UH-MEM: Utility-Based Hybrid Memory Management

Yang Li, Saugata Ghose, Jongmoo Choi,
Jin Sun, Hui Wang, Onur Mutlu

Carnegie Mellon
SAFARI

ETH zürich

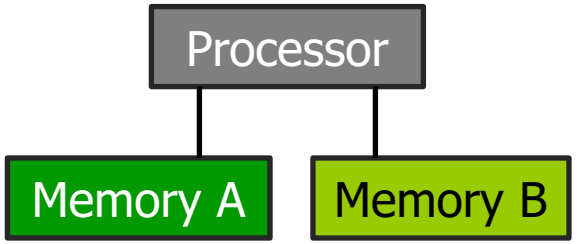
DKU
DANKOOK UNIVERSITY

北京航空航天大学
BEIHANG UNIVERSITY
1952

Executive Summary

- DRAM faces **significant technology scaling difficulties**
- Emerging memory technologies overcome difficulties (e.g., **high capacity, low idle power**), but have other shortcomings (e.g., **slower than DRAM**)

- **Hybrid memory system** pairs DRAM with emerging memory technology
  - Goal: combine **benefits of both memories** in a cost-effective manner
  - **Problem: Which memory do we place each page in, to optimize system performance?**



- Our approach: **UH-MEM** (Utility-based Hybrid MEmory Management)
  - **Key Idea:** for each page, estimate ***utility*** (i.e., performance impact) of migrating page, then use utility to guide page placement
  - **Key Mechanism:** a **comprehensive model** to estimate utility using memory access characteristics, application impact on system performance

- UH-MEM **improves performance by 14%** on average over the best of three state-of-the-art hybrid memory managers

Outline

- **Background**
- Existing Solutions
- UH-MEM: Goal and Key Idea
- Key Mechanisms of UH-MEM
- Evaluation
- Conclusion

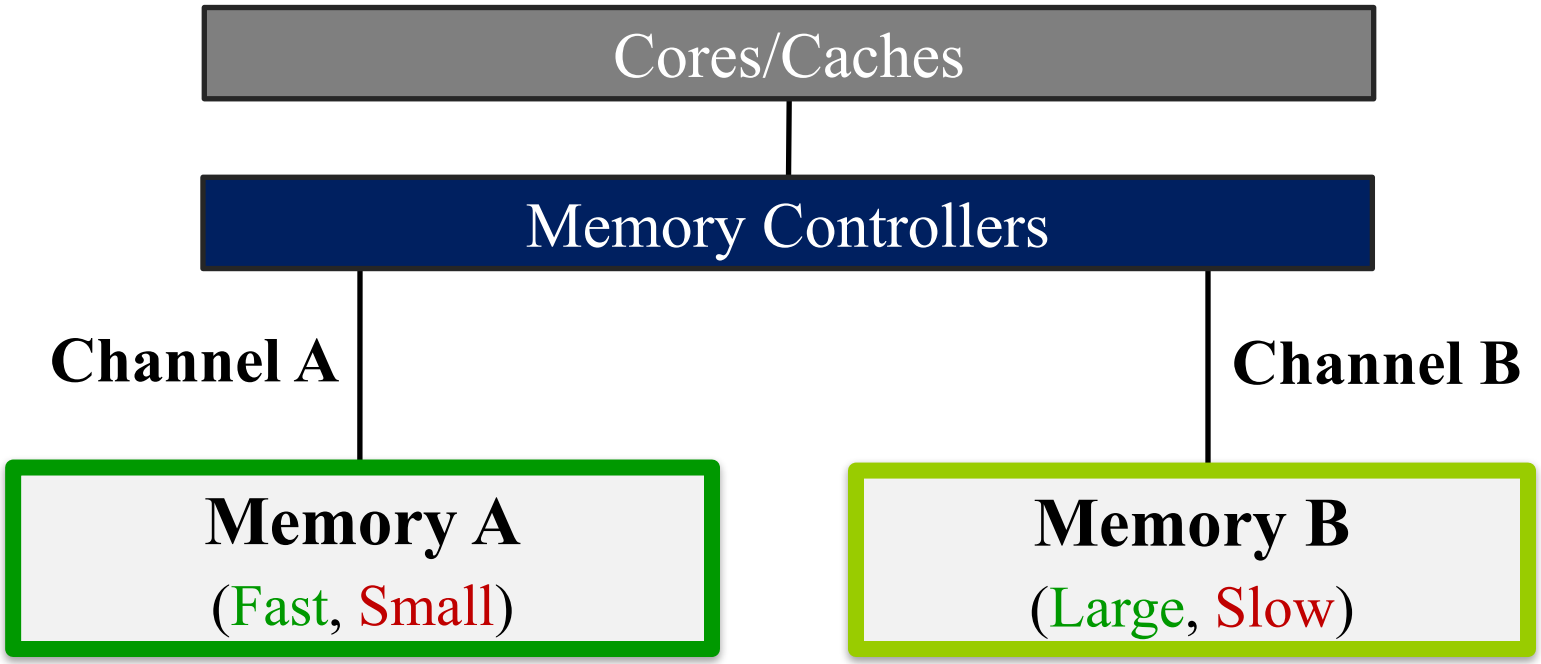
State-of-the-Art Memory Technologies

- DRAM scaling has already become increasingly difficult
  - Increasing cell leakage current, reduced cell reliability, increasing manufacturing difficulties [Kim+ ISCA 2014], [Liu+ ISCA 2013], [Mutlu IMW 2017], [Mutlu DATE 2017]
  - **Difficult to significantly improve capacity, speed, energy**
- **Emerging memory technologies** are promising

3D-Stacked DRAM	higher bandwidth	smaller capacity
Reduced-Latency DRAM (e.g., RLDRAM, TL-DRAM)	lower latency	higher cost
Low-Power DRAM (e.g., LPDDR3, LPDDR4)	lower power	higher latency higher cost
Non-Volatile Memory (NVM) (e.g., PCM, STTRAM, ReRAM, 3D Xpoint)	larger capacity	higher latency higher dynamic power lower endurance

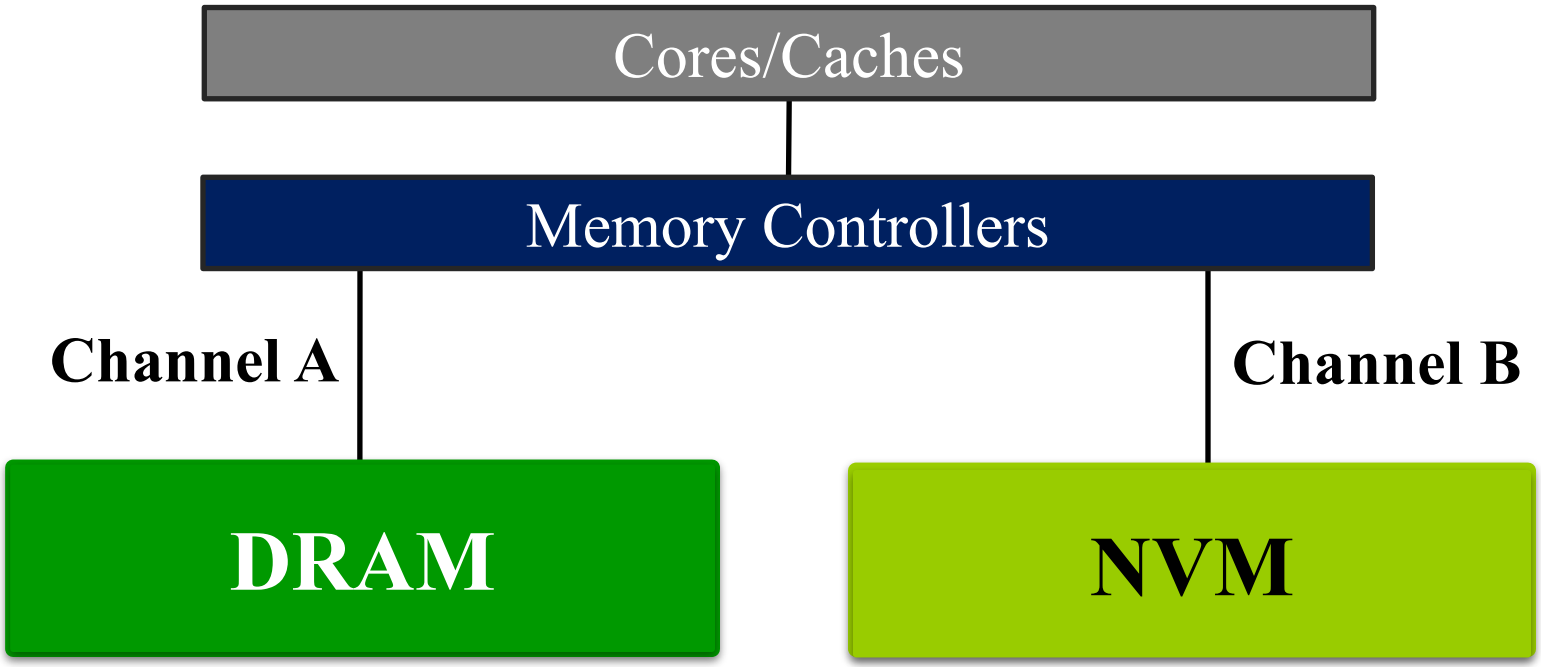
Hybrid Memory System

- Pair **multiple memory technologies** together to exploit the **advantages of each memory**
  - e.g., DRAM-NVM: DRAM-like latency, NVM-like capacity



