Tierce: Observability-Driven Tiered Memory Management for Colocated Workloads

Hanchen Xu*
Virginia Tech
Blacksburg, USA

Berkay Inceisci*
Virginia Tech
Blacksburg, USA

Hao Li
Virginia Tech
Blacksburg, USA

Zhenyu Zhang
Virginia Tech
Blacksburg, USA

Huaicheng 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. Colocation corrupts these estimates at two points: shared LLC interference changes whether accesses reach memory, and downstream memory contention changes their stall cost. Both effects blend colocated workloads' behavior into shared page-criticality estimates, creating a fundamental observability gap for tiered memory management.

We present Tierce, an observability substrate for colocated tiered memory. Tierce estimates workload-local exposed stall by measuring per-workload memory-level parallelism and scoring pages by realized access latency. It bounds LLC interference with explicit cache-way allocation. Across 8 colocated pairs that run four foreground workloads against a bandwidth-intensive and a bursty, phase-varying co-runner, Tierce outperforms PACT, the best prior art, on every pair, by 2.2% to 12.6%.

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 scarce and expensive in modern datacenters [15, 39, 54, 79, 83]. Hyperscalers therefore deploy tiered memory widely, using technologies from Intel Optane to Compute Express Link (CXL) to expand capacity beyond a server's local DRAM [7, 15, 17, 39, 58]. This capacity reduces cost but raises a placement problem: accesses served from the slower tier incur much higher latency, so the system must decide which pages stay in fast memory.

Production tiered-memory systems rarely manage workloads in isolation. An operator-designated foreground (**FG**) workload shares the CPU last-level cache (LLC) and memory resources with best-effort background (**BG**) co-runners. Colocated workloads differ substantially in their sensitivity to memory latency [15].

Existing placement engines rank pages using hardware signals ranging from access frequency (*hotness*) to estimated CPU stall cost (*criticality*) [20, 38, 44, 54, 64, 82]. Criticality-aware systems such as PACT [20] account for memory-level parallelism (MLP) [10, 63], but still estimate page criticality from aggregate hardware observations. Under colocation, shared cache and memory interference distort these observations, making the resulting estimates unreliable.

Colocation turns page placement into an *ownership problem*: the signals needed to estimate criticality no longer share a common workload scope. **Time aliasing** occurs after requests arrive at 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 the effects of page placement [20, 38]. Page migration alone cannot correct either form of lost attribution.

Characterizing 138 colocated executions, we find that the change in L2 miss latency is the most consistent signal to predict colocation-induced slowdowns across workload classes and placements, reflecting both whether an L2 miss escapes the LLC and the cost incurred afterward. Yet it requires an interference-free baseline that production cannot supply, and it provides no page-level attribution (§3). The challenge is therefore to recover actionable workload information from signals available during colocation.

We present Tierce, an observability substrate that preserves workload ownership across measurement and placement. Tierce estimates workload-local exposed stall with workload-specific MLP (wMLP) and latency-filtered page scoring. Domain-private scoring, migration, and capacity

*Equal contribution.

SOSP '26, September 29–October 2, 2026, Prague, Czech Republic

ACM ISBN 979-8-4007-2585-2/26/09
https://doi.org/10.1145/3830418.3843905

enforcement preserve that attribution. Explicit LLC allocations bound interference in the shared cache, upstream of the miss stream a tiering policy observes. The individual mechanisms are known; the contribution is their composition into a workload-isolated tiering pipeline. Shared caches and memory systems have long made application performance hard to attribute [71, 72, 93]. Tierce identifies the page-placement consequence: under colocation, the signals that page criticality is estimated from fragment across mechanisms (§2).

Our evaluation spans 8 colocated pairs on Skylake (SKX) servers (four foregrounds against two co-runners). Tierce outperforms the best prior art, PACT [20], on every pair by 2.2% to 12.6%. Its worst cases are 0.5% above the untiered baseline and 7.4% below a pair's colocated all-DRAM reference. Its largest margins on that suite over NUMA balancing memory tiering (NBT) [89], Colloid [82], and Memtis [38] are 22.7%, 25.9%, and 64%, respectively. In summary, we make the following contributions:

- We identify time and space aliasing as an observability gap in colocated tiered memory, where signals lose workload ownership before the policy acts.
- We design Tierce, an observability substrate that combines workload-isolated placement, where each workload's cores and memory cgroup form a disjoint execution domain, per-workload MLP measurement, per-access latency-filtered page scoring, and bounded LLC residency policy into an online tiering pipeline.
- We evaluate Tierce against four state-of-the-art tiering designs. Across 8 colocation pairs, Tierce improves performance by up to 12.6% over the best prior art, PACT.

2 Background and Related Work

2.1 Tiered Memory Request Processing Flow

Figure 1 shows the path of a memory request through the shared LLC and memory tiers, where cross-workload interference begins. After an LLC miss, the request enters the uncore, where the Caching/Home Agent (CHA) routes it to local DRAM or CXL and tracks it until completion. Intel processors expose complementary mechanisms along this path: Precise Event-Based Sampling (PEBS) samples LLC-miss accesses, the CHA's Table of Requests (TOR) measures outstanding requests and MLP, and Cache Allocation Technology (CAT) partitions cache capacity [28, 32, 41]. Each covers only one stage of the request lifetime, so tiered-memory systems must combine them to estimate page importance.

2.2 From Hotness to Criticality

Tiered-memory systems initially ranked pages by access frequency (*hotness*) [38, 54, 64, 84, 86], while recent systems prioritize the CPU stall saved by promotion [14, 20, 44]. A parallel line of work targets CXL specifically, optimizing migration placement for the higher-latency tier [19, 69, 73]. PACT formalizes the stall-saving objective as *Page Access Criticality (PAC)*, which discounts latency hidden by MLP [10, 63]. MLP has long shaped memory management, from scheduling [59] to indexing [91], yet hardware exposes it only in aggregate [34]. PACT samples accesses with PEBS at page granularity, estimates per-tier MLP from aggregate counters, and updates each sampled page's PAC from its accesses and MLP-discounted stall cost. Pages above the third quartile (Q_3) of the score distribution become promotion candidates. Demotion candidates come from Linux LRU. Score distribution, Q_3 threshold, promotion queue, and fast-tier budget are all global across workloads. Tierce builds directly on PACT: it retains PAC as the placement objective but replaces the aggregate measurement, attribution, admission, and LLC-control pipeline beneath it with workload-local ones (Table 1).

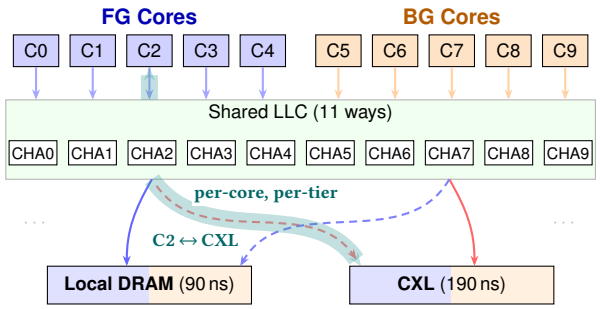

Figure 1. CHA/TOR on Skylake. *FG and BG cores share an 11-way LLC across 10 CHA slices, which route misses to local DRAM or CXL. Teal highlights one per-core, per-tier request path, the granularity Tierce isolates with the Thread/Core-ID (TID) filter (§4.3).*

Table 1. Tierce vs. PACT. *Pipeline differences beneath PAC.*

	PACT [20]	Tierce
Scoring	Global pipeline	Independent per-domain
MLP	Aggregate per-tier	Workload-isolated (wMLP)
Attribution	Uniform per-access	LDLAT-selective per-access
Admission	Global threshold	Per-domain threshold
LLC	No control	Predictive CAT control

2.3 Observability Gap for Colocated Tiered Memory

Isolated executions obtain PAC from a single workload, making aggregation across the memory hierarchy a reasonable approximation. Colocation interleaves these observations across workloads (Figure 1). Aggregate hardware measurements therefore fail to estimate workload-local PAC reliably.

Reconstructing PAC under colocation means recovering, for every sampled memory access, each of the dimensions listed in Table 2. PEBS, CHA/TOR, PEBS load-latency sampling (LDLAT), and CAT each expose a different subset of them at a different granularity, and none is sufficient on its own. OS tiering adds another source of coupling: Linux demotion is driven by per-node `kswapd` daemons reacting to node-wide fast-memory pressure, so reclaim is shared by every workload on a node and cannot be steered per workload [11, 47, 92].

Table 2. Signals. *Existing hardware exposes complementary observations at different granularities. Ownership: which workload and page issued a sampled access. Residency: whether it was served from the LLC or reached a memory tier. Overlap: how much of its latency was hidden by concurrent outstanding requests (MLP). Latency: its realized time to complete in the cache and memory hierarchy.*

Mechanism	Ownership	Residency	Overlap	Latency
PEBS LLC-miss	Page	LLC misses	×	×
PEBS LDLAT	Page	Data source	×	Per-access
CHA/TOR	×	Memory tier	Per-tier	×
CAT	×	LLC ways	×	×
Linux tiering	Page	Aggregate	Aggregate	Aggregate

2.4 Relation to Prior Systems

Memory disaggregation. Far-memory systems expand capacity at warehouse scale [37, 83], in the application [67], over RDMA [18, 53], and by harvesting stall cycles [48], while migration mechanisms and multi-host CXL [23, 86] reduce the cost of moving pages over a substrate that is specified [12], surveyed [13], and measured on deployed silicon [17, 77]. They decide where capacity comes from; Tierce decides which pages occupy it, attributing access cost to the workload and page that incurred it.

Multi-tenant memory management. Existing systems regulate fast-memory allocation, fairness, admission, or VM placement [15, 36, 44, 47, 65, 68, 76, 92], determining how much fast memory a workload receives; Tierce instead estimates which pages are most valuable. Memstrata [94] estimates per-VM slowdown under hardware flat memory mode to allocate fast-memory capacity, whereas Tierce estimates per-page PAC online to guide page placement.

Cache management. Intel CAT and cache-QoS systems partition LLC capacity or bandwidth for performance isolation [8, 9, 33, 46, 52, 60, 75, 90, 95]. Tierce uses bounded LLC residency control to allocate and stabilize the cache residency observed by the tiering engine.

Memory-system observability. Recent work expands telemetry for DRAM/CXL controllers and system monitoring [39, 40, 42, 43, 45, 51, 74, 78]. CPUs increasingly expose fine-grained sampling capabilities through hardware counter facilities [2, 29]. OS access tracking has likewise moved from `AutoNUMA` hint faults [54, 88] to region-based monitors [22, 61, 62]. Tierce composes existing measurements and controls into a workload-isolated PAC abstraction.

3 Characterizing Colocation Interference

We first characterize workload performance under colocation and analyze hardware metrics to explain the observed slowdowns, providing insights for an online OS policy.

3.1 Methodology

Setup. We use dual-socket SKX servers on CloudLab (Figure 1), with 90 ns local DRAM and a remote NUMA node as a controlled 190 ns CXL-latency proxy, which we refer to as "CXL" throughout for brevity. Nine FGs span three classes: memory-bound (`bc-kron` from GAPBS, Silo, XSBench, Masstree, VectorDB), cache-sensitive trace replays (CacheLib Meta/Twitter traces), and network-bound services (Redis, Memcached) [16, 55–57, 66, 80, 81, 85]; "memory-bound" means colocation slowdown is dominated by memory-access cost rather than network/compute; classes follow the dominant bottleneck, so the client-server VectorDB counts as memory-bound. Six BG co-runners (SPEC's 649.fotonik3d, GAPBS `bc-kron`, `bc-urand`, `pr-urand`, Intel MLC, VectorDB) [30, 70] are pinned to cores disjoint from the FG. We measure 52 of the 54 FG/BG pairs, omitting the two self-colocations (bc-kron, VectorDB); 17 of the 52 also run under a second BG core allocation; each configuration runs once per placement (all pages on one tier, DRAM or CXL): 69 runs per placement, 138 in all. We use `perf` to sample core and CHA/TOR counters every 100 ms to derive various performance metrics (Figure 2). L3Lat denotes LLC-miss latency, and TORLatLoc and TORLatRem denote the CHA/TOR-derived latencies of local-DRAM and CXL, respectively; L3 and LLC are interchangeable. We correlate experiment means with overall slowdown and instruction-equal phases with within-run slowdown [42]; Pearson and Spearman agree closely for the signals we report.

