Today's Lecture

VICTIMA
Drastically Increasing Address Translation Reach by Leveraging Underutilized Cache Resources

Presented in MICRO 2023

UTOPIA
Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings

Presented in MICRO 2023

Today's Lecture

An Open-Source, Comprehensive and Modular Simulation Framework for Virtual Memory Research

https://github.com/CMU-SAFARI/Virtuoso

UTOPIA

Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings

Presented at MICRO 2023

Konstantinos Kanellopoulos

Rahul Bera, Kosta Stojiljkovic, Nisa Bostanci, Can Firtina, Rachata Ausavarungnirun, Rakesh Kumar, Nastaran Hajinazar, Mohammad Sadrosadati, Nandita Vijaykumar, and Onur Mutlu

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

UNIVERSITY OF TORONTO

NTNU

Executive Summary

Problem: Conventional virtual memory (VM) frameworks enable a virtual address to flexibly map to any physical address. This flexibility necessitates large translation structures leading to:

(1) high translation **latency** and (2) large translation-induced **interference** in the memory hierarchy

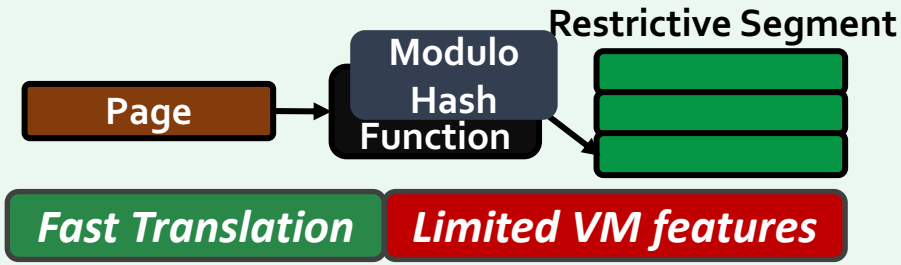
Motivation: Restricting the address mapping leads to compact translation structures and reduces the overheads of address translation. Doing so across the **entire memory** has two major drawbacks:

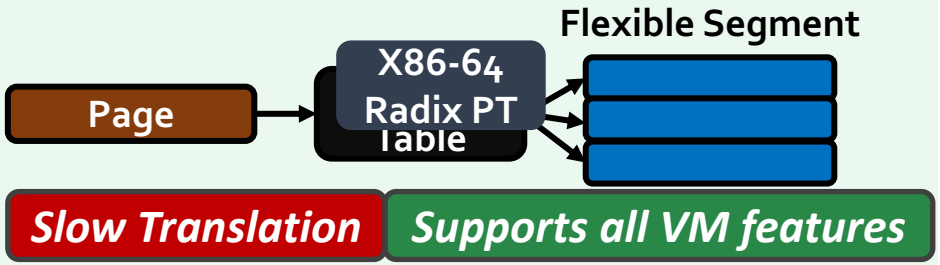
(1) Limits core VM functionalities (e.g., data sharing)

(2) Increases swapping activity in the presence of free physical memory

Key Idea: Utopia is a new hybrid virtual-to-physical address mapping scheme that allows both **flexible** and **restrictive** hash-based address mappings to **harmoniously co-exist** in the system

Utopia manages physical memory using two types of physical memory segments:





Key Results: Outperforms (i) the state-of-the-art **contiguity-aware translation scheme by 13%,** and (ii) achieves **95% of the performance of an ideal perfect-TLB**

https://github.com/CMU-SAFARI/Utopia

Talk Outline

Virtual Memory Background

Address Translation Overheads

Utopia: Hybrid Address Mappings

Utopia: Key Challenges

Evaluation Results

Talk Outline

Virtual Memory Background

Address Translation Overheads

Utopia: Hybrid Address Mappings

Utopia: Key Challenges

Evaluation Results

Virtual Memory Basics

Virtual Memory (VM) is one of the **cornerstones** of most modern computing systems

Conventional VM designs provide :

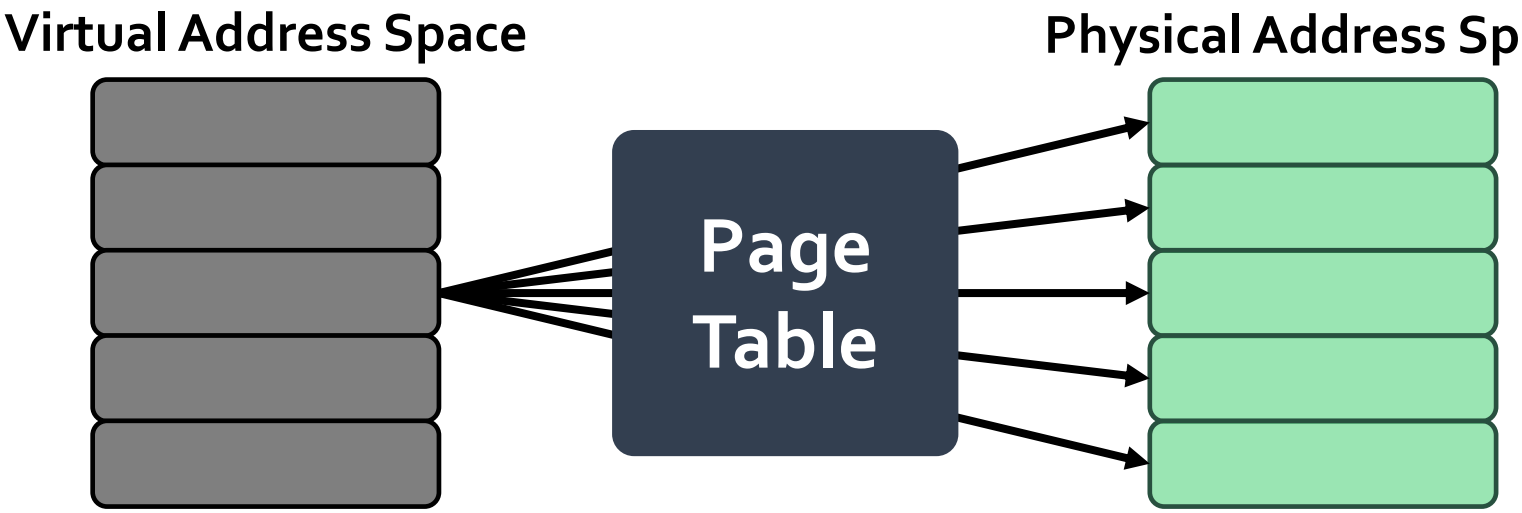
(1) Application-transparent **memory management**

(2) Data **sharing**

(3) Process **isolation**

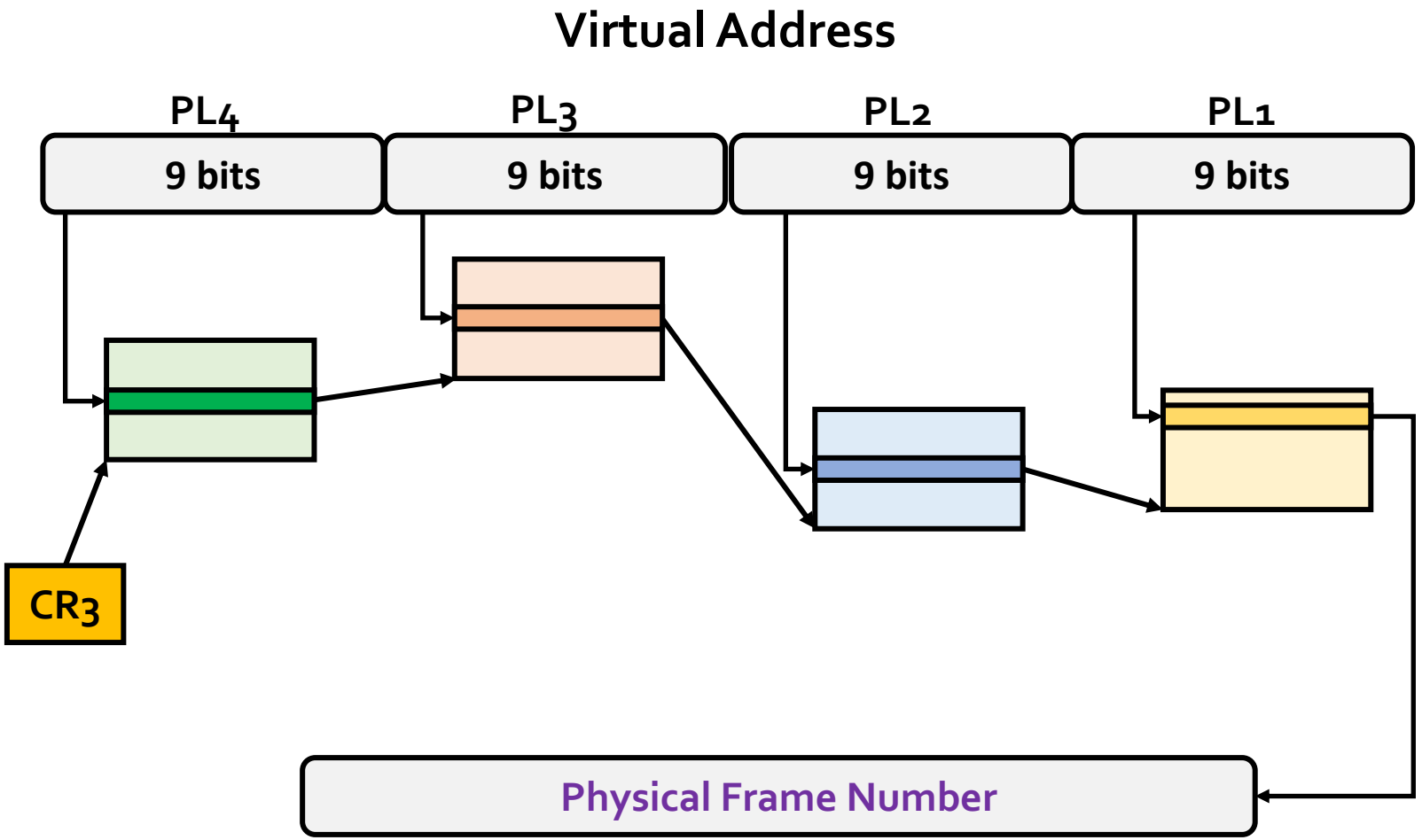
Virtual Memory Basics

A core feature of virtual memory management is that the **mapping** between virtual-to-physical pages is **fully-associative**

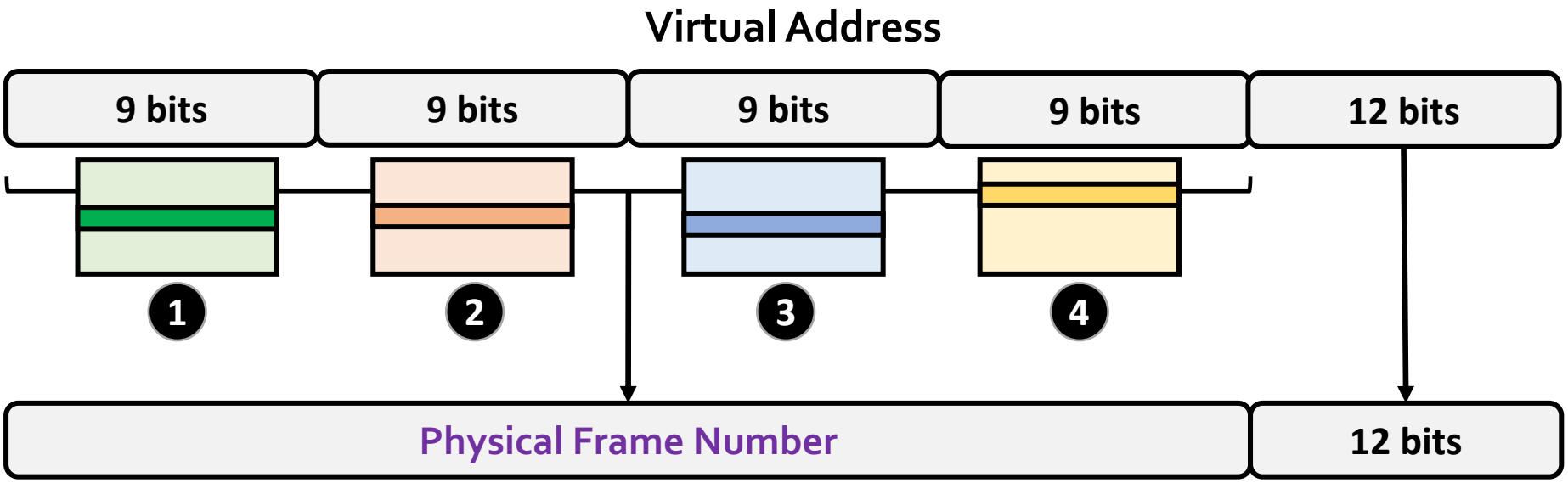


Perform **Page Table Walk (PTW)** to retrieve the mapping

Page Table Walk in x86-64



Page Table Walk in x86-64

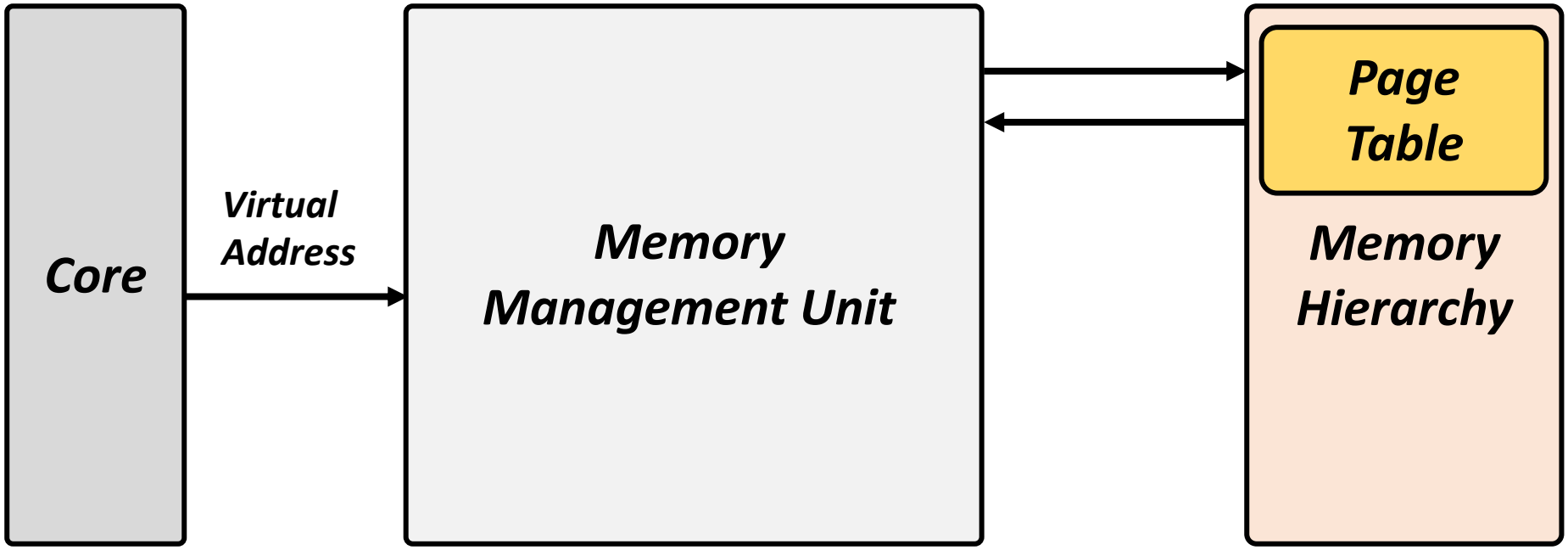


***Four* sequential memory accesses during a page table walk in x86-64**

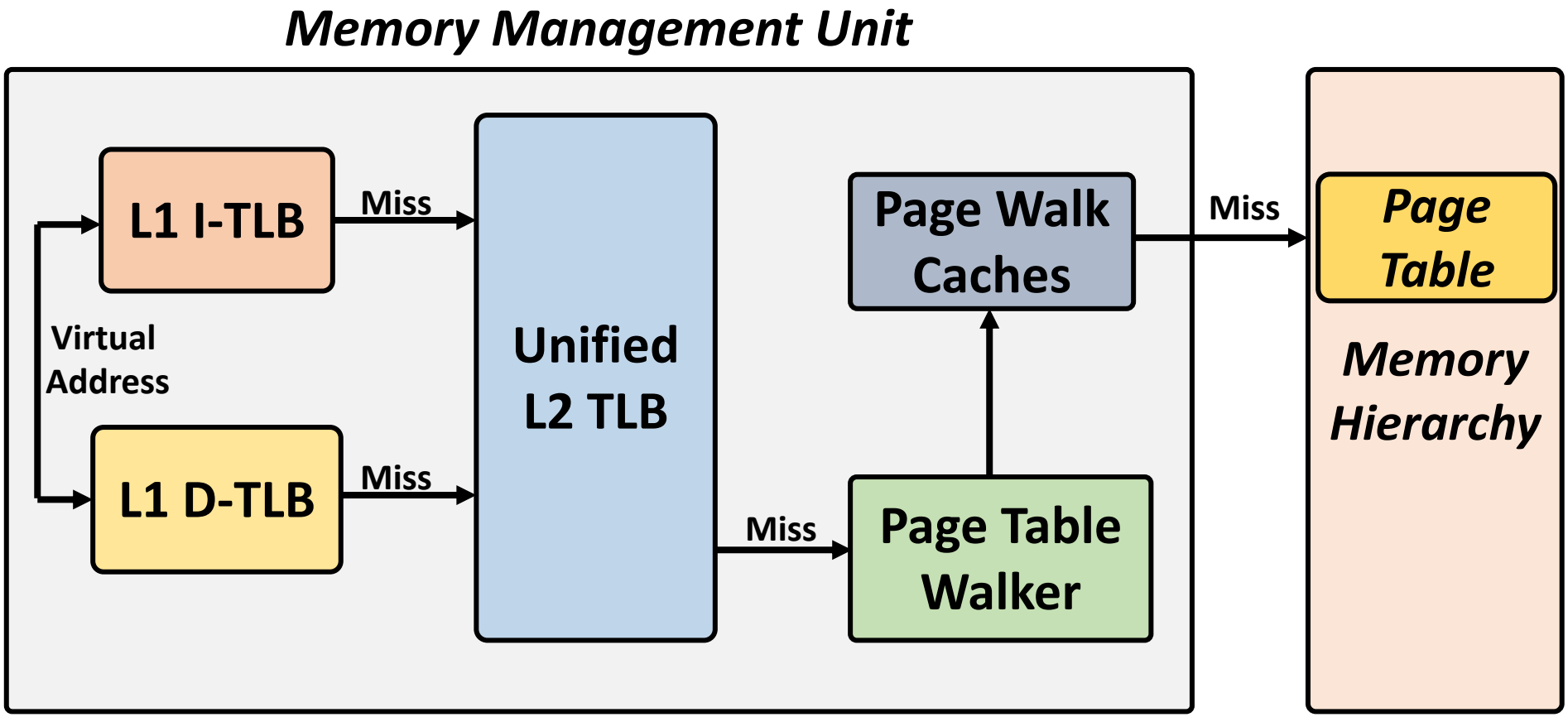
Architectural Support for VM

- To send a memory request to the memory hierarchy, the processor needs to translate the virtual address to the corresponding physical
- Resolving an address translation request requires accessing the **Page Table**
- Modern processors employ a specialized hardware unit called **Memory Management Unit (MMU)** to accelerate address translation

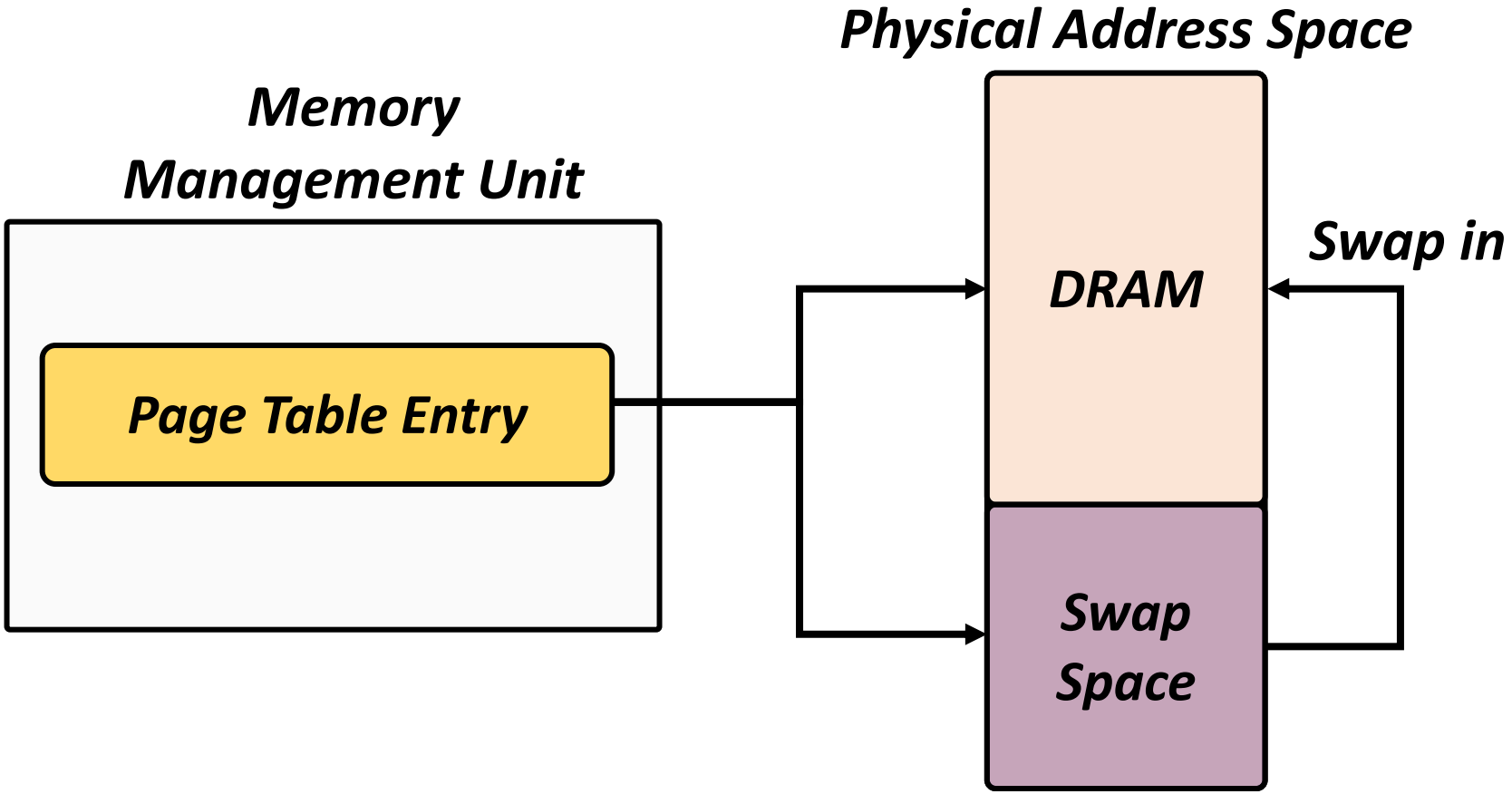
Address Translation Flow (I)



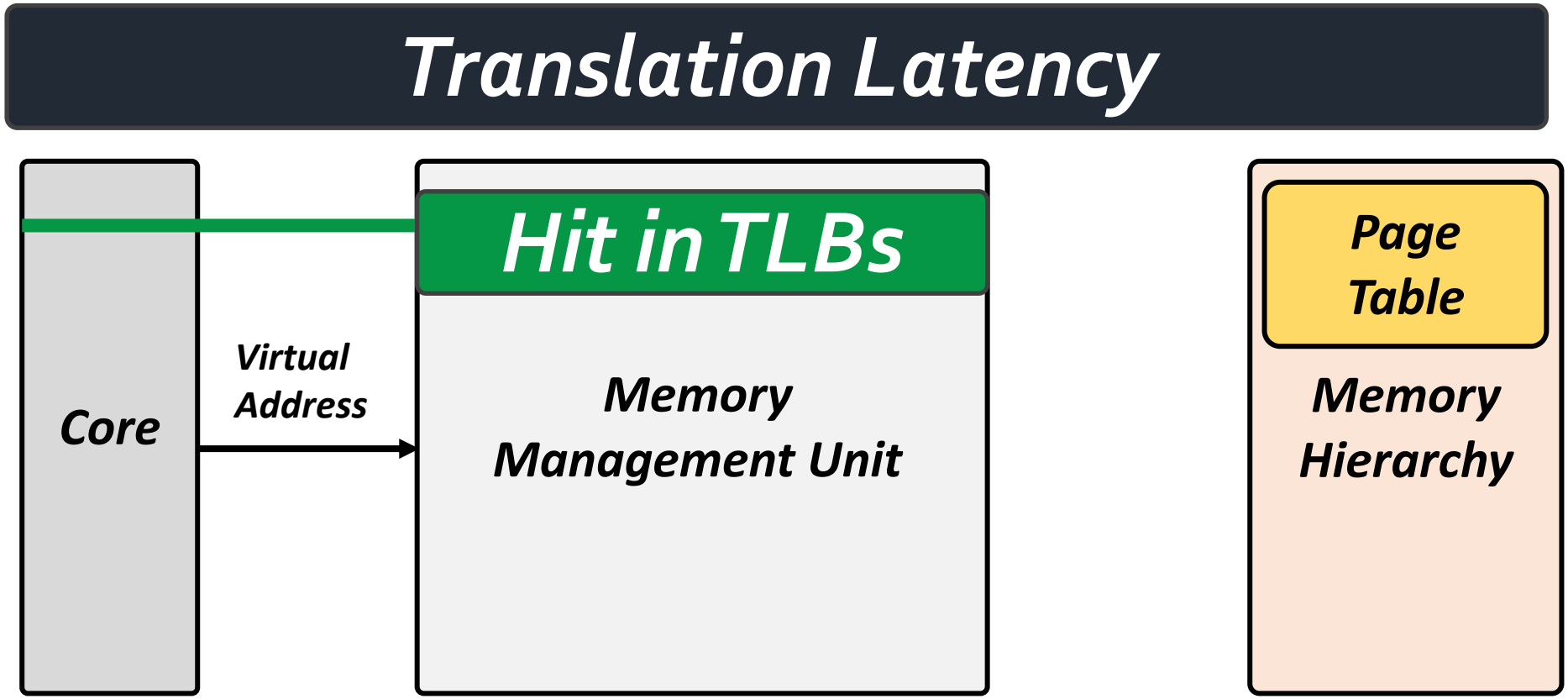
Address Translation Flow (II)



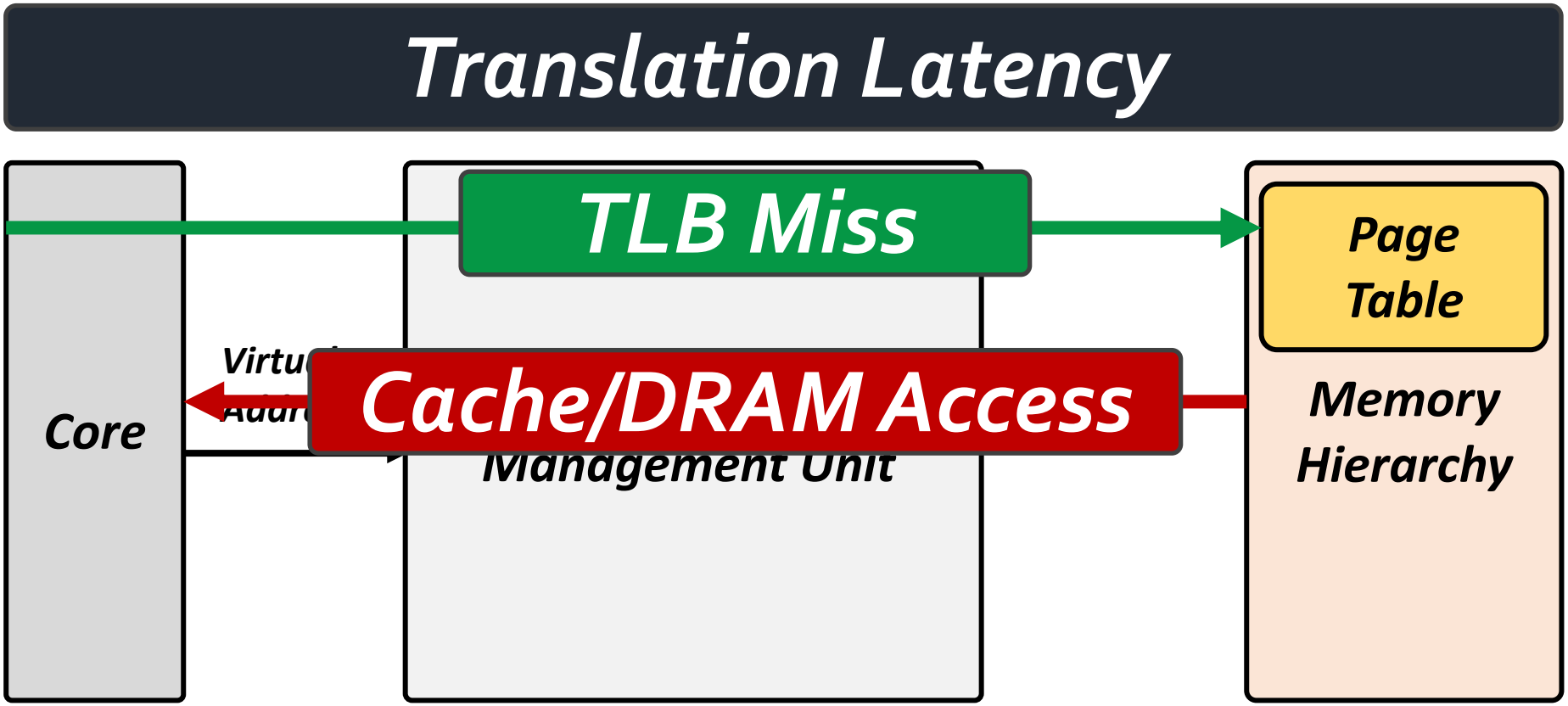
Address Translation Flow (III)



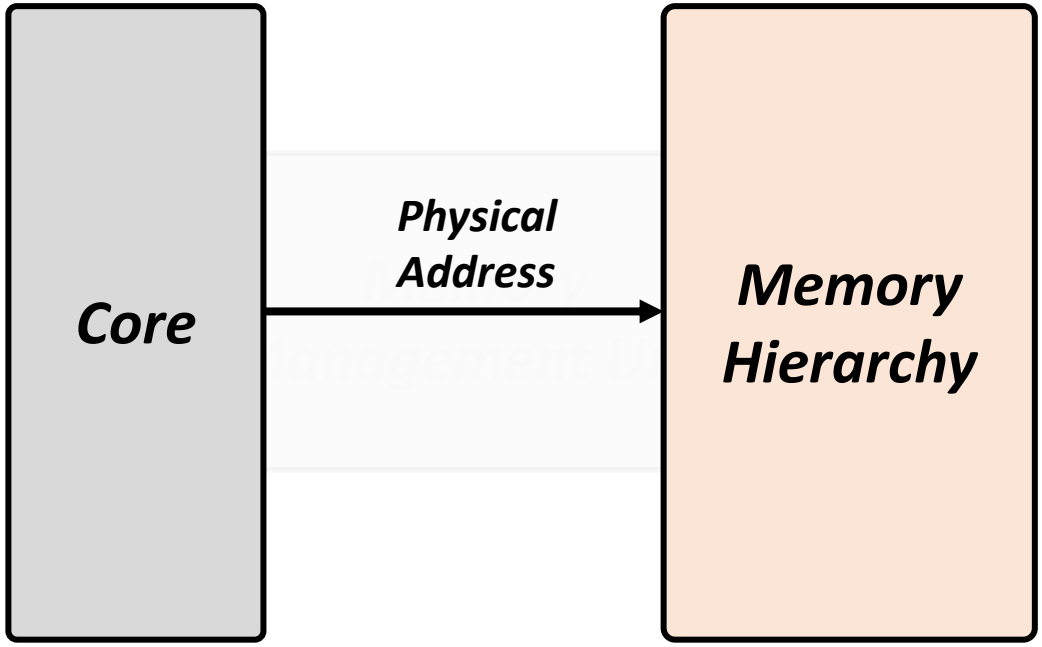
Address Translation Flow (III)



Address Translation Flow (III)



Address Translation Flow (IV)



