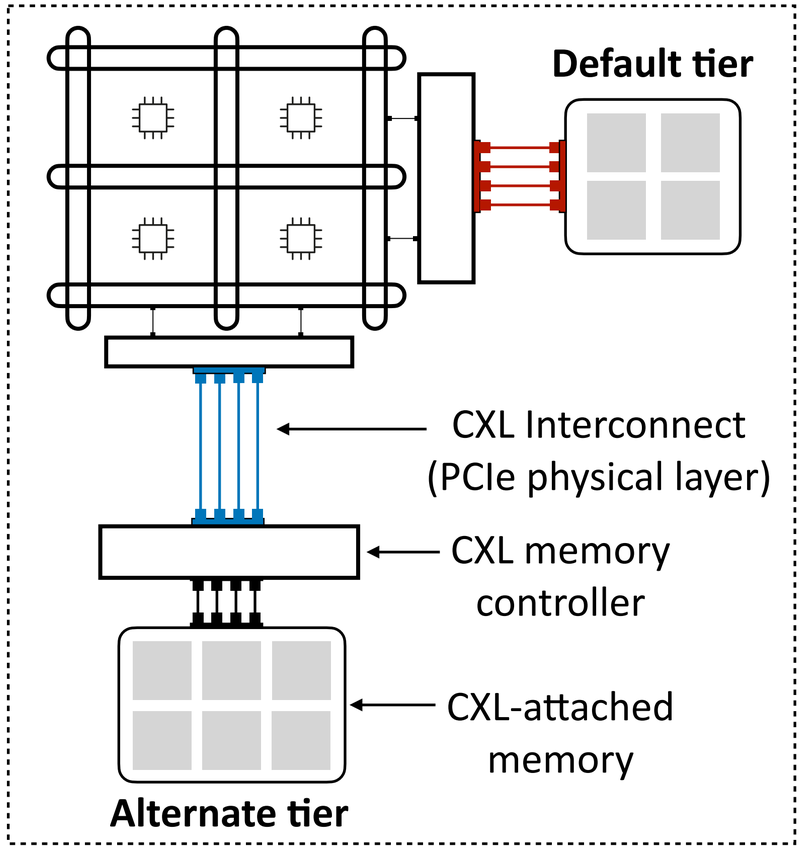
Memory tiers have different performance characteristics

Example: CXL-attached memory (compared to DDR-attached memory)

- Upto 1.04x additional bandwidth
- 2x higher access latency

Emergence of tiered memory architectures

New memory tiers via alternate interconnects



Example alternate interconnect: Compute Express Link (CXL)

Transparent, cache-coherent access to memory (via standard load/store)

Memory tiers have different performance characteristics

Example: CXL-attached memory (compared to DDR-attached memory)

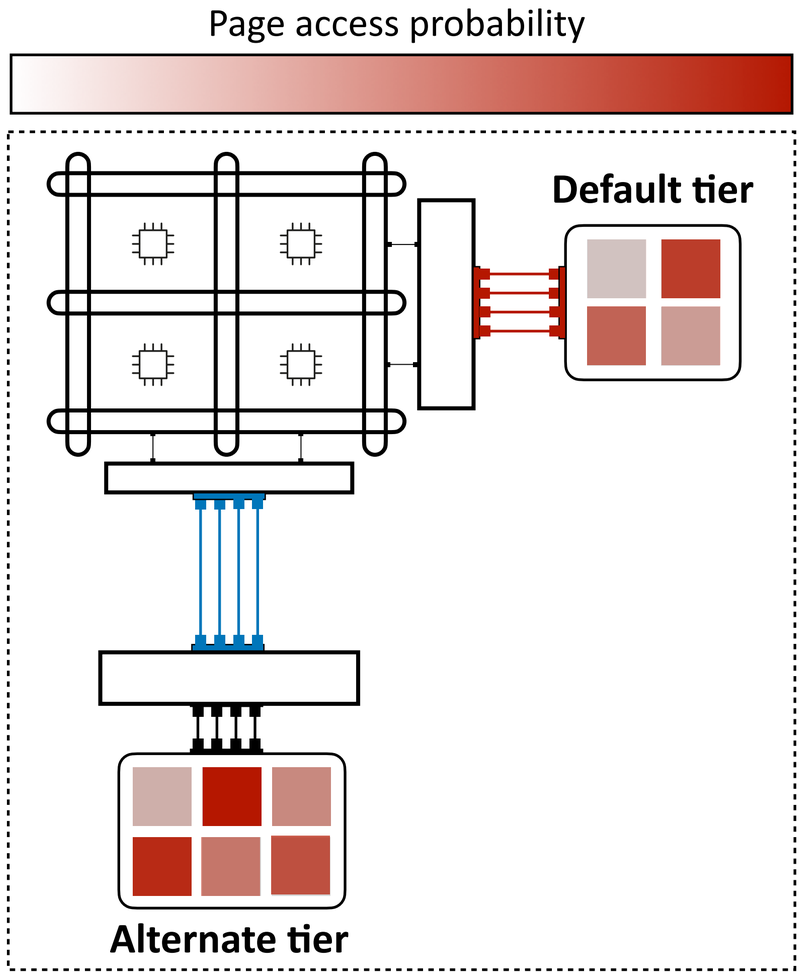
- Upto 1.04x additional bandwidth
- 2x higher access latency

Data placement across tiers critically impacts applications performance

[MemStrata, OSDI'24]	[Demystifying CXL, MICRO'23]	[Pond, ASPLOS'23]
Upto 1.61x	Upto 1.85x	Upto 2x

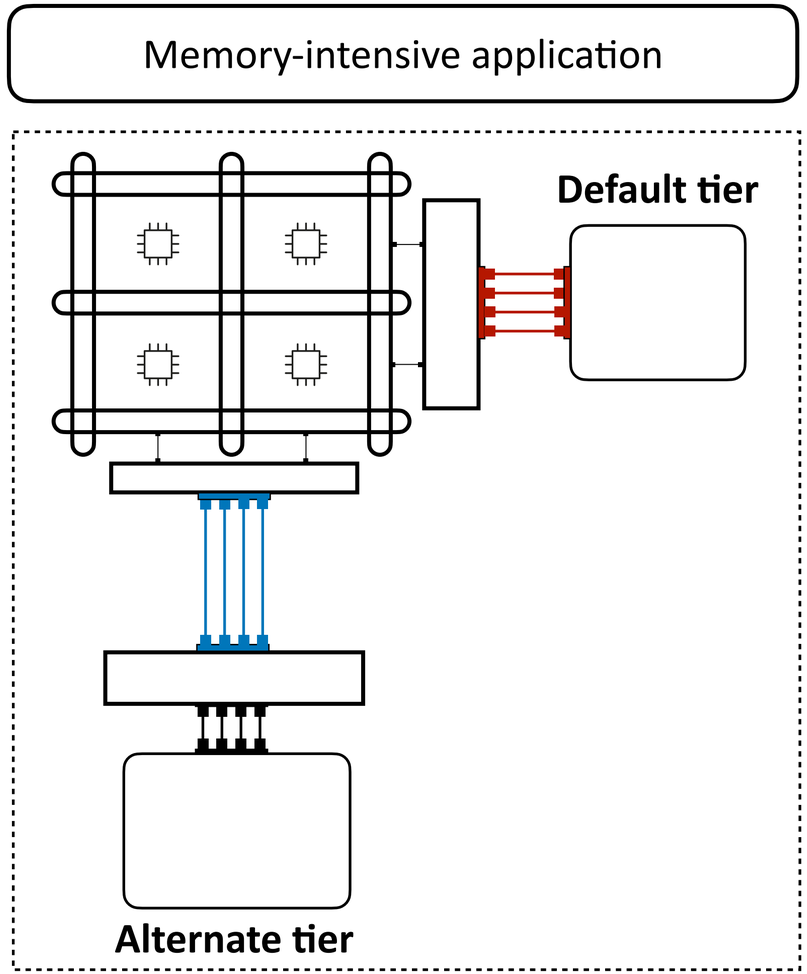
Software-based tiered memory management

Goal: Transparently adapt page placement across tiers to maximize application performance



Design dimensions	Existing systems		
	HeMem [SOSP'21]	TPP [ASPLOS'23]	MEMTIS [SOSP'23]
Access tracking Identify hot/cold pages			
Page migration Relocate pages between tiers			
Page size determination Pages vs hugepages			
Page placement Which set of pages to place in each tier	**Pack hottest pages in the default tier** Place remaining pages in the alternate tier		

Do existing systems perform optimal page placement?



Randomly access objects (1KB size)

90% 10%

Hot set Cold set

Fits in default tier

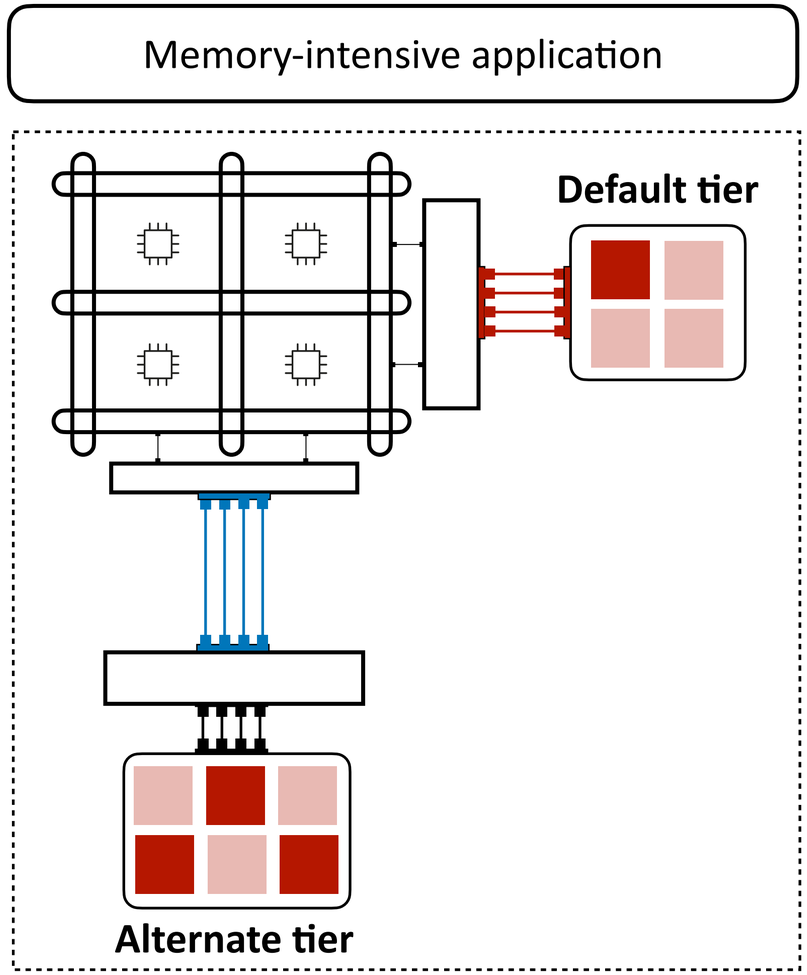
Does not fit in default tier

Run on 1/2 the cores of server

Best-case placement

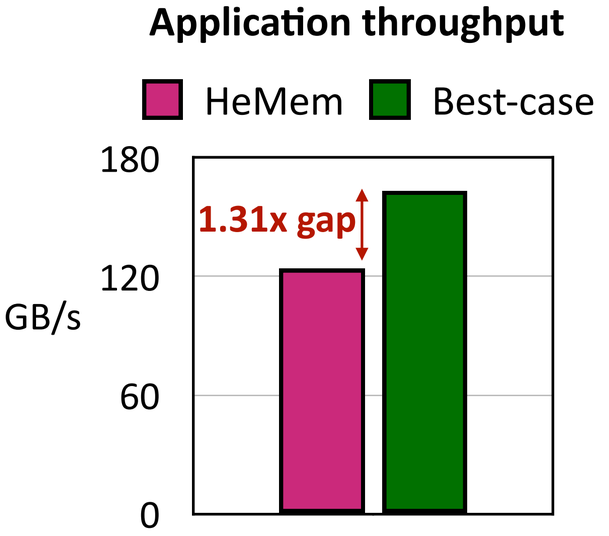
Manual sweep of different possible page placements

Do existing systems perform optimal page placement?