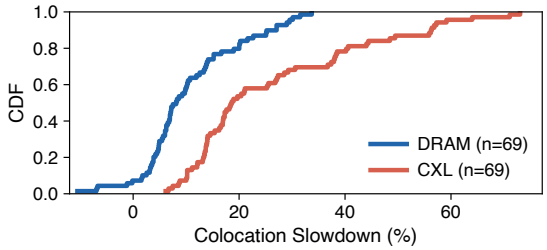

Figure 2. Colocation slowdown CDF under DRAM and CXL. *Slowdown is the colocated runtime relative to solo, minus one.*

3.2 Colocation Slowdown

Figure 2 shows that CXL amplifies colocation slowdown: median slowdown rises from 8% on DRAM to 19% on CXL, and the largest observed slowdown rises from 34% to 73%. Memory-bound workloads slow by up to 34% on DRAM and 71% on CXL, while the network-bound Redis and Memcached stay within 7% and 14% on DRAM but reach 41% and 21% on CXL. Memcached p99 read latency (closed-loop YCSB: 50/50 reads and updates over 10M ≈1 KB records, zipfian θ=0.99, eight client threads, single-threaded server) rises 1.06–1.24× on local DRAM but 1.42–4.25× on CXL. Throughput therefore understates the client-server impact, especially on CXL. Our running example, `bc-kron` with `fotonik3d`, slows by 29% on DRAM and 64% on CXL.

3.3 Which Signals Explain Slowdown?

Figure 3 compares signals across the memory pipeline.

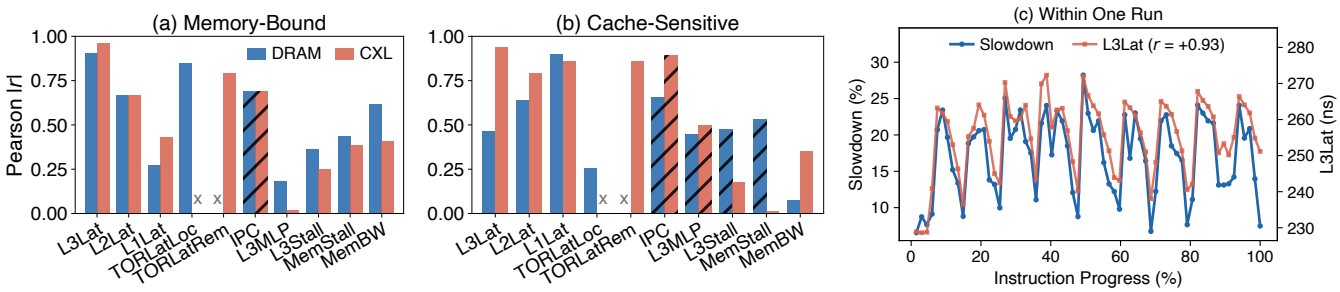

Figure 3. Latency tracks colocation slowdown. (a) *memory-bound and* (b) *cache-sensitive workloads; bars show* $|r|$ *between each signal and colocation slowdown, and hatching marks negative correlations. L1Lat/L2Lat/L3Lat: miss latencies; L3MLP: LLC-miss MLP; L3Stall/MemStall: LLC/memory stalls; MemBW: bandwidth. TORLatLoc/TORLatRem: CHA/TOR-derived latencies of local-DRAM/CXL requests, each defined for one placement only (*x *where undefined).* (c) `silo+bc-kron` *on CXL.*

For memory-bound workloads (Figure 3a), L3Lat is strongest under both placements, reaching $r = 0.91$ on DRAM and $r = 0.96$ on CXL. Bandwidth omits realized latency, stall counters omit ownership, and aggregate per-tier MLP aliases colocated streams, compressing a low-MLP workload's criticality scale (§5.3). For cache-sensitive workloads (Figure 3b), only L1-level load latency tracks slowdown under both placements ($r = 0.90$ DRAM, 0.86 CXL): deeper miss latencies correlate strongly on CXL ($r = 0.79$–0.94) but weaken on DRAM, and instructions per cycle (IPC) and L3MLP correlate negatively ($r = -0.45$ to -0.89), so aggregate per-tier MLP again reflects interference rather than the FG workload's exposed stall. As absolute L3Lat depends on workload baseline, we subtract the solo baseline and use L2Lat to capture LLC escape and downstream queuing.

Stability across phases. Within one Silo run, slowdown varies from 7% to 28% while L3Lat tracks it at $r = 0.93$ (Figure 3c); across instruction-equal phases of memory-bound executions, the correlation remains 0.91 on DRAM and 0.94 on CXL. Latency therefore tracks interference within runs and across workload pairs. For workload w under a fixed placement, define $\Delta\text{L2Lat}_w = L^{\text{colo}}_{2,w} - L^{\text{solo}}_{2,w}$. Subtracting the solo baseline isolates the colocation-induced shift. Table 3 shows that no solo metric exceeds $|r| = 0.45$, while ΔL2Lat reaches $r = 0.91$ across all workloads and $r = 0.96$ after excluding the client-server workloads. Although ΔL3Lat reaches $r = 0.99$ for the memory-bound subset, ΔL2Lat is most consistent across classes and placements.

3.4 ΔL2Lat: An Ideal Hindsight Signal

Why ΔL2Lat? A co-runner increases L2 misses escaping the LLC (*space aliasing*) and inflates their downstream memory service time (*time aliasing*). L2Lat integrates both, which is why it generalizes better than signals isolating cache behavior, memory service, traffic volume, or overlap.

The observability barrier. Despite its predictive power, ΔL2Lat cannot drive an online placement policy: it requires the same workload and placement with and without a co-runner, and it provides no page-level attribution. Thus, it cannot identify which pages the policy should move.

Table 3. Colocation slowdown best correlation. *Rows group scope and placement; columns report best colocated, solo, and Delta correlations. Delta is colocated minus solo; bold marks row maxima. RpqLat uses the read-pending queue; No CS excludes the client-server Redis, Memcached, and VectorDB.*

Scope	Placement	Colo (Best)	Solo (Best)	Delta (Best)
All Apps	CXL	0.79 (TORLatRem)	0.44 (RpqLat)	**0.91 (L2Lat)**
No CS Apps	CXL	0.91 (TORLatRem)	0.28 (RpqLat)	**0.96 (L2Lat)**
Mem-Bound	CXL	0.96 (L3Lat)	0.18 (RpqLat)	**0.99 (L3Lat)**

4 Tierce Design and Implementation

Tierce recovers the PAC dimensions in Table 2 per domain and preserves their attribution through page placement for effective tiering under colocation.

4.1 Design Overview

Existing tiering systems share ranking, admission, migration, and fast-tier capacity across workloads. A high-throughput workload can therefore consume the placement budget of a latency-sensitive one. Commodity hardware also lacks a direct per-workload MLP signal, and page migration does not control the cache residency that determines which accesses reach memory. Tierce addresses these limits with four coordinated components (Figure 4).

- **Domain-isolated placement** establishes ownership boundaries across the pipeline: each execution domain maintains its own page-access samples, PAC statistics, migration queue, and fast-tier capacity.
- **Per-workload overlap reconstruction** derives workload-specific MLP (wMLP) from calibrated per-core, per-tier CHA/TOR telemetry instead of aggregate MLP.
- **Latency-selective admission** refines PAC with PEBS LDLAT samples, scoring only accesses whose realized latency exceeds a hardware threshold. This concentrates the migration budget on pages that incur slow-tier latency.
- **Bounded LLC residency control** regulates upstream cache state with Intel CAT, so the misses Tierce scores are generated under the workload's assigned cache allocation.

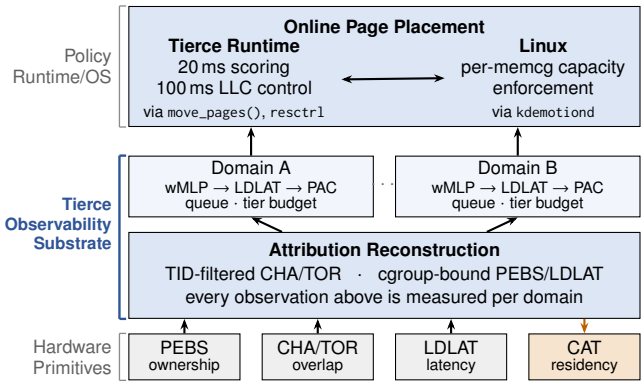

Figure 4. Tierce architecture. *Each stage supplies or preserves one PAC dimension from Table 2.*

4.2 Domain-Isolated Placement

Separating hardware observations is insufficient when workloads still contend for the same OS placement queues. A high-volume co-runner can skew a global admission threshold or consume another domain's fast-tier capacity: under a single shared queue, a memory-intensive BG takes as much as 95% of all promotions in our measurements. The BG starved the FG even when the FG's pages had higher per-access criticality. Algorithm 1 reconstructs scores independently for each domain; the domain-private histogram, admission threshold, queue, and capacity enforcement described here carry that isolation into placement.

Domain-relative ranking and admission. Tierce retains PACT's percentile-threshold admission rule and Linux-LRU demotion candidates [20], but changes their scope: each domain has its own score distribution, threshold, queue, and capacity limit, so one workload's sampling volume cannot alter another's candidate set. We also raise admission from PACT's Q_3 to the 90th percentile to concentrate each domain's budget on fewer, higher-scoring pages (Table 4).

Independent service and enforced capacity. A migration worker visits private domain queues round-robin to ensure fair migration bandwidth, and each domain holds its own fast-tier budget. A promotion into a domain that is already at its limit must first displace one of that domain's own pages, never a neighbor's. §4.6 describes how Tierce makes this budget authoritative against every kernel allocation path.

4.3 Workload-Specific MLP

PAC discounts memory latency hidden by concurrent requests, so Tierce needs $\mathrm{MLP}(\text{domain}, \text{tier})$. Standard core counters conflate destination tiers, while unfiltered uncore (CHA/TOR) counters merge requests from all cores.

Turning uncore filtering into an OS signal. Intel CHAs expose a Thread/Core-ID (**TID**) filter and memory-destination filters [24], but the platform-specific encoding is not mapped to Linux logical CPU IDs, so OS policies cannot use it directly. Tierce calibrates the TID-to-CPU mapping once at initialization. It pins an LLC-miss generator to one logical CPU, sweeps the TID range across all CHA slices, and records the encoding whose TOR activity matches the pinned generator. Tierce repeats the calibration for all cores. Tierce measures $\mathrm{MLP}_{w,t}$ for domain w and memory tier t. For PAC's CXL samples, $\mathrm{MLP}_w \equiv \mathrm{MLP}_{w,\mathrm{slow}}$.

Algorithm 1: Domain-specific PAC sampling

Input: for each domain w, its assigned core set (C_w) and sample count (N_w); PEBS period (P); CXL base cost (k, constant)

foreach *domain* w *with* $N_w > 0$ **do**
  foreach *assigned core* c **do**
    Read TID-filtered, slow-tier CHA/TOR counters;
    $\mathrm{MLP}_c \leftarrow \frac{\Delta\mathrm{TOR\_OCC}_c}{\Delta\mathrm{TOR\_CYC}_c}$;
  $\mathrm{MLP}_w \leftarrow \frac{\sum_{c\in C_w} \mathrm{MLP}_c}{|C_w|}$;
  $S_w \leftarrow k \cdot \frac{\mathrm{LLC\text{-}misses}_w}{\mathrm{MLP}_w \cdot N_w \cdot P}$;
Aggregate PEBS samples by domain and page as counts $A_{w,p}$;
foreach *sampled page* p *owned by* w **do**
  $\mathrm{PAC}_w[p] \leftarrow \mathrm{PAC}_w[p] + A_{w,p} \cdot S_w$;

Deriving wMLP. Because physical-address hashing scatters a core's requests across all CHA slices, Tierce applies the calibrated filters globally. The hardware lacks enough counters to monitor all core–tier pairs simultaneously, so Tierce time-multiplexes selected pairs across 20 ms epochs.

For a core c, Tierce computes $\mathrm{MLP}_c = \frac{\Delta\mathrm{TOR\_OCC}_c}{\Delta\mathrm{TOR\_CYC}_c}$, where TOR_OCC accumulates outstanding slow-tier requests each cycle and the denominator counts cycles with active TOR requests. For a domain w spanning cores C_w, the workload-specific MLP is the average across its cores: $\mathrm{MLP}_w = \frac{\sum_{c\in C_w} \mathrm{MLP}_c}{|C_w|}$. Averaging reduces variation among core-level samples without requiring Tierce to record which thread accessed each page.

Domain-specific PAC. Every 20 ms, Algorithm 1 calculates the domain's base per-sample stall: $S_w = k \cdot \frac{\mathrm{LLC\text{-}misses}_w}{\mathrm{MLP}_w \cdot N_w \cdot P}$. Here, Tierce measures $\mathrm{LLC\text{-}misses}_w$ and MLP_w on domain w's CPUs during the current epoch, so the PAC numerator and denominator describe the same execution domain. The base cost k is PACT's default slow-tier access cost in cycles, retained so the score matches PACT's definition. It does not affect placement because a constant k scales every page in a domain identically and cancels out of the per-domain percentile threshold (§4.2). $N_w \cdot P$ estimates the total accesses represented by the N_w PEBS samples at sampling period P. P counts sampled events, not time: one record per P slow-tier LLC misses, or threshold-exceeding loads under LDLAT. Tierce recomputes these inputs every epoch to follow phase changes.

4.4 Online Latency-Selective Admission

wMLP captures average request overlap but not latency variation among accesses. With a uniform stall cost, a 200-cycle access contributes the same score as one delayed to 1,000 cycles by congestion. PEBS LDLAT tags sampled loads with their realized latency (ℓ_p) and data source. Tierce admits only samples whose latency clears a threshold.

