

Data-Centric and Data-Aware Frameworks for Fundamentally Efficient Data Handling in Modern Computing Systems

Nastaran Hajinazar

PhD Thesis Defense Presentation - 4 June 2021

Committee: Prof. Onur Mutlu (Co-Senior Supervisor)
Prof. Arrvindh Shriraman (Co-Senior Supervisor)
Prof. Saugata Ghose
Dr. Vivek Seshadri (Microsoft)
Prof. Alaa Alameldeen
Prof. Myoungsoo Jung

SAFARI



SIMON FRASER
UNIVERSITY

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

Our Approach

SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

Our Approach

SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

Data Is Increasing

- There is an **explosive growth** in the amount of data processed in modern computing systems
- Important applications and workloads of a wide range of domains are all **data intensive**
- **Efficient** and **fast** accessing, moving, and processing of large amounts of data is **critical**

Question:

Do We Handle Data *Well*?!

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

Our Approach

SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

Our Approach

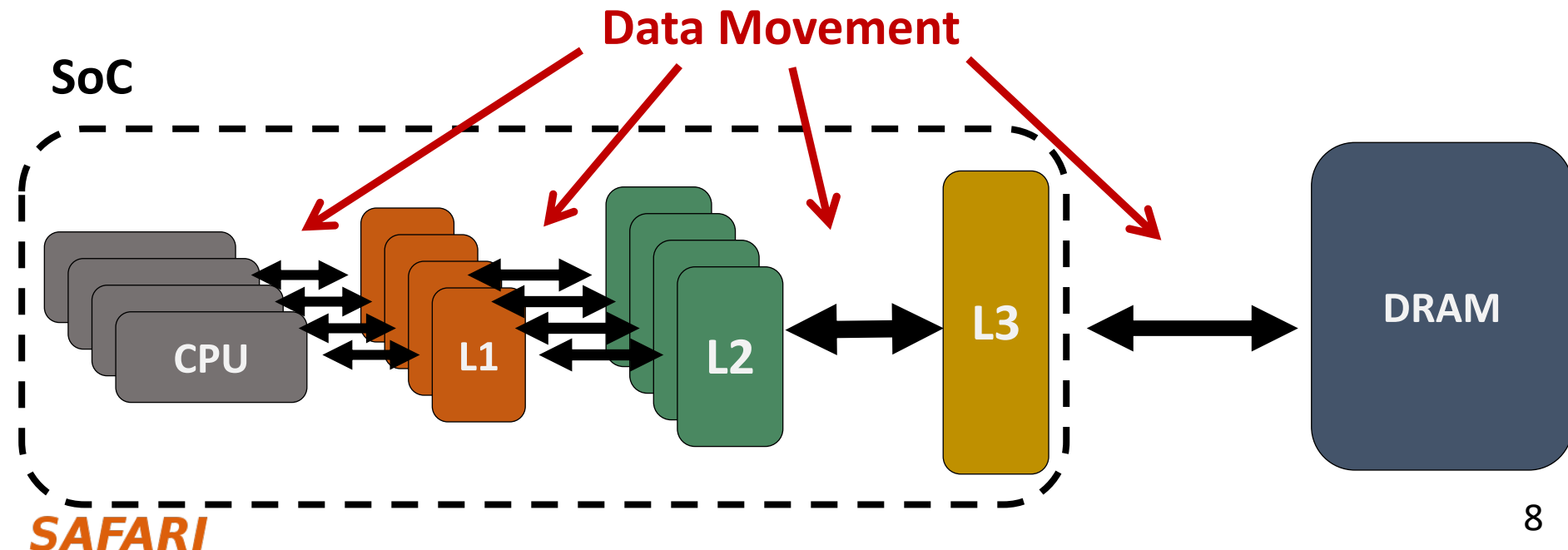
SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

Computing Systems Today

- Are overwhelmingly **processor-centric**
 - Computation is performed **only in the processor**
 - Every piece of data needs to be **transferred** to the processor enable the computation.



Processor-Centric Design Implications

- High data movement volume
 - Energy overhead
 - Performance overhead

**Computation is bottlenecked by
data movement**

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

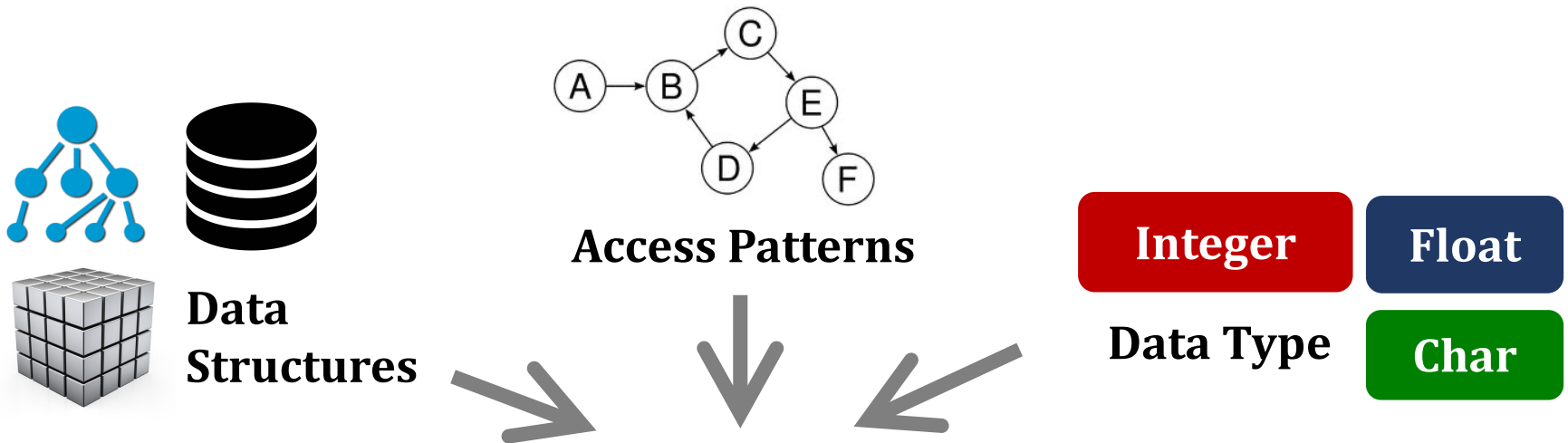
Our Approach

SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

Higher-Level Information Is Not Visible to HW



Software

Hardware

**Virtual
Memory**

100011111...

Instructions

101010011...

Memory Addresses

**Data-Oblivious
Policies
SAFARI**

**Data
Placement**

**Cache
Management**

**Address
Translation**

...

Data-Oblivious Policies Implications

- Challenging and often not very effective
 - Ineffective policies
 - Lost performance improvement opportunities

The conventional virtual memory frameworks are **not efficient** moving forward

The Problem

Question:
Do We Handle Data *Well*?
Answer:
No!

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

Our Approach

SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

Overview of Our Approach

Data and the efficient computation of data should be the ultimate priority of the system

- Data-Centric Architectures

- Enable computation with minimal data movement
- Compute where data resides

- Data-Aware Architectures

- Understand what they can do with and to each piece of data
- Make use of different properties of data to improve performance, efficiency, etc.

