Computer Architecture
Lecture 8: Computation in Memory III

Prof. Onur Mutlu

ETH Zürich

Fall 2019

11 October 2019

Sub-Agenda: In-Memory Computation

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
  - Bottom Up: Push from Circuits and Devices
  - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
  - Minimally Changing Memory Chips
  - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion

SAFARI

Several Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
  - By changing the entire system
  - By performing simple function offloading

- What is the minimal processing-in-memory support we can provide?
  - With minimal changes to system and programming

Recall: Tesseract

- Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi,
  "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing"
  Proceedings of the 42nd International Symposium on Computer Architecture (**ISCA**), Portland, OR, June 2015.
  [Slides (pdf)] [Lightning Session Slides (pdf)]

A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing

Junwhan Ahn   Sungpack Hong[§]   Sungjoo Yoo   Onur Mutlu[†]   Kiyoung Choi

junwhan@snu.ac.kr, sungpack.hong@oracle.com, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr

Seoul National University   [§]Oracle Labs   [†]Carnegie Mellon University

Several Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
  - By changing the entire system
  - By performing simple function offloading

- What is the minimal processing-in-memory support we can provide?
  - With minimal changes to system and programming

3D-Stacked PIM on Mobile Devices

- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu,
"Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"
Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**), Williamsburg, VA, USA, March 2018.

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand[1]　　Saugata Ghose[1]　　Youngsok Kim[2]

Rachata Ausavarungnirun[1]　　Eric Shiu[3]　　Rahul Thakur[3]　　Daehyun Kim[4,3]

Aki Kuusela[3]　　Allan Knies[3]　　Parthasarathy Ranganathan[3]　　Onur Mutlu[5,1]

Consumer Devices

Consumer devices are everywhere!

**Energy consumption is
a first-class concern in consumer devices**

Four Important Workloads

Chrome

Google's web browser

TensorFlow Mobile

Google's machine learning framework
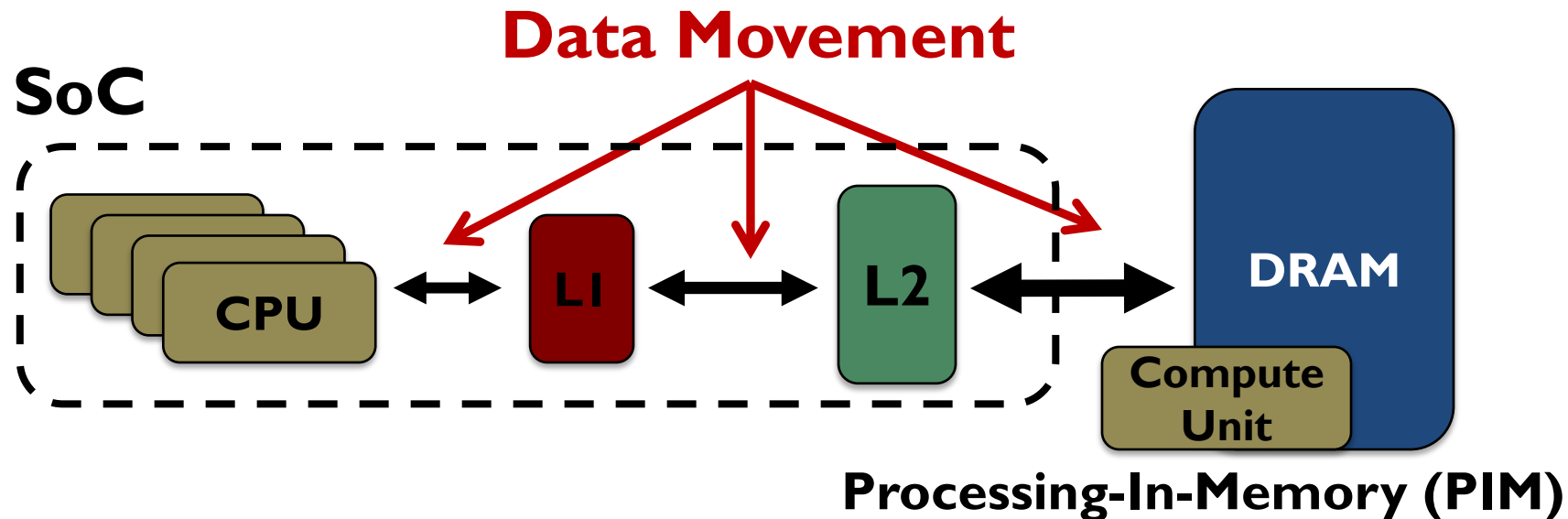
Video Playback

Google's **video codec**

Video Capture

Google's **video codec**

Energy Cost of Data Movement

1st key observation: 62.7% of the total system energy is spent on data movement

Data Movement

SoC

CPU ⟷ L1 ⟷ L2 ⟷ DRAM

Compute Unit

Processing-In-Memory (PIM)

Potential solution: move computation close to data

Challenge: limited area and energy budget

SAFARI

Using PIM to Reduce Data Movement

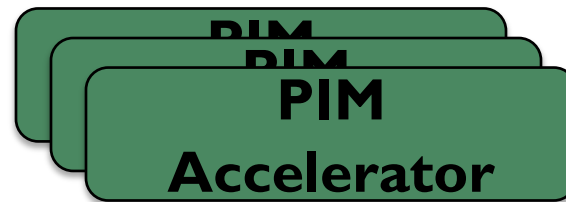**2nd key observation: a significant fraction of the data movement often comes from simple functions**

We can design lightweight logic to implement these *simple functions* in **memory**

Small embedded low-power core

Small fixed-function accelerators

PIM
Core

PIM
PIM
PIM
Accelerator

Offloading to PIM logic reduces energy and improves performance, on average, by 55.4% and 54.2%

Workload Analysis

Chrome

Google's web browser

TensorFlow Mobile

Google's machine learning framework
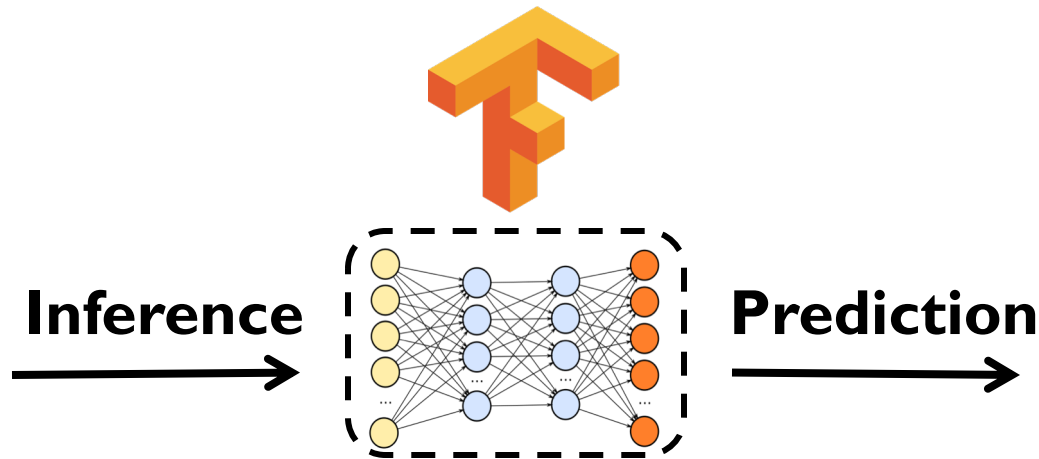
Video Playback

Google's **video codec**

Video Capture

Google's **video codec**

SAFARI

TensorFlow Mobile

Inference → **Prediction**

57.3% of the inference energy is spent on
data movement

↓

54.4% of the **data movement** energy comes from
packing/unpacking and **quantization**

SAFARI

Packing

Matrix → **Packing** → Packed Matrix

Reorders elements of matrices to minimize **cache misses** during **matrix multiplication**

↓

Up to **40%** of the inference **energy** and **31%** of inference **execution time**

↓

Packing's data movement accounts for up to **35.3%** of the inference **energy**

A simple data reorganization process that requires simple arithmetic

SAFARI

Quantization

floating point → **Quantization** → integer

Converts <u>32-bit floating point</u> to <u>8-bit integers</u> to improve inference <u>execution time</u> and <u>energy consumption</u>

↓ ↓

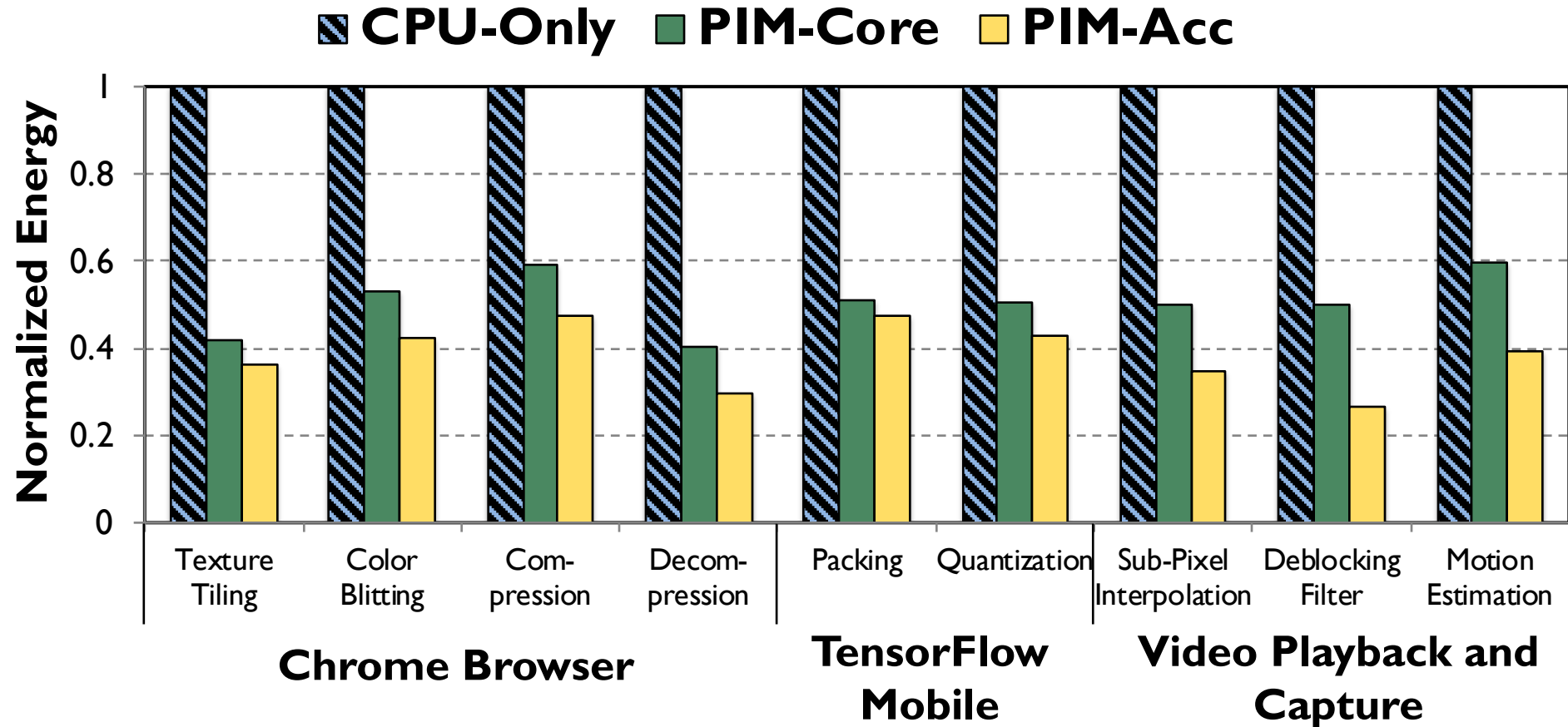**Up to 16.8% of the inference energy and 16.1% of inference execution time**

Majority of quantization energy comes from data movement

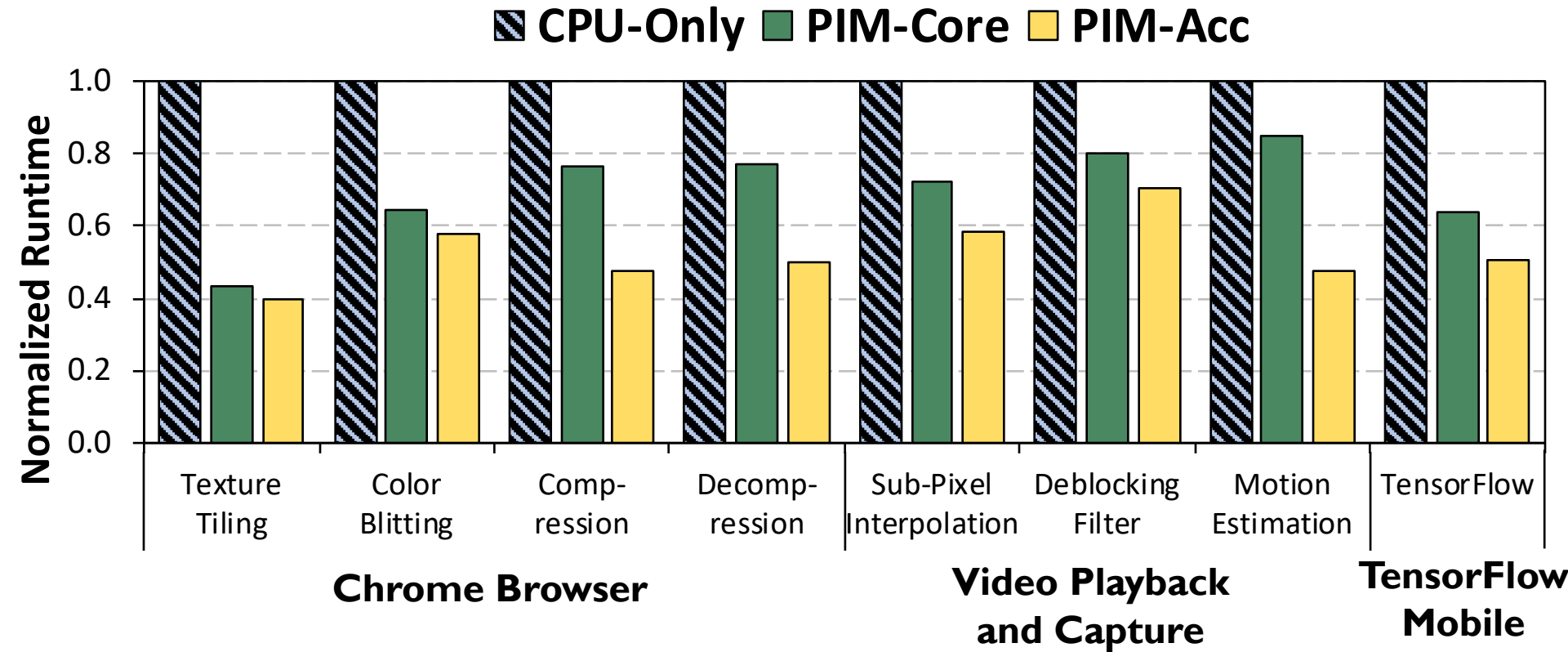**A simple data conversion operation that requires shift, addition, and multiplication operations**

Normalized Energy

PIM core and PIM accelerator reduce energy consumption on average by 49.1% and 55.4%

Normalized Runtime

Offloading these kernels to PIM core and PIM accelerator improves performance on average by 44.6% and 54.2%

Workload Analysis

Chrome

Google's web browser

TensorFlow

Google's machine learning framework
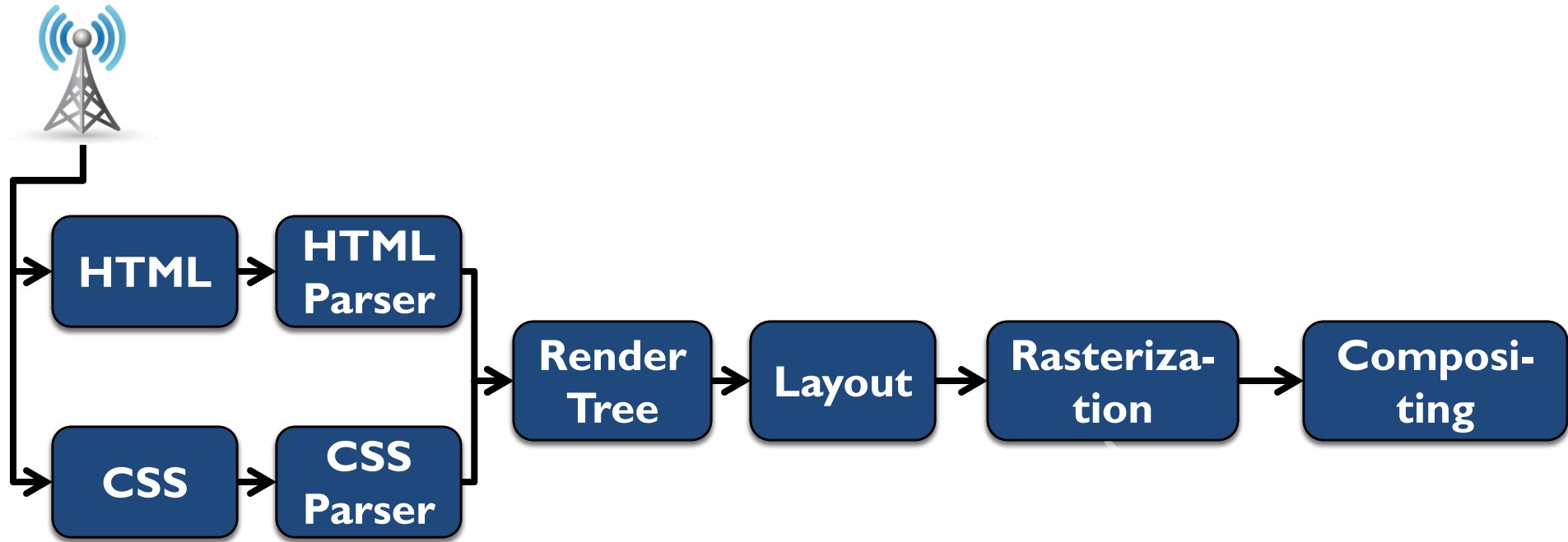
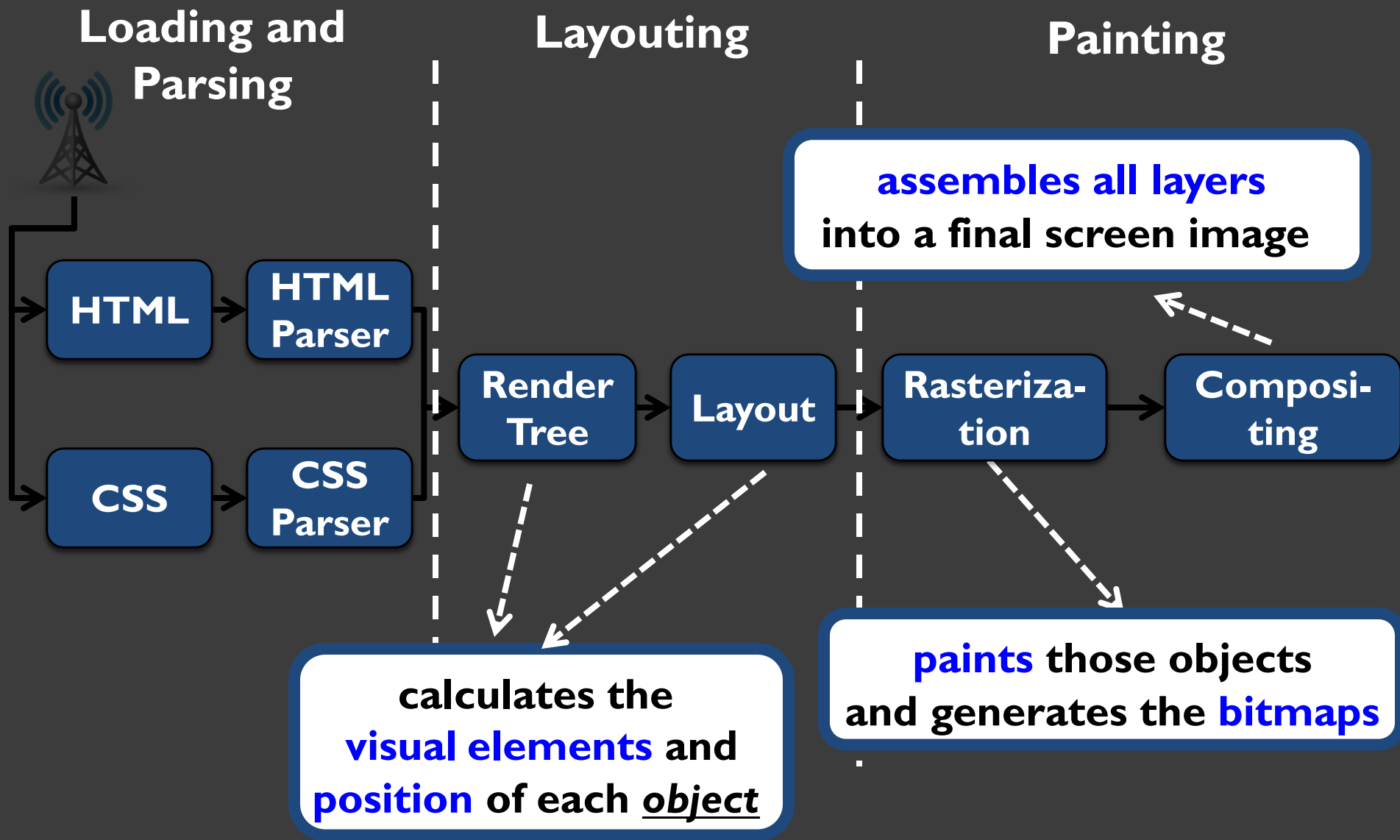**Video Playback**

Google's video codec

Video Capture

Google's video codec

How Chrome Renders a Web Page

```
            [HTML] → [HTML Parser]
                              ↘
                                [Render Tree] → [Layout] → [Rasteriza-tion] → [Composi-ting]
                              ↗
            [CSS] → [CSS Parser]
```

How Chrome Renders a Web Page

Loading and Parsing

Layouting

Painting

HTML → **HTML Parser**

CSS → **CSS Parser**

Render Tree → **Layout** → **Rasteriza- tion** → **Composi- ting**

assembles all layers into a final screen image

calculates the visual elements and position of each _object_

paints those objects and generates the bitmaps

Browser Analysis

- **To satisfy user experience, the browser must provide:**
  - **Fast loading of webpages**
  - **Smooth scrolling of webpages**
  - **Quick switching between browser tabs**

- **We focus on two important user interactions:**
  1. **Page Scrolling**
  2. **Tab Switching**
  - **Both include <u>page loading</u>**

Tab Switching

What Happens During Tab Switching?

