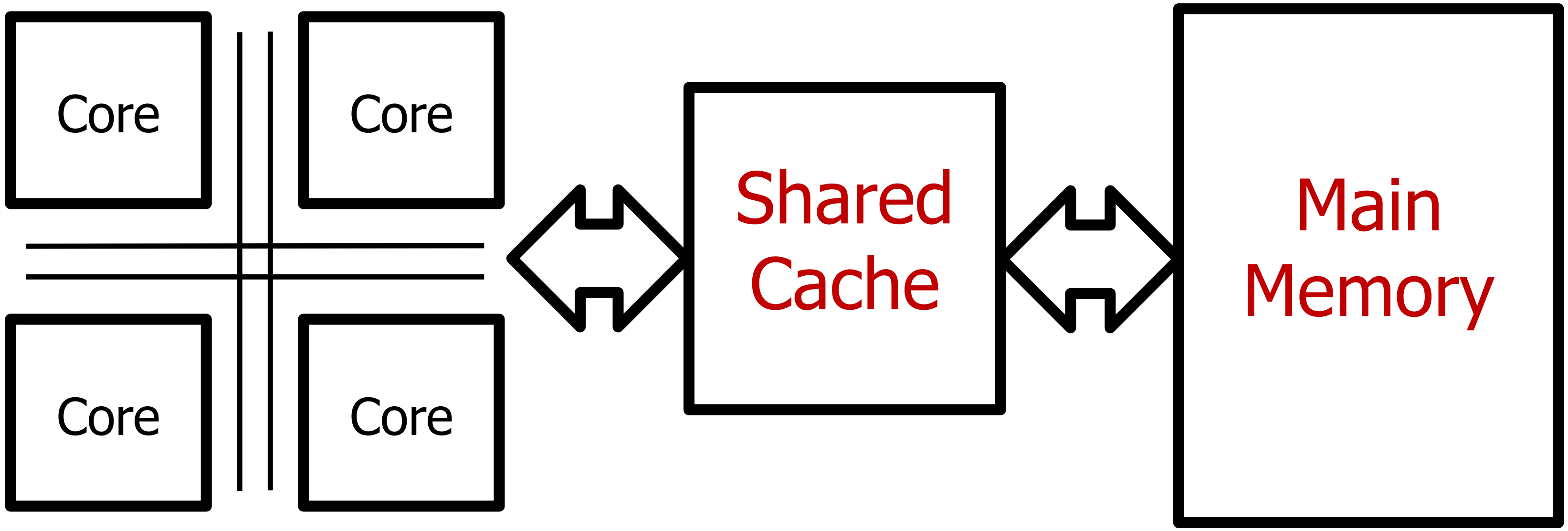
Application Slowdown Model

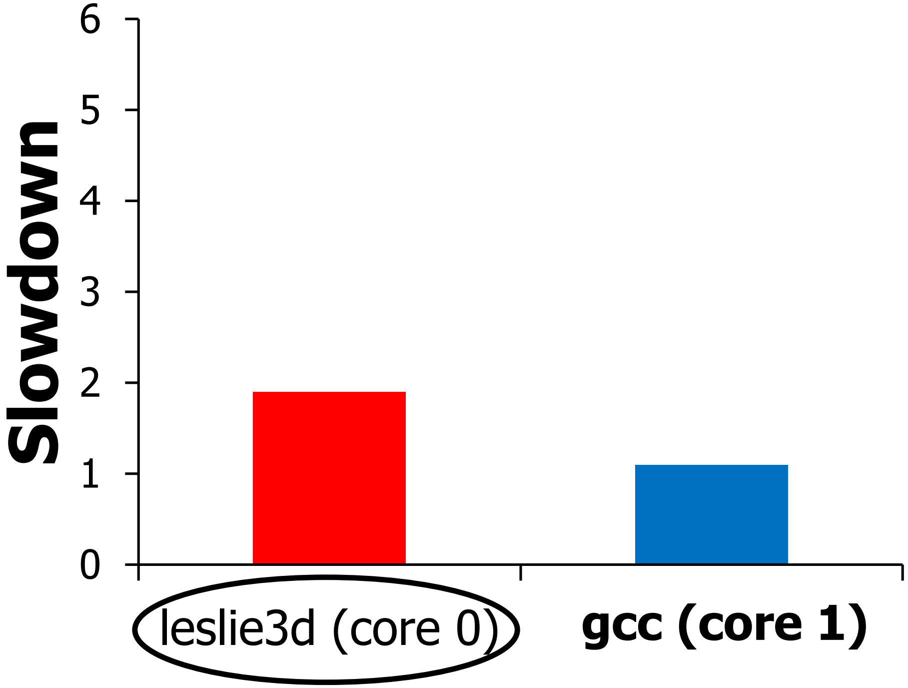
Quantifying and Controlling Impact of Interference at Shared Caches and Main Memory

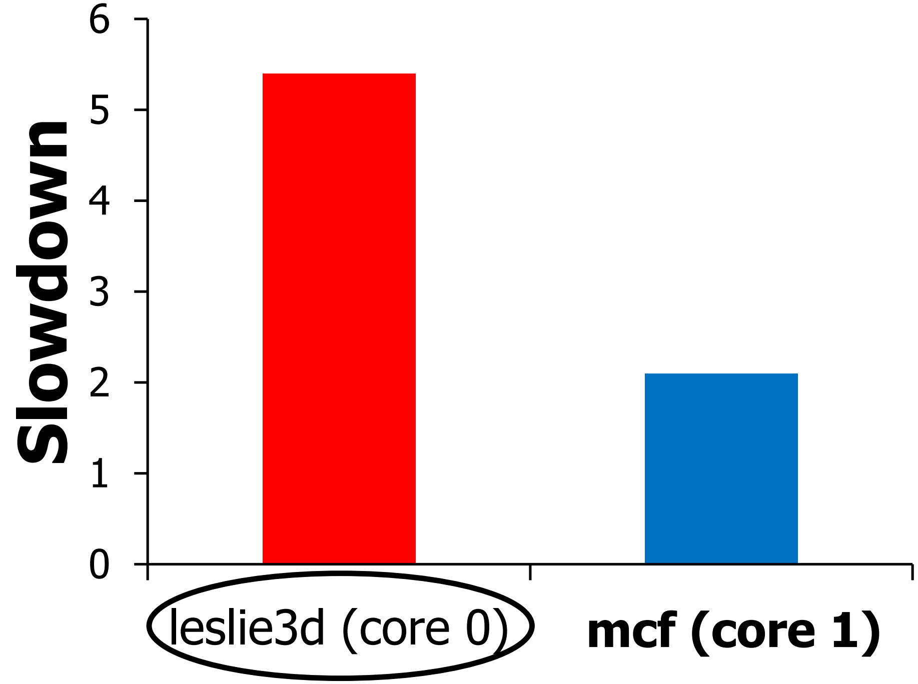
Lavanya Subramanian, Vivek Seshadri,
Arnab Ghosh, Samira Khan, Onur Mutlu

Problem: Interference at Shared Resources



Impact of Shared Resource Interference



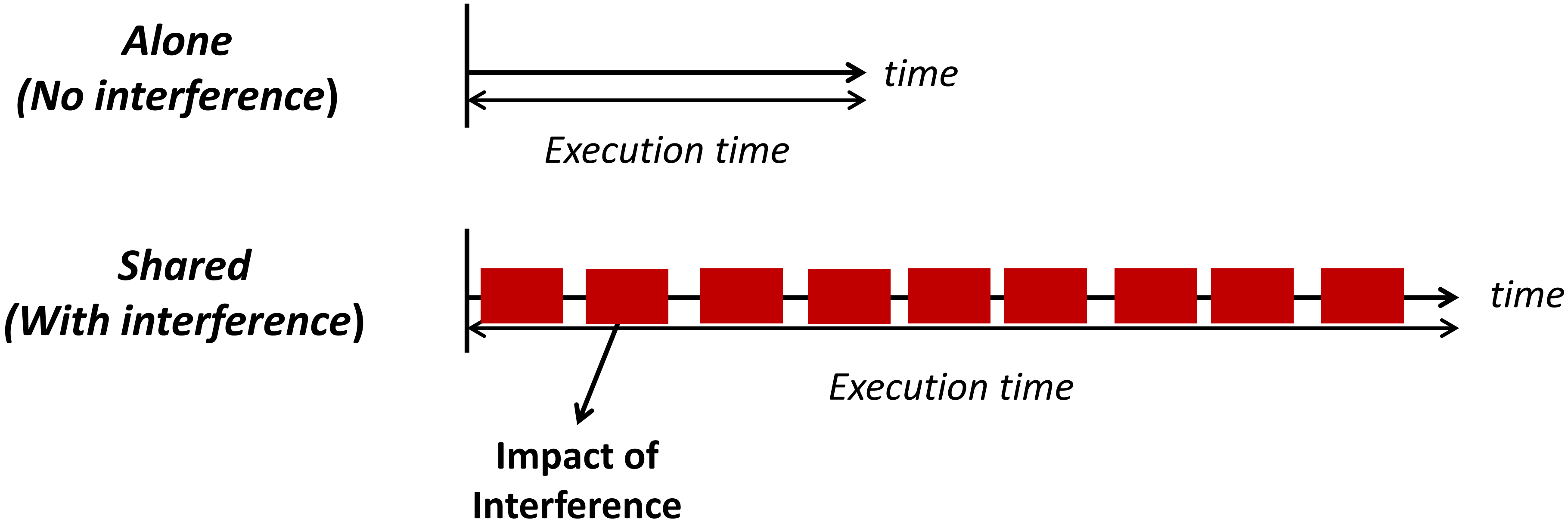


Our Goal: Achieve High and Predictable Performance

Outline

1. Quantify Impact of Interference - *Slowdown*
   - Key Observation
   - Estimating Cache Access Rate $_{\text{Alone}}$
   - ASM: Putting it All Together
   - Evaluation
2. Control Slowdown
   - Slowdown-aware Cache Capacity Allocation
   - Slowdown-aware Memory Bandwidth Allocation
   - Coordinated Cache/Memory Management

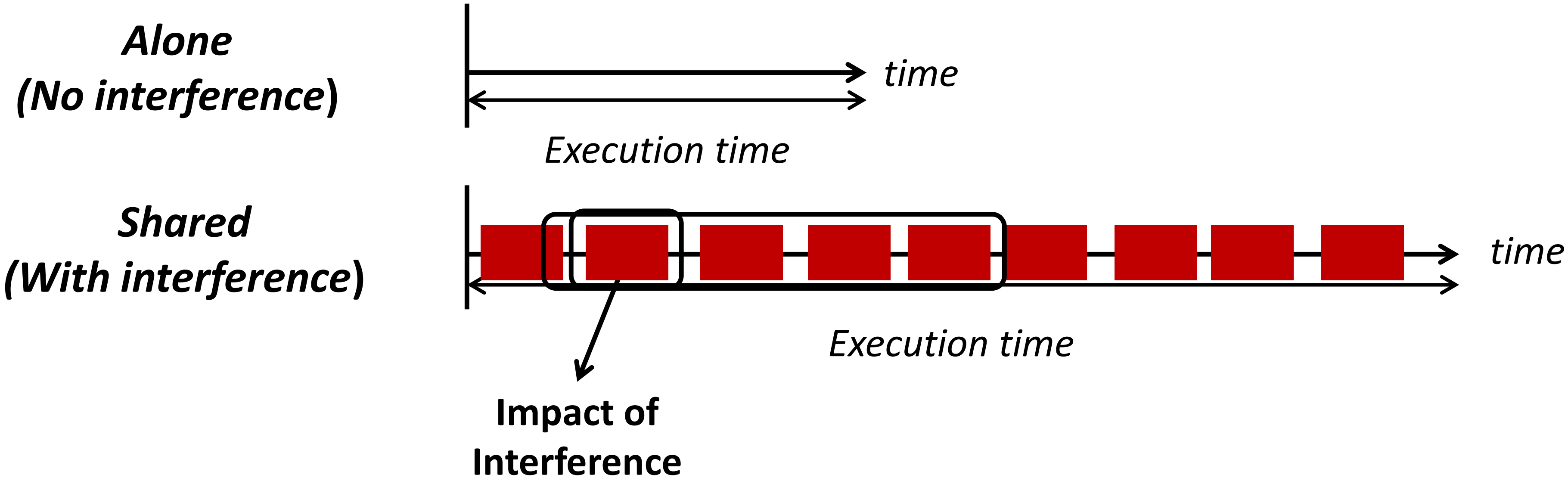
Quantifying Impact of Shared Resource Interference