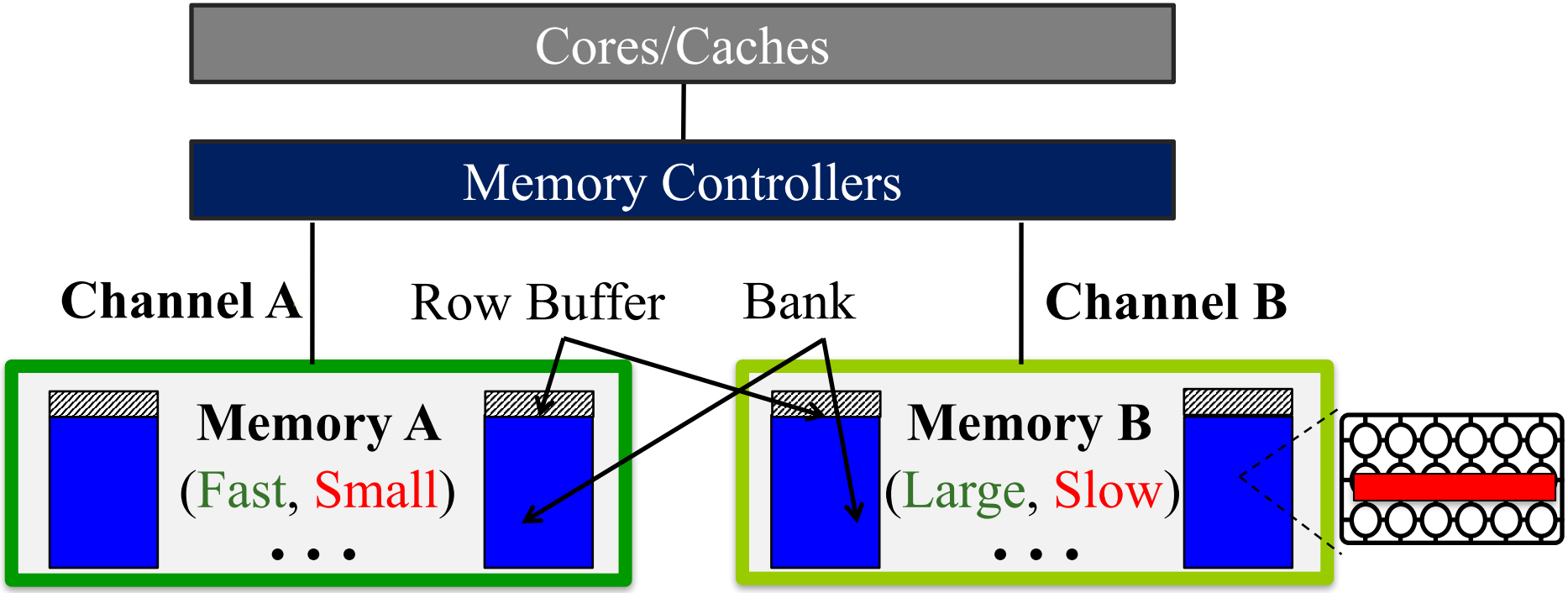
Hybrid Memory System

- Pair **multiple memory technologies** together to exploit the **advantages of each memory**
  - e.g., DRAM-NVM: DRAM-like latency, NVM-like capacity



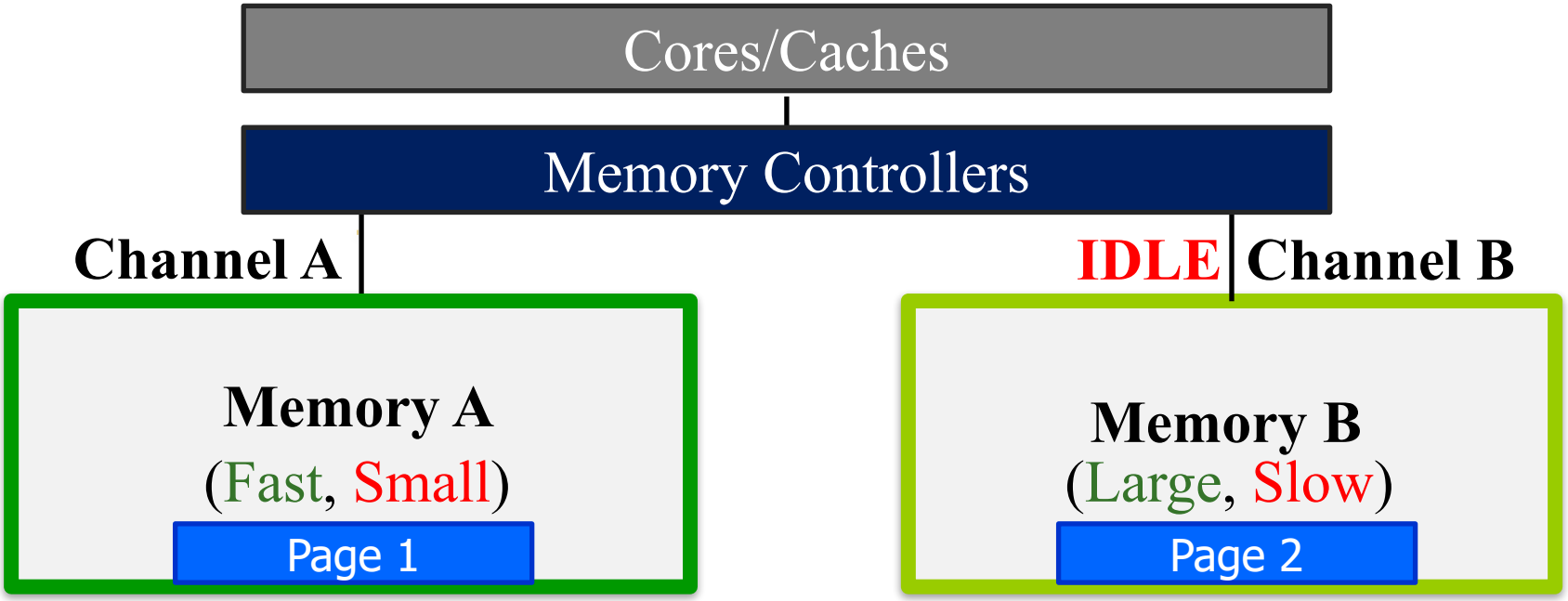
- Separate channels: Memory A and B can be **accessed in parallel**

Hybrid Memory System: Background



- Bank: Contains multiple arrays of cells
  - Multiple banks can serve accesses in parallel
- Row buffer: Internal buffer that holds the open row in a bank
  - Row buffer hits: serviced ~3x faster, **similar latency for both Memory A and B**
  - Row buffer misses: **latency is higher for Memory B**

Hybrid Memory System: Problem



Which memory do we place each page in, to **maximize system performance**?

