High-performance Virtual Memory Design for Modern Architectures

Rachata Ausavarungnirun

https://rausavar.github.io/

Architecture Research Group

TGGS, KMUTNB

Modern-day Cloud Cluster

- Very heterogeneous
  - Using GPUs to accelerate applications is very common
- Various workload types
  - Each with different demands
  - No one size fits all for the page size
  - Demand special accelerators

Modern-day Cloud Cluster

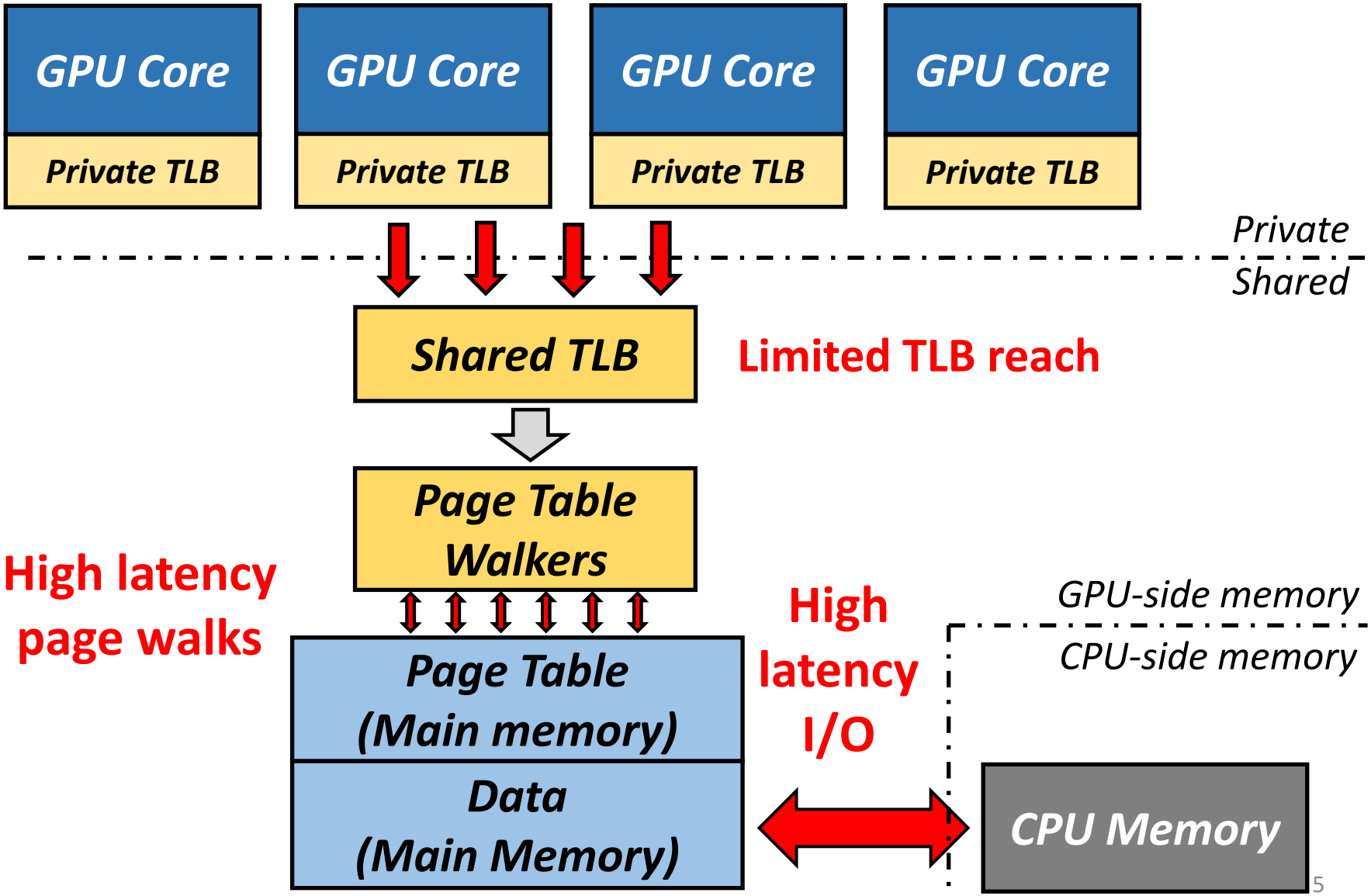
- Very heterogeneous
  - Using GPUs to accelerate applications is very common
- Various workload types
  - Each with different demands
  - No one size fits all for the page size
  - Demand special accelerators

GPU in Modern Systems

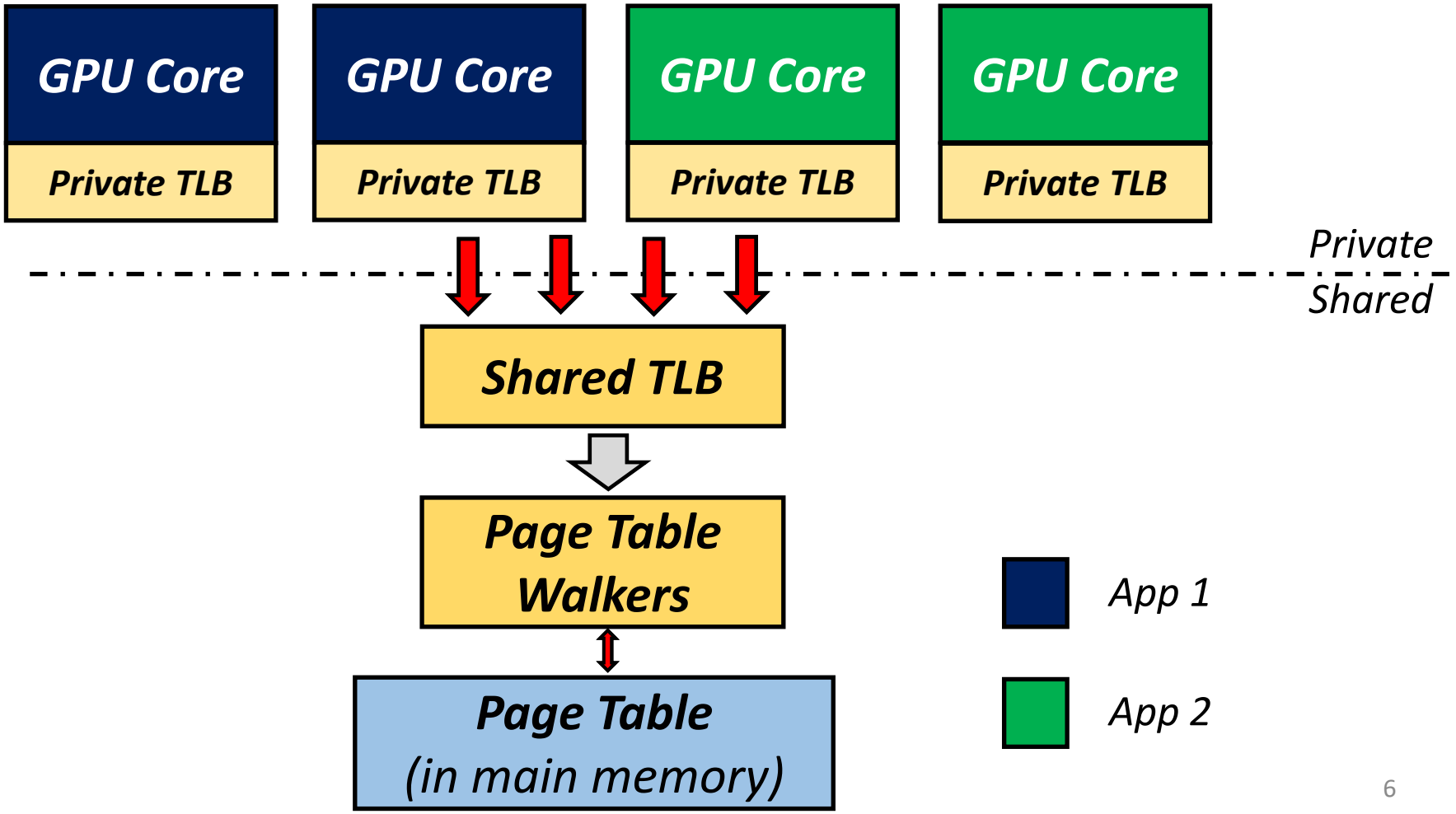
GPUs gain their popularity through their **massive parallelism**

https://developer.nvidia.com/cuda-toolkit https://blogs.nvidia.com/news/nvidia-gpu-cloud-now-available-to-hundreds-of-thousands-of-ai-researchers-using-nvidia-desktop-gpus

