Techniques for Shared Resource Management in Systems with GPUs

Thesis Oral

Rachata Ausavarungnirun

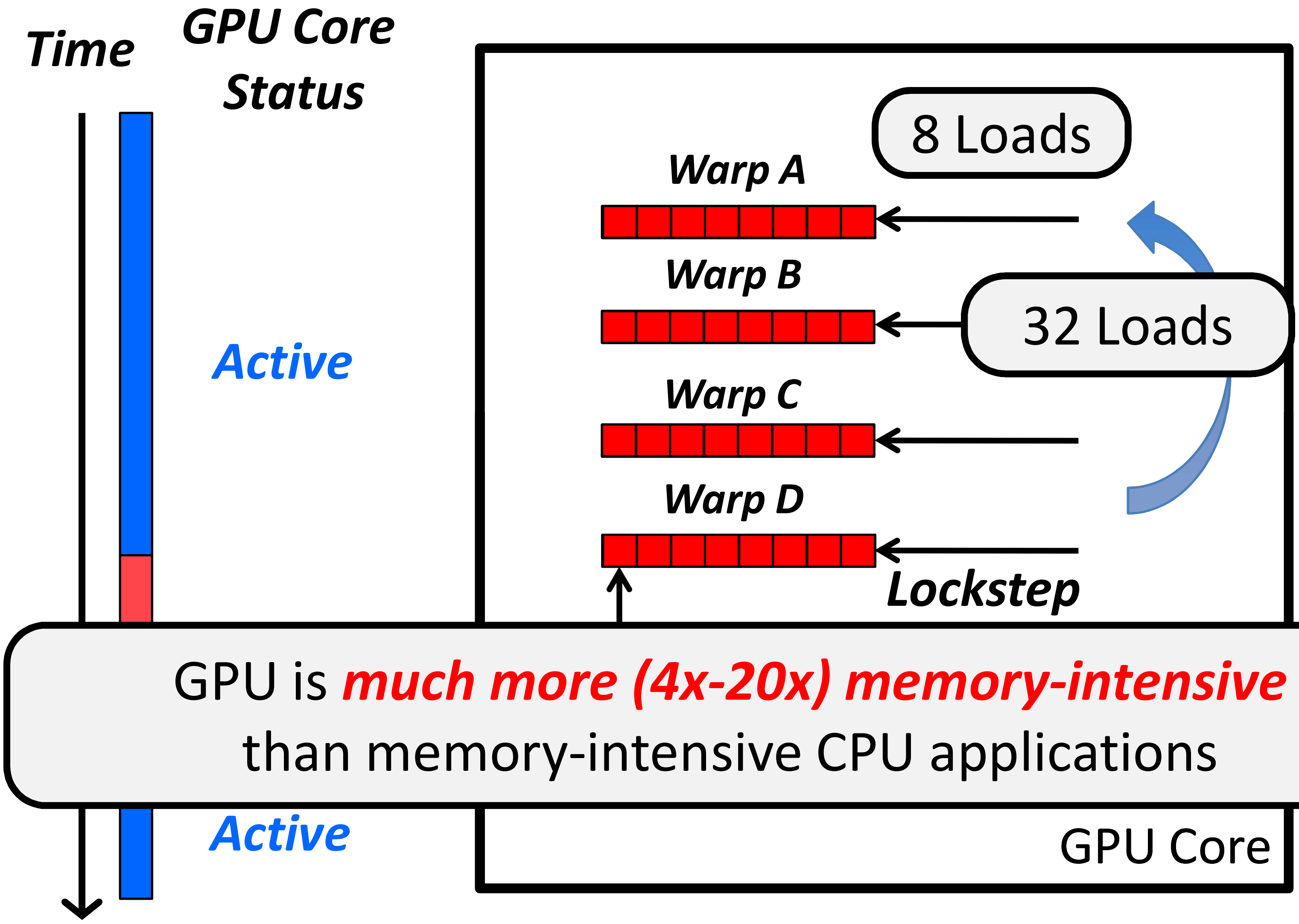
Committees:

Advisor: Onur Mutlu (CMU and ETH Zürich)
James C. Hoe (CMU)
Kayvon Fatahalian (CMU)
Gabriel H. Loh (AMD Research)
Christopher J. Rossbach (UT Austin and VMware Research)

SAFARI

Carnegie Mellon

Parallelism in GPU

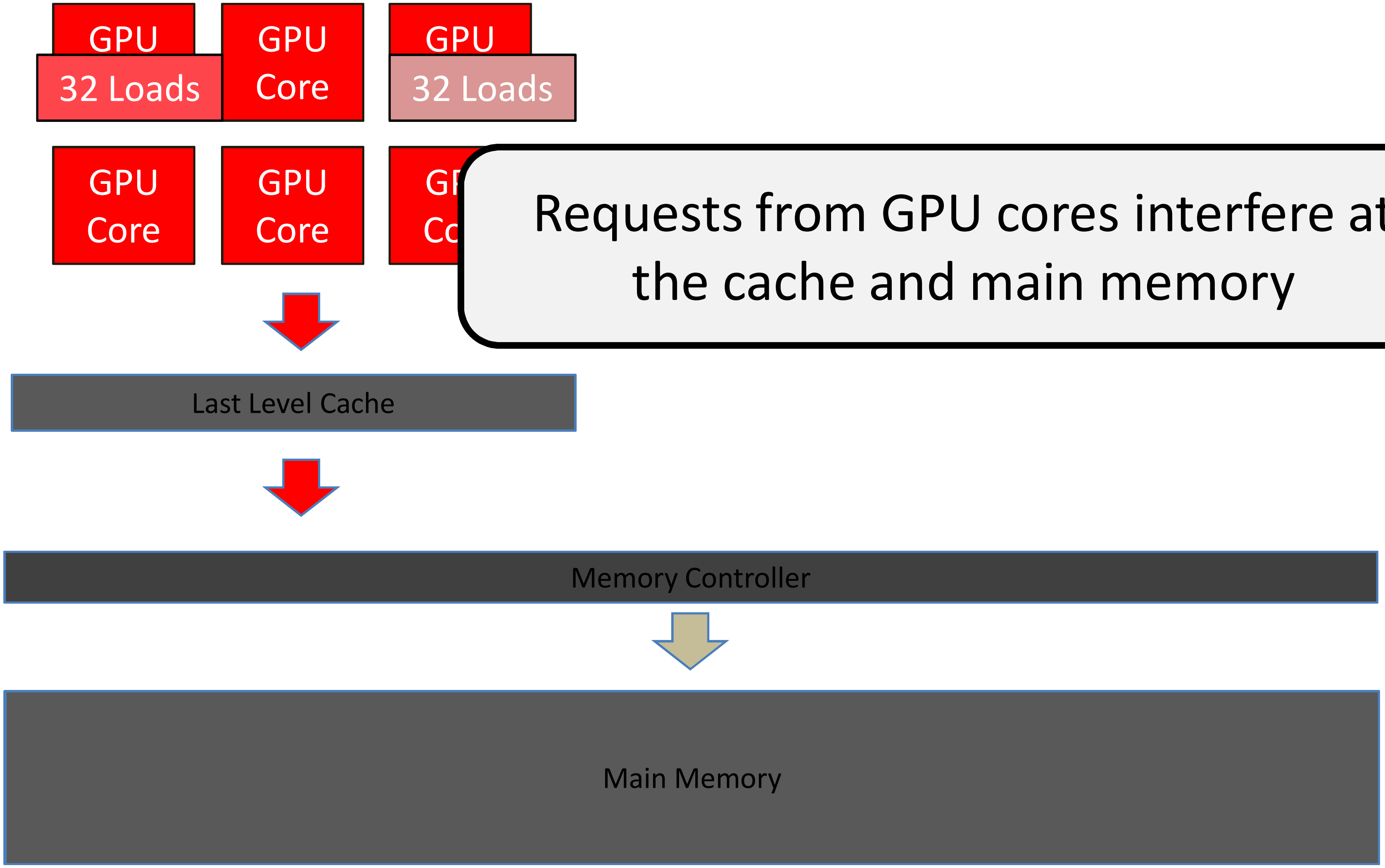


GPU is ***much more (4x-20x) memory-intensive***
than memory-intensive CPU applications

Three Types of Memory Interference

- Intra-application Interference

Intra-application Interference



Requests from GPU cores interfere at the cache and main memory

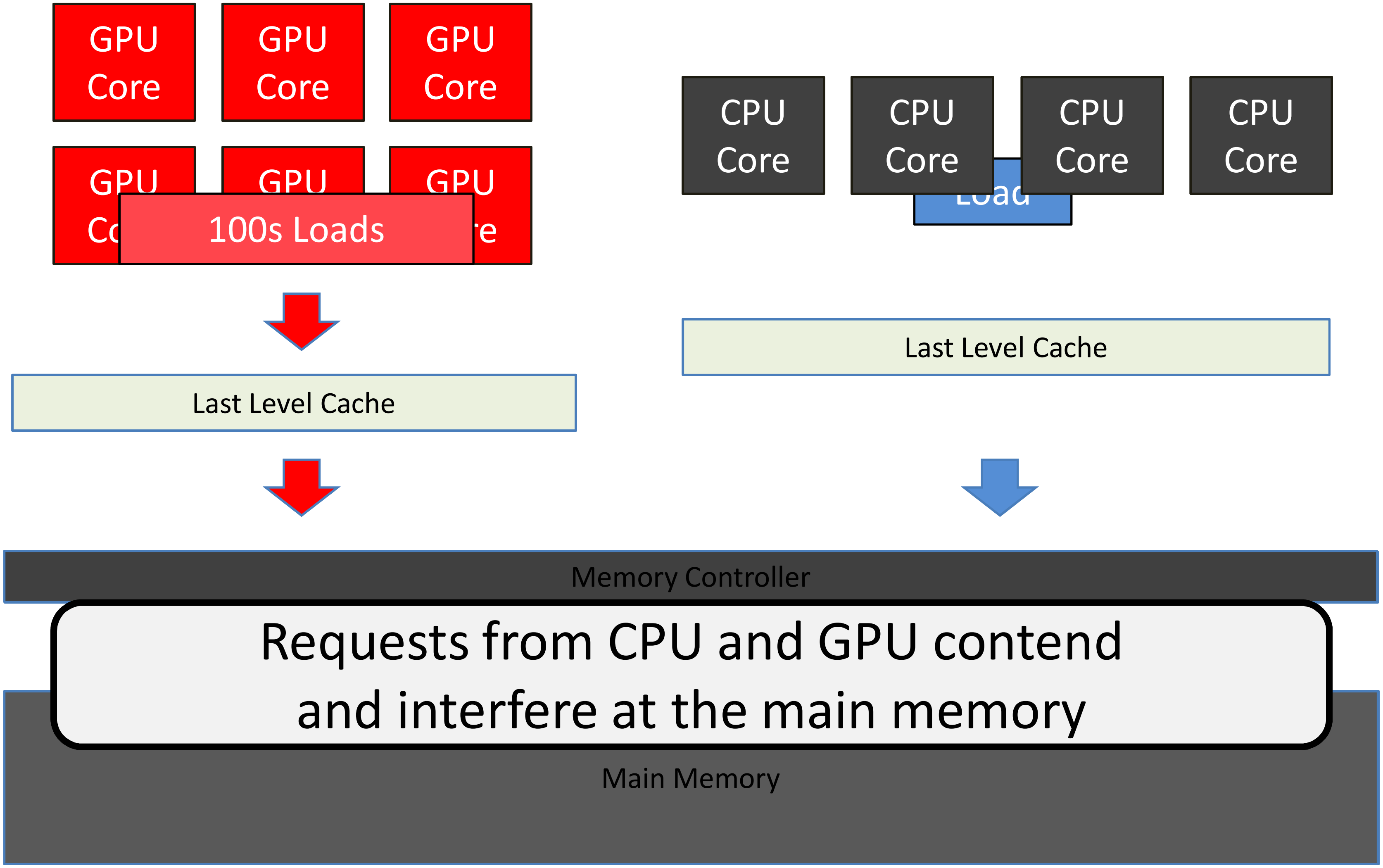
SAFARI

Three Types of Memory Interference

- Intra-application Interference

- Inter-application Interference

Inter-application Interference

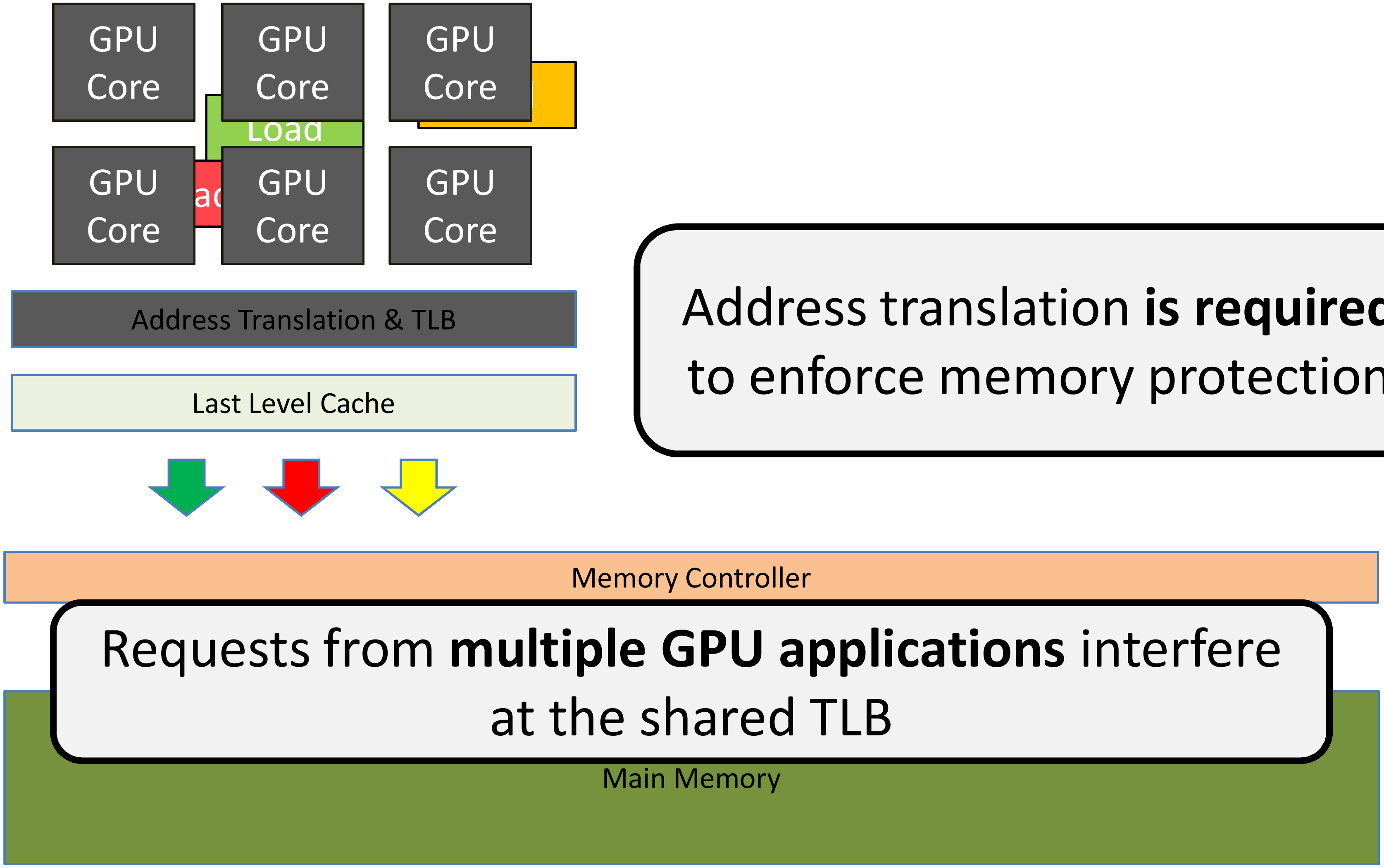


Requests from CPU and GPU contend and interfere at the main memory

Three Types of Memory Interference

- Intra-application Interference
- Inter-application Interference
- Inter-address-space Interference