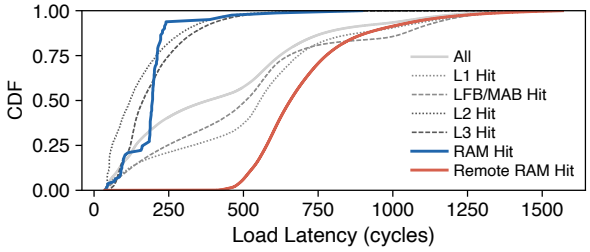

Figure 5. Latency CDF of LDLAT samples by data source. *Solo bc-kron on CXL ($T=32$, $p=10$; one run). Curves partition sampled load latencies by reported data source.*

Characterizing LDLAT selectivity. The PEBS load-latency event emits a record only when a load's latency exceeds a hardware threshold T. At $T = 32$, cache-labeled samples represent the delayed tail of their class (*e.g.*, loads blocked by dependencies) rather than typical hit latencies (Figure 5). Remote-memory accesses separate cleanly at 400 cycles. Tierce therefore sets the hardware floor to $T = 350$ cycles. This captures the slow-tier population while rejecting typical cache hits before they consume PEBS and software-processing capacity. Tierce then filters by data source to remove any remaining cache and local-memory records.

Latency filtering matches PACT with fewer samples. Replacing PACT's PEBS event with LDLAT ($T = 350$, period 5) leaves the published runtime at baseline performance while collecting fewer samples and issuing fewer promotions. Raising the threshold to 700 cycles cuts both again without changing that result (on `bc-kron`). We use these two thresholds as the self-tuning controller's operating points.

Self-tuning software thresholds (*LDLAT-Dyn*). Tierce changes the software admission threshold over accepted records as fast-tier pressure changes. This curbs unproductive migrations when DRAM is scarce. Every second, Tierce computes each domain's eviction pressure $E_w = \frac{\text{pages evicted/s}}{\text{fast-tier pages}}$ (EWMA-smoothed) and takes the maximum across domains, $E = \max_w E_w$. Every 30s, if $E > \tau_E$, it applies a 700-cycle software threshold over the 350-cycle hardware floor and admits only samples above 700 cycles. If promotions still flow and E does not exceed τ_E, it returns to the 350-cycle floor. Whenever E exceeds τ_E^{em}, a per-second emergency check immediately re-applies the 700-cycle threshold, so an overly permissive threshold lasts only a few seconds. We use $\tau_E = 0.002$ and $\tau_E^{\text{em}} = 0.015$ (fractions of fast-tier capacity).

Per-access weighting. Accepted samples also scale the domain's base stall score by the ratio of the observed latency to the slow-tier baseline ($S_p = \frac{S_w \cdot \ell_p}{k_{\text{slow}}}$). This replaces the uniform $A_{w,p} \cdot S_w$ term of Algorithm 1 so a page is never charged twice for the same epoch. Because k_{slow} is an EWMA of $\bar{\ell}$, the mean latency of the accepted samples, the average weight stays near 1.0 in steady state: the ratio moves score toward a workload's slowest pages but leaves its total score unchanged. Latency-based admission speeds convergence after a hot-set relocation (§5.2).

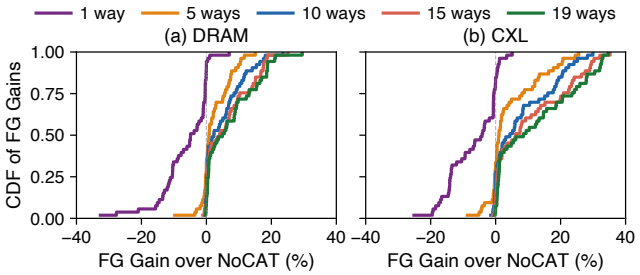

Figure 6. Distribution of FG gain from way allocation. *Per-pair FG gain over the unpartitioned LLC (NoCAT) at each allocation; all 53 pairs, one run per pair and placement (§4.5).*

4.5 Bounded LLC Residency Control

wMLP and LDLAT estimate the cost of observed misses, but neither identifies misses caused by a co-runner's LLC evictions. A line evicted by a co-runner is already gone by the time either signal observes it. We use **NoCAT** for an unpartitioned LLC, **sCAT** for a static partition, and **dCAT** for dynamic allocation.

Cache residency and tiering. CAT controls LLC residency. Its role follows from the stall decomposition [42, 45, 87]: Memory stall ≈ LLC misses × exposed cost per miss. Page placement optimizes the second factor, while unmanaged LLC contention inflates the first before Tierce can sample or migrate a page. Tierce partitions the LLC so that its PACs are computed from misses produced under a controlled cache allocation.

An LLC way is worth more when the miss it avoids reaches CXL. Sweeping foreground allocations across 53 colocated pairs (five FGs against co-runners at 4–16 threads, so pairs vary in BG pressure; a matrix distinct from §3) on an Emerald Rapids (EMR) 20-way LLC (Figure 6) shows wide variation in cache sensitivity, so no single static allocation works for all workloads. Most workloads recover unpartitioned performance using fewer than half the available ways. Preventing an eviction saves more stall when the avoided miss would otherwise access a slower, congested tier. Accordingly, at the sweep's 15/5 setting, the geomean foreground gain over an unpartitioned LLC is 9.9% when pages reside on CXL but 6.1% on DRAM. The distribution is bimodal (Figure 6): at the 15-way setting, 38% of pairs on DRAM and 32% on CXL gain under 1%, while 26% and 40% gain at least 10%.

Two calibration points predict the whole colocated curve. Across our sweeps, an affine transformation of a workload's solo curve ($g(x) = a \cdot f(x) + b$) predicts its colocated cache sensitivity. In Figure 7, fitting a, b at the two circled allocations rescales the solo curve onto the colocated one and predicts the other three (crosses) within 0.9 points. The affine model uses two endpoint allocations to predict the held-out curve with 2.0% mean absolute error (worst case 19%) across the 106 sweeps (53 pairs, two placements). The predicted curve yields the *minimal protective allocation*: the fewest ways that keep predicted slowdown within ϵ=1% of the unpartitioned baseline. In 91 of the 106 cases the predic-

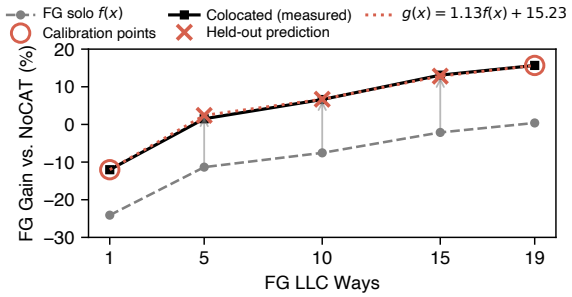

Figure 7. Affine prediction of a colocated ways curve. *Silo vs.* `bc-urand`*, a cache-sensitive pair with below-median fit error (§4.5).*

tion picks the same allocation as the measured curve (the *empirical ideal*); 12 of the 15 mismatches over-protect, and 3 under-protect, so an operator can size a static split without an exhaustive online sweep (§5.8).

Dynamic runtime control. A static allocation cannot return unused LLC ways when a workload enters a phase with less cache demand. For each protected domain, Tierce sets $W_{\min}$ to the minimal protective allocation supplied by the affine model and sets $W_{\max}$ to the operator's maximum protected allocation. Our prototype evaluation fixes these values to $W_{\min}$=5 and $W_{\max}$=15 of 20 ways on Intel EMR CPU; this places the static 15/5 reference at the controller's own $W_{\max}$.

Algorithm 2 by default uses the FG's LLC misses-per-kilo-instructions (MPKI) as a conservative demand indicator. Every control interval (I=100 ms), if MPKI falls below a low threshold (τ_{lo}=10), Tierce releases one way down to $W_{\min}$; if MPKI spikes above a high threshold (τ_{hi}=25), it restores $W_{\max}$ in a single step. The response is deliberately asymmetric: a mistaken release can immediately hurt the FG, whereas retaining an unneeded way costs only BG opportunity. The same control loop can use different demand signals; §5.8 evaluates L2Lat and a probing policy. CAT benefits an FG with cache-timescale reuse when an eviction would expose a miss to an expensive tier; it provides little benefit to streaming domains with little reusable LLC state. Over-allocation can also harm a cache-sensitive BG, so the bounded controller favors FG protection over aggressive reclamation.

Interaction with migrations. CAT changes the LLC miss stream at the 100 ms interval before page placement acts; migration determines the cost of requests that still miss as page residency changes over longer intervals.

4.6 Implementation

We implement Tierce by extending PACT's userspace runtime and the Linux kernel. The runtime reconstructs workload-isolated PAC and decides placement (Figure 4), while the kernel provides a per-memory-cgroup tiering substrate that enforces domain-local capacity and migration.

Runtime. For each execution domain, the runtime maintains an independent PAC table, score histogram, promotion queue, and LLC/CAT allocation state. Using the Linux `perf_event` interface, it programs both core and uncore counters

Algorithm 2: Bounded LLC allocation. W is the FG's way count out of W_{tot}; the co-runner receives the remainder. Tierce uses the FG's MPKI as the demand signal σ (§4.5).

Input: demand signal σ; release/restore band $\tau_{\text{lo}}, \tau_{\text{hi}}$; control interval I; bounded range $[W_{\min}, W_{\max}]$ of W_{tot} ways
Output: CAT way allocation $(W, W_{\text{tot}} - W)$, updated every I

$W \leftarrow W_{\max}$; apply CAT allocation $(W, W_{\text{tot}} - W)$;
while *protected domain is running* **do**
  Sleep I; $s \leftarrow \sigma(\text{domain})$;
  if $s > \tau_{\text{hi}}$ **then** $W \leftarrow W_{\max}$;
  else if $s < \tau_{\text{lo}}$ **then** $W \leftarrow \max(W - 1, W_{\min})$;
  else W unchanged;
  Apply CAT allocation $(W, W_{\text{tot}} - W)$;

to collect the telemetry required to reconstruct PAC. During each epoch, the runtime drains PEBS/LDLAT records, snapshots the corresponding CHA/TOR and LLC-miss counters, reconstructs wMLP, updates PAC, refreshes the admission threshold, and enqueues promotion candidates. The runtime batches promotion requests via `move_pages()` and updates CAT allocations through Intel's `resctrl` interface [41]. The runtime uses the multiplexed CHA/TOR collection of §4.3.

Linux kernel. The kernel implements a per-memory-cgroup tiering substrate that provides domain-local fast-tier accounting, pressure tracking, promotion, and demotion. Each memory cgroup maintains an independent fast-tier capacity and associated fast-tier pressure watermark. Like Linux's per-node `kswapd`, Tierce instantiates one `kdemotiond` daemon per cgroup–node pair. When a promotion would exceed a domain's fast-tier budget, `kdemotiond` first reclaims a cold page from the same domain's LRU before admitting the promotion. Promotion, demotion, and capacity enforcement therefore never cross domain boundaries.

To make the fast-tier budget authoritative across all kernel allocation paths, Tierce replaces Linux's deferred residency accounting with synchronous per-cgroup, per-node accounting and drains per-CPU page staging vectors before each `kdemotiond` scan. Thus, admission accounting and demotion always operate on the same resident page set regardless of whether pages are promoted through `move_pages()`, NUMA balancing, or other kernel migration paths.

4.7 Discussion and Limitations

Configuration constants. Table 4 collects the fixed constants of §4 with the basis for each value. Several values carry sensitivity evidence: the period is swept in Table 5, the latency threshold across capacity regimes in §5.7, and the LLC controller's constants in §5.8; the rest were selected empirically during development and were not swept.

Hardware assumptions and evolution. The Tierce prototype uses Intel PEBS/LDLAT, CHA/TOR counters, and CAT, validating TID-filtered CHA/TOR on both SKX and EMR despite differing TID encodings [24–27]. Analogous primitives exist elsewhere (AMD IBS/PQoS, Arm SPE/MPAM [1, 3–6]).

Table 4. Tierce configuration constants. *The prototype's constants and the basis for each value.*

	Parameter	Value	Basis
PAC	PAC epoch	20 ms	PACT default
	Admission percentile	0.90	empirical
LDLAT (§4.4)	Hardware floor T	350 cyc	characterization (Fig. 5)
	LDLAT PEBS period	5 events	matched volume
	wMLP PEBS period	100 events	PACT-matched
	Software gate	700 cyc	threshold sweep
	τ_E / τ_E^{em}	0.002 / 0.015	empirical
	Dyn decision / emergency	30 s / 1 s	empirical
	EWMA β (k_{slow})	0.3	empirical
LLC (§4.5)	Control interval I	100 ms	empirical
	τ_{lo} / τ_{hi} (MPKI)	10 / 25	empirical
	W_{min} / W_{max}	5 / 15 of 20	affine predictor
	Tolerance ϵ	1%	predictor objective

When a capability is unavailable, Tierce falls back to coarser attribution, such as aggregate MLP or uniform PAC weighting, while preserving domain-isolated placement. Richer counters can improve these signals but do not remove the ownership problem: Intel Diamond Rapids, for example, can attribute PEBS accesses across memory regions including CXL [31, 49], but still identifies a memory region rather than the execution domain that generated the access.

