The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework

Nastaran Hajinazar Pratyush Patel Minesh Patel
Konstantinos Kanellopoulos Saugata Ghose
Rachata Ausavarungnirun Geraldo F. Oliveira
Jonathan Appavoo Vivek Seshadri Onur Mutlu

Executive Summary

- **Motivation**: Modern computing systems continue to **diversify** with respect to system architecture, memory technologies, and applications' memory needs

- **Problem**: Continually adapting the conventional virtual memory framework to each possible system configuration is challenging
  - Results in performance loss or requires non-trivial workarounds

- **Goal**: Design an alternative virtual memory framework that
  (1) Efficiently supports a wide variety of new system configurations
  (2) Provides the key features and eliminates the key inefficiencies of the conventional virtual memory framework

- **Virtual Block Interface (VBI):**
  Delegates memory management to dedicated hardware in the memory controller
  - Efficiently adapts to diverse system configurations
  - Reduces overheads and complexities associated with conventional virtual memory
  - Enables many optimizations (e.g., low-overhead page walks in virtual machines, virtual caches)

- **Evaluation:** Two example use cases
  1. **VBI** significantly improves performance for both native execution (**2.4x**) and virtual machine environments (**4.3x**)
  2. **VBI** significantly improves heterogeneous memory architecture effectiveness

SAFARI

Outline

Outline

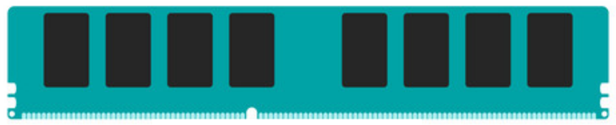
Virtual Memory

Application

Virtual Memory

managed by the operating system

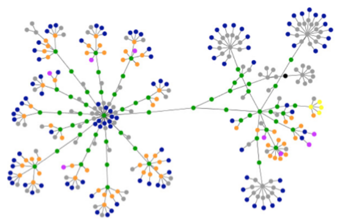
Hardware

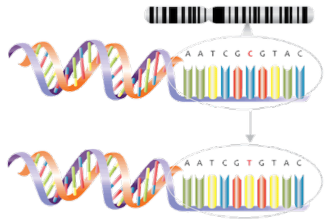


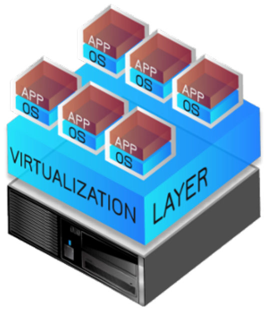
SAFARI

Computing Systems Are Diversifying

Application





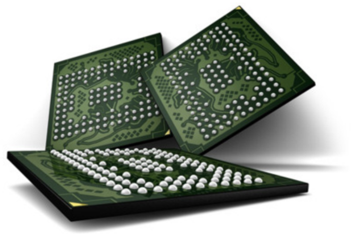


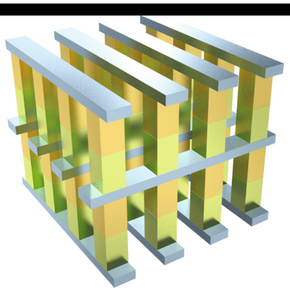
Cannot adapt efficiently

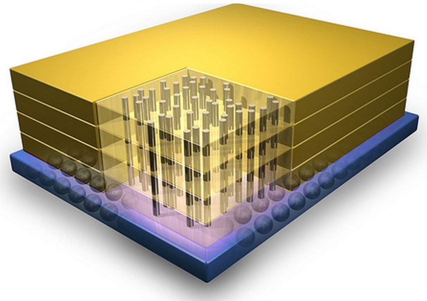
Virtual Memory

managed by the operating system

Hardware







SAFARI

Computing Systems Are Diversifying

Application

Cannot adapt efficiently

Virtual Memory

managed by the operating system

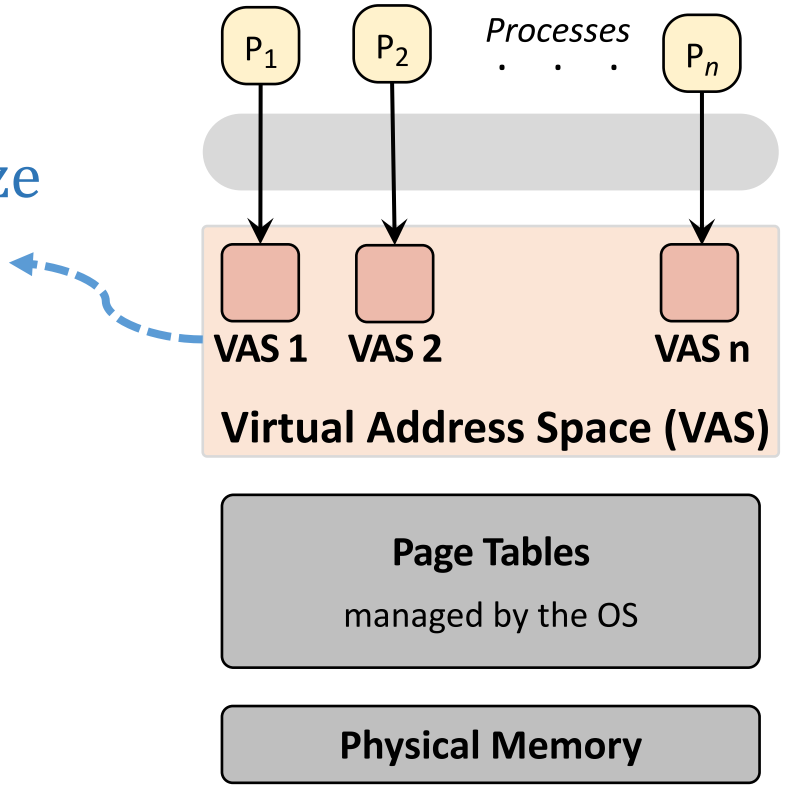
Continually adapting the conventional virtual memory framework is challenging

SAFARI

Conventional Virtual Memory Framework

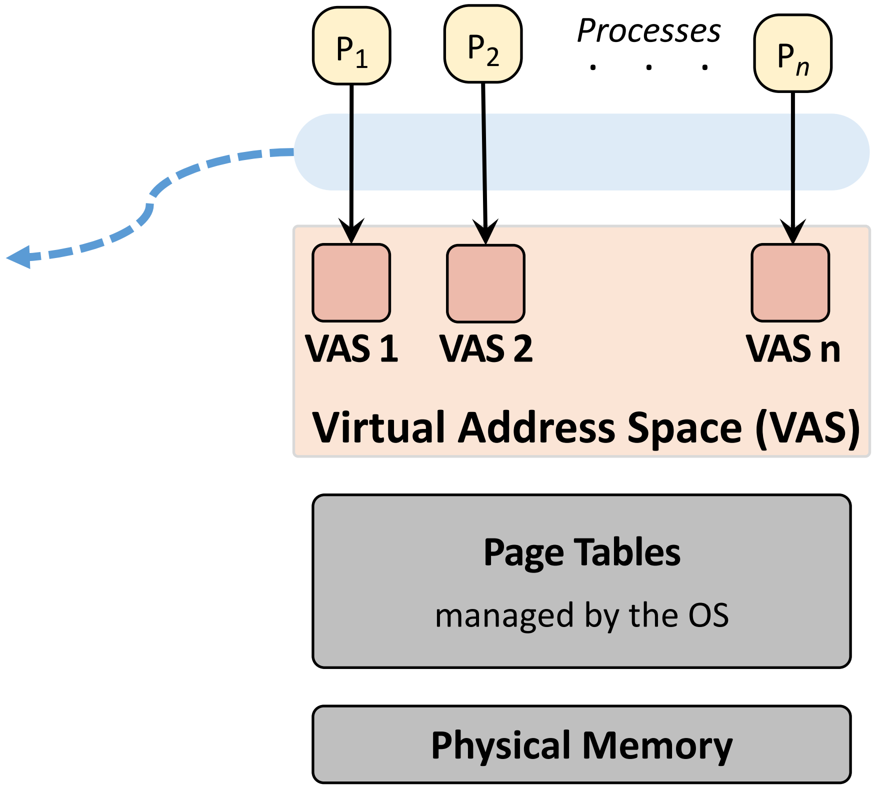
each process is mapped to a fixed-size virtual address space

e.g., 256 TB in Intel x86-64

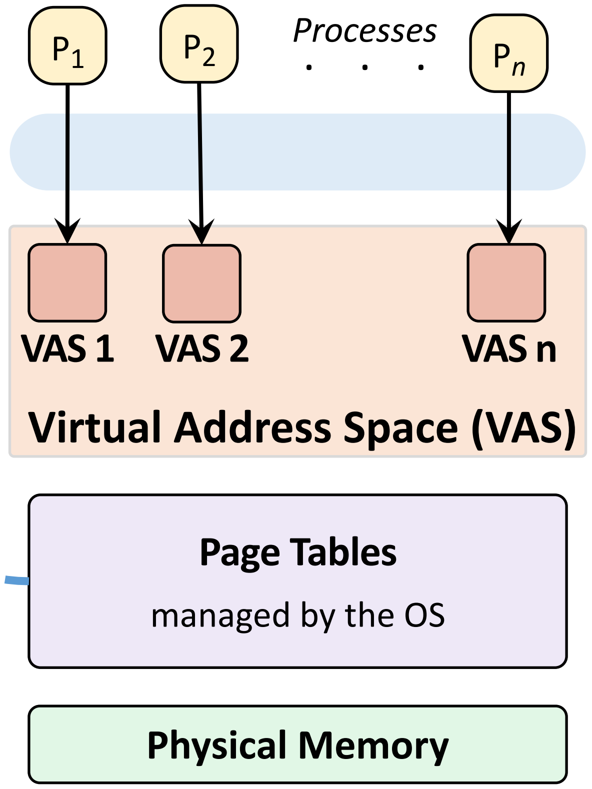


Conventional Virtual Memory Framework

one-to-one mapping
managed by the OS



Conventional Virtual Memory Framework



per-process page tables to
map each VAS to physical memory

managed by the OS

read by hardware

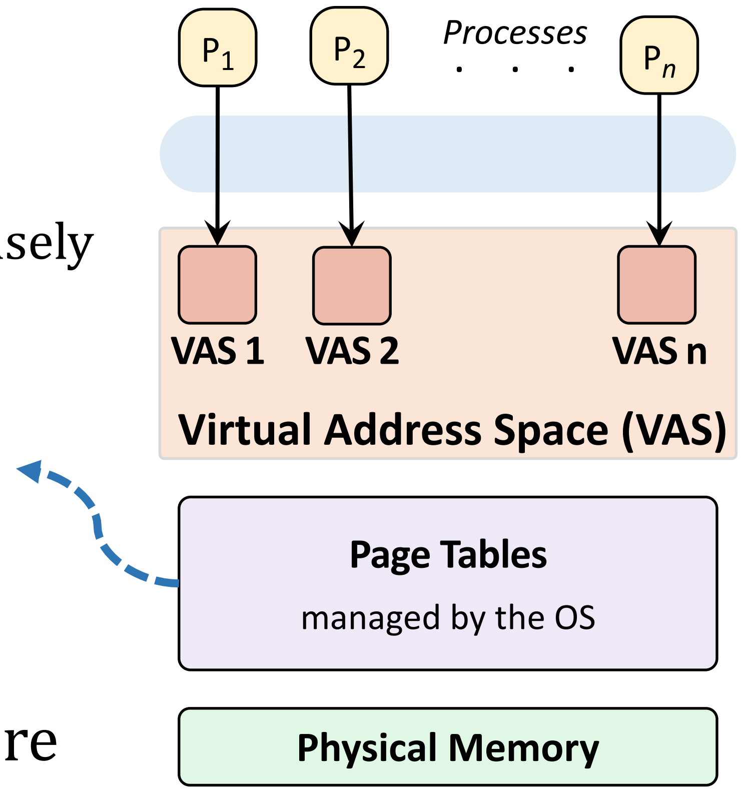
Challenges

- **Three examples** of the **challenges** in adapting conventional virtual memory frameworks for increasingly-diverse systems:
  - Requiring a rigid page table structure
  - High address translation overhead in virtual machines
  - Inefficient heterogeneous memory management

Challenge 1: Rigid Page Table Structures

- **Flexibly customized** page tables can reduce the address translation overhead
  - Customized to the application's memory behavior
    - e.g., larger granularities for more densely allocated memory regions

accessed by both **OS** and **hardware**

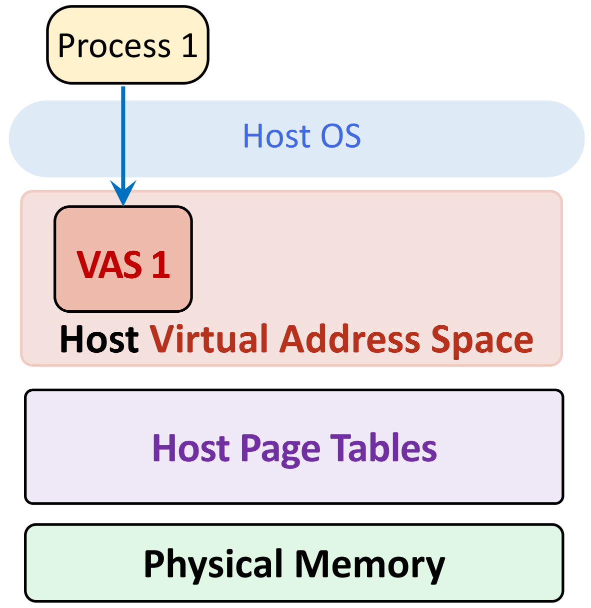


- **Con:**
  - Requires a rigid page table structure
    - e.g., fixed-granularity 4-level page table in Intel x86

Challenges

- **Three examples** of the **challenges** in adapting conventional virtual memory frameworks for increasingly-diverse systems:
  - Requiring a rigid page table structure
  - High address translation overhead in virtual machines
  - Inefficient heterogeneous memory management

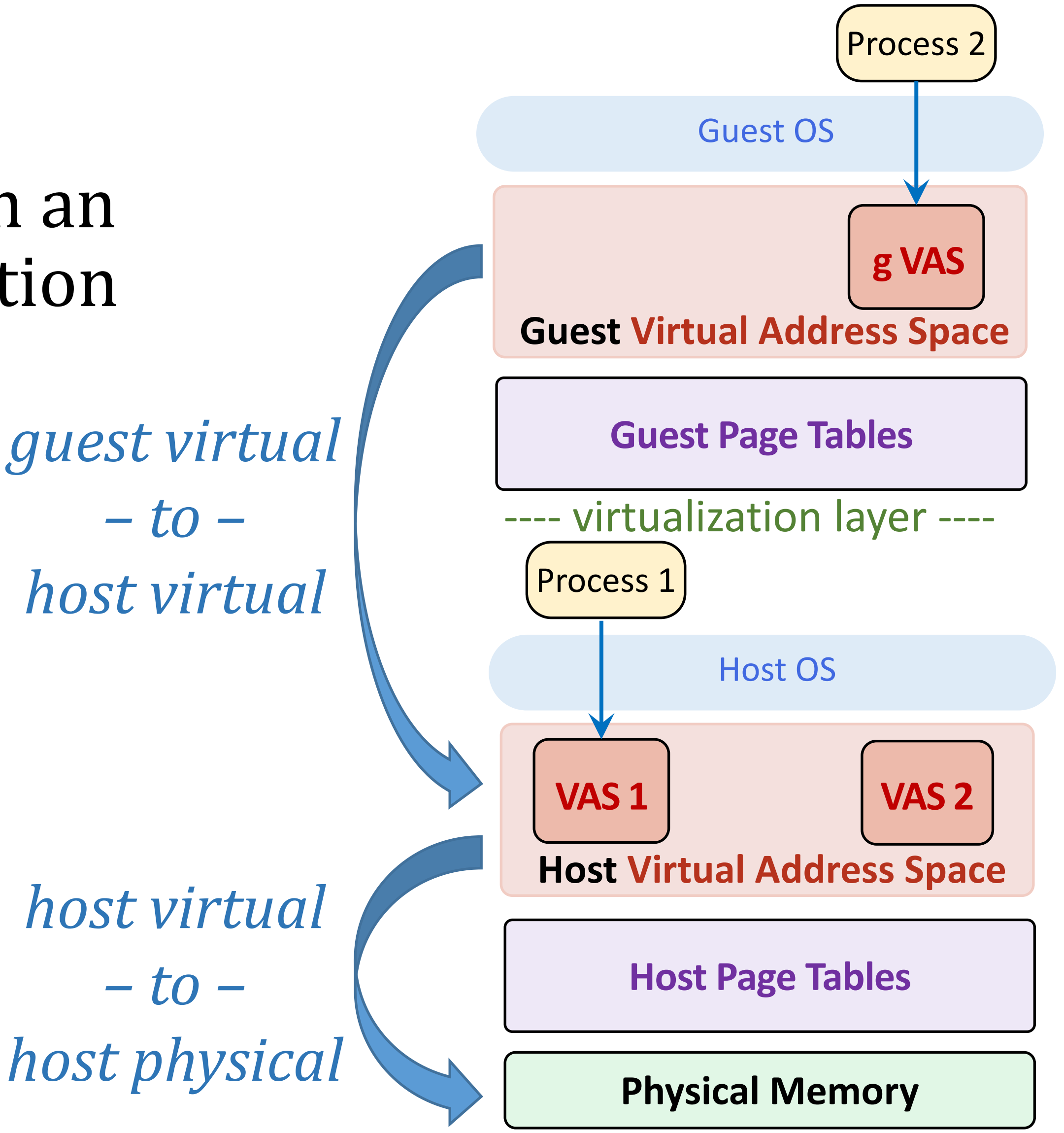
Challenge 2: Overheads in Virtual Machines