Talk Outline

Virtual Memory Background

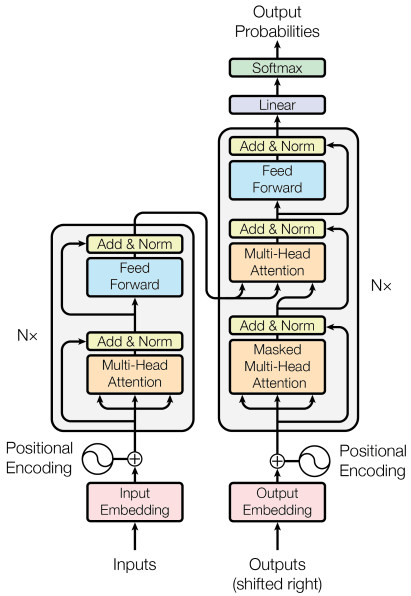
Address Translation Overheads

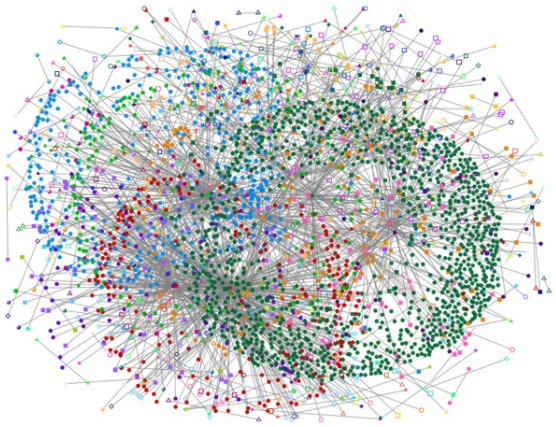
Utopia: Hybrid Address Mappings

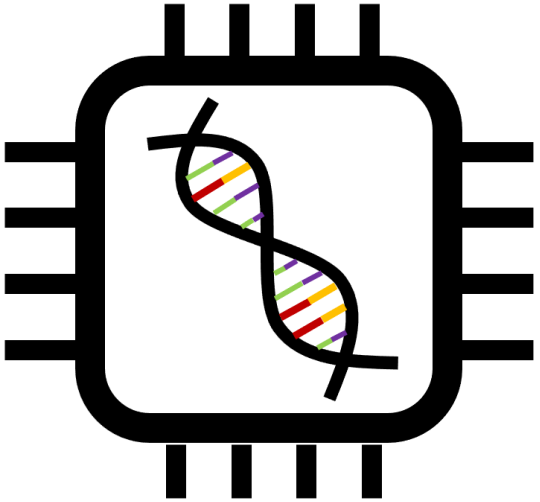
Utopia: Key Challenges

Evaluation Results

Data-Intensive Workloads







Generative AI | **Graph Analytics** | **Bioinformatics**

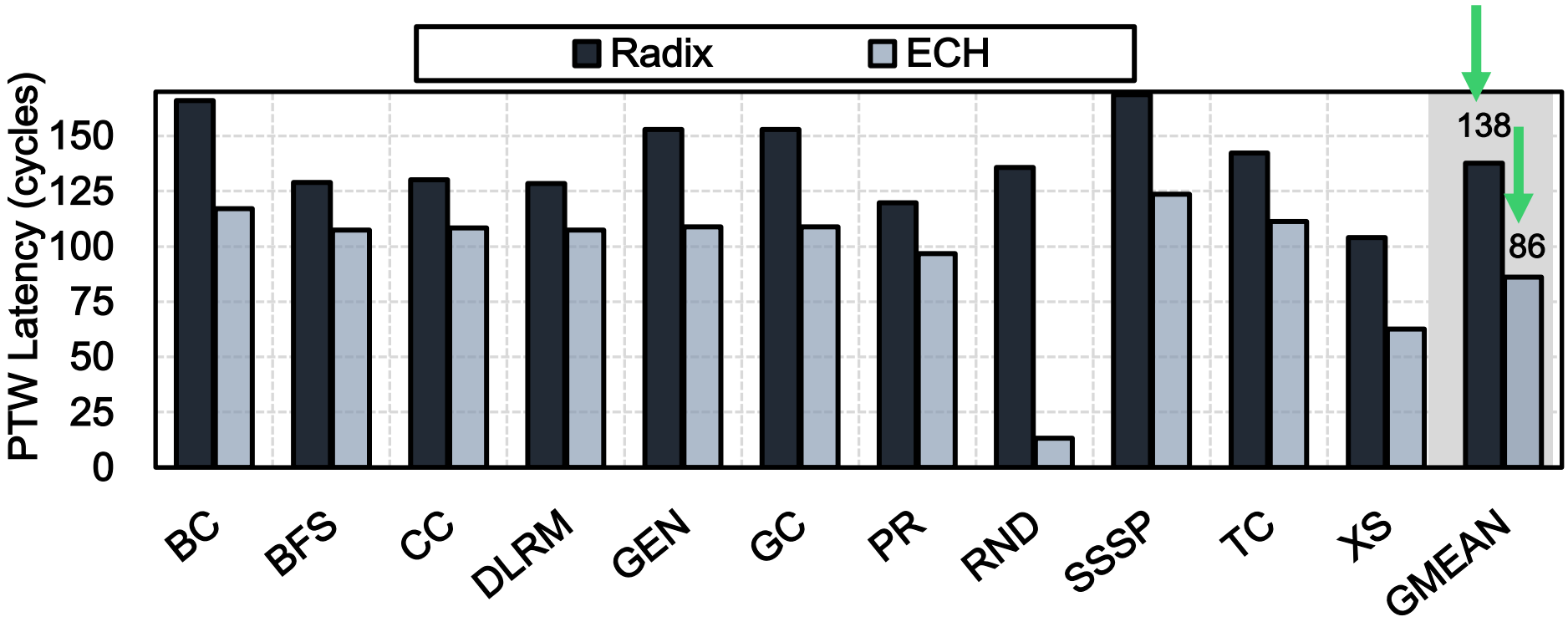
High address translation overheads

Address Translation Overhead

High-latency page table walks

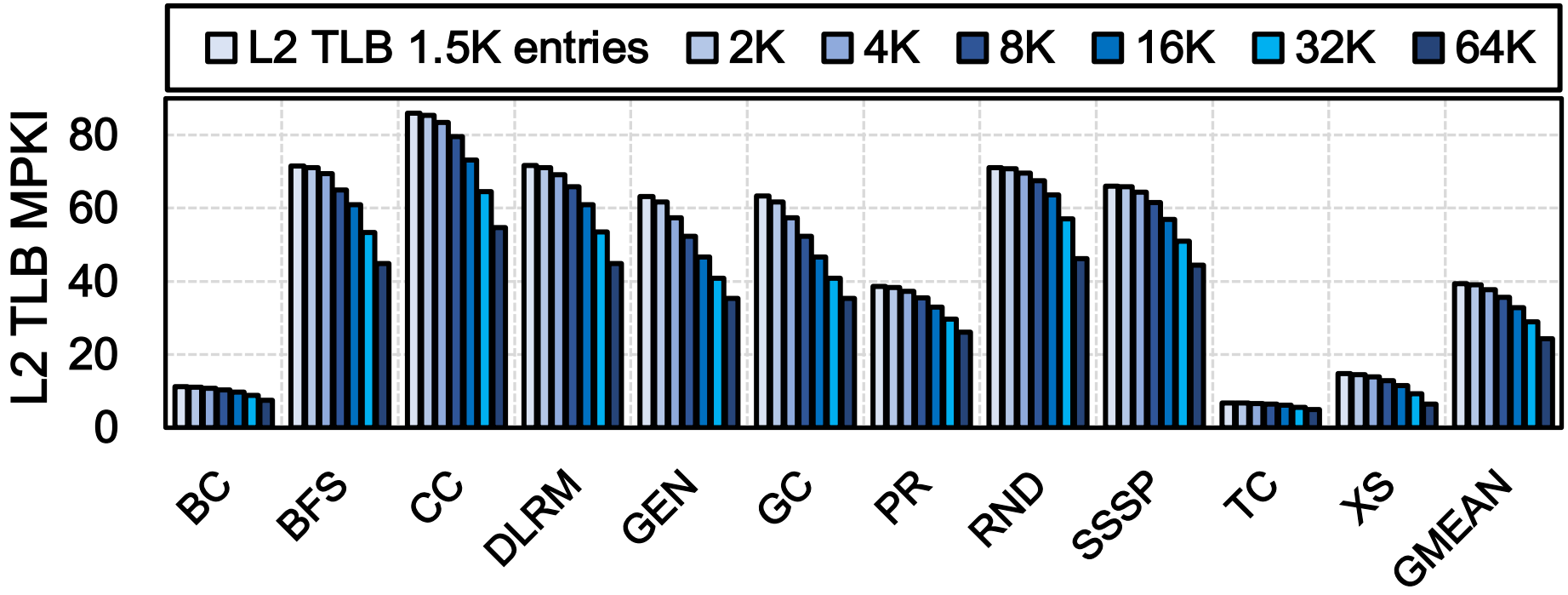
Frequent page table walks

High Latency PTWs



86 cycles on average to access the state-of-the-art hash-based PT

Frequent PTWs



Even the largest 64K-entry L2 TLB experiences 24 MPKI on average

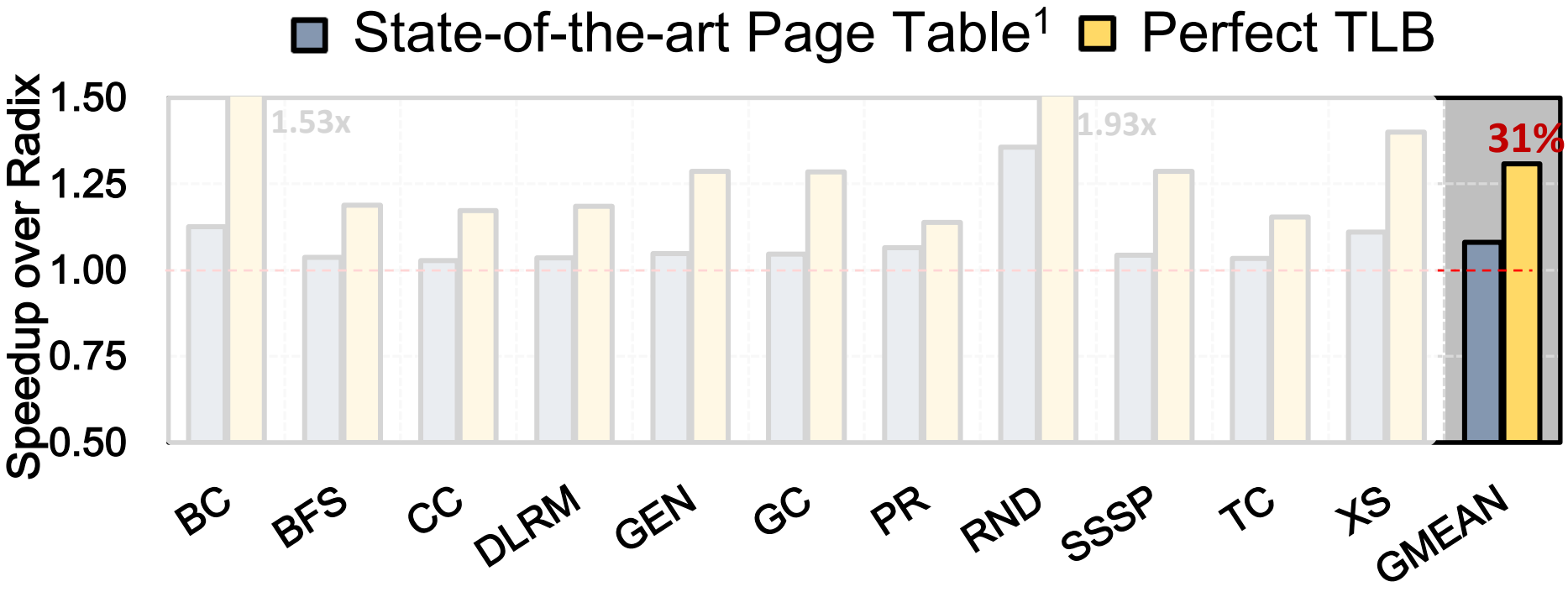
Address Translation Overhead

High latency ***PTWs*** + ***Frequent*** ***PTWs***

High performance overheads

High interference in memory hierarchy

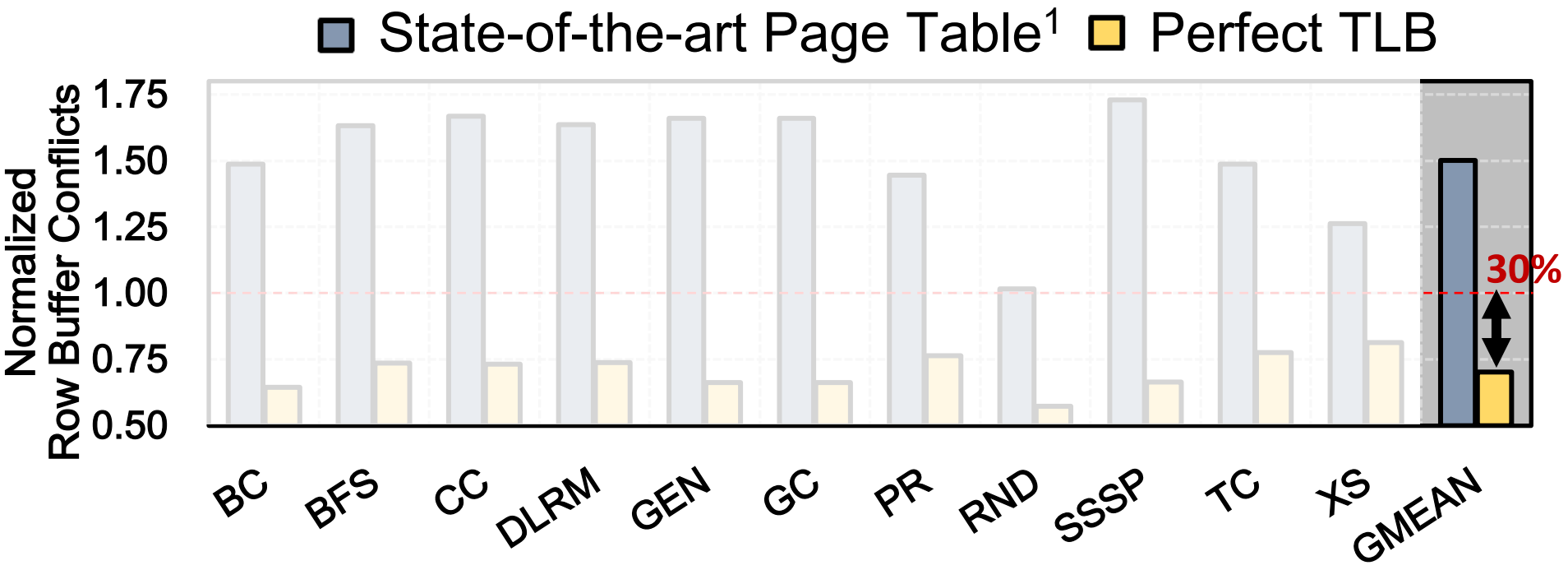
High Performance Overhead



Completely avoiding address translation leads on average to 31% higher performance

[1] Skarlatos et al. "Elastic Cuckoo Page Tables: Rethinking Virtual Memory Translation for Parallelism" ASPLOS 2020

High Interference in Main Memory

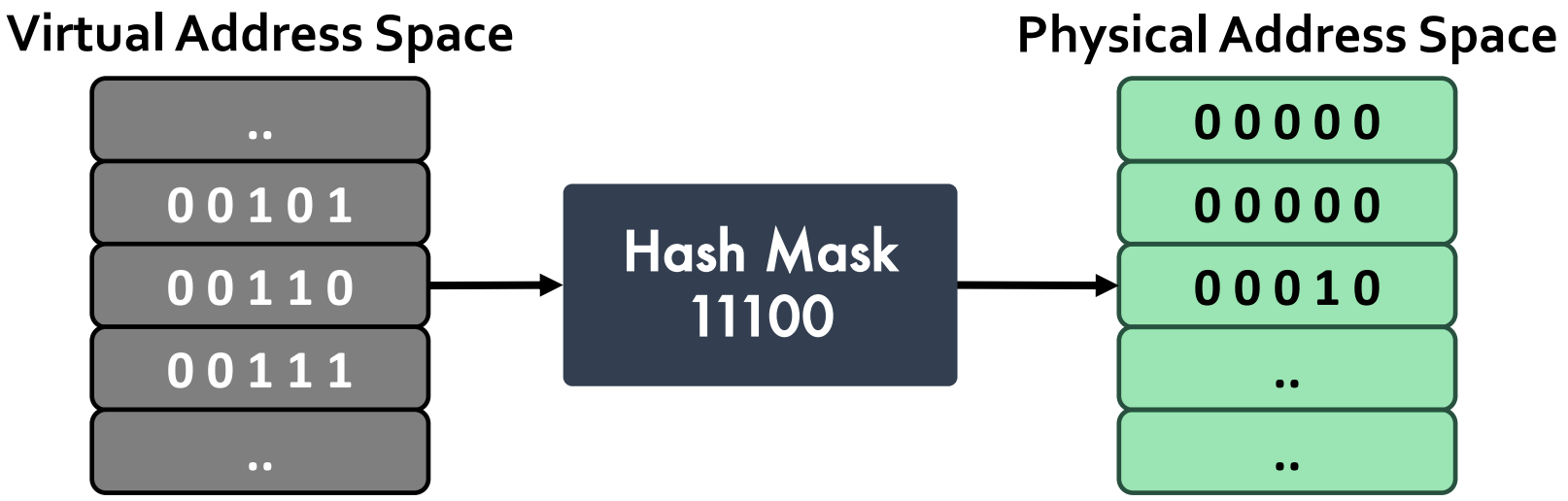


Completely avoiding address translation leads to 30% fewer DRAM row buffer conflicts

[1] Skarlatos et al. "Elastic Cuckoo Page Tables: Rethinking Virtual Memory Translation for Parallelism" ASPLOS 2020

Idea: Restricting VA-to-PA Mapping

Restrict the VA-to-PA mapping to perform fast address translation[1,2]

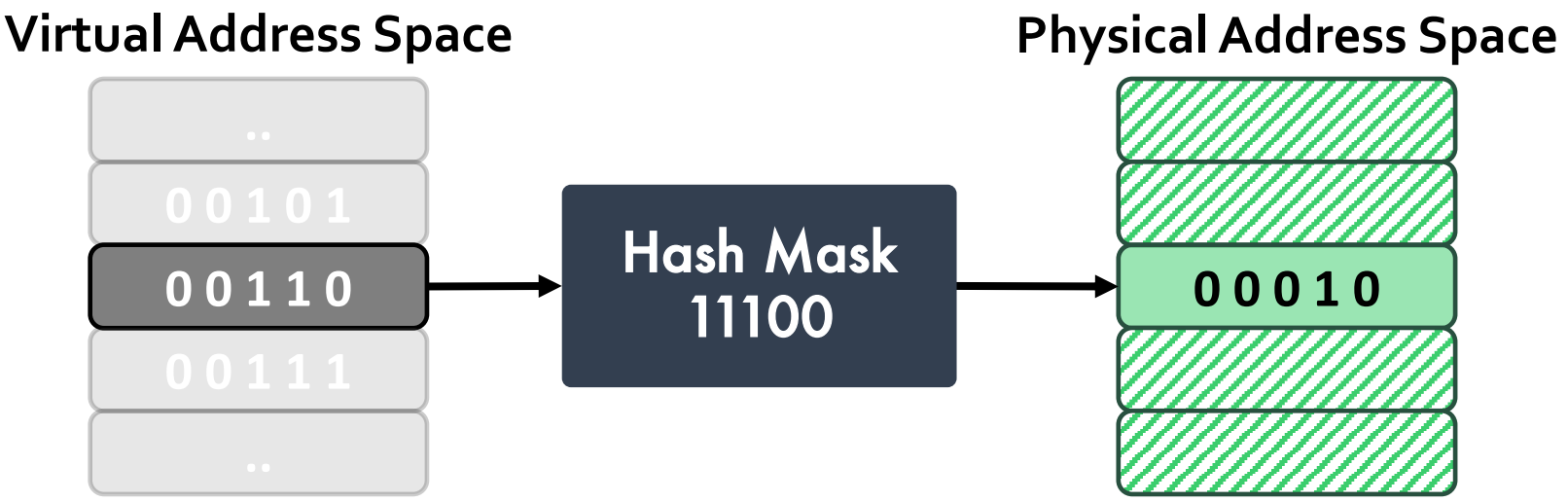


[1] Picorel et al. "Near-Memory Address Translation" PACT 2017

[2] Gosakan et al. "Mosaic Pages: Big TLB Reach with Small Pages" ASPLOS 2023

Idea: Restricting VA-to-PA Mapping

Restrict the VA-to-PA mapping to perform fast address translation[1,2]



[1] Picorel et al. "Near-Memory Address Translation" PACT 2016

[2] Gosakan et al. "Mosaic Pages: Big TLB Reach with Small Pages" ASPLOS 2023

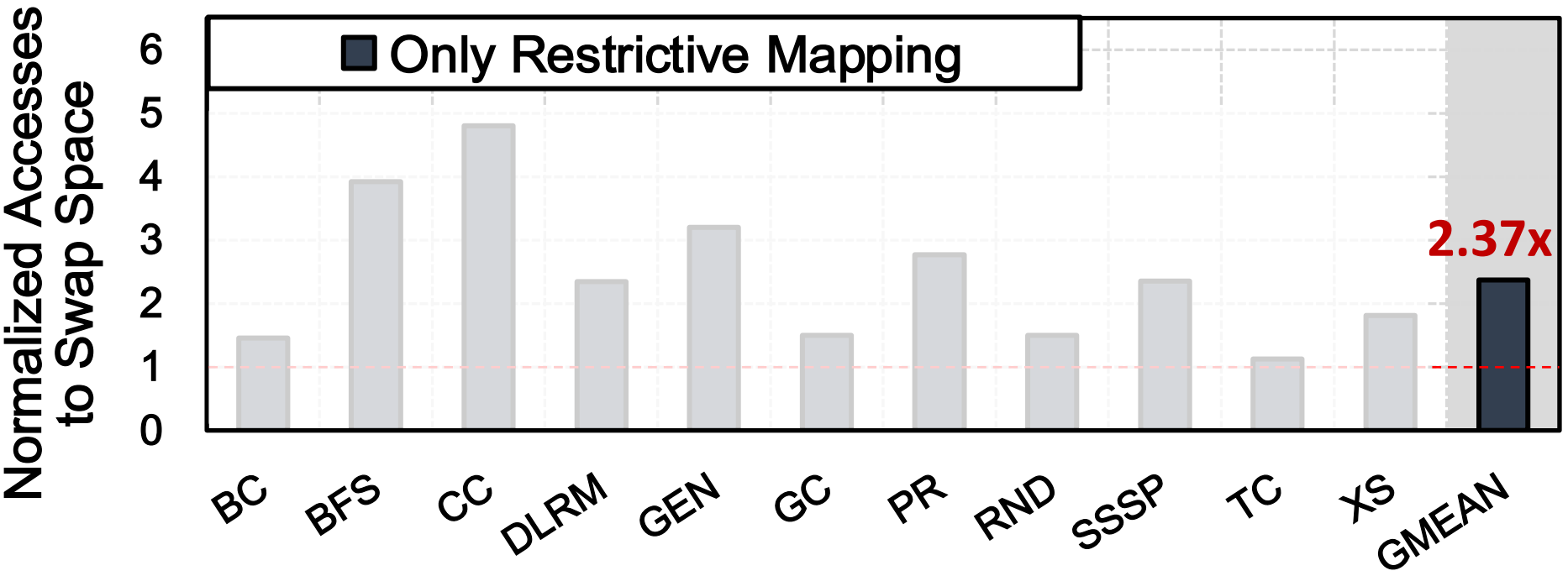
Drawbacks of Restricting VA-to-PA Mapping

Employing a restrictive mapping across the entire memory comes with two key drawbacks:

1. Limits core **VM functionalities** such as data sharing

2. Increases **swapping activity** since the system cannot map virtual pages to free physical pages

Effect on Swapping Activity



Sole use of restrictive mapping leads to 2.37x higher swapping activity over the baseline

Our Goal

Design a virtual-to-physical address mapping scheme that:

- Provides fast and efficient translation using a **restrictive hash-based** address mapping
- Enjoys the benefits of the conventional **fully-flexible** address mapping

Talk Outline

Virtual Memory Background

Address Translation Overheads

Utopia: Hybrid Address Mappings

Utopia: Key Challenges

Evaluation Results

Utopia: Key Idea

We propose **Utopia**, a new virtual-to-physical mapping scheme that enables both:

Restrictive Mapping **Flexible Mapping**

Harmoniously co-exist in the system

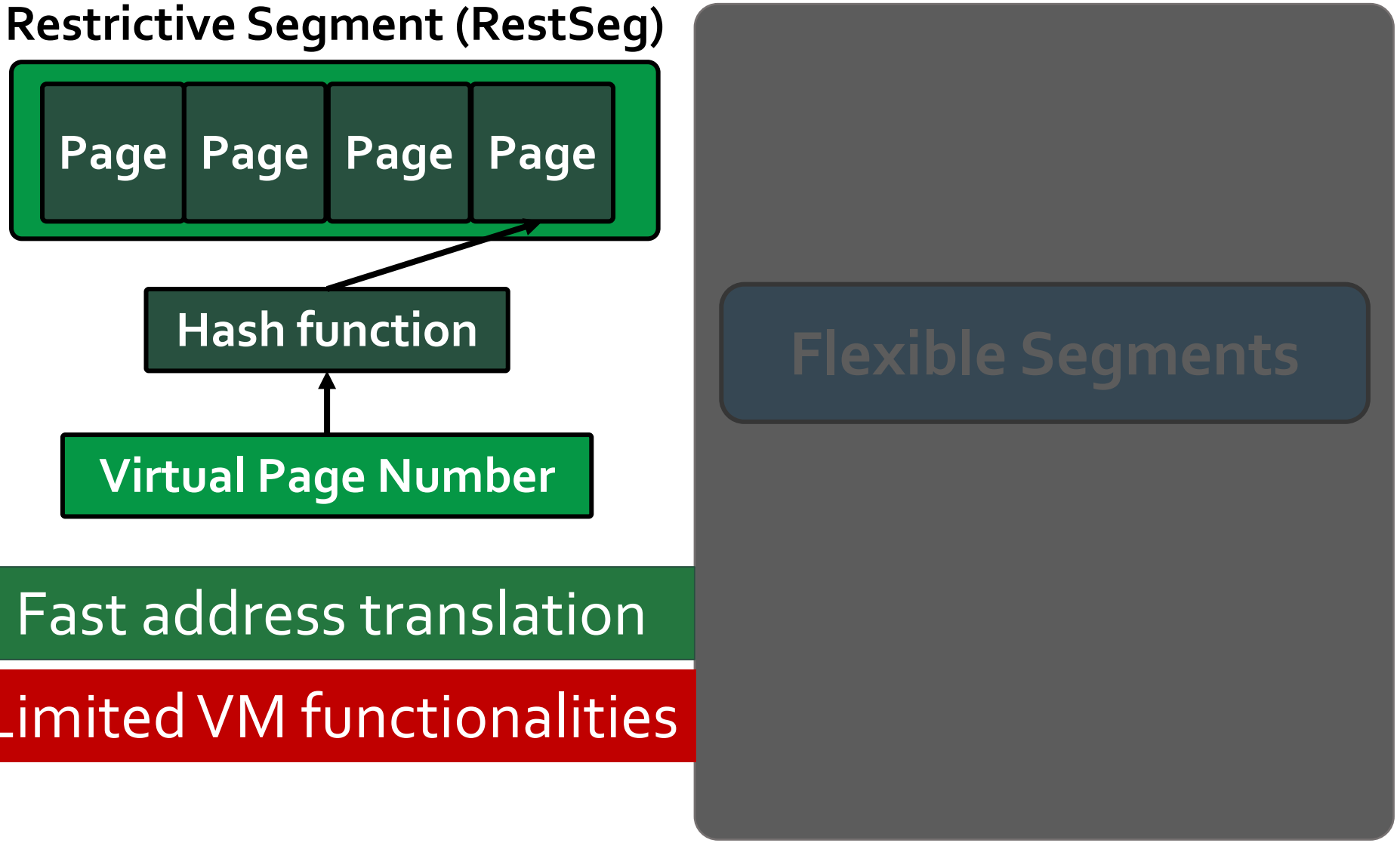
Utopia: Key Idea

Manage physical memory using two types of physical memory segments:

Restrictive Segments

Flexible Segments

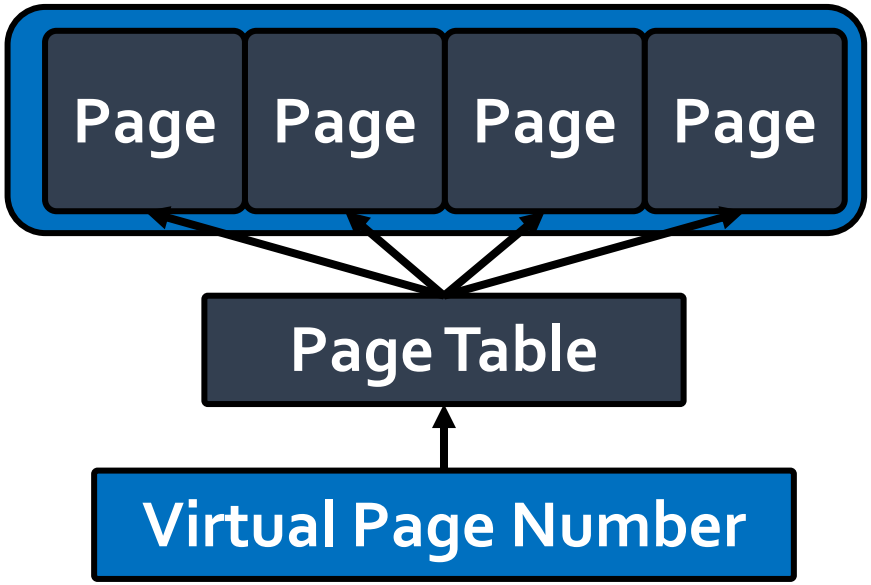
Utopia: Key Idea (I)



Utopia: Key Idea (II)

Restrictive Segments

Flexible Segment (FlexSeg)



Supports all conventional VM features

High-latency address translation

RestSeg Properties

Structural Properties

Address Translation for Data in RestSeg

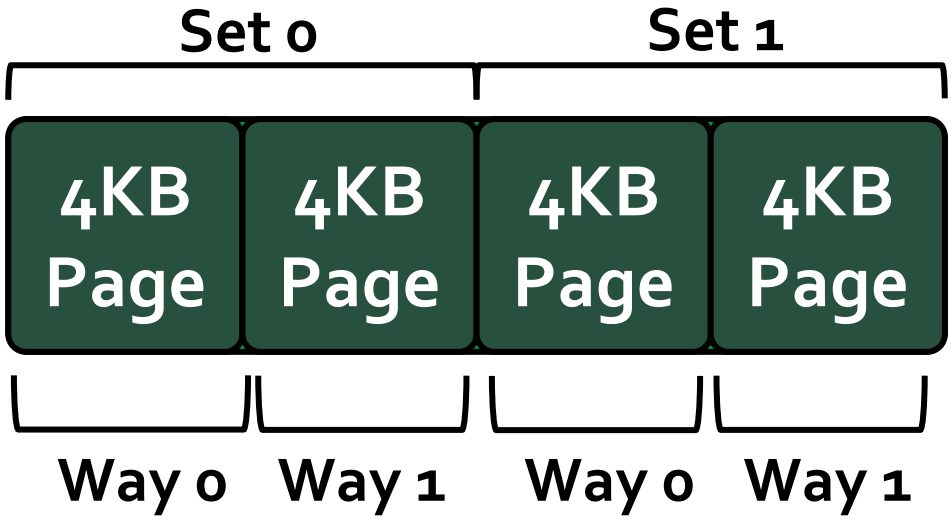
RestSeg Properties

Structural Properties

RestSeg is organized in
a **set-associative** manner
similar to how hardware caches operate

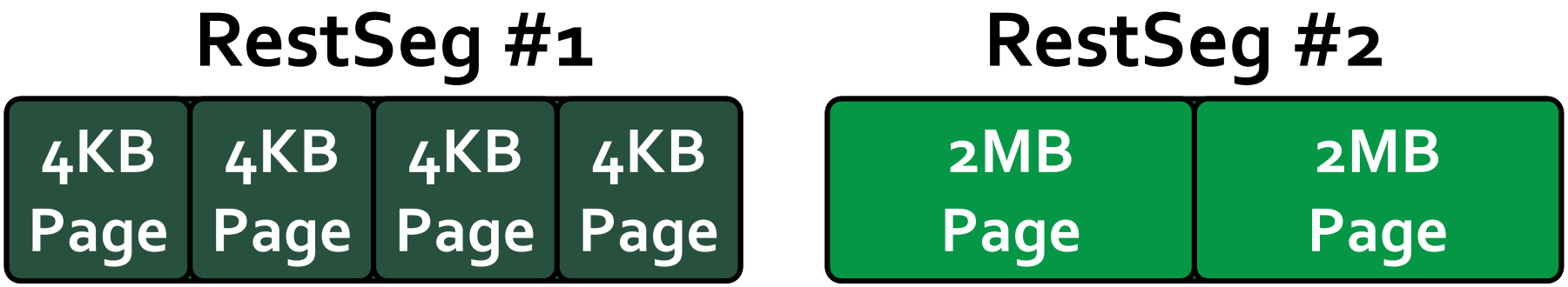
RestSeg: Structural Properties

Example: 2-way associative RestSeg with 2 sets



Set-associative design offers high flexibility

Multiple RestSegs in the System



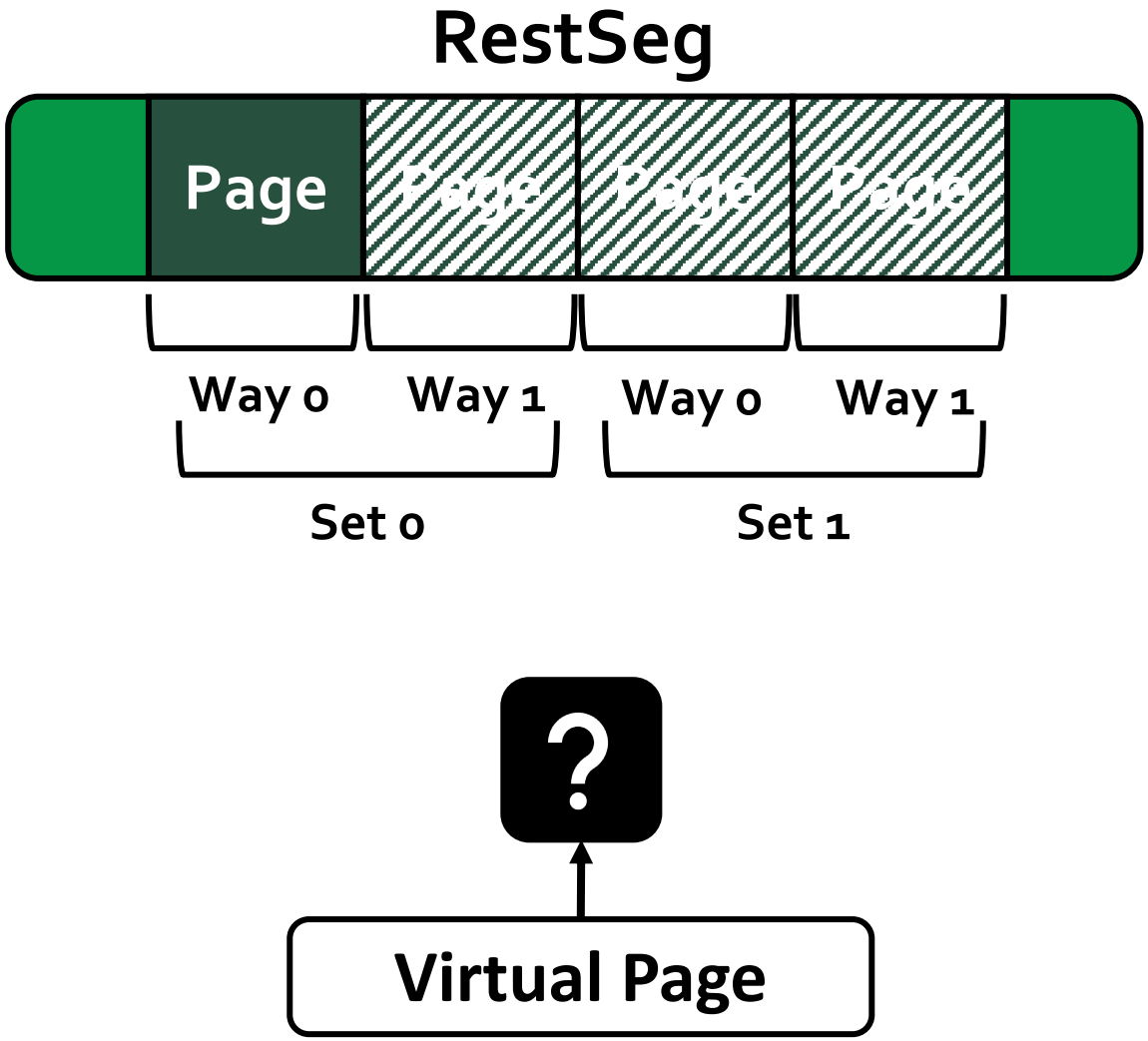
Backward compatible with large page mechanisms

RestSeg Properties

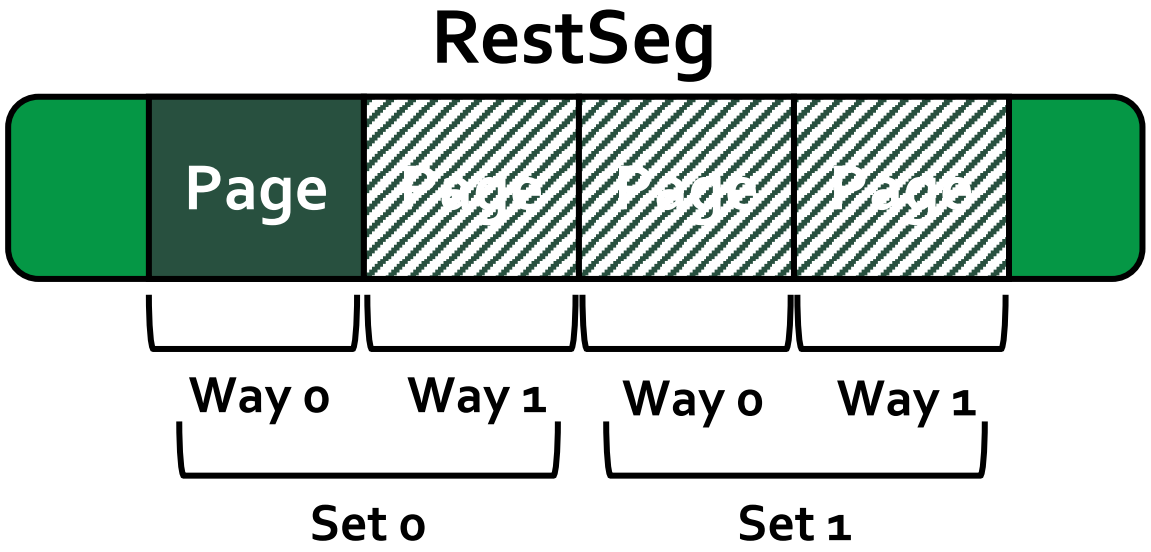
Structural Properties

Address Translation for Data in RestSeg

Restrictive Segment Walk (RSW)



Restrictive Segment Walk (RSW)



How can we find out the physical location of the virtual page?

Virtual Page

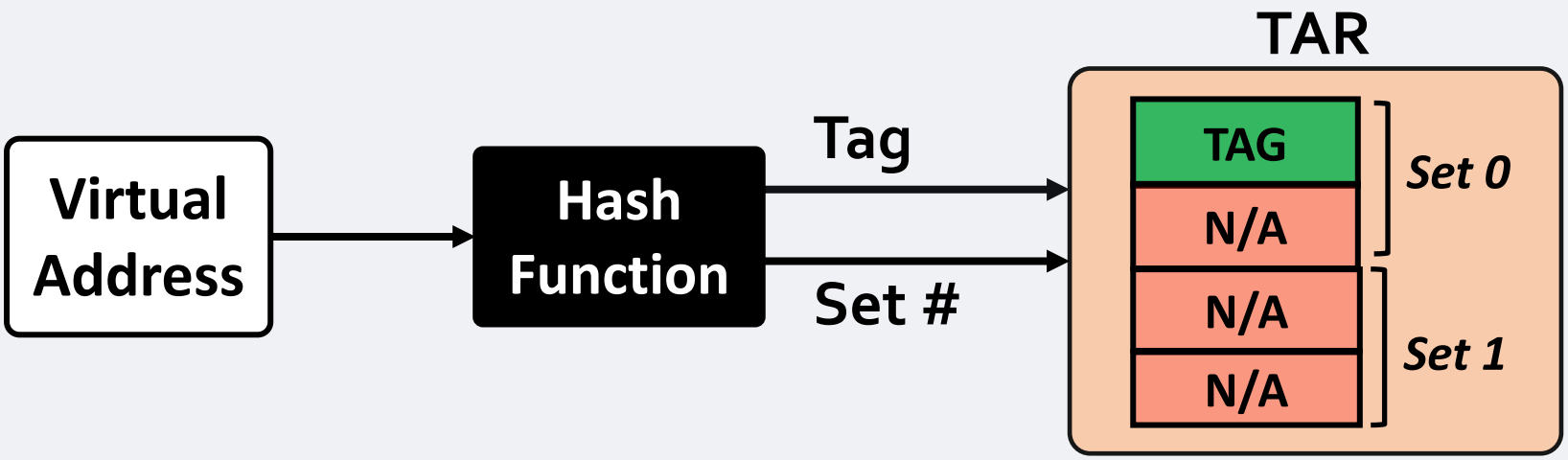
RSW Operations

1. *Tag Matching*

2. *Set Filtering*

RestSeg: Tag Matching

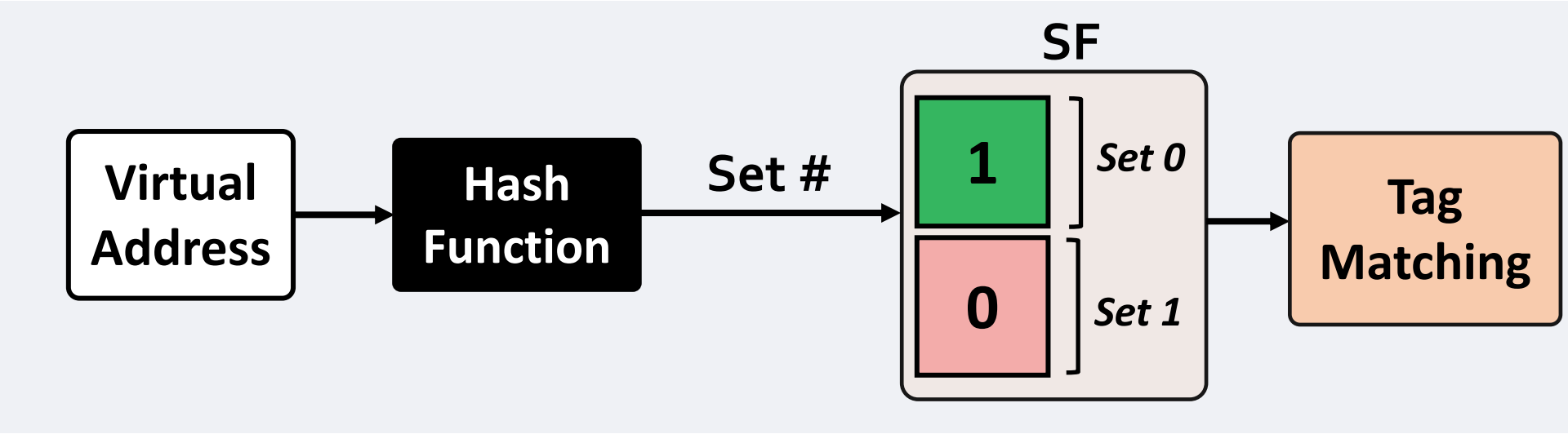
- **Tag matching** requires comparing the tags of all ways with the tag of the virtual page
- **Tag Array (TAR):** Array that stores the tag of each entry



Do we always have to do tag matching?

