

Computer Architecture

Lecture 4: Processing near Memory

Prof. Onur Mutlu

ETH Zürich

Fall 2022

7 October 2022

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
 - **Processing using Memory**
 - Processing near Memory
- How to Enable Adoption of Processing in Memory
- Conclusion

Two PIM Approaches

5.2. Two Approaches: Processing Using Memory (PUM) vs. Processing Near Memory (PNM)

Many recent works take advantage of the memory technology innovations that we discuss in Section 5.1 to enable and implement PIM. We find that these works generally take one of two approaches, which are categorized in Table 1: (1) *processing using memory* or (2) *processing near memory*. We briefly describe each approach here. Sections 6 and 7 will provide example approaches and more detail for both.

Table 1: Summary of enabling technologies for the two approaches to PIM used by recent works. Adapted from [341] and extended.

| Approach | Example Enabling Technologies |
|-------------------------|---|
| Processing Using Memory | SRAM DRAM Phase-change memory (PCM) Magnetic RAM (MRAM) Resistive RAM (RRAM)/memristors |
| Processing Near Memory | Logic layers in 3D-stacked memory Silicon interposers Logic in memory controllers Logic in memory chips (e.g., near bank) Logic in memory modules Logic near caches Logic near/in storage devices |

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungrun,
["A Modern Primer on Processing in Memory"](#)

*Invited Book Chapter in **Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann**, Springer, to be published in 2021.*
[\[Tutorial Video on "Memory-Centric Computing Systems" \(1 hour 51 minutes\)\]](#)

Processing in Memory: Two Approaches

1. Processing using Memory
2. Processing near Memory

More on RowClone

- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,
"RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"
Proceedings of the 46th International Symposium on Microarchitecture (MICRO), Davis, CA, December 2013. [[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides \(pptx\)](#)] [[pdf](#)] [[Poster \(pptx\)](#)] [[pdf](#)]

RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Vivek Seshadri Yoongu Kim Chris Fallin* Donghyuk Lee
vseshadr@cs.cmu.edu yoongukim@cmu.edu cfallin@c1f.net donghyuk1@cmu.edu

Rachata Ausavarungnirun Gennady Pekhimenko Yixin Luo
rachata@cmu.edu gpekhime@cs.cmu.edu yixinluo@andrew.cmu.edu

Onur Mutlu Phillip B. Gibbons† Michael A. Kozuch† Todd C. Mowry
onur@cmu.edu phillip.b.gibbons@intel.com michael.a.kozuch@intel.com tcm@cs.cmu.edu

Carnegie Mellon University †Intel Pittsburgh

More on In-DRAM Bulk AND/OR

- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
"Fast Bulk Bitwise AND and OR in DRAM"
IEEE Computer Architecture Letters (***CAL***), April 2015.

Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*,
Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

*Carnegie Mellon University

†Intel Pittsburgh

More on In-DRAM Bitwise Operations

- Vivek Seshadri et al., “[Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology](#),” MICRO 2017.

Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

Vivek Seshadri^{1,5} Donghyuk Lee^{2,5} Thomas Mullins^{3,5} Hasan Hassan⁴ Amirali Boroumand⁵
Jeremie Kim^{4,5} Michael A. Kozuch³ Onur Mutlu^{4,5} Phillip B. Gibbons⁵ Todd C. Mowry⁵

¹Microsoft Research India ²NVIDIA Research ³Intel ⁴ETH Zürich ⁵Carnegie Mellon University

More on In-DRAM Bulk Bitwise Execution

- Vivek Seshadri and Onur Mutlu,
"In-DRAM Bulk Bitwise Execution Engine"
Invited Book Chapter in Advances in Computers, to appear
in 2020.
[[Preliminary arXiv version](#)]

In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri
Microsoft Research India
`visesha@microsoft.com`

Onur Mutlu
ETH Zürich
`onur.mutlu@inf.ethz.ch`

RowClone & Bitwise Ops in Real DRAM Chips

ComputeDRAM: In-Memory Compute Using Off-the-Shelf DRAMs

Fei Gao

feig@princeton.edu

Department of Electrical Engineering
Princeton University

Georgios Tziantzioulis

georgios.tziantzioulis@princeton.edu

Department of Electrical Engineering
Princeton University

David Wentzlaff

wentzlaf@princeton.edu

Department of Electrical Engineering
Princeton University

Pinatubo: RowClone and Bitwise Ops in PCM

Pinatubo: A Processing-in-Memory Architecture for Bulk Bitwise Operations in Emerging Non-volatile Memories

Shuangchen Li^{1*}, Cong Xu², Qiaosha Zou^{1,5}, Jishen Zhao³, Yu Lu⁴, and Yuan Xie¹

University of California, Santa Barbara¹, Hewlett Packard Labs²

University of California, Santa Cruz³, Qualcomm Inc.⁴, Huawei Technologies Inc.⁵
{shuangchenli, yuanxie}@ece.ucsb.edu¹

SIMDRAM Framework

- 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, **["SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM"](#)** *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Virtual, March-April 2021.
[[2-page Extended Abstract](#)]
[[Short Talk Slides \(pptx\)](#) ([pdf](#))]
[[Talk Slides \(pptx\)](#) ([pdf](#))]
[[Short Talk Video](#) (5 mins)]
[[Full Talk Video](#) (27 mins)]

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

| | | | |
|------------------------------------|-----------------------------------|-----------------------------|----------------------------------|
| *Nastaran Hajinazar ^{1,2} | *Geraldo F. Oliveira ¹ | Sven Gregorio ¹ | João Dinis Ferreira ¹ |
| Nika Mansouri Ghiasi ¹ | Minesh Patel ¹ | Mohammed Alser ¹ | Saugata Ghose ³ |
| | Juan Gómez-Luna ¹ | Onur Mutlu ¹ | |

¹ETH Zürich

²Simon Fraser University

³University of Illinois at Urbana–Champaign

In-DRAM Lookup-Table Based Execution

- To appear at MICRO 2022



pLUTo: Enabling Massively Parallel Computation in DRAM via Lookup Tables

João Dinis Ferreira[§]

Gabriel Falcao[†]

Juan Gómez-Luna[§]

Mohammed Alser[§]

Lois Orosa^{§∇}

Mohammad Sadrosadati[§]

Jeremie S. Kim[§]

Geraldo F. Oliveira[§]

Taha Shahroodi[‡]

Anant Nori^{*}

Onur Mutlu[§]

[§]ETH Zürich

[†]IT, University of Coimbra

[∇]Galicia Supercomputing Center

[‡]TU Delft

^{*}Intel

In-Flash Bulk Bitwise Execution

- To appear at MICRO 2022

Flash-Cosmos: In-Flash Bulk Bitwise Operations Using Inherent Computation Capability of NAND Flash Memory

Jisung Park^{§∇} Roknoddin Azizi[§] Geraldo F. Oliveira[§] Mohammad Sadrosadati[§]
Rakesh Nadig[§] David Novo[†] Juan Gómez-Luna[§] Myungsuk Kim[‡] Onur Mutlu[§]

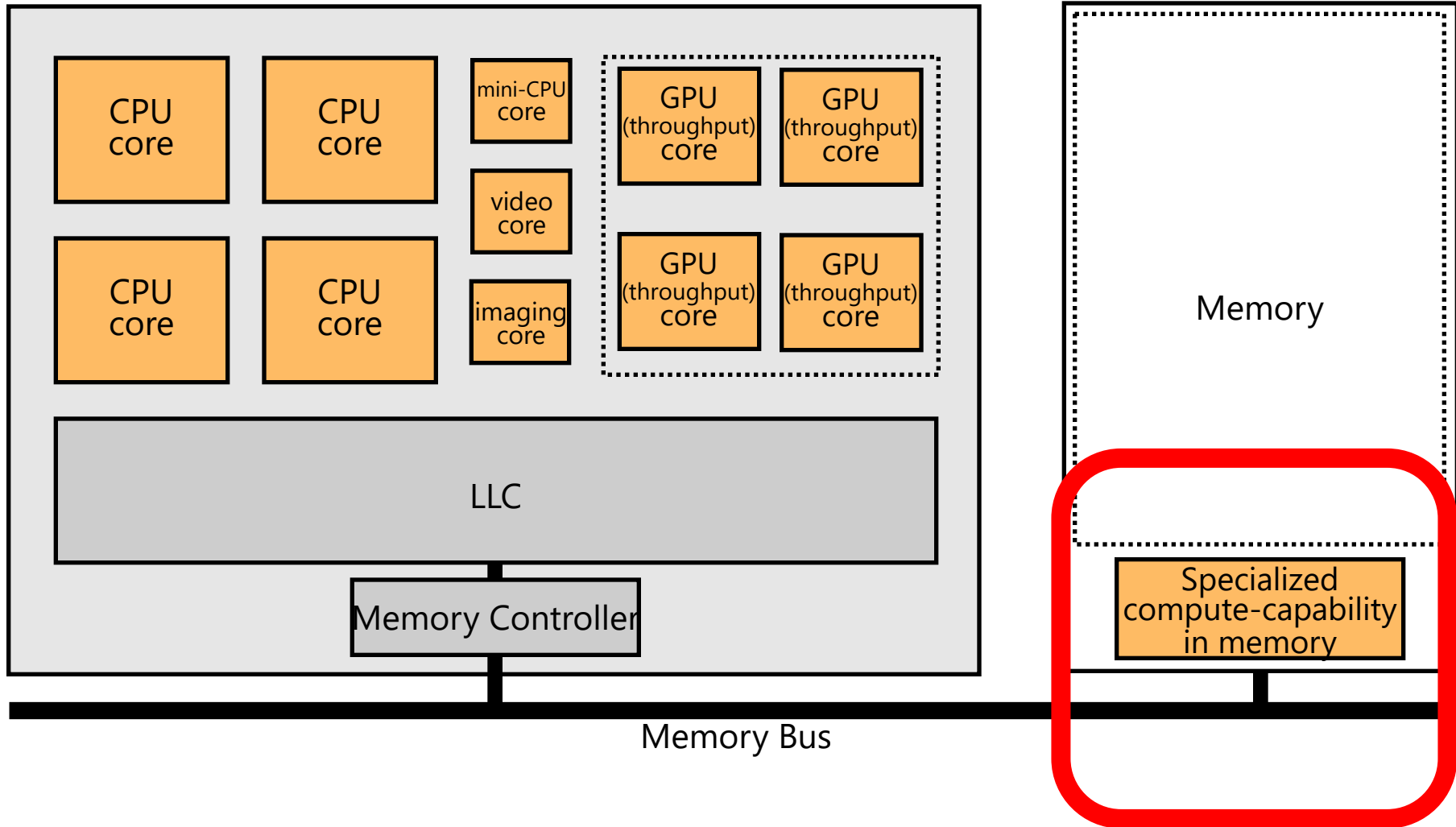
[§]*ETH Zürich* [∇]*POSTECH* [†]*LIRMM, Univ. Montpellier, CNRS* [‡]*Kyungpook National University*

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
 - Processing using Memory
 - **Processing near Memory**
- How to Enable Adoption of Processing in Memory
- Conclusion

We Need to Think Differently
from the Past Approaches

Memory as an Accelerator



Memory similar to a "conventional" accelerator

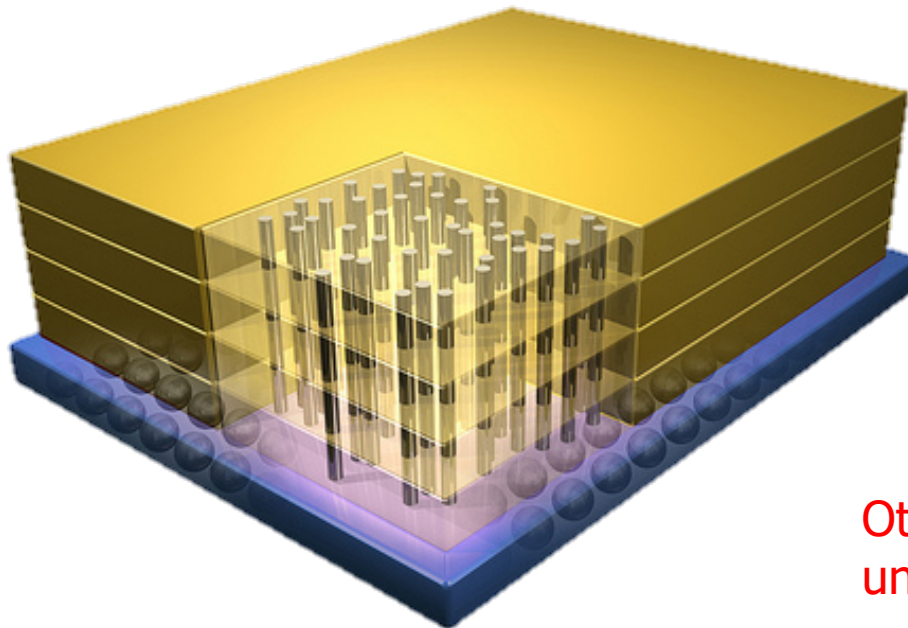
Processing in Memory: Two Approaches

1. Processing using Memory
2. Processing near Memory

Opportunity: 3D-Stacked Logic+Memory



Hybrid Memory Cube
C O N S O R T I U M



Memory

Logic

Other "True 3D" technologies
under development

DRAM Landscape (circa 2015)

| <i>Segment</i> | <i>DRAM Standards & Architectures</i> |
|----------------|---|
| Commodity | DDR3 (2007) [14]; DDR4 (2012) [18] |
| Low-Power | LPDDR3 (2012) [17]; LPDDR4 (2014) [20] |
| Graphics | GDDR5 (2009) [15] |
| Performance | eDRAM [28], [32]; RLDram3 (2011) [29] |
| 3D-Stacked | WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13]; HBM (2013) [19]; HMC1.0 (2013) [10]; HMC1.1 (2014) [11] |
| Academic | SBA/SSA (2010) [38]; Staged Reads (2012) [8]; RAIDR (2012) [27]; SALP (2012) [24]; TL-DRAM (2013) [26]; RowClone (2013) [37]; Half-DRAM (2014) [39]; Row-Buffer Decoupling (2014) [33]; SARP (2014) [6]; AL-DRAM (2015) [25] |

Table 1. Landscape of DRAM-based memory

Kim+, “[Ramulator: A Flexible and Extensible DRAM Simulator](#)”, IEEE CAL 2015.