Virtual Memory on GPUs

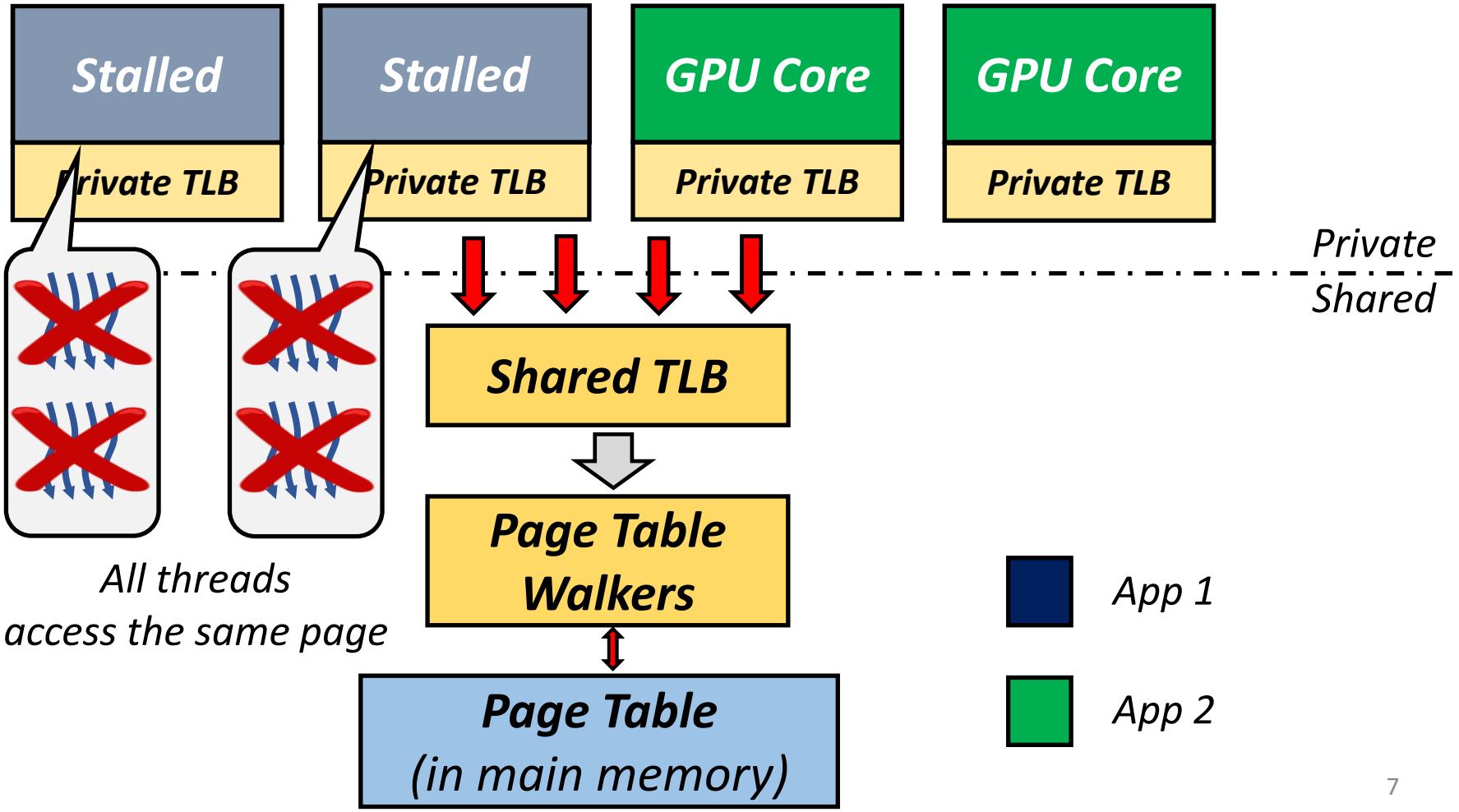


Sharing Makes the Problem Worse

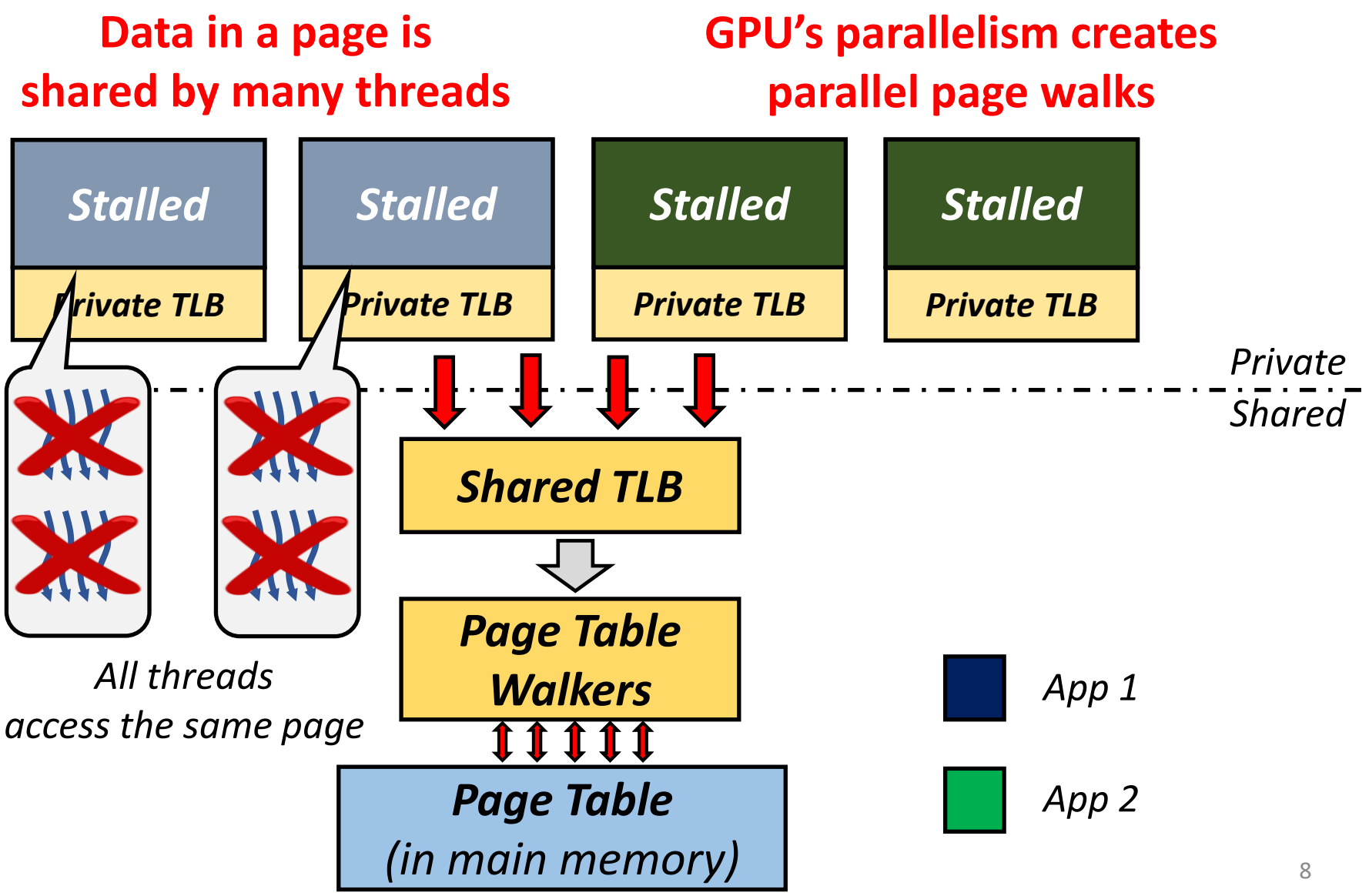


A TLB Miss Stalls Multiple Warps

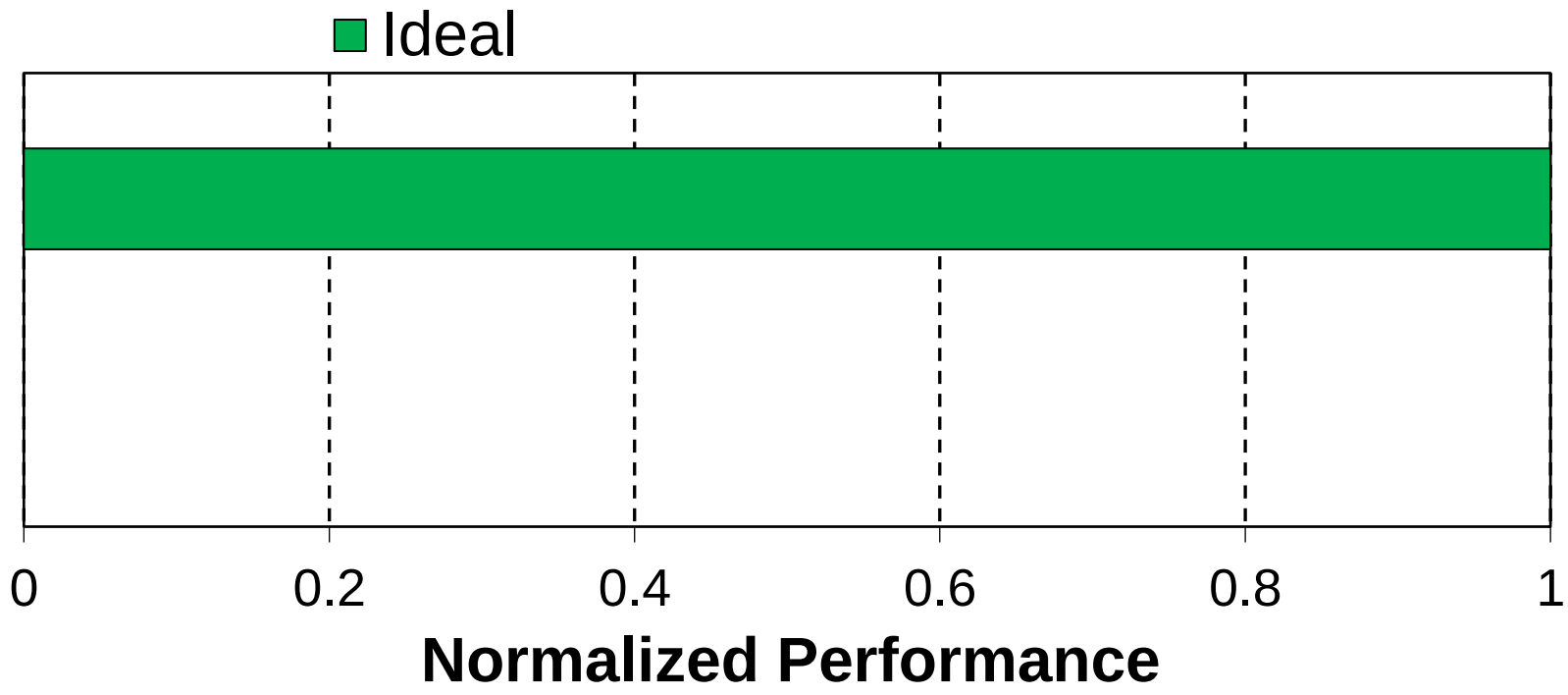
Data in a page is shared by many threads



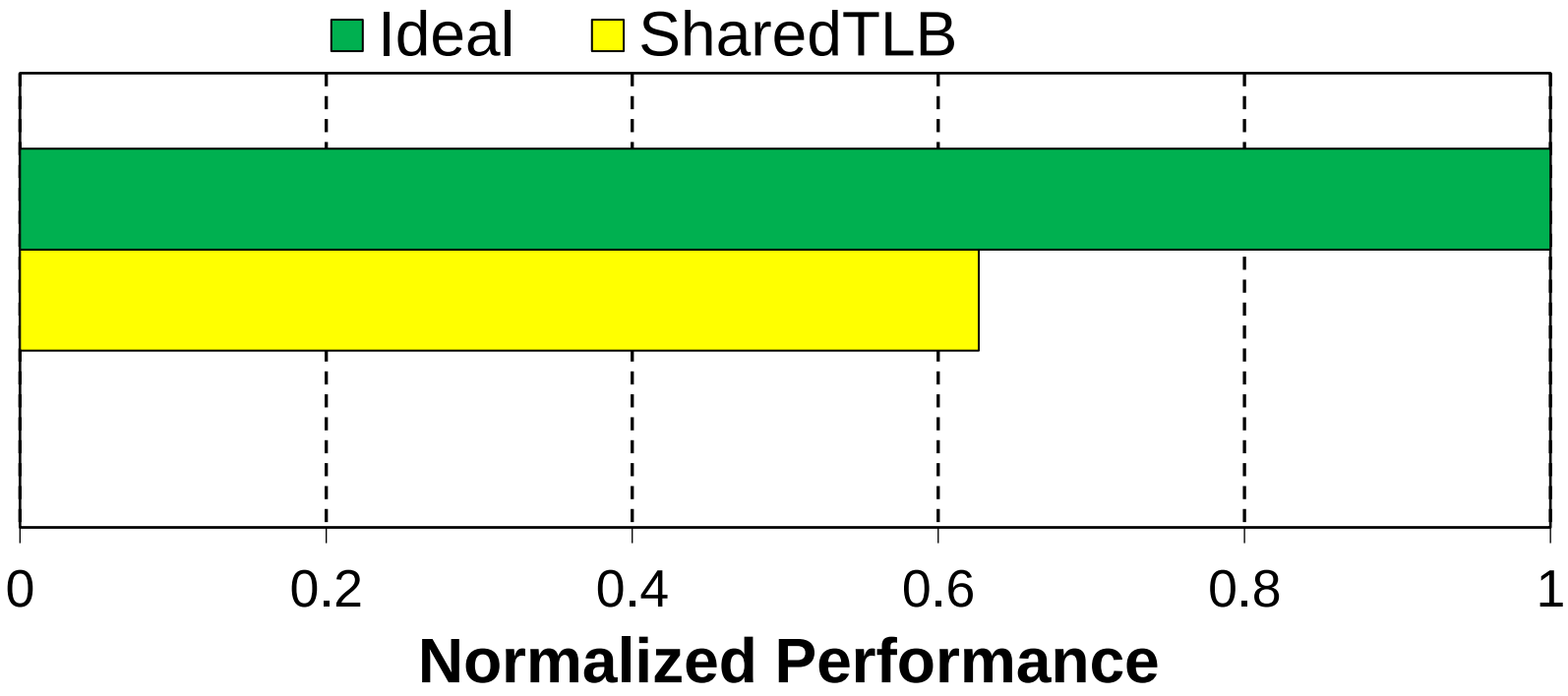
Multiple Page Walks Happen Together



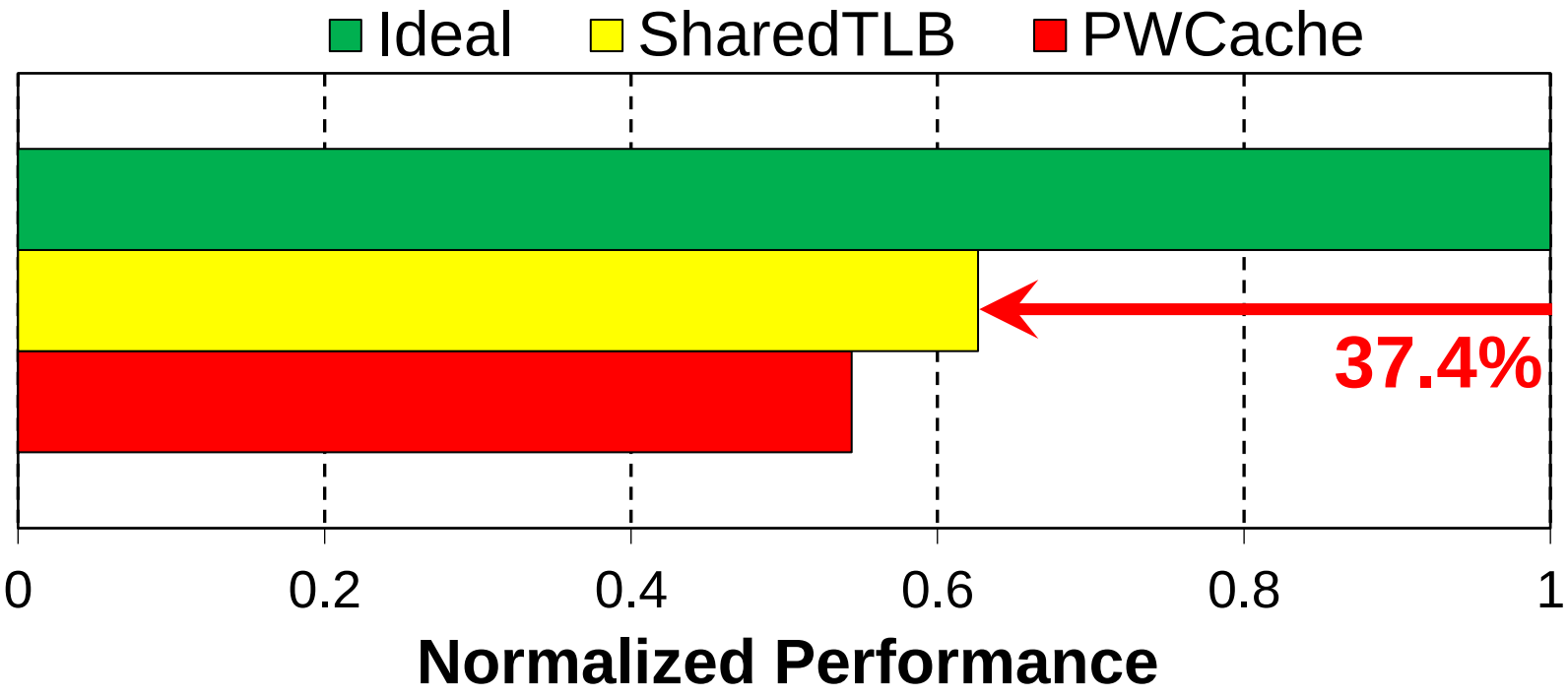
Effect of Translation on Performance



Effect of Translation on Performance



Effect of Translation on Performance



Key Problems: TLB Reach and High PW Latency

Redesign GPU Translation

- **Goals:**
  1. Higher TLB Reach
  2. Low page walk latency

Redesign GPU Translation

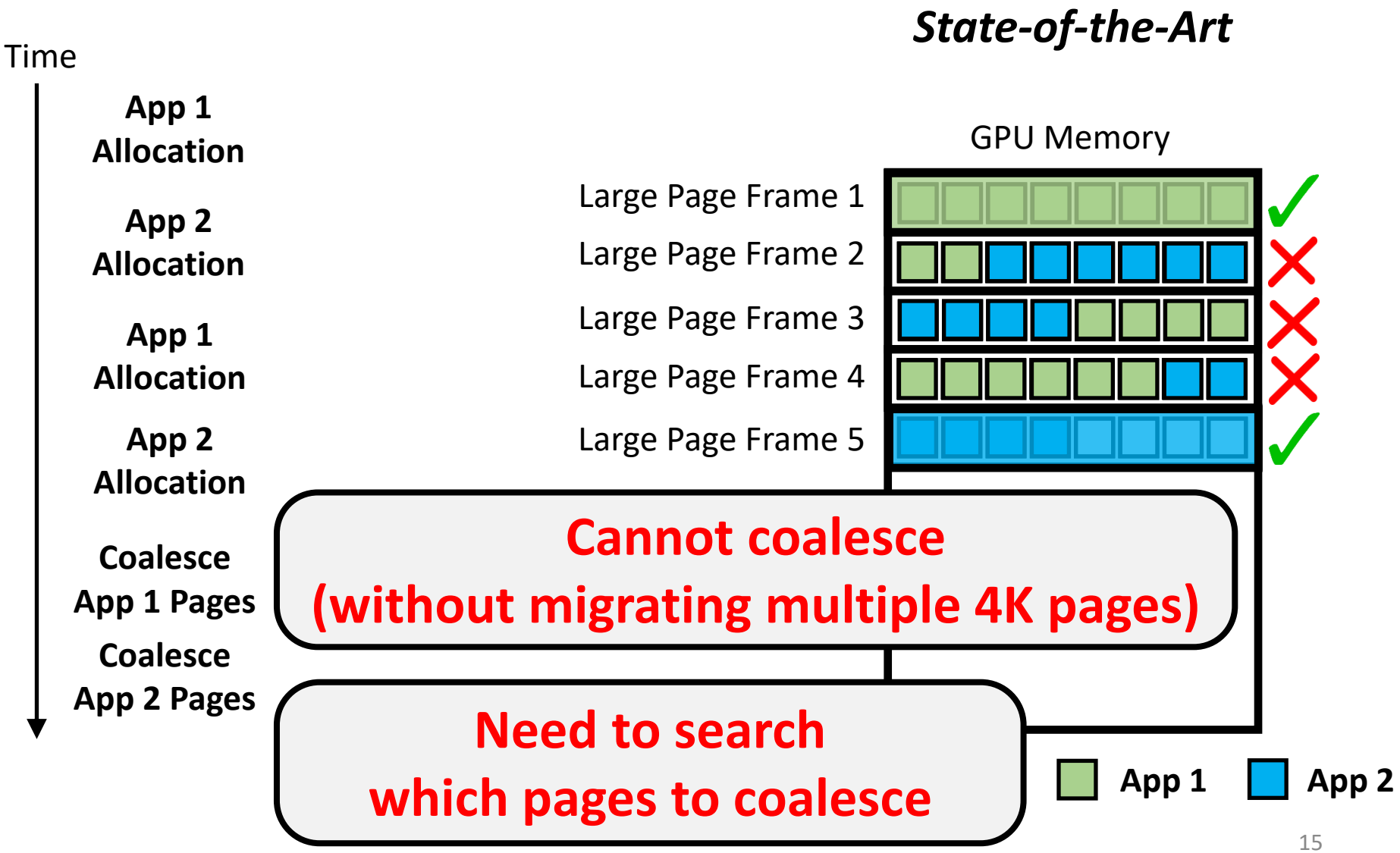
- **Goals:**
  1. Higher TLB Reach
  2. Low page walk latency