RestSeg: Set Filtering

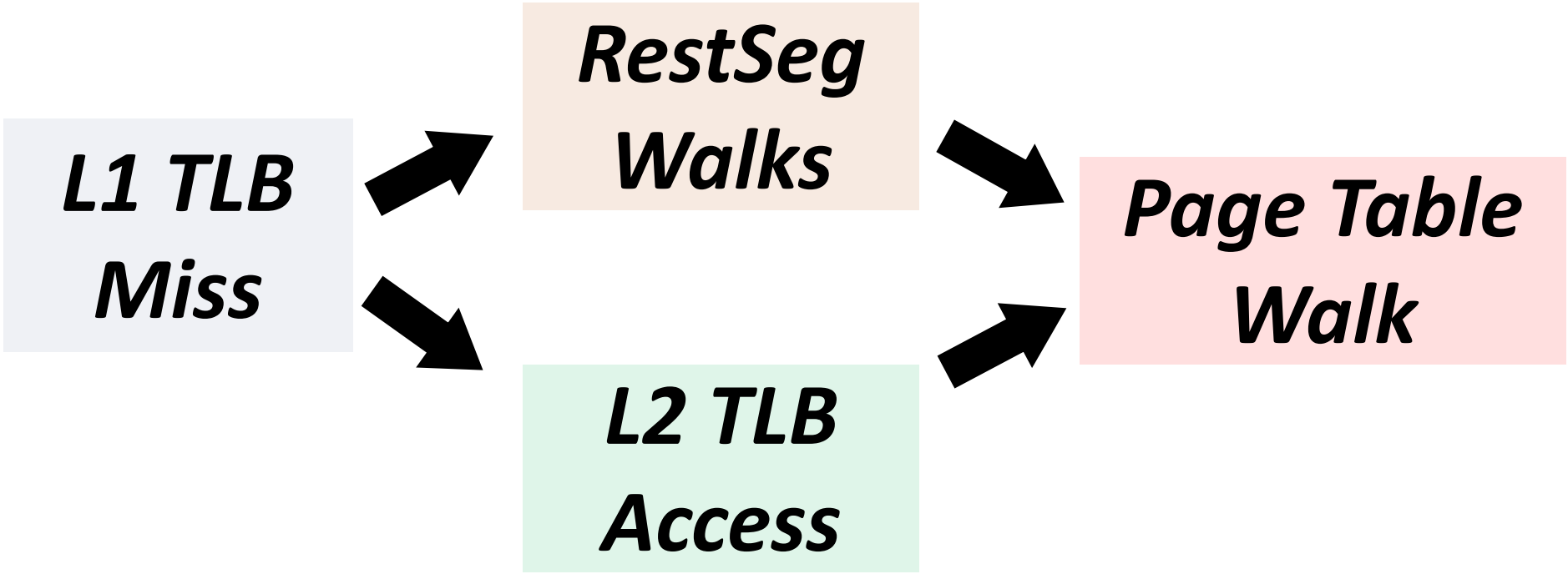
- **Set Filtering:** quickly discover if a set in the RestSeg is empty or not and filter tag mismatches
- **Set Filter (SF)**: Array of counters that keep track of the number of pages inside each set



Address Translation in Utopia

System employs:

- 2 RestSegs, one for 4KB and one for 2MB pages
- 1 FlexSeg



Talk Outline

Virtual Memory Background

Address Translation Overheads

Utopia: Hybrid Address Mappings

Utopia: Key Challenges

Evaluation Results

Utopia: Key Challenges

1. Which data should be placed in the RestSeg?

2. How to maintain RestSegs in the system

3. How to integrate Utopia in the address translation pipeline

Utopia: Key Challenges

1. **Which data should be placed in the RestSegs?**
2. How to maintain RestSegs in the system
3. How to integrate Utopia in the address translation pipeline

Page Placement in RestSeg

Our **goal** is to place costly-to-translate pages into a RestSeg

We propose two techniques to perform data placement in Utopia:

- **Page-Fault-based Allocation Policy**
- **PTW-Tracking-based Migration Policy**

Page-Fault-Based Allocation Policy

On a page fault

the page is directly allocated in a RestSeg

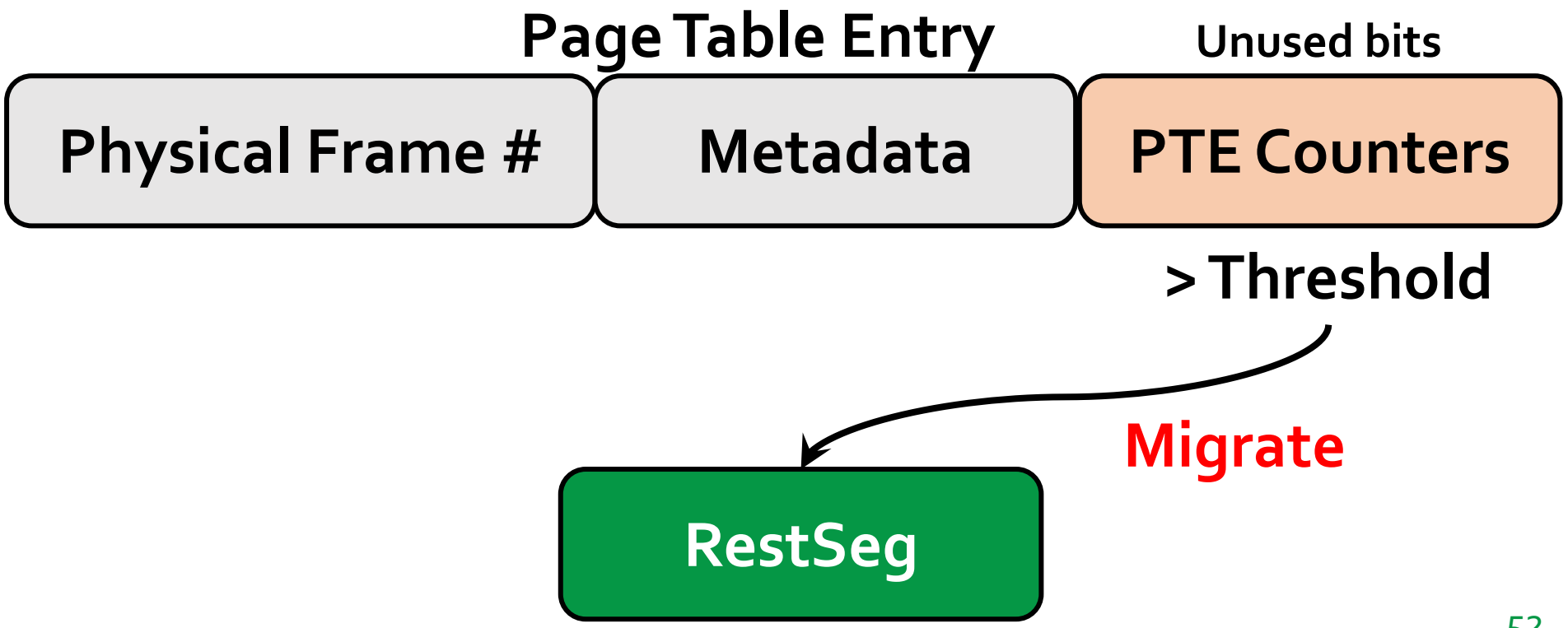
Page Fault

What about costly-to-translate pages that reside in a FlexSeg?

RestSeg

PTW-Tracking-Based Migration Policy

Use unused bits of each PTE as a counter that tracks the number and cost of PTWs for each page



Utopia: Three Key Challenges

1. Which data should be placed in the RestSegs?

2. **How to maintain RestSegs in the system**

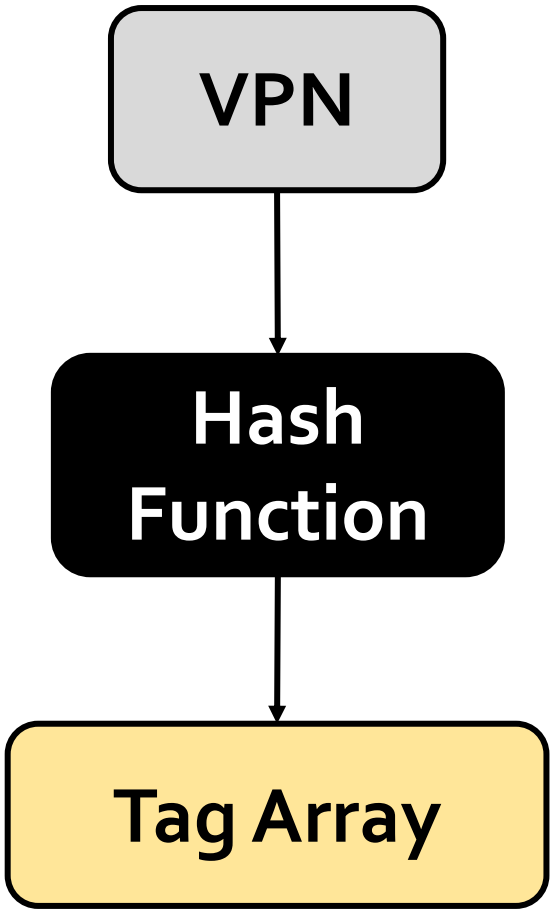
3. How to integrate Utopia in the address translation pipeline

OS Support for Utopia

OS supports Utopia in three ways by handling:

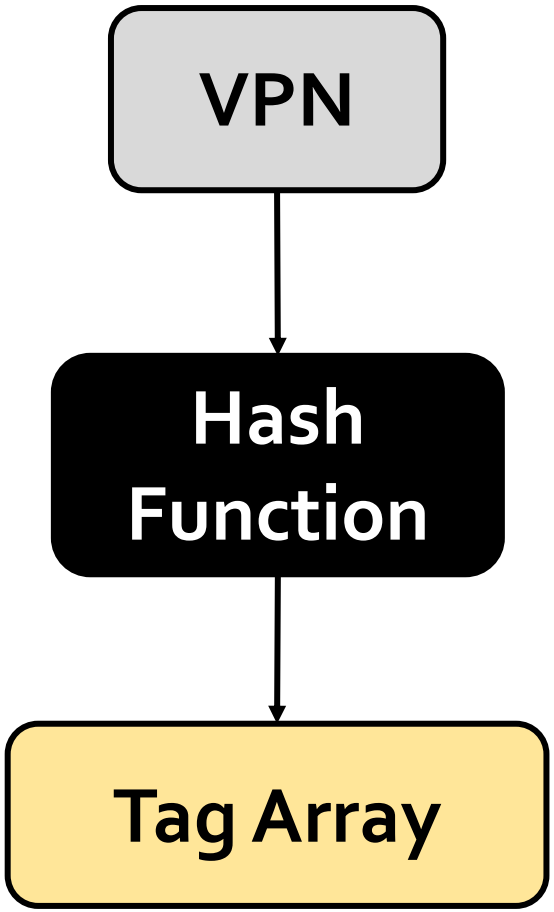
1. **Allocations** in a RestSeg
2. **Replacements** in a RestSeg
3. **Migrations** to/from a RestSeg

OS: Allocation in RestSeg



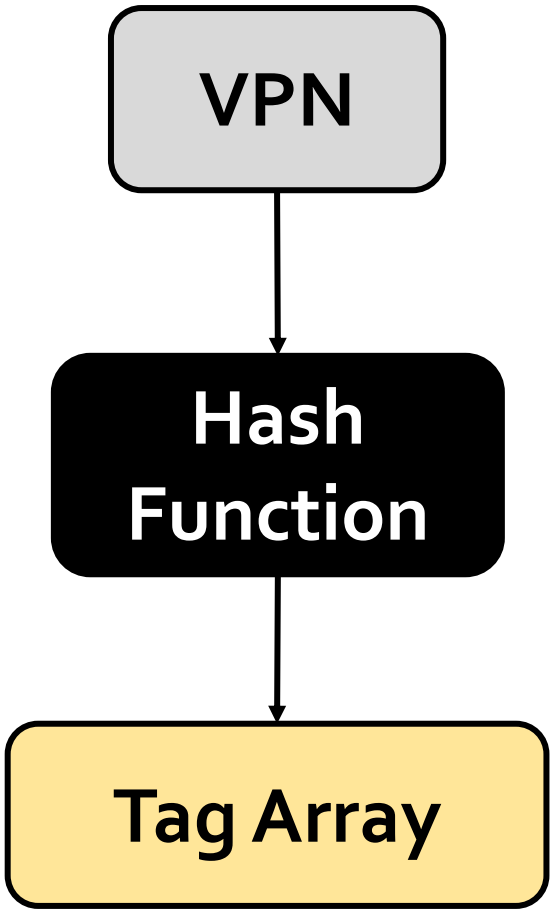
There is a free way!
Store data there

OS: Allocation in RestSeg



Discover if there is a free way in the set

OS: Replacement in RestSeg

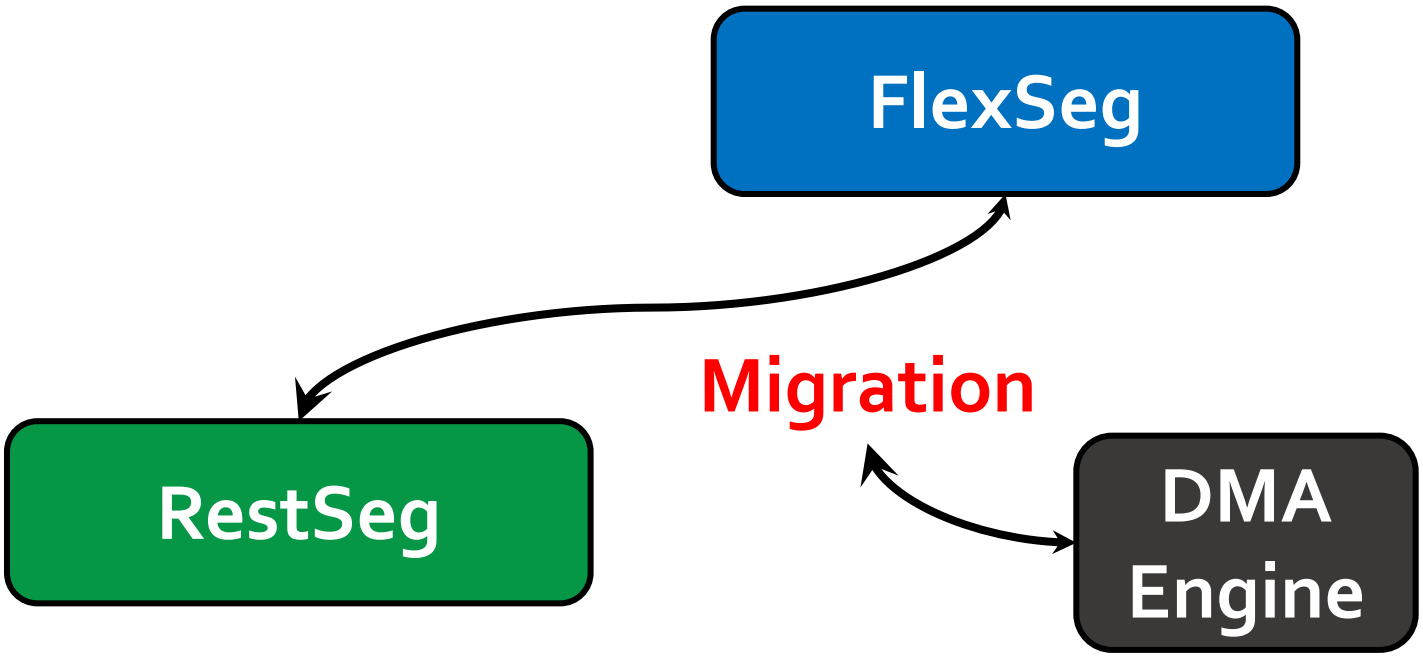


No way is free!

If no page-fault, evict a page!

OS: Migration to/from RestSeg

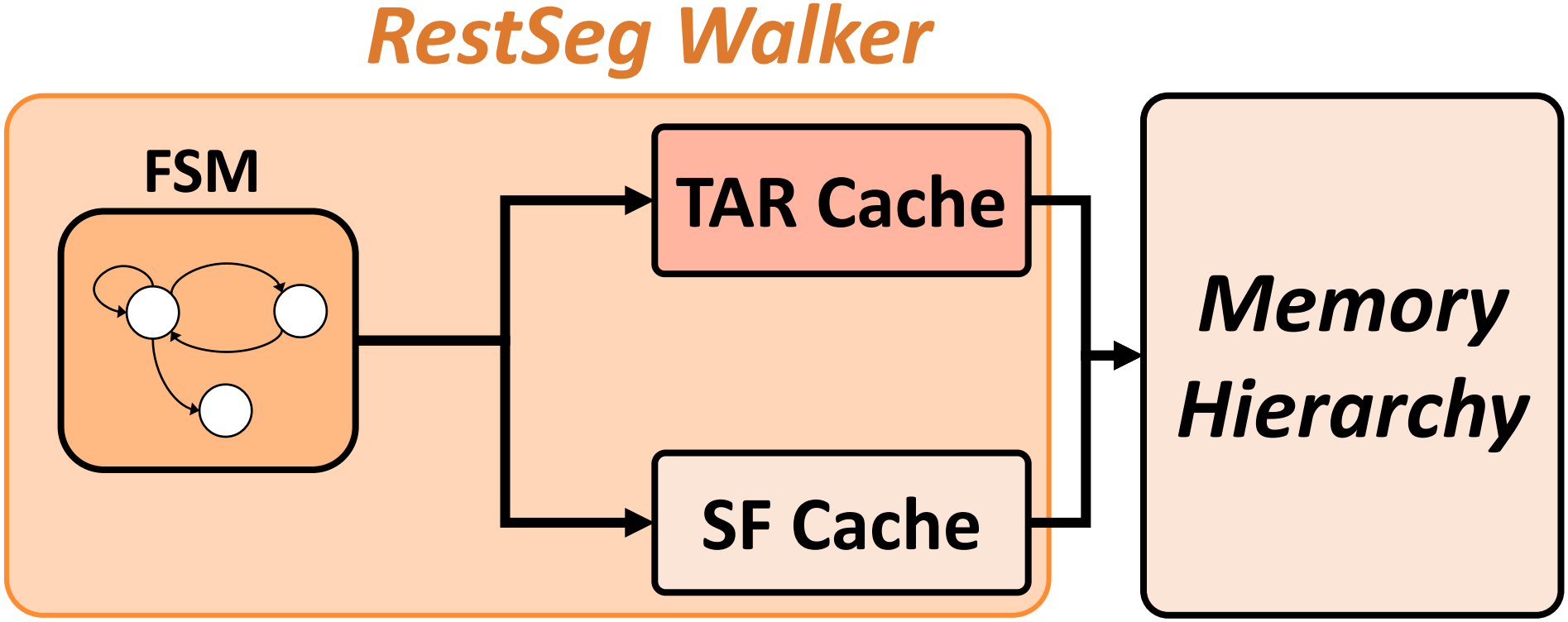
DMA engine is responsible for migrating data between RestSegs and FlexSegs



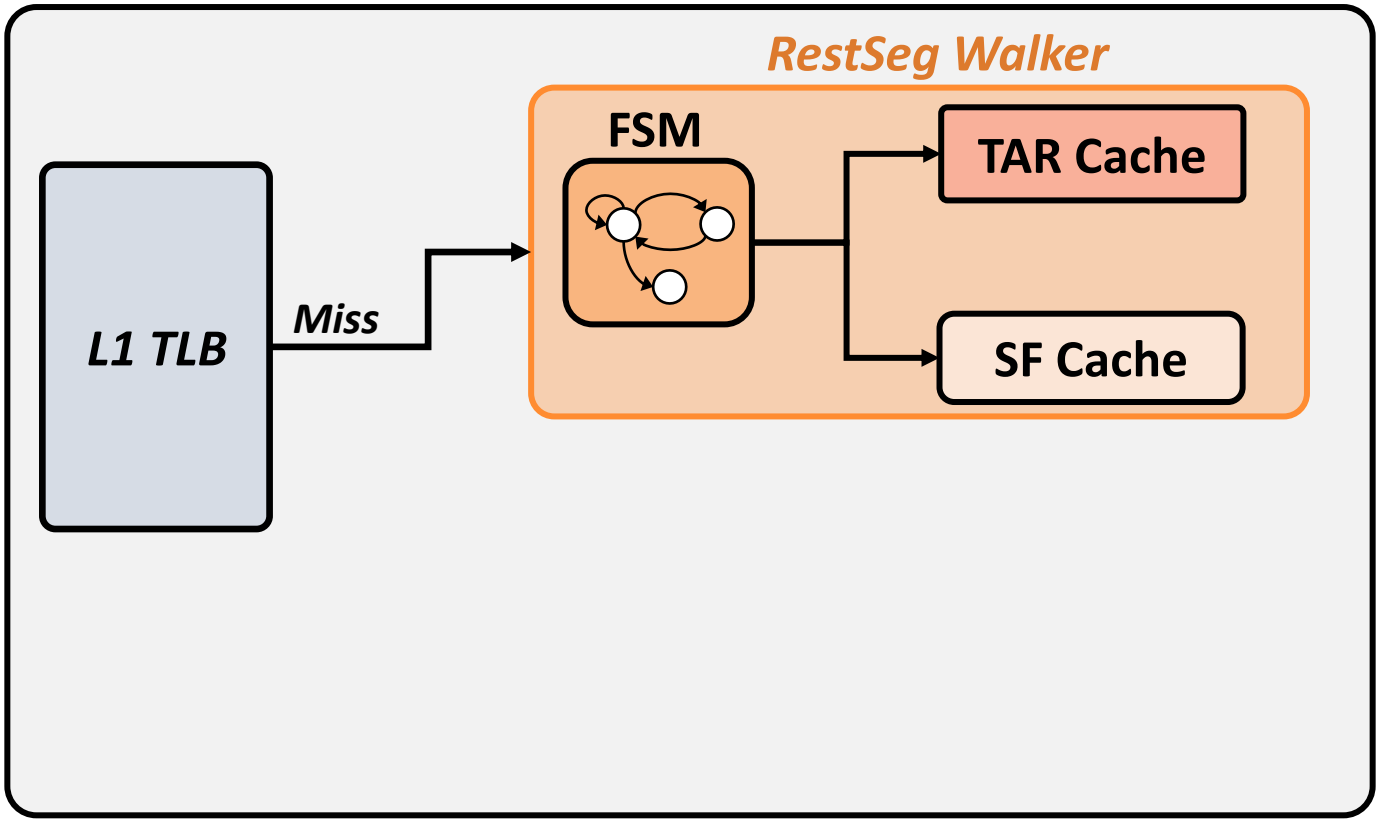
Utopia: 3 Key Challenges

- Which data should be placed in the RestSegs?
- How to maintain RestSegs in the system
- **How to integrate Utopia in the address translation pipeline**

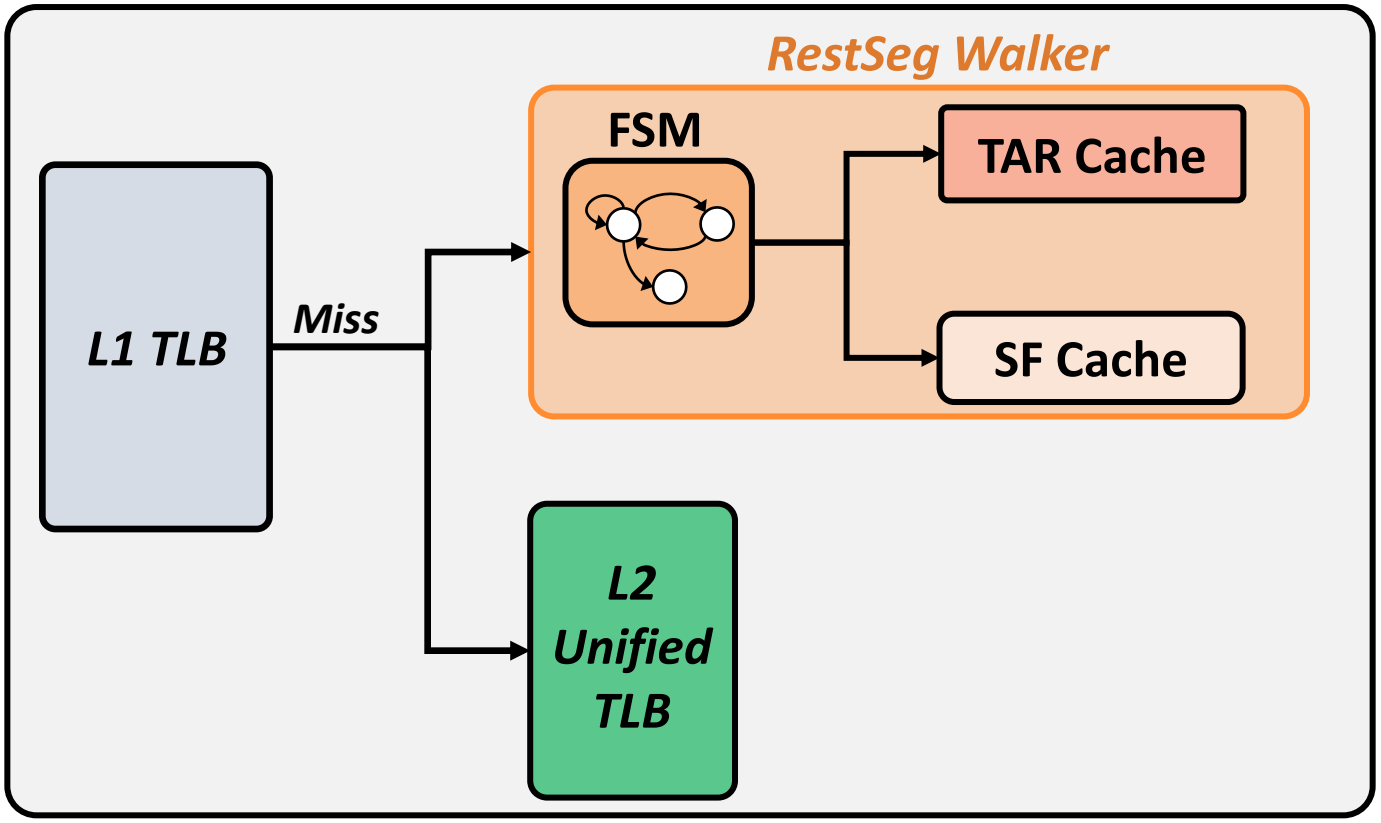
RestSeg Walker in MMU



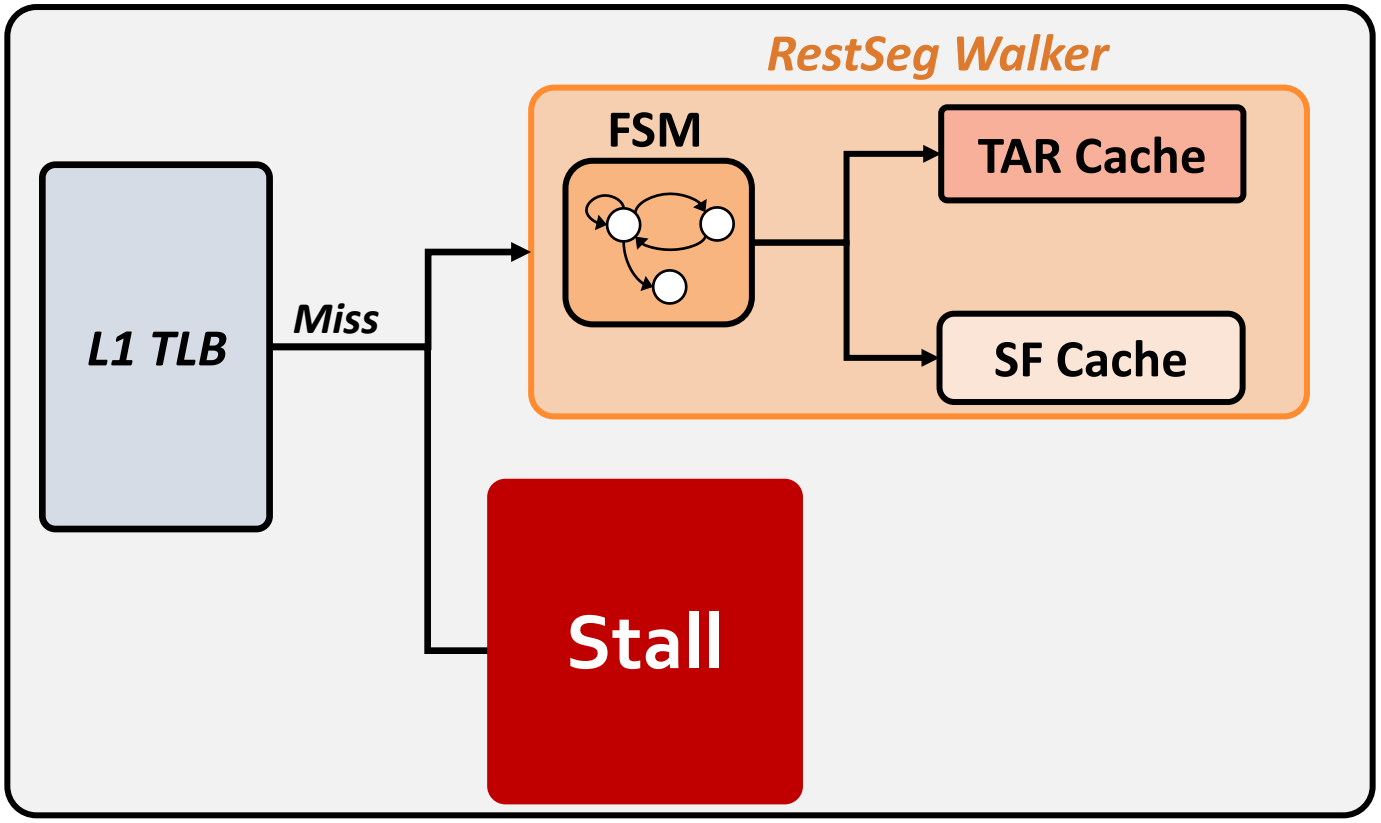
MMU: Page inside RestSeg (I)



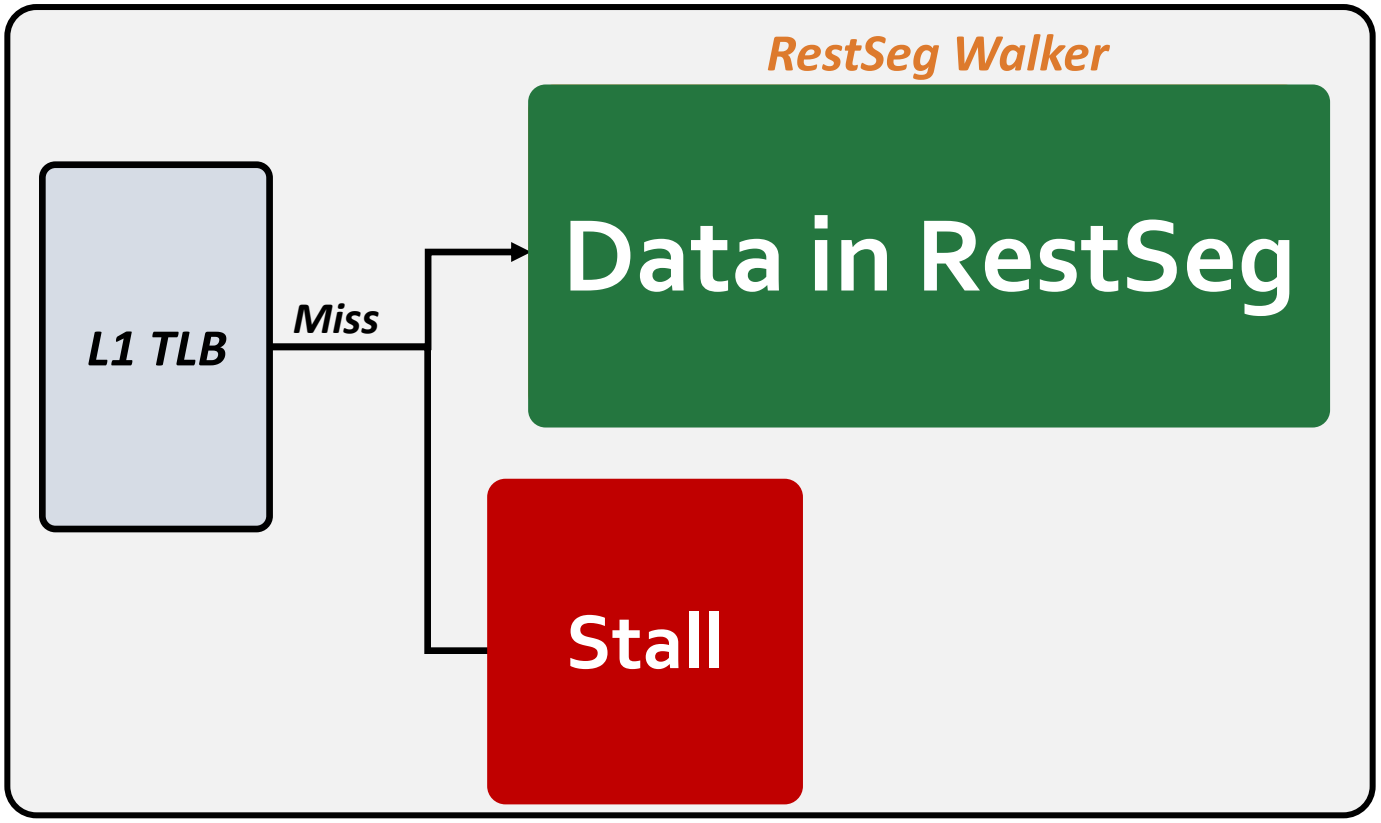
MMU: Page inside RestSeg (I)



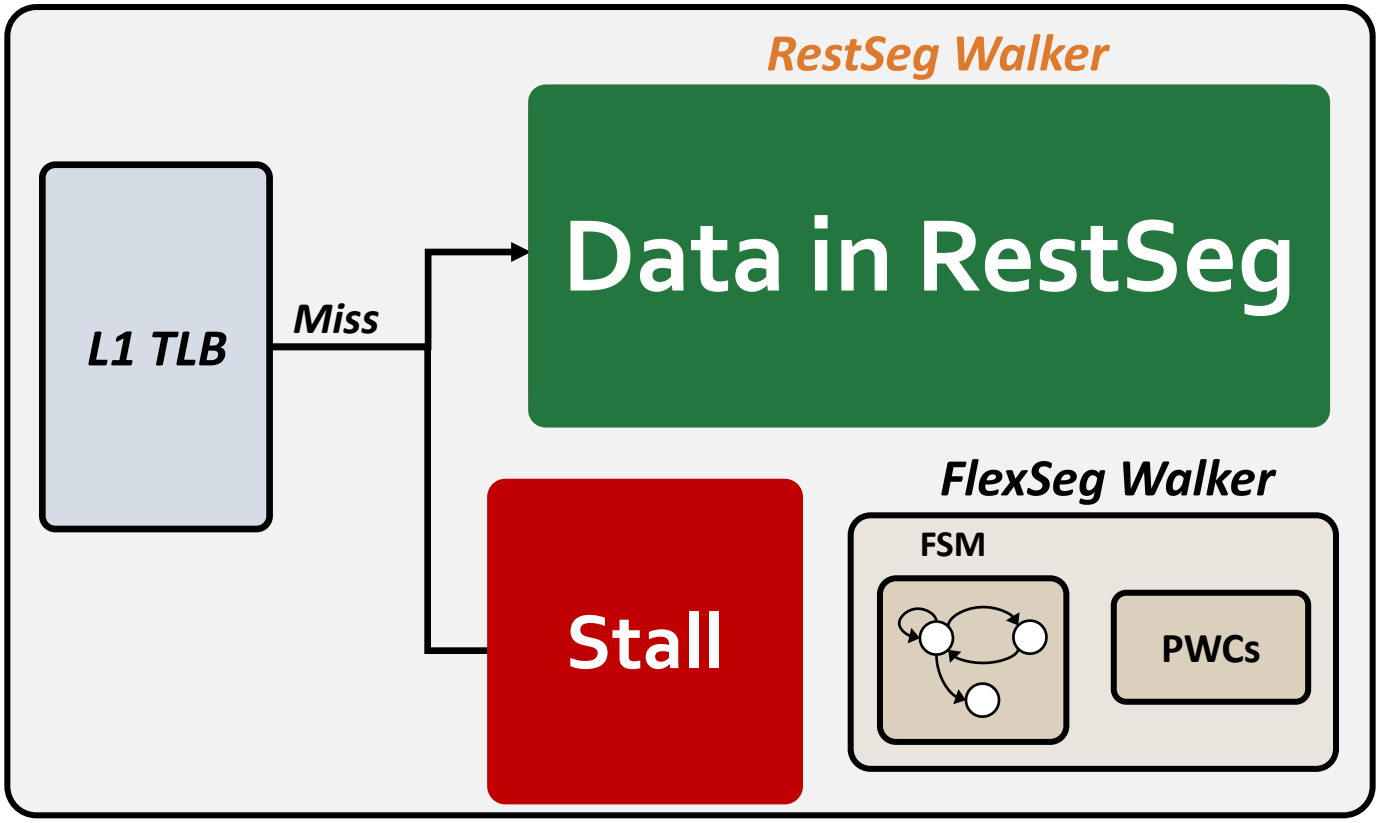
MMU: Page inside RestSeg (I)



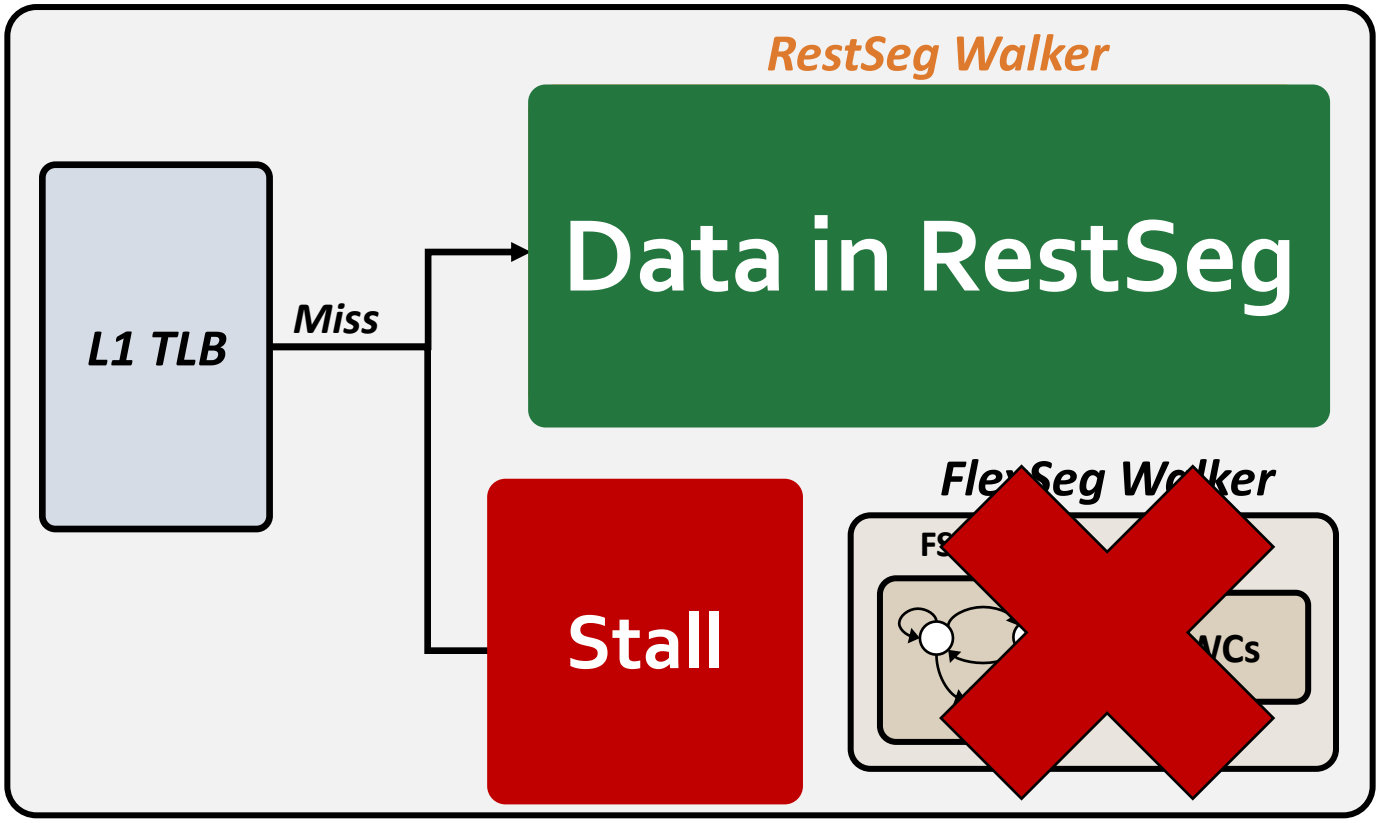
MMU: Page inside RestSeg (I)



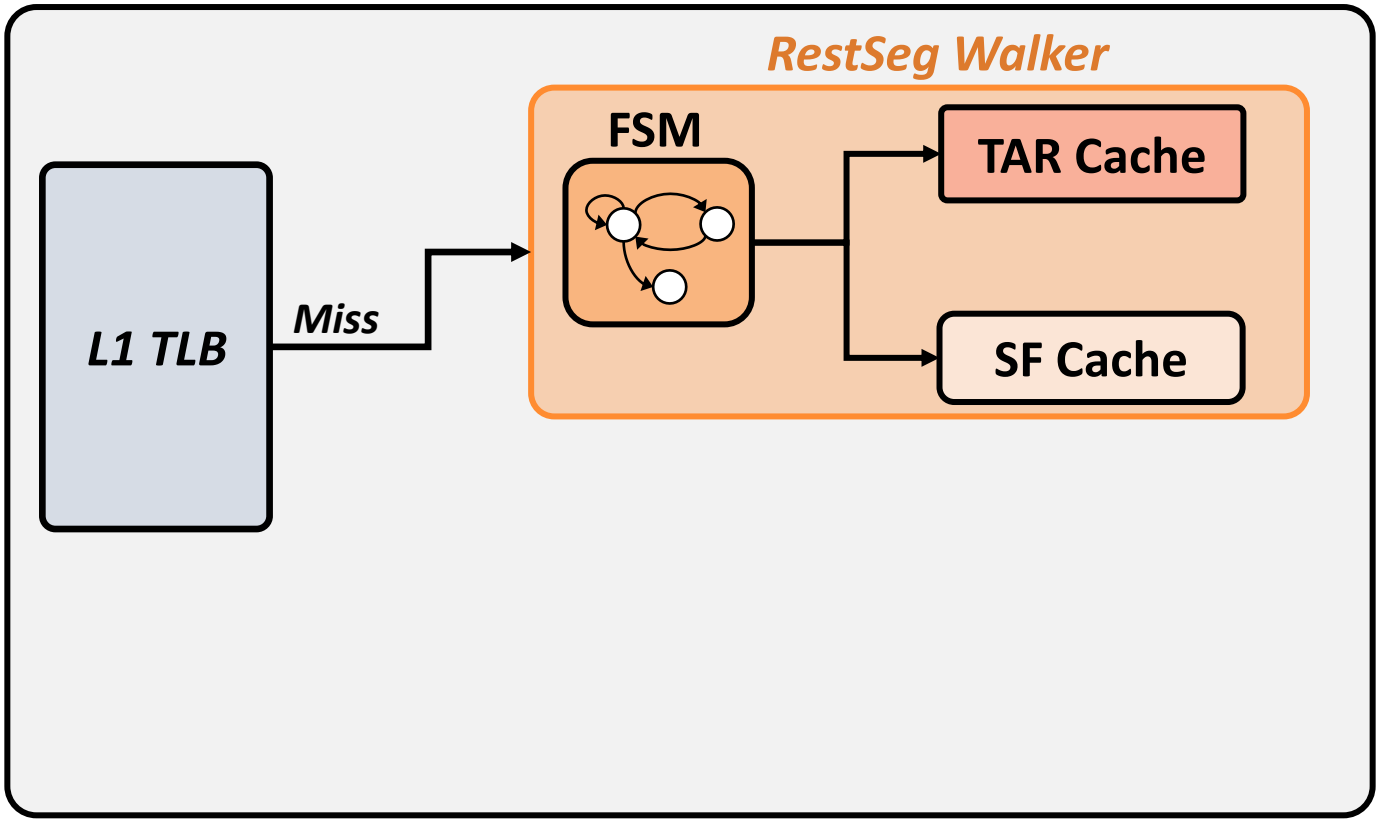
MMU: Page inside RestSeg (I)