Inter-address-space Interference



Address translation **is required** to enforce memory protection

Requests from **multiple GPU applications** interfere at the shared TLB

Previous Works

- Cache management schemes
  - Li et al. (HPCA’15), Li et al. (ICS’15), Jia et al. (HPCA’14), Chen et al. (MICRO’14, MES’14), Rogers et al. (MICRO’12), Seshadri et al. (PACT’12), Jaleel et al. (PACT’08), Jaleel et al. (ISCA’10)
  - **Does not take GPU’s memory divergence into account**
- Memory Scheduling
  - Rixner et al. (ISCA’00), Yuan et al. (MICRO’09), Kim et al. (HPCA’10), Kim et al. (MICRO’10), Mutlu et al. (MICRO’07), Kim et al. (MICRO’10)
  - **Does not take GPU’s traffic into account**
- TLB designs
  - Power et al. (HPCA’14), Cong et al. (HPCA’16)
  - **Only works for CPU-GPU heterogeneous systems**
- There is **no previous work** that holistically aims to solve all three types of interference **in GPU-based systems**

Thesis Statement

Approach

A combination of GPU-aware cache and memory management techniques

can mitigate interference caused by GPUs on current and future systems with GPUs.

Goals

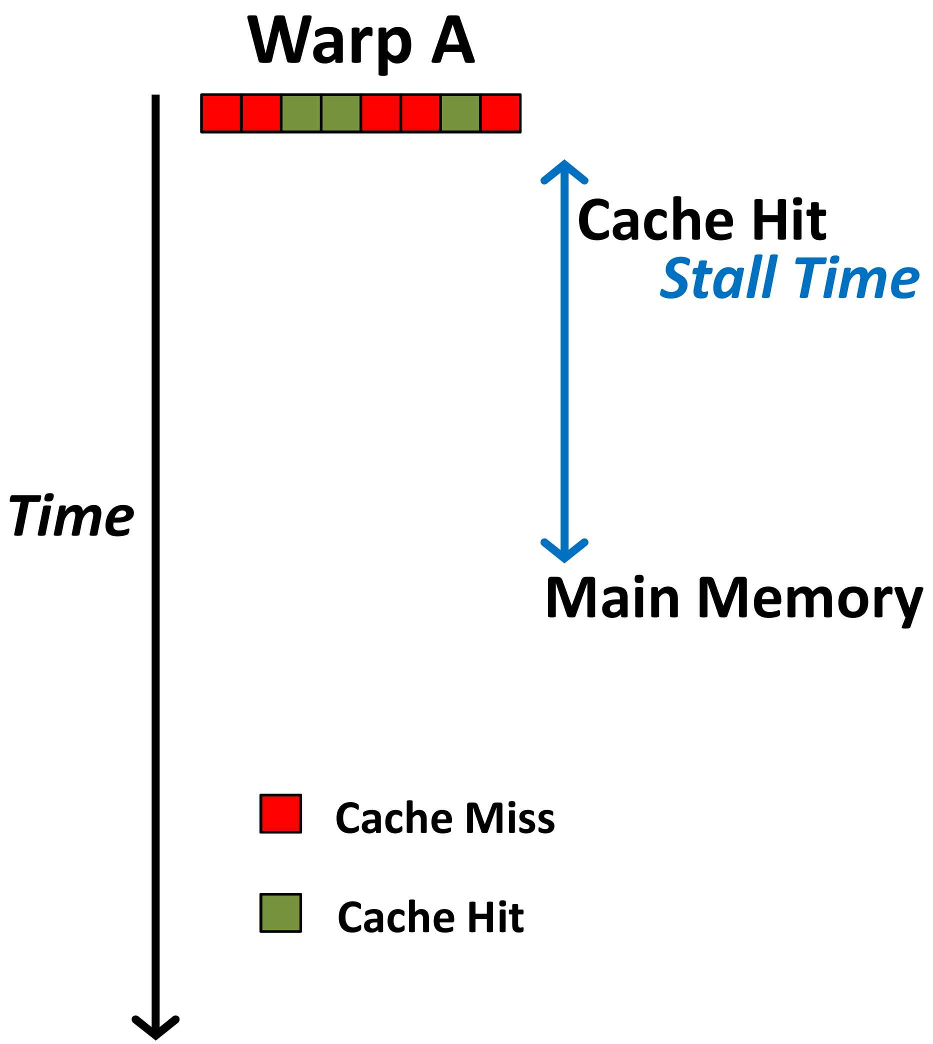
Our Approach

- Intra-application interference
  - Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance, PACT 2015
- Inter-application interference
  - Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems, ISCA 2012
- Inter-address-space interference
  - Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency, Submitted to MICRO 2017
  - Mosaic: A Transparent Hardware-Software Cooperative Memory Management in GPU, Submitted to MICRO 2017

Our Approach

- Intra-application interference
  - Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance, PACT 2015
- Inter-application interference
  - Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems, ISCA 2012
- Inter-address-space interference
  - Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency, Submitted to MICRO 2017
  - Mosaic: A Transparent Hardware-Software Cooperative Memory Management in GPU, Submitted to MICRO 2017

Inefficiency: Memory Divergence



Warp A

Cache Hit

Stall Time

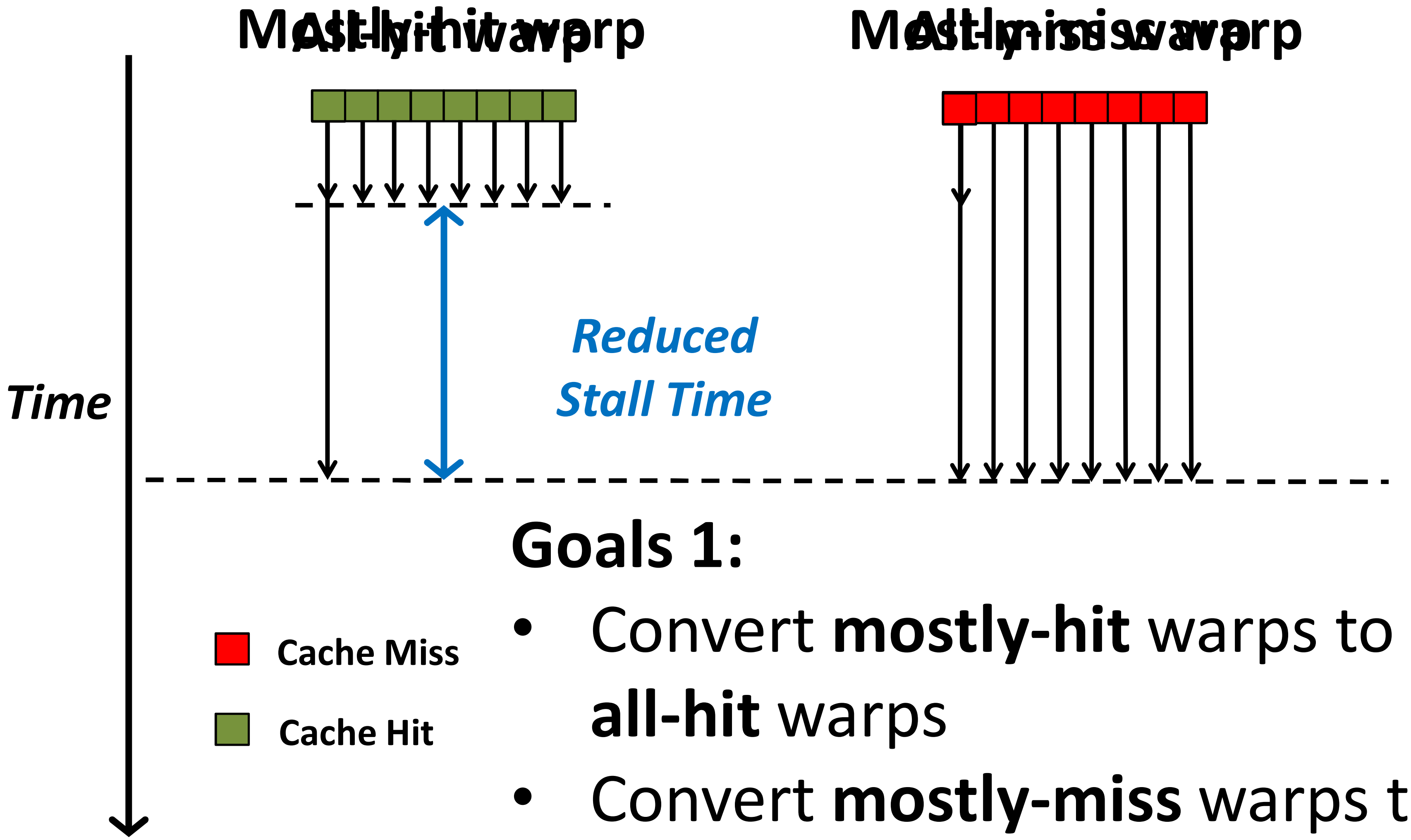
Time

Main Memory

🟥 Cache Miss

🟩 Cache Hit

Observation 1: Divergence Heterogeneity

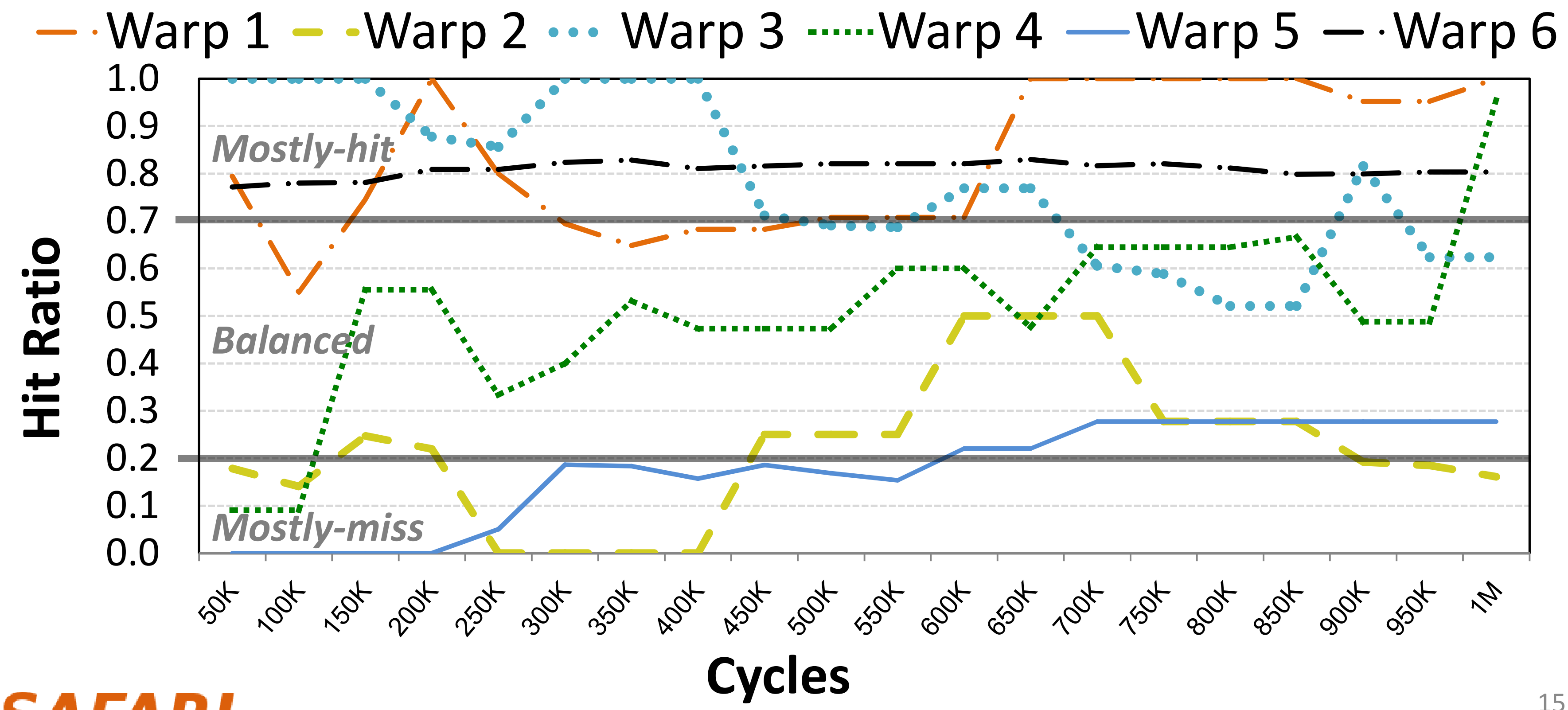


Goals 1:

- Convert **mostly-hit** warps to **all-hit** warps
- Convert **mostly-miss** warps to **all-miss** warps