Thesis Statement

The **performance** and energy **efficiency** of computing systems can improve significantly when handling large amounts of data by employing **data-centric** and **data-aware** architectures that can

- Remove the overheads associated with data movement by processing data where it resides
- Efficiently adopt the diversity in today's system configurations and memory architectures
- Understand, convey, and exploit the characteristics of the data to make more intelligent memory management decisions

Contributions

- **SIMDRAM:** A **Data-Centric** Framework for Bit-Serial SIMD Processing using DRAM [ASPLOS 2021]
 - Efficiently implements **complex operations**
 - Flexibly supports **new desired operations**
 - Requires **minimal changes** to the DRAM architecture
- **The Virtual Block Interface:** A Flexible **Data-Aware** Alternative to the Conventional Virtual Memory Framework [ISCA 2020]
 - Understands, conveys, and exploits **data properties**
 - Efficiently supports **diverse system configurations**
 - Efficiently handles **large amounts of data**

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

Our Approach

SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

Processing-using-Memory: Prior Works

- DRAM and other memory technologies that are capable of performing **computation using memory**

Shortcomings:

- Support **only basic** operations (e.g., Boolean operations, addition)
 - Not widely applicable
- Support a **limited** set of operations
 - Lack the flexibility to support new operations
- Require **significant changes** to the DRAM
 - Costly (e.g., area, power)

Processing-using-Memory: Prior Works

- DRAM and other memory technologies that are capable of performing **computation using memory**

Shortcomings:

- Support **only basic** operations (e.g., Boolean operations, addition)

Need a framework that aids **general adoption of PuM, by:**

- Efficiently implementing **complex operations**
- Providing flexibility to support **new operations**

- Costly (e.g., area, power)

Goal

Goal: Design a PuM framework that

- Efficiently implements complex operations
- Provides the flexibility to support new desired operations
- Minimally changes the DRAM architecture

SIMDRAM: A Framework for Bit-Serial SIMD Processing using DRAM

*Nastaran Hajinazar^{1,2}

Nika Mansouri Ghiasi¹

*Geraldo F. Oliveira¹

Minesh Patel¹

Juan Gómez-Luna¹

Sven Gregorio¹

Mohammed Alser¹

Onur Mutlu¹

João Dinis Ferreira¹

Saugata Ghose³

¹ETH Zürich

²Simon Fraser University

³University of Illinois at Urbana–Champaign

Key Idea:

Provide the programming interface, the ISA, and the hardware support for:

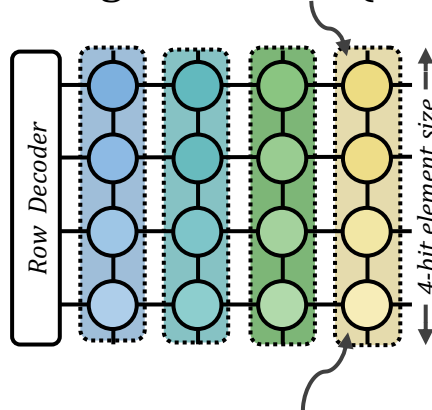
- **Efficiently** computing **complex** operations in DRAM
- Providing the ability to implement **arbitrary** operations as required
- Requiring **minimal** changes to DRAM architecture

SIMDRAM: PuM Substrate

- SIMDRAM framework is built around a DRAM substrate that enables two techniques:

(1) Vertical data layout

most significant bit (MSB)



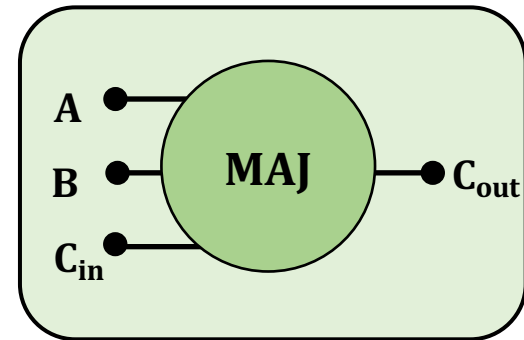
least significant bit (LSB)

Pros compared to the conventional **horizontal layout**:

- Implicit shift operation
- Massive parallelism

(2) Majority-based computation

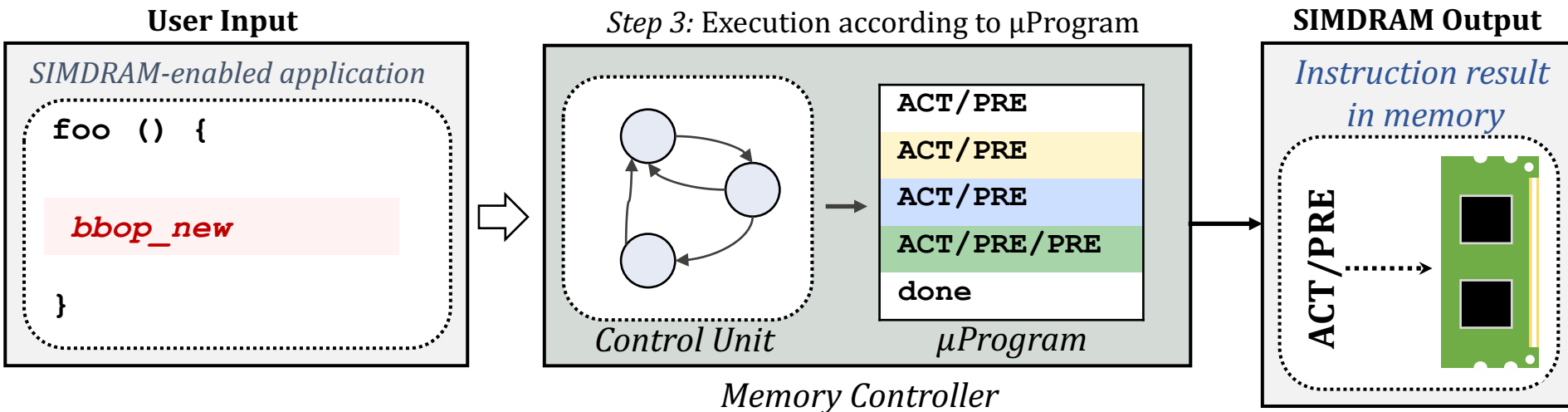
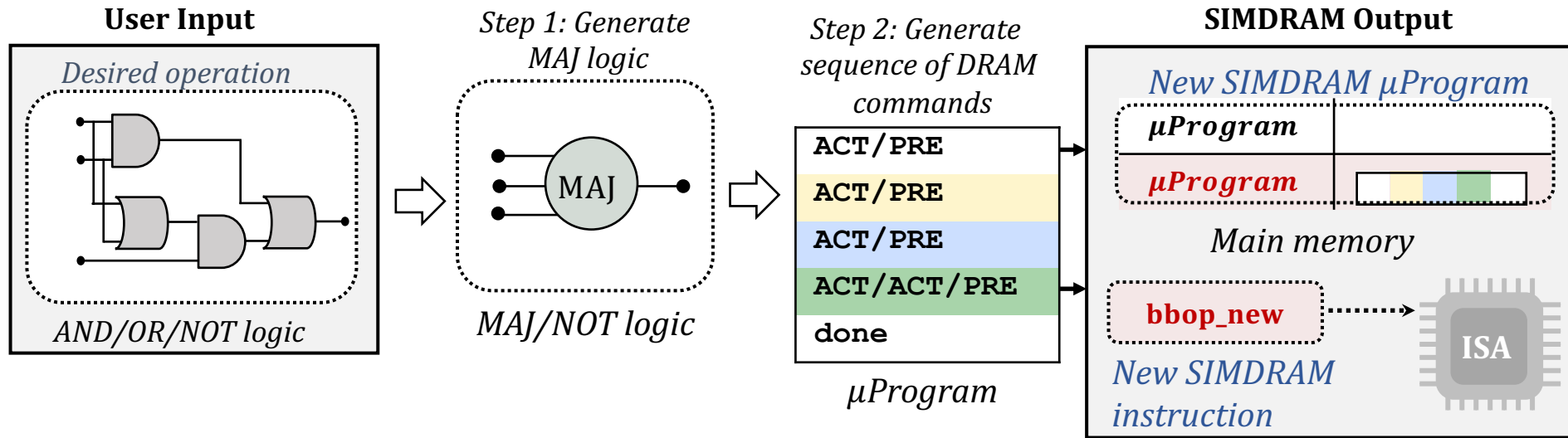
$$C_{out} = AB + AC_{in} + BC_{in}$$



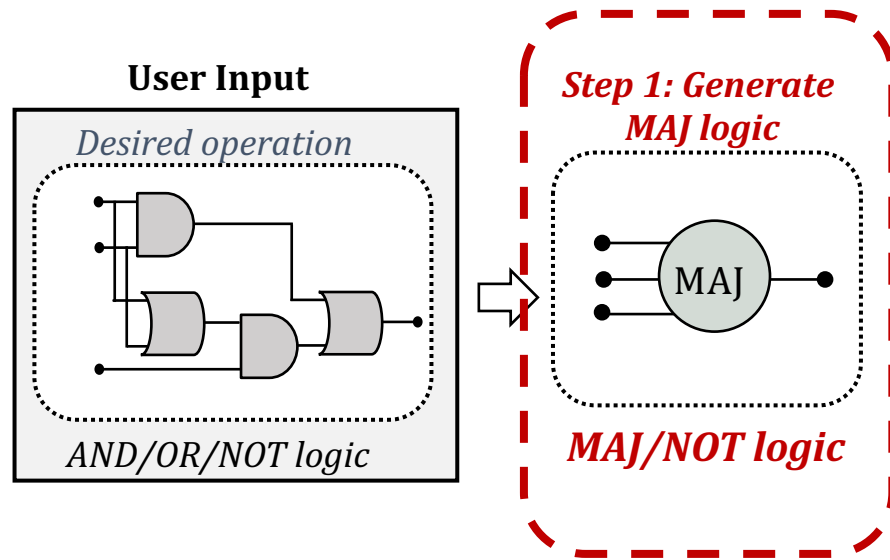
Pros compared to **AND/OR/NOT-based** computation:

- Higher performance
- Higher throughput
- Lower energy consumption

SIMDRAM Framework: Overview



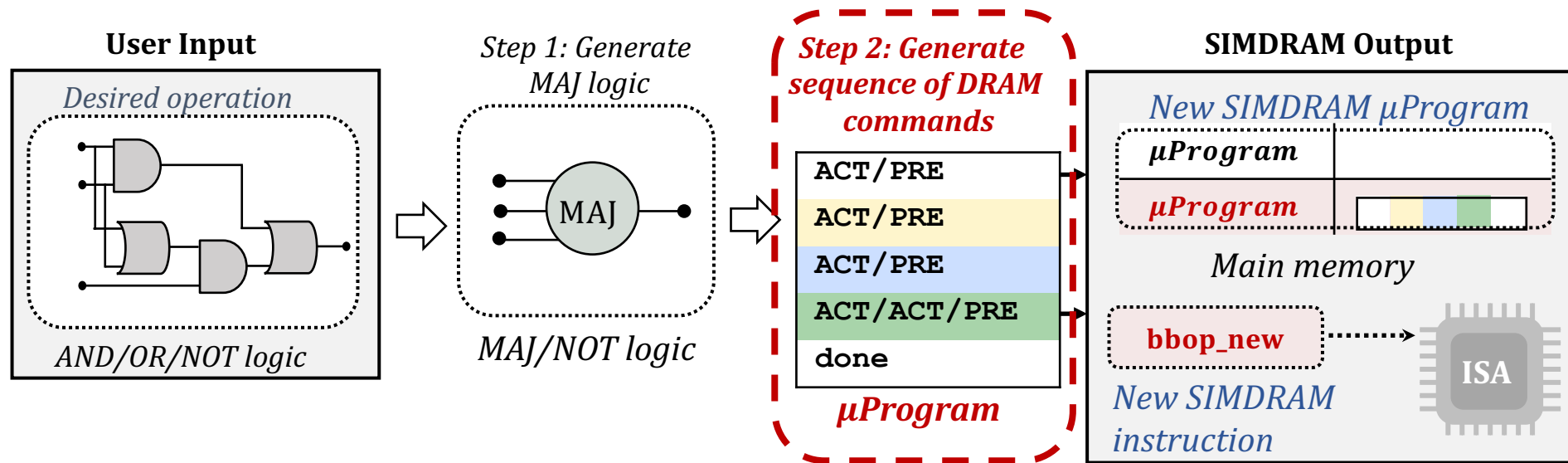
SIMDRAM Framework: Overview



Step 1:

- Builds an **efficient MAJ/NOT representation** of a given desired operation from its AND/OR/NOT-based implementation

SIMDRAM Framework: Overview



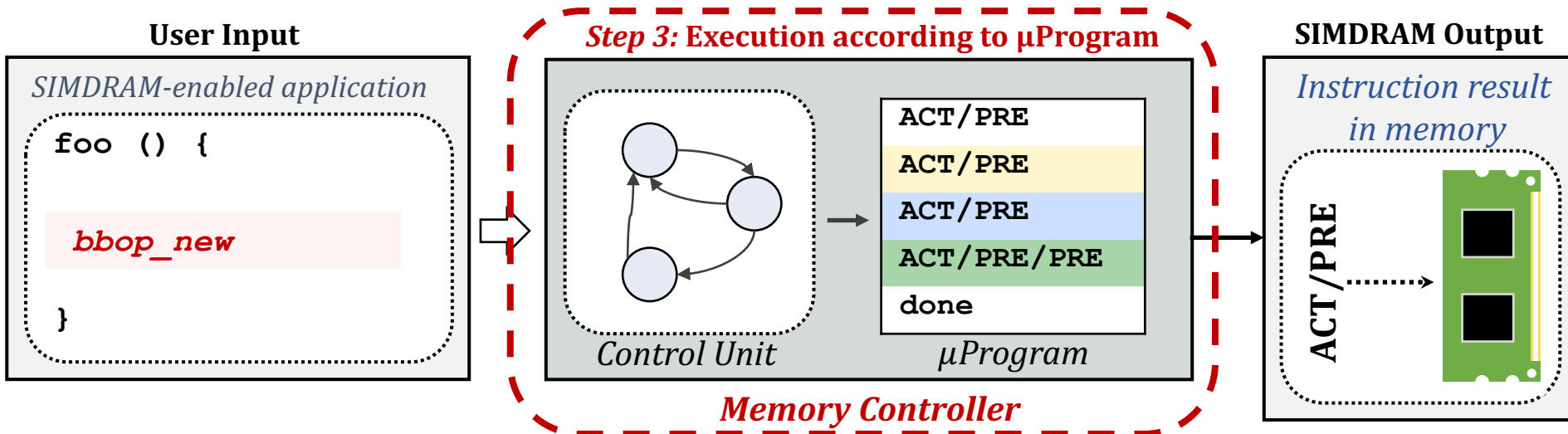
Step 2:

- **Allocates DRAM** rows to the operation's inputs and outputs
- Generates the **sequence of DRAM commands** (*μProgram*) to execute the desired operation

SIMDRAM Framework: Overview

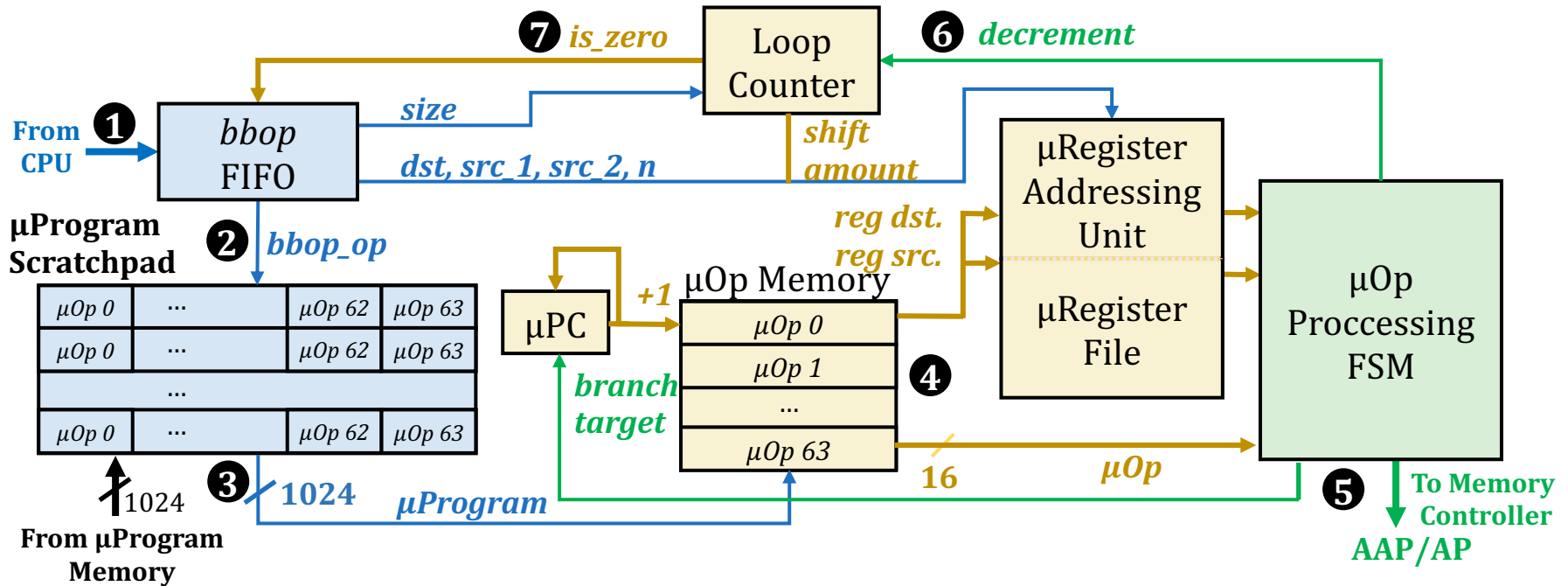
Step 3:

- Executes the μ Program to perform the operation
- Uses a **control unit** in the memory controller



More in the Thesis

- Detailed reference implementation and microarchitecture of the SIMD RAM control unit



System Integration

Efficiently transposing data

Programming interface

Handling page faults, address translation,
coherence, and interrupts

Handling limited subarray size

Security implications

Limitations of our framework

System Integration

Efficiently transposing data

Programming interface

Handling page faults, address translation,
coherence, and interrupts

Handling limited subarray size

Security implications

Limitations of our framework

Transposing Data

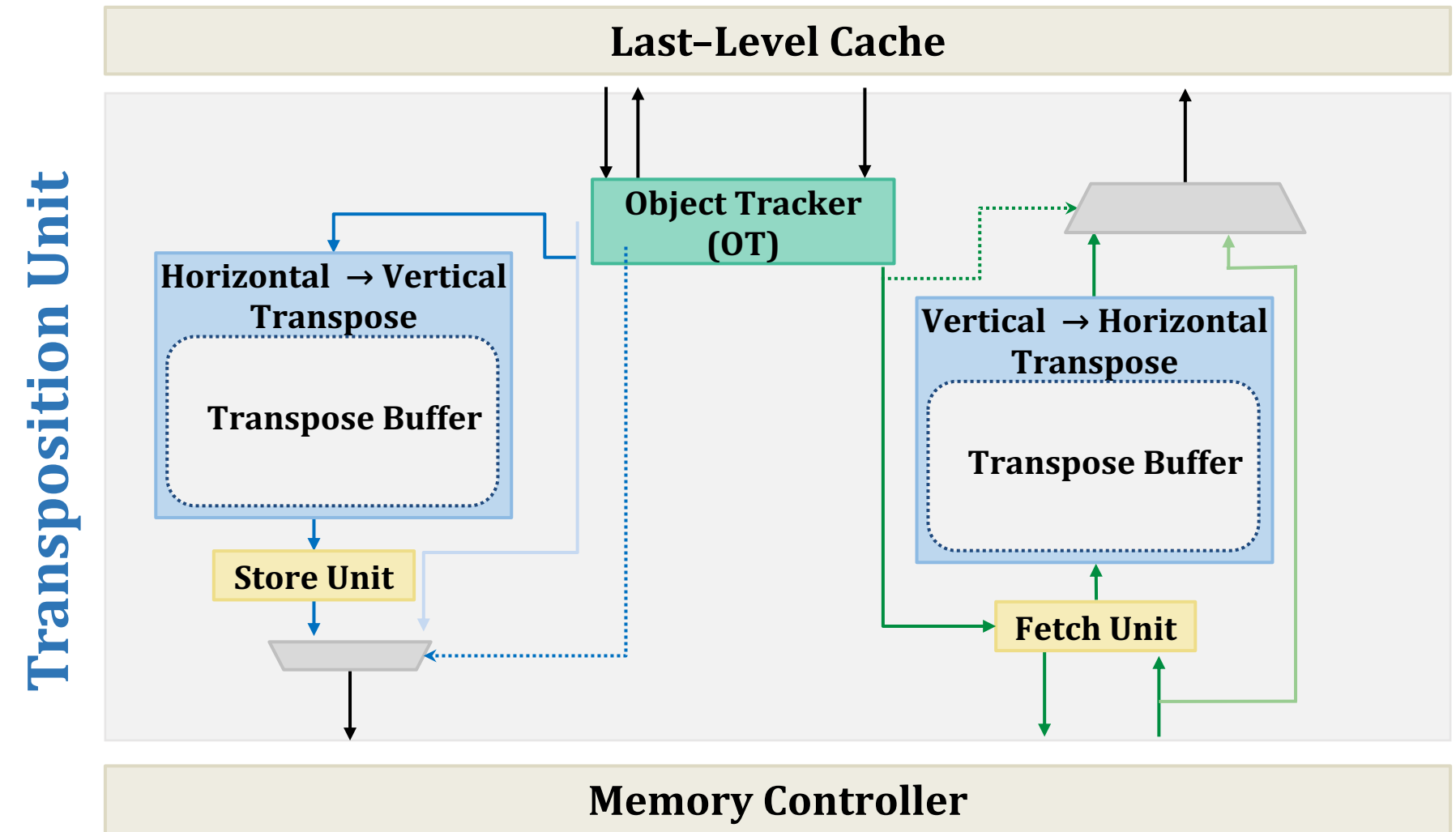
- SIMD RAM operates on **vertically-laid-out** data
- Other system components expect data to be laid out **horizontally**



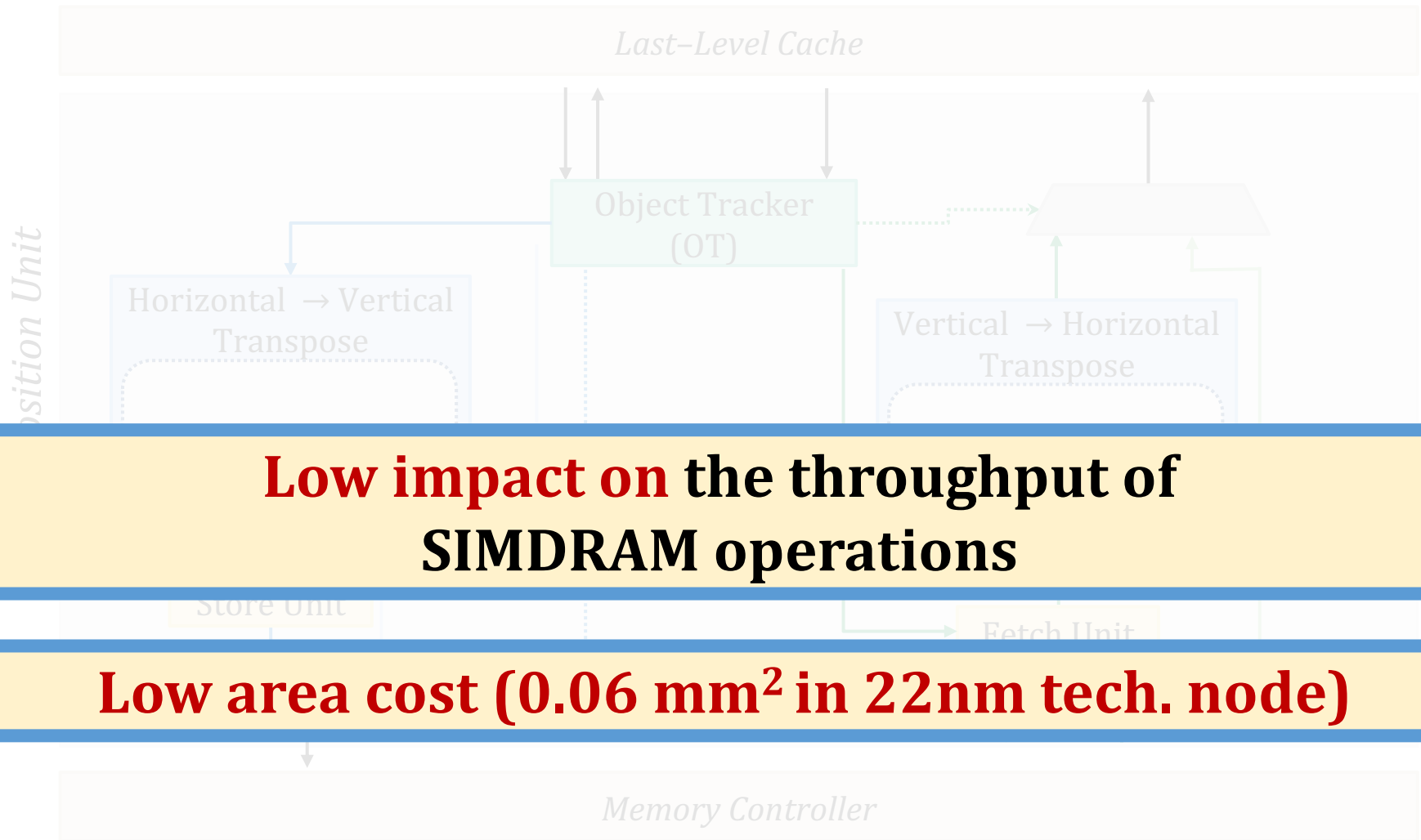
Challenging to share data between SIMD RAM and CPU

