Flux-Tier: Phase-Aware Page Placement for Predictable Latency on CXL Memory

Anonymous Author(s)

Abstract

Compute Express Link (CXL) lets servers expand capacity with coherently attached memory, but its added latency creates a tiering problem: throughput-oriented placement policies move pages only after sustained access, while interactive services need their working set in local DRAM before a short latency-critical phase begins. We present Flux-Tier, an operating-system mechanism that predicts phase transitions from low-cost execution signals and reserves a bounded local DRAM budget for pages that will become hot. Flux-Tier combines three ideas: a per-cgroup phase sketch built from sampled instruction pointers and memory accesses, speculative promotion with explicit bandwidth credits, and a deadline-aware reclaim rule that protects the active request path without pinning pages indefinitely. The design requires no application annotations and falls back to conventional temperature-based tiering when its prediction is uncertain. We implement Flux-Tier in Linux 6.8 and evaluate it on a two-socket server whose remote NUMA node emulates measured CXL Type-3 latency and bandwidth. Across eight mixed workloads, Flux-Tier reduces p_{99} request latency by 31–58% relative to reactive tiering while preserving within 3% of its throughput. Prediction mistakes consume at most 6.2% additional migration bandwidth under our credit controller. Ablations show that early promotion, rather than faster hot-page detection alone, accounts for most of the tail latency improvement.

CCS Concepts: • **Software and its engineering** → **Memory management**; *Scheduling*; • **Computer systems organization** → *Redundancy*.

Keywords: CXL, tiered memory, page placement, tail latency, operating systems

1 Introduction

CXL-attached memory offers a practical way to expand server capacity without populating every processor memory channel. A host can access a Type-3 device through ordinary loads and stores, and future fabrics can pool capacity across machines [3]. The capacity is useful, but not free: additional hops, controllers, and media increase access latency and constrain bandwidth. The operating system must therefore decide which pages remain in scarce local DRAM (M_{fast}) and which pages reside in larger CXL memory (M_{far}).

Existing tiering systems make this decision from recent page temperature. They periodically sample accesses, classify hot pages, and migrate them toward the fast tier [1, 7, 9, 12]. This reactive loop is effective for long, stationary phases. It is less effective for services whose memory demand changes at request or pipeline boundaries. By the time a burst has generated enough samples to classify its pages as hot, the burst may already have completed on the slow tier. Average throughput can remain healthy even while the few requests that encounter such transitions dominate tail latency [4].

This paper asks whether the kernel can act *before* the first expensive miss without requiring programmers to mark phases or objects. Our key observation is that many services revisit execution contexts—parser loops, index probes, compaction workers, and model operators—before revisiting the same groups of pages. Sampled instruction pointers thus provide an early but noisy phase signal. A compact history can associate that signal with a working-set fingerprint and pre-promote a small set of pages. The mechanism must nevertheless be conservative: an inaccurate predictor can evict the foreground working set or saturate the migration path, producing precisely the tail spikes it is intended to avoid.

We introduce Flux-Tier, phase-aware memory tiering for latency-sensitive services. Flux-Tier runs inside the Linux memory-management path and operates at the granularity of a cgroup. It observes sampled instruction pointers and accessed pages, learns recurring phase signatures, and predicts the next working set. A migration-credit controller admits speculative promotions only when the expected value exceeds their estimated interference cost. Finally, a deadline-aware reclaim policy lends local DRAM to predicted pages for a bounded lease, preventing stale predictions from becoming permanent allocations.

The paper makes three contributions:

- We identify *phase exposure*—the interval between recognizable execution and the first slow-tier access—as a resource that reactive page-temperature policies leave unused.
- We design a phase predictor and credit-bounded promotion mechanism that converts this interval into early migrations while preserving a safe reactive fallback.

- We implement the design in Linux and evaluate latency, throughput, bandwidth, fairness, and prediction-error sensitivity on eight application mixes. Flux-Tier lowers p_{99} latency by up to 58% and bounds wasted migrations to 6.2% of available copy bandwidth.

