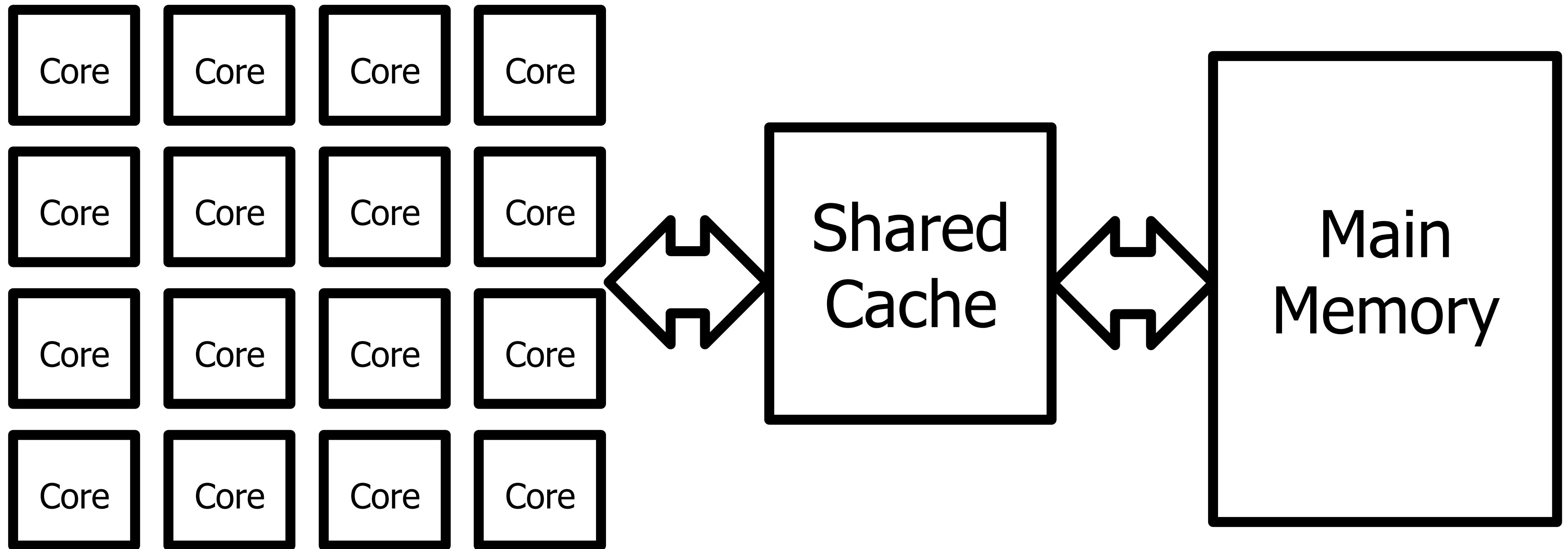
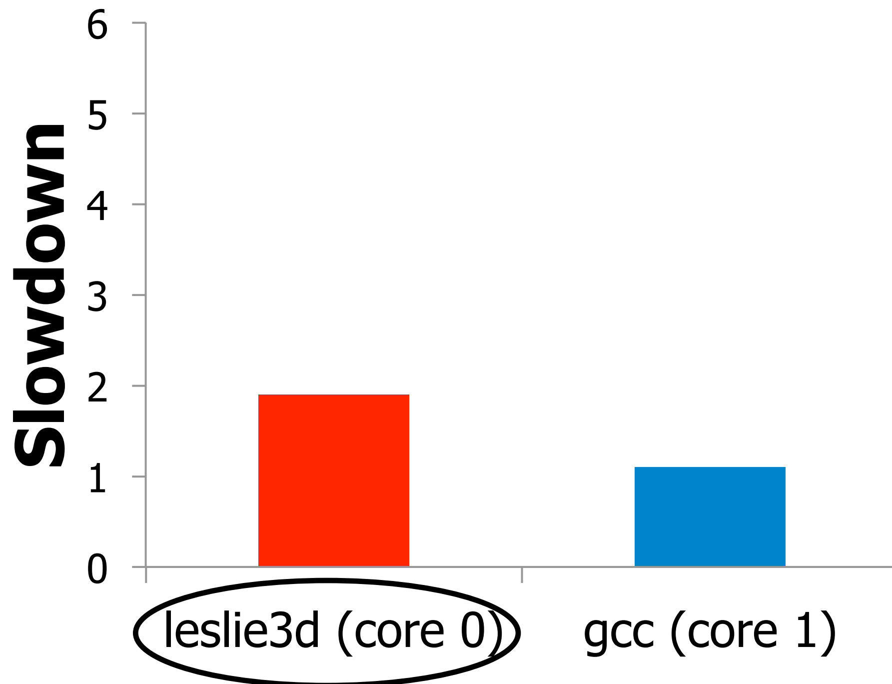
18-447
Computer Architecture
Lecture 31: Predictable Performance

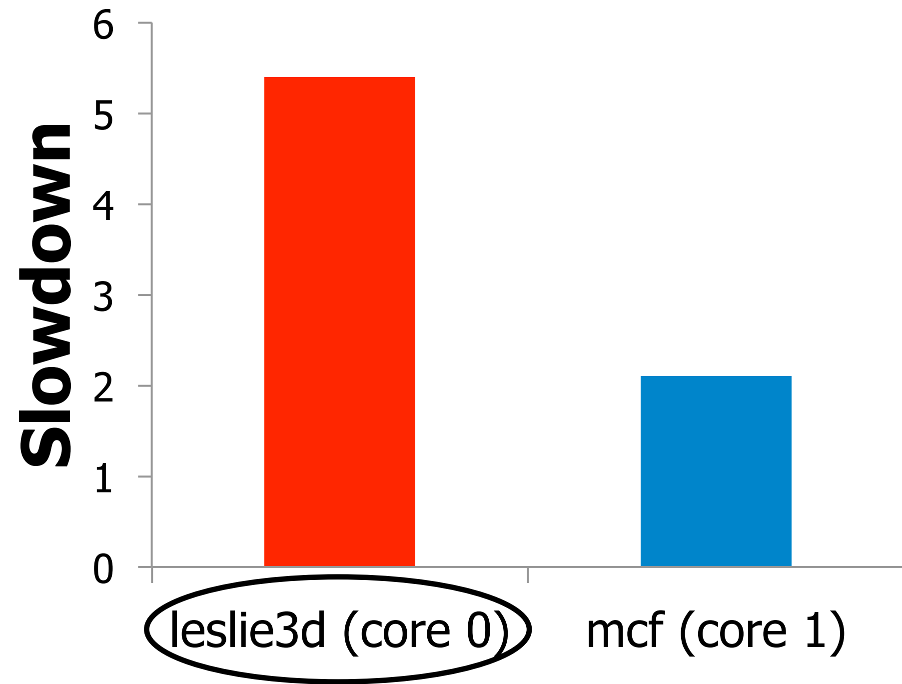
Lavanya Subramanian
Carnegie Mellon University
Spring 2015, 4/15/2015

Shared Resource Interference



High and Unpredictable Application Slowdowns





1. High application slowdowns due to shared resource interference

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

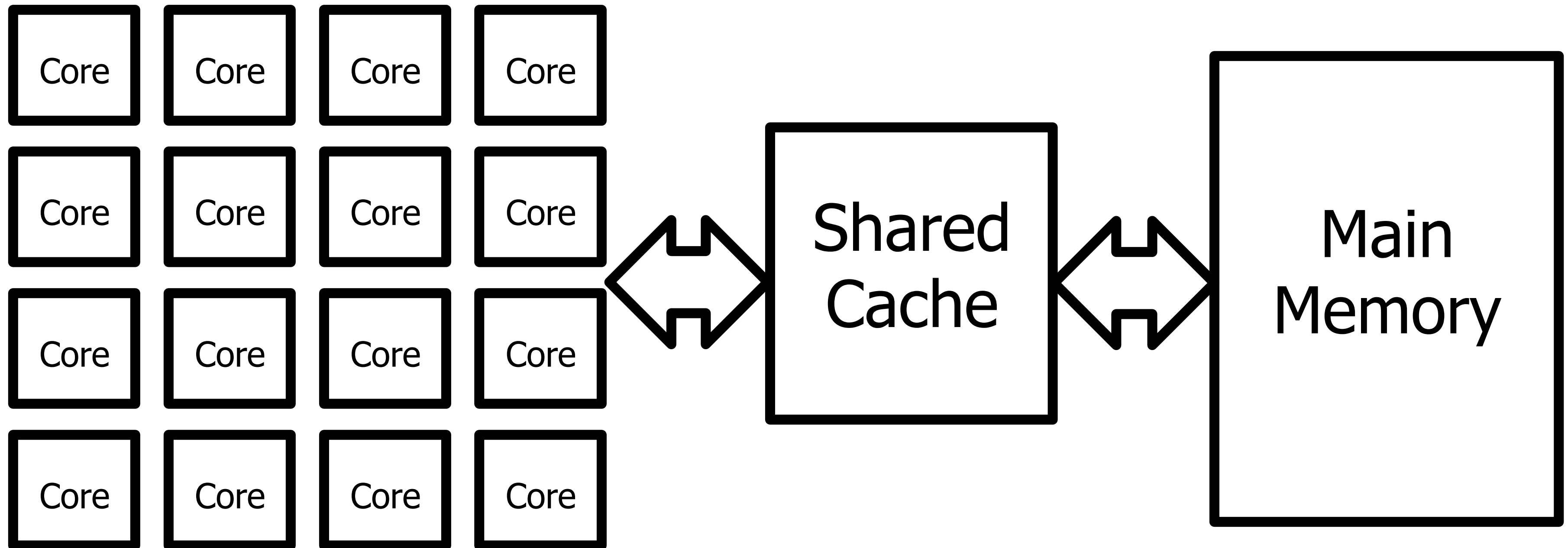
Need for Predictable Performance

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

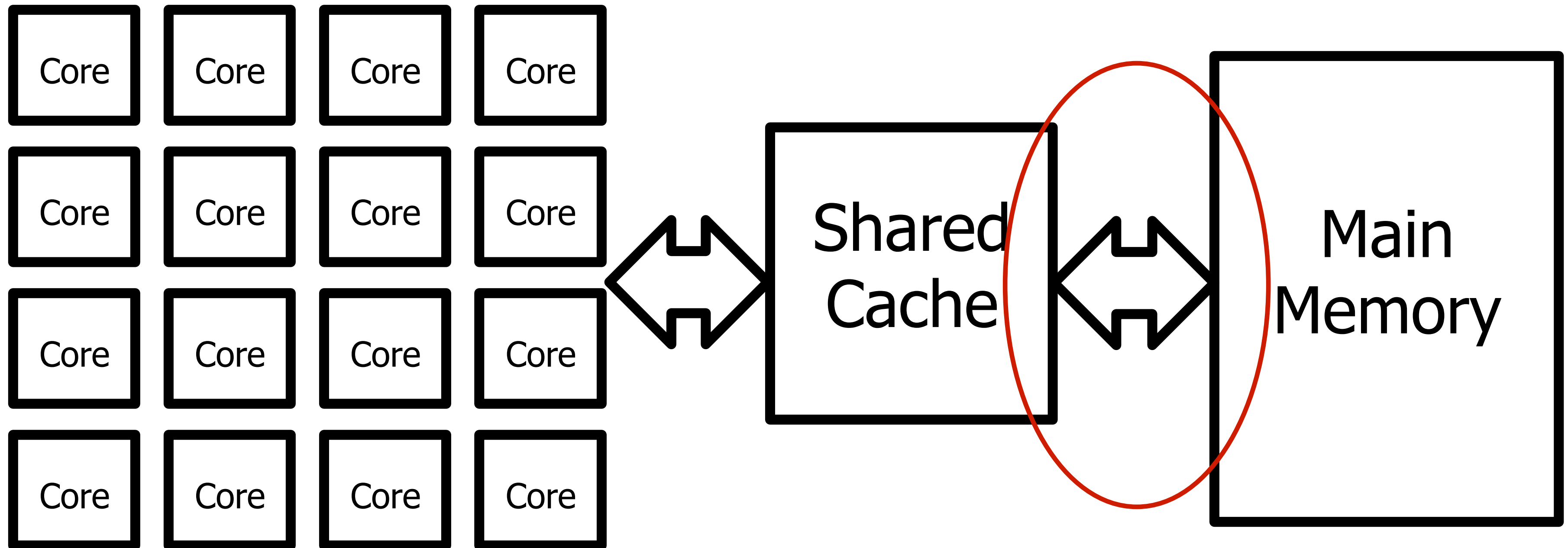
Our Goal: Predictable performance in the presence of shared resources