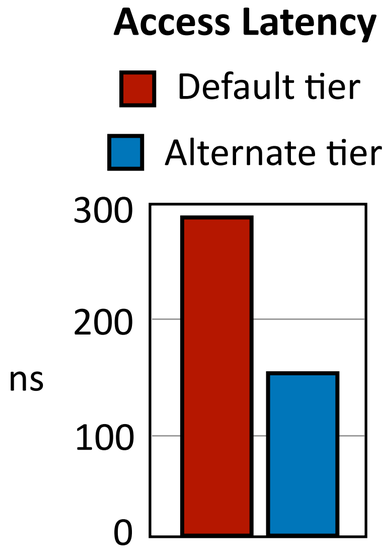


Best-case placement
Manual sweep of different possible page placements

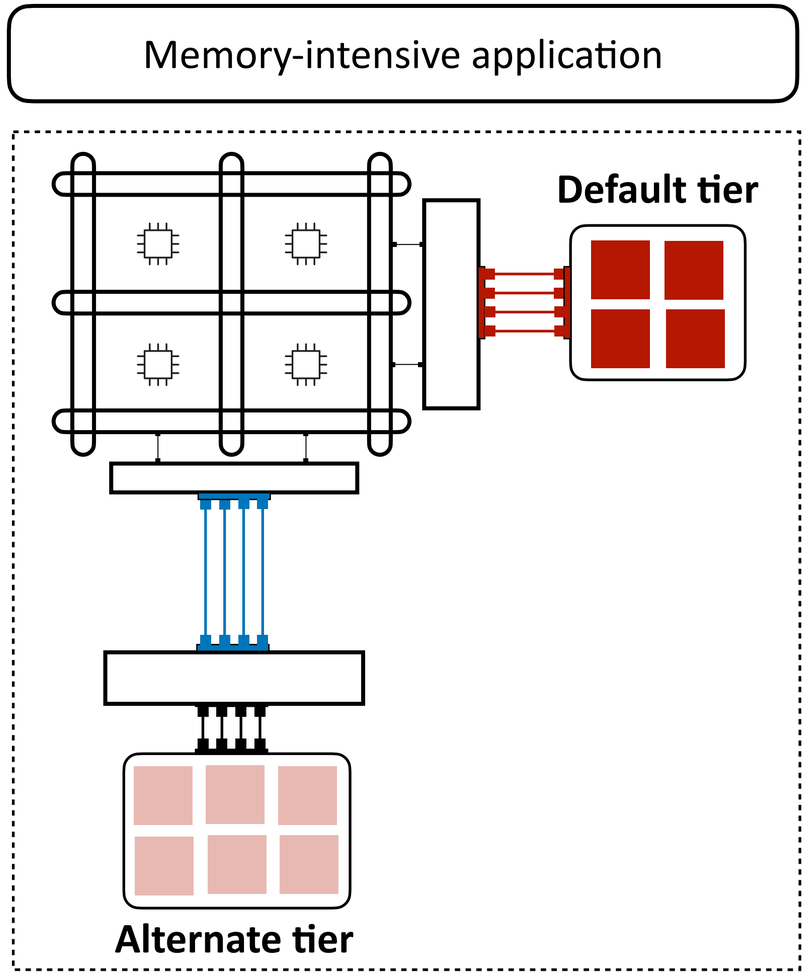
HeMem [SOSP'21]
Identifies and places all hot pages in default tier



Implicit assumption: default tier access latency < alternate tier access latency
Despite default tier serving the hottest pages



Do existing systems perform optimal page placement?



Best-case placement

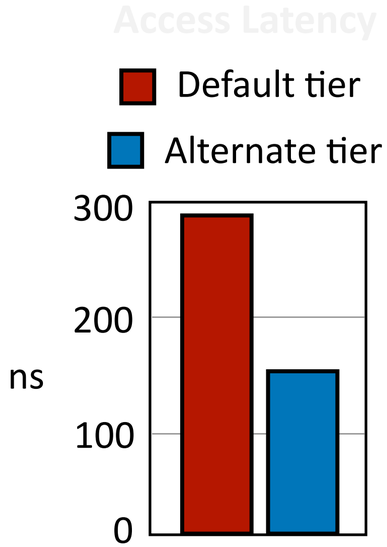
Manual sweep of different possible page placements

HeMem [SOSP'21]

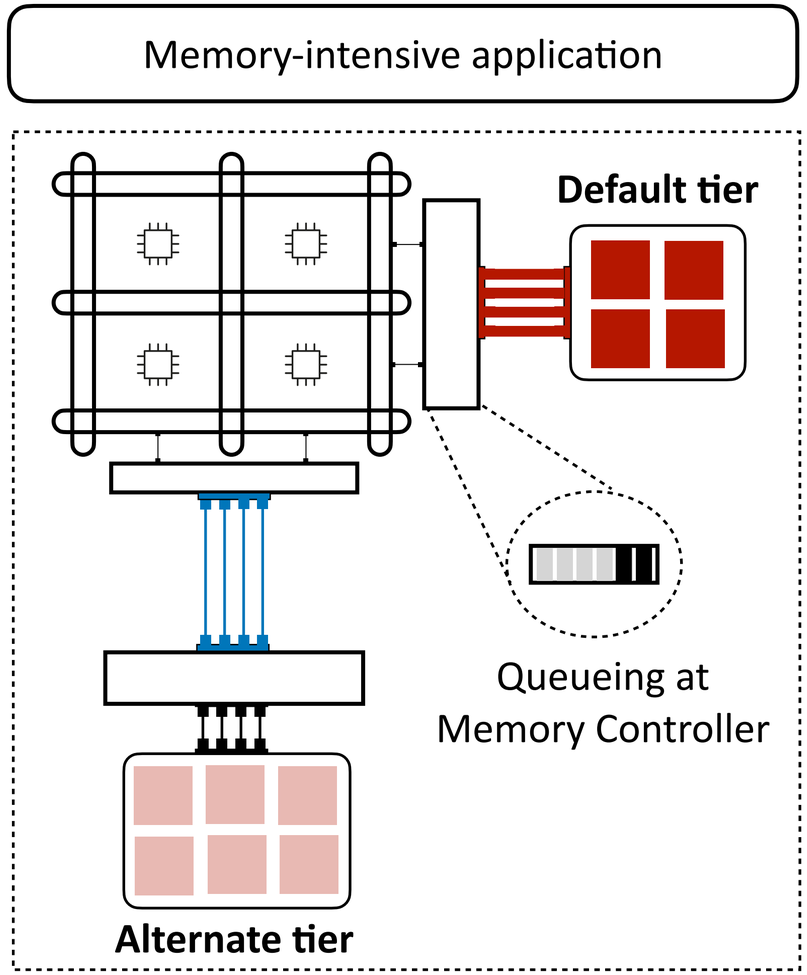
Identifies and places all hot pages in default tier

Implicit assumption: default tier access latency < alternate tier access latency

Despite default tier serving the hottest pages



Do existing systems perform optimal page placement?



Latency under **multiple** in-flight requests

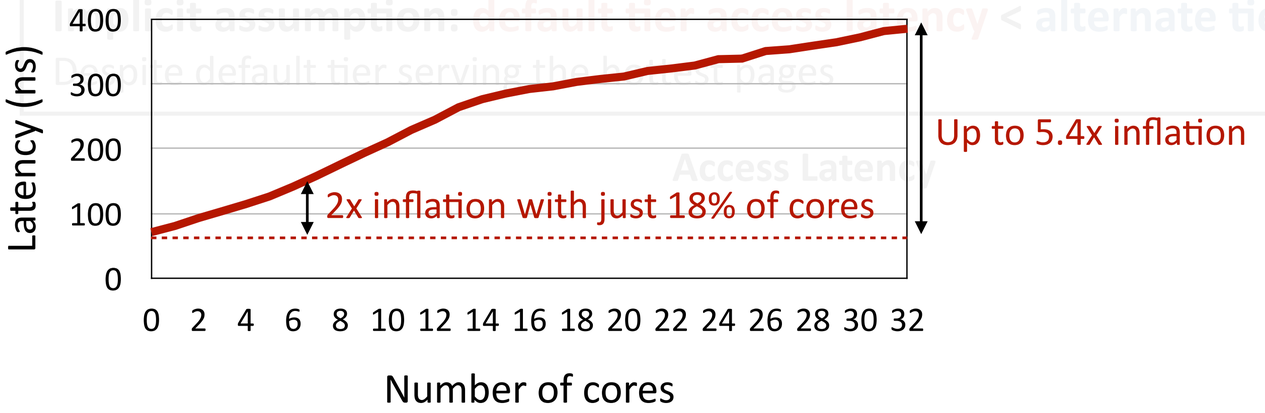
Latency under **one** in-flight request

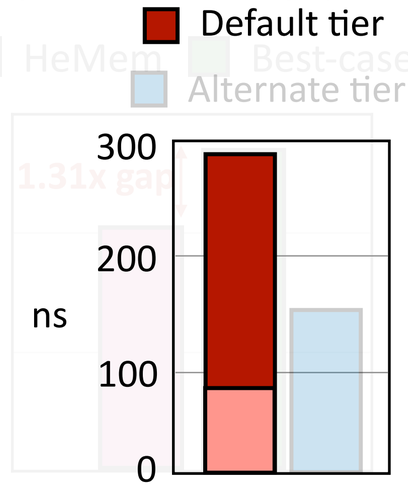
Reasons for access latency > unloaded latency

Queueing of requests in the CPU-to-memory datapath

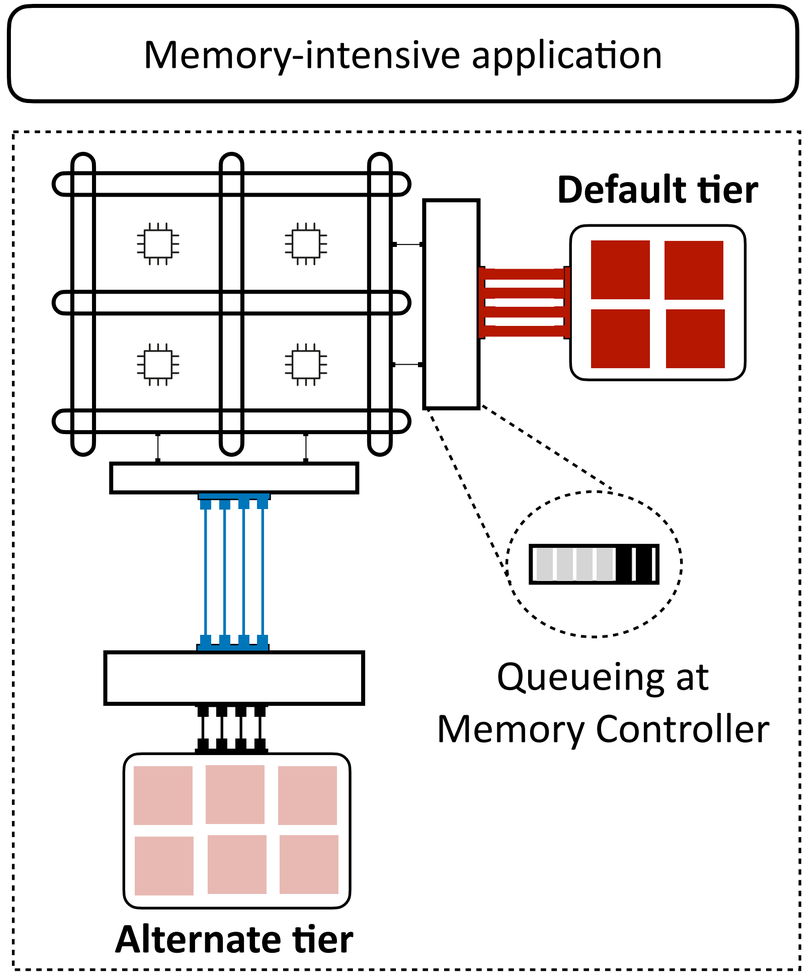
Interconnect bandwidth saturation

Contention within internal hierarchy of memory modules





Do existing systems perform optimal page placement?

