



# Virtuoso

## Enabling Fast and Accurate Virtual Memory Research with an Imitation-based Operating System Simulation Methodology

**Konstantinos Kanellopoulos**

Konstantinos Sgouras

F. Nisa Bostanci

Andreas Kosmas Kakolyris

Berkin Kerim Konar

Rahul Bera

Mohammad Sadrosadati

Rakesh Kumar

Nandita Vijaykumar

Onur Mutlu

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

**SAFARI**  
SAFARI Research Group  
safari.ethz.ch

**ETH** zürich

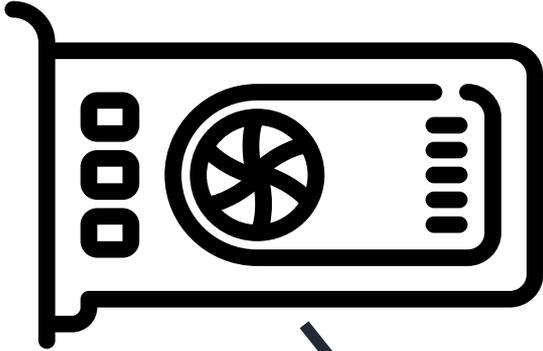


UNIVERSITY OF  
TORONTO

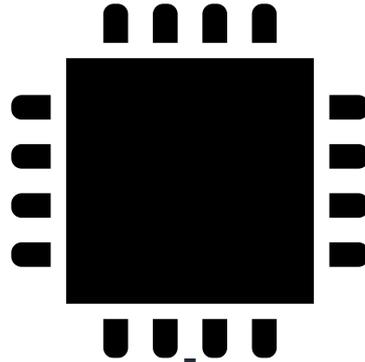


NTNU

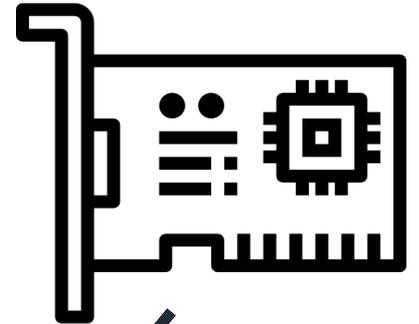
**GPUs**



**CPUs**



**NICs**



**Virtual  
Memory**

GPUs

Accelerator

# **Virtual Memory is a cornerstone of modern computing systems**



**Does Virtual Memory  
come for free?**

# Virtual memory causes high **performance overheads**

**Memory  
Allocation**

**1**

**Address  
Translation**

**2**

# Memory Allocation Overheads

1

## Emerging Workloads



Function-as-a-Service

Short Running (<1s)

Gemini



Short-input Short-output  
Large Language Model Inference

High spawning throughput

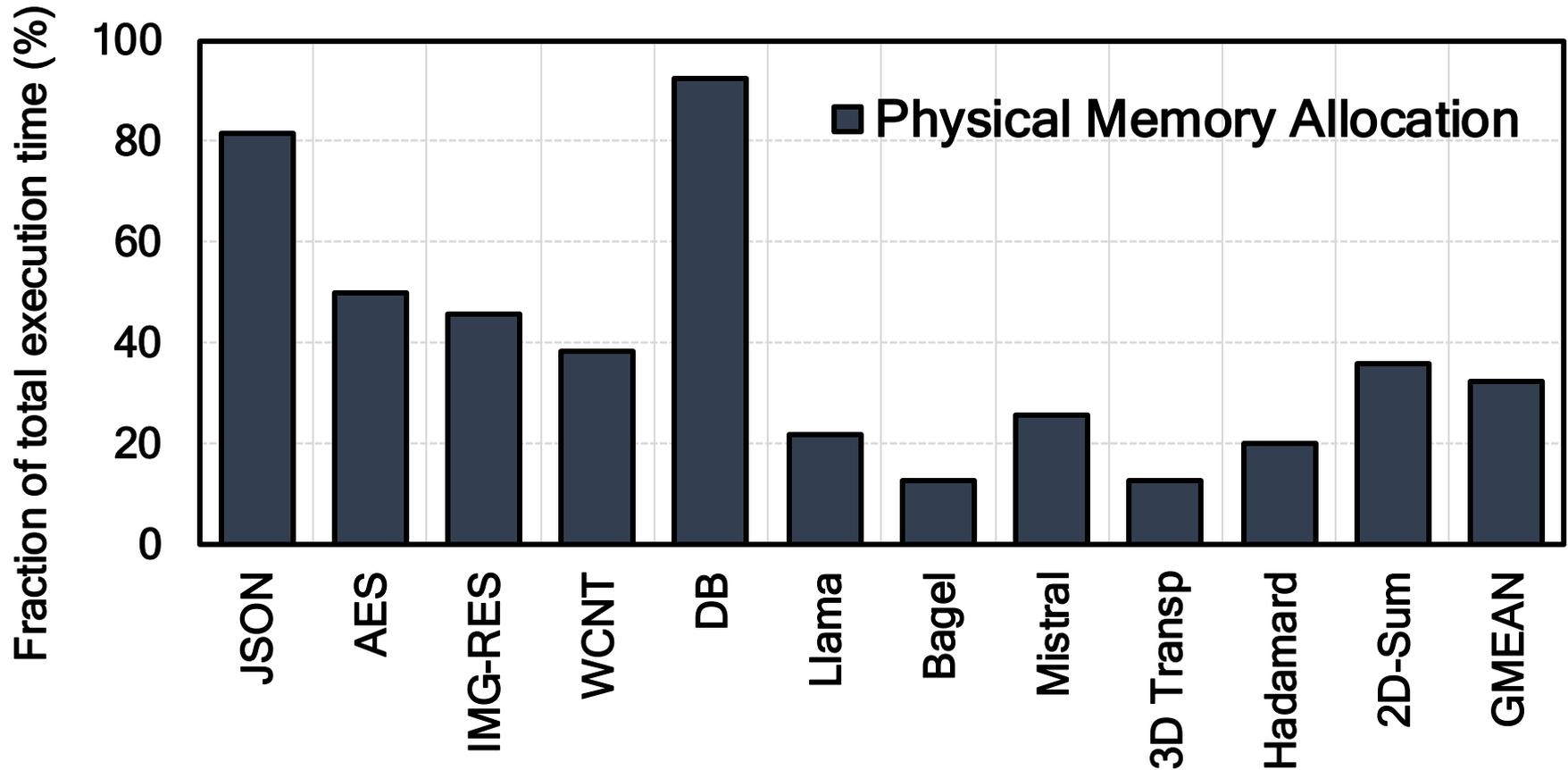
*Time spent in OS cannot be amortized due to short execution*

OS Memory  
Allocation Routines

Execution

Time

# Physical Memory Allocation Overheads

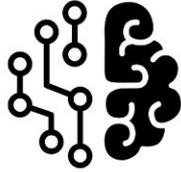


On average 32% of execution time (measured in a real system) spent on physical memory allocation

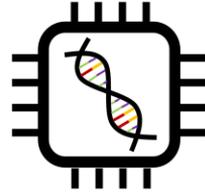
# Address Translation Overheads

2

## Emerging Workloads



Sparse Machine Learning



Bioinformatics



Graph Analytics



Databases

Long Running (>100s)    Irregular Memory Accesses

*High Latency Address Translation*

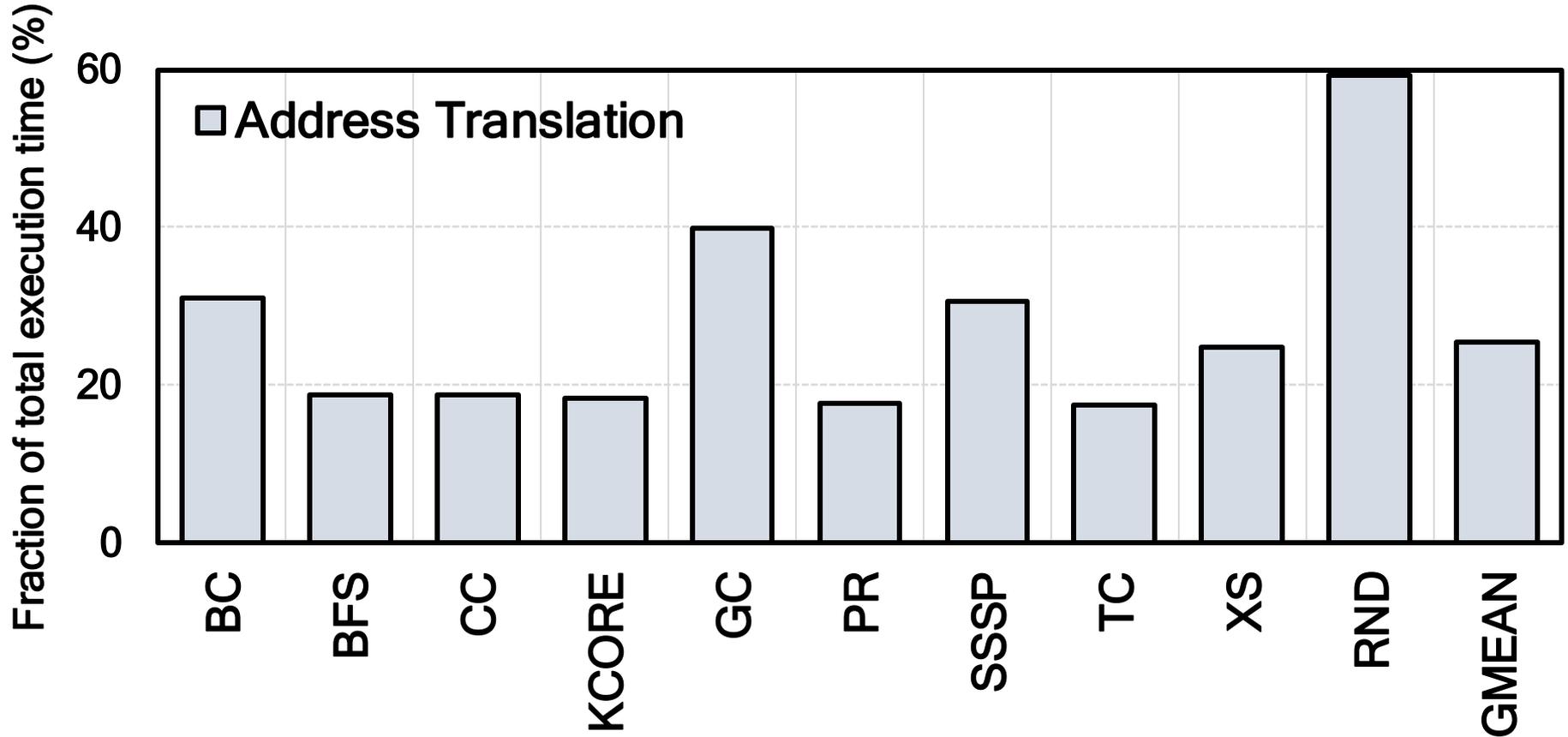
**TLB Miss**

**Page Table Walk**

**Data Fetching**

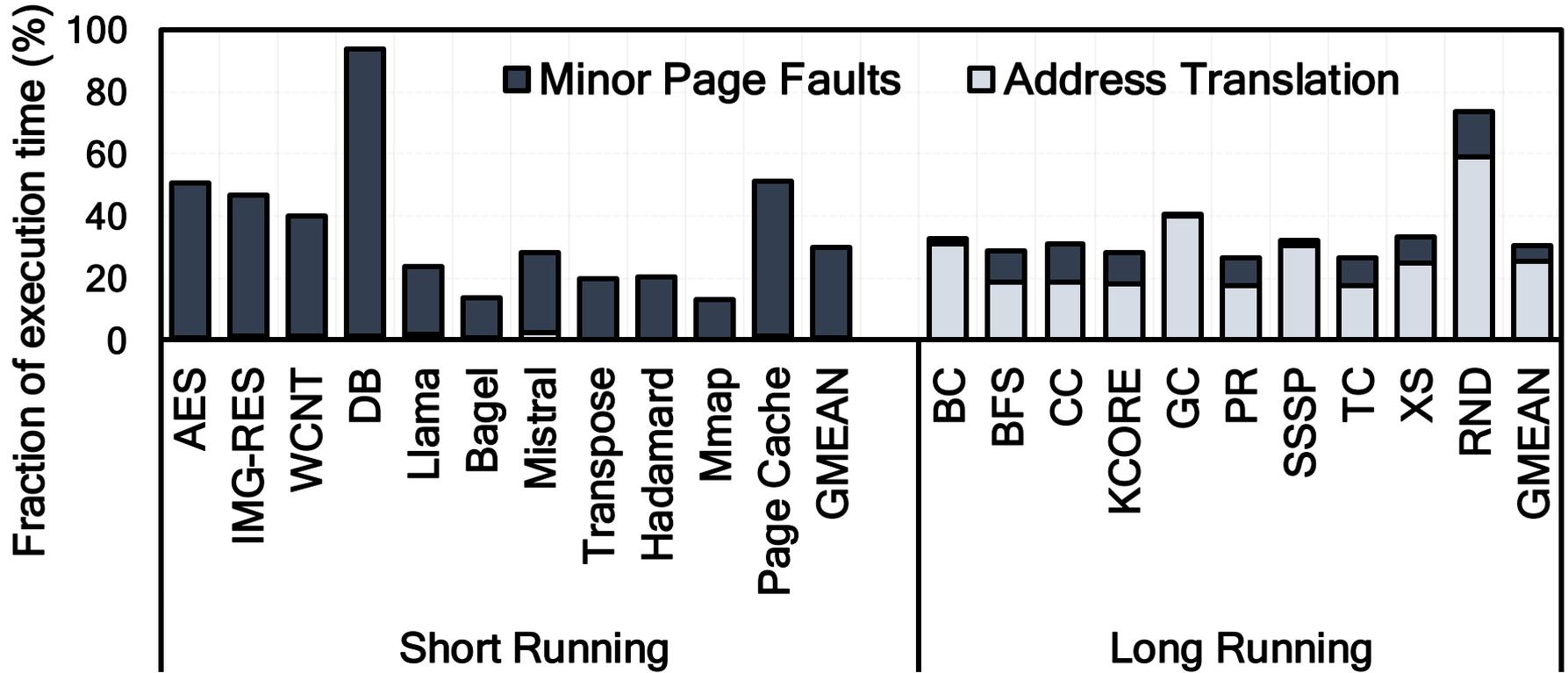
**Time**

# Address Translation Overheads



On average 26% of execution time (measured in a real system) is spent on address translation

# High VM Overheads

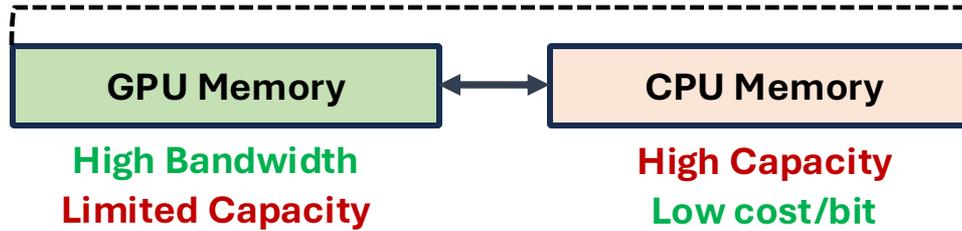


**VM causes high overheads  
in diverse emerging workloads**

Virtual Memory overheads are **expected to increase** as we transition to larger physical address spaces

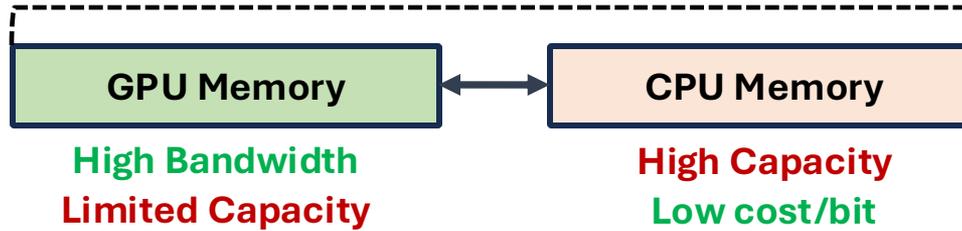
# Going into the Future

## 1 Unified Virtual Memory

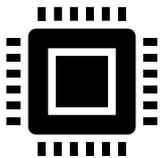


# Going into the Future

## 1 Unified Virtual Memory



## 2 Direct Storage Access

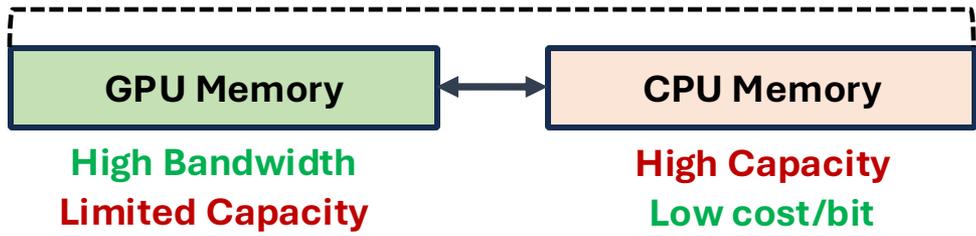


*High-end SSD*

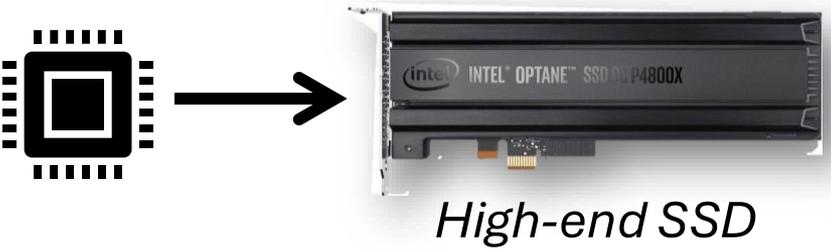
**Byte-addressable  
interface**

# Going into the Future

## 1 Unified Virtual Memory

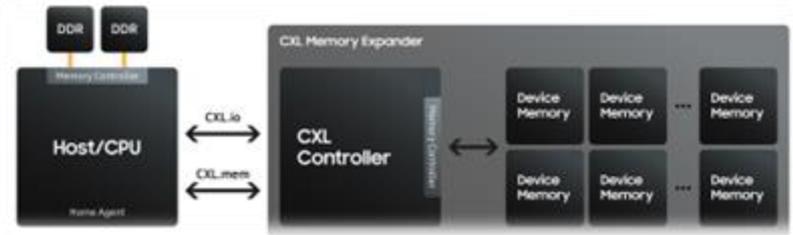


## 2 Direct Storage Access



Byte-addressable interface

## 3 Memory Disaggregation



Access to both local and remote memory modules



**Researchers try to  
save the day**

# Co-design the hardware and the OS to reduce VM overheads

## Software-managed TLB Subsystem

**Nagle+ ISCA 1993**

**Design Tradeoffs for Software-Managed TLBs**

**Ryoo+ ISCA 2017**

**Rethinking TLB Designs in Virtualized Environments:  
A Very Large Part-of-Memory TLB**

**Marathe+ MICRO 2017**

**CSALT: Context Switch Aware Large TLB**

**Jaleel+ TACO 2019**

**DUCATI: High-performance Address Translation  
by Extending TLB Reach of GPU-accelerated Systems**

# Co-design the hardware and the OS to reduce VM overheads

## Leveraging VA-to-PA Contiguity

**Pham+ MICRO 2012**

**CoLT: Coalesced Large-Reach TLBs**

**Karakostas\*,Gandhi\*+ ISCA 2015**

**Redundant Memory Mappings for Fast Access to Large Memories**

**Ausavarungnirun+ MICRO 2017**

**Mosaic: An Application-Transparent  
Hardware-Software Cooperative Memory Manager for GPUs**

**Zhao+ ISCA 2022**

**Contiguitas: The Pursuit of Physical Memory Contiguity in  
Datacenters**

Software-managed  
TLB Subsystem

# Co-design the hardware and the OS to reduce VM overheads

## Accelerating Memory Allocation

**Lee+ ISCA 2020**

A Case for Hardware-Based Demand Paging

**Tirumalasetty+ TACO 2022**

Reducing Minor Page Fault Overheads through Enhanced Page Walker

**Wang+ MICRO 2023**

Memento: Architectural Support for Ephemeral Memory Management in Serverless Environments