- Example 2: In mobile systems
  - Interactive applications run with non-interactive applications
  - Need to guarantee performance for interactive applications

Tackling Different Parts of the Shared Memory Hierarchy



Predictability in the Presence of Memory Bandwidth Interference



Predictability in the Presence of Memory Bandwidth Interference (HPCA 2013)

1. Estimate Slowdown

2. Control Slowdown

Predictability in the Presence of Memory Bandwidth Interference

1. Estimate Slowdown
   - Key Observations
   - Implementation
   - MISE Model: Putting it All Together
   - Evaluating the Model
2. Control Slowdown
   - Providing Soft Slowdown Guarantees

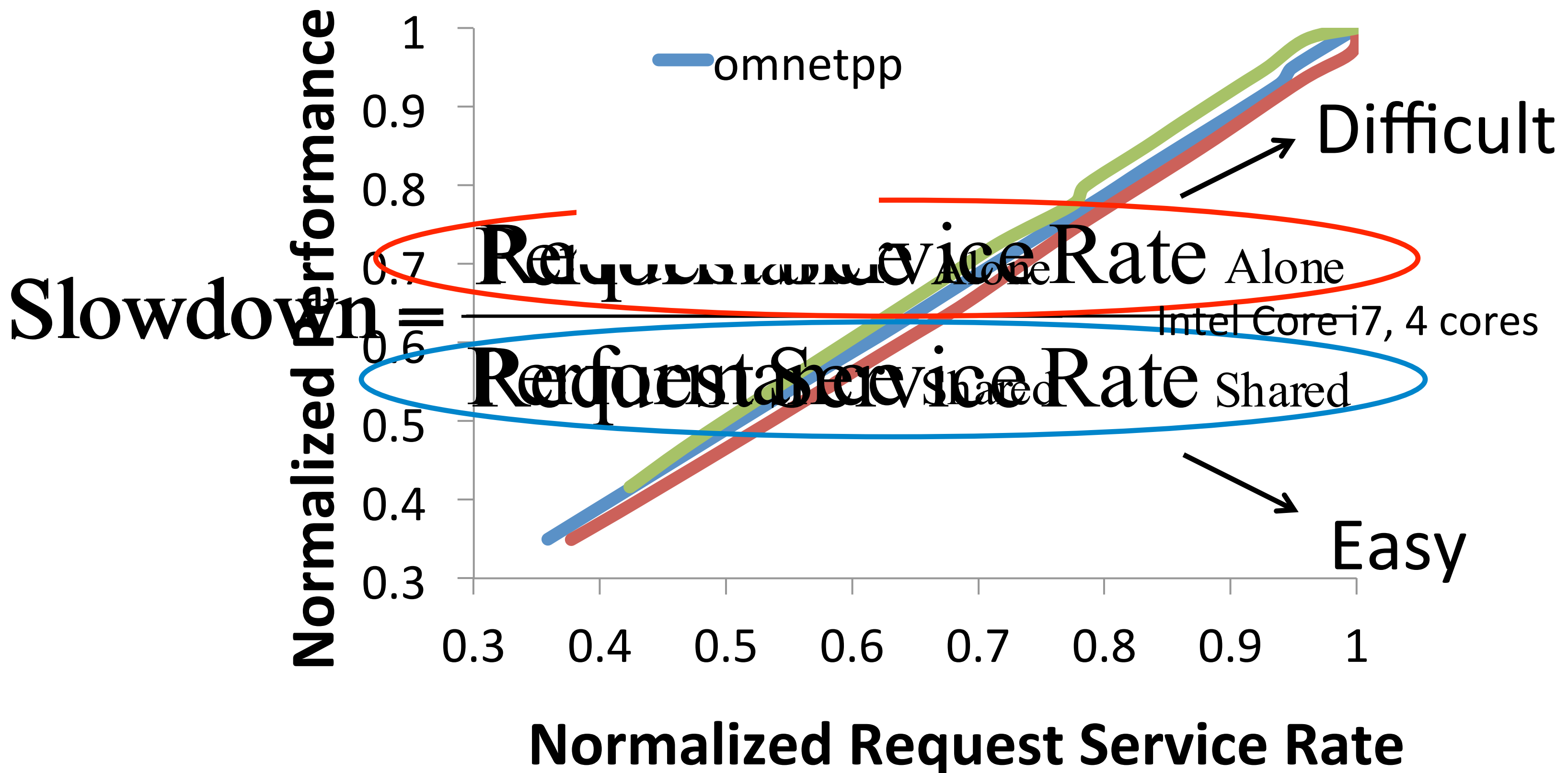
Slowdown: Definition

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

Key Observation 1

For a memory bound application,

Performance ∝ 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 at the *memory controller*

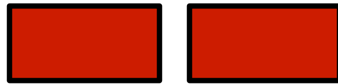
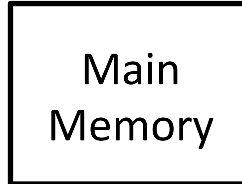
Highest priority → Little interference

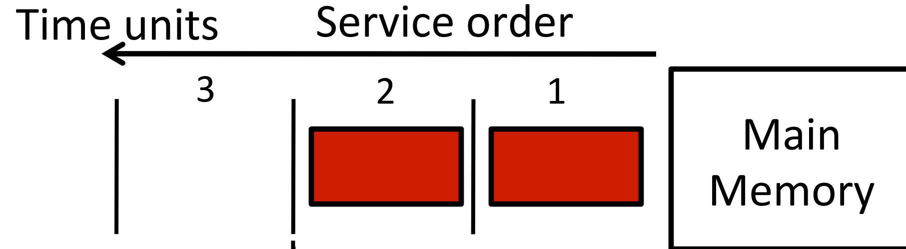
(almost as if the application were run alone)

Key Observation 2

1. Run alone

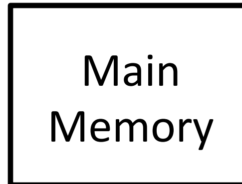
Request Buffer State

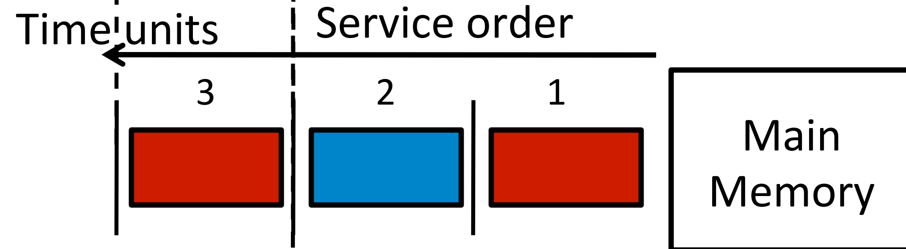





2. Run with another application

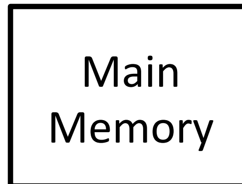
Request Buffer State

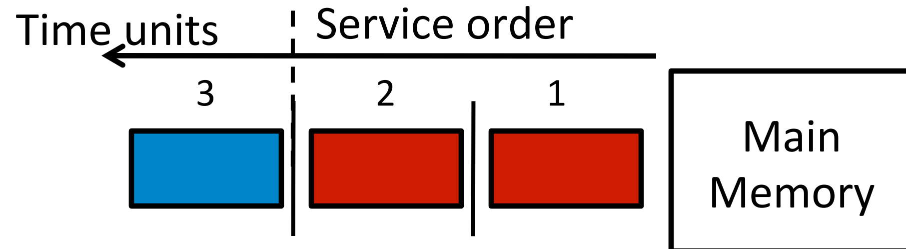




3. Run with another application: highest priority

Request Buffer State



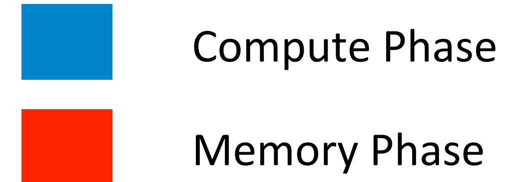


Memory Interference-induced Slowdown Estimation (MISE) model for memory bound applications

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

Key Observation 3

- Memory-bound application



Memory phase slowdown dominates overall slowdown

Key Observation 3

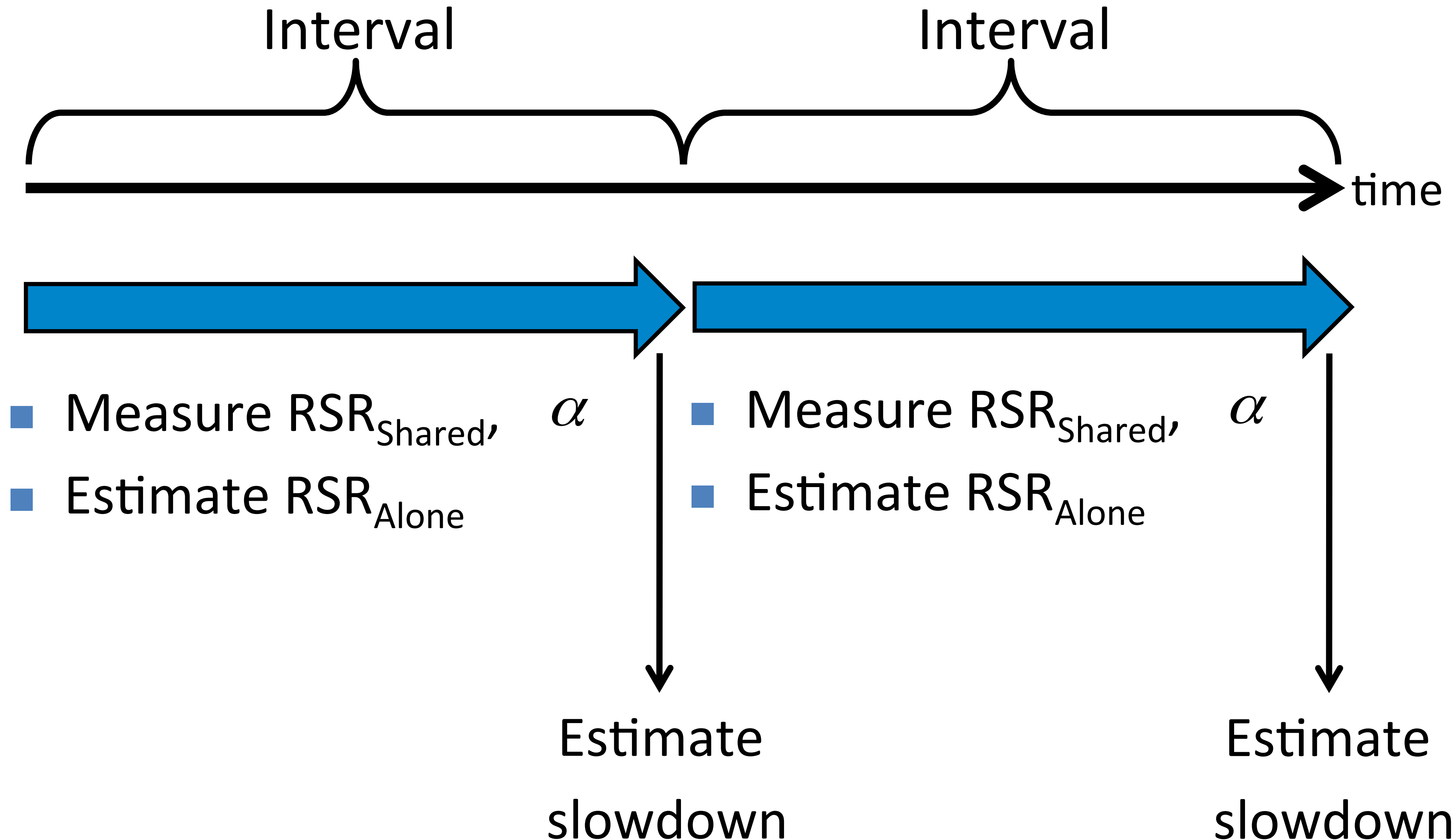
Memory Interference-induced Slowdown Estimation (MISE) model for non-memory bound applications