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

Another Example: In-Memory Graph Processing

- Large graphs are everywhere (circa 2015)



36 Million
Wikipedia Pages



1.4 Billion
Facebook Users

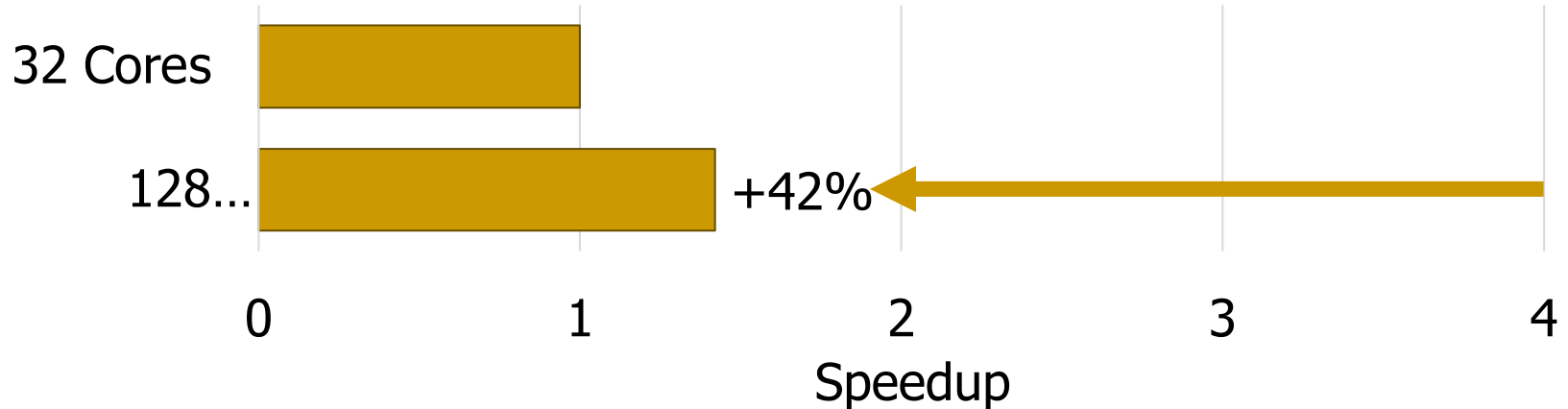


300 Million
Twitter Users



30 Billion
Instagram Photos

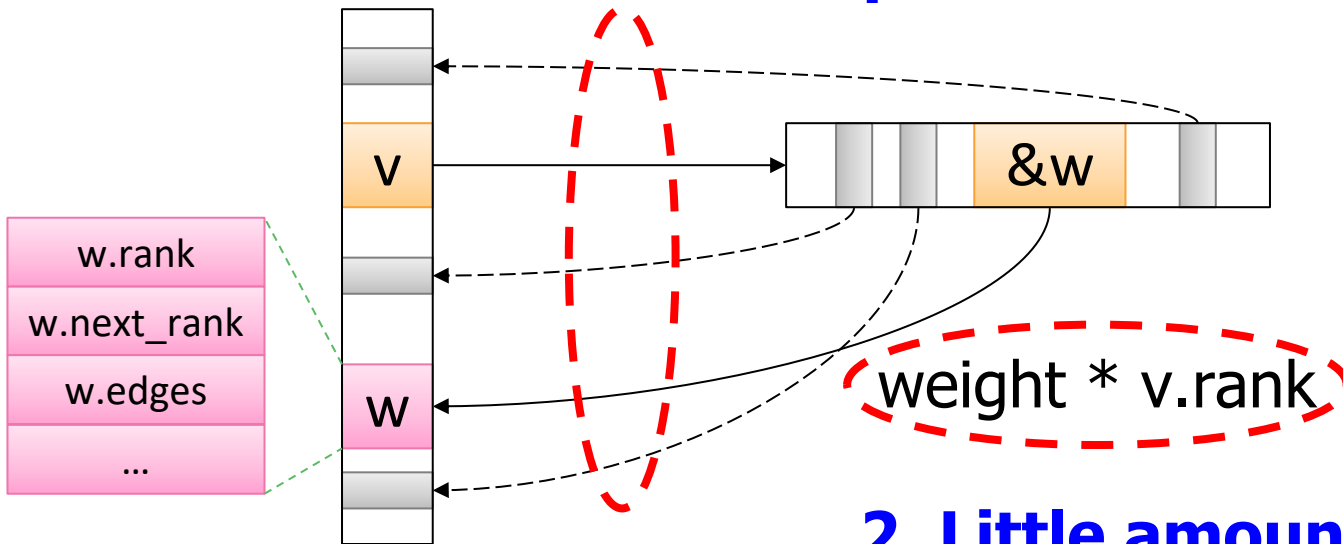
- Scalable large-scale graph processing is challenging