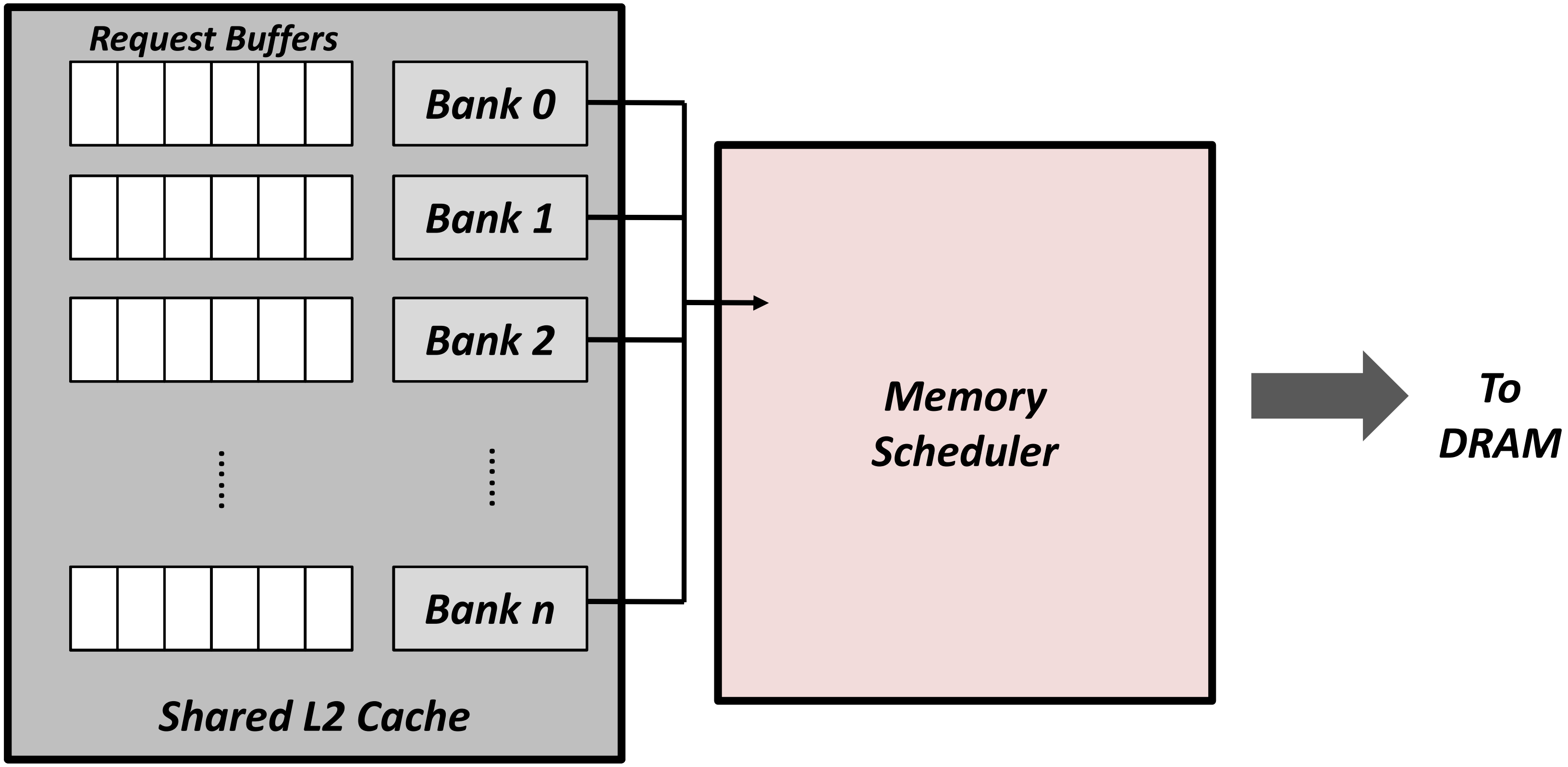
Observation 2: Stable Divergence Char.

- Warp retains its hit ratio during a program phase



SAFARI

Observation 3: Queuing at L2 Banks

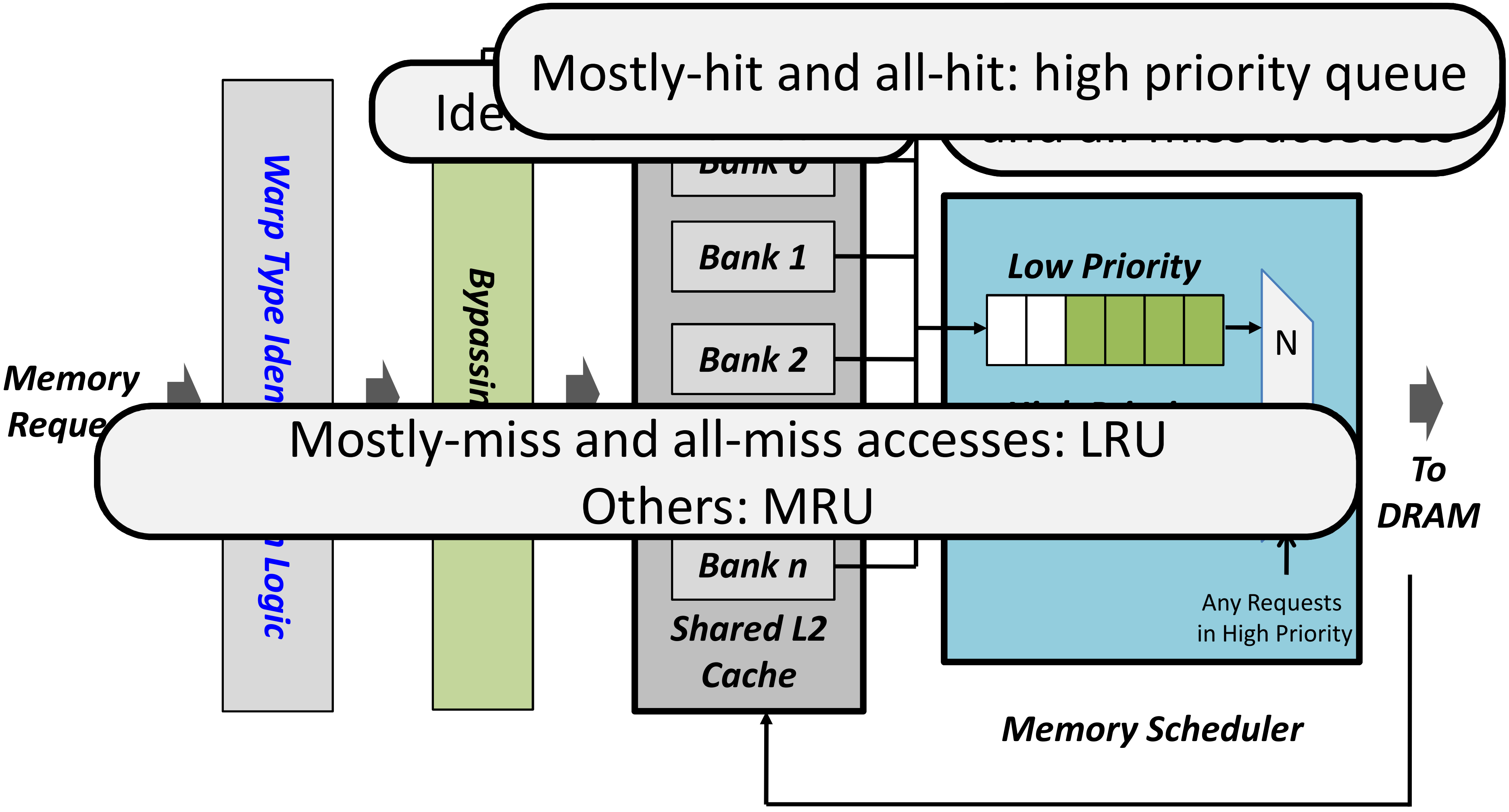


45% of requests stall 20+ cycles at the L2 queue

Goal 2: Reduce queuing latency

Memory Divergence Correction

Warp-type-aware Cache Bypassing



Mostly-hit and all-hit: high priority queue

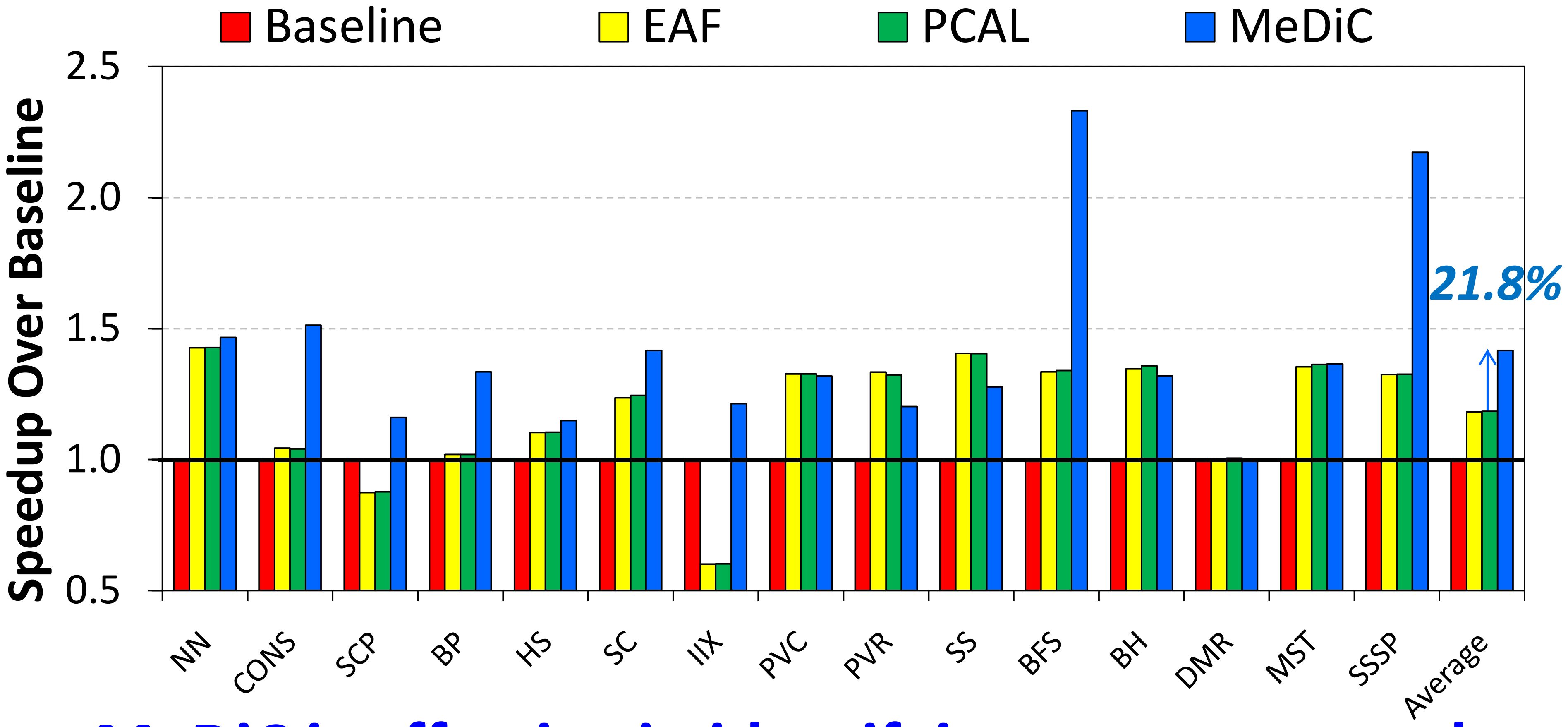
Mostly-miss and all-miss accesses: LRU

Others: MRU

Warp-type-aware Cache Insertion Policy

SAFARI

Results: Performance of MeDiC



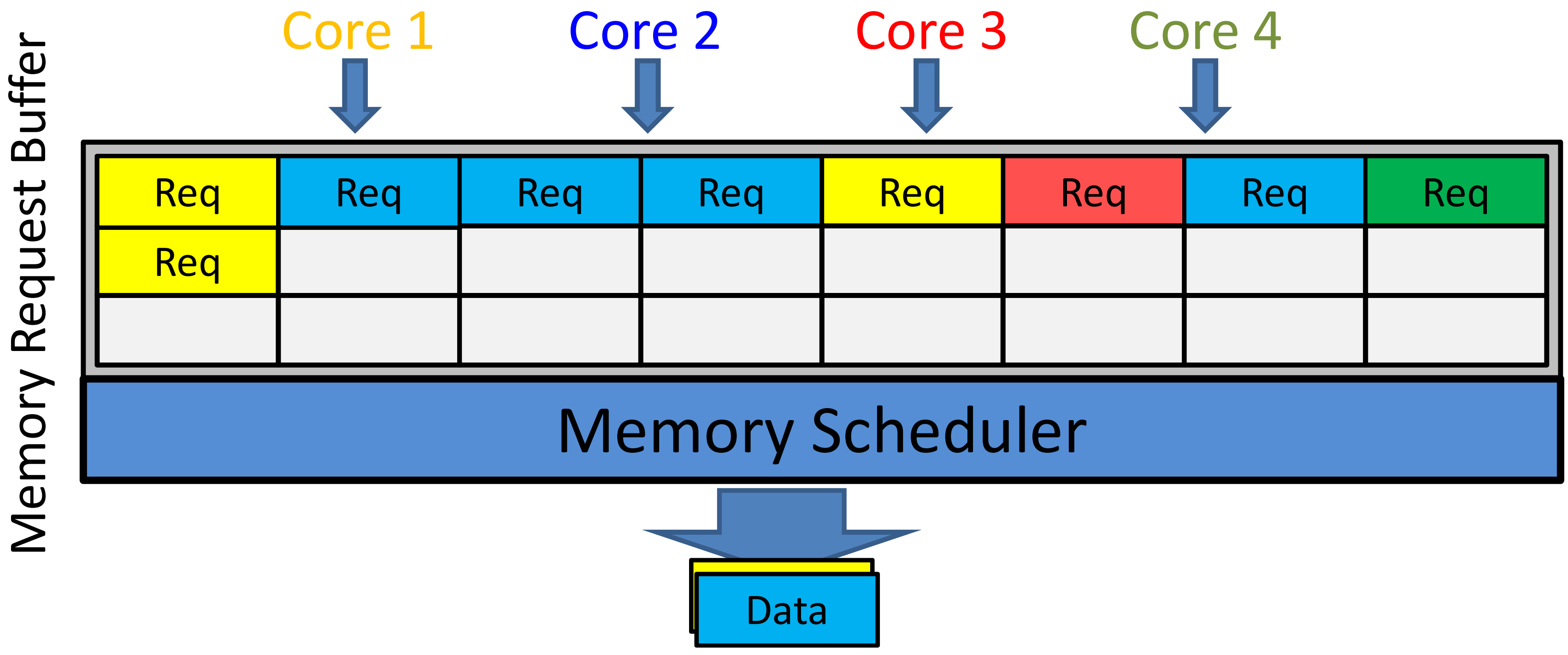
MeDiC is effective in identifying warp-type and taking advantage of divergence heterogeneity