- **Chrome employs a multi-process architecture**
  - **Each tab is a <u>separate process</u>**

Chrome Process

Tab 1 Process   Tab 2 Process   • • •   Tab N Process
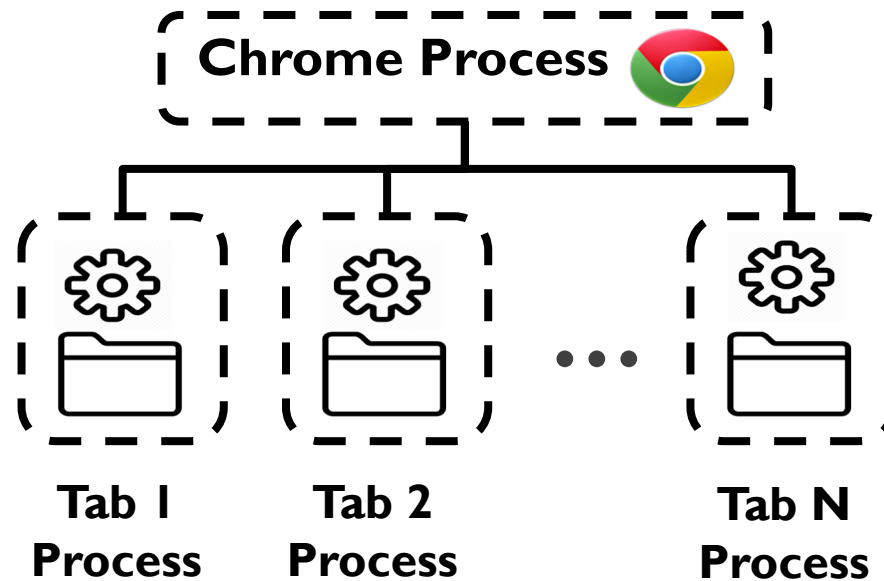
- **Main operations during tab switching:**
  - **Context switch**
  - **Load the new page**

Memory Consumption

- **Primary concerns during tab switching:**
  - **How fast a new tab loads and becomes interactive**
  - **Memory consumption**

Chrome uses compression to reduce each tab's memory footprint

CPU

Compress Tab
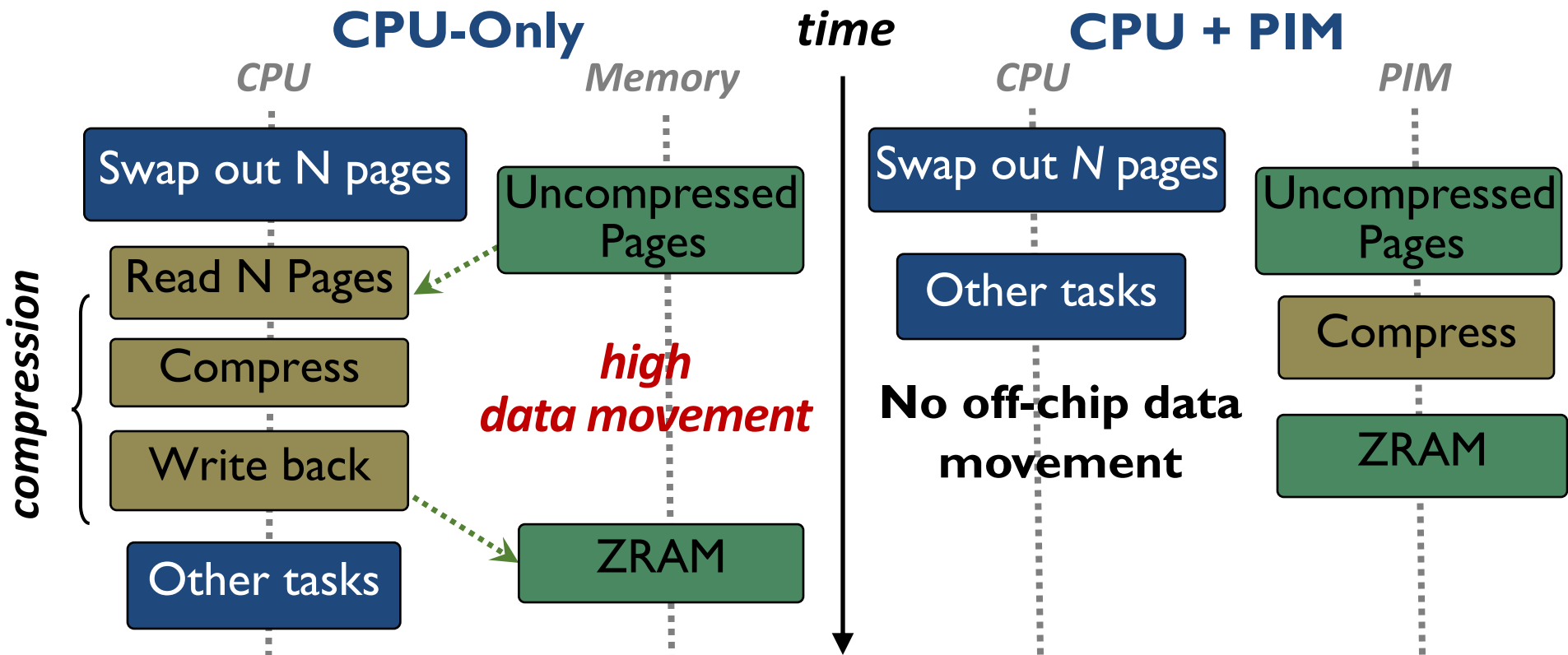Inactive Tab
Compression
Decompression

DRAM

ZRAM
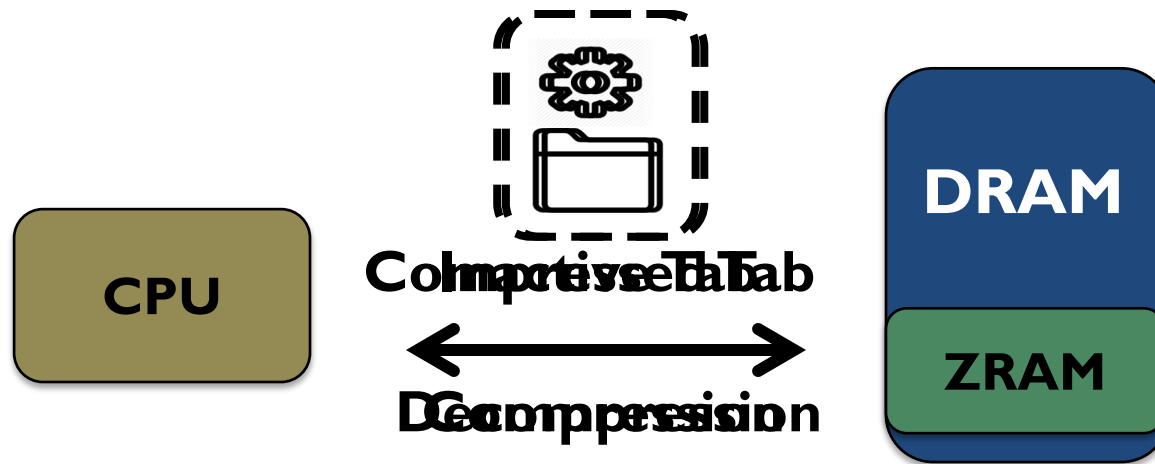
Data Movement Study

- **To study data movement during tab switching, we emulate a user switching through 50 tabs**

We make two key observations:

1	**Compression and decompression** contribute to **18.1%** of the total system energy
2	**19.6 GB** of data moves between **CPU and ZRAM**

Can We Use PIM to Mitigate the Cost?

CPU-Only

time

CPU + PIM

CPU   *Memory*   *CPU*   *PIM*

Swap out N pages

Uncompressed Pages

compression

Read N Pages

Compress

Write back

high data movement

Other tasks

ZRAM

Swap out *N* pages

Other tasks

No off-chip data movement

Uncompressed Pages

Compress

ZRAM

PIM core and PIM accelerator are feasible to implement in-memory compression/decompression

SAFARI

30

Tab Switching Wrap Up

A large amount of data movement happens during tab switching as Chrome attempts to compress and decompress tabs

Both functions can benefit from PIM execution and can be implemented as PIM logic

More on PIM for Mobile Devices

- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu, **"Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"** *Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems* (**ASPLOS**), Williamsburg, VA, USA, March 2018.

62.7% of the total system energy is spent on **data movement**

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand[1]   Saugata Ghose[1]   Youngsok Kim[2]

Rachata Ausavarungnirun[1]   Eric Shiu[3]   Rahul Thakur[3]   Daehyun Kim[4,3]

Aki Kuusela[3]   Allan Knies[3]   Parthasarathy Ranganathan[3]   Onur Mutlu[5,1]

Truly Distributed GPU Processing with PIM?
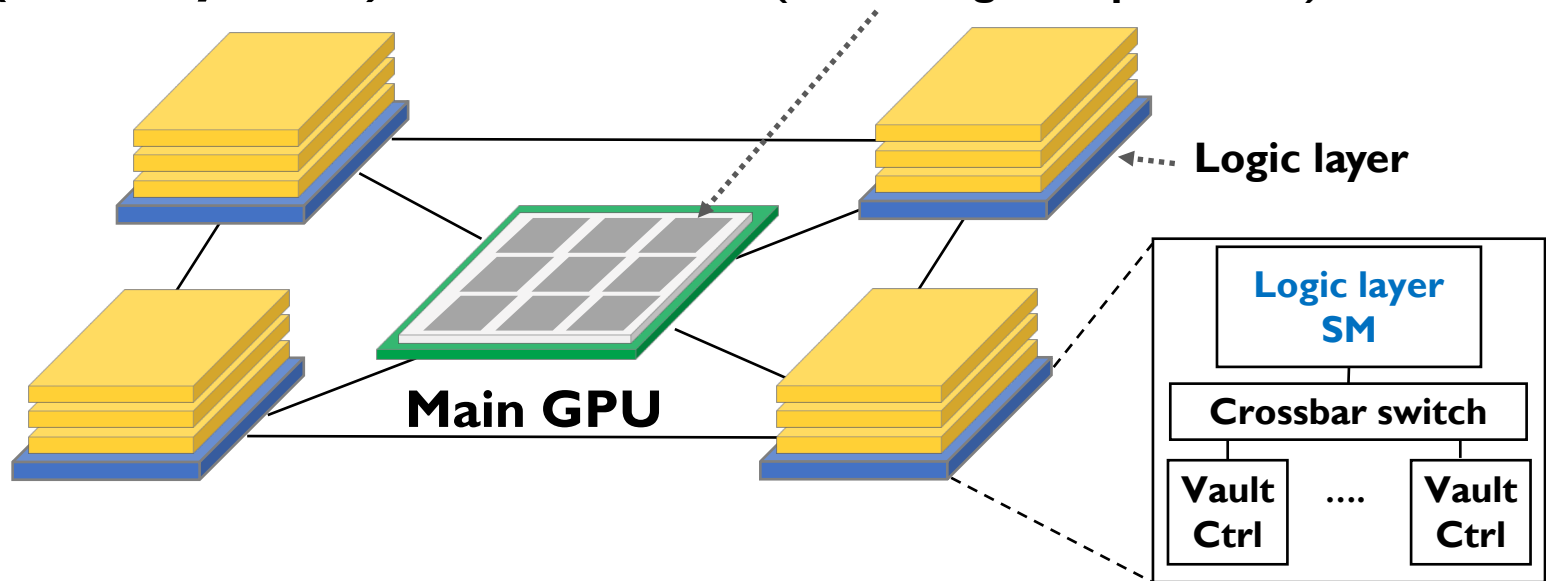
```
__global__
void applyScaleFactorsKernel( uint8_T * const out,
    uint8_T const * const in, const double *factor,
    size_t const numRows, size_t const numCols )
{
    // Work out which pixel we are working on.
    const int rowIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int colIdx = blockIdx.y;
    const int sliceIdx = threadIdx.z;

    // Check this thread isn't off the image
    if( rowIdx >= numRows ) return;

    // Compute the index of my element
    size_t linearIdx = rowIdx + colIdx*numRows +
        sliceIdx*numRows*numCols;
```

3D-stacked memory (memory stack)

SM (Streaming Multiprocessor)

Logic layer

Main GPU

Logic layer SM

Crossbar switch

Vault Ctrl .... **Vault Ctrl**

Accelerating GPU Execution with PIM (I)

- Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler,
"Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems"
Proceedings of the 43rd International Symposium on Computer Architecture (**ISCA**), Seoul, South Korea, June 2016.
[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]

Transparent Offloading and Mapping (TOM):
Enabling Programmer-Transparent Near-Data Processing in GPU Systems

Kevin Hsieh[‡]   Eiman Ebrahimi[†]   Gwangsun Kim[*]   Niladrish Chatterjee[†]   Mike O'Connor[†]
Nandita Vijaykumar[‡]   Onur Mutlu[§‡]   Stephen W. Keckler[†]
[‡]**Carnegie Mellon University**   [†]**NVIDIA**   [*]**KAIST**   [§]**ETH Zürich**

Accelerating GPU Execution with PIM (II)

- Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, <u>Onur Mutlu</u>, and Chita R. Das,
  "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities"
  Proceedings of the <u>*25th International Conference on Parallel Architectures and Compilation Techniques*</u> (**PACT**), Haifa, Israel, September 2016.

Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities

Ashutosh Pattnaik[1]   Xulong Tang[1]   Adwait Jog[2]   Onur Kayıran[3]
Asit K. Mishra[4]   Mahmut T. Kandemir[1]   Onur Mutlu[5,6]   Chita R. Das[1]

[1]Pennsylvania State University   [2]College of William and Mary
[3]Advanced Micro Devices, Inc.   [4]Intel Labs   [5]ETH Zürich   [6]Carnegie Mellon University

Accelerating Linked Data Structures

- Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu,
"Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation"
Proceedings of the 34th IEEE International Conference on Computer Design (**ICCD**), Phoenix, AZ, USA, October 2016.

Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation

Kevin Hsieh[†]   Samira Khan[‡]   Nandita Vijaykumar[†]
Kevin K. Chang[†]   Amirali Boroumand[†]   Saugata Ghose[†]   Onur Mutlu[§†]
[†]*Carnegie Mellon University*   [‡]*University of Virginia*   [§]*ETH Zürich*

Accelerating Dependent Cache Misses

- Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt,
"Accelerating Dependent Cache Misses with an Enhanced Memory Controller"
Proceedings of the 43rd International Symposium on Computer Architecture (**ISCA**), Seoul, South Korea, June 2016.
[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

Milad Hashemi[*], Khubaib[†], Eiman Ebrahimi[‡], Onur Mutlu[§], Yale N. Patt[*]

[*]*The University of Texas at Austin*   [†]*Apple*   [‡]*NVIDIA*   [§]*ETH Zürich & Carnegie Mellon University*

Accelerating Runahead Execution

- Milad Hashemi, Onur Mutlu, and Yale N. Patt,
  "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads"
  Proceedings of the 49th International Symposium on Microarchitecture (**MICRO**), Taipei, Taiwan, October 2016.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pdf)] [Poster (pptx) (pdf)]

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi[*], Onur Mutlu[§], Yale N. Patt[*]

[*]The University of Texas at Austin   [§]ETH Zürich

Several Questions in 3D-Stacked PIM

- What are the performance and energy benefits of using 3D-stacked memory as a coarse-grained accelerator?
  - By changing the entire system
  - By performing simple function offloading

- What is the minimal processing-in-memory support we can provide?
  - With minimal changes to system and programming

SAFARI

PIM-Enabled Instructions

- Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi,
  "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture"
  Proceedings of the 42nd International Symposium on Computer Architecture (**ISCA**), Portland, OR, June 2015.
  [Slides (pdf)] [Lightning Session Slides (pdf)]

PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture

Junwhan Ahn   Sungjoo Yoo   Onur Mutlu[†]   Kiyoung Choi
junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University   [†]Carnegie Mellon University

SAFARI

PEI: PIM-Enabled Instructions (Ideas)

- **Goal:** Develop mechanisms to get the most out of near-data processing with minimal cost, minimal changes to the system, no changes to the programming model

- **Key Idea 1:** Expose each PIM operation as a cache-coherent, virtually-addressed host processor instruction (called PEI) that operates on only a single cache block
  - e.g., __pim_add(&w.next_rank, value) → pim.add r1, (r2)
  - No changes sequential execution/programming model
  - No changes to virtual memory
  - Minimal changes to cache coherence
  - No need for data mapping: Each PEI restricted to a single memory module

- **Key Idea 2:** Dynamically decide where to execute a PEI (i.e., the host processor or PIM accelerator) based on simple locality characteristics and simple hardware predictors
  - Execute each operation at the location that provides the best performance

Simple PIM Operations as ISA Extensions (II)

```
for (v: graph.vertices) {
    value = weight * v.rank;
    for (w: v.successors) {
        w.next_rank += value;
    }
}
```

Host Processor

w.next_rank

Main Memory

w.next_rank

64 bytes **in**
64 bytes **out**
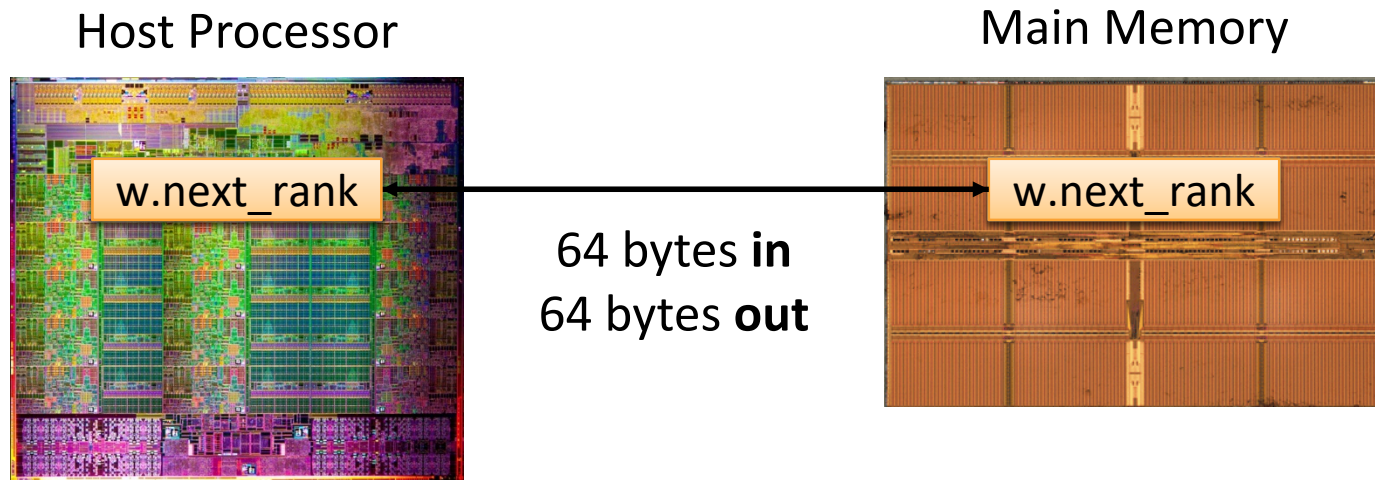
Conventional Architecture

Simple PIM Operations as ISA Extensions (III)

```
for (v: graph.vertices) {
    value = weight * v.rank;
    for (w: v.successors) {
        __pim_add(&w.next_rank, value);
    }
}
```

pim.add r1, (r2)