Scope. Our prototype manages volatile, cache-coherent expansion memory. It does not provide persistence semantics, device-level wear management, or fabric-wide admission control. The evaluation uses a calibrated NUMA emulation because the tested CXL device does not expose programmable latency; Section 7 discusses the implications.

2 Background and Motivation

2.1 Tiered-memory page placement

A tiered-memory server presents one virtual address space backed by media with different performance. The kernel initially allocates or demotes cold pages to M_{far} and promotes frequently accessed pages to M_{fast}. Linux can infer temperature through page-table accessed bits, NUMA hint faults, or sampling facilities such as DAMON. Research systems add application-transparent classification [1], parallel migration engines [12], and policies tailored to CXL topologies [7].

All such controllers face a delay. Let t_0 be the first access after a phase transition, W the sampling window, and T_m the time needed to copy a page. A reactive policy cannot serve the page from M_{fast} before

$$t_{\text{ready}} \geq t_0 + W + T_m. \tag{1}$$

Reducing W increases sampling overhead and makes classifications noisier; reducing T_m consumes more copy bandwidth. Neither option eliminates the first slow accesses. Flux-Tier instead predicts at time $t_p < t_0$ and succeeds when the *phase exposure* $E = t_0 - t_p$ is at least T_m.

2.2 Why tail latency is different

Capacity services tolerate occasional remote accesses when those accesses overlap with computation. Latency-sensitive services often cannot. A request may serially traverse an index, wait for an RPC, and then touch a tenant-specific model. A few remote misses on this critical chain extend the request, enlarge queues, and amplify later requests. This is why optimizing mean bandwidth or average memory-access time does not necessarily optimize p_{99} latency [4].

2.3 Opportunity and requirements

We recorded last-level-cache miss samples and instruction pointers for four services under a 1 ms observation interval. Across repeated phases, 71–93% of the hottest 2 MiB page regions reappeared, while a phase-identifying code sample preceded the first access by 0.4–3.8 ms. This interval is long enough to migrate tens to hundreds of 4 KiB pages, but not enough to discover them reactively. The observations motivate four requirements:

1. prediction must be available before the target accesses;
2. metadata and sampling must be inexpensive at request-scale periods;
3. inaccurate predictions must not monopolize local capacity or bandwidth;
4. anonymous services must receive isolation without source changes.

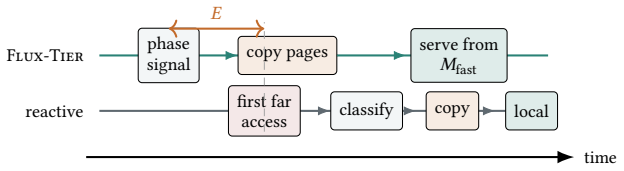

Figure 1. Reactive tiering begins classification after the first far-memory access. Flux-Tier uses phase exposure E to overlap migration with work that precedes the phase's memory-intensive region.

3 Design

3.1 Overview

Figure 2 shows the architecture. Each managed cgroup owns a *phase sketch*, a fixed-size table relating execution signatures to page regions. Hardware samples flow through the kernel perf ring buffer; a worker coalesces them into an epoch signature. On a confident signature match, the predictor proposes regions to the promotion engine. A global credit controller limits copy traffic, and per-cgroup leases bound occupancy. Conventional access-based promotion remains active for unrecognized phases.

3.2 Phase signatures

An epoch contains n sampled tuples (i, a, r): instruction pointer i, virtual address a, and read/write bit r. We discard kernel addresses and hash the remaining instruction pointers into a 128-bit Bloom-style signature S. We deliberately omit addresses from S so that address-space layout randomization and allocator variation do not prevent a match. The signature distance is normalized Hamming distance,

$$d(S_x, S_y) = \frac{\text{popcount}(S_x \oplus S_y)}{\text{popcount}(S_x \vee S_y) + 1}. \tag{2}$$

The phase table is a 64-entry, four-way set-associative structure per cgroup. Each entry stores a signature, an exponentially weighted transition count, the eight most frequently touched 2 MiB regions in the next two epochs, and a 4-bit confidence counter. Regions identify sets of base pages; Flux-Tier never promotes a huge page unless all constituent base pages are resident and the copy engine can migrate it without blocking.