Challenge 2: Overheads in Virtual Machines

- In virtual machines, processes go through an extra level of indirection

- **Con:**
  - 2D page table walks

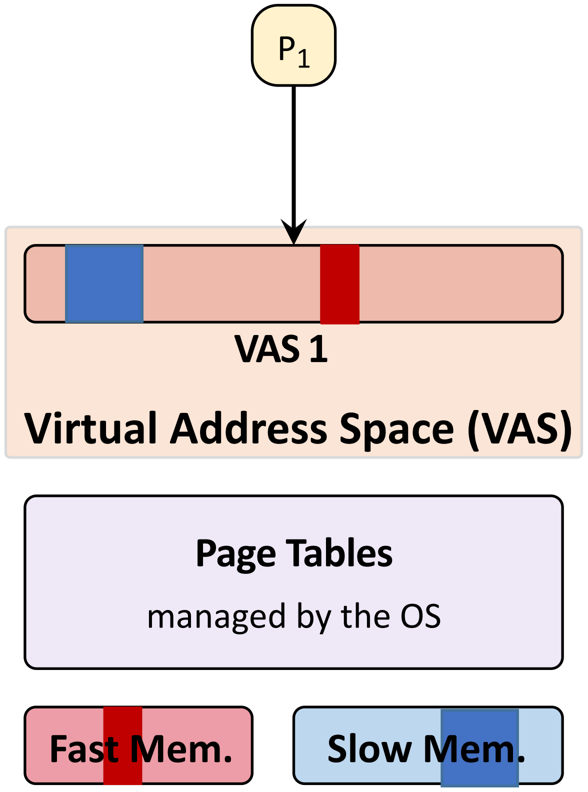

Process 2
Guest OS
g VAS
Guest Virtual Address Space
Guest Page Tables
---- virtualization layer ----
Process 1
Host OS
VAS 1
VAS 2
Host Virtual Address Space
Host Page Tables
Physical Memory
guest virtual – to – host virtual
host virtual – to – host physical

Challenges

- **Three examples** of the **challenges** in adapting conventional virtual memory frameworks for increasingly-diverse systems:
  - Requiring a rigid page table structure
  - High address translation overhead in virtual machines
  - Inefficient heterogeneous memory management

Challenge 3: Managing Heterogeneous Memory

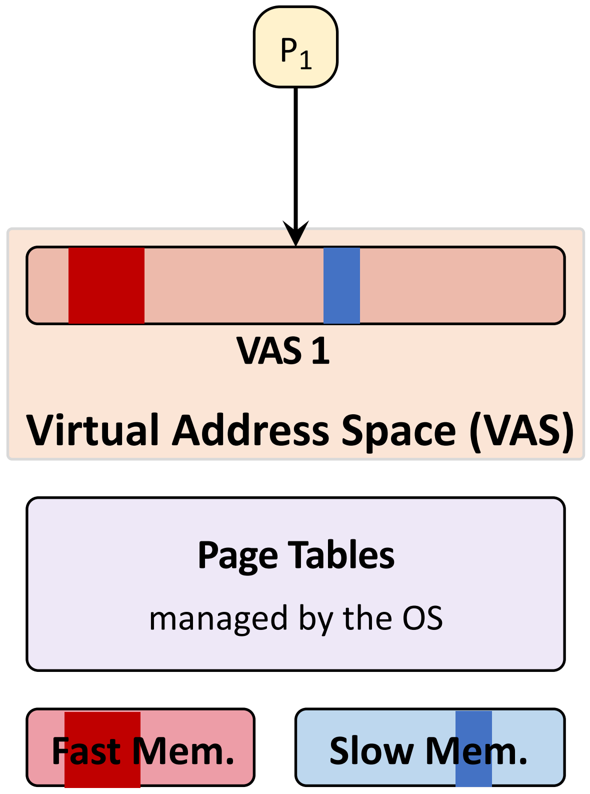
- Enhancing performance with heterogenous memories requires:
  - Data mapping



SAFARI

Challenge 3: Managing Heterogeneous Memory

- Enhancing performance with heterogenous memories requires:
  - Data mapping
  - Data migration



- **Con:**
  - OS has low visibility into runtime memory behavior
    - Timely reaction to the changes is challenging

SAFARI

Prior Works

- Optimizations that **alleviate the overheads** of the conventional virtual memory framework

Shortcomings:

- Based on **specific** system or workload characteristics
  - Are applicable to only limited problems or applications

- Require **specialized** and **not necessarily compatible** changes to both the OS and hardware
  - Implementing all in a system is a daunting prospect

Prior Works

- Optimizations that **alleviate the overheads** of the conventional virtual memory framework

Shortcomings:

- Based on **specific** system or workload characteristics
  - Are applicable to only limited problems or applications

We need a holistic solution that efficiently supports increasingly diverse system configurations

Our Goal

Design an alternative virtual memory framework that

- Efficiently and flexibly supports increasingly diverse system configurations
- Provides the key features of conventional virtual memory framework while eliminating its key inefficiencies

Outline

Virtual Block Interface (VBI)

VBI is an alternative virtual memory framework

Key idea:

Delegate physical memory management to dedicated hardware in the **memory controller**

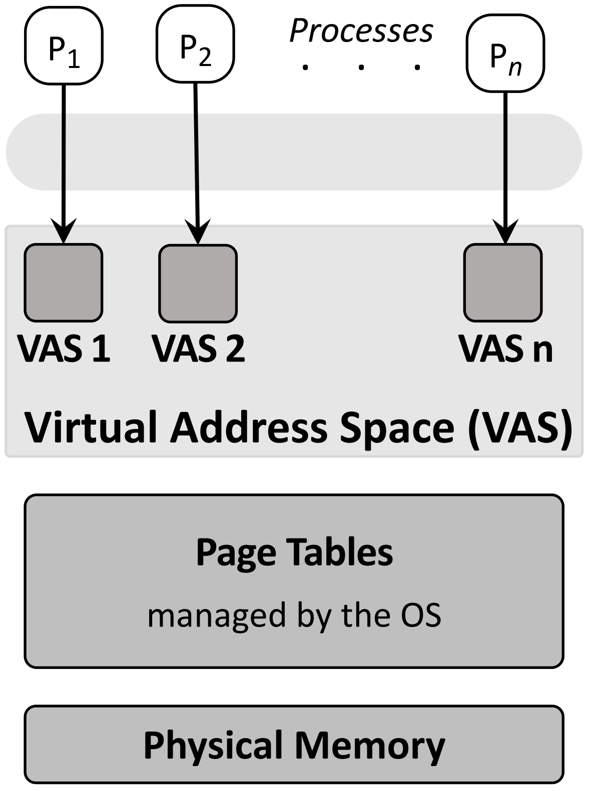
VBI: Guiding Principles

- **Size virtual address spaces appropriately for processes**
  - Mitigates translation overheads of unnecessarily large address spaces
- **Decouple address translation from access protection**
  - Defers address translation until necessary to access memory
  - Enables the flexibility of managing translation and protection using separate structures
- **Communicate data semantics to the hardware**
  - Enables intelligent resource management

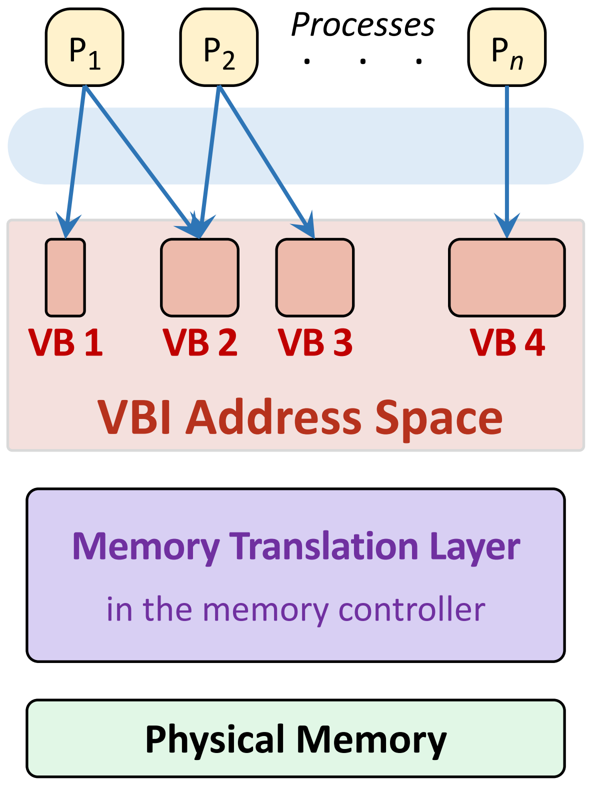
Addresses the rigidness and lack of information in current frameworks, to reduce large overheads

Outline

VBI: Overview



Conventional Virtual Memory

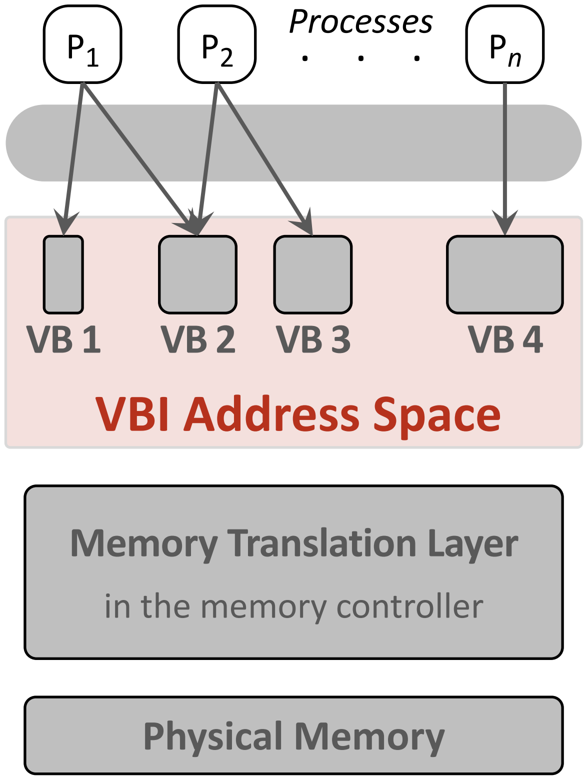


VBI

SAFARI

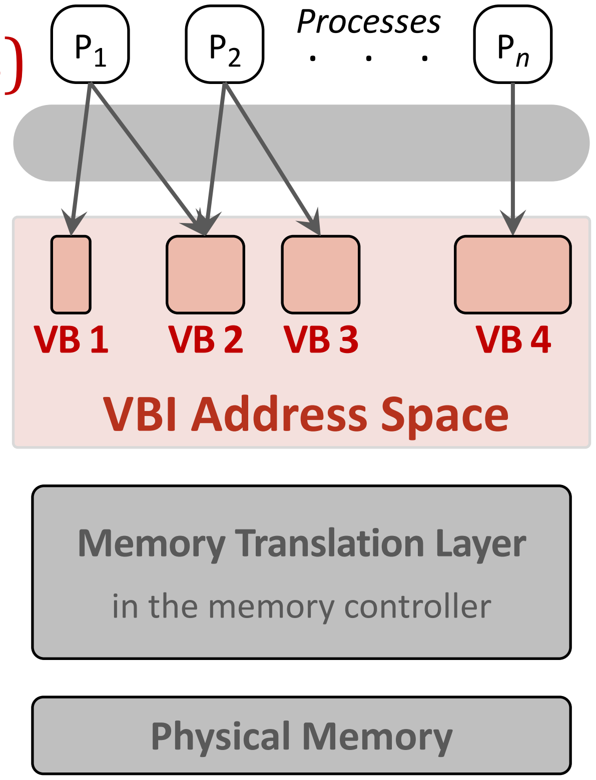
Virtual Blocks

- Globally-visible ***VBI address space***



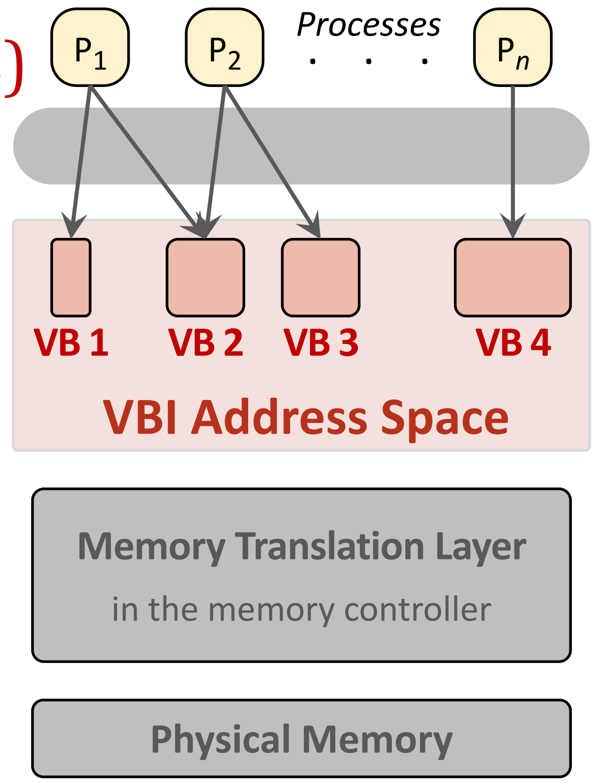
Virtual Blocks

- **Globally-visible *VBI address space***
  - Consists of a set of ***virtual blocks*** (VBs) of different sizes
    - Example size classes: 4 KB, 128 KB, 4 MB, 128 MB, 4 GB, 128 GB, 4 TB, 128 TB



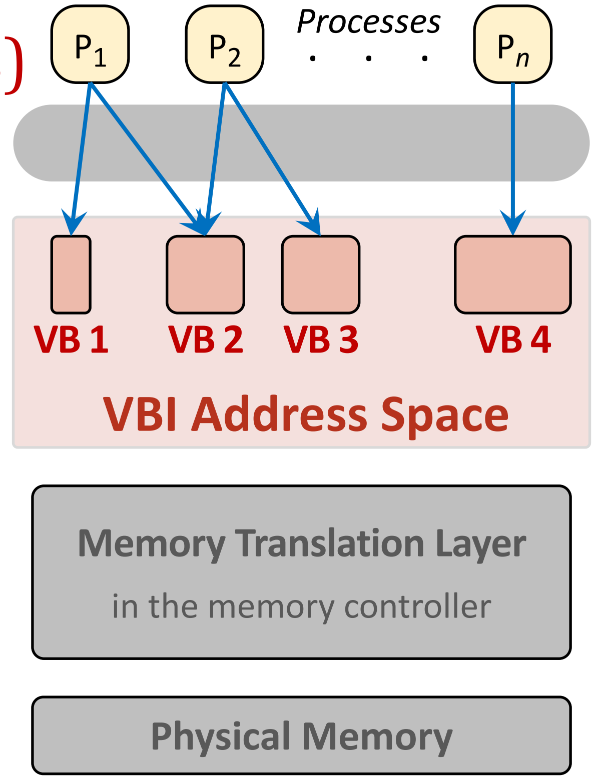
Virtual Blocks

- **Globally-visible *VBI address space***
  - Consists of a set of ***virtual blocks*** (VBs) of different sizes
    - Example size classes: 4 KB, 128 KB, 4 MB, 128 MB, 4 GB, 128 GB, 4 TB, 128 TB
- All VBs are visible to all processes



Virtual Blocks

- **Globally-visible *VBI address space***
  - Consists of a set of ***virtual blocks*** (VBs) of different sizes
    - Example size classes: 4 KB, 128 KB, 4 MB, 128 MB, 4 GB, 128 GB, 4 TB, 128 TB
- All VBs are visible to all processes
- Processes map each **semantically meaningful unit of information** to a separate VB
  - e.g., a data structure, a shared library

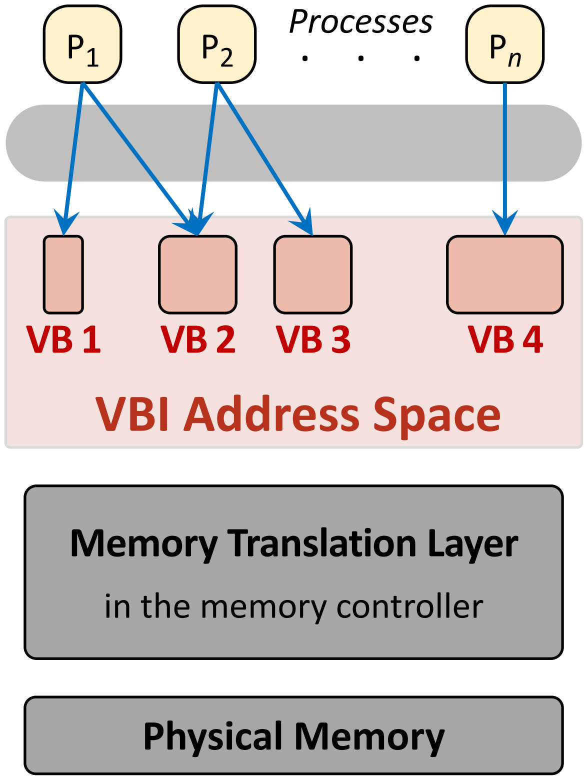


Inherently Virtual Caches

- VBI address space provides ***system-wide* unique VBI addresses**

- **VBI addresses** are **directly** used to access on-chip caches
  - No longer require address translation

