The Blacklisting Memory Scheduler

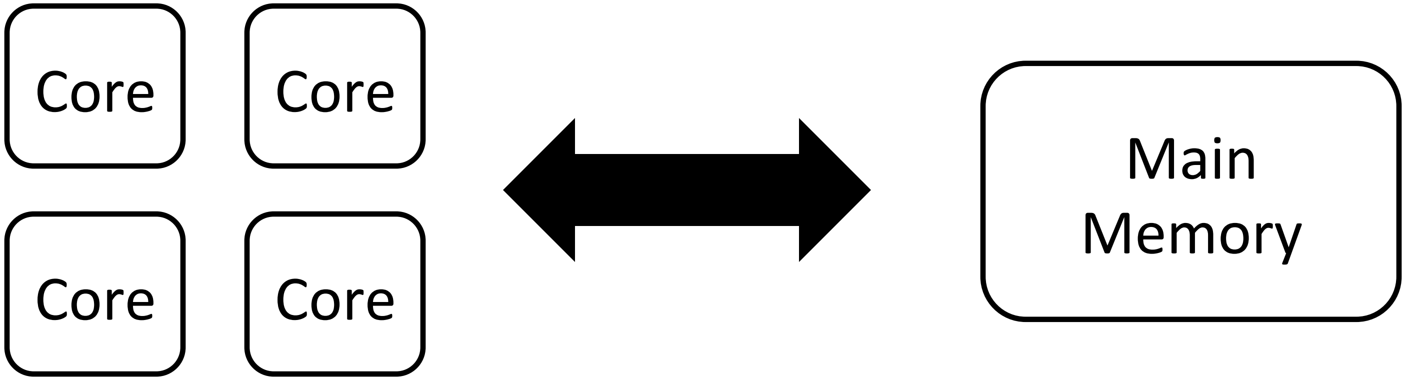
Achieving High Performance and Fairness at Low Cost

Lavanya Subramanian, Donghyuk Lee,
Vivek Seshadri, Harsha Rastogi, Onur Mutlu

SAFARI

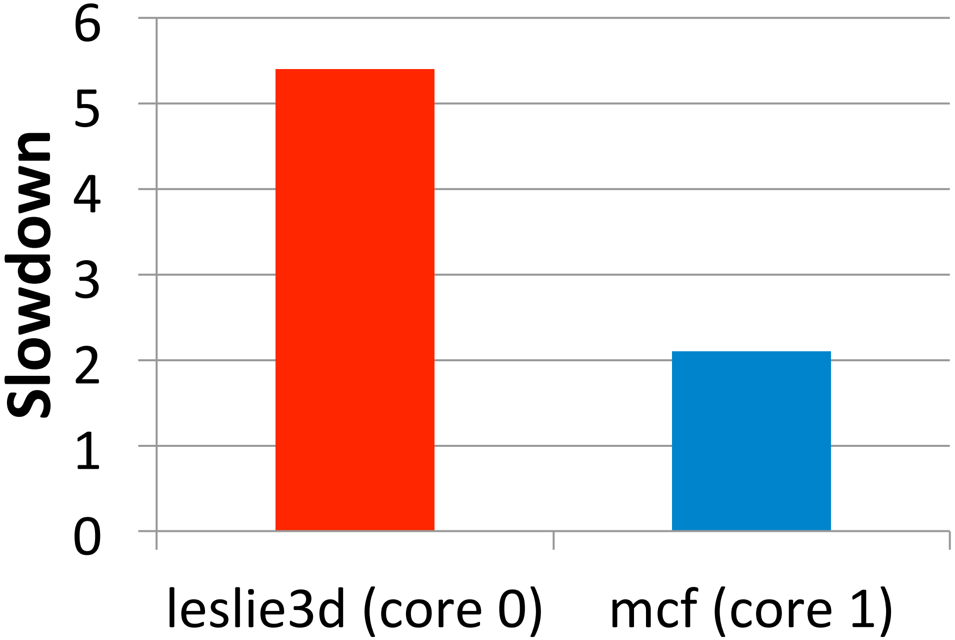
Carnegie Mellon

Main Memory Interference Problem



Causes interference between applications' requests

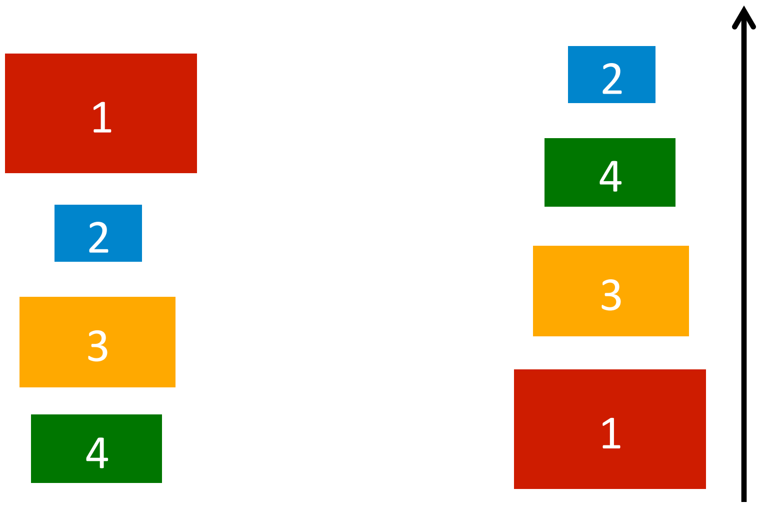
Inter-Application Interference Results in Performance Degradation

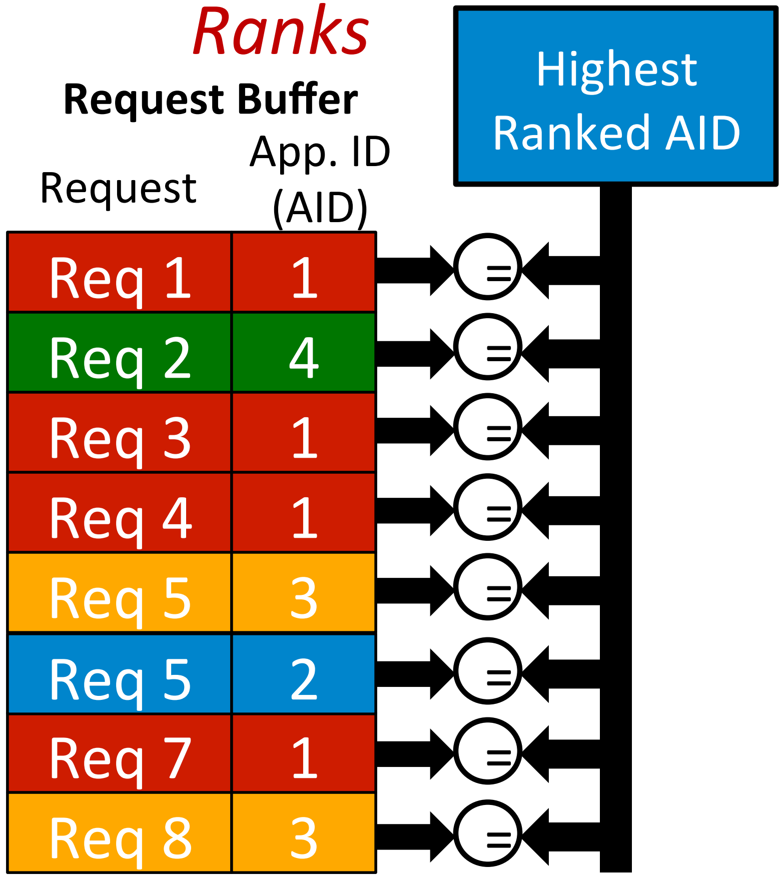


High application slowdowns due to interference

Tackling Inter-Application Interference: Application-aware Memory Scheduling

Monitor *Rank* *Enforce Ranks*



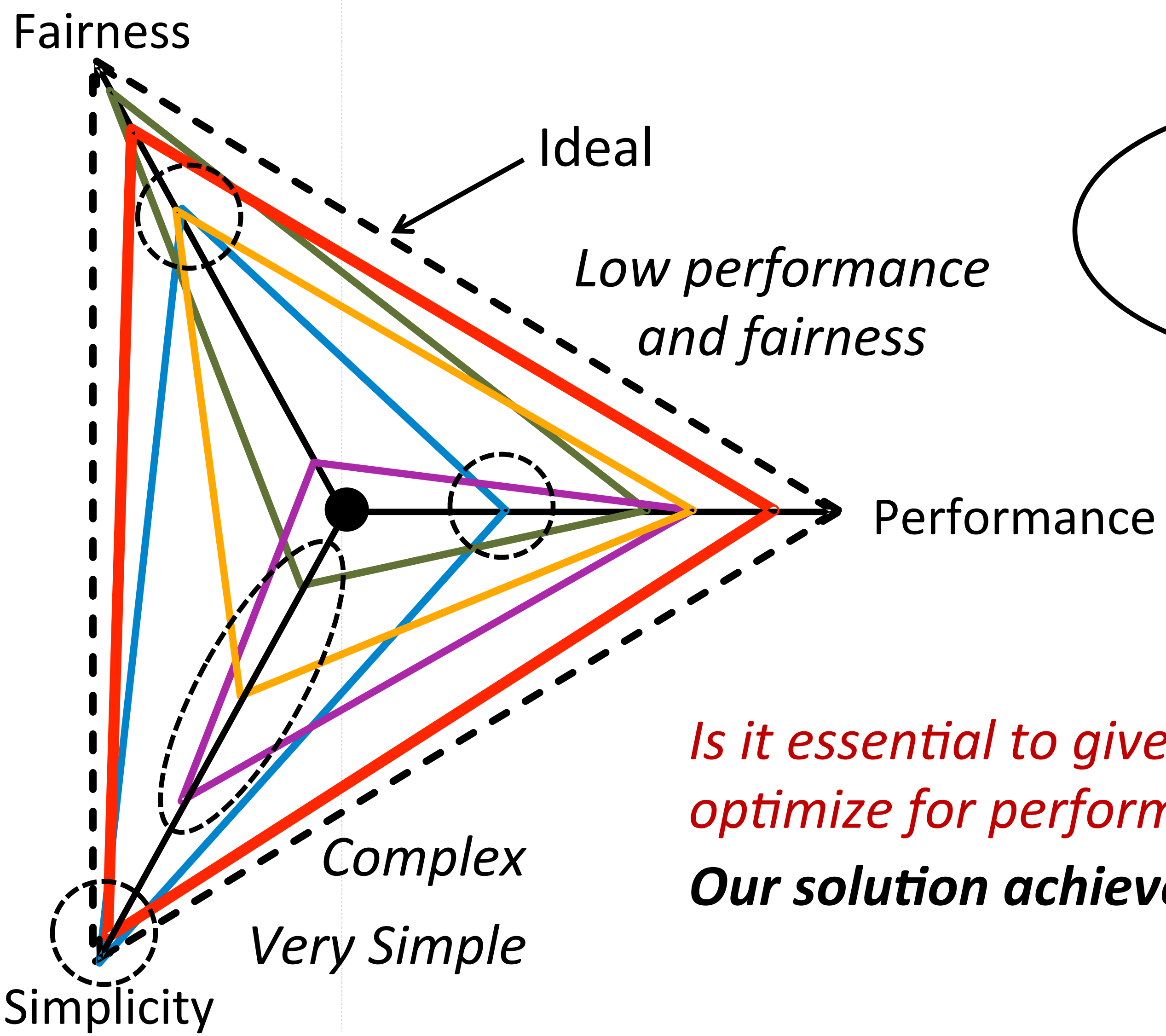


Request Buffer

Request	App. ID (AID)
Req 1	1
Req 2	4
Req 3	1
Req 4	1
Req 5	3
Req 5	2
Req 7	1
Req 8	3

Full ranking increases critical path latency and area significantly to improve performance and fairness

Performance vs. Fairness vs. Simplicity



Is it essential to give up simplicity to optimize for performance and/or fairness?