When the mechanisms pay. Page placement needs tiering headroom: where migration cost exceeds the placement benefit, tiering policy trails NoTier (§5.6). LLC protection needs a protected workload that benefits from more cache, and reclamation needs a co-runner that can exploit released ways. FG protection applies across the evaluated co-runners. The BG throughput gains require a cache-elastic co-runner and therefore have narrower scope (§5.8).

Deployment and evaluation scope. Tierce targets trusted host/hypervisor deployment, where execution domains, counter telemetry, and LLC partitions can be managed without guest modification. Because PEBS/LDLAT records may expose instruction pointers, virtual addresses, and access latencies, sampled data remain within the trusted runtime; guest-visible virtualized counter telemetry remains future work [22, 68]. Our evaluation uses an emulated CXL setup (§5.1), so we do not isolate device, controller, or link effects. The LLC way-allocation mechanism, the sweep, affine predictor, and the release-signal study, is characterized on EMR's 20-way LLC only; on SKX we use a fixed split without repeating that analysis. Finally, scaling Tierce's time-multiplexed CHA/TOR monitoring and CAT controller beyond two concurrent domains remains outside our evaluation.

5 Evaluation

We isolate each mechanism (§5.2), test combined gains and tail latency on EMR (§5.3, §5.4), generalize to the 8-pair suite and solo runs on SKX (§5.5, §5.6), test robustness without retuning (§5.7), and compare LLC control signals (§5.8).

5.1 Methodology

Hardware. We evaluate on dual-socket Intel SKX and EMR servers and emulate a CXL tier with the remote NUMA node (190 ns vs. 90 ns local DRAM on SKX; 199 ns vs. 110 ns on EMR, measured with Intel MLC). A per-cgroup capacity limit splits each workload's memory between the fast and slow tiers; unless stated otherwise, the ratio is *1:1* (50% of RSS on DRAM). Both machines run our modified Linux 6.3. We use SKX for the larger suite and to compare with prior art. We use EMR's 20-way LLC to characterize way allocation (§4.5).

Workloads. Our main suite uses 9 foreground workloads spanning three application classes: Silo [80] (OLTP database), 657.xz from SPEC CPU 2017, and seven GAPBS [16] graph kernels (bc-twitter, bc-kron, bc-urand, pr-twitter, sssp-kron, tc-kron, and tc-twitter). Resident set sizes span 7–42 GB, so foregrounds overflow their fast-tier budgets and contend for placement. The suite contrasts irregular, latency-bound access (GAPBS and Silo) with streaming, bandwidth-bound access (657.xz, and 649.fotonik3d as a co-runner). These workloads also appear in the Memtis, Colloid, and PACT evaluations. Solo experiments run 9 workloads with 8 threads; colocation experiments run 4 threads per workload and pair four FGs (bc-kron, bc-urand, silo, sssp-kron) against two BGs. We choose FGs for tiering headroom and access-pattern diversity: `bc-kron` and `bc-urand` carry the largest headroom in the solo suite (Figure 14), while `silo` (OLTP) and `sssp-kron` (low-MLP pointer chasing) extend workload-class coverage; the BGs contrast a bandwidth-saturating co-runner (649.fotonik3d) with a bursty, moderate one (pr-twitter). 649.fotonik3d is also a foreground only in the capacity sweep of §5.7. The LLC way-allocation experiments in §5.8 use `cc-twitter`, `pr-kron`, and `cc-urand` as foregrounds and GAPBS's Shiloach-Vishkin connected-components kernel on `kron25` (`cc-sv-kron`) as a cache-elastic background. The characterization suite (§3) spans more workload classes; the evaluation focuses on workloads with tiering headroom, which we define as the performance gap between all-DRAM and no-migration execution.

Systems. We compare eight tiered configurations plus an *All-DRAM* reference (all memory on DRAM, no CXL): *NoTier* (memory spread across tiers with no migration), *NBT* [89], *Colloid* [82], *Memtis* [38], *PACT* [20], and two Tierce ablations named by the signals they enable, *wMLP* (per-workload MLP, which requires per-workload isolated migration management to act on it) and *wMLP+LDLAT* (adding per-sample latency-filtered scoring at threshold 350 cycles). The complete Tierce adds a static LLC way allocation (sCAT) on top of wMLP+LDLAT: 15/5 of EMR's 20 ways in §5.3, and 8/3 of SKX's 11 ways in §5.5. We evaluate the dynamic controller of §4.5 separately in §5.8. wMLP samples at period 100 and wMLP+LDLAT samples the dedicated load-latency event at period 5 (§5.7). We size PACT's PAC metadata pool to cover each working

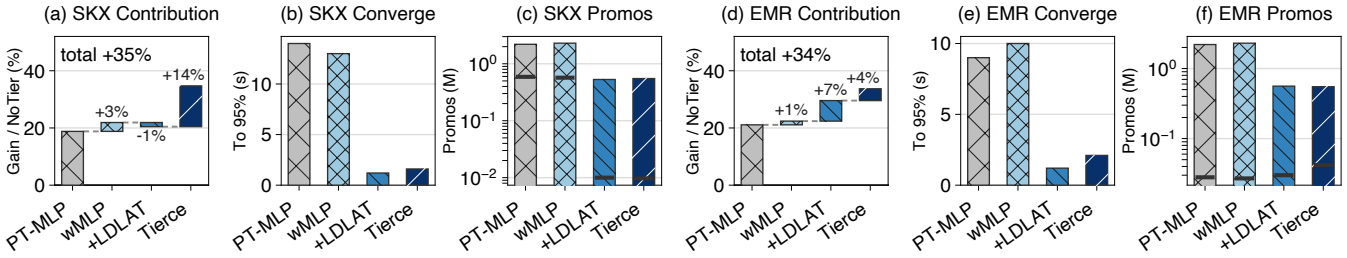

Figure 8. Per-signal microbenchmark. *Signals are enabled in turn from NoTier; wMLP replaces per-tier MLP (PT-MLP) under an otherwise identical per-workload pipeline. (a–c) are SKX (11-way, 13.75 MiB LLC), (d–f) EMR (20-way, 160 MiB).* (a,d) *per-step change in post-swap chase rate, as a percentage of that platform's NoTier rate, so the steps sum to the total.* (b,e) *time to reach 95% of the phase-final rate.* (c,f) *pages promoted (log axis); bars are FG, horizontal dashes BG. Values are means of three to eight runs each.*

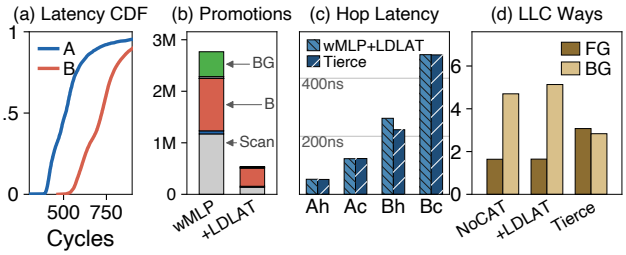

Figure 9. Microbench. *(a) Latency CDF of the FG's sampled loads, dataset A vs. B; (b) Promotions by target region; (c) Per-hop chase latency; (d) Average FG and BG LLC occupancy.*

Table 5. Sampling-period sensitivity. *wMLP at 4 PEBS periods vs. wMLP+LDLAT at (period 5, T=350), on the solo EMR microbenchmark. Columns report run-total promotions, post-swap chase rate, and B-hot fast-tier residency at 600 s; cells are run means.*

Policy	Promotions	Rate (Mops)	B @600
wMLP p100	2.21M	6.02	0.92
wMLP p300	833K	5.89	0.66
wMLP p500	556K	5.83	0.59
wMLP p1000	300K	5.70	0.39
wMLP+LDLAT	**558K**	**6.24**	**0.87**

set because its shipped 2Mi-entry default fills on every pair, after which PACT cannot add pages to the managed set. Every PACT result uses the uncapped configuration. PACT's released artifact defaults to PEBS period 400; we evaluate it at both that default and wMLP's period 100. For each workload or pair, we report the better PACT result.

Metrics. We report application throughput and inverse runtime as a percentage of the all-DRAM baseline; §5.4 reports p99 in µs. The controlled microbenchmark studies (§5.2, §5.8) have no all-DRAM configuration; they report percent change against the baselines of NoTier or the unpartitioned configuration. Unless noted otherwise, we run each experiment at least three times and report the mean or median.

5.2 Microbenchmark: Understanding Each Signal

Application-level results cannot isolate which mechanism corrects a placement decision, so we begin with a controlled benchmark that adds Tierce components incrementally.

Setup. The foreground (FG) emulates an in-memory database with two pointer-chasing datasets (A and B), an index (I), and a scan buffer (S). The datasets have identical sizes and access frequencies but differ in memory layout. Accesses to B are 1.4–2× more expensive than accesses to A. The BG is a bandwidth-intensive streaming workload with high MLP. Both domains are capped at half their working set in the fast tier, so the scan buffer, index, and hot regions cannot all be fast at once. Midway through execution A and B exchange layouts without moving pages, and we measure only the post-swap chase rate, so that rate reflects adaptation rather than initial placement. Scan throughput varies by only 9–13% across configurations; the chase rate is therefore the more sensitive measure of placement quality.

wMLP restores the FG's PAC scale. Aggregate per-tier MLP mixes the FG's requests with the BG's and thus overestimates the FG's overlap. Replacing it with wMLP promotes stranded hot pages 1.4–1.8× sooner than scan-buffer pages. wMLP keeps 90% of B's hot pages in the fast tier vs. 86% under aggregate MLP, and improves end-to-end throughput by 2.6% with near-identical promotion volume (Figure 8a,c).

LDLAT concentrates promotions on expensive accesses. The two hot datasets are intentionally indistinguishable by access frequency, but their realized-latency distributions diverge substantially (Figure 9a). LDLAT directs a larger fraction of a smaller promotion volume to B: over the 20 s after each phase change, the share of FG promotions reaching B's hot region rises from 9% to 40%, while FG and total promotions fall by 4.3× and 5×, respectively, as unnecessary BG promotions nearly disappear (Figure 9b). The absolute number of promotions reaching B remains essentially unchanged. After the swap, wMLP+LDLAT recovers 95% of the phase-final chase rate within 1.2 s versus 13 s for wMLP (Figure 8b), because the threshold surfaces the displaced hot pages first. Sweeping wMLP's sampling period shows that lower promotion volume alone does not explain the result (Table 5): p500 matches wMLP+LDLAT's promotion budget (556K vs. 558K) but recovers far less of the hot set (0.59 vs. 0.87) and achieves a lower post-swap rate; p100 requires 4× the promotions, while longer periods fail to retain the hot set. wMLP+LDLAT's threshold selects its own volume (§5.7).

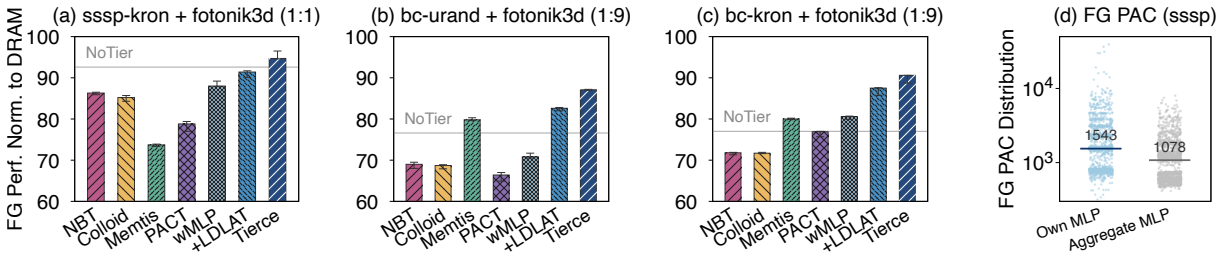

Figure 10. Three colocation pairs, and the FG's PAC values. *(a)–(c) FG performance for three EMR pairs, normalized to each pair's all-DRAM result; BG is* `fotonik3d`*. Bars are medians; error bars span min–max over three runs. The y-axis starts at 60. (d) PAC values for* `sssp-kron` *under per-workload and aggregate MLP, on a log axis; rules mark medians.*

CAT bounds upstream cache interference. Bounded LLC residency control improves the post-swap chase rate by a further 14% of NoTier (Figure 8a), and 98% of the latency reduction comes from B's hot pointer-chasing hops (Figure 9c). The index I is the only FG structure whose reusable working set fits in the LLC; the scan buffer and BG stream through memory. Hardware counters confirm that CAT doubles the FG's LLC occupancy and halves the BG's occupancy (Figure 9d). The BG incurs negligible loss because it has essentially no cache-timescale reuse.