Transposition Unit

Transforms the data layout from **horizontal** to **vertical**, and vice versa



Efficiently Transposing Data



More in the Paper

Efficiently transposing data

Programming interface

Handling page faults, address translation,
coherence, and interrupts

Handling limited subarray size

Security implications

Limitations of our framework

Key Results

Evaluated on:

- 16 complex in-DRAM operations
- 7 commonly-used real-world applications

SIMDRAM provides:

- **88×** and **5.8×** the **throughput** of a **CPU** and a **high-end GPU**, respectively, over **16 operations**
- **257×** and **31×** the **energy efficiency** of a **CPU** and a **high-end GPU**, respectively, over **16 operations**
- **21×** and **2.1×** the **performance** of a **CPU** and a **high-end GPU**, over **seven real-world applications**

Conclusion

- **SIMDRAM:**

- Enables **efficient** computation of a **flexible** set and wide range of operations in a PuM **massively parallel** SIMD substrate
- Provides the hardware, programming, and ISA support, to:
 - Address key **system integration** challenges
 - Allow programmers to define and employ **new operations** without hardware changes

SIMDRAM is a promising PuM framework

- Can **ease the adoption** of processing-using-DRAM architectures
- Improve the **performance** and **efficiency** of processing-using-DRAM architectures

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

Our Approach

SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

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

Need a **holistic solution** to efficiently support modern applications, by:

- Efficiently handling **large amount of data**
- Exploiting **diverse properties** of modern applications data

Goal

Design an alternative virtual memory framework that

- Efficiently and flexibly supports increasingly diverse data properties and system configurations that come with it
- Provides the key features of conventional virtual memory frameworks while eliminating its key inefficiencies when handling large amount of data

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^{*‡}

^{*}*ETH Zürich* [†]*Simon Fraser University* [⌘]*University of Washington* [‡]*Carnegie Mellon University*
[⊙]*King Mongkut's University of Technology North Bangkok* [◇]*Boston University* [▽]*Microsoft Research India*

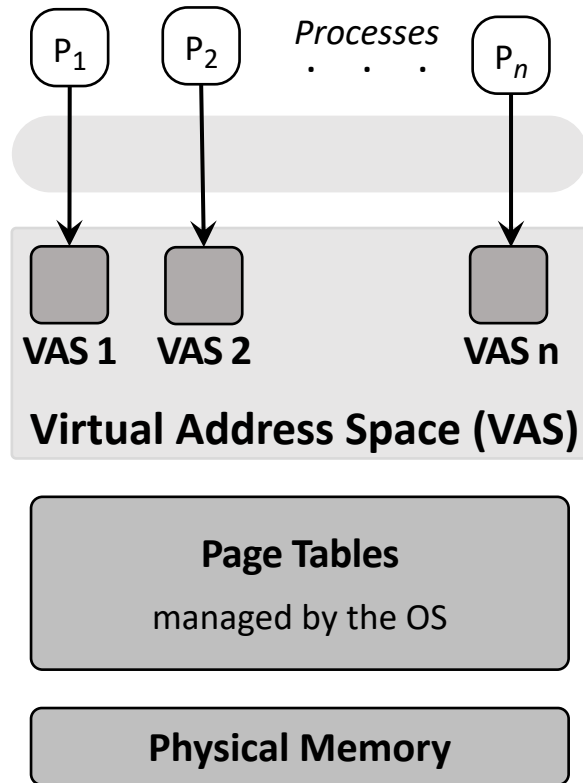
Key idea:

Delegate physical memory allocation and address translation to dedicated hardware in the **memory controller**

