

SOSP 2026

Tierce: Observability-Driven Tiered Memory Management for Colocated Workloads

Hanchen Xu*, Berkay Inceisci*, Hao Li, Zhenyu Zhang, Huaicheng Li



* Equal contribution

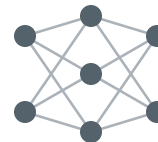
Choosing Pages for Fast Memory



In-memory caching



Graph analytics



Machine learning

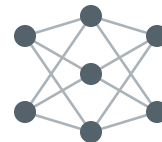
Choosing Pages for Fast Memory



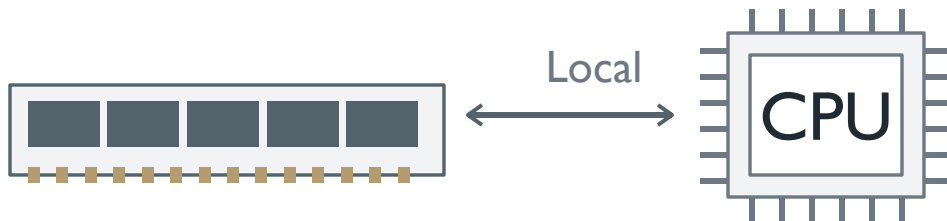
In-memory caching



Graph analytics



Machine learning



Fast: local DRAM

- Limited capacity
- + Low latency

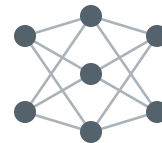
Choosing Pages for Fast Memory



In-memory caching



Graph analytics

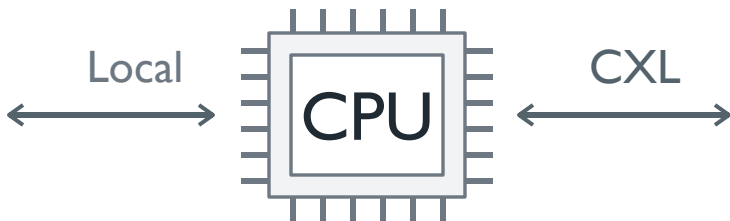
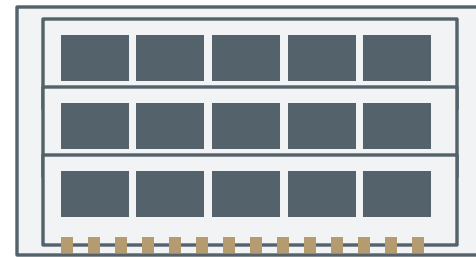


Machine learning



Fast: local DRAM

- Limited capacity
- + Low latency



Slow: memory expansion

- + More capacity
- Higher latency

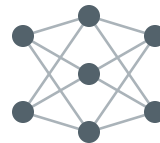
Choosing Pages for Fast Memory



In-memory caching



Graph analytics

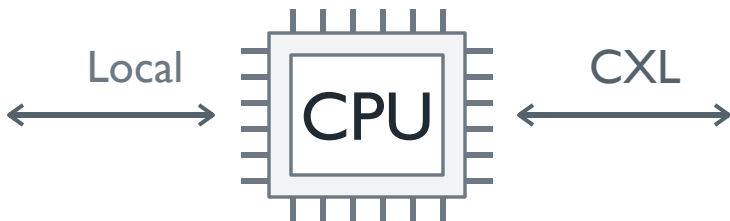
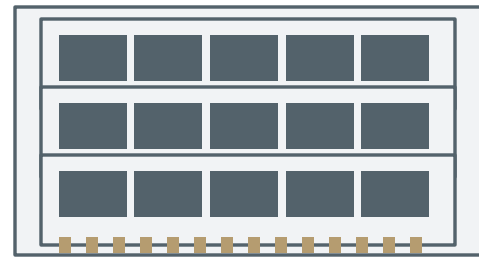


Machine learning



Fast: local DRAM

- Limited capacity
- + Low latency

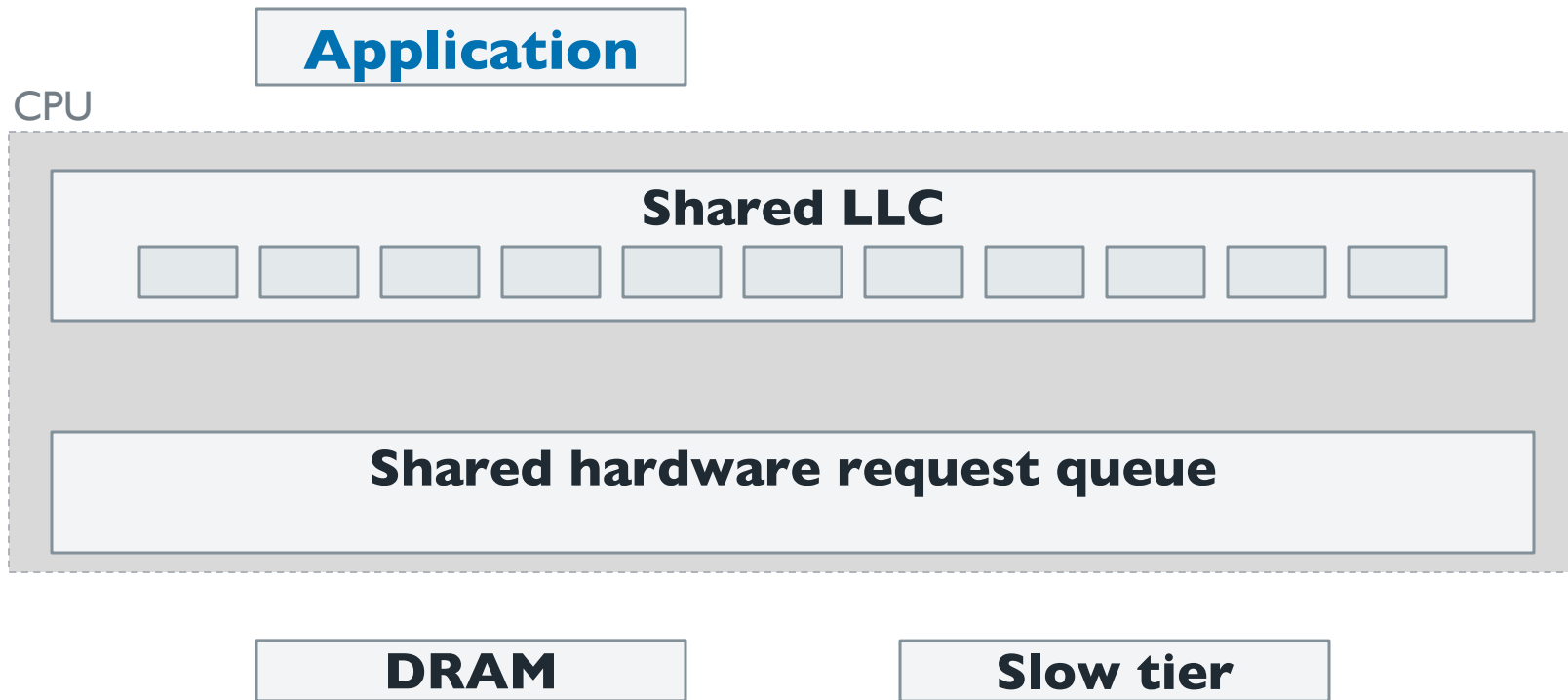


Slow: memory expansion

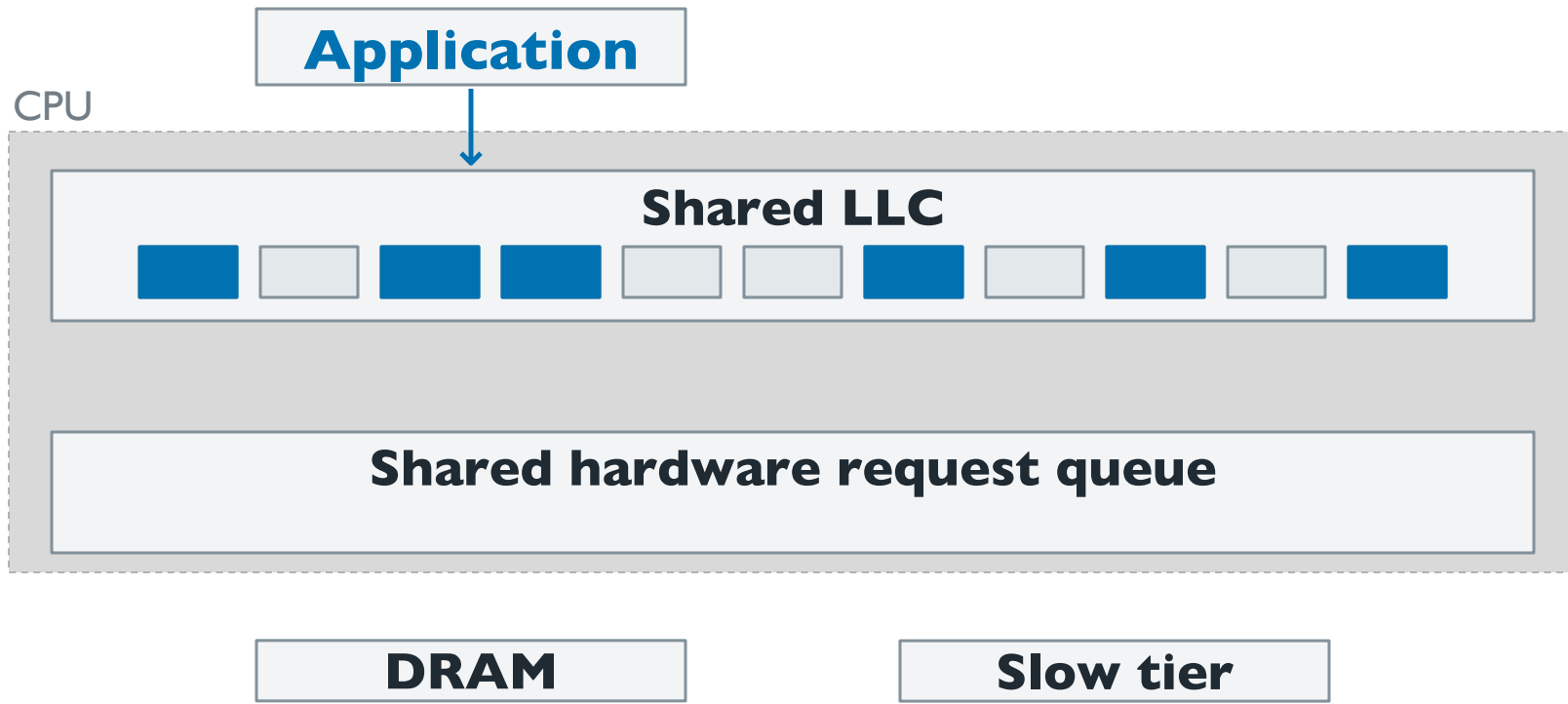
- + More capacity
- Higher latency

The challenge is deciding which pages belong in fast memory.

