MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency

Rachata Ausavarungnirun

Vance Miller Joshua Landgraf Saugata Ghose

Jayneel Gandhi Adwait Jog Christopher J. Rossbach Onur Mutlu

Carnegie Mellon

TEXAS
The University of Texas at Austin

ETH Zürich

vmware

SAFARI

Executive Summary

Problem: Address translation overheads
limit the latency hiding capability of a GPU

High contention at the shared TLB

Low L2 cache utilization

Large performance loss vs. no translation

Key Idea
Prioritize **address translation requests** over **data requests**

MASK: a GPU memory hierarchy that

A. Reduces shared TLB contention
B. Improves L2 cache utilization
C. Reduces page walk latency

MASK **improves system throughput by 57.8%** on average over state-of-the-art address translation mechanisms

Outline

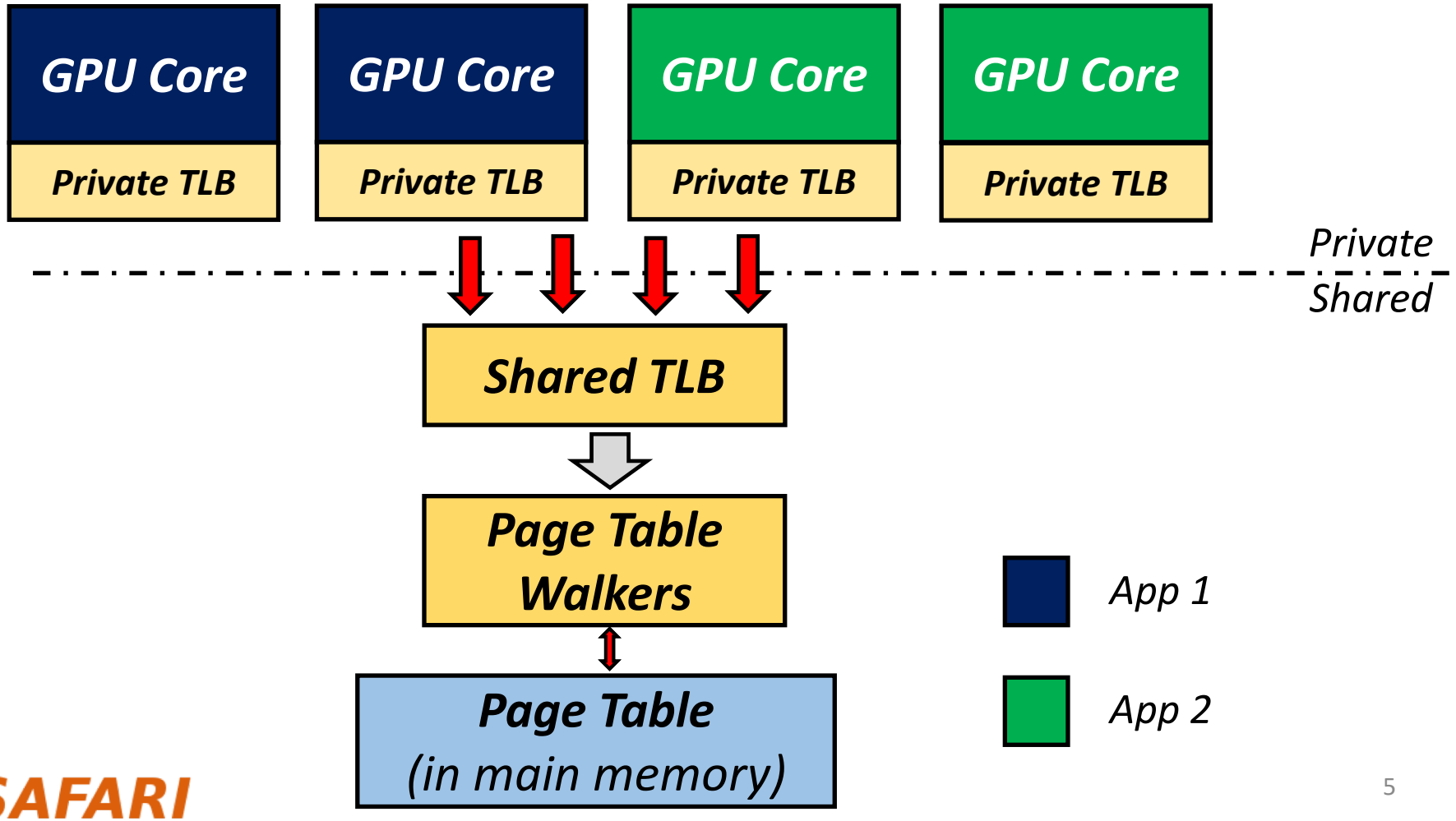
- Executive Summary
- **Background, Key Challenges and Our Goal**
- **MASK:** A Translation-aware Memory Hierarchy
- Evaluation
- Conclusion

Why Share Discrete GPUs?

- Enables multiple GPGPU applications to run concurrently
- Better resource utilization
  - An application often cannot utilize an entire GPU
  - Different compute and bandwidth demands
- Enables GPU sharing in the cloud
  - Multiple users spatially share each GPU

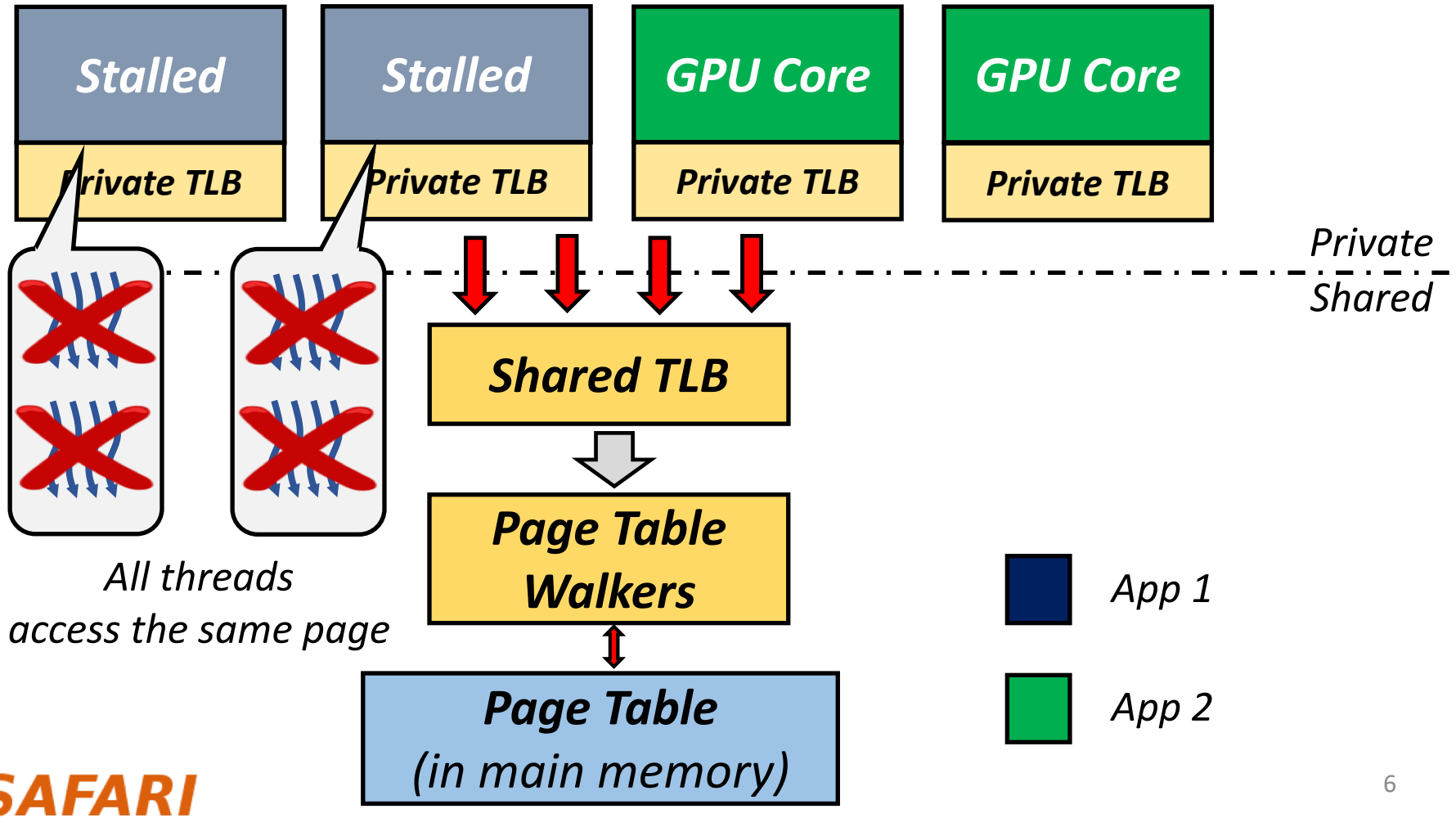
Key requirement: fine-grained memory protection