Our solution achieves all three goals

Outline

- Introduction
- Problems with Application-aware Schedulers
- Key Observations
- The Blacklisting Memory Scheduler Design
- Evaluation
- Conclusion

Outline

- Introduction
- **Problems with Application-aware Schedulers**
- Key Observations
- The Blacklisting Memory Scheduler Design
- Evaluation
- Conclusion

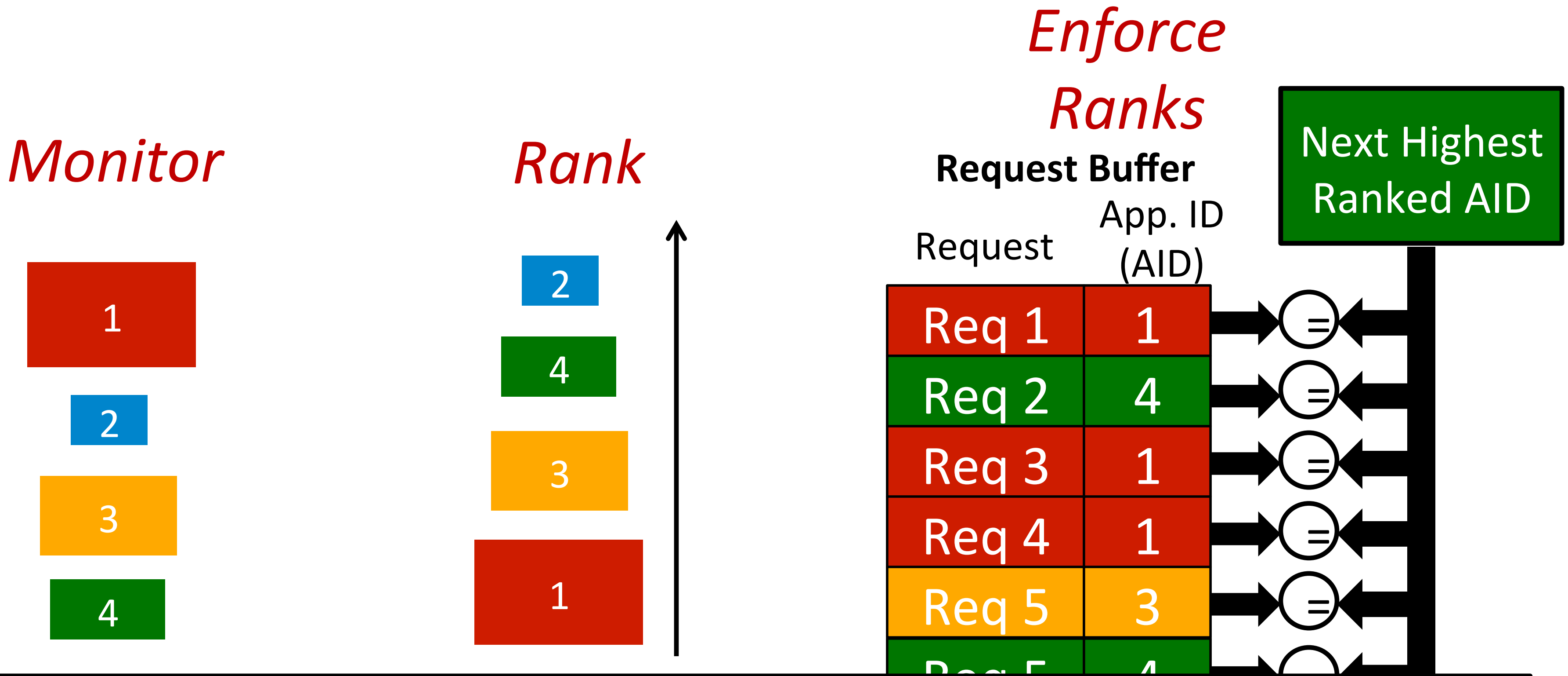
Problems with Previous Application-aware Memory Schedulers

1. Full ranking increases hardware complexity
2. Full ranking causes unfair slowdowns

Problems with Previous Application-aware Memory Schedulers

1. **Full ranking increases hardware complexity**
2. Full ranking causes unfair slowdowns

Ranking Increases Hardware Complexity



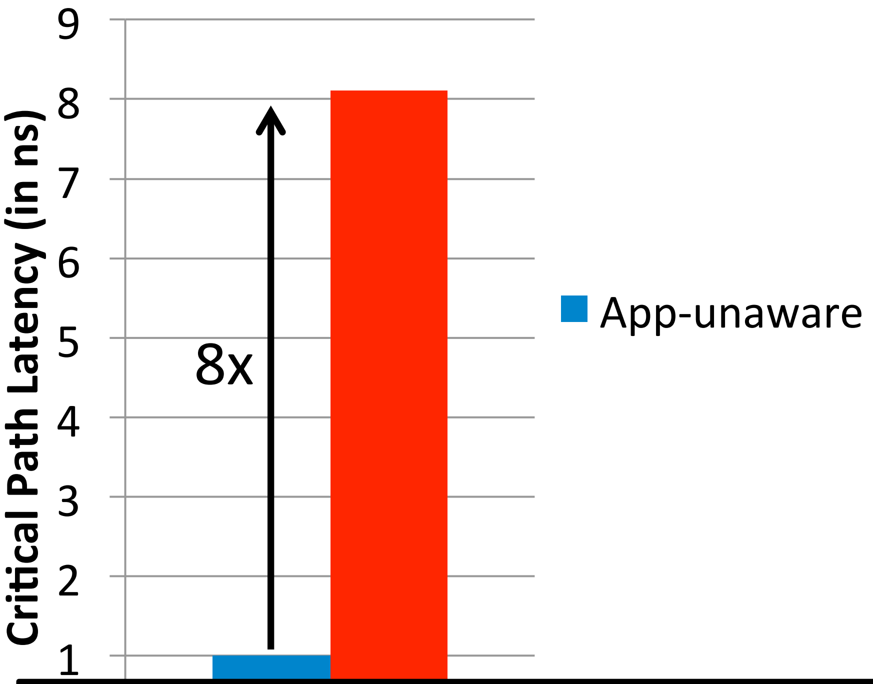
Request Buffer

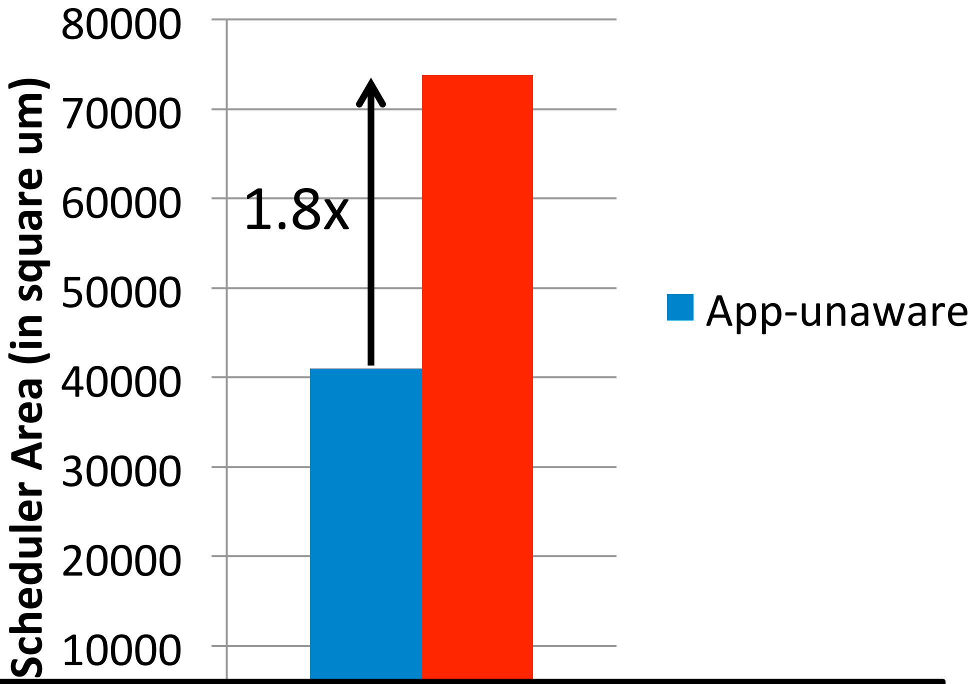
Request	App. ID (AID)
Req 1	1
Req 2	4
Req 3	1
Req 4	1
Req 5	3
Req 5	4

Hardware complexity increases with application/core count

Ranking Increases Hardware Complexity

From synthesis of RTL implementations using a 32nm library





Ranking-based application-aware schedulers incur high hardware cost

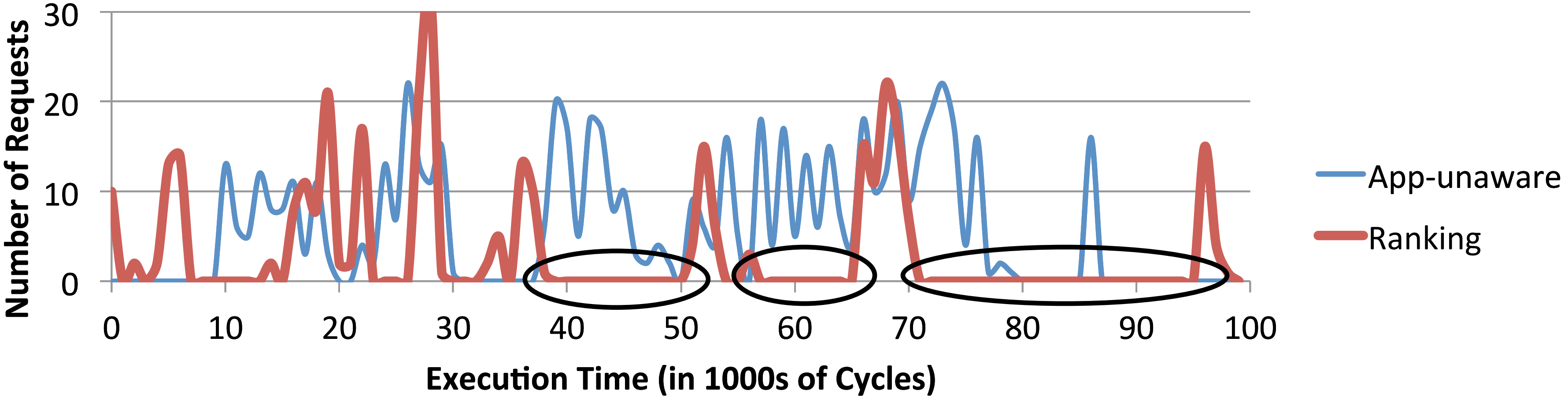
SAFARI

Problems with Previous Application-aware Memory Schedulers

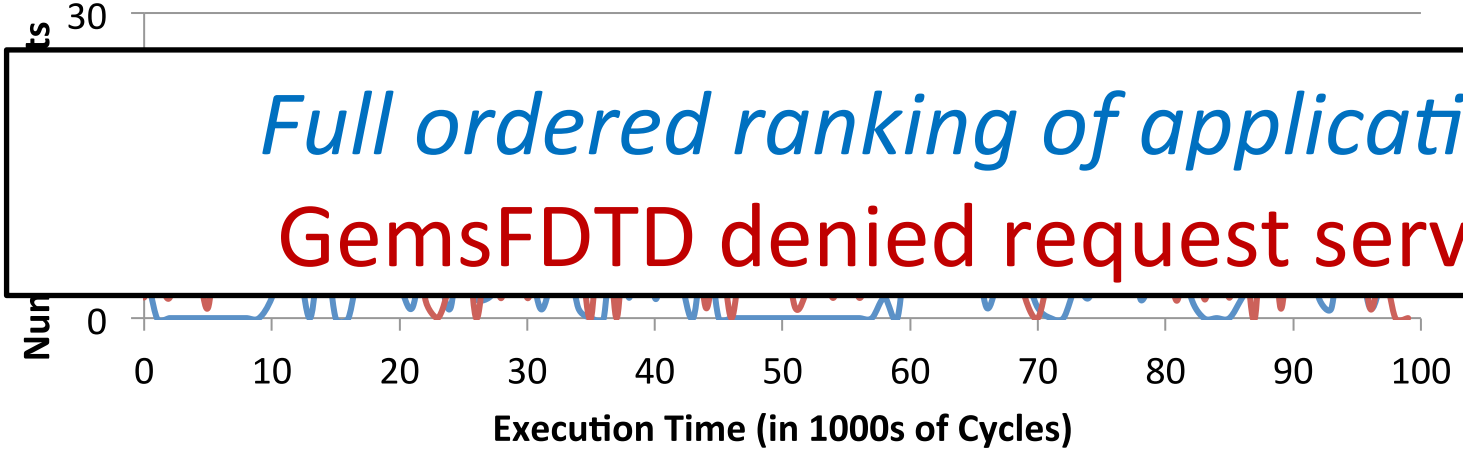
1. Full ranking increases hardware complexity
2. **Full ranking causes unfair slowdowns**