- Memory A is fast, but small
- Load should be balanced on both channels
- Page migrations have performance and energy overhead

Outline

- Background
- **Existing Solutions**
- UH-MEM: Goal and Key Idea
- Key Mechanisms of UH-MEM
- Evaluation
- Conclusion

Existing Solutions

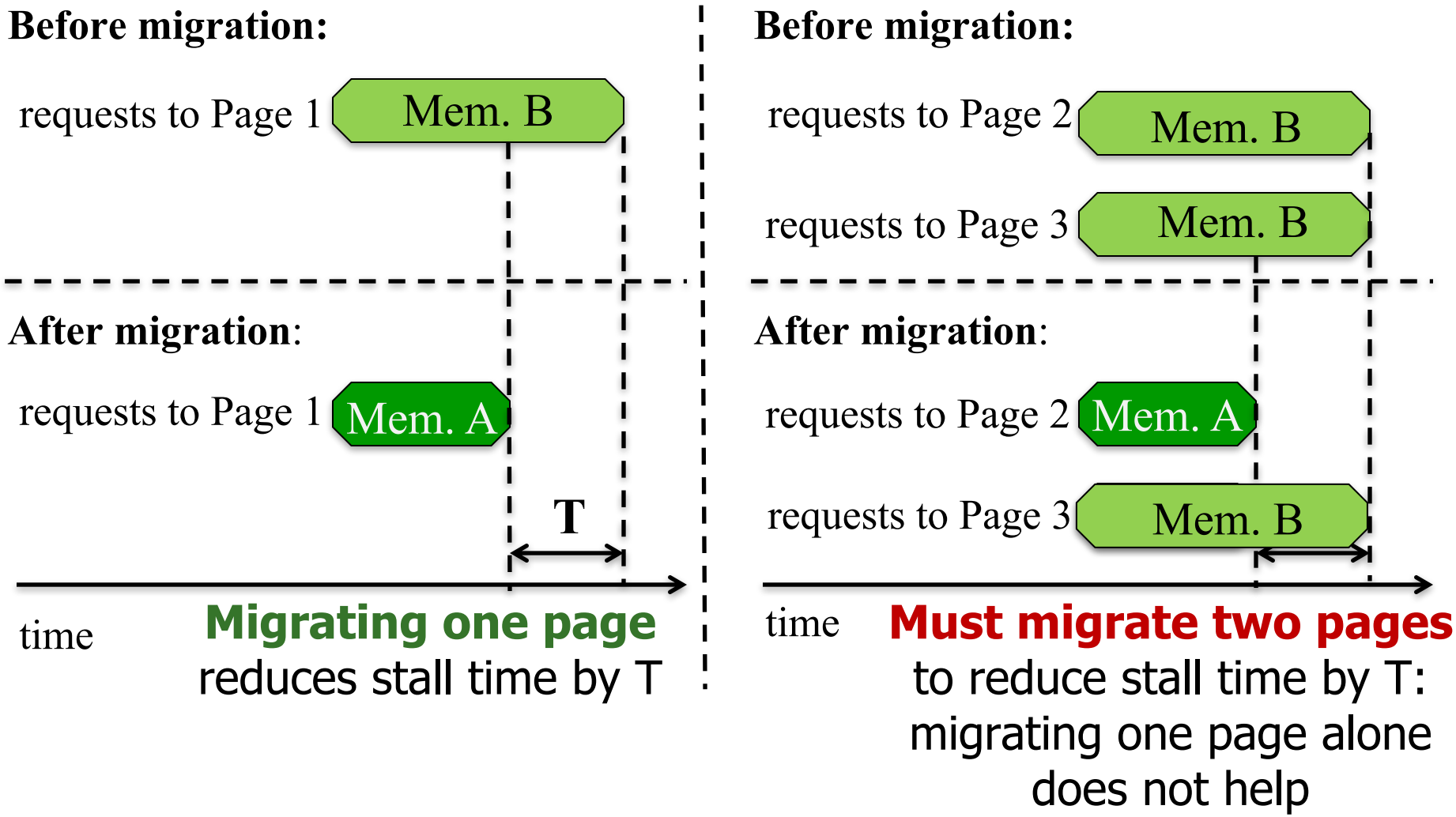
- ***ALL*** [Qureshi+ ISCA 2009]
  - Migrate **all pages** from slow memory to fast memory **when the page is accessed**
  - Fast memory is LRU cache: pages evicted when memory is full
- ***FREQ:*** **Access frequency** based approach [Ramos+ ICS 2011], [Jiang+ HPCA 2010]
  - Access frequency: number of times a page is accessed
  - Keep pages with **high access frequency** in fast memory
- ***RBLA:*** **Access frequency** and **row buffer locality** based approach [Yoon+ ICCD 2012]
  - Row buffer locality: fraction of row buffer hits out of all memory accesses to a page
  - Keep pages with **high access frequency and low row buffer locality** in fast memory

Weaknesses of Existing Solutions

- They are all heuristics that consider only a ***limited part* of memory access behavior**
- **Do not *directly* capture the overall system performance impact** of data placement decisions

- Example: None capture **memory-level parallelism** (MLP)
  - Number of ***concurrent* memory requests** from the same application when a page is accessed
  - Affects how much page migration helps performance

Importance of Memory-Level Parallelism



Page migration decisions **need to consider MLP**

Outline

- Background
- Existing Solutions
- **UH-MEM: Goal and Key Idea**
- Key Mechanisms of UH-MEM
- Evaluation
- Conclusion

Our Goal

A **generalized** mechanism that

1. Directly estimates the **performance benefit of migrating a page** between **any two types of memory**

2. Places **only** the **performance-critical data** in the fast memory

Utility-Based Hybrid Memory Management

- A memory manager that works for ***any*** hybrid memory
  - e.g., DRAM-NVM, DRAM-RLDRAM

- **Key Idea**
  - For each page, use **comprehensive** characteristics to calculate estimated ***utility*** (i.e., performance impact) of migrating page from one memory to the other in the system
  - **Migrate only pages with the highest utility** (i.e., pages that improve system performance the most when migrated)

Outline

- Background
- Existing Solutions
- UH-MEM: Goal and Key Idea
- **Key Mechanisms of UH-MEM**
- Evaluation
- Conclusion

Key Mechanisms of UH-MEM

- For each page, estimate **utility** using a **performance model**
  - **Application stall time reduction**

    How much would migrating a page benefit the performance of the application that the page belongs to?
  - **Application performance sensitivity**

    How much does the improvement of a single application's performance increase the *overall* system performance?

$$Utility = \Delta StallTime_i \times Sensitivity_i$$

- **Migrate** only pages whose utility exceeds the migration threshold from slow memory to fast memory
- Periodically **adjust migration threshold**

Key Mechanisms of UH-MEM

- For each page, estimate **utility** using a **performance model**
  - **Application stall time reduction**

    How much would migrating a page benefit the performance of the application that the page belongs to?
  - **Application performance sensitivity**

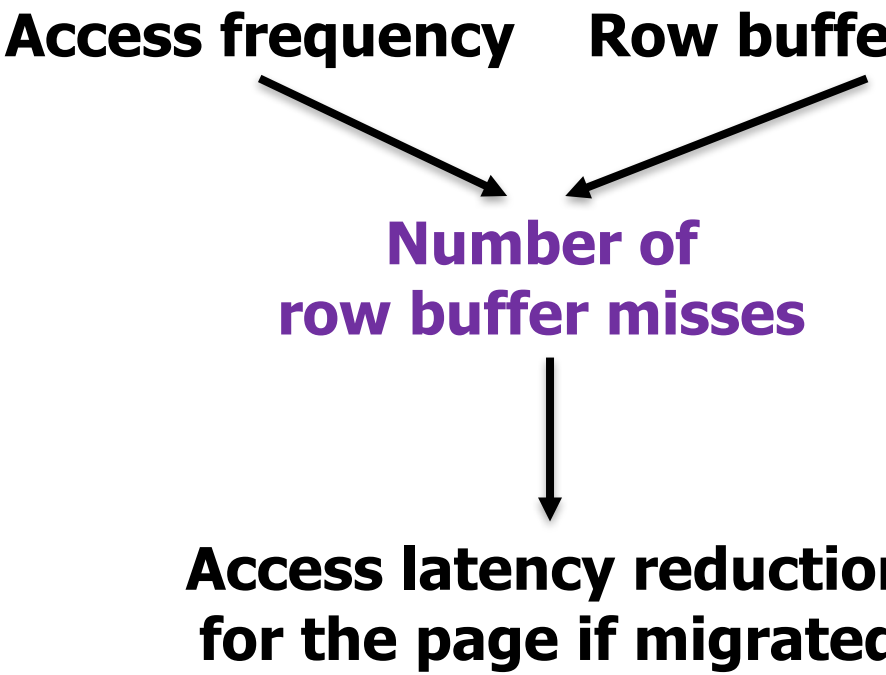
    How much does the improvement of a single application's performance increase the *overall* system performance?

$$Utility = \Delta StallTime_i \times Sensitivity_i$$

- **Migrate** only pages whose utility exceeds the migration threshold from slow memory to fast memory
- Periodically **adjust migration threshold**

Estimating Application Stall Time Reduction

- Use **access frequency**, **row buffer locality**, and **MLP** to estimate application's stall time reduction



Why not row buffer hits?
Hits have **equal latency** in both memories, so **no need to track them**.