Replication on EMR. Together, the four mechanisms reach 34% over NoTier on EMR, close to the 35% on SKX (Figure 8a,d), but each mechanism contributes differently: CAT adds 14% on SKX and 4% on EMR, while LDLAT gains 7% on EMR and has no measurable rate effect on SKX (the 1% step is within run-to-run variation). EMR differs from SKX, with a 160 MiB LLC (11.6× larger). EMR's much larger LLC already retains most of the FG index, so CAT has little interference to remove. Its slower far-memory path also makes identifying the costly pages early more valuable. On SKX, by contrast, the 75 s chase phase is long enough for frequency-based promotion to recover the displaced hot set on its own, so it masks LDLAT's faster convergence. wMLP+LDLAT again reaches nearly the same hot-set residency as wMLP while using 4× fewer promotions (Figure 8f).

5.3 Representative Colocated Applications

We evaluate three graph workloads on EMR: placement-insensitive traversal (`sssp-kron`), stable locality (`bc-urand`), and phased locality (`bc-kron`). Each runs with the bandwidth-intensive SPEC CPU2017 workload `fotonik3d`. Under tight DRAM capacity, we use the 700-cycle LDLAT threshold (§5.7), and a 1:9 fast-tier ratio for `bc-urand` and `bc-kron`, and 1:1 for `sssp-kron`. For all three applications, wMLP outperforms PACT, wMLP+LDLAT outperforms wMLP, and the complete Tierce performs best (Figure 10a–c).

PACT runs here at wMLP's own sampling period (§5.1), so the 4.8–11.7% foreground improvement from PACT to wMLP reflects workload-specific MLP, per-domain placement isolation, and the 0.90 admission percentile (§4.2) rather than sampling rate; those three mechanisms change together at this level and are not measured separately. LDLAT adds 3.9–16.7% and cuts promotion traffic by 3–4.5× by concentrating migration on costly accesses. The fixed 15/5 CAT allocation limits upstream cache interference and adds a further 3.5–5.4%. Tierce exceeds PACT by 20.2%, 31.2%, and 17.8% on `sssp-kron`, `bc-urand`, and `bc-kron`, and remains the only configuration that surpasses the NoTier baseline on the placement-insensitive `sssp-kron`.

Workload-specific MLP rescales PAC rather than reorders pages. We disable page promotion and compute PAC from either aggregate per-tier or workload-specific MLP, so both observe identical accesses and produce identical rankings; only the PAC scale differs. With the same 17.2M foreground samples, workload-specific MLP shifts the median PAC from 1078 to 1543 by normalizing each page against the foreground's own memory overlap rather than the aggregate overlap of both workloads. The aggregate per-tier MLP on this platform is 1.31 vs. workload-specific 1.01, so the compression is modest, but PACT's shared admission threshold still lowers the FG's effective priority. Tierce's per-domain thresholds remove that effect.

Migration overhead can erase the placement gain. NBT and Colloid underperform NoTier on both placement-sensitive workloads: migration overhead can outweigh the benefit of improved placement. Memtis is the strongest prior system on both `bc` pairs but the weakest on `sssp-kron`, where its hotness-driven migration lands 19 points below NoTier. wMLP remains below the NoTier baseline on `bc-urand`; only after LDLAT suppresses unnecessary promotions does performance on these pairs exceed NoTier execution.

Cache isolation incurs little cost to the co-runner. The static 15/5 CAT partition reduces the background workload's throughput by only 2–3% relative to wMLP+LDLAT, whereas LDLAT's reduction in promotion traffic improves the same workload by 17–19%. For this streaming co-runner, memory bandwidth rather than LLC capacity is the dominant shared resource. Across the three pairs, the 3.5–5.4% FG gains from cache isolation exceed the 2–3% BG cost.

5.4 Tail Latency under Colocation

Setup and method. We use the systems of §5.1 on the EMR platform, at a 1:3 fast-to-slow ratio. The `hnswlib` [21, 50]

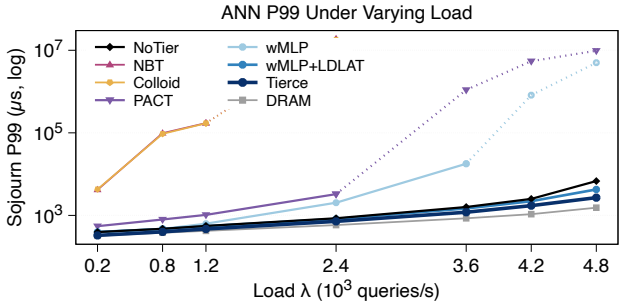

Figure 11. Serving tail vs. load. *Sojourn p99 for the ANN service colocated with the 16-thread* `bc-kron` *batch. Filled markers: the system still serves the offered rate; open markers: load beyond its measured capacity, where p99 is queueing delay. Medians.*

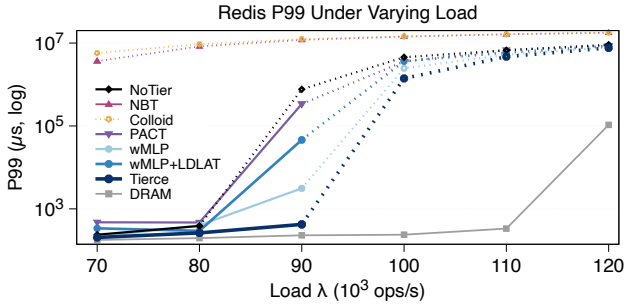

Figure 12. Redis tail vs. load. *Redis p99 under* `bc-kron` *co-runner; Markers follow Figure 11; values are medians.*

approximate nearest-neighbor (ANN) service runs k-NN over SIFT1M [35], one million real image descriptors and their standard query set. Its 660 MB index contains the vectors and per-vector neighbor links traversed by each query and exceeds the fast-tier budget. To model colocation under memory pressure, one thread serves queries while a 16-thread `bc-kron` graph batch runs alongside, and an all-in-fast-memory run provides the reference. Load is *open loop*: an identical precomputed random schedule issues queries at average rate λ independent of completions, so server slowdown does not throttle arrivals and hide queueing. We report *sojourn* time from scheduled arrival to completion, including both queueing and service.

Results. At near idle, every system maintains the offered throughput. Performance differences emerge as queueing pressure increases while we sweep λ from 200 q/s to 4,800 q/s, just below the reference's saturation. NBT and Colloid fail first: the kernel unmaps pages for NUMA hinting and repairs them in whichever thread touches them next. Even before queueing, the median query service time rises from 168 µs to 539 µs, and p99 reaches 2.6 ms. The millions of hint faults logged show that NUMA hinting raises request cost before queues build. Queueing drives tails to ~100 ms by 800 q/s; 1,200 q/s is the last load they serve in full, and from 2,400 q/s their queues never drain. PACT and wMLP sustain higher loads, but promotion churn causes their p99 to more than double at moderate load, and they collapse past 2,400 q/s and 3,600 q/s, respectively. NoTier, wMLP+LDLAT, and Tierce keep up through 4,800 q/s, at 6.8, 4.3, and 2.7 ms p99 against the reference's 1.5 ms. Only the latency-filtered configurations improve on NoTier: Tierce has the lowest p99 at every measured load, 16% below NoTier at low load and 60% below it at 4,800 q/s, where queueing amplifies the service-time gap. LDLAT limits promotions and the way partition shields the hot working set from the batch, as measured in §5.2.

Redis. We sweep single-threaded `Redis` (10M records of ten 100-byte fields, ≈19.8 GB RSS) under open-loop YCSB zipfian reads (θ=0.99, 64 client threads, 60 s cells, latency from each request's scheduled arrival, warmup trimmed) beside `bc-kron`. The configurations reach different capacity limits in the 70–120 k ops/s range (Figure 12). Colloid last keeps up at 60 k ops/s, NBT at 70 k (hint faults in the serving thread), NoTier at 80 k, every sampling-based tiering configuration at 90 k (measured ceilings 94–100 k), and all-DRAM at 120 k. `Redis` serves on one thread, so per-request cost sets these ceilings: slow-tier accesses raise it, and hint-fault repair raises it further, which is why NBT and Colloid saturate below NoTier. Below the knee, placement is not what sets the tail. At 70–80 k ops/s NoTier and every sampling-based tiering configuration hold p99 in the same microsecond band, 204–474 µs against the reference's 178–197 µs. The crossover occurs at 90 k ops/s. NoTier, past its knee, queues to 765 ms p99. PACT, wMLP, and wMLP+LDLAT still serve the offered rate, but without a way partition their p99 remains at 339, 3.1, and 46 ms, respectively, far above the 230 µs all-DRAM reference. Each cell starts with the dataset on the slow tier, so it measures convergence speed as well as final placement, and none of the three converges within its 60 s window. Cache isolation is what closes the gap: adding the way partition reduces the tail to 422 µs, approaching the 230 µs all-DRAM reference within the measurement window.

5.5 Colocation Performance

We now widen to the full 8-pair SKX suite. Figure 13 normalizes FG performance to each pair's colocated all-DRAM execution, so a value above 1.0 means the managed configuration outperforms that reference, consistent with tiering reducing the pair's shared-resource contention.

Each added signal lifts the whole eight-pair suite. NBT, Colloid, and Memtis remain below the NoTier baseline on both `sssp-kron` pairs because migration overhead outweighs placement benefit where locality is weak. PACT improves over these baselines on six of the eight pairs through stall-cost-aware placement. Tierce improves further. On these eight SKX pairs, Tierce beats PACT on every pair by 2.2% to 12.6%, the maximum on `silo+pr-twitter` (95.6% versus PACT's 84.9%). Its largest gains are 22.7% over NBT and 25.9% over Colloid (both on `bc-urand+pr-twitter`) and 64% over Memtis (`bc-kron+fotonik3d`). It also leads NoTier everywhere, by 0.5% at worst, and stays within 7.4% of each

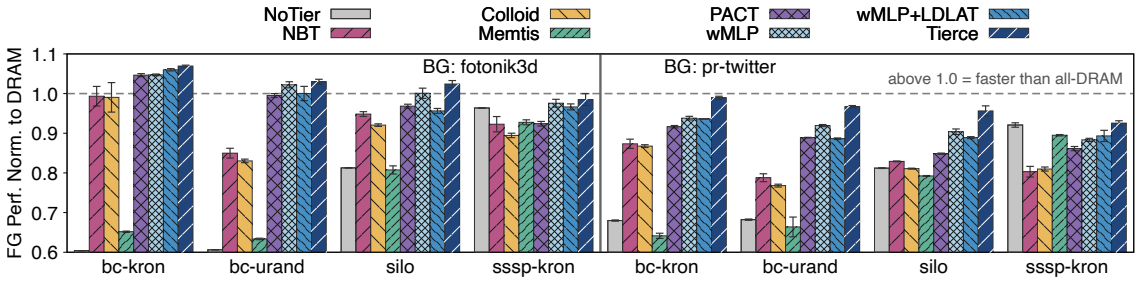

Figure 13. Colocation performance on SKX. *FG performance normalized to each pair's all-DRAM run, 1:1 DRAM:CXL. The y-axis starts at 0.6; bars are means, error bars are standard deviation over three runs.*

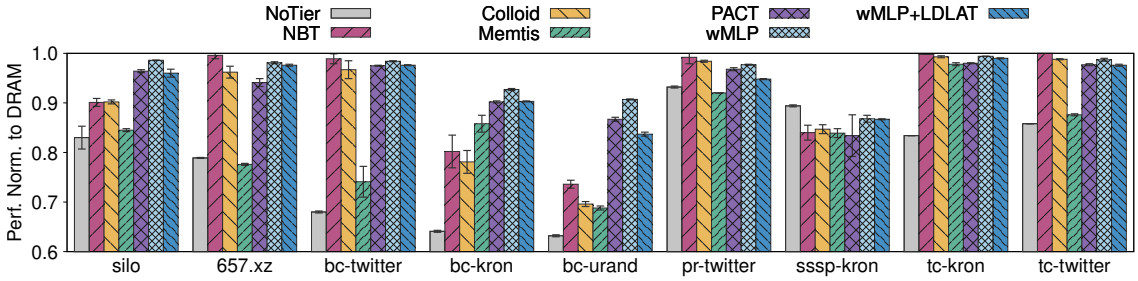

Figure 14. Solo performance on SKX. *Solo performance relative to all-DRAM (= 1) at 1:1. Bars are means, error bars are one standard deviation over three runs.*

pair's colocated all-DRAM reference, both worst cases on `sssp-kron+pr-twitter`.

Bandwidth-intensive co-runners yield the largest gains. The largest improvements over NoTier occur under the bandwidth-intensive `fotonik3d`, where accurate stall attribution enables Tierce to distinguish costly foreground misses from the BG's streaming memory traffic. Against the more cache-sensitive `pr-twitter`, gains over NoTier are smaller, while LLC isolation still improves FG performance.