Ranking Causes Unfair Slowdowns

GemsFDTD *(high memory intensity)*



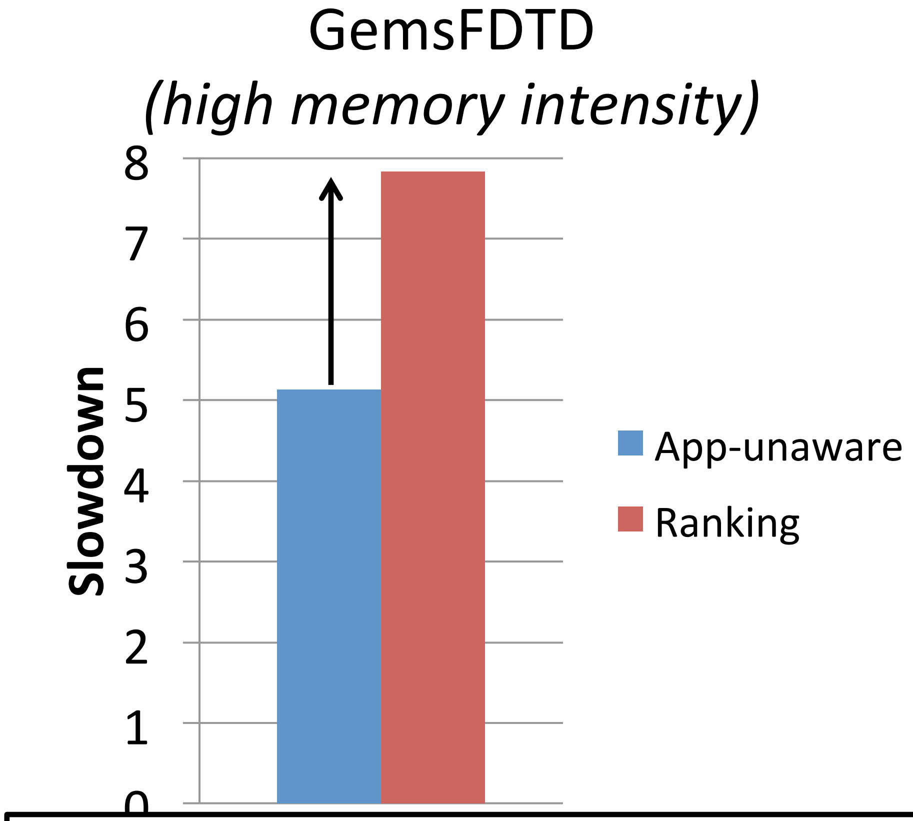
sjeng *(low memory intensity)*

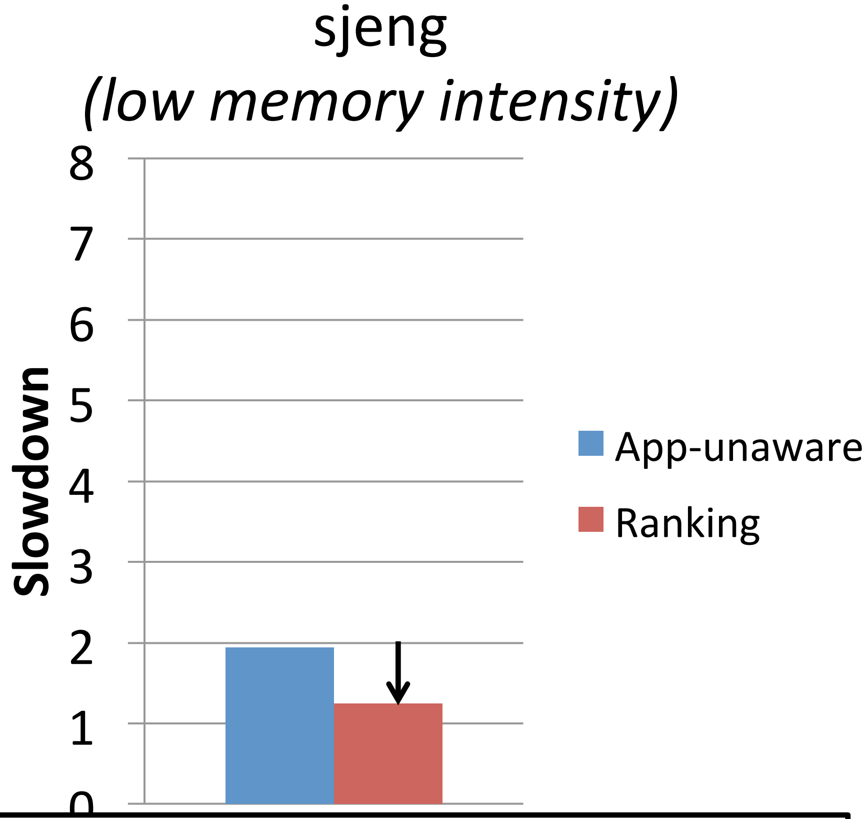


Full ordered ranking of applications

GemsFDTD denied request service

Ranking Causes Unfair Slowdowns





Ranking-based application-aware schedulers cause unfair slowdowns

Problems with Previous Application-aware Memory Schedulers

1. Full ranking increases hardware complexity
2. Full ranking causes unfair slowdowns

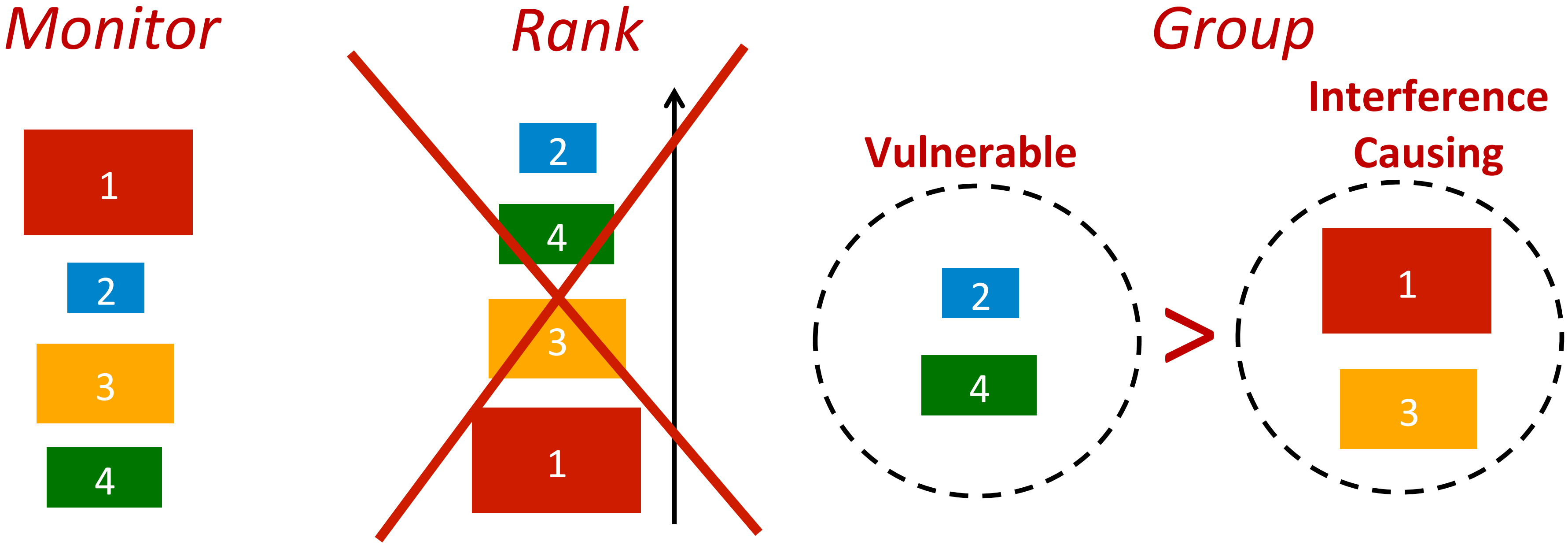
Our Goal: Design a memory scheduler with *Low Complexity, High Performance, and Fairness*

Outline

- Introduction
- Problems with application-aware schedulers
- **Key Observations**
- The Blacklisting Memory Scheduler Design
- Evaluation
- Conclusion

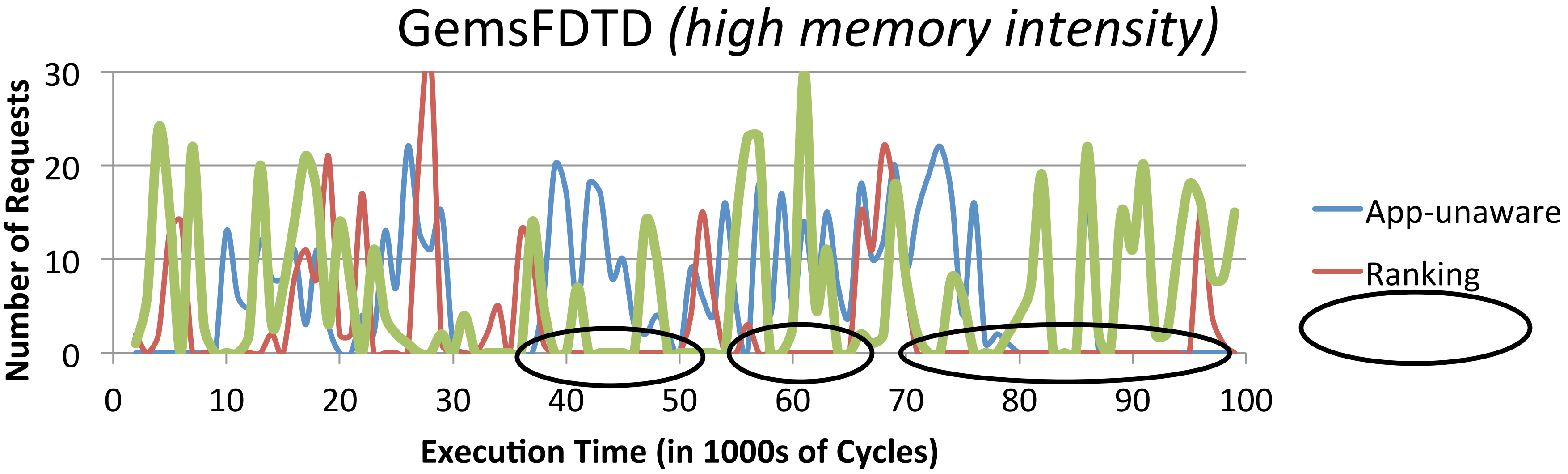
Key Observation 1: Group Rather Than Rank

Observation 1: Sufficient to separate applications into two groups, rather than do full ranking

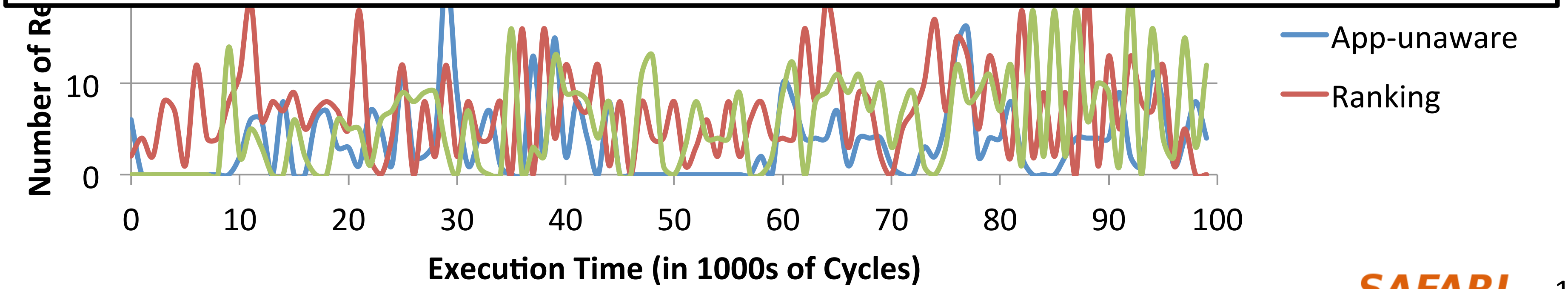


Benefit 1: Low complexity compared to ranking

Key Observation 1: Group Rather Than Rank


GemsFDTD *(high memory intensity)*

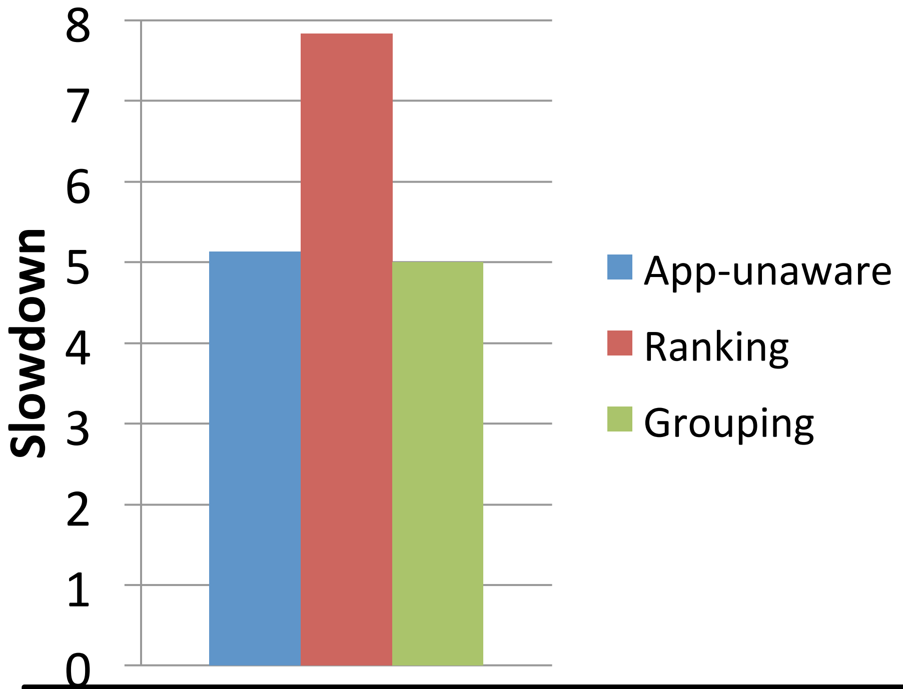
No denial of request service



Key Observation 1: Group Rather Than Rank

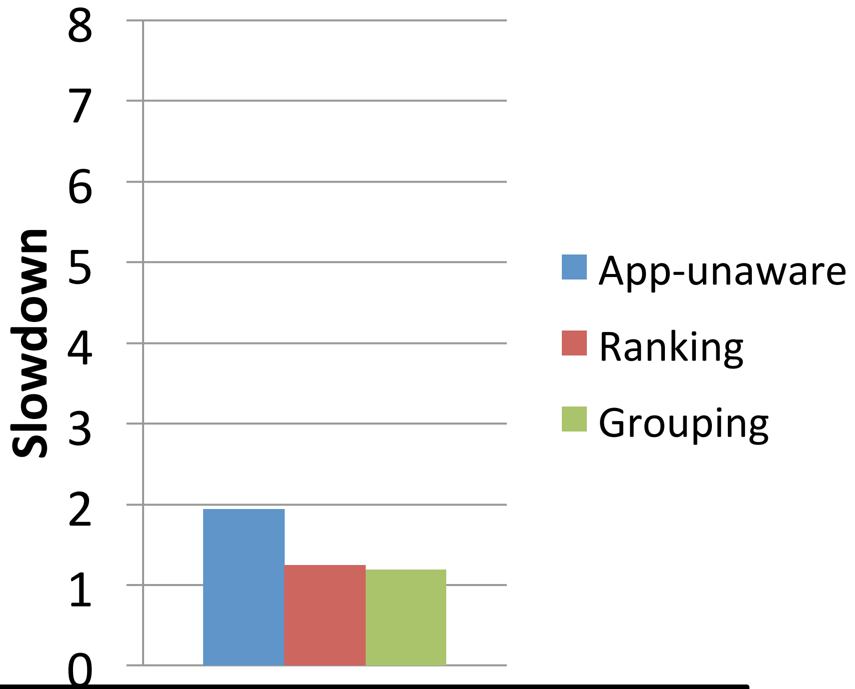
GemsFDTD

(high memory intensity)