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

Predictability in the Presence of Memory Bandwidth Interference

1. Estimate Slowdown
   - Key Observations
   - Implementation
   - MISE Model: Putting it All Together
   - Evaluating the Model
2. Control Slowdown
   - Providing Soft Slowdown Guarantees

Interval Based Operation



Measuring RSR_{Shared} and α

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

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

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

Estimating Request Service Rate $_{Alone}$ (RSR_{Alone})

- Divide each interval into shorter epochs
- At the beginning of each epoch
  - Randomly pick an application as the highest priority application
- At the end of an interval, for each application, estimate

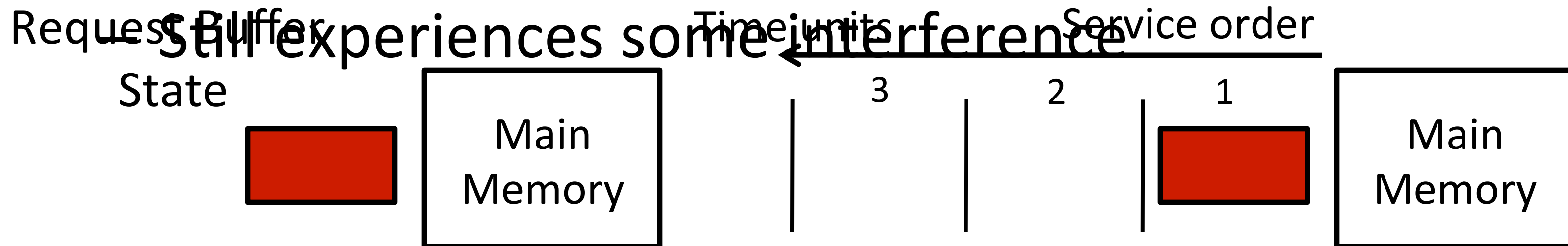
Goal: Estimate RSR_{Alone}

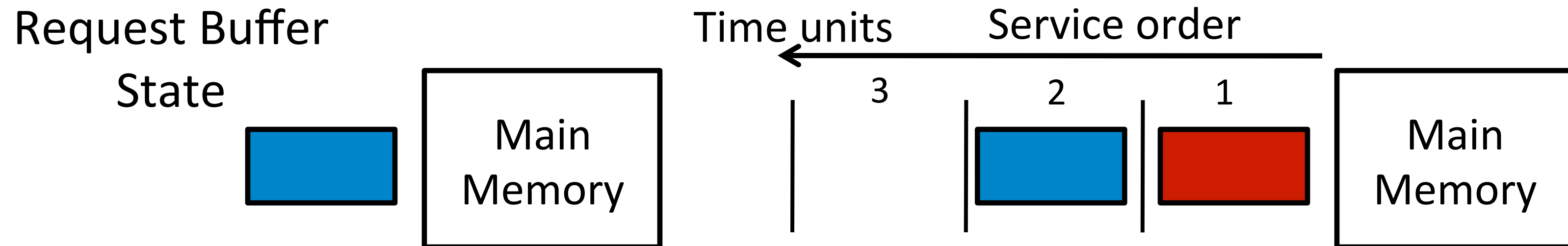
How: Periodically give each application highest priority in accessing memory

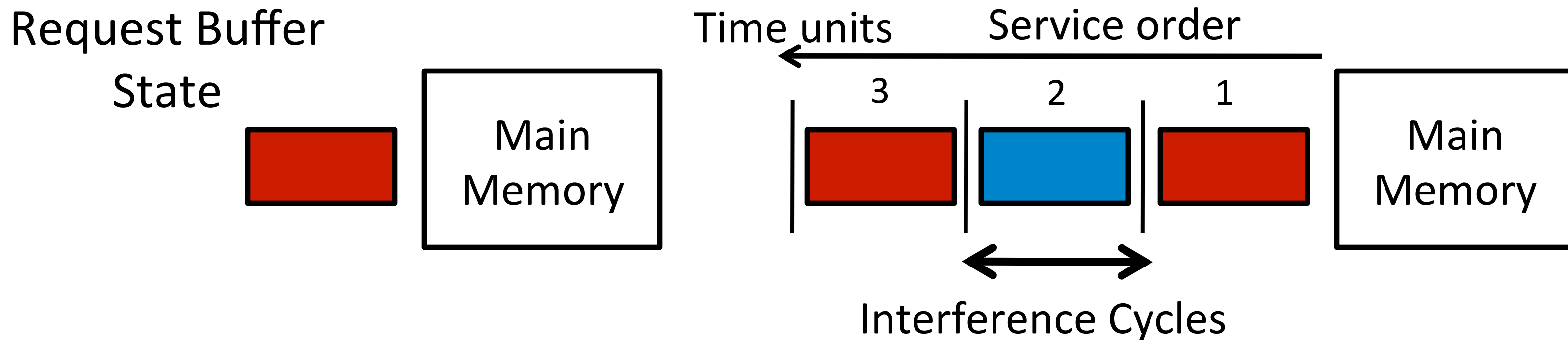
$$RSR_{Alone} = \frac{\text{Number of Requests During High Priority Epochs}}{\text{Number of Cycles Application Given High Priority}}$$

Inaccuracy in Estimating RSR_{Alone}

- When an application has highest priority
  - Still experiences some interference







Accounting for Interference in RSR_{Alone} Estimation

- Solution: Determine and remove interference cycles from ARSR calculation

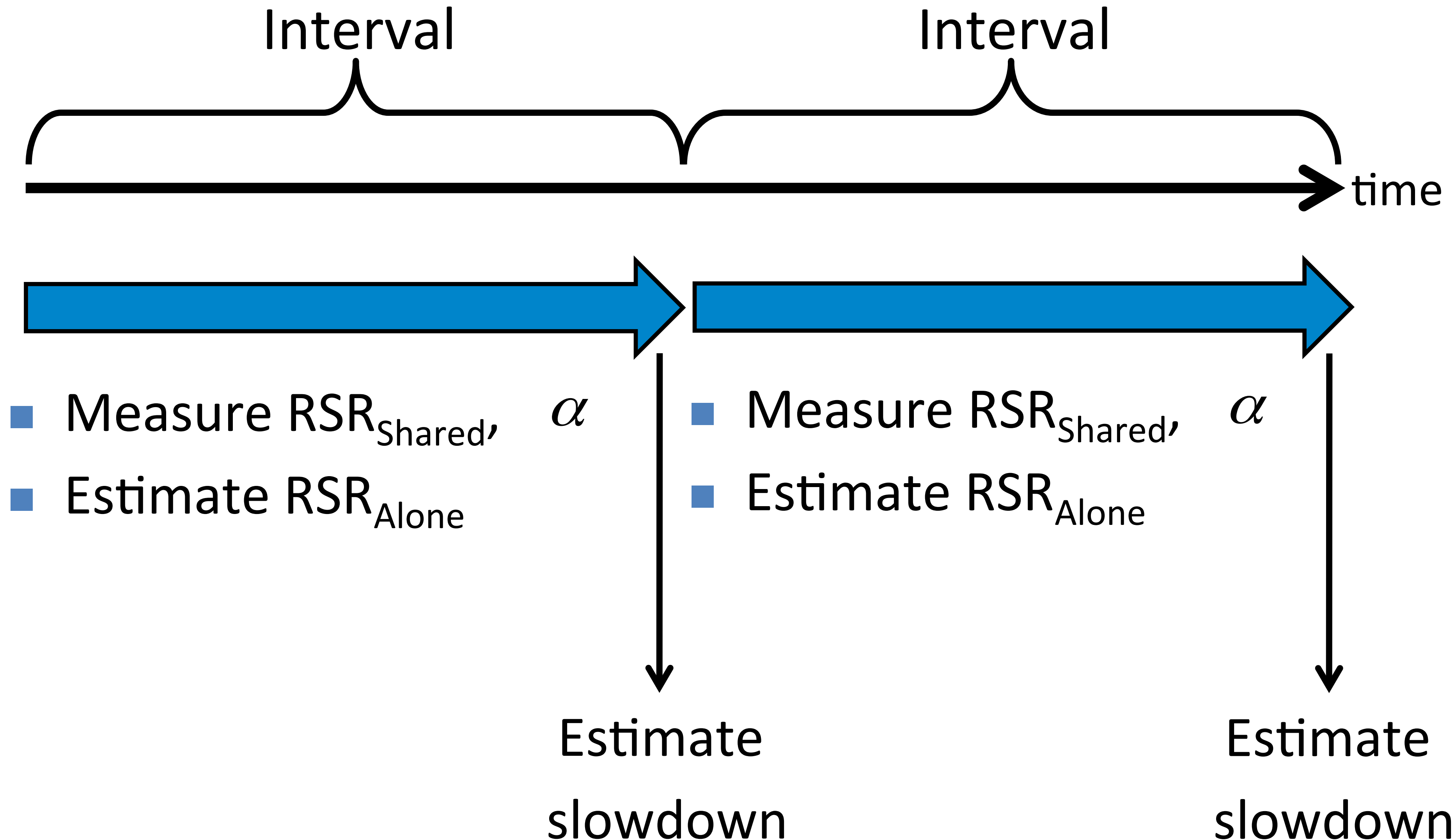
$$\text{ARSR} = \frac{\text{Number of Requests During High Priority Epochs}}{\text{Number of Cycles Application Given High Priority} - \text{Interference Cycles}}$$

- 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

Predictability in the Presence of Memory Bandwidth Interference

1. Estimate Slowdown
   - Key Observations
   - Implementation
   - MISE Model: Putting it All Together
   - Evaluating the Model
2. Control Slowdown
   - Providing Soft Slowdown Guarantees

MISE Operation: Putting it All Together



Predictability in the Presence of Memory Bandwidth Interference

1. Estimate Slowdown
   - Key Observations
   - Implementation
   - MISE Model: Putting it All Together
   - Evaluating the Model
2. Control Slowdown
   - Providing Soft Slowdown Guarantees

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:

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

Stall Time Alone → Difficult

Stall Time Shared → Easy

Count number of cycles application receives interference

Two Major Advantages of MISE Over STFM

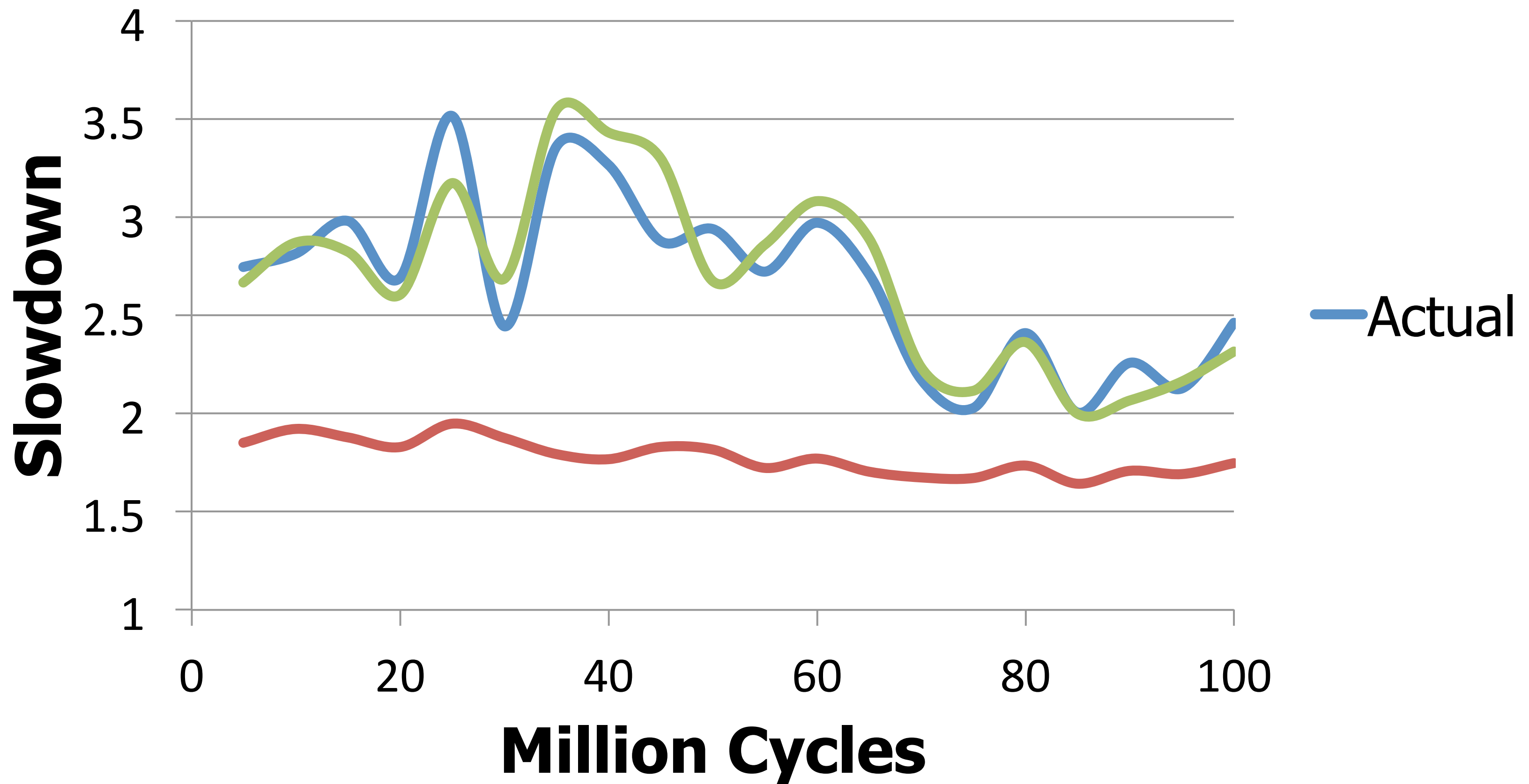
- Advantage 1:
  - STFM estimates alone performance while an application is receiving interference → Difficult
  - MISE estimates alone performance while giving an application the highest priority → Easier
- Advantage 2:
  - STFM does not take into account compute phase for non-memory-bound applications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

Methodology

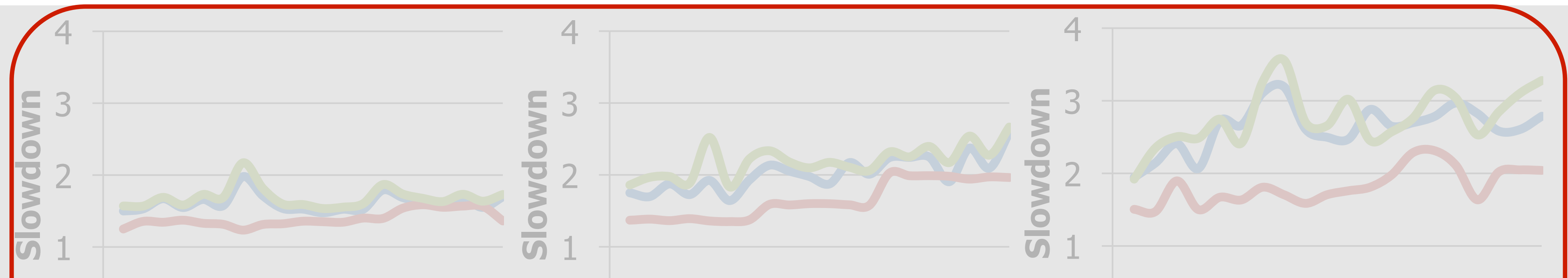
- Configuration of our simulated system
  - 4 cores
  - 1 channel, 8 banks/channel
  - DDR3 1066 DRAM
  - 512 KB private cache/core

- Workloads
  - SPEC CPU2006
  - 300 multi programmed workloads

Quantitative Comparison



Comparison to STFM



Average error of MISE: 8.2%

Average error of STFM: 29.4%

(across 300 workloads)

wrf

calculix

povray

Predictability in the Presence of Memory Bandwidth Interference

1. Estimate Slowdown
   - Key Observations
   - Implementation
   - MISE Model: Putting it All Together
   - Evaluating the Model
2. **Control Slowdown**
   - Providing Soft Slowdown Guarantees

Possible Use Cases

- *Bounding application slowdowns* ***[HPCA '14]***
- *VM migration and admission control schemes* ***[VEE '15]***
- *Fair billing schemes in a commodity cloud*

Predictability in the Presence of Memory Bandwidth Interference

1. Estimate Slowdown
   – Key Observations
   – Implementation
   – MISE Model: Putting it All Together
   – Evaluating the Model
2. **Control Slowdown**
   – Providing Soft Slowdown Guarantees

MISE-QoS: 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

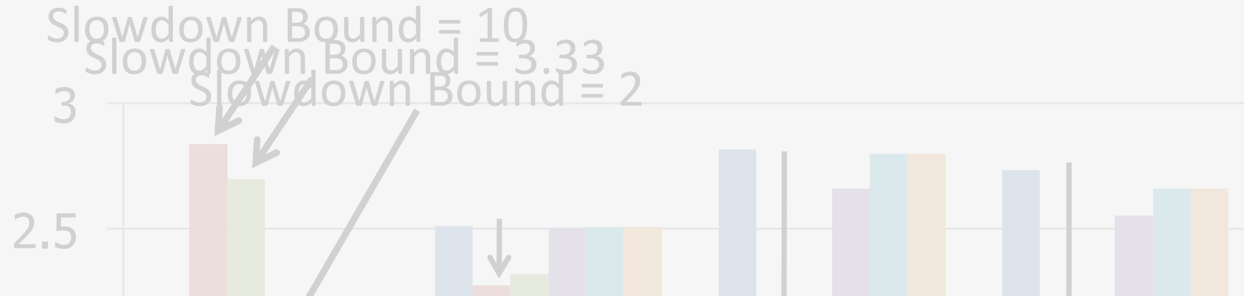
Methodology

- 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 multi programmed workloads
- Each workload run with 10 slowdown bound values
- Baseline memory scheduling mechanism
  - Always prioritize QoS-critical application

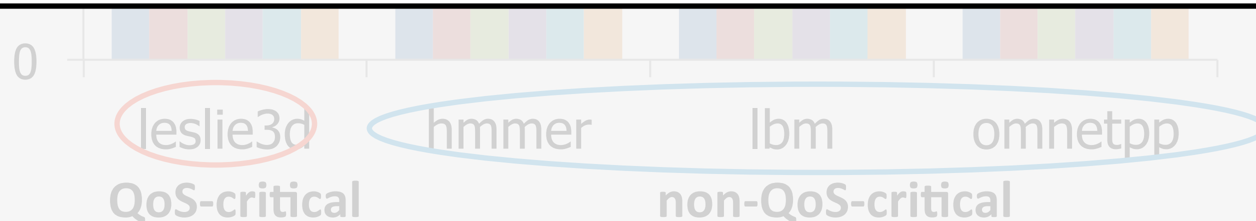
    [Iyer et al., SIGMETRICS 2007]
  - Other applications' requests scheduled in FR-FCFS order