- **Pros:**
  - Enables inherently virtual caches
    - no synonyms and homonyms

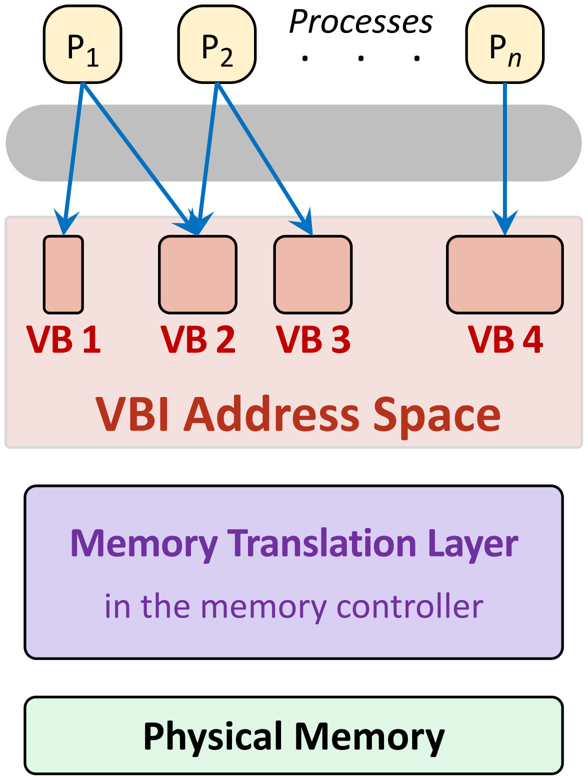


SAFARI

Hardware-Managed Memory

- Memory management is **delegated** to the **Memory Translation Layer (MTL)** in the memory controller
  - Address translation
  - Physical memory allocation

- **Pros:** Many benefits, including
  - Physical memory is allocated only when the location needs to be written to memory
  - No need for 2D page walks in virtual machines
  - Enabling flexible translation structures

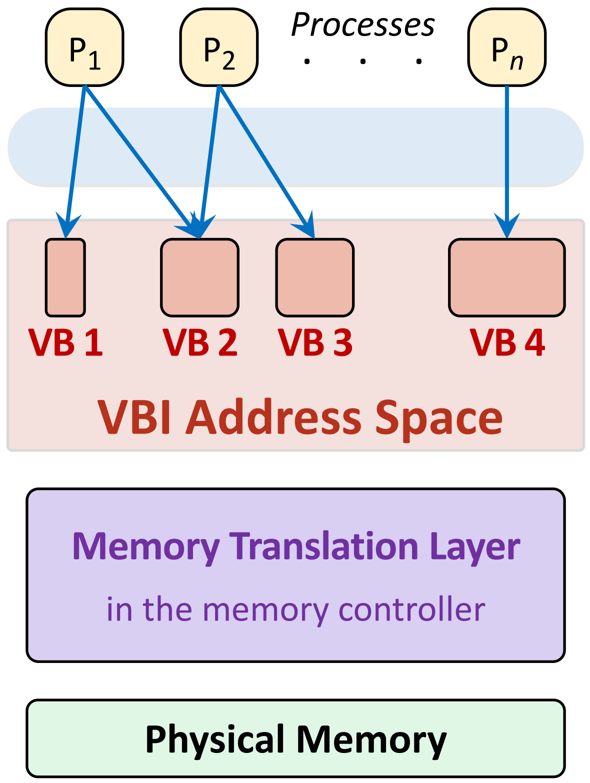


OS-Managed Access Permissions

- OS controls which processes access which VBs

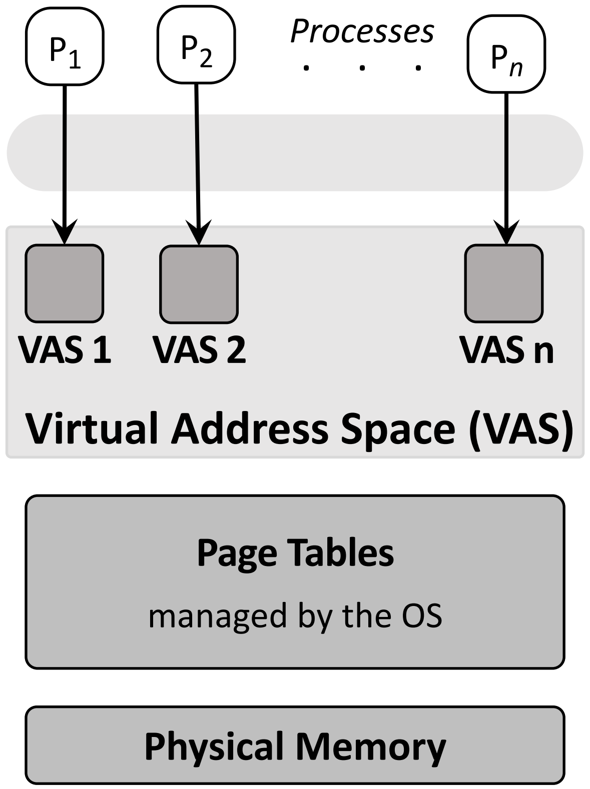
- Each process has its own permissions (read/write/execute) when ***attaching*** to a VB

- OS maintains a list of VBs attached to each process
  - Stored in a per-process table
  - Used during permission checks



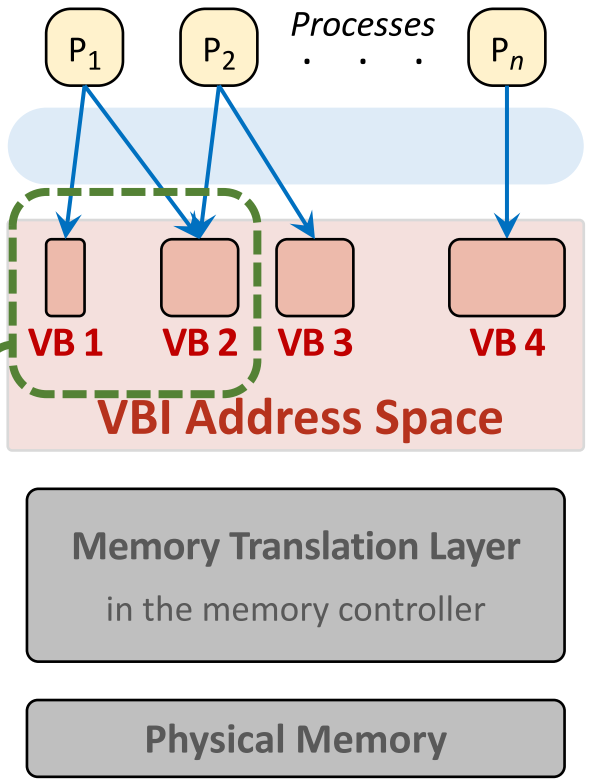
Process Address Space

Conventional virtual memory



Process Address Space in VBI

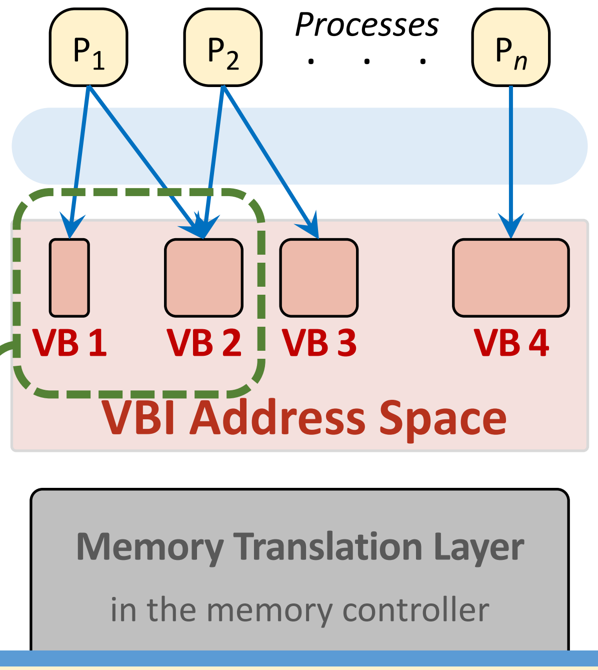
- Any process can attach to any VB
- A process' VBs define its **address space**
    - Address space size is determined by the actual needs of the process



the address space of process P_1

Process Address Space in VBI

- Any process can attach to any VB
- A process' VBs define its **address space**
    - Address space size is determined by the actual needs of the process



the address space of

First guiding principle:

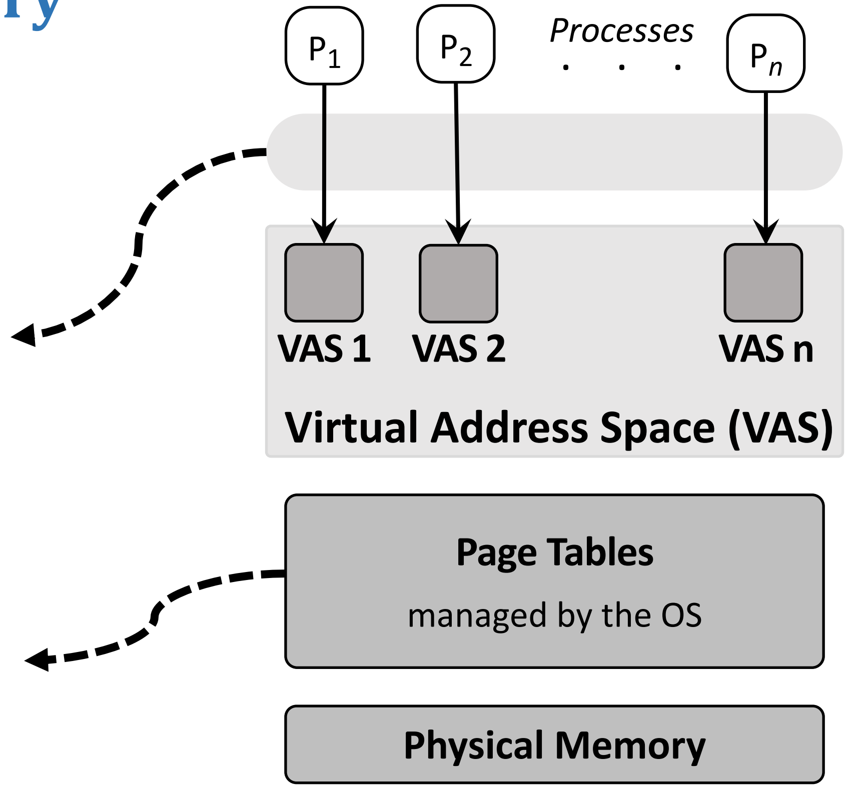
Appropriately-sized virtual address spaces

Decoupled Protection and Translation

Conventional virtual memory

Access permissions managed by OS

Address mapping managed by OS

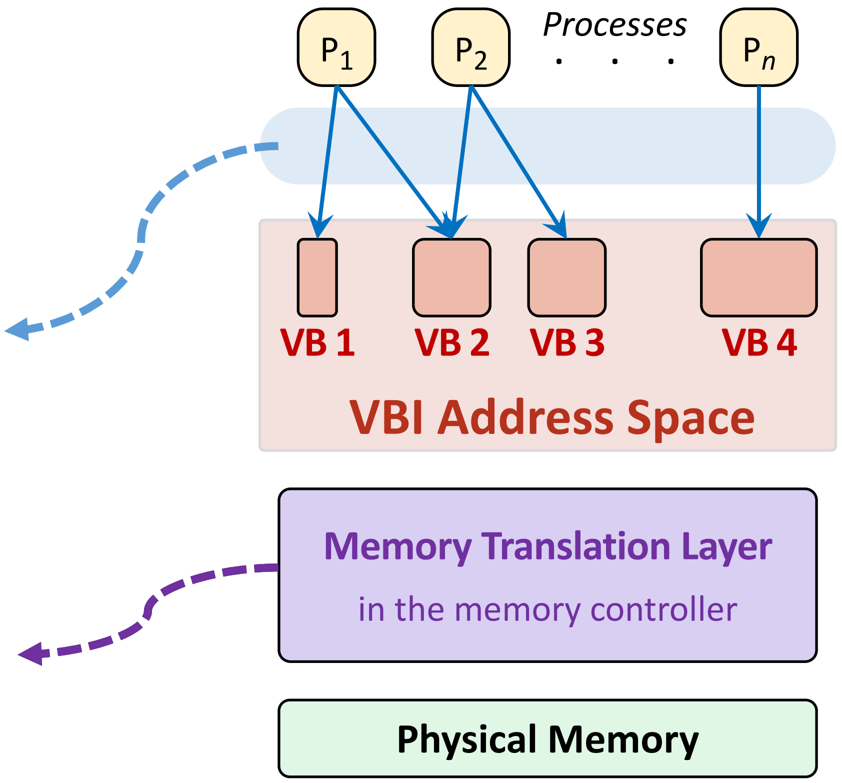


Decoupled Protection and Translation

VBI

Access permissions managed by OS

Address mapping managed by the MTL

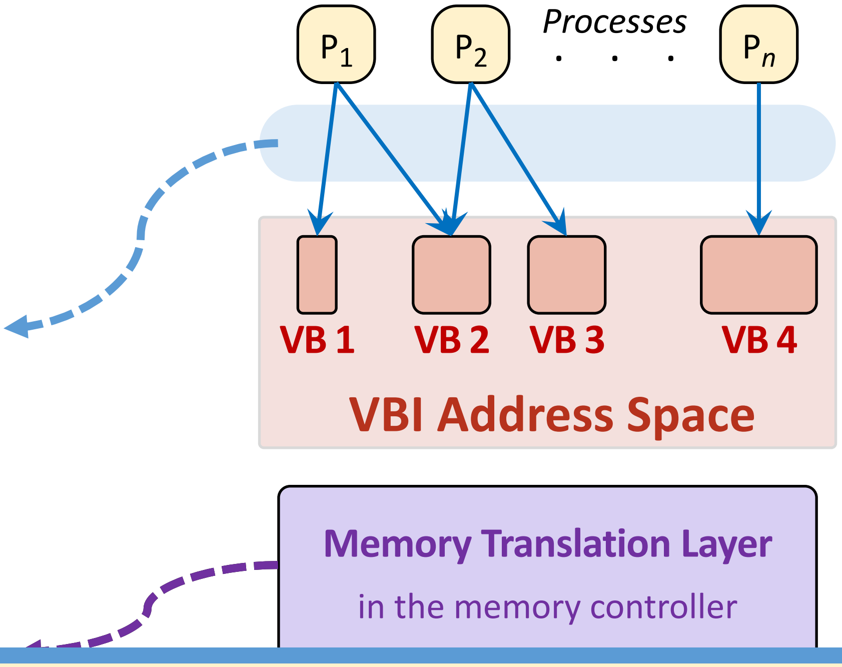


Decoupled Protection and Translation

VBI

Access permissions managed by OS

Address mapping



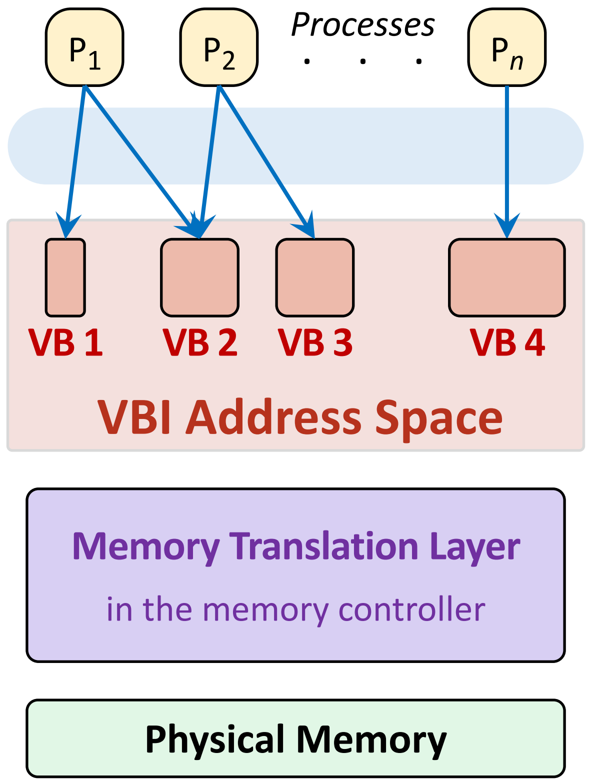
Second guiding principle:

Decoupling address translation from access protection

SAFARI

Address Translation Structures in VBI

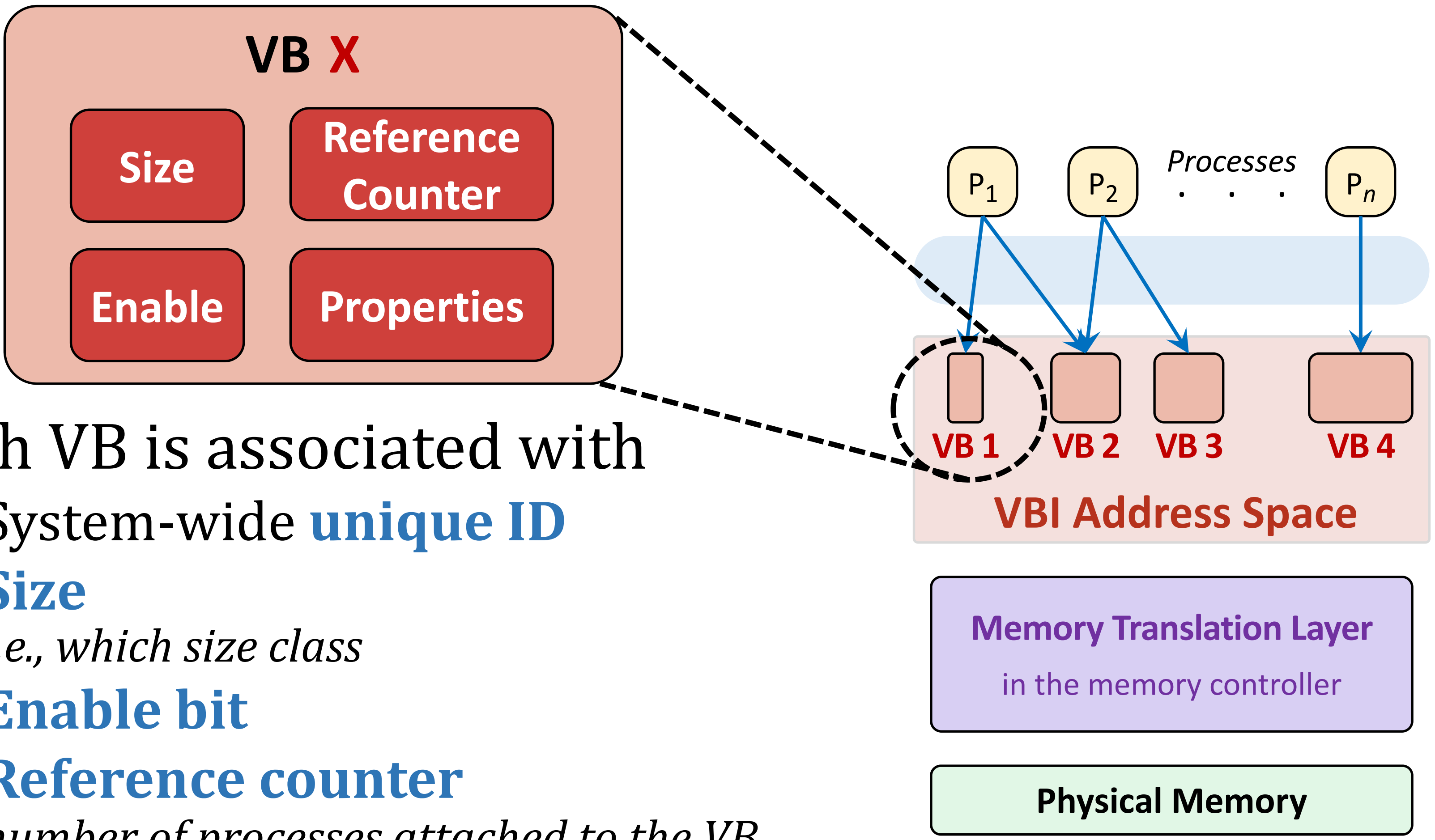
- Translation structures are **not shared** with the OS
  - Separate structures for translation and permission information
  - Allows flexible translation structures
  - Per-VB translation structure tuned to the VB's characteristics
    e.g., single-level tables for small VBs



- **Pros:**
  - Lowers overheads and allows for customization

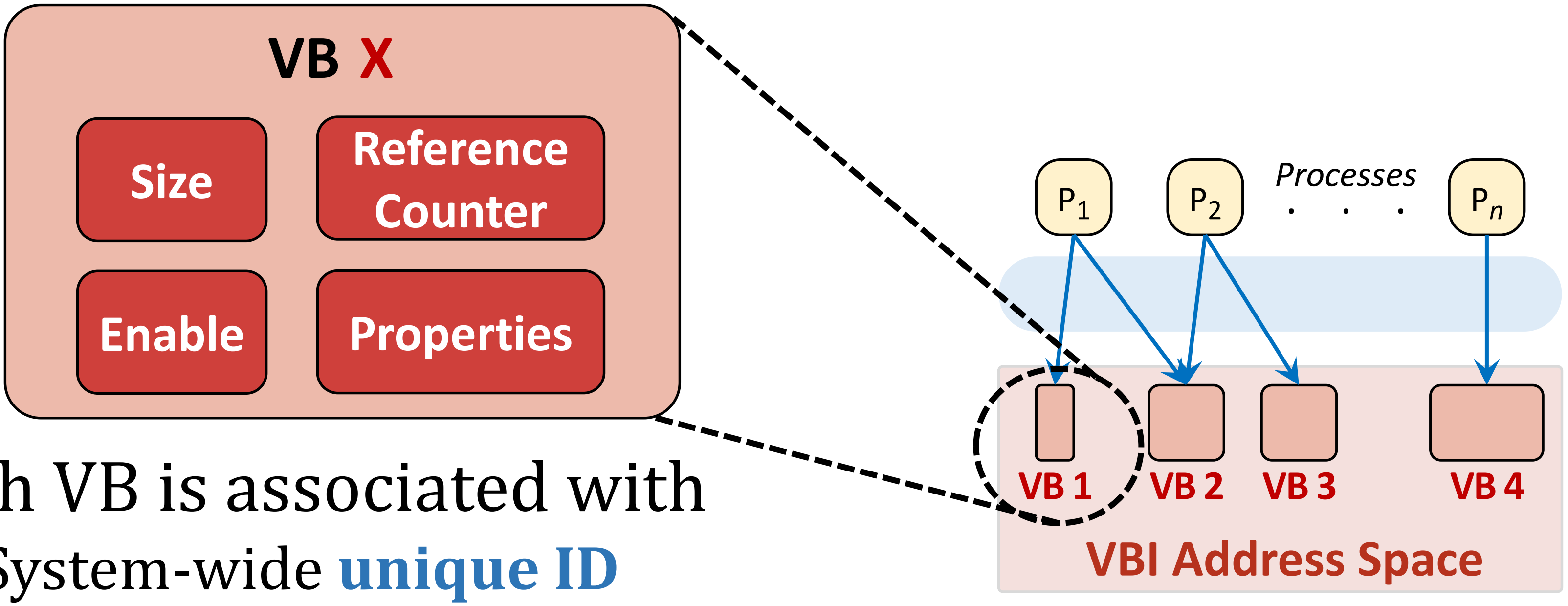
SAFARI

VB Information



- Each VB is associated with
  - System-wide **unique ID**
  - **Size**
    i.e., which size class
  - **Enable bit**
  - **Reference counter**
    number of processes attached to the VB
  - **Properties bit vector**
    semantic information about VB contents,
    e.g., access pattern, latency sensitive vs. bandwidth sensitive

VB Information



- Each VB is associated with
  - System-wide **unique ID**
  - **Size**
    i.e., which size class
  - **Enable bit**

Memory Translation Layer
in the memory controller

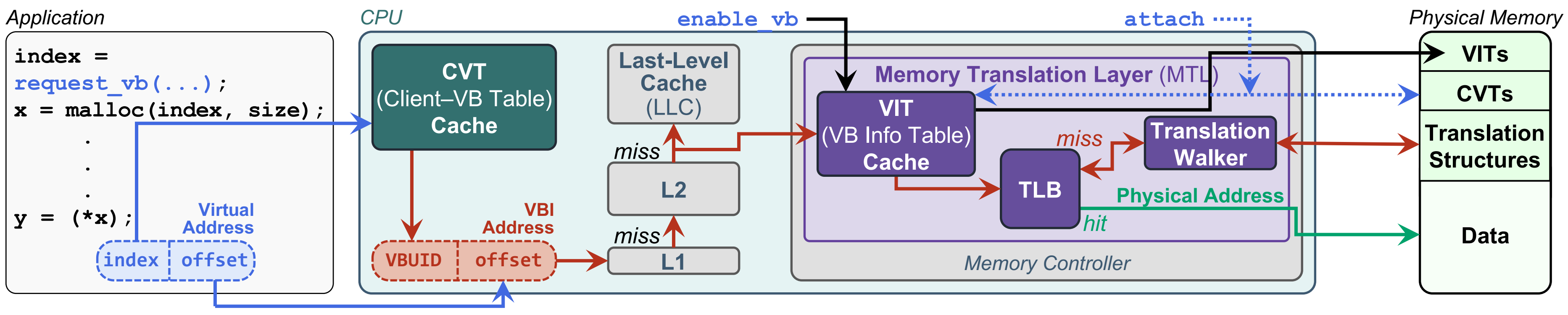
Third guiding principle:

Communicating data semantics to the hardware

SAFARI

Implementing VBI

- Please refer to our paper
  - Detailed reference implementation and microarchitecture



Outline

Optimizations Naturally Enabled by VBI

- **Many optimizations not easily attainable before**

- **Examples:**
  - Appropriately sized process address space
  - Flexible address translation structures
  - Communicating data semantics to the hardware

  Achieved through guiding principles

  - Inherently virtual caches
  - Delayed physical memory allocation
  - Eliminating 2D page walks in virtual machines

  Covered next

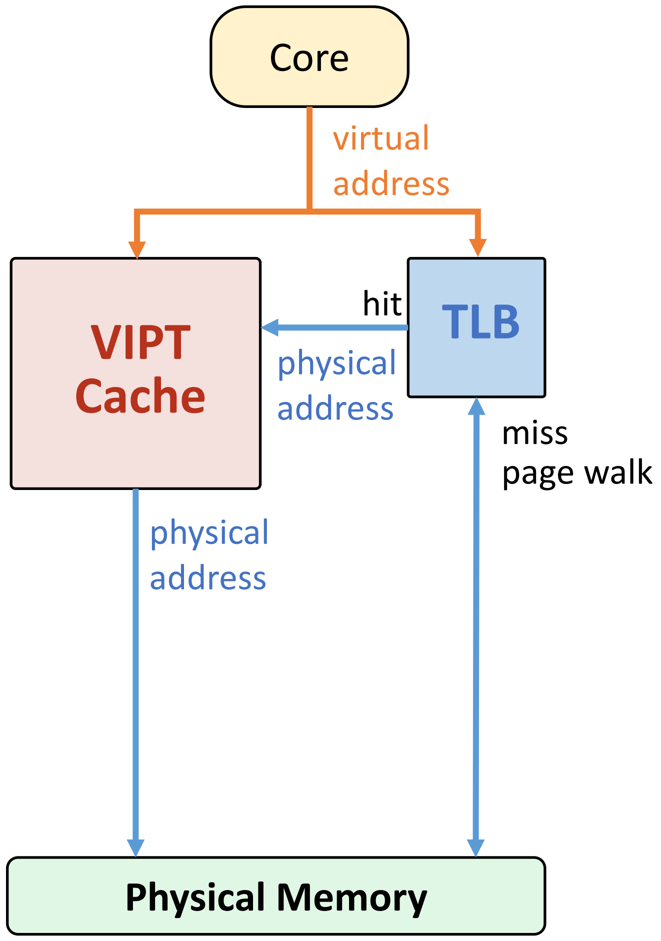
  - Early memory reservation mechanism ***Covered in our paper***

SAFARI

Example Optimizations

- Inherently virtual caches
- Delayed physical memory allocation
- Eliminating 2D page walks in virtual machines

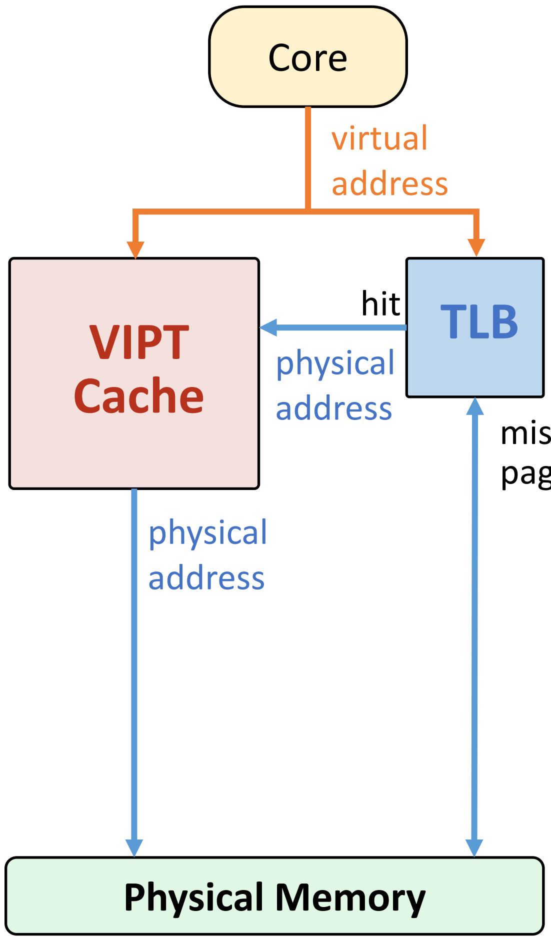
Inherently Virtual Caches



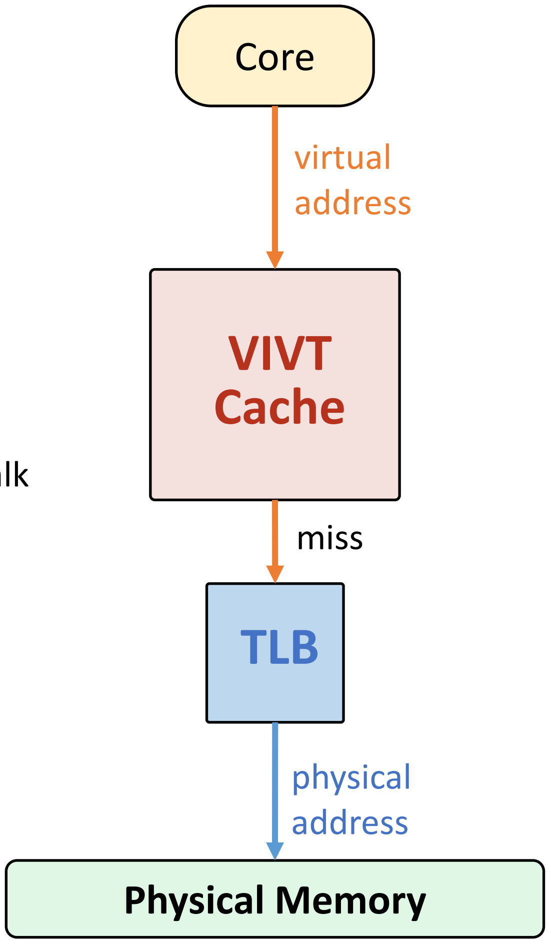
virtually-indexed physically-tagged (VIPT)

In Conventional Virtual Memory

Inherently Virtual Caches



virtually-indexed physically-tagged (VIPT)

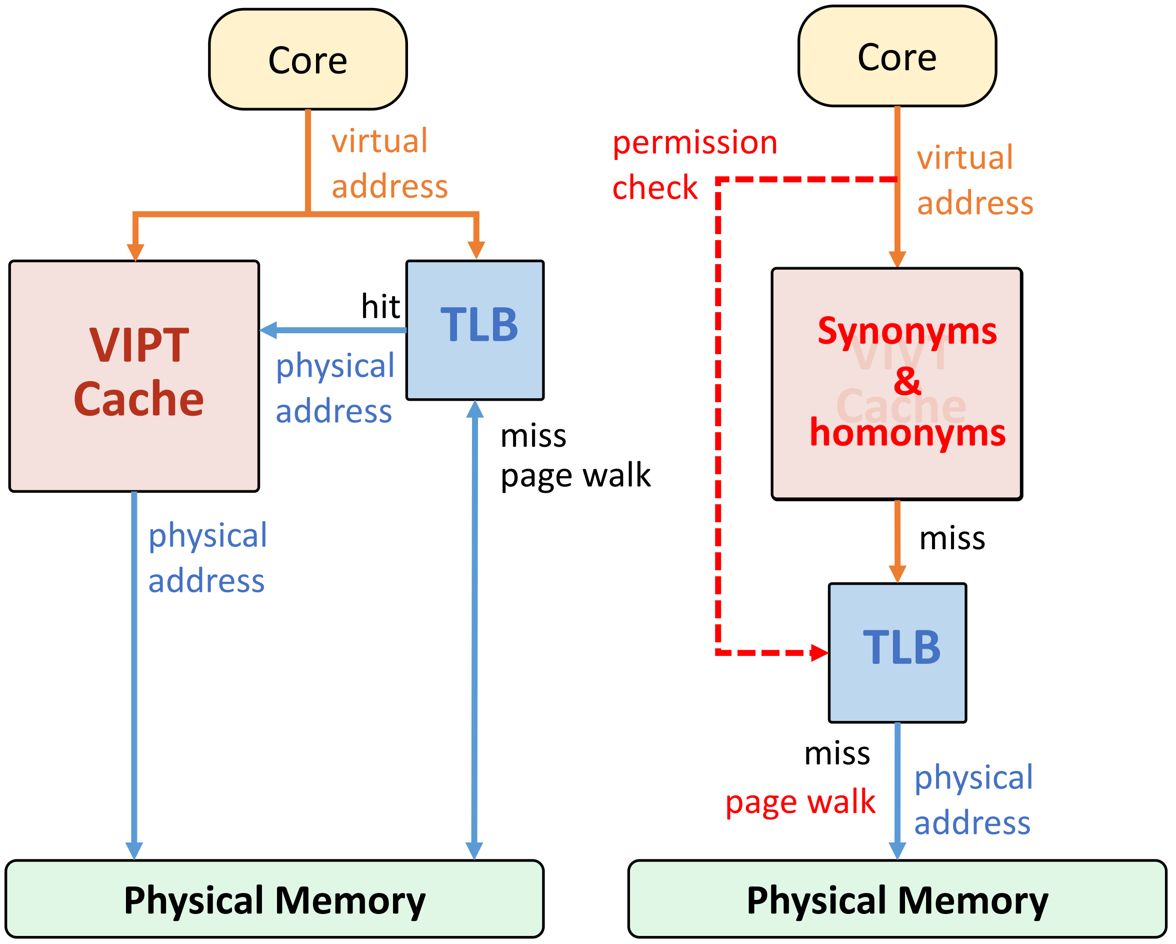


virtually-indexed virtually-tagged (VIVT)