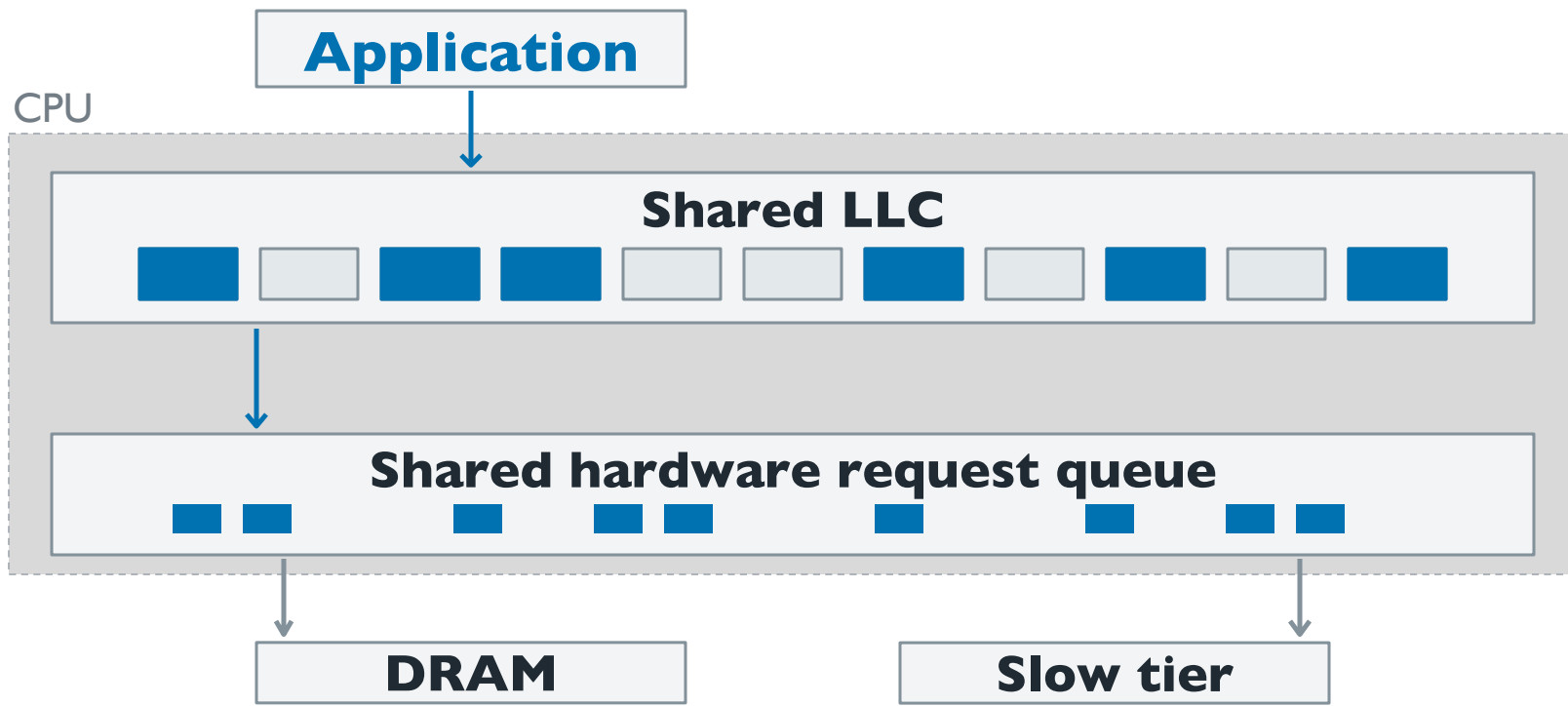
Colocated Workloads on Tiered Memory



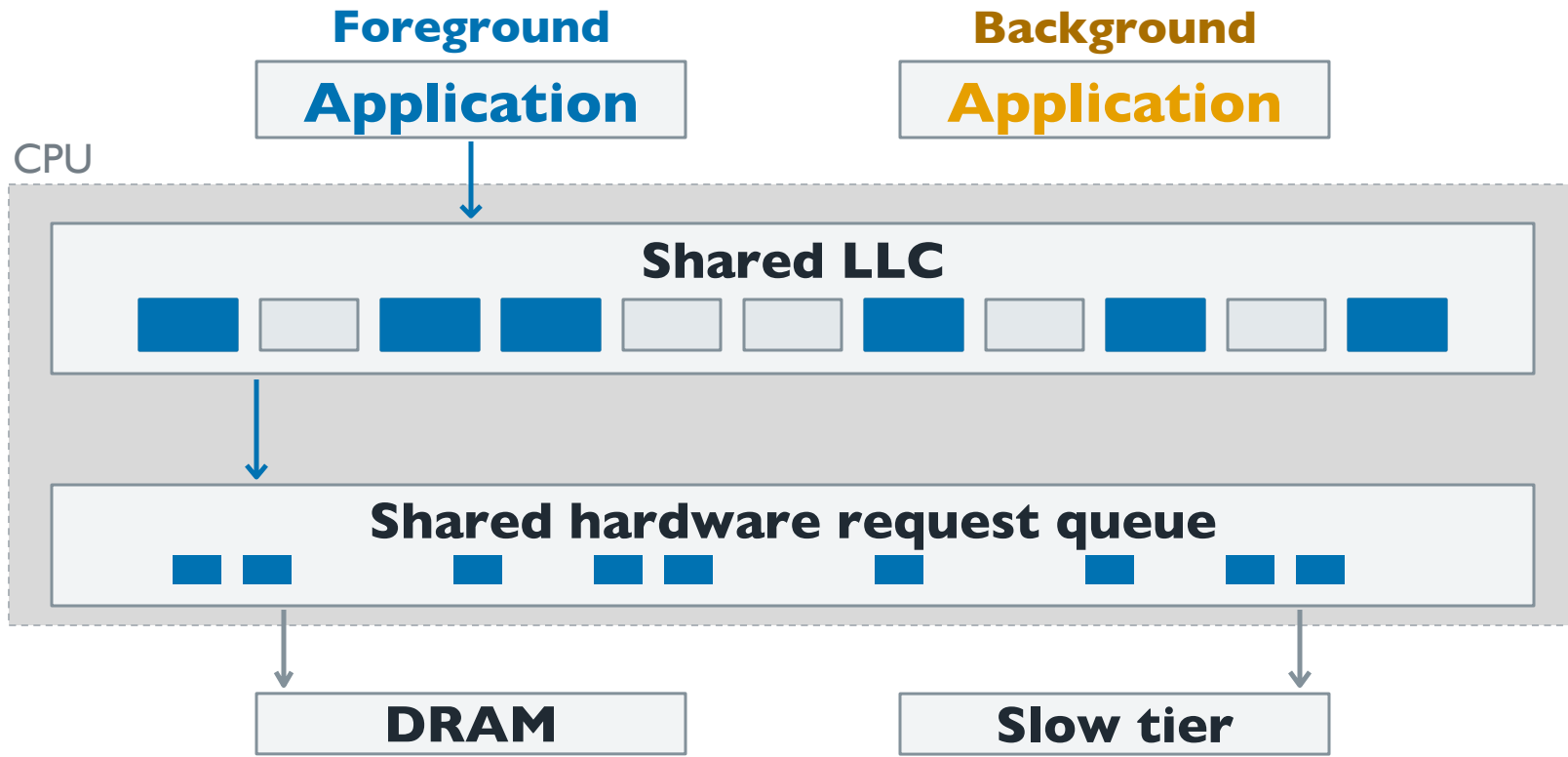
Colocated Workloads on Tiered Memory



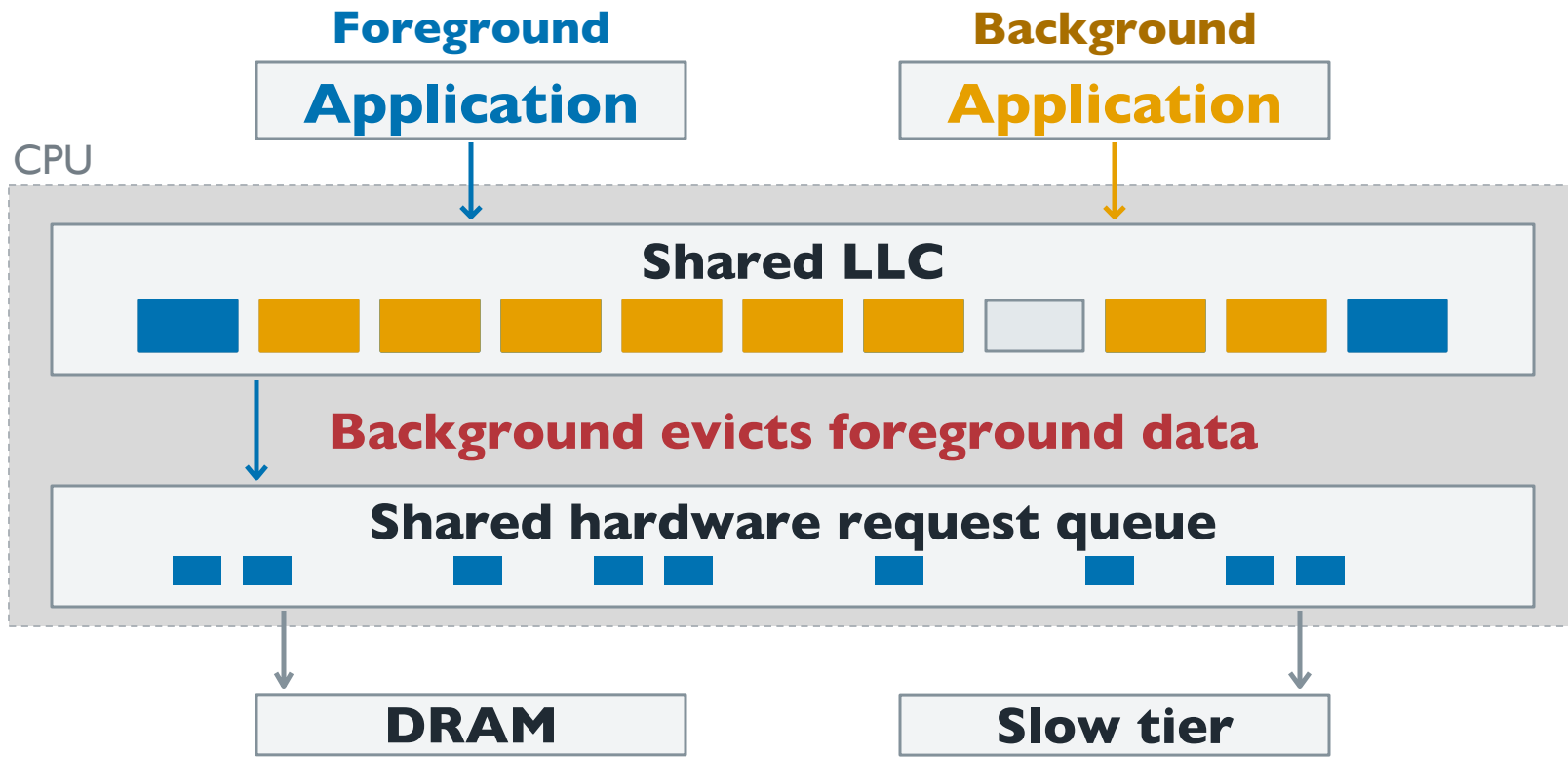
Colocated Workloads on Tiered Memory



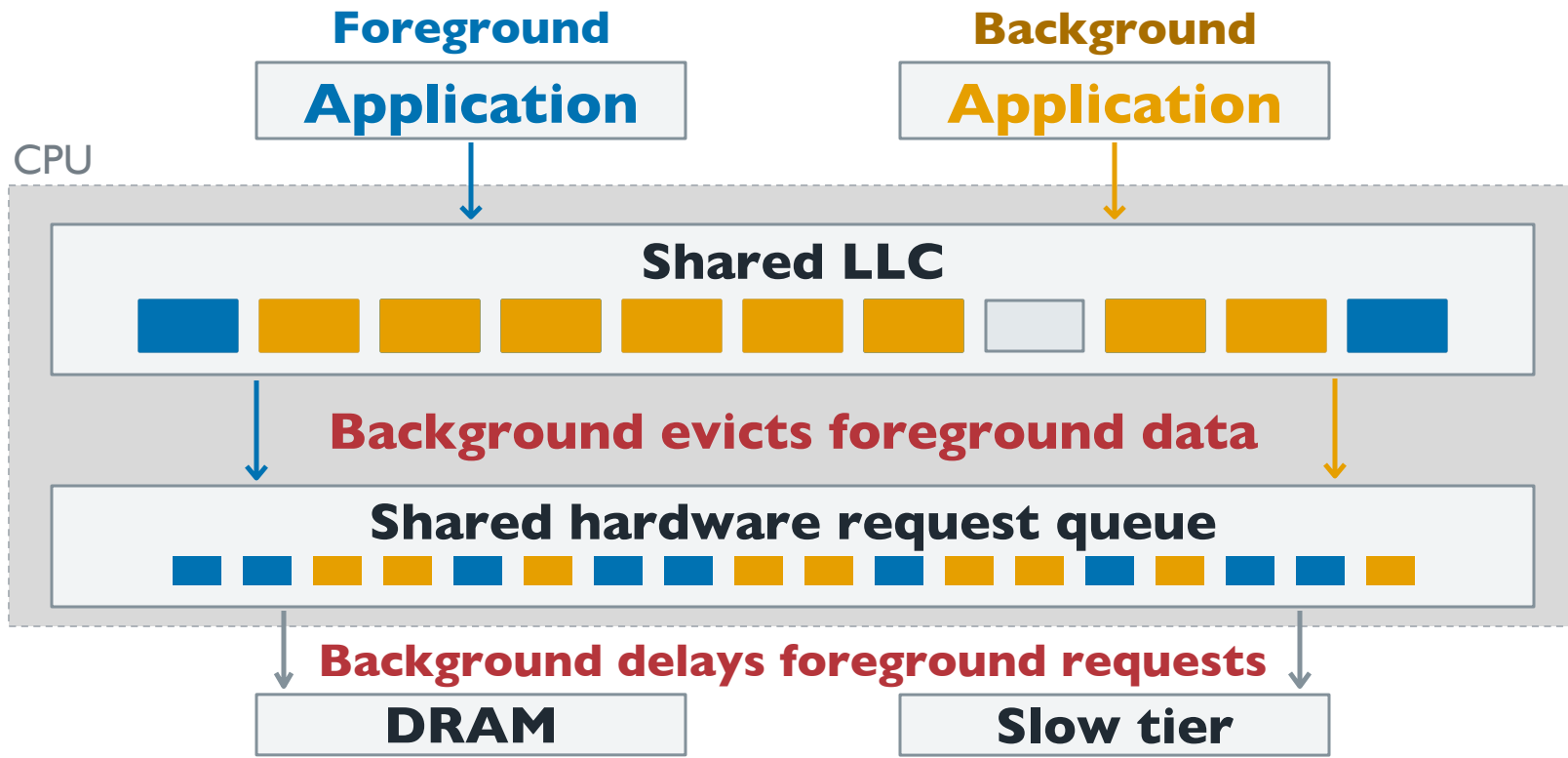
Colocated Workloads on Tiered Memory



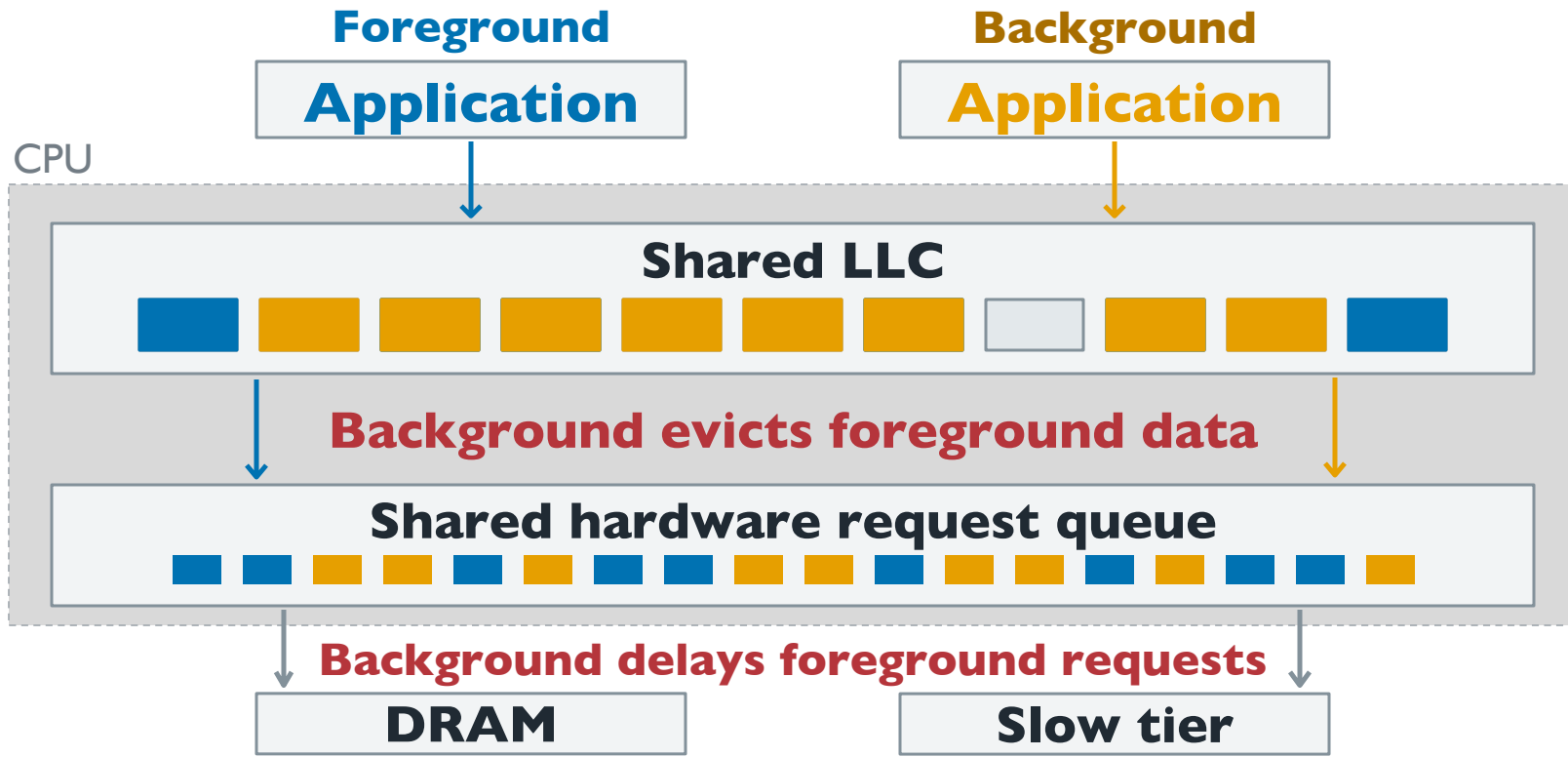
Colocated Workloads on Tiered Memory



Colocated Workloads on Tiered Memory



Colocated Workloads on Tiered Memory

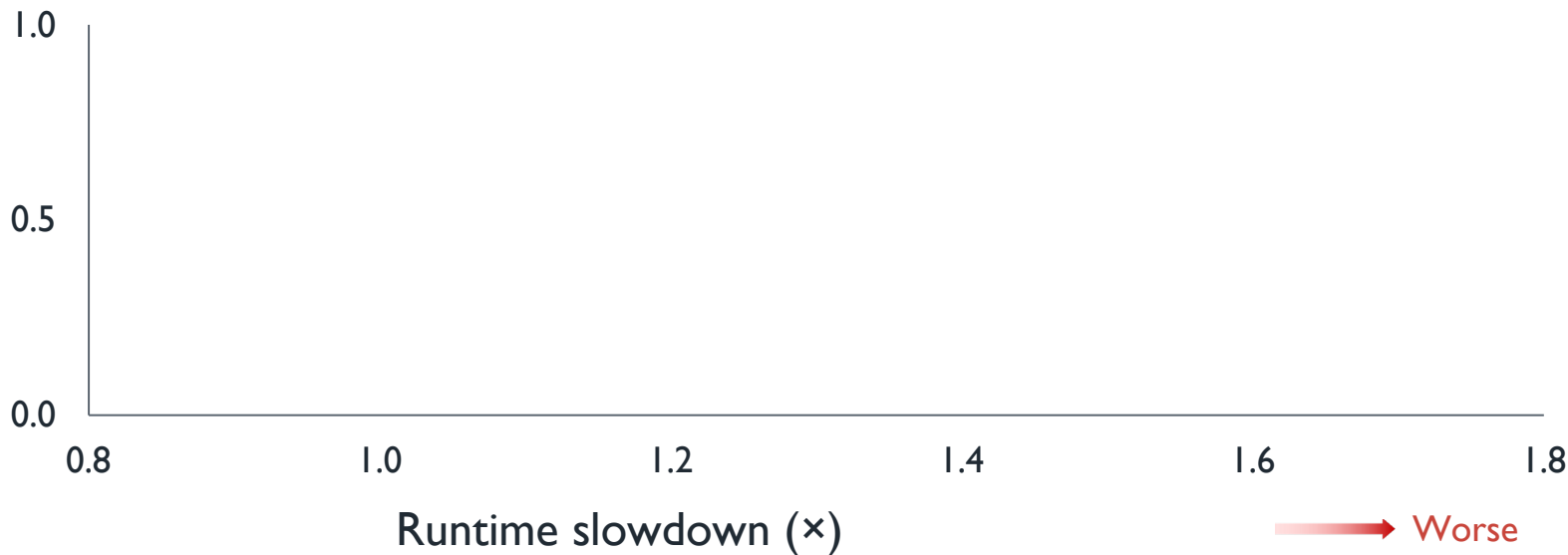


Colocation can increase LLC misses and memory waiting time.

Slower Memory Increases Colocation Slowdown

Slowdown (×) = colocated ÷ solo

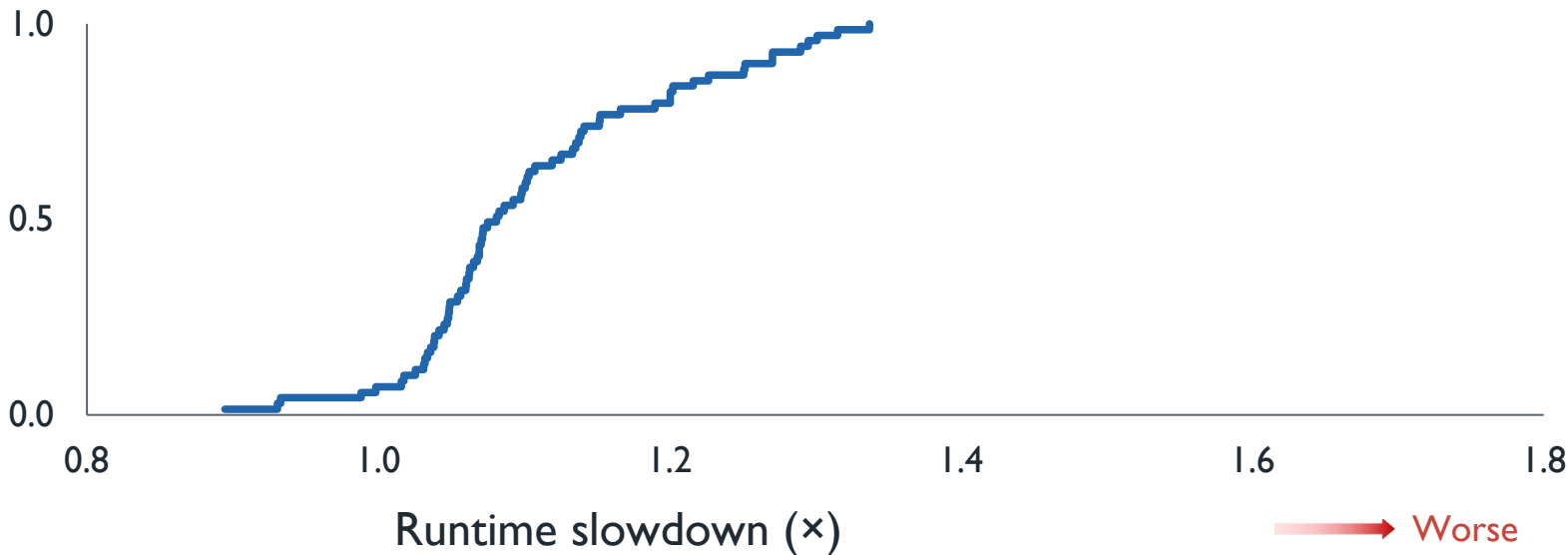
CDF of colocation runs



Slower Memory Increases Colocation Slowdown

Slowdown (×) = colocated ÷ solo

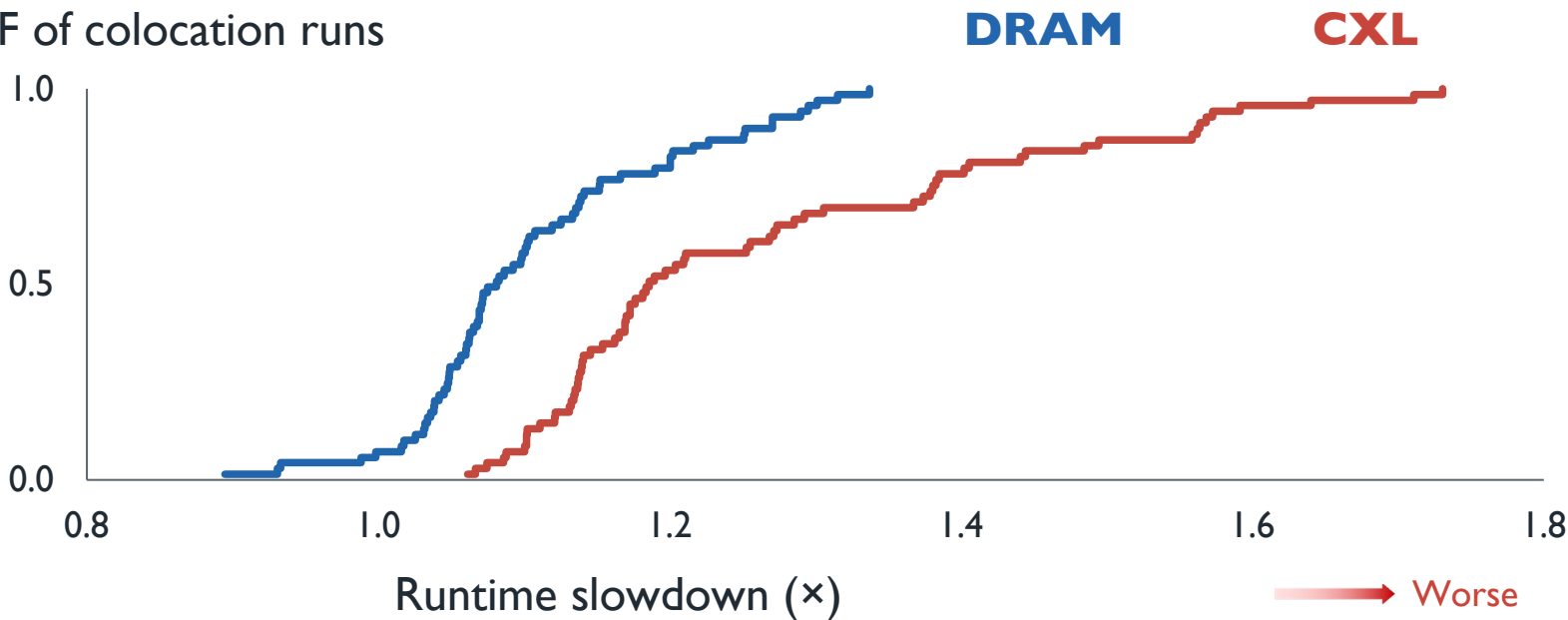
CDF of colocation runs



Slower Memory Increases Colocation Slowdown

Slowdown (×) = colocated ÷ solo

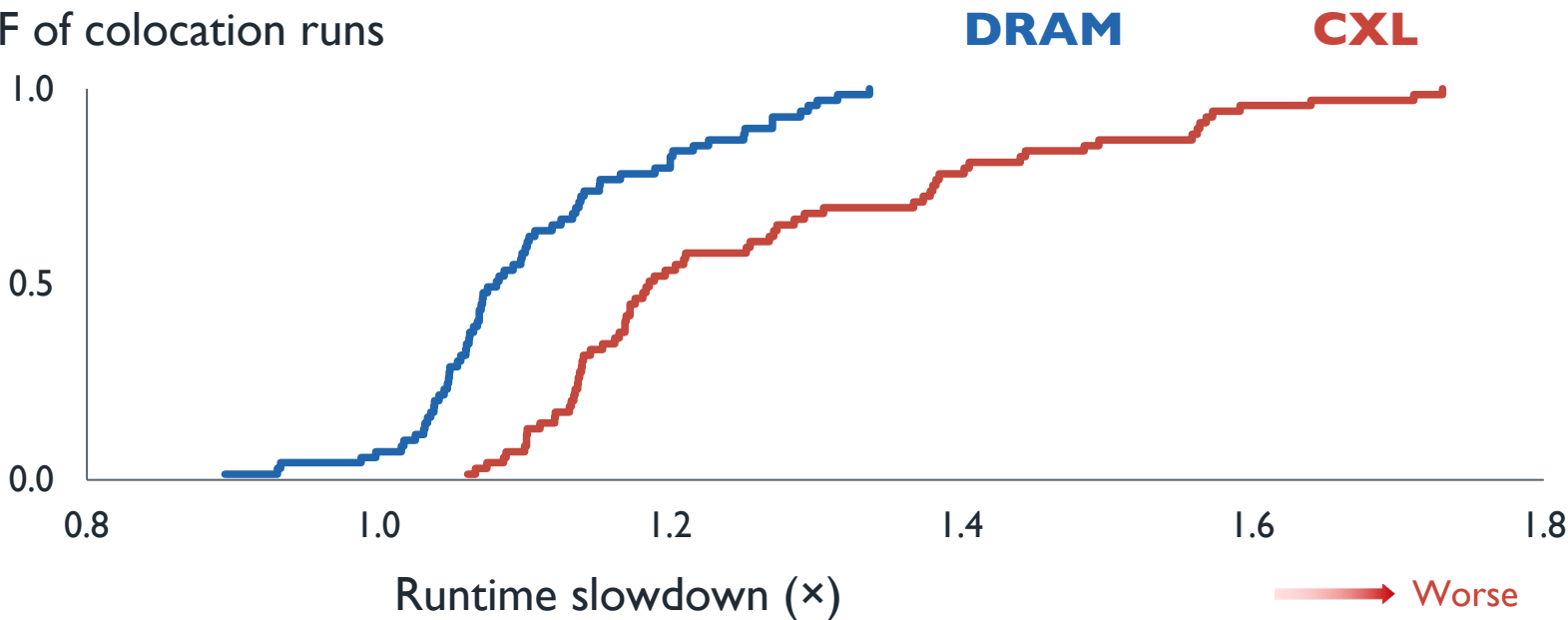
CDF of colocation runs



Slower Memory Increases Colocation Slowdown

Slowdown (×) = colocated ÷ solo

CDF of colocation runs

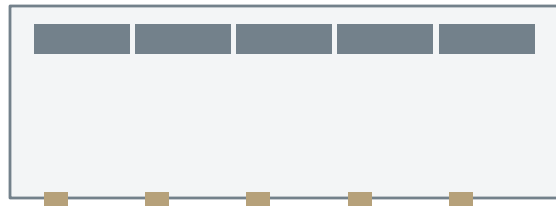


Slower memory amplifies colocation slowdown, from a median of 1.08× to 1.19×.

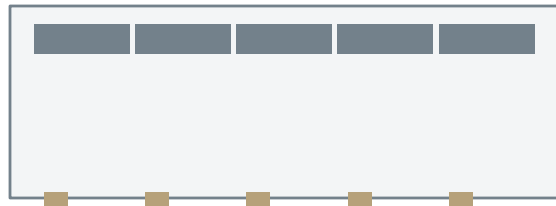
Hotness Prioritizes Frequently Accessed Pages



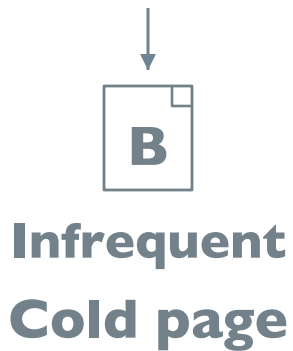
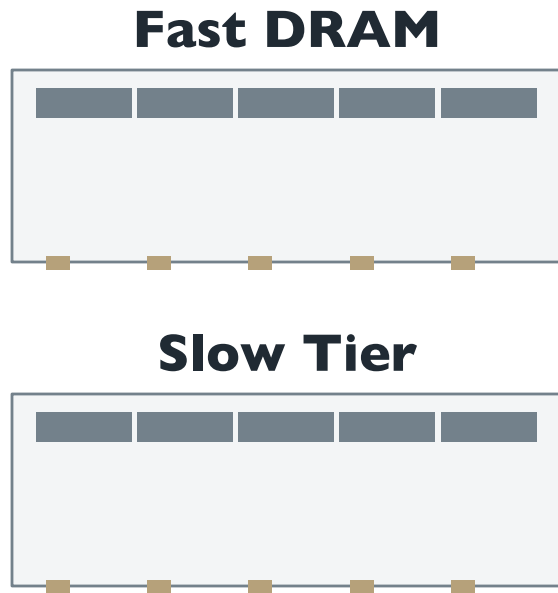
Fast DRAM



Slow Tier



Hotness Prioritizes Frequently Accessed Pages

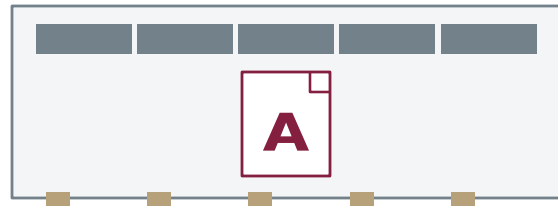


Hotness Prioritizes Frequently Accessed Pages

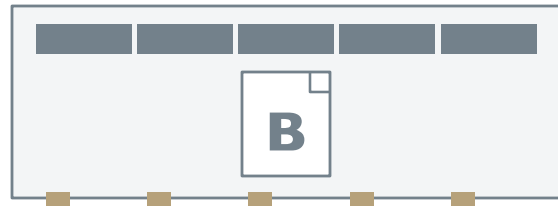
**Frequent
Hot page**

**Infrequent
Cold page**

Fast DRAM



Slow Tier



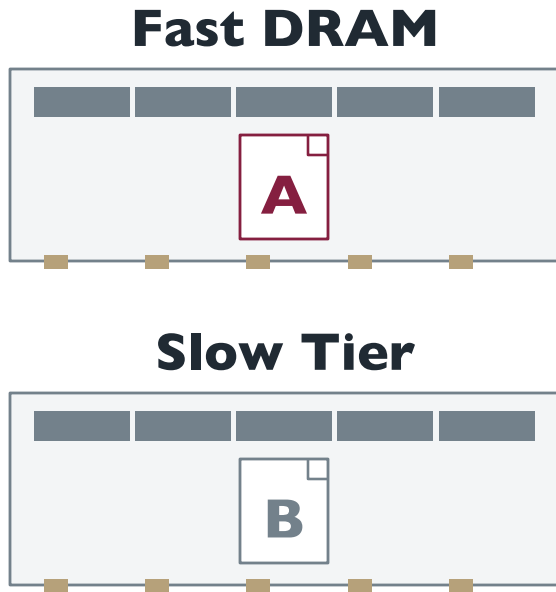
Hotness Prioritizes Frequently Accessed Pages

**Frequent
Hot page**

**Infrequent
Cold page**

Examples: NBT and Memtis

- NBT: Linux memory tiering
- Memtis: SOSP 2023



Frequent accesses do not necessarily save the most time in fast memory.

Overlap Changes the Benefit of Fast Memory

Dependent accesses

Three requests

Slow tier



Independent accesses

Three requests

Slow tier



Overlap Changes the Benefit of Fast Memory

Dependent accesses

Three requests

Slow tier



Fast tier



Independent accesses

Three requests

Slow tier



Overlap Changes the Benefit of Fast Memory

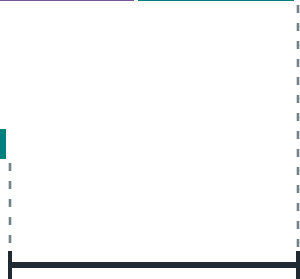
Dependent accesses

Three requests

Slow tier



Fast tier



Time saved in fast memory

Independent accesses

Three requests

Slow tier



Overlap Changes the Benefit of Fast Memory

Dependent accesses

Three requests

Slow tier



Fast tier



Time saved in fast memory

Independent accesses

Three requests

Slow tier



Fast tier



Overlap Changes the Benefit of Fast Memory

Dependent accesses

Three requests

Slow tier



Fast tier



Time saved in fast memory

Independent accesses

Three requests

Slow tier



Fast tier



Time saved in fast memory

Overlap Changes the Benefit of Fast Memory

Dependent accesses

Three requests

Slow tier



Fast tier



Time saved in fast memory

$$\text{MLP} = 1$$

Independent accesses

Three requests

Slow tier



Fast tier



Time saved in fast memory

$$\text{MLP} = 3$$

Overlap Changes the Benefit of Fast Memory

Dependent accesses

Three requests

Slow tier



Fast tier



Time saved in fast memory

$$\text{MLP} = 1$$

Independent accesses

Three requests

Slow tier



Fast tier



Time saved in fast memory

$$\text{MLP} = 3$$

When requests overlap, fast memory has less waiting time to save.

PAC Prioritizes Performance-Critical Pages

Page Access Criticality (PAC)

Estimates the stall time fast memory could save



Score $\propto$

PAC Prioritizes Performance-Critical Pages

Page Access Criticality (PAC)

Estimates the stall time fast memory could save



Score $\propto$ **Accesses**

More accesses



More stalls to save

PAC Prioritizes Performance-Critical Pages

Page Access Criticality (PAC)

Estimates the stall time fast memory could save



Score $\propto$

Accesses $\times$ **Fixed cost**

More accesses



More stalls to save

Fixed miss cost



PACT uses the same cost
for every slow-tier miss

PAC Prioritizes Performance-Critical Pages

Page Access Criticality (PAC)

Estimates the stall time fast memory could save



$$\text{Score} \propto \frac{\text{Accesses} \times \text{Fixed cost}}{\text{MLP}}$$

More accesses



More stalls to save

Fixed miss cost



PACT uses the same cost
for every slow-tier miss

More overlap (MLP)



Less waiting time to save

PAC Prioritizes Performance-Critical Pages

Page Access Criticality (PAC)

Estimates the stall time fast memory could save



$$\text{Score} \propto \frac{\text{Accesses} \times \text{Fixed cost}}{\text{MLP}}$$

More accesses



More stalls to save

Fixed miss cost



PACT uses the same cost
for every slow-tier miss

More overlap (MLP)



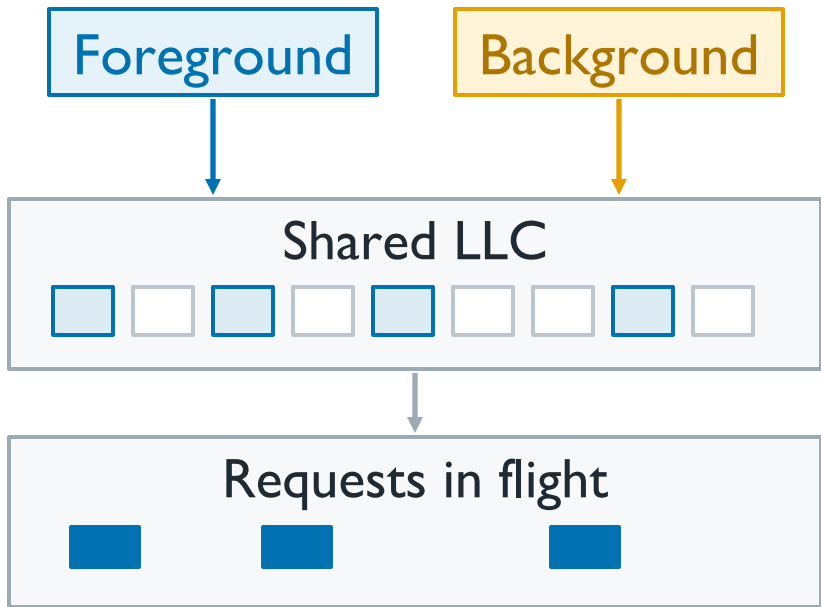
Less waiting time to save

PACT: A Criticality-First Design for Tiered Memory · ASPLOS 2026

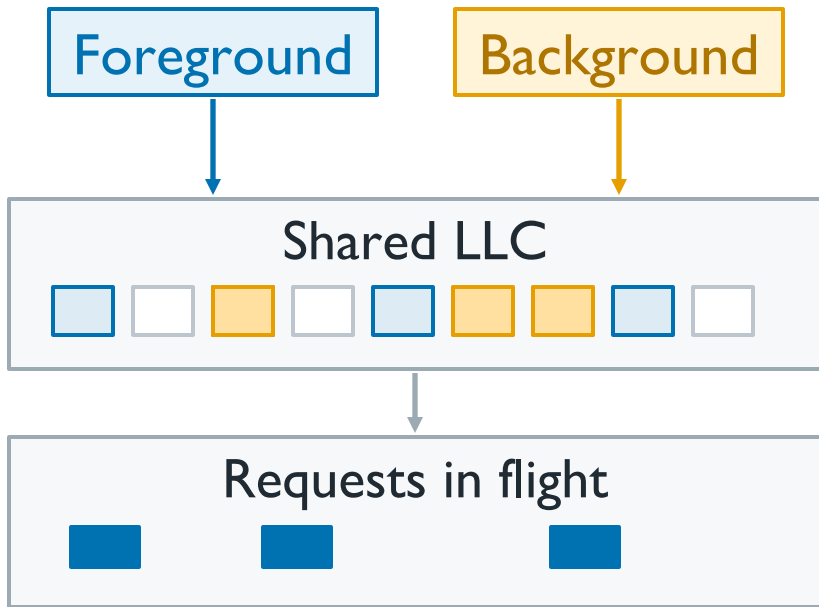
We need accurate PAC estimates to choose which pages to migrate.

Colocation Mixes the Signals Used by Tiering

Observability gap



Colocation Mixes the Signals Used by Tiering

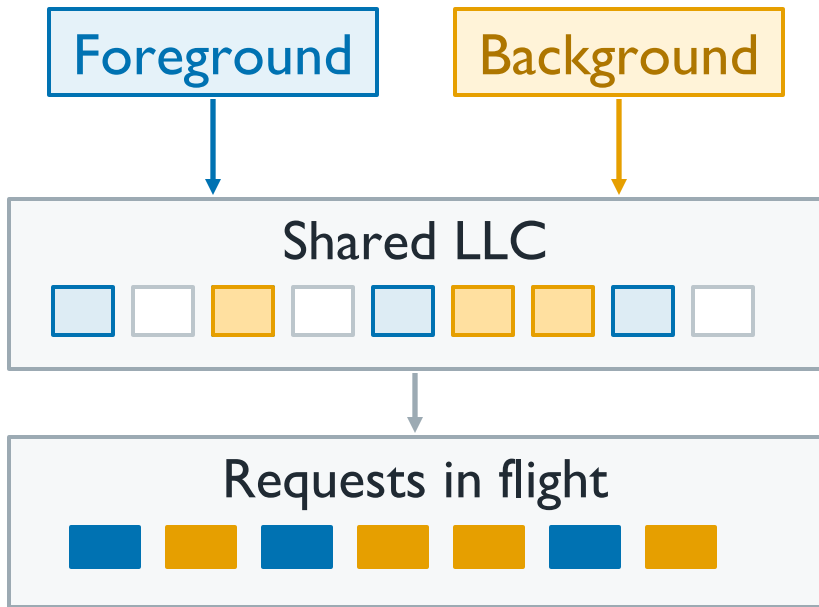


Observability gap

Space aliasing

LLC interference changes which accesses reach memory.

Colocation Mixes the Signals Used by Tiering



Observability gap

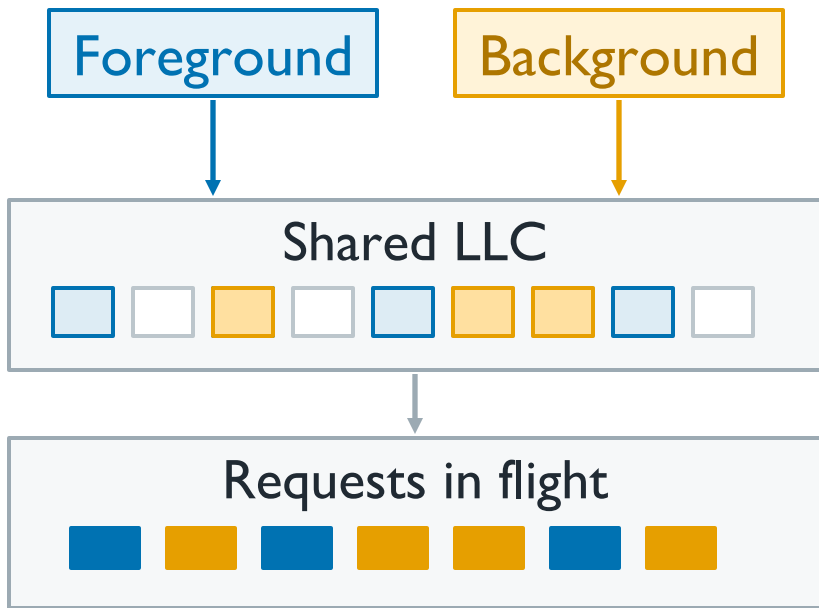
Space aliasing

LLC interference changes which accesses reach memory.

