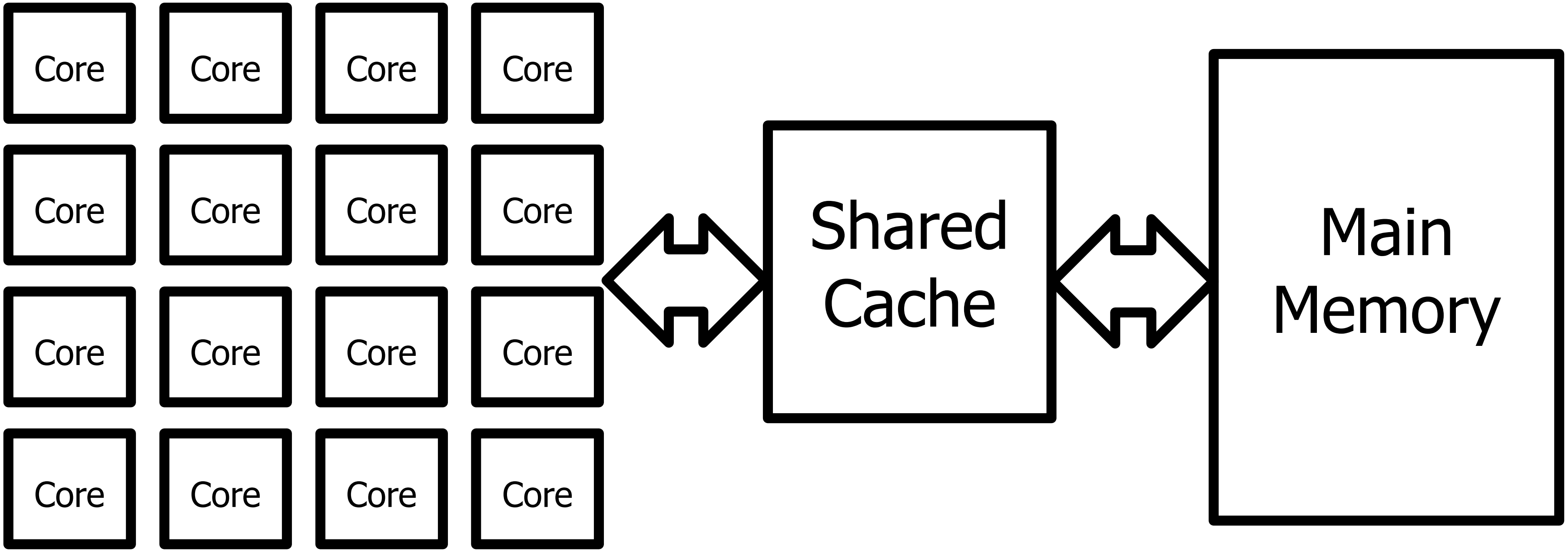
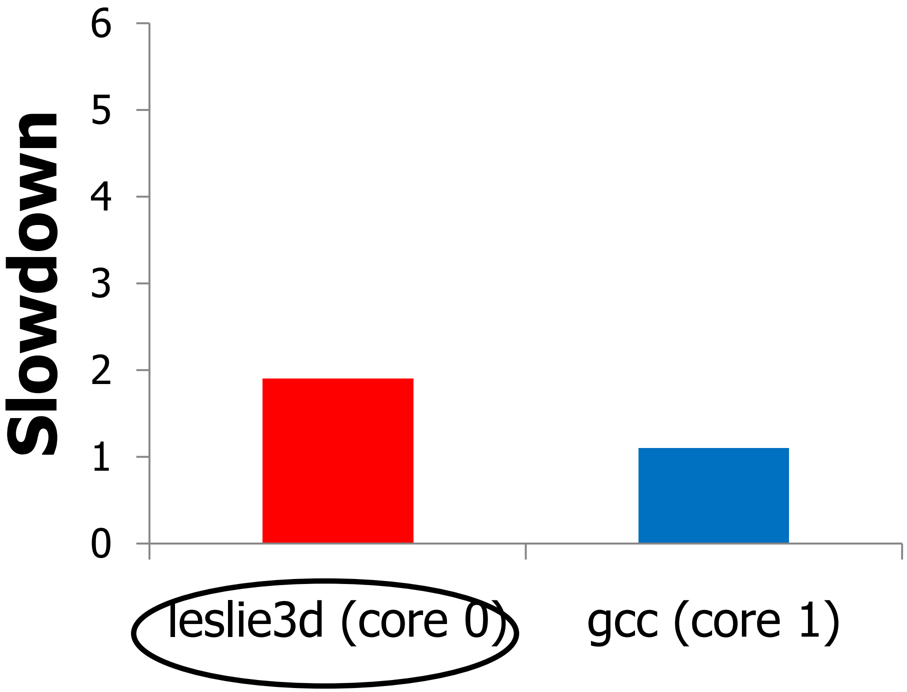
Providing High and Predictable Performance in Multicore Systems Through Shared Resource Management

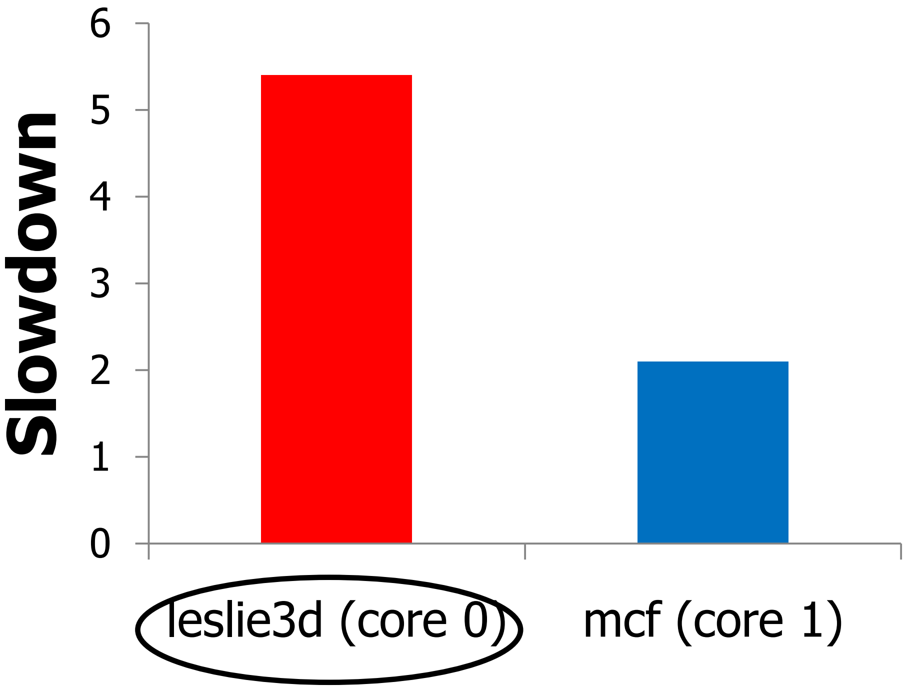
Lavanya Subramanian

Shared Resource Interference



High and Unpredictable Application Slowdowns

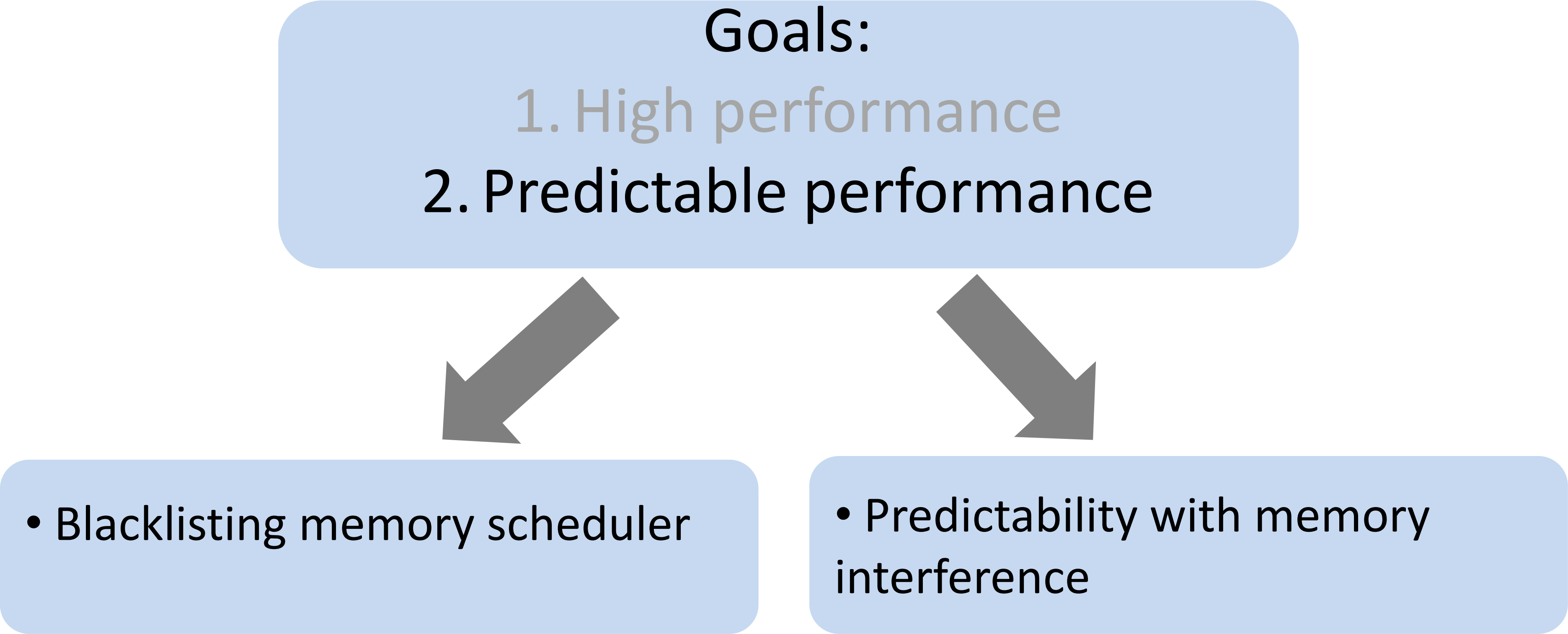




1. High application slowdowns due to shared resource interference

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

Outline



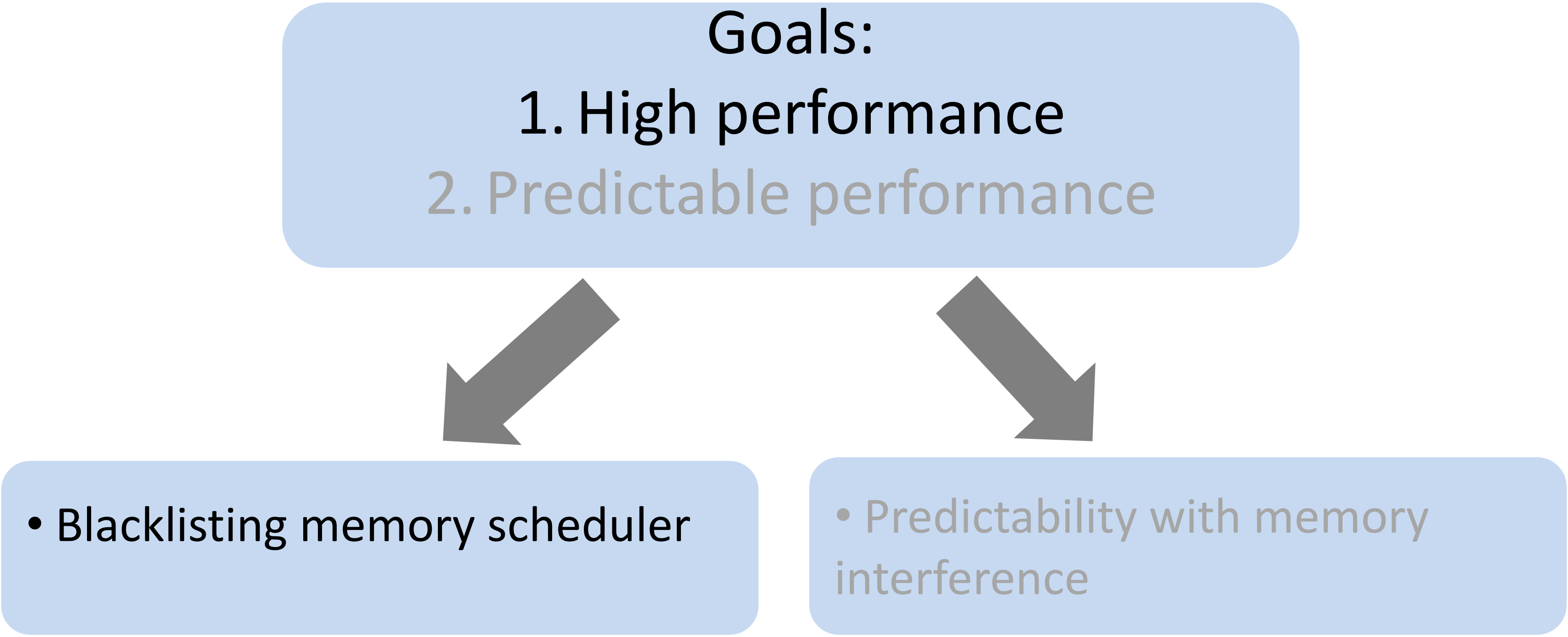
Goals:

1. High performance
2. Predictable performance

- Blacklisting memory scheduler

- Predictability with memory interference

Outline



Goals:

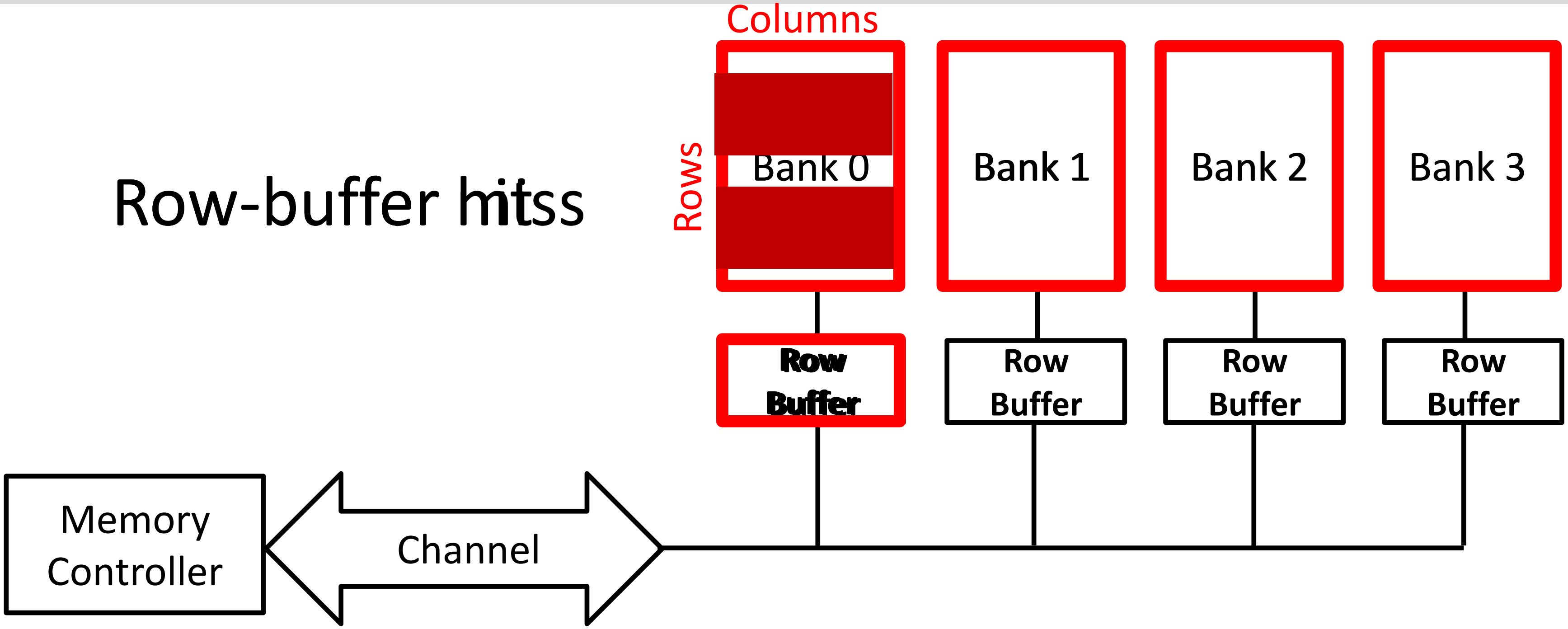
1. High performance
2. Predictable performance

- Blacklisting memory scheduler

- Predictability with memory interference

Background: Main Memory

Row-buffer hitss



- FR-FCFS Memory Scheduler [Zuravleff and Robinson, US Patent '97; Rixner et al., ISCA '00]
  - Row-buffer hit first
  - Older request first
- Unaware of inter-application interference

Tackling Inter-Application Interference: Memory Request Scheduling

- Monitor application memory access characteristics

- Rank applications based on memory access characteristics

- Prioritize requests at the memory controller, based on ranking

An Example:
Thread Cluster Memory Scheduling

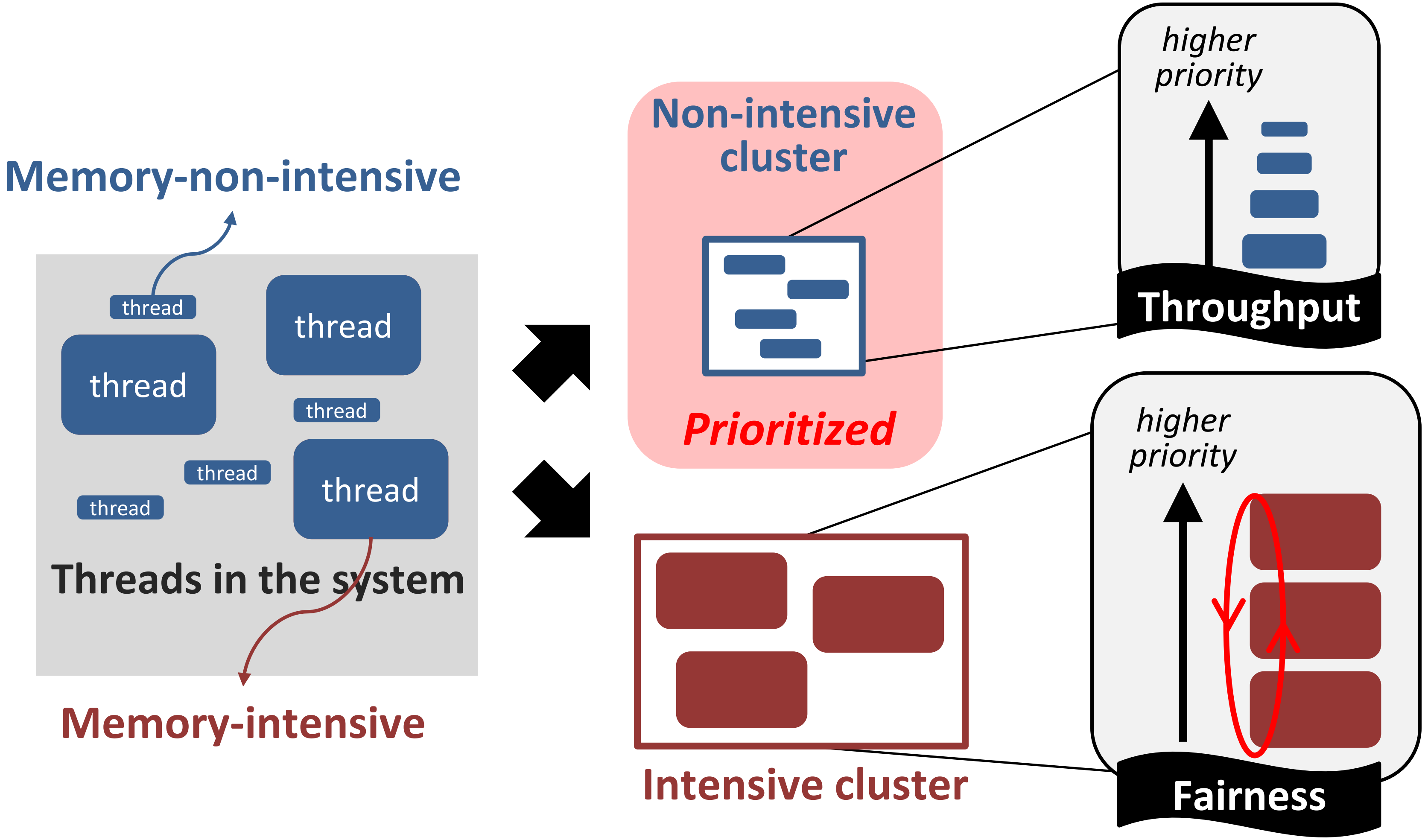


Figure: Kim et al., MICRO 2010

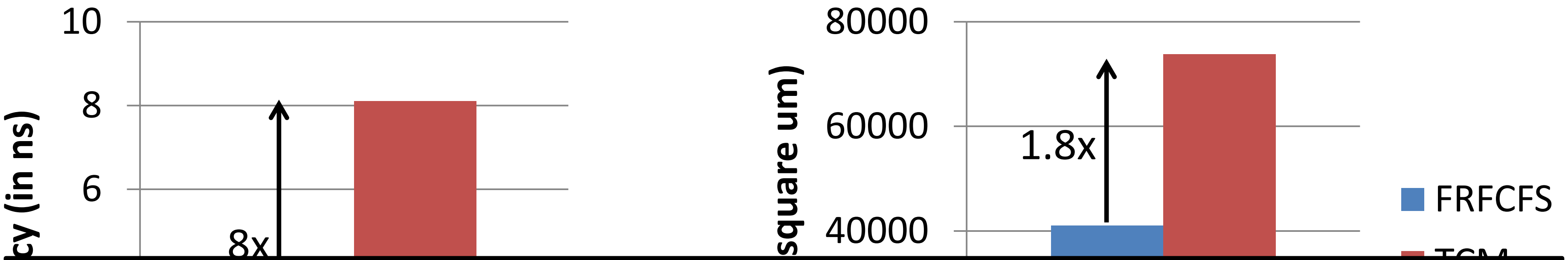
Problems with Previous Application-aware Memory Schedulers

- Hardware Complexity
  - Ranking incurs high hardware cost

- Unfair slowdowns of some applications
  - Ranking causes unfairness

High Hardware Complexity

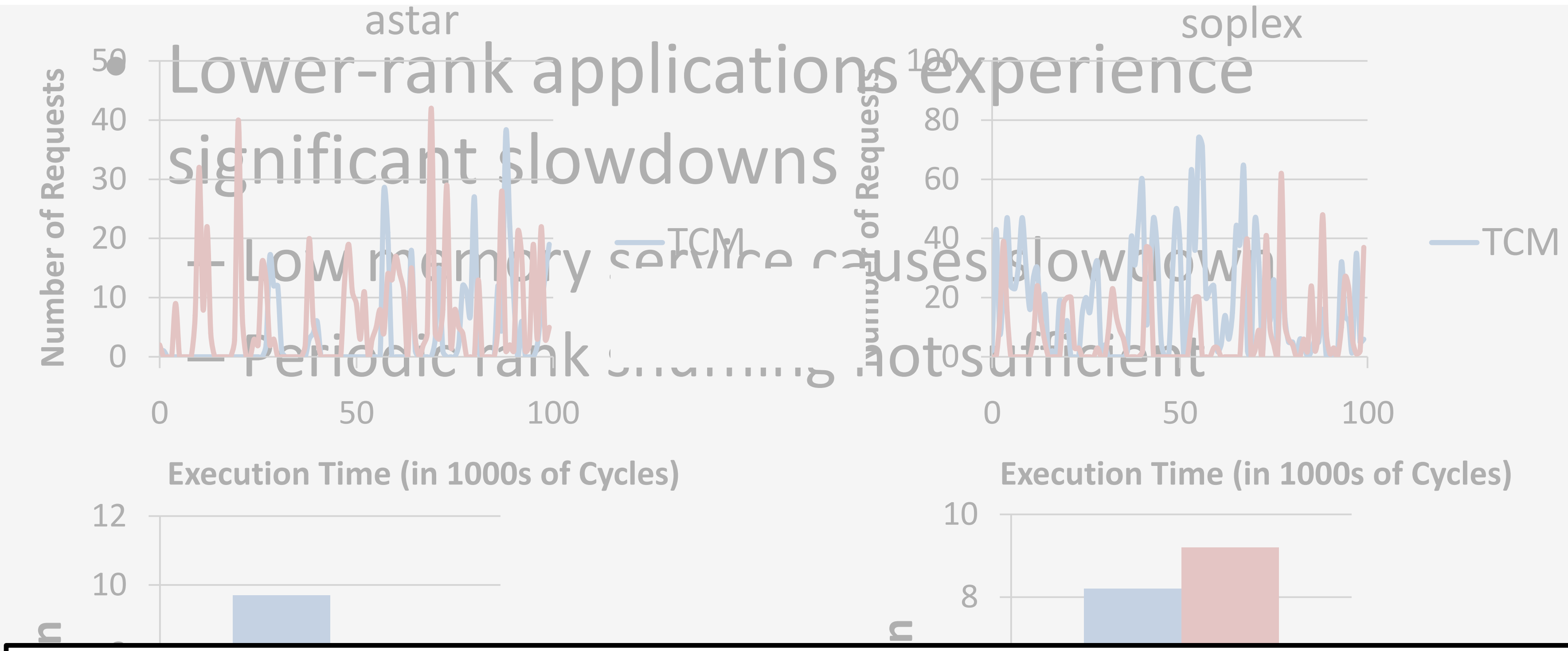
- Ranking incurs high hardware cost
  - Rank computation incurs logic/storage cost
  - Rank enforcement requires comparison logic



Avoid ranking to achieve low hardware cost

Ranking Causes Unfair Application Slowdowns

- Lower-rank applications experience significant slowdowns
  - Low memory service causes slowdown
  - Periodic rank shuffling not sufficient



Grouping offers lower unfairness than ranking

Problems with Previous Application-Aware Memory Schedulers

- Hardware Complexity
  - Ranking incurs high hardware cost
- Unfair slowdowns of some applications
  - Ranking causes unfairness

Our Goal: Design a memory scheduler with
Low Complexity, High Performance, and Fairness

Towards a New Scheduler Design

- Monitor applications that have a number of consecutive requests served

1. Simple Grouping Mechanism

- Blacklist such applications

- Prioritize requests of non-blacklisted applications

2. Enforcing Priorities Based On Grouping

- Periodically clear blacklists

Methodology

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

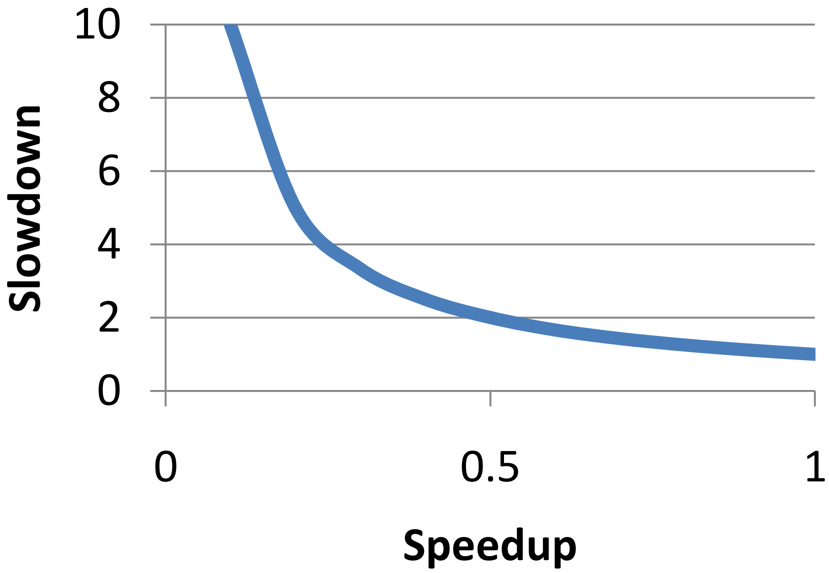
- Workloads
  - SPEC CPU2006, TPCC, Matlab
  - 80 multi programmed workloads

Metrics

- System Performance:

$$\text{Weighted Speedup} = \sum_i \frac{IPC_i^{shared}}{IPC_i^{alone}}$$

$$\text{Harmonic Speedup} = \frac{N}{\sum_i \frac{IPC_i^{alone}}{IPC_i^{shared}}}$$