Slowdown: Definition

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

Approach: Impact of Interference on Performance



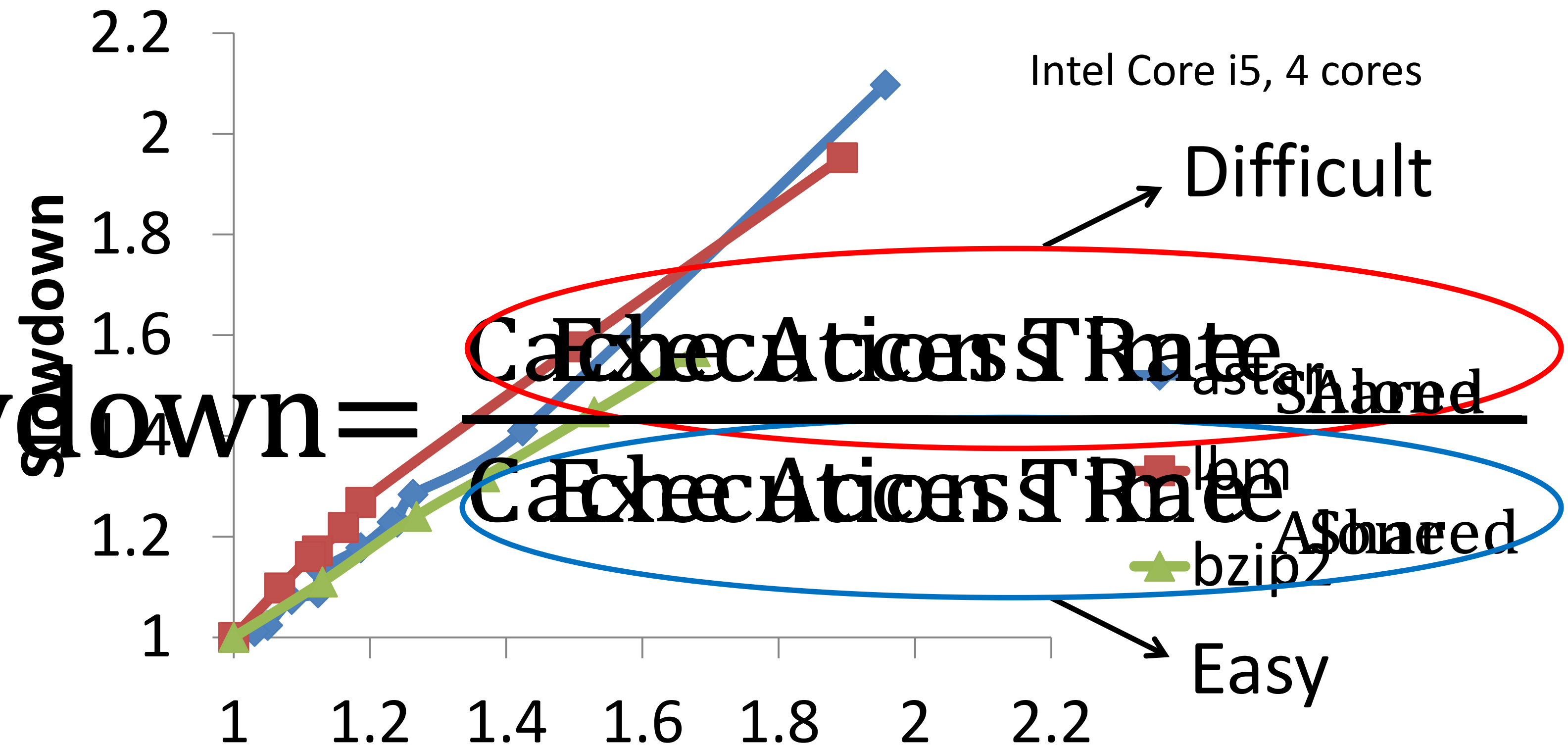
Our Approach: Estimate impact of interference aggregated over requests

Outline

1. **Quantify Slowdown**
   - Key Observation
   - Estimating Cache Access Rate $_{\text{Alone}}$
   - ASM: Putting it All Together
   - Evaluation
2. Control Slowdown
   - Slowdown-aware Cache Capacity Allocation
   - Slowdown-aware Memory Bandwidth Allocation
   - Coordinated Cache/Memory Management

Observation: Shared Cache Access Rate is a Proxy for Performance

Performance ∝ Shared Cache Access rate



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

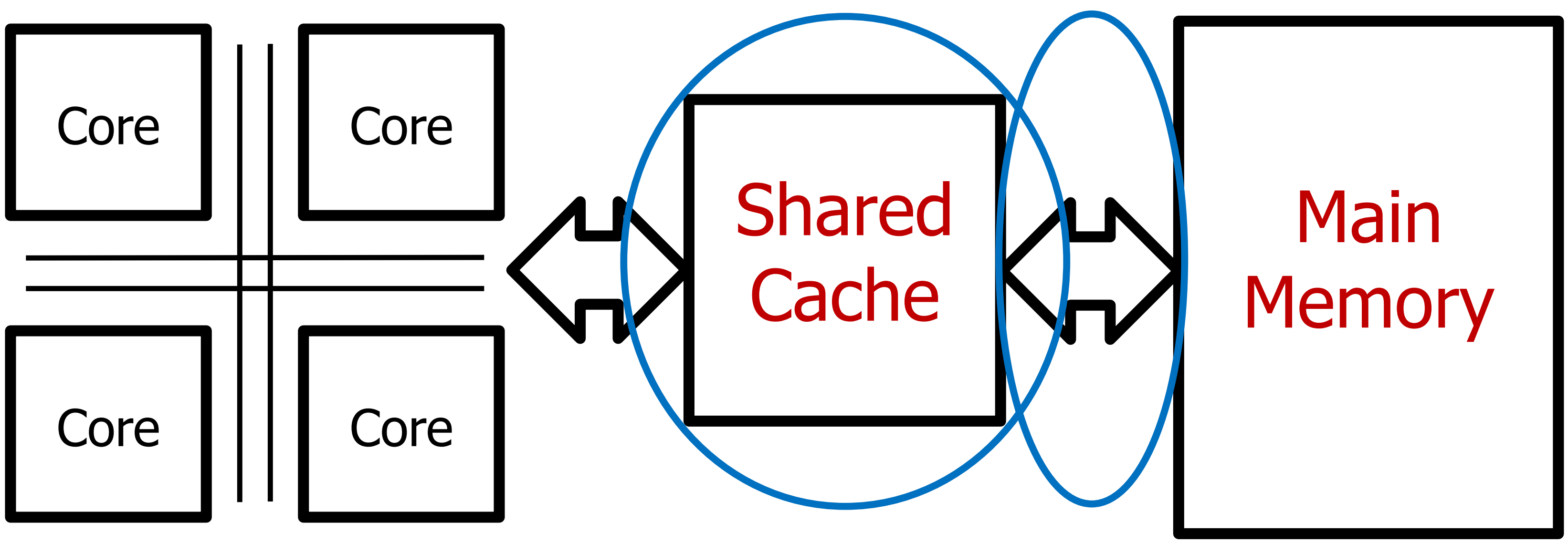
$$\text{Slowdown} = \frac{\text{Execution Time}_{\text{Shared}}}{\text{Execution Time}_{\text{Alone}}}$$

$$\frac{\textbf{Shared Cache Access Rate}_{\textbf{Alone}}}{\textbf{Shared Cache Access Rate}_{\textbf{Shared}}}$$

Outline

1. **Quantify Slowdown**
   - Key Observation
   - Estimating Cache Access Rate $_{\text{Alone}}$
   - ASM: Putting it All Together
   - Evaluation
2. Control Slowdown
   - Slowdown-aware Cache Capacity Allocation
   - Slowdown-aware Memory Bandwidth Allocation
   - Coordinated Cache/Memory Management

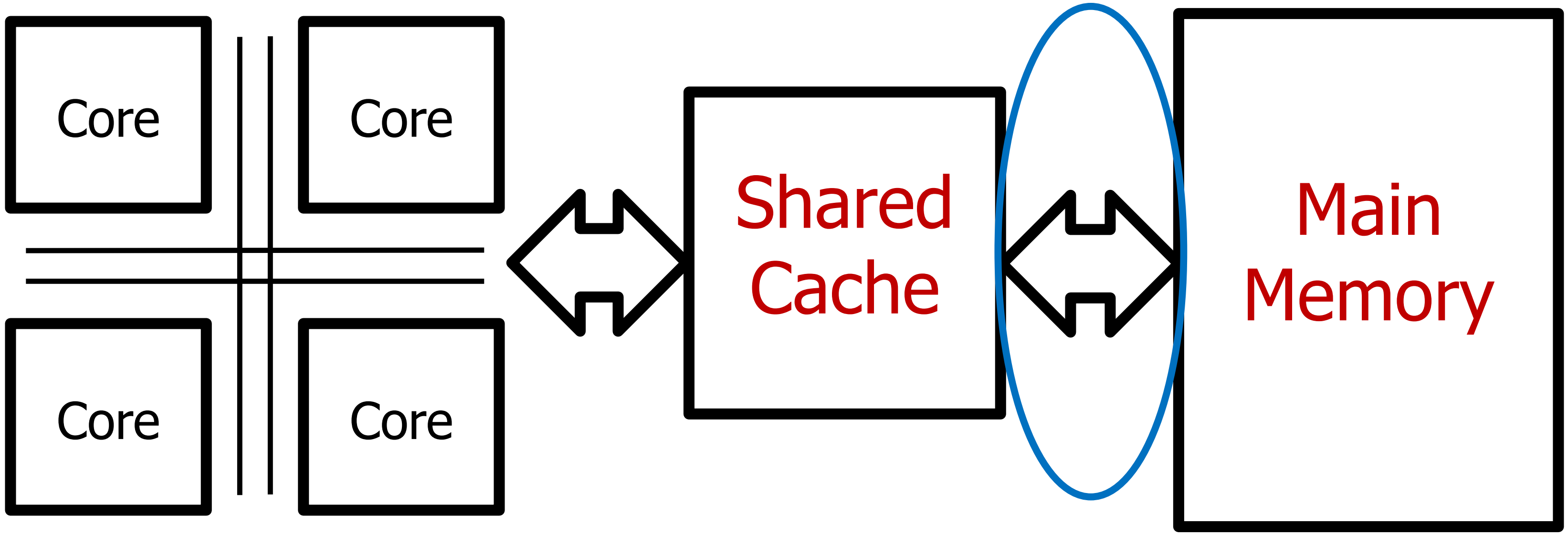
Estimating Cache Access Rate $_{\text{Alone}}$



Challenge 2:
Shared cache capacity interference

Challenge 1:
Main memory bandwidth interference

Estimating Cache Access Rate $_{Alone}$



Challenge 1:
Main memory bandwidth interference

Highest Priority Minimizes Memory Bandwidth Interference

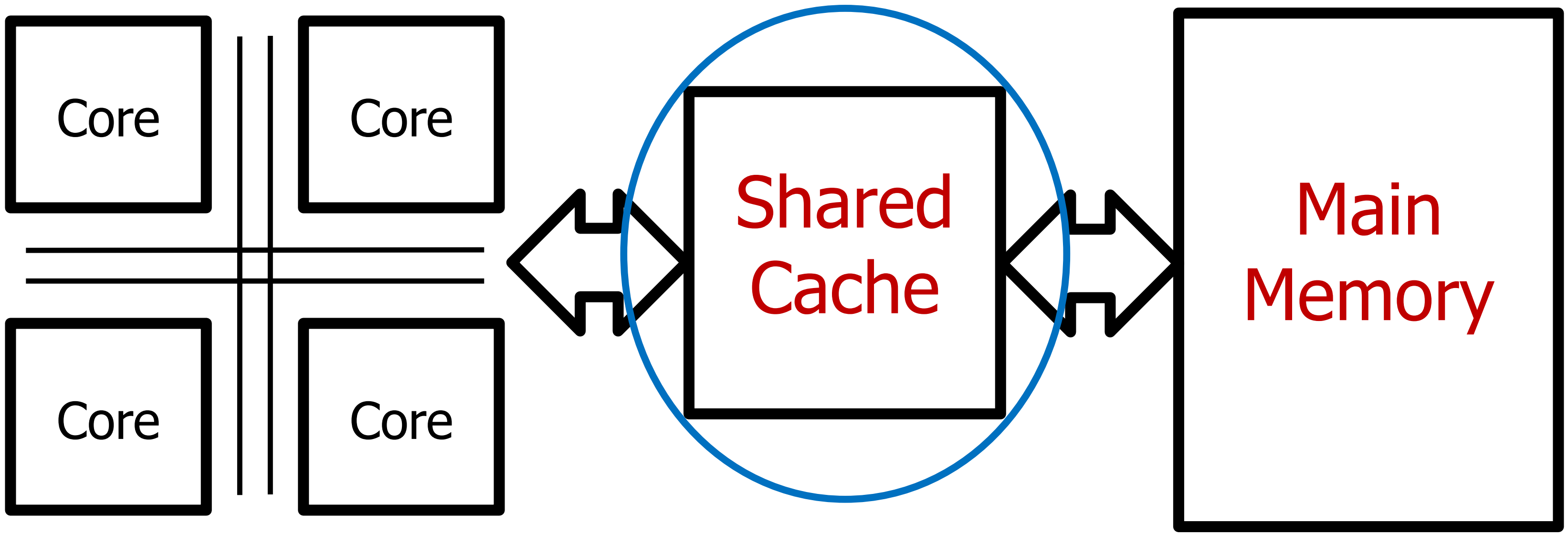
Can minimize impact of main memory interference by giving the application highest priority at the *memory controller*

(Subramanian et al., HPCA 2013)

1. *Highest priority minimizes interference*
2. *Enables estimation of miss service time*