Time aliasing

Measurements mix workloads' overlap.

Colocation Mixes the Signals Used by Tiering



Observability gap

Space aliasing

LLC interference changes which accesses reach memory.

Time aliasing

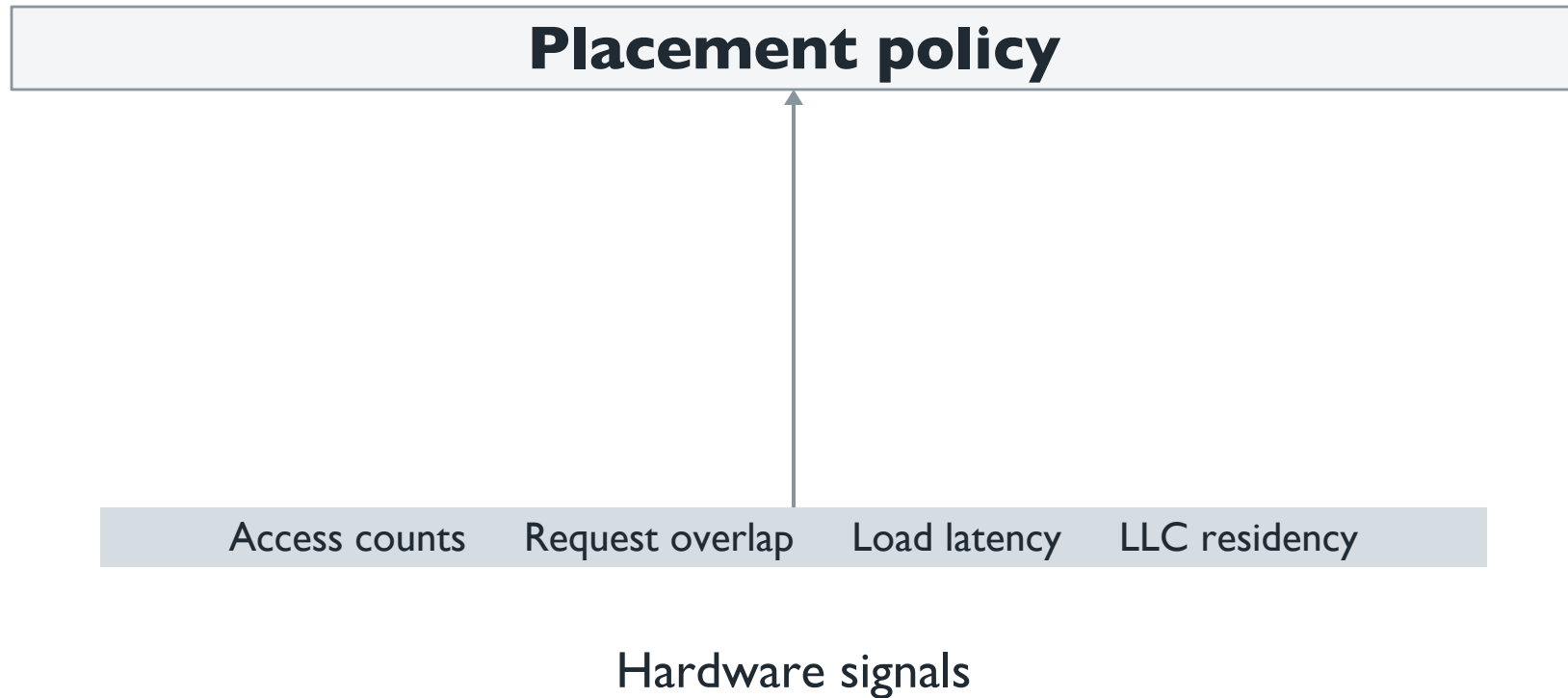
Measurements mix workloads' overlap.

Space and time aliasing distort the measurements that tiering policies rely on.

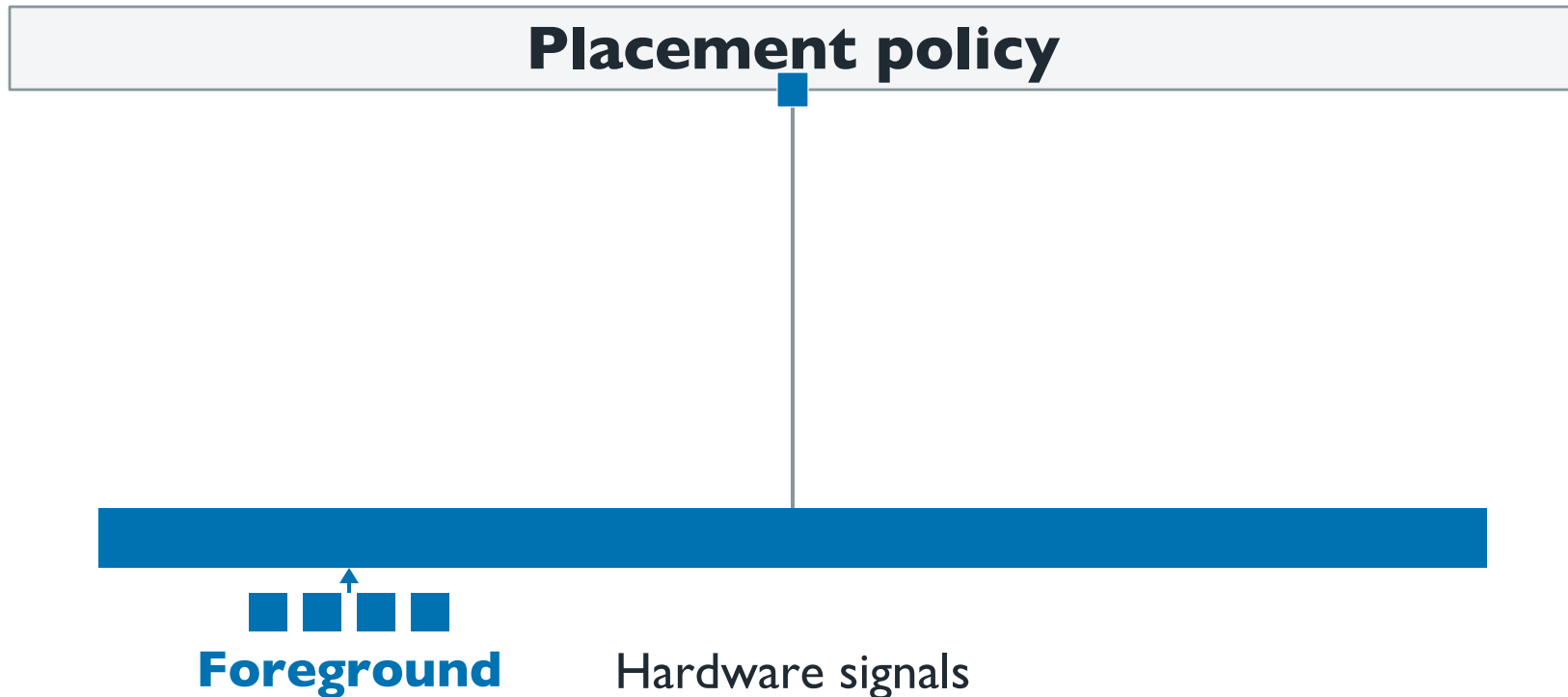
Tierce Separates the Signals by Workload

Placement policy

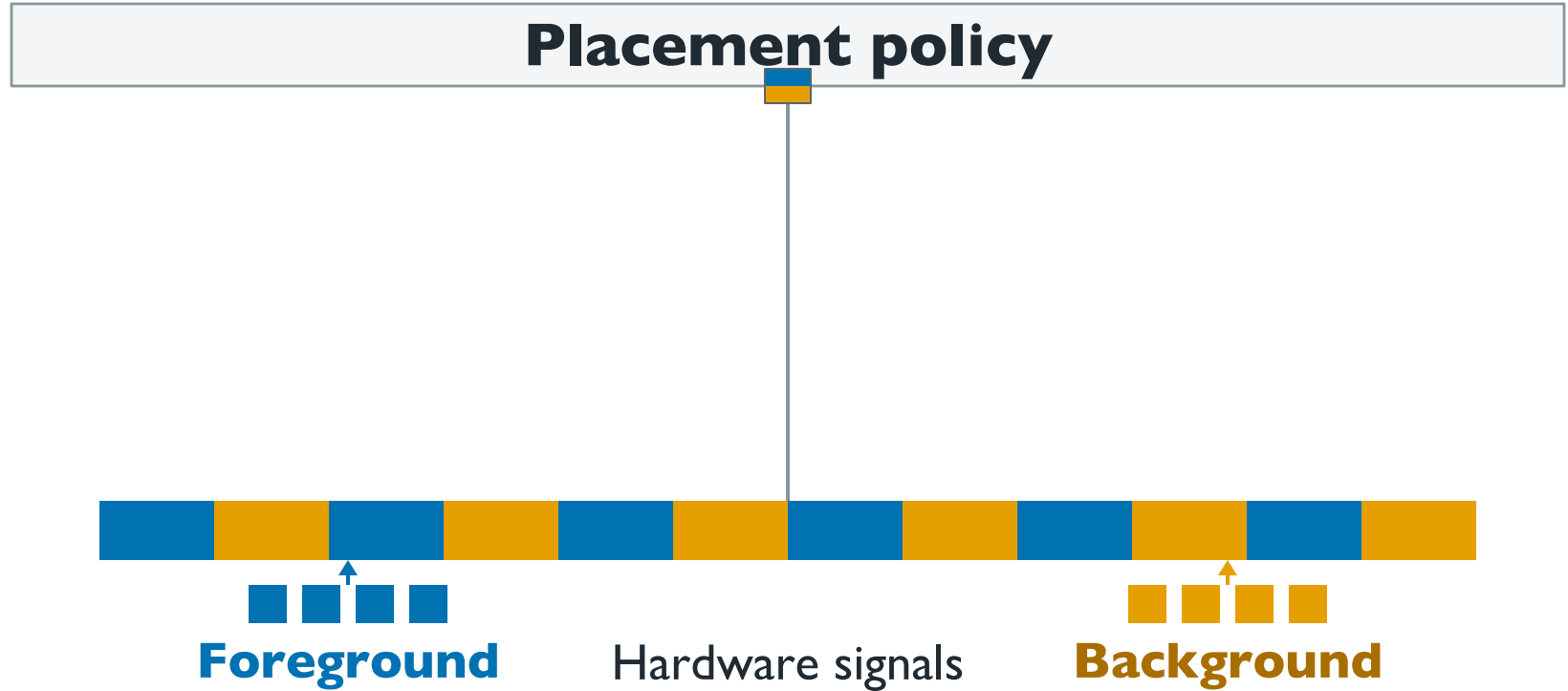
Tierce Separates the Signals by Workload



Tierce Separates the Signals by Workload



Tierce Separates the Signals by Workload



Tierce Separates the Signals by Workload

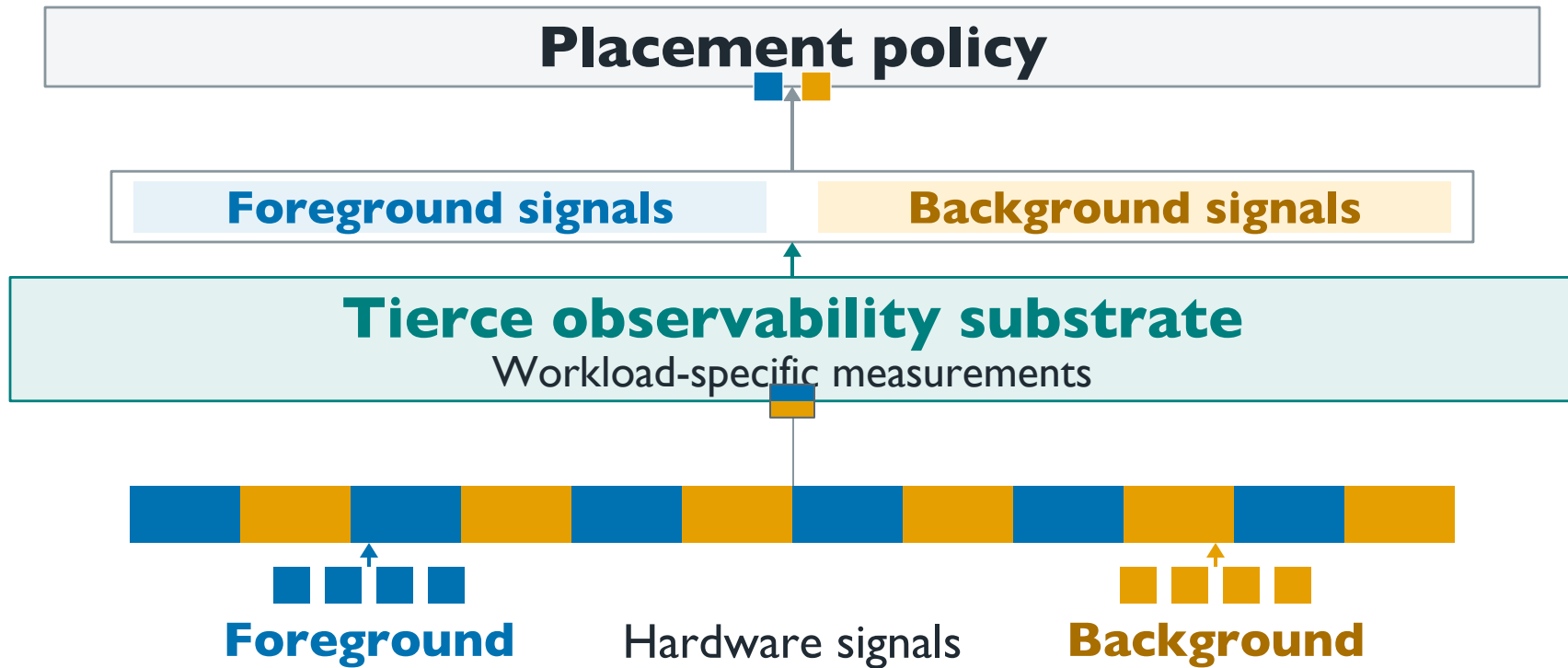
Placement policy

Tierce observability substrate

Workload-specific measurements

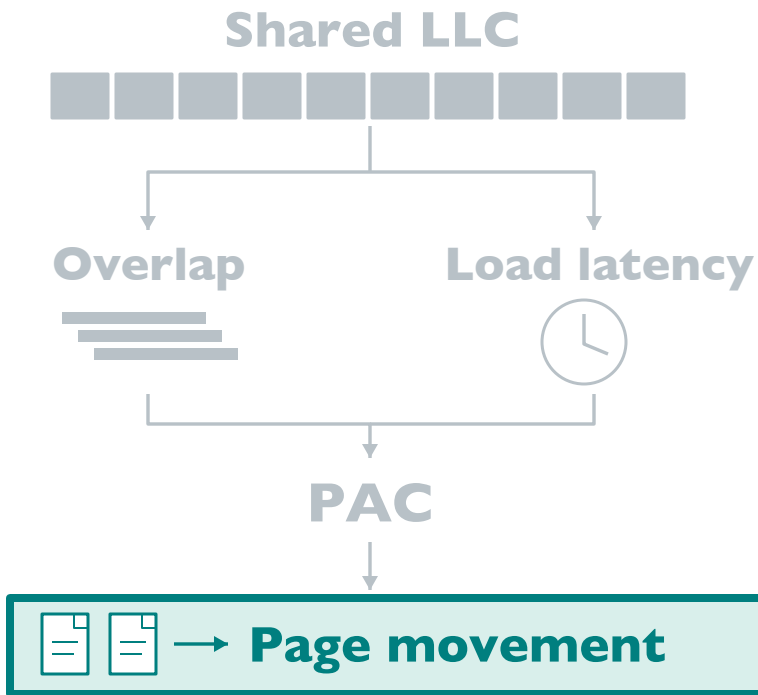


Tierce Separates the Signals by Workload



Tierce separates hardware measurements by workload.

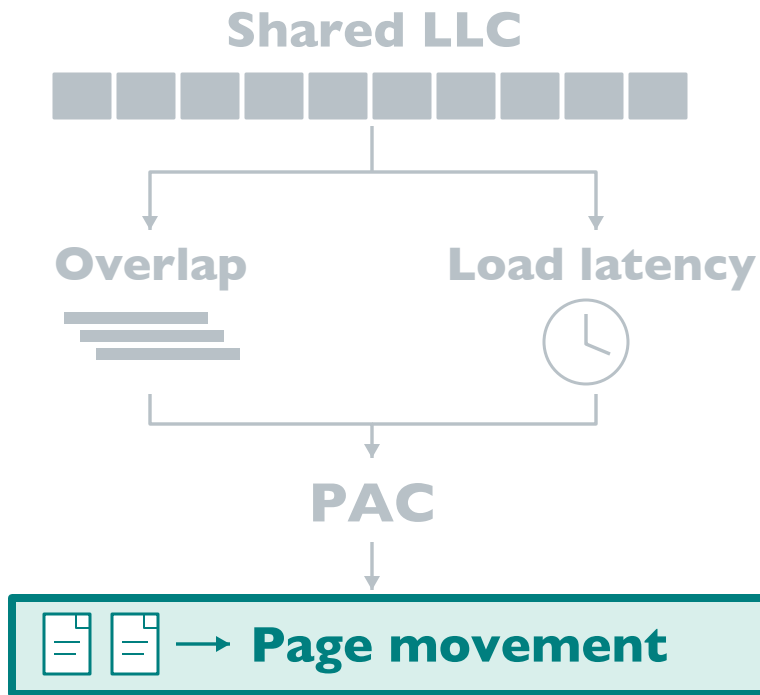
Gap #1: Keep Page Placement Per Workload



Existing

- A **shared promotion queue**
- Linux: reclaim **per NUMA node**

Gap #1: Keep Page Placement Per Workload



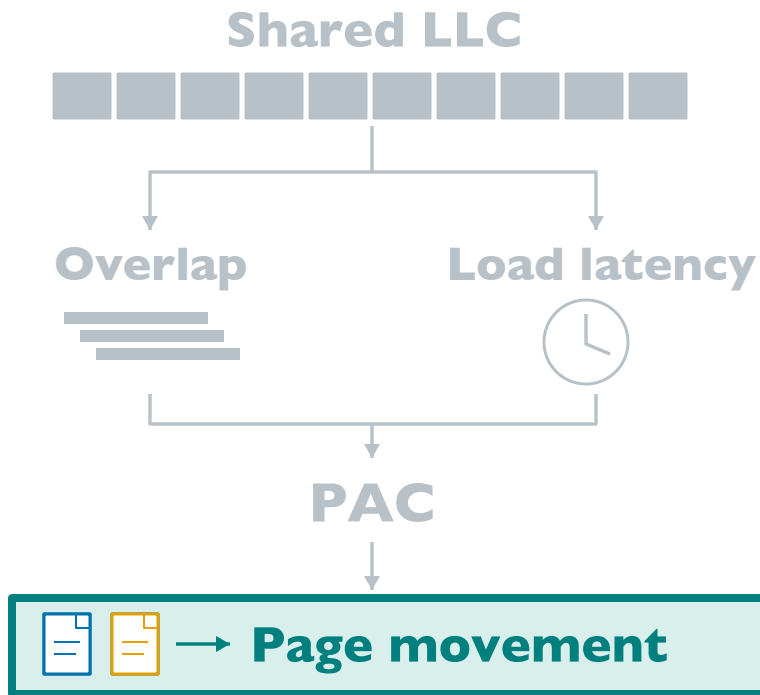
Existing

- A **shared promotion queue**
- Linux: reclaim **per NUMA node**

Under colocation

- Background pages can dominate promotions and occupy DRAM

Gap #1: Keep Page Placement Per Workload



Existing

- A **shared promotion queue**
- Linux: reclaim **per NUMA node**

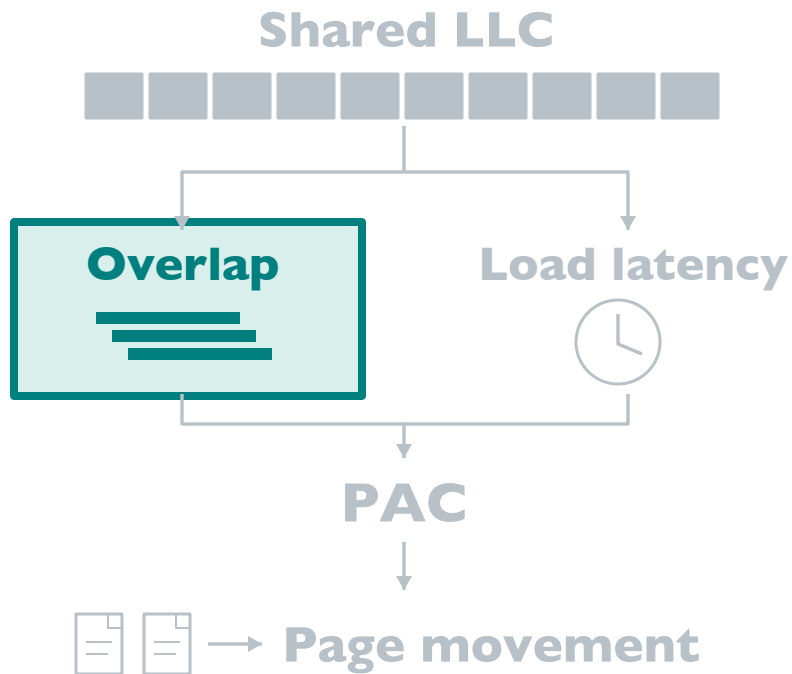
Under colocation

- Background pages can dominate promotions and occupy DRAM

Tierce

- **Private queues** and **per-tier limits**

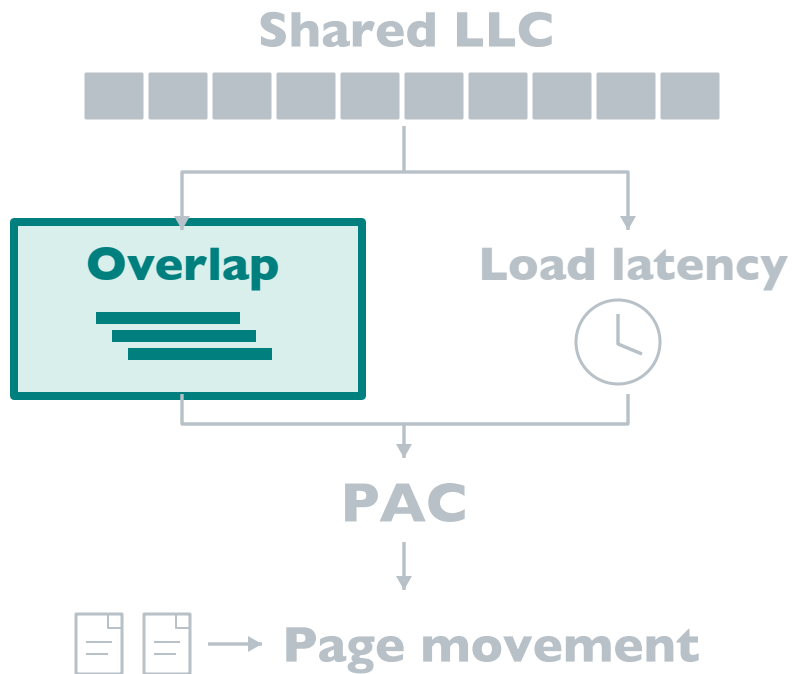
Gap #2: Measure the Workload's Own Parallelism