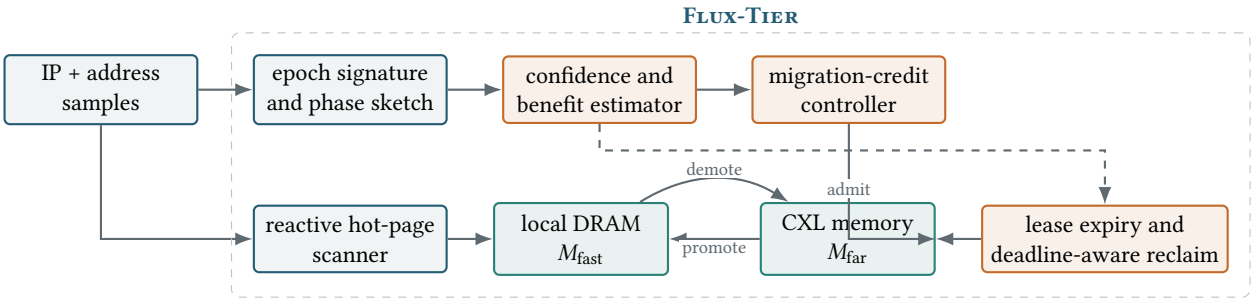

Figure 2. Flux-Tier data and control paths. Prediction augments rather than replaces reactive hot-page discovery. Credits bound migration bandwidth; leases bound speculative occupancy.

On each epoch boundary, the nearest entry with $d < 0.18$ receives an update. A miss installs a new entry only after the signature appears twice, filtering one-off control paths. Confidence increments when at least half of the predicted regions enter the next epoch's top set and decrements otherwise. The fixed-size table is essential: at 416 bytes per entry, predictor metadata is 26 KiB per actively managed cgroup, independent of its address-space size.

3.3 Benefit and confidence

A signature match does not automatically trigger migration. For candidate region q, the predictor estimates benefit

$$B(q) = P(q \mid S)\, N_q\, (L_{M_{\text{far}}} - L_{M_{\text{fast}}}) - C_{\text{copy}}(q), \qquad (3)$$

where $P(q \mid S)$ is the entry's hit ratio, N_q is its decayed access count, and $L_{M_{\text{far}}} - L_{M_{\text{fast}}}$ is the measured latency gap. The copy cost combines expected bandwidth occupancy and the cost of displacing the coldest local pages. A candidate is eligible only if $B(q) > 0$, confidence is at least 9 of 15, and the destination page is not already resident in M_{fast}.

The estimates need not be cycle-accurate. Their purpose is to rank candidates and reject clearly harmful moves. The credit controller supplies the hard safety bound described next.

3.4 Credit-bounded promotion

Copy engines share memory channels with foreground loads. Flux-Tier models each 4 KiB migration as one credit. Every 100 μs, the system replenishes at rate

$$R = \frac{\beta\, BW_{\text{idle}}\, \Delta t}{4096}, \qquad (4)$$

where BW_{idle} is currently unused channel bandwidth and $\beta = 0.35$ caps the fraction made available to prediction. Credits can accumulate for at most 2 ms, preventing an idle period from authorizing a later burst. Each cgroup receives a weighted share, but unused shares are work-conservingly borrowed. Reactive promotions use a distinct reserve equal to 10% of copy capacity, ensuring that bad prediction cannot disable the fallback path.