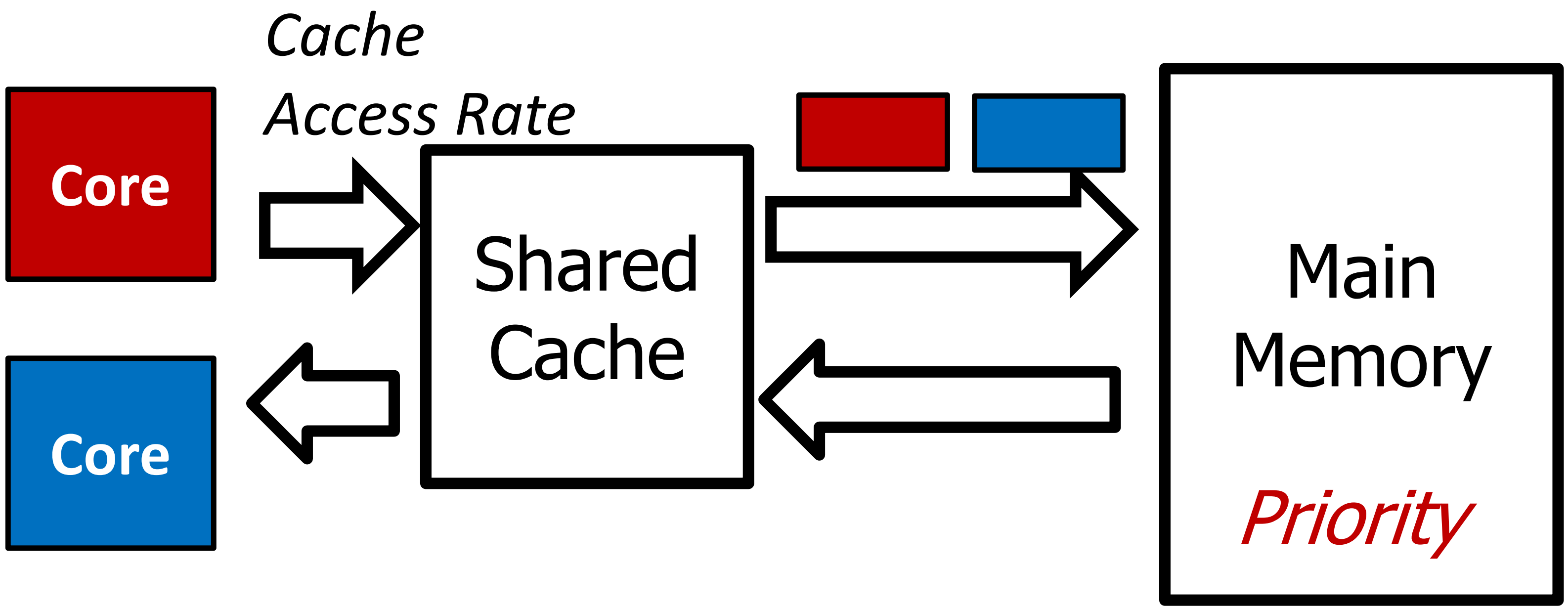
(used to account for shared cache interference)

Estimating Cache Access Rate $_{\text{Alone}}$



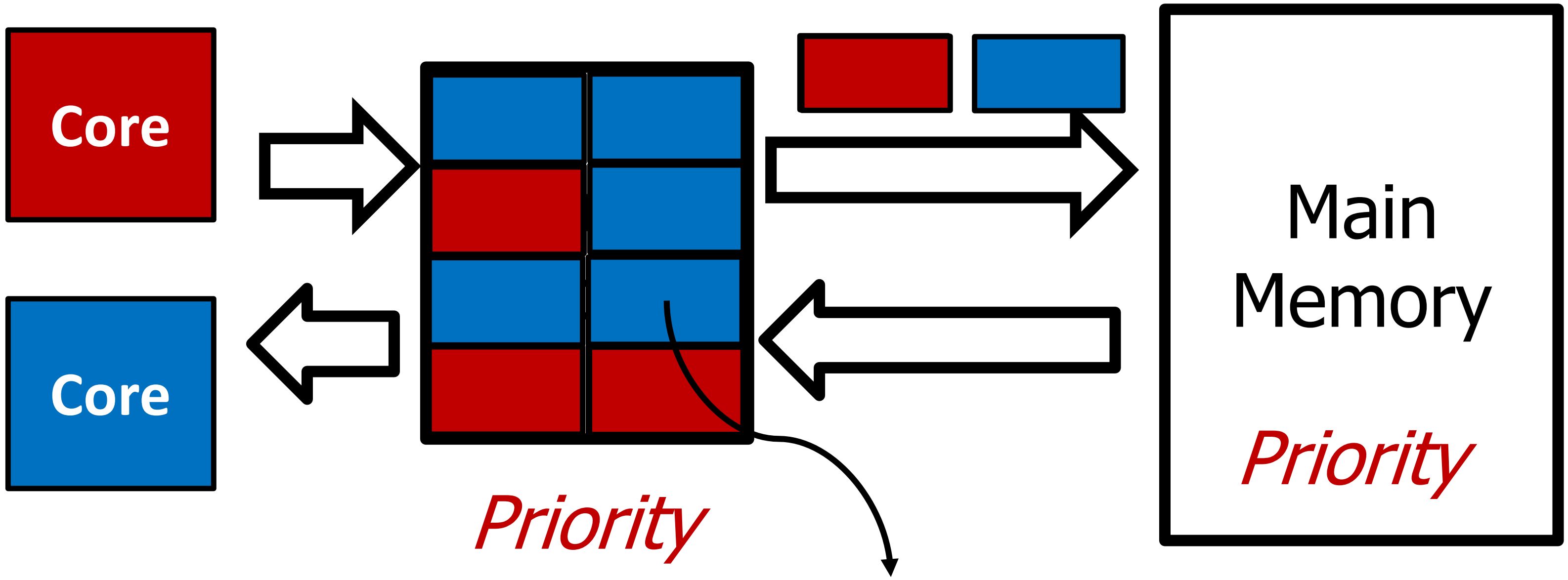
Challenge 2:
Shared cache capacity interference

Cache Capacity Contention



Applications evict each other's blocks from the shared cache

Shared Cache Interference is Hard to Minimize Through Priority



Many blocks of the blue core

Takes effect instantly

Long warmup

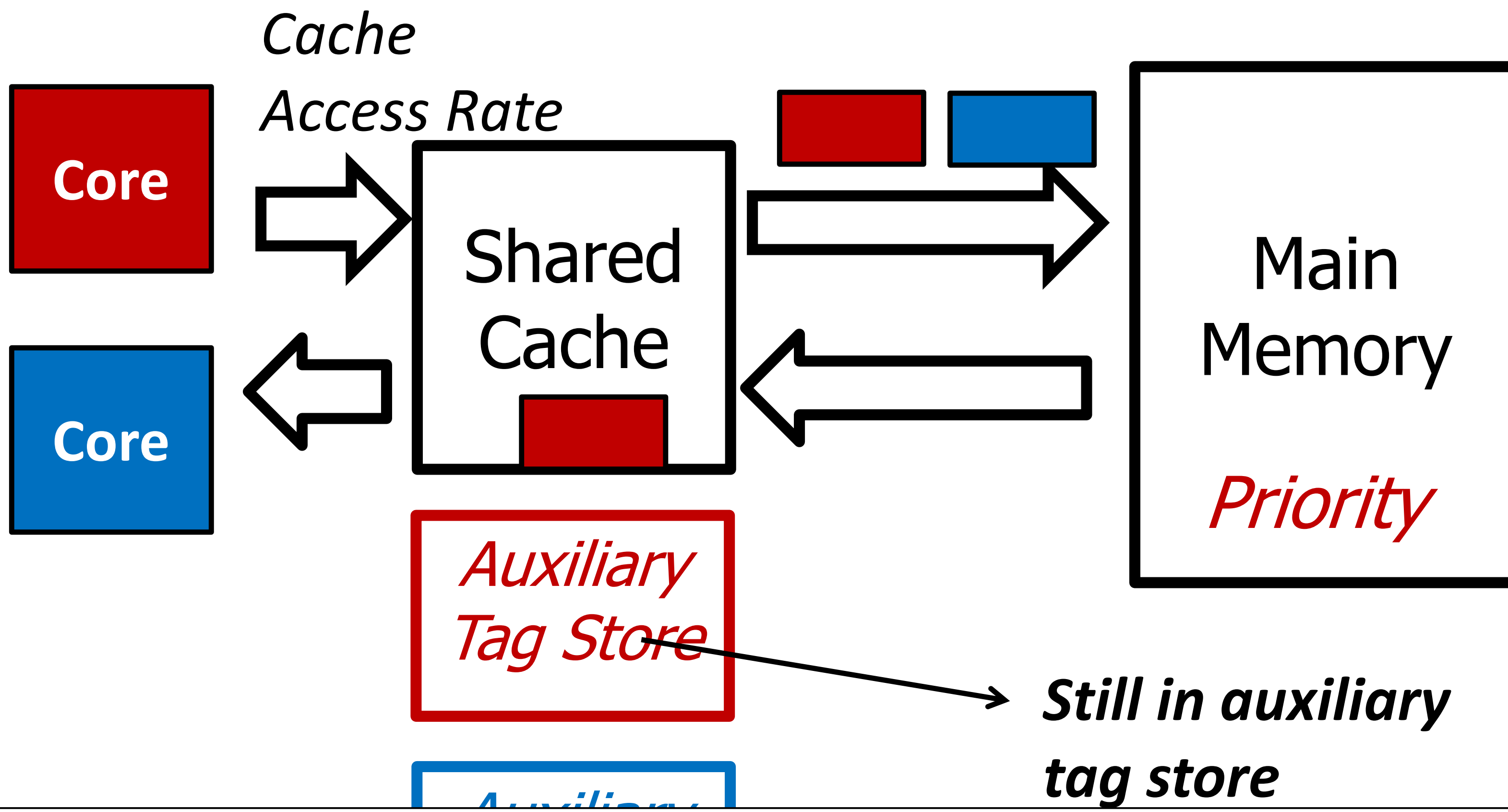
Lots of interference to other applications

Our Approach: Quantify and Remove Cache Interference

1. *Quantify impact* of shared cache interference

2. *Remove impact* of shared cache interference on CAR_{Alone} estimates

1. Quantify Shared Cache Interference



Count number of such ***contention misses***

2. Remove Cycles to Serve Contention Misses from CAR_{Alone} Estimates

Cache Contention Cycles = #Contention Misses x Average Miss Service Time

From auxiliary tag store when given high priority

Measured when application is given high priority

Remove cache contention cycles when estimating

Cache Access Rate $_{Alone}$ (CAR_{Alone})

Accounting for Memory and Shared Cache Interference

- Accounting for memory interference

$$\text{CAR}_{\text{Alone}} = \frac{\text{\# Accesses During High Priority Epochs}}{\text{\# High Priority Cycles}}$$

- Accounting for memory and cache interference

$$\text{CAR}_{\text{Alone}} = \frac{\text{\# Accesses During High Priority Epochs}}{\text{\# High Priority Cycles} - \text{\# Contention Cycles}}$$

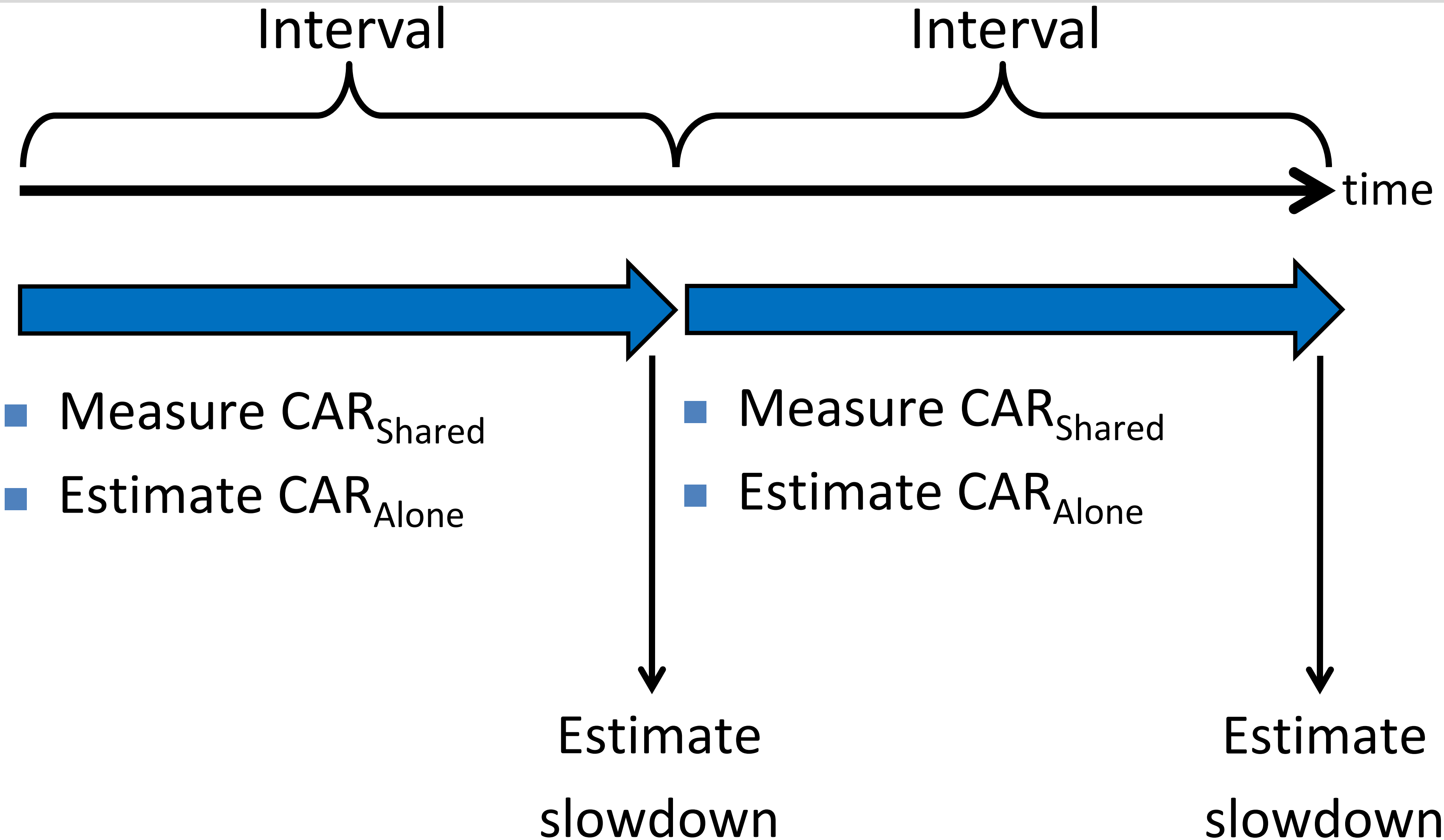
Outline

1. **Quantify Slowdown**
   - Key Observation
   - Estimating Cache Access Rate $_{Alone}$
   - ASM: Putting it All Together
   - Evaluation
2. Control Slowdown
   - Slowdown-aware Cache Capacity Allocation
   - Slowdown-aware Memory Bandwidth Allocation
   - Coordinated Cache/Memory Management