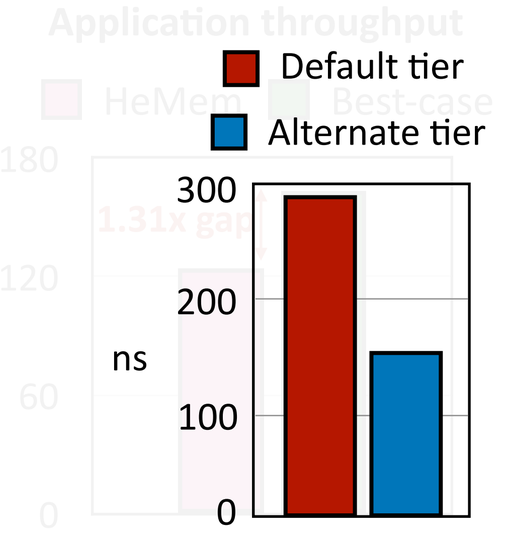


Best-case placement

Manual sweep of different possible page placements

HeMem [SOSP'21]

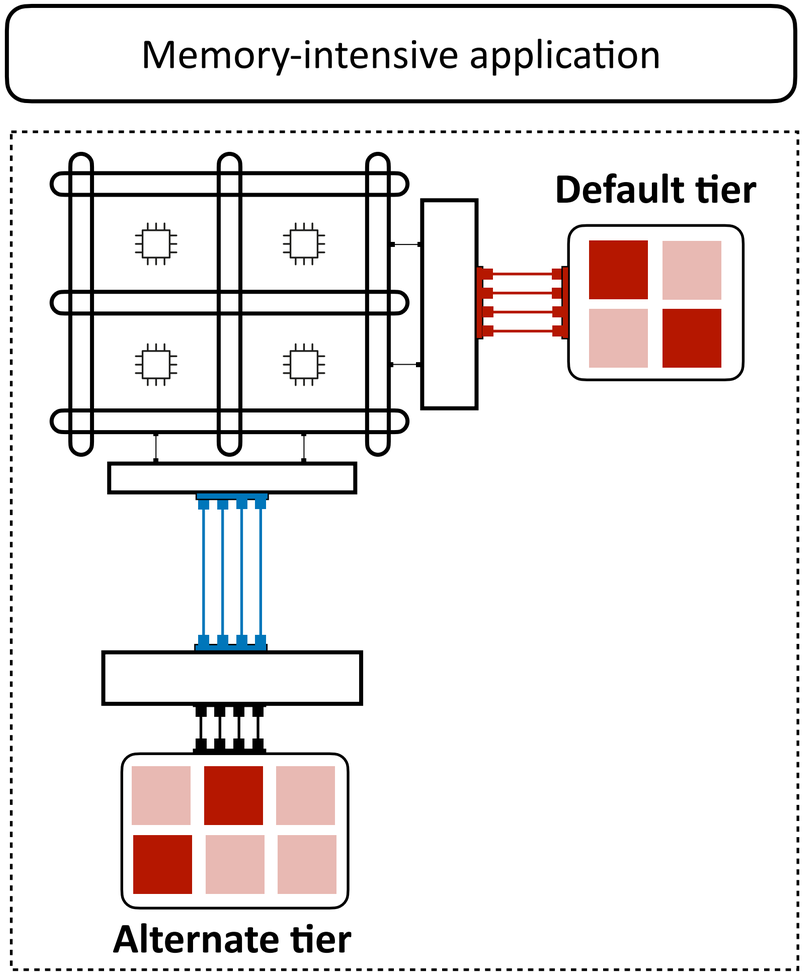
Identifies and places all hot pages in default tier



Implicit assumption: default tier access latency < alternate tier access latency

Despite default tier serving the hottest pages

Access Latency is the key!



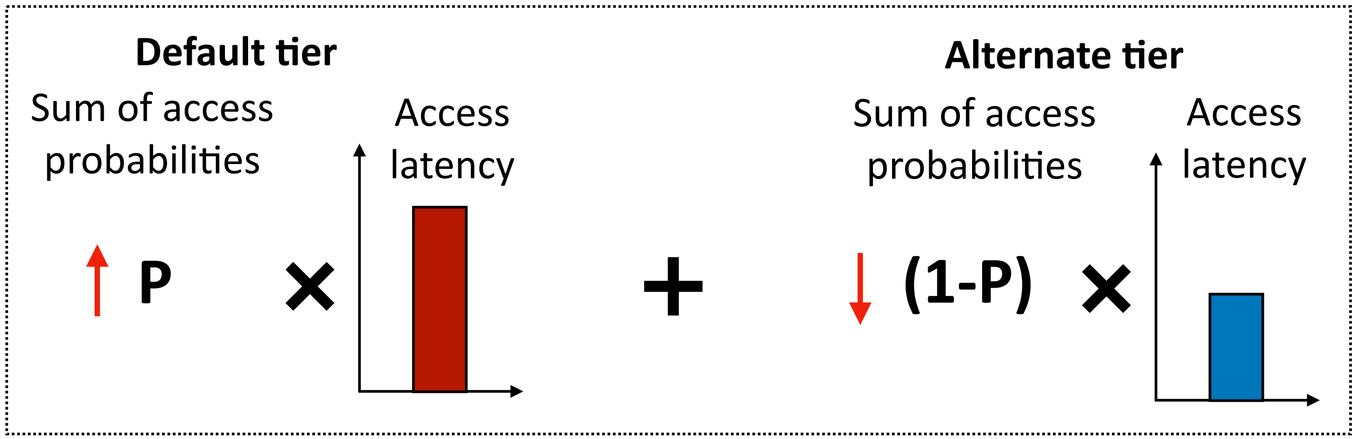
Application performance: function of memory access throughput

Minimize average access latency => maximize access throughput

- Throughput directly related to latency

$$\text{Per-core Throughput} = \frac{\text{\# in-flight requests}}{\text{Average access latency}}$$

← Limited by hardware buffer sizes (e.g Line Fill Buffer)



Packing hottest pages in default tier is no longer optimal

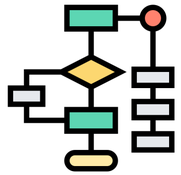
Colloid overview

Key principle: Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers

Access latency with nanosecond precision

Low-overhead mechanism to measure per-tier access latency

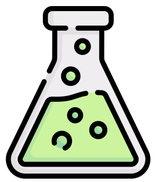


Page placement algorithm

Decide which set of pages to place in each tier

Integration with existing systems

Leverages existing memory management innovations



Evaluation

Understand effectiveness over wide range of workloads

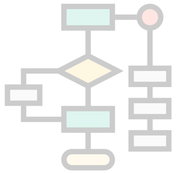
Colloid overview

Key principle: Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers

Access latency with nanosecond precision

Low-overhead mechanism to measure per-tier access latency



Page placement algorithm

Decide which set of pages to place in each tier

Integration with existing systems

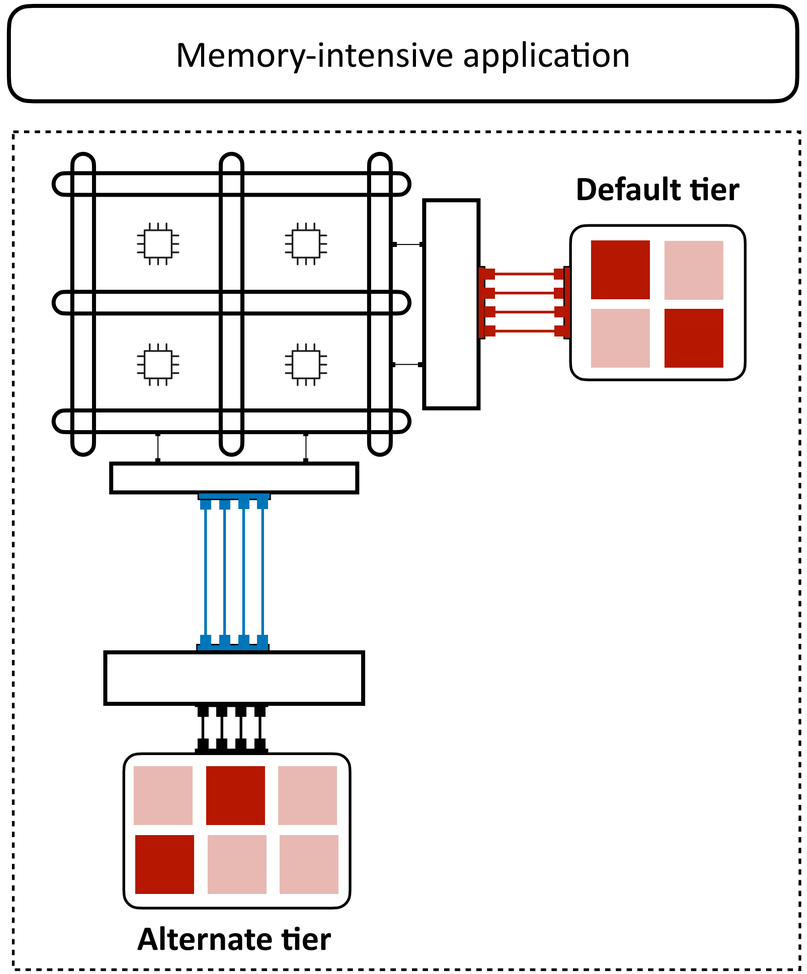
Leverages existing memory management innovations

Evaluation

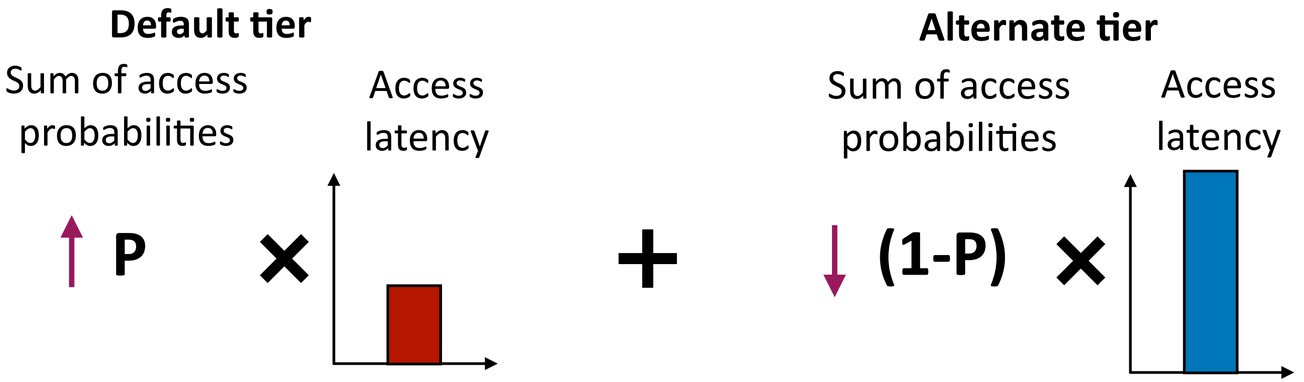
Understand effectiveness over wide range of workloads

Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers



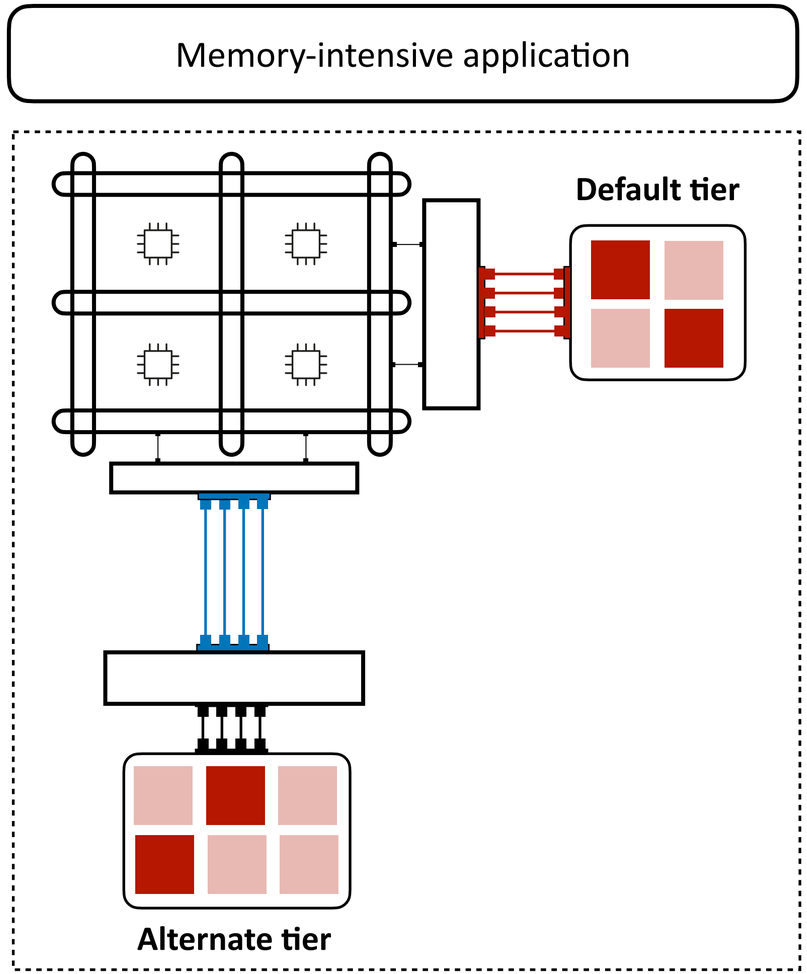
Goal: Minimize average access latency



Place more hot pages in the default tier

Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers



Goal: Minimize average access latency

Default tier

Sum of access probabilities

↓ **P** × Access latency

+

Alternate tier

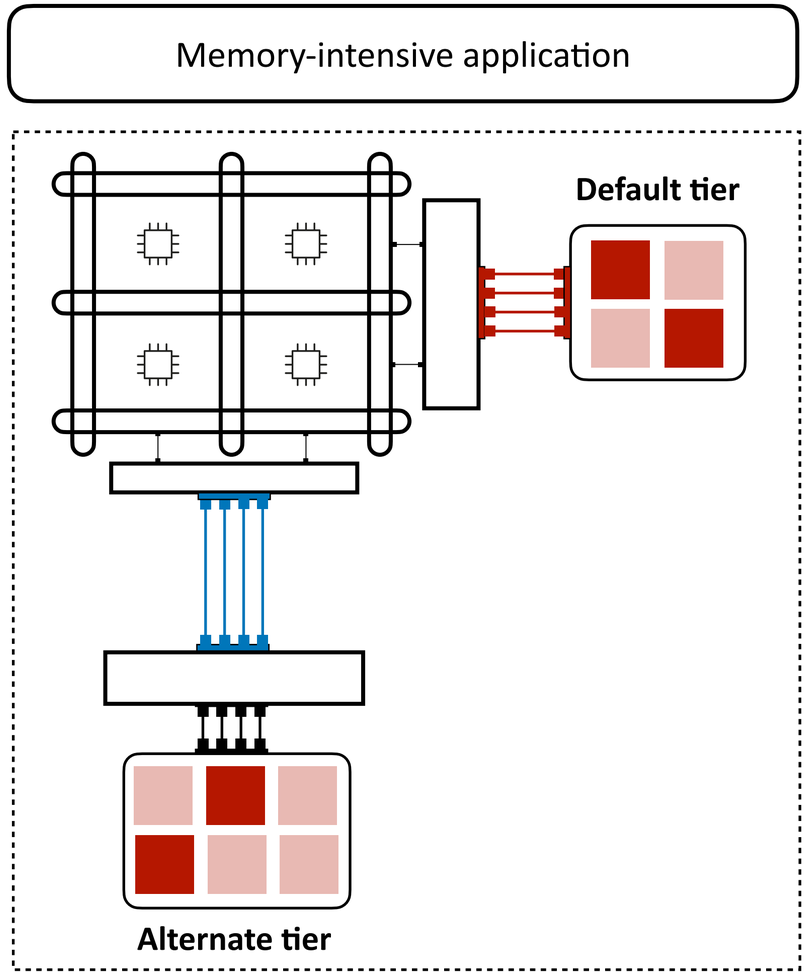
Sum of access probabilities

↑ **(1-P)** × Access latency

Place more hot pages in the alternate tier

Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers



Goal: Minimize average access latency

Default tier

Sum of access probabilities: **P**

Access latency

×

+

Alternate tier

Sum of access probabilities: **(1-P)**

Access latency

×

Do not adapt page placement

State of balanced access latencies: ideal equilibrium point

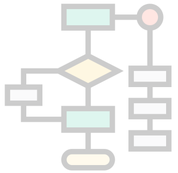
Colloid overview

Key principle: Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers

Access latency with nanosecond-scale precision

Low-overhead mechanism to measure per-tier access latency



Page placement algorithm

Decide which set of pages to place in each tier

Integration with existing systems

Leverages existing memory management innovations

Evaluation

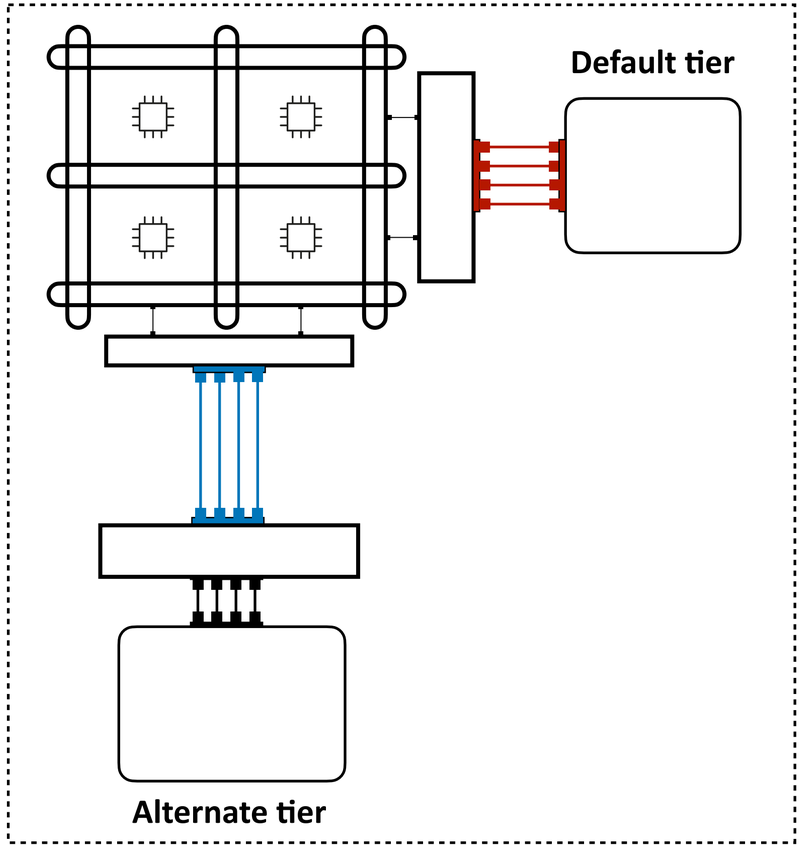
Understand effectiveness over wide range of workloads

Access latency with nanosecond-scale precision

Fundamental design aspects of CPU-to-memory datapath enable fine-grained visibility into per-tier access latency

Understanding the host network

SIGCOMM’24

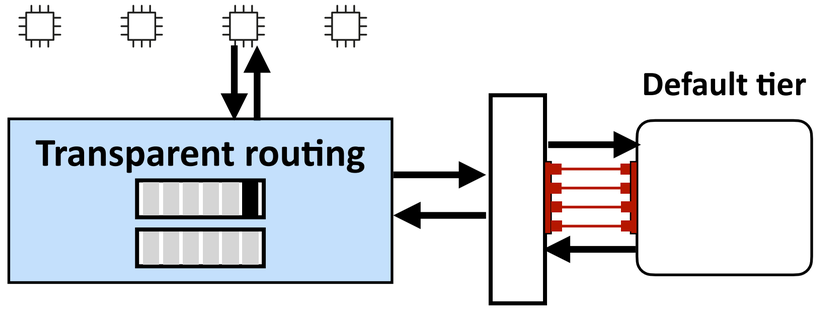


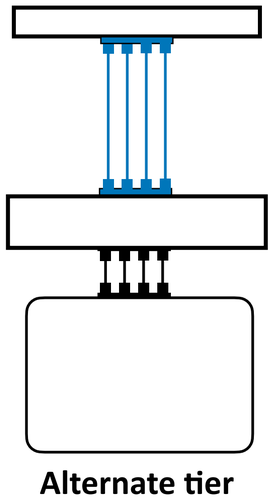
Access latency with nanosecond-scale precision

Fundamental design aspects of CPU-to-memory datapath enable fine-grained visibility into per-tier access latency