sjeng

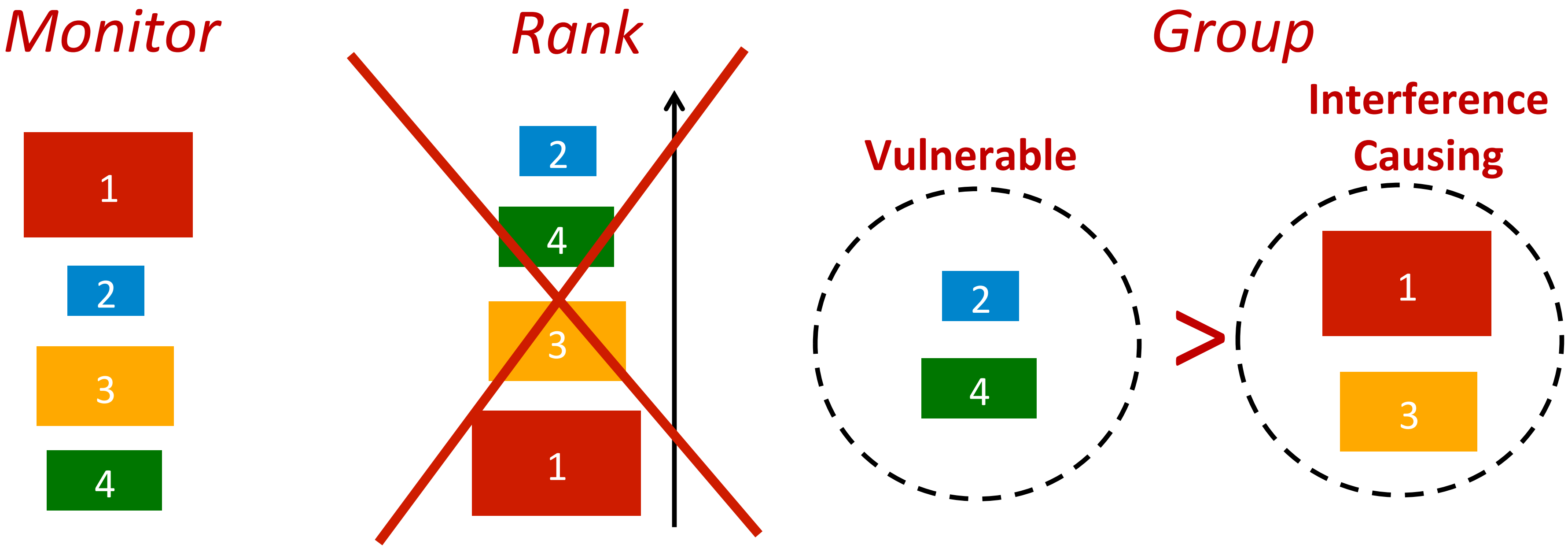
(low memory intensity)



Benefit 2: Lower slowdowns than ranking

Key Observation 1: Group Rather Than Rank

Observation 1: Sufficient to separate applications into two groups, rather than do full ranking



How to classify applications into groups?

Key Observation 2

Observation 2: Serving a large number of consecutive requests from an application causes interference

Basic Idea:

- *Group* applications with a large number of consecutive requests as *interference-causing* → *Blacklisting*
- *Deprioritize* blacklisted applications
- *Clear* blacklist periodically (1000s of cycles)

Benefits:

- *Lower complexity*
- *Finer grained grouping decisions* → *Lower unfairness*

Outline

- Introduction
- Problems with application-aware schedulers
- Key Observations
- **The Blacklisting Memory Scheduler Design**
- Evaluation
- Conclusion

The Blacklisting Memory Scheduler (BLISS)

1. Monitor

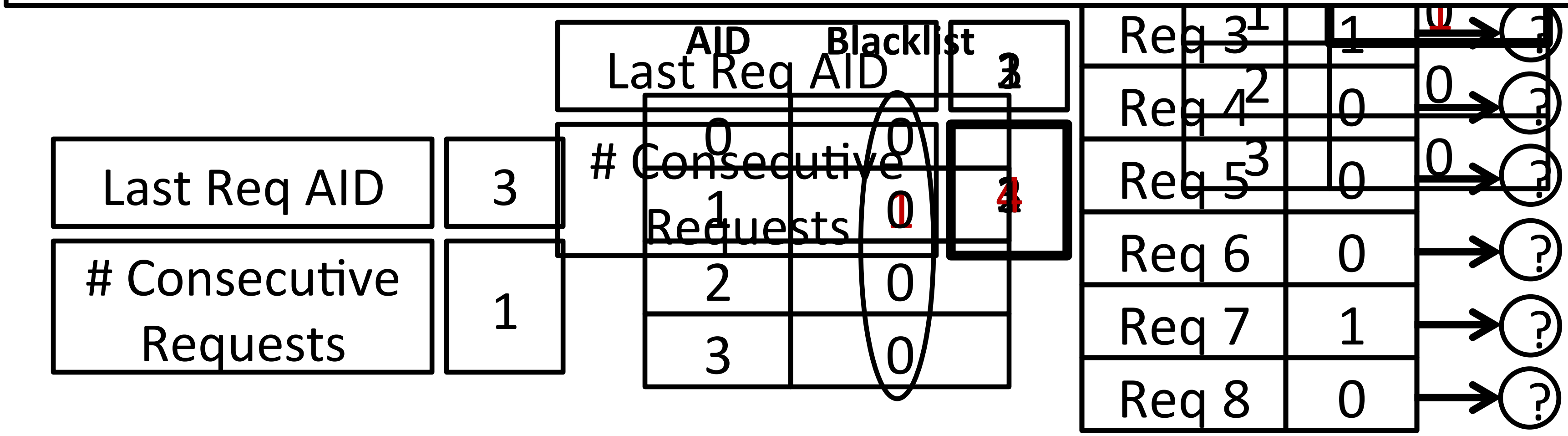
2. Blacklist

3. Prioritize

4. Clear Periodically

Request Buffer

Simple and scalable design



Outline

- Introduction
- Problems with application-aware schedulers
- Key Observations
- The Blacklisting Memory Scheduler Design
- **Evaluation**
- Conclusion

Methodology

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

- Workloads
  - SPEC CPU2006, TPC-C, Matlab , NAS
  - 80 multiprogrammed workloads

Metrics

- System Performance:

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

- Fairness:

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

- Complexity:

Critical path latency and area from synthesis with 32 nm library

Previous Memory Schedulers

- FRFCFS [Zuravleff and Robinson, US Patent 1997, Rixner et al., ISCA 2000]
  - Prioritizes row-buffer hits and older requests
- FRFCFS-Cap [Mutlu and Moscibroda, MICRO 2007]
  - Caps number of consecutive row-buffer hits

Application-unaware

+ Low complexity

- Low performance and fairness

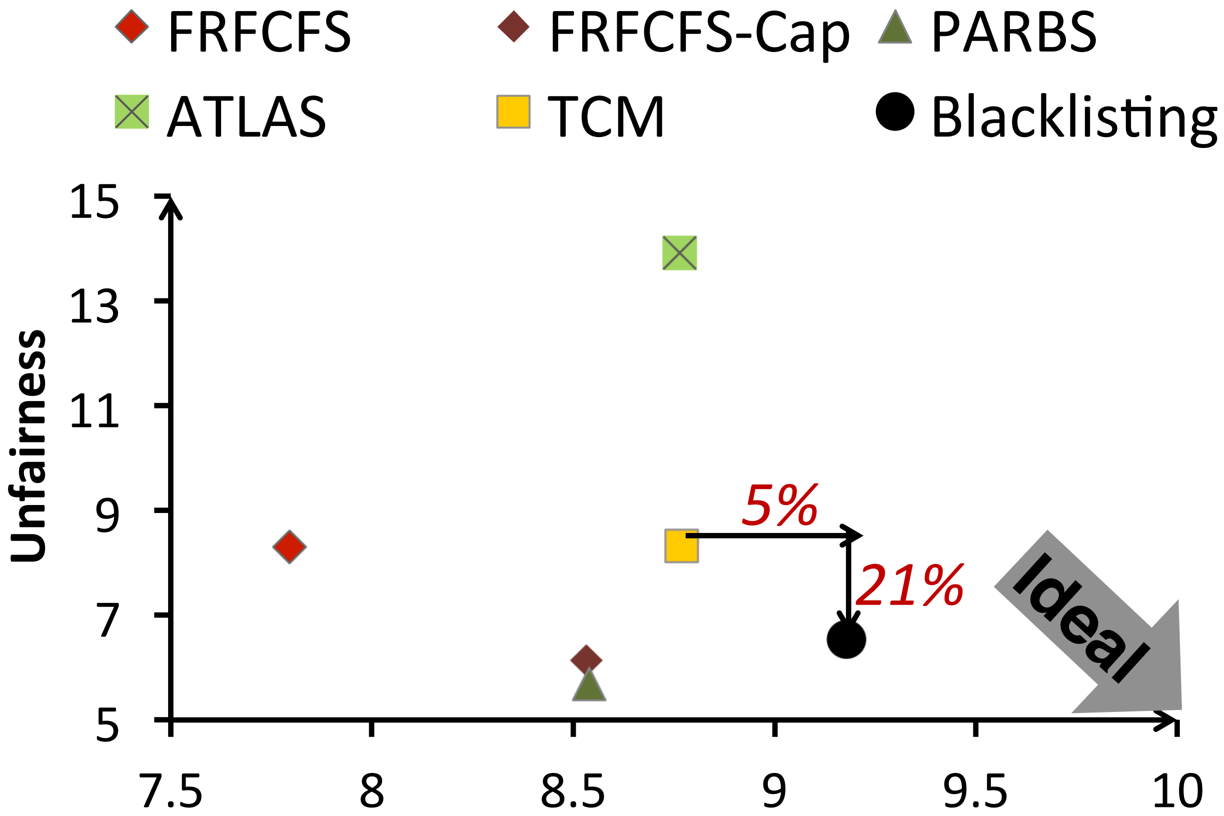
- PARBS [Mutlu and Moscibroda, ISCA 2008]
  - Batches oldest requests from each application; prioritizes batch
  - Employs ranking within a batch
- ATLAS [Kim et al., HPCA 2010]
  - Prioritizes applications with low memory-intensity
- TCM [Kim et al., MICRO 2010]
  - Always prioritizes low memory-intensity applications
  - Shuffles thread ranks of high memory-intensity applications