Calculating Total Access Latency Reduction

- Use counter to track number of row buffer misses (#RBMiss)

- Calculate change in number of memory cycles if page is migrated

Reduction in miss latency if we migrate from Memory B to Memory A

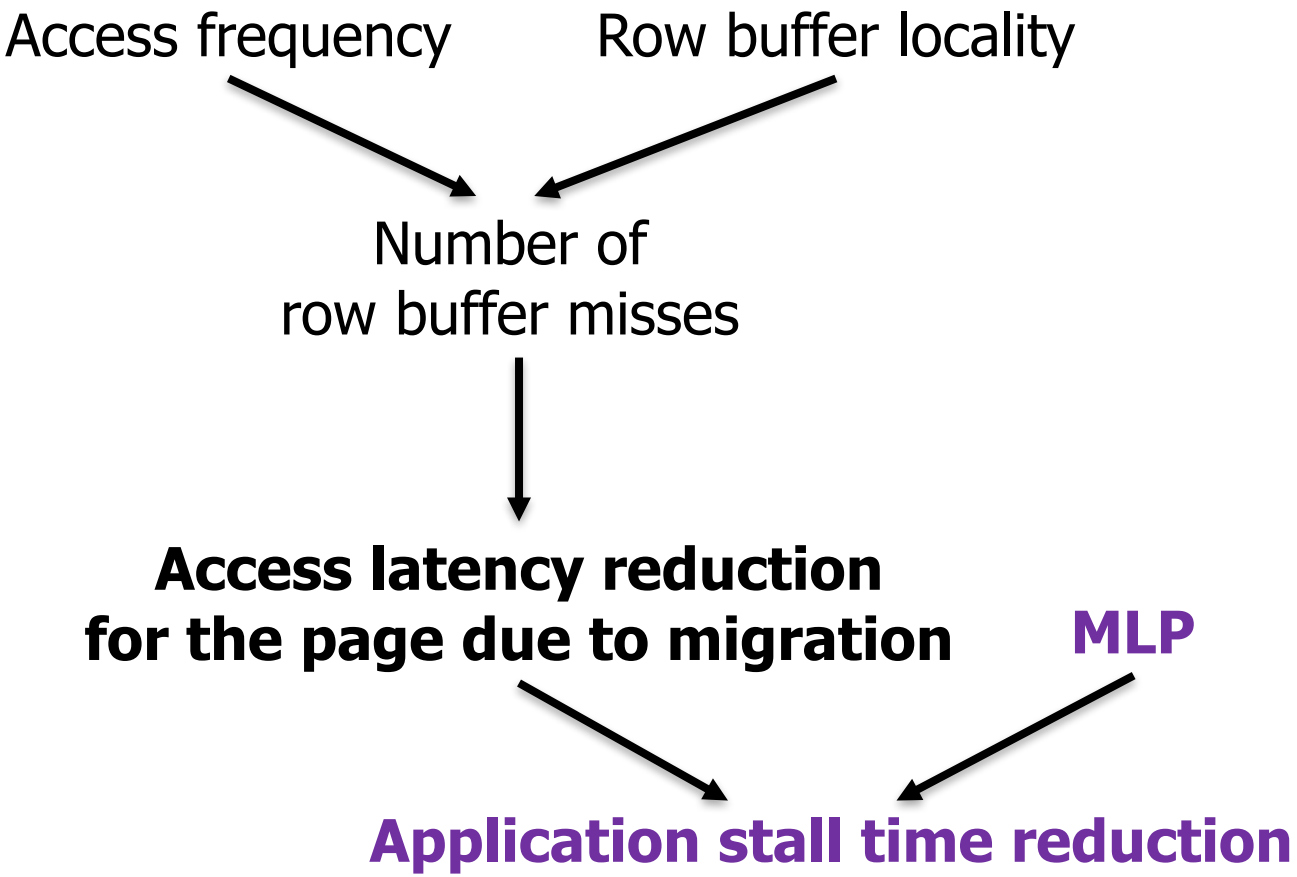
$$\Delta Access\ Latency_{read} = \#RB\ Miss_{read} \times (t_{Mem\ B,read} - t_{Mem\ A,read})$$

$$\Delta Access\ Latency_{write} = \#RB\ Miss_{write} \times (t_{Mem\ B,write} - t_{Mem\ A,write})$$

- Need **separate counters for reads and writes**
  - Each memory can have **different latencies** for reads and for writes
  - Allows us to generalize UH-MEM for *any* memory

Estimating Application Stall Time Reduction

- Use **access frequency**, **row buffer locality**, and **MLP** to estimate application's stall time reduction



Calculating Stall Time Reduction

- **MLP** characterizes number of accesses that **overlap**
- **How do overlaps affect application performance?**

More requests overlap → **reductions overlap → smaller improvement** from migrating the page

Total number of cycles reduced for all requests

$$\Delta Stall\ Time_i = \Delta Access\ Latency_{read} / MLP_{read} + p \times \Delta Access\ Latency_{write} / MLP_{write}$$

Not all writes fall on **critical path** of execution due to write queues inside memory:
p is probability that **application stalls when queue is being drained**

- Determining MLP_{read} and MLP_{write} for each page:
  - Count concurrent read/write requests from the same application
  - Amount of concurrency may vary across time → use **average concurrency** over an interval

Key Mechanisms of UH-MEM

- For each page, estimate **utility** using a **performance model**
  - **Application stall time reduction**

    How much would migrating a page benefit the performance of the application that the page belongs to?
  - **Application performance sensitivity**

    How much does the improvement of a single application's performance increase the *overall* system performance?

$$Utility = \Delta StallTime_i \times Sensitivity_i$$

- **Migrate** only pages whose utility exceeds the migration threshold from slow memory to fast memory
- Periodically **adjust migration threshold**

Estimating Application Performance Sensitivity

- Objective: **improve system job throughput**
- **Weighted speedup** for multiprogrammed workloads
  - Number of jobs (i.e., applications) completed per unit time [Eyerman+ IEEE Micro 2008]
  - For each application *i*, $Speedup_i = \frac{T_{alone,i}}{T_{shared,i}}$
  - $Weighted\ Speedup = \sum_{i=1}^{N} Speedup_i$

How does **each application** contribute to the weighted speedup?

Estimating Application Performance Sensitivity

- **Sensitivity**: for each cycle of stall time saved, how much does our target metric improve by?

How much of the weighted speedup
did the application save in the last interval?

$$Sensitivity_i = \frac{\Delta Performance_i}{\Delta T_{shared,i}} = \frac{Speedup_i}{T_{shared,i}}$$

Derived mathematically (see paper for details)

Number of cycles saved
in last interval

- $T_{shared,i}$ can be counted at runtime
- $Speedup_i$ can be estimated using previously-proposed models
  [Moscibroda+ USENIX Security 2007], [Mutlu+ MICRO 2007], [Ebrahimi+ ASLPLOS 2010], [Ebrahimi+ ISCA 2011], [Subramanian+ HPCA 2013], [Subramanian+ MICRO 2015]

Key Mechanisms of UH-MEM

- For each page, estimate **utility** using a **performance model**
  - **Application stall time reduction**

    How much would migrating a page benefit the performance of the application that the page belongs to?
  - **Application performance sensitivity**

    How much does the improvement of a single application's performance increase the *overall* system performance?

$$Utility = \Delta StallTime_i \times Sensitivity_i$$

- **Migrate** only pages whose utility exceeds the migration threshold from slow memory to fast memory
- Periodically **adjust migration threshold**
  - Hill-climbing algorithm to balance migrations and channel load
  - More details in the paper

Outline

- Background
- Existing Solutions
- UH-MEM: Goal and Key Idea
- Key Mechanisms of UH-MEM
- **Evaluation**
- Conclusion

System Configuration

- Cycle-accurate hybrid memory simulator
  - Models memory system in details
  - **8 cores**, 2.67GHz
  - LLC: 2MB, 8-way, 64B cache block
  - **We will release the simulator on GitHub:** https://github.com/CMU-SAFARI/UHMEM
- Baseline configuration (DRAM-NVM)
  - Memory latencies based on real products, prior studies