    [Zuravleff and Robinson, 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



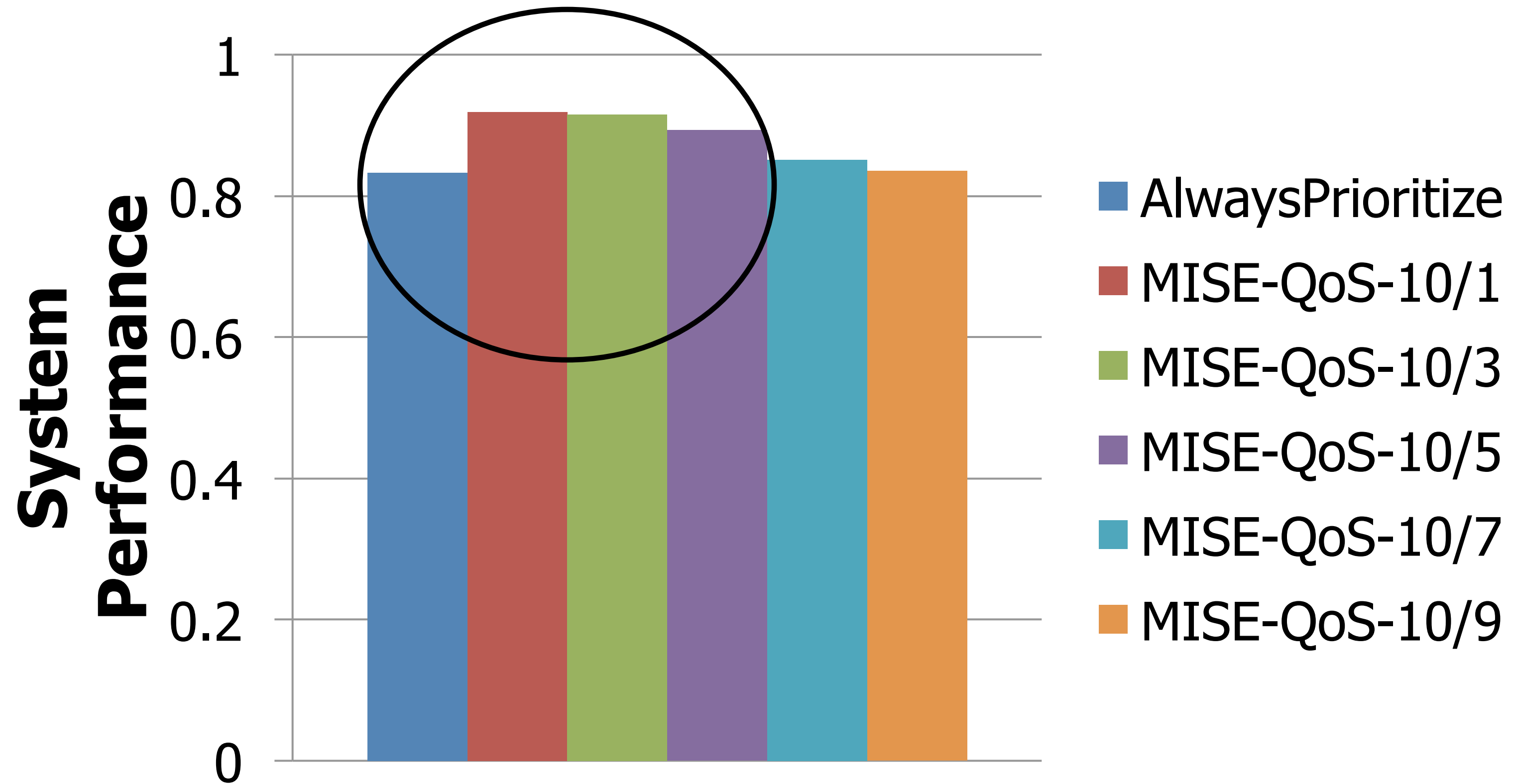
Effectiveness of MISE in Enforcing QoS

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

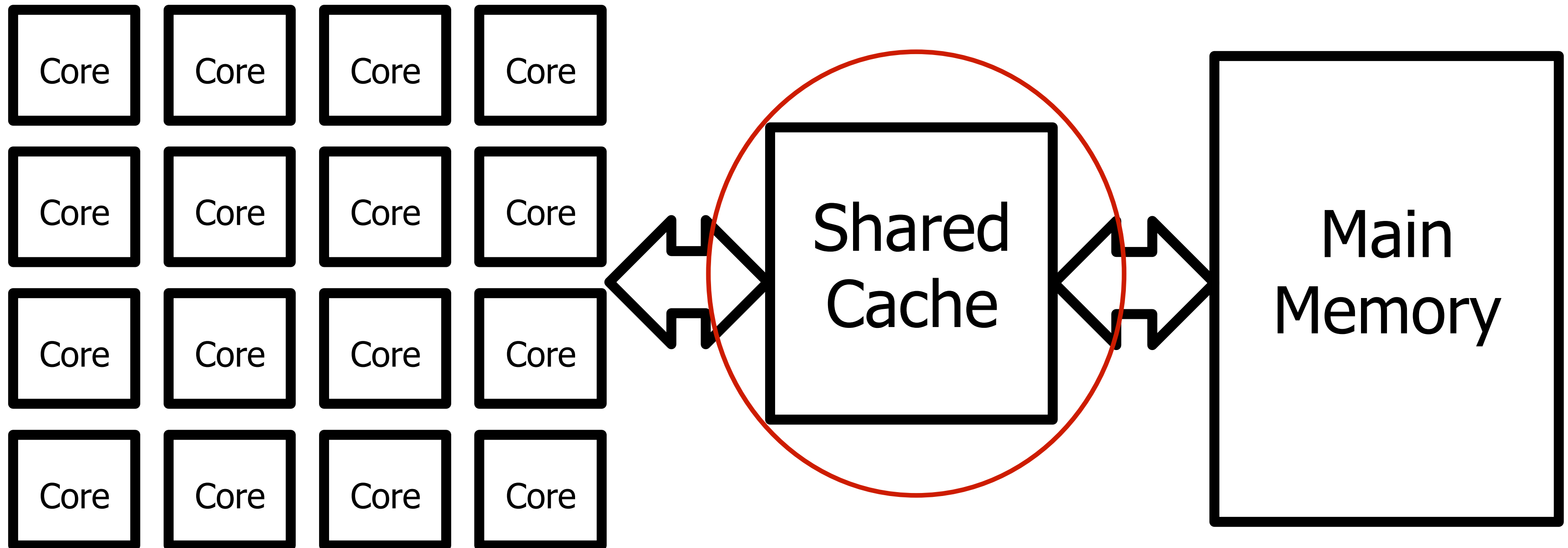


When slowdown bound is 10/3
MISE-QoS improves system performance by 10%

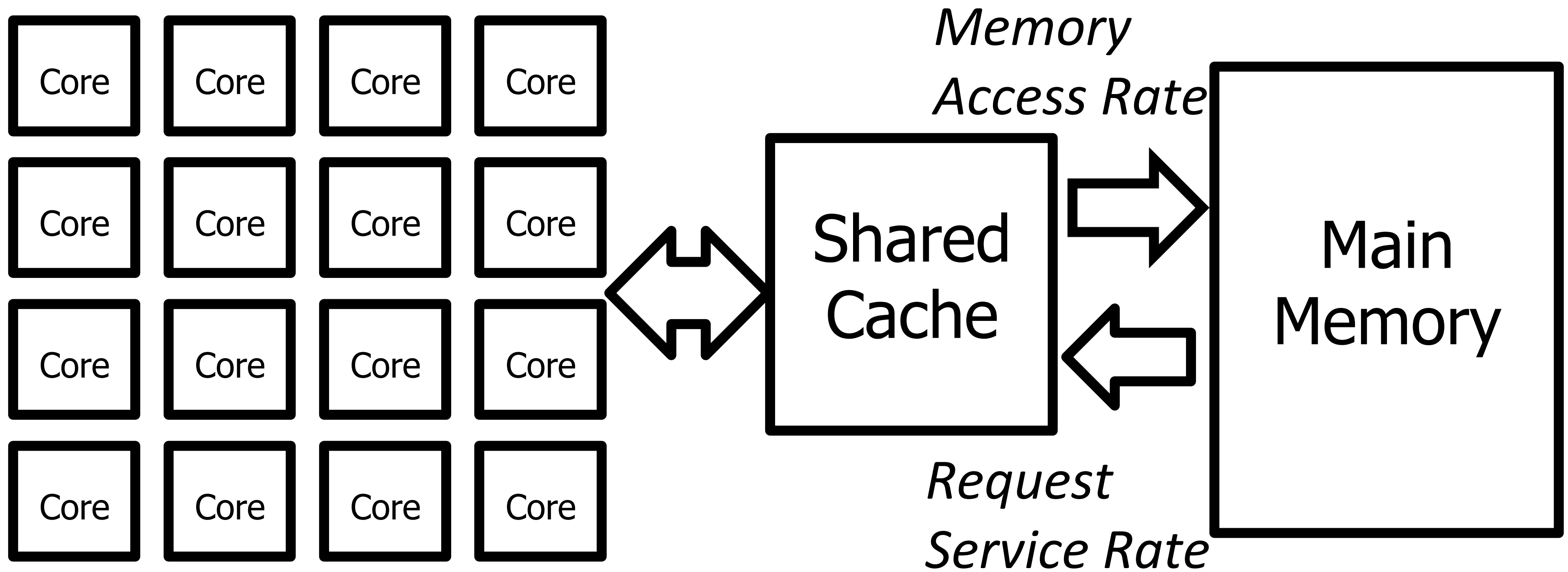
Summary: Predictability in the Presence of Memory Bandwidth Interference

- 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
- Leverage slowdown estimates to control slowdowns; Many more applications exist