Application Slowdown Model (ASM)

$$\text{Slowdown} = \frac{\text{Cache Access Rate}_{\text{Alone}}\ (\text{CARAl}_{\text{one}})}{\text{Cache Access Rate}_{\text{Shared}}\ (\text{CARSh}_{\text{ared}})}$$

ASM: Interval Based Operation



A More Accurate and Simple Model

- More accurate: Takes into account request overlap behavior
  - Implicit through aggregate estimation of cache access rate and miss service time
  - Unlike prior works that estimate per-request interference

- Simpler hardware: Amenable to set sampling in the auxiliary tag store
  - Need to measure only contention miss count
  - Unlike prior works that need to know if each request is a contention miss or not

Outline

1. **Quantify Slowdown**
   - Key Observation
   - Estimating Cache Access Rate $_{Alone}$
   - ASM: Putting it All Together
   - Evaluation
2. Control Slowdown
   - Slowdown-aware Cache Capacity Allocation
   - Slowdown-aware Memory Bandwidth Allocation

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:

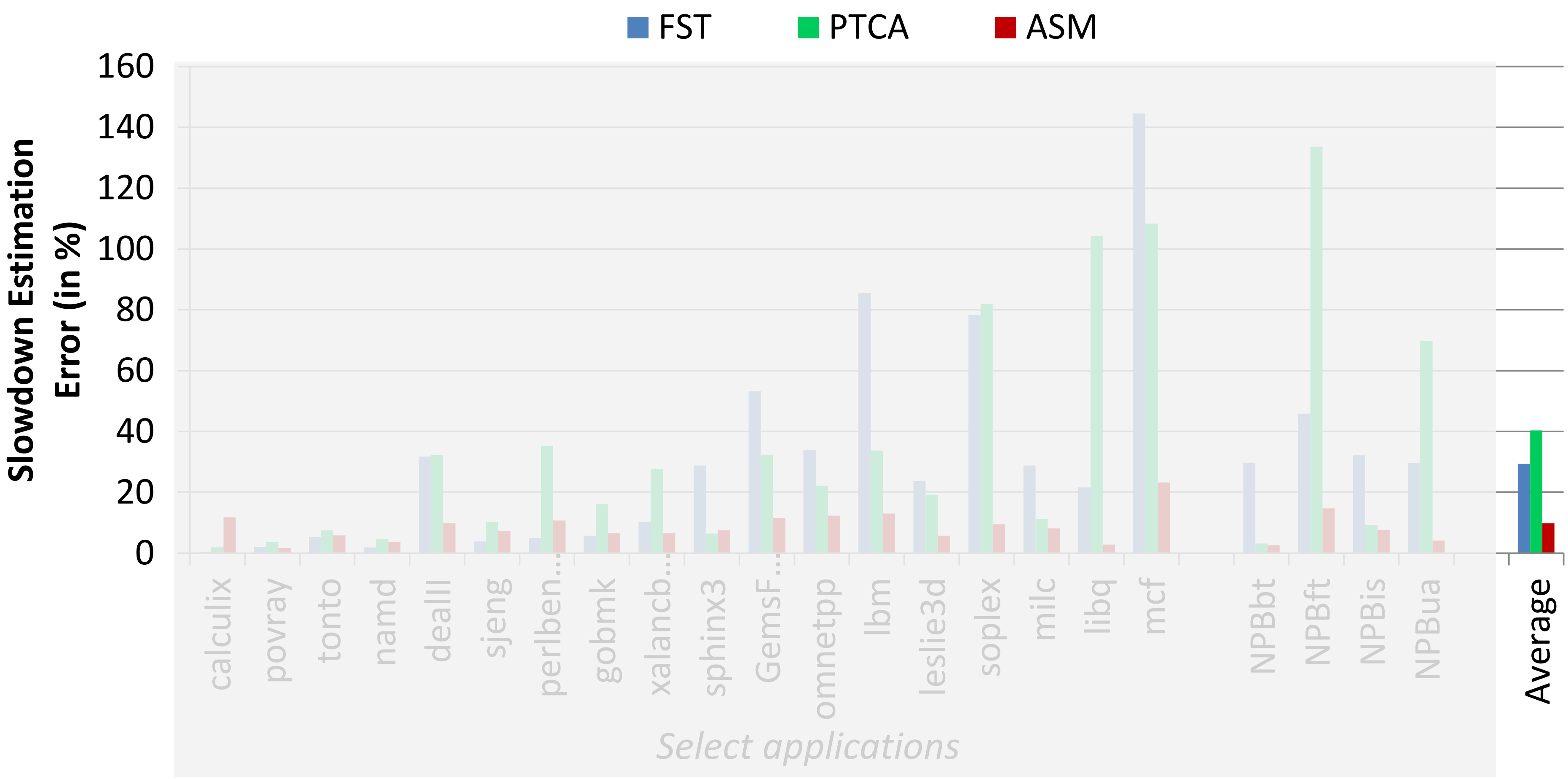
$$\text{Slowdown} = \frac{\text{Execution Time}_{\text{Shared}}}{\text{Execution Time}_{\text{Alone}}}$$

Count interference cycles experienced by each request

Methodology

- Configuration of our simulated system
  - 4 cores
  - 1 channel, 8 banks/channel
  - DDR3 1333 DRAM
  - 2MB shared cache
- Workloads
  - SPEC CPU2006 and NAS
  - 100 multiprogrammed workloads

Model Accuracy Results



Average error of ASM's slowdown estimates: 10%

Previous models have 29%/40% average error

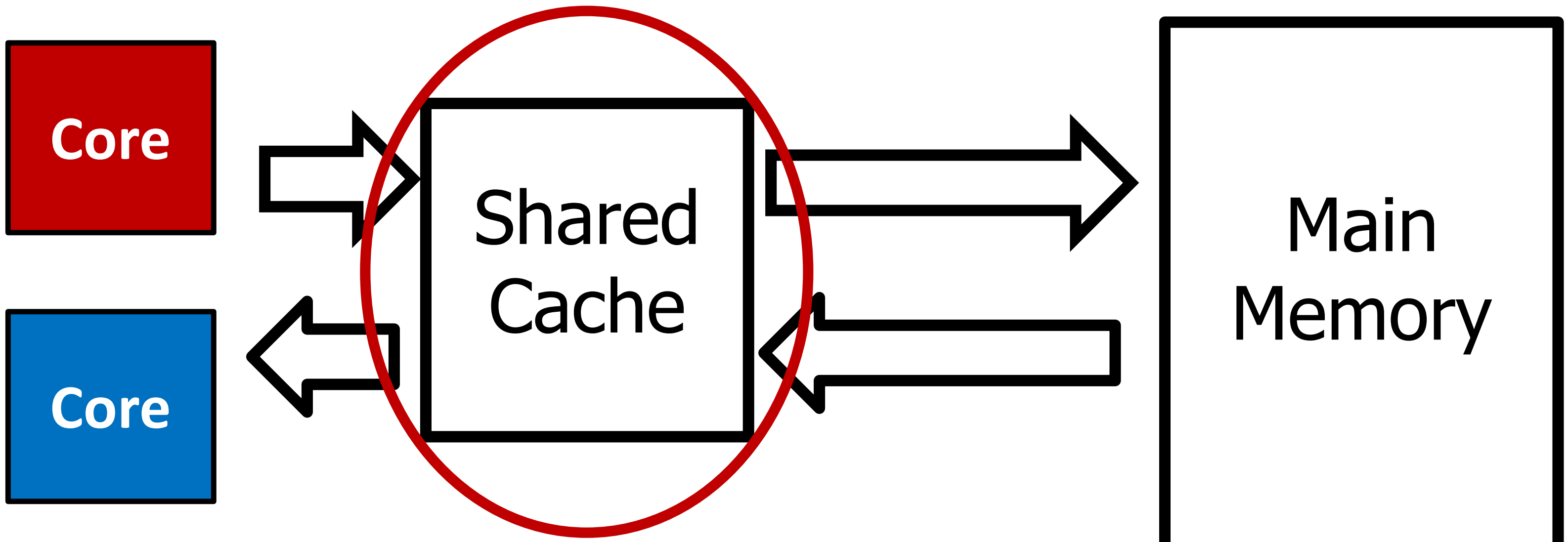
Outline

1. Quantify Slowdown
   - Key Observation
   - Estimating Cache Access Rate $_{\text{Alone}}$
   - ASM: Putting it All Together
   - Evaluation
2. **Control Slowdown**
   - Slowdown-aware Cache Capacity Allocation
   - Slowdown-aware Memory Bandwidth Allocation
   - Coordinated Cache/Memory Management

Outline

1. Quantify Slowdown
   – Key Observation
   – Estimating Cache Access Rate $_{Alone}$
   – ASM: Putting it All Together
   – Evaluation
2. **Control Slowdown**
   – Slowdown-aware Cache Capacity Allocation
   – Slowdown-aware Memory Bandwidth Allocation
   – Coordinated Cache/Memory Management

Cache Capacity Partitioning



Previous partitioning schemes mainly focus on miss count reduction

Problem: Does not directly translate to performance and slowdowns

ASM-Cache: Slowdown-aware Cache Capacity Partitioning

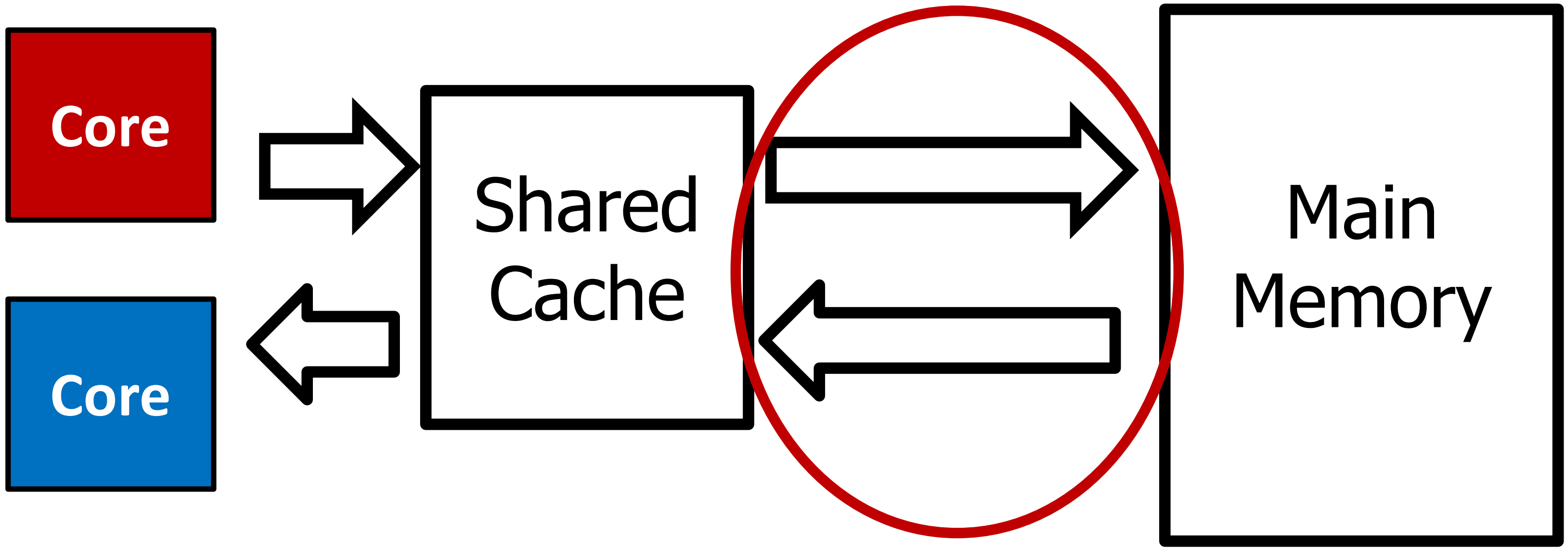
- *Goal: Achieve high fairness and performance through slowdown-aware cache partitioning*

- *Key Idea: Allocate more cache space to applications whose slowdowns reduce the most with more cache space*

Outline

1. Quantify Slowdown
   - Key Observation
   - Estimating Cache Access Rate $_{Alone}$
   - ASM: Putting it All Together
   - Evaluation
2. **Control Slowdown**
   - Slowdown-aware Cache Capacity Allocation
   - Slowdown-aware Memory Bandwidth Allocation
   - Coordinated Cache/Memory Management

Memory Bandwidth Partitioning



Goal: Achieve high fairness and performance through slowdown-aware bandwidth partitioning