Existing

- **Aggregate counters** mix foreground and background requests

Gap #2: Measure the Workload's Own Parallelism



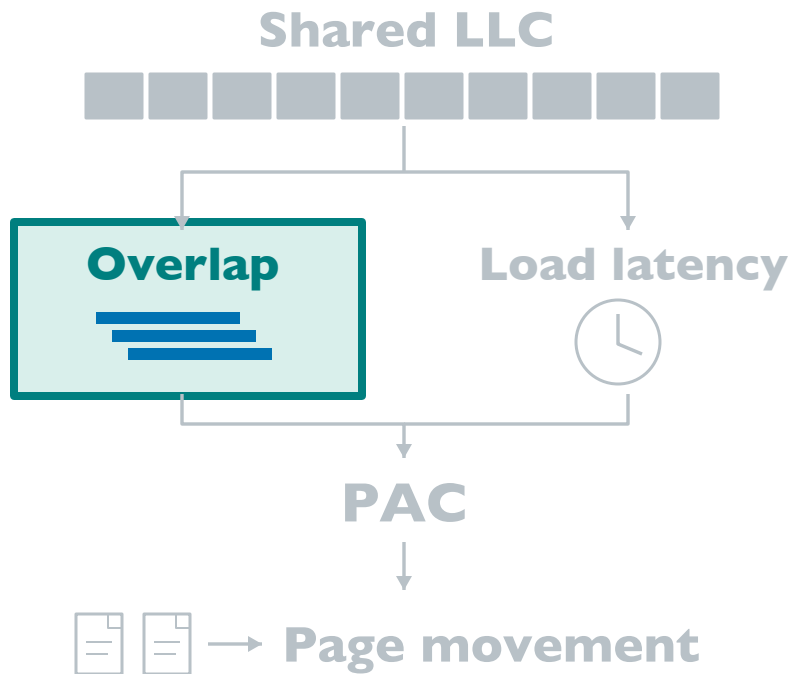
Existing

- **Aggregate counters** mix foreground and background requests

Under colocation

- Background requests raise measured MLP and lower foreground scores

Gap #2: Measure the Workload's Own Parallelism



Existing

- **Aggregate counters** mix foreground and background requests

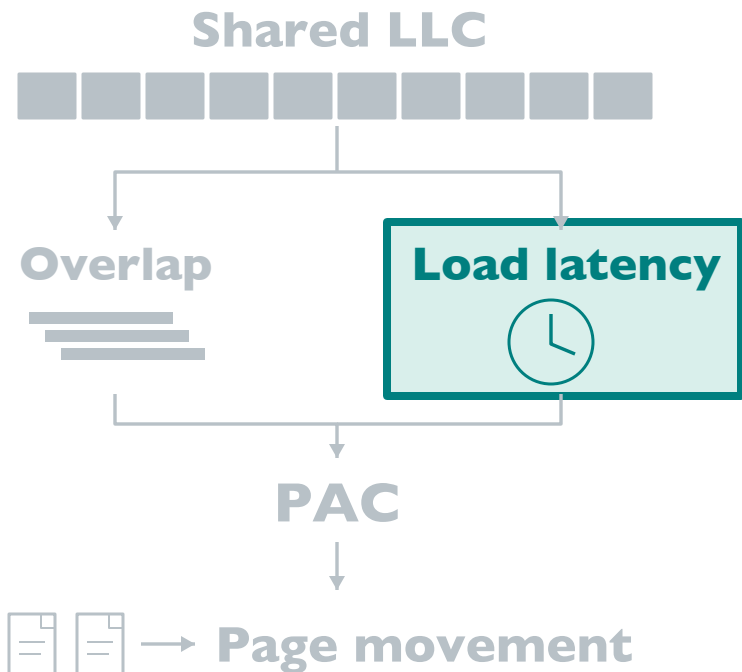
Under colocation

- Background requests raise measured MLP and lower foreground scores

Tierce

- **Workload-specific MLP (wMLP)**

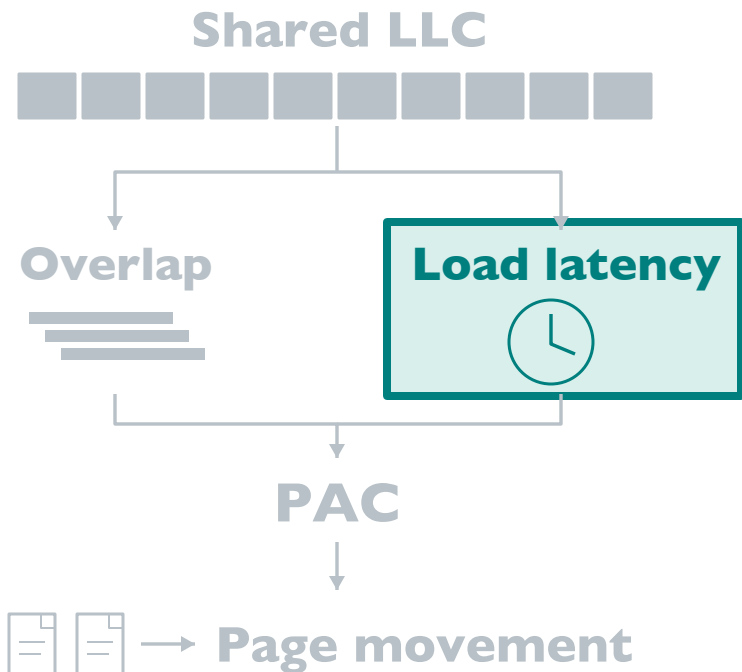
Gap #3: Focus on Costly Accesses



Existing

- PACT gives slow-tier accesses the **same fixed cost**

Gap #3: Focus on Costly Accesses



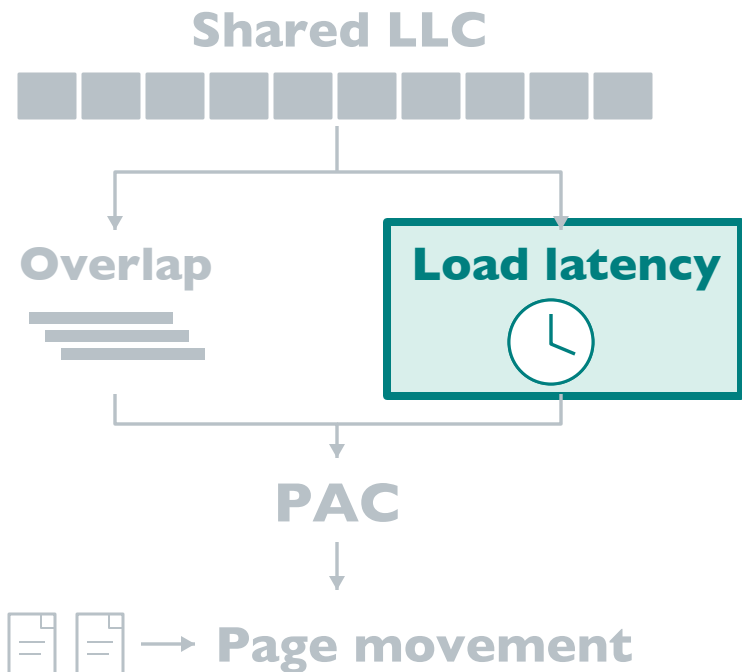
Existing

- PACT gives slow-tier accesses the **same fixed cost**

Under colocation

- Contention makes some accesses much more costly than others

Gap #3: Focus on Costly Accesses



Existing

- PACT gives slow-tier accesses the **same fixed cost**

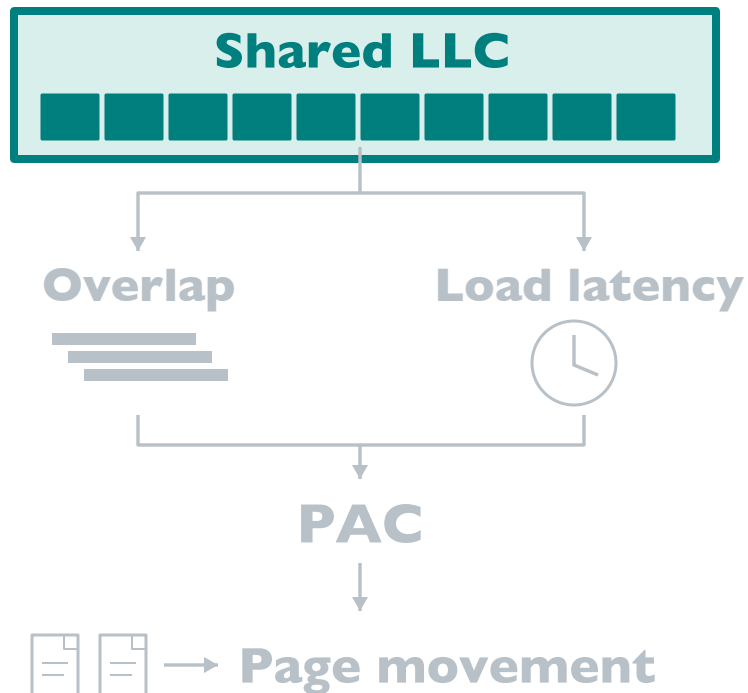
Under colocation

- Contention makes some accesses much more costly than others

Tierce

- **Latency selection** focuses scoring on **costly accesses**

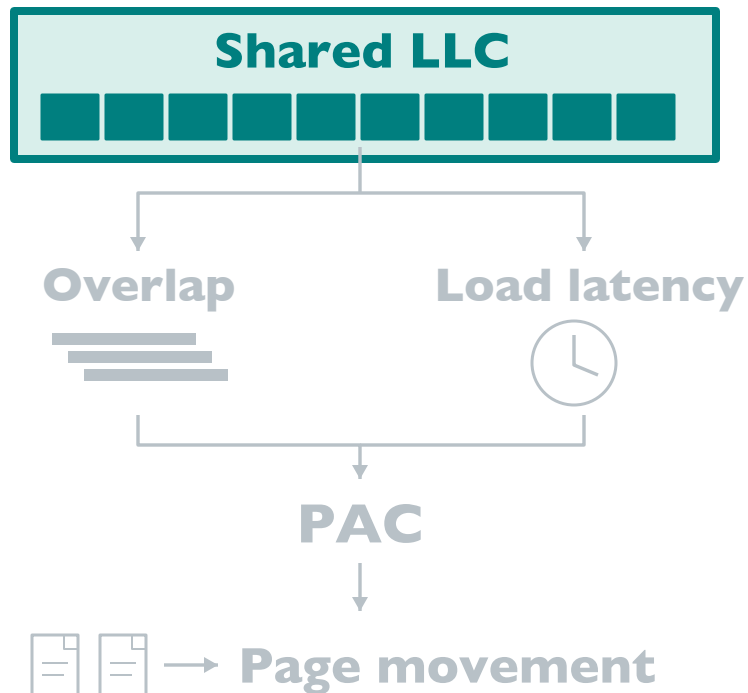
Gap #4: Reduce LLC Interference



Existing

- Page placement alone leaves **shared-LLC interference unmanaged**

Gap #4: Reduce LLC Interference



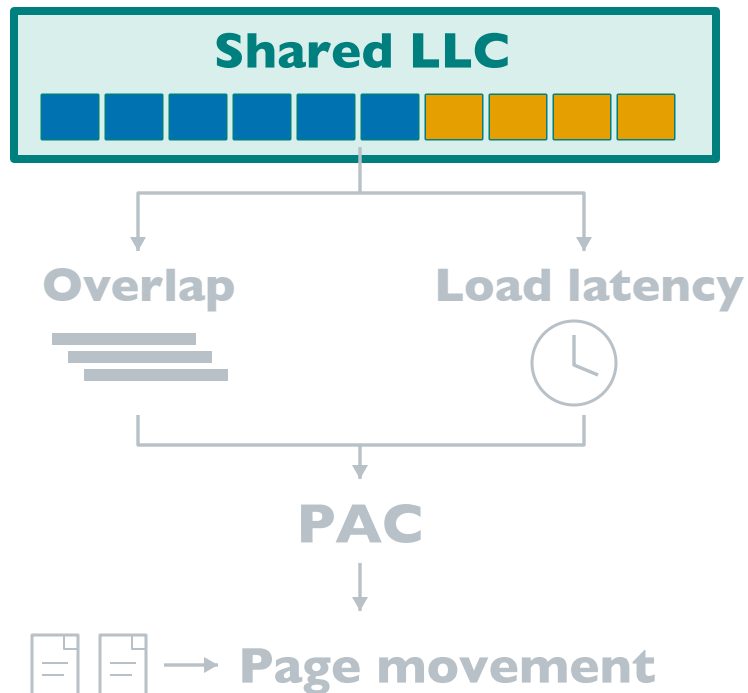
Existing

- Page placement alone leaves **shared-LLC interference unmanaged**

Under colocation

- Background evictions create extra foreground memory accesses

Gap #4: Reduce LLC Interference



Existing

- Page placement alone leaves **shared-LLC interference unmanaged**

Under colocation

- Background evictions create extra foreground memory accesses

Tierce

- **Predictive LLC allocation**

Talk Outline

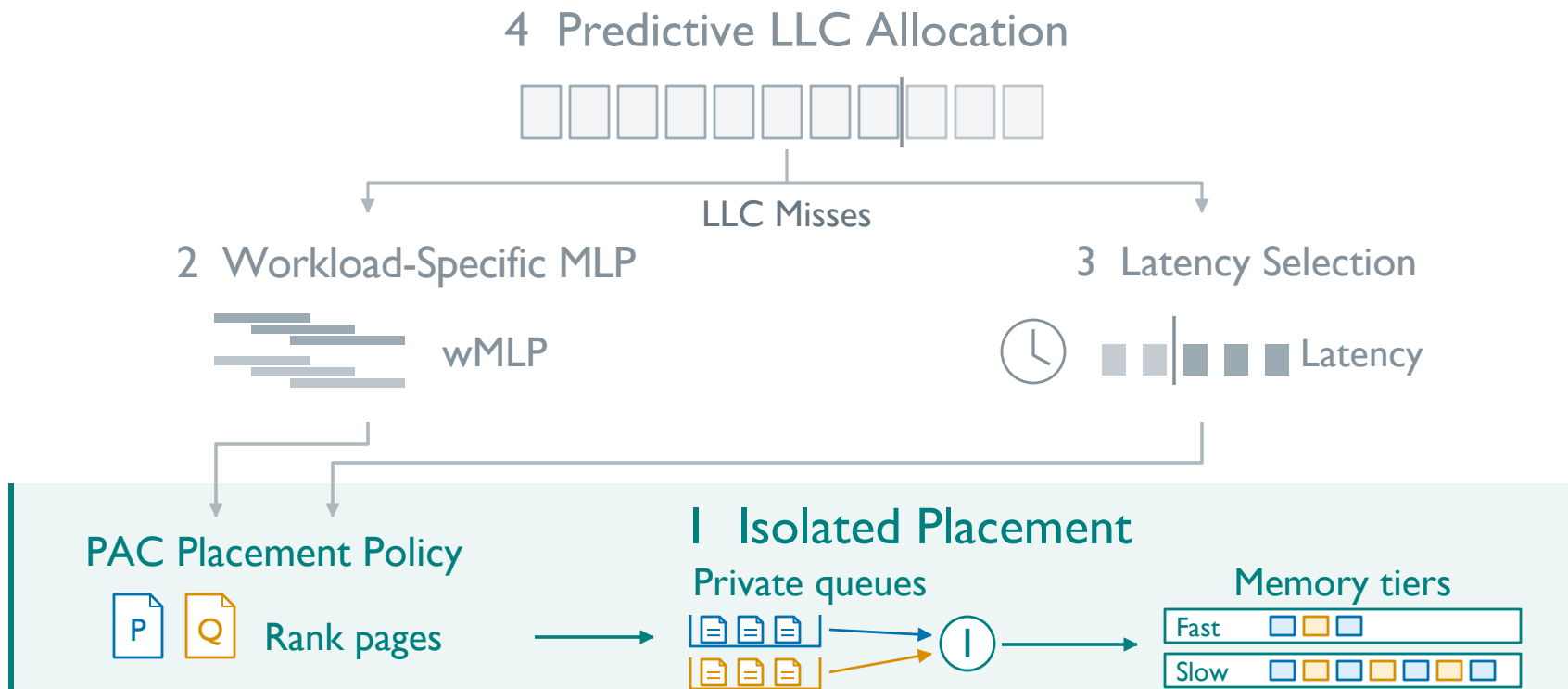
Overview

Tierce Design

1. Isolated placement
2. Workload-specific MLP
3. Latency selection
4. Predictive LLC allocation

Evaluation

I. Keep Placement Per Workload



Separate Queues and Memory Limits

Shared promotion queue



Foreground pages wait

Separate Queues and Memory Limits

Shared promotion queue



Foreground pages wait

Up to 95% of promotions went to the background

Separate Queues and Memory Limits

Shared promotion queue

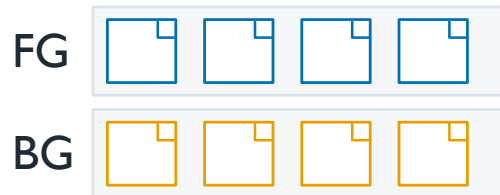


Foreground pages wait

Up to 95% of promotions went to the background



Own promotion queue



Separate Queues and Memory Limits

Shared promotion queue

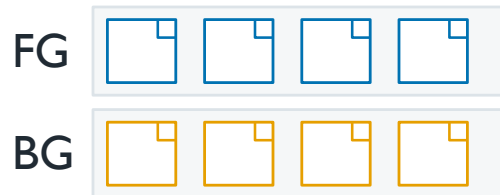


Foreground pages wait

Up to 95% of promotions went to the background



Own promotion queue



DRAM under a total limit



BG fills DRAM; foreground pages have nowhere to go.

Separate Queues and Memory Limits

Shared promotion queue

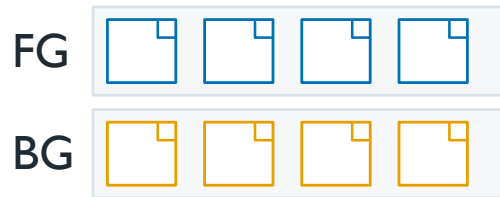


Foreground pages wait

Up to 95% of promotions went to the background



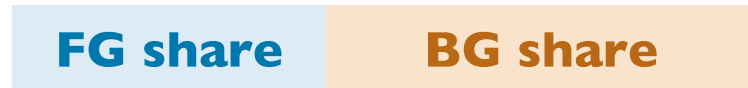
Own promotion queue



DRAM under a total limit



Own DRAM share



BG fills DRAM; foreground pages have nowhere to go.

Separate Queues and Memory Limits

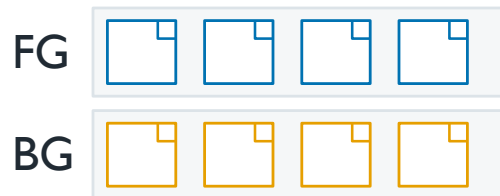
Shared promotion queue



Foreground pages wait

Up to 95% of promotions went to the background

Own promotion queue

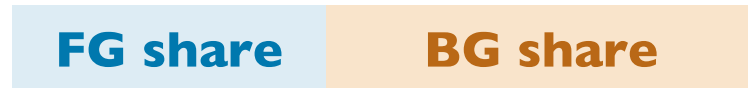


DRAM under a total limit



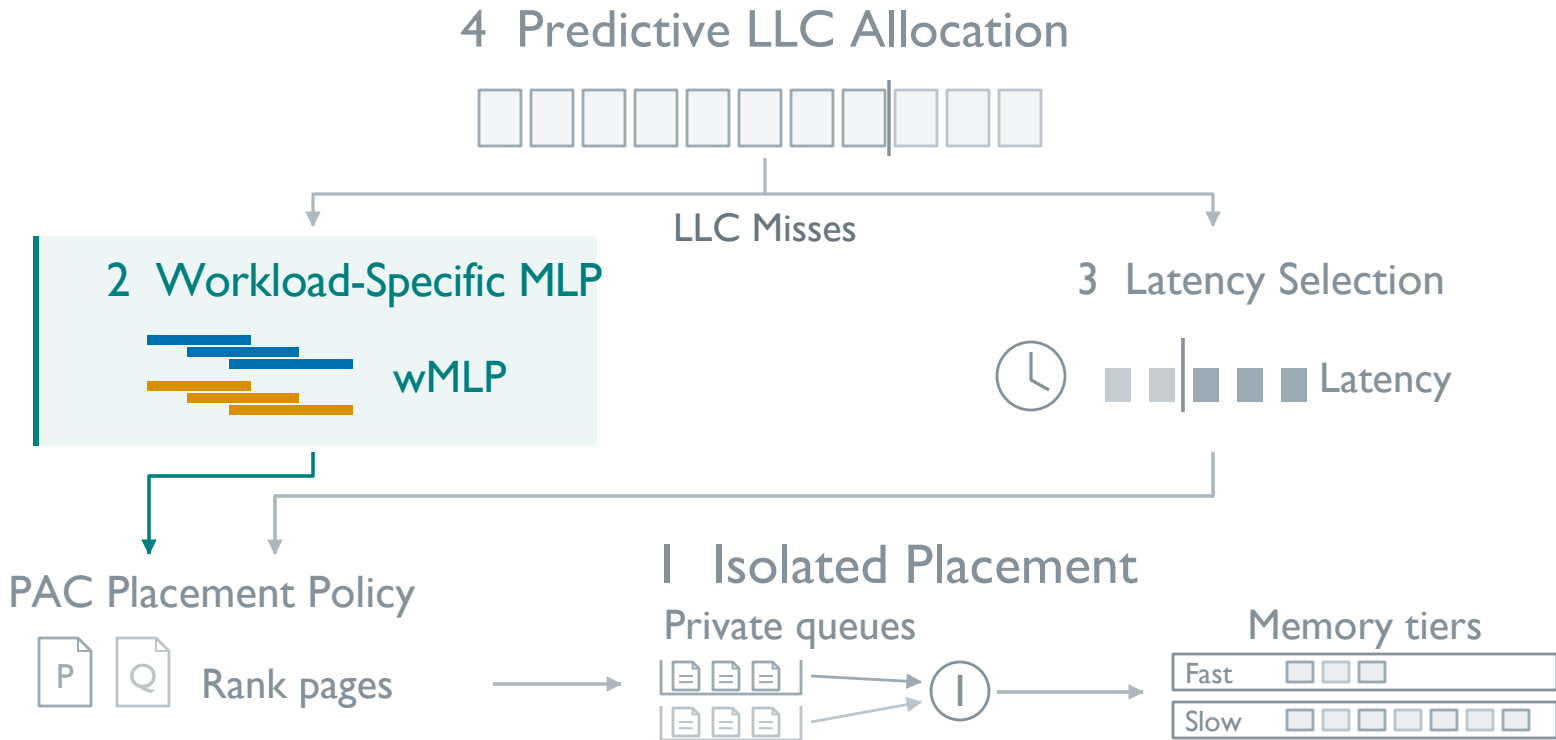
BG fills DRAM; foreground pages have nowhere to go.

Own DRAM share



Each workload can move pages within its own memory limits.

2. Measure the Workload's Own Parallelism



From Aggregate to Workload-Specific MLP

Slow-tier requests in flight

Foreground



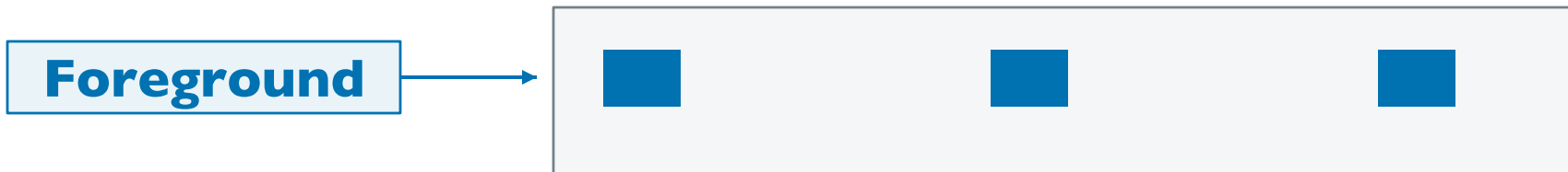
From Aggregate to Workload-Specific MLP

Slow-tier requests in flight



From Aggregate to Workload-Specific MLP

Slow-tier requests in flight



Foreground page score

$$\text{FG score} \propto \frac{\text{Accesses} \times \text{Cost}}{\text{MLP}(\text{FG})}$$

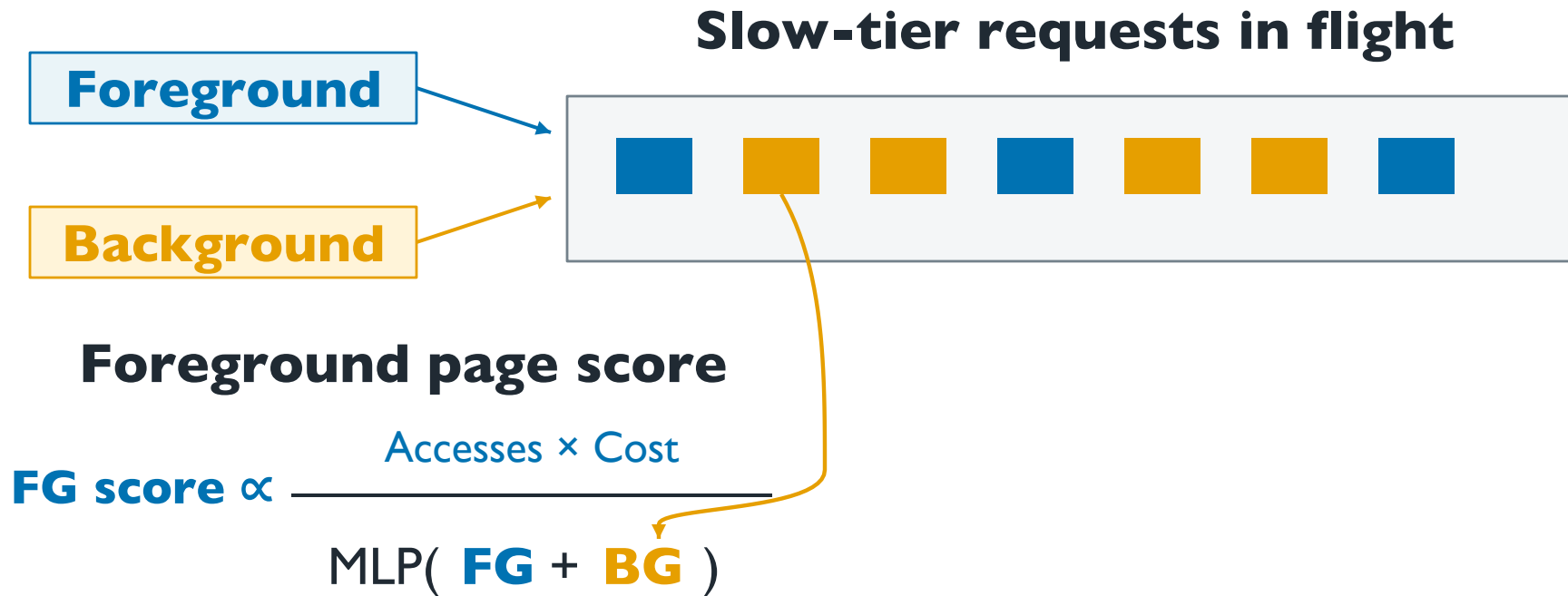
From Aggregate to Workload-Specific MLP



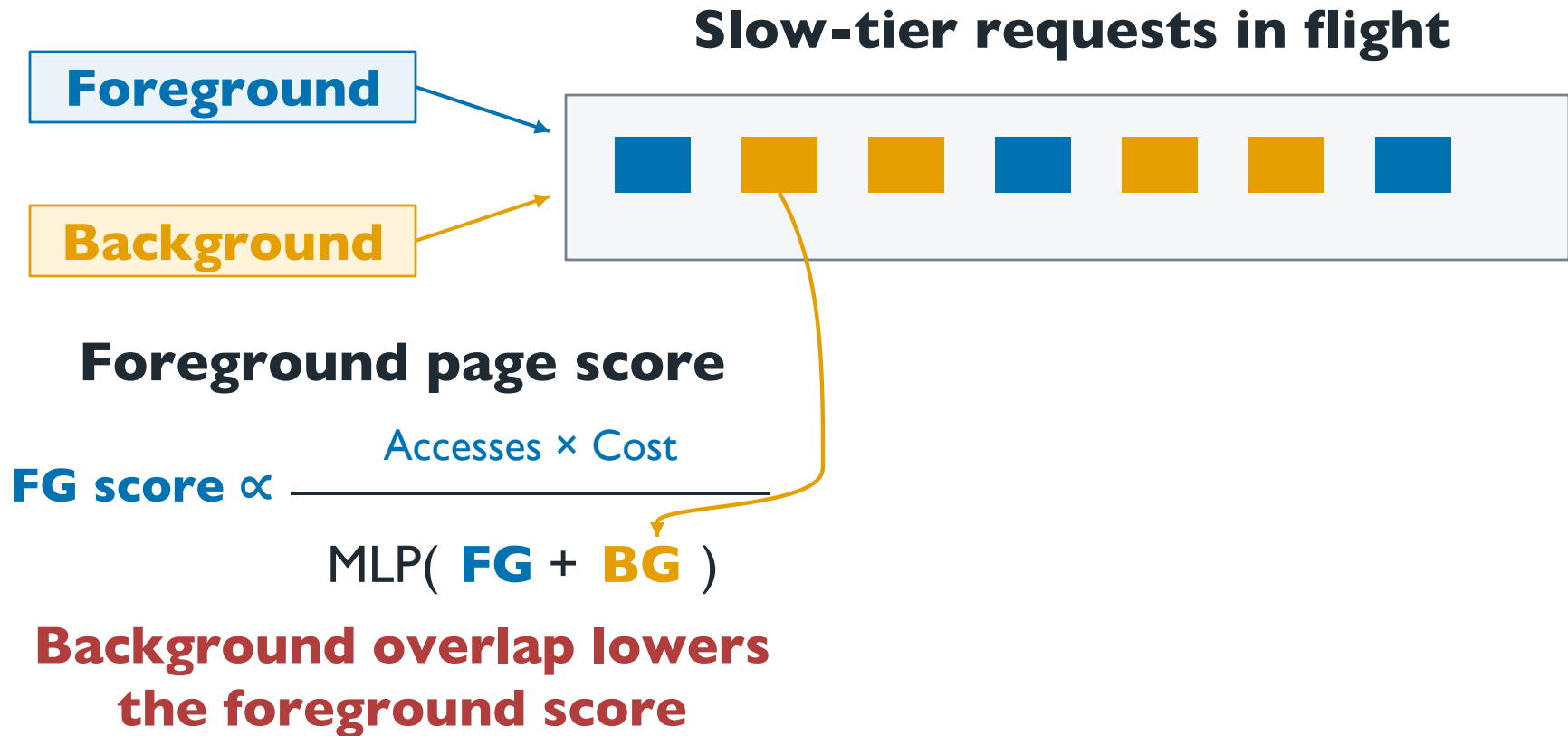
Foreground page score

$$\text{FG score} \propto \frac{\text{Accesses} \times \text{Cost}}{\text{MLP}(\text{FG})}$$