Software-managed  
TLB Subsystem

Leveraging VA-to-  
PA Contiguity

# Co-design the hardware and the OS to reduce VM overheads

## Alternative Address Mappings

**Picorel+ PACT 2016**

**Near-Memory Address Translation**

**Gosakan+ ASPLOS 2023**

**Mosaic Pages: Big TLB Reach with Small Pages**

**Kanellopoulos+ MICRO 2023**

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

Software-managed  
TLB Subsystem

Leverage VA-to-  
PA Contiguity

Accelerating  
Memory  
Allocation

# Co-design the hardware and the OS to reduce VM overheads

Rethinking the Page Table Design

Employing Multiple Page Sizes

**and more...**

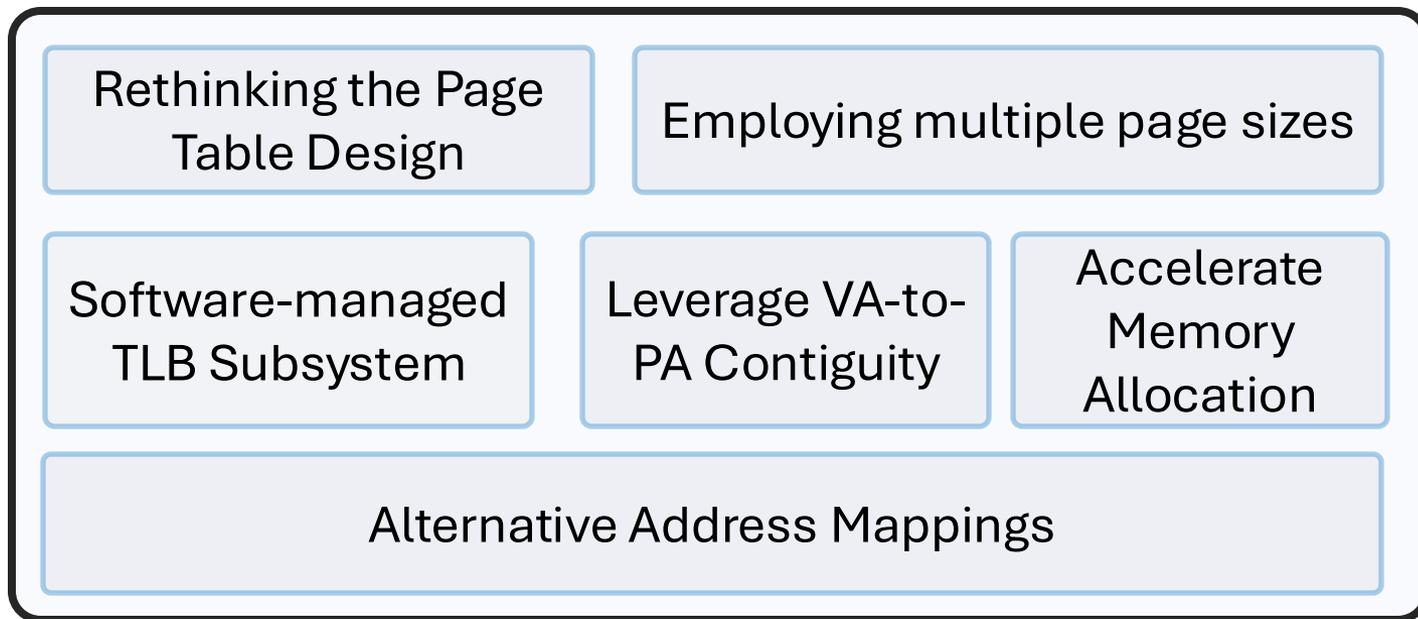
Software-managed  
TLB Subsystem

Leverage VA-to-  
PA Contiguity

Accelerating  
Memory  
Allocation

Alternative  
Address Mappings

# Co-design the hardware and the OS to reduce VM overheads

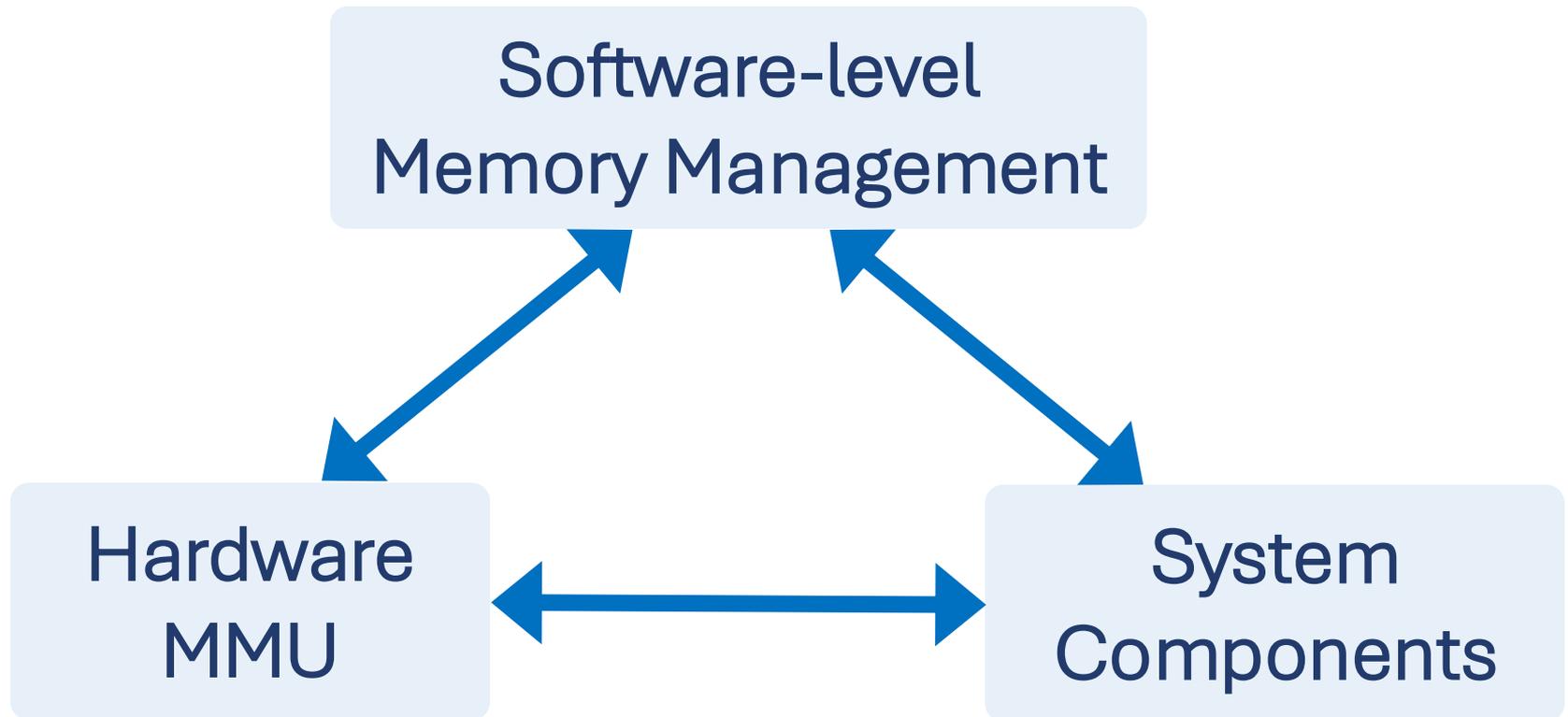


**Wide range of hardware-OS co-design techniques that aim to improve virtual memory**

**Effectively evaluating VM techniques  
is crucial for progress in the domain**

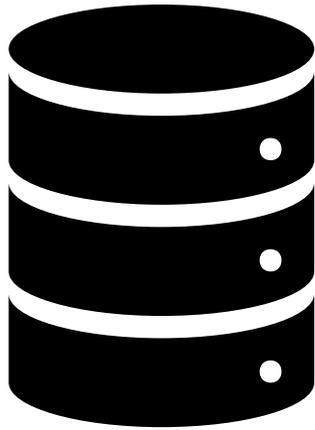
# Challenges in Evaluation

## Interplay between system components

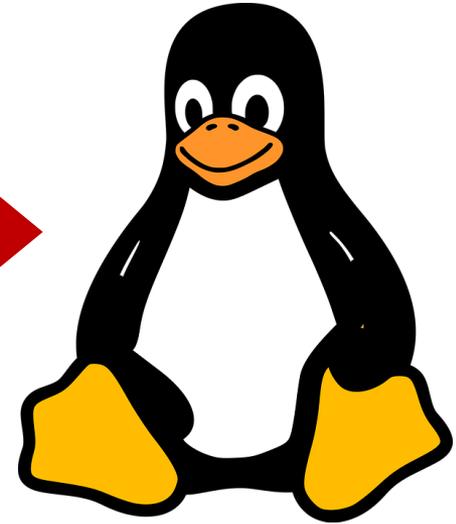


# Example: Page Table and Large Pages

**Page Table  
Size**



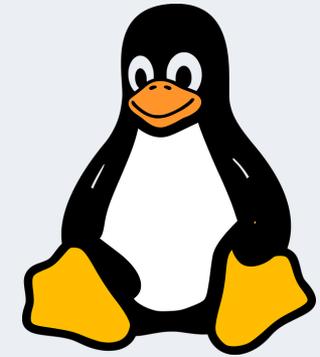
**OS Allocation  
Policy**



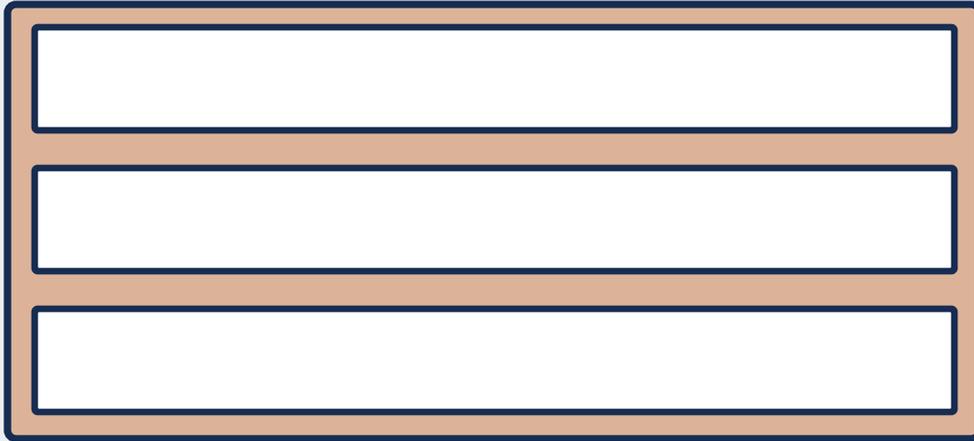
**Size of the page table depends on the number of large pages provided by the OS**