In Conventional Virtual Memory

SAFARI

Inherently Virtual Caches



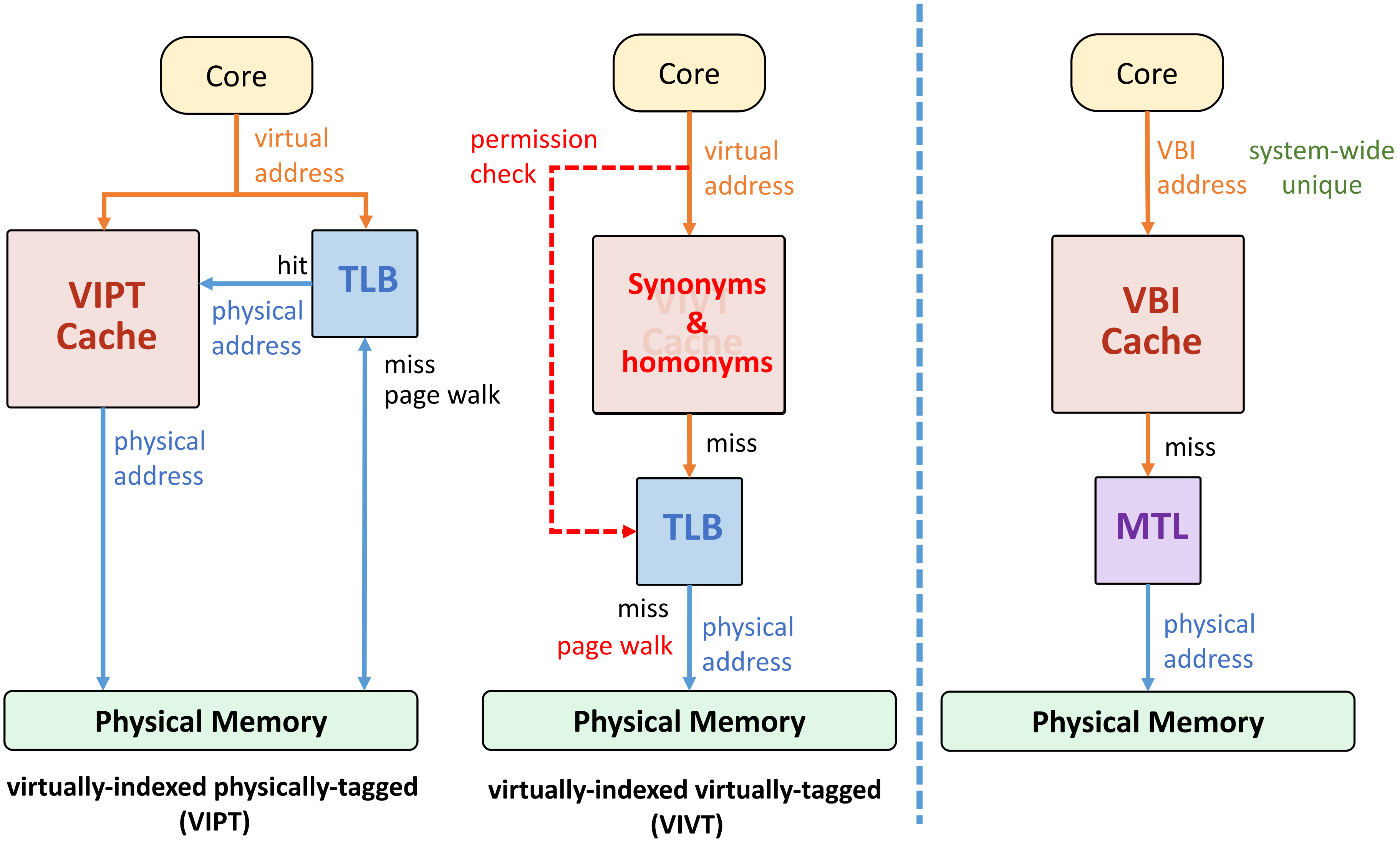
virtually-indexed physically-tagged (VIPT)

virtually-indexed virtually-tagged (VIVT)

In Conventional Virtual Memory

SAFARI

Inherently Virtual Caches

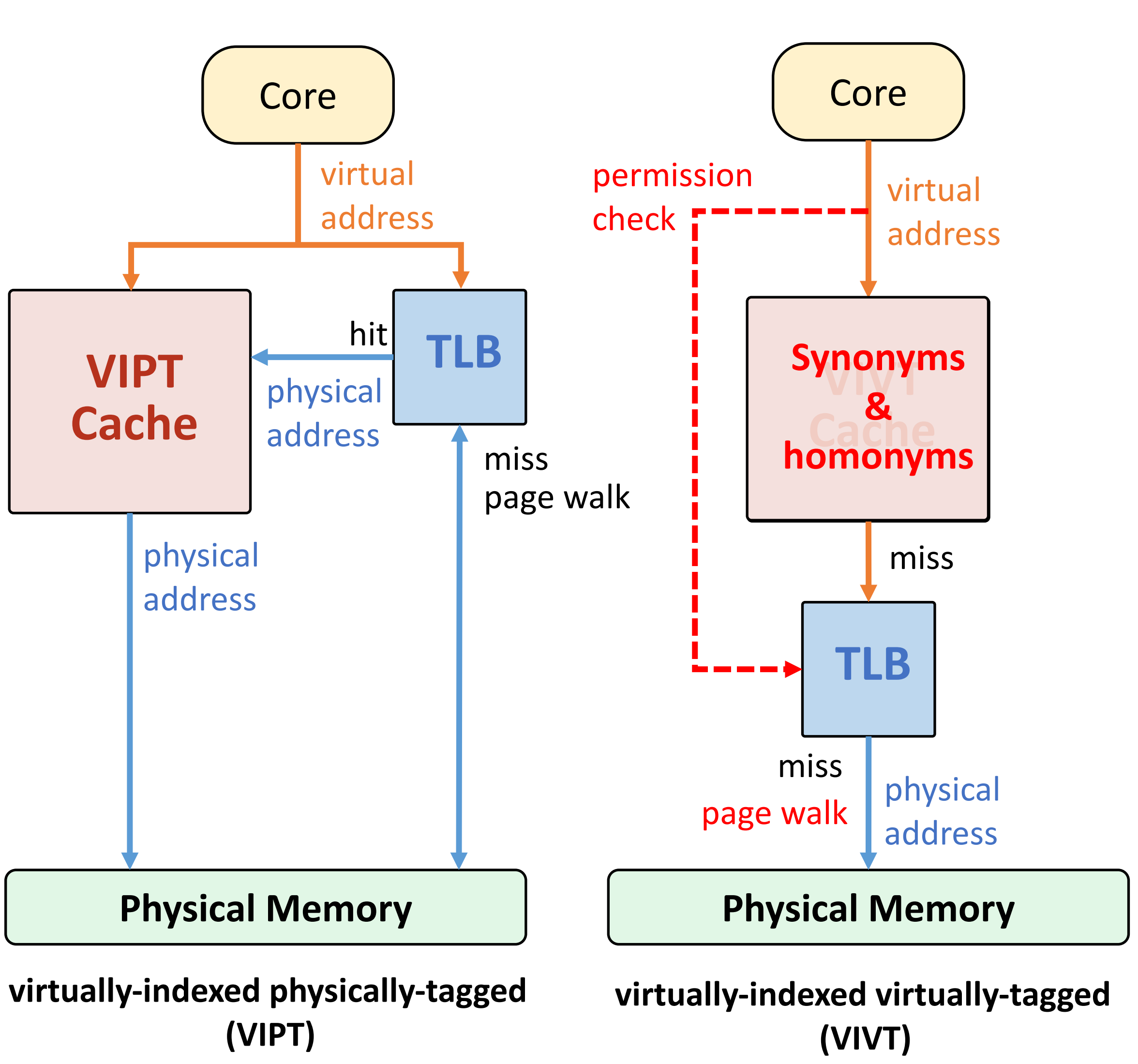


In Conventional Virtual Memory

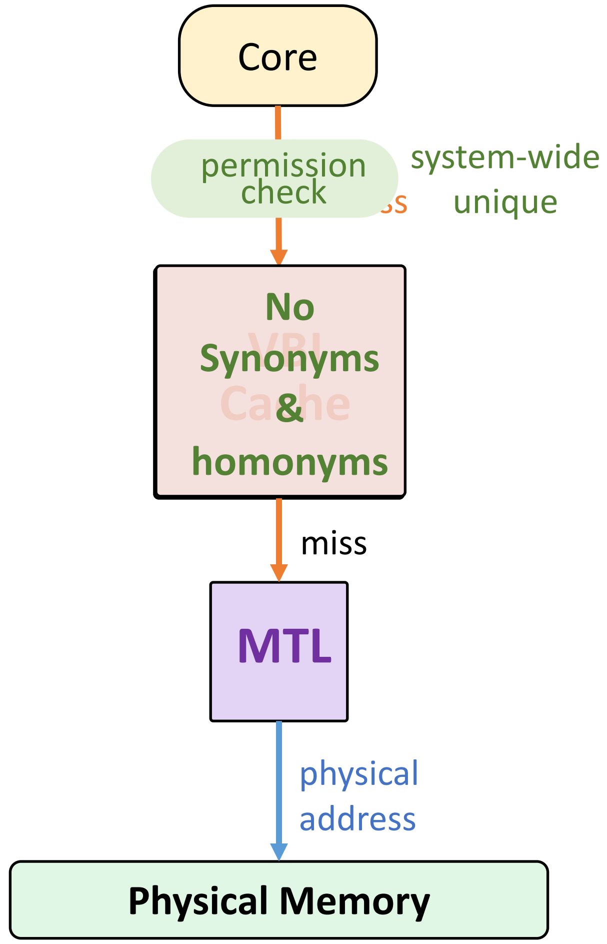
In VBI

SAFARI

Inherently Virtual Caches

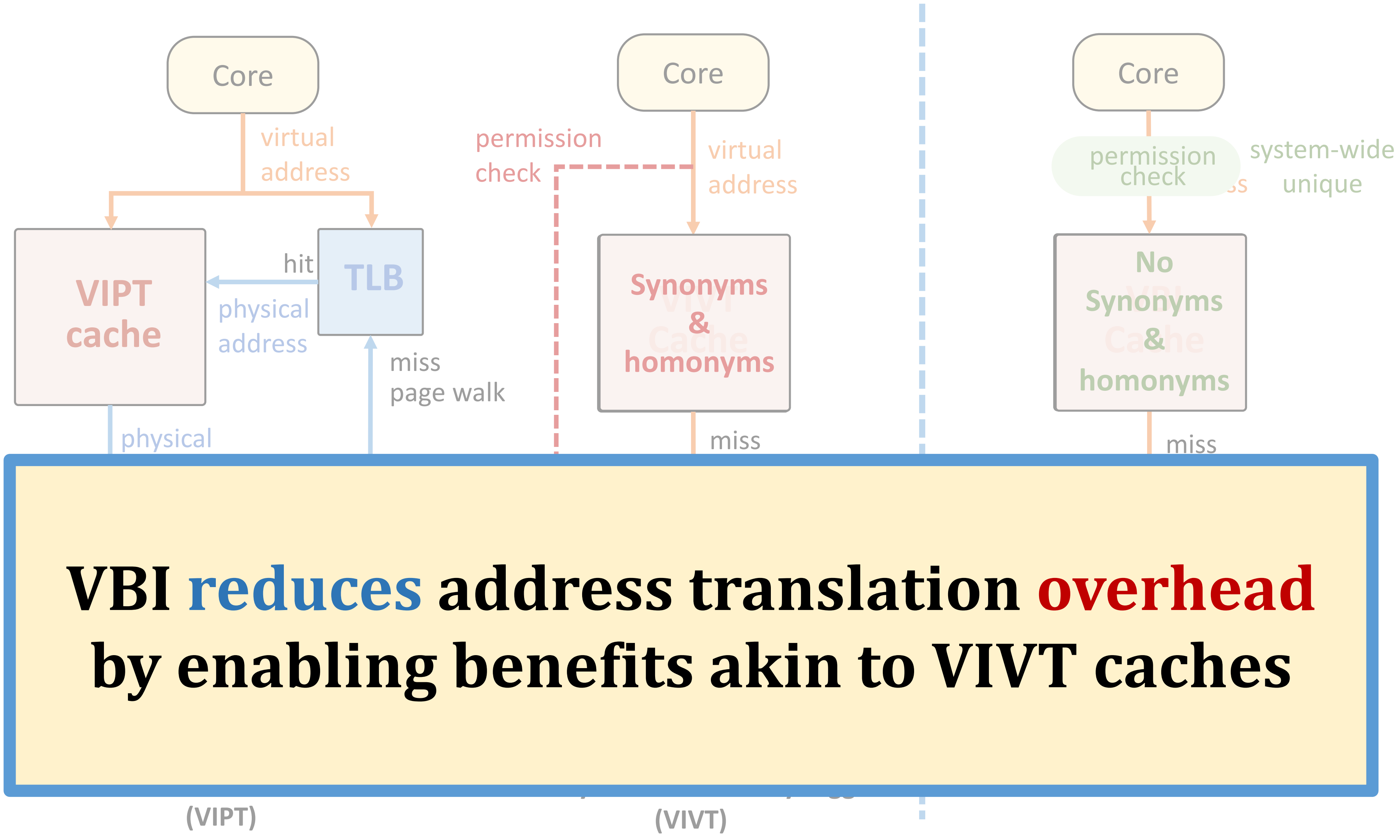


In Conventional Virtual Memory



In VBI

Inherently Virtual Caches



VBI **reduces** address translation **overhead** by enabling benefits akin to VIVT caches

In Conventional Virtual Memory

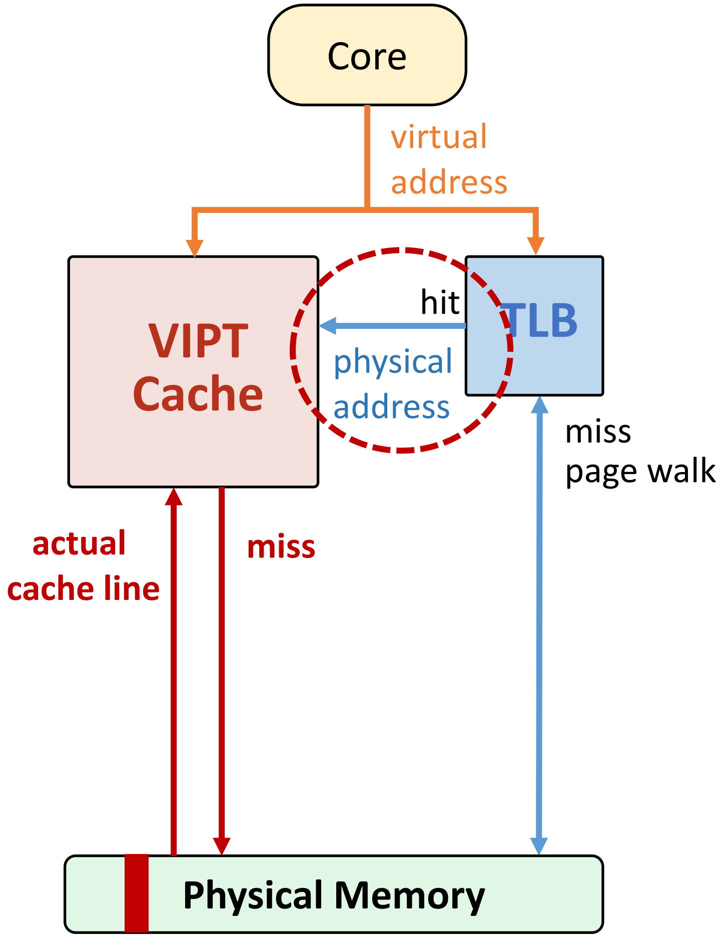
In VBI

SAFARI

Example Optimizations

- Inherently virtual caches
- Delayed physical memory allocation
- Eliminating 2D page walks in virtual machines