- Fairness:

$$\text{Maximum Slowdown} = \max \frac{IPC_i^{alone}}{IPC_i^{shared}}$$

Previous Memory Schedulers

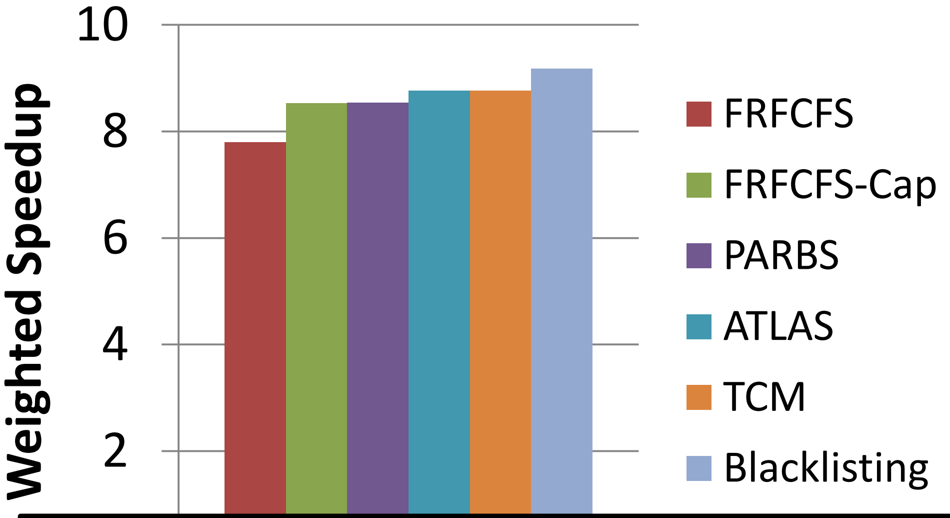
- FR-FCFS [Zuravleff and Robinson, US Patent 1997, Rixner et al., ISCA 2000]
  - Prioritizes row-buffer hits and older requests
  - Application-unaware

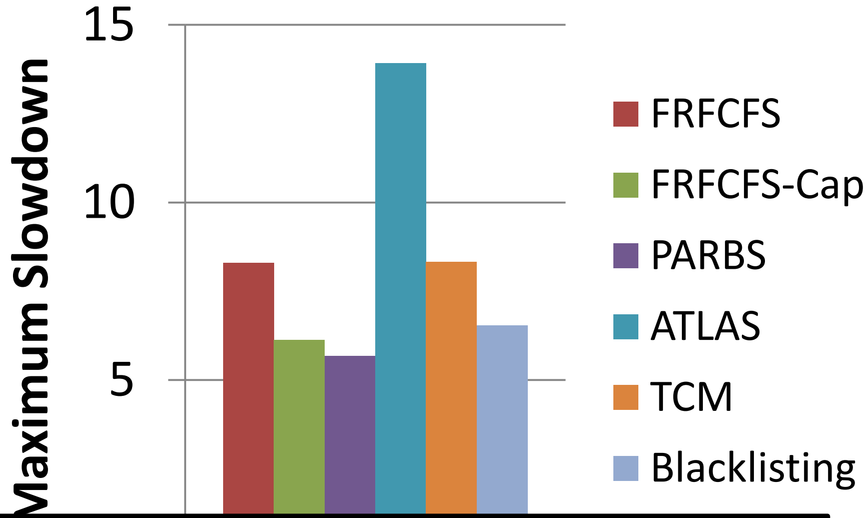
- PARBS [Mutlu and Moscibroda, ISCA 2008]
  - Batches oldest requests from each application; prioritizes batch
  - Employs ranking within a batch

- ATLAS [Kim et al., HPCA 2010]
  - Prioritizes applications with low memory-intensity

- TCM [Kim et al., MICRO 2010]
  - Always prioritizes low memory-intensity applications
  - Shuffles request priorities of high memory-intensity applications

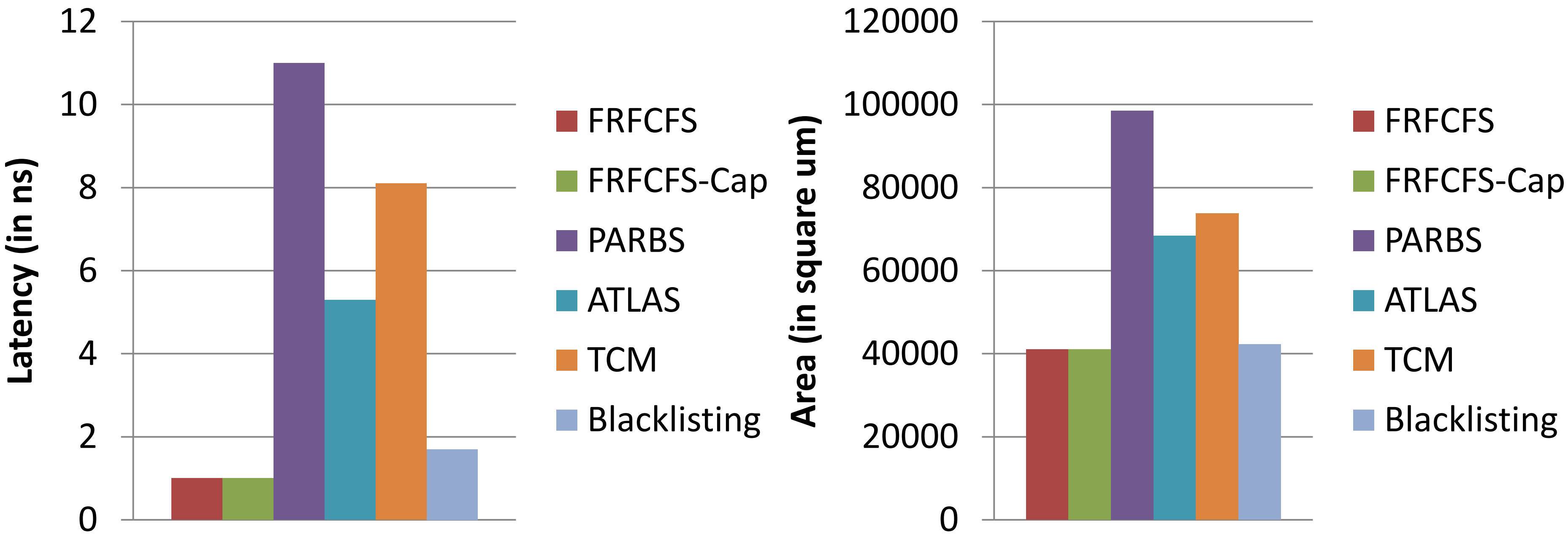
Performance Results





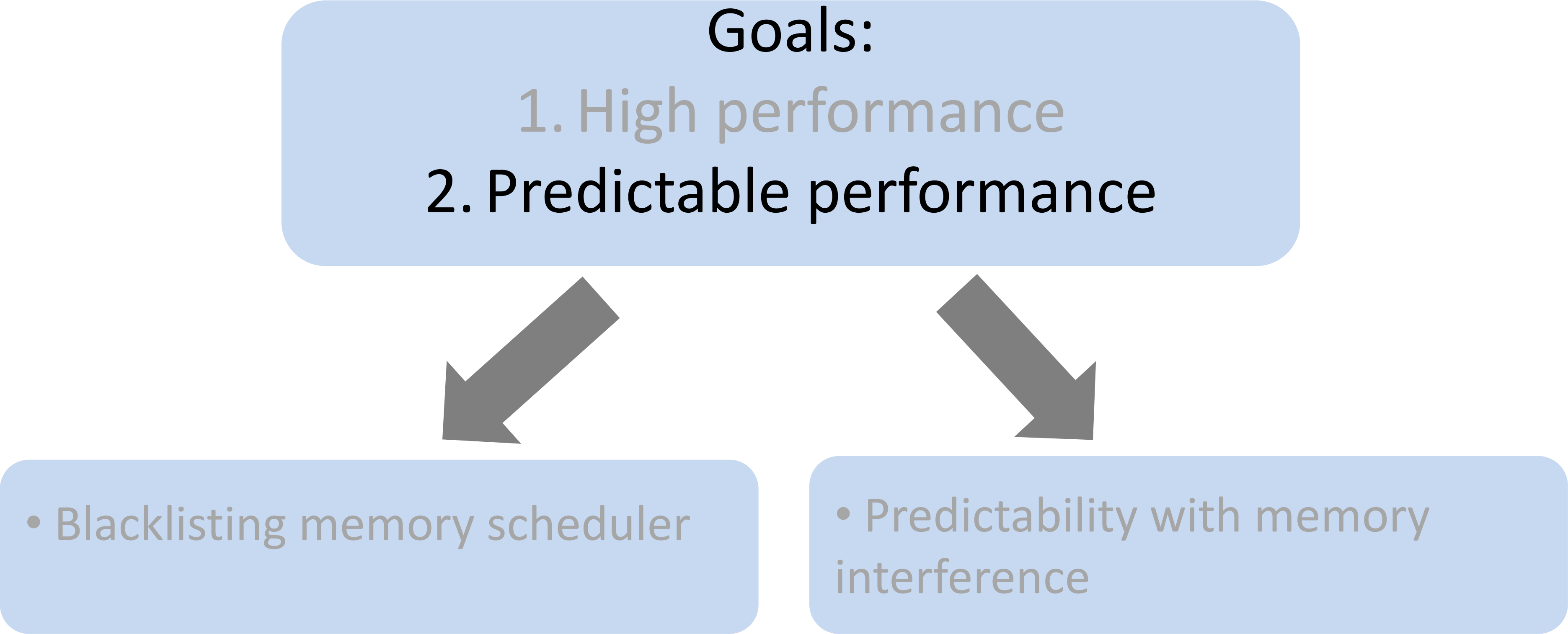
Approaches fairness of PARBS and FRFCFS-Cap achieving better performance than TCM

Complexity Results



Blacklisting achieves
43% lower area than TCM

Outline



Goals:

1. High performance
2. Predictable performance

- Blacklisting memory scheduler

- Predictability with memory interference

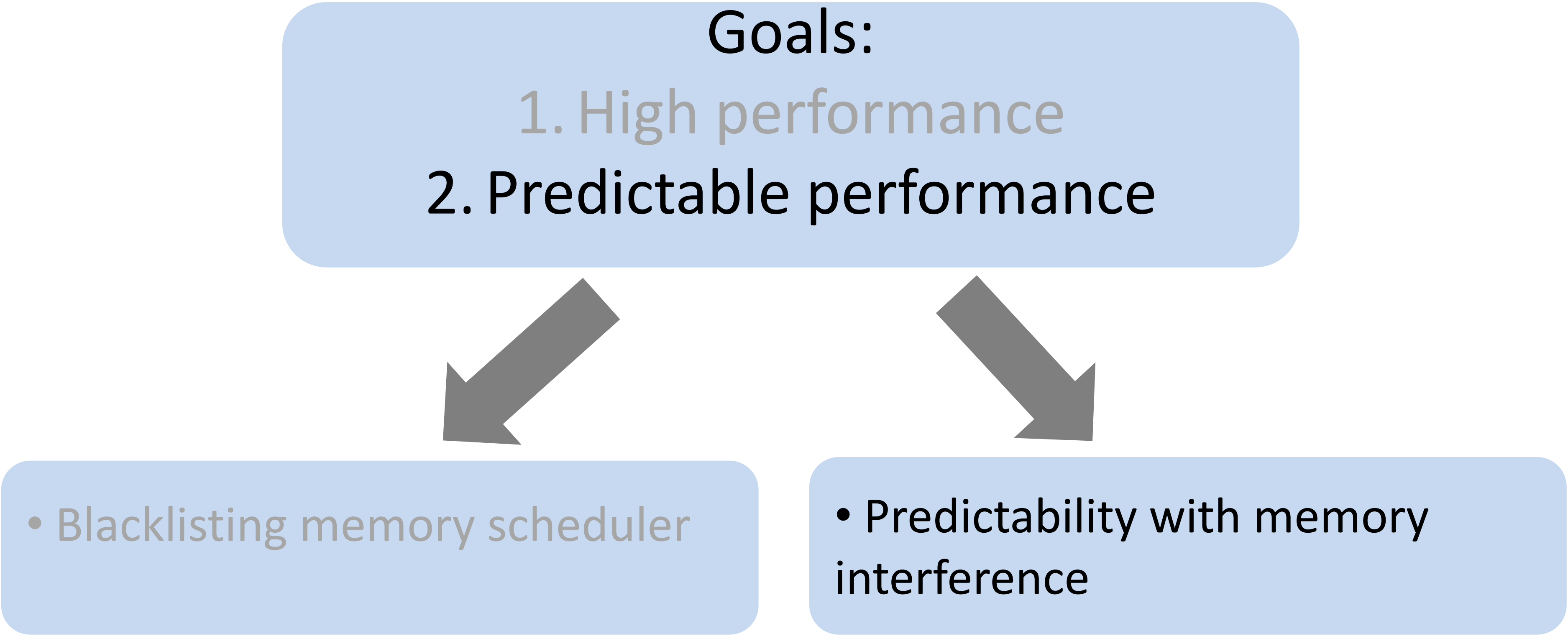
Need for Predictable Performance

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

As a first step: Predictable performance in the presence of memory interference

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

Outline



Goals:

1. High performance
2. Predictable performance

- Blacklisting memory scheduler

- Predictability with memory interference

Predictability in the Presence of Memory Interference

1. Estimate Slowdown

2. Control Slowdown

Predictability in the Presence of Memory Interference

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

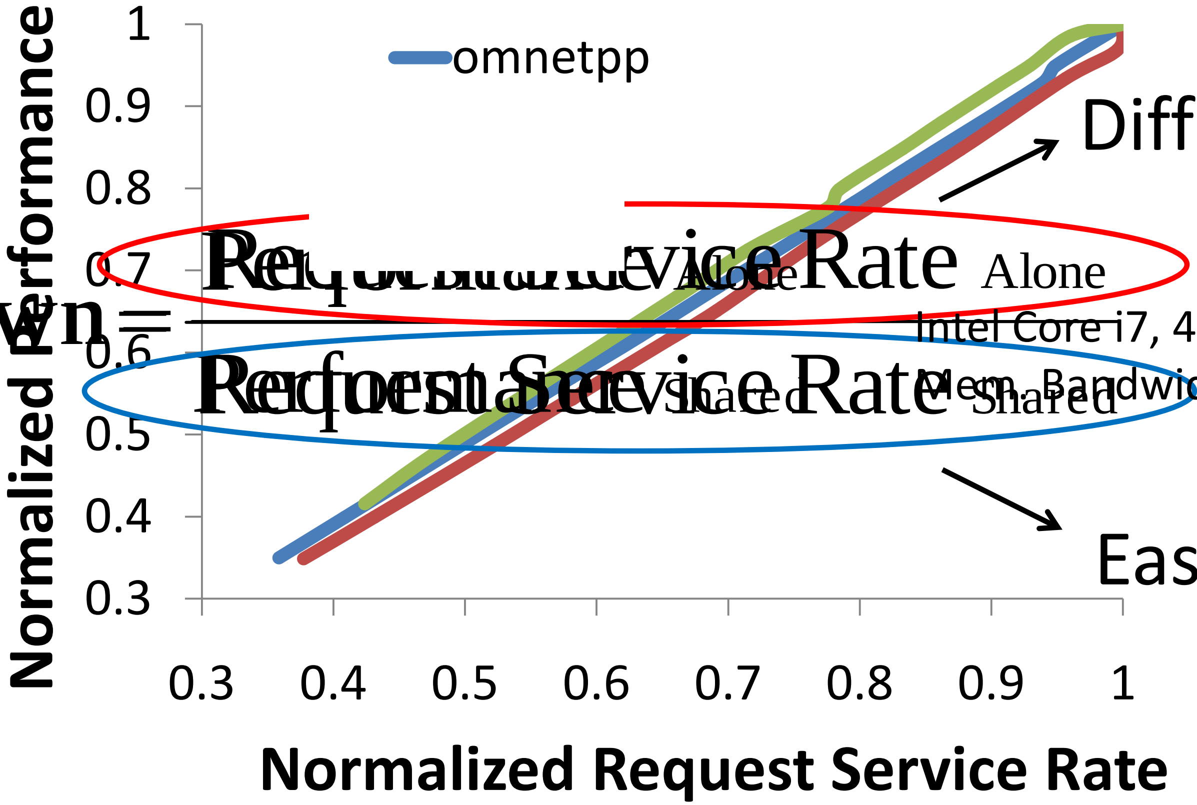
Slowdown: Definition

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

Key Observation 1

For a memory bound application,

Performance ∝ Memory request service rate



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

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

Difficult

Easy

Intel Core i7, 4 cores

Mem. Bandwidth: 8.5 GB/s

Key Observation 2

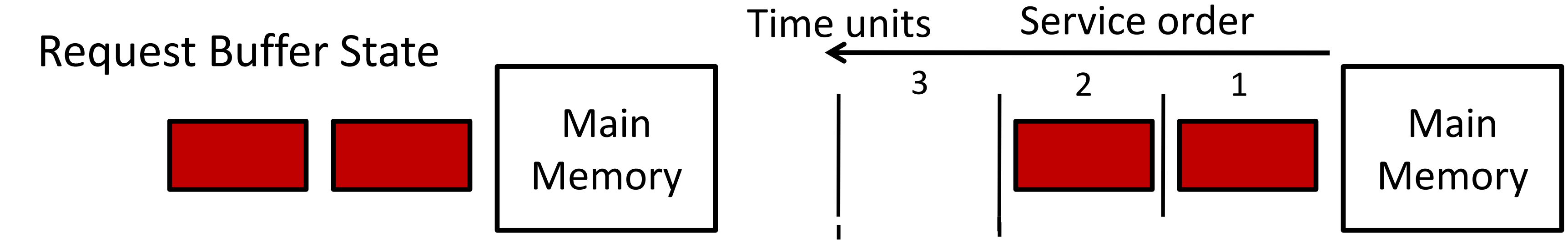
Request Service Rate $_{\text{Alone}}$ ($\text{RSR}_{\text{Alone}}$) of an application can be estimated by giving the application highest priority in accessing memory

Highest priority → Little interference

(almost as if the application were run alone)

Key Observation 2

1. Run alone



Request Buffer State

Main Memory

Time units

Service order

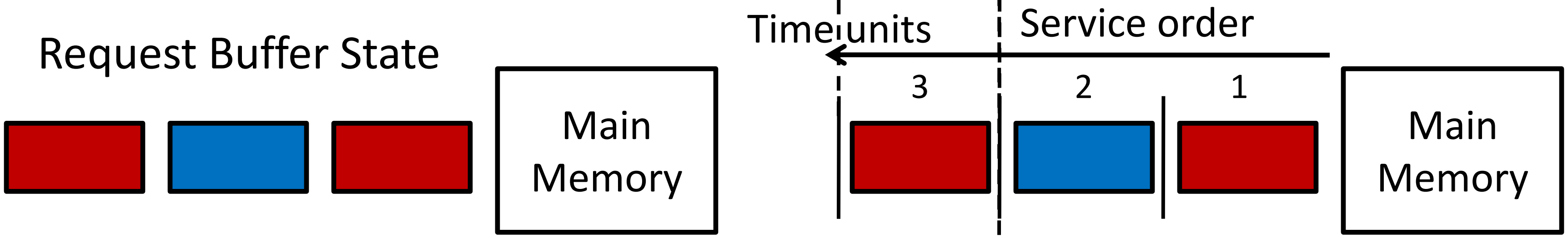
3

2

1

Main Memory

2. Run with another application



Request Buffer State

Main Memory

Time units

Service order

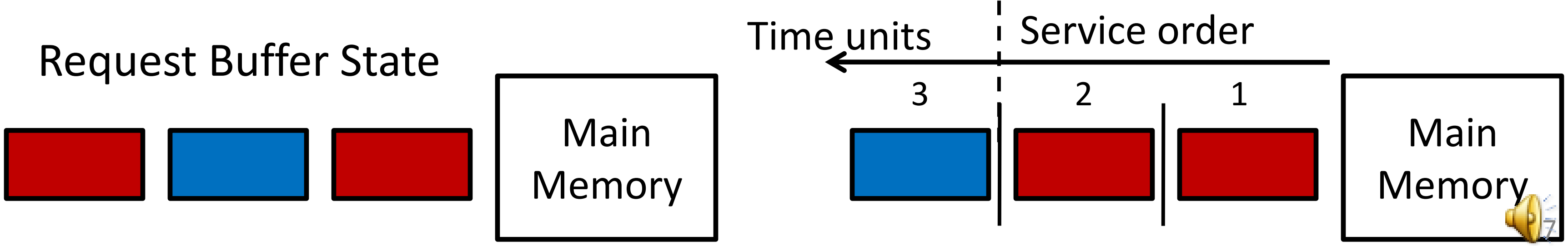
3

2

1

Main Memory

3. Run with another application: highest priority



Request Buffer State

Main Memory

Time units

Service order

3

2

1

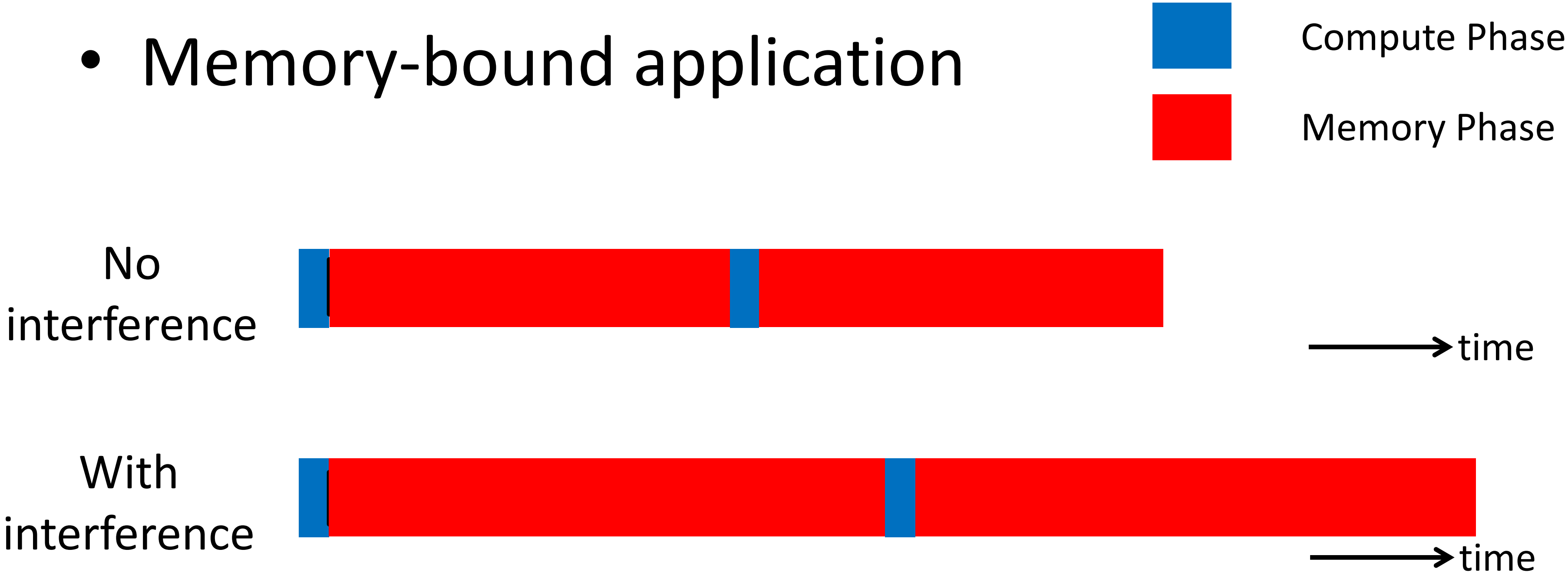
Main Memory

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



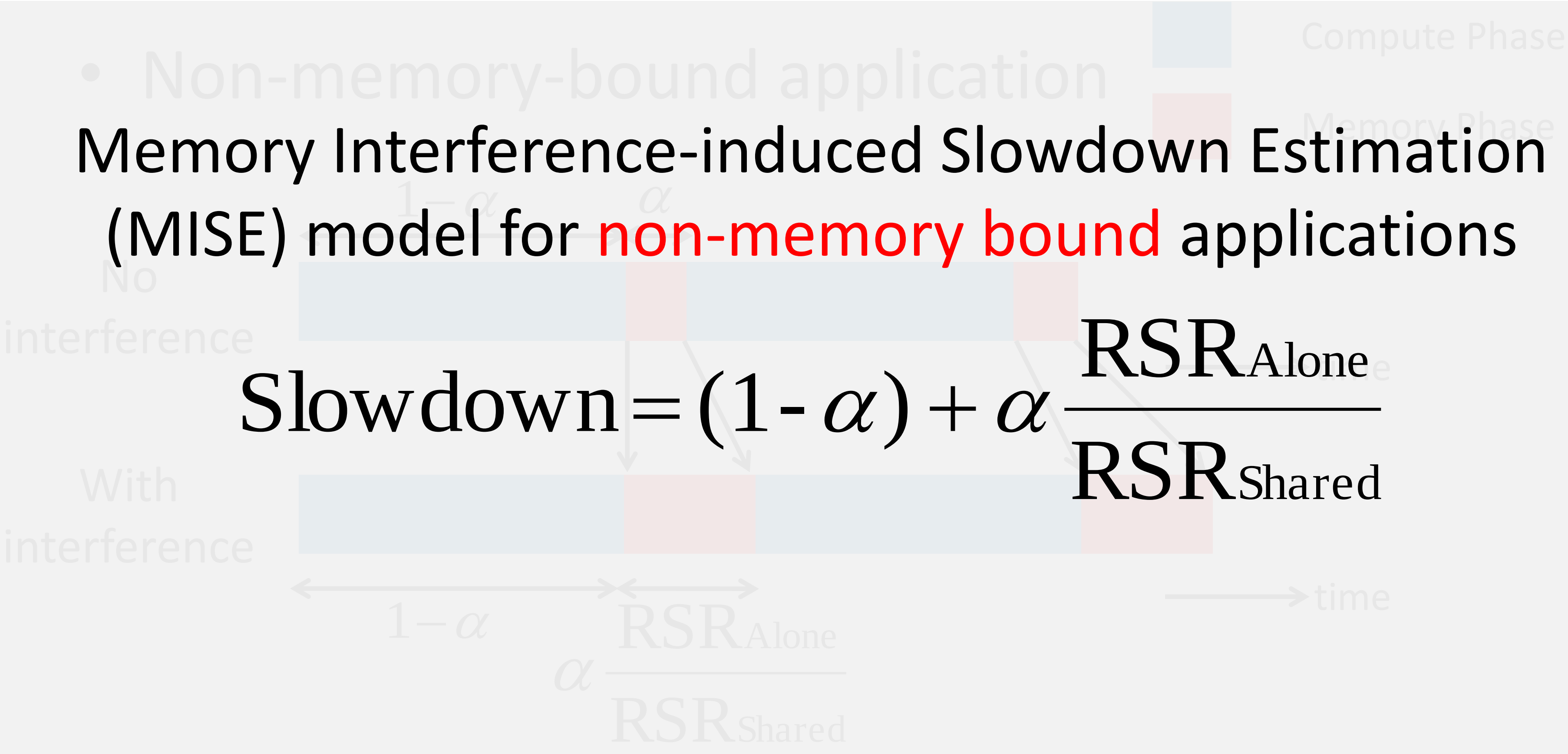
Memory phase slowdown dominates overall slowdown

Key Observation 3

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

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

- Non-memory-bound application

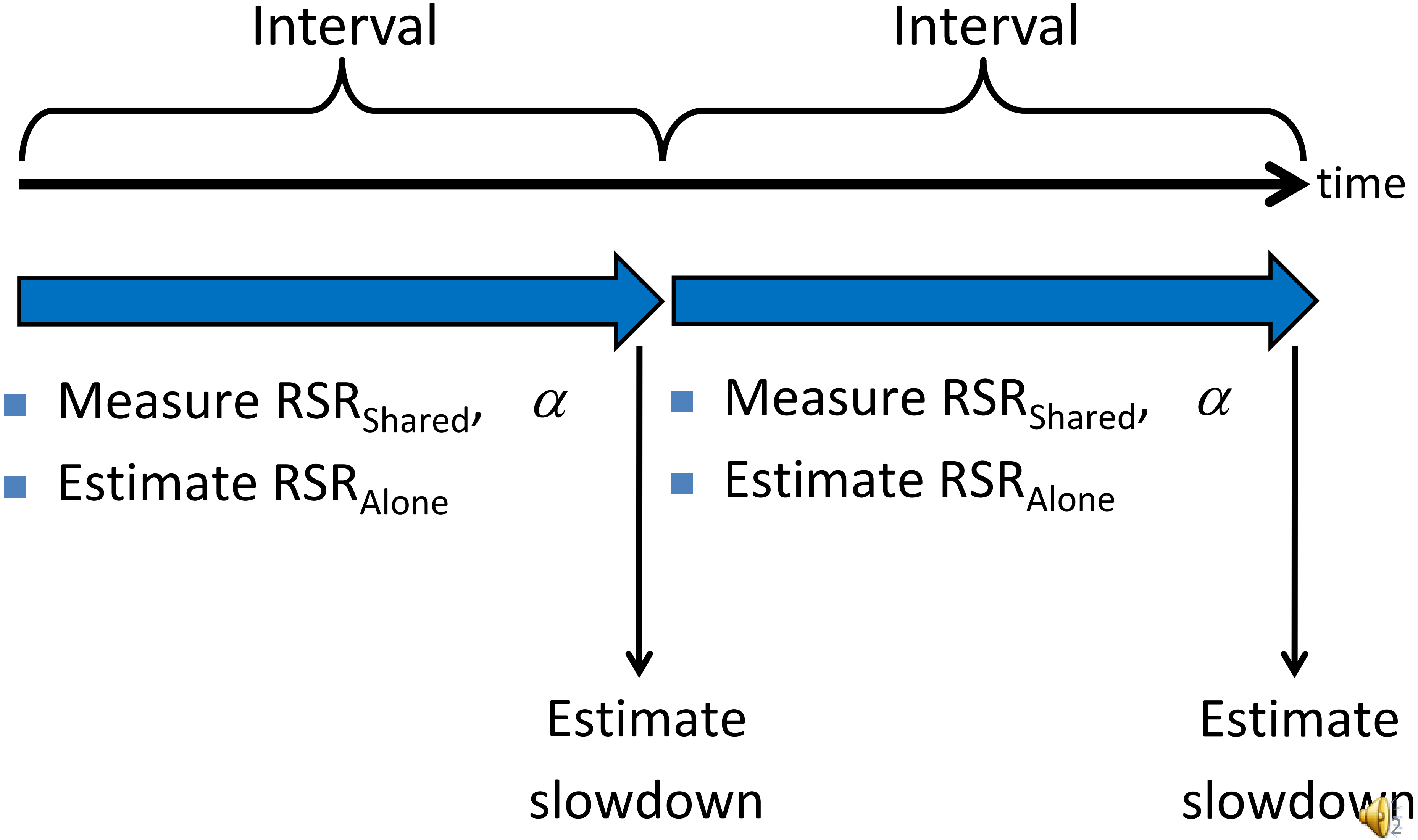


Only memory fraction (α) slows down with interference

Predictability in the Presence of Memory Interference

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

MISE Operation: Putting it All Together



Interval

Interval

time

- Measure RSR_{Shared}, α
- Estimate RSR_{Alone}

- Measure RSR_{Shared}, α
- Estimate RSR_{Alone}

Estimate slowdown

Estimate slowdown

Predictability in the Presence of Memory Interference

1. Estimate Slowdown
   - Key Observations
   - MISE Operation: 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 et al., MICRO '07]
  - **FST** (Fairness via Source Throttling) [Ebrahimi et al., ASPLOS '10]

