

Computer Architecture

Lecture 17: Memory Interference and Quality of Service II

Prof. Onur Mutlu

ETH Zürich

Fall 2018

21 November 2018

Lecture Announcement

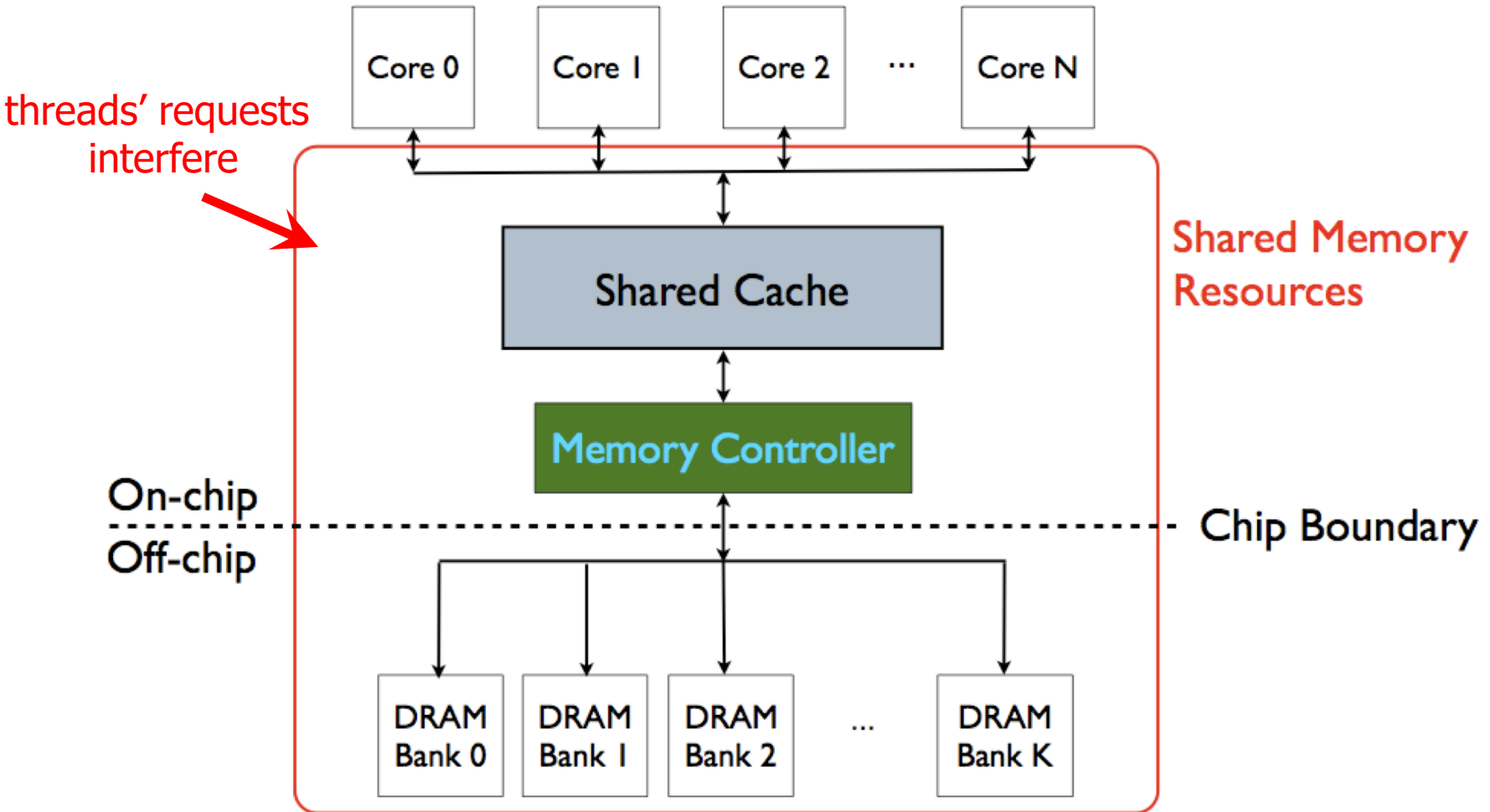
- Monday, November 26, 2018
- 16:15-17:15
- CAB G 61
- Apéro after the lecture 😊



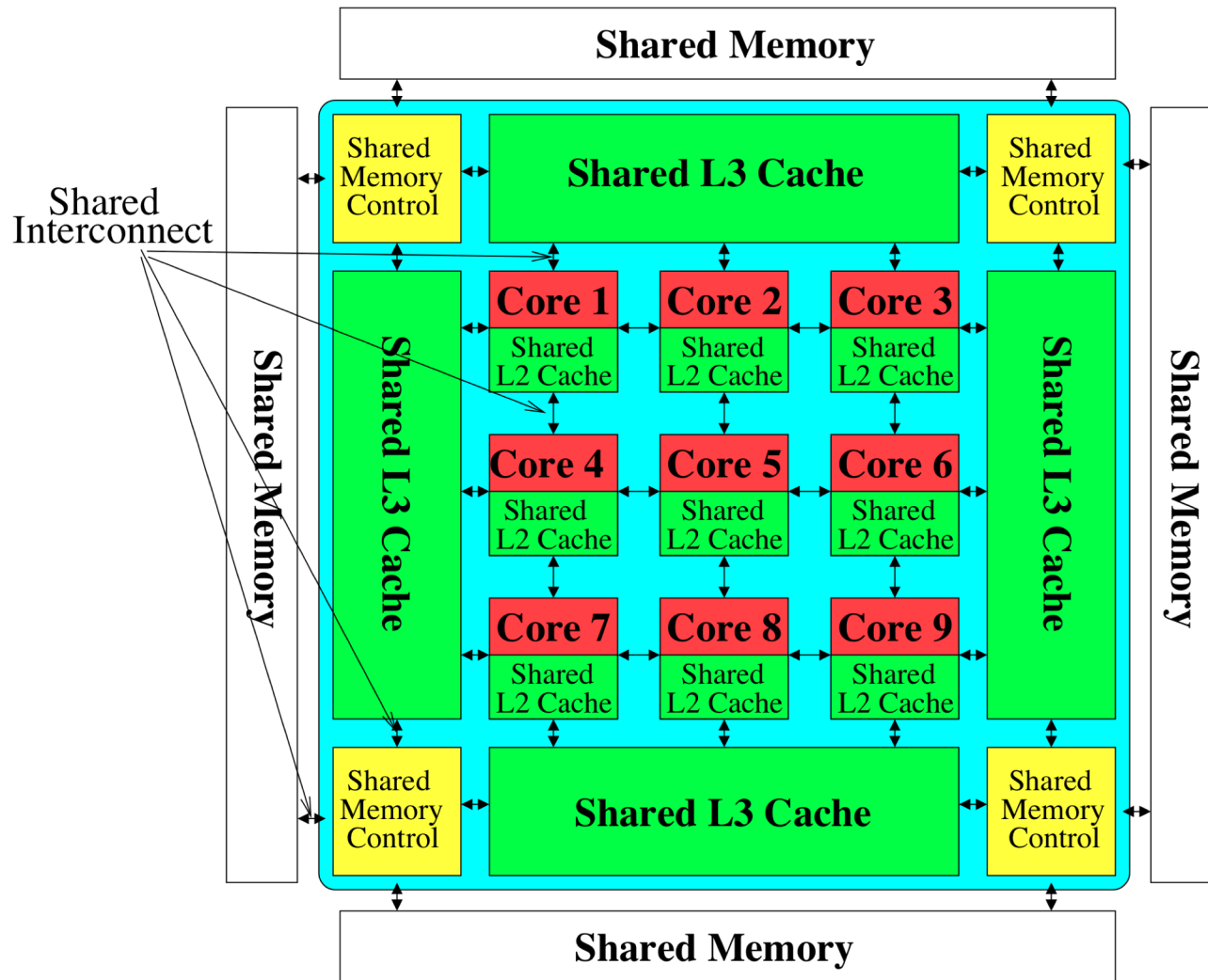
- Prof. Arvind (Massachusetts Institute of Technology)
- D-INFK Distinguished Colloquium
- **The Risky Expedition**

- <https://www.inf.ethz.ch/news-and-events/colloquium/event-detail.html?eventFeedId=42658>

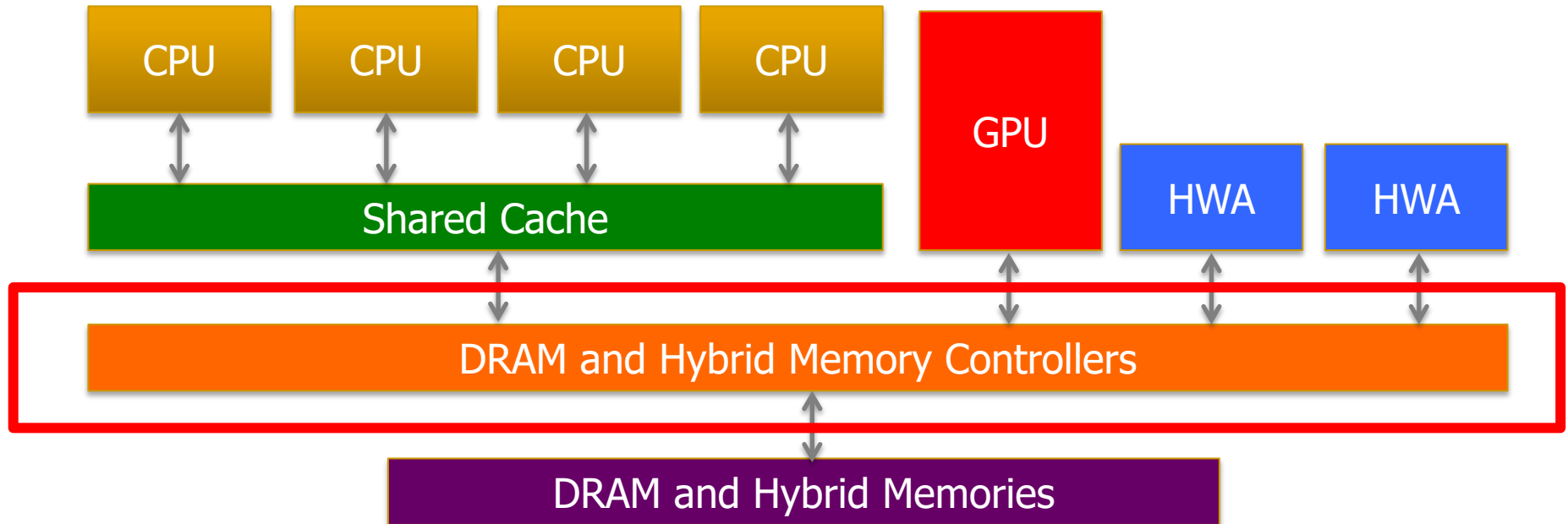
Memory System is the Major Shared Resource



Much More of a Shared Resource in Future



Memory System Shared by Heterogeneous Agents



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs

How to allocate resources to heterogeneous agents to mitigate interference and provide predictable performance?

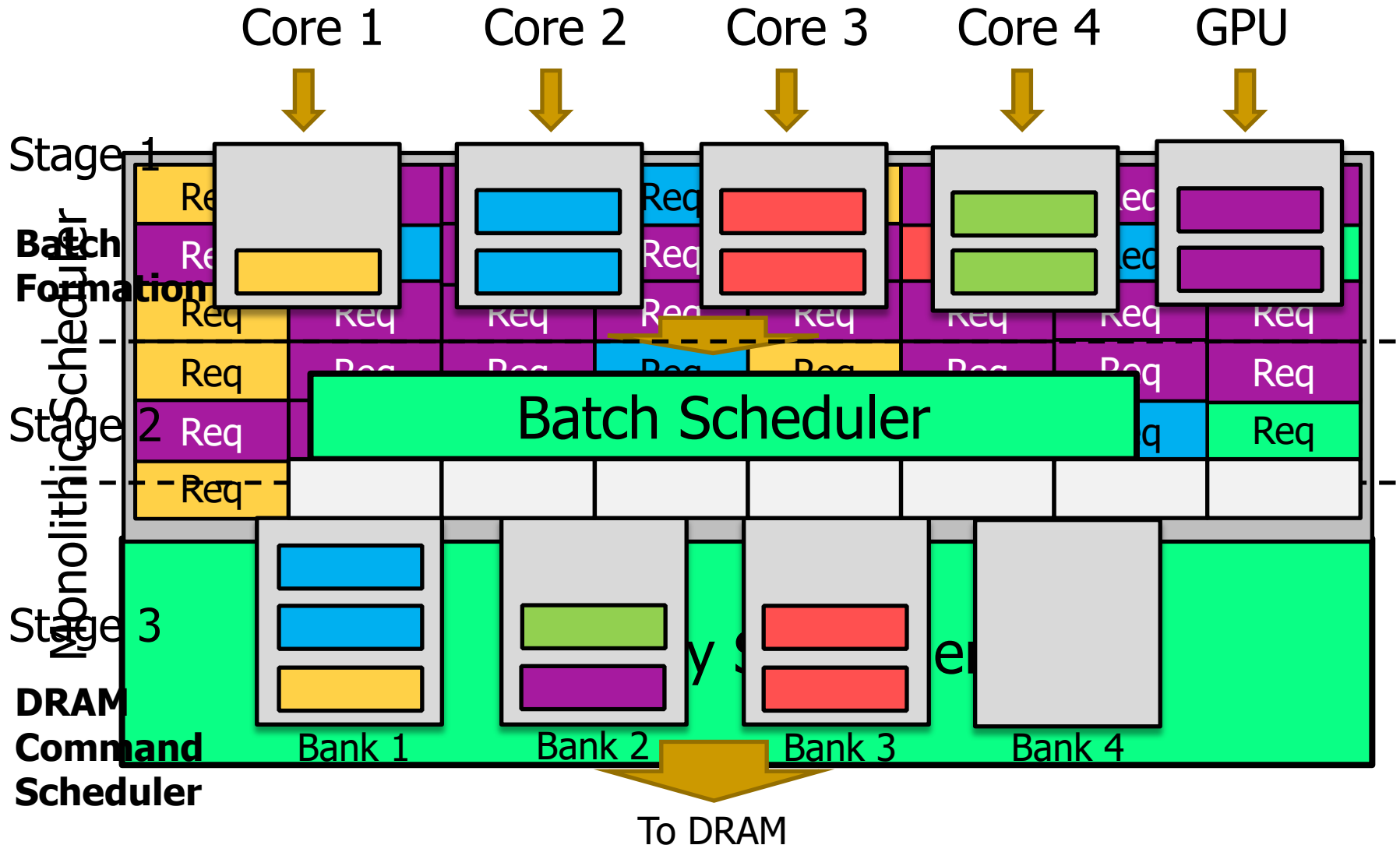
Staged Memory Scheduling

Rachata Ausavarungnirun, Kevin Chang, Lavanya Subramanian, Gabriel Loh, and Onur Mutlu,
**"Staged Memory Scheduling: Achieving High Performance
and Scalability in Heterogeneous Systems"**
39th International Symposium on Computer Architecture (ISCA),
Portland, OR, June 2012.

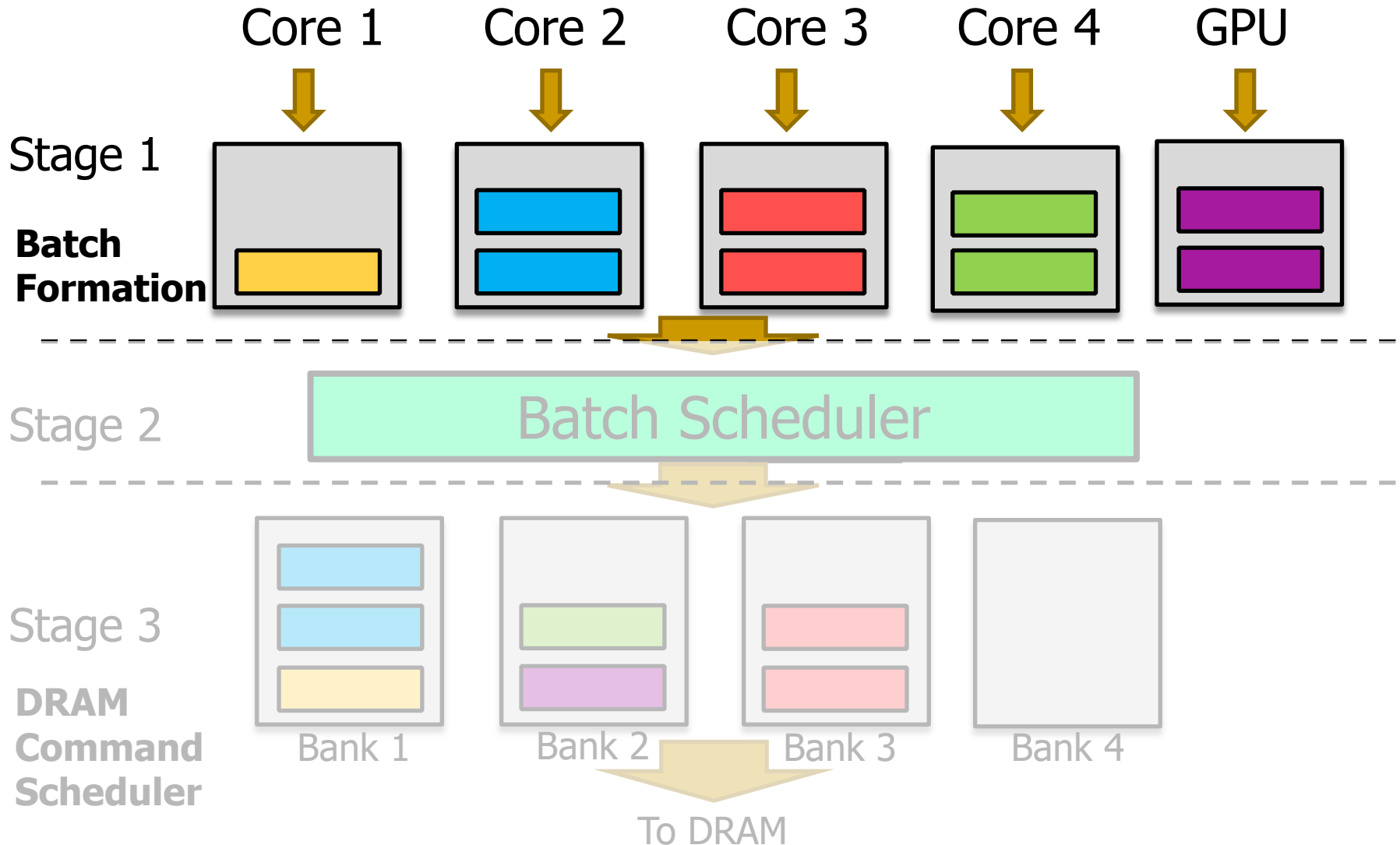
SMS: Executive Summary

- **Observation:** Heterogeneous CPU-GPU systems require memory schedulers with **large request buffers**
- **Problem:** Existing monolithic application-aware memory scheduler designs are **hard to scale** to large request buffer sizes
- **Solution:** Staged Memory Scheduling (SMS)
decomposes the memory controller into three simple stages:
 - 1) Batch formation: maintains row buffer locality
 - 2) Batch scheduler: reduces interference between applications
 - 3) DRAM command scheduler: issues requests to DRAM
- Compared to state-of-the-art memory schedulers:
 - SMS is significantly simpler and more scalable
 - SMS provides higher performance and fairness

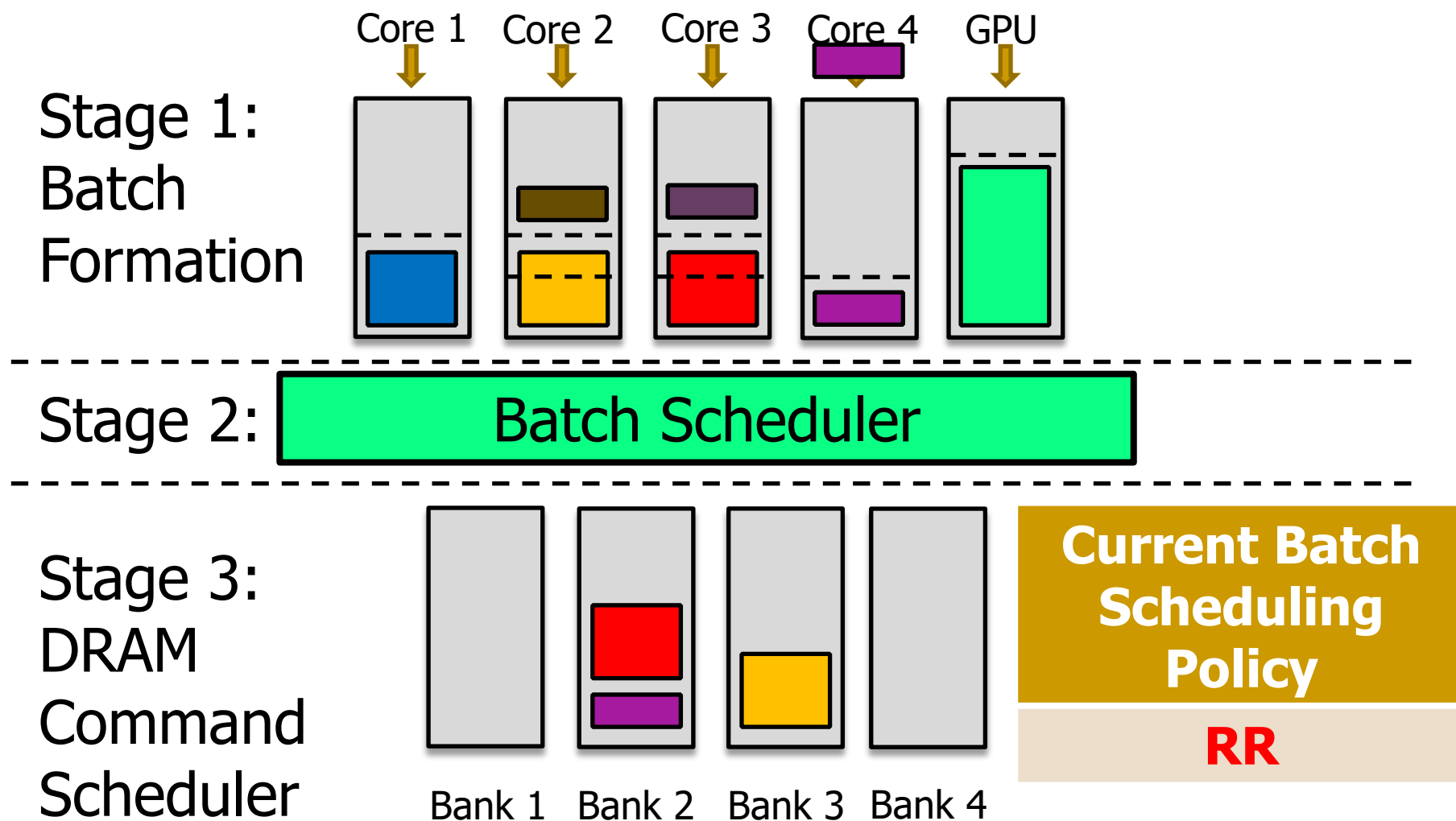
SMS: Staged Memory Scheduling



SMS: Staged Memory Scheduling



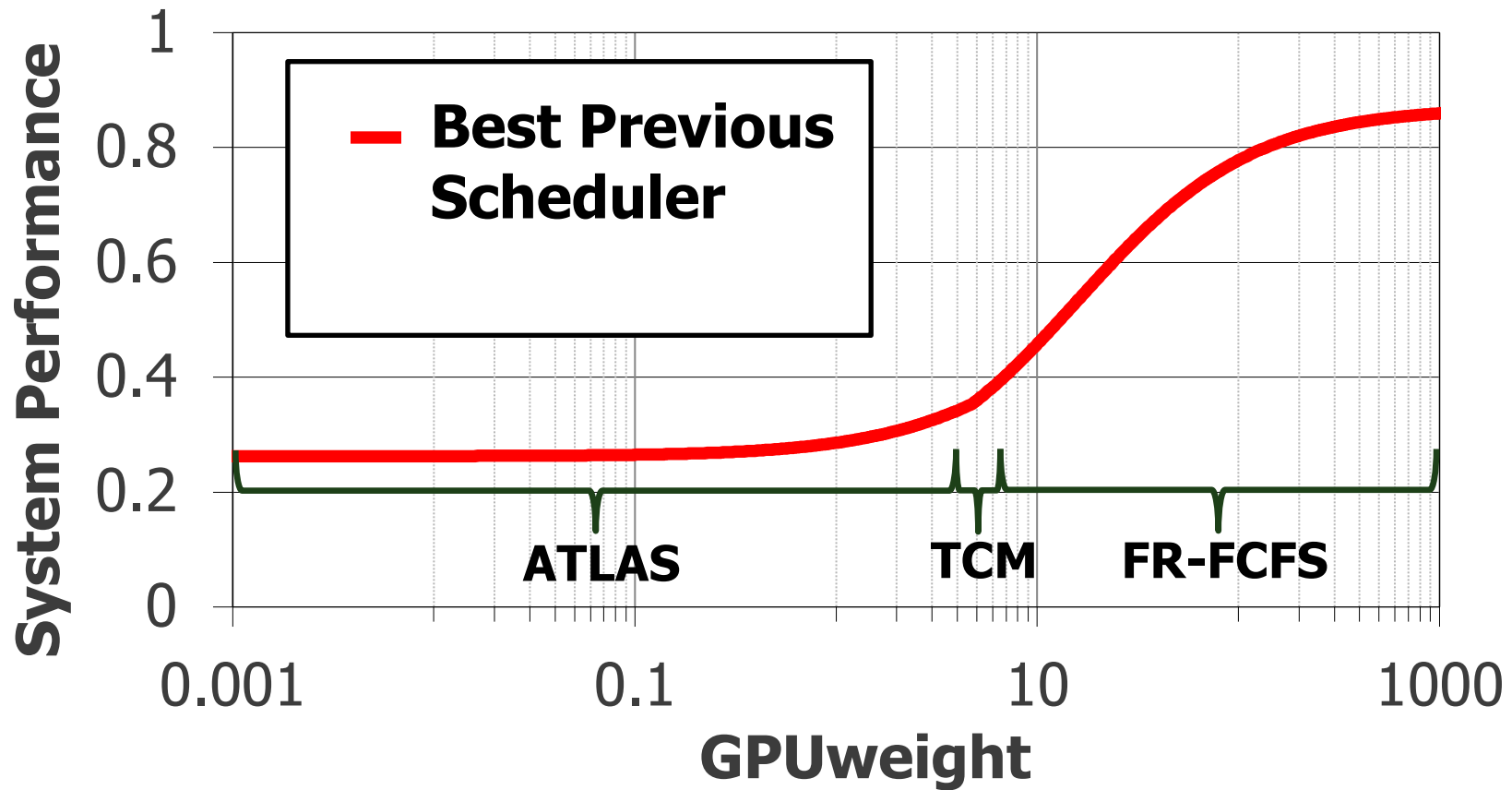
Putting Everything Together



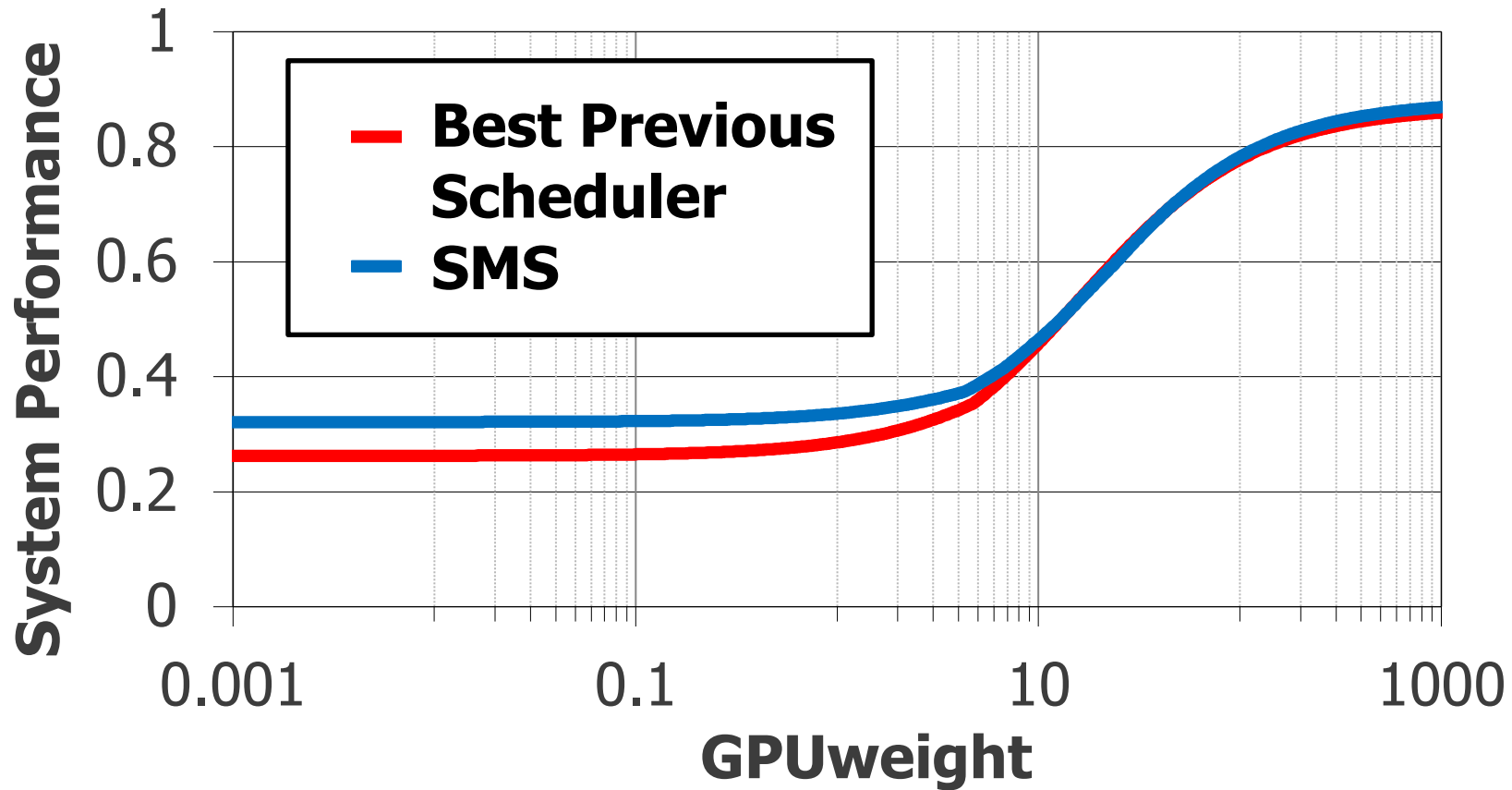
Complexity

- Compared to a row hit first scheduler, SMS consumes*
 - ❑ 66% less area
 - ❑ 46% less static power
- Reduction comes from:
 - ❑ Monolithic scheduler → stages of simpler schedulers
 - ❑ Each stage has a simpler scheduler (considers fewer properties at a time to make the scheduling decision)
 - ❑ Each stage has simpler buffers (FIFO instead of out-of-order)
 - ❑ Each stage has a portion of the total buffer size (buffering is distributed across stages)

Performance at Different GPU Weights



Performance at Different GPU Weights



- At every GPU weight, SMS outperforms the best previous scheduling algorithm for that weight

More on SMS

- Rachata Ausavarungnirun, Kevin Chang, Lavanya Subramanian, Gabriel Loh, and Onur Mutlu,
"Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems"
Proceedings of the 39th International Symposium on Computer Architecture (ISCA), Portland, OR, June 2012. [Slides \(pptx\)](#)

Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems

Rachata Ausavarungnirun[†] Kevin Kai-Wei Chang[†] Lavanya Subramanian[†] Gabriel H. Loh[‡] Onur Mutlu[†]

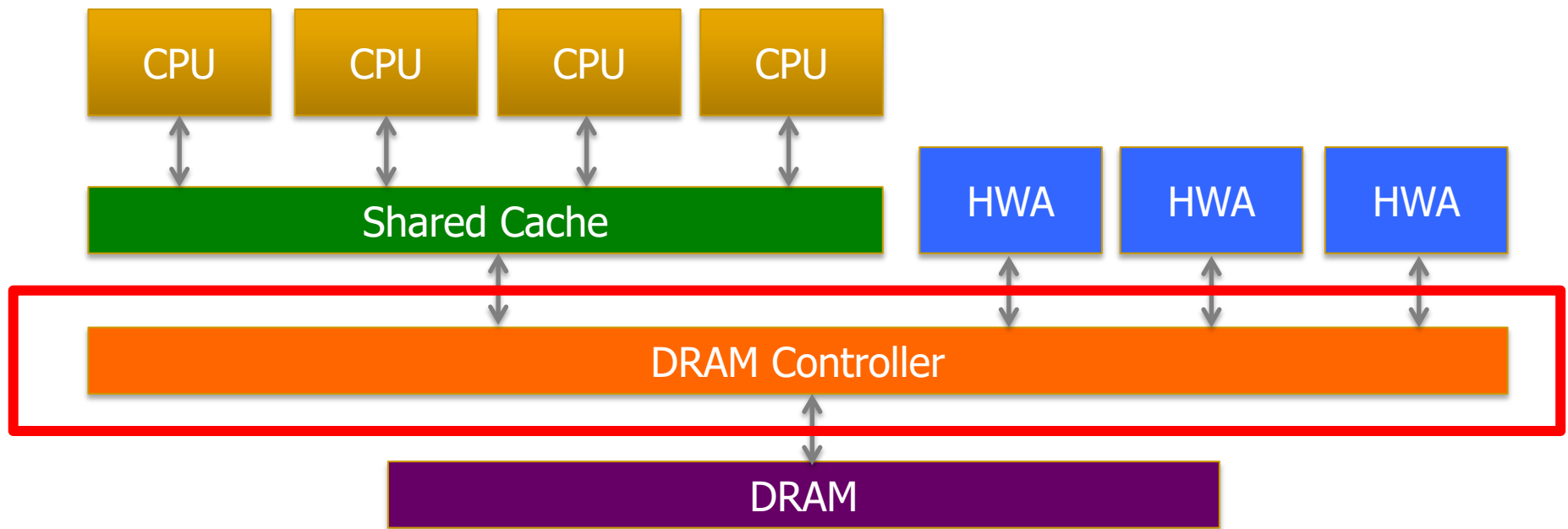
[†]Carnegie Mellon University
{rachata,kevincha,lsubrama,onur}@cmu.edu

[‡]Advanced Micro Devices, Inc.
gabe.loh@amd.com

DASH Memory Scheduler

[TACO 2016]

Current SoC Architectures



- Heterogeneous agents: CPUs and HWAs
 - HWA : Hardware Accelerator
- Main memory is shared by CPUs and HWAs → Interference

How to schedule memory requests from CPUs and HWAs to mitigate interference?

Example Heterogeneous SoC

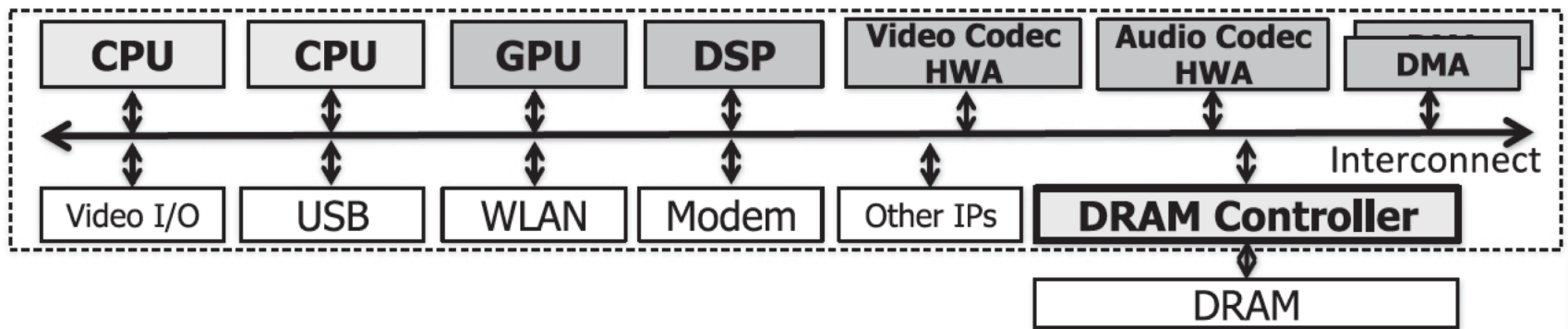


Fig. 1. Example heterogeneous SoC architecture.

DASH Scheduler: Executive Summary

- Problem: Hardware accelerators (HWAs) and CPUs share the same memory subsystem and interfere with each other in main memory
- Goal: Design a memory scheduler that improves CPU performance while meeting HWAs' deadlines
- Challenge: Different HWAs have different memory access characteristics and different deadlines, which current schedulers do not smoothly handle
 - ❑ Memory-intensive and long-deadline HWAs significantly degrade CPU performance *when they become high priority* (due to slow progress)
 - ❑ Short-deadline HWAs sometimes miss their deadlines *despite high priority*
- Solution: DASH Memory Scheduler
 - ❑ Prioritize HWAs over CPU anytime when the HWA is not making good progress
 - ❑ Application-aware scheduling for CPUs and HWAs
- Key Results:
 - 1) Improves CPU performance for a wide variety of workloads by 9.5%
 - 2) Meets 100% deadline met ratio for HWAs
- DASH source code freely available on our GitHub

Goal of Our Scheduler (DASH)

- **Goal:** Design a memory scheduler that
 - Meets GPU/accelerators' frame rates/deadlines *and*
 - Achieves high CPU performance
- **Basic Idea:**
 - *Different CPU applications and hardware accelerators have different memory requirements*
 - Track progress of different agents and prioritize accordingly

Key Observation:

Distribute Priority for Accelerators

- GPU/accelerators need priority to meet deadlines
- Worst case prioritization not always the best
- Prioritize when they are **not** on track to meet a deadline

Distributing priority over time mitigates impact of accelerators on CPU cores' requests

Existing QoS-Aware Scheduling Scheme

- **Dynamic Prioritization for a CPU-GPU System** [Jeong et al., DAC 2012]
 - Dynamically adjust GPU priority based on its progress
 - Lower GPU priority if GPU is making a good progress to achieve its target frame rate

- We apply this scheme for a wide variety of HWAs
 - Compare HWA's current progress against expected progress
 - **Current Progress** : $\frac{(\text{The number of finished memory requests for a period})}{(\text{The number of total memory requests for a period})}$
 - **Expected Progress** : $\frac{(\text{Elapsed cycles in a period})}{(\text{Total cycles in a period})}$
 - Every scheduling unit, dynamically adjust HWA priority
 - If **Expected Progress** > EmergentThreshold (=0.9) : **HWA > CPU**
 - If **(Current Progress)** > **(Expected Progress)** : **HWA < CPU**
 - If **(Current Progress)** <= **(Expected Progress)** : **HWA = CPU**

Problems in Dynamic Prioritization

■ Dynamic Prioritization for a CPU-HWA system

- Compares HWA's **current progress** against **expected progress**

- **Current Progress** : $\frac{(\text{The number of finished memory requests for a period})}{(\text{The number of total memory requests for a period})}$

- **Expected Progress** : $\frac{(\text{Elapsed cycles in a period})}{(\text{Total cycles in a period})}$

- Every scheduling unit, dynamically adjust HWA priority

- If **Expected Progress** > EmergentThreshold (=0.9) : **HWA > CPU**
- If **(Current Progress)** > **(Expected Progress)** : **HWA < CPU**
- If **(Current Progress)** <= **(Expected Progress)** : **HWA = CPU**

1. An HWA is prioritized over CPU cores *only when* it is closed to HWA's deadline

➡ **The HWA often misses deadlines**

2. This scheme does not consider the diverse memory access characteristics of CPUs and HWAs

- It treats each CPU and each HWA equally

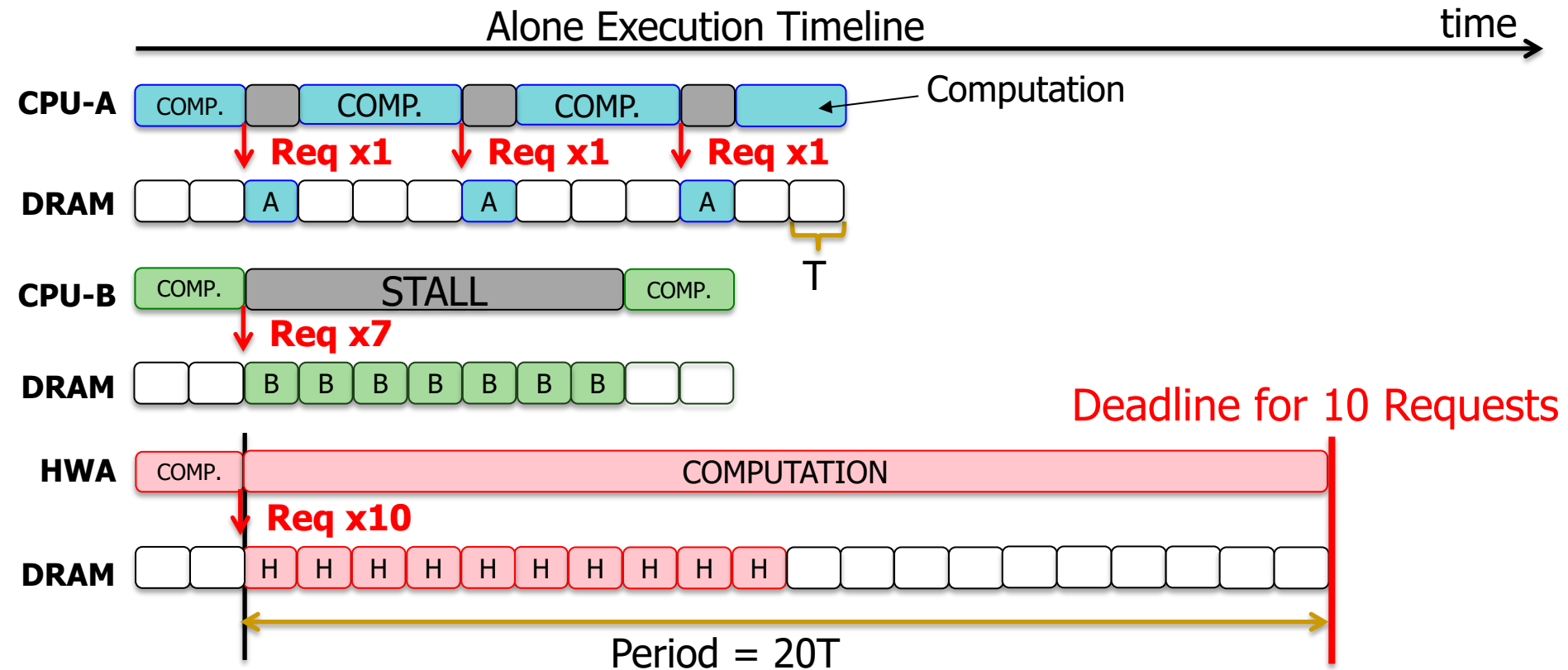
➡ **Missing opportunities to improve system performance**

Key Idea 1: Distributed Priority

- Problem 1: An HWA is prioritized over CPU cores *only when* it is close to HWA's deadline
 - Key Idea 1: **Distributed Prioritization for a CPU-HWA system**
 - Compares HWA's **current progress** against **expected progress**
 - **Current Progress** : $\frac{(\text{The number of finished memory requests for a period})}{(\text{The number of total memory requests for a period})}$
 - **Expected Progress** : $\frac{(\text{Elapsed cycles in a period})}{(\text{Total cycles in a period})}$
 - Dynamically adjust HWA priority based on its progress every scheduling unit
 - If **Expected Progress** > EmergentThreshold (=0.9) : **HWA > CPU**
 - If **(Current Progress) > (Expected Progress)** : **HWA < CPU**
 - If **(Current Progress) <= (Expected Progress)** : **HWA > CPU**
- Prioritize HWAs over CPU anytime when the HWA is not making good progress

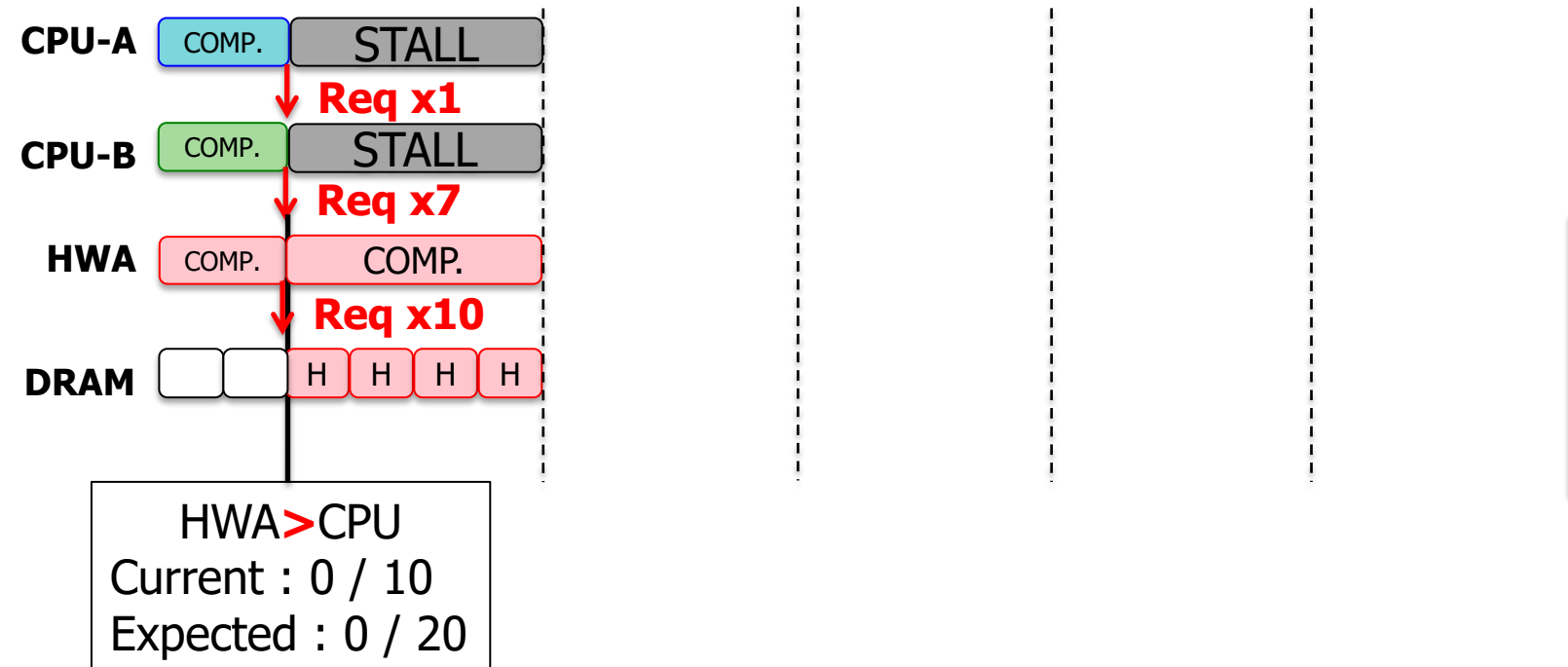
Example: Scheduling HWA and CPU Requests

- Scheduling requests from 2 CPU applications and a HWA
 - CPU-A : memory non-intensive application
 - CPU-B : memory intensive application



