IDT: Intelligent Data Placement for Multi-tiered Main Memory with Reinforcement Learning

Juneseo Chang[†], Wanju Doh[†], Yaebin Moon[‡],
Eojin Lee[§], and Jung Ho Ahn[†]

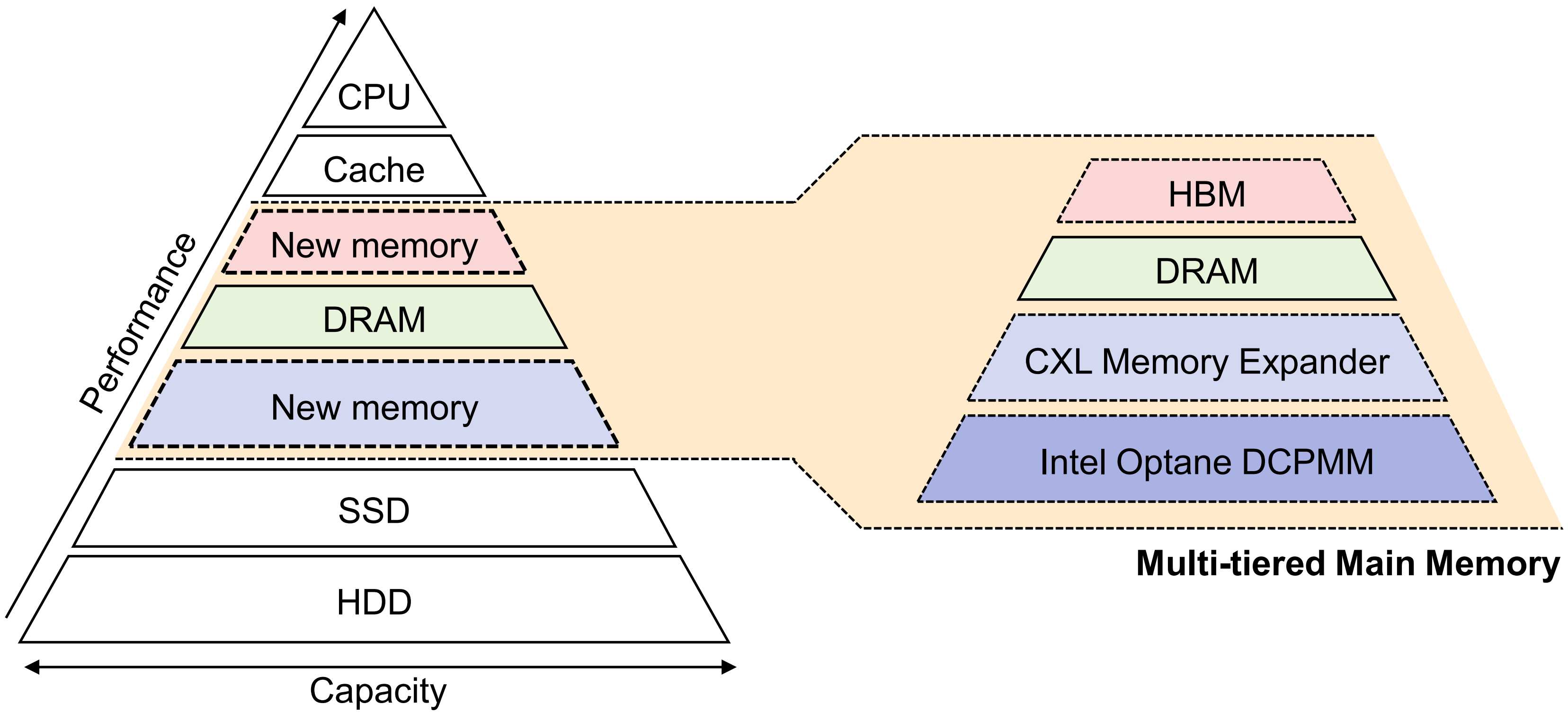
[†]Seoul National University, [‡]Samsung Electronics, [§] Inha University

Presenter: Juneseo Chang (jschang0215@snu.ac.kr)

[‡]This work was done while at Seoul National University

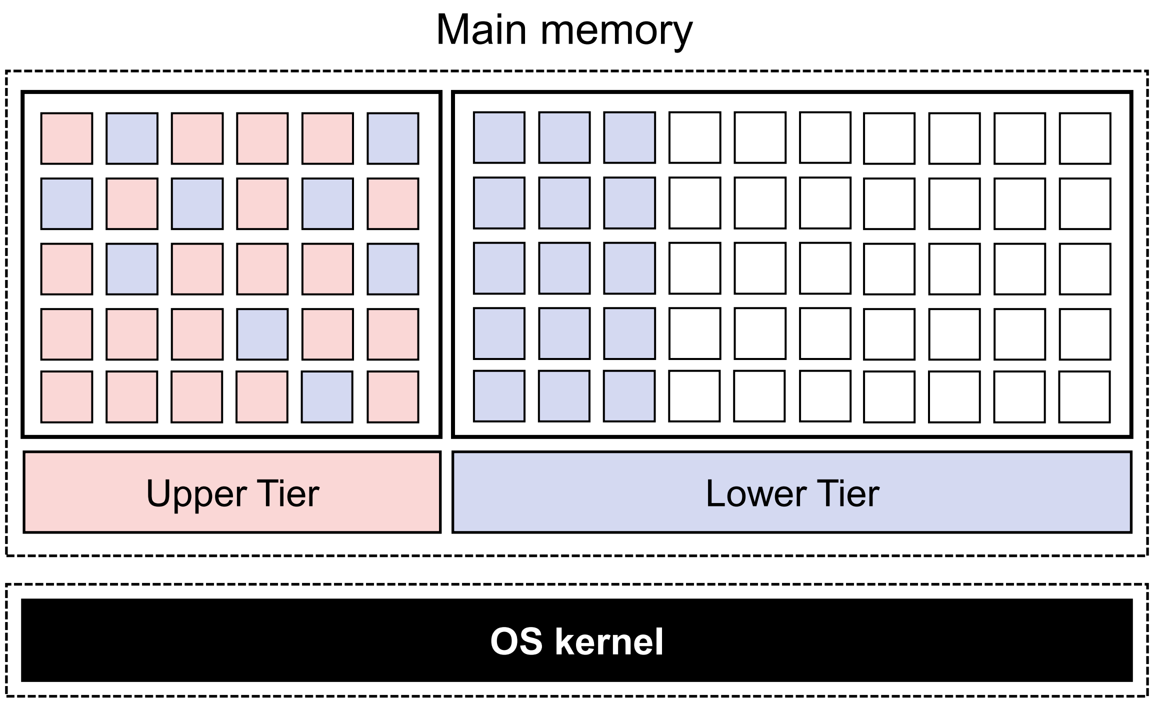
Tiered Memory Systems

- Emerging memory technologies are introducing **multiple tiers** in the **main memory**
  - CXL Memory, HBM-enabled processors, Intel Optane DCPMM, …



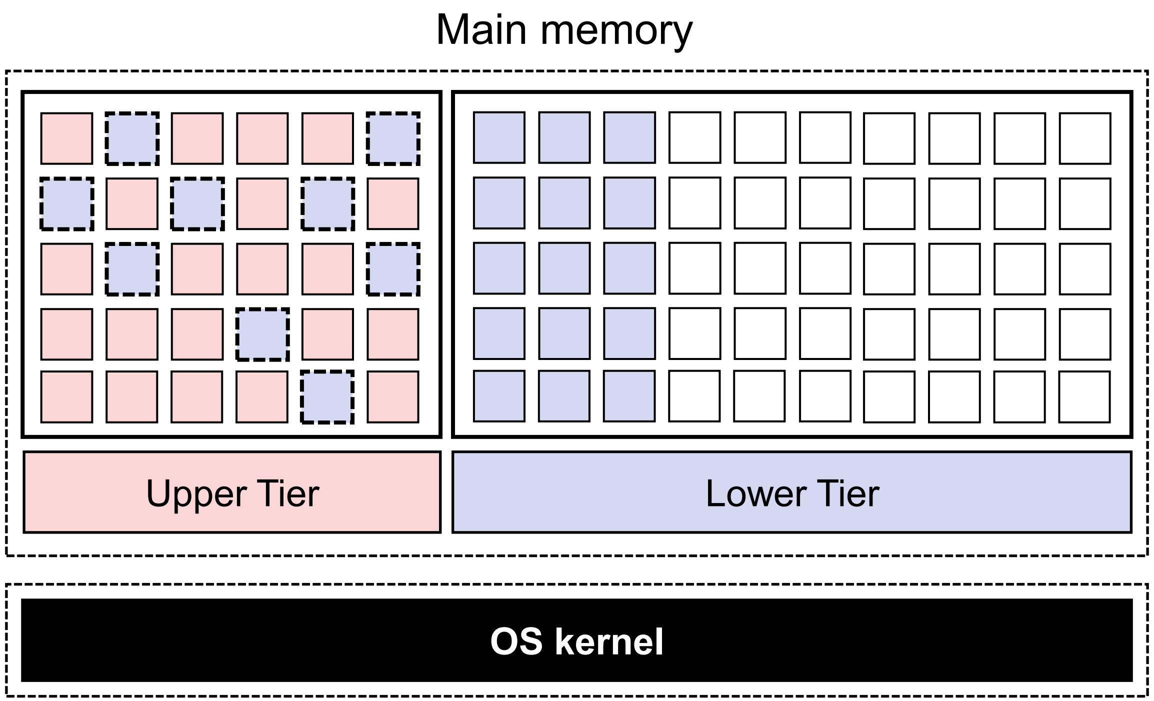
OS-level Tiered Memory Management

- **OS** kernel manages **data placement** across tiers



OS-level Tiered Memory Management

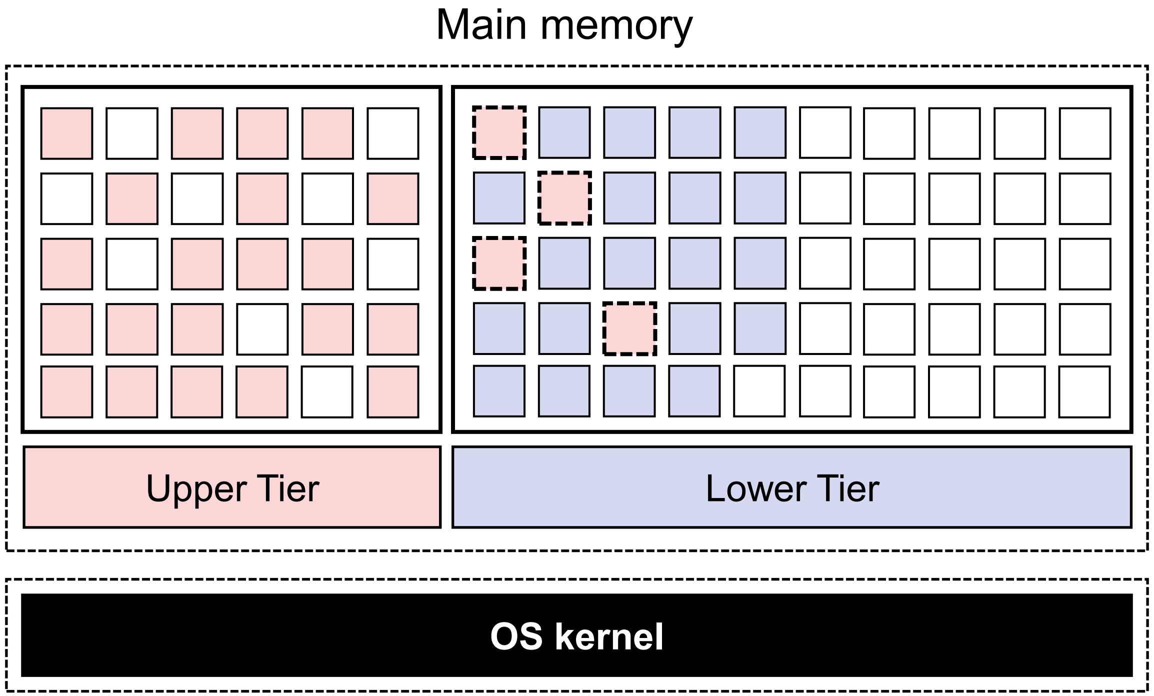
- **OS** kernel manages **data placement** across tiers
- **OS kernel demotes cold** pages to **lower**-tier memory



Accurately **identifying data hotness** and effective **demotion criteria** are necessary!

OS-level Tiered Memory Management

- **OS** kernel manages **data placement** across tiers
- **OS kernel demotes cold** pages to **lower**-tier memory
- **OS kernel promotes hot** pages to **upper**-tier memory



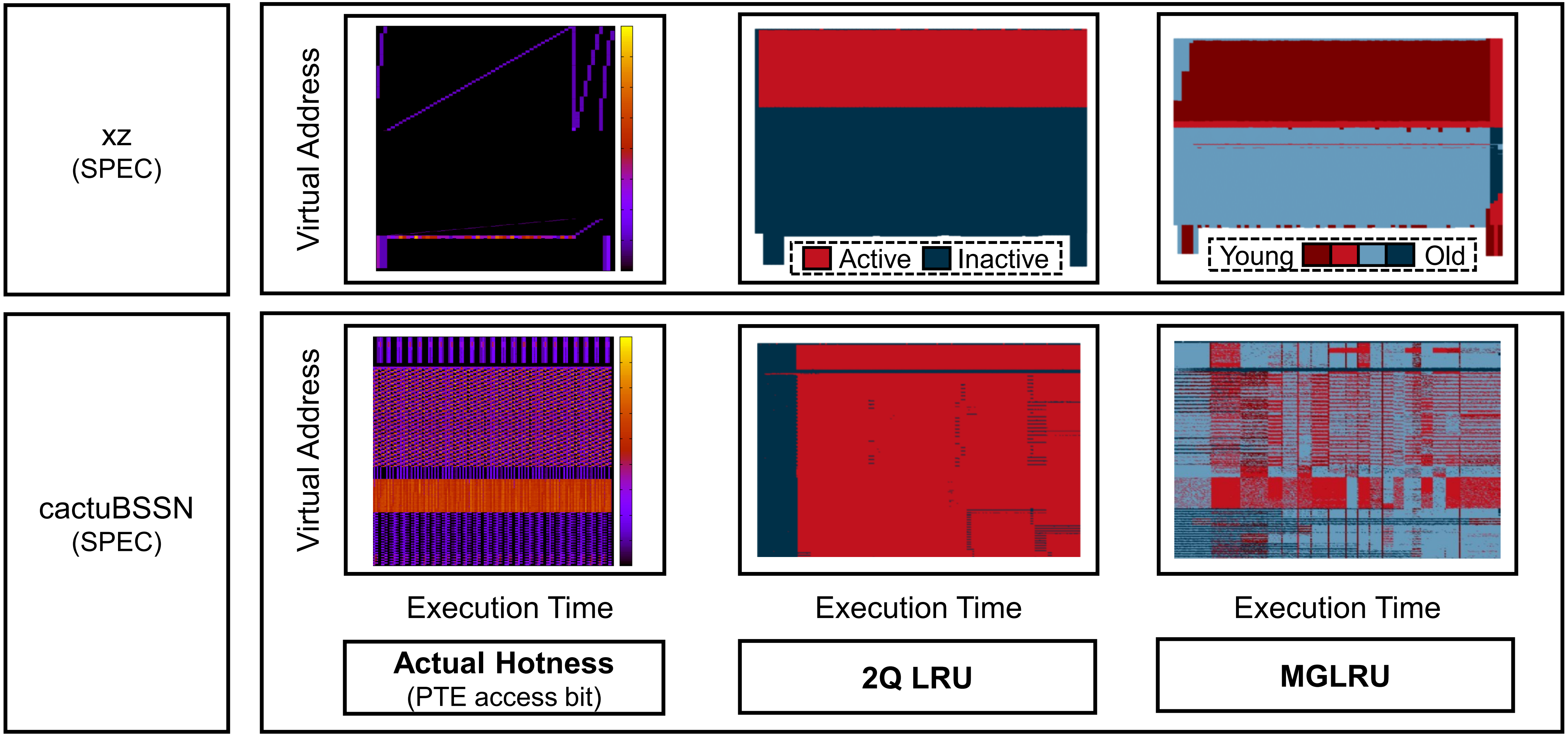
Selecting Demotion Candidates: 2Q LRU and MGLRU

- Effective demotion candidate selection is crucial
  - Impacts **promotion**
  - Incorrectly identifying demotion targets causes **ping-pong** of demotion and promotion
- Prior works used Linux kernel's **active/inactive LRU lists** (**2Q LRU)**
  - Since 2022, **multi-generational LRU lists[1]** (**MGLRU)** for more fine-grained policy

[1] Yu Zhao. 2022. Multigenerational LRU Framework. https://lwn.net/Articles/880393/.