Key Bottlenecks in Graph Processing

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

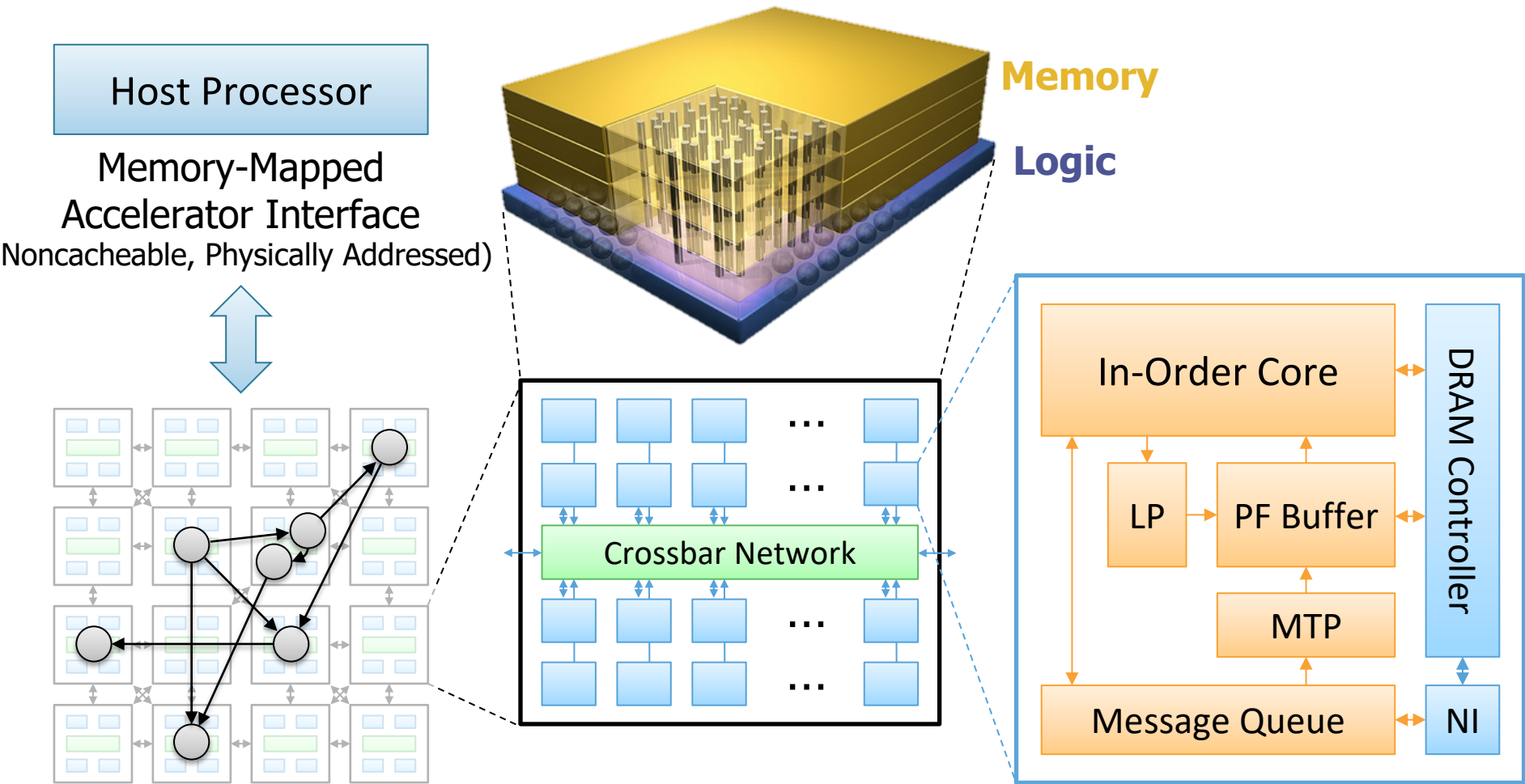
1. Frequent random memory accesses



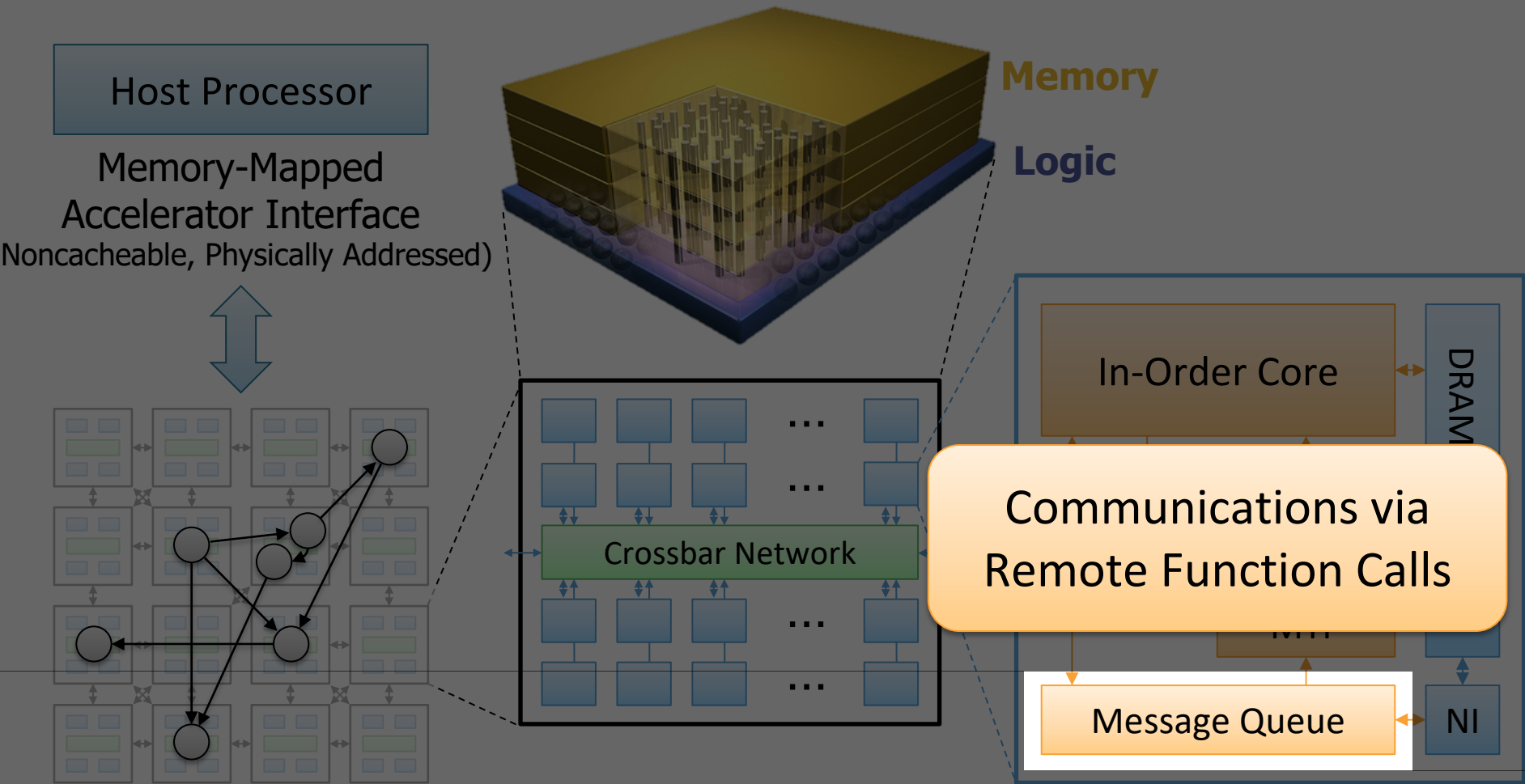
2. Little amount of computation

Tesseract System for Graph Processing

Interconnected set of 3D-stacked memory+logic chips with simple cores

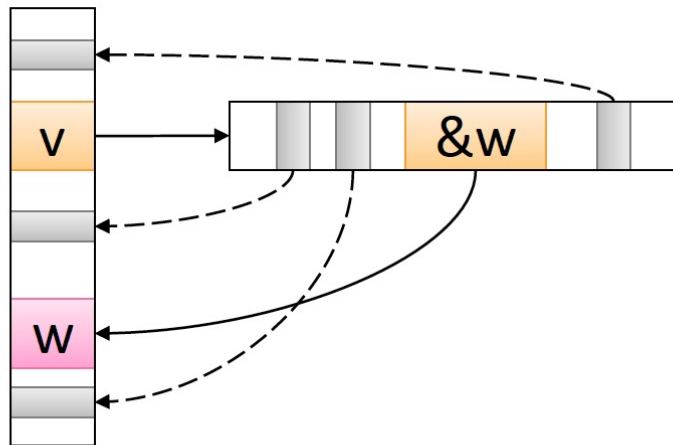


Tesseract System for Graph Processing



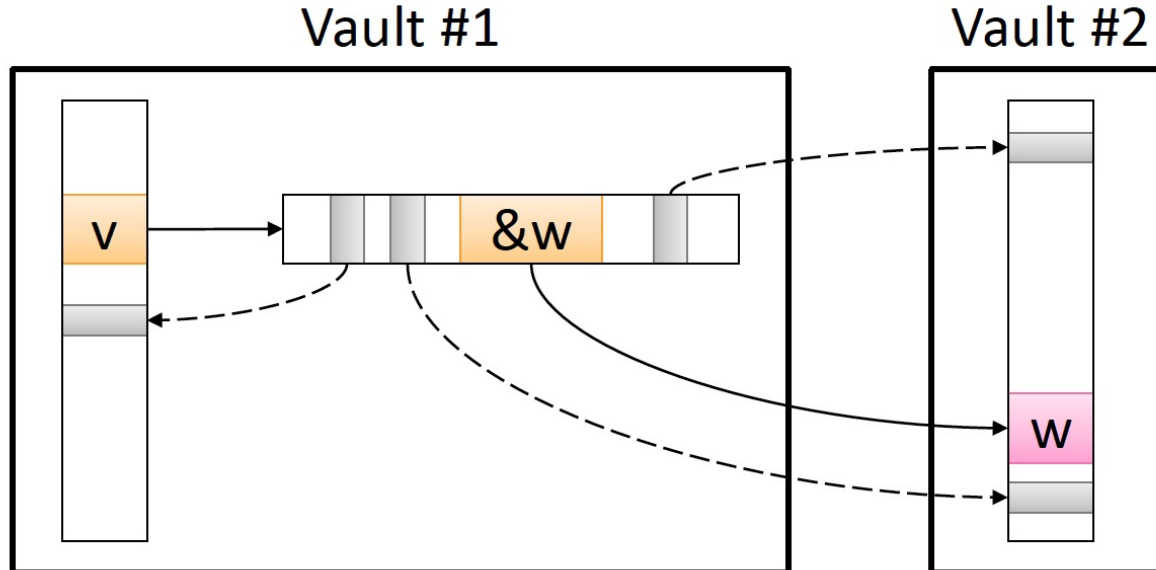
Communications In Tesseract (I)

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



Communications In Tesseract (II)

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

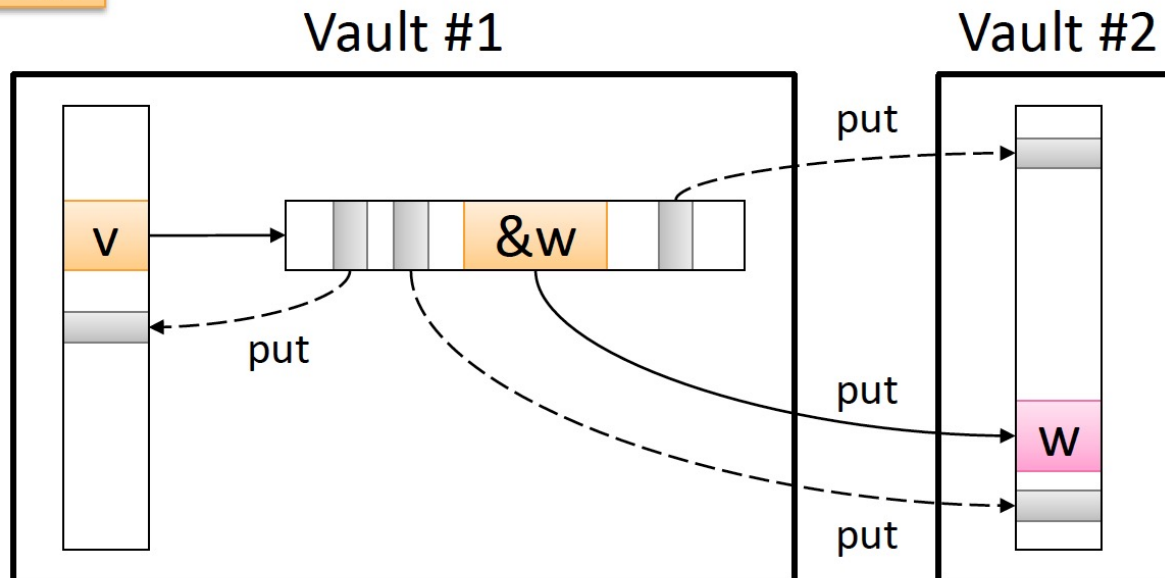


Communications In Tesseract (III)

```
for (v: graph.vertices) {
  for (w: v.successors) {
    put(w.id, function() { w.next_rank += weight * v.rank; });
  }
}
barrier();
```

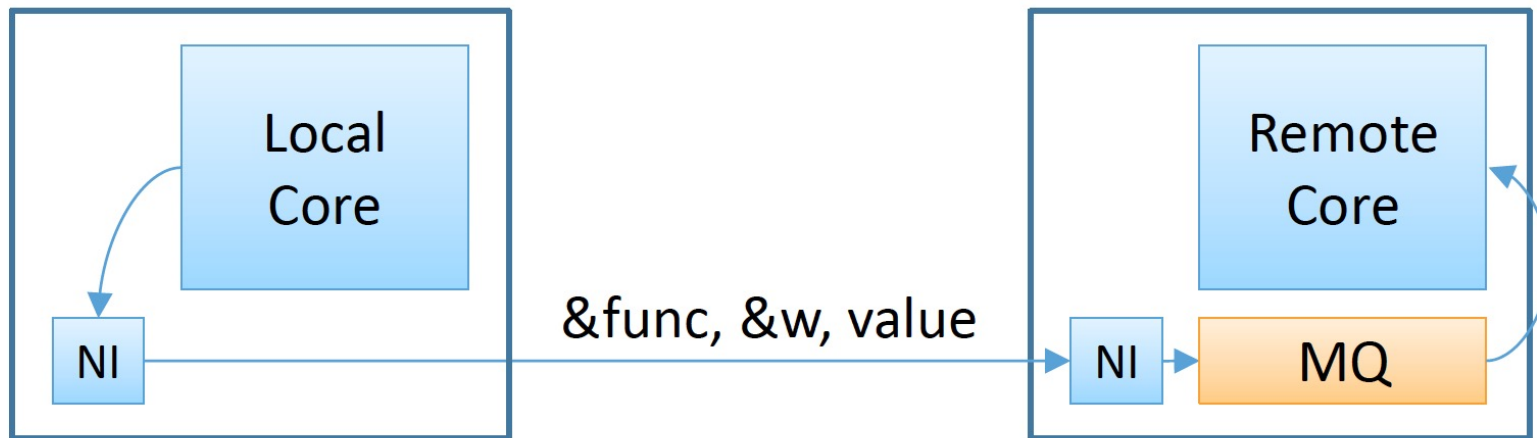
Non-blocking Remote Function Call

Can be **delayed**
until the nearest barrier



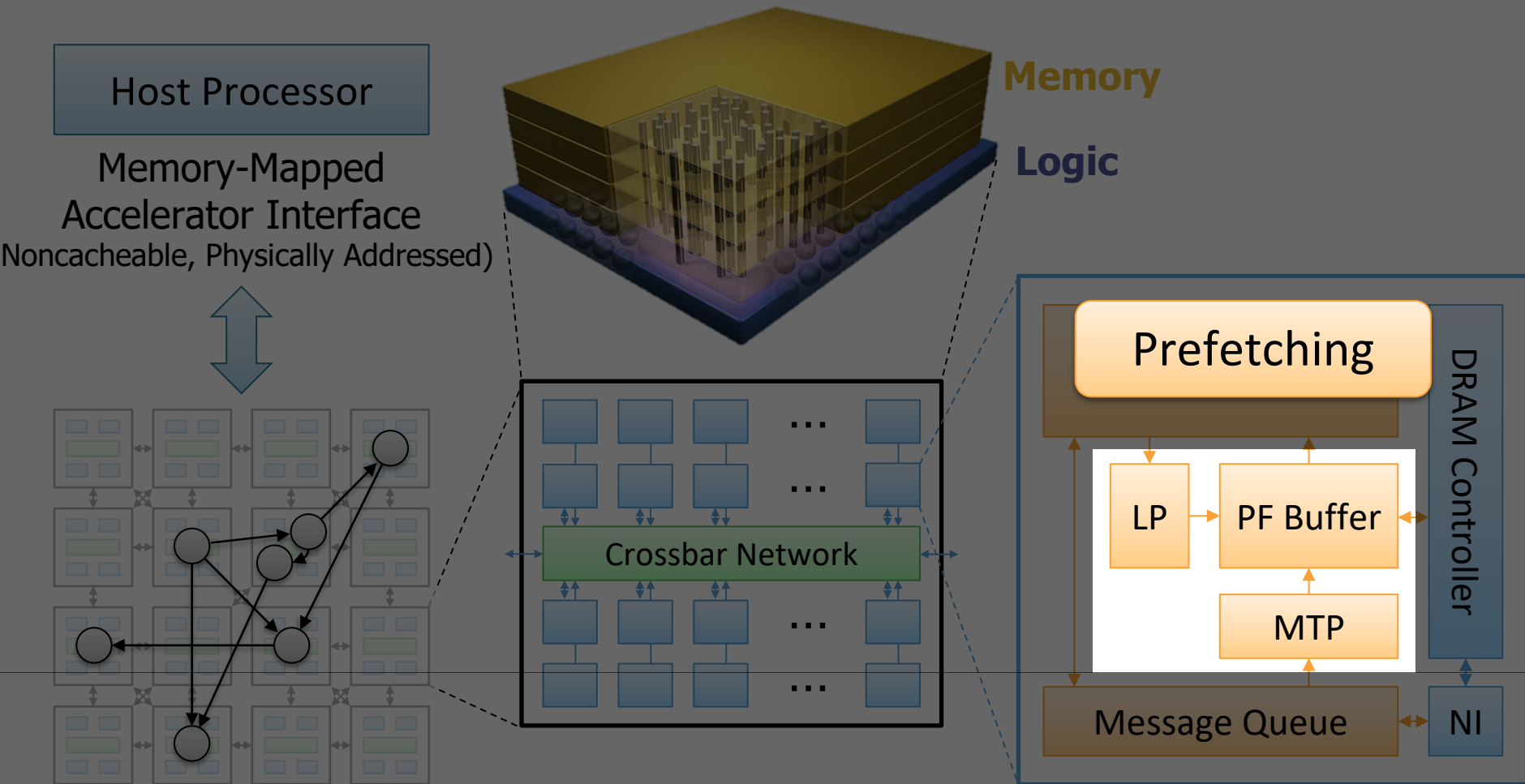
Remote Function Call (Non-Blocking)

1. Send function address & args to the remote core
2. Store the incoming message to the message queue
3. Flush the message queue when it is full or a synchronization barrier is reached



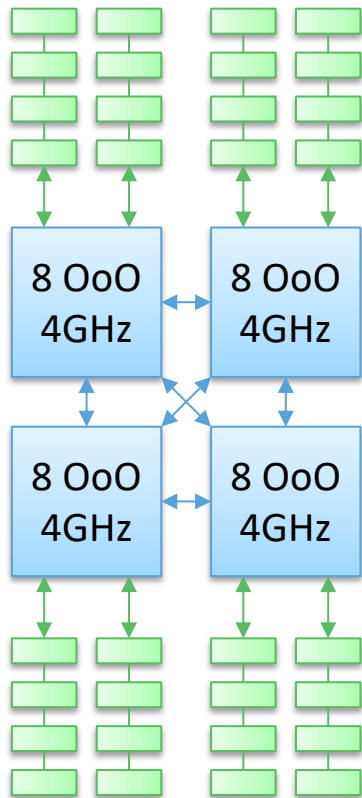
```
put(w.id, function() { w.next_rank += value; })
```