Application-aware

+ High performance and fairness

- High complexity

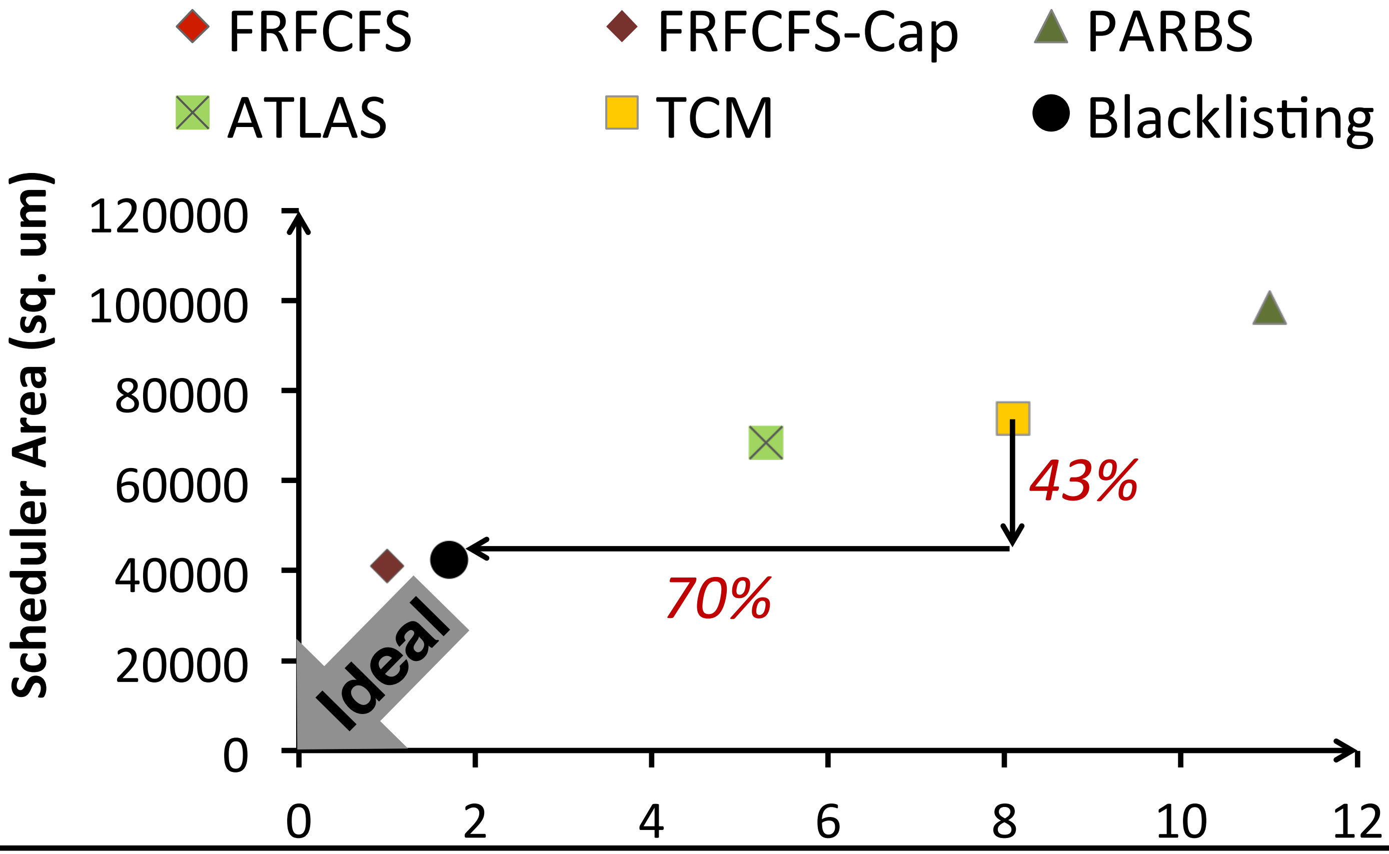
Performance and Fairness



1. Blacklisting achieves the highest performance

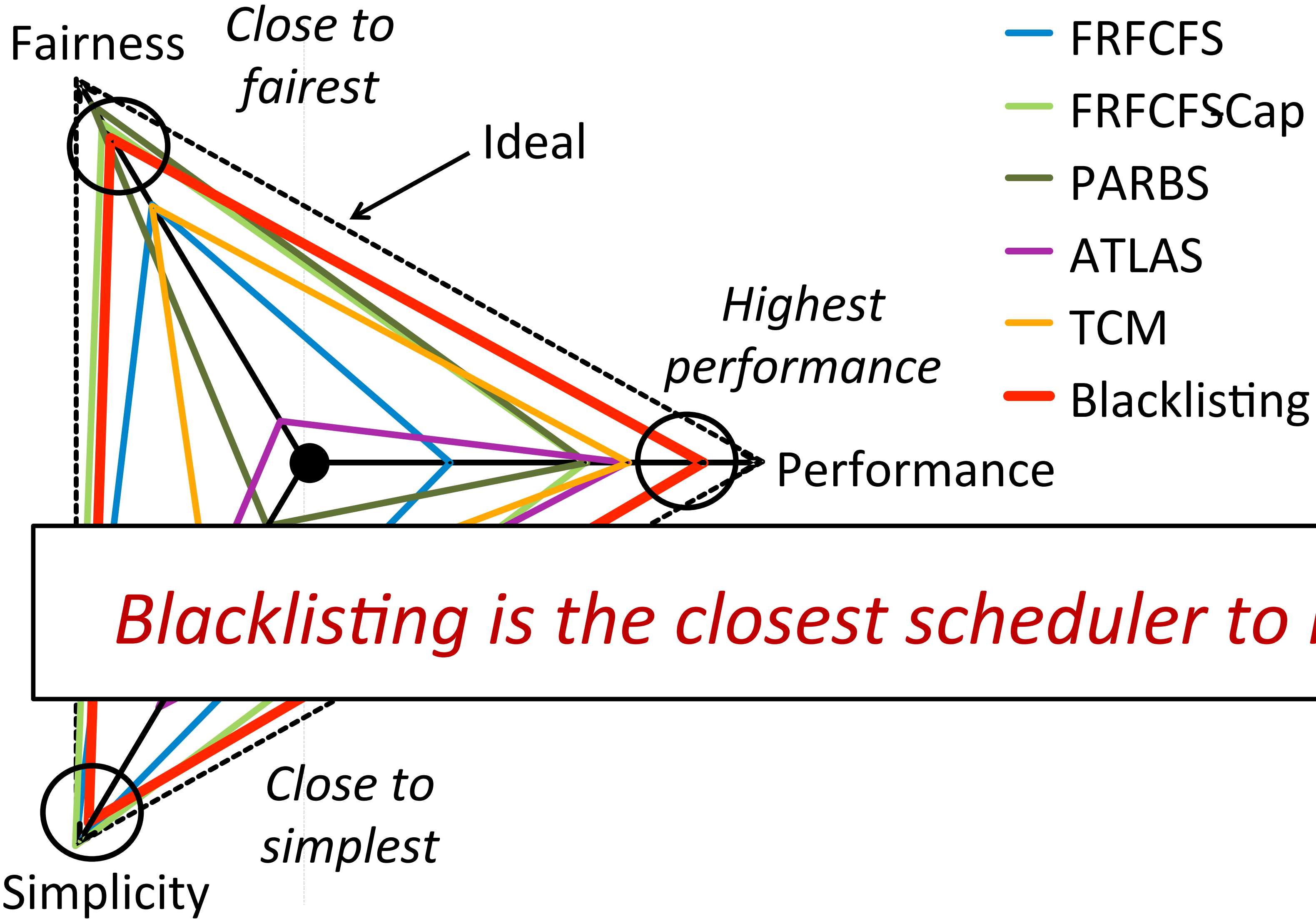
2. Blacklisting balances performance and fairness

Complexity



Blacklisting reduces complexity significantly

Performance vs. Fairness vs. Simplicity



Blacklisting is the closest scheduler to ideal

Summary

- Applications' requests interfere at main memory
- Prevalent solution approach
  - *Application-aware memory request scheduling*
- Key shortcoming of previous schedulers: *Full ranking*
  - *High hardware complexity*
  - *Unfair application slowdowns*

- Our Solution: Blacklisting memory scheduler
  - *Sufficient to group applications rather than rank*
  - *Group by tracking number of consecutive requests*

- *Much simpler than application-aware schedulers at higher performance and fairness*

The Blacklisting Memory Scheduler

Achieving High Performance and Fairness at Low Cost

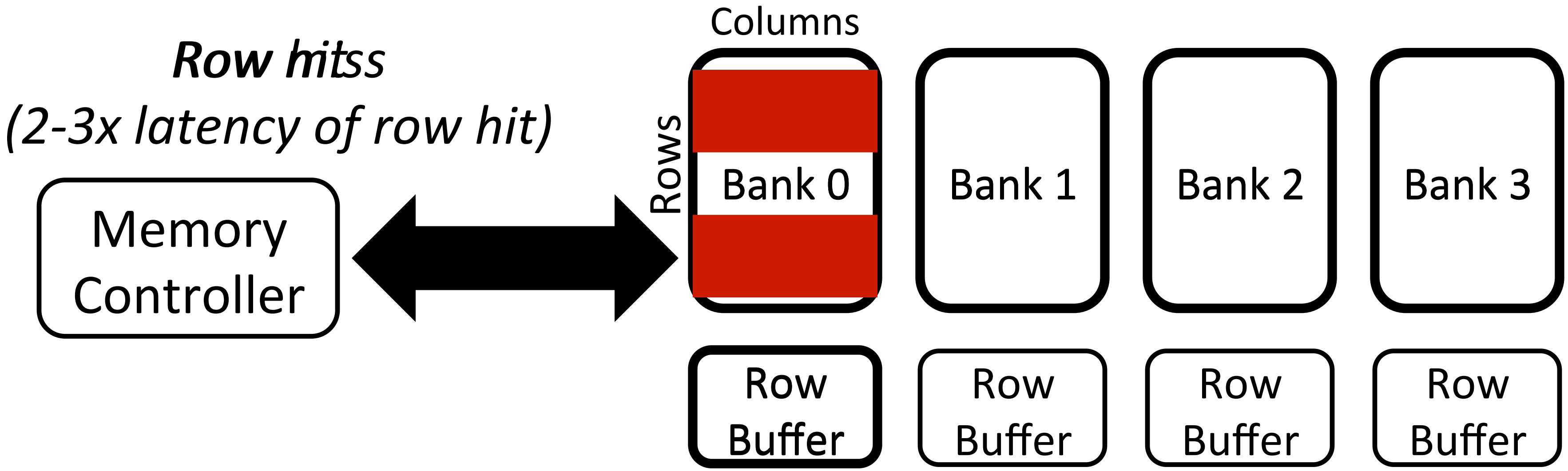
Lavanya Subramanian, Donghyuk Lee,
Vivek Seshadri, Harsha Rastogi, Onur Mutlu

SAFARI

Carnegie Mellon

Backup Slides

DRAM Memory Organization

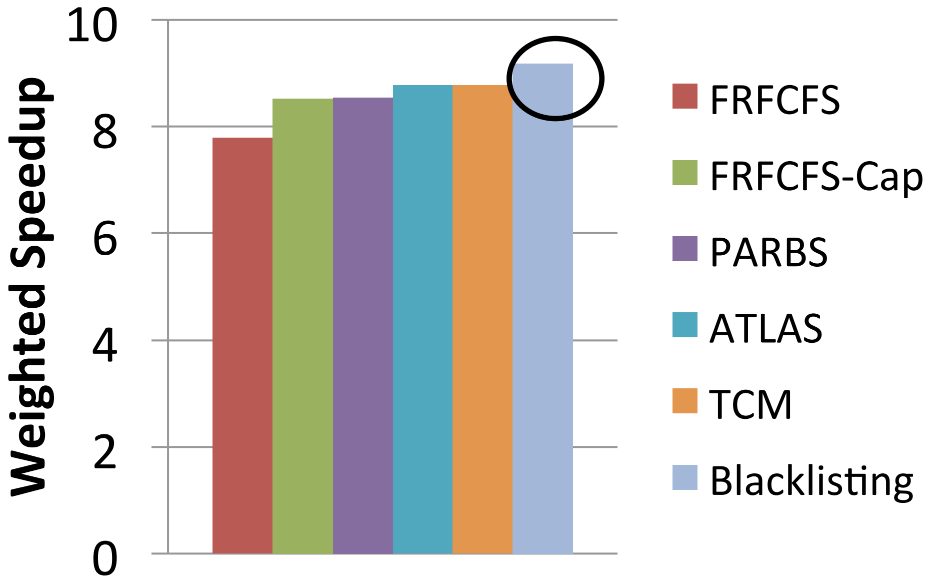


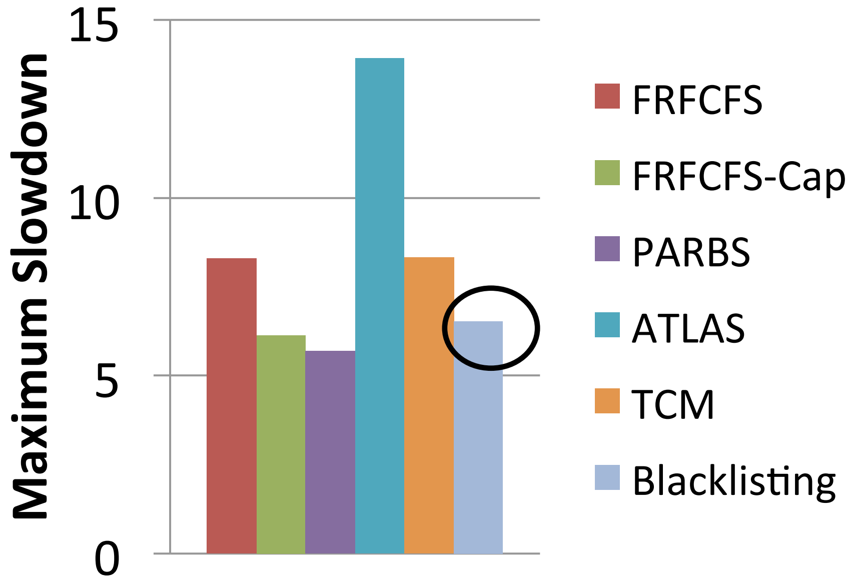
- FR-FCFS Memory Scheduler [Zuravleff and Robinson, US Patent '97; Rixner et al., ISCA '00]
  - Row-buffer hit first
  - Older request first
- *Unaware of inter-application interference*

Tackling Inter-Application Interference: Application-aware Memory Scheduling

- Monitor application memory access characteristics (e.g., memory intensity)
- Rank applications based on memory access characteristics
- Prioritize requests at the memory controller, based on ranking