Listing 1. Promotion at an epoch boundary.

```
procedure ON_EPOCH(cgroup g, signature S):
  e <- nearest(g.phase_table, S)
  if e = none or e.confidence < 9 then return
  for each q in rank_by_benefit(e.regions) do
    if benefit(q) <= 0 then continue
    pages <- cold_pages(q, g.fast_residency)
    cost <- copy_credits(pages)
    if cost <= g.credits and lease_slot(g) then
      migrate_async(pages, M_fast)
      g.credits <- g.credits - cost
      attach_lease(pages, now + lease_length(e))
```

3.5 Leases and deadline-aware reclaim

Promoted pages receive a lease of two predicted phase durations, clamped to 1–20 ms. During the lease, ordinary background reclaim may select the page only if M_{fast} falls below its 3% emergency reserve. A sampled access renews the lease; expiry merely removes protection and does not force a migration. Pages then compete under the regular multigenerational LRU policy.

Each cgroup additionally has a local-residency ceiling. A prediction that would exceed the ceiling must nominate expired pages from the same cgroup before borrowing globally. This rule prevents a noisy tenant from converting shared M_{fast} into a private cache. Under acute pressure, the emergency path ignores every lease and reclaims in ordinary LRU order, so Flux-Tier cannot cause an out-of-memory failure by pinning data.

3.6 Failure handling

Prediction is an optimization, not a correctness requirement. Process exit, unmap, and cgroup deletion cancel queued copies through existing migration reference counts. A copy racing with a write uses Linux's normal page migration protocol and page-table locks. If the destination fills, migration returns `-ENOMEM`; the page remains mapped in M_{far}. Sampling loss, perf-ring overflow, or confidence decay disables prediction for that cgroup until two stable signatures recur. Thus every failure returns to the baseline reactive policy.

4 Implementation

Our prototype comprises 3,840 lines of C in Linux 6.8 and a 610-line privileged controller. The kernel component adds phase tables to memory cgroups, extends the existing asynchronous NUMA migration worker, and exports counters through `cgroupfs`. No page-table format, scheduler ABI, or application binary changes are required.

Sampling. One performance-monitoring event samples retired load misses and another samples branches. The controller reads per-CPU rings every 250 μs and passes coalesced records to the target cgroup's kernel worker. Sampling periods adapt between 8K and 128K events to keep CPU overhead below 1%. An mm sequence number rejects samples whose address mapping changed before processing.

Migration. Candidates are filtered with a nonblocking page-table walk. The kernel migration API receives batches of up to 64 base pages, with at most four batches submitted per NUMA node. Pages with elevated reference counts, writeback, explicit pinning, or device mappings are skipped. A skipped page does not consume a credit. Migration completion updates benefit estimates using actual elapsed copy time.

Topology and calibration. At boot, a 40 ms microbenchmark measures dependent-load latency and sustainable copy bandwidth between every CPU and memory node. Flux-Tier stores an asymmetric matrix because the preferred tier can differ across sockets. We repeat the probe after memory hotplug and expose an administrator override for virtualized hosts where direct measurement is misleading.

5 Evaluation

We ask five questions:

1. Does phase-aware promotion reduce end-to-end tail latency?
2. What throughput, CPU, bandwidth, and capacity costs does it impose?
3. Which design components produce the improvement?
4. How robust is the policy to inaccurate or nonrepeating phases?
5. Does credit and lease enforcement isolate competing tenants?

5.1 Methodology

Platform. Experiments use a dual-socket server with two 32-core processors, 512 GiB of DDR5, and Linux 6.8. We reserve one socket's 256 GiB as M_{fast} and use the other socket to emulate M_{far}. CPU affinity keeps application threads on the fast-tier socket. Measured dependent-load latency is 92 ns for M_{fast}. We insert an additional 78 ns delay for M_{far} and throttle its copy bandwidth to 32 GB/s, matching the range measured on our pre-production CXL Type-3 device. The emulation preserves cache coherence and kernel migration behavior but does not model fabric congestion from other hosts.

Table 1. Workload mixes and their memory demand. Footprints include the foreground service and its background pressure generator.