SAFARI

Our Approach

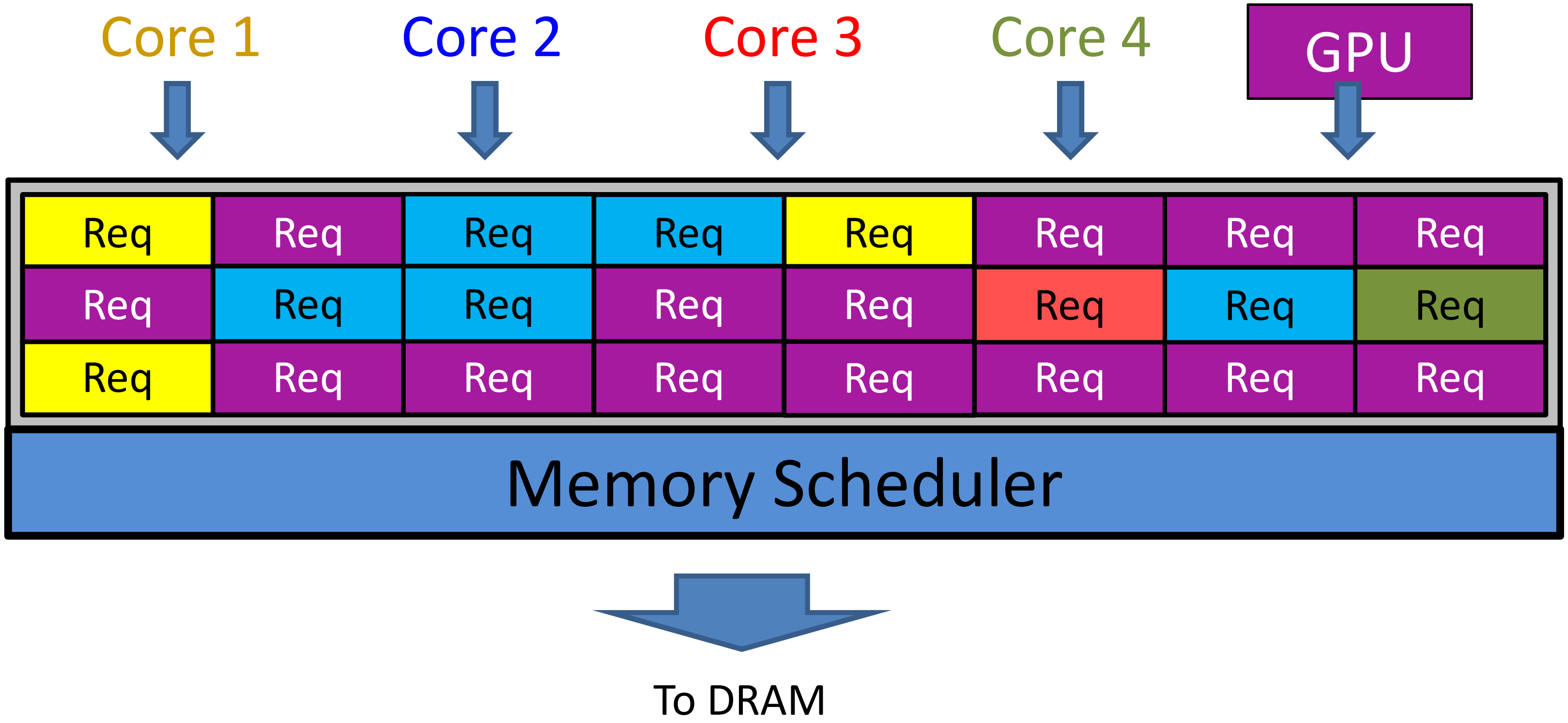
- Intra-application interference
  - Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance, PACT 2015
- Inter-application interference
  - Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems, ISCA 2012
- Inter-address-space interference
  - Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency, Submitted to MICRO 2017
  - Mosaic: A Transparent Hardware-Software Cooperative Memory Management in GPU, Submitted to MICRO 2017

Interference in the Main Memory



- All cores contend for limited off-chip bandwidth
  - Inter-application interference **degrades system performance**
  - The memory scheduler can help mitigate the problem

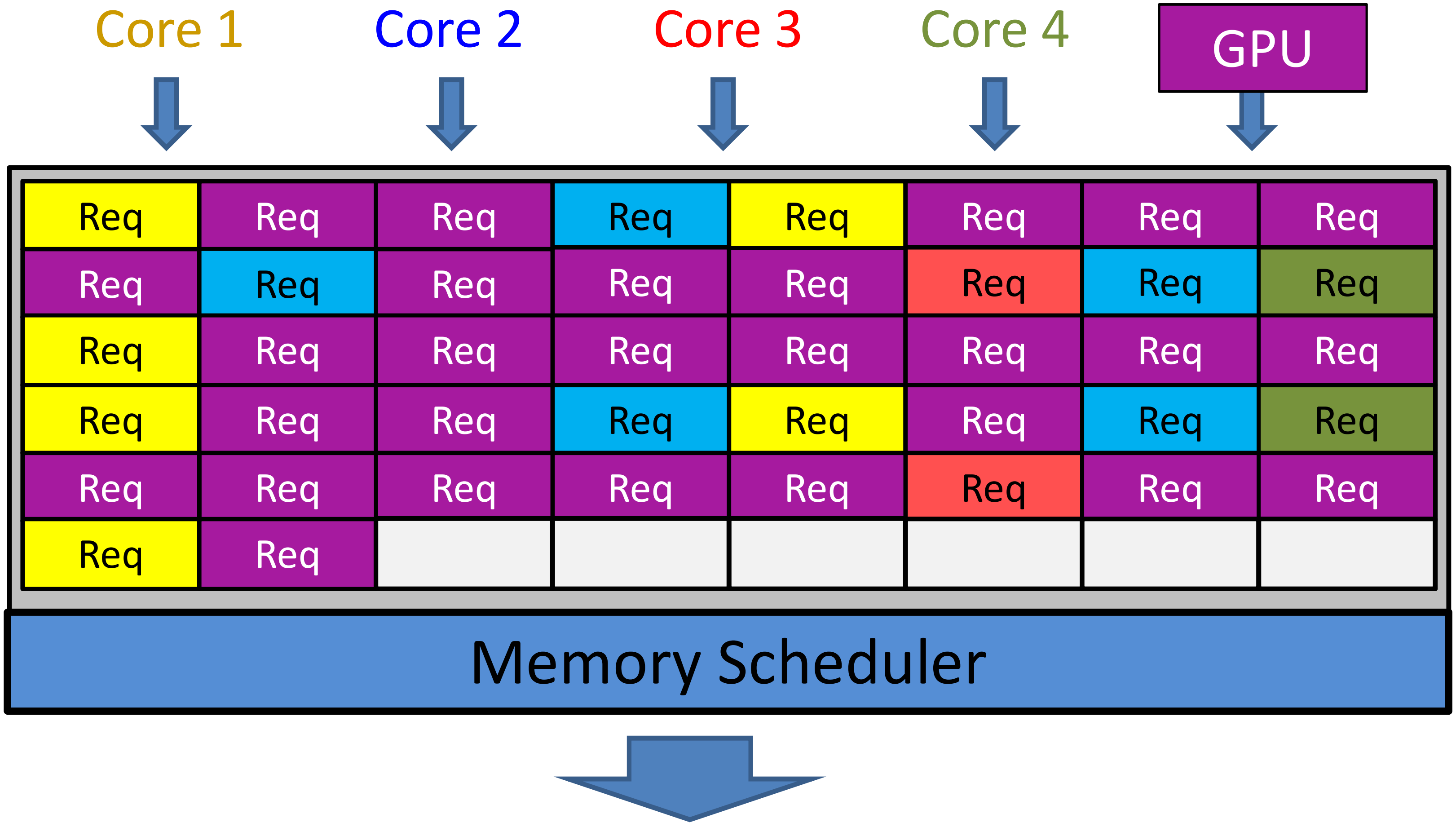
Introducing the GPU into the System



- GPU occupies a significant portion of the request buffers
  - Limits the MC’s visibility of the CPU applications’ differing memory behavior → can lead to a **poor scheduling decision**

SAFARI

Naïve Solution: Large Monolithic Buffer



Core 1

Core 2

Core 3

Core 4

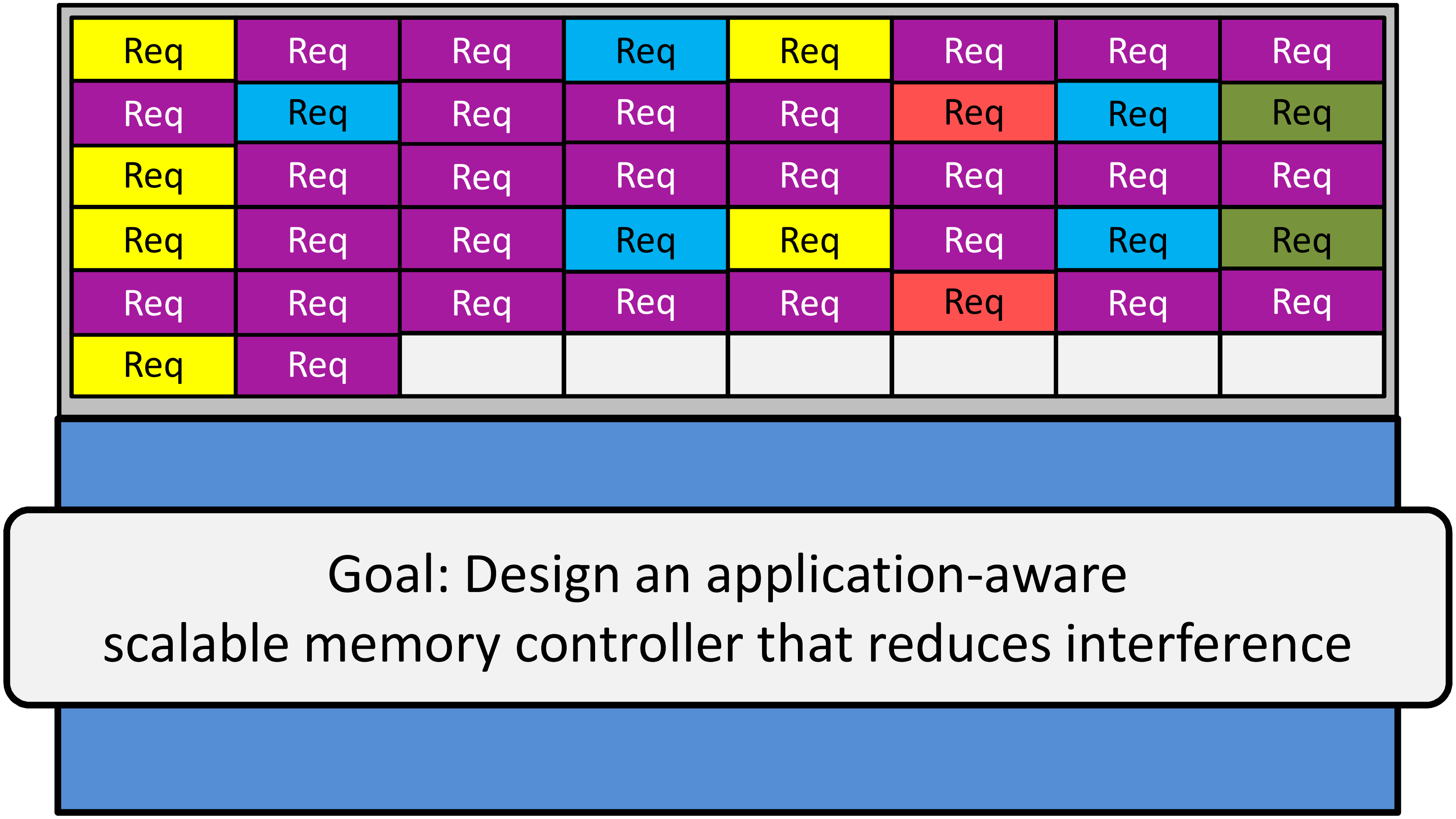
GPU

Req	Req	Req	Req	Req	Req	Req	Req
Req	Req	Req	Req	Req	Req	Req	Req
Req	Req	Req	Req	Req	Req	Req	Req
Req	Req	Req	Req	Req	Req	Req	Req
Req	Req	Req	Req	Req	Req	Req	Req
Req	Req						

Memory Scheduler

To DRAM

Problems with Large Monolithic Buffer



🟨 Req	🟪 Req	🟪 Req	🟦 Req	🟨 Req	🟪 Req	🟪 Req	🟪 Req
🟪 Req	🟦 Req	🟪 Req	🟪 Req	🟪 Req	🟥 Req	🟦 Req	🟩 Req
🟨 Req	🟪 Req	🟪 Req	🟪 Req	🟪 Req	🟪 Req	🟪 Req	🟪 Req
🟨 Req	🟪 Req	🟪 Req	🟦 Req	🟨 Req	🟪 Req	🟦 Req	🟩 Req
🟪 Req	🟪 Req	🟪 Req	🟪 Req	🟪 Req	🟥 Req	🟪 Req	🟪 Req
🟨 Req	🟪 Req						

Goal: Design an application-aware
scalable memory controller that reduces interference

- This leads to high complexity, high power, large die area

SAFARI

Key Functions of a Memory Controller

1) Maximize row buffer hits

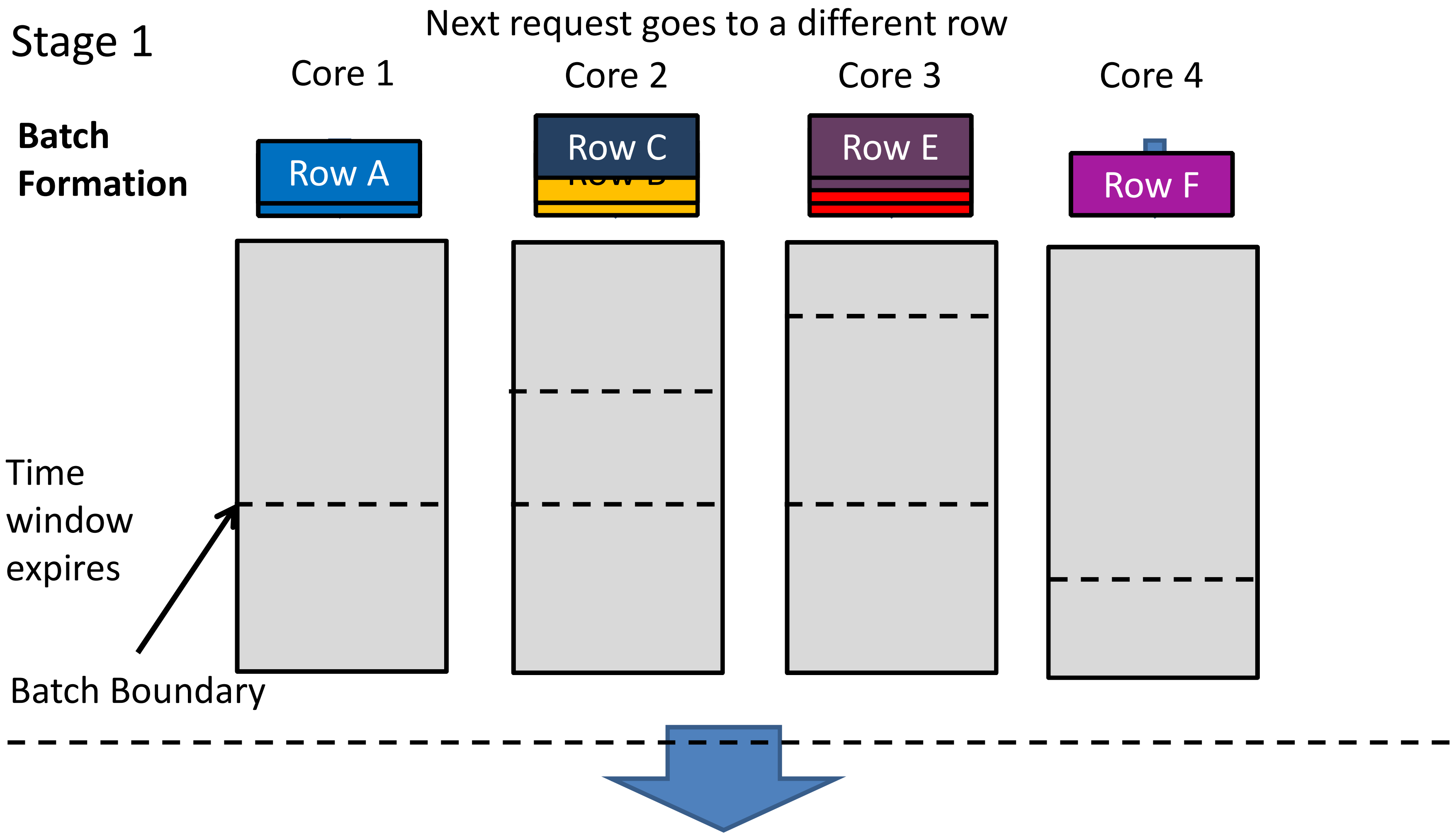
- Maximize memory bandwidth
- Stage 1: Batch Formation
  - → **Group requests** within an application into batches