# Example: Page Table and Large Pages

OS Kernel

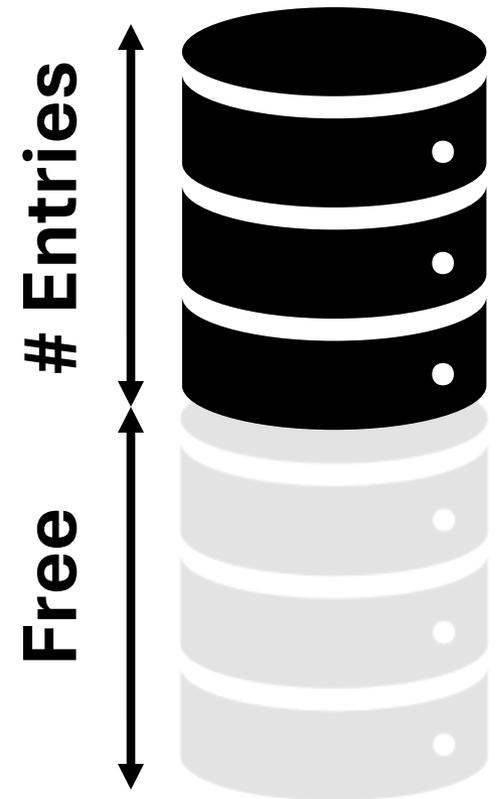


*"I can provide plenty of large pages"*



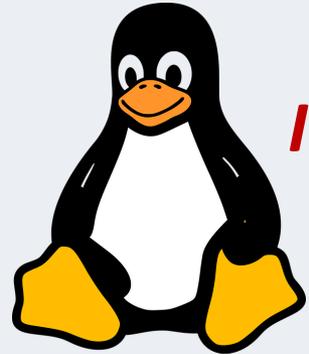
**2MB Pages**

**Page Table**

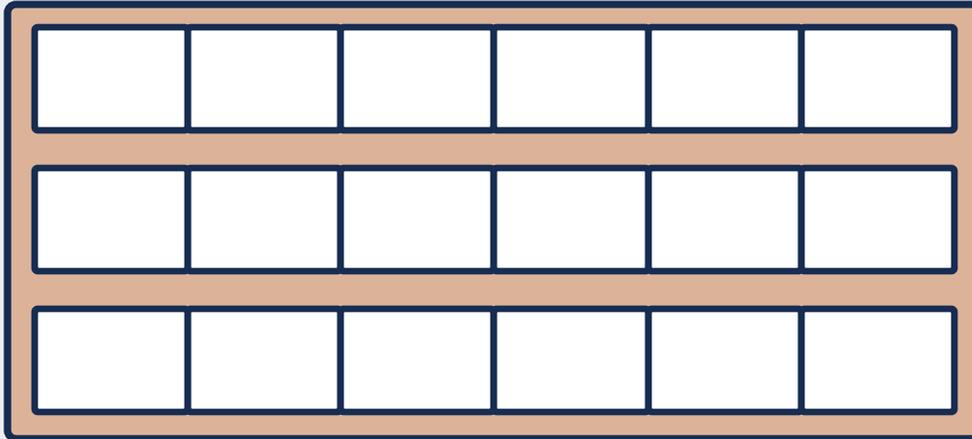


# Example: Page Table and Large Pages

OS Kernel

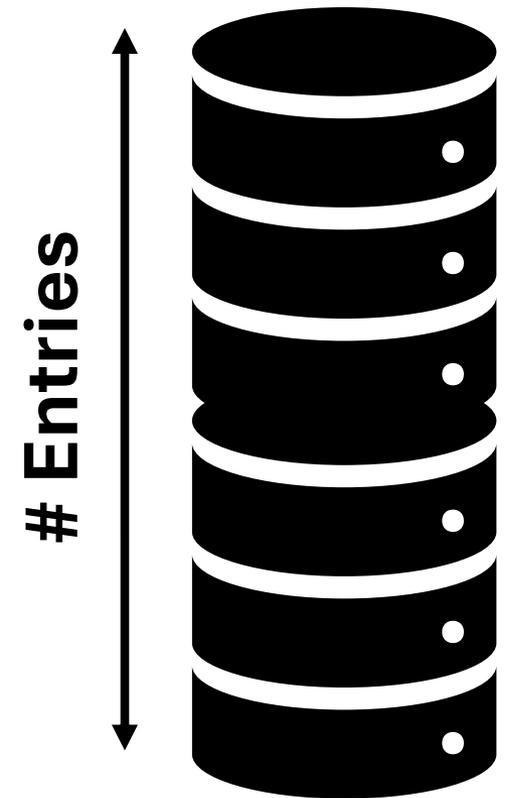


*“Fragmentation is rising so  
I cannot allocate 2MB pages”*



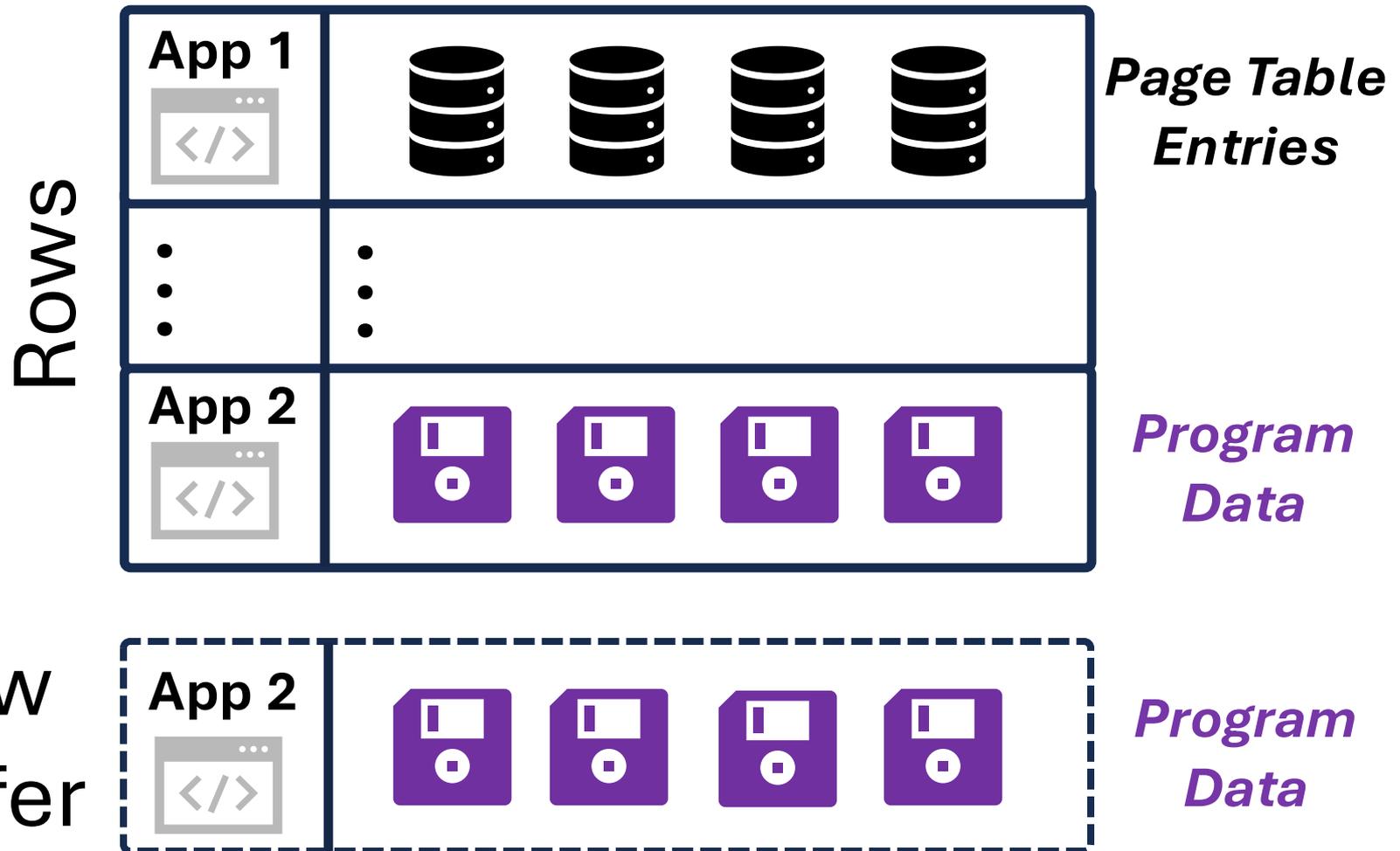
**4KB Pages**

**Page Table**



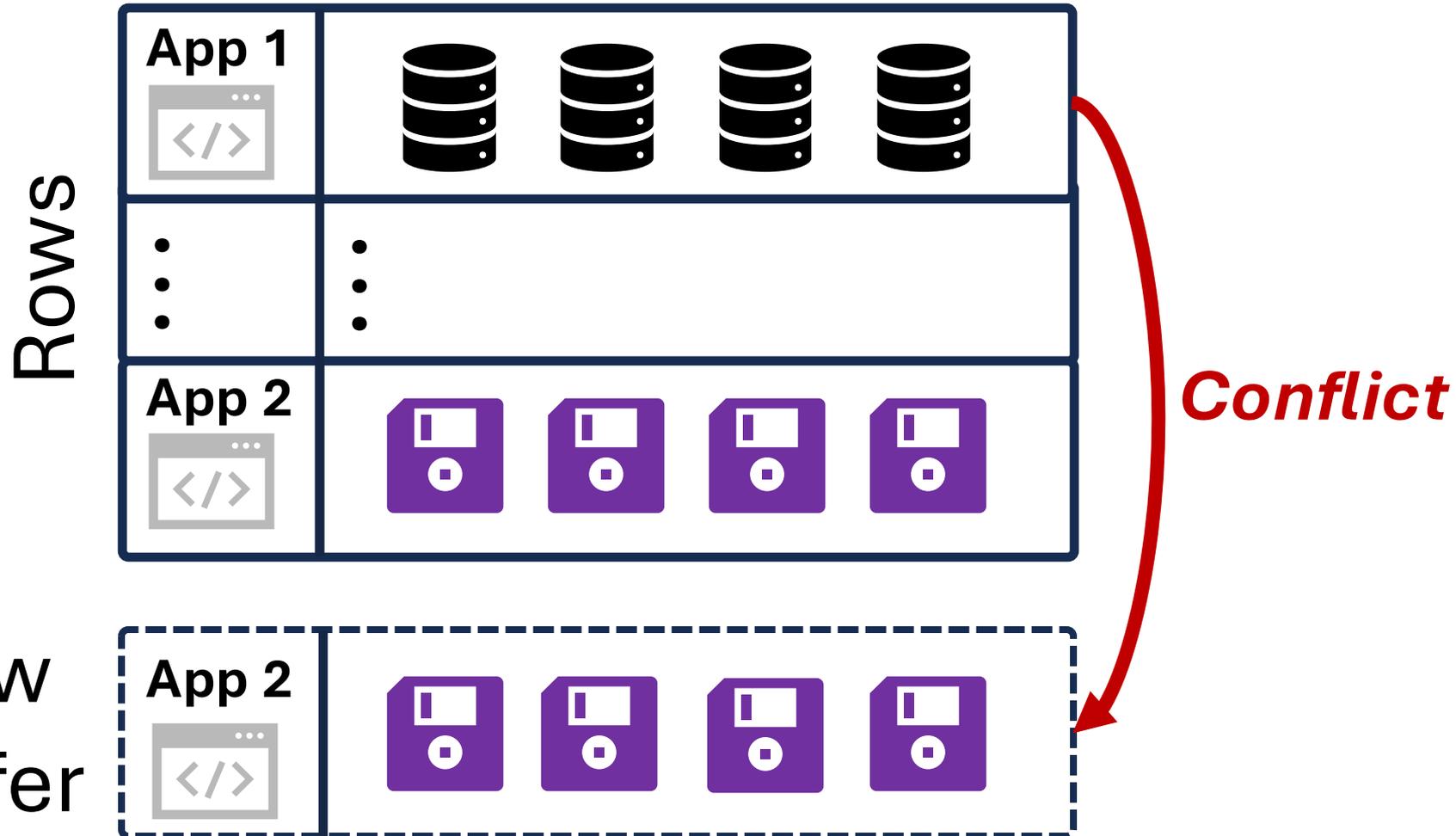
# Page Table and Memory Interference

## DRAM Bank



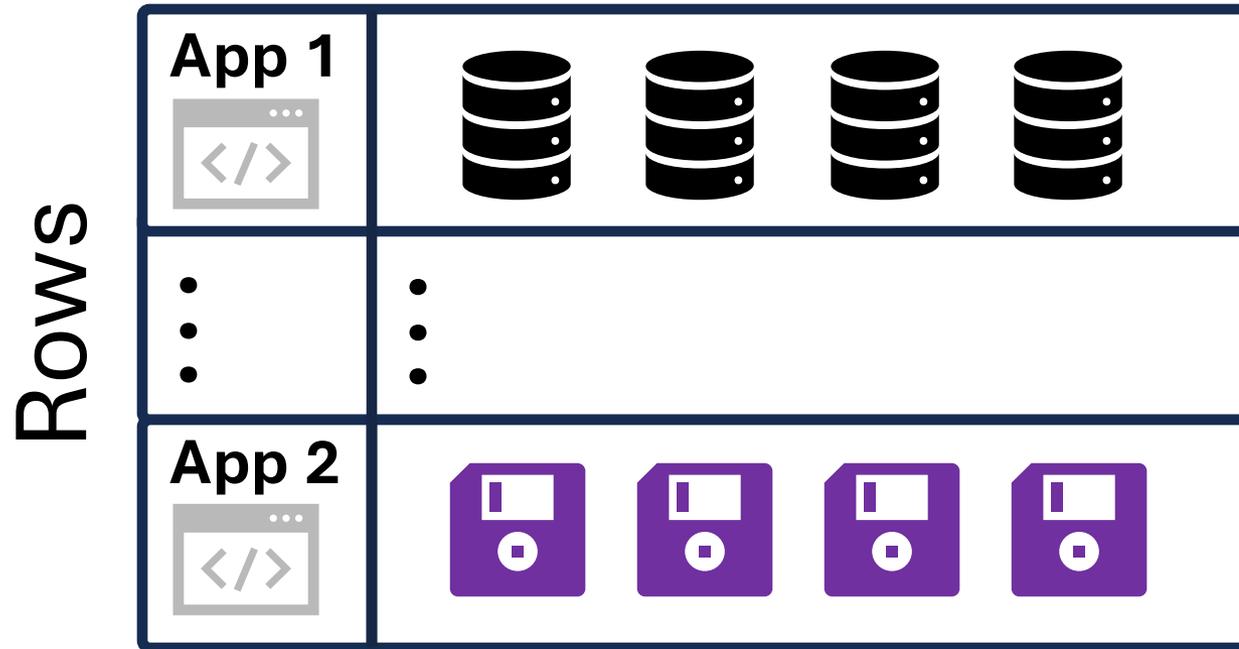
# Page Table and Memory Interference

## DRAM Bank

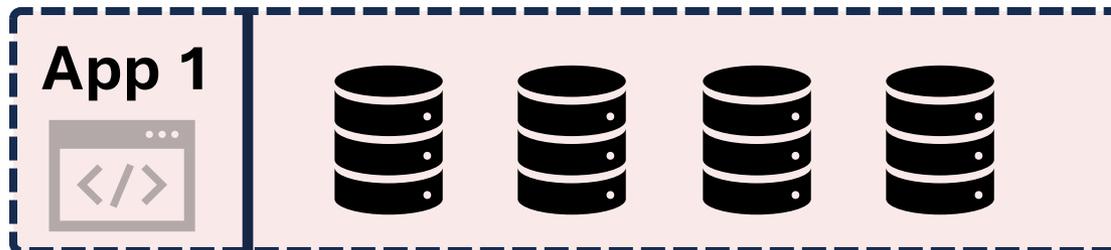


# Page Table and Memory Interference

## DRAM Bank

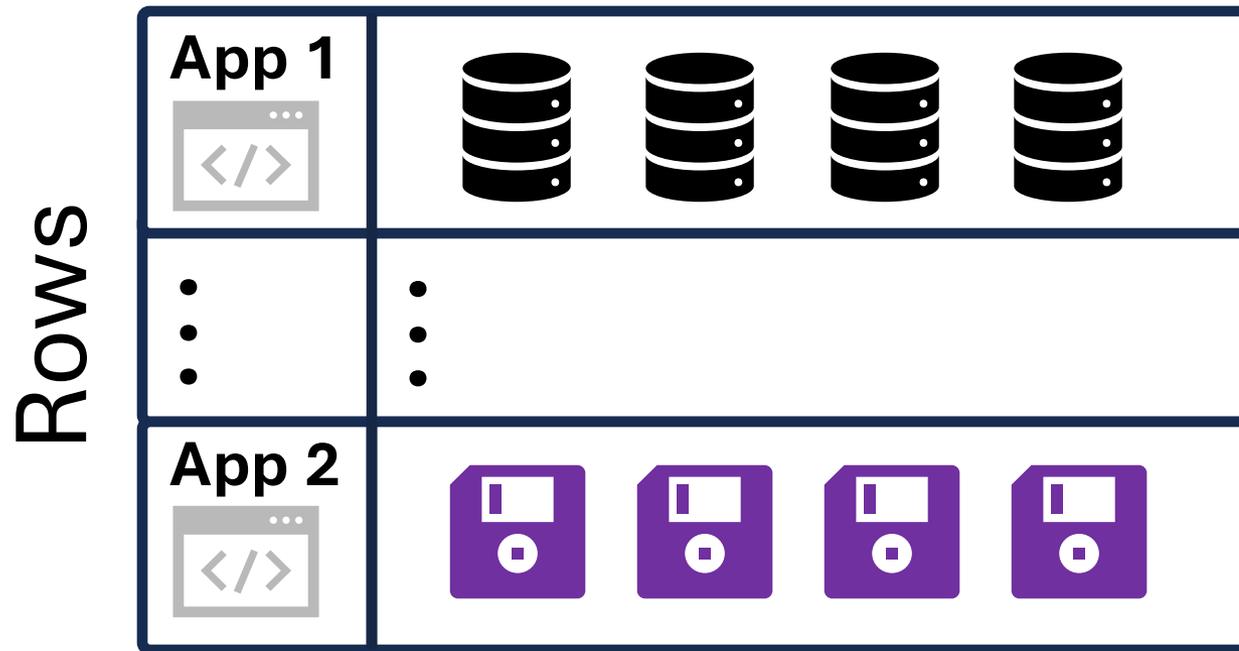


Row  
Buffer



# Page Table and Memory Interference

## DRAM Bank



Row  
Buffer

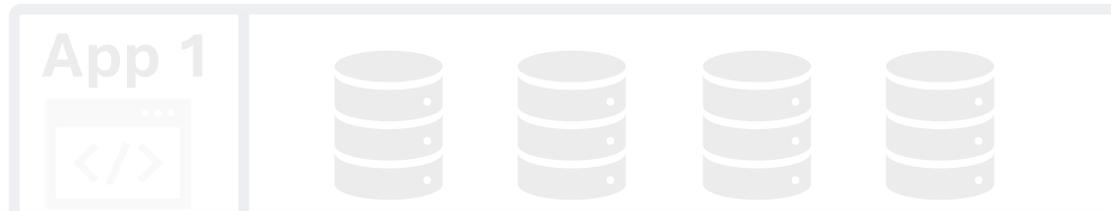


**Conflict**



# Page Table and Memory Interference

DRAM Bank



**Frequent accesses  
to page table entries cause  
high interference in main memory**

Row  
Buffer

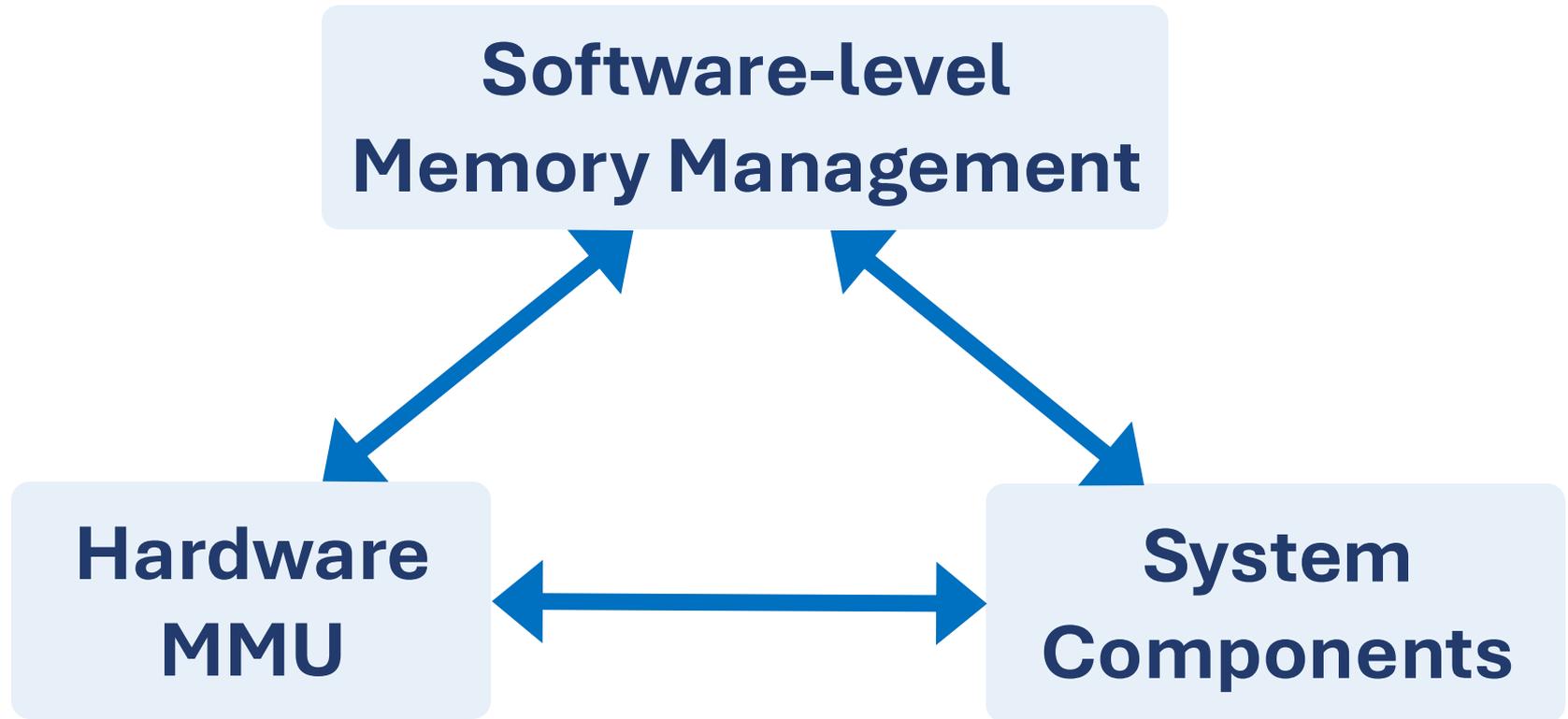


*Conflict*



# Challenges in Evaluation

---

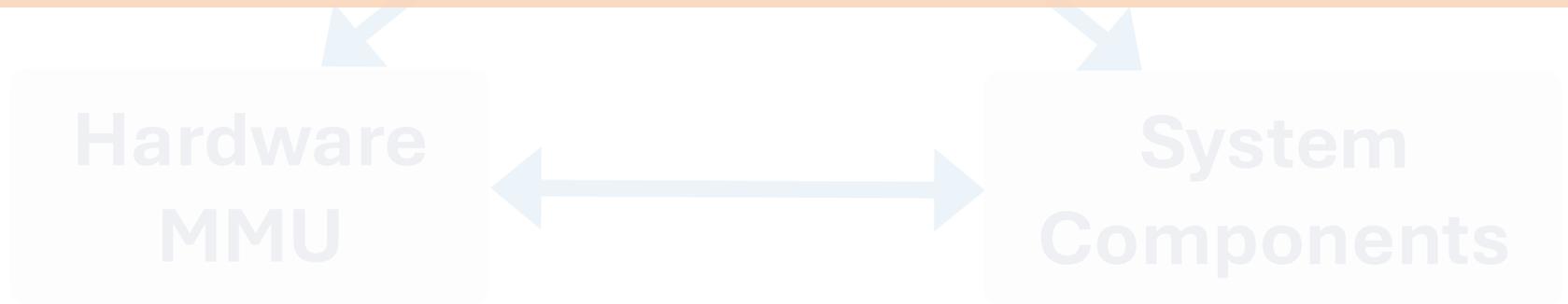


# Challenges in Evaluation

---

Software-level

**Evaluating the interplay between components is critical when assessing current and new VM techniques**



**Researchers have  
architectural simulators  
at their disposal**

# Existing Architectural Simulators

Two main classes of simulators

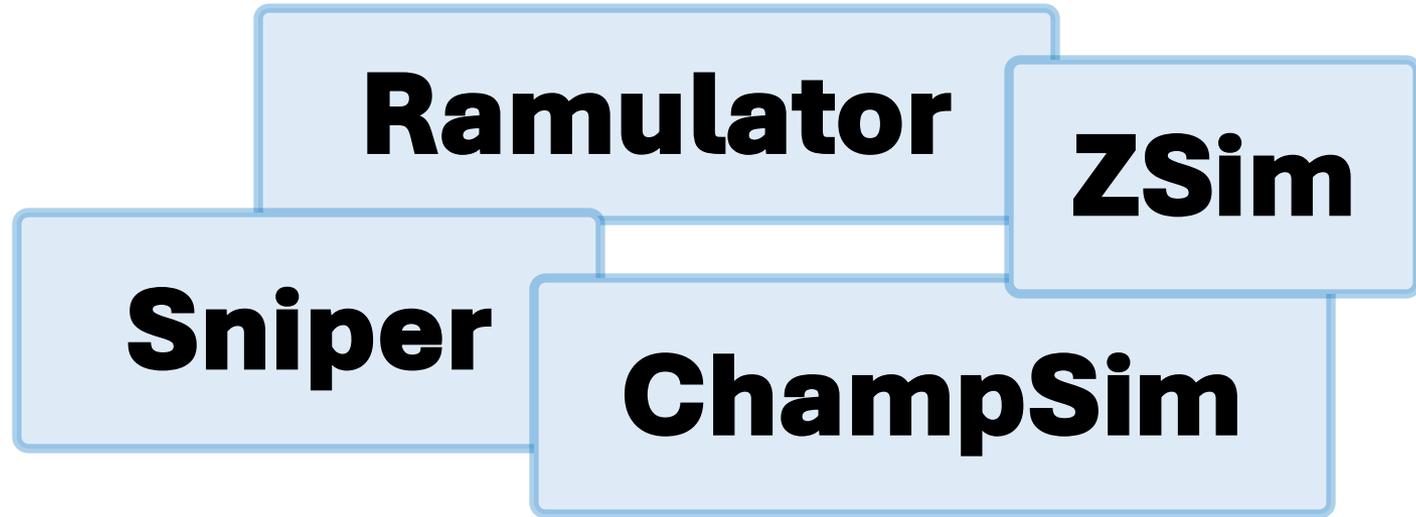
1

**Emulation-based**

2

**Full-system**

# Emulation-based Simulators



**High simulation speed** with a focus on modeling microarchitectural features

Limited support for OS primitives

Analytically estimate VM Overheads

# Analytical Estimation of VM Overheads

## First-order Models

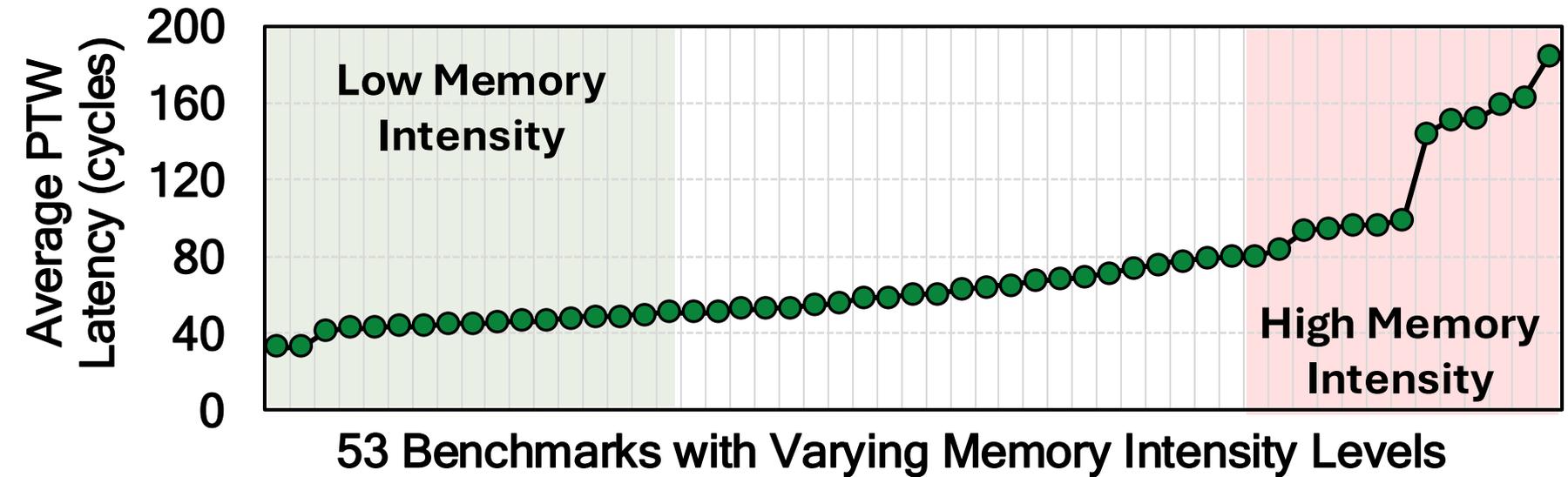
**Simple** mathematical or algorithmic approximation