ASM-Mem: Slowdown-aware Memory Bandwidth Partitioning

- *Key Idea: Prioritize an application proportionally to its slowdown*

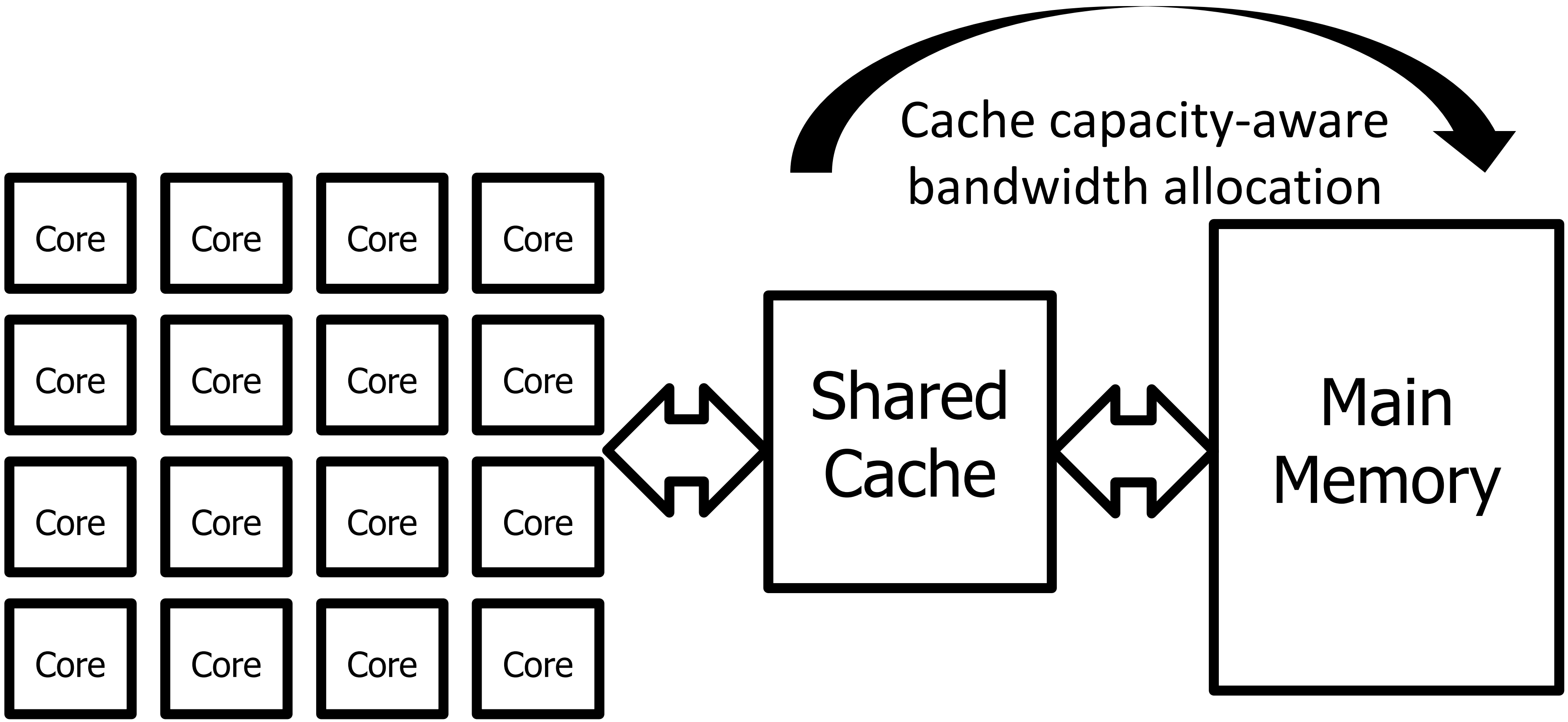
$$\text{High Priority Fraction}_i = \frac{\text{Slowdown}_i}{\sum_j \text{Slowdown}_j}$$

- *Application i's requests prioritized at the memory controller for its fraction*

Outline

1. Quantify Slowdown
   - Key Observation
   - Estimating Cache Access Rate $_{Alone}$
   - ASM: Putting it All Together
   - Evaluation
2. **Control Slowdown**
   - Slowdown-aware Cache Capacity Allocation
   - Slowdown-aware Memory Bandwidth Allocation
   - Coordinated Cache/Memory Management

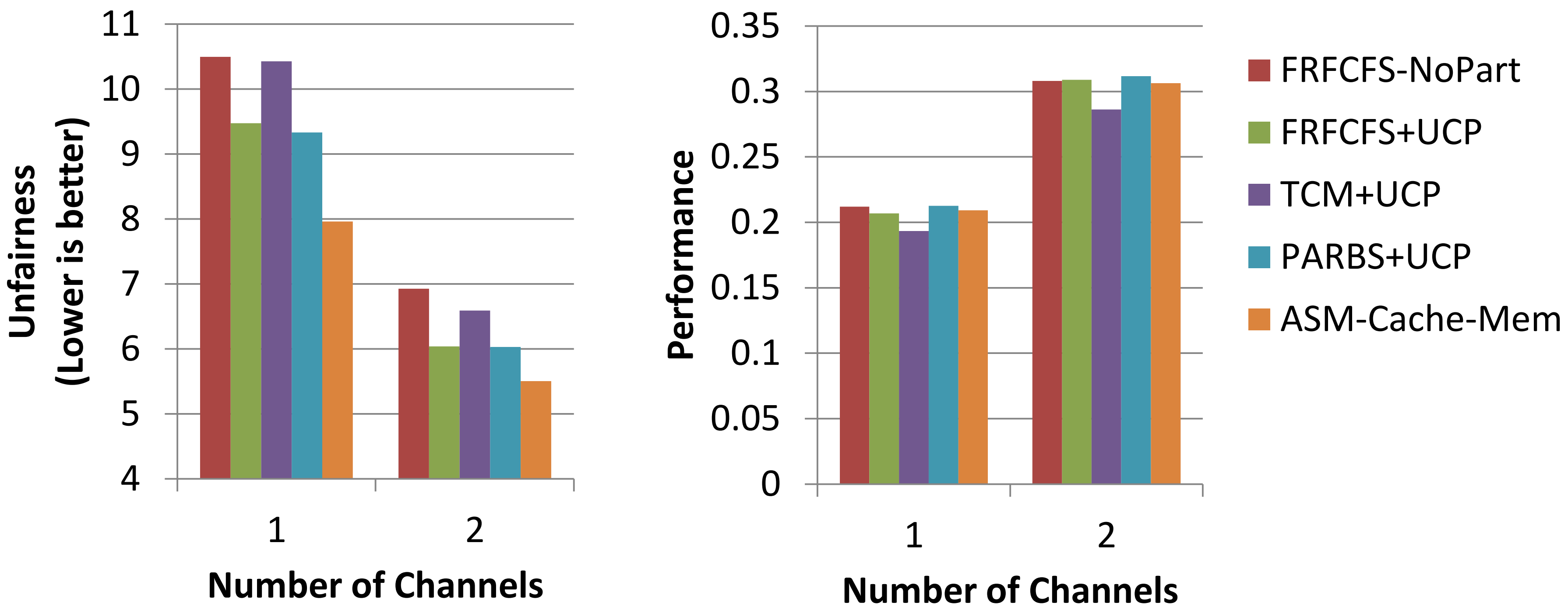
Coordinated Resource Allocation Schemes



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



14%/8% unfairness reduction on 1/2 channel systems compared to PARBS+UCP with similar performance

Other Results in the Paper

- Distribution of slowdown estimation error
- Sensitivity to system parameters
  - Core count, memory channel count, cache size
- Sensitivity to model parameters
- Impact of prefetching
- Case study showing ASM's potential for providing slowdown guarantees

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 Release by January 2016*

Application Slowdown Model

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

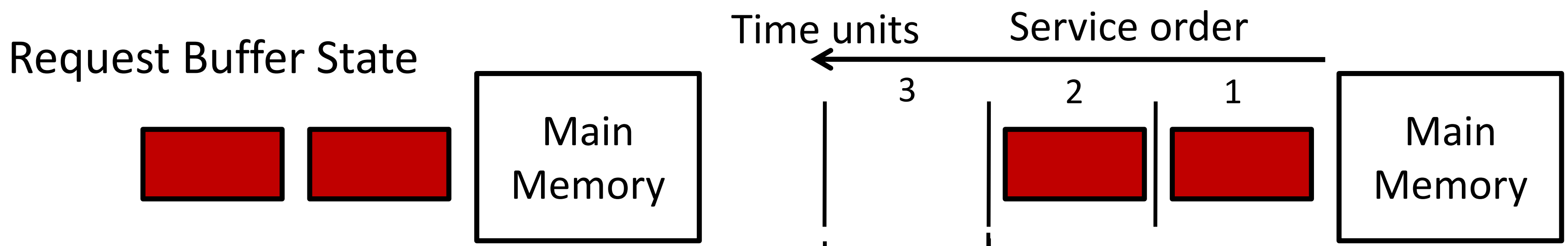
Lavanya Subramanian, Vivek Seshadri,

Arnab Ghosh, Samira Khan, Onur Mutlu

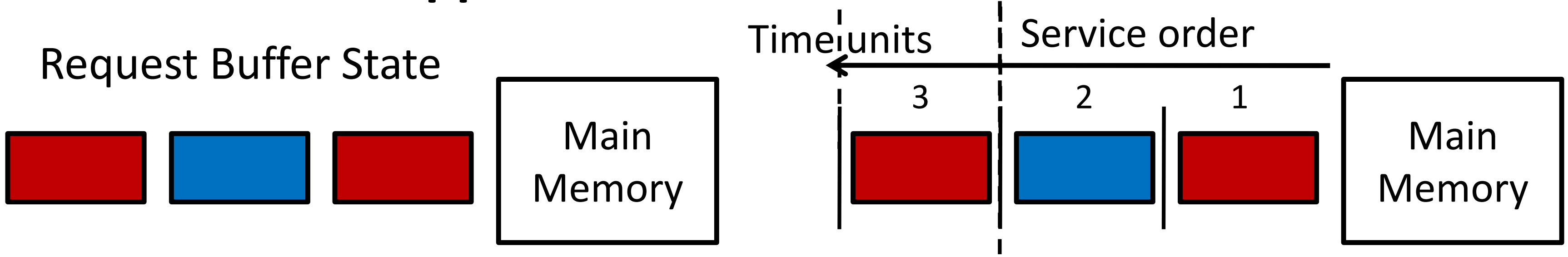
Backup

Highest Priority Minimizes Memory Bandwidth Interference

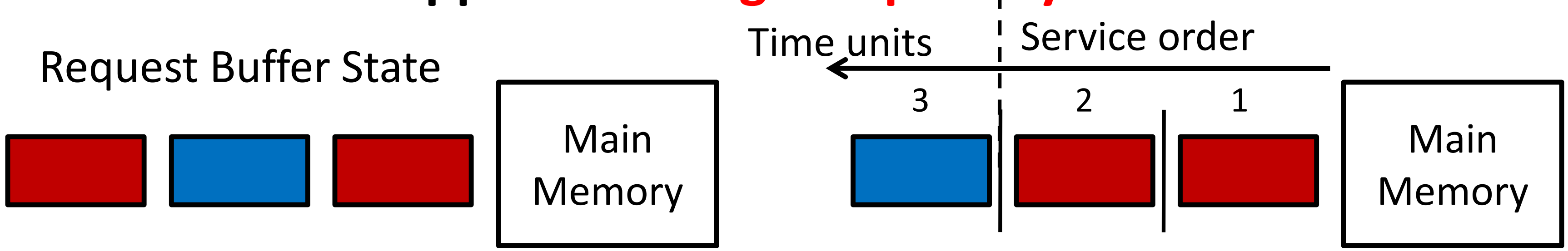
1. Run alone



2. Run with another application



3. Run with another application: highest priority

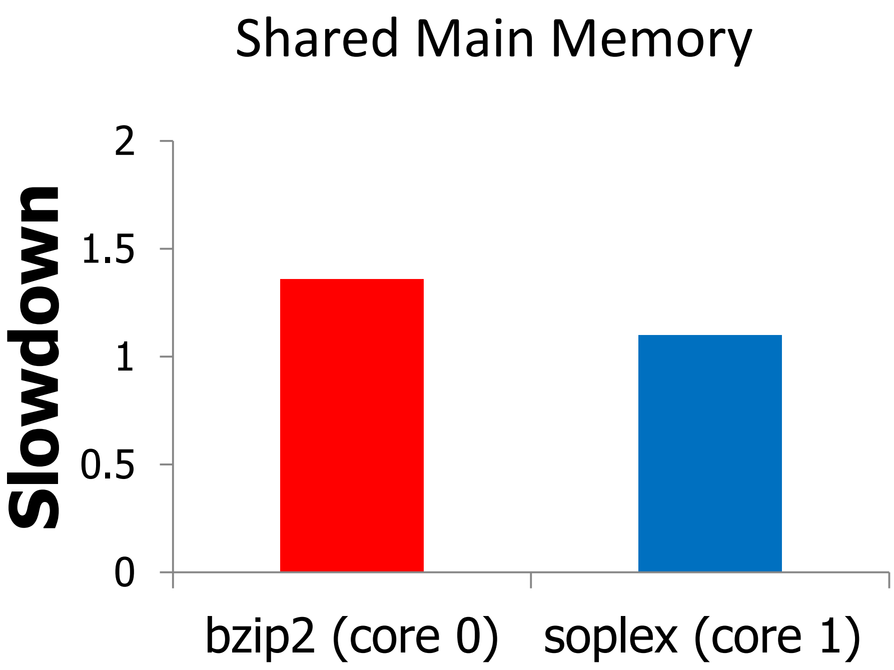


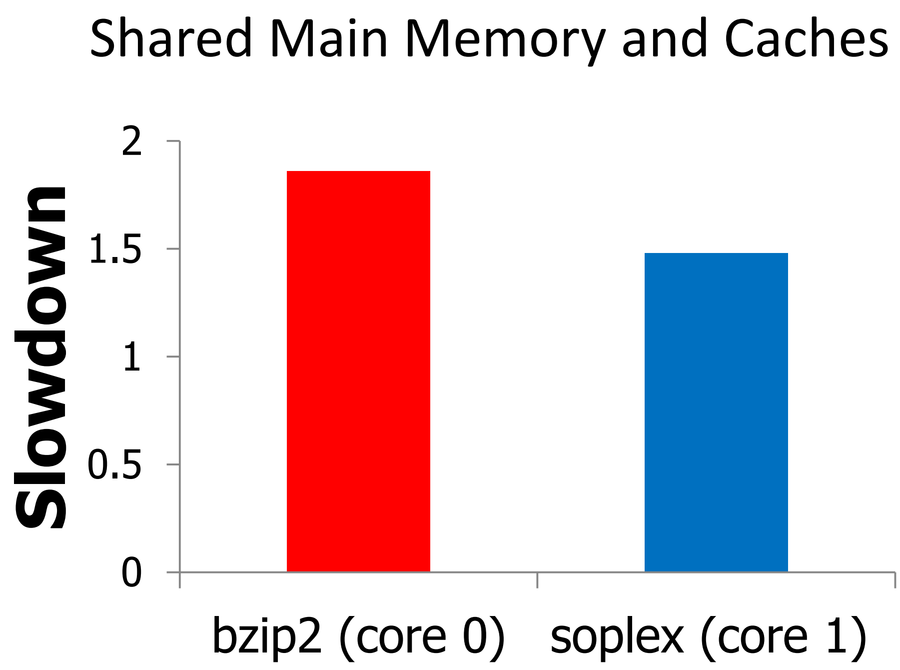
Accounting for Queueing

$$\text{CAR}_{\text{Alone}} = \frac{\text{\# Accesses During High Priority Epochs}}{\text{\# High Priority Cycles} - \text{\# Contention Cycles} - \text{\#Queueing Cycles}}$$