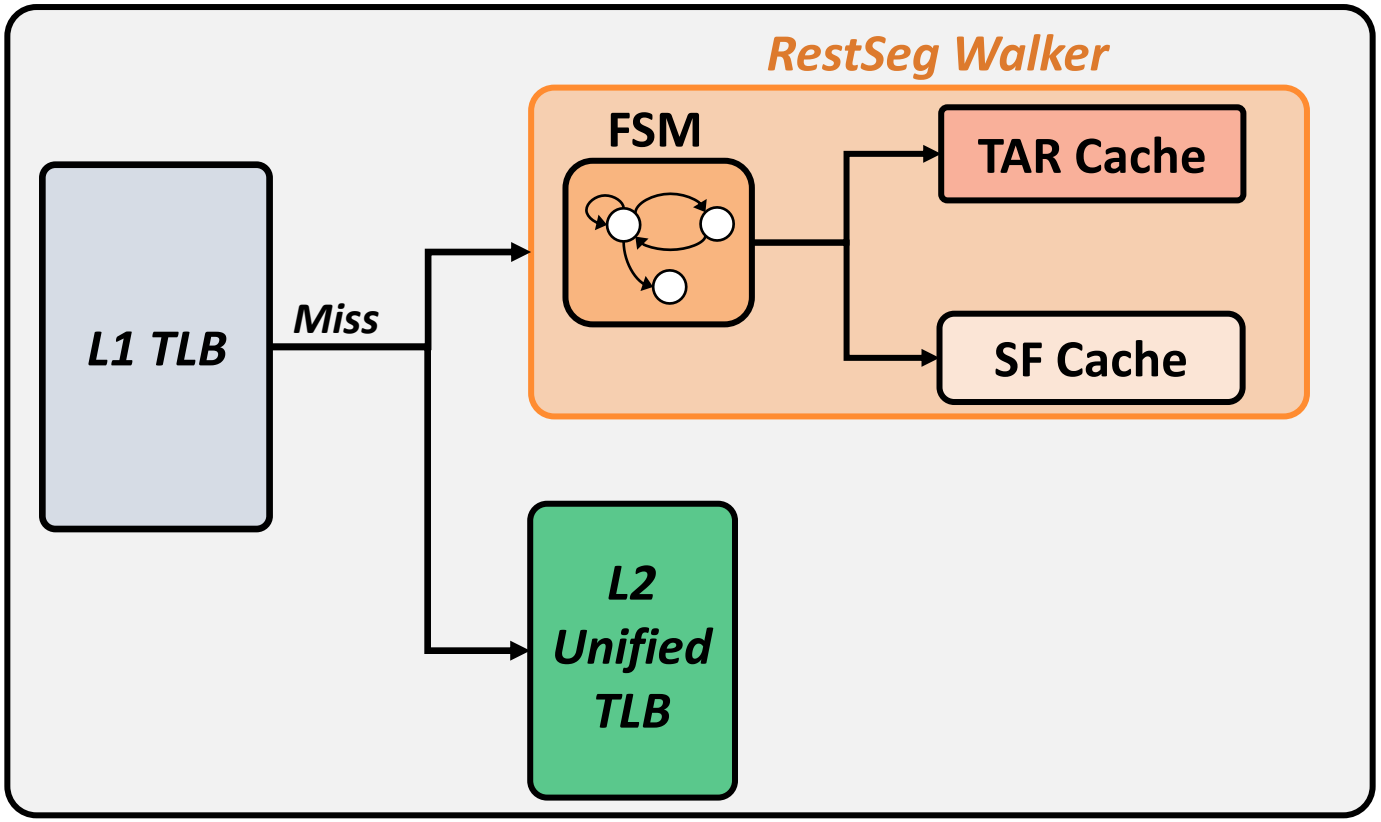
MMU: Page inside RestSeg (I)



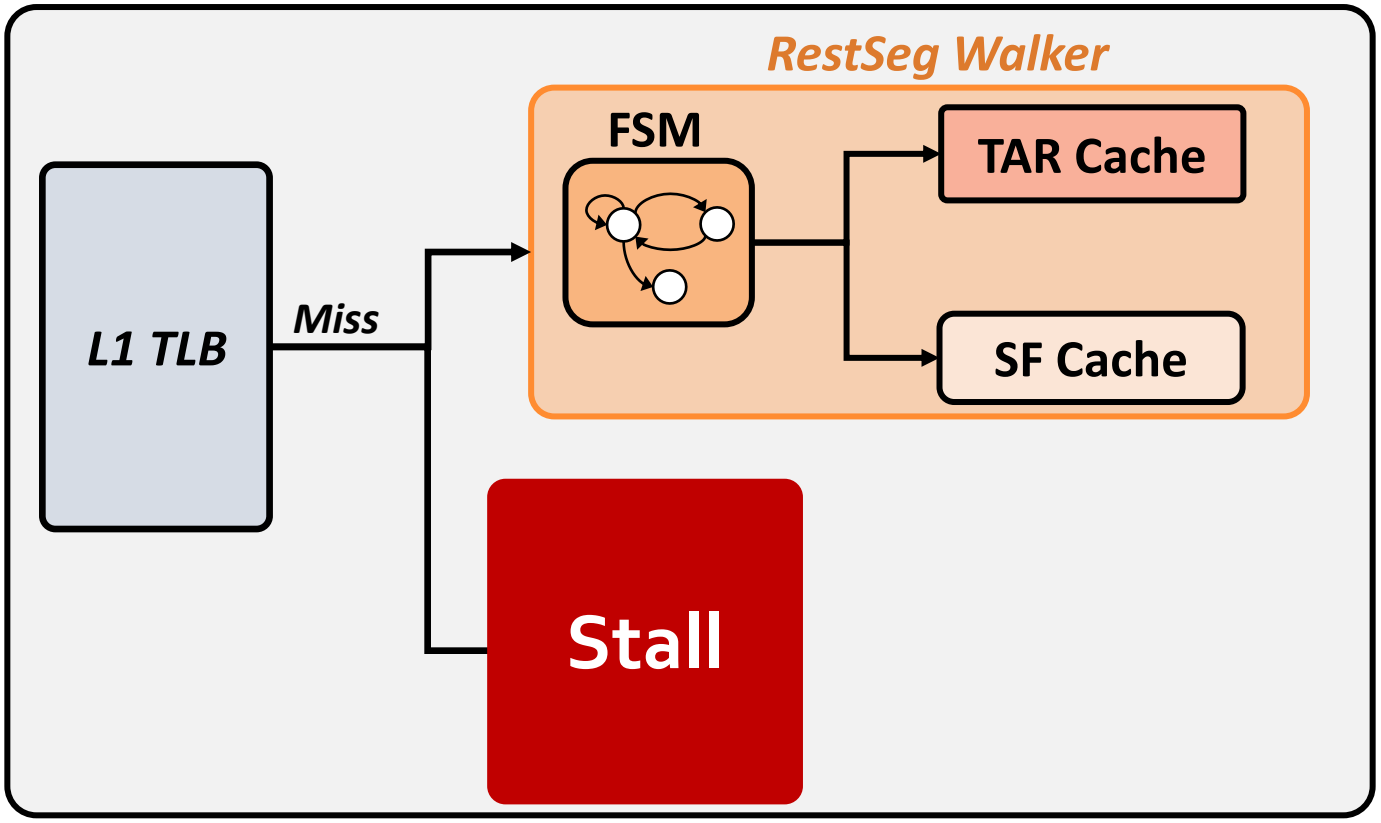
MMU: Page inside FlexSeg (II)



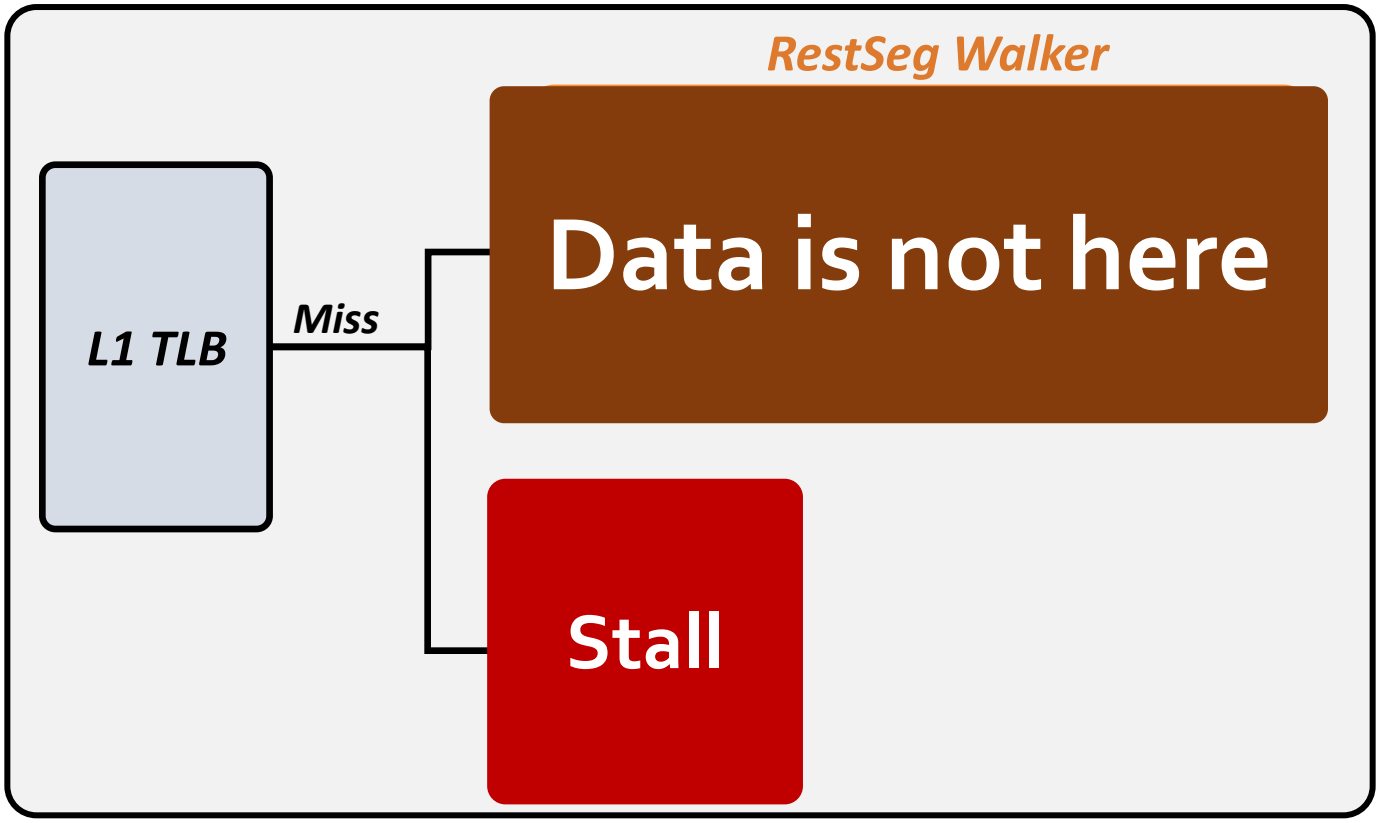
MMU: Page inside FlexSeg (II)



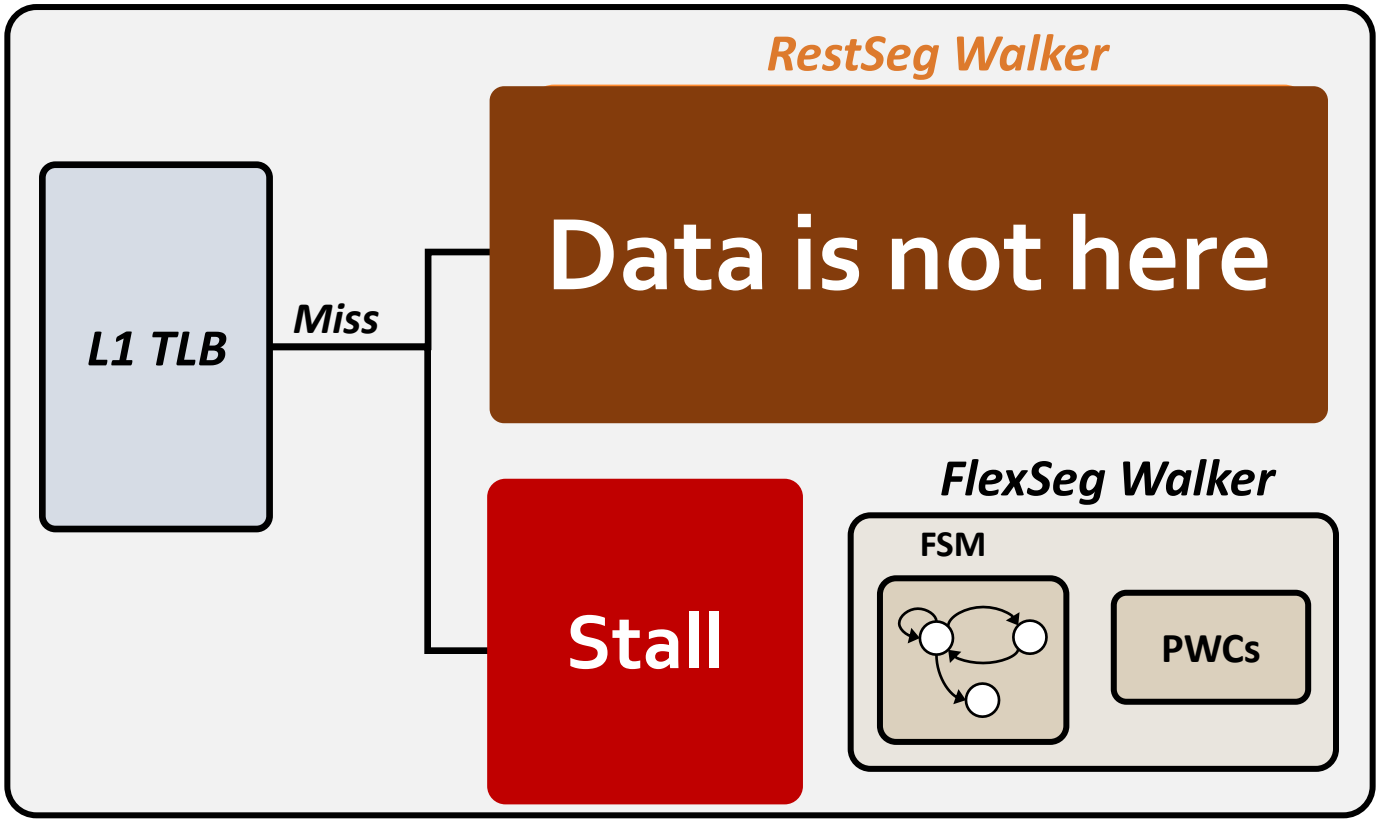
MMU: Page inside FlexSeg (II)



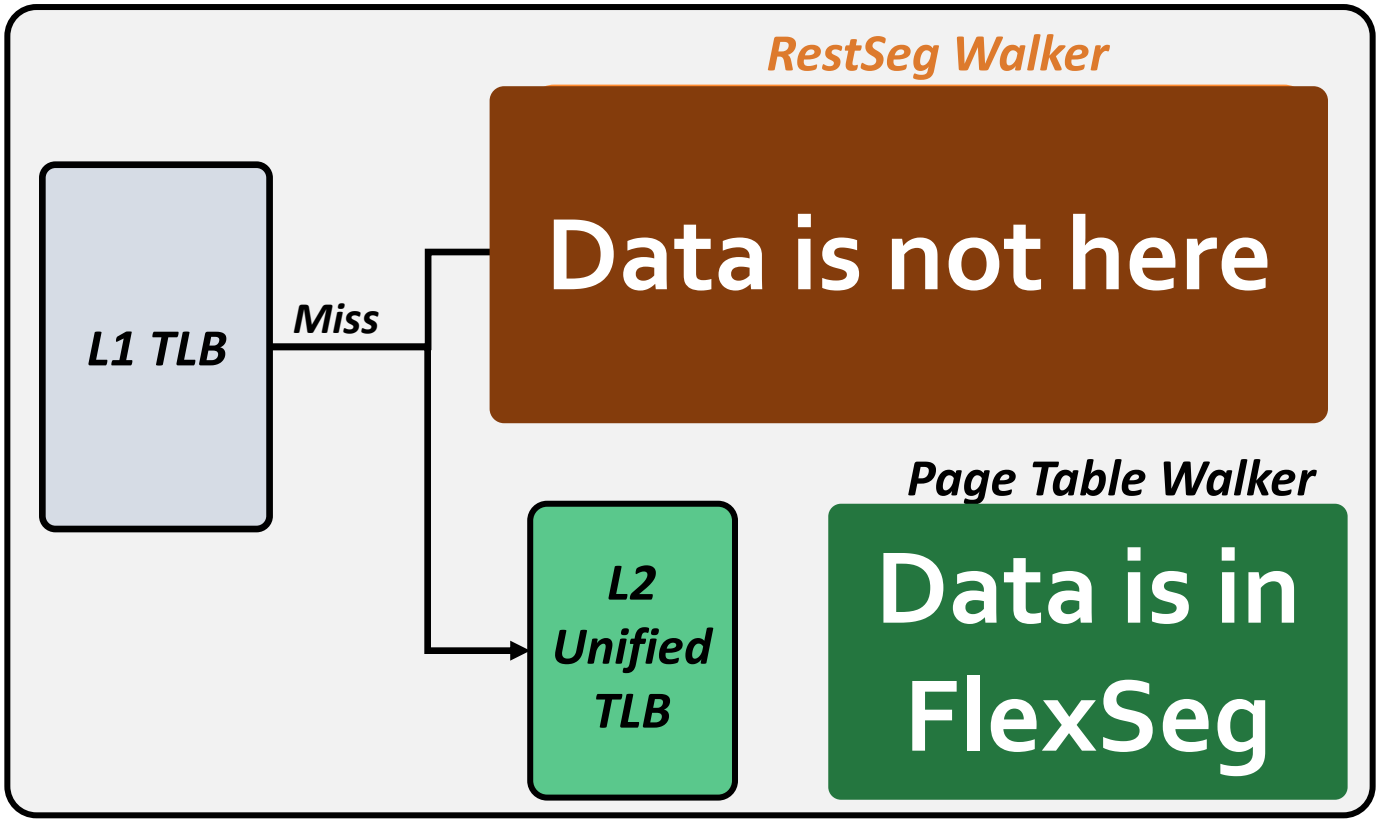
MMU: Page inside FlexSeg (II)



MMU: Page inside FlexSeg (II)



MMU: Page inside FlexSeg (II)



Area & Power Overhead

- Area and power overhead evaluation using McPat
- Comparison to a high-end Intel Raptor Lake

Utopia incurs 0.64% area and 0.72% power overhead per core

Talk Outline

Virtual Memory Background

Address Translation Overheads

Utopia: Hybrid Address Mappings

Utopia: Key Challenges

Evaluation Results

Evaluation Methodology

- Sniper Multicore Simulator extended with:
  - Page table walker & page walk caches
  - Buddy allocator
  - Migration Latency

Our poster at MICRO 2023 SRC introduces a new open-source simulation framework for VM research

https://github.com/CMU-SAFARI/Virtuoso

Evaluation Methodology

- Sniper Multicore Simulator extended with:
  - Page table walker & page walk caches
  - Buddy allocator
  - Migration Latency

- Workloads**: Executed for 500M instructions**

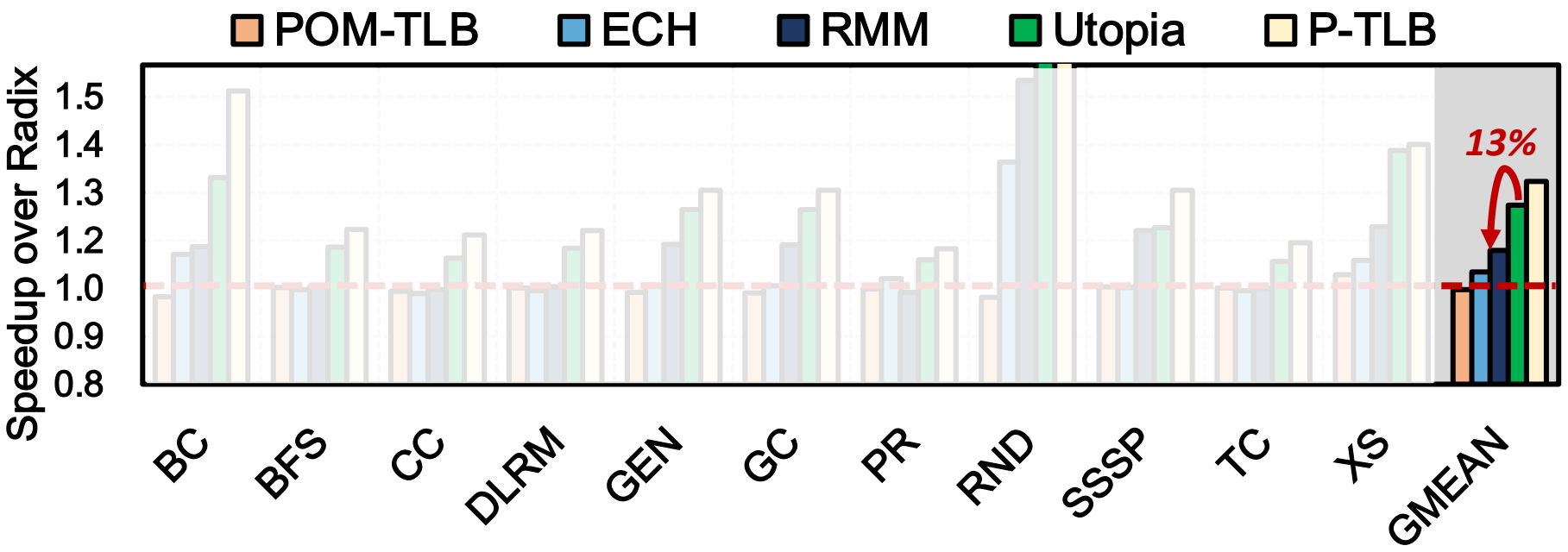
  - GraphBIG: PR, BFS, BC, GC, CC
  - HPCC: Randacc
  - XSBench: Particle Simulation with 15K grid
  - DLRM: SLS-like
  - GenomicsBench: k-mer count

Evaluated Mechanisms

- Baseline with **Radix PT** and **Transparent Huge Pages** enabled
- **POM-TLB**[1]: State-of-the art large **software-managed TLB**
- **ECH**[2]: State-of-the-art **hash-based page table**
- **RMM**[3]**: Contiguity-aware** address translation
- **Utopia:** 512MB RestSegs (one for 4KB and one for 2MB pages)
- **P-TLB**: Perfect L1 TLB (translation requests always hit in L1 TLB)

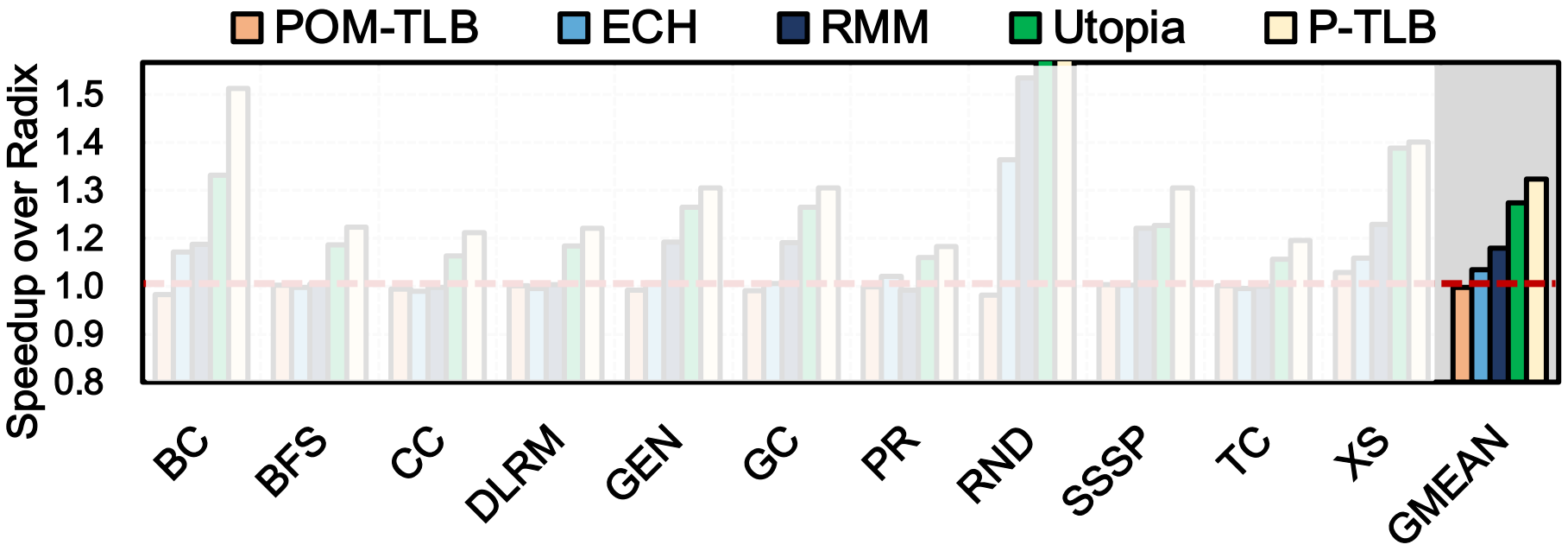
[1] Ryoo et al. "Rethinking TLB designs in virtualized environments: A very large part-of-memory TLB" ISCA '17
[2] Skarlatos et al. "Elastic Cuckoo Page Tables: Rethinking Virtual Memory Translation for Parallelism" ASPLOS '20
[3] Karakostas et al. "Redundant memory mappings for fast access to large memories" ISCA '15

Performance Results



Utopia outperforms the second-best performing scheme (RMM) by 13% and Radix by 24%

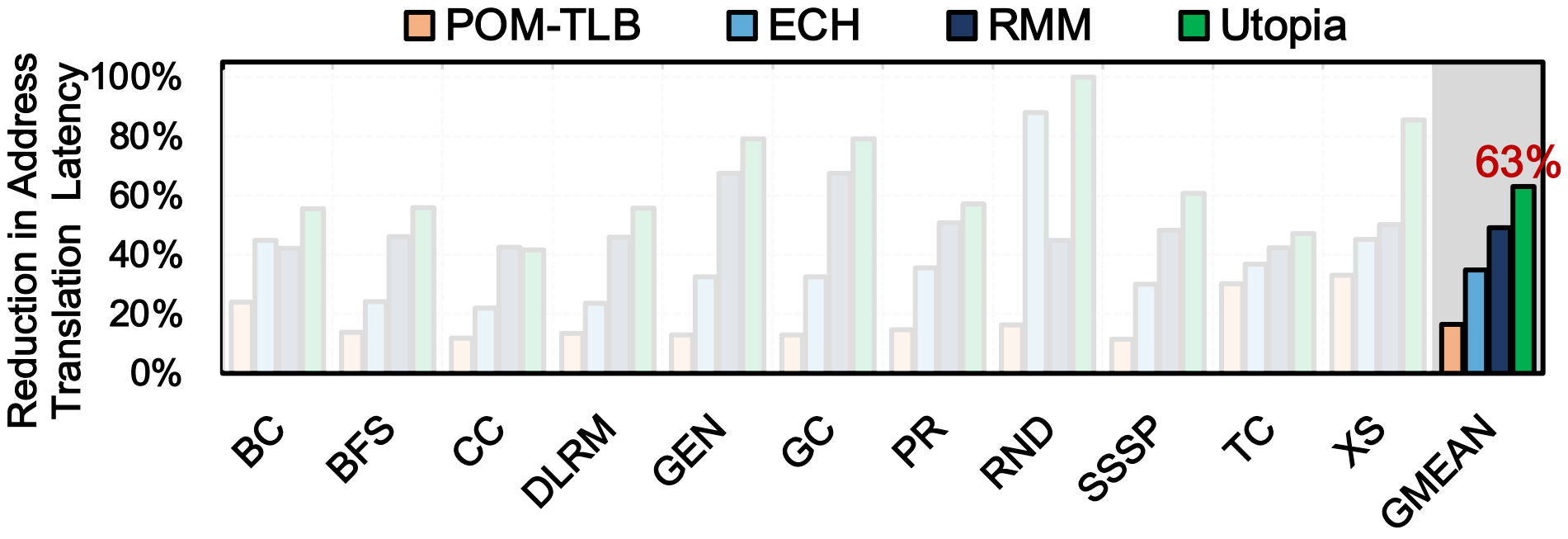
Performance Results



Utopia outperforms the second-best performing scheme (RMM) by 13% and Radix by 24%

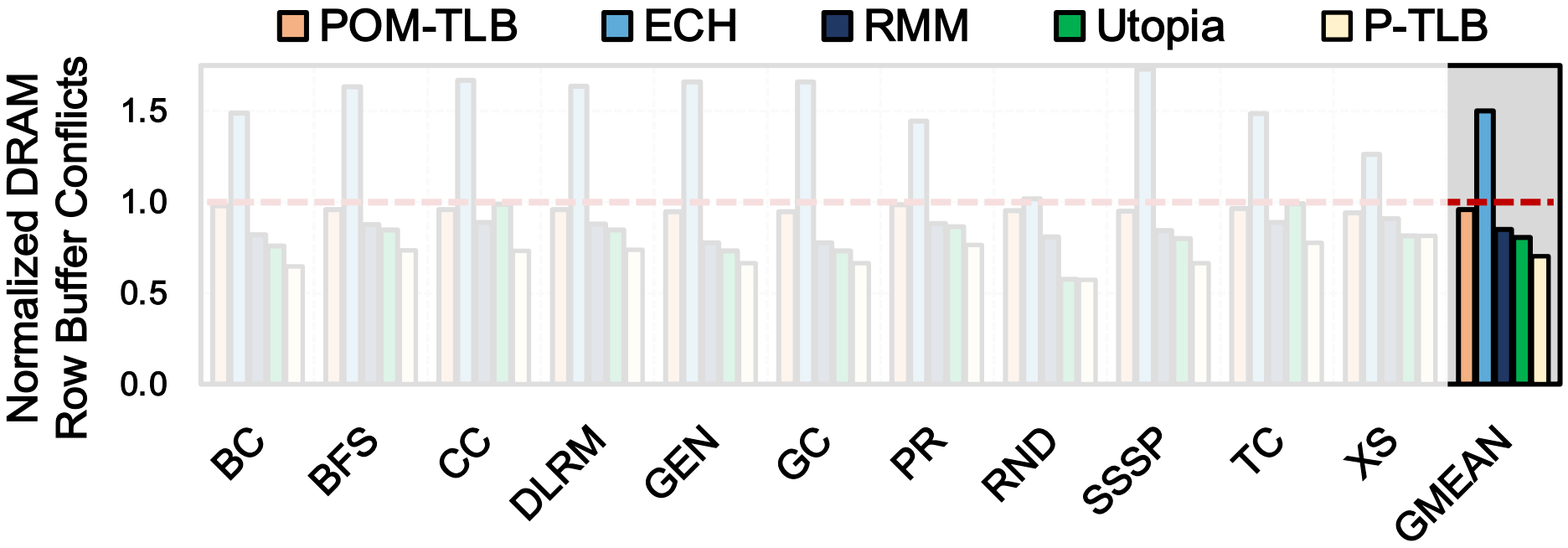
Utopia's performance is within 95% of P-TLB

Translation Latency



Utopia reduces average translation latency by 63% over Radix and 14% over RMM

Main Memory Interference



Utopia reduces row buffer conflicts by 20% over Radix and 9% less than P-TLB

More Results and Details in the Paper

- Sensitivity across different hash functions
- Sensitivity to parallel/serial TLB access and RSW
- Sensitivity to RestSeg size
- Reuse-level distribution of 4KB pages
- Effect of migration to memory requests
- TAR & SF cache hit rate
- Overhead across different context-switch quanta

https://arxiv.org/abs/2211.12205

More Results and Details in the Paper

- Se
- Se
- Se
- Re
- Ef
- TA
- Ov

RSW

uanta

Utopia: Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings

Konstantinos Kanellopoulos[1] Rahul Bera[1] Kosta Stojiljkovic[1] Nisa Bostanci[1] Can Firtina[1]
Rachata Ausavarungnirun[2] Rakesh Kumar[3] Nastaran Hajinazar[4] Mohammad Sadrosadati[1]
Nandita Vijaykumar[5] Onur Mutlu[1]

[1]ETH Zürich [2]King Mongkut's University of Technology North Bangkok
[3]Norwegian University of Science and Technology [4]Intel Labs [5]University of Toronto

Abstract

Conventional virtual memory (VM) frameworks enable a virtual address to flexibly map to *any* physical address. This flexibility necessitates large data structures to store virtual-to-physical mappings, which leads to high address translation latency and large translation-induced interference in the memory hierarchy, especially in data-intensive workloads. On the other hand, restricting the address mapping so that a virtual address can only map to a specific set of physical addresses can significantly reduce address translation overheads by making use of compact and efficient translation structures. However, restricting the address mapping flexibility across the entire main memory severely limits data sharing across different processes and increases data accesses to the swap space of the storage device even in the presence of free memory.

We propose *Utopia*, a new hybrid virtual-to-physical address mapping scheme that allows *both* flexible and restrictive hash-based address mapping schemes to harmoniously *co-exist* in the system. The key idea of Utopia is to manage physical memory using two types of physical memory segments: restrictive segments and flexi-

1 Introduction

Virtual memory (VM) serves as a foundational element in most computing systems, simplifying the programming model by offering an abstraction layer over physical memory [2–24]. In the presence of VM, the operating system (OS) maps each virtual address to its corresponding physical memory address to facilitate application-transparent memory management, process isolation, and memory protection. The virtual-to-physical mapping scheme in conventional VM frameworks allows a virtual address to flexibly map to *any* physical address. This flexibility enables key VM functionalities, such as (i) data sharing between processes while maintaining process isolation and (ii) avoiding frequent swapping (i.e., avoiding storing data in the swap space of the storage device in the presence of free main memory space). However, a flexible mapping scheme requires mapping metadata for every virtual address and its corresponding physical address, which is stored in the page table (PT). As shown in multiple prior works [25–35], data-intensive workloads do not efficiently use translation-dedicated hardware structures and the processor performs frequent PT accesses, i.e., a process called

https://arxiv.org/abs/2211.12205

Conclusion

We propose **Utopia**, a new virtual-to-physical mapping scheme that enables both:

Restrictive Mapping **Flexible Mapping**

Harmoniously co-exist in the system

Utopia achieves (i) **13% higher performance** than the state-of-the-art **contiguity-aware translation scheme** and (ii) **95% of the performance** of an ideal **perfect-TLB**

Utopia is open source

https://github.com/CMU-SAFARI/Utopia

Arxiv

UTOPIA

Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings

https://github.com/CMU-SAFARI/Utopia

Github

Konstantinos Kanellopoulos

Rahul Bera, Kosta Stojiljkovic, Nisa Bostanci, Can Firtina, Rachata Ausavarungnirun, Rakesh Kumar, Nastaran Hajinazar, Mohammad Sadrosadati, Nandita Vijaykumar, and Onur Mutlu

SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

UNIVERSITY OF TORONTO

NTNU

High Latency PTWs

Irregular Memory Access Patterns

Large Datasets

Large and costly-to-access PT

Architectural Support for Utopia

New hardware components to support Utopia:

- Specialized hardware circuitry to perform the tag matching and the set filtering
  - Avoid expensive software-based accesses to translation structures
- Small caches for the the Tag Array and Set Filter
  - Accesses to Permissions Filter and Tag Array may exhibit high spatial and temporal locality
- Minor modifications in the address translation pipeline:
  - RSW occurs in parallel with L2 TLB access

VICTIMA

Drastically Increasing Address Translation Reach by Leveraging Underutilized Cache Resources

Presented at MICRO 2023

Konstantinos Kanellopoulos
Hong Chul Nam, Nisa Bostanci, Rahul Bera, Mohammad Sadrosadati,
Rakesh Kumar, Davide Basilio Bartolini and Onur Mutlu

SAFARI
SAFARI Research Group
safari.ethz.ch

ETH zürich

intel

UNIVERSITY OF TORONTO

NTNU

Executive Summary

Problem: Address translation is a major **performance bottleneck** in data-intensive workloads

Large datasets and irregular memory access patterns lead to **frequent L2 TLB misses** (e.g., 20-50 MPKI) and **frequent high-latency** (e.g., 100-150 cycles) page table walks (PTW)

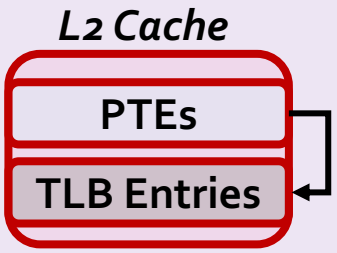
Motivation: Increasing the translation reach (i.e., memory covered by the TLBs) reduces PTWs. However, employing large TLBs leads to increased area, power and latency overheads.