### *Example*

**Page Table Walk Latency = 100 cycles (fixed)**  
**Minor Page Fault Latency = 2000 cycles (fixed)**

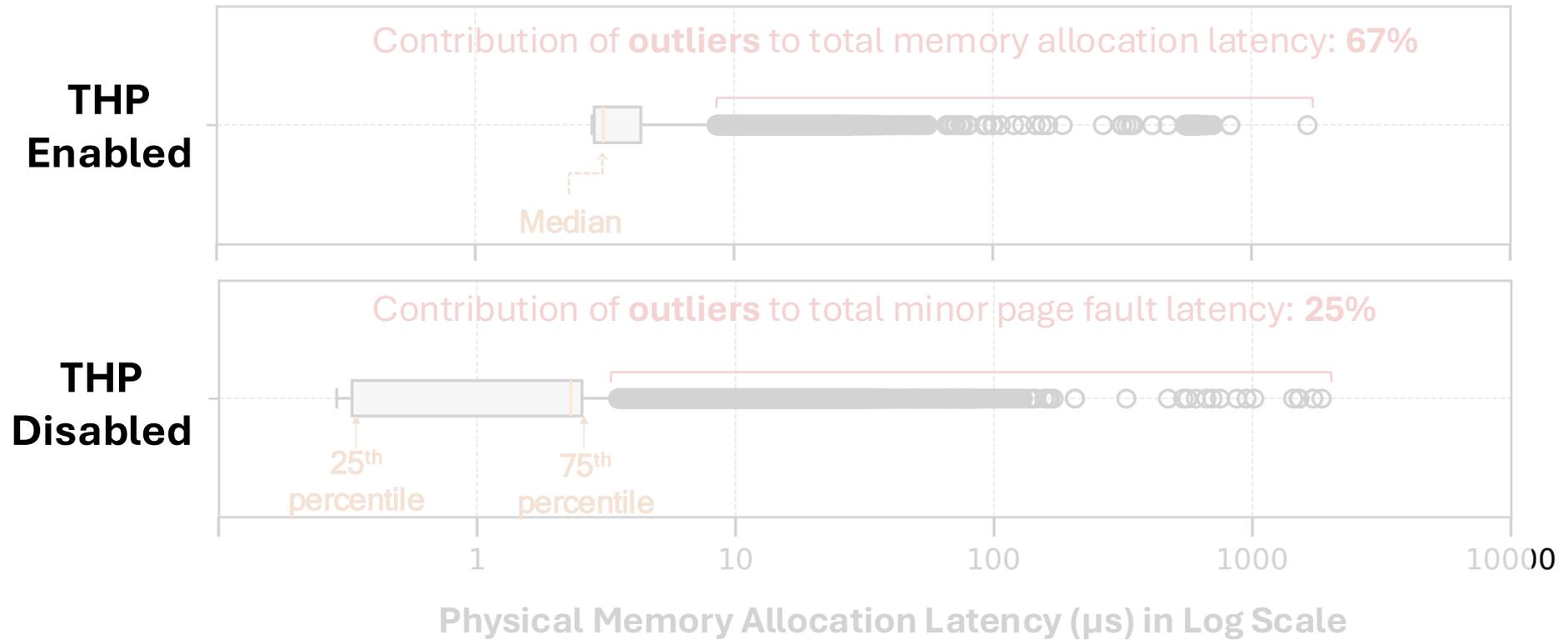
**Is this good enough?**

# Variable PTW Latency

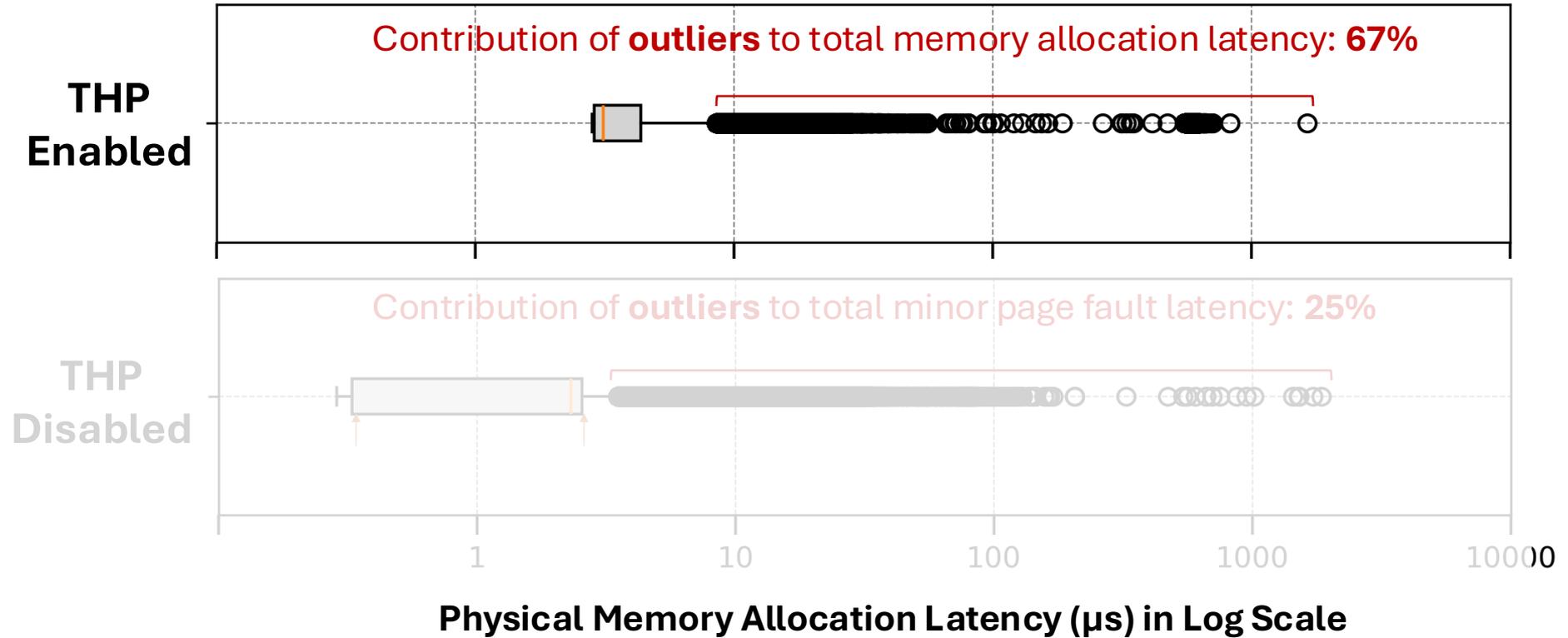


Average PTW Latency (measured in a real system) ranges from 33 up to 184 cycles

# Variable Memory Allocation Latency

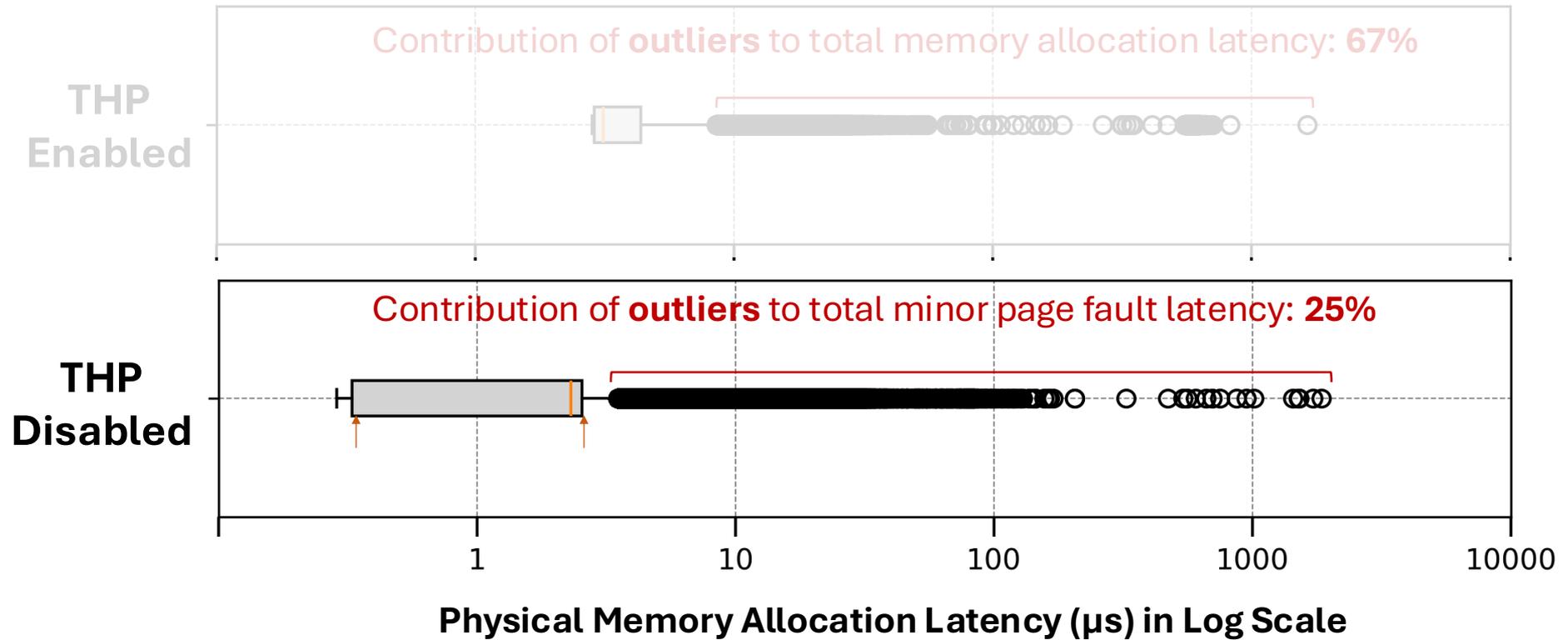


# Variable Memory Allocation Latency



The latency of handling memory allocation  
(measured in a real system)  
exhibits high variability

# Variable Memory Allocation Latency



The latency of handling memory allocation (measured in a real system) exhibits high variability

## Use of First-order Models

**First-order models do not accurately capture the dynamic nature of VM overheads**

# Full-System Simulators

---

**gem5-FS**

**QFlex**

**PTLsim**

Enable the execution of a **full-blown OS**  
on top of a hardware simulator

**Low simulation  
speed**

**Hard to develop**

# Existing Architectural Simulators

Two main classes of simulators

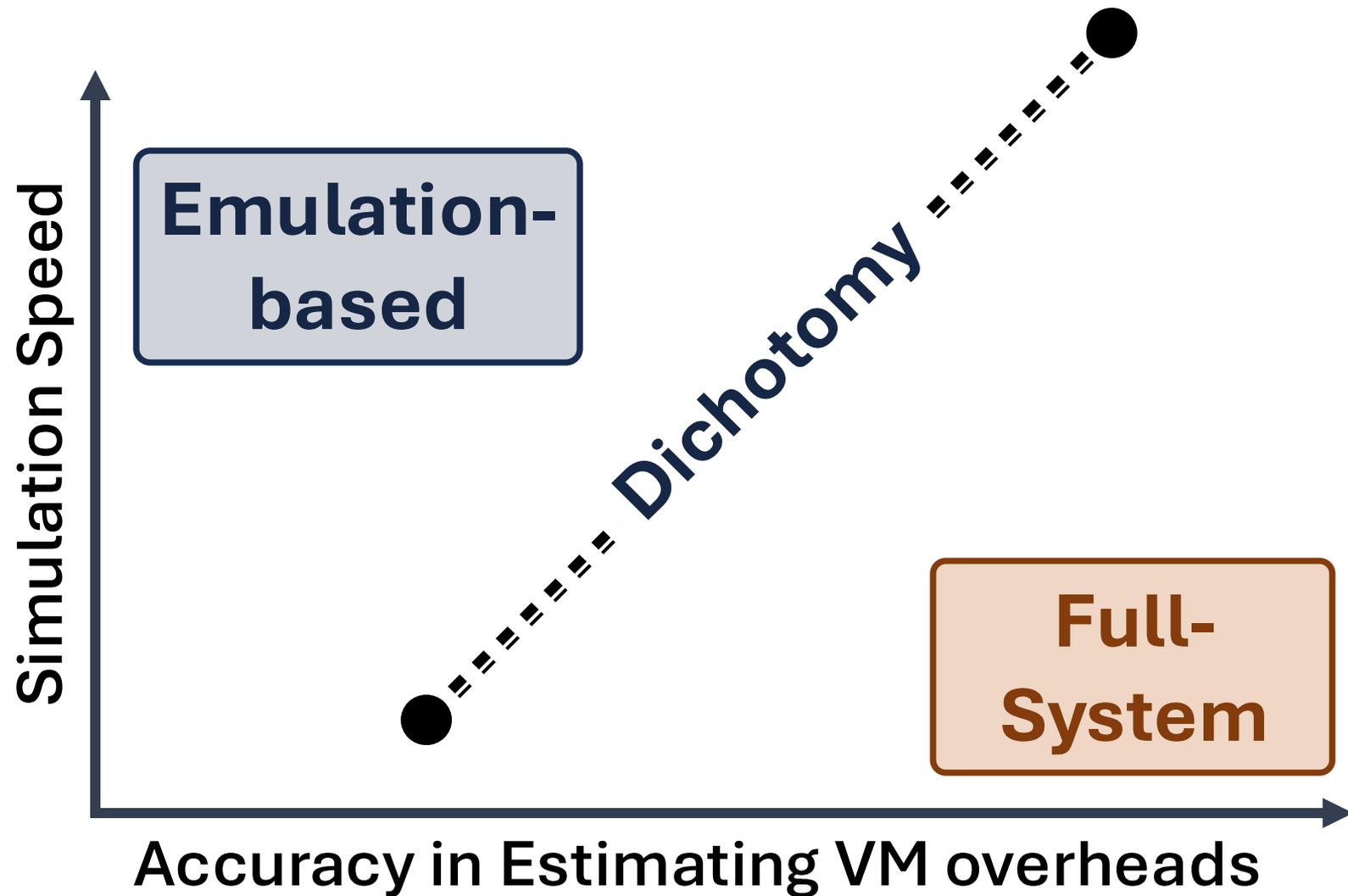
1

**Emulation-based**

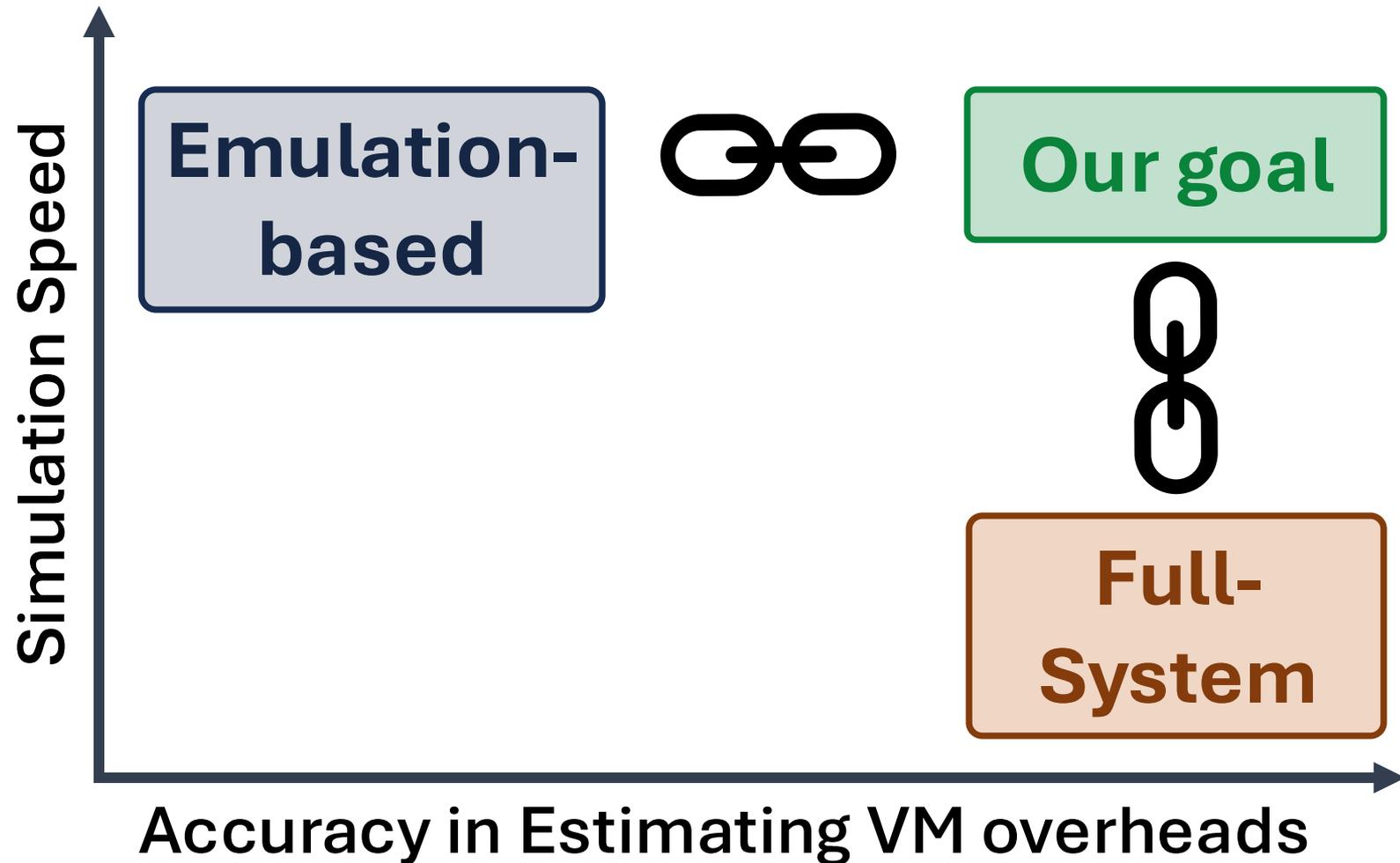
2

**Full-System**

# Existing Architectural Simulators

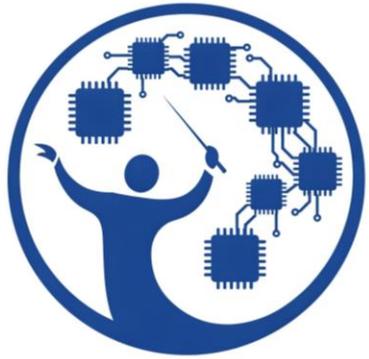


# Existing Architectural Simulators



# Enabling Fast and Accurate VM Research

---



# Virtuoso

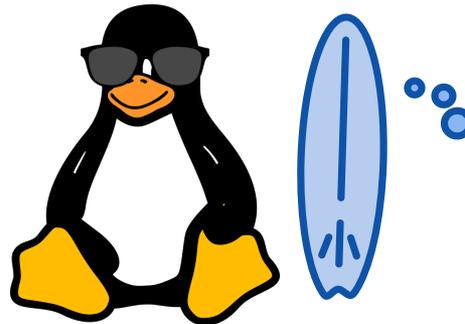
New simulation framework that enables **fast and accurate** prototyping and evaluation of the ***software and hardware*** components of VM

# Imitation-based Simulation Methodology

## Lightweight Userspace Kernel

*written in a high-level programming language*

*“I will try to imitate  
the full-blown OS”*



**Choose OS modules only related  
to the desired research**

# Imitation-based Simulation Methodology

Monitoring

**Full-blown Kernel**

Interrupts

Thread Scheduler

Managing NUMA

Data Sharing  
Support

Transparent  
Huge Pages

Swapping

Virtual Memory  
Area Handling

Cgroups

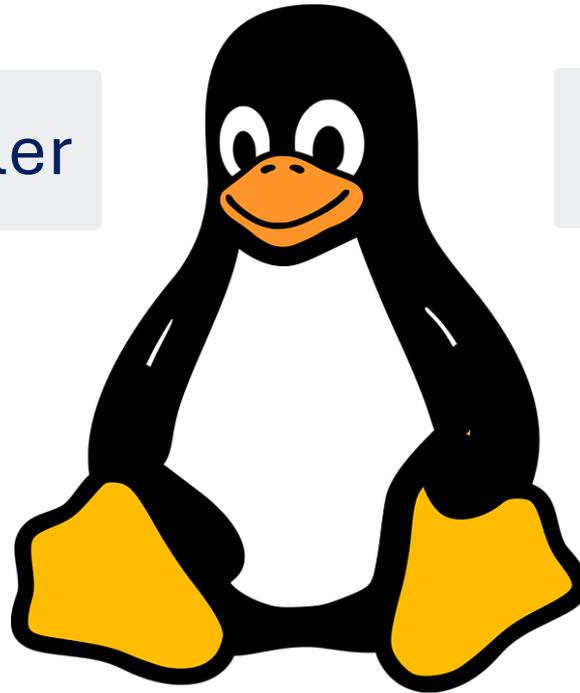
Boot process

Protection

Block device  
driver

SLAB Allocator

hugetlbfs



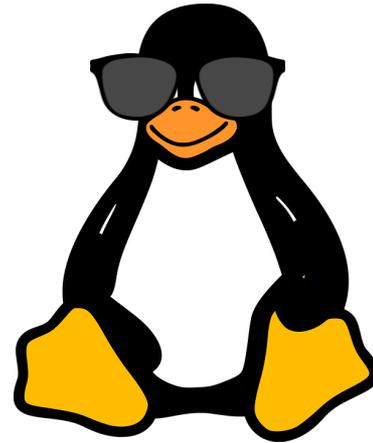
# Imitation-based Simulation Methodology

## Lightweight Userspace Kernel

Swapping

Virtual Memory  
Area Handling

Transparent  
Huge Pages



# Imitation-based Simulation Methodology

---

1

**Rapid development and prototyping**

2

**High simulation speed** by executing only the desired OS kernel functionality

3

**Accurate evaluation** by integrating Virtuoso with an architectural simulator

# Imitation-based Simulation Methodology

## Lightweight Userspace Kernel

Desired Memory Management  
modules written in a high-level language



Binary Instrumentation

Userspace Kernel  
Instruction Stream

+ Imitates the OS-level  
Overhead

```
add R1, R1 ,R2
```

```
load R1, [0xA]
```

Simulator

Core Model

# Imitation-based Simulation Methodology

## Lightweight Userspace Kernel

**Desired Memory Management**  
modules written in a high-level language



Binary Instrumentation

Userspace Kernel  
Instruction Stream

+ Enables emulation  
of kernel modules

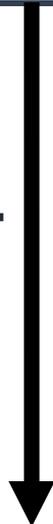
add R1, R1, R2

load R1, [0xA]