- Basic Idea:

Difficult

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

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

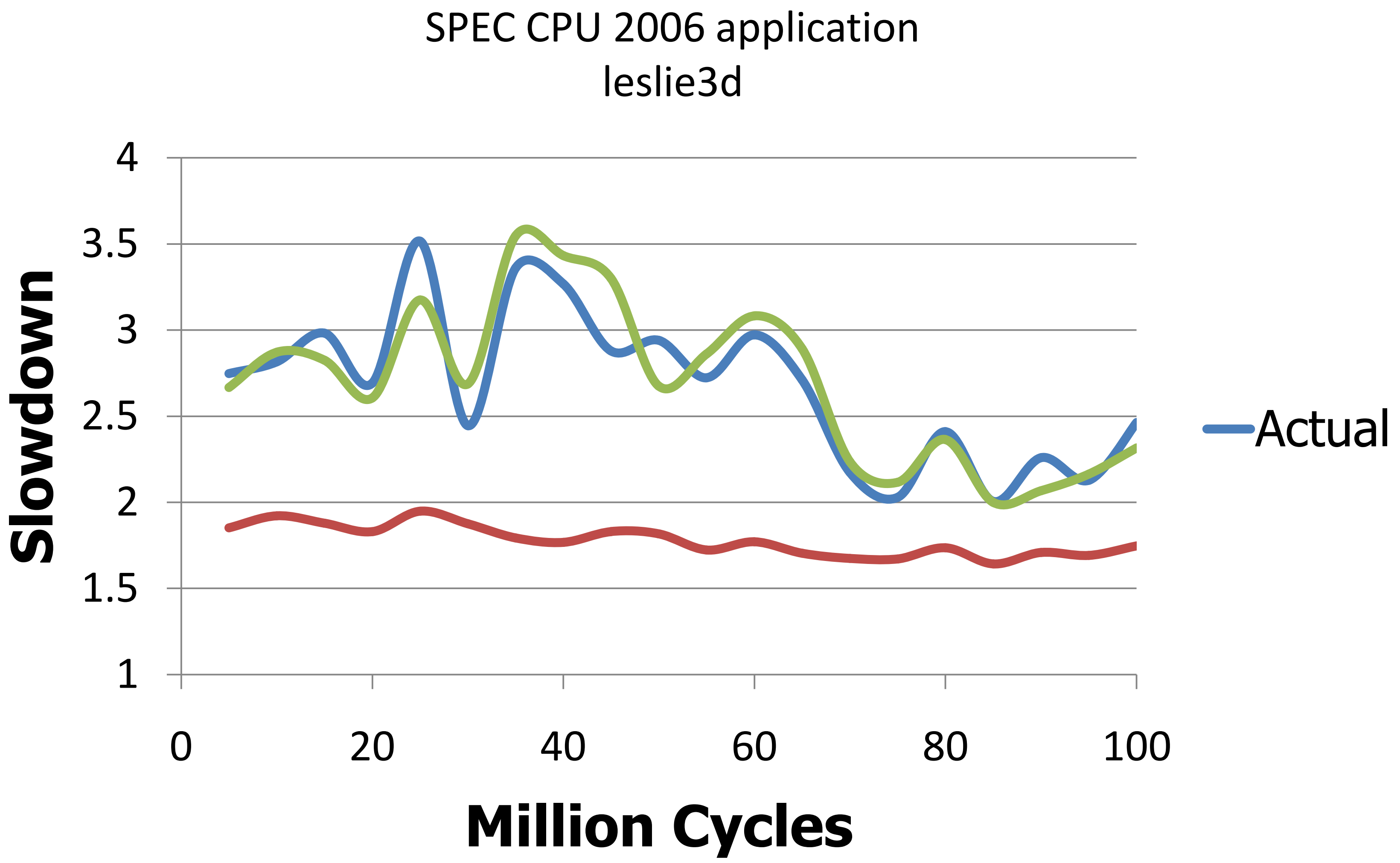
- Advantage 2:
  - STFM does not take into account compute phase for non-memory-bound applications
  - MISE accounts for compute phase → Better accuracy

Methodology

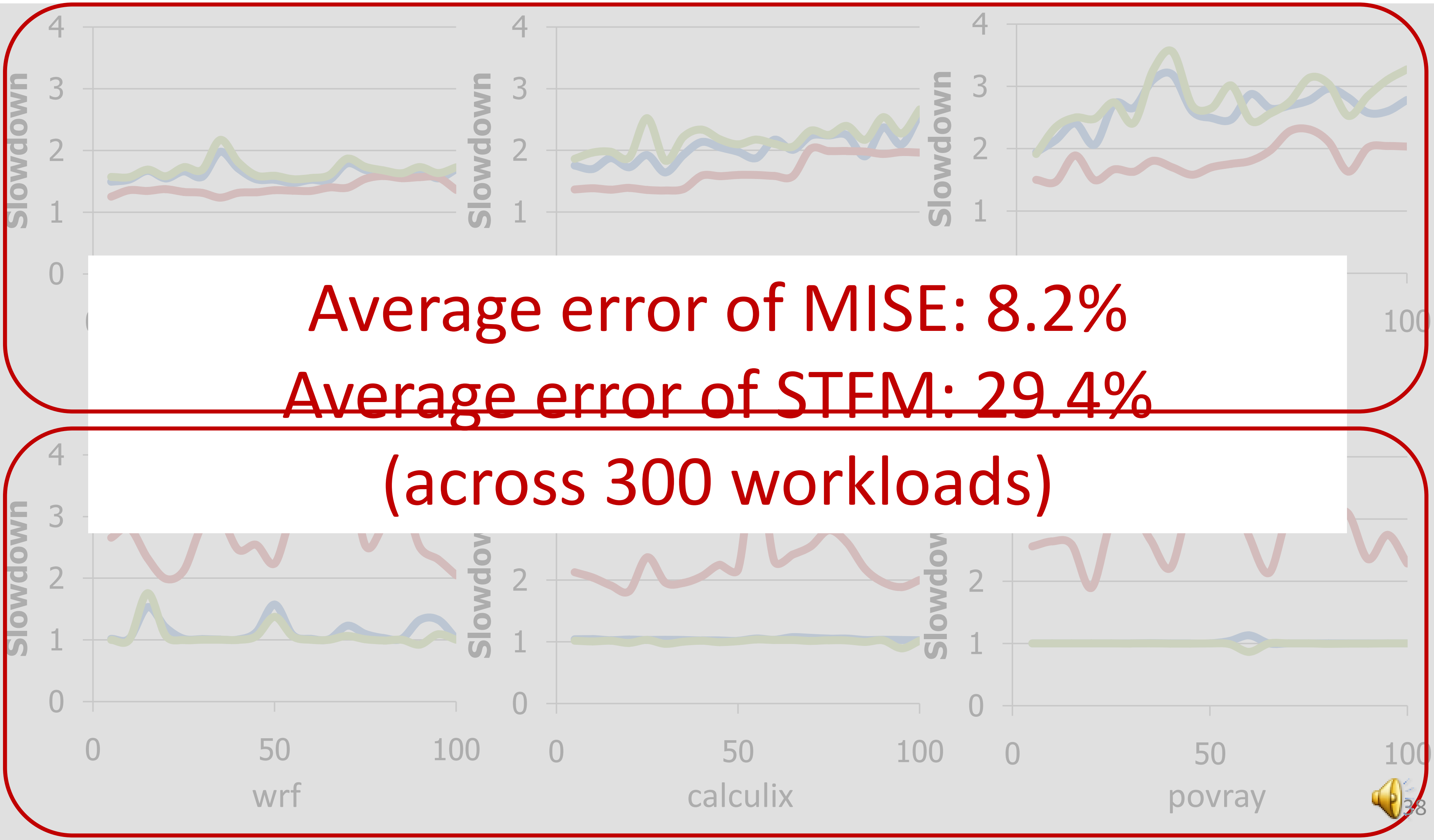
- Configuration of our simulated system
  - 4 cores
  - 1 channel, 8 banks/channel
  - DDR3 1066 DRAM
  - 512 KB private cache/core

- 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)

Predictability in the Presence of Memory Interference

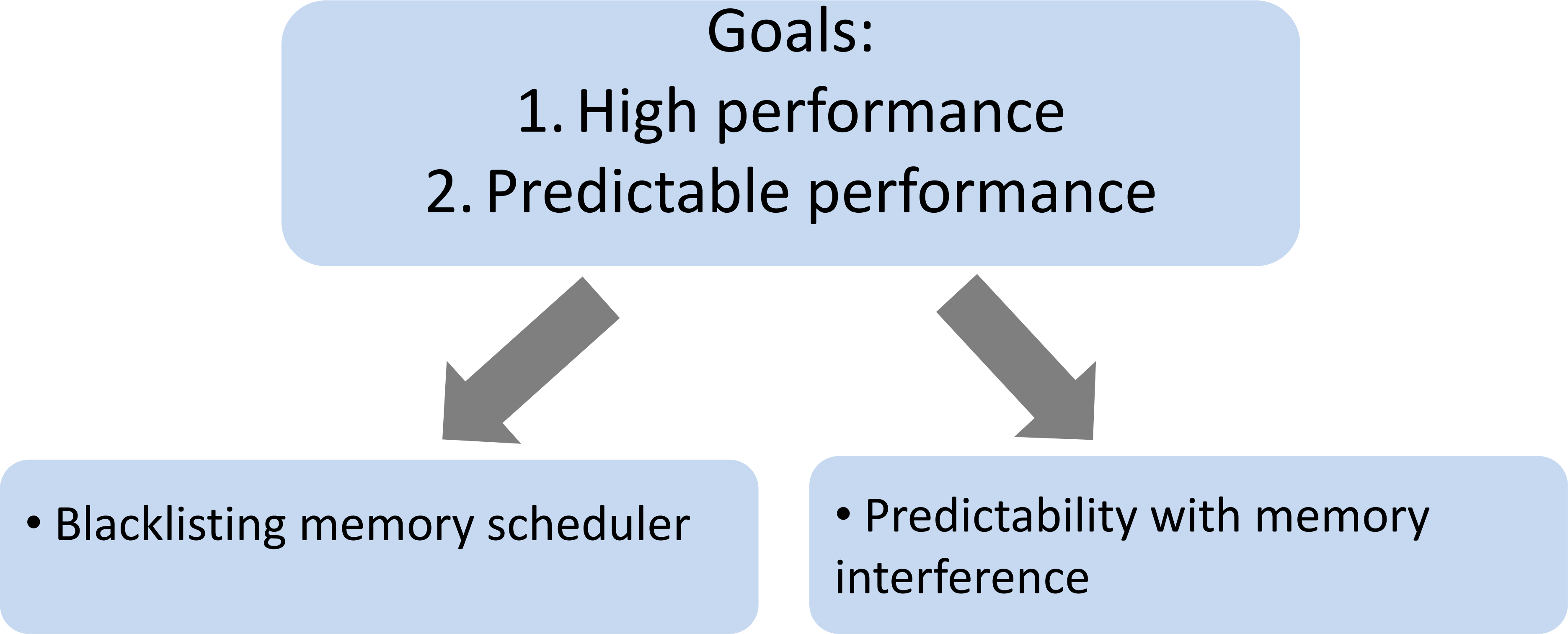
1. Estimate Slowdown
   - Key Observations
   - MISE Operation: Putting it All Together
   - Evaluating the Model
2. Control Slowdown
   - Providing Soft Slowdown Guarantees
   - Minimizing Maximum Slowdown

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

Outline



Goals:

1. High performance
2. Predictable performance

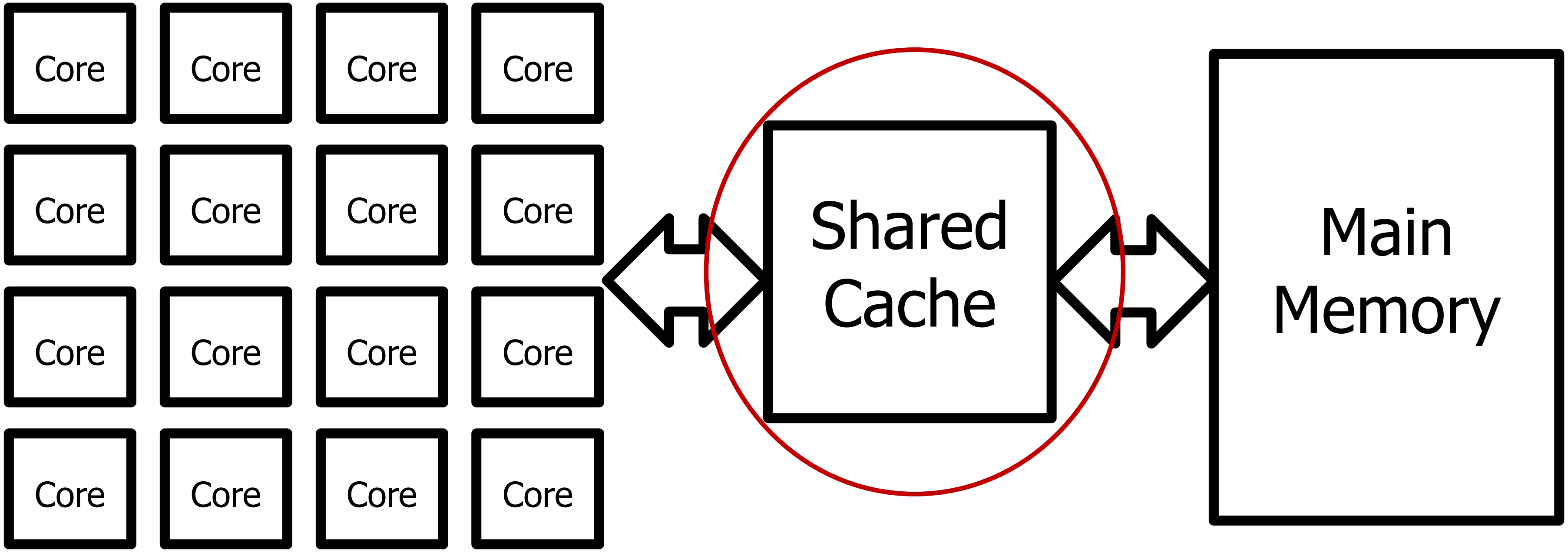
- Blacklisting memory scheduler

- Predictability with memory interference

A Recap

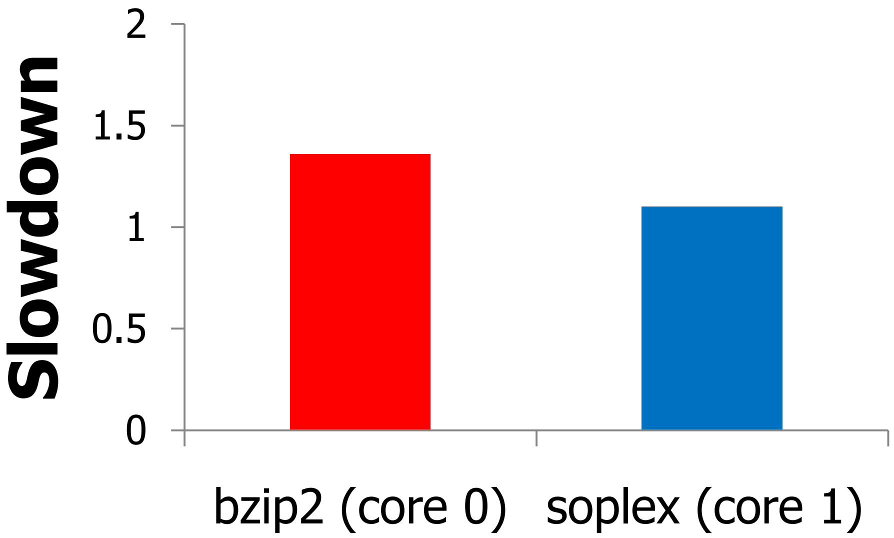
- Problem: Shared resource interference causes high and unpredictable application slowdowns
- Approach:
  - Simple mechanisms to mitigate interference
  - Slowdown estimation models
  - Slowdown control mechanisms
- Future Work:
  - Extending to shared caches

Shared Cache Interference

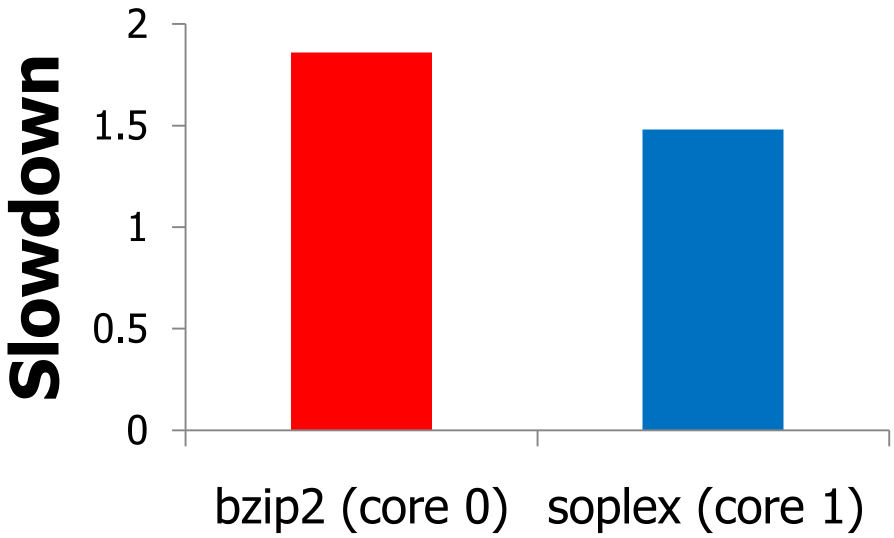


Impact of Cache Capacity Contention

Shared Main Memory



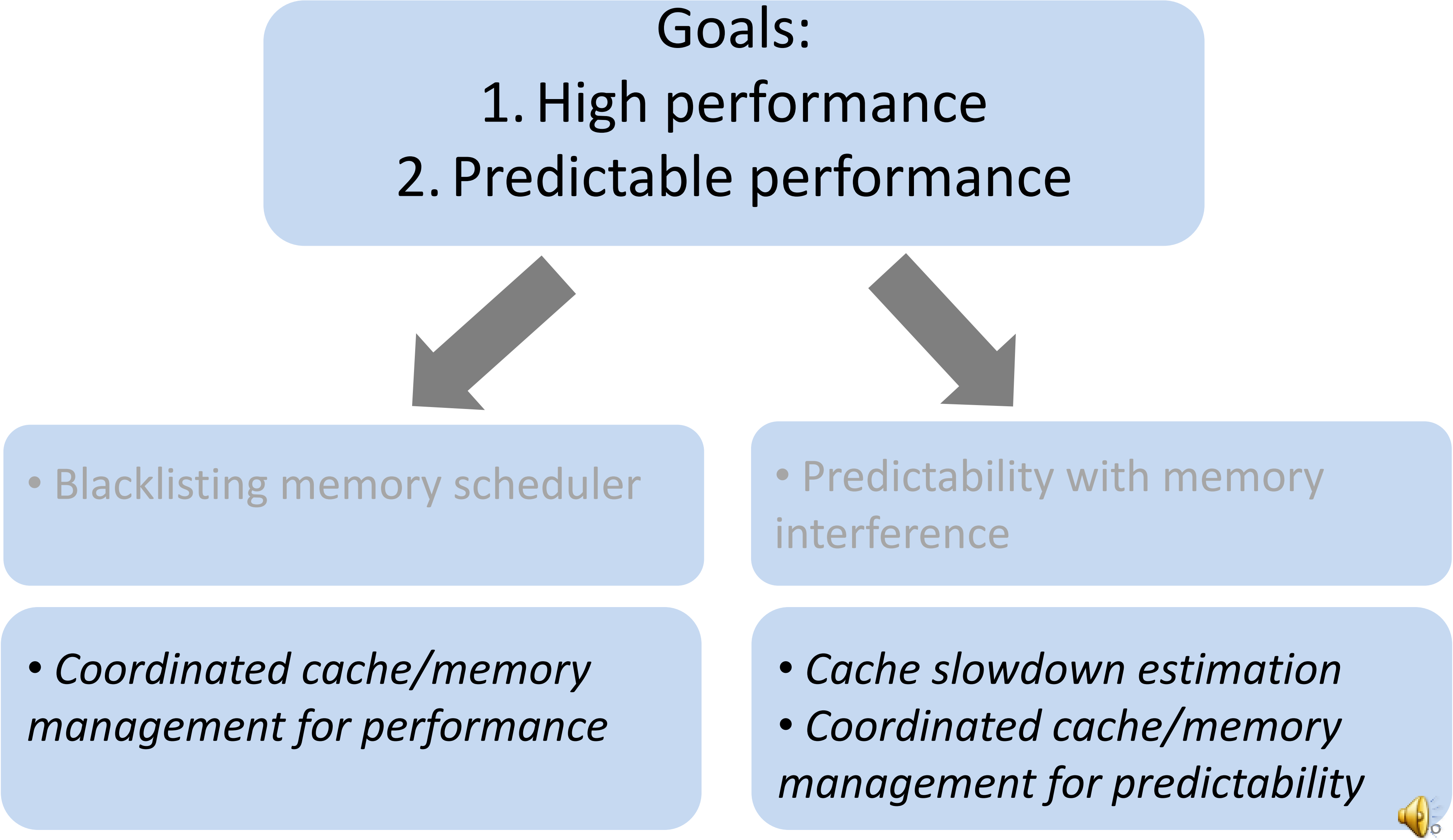
Shared Main Memory and Caches



Cache capacity interference causes high application slowdowns

Backup Slides

Outline



Goals:

1. High performance
2. Predictable performance

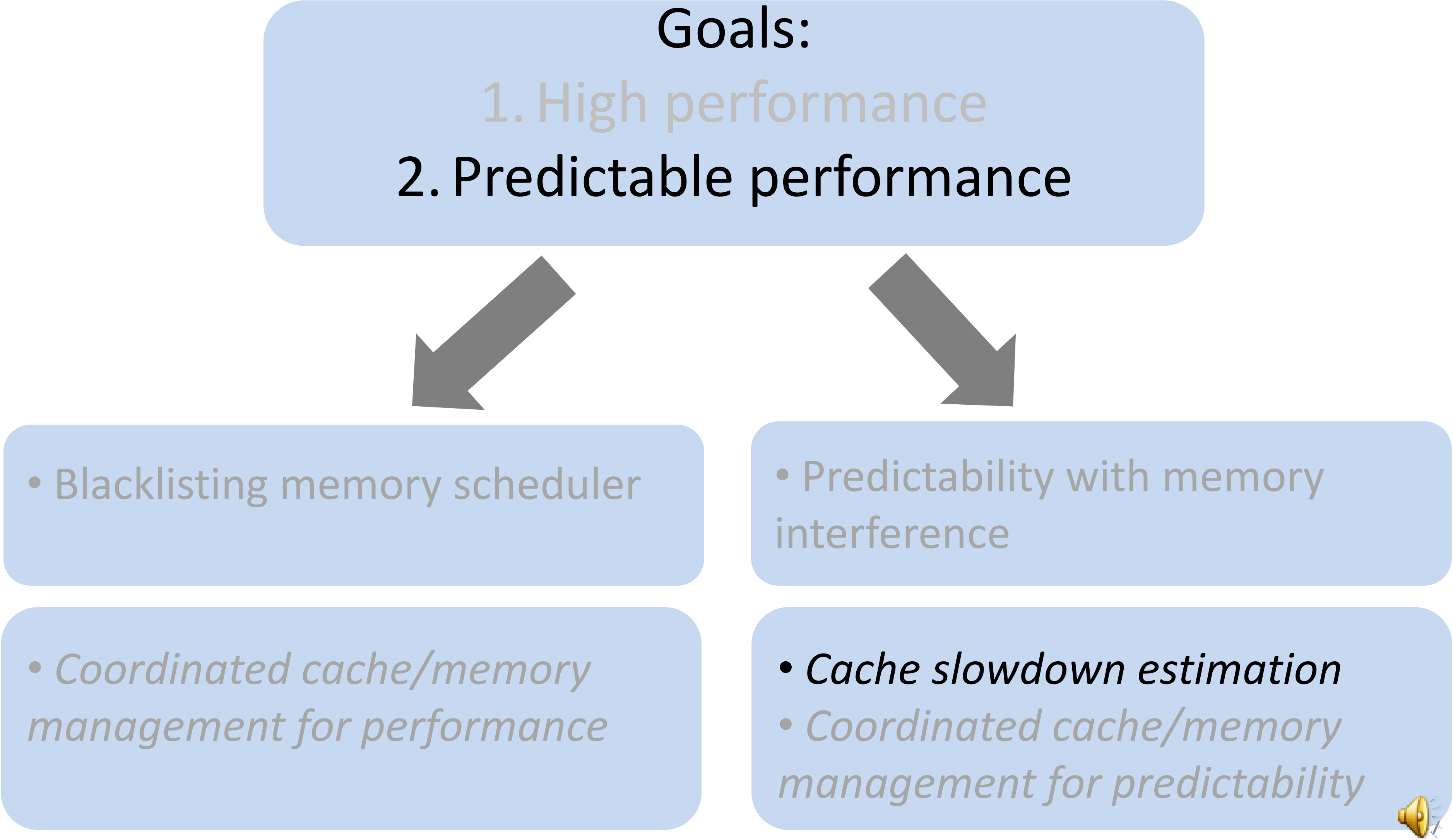
- Blacklisting memory scheduler

- Predictability with memory interference

- *Coordinated cache/memory management for performance*

- *Cache slowdown estimation*
- *Coordinated cache/memory management for predictability*

Outline



Goals:

1. High performance
2. Predictable performance

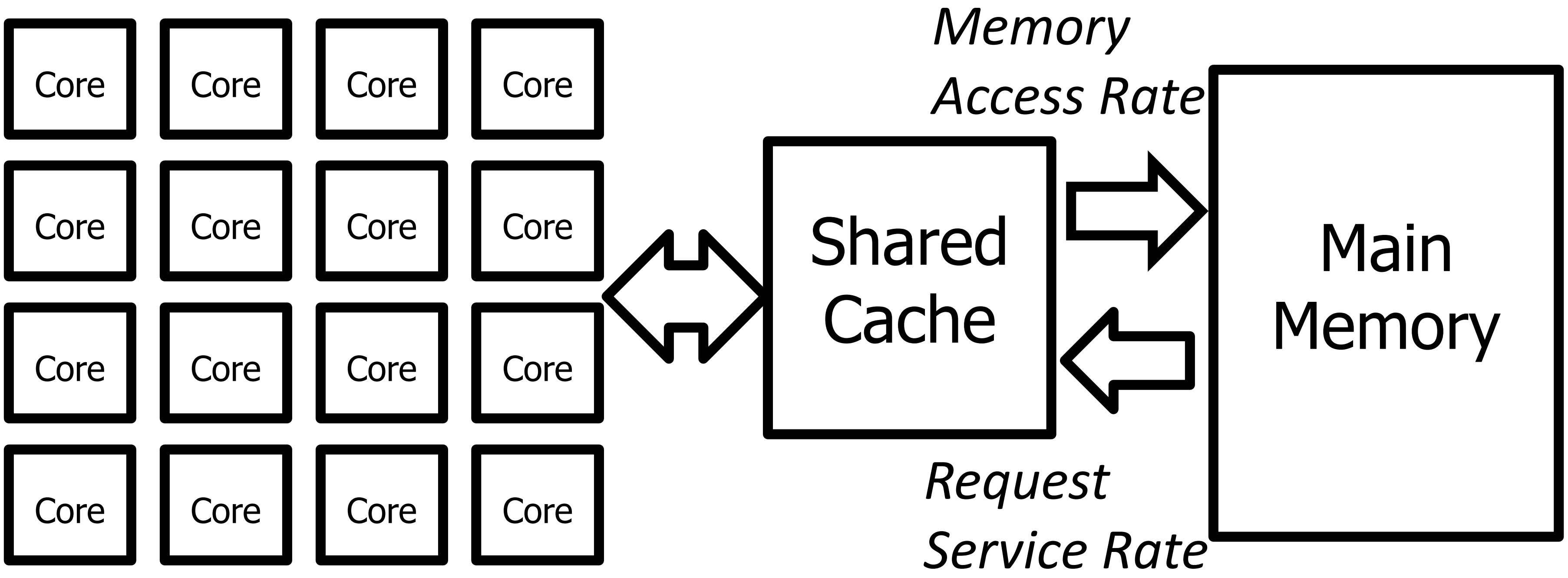
- Blacklisting memory scheduler

- *Coordinated cache/memory management for performance*

- Predictability with memory interference

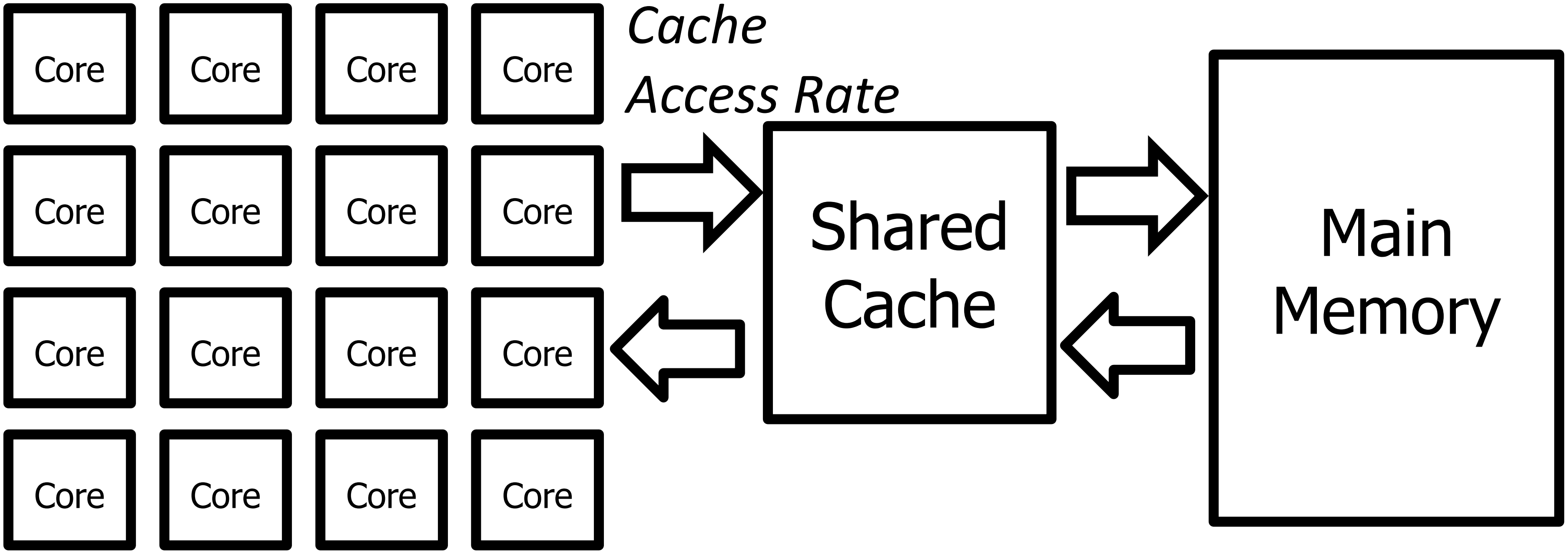
- *Cache slowdown estimation*
- *Coordinated cache/memory management for predictability*

Request Service vs. Memory Access



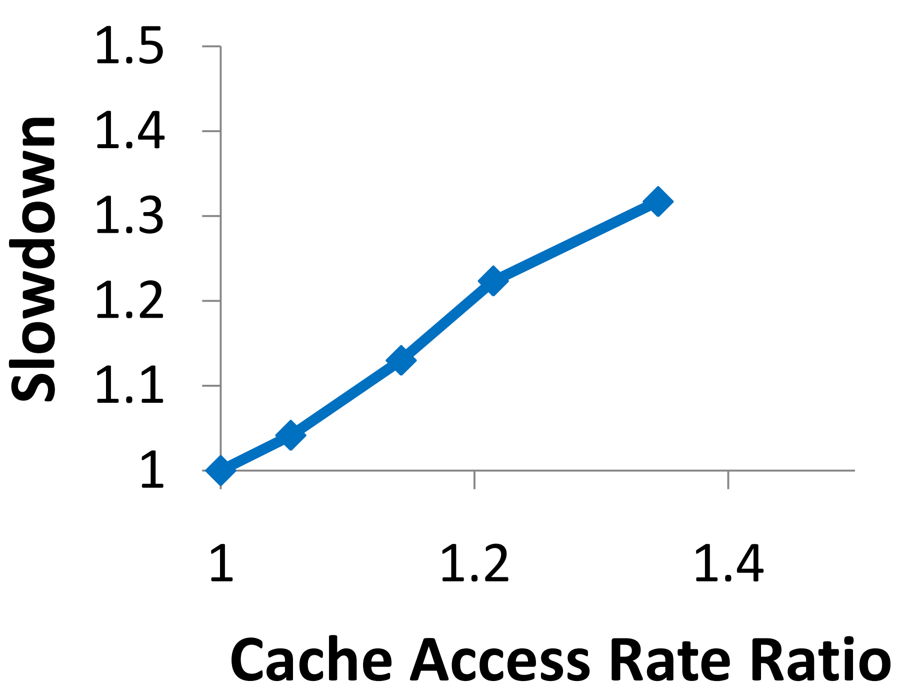
Request service and access rates tightly coupled

Estimating Cache and Memory Slowdowns Through Cache Access Rates

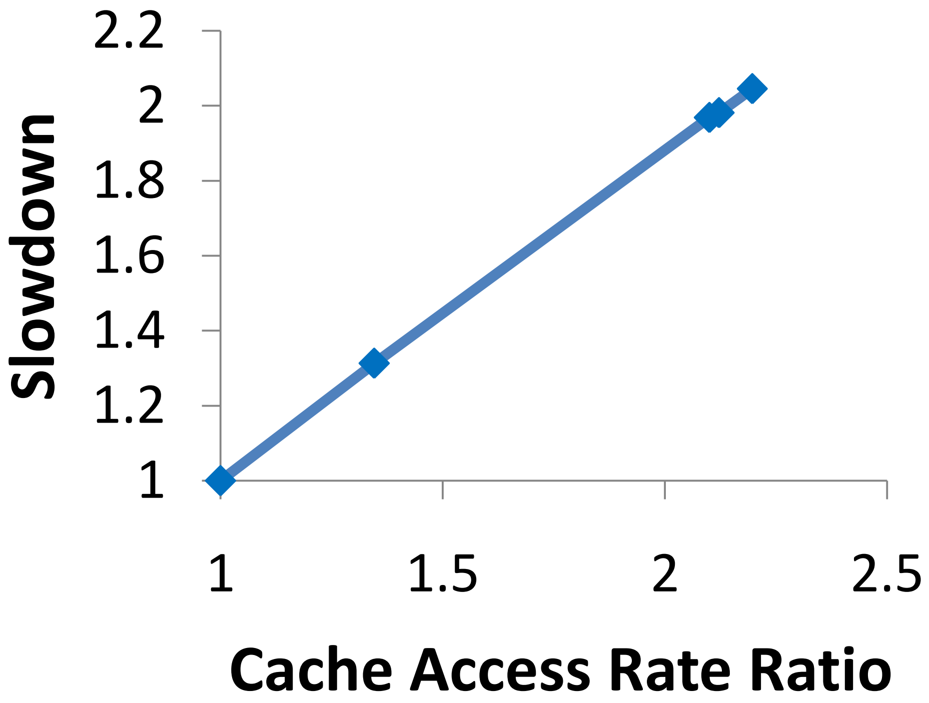


$$\text{Slowdown} = \frac{\text{Cache Access Rate}_{\text{Alone}}}{\text{Cache Access Rate}_{\text{Shared}}}$$

Cache Access Rate vs. Slowdown



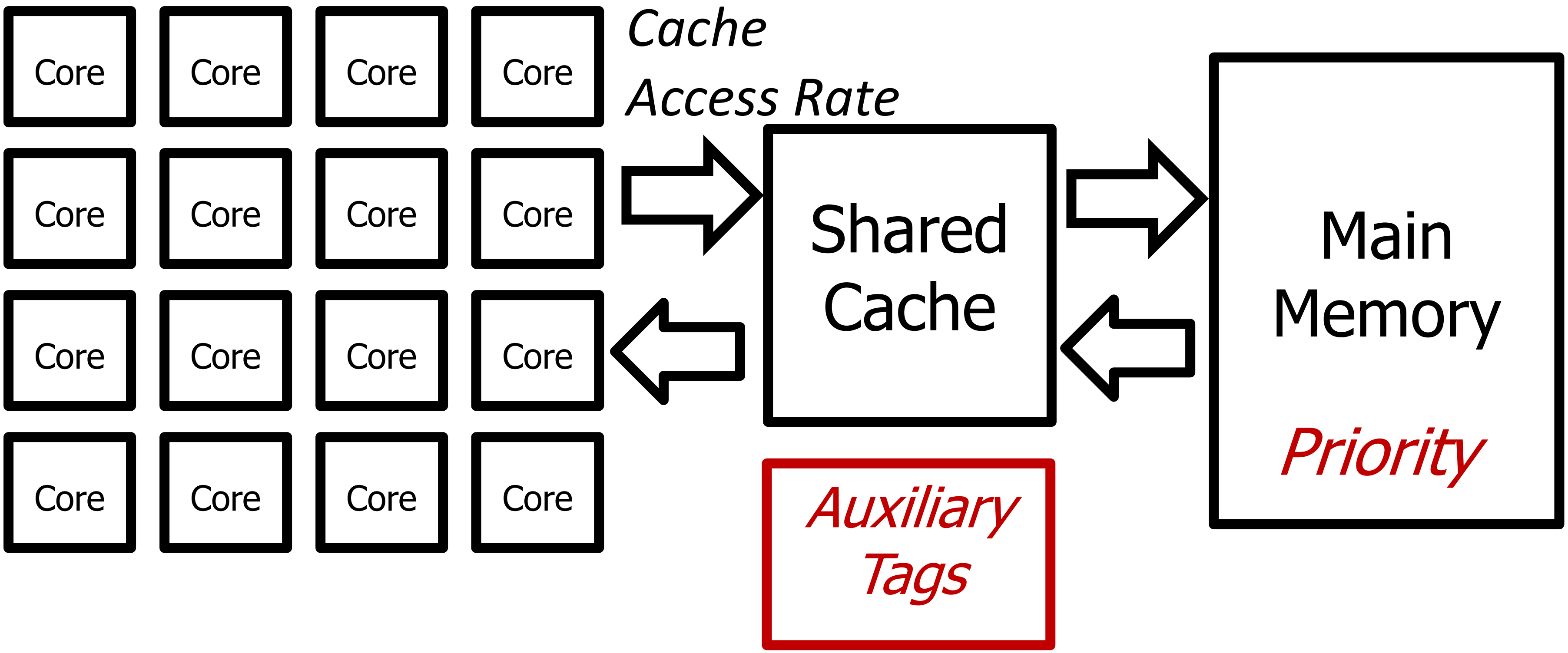
bzip2



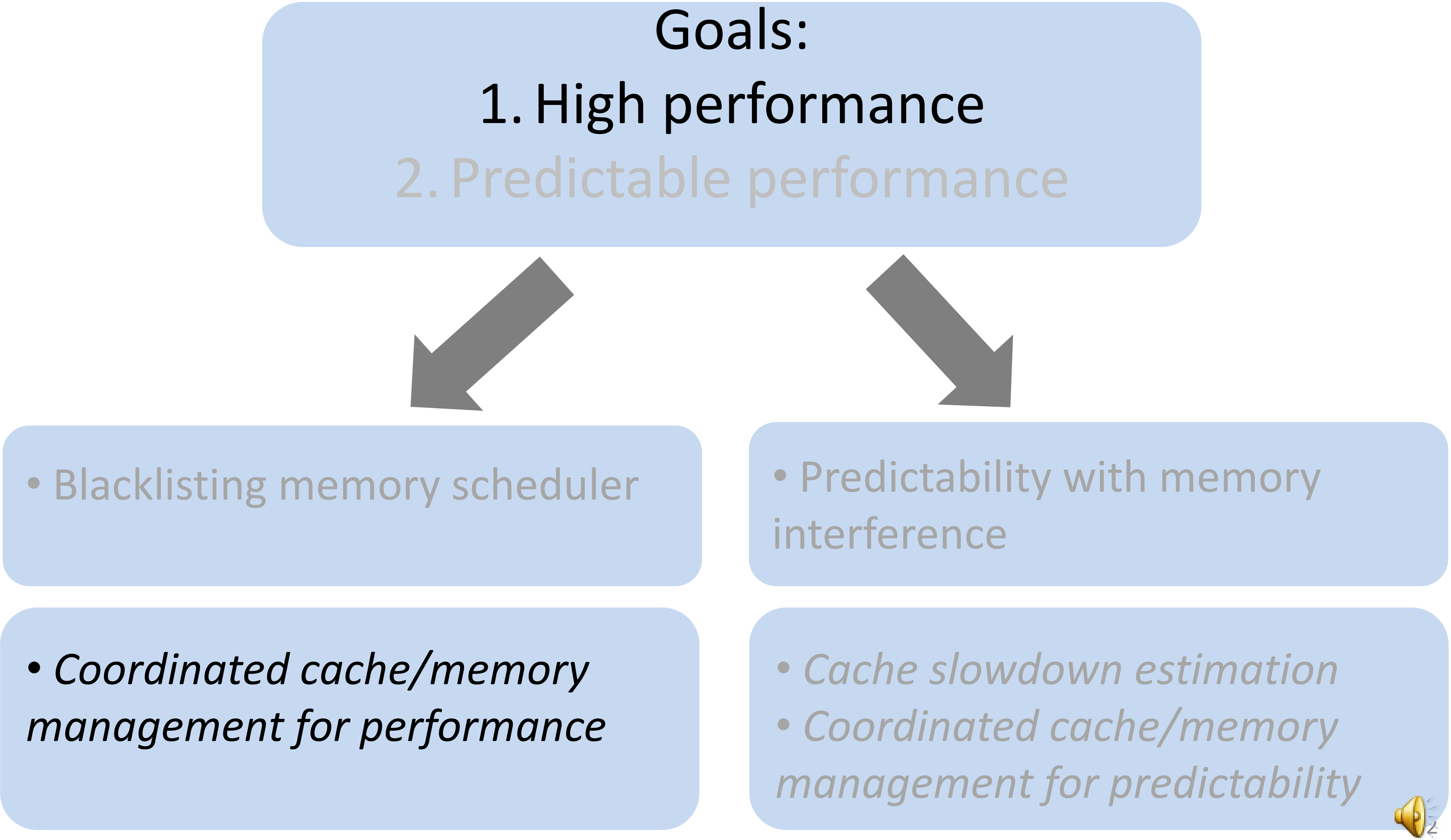
xalancbmk

Challenge

How to estimate alone cache access rate?