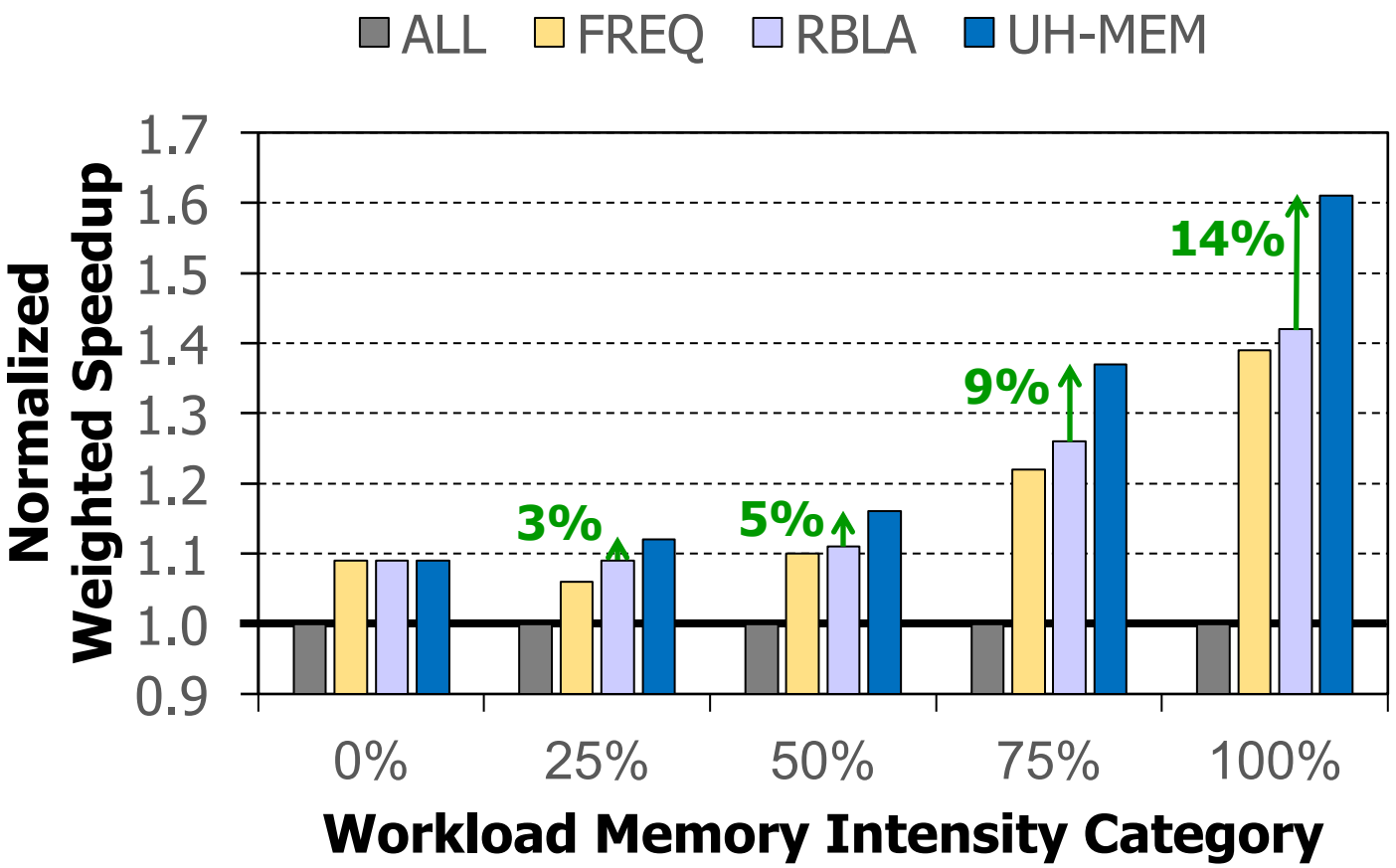
Memory Timing Parameter	DRAM	NVM
row activation time (t_{RCD})	**15 ns**	**67.5 ns**
column access latency (t_{CL})	15 ns	15 ns
write recovery time (t_{WR})	**15 ns**	**180 ns**
precharge latency (t_{RP})	15 ns	15 ns

  - Energy numbers derived from prior works [Lee+ ISCA 2009]

Benchmarks

- Individual applications
  - From SPEC CPU 2006, YCSB benchmark suites
  - Classify applications as **memory-intensive** or **non-memory- intensive** based on last-level cache **MPKI** (misses per kilo-instruction)
- Generate **40 multiprogrammed workloads**, each consisting of 8 applications
- Workload **memory intensity**: the ***proportion*** of memory-intensive applications within the workload

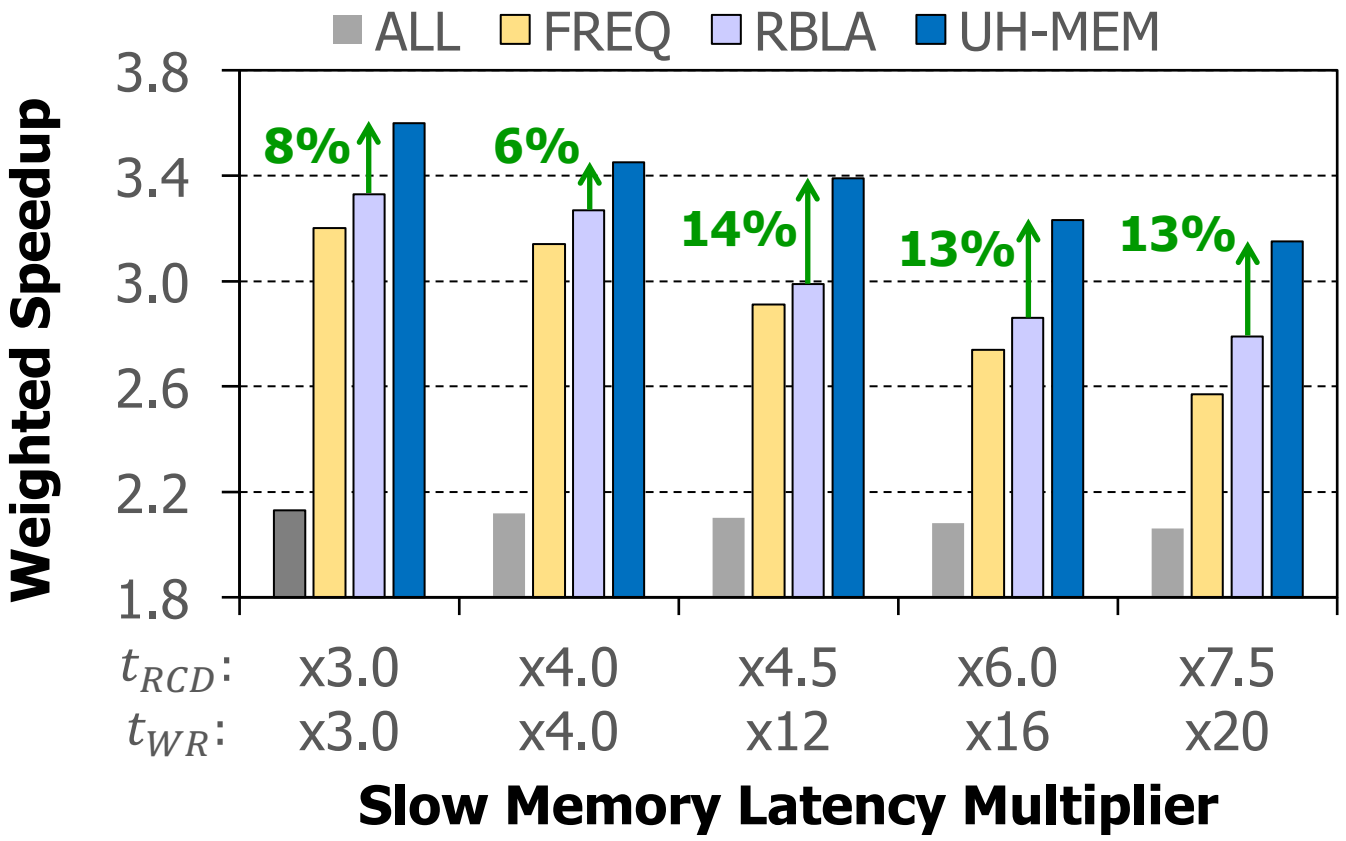
Results: System Performance



UH-MEM improves system performance
over the best state-of-the-art hybrid memory manager

Results: Sensitivity to Slow Memory Latency

- We vary t_{RCD} and t_{WR} of the slow memory



UH-MEM improves system performance for a wide variety of hybrid memory systems

More Results in the Paper

- UH-MEM **reduces energy consumption**, **achieves similar or better fairness** compared with prior proposals

- Sensitivity study on size of fast memory
  - We vary the size of fast memory from 256MB to 2GB
  - UH-MEM **consistently outperforms prior proposals**

- Hardware overhead: 42.9kB (~2% of last-level cache)
  - Main overhead: hardware structure to store access frequency, row buffer locality, and MLP

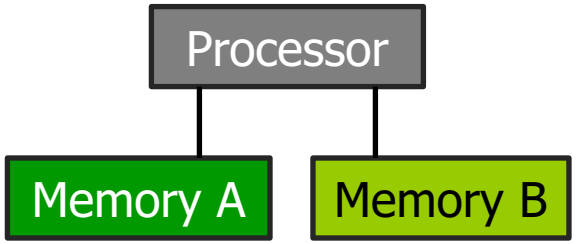
Outline

- Background
- Existing Solutions
- UH-MEM: Goal and Key Idea
- Key Mechanisms of UH-MEM
- Evaluation
- **Conclusion**

Conclusion

- DRAM faces **significant technology scaling difficulties**
- Emerging memory technologies overcome difficulties (e.g., **high capacity, low idle power**), but have other shortcomings (e.g., **slower than DRAM**)

- **Hybrid memory system** pairs DRAM with emerging memory technology
  - Goal: combine **benefits of both memories** in a cost-effective manner
  - **Problem: Which memory do we place each page in, to optimize system performance?**