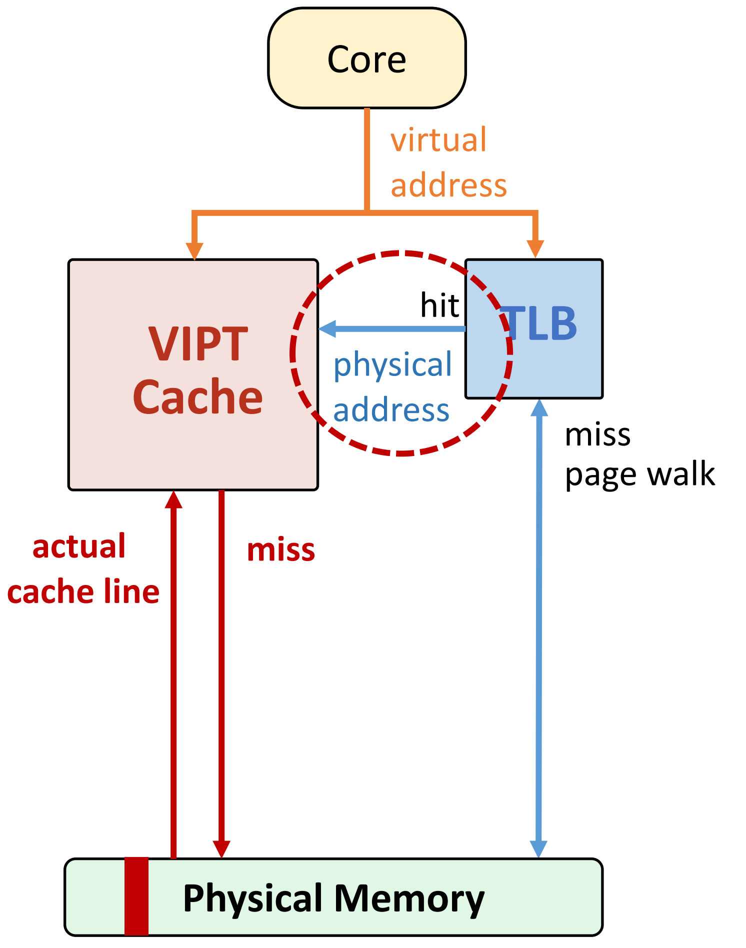
Delayed Physical Memory Allocation



virtually-indexed physically-tagged (VIPT)

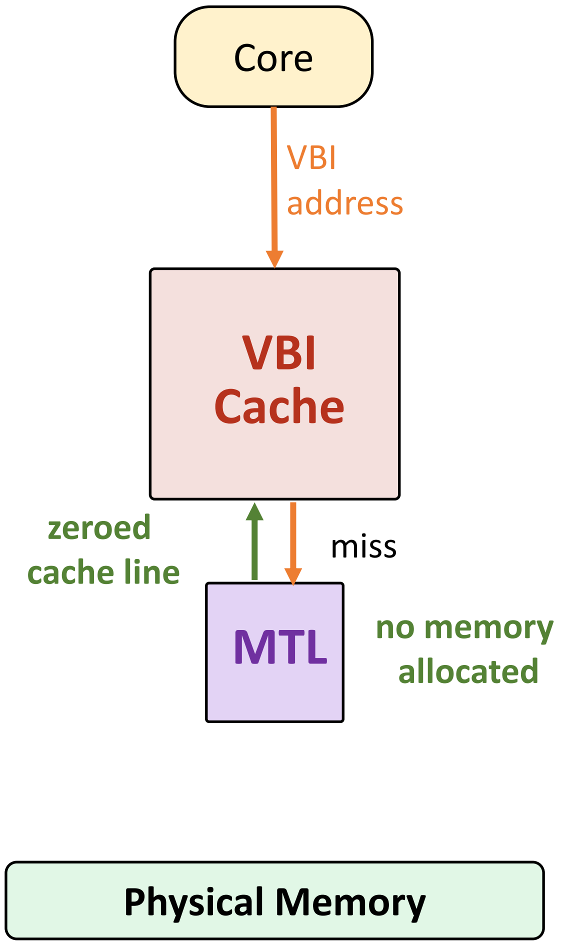
In Conventional Virtual Memory

Delayed Physical Memory Allocation



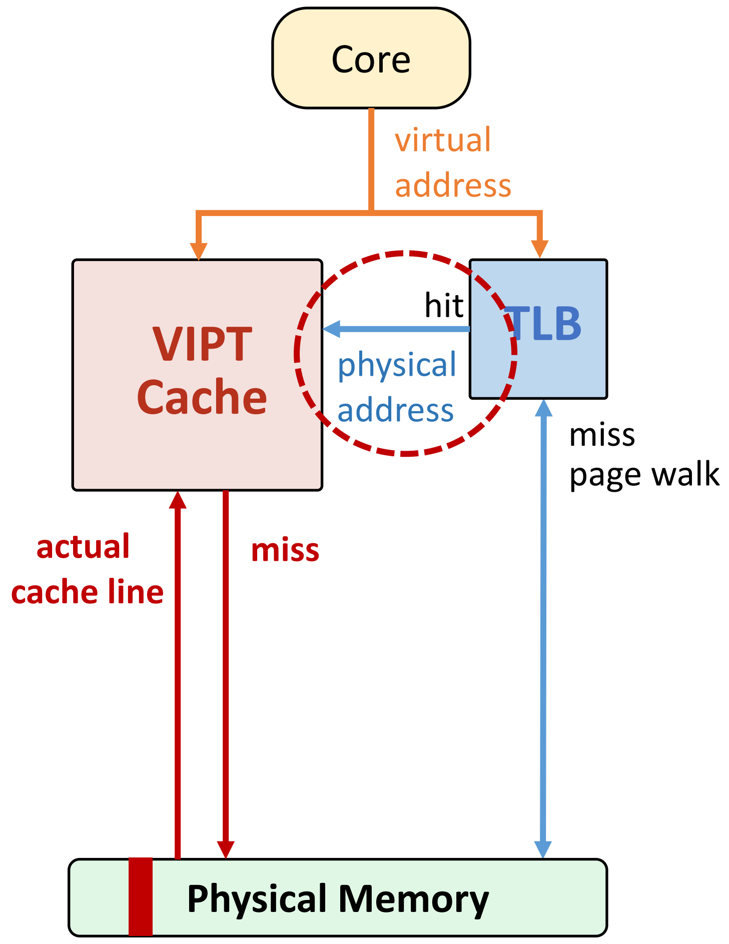
virtually-indexed physically-tagged (VIPT)

In Conventional Virtual Memory



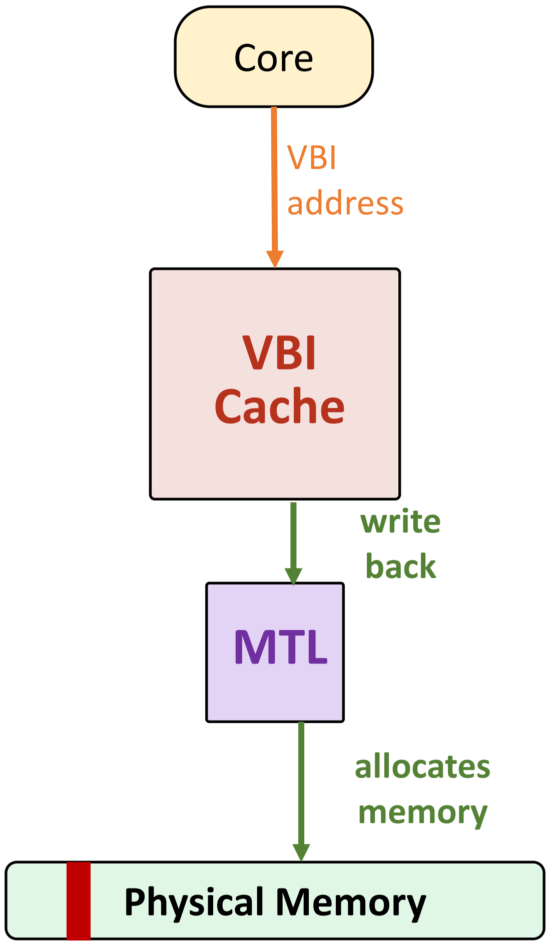
In VBI

Delayed Physical Memory Allocation



virtually-indexed physically-tagged (VIPT)

In Conventional Virtual Memory



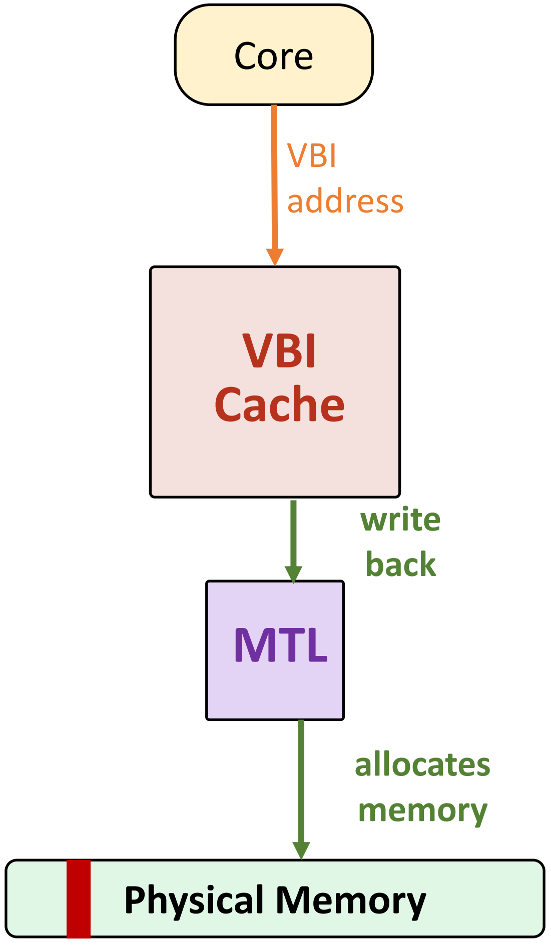
In VBI

Delayed Physical Memory Allocation

- No address translation
  for accesses to regions with
  no allocation

- No memory accesses
  to regions with no allocation yet

- No memory allocation
  for VBs that never leave the cache
  during their lifetime



In VBI

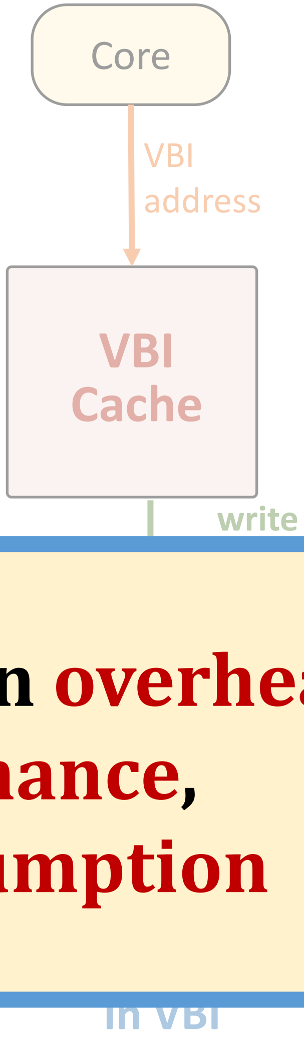
SAFARI

Delayed Physical Memory Allocation

- No address translation for accesses to regions with no allocation

- No memory access to regions with no allocation yet

-



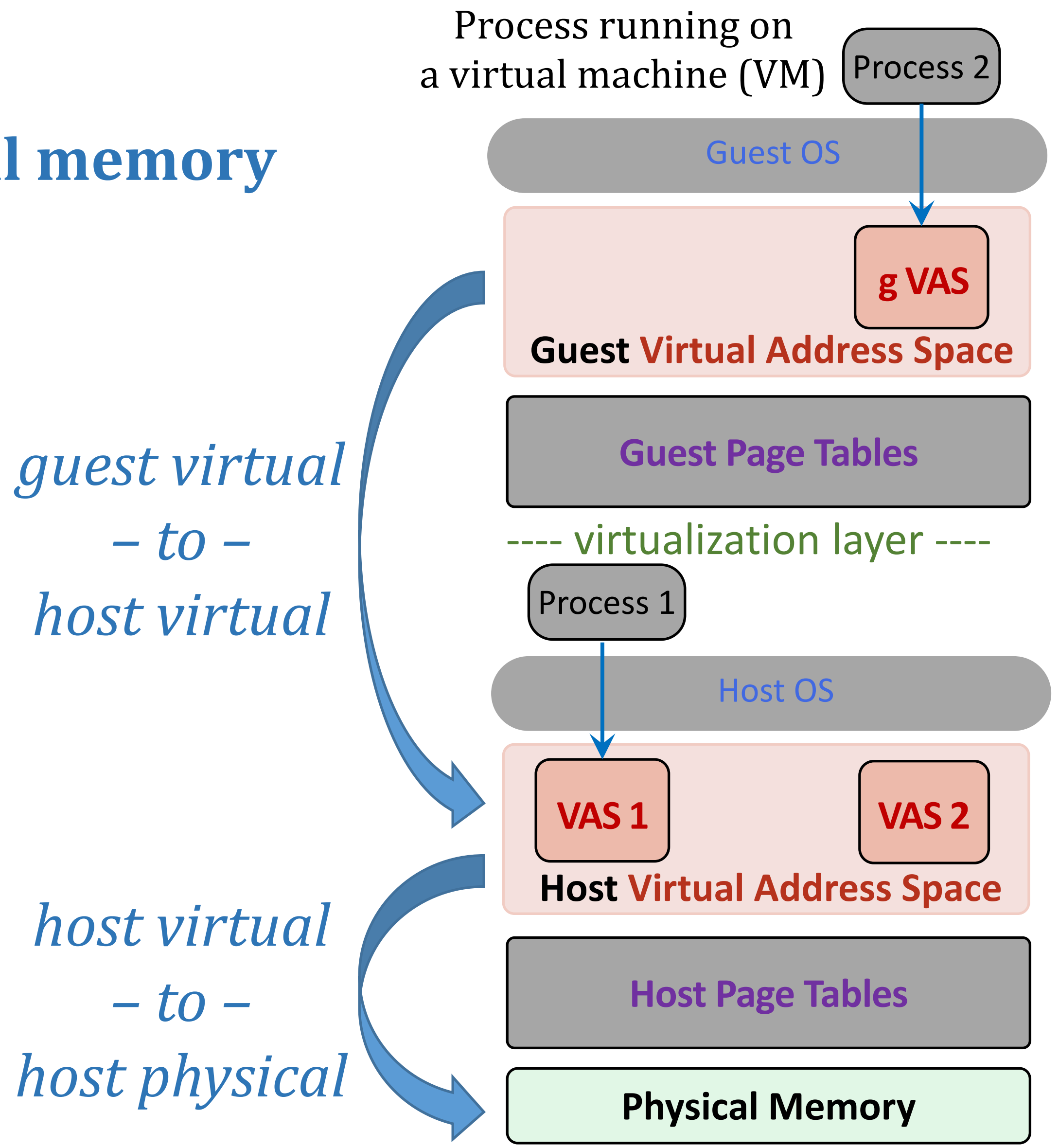
VBI reduces address translation overhead, improves overall performance, and reduces memory consumption

Example Optimizations

- Inherently virtual caches
- Inherently virtual caches
- Eliminating 2D page walks in virtual machines

Eliminating 2D Page Walks in Virtual Machines

Conventional virtual memory



Process running on a virtual machine (VM)
Process 2
Guest OS
g VAS
Guest Virtual Address Space
Guest Page Tables
---- virtualization layer ----
Process 1
Host OS
VAS 1
VAS 2
Host Virtual Address Space
Host Page Tables
Physical Memory
guest virtual – to – host virtual
host virtual – to – host physical