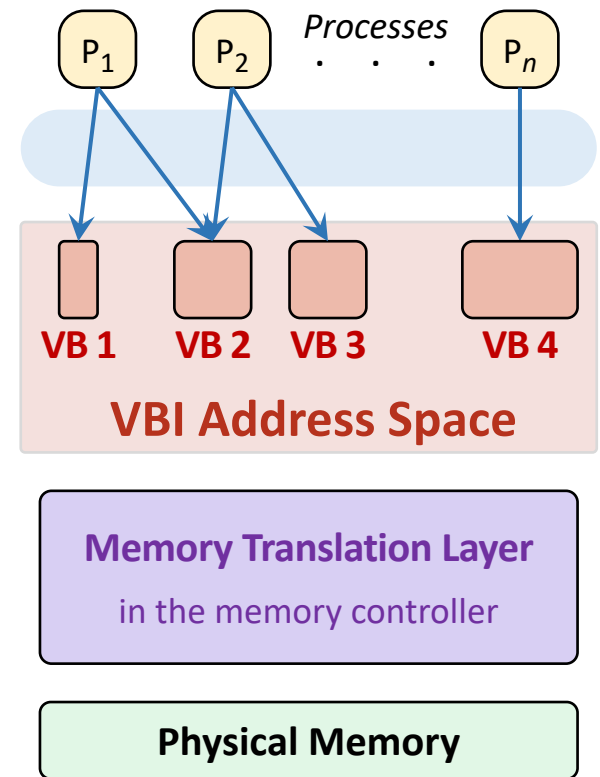
VBI: Guiding Principles

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

VBI: Overview



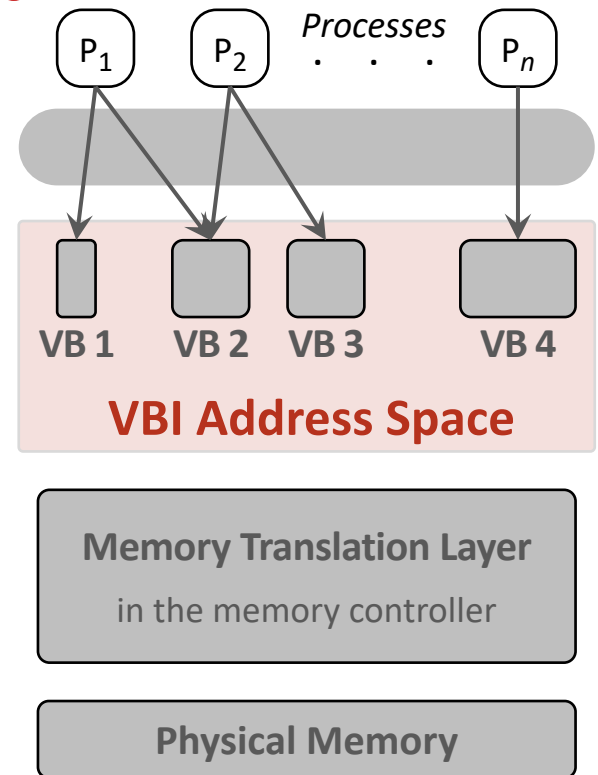
conventional virtual memory



VBI

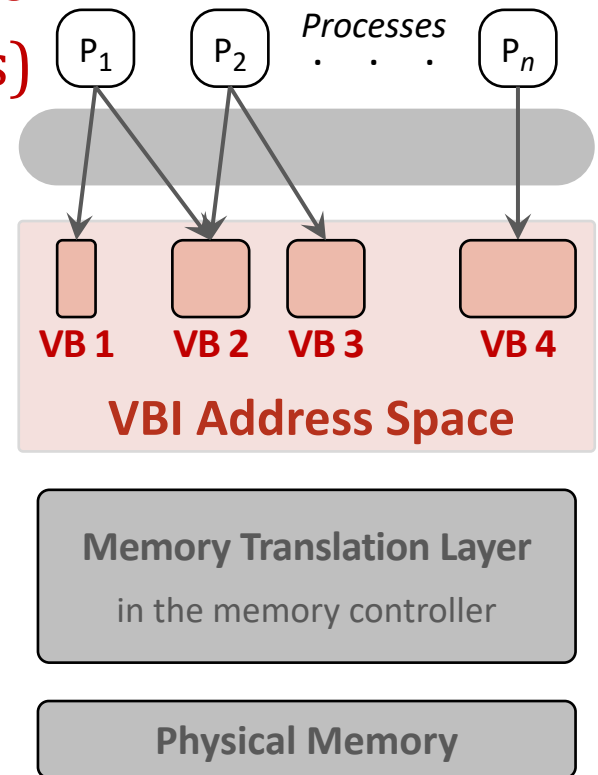
VBI: Overview

- Globally-visible *VBI address space*



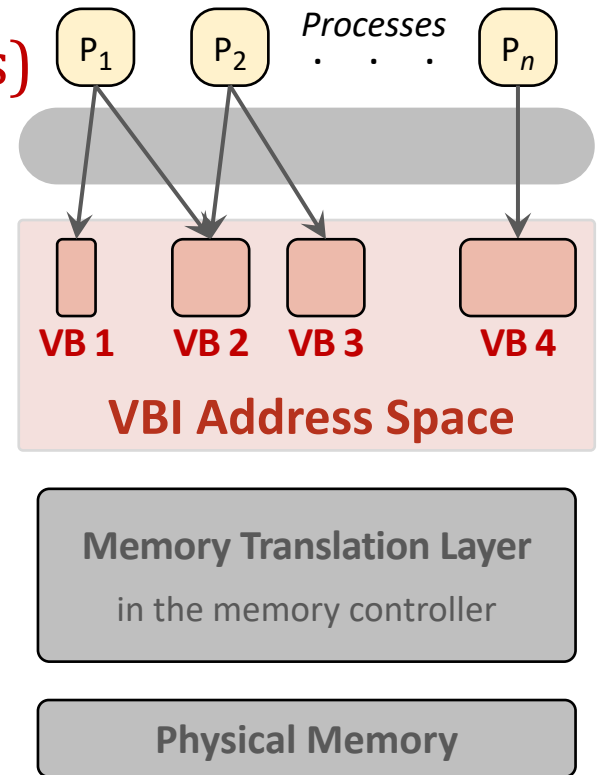
VBI: Overview

- 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



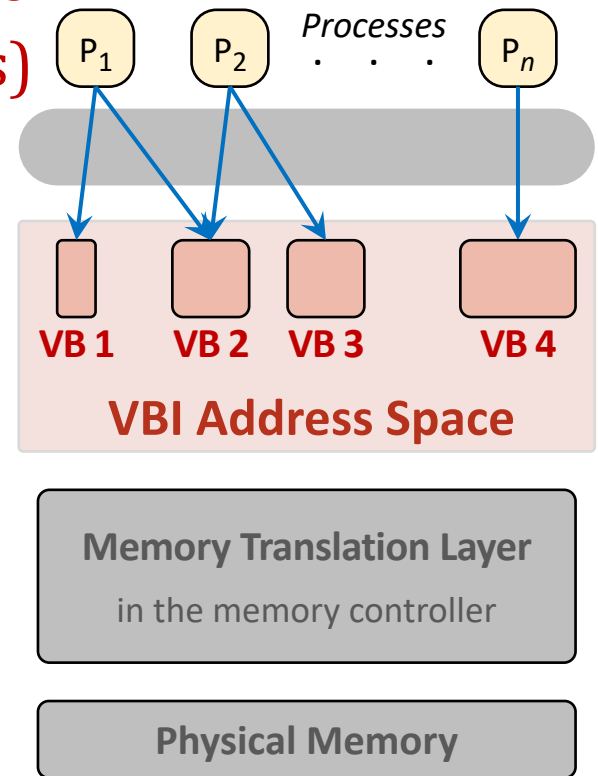
VBI: Overview

- 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



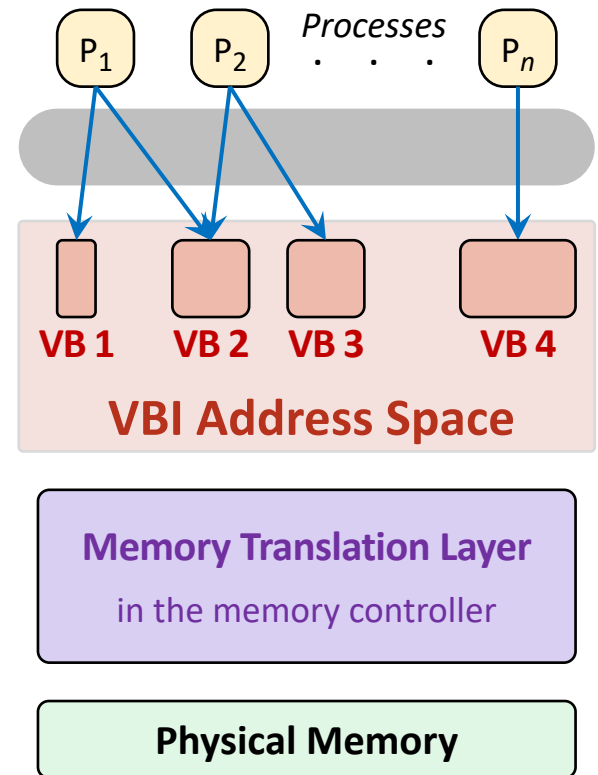
VBI: Overview

- 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



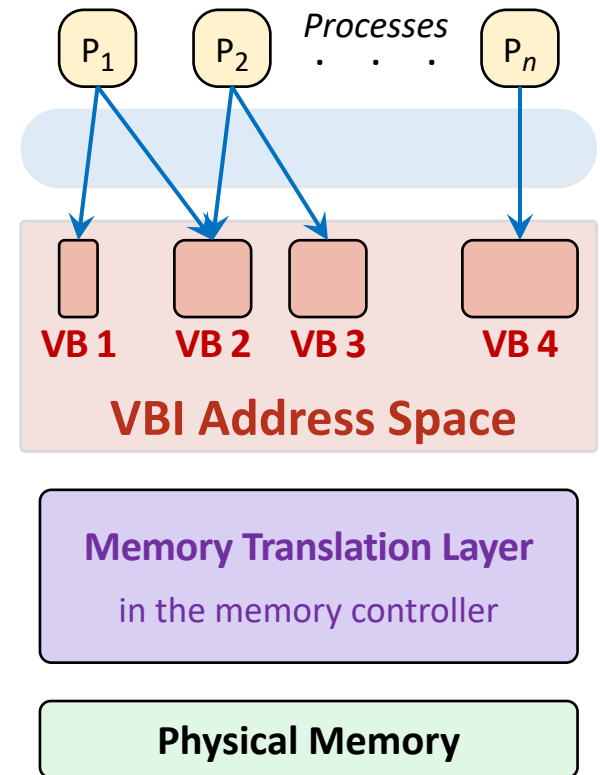
Hardware-Managed Memory

- VBI address space provides *system-wide* **unique VBI addresses**
- **VBI addresses** are **directly** used to access on-chip caches
 - No longer require address translation
- Memory management is **delegated** to the **Memory Translation Layer (MTL)** at the memory controller
 - Address translation
 - Physical memory allocation



OS-Managed Access Protection

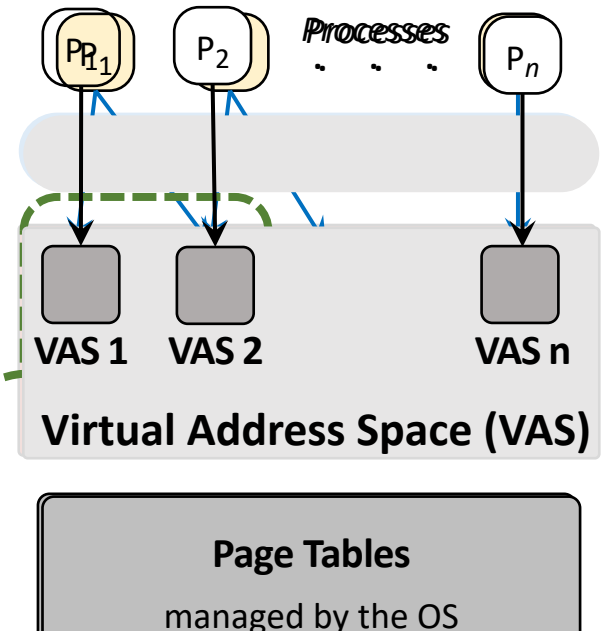
- OS controls which processes access which VBs
- Each process has **its own permissions** (read/write/execute) when **attaching** to a VB
- OS maintains a list of **VBs attached to each process**
 - Stored in a per-process table
 - Used during permission checks