State-of-the-art Address Translation in GPUs

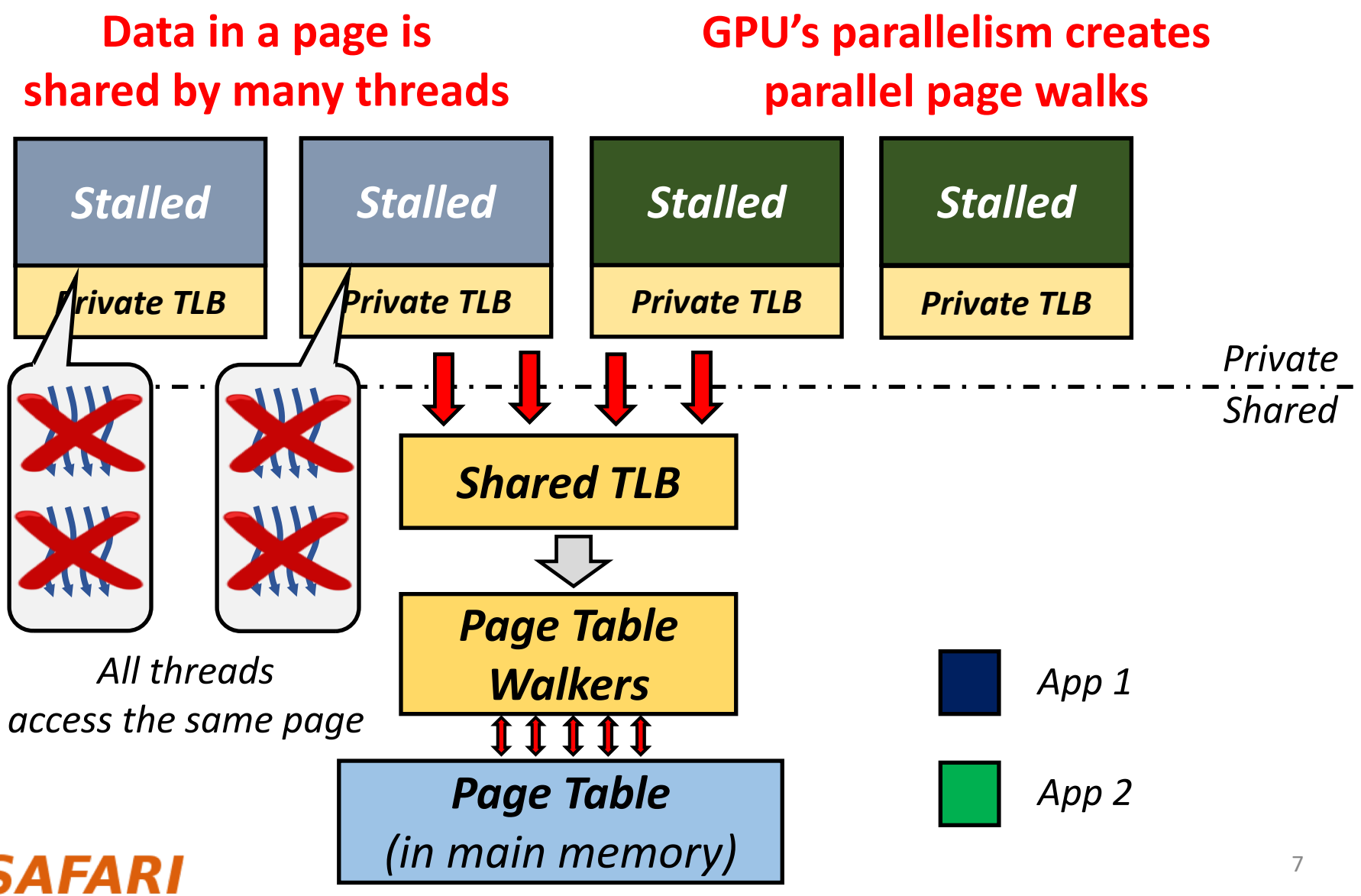


A TLB Miss Stalls Multiple Warps

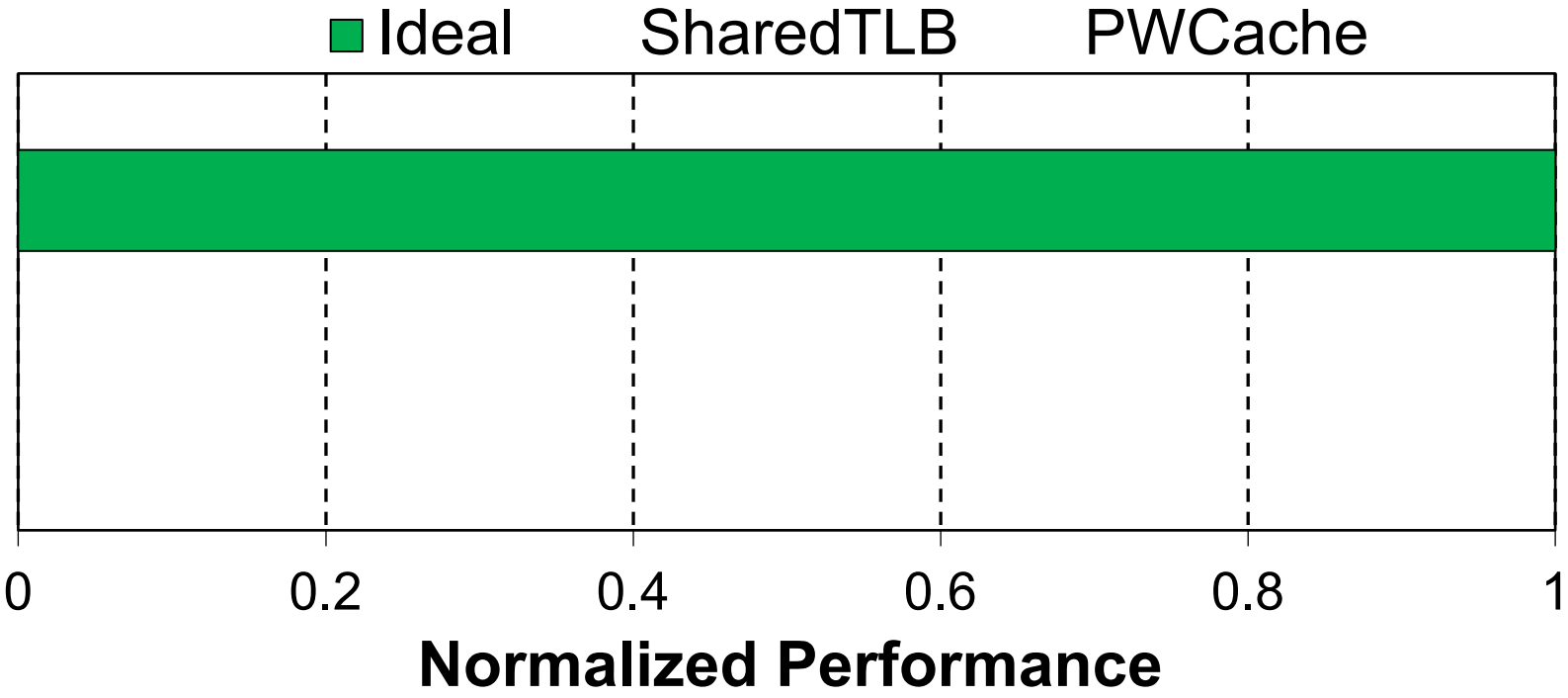
Data in a page is shared by many threads