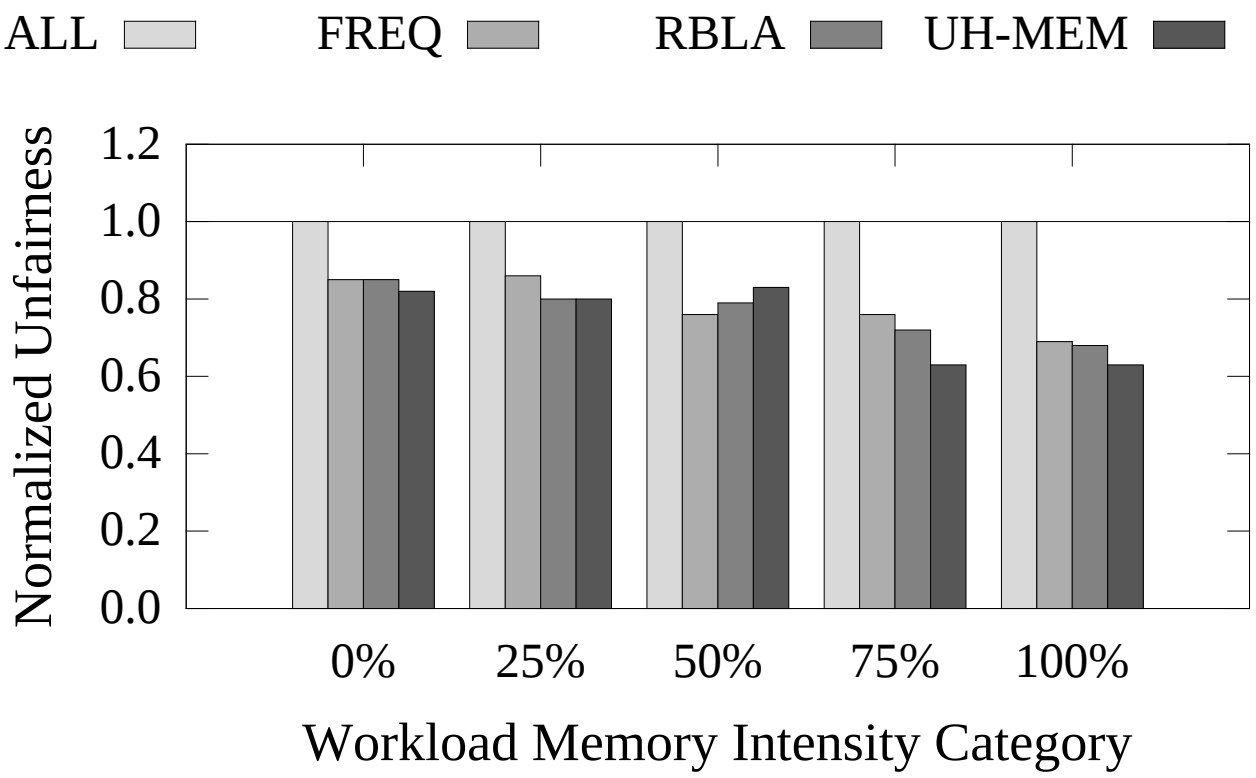
- Our approach: **UH-MEM** (Utility-based Hybrid MEmory Management)
  - **Key Idea:** for each page, estimate ***utility*** (i.e., performance impact) of migrating page, then use utility to guide page placement
  - **Key Mechanism:** a **comprehensive model** to estimate utility using memory access characteristics, application impact on system performance

- UH-MEM **improves performance by 14%** on average over the best of three state-of-the-art hybrid memory managers

UH-MEM: Utility-Based Hybrid Memory Management

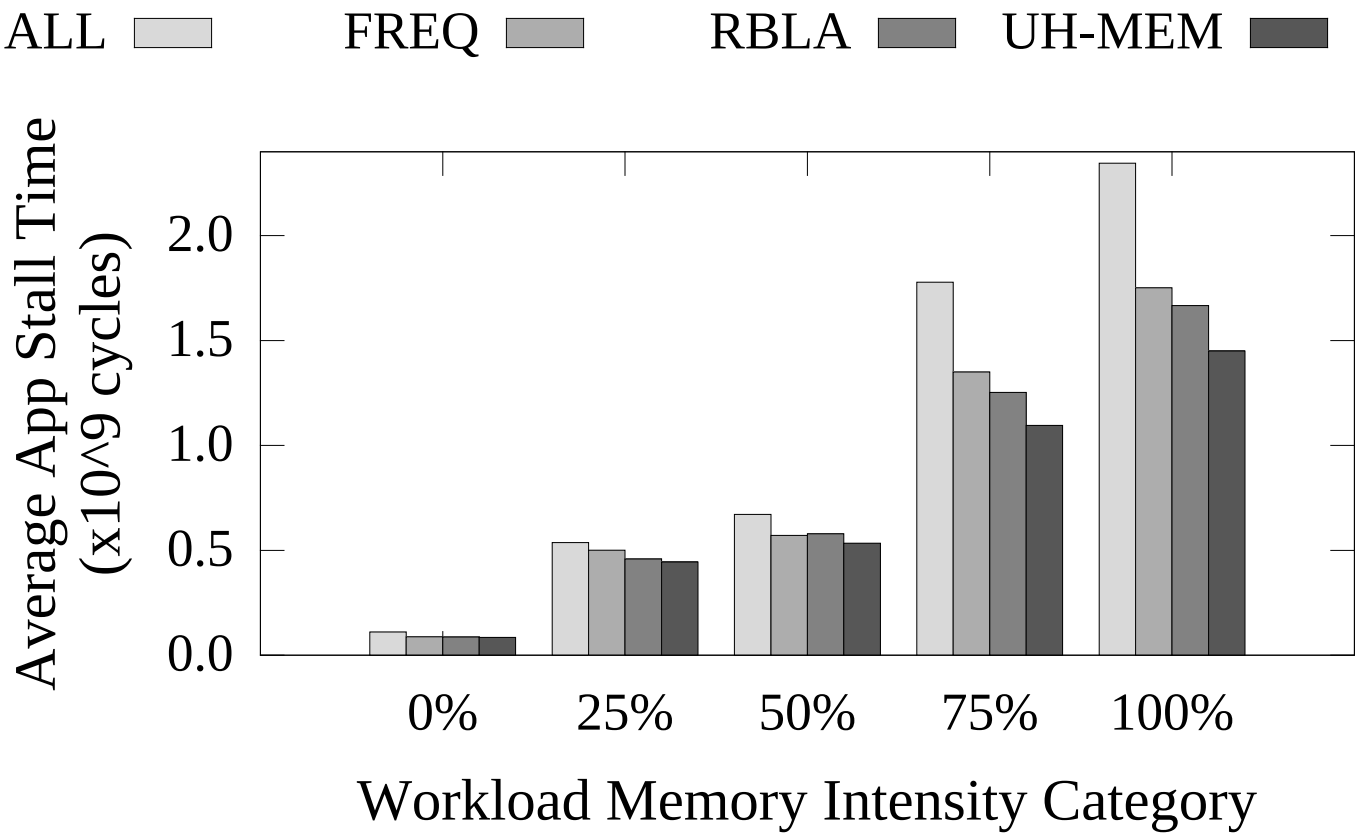
Yang Li, Saugata Ghose, Jongmoo Choi,
Jin Sun, Hui Wang, Onur Mutlu