Host Processor
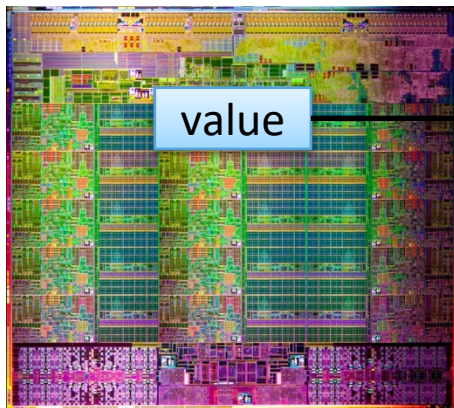
Main Memory

value

w.next_rank

8 bytes **in**
0 bytes **out**

In-Memory Addition

Always Executing in Memory? Not A Good Idea

Increased Memory Bandwidth Consumption
Caching very effective

Reduced Memory Bandwidth Consumption due to
In-Memory Computation

Speedup

More Vertices

p2p-Gnu tella31 | soc-Slash dot0811 | web-Stanford | amazon-2008 | frw 20 | w T | ci Pate | soc-L Journ | ljour 200
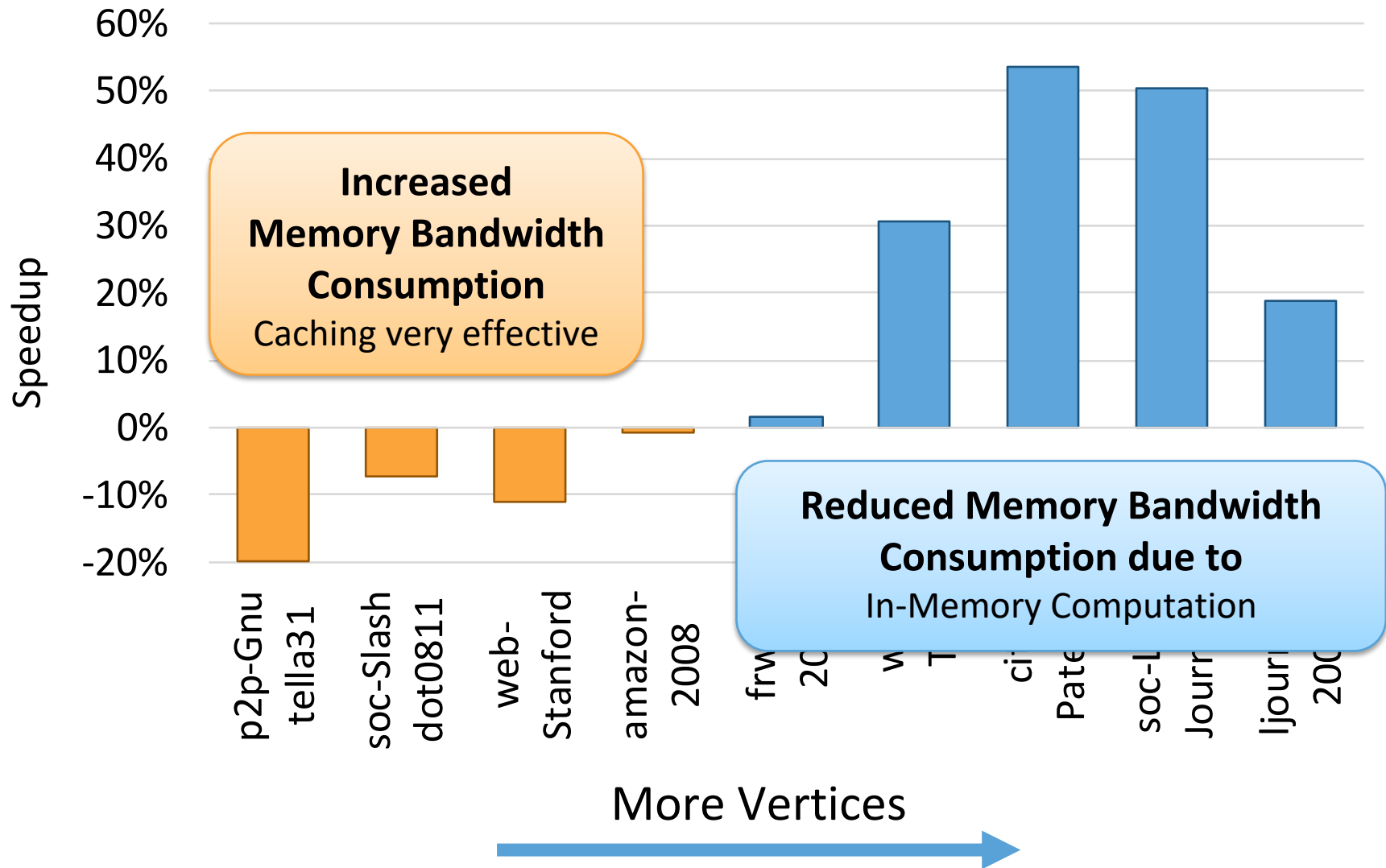
PEI: PIM-Enabled Instructions (Example)

```
for (v: graph.vertices) {
    value = weight * v.rank;
    for (w: v.successors) {
        __pim_add(&w.next_rank, value);
    }
}
pfence();
```

pim.add r1, (r2)

pfence

Table 1: Summary of Supported PIM Operations

Operation	R	W	Input	Output	Applications
8-byte integer increment	O	O	0 bytes	0 bytes	AT
8-byte integer min	O	O	8 bytes	0 bytes	BFS, SP, WCC
Floating-point add	O	O	8 bytes	0 bytes	PR
Hash table probing	O	X	8 bytes	9 bytes	HJ
Histogram bin index	O	X	1 byte	16 bytes	HG, RP
Euclidean distance	O	X	64 bytes	4 bytes	SC
Dot product	O	X	32 bytes	8 bytes	SVM

- Executed either in memory or in the processor: dynamic decision
  - Low-cost locality monitoring for a single instruction
- Cache-coherent, virtually-addressed, single cache block only
- Atomic between different PEIs
- *Not* atomic with normal instructions (use *pfence* for ordering)

PIM-Enabled Instructions

- Key to practicality: single-cache-block restriction
  - Each PEI can access *at most one last-level cache block*
  - Similar restrictions exist in atomic instructions

- Benefits
  - **Localization**: each PEI is bounded to one memory module
  - **Interoperability**: easier support for cache coherence and virtual memory
  - **Simplified locality monitoring**: data locality of PEIs can be identified simply by the cache control logic

PEI: Initial Evaluation Results

- Initial evaluations with 10 emerging data-intensive workloads
  - Large-scale graph processing
  - In-memory data analytics
  - Machine learning and data mining
  - Three input sets (small, medium, large) for each workload to analyze the impact of data locality

Table 2: Baseline Simulation Configuration

Component	Configuration
Core	16 out-of-order cores, 4 GHz, 4-issue
L1 I/D-Cache	Private, 32 KB, 4/8-way, 64 B blocks, 16 MSHRs
L2 Cache	Private, 256 KB, 8-way, 64 B blocks, 16 MSHRs
L3 Cache	Shared, 16 MB, 16-way, 64 B blocks, 64 MSHRs
On-Chip Network	Crossbar, 2 GHz, 144-bit links
Main Memory	32 GB, 8 HMCs, daisy-chain (80 GB/s full-duplex)
HMC	4 GB, 16 vaults, 256 DRAM banks [20]
– DRAM	FR-FCFS, tCL = tRCD = tRP = 13.75 ns [27]
– Vertical Links	64 TSVs per vault with 2 Gb/s signaling rate [23]

- Pin-based cycle-level x86-64 simulation

- **Performance Improvement and Energy Reduction:**
  - 47% average speedup with large input data sets
  - 32% speedup with small input data sets
  - 25% avg. energy reduction in a single node with large input data sets

SAFARI

Evaluated Data-Intensive Applications

- **Ten emerging data-intensive workloads**
  - Large-scale graph processing
    - Average teenage follower, BFS, PageRank, single-source shortest path, weakly connected components
  - In-memory data analytics
    - Hash join, histogram, radix partitioning
  - Machine learning and data mining
    - Streamcluster, SVM-RFE

- Three input sets (small, medium, large) for each workload to show the impact of data locality

SAFARI

PEI Performance Delta: Large Data Sets

(Large Inputs, Baseline: Host-Only)

Legend: ■ PIM-Only ■ Locality-Aware

Categories: ATF, BFS, PR, SP, WCC, HJ, HG, RP, SC, SVM, GM

SAFARI
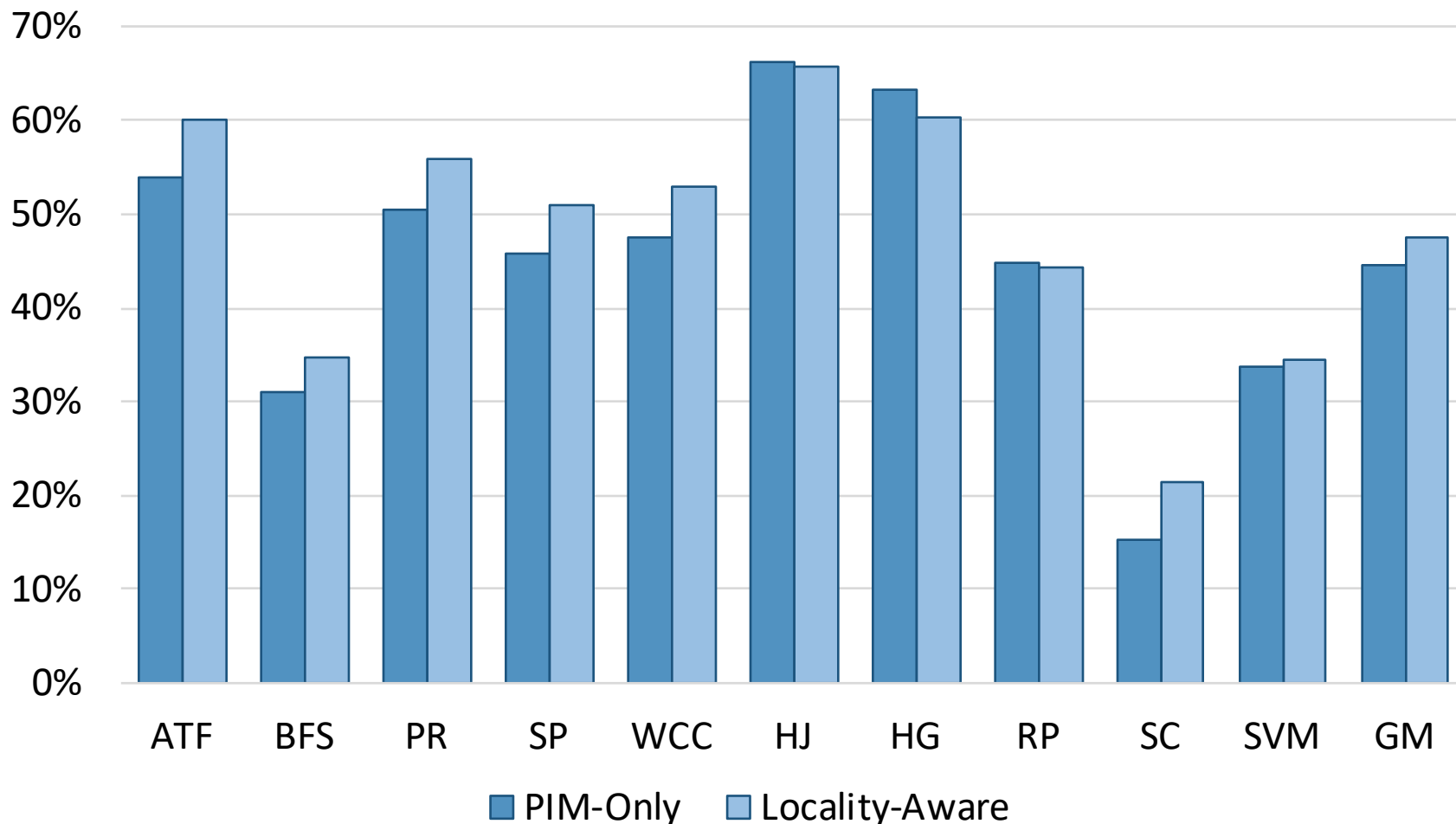
PEI Performance: Large Data Sets

Normalized Amount of Off-chip Transfer

Legend: Host-Only, PIM-Only, Locality-Aware

Categories: ATF, BFS, PR, SP, WCC, HJ, HG, RP, SC, SVM

SAFARI
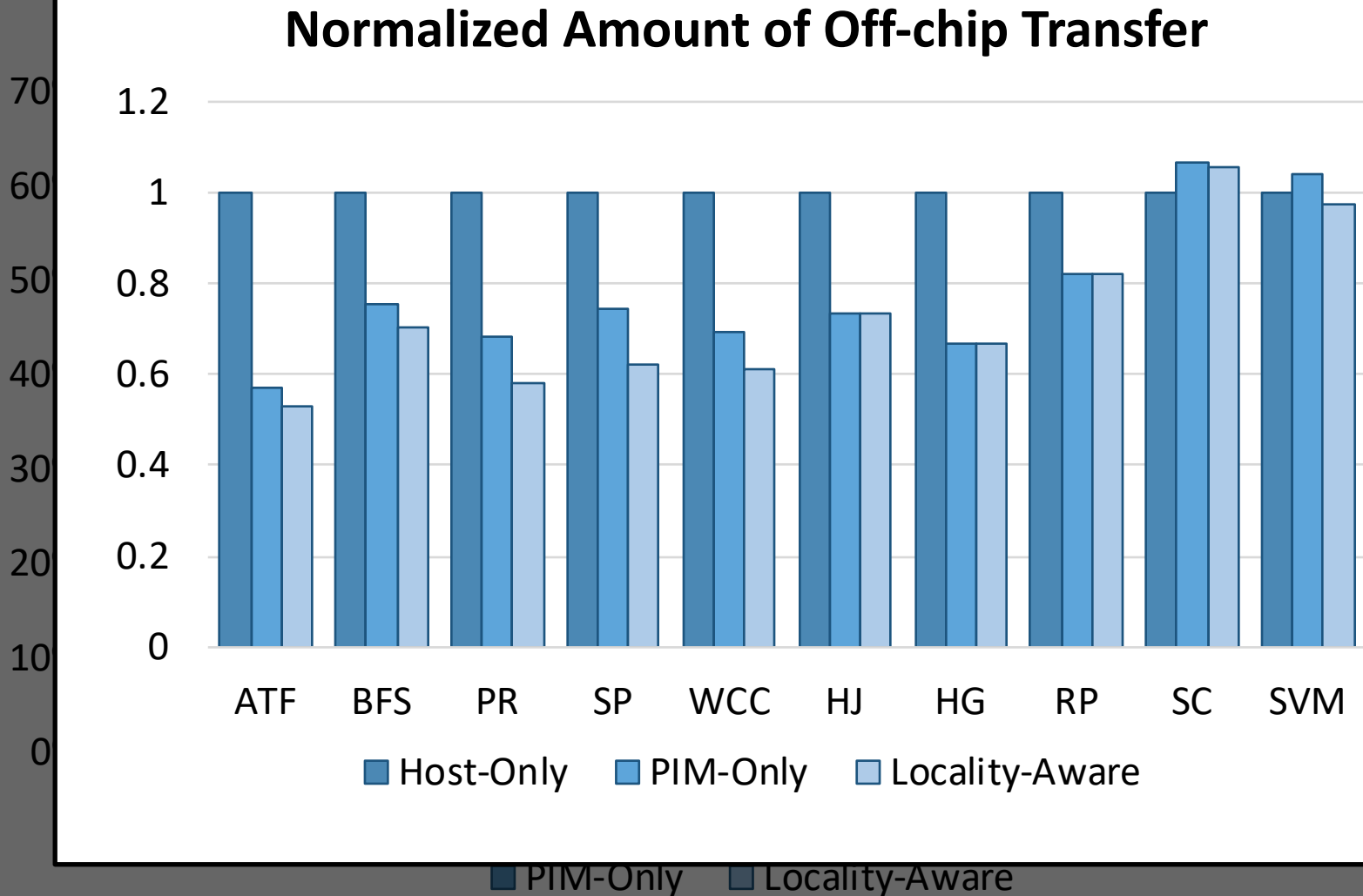
PEI Performance Delta: Small Data Sets

(Small Inputs, Baseline: Host-Only)

Legend: ■ PIM-Only ■ Locality-Aware

Categories: ATF, BFS, PR, SP, WCC, HJ, HG, RP, SC, SVM, GM

SAFARI
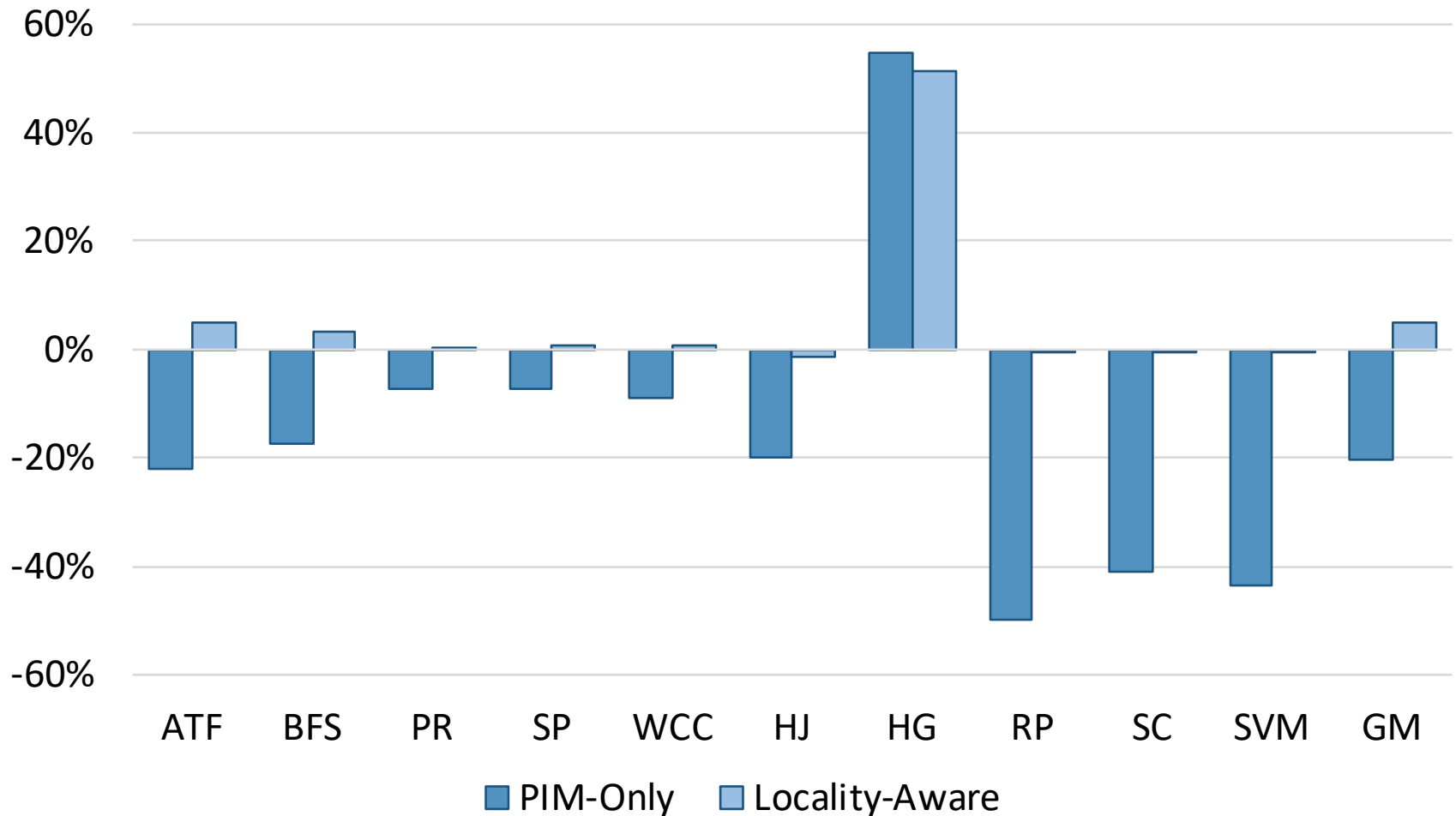
PEI Performance: Small Data Sets

Normalized Amount of Off-chip Transfer

Categories: ATF, BFS, PR, SP, WCC, HJ, HG, RP, SC, SVM

Legend: Host-Only, PIM-Only, Locality-Aware

SAFARI
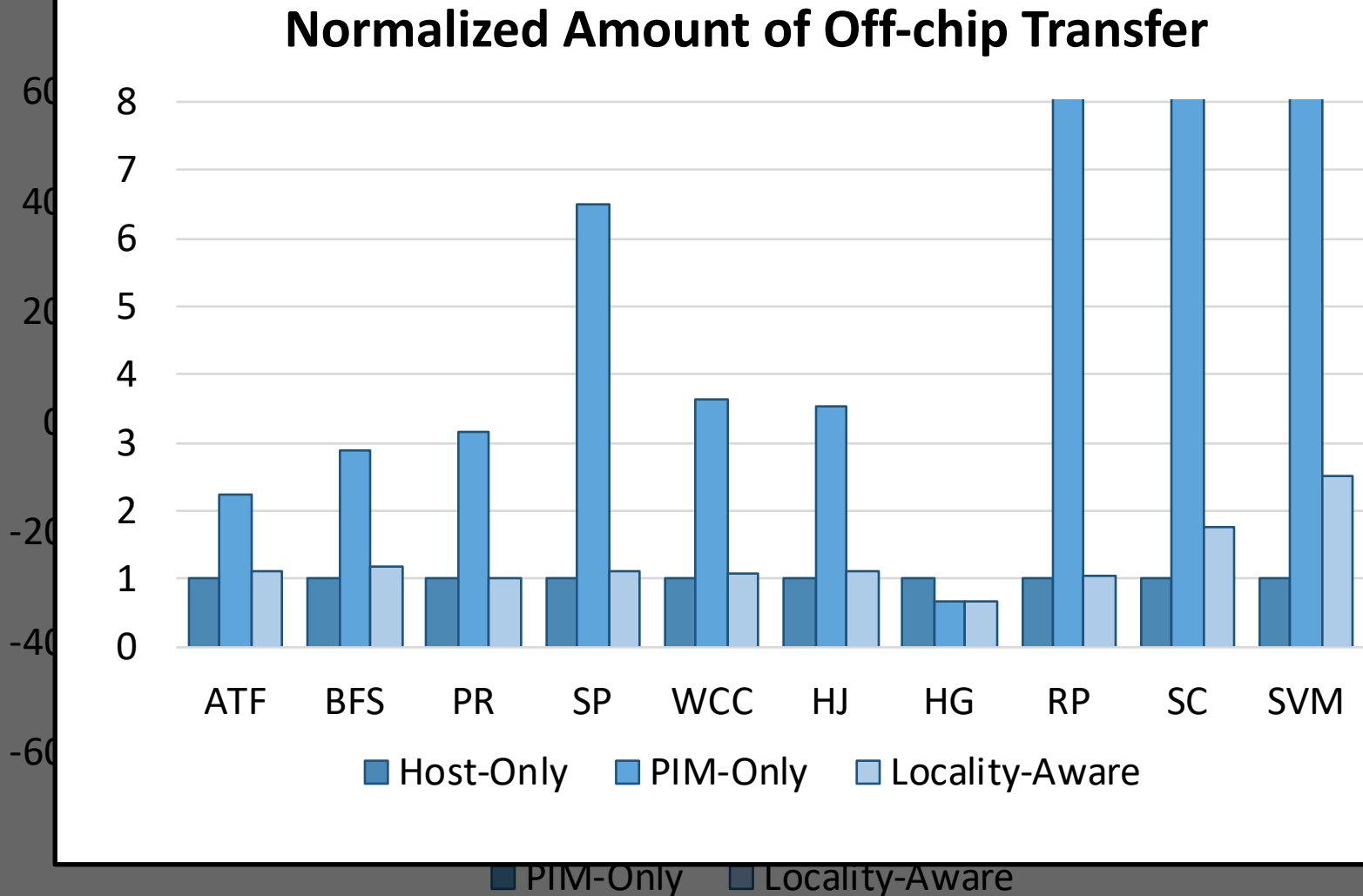
PEI Performance Delta: Medium Data Sets