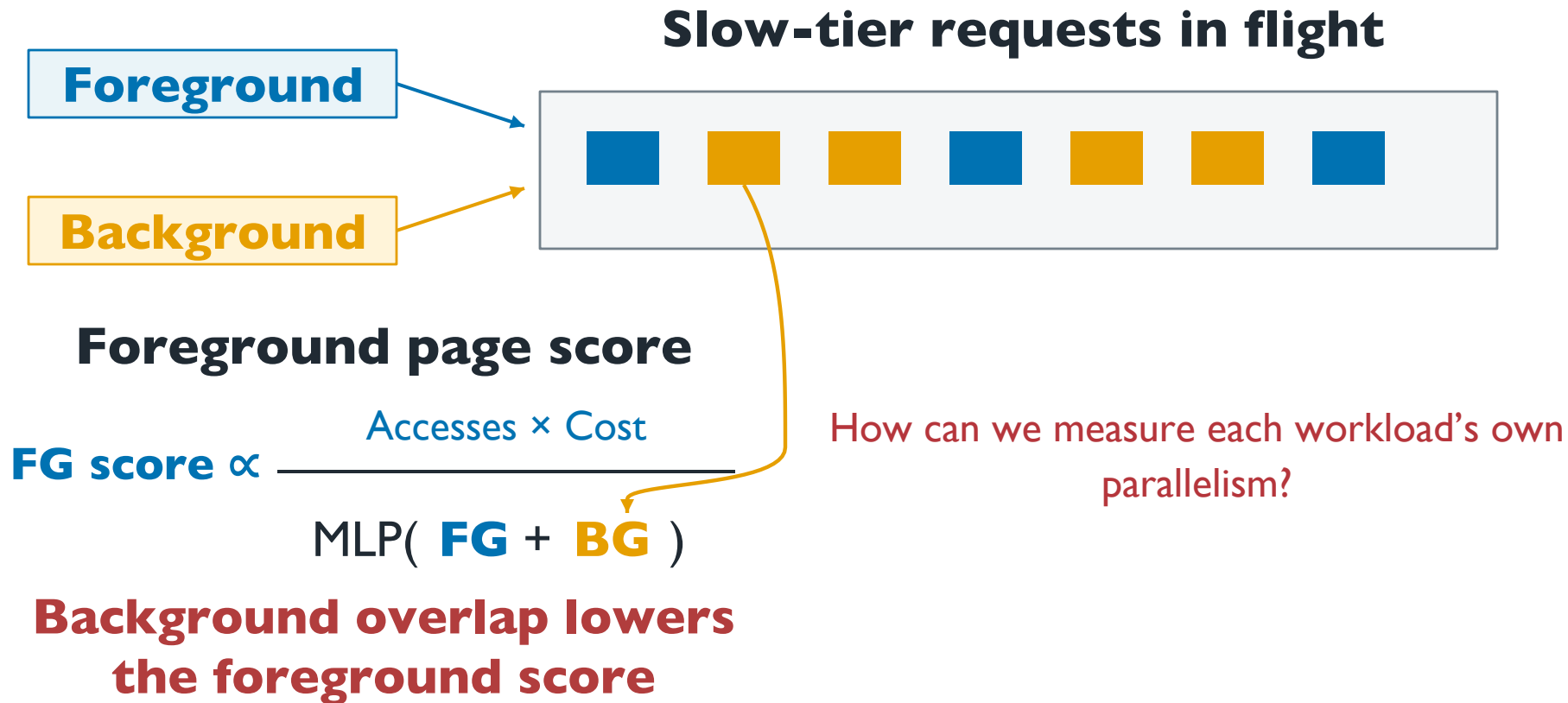
From Aggregate to Workload-Specific MLP



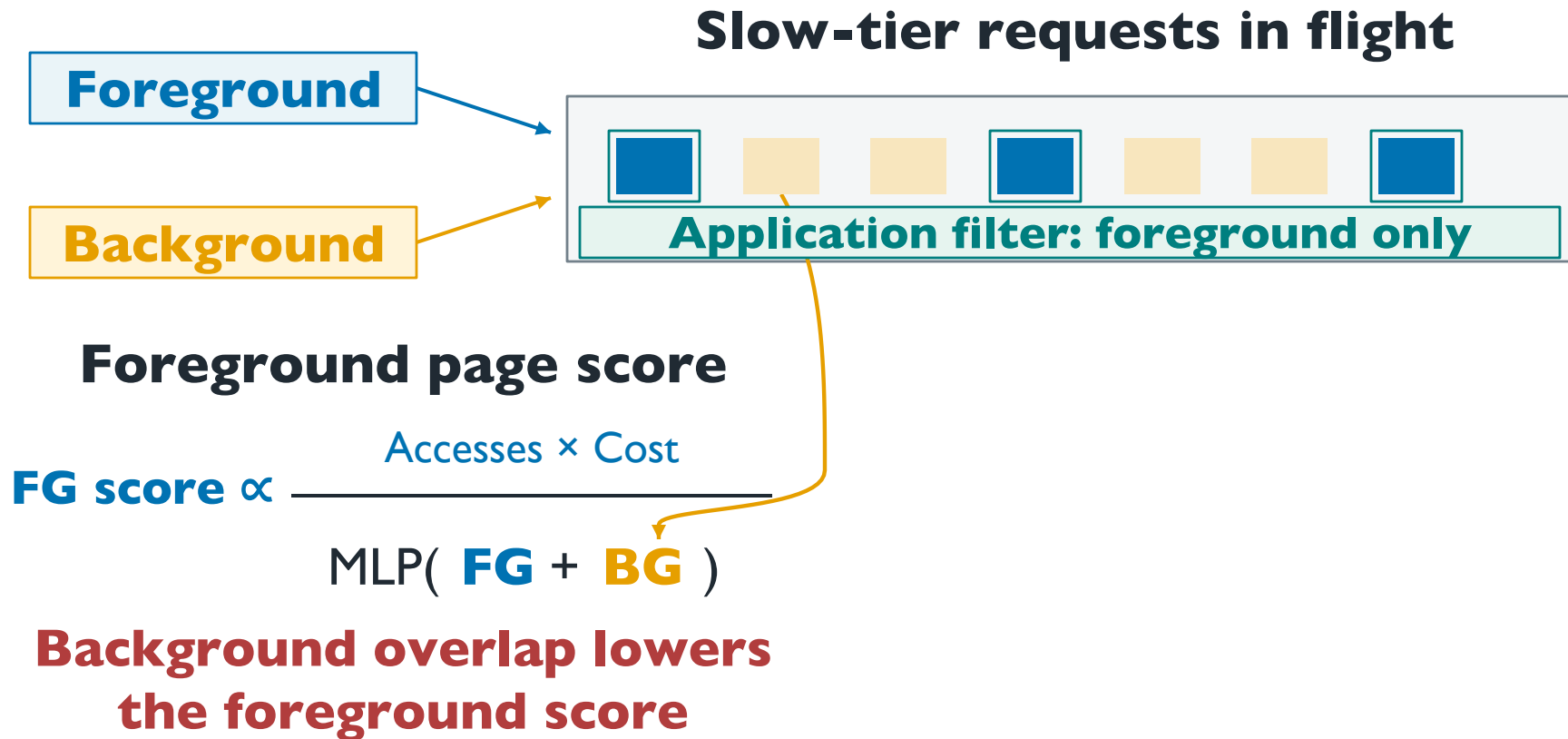
From Aggregate to Workload-Specific MLP



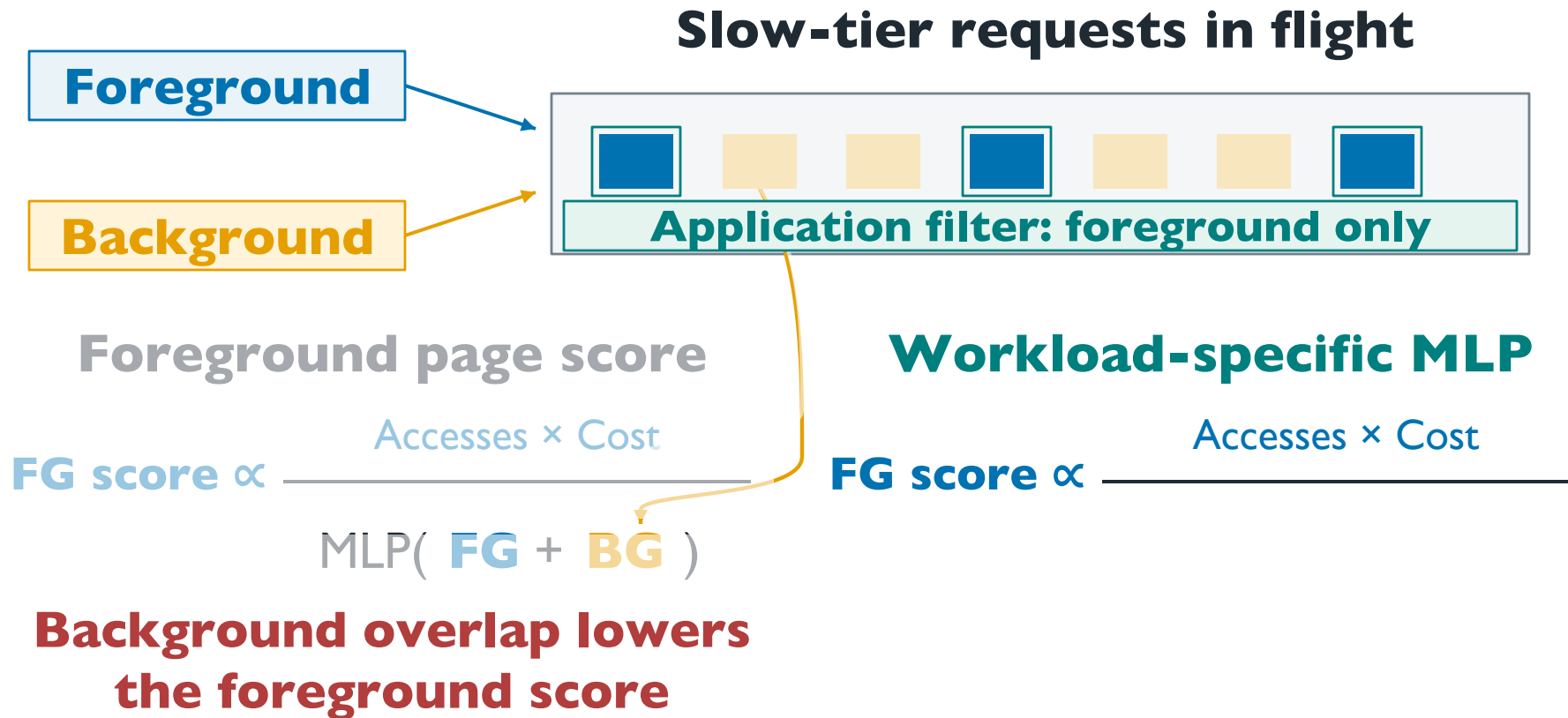
From Aggregate to Workload-Specific MLP



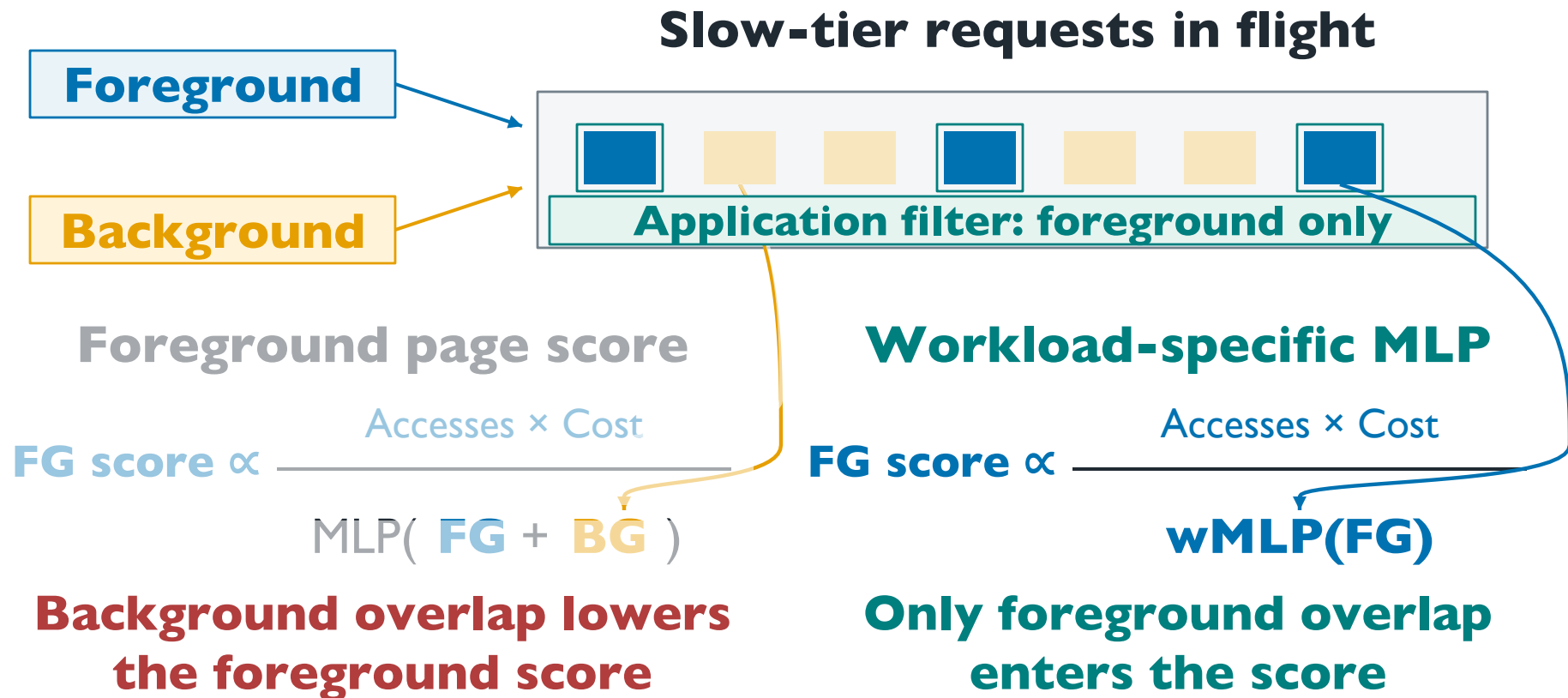
From Aggregate to Workload-Specific MLP



From Aggregate to Workload-Specific MLP



From Aggregate to Workload-Specific MLP



Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

Tierce

Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

Tierce

Destination bits



Slow-tier requests from foreground + background

Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

Tierce

Destination bits



Slow-tier requests from foreground + background

Source bits



Select one core's slow-tier requests

Which hardware source ID belongs to each core?

The mapping is under-documented.

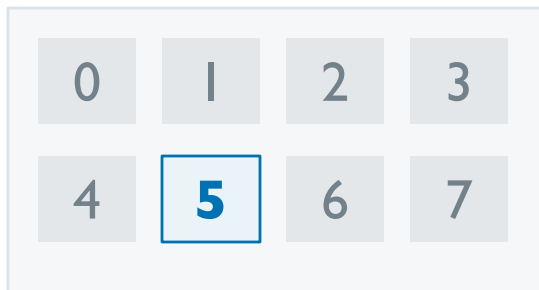
Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

Tierce

CPU cores



Foreground core

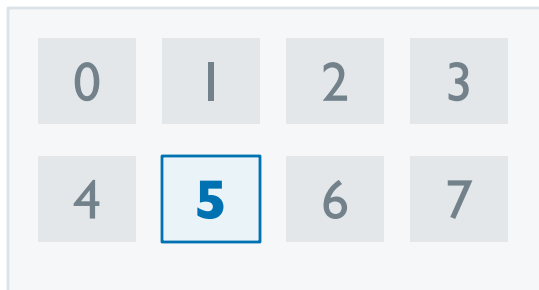
Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

Tierce

CPU cores



Foreground core

Source filter



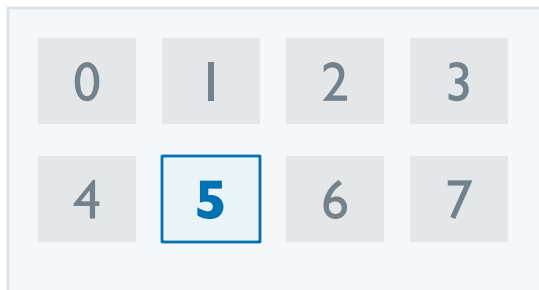
Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

Tierce

CPU cores



Foreground core

Source filter



0x28

0x20

0x08



Requests the filter counts

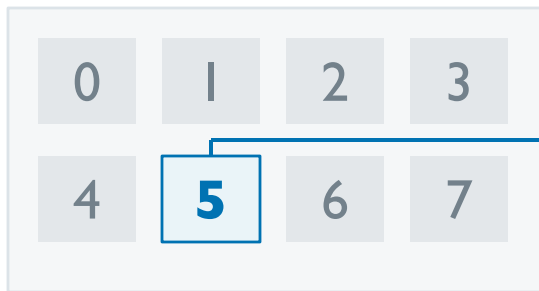
Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

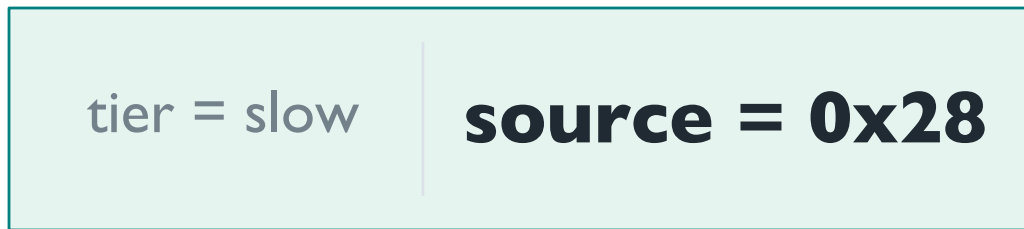
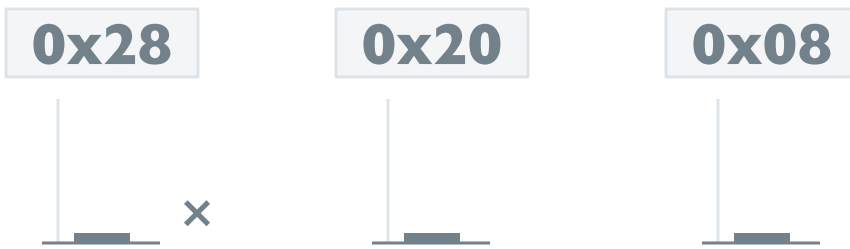
Tierce

CPU cores



Foreground core

Source filter



Requests the filter counts

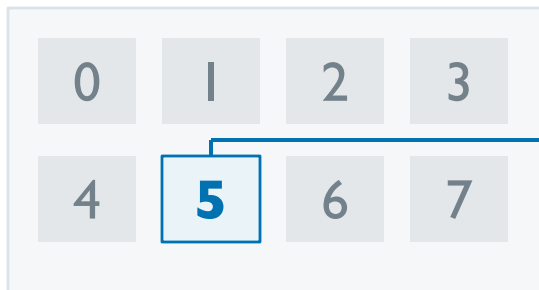
Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

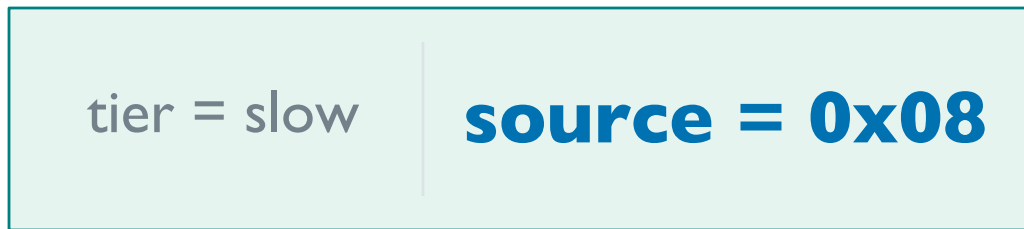
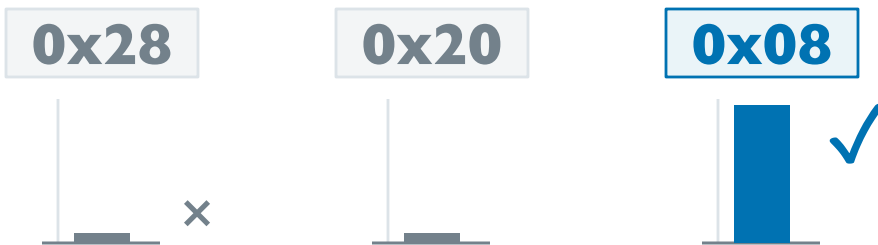
Tierce

CPU cores



Foreground core

Source filter



Requests the filter counts

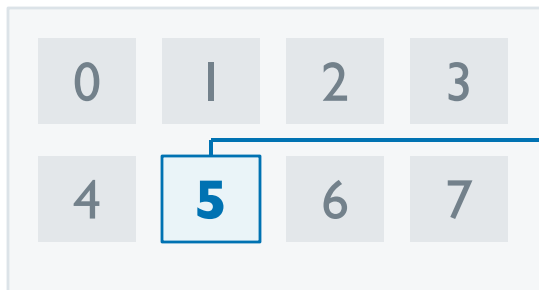
Identifying the Foreground's Requests

Per-tier MLP → **Per-core MLP** → **Workload MLP**

PACT (ASPLOS '26)

Tierce

CPU cores



Foreground core

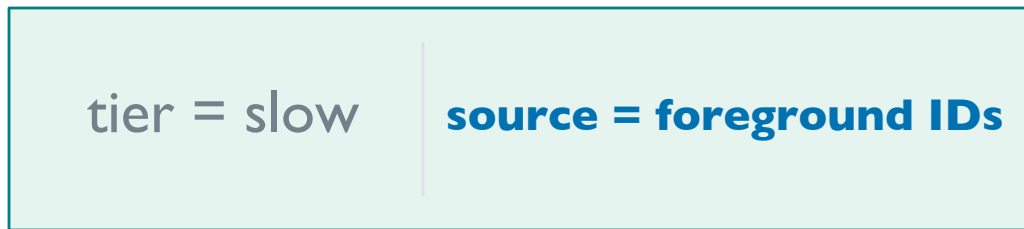
Once per platform

Core 5 → 0x08

...

...

Source filter



0x28

0x20

0x08

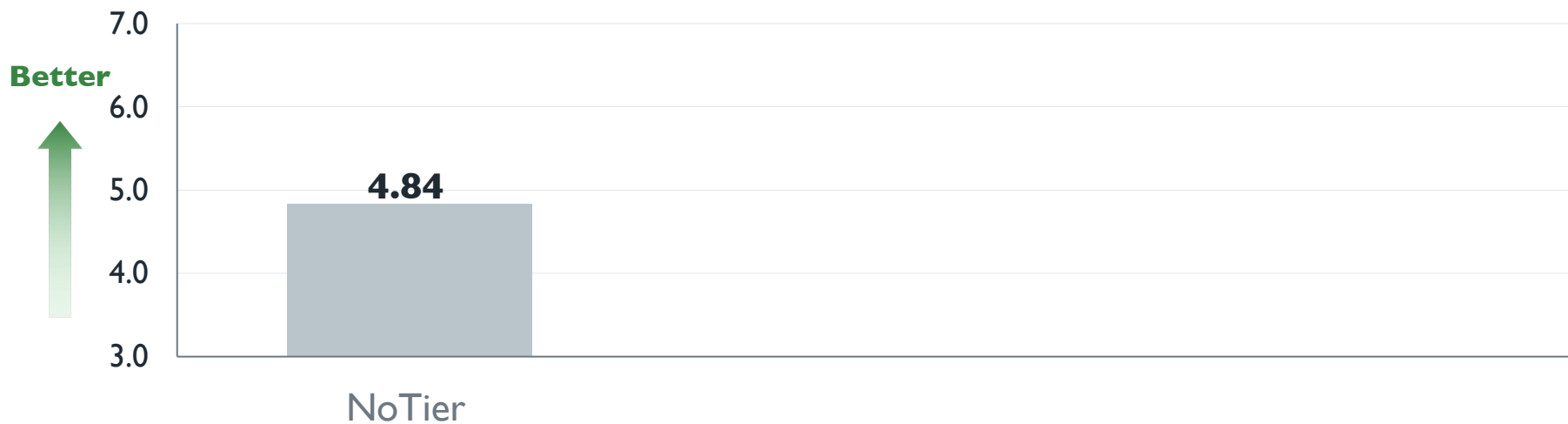


Requests the filter counts

Per-core measurements turn per-tier MLP into workload-specific MLP.

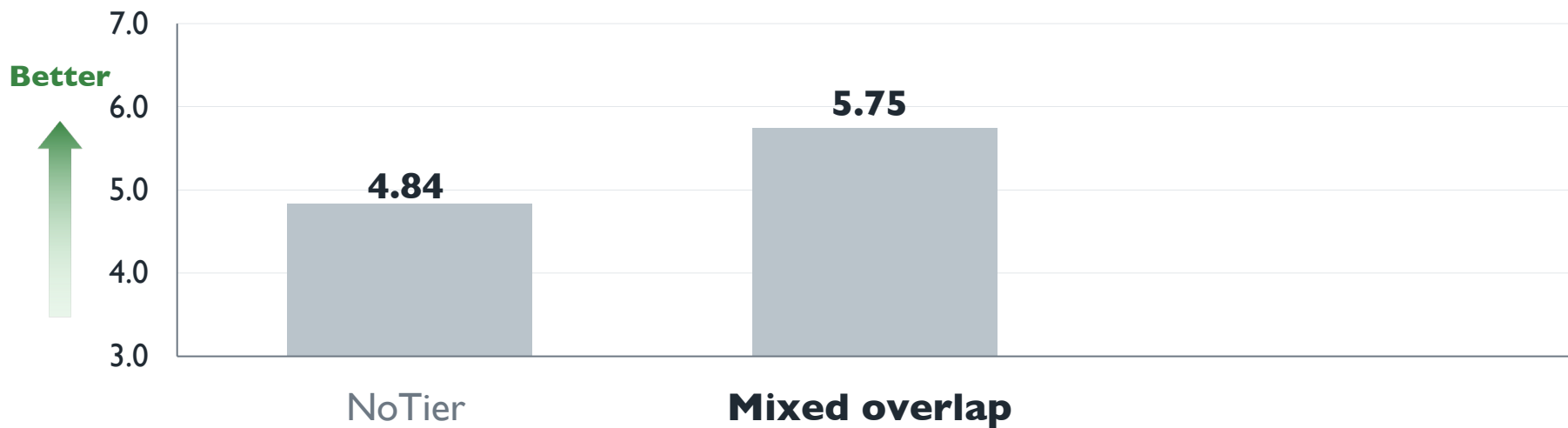
Pointer-chasing foreground + streaming background

Throughput

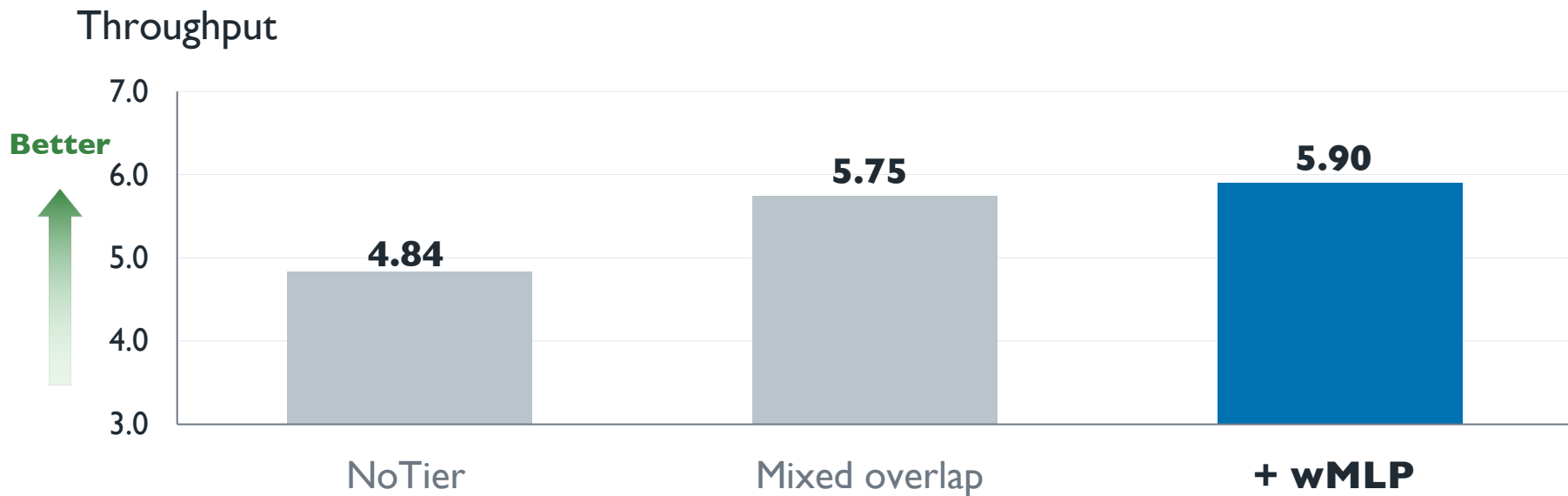


Pointer-chasing foreground + streaming background

Throughput

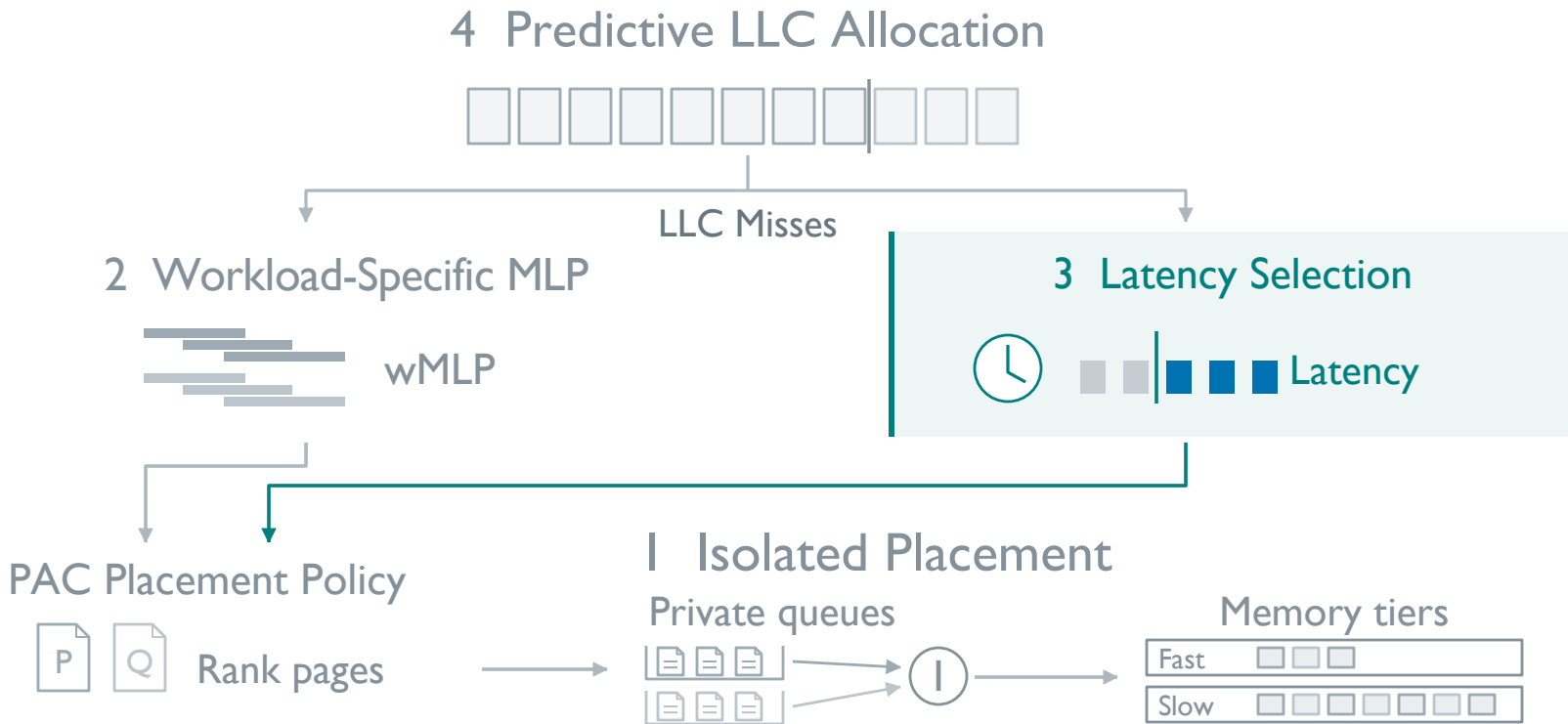


Pointer-chasing foreground + streaming background



wMLP finds critical pages left behind by mixed overlap.