Tesseract System for Graph Processing



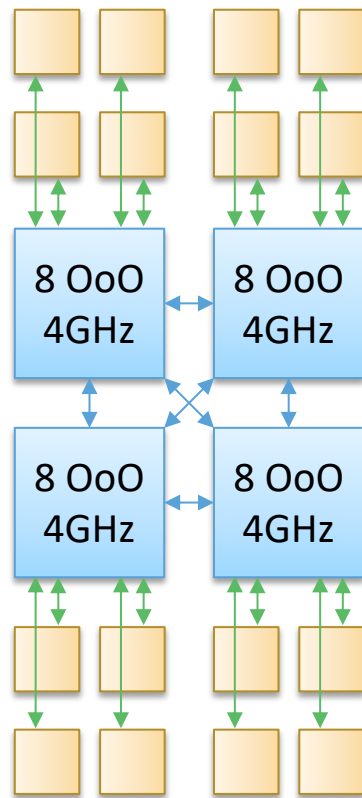
Evaluated Systems

DDR3-OoO



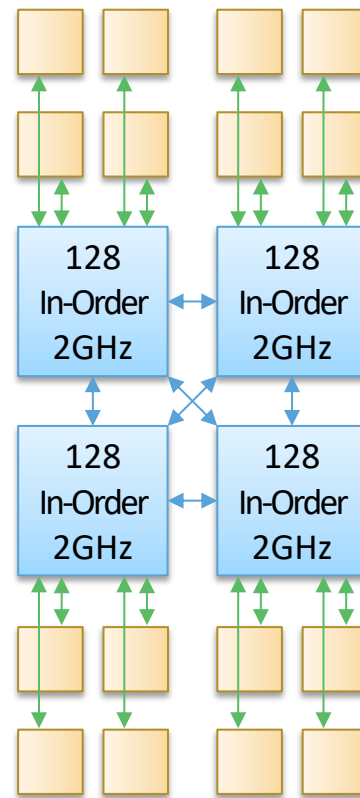
102.4GB/s

HMC-OoO



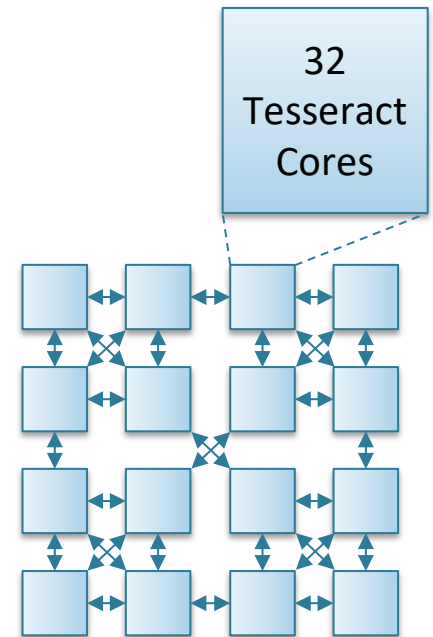
640GB/s

HMC-MC



640GB/s

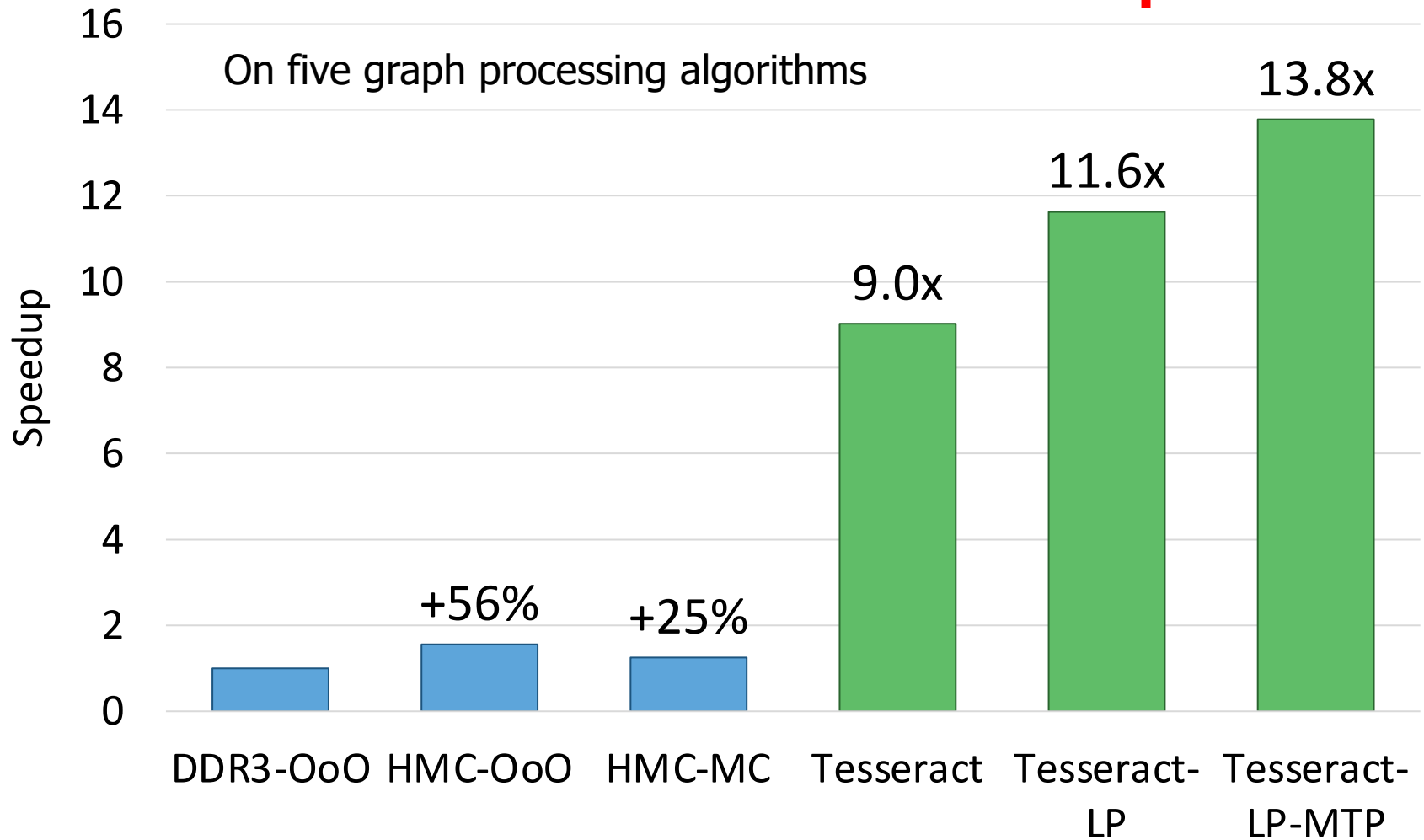
Tesseract



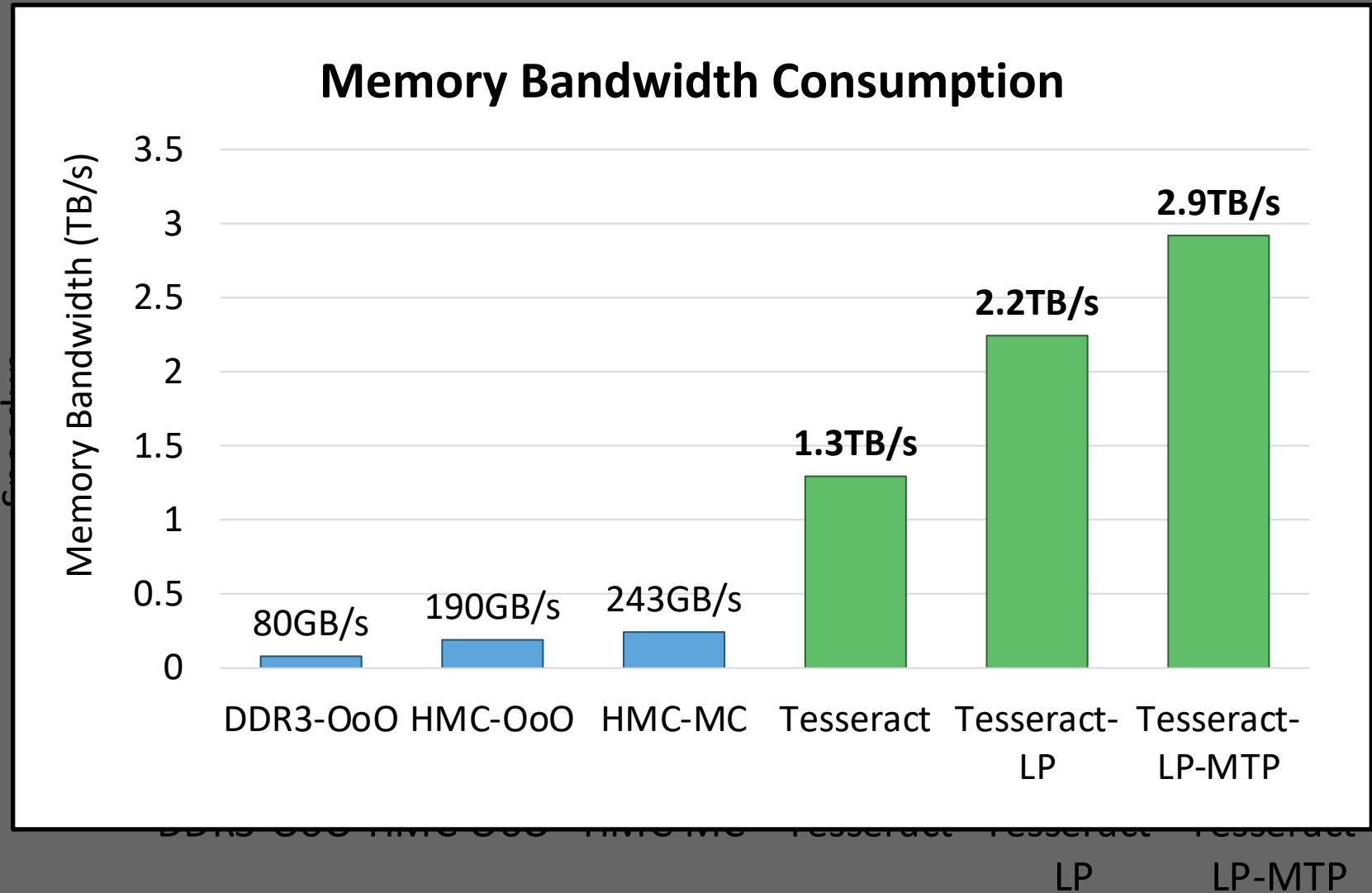
8TB/s

Tesseract Graph Processing Performance

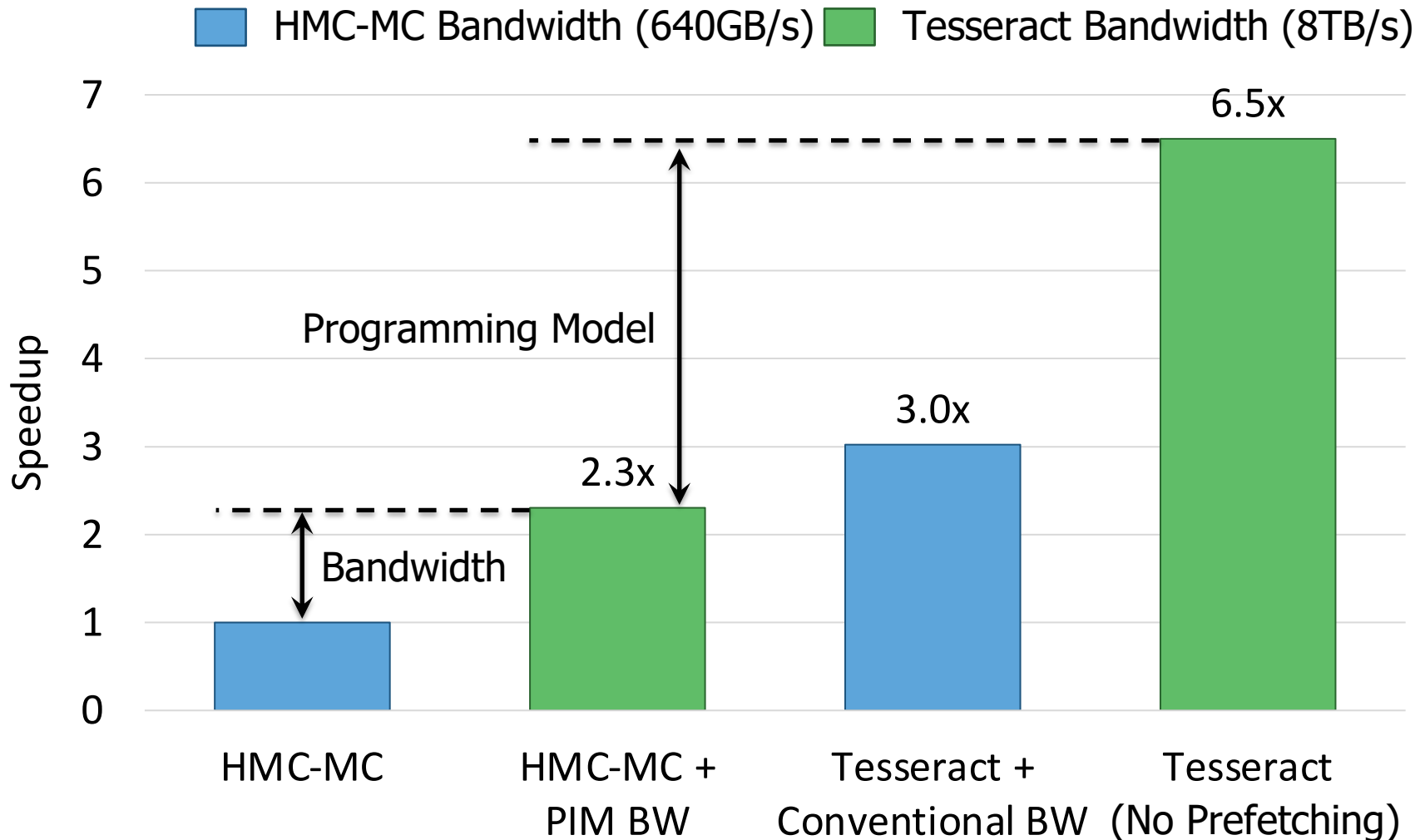
>13X Performance Improvement



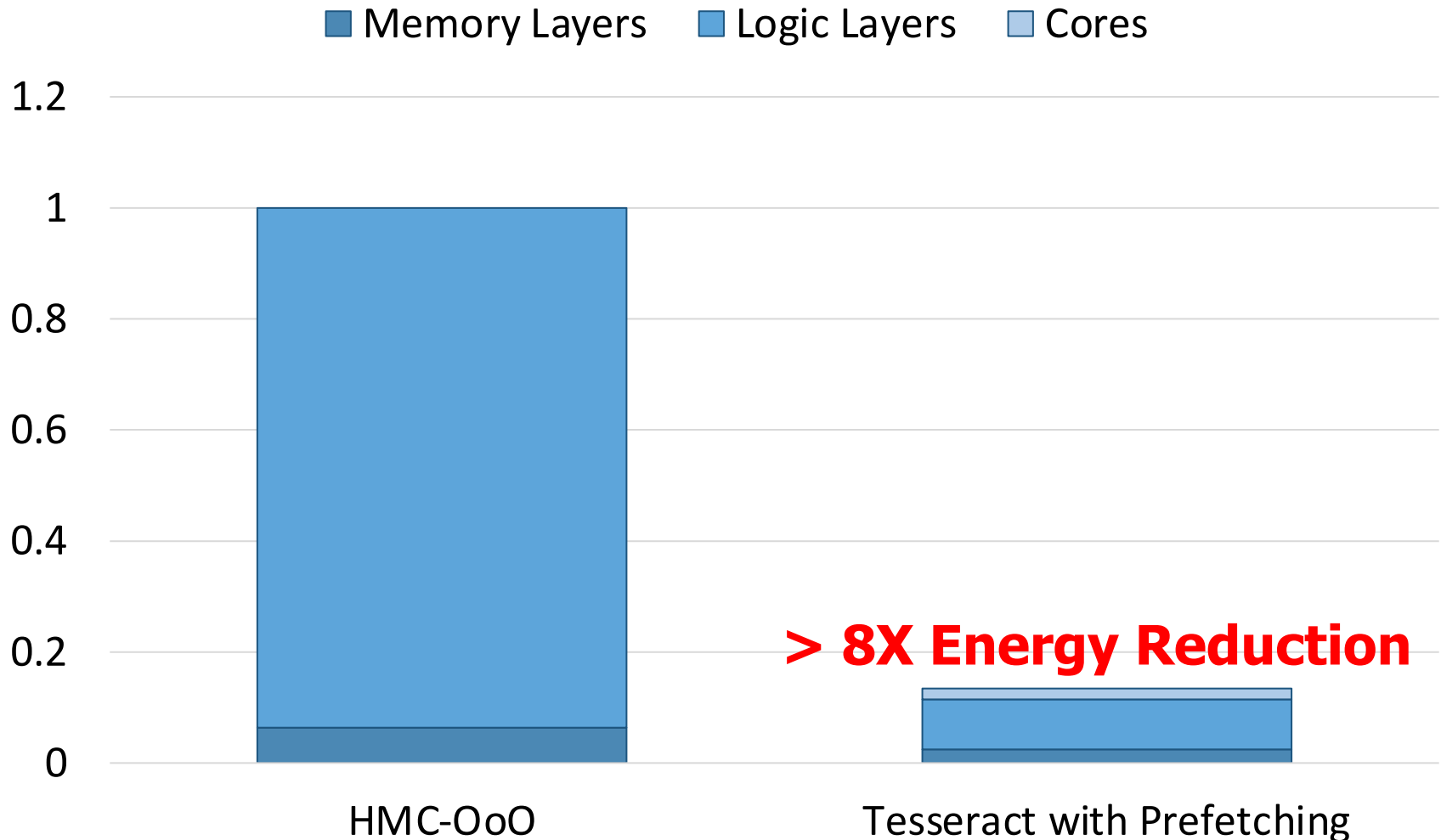
Tesseract Graph Processing Performance



Effect of Bandwidth & Programming Model



Tesseract Graph Processing System Energy



Tesseract: Advantages & Disadvantages

■ Advantages

- + Specialized graph processing accelerator using PIM
- + Large system performance and energy benefits
- + Takes advantage of 3D stacking for an important workload
- + More general than just graph processing

■ Disadvantages

- Changes a lot in the system
 - New programming model
 - Specialized Tesseract cores for graph processing
- Cost
- Scalability limited by off-chip links or graph partitioning

More on Tesseract

- Junwhan Ahn, Sungpack Hong, Sungjoo Yoo, Onur Mutlu, and Kiyoungh 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 (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)]
Top Picks Honorable Mention by IEEE Micro.

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

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