Trade-Off with Page Size

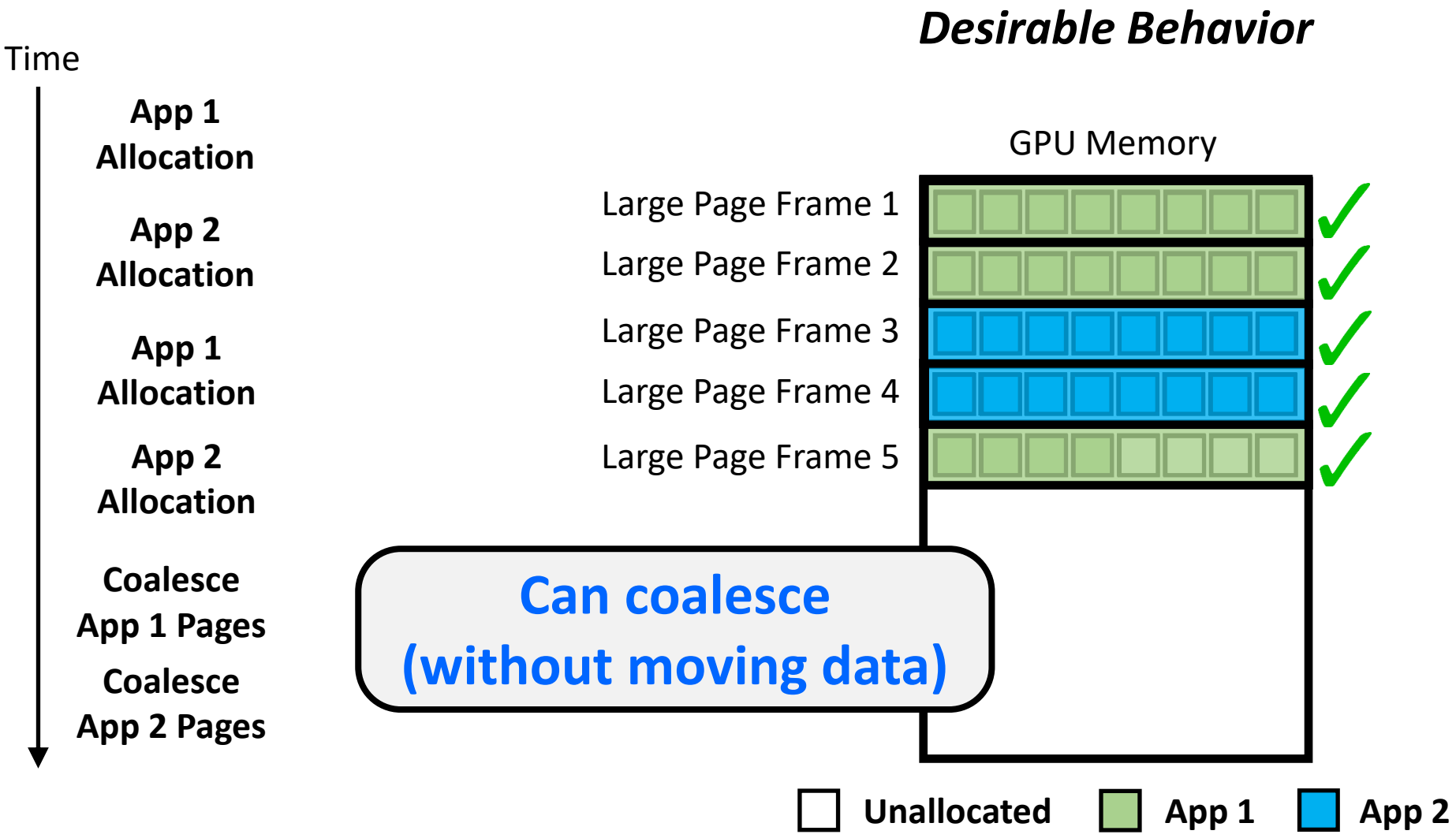
- **Larger pages:**
  - **Better TLB reach**
  - **High demand paging latency**
- **Smaller pages:**
  - **Lower demand paging latency**
  - **Limited TLB reach**

Can we get the best of both page sizes?

Challenges with Multiple Page Sizes



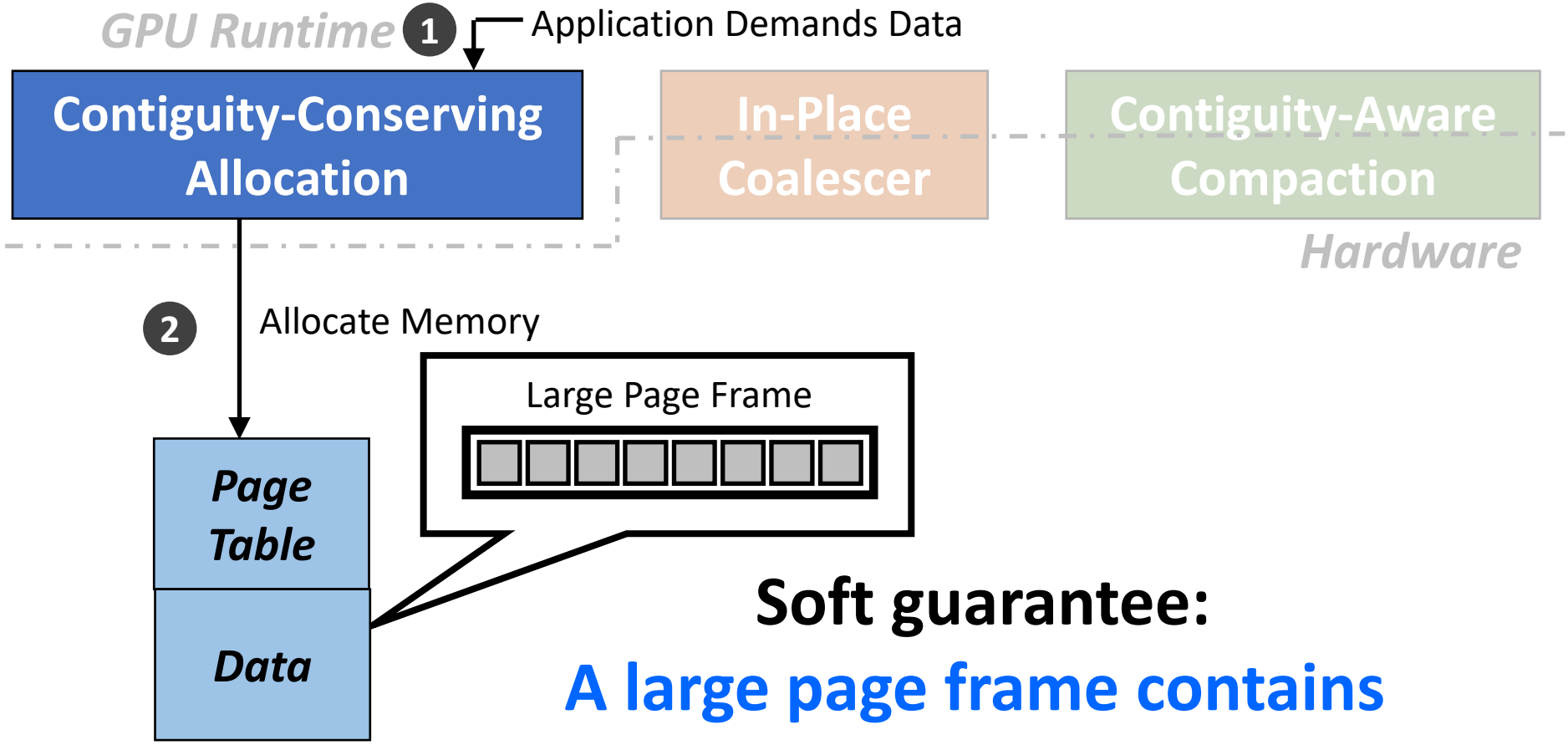
Desirable Allocation



Mosaic


GPU Runtime
Contiguity-Conserving Allocation
In-Place Coalescer
Contiguity-Aware Compaction
Hardware

Mosaic: Data Allocation

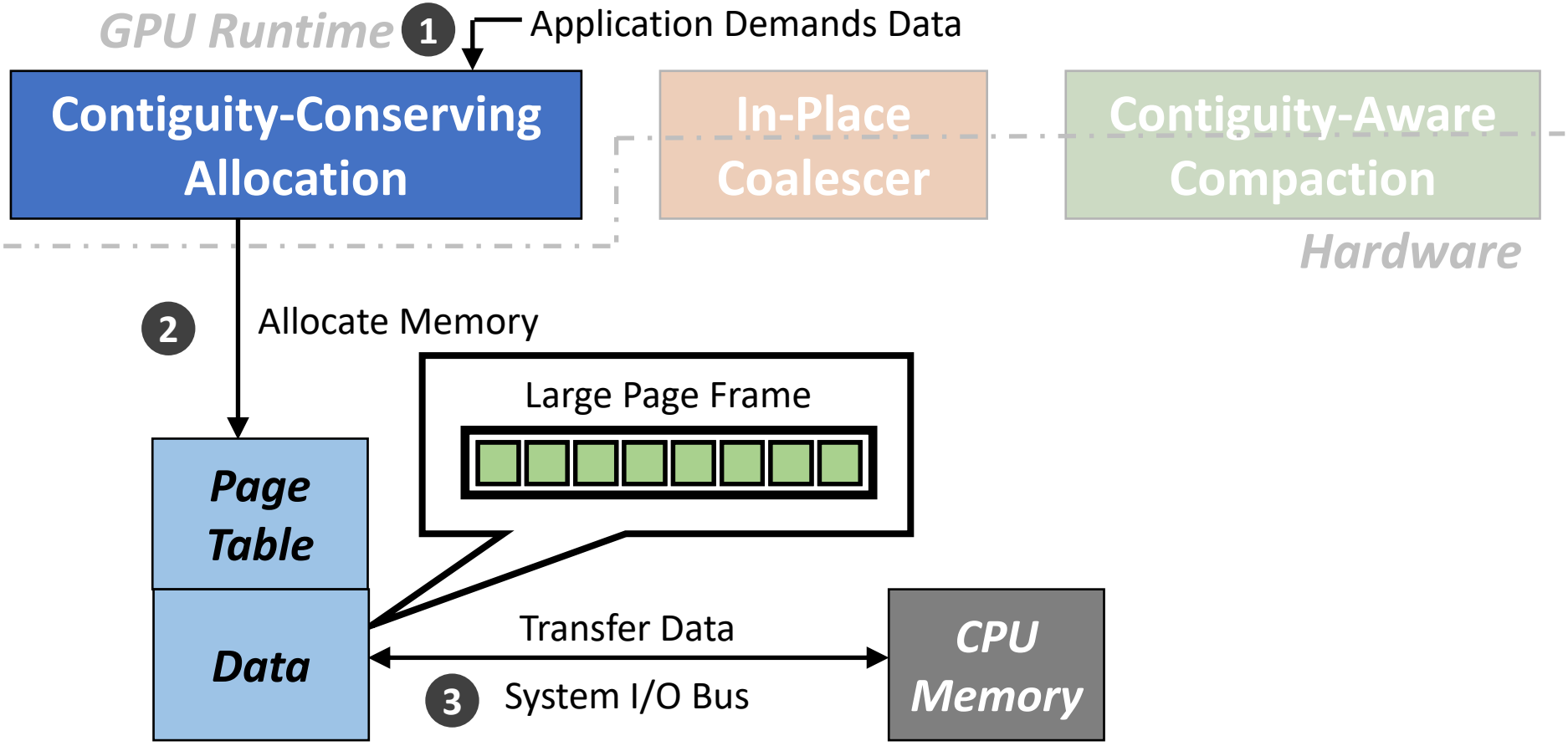


Soft guarantee:

A large page frame contains pages from only a single address space

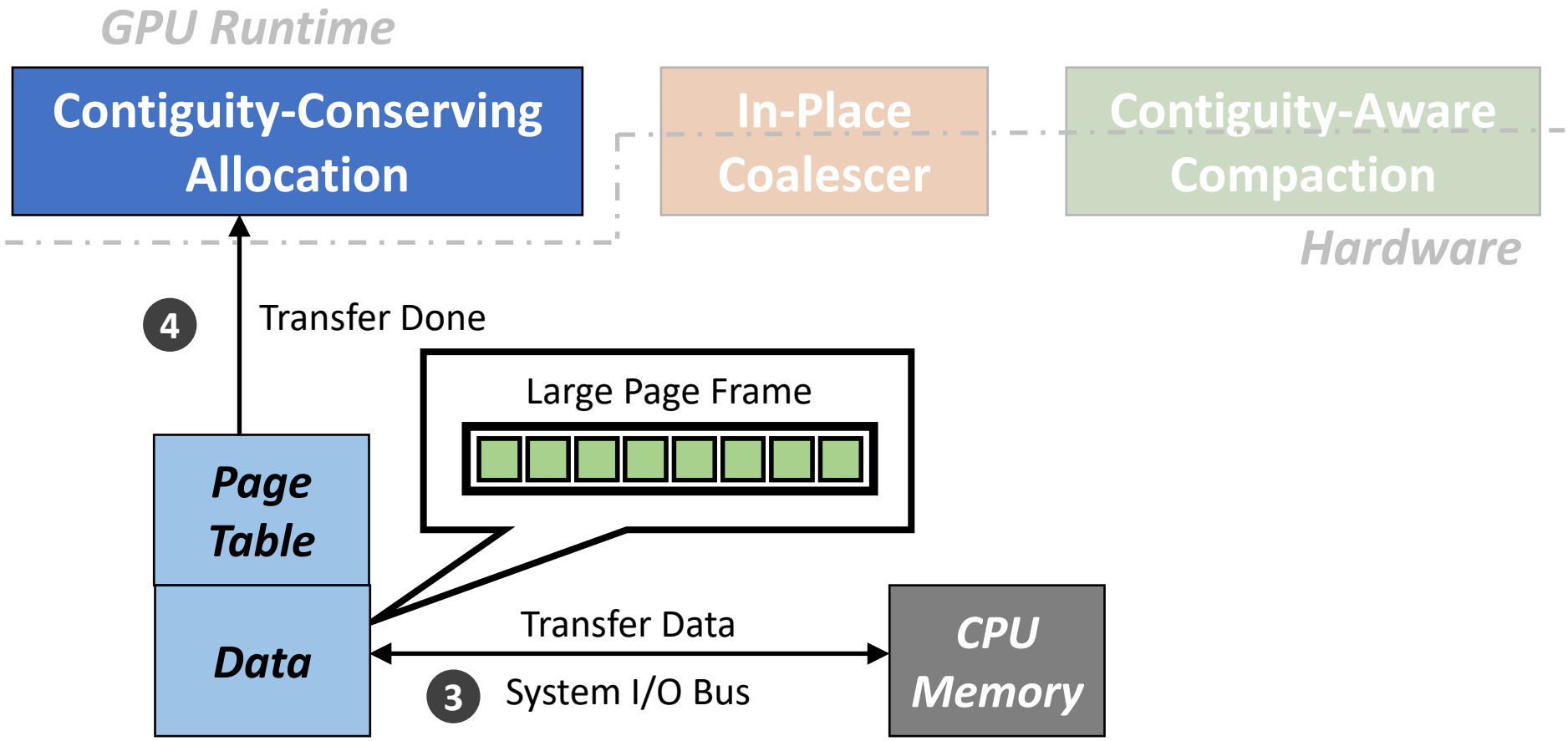
Conserves contiguity within the large page frame