3. Focus on Costly Accesses



Equal Frequency, Different Access Latency

A: nearby nodes within a page



Page 1



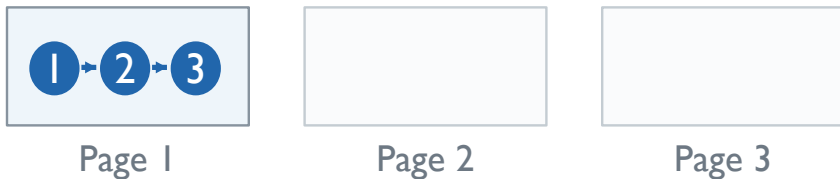
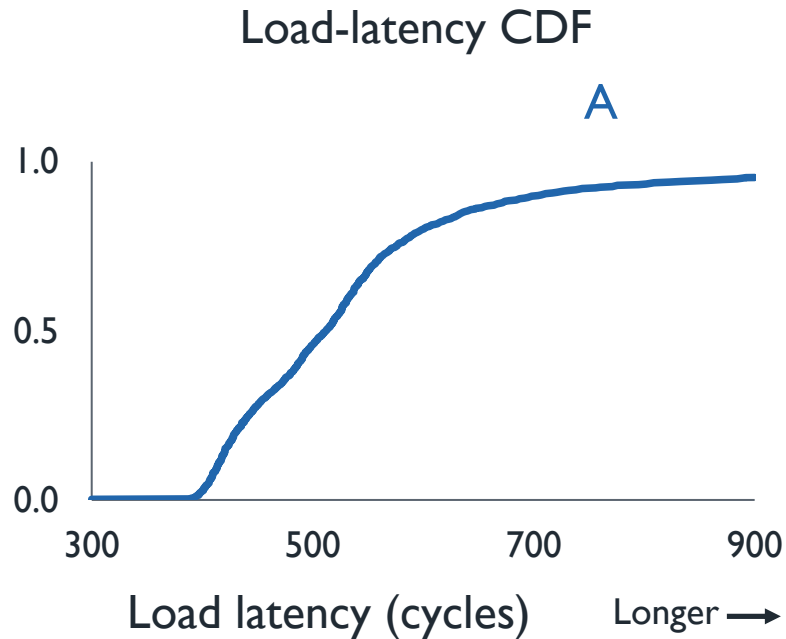
Page 2



Page 3

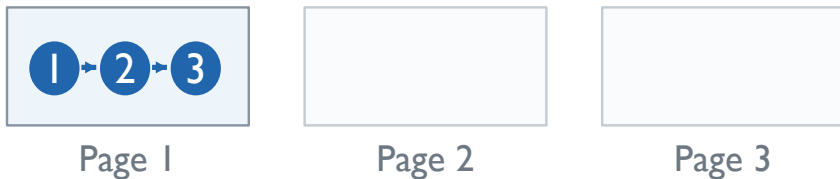
Equal Frequency, Different Access Latency

A: nearby nodes within a page

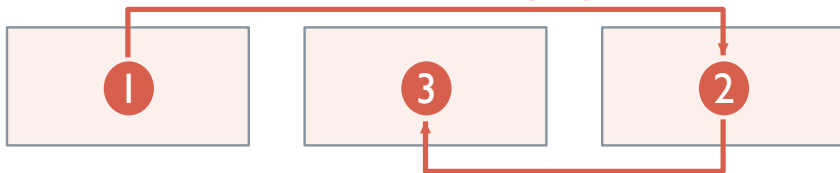


Equal Frequency, Different Access Latency

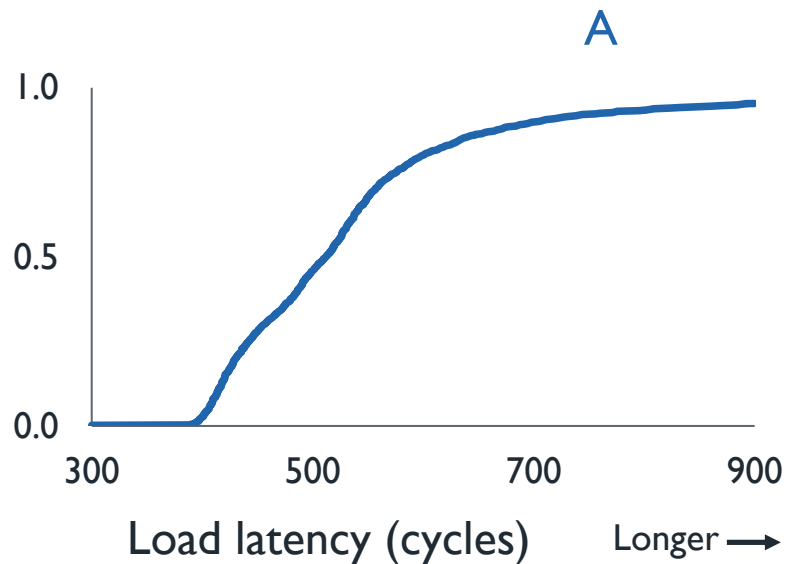
A: nearby nodes within a page



B: shuffled nodes across pages

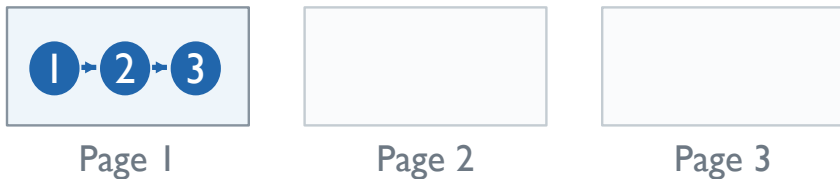


Load-latency CDF

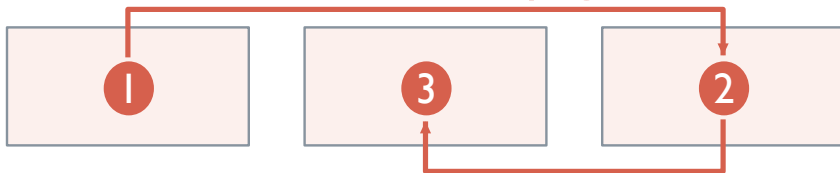


Equal Frequency, Different Access Latency

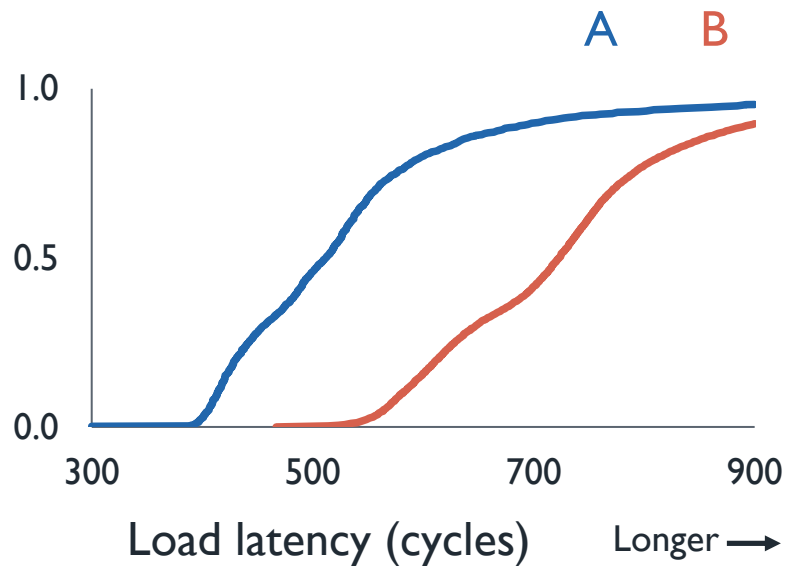
A: nearby nodes within a page



B: shuffled nodes across pages

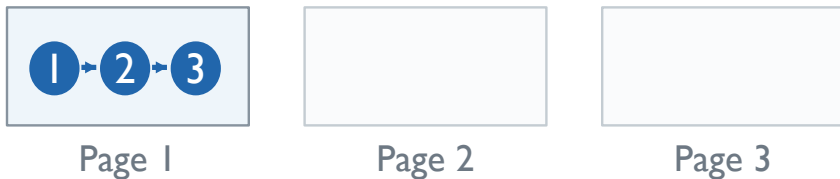


Load-latency CDF

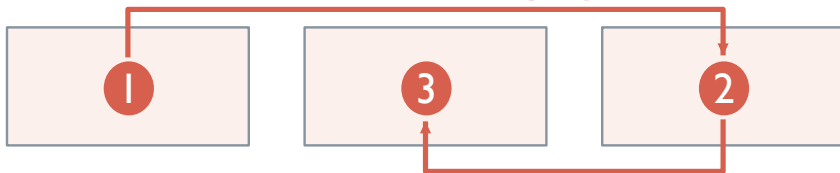


Equal Frequency, Different Access Latency

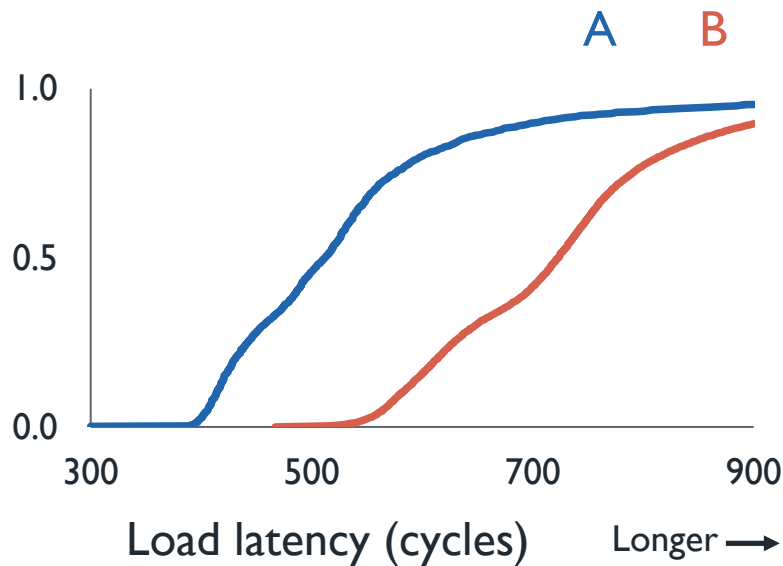
A: nearby nodes within a page



B: shuffled nodes across pages



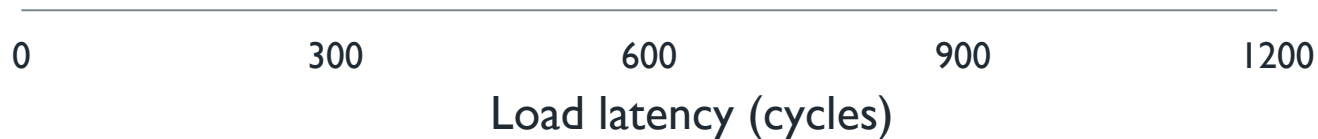
Load-latency CDF



Some accesses cost more, even when they occur equally often.

Characterizing the Latency Floor

LDLAT samples all loads, filtered by latency rather than memory tier

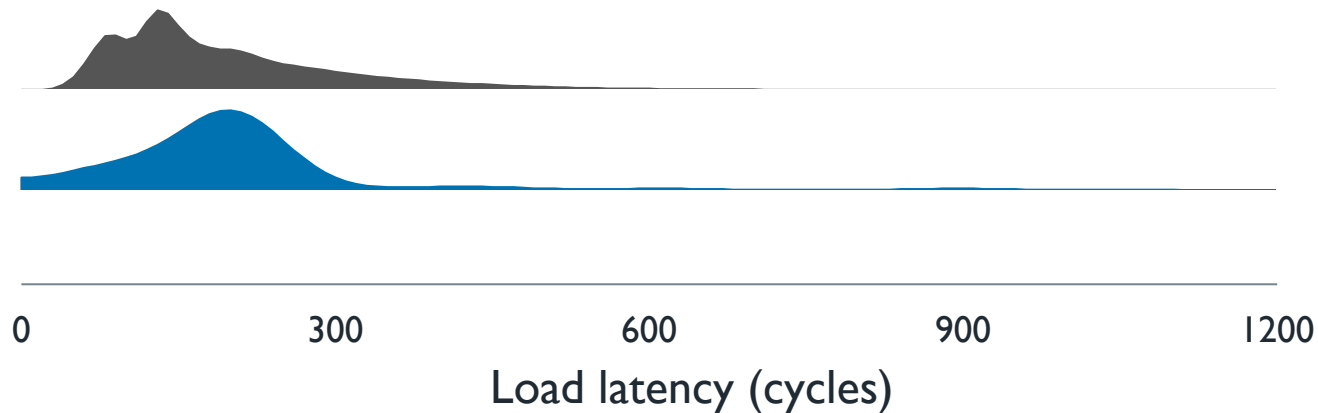


Characterizing the Latency Floor

LDLAT samples all loads, filtered by latency rather than memory tier

LLC hits

Local DRAM



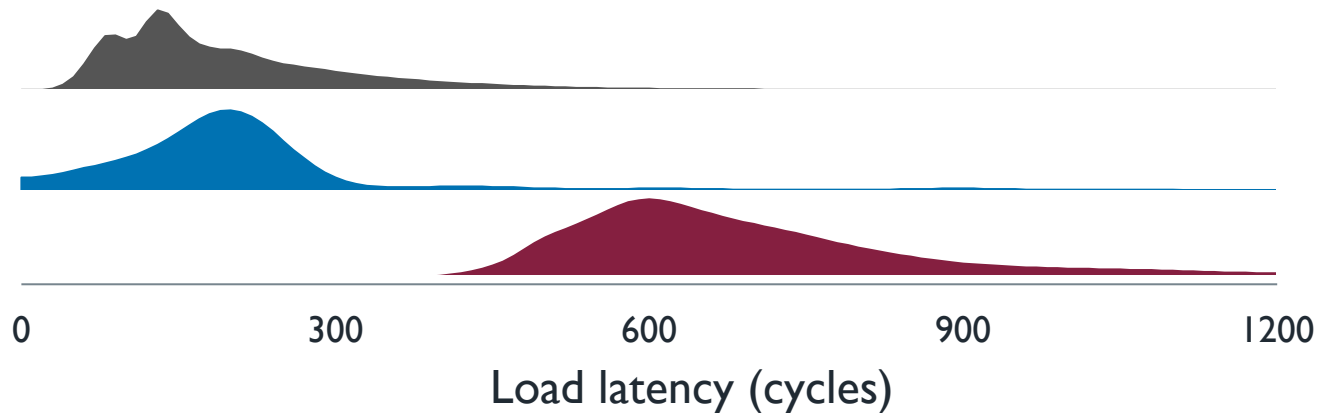
Characterizing the Latency Floor

LDLAT samples all loads, filtered by latency rather than memory tier

LLC hits

Local DRAM

Slow tier



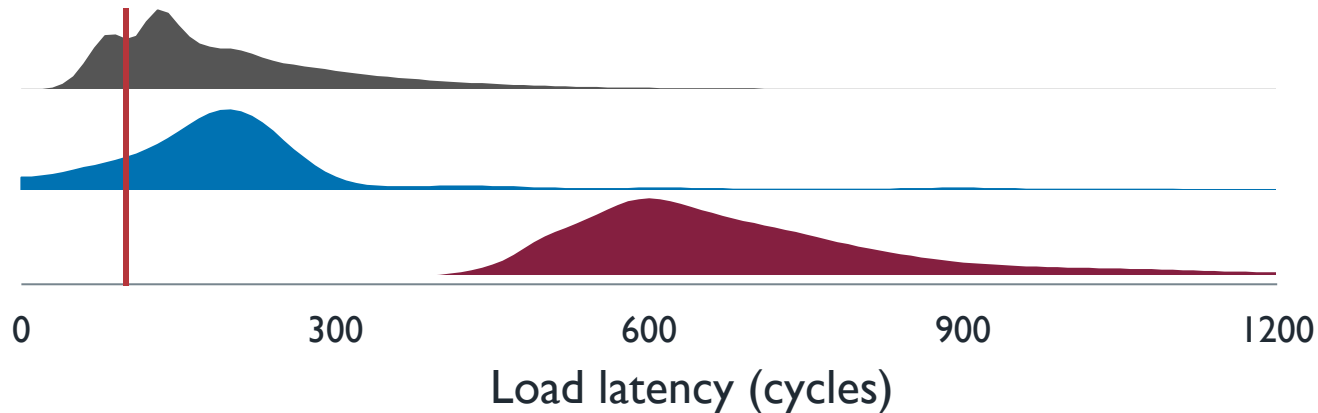
Characterizing the Latency Floor

LDLAT samples all loads, filtered by latency rather than memory tier

LLC hits

Local DRAM

Slow tier



Lower floor
More overhead

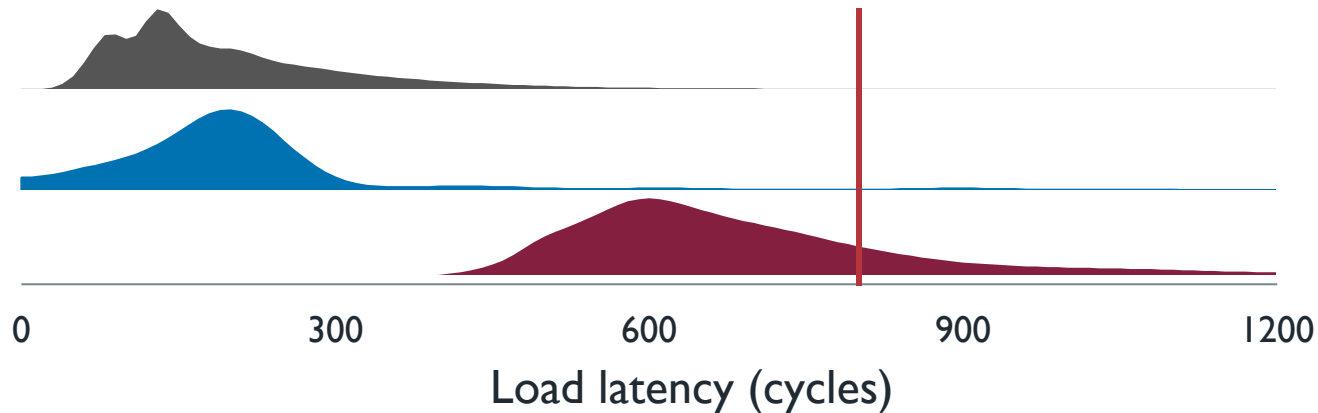
Characterizing the Latency Floor

LDLAT samples all loads, filtered by latency rather than memory tier

LLC hits

Local DRAM

Slow tier



Lower floor

More overhead

Higher floor

Fewer useful samples

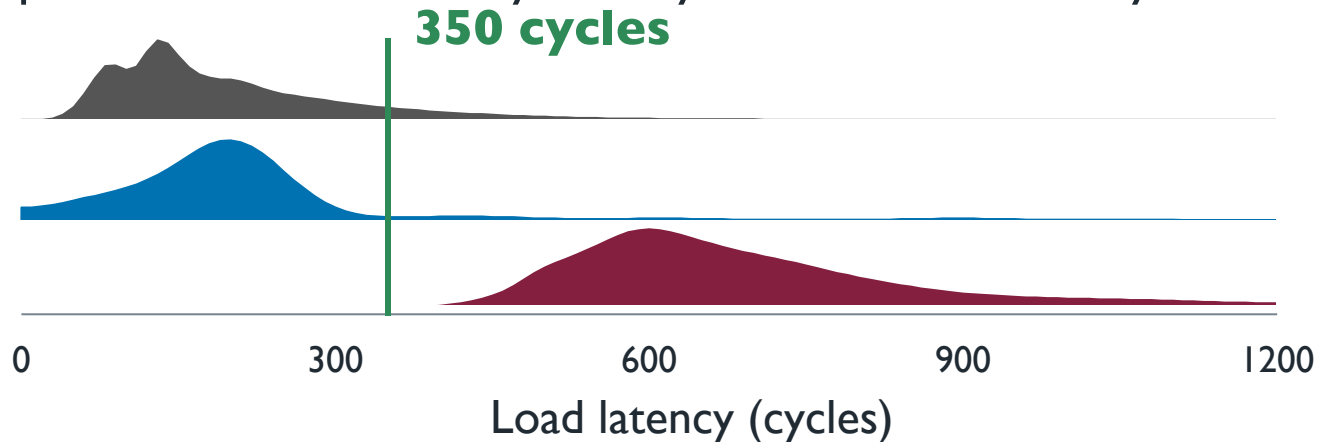
Characterizing the Latency Floor

LDLAT samples all loads, filtered by latency rather than memory tier

LLC hits

Local DRAM

Slow tier



Lower floor

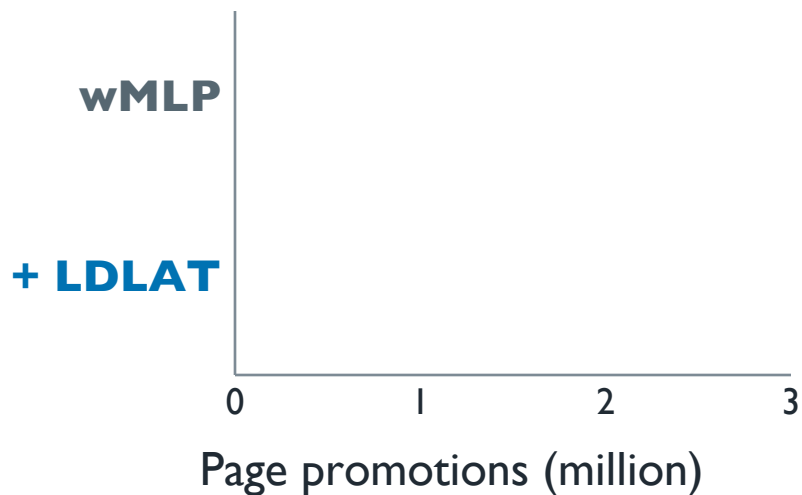
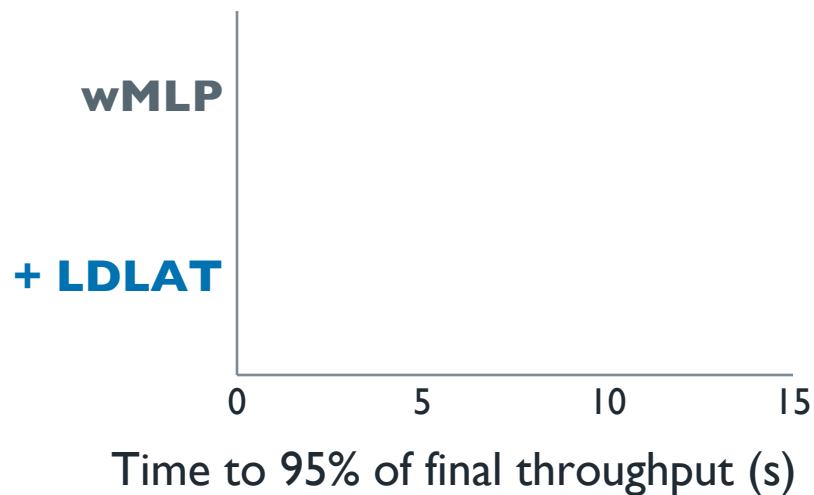
More overhead

Higher floor

Fewer useful samples

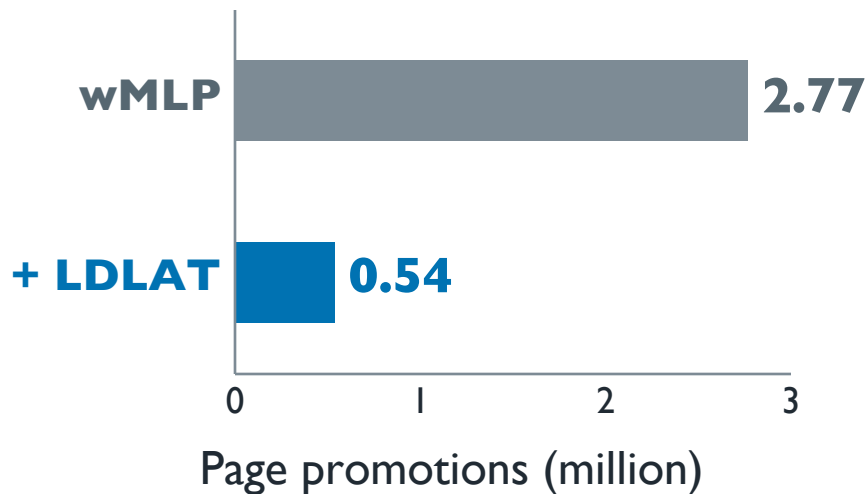
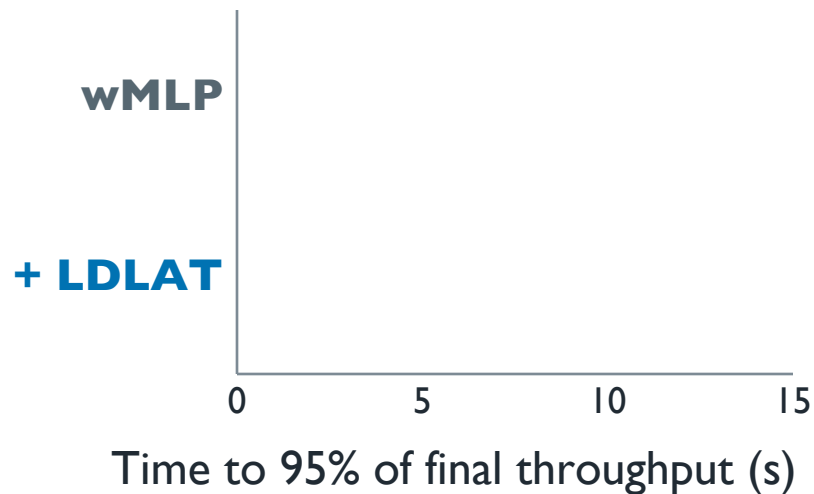
The latency floor balances recording overhead and useful samples.