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
 - Processing using Memory
 - **Processing near Memory**
- How to Enable Adoption of Processing in Memory
- Conclusion

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 Boroumand¹

Saugata Ghose¹

Youngsok Kim²

Rachata Ausavarungnirun¹

Eric Shiu³

Rahul Thakur³

Daehyun Kim^{4,3}

Aki Kuusela³

Allan Knies³

Parthasarathy Ranganathan³

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

VP9



Video Playback

Google's **video codec**

VP9

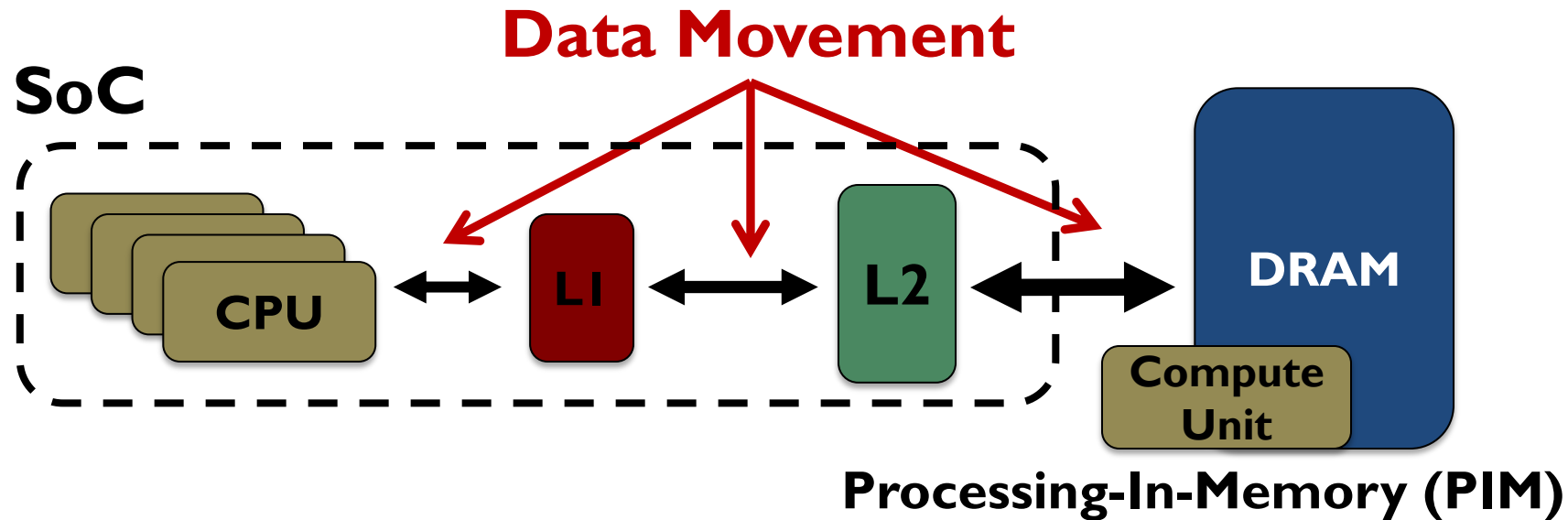


Video Capture

Google's **video codec**

Energy Cost of Data Movement

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



Potential solution: move computation **close to data**

Challenge: limited area and energy budget

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



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

VP9



Video Playback

Google's **video codec**

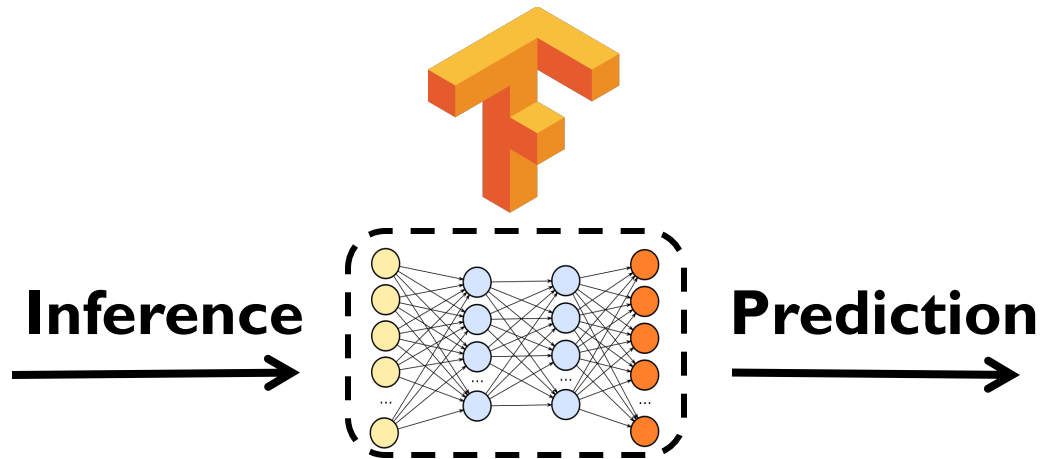
VP9



Video Capture

Google's **video codec**

TensorFlow Mobile



57.3% of the inference energy is spent on data movement



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

Packing



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**

Quantization



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



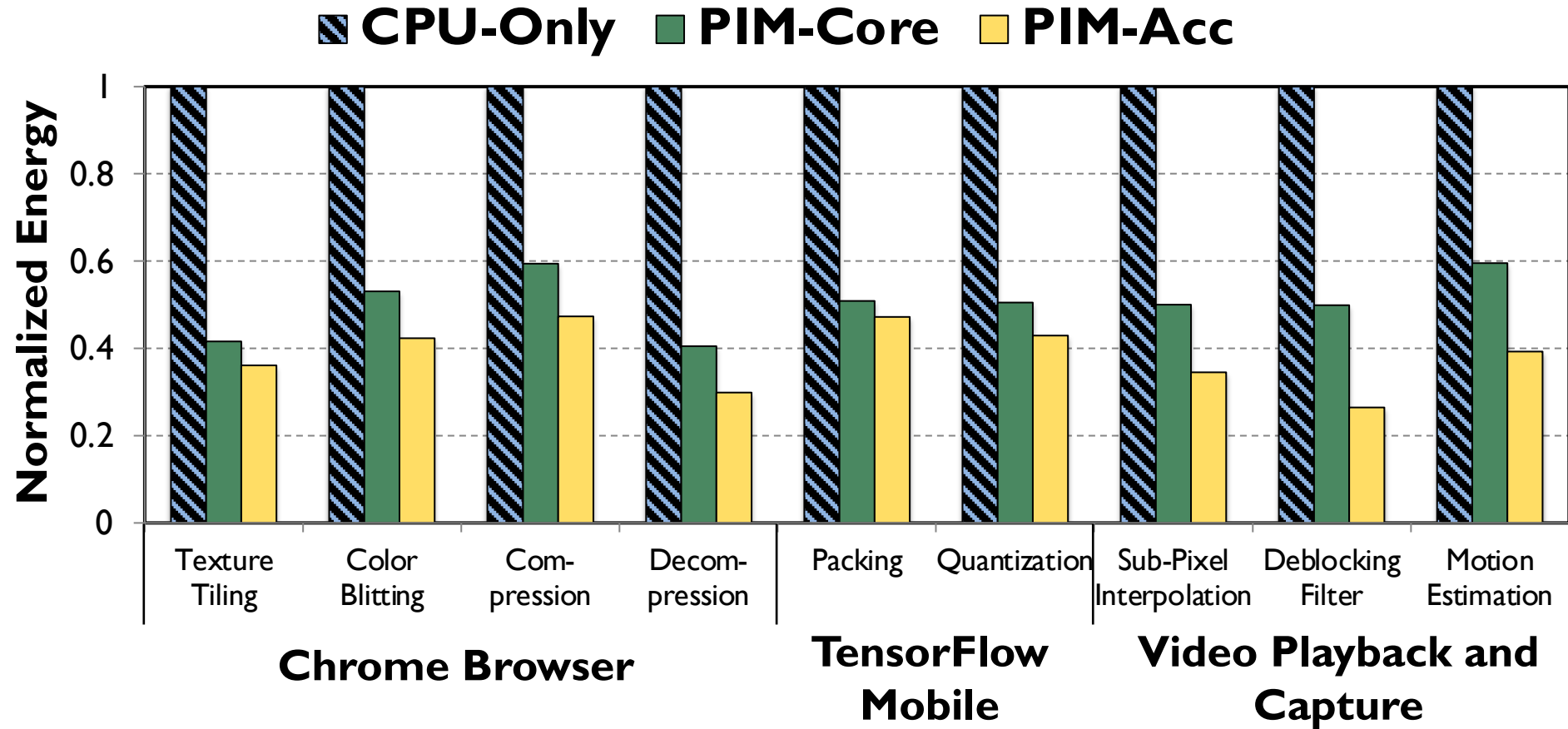
Up to **16.8%** of the inference **energy** and **16.1%** of inference **execution time**