Taking Into Account Shared Cache Interference

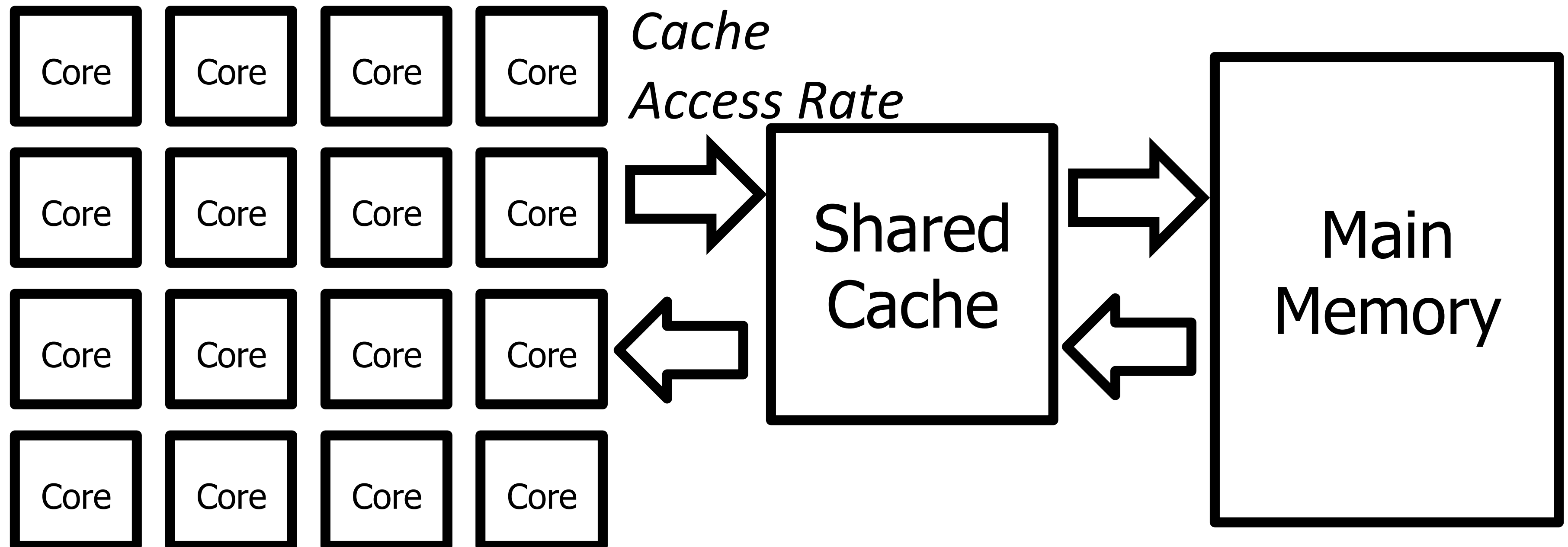


Revisiting Request Service Rates

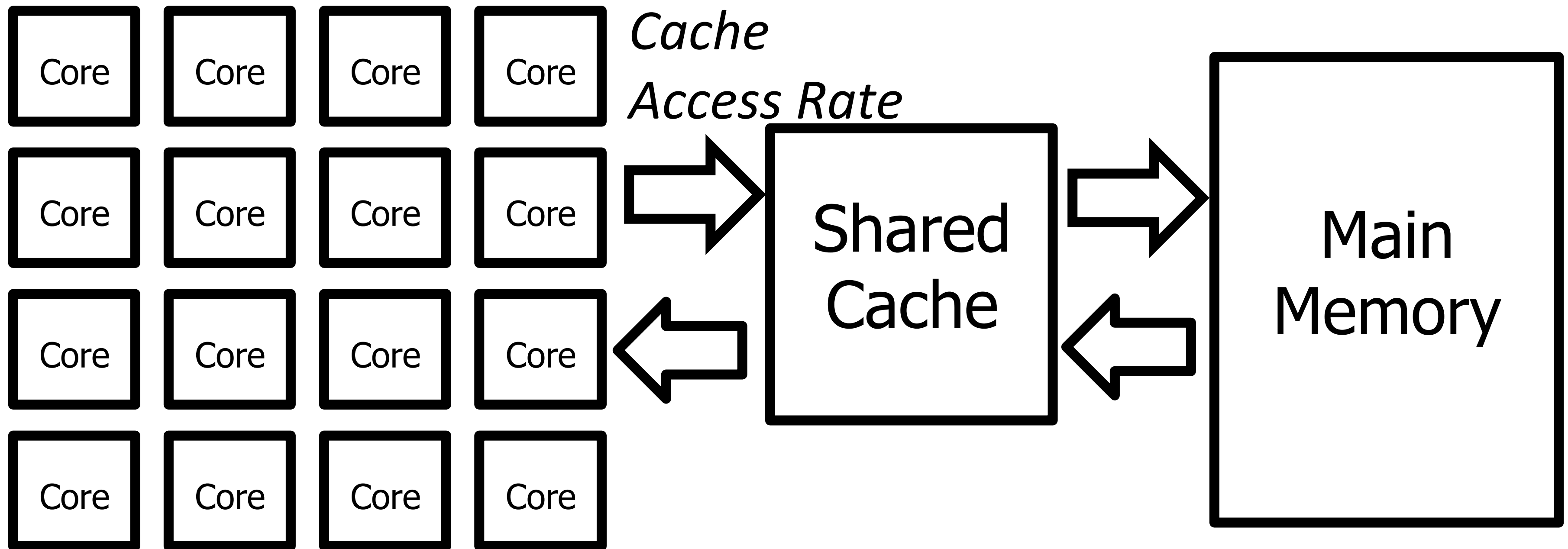


Request service and access rates tightly coupled

Estimating Cache and Memory Slowdowns Through Cache Access Rates

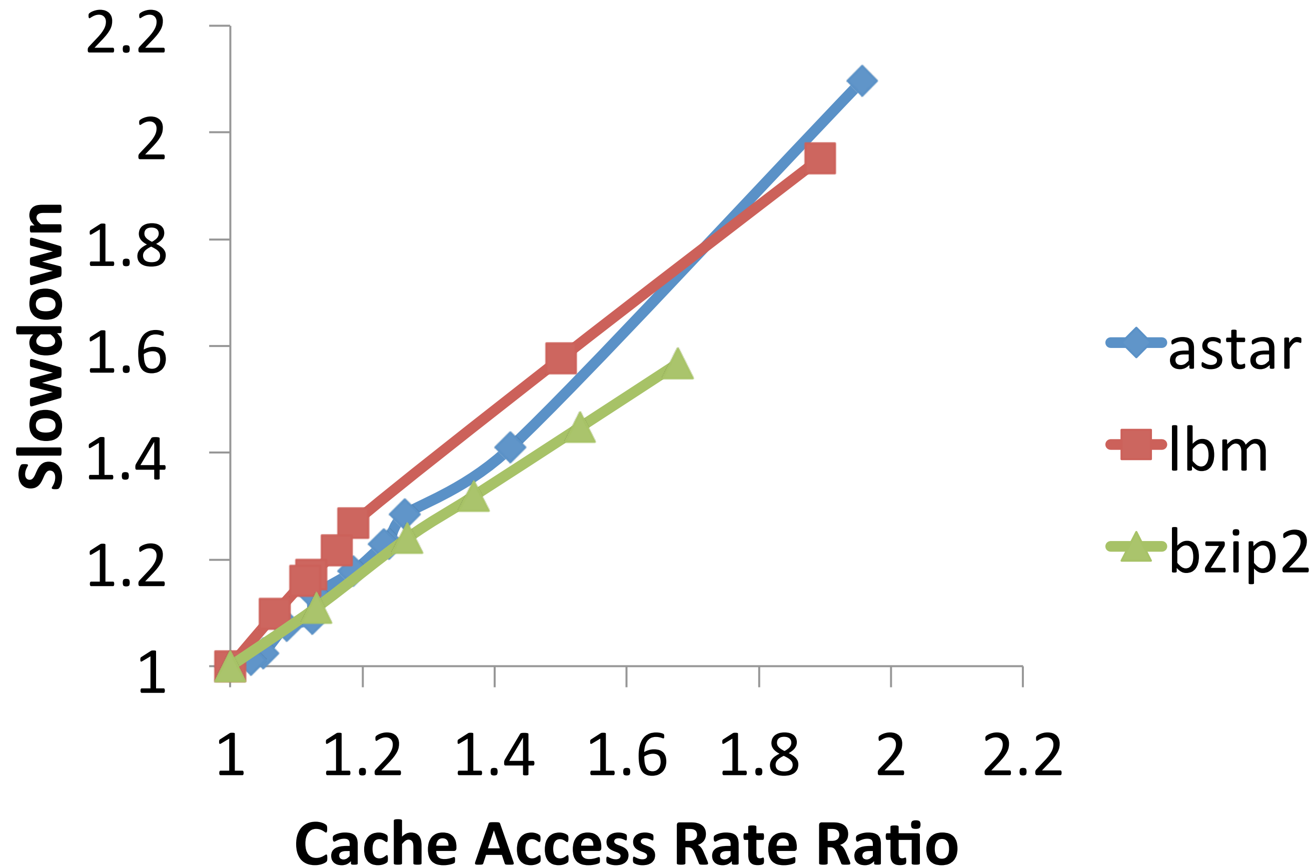


The Application Slowdown Model



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

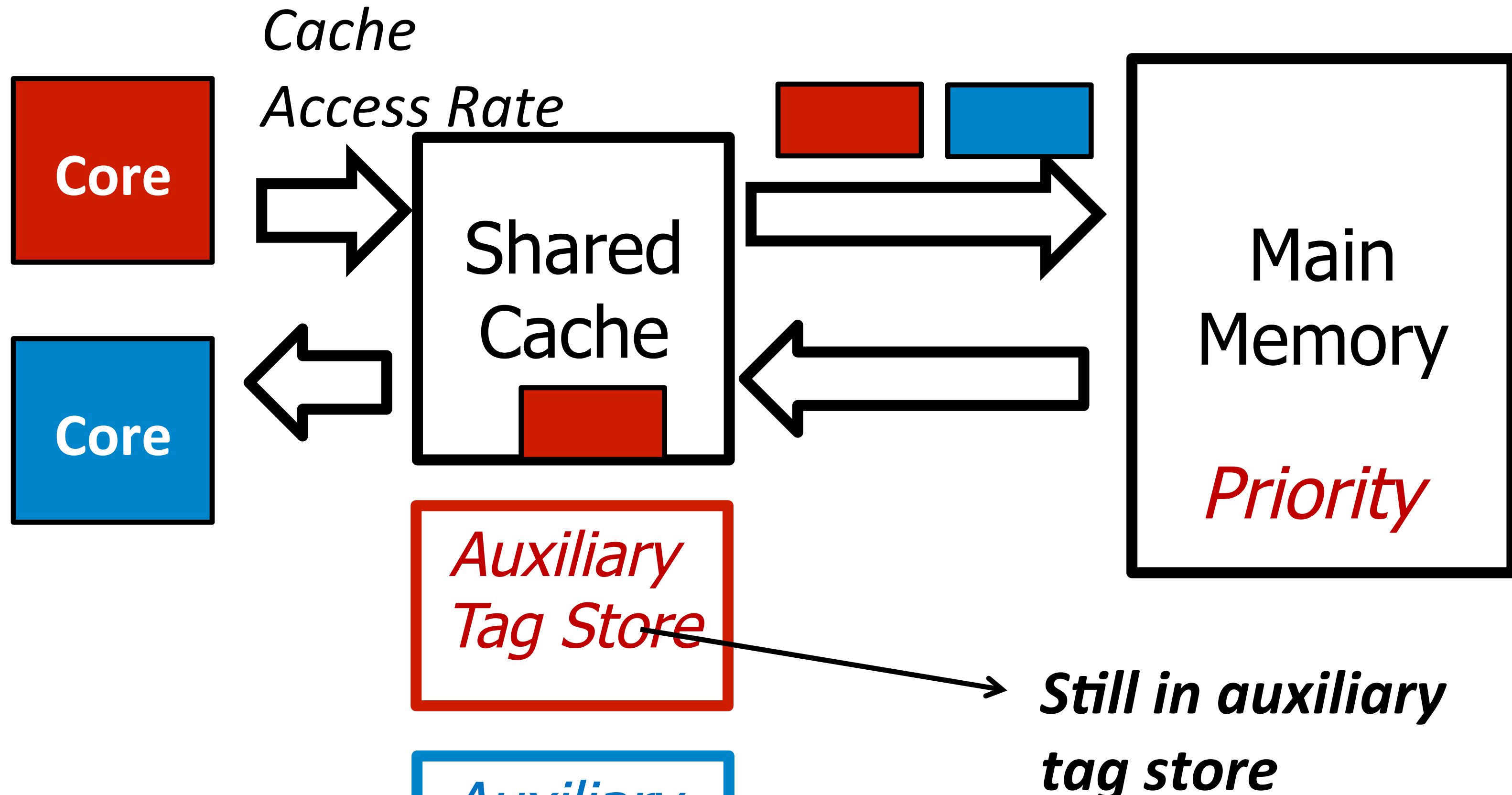
Real System Studies: Cache Access Rate vs. Slowdown



Challenge

How to estimate alone cache access rate?

Auxiliary Tag Store



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

Revisiting Request Service Rate Alone

- Revisiting alone memory request service rate

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

Cycles serving contention misses are not high priority cycles

Cache Access Rate Alone

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

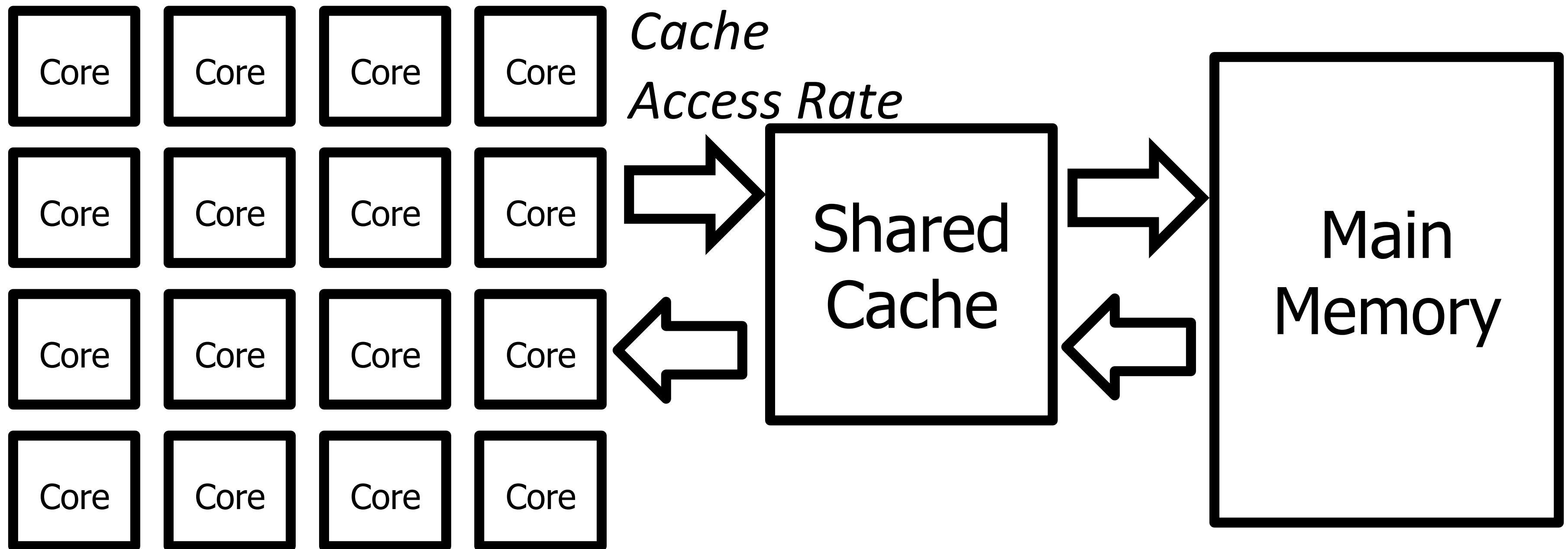
Cache Contention Cycles: Cycles spent serving contention misses

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

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:

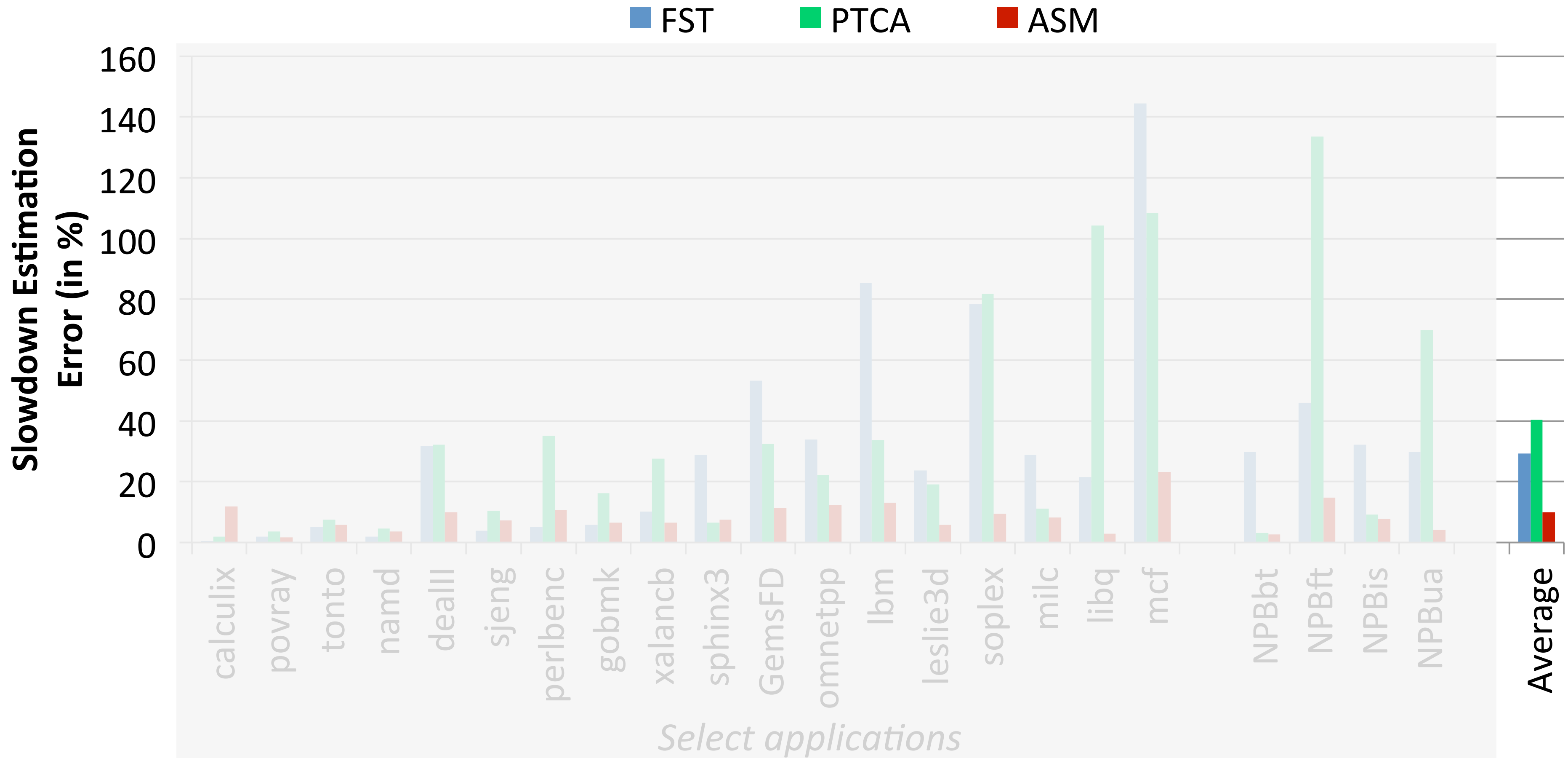
$$\text{Slowdown} = \frac{\text{Execution Time}_{\text{Alone}}}{\text{Execution Time}_{\text{Shared}}}$$