(Medium Inputs, Baseline: Host-Only)

Legend: PIM-Only, Locality-Aware

Categories: ATF, BFS, PR, SP, WCC, HJ, HG, RP, SC, SVM, GM
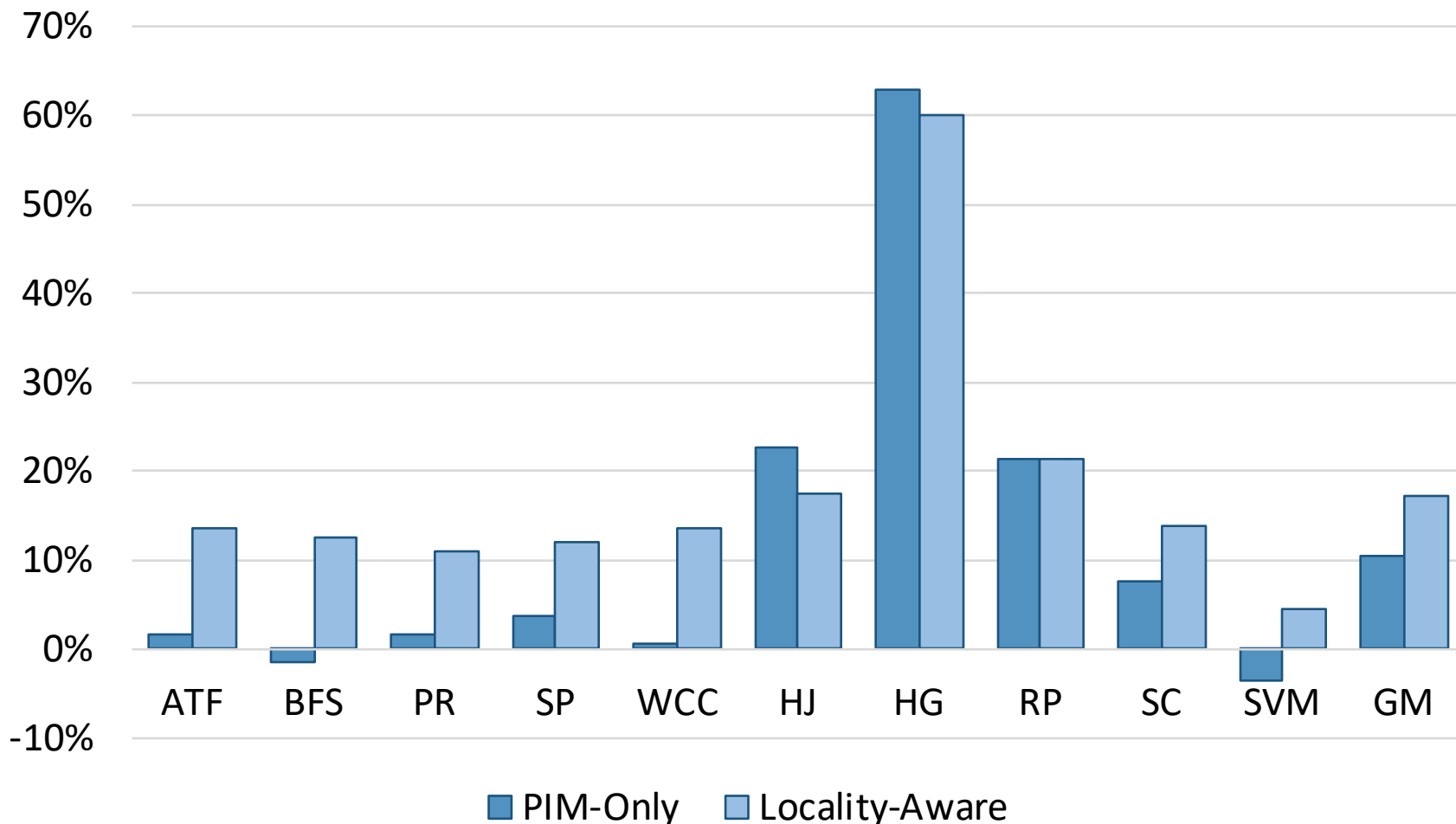
PEI Energy Consumption
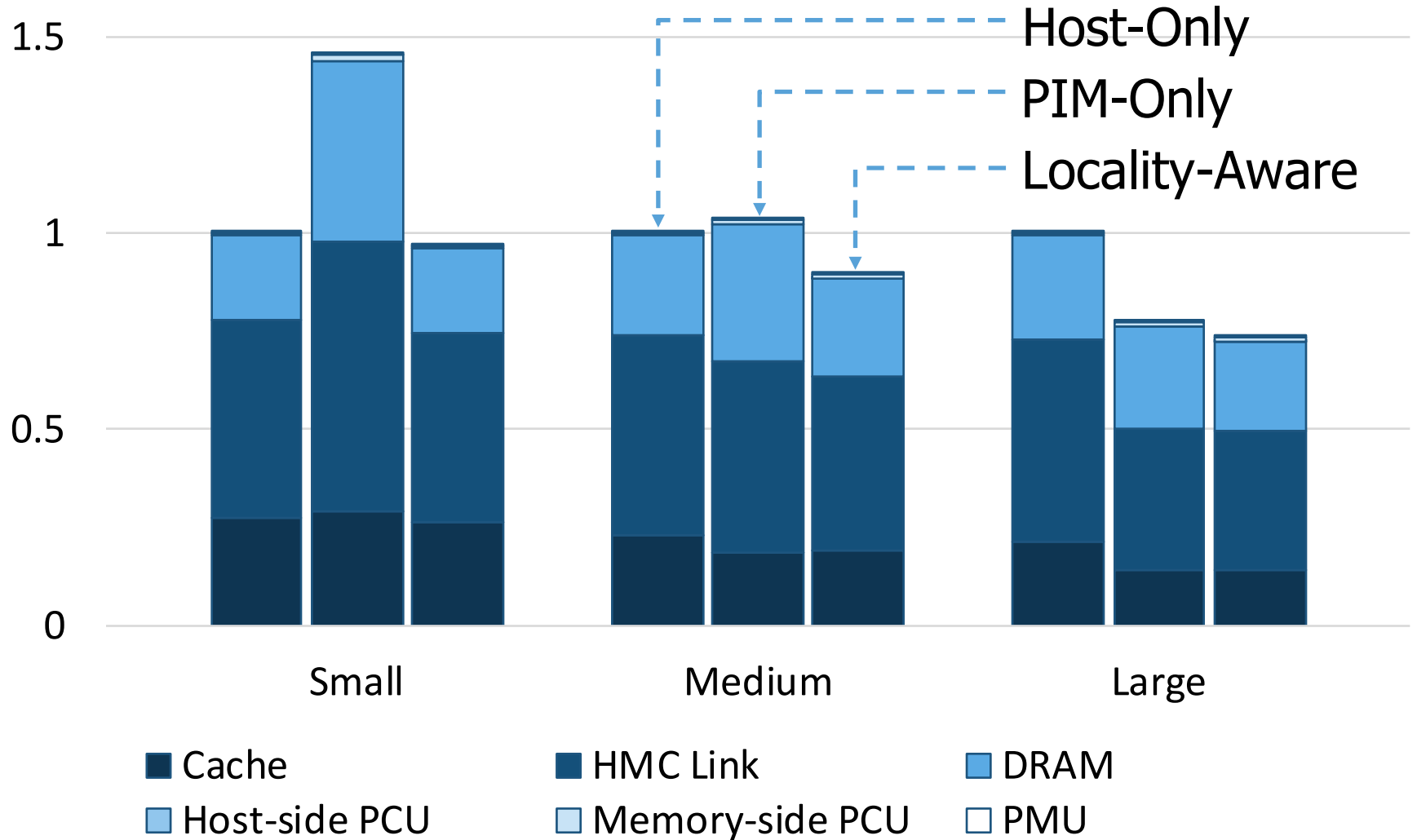
PEI: Advantages & Disadvantages

- **Advantages**

  + Simple and low cost approach to PIM

  + No changes to programming model, virtual memory

  + Dynamically decides where to execute an instruction

- **Disadvantages**

  - Does not take full advantage of PIM potential

    - Single cache block restriction is limiting

SAFARI

Simpler PIM: PIM-Enabled Instructions

- Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoung Choi,
  "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture"
  Proceedings of the 42nd International Symposium on Computer Architecture (**ISCA**), Portland, OR, June 2015.
  [Slides (pdf)] [Lightning Session Slides (pdf)]

PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture

Junwhan Ahn   Sungjoo Yoo   Onur Mutlu[†]   Kiyoung Choi
junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University   [†]Carnegie Mellon University

SAFARI

Automatic Code and Data Mapping

- Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler,
"Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems"
Proceedings of the 43rd International Symposium on Computer Architecture (**ISCA**), Seoul, South Korea, June 2016.
[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]

Transparent Offloading and Mapping (TOM):
Enabling Programmer-Transparent Near-Data Processing in GPU Systems

Kevin Hsieh[‡]   Eiman Ebrahimi[†]   Gwangsun Kim[*]   Niladrish Chatterjee[†]   Mike O'Connor[†]
Nandita Vijaykumar[‡]   Onur Mutlu[§‡]   Stephen W. Keckler[†]

[‡]**Carnegie Mellon University**   [†]**NVIDIA**   [*]**KAIST**   [§]**ETH Zürich**

Automatic Offloading of Critical Code

- Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt,
"Accelerating Dependent Cache Misses with an Enhanced Memory Controller"
Proceedings of the 43rd International Symposium on Computer Architecture (**ISCA**), Seoul, South Korea, June 2016.
[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

Milad Hashemi*, Khubaib†, Eiman Ebrahimi‡, Onur Mutlu§, Yale N. Patt*

*The University of Texas at Austin   †Apple   ‡NVIDIA   §ETH Zürich & Carnegie Mellon University

Automatic Offloading of Prefetch Mechanisms

- Milad Hashemi, Onur Mutlu, and Yale N. Patt,
  "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads"
  Proceedings of the 49th International Symposium on Microarchitecture (**MICRO**), Taipei, Taiwan, October 2016.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pdf)] [Poster (pptx) (pdf)]

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi*, Onur Mutlu[§], Yale N. Patt*

*The University of Texas at Austin   [§]ETH Zürich

SAFARI

Efficient Automatic Data Coherence Support

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
"LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory"
IEEE Computer Architecture Letters (**CAL**), June 2016.

LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory

Amirali Boroumand[†], Saugata Ghose[†], Minesh Patel[†], Hasan Hassan[†§], Brandon Lucia[†],
Kevin Hsieh[†], Krishna T. Malladi[*], Hongzhong Zheng[*], and Onur Mutlu[‡†]

[†] *Carnegie Mellon University*   [*] *Samsung Semiconductor, Inc.*   [§] *TOBB ETÜ*   [‡] *ETH Zürich*

Efficient Automatic Data Coherence Support

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"
Proceedings of the 46th International Symposium on Computer Architecture (**ISCA**), Phoenix, AZ, USA, June 2019.

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand[†]   Saugata Ghose[†]   Minesh Patel[⋆]   Hasan Hassan[⋆]

Brandon Lucia[†]   Rachata Ausavarungnirun[†‡]   Kevin Hsieh[†]

Nastaran Hajinazar[◇†]   Krishna T. Malladi[§]   Hongzhong Zheng[§]   Onur Mutlu[⋆†]

[†]Carnegie Mellon University   [⋆]ETH Zürich   [‡]KMUTNB
[◇]Simon Fraser University   [§]Samsung Semiconductor, Inc.

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures

Fundamentally High-Performance (Data-Centric) Computing Architectures

Computing Architectures with Minimal Data Movement

Sub-Agenda: In-Memory Computation

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
    - Bottom Up: Push from Circuits and Devices
    - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
    - Minimally Changing Memory Chips
    - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion

SAFARI

Eliminating the Adoption Barriers

How to Enable Adoption of Processing in Memory

Barriers to Adoption of PIM

1. Functionality of and applications & software for PIM

2. Ease of programming (interfaces and compiler/HW support)

3. System support: coherence & virtual memory

4. Runtime and compilation systems for adaptive scheduling, data mapping, access/sharing control

5. Infrastructures to assess benefits and feasibility

All can be solved with change of mindset

We Need to Revisit the Entire Stack

Problem
Algorithm
Program/Language
System Software
SW/HW Interface
Micro-architecture
Logic
Devices
Electrons
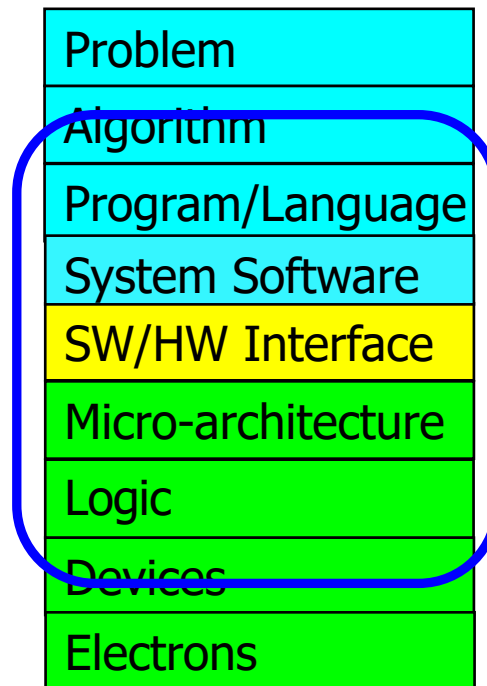
We can get there step by step

PIM Review and Open Problems

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu[a,b], Saugata Ghose[b], Juan Gómez-Luna[a], Rachata Ausavarungnirun[b,c]

[a]ETH Zürich
[b]Carnegie Mellon University
[c]King Mongkut's University of Technology North Bangkok

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"Processing Data Where It Makes Sense: Enabling In-Memory Computation"
Invited paper in Microprocessors and Microsystems (**MICPRO**), June 2019.
[arXiv version]

PIM Review and Open Problems (II)

A Workload and Programming Ease Driven Perspective of Processing-in-Memory

Saugata Ghose[†]   Amirali Boroumand[†]   Jeremie S. Kim[†§]   Juan Gómez-Luna[§]   Onur Mutlu[§†]

[†]Carnegie Mellon University   [§]ETH Zürich

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,
"Processing-in-Memory: A Workload-Driven Perspective"
Invited Article in IBM Journal of Research & Development, Special Issue on Hardware for Artificial Intelligence, to appear in November 2019.
[Preliminary arXiv version]

Key Challenge 1: Code Mapping

- **Challenge 1:** Which operations should be executed in memory vs. in CPU?

```
__global__
void applyScaleFactorsKernel( uint8_T * const out,
    uint8_T const * const in, const double *factor,
    size_t const numRows, size_t const numCols )
{
    // Work out which pixel we are working on.
    const int rowIdx = blockIdx.x * blockDim.x + threadIdx.x;
    const int colIdx = blockIdx.y;
    const int sliceIdx = threadIdx.z;

    // Check this thread isn't off the image
    if( rowIdx >= numRows ) return;

    // Compute the index of my element
    size_t linearIdx = rowIdx + colIdx*numRows +
        sliceIdx*numRows*numCols;
```

3D-stacked memory (memory stack)

SM (Streaming Multiprocessor)

Logic layer

Main GPU

Logic layer SM

Crossbar switch

Vault Ctrl .... **Vault Ctrl**

Key Challenge 2: Data Mapping

- **Challenge 2:** How should data be mapped to different 3D memory stacks?

3D-stacked memory (memory stack)

SM (Streaming Multiprocessor)

Logic layer

Main GPU

Logic layer SM

Crossbar switch

Vault Ctrl .... **Vault Ctrl**
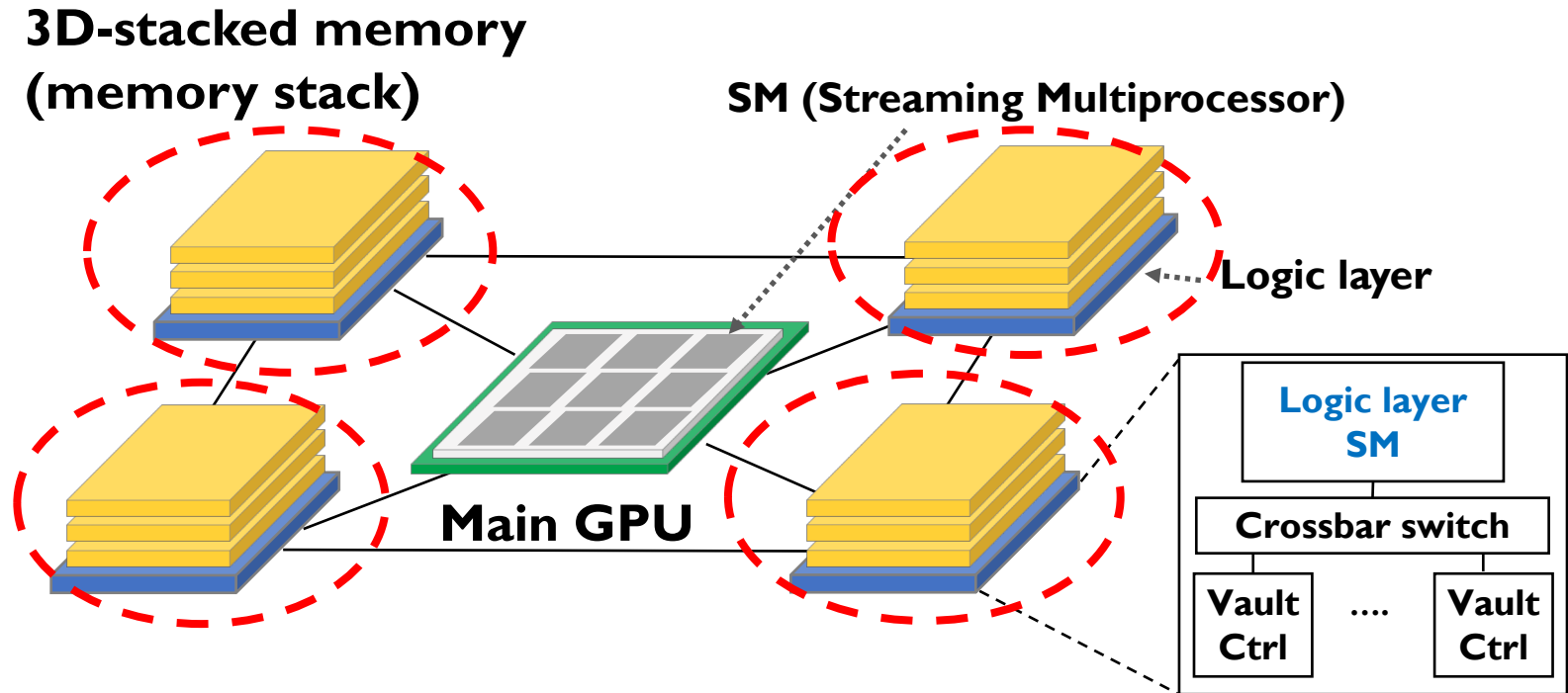
How to Do the Code and Data Mapping?

- Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler,
"Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems"
*Proceedings of the 43rd International Symposium on Computer Architecture (**ISCA**)*, Seoul, South Korea, June 2016.
[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]

Transparent Offloading and Mapping (TOM):
Enabling Programmer-Transparent Near-Data Processing in GPU Systems

Kevin Hsieh[‡]   Eiman Ebrahimi[†]   Gwangsun Kim[*]   Niladrish Chatterjee[†]   Mike O'Connor[†]
Nandita Vijaykumar[‡]   Onur Mutlu[§‡]   Stephen W. Keckler[†]

[‡]**Carnegie Mellon University**   [†]**NVIDIA**   [*]**KAIST**   [§]**ETH Zürich**

How to Schedule Code? (I)

- Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, and Chita R. Das,
  "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities"
  Proceedings of the 25th International Conference on Parallel Architectures and Compilation Techniques (**PACT**), Haifa, Israel, September 2016.

Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities

Ashutosh Pattnaik[1]   Xulong Tang[1]   Adwait Jog[2]   Onur Kayıran[3]
Asit K. Mishra[4]   Mahmut T. Kandemir[1]   Onur Mutlu[5,6]   Chita R. Das[1]

[1]Pennsylvania State University   [2]College of William and Mary
[3]Advanced Micro Devices, Inc.   [4]Intel Labs   [5]ETH Zürich   [6]Carnegie Mellon University

How to Schedule Code? (II)

- Milad Hashemi, Khubaib, Eiman Ebrahimi, Onur Mutlu, and Yale N. Patt,
"Accelerating Dependent Cache Misses with an Enhanced Memory Controller"
Proceedings of the 43rd International Symposium on Computer Architecture (**ISCA**), Seoul, South Korea, June 2016.
[Slides (pptx) (pdf)]
[Lightning Session Slides (pptx) (pdf)]

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

Milad Hashemi[*], Khubaib[†], Eiman Ebrahimi[‡], Onur Mutlu[§], Yale N. Patt[*]

[*]The University of Texas at Austin   [†]Apple   [‡]NVIDIA   [§]ETH Zürich & Carnegie Mellon University

How to Schedule Code? (III)

- Milad Hashemi, Onur Mutlu, and Yale N. Patt,
  "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads"
  Proceedings of the 49th International Symposium on Microarchitecture (**MICRO**), Taipei, Taiwan, October 2016.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pdf)] [Poster (pptx) (pdf)]

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

Milad Hashemi*, Onur Mutlu§, Yale N. Patt*

*The University of Texas at Austin   §ETH Zürich

Challenge: Coherence for Hybrid CPU-PIM Apps
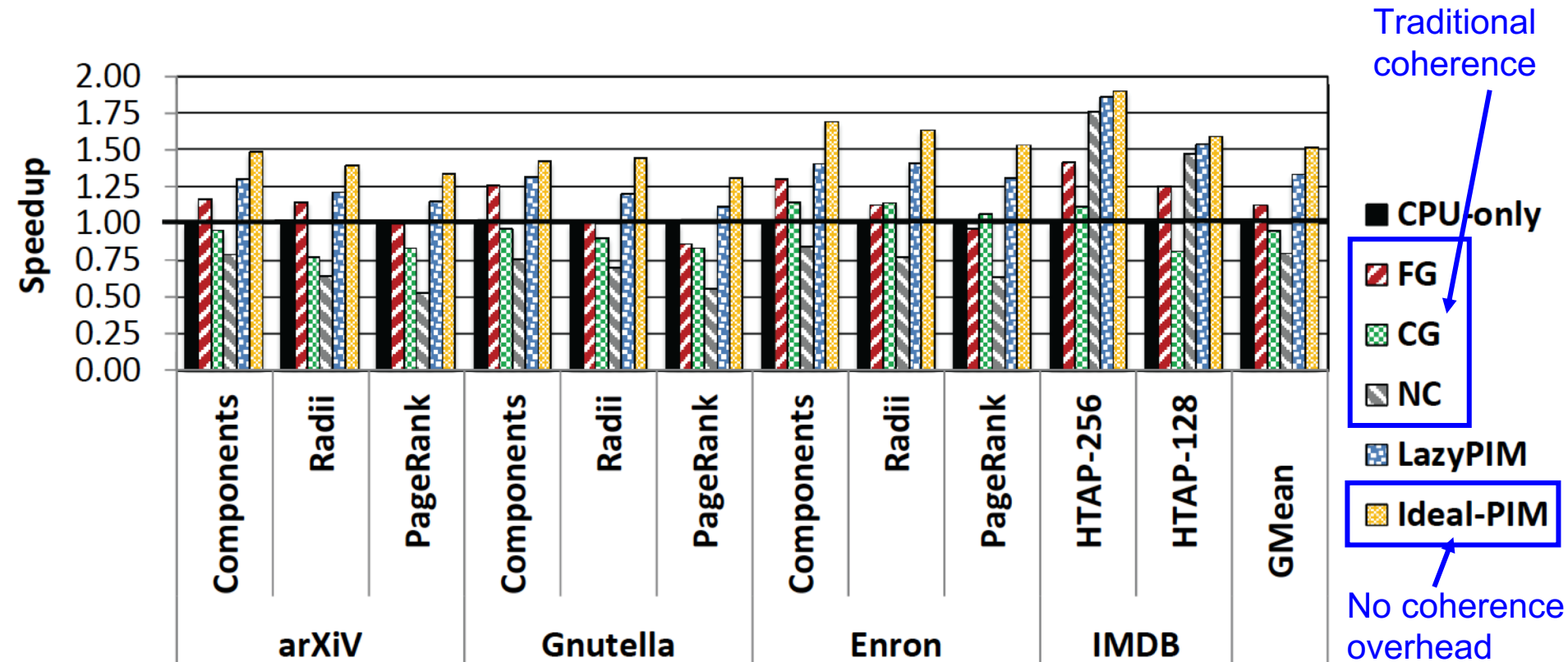
How to Maintain Coherence? (I)

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
  "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory"
  IEEE Computer Architecture Letters (**CAL**), June 2016.

LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory

Amirali Boroumand[†], Saugata Ghose[†], Minesh Patel[†], Hasan Hassan[†§], Brandon Lucia[†],
Kevin Hsieh[†], Krishna T. Malladi[*], Hongzhong Zheng[*], and Onur Mutlu[‡†]

[†]*Carnegie Mellon University*   [*]*Samsung Semiconductor, Inc.*   [§]*TOBB ETÜ*   [‡]*ETH Zürich*

How to Maintain Coherence? (II)

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"
Proceedings of the 46th International Symposium on Computer Architecture (**ISCA**), Phoenix, AZ, USA, June 2019.

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand[†] Saugata Ghose[†] Minesh Patel[★] Hasan Hassan[★]
Brandon Lucia[†] Rachata Ausavarungnirun[†‡] Kevin Hsieh[†]
Nastaran Hajinazar[◇†] Krishna T. Malladi[§] Hongzhong Zheng[§] Onur Mutlu[★†]

[†]Carnegie Mellon University [★]ETH Zürich [‡]KMUTNB
[◇]Simon Fraser University [§]Samsung Semiconductor, Inc.

CoNDA:
Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand

**Saugata Ghose, Minesh Patel, Hasan Hassan,
Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh,
Nastaran Hajinazar, Krishna Malladi, Hongzhong Zheng,
Onur Mutlu**

Specialized Accelerators

Specialized accelerators are now everywhere!
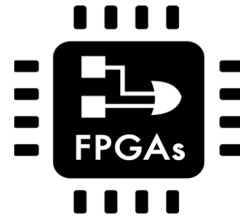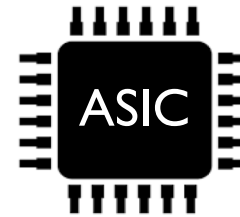
GPU　　**FPGA**　　**ASIC**

Recent advancement in 3D-stacked technology enabled Near-Data Accelerators (NDA)

Coherence For NDAs

(1) Large cost of off-chip communication

CPU ↔ L1 ↔ L2

DRAM

Compute Unit

NDA

(2) NDA applications generate a large amount of off-chip data movement
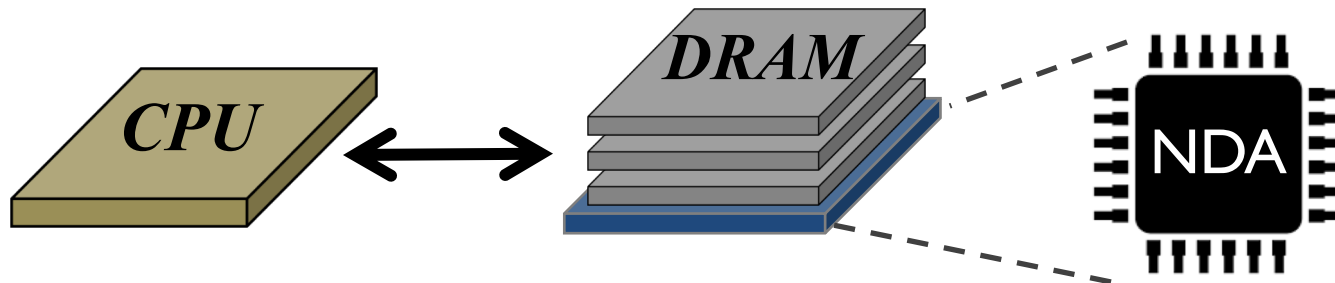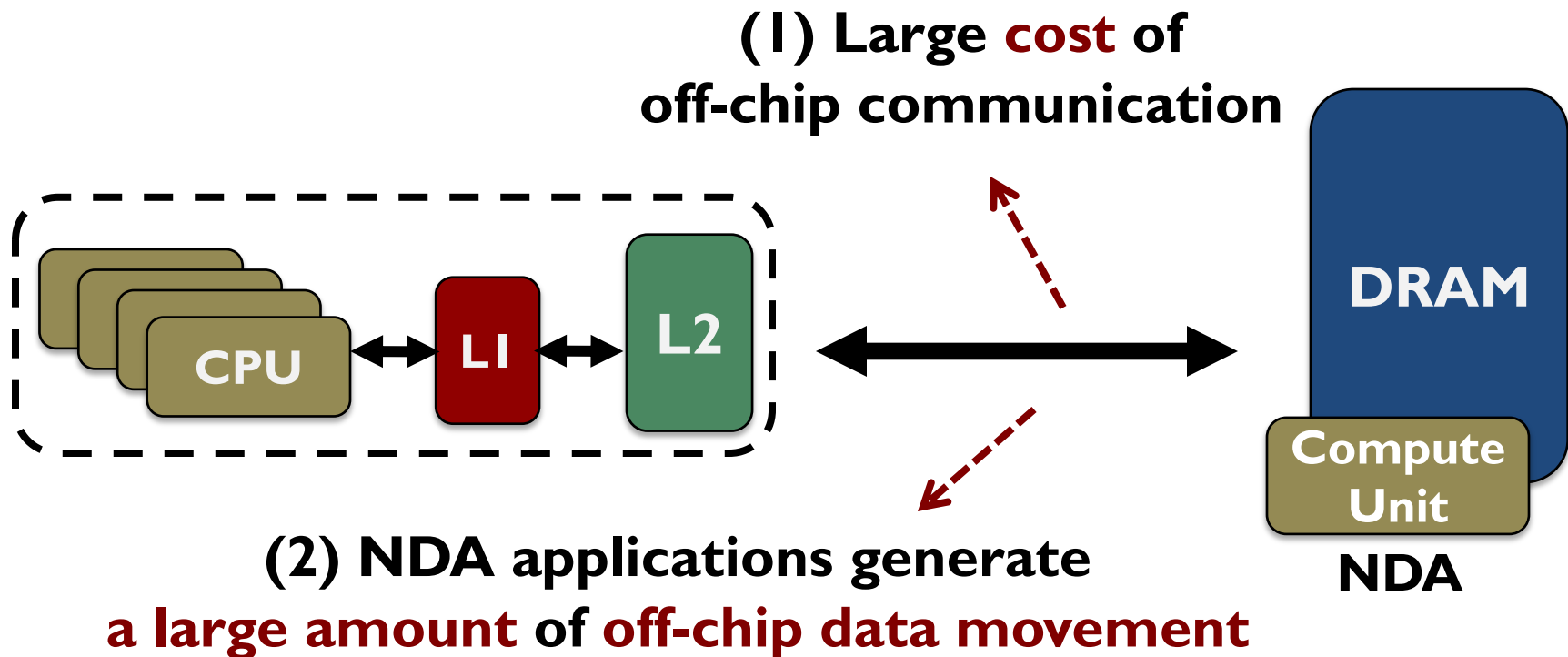
It is impractical to use traditional coherence protocols

Existing Coherence Mechanisms

We extensively study existing NDA coherence mechanisms and make three key observations:

1   **These mechanisms eliminate a significant portion of NDA's benefits**

2   **The majority of off-chip coherence traffic generated by these mechanisms is unnecessary**

3   **Much of the off-chip traffic can be eliminated if the coherence mechanism has insight into the memory accesses**

An Optimistic Approach

We find that **an optimistic approach** to coherence can address the **challenges** related to NDA coherence

1 **Gain insights _before_ any coherence checks happens**

2 **Perform only the necessary coherence requests**

CoNDA

We propose **CoNDA,** a mechanism that uses **optimistic NDA execution** to avoid **unnecessary coherence traffic**

Time

CPU   **NDA**

CPU Thread Execution

Offload NDA kernel

Concurrent CPU + NDA Execution

No Coherence Request

Optimistic execution

Signature

Signature

Send signatures

Coherence Resolution

Commit or Re-execute

CoNDA

We propose **CoNDA,** a mechanism that uses **optimistic NDA execution** to avoid **unnecessary coherence traffic**

CPU　　　　　　　　　　　　**NDA**

Time

CPU Thread Execution

Offload NDA kernel

Concurrent CPU + NDA Execution

No Coherence Request

Optimistic execution

Signature　　　　　　　Signature

Send signatures

CoNDA comes within 10.4% and 4.4% of performance and energy of an ideal NDA coherence mechanism

it or Re-execute

CoNDA:
Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand

**Saugata Ghose, Minesh Patel, Hasan Hassan,
Brandon Lucia, Rachata Ausavarungnirun, Kevin Hsieh,
Nastaran Hajinazar, Krishna Malladi, Hongzhong Zheng,
Onur Mutlu**

How to Maintain Coherence? (II)

- Amirali Boroumand, Saugata Ghose, Minesh Patel, Hasan Hassan, Brandon Lucia, Kevin Hsieh, Krishna T. Malladi, Hongzhong Zheng, and Onur Mutlu,
"CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators"
Proceedings of the 46th International Symposium on Computer Architecture (**ISCA**), Phoenix, AZ, USA, June 2019.

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand[†]   Saugata Ghose[†]   Minesh Patel[⋆]   Hasan Hassan[⋆]
Brandon Lucia[†]   Rachata Ausavarungnirun[†‡]   Kevin Hsieh[†]
Nastaran Hajinazar[◇†]   Krishna T. Malladi[§]   Hongzhong Zheng[§]   Onur Mutlu[⋆†]

[†]Carnegie Mellon University   [⋆]ETH Zürich   [‡]KMUTNB
[◇]Simon Fraser University   [§]Samsung Semiconductor, Inc.

How to Support Virtual Memory?

- Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu,
"Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation"
Proceedings of the 34th IEEE International Conference on Computer Design (**ICCD**), Phoenix, AZ, USA, October 2016.

Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation

Kevin Hsieh[†]   Samira Khan[‡]   Nandita Vijaykumar[†]
Kevin K. Chang[†]   Amirali Boroumand[†]   Saugata Ghose[†]   Onur Mutlu[§†]
[†]*Carnegie Mellon University*   [‡]*University of Virginia*   [§]*ETH Zürich*

How to Design Data Structures for PIM?

- Zhiyu Liu, Irina Calciu, Maurice Herlihy, and Onur Mutlu,
  "Concurrent Data Structures for Near-Memory Computing"
  Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (**SPAA**), Washington, DC, USA, July 2017.
  [Slides (pptx) (pdf)]

Concurrent Data Structures for Near-Memory Computing

Zhiyu Liu
Computer Science Department
Brown University
zhiyu_liu@brown.edu

Irina Calciu
VMware Research Group
icalciu@vmware.com

Maurice Herlihy
Computer Science Department
Brown University
mph@cs.brown.edu

Onur Mutlu
Computer Science Department
ETH Zürich
onur.mutlu@inf.ethz.ch

SAFARI

Simulation Infrastructures for PIM

- **Ramulator** extended for PIM
  - Flexible and extensible DRAM simulator
  - Can model many different memory standards and proposals
  - Kim+, "**Ramulator: A Flexible and Extensible DRAM Simulator**", IEEE CAL 2015.
  - https://github.com/CMU-SAFARI/ramulator-pim
  - https://github.com/CMU-SAFARI/ramulator
  - [Source Code for Ramulator-PIM]

Ramulator: A Fast and Extensible DRAM Simulator

Yoongu Kim[1]   Weikun Yang[1,2]   Onur Mutlu[1]
[1]Carnegie Mellon University   [2]Peking University

Performance & Energy Models for PIM

- Gagandeep Singh, Juan Gomez-Luna, Giovanni Mariani, Geraldo F. Oliveira, Stefano Corda, Sander Stujik, Onur Mutlu, and Henk Corporaal, **"NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning"** *Proceedings of the 56th Design Automation Conference* (**DAC**), Las Vegas, NV, USA, June 2019.
[Slides (pptx) (pdf)]
[Poster (pptx) (pdf)]
[Source Code for Ramulator-PIM]

NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning

Gagandeep Singh[a,c]   Juan Gómez-Luna[b]   Giovanni Mariani[c]   Geraldo F. Oliveira[b]

Stefano Corda[a,c]   Sander Stuijk[a]   Onur Mutlu[b]   Henk Corporaal[a]

[a]Eindhoven University of Technology   [b]ETH Zürich   [c]IBM Research - Zurich

An FPGA-based Test-bed for PIM?

- Hasan Hassan et al., **SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies** HPCA 2017.

- **Flexible**
- **Easy to Use (C++ API)**
- **Open-source**

  github.com/CMU-SAFARI/SoftMC

SAFARI
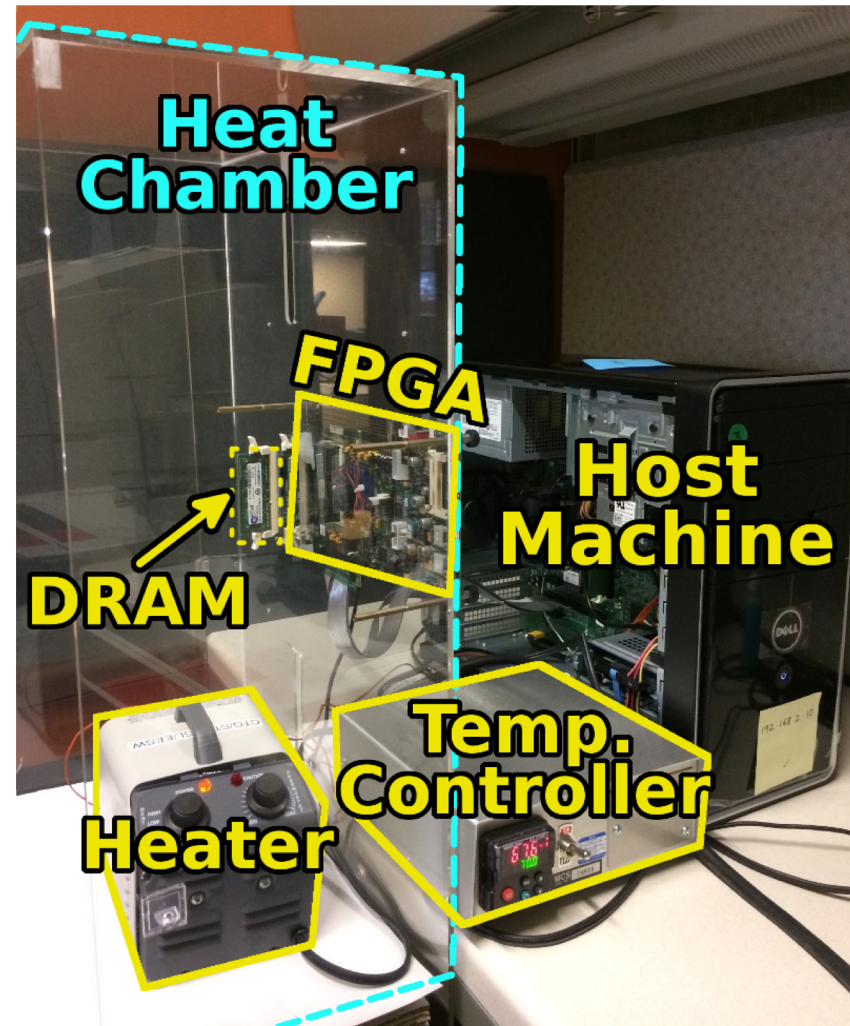
Simulation Infrastructures for PIM (in SSDs)

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu,
  "MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices"
  Proceedings of the 16th USENIX Conference on File and Storage Technologies (**FAST**), Oakland, CA, USA, February 2018.
  [Slides (pptx) (pdf)]
  [Source Code]

MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices

Arash Tavakkol[†], Juan Gómez-Luna[†], Mohammad Sadrosadati[†], Saugata Ghose[‡], Onur Mutlu[†‡]

[†]*ETH Zürich*   [‡]*Carnegie Mellon University*

New Applications and Use Cases for PIM

- Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu,
"GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies"
BMC Genomics, 2018.
Proceedings of the 16th Asia Pacific Bioinformatics Conference (**APBC**), Yokohama, Japan, January 2018.
arxiv.org Version (pdf)

GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies

Jeremie S. Kim[1,6*], Damla Senol Cali[1], Hongyi Xin[2], Donghyuk Lee[3], Saugata Ghose[1], Mohammed Alser[4], Hasan Hassan[6], Oguz Ergin[5], Can Alkan[4*] and Onur Mutlu[6,1*]

Genome Read In-Memory (GRIM) Filter:

Fast Seed Location Filtering in DNA Read Mapping
using Processing-in-Memory Technologies

Jeremie Kim,

Damla Senol, Hongyi Xin, Donghyuk Lee,
Saugata Ghose, Mohammed Alser, Hasan Hassan,
Oguz Ergin, Can Alkan, and Onur Mutlu

Executive Summary

- **Genome Read Mapping** is a very important problem and is the first step in many types of genomic analysis
  - Could lead to improved health care, medicine, quality of life

- Read mapping is an **approximate string matching** problem
  - Find the best fit of 100 character strings into a 3 billion character dictionary
  - **Alignment** is currently the best method for determining the similarity between two strings, but is **very expensive**

- We propose an in-memory processing algorithm **GRIM-Filter** for accelerating read mapping, by reducing the number of required alignments

- We implement GRIM-Filter using **in-memory processing** within **3D-stacked memory** and show up to **3.7x speedup**.