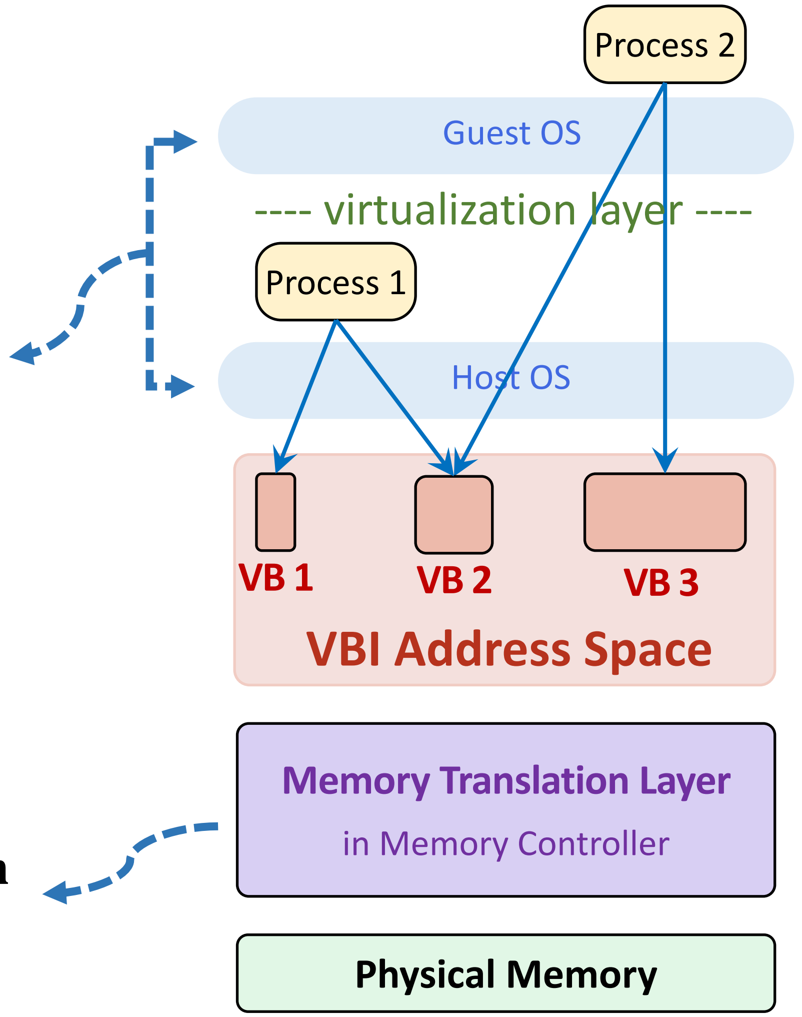
SAFARI

Eliminating 2D Page Walks in Virtual Machines

VBI

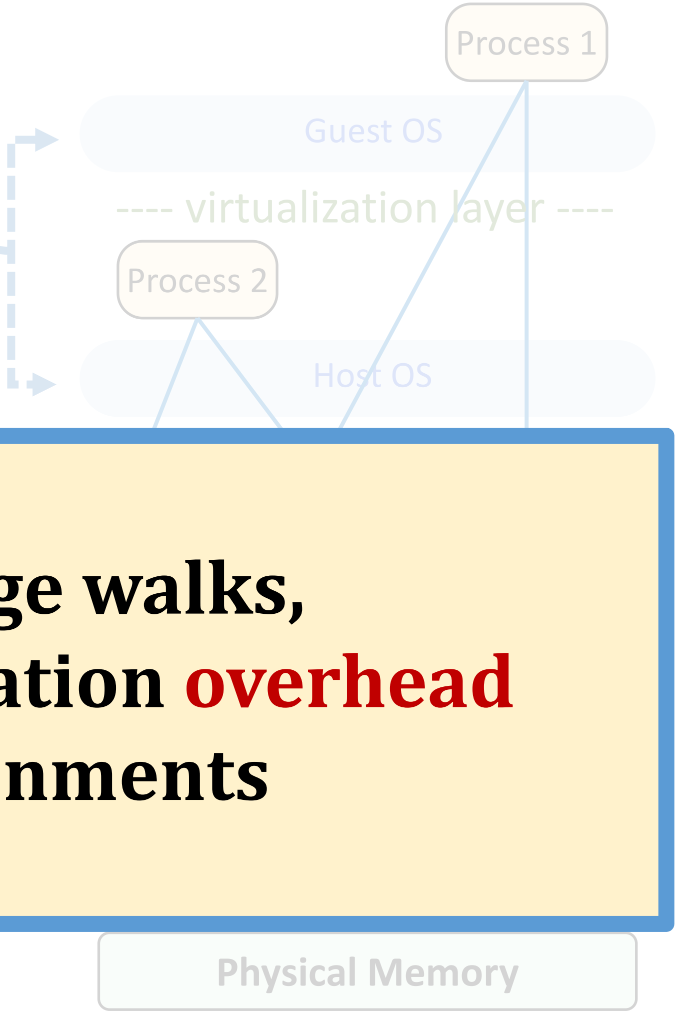
Guest OS and host OS interact once to attach Process 1 to its VBs

MTL is the only component in the system that manages address mapping



Eliminating 2D Page Walks in Virtual Machines

Guest OS and host OS interact once to attach Process 1 to its VBs



**By eliminating 2D page walks,
VBI reduces address translation overhead
in virtualized environments**

SAFARI

Outline

Methodology

- **Simulator:** heavily-modified version of Ramulator
  - Models virtual memory components (e.g., TLBs, page tables)
  - Available at https://github.com/CMU-SAFARI/Ramulator-VBI
- **Workloads:** SPECspeed 2017, SPEC CPU 2006, TailBench, Graph 500
- **System parameters:**
  - Core: 4-wide issue, OOO, 128-entry ROB
  - L1 Cache: 32 KB, 8-way associative, 4 cycles
  - L2 Cache: 256 KB, 8-way associative, 8 cycles
  - L3 Cache: 8 MB (2 MB per-core), 16-way associative, 31 cycles
  - L1 DTLB: - 4 KB pages: 64-entry, fully associative
    - 2 MB pages: 32-entry, fully associative
  - L2 DTLB: 4 KB and 2 MB pages: 512-entry, 4-way associative
  - Page Walk Cache: 32-entry, fully associative
  - DRAM: DDR3-1600, 1 channel, 1 rank/channel, 8 banks/rank
  - PCM: PCM-800, 1 channel, 1 rank/channel, 8 banks/rank

Outline

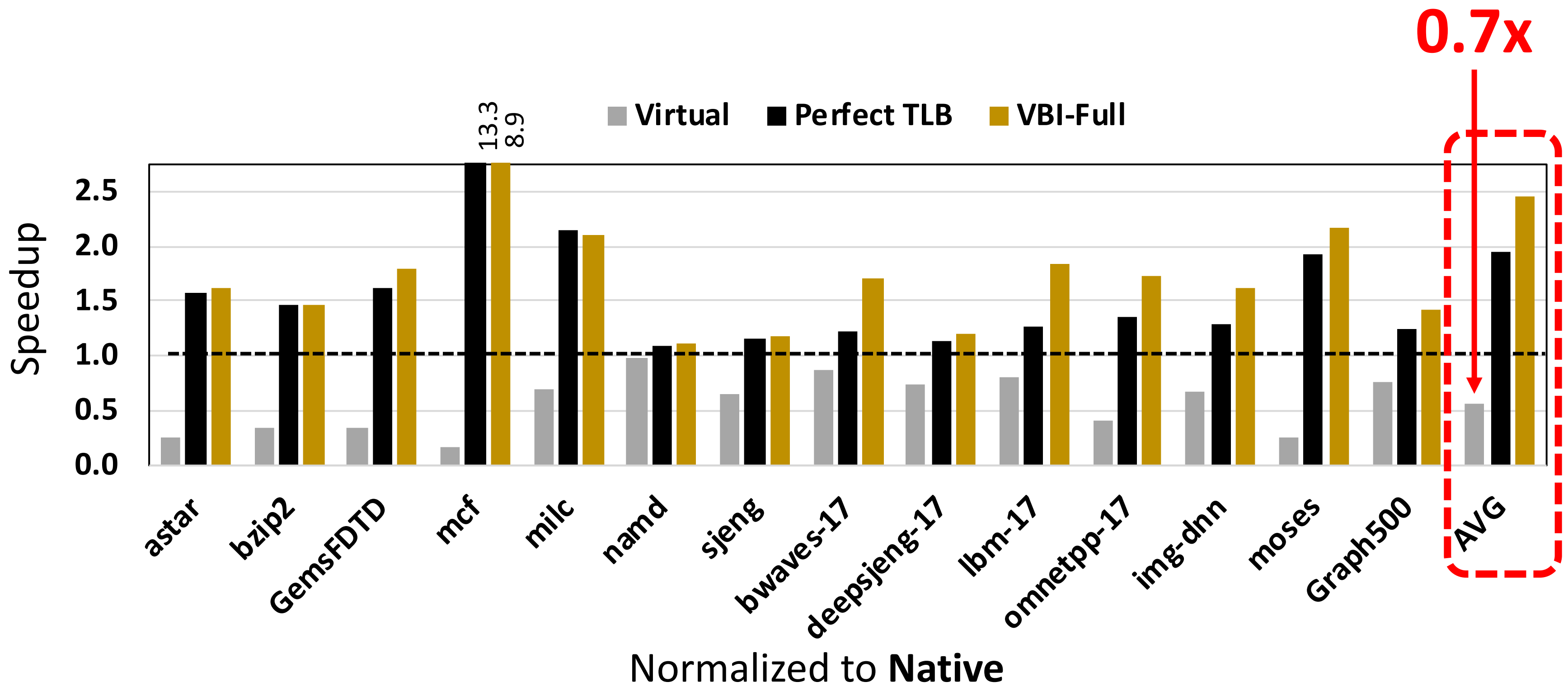
Use Case 1: Address Translation

- The impact of VBI on reducing the address translation overhead in both native execution and virtual machines

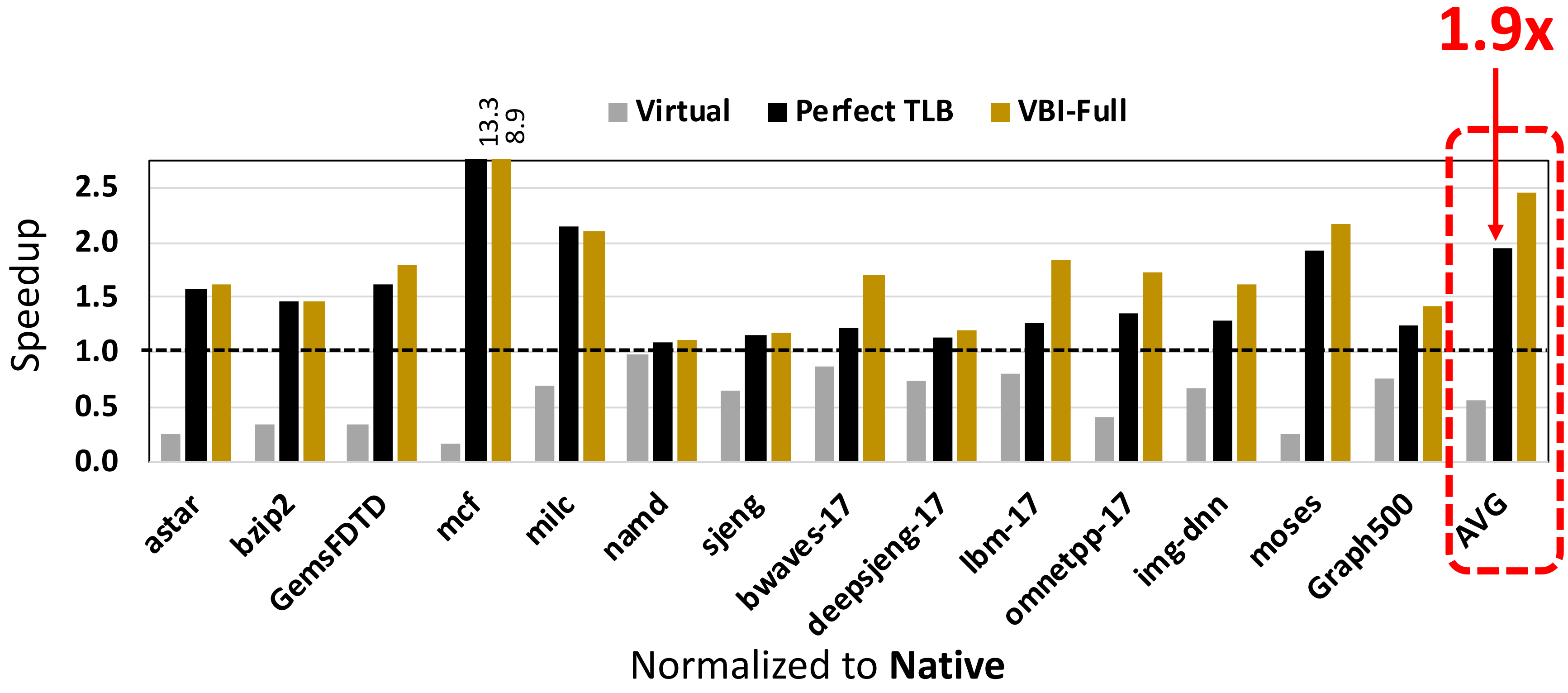
- **Evaluated systems:**
  - **Three baselines:**
    - **Native:** applications run natively on an x86-64 system
    - **Virtual:** applications run inside a virtual machine (accelerated using 2D page walk cache [Bhargava+, ASPLOS'08])
    - **Perfect TLB:** an unrealistic version of Native with no translation overhead
  - **One VBI configuration:**
    - **VBI-Full:** VBI with all the optimizations that it enables