Majority of **quantization** energy comes from **data movement**

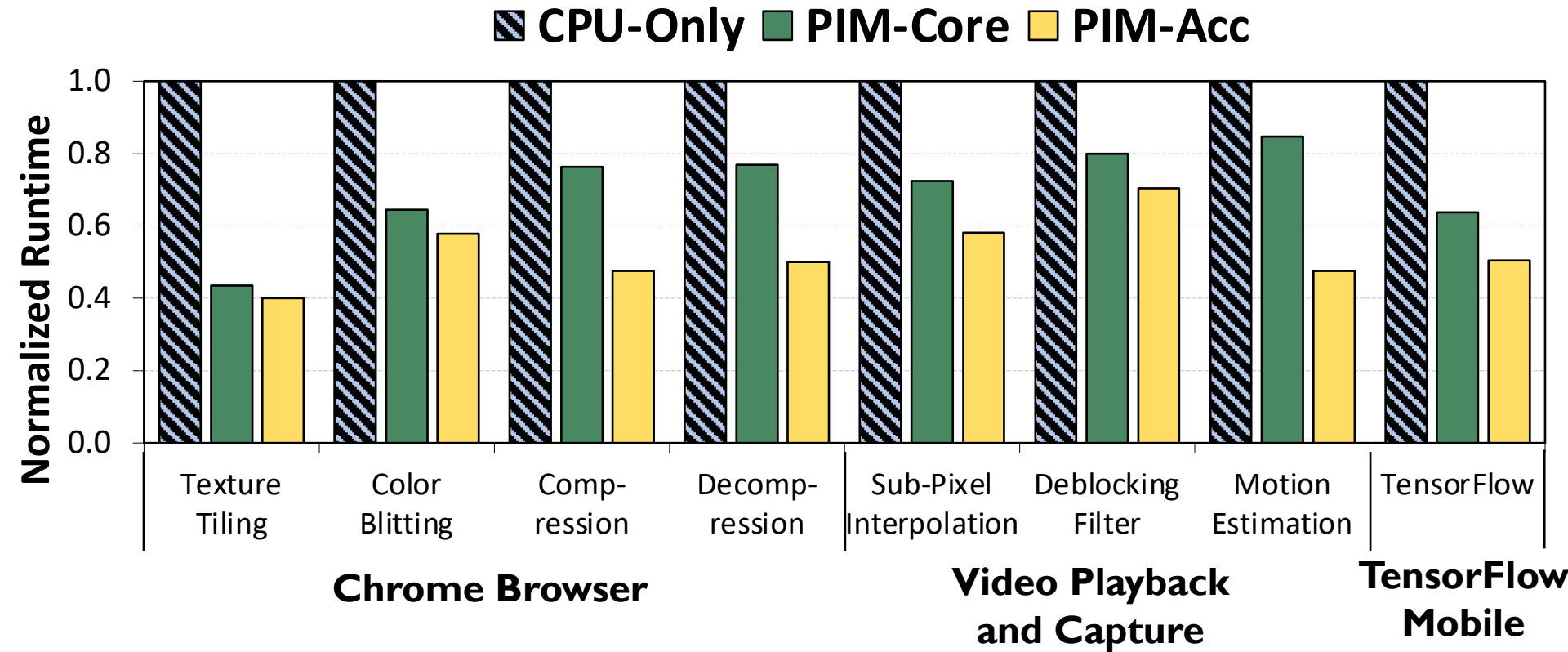
A simple **data conversion** operation that requires **shift, addition, and multiplication** operations

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

VP9



Video Playback

Google's video codec

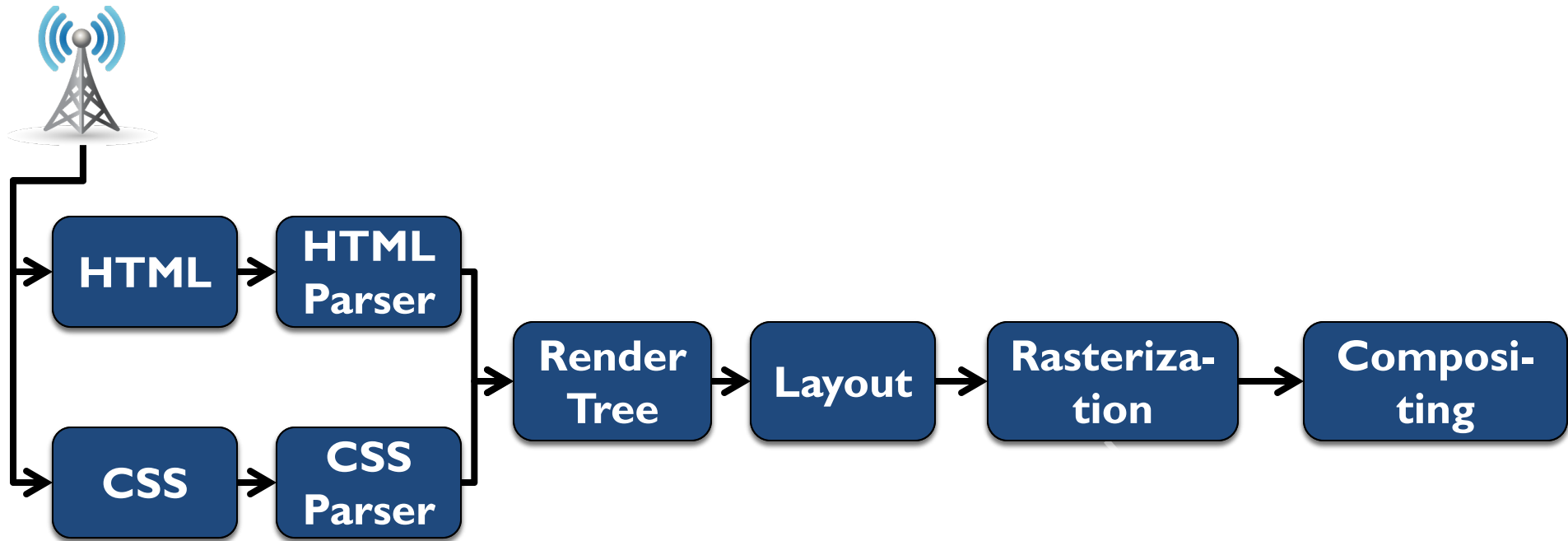
VP9



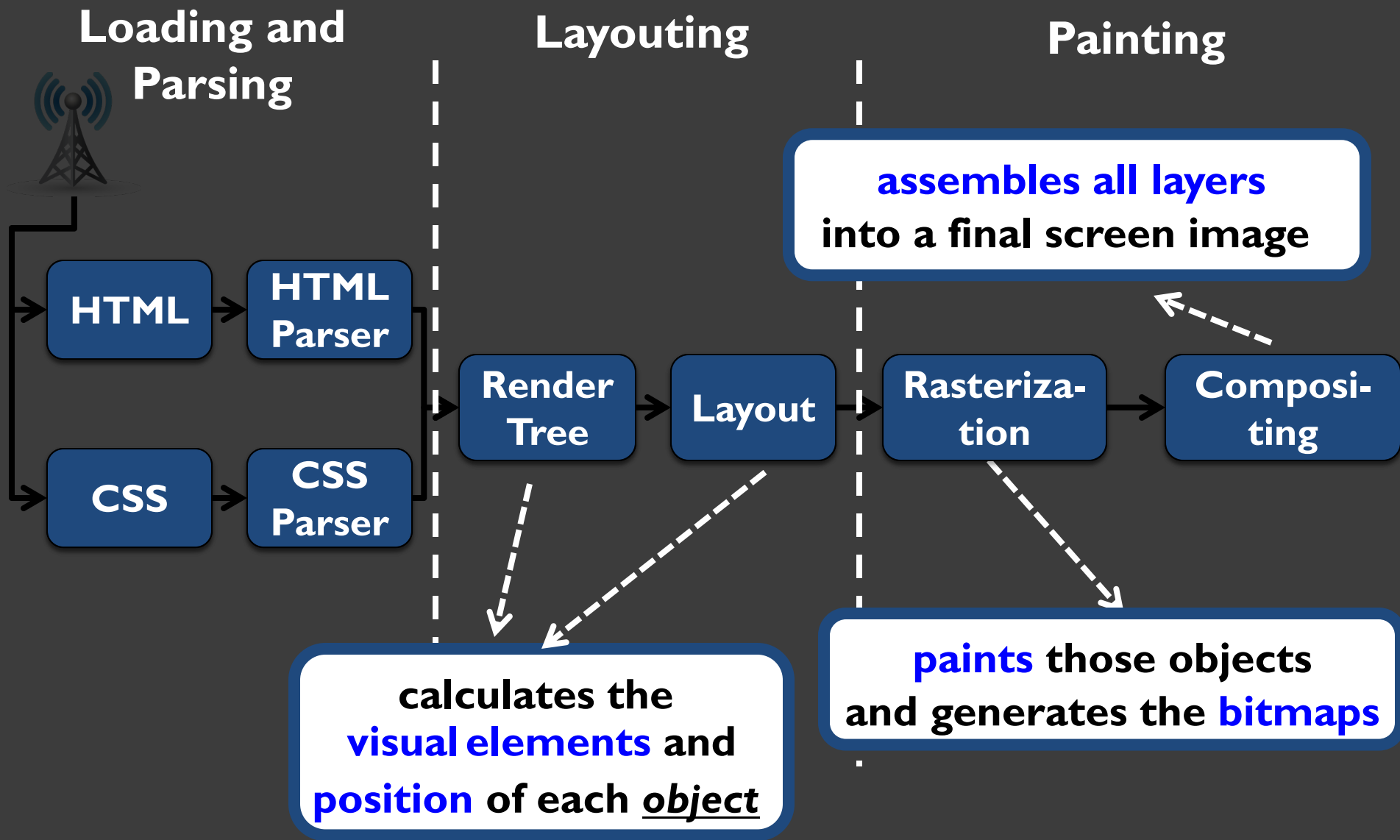
Video Capture

Google's video codec

How Chrome Renders a Web Page



How Chrome Renders a Web Page



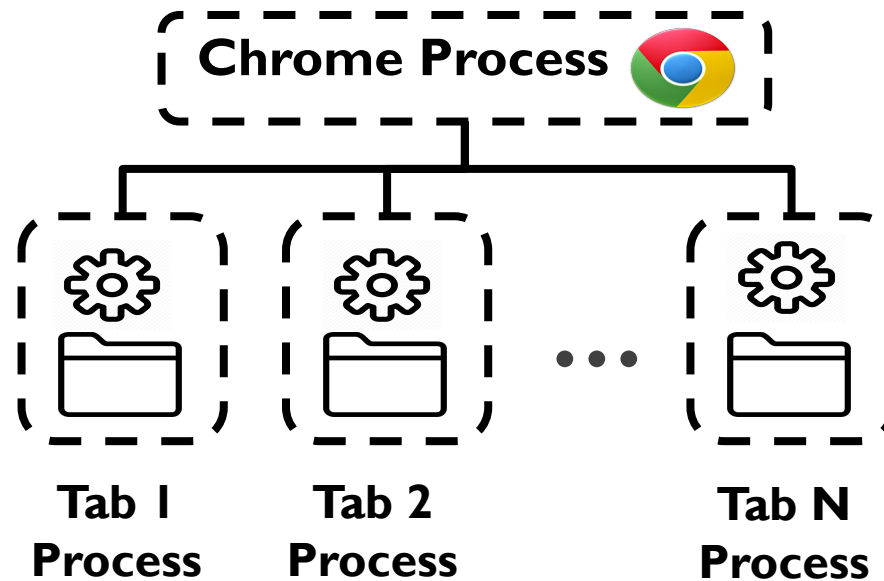
Browser Analysis

- 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 page loading

Tab Switching

What Happens During Tab Switching?