- A cycles is a queueing cycle if
  - a request from the highest priority application is outstanding *and*
  - the previously scheduled request was from another application

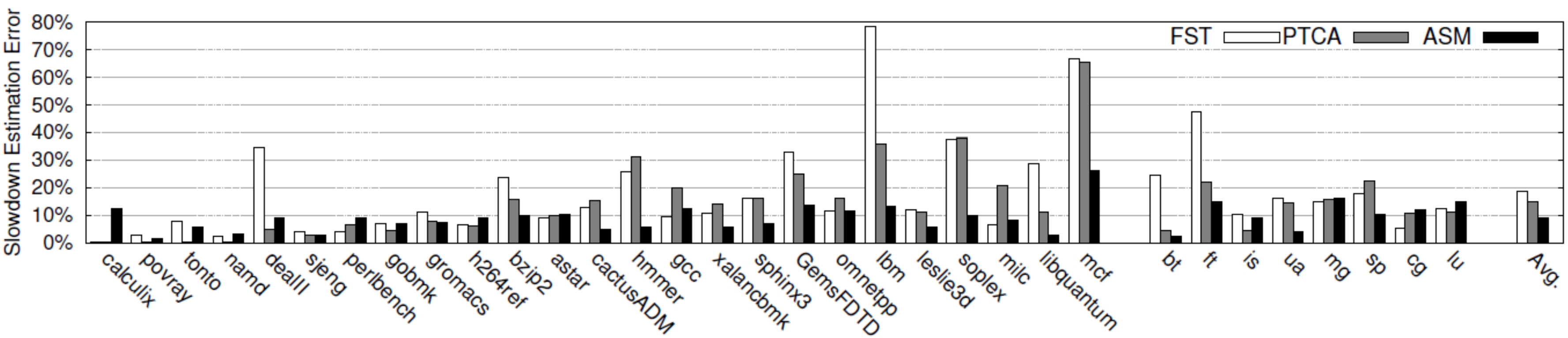
Impact of Cache Capacity Contention



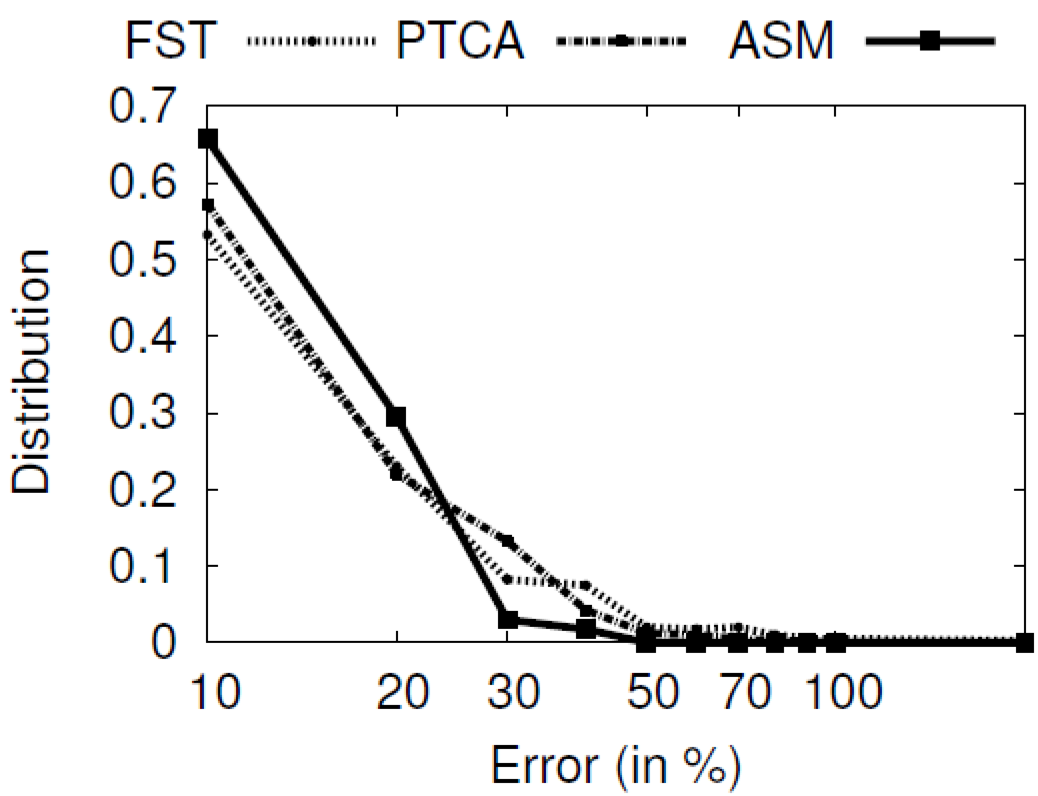


Cache capacity interference causes high application slowdowns

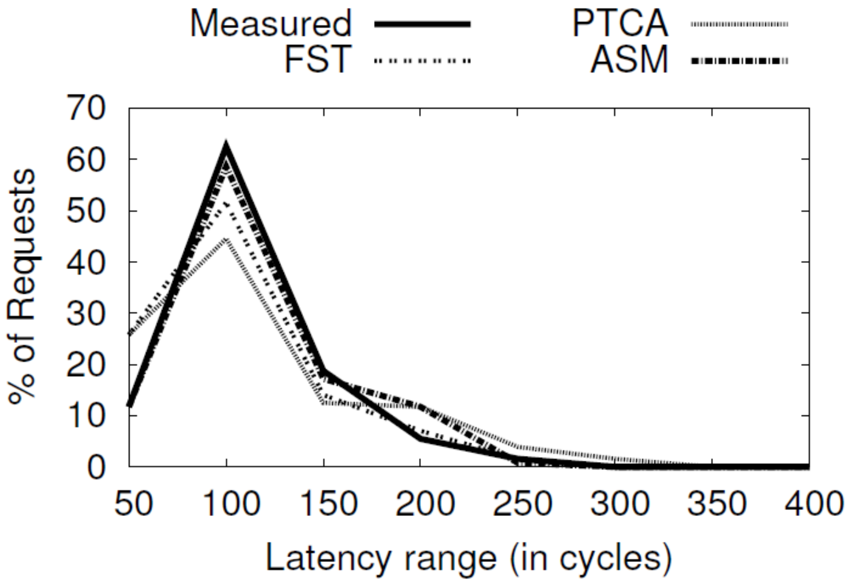
Error with No Sampling



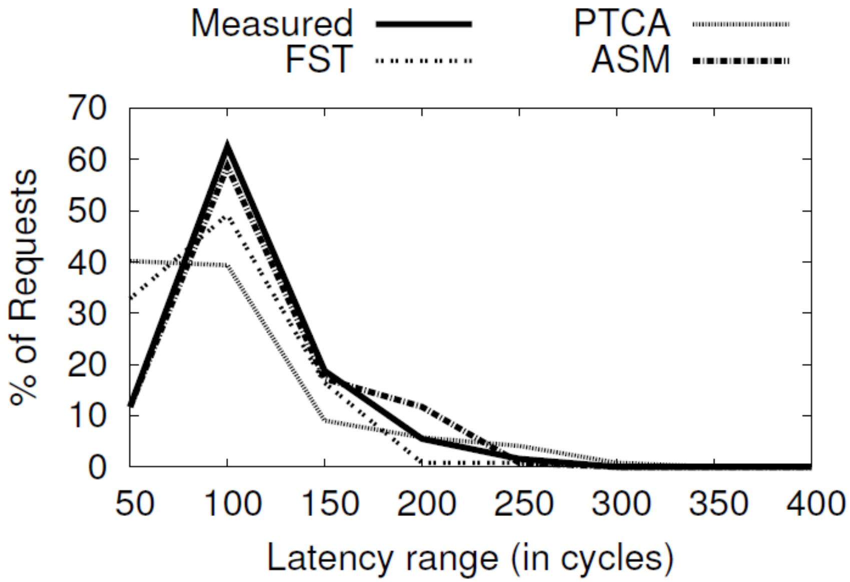
Error Distribution



Miss Service Time Distributions

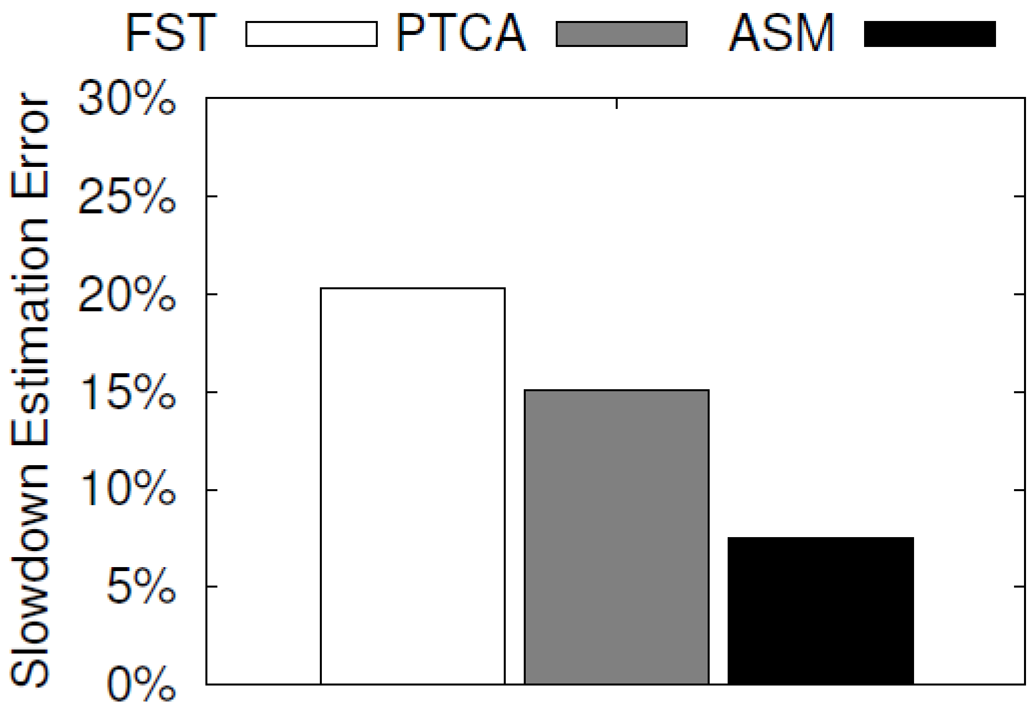


(a) No sampling



(b) With sampling