Outline



Goals:

1. High performance
2. Predictable performance

- Blacklisting memory scheduler

- Predictability with memory interference

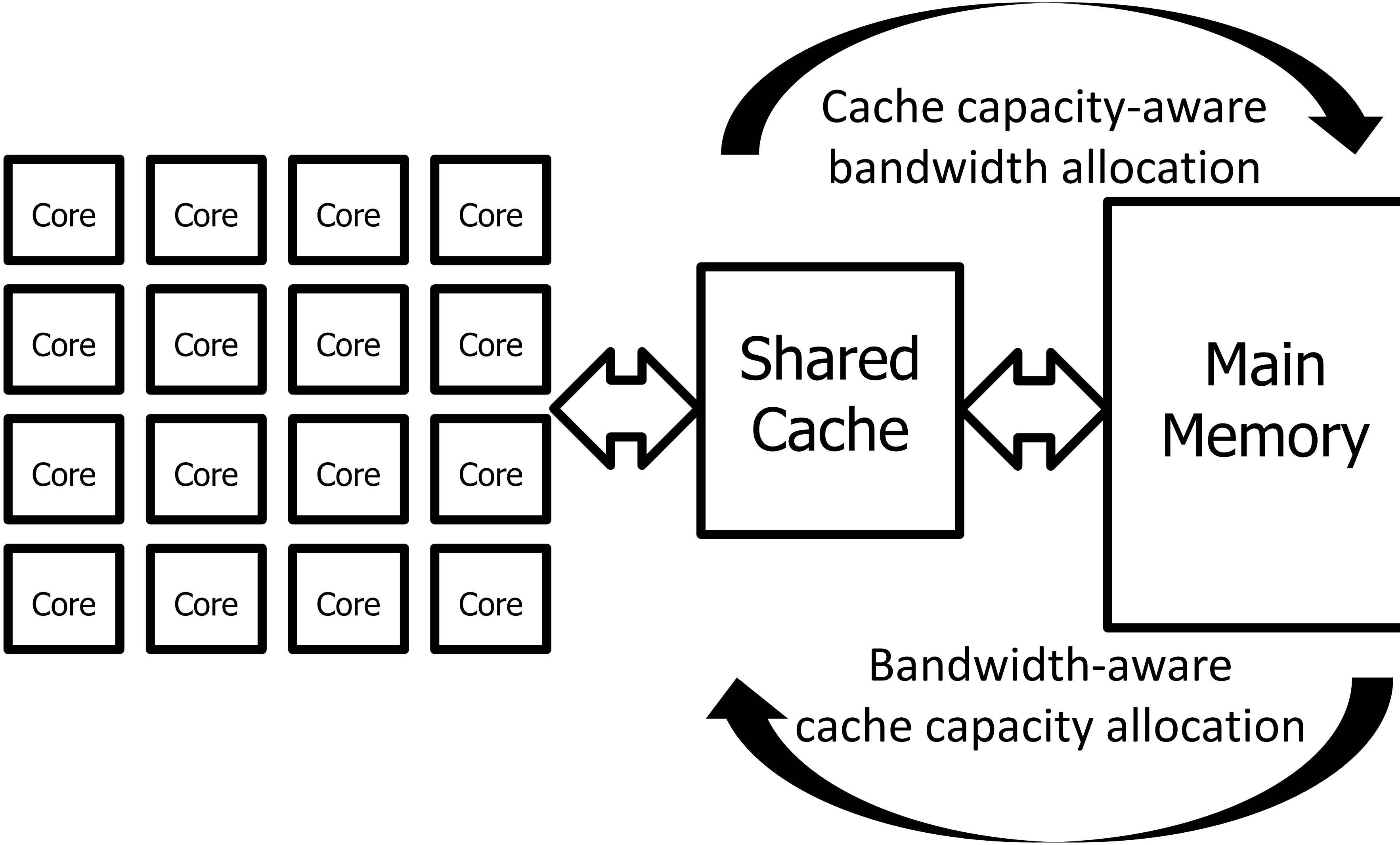
- *Coordinated cache/memory management for performance*

- *Cache slowdown estimation*
- *Coordinated cache/memory management for predictability*

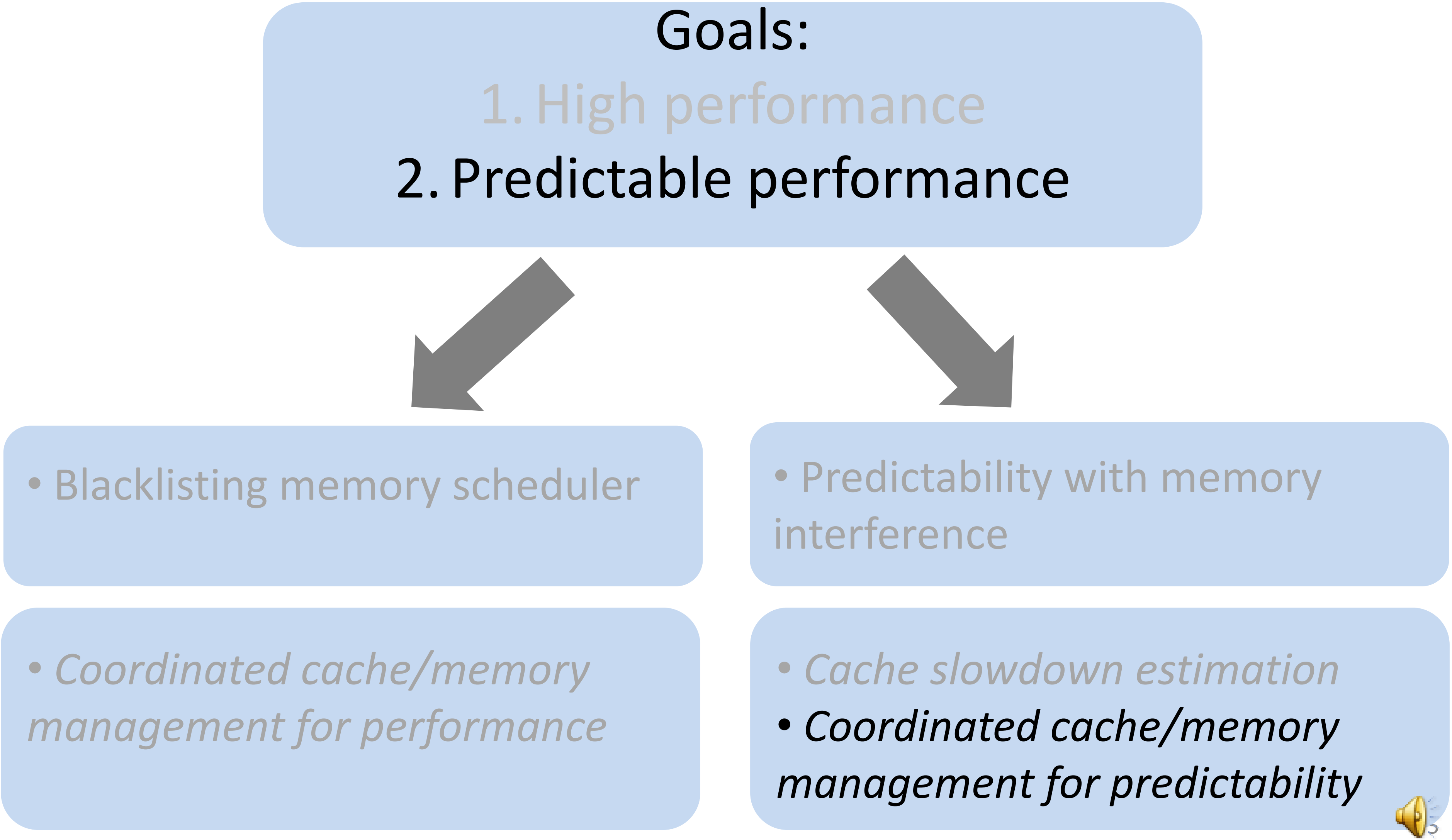
Leveraging Slowdown Estimates for Performance Optimization

- How do we leverage slowdown estimates to achieve high performance by allocating
  - Memory bandwidth?
  - Cache capacity?
- Leverage other metrics along with slowdowns
  - Memory intensity
  - Cache miss rates

Coordinated Resource Allocation Schemes



Outline



Goals:

1. High performance
2. Predictable performance

- Blacklisting memory scheduler

- Predictability with memory interference

- *Coordinated cache/memory management for performance*

- *Cache slowdown estimation*
- *Coordinated cache/memory management for predictability*

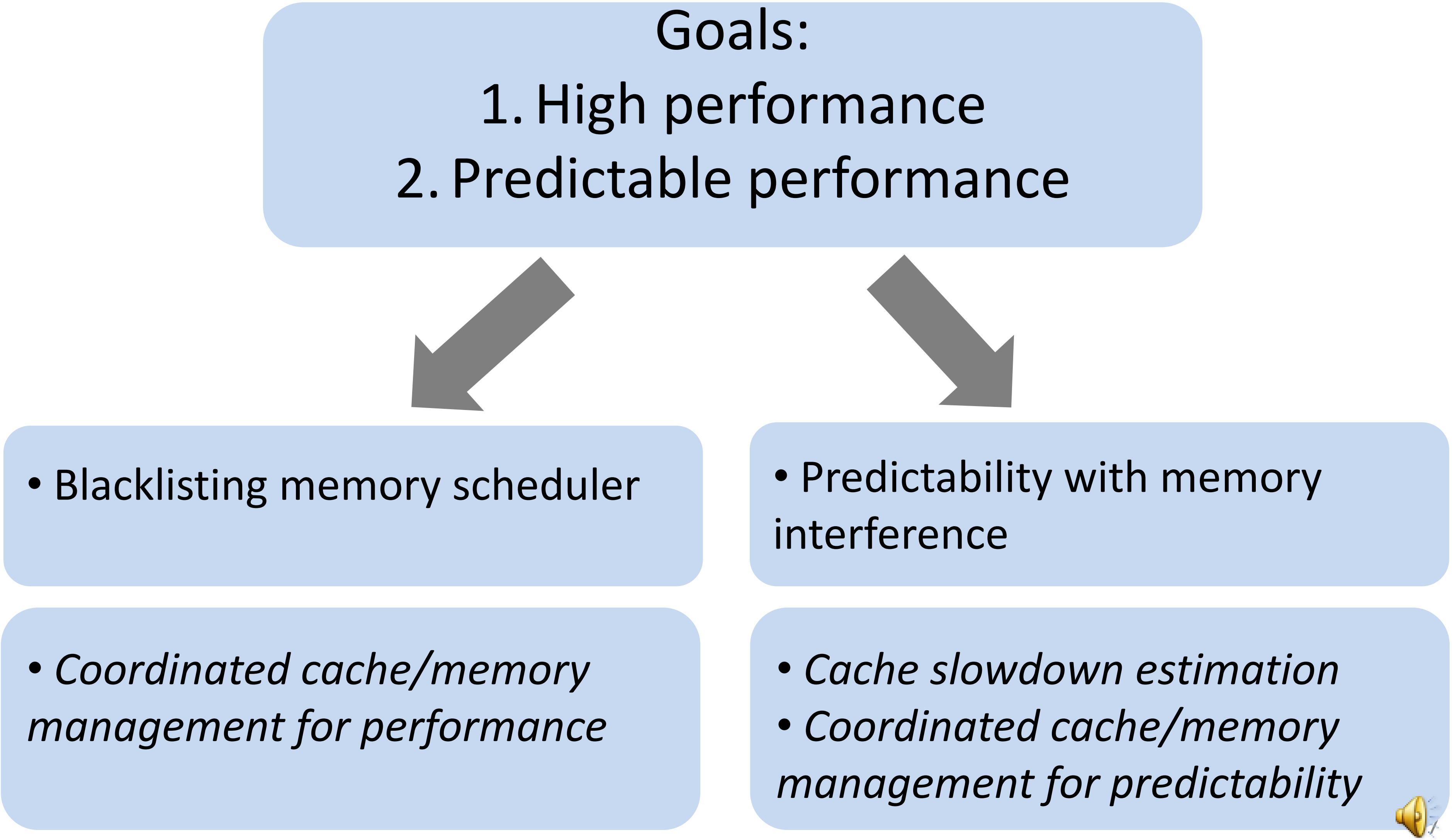
Coordinated Resource Management Schemes for Predictable Performance

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

Outline



Goals:

1. High performance
2. Predictable performance

- Blacklisting memory scheduler

- Predictability with memory interference

- *Coordinated cache/memory management for performance*

- *Cache slowdown estimation*
- *Coordinated cache/memory management for predictability*

Timeline

	2014									2015				
	Apr.	May	Jun.	Jul.	Aug.	Sep.	Oct.	Nov.	Dec.	Jan.	Feb.	Mar.	Apr.	May
Cache slowdown estimation (75% Goal)	🟦	🟦												
Coordinated cache/memory management for performance (100% Goal)			🟦	🟦	🟦	🟦	🟦							
Coordinated cache/memory management for predictability (125% Goal)								🟦	🟦	🟦	🟦			
Writeup thesis and defend												🟦	🟦	🟦

Summary

- Problem: Shared resource interference causes high and unpredictable application slowdowns
- Goals: High and predictable performance
- Our Approach:
  - Simple mechanisms to mitigate interference
  - Slowdown estimation models
  - Coordinated cache/memory management