2) Manage contention between applications

- Maximize system throughput and fairness
- Stage 2: Batch Scheduler
  - → **Schedule batches** from different applications

- Idea: **Decouple the functional tasks** of the memory controller
  - Partition tasks across several simpler HW structures

Stage 1: Batch Formation Example



Stage 1

Next request goes to a different row

Core 1

Core 2

Core 3

Core 4

Batch Formation

Row A

Row C

Row E

Row F

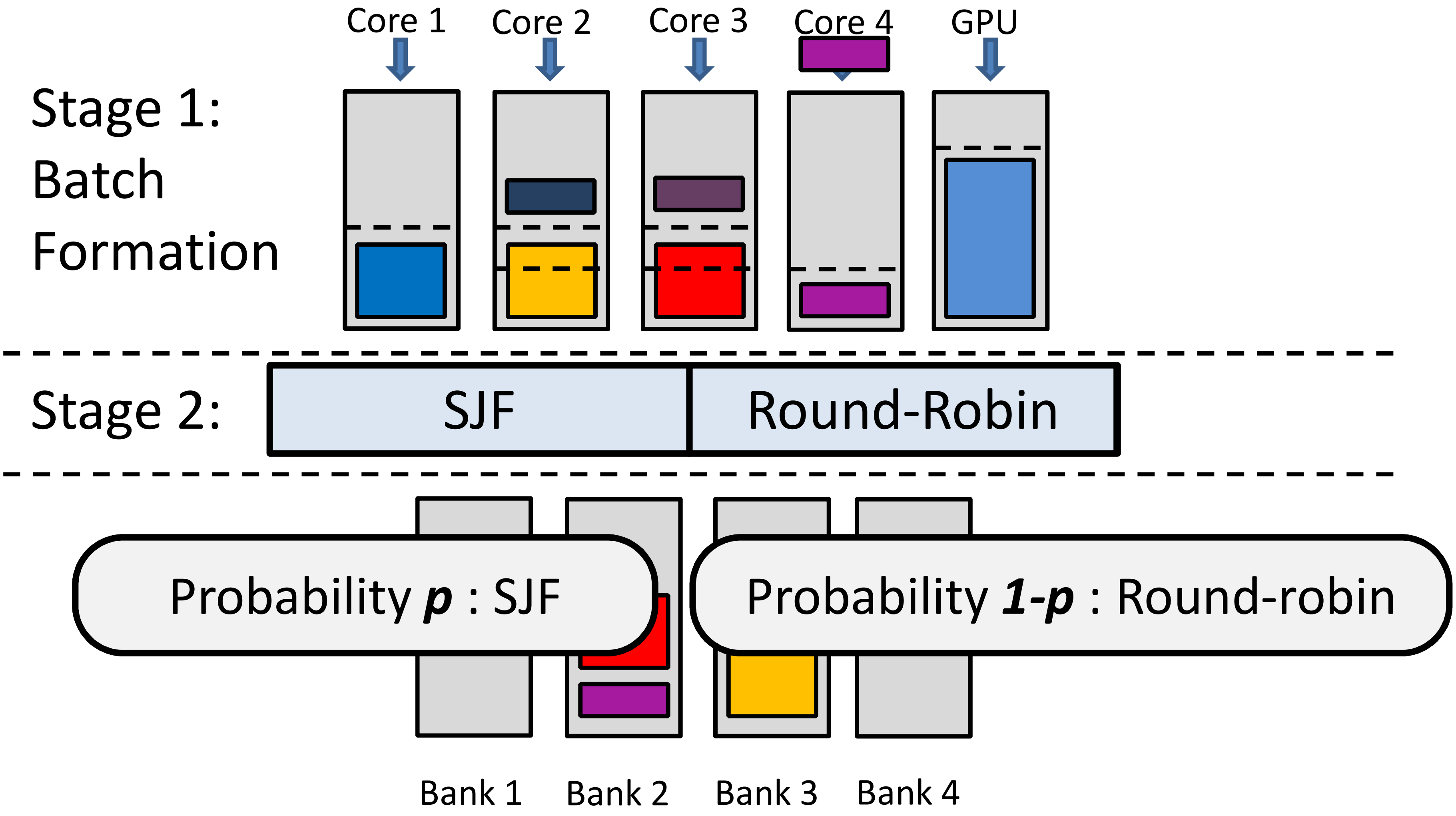
Time window expires

Batch Boundary

To Stage 2 (Batch Scheduling)

SAFARI

Staged Memory Scheduling



Probability ***p*** : SJF

Probability ***1-p*** : Round-robin

SJF prioritizes CP

Round-robin prioritizes GPU applications

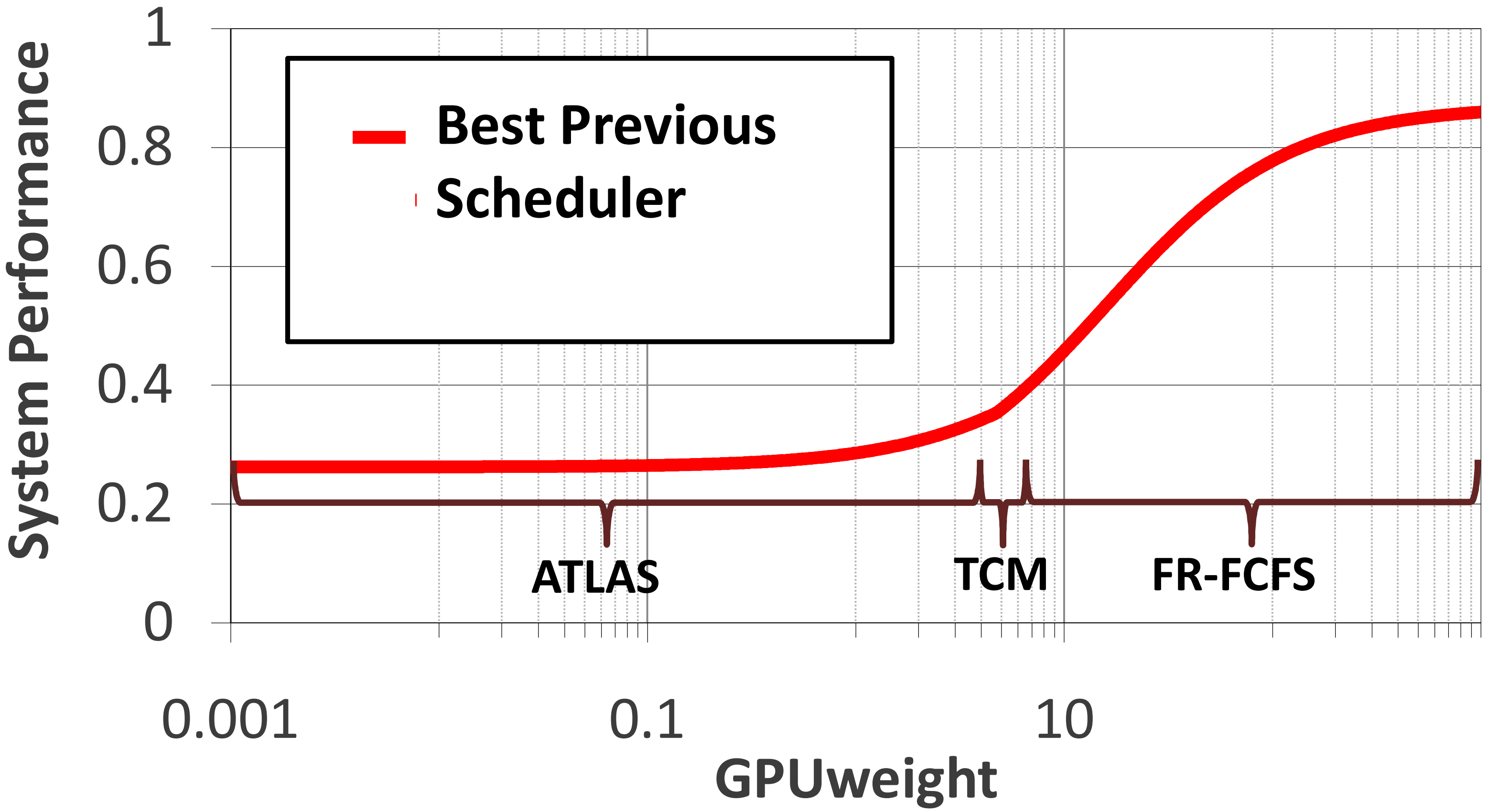
Complexity

- Compared to a row hit first scheduler, SMS consumes*
  - 66% **less area**
  - 46% **less static power**

- Reduction comes from:
  - **Simpler scheduler** (considers fewer properties at a time to make the scheduling decision)
  - **Simpler buffers** (FIFO instead of out-of-order)