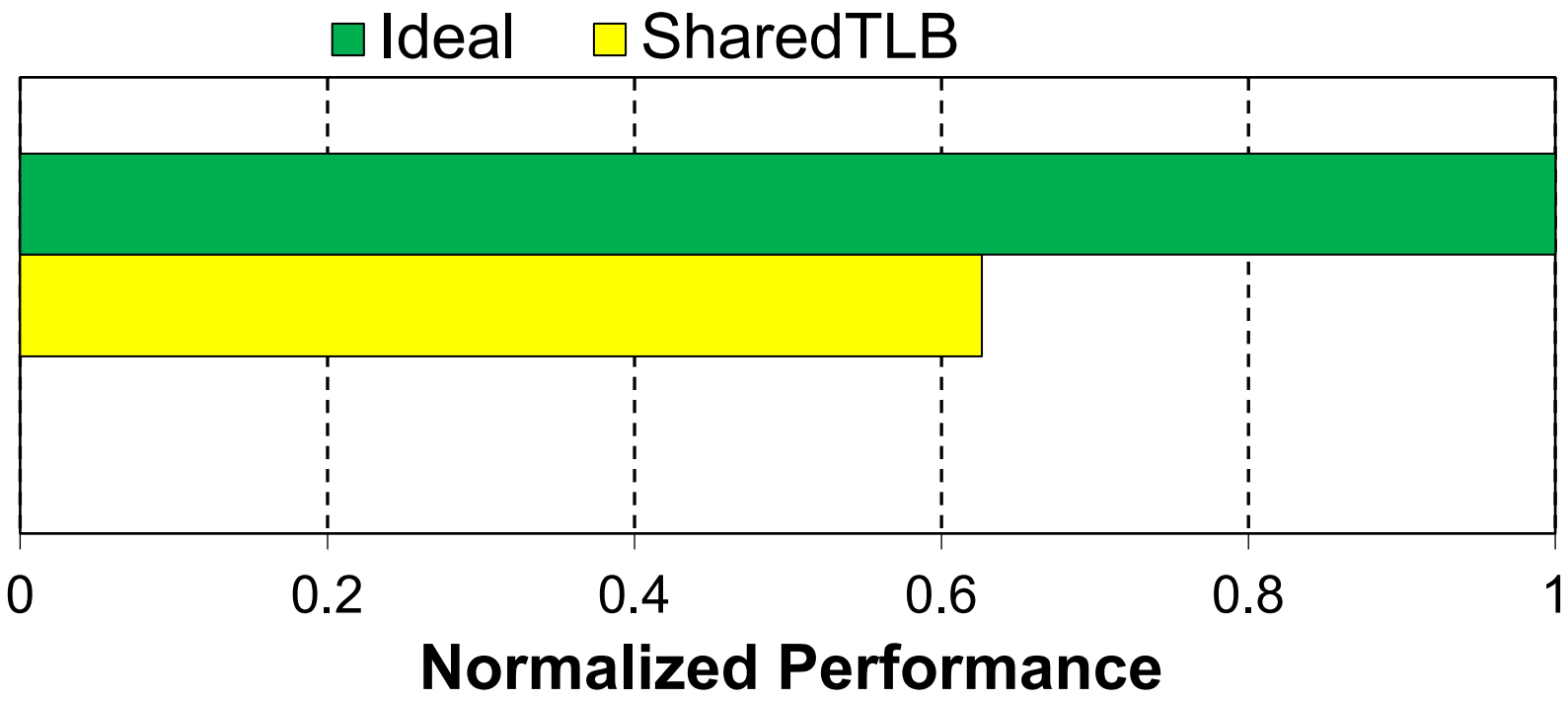
Multiple Page Walks Happen Together



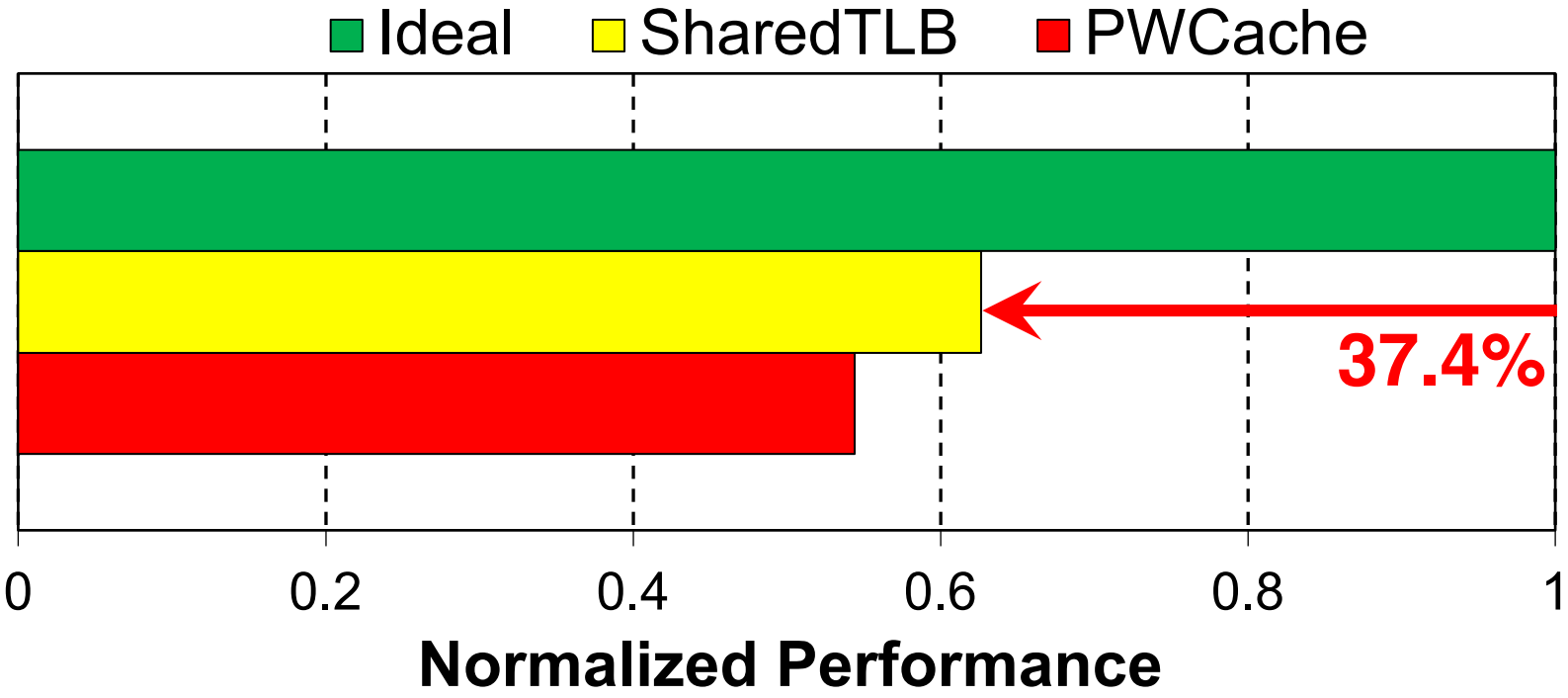
Effect of Translation on Performance



Effect of Translation on Performance



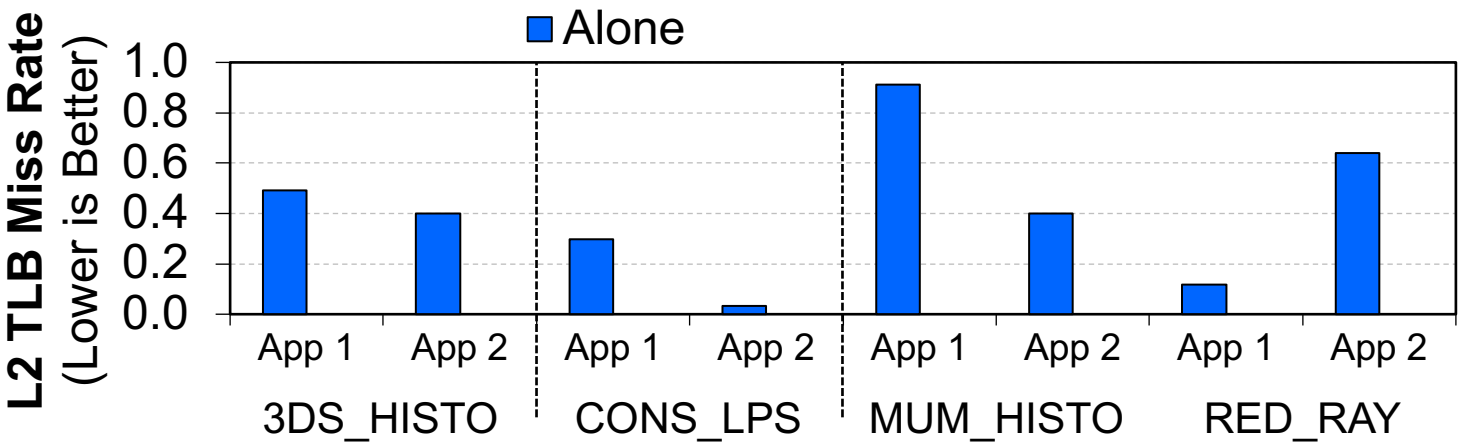
Effect of Translation on Performance



What causes the large performance loss?

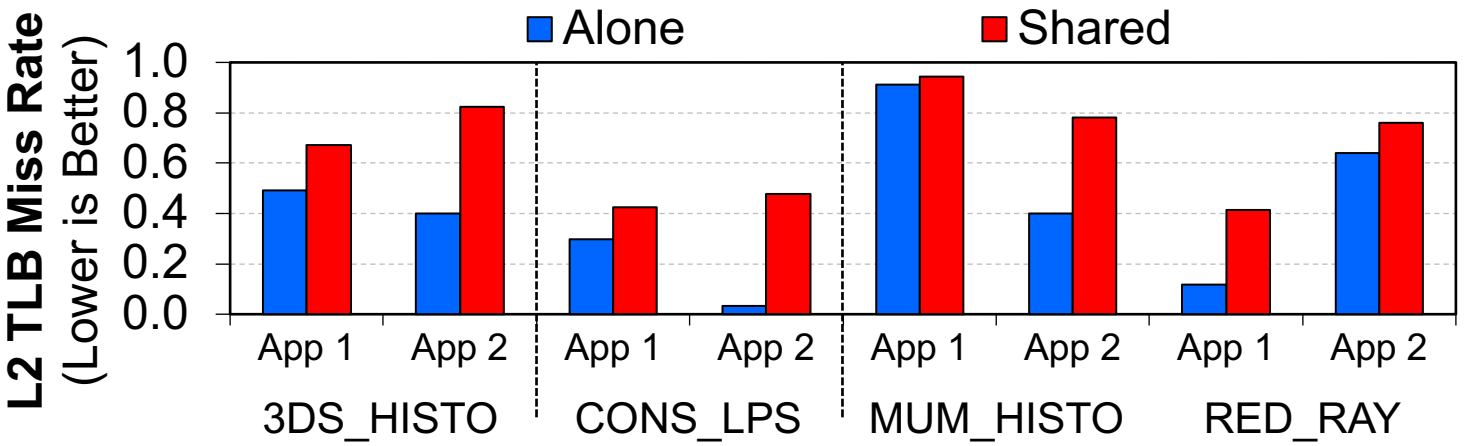
Problem 1: Contention at the Shared TLB

- Multiple GPU applications contend for the TLB



Problem 1: Contention at the Shared TLB