- Chrome employs a **multi-process** architecture
 - Each tab is a separate process

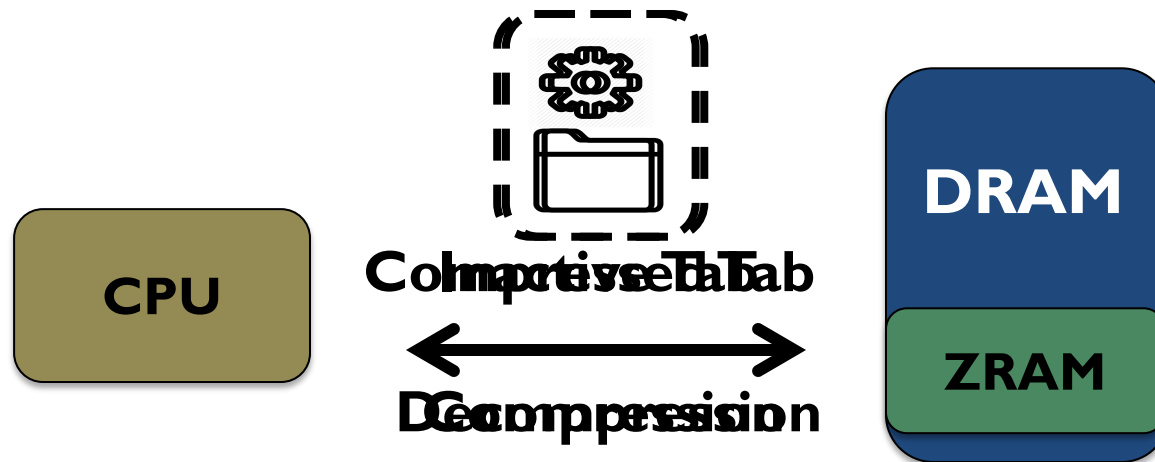


- Main operations during **tab switching**:
 - Context switch
 - Load the new page

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**



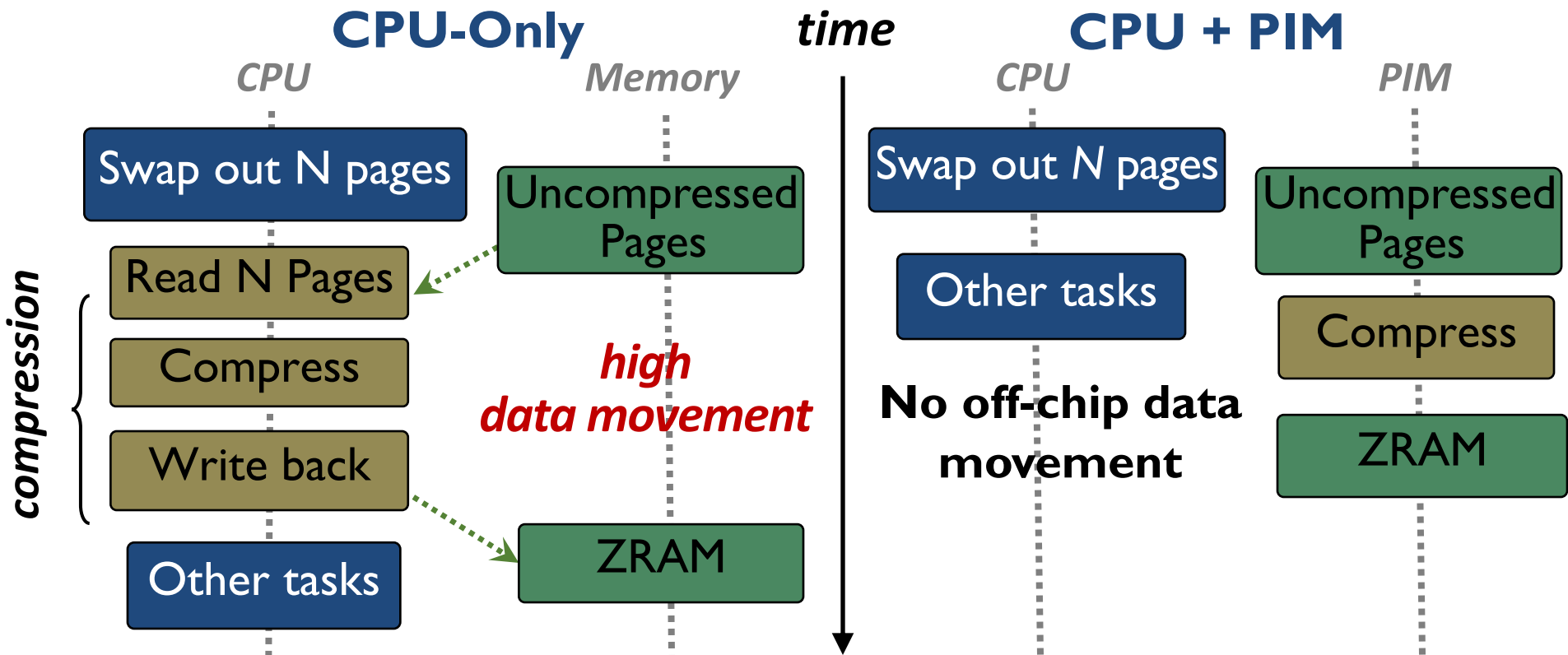
Data Movement Study

- 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?



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

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 Boroumand¹

Saugata Ghose¹

Youngsok Kim²

Rachata Ausavarungnirun¹

Eric Shiu³

Rahul Thakur³

Daehyun Kim^{4,3}

Aki Kuusela³

Allan Knies³

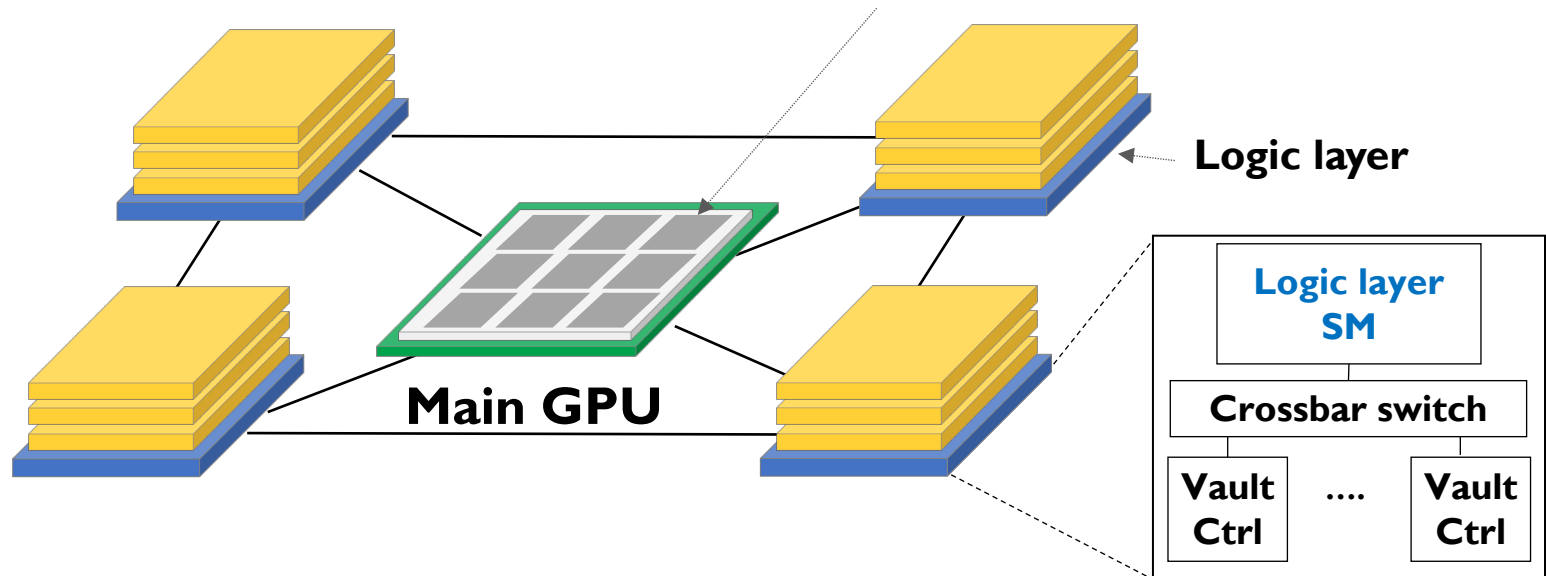
Parthasarathy Ranganathan³

Onur Mutlu^{5,1}

Truly Distributed GPU Processing with PIM?

**3D-stacked memory
(memory stack)**

SM (Streaming Multiprocessor)



```
__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;
```

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, 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 Pattnaik¹ Xulong Tang¹ Adwait Jog² Onur Kayiran³
Asit K. Mishra⁴ Mahmut T. Kandemir¹ Onur Mutlu^{5,6} Chita R. Das¹

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

Accelerating Climate Modeling

- Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gómez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal,
"NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling"
Proceedings of the 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, September 2020.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Lightning Talk Slides \(pptx\)](#)] [[pdf](#)]
[[Talk Video](#) (23 minutes)]
Nominated for the Stamatis Vassiliadis Memorial Award.

NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling

Gagandeep Singh^{a,b,c} Dionysios Diamantopoulos^c Christoph Hagleitner^c Juan Gómez-Luna^b
Sander Stuijk^a Onur Mutlu^b Henk Corporaal^a
^aEindhoven University of Technology ^bETH Zürich ^cIBM Research Europe, Zurich

Accelerating Approximate String Matching

- Damla Senol Cali, Gurpreet S. Kalsi, Zulal Bingol, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Nori, Allison Scibisz, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu, **"GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis"**

Proceedings of the 53rd International Symposium on Microarchitecture (MICRO), Virtual, October 2020.

[[Lighting Talk Video](#) (1.5 minutes)]

[[Lightning Talk Slides \(pptx\)](#) ([pdf](#))]

[[Talk Video](#) (18 minutes)]

[[Slides \(pptx\)](#) ([pdf](#))]

GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali[†]✕ Gurpreet S. Kalsi[✕] Zülal Bingöl[▽] Can Firtina[◇] Lavanya Subramanian[‡] Jeremie S. Kim[◇]†
Rachata Ausavarungnirun[○] Mohammed Alser[◇] Juan Gomez-Luna[◇] Amirali Boroumand[†] Anant Nori[✕]
Allison Scibisz[†] Sreenivas Subramoney[✕] Can Alkan[▽] Saugata Ghose^{*†} Onur Mutlu[◇]†[▽]

[†]Carnegie Mellon University [✕]Processor Architecture Research Lab, Intel Labs [▽]Bilkent University [◇]ETH Zürich

[‡]Facebook [○]King Mongkut's University of Technology North Bangkok ^{*}University of Illinois at Urbana-Champaign

Accelerating Time Series Analysis

- Ivan Fernandez, Ricardo Quisiant, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, Eladio Gutiérrez, Oscar Plata, and Onur Mutlu,
"NATSA: A Near-Data Processing Accelerator for Time Series Analysis"
Proceedings of the 38th IEEE International Conference on Computer Design (ICCD), Virtual, October 2020.