Google Workloads
for Consumer Devices:
Mitigating Data Movement Bottlenecks

Amirali Boroumand

**Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun,
Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela,
Allan Knies, Parthasarathy Ranganathan, Onur Mutlu**

SAFARI　　**Carnegie Mellon**　　**Google**

SAMSUNG　　**SEOUL NATIONAL UNIVERSITY**　　**ETH** *Zürich*

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu[a,b], Saugata Ghose[b], Juan Gómez-Luna[a], Rachata Ausavarungnirun[b,c]

[a]*ETH Zürich*
[b]*Carnegie Mellon University*
[c]*King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"Processing Data Where It Makes Sense: Enabling In-Memory Computation"
*Invited paper in Microprocessors and Microsystems (**MICPRO**), June 2019.*
[arXiv version]

https://arxiv.org/pdf/1903.03988.pdf

PIM Review and Open Problems (II)

A Workload and Programming Ease Driven Perspective of Processing-in-Memory

Saugata Ghose[†]   Amirali Boroumand[†]   Jeremie S. Kim[†§]   Juan Gómez-Luna[§]   Onur Mutlu[§†]

[†]*Carnegie Mellon University*   [§]*ETH Zürich*

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,
"Processing-in-Memory: A Workload-Driven Perspective"
Invited Article in IBM Journal of Research & Development, Special Issue on Hardware for Artificial Intelligence, to appear in November 2019.
[Preliminary arXiv version]

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures

SAFARI

Fundamentally High-Performance (Data-Centric) Computing Architectures

SAFARI

Computing Architectures with Minimal Data Movement

One Important Takeaway

Main Memory Needs
Intelligent Controllers

Enabling the Paradigm Shift

Recall: Computer Architecture Today

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)

- You can invent new paradigms for computation, communication, and storage

- Recommended book: Thomas Kuhn, "The Structure of Scientific Revolutions" (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions re-examined

Recall: Computer Architecture Today

- You can revolutionize the way computers are built, if you understand both the hardware and the software (and change each accordingly)

- You can in_____ms
communic_____

- Recomme_____as_____ure of
Scientific _____:)
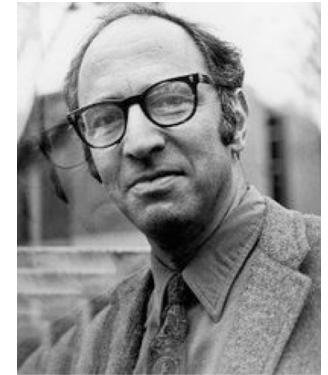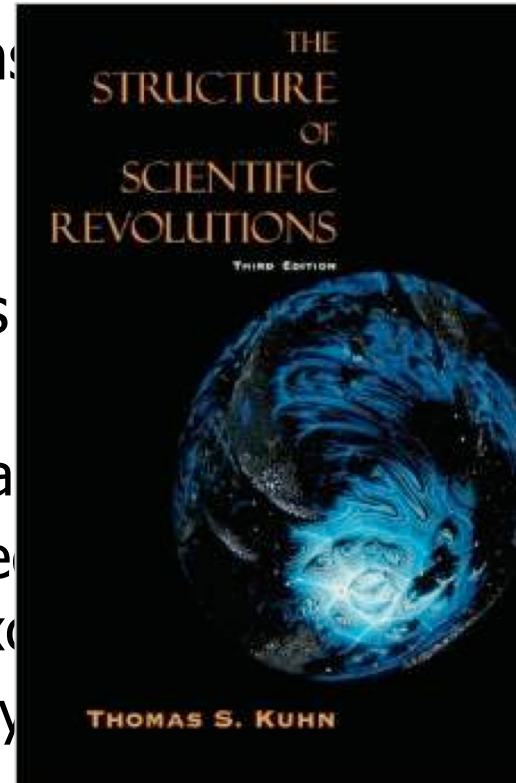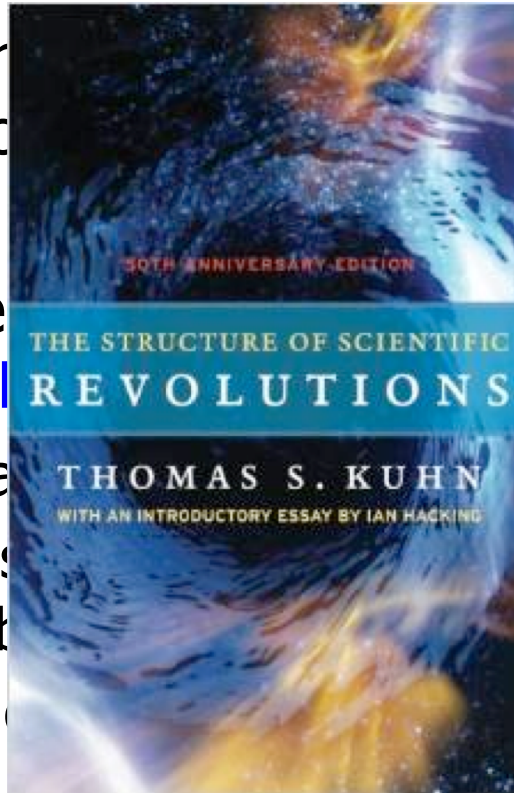
  - Pre-para_____ea_____ield
  - Normal _____he_____improve
    things (b_____exc_____anomalies
  - Revoluti_____rly_____examined

UPMEM Processing-in-DRAM Engine (2019)

- Processing in DRAM Engine
- Includes **standard DIMM modules**, with a **large number of DPU processors** combined with DRAM chips.

- Replaces **standard** DIMMs
  - DDR4 R-DIMM modules
    - 8GB+128 DPUs (16 PIM chips)
    - Standard 2x-nm DRAM process
  - **Large amounts of** compute & memory bandwidth
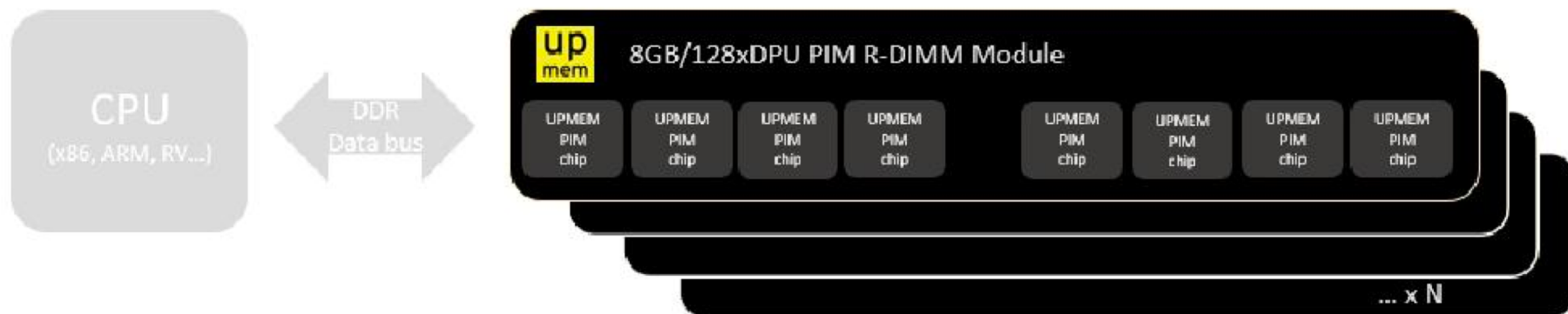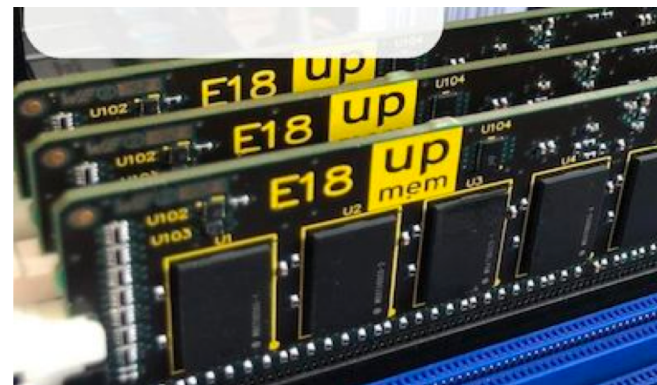
Sub-Agenda: In-Memory Computation

- Major Trends Affecting Main Memory
- The Need for Intelligent Memory Controllers
  - Bottom Up: Push from Circuits and Devices
  - Top Down: Pull from Systems and Applications
- Processing in Memory: Two Directions
  - Minimally Changing Memory Chips
  - Exploiting 3D-Stacked Memory
- How to Enable Adoption of Processing in Memory
- Conclusion

SAFARI

Maslow's Hierarchy of Needs, A Third Time

Maslow, "A Theory of Human Motivation,"
Psychological Review, 1943.

Maslow, "Motivation and Personality,"
Book, 1954-1970.

Source: https://www.simplypsychology.org/maslow.html
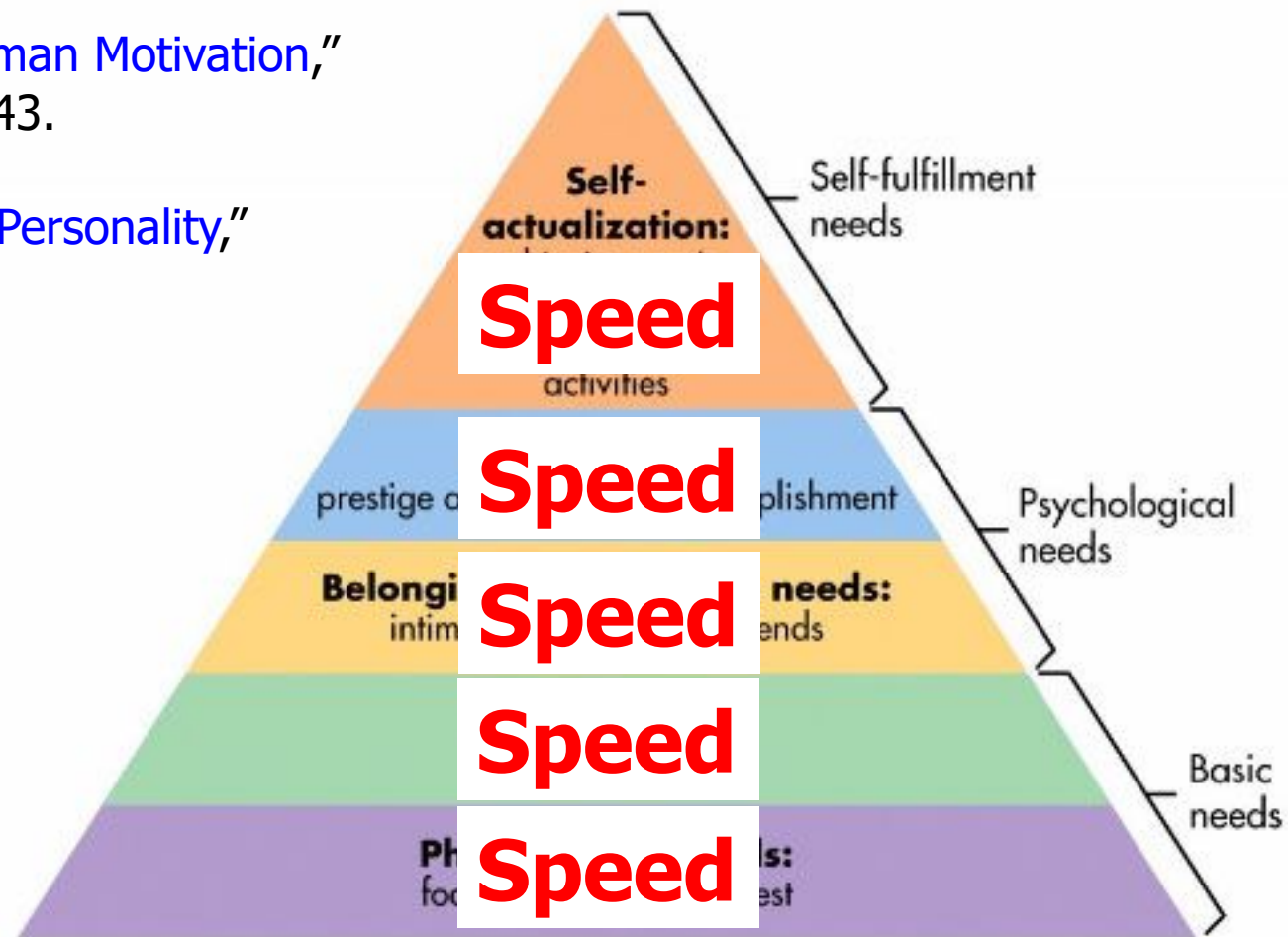
Fundamentally High-Performance (Data-Centric) Computing Architectures

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures

Fundamentally Low-Latency (Data-Centric) Computing Architectures

Computing Architectures with Minimal Data Movement

PIM: Concluding Remarks

A Quote from A Famous Architect

- "architecture [...] based upon principle, and not upon precedent"
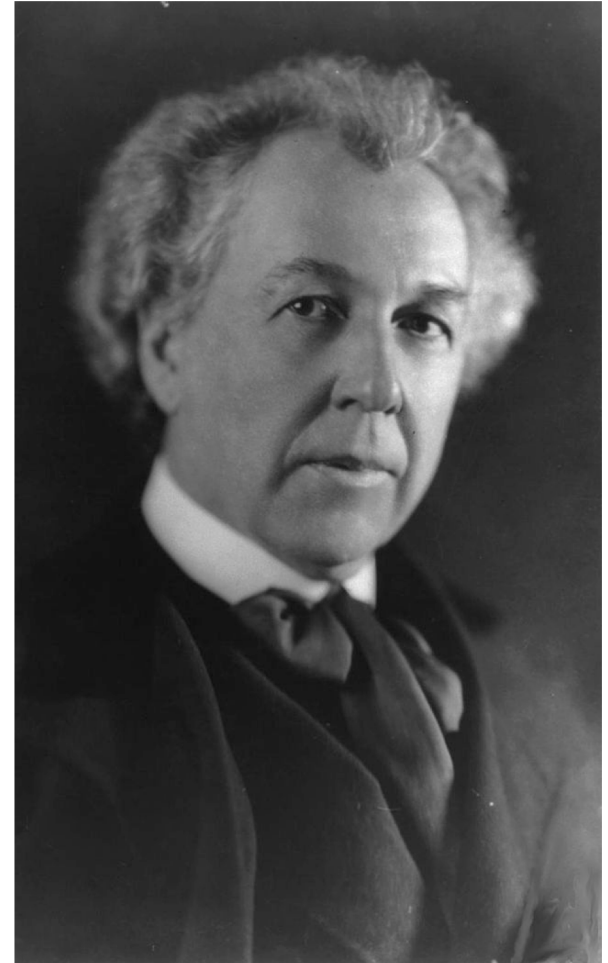
Precedent-Based Design?

- "architecture […] based upon principle, and not upon precedent"

Principled Design

- "architecture […] based upon principle, and not upon precedent"

www.GreatBuildings.com

The Overarching Principle

Organic architecture

From Wikipedia, the free encyclopedia

Organic architecture is a philosophy of architecture which promotes harmony between human habitation and the natural world through design approaches so sympathetic and well integrated with its site, that buildings, furnishings, and surroundings become part of a unified, interrelated composition.

A well-known example of organic architecture is Fallingwater, the residence Frank Lloyd Wright designed for the Kaufmann family in rural Pennsylvania. Wright had many choices to locate a home on this large site, but chose to place the home directly over the waterfall and creek creating a close, yet noisy dialog with the rushing water and the steep site. The horizontal striations of stone masonry with daring cantilevers of colored beige concrete blend with native rock outcroppings and the wooded environment.

Another Example: Precedent-Based Design

Principled Design

Another Principled Design

Source: http://www.arcspace.com/exhibitions/unsorted/santiago-calatrava/

118

Another Principled Design

Principle Applied to Another Structure

The Overarching Principle

Zoomorphic architecture

From Wikipedia, the free encyclopedia

Zoomorphic architecture is the practice of using animal forms as the inspirational basis and blueprint for architectural design. "While animal forms have always played a role adding some of the deepest layers of meaning in architecture, it is now becoming evident that a new strand of biomorphism is emerging where the meaning derives not from any specific representation but from a more general allusion to biological processes."[1]

Some well-known examples of Zoomorphic architecture can be found in the TWA Flight Center building in New York City, by Eero Saarinen, or the Milwaukee Art Museum by Santiago Calatrava, both inspired by the form of a bird's wings.[3]

Overarching Principle for Computing?

Concluding Remarks

- It is time to design principled system architectures to solve the memory problem

- Design complete systems to be balanced, high-performance, and energy-efficient, i.e., data-centric (or memory-centric)

- Enable computation capability inside and close to memory

- This can
  - Lead to **orders-of-magnitude** improvements
  - **Enable new applications & computing platforms**
  - **Enable better understanding of nature**
  - **…**

The Future of Processing in Memory is Bright

- **Regardless of challenges**
  - in underlying technology and overlying problems/requirements

Can enable:

- Orders of magnitude improvements

- New applications and computing systems

| Problem |
| Algorithm |
| Program/Language |
| System Software |
| SW/HW Interface |
| Micro-architecture |
| Logic |
| Devices |
| Electrons |
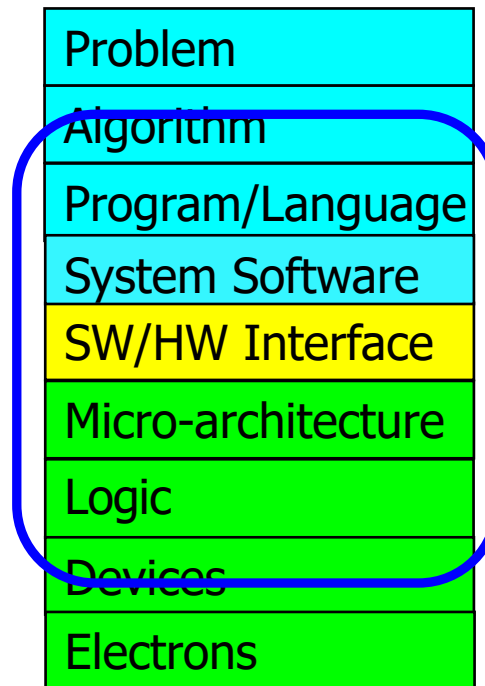
Yet, we have to

- Think across the stack

- Design enabling systems

We Need to Revisit the Entire Stack

Problem
Algorithm
Program/Language
System Software
SW/HW Interface
Micro-architecture
Logic
Devices
Electrons
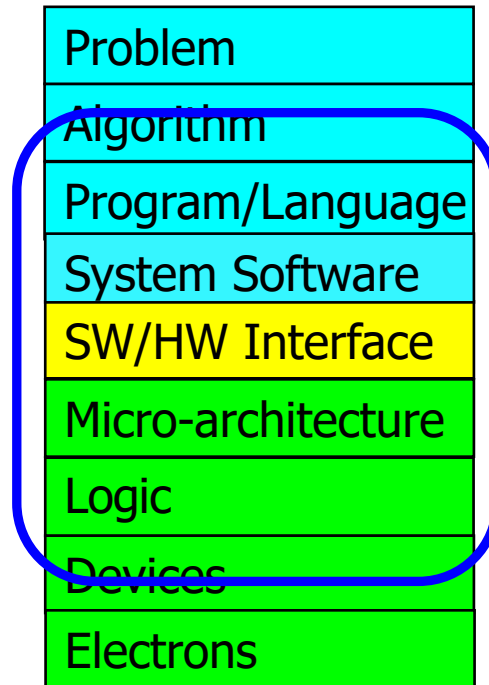
We can get there step by step

If In Doubt, See Other Doubtful Technologies

- A very "doubtful" emerging technology
  - for at least two decades

INVITED PAPER

Proceedings of the IEEE, Sept. 2017

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD's reliability and lifetime.