Promotion analysis. The performance gain does not come from promoting more pages. On the `bc-kron+fotonik3d` pair, we measured 7.7M promotions under PACT, and the FG received only 40% of them (3.0M FG, 4.7M BG), because the aggregate signal follows the BG's higher miss volume. wMLP issues 3.1M and gives the FG 55% (1.7M FG, 1.4M BG): the cut falls disproportionately on the BG, 3.4× fewer against 1.8× fewer for the FG, because per-workload scoring ranks each domain's candidates on its own scale. LDLAT's admission drops samples whose service latency falls below its threshold and trims the volume further (to 2.1M), so Tierce's gain comes from which pages it promotes rather than how many.

5.6 Solo Performance

Solo runs isolate page-ranking quality from colocation interference. At 1:1, wMLP reaches 87–99% of all-DRAM (Figure 14) and is at or above NoTier on eight of nine workloads. On `bc-kron` and `bc-urand`, hint-fault and hotness policies underperform because they promote frequently accessed pages even when those pages' misses do not expose stall. NBT and Memtis reach only 69–86%, while wMLP reaches 91–93%. On `bc-twitter`, `pr-twitter`, `tc-kron`, and `tc-twitter`, NBT, Colloid, PACT, and wMLP all stay above 96% of all-DRAM, and NBT leads wMLP by at most 2.2 points. NoTier and Memtis remain lower, reaching only 68% and 74% on `bc-twitter`. With one workload running, aggregate and workload-specific MLP describe the same request stream, so the gap to PACT is small. PACT stays within 1–4% of wMLP on every workload when we take the better of its two sampling periods (§5.1). Under PACT's own signal design, our runtime also remains in that band on eight of the nine workloads. It trails on `sssp-kron`, at 75% against wMLP's 87%. As expected, Tierce's isolation mechanisms matter only under colocation (§5.5).

Migration overhead can outweigh better page placement. Each migration consumes bandwidth and can stall the application. On `sssp-kron`, every tiering system lands below NoTier (89%). wMLP promotes the fewest pages and loses the least. It reaches 87% against PACT's 83% at 0.7M promotions against PACT's 0.8M. wMLP+LDLAT suppresses promotions further through latency-filtered admission. It stays within 3% of wMLP on eight of nine workloads and promotes 1.3–10× fewer pages; on `bc-urand` the stricter filter costs 7%.

5.7 Robustness without Period Retuning

On EMR, we evaluate four colocated pairs from 1:1 to 1:9 and report the extremes. The FG runs with a continuous BG: `pr-twitter` for GAPBS and `sssp-kron` for `fotonik3d`. We compare NoTier, wMLP (period 100), and wMLP+LDLAT (period 5, threshold 700 cycles) at every ratio.

The admission threshold matters most when the fast

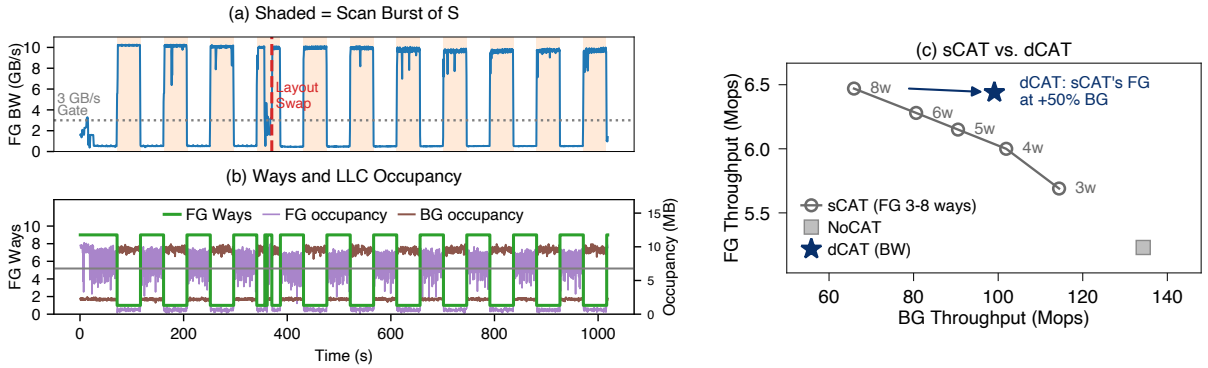

Figure 15. BW on the microbenchmark (SKX; ways out of 11; one run). *(a) FG memory bandwidth, with the* 3 *GB/s gate (dotted). (b) Ways programmed and the resulting LLC occupancy. (c) FG and BG throughput; circles are static splits giving the FG 3–8 ways.*

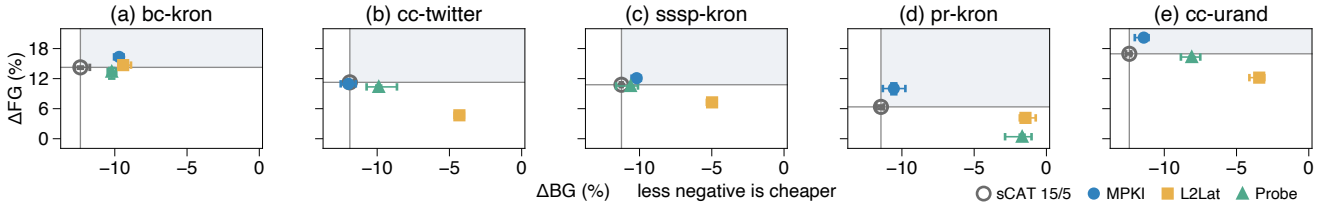

Figure 16. Way allocation tradeoff. *Per-FG throughput against* `cc-sv-kron`*, relative to Tierce+NoCAT; means, error bars min–max over three runs. Dotted rules mark the sCAT 15/5 split; shading marks higher FG throughput at lower BG cost.*

Table 6. Cap robustness. *Performance as % of all-DRAM at 1:1 and 1:9; each arm at its fixed §5.1 configuration (wMLP period 100; wMLP+LDLAT period 5, 700-cycle gate). Promotions at 1:9. Means.*

	% all-DRAM at 1:1			% all-DRAM at 1:9			Promos at 1:9	
Workload	NoTier	wMLP	+LDLAT	NoTier	wMLP	+LDLAT	wMLP	+LDLAT
`bc-kron`	67.2	90.4	83.0	66.5	72.6	**77.5**	8.73M	**764K**
`bc-urand`	67.6	90.2	82.3	64.8	62.2	**71.6**	15.59M	**2.08M**
`sssp-kron`	83.1	76.3	79.7	75.2	66.2	**72.7**	3.21M	**119K**
`fotonik3d`	72.6	74.5	73.5	64.5	59.6	**66.0**	4.21M	**158K**

tier is small. With each configuration held fixed (Table 6), the 700-cycle gate costs performance at 1:1: wMLP leads on three of the four workloads, by up to 7.9 points. At 1:9, wMLP+LDLAT instead leads wMLP on all four workloads, while wMLP falls below NoTier on three.

Latency filtering limits promotions as fast-tier capacity shrinks. As fast-tier capacity tightens, frequency sampling amplifies migration pressure: on `bc-urand`, wMLP promotions grow 5.5× vs. 2.8× for wMLP+LDLAT from a much lower base; on `fotonik3d`, wMLP+LDLAT stays at 153–160K while wMLP roughly doubles. A fixed-period sweep at 1:9 shows the cost of additional promotions: increasing wMLP promotions 5.3× reduces performance from 64.3% to 59.6% of all-DRAM.

Adapting the latency threshold. The better fixed threshold depends on fast-memory capacity. A 350-cycle threshold performs best when capacity is abundant, whereas scarce capacity favors 700 cycles. *LDLAT-Dyn* keeps the 350-cycle hardware threshold and applies a software gate selected from {350, 700} cycles using per-domain eviction pressure every 30s. It keeps the lower threshold under light pressure and switches to 700 cycles once eviction pressure rises. Across all four evaluated cells (`sssp-kron` 1:1, `bc-kron` 1:1 and 1:9, `fotonik3d` 1:9), it lands within 0.6% of the better fixed threshold and bounds promotion traffic without retuning the sampling period. At 1:9, wMLP falls below NoTier due to its promotion volume (Table 6). Promotion decreases monotonically as the sampling period grows (Table 5), but matching the required volume would take reductions of 7–27× across the four workloads, so no single period serves all four.

5.8 LLC Way Allocation

A controller should release a way only when its marginal value to the FG is small. MPKI and L2-miss latency can be collected without changing the allocation, but neither directly measures the benefit of an additional cache way. A probing policy can measure that benefit by temporarily withdrawing a way, at the cost of perturbing execution. We compare passive and probing signals on a phase-changing microbenchmark and five application workloads.

The controller tracks application phases. We reuse the §5.2 foreground but replace the streaming BG with *batch-hog*, a cache-elastic co-runner that converts released ways into work. The bandwidth-gated BW controller releases ways during the scan phase and restores them for the chase phase on the first sample after the bandwidth threshold is crossed, without oscillation: 26 reallocations over the run (Figure 15a,b). Static splits define a tradeoff curve over the 11 cache ways (Figure 15c). The controller matches the FG throughput of the widest split tested (8 ways, the FG allo-

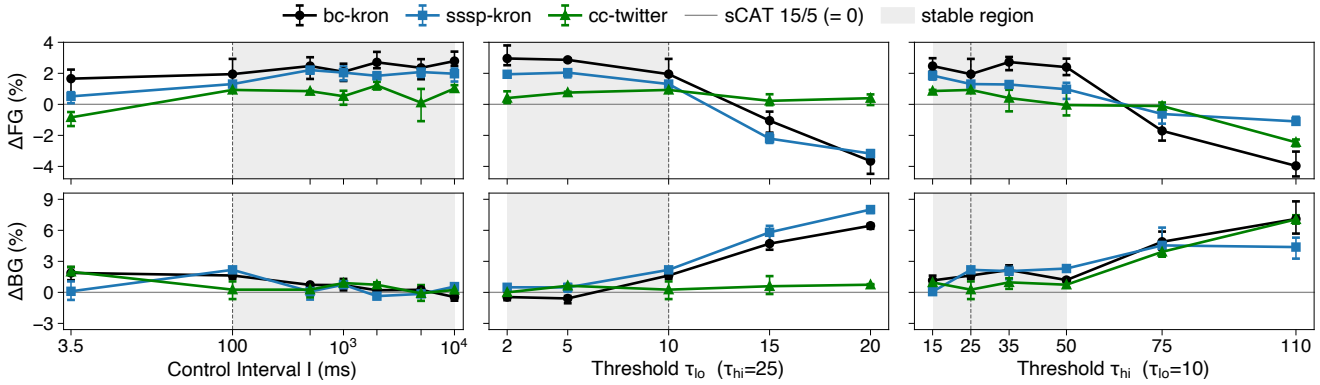

Figure 17. LLC controller sensitivity. *Each Table 4 constant is swept on three representative foreground pairs. Top/bottom panels show the FG/BG throughput change relative to the static 15/5 CAT allocation (grey rule at zero); shading marks each constant's stable range and the dashed line the selected value. Points show means and error bars span min–max across runs.*

cation the SKX evaluation uses, §5.5) while increasing BG throughput from 66 to 99 Mops, or 50%.

MPKI retains excess ways, whereas L2Lat releases useful ways. On EMR's 20-way, 160 MiB LLC, we evaluate five GAPBS foregrounds. dCAT moves each FG's allocation within a fixed range of 5–15 ways. We compare dCAT with a static sCAT 15/5 partition at the controller's own $W_{\max}$, so differences reflect releases alone. We pair each foreground with `cc-sv-kron`, the only cache-elastic workload in our suite (§5.1). MPKI and L2Lat use Algorithm 2 directly, changing only the release signal; Probe adds the trial withdrawal described below. The MPKI policy shipped with Tierce treats continued misses as demand, though a workload can miss heavily even when extra cache would not help it. A low L2Lat, the average L2-miss latency, can indicate either low demand or successful protection. Probe measures value by withdrawing one way periodically and keeps the release only if FG throughput holds. We exclude BW because it targets bandwidth phases. We report allocations against each workload's *knee*, the fewest ways at which it reaches 90% of its full-cache throughput.

Relative to Tierce+NoCAT, the release policies make different tradeoffs between FG recovery and BG cost (Figure 16a–e). MPKI improves on the static split on *both* axes for 4 of 5 workloads. It adds 1.3–3.7 points of FG throughput and returns 0.9–2.7 points to the BG. The exception is `cc-twitter`, where MPKI holds the static allocation for 95% of the run and alone matches sCAT. Probe stays within 0.9 points of sCAT's FG recovery and returns 0.6–4.3 points of BG throughput. On 4 of 5 workloads, L2Lat's more aggressive releases give up 2.2–6.6 FG points for 3.0–10.0 points of BG relief. On `cc-twitter` it spends 61% of the run below the knee; MPKI averages 14.9 ways, Probe 14.0, and neither crosses it. L2Lat beats sCAT only on `bc-kron`, where it releases little. Tierce defaults to MPKI because no online signal captures a way's value directly, and the controller treats an erroneous release as costlier than unnecessary retention.