**Simulator**

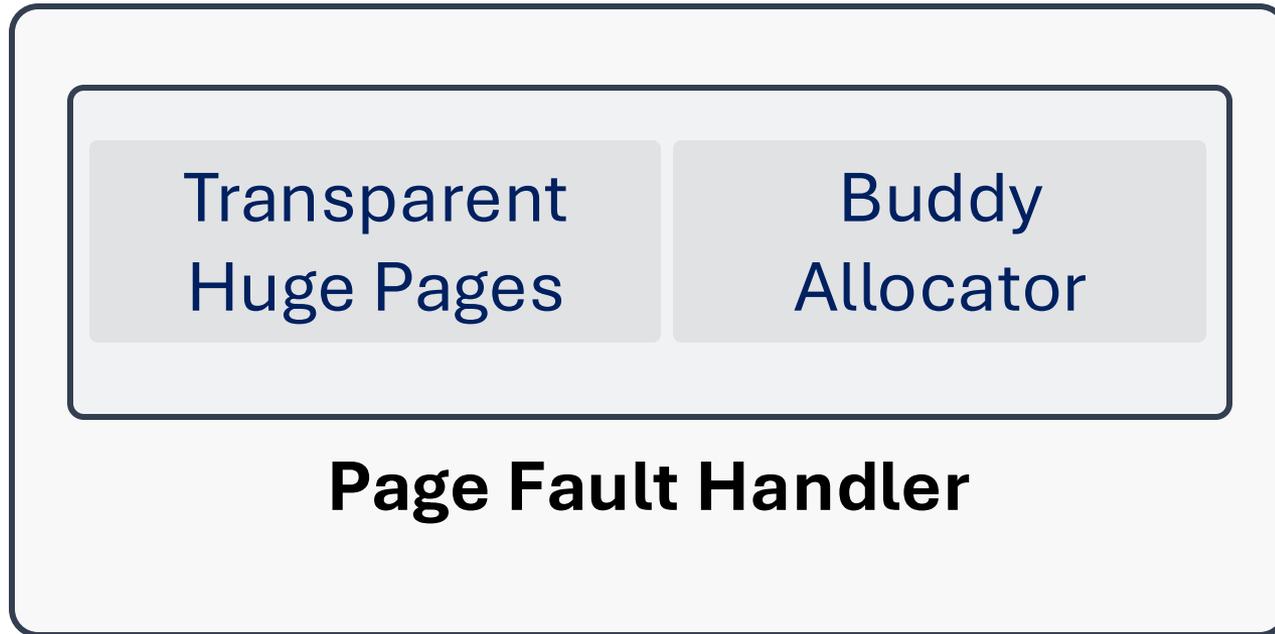
**Core Model**

**Functional  
Outcome**



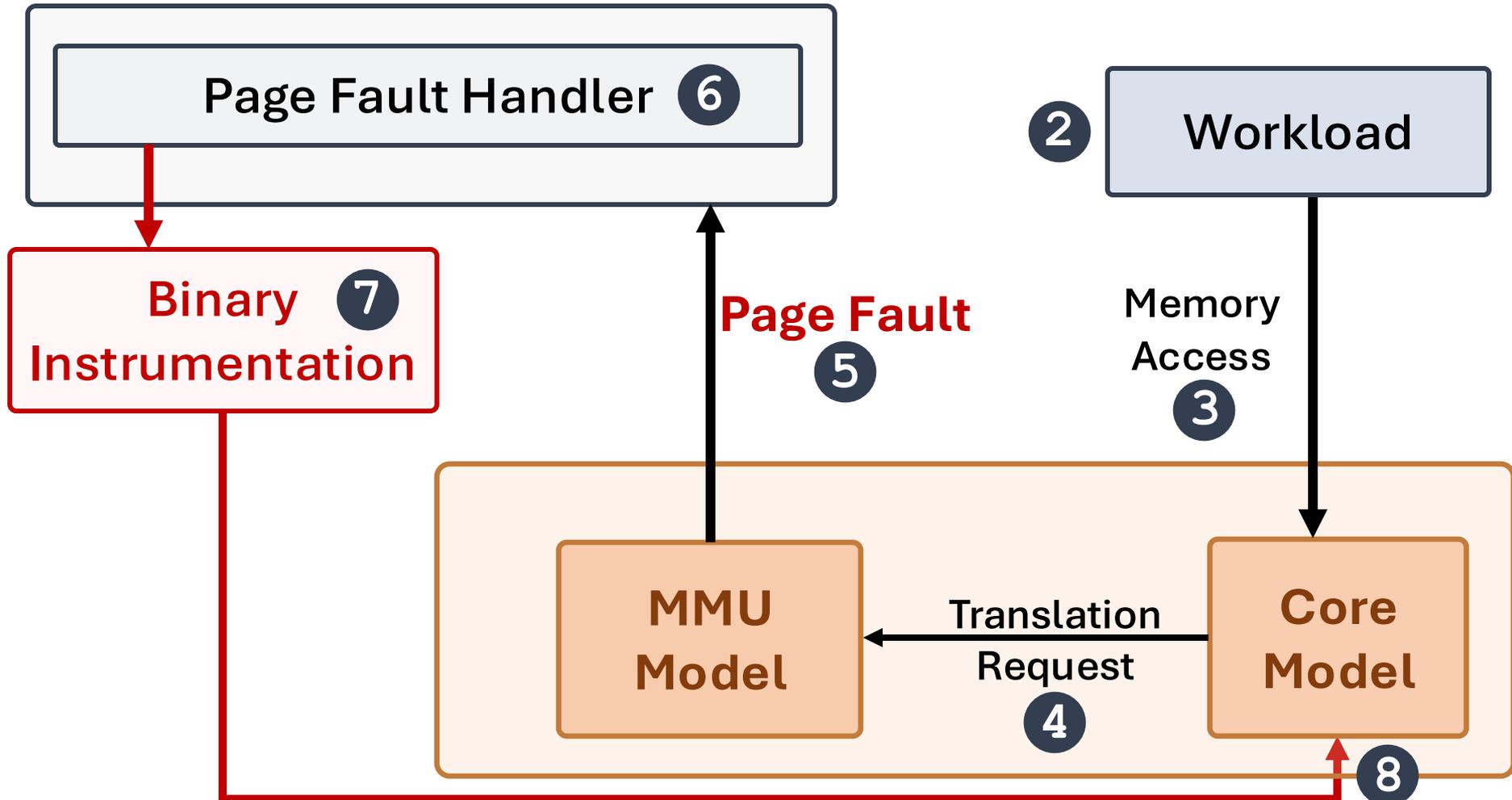
# Workflow: Page Fault Handling Example

## Example Userspace Kernel



# Workflow: Page Fault Handling Example

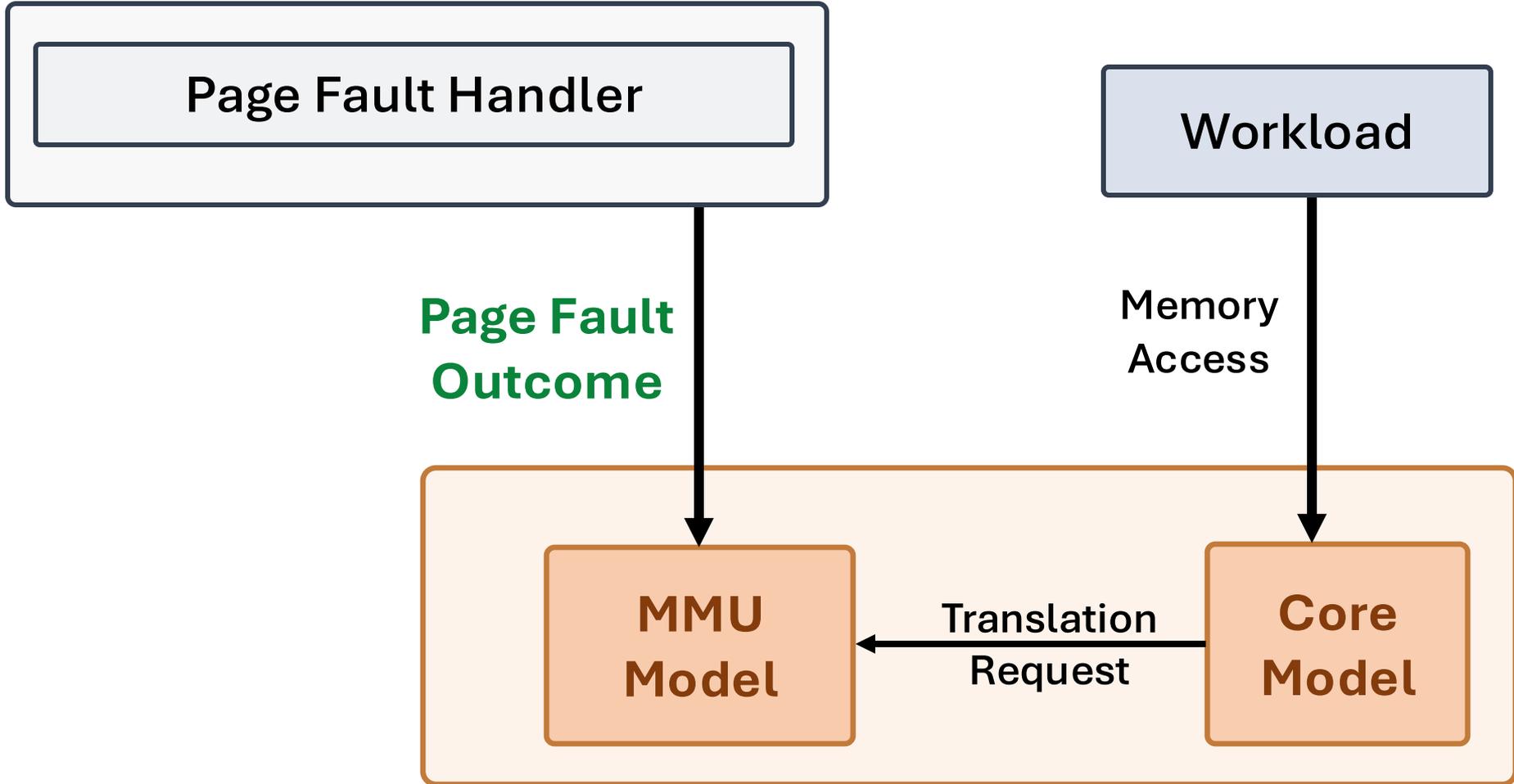
## Example Userspace Kernel ①



Example: Sniper Simulator [Carlson+ TACO 12]

# Workflow: Page Fault Handling Example

## Example Userspace Kernel



Example: Sniper Simulator [Carlson+ TACO 12]

# MimicOS: Imitating the Linux Kernel

## MimicOS modules written in C++

Buddy  
Allocator

Radix-based  
page table

hugetlbfs

Transparent  
Huge Pages

Swap Space

Page Cache

**More details in the paper**

**<https://arxiv.org/pdf/2403.04635>**

# VirTool: State-of-the-art VM techniques



Toolset encompassing the HW/SW components of multiple state-of-the-art VM techniques

4 page table designs

Software-Managed TLB

Page-Size Prediction

Nested MMU Support

2 Hash-based address mapping schemes

2 Memory Tagging Schemes

2 THP Policies

2 Contiguity-aware Schemes

2 Intermediate Address Space Schemes

Speculative Translation

TLB Prefetching

# VirTool: State-of-the-art VM techniques



Toolset encompassing multiple state-of-the-art VM techniques

4 page table

Software-

Page-Size Prediction

**Enables researchers to evaluate and prototype current and brand new VM techniques**

2 THP Policies

2 Intermediate Address Space Schemes

# VirTool: State-of-the-art VM techniques



Toolset encompassing multiple state-of-the-art VM techniques

4 page table

Software-

Page-Size Prediction

**Virtuoso is a highly versatile simulation framework**

2 IHP Policies

2 Intermediate Address Space Schemes

# Integrated Virtuoso with 5 Diverse Architectural Simulators

gem5-SE

**System-call emulation  
mode of gem5**

**[Lowe-Power+ arXiv 2020]**

**[Binkert+ SIGARCH News 2011]**

<http://gem5.org/>

# Integrated Virtuoso with 5 Diverse Architectural Simulators

gem5-SE

Ramulator

**Focus on main  
memory subsystem**

**[Luo+ CAL 2023]**

<https://github.com/CMU-SAFARI/ramulator2>

# Integrated Virtuoso with 5 Diverse Architectural Simulators

gem5-SE

Ramulator

Sniper

**Focus on  
multicore systems**

**[Carlson+ TACO 2012]**

<https://github.com/snipersim/snipersim>

# Integrated Virtuoso with 5 Diverse Architectural Simulators

gem5-SE

Ramulator

Sniper

**ChampSim**

**Focus on  
microarchitecture**

**[Gober+ arXiv 2022]**

<https://github.com/ChampSim/ChampSim>

# Integrated Virtuoso with 5 Diverse Architectural Simulators

gem5-SE

Ramulator

Sniper

ChampSim

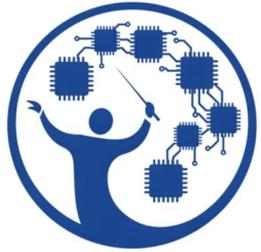
**Focus on  
storage devices**

**[Tavakkol+ FAST 2018]**

**MQSim**

<https://github.com/CMU-SAFARI/MQSim>

# Validation against a Real System

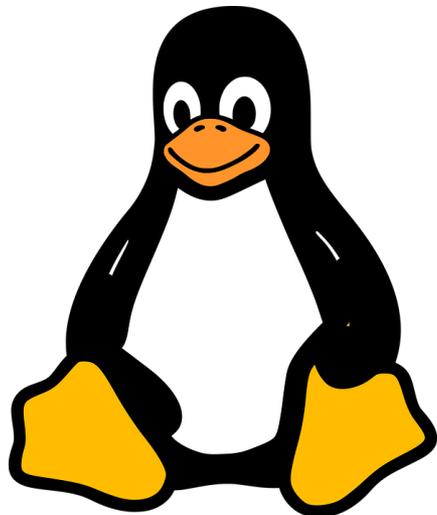


**Virtuoso** ∞

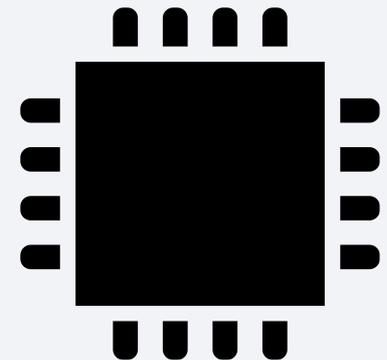
Sniper

**Against**

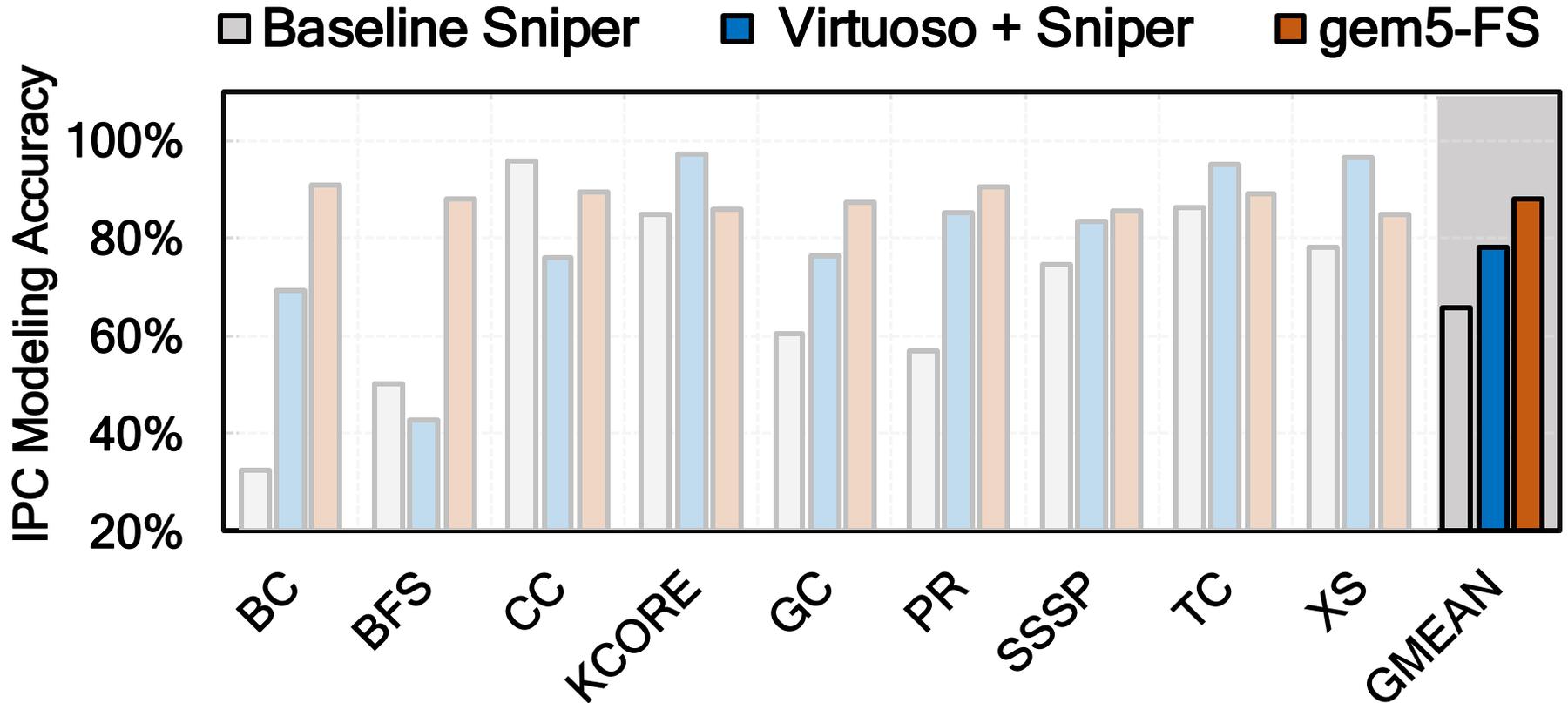
Linux Kernel



High-end  
Server-grade  
CPU

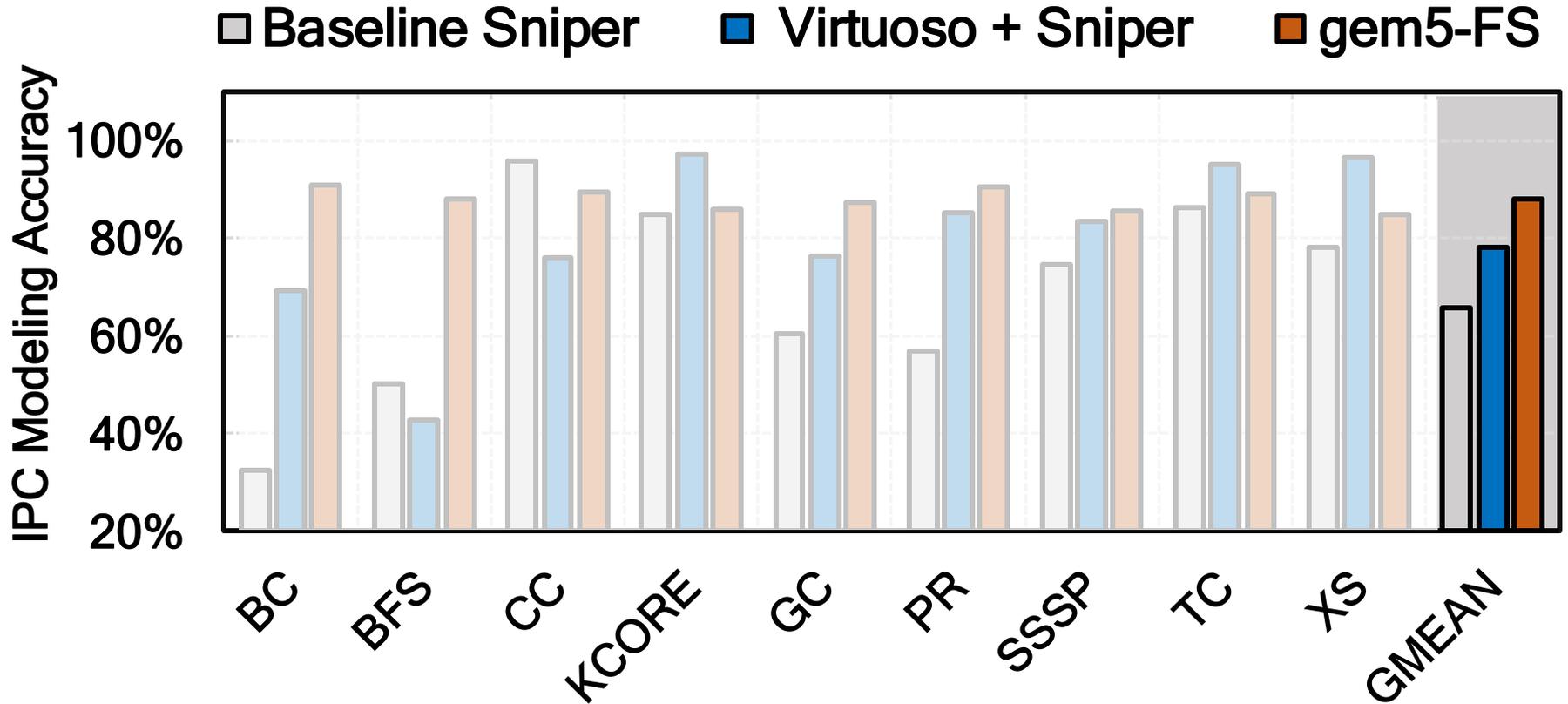


# Validation: Instructions Per Cycle (IPC)



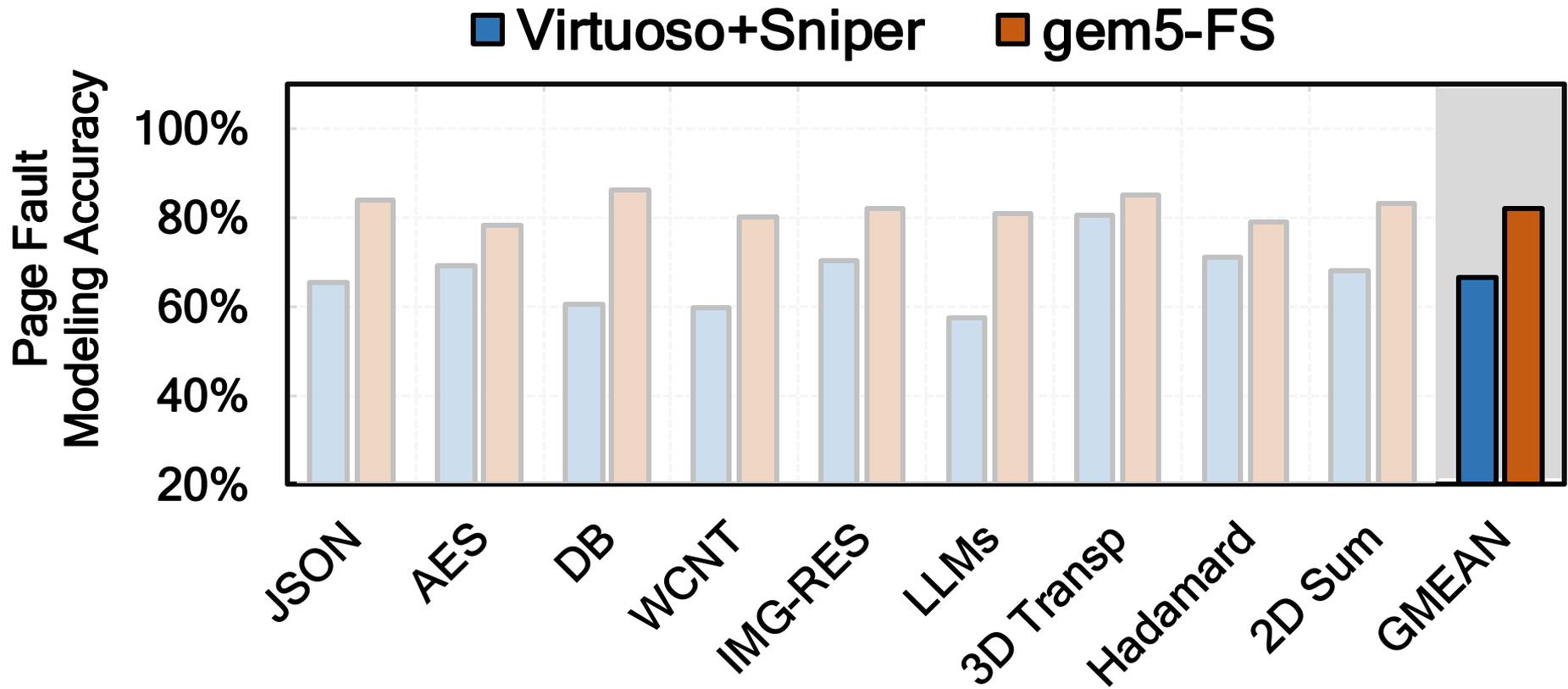
Virtuoso integrated with Sniper improves IPC modeling accuracy by 21% compared to baseline Sniper

# Validation: Instructions Per Cycle (IPC)



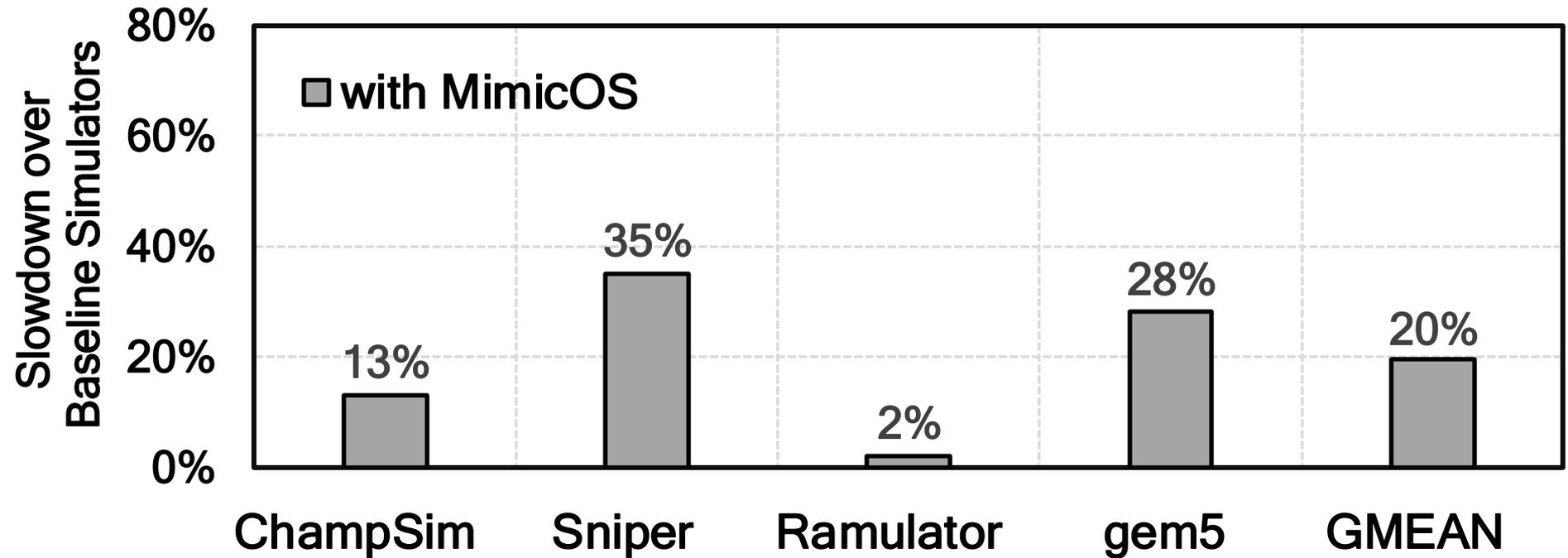
**Virtuoso integrated with Sniper achieves IPC modeling accuracy within 9% of gem5-FS**