*** Based on a Verilog model using 180nm library**

Performance at Different GPU Weights

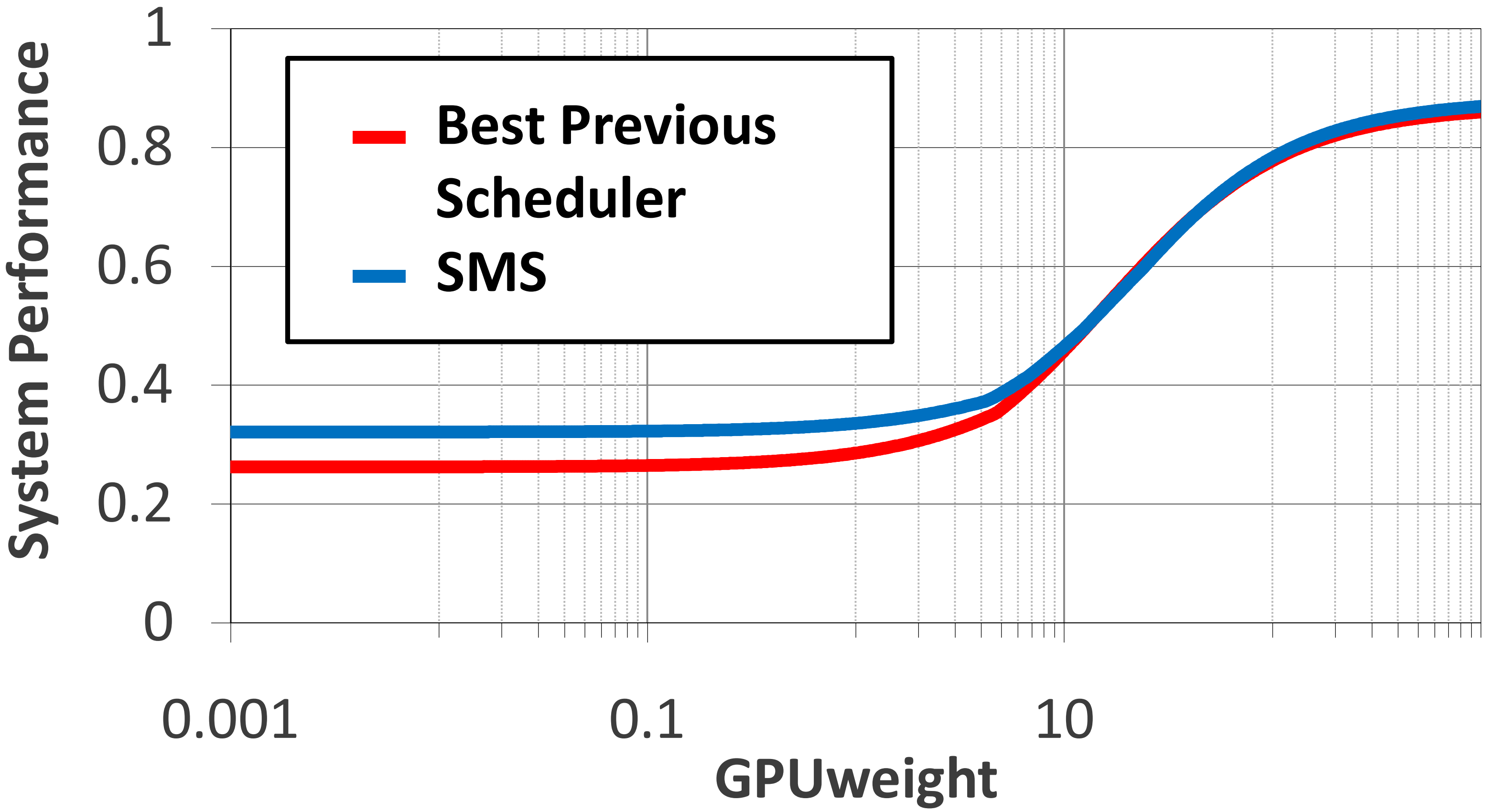


ATLAS [K
Good Multi

TCM [Kim e

FR-FCFS [Rixner et al., ISCA'00]
Good Throughput

Performance at Different GPU Weights

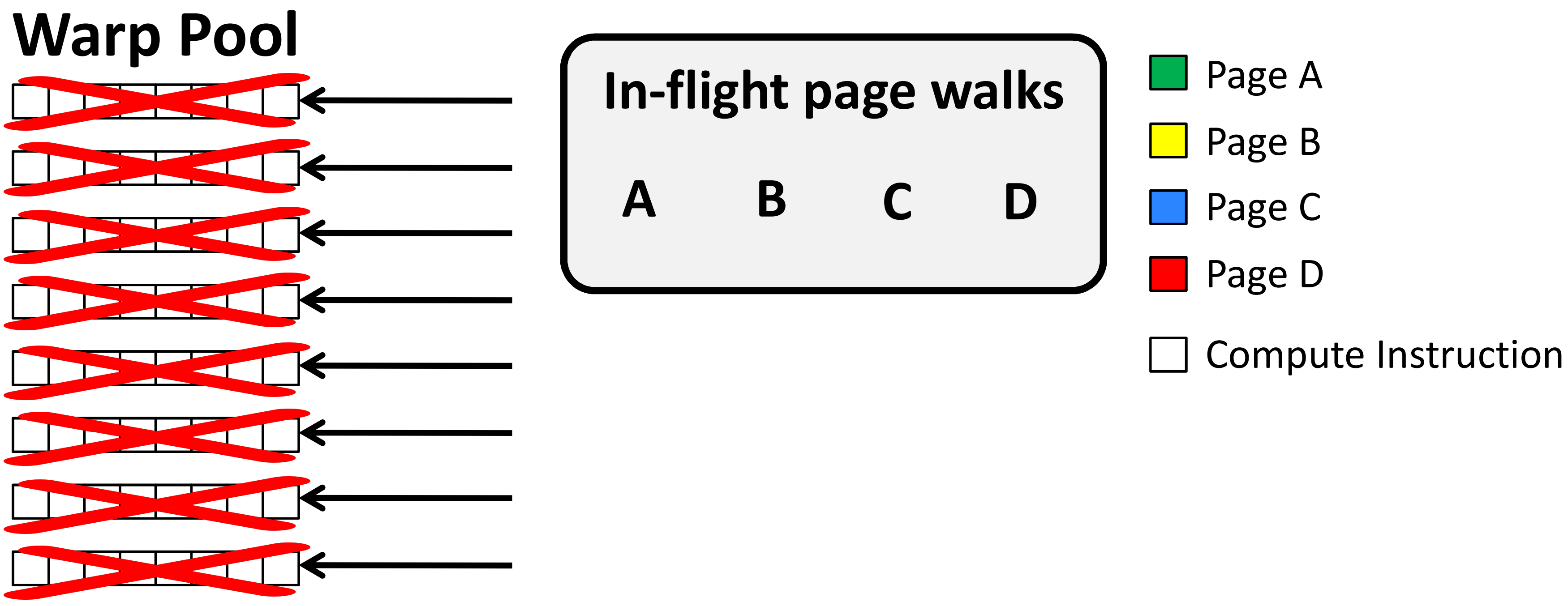


- At every GPU weight, SMS outperforms the best previous scheduling algorithm for that weight

Our Approach

- Intra-application interference
  - Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance, PACT 2015
- Inter-application interference
  - Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems, ISCA 2012
- Inter-address-space interference
  - Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency, Submitted to MICRO 2017
  - Mosaic: A Transparent Hardware-Software Cooperative Memory Management in GPU, Submitted to MICRO 2017

Bottleneck from GPU Address Translation



Warp Pool

In-flight page walks

A B C D

🟩 Page A

🟨 Page B

🟦 Page C

🟥 Page D

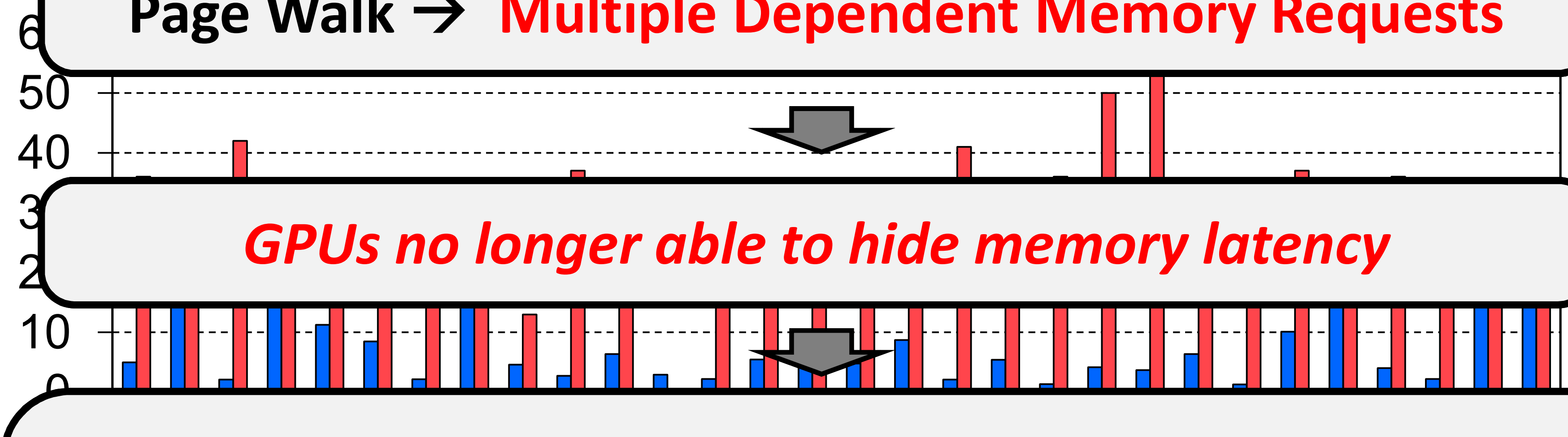
□ Compute Instruction

A single page walk can stall multiple warps

Parallelism of the GPUs → Multiple page walks

Limited Latency Hiding Capability

Page Walk → **Multiple Dependent Memory Requests**



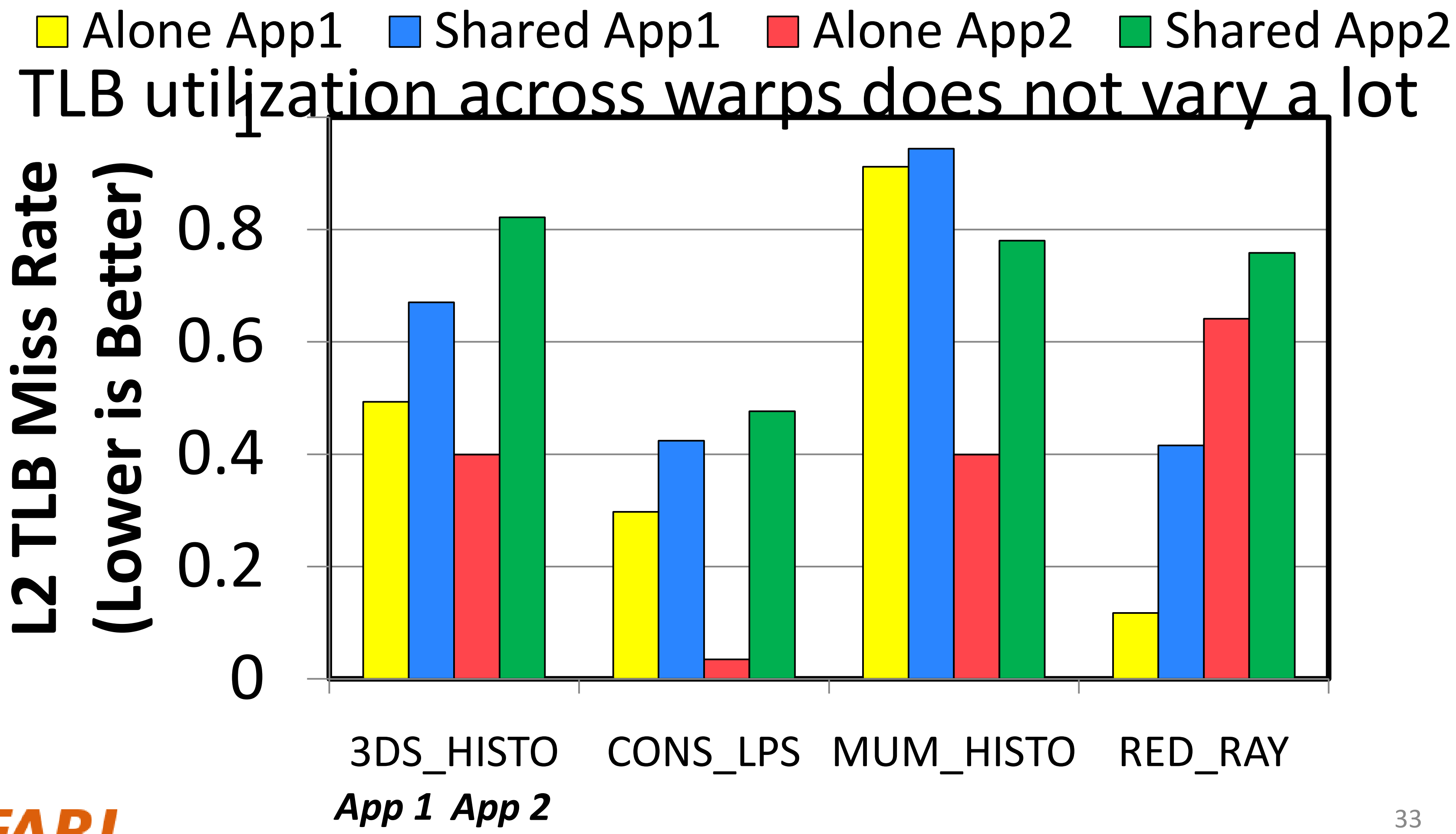
GPUs no longer able to hide memory latency

Address translation slowdowns GPUs by 47.6% on average on the state-of-the-art design [Power et al., HPCA '14]

Design Goal of MASK: Reduce the overhead of GPU address translation with a ***TLB-aware design***

Observation 1: Thrashing at the Shared TLB

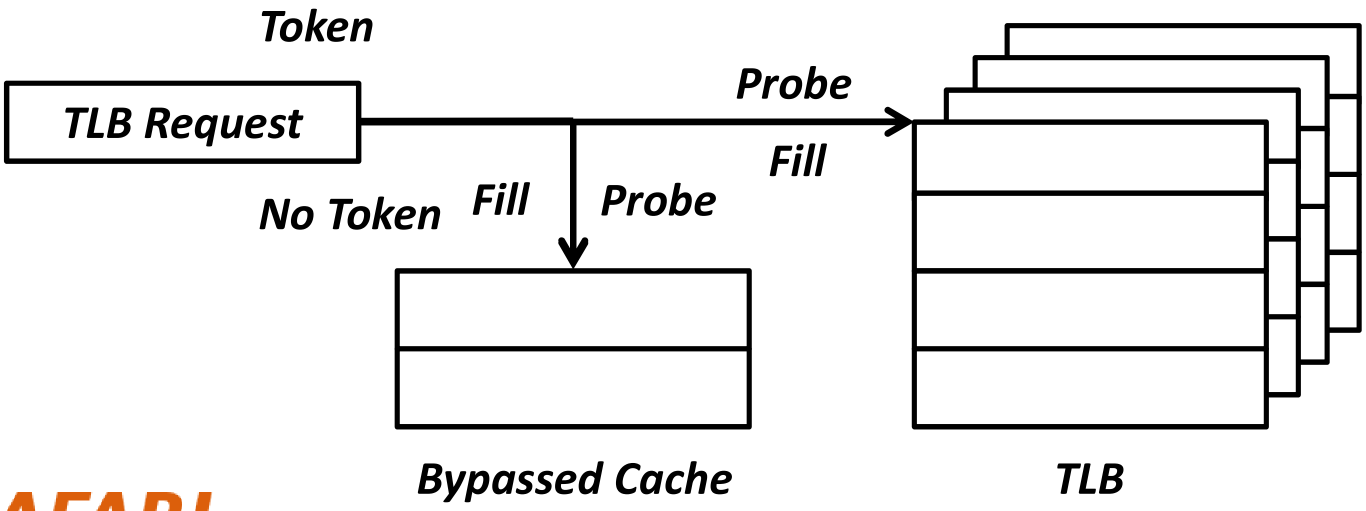
- Multiple GPU applications contend for the TLB



- TLB utilization across warps does not vary a lot

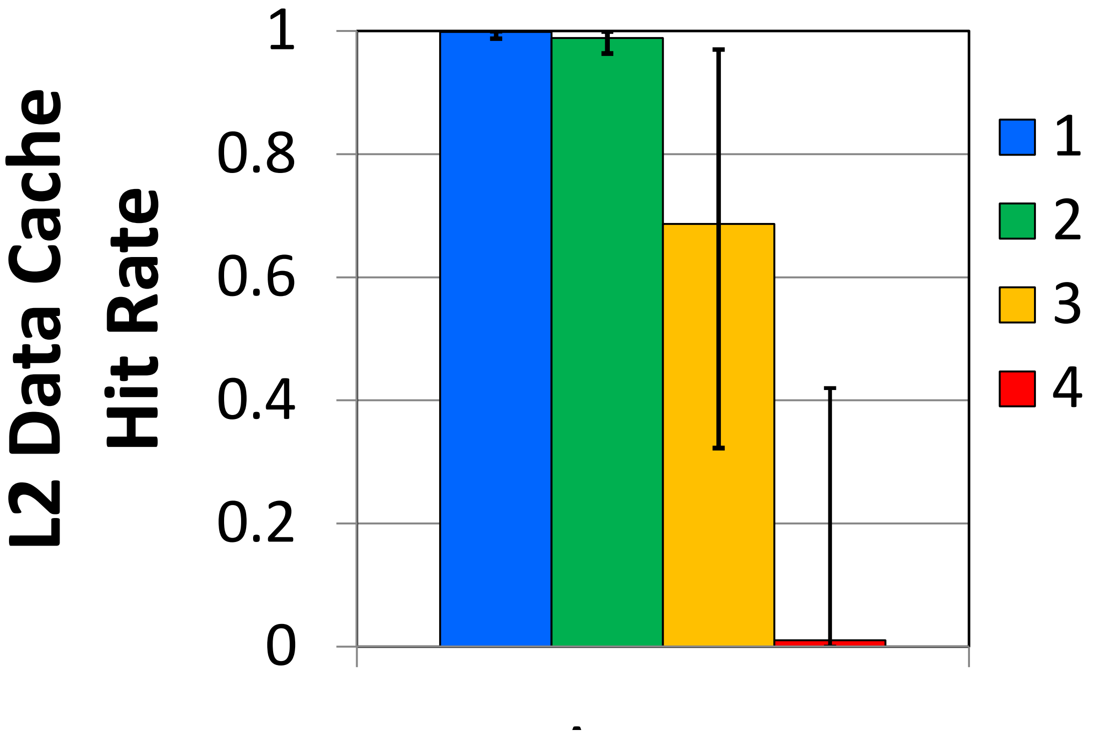
MASK: TLB-fill Bypassing

- Limit number of warps that can fill the TLB
  - Only warps with a token can fill the shared TLB
  - Otherwise fills into the tiny **bypassed cache**
- Tokens are distributed **equally across all cores**
- Within each core, **randomly distribute to warps**