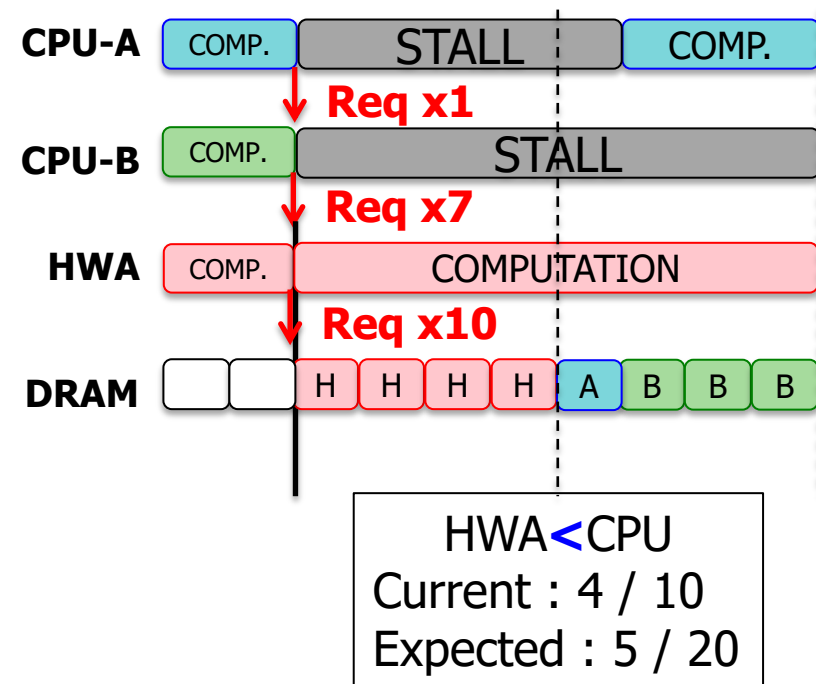
DASH: Distributed Priority

- **Distributed Priority (Scheduling unit = 4T)**



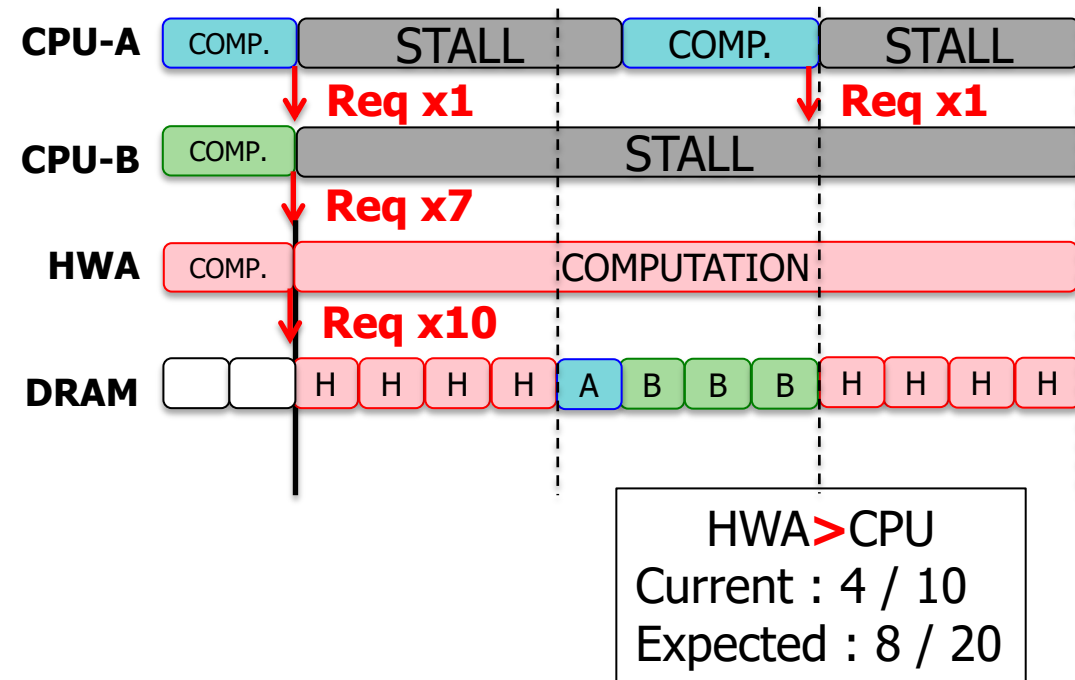
DASH: Distributed Priority

■ Distributed Priority (Scheduling unit = 4T)



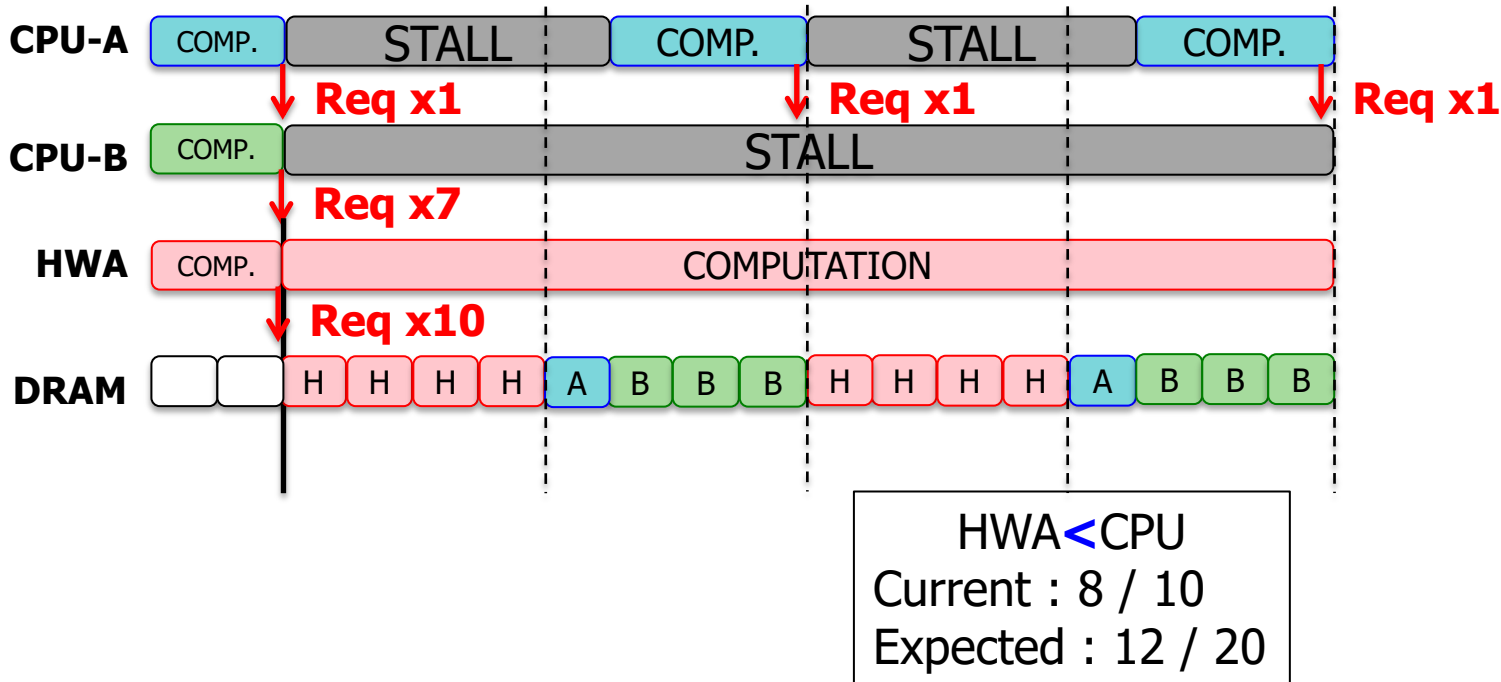
DASH: Distributed Priority

■ Distributed Priority (Scheduling unit = 4T)



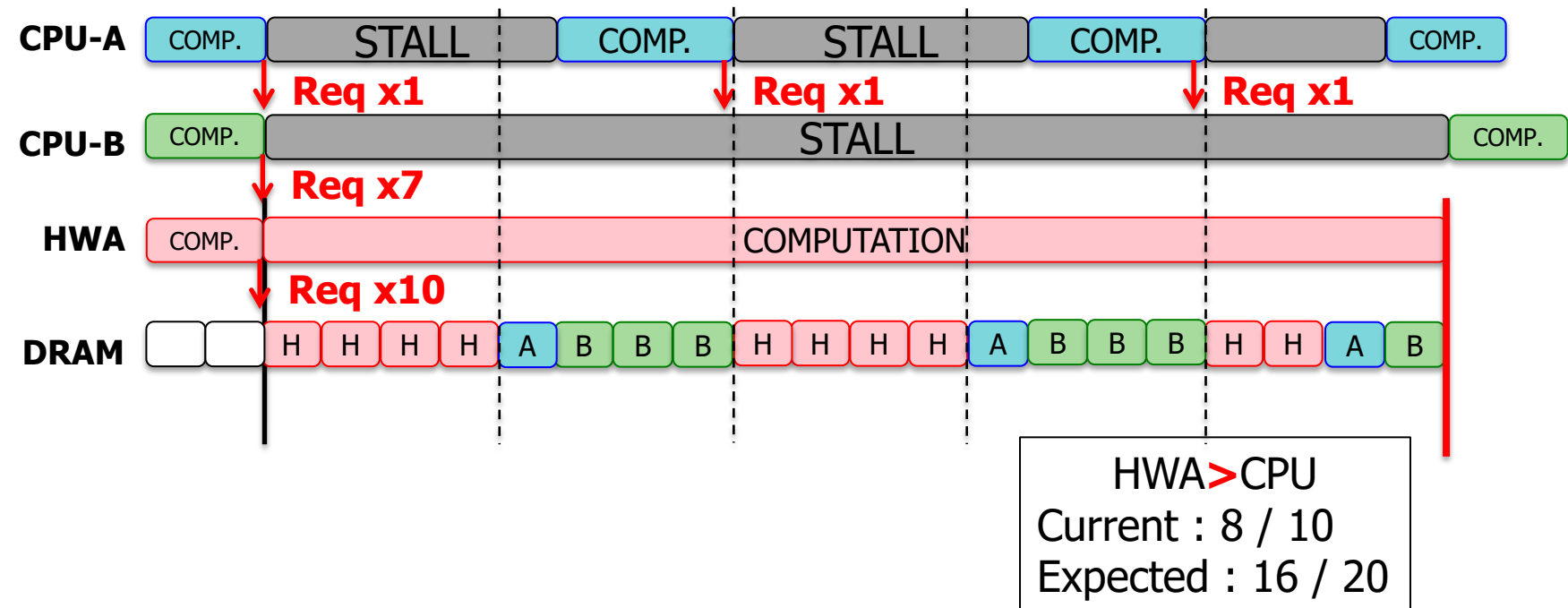
DASH: Distributed Priority

■ Distributed Priority (Scheduling unit = 4T)



DASH: Distributed Priority

■ Distributed Priority (Scheduling unit = 4T)



Key Observation:

Not All Accelerators are Equal

- **Long-deadline** accelerators are more likely to **meet** their deadlines
- **Short-deadline** accelerators are more likely to **miss** their deadlines

*Schedule short-deadline accelerators
based on worst-case memory access time*

Key Observation:

Not All CPU cores are Equal

- **Memory-intensive** cores are much **less vulnerable** to interference
- **Memory non-intensive** cores are much **more vulnerable** to interference

Prioritize accelerators over memory-intensive cores to ensure accelerators do not become urgent

DASH Summary:

Key Ideas and Results

- *Distribute priority for HWAs*
- *Prioritize HWAs over memory-intensive CPU cores even when not urgent*
- *Prioritize short-deadline-period HWAs based on worst case estimates*

Improves CPU performance by 7-21%
Meets (almost) 100% of deadlines for HWAs

DASH: Scheduling Policy

- DASH scheduling policy
 1. Short-deadline-period HWAs with high priority
 2. Long-deadline-period HWAs with high priority
 3. Memory non-intensive CPU applications
 4. Long-deadline-period HWAs with low priority
 5. Memory-intensive CPU applications
 6. Short-deadline-period HWAs with low priority
- } Switch probabilistically

Experimental Methodology (1/2)

■ New Heterogeneous System Simulator

- ❑ We have released this at GitHub (<https://github.com/CMU-SAFARI/HWASim>)

■ Configurations

- ❑ 8 CPUs (2.66GHz), 32KB/L1, 4MB Shared/L2
- ❑ 4 HWAs
- ❑ DDR3 1333 DRAM x 2 channels

■ Workloads

- ❑ CPUs: 80 multi-programmed workloads
 - SPEC CPU2006, TPC, NAS parallel benchmark
- ❑ HWAs:
 - Image processing
 - Image recognition [Lee+ ICCD 2009] [Viola and Jones CVPR 2001]

■ Metrics

- ❑ CPUs : Weighted Speedup
- ❑ HWAs : Deadline met ratio (%)

Experimental Methodology (2/2)

■ Parameters of the HWAs

	Period	Bandwidth	Deadline Group
IMG : Image Processing	33 ms	360MB/s	Long
HES : Hessian	2 us	478MB/s	Short
MAT : Matching (1) 20fps	35.4 us	8.32 GB/s	Long
MAT : Matching (2) 30fps	23.6 us	5.55 GB/s	Long
RSZ : Resize	46.5 – 5183 us	2.07 – 3.33 GB/s	Long
DET : Detect	0.8 – 9.6 us	1.60 – 1.86 GB/s	Short

■ Configurations of 4 HWAs

	Configuration
Config-A	IMG x 2, HES, MAT(2)
Config-B	HES, MAT(1), RSZ, DET

Evaluated Memory Schedulers

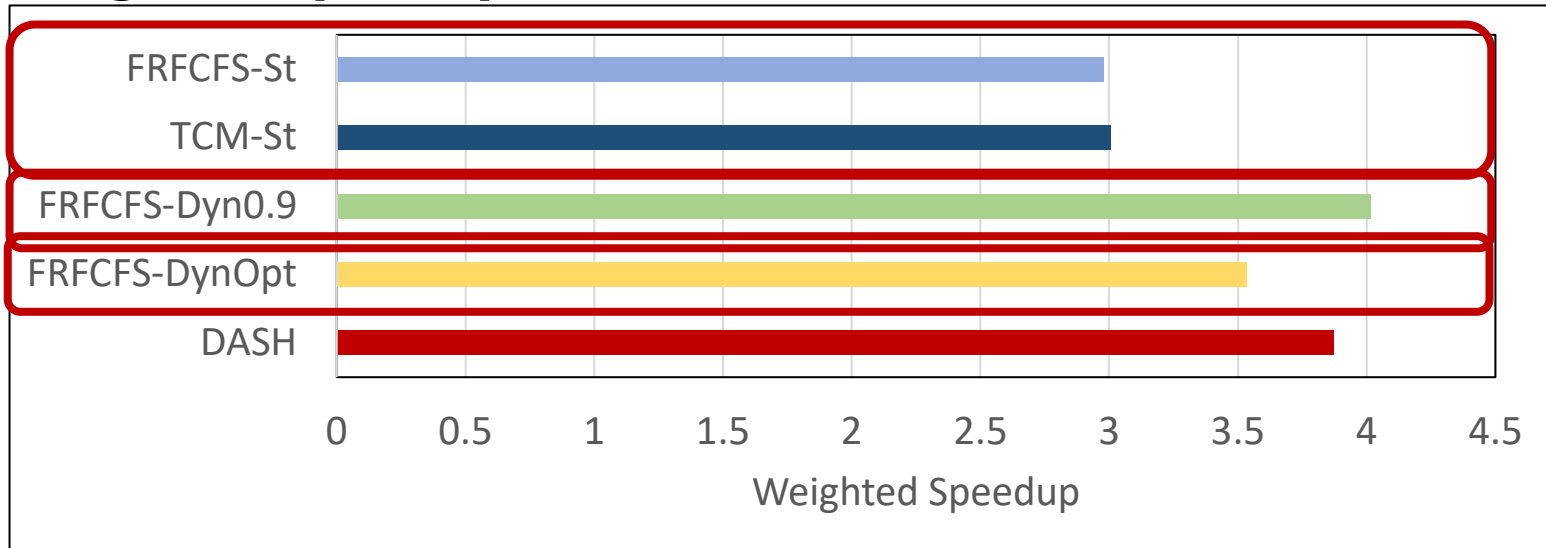
- **FRFCFS-St, TCM-St:** FRFCFS or TCM with *static priority* for HWAs
 - HWAs *always* have higher priority than CPUs
 - **FRFCFS-St:** FRFCFS [Zuravleff and Robinson US Patent 1997, Rixner et al. ISCA 2000] for CPUs
 - Prioritizes row-buffer hits and older requests
 - **TCM-St:** TCM [Kim+ MICRO 2010] for CPUs
 - Always prioritizes memory-non-intensive applications
 - Shuffles thread ranks of memory-intensive applications
- **FRFCFS-Dyn:** FRFCFS with *dynamic priority* for HWAs [Jeong et al., DAC 2012]
 - HWA's priority is dynamically adjusted based on its progress
 - **FRFCFS-Dyn0.9:** EmergentThreshold = **0.9** for all HWAs (Only after 90% of the HWA's period elapsed, the HWA has higher priority than CPUs)
 - **FRFCFS-DynOpt:** Each HWA has different EmergentThreshold to meet its deadline

Config-A			Config-B			
IMG	HES	MAT	HES	MAT	RSZ	DET
0.9	0.2	0.2	0.5	0.4	0.7	0.5

- **DASH:** *Distributed Priority* + *Application-aware scheduling* for CPUs + HWAs
 - TCM is used for CPUs to classify memory intensity of CPUs
 - EmergentThreshold = 0.8 for all HWAs

Performance and Deadline Met Ratio

■ Weighted Speedup for CPUs

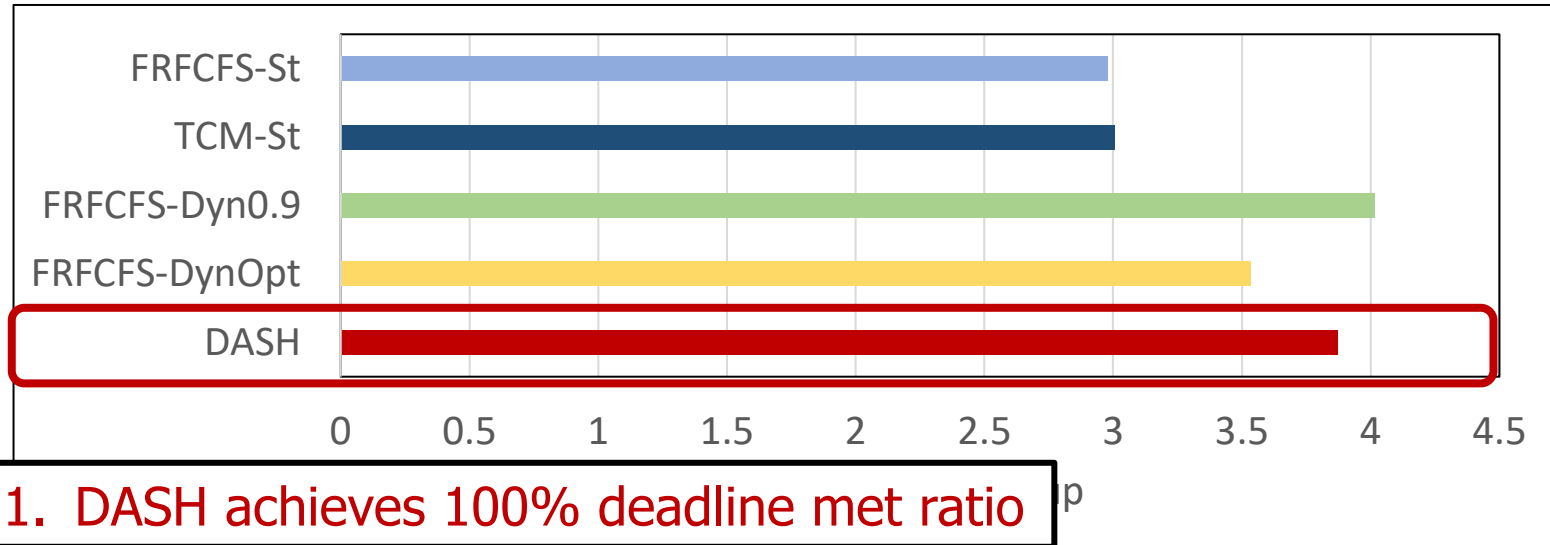


■ Deadline Met Ratio (%) for HWAs

	IMG	HES	MAT	RSZ	DET
FRFCFS-St	100	100	100	100	100
TCM-St	100	100	100	100	100
FRFCFS-Dyn0.9	100	99.4	46.01	97.98	97.14
FRFCFS-DynOpt	100	100	99.997	100	99.99
DASH	100	100	100	100	100

Performance and Deadline Met Ratio

■ Weighted Speedup for CPUs

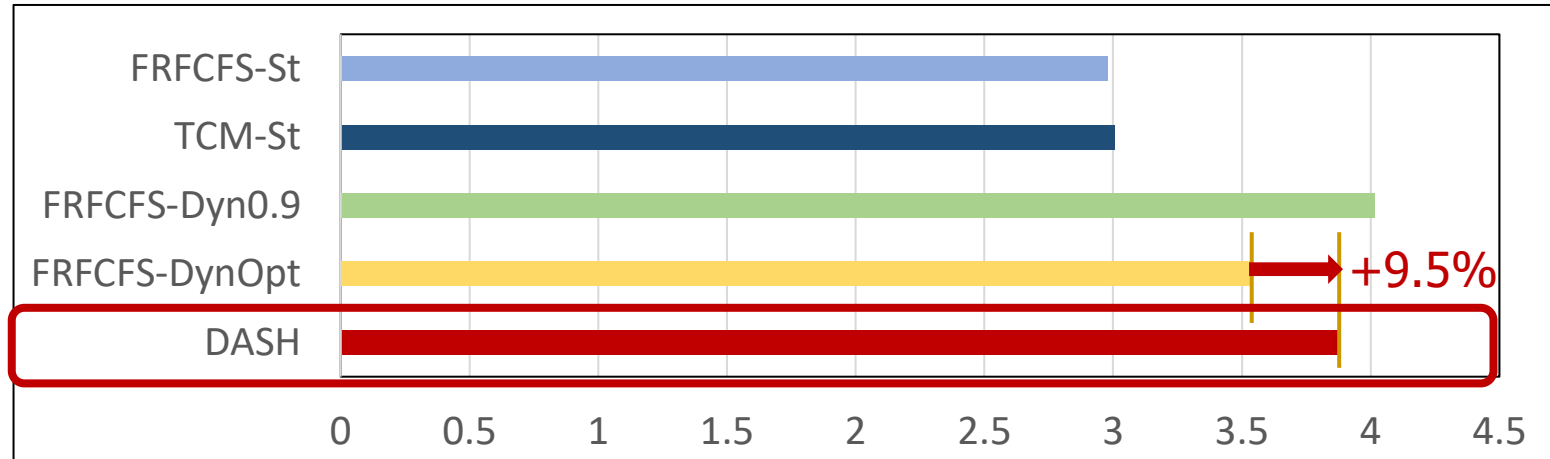


■ Deadline Met Ratio (%) for HWAs

	IMG	HES	MAT	RSZ	DET
FRFCFS-St	100	100	100	100	100
TCM-St	100	100	100	100	100
FRFCFS-Dyn0.9	100	99.4	46.01	97.98	97.14
FRFCFS-DynOpt	100	100	99.997	100	99.99
DASH	100	100	100	100	100

Performance and Deadline Met Ratio

■ Weighted Speedup for CPUs

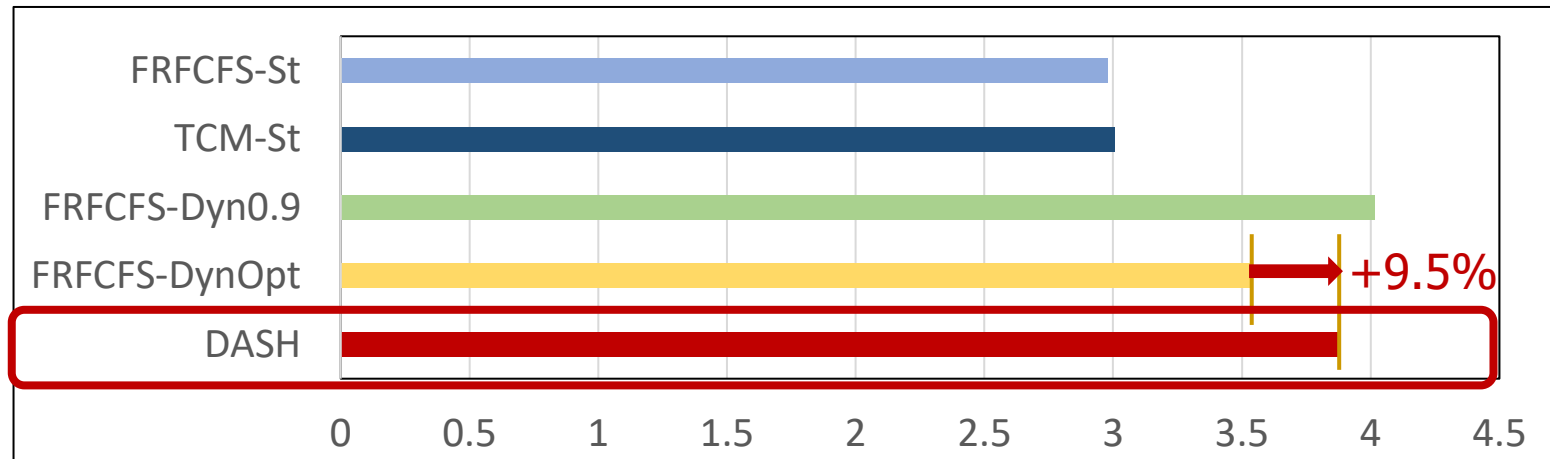


1. DASH achieves 100% deadline met ratio
2. DASH achieves better performance (+9.5%) than FRFCFS-DynOpt that meets the most of HWAs' deadlines (Optimized for HWAs)

FRFCFS-St	100	100	100	100	100
TCM-St	100	100	100	100	100
FRFCFS-Dyn0.9	100	99.4	46.01	97.98	97.14
FRFCFS-DynOpt	100	100	99.997	100	99.99
DASH	100	100	100	100	100

Performance and Deadline Met Ratio

■ Weighted Speedup for CPUs



1. DASH achieves 100% deadline met ratio
2. DASH achieves better performance (+9.5%) than FRFCFS-DynOpt that meets the most of HWAs' deadlines (Optimized for HWAs)
3. DASH achieves comparable performance to FRFCFS-Dyn0.9 that frequently misses HWAs' deadlines (Optimized for CPUs)

TCM-St	100	100	100	100	100
FRFCFS-Dyn0.9	100	99.4	46.01	97.98	97.14
FRFCFS-DynOpt	100	100	99.997	100	99.99
DASH	100	100	100	100	100

More on DASH

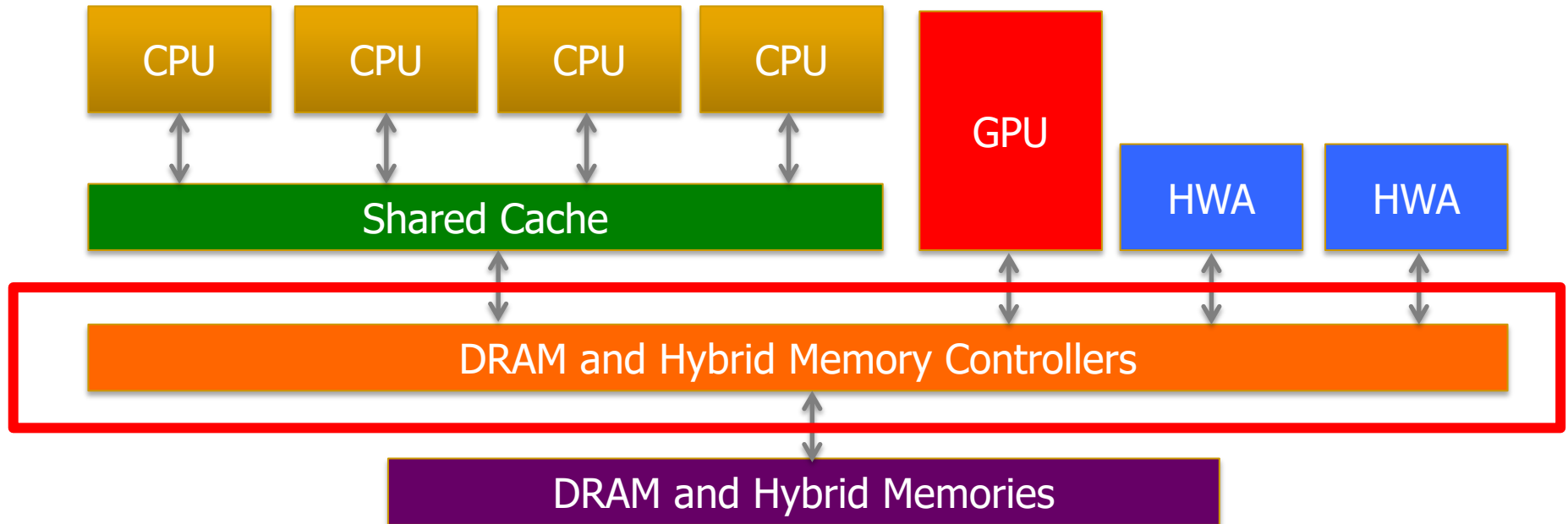
- Hiroyuki Usui, Lavanya Subramanian, Kevin Kai-Wei Chang, and Onur Mutlu,
"DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators"
ACM Transactions on Architecture and Code Optimization (TACO), Vol. 12, January 2016.
Presented at the 11th HiPEAC Conference, Prague, Czech Republic, January 2016.
[Slides (pptx)] [pdf]
[Source Code]

DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators

HIROYUKI USUI, LAVANYA SUBRAMANIAN, KEVIN KAI-WEI CHANG,
and ONUR MUTLU, Carnegie Mellon University

Predictable Performance: Strong Memory Service Guarantees

Goal: Predictable Performance in Complex Systems



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs

How to allocate resources to heterogeneous agents to mitigate interference and provide predictable performance?

Strong Memory Service Guarantees

- Goal: Satisfy performance/SLA requirements in the presence of shared main memory, heterogeneous agents, and hybrid memory/storage
- Approach:
 - Develop techniques/models to accurately estimate the performance loss of an application/agent in the presence of resource sharing
 - Develop mechanisms (hardware and software) to enable the resource partitioning/prioritization needed to achieve the required performance levels for all applications
 - All the while providing high system performance
- Subramanian et al., “MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems,” HPCA 2013.
- Subramanian et al., “The Application Slowdown Model,” MICRO 2015.

Predictable Performance Readings (I)

- Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N. Patt, **"Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems"**
*Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**), pages 335-346, Pittsburgh, PA, March 2010.*
Slides (pdf)

Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems

Eiman Ebrahimi[†] Chang Joo Lee[†] Onur Mutlu[§] Yale N. Patt[†]

[†]Department of Electrical and Computer Engineering
The University of Texas at Austin
{ebrahimi, cjlee, patt}@ece.utexas.edu

[§]Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
onur@cmu.edu

Predictable Performance Readings (II)

- Lavanya Subramanian, Vivek Seshadri, Yoongu Kim, Ben Jaiyen, and Onur Mutlu,
"MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems"
Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), Shenzhen, China, February 2013. [Slides \(pptx\)](#)

MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems

Lavanya Subramanian

Vivek Seshadri

Yoongu Kim

Ben Jaiyen

Onur Mutlu

Carnegie Mellon University

Predictable Performance Readings (III)

- Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu,
"The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory"
Proceedings of the 48th International Symposium on Microarchitecture (MICRO), Waikiki, Hawaii, USA, December 2015.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides \(pptx\)](#)] [[pdf](#)] [[Poster \(pptx\)](#)] [[pdf](#)]
[[Source Code](#)]

The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory

Lavanya Subramanian*§ Vivek Seshadri* Arnab Ghosh*†
Samira Khan*‡ Onur Mutlu*

*Carnegie Mellon University §Intel Labs †IIT Kanpur ‡University of Virginia

MISE:

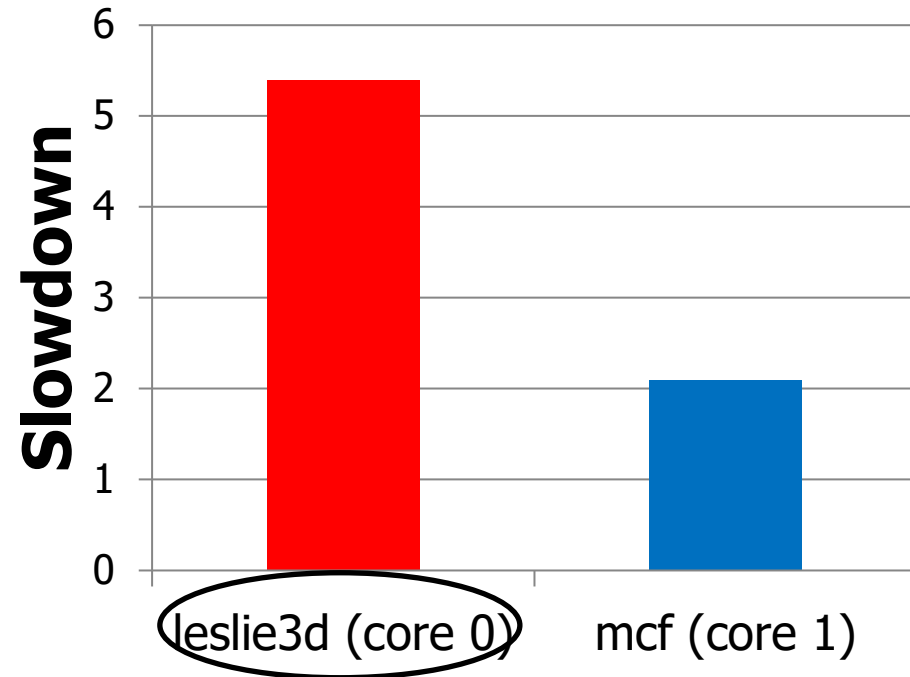
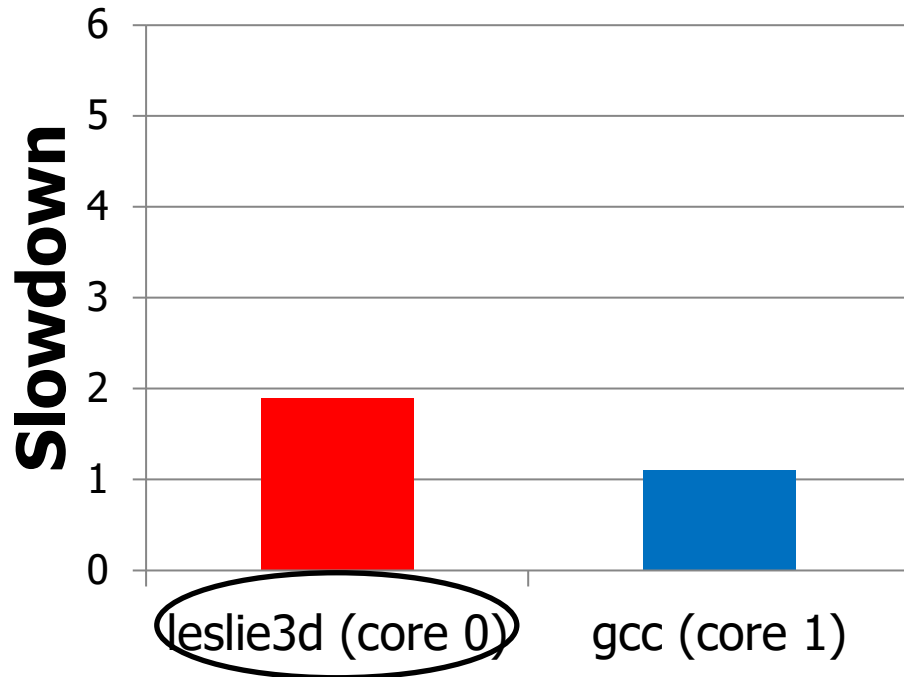
Providing Performance Predictability in Shared Main Memory Systems

Lavanya Subramanian, Vivek Seshadri,
Yoongu Kim, Ben Gai, Onur Mutlu

SAFARI

Carnegie Mellon

Unpredictable Application Slowdowns



An application's performance depends on which application it is running with

Need for Predictable Performance

- There is a need for predictable performance
 - When multiple applications share resources
 - Especially if some applications require performance guarantees

**Our Goal: Predictable performance
in the presence of memory interference**

- Example 2: In server systems
 - Different users' jobs consolidated onto the same server
 - Need to provide bounded slowdowns to critical jobs

Outline

1. Estimate Slowdown

2. Control Slowdown

Outline

1. Estimate Slowdown

- ❑ Key Observations
- ❑ Implementation
- ❑ MISE Model: Putting it All Together
- ❑ Evaluating the Model

2. Control Slowdown

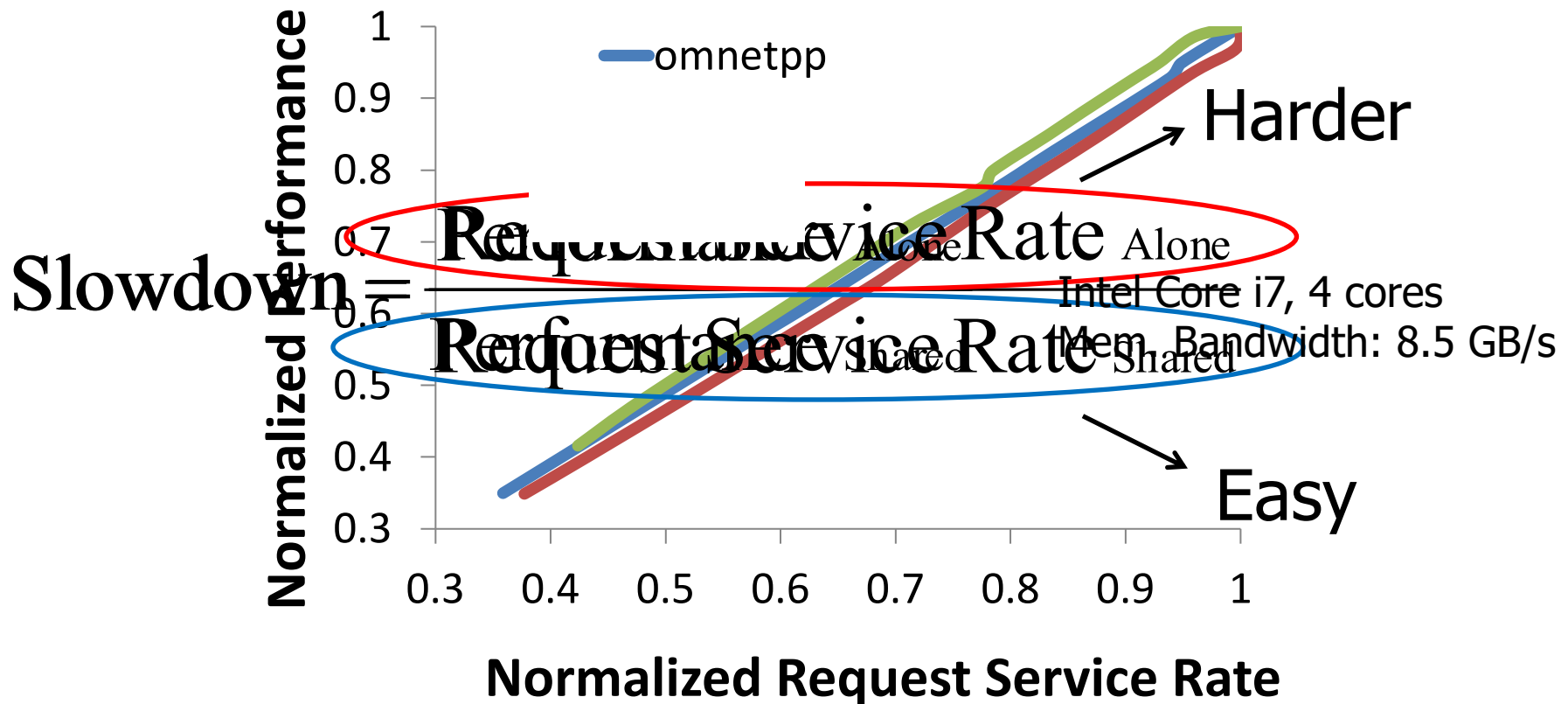
- ❑ Providing Soft Slowdown Guarantees
- ❑ Minimizing Maximum Slowdown

Slowdown: Definition

$$\text{Slowdown} = \frac{\text{Performance}_{\text{Alone}}}{\text{Performance}_{\text{Shared}}}$$

Key Observation 1

For a memory bound application,
Performance \propto Memory request service rate



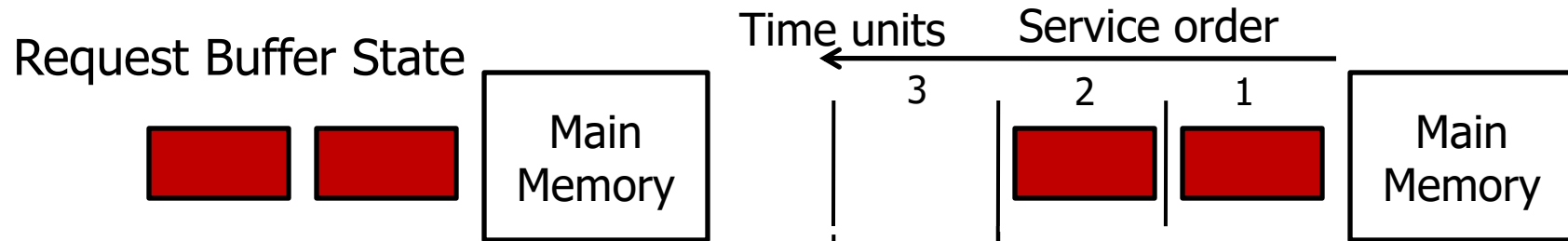
Key Observation 2

Request Service Rate_{Alone} (RSR_{Alone}) of an application can be estimated by giving the application highest priority in accessing memory

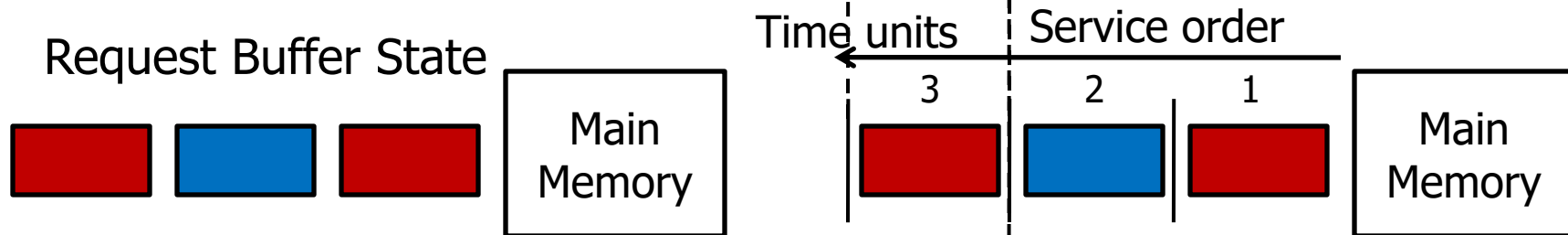
Highest priority → Little interference
(almost as if the application were run alone)

Key Observation 2

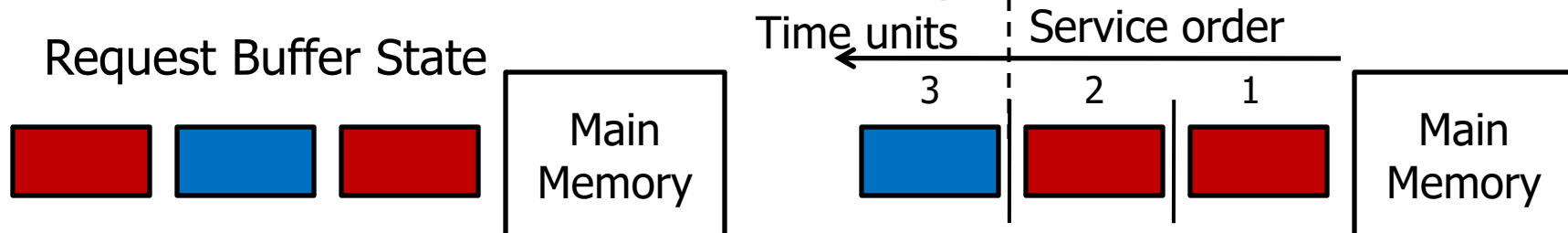
1. Run alone



2. Run with another application



3. Run with another application: **highest priority**

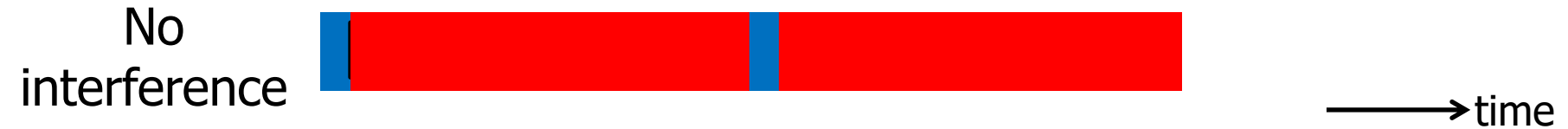
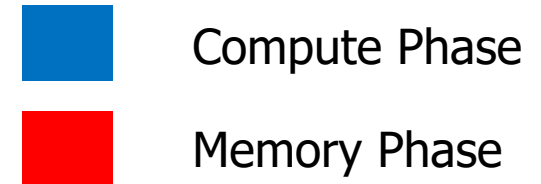


Memory Interference-induced Slowdown Estimation (MISE) model for **memory bound** applications

$$\text{Slowdown} = \frac{\text{Request Service Rate}_{\text{Alone}} (\text{RSR}_{\text{Alone}})}{\text{Request Service Rate}_{\text{Shared}} (\text{RSR}_{\text{Shared}})}$$

Key Observation 3

- Memory-bound application



Memory phase slowdown dominates overall slowdown

Key Observation 3

■ Non-memory-bound application

Memory Interference-induced Slowdown Estimation (MISE) model for **non-memory bound** applications

$$\text{Slowdown} = (1 - \alpha) + \alpha \frac{\text{RSR}_{\text{Alone}}}{\text{RSR}_{\text{Shared}}}$$