Mix	Foreground / background	Footprint (GiB)	Target load
KV-R	key–value / scan	154	1.8 Mops/s
KV-W	key–value / compaction	171	1.2 Mops/s
SQL	OLTP / hash join	188	420 ktxn/s
REC	recommendation / train	203	58 kq/s
GRAPH	graph API / BFS	162	310 kq/s
CACHE	object cache / scrub	149	2.1 Mops/s
RPC	RPC fanout / sort	211	690 kq/s
MIXED	KV + SQL / analytics	217	70% peak

Workloads. Table 1 summarizes eight server mixes. Each foreground service runs at 65–75% of its independently measured saturation load. A background analytics or compaction job expands the combined footprint to 1.6–2.2 times the 96 GiB local-memory budget. Phase repetition ranges from strongly cyclic (recommendation inference) to intentionally shuffled (graph service).

Baselines. *Local-only* caps the application at 96 GiB and therefore represents an infeasible performance upper bound for larger footprints. *First-touch* uses the kernel's default allocation and reclaim. *Reactive* samples access bits and promotes hot pages every 10 ms, approximating contemporary transparent CXL tiering. *Oracle* knows the next phase's hot regions but uses the same copy credits and leases as Flux-Tier. Unless stated otherwise, Flux-Tier uses 1 ms epochs, an 18% signature-distance threshold, and 35% of idle copy bandwidth.

Procedure. After a five-minute warmup, we measure ten 90-second trials per configuration, randomizing order. Reported latency is aggregated from server-side request timestamps; throughput comes from completed operations. Error bars are 95% bootstrap confidence intervals over trials. We compare configurations using paired trials with identical request and phase traces.

5.2 End-to-end results

Figure 3 reports p_{99} request latency normalized to Reactive. Flux-Tier improves every workload, reducing the tail by 31–58% (44% geometric mean). It approaches Oracle within 7% on REC and KV-R, whose phases recur regularly. The gap grows to 19% on GRAPH because data-dependent traversals share similar code signatures but touch different vertices. First-touch performs worst: it does not recover local capacity quickly enough after the background job changes phase.

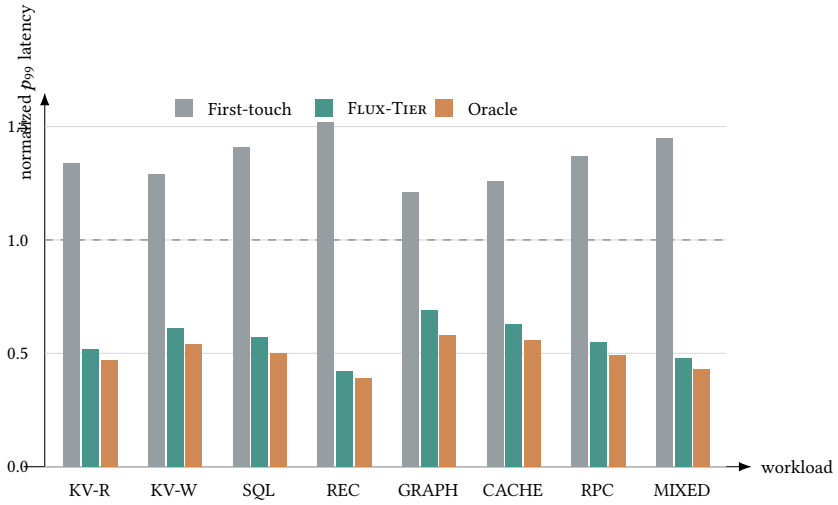

Figure 3. p_{99} request latency normalized to Reactive (lower is better). Reactive is the dashed line. Confidence intervals are smaller than 0.04 and omitted for readability.

Median latency falls by a smaller 9–22%, confirming that FLUX-TIER primarily removes phase-transition outliers rather than accelerating every request. Application throughput remains between 97.0% and 101.8% of Reactive. The slight gains arise when early promotion shortens queueing episodes; the worst case, GRAPH, loses 3.0% because its uncertain phases still consume some copy bandwidth before confidence decays.