Difficult

Easy

Count number of cycles application receives interference

Model Accuracy Results



Average error of ASM's slowdown estimates: 10%

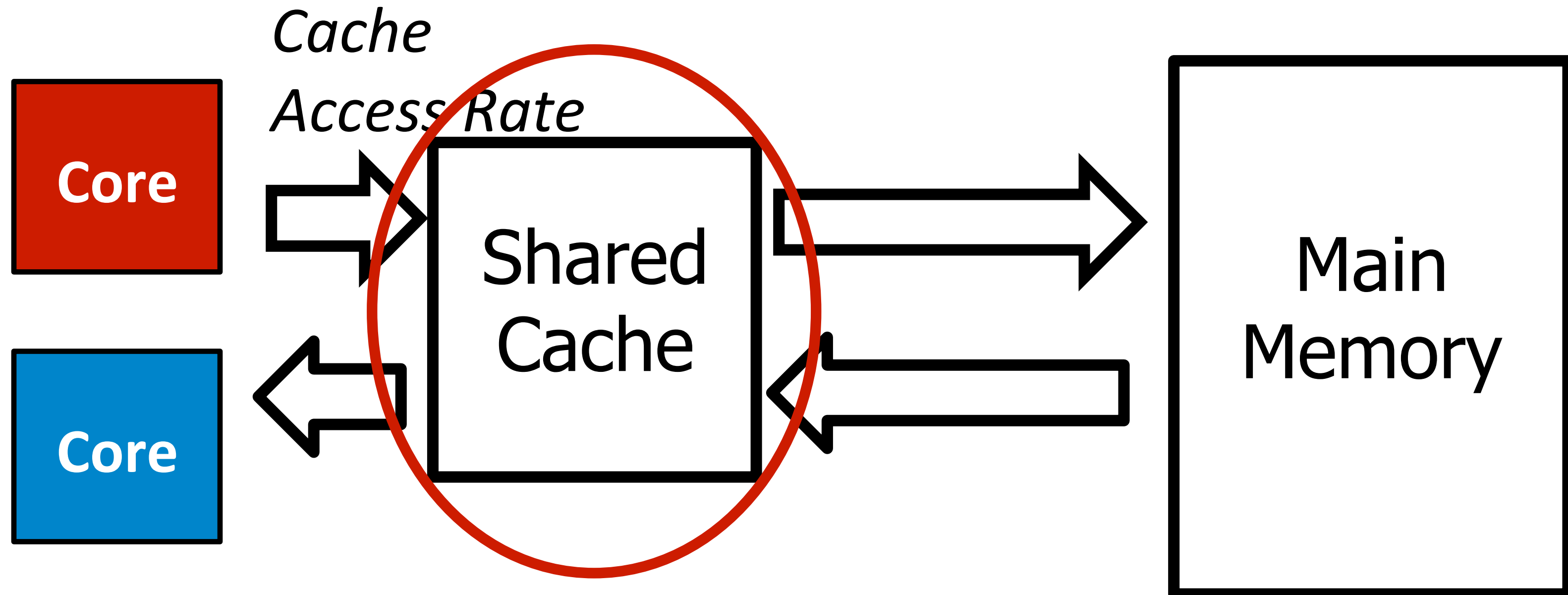
Leveraging Slowdown Estimates for Performance Optimization

- How do we leverage slowdown estimates from our model?

- To achieve high performance
  – Slowdown-aware cache allocation
  – Slowdown-aware bandwidth allocation

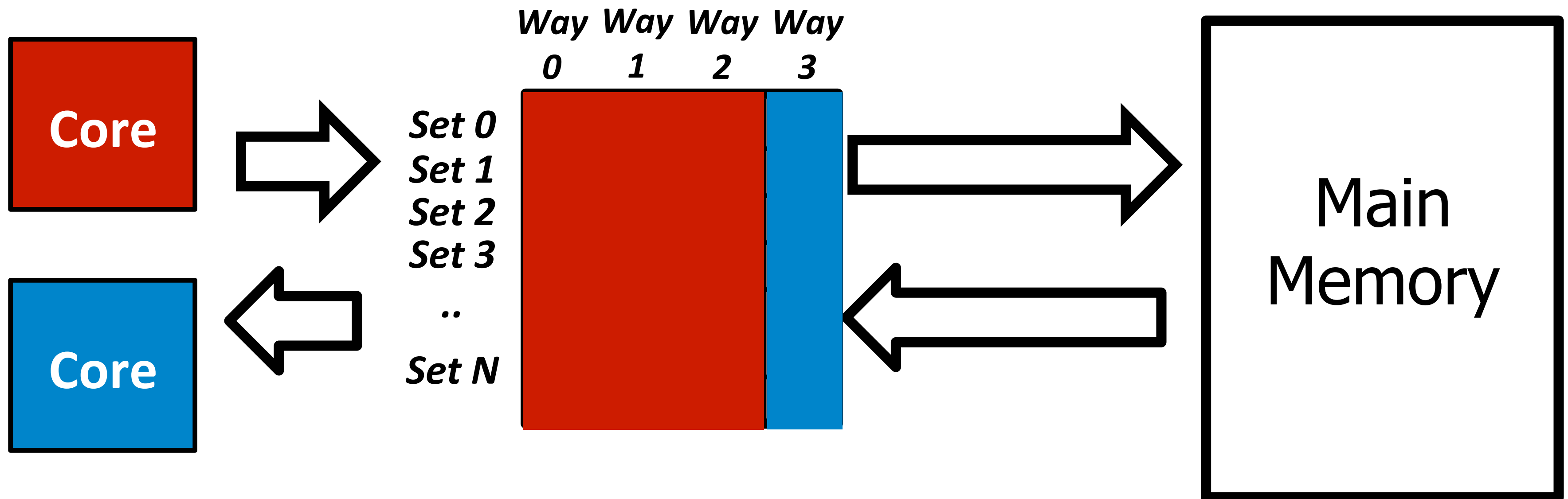
- To achieve performance predictability?

Cache Capacity Partitioning



Goal: Partition the shared cache among applications to mitigate contention

Cache Capacity Partitioning



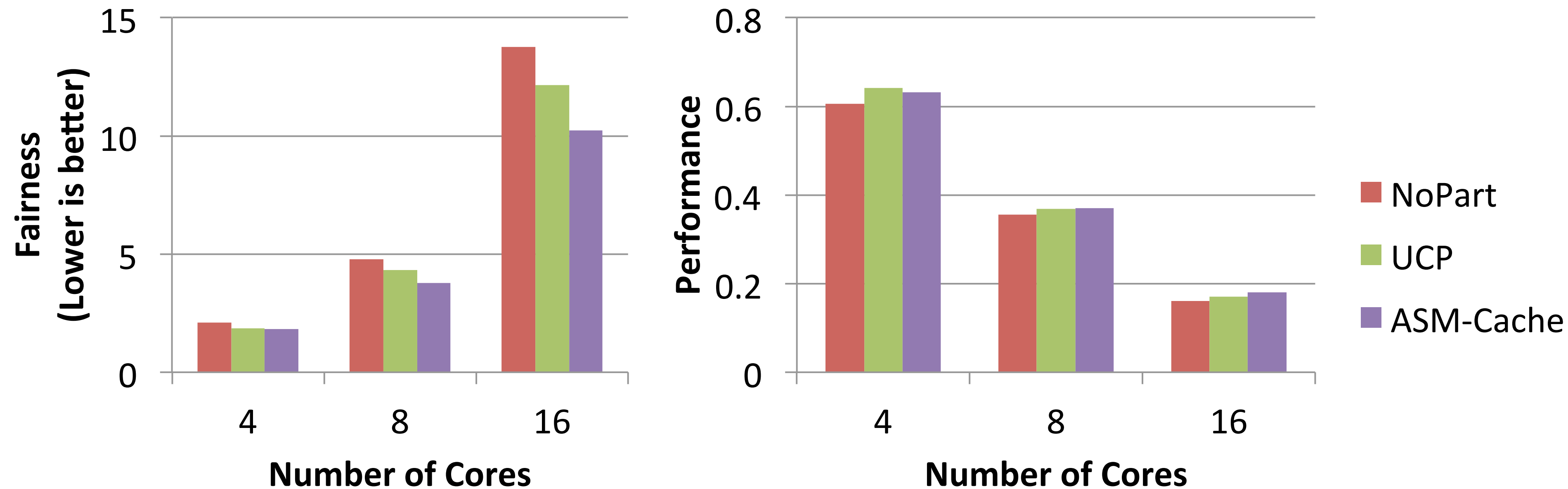
Previous way partitioning schemes optimize for miss count

Problem: Not aware of performance and slowdowns

ASM-Cache: Slowdown-aware Cache Way Partitioning

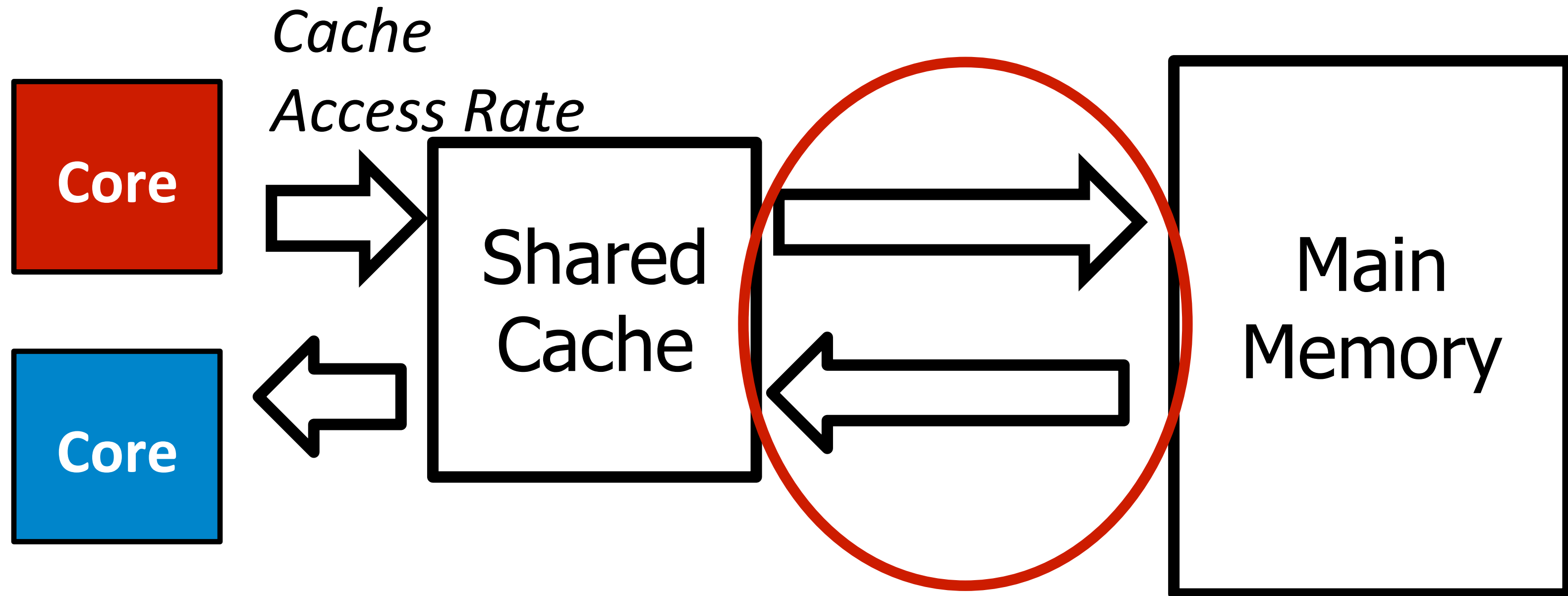
- *Key Requirement: Slowdown estimates for all possible way partitions*
- *Extend ASM to estimate slowdown for all possible cache way allocations*
- *Key Idea: Allocate each way to the application whose slowdown reduces the most*