Understanding the host network

SIGCOMM'24





Insight #1: Transparent routing of requests to tiers provides vantage point

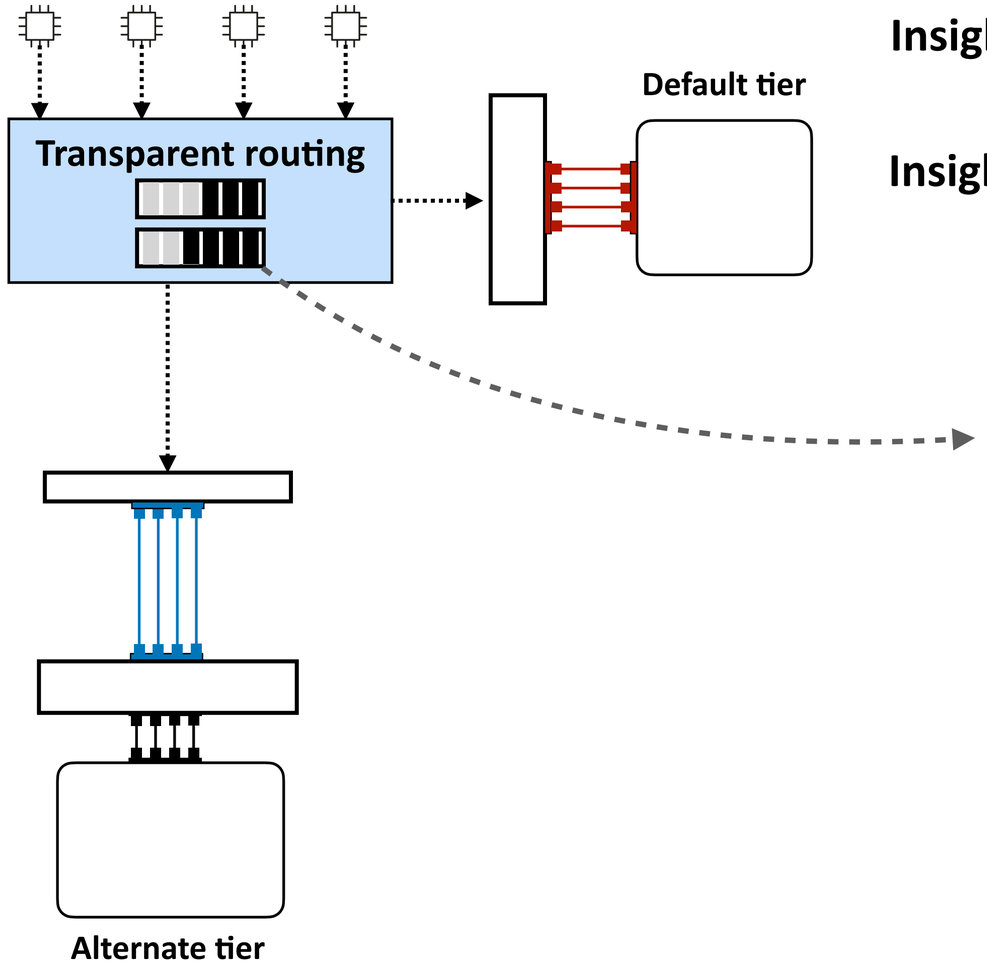
Insight #2: Requests remain queued until they are serviced from tier

Access latency with nanosecond-scale precision

Fundamental design aspects of CPU-to-memory datapath enable fine-grained visibility into per-tier access latency

Understanding the host network

SIGCOMM'24

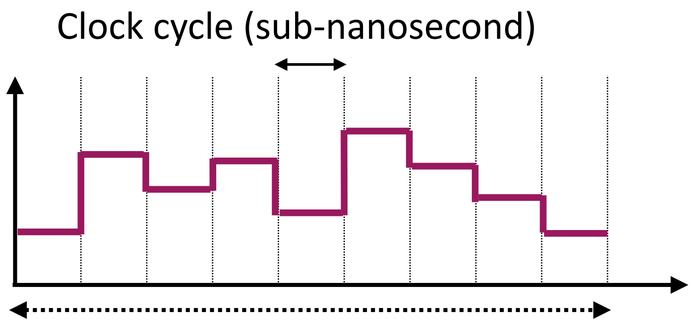


Insight #1: Transparent routing of requests to tiers provides vantage point

Insight #2: Requests remain queued until they are serviced from tier

Per-tier queue occupancy and request arrival rates

Measurement enabled by hardware counters



Apply Little's Law to compute access latency

$$\text{Average access latency} = \frac{\text{Average queue occupancy}}{\text{Average request arrival rate}}$$

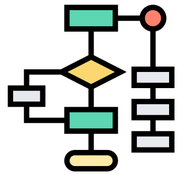
Colloid overview

Key principle: Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers

Access latency with nanosecond precision

Low-overhead mechanism to measure per-tier access latency



Page placement algorithm

Decide which set of pages to place in each tier

Integration with existing systems

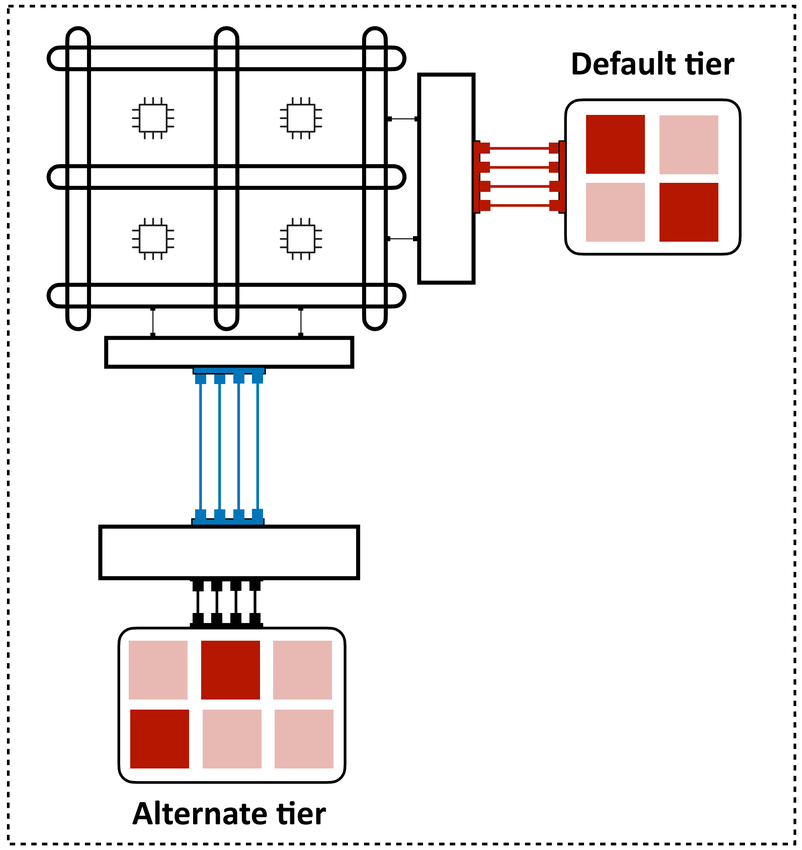
Leverages existing memory management innovations

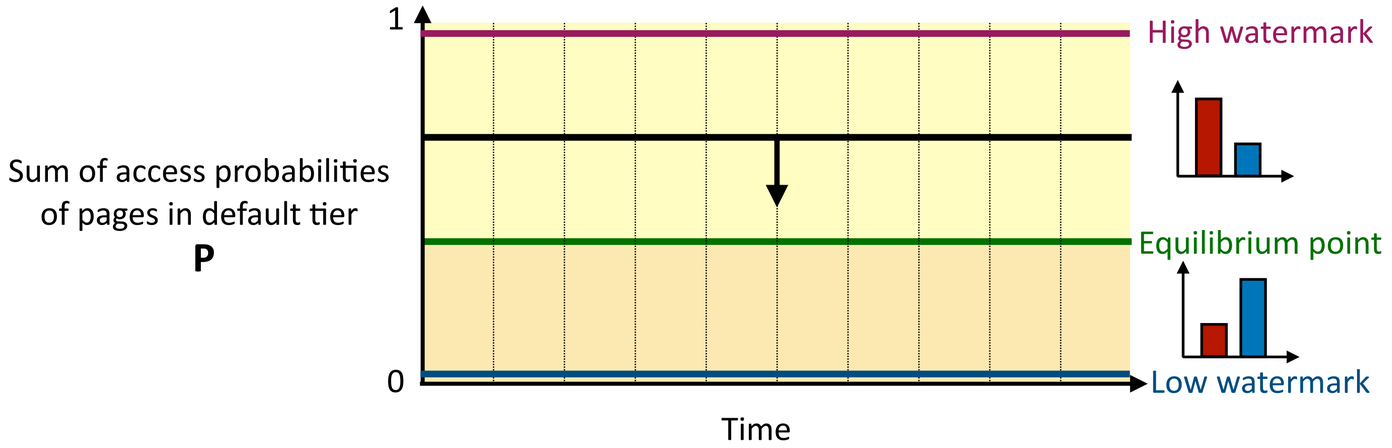
Evaluation

Understand effectiveness over wide range of workloads

Colloid page placement algorithm

Executes periodically at fixed time intervals (quanta) and adapts page placement based on access latencies





Principle of balancing access latencies

- Whether to increase or decrease P

By how much to change P (ΔP)?

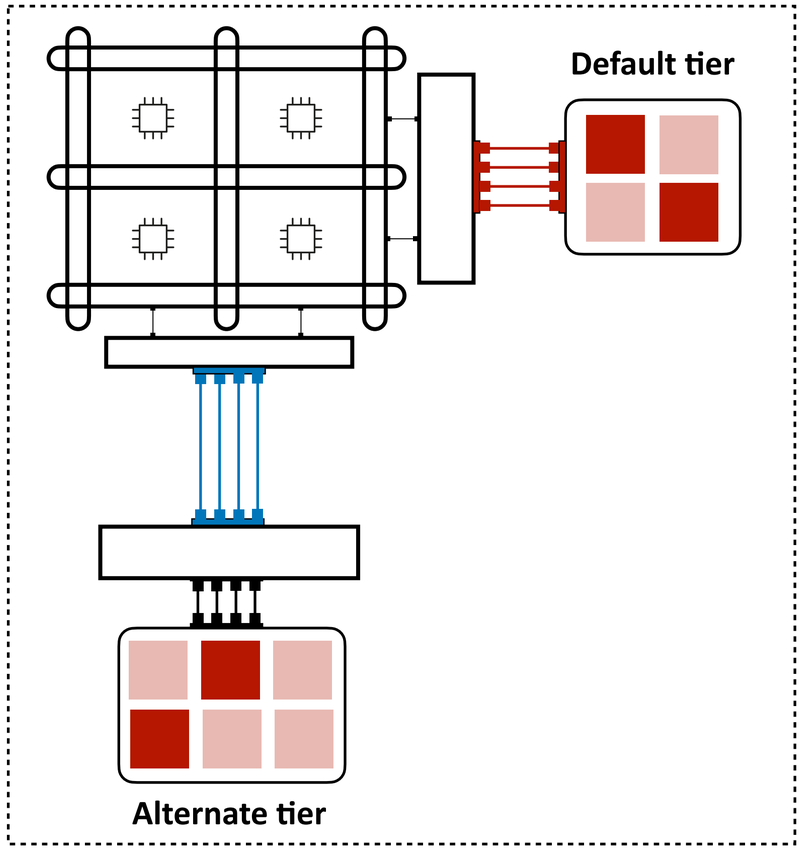
Maintain low watermark and high watermark

Invariant #1: P and equilibrium point between low and high watermark

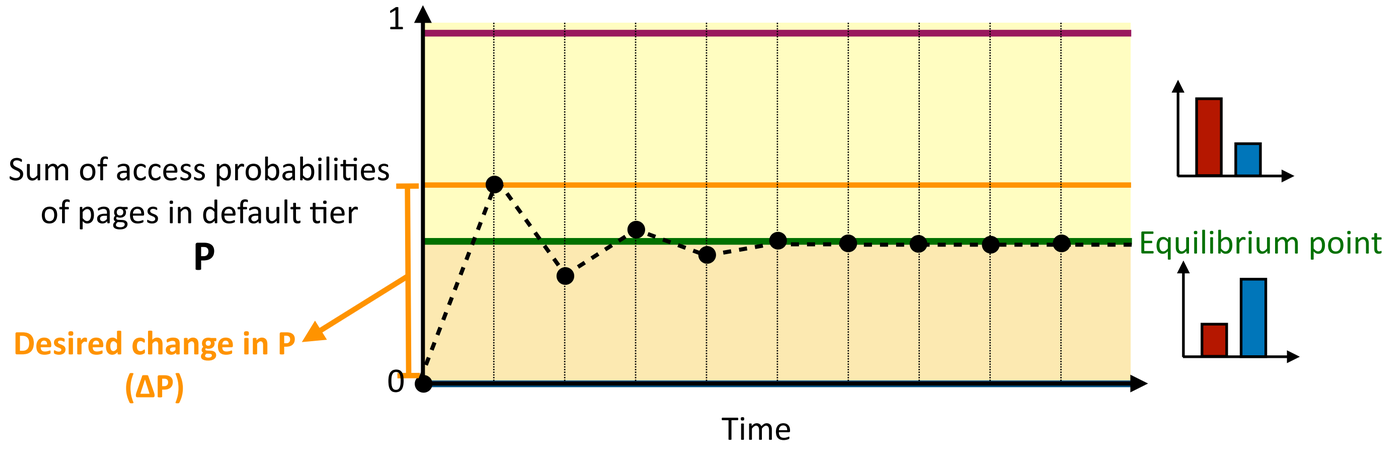
Invariant #2: Gap between watermarks reduces

Colloid page placement algorithm

Executes periodically at fixed time intervals (quanta) and adapts page placement based on access latencies



Illustrative example (static workload)



Principle of balancing access latencies

- Whether to increase or decrease P

By how much to change P (ΔP)?