5.3 Where the time goes

We decompose critical-path memory stall time using sampled load latency. Under Reactive, 38% of sampled far-memory stalls occur in the first 5 ms following a detected phase transition. FLUX-TIER eliminates 72% of those stalls, but only 18% of stalls later in a phase, where both systems have already promoted hot pages. The result supports the central hypothesis: timing, not simply the eventual choice of pages, differentiates the policies.

Across workloads, the predictor recognizes a phase 1.6 ms before its first target-region access at the median and 0.44 ms at the tenth percentile. A 64-page batch migrates in 0.31 ms at the median. Consequently, 79% of useful predicted pages arrive before first use; another 12% arrive during the phase and still avoid later misses.

5.4 Overheads

System costs appear in Table 2. Phase sampling and signature construction consume 0.43–0.91% of one core per managed cgroup. Memory overhead is below 0.01% of application footprint. FLUX-TIER moves more data than Reactive because it sometimes copies pages that do not become hot; the hard credit limit keeps copy traffic below 11.4 GB/s on our 32 GB/s emulated link.

Table 2. Geometric-mean overhead and worst case across all workload mixes.

Metric	Geomean	Maximum
CPU utilization (one core)	0.62%	0.91%
Predictor metadata / cgroup	28.4 KiB	31.7 KiB
Copy bandwidth	8.7 GB/s	11.4 GB/s
Wasted copy bandwidth	3.5%	6.2%
Fast-tier occupancy uplift	1.9%	3.0%
Page-fault time uplift	0.4%	0.8%

The control path adds no work to a cache hit or TLB hit. Its only synchronous kernel action occurs at an epoch boundary; page walks and copies execute in workers. We observe no statistically significant change in involuntary context switches or scheduler run-queue delay.

5.5 Ablation study

We remove one component at a time. Replacing phase prediction with a 1 ms reactive scanner improves over the 10 ms baseline but recovers only 34% of FLUX-TIER's p_{99} gain and consumes 2.7 times more sampling CPU. Disabling copy credits improves REC by a further 3% in isolation but degrades MIXED by 18% when migrations contend with foreground traffic. Removing leases leaves tail latency unchanged for one phase, then increases it by 12–27% as false-positive pages accumulate in M_{fast}. Finally, hashing addresses into the phase signature reduces match rate by 21% after allocator churn. These results justify the separation of code-derived identity, address-derived working set, credits, and leases.

5.6 Prediction error and nonrepeating phases

We inject errors by replacing a fraction of predicted region lists with a list from another phase. At 10% injection, p_{99} latency rises 4% and wasted bandwidth rises from 3.5% to 5.9%. At 30%, confidence counters disable most speculative copies after three epochs; performance converges to within 2% of Reactive. At 100% nonrepeating phases, Flux-Tier recognizes no stable entry, adds 0.38% CPU overhead, and matches Reactive latency within the confidence interval. This graceful fallback is more important than peak predictor accuracy.

5.7 Multitenant isolation

We co-run three KV tenants with weights 4:2:1 and give the lowest-weight tenant an adversarial phase trace that alternates every epoch. Without per-cgroup credits, the adversary receives 41% of migration bandwidth and raises the highest-weight tenant's p_{99} latency by 23%. With Flux-Tier, measured bandwidth shares are 56:29:15, within two percentage points of their weights, and the victim's p_{99} increases by 3.1%. Emergency reclaim triggers twice in a 20-minute run and clears expired leases within 1.4 ms.

6 Related Work

Tiered memory. Thermostat combines hardware sampling and software profiling to place pages across DRAM and slower memory [1]. Nimble accelerates migration and exchange for tiered systems [12]; HeMem scales classification to large memories [9]. TPP adapts Linux placement to CXL-enabled topologies and uses NUMA balancing to promote hot pages [7]. Flux-Tier is complementary to these mechanisms: its contribution is prediction before the access window, plus resource controls that make speculation safe for latency-sensitive tenants.