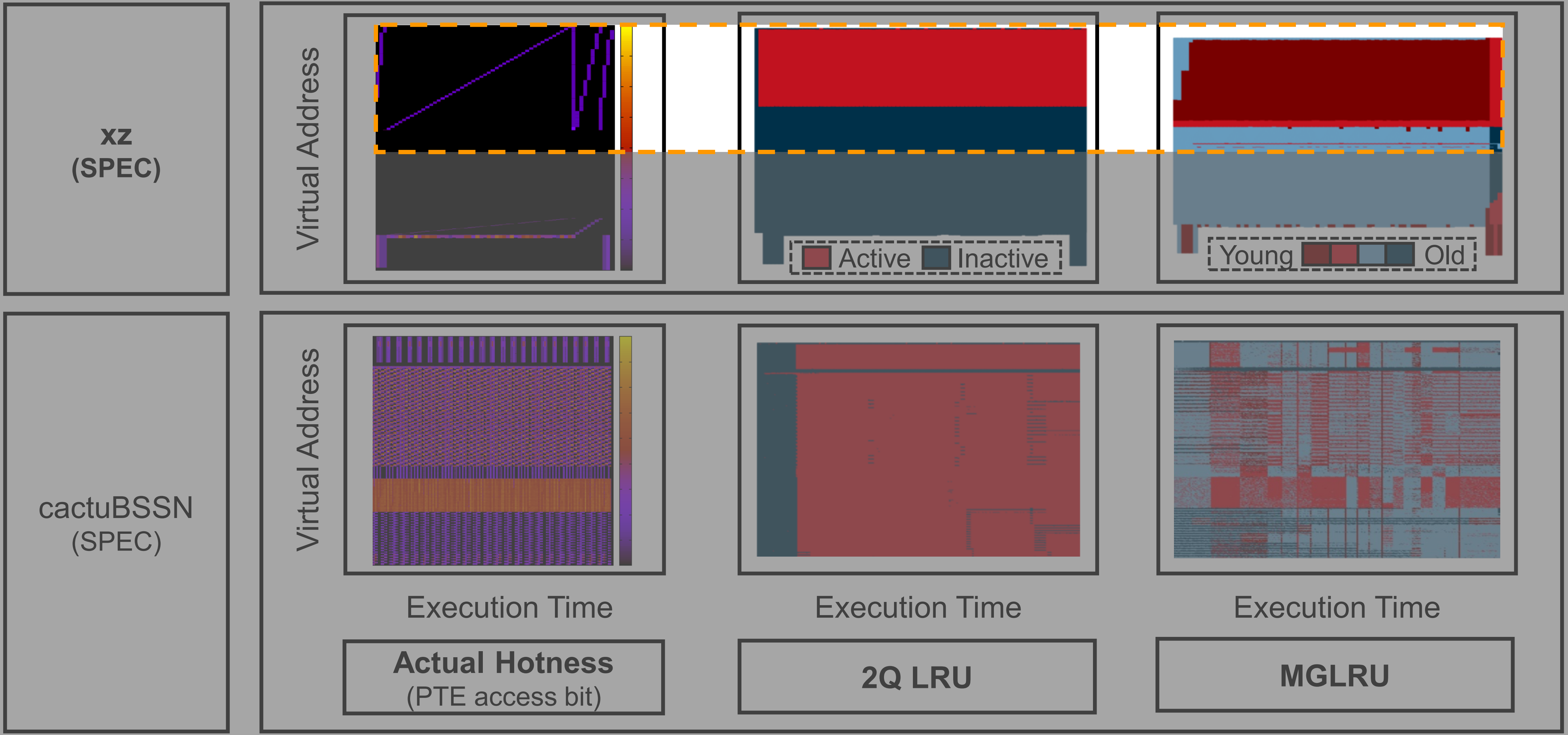
Selecting Demotion Candidates: 2Q LRU and MGLRU

- However, **2Q LRU** and **MGLRU** often deviate from the **actual data hotness** (PTE access bit scanning)



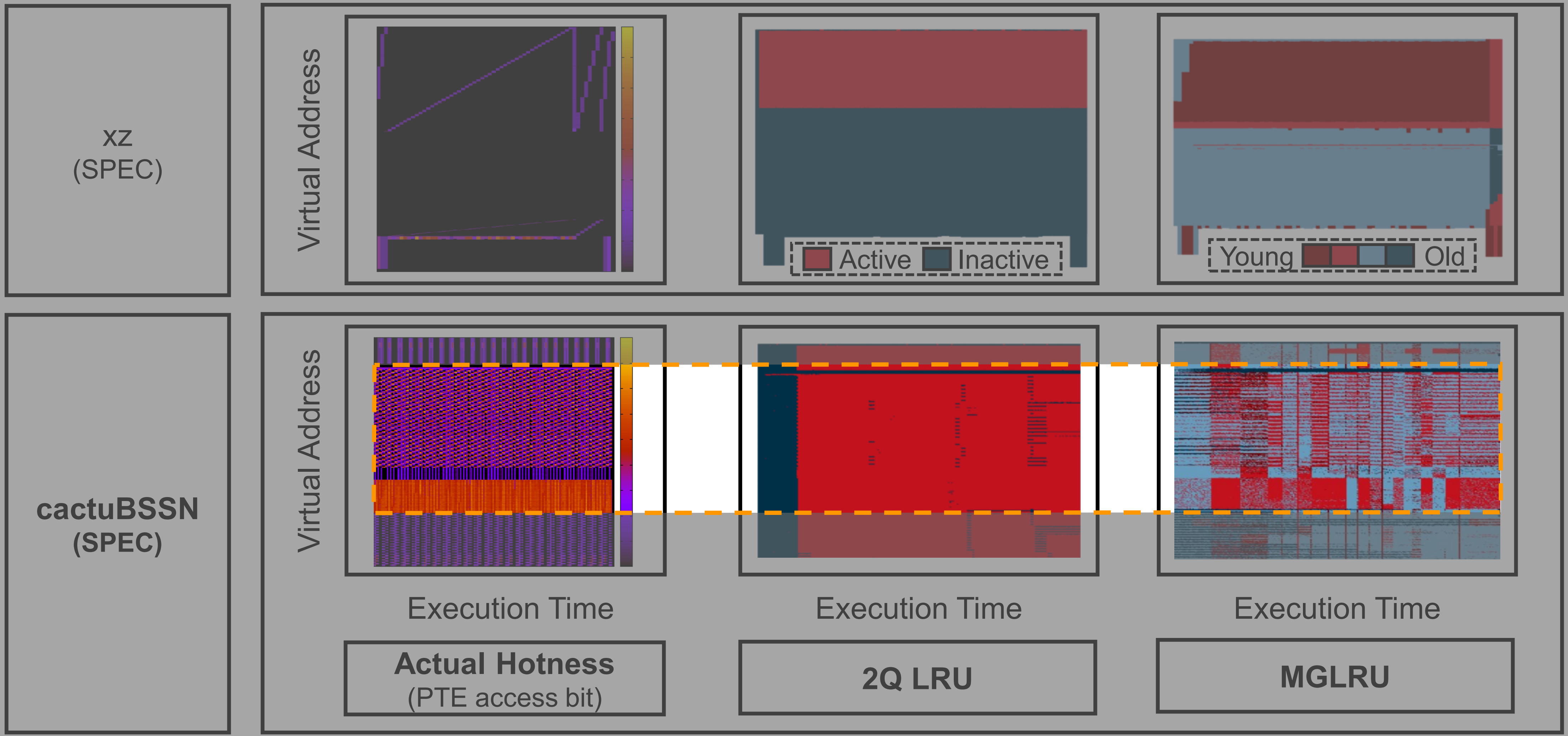
Selecting Demotion Candidates: 2Q LRU and MGLRU

- However, **2Q LRU** and **MGLRU** often deviate from the **actual data hotness** (PTE access bit scanning)



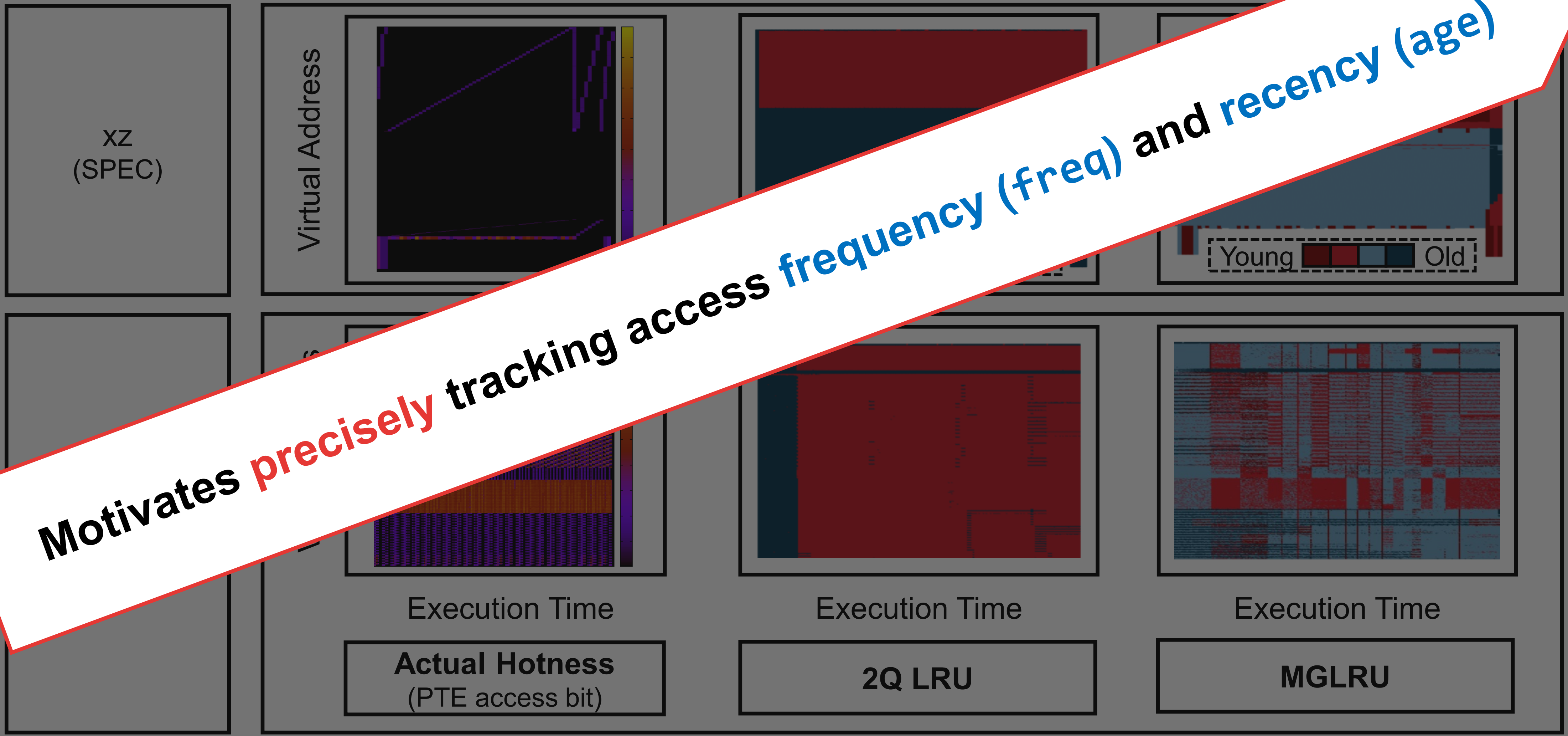
Selecting Demotion Candidates: 2Q LRU and MGLRU

- However, **2Q LRU** and **MGLRU** often deviate from the **actual data hotness** (PTE access bit scanning)



Selecting Demotion Candidates: 2Q LRU and MGLRU

- However, **2Q LRU** and **MGLRU** often deviate from the **actual data hotness** (PTE access bit



Motivates precisely tracking access frequency (freq) and recency (age)

Selecting Demotion Candidates: Using more precise standards

`freq` (frequency)	Most pages are known to be **cold** (not accessed for a few seconds) in memory-intensive workloads[1]
`age` (recency)	Pages with **lower age** are **likely to be used** in the future

⇒

Select pages with **minimum `freq`**

AND

`age > age_thres`

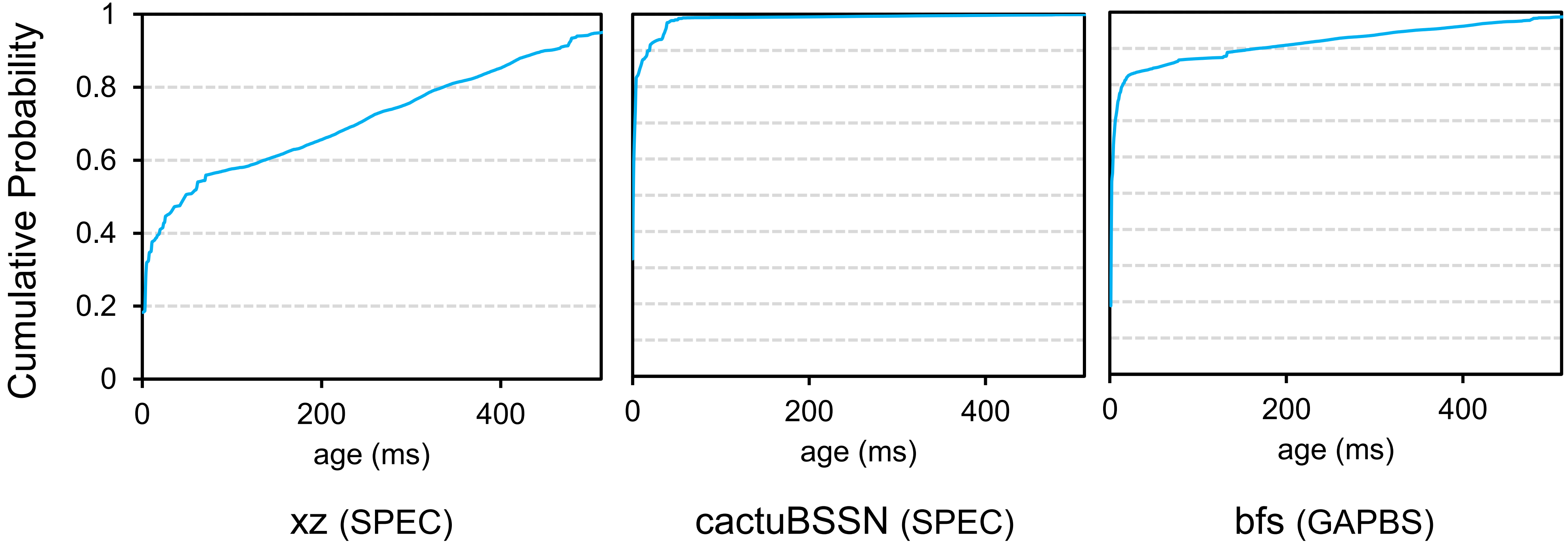
[1] Andres Lagar-Cavilla et al., “Software-Defined Far Memory in Warehouse-Scale Computers,” ASPLOS. 2019

Selecting Demotion Candidates: Using more precise standards

`freq`	**Minimum `freq`**	AND	`age`	`age > age_thres`

Selecting Demotion Candidates: Using more precise standards

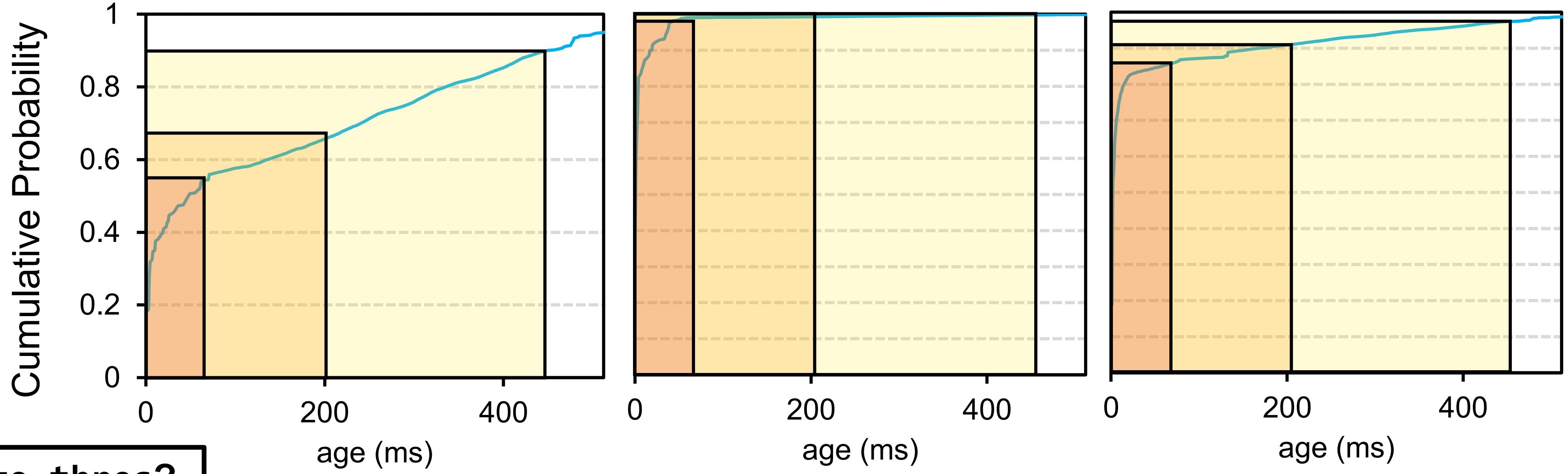
`freq`	Minimum `freq`	AND	`age`	`age > age_thres`



Cumulative probability distribution of accessed page's age varies across workloads

Selecting Demotion Candidates: Using more precise standards

freq	Minimum freq	AND	age	age > age_thres



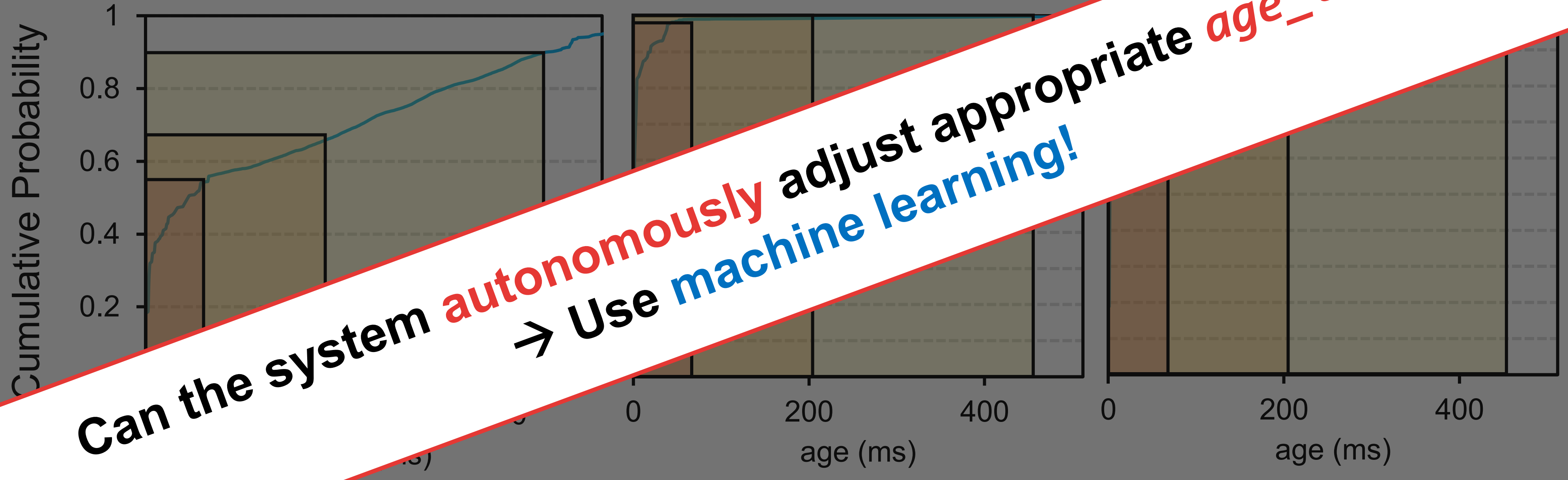
age_thres?

xz (SPEC)　cactuBSSN (SPEC)　bfs (GAPBS)

Cumulative probability distribution of accessed page's age varies across workloads

Selecting Demotion Candidates: Using more precise standards

freq | Minimum freq | AND | age | age >



xz (SPEC) | cactuBSSN (SPEC) | bfs (GAPBS)

Can the system autonomously adjust appropriate *age_thres*?

→ Use machine learning!