Sensitivity of the controller constants. We sweep the LLC controller's three constants (Table 4) on the §5.8 foreground pairs; Figure 17 shows three representative results. All chosen values lie within broad stable regions. The control interval is robust from 100 ms to 10 s. Longer intervals reduce BG gains by releasing fewer ways, while faster control ($I \approx 3.5$ ms) reacts to noise and costs the FG up to 2%. The threshold τ_{lo} is insensitive over $\tau_{lo} \in [2, 10]$. Increasing it from 10 to 20 moves `sssp-kron` from 1.3% above the static split to 3.2% below it, while the BG gain grows to 8%, as the controller reclaims ways more aggressively. The threshold τ_{hi} is stable over $[15, 50]$. Increasing it to 75 and 110 delays restoration and leaves `bc-kron` 1.7% and 4.0% below the static split, respectively. Inside the shaded regions the worst FG cost is 0.1%, while outside them it reaches 4.0%: constants beyond the stable range expose the expected tradeoff between FG protection and BG reclamation.

6 Conclusion

Colocation silently aliases the signals on which tiered-memory policies depend. As an observability substrate, Tierce restores their workload scope by isolating placement state, measuring MLP per workload, filtering samples by latency, and bounding LLC interference. By preserving attribution from observation through placement, Tierce estimates workload-local exposed stall for page placement and consistently outperforms the best prior art.

Acknowledgments

We thank the anonymous shepherd and SOSP'26 reviewers for feedback that greatly improved the paper. We used Claude Code and ChatGPT to assist with code and writing; the work is otherwise the authors' own. This research is partially supported by an NSF CAREER Award CNS-2339901, NSF Grants CNS-2312785 and TIP-2550145, and Google.

References

[1] AMD. 2024. AMD64 Architecture Programmer's Manual. Retrieved August 8, 2026 from https://www.amd.com/content/dam/amd/en/documents/processor-tech-docs/programmer-references/24593.pdf

[2] AMD. 2024. Performance Monitor Counters for AMD Family 1Ah Model 00h-0Fh Processors. Retrieved August 8, 2026 from https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/programmer-references/58550-0.01.pdf

[3] AMD. 2025. AMD PQOS White Paper for AMD EPYC 9004 and 9005 Series Processors. Retrieved August 8, 2026 from https://docs.amd.com/v/u/en-US/69127_1.00_PQOSWP

[4] AMD. 2026. AMD uProf User Guide. Retrieved August 8, 2026 from https://docs.amd.com/r/en-US/57368-uProf-user-guide

[5] Arm. 2022. Memory System Resource Partitioning and Monitoring (MPAM), for A-profile Architecture. Retrieved August 8, 2026 from https://support.arm.com/documentation/ddi0598/db

[6] Arm. 2023. Arm Statistical Profiling Extension: Performance Analysis Methodology. Retrieved August 8, 2026 from https://support.arm.com/documentation/109429/0100/

[7] Daniel S. Berger, Daniel Ernst, Huaicheng Li, Pantea Zardoshti, Monish Shah, Samir Rajadnya, Scott Lee, Lisa Hsu, Ishwar Agarwal, Mark D. Hill, and Ricardo Bianchini. 2023. Design Tradeoffs in CXL-Based Memory Pools for Public Cloud Platforms. *IEEE Micro Special Issue on Emerging System Interconnects* 43, 2 (2023). https://doi.org/10.1109/MM.2023.3241586

[8] Ruobing Chen, Haosen Shi, Yusen Li, Xiaoguang Liu, and Gang Wang. 2023. OLPart: Online Learning Based Resource Partitioning for Colocating Multiple Latency-Critical Jobs on Commodity Computers. In *Proceedings of the 18th European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/3552326.3567490

[9] Shuang Chen, Christina Delimitrou, and José F. Martínez. 2019. PARTIES: QoS-Aware Resource Partitioning for Multiple Interactive Services. In *Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3297858.3304005

[10] Yuan Chou, Brian Fahs, and Santosh Abraham. 2004. Microarchitecture Optimizations for Exploiting Memory-Level Parallelism. In *Proceedings of the 31st ACM/IEEE International Symposium on Computer Architecture (ISCA)*. https://doi.org/10.1109/ISCA.2004.1310765

[11] Jonathan Corbet. 2024. Better Support for Locally-attached-memory Tiering. Retrieved August 8, 2026 from https://lwn.net/Articles/974126/

[12] CXL Consortium. 2024. Compute Express Link. Retrieved August 8, 2026 from https://www.computeexpresslink.org

[13] Debendra Das Sharma, Robert Blankenship, and Daniel Berger. 2024. An Introduction to the Compute Express Link (CXL) Interconnect. *ACM Comput. Surv.* 56, 11 (July 2024). https://doi.org/10.1145/3669900

[14] Subramanya R. Dulloor, Amitabha Roy, Zheguang Zhao, Narayanan Sundaram, Nadathur Satish, Rajesh Sankaran, Jeff Jackson, and Karsten Schwan. 2016. Data Tiering in Heterogeneous Memory Systems. In *Proceedings of the 11th European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/2901318.2901344

[15] Padmapriya Duraisamy, Wei Xu, Scott Hare, Ravi Rajwar, David Culler, Zhiyi Xu, Jianing Fan, Christopher Kennelly, Bill McCloskey, Danijela Mijailovic, Brian Morris, Chiranjit Mukherjee, Jingliang Ren, Greg Thelen, Paul Turner, Carlos Villavieja, Parthasarathy Ranganathan, and Amin Vahdat. 2023. Towards an Adaptable Systems Architecture for Memory Tiering at Warehouse-Scale. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3582016.3582031

[16] GAP Benchmark Suite. 2021. GAP Benchmark Suite. Retrieved August 8, 2026 from https://github.com/sbeamer/gapbs.git

[17] Neha Gholkar, Jovan Stojkovic, Hasan Al Maruf, Gregory Price, Prakash Chauhan, Hiral Patel, Cedric Van Goethem, Kiran Vemuri, Kishore Sriadibhatla, Kalyan Subramanian, Shobhit Kanaujia, Chunqiang Tang, and Abhishek Dhanotia. 2026. Vistara: Making CXL Real—Full Path from ASIC Design and OS Support to Hyperscale Deployment. In *Proceedings of the 53rd ACM/IEEE International Symposium on Computer Architecture (ISCA)*. https://doi.org/10.1109/ISCA66397.2026.00061

[18] Juncheng Gu, Youngmoon Lee, Yiwen Zhang, Mosharaf Chowdhury, and Kang G. Shin. 2017. Efficient Memory Disaggregation with Infiniswap. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*.

[19] Zhiyuan Guo, Zijian He, and Yiying Zhang. 2023. Mira: A Program-Behavior-Guided Far Memory System. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/3600006.3613157

[20] Hamid Hadian, Jinshu Liu, Hanchen Xu, Hansen Idden, and Huaicheng Li. 2026. PACT: A Criticality-First Design for Tiered Memory. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3779212.3790198

[21] hnswlib. [n. d.]. hnswlib: Header-Only C++/Python Library for Fast Approximate Nearest Neighbors. Retrieved August 19, 2026 from https://github.com/nmslib/hnswlib

[22] Junliang Hu, Zhisheng Hu, Chun-Feng Wu, and Ming-Chang Yang. 2025. Demeter: A Scalable and Elastic Tiered Memory Solution for Virtualized Cloud via Guest Delegation. In *Proceedings of the 31st ACM Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/3731569.3764801

[23] Gangqi Huang, Heiner Litz, and Yuanchao Xu. 2026. PIPM: Partial and Incremental Page Migration for Multi-Host CXL Disaggregated Shared Memory. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3779212.3790203

[24] Intel. 2017. Intel Xeon Processor Scalable Memory Family Uncore Performance Monitoring Reference Manual. Retrieved August 8, 2026 from https://www.intel.com/content/www/us/en/content-details/671389/intel-xeon-processor-scalable-memory-family-uncore-performance-monitoring-reference-manual.html

[25] Intel. 2021. 3rd Gen Intel Xeon Processor Scalable Family,

Codename Ice Lake, Uncore Performance Monitoring Reference Manual. Retrieved August 8, 2026 from https://cdrdv2-public.intel.com/679093/639778%20ICX%20UPG%20v1.pdf

[26] Intel. 2024. 4th Gen Intel Xeon Scalable Processor XCC (Codename Sapphire Rapids) Uncore Performance Monitoring Guide. Retrieved August 8, 2026 from https://cdrdv2.intel.com/v1/dl/getContent/639667?fileName=639667-SPR_XCC_UPG_Guide-Rev_001.pdf

[27] Intel. 2024. 5th Gen Intel Xeon Scalable Processor XCC (Codename Emerald Rapids) Uncore Performance Monitoring Guide. Retrieved August 8, 2026 from https://cdrdv2-public.intel.com/817509/817509-EMR_XCC_UPG_Guide-Rev_001.pdf

[28] Intel. 2024. Intel 64 and IA-32 Architectures Software Developer Manuals. Retrieved August 8, 2026 from https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html

[29] Intel. 2024. Timed Process Event-Based Sampling (TPEBS). Retrieved August 8, 2026 from https://www.intel.com/content/www/us/en/developer/articles/technical/timed-process-event-based-sampling-tpebs.html

[30] Intel. 2025. Intel Memory Latency Checker (Intel MLC). Retrieved August 8, 2026 from https://www.intel.com/content/www/us/en/download/736633/intel-memory-latency-checker-intel-mlc.html

[31] Intel. 2026. Intel Architecture Instruction Set Extensions and Future Features Programming Reference, Section 9.2: PEBS Load Latency and Store Latency Facility. Retrieved August 8, 2026 from https://www.intel.com/content/www/us/en/content-details/922690/intel-architecture-instruction-set-extensions-programming-reference.html

[32] Intel. [n. d.]. Intel Performance Monitoring Events. Retrieved August 8, 2026 from https://perfmon-events.intel.com/

[33] Intel. [n. d.]. Intel Resource Director Technology (Intel RDT) Framework. Retrieved August 8, 2026 from https://www.intel.com/content/www/us/en/architecture-and-technology/resource-director-technology.html

[34] Intel. [n. d.]. Intel TMA Metric: L2_Parallel_Requests. Retrieved August 8, 2026 from https://perfmon-events.intel.com/

[35] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011). https://doi.org/10.1109/TPAMI.2010.57

[36] Minho Kim, Seonggyu Han, Gyeongseo Park, and Daehoon Kim. 2025. MTAT: Adaptive Fast Memory Management for Co-located Latency-Critical Workloads in Tiered Memory System. In *Proceedings of the 26th International Middleware Conference (Middleware)*. https://doi.org/10.1145/3721462.3770767

[37] Andres Lagar-Cavilla, Junwhan Ahn, Suleiman Souhlal, Neha Agarwal, Radoslaw Burny, Shakeel Butt, Jichuan Chang, Ashwin Chaugule, Nan Deng, Junaid Shahid, Greg Thelen, Kamil Adam Yurtsever, Yu Zhao, and Parthasarathy Ranganathan. 2019. Software-Defined Far Memory in Warehouse-Scale Computers. In *Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3297858.3304053

[38] Taehyung Lee, Sumit Kumar Monga, Changwoo Min, and Young Ik Eom. 2023. Memtis: Efficient Memory Tiering with Dynamic Page Classification and Page Size Determination. In *Proceedings of the 29th ACM Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/3600006.3613167

[39] Huaicheng Li, Daniel S. Berger, Lisa Hsu, Daniel Ernst, Pantea Zardoshti, Stanko Novakovic, Monish Shah, Samir Rajadnya, Scott Lee, Ishwar Agarwal, Mark D. Hill, Marcus Fontoura, and Ricardo Bianchini. 2023. Pond: CXL-Based Memory Pooling Systems for Cloud Platforms. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3575693.3578835

[40] Shihang Li, Matthew Giordano, Tushar Garg, Rohan Kadekodi, Daniel S. Berger, Baris Kasikci, Thomas Anderson, and Simon Peter. 2026. Finding NEMO: Nimble and Expressive Memory Observability. In *Proceedings of the 20th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[41] Linux Kernel Documentation. [n. d.]. User Interface for Resource Control feature (resctrl). Retrieved August 8, 2026 from https://docs.kernel.org/filesystems/resctrl.html

[42] Jinshu Liu, Hamid Hadian, Yuyue Wang, Daniel S. Berger, Marie Nguyen, Xun Jian, Sam H. Noh, and Huaicheng Li. 2025. Systematic CXL Memory Characterization and Performance Analysis at Scale. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3676641.3715987

[43] Jinshu Liu, Hamid Hadian, Hanchen Xu, Daniel S. Berger, and Huaicheng Li. 2024. Dissecting CXL Memory Performance at Scale: Analysis, Modeling, and Optimization. https://doi.org/10.48550/arXiv.2409.14317

[44] Jinshu Liu, Hamid Hadian, Hanchen Xu, and Huaicheng Li. 2025. Tiered Memory Management Beyond Hotness. In *Proceedings of the 19th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[45] Jinshu Liu, Hanchen Xu, Daniel S. Berger, Marcos K. Aguilera, and Huaicheng Li. 2026. Performance Predictability in Heterogeneous Memory. In *Proceedings of the 31st ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3779212.3790201

[46] David Lo, Liqun Cheng, Rama Govindaraju, Parthasarathy Ranganathan, and Christos Kozyrakis. 2015. Heracles: Improving Resource Efficiency at Scale. In *Proceedings of the 42nd ACM/IEEE International Symposium on Computer Architecture (ISCA)*. https://doi.org/10.1145/2749469.2749475

[47] Jiaheng Lu, Yiwen Zhang, Hasan Al Maruf, Minseo Park, Yunxuan Tang, Fan Lai, and Mosharaf Chowdhury. 2024. Mercury: QoS-Aware Tiered Memory System. https://doi.org/10.48550/arXiv.2412.08938

[48] Zhihong Luo, Sam Son, Sylvia Ratnasamy, and Scott Shenker. 2024. Harvesting Memory-bound CPU Stall Cycles in Software with MSH. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[49] LWN.net. 2025. Enable core PMU for DMR and NVL. Retrieved August 8, 2026 from https://lwn.net/Articles/1047256/

[50] Yu A. Malkov and D. A. Yashunin. 2020. Efficient and Robust

Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42, 4 (2020). https://doi.org/10.1109/TPAMI.2018.2889473

[51] Shunyu Mao, Jiajun Luo, Yixin Li, Jiapeng Zhou, Weidong Zhang, Zheng Liu, Teng Ma, and Shuwen Deng. 2025. CXL-Interplay: Unraveling and Characterizing CXL Interference in Modern Computer Systems. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*. https://doi.org/10.1109/DAC63849.2025.11132607

[52] Jason Mars, Lingjia Tang, Robert Hundt, Kevin Skadron, and Mary Lou Soffa. 2011. Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-Locations. In *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. https://doi.org/10.1145/2155620.2155650

[53] Hasan Al Maruf and Mosharaf Chowdhury. 2020. Effectively Prefetching Remote Memory with Leap. In *Proceedings of the 2020 USENIX Annual Technical Conference (ATC)*.