Opportunity: Increase the translation reach of the TLB hierarchy by storing the existing TLB entries within the ***existing cache hierarchy***

Victima: New software-transparent scheme that drastically increases the address translation reach of the processor's TLB hierarchy by leveraging the underutilized cache resources

Key Idea:

Transform L2 cache blocks that store PTEs into blocks that store TLB entries



Key Benefits:

+ **Efficient** in native/virtualized environments
+ **Fully transparent** to application/OS software
+ **Compatible** with huge page schemes

Key Results: Victima (i) **outperforms by 5.1%** a state-of-the-art large TLB design and (ii) achieves **similar performance** to an optimistically fast 128K-entry L2 TLB

https://github.com/CMU-SAFARI/Victima

Talk Outline

Background & Motivation

Opportunity: Leverage Caches

Victima: Overview

Victima: Detailed Design

Evaluation Results

Talk Outline

Background & Motivation

Opportunity: Leverage Caches

Victima: Overview

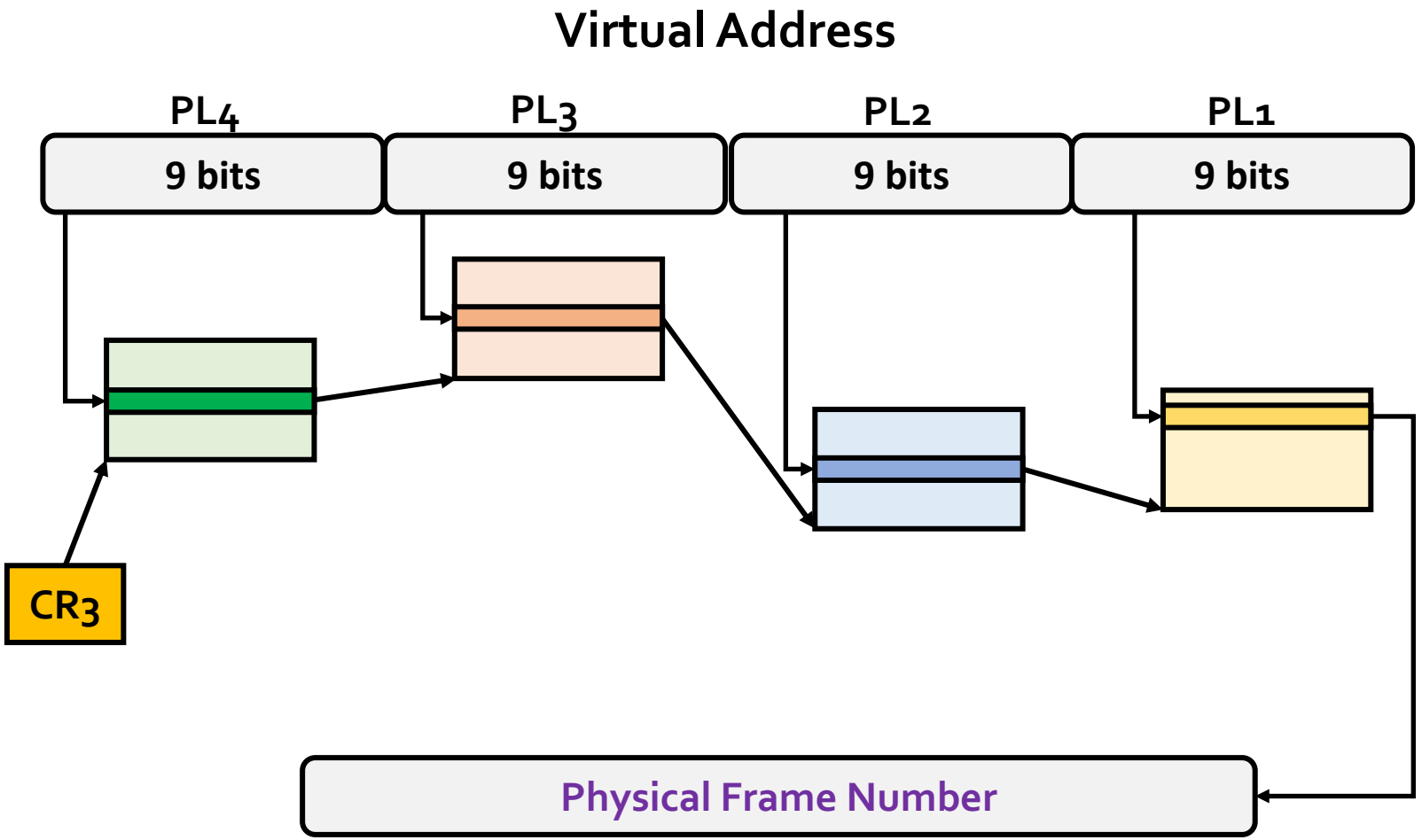
Victima: Detailed Design

Evaluation Results

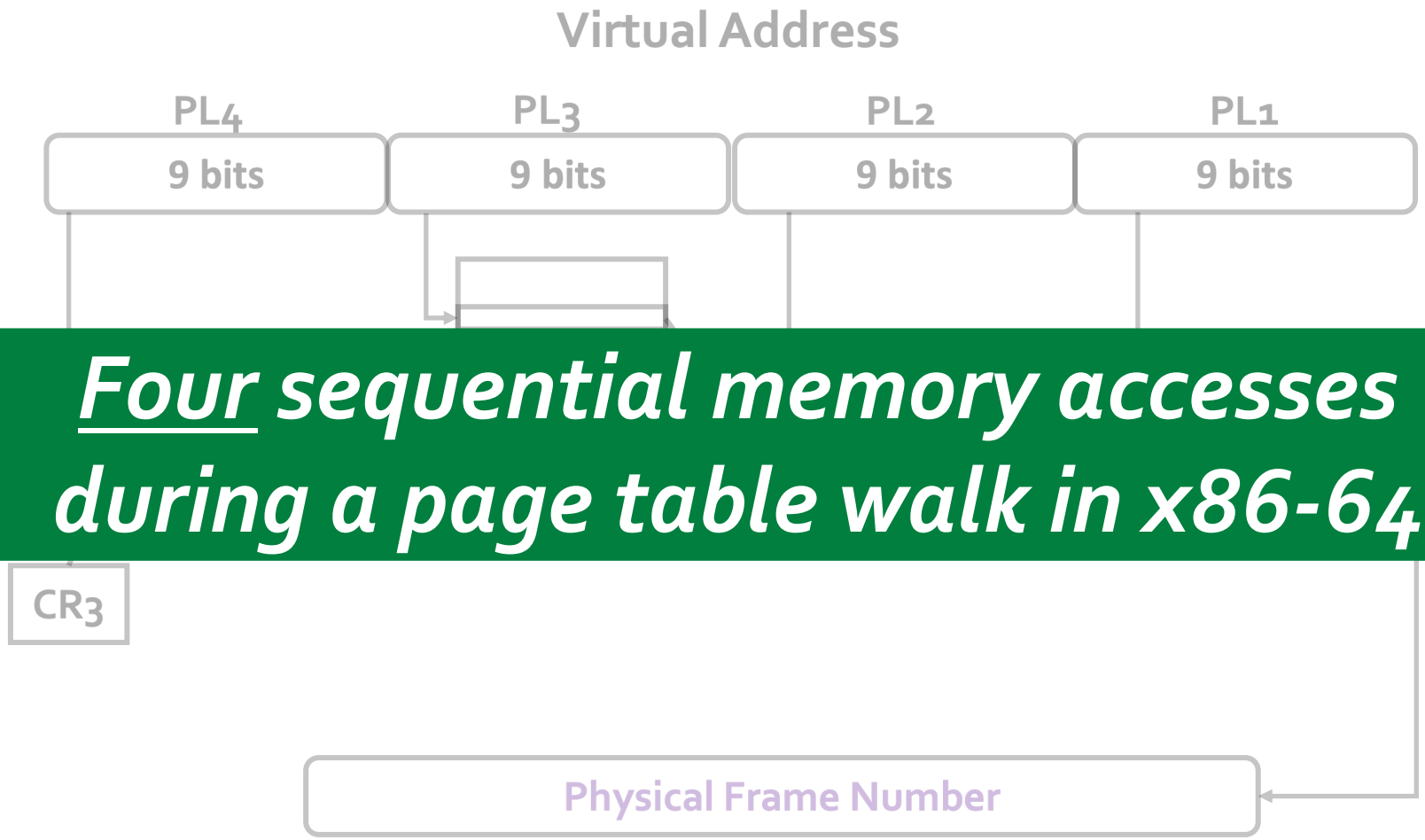
Virtual Memory Basics

- The **Page Table (PT)** stores all virtual-to-physical address mappings
- The x86-64 PT is organized as a **4/5-level radix tree**
- To access the PT, the system performs a **Page Table Walk (PTW)**

Page Table Walk in x86-64

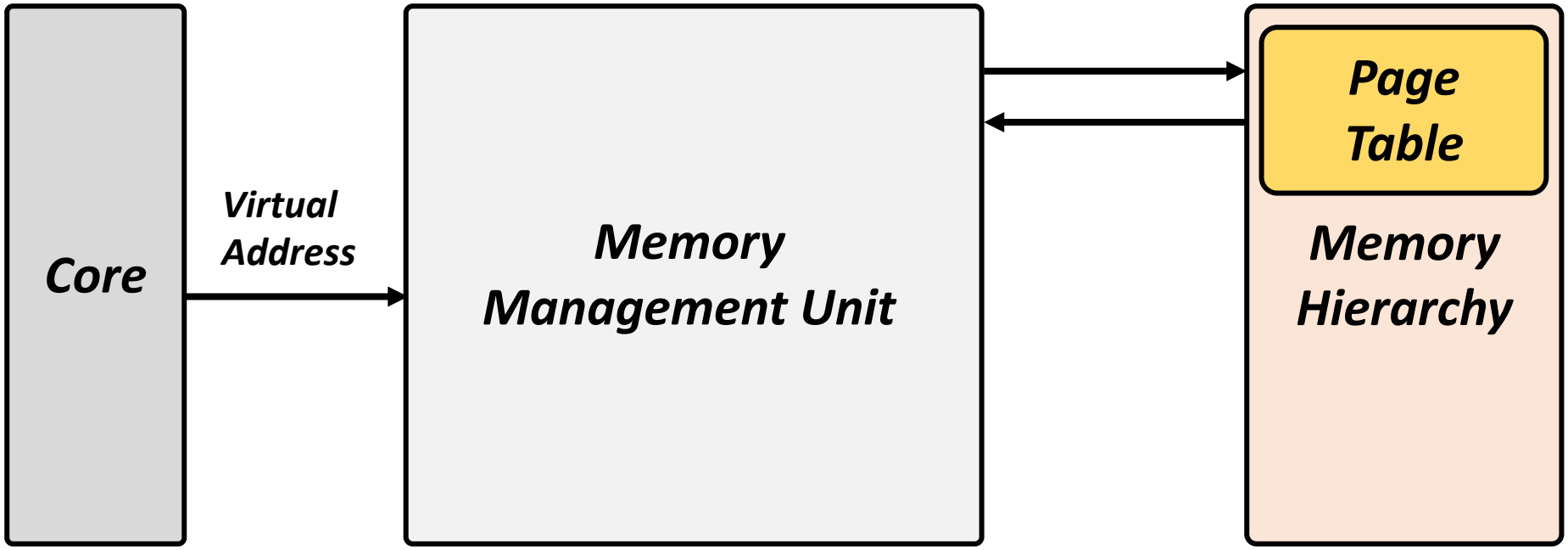


Page Table Walk in x86-64

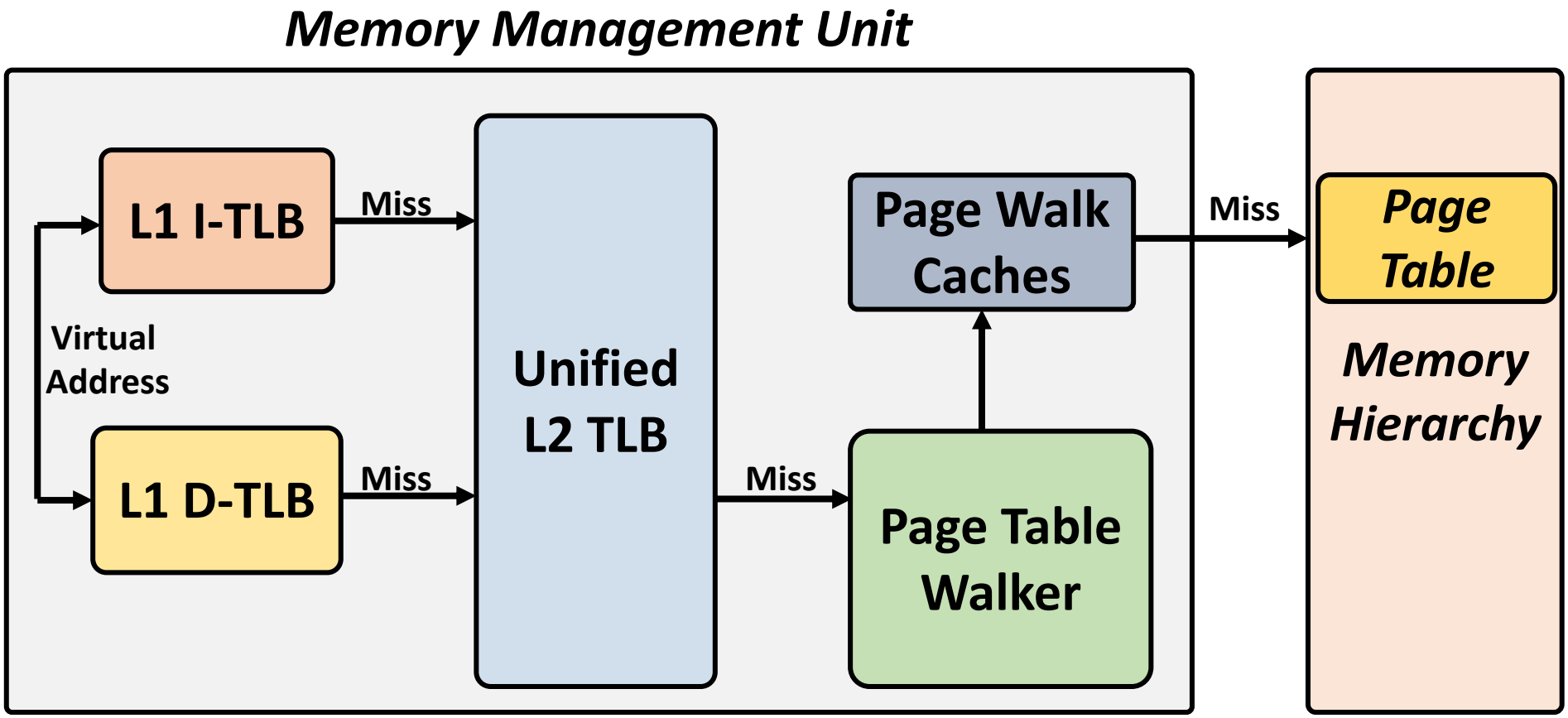


***Four* sequential memory accesses during a page table walk in x86-64**

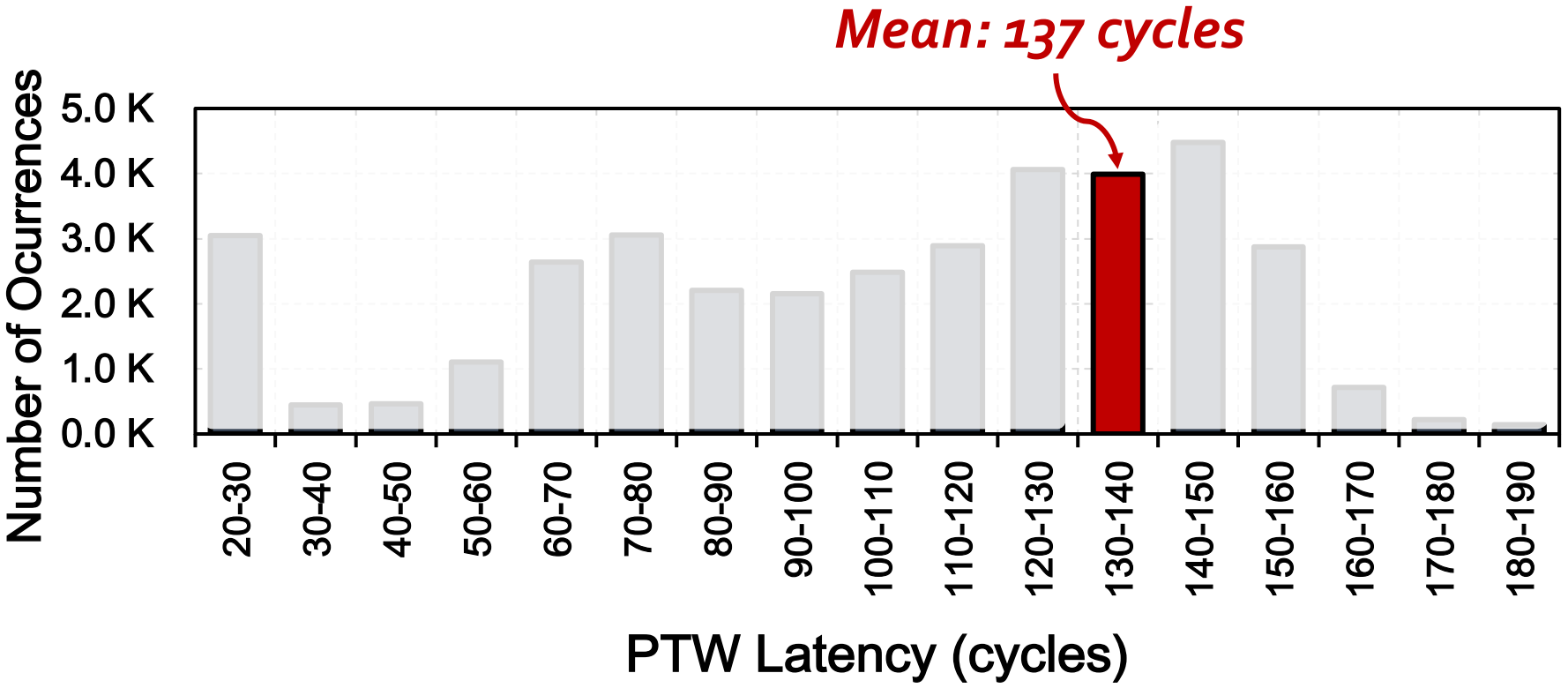
Address Translation Flow (I)



Address Translation Flow (II)

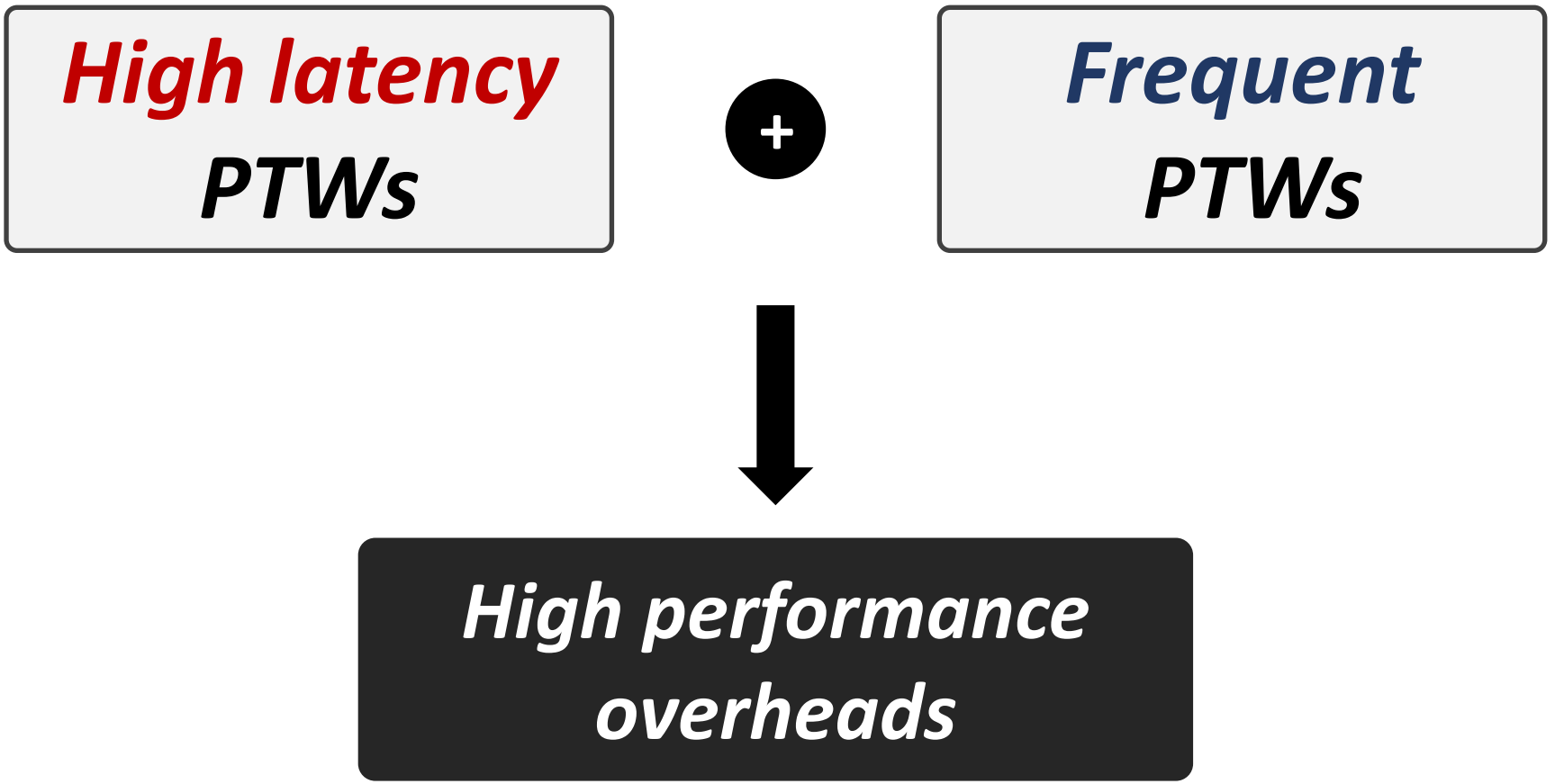


Address Translation Overhead



Core spends 137 cycles on average to perform a PTW

Address Translation Overhead



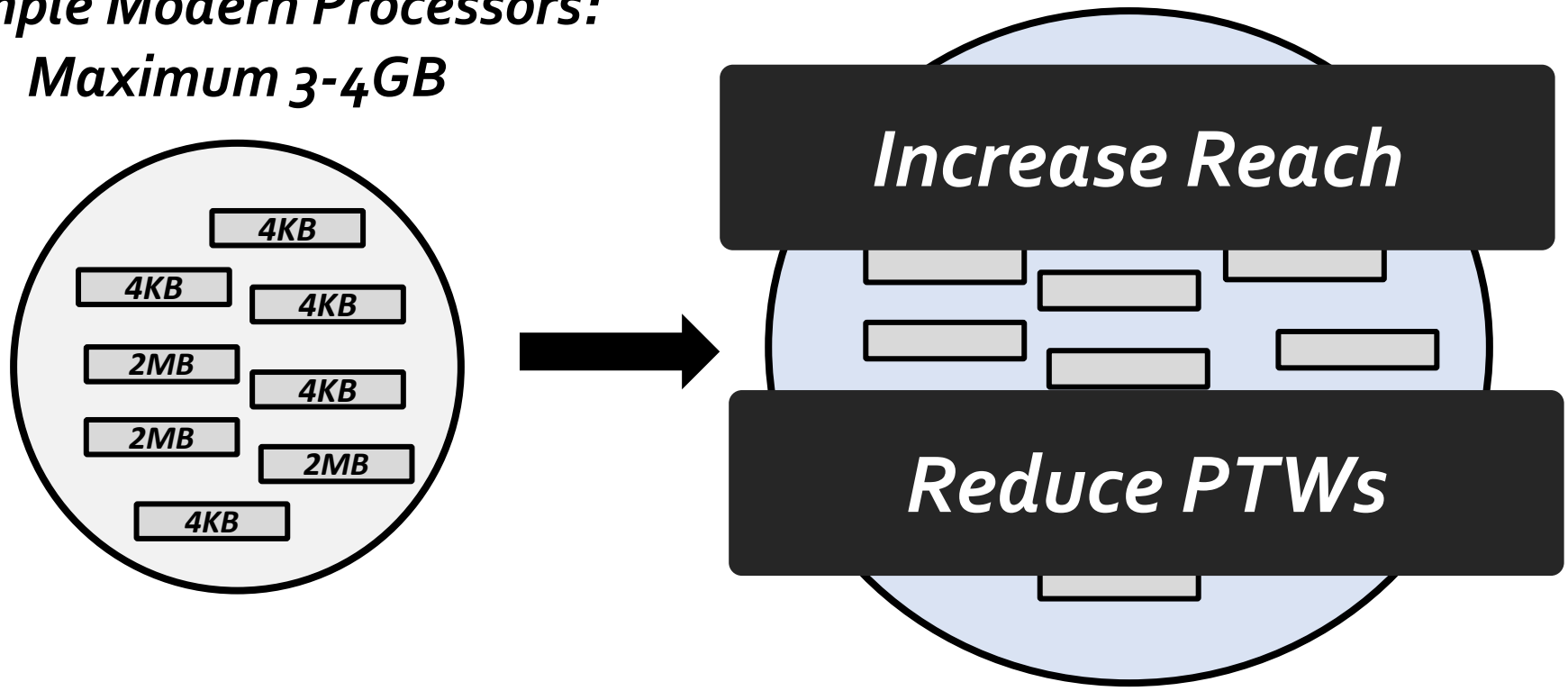
Potential Solution

Reduce PTW frequency by increasing address translation reach

Address Translation Reach: Definition

Amount of VA-to-PA mappings stored by the processor's TLB hierarchy

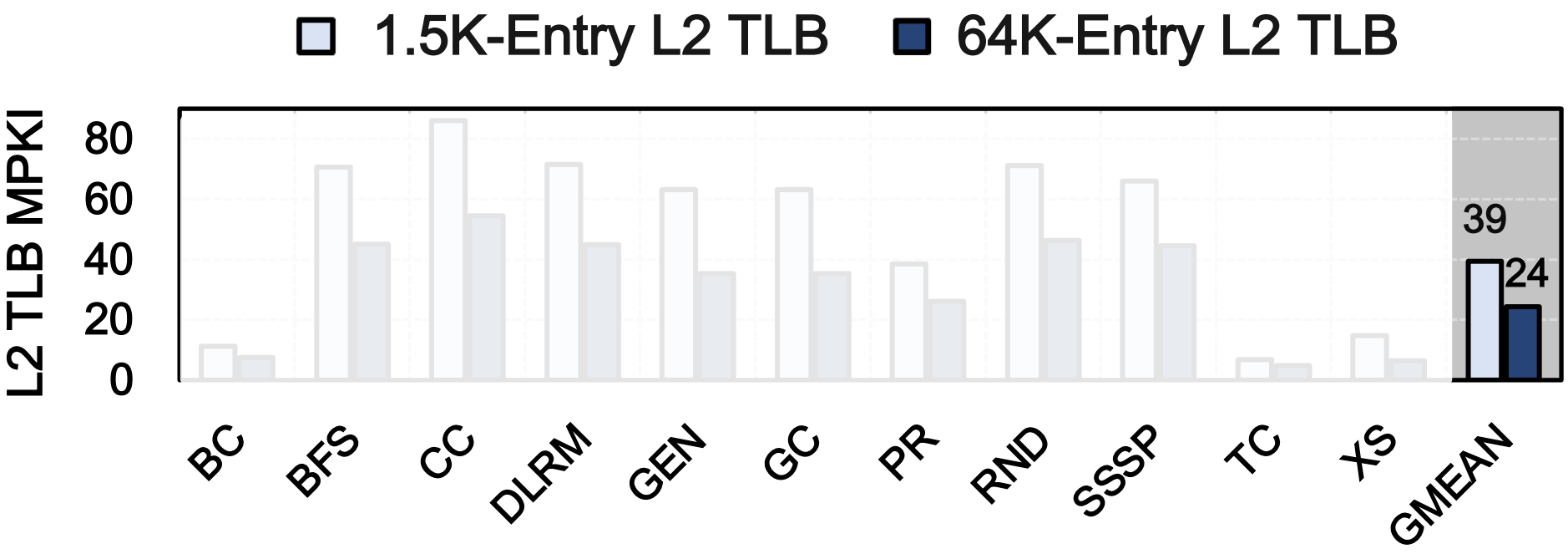
Example Modern Processors: Maximum 3-4GB



Increasing Address Translation Reach

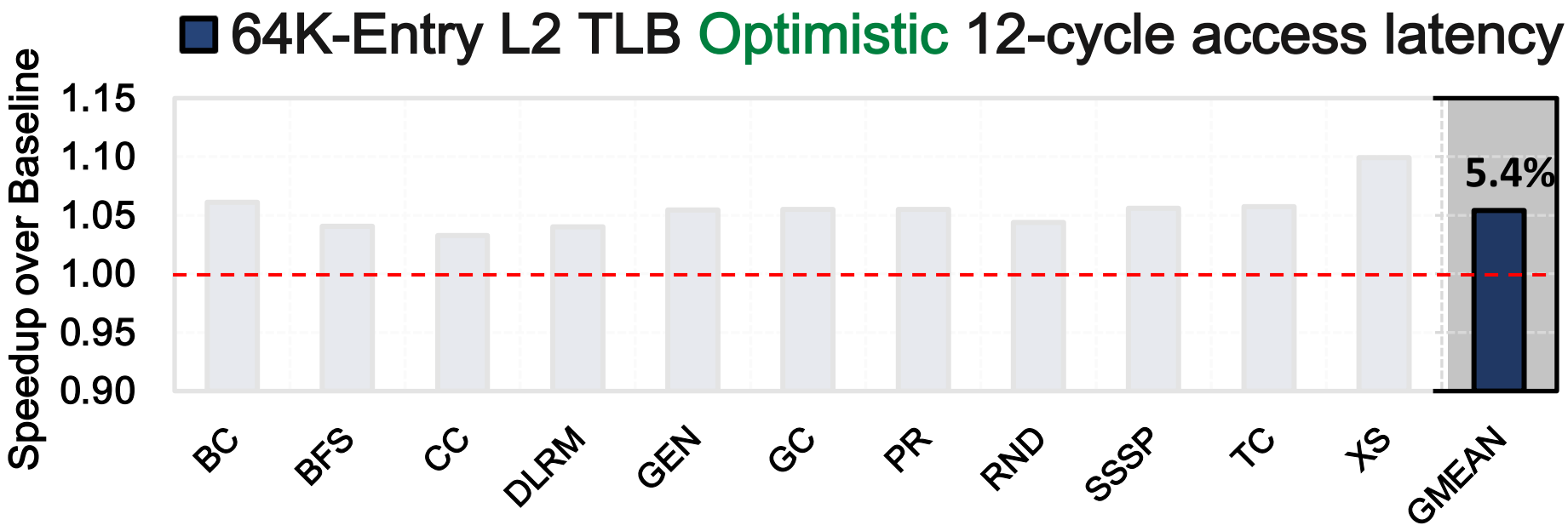
Large Hardware TLBs

Scaling Hardware L2 TLB (I)



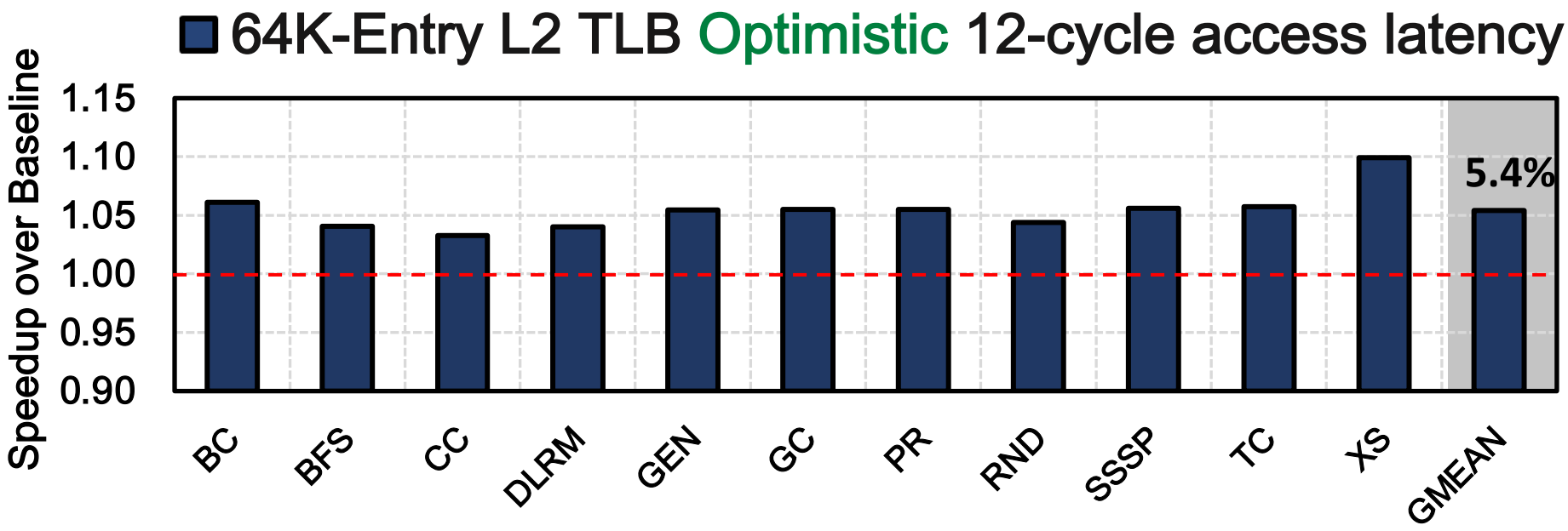
Employing a 64K-entry L2 TLB reduces MPKI from 39 to 24

Scaling Hardware L2 TLB (II)



64K-entry L2 TLB with optimistic access latency provides 5.4% speedup over baseline

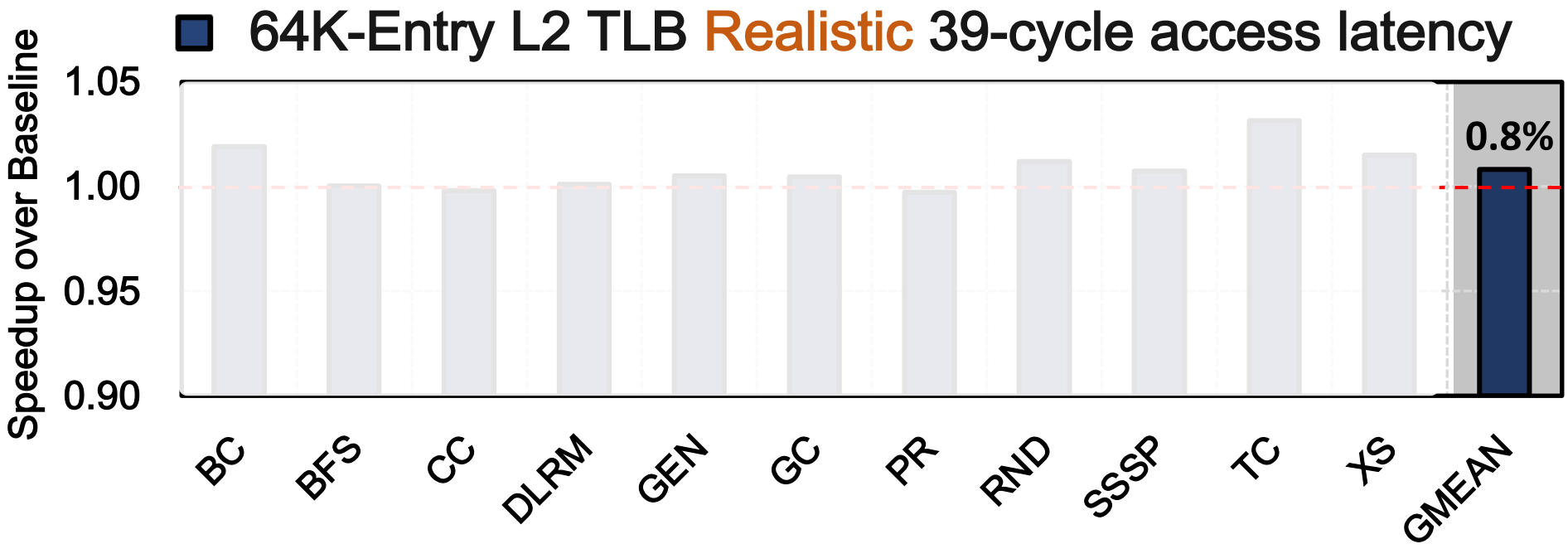
Scaling Hardware L2 TLB (II)



64K-entry L2 TLB with optimistic access latency provides 5.4% speedup over baseline

Benefits come for free?

Scaling Hardware L2 TLB (III)



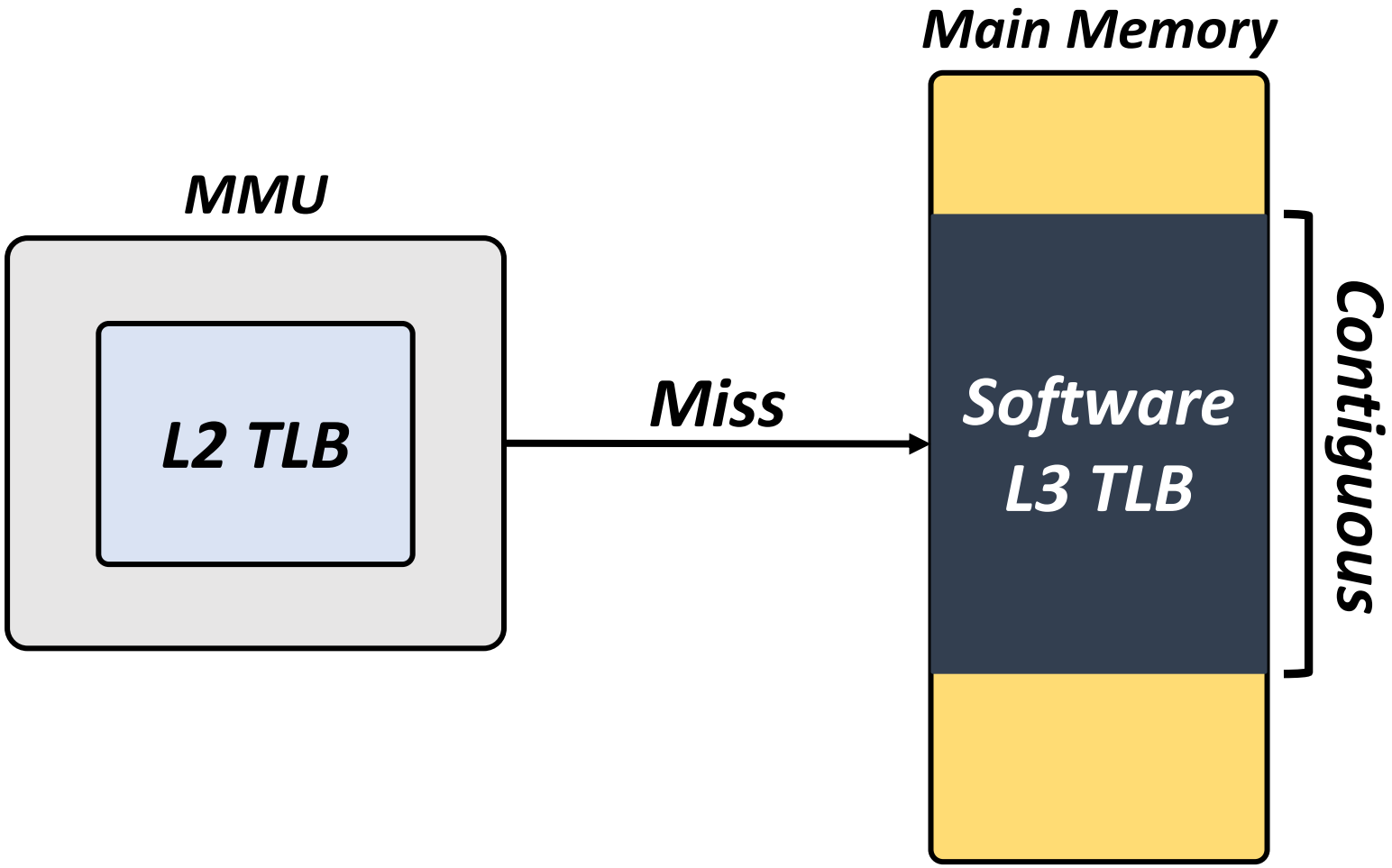
64K-entry L2 TLB with realistic access latency provides only 0.8% speedup over baseline

Increasing Translation Reach

Large Hardware TLBs

Large Software-Managed TLBs

Large Software-Managed L3 TLB



Drawbacks of Software-Managed TLB

1. High Latency
2. Contiguous Physical Allocations
3. OS Modifications

Increasing Translation Reach

Large Hardware TLBs	*Large Software-Managed TLBs*

Both approaches come with major drawbacks

Talk Outline

Background & Motivation

Opportunity: Leverage Caches

Victima: Overview

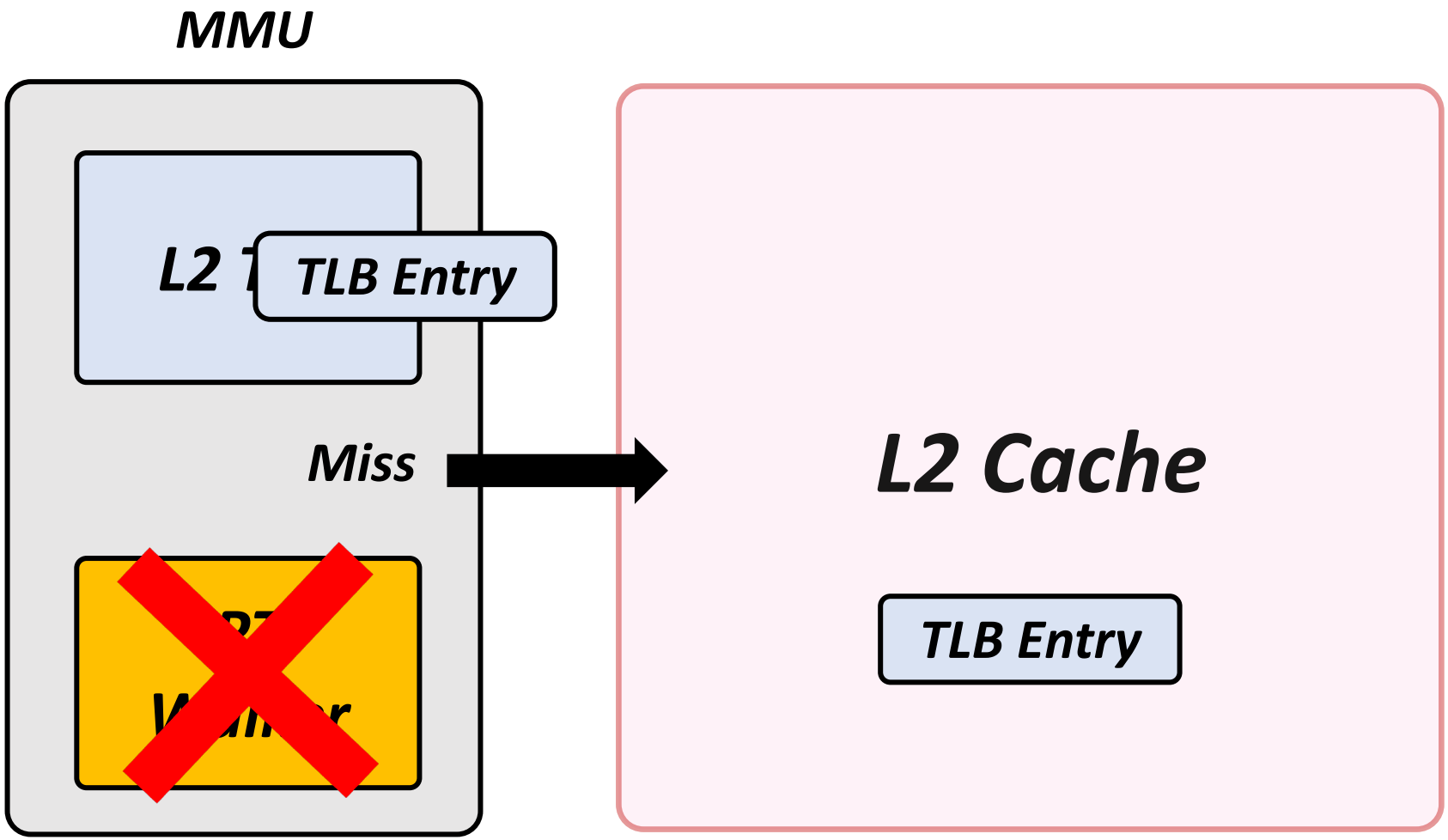
Victima: Detailed Design

Evaluation Results

Opportunity: Leverage Caches

Store TLB entries in hardware caches

Leverage Cache Hierarchy



Where is the Benefit?