How fast does placement adapt when page criticality changes?



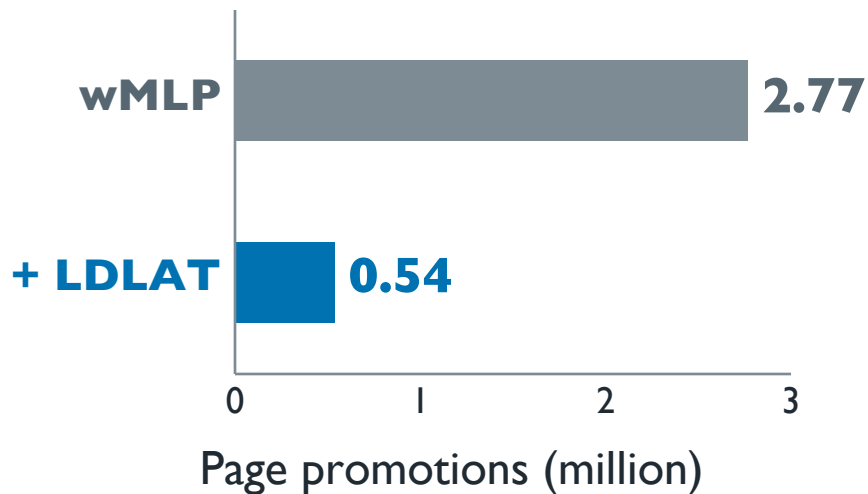
How fast does placement adapt when page criticality changes?

Fewer pages moved

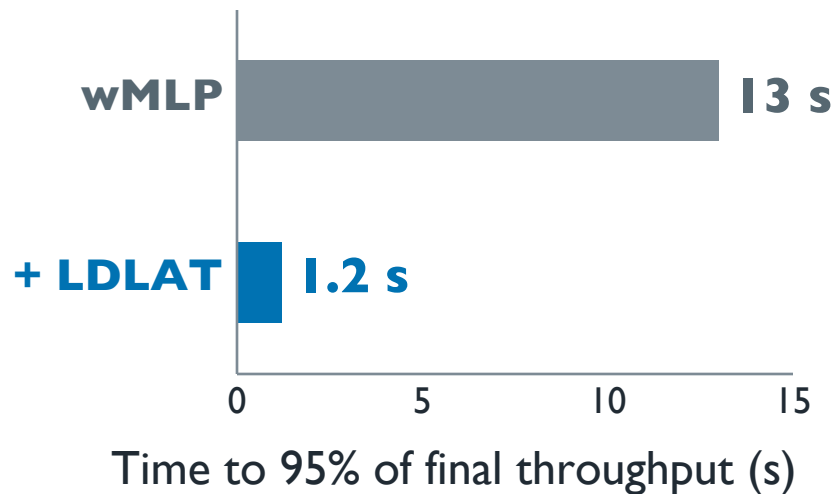


How fast does placement adapt when page criticality changes?

Fewer pages moved

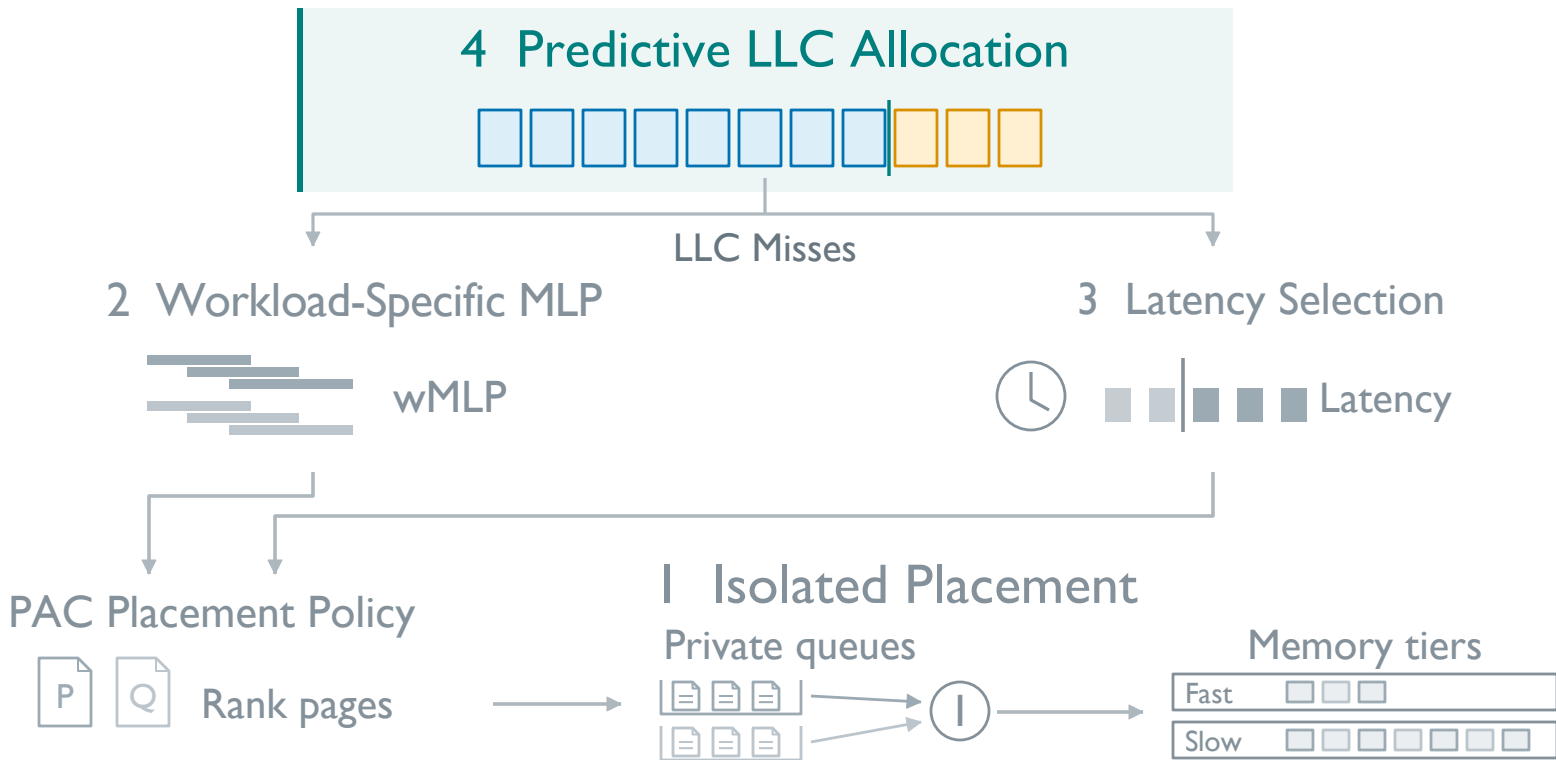


Faster recovery



Latency selection speeds recovery from 13 to 1.2 s with $\sim 5\times$ fewer promotions.

4. Reduce LLC Interference



Choosing the LLC Split

**How should we split the LLC to
preserve foreground performance while
minimizing background slowdown?**

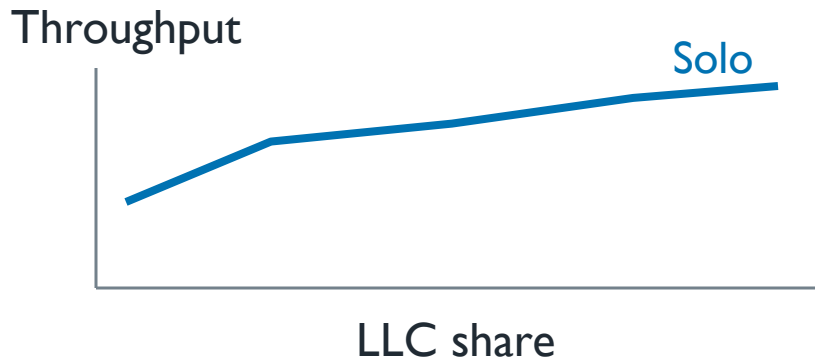
Predictive LLC Allocation

From solo LLC sensitivity to a colocated allocation

Predictive LLC Allocation

From solo LLC sensitivity to a colocated allocation

Offline characterization



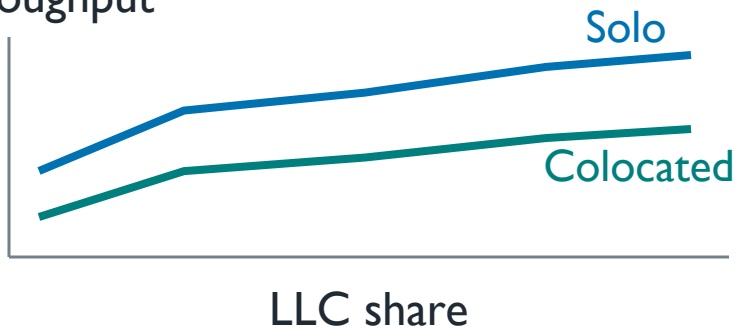
Predictive LLC Allocation

From solo LLC sensitivity to a colocated allocation

Offline characterization

53 colocated workload pairs

Throughput



Predictive LLC Allocation

From solo LLC sensitivity to a colocated allocation

Offline characterization

53 colocated workload pairs

Throughput



LLC share



Predict foreground needs

LLC



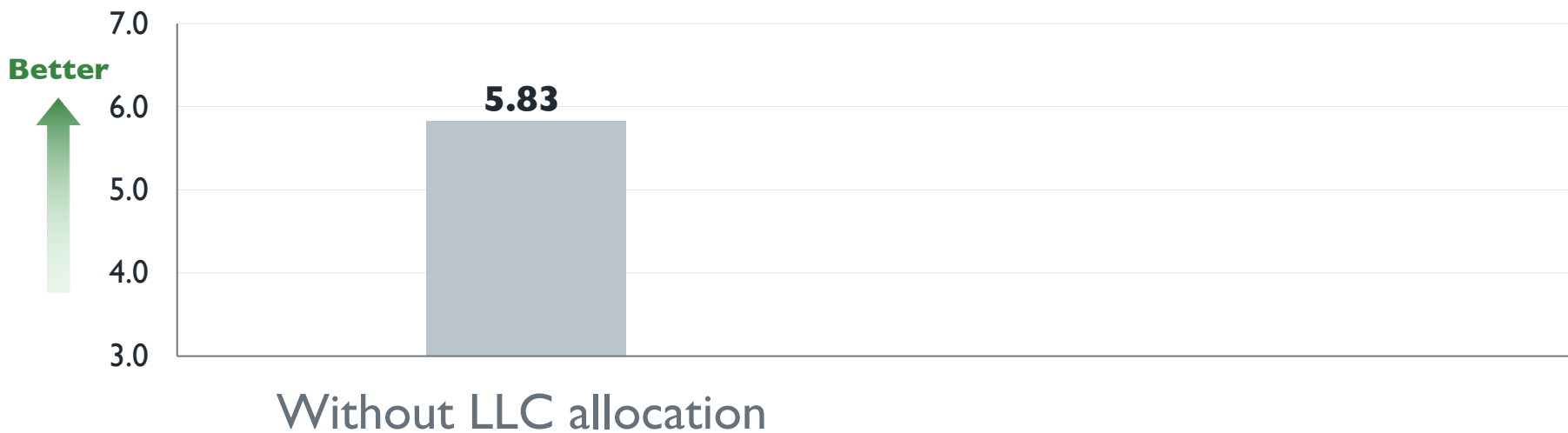
Foreground

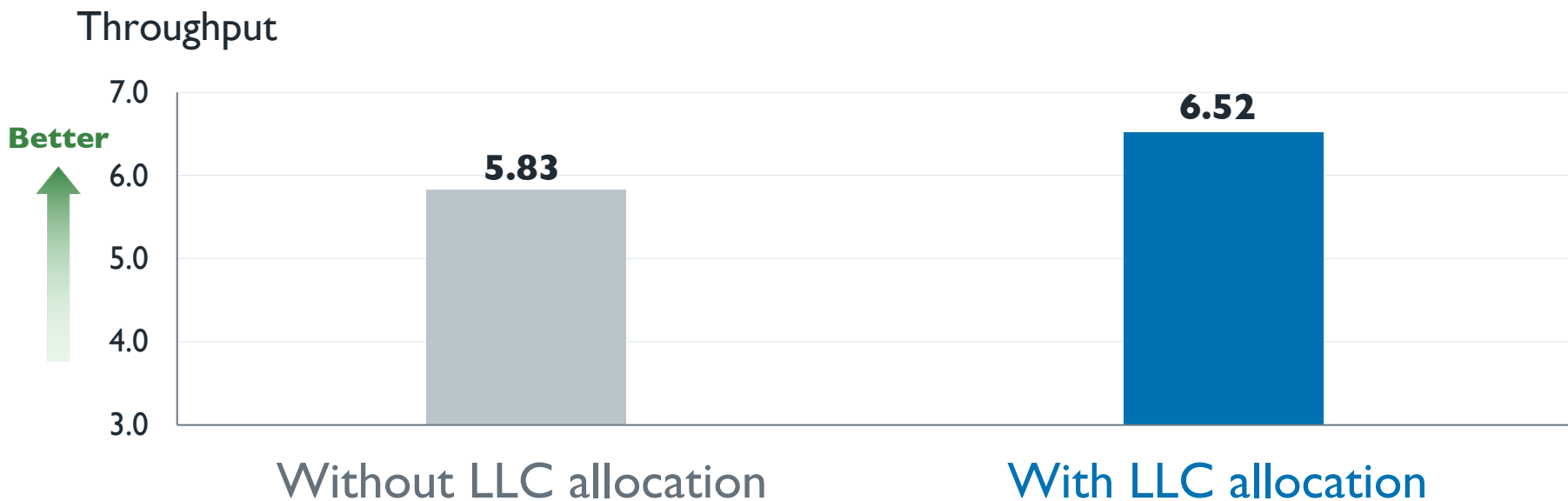
Background

Static allocation

Predict foreground LLC needs before choosing the split.

Throughput





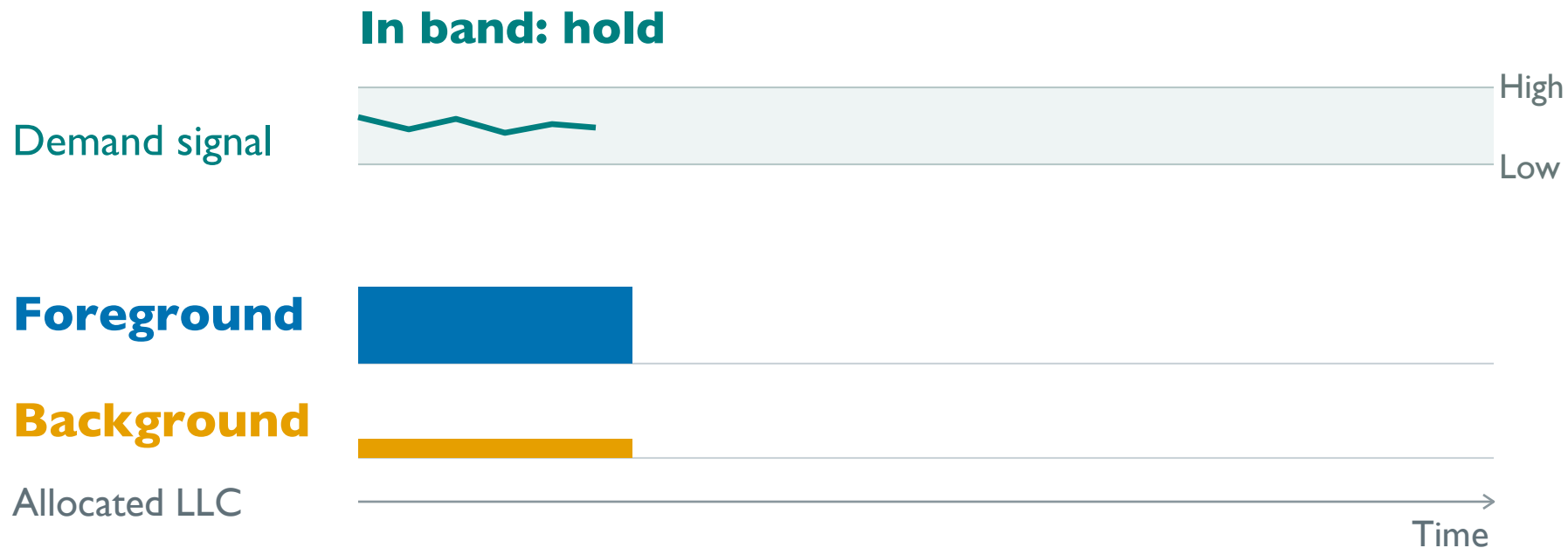
LLC allocation raises foreground throughput by 12%
while the background loses only ~3%.

Dynamic LLC Allocation

A fixed split cannot follow changing LLC needs.

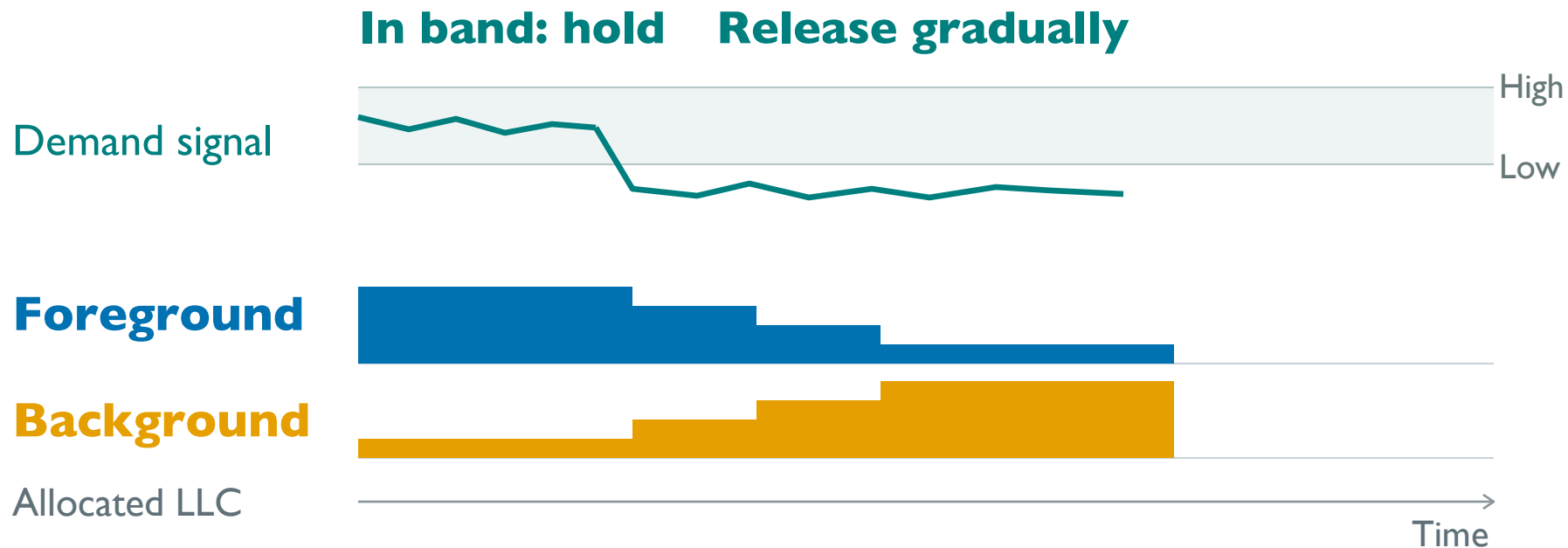
Dynamic LLC Allocation

A fixed split cannot follow changing LLC needs.



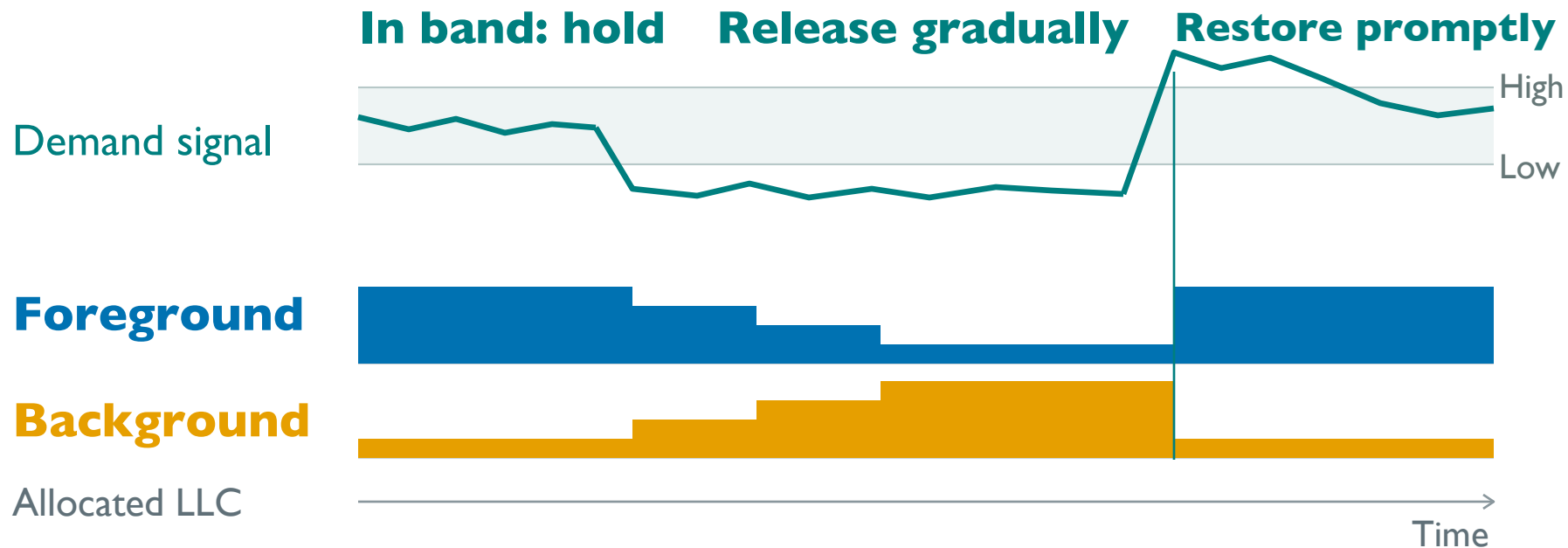
Dynamic LLC Allocation

A fixed split cannot follow changing LLC needs.



Dynamic LLC Allocation

A fixed split cannot follow changing LLC needs.



Track a demand signal and adjust the LLC split without reacting to noise.

Dynamic LLC Allocation

Same foreground, a background that benefits from more LLC space

Dynamic LLC Allocation

Same foreground, a background that benefits from more LLC space

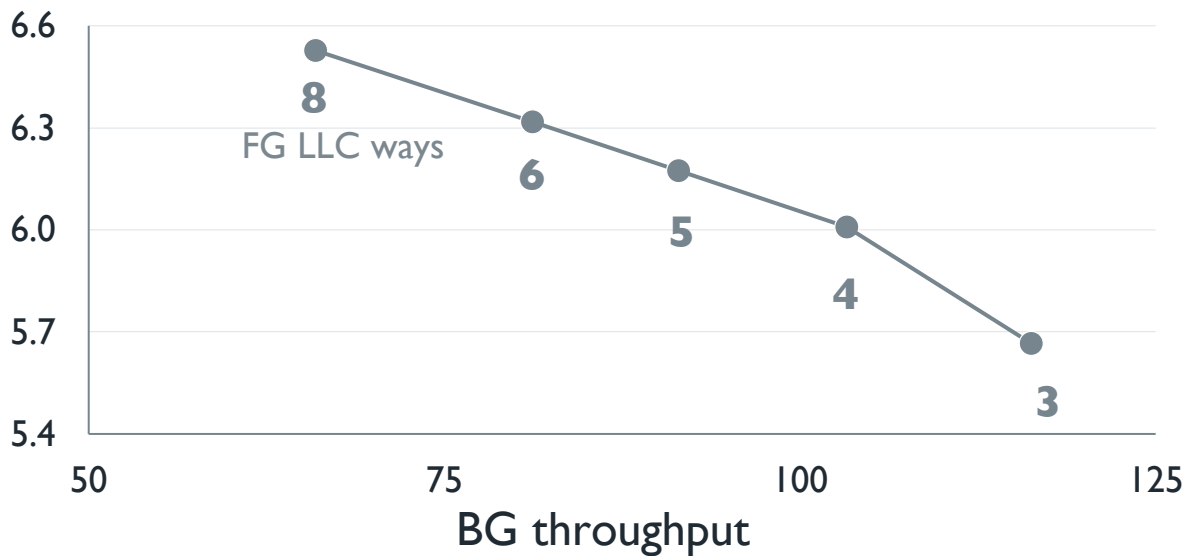
FG throughput



Dynamic LLC Allocation

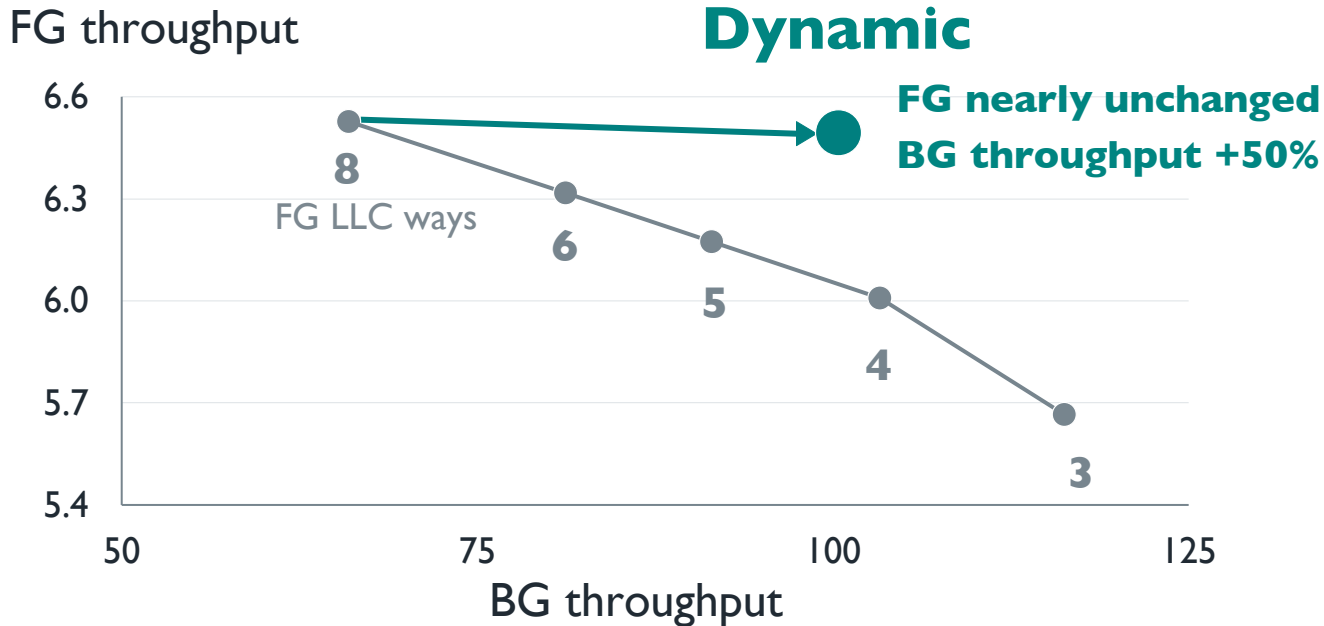
Same foreground, a background that benefits from more LLC space

FG throughput



Dynamic LLC Allocation

Same foreground, a background that benefits from more LLC space



Dynamic allocation adds 50% background throughput
with nearly unchanged foreground performance.