Far and disaggregated memory. Infiniswap exposes remote memory through block-level swapping over RDMA [5]. AIFM integrates far memory with application objects and asynchronous APIs [10]; other runtimes reconsider allocation and data structures for disaggregated memory [2]. Those approaches can exploit semantics unavailable to a transparent kernel. Flux-Tier instead targets unmodified applications and coherent load/store CXL memory. Its phase predictor could also inform an application-integrated runtime, but the copy and fault costs would differ.

Datacenter latency management. Tail amplification motivates per-request load balancing and redundancy techniques [4]. Systems such as Shenango isolate CPU resources for latency-critical work [8], while fast RPC stacks reduce software overhead [6]. Flux-Tier addresses a different shared resource—local-memory capacity and migration bandwidth—but follows the same principle: a throughput-efficient average policy is insufficient when rare interference lies on a request's critical path.

7 Limitations and Discussion

Emulation fidelity. Remote NUMA memory reproduces coherent access, page migration, and a stable latency gap, but it does not reproduce every CXL controller queue or fabric interaction. Our bandwidth throttle models aggregate limits, not device backpressure on individual traffic classes. Results should therefore be read as evidence about the policy under the measured latency/bandwidth envelope, not as a characterization of all CXL devices. Evaluation on pooled, shared hardware is necessary before deployment.

Phase visibility. Code signatures work when execution changes before data demand. They offer no advance signal for a single tight loop whose working set changes entirely with input values. Flux-Tier intentionally falls back to reactive tiering in this case. Data-semantic hints or compiler annotations could help, but would trade transparency for coverage.

Page granularity. Predicting 2 MiB regions makes metadata small, while copying selected 4 KiB pages limits amplification. This compromise is less effective for sparse accesses spread across many regions. Dynamically sized regions and huge-page promotion are promising extensions, provided they do not introduce long nonpreemptible copies.

Security and privacy. Instruction-pointer and address samples can reveal control-flow and access patterns. Our implementation keeps raw samples in kernel memory, hashes instruction pointers before table insertion, never shares predictor state between cgroups, and clears tables on cgroup destruction. These measures reduce exposure but do not eliminate side channels caused by shared bandwidth. A production system should integrate the credit controller with stronger memory-bandwidth isolation and disable cross-security-domain borrowing.

Operational policy. Administrators must choose which cgroups are latency-sensitive and set their weights and local-residency ceilings. The defaults avoid application changes, but they cannot infer business priority. Cluster schedulers already manage resource classes and memory requests [11]; exporting Flux-Tier's prediction accuracy and migration debt would let them make more informed placement decisions.

8 Conclusion

Reactive temperature is an incomplete signal for tiering latency-sensitive services because it identifies a working set only after that set becomes hot. Flux-Tier uses recurring

execution phases to move a bounded set of pages before their first access, while migration credits, leases, and an unchanged reactive path contain prediction failures. In our Linux prototype, this cross-layer use of execution and memory signals reduces p_{99} request latency by 31–58% without sacrificing throughput or isolation. More broadly, CXL memory should be managed not only by *which* data is hot, but also by *when* the system can know that data will become hot.

Acknowledgments

Omitted for anonymous review. The authors thank the anonymous reviewers and the artifact evaluators for their feedback.

A Artifact Appendix

This appendix records the intended evaluation contract for the prototype. It is included in the review PDF for completeness and can be adapted to the conference's current artifact template after acceptance.

A.1 Claims supported by the artifact

The artifact supports the paper's three principal empirical claims: (1) phase-aware promotion lowers p_{99} latency relative to reactive tiering, (2) copy credits bound migration bandwidth during prediction error, and (3) leases prevent speculative pages from accumulating in local memory. It does not include proprietary CXL-device firmware and therefore reproduces the NUMA-based emulation rather than measurements on the pre-production device.