L2 TLB

1.5K entries
12-cycle latency

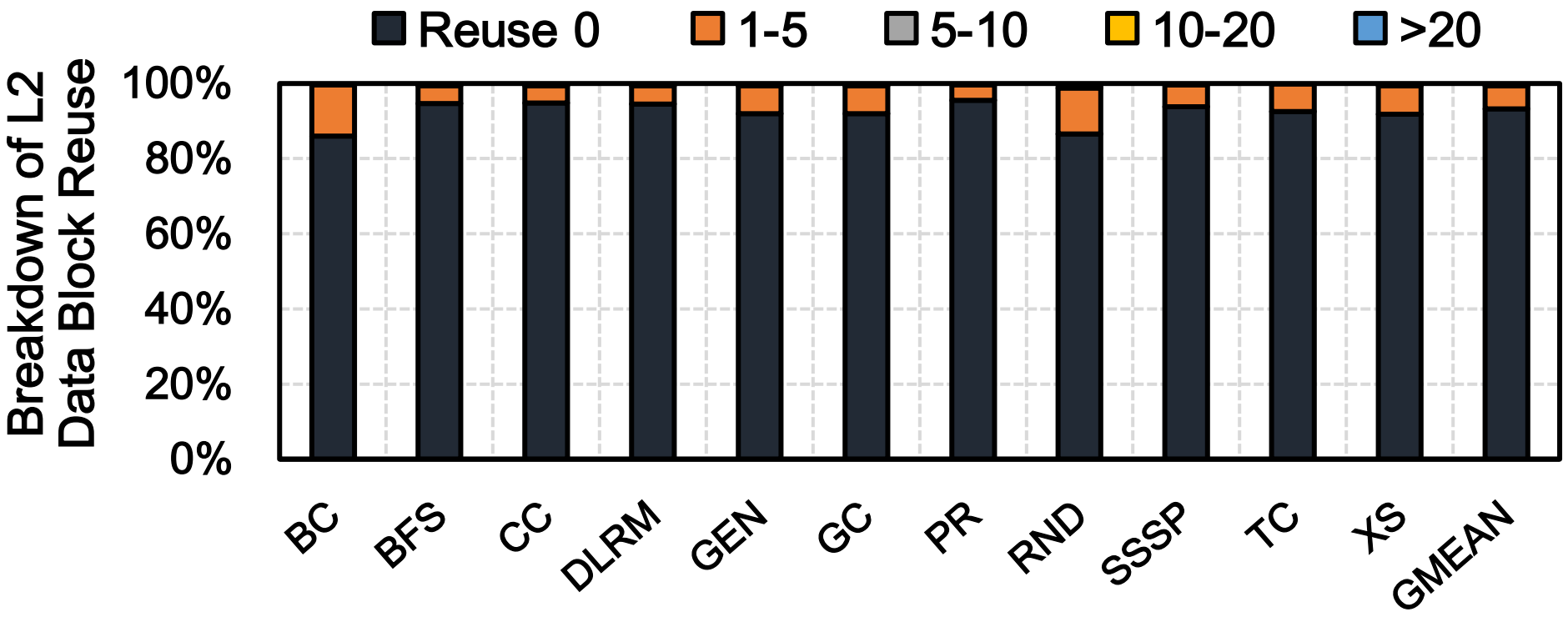
2MB L2 Cache

Fits 36x more
TLB entries

Low latency
(e.g., 16 cycles)

PTW takes 137 cycles on average

Interference with Program Data?



L2 cache is heavily underutilized

Talk Outline

Background & Motivation

Opportunity: Leverage Caches

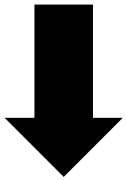
Victima: Overview

Victima: Detailed Design

Evaluation Results

Our Goal

Leverage cache resources
to store TLB entries

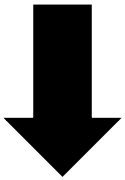


Drastically increase
the address translation reach of the processor

Victima: Key Idea

Repurpose L2 cache blocks

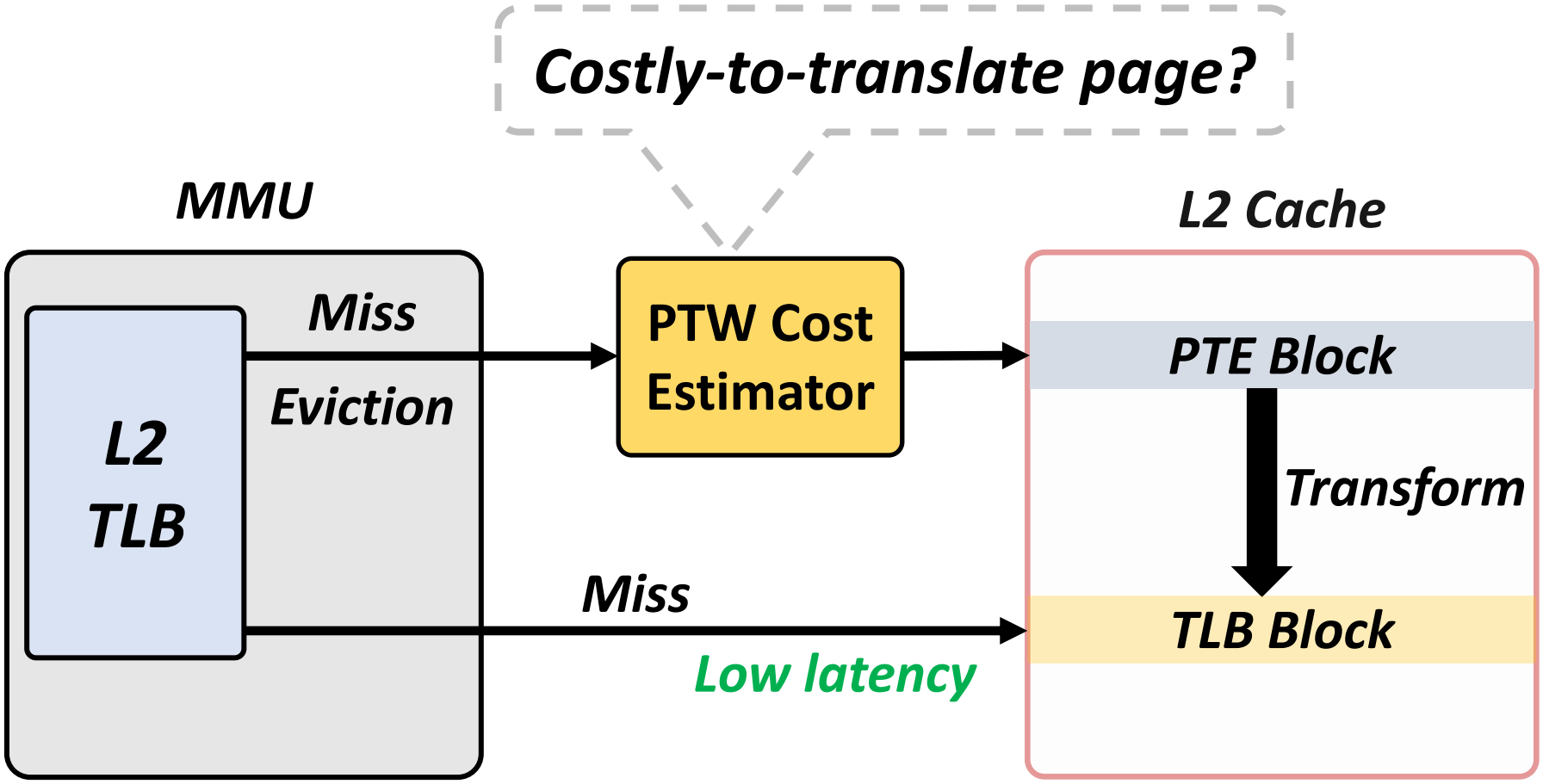
to store clusters of TLB entries



Low-latency *and* ***high-capacity*** *component*

to back up the L2 TLB

Victima: Overview



Victima Benefits

+ **Drastic increase** in address translation reach

+ **Fully transparent** to application/OS software

+ **No need** for contiguous physical allocations

+ **Compatible** with huge pages

Talk Outline

Background & Motivation

Opportunity: Leverage Caches

Victima: Overview

Victima: Detailed Design

Evaluation Results

Victima: Detailed Design

1. *L2 Cache Modifications*
2. *Allocation of TLB Entries in L2 Cache*
3. *Page Table Walk Cost Predictor*

Victima: L2 Cache Modifications

1. Access TLB blocks using virtual address

2. Perform tag matching for TLB blocks

Example: Cache Configuration

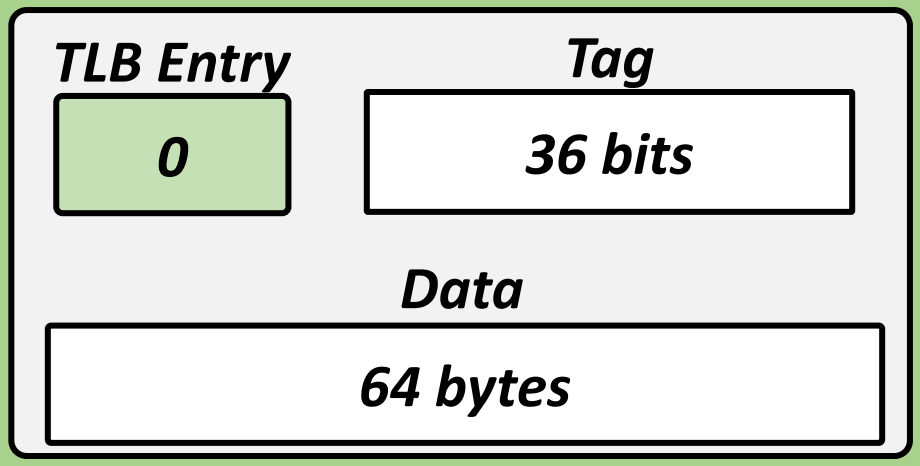
L2 Cache

1MB
16-way associative

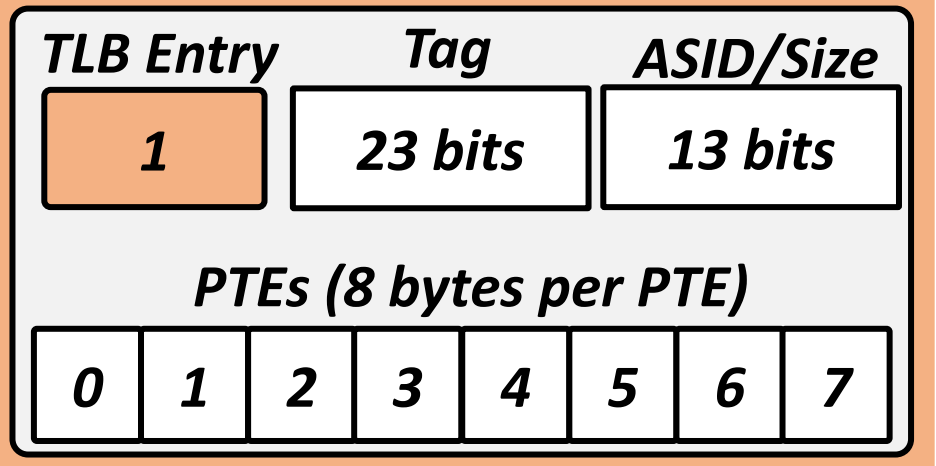
Set index | *10 bits*

Data Blocks vs. TLB Blocks in Caches

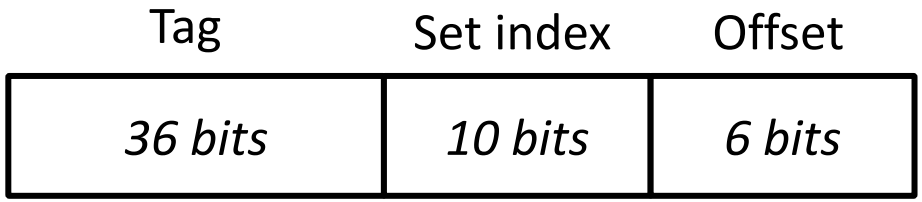
Data Block



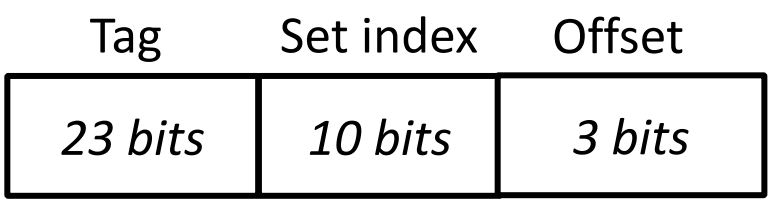
TLB Block



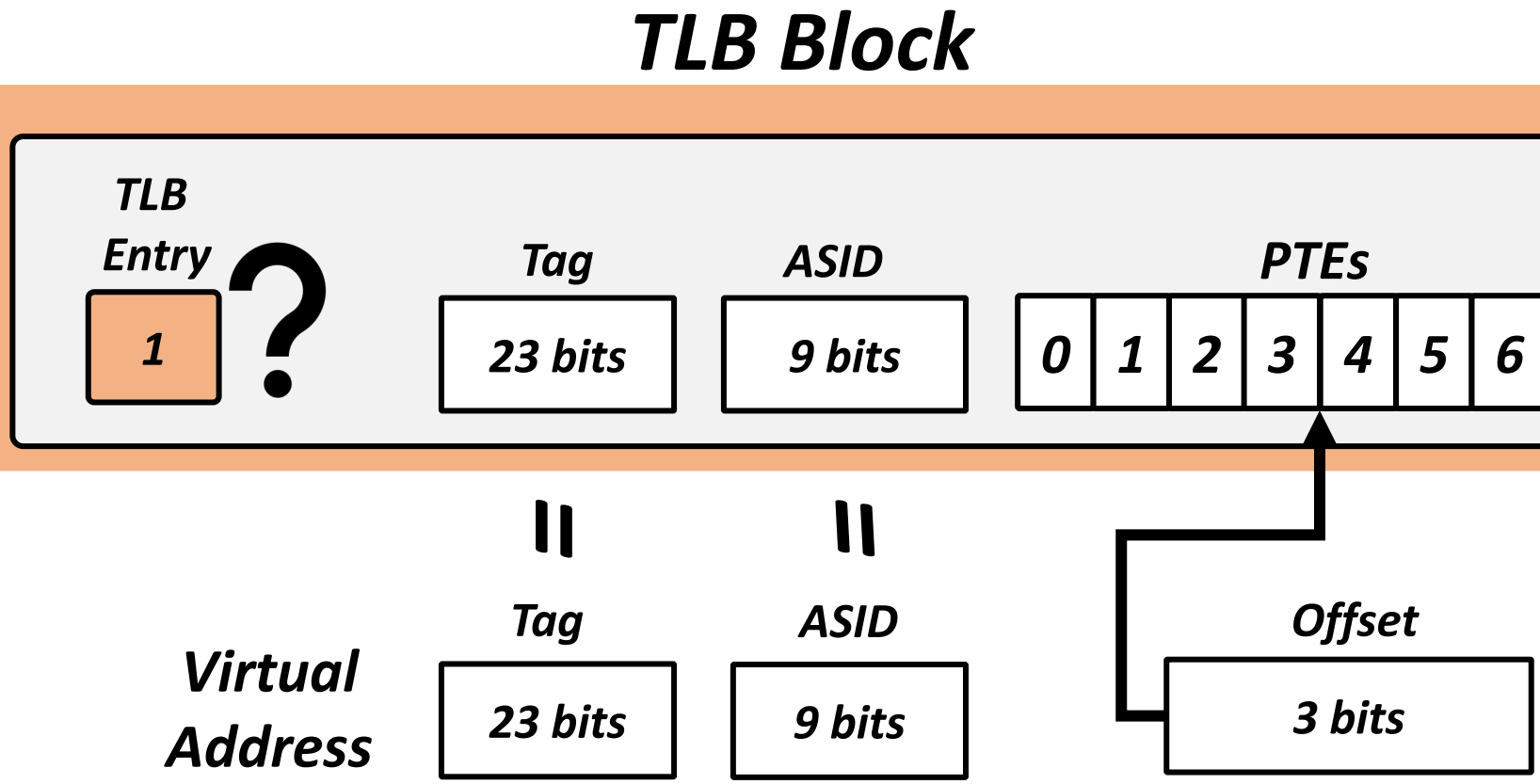
52-bit Physical Address



36-bit Virtual Page Number (4KB)



Tag Matching for TLB Block



Victima: L2 Cache Modifications

1 *L2 Cache Modifications*

2 ***Allocation of TLB Entries in L2 Cache***

3 *Page Table Walk Cost Predictor*

Allocation of TLB Entries in L2 Cache

1. On L2 TLB Miss

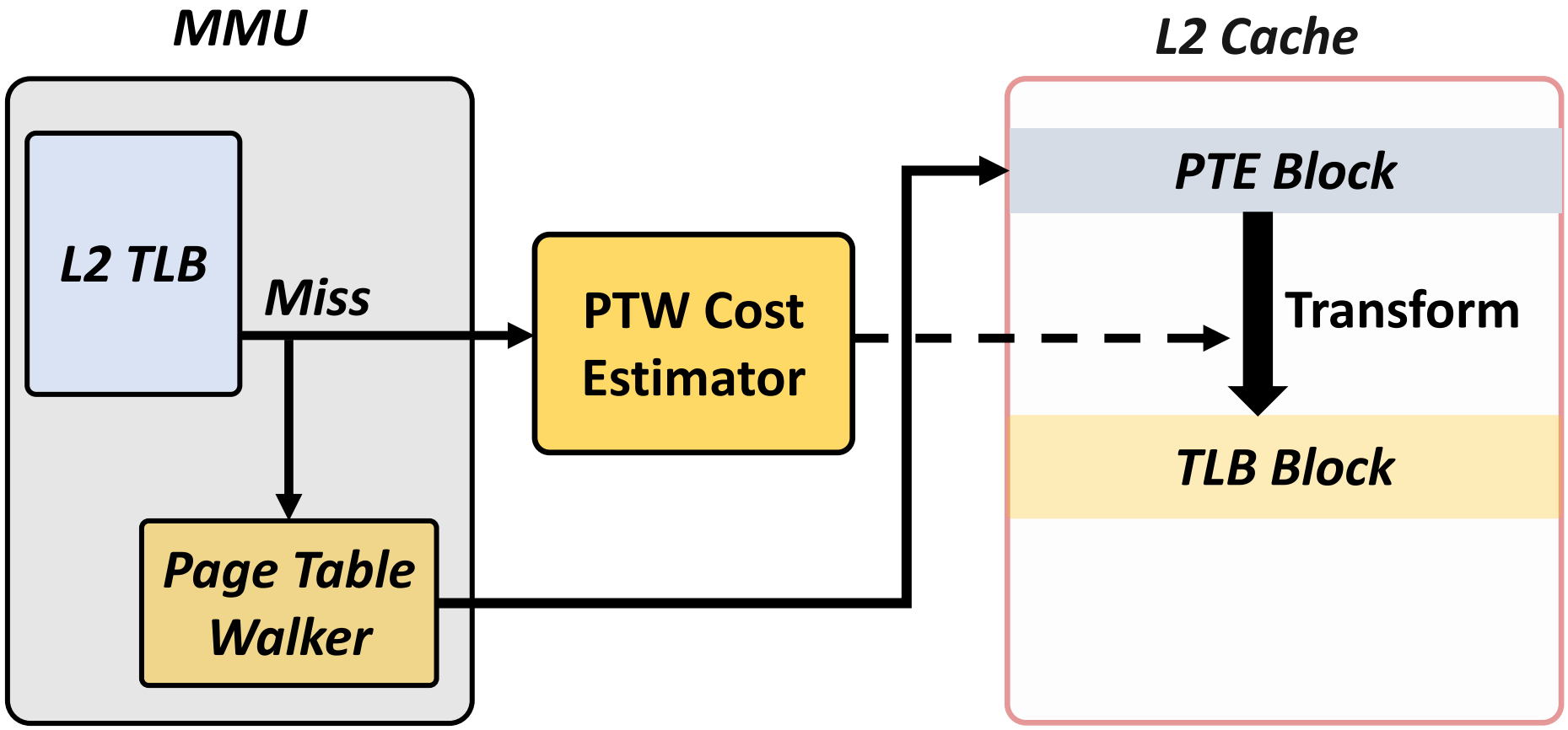
2. On L2 TLB Eviction

Allocation of TLB Entries in L2 Cache

1. On L2 TLB Miss

2. On L2 TLB Eviction

Allocating TLB Blocks – L2 TLB Miss

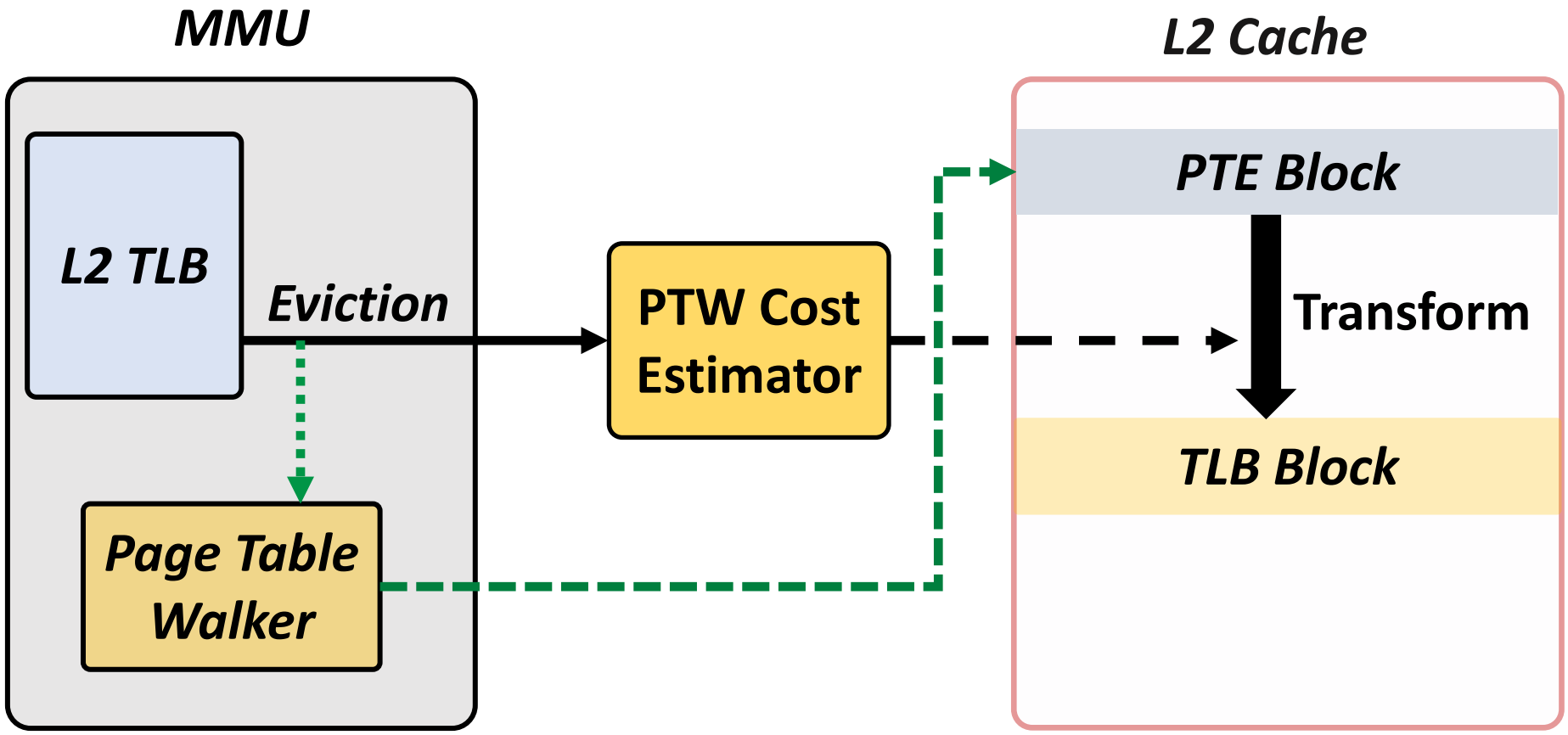


Allocation of TLB Entries in L2 Cache

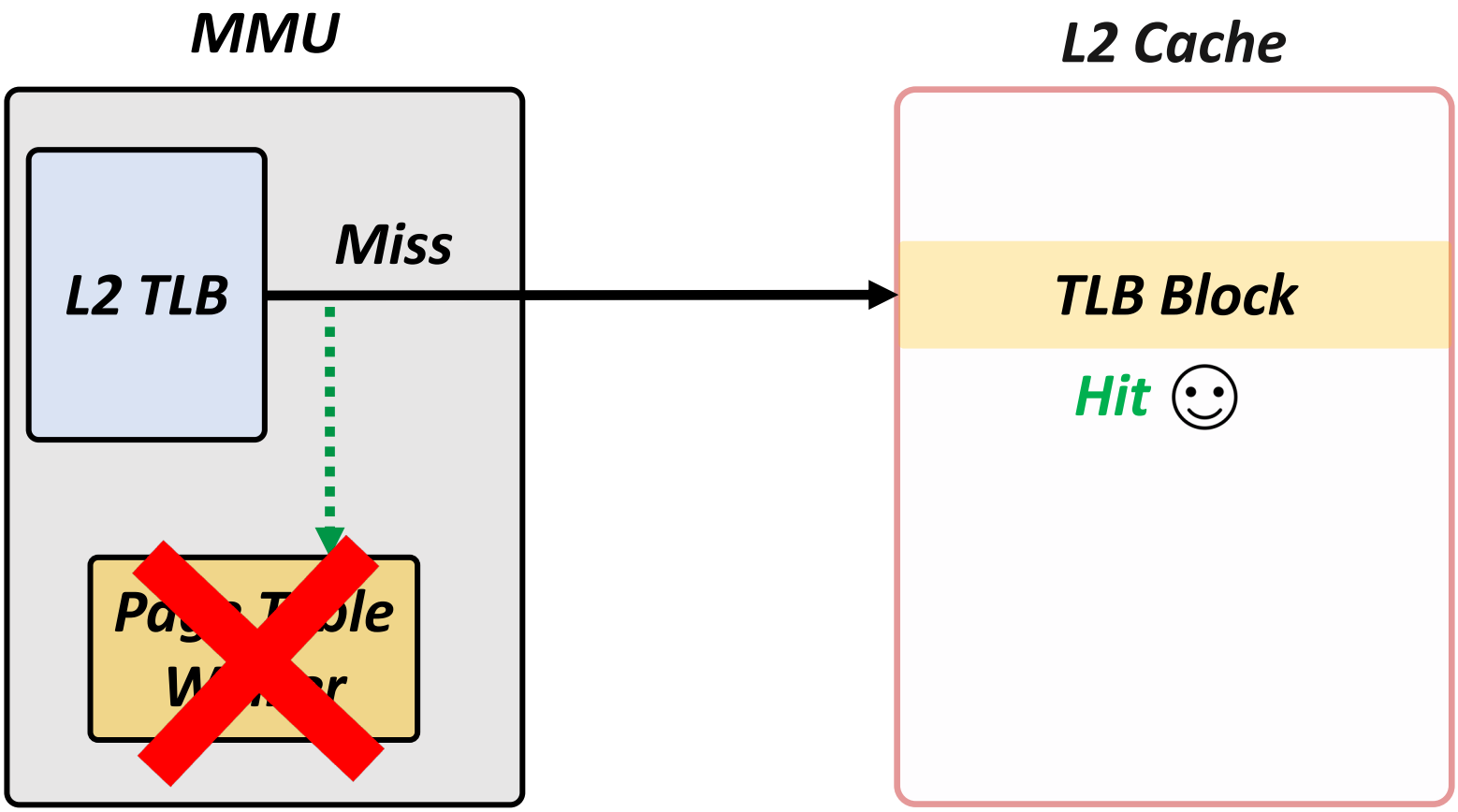
1. On L2 TLB Miss

2. On L2 TLB Eviction

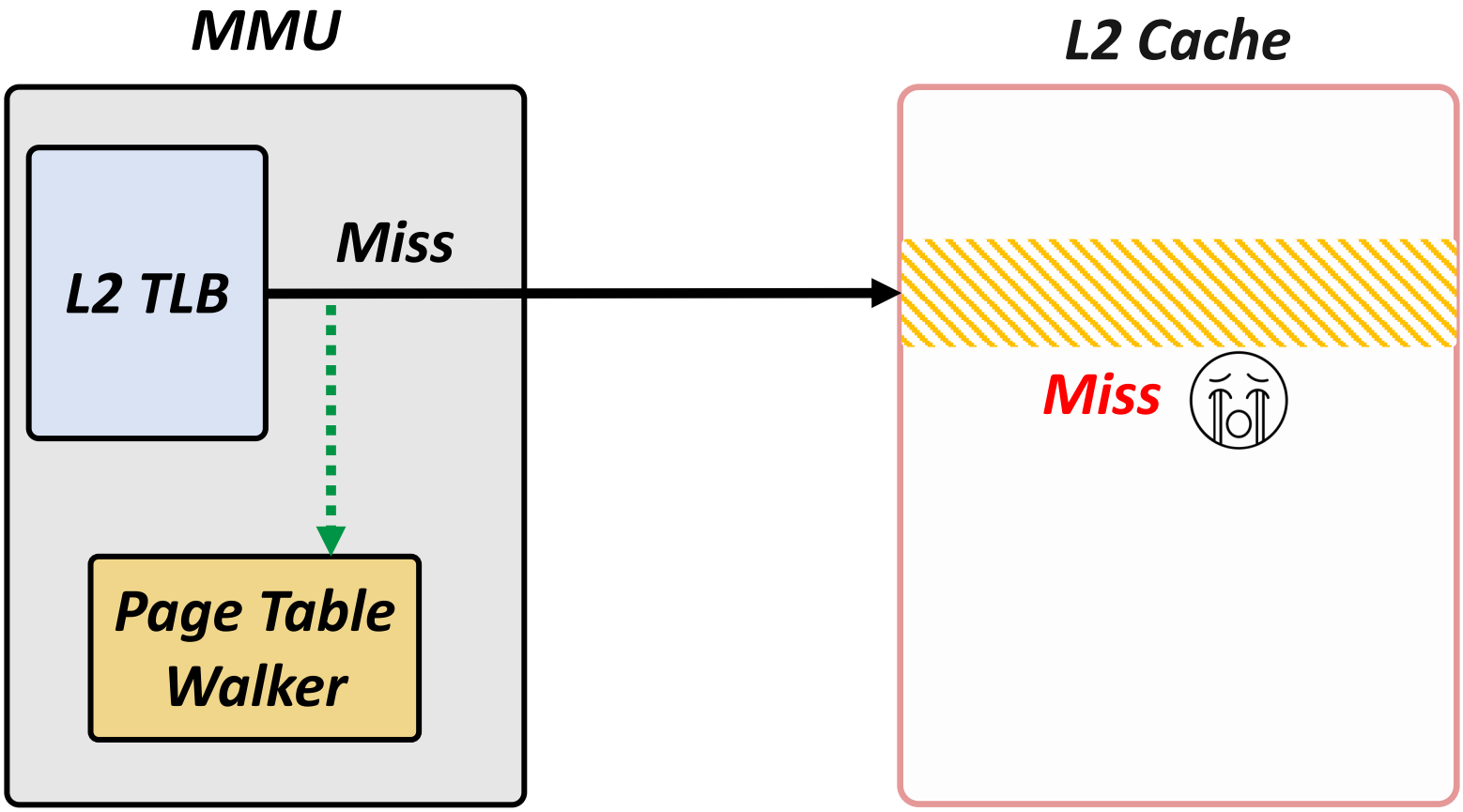
Allocating TLB Blocks – L2 TLB Eviction



Address Translation in Victima (I)



Address Translation in Victima (II)



Victima: Detailed Design

1 *L2 Cache Modifications*

2 *Allocation of TLB Blocks in L2 Cache*

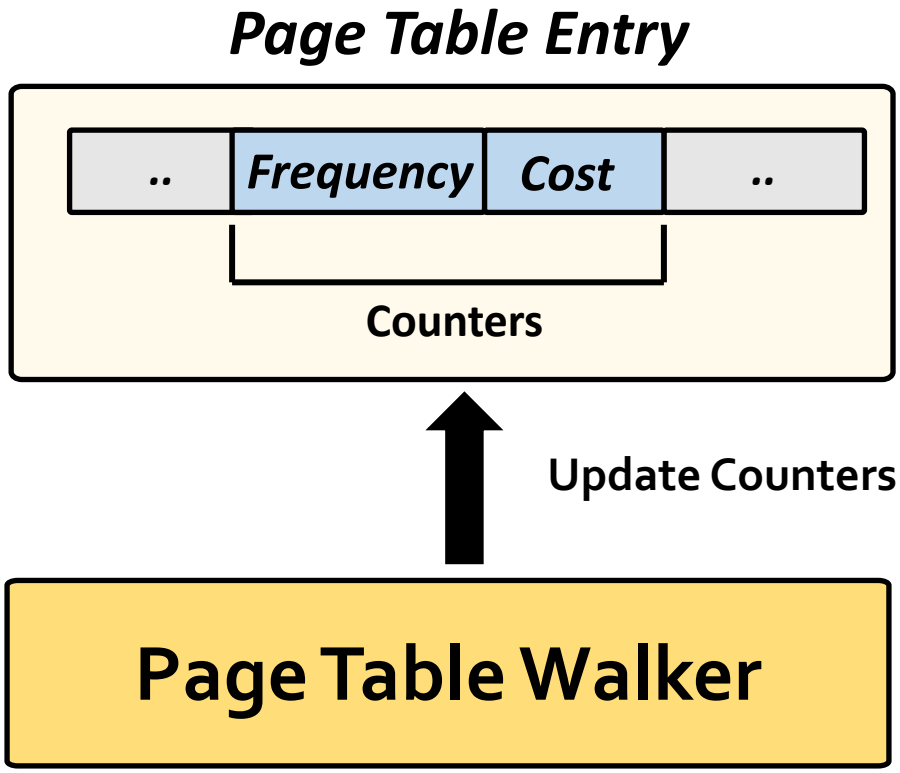
3 ***Page Table Walk Cost Predictor***

PTW Cost Predictor: Objective

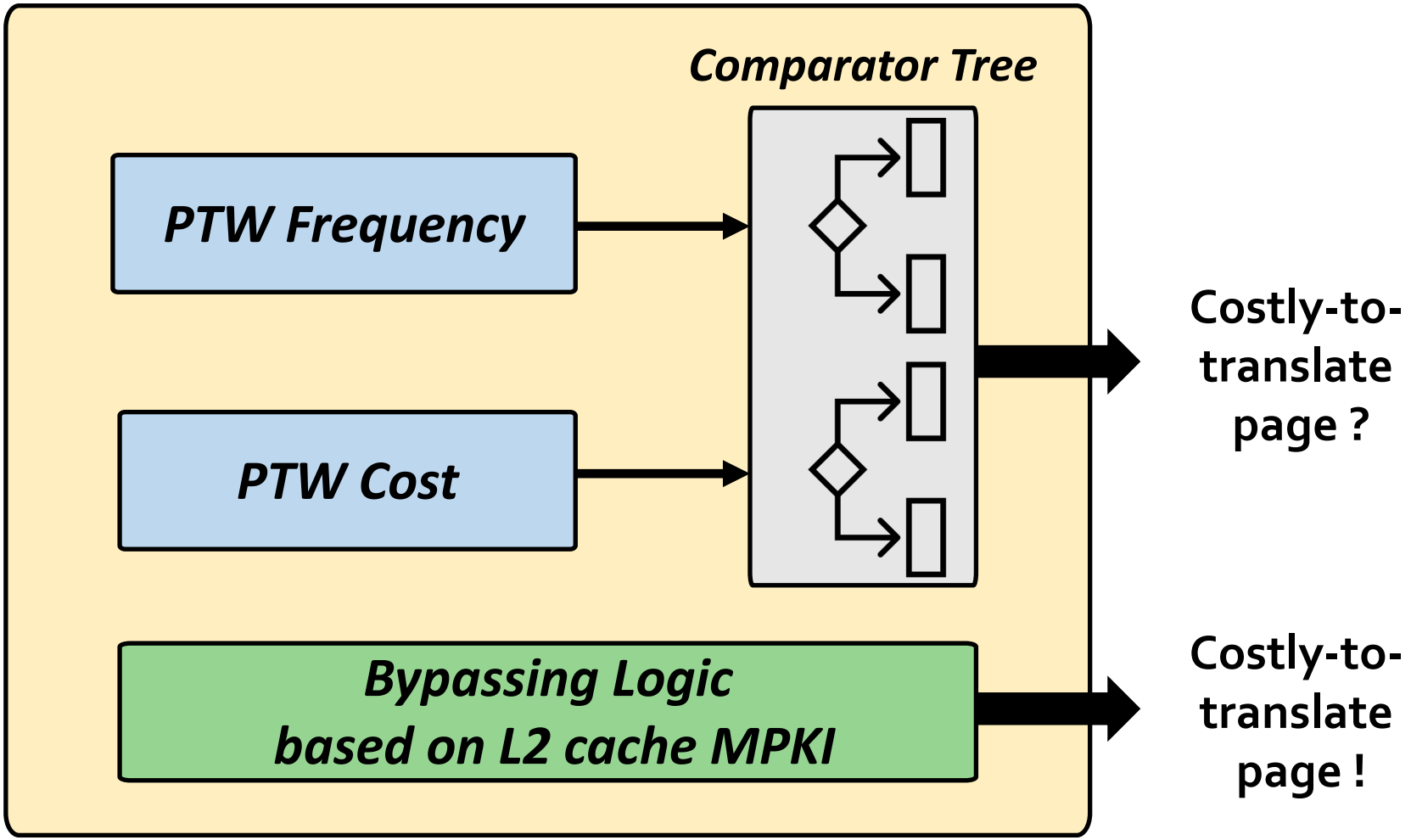
Predict which pages are costly-to-translate