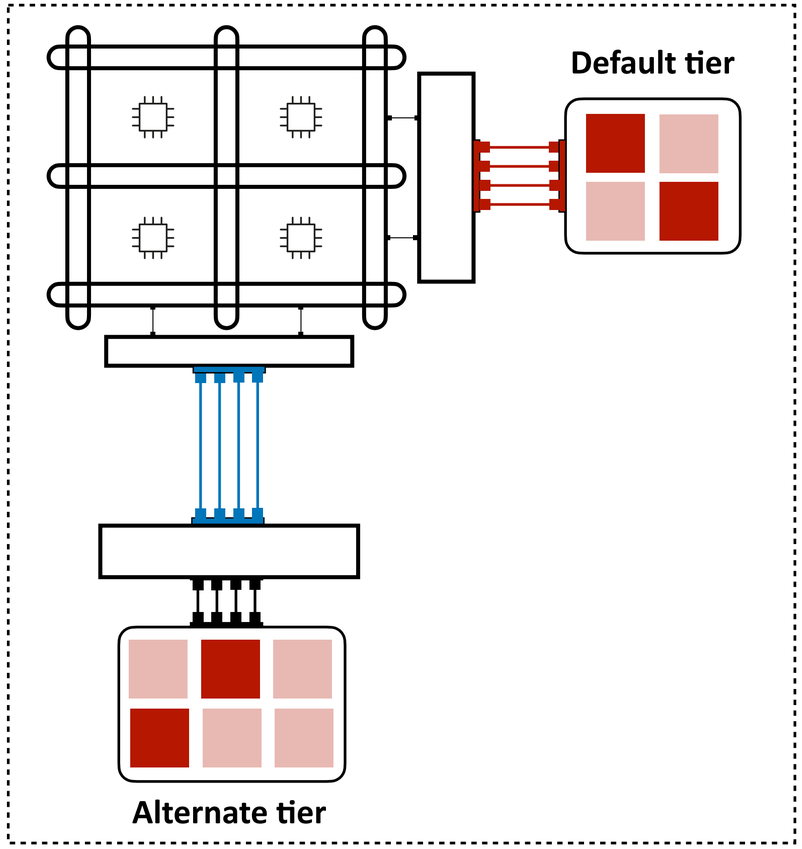
Maintain low watermark and high watermark

Invariant #1: P and equilibrium point between low and high watermark

Invariant #2: Gap between watermarks reduces

Colloid page placement algorithm

Handling dynamic changes in workload

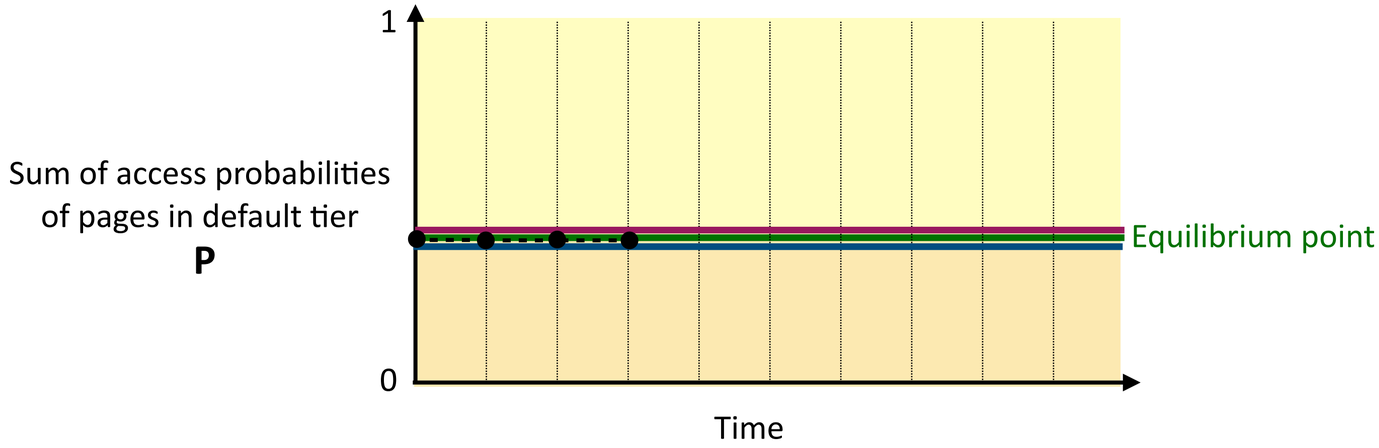


Change in memory access pattern

- Abrupt change in access probabilities of pages

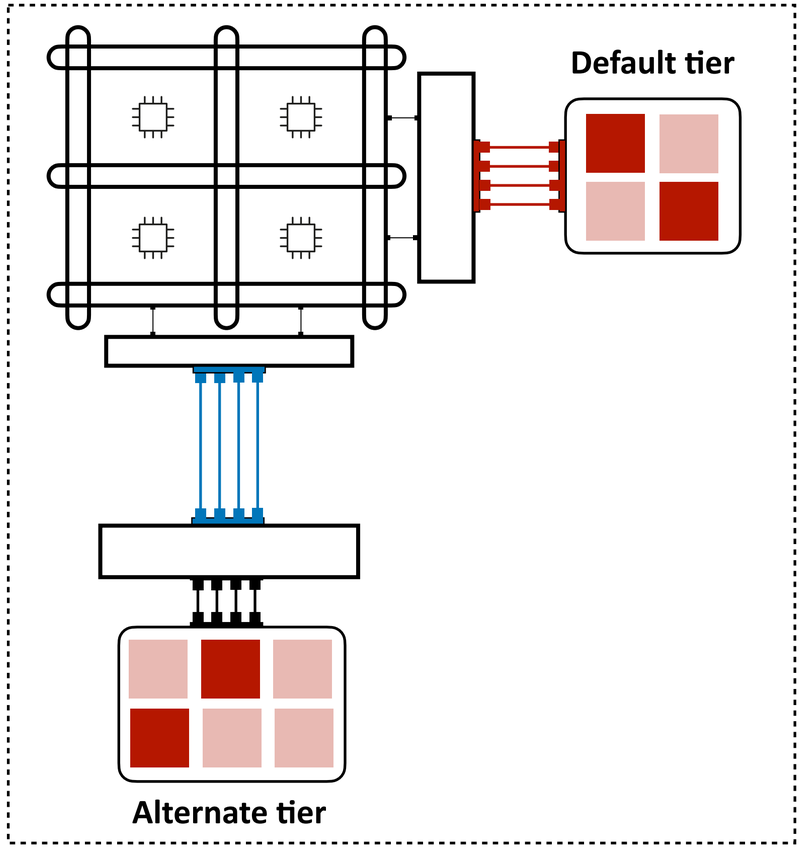
Change in memory interconnect contention

- Abrupt change in equilibrium point



Colloid page placement algorithm

Handling dynamic changes in workload

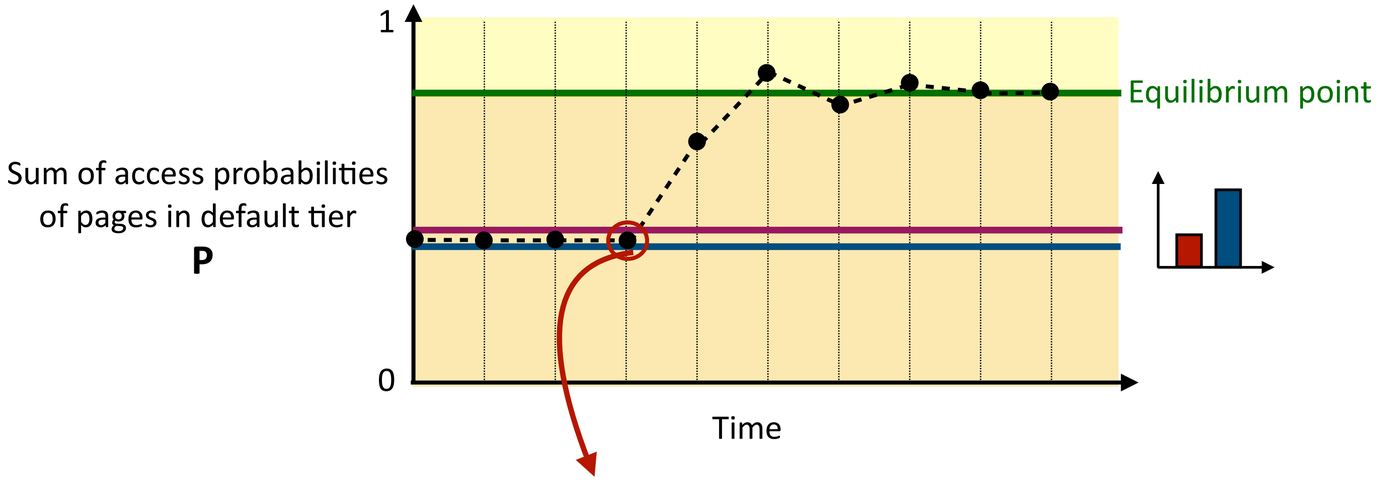


Change in memory access pattern

- Abrupt change in access probabilities of pages

Change in memory interconnect contention

- Abrupt change in equilibrium point



Watermarks close and **access latencies not close** → **Reset watermark**

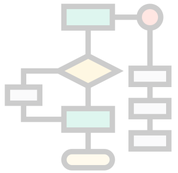
Colloid overview

Key principle: Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers

Access latency with nanosecond precision

Low-overhead mechanism to measure per-tier access latency



Page placement algorithm

Decide which set of pages to place in each tier

Integration with existing systems

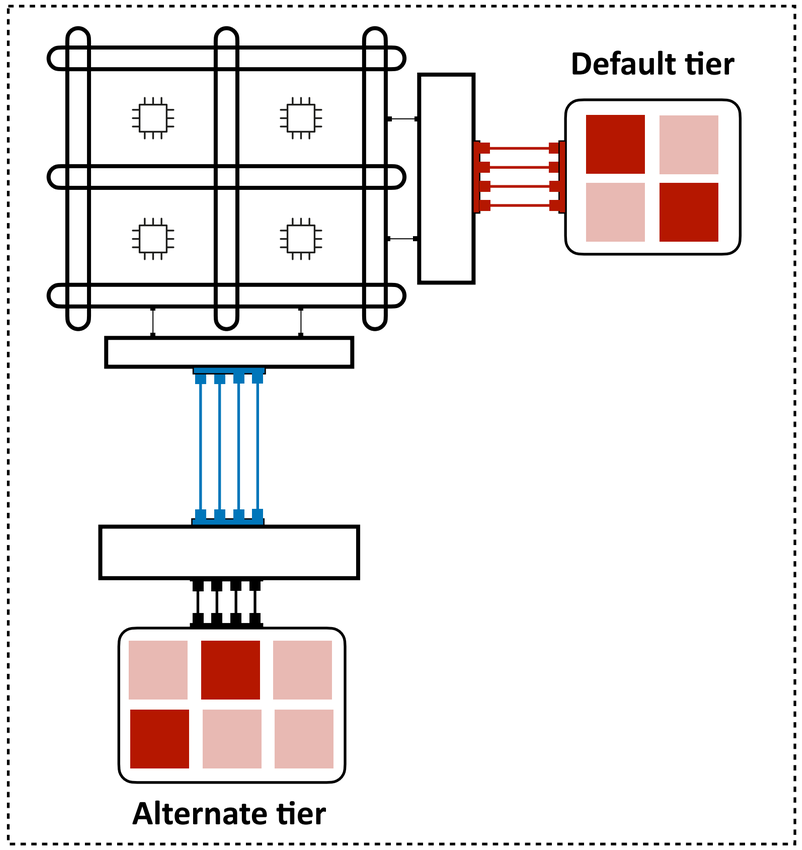
Leverages existing memory management innovations