Mosaic: Data Allocation

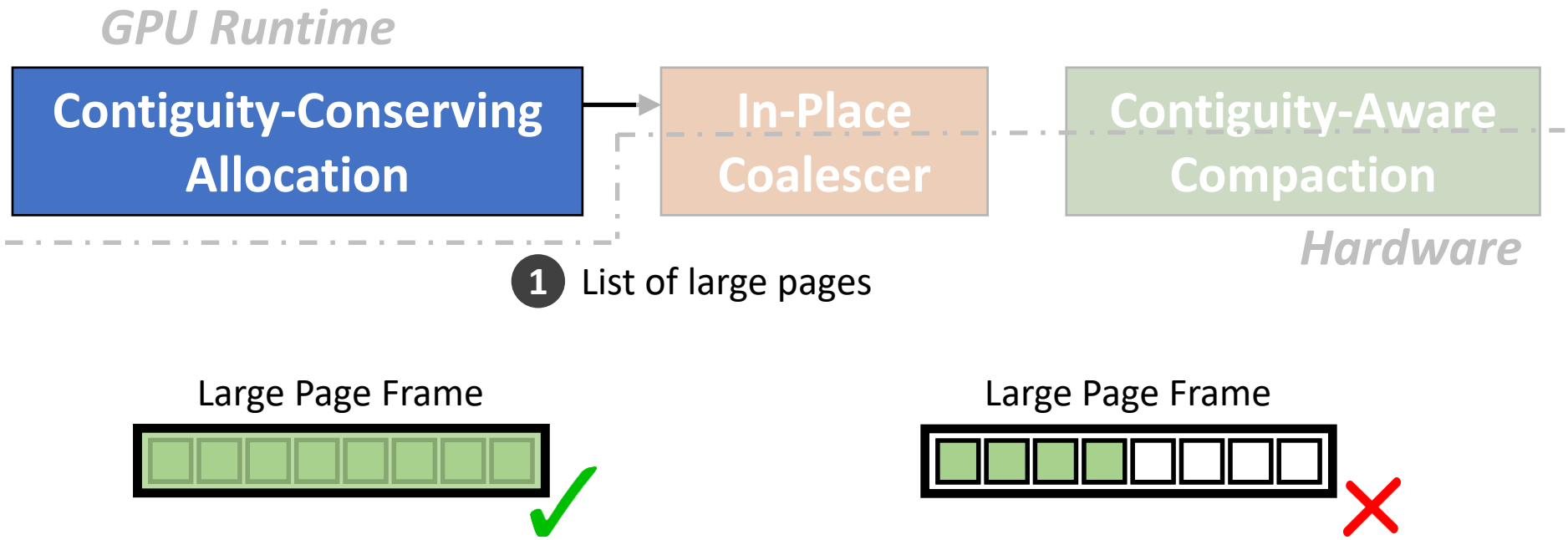


- **Data transfer is done at a small page granularity**
  - A page that is transferred is immediately ready to use

Mosaic: Data Allocation

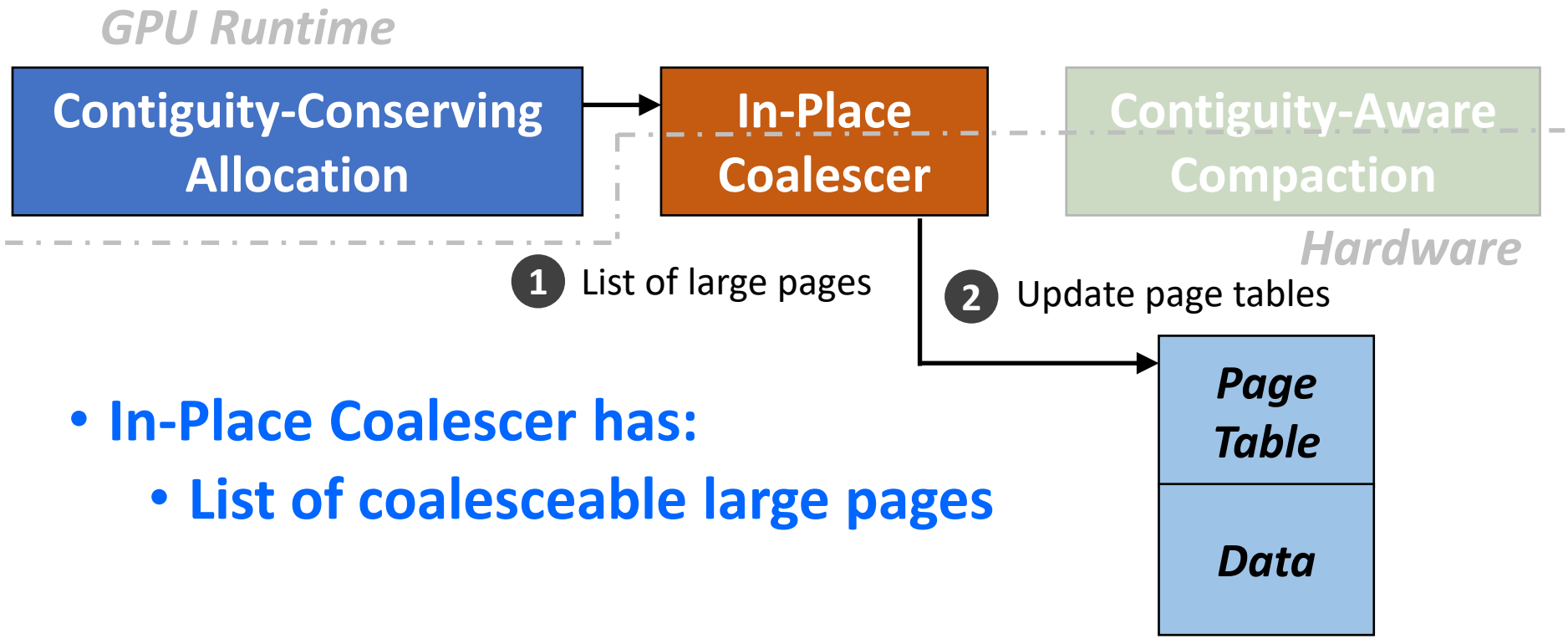


Mosaic: Coalescing



- **Fully-allocated large page frame → Coalesceable**
- **Allocator sends the list of coalesceable pages to the In-Place Coalescer**

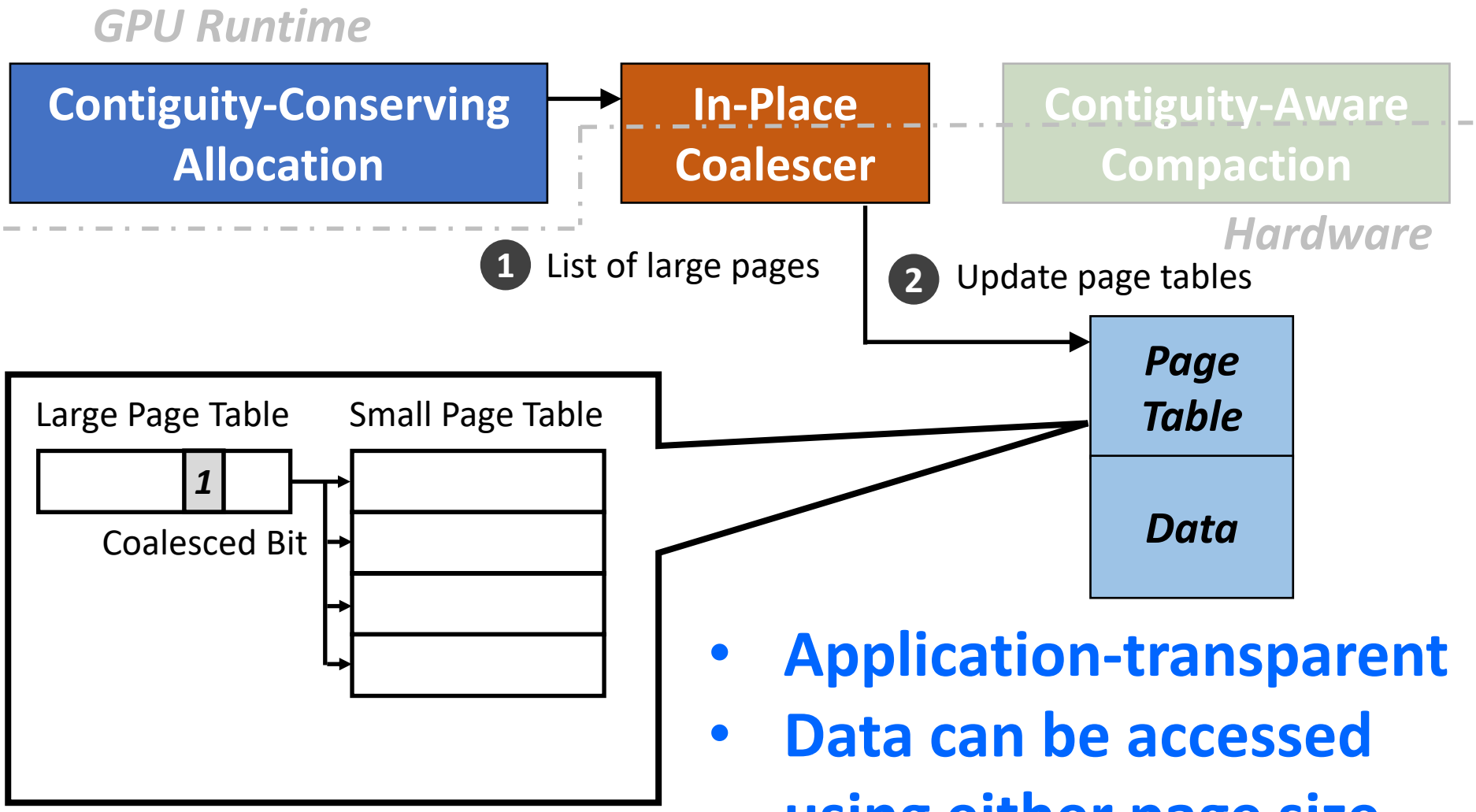
Mosaic: Coalescing



- **In-Place Coalescer has:**
  - **List of coalesceable large pages**

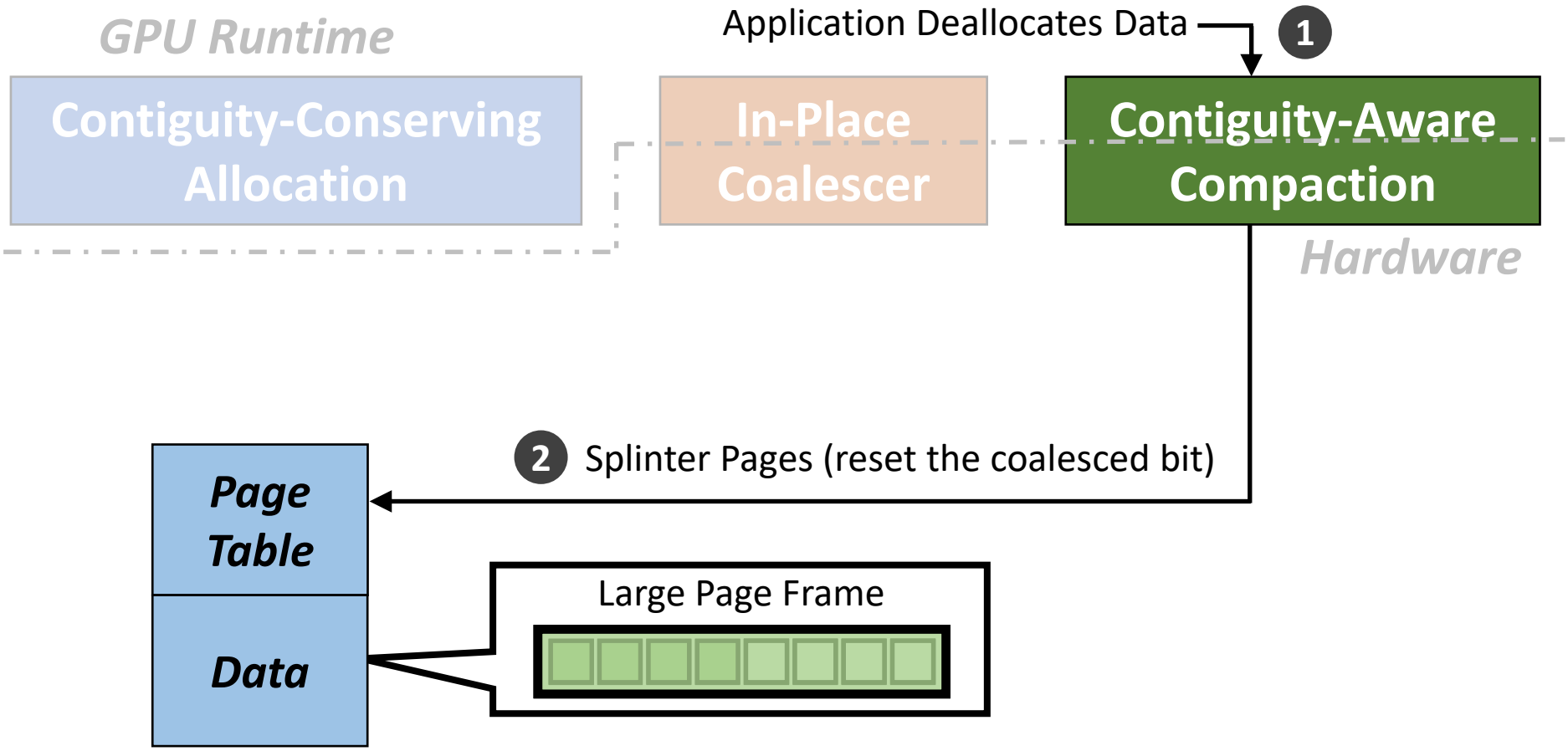
- **Key Task:** Perform coalescing without moving data
  - Simply need to update the page tables

Mosaic: Coalescing



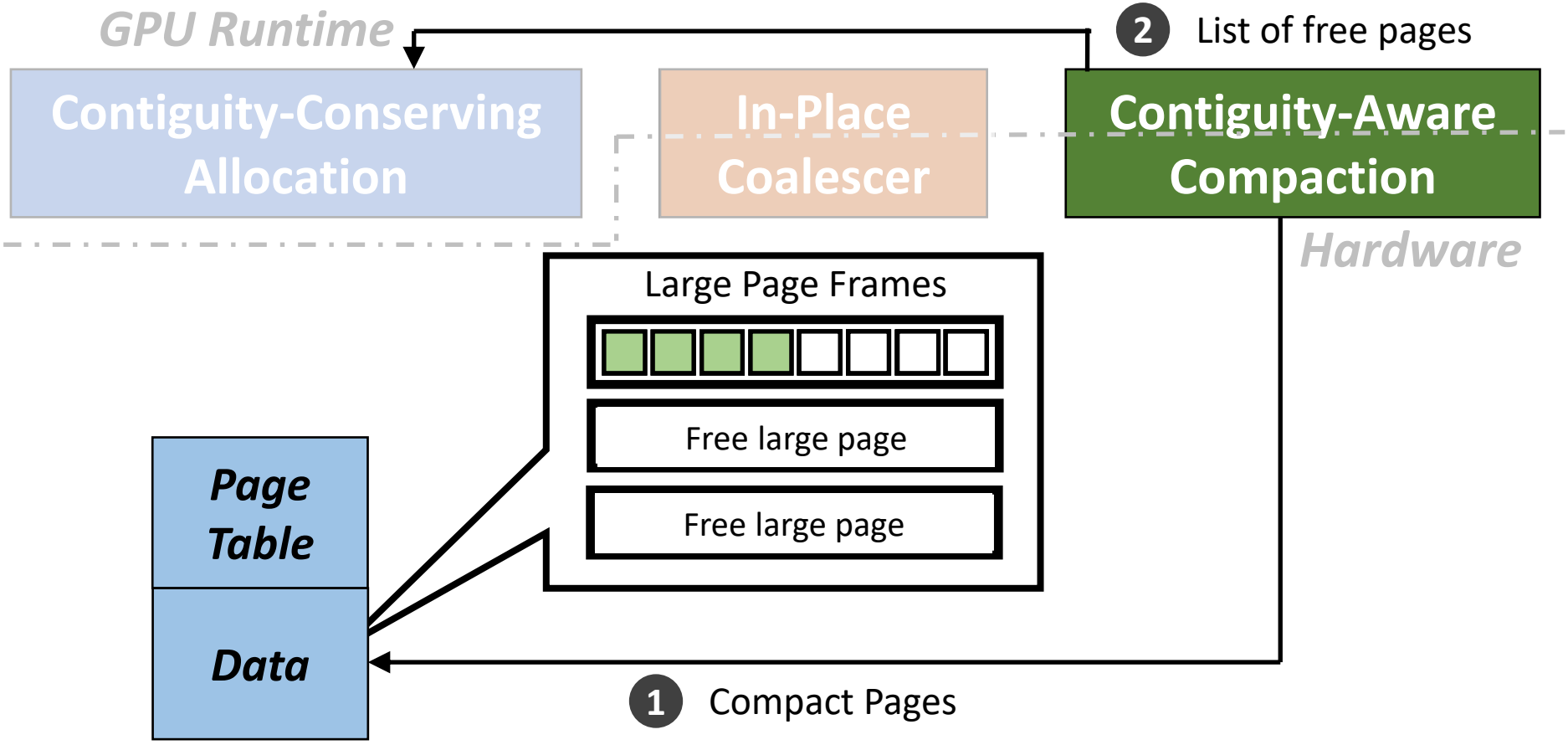
- **Application-transparent**
- **Data can be accessed using either page size**
- **No TLB flush**