Observation 2: Inefficient Caching

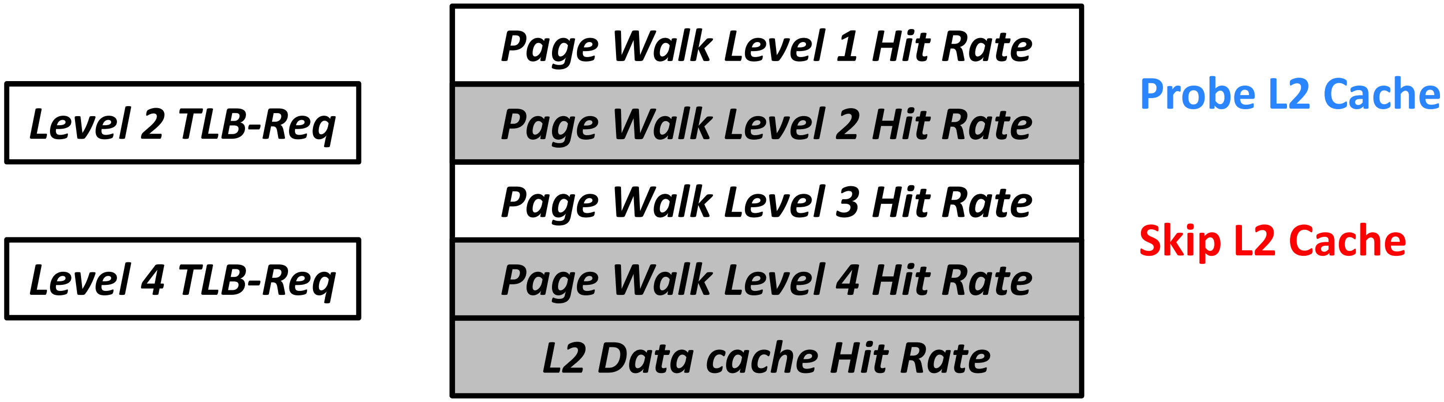
- Partial address translation data can be cached
  - – Not all TLB-related data are the same



- Cache is unaware of the page walk depth

MASK: TLB-aware Shared L2 Cache Design

- Bypass TLB-data with low hit rate



Level 2 TLB-Req

Level 4 TLB-Req

Page Walk Level 1 Hit Rate

Page Walk Level 2 Hit Rate

Page Walk Level 3 Hit Rate

Page Walk Level 4 Hit Rate

L2 Data cache Hit Rate

Probe L2 Cache

Skip L2 Cache

Benefit 1: **Better L2 cache utilization** for TLB-data

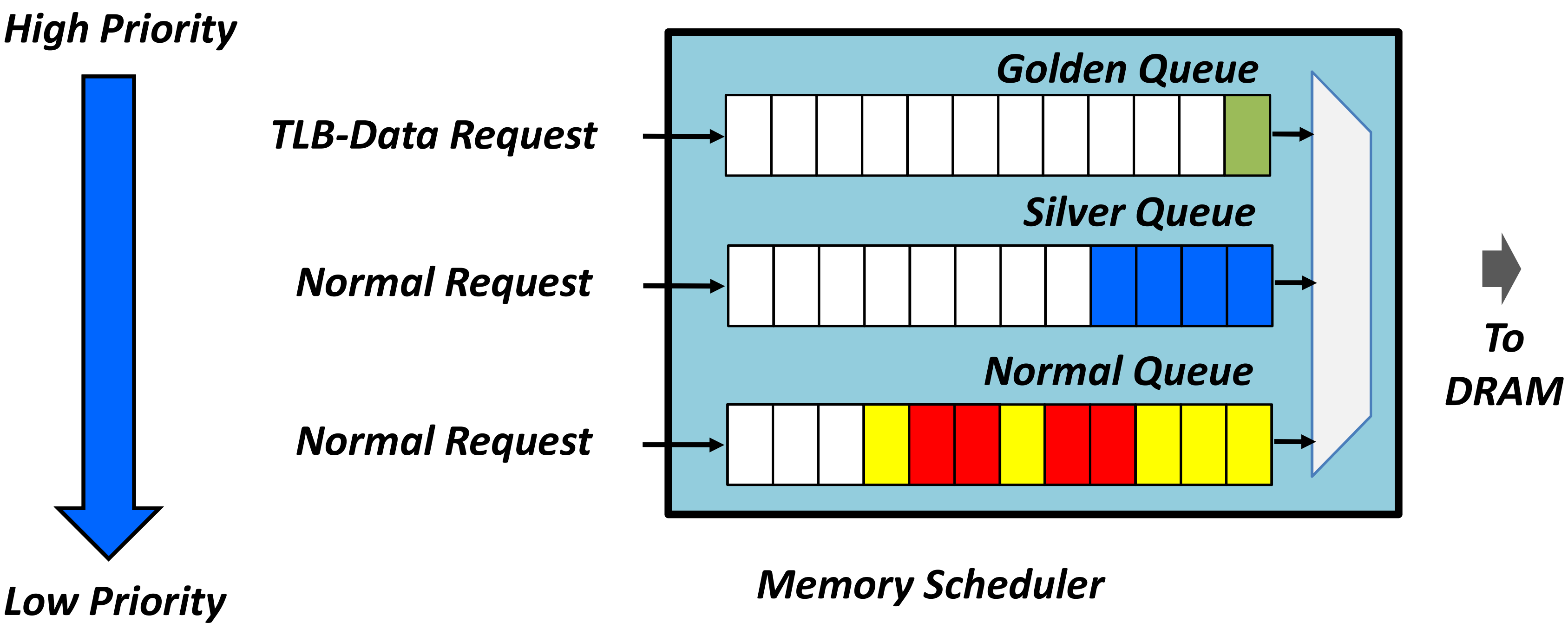
Benefit 2: TLB-data that is less likely to hit do not have to queue at L2 data cache, **reducing the latency of a page walk**

Observation 3: TLB- and App-awareness

- TLB requests are latency sensitive
- GPU memory controller is unaware of TLB-data
  - Data requests can **starve TLB-related requests**
- GPU memory controller is unaware of multiple GPU applications
  - One application can starve others

MASK: TLB-aware Memory Controller Design

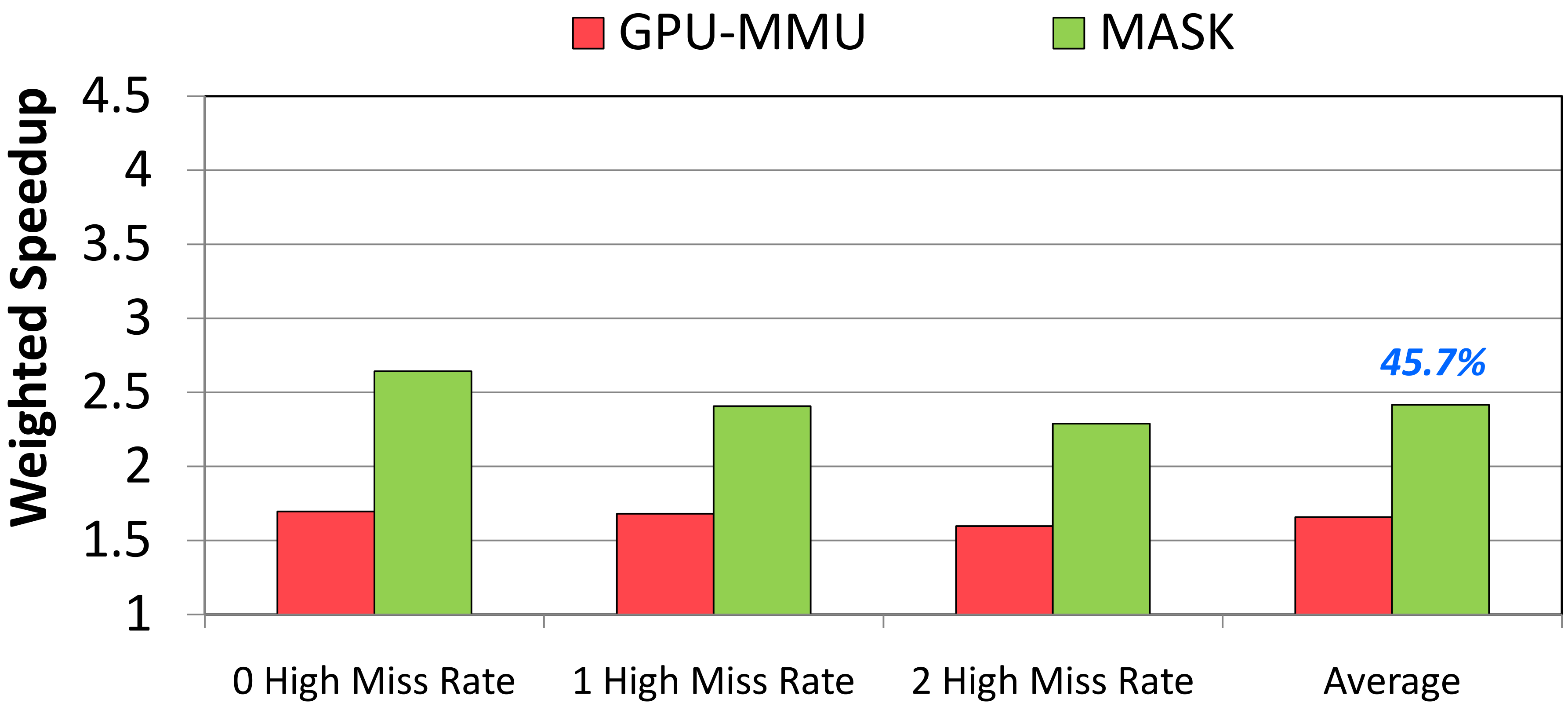
- **Goals:**
  - **Prioritize TLB-data** over normal data
  - **Ensure fairness** across all applications



Each application takes turn injecting into the silver queue

SAFARI

Results: Performance of MASK

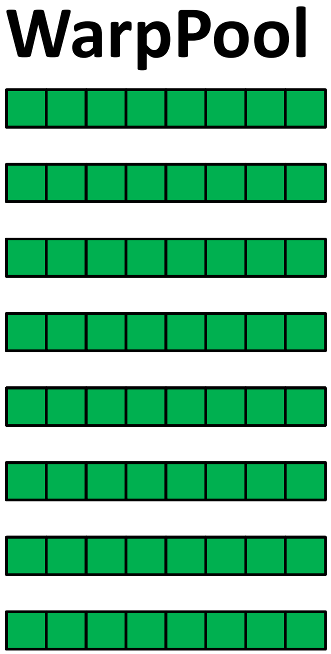


MASK is effective in reducing TLB contention and TLB-requests latency throughout the memory hierarchy

Our Approach

- Intra-application interference
  - Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance, PACT 2015
- Inter-application interference
  - Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems, ISCA 2012
- Inter-address-space interference
  - Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency, Submitted to MICRO 2017
  - Mosaic: A Transparent Hardware-Software Cooperative Memory Management in GPU, Submitted to MICRO 2017

Problems with Using Large Page



- 🟩 Page A
- 🟨 Page B
- 🟦 Page C
- 🟥 Page D

Problem: Paging large pages ***incurs significant slowdown***

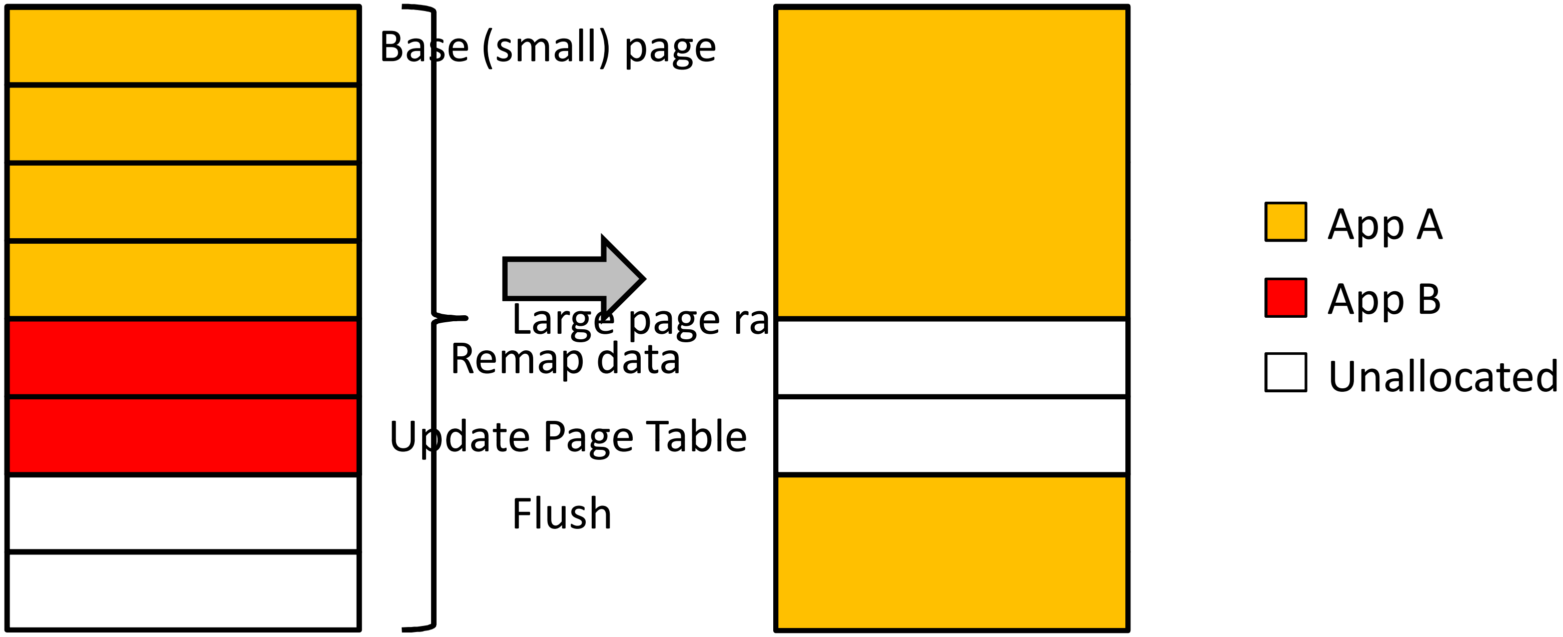
For a 2MB page size → **93% slowdown compared to 4KB**

Utilizing Multiple Page Sizes

- **Goals: Multi-page-size support**
  - Allow demand paging using small page size
  - Translate addresses using large page size
  - Low-cost page coalescing and splintering