Evaluation

Understand effectiveness over wide range of workloads

Colloid integrates with existing systems

Implemented on top of three state-of-the-art tiered memory management systems



	Existing systems		
Design dimensions	HeMem [SOSP'21]	TPP [ASPLOS'23]	MEMTIS [SOSP'23]
Access tracking	💡	💡	💡
Page migration	💡		
Page size determination			💡
Access latency measurement	**Colloid**		
Page placement	**Colloid**		
Lines of code	520	411	315

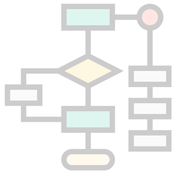
Colloid overview

Key principle: Principle of balancing access latencies

Adapt page placement to balance (loaded) access latencies of tiers

Access latency with nanosecond precision

Low-overhead mechanism to measure per-tier access latency

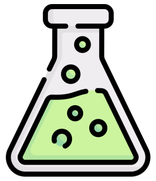


Page placement algorithm

Decide which set of pages to place in each tier

Integration with existing systems

Leverages existing memory management innovations

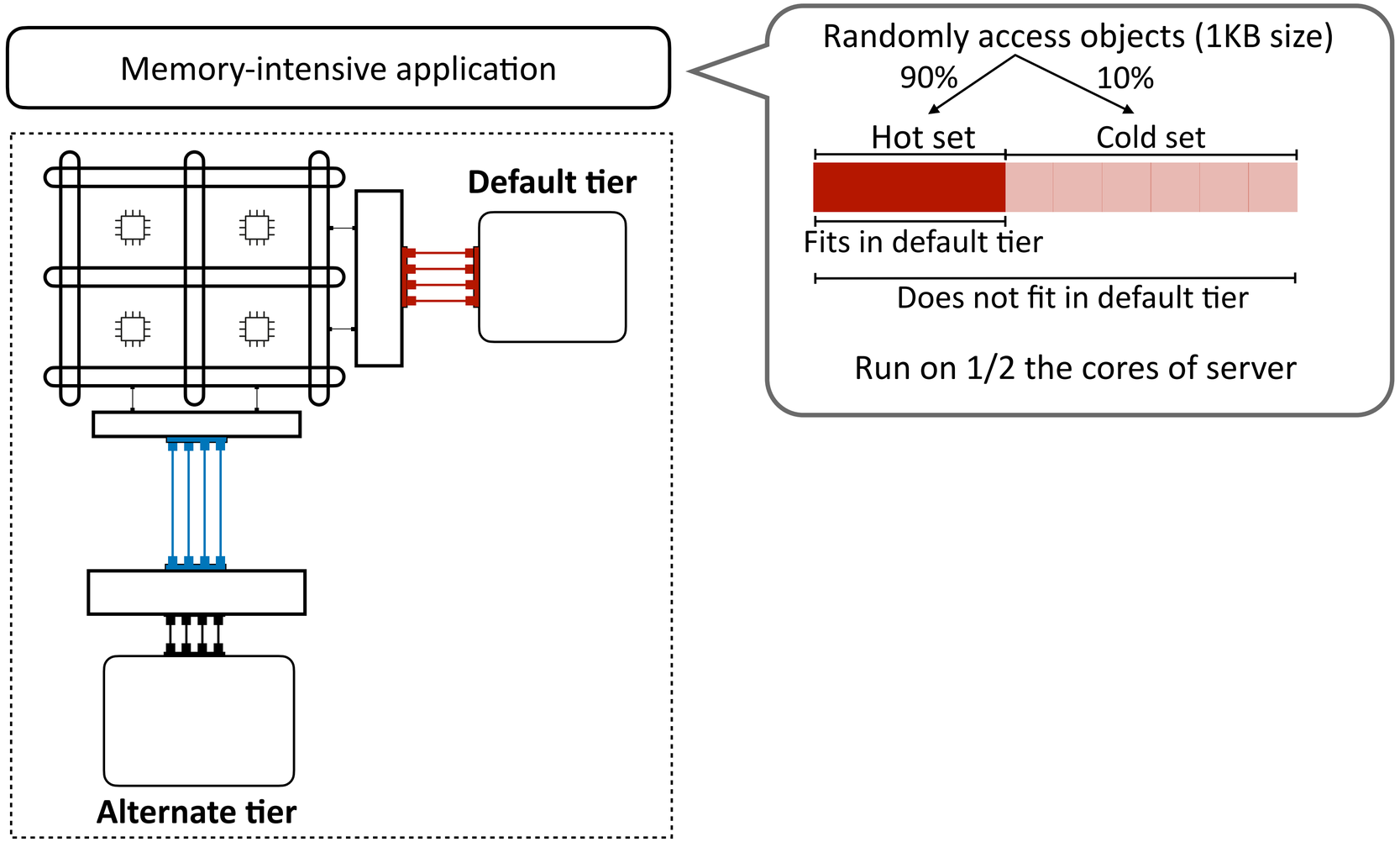


Evaluation

Understand effectiveness over wide range of workloads

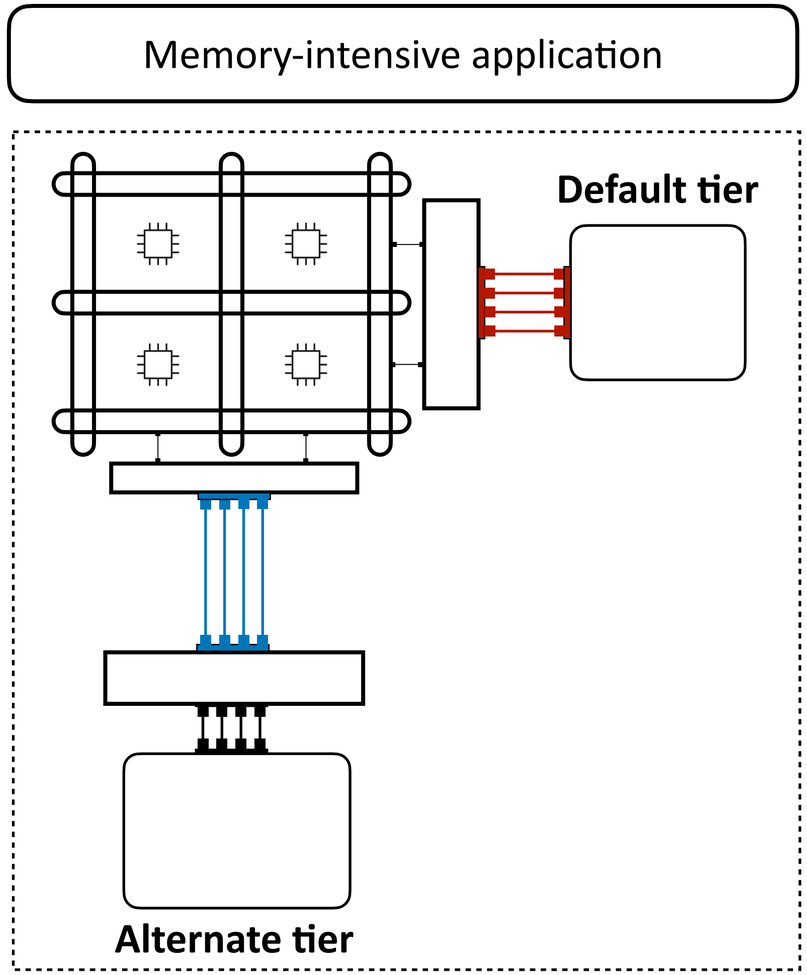
Colloid evaluation

Colloid enables existing systems to achieve near-optimal performance

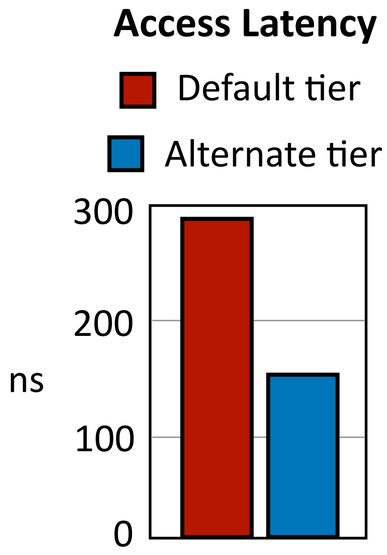


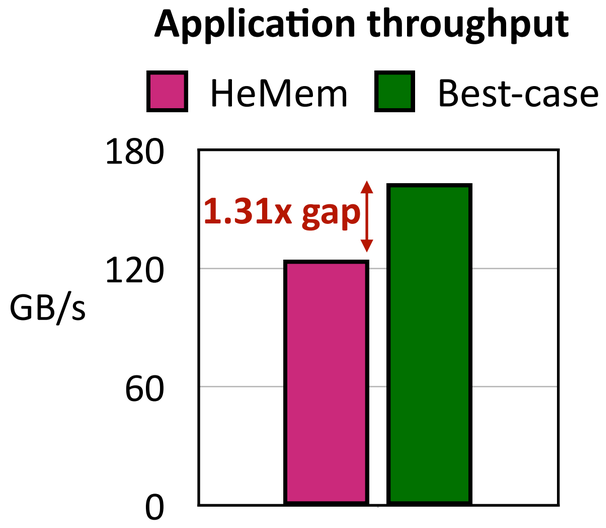
Colloid evaluation

Colloid enables existing systems to achieve near-optimal performance

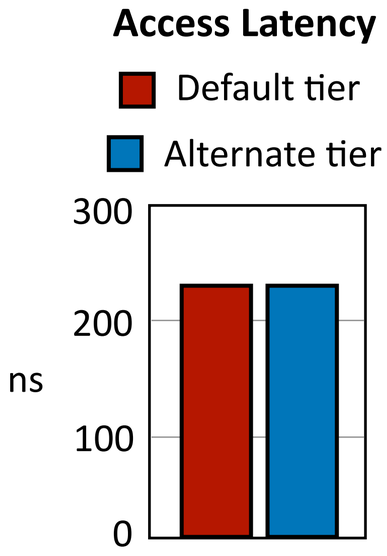


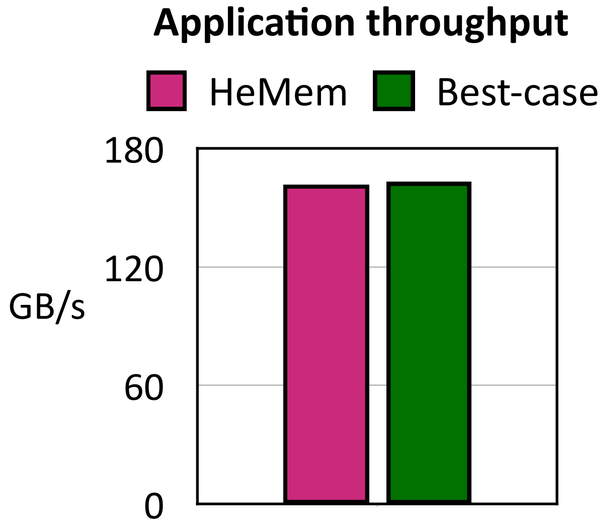
Without Colloid





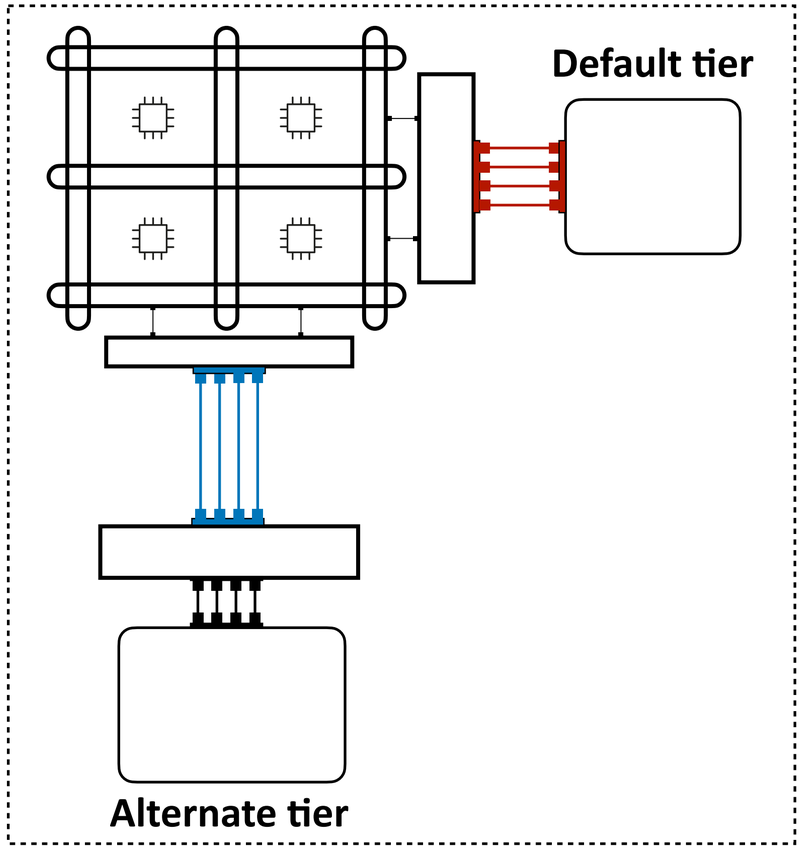
With Colloid