Only memory fraction (α) slows down with interference

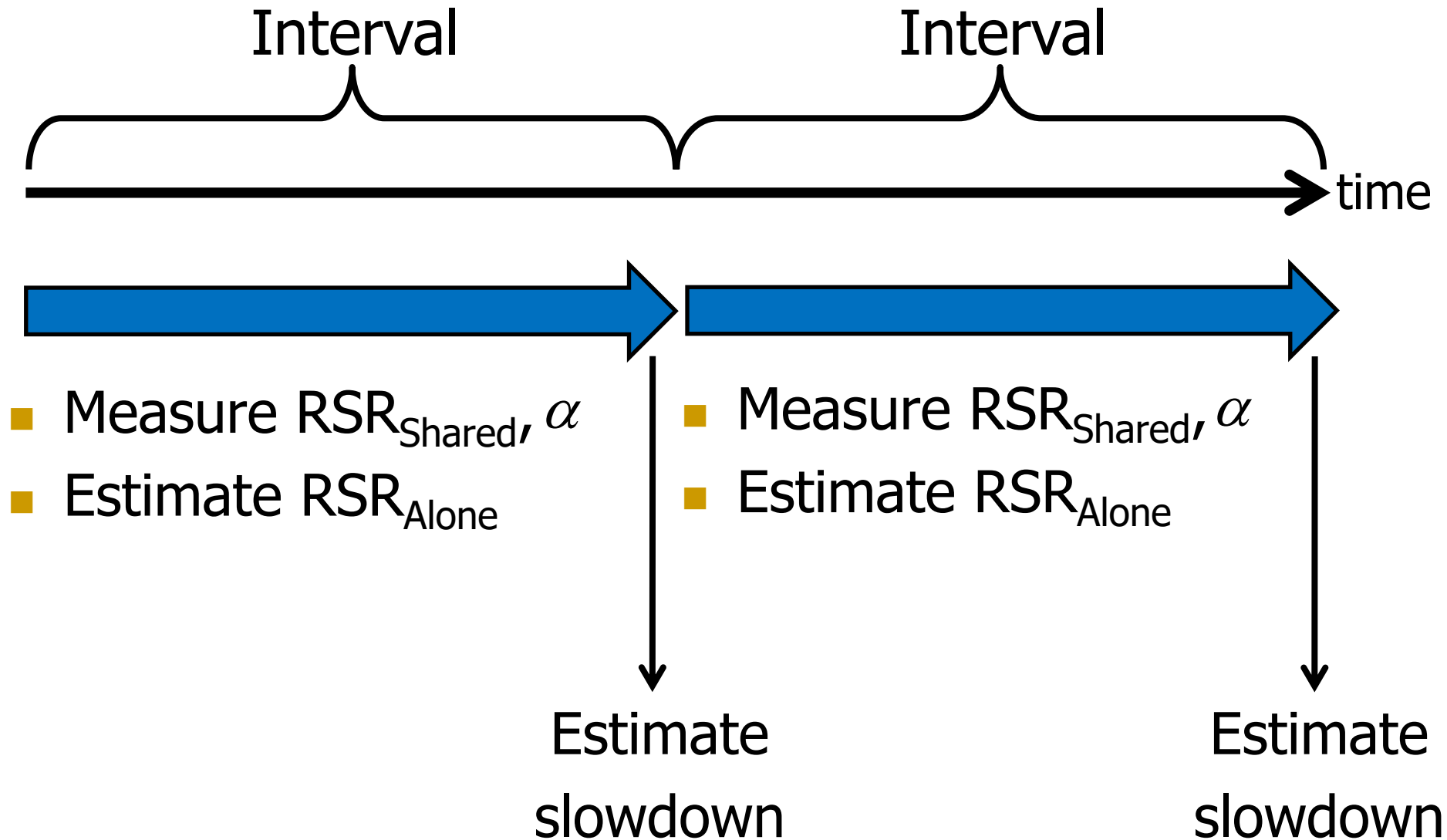
1. Estimate Slowdown

- ❑ Key Observations
- ❑ Implementation
- ❑ MISE Model: Putting it All Together
- ❑ Evaluating the Model

2. Control Slowdown

- ❑ Providing Soft Slowdown Guarantees
- ❑ Minimizing Maximum Slowdown

Interval Based Operation



Measuring RSR_{Shared} and α

- Request Service Rate $_{\text{Shared}}$ (RSR_{Shared})
 - Per-core counter to track number of requests serviced
 - At the end of each interval, measure

$$RSR_{\text{Shared}} = \frac{\text{Number of Requests Serviced}}{\text{Interval Length}}$$

- Memory Phase Fraction (α)
 - Count number of stall cycles at the core
 - Compute fraction of cycles stalled for memory

Estimating Request Service Rate $_{\text{Alone}}$ ($\text{RSR}_{\text{Alone}}$)

- Divide each interval into shorter epochs
- At the beginning of each epoch
 - Memory controller randomly picks an application as the highest priority application

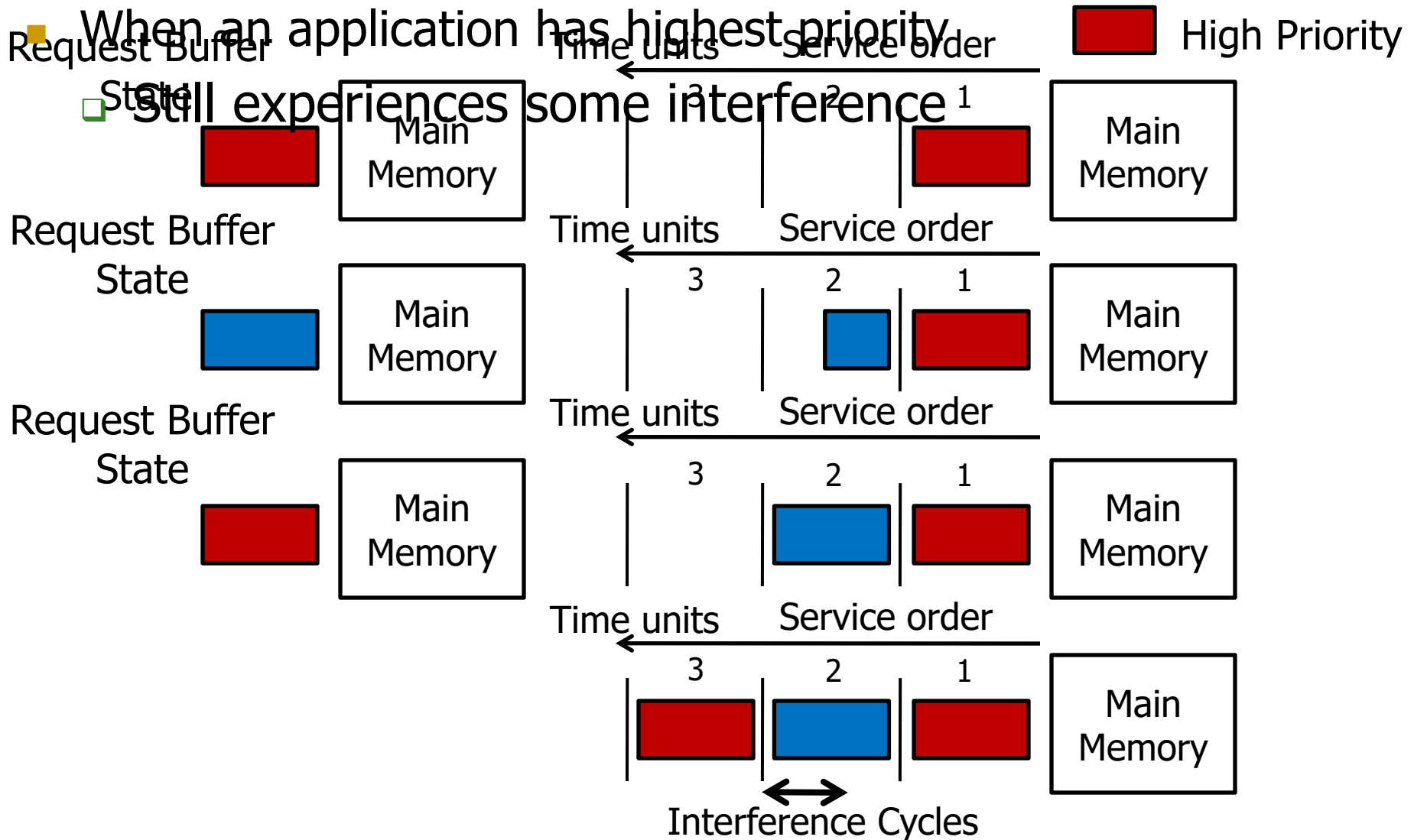
Goal: Estimate $\text{RSR}_{\text{Alone}}$

How: Periodically give each application

- At the end of an interval, for each application, estimate highest priority in accessing memory

$$\text{RSR}_{\text{Alone}} = \frac{\text{Number of Requests During High Priority Epochs}}{\text{Number of Cycles Application Given High Priority}}$$

Inaccuracy in Estimating RSR_{Alone}



Accounting for Interference in RSR_{Alone} Estimation

- **Solution: Determine and remove interference cycles from RSR_{Alone} calculation**

$$RSR_{\text{Alone}} = \frac{\text{Number of Requests During High Priority Epochs}}{\text{Number of Cycles Application Given High Priority} - \text{Interference Cycles}}$$

- A cycle is an interference cycle if
 - a request from the highest priority application is waiting in the request buffer *and*
 - another application's request was issued previously

Outline

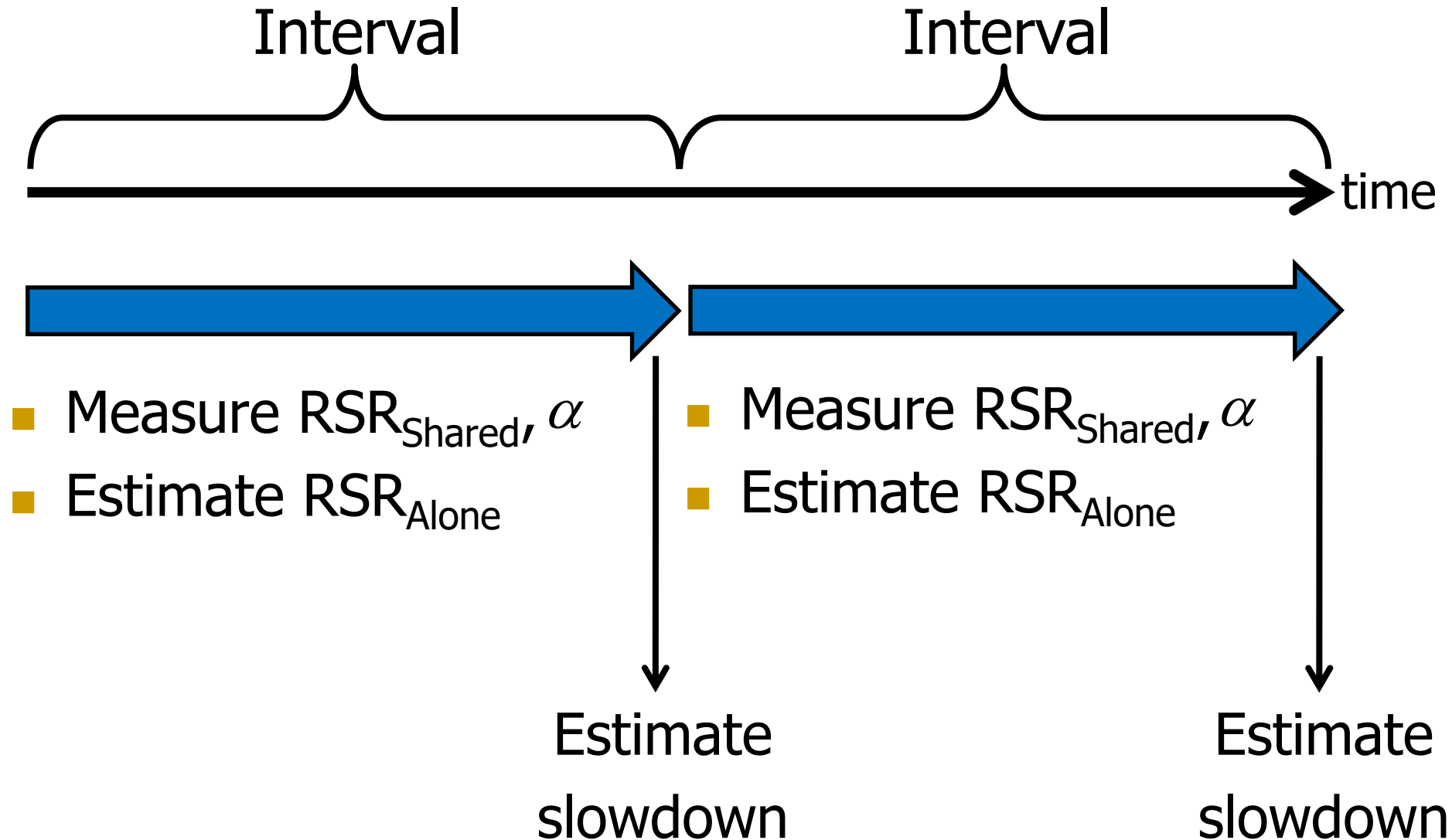
1. Estimate Slowdown

- ❑ Key Observations
- ❑ Implementation
- ❑ MISE Model: Putting it All Together
- ❑ Evaluating the Model

2. Control Slowdown

- ❑ Providing Soft Slowdown Guarantees
- ❑ Minimizing Maximum Slowdown

MISE Model: Putting it All Together



Outline

1. Estimate Slowdown

- ❑ Key Observations
- ❑ Implementation
- ❑ MISE Model: Putting it All Together
- ❑ Evaluating the Model

2. Control Slowdown

- ❑ Providing Soft Slowdown Guarantees
- ❑ Minimizing Maximum Slowdown

Previous Work on Slowdown Estimation

- Previous work on slowdown estimation
 - **STFM** (Stall Time Fair Memory) Scheduling [Mutlu+, MICRO '07]
 - **FST** (Fairness via Source Throttling) [Ebrahimi+, ASPLOS '10]
 - **Per-thread Cycle Accounting** [Du Bois+, HiPEAC '13]
- Basic Idea:

$$\text{Slowdown} = \frac{\text{Stall Time Alone}}{\text{Stall Time Shared}}$$

Diagram illustrating the Basic Idea of Slowdown Estimation:

- The numerator, **Stall Time Alone**, is circled and labeled **Hard** (indicating a difficult or high cost scenario).
- The denominator, **Stall Time Shared**, is labeled **Easy** (indicating an easier or lower cost scenario).

Count number of cycles application receives interference

Two Major Advantages of MISE Over STFM

- Advantage 1:
 - STFM estimates alone performance while an application is receiving interference → Hard
 - MISE estimates alone performance while giving an application the highest priority → Easier
- Advantage 2:
 - STFM does not take into account compute phase for non-memory-bound applications
 - MISE accounts for compute phase → Better accuracy

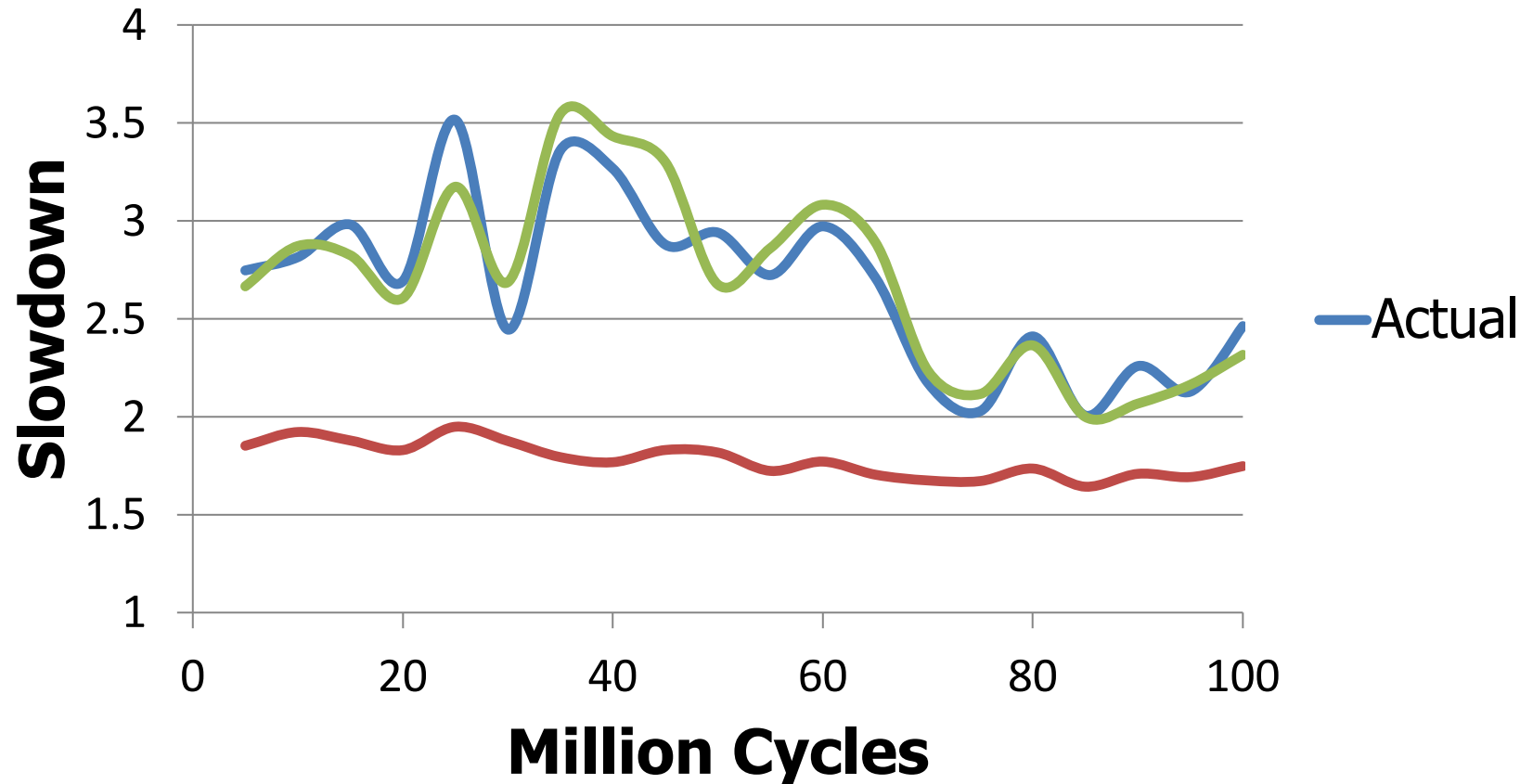
Methodology

- Configuration of our simulated system
 - ❑ 4 cores
 - ❑ 1 channel, 8 banks/channel
 - ❑ DDR3 1066 DRAM
 - ❑ 512 KB private cache/core

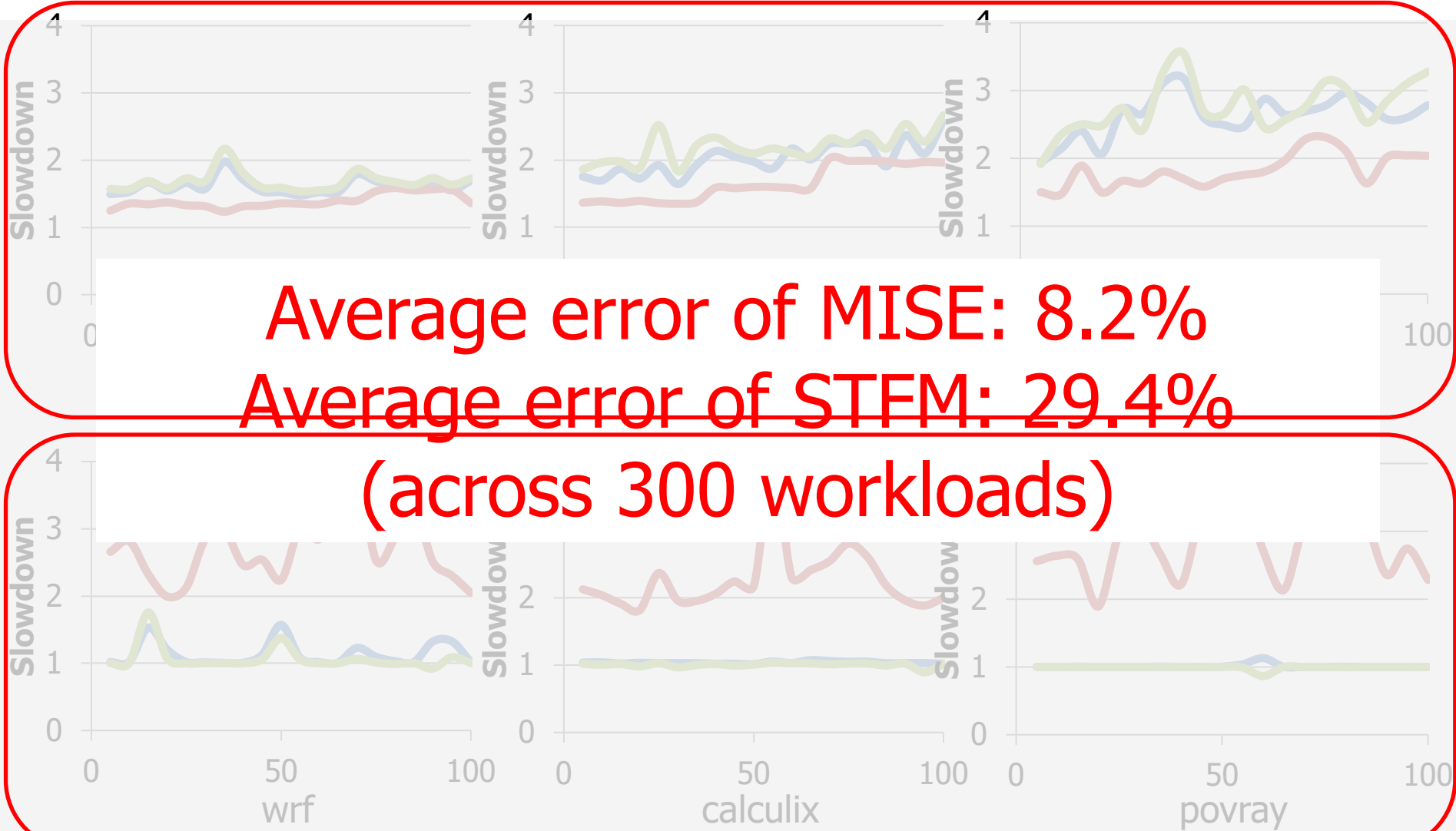
- Workloads
 - ❑ SPEC CPU2006
 - ❑ 300 multi programmed workloads

Quantitative Comparison

SPEC CPU 2006 application
leslie3d



Comparison to STFM



Outline

1. Estimate Slowdown

- ❑ Key Observations
- ❑ Implementation
- ❑ MISE Model: Putting it All Together
- ❑ Evaluating the Model

2. Control Slowdown

- ❑ Providing Soft Slowdown Guarantees
- ❑ Minimizing Maximum Slowdown

Providing “Soft” Slowdown Guarantees

- Goal

1. Ensure QoS-critical applications meet a prescribed slowdown bound
2. Maximize system performance for other applications

- Basic Idea

- Allocate just enough bandwidth to QoS-critical application
- Assign remaining bandwidth to other applications

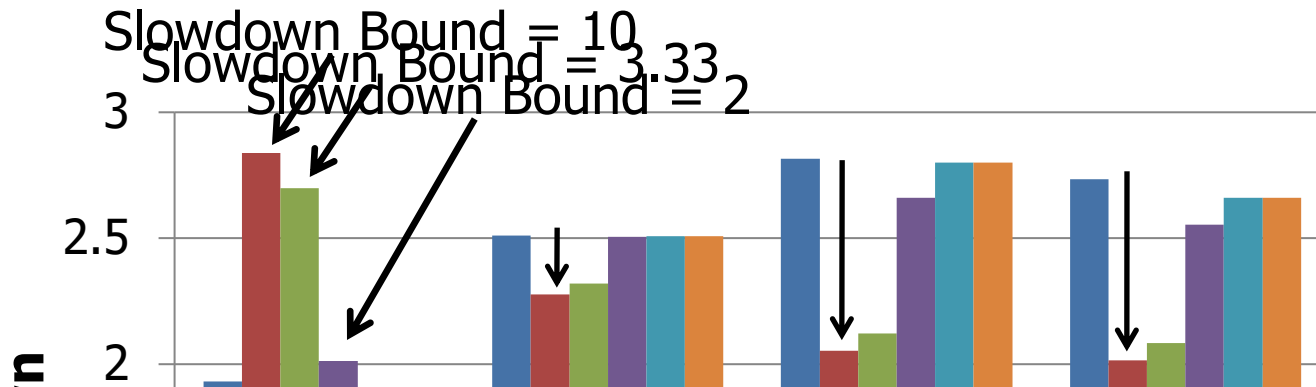
MISE-QoS: Mechanism to Provide Soft QoS

- Assign an initial bandwidth allocation to QoS-critical application
- Estimate slowdown of QoS-critical application using the MISE model
- After every N intervals
 - If slowdown $>$ bound $B \pm \epsilon$, increase bandwidth allocation
 - If slowdown $<$ bound $B \pm \epsilon$, decrease bandwidth allocation
- When slowdown bound not met for N intervals
 - Notify the OS so it can migrate/de-schedule jobs

Methodology

- Each application (25 applications in total) considered the QoS-critical application
- Run with 12 sets of co-runners of different memory intensities
- Total of 300 multiprogrammed workloads
- Each workload run with 10 slowdown bound values
- Baseline memory scheduling mechanism
 - Always prioritize QoS-critical application
[Iyer+, SIGMETRICS 2007]
 - Other applications' requests scheduled in FRFCFS order
[Zuravleff +, US Patent 1997, Rixner+, ISCA 2000]

A Look at One Workload



MISE is effective in

1. meeting the slowdown bound for the QoS-critical application
2. improving performance of non-QoS-critical applications

leslie3d hmmer lbm omnetpp
QoS-critical non-QoS-critical

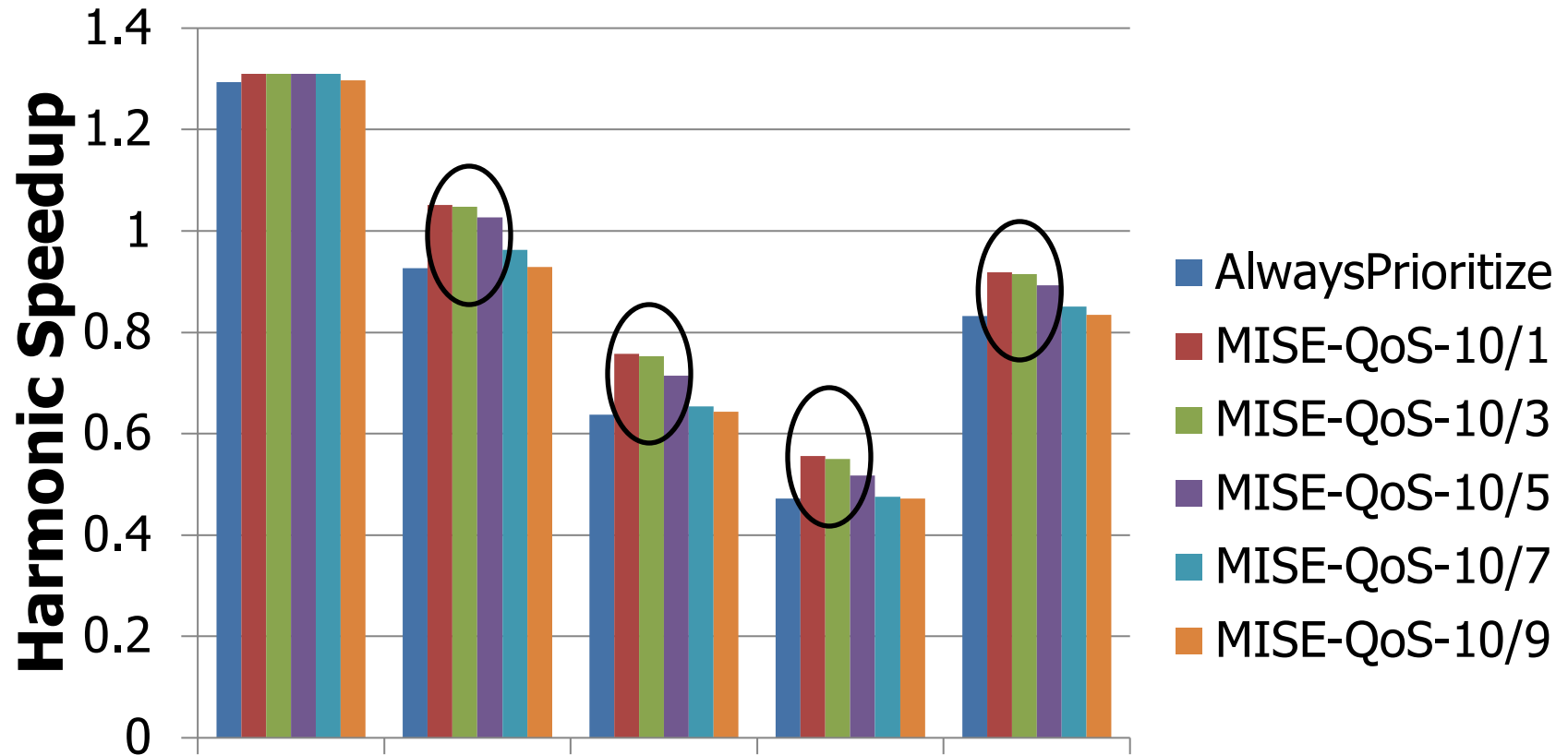
Effectiveness of MISE in Enforcing QoS

Across 3000 data points

	Predicted Met	Predicted Not Met
QoS Bound Met	78.8%	2.1%
QoS Bound Not Met	2.2%	16.9%

MISE-QoS correctly predicts whether or not the bound is met for 95.7% of workloads

Performance of Non-QoS-Critical Applications



When slowdown bound is 10/3
MISE-QoS improves system performance by 10%

Outline

1. Estimate Slowdown

- ❑ Key Observations
- ❑ Implementation
- ❑ MISE Model: Putting it All Together
- ❑ Evaluating the Model

2. Control Slowdown

- ❑ Providing Soft Slowdown Guarantees
- ❑ Minimizing Maximum Slowdown

Other Results in the Paper

- Sensitivity to model parameters
 - Robust across different values of model parameters
- Comparison of STFM and MISE models in enforcing soft slowdown guarantees
 - MISE significantly more effective in enforcing guarantees
- Minimizing maximum slowdown
 - MISE improves fairness across several system configurations

Summary

- Uncontrolled memory interference slows down applications unpredictably
- Goal: **Estimate and control** slowdowns
- Key contribution
 - MISE: An accurate slowdown estimation model
 - Average error of MISE: 8.2%
- Key Idea
 - Request Service Rate is a proxy for performance
 - Request Service Rate_{Alone} estimated by giving an application highest priority in accessing memory
- **Leverage slowdown estimates to control slowdowns**
 - Providing soft slowdown guarantees
 - Minimizing maximum slowdown

MISE: Pros and Cons

■ Upsides:

- ❑ Simple new insight to estimate slowdown
- ❑ Much more accurate slowdown estimations than prior techniques (STFM, FST)
- ❑ Enables a number of QoS mechanisms that can use slowdown estimates to satisfy performance requirements

■ Downsides:

- ❑ Slowdown estimation is not perfect - there are still errors
- ❑ Does not take into account caches and other shared resources in slowdown estimation

More on MISE

- Lavanya Subramanian, Vivek Seshadri, Yoongu Kim, Ben Jaiyen, and Onur Mutlu,
"MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems"
Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), Shenzhen, China, February 2013. [Slides \(pptx\)](#)

MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems

Lavanya Subramanian

Vivek Seshadri

Yoongu Kim

Ben Jaiyen

Onur Mutlu

Carnegie Mellon University

Extending MISE to Shared Caches: ASM

- Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu,
"The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory"
Proceedings of the 48th International Symposium on Microarchitecture (MICRO), Waikiki, Hawaii, USA, December 2015.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides \(pptx\)](#)] [[pdf](#)] [[Poster \(pptx\)](#)] [[pdf](#)]
[[Source Code](#)]

The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory

Lavanya Subramanian*§ Vivek Seshadri* Arnab Ghosh*†
Samira Khan*‡ Onur Mutlu*

*Carnegie Mellon University §Intel Labs †IIT Kanpur ‡University of Virginia

Handling Memory Interference In Multithreaded Applications

Eiman Ebrahimi, Rustam Miftakhutdinov, Chris Fallin,
Chang Joo Lee, Onur Mutlu, and Yale N. Patt,

"Parallel Application Memory Scheduling"

*Proceedings of the 44th International Symposium on Microarchitecture (**MICRO**),
Porto Alegre, Brazil, December 2011. Slides (pptx)*

Multithreaded (Parallel) Applications

- Threads in a multi-threaded application can be inter-dependent
 - As opposed to threads from different applications
- Such threads can synchronize with each other
 - Locks, barriers, pipeline stages, condition variables, semaphores, ...
- Some threads can be on the critical path of execution due to synchronization; some threads are not
- Even within a thread, some “code segments” may be on the critical path of execution; some are not

Critical Sections

- Enforce mutually exclusive access to shared data
- Only one thread can be executing it at a time
- Contended critical sections make threads wait → threads causing serialization can be on the critical path

Each thread:

```
loop {
```

```
  Compute
```

N

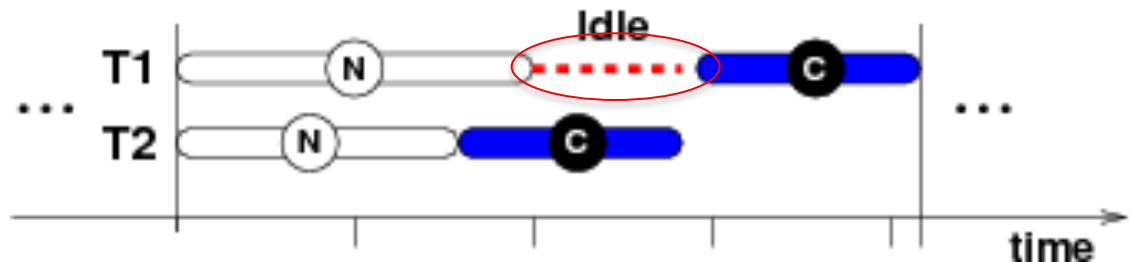
```
  lock(A)
```

```
    Update shared data
```

```
  unlock(A)
```

C

```
}
```

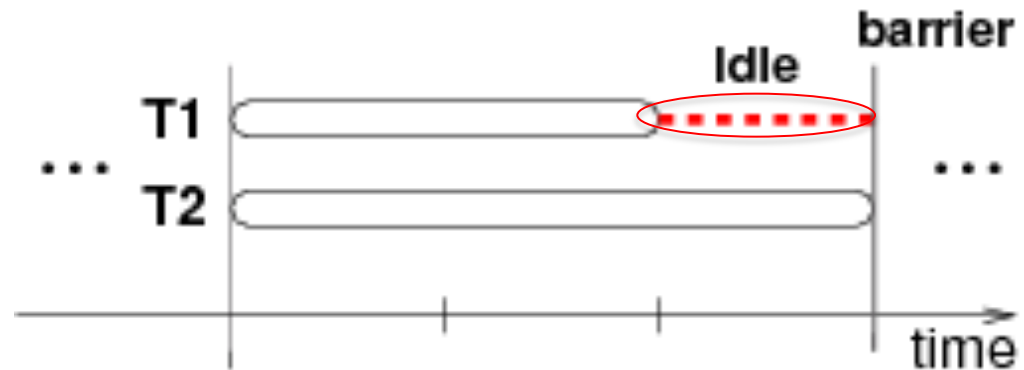


Barriers

- Synchronization point
- Threads have to wait until all threads reach the barrier
- Last thread arriving at the barrier is on the critical path

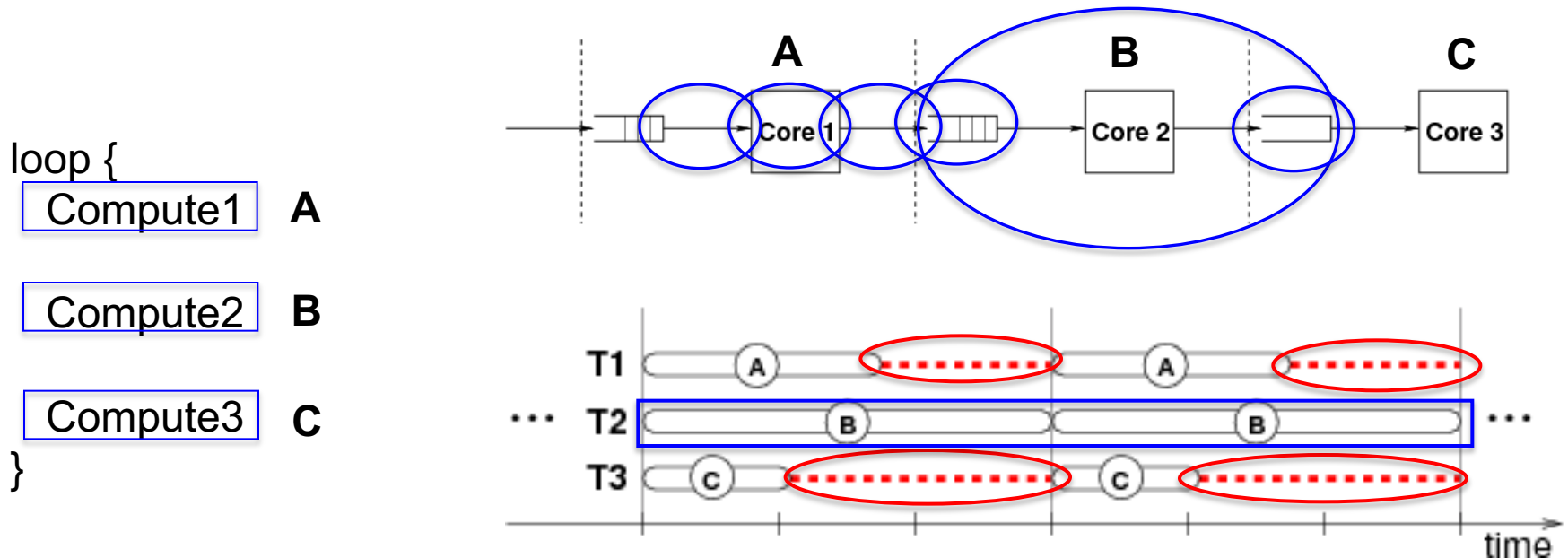
Each thread:

```
loop1 {  
    Compute  
}  
barrier  
loop2 {  
    Compute  
}
```



Stages of Pipelined Programs

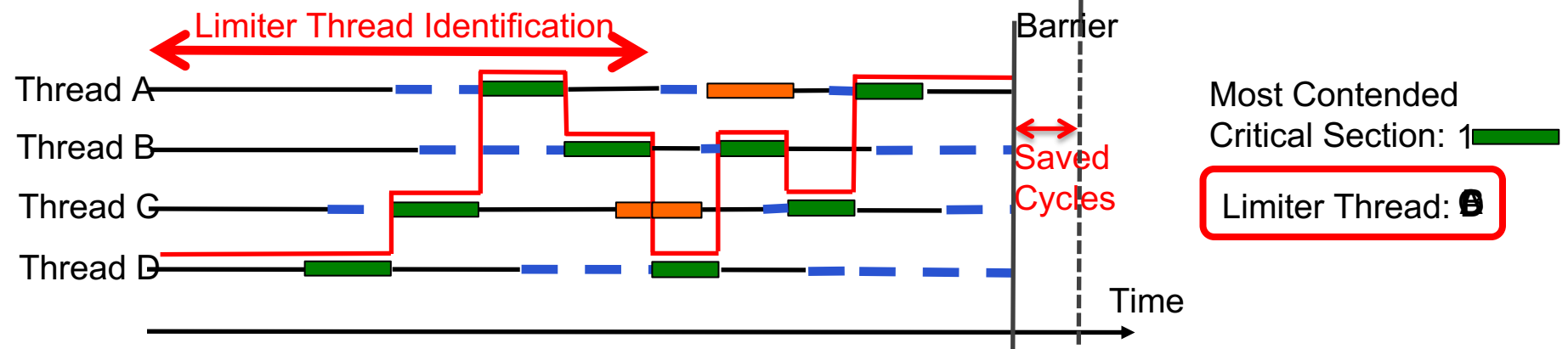
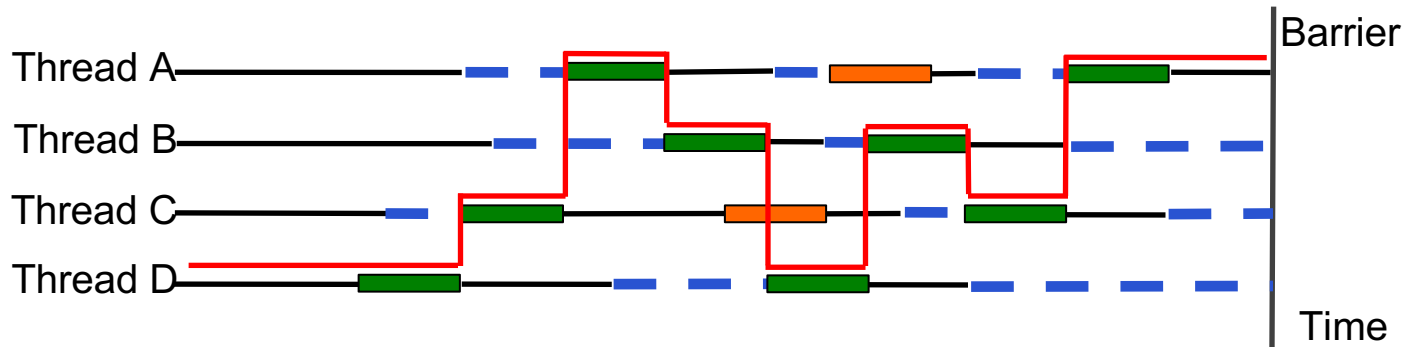
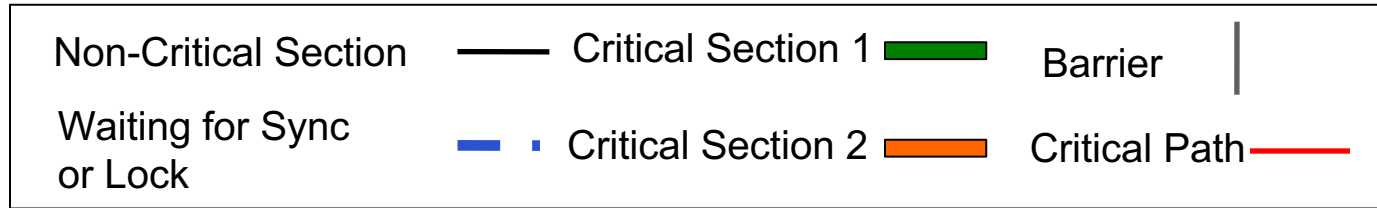
- Loop iterations are statically divided into code segments called *stages*
- Threads execute stages on different cores
- Thread executing the slowest stage is on the critical path



Handling Interference in Parallel Applications

- Threads in a multithreaded application are inter-dependent
- Some threads can be on the critical path of execution due to synchronization; some threads are not
- How do we schedule requests of inter-dependent threads to maximize multithreaded application performance?
- Idea: **Estimate limiter threads** likely to be on the critical path and prioritize their requests; **shuffle priorities of non-limiter threads** to reduce memory interference among them [Ebrahimi+, MICRO'11]
- Hardware/software cooperative limiter thread estimation:
 - Thread executing the most contended critical section
 - Thread executing the slowest pipeline stage
 - Thread that is falling behind the most in reaching a barrier

Prioritizing Requests from Limiter Threads



Parallel App Mem Scheduling: Pros and Cons

- Upsides:
 - ❑ Improves the performance of multi-threaded applications
 - ❑ Provides a mechanism for estimating “limiter threads”
 - ❑ Opens a path for slowdown estimation for multi-threaded applications

- Downsides:
 - ❑ What if there are multiple multi-threaded applications running together?
 - ❑ Limiter thread estimation can become complex

More on PAMS

- Eiman Ebrahimi, Rustam Miftakhutdinov, Chris Fallin, Chang Joo Lee, Onur Mutlu, and Yale N. Patt,
"Parallel Application Memory Scheduling"
*Proceedings of the 44th International Symposium on Microarchitecture (**MICRO**), Porto Alegre, Brazil, December 2011. Slides (pptx)*

Parallel Application Memory Scheduling

Eiman Ebrahimi[†] Rustam Miftakhutdinov[†] Chris Fallin[§]
Chang Joo Lee[‡] José A. Joao[†] Onur Mutlu[§] Yale N. Patt[†]

[†]Department of Electrical and Computer Engineering
The University of Texas at Austin
{ebrahimi, rustam, joao, patt}@ece.utexas.edu

[§]Carnegie Mellon University
{cfallin,onur}@cmu.edu

[‡]Intel Corporation
chang.joo.lee@intel.com

Other Ways of Handling Memory Interference

Fundamental Interference Control Techniques

- **Goal:** to reduce/control inter-thread memory interference

1. Prioritization or request scheduling

2. Data mapping to banks/channels/ranks

3. Core/source throttling

4. Application/thread scheduling

Designing QoS-Aware Memory Systems: Approaches

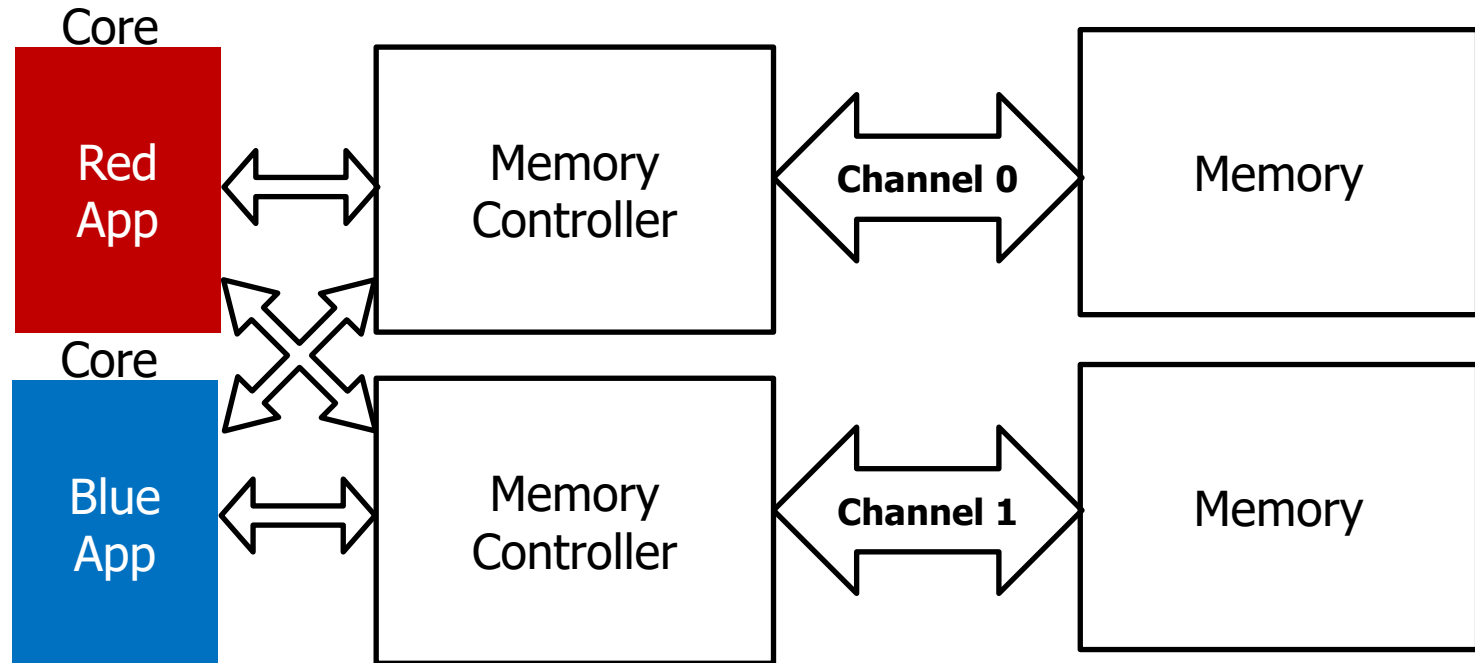
- **Smart resources:** Design each shared resource to have a configurable interference control/reduction mechanism
 - ❑ QoS-aware memory controllers
 - ❑ QoS-aware interconnects
 - ❑ QoS-aware caches

- **Dumb resources:** Keep each resource free-for-all, but reduce/control interference by injection control or data mapping
 - ❑ Source throttling to control access to memory system
 - ❑ QoS-aware data mapping to memory controllers
 - ❑ QoS-aware thread scheduling to cores