- **Key Constraint:** No operating system support

Performance Overhead of Coalescing



Base (small) page

Large page ra

Remap data

Update Page Table

Flush

🟨 App A

🟥 App B

□ Unallocated

Significant performance overhead

GPGPU Allocation Patterns

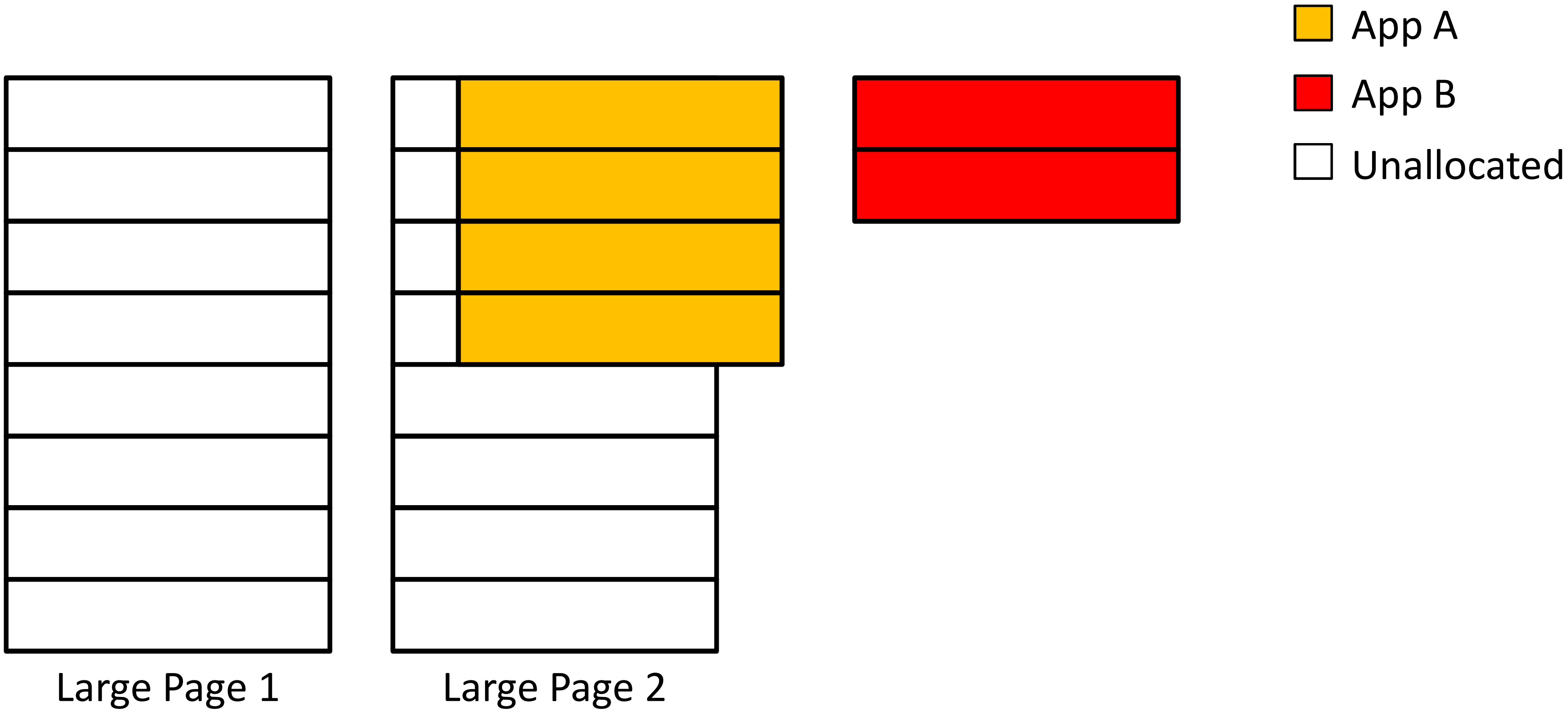
- **Observation 1:** Allocations happen infrequently
  - Allocation at the beginning of a kernel
  - Deallocation at the end of a kernel

- **Observation 2:** Allocations are typically for a large block of data

- **Mosaic utilizes these observations to provide transparent multi-page support**

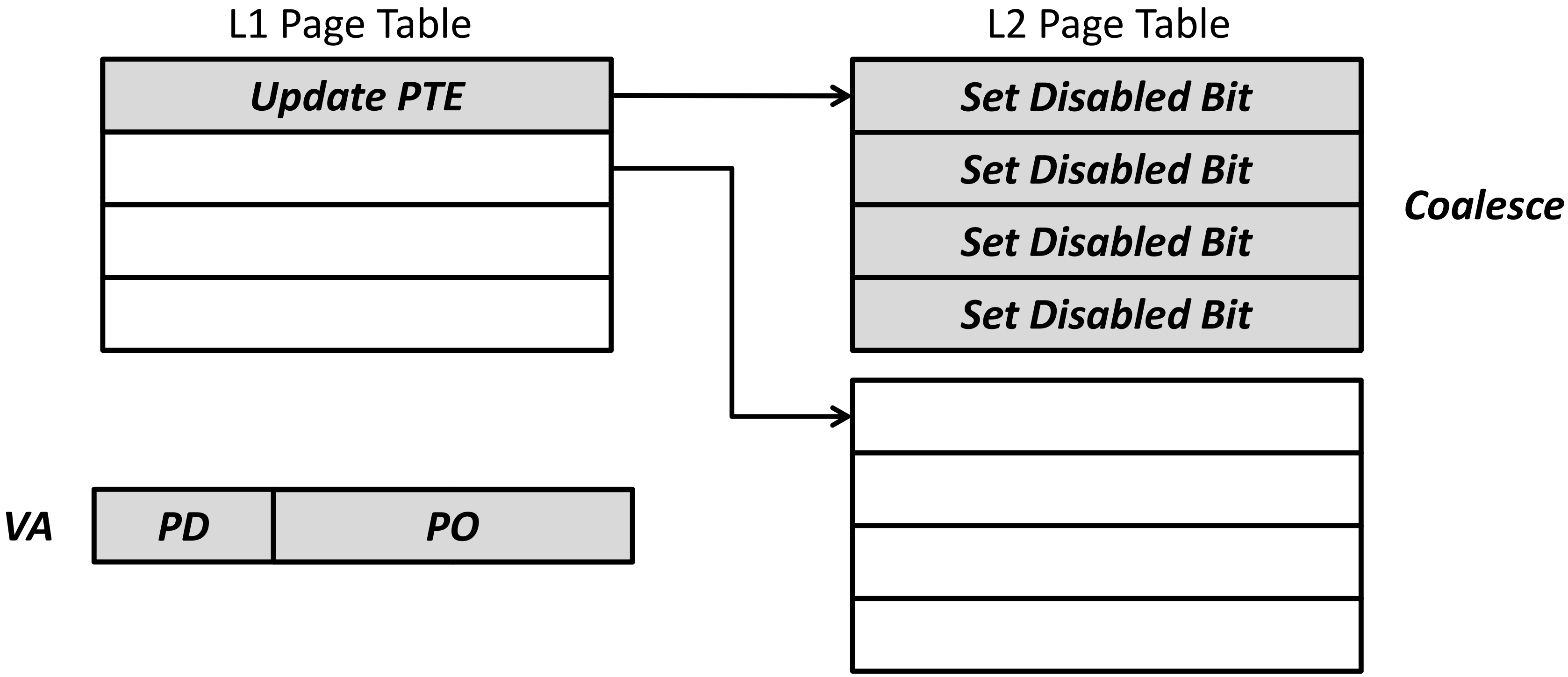
Mosaic: Enforcing a Soft Guarantee

- Small pages from different applications never fall in the same large page range



Mosaic: Low Overhead Coalescing

- **Key assumption:** Soft guarantee
  - large page range always contains pages of the same application



Benefit: No flush, no data movement

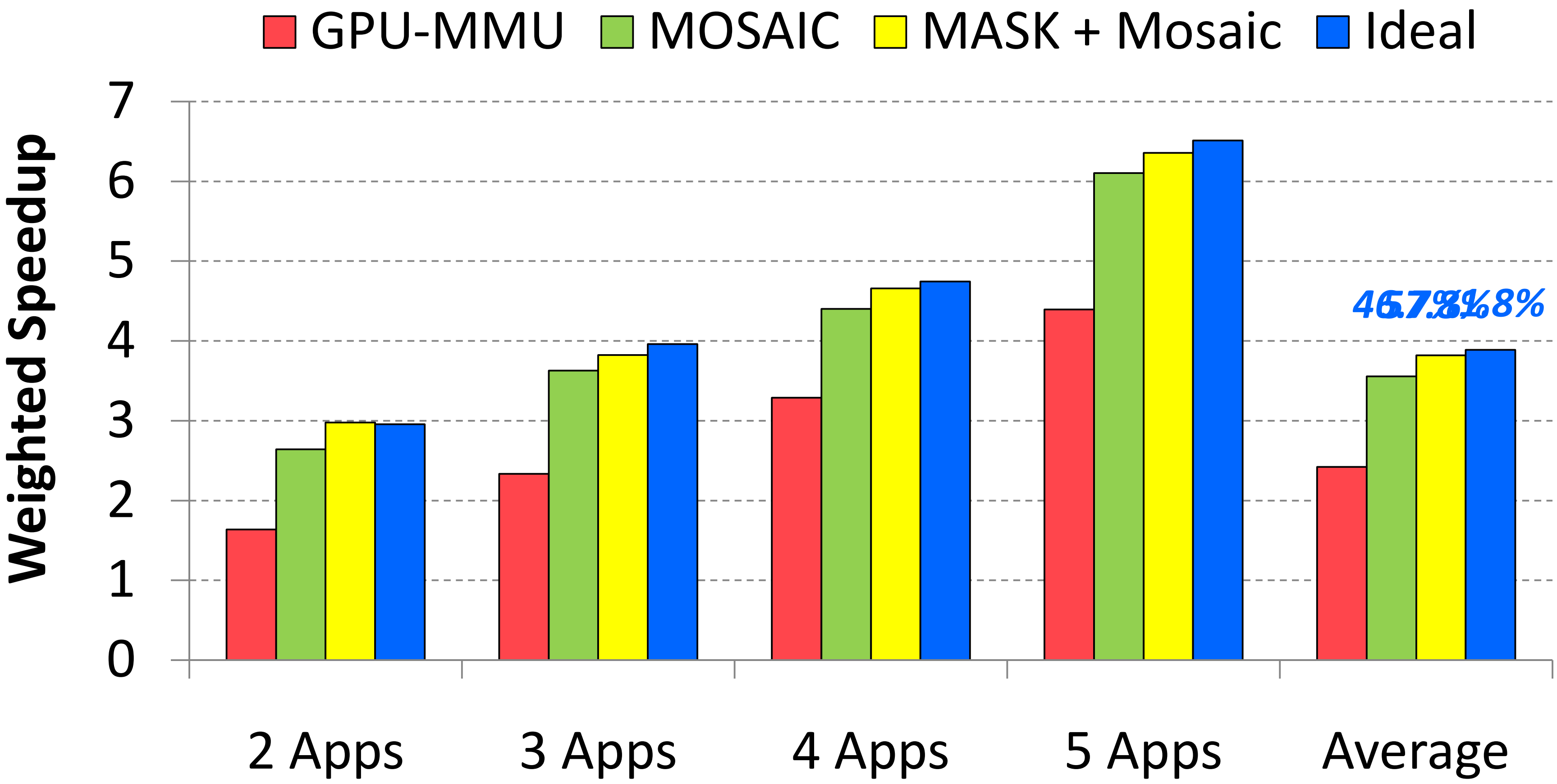
SAFARI

When to Coalesce/Splinter

- **Coalesce:**
  - **Proactively coalesce** fully allocated large pages
    - Once all data within a large page are transferred
  - **Keep translations at large page** most of the time

- **Splinter:**
  - Splinter **when the page is evicted** from the main memory
    - Enforce demand paging to be done **at small size**

Results: Performance of Mosaic



Mosaic is effective at increasing TLB range

MASK-Mosaic is effective in reducing address translation overhead

SAFARI

Mitigating Memory Interference

- Intra-application interference
  - Exploiting Inter-Warp Heterogeneity to Improve GPGPU Performance, PACT 2015
- Inter-application interference
  - Staged Memory Scheduling: Achieving High Performance and Scalability in Heterogeneous Systems, ISCA 2012
- Inter-address-space interference
  - Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency, Submitted to MICRO 2017
  - Mosaic: A Transparent Hardware-Software Cooperative Memory Management in GPU, Submitted to MICRO 2017

Summary

- **Problem:** Memory interference in GPU-based systems leads to **poor performance**
  - Intra-application interference
  - Inter-application interference
  - Inter-address-space interference
- **Thesis statement:** A combination of GPU-aware cache and memory management techniques can mitigate interference
- **Approach:** A holistic memory hierarchy design that is
  - GPU-aware
  - Application-aware
  - Divergence-aware
  - Page-walk-aware
- **Key Result:** Our mechanisms **significantly reduce memory interference** in multiple GPU-based systems

Thesis Contributions

- **In-depth analysis** of three types of memory interference in GPU-based systems
- **MeDiC** utilizes divergence heterogeneity to **reduce intra-application interference**
- **SMS** introduces CPU- and GPU-awareness memory controller design to **reduce inter-application interference**
- **MASK** proposes a TLB-aware GPU memory hierarchy to **reduce the latency of page walks**
- **Mosaic** **increases the TLB reach** resulting in the reduction of TLB contention

Future Research Directions

- **GPU memory hierarchy design**
  - Integration of **high bandwidth 3D memory**
  - Other methods to **exploit divergence heterogeneity**
- **Low-overhead virtualization support for GPUs**
  - **Interference-aware and VM-aware** designs
  - Provide limited **operating system support**
- **Co-schedule multiple GPGPU applications**
  - Kernel **scheduling** and GPU core **partitioning**
- **Sharing the GPUs for emerging applications**
  - Real-time embedded applications **with deadlines**

Other Contributions

- **GPU Designs:**
  - Managing GPU concurrency
    - *Kayiran et al., MICRO'14*
  - Improving GPU efficiency
    - *Vijaykumar et al. ISCA'15*
    - *Kayiran et al., PACT '16*
- **DRAM Designs:**
  - Low-latency DRAM
    - *Seshadri et al., MICRO '13*
    - *Lee et al. PACT '15*
    - *Lee et al., SIGMETRICS '17*
  - Hybrid memory
    - *Yoon et al., ICCD'12*

Other Contributions

- **Network-on-chip Designs:**
  - Energy efficient on-chip network design
    - *Chang et al., SAFARI Tech Report 2011-006*
    - *Fallin et al., NOCs '12*
    - *Chang et al., SBAC-PAD '12*
    - *Das et al., HPCA'13*
    - *Ausavarungnirun et al., SBAC-PAD'14,*
    - *Ausavarungnirun et al., PARCO '16*
  - Handling faults in on-chip network
    - *Fattah et al., NoCs '15*
- **Data center power management**
    - *Li et al., HPCA '16*

Acknowledgements

- My advisor: Onur Mutlu
- James Hoe, Gabriel Loh, Chris Rossbach, Kayvon Fatahalian
- SAFARI group members

Techniques for Shared Resource Management in Systems with GPUs

Thesis Defense

Rachata Ausavarungnirun

Committees:

Advisor: Onur Mutlu (CMU and ETH Zürich)
James C. Hoe (CMU)
Kayvon Fatahalian (CMU)
Gabriel H. Loh (AMD Research)
Christopher J. Rossbach (UT Austin and VMware Research)

SAFARI

Carnegie Mellon