- Multiple GPU applications contend for the TLB



Contention at the shared TLB leads to lower performance

Problem 2: Thrashing at the L2 Cache

- L2 cache can be used to reduce page walk latency
  → **Partial translation data can be cached**

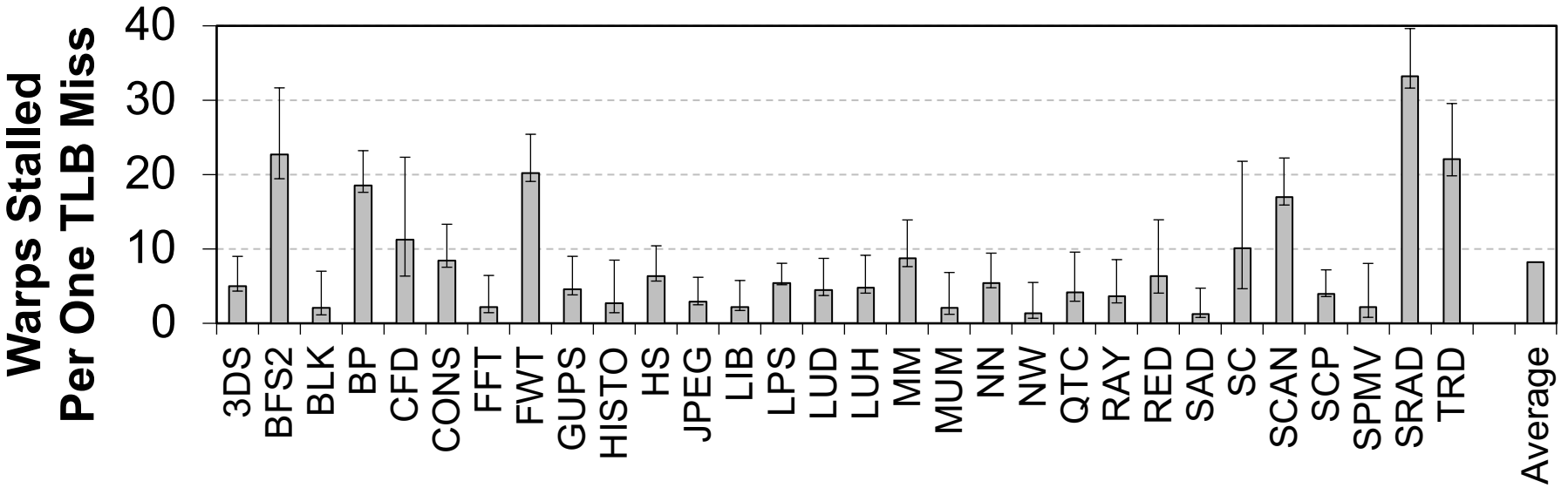
- **Thrashing Source 1:** Parallel page walks
  → **Different address translation data evicts each other**

- **Thrashing Source 2:** GPU memory intensity
  → **Demand-fetched data evicts address translation data**