Memory Channel Partitioning

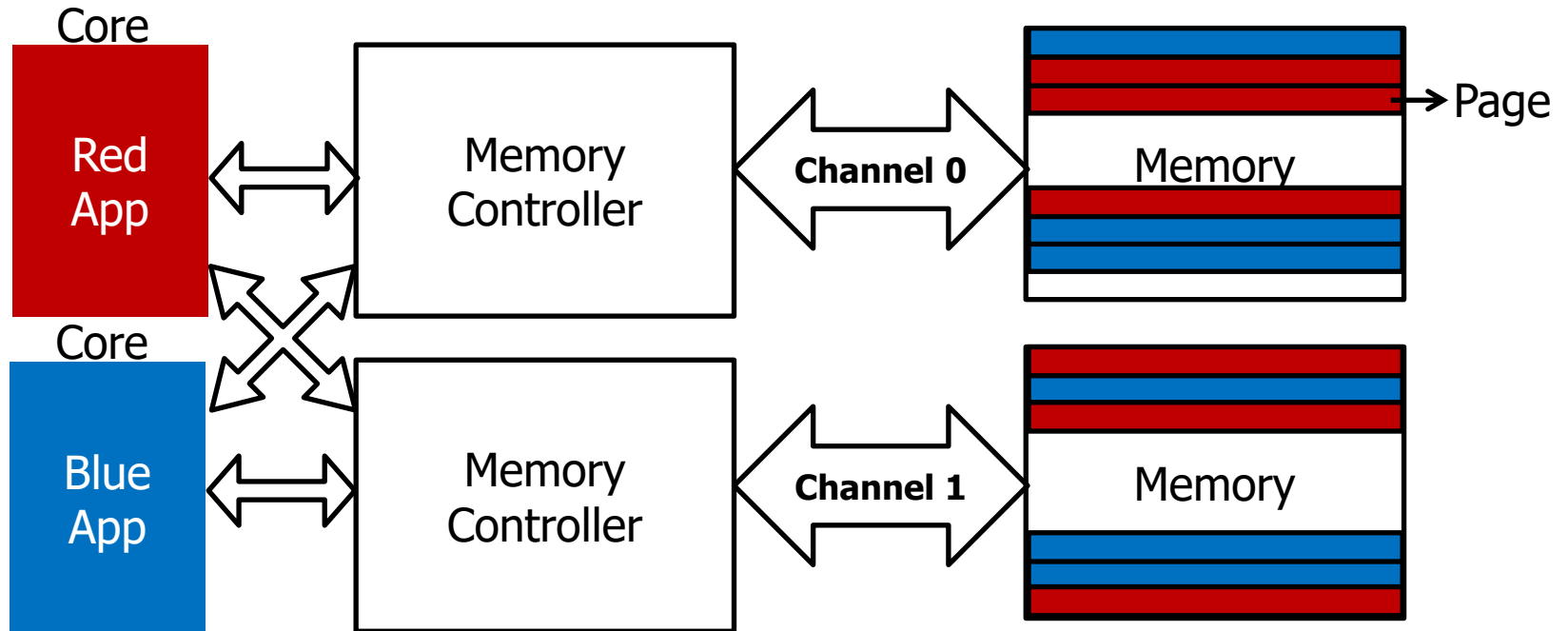
Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda,
**"Reducing Memory Interference in Multicore Systems via
Application-Aware Memory Channel Partitioning"**
*44th International Symposium on Microarchitecture (**MICRO**)*,
Porto Alegre, Brazil, December 2011. [Slides \(pptx\)](#)

Observation: Modern Systems Have Multiple Channels



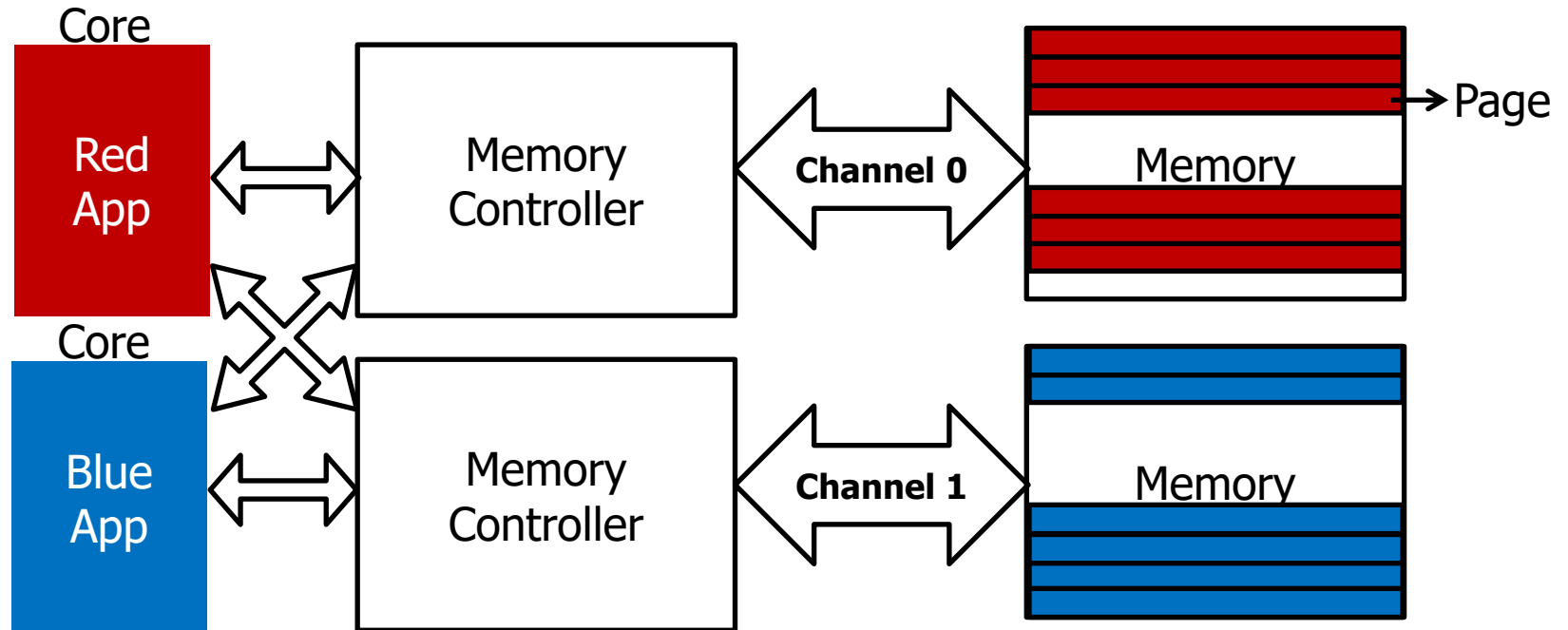
A new degree of freedom
Mapping data across multiple channels

Data Mapping in Current Systems



Causes interference between applications' requests

Partitioning Channels Between Applications



Eliminates interference between applications' requests

Overview: Memory Channel Partitioning (MCP)

■ Goal

- Eliminate harmful interference between applications

■ Basic Idea

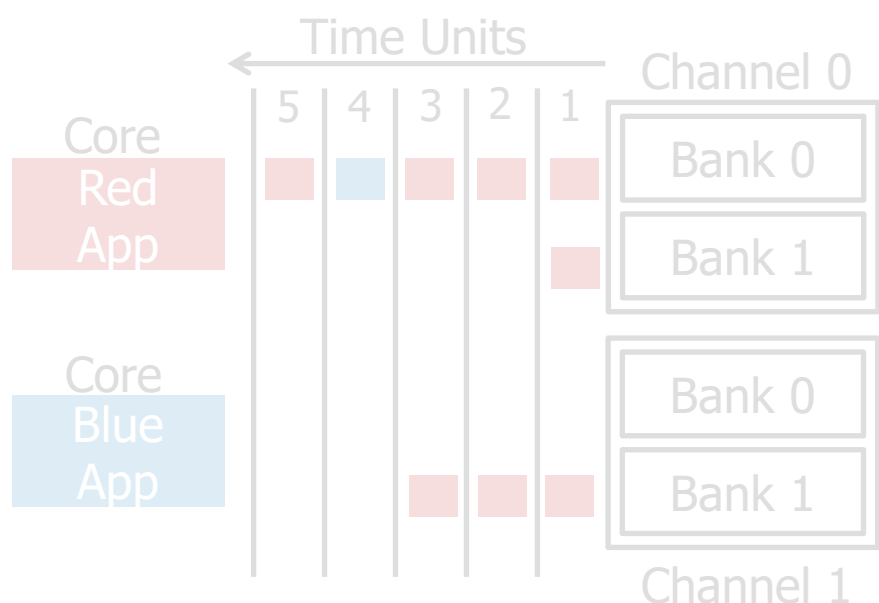
- Map the data of **badly-interfering applications** to different channels

■ Key Principles

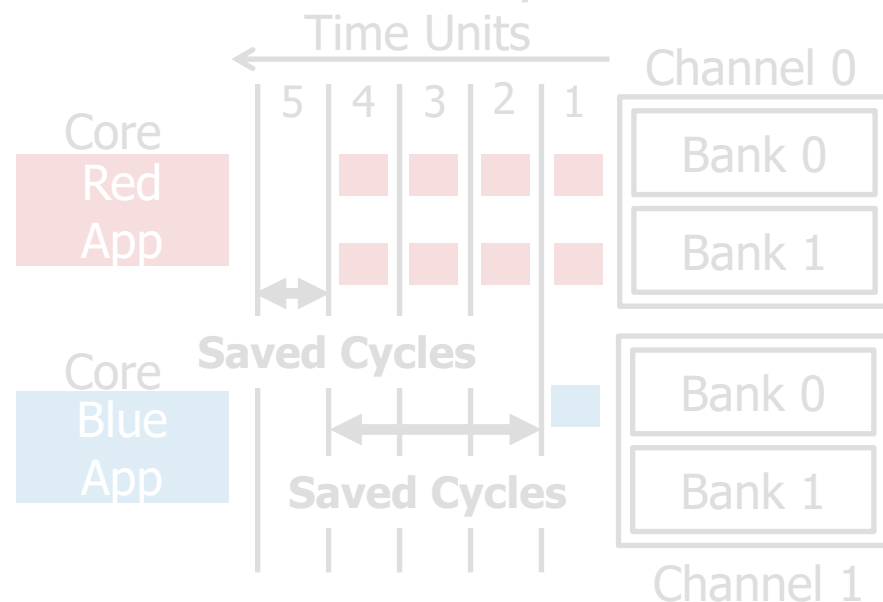
- Separate **low and high memory-intensity applications**
- Separate **low and high row-buffer locality applications**

Key Insight 1: Separate by Memory Intensity

High memory-intensity applications interfere with low memory-intensity applications in shared memory channels



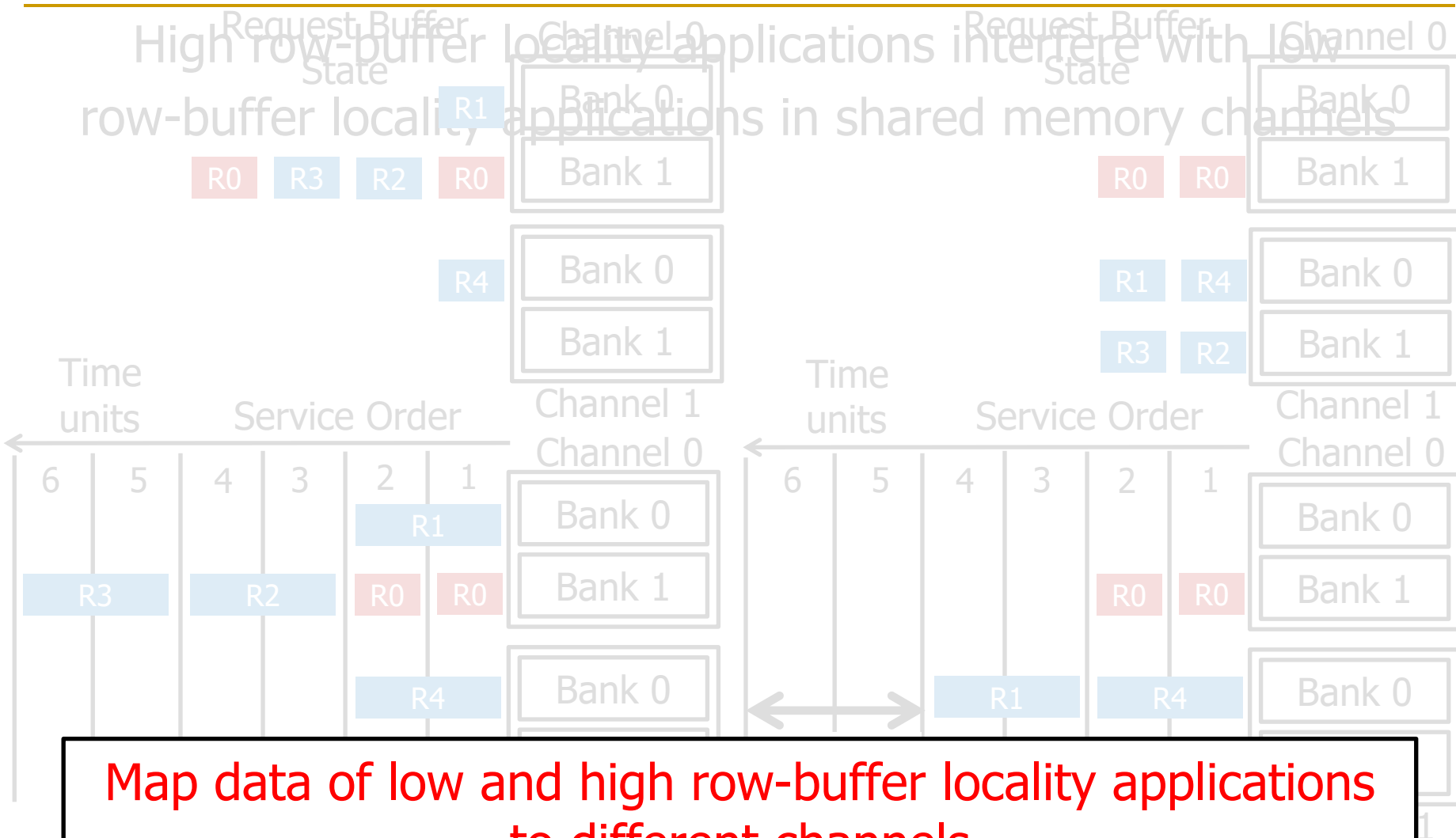
Conventional Page Mapping



Channel Partitioning

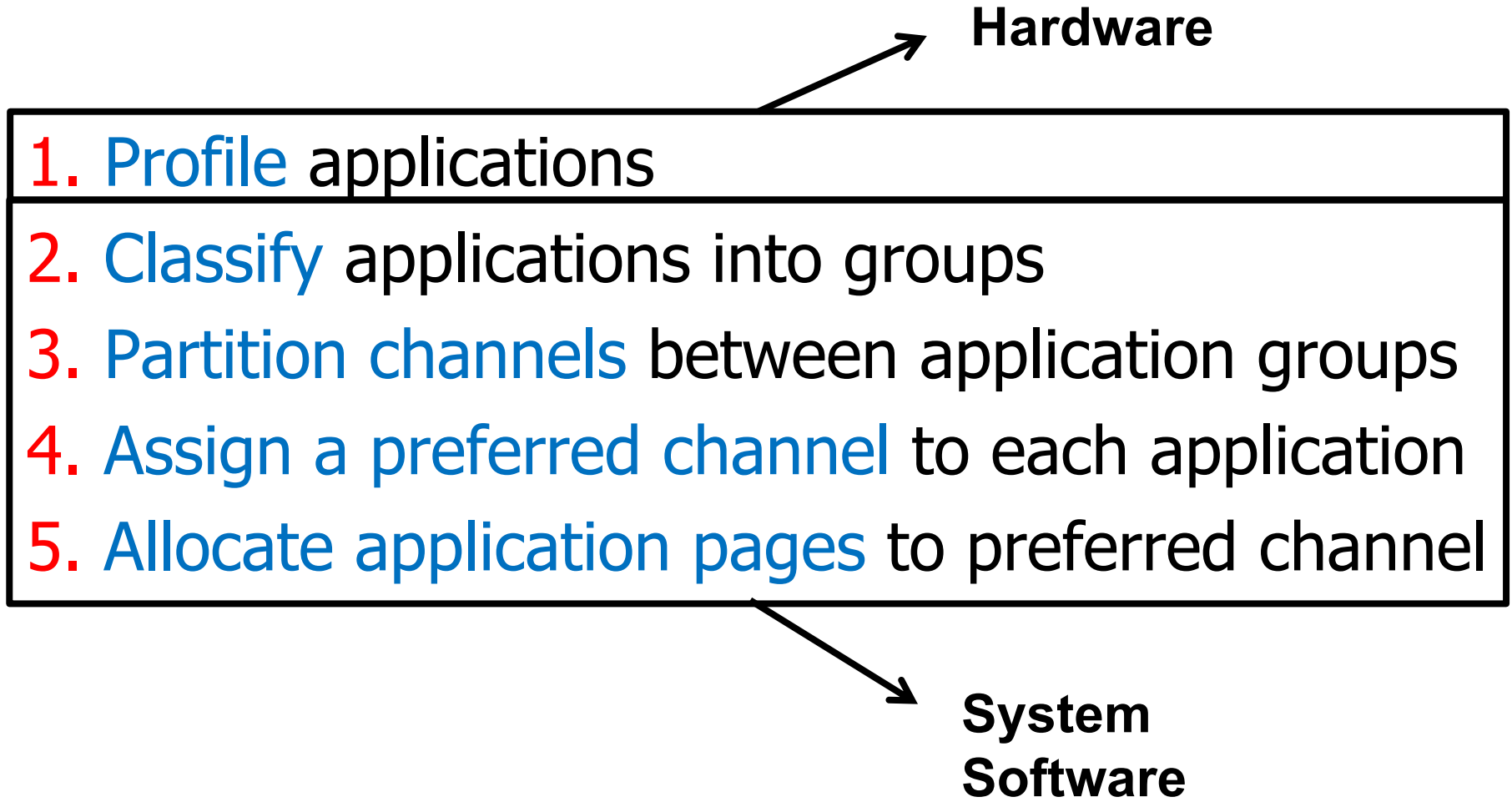
Map data of low and high memory-intensity applications to different channels

Key Insight 2: Separate by Row-Buffer Locality

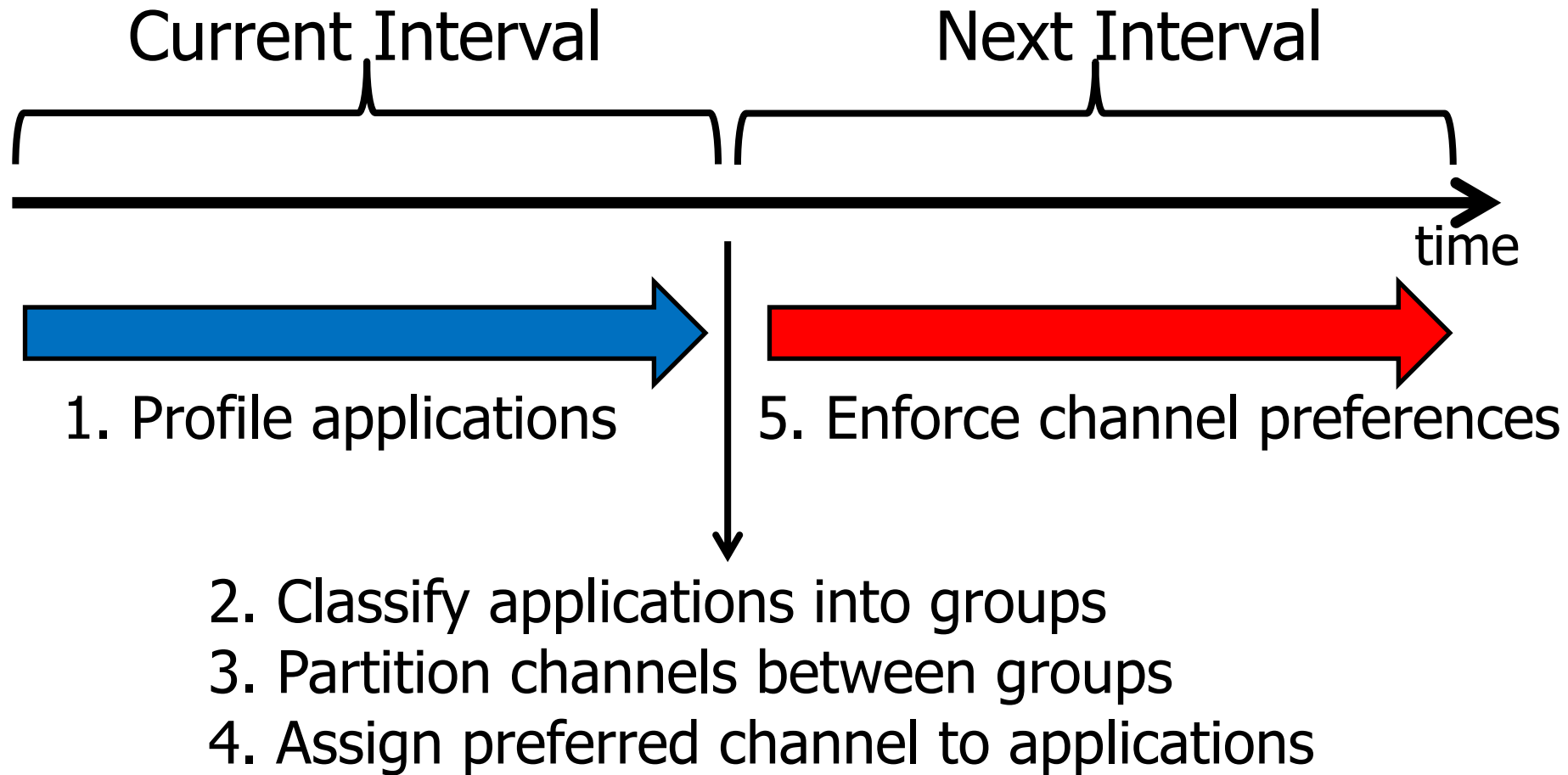


Map data of low and high row-buffer locality applications to different channels

Memory Channel Partitioning (MCP) Mechanism



Interval Based Operation



Observations

- Applications with very low memory-intensity rarely access memory
 - Dedicating channels to them results in precious memory bandwidth waste
- They have the most potential to keep their cores busy
 - We would really like to prioritize them
- They interfere minimally with other applications
 - Prioritizing them does not hurt others

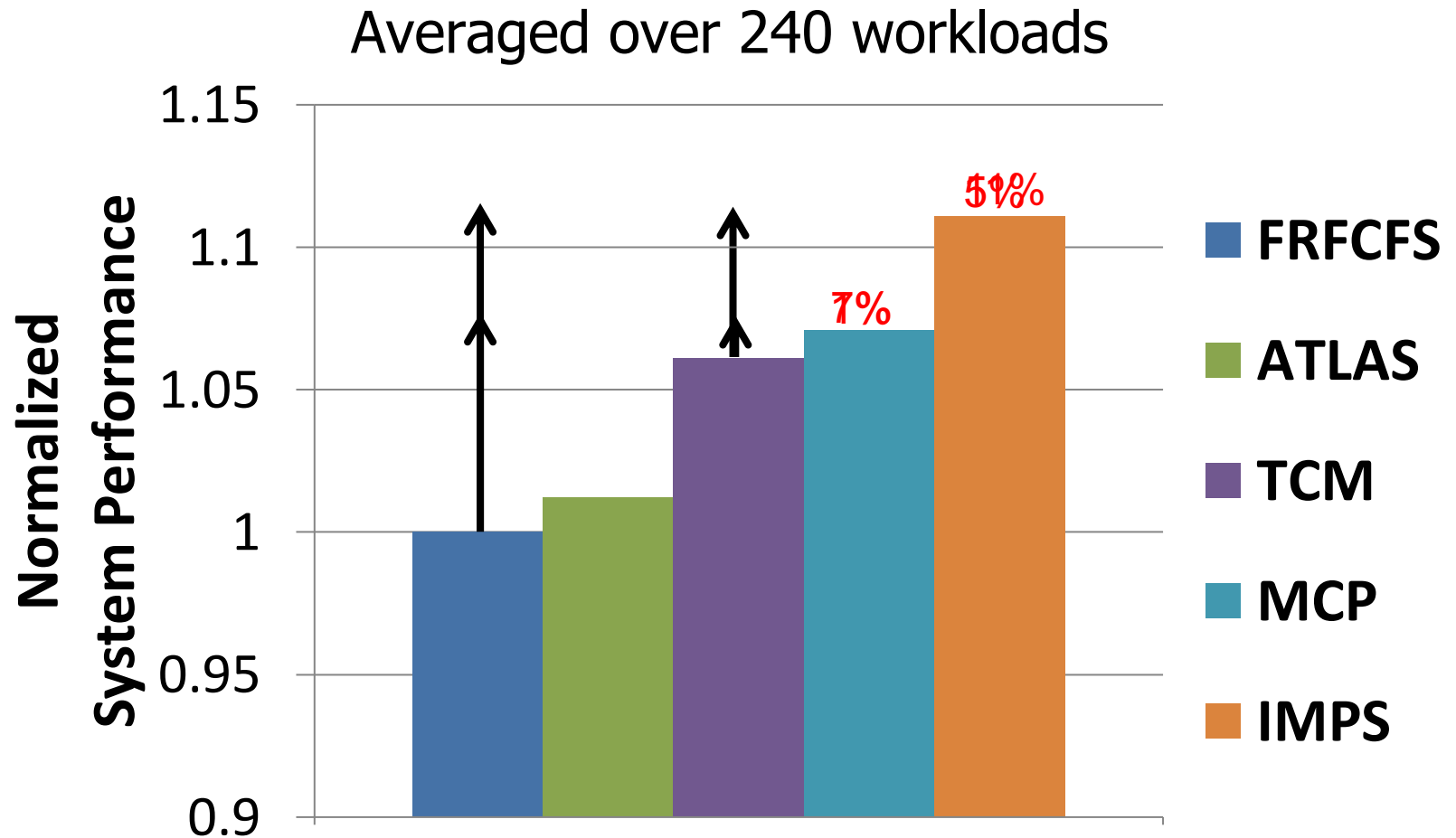
Integrated Memory Partitioning and Scheduling (IMPS)

- Always prioritize very low memory-intensity applications in the memory scheduler
- Use memory channel partitioning to mitigate interference between other applications

Hardware Cost

- **Memory Channel Partitioning (MCP)**
 - ❑ Only profiling counters in hardware
 - ❑ No modifications to memory scheduling logic
 - ❑ 1.5 KB storage cost for a 24-core, 4-channel system
- **Integrated Memory Partitioning and Scheduling (IMPS)**
 - ❑ A single bit per request
 - ❑ Scheduler prioritizes based on this single bit

Performance of Channel Partitioning



Better system performance than the best previous scheduler
at lower hardware cost

Combining Multiple Interference Control Techniques

- Combined interference control techniques can mitigate interference much more than a single technique alone can do
- The key challenge is:
 - Deciding what technique to apply when
 - Partitioning work appropriately between software and hardware

MCP and IMPS: Pros and Cons

■ Upsides:

- ❑ Keeps the memory scheduling hardware simple
- ❑ Combines multiple interference reduction techniques
- ❑ Can provide performance isolation across applications mapped to different channels
- ❑ General idea of partitioning can be extended to smaller granularities in the memory hierarchy: banks, subarrays, etc.

■ Downsides:

- ❑ Reacting is difficult if workload changes behavior after profiling
- ❑ Overhead of moving pages between channels restricts benefits

More on Memory Channel Partitioning

- Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda,
"Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning"
*Proceedings of the 44th International Symposium on Microarchitecture (**MICRO**), Porto Alegre, Brazil, December 2011. Slides (pptx)*

Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning

Sai Prashanth Muralidhara
Pennsylvania State University
smuralid@cse.psu.edu

Lavanya Subramanian
Carnegie Mellon University
lsubrama@ece.cmu.edu

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

Mahmut Kandemir
Pennsylvania State University
kandemir@cse.psu.edu

Thomas Moscibroda
Microsoft Research Asia
moscitho@microsoft.com

Computer Architecture

Lecture 17: Memory Interference and Quality of Service II

Prof. Onur Mutlu

ETH Zürich

Fall 2018

21 November 2018

We did not cover the following slides in lecture.
These are for your preparation for the next lecture.

Fundamental Interference Control Techniques

- **Goal:** to reduce/control inter-thread memory interference

- 1. **Prioritization** or request scheduling

- 2. **Data mapping** to banks/channels/ranks

- 3. **Core/source throttling**

- 4. **Application/thread scheduling**

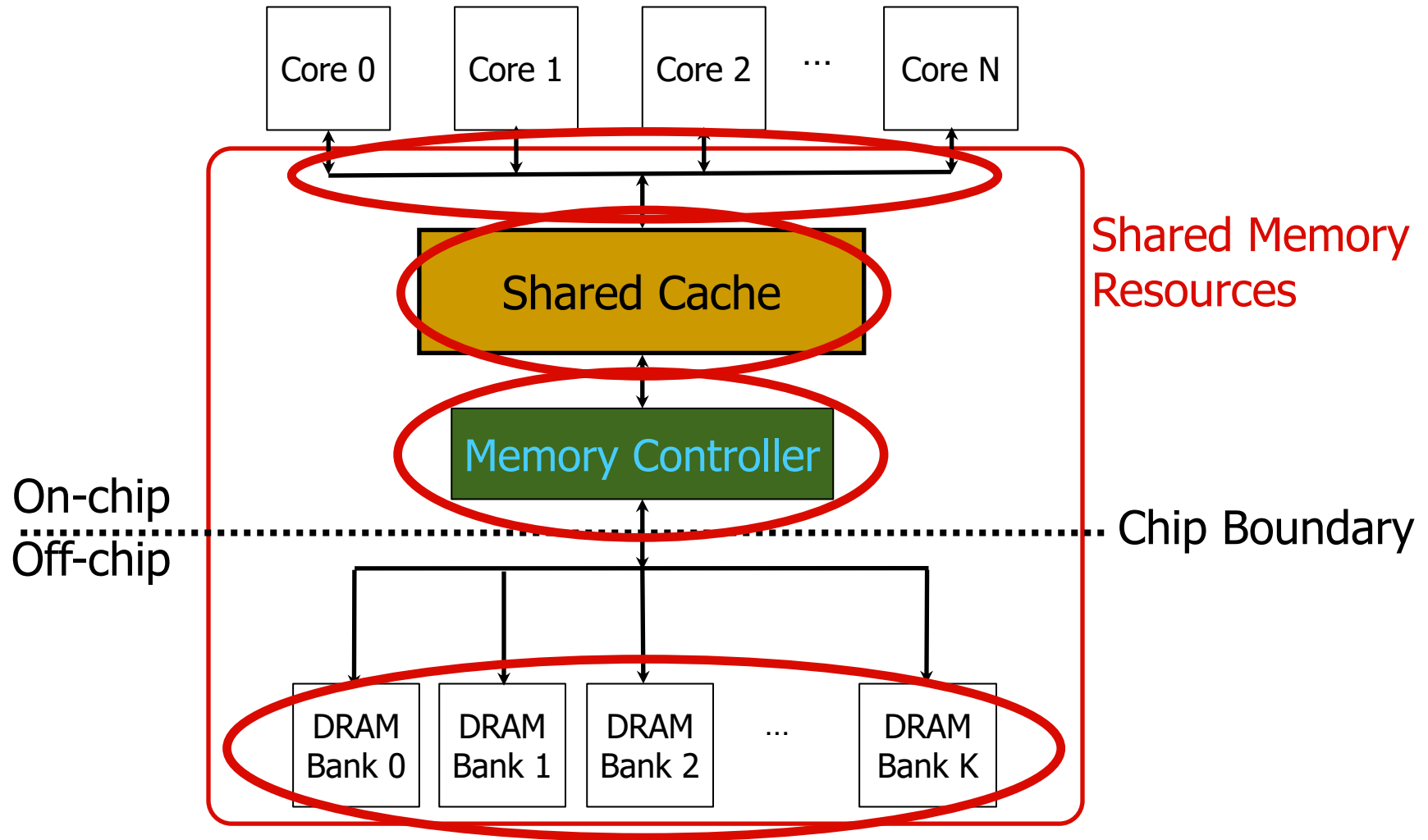
Fairness via Source Throttling

Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N. Patt,

**"Fairness via Source Throttling: A Configurable and High-Performance
Fairness Substrate for Multi-Core Memory Systems"**

15th Intl. Conf. on Architectural Support for Programming Languages and Operating Systems (ASPLOS),
pages 335-346, Pittsburgh, PA, March 2010. [Slides \(pdf\)](#)

Many Shared Resources



The Problem with “Smart Resources”

- Independent interference control mechanisms in caches, interconnect, and memory can contradict each other
- Explicitly coordinating mechanisms for different resources requires complex implementation
- How do we enable fair sharing of the **entire memory system** by controlling interference in a **coordinated manner**?

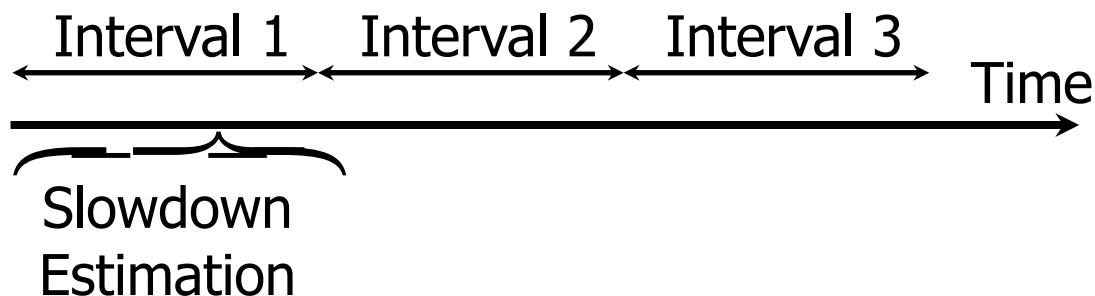
Source Throttling: A Fairness Substrate

- Key idea: Manage inter-thread interference at the **cores (sources)**, **not** at the **shared resources**
- **Dynamically estimate unfairness** in the memory system
- Feed back this information into a controller
- **Throttle cores' memory access rates** accordingly
 - Whom to throttle and by how much depends on performance target (throughput, fairness, per-thread QoS, etc)
 - E.g., if unfairness > system-software-specified target then **throttle down** core causing unfairness & **throttle up** core that was unfairly treated
- Ebrahimi et al., "**Fairness via Source Throttling**," ASPLOS'10, TOCS'12.

Fairness via Source Throttling (FST)

- Two components (interval-based)
- Run-time unfairness evaluation (in hardware)
 - Dynamically estimates the unfairness (application slowdowns) in the memory system
 - Estimates which application is slowing down which other
- Dynamic request throttling (hardware or software)
 - Adjusts how aggressively each core makes requests to the shared resources
 - Throttles down request rates of cores causing unfairness
 - Limit miss buffers, limit injection rate

Fairness via Source Throttling (FST) [ASPLOS'10]



FST



- 1- Estimating system unfairness
- 2- Find app. with the highest slowdown (App-slowest)
- 3- Find app. causing most interference for App-slowest (App-interfering)

```
if (Unfairness Estimate > Target)
{
  1-Throttle down App-interfering
    (limit injection rate and parallelism)
  2-Throttle up App-slowest
}
```

Dynamic Request Throttling

- Goal: Adjust **how aggressively** each core makes requests to the shared memory system
- Mechanisms:
 - Miss Status Holding Register (MSHR) quota
 - Controls the **number of concurrent requests** accessing shared resources from each application
 - Request injection frequency
 - Controls **how often memory requests are issued** to the last level cache from the MSHRs

Dynamic Request Throttling

- **Throttling level** assigned to each core determines both **MSHR quota** and **request injection rate**

Throttling level	MSHR quota	Request Injection Rate
100%	128	Every cycle
50%	64	Every other cycle
25%	32	Once every 4 cycles
10%	12	Once every 10 cycles
5%	6	Once every 20 cycles
4%	5	Once every 25 cycles
3%	3	Once every 30 cycles
2%	2	Once every 50 cycles

Total # of
MSHRs: 128

System Software Support

- Different fairness objectives can be configured by system software
 - Keep maximum slowdown in check
 - Estimated **Max Slowdown** < Target **Max Slowdown**
 - Keep slowdown of particular applications in check to achieve a particular performance target
 - Estimated **Slowdown(i)** < Target **Slowdown(i)**
- Support for thread priorities
 - $\text{Weighted Slowdown}(i) = \text{Estimated Slowdown}(i) \times \text{Weight}(i)$

Source Throttling Results: Takeaways

- Source throttling alone provides better performance than a combination of “smart” memory scheduling and fair caching
 - Decisions made at the memory scheduler and the cache sometimes contradict each other
- Neither source throttling alone nor “smart resources” alone provides the best performance
- Combined approaches are even more powerful
 - Source throttling and resource-based interference control

Source Throttling: Ups and Downs

■ Advantages

- + Core/request throttling is easy to implement: no need to change the memory scheduling algorithm
- + Can be a general way of handling shared resource contention
- + Can reduce overall load/contention in the memory system

■ Disadvantages

- Requires slowdown estimations → difficult to estimate
- Thresholds can become difficult to optimize
 - throughput loss due to too much throttling
 - can be difficult to find an overall-good configuration

More on Source Throttling (I)

- Eiman Ebrahimi, Chang Joo Lee, Onur Mutlu, and Yale N. Patt, **"Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems"**
*Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**), pages 335-346, Pittsburgh, PA, March 2010.*
Slides (pdf)

Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems

Eiman Ebrahimi[†] Chang Joo Lee[†] Onur Mutlu[§] Yale N. Patt[†]

[†]Department of Electrical and Computer Engineering
The University of Texas at Austin
{ebrahimi, cjlee, patt}@ece.utexas.edu

[§]Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
onur@cmu.edu

More on Source Throttling (II)

- Kevin Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu, **"HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"**
Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), New York, NY, October 2012. [Slides \(pptx\)](#) [\(pdf\)](#)

HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Kai-Wei Chang, Rachata Ausavarungnirun, Chris Fallin, Onur Mutlu
Carnegie Mellon University
{kevincha, rachata, cfallin, onur}@cmu.edu

More on Source Throttling (III)

- George Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan,
"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"
Proceedings of the 2012 ACM SIGCOMM Conference
(SIGCOMM), Helsinki, Finland, August 2012. [Slides \(pptx\)](#)

On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-Core Interconnects

George Nychis[†], Chris Fallin[†], Thomas Moscibroda[§], Onur Mutlu[†], Srinivasan Seshan[†]

[†] Carnegie Mellon University
{gnychis,cfallin,onur,srini}@cmu.edu

[§] Microsoft Research Asia
moscitho@microsoft.com

Fundamental Interference Control Techniques

- **Goal:** to reduce/control interference

1. **Prioritization** or request scheduling
2. **Data mapping** to banks/channels/ranks
3. **Core/source throttling**

4. **Application/thread scheduling**

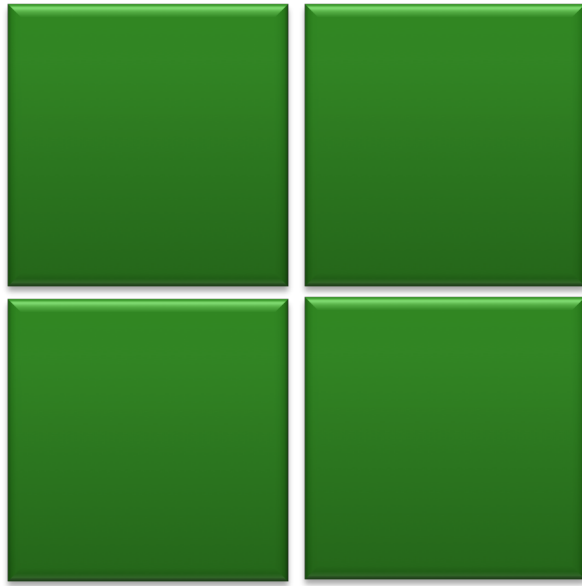
Idea: Pick threads that do not badly interfere with each other to be scheduled together on cores sharing the memory system

Application-to-Core Mapping to Reduce Interference

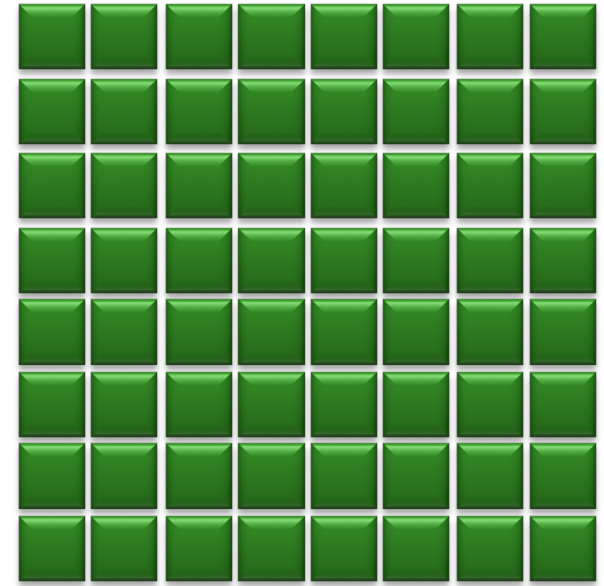
- Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi,
"Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems"
Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), Shenzhen, China, February 2013.
Slides (pptx)

- Key ideas:
 - ❑ Cluster threads to memory controllers (to reduce across chip interference)
 - ❑ Isolate interference-sensitive (low-intensity) applications in a separate cluster (to reduce interference from high-intensity applications)
 - ❑ Place applications that benefit from memory bandwidth closer to the controller

Multi-Core to Many-Core



Multi-Core



Many-Core

Many-Core On-Chip Communication

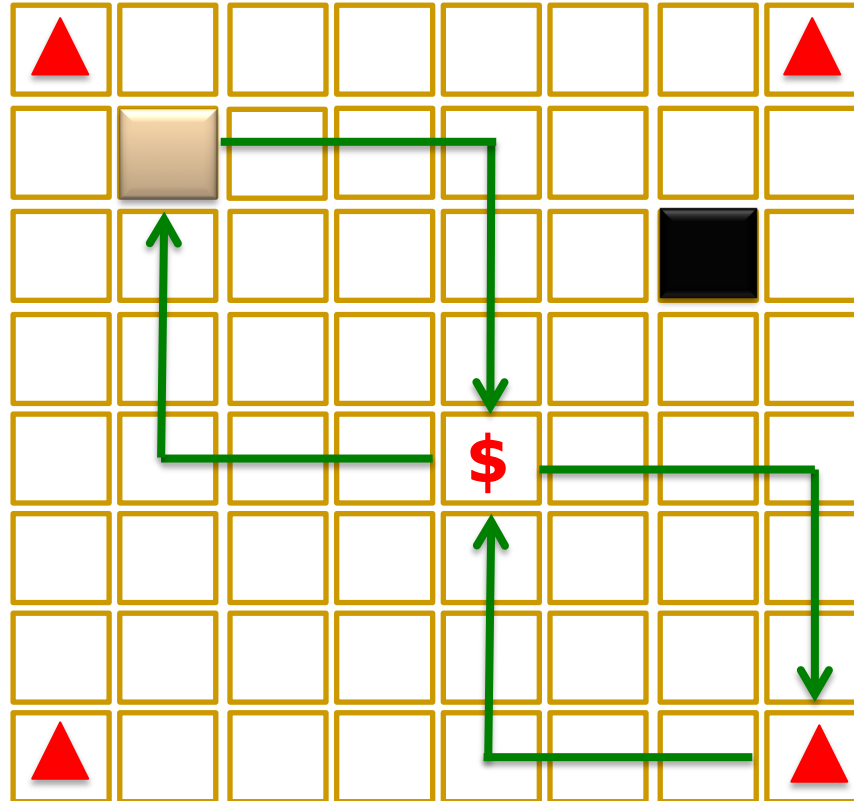
Applications



Light



Heavy



**Memory
Controller**

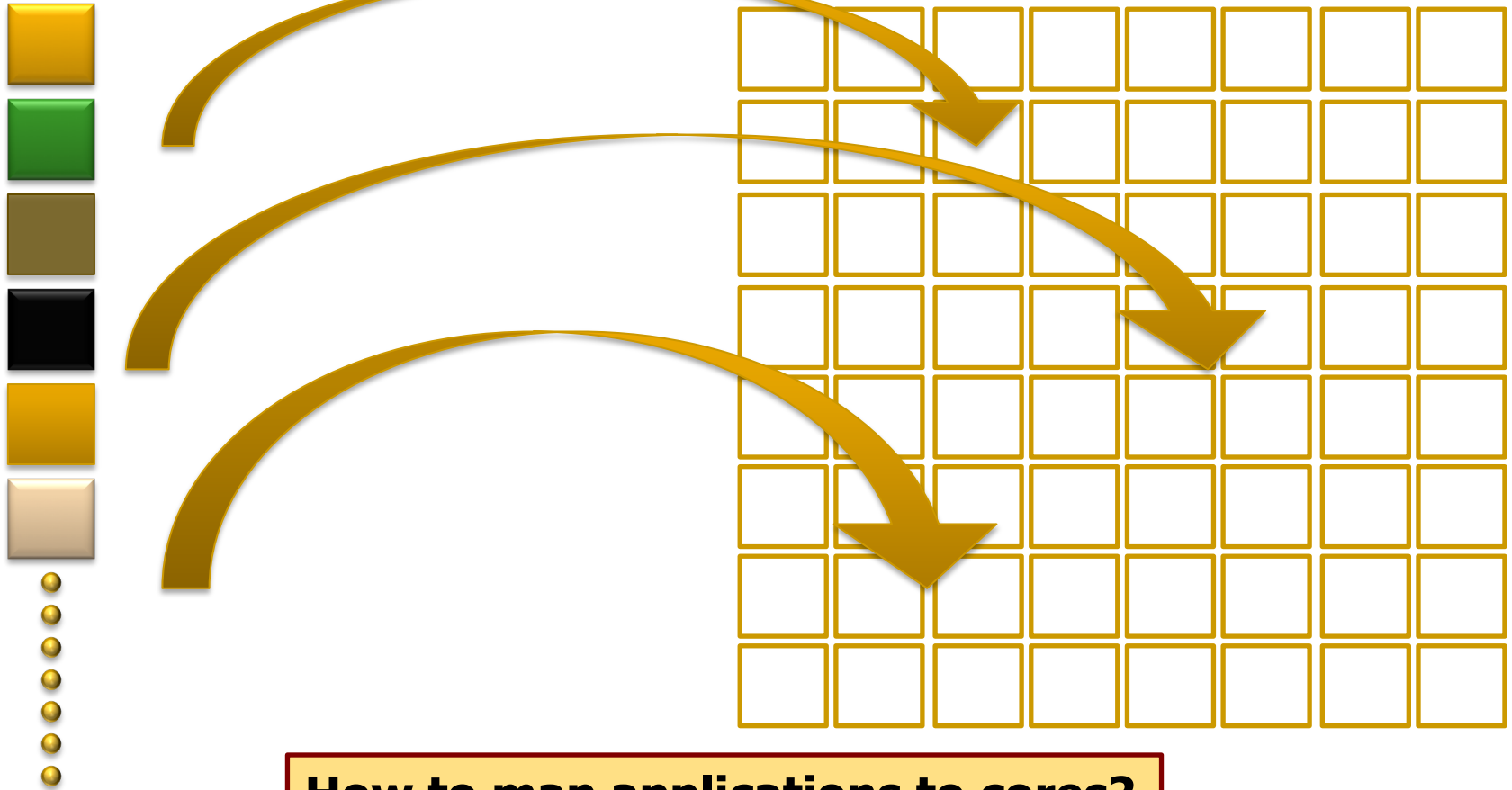


**Shared
Cache Bank**

Problem: Spatial Task Scheduling

Applications

Cores



How to map applications to cores?