Mosaic: Data Deallocation



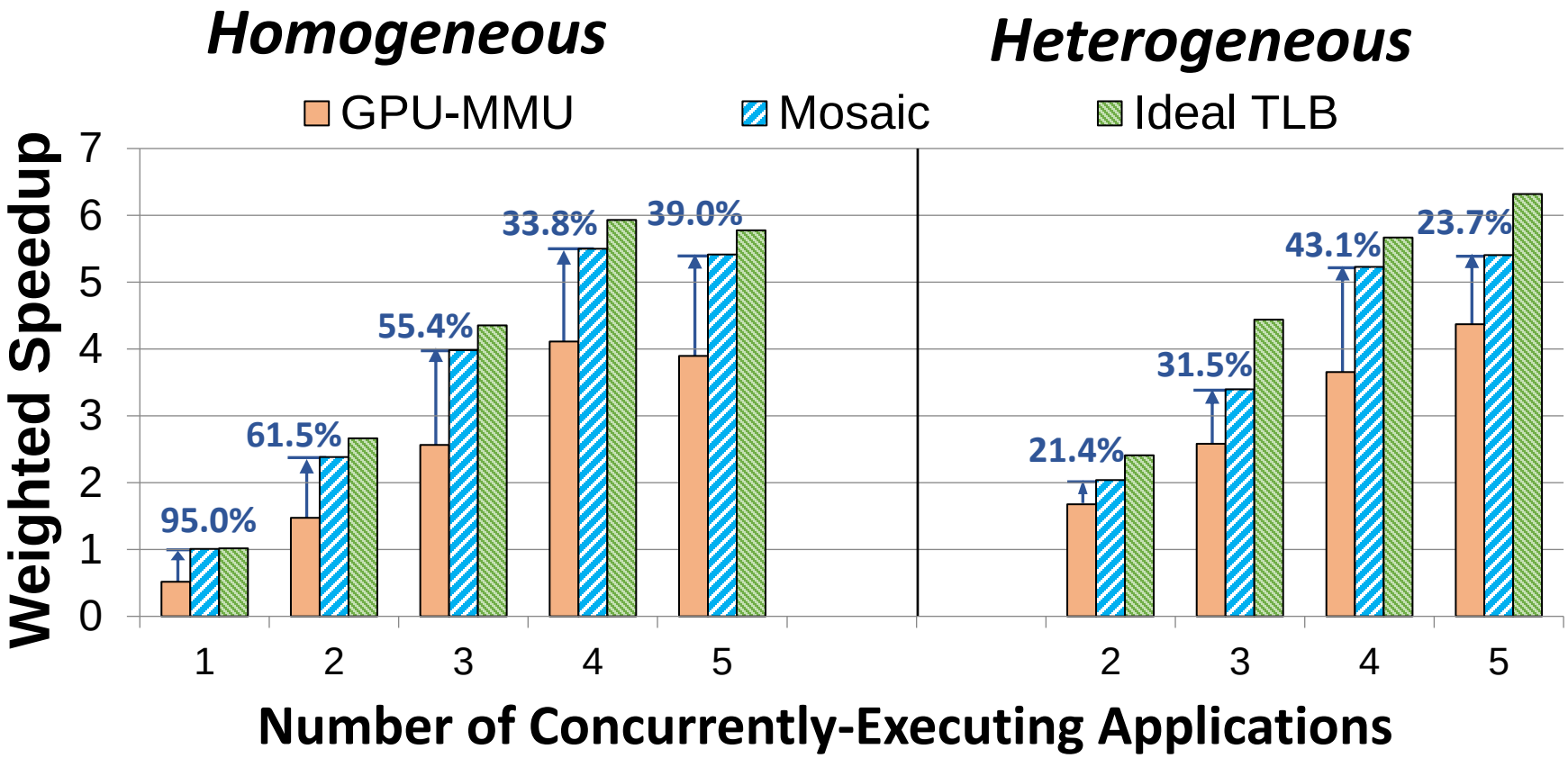
- **Splinter only frames with deallocated pages**

Mosaic: Compaction



- **Compaction decreases memory bloat**
  - Happens only when memory is highly fragmented

Performance



Mosaic consistently improves performance across a wide variety of workloads

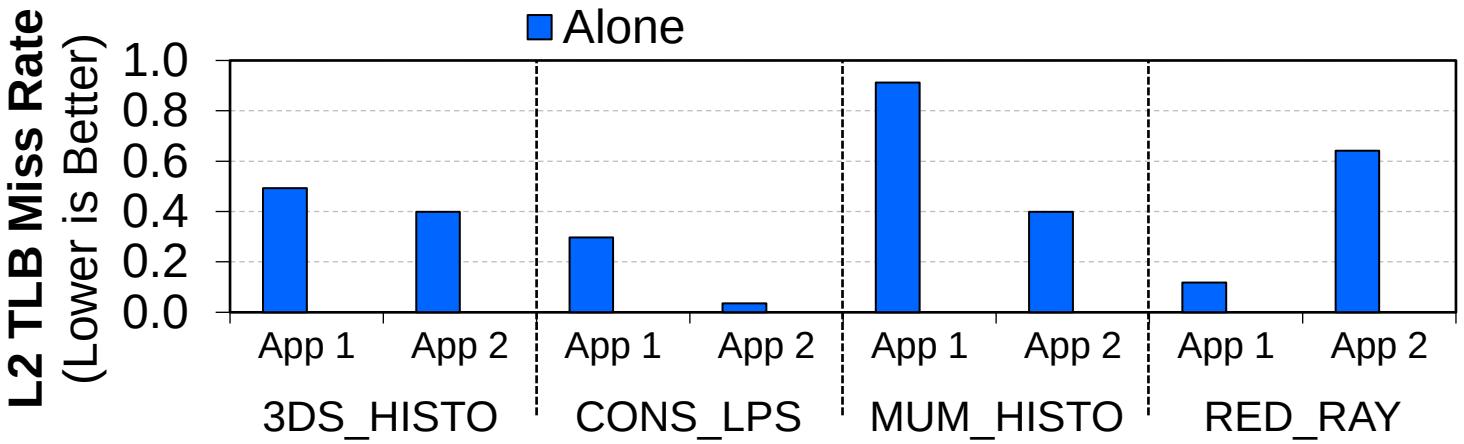
Mosaic performs within 10% of the ideal TLB

Redesign GPU Translation

- **Goals:**
  1. Higher TLB Reach
  2. Low page walk latency

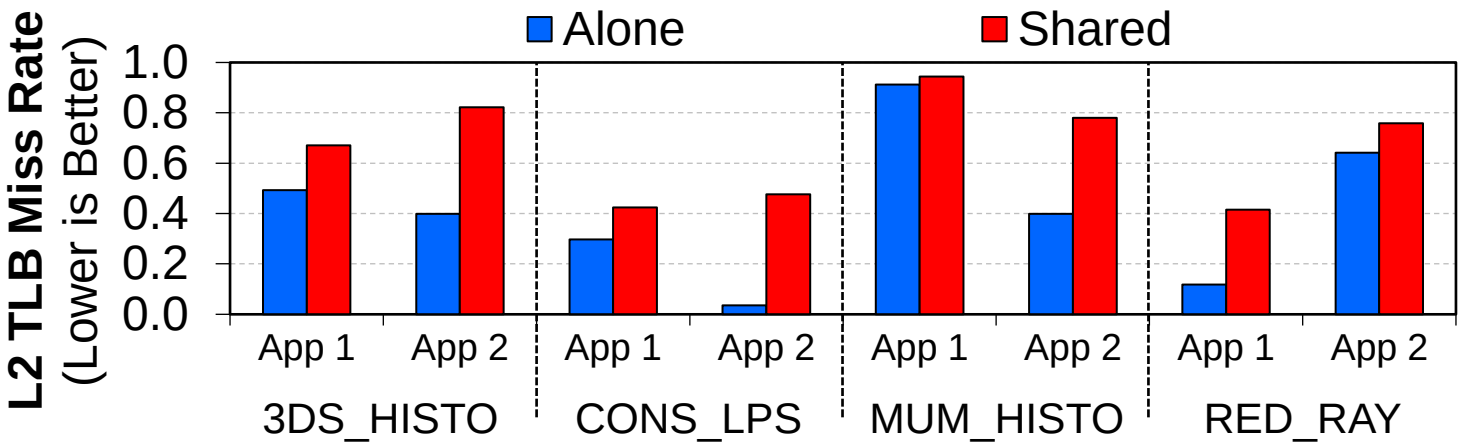
Problem 1: Contention at the Shared TLB

- 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**

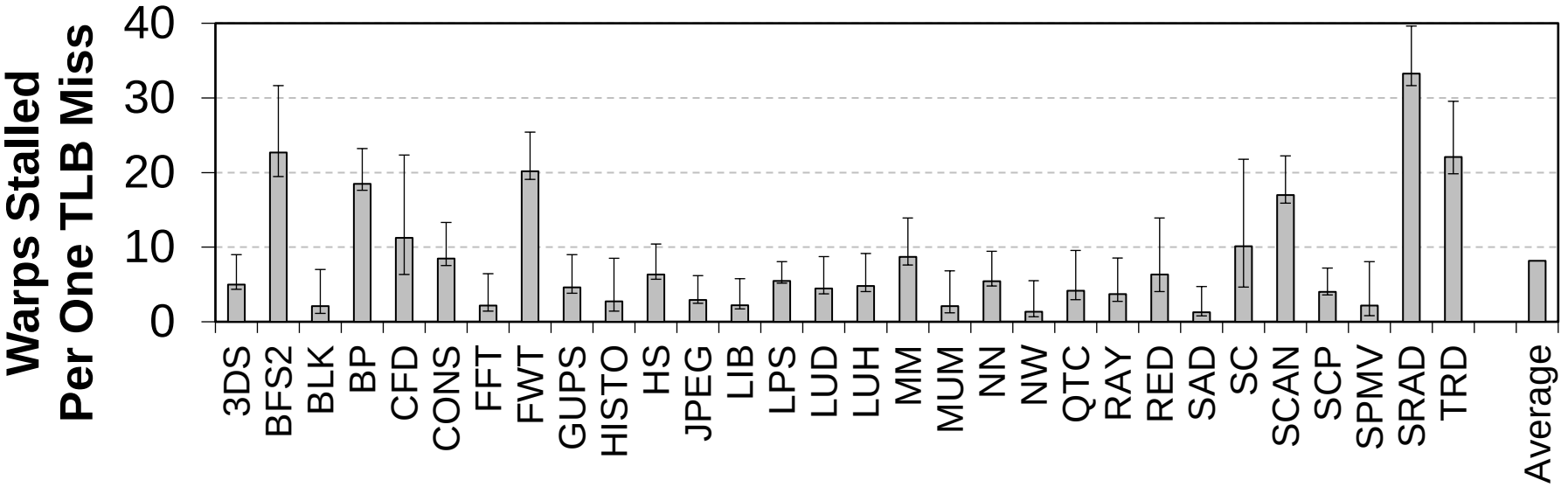
- **Thrashing Source 1:** Parallel page walks
  → **Different address translation data evicts each other**

- **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

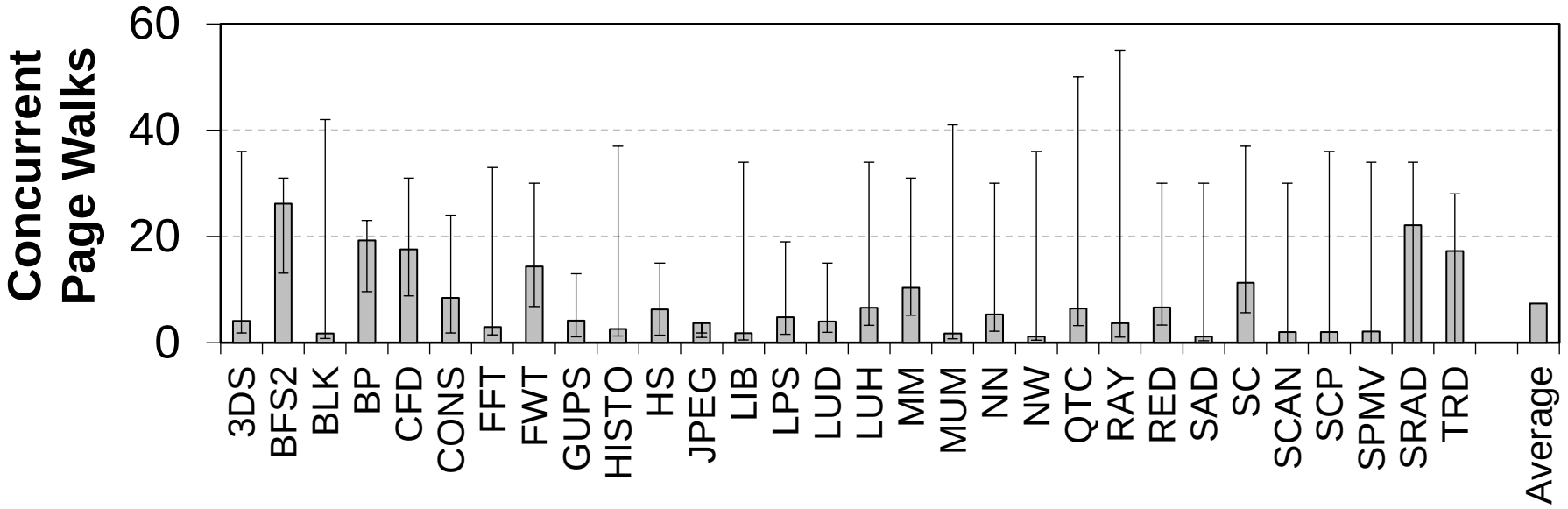
- Multiple warps share data from a single page