Insert only those TLB blocks in L2 cache

Tracking Costly-to-Translate Pages



PTW Cost Predictor (PTW-CP)



PTW-CP Details in the Paper

Feature engineering to find minimal set of useful features

2-feature comparator predicts costly-to-translate pages with 82% accuracy

PTW-CP Feature Set

Feature (per PTE)	Bits	Description
Page Size	1	The size of the page (4KB or 2MB)
Page Table Walk Cost	**3**	**DRAM accesses during a PTW**
Page Table Walk Frequency	**3**	**The number of PTWs**
LLPWC Hits	5	The number of third-level PWC hits
L1 TLB Misses	5	The number of L1 TLB misses
L2 TLB Misses	5	The number of L2 TLB hits
L2 Cache Hits	5	The number of L2 cache hits
L1 TLB Evictions	5	The number of L1 TLB evictions
L2 TLB Evictions	6	The number of L2 TLB evictions
Accesses	6	The number of accesses to the page

Feature engineering to find minimal set of useful features

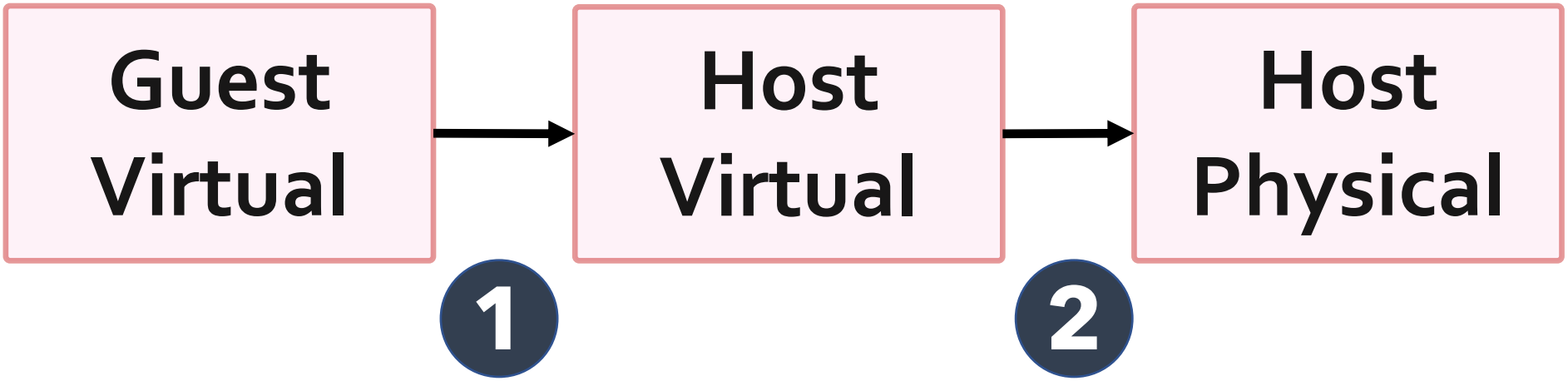
PTW-CP Exploration

	NN-10	NN-5	NN-2	**Comparator**
Feature Size	10	5	2	**2**
Number of Layers	4	4	6	**N/A**
Size of Hidden Layers	16	64	4	**N/A**
Number of Neurons	737	8769	97	**N/A**
Size (B)	5896	70152	776	**24**
Recall	0.9334	0.9244	0.8962	**0.8961**
Accuracy	0.9213	0.9172	0.8290	**0.8290**
Precision	0.8768	0.8747	0.7333	**0.7334**
F1-score	0.9042	0.8989	0.8066	**0.8066**

2-feature comparator predicts costly-to-translate pages with 82% accuracy

Virtualized Environments

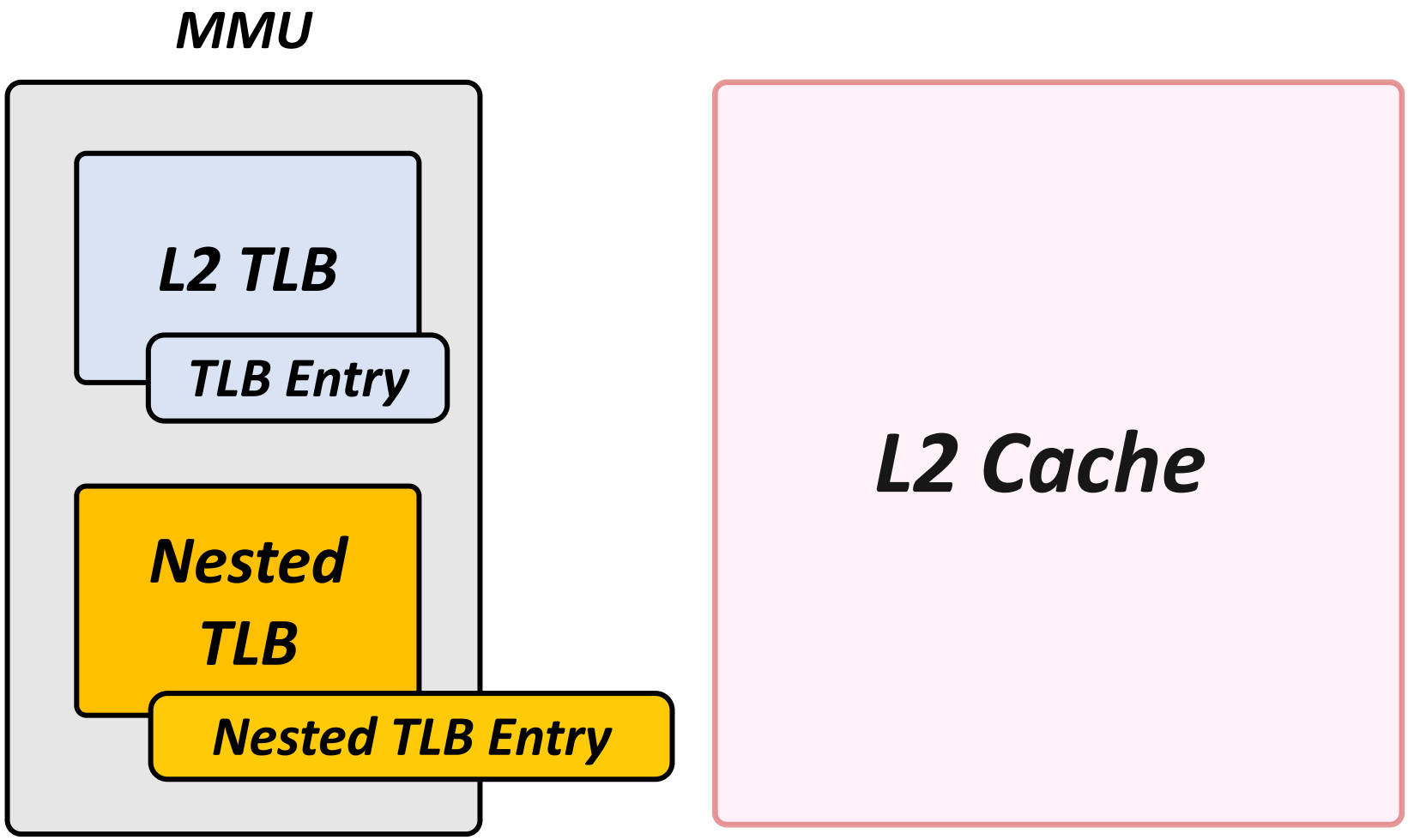
Two-level address translation



Virtualized Environments

L2 TLB	Guest-Virtual Host-Physical
Nested TLB	Guest-Virtual Host-Virtual

Virtualized Environments



Talk Outline

Background & Motivation

Opportunity: Leverage Caches

Victima: Overview

Victima: Detailed Design

Evaluation Results

Evaluation Methodology

Sniper Multicore Simulator extended with:

- TLB Hierarchy with multiple page sizes
- Radix page table walker
- Page walk caches

https://github.com/CMU-SAFARI/Victima

Workloads: Executed for 500M instructions

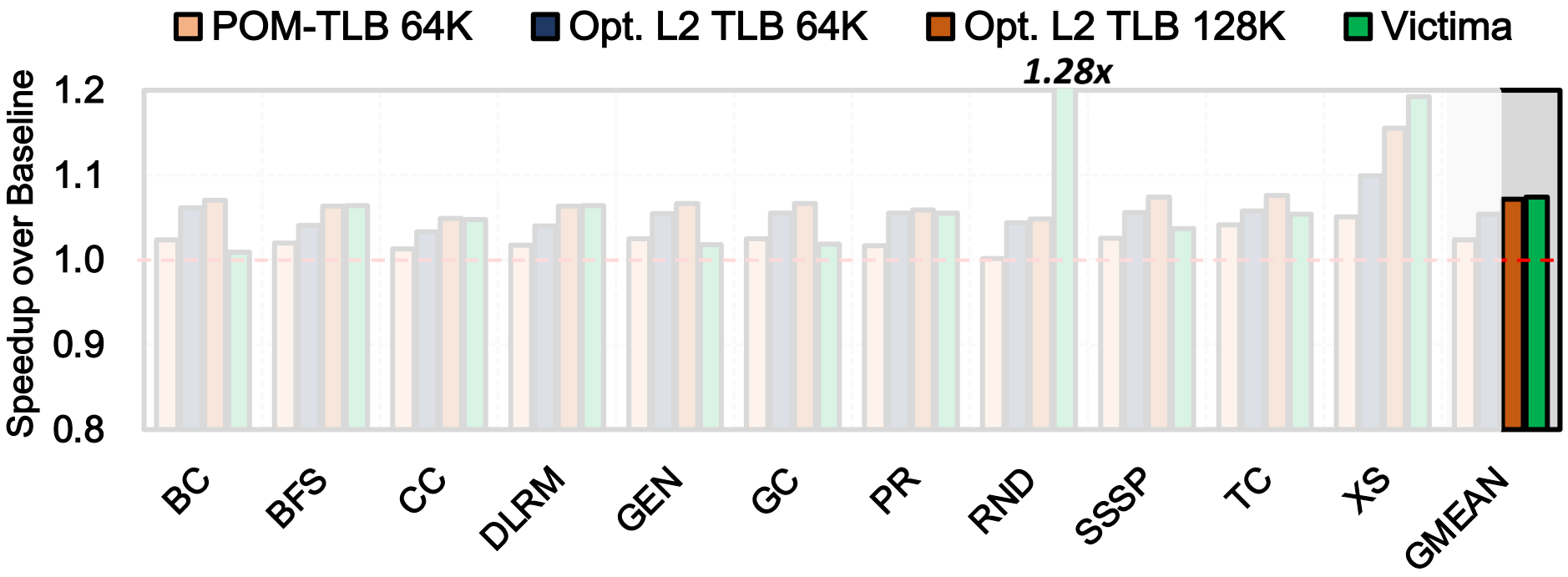
- GraphBIG: PR, BFS, BC, GC, CC
- HPCC: Randacc
- XSBench: Particle Simulation
- DLRM: Sparse-length sum
- GenomicsBench: k-mer counting

Configurations – Native Execution

- **Radix**: Baseline system with 1.5K-entry L2 TLB and Transparent Huge pages enabled
- **Optimistic L2 TLB-64K**: System with 64K-entry L2 TLB (optimistic 12-cycle access latency)
- **Optimistic L2 TLB-128K**: System with 128K-entry L2 TLB (optimistic 12-cycle access latency)
- **POM-TLB**[1]: System with 64K-entry software-managed L3 TLB
- **Victima**

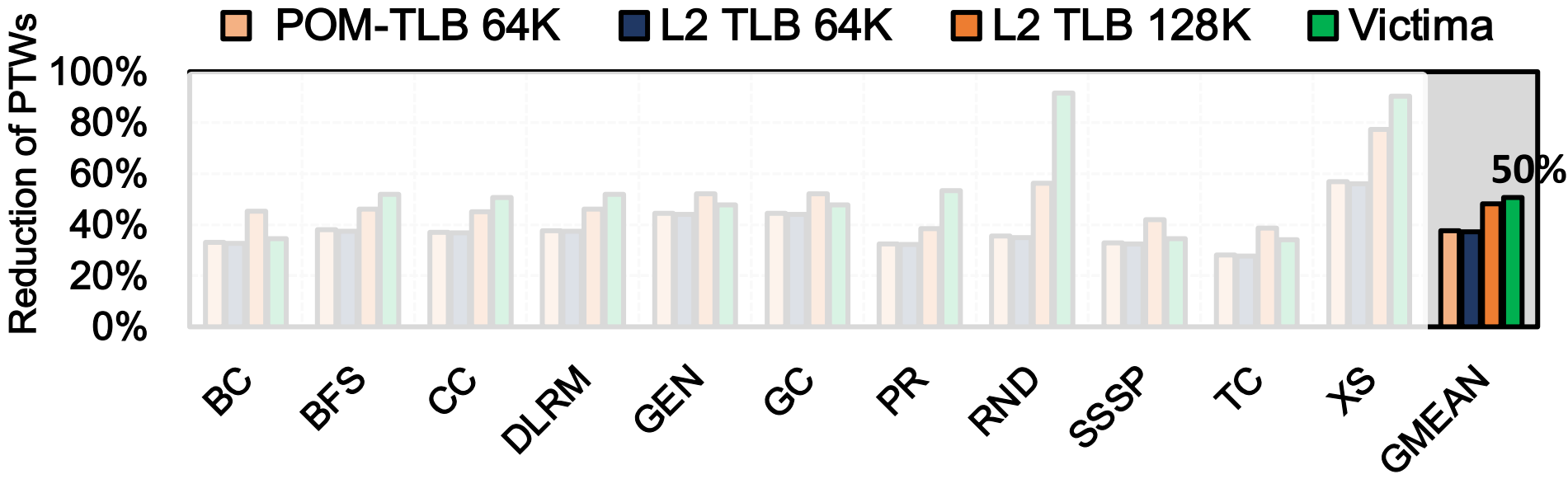
[1] Ryoo et al. "Rethinking TLB designs in virtualized environments: A very large part-of-memory TLB" ISCA 2017

Performance Speedup



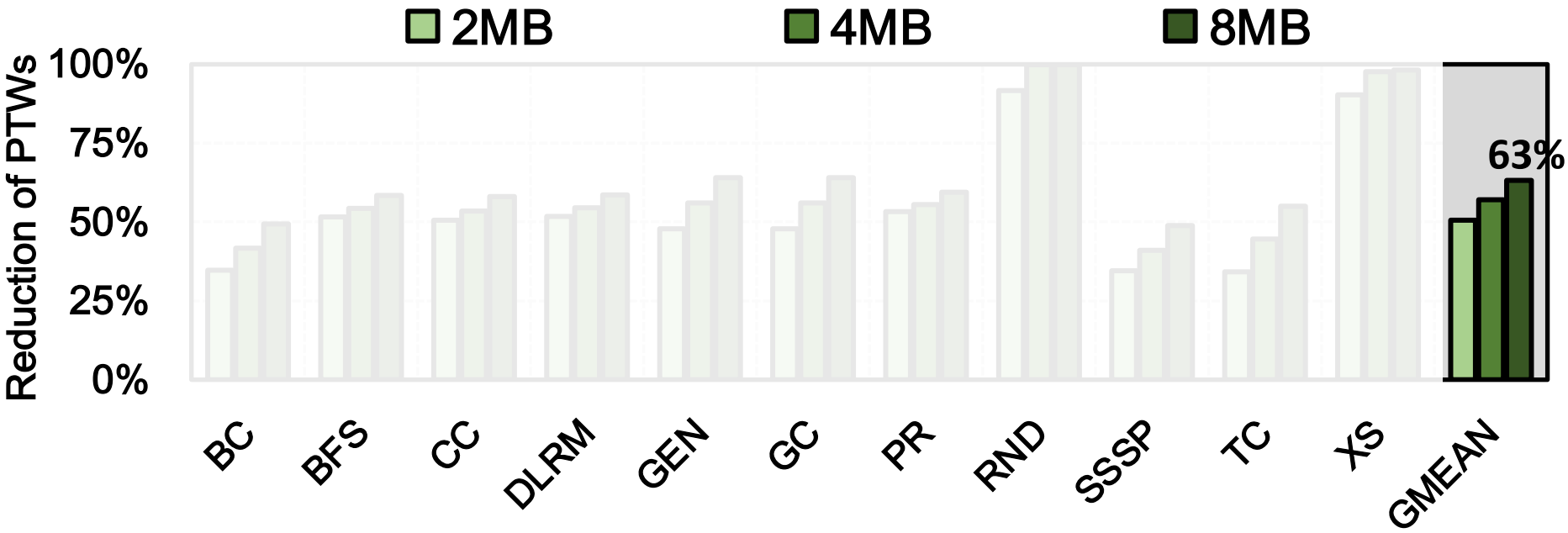
Victima achieves similar performance to the optimistically fast 128K-entry L2 TLB

Reduction of Page Table Walks



Victima reduces PTWs by 50% on average compared to the baseline

Effect of L2 Cache Size on Victima



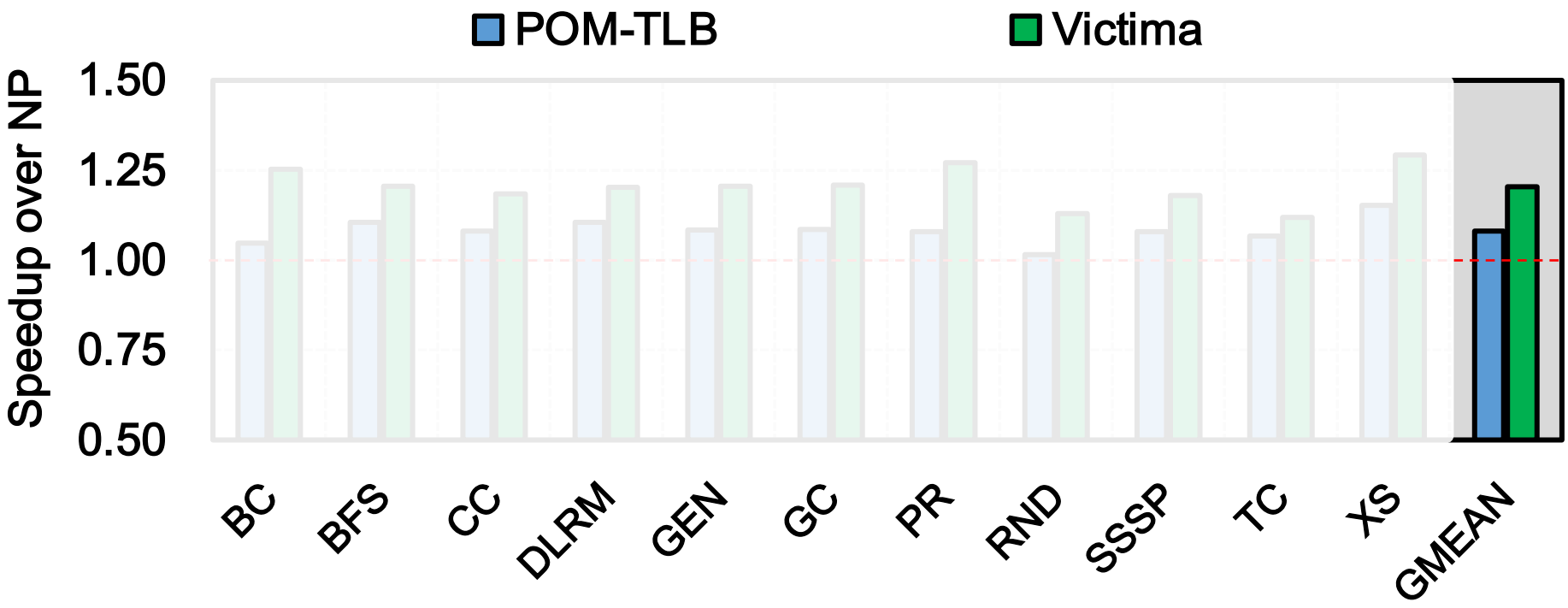
Employing an 8MB L2 cache with Victima reduces PTWs by 63%

Configurations in Virtualized Environments

- **Nested Paging[1]:** Baseline system that performs Nested PTWs
- **POM-TLB[2]**: System with 64K-entry software-managed L3 TLB
- **Ideal Shadow Paging[3]**: System that employs an ideal version of Shadow Paging
- **Victima**: Caching both TLB and Nested TLB entries in the L2 cache

[1] Bhargava et al. "Accelerating two-dimensional page walks for virtualized systems" ASPLOS 2008
[2] Ryoo et al. "Rethinking TLB designs in virtualized environments: A very large part-of-memory TLB" ISCA 2017
[3] "Agile paging: Exceeding the best of Nested and Shadow Paging" ISCA 2016

Performance in Virtualized Environments



Victima outperforms 64K-entry software-managed TLB by 12%

Area & Power Overhead

- Area and power overhead evalution using McPAT
- Comparison to a high-end Intel Raptor Lake

Victima incurs 0.04% area and 0.08% power overheads

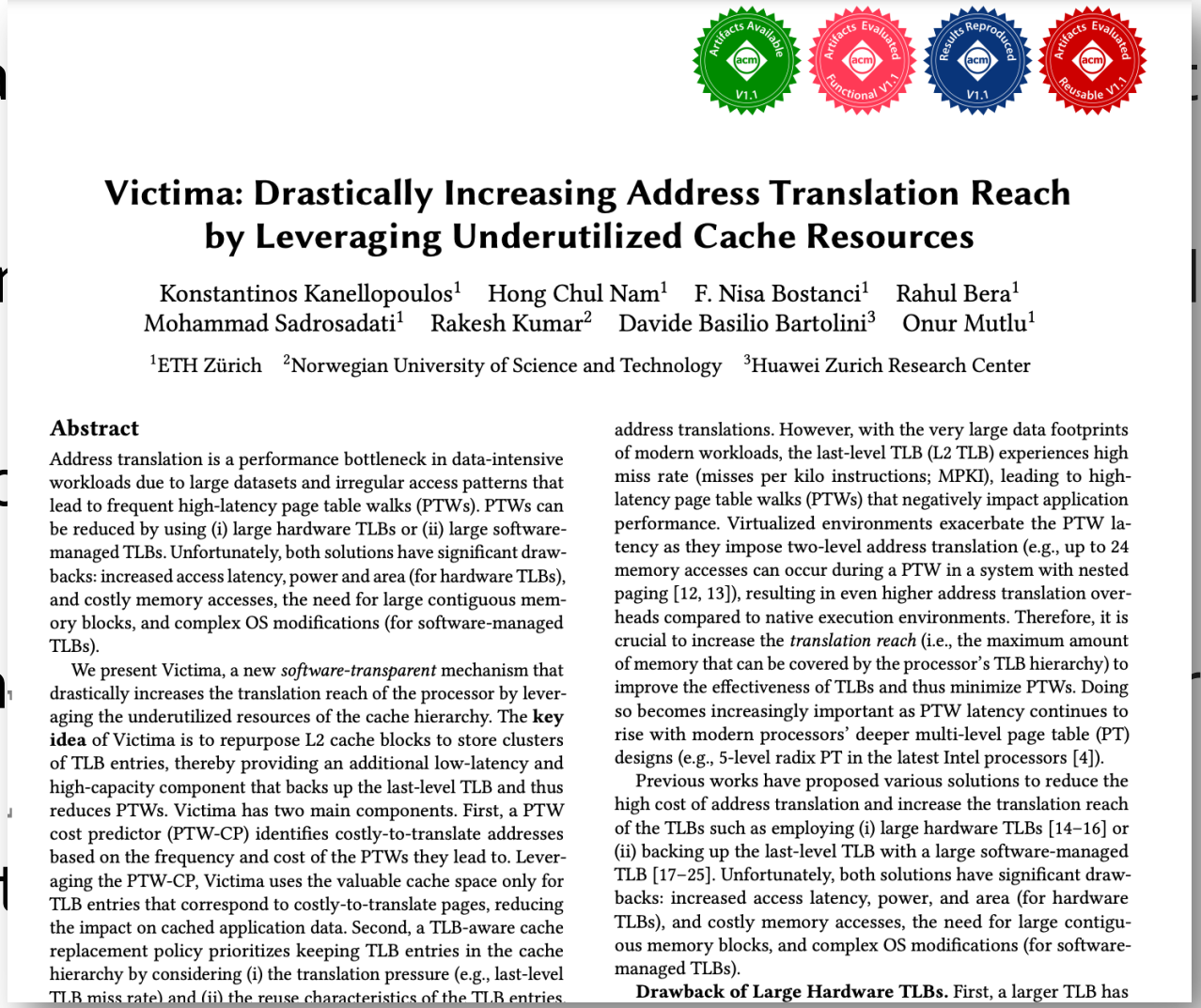
More in the paper

- Victima integration in virtualized environments
- Maintenance operations to handle TLB shootdowns
- TLB-Block-aware replacement policy
- Implementation details of PTW cost estimator
- Translation reach provided by Victima

https://arxiv.org/abs/2310.04158

More in the paper

- Victima
- Mainter
- TLB-Blo
- Implem
- Translat



Victima: Drastically Increasing Address Translation Reach by Leveraging Underutilized Cache Resources

Konstantinos Kanellopoulos[1] Hong Chul Nam[1] F. Nisa Bostanci[1] Rahul Bera[1]
Mohammad Sadrosadati[1] Rakesh Kumar[2] Davide Basilio Bartolini[3] Onur Mutlu[1]

[1]ETH Zürich [2]Norwegian University of Science and Technology [3]Huawei Zurich Research Center

Abstract

Address translation is a performance bottleneck in data-intensive workloads due to large datasets and irregular access patterns that lead to frequent high-latency page table walks (PTWs). PTWs can be reduced by using (i) large hardware TLBs or (ii) large software-managed TLBs. Unfortunately, both solutions have significant drawbacks: increased access latency, power and area (for hardware TLBs), and costly memory accesses, the need for large contiguous memory blocks, and complex OS modifications (for software-managed TLBs).

We present Victima, a new *software-transparent* mechanism that drastically increases the translation reach of the processor by leveraging the underutilized resources of the cache hierarchy. The **key idea** of Victima is to repurpose L2 cache blocks to store clusters of TLB entries, thereby providing an additional low-latency and high-capacity component that backs up the last-level TLB and thus reduces PTWs. Victima has two main components. First, a PTW cost predictor (PTW-CP) identifies costly-to-translate addresses based on the frequency and cost of the PTWs they lead to. Leveraging the PTW-CP, Victima uses the valuable cache space only for TLB entries that correspond to costly-to-translate pages, reducing the impact on cached application data. Second, a TLB-aware cache replacement policy prioritizes keeping TLB entries in the cache hierarchy by considering (i) the translation pressure (e.g., last-level TLB miss rate) and (ii) the reuse characteristics of the TLB entries

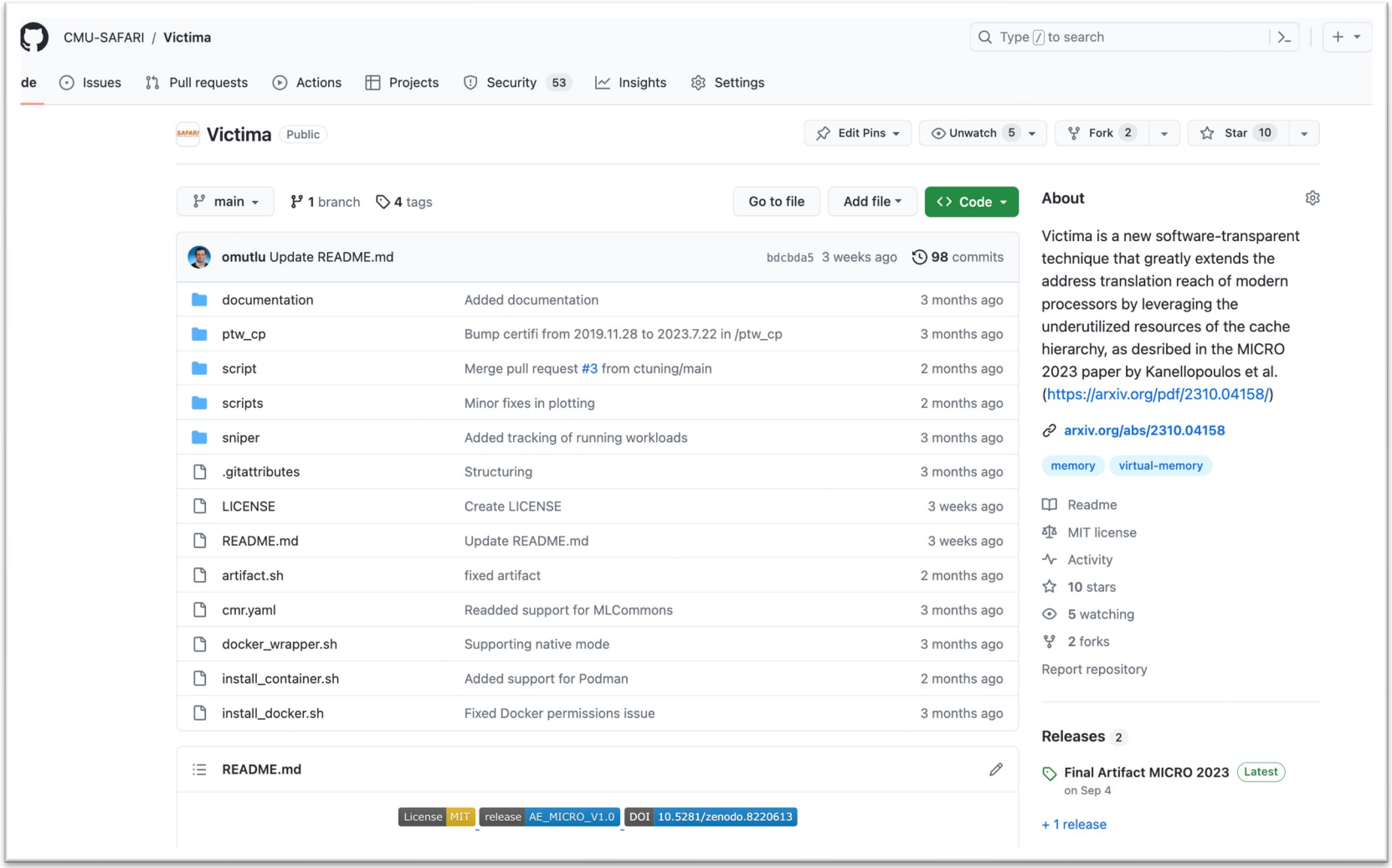
address translations. However, with the very large data footprints of modern workloads, the last-level TLB (L2 TLB) experiences high miss rate (misses per kilo instructions; MPKI), leading to high-latency page table walks (PTWs) that negatively impact application performance. Virtualized environments exacerbate the PTW latency as they impose two-level address translation (e.g., up to 24 memory accesses can occur during a PTW in a system with nested paging [12, 13]), resulting in even higher address translation overheads compared to native execution environments. Therefore, it is crucial to increase the *translation reach* (i.e., the maximum amount of memory that can be covered by the processor's TLB hierarchy) to improve the effectiveness of TLBs and thus minimize PTWs. Doing so becomes increasingly important as PTW latency continues to rise with modern processors' deeper multi-level page table (PT) designs (e.g., 5-level radix PT in the latest Intel processors [4]).

Previous works have proposed various solutions to reduce the high cost of address translation and increase the translation reach of the TLBs such as employing (i) large hardware TLBs [14–16] or (ii) backing up the last-level TLB with a large software-managed TLB [17–25]. Unfortunately, both solutions have significant drawbacks: increased access latency, power, and area (for hardware TLBs), and costly memory accesses, the need for large contiguous memory blocks, and complex OS modifications (for software-managed TLBs).

Drawback of Large Hardware TLBs. First, a larger TLB has

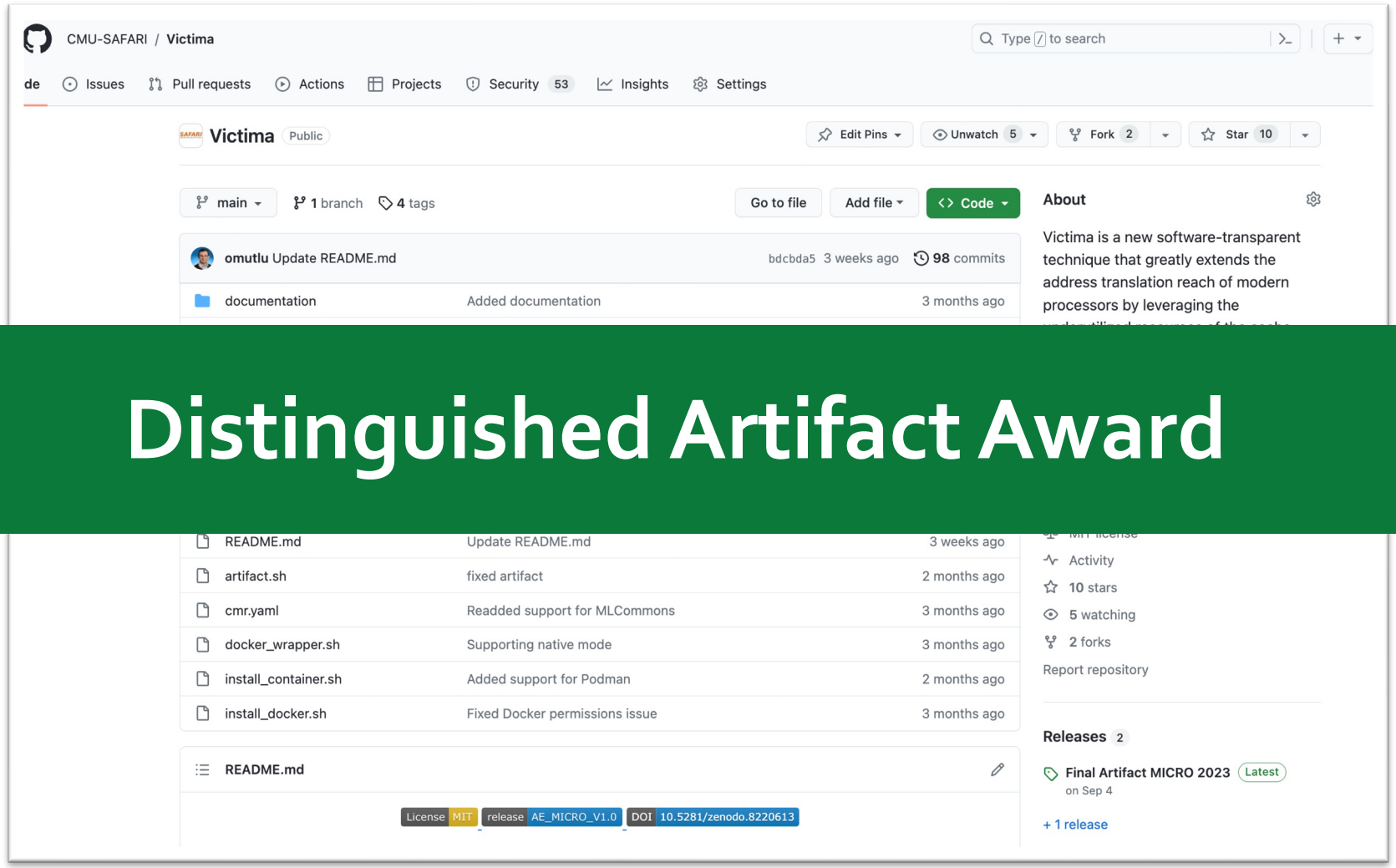
https://arxiv.org/abs/2310.04158

Victima is Open Source



https://github.com/CMU-SAFARI/Victima

Victima is Open Source



Distinguished Artifact Award

https://github.com/CMU-SAFARI/Victima

Victima is Open Source

Documentation is available

TLB Lookup Model

Modifications to the TLB lookup function to implement Victima.

TLB Allocation Model

Modifications to the TLB allocation function to implement Victima.

TLB::lookup

/common/core/memory_subsystem/parametric_dram_directory/tlb.cc

```
bool hit = m_cache.accessSingleLineTLB(address, Cache::LOAD, NULL, 0, now, true);
```

We call the accessSingleLineTLB function of the cache that acts as a TLB. This function is defined in cache.cc. It is used to access a single line in the TLB. It takes as parameters the address, the memory operation type (LOAD, STORE), the data buffer, the data length, the time and the memory model. It returns a boolean value that indicates whether the TLB access was a hit or a miss. We set the modeled parameter to true because we want to model the TLB access. We set the data buffer and the data length to NULL and 0 because we don't need them. We set the memory operation type to LOAD because we are loading the data from the TLB. We set the address to the address of the page table entry. We set the hit variable to the return value of the function.

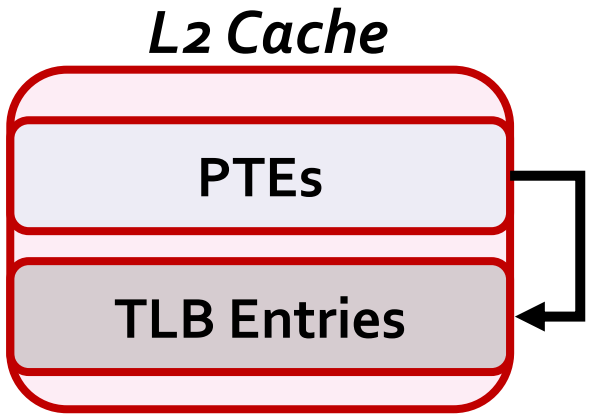
```
bool l2tlb_miss = true;

if (m_next_level) // is there a second level TLB?
{

    where_next = m_next_level->lookup(address, now, false , 2 /* no allocation */,model_count, lock_signal);
    if( where_next != TLB::MISS)
        l2tlb_miss = false;
}
else if(victima_enabled){ // We are at L2 TLB

    //L2 TLB Miss - > check the cache hierarchy to see if TLB entry is cached
        UInt32 set;
        IntPtr tag;

        IntPtr cache_address = address >> (page_size - 3);
        Cache* l1dcache = m_manager->getCache(MemComponent::component_t::L1_DCACHE);
        Cache* l2cache = m_manager->getCache(MemComponent::component_t::L2_CACHE);
        Cache* nuca = m_manager->getNucaCache()->getCache();


        CacheBlockInfo* cb_l1d = l1dcache->peekSingleLine(cache_address);
        CacheBlockInfo* cb_l2 = l2cache->peekSingleLine(cache_address);
        CacheBlockInfo* cb_nuca = nuca->peekSingleLine(cache_address);
```

In case of an L2 TLB miss, we check if the TLB entry is cached in the cache hierarchy in case Victima is enabled. In order to do that, we first calculate the address of the cache line that contains the TLB entry. We do that by shifting the address to the right by the page size minus 3 bits (8 PTEs are stored in the cache line). We then get the L1 data cache, the L2 cache and the NUCA cache from the memory manager. We then peek the cache line that contains the TLB entry from each cache. In the case of Victima, we only need to check the L2 cache and the NUCA cache.