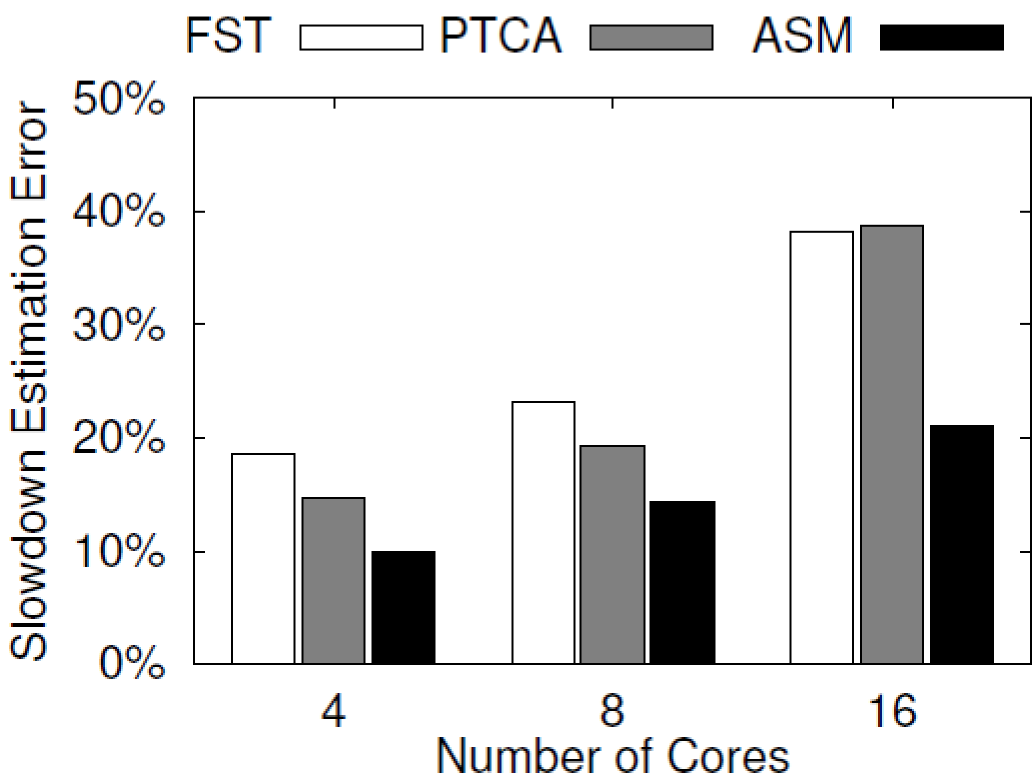
Impact of Prefetching



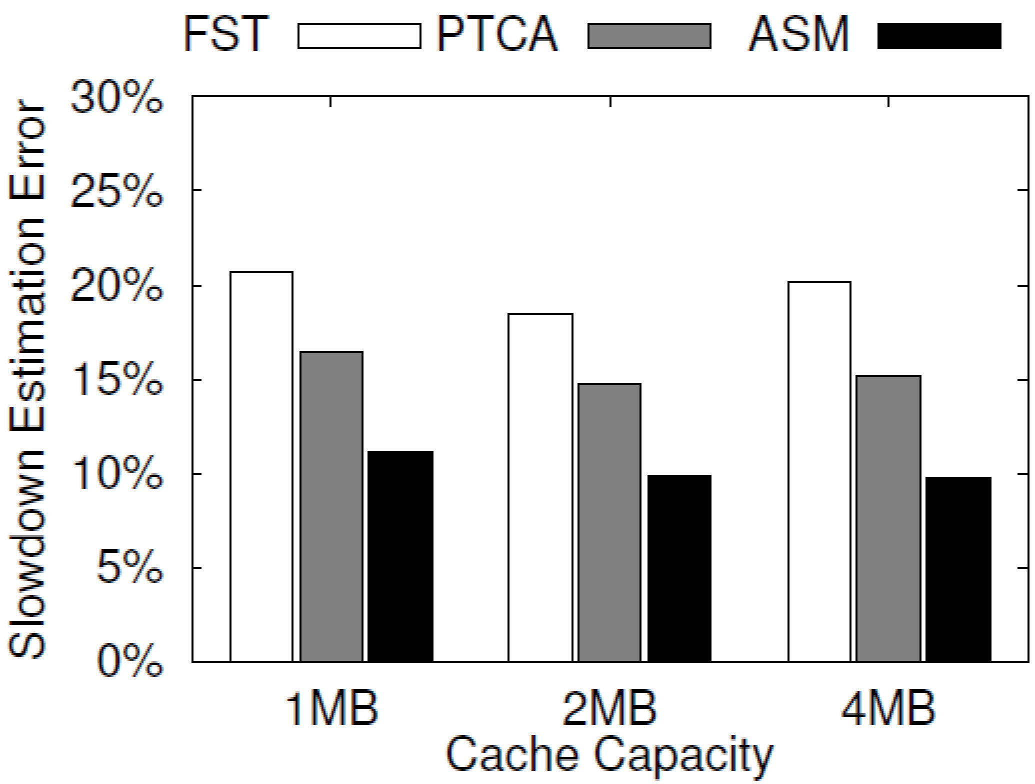
Sensitivity to Epoch and Quantum Lengths

Quantum Length \ Epoch Length	1000	10000	50000	100000
1000000	18.4%	12%	14%	16.6%
5000000	17.1%	9.9%	10.6%	11.5%
10000000	16.9%	9.2%	9.9%	10.5%

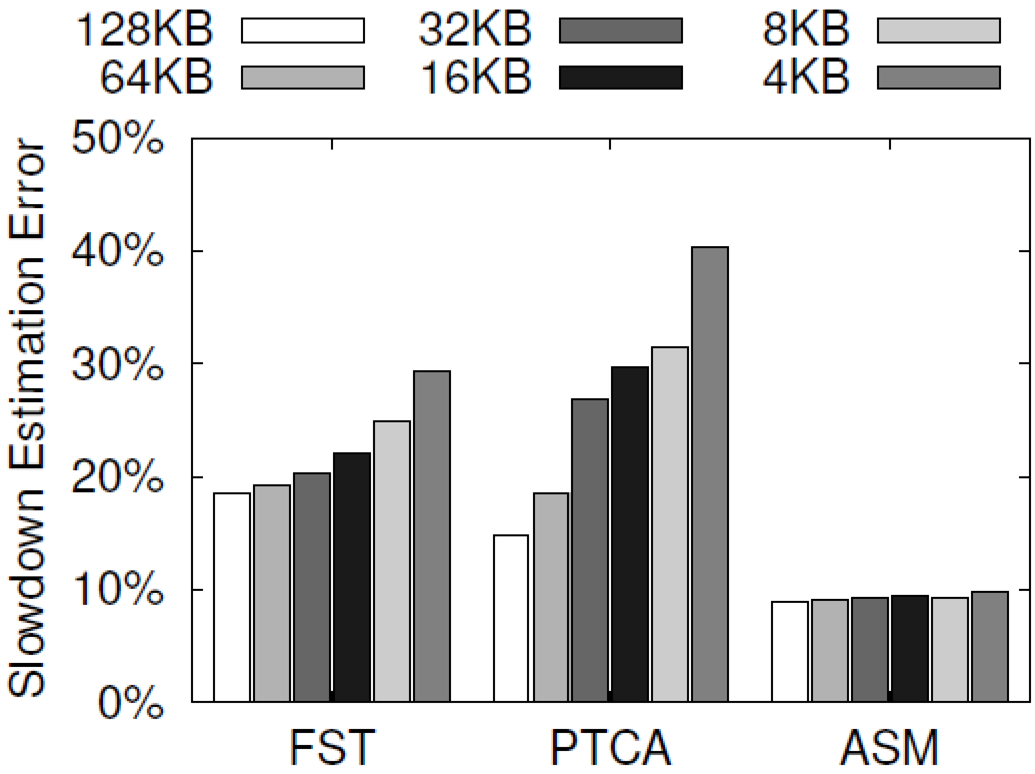
Sensitivity to Core Count



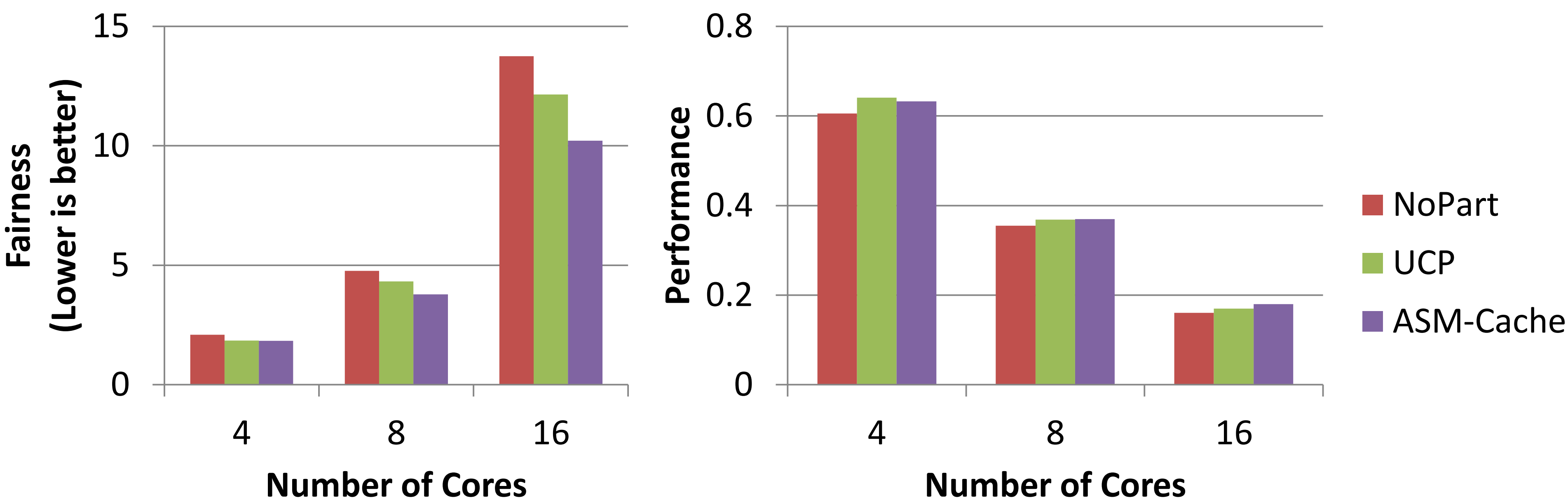
Sensitivity to Cache Capacity



Sensitivity to Auxiliary Tag Store Sampling

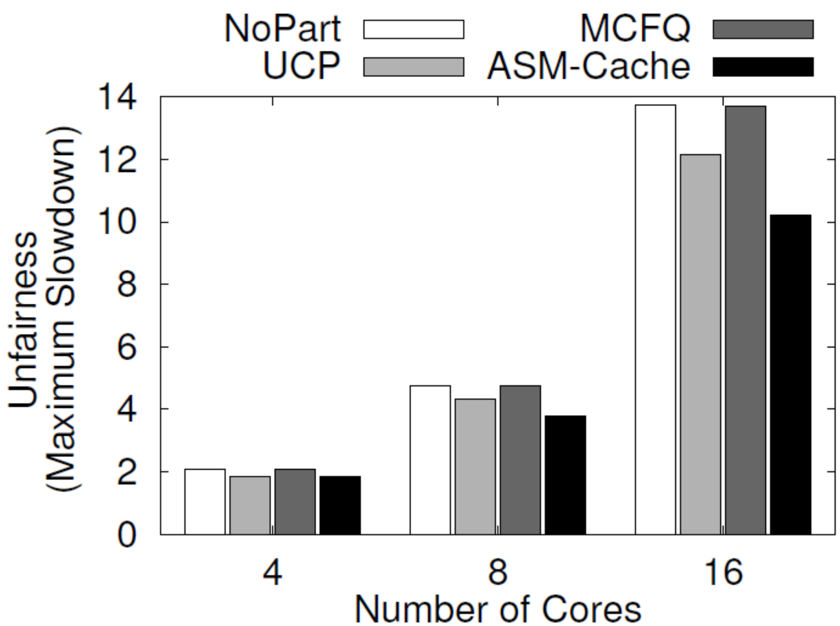


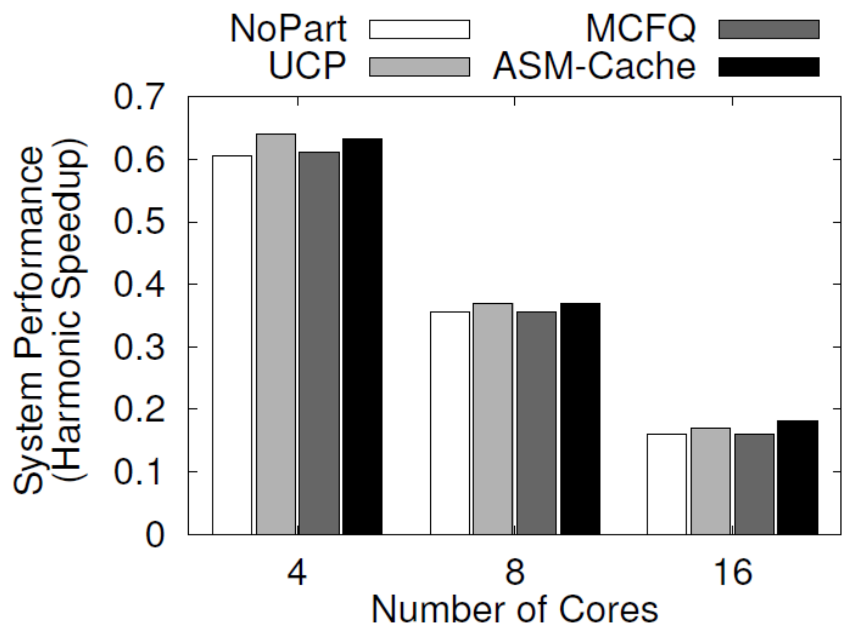
ASM-Cache: Fairness and Performance Results