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**

MASK Design Goals

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

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**

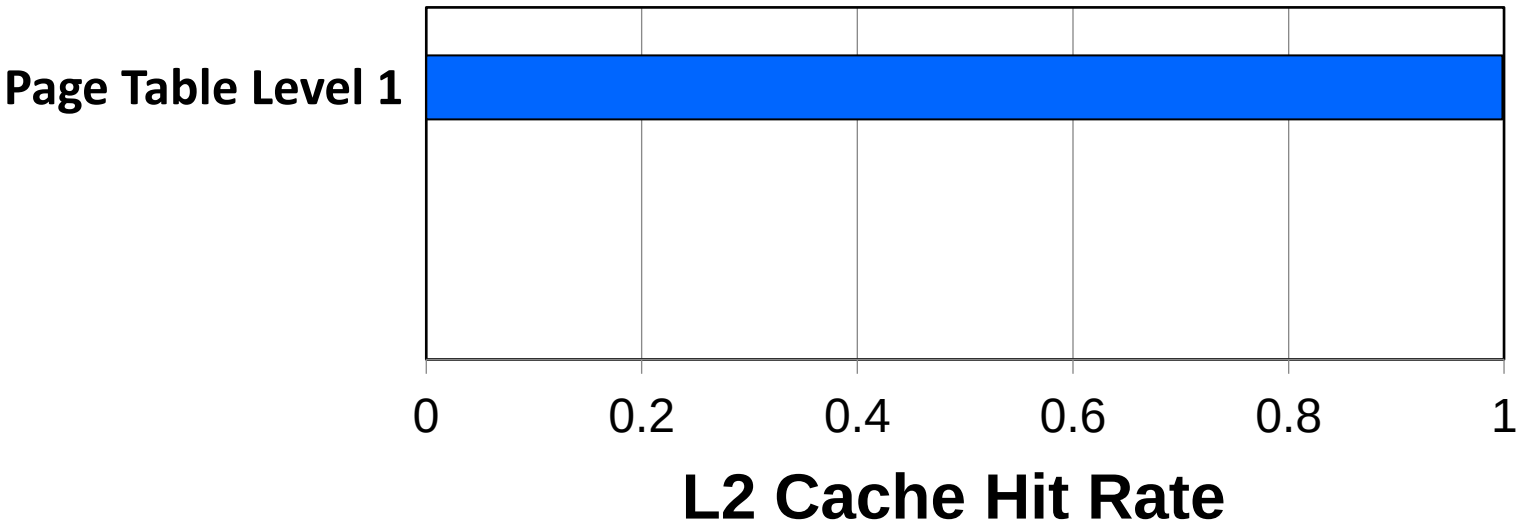
- Number of tokens changes based on TLB miss rate
  - → Updated every epoch

- Tokens are assigned based on warp ID

Benefit: Limits contention at the shared TLB

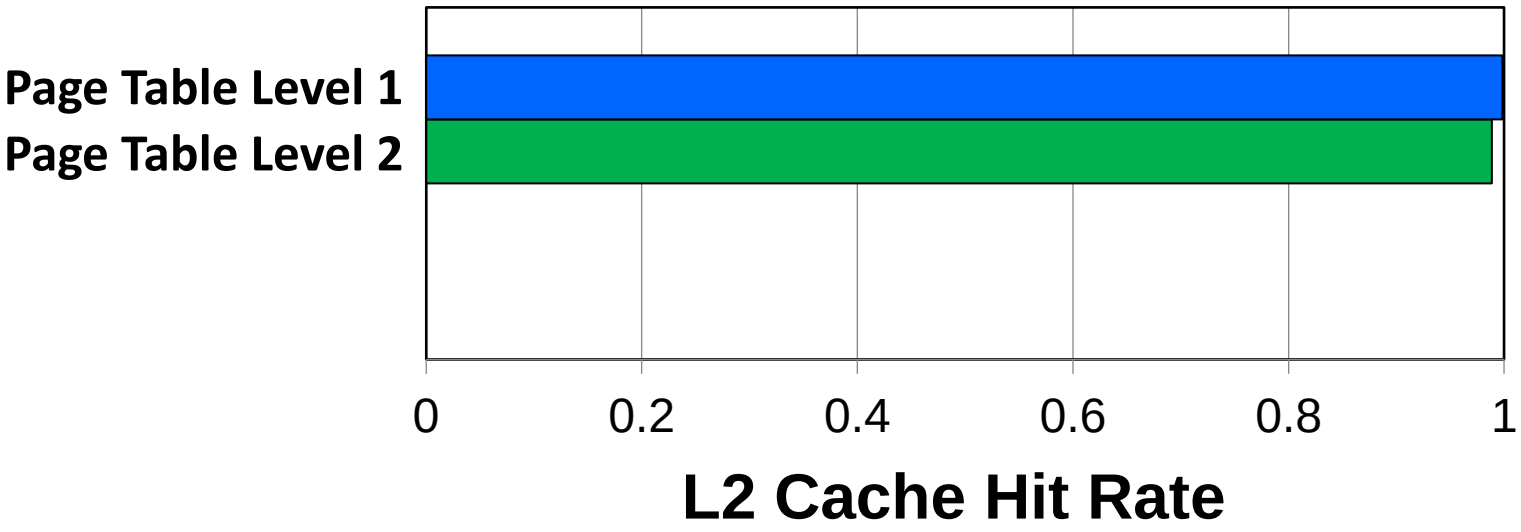
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



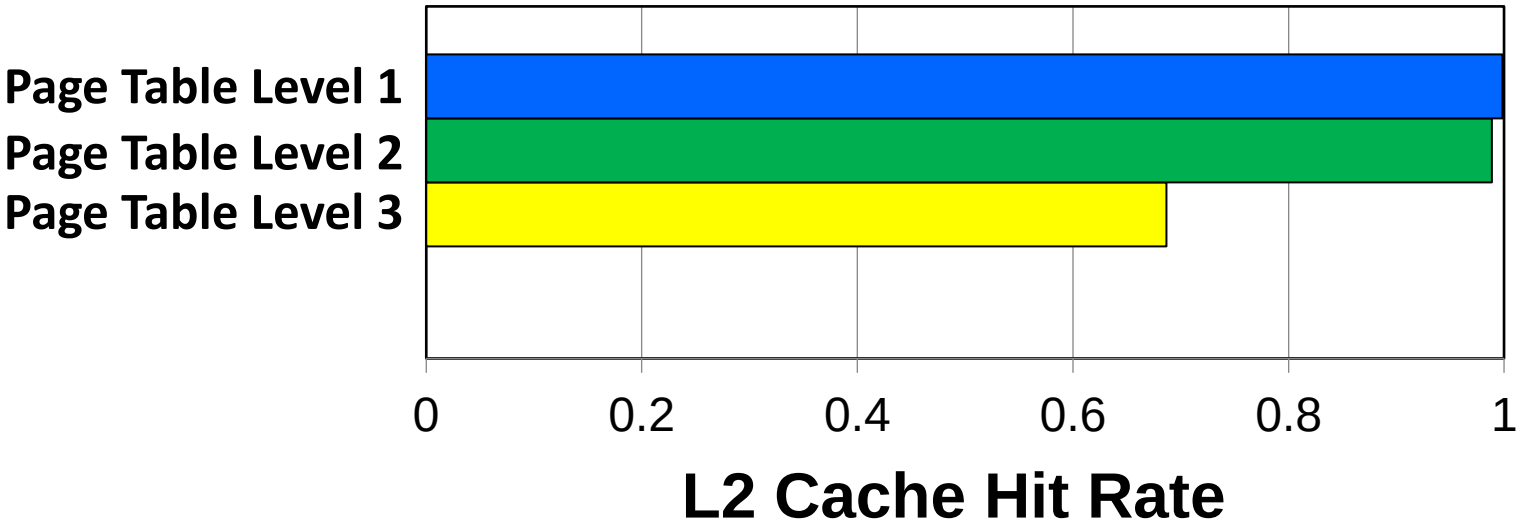
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



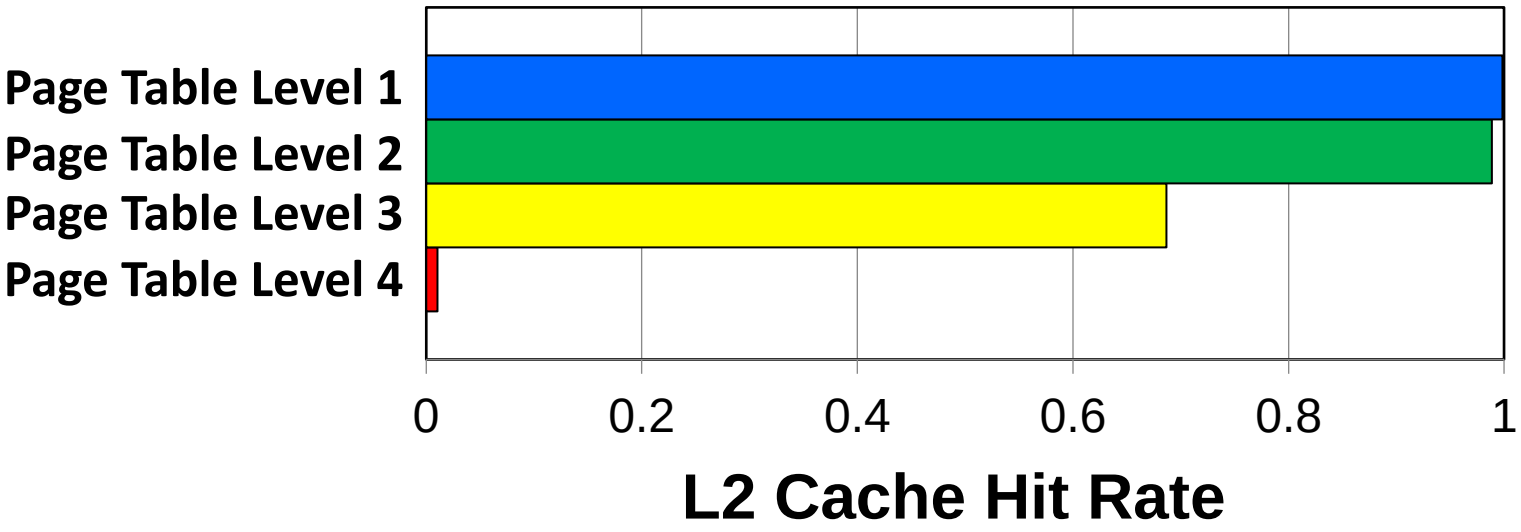
B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels



B: Translation-aware L2 Bypass

- L2 hit rate decreases for deep page walk levels

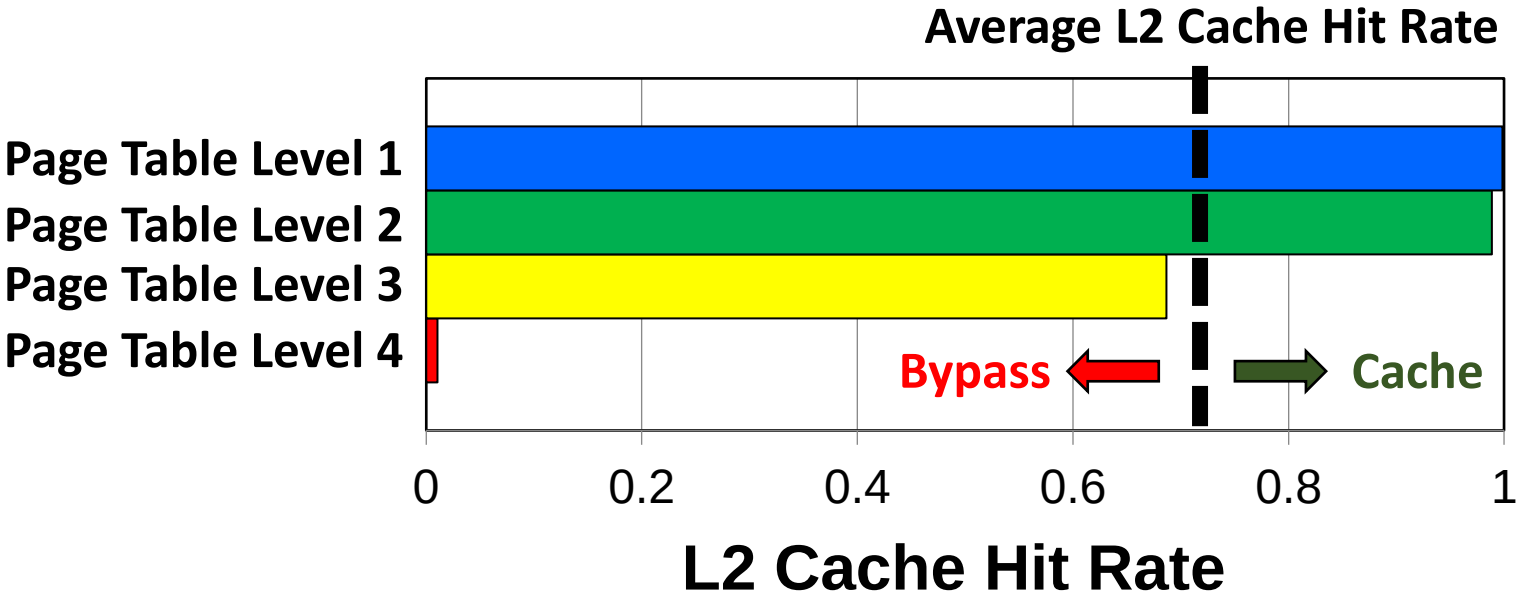


Some address translation data **does not benefit from caching**

Only cache address translation data with high hit rate

B: Translation-aware L2 Bypass

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

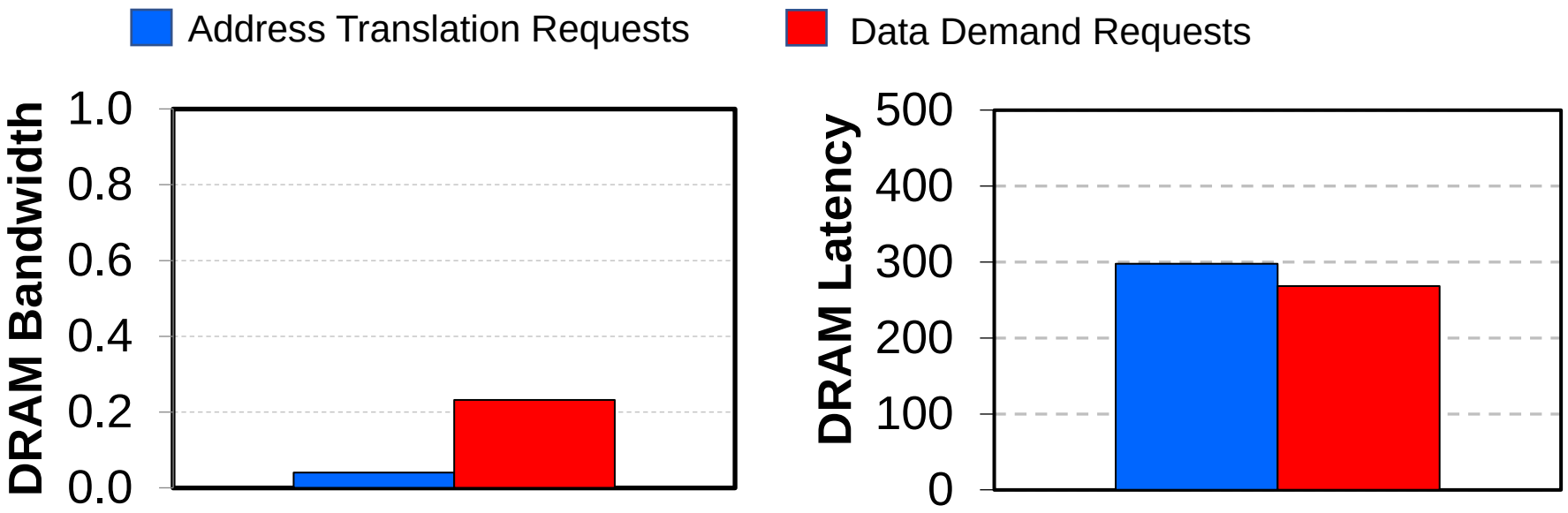


Benefit 1: **Better L2 cache utilization** for translation data

Benefit 2: Bypassed requests → **No L2 queuing delay**

C: Address-space-aware Memory Scheduler

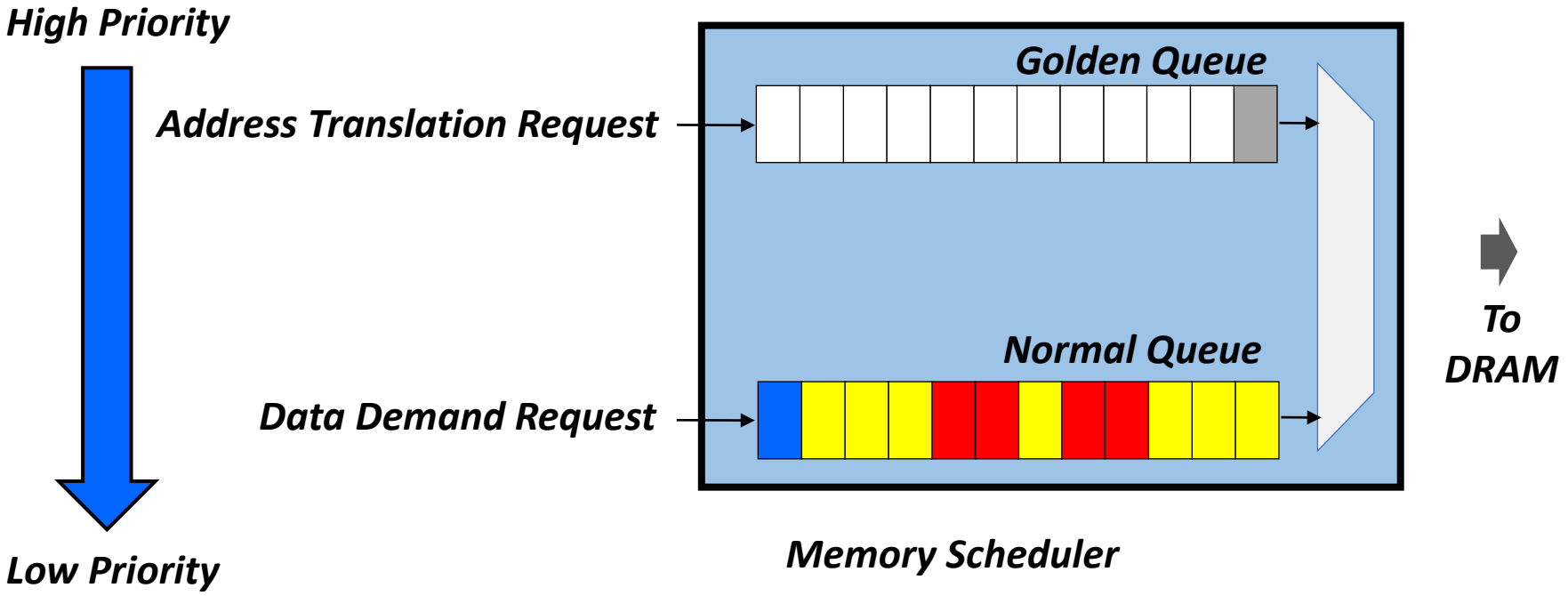
- **Cause:** Address translation requests **are treated similarly** to data demand requests



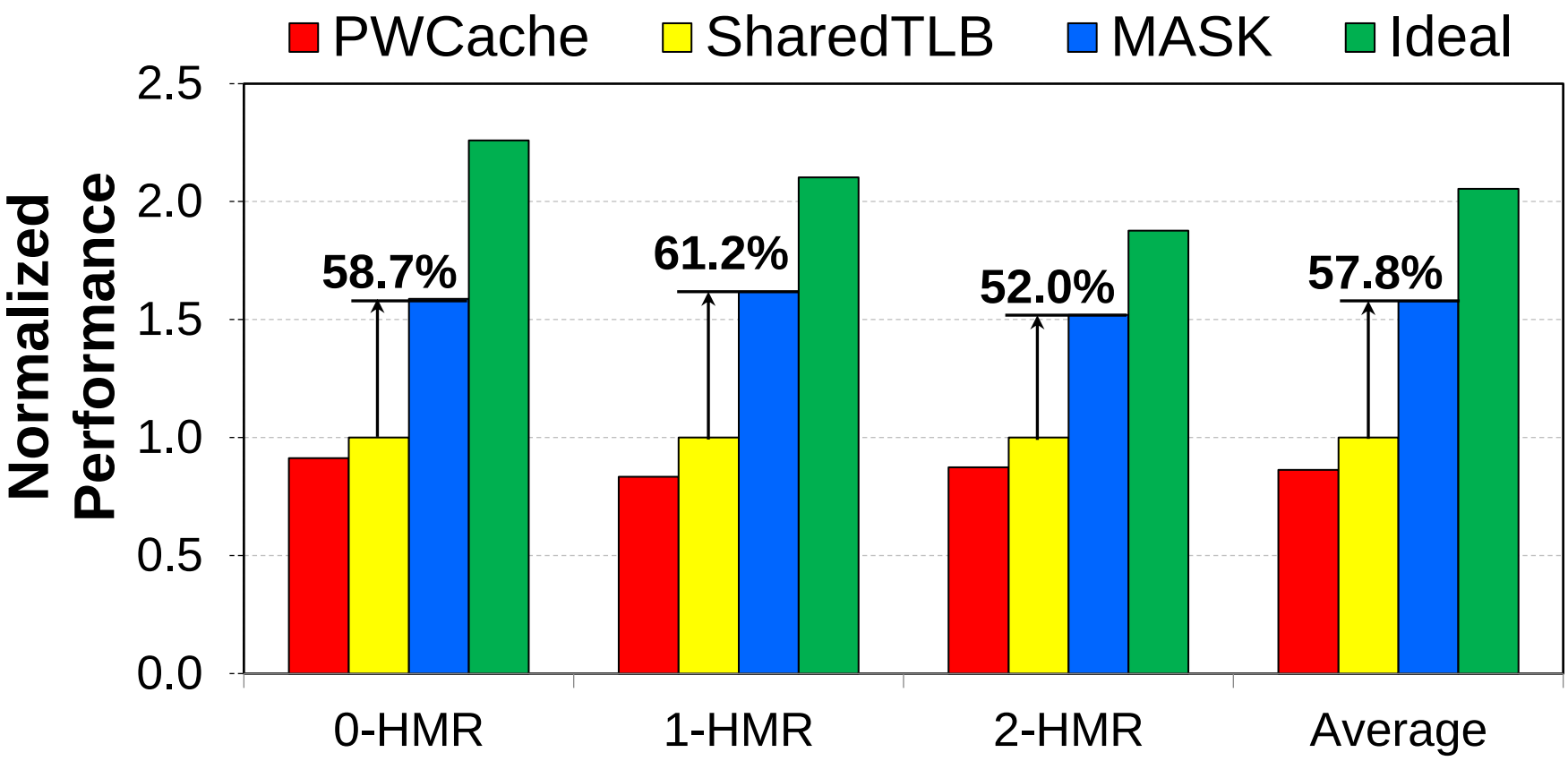
Idea: **Lower address translation request** latency

C: Address-space-aware Memory Scheduler

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



Performance



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

Modern-day Cloud Cluster

- Very heterogeneous
  - Using GPUs to accelerate applications is very common
- Various workload types
  - Each with different demands
  - No one size fits all for the page size
  - Demand special accelerators

System with Huge Pages

- Benefit from larger TLB reach

- Key problems:
  - Fragmentation and bloat
  - Page replacement at large granularity
  - Huge page promotion/demotion

Problem: Fidelity loss from using huge page

Fidelity Loss in 2MB Page Size

Small Page

Ok … That window is dirty

Let me go and clean it

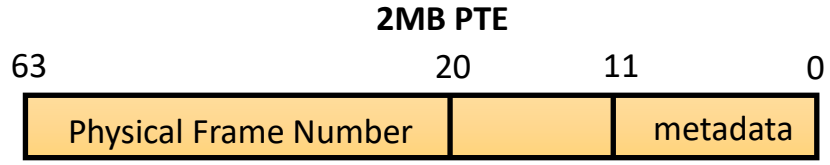
Large Page

Ok … That whole side is dirty

Let's clean this whole side!

Metadata Bits

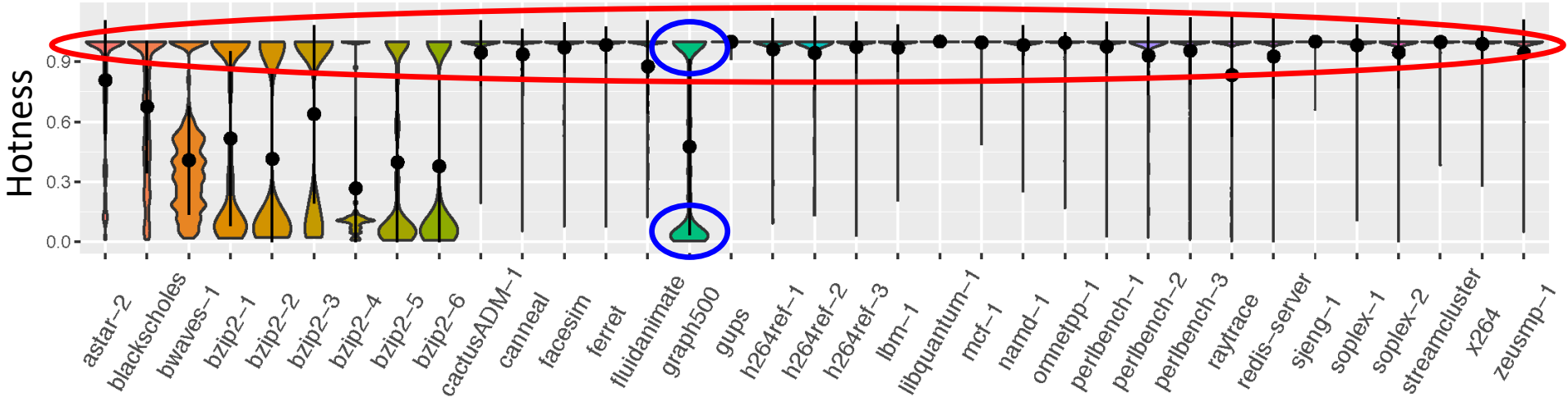
- Allows hardware to tag pages' information
  - Accessed bits
  - Present bits
  - Dirty bits
- OS uses this information to manage pages
  - Replacement policy
  - Mapping of fast/slow memory



Problems: Metadata bit is coupled to page size

1) Reduced fidelity as page size increases
2) OS' visibility into sub-page information reduces

Issues with Single Granularity



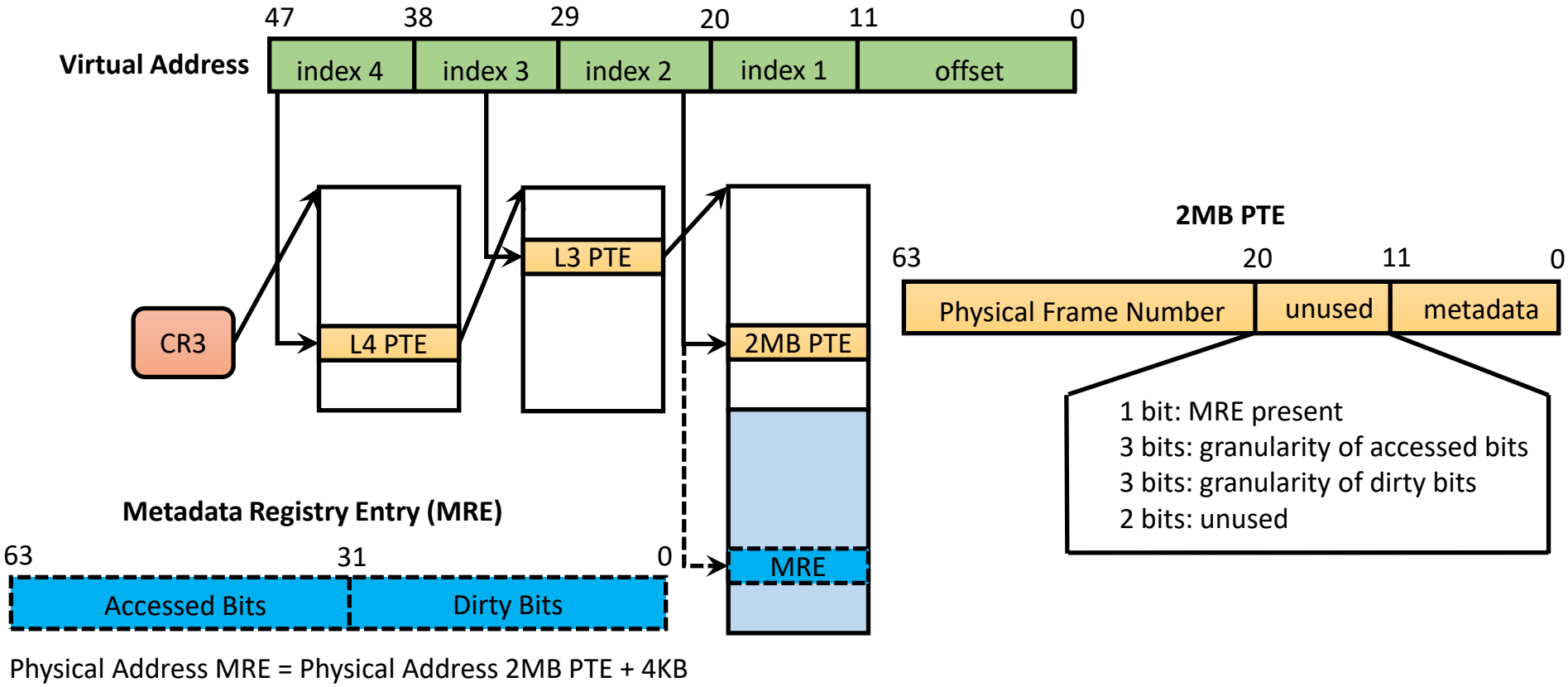
Only a small fraction of the huge page are actually hot

Huge page metadata marks the entire huge page as hot!
(More analysis in the paper)

PRISM Goals

- Decouple metadata granularity from page sizes
- Allow finer-grain and variable metadata granularity
  - Based on applications' demand
- Work on systems that utilize huge pages
  - Applicable to CPU
  - Applicable to GPU