# Validation: Page Fault Handling Latency



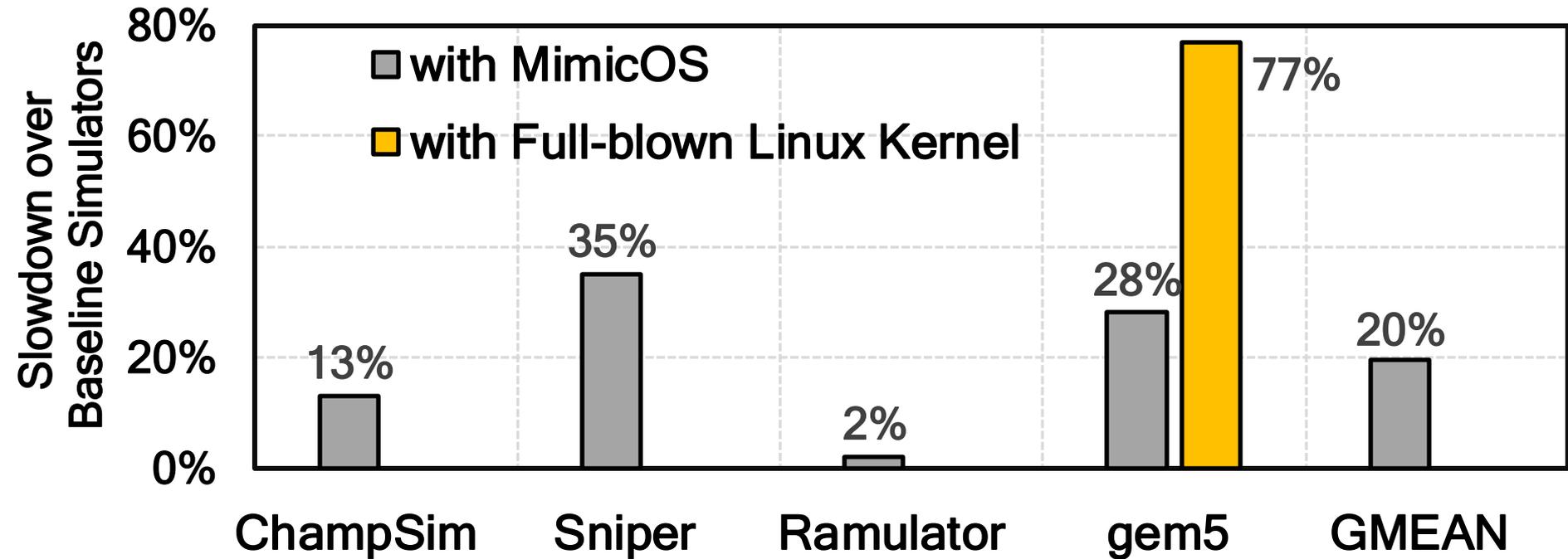
Virtuoso integrated with Sniper models the page fault handling latency with 66% accuracy, within 15% of gem5-FS

# Simulation Speed Comparison



**MimicOS leads to an average 20% simulation time overhead over the baseline version of the simulator**

# Simulation Speed Comparison



**Enabling full-system execution mode in gem5 leads to 77% simulation time overhead**

# Virtuoso's Versatility

---

**We showcase Virtuoso's versatility by evaluating multiple different use cases**

Demonstrate new insights on state-of-the-art VM techniques

**5 use cases in the paper**

**<https://arxiv.org/pdf/2403.04635>**

# Virtuoso Example Use Cases

---

**Evaluating Different  
Page Table Designs**

**Evaluating Physical Memory  
Allocation Policies**

# Virtuoso Example Use Cases

---

**Evaluating Different  
Page Table Designs**

**Evaluating Physical Memory  
Allocation Policies**

# Evaluation Methodology

---

**Radix:** Baseline radix-based page table

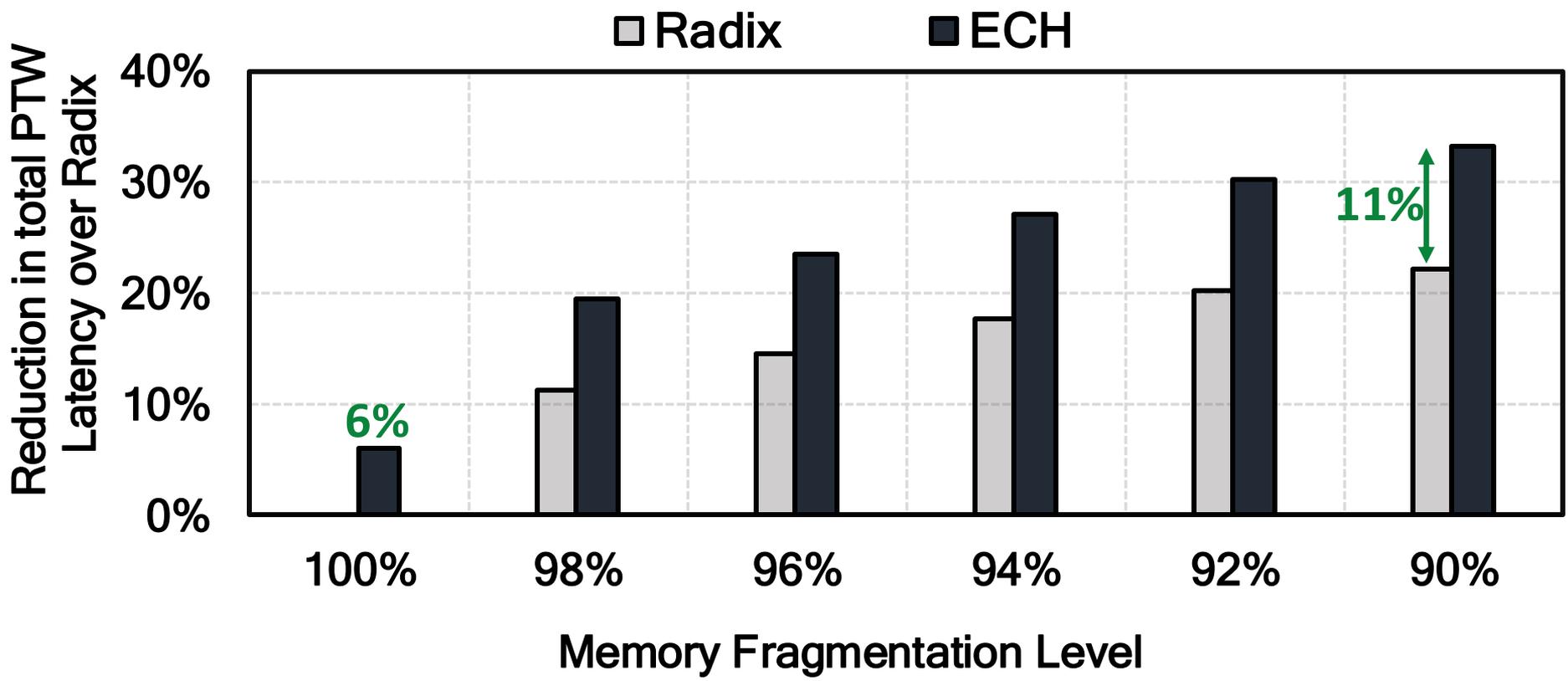
**ECH<sup>1</sup>:** Elastic Cuckoo Hash-based Page Table

***Fragmentation:*** Number of available 2MB pages compared to the total number of 2MB pages

*Workloads:*            **GraphBIG, XSBench, HPCC**

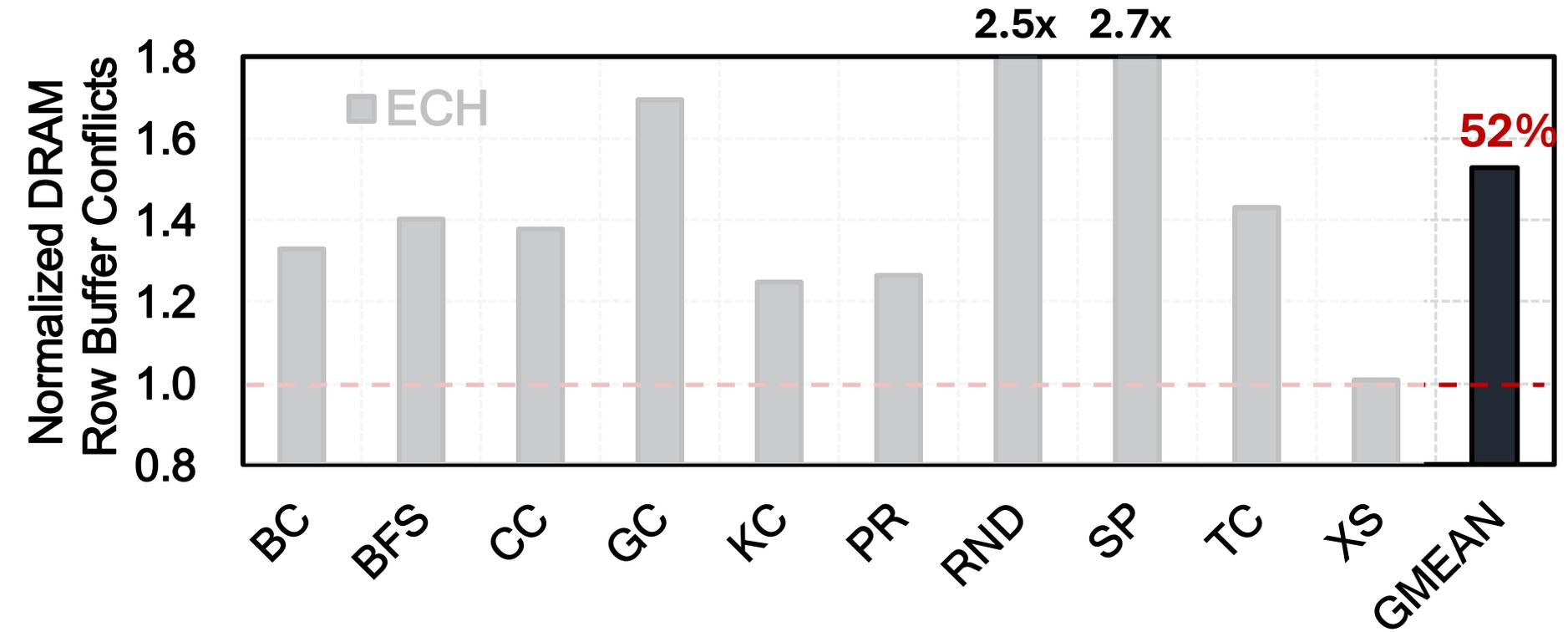
[1] Skarlatos et al. “Elastic Cuckoo Page Tables: Rethinking Virtual Memory Translation for Parallelism” ASPLOS 2020

# Reduction in PTW Latency



As memory fragmentation decreases, the PTW latency reduction of ECH compared to Radix increases

# Main Memory Interference



**ECH leads to an average 52% increase in DRAM row buffer conflicts over Radix**

# Virtuoso Example Use Cases

---

Evaluating Different  
Page Table Designs

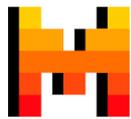
**Evaluating Physical Memory  
Allocation Policies**

# Evaluation Methodology

**Buddy:** Baseline allocator with the buddy system

**Utopia:**<sup>1</sup> Part of memory is organized using a hash-based mapping

*Workloads:* **Short-input Short-output LLM inference**



Mistral-7B



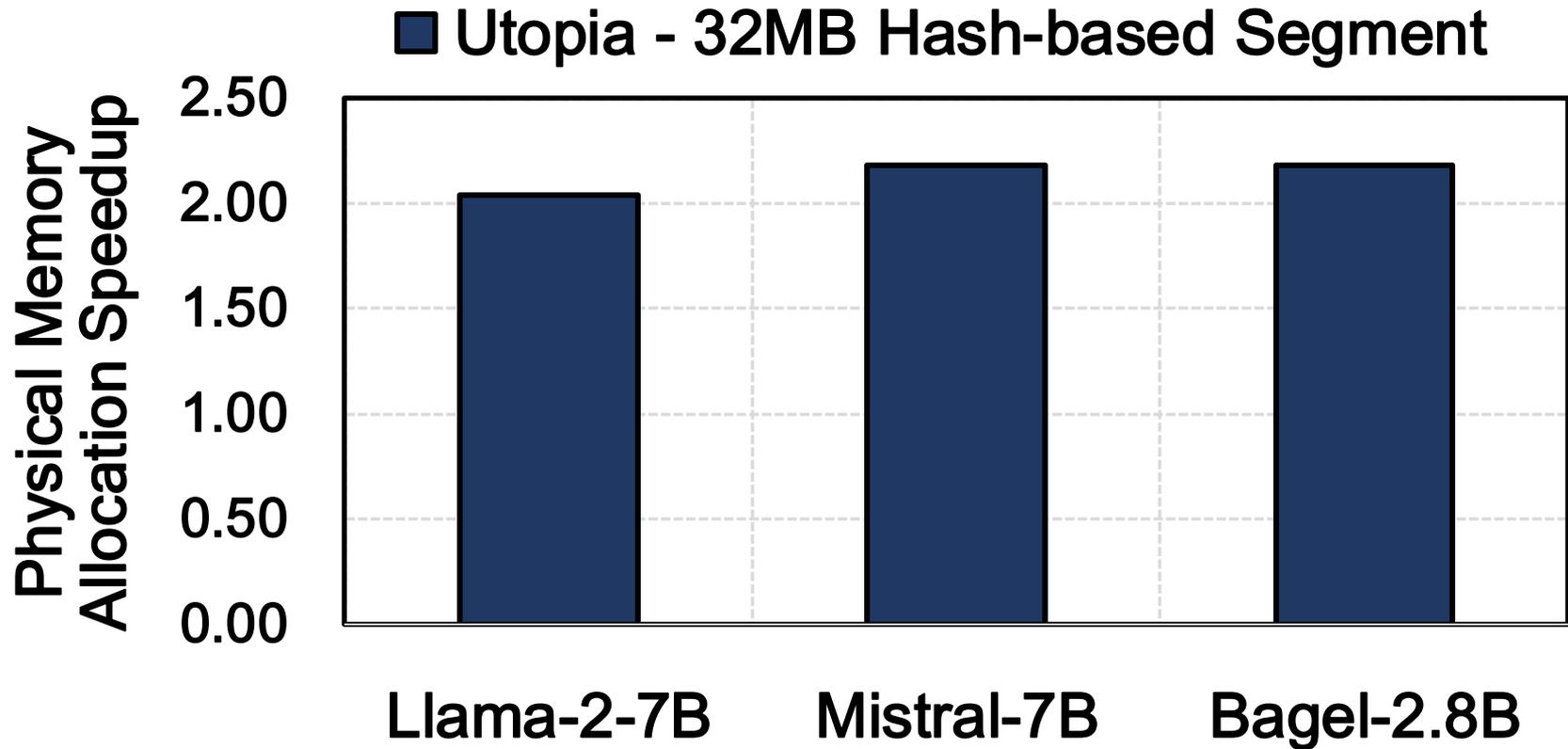
Llama-2-7B

**BAGEL**

Bagel-2.8B

[1] Kanellopoulos et al. “Utopia: Fast and Efficient Address Translation via Hybrid Restrictive & Flexible Virtual-to-Physical Address Mappings” MICRO 2023

# Accelerating Memory Allocation



Restricting the virtual-to-physical mapping speeds up memory allocation by up to 2.17x

# More Details in the Paper

---

- Detailed description of communication primitives
- Detailed description of MimicOS modules
- Integration methodology with different simulators
- Integration with heterogeneous system simulation
- Evaluation of intermediate address space schemes
- Evaluation of swapping activity and translation latency in hash-based address mapping schemes
- Evaluation of two additional hash-based page tables

**and more to come ...**

# More Details in the Paper

## Virtuoso: Enabling Fast and Accurate Virtual Memory Research via an Imitation-based Operating System Simulation Methodology

Konstantinos Kanellopoulos<sup>1</sup> Konstantinos Sgouras<sup>1</sup> F. Nisa Bostanci<sup>1</sup>  
Andreas Kosmas Kakolyris<sup>1</sup> Berkin Kerim Konar<sup>1</sup> Rahul Bera<sup>1</sup>  
Mohammad Sadrosadati<sup>1</sup> Rakesh Kumar<sup>2</sup> Nandita Vijaykumar<sup>3</sup> Onur Mutlu<sup>1</sup>

<sup>1</sup>ETH Zürich <sup>2</sup>Norwegian University of Science and Technology <sup>3</sup>University of Toronto

### Abstract

The unprecedented growth in data demand from emerging applications has turned virtual memory (VM) into a major performance bottleneck. VM's overheads are expected to persist as memory requirements continue to increase. Researchers explore new hardware/OS co-designs to optimize VM across diverse applications and systems. To evaluate such designs, researchers rely on various simulation methodologies to model VM components. Unfortunately, current simulation tools (i) either lack the desired accuracy in modeling VM's software components or (ii) are too slow and complex to prototype and evaluate schemes that span across the hardware/software boundary.

We introduce Virtuoso, a new simulation framework that enables quick and accurate prototyping and evaluation of the *software and hardware components* of the VM subsystem. The key idea of Virtuoso is to employ a *lightweight userspace OS kernel*, called MimicOS, that (i) accelerates simulation time by imitating *only* the desired kernel functionalities, (ii) facilitates the development of new OS routines that imitate

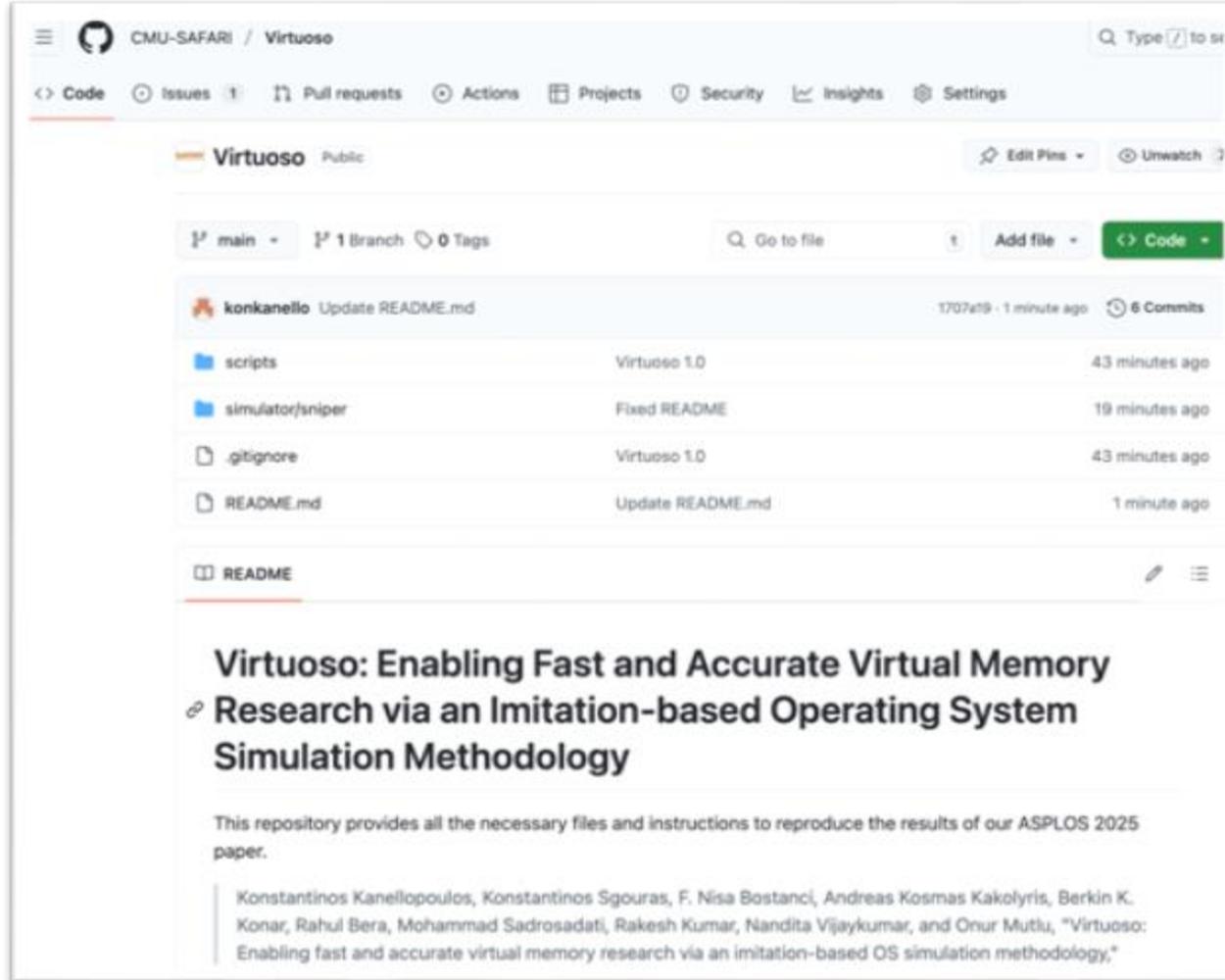
simulation time overhead of only 20%, on top of four baseline architectural simulators. The source code of Virtuoso is freely available at <https://github.com/CMU-SAFARI/Virtuoso>.