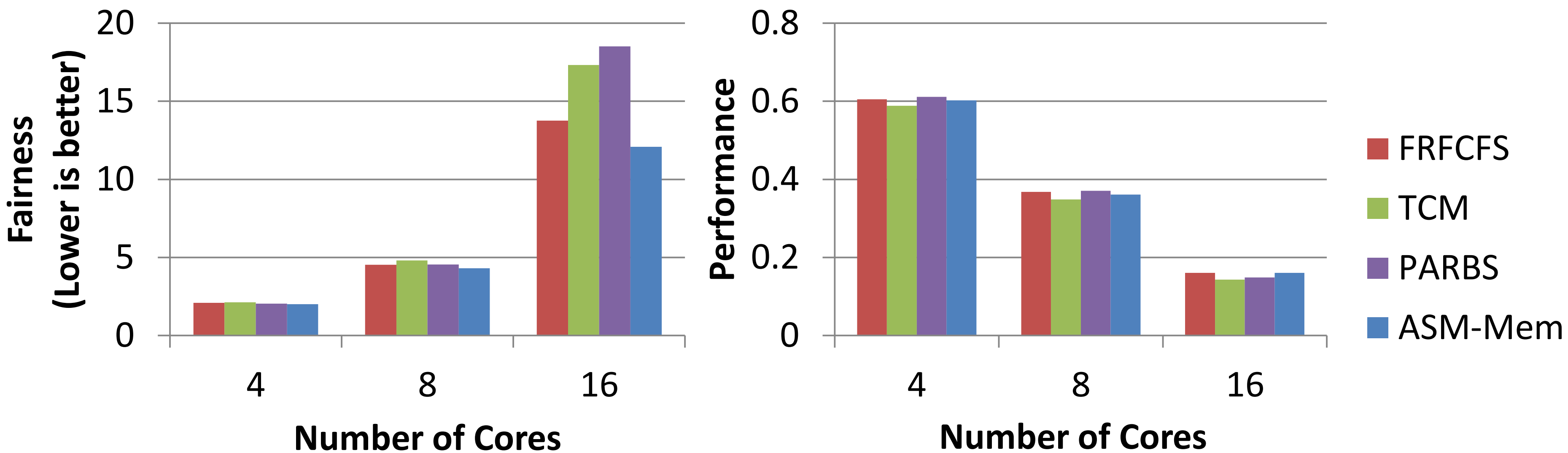
Significant fairness benefits across different systems

ASM-Cache: Fairness and Performance Results



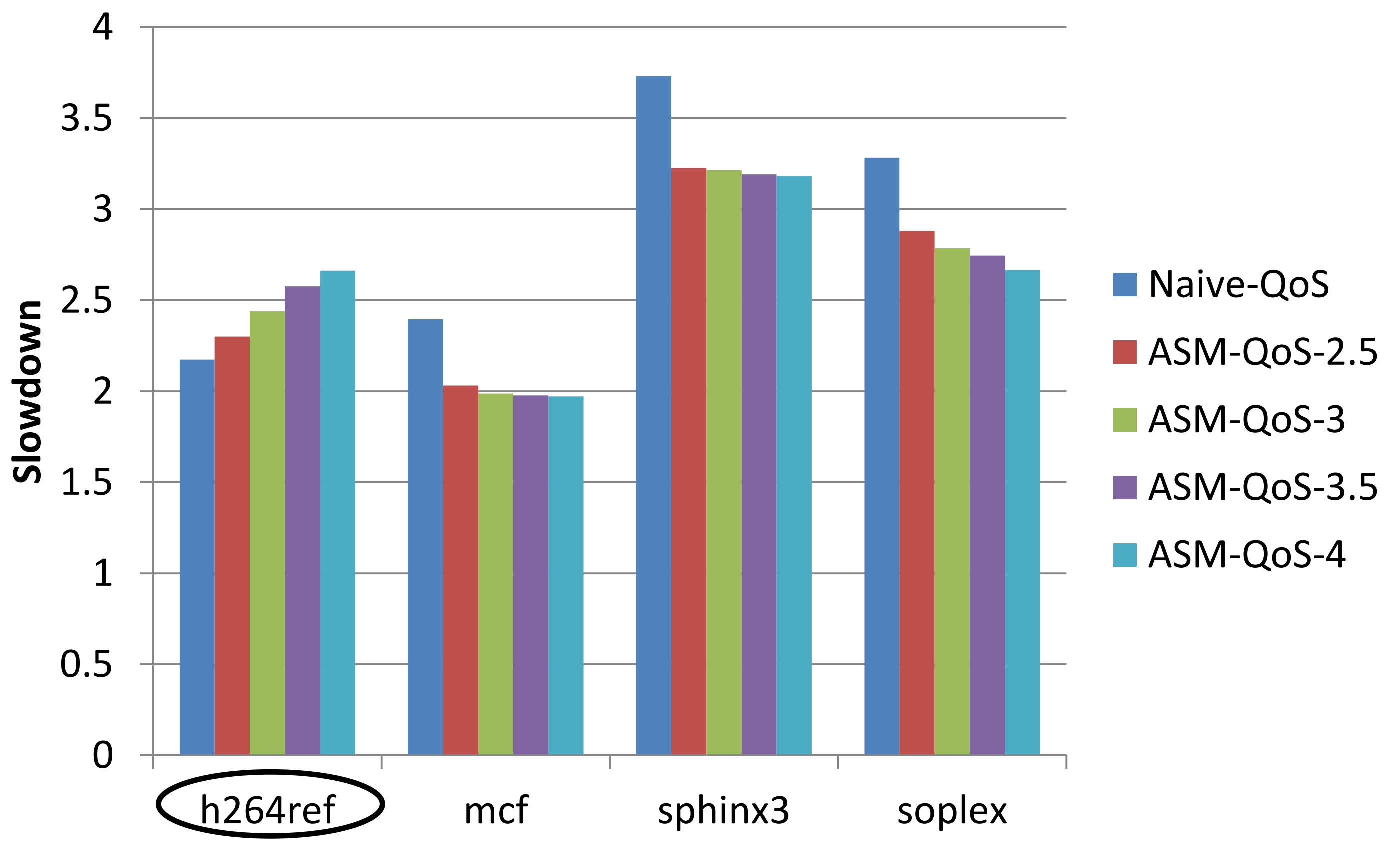


ASM-Mem: Fairness and Performance Results

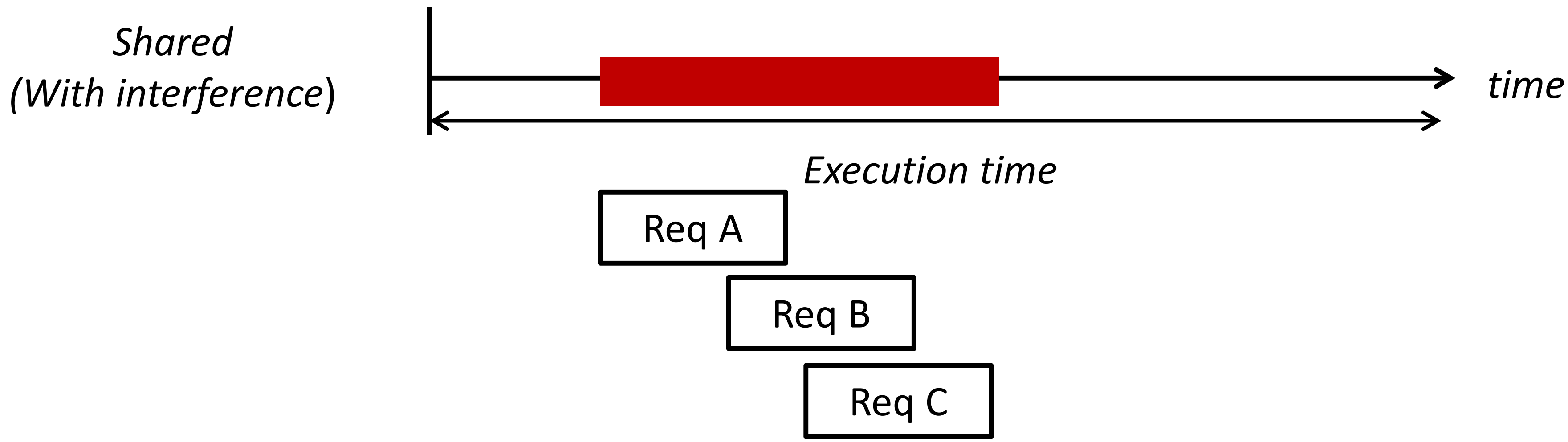


Significant fairness benefits across different systems

ASM-QoS: Meeting Slowdown Bounds

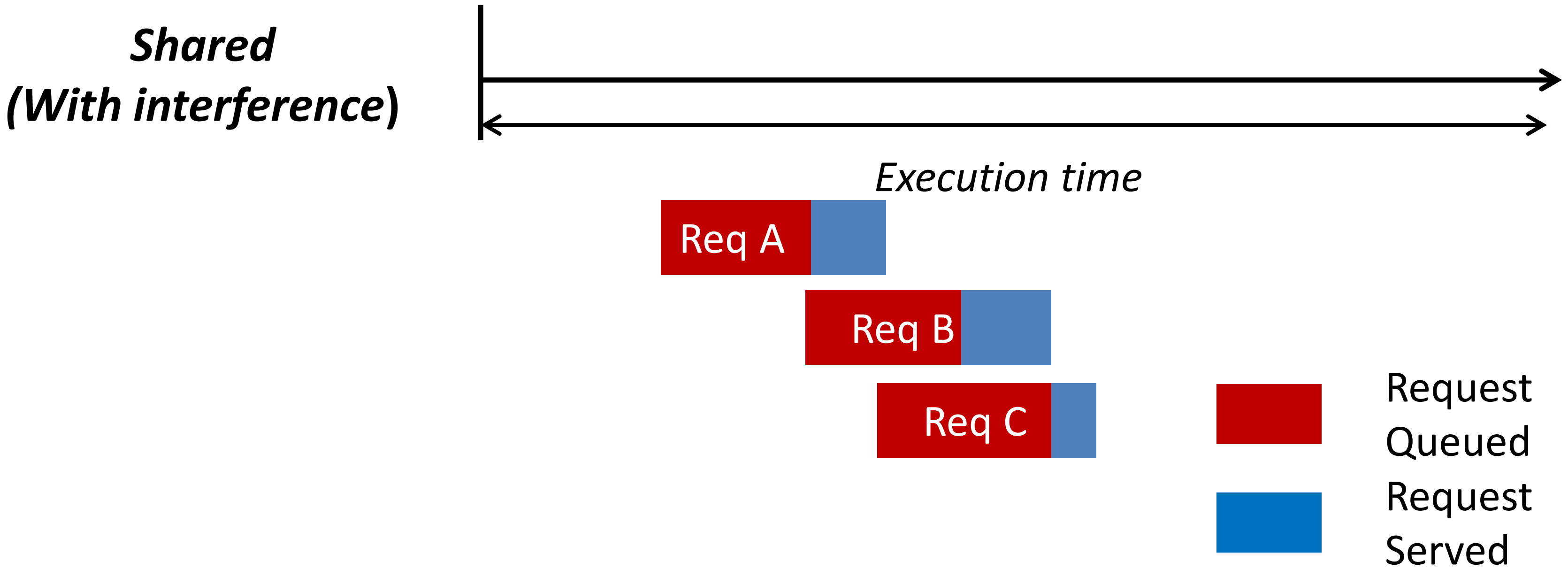


Previous Approach: Estimate Interference Experienced Per-Request



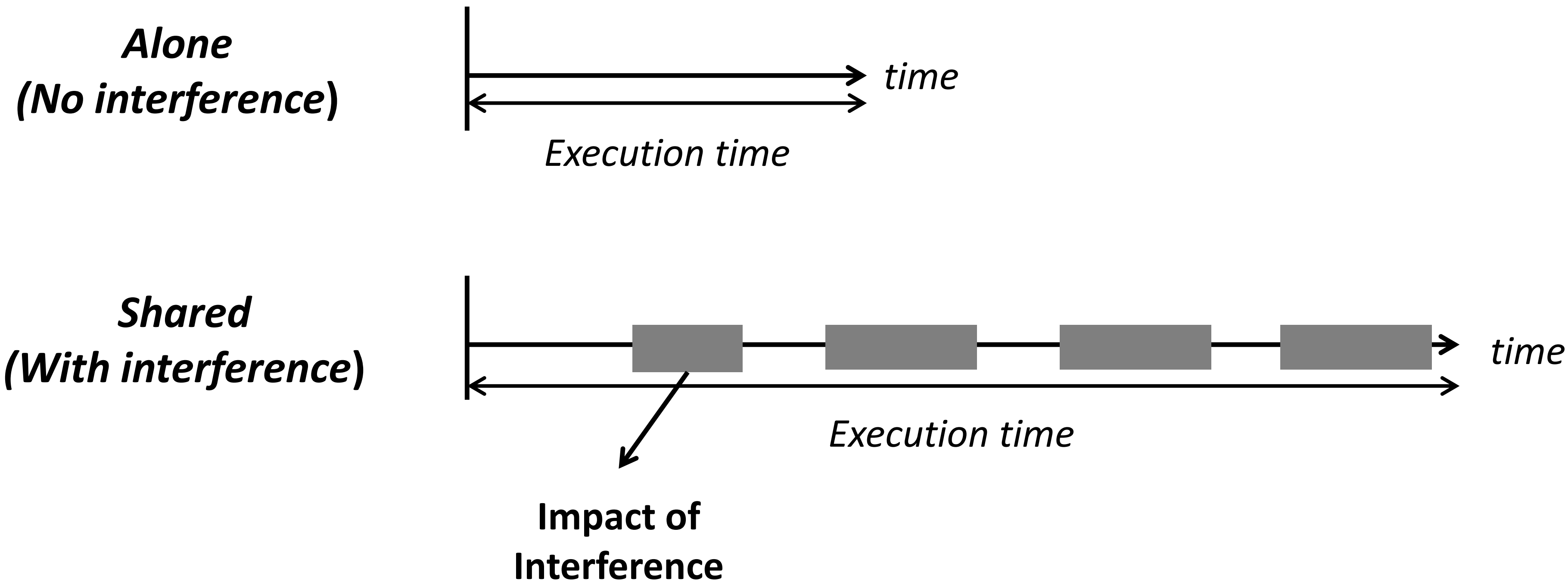
Request Overlap Makes Interference Estimation Per-Request Difficult

Estimating Performance$_{Alone}$



Difficult to estimate impact of interference per-request due to request overlap

Impact of Interference on Performance



Previous Approach: Estimate impact of interference at a per-request granularity

Difficult to estimate due to request overlap