Performance and Fairness

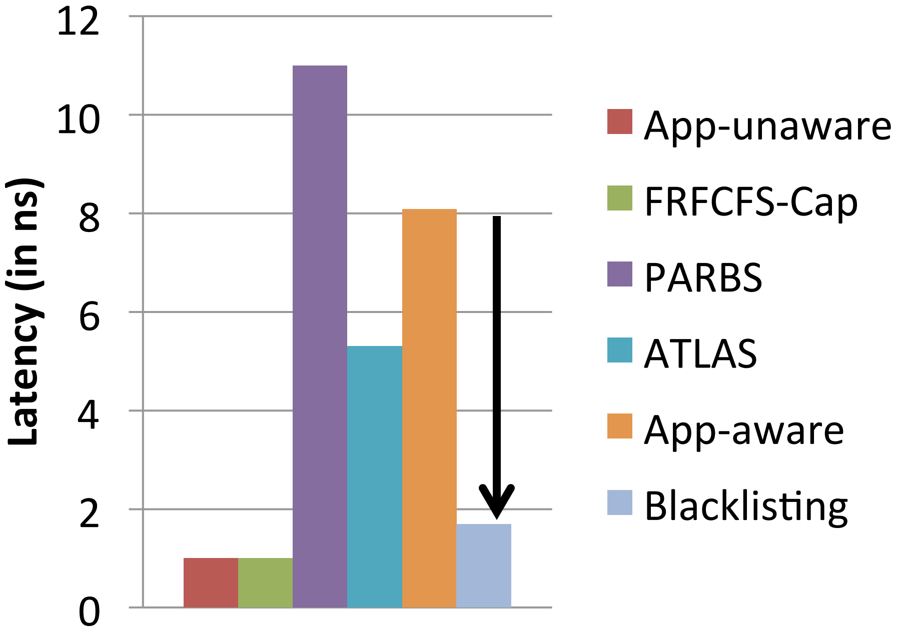


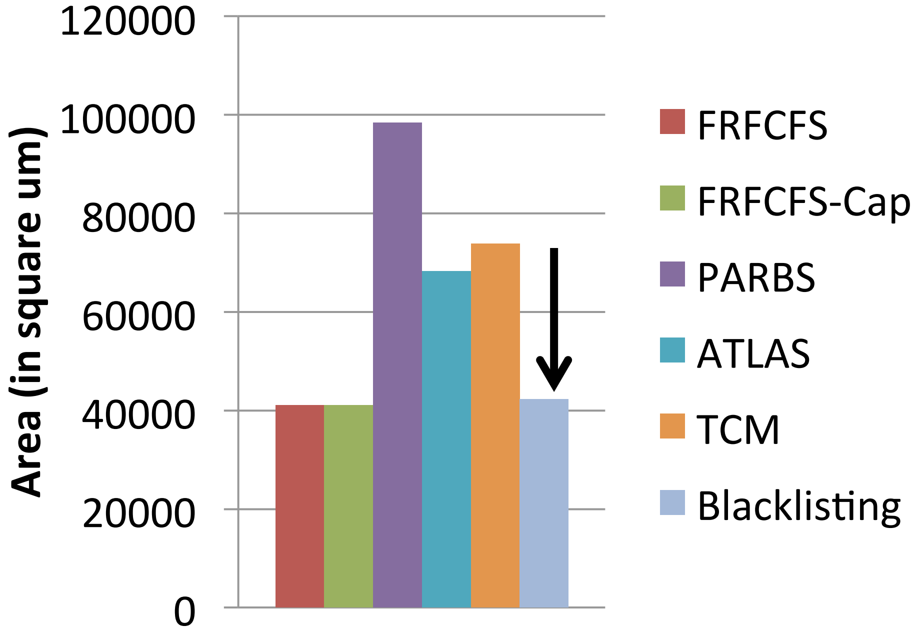


5% higher system performance and 21% lower maximum slowdown than TCM

SAFARI

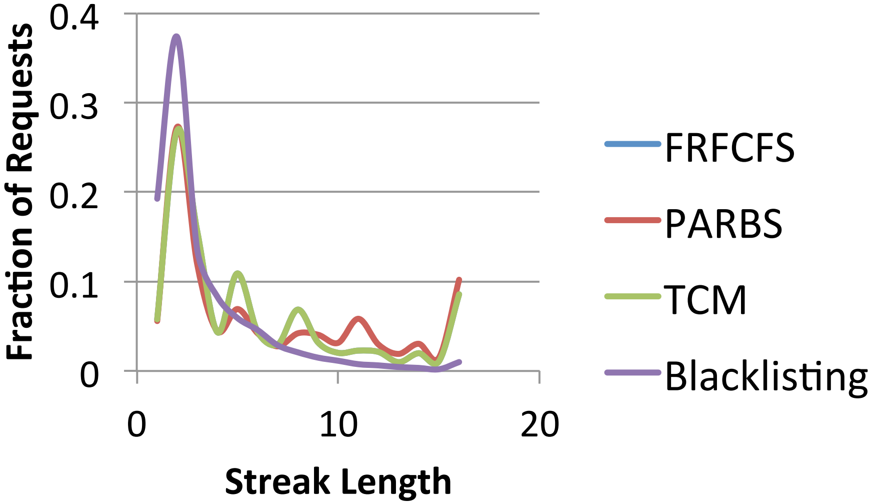
Complexity Results



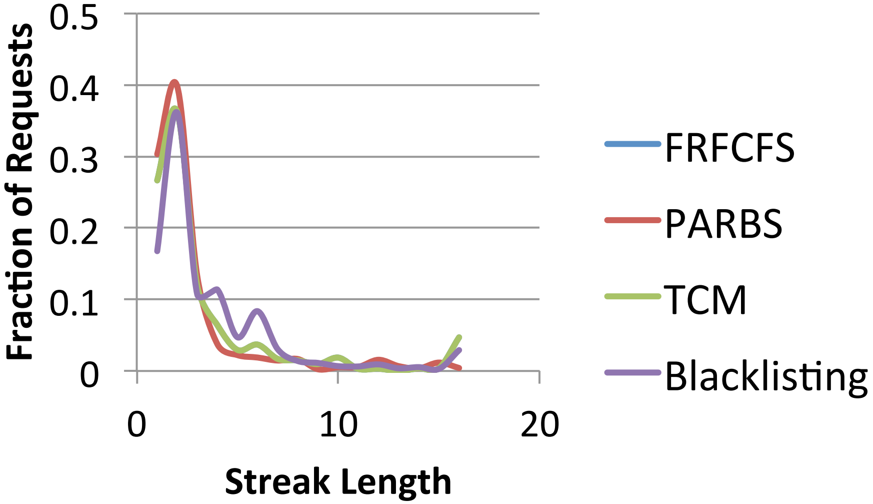


Blacklisting achieves
43% lower area than TCM

Understanding Why Blacklisting Works



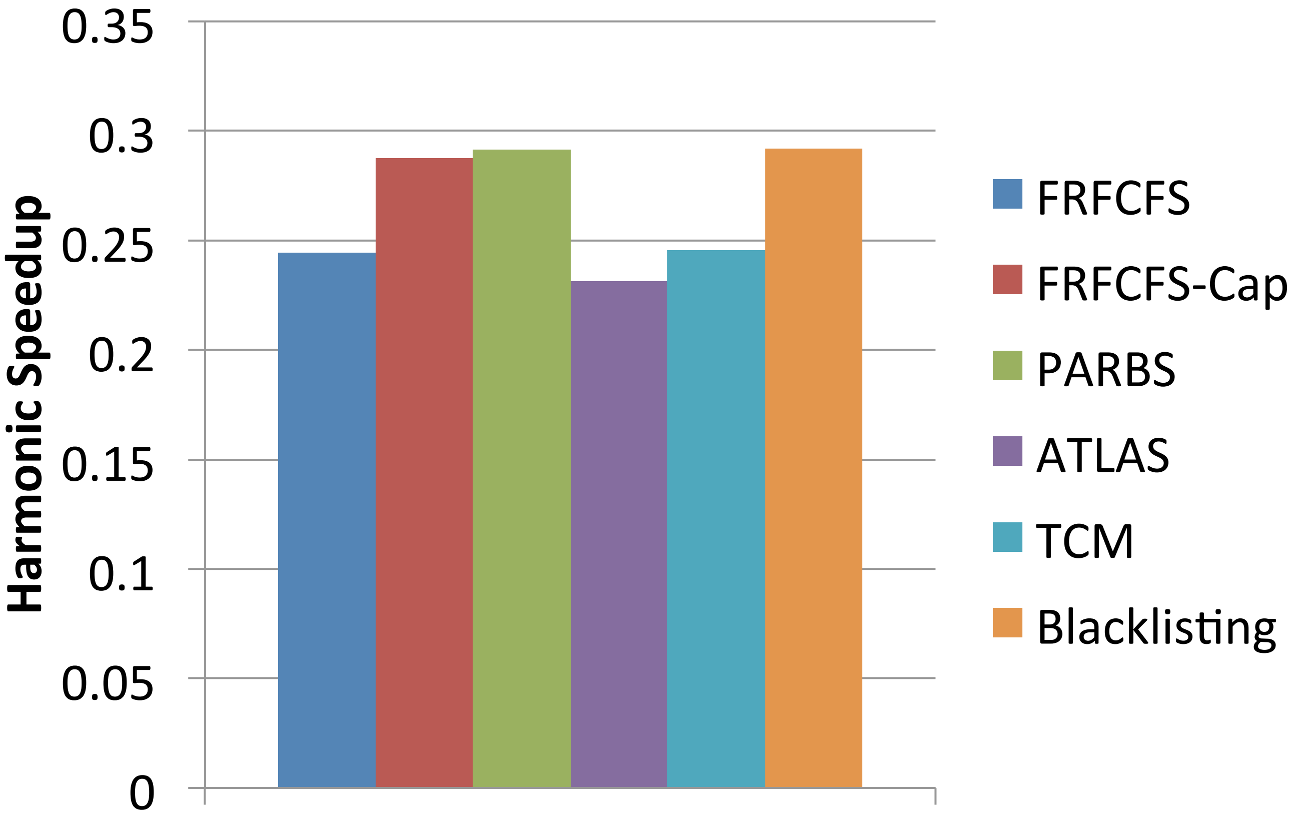
libquantum
(High memory-intensity application)



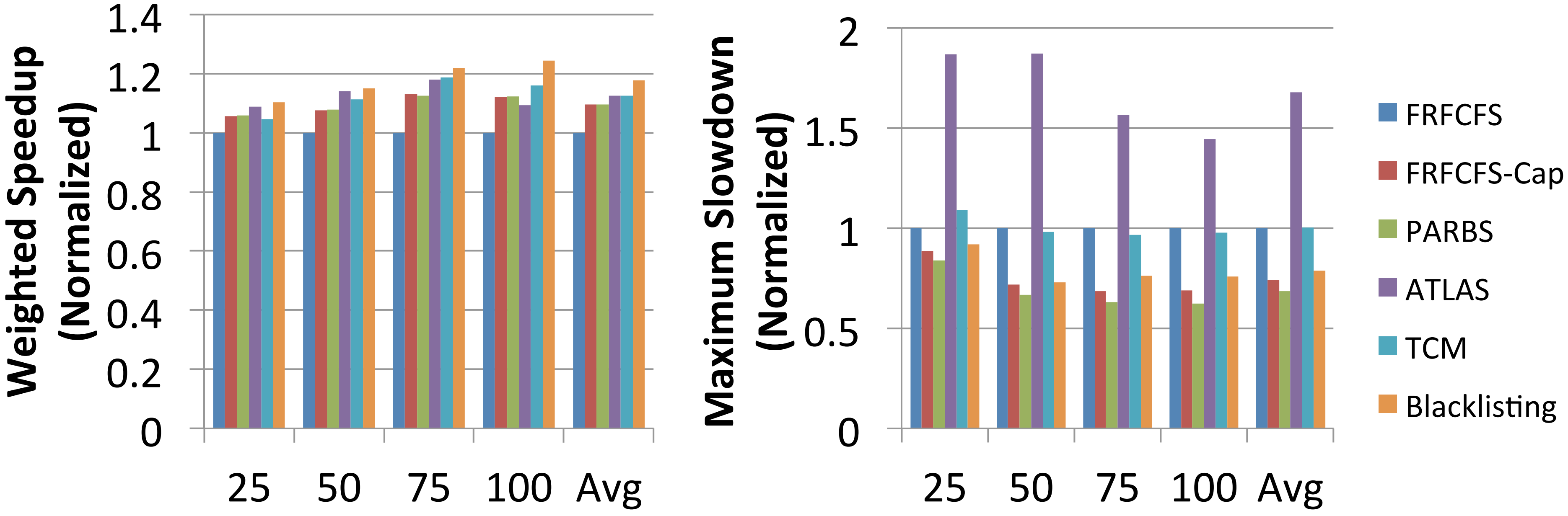
calculix
(Low memory-intensity application)