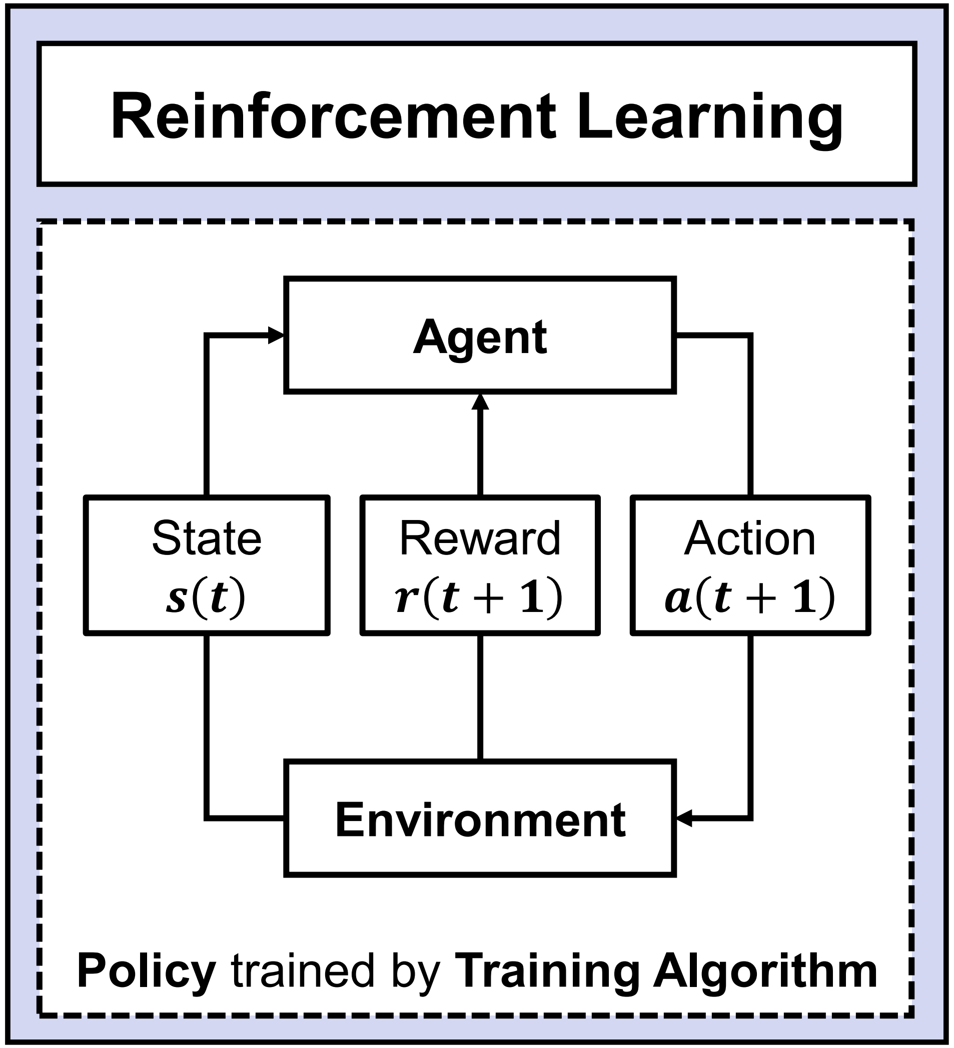
Cumulative probability distribution of accessed page's age varies across workloads

ML for Demotion Policy

Lightweight	Prior **supervised learning** approaches have high **execution time overhead** and **memory usage**
Adaptability	**Adapt to dynamic runtime behavior** with low overhead (without full retraining)
Extensibility	Easily extend to support **multi-tiered memory**

ML for Demotion Policy

Lightweight	Prior **supervised learning** approaches have high **execution time overhead** and **memory usage**
Adaptability	**Adapt to dynamic runtime behavior** with low overhead (without full retraining)
Extensibility	Easily extend to support **multi-tiered memory**



ML for Demotion Policy

Lightweight

Lower overhead than LSTM [1] or Bandit[2] algorithms used in prior works

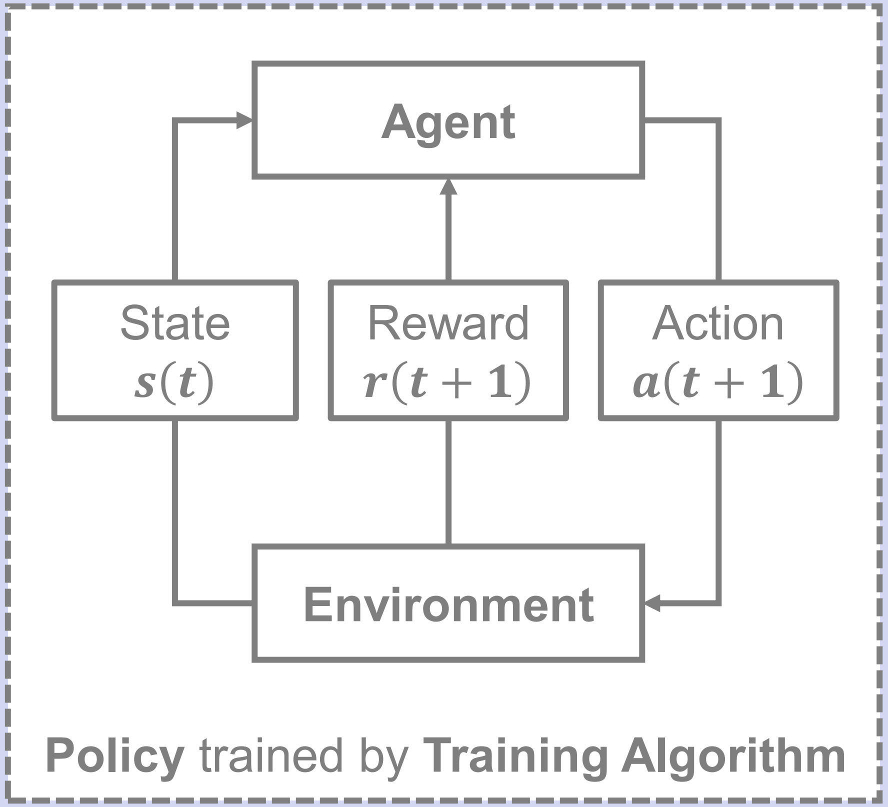
Adaptability

Adapt to dynamic environment

Extensibility

Run inference for each memory tier

Reinforcement Learning



[1] Thaleia Dimitra Doudali et al., “Kleio: A Hybrid Memory Page Scheduler with Machine Intelligence,” HPDC, 2019
[2] Andres Lagar-Cavilla et al., “Software-Defined Far Memory in Warehouse-Scale Computers,” ASPLOS. 2019

ML for Demotion Policy

Lightweight

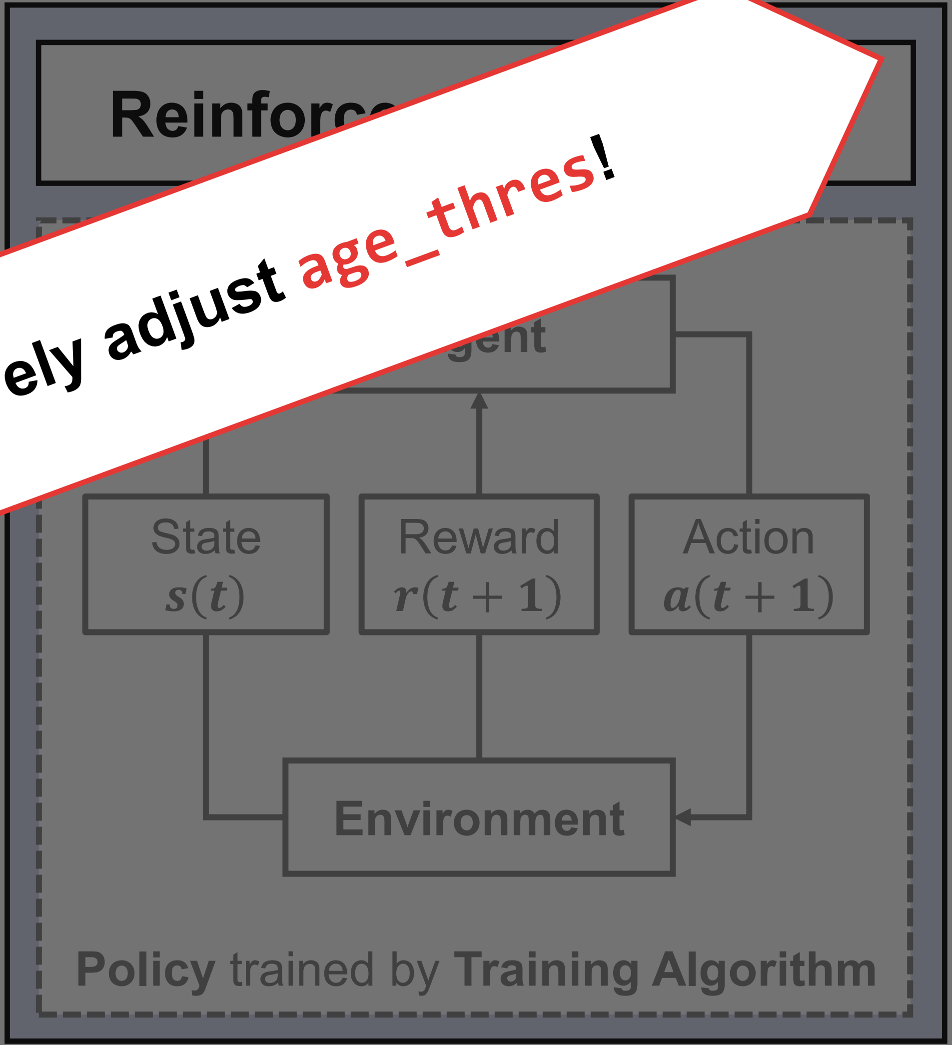
Lower overhead than LSTM [1] or Bandit[2] algorithms used in prior works

Adaptability

Adapt

Run inference for each memory tier

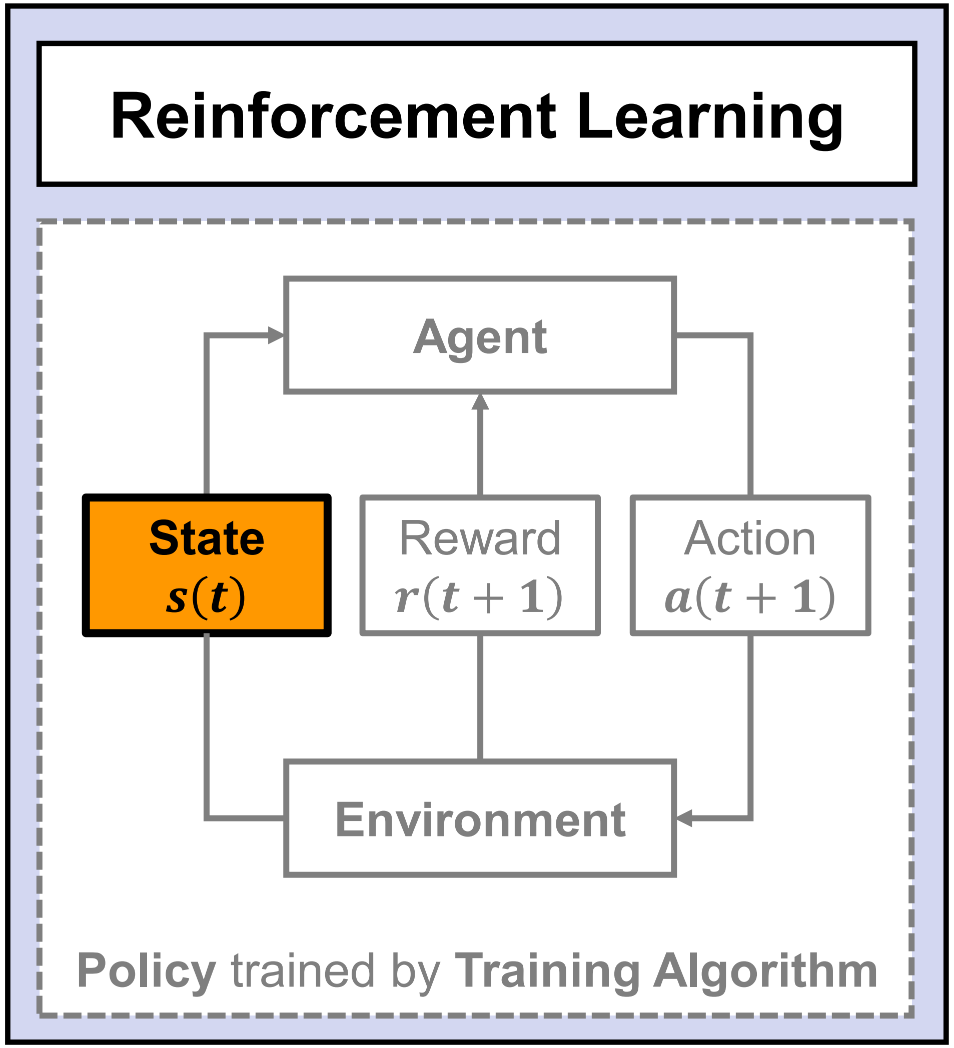
Reinforcement Learning (RL) can effectively adjust age_thres!



[1] Thaleia Dimitra Doudali et al., "Kleio: A Hybrid Memory Page Scheduler with Machine Intelligence," HPDC, 2019
[2] Andres Lagar-Cavilla et al., "Software-Defined Far Memory in Warehouse-Scale Computers," ASPLOS. 2019

ML for Demotion Policy

Lightweight	Lower overhead than LSTM or Bandit algorithms used in prior works
Adaptability	Adapt to dynamic environment
Extensibility	Run inference for each memory tier



ML for Demotion Policy

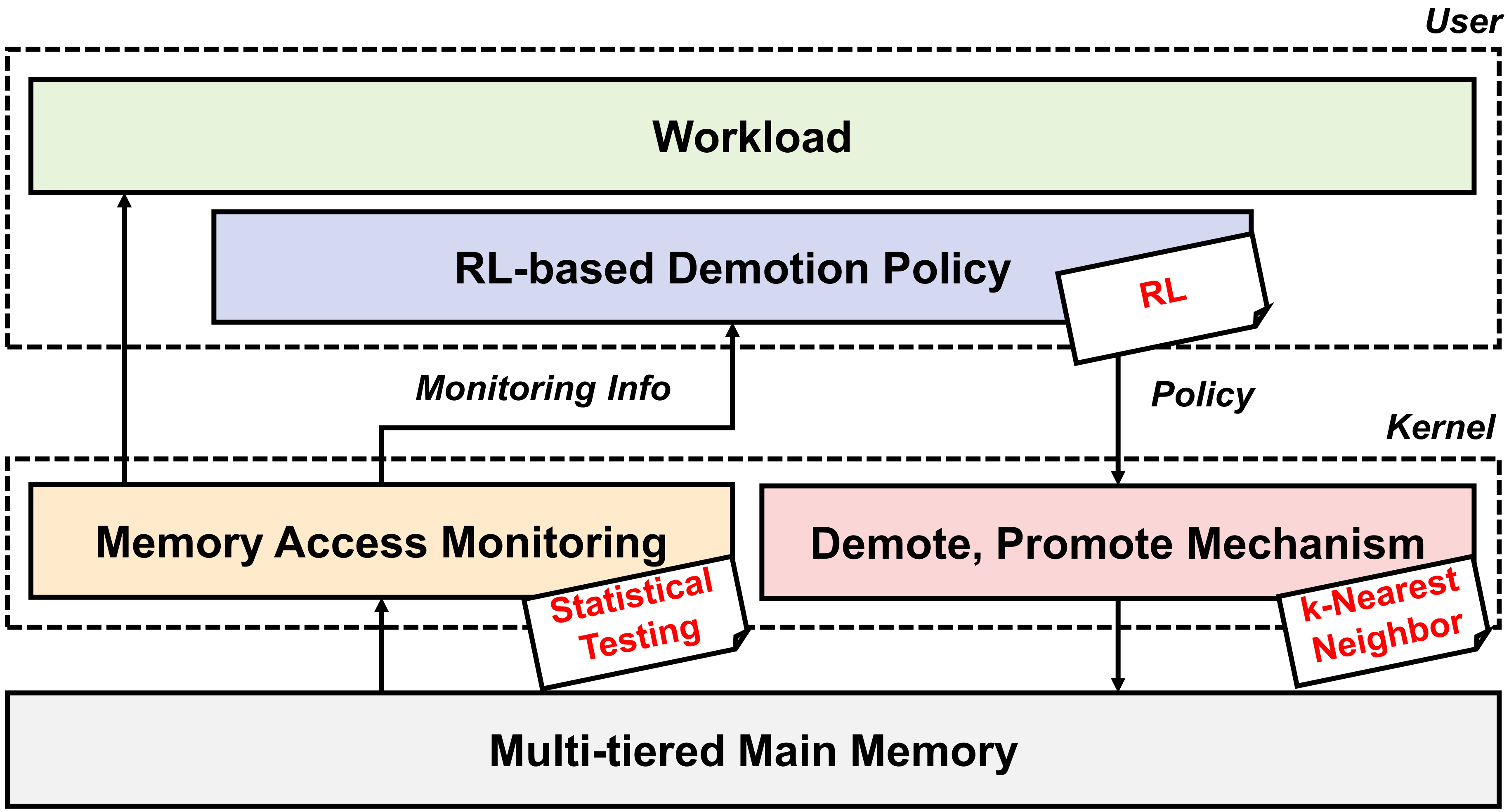
State: **Memory access information of the system**

Page granularity memory access monitoring has a high overhead

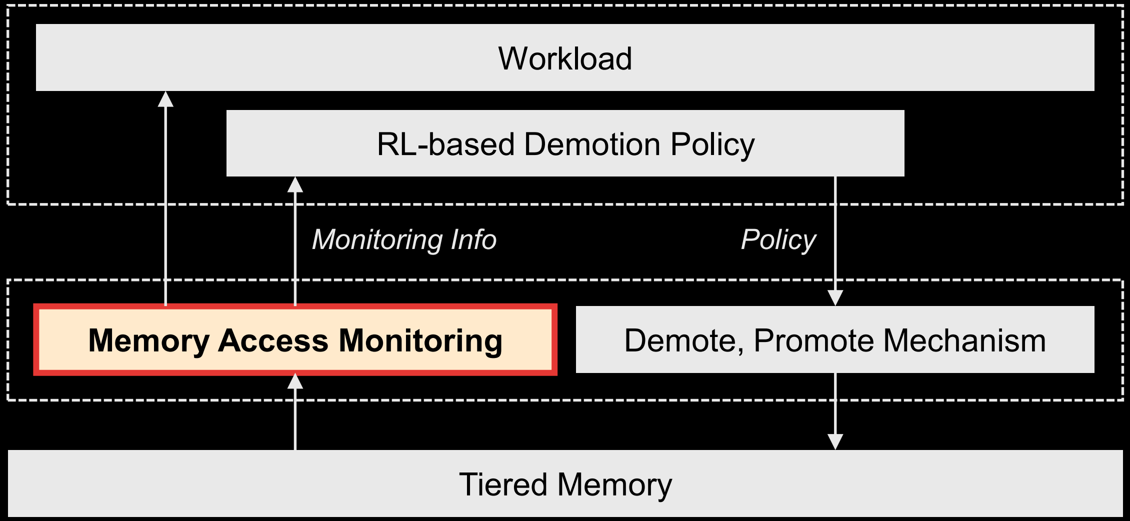
→ Group similar pages with region-granularity monitoring