Process Address Space in VBI

- Any process can attach to any VB
- A process' VBs define its **address space**
 - i.e., by the process' **actual** memory needs

the address space of
process P_1



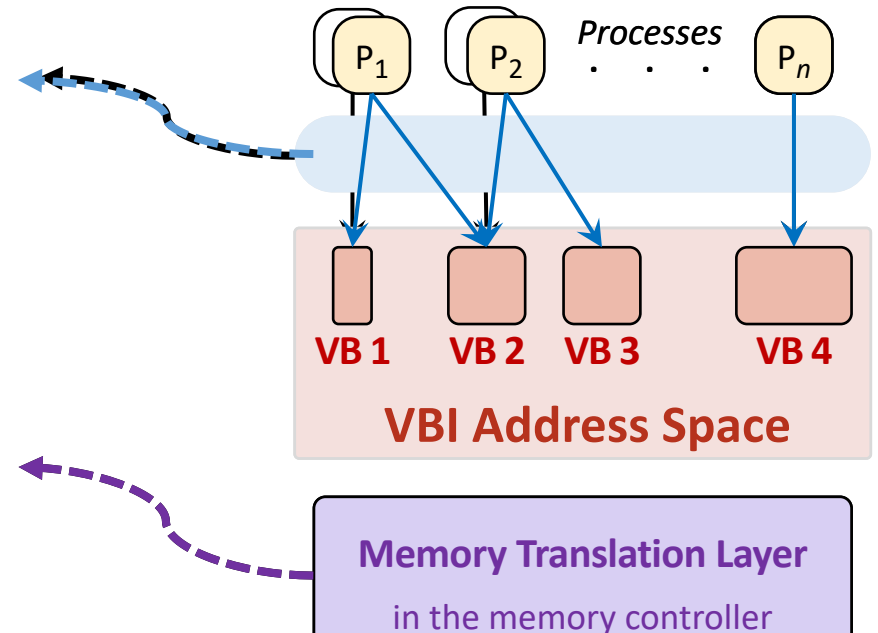
First guiding principle

Appropriately-sized virtual address spaces

Decoupled Protection and Translation

Access protection
managed by **OS**

Address mapping
managed by the **MTL**

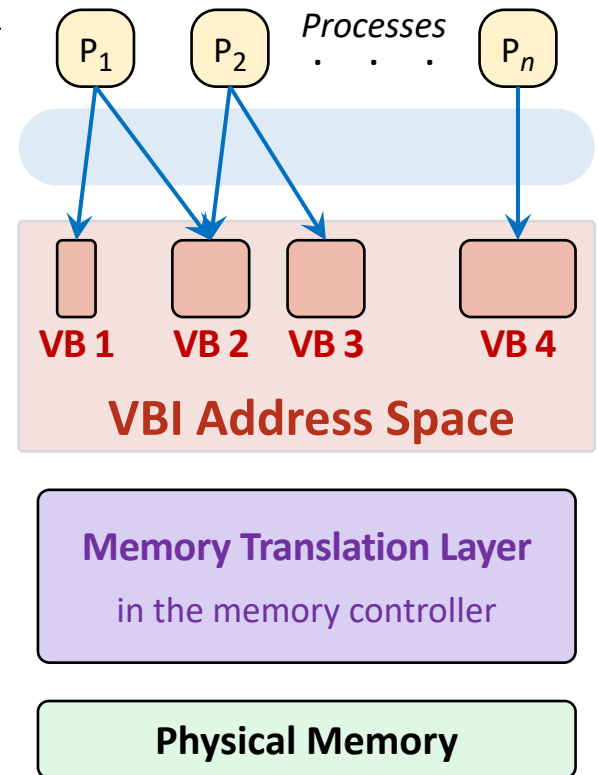


Second guiding principle

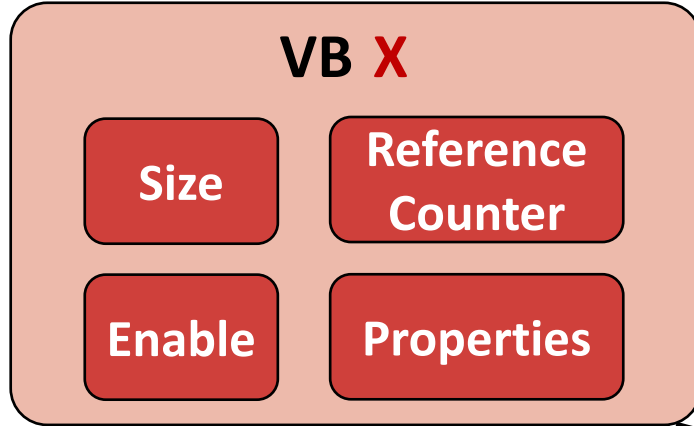
Decoupling address translation from access protection

Address Translation Structures in VBI

- **Separate** structures for translation and permission information
- Translation structures are **not shared** with the OS
 - Allows **flexible** translation structures
 - Per-VB translation structure
 - **Tuned** to the VB's characteristics
 - e.g., single-level tables for small VBs or those with many large contiguously allocated regions



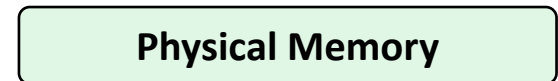
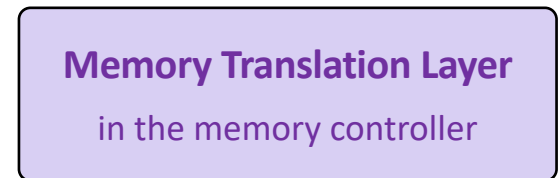
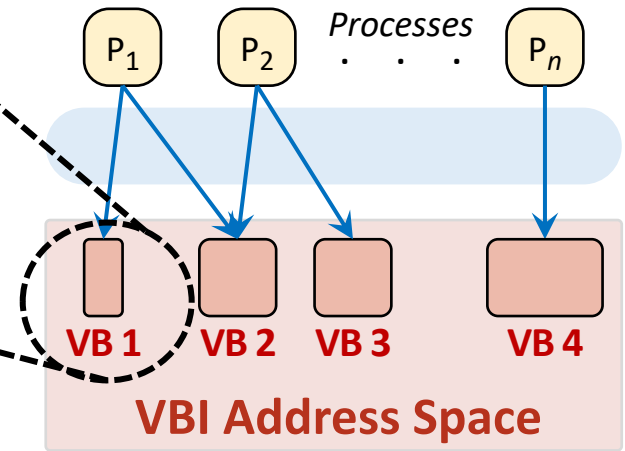
VB Information