L2 cache is **ineffective** at reducing page walk latency

Observation: Address Translation Is Latency Sensitive

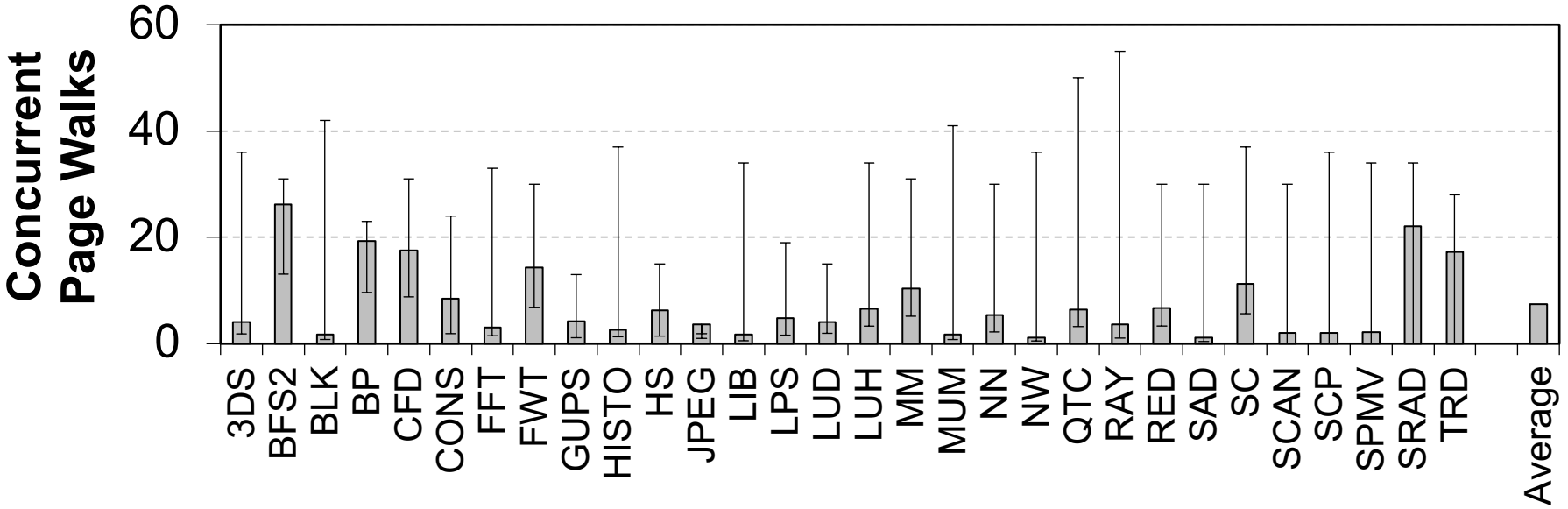
- Multiple warps share data from a single page



A single TLB miss causes **8 warps** to stall on average

Observation: Address Translation Is Latency Sensitive

- Multiple warps share data from a single page
- GPU's parallelism causes multiple concurrent page walks



High address translation latency → **More stalled warps**

Our Goals

- **Reduce shared TLB contention**
- **Improve L2 cache utilization**
- **Lower page walk latency**

Outline

- Executive Summary
- Background, Key Challenges and Our Goal
- **MASK:** A Translation-aware Memory Hierarchy
- Evaluation
- Conclusion

MASK: A Translation-aware Memory Hierarchy

- **Reduce shared TLB contention**
  - **A. TLB-fill Tokens**
- **Improve L2 cache utilization**
  - **B. Translation-aware L2 Bypass**
- **Lower page walk latency**
  - **C. Address-space-aware Memory Scheduler**

A: TLB-fill Tokens

- **Goal:** Limit the number of warps that can fill the TLB
  - → A warp with **a token** fills the **shared TLB**
  - → A warp with **no token** fills a very small **bypass cache**

- Number of tokens changes based on TLB miss rate
  - → Updated every epoch

- Tokens are assigned based on warp ID

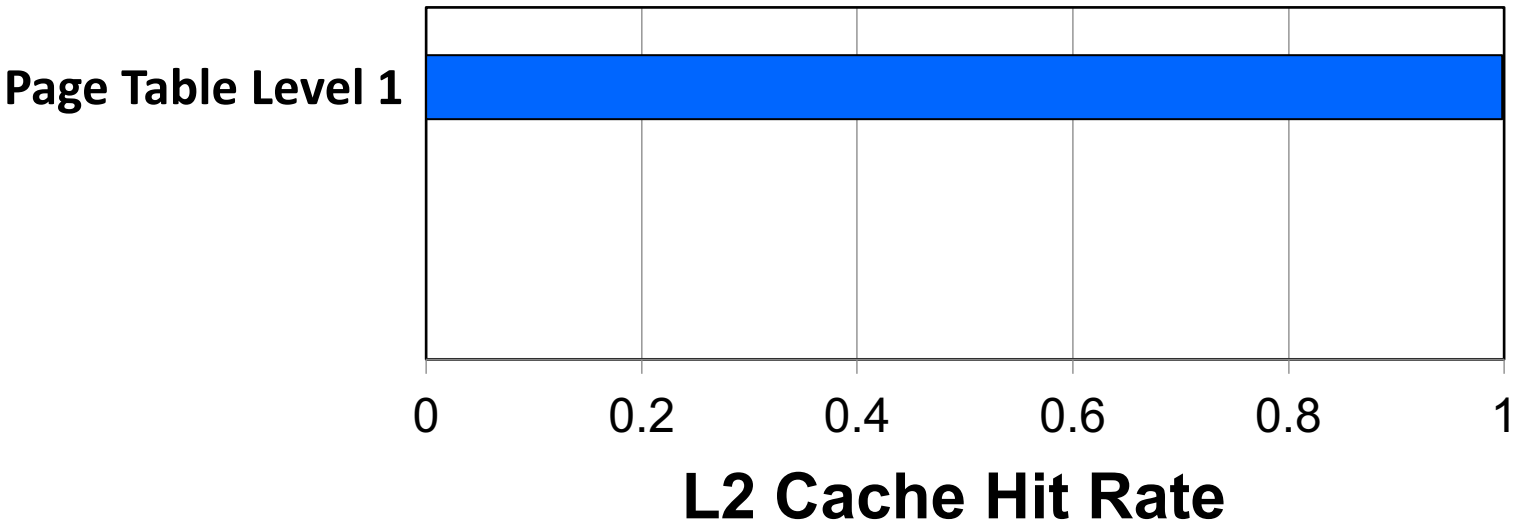
Benefit: **Limits contention** at the shared TLB

MASK: A Translation-aware Memory Hierarchy

- Reduce shared TLB contention
  - A. TLB-fill Tokens
- **Improve L2 cache utilization**
  - **B. Translation-aware L2 Bypass**
- Lower page walk latency
  - C. Address-space-aware Memory Scheduler

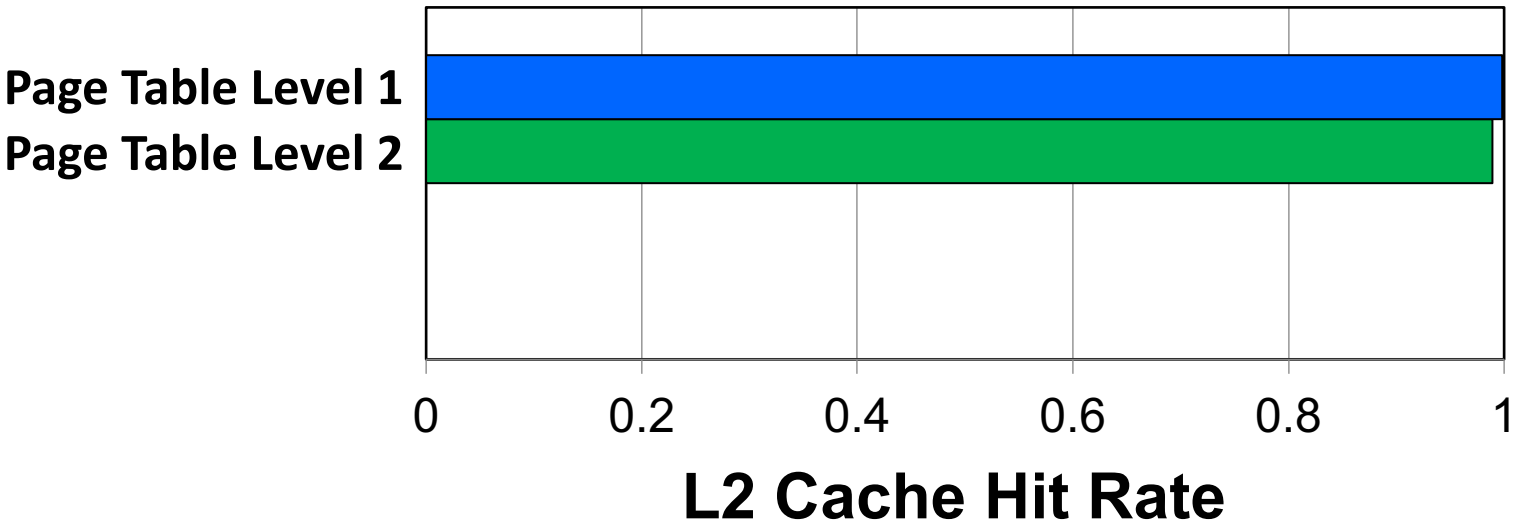
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