- See our paper for results on more system configurations

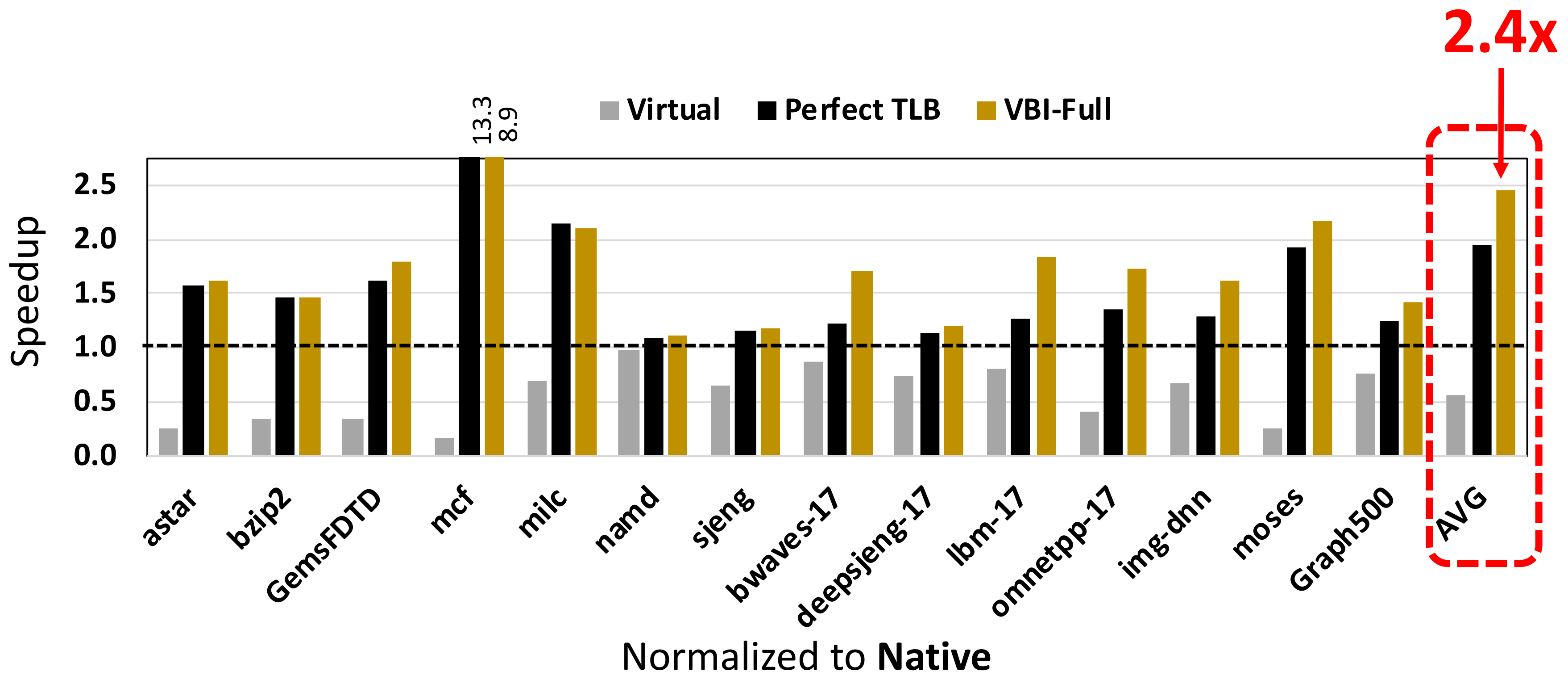
Use Case 1: Address Translation



Use Case 1: Address Translation

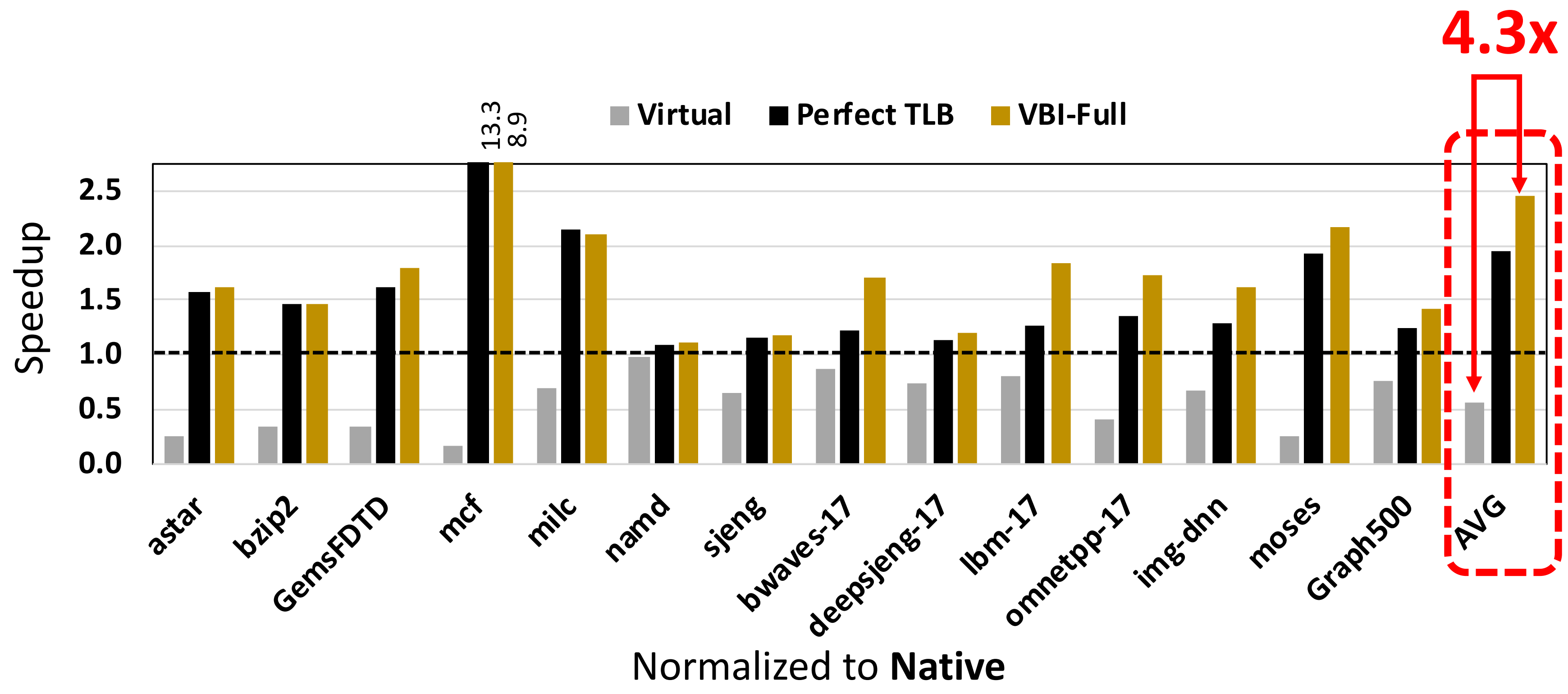


Use Case 1: Address Translation

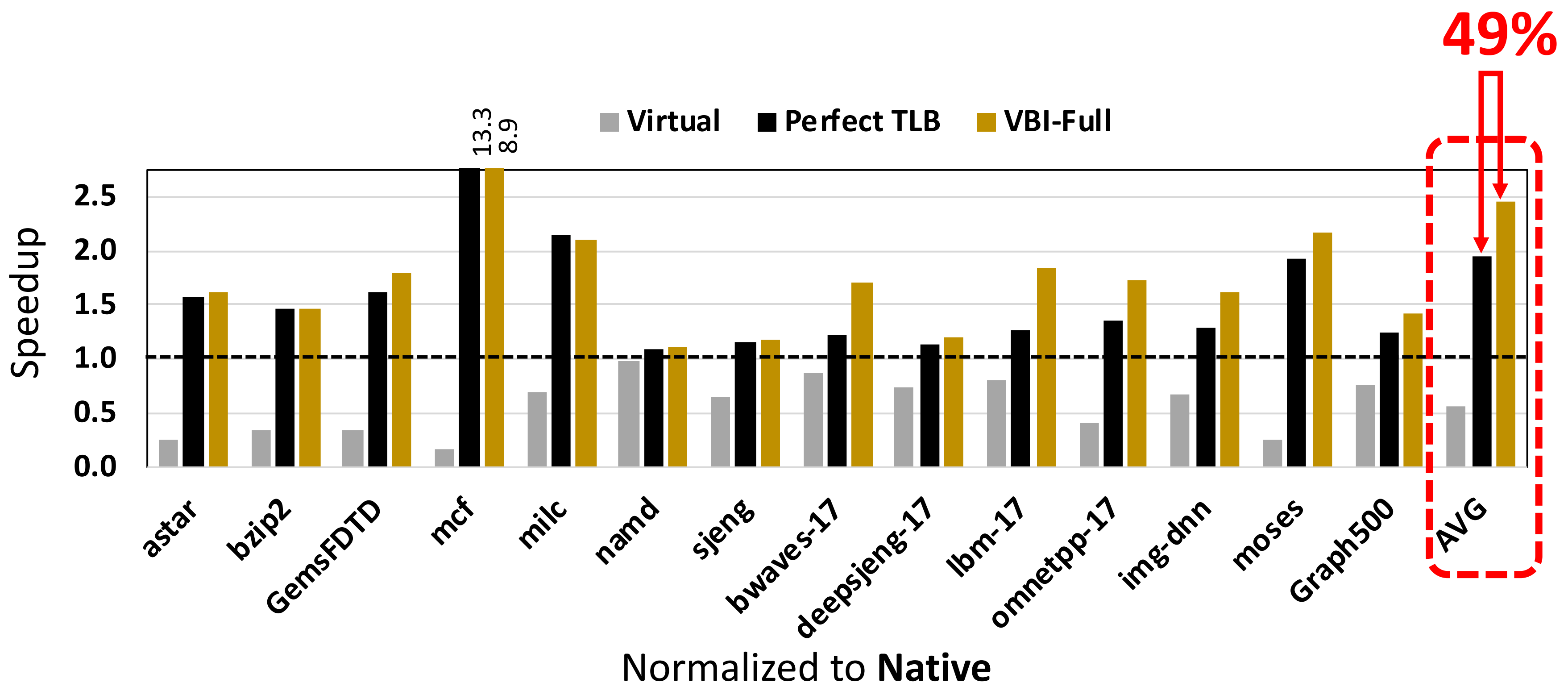


SAFARI

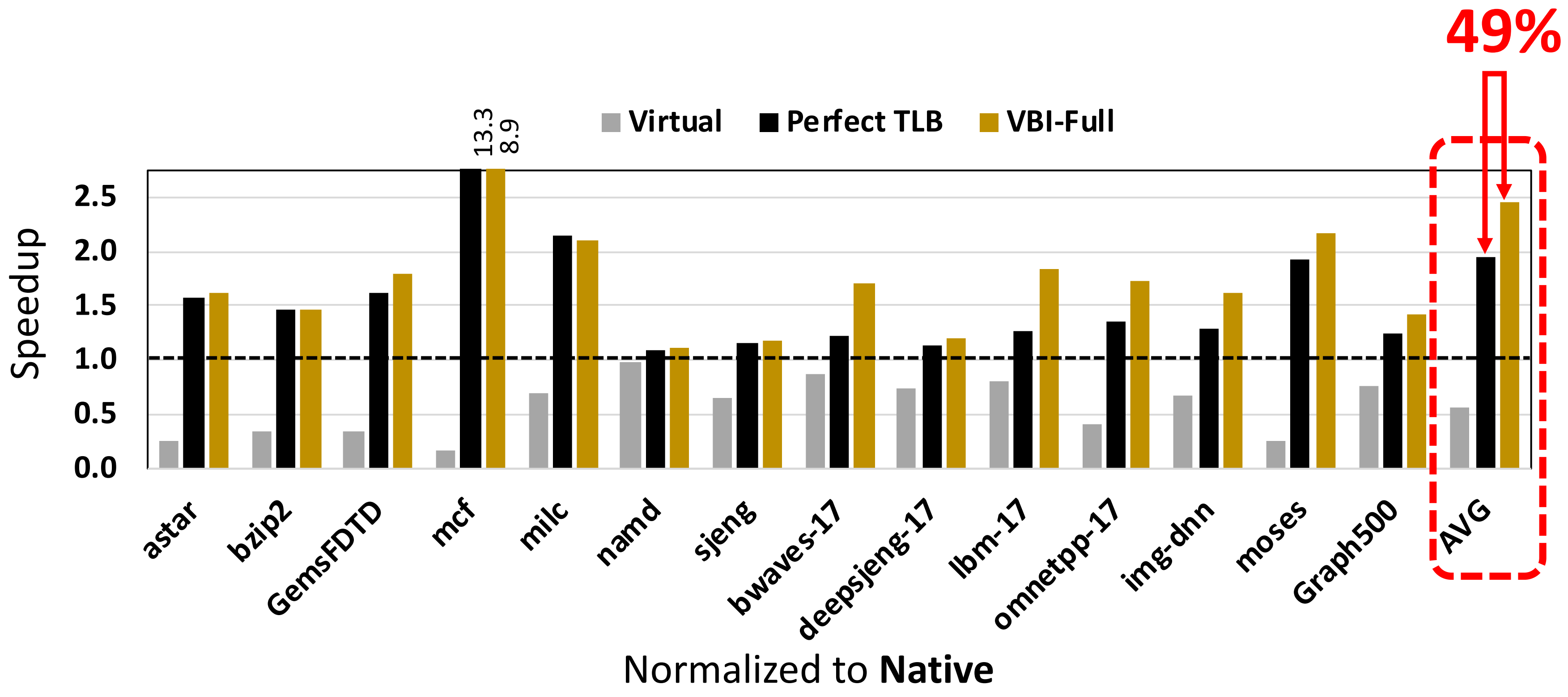
Use Case 1: Address Translation



Use Case 1: Address Translation



Use Case 1: Address Translation

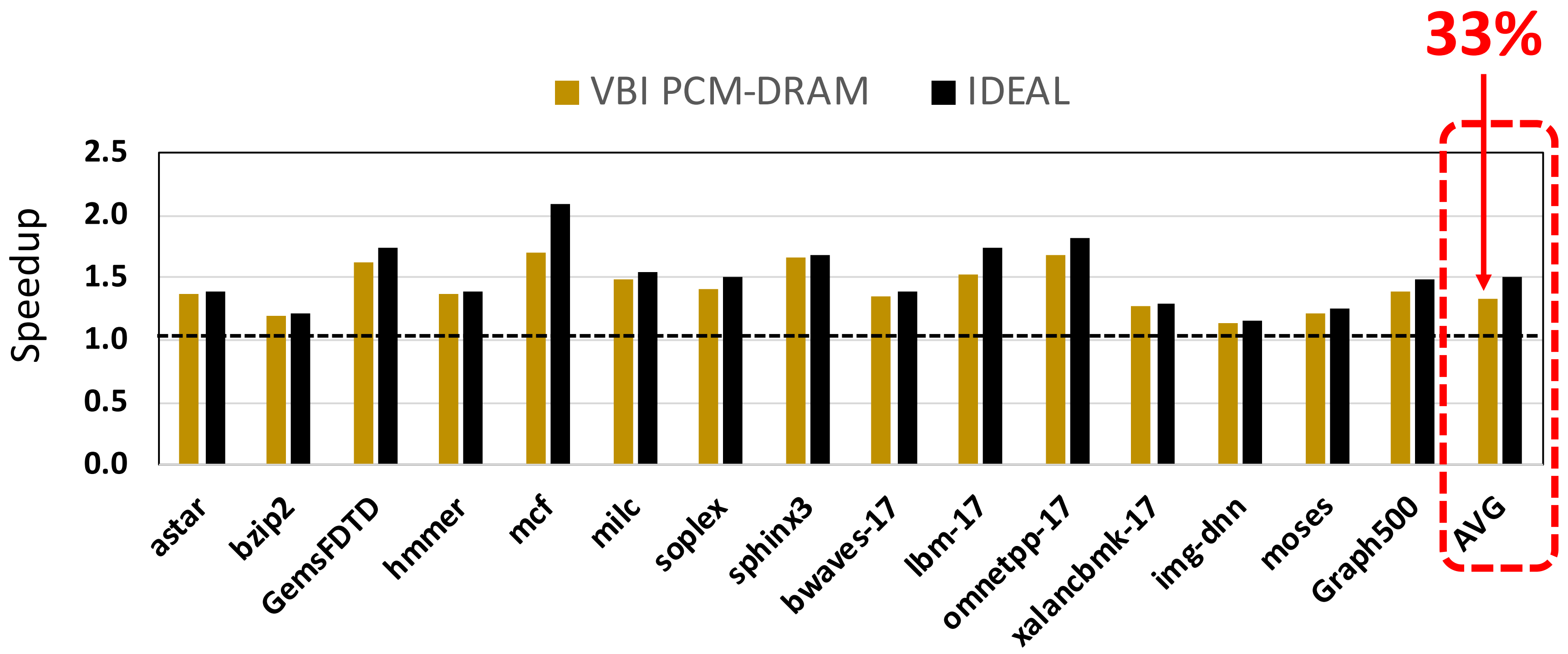


VBI significantly improves performance in both native execution and virtual machines

Use Case 2: Memory Heterogeneity

- The benefits of VBI in harnessing the full potential of heterogeneous memory architectures
  - Hybrid PCM–DRAM memory architecture
- **Evaluated systems:**
  - **Two baselines:**
    - **Hotness-Unaware PCM–DRAM:** unaware of the data hotness
    - **IDEAL:** always maps frequently-accessed data to DRAM
  - **One VBI configuration:**
    - **VBI PCM–DRAM:** VBI maps and migrates frequently-accessed VBs to the DRAM

Use Case 2: Memory Heterogeneity

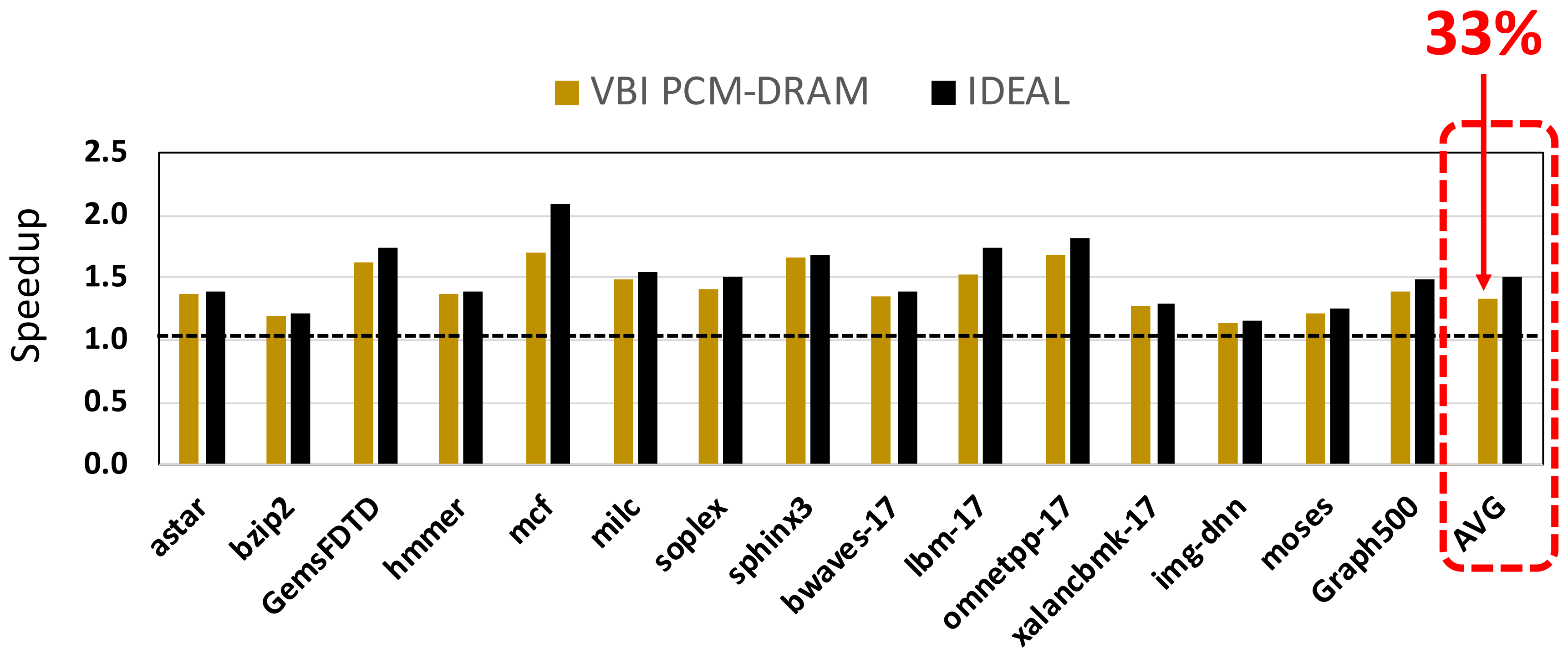


Normalized to **Hotness-Unaware PCM–DRAM**

More in our paper:

- Similar performance improvement for Tiered-Latency-DRAM [Lee+, HPCA'13]

Use Case 2: Memory Heterogeneity



Normalized to **Hotness-Unaware PCM–DRAM**

VBI enables efficient data mapping and data migration for heterogeneous memory systems

SAFARI

Outline

Summary

- **Virtual Block Interface (VBI):** A new virtual memory framework
  - Addresses the challenges in adapting conventional virtual memory to increasingly diverse system configurations and workloads
- **Key Idea:** Delegate physical memory management to dedicated hardware in the memory controller
- **Benefits:** Not easily attainable in conventional virtual memory (e.g., inherently virtual caches , delaying physical memory allocation, and avoiding 2D page walks in virtual machines)
- **Evaluation:**
  - VBI significantly improves performance in both native execution and virtual machines
  - Increases the effectiveness of managing heterogeneous memory architectures
- **Conclusion:** VBI is a promising new virtual memory framework
  - Can enable several important optimizations
  - Increases design flexibility for virtual memory
  - A new direction for future work in novel virtual memory frameworks

The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework

Nastaran Hajinazar Pratyush Patel Minesh Patel
Konstantinos Kanellopoulos Saugata Ghose
Rachata Ausavarungnirun Geraldo F. Oliveira
Jonathan Appavoo Vivek Seshadri Onur Mutlu

SAFARI ETH Zürich SFU SIMON FRASER UNIVERSITY UNIVERSITY of WASHINGTON
Carnegie Mellon KMUTNB Microsoft BOSTON UNIVERSITY