### 1 Introduction

Virtual memory (VM) [1–23] is a cornerstone of modern computing systems, enabling application-transparent physical memory management, isolation and data sharing. Contemporary applications (e.g., [24–45]) exhibit different characteristics that stress the VM subsystem. We classify these workloads into two broad categories: (i) long-running workloads (i.e., execution time larger than 100s of seconds) [24, 28–31, 33–35] with large data footprints and irregular memory access patterns, that exhibit high address translation overheads, and (ii) short-running workloads (i.e., execution time often lower than 1 second) [36–45] whose execution time does not amortize the overheads of system software operations (e.g., physical memory allocation). Multiple prior works and industrial studies [46–57] have shown that address trans-

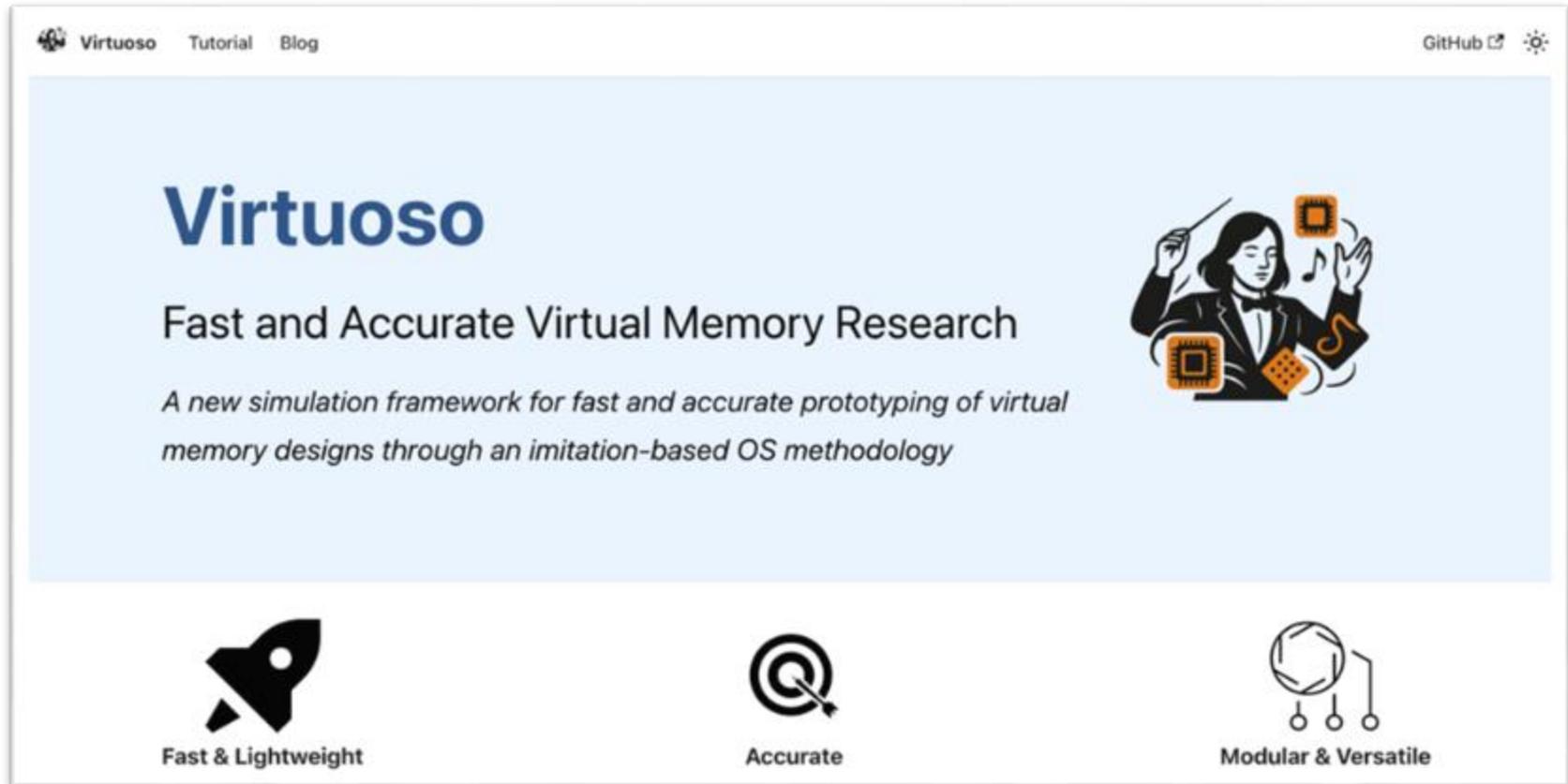
<https://arxiv.org/pdf/2403.04635>

# Virtuoso is Open-Source



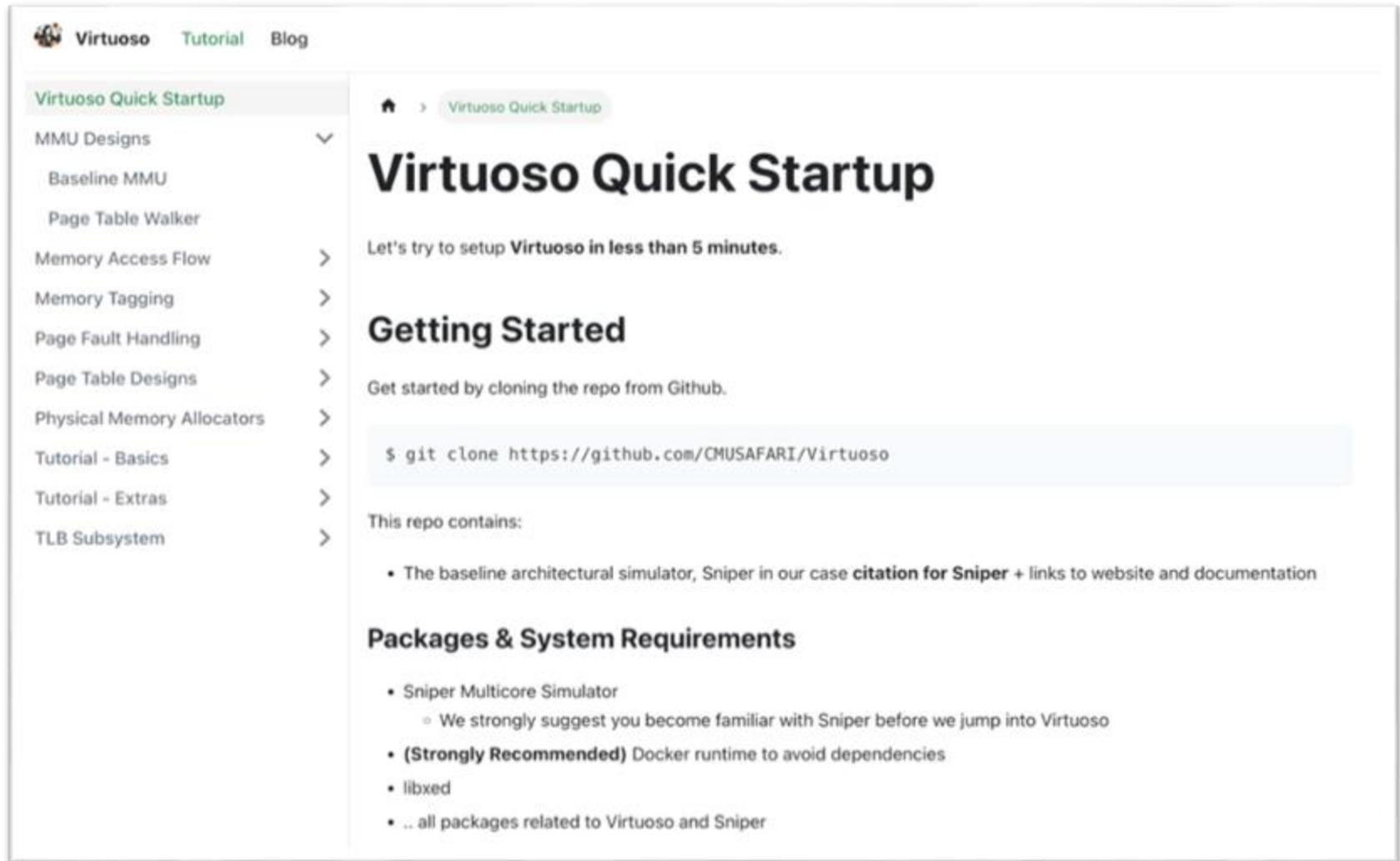
<https://github.com/CMU-SAFARI/Virtuoso>

# Virtuoso Website – Getting Started



<https://github.com/CMU-SAFARI/Virtuoso/website>

# Virtuoso Website - Documentation



The screenshot shows the Virtuoso website's documentation page for 'Virtuoso Quick Startup'. The page has a navigation menu on the left with items like 'Virtuoso Quick Startup', 'MMU Designs', 'Memory Access Flow', etc. The main content area features a breadcrumb trail, a title 'Virtuoso Quick Startup', a sub-header 'Getting Started', a code block for cloning the repository, and a list of system requirements.

Virtuoso Tutorial Blog

Virtuoso Quick Startup

MMU Designs

Baseline MMU

Page Table Walker

Memory Access Flow

Memory Tagging

Page Fault Handling

Page Table Designs

Physical Memory Allocators

Tutorial - Basics

Tutorial - Extras

TLB Subsystem

Home > Virtuoso Quick Startup

## Virtuoso Quick Startup

Let's try to setup Virtuoso in less than 5 minutes.

### Getting Started

Get started by cloning the repo from Github.

```
$ git clone https://github.com/CMUSAFARI/Virtuoso
```

This repo contains:

- The baseline architectural simulator, Sniper in our case **citation for Sniper** + links to website and documentation

### Packages & System Requirements

- Sniper Multicore Simulator
  - We strongly suggest you become familiar with Sniper before we jump into Virtuoso
- (Strongly Recommended)** Docker runtime to avoid dependencies
- libxed
- .. all packages related to Virtuoso and Sniper

<https://safari.ethz.ch/virtuoso>

# Virtuoso Website – Incoming Features

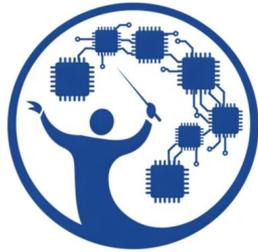
## 2025-04-02: Initial Release

- **Virtuoso Integration:**
  - [Sniper Multi-Core Simulator] (<https://github.com/snipersim/>)
- **MMU Models:**
  - i. **MMU Baseline:**
    - Page Walk Caches
    - Configurable TLB hierarchy.
    - Configurable Page Walk Cache (PWC) hierarchy
    - Large page prediction based on [Papadopoulou et al.](#)
  - ii. **MMU Speculation:** Speculative address translation as described in [SpecTLB](#)
  - iii. **MMU Software-Managed TLB:** Software-managed L3 TLB as described in [POM-TLB](#)
  - iv. **MMU Utopia:** Implements [Utopia](#)
  - v. **MMU Midgard:** Implements [Midgard](#)
  - vi. **MMU RMM (and Direct Segments):** Implements [RMM](#)
  - vii. **MMU Virtualized:** Nested Paging and Nested Page Tables (NPT) for modern hypervisors
- **Page Table Designs:**
  - i. **Page Table Baseline:** Radix page table with configurable page sizes
  - ii. **Range Table:** B++ Tree-like translation table for [virtual-to-physical address ranges](#)
  - iii. **Hash Don't Cache:** [Open-addressing hash-based page table](#)
  - iv. **Conventional Hash-Based:** Chain-based hash table design
  - v. **ECH:** [Cuckoo hashing-based organization of the page table](#)
  - vi. **RobinHood:** Open-addressing with element re-ordering
- **Memory Allocation Policies:**
  - i. **Reservation-Based THP:** Implements reservation-based Transparent Huge Pages

<https://safari.ethz.ch/virtuoso>

# Conclusion

**VM causes high overheads  
in emerging workloads**



# Virtuoso

New simulation framework that enables **fast and accurate** prototyping and evaluation of the ***software and hardware*** components of VM

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

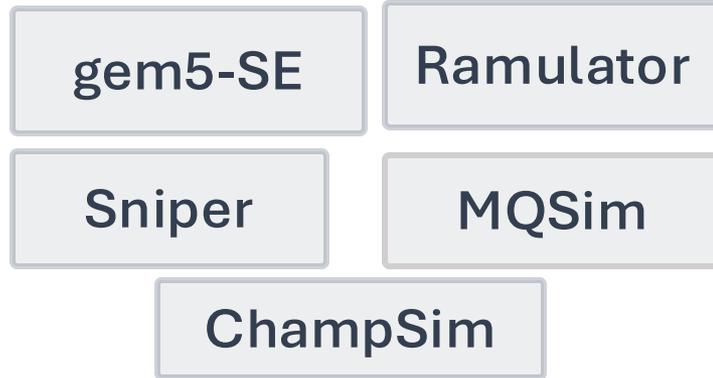
# Imitation-based Simulation Methodology

**1** Rapid development and versatility

**2** High simulation speed

**3** Accurate simulation

## Integration with 5 simulators



# Virtuoso

Toolset encompassing  
multiple state-of-the-art  
VM techniques

**Validation against high-end server-grade CPU**

**Implemented 5 diverse use cases to  
showcase Virtuoso's versatility**

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

**We hope Virtuoso establishes a common ground for VM research**

Github



# Virtuoso

arXiv



## Enabling Fast and Accurate Virtual Memory Research with an Imitation-based Operating System Simulation Methodology

**Konstantinos Kanellopoulos**

Konstantinos Sgouras

F. Nisa Bostanci

Andreas Kosmas Kakolyris

Berkin Kerim Konar

Rahul Bera

Mohammad Sadrosadati

Rakesh Kumar

Nandita Vijaykumar

Onur Mutlu

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

**SAFARI**  
SAFARI Research Group  
safari.ethz.ch

**ETH** zürich



UNIVERSITY OF  
TORONTO



NTNU

# Why do we need Virtual Memory?

# Virtual Memory Benefits

**1**

**Programmer-transparent  
memory management**

**2**

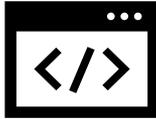
**Process isolation**

**3**

**Data sharing between processes**

# How does Virtual Memory work?

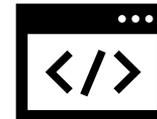
App 1



Virtual Address  
Space #1



App 2

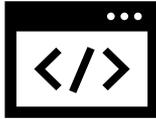


Virtual Address  
Space #2



Physical Address Space (e.g., physical memory)

App 1

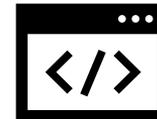


Virtual Address  
Space #1



(e.g., 4KB)

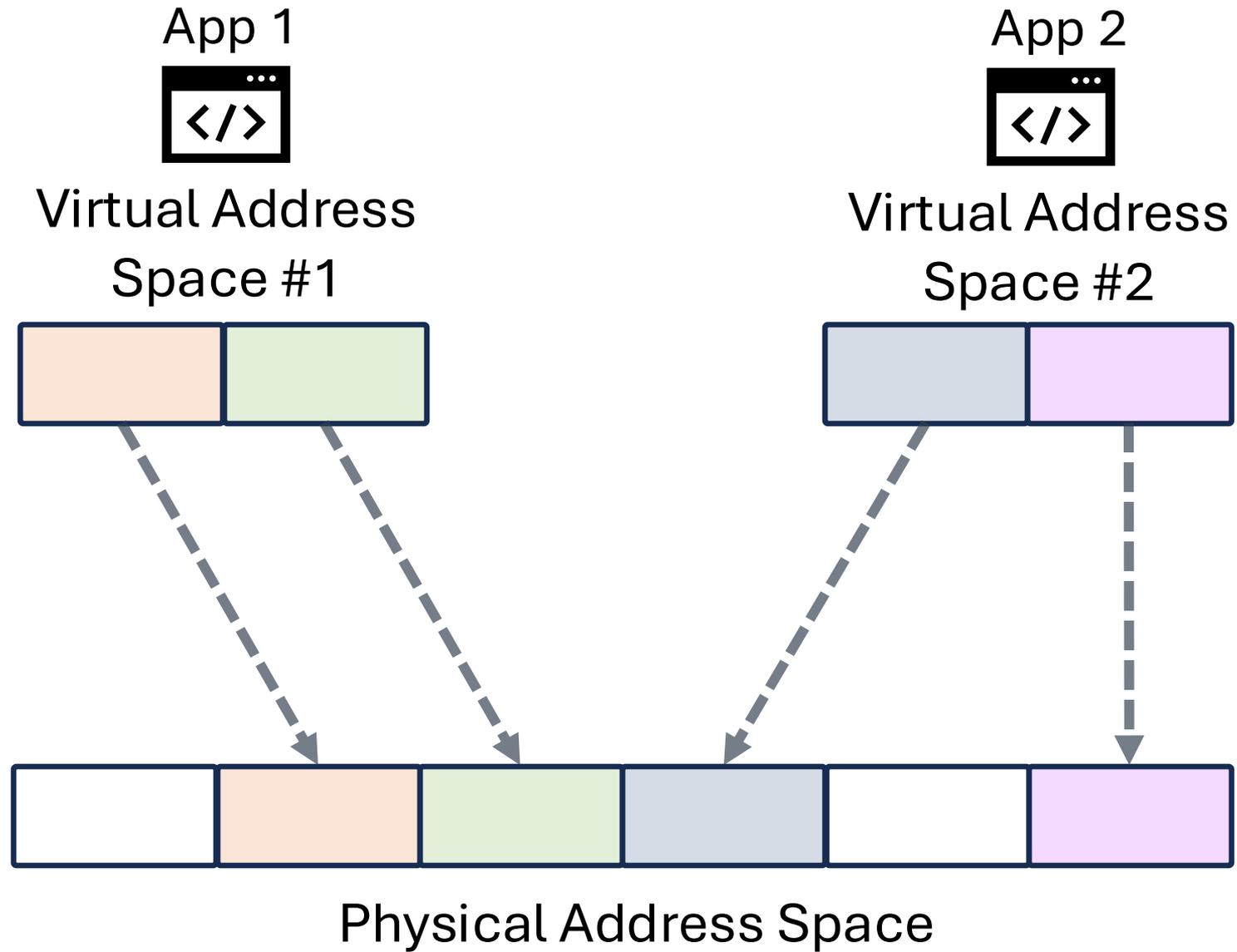
App 2

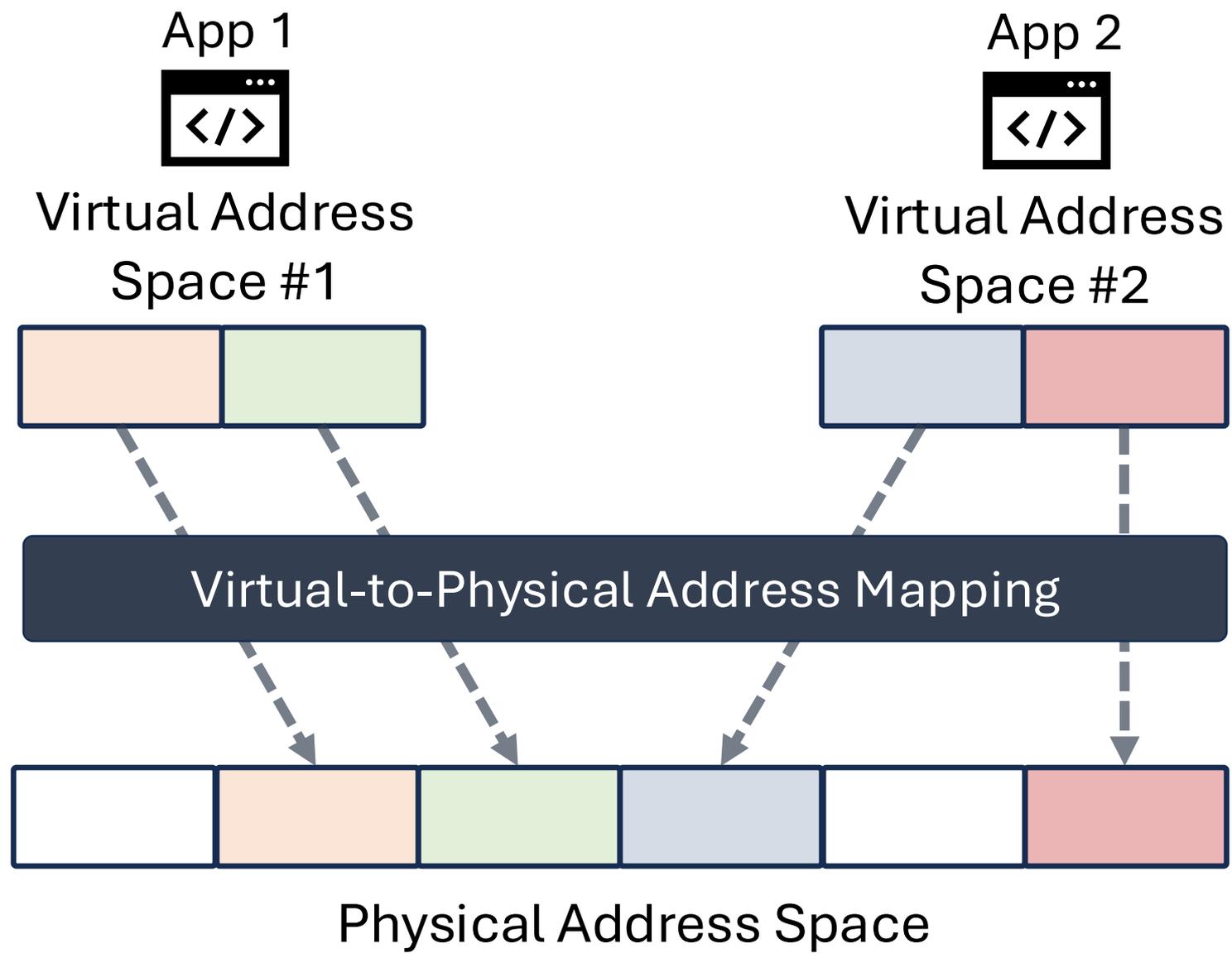


Virtual Address  
Space #2

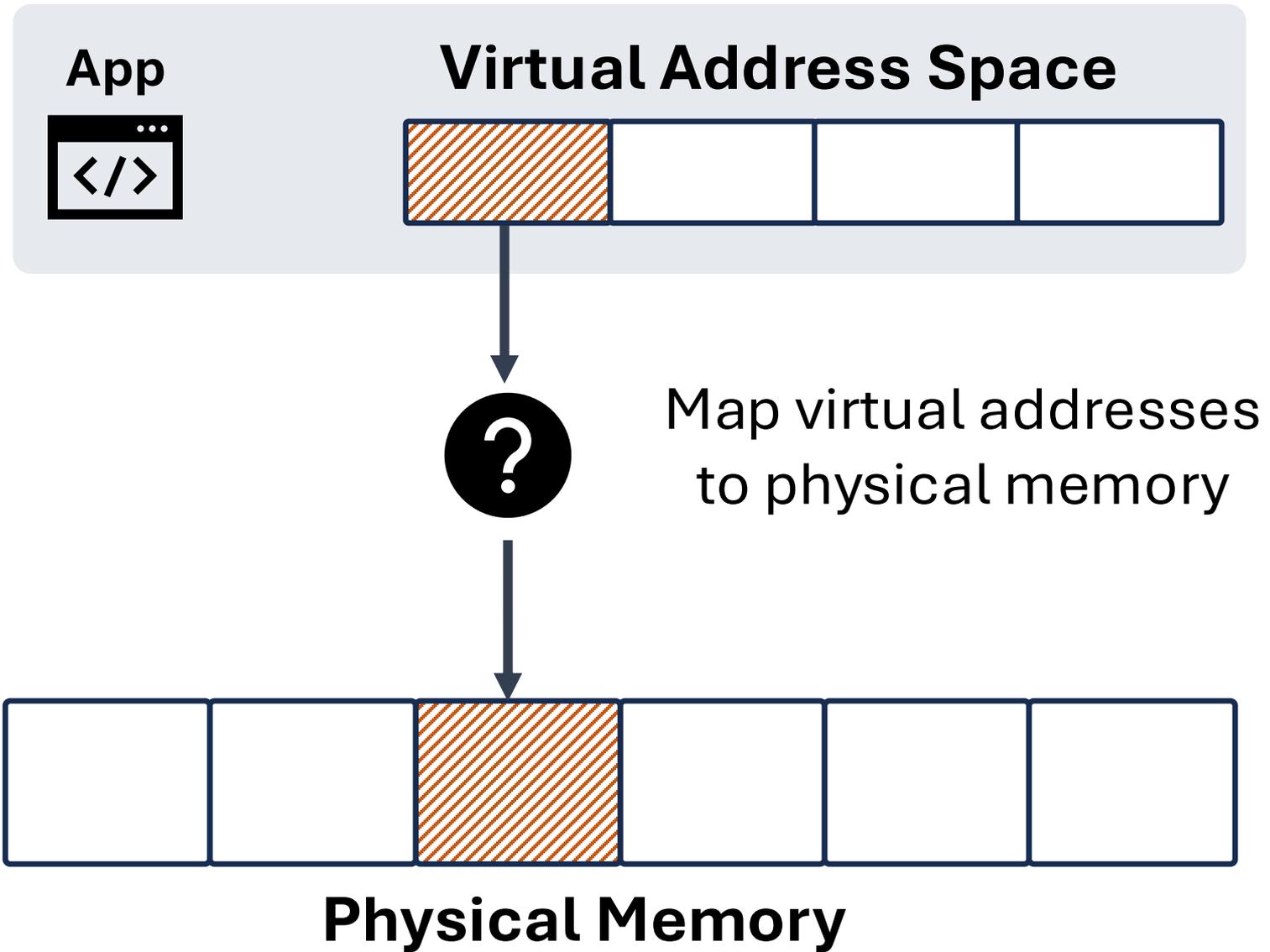


Physical Address Space (e.g., physical memory)