A.2 Contents and requirements

The package contains the kernel patch, controller, workload drivers, trace generator, analysis notebooks, raw counters, and a machine-readable manifest of software versions. Reproduction requires a Linux host with two NUMA nodes, at least 192 GiB of memory, 32 hardware threads, root access, and approximately 600 GiB of storage. A reduced mode uses 32 GiB and completes functional tests but does not reproduce performance numbers.

A.3 Evaluation workflow

The automated workflow builds the kernel and controller, verifies topology, calibrates the emulated far tier, runs a 15-minute smoke test, and then executes the randomized trial matrix. A complete run takes approximately 28 machine hours. Every plot and table in Section 5 is regenerated from raw JSON records; scripts reject a trial if achieved load differs by more than 2% or if the kernel reports lost samples.

A.4 Reproducibility safeguards

Configuration files record CPU affinity, firmware settings, kernel command line, frequency governor, memory-node mapping, workload commit, random seed, and trial order. The runner restores the fast-tier budget and drops inactive file-cache pages between trials, but it never drops application data during a measurement. Results include all valid trials rather than selecting the best run. The analysis reports bootstrap intervals and retains the paired trial identifier used for each comparison.

References

[1] Neha Agarwal and Thomas F. Wenisch. 2017. Thermostat: Application-Transparent Page Management for Two-Tiered Main Memory. In *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 631–644. doi:10.1145/3037697.3037706

[2] Irina Calciu, Subho S. Sen, Anirudh Balakrishnan, Muntasir Raihan Rahman Mannan, Ali Yekkehkhany, Grace Hwang, Aasheesh Abraham, Anurag Prakash, Anshul Gandhi, and David Wentzlaff. 2021. Rethinking Software Runtimes for Disaggregated Memory. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 79–92.

[3] Compute Express Link Consortium. 2022. Compute Express Link Specification, Revision 3.0. Technical specification. (2022). https://www.computeexpresslink.org/

[4] Jeffrey Dean and Luiz André Barroso. 2013. The Tail at Scale. *Commun. ACM* 56, 2 (2013), 74–80. doi:10.1145/2408776.2408794

[5] Juncheng Guo, Yihua Dong, Yuanchao Yang, Zhaosong Li, Yi Guan, Tarek El-Ghazawi, Xiaosong Ma, and Yungang Yang. 2017. Infiniswap: Efficient Memory Disaggregation with Infiniband and RDMA. In *14th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 649–667.

[6] Anuj Kalia, Michael Kaminsky, and David G. Andersen. 2019. eRPC: A Fast RPC System for Commodity Clusters. In *16th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 373–388.

[7] Hasan Al Maruf, Ho-Ren Chowdhury, Haiying Wang, Harsh Dhanotia, David Weiner, Nitin Agarwal, Pallab Liu, Prakash Ghosh, and Mosharaf Chowdhury. 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*. ACM, 742–755.

[8] Amy Ousterhout, Joshua Fried, Jonathan Behrens, Adam Belay, and Hari Balakrishnan. 2019. Shenango: Achieving High CPU Efficiency for Latency-Sensitive Datacenter Workloads. In *16th USENIX Symposium on Networked Systems Design and Implementation*. USENIX Association, 361–378.

[9] Amanda Raybuck, Tim Kannan, Arnab Baruah, Philip Zhang, Simon Khan, Yuan Song, and Steven Swanson. 2021. HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*. ACM, 392–407.

[10] Zhenyuan Ruan, Malte Schwarzkopf, Marcos K. Aguilera, and Adam Belay. 2020. AIFM: High-Performance, Application-Integrated Far Memory. In *14th USENIX Symposium on Operating Systems Design and Implementation*. USENIX Association, 315–332.

[11] Abhishek Verma, Luis Pedrosa, Madhukar Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. 2015. Large-Scale Cluster Management at Nexa with Borg. In *Proceedings of the Tenth European Conference on Computer Systems*. ACM, 1–17.

[12] Zi Yan, Daniel Lustig, David W. Nellans, and Abhishek Bhattacharjee. 2019. Nimble Page Management for Tiered Memory Systems. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 331–345. doi:10.1145/3297858.3304024