NATSA: A Near-Data Processing Accelerator for Time Series Analysis

| | | | |
|--|---|----------------------------------|-----------------------------|
| Ivan Fernandez [§] | Ricardo Quisiant [§] | Christina Giannoula [†] | Mohammed Alser [‡] |
| Juan Gómez-Luna [‡] | Eladio Gutiérrez [§] | Oscar Plata [§] | Onur Mutlu [‡] |
| [§] <i>University of Malaga</i> | [†] <i>National Technical University of Athens</i> | [‡] <i>ETH Zürich</i> | |

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

PIM-Enabled Instructions

- Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoungh 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[†] Kiyoungh Choi

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

Seoul National University

[†]Carnegie Mellon University

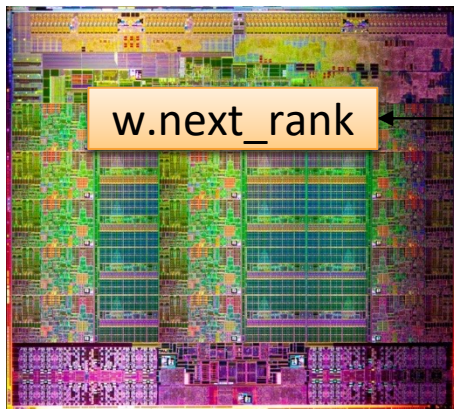
PEI: PIM-Enabled Instructions (Ideas)

- **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



Main Memory



64 bytes in
64 bytes out

Conventional Architecture

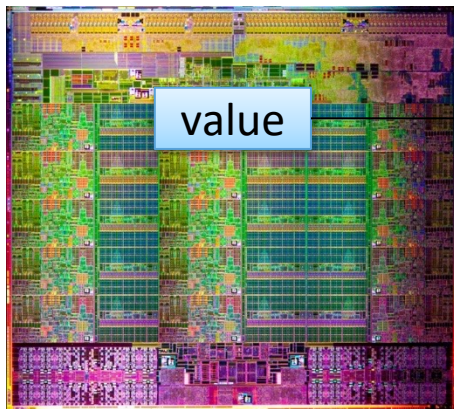
Simple PIM Operations as ISA Extensions (III)

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

pim.add r1, (r2)

__pim_add(&w.next_rank, value);

Host Processor



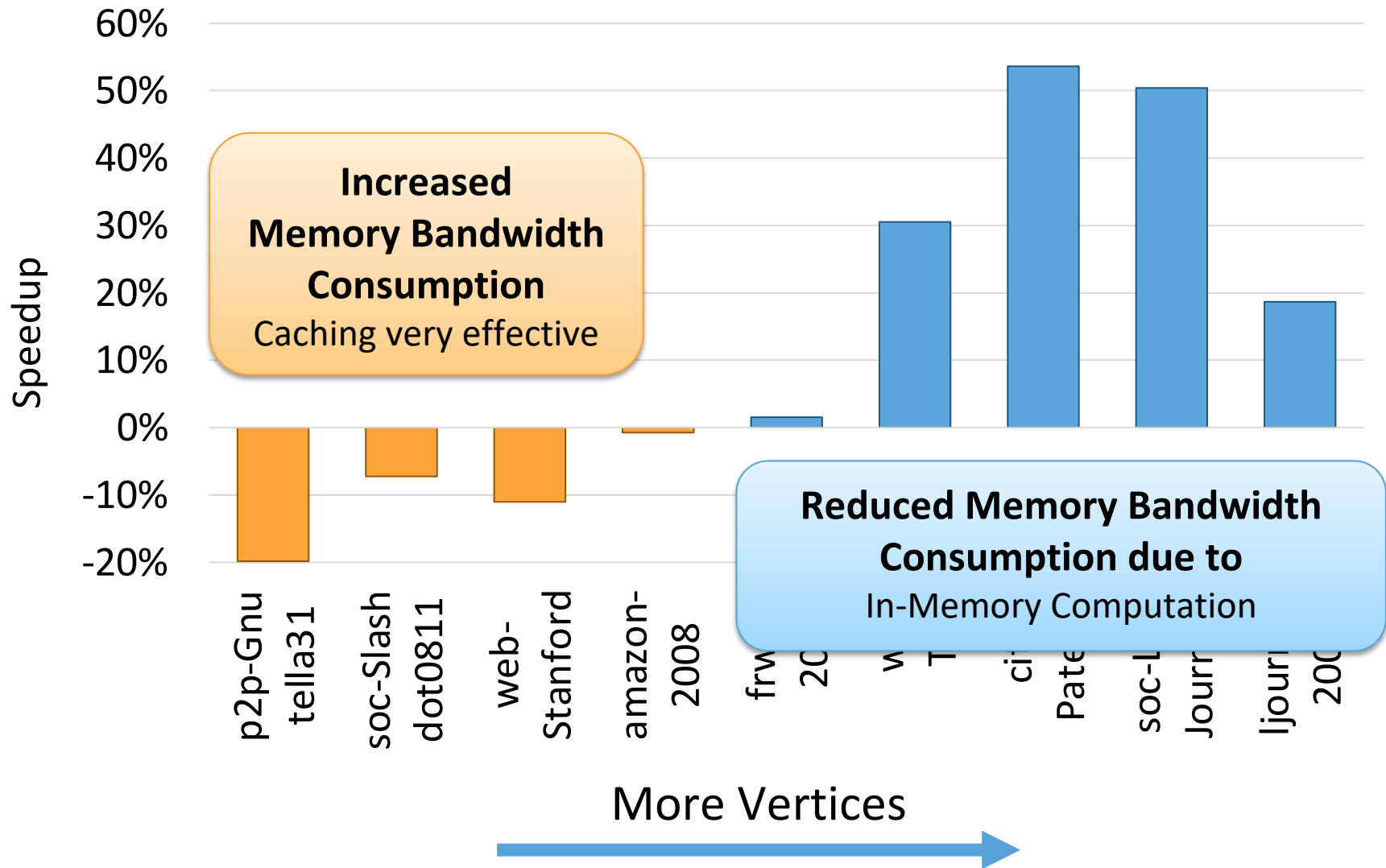
Main Memory



8 bytes in
0 bytes out

In-Memory Addition

Always Executing in Memory? Not A Good Idea



PEI: PIM-Enabled Instructions (Example)

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

pim.add r1, (r2)

__pim_add(&w.next_rank, value);

pfence

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
- 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

Table 2: Baseline Simulation Configuration

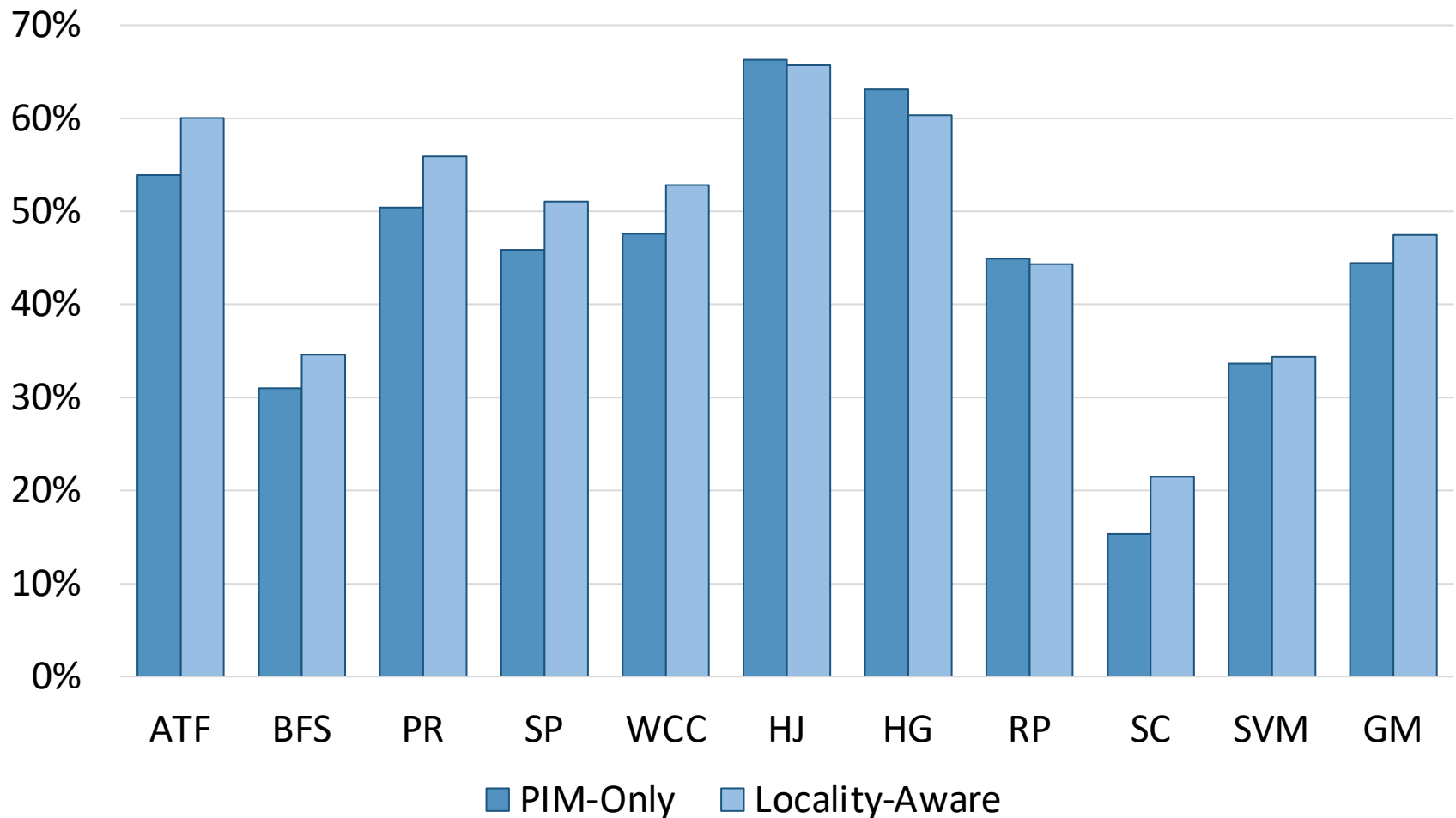
| Component | Configuration |
|------------------|---|
| Core | 16 out-of-order cores, 4 GHz, 4-issue |
| L1 I/D-Cache | Private, 32 KB, 4/8-way, 64 B blocks, 16 MSHRs |
| L2 Cache | Private, 256 KB, 8-way, 64 B blocks, 16 MSHRs |
| L3 Cache | Shared, 16 MB, 16-way, 64 B blocks, 64 MSHRs |
| On-Chip 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] |

Evaluated Data-Intensive Applications

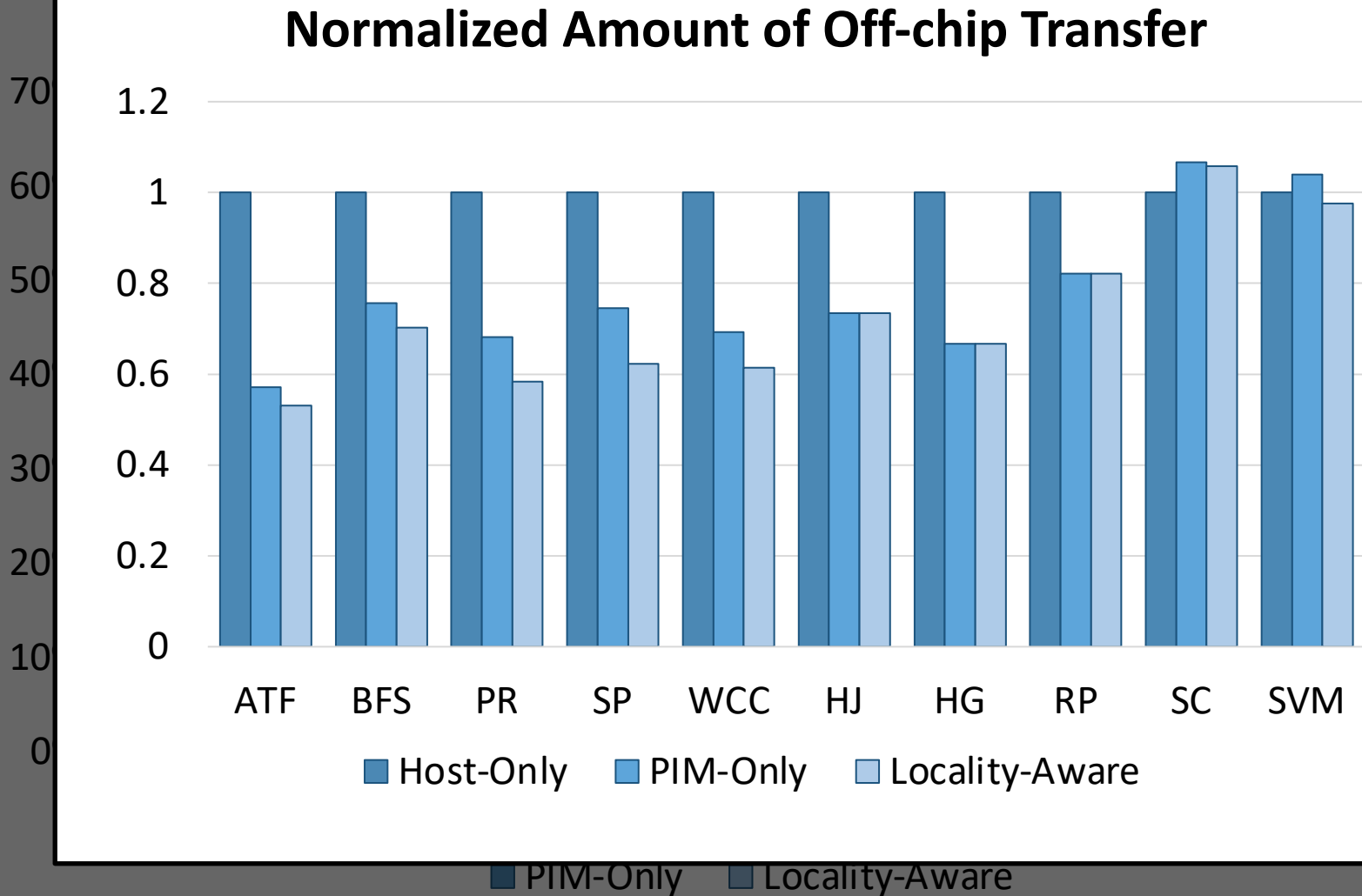
- Ten emerging data-intensive workloads
 - Large-scale graph processing
 - Average teenage follower, BFS, PageRank, single-source shortest path, weakly connected components
 - In-memory data analytics
 - Hash join, histogram, radix partitioning
 - Machine learning and data mining
 - Streamcluster, SVM-RFE
- Three input sets (small, medium, large) for each workload to show