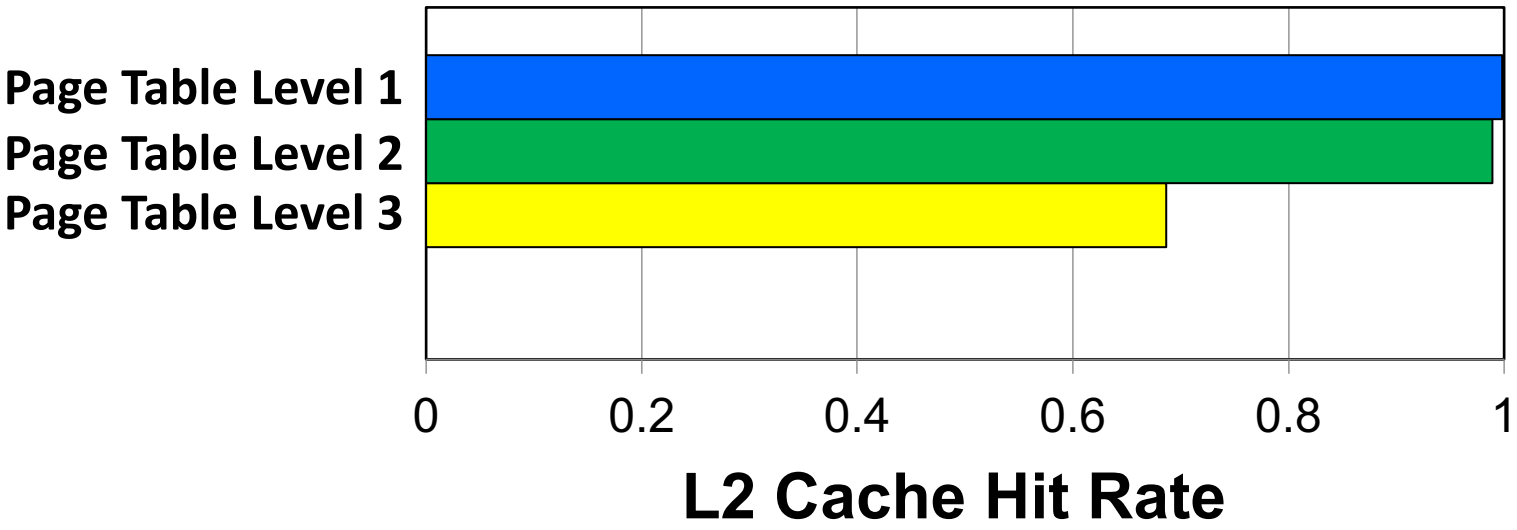
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



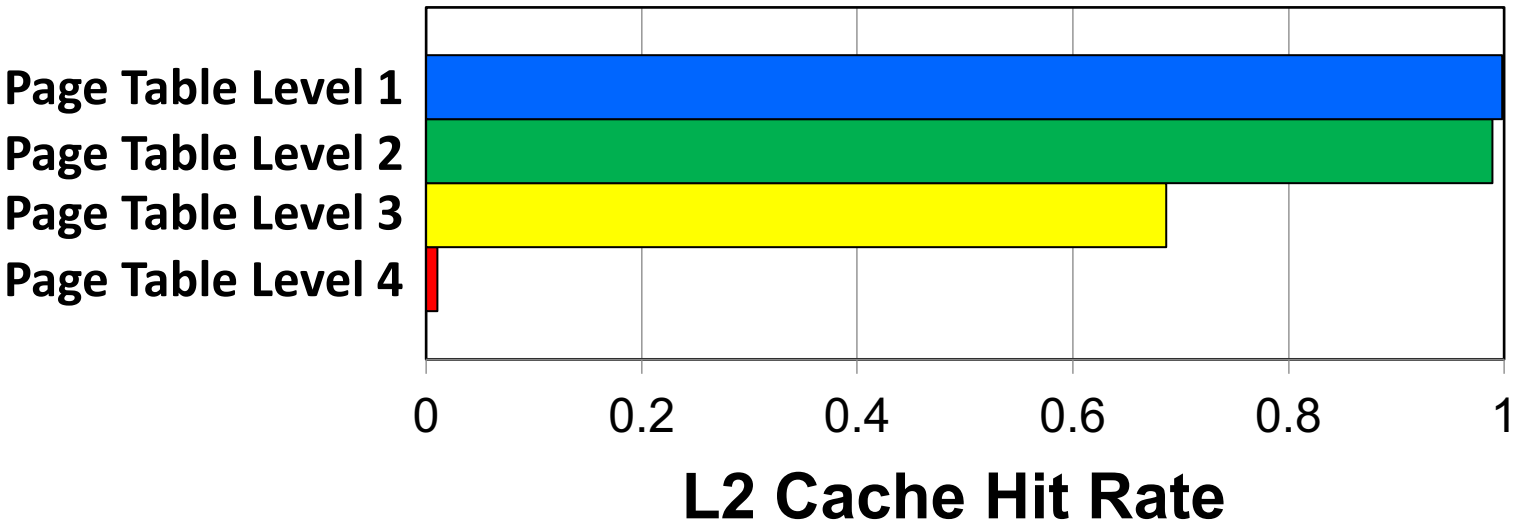
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels

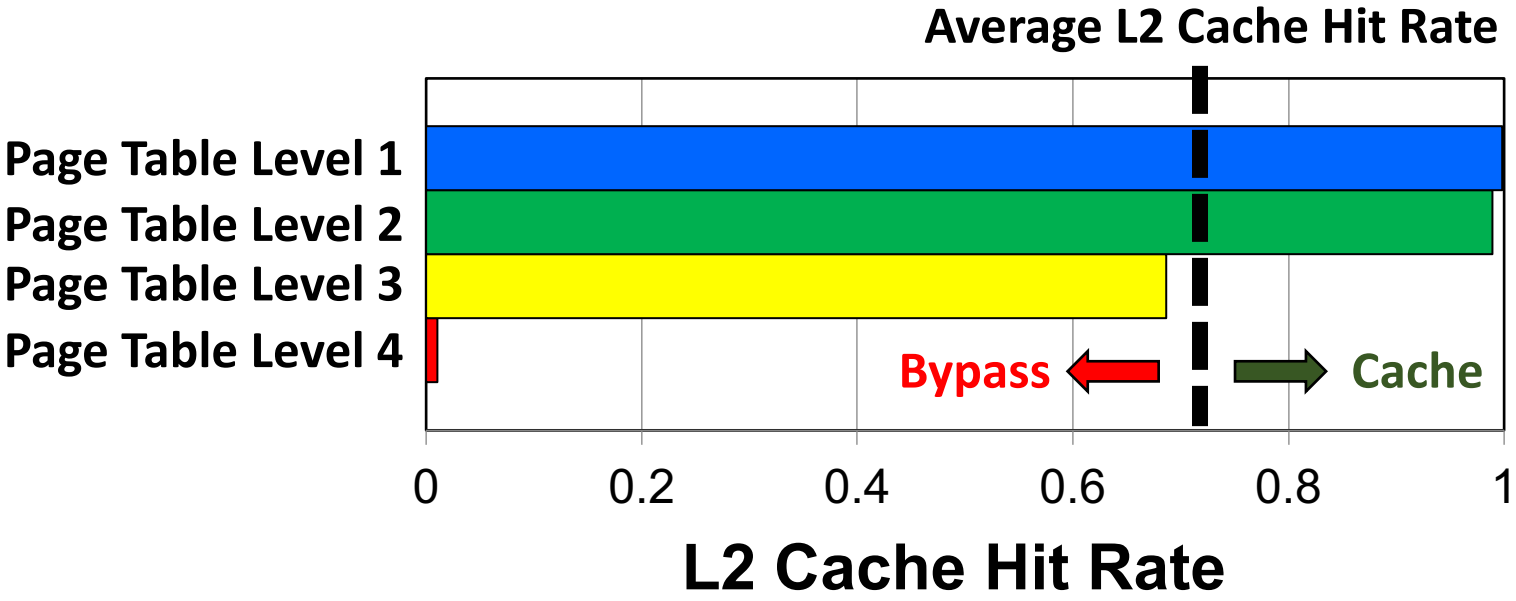


Some address translation data **does not benefit from caching**

Only cache address translation data with high hit rate

B: Translation-aware L2 Bypass

- **Goal:** Cache address translation data with high hit rate



Benefit 1: Better L2 cache utilization for translation data

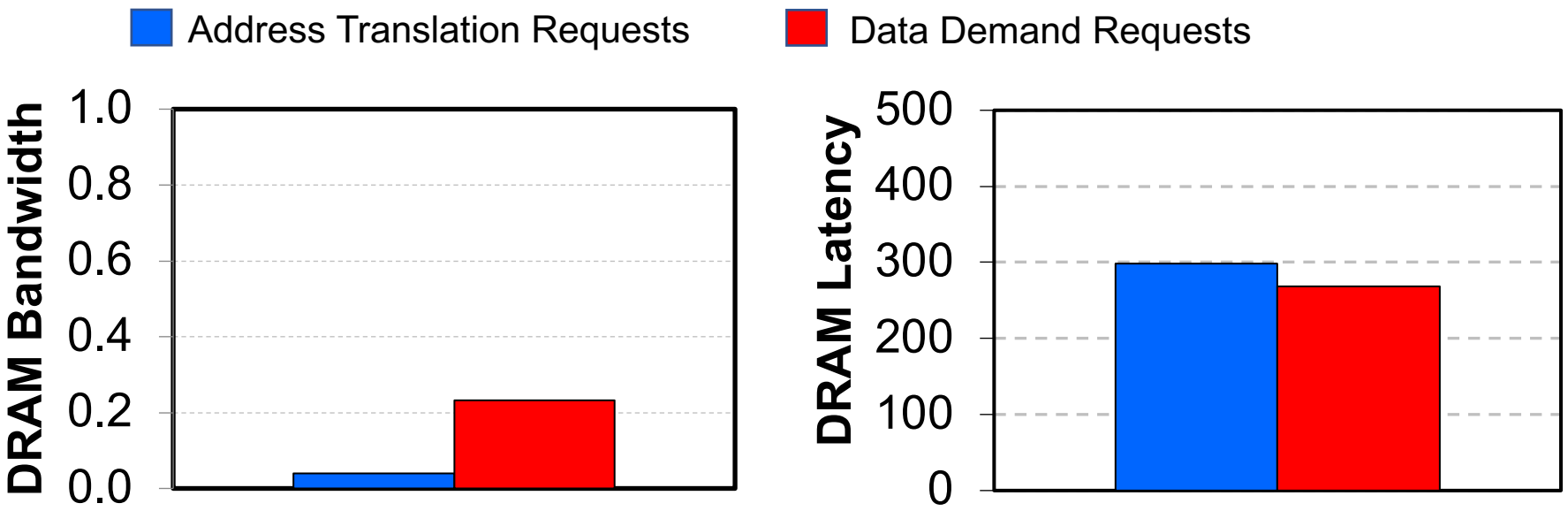
Benefit 2: Bypassed requests → No L2 queuing delay

MASK: A Translation-aware Memory Hierarchy

- **Reduce shared TLB contention**
  - **A. TLB-fill Tokens**
- **Improve L2 cache utilization**
  - **B. Translation-aware L2 Bypass**
- **Lower page walk latency**
  - **C. Address-space-aware Memory Scheduler**

C: Address-space-aware Memory Scheduler

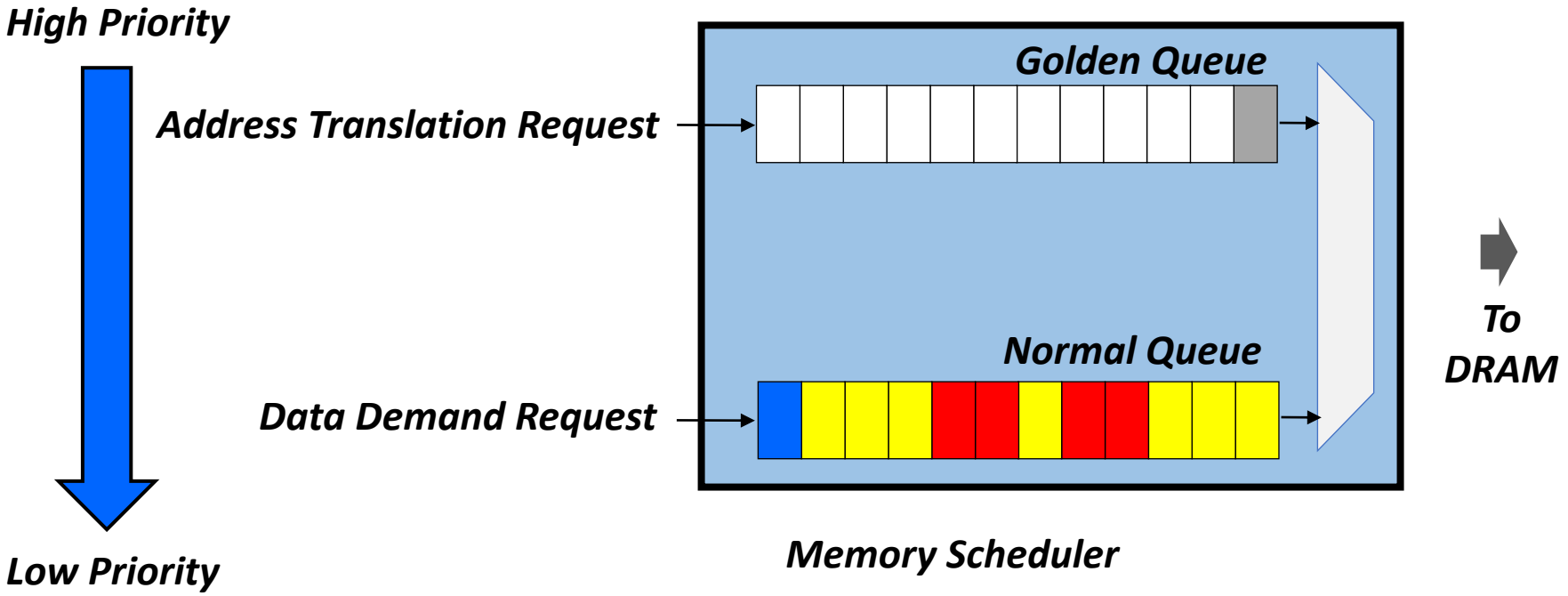
- **Cause:** Address translation requests **are treated similarly** to data demand requests