IDT: Design and Implementation

IDT: Overview



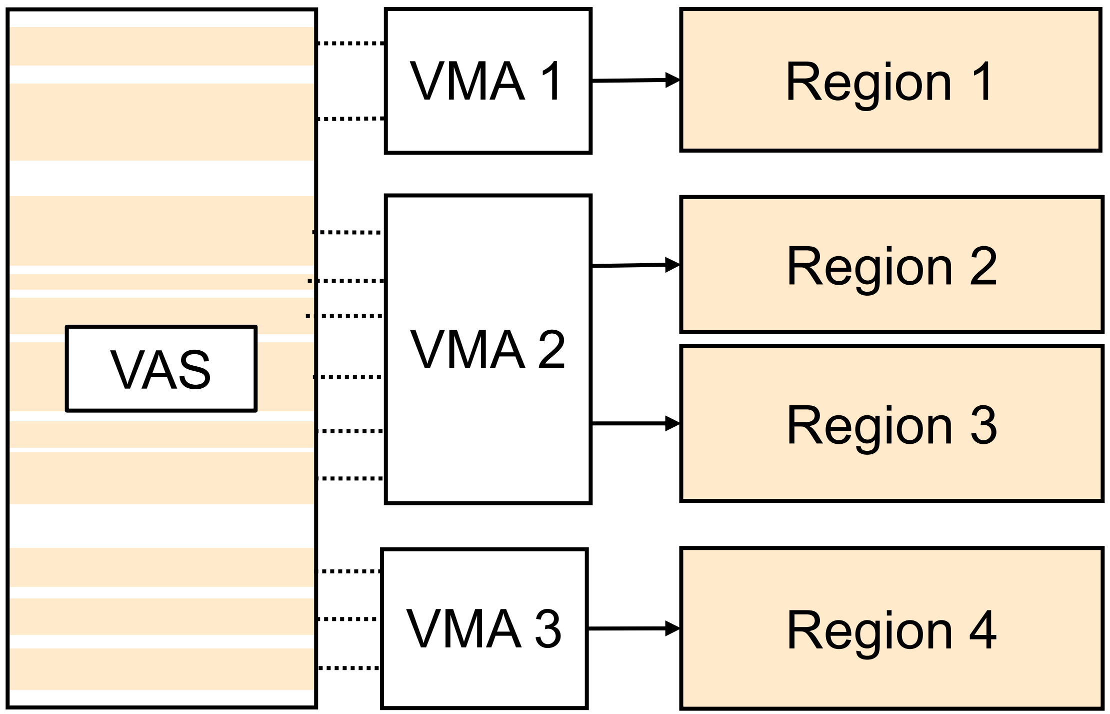
IDT Overview



Memory Access Monitoring

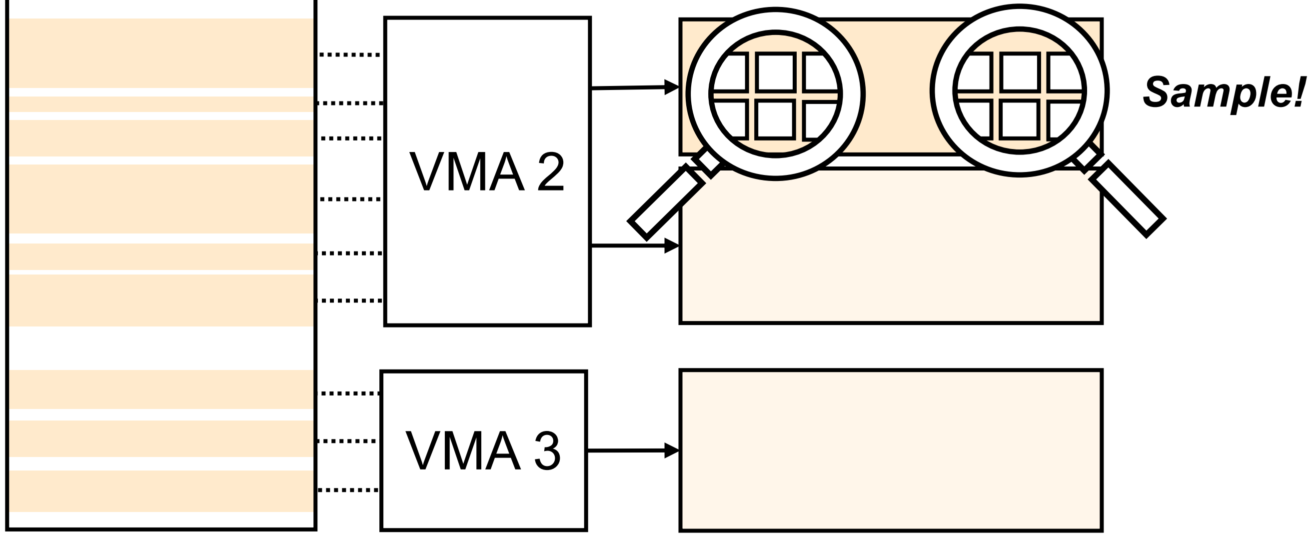
Region-granularity Monitoring

- Monitor **group** of pages with **similar access patterns**
  - **Partition** Virtual Memory Area (VMA) into **regions**



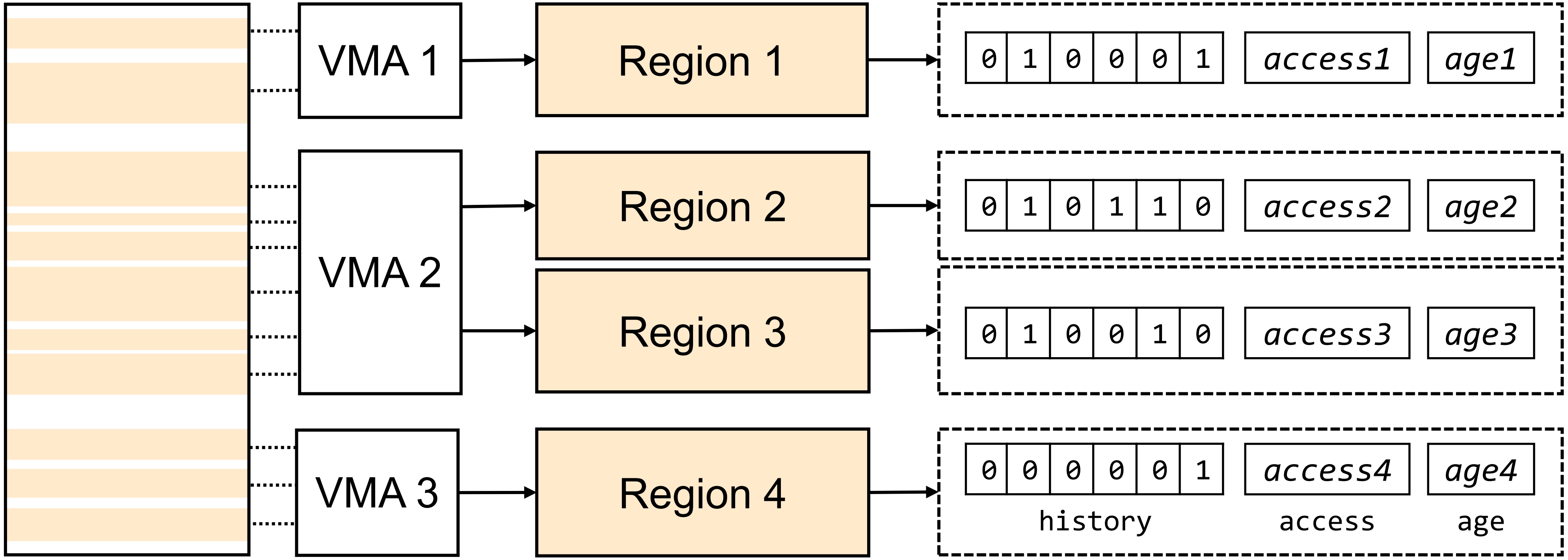
Region-granularity Monitoring

- Monitor **group** of pages with **similar access patterns**
  - **Partition** Virtual Memory Area (VMA) into **regions**
- **Sample 2 pages** at each `sample_interval`



Region-granularity Monitoring

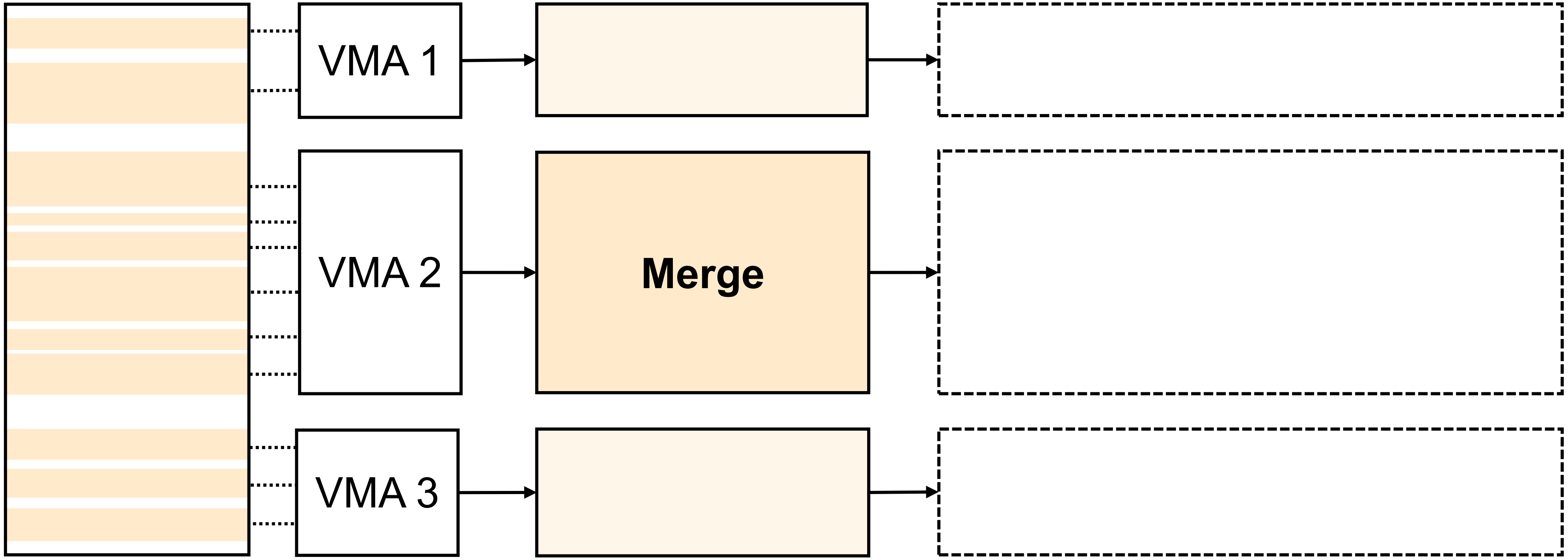
- Monitor **group** of pages with **similar access patterns**
  - **Partition** Virtual Memory Area (VMA) into **regions**
- **Sample 2 pages** at each `sample_interval`
  - Manage **`history`**, **`access`**, **`age`**[1]



[1] SeongJae Park. 2020. DAMON: Data Access Monitor. https://docs.kernel.org/mm/damon/index.html.

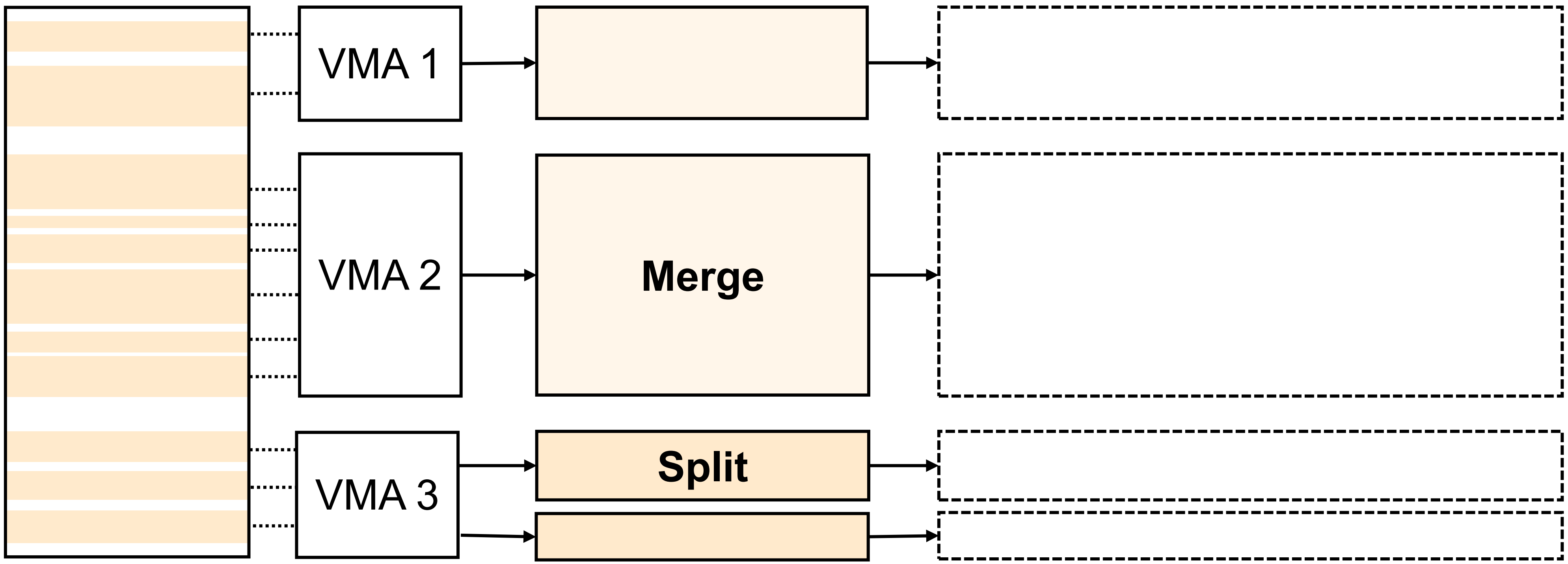
Region Reconfiguration

- **Merge** or **split** adjacent regions for reconfiguration at each `aggregate_interval`
  - **Merge** regions with **similar access patterns** to **reduce** monitoring overhead



Region Reconfiguration

- **Merge** or **split** adjacent regions for reconfiguration at each `aggregate_interval`
  - **Merge** regions with **similar access patterns** to **reduce** monitoring overhead
  - **Split** when pages in a region have **different access patterns**



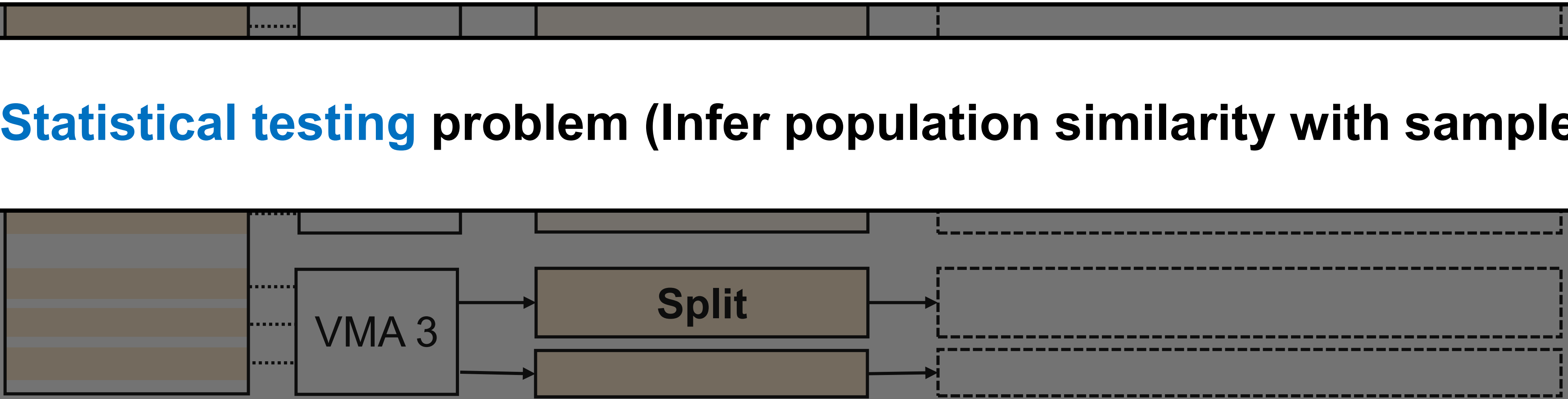
Region Reconfiguration

- **Merge** or **split** adjacent regions for reconfiguration at each `aggregate_interval`

Assume similar pages are grouped in the same region

Sampling page's information determines the similarity of regions

→ Statistical testing problem (Infer population similarity with samples)



Region Reconfiguration: Merge

- Validate the similarity of region's `history` vector by **Fisher's exact test** with a 90% significance level

	Accessed	Not	Total
Region i	a_i	$n - a_i$	n
Region $(i+1)$	a_{i+1}	$n - a_{i+1}$	n

`window size = n`

$$P_{i,i+1} = \frac{\binom{n}{a_i} \times \binom{n}{a_{i+1}}}{\binom{2n}{a_i + a_{i+1}}}$$

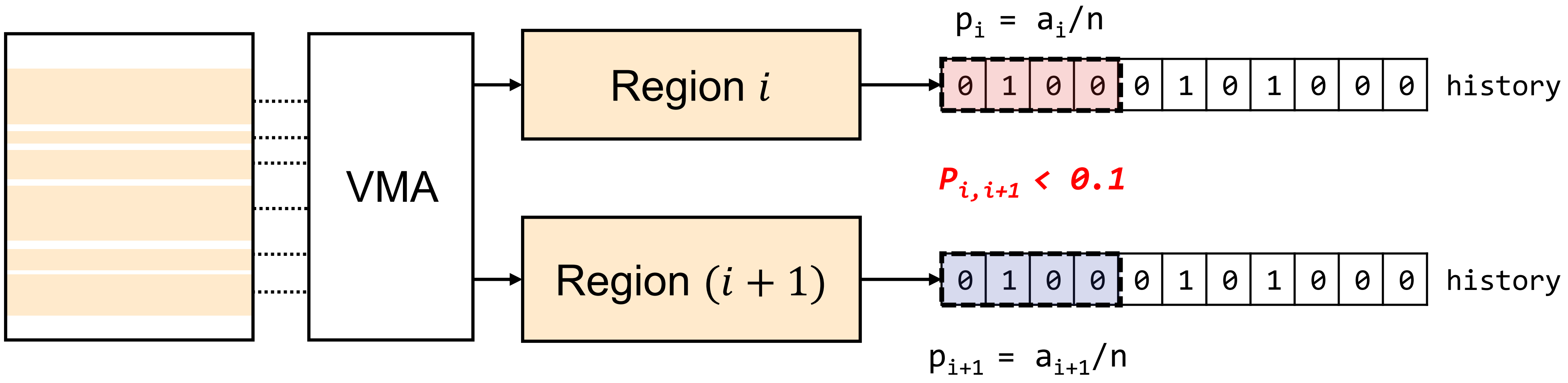
Region Reconfiguration: Merge

- Validate the similarity of region's `history` vector by **Fisher's exact test** with a 90% significance level
- **Sliding window** → Compare the **access ratio** of each region's window

	Accessed	Not	Total
Region i	a_i	$n - a_i$	n
Region $(i+1)$	a_{i+1}	$n - a_{i+1}$	n

`window size = n`

$$P_{i,i+1} = \frac{\binom{n}{a_i} \times \binom{n}{a_{i+1}}}{\binom{2n}{a_i + a_{i+1}}}$$