[54] Hasan Al Maruf, Hao Wang, Abhishek Dhanotia, Johannes Weiner, Niket Agarwal, Pallab Bhattacharya, Chris Petersen, Mosharaf Chowdhury, Shobhit Kanaujia, and Prakash Chauhan. 2023. TPP: Transparent Page Placement for CXL-Enabled Tiered Memory. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3582016.3582063

[55] Masstree. 2023. Masstree mttest Benchmark. Retrieved August 8, 2026 from https://github.com/kohler/masstree-beta

[56] Memcached. [n. d.]. Memcached: A Distributed Memory Object Caching System. Retrieved August 8, 2026 from https://memcached.org

[57] Meta. [n. d.]. The CacheLib Caching Engine. Retrieved August 8, 2026 from https://cachelib.org

[58] Microsoft. 2025. Azure delivers the first cloud VM with Intel Xeon 6 and CXL memory - now in Private Preview. Retrieved August 8, 2026 from https://techcommunity.microsoft.com/blog/sapapplications/azure-delivers-the-first-cloud-vm-with-intel-xeon-6-and-cxl-memory---now-in-priv/4470067

[59] Onur Mutlu and Thomas Moscibroda. 2007. Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors. In *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. https://doi.org/10.1109/MICRO.2007.21

[60] Jinsu Park, Seongbeom Park, and Woongki Baek. 2019. CoPart: Coordinated Partitioning of Last-Level Cache and Memory Bandwidth for Fairness-Aware Workload Consolidation on Commodity Servers. In *Proceedings of the 14th European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/3302424.3303963

[61] SeongJae Park. 2020. Introduce Data Access MONitor (DAMON). Retrieved August 8, 2026 from https://lwn.net/Articles/813108/

[62] SeongJae Park, Yunjae Lee, and Heon Y. Yeom. 2019. Profiling Dynamic Data Access Patterns with Controlled Overhead and Quality. In *Proceedings of the 22nd International Middleware Conference (Middleware)*. https://doi.org/10.1145/3366626.3368125

[63] Moinuddin K. Qureshi, Daniel N. Lynch, Onur Mutlu, and Yale N. Patt. 2006. A Case for MLP-Aware Cache Replacement. In *Proceedings of the 33rd ACM/IEEE International Symposium on Computer Architecture (ISCA)*. https://doi.org/10.1109/ISCA.2006.5

[64] Amanda Raybuck, Tim Stamler, Wei Zhang, Mattan Erez, and Simon Peter. 2021. HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/3477132.3483550

[65] Amanda Raybuck, Wei Zhang, Kayvan Mansoorshahi, Aditya K Kamath, Mattan Erez, and Simon Peter. 2023. MaxMem: Colocation and Performance for Big Data Applications on Tiered Main Memory Servers. https://doi.org/10.48550/arXiv.2312.00647

[66] Redis Ltd. [n. d.]. Redis. Retrieved August 8, 2026 from https://redis.io

[67] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. 2020. AIFM: High-Performance, Application-Integrated Far Memory. In *Proceedings of the 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[68] Sai Sha, Chuandong Li, Yingwei Luo, Xiaolin Wang, and Zhenlin Wang. 2023. vTMM: Tiered Memory Management for Virtual Machines. In *Proceedings of the 18th European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/3552326.3587449

[69] Kevin Song, Jiacheng Yang, Zixuan Wang, Jishen Zhao, Sihang Liu, and Gennady Pekhimenko. 2025. HybridTier: an Adaptive and Lightweight CXL-Memory Tiering System. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3676642.3736119

[70] Standard Performance Evaluation Corporation. 2017. SPEC CPU 2017. Retrieved August 8, 2026 from https://www.spec.org/cpu2017

[71] Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Manabi Khan, and Onur Mutlu. 2015. The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory. In *Proceedings of the 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. https://doi.org/10.1145/2830772.2830803

[72] Lavanya Subramanian, Vivek Seshadri, Yoongu Kim, Ben Jaiyen, and Onur Mutlu. 2013. MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems. In *Proceedings of the 19th IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. https://doi.org/10.1109/HPCA.2013.6522356

[73] Yan Sun, Jongyul Kim, Zeduo Yu, Jiyuan Zhang, Siyuan Chai, Michael Jaemin Kim, Hwayong Nam, Jaehyun Park, Eojin Na, Yifan Yuan, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2025. M5: Mastering Page Migration and Memory Management for CXL-based Tiered Memory Systems. In *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3676641.3711999

[74] Yan Sun, Yifan Yuan, Zeduo Yu, Reese Kuper, Chihun Song, Jinghan Huang, Houxiang Ji, Siddharth Agarwal, Jiaqi Lou, Ipoom Jeong, Ren Wang, Jung Ho Ahn, Tianyin Xu, and Nam Sung Kim. 2023. Demystifying CXL Memory with Genuine CXL-Ready Systems and Devices. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. https://doi.org/10.1145/3613424.3614256

[75] Wenda Tang, Senbo Fu, Yutao Ke, Qian Peng, and Feng Gao. 2022. Themis: Fair Memory Subsystem Resource Sharing with Differentiated QoS in Public Clouds. In *Proceedings of the 51st International Conference on Parallel Processing (ICPP)*. https://doi.org/10.1145/3545008.3545064

[76] Wenda Tang, Yiduo Wang, Yanwen Wang, and Jie Wu. 2025. Leave No One Behind: Fair and Efficient Tiered Memory Management for Multi-Applications. In *Proceedings of the 54th International Conference on Parallel Processing (ICPP)*. https://doi.org/10.1145/3754598.3754626

[77] Yupeng Tang, Ping Zhou, Wenhui Zhang, Henry Hu, Qirui Yang, Hao Xiang, Tongping Liu, Jiaxin Shan, Ruoyun Huang, Cheng Zhao, Cheng Chen, Hui Zhang, Fei Liu, Shuai Zhang, Xiaoning Ding, and Jianjun Chen. 2024. Exploring Performance and Cost Optimization with ASIC-Based CXL Memory. In *Proceedings of the 19th European Conference on Computer Systems (EuroSys)*. https://doi.org/10.1145/3627703.3650061

[78] Chandrahas Tirumalasetty and Narasimha Reddy Annapareddy. 2024. Contention Aware DRAM Caching for CXL-Enabled Pooled Memory. In *Proceedings of the 10th International Symposium on Memory Systems (MEMSYS)*. https://doi.org/10.1145/3695794.3695808

[79] TrendForce. 2026. AI Server Demand to Drive Memory Contract Price Increases in 2Q26 as CSPs Secure Supply via Long-Term Agreements. Retrieved August 8, 2026 from https://www.trendforce.com/presscenter/news/20260331-12995.html

[80] Stephen Tu, Wenting Zheng, Eddie Kohler, Barbara Liskov, and Samuel Madden. 2013. Speedy Transactions in Multicore In-Memory Databases. In *Proceedings of the 24th ACM Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/2517349.2522713

[81] VectorDBBench. [n. d.]. VectorDBBench: A Benchmark Tool for VectorDB. Retrieved August 8, 2026 from https://github.com/zilliztech/VectorDBBench

[82] Midhul Vuppalapati and Rachit Agarwal. 2024. Tiered Memory Management: Access Latency is the Key!. In *Proceedings of the 30th ACM Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/3694715.3695968

[83] Johannes Weiner, Niket Agarwal, Dan Schatzberg, Leon Yang, Hao Wang, Blaise Sanouillet, Bikash Sharma, Tejun Heo, Mayank Jain, Chunqiang Tang, and Dimitrios Skarlatos. 2022. TMO: Transparent Memory Offloading in Datacenters. In *Proceedings of the 27th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3503222.3507731

[84] Lingfeng Xiang, Zhen Lin, Weishu Deng, Hui Lu, Jia Rao, Yifan Yuan, and Ren Wang. 2024. NOMAD: Non-Exclusive Memory Tiering via Transactional Page Migration. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[85] XSBench. 2021. XSBench: The Monte Carlo Macroscopic Cross Section Lookup Benchmark. Retrieved August 8, 2026 from https://github.com/ANL-CESAR/XSBench

[86] Zi Yan, Daniel Lustig, David Nellans, and Abhishek Bhattacharjee. 2019. Nimble Page Management for Tiered Memory Systems. In *Proceedings of the 24th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. https://doi.org/10.1145/3297858.3304024

[87] Ahmad Yasin. 2014. A Top-Down Method for Performance Analysis and Counters Architecture. In *Proceedings of the 2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*. https://doi.org/10.1109/ISPASS.2014.6844459

[88] Huang Ying. 2020. AutoNUMA: Optimize Memory Placement for Memory Tiering System. Retrieved August 8, 2026 from https://lwn.net/Articles/835402/

[89] Huang Ying. 2022. NUMA Balancing: Optimize Page Placement for Memory Tiering System. Linux kernel commit `c574bbe91703`. Retrieved August 19, 2026 from https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/commit/?id=c574bbe917036c8968b984c82c7b13194fe5ce98

[90] Heechul Yun, Gang Yao, Rodolfo Pellizzoni, Marco Caccamo, and Lui Sha. 2013. MemGuard: Memory Bandwidth Reservation System for Efficient Performance Isolation in Multi-Core Platforms. In *Proceedings of the 19th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. https://doi.org/10.1109/RTAS.2013.6531079

[91] Adar Zeitak and Adam Morrison. 2021. Cuckoo Trie: Exploiting Memory-Level Parallelism for Efficient DRAM Indexing. In *Proceedings of the 28th ACM Symposium on Operating Systems Principles (SOSP)*. https://doi.org/10.1145/3477132.3483551

[92] Kaiyang Zhao, Neha Gholkar, Hasan Maruf, Abhishek Dhanotia, Johannes Weiner, Gregory Price, Ning Sun, Bhavya Dwivedi, Stuart Clark, and Dimitrios Skarlatos. 2026. Equilibria: Fair Multi-Tenant CXL Memory Tiering at Scale. https://doi.org/10.48550/arXiv.2602.08800

[93] Li Zhao, Ravi R. Iyer, Ramesh Illikkal, Jaideep Moses, Srihari Makineni, and Donald Newell. 2007. CacheScouts: Fine-Grain Monitoring of Shared Caches in CMP Platforms. In *Proceedings of the 16th International Conference on Parallel Architectures and Compilation Techniques (PACT)*.

[94] Yuhong Zhong, Daniel S. Berger, Carl Waldspurger, Ryan Wee, Ishwar Agarwal, Rajat Agarwal, Frank Hady, Karthik Kumar, Mark D. Hill, Mosharaf Chowdhury, and Asaf Cidon. 2024. Managing Memory Tiers with CXL in Virtualized Environments. In *Proceedings of the 18th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*.

[95] Liren Zhu, Liujia Li, Jianyu Wu, Yiming Yao, Zhan Shi, Jie Zhang, Zhenlin Wang, Xiaolin Wang, Yingwei Luo, and Diyu Zhou. 2025. Criticality-Aware Instruction-Centric Bandwidth Partitioning for Data Center Applications. In *Proceedings of the 31st IEEE International Symposium on High-Performance Computer Architecture (HPCA)*. https://doi.org/10.1109/HPCA61900.2025.00042