PRISM: Overview



Physical Address MRE = Physical Address 2MB PTE + 4KB

PRISM: Software Interface

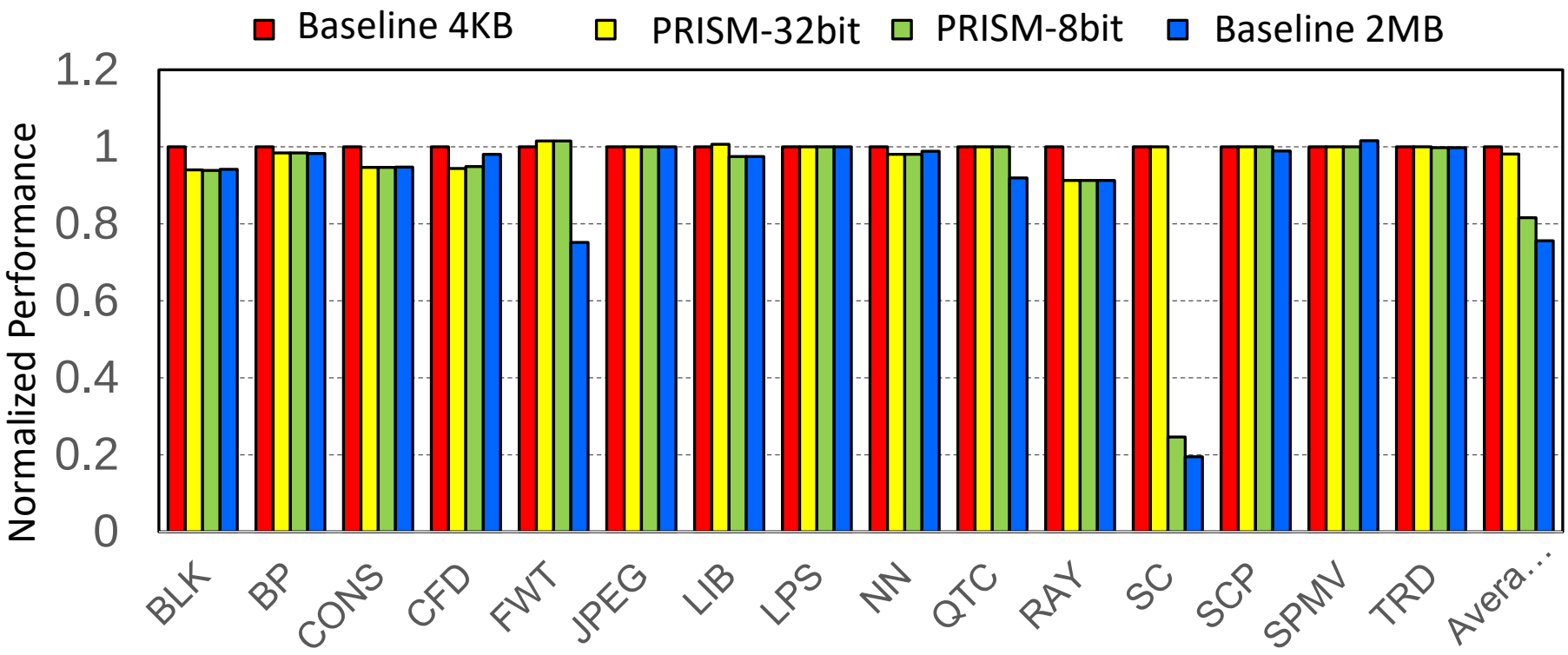
- System call interface similar to mprotect

```
int mmdconfig(void *addr, size_t len, int mdtype, int bits);
```

Case Study: Extend Mosaic for Demand Paging

- Extend Mosaic to
  - Models metadata (accessed/dirty) bits
  - Models LRU page replacement policy
  - Tracks page location (CPU's DRAM vs. GPU's DRAM)
- NVIDIA GTX750 Ti
- 15 GPGPU workloads

Demand Paging Performance



PRISM is effective when applications demand fine-grain metadata

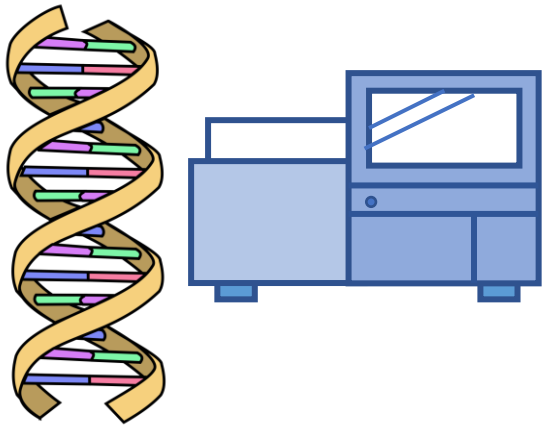
Modern-day Cloud Cluster

- Very heterogeneous
  - Using GPUs to accelerate applications is very common
- Various workload types
  - Each with different demands
  - No one size fits all for the page size
  - Demand special accelerators

Genome Sequence Analysis

- Genome sequence analysis is critical for many applications
  - Personalized medicine
  - Outbreak tracing
  - Evolutionary studies

- Genome sequencing machines extract smaller fragments of the original DNA sequence, known as reads

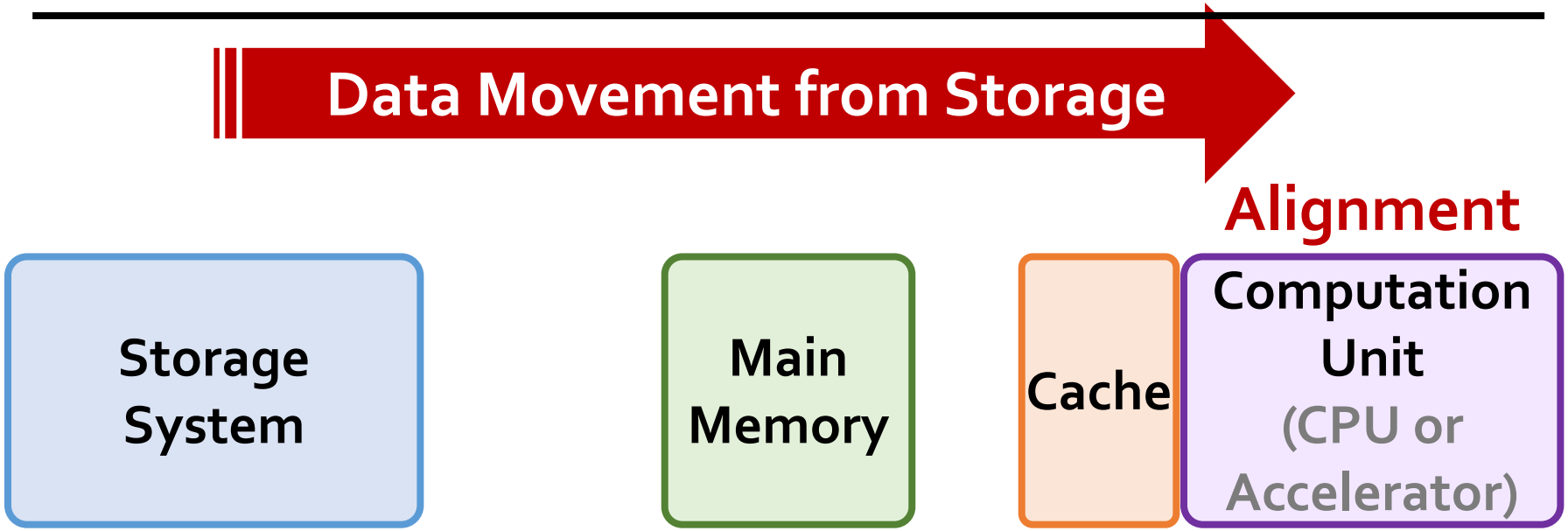


Genome Sequence Analysis

- **Read mapping:** first key step in genome sequence analysis
  - Aligns reads to potential matching locations in the reference genome
  - For each matching location, the alignment step finds the degree of similarity (alignment score)

- Calculating the alignment score requires computationally-expensive approximate string matching (ASM) to account for differences between reads and the reference genome due to:
  - Sequencing errors
  - Genetic variation

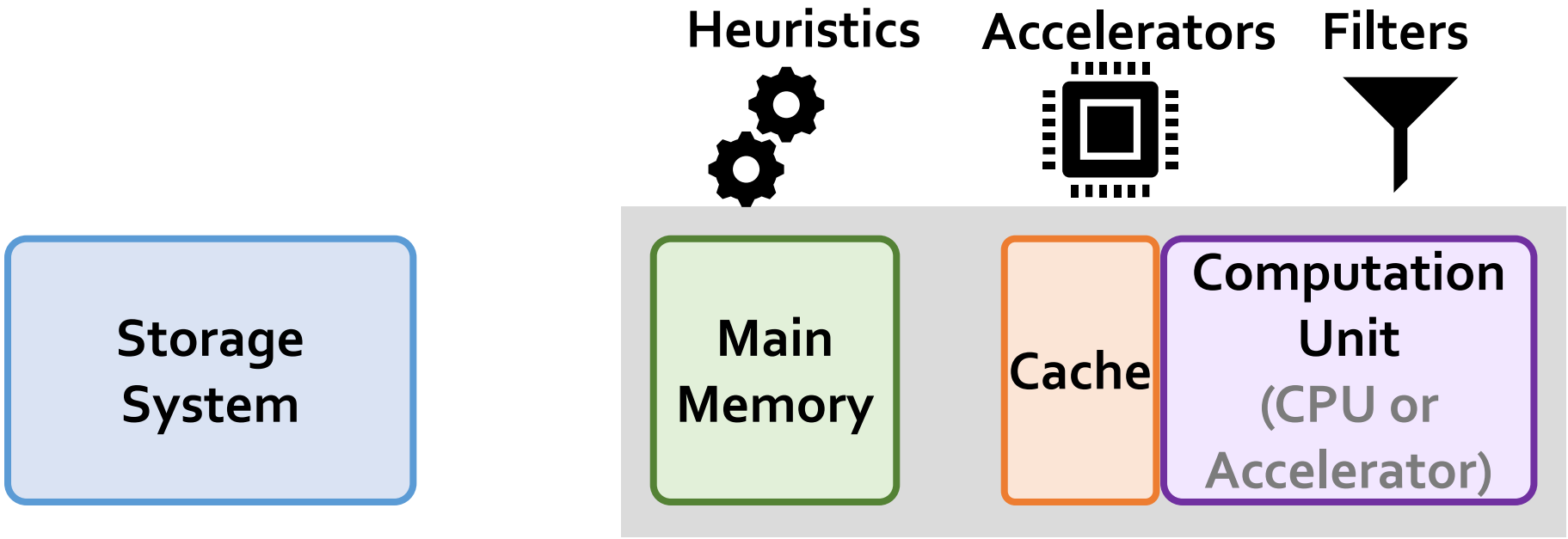
Genome Sequence Analysis



✗ Computation overhead

✗ Data movement overhead

Traditional Solution: Accelerator

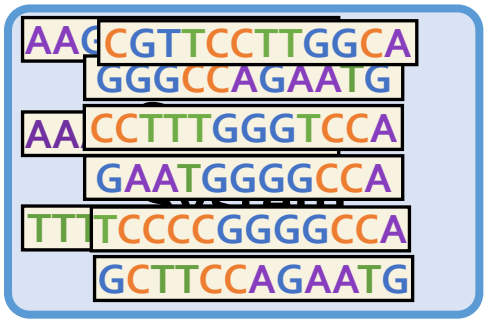


✓ Computation overhead

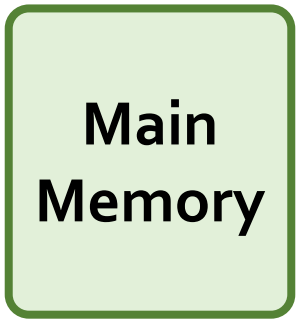
✗ Data movement overhead

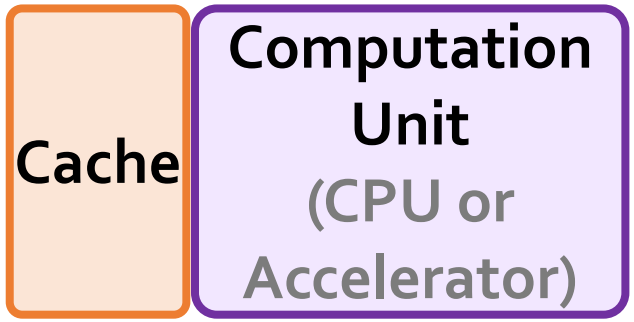
Alternate Solution: In-SSD Computation

Filter* reads that do *not* require alignment *inside the storage system



Filtered Reads





Exactly-matching reads

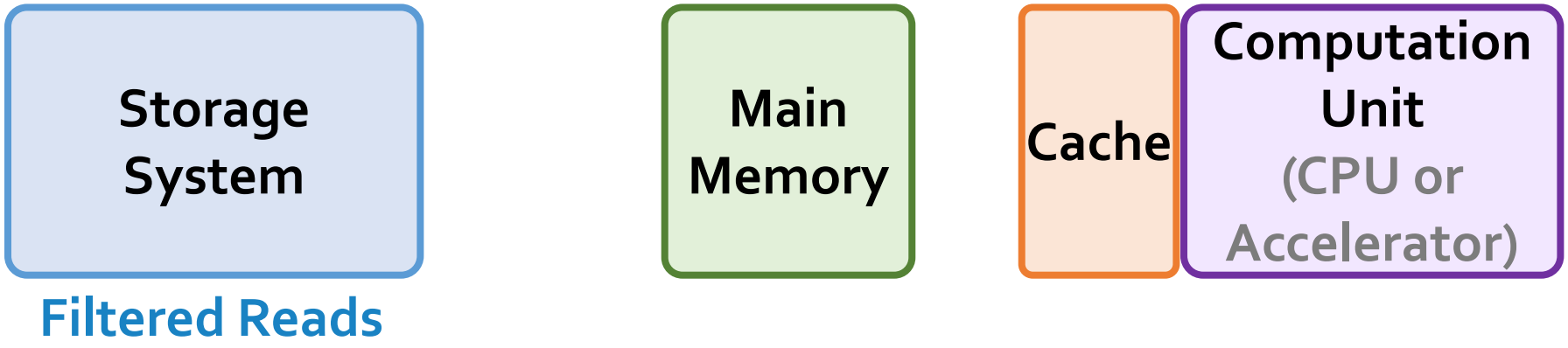
Do not need expensive approximate string matching during alignment

Non-matching reads

Do not have potential matching locations and can skip alignment

Key Challenges

Filter* reads that do *not* require alignment *inside the storage system



Read mapping workloads can exhibit different behavior

There are limited hardware resources in the storage system

GenStore: In-storage Processing

Filter* reads that do *not* require alignment *inside the storage system

GenStore-Enabled Storage System

Main Memory

Cache

Computation Unit (CPU or Accelerator)

✓ **Computation overhead**

✓ **Data movement overhead**

GenStore provides significant speedup (1.4x - 33.6x) and energy reduction (3.9x – 29.2x) at low cost

Recap

- Modern systems are very heterogeneous
  - Require CPU-like virtual memory support
- Issues with virtual memory design
  - **Limited TLB reach and high PW latency**
  - **Solution:**
    - Smart use of huge page
    - Accelerate page walk requests
    - Decouple metadata from page sizes
- Various workload types, each with different demands
  - **Integration with accelerators**
  - **Solution:** In-memory and In-storage accelerators

More Info on the Direction

- Virtual memory design for heterogeneous datacenters
  - Ausavarungnirun et al. “Mosaic: A GPU Memory Manager with Application-Transparent Support for Multiple Page Sizes”, ***MICRO 2017***
  - Ausavarungnirun et al. “MASK: Redesigning the GPU Memory Hierarchy to Support Multi-Application Concurrency”, ***ASPLOS 2018***
  - Ausavarungnirun et al. “PRISM: Architectural Support for Variable-granularity Memory Metadata”, ***PACT 2020***
- Memory management in datacenters
  - Li et al. “A Framework for Memory Oversubscription Management in Graphics Processing Units”, ***ASPLOS 2019***
  - Li et al. “Improving Inter-kernel Data Reuse With CTA-Page Coordination in GPGPU”, ***ICCAD 2021***
  - Choi et al. “Memory Harvesting in Multi-GPU Systems with Hierarchical Unified Virtual Memory”, ***USENIX ATC 2022***
- In-memory and In-storage accelerators for DNA sequencing
  - Senol et al. “GenASM: A Low-Power, Memory-Efficient Approximate String Matching Acceleration Framework for Genome Sequence Analysis”, ***MICRO 2020***
  - Ghiasi et al. “GenStore: A High-Performance and Energy-Efficient In-Storage Computing System for Genome Sequence Analysis”, ***ASPLOS 2022***

High-performance Virtual Memory Design for Modern Architectures

Rachata Ausavarungnirun

https://rausavar.github.io/

Architecture Research Group

TGGS, KMUTNB