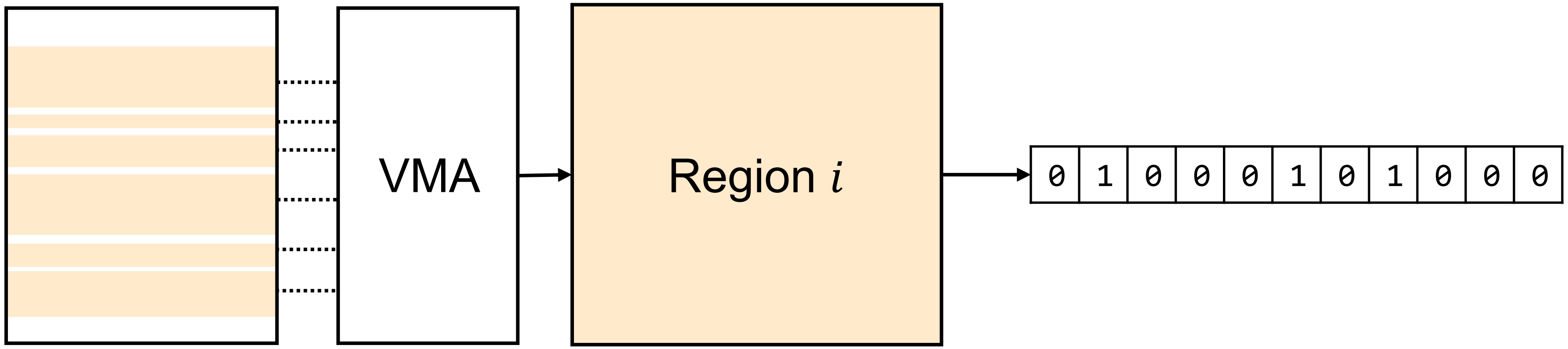
Region Reconfiguration: Merge

- Validate the similarity of region's `history` vector by **Fisher's exact test** with a 90% significance level
- **Sliding window** → Compare the **access ratio** of each region's window
  - If every window yields a *similar access ratio* → **Merge**

	Accessed	Not	Total
Region i	a_i	$n - a_i$	n
Region $(i+1)$	a_{i+1}	$n - a_{i+1}$	n

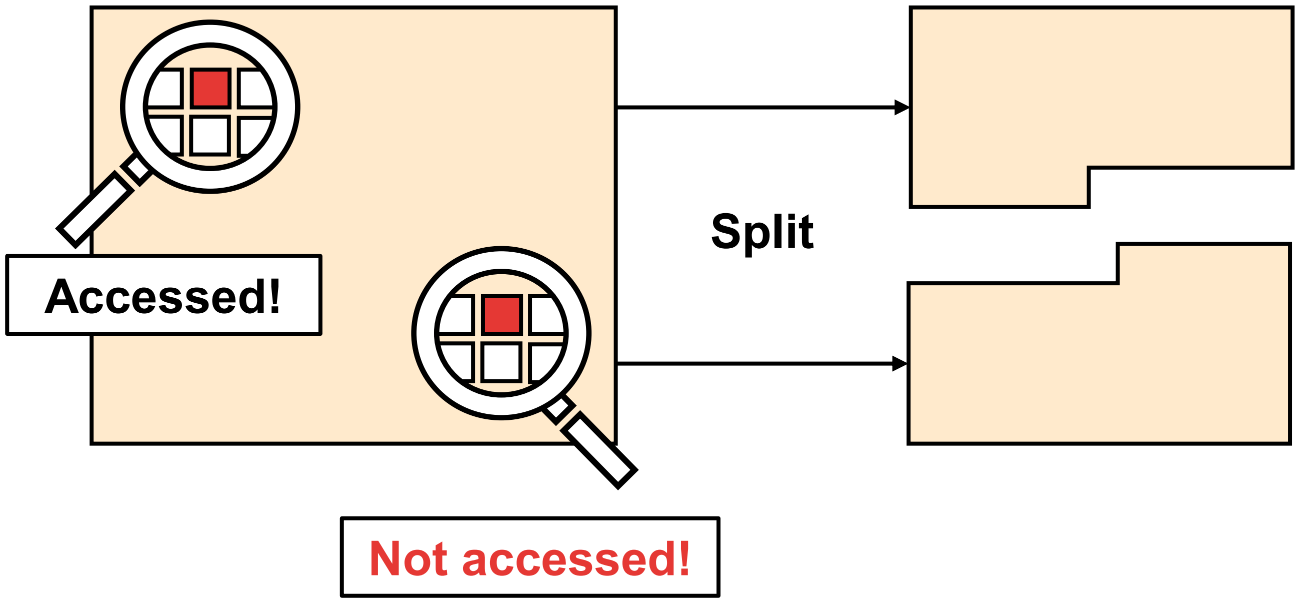
`window size = n`

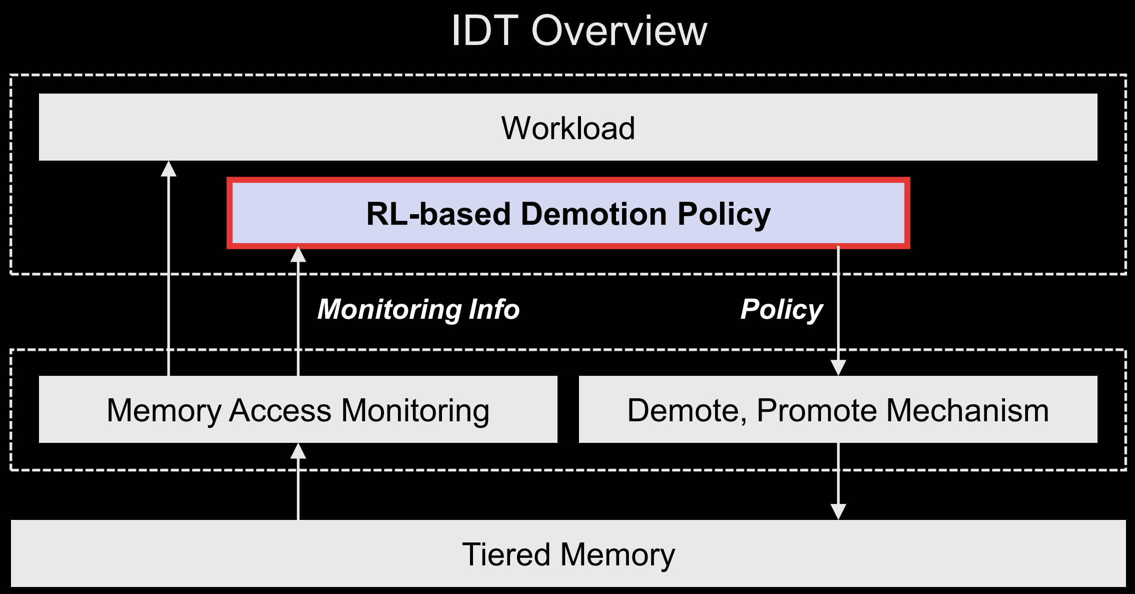
$$P_{i,i+1} = \frac{\binom{n}{a_i} \times \binom{n}{a_{i+1}}}{\binom{2n}{a_i + a_{i+1}}}$$



Region Reconfiguration: Split

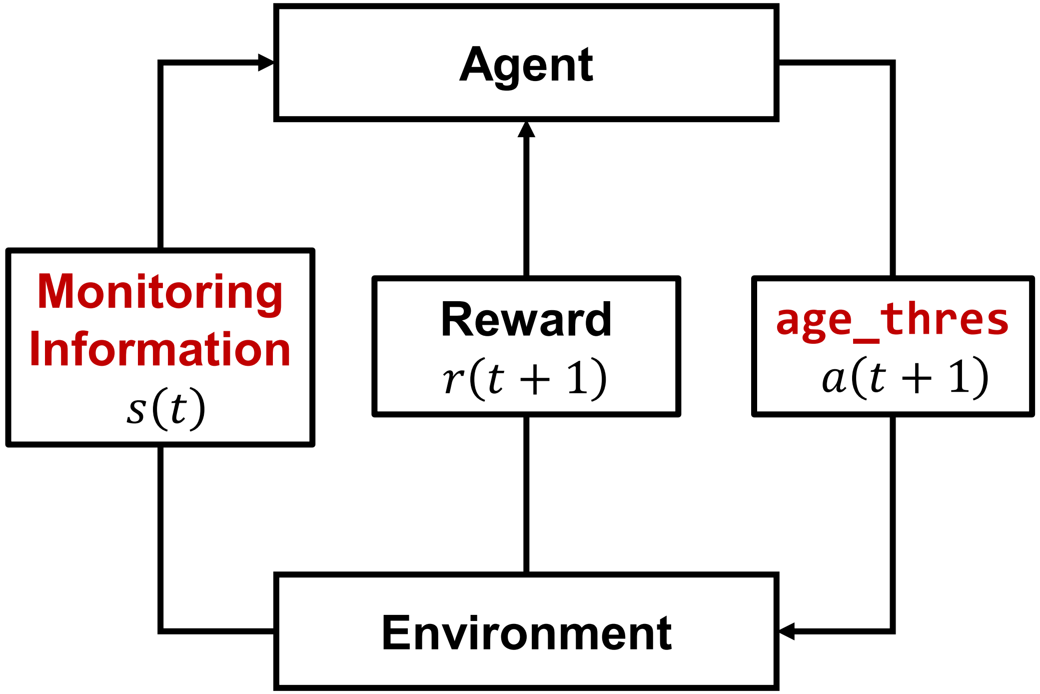
- **Split** region when the **access status of the sampling pages differs** at `sample_interval`





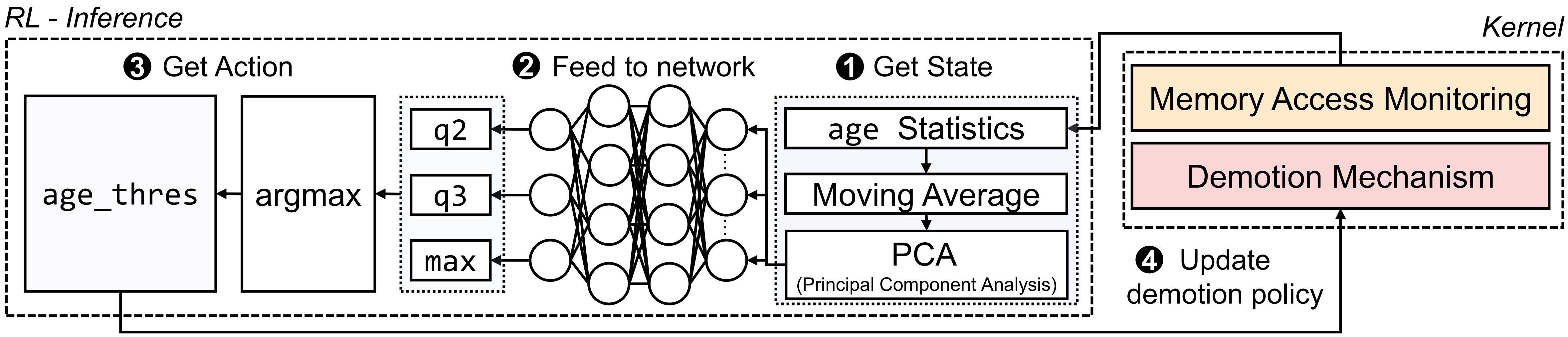
RL-based Demotion Policy

RL: Recall



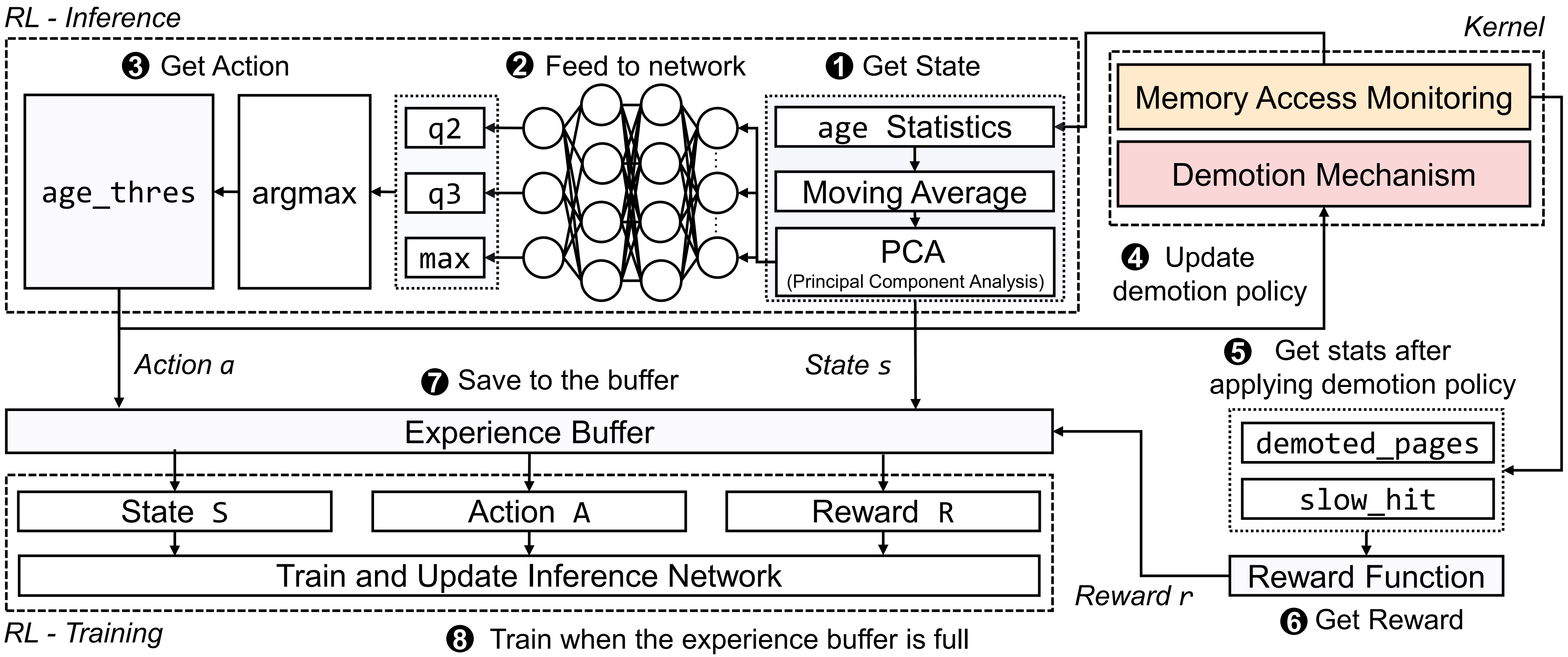
Agent
Monitoring Information
$s(t)$
Reward
$r(t+1)$
age_thres
$a(t+1)$
Environment

RL: Design



RL - Inference
Kernel
❸ Get Action
❷ Feed to network
❶ Get State
age_thres
argmax
q2
q3
max
age Statistics
Moving Average
PCA
(Principal Component Analysis)
Memory Access Monitoring
Demotion Mechanism
❹ Update demotion policy

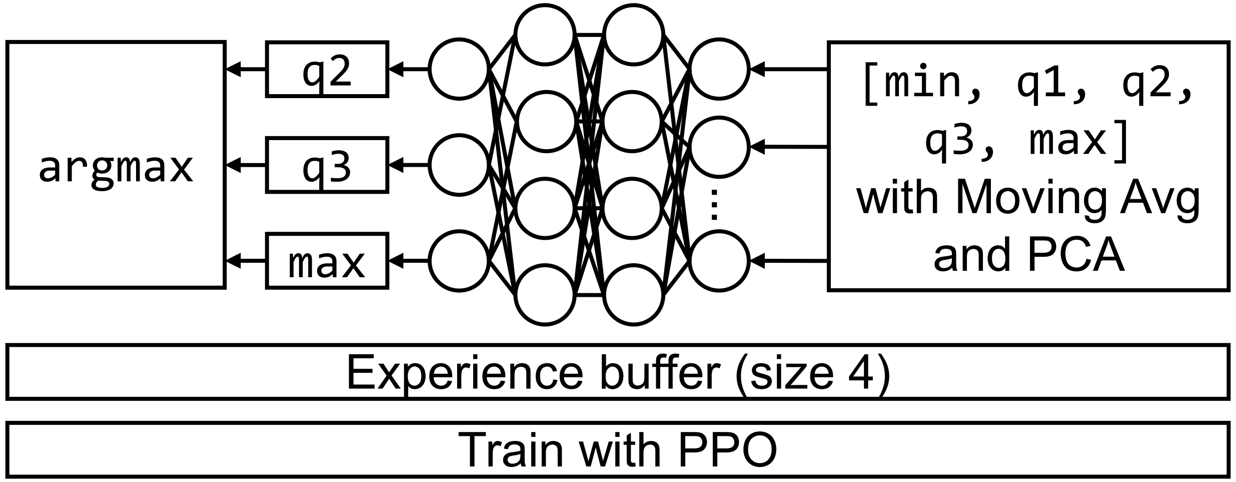
RL: Design



r(t) = log(**demoted_pages(t)** / **slow_hit(t)**)

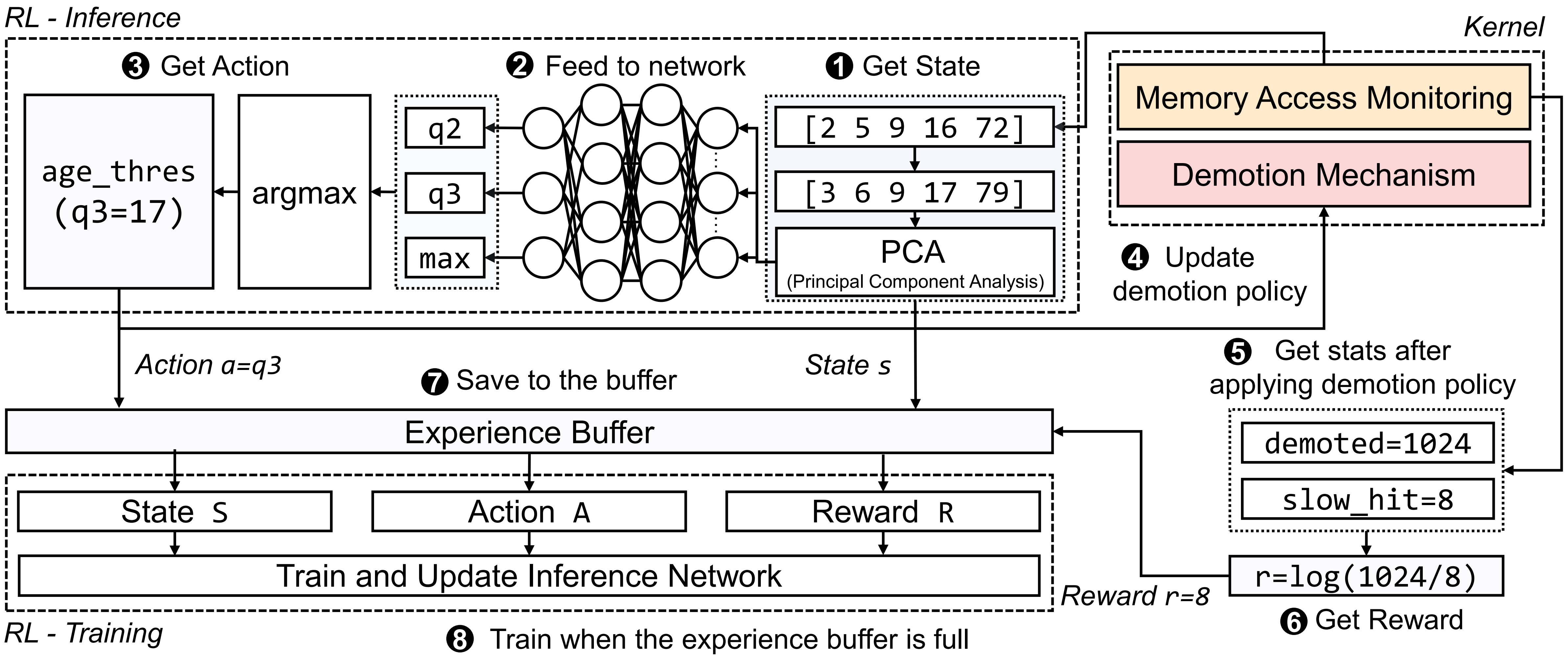
RL: Detail

- Input Layer
  - `min`, `q1` (25 percentile), `q2` (50 percentile), `q3` (75 percentile), `max` age distribution
  - 1x5 state vector
- **2 Hidden Layers**
  - 16, 32 nodes
- **Proximal Policy Optimization[1] (PPO)** Training Algorithm
- Experience buffer size: 4
  - Trained every 4 inferences
- **Pre-train** with GUPS microbenchmark
  - 3 memory access patterns used in HeMem[2]
- Implemented with `PyTorch`-based `Rllib`