Blacklisting shifts the request distribution towards the right

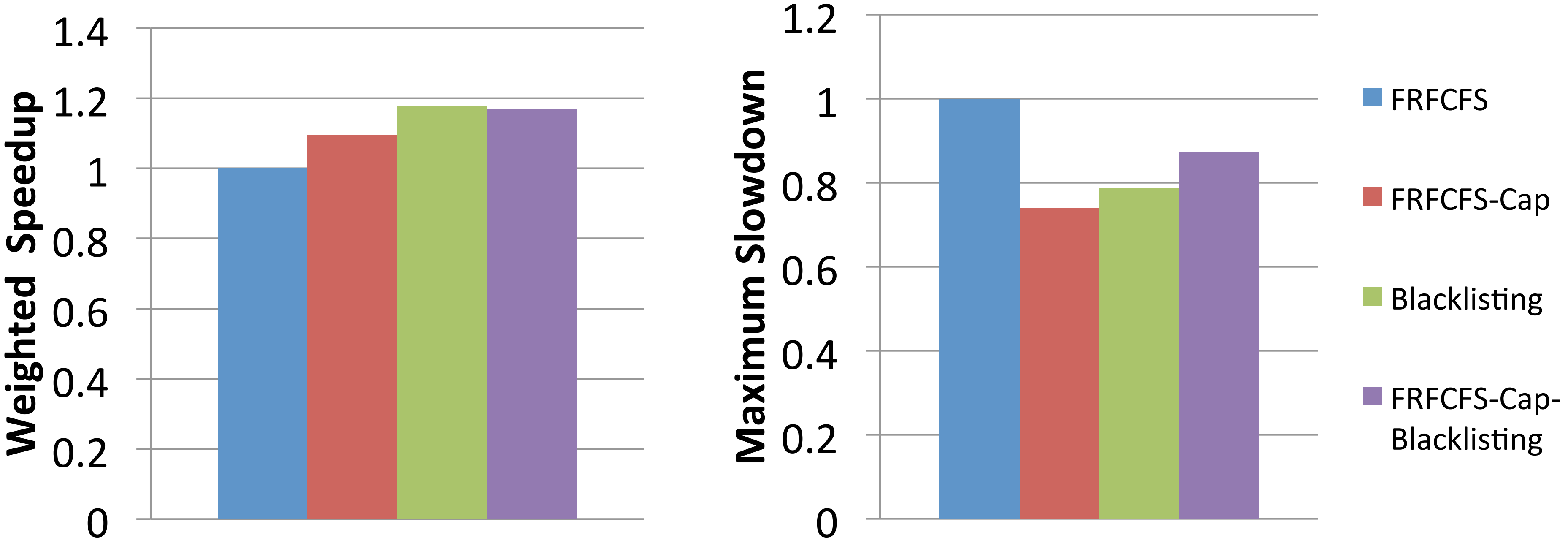
Harmonic Speedup



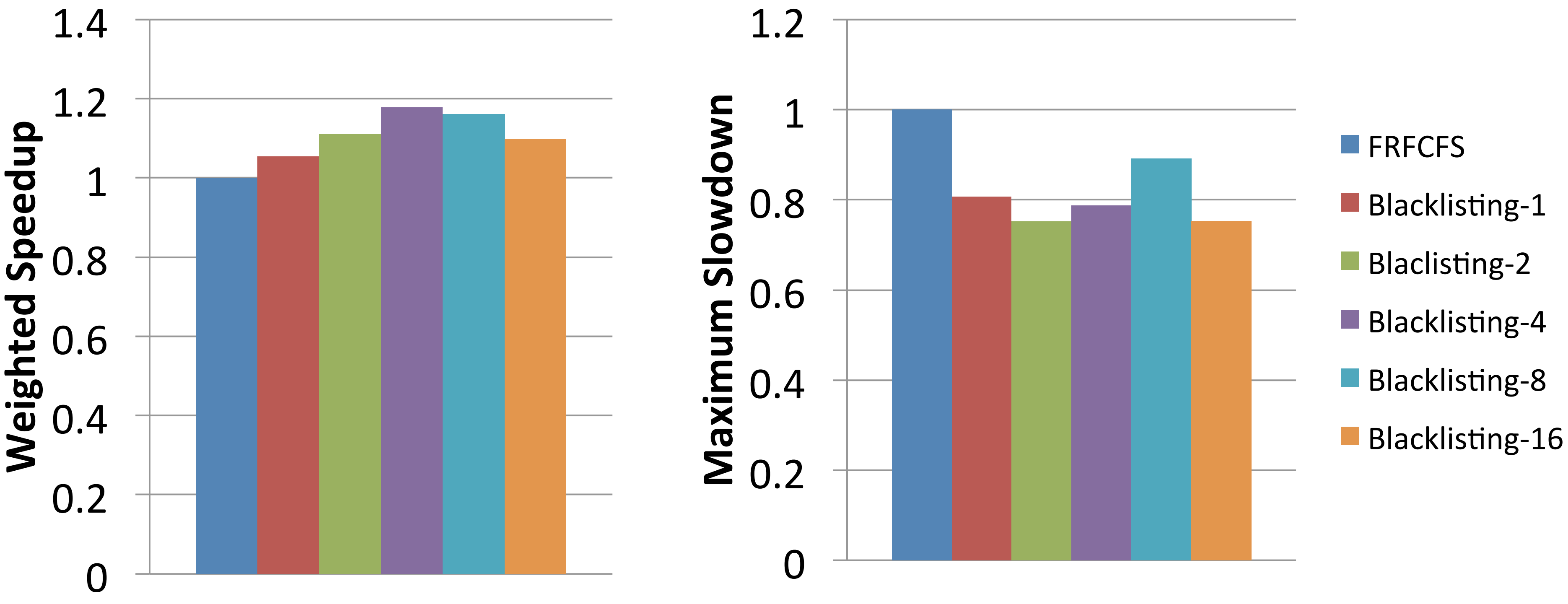
Effect of Workload Memory Intensity



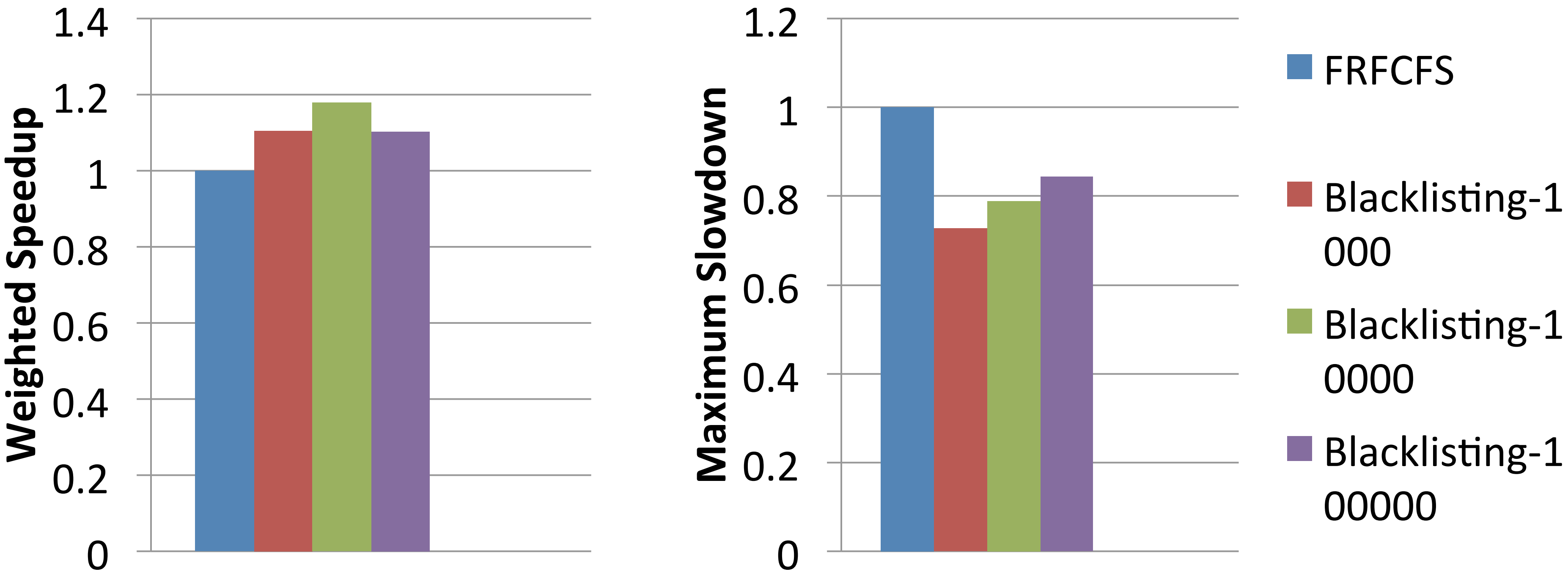
Combining FRFCFS-Cap and Blacklisting



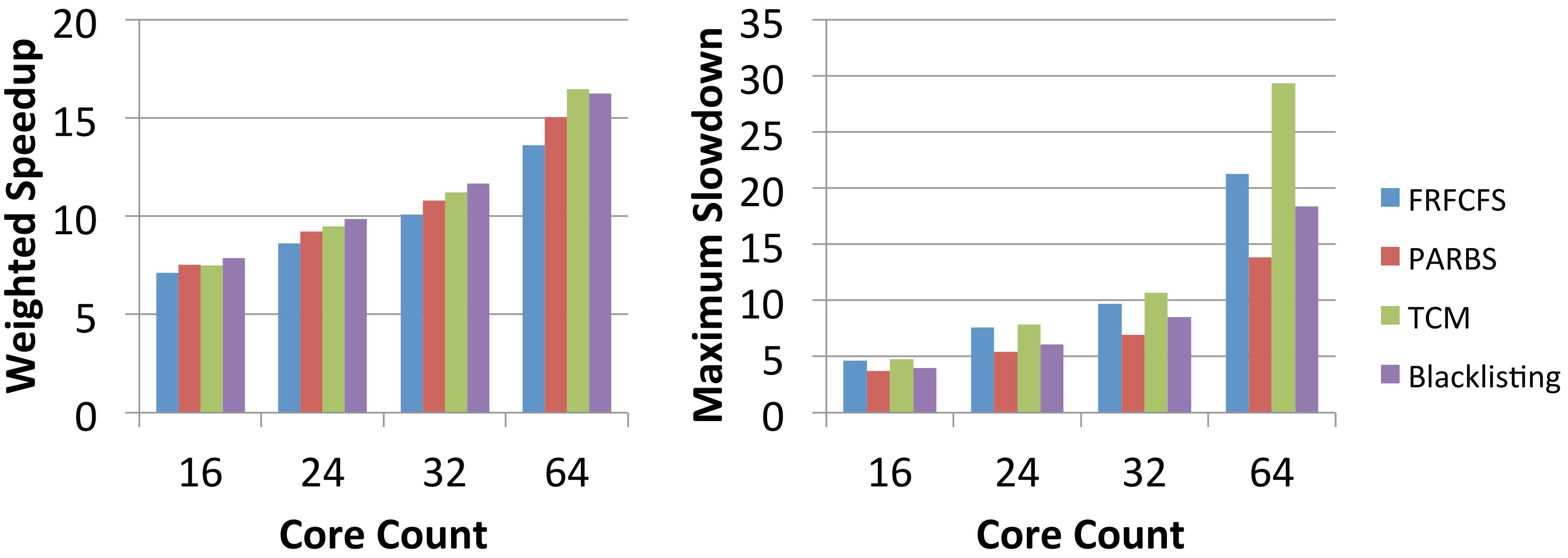
Sensitivity to Blacklisting Threshold



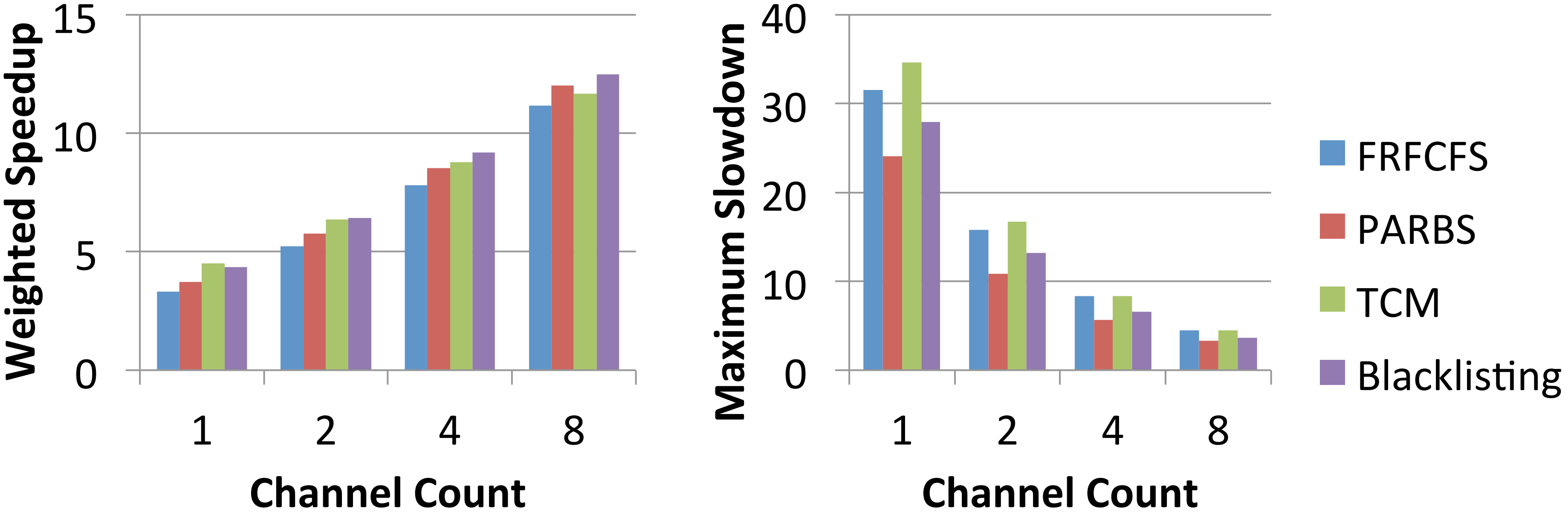
Sensitivity to Clearing Interval



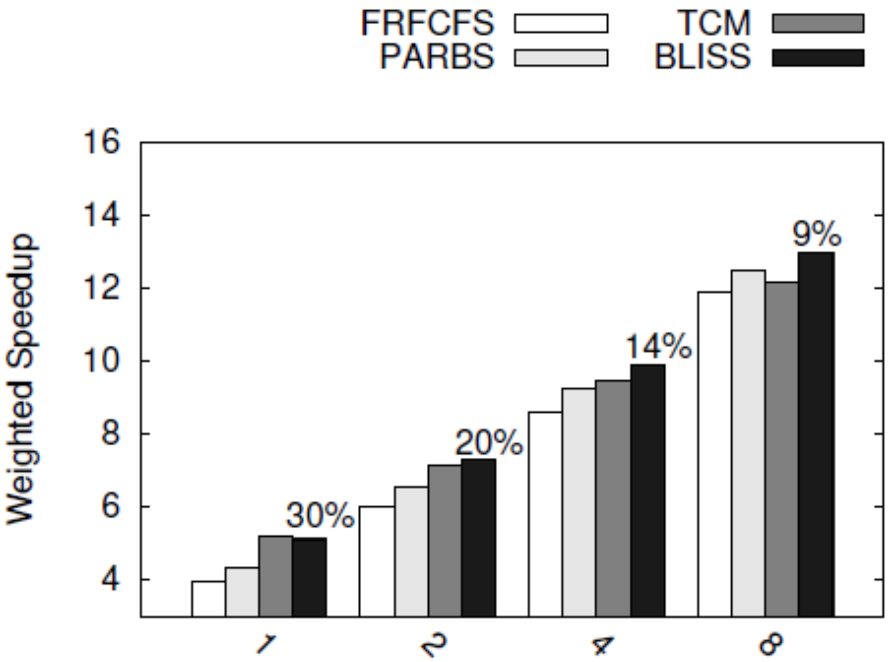
Sensitivity to Core Count