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

https://arxiv.org/pdf/1706.08642

Flash Memory Timeline

Flash Memory Timeline

PIM Review and Open Problems

Processing Data Where It Makes Sense: Enabling In-Memory Computation

Onur Mutlu[a,b], Saugata Ghose[b], Juan Gómez-Luna[a], Rachata Ausavarungnirun[b,c]

[a]ETH Zürich
[b]Carnegie Mellon University
[c]King Mongkut's University of Technology North Bangkok

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"Processing Data Where It Makes Sense: Enabling In-Memory Computation"
Invited paper in Microprocessors and Microsystems (**MICPRO**), June 2019.
[arXiv version]

PIM Review and Open Problems (II)

A Workload and Programming Ease Driven Perspective of Processing-in-Memory

Saugata Ghose[†]   Amirali Boroumand[†]   Jeremie S. Kim[†§]   Juan Gómez-Luna[§]   Onur Mutlu[§†]

[†]*Carnegie Mellon University*   [§]*ETH Zürich*

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,
"Processing-in-Memory: A Workload-Driven Perspective"
Invited Article in IBM Journal of Research & Development, Special Issue on Hardware for Artificial Intelligence, to appear in November 2019.
[Preliminary arXiv version]

https://arxiv.org/pdf/1907.12947.pdf

Computer Architecture
Lecture 8: Computation in Memory III

Prof. Onur Mutlu

ETH Zürich

Fall 2019

11 October 2019

Accelerating Linked Data Structures

- Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu,
  "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation"
  Proceedings of the 34th IEEE International Conference on Computer Design (**ICCD**), Phoenix, AZ, USA, October 2016.

Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation

Kevin Hsieh[†]   Samira Khan[‡]   Nandita Vijaykumar[†]
Kevin K. Chang[†]   Amirali Boroumand[†]   Saugata Ghose[†]   Onur Mutlu[§†]
[†]*Carnegie Mellon University*   [‡]*University of Virginia*   [§]*ETH Zürich*

Executive Summary

- **Our Goal:** Accelerating pointer chasing inside main memory

- **Challenges:** Parallelism challenge and Address translation challenge

- **Our Solution:** In-Memory PoInter Chasing Accelerator (IMPICA)
  - Address-access decoupling: enabling parallelism in the accelerator with low cost
  - IMPICA page table: low cost page table in logic layer

- **Key Results**:
  - 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
  - 6% - 41% reduction in energy consumption

Linked Data Structures

- Linked data structures are widely used in many important applications

Linked data structures are connected by pointers

B-Tree

Hash Table
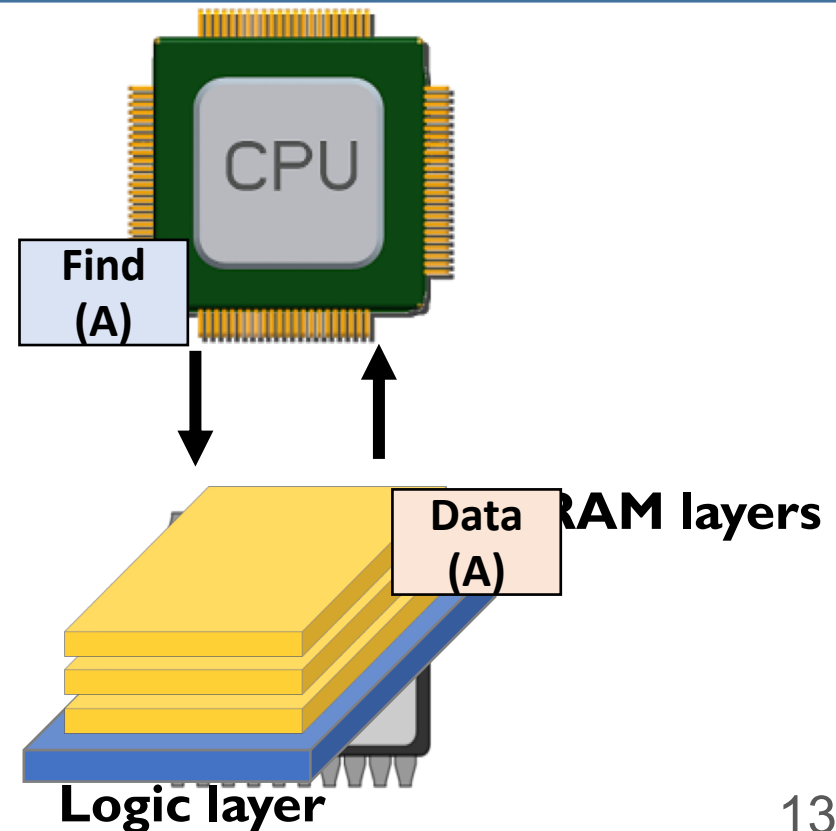
Key 1
Key 2
Key 3
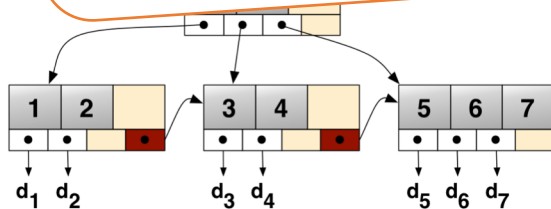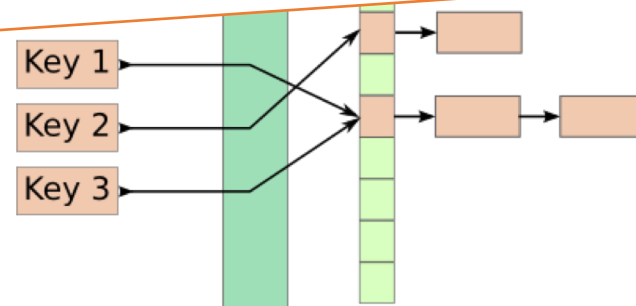
The Problem: Pointer Chasing

- Traversing linked data structures requires chasing pointers

Find(A)

H

E   Q

A   E   M

Addr
(A)

CPU

Data
(A)
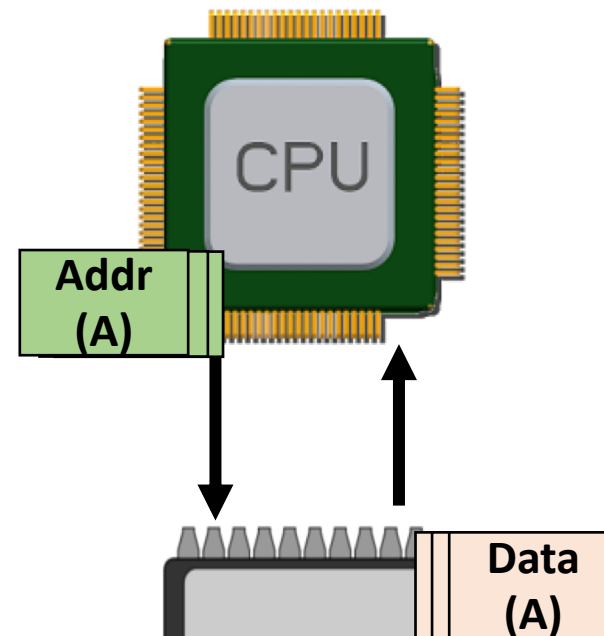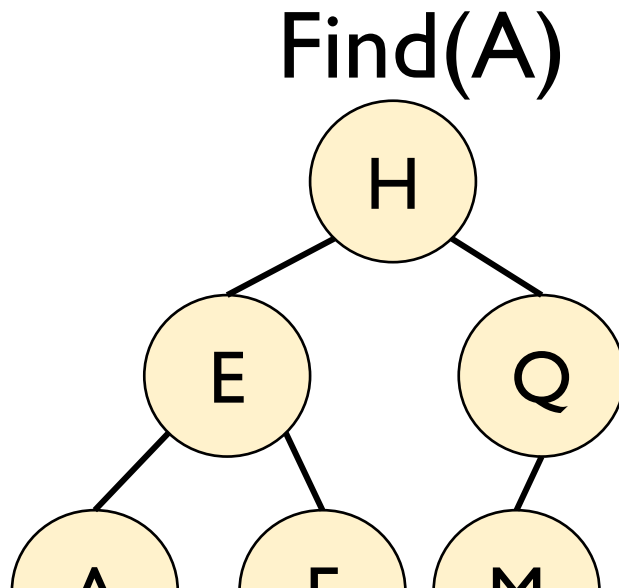
**Serialized and irregular access pattern
6X cycles per instruction in real workloads**

# Our Goal
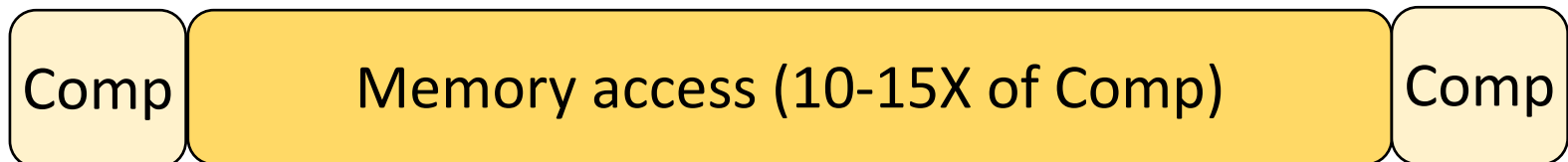
Accelerating pointer chasing inside main memory

Find(A)

CPU

Find (A)

Data (A)

RAM layers

Logic layer

Parallelism Challenge

Time →

CPU core

Comp | Memory access | Comp

CPU core

Comp | Memory access | Comp

In-Memory Accelerator

Comp | Memory access | Comp | Comp | Memory access | Comp

Slower for two operations
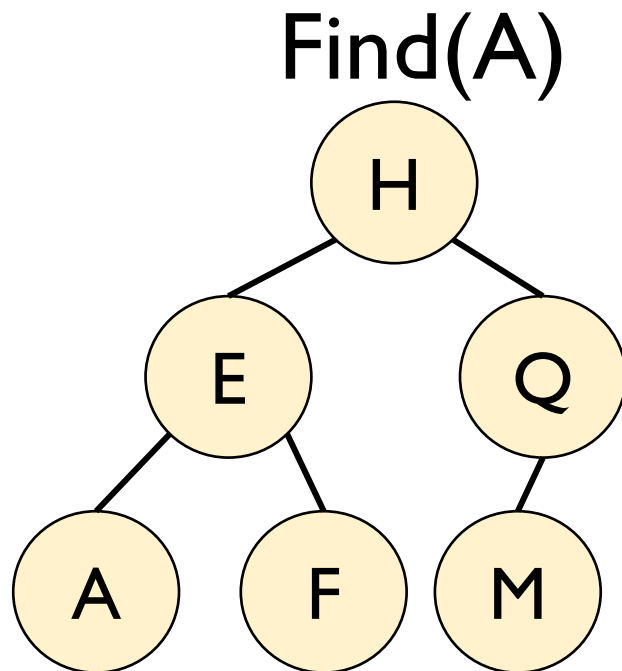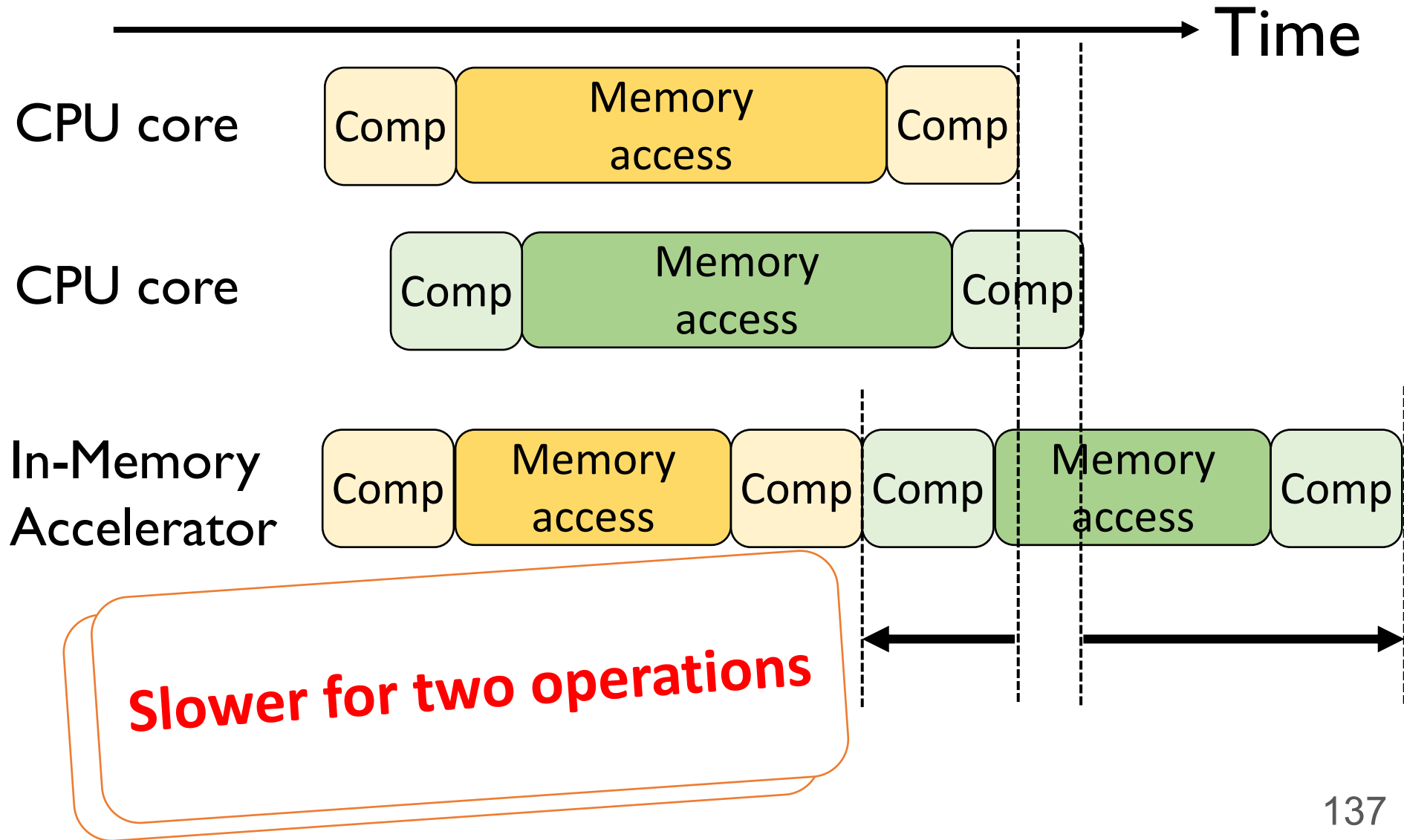
Parallelism Challenge and Opportunity

- A simple in-memory accelerator can still be slower than multiple CPU cores
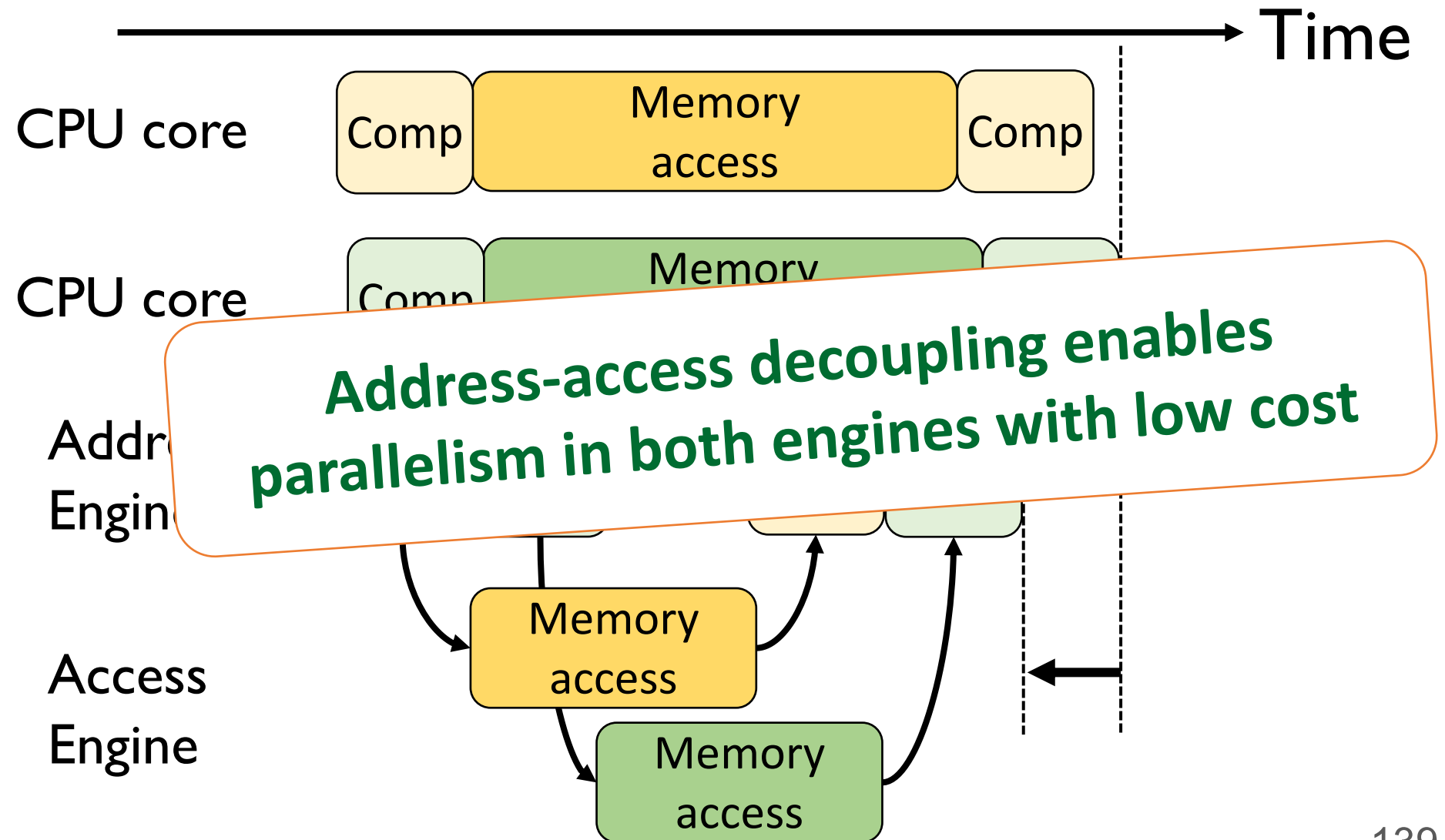
CPU core	CPU core	CPU core
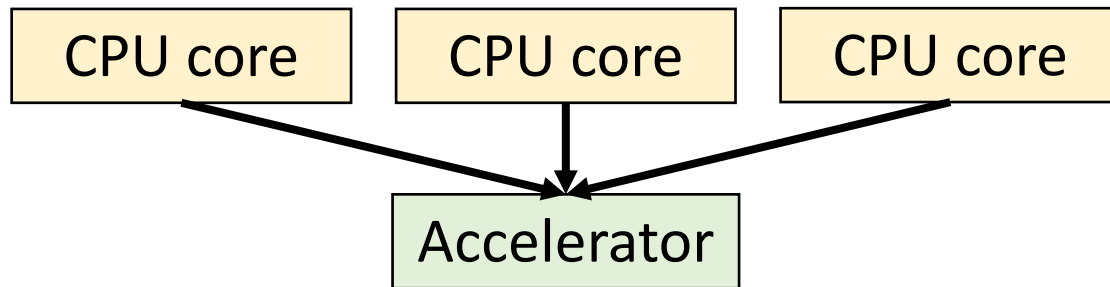
Accelerator

- Opportunity: a pointer-chasing accelerator spends a long time waiting for memory

Comp	Memory access (10-15X of Comp)	Comp

Our Solution: Address-Access Decoupling

Time →

CPU core

| Comp | Memory access | Comp |

CPU core

| Comp | Memory access | |

Address Engine

Access Engine

Memory access

Memory access
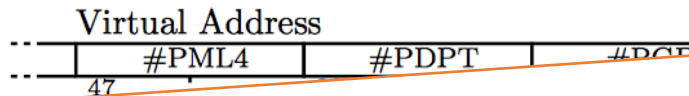
Address-access decoupling enables parallelism in both engines with low cost