Colloid evaluation

Colloid benefits translate to end-to-end performance improvements for real applications



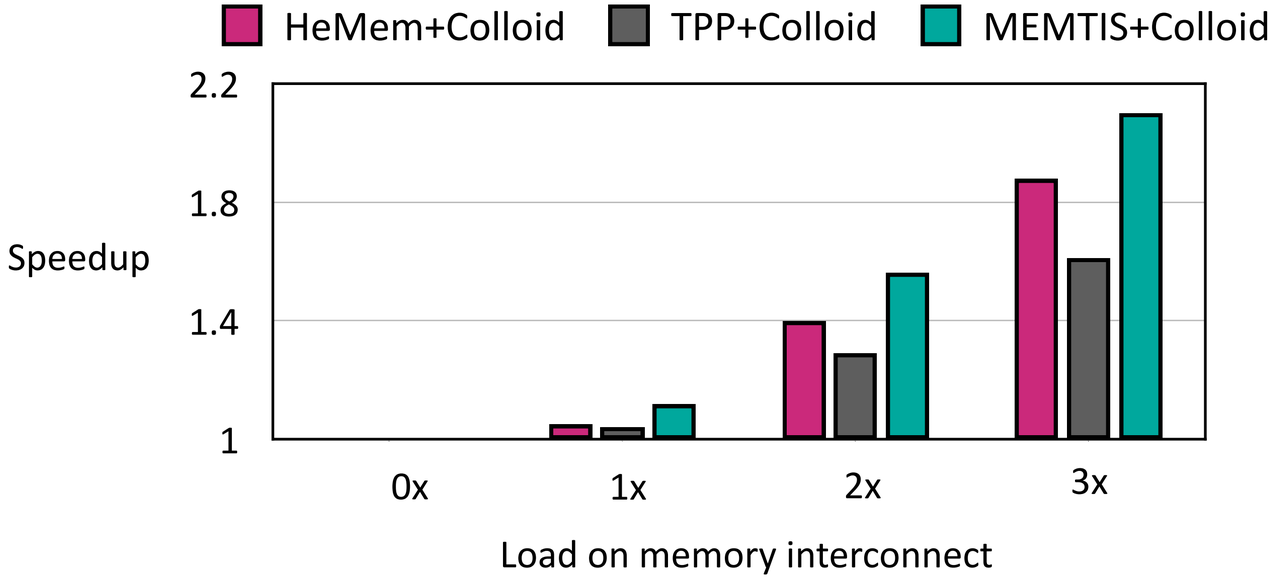
PageRank algorithm

Twitter graph

16 iterations

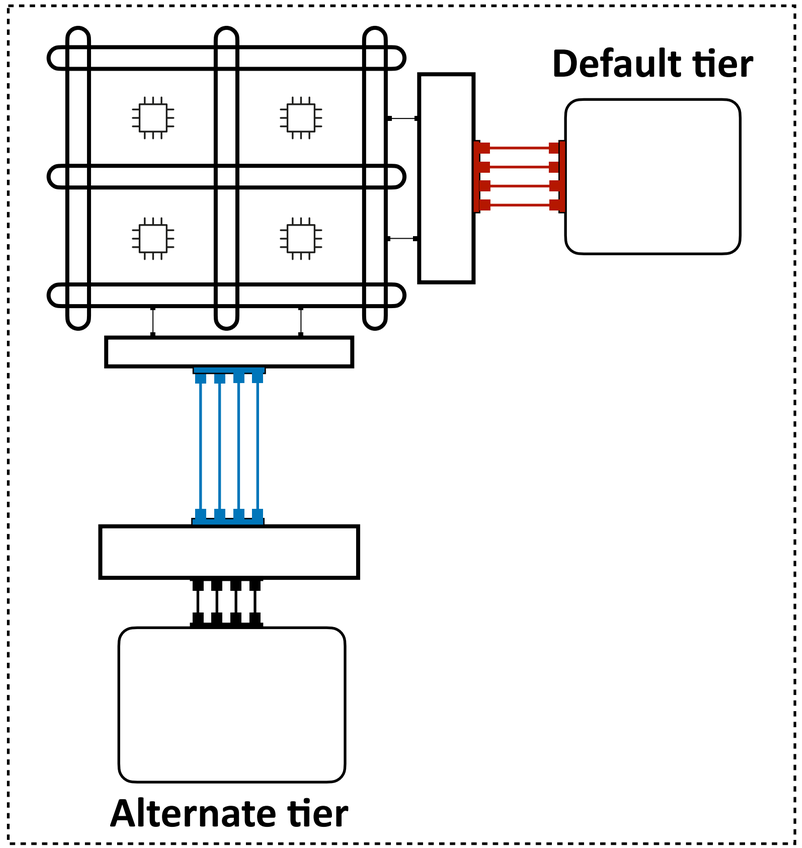
Run on 1/2 the server cores

(Controlled load on default tier memory interconnect generated via memory antagonist)



Colloid evaluation

Colloid consistently enables benefits for a wide variety of workloads and hardware parameters



Varying alternate tier unloaded latency

Colloid provides benefits even with larger alternate tier unloaded latencies

Varying object size, core count, read/write ratio

Colloid continues to achieve near-optimal performance

Dynamic workloads

Colloid does not impact timescale for reaction to change in access patterns

Enables reacting to change in memory interconnect contention as similar timescale

Real applications

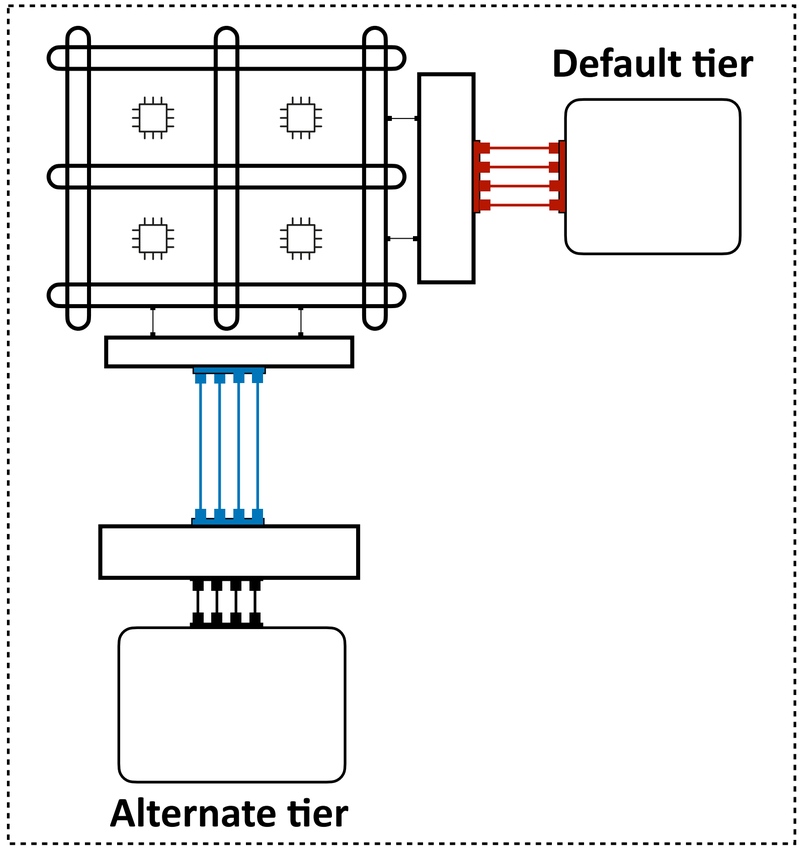
Colloid achieves up to 2.1x improvement in end-to-end performance

GAP — Graph processing engine

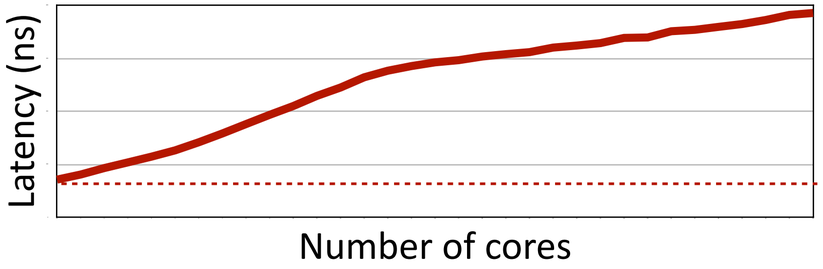
Silo — In-memory transactional database

In-memory key-value cache

Access latency is the key!

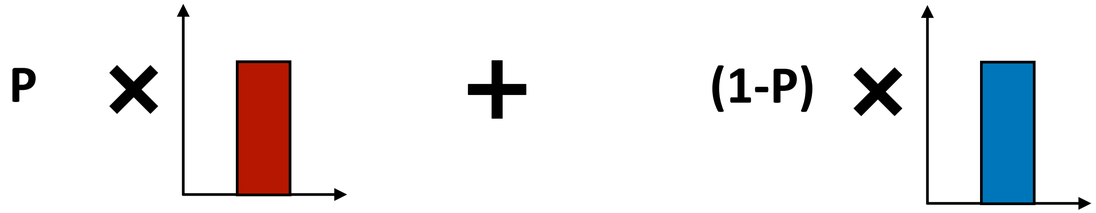


Access latency != unloaded latency



Principle of balancing access latencies

Minimize average access latency



Colloid: Tiered memory management mechanism

Enables existing systems to realize the principle of balancing access latencies

https://github.com/host-architecture/colloid