Conclusion

We present Victima, a new software-transparent scheme that drastically increases the translation reach of the processor's TLB hierarchy by leveraging the underutilized cache resources

Key idea: Transform L2 cache blocks that store PTEs into blocks that store TLB entries



Key Results: Victima (i) **outperforms by 5.1%** a state-of-the-art software-managed TLB and (ii) achieves similar performance to an optimistically fast 128K-entry L2 TLB design **without** the associated area and power overheads

https://github.com/CMU-SAFARI/Victima

Arxiv

VICTIMA

Drastically Increasing Translation Reach by Leveraging Underutilized Cache Resources

https://github.com/CMU-SAFARI/Victima

Konstantinos Kanellopoulos
Hong Chul Nam, Nisa Bostanci, Rahul Bera, Mohammad Sadrosadati,
Rakesh Kumar, Davide Basilio Bartolini and Onur Mutlu

SAFARI
SAFARI Research Group
safari.ethz.ch

ETH zürich

HUAWEI

NTNU

Virtuoso

An Open-Source, Comprehensive and Modular Simulation Framework for Virtual Memory Research

3rd place @ MICRO 2023 SRC

Konstantinos Kanellopoulos

Konstantinos Sgouras Onur Mutlu

SAFARI Research Group
safari.ethz.ch

ETH zürich

HUAWEI

NTNU

Improving Virtual Memory

- Virtual memory causes high **performance overheads**

- Various academic and industrial solutions were proposed to **reduce these overheads**

Virtual Memory (VM) Solutions

Improving the TLB Subsystem

Employing Large Pages

Leveraging Contiguity

Rethinking Page Tables

Reducing Page Fault Latency

Employing Better Address Mappings

Evaluating VM Solutions

Effectively evaluating VM techniques is crucial for progress in the domain

Evaluation Requirements

Flexibility to model many VM schemes

Interactions between VM components

Impact of VM techniques on system

Evaluation Requirements

Flexibility to model many VM schemes

Interactions between VM components

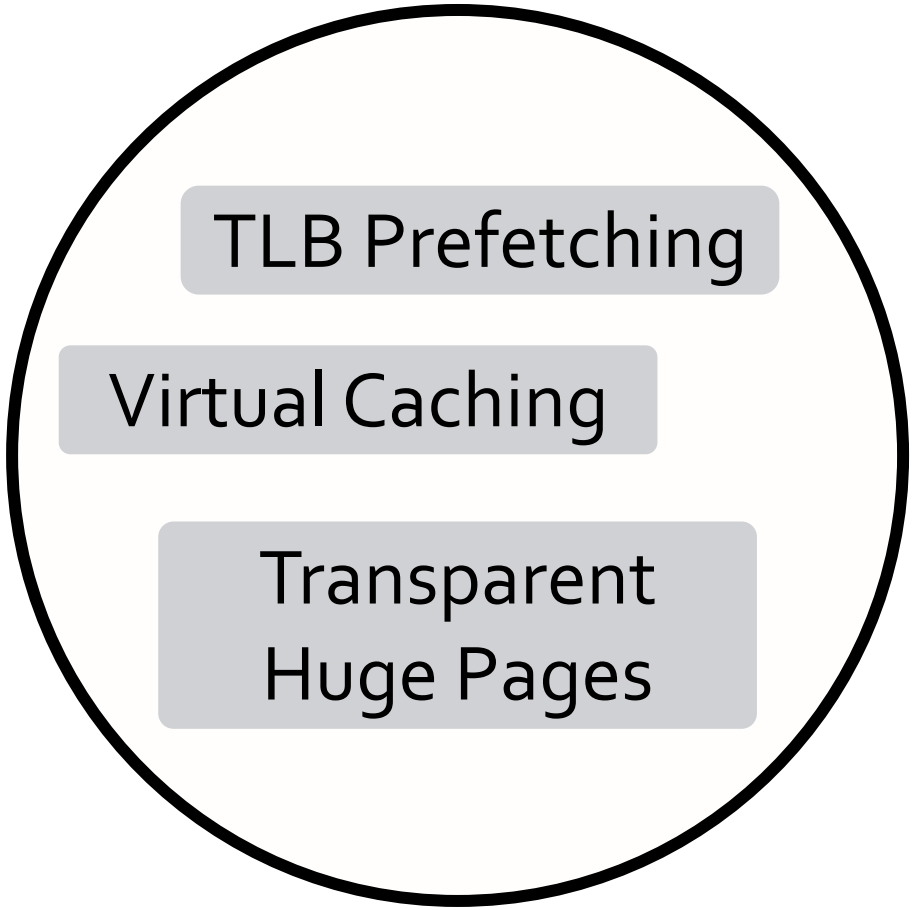
Impact of VM techniques on system

Evaluating Various VM Schemes

New idea on VM

vs.

Set of comparison points



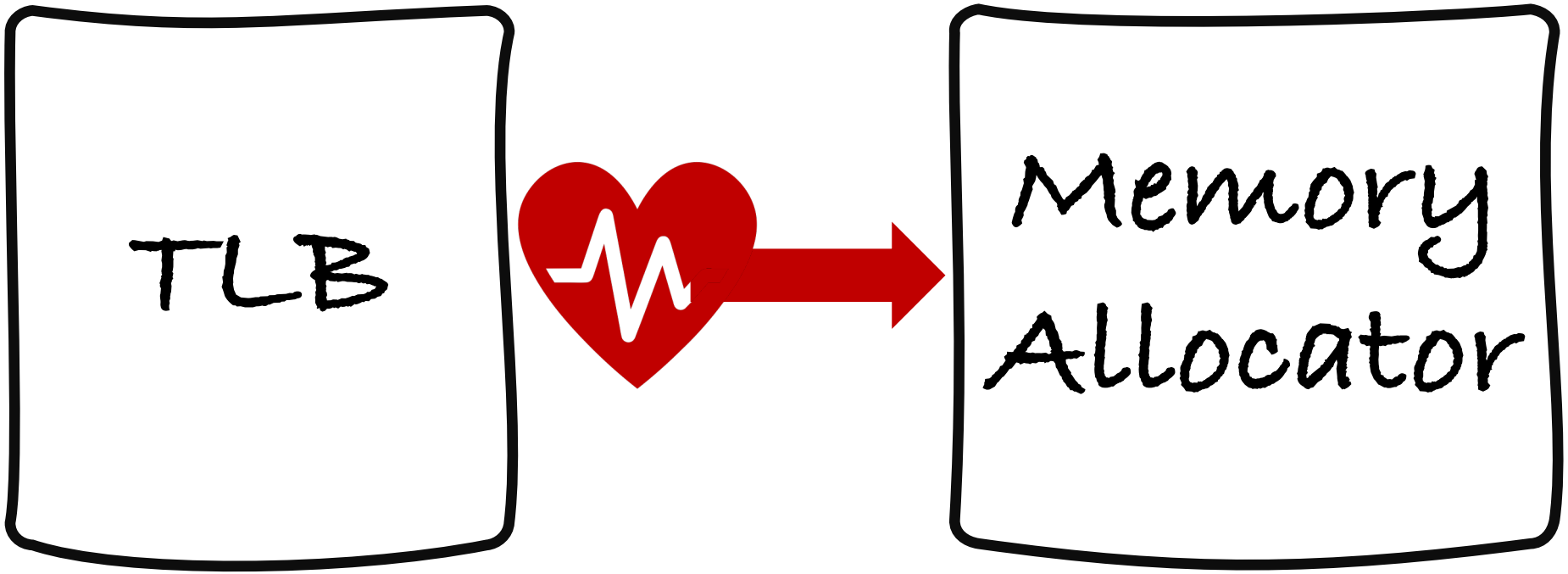
Evaluation Requirements

Flexibility to model many VM schemes

Interactions between VM components

Impact of VM techniques on system

Example: TLB and Memory Allocator



TLB performance heavily depends on the memory allocator (e.g., # of 2MB pages)

Evaluation Requirements

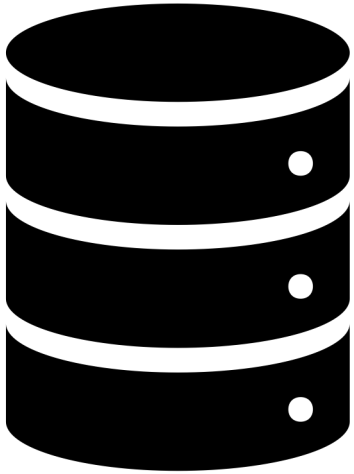
Flexibility to model many VM schemes

Interactions between VM components

Impact of VM techniques on system

Example: Main Memory Interference

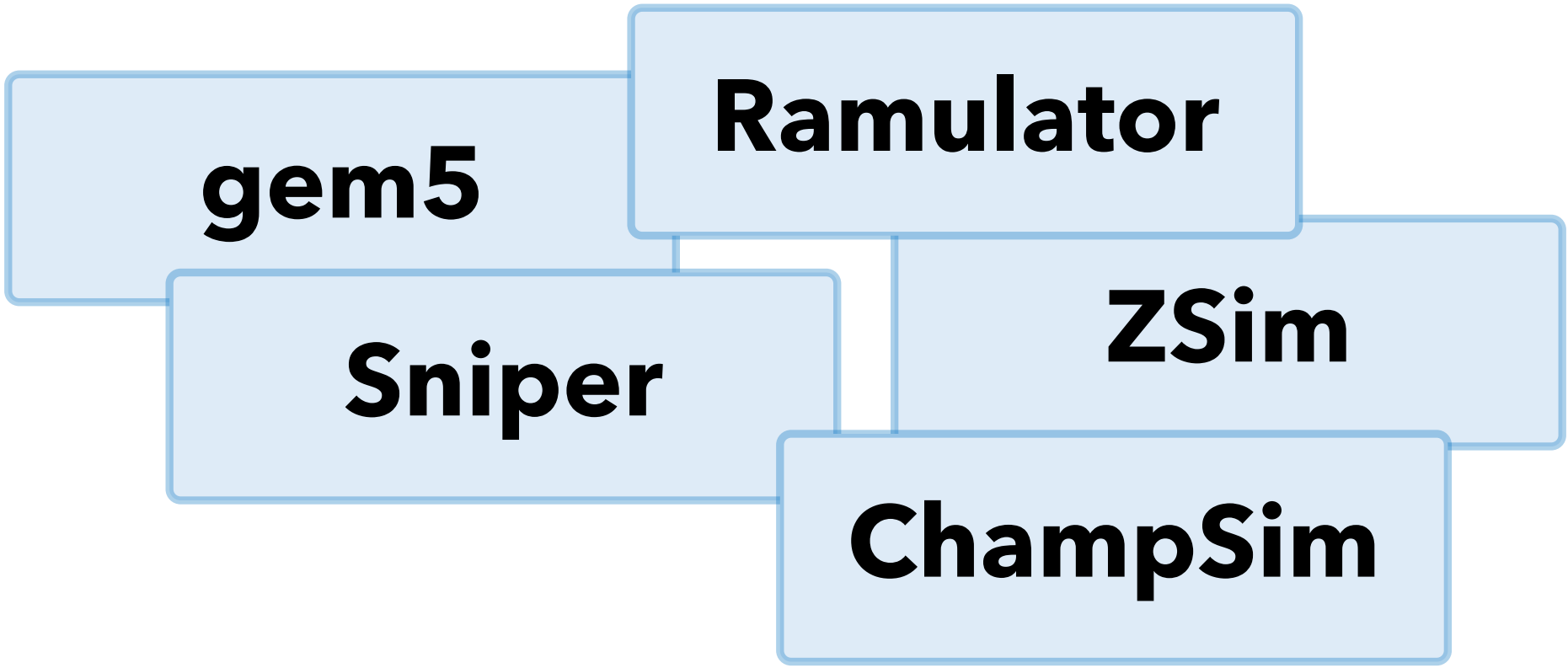
New Page Table



Low Latency

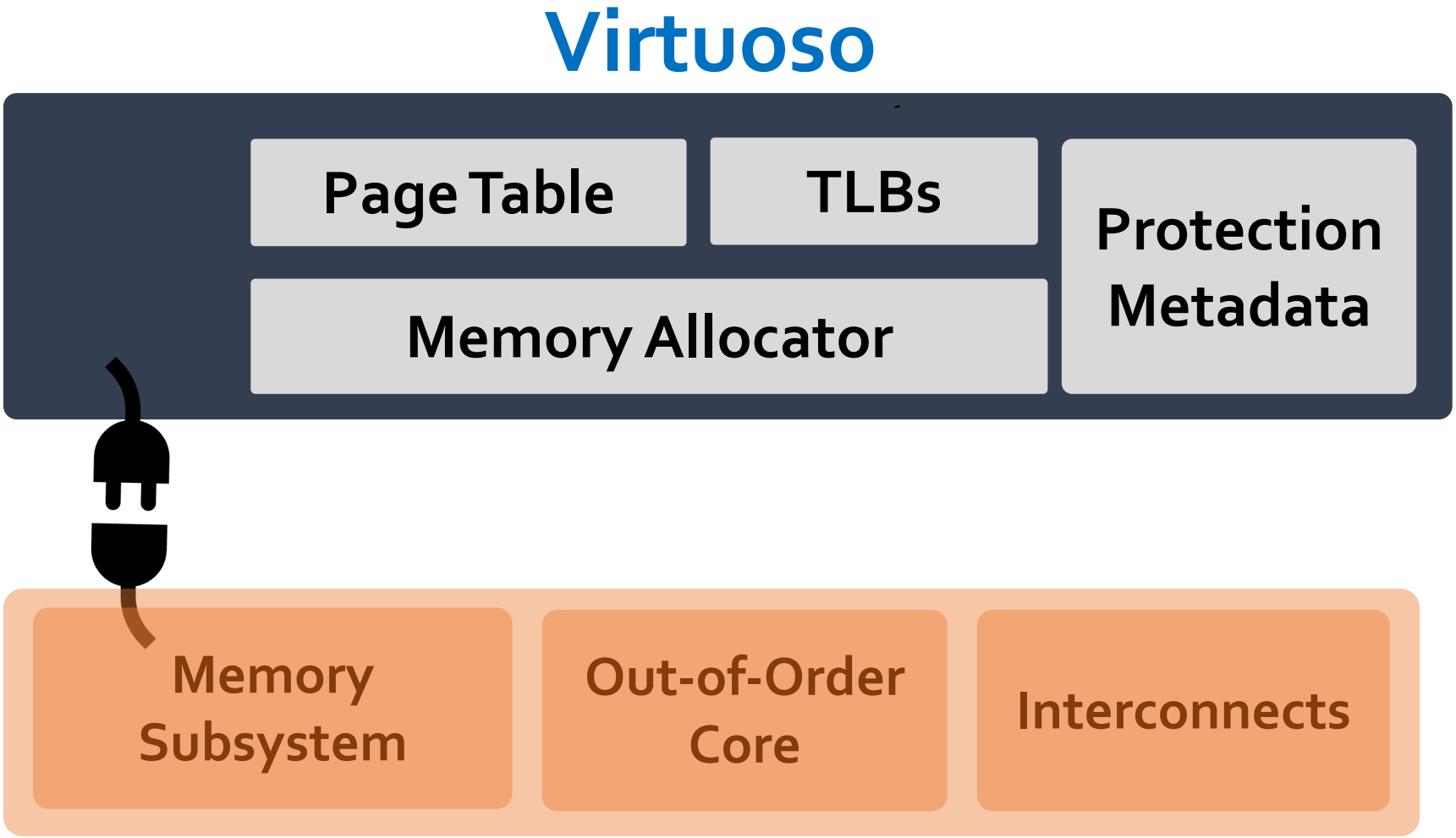
High Interference

Lack of Comprehensive Simulators



Modern simulators lack the capability to model a wide range of state-of-the-art VM techniques

Our Approach: Virtuoso



Example: Sniper Multicore Simulator[1]

[1] Carlson et al. "An Evaluation of High-Level Mechanistic Core Models" TACO 2014

Virtuoso: Key Benefits

Comprehensive

Modular

Open-Source

https://github.com/CMU-SAFARI/Virtuoso

Virtuoso: Key Benefits

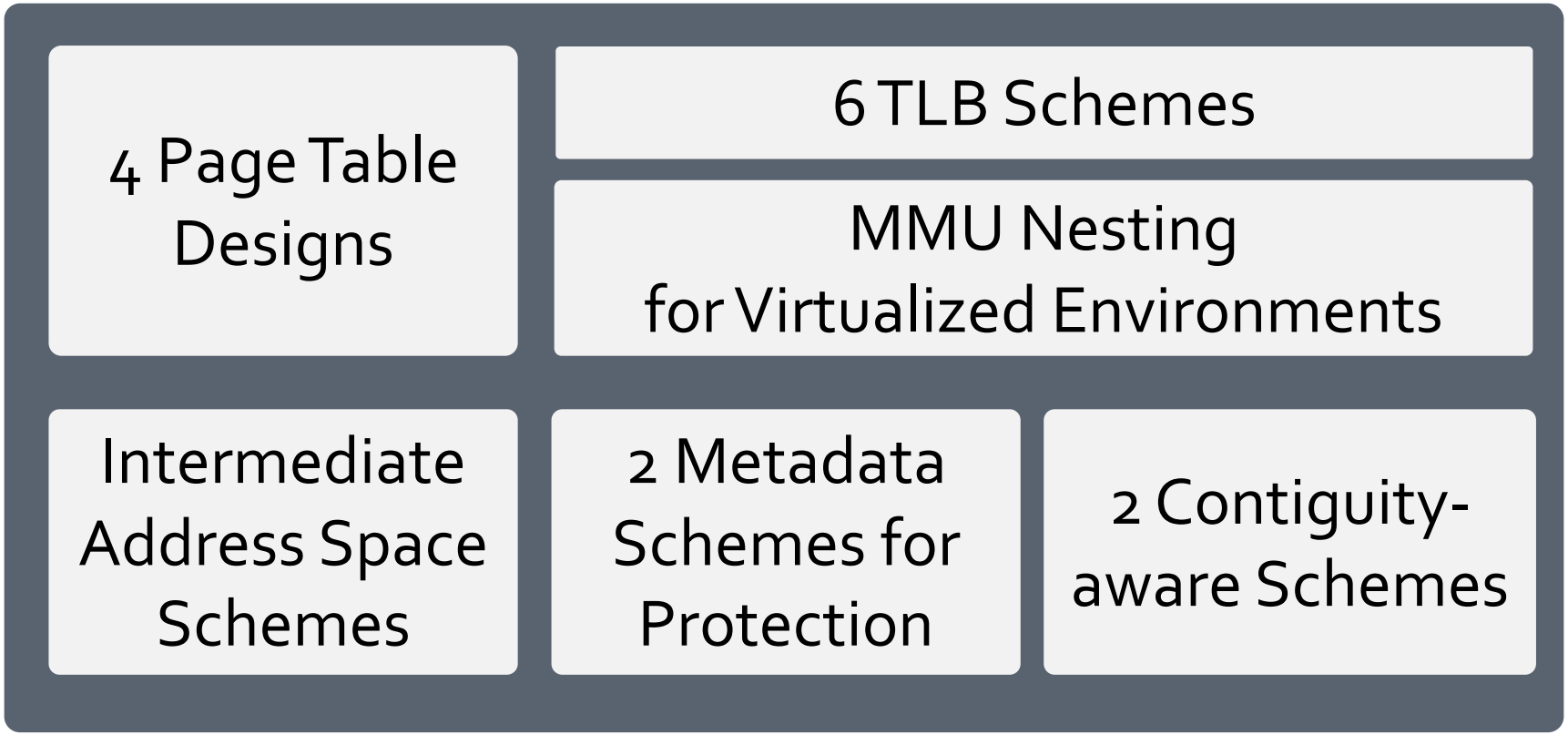
Comprehensive

Modular

Open-Source

https://github.com/CMU-SAFARI/Virtuoso

Virtuoso: Tool Set



https://github.com/CMU-SAFARI/Virtuoso

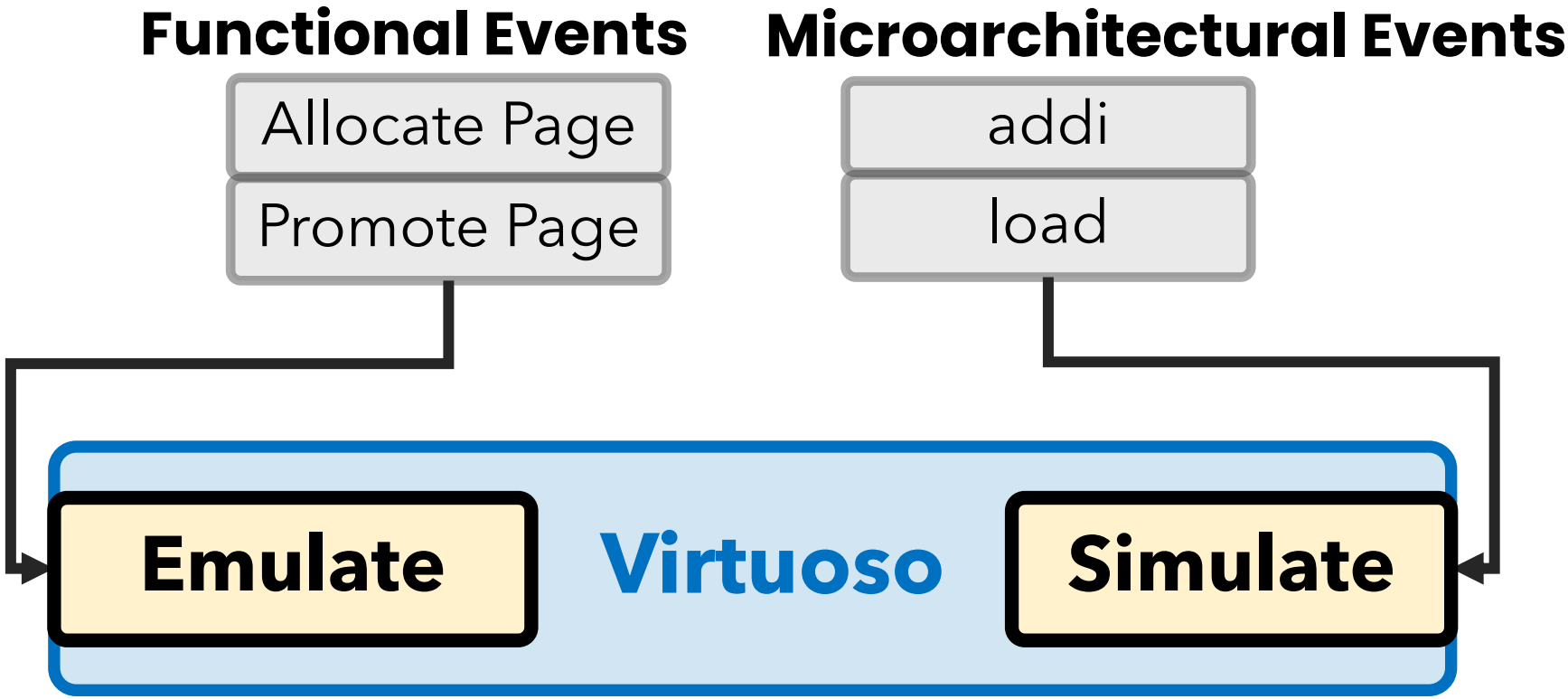
VirtuOS: Mini-OS for Memory Management

What about the OS?

Challenge: We need to both **simulate** and **emulate** it

VirtuOS: Mini-OS for Memory Management

Example Routine: Page Fault Handler



Virtuoso: Key Benefits

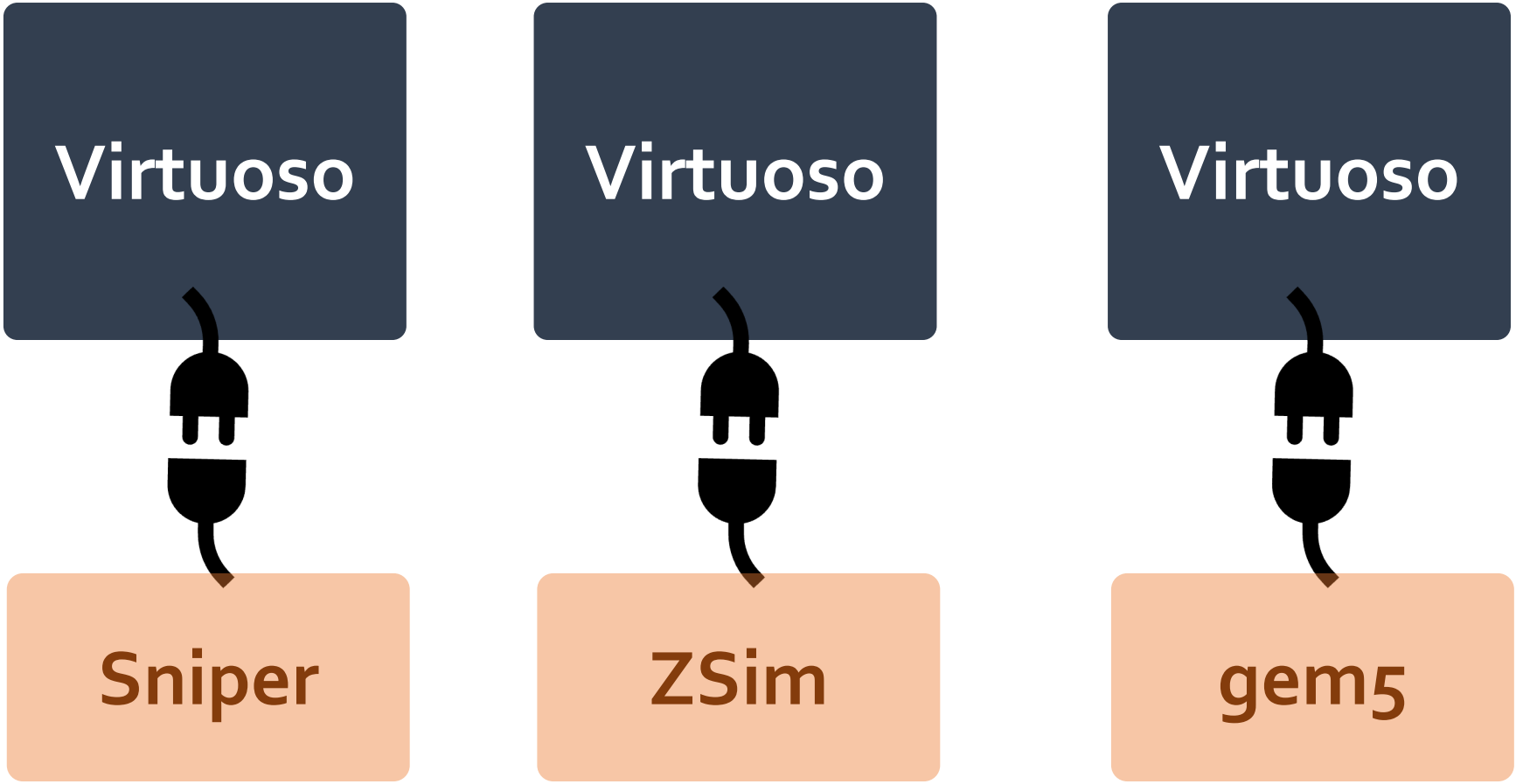
Comprehensive

Modular

Open-Source

https://github.com/CMU-SAFARI/Virtuoso

Virtuoso: Modularity



Virtuoso: Key Benefits

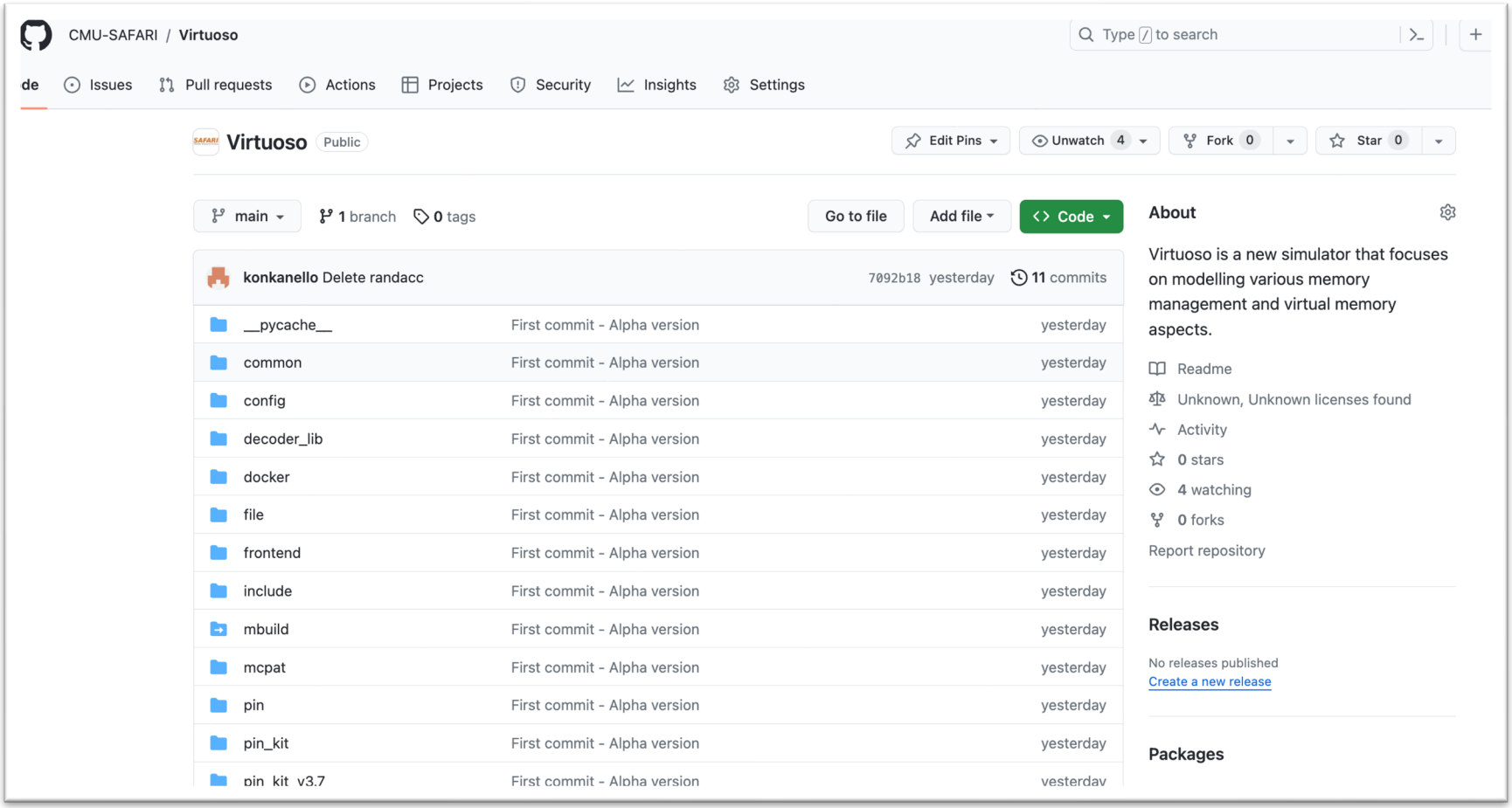
Comprehensive

Modular

Open-Source

https://github.com/CMU-SAFARI/Virtuoso

Virtuoso is Open Source



https://github.com/CMU-SAFARI/Virtuoso

Virtuoso Example Use Cases

Comparing MMU Designs

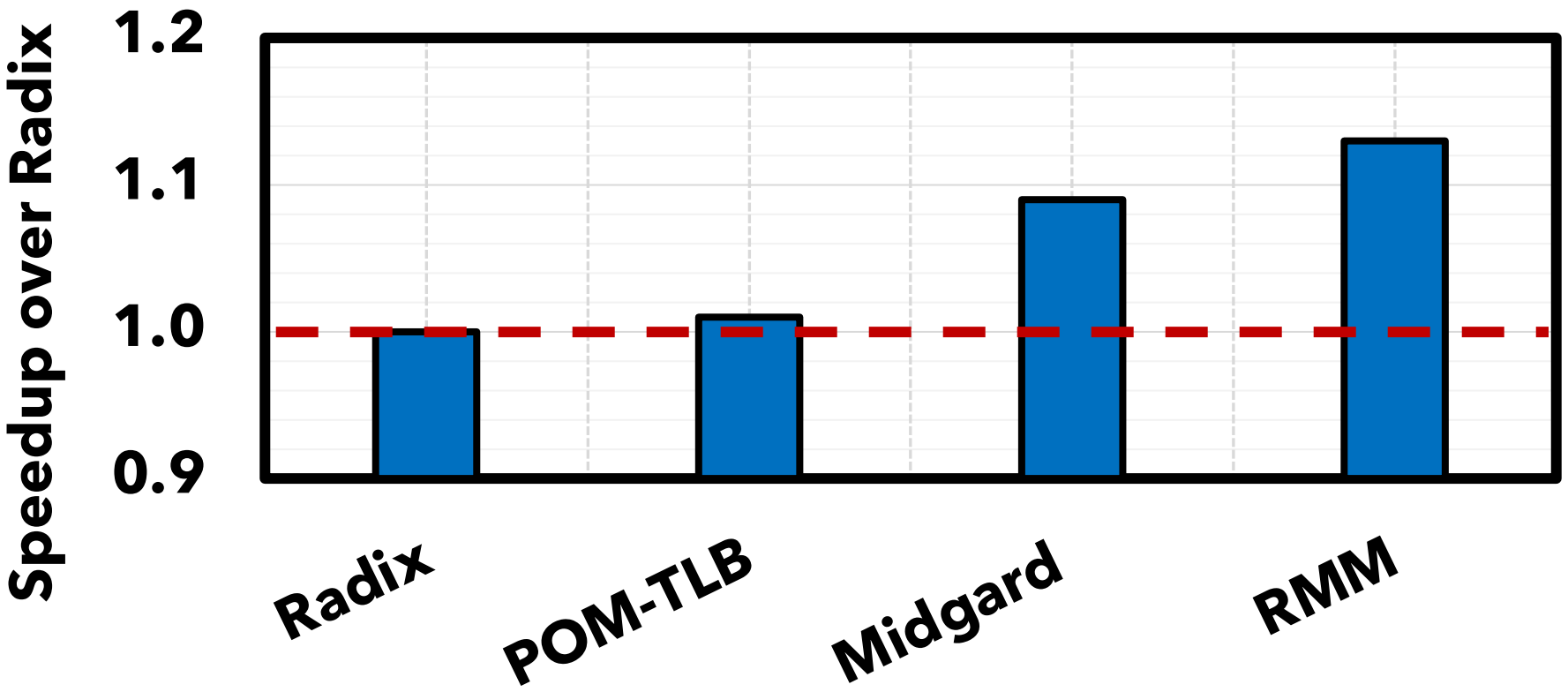
Effect of THP on PTW Latency

Virtuoso Example Use Cases

Comparing MMU Designs

Effect of THP on PTW Latency

Comparing MMU Designs



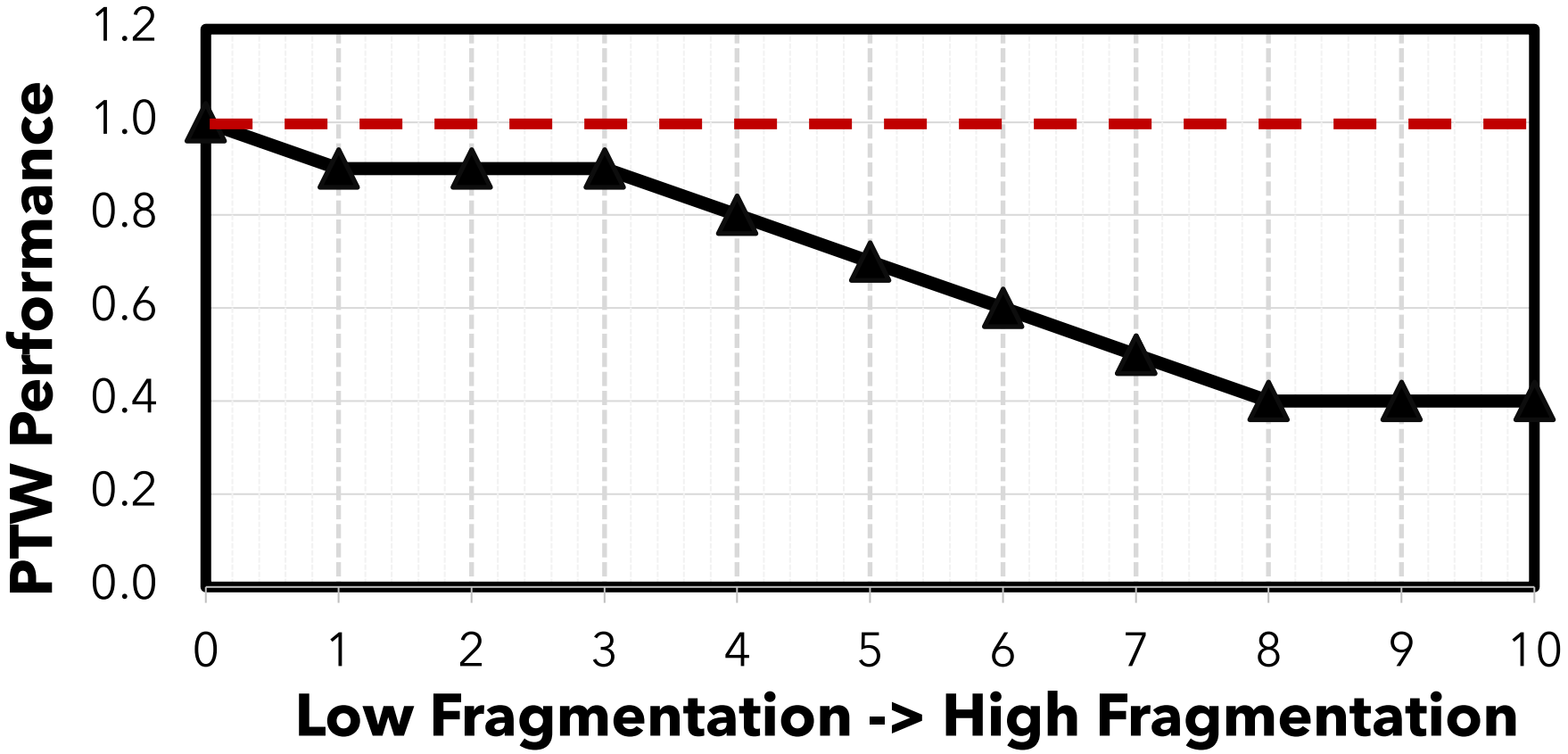
Effective comparison of different MMU designs

Virtuoso Example Use Cases

Comparing MMU Designs

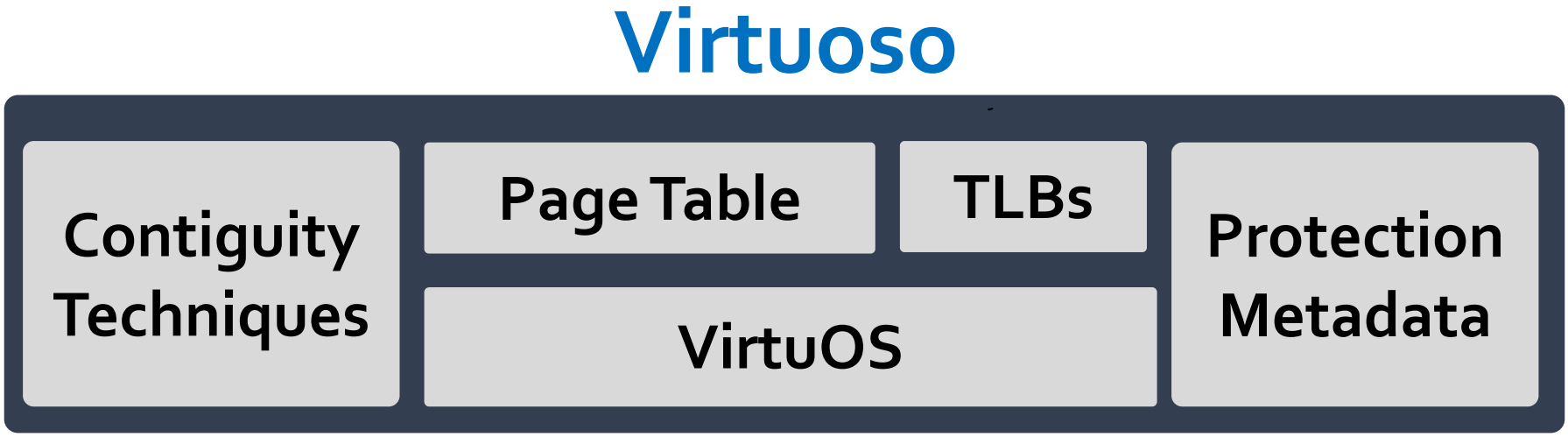
Effect of THP on PTW Latency

Interplay between PTW and THP



Effective evaluation of THP & PTW interactions

Conclusion



Virtuoso is a good start for establishing a common ground for VM research

https://github.com/CMU-SAFARI/Virtuoso

Virtuoso

An Open-Source, Comprehensive and Modular Simulation Framework for Virtual Memory Research

https://github.com/CMU-SAFARI/Virtuoso

Konstantinos Kanellopoulos

Konstantinos Sgouras Onur Mutlu

SAFARI
SAFARI Research Group
safari.ethz.ch

ETH zürich

HELLENIC REPUBLIC
National and Kapodistrian University of Athens
EST. 1837