Results: Fairness



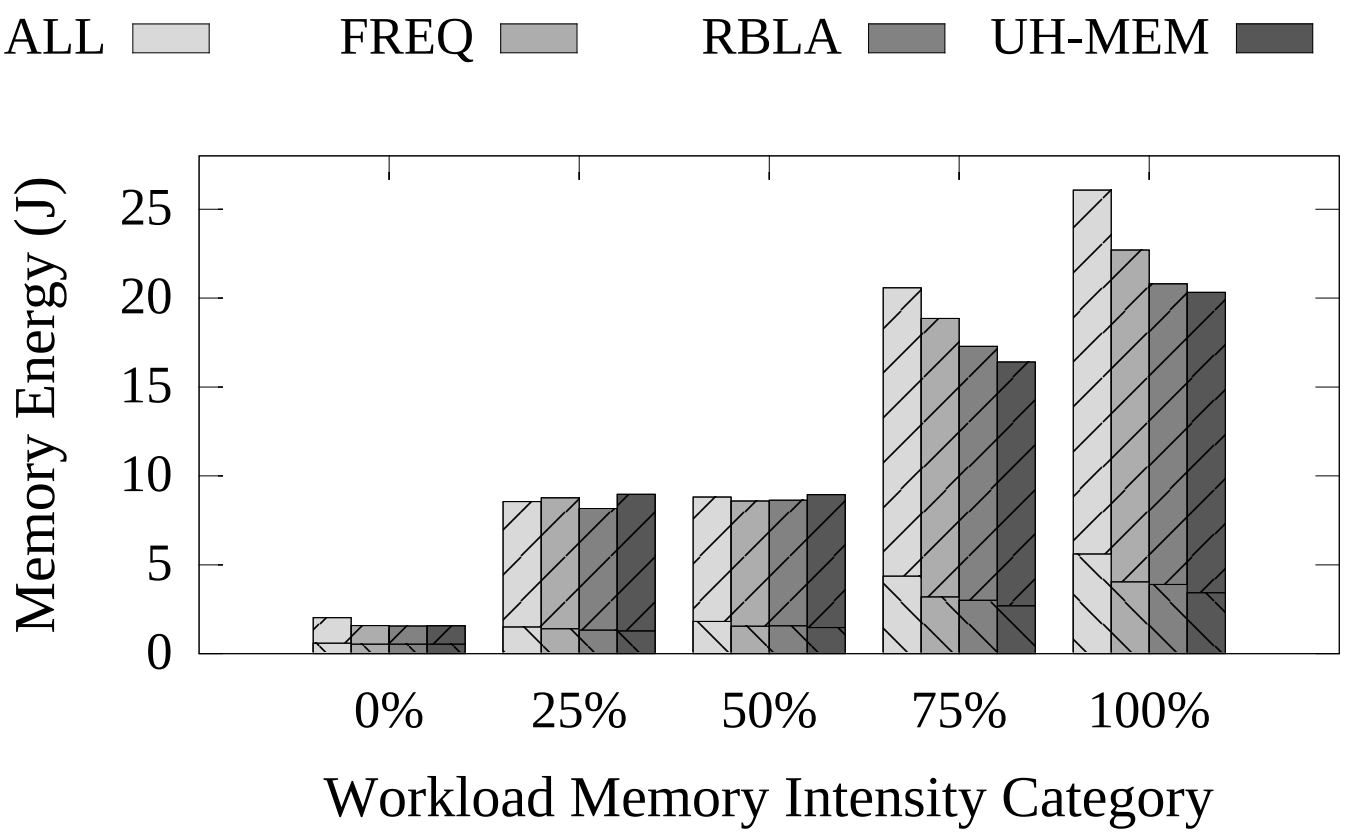
UH-MEM achieves similar or better fairness compared with prior proposals

Results: Total Stall Time



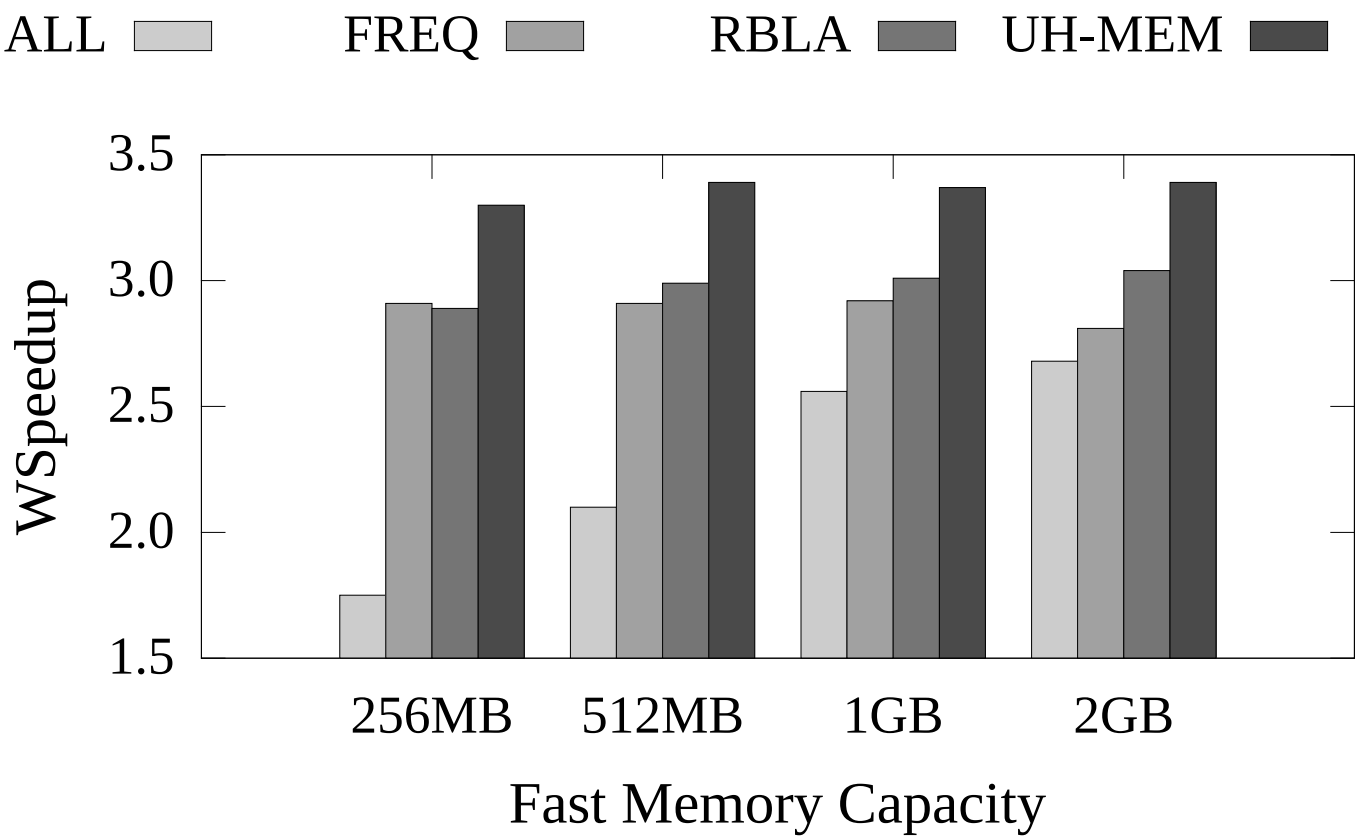
UH-MEM reduces application stall time compared with prior proposals

Results: Energy Consumption



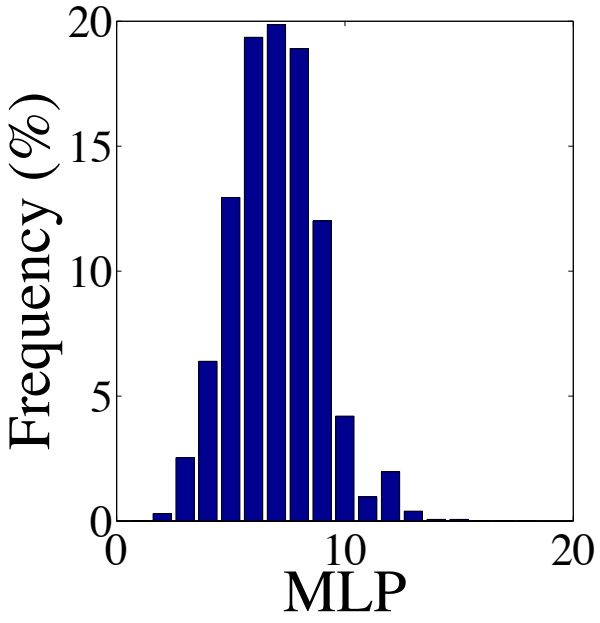
UH-MEM reduces energy consumption for data-intensive workloads