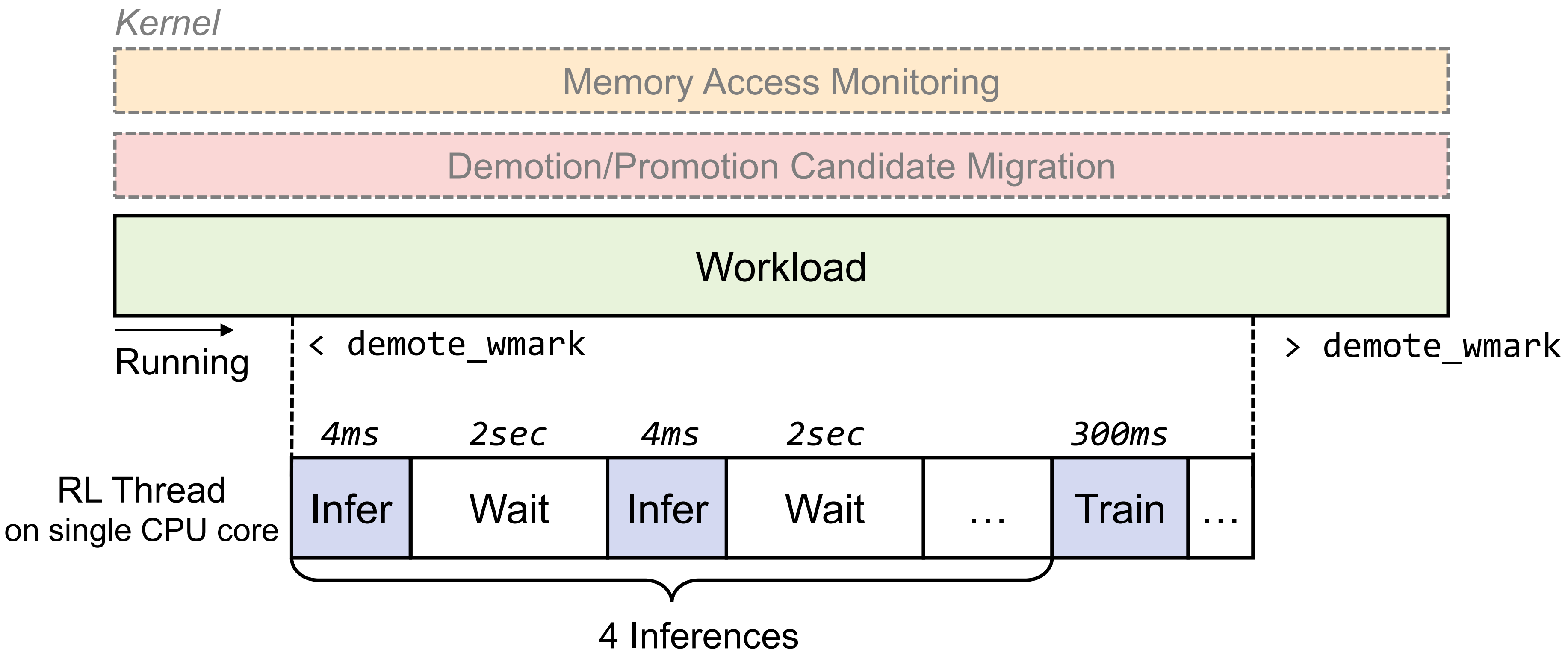


[1] John Schulman et al., "Proximal Policy Optimization Algorithms.", arXiv 2017
[2] Amanda Raybuck et al.,."HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM.", SOSP 2021

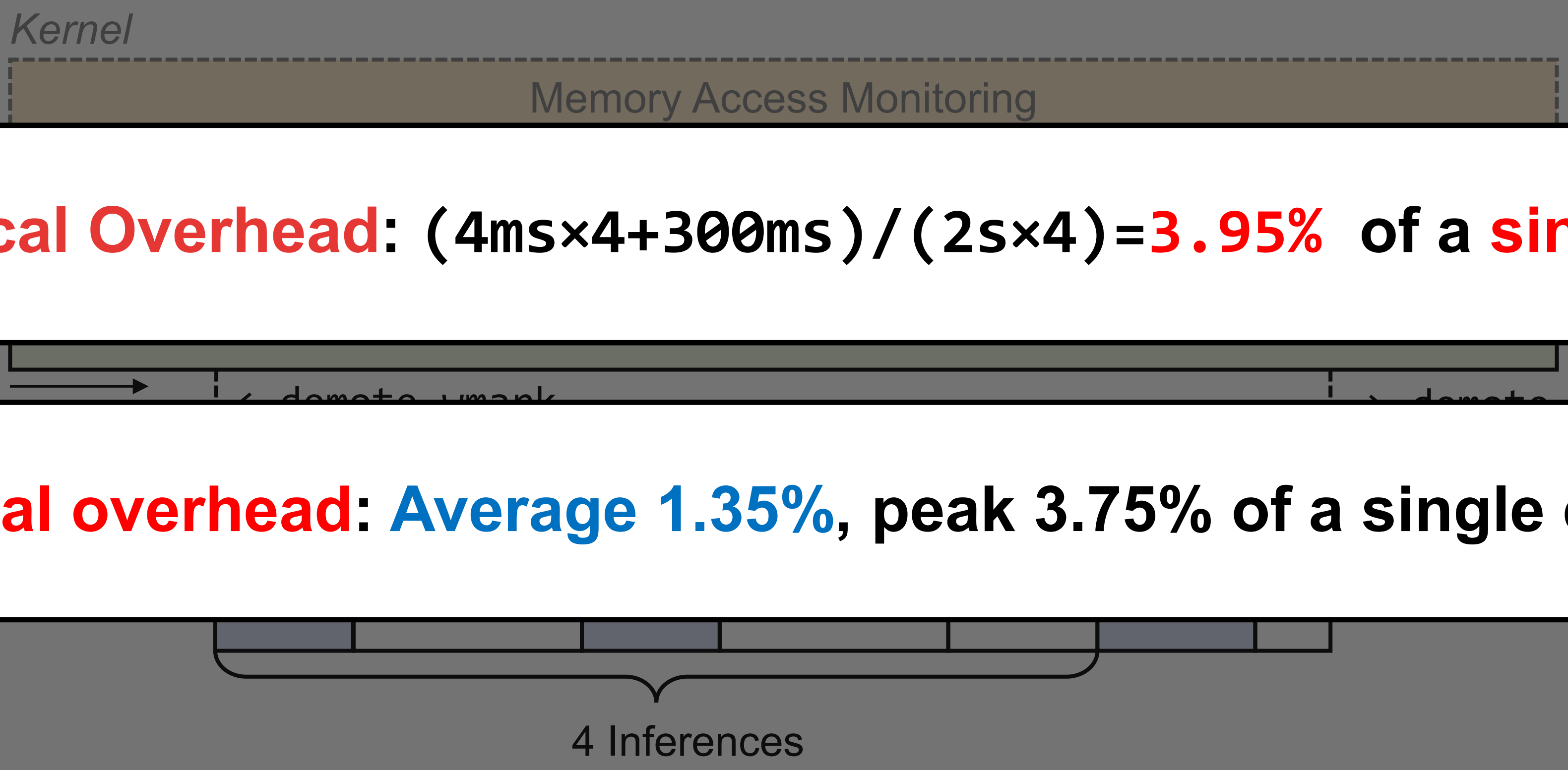
RL: Example


RL - Inference
Kernel
❸ Get Action
❷ Feed to network
❶ Get State
age_thres
(q3=17)
argmax
q2
q3
max
[2 5 9 16 72]
[3 6 9 17 79]
PCA
(Principal Component Analysis)
Memory Access Monitoring
Demotion Mechanism
❹ Update
demotion policy
❺ Get stats after
applying demotion policy
Action a=q3
❼ Save to the buffer
State s
Experience Buffer
demoted=1024
slow_hit=8
State S
Action A
Reward R
Train and Update Inference Network
r=log(1024/8)
Reward r=8
❻ Get Reward
RL - Training
❽ Train when the experience buffer is full

RL: Execution Phases

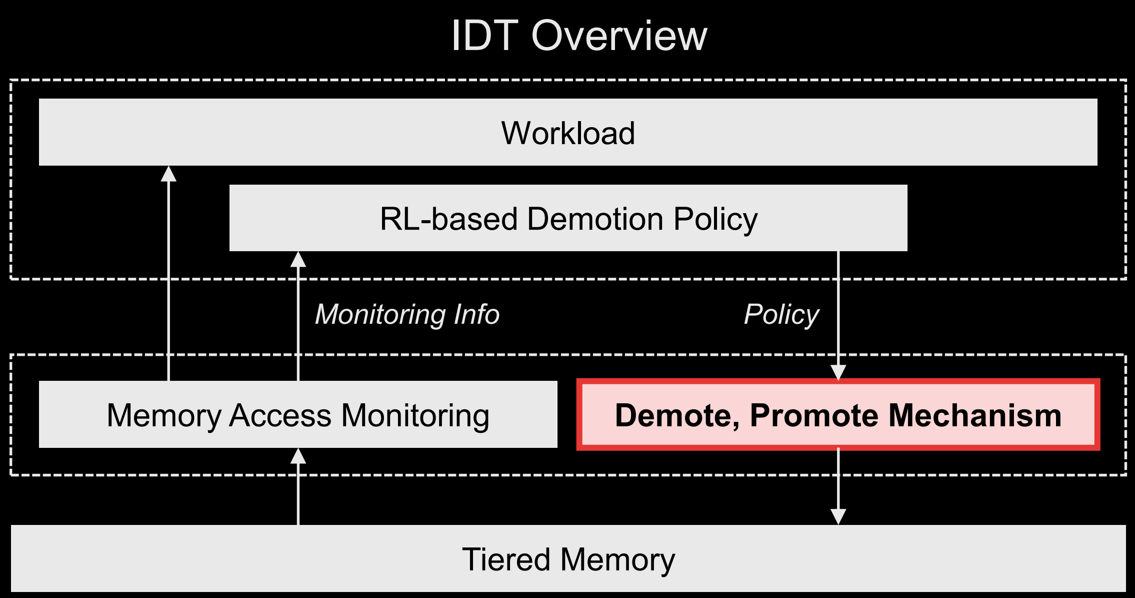


RL Execution Phases



Theoretical Overhead: `(4ms×4+300ms)/(2s×4)=3.95%` **of a single core**

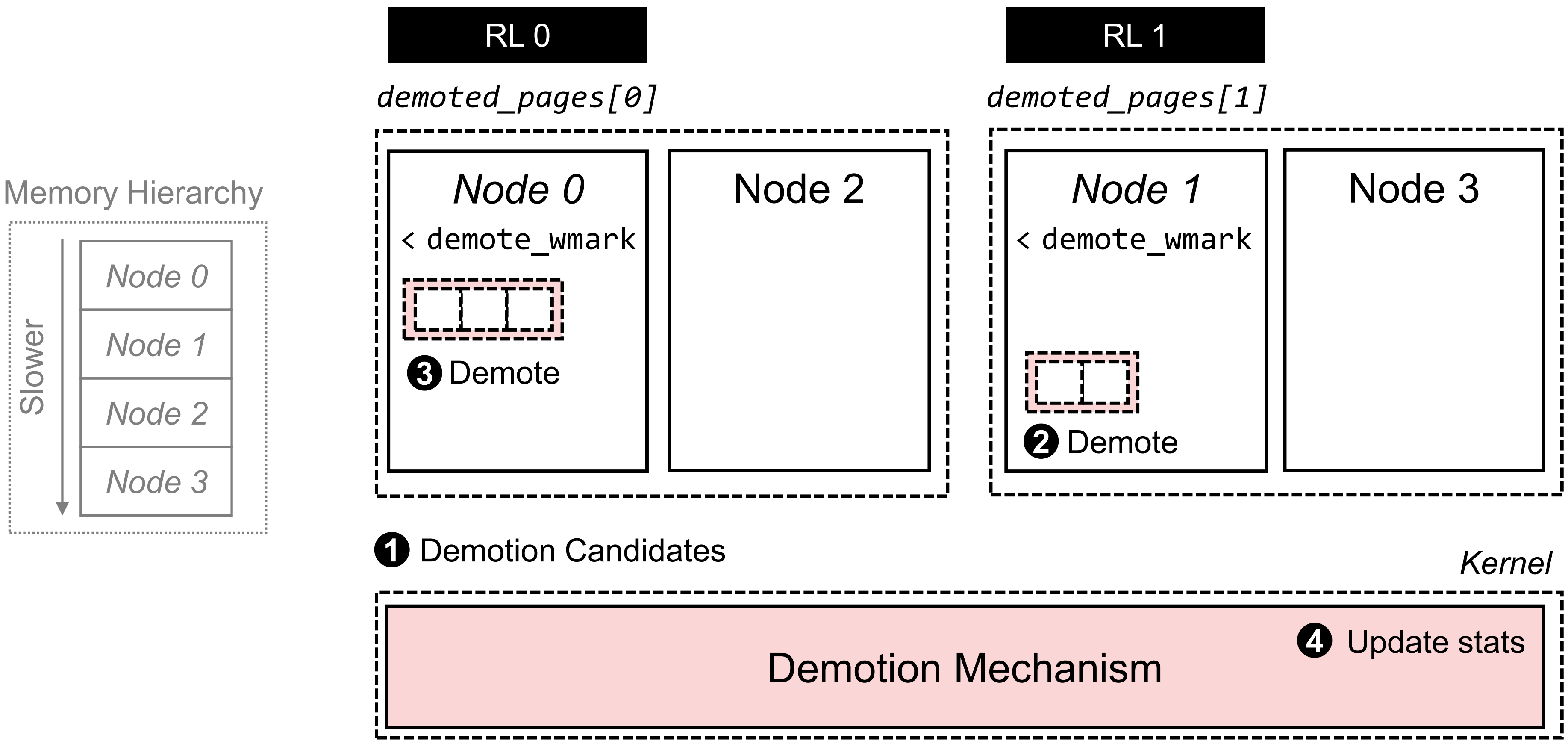
Actual overhead: Average 1.35%, peak 3.75% of a single core



Demotion, Promotion Mechanism

Demotion

- When a memory node's available space < `demote_wmark` (Set to 10%)
- Demote regions with `age > age_thres` and minimum `access`

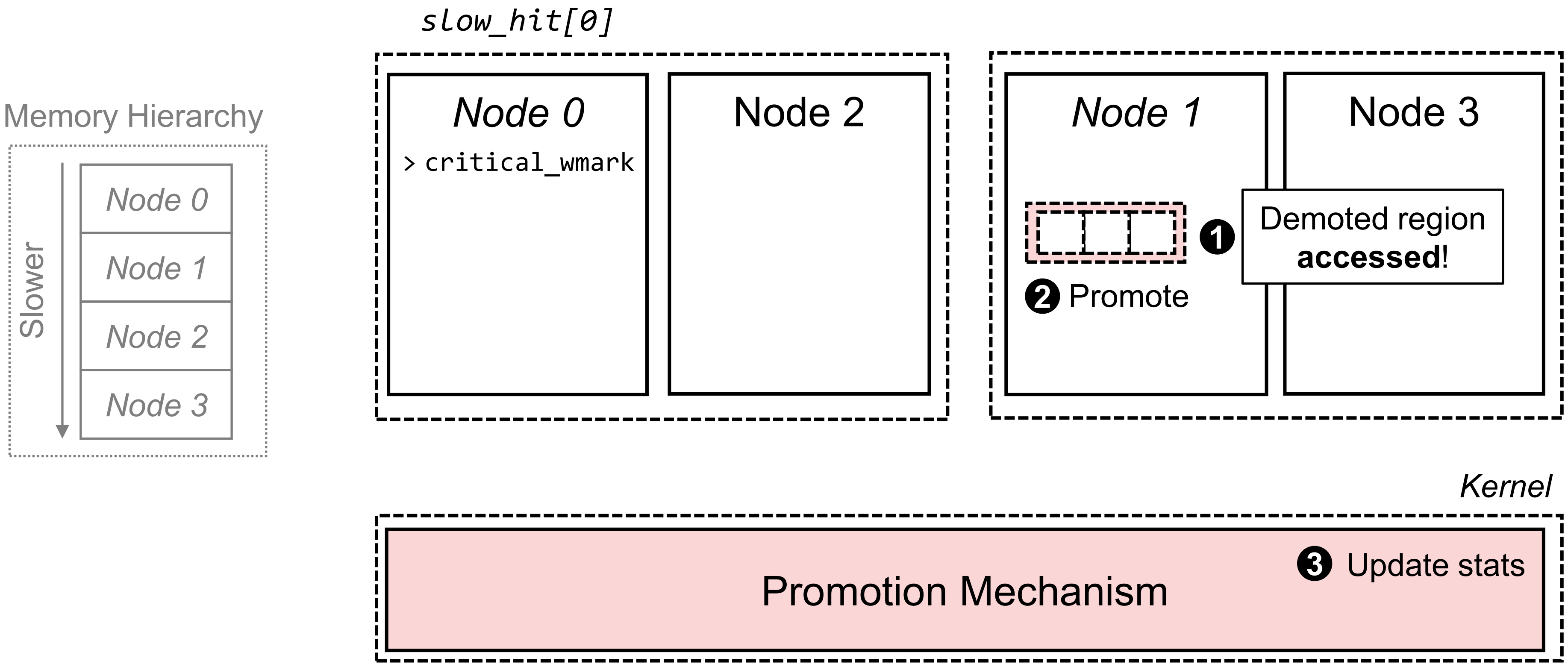


Promotion

- **ARP** (**A**ccessed **R**egion **P**romotion)
- **PRP** (**Predictive** **R**egion **P**romotion)