Challenges in Spatial Task Scheduling

Applications

Cores

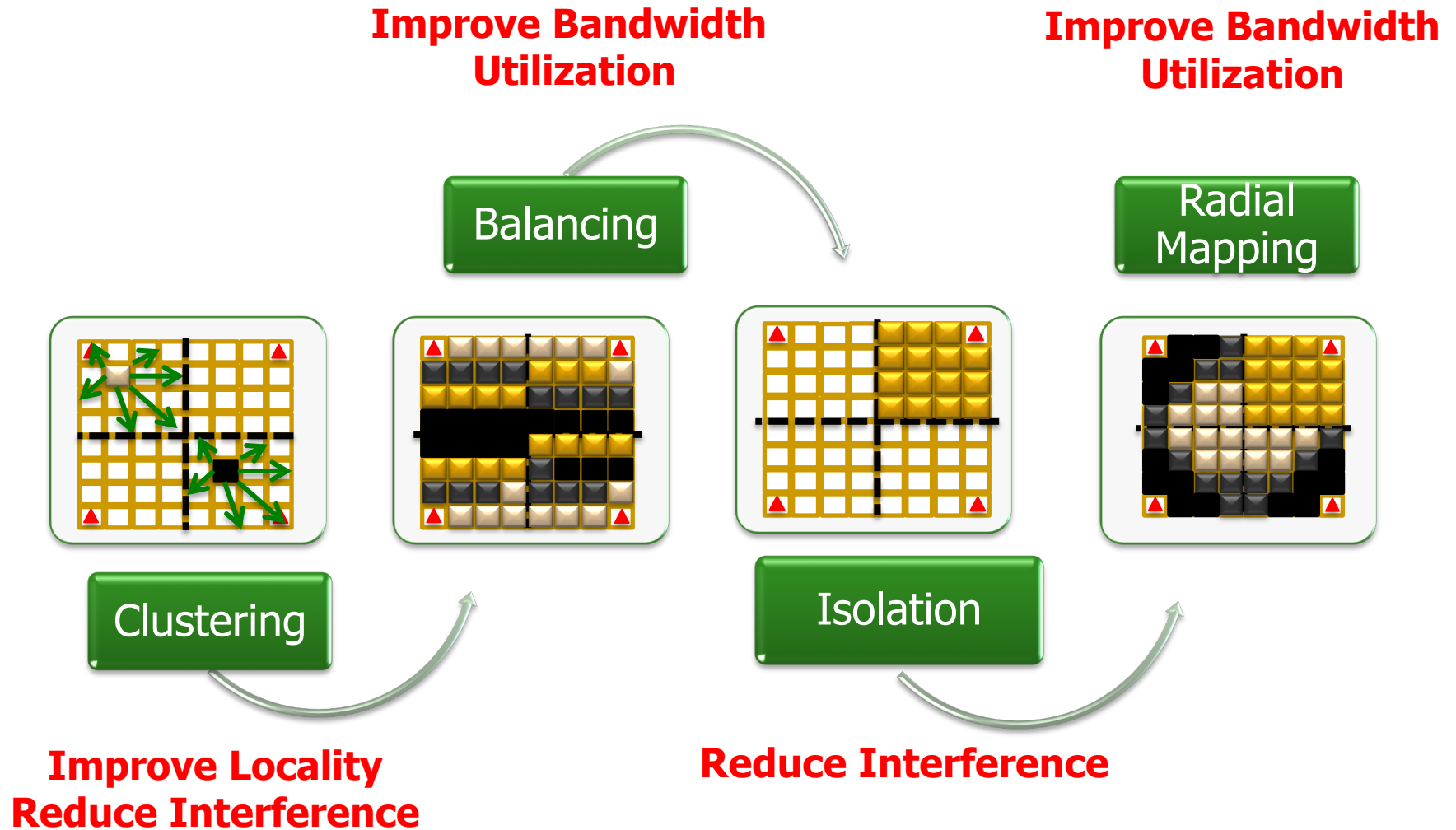


How to reduce communication distance?

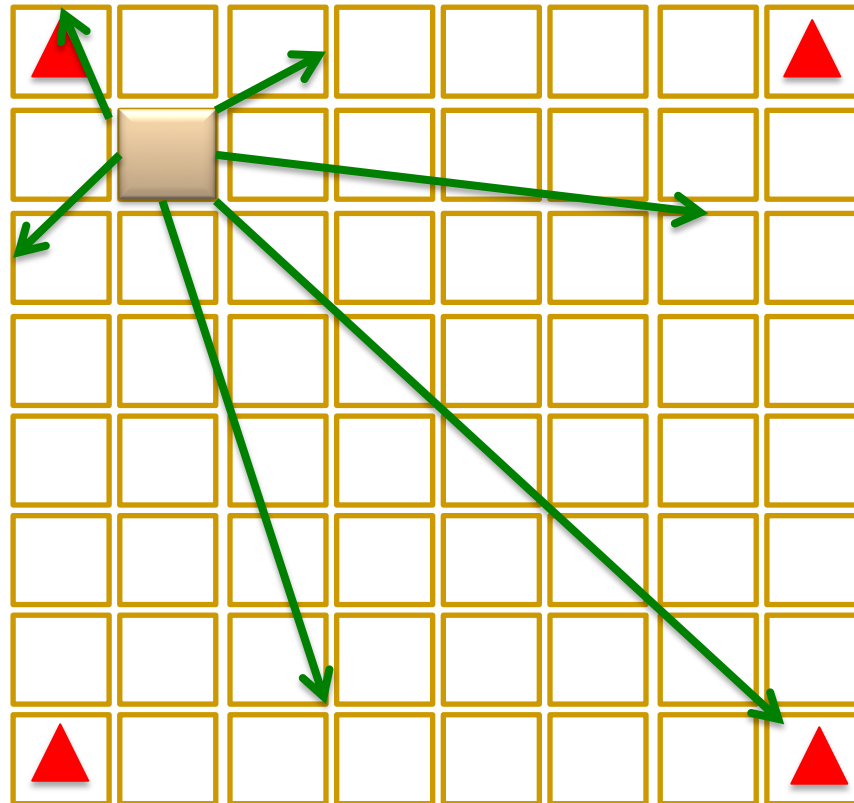
How to reduce destructive interference between applications?

How to prioritize applications to improve throughput?

Application-to-Core Mapping



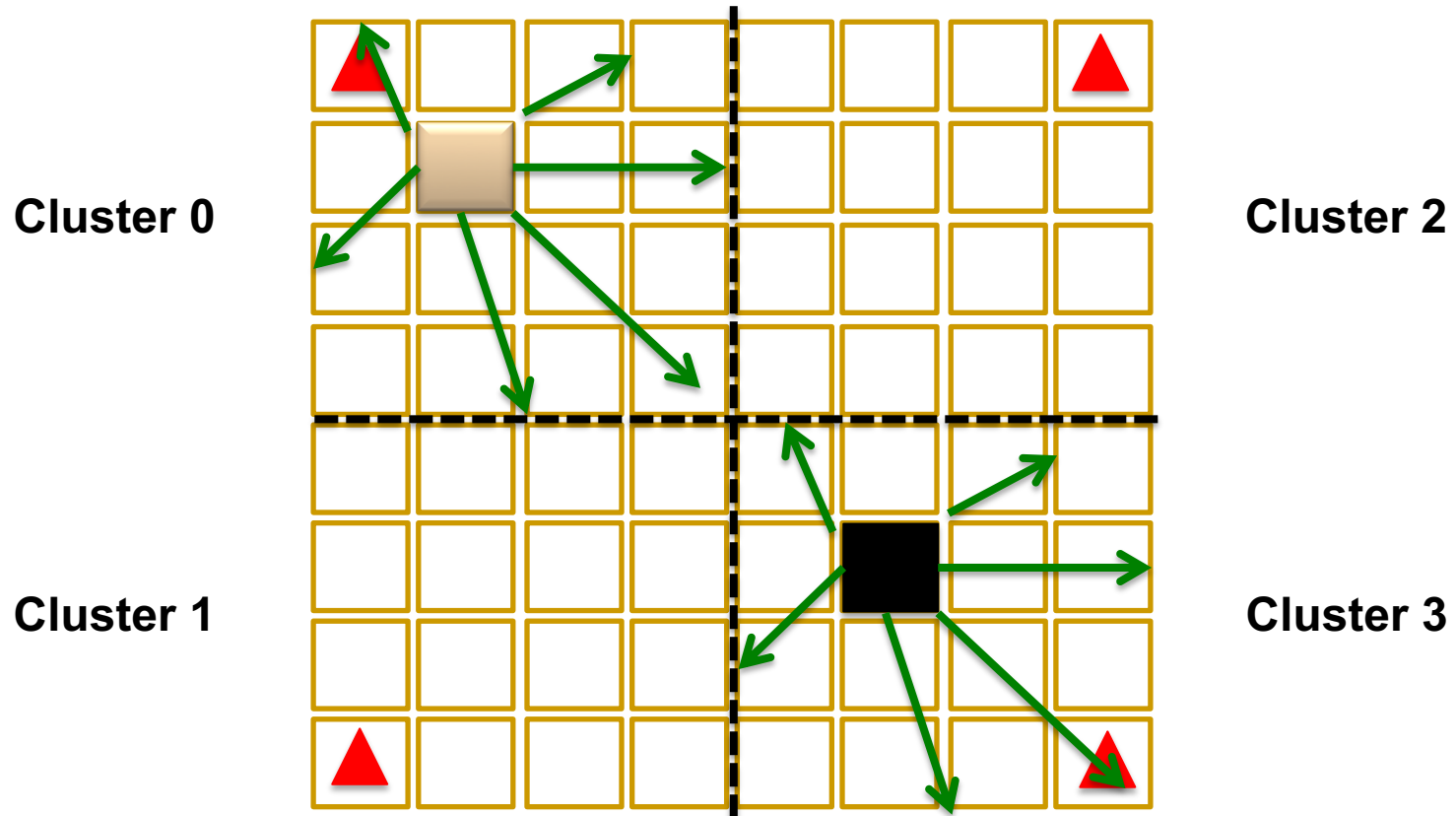
Step 1 — Clustering



 **Memory
Controller**

Inefficient data mapping to memory and caches

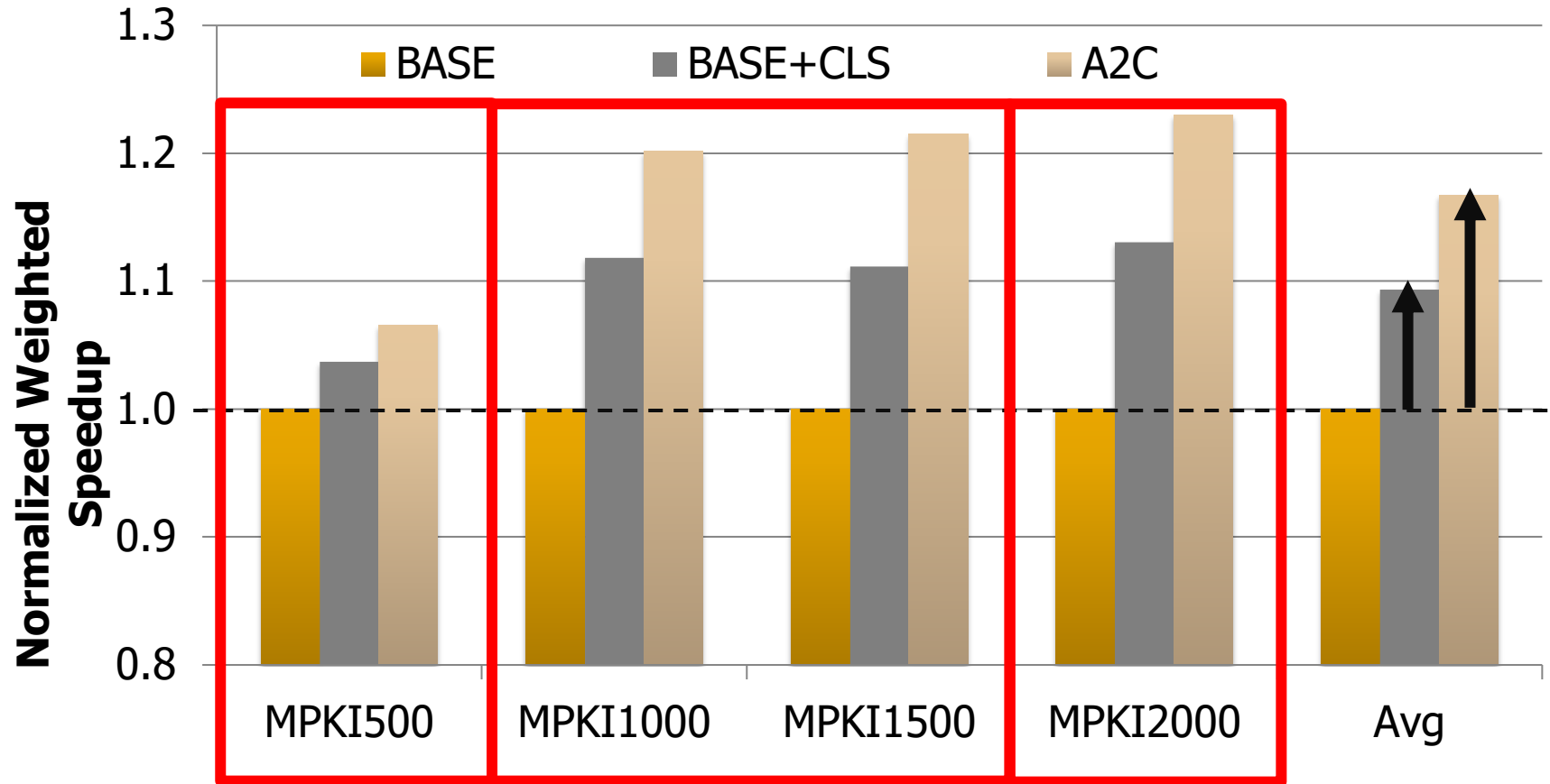
Step 1 — Clustering



Improved Locality

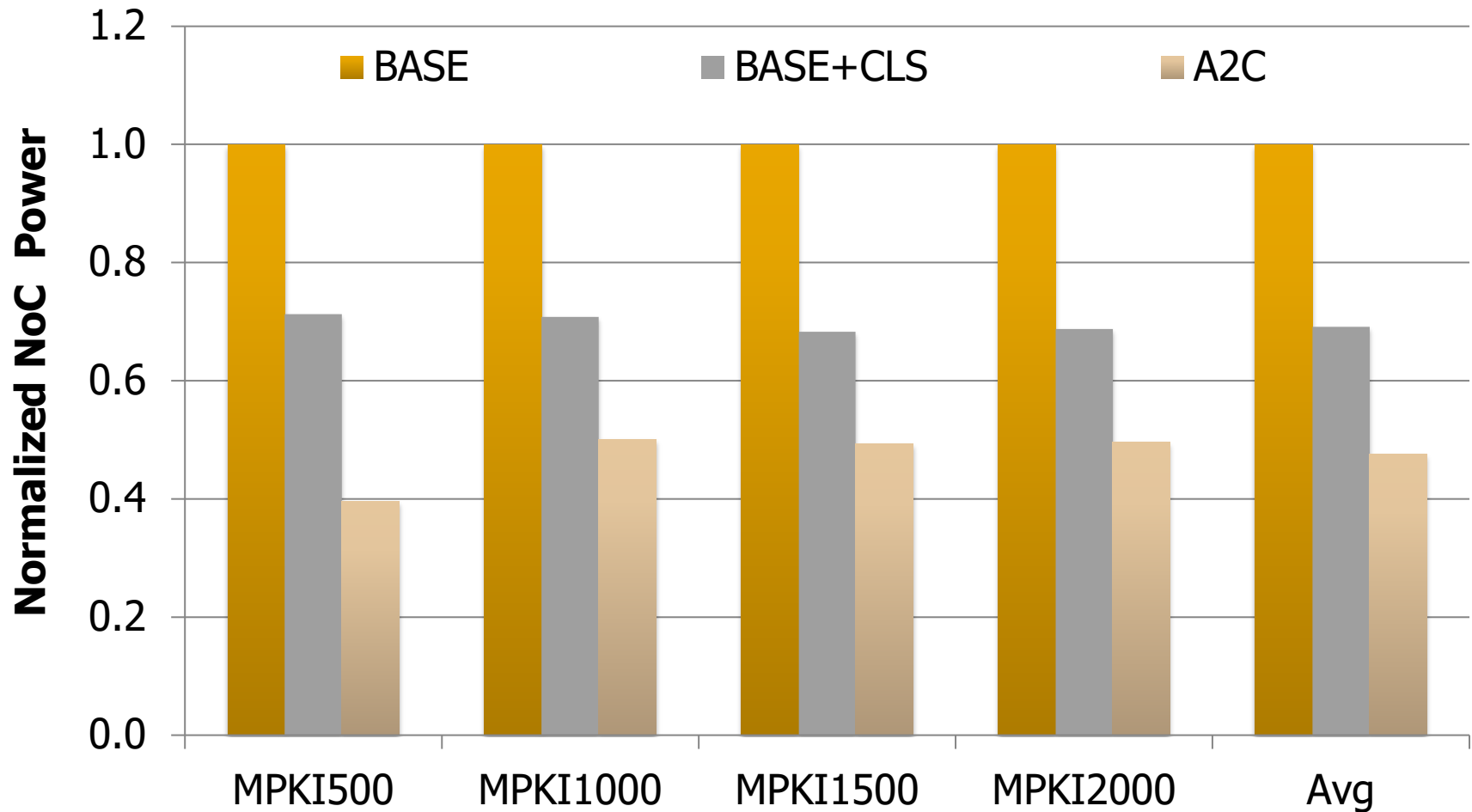
Reduced Interference

System Performance



System performance improves by 17%

Network Power



Average network power consumption reduces by 52%

More on App-to-Core Mapping

- Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi,

"Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems"

Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), Shenzhen, China, February 2013.

Slides (pptx)

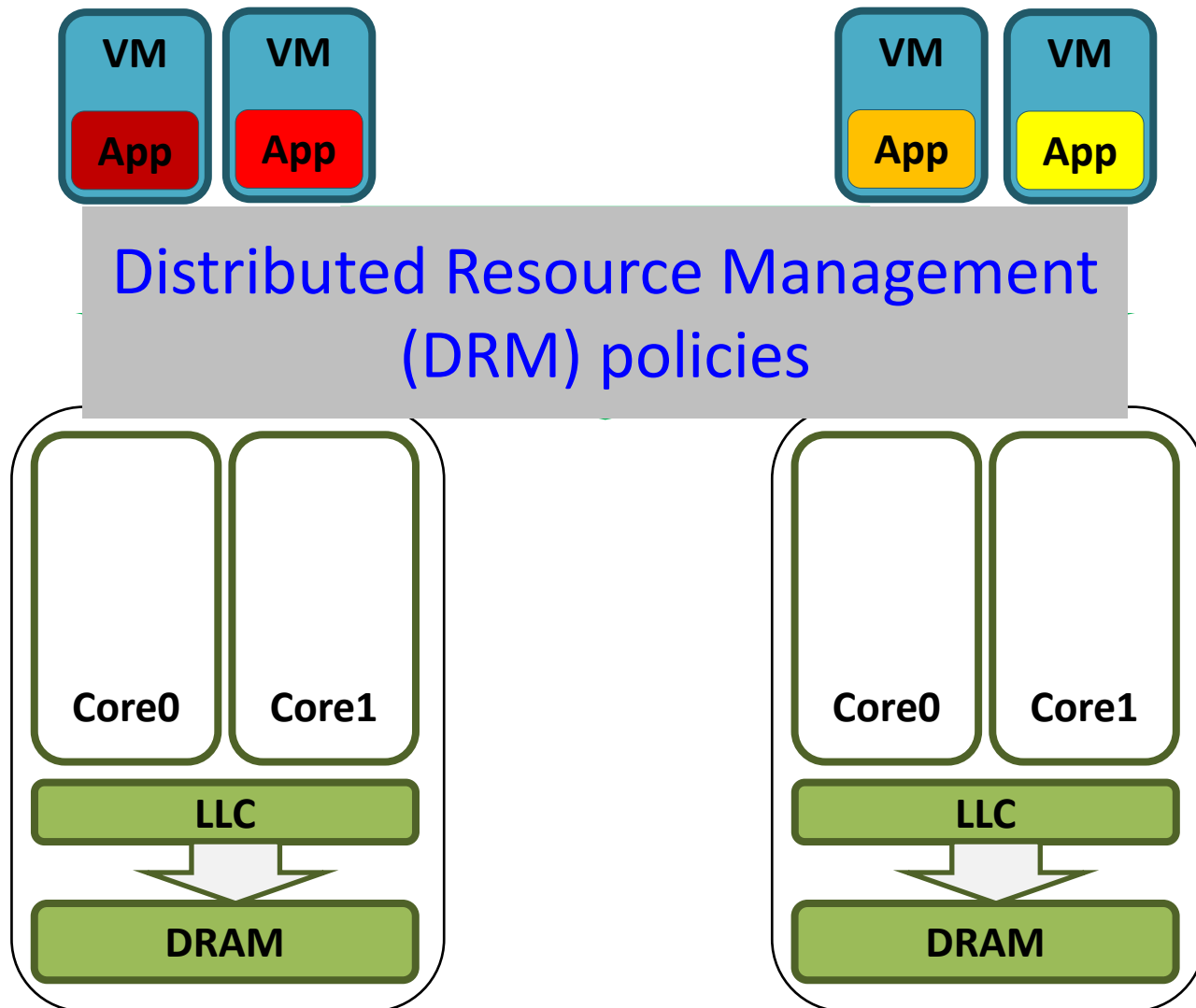
Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems

Reetuparna Das* Rachata Ausavarungnirun† Onur Mutlu† Akhilesh Kumar‡ Mani Azimi‡
University of Michigan* Carnegie Mellon University† Intel Labs‡

Interference-Aware Thread Scheduling

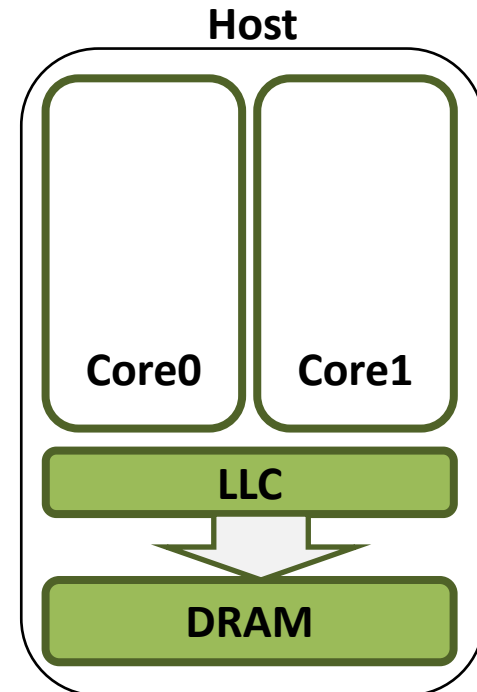
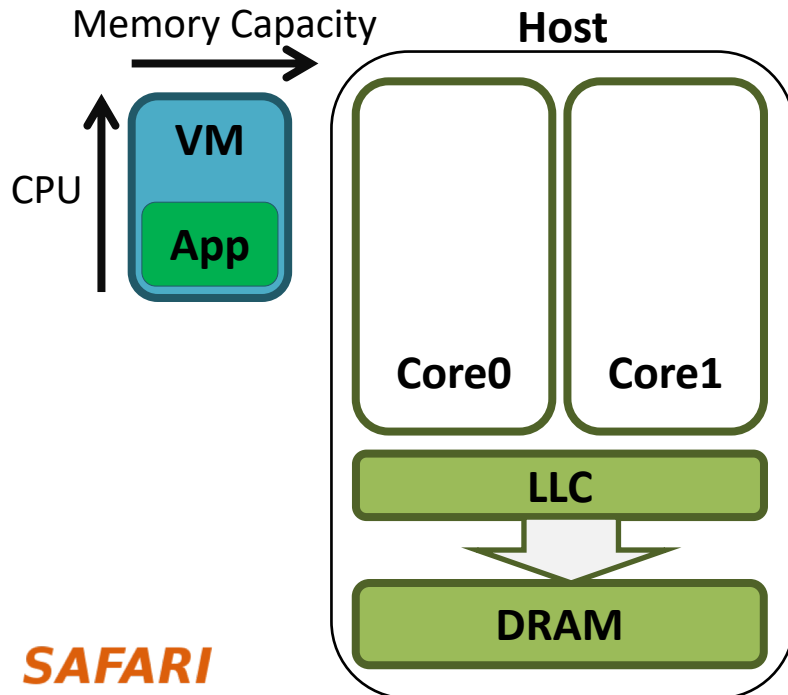
- An example from scheduling in compute clusters (data centers)
- Data centers can be running virtual machines

Virtualized Cluster



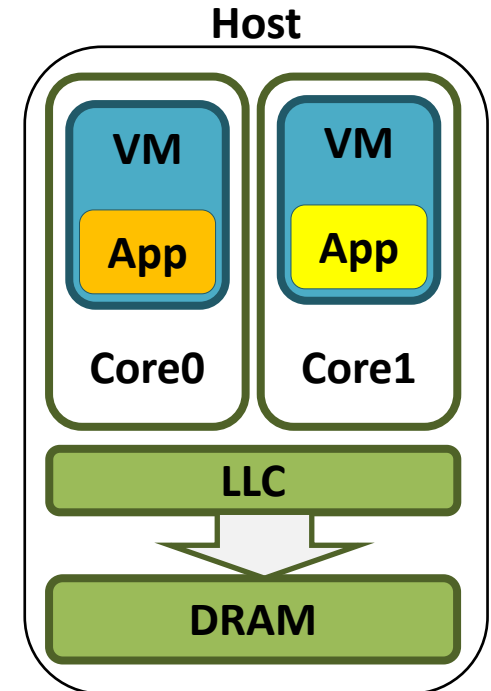
Conventional DRM Policies

Based on **operating-system-level metrics**
e.g., **CPU utilization**, **memory capacity**
demand



Microarchitecture-level Interference

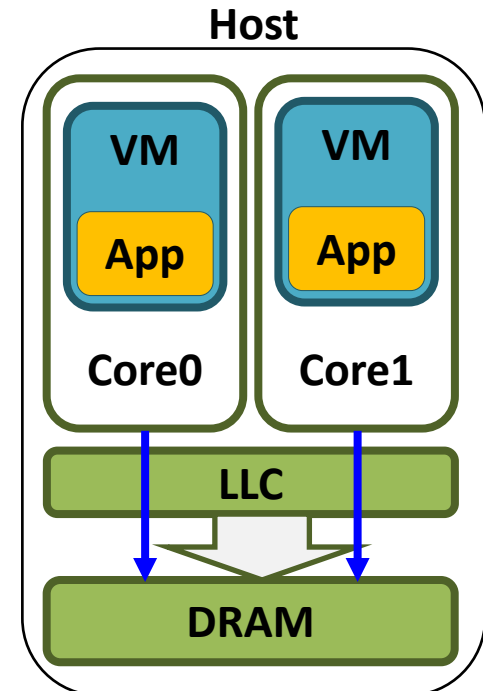
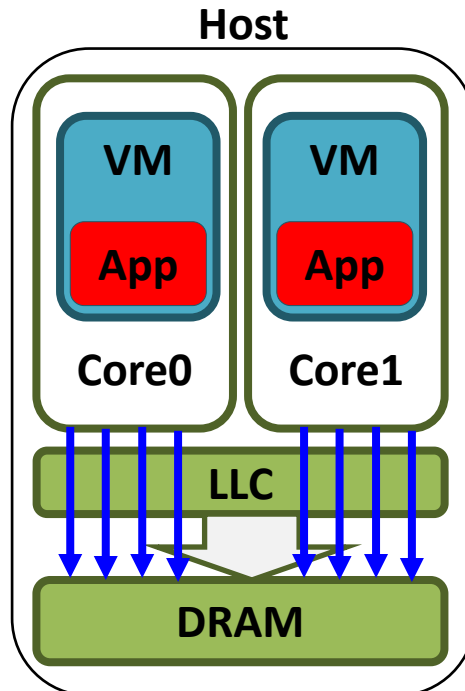
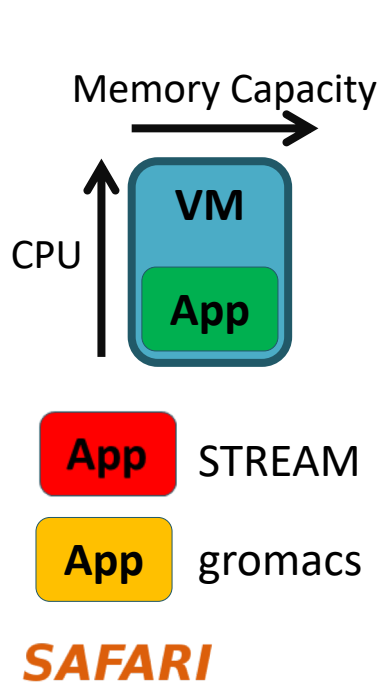
- VMs within a host compete for:
 - Shared cache capacity
 - Shared memory bandwidth



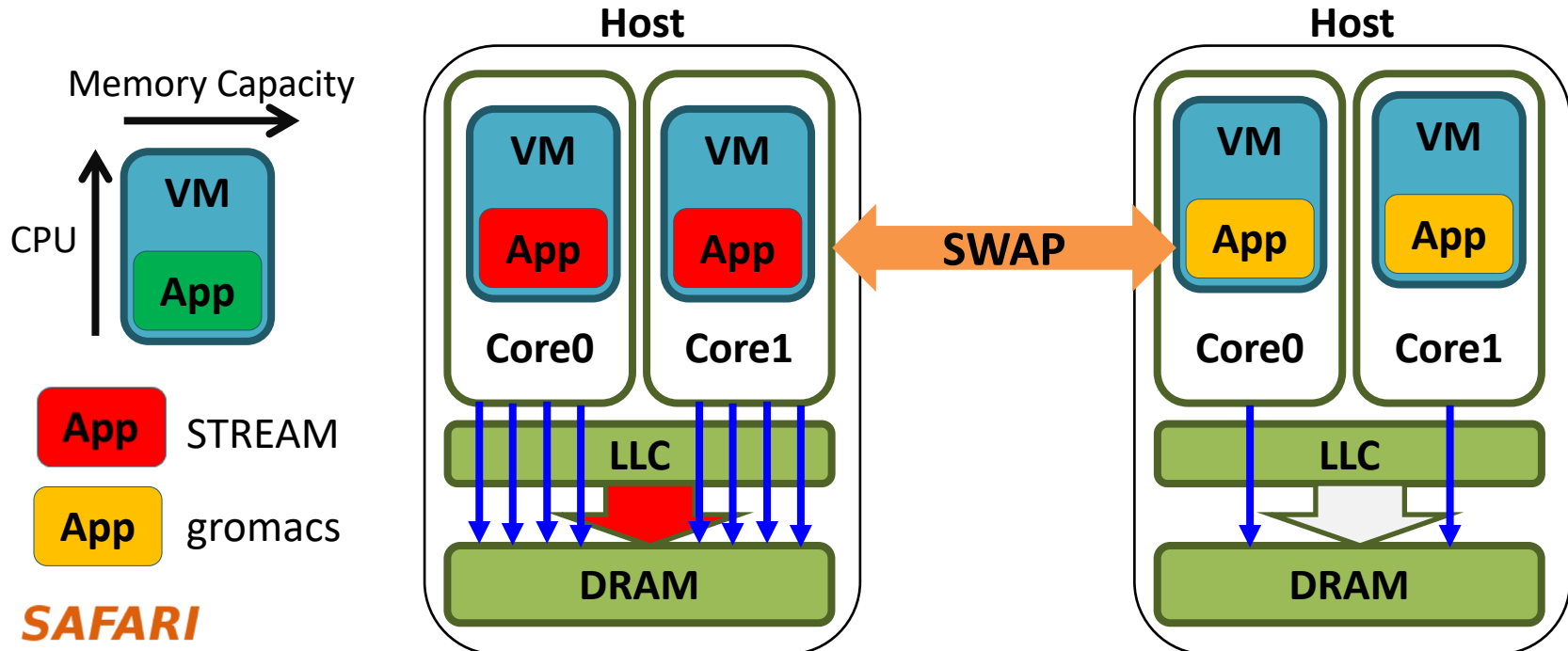
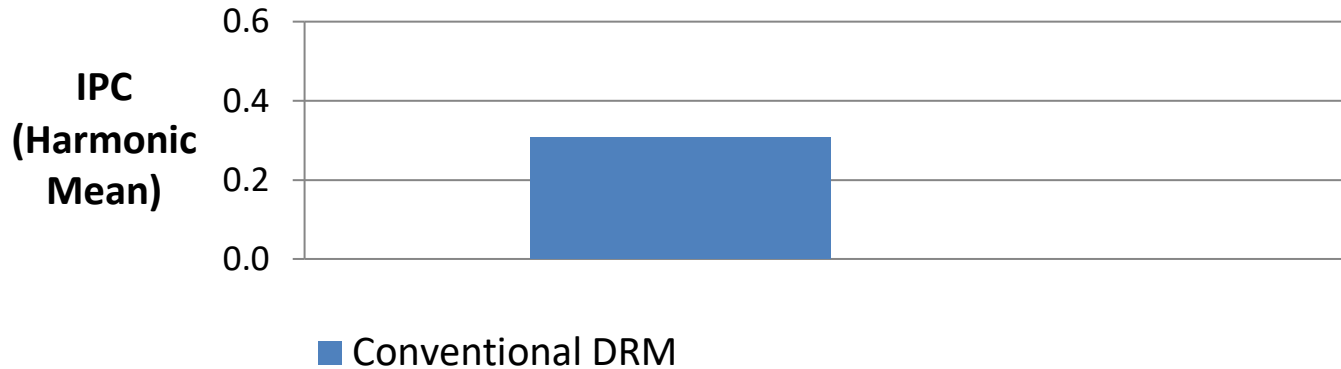
Can operating-system-level metrics capture the microarchitecture-level resource interference?

Microarchitecture Unawareness

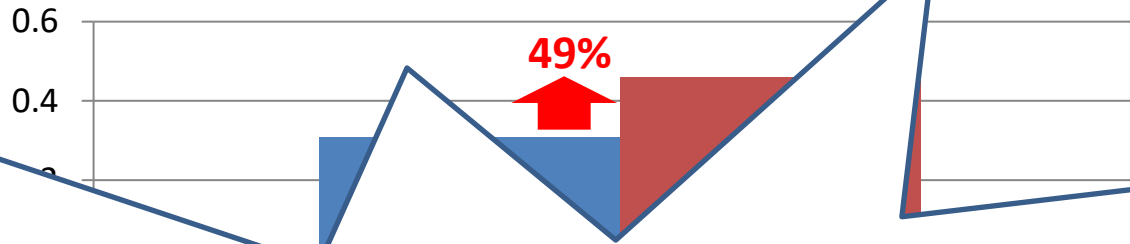
VM	Operating-system-level metrics		Microarchitecture-level metrics	
	CPU Utilization	Memory Capacity	LLC Hit Ratio	Memory Bandwidth
App	92%	369 MB	2%	2267 MB/s
App	93%	348 MB	98%	1 MB/s



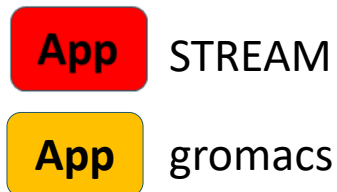
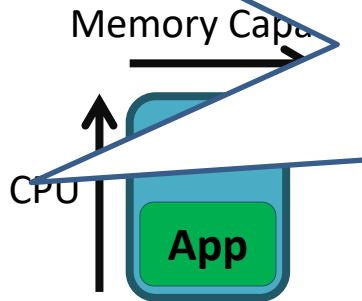
Impact on Performance



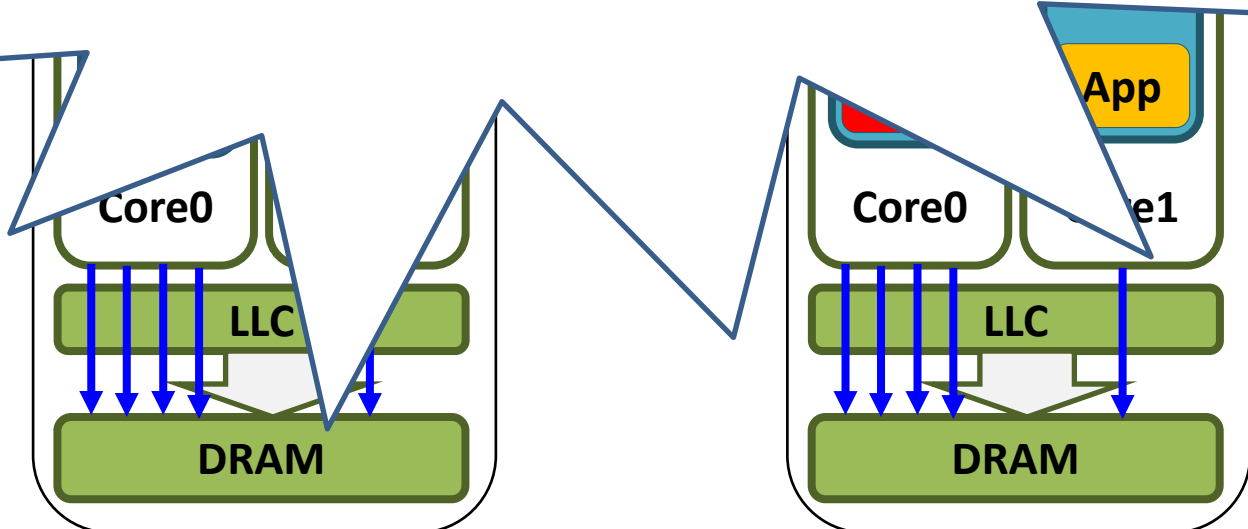
Impact on Performance



**We need microarchitecture-level interference awareness in
DRAM!**



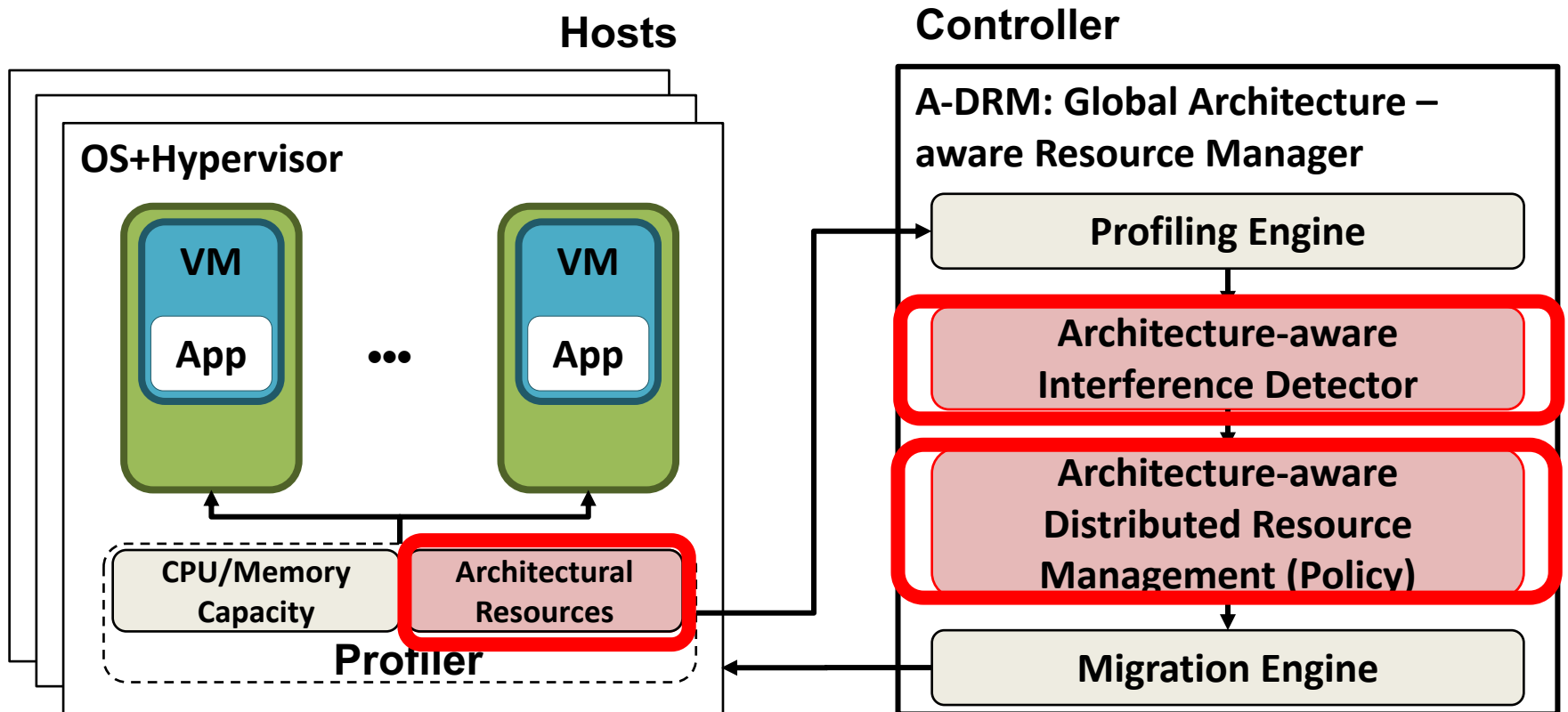
SAFARI



A-DRM: Architecture-aware DRM

- **Goal**: Take into account microarchitecture-level shared resource interference
 - Shared cache capacity
 - Shared memory bandwidth
- **Key Idea**:
 - Monitor and detect microarchitecture-level shared resource interference
 - Balance microarchitecture-level resource usage across cluster to minimize memory interference while maximizing system performance

A-DRM: Architecture-aware DRM



More on Architecture-Aware DRM

- Hui Wang, Canturk Isci, Lavanya Subramanian, Jongmoo Choi, Depei Qian, and Onur Mutlu,

"A-DRM: Architecture-aware Distributed Resource Management of Virtualized Clusters"

Proceedings of the 11th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), Istanbul, Turkey, March 2015.

[[Slides \(pptx\)](#) ([pdf](#))]

A-DRM: Architecture-aware Distributed Resource Management of Virtualized Clusters

Hui Wang^{†*}, Canturk Isci[‡], Lavanya Subramanian^{*}, Jongmoo Choi^{‡*}, Depei Qian[†], Onur Mutlu^{*}

[†]Beihang University, [‡]IBM Thomas J. Watson Research Center, ^{*}Carnegie Mellon University, [‡]Dankook University

{hui.wang, depei.qian}@buaa.edu.cn, canturk@us.ibm.com, {lsubrama, onur}@cmu.edu, choijm@dankook.ac.kr

Interference-Aware Thread Scheduling

■ Advantages

- + Can eliminate/minimize interference by scheduling “symbiotic applications” together (as opposed to just managing the interference)
- + Less intrusive to hardware (less need to modify the hardware resources)

■ Disadvantages and Limitations

- High overhead to migrate threads and data between cores and machines
- Does not work (well) if all threads are similar and they interfere

Summary

Summary: Fundamental Interference Control Techniques

- **Goal:** to reduce/control interference

- 1. **Prioritization** or request scheduling

- 2. **Data mapping** to banks/channels/ranks

- 3. **Core/source throttling**

- 4. **Application/thread scheduling**

Best is to combine all. How would you do that?

Summary: Memory QoS Approaches and Techniques

- Approaches: **Smart** vs. **dumb** resources
 - Smart resources: QoS-aware memory scheduling
 - Dumb resources: Source throttling; channel partitioning
 - Both approaches are effective in reducing interference
 - No single best approach for all workloads
- Techniques: Request/thread **scheduling**, source **throttling**, memory **partitioning**
 - All approaches are effective in reducing interference
 - Can be applied at different levels: hardware vs. software
 - No single best technique for all workloads
- **Combined approaches and techniques are the most powerful**
 - **Integrated Memory Channel Partitioning and Scheduling [MICRO'11]**

Summary: Memory Interference and QoS

- QoS-unaware memory → uncontrollable and unpredictable system
- Providing QoS awareness improves performance, predictability, fairness, and utilization of the memory system
- Discussed many new techniques to:
 - Minimize memory interference
 - Provide predictable performance
- Many new research ideas needed for integrated techniques and closing the interaction with software

What Did We Not Cover?

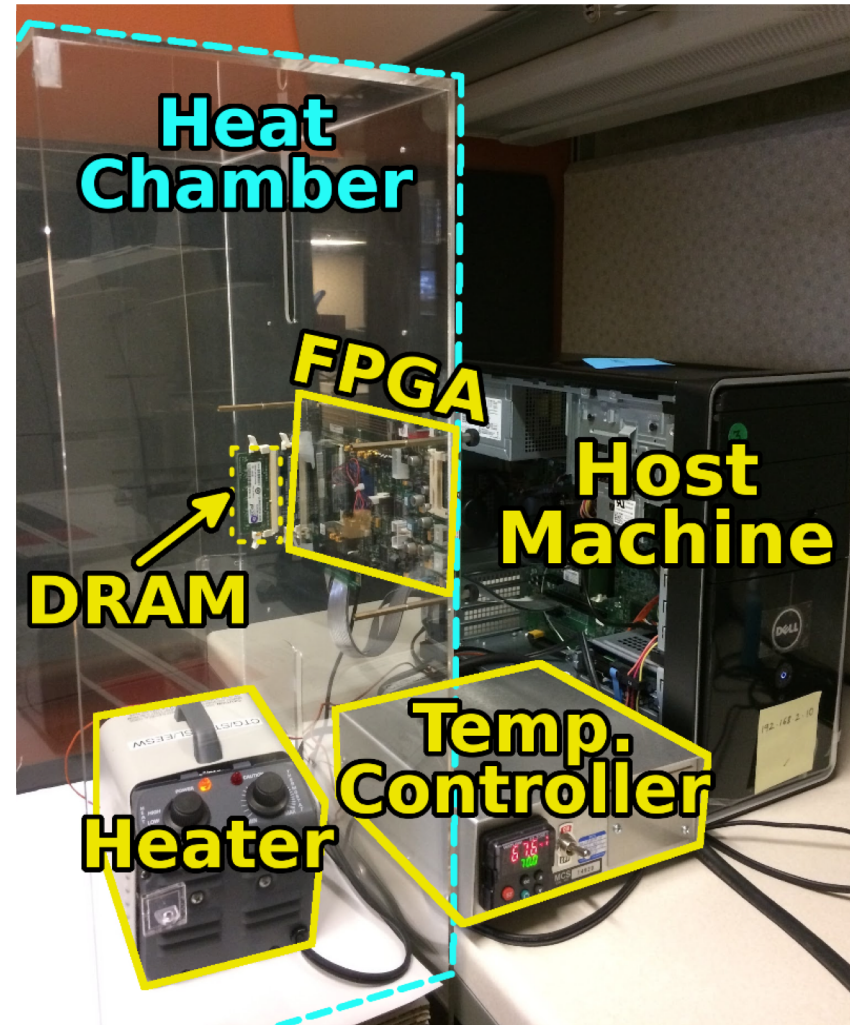
- Prefetch-aware shared resource management
- DRAM-controller co-design
- Cache interference management
- **Interconnect interference management**
- Write-read scheduling
- **DRAM designs to reduce interference**
- Interference issues in near-memory processing
- ...

What the Future May Bring

- **Simple** yet powerful interference control and scheduling mechanisms
 - memory scheduling + interconnect scheduling
- **Real** implementations and investigations
 - SoftMC infrastructure, FPGA-based implementations
- Interference and QoS in the presence of **even more heterogeneity**
 - PIM, accelerators, ...

SoftMC: Open Source DRAM Infrastructure

- Hasan Hassan et al., “**SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies**,” HPCA 2017.
- Flexible
- Easy to Use (C++ API)
- Open-source
github.com/CMU-SAFARI/SoftMC



- <https://github.com/CMU-SAFARI/SoftMC>

SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies

Hasan Hassan^{1,2,3} Nandita Vijaykumar³ Samira Khan^{4,3} Saugata Ghose³ Kevin Chang³
Gennady Pekhimenko^{5,3} Donghyuk Lee^{6,3} Oguz Ergin² Onur Mutlu^{1,3}

¹*ETH Zürich* ²*TOBB University of Economics & Technology* ³*Carnegie Mellon University*
⁴*University of Virginia* ⁵*Microsoft Research* ⁶*NVIDIA Research*

Some Other Ideas ...