IMPICA Core Architecture
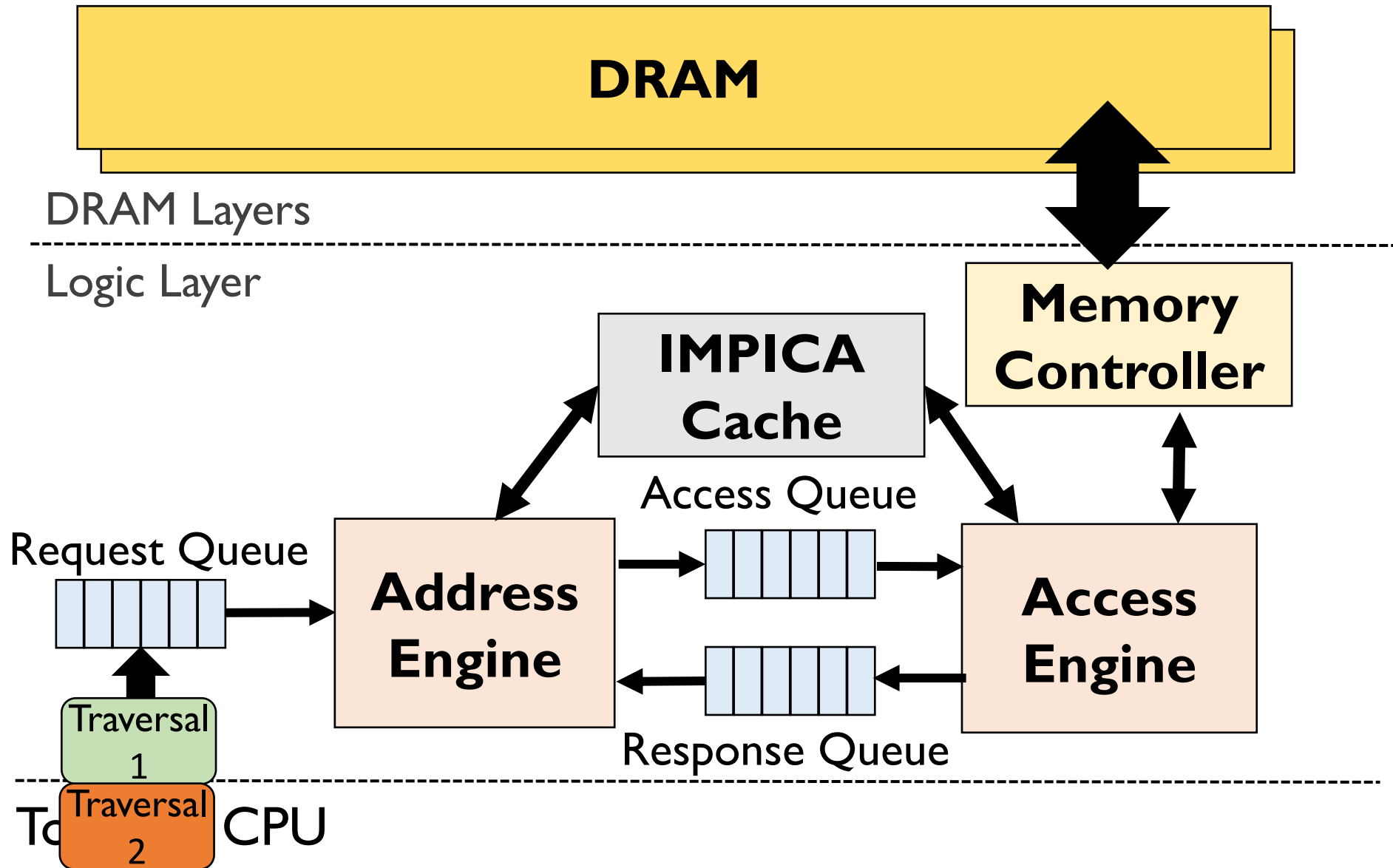
Address Translation Challenge

The page table walk requires multiple memory accesses

Virtual Address

#PML4   #PDPT   #PGD

47

No TLB/MMU on the memory side
Duplicating it is costly and creates compatibility issue
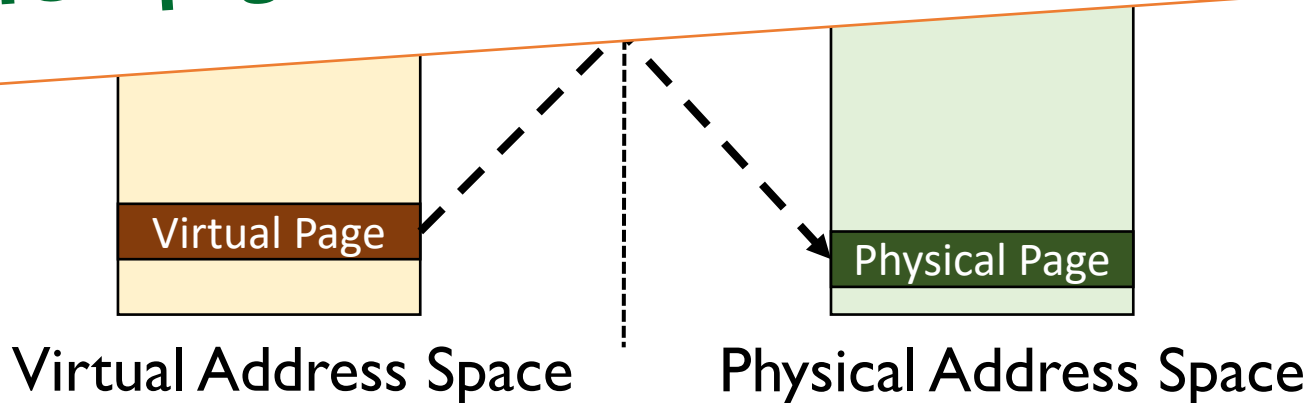
PML4   PDPT   PGD   PGT 2^9
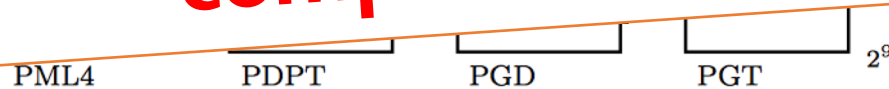
Page table walk

Our Solution: IMPICA Page Table

- Completely decouple the page table of IMPICA from the page table of the CPUs

IMPICA Page Table

Map linked data structure into IMPICA regions
IMPICA page table is a partial-to-any mapping

Virtual Page

Physical Page

Virtual Address Space   Physical Address Space

IMPICA Page Table: Mechanism

Virtual Address

Bit [47:4 ...]

Bit [11:0]

Region Table

Flat page table
saves one memory access

Tiny region table is almost
always in the cache

Flat Page Table
(2MB)

Small Page Table
(4KB)

+

+

+
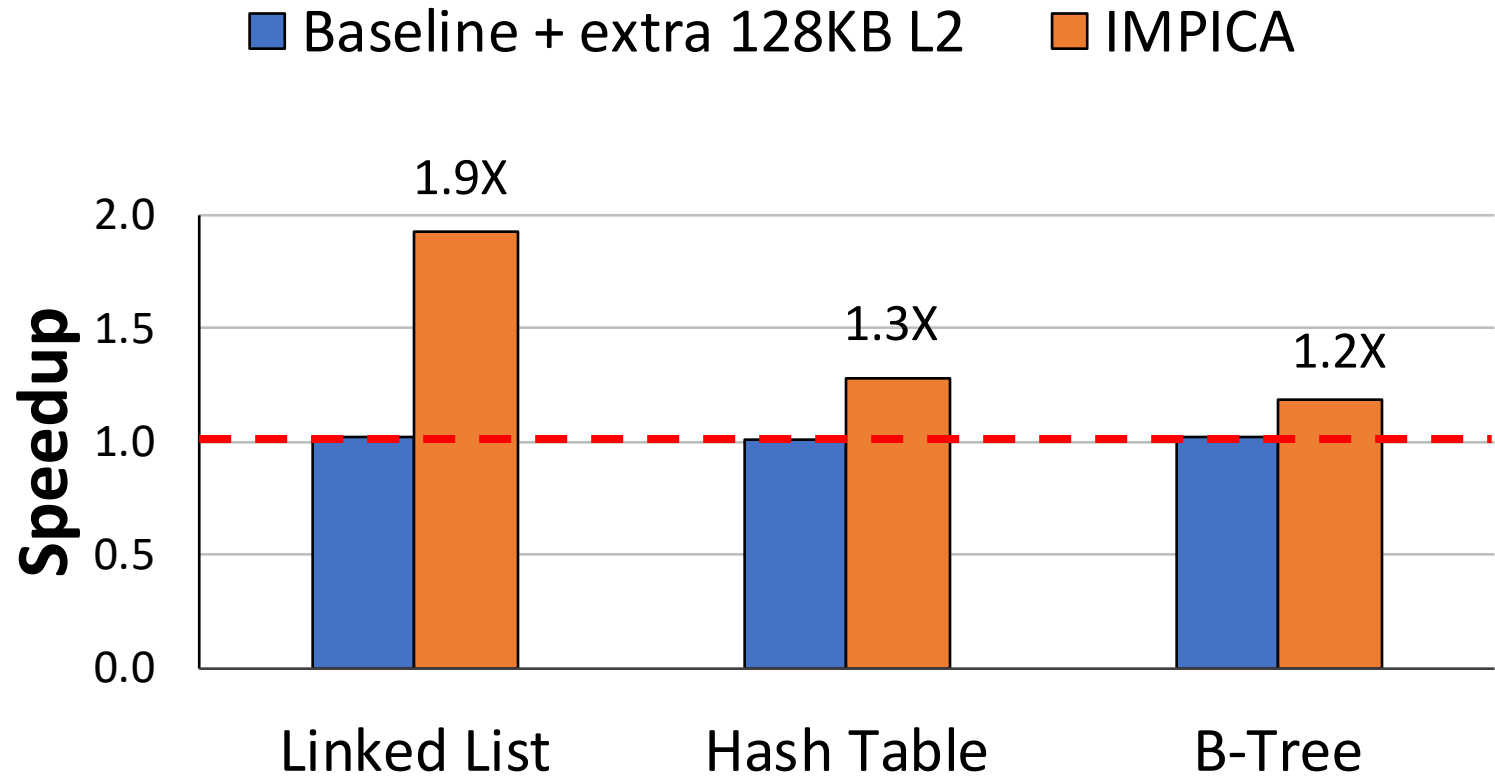
Physical Address
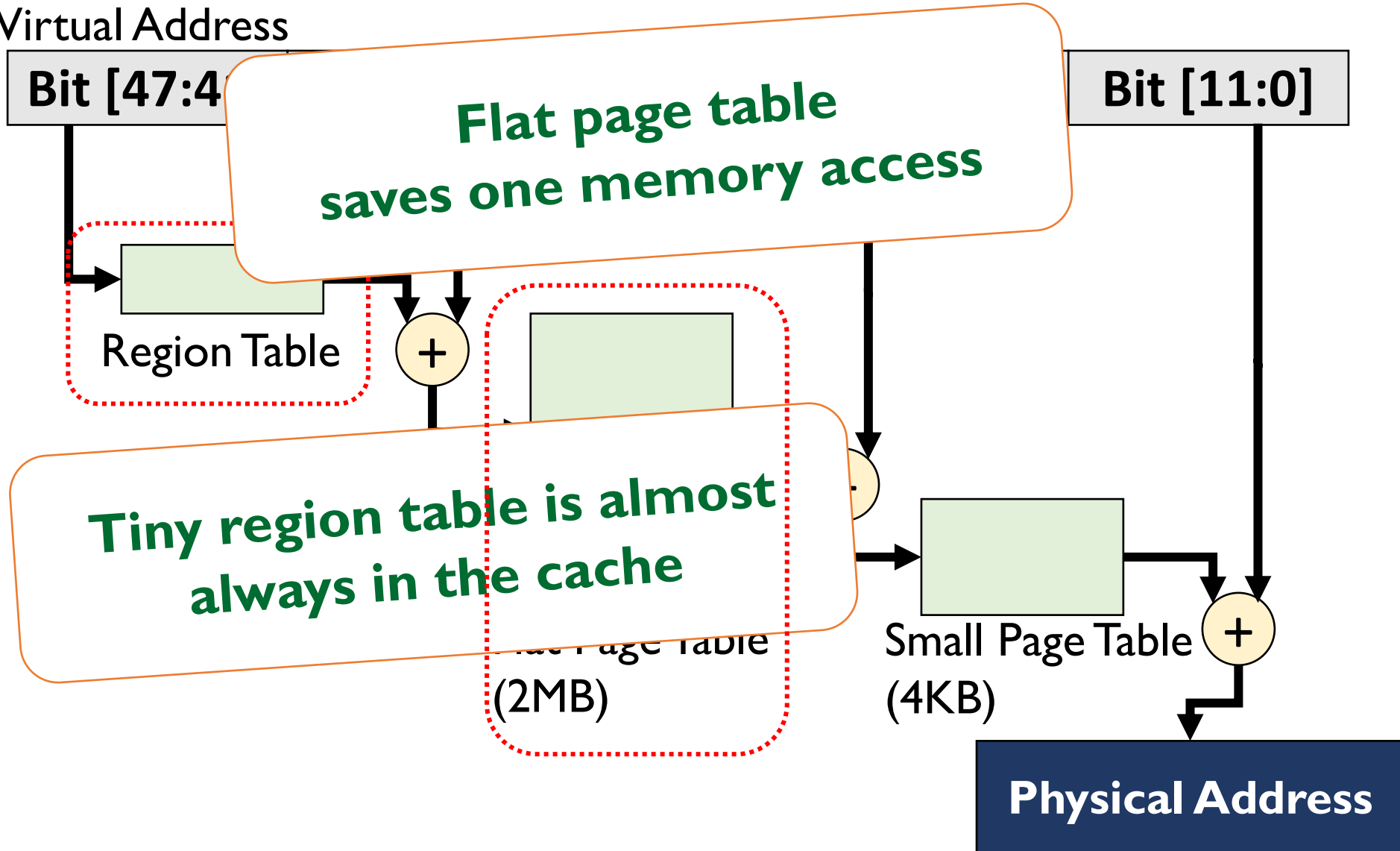
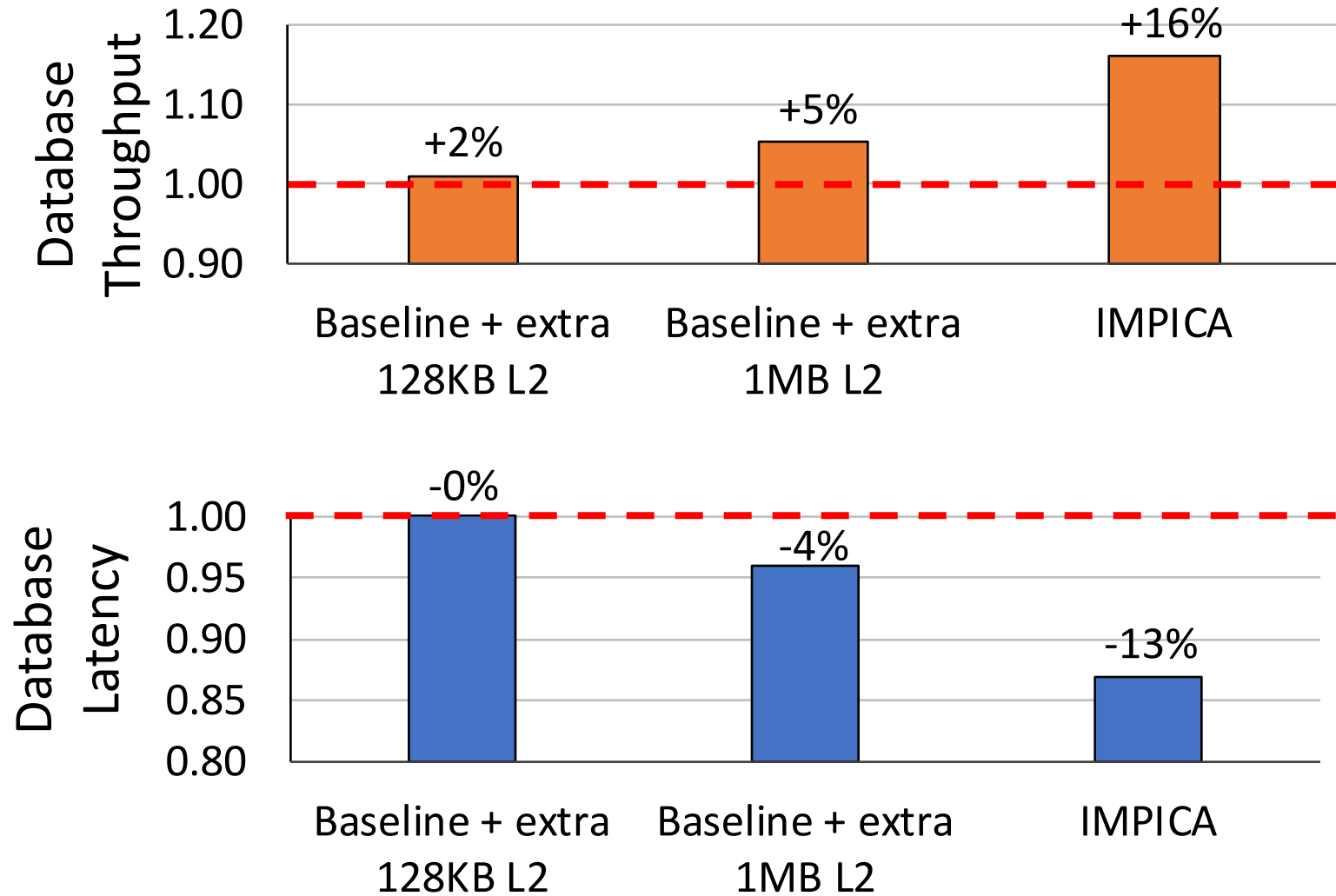# Evaluation Methodology

- Simulator: gem5
- System Configuration
  - CPU
    - 4 OoO cores, 2GHz
    - Cache: 32KB L1, 1MB L2
  - IMPICA
    - 1 core, 500MHz, 32KB Cache
  - Memory Bandwidth
    - 12.8 GB/s for CPU, 51.2 GB/s for IMPICA
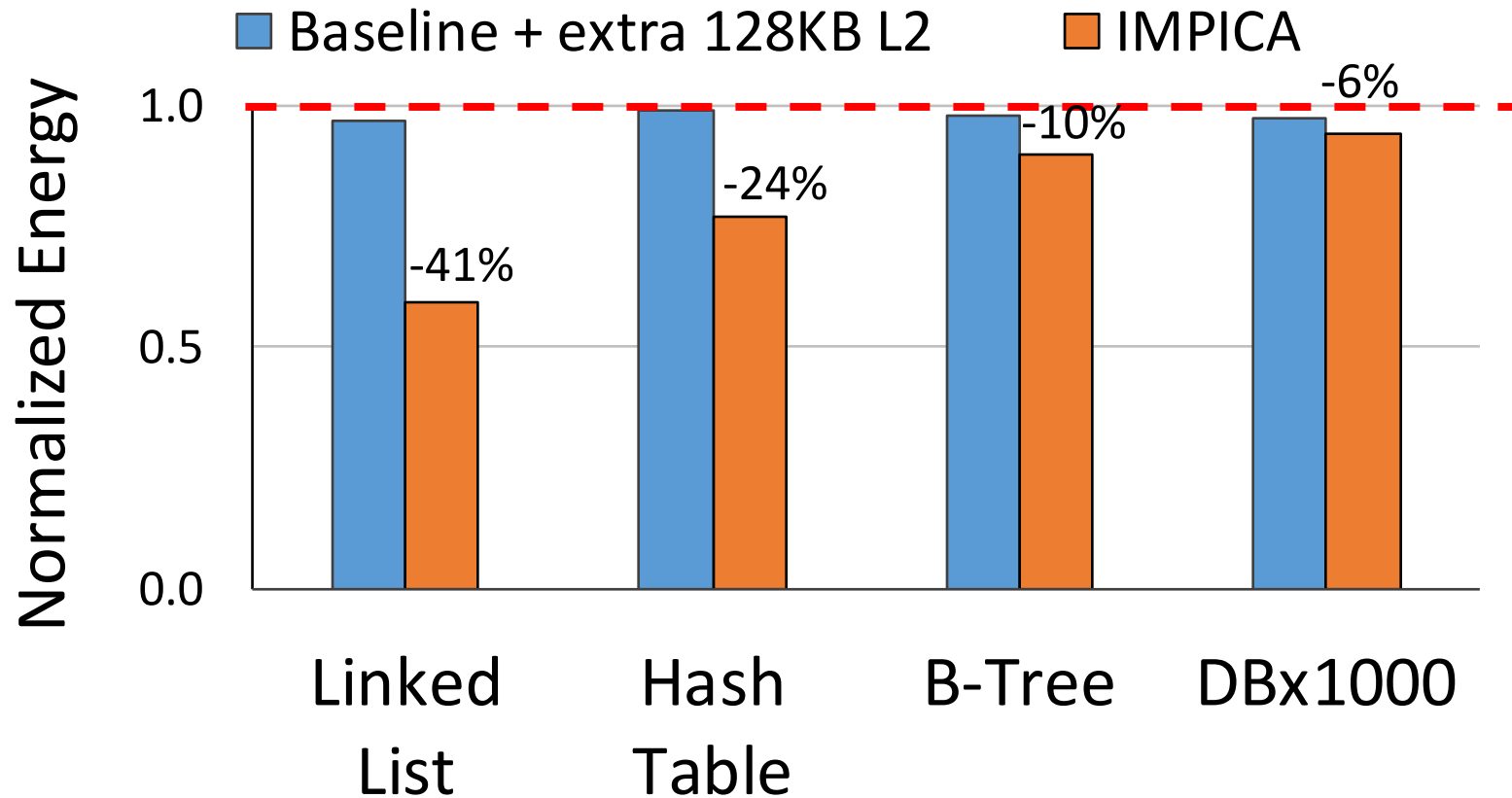- Our simulator code is open source
  - https://github.com/CMU-SAFARI/IMPICA

Result – Microbenchmark Performance

Result – Database Performance

System Energy Consumption

Area and Power Overhead

CPU (Cortex-A57)	5.85 mm^2 per core
L2 Cache	5 mm^2 per MB
Memory Controller	10 mm^2
IMPICA (+32KB cache)	0.45 mm^2

- Power overhead: average power increases by 5.6%