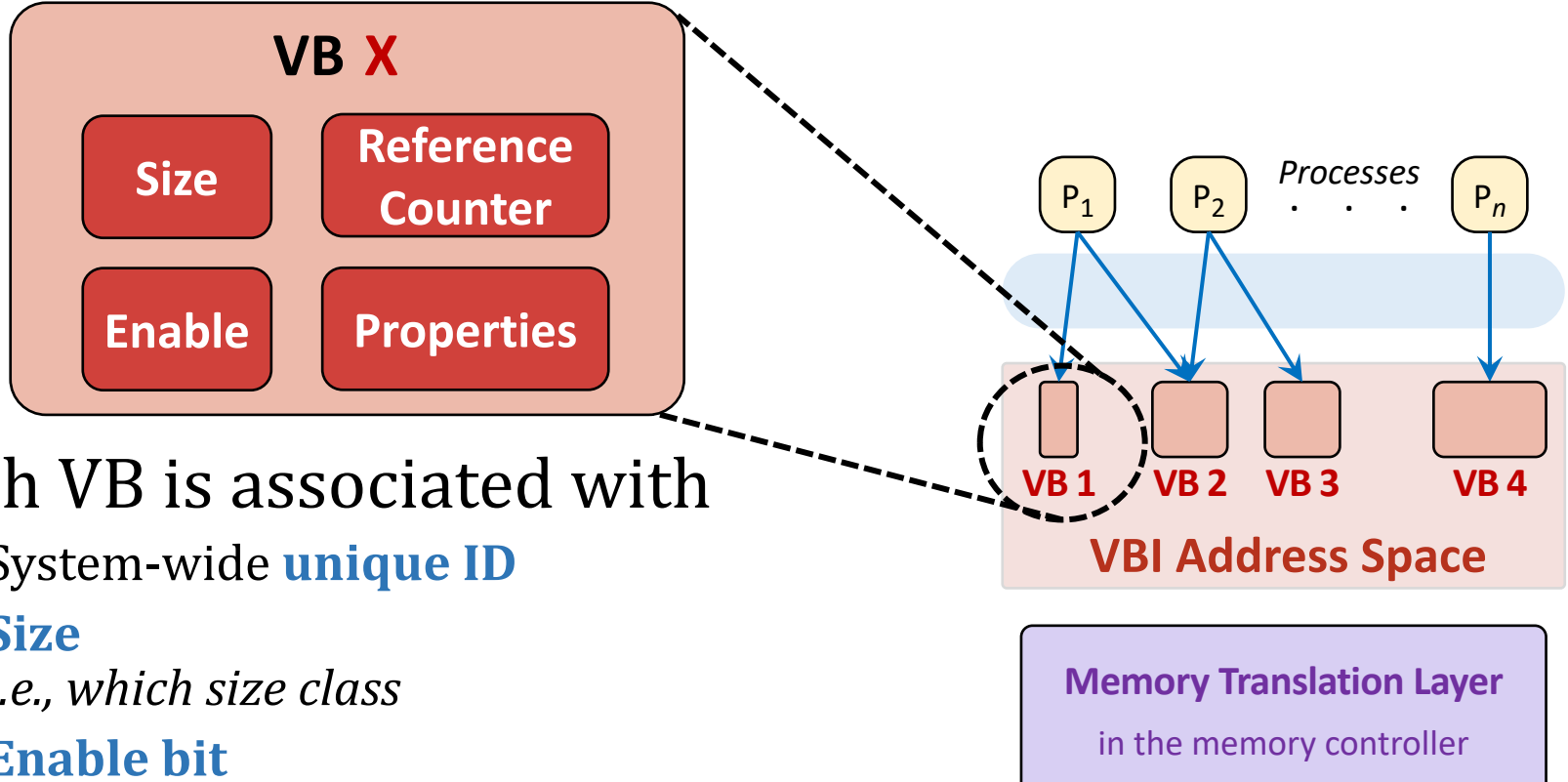
- Each VB is associated with

- 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



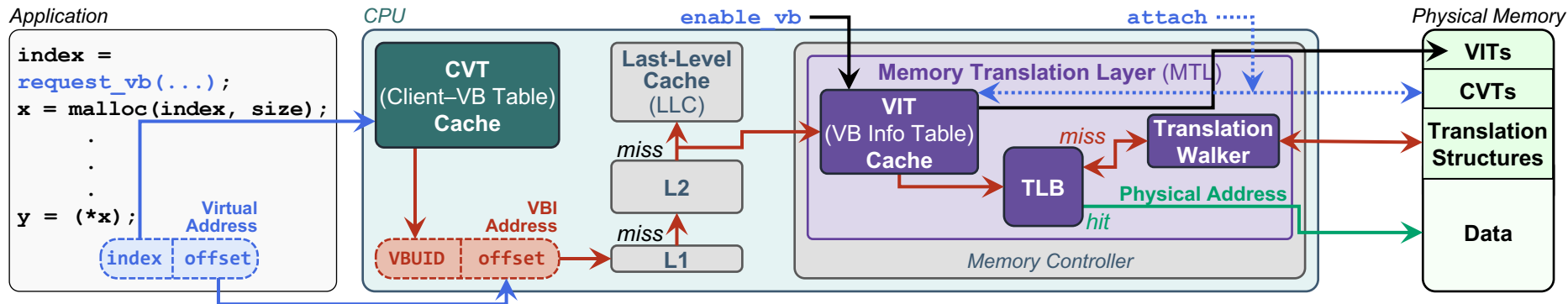
- Each VB is associated with
 - System-wide **unique ID**
 - **Size**
i.e., which size class
 - **Enable bit**

Third guiding principle

Communicating data semantics to the hardware

More in the Thesis

- More details on the challenges of adapting conventional virtual memory
- Detailed reference implementation and microarchitecture



Key Optimizations and Results

- **Benefits:** Many optimizations not easily attainable before. Examples:

- Appropriately sized process address space
- Flexible address translation structures
- Communicating data semantics to the hardware
- Inherently virtual caches
- Eliminating 2D page walks in virtual machines
- Delayed physical memory allocation
- Early memory reservation mechanism

Inherent to VBI design

Covered in the paper

- **Evaluation:** Two example use cases

- VBI significantly improves performance in both native execution and virtual machines (by 2.4x and 4.3x on average, respectively)
- Increases the effectiveness of managing heterogeneous memory architectures

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

Outline

Motivation

Computing Architectures Today

Processor-Centric Design

Data-Oblivious Policies

Our Approach

SIMDRAM: A Data-Centric Framework

VBI: A Data-Aware Framework

Conclusion and Future Work

Conclusion

Efficient data handling can be enabled by fundamental rethinking of the **computing paradigm** and **key concepts and components** in modern computing systems

- **Data-centric architectures** that minimize data movement and compute data in or near where the data resides
- **Data-aware frameworks** that makes use of different properties of data to improve performance, efficiency and other metrics

Future Directions

The ideas and approaches presented in this thesis can be **extended to tackle other issues** in modern computing systems. **For example:**

- Data-Aware Memory Architectures
 - Memory architectures that understand and exploit the properties of the data to make intelligent utilization decisions
- Virtual Memory Support for Processing-Using-Memory architectures
 - Efficient support in processing-using-memory architectures for critical virtual memory functionalities

Thesis Publications

- Enabling efficient data handling in modern computing systems
 - **“SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM”**
Nastaran Hajinazar^{*}, Geraldo F. Oliveira^{*}, Sven Gregorio, Joao Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gomez-Luna, and Onur Mutlu **[ASPLOS 2021]**
 - **“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 Francisco de Oliveira Jr., Jonathan Appavoo, Vivek Seshadri, and Onur Mutlu **[ISCA 2020]**

Other PhD Publications

- **A Case for Richer Cross-layer Abstractions: Bridging the Semantic Gap with Expressive Memory** [Vijaykumar+, ISCA 2018]
- **CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators** [Boroumand+, ISCA 2019]
- **Demystifying complex workload-DRAM interactions: An experimental study** [Ghose+, SIGMETRICS 2019]
- **AirLift: A Fast and Comprehensive Technique for Remapping Alignments between Reference Genomes** [Kim+, Preprint in bioRxiv]

Acknowledgement



- **Onur Mutlu**
- **Arrvindh Shriraman, Saugata Ghose, and Vivek Seshadri**
- **SAFARI research group members at CMU and ETH:**
Giray Yaglikci, Geraldo De Oliveira, Juan Gómez Luna, Damla Senol Cali, Rachata Ausavarungnirun, Minesh Patel, Hasan Hassan, João Dinis Sanches Ferreira, Konstantinos Kanellopoulos, Mohammed Alser, Christian Rossi, Tracy Ewen, Gagandeep Singh, Jeremie Kim, Can Firtina, Nika Mansourighiasi, Jisung Park, Lois Orosa, Kevin Hsieh, Kevin Chang
- **Friends:** Rajesh, Teena, Amir, Nasibeh, Sogol, ...
- **Family:** My husband, my parents, my brothers, and my soon-to-be-born son

Data-Centric and Data-Aware Frameworks for Fundamentally Efficient Data Handling in Modern Computing Systems

Nastaran Hajinazar

PhD Thesis Defense Presentation - 4 June 2021

Committee: Prof. Onur Mutlu (Co-Senior Supervisor)
Prof. Arrvindh Shriraman (Co-Senior Supervisor)
Prof. Saugata Ghose
Dr. Vivek Seshadri (Microsoft)
Prof. Alaa Alameldeen
Prof. Myoungsoo Jung

SAFARI



SIMON FRASER
UNIVERSITY

Backup Slides