Promotion: ARP (Accessed Region Promotion)

- Promote when demoted region is **accessed**
  - Destination node should have available space > `critical_wmark` (Set to 1%)

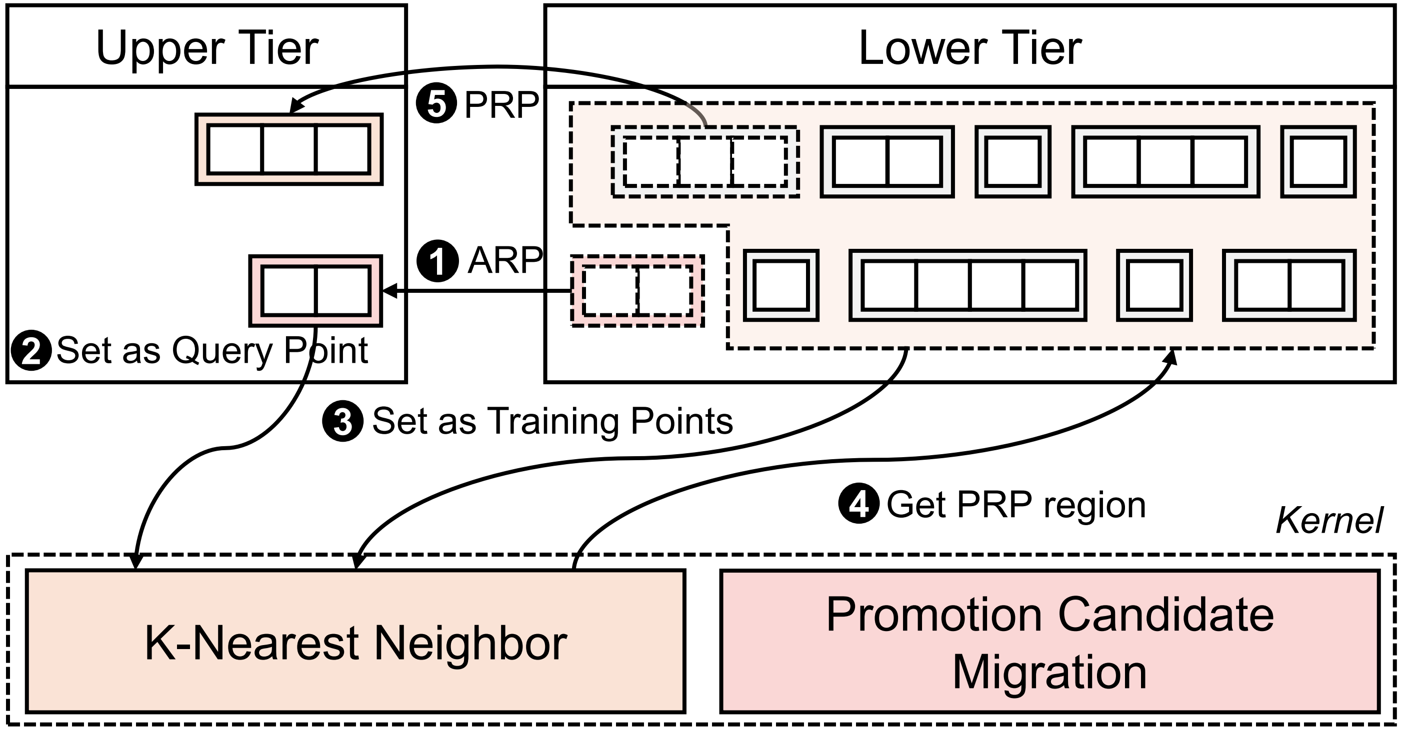


Promotion: PRP (Predictive Region Promotion)

- ARP does not promote until access to the region’s sampling pages is observed
  - Preemptively promoting regions similar to ARP region may be beneficial
- Identify a **similar** region with **k-Nearest Neighbor** and promote

Promotion: PRP (Predictive Region Promotion)

- ARP does not promote until access to the region's sampling pages is observed
  - Preemptively promoting regions similar to ARP region may be beneficial
- Identify a **similar** region with **k-Nearest Neighbor** and promote



distance = Normalized(vaddr distance) + Normalized(access_history distance)

Spatial Locality — Temporal Locality

More Details in the Paper

- Aggressive demotion
  - Tighten demotion criteria when scarce fast memory
- Misplaced region promotion
  - Handle promotion of regions demoted by `kswapd`
- RL formulation
  - Problem formulation
  - Approximation for feasible implementation
- Sensitivity study

IDT: Intelligent Data Placement for Multi-tiered Main Memory with Reinforcement Learning

Juneseo Chang, Seoul National University, South Korea, jschang0215@snu.ac.kr
Wanju Doh, Seoul National University, South Korea, wj.doh@scale.snu.ac.kr
Yaebin Moon, Samsung Electronics, South Korea, yaebin.moon@samsung.com
Eojin Lee, Inha University, South Korea, ejlee@inha.ac.kr
Jung Ho Ahn, Seoul National University, South Korea, gajh@snu.ac.kr

ABSTRACT

To address the limitation of a DRAM-based single-tier in satisfying the comprehensive demands of main memory, multi-tiered memory systems are gaining widespread adoption. To support these systems, operating-system-level solutions that analyze the application's memory access patterns and ensure data placement in the appropriate memory tier have been vastly explored.

In this paper, we identify reinforcement learning (RL) as an effective solution for tiered memory management, and its policy can be formulated in a solvable form using RL. We also demonstrate that an effective region-granularity memory access monitoring method is necessary to provide an accurate environment state to the RL model. Thus, we propose **IDT**, an **i**ntelligent **d**ata placement for multi-**t**iered main memory. IDT incorporates an RL-based demotion policy autotuning and a mechanism that efficiently demotes cold pages to lower-tier memory. IDT also promotes hot pages to upper-tier memory to minimize access on slow memory, featuring a lightweight machine learning algorithm. IDT employs region-granularity memory access monitoring with statistical-testing-based adjacent region merge and split to improve precision and mitigate ambiguity observed in prior works. Experiments on an actual four-tiered memory system show that IDT achieves an average 2.08× speedup over the default Linux kernel and 11.2% performance improvement compared to the state-of-the-art solution.

CCS CONCEPTS

• **Software and its engineering → Memory management**; • **Computer systems organization → Heterogeneous (hybrid) systems**; • **Computing methodologies → Reinforcement learning**.

KEYWORDS

Memory Tiering, Emerging Memory Technologies, Memory Management, Reinforcement Learning

HPDC '24, June 3–7, 2024, Pisa, Italy

ACM ISBN 979-8-4007-0413-0/24/06.
https://doi.org/10.1145/3625549.3658659

ACM Reference Format:
Juneseo Chang, Wanju Doh, Yaebin Moon, Eojin Lee, and Jung Ho Ahn. 2024. IDT: Intelligent Data Placement for Multi-tiered Main Memory with Reinforcement Learning. In *The 33rd International Symposium on High-Performance Parallel and Distributed Computing (HPDC '24), June 3–7, 2024, Pisa, Italy*. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3625549.3658659

1 INTRODUCTION

The growing demand for memory-intensive workloads, such as high-performance computing, graph analytics, and in-memory databases, is highlighting the scaling limitations of a DRAM-based single-tier main memory [39]. To tackle this issue, a variety of memory types with diverse performance characteristics have been adopted to compose tiered memory systems. Recently, the rising interest in memories attached to Compute Express Link (CXL-Memory [9]) underscores that the future lies in multi-tiered memory systems by integrating various heterogeneous memories with a main-memory-like appearance to a system [36]. Cloud vendors, such as Amazon and Google, already serve large memory cloud instances based on multi-tiered memory systems [20, 33].

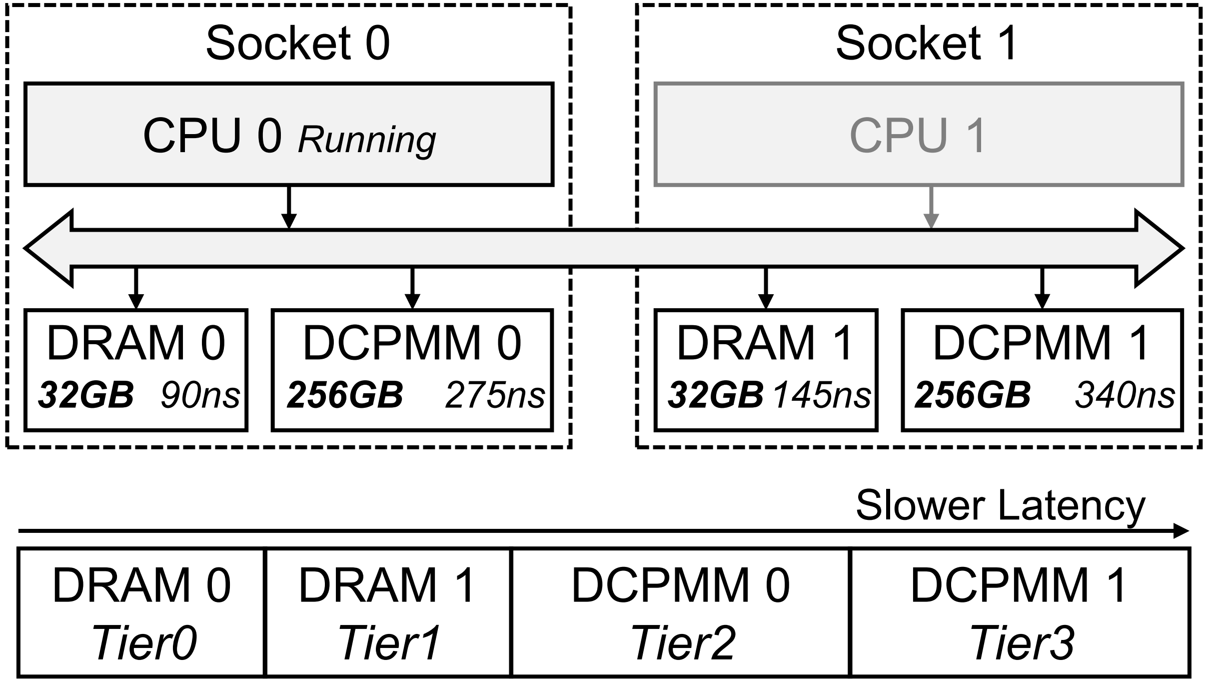
Tiered memory management requires a keen insight into an application's memory usage and placing the data in the proper memory tier according to its hotness. Thus, a number of prior studies have proposed operating-system-(OS)-level solutions to improve application performance by attentively exploiting the tiered memory system [2, 12, 15, 19, 23, 27, 36, 38, 48, 51, 55]. These OS-level tiered memory solutions typically consist of *data placement* to fully leverage diverse memory types and *memory access monitoring* to gather information for guiding data placement.

Data placement. Infrequently accessed pages in tiered memory should be *demoted* to lower-tier slow memory for efficient utilization of upper-tier fast memory. Moreover, to complement demotion, hot pages trapped in slow memory should be identified and *promoted* to upper-tier memory. Several prior studies have utilized the Linux kernel's 2Q LRU [19, 21, 35, 36, 56, 57] or multi-generational LRU (MGLRU) [58] to determine demotion candidates. However, the data hotness identified by 2Q LRU or MGLRU often fails to reflect the actual data hotness of the application. Therefore, precisely tracking both access frequency and recency for each page, and establishing a demotion policy with solid criteria would be more effective. Yet, implementing this method presents the challenge of

Evaluation

Experimental Setup

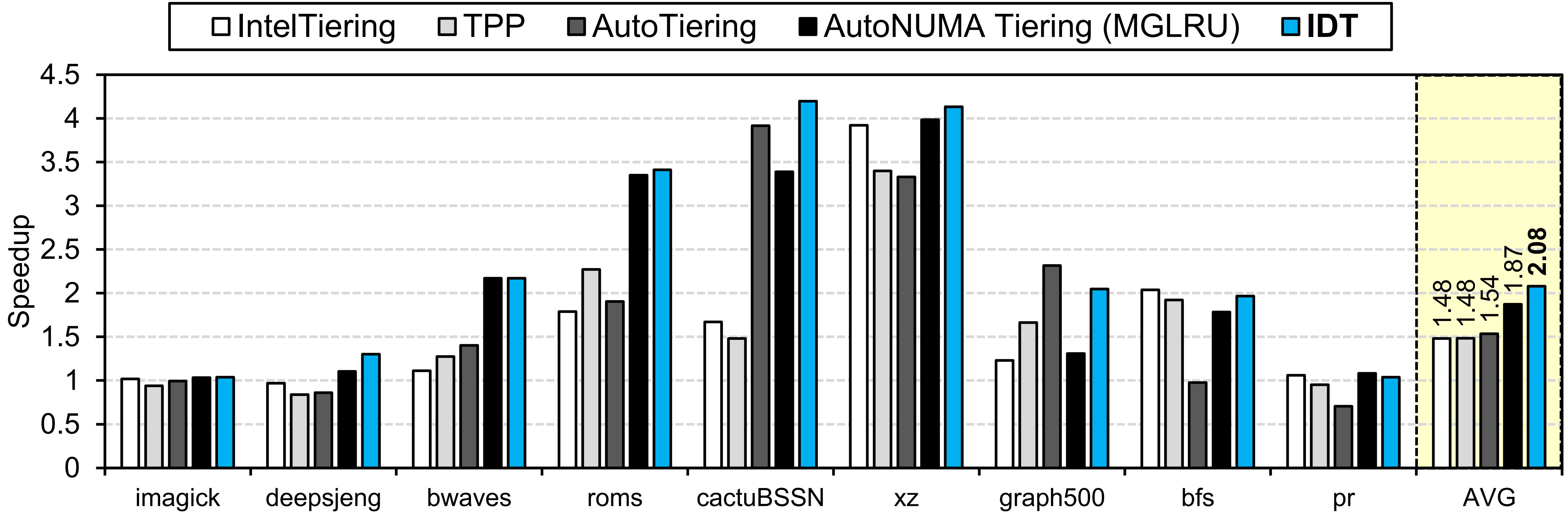
- Based on Linux kernel v6.0.19
  - Memory access monitoring developed with DAMON
- **Multi-tiered memory** setup
  - 2 socket machine with **DRAM** (fast memory) and Intel Optane **DCPMM** (slow memory)
- 4 State-of-the-art solutions for comparison
  - **Intel Tiering 0.8[1], TPP[2], AutoTiering[3], AutoNUMA Tiering (MGRLU)[4]**
- Workloads: SPEC CPU2017, graph500, GAPBS
  - RSS set **96GB~110GB** to facilitate using 3 tiers
- Evaluation metric: Speedup (execution time) normalized against AutoNUMA Balancing



[1] Intel. 2022. Tiering-0.8. https://git.kernel.org/pub/scm/linux/kernel/git/vishal/tiering.git/.
[2] Hasan Al Maruf et al., "TPP: Transparent Page Placement for CXL-Enabled Tiered-Memory," ASPLOS, 2023
[3] Jonghyeon Kim et al., "Exploring the Design Space of Page Management for Multi-Tiered Memory Systems," USENIC ATC (Virtual Event), 2021
[4] Yu Zhao. 2022. Multigenerational LRU Framework. https://lwn.net/Articles/880393/.

Performance

- Outperforms the best-performing state-of-the-art solution AutoNUMA Tiering (MGLRU)) by **11.2%**
  - Average **2.08x** speedup against AutoNUMA Balancing



Limitations

- Other parameters (e.g. 10% and 1% watermarks, sliding window size) are not determined by RL (or ML)
  - Our **goal was to advance the state-of-the-art** solution by **appropriately utilizing RL** (or ML)
  - Future works may apply ML to optimize other parameters
- **Blackbox**: Difficult to explain clear reasons for performance improvement by using RL

Summary

Inaccurate data hotness determined from 2Q/MGLRU

ML for selecting demotion candidates

Statistical testing for region-granularity monitoring

Summary

Inaccurate data hotness determined from 2Q/MGLRU

ML for selecting demotion candidates

Statistical testing for region-granularity monitoring

→

RL-based **Demotion** policy autotuning

Fisher's exact test for region **merge**

Predictive promotion

Outperforms the default Linux kernel by **2.08×**, state-of-the-art solution by **11.2%**

Thank you!

Contact the author: jschang0215@snu.ac.kr

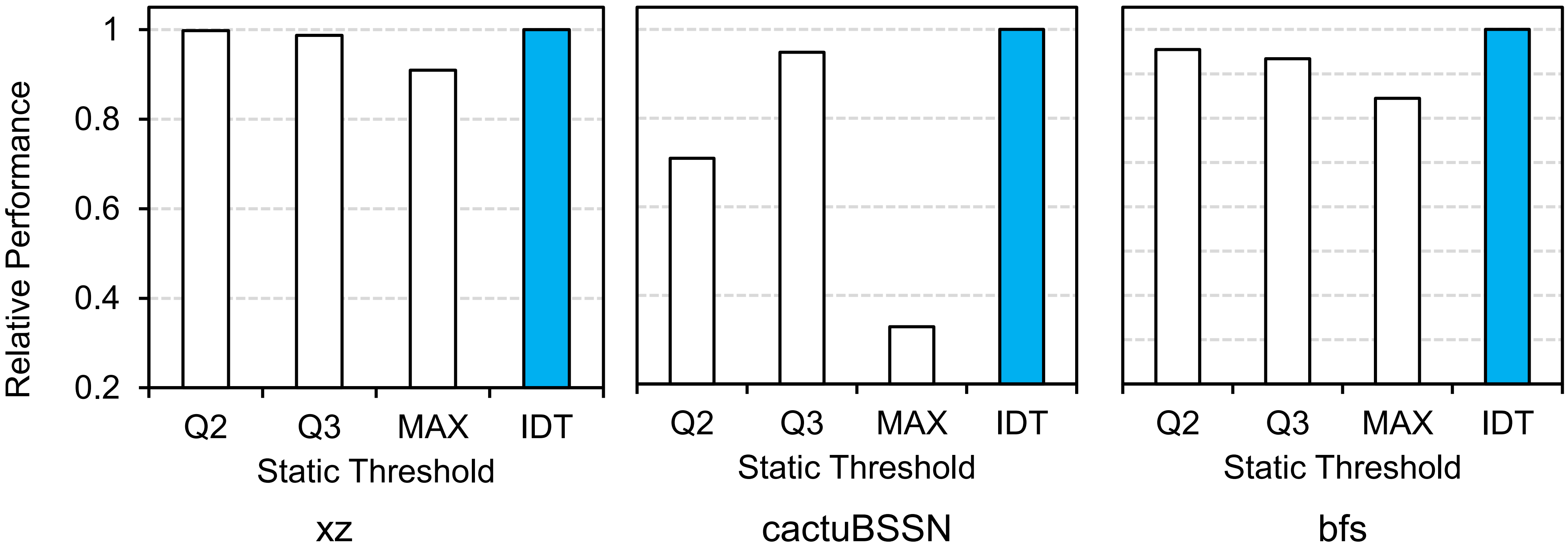
Thank you!

Contact the author: jschang0215@snu.ac.kr

Backup Slides

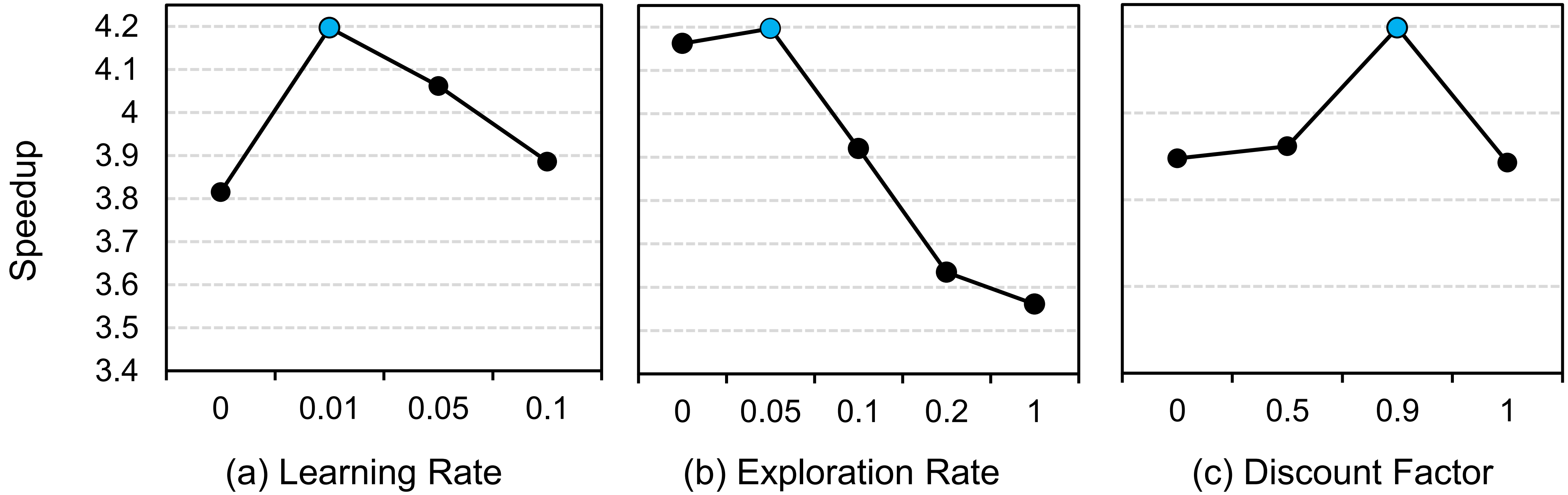
RL Effectiveness

- RL outperforms against static `age_thres`
  - When setting `age_thres` to `q2, q3, max` of `age` distribution (Potential RL actions)



RL Effectiveness (cont'd)

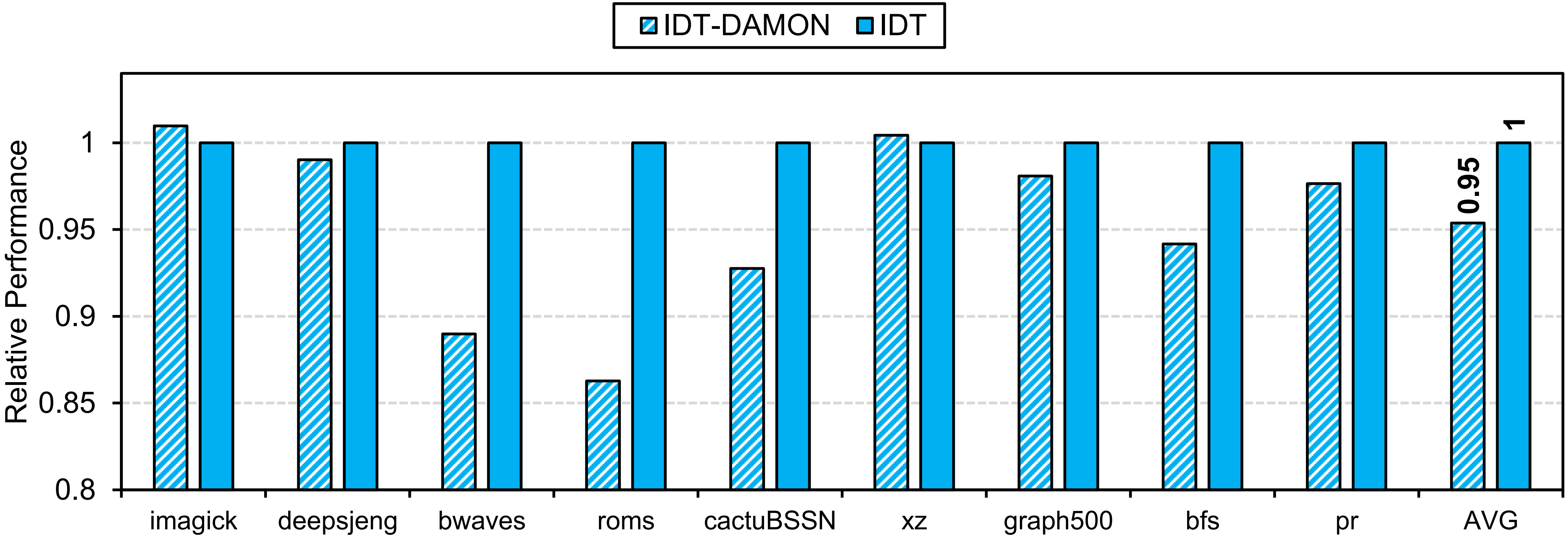
- Performance variation on hyperparameters
  - **Learning rate (α)**: Improvement over α=0 shows efficacy of online training
  - **Exploration rate (ϵ)**: Improvement over ϵ=1 shows effective than random policy
  - **Discount factor (γ)**: Improvement over γ=0 shows effective than only accounting immediate reward



cactuBSSN (SPEC)

Memory Access Monitoring Effectiveness

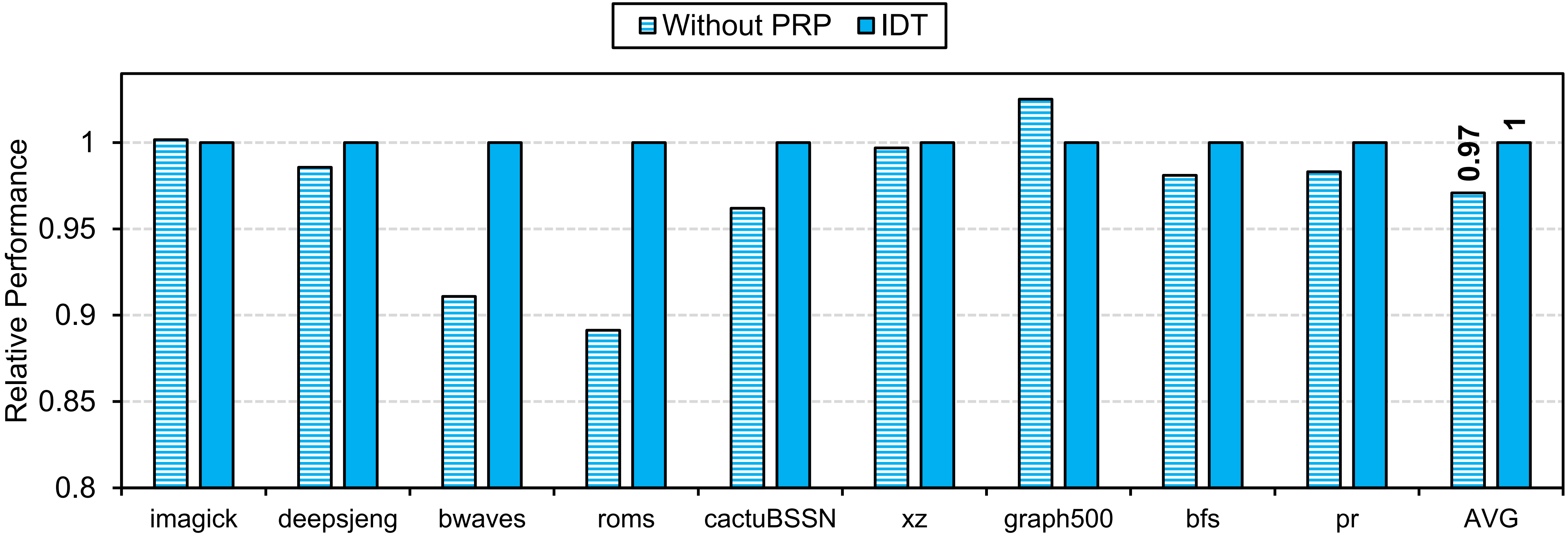
- Compare against applying DAMON[1]
  - Average `history` vector's hamming distance of merge region: 8.13 (DAMON) → **5.15 (IDT)**



[1] SeongJae Park. 2020. DAMON: Data Access Monitor. https://docs.kernel.org/mm/damon/index.html.

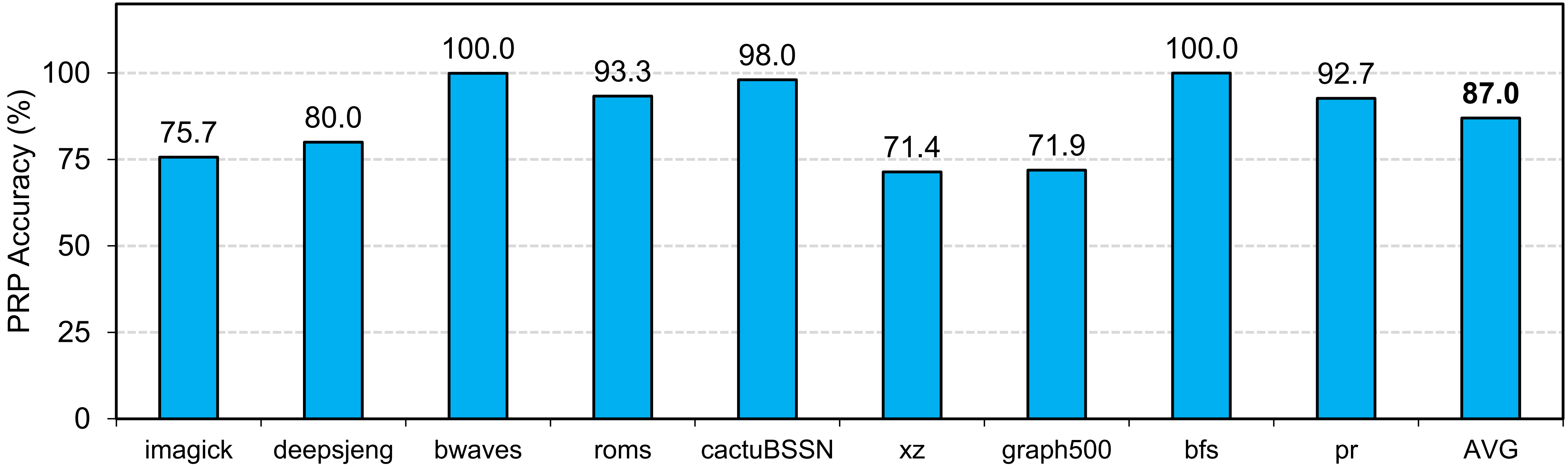
Predictive Region Promotion Effectiveness

- Compare against without PRP



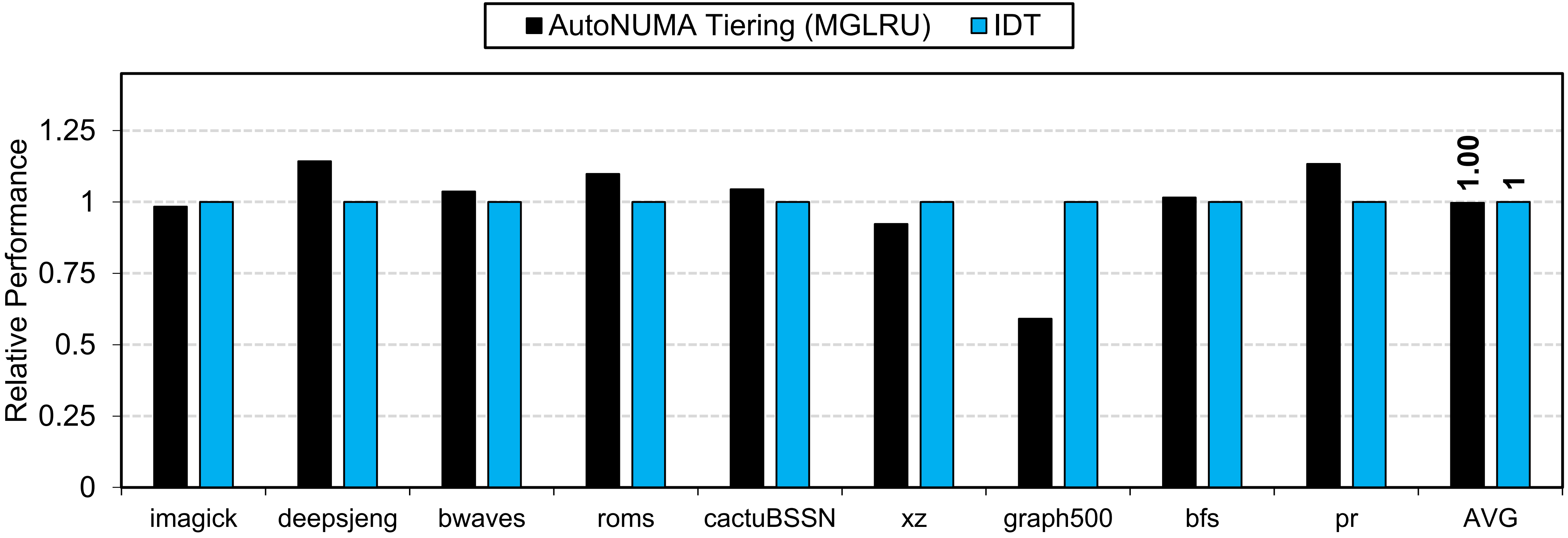
Predictive Region Promotion Effectiveness (cont'd)

- Compare against without PRP
  - High PRP accuracy (ratio of region accessed that was promoted by PRP)



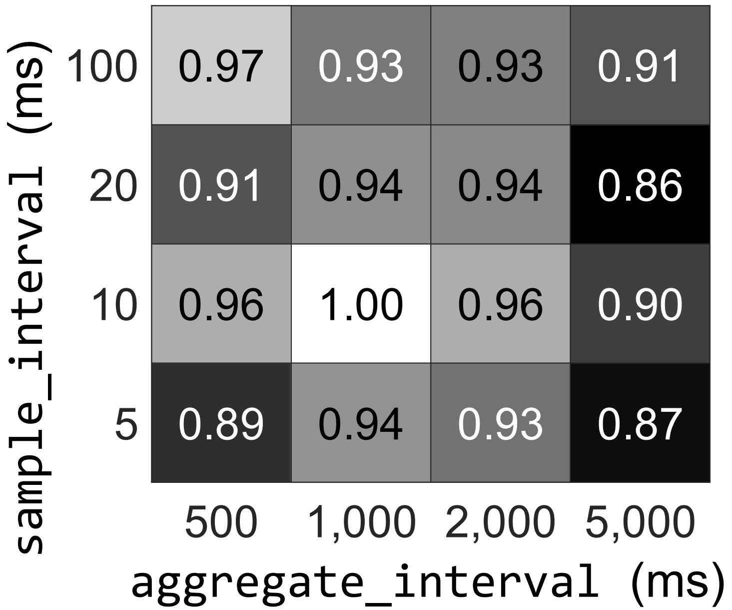
Two-tiered memory

- Performance on two-tiered memory configuration
  - Set DRAM 0 (Tier 0) and DRAM 1 (Tier 1) to 64GB and RSS to 96GB~110GB
  - Similar performance to AutoNUMA Tiering (MGRLU)

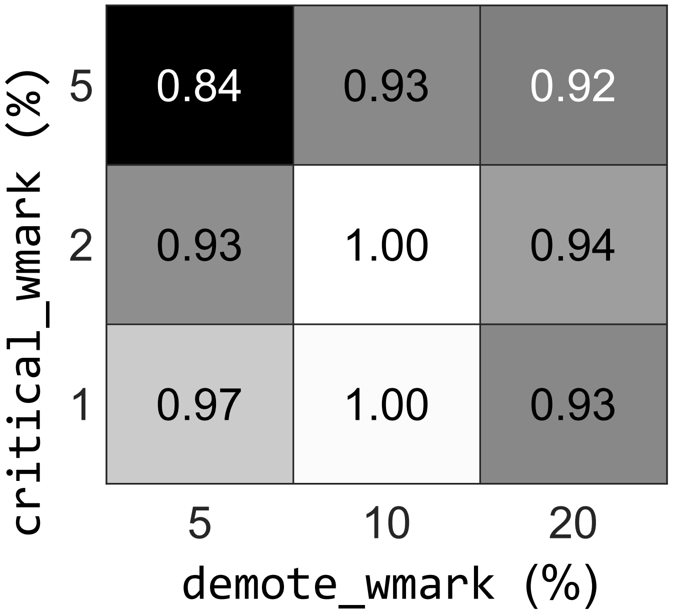


Sensitivity Study

- Smaller interval: Finer sampling and responsive demotion/promotion
  - **but** overhead increase
- Larger watermark: Reserve fast memory for potential allocation requests
  - **but** may not fully leverage the performance benefits of fast memory



(a) Intervals



(b) Watermarks

RL Pretraining

- Pre-trained using the Giga Update operations Per Second (GUPS) microbenchmark[1] with 100GB RSS
  - **Uniform random access**: Random access over the working set
  - **Hot set**: 90% of access on 4GB hot objects and the remaining uniform randomly
  - **Dynamic hot set**: Change hot objects every 150-second intervals.

[1] Amanda Raybuck et al.,."HeMem: Scalable Tiered Memory Management for Big Data Applications and Real NVM.", SOSP 2021

Aggressive Demotion

- If available space < `critical_wmark` (Set to 1%)
- **Tighten demotion candidate criteria**
  - `age > age_thres`
  - `access < (min_access + max_access) / 2`

Algorithm 1 Demotion in memory node *nid*

$max\_access \leftarrow max\{access(r) \mid r \in regions(nid)\}$
$min\_access \leftarrow min\{access(r) \mid r \in regions(nid)\}$
if capacity(nid) < `aggressive_demote_wmark` **then**
    $access\_thres \leftarrow (min\_access + max\_access)/2$
    Demote regions with:
        age > `age_thres` and access < $access\_thres$
    if no demoted pages **then**
        Try demote all regions
    end if
else if capacity(nid) < `demote_wmark` **then**
    Demote regions with:
        age > `age_thres` and access < min_access
end if

Misplaced Region Promotion

- IDT's promotion may place region in suboptimal tier
- `kswapd` may demote in intensive memory usage
- Track by setting `demoted` flag when region is demoted
    1. Detect misplaced region with `demoted` flag
    2. Check upper-tier available space > `critical_wmark`
    3. Promote

Region Reconfiguration Methods

- DAMON[1]
  - Merge if access frequency difference less than 10% of the maximum frequency across all regions
  - Split randomly
- MTM[2]
  - Merge if access frequency difference less than 1/3 of total scan counts
  - Split if two sampling page access status differ

Merge criteria is heuristic and requires magic number