Talk Outline

Overview

Tierce Design

1. Isolated placement
2. Workload-specific MLP
3. Latency selection
4. Predictive LLC allocation

Evaluation

Evaluation with Colocated Applications

Baselines

Local DRAM

All pages in fast memory

NoTier

Two tiers, no migration

NBT (Linux, 2022)

Colloid (SOSP 2024)

Memtis (SOSP 2023)

PACT (ASPLOS 2026)

Workloads

Metrics

Evaluation with Colocated Applications

Baselines

Local DRAM

All pages in fast memory

NoTier

Two tiers, no migration

NBT (Linux, 2022)

Colloid (SOSP 2024)

Memtis (SOSP 2023)

PACT (ASPLOS 2026)

Workloads

Graph analytics,
transaction processing
+ streaming background

Metrics

**Foreground
Throughput**

Higher is better

Evaluation with Colocated Applications

Baselines

Local DRAM

All pages in fast memory

NoTier

Two tiers, no migration

NBT (Linux, 2022)

Colloid (SOSP 2024)

Memtis (SOSP 2023)

PACT (ASPLOS 2026)

Workloads

Graph analytics,
transaction processing
+ streaming background

ANN search
+ graph-processing
background

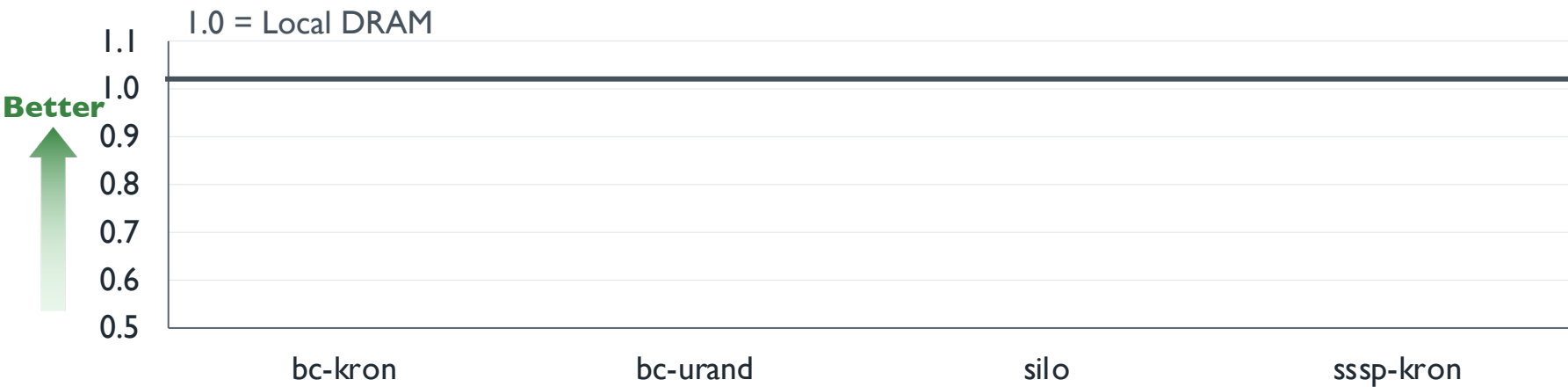
Metrics

**Foreground
Throughput**

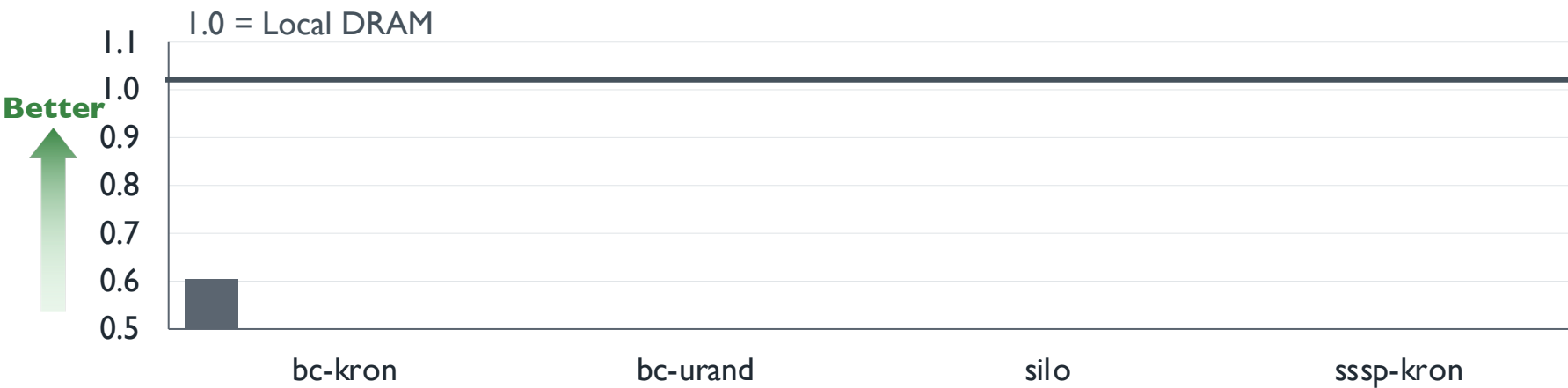
Higher is better

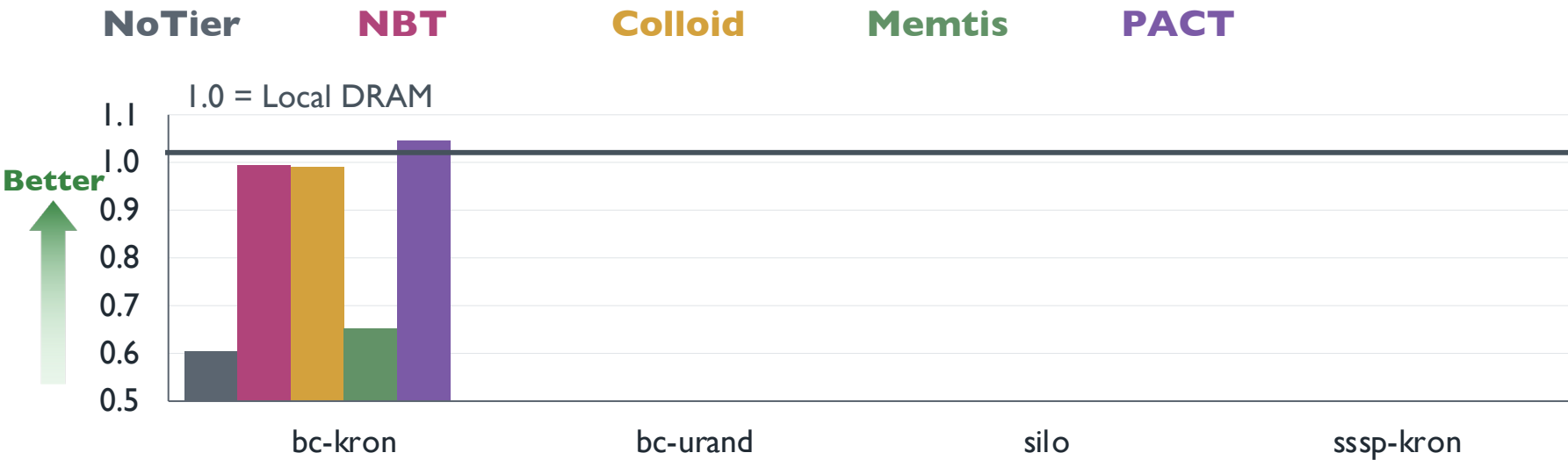
**p99 Response
Time**

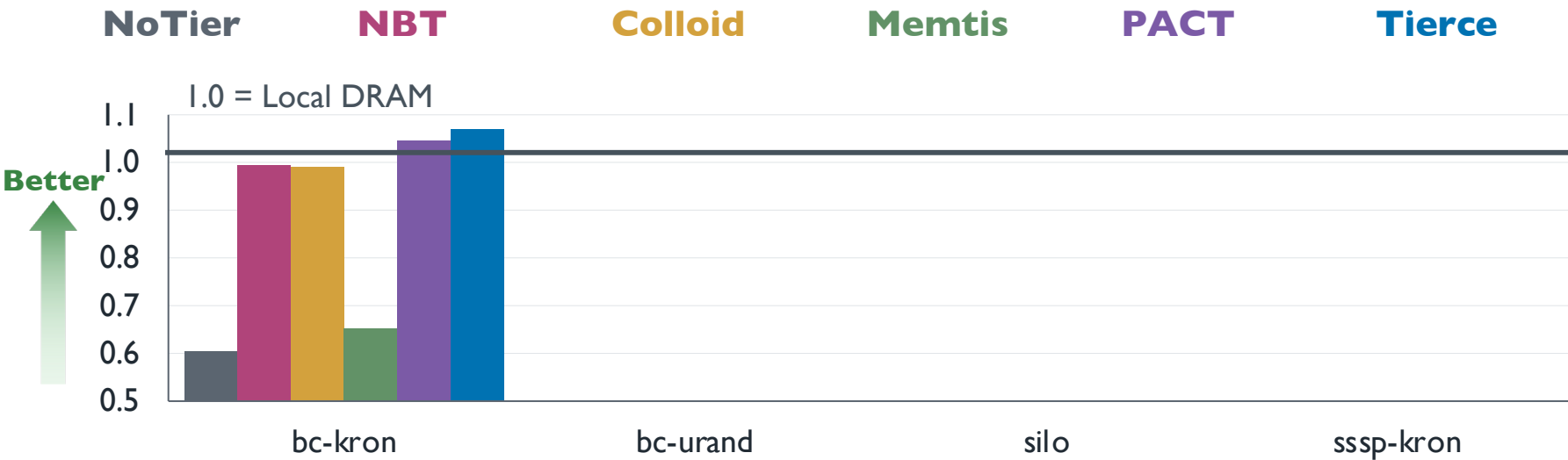
Lower is better

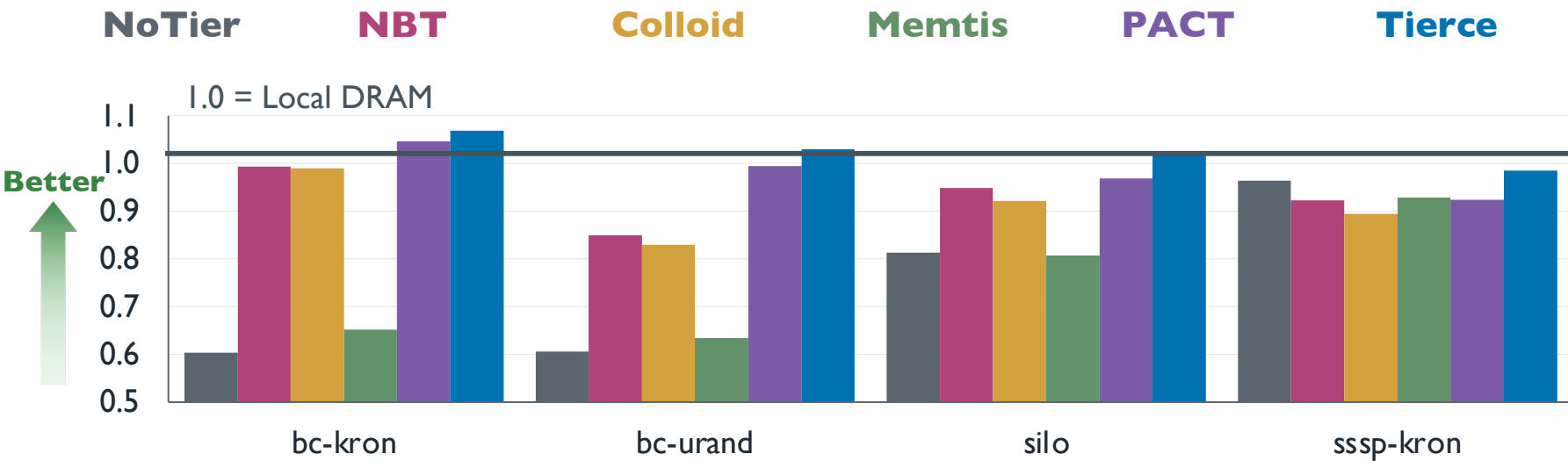


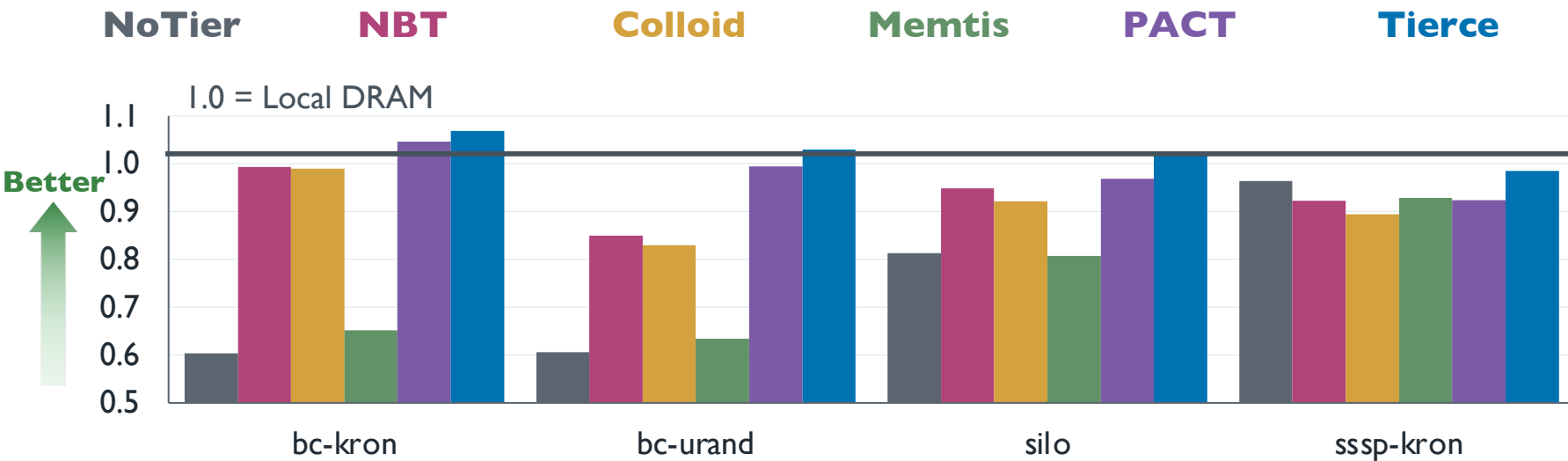
NoTier







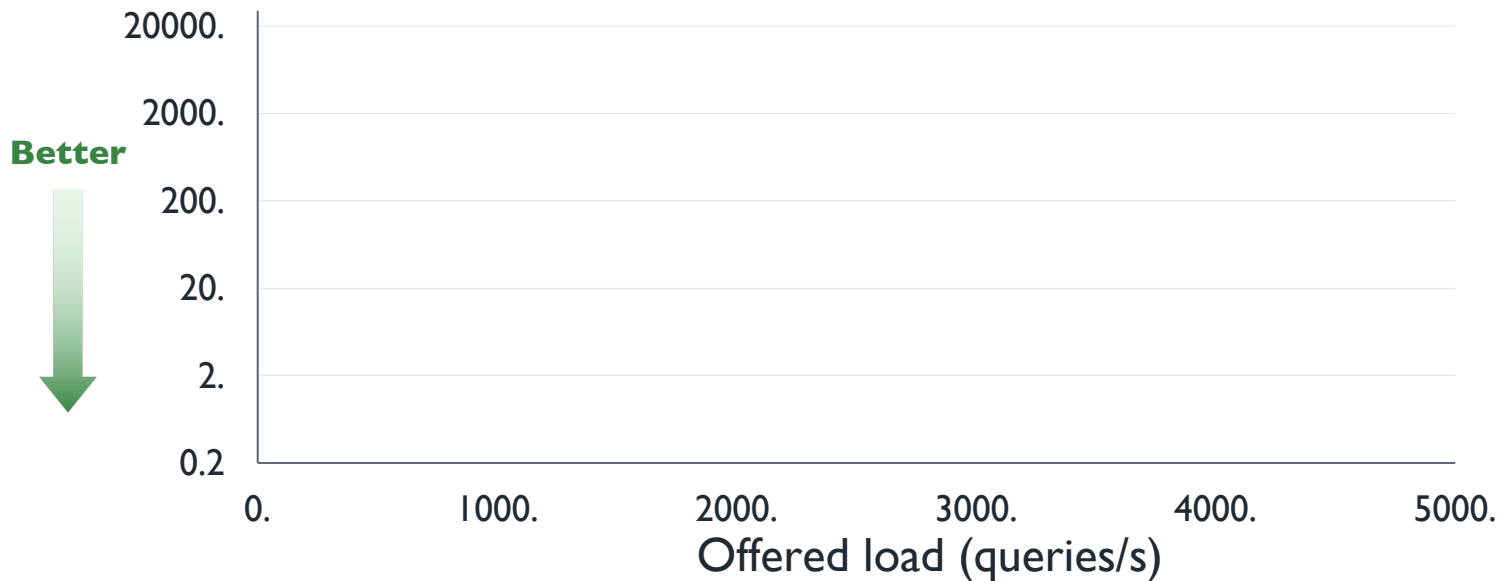




Tierce improves throughput by 2.2–12.6% over PACT, with gains up to 64% over Memtis and 25.9% over Colloid.

ANN search + bc-kron

p99 latency (ms, log scale)



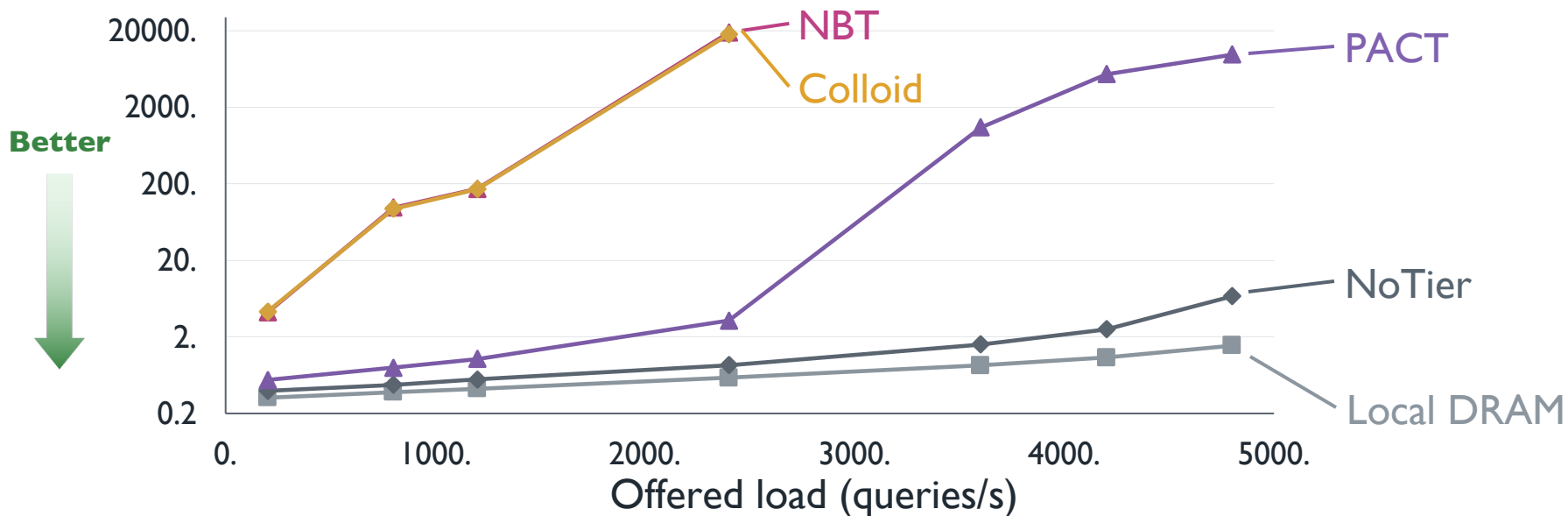
ANN search + bc-kron

p99 latency (ms, log scale)



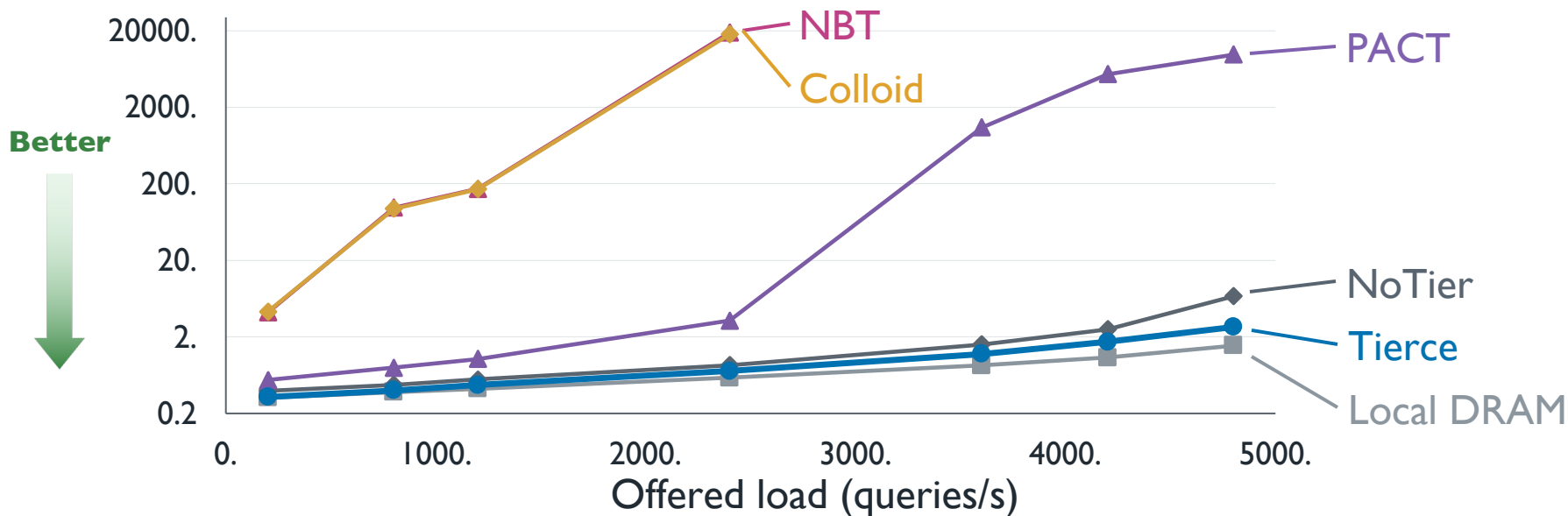
ANN search + bc-kron

p99 latency (ms, log scale)



ANN search + bc-kron

p99 latency (ms, log scale)



Tierce keeps p99 below other tiering systems as query rate rises.

Tierce Provides Observability for Colocated Tiering

Colocation gaps Tierce responses

- Shared promotions → Isolated placement
- Mixed overlap → Workload-specific MLP
- Unequal latency → Latency selection
- LLC interference → Predictive LLC allocation

Tierce Provides Observability for Colocated Tiering

Colocation gaps Tierce responses

Shared promotions → Isolated placement

Mixed overlap → Workload-specific MLP

Unequal latency → Latency selection

LLC interference → Predictive LLC allocation

More in the paper

Identifying each workload's requests
Effects of latency floors and memory limits
Predicting LLC needs
Dynamic LLC allocation
More workload pairs and evaluation results



Tierce: Observability-Driven Tiered Memory Management for Colocated Workloads

Hanchen Xu^{*}
Virginia Tech
Blacksburg, USA

Berkay Incelesci^{*}
Virginia Tech
Blacksburg, USA

Hao Li
Virginia Tech
Blacksburg, USA

Zhenyu Zhang
Virginia Tech
Blacksburg, USA

Huicheng Li
Virginia Tech
Blacksburg, USA

Abstract

Tiered memory systems rely on hardware signals to estimate page criticality and decide which pages should remain in fast memory. Under workload colocation, however, these estimates become aliased across workloads. Shared LLC interference distorts whether accesses reach memory, while downstream memory contention alters the stall cost they incur, causing page criticality estimates to conflate the behavior across workloads. Together, they create a fundamental observability gap for tiered memory management.

We present Tierce, an observability substrate for colocated tiered memory. Tierce estimates workload-local exposed stall through per-workload memory-level parallelism measurement and per-access latency-weighted page scoring, and bounds LLC interference through dynamic cache allocation. Under colocated tiering, Tierce outperforms the strongest prior design by up to 12.6% and remains within 8% of each pair's colocated all-DRAM reference in the worst case; some configurations exceed that reference by reducing shared-resource contention.

CCS Concepts: • Hardware → Emerging technologies; • Computer systems organization → Architectures.

Keywords: Tiered Memory, Compute Express Link (CXL), Operating Systems

1 Introduction

AI-driven demand and slowing DRAM density scaling have made memory capacity an increasingly scarce and costly resource in modern datacenters [3, 22, 36, 46, 63]. Consequently, hyperscalers have widely adopted tiered memory systems, using technologies ranging from Intel Optane to

Compute Express Link (CXL) to expand memory beyond a server's local DRAM [2, 17, 22, 23, 36]. While this additional capacity reduces cost, it also introduces a fundamental placement problem: accesses served from the slower memory tier incur significantly higher latency, making it critical to identify which pages should remain in fast memory.

Production tiered-memory systems rarely manage workloads in isolation. Instead, an operator-designated foreground (FG) workload shares the CPU last-level cache (LLC) and memory resources with one or more best-effort background (BG) co-runners. These colocated workloads may differ substantially in their sensitivity to memory latency.

Existing placement engines rank pages using hardware signals ranging from access frequency (*hotness*) to estimated CPU stall cost (*criticality*) [25, 40, 46, 50, 62]. Criticality-aware systems such as PACT account for memory-level parallelism (MLP) [20, 49], but still estimate page criticality from aggregate hardware observations. Under colocation, shared cache and memory interference distort these observations, making page criticality itself unreliable.

Our key observation is that colocation turns page placement into an *ownership problem*. The observations needed to estimate PAC lose their association with the workload whose placement they should influence before the policy acts. **Time aliasing** occurs after requests enter DRAM/CXL: workloads experience different MLP and queuing delays, but existing systems combine them into aggregate measurements. **Space aliasing** occurs before requests reach memory: shared LLC interference changes which accesses become memory requests, making cache interference indistinguishable from poor page placement [25, 35, 40, 50, 62]. Page migration alone cannot correct either form of lost attribution.

Characterizing 138 colocated executions, we find that $\Delta L2Lat$ is the most consistent explanatory signal across workload classes and placements: it reflects both whether an L2 miss escapes the LLC and the cost incurred afterward. Yet it requires an interference-free baseline that production does not provide and identifies neither the responsible workload nor the page (§3). The central challenge is therefore to recover actionable workload-local information from signals available during colocation.

^{*}Equal contribution.



This work is licensed under a Creative Commons Attribution 4.0 International License.

SCOP² 26, Prague, Czechia

© 2026 Copyright held by the owner(s).

ACM ISBN 978-0-XXXX-XXXX-X/2026/09

<https://doi.org/10.1145/XXXXXX.XXXXXX>

Thank You! Questions?