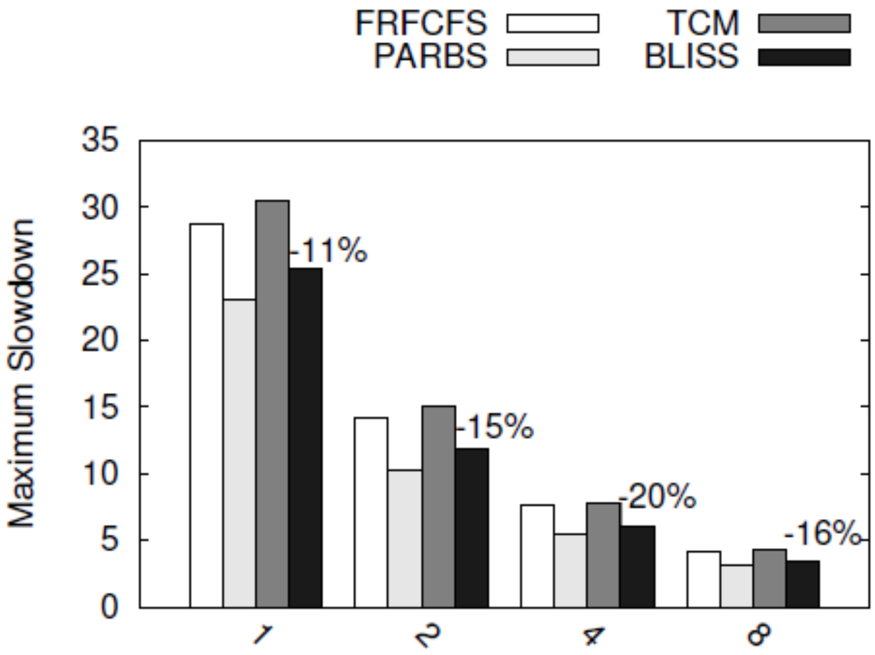


Sensitivity to Channel Count

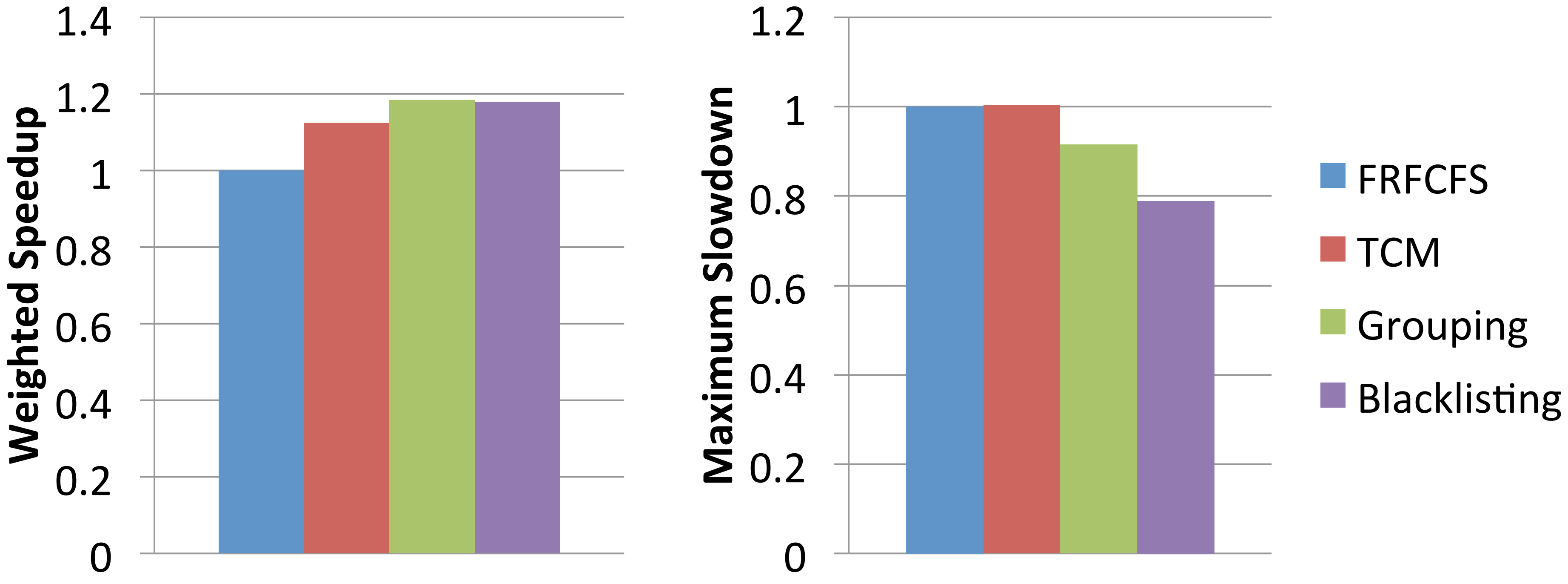


Sensitivity to Channel Count

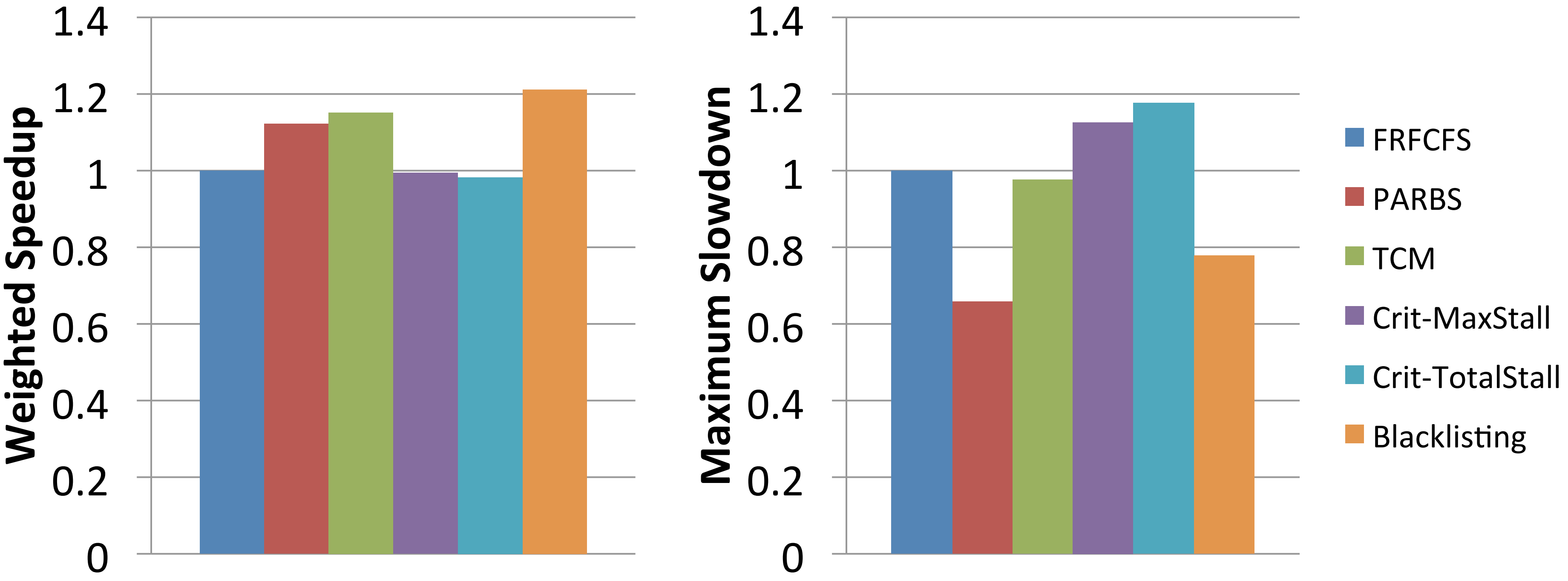




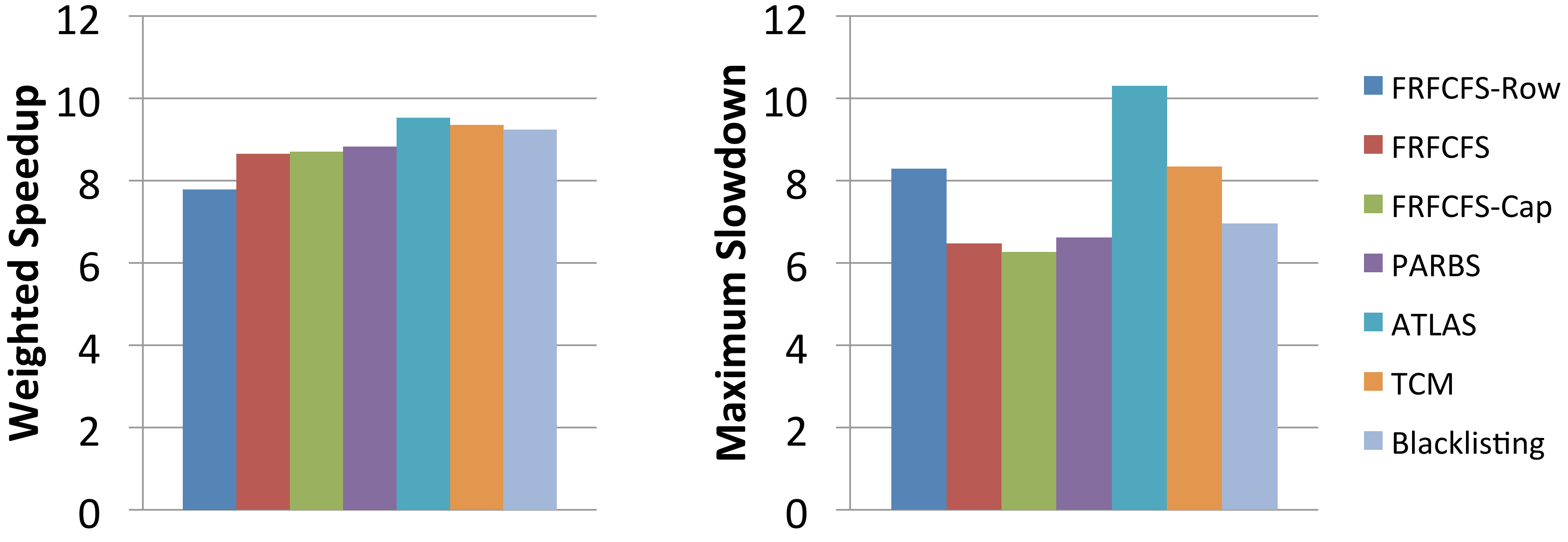
Breakdown of Benefits



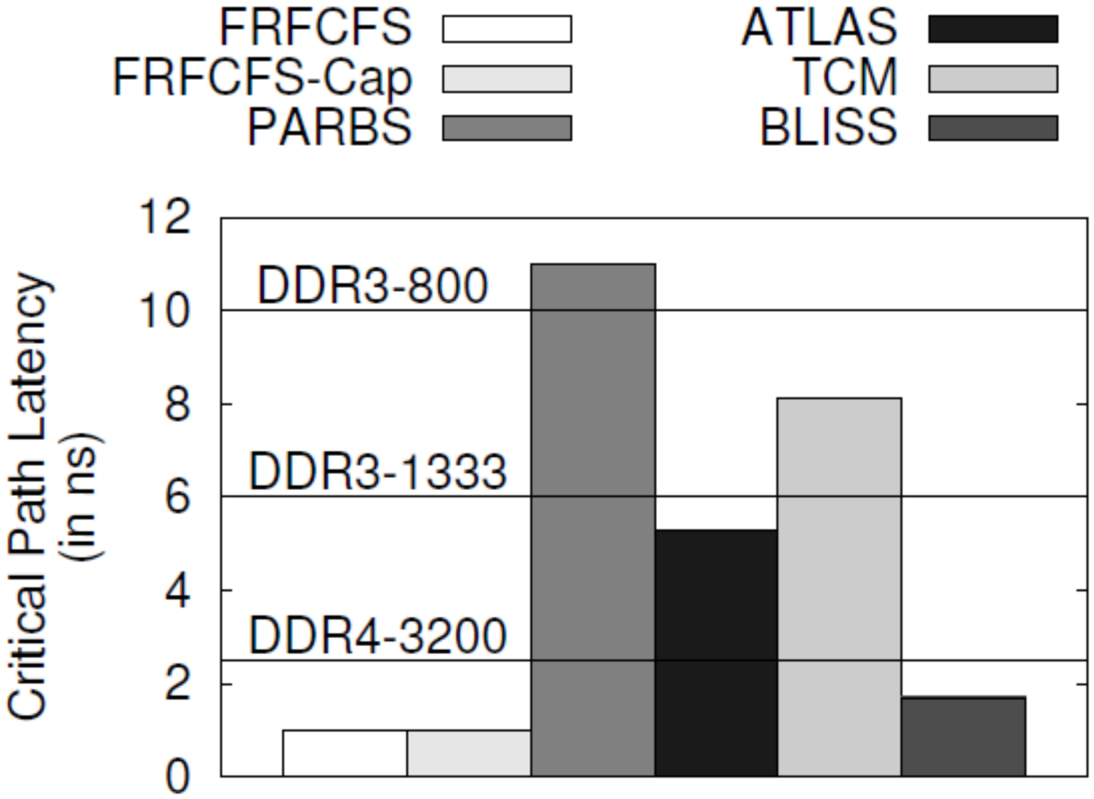
BLISS vs. Criticality-aware Scheduling



Sub-row Interleaving



Meeting DDR Timing Requirements


FRFCFS
FRFCFS-Cap
PARBS
ATLAS
TCM
BLISS
Critical Path Latency (in ns)
DDR3-800
DDR3-1333
DDR4-3200