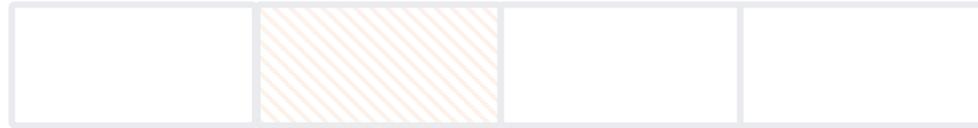




**Who establishes  
virtual-to-physical  
address mappings?**



## Virtual Address Space

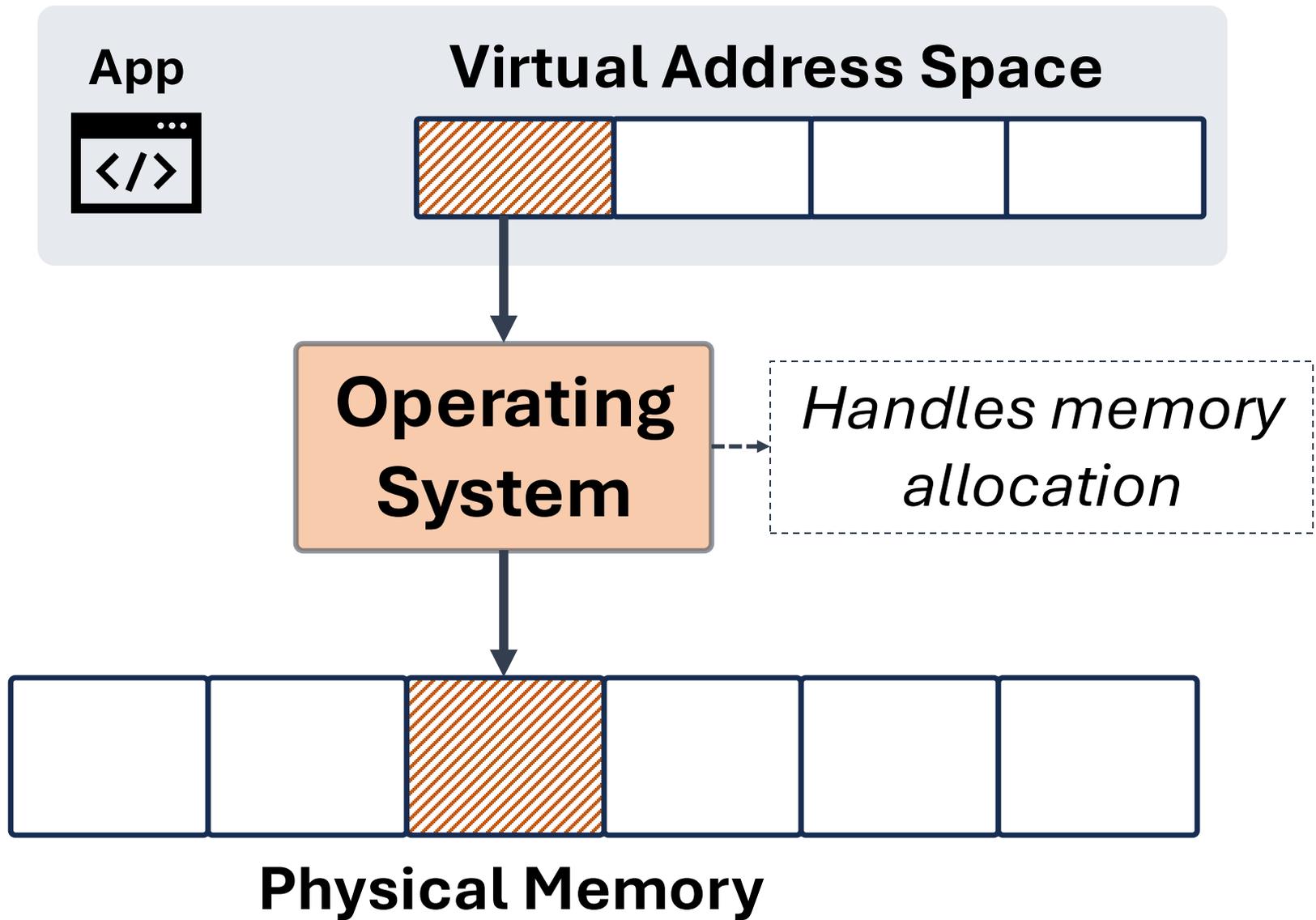


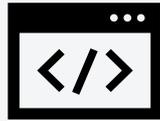
**Fault**

Virtual address  
not mapped to  
physical memory



## Physical Address Space





# Application is stalled

**Operating System**

*Minor Fault*

*Major Fault*

**Finds free space in  
physical memory**

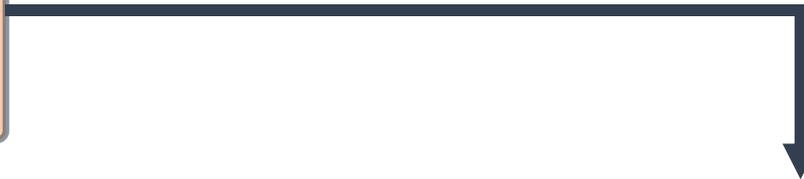
**Fetches data  
from the disk**

## Virtual Address Space

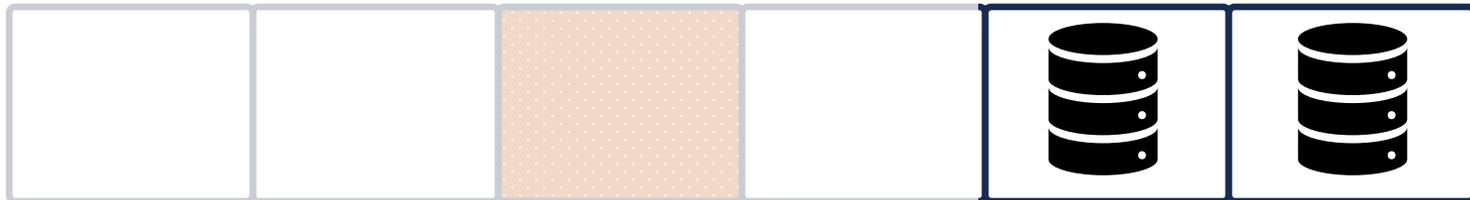


*Store/Update the  
virtual-to-physical address mapping*

**Operating  
System**



## Page Table



## Physical Memory

**How does the CPU  
discover the virtual-  
to-physical mapping?**

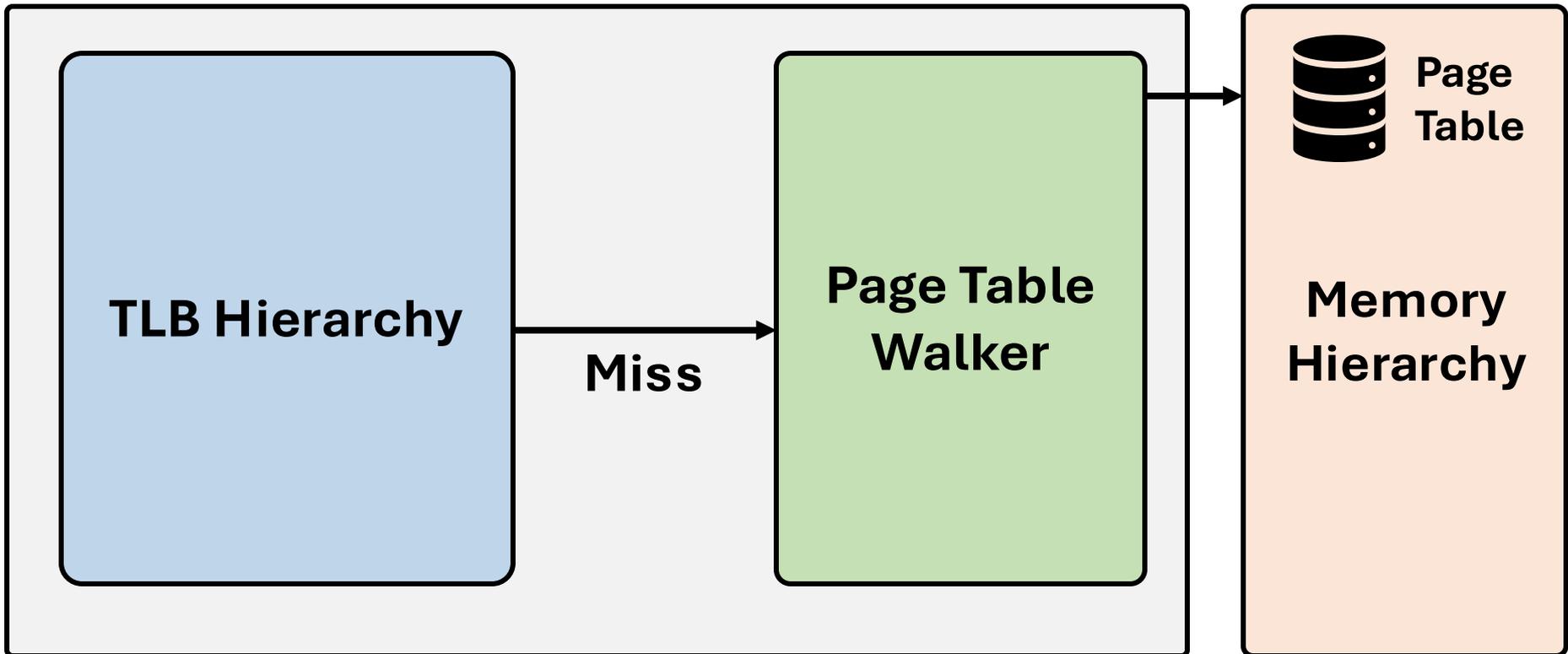
# Address Translation



Hardware unit responsible  
for address translation

# Address Translation

## Memory Management Unit



# Validation against a Real System



Virtuoso

Sniper

## Validation Metrics

MMU  
Performance

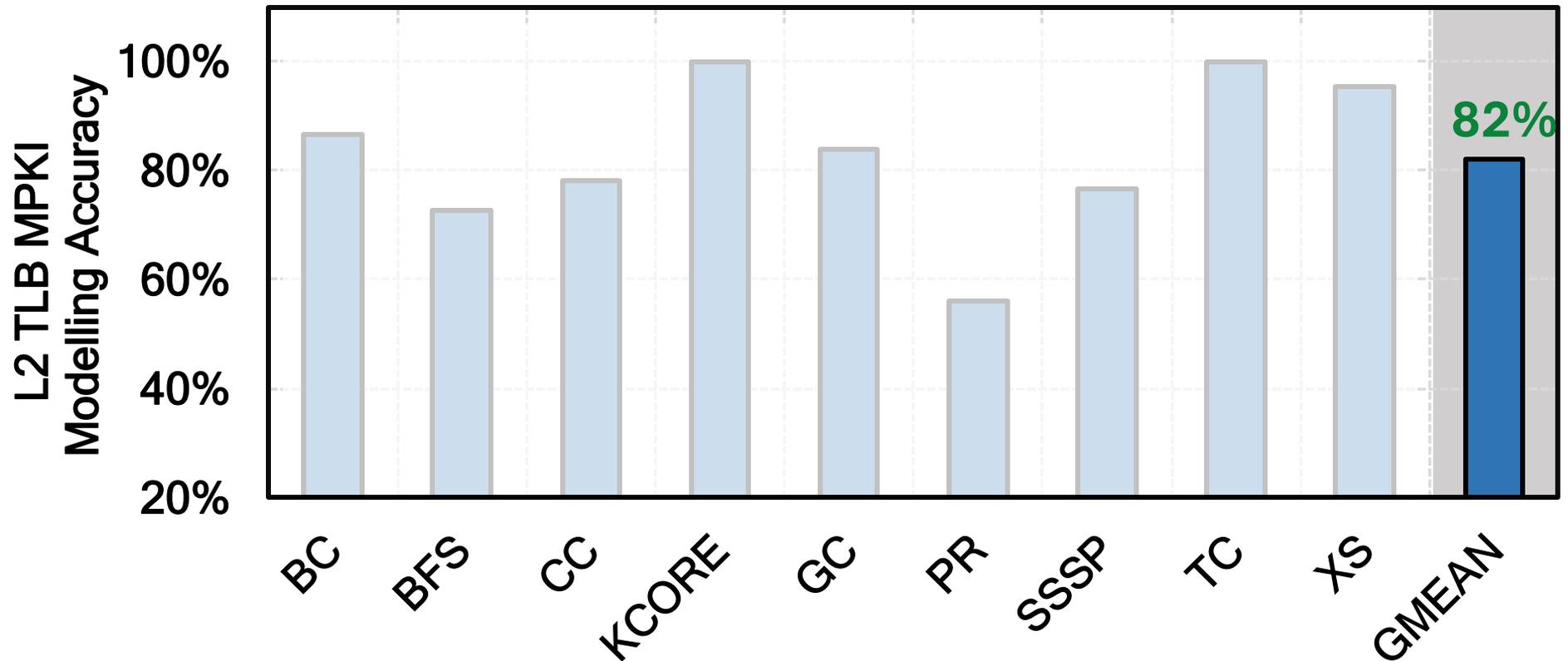
Instructions  
Per Cycle

Page Fault  
Latency

High-end  
Server-grade  
CPU

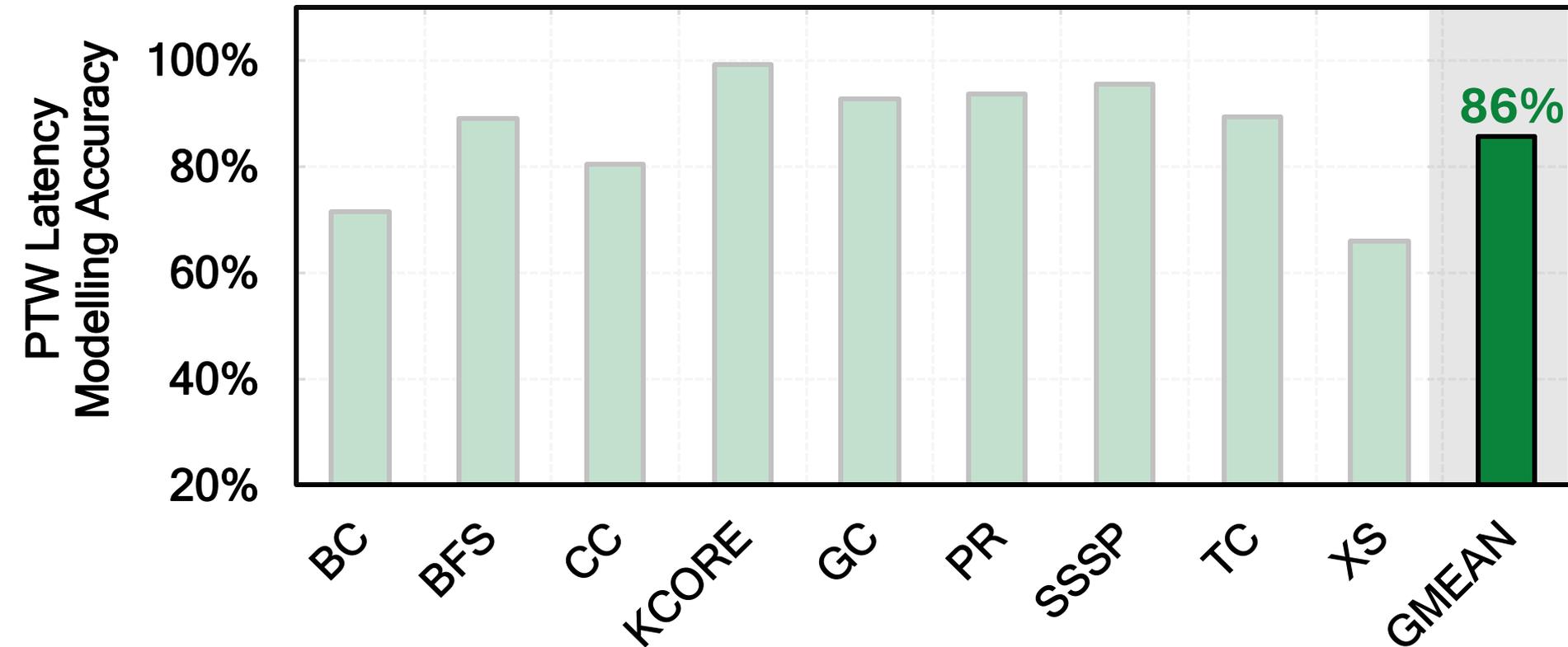


# Validation: L2 TLB MPKI



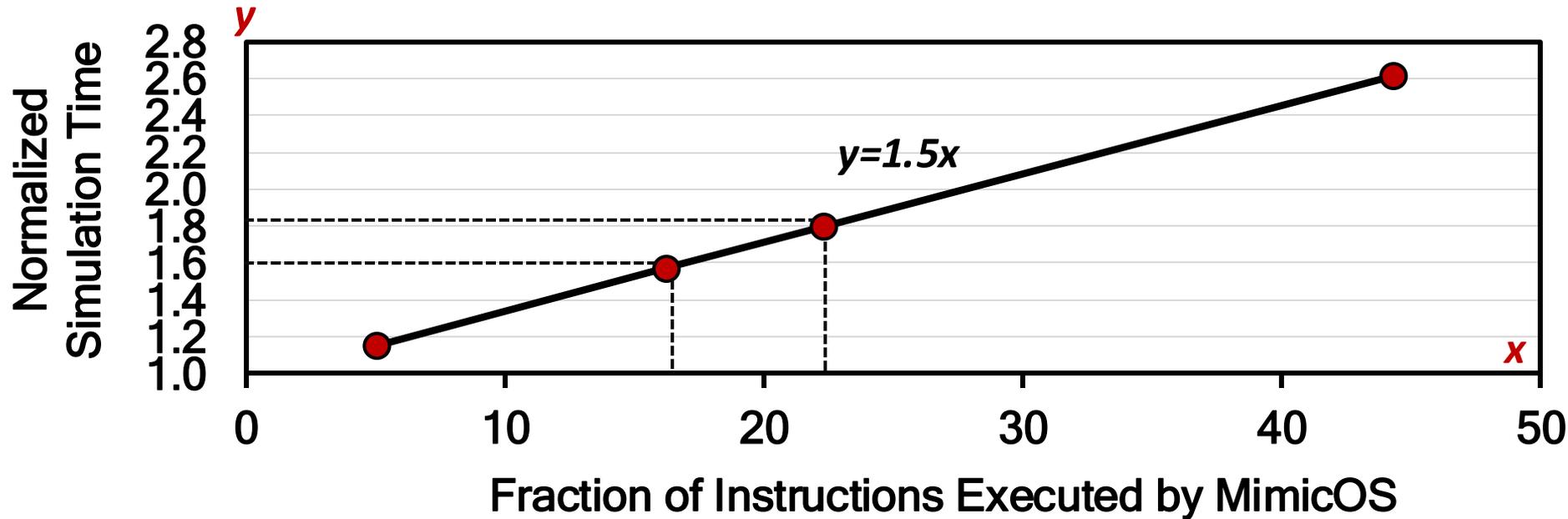
Virtuoso integrated with Sniper models the L2 TLB misses per kilo instructions of a real high-end CPU with 82% accuracy

# Validation: Page Table Walk Latency



Virtuoso integrated with Sniper models the Page Table Walk latency of a real high-end CPU with 86% accuracy

# Instructions vs Simulation Time



Linear relationship between instructions executed by MimicOS and simulation time