Decoupled DMA w/ Dual-Port DRAM

[PACT 2015]

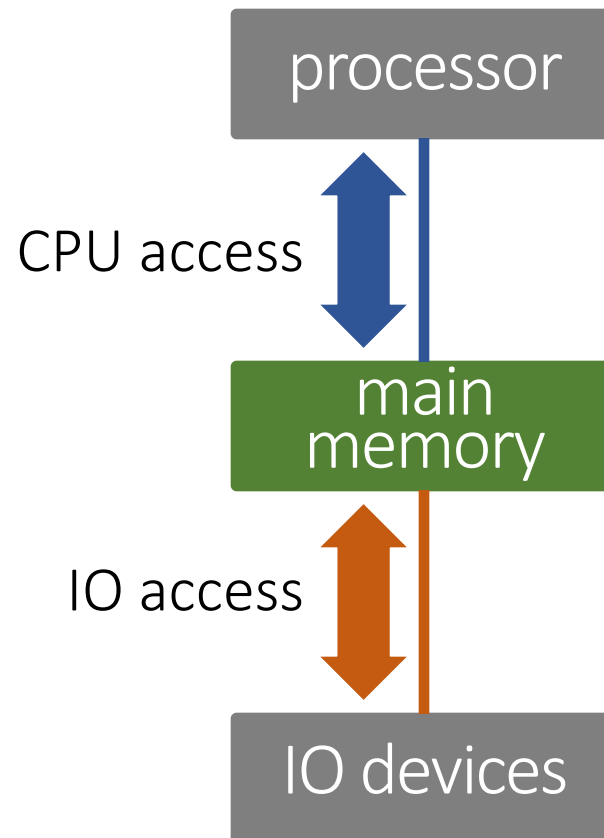
Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM

Decoupled Direct Memory Access

Donghyuk Lee

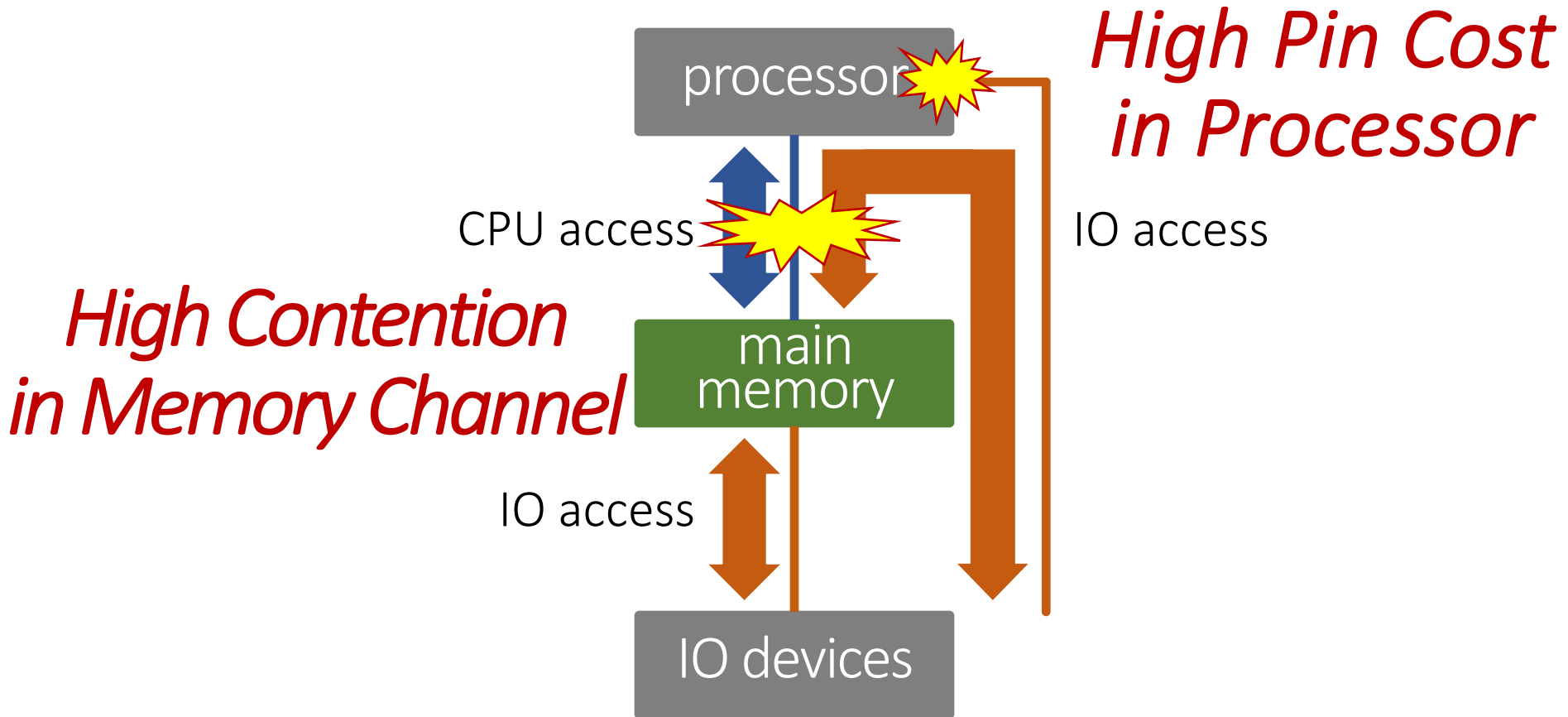
Lavanya Subramanian, Rachata Ausavarungnirun,
Jongmoo Choi, Onur Mutlu

Logical System Organization

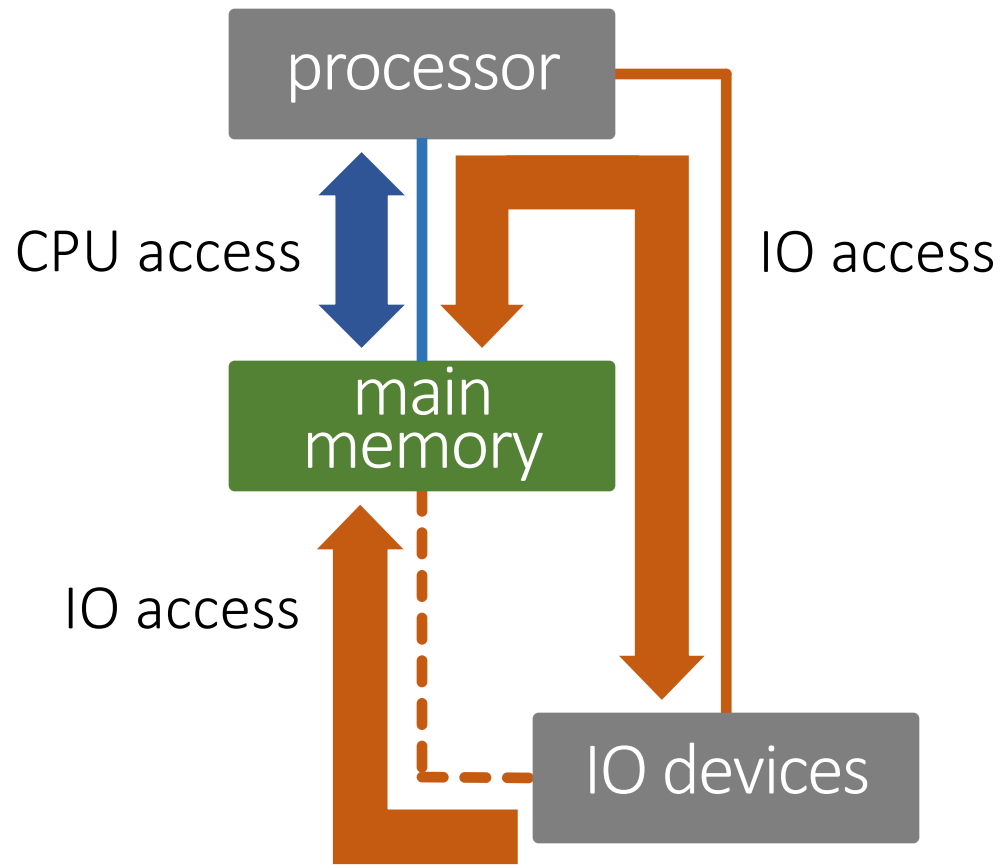


Main memory connects processor and IO devices as an *intermediate layer*

Physical System Implementation



Our Approach



Enabling IO channel,
decoupled & isolated from CPU channel

Executive Summary

- Problem
 - CPU and IO accesses contend for the shared memory channel
- Our Approach: *Decoupled Direct Memory Access (DDMA)*
 - Design new DRAM architecture with two independent data ports
→ *Dual-Data-Port DRAM*
 - Connect one port to CPU and the other port to IO devices
→ *Decouple CPU and IO accesses*
- Application
 - Communication between compute units (e.g., CPU – GPU)
 - In-memory communication (e.g., bulk in-memory copy/init.)
 - Memory-storage communication (e.g., page fault, IO prefetch)
- Result
 - Significant *performance improvement* (20% in 2 ch. & 2 rank system)
 - *CPU pin count reduction* (4.5%)

Outline

1. Problem

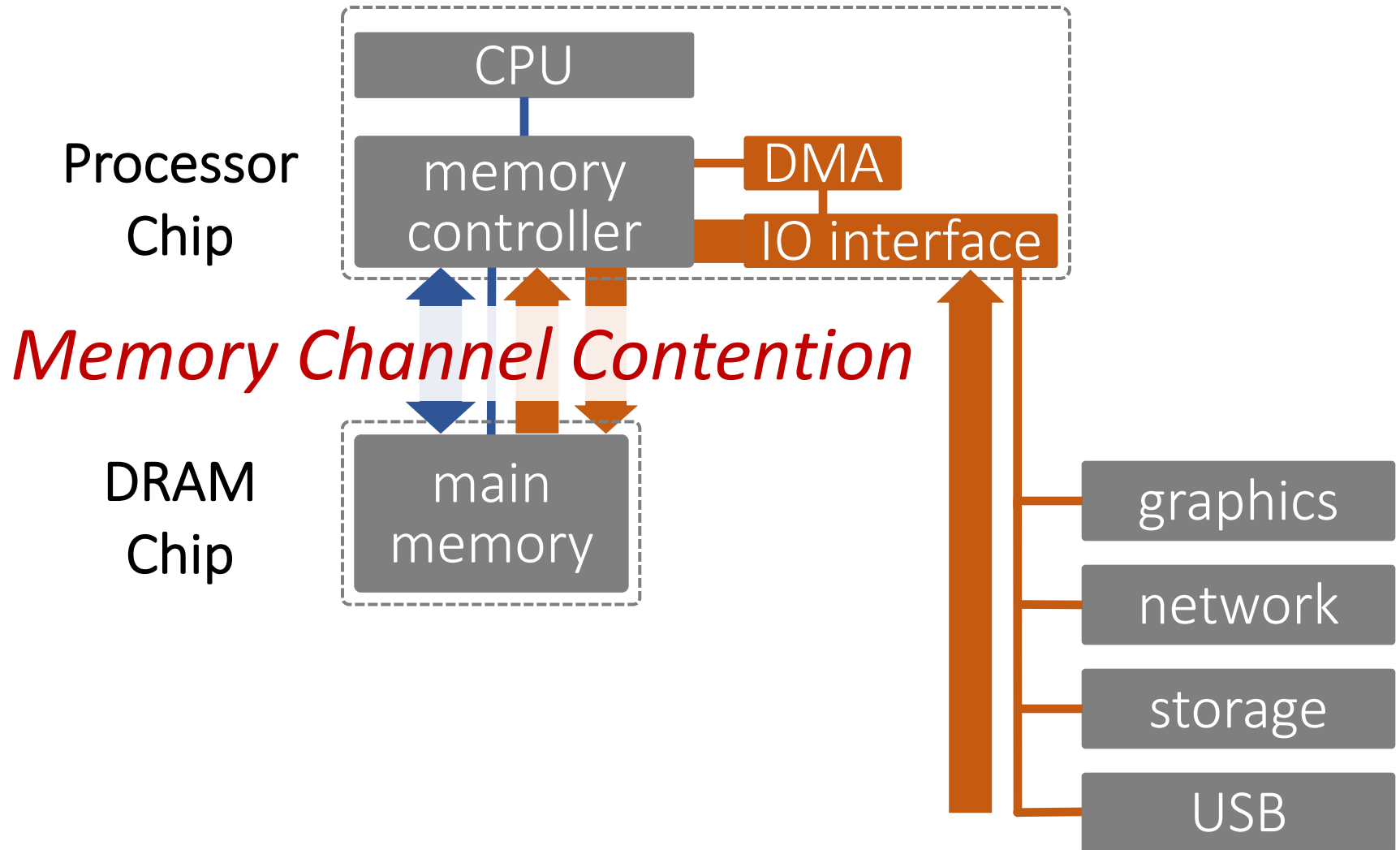
2. Our Approach

3. Dual-Data-Port DRAM

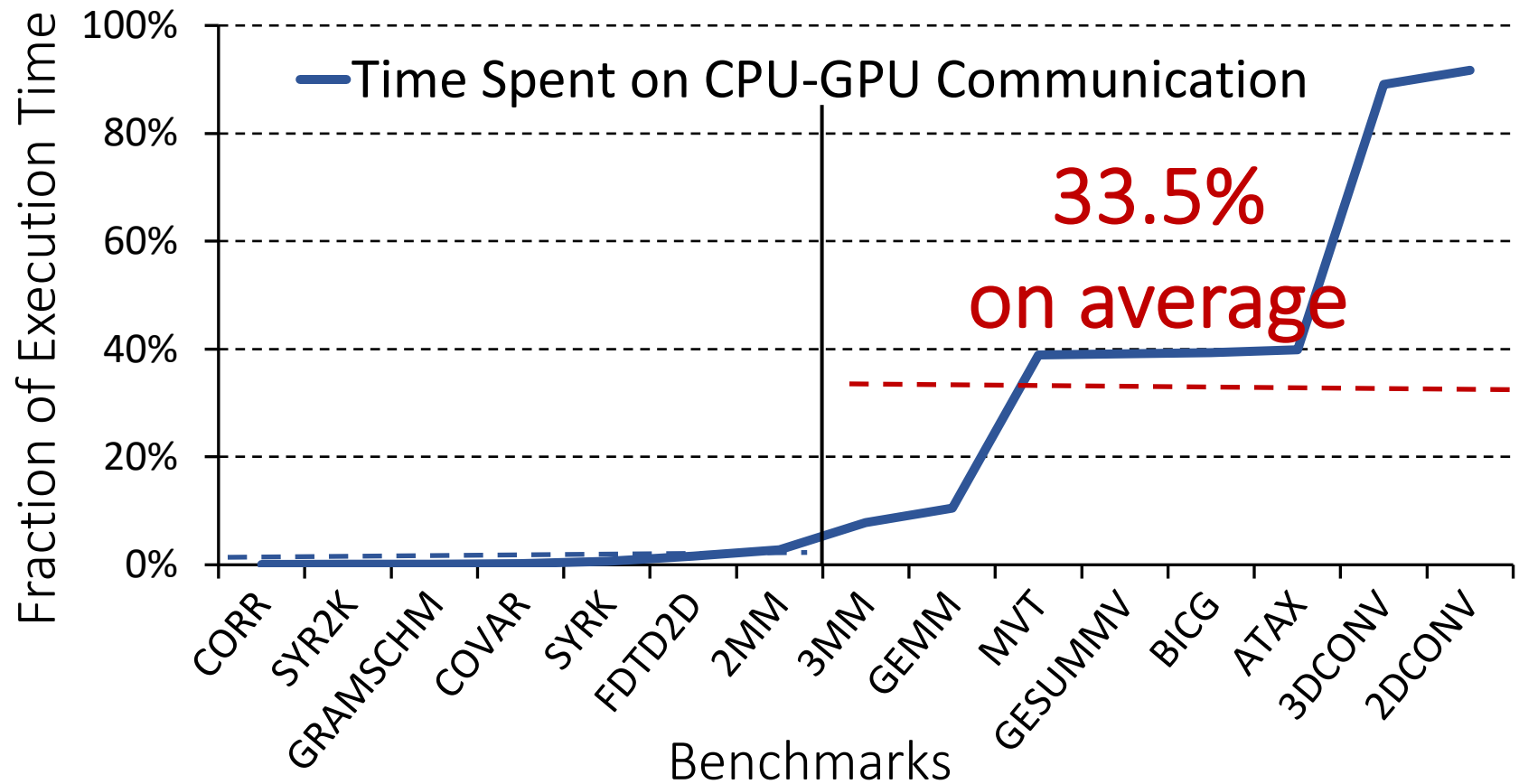
4. Applications for DDMA

5. Evaluation

Problem 1: Memory Channel Contention

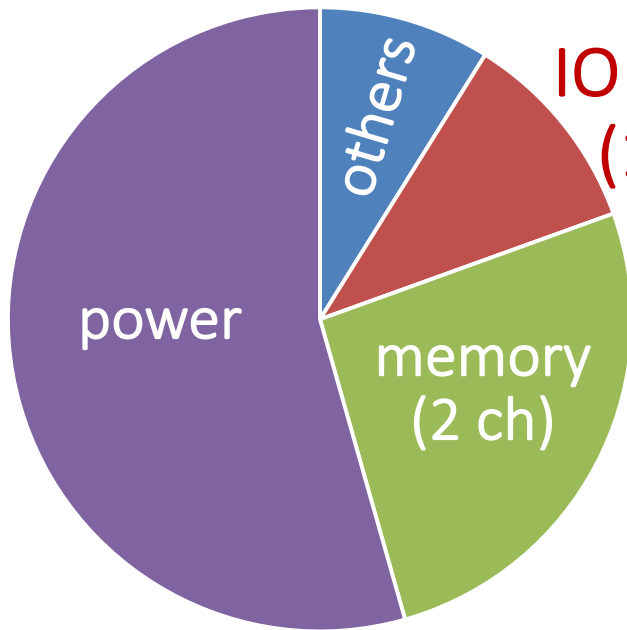


Problem 1: Memory Channel Contention



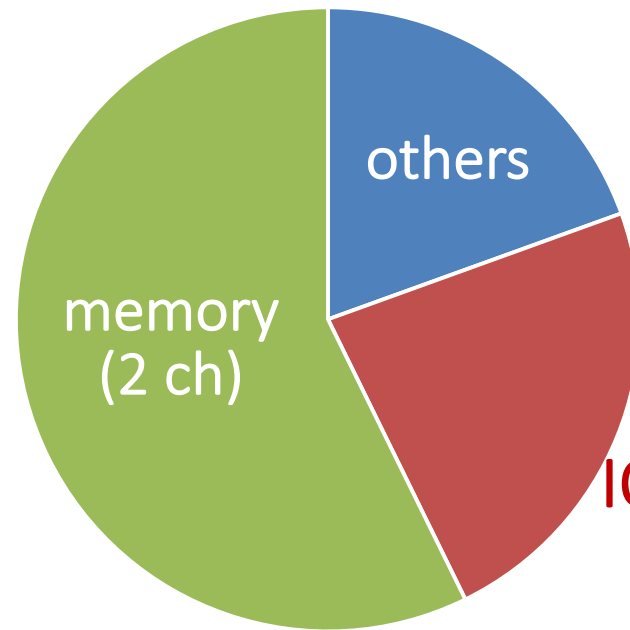
A large fraction of the execution time
is spent on IO accesses

Problem 2: High Cost for IO Interfaces



959 pins in total

Processor Pin Count
(w/ power pins)



359 pins in total

Processor Pin Count
(w/o power pins)

Integrating IO interface on the processor chip
leads to *high area cost*

Shared Memory Channel

- Memory channel contention for IO access and CPU access
- High area cost for integrating IO interfaces on processor chip

Outline

1. Problem

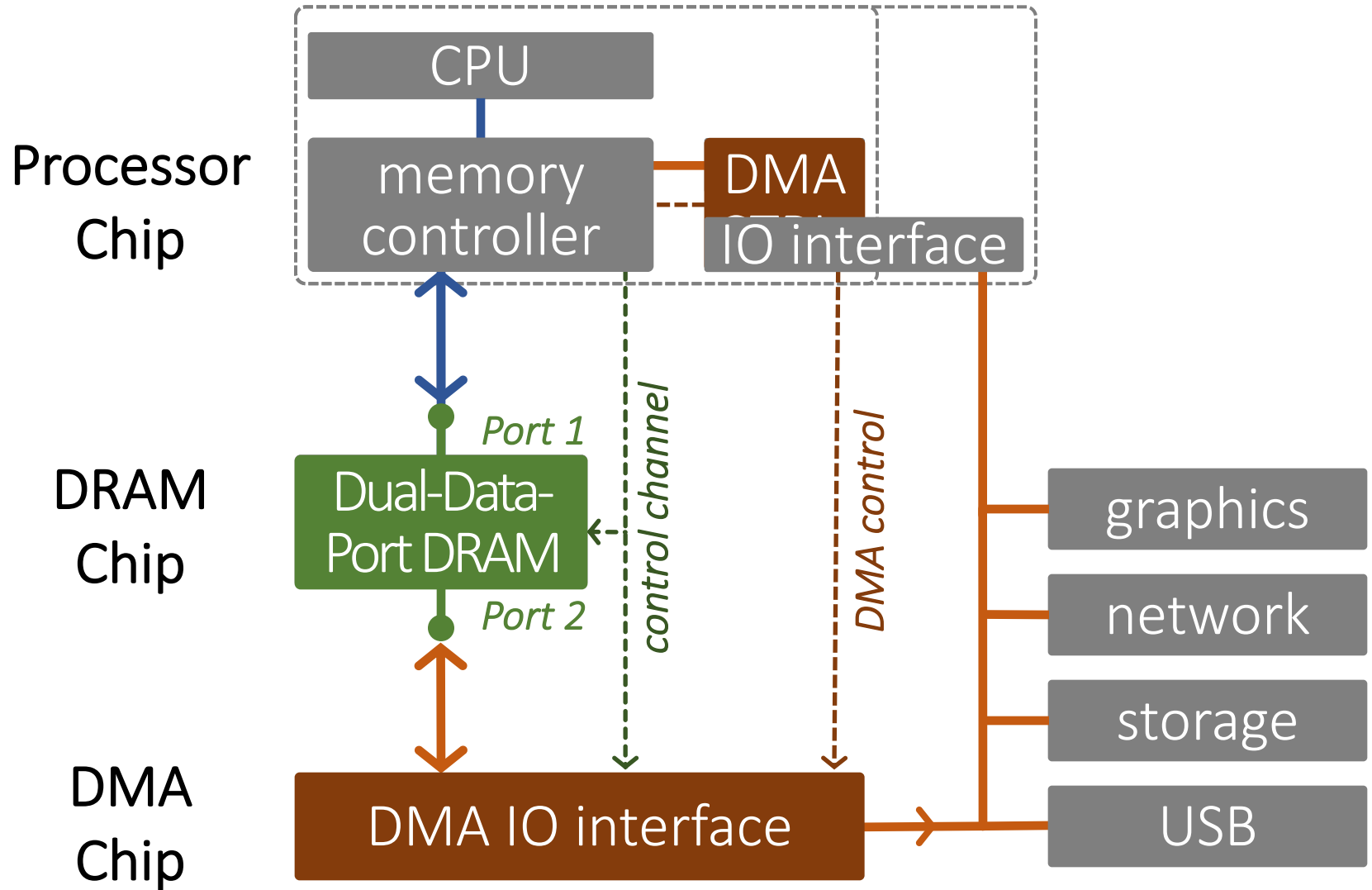
2. Our Approach

3. Dual-Data-Port DRAM

4. Applications for DDMA

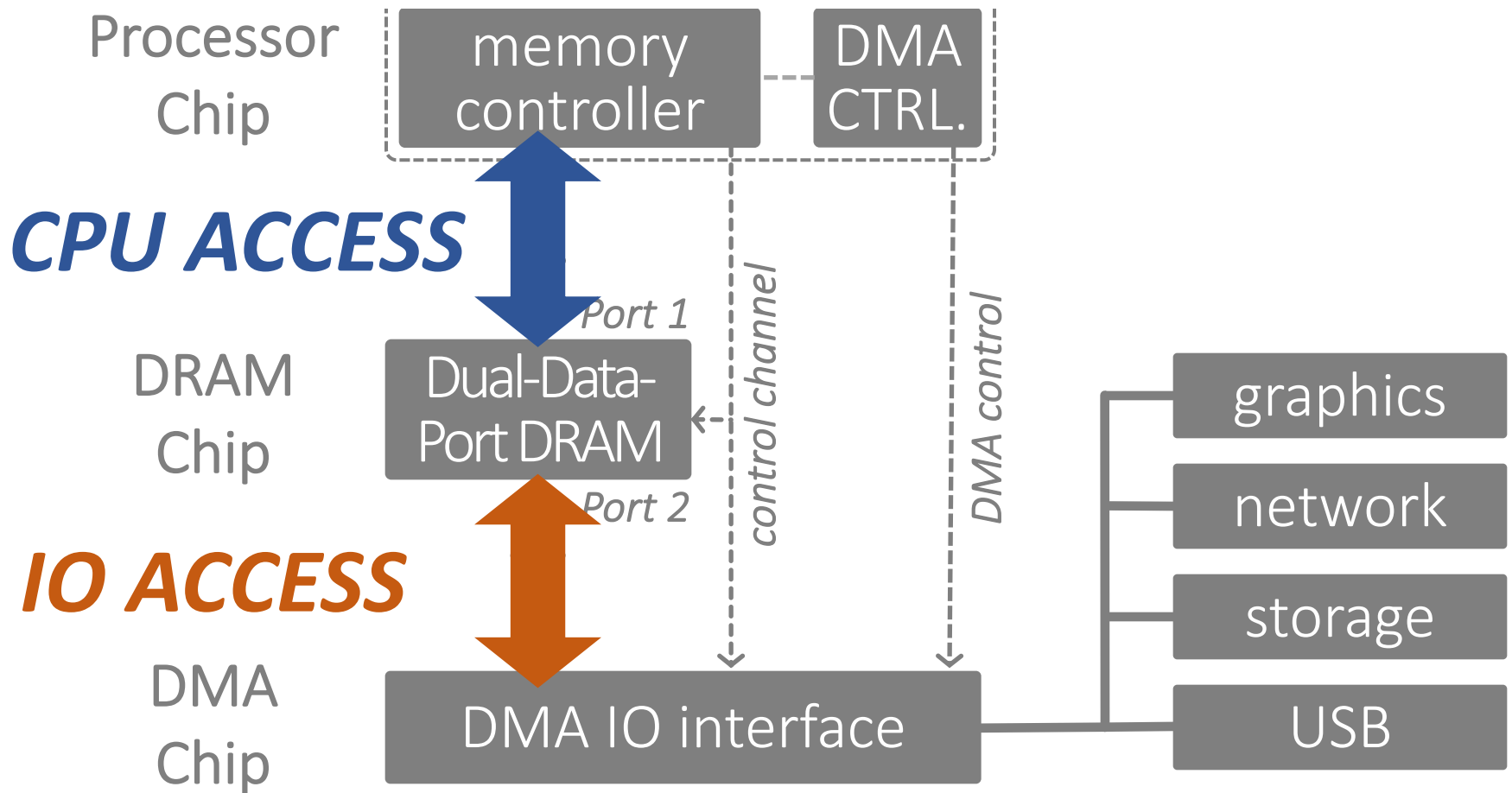
5. Evaluation

Our Approach



Our Approach

Decoupled Direct Memory Access



Outline

1. Problem

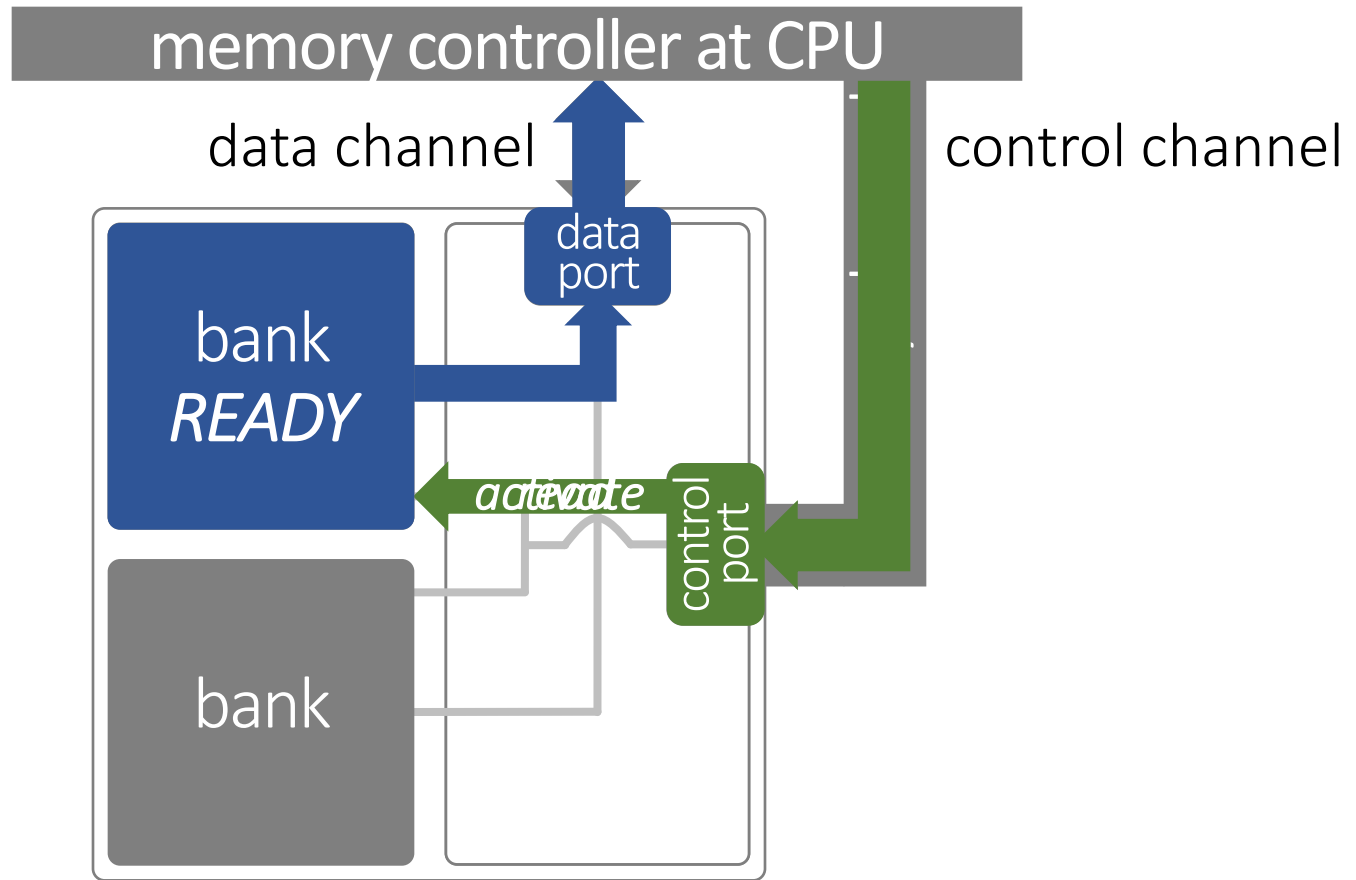
2. Our Approach

3. Dual-Data-Port DRAM

4. Applications for DDMA

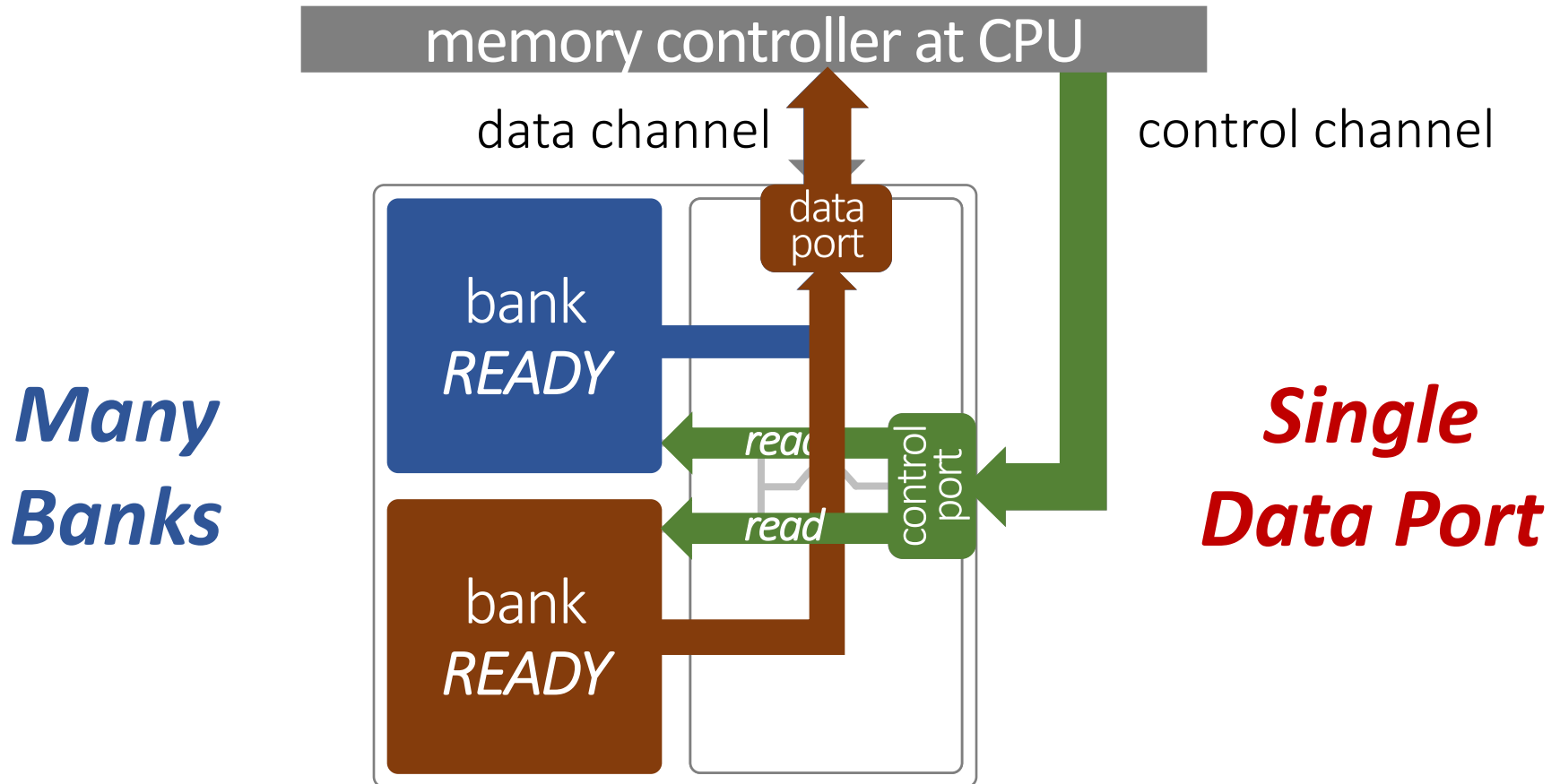
5. Evaluation

Background: DRAM Operation



DRAM peripheral logic: *i) controls banks*, and *ii) transfers data* over memory channel

Problem: Single Data Port

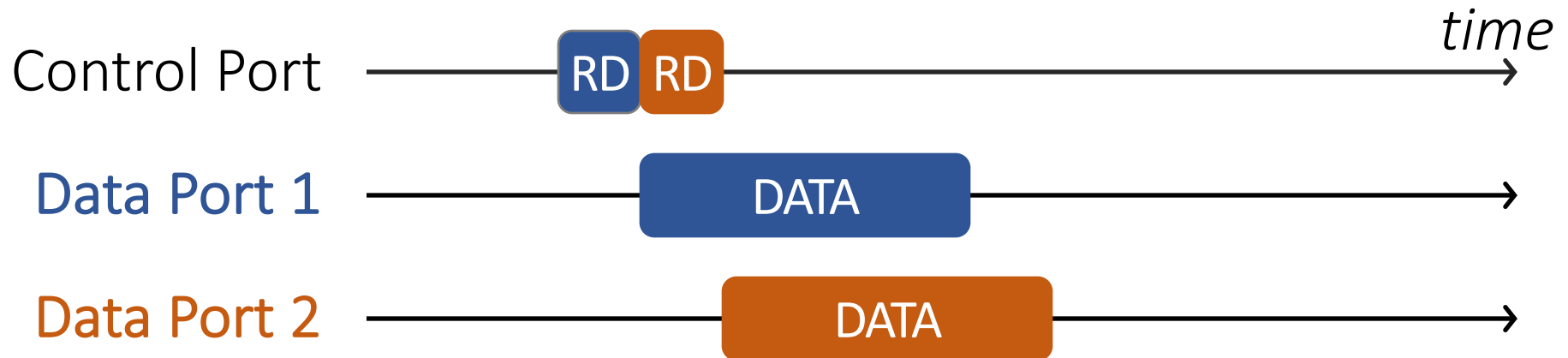


Requests are served *serially*
due to *single data port*

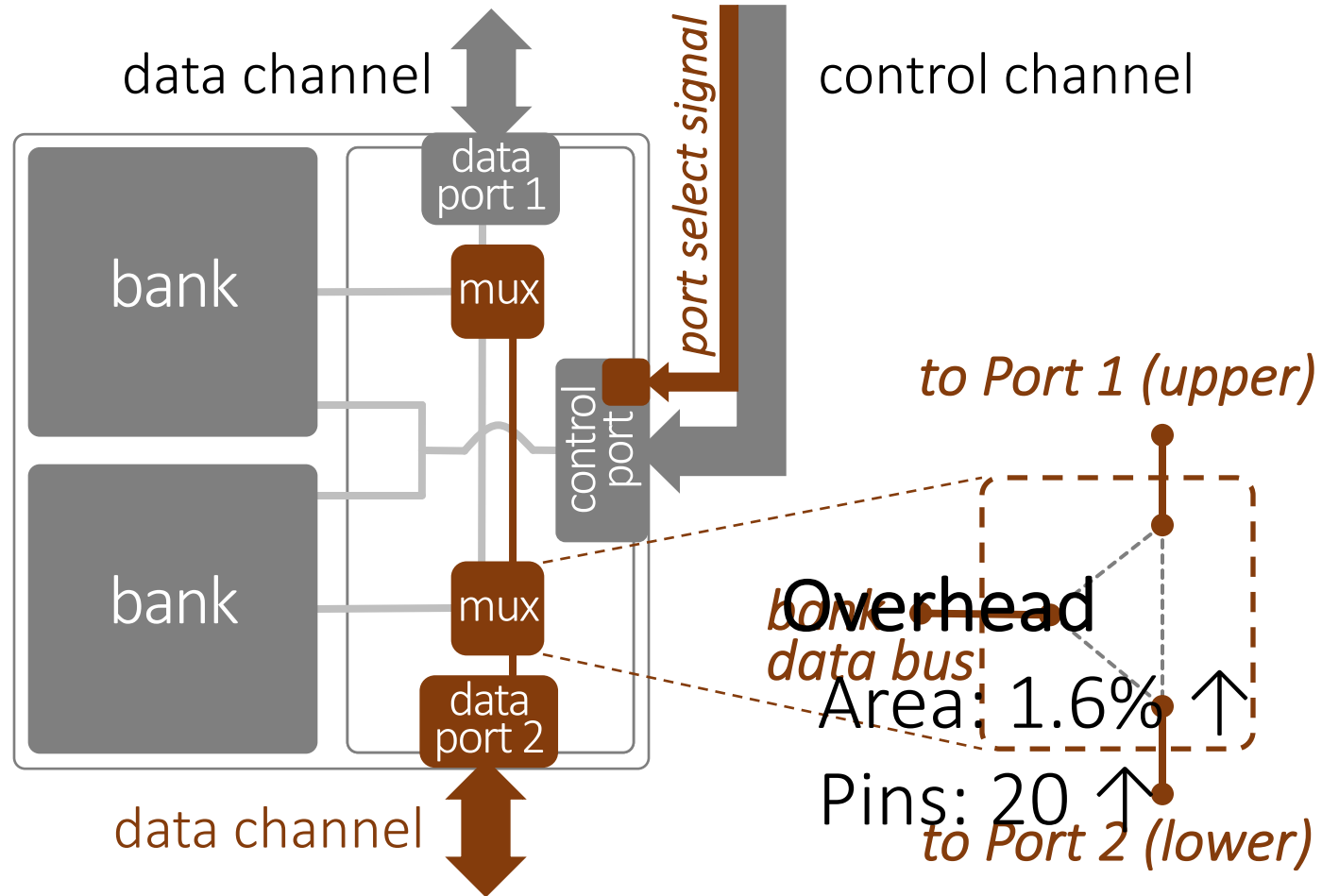
Problem: Single Data Port



What about a DRAM with **two data ports**?

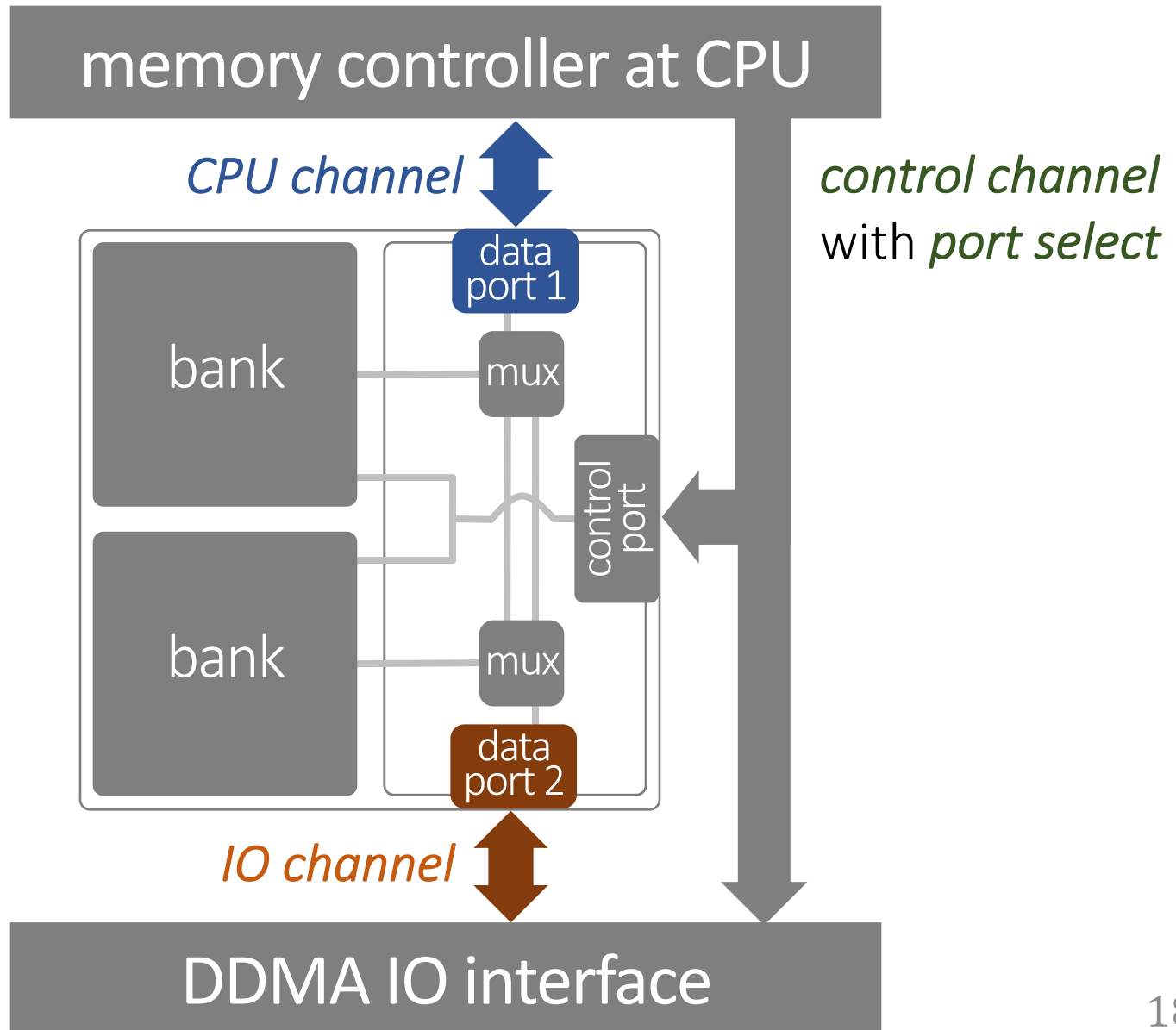


Dual-Data-Port DRAM



*twice the bandwidth & independent data ports
with low overhead*

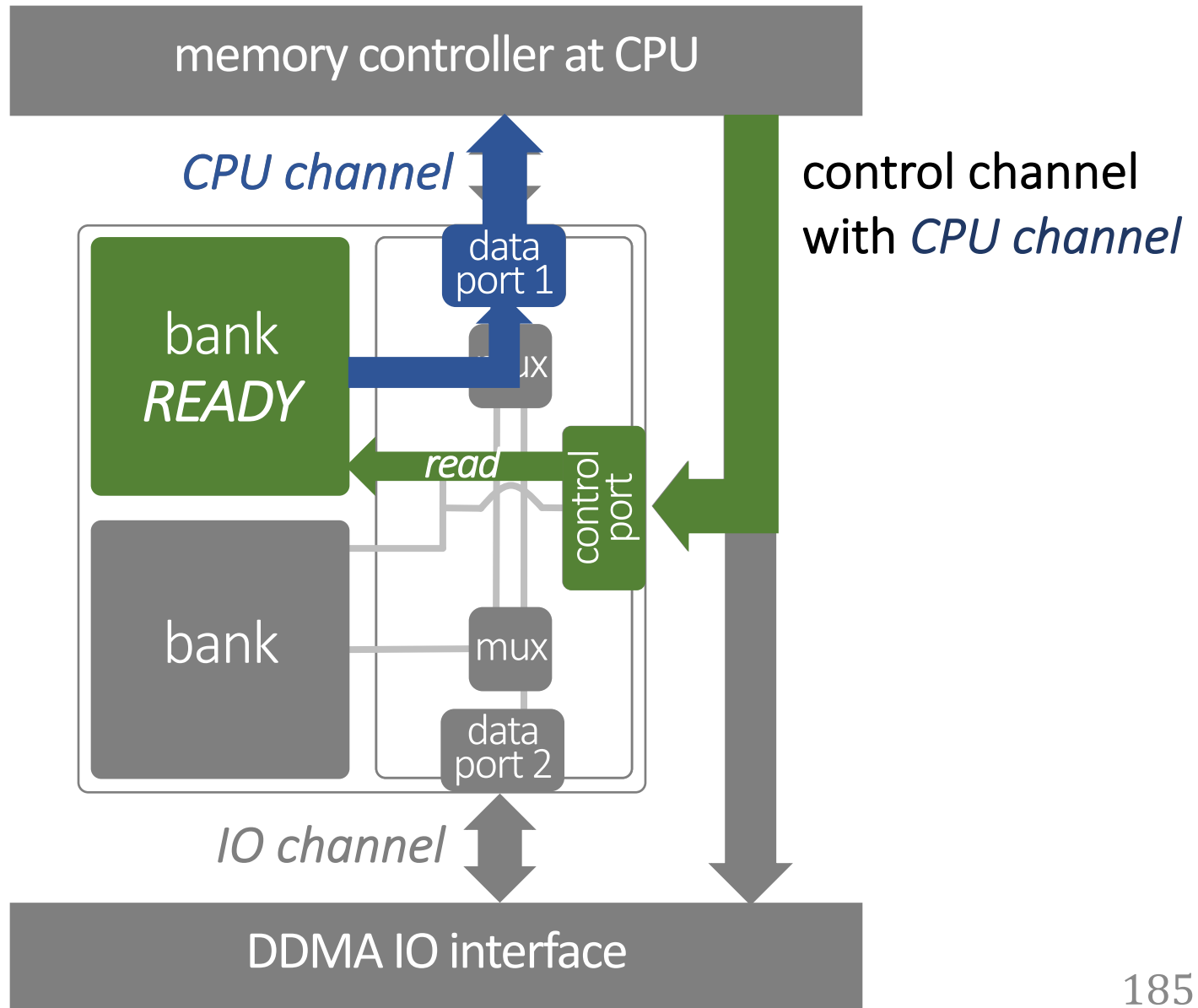
DDP-DRAM Memory System



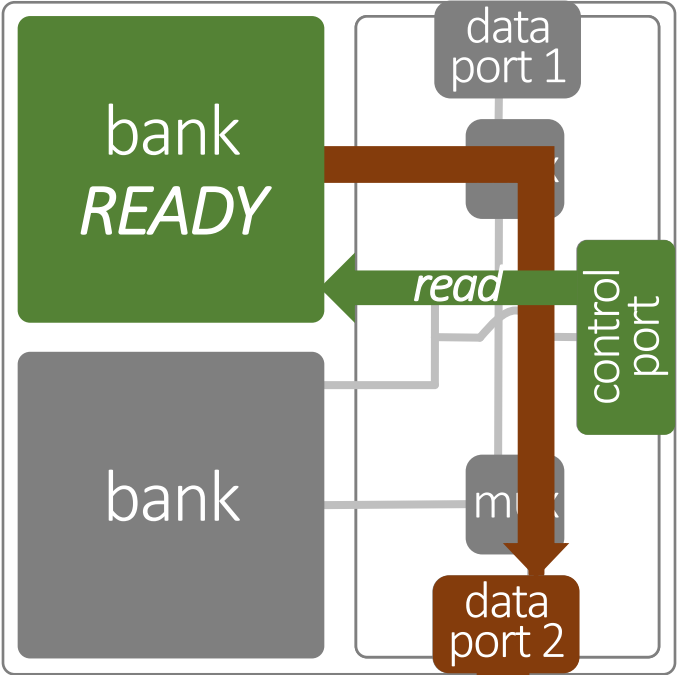
Three Data Transfer Modes

- **CPU Access:** Access through CPU channel
 - DRAM read/write with CPU port selection
- **IO Access:** Access through IO channel
 - DRAM read/write with IO port selection
- **Port Bypass:** Direct transfer between channels
 - DRAM access with port bypass selection

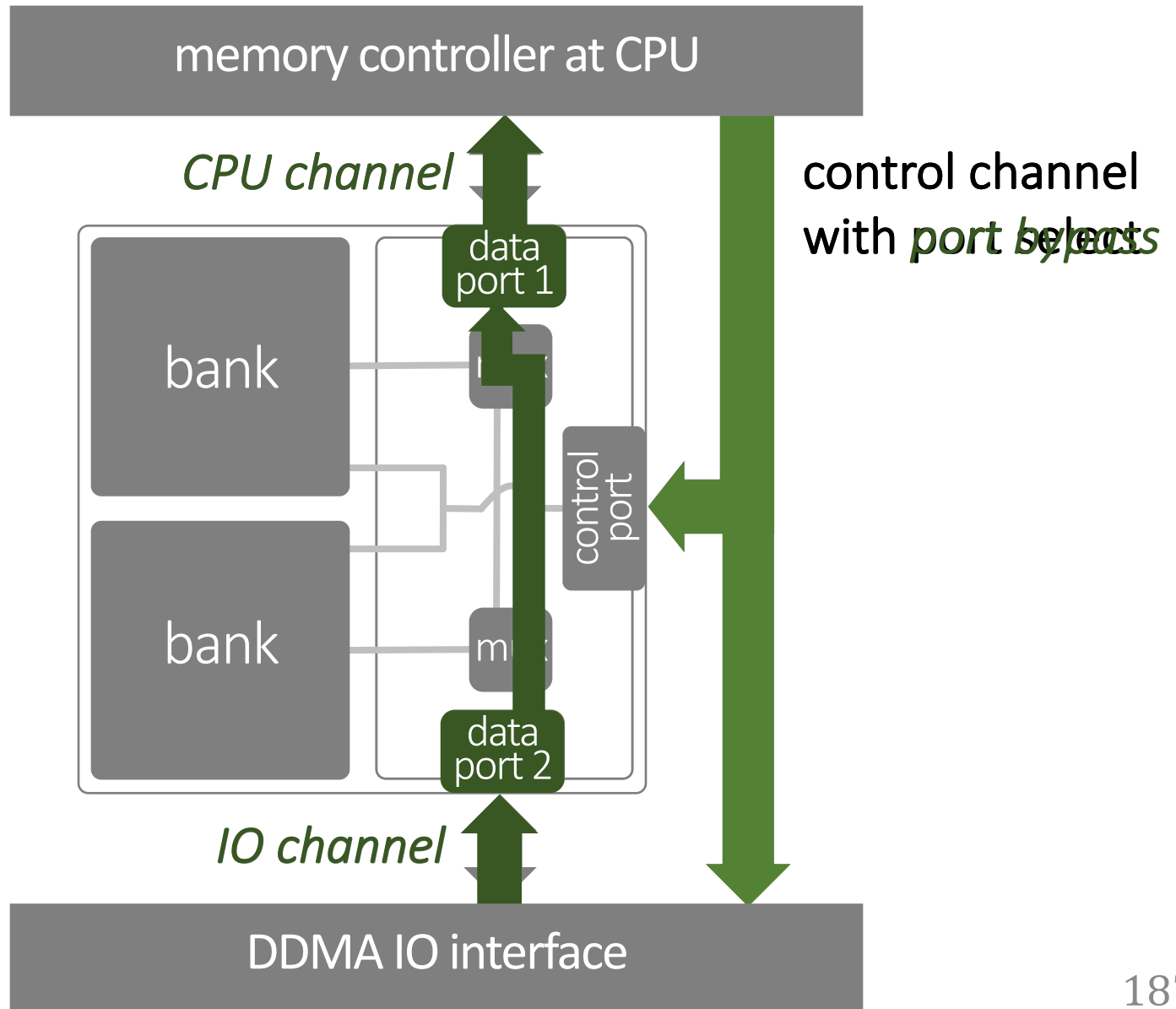
1. CPU Access Mode



memory controller at CPU



3. Port Bypass Mode



Outline

1. Problem

2. Our Approach

3. Dual-Data-Port DRAM

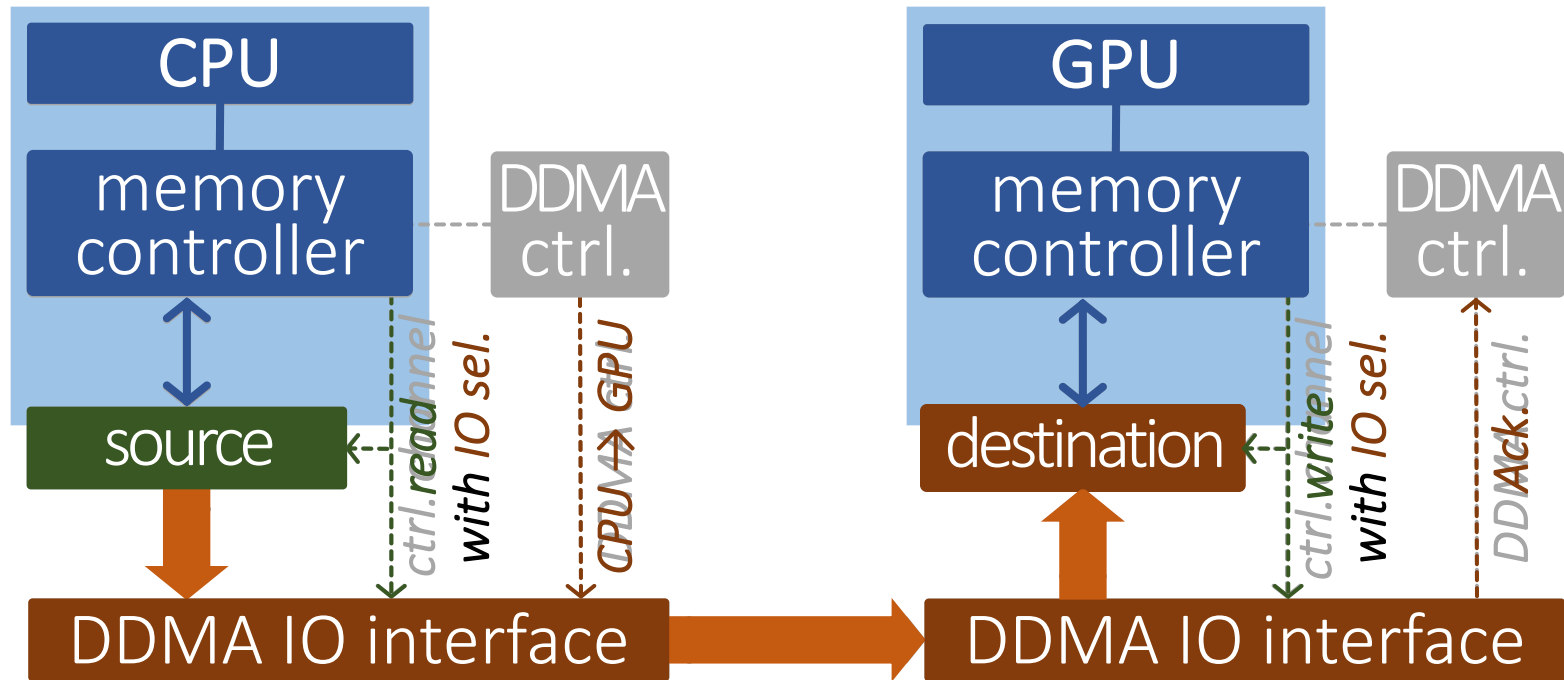
4. Applications for DDMA

5. Evaluation

Three Applications for DDMA

- Communication b/w Compute Units
 - CPU-GPU communication
- In-Memory Communication and Initialization
 - Bulk page copy/initialization
- Communication b/w Memory and Storage
 - Serving page fault/file read & write

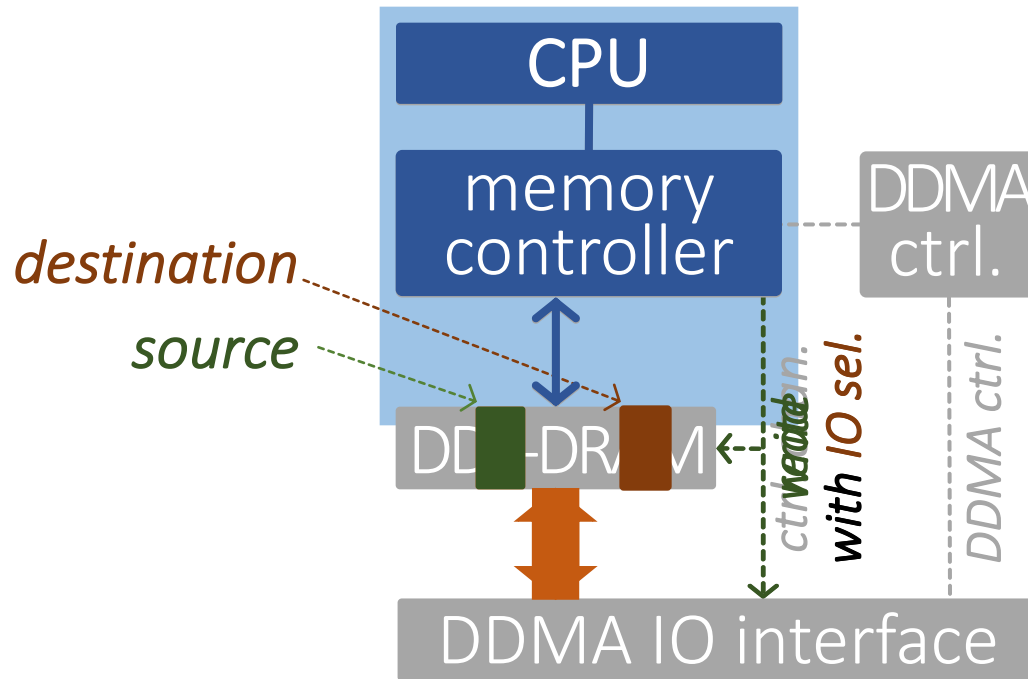
1. Compute Unit \leftrightarrow Compute Unit



Transfer data through DDMA

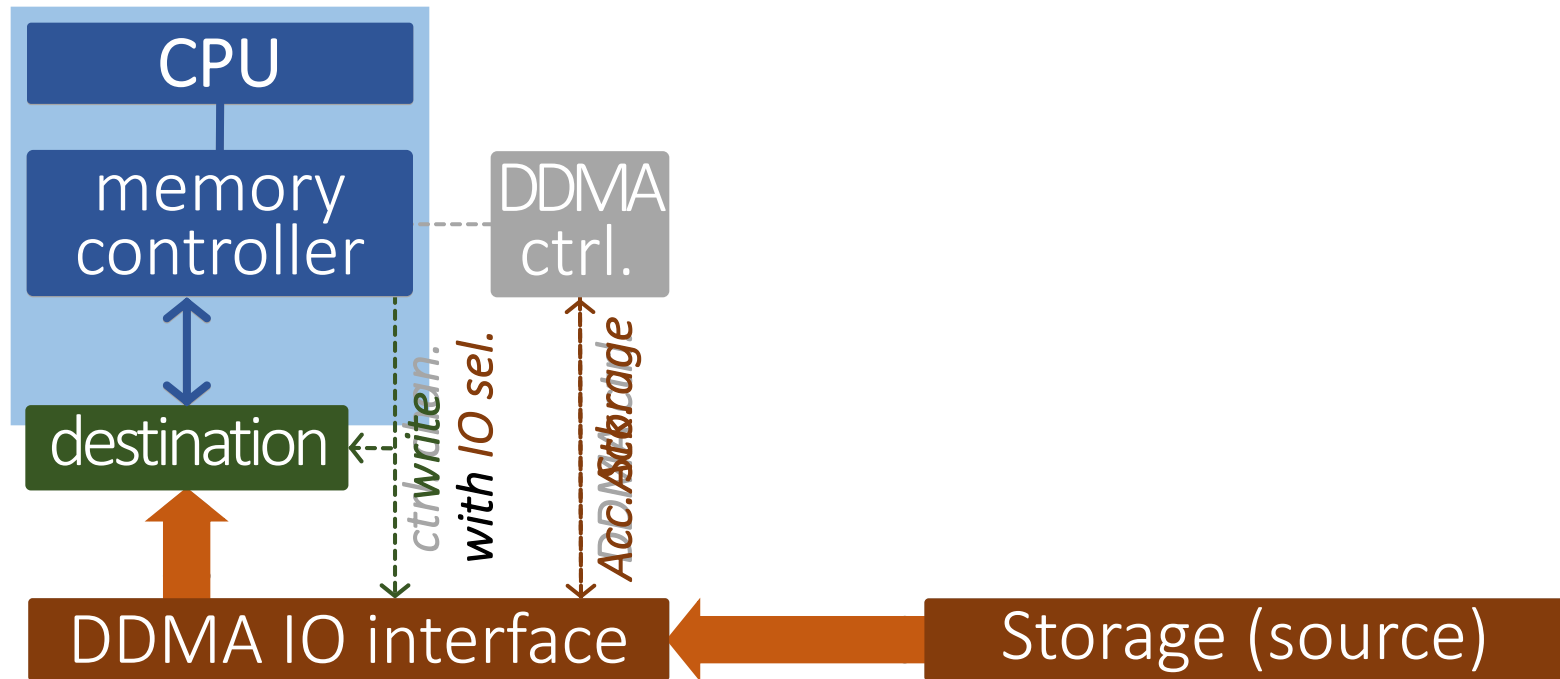
without interfering w/ CPU/GPU memory accesses

2. In-Memory Communication



Transfer data in DRAM through DDAM
without interfering with CPU memory accesses

3. Memory ↔ Storage



Transfer data from storage through DDMA
without interfering with CPU memory accesses

Outline

1. Problem

2. Our Approach

3. Dual-Data-Port DRAM

4. Applications for DDMA

5. Evaluation

Evaluation Methods

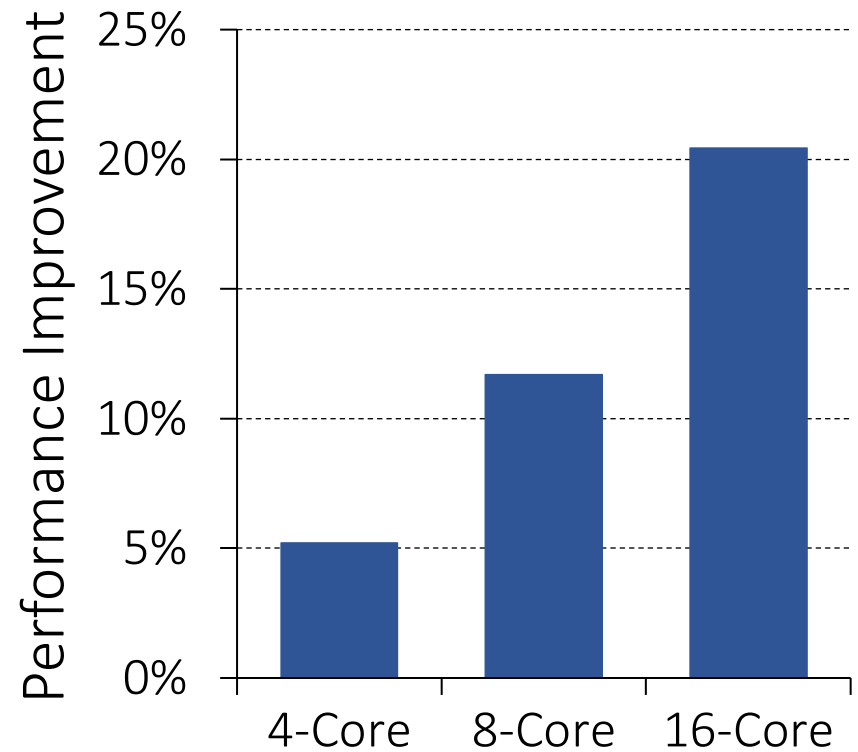
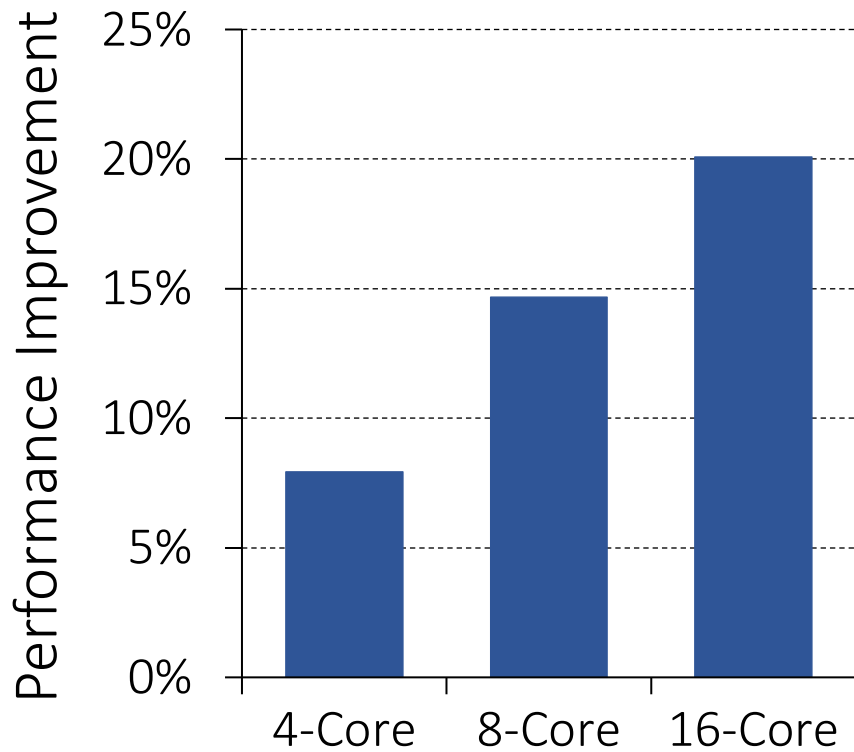
- **System**

- Processor: 4 – 16 cores
- LLC: 16-way associative, 512KB private cache-slice/core
- Memory: 1 – 4 ranks and 1 – 4 channels

- **Workloads**

- **Memory intensive:**
SPEC CPU2006, TPC, stream (31 benchmarks)
- **CPU-GPU communication intensive:**
polybench (8 benchmarks)
- **In-memory communication intensive:**
apache, bootup, compiler, filecopy, mysql, fork, shell, memcached (8 in total)

Performance (2 Channel, 2 Rank)



CPU-GPU Comm.-Intensive

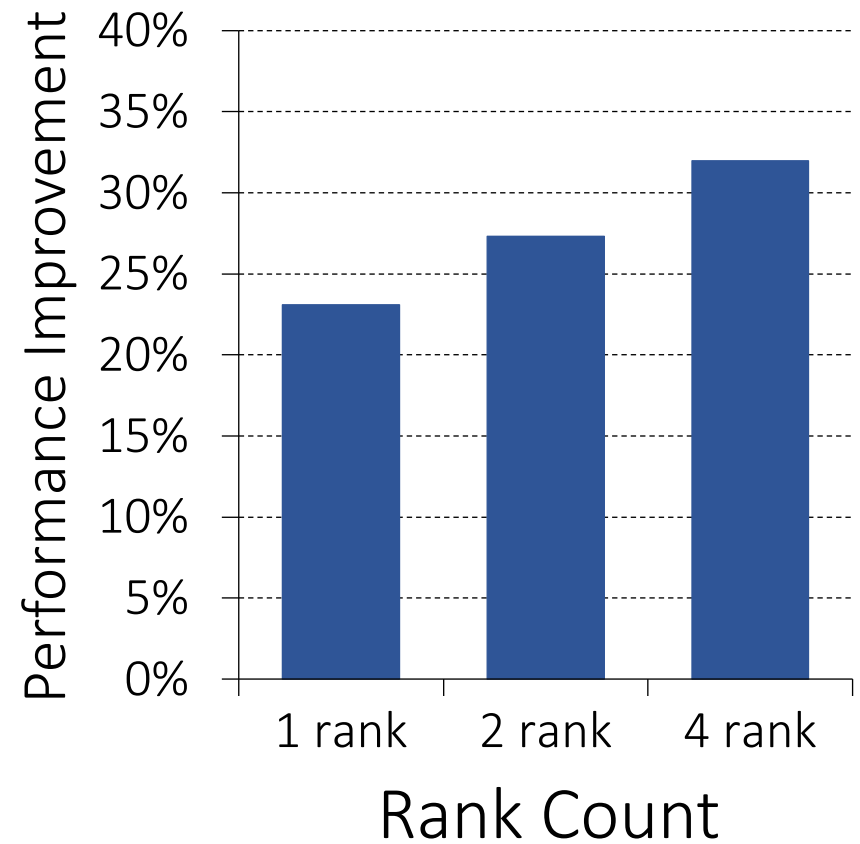
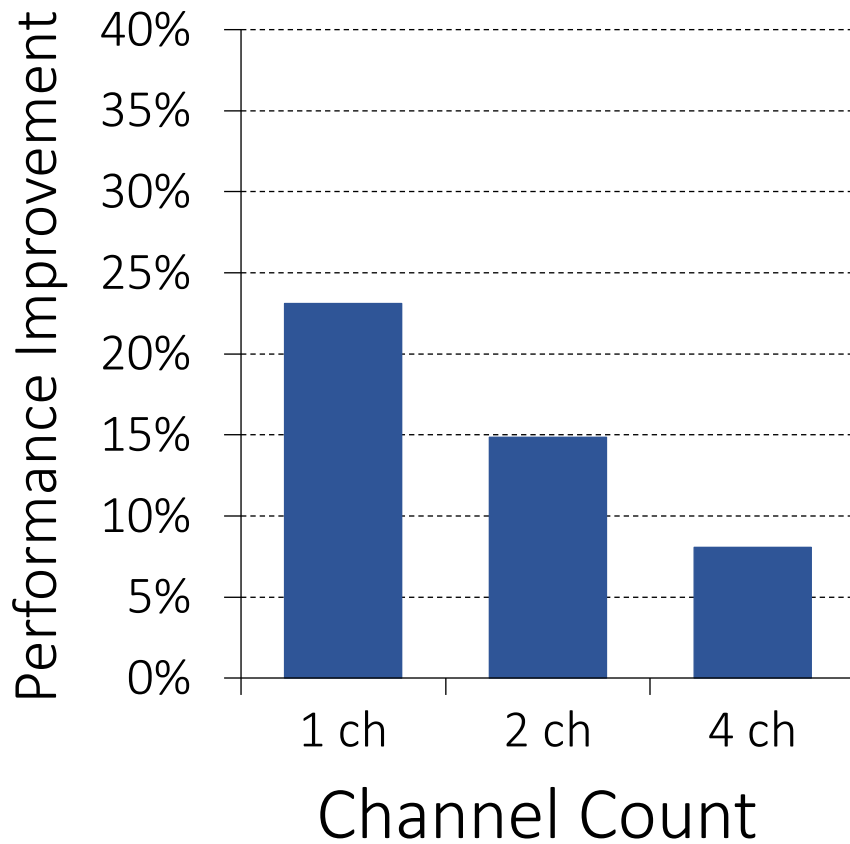
In-Memory Comm.-Intensive

High performance improvement

More performance improvement at *higher core count*

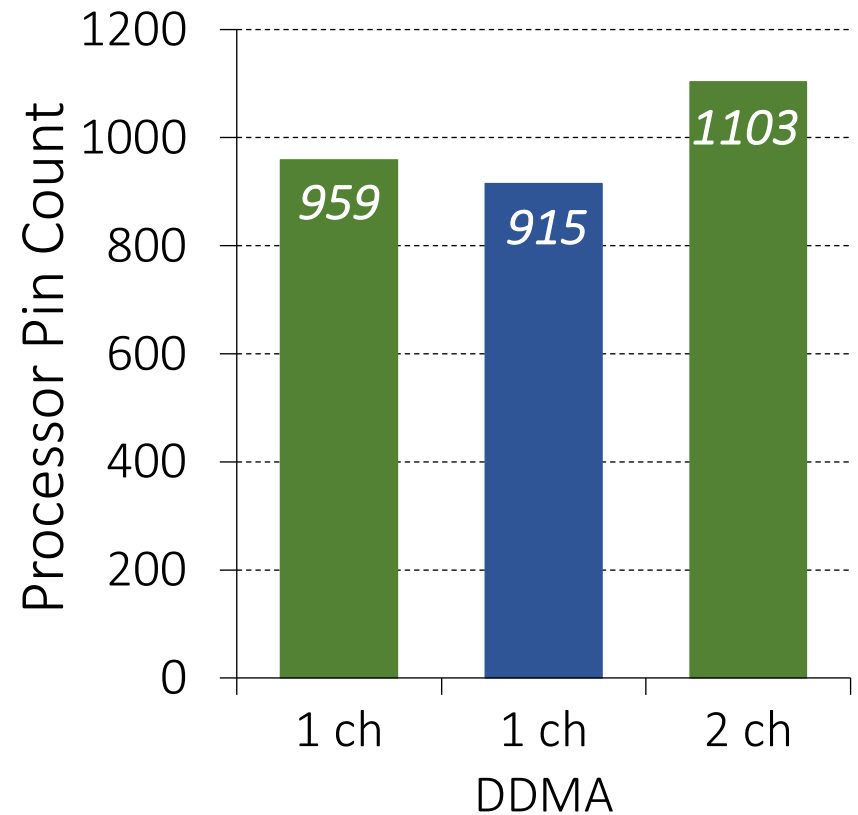
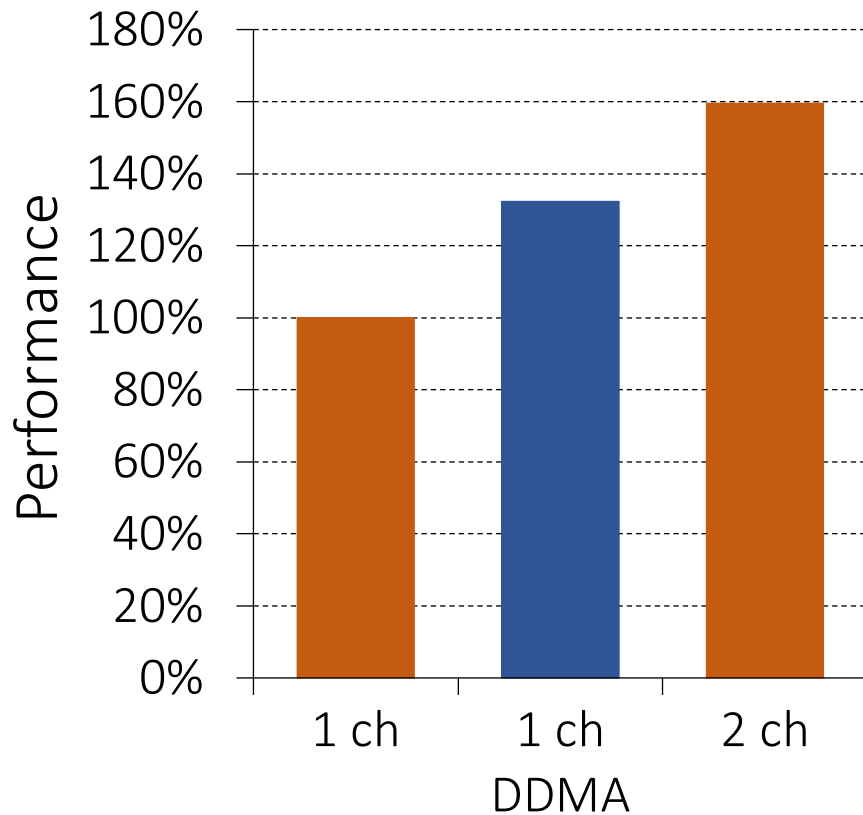
SAFARI

Performance on Various Systems



Performance increases with rank count

DDMA vs. Dual Channel



DDMA achieves *higher performance*
at *lower processor pin count*

More on Decoupled DMA

- Donghyuk Lee, Lavanya Subramanian, Rachata Ausavarungnirun, Jongmoo Choi, and Onur Mutlu,
"Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM"
Proceedings of the 24th International Conference on Parallel Architectures and Compilation Techniques (PACT), San Francisco, CA, USA, October 2015.
[[Slides \(pptx\)](#) ([pdf](#))]

Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM

Donghyuk Lee* Lavanya Subramanian* Rachata Ausavarungnirun* Jongmoo Choi[†] Onur Mutlu*

*Carnegie Mellon University

{donghyu1, lsubrama, rachata, onur}@cmu.edu

[†]Dankook University

choijm@dankook.ac.kr

Predictable Performance Again: Strong Memory Service Guarantees

Remember MISE?

- Lavanya Subramanian, Vivek Seshadri, Yoongu Kim, Ben Jaiyen, and Onur Mutlu,
"MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems"
Proceedings of the 19th International Symposium on High-Performance Computer Architecture (HPCA), Shenzhen, China, February 2013. [Slides \(pptx\)](#)

MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems

Lavanya Subramanian

Vivek Seshadri

Yoongu Kim

Ben Jaiyen

Onur Mutlu

Carnegie Mellon University

Extending Slowdown Estimation to Caches

- How do we extend the MISE model to include shared cache interference?
- Answer: Application Slowdown Model
- Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu,
"The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory"
Proceedings of the 48th International Symposium on Microarchitecture (MICRO), Waikiki, Hawaii, USA, December 2015.
[\[Slides \(pptx\) \(pdf\)\]](#) [\[Lightning Session Slides \(pptx\) \(pdf\)\]](#) [\[Poster \(pptx\) \(pdf\)\]](#)
[\[Source Code\]](#)

Application Slowdown Model

Quantifying and Controlling Impact of Interference at Shared Caches and Main Memory

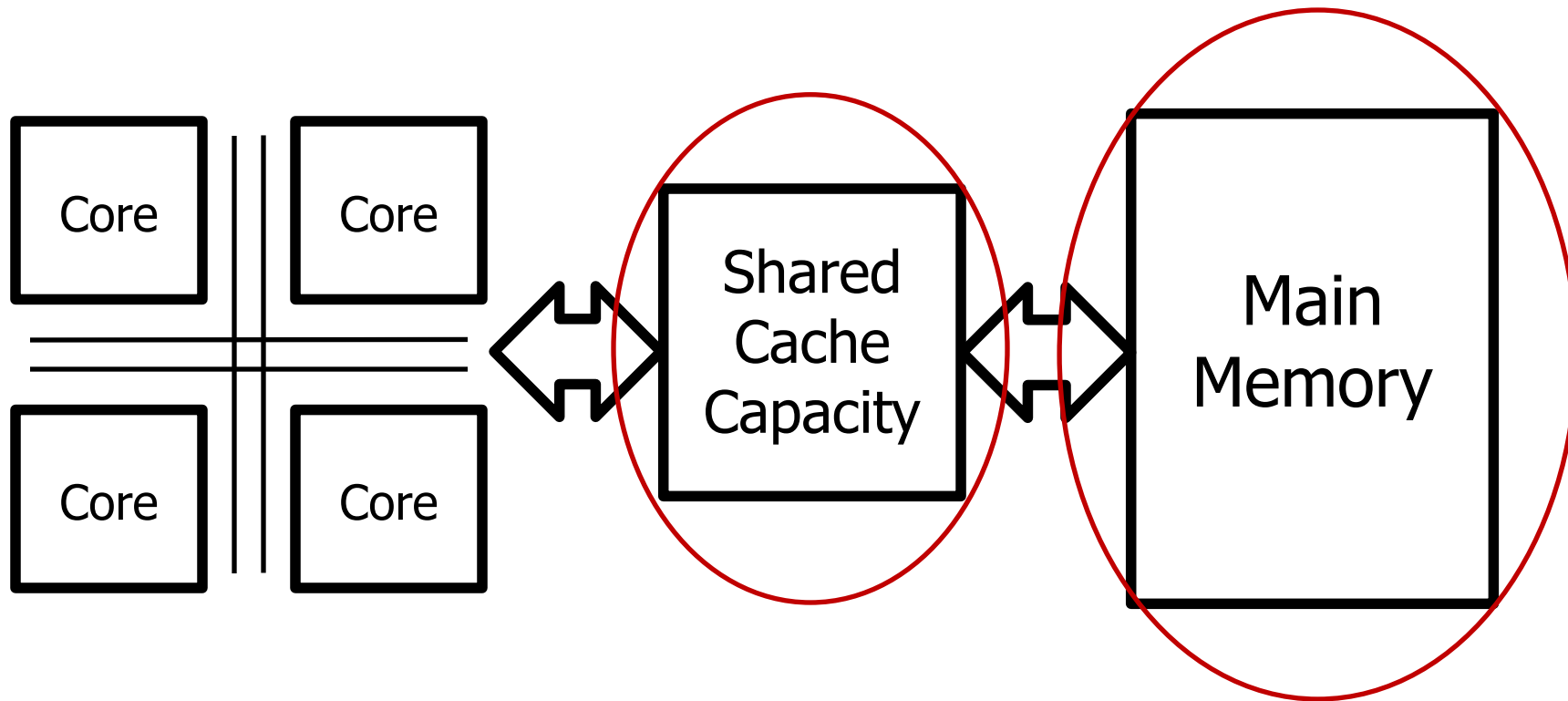
Lavanya Subramanian, Vivek Seshadri,
Arnab Ghosh, Samira Khan, Onur Mutlu

SAFARI

Carnegie Mellon



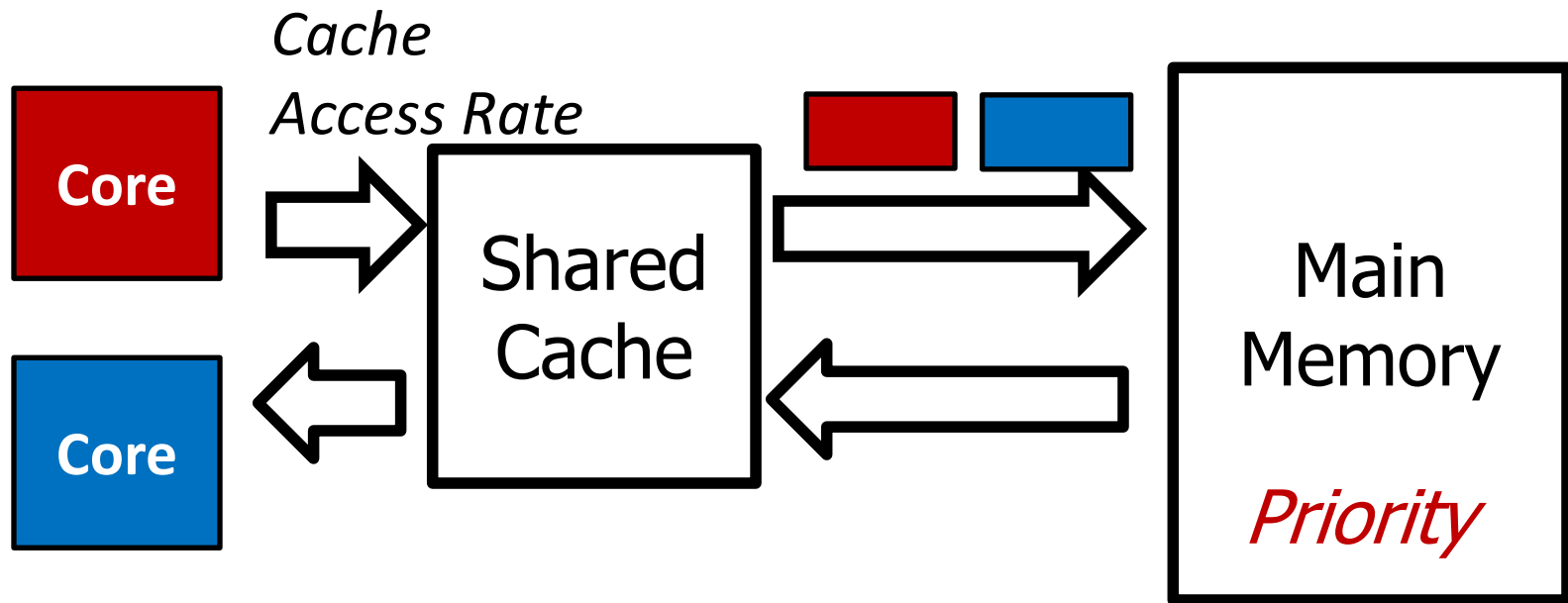
Shared Cache and Memory Contention



$$\text{Slowdown} = \frac{\text{Request Service Rate}_{\text{Alone}}}{\text{Request Service Rate}_{\text{Shared}}}$$

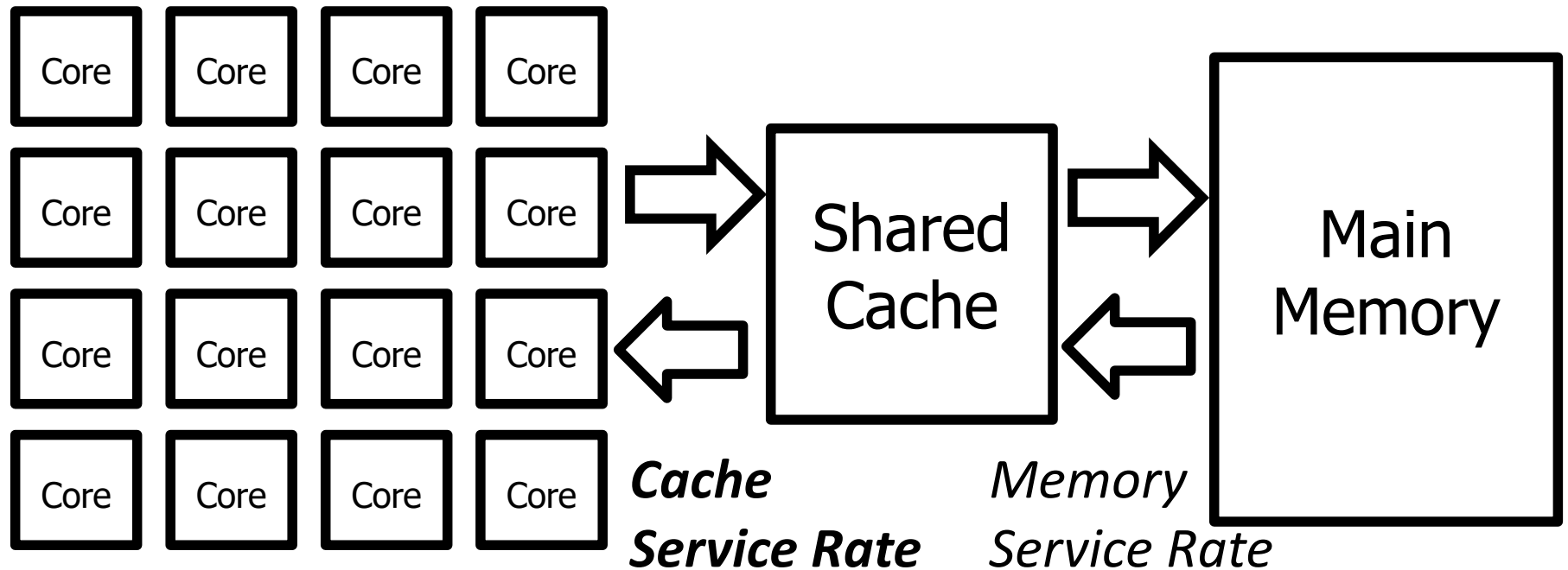
MISE [HPCA'13]

Cache Capacity Contention

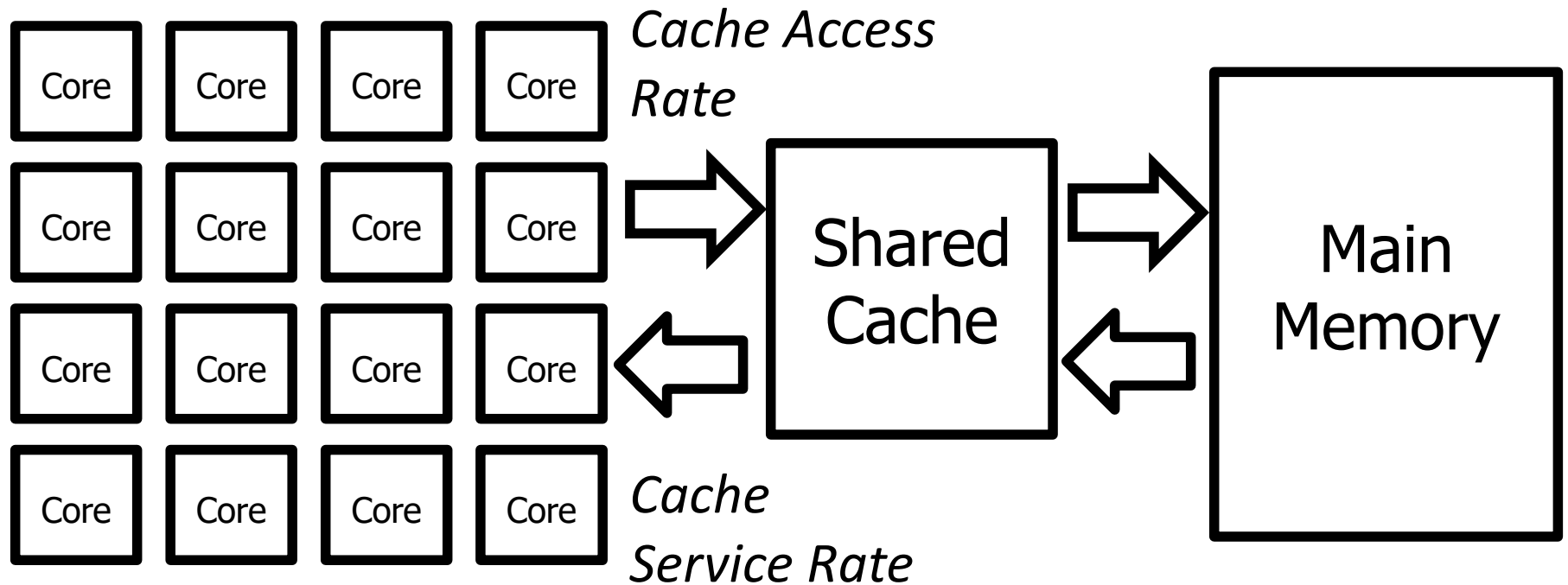


Applications evict each other's blocks from the shared cache

Estimating Cache and Memory Slowdowns

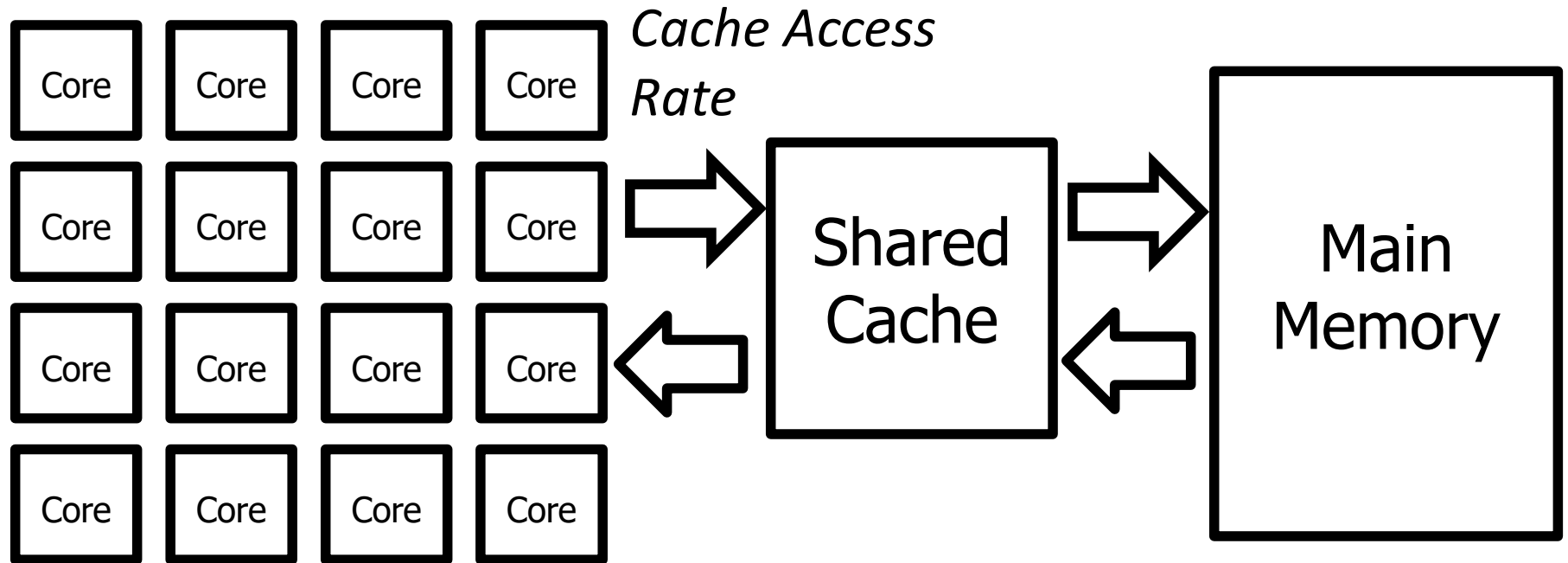


Service Rates vs. Access Rates



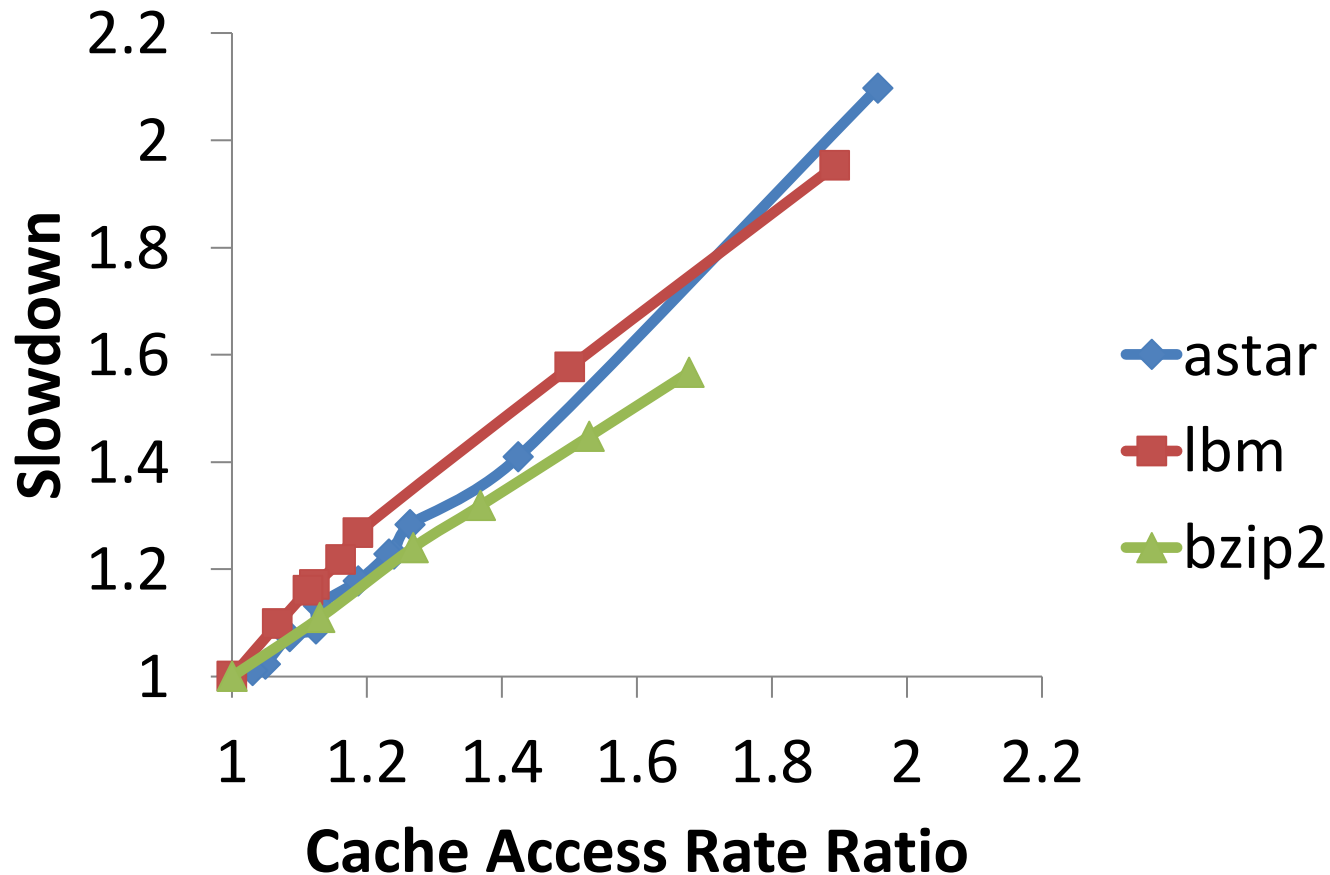
**Request service and access rates
are tightly coupled**

The Application Slowdown Model



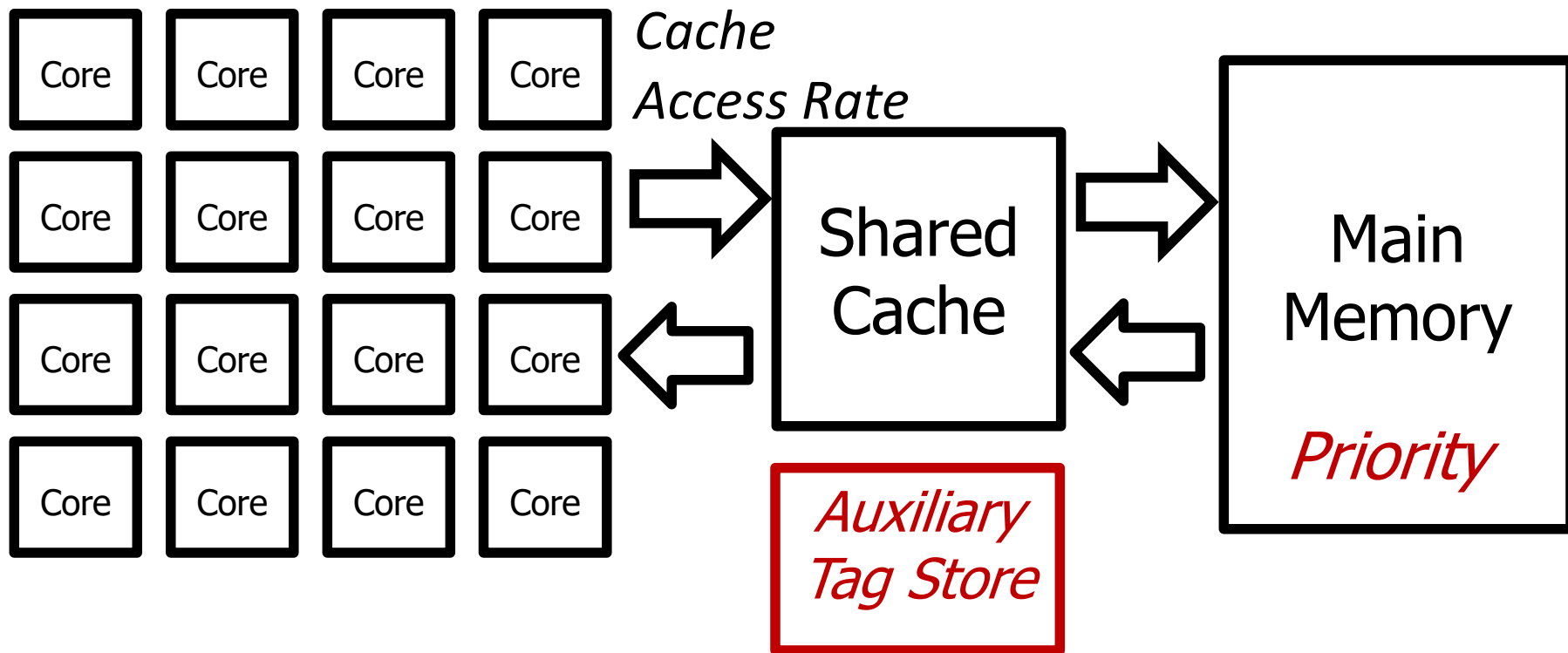
$$\text{Slowdown} = \frac{\text{Cache Access Rate}_{\text{Alone}}}{\text{Cache Access Rate}_{\text{Shared}}}$$

Real System Studies: Cache Access Rate vs. Slowdown

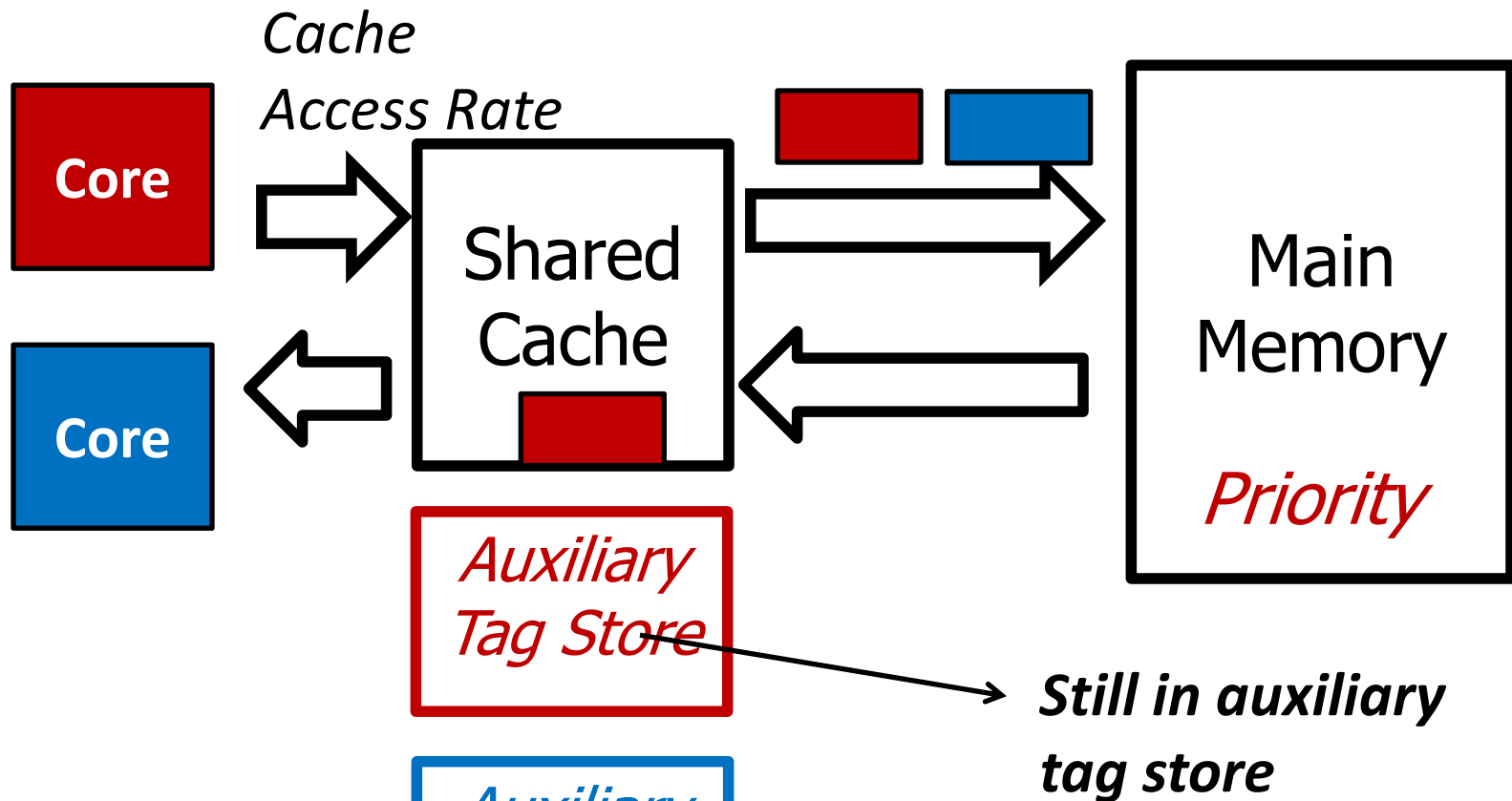


Challenge

How to estimate alone cache access rate?



Auxiliary Tag Store



Auxiliary tag store tracks such ***contention misses***

Accounting for Contention Misses

- Revisiting alone memory request service rate

$$\text{Alone Request Service Rate of an Application} = \frac{\text{\# Requests During High Priority Epochs}}{\text{\# High Priority Cycles}}$$

Cycles serving contention misses should not count as high priority cycles

Alone Cache Access Rate Estimation

$$\text{Cache Access Rate}_{\text{Alone of an Application}} = \frac{\# \text{ Requests During High Priority Epochs}}{\# \text{ High Priority Cycles} - \# \text{ Cache Contention Cycles}}$$

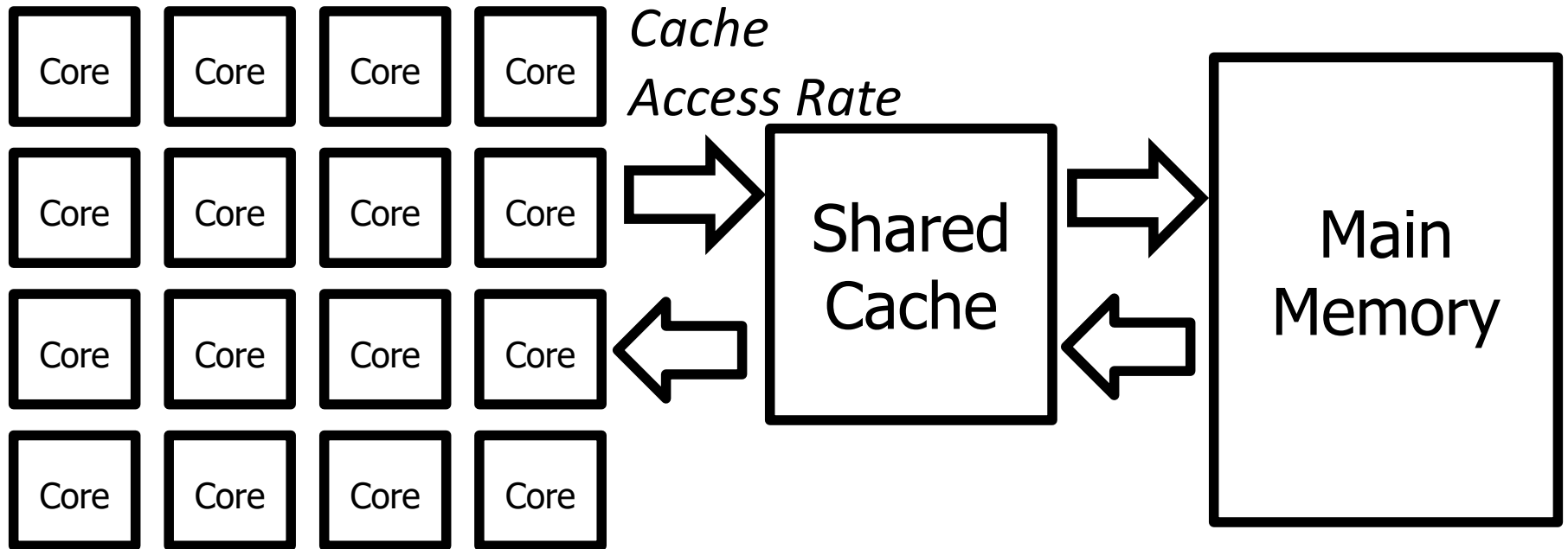
Cache Contention Cycles: Cycles spent serving contention misses

$$\text{Cache Contention Cycles} = \# \text{ Contention Misses} \times \text{Average Memory Service Time}$$

*From auxiliary tag store
when given high priority*

*Measured when given
high priority*

Application Slowdown Model (ASM)



$$\text{Slowdown} = \frac{\text{Cache Access Rate}_{\text{Alone}}}{\text{Cache Access Rate}_{\text{Shared}}}$$

Previous Work on Slowdown Estimation

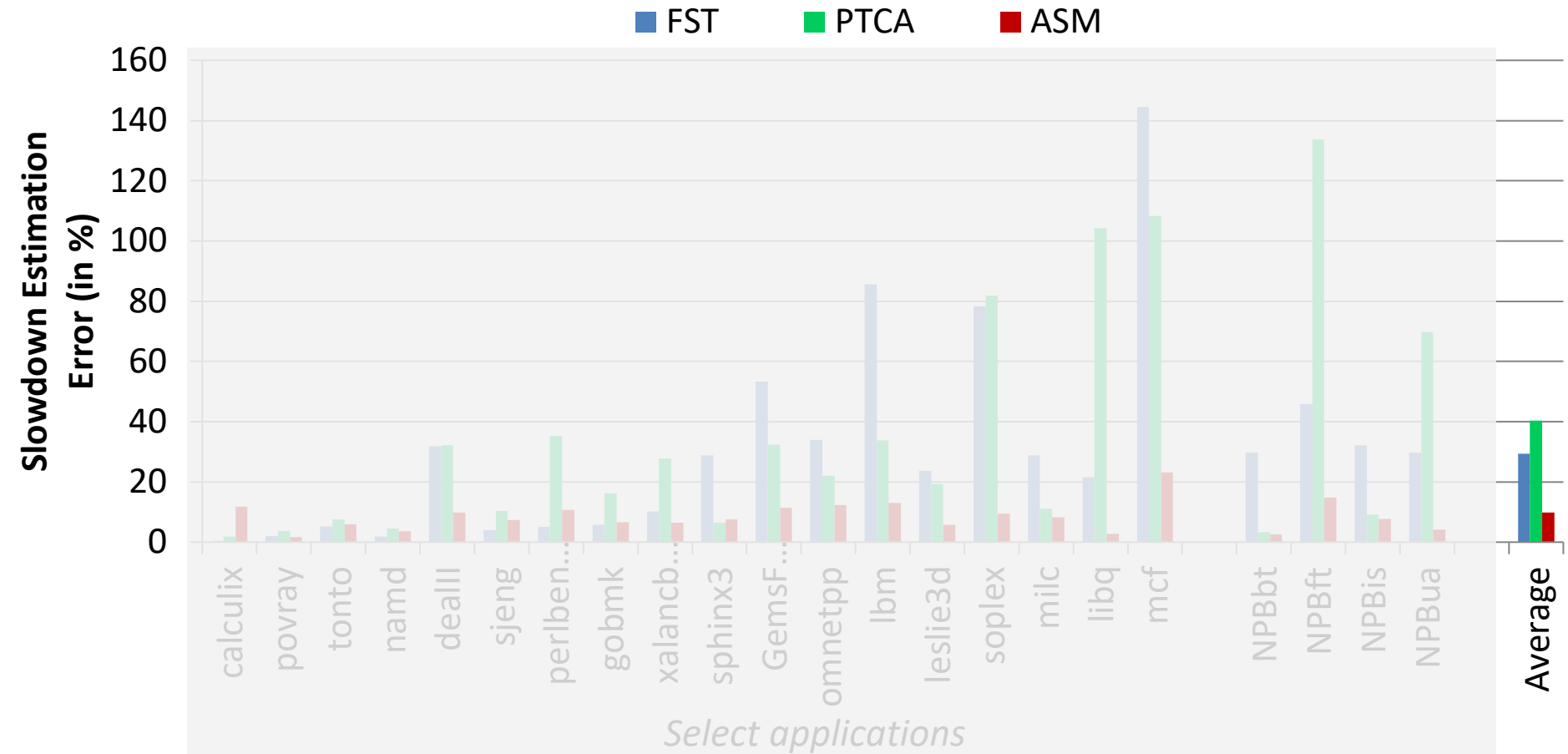
- Previous work on slowdown estimation
 - **STFM** (Stall Time Fair Memory) Scheduling [Mutlu et al., MICRO '07]
 - **FST** (Fairness via Source Throttling) [Ebrahimi et al., ASPLOS '10]
 - **Per-thread Cycle Accounting** [Du Bois et al., HiPEAC '13]

- Basic Idea:

$$\text{Slowdown} = \frac{\text{Execution Time}_{\text{Alone}}}{\text{Execution Time}_{\text{Shared}}}$$

Count interference experienced by each request → Difficult
ASM's estimates are much more coarse grained → Easier

Model Accuracy Results

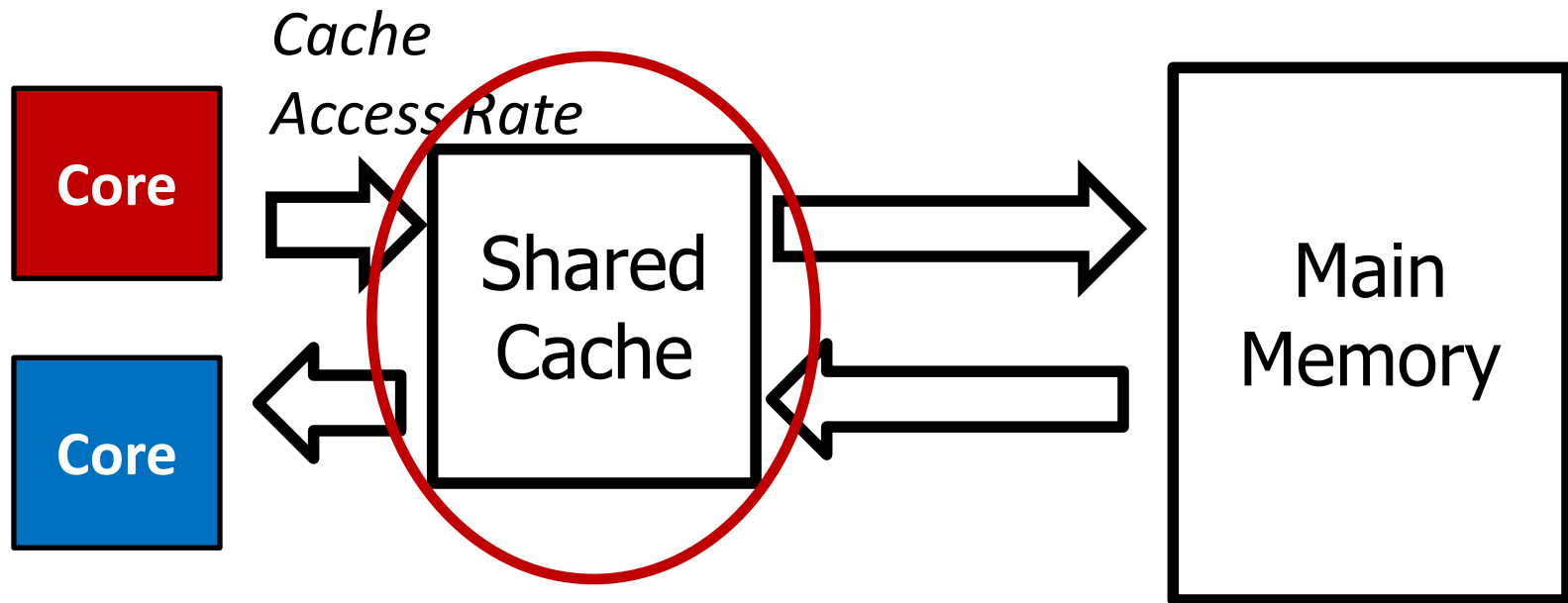


Average error of ASM's slowdown estimates: 10%

Leveraging ASM's Slowdown Estimates

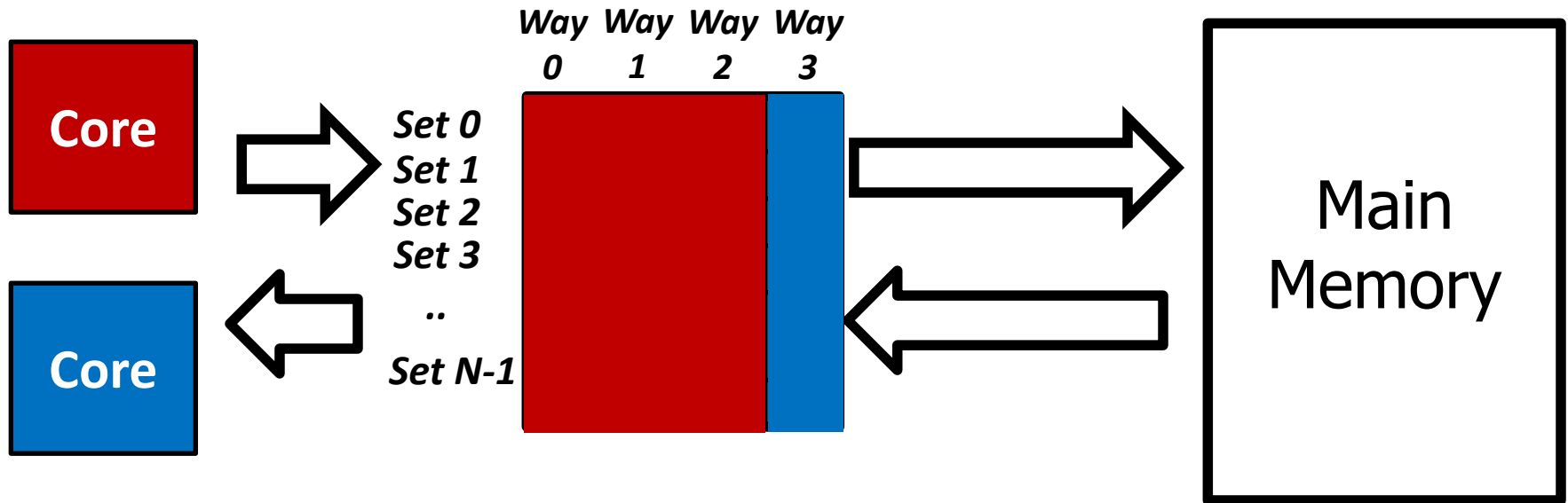
- *Slowdown-aware resource allocation for high performance and fairness*
- *Slowdown-aware resource allocation to bound application slowdowns*
- *VM migration and admission control schemes [VEE '15]*
- *Fair billing schemes in a commodity cloud*

Cache Capacity Partitioning



Goal: Partition the shared cache among applications to mitigate contention

Cache Capacity Partitioning



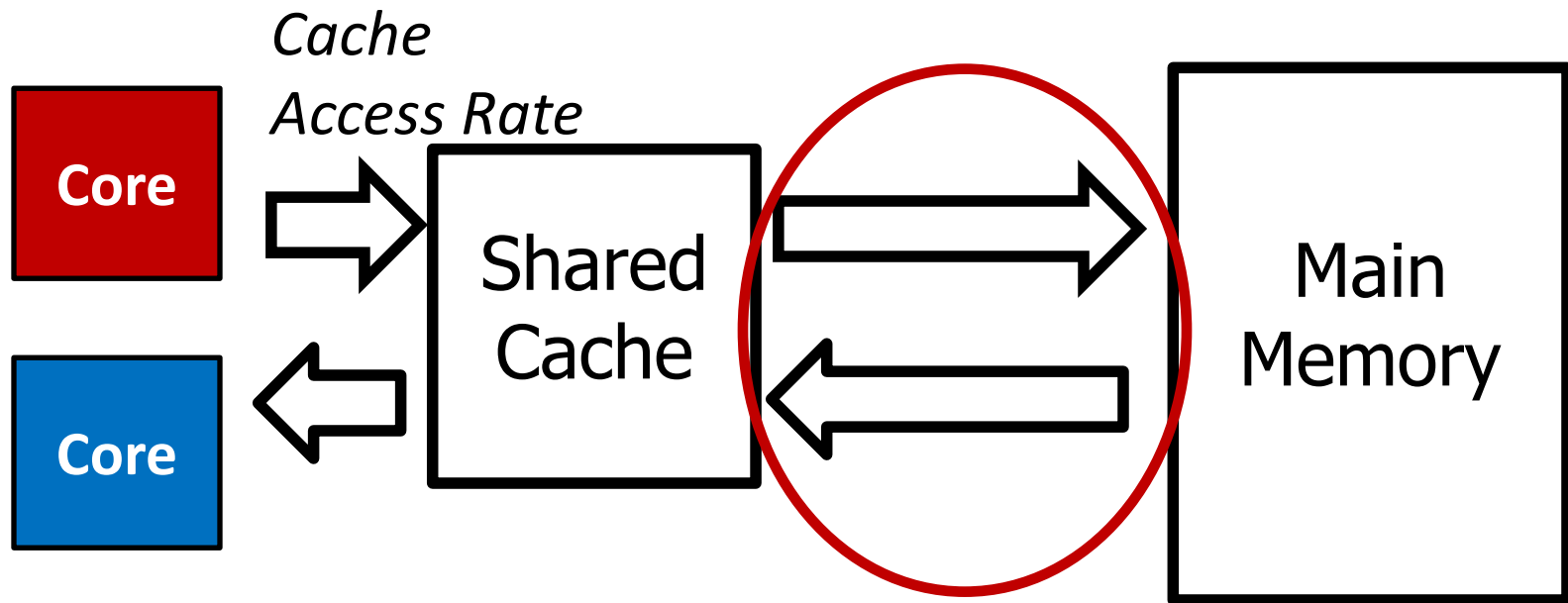
Previous partitioning schemes optimize for miss count

Problem: Not aware of performance and slowdowns

ASM-Cache: Slowdown-aware Cache Way Partitioning

- *Key Requirement: Slowdown estimates for all possible way partitions*
- *Extend ASM to estimate slowdown for all possible cache way allocations*
- *Key Idea: Allocate each way to the application whose slowdown reduces the most*

Memory Bandwidth Partitioning



Goal: Partition the main memory bandwidth among applications to mitigate contention

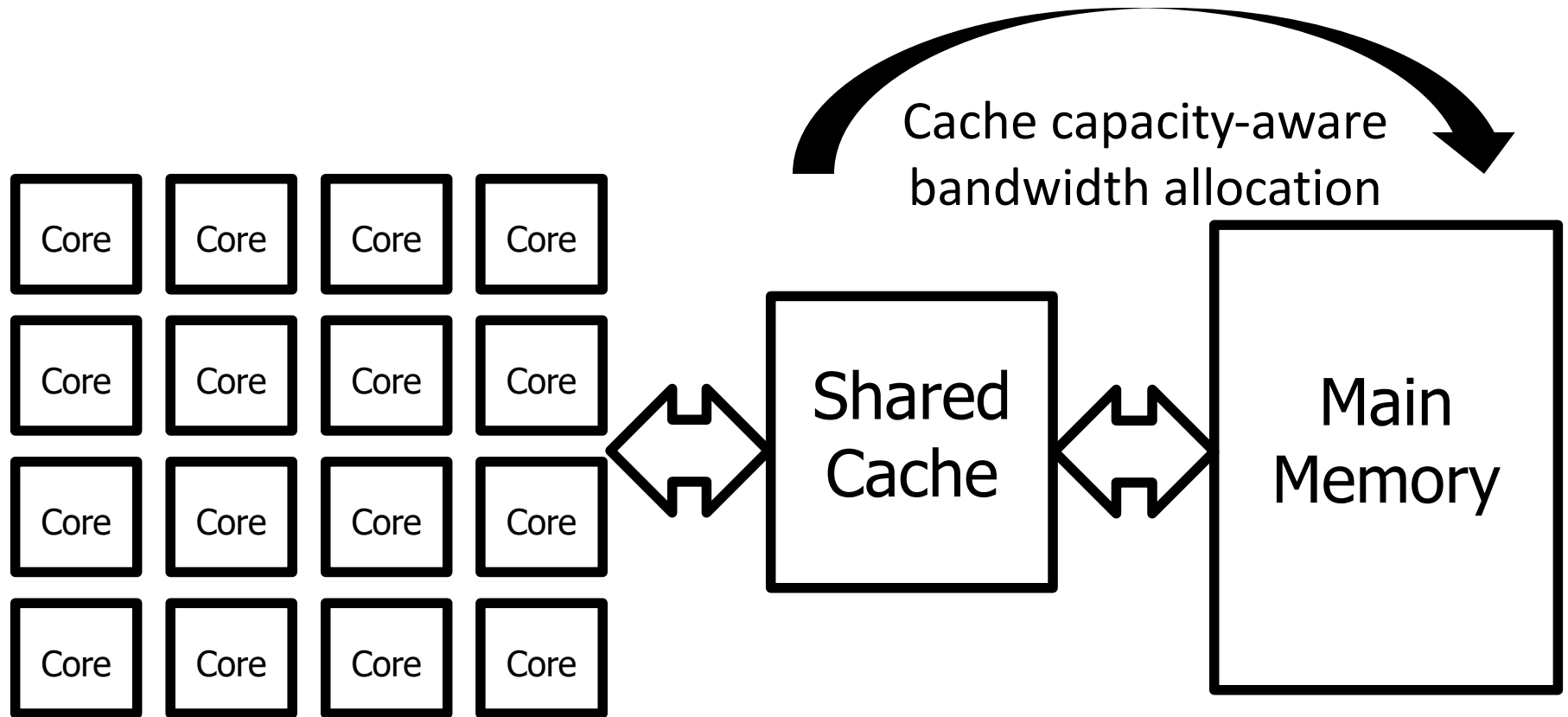
ASM-Mem: Slowdown-aware Memory Bandwidth Partitioning

- *Key Idea: Allocate high priority proportional to an application's slowdown*

$$\text{High Priority Fraction}_i = \frac{\text{Slowdown}_i}{\sum_j \text{Slowdown}_j}$$

- *Application i 's requests given highest priority at the memory controller for its fraction*

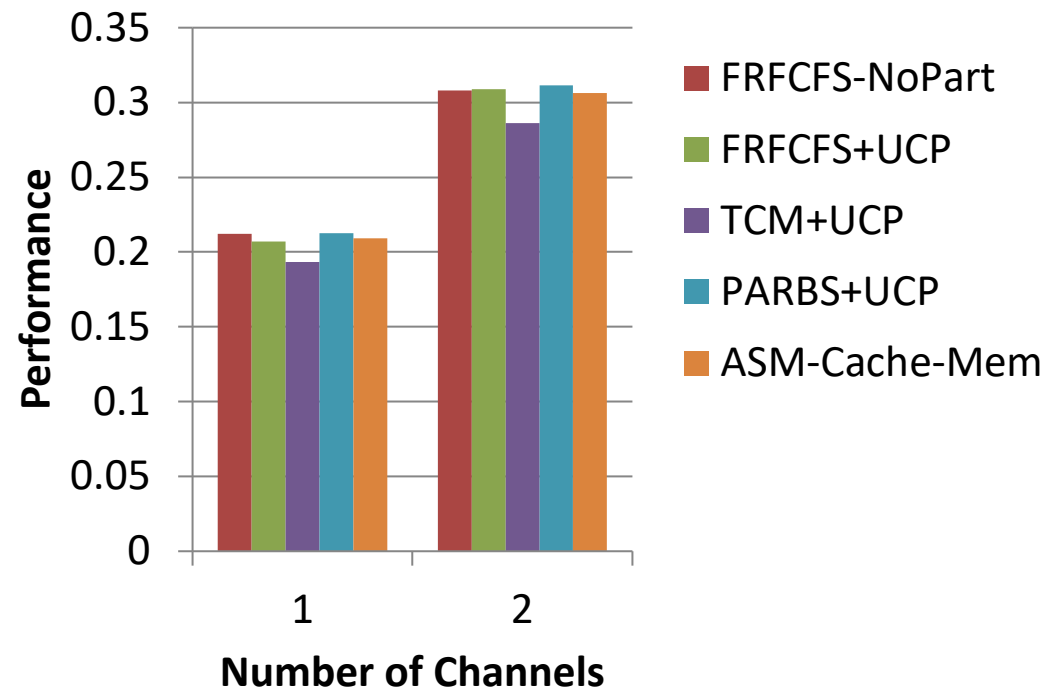
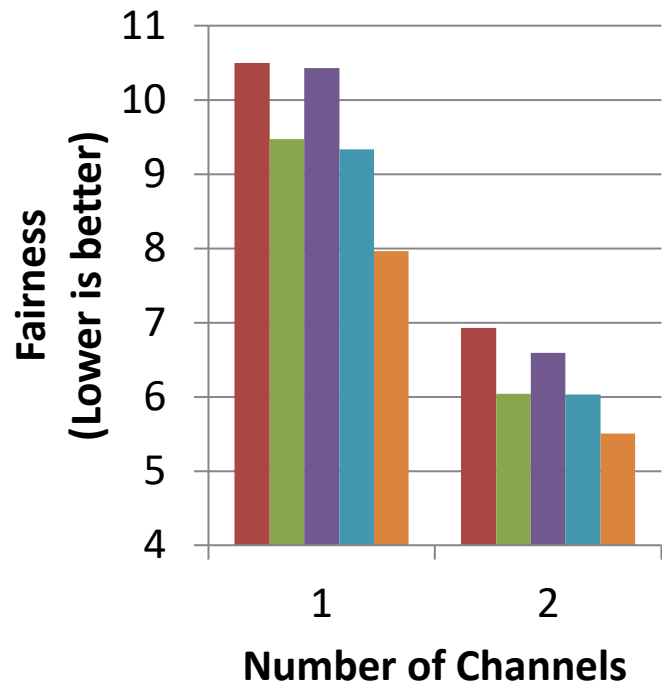
Coordinated Resource Allocation Schemes



- 1. Employ ASM-Cache to partition cache capacity*
- 2. Drive ASM-Mem with slowdowns from ASM-Cache*

Fairness and Performance Results

*16-core system
100 workloads*



Significant fairness benefits across different channel counts

Summary

- Problem: Uncontrolled memory interference cause high and unpredictable application slowdowns
- Goal: Quantify and control slowdowns
- Key Contribution:
 - ASM: An accurate slowdown estimation model
 - Average error of ASM: 10%
- Key Ideas:
 - Shared cache access rate is a proxy for performance
 - Cache Access Rate_{Alone} can be estimated by minimizing memory interference and quantifying cache interference
- Applications of Our Model
 - Slowdown-aware cache and memory management to achieve high performance, fairness and performance guarantees
- *Source Code Released in January 2016*

More on Application Slowdown Model

- Lavanya Subramanian, Vivek Seshadri, Arnab Ghosh, Samira Khan, and Onur Mutlu,
"The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory"
Proceedings of the 48th International Symposium on Microarchitecture (MICRO), Waikiki, Hawaii, USA, December 2015.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides \(pptx\)](#)] [[pdf](#)] [[Poster \(pptx\)](#)] [[pdf](#)]
[[Source Code](#)]

The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory

Lavanya Subramanian*§ Vivek Seshadri* Arnab Ghosh*†
Samira Khan*‡ Onur Mutlu*

*Carnegie Mellon University §Intel Labs †IIT Kanpur ‡University of Virginia

Interconnect QoS/Performance Ideas

Application-Aware Prioritization in NoCs

- Das et al., “Application-Aware Prioritization Mechanisms for On-Chip Networks,” MICRO 2009.
 - https://users.ece.cmu.edu/~omutlu/pub/app-aware-noc_micro09.pdf

Application-Aware Prioritization Mechanisms for On-Chip Networks

Reetuparna Das[§] Onur Mutlu[†] Thomas Moscibroda[‡] Chita R. Das[§]
§Pennsylvania State University †Carnegie Mellon University ‡Microsoft Research
{rdas,das}@cse.psu.edu onur@cmu.edu moscitho@microsoft.com

Slack-Based Packet Scheduling

- Reetuparna Das, Onur Mutlu, Thomas Moscibroda, and Chita R. Das, **"Aergia: Exploiting Packet Latency Slack in On-Chip Networks"** *Proceedings of the 37th International Symposium on Computer Architecture (ISCA)*, pages 106-116, Saint-Malo, France, June 2010. [Slides \(pptx\)](#)

Aérgia: Exploiting Packet Latency Slack in On-Chip Networks

Reetuparna Das[§] Onur Mutlu[†] Thomas Moscibroda[‡] Chita R. Das[§]

[§]Pennsylvania State University
{rdas,das}@cse.psu.edu

[†]Carnegie Mellon University
onur@cmu.edu

[‡]Microsoft Research
moscitho@microsoft.com

Low-Cost QoS in On-Chip Networks (I)

- Boris Grot, Stephen W. Keckler, and Onur Mutlu,
"Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip"
Proceedings of the 42nd International Symposium on Microarchitecture (MICRO), pages 268-279, New York, NY, December 2009. [Slides \(pdf\)](#)

Preemptive Virtual Clock: A Flexible, Efficient, and Cost-effective QOS Scheme for Networks-on-Chip

Boris Grot

Stephen W. Keckler

Onur Mutlu[†]

Department of Computer Sciences
The University of Texas at Austin
{bgrot, skeckler}@cs.utexas.edu}

[†]Computer Architecture Laboratory (CALCM)
Carnegie Mellon University
onur@cmu.edu

Low-Cost QoS in On-Chip Networks (II)

- Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu,
"Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees"
Proceedings of the 38th International Symposium on Computer Architecture (ISCA), San Jose, CA, June 2011. [Slides \(pptx\)](#)

Kilo-NOC: A Heterogeneous Network-on-Chip Architecture for Scalability and Service Guarantees

Boris Grot¹
bgrot@cs.utexas.edu

Joel Hestness¹
hestness@cs.utexas.edu

Stephen W. Keckler^{1,2}
skeckler@nvidia.com

Onur Mutlu³
onur@cmu.edu

¹The University of Texas at Austin
Austin, TX

²NVIDIA
Santa Clara, CA

³Carnegie Mellon University
Pittsburgh, PA

Throttling Based Fairness in NoCs

- Kevin Chang, Rachata Ausavarungnirun, Chris Fallin, and Onur Mutlu, **"HAT: Heterogeneous Adaptive Throttling for On-Chip Networks"**
Proceedings of the 24th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD), New York, NY, October 2012. [Slides \(pptx\)](#) [\(pdf\)](#)

HAT: Heterogeneous Adaptive Throttling for On-Chip Networks

Kevin Kai-Wei Chang, Rachata Ausavarungnirun, Chris Fallin, Onur Mutlu
Carnegie Mellon University
{kevincha, rachata, cfallin, onur}@cmu.edu

Scalability: Express Cube Topologies

- Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu, **"Express Cube Topologies for On-Chip Interconnects"** *Proceedings of the 15th International Symposium on High-Performance Computer Architecture (HPCA)*, pages 163-174, Raleigh, NC, February 2009. [Slides \(ppt\)](#)

Express Cube Topologies for On-Chip Interconnects

Boris Grot

Joel Hestness

Stephen W. Keckler

Onur Mutlu[†]

Department of Computer Sciences

The University of Texas at Austin

{bgrot, hestness, skeckler}@cs.utexas.edu

[†]Computer Architecture Laboratory (CALCM)

Carnegie Mellon University

onur@cmu.edu

Scalability: Slim NoC

- Maciej Besta, Syed Minhaj Hassan, Sudhakar Yalamanchili, Rachata Ausavarungnirun, Onur Mutlu, Torsten Hoefler, **"Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability"**
Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS), Williamsburg, VA, USA, March 2018.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides \(pptx\)](#)] [[pdf](#)]
[[Poster \(pdf\)](#)]

Slim NoC: A Low-Diameter On-Chip Network Topology for High Energy Efficiency and Scalability

Maciej Besta¹

Syed Minhaj Hassan²

Sudhakar Yalamanchili²

Rachata Ausavarungnirun³

Onur Mutlu^{1,3}

Torsten Hoefler¹

¹ETH Zürich

²Georgia Institute of Technology

³Carnegie Mellon University

Bufferless Routing in NoCs

- Moscibroda and Mutlu, “A Case for Bufferless Routing in On-Chip Networks,” ISCA 2009.
 - https://users.ece.cmu.edu/~omutlu/pub/bless_isca09.pdf

A Case for Bufferless Routing in On-Chip Networks

Thomas Moscibroda
Microsoft Research
moscitho@microsoft.com

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

CHIPPER: Low-Complexity Bufferless

- Chris Fallin, Chris Craik, and Onur Mutlu,
"CHIPPER: A Low-Complexity Bufferless Deflection Router"
Proceedings of the 17th International Symposium on High-Performance Computer Architecture (HPCA), pages 144-155, San Antonio, TX, February 2011. Slides (pptx)
An extended version as *SAFARI Technical Report*, TR-SAFARI-2010-001, Carnegie Mellon University, December 2010.

CHIPPER: A Low-complexity Bufferless Deflection Router

Chris Fallin Chris Craik Onur Mutlu
cfallin@cmu.edu craik@cmu.edu onur@cmu.edu

Computer Architecture Lab (CALCM)
Carnegie Mellon University

Minimally-Buffered Deflection Routing

- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,
"MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect"
Proceedings of the 6th ACM/IEEE International Symposium on Networks on Chip (NOCS), Lyngby, Denmark, May 2012. [Slides](#) (pptx) (pdf)

MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect

Chris Fallin, Greg Nazario, Xiangyao Yu[†], Kevin Chang, Rachata Ausavarungnirun, Onur Mutlu

Carnegie Mellon University
{cfallin,gnazario,kevincha,rachata,onur}@cmu.edu

[†]Tsinghua University & Carnegie Mellon University
yxythu@gmail.com

“Bufferless” Hierarchical Rings

- Ausavarungnirun et al., “Design and Evaluation of Hierarchical Rings with Deflection Routing,” SBAC-PAD 2014.
 - http://users.ece.cmu.edu/~omutlu/pub/hierarchical-rings-with-deflection_sbacpad14.pdf
- Discusses the design and implementation of a mostly-bufferless hierarchical ring

Design and Evaluation of Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun Chris Fallin Xiangyao Yu† Kevin Kai-Wei Chang
Greg Nazario Reetuparna Das§ Gabriel H. Loh‡ Onur Mutlu

Carnegie Mellon University §University of Michigan †MIT ‡Advanced Micro Devices, Inc.

“Bufferless” Hierarchical Rings (II)

- Rachata Ausavarungnirun, Chris Fallin, Xiangyao Yu, Kevin Chang, Greg Nazario, Reetuparna Das, Gabriel Loh, and Onur Mutlu,
"A Case for Hierarchical Rings with Deflection Routing: An Energy-Efficient On-Chip Communication Substrate"
Parallel Computing (PARCO), to appear in 2016.
 - [arXiv.org version](https://arxiv.org/abs/1602.04878), February 2016.

Achieving both High Energy Efficiency
and High Performance in On-Chip Communication
using Hierarchical Rings with Deflection Routing

Rachata Ausavarungnirun Chris Fallin Xiangyao Yu[†] Kevin Kai-Wei Chang
Greg Nazario Reetuparna Das[§] Gabriel H. Loh[‡] Onur Mutlu
Carnegie Mellon University §University of Michigan †MIT ‡AMD

Summary of Six Years of Research

- Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, and Onur Mutlu,
"Bufferless and Minimally-Buffered Deflection Routing"
Invited Book Chapter in Routing Algorithms in Networks-on-Chip, pp. 241-275, Springer, 2014.

Chapter 1

Bufferless and Minimally-Buffered Deflection Routing

Chris Fallin, Greg Nazario, Xiangyao Yu, Kevin Chang, Rachata Ausavarungnirun, Onur Mutlu

On-Chip vs. Off-Chip Tradeoffs

- George Nychis, Chris Fallin, Thomas Moscibroda, Onur Mutlu, and Srinivasan Seshan,
"On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-core Interconnects"
*Proceedings of the 2012 ACM SIGCOMM Conference (**SIGCOMM**)*, Helsinki, Finland, August 2012. [Slides \(pptx\)](#)

On-Chip Networks from a Networking Perspective: Congestion and Scalability in Many-Core Interconnects

George Nychis[†], Chris Fallin[†], Thomas Moscibroda[§], Onur Mutlu[†], Srinivasan Seshan[†]

[†] Carnegie Mellon University
{gnychis,cfallin,onur,srini}@cmu.edu

[§] Microsoft Research Asia
moscitho@microsoft.com

Slowdown Estimation in NoCs

- Xiyue Xiang, Saugata Ghose, Onur Mutlu, and Nian-Feng Tzeng,
"A Model for Application Slowdown Estimation in On-Chip Networks and Its Use for Improving System Fairness and Performance"
Proceedings of the 34th IEEE International Conference on Computer Design (ICCD), Phoenix, AZ, USA, October 2016.
[[Slides \(pptx\)](#) ([pdf](#))]

A Model for Application Slowdown Estimation in On-Chip Networks and Its Use for Improving System Fairness and Performance

Xiyue Xiang[†]

Saugata Ghose[‡]

Onur Mutlu^{§‡}

Nian-Feng Tzeng[†]

[†]*University of Louisiana at Lafayette*

[‡]*Carnegie Mellon University*

[§]*ETH Zürich*

Handling Multicast and Hotspot Issues

- Xiyue Xiang, Wentao Shi, Saugata Ghose, Lu Peng, Onur Mutlu, and Nian-Feng Tzeng,
"Carpool: A Bufferless On-Chip Network Supporting Adaptive Multicast and Hotspot Alleviation"
Proceedings of the International Conference on Supercomputing (ICS), Chicago, IL, USA, June 2017.
[[Slides \(pptx\)](#) ([pdf](#))]

Carpool: A Bufferless On-Chip Network Supporting Adaptive Multicast and Hotspot Alleviation

Xiyue Xiang[†] Wentao Shi[★] Saugata Ghose[‡] Lu Peng[★] Onur Mutlu^{§‡} Nian-Feng Tzeng[†]

[†]University of Louisiana at Lafayette [★]Louisiana State University [‡]Carnegie Mellon University [§]ETH Zürich