Performance and Fairness Results



Significant fairness benefits across different systems

Memory Bandwidth Partitioning



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

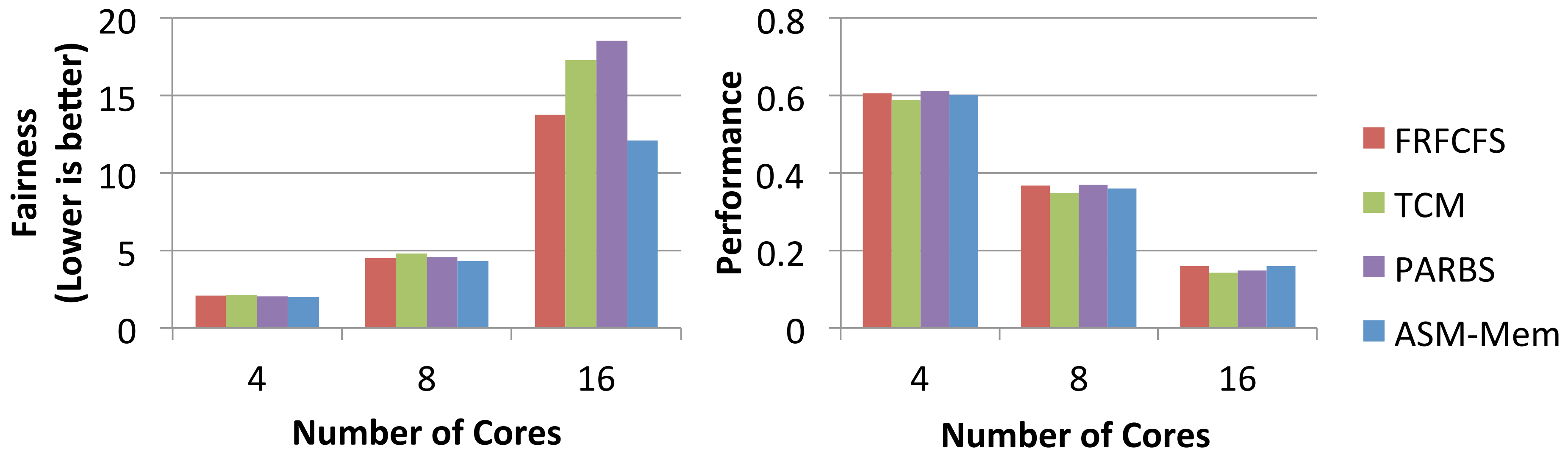
ASM-Mem: Slowdown-aware Memory Bandwidth Partitioning

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

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

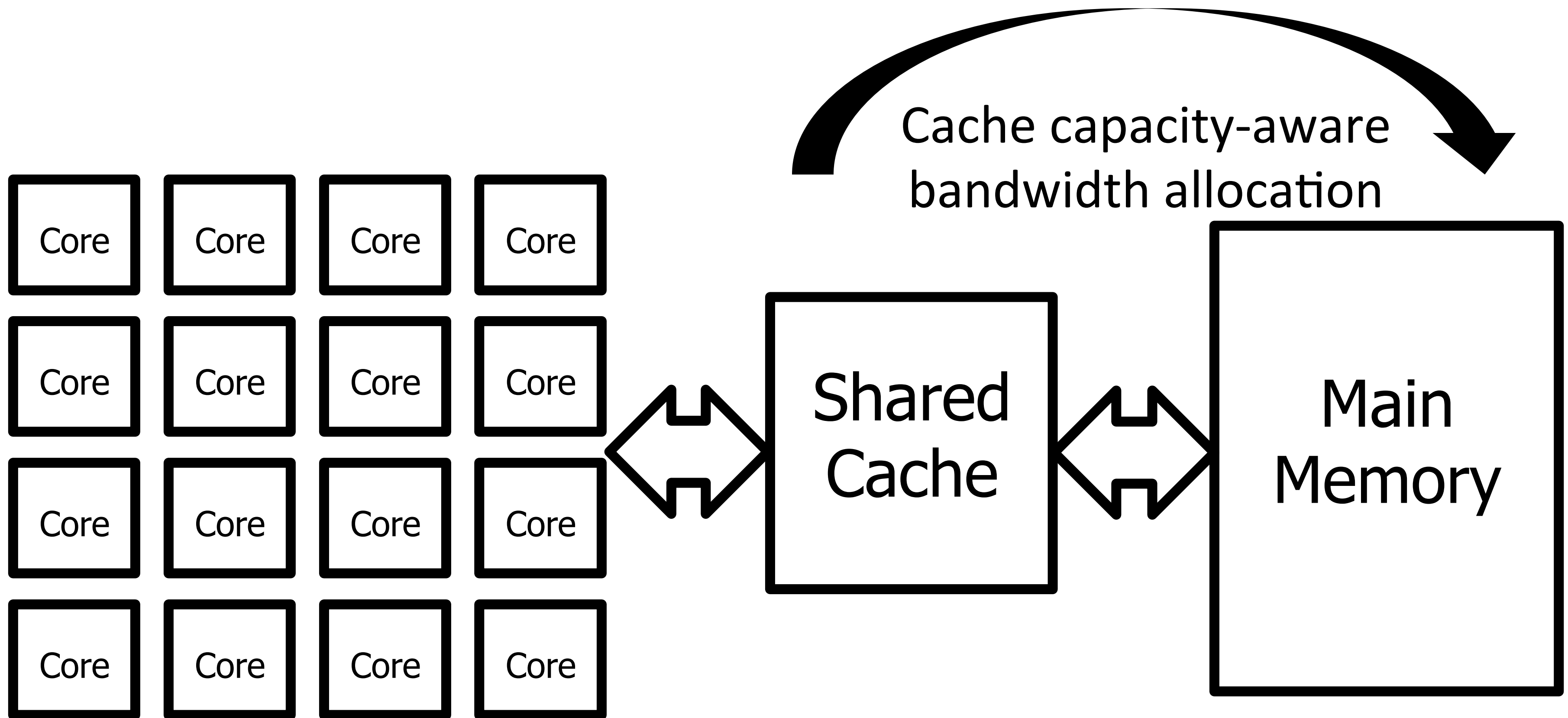
- *Application i's requests given highest priority at the memory controller for its fraction*

ASM-Mem: Fairness and Performance Results



Significant fairness benefits across different systems

Coordinated Resource Allocation Schemes

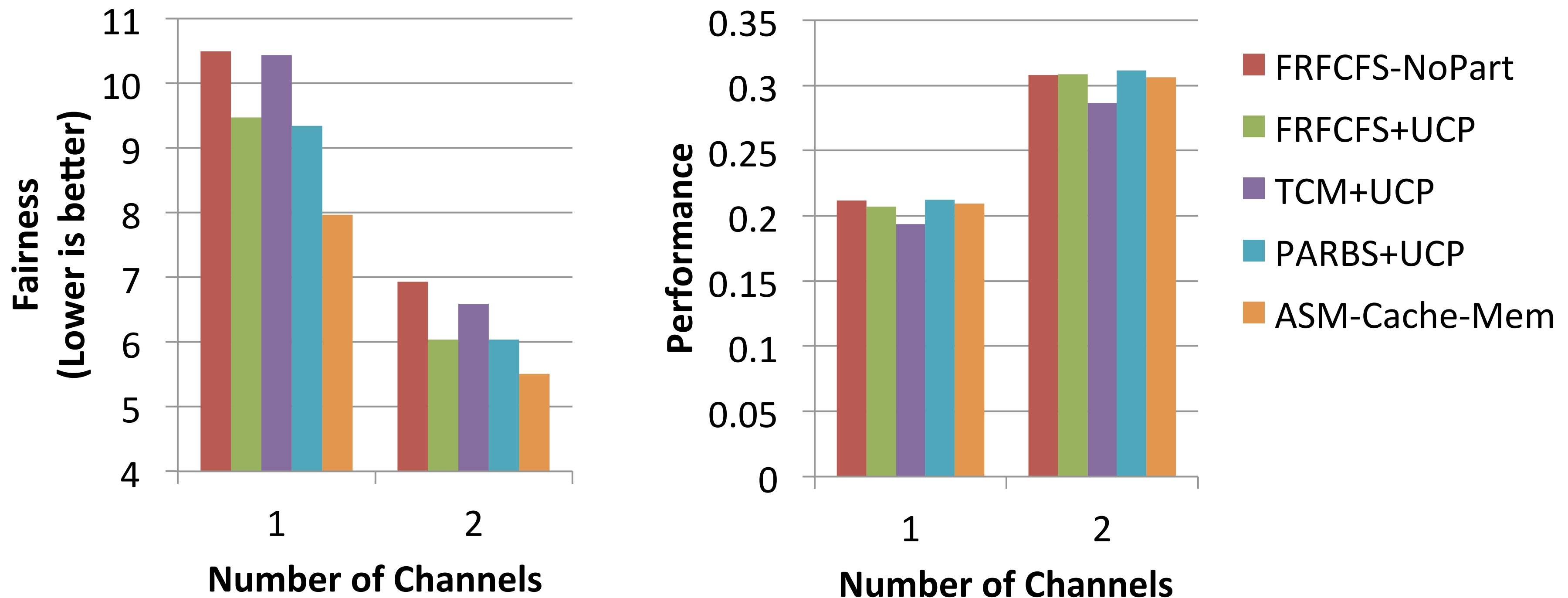


1. Employ ASM-Cache to partition cache capacity

2. Drive ASM-Mem with slowdowns from ASM-Cache

Fairness and Performance Results

16-core system



Significant fairness benefits across different channel counts

Other Possible Applications

- *VM migration and admission control schemes* ***[VEE '15]***
- *Fair billing schemes in a commodity cloud*
- *Bounding application slowdowns*

Summary: Predictability in the Presence of Shared Cache Interference

- Key Ideas:
  - Cache access rate is a proxy for performance
  - Auxiliary tag stores and high priority can be combined to estimate slowdowns

- Key Result: Slowdown estimation error - ~10%

- Some Applications:
  - Slowdown-aware cache partitioning
  - Slowdown-aware memory bandwidth partitioning
  - Many more possible

Future Work: Coordinated Resource Management for Predictable Performance

Goal: Cache capacity and memory bandwidth allocation for an application to meet a bound

Challenges:

- Large search space of potential cache capacity and memory bandwidth allocations
- Multiple possible combinations of cache/memory allocations for each application

18-447
Computer Architecture
Lecture 31: Predictable Performance

Lavanya Subramanian
Carnegie Mellon University
Spring 2015, 4/15/2015