Idea: **Lower address translation request** latency

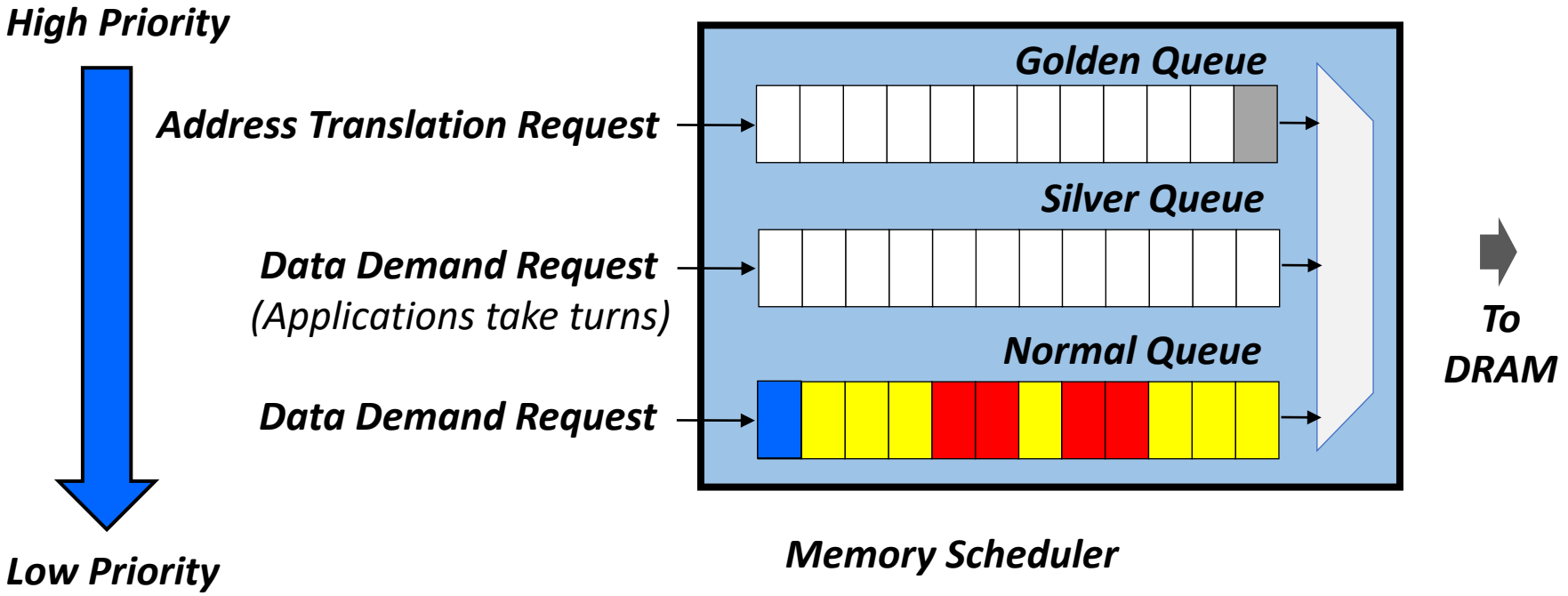
C: Address-space-aware Memory Scheduler

- **Idea 1: Prioritize address translation requests** over data demand requests



C: Address-space-aware Memory Scheduler

- **Idea 1: Prioritize address translation requests** over data demand requests
- **Idea 2: Improve quality-of-service** using the Silver Queue



Each application takes turn injecting into the Silver Queue

Outline

- **Executive summary**
- **Background, Key Challenges and Our Goal**
- **MASK:** A Translation-aware Memory Hierarchy
- **Evaluation**
- **Conclusion**

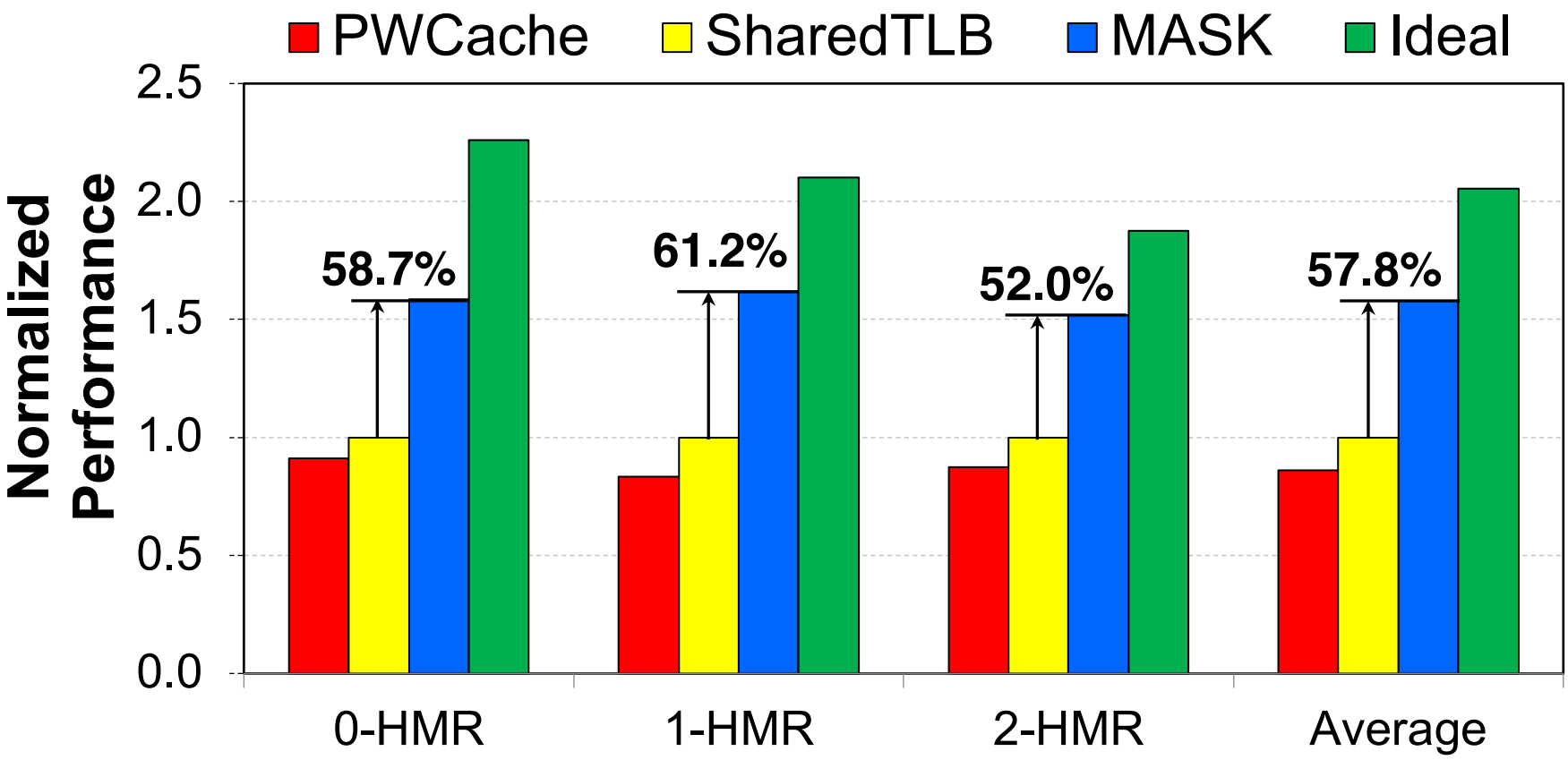
Methodology

- Mosaic simulation platform [MICRO '17]
  - Based on GPGPU-Sim and MAFIA [Jog et al., MEMSYS '15]
  - Models page walks and virtual-to-physical mapping
  - Available at **https://github.com/CMU-SAFARI/Mosaic**
- NVIDIA GTX750 Ti
- Two GPGPU applications execute concurrently
- CUDA-SDK, Rodinia, Parboil, LULESH, SHOC suites
  - 3 workload categories based on TLB miss rate

Comparison Points

- State-of-the-art CPU–GPU memory management [Power et al., HPCA ’14]
  - → **PWCache:** Page Walk Cache GPU MMU design
  - → **SharedTLB:** Shared TLB GPU MMU design
- **Ideal:** Every TLB access is an L1 TLB hit

Performance



MASK outperforms state-of-the-art design for **every workload**

Other Results in the Paper

- MASK reduces unfairness
- Effectiveness of each individual component
  - **All three MASK components are effective**
- Sensitivity analysis over multiple GPU architectures
  - **MASK improves performance on all evaluated architectures,** including CPU–GPU heterogeneous systems
- Sensitivity analysis to different TLB sizes
  - **MASK improves performance on all evaluated sizes**
- Performance improvement over different memory scheduling policies
  - **MASK improves performance over other state-of-the-art memory schedulers**

Conclusion

Problem: Address translation overheads
limit the latency hiding capability of a GPU

High contention at the shared TLB

Low L2 cache utilization

Large performance loss vs. no translation

Key Idea
Prioritize **address translation requests** over **data requests**

MASK: a translation-aware GPU memory hierarchy

A. TLB-fill Tokens **reduces shared TLB contention**
B. Translation-aware L2 Bypass **improves L2 cache utilization**
C. Address-space-aware Memory Scheduler **reduces page walk latency**

MASK **improves system throughput by 57.8%** on average over state-of-the-art address translation mechanisms

MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency

Rachata Ausavarungnirun

Vance Miller Joshua Landgraf Saugata Ghose

Jayneel Gandhi Adwait Jog Christopher J. Rossbach Onur Mutlu

Carnegie Mellon

TEXAS
The University of Texas at Austin

ETH Zürich

vmware

SAFARI