Results: Sensitivity to Fast Memory Capacity

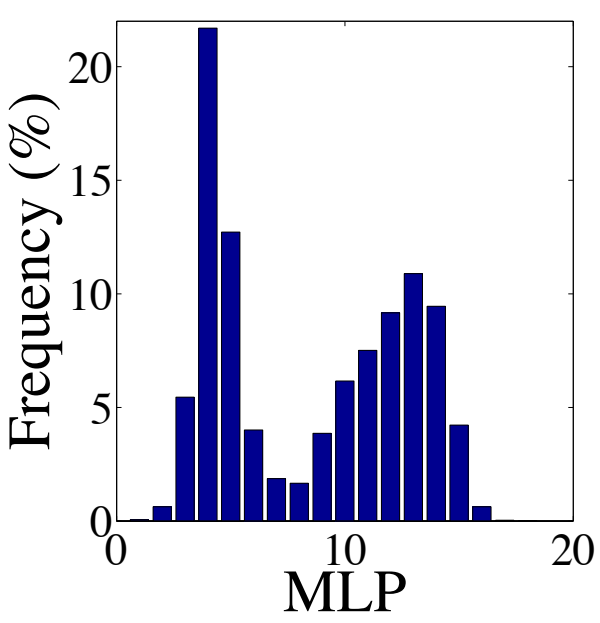


UH-MEM outperforms prior proposals under different fast memory capacities

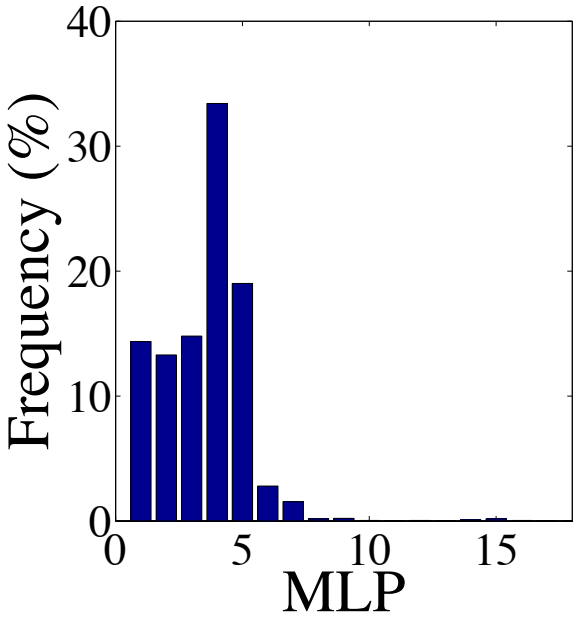
MLP Distribution for All Pages



(a) soplex



(b) xalancbmk



(c) YCSB-B

Main Hardware Cost

Name	**Purpose**	**Structure** (number of bits in parentheses)	**Size**
Stats store	Tracks statistical information for recently-accessed pages	2048 entries; each entry consists of read row buffer miss count (14), write row miss count (14), $MLPAcc_{read}$ (30), $MLPAcc_{write}$ (30), $MLPWeight_{read}$ (21), $MLPWeight_{write}$ (21) and page number tag (30)	40.00KB
Counters for outstanding pages in slow memory	Records updates of *MLPAcc* and *MLPWeight* for pages with outstanding requests	For each page with outstanding requests in slow memory (96 at most), $MLPAcc_{read}$ (30), $MLPAcc_{write}$ (30), $MLPWeight_{read}$ (21), $MLPWeight_{write}$ (21) and page number (36)	1.62KB
ROM table for MLP ratios	Stores precomputed results of division used to calculate MLP ratios	32 x 32 entries; each entry consumes 10 bits	1.25KB
Total Hardware Cost (for our evaluated system in Table 3)			42.87KB

Baseline System Parameters

Processor	8 cores, 2.67GHz, 3-wide issue, 128-entry instruction window
L1 Cache	32KB per core, 4-way, 64B cache block
L2 Cache	256KB per core, 8-way, 32 MSHR entries per core, 64B cache block
Fast Memory Controller	64-bit channel, 64-entry read request queue, 32-entry write buffer, FR-FCFS scheduling policy [108, 132]
Slow Memory Controller	64-bit channel, 64-entry read request queue, 32-entry write buffer, FR-FCFS scheduling policy [108, 132]
Baseline Fast Memory System	512MB DRAM, 1 rank (8 banks), t_{CLK}=1.875ns, t_{CL}=15ns, t_{RCD}=15ns, t_{RP}=15ns, t_{WR}=15ns, array read (write) energy = 1.17 (0.39) pJ/bit, row buffer read (write) energy = 0.93 (1.02) pJ/bit
Baseline Slow Memory System	16GB NVM, 1 rank (8 banks), t_{CLK}=1.875ns, t_{CL}=15ns, t_{RCD}=67.5ns, t_{RP}=15ns, t_{WR}=180ns, array read (write) energy = 2.47 (16.82) pJ/bit, row buffer read (write) energy = 0.93 (1.02) pJ/bit

Impact of Different Factors on Stall Time

Correlation coefficients between the average stall time per page and different factors (AF: access frequency; RBL: row buffer locality; MLP: memory level parallelism)

	AF	RBL	MLP
Correlation	0.74	0.59	0.54

	AF+RBL	AF+MLP	AF+RBL+MLP
Correlation	0.76	0.86	0.92

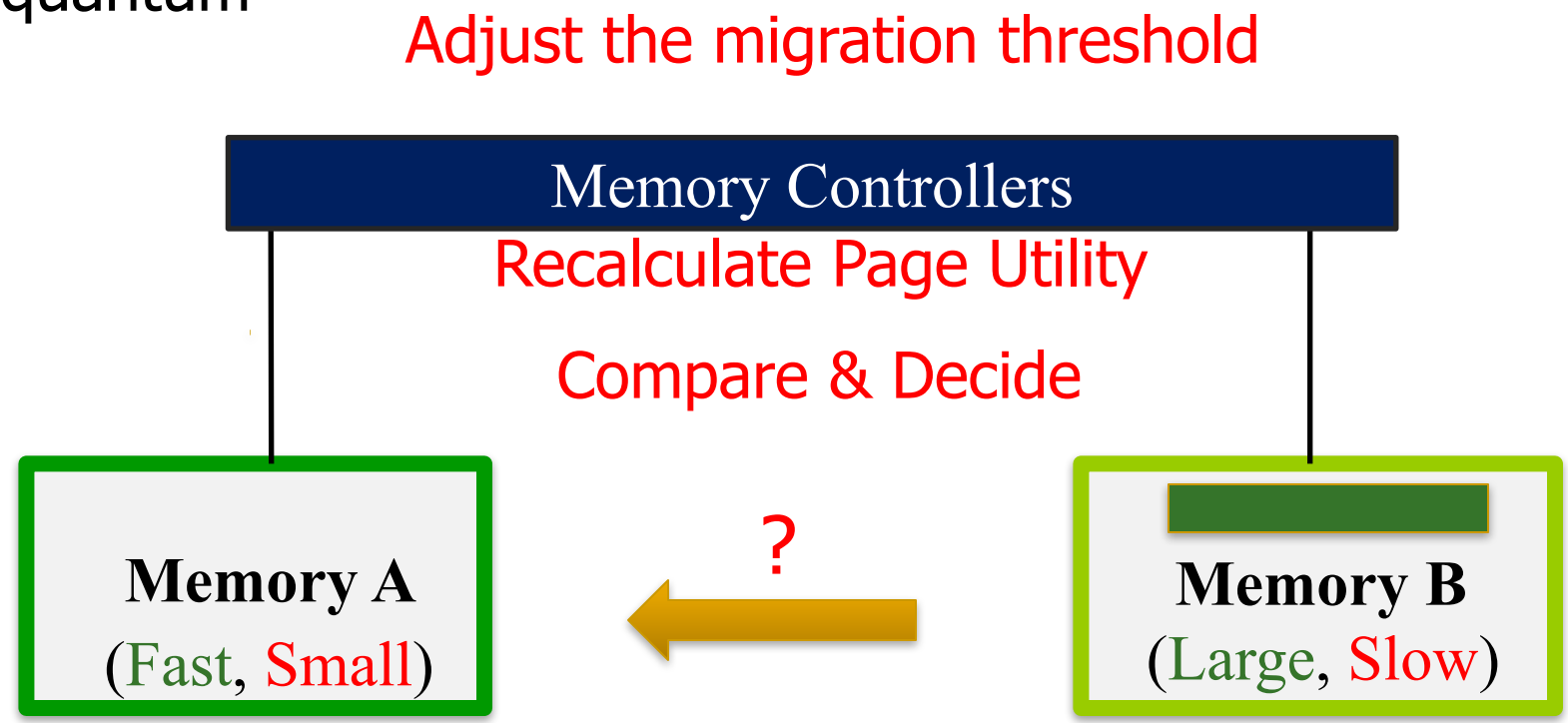
Migration Threshold Determination

- Use hill climbing to determine the threshold
- At the end of the current period,

  compare $Performance_{Current\ Period}$ with $Performance_{Last\ Period}$
- If the system performance improves,
  - the threshold adjustment at the end of last period benefits system performance
  - adjust the threshold in the same direction
- Otherwise,
  - adjust the threshold in the opposite direction

Overall UH-MEM Mechanism

- Period-based approach - each period has a migration threshold
- Action 1: Recalculate the page's utility
- Action 2: Compare the page's utility with the migration threshold, and decide whether to migrate
- Action 3: Adjust the migration threshold at the end of a quantum



UH-MEM: Putting It All Together

Algorithm 1 Migrating pages with UH-MEM.

```
1:  for every interval do
2:      for every completed memory request do
3:          Update the corresponding page's statistics counters
4:          Calculate the page's utility (Section 4.2)
5:          if the page's utility exceeds the migration threshold
            then
6:              Migrate the page to the fast memory
7:          end if
8:      end for
9:      if at the end of the interval then
10:         Adjust the migration threshold (Section 4.3)
11:         Estimate speedup for each application (Section 4.2.2)
12:         Reset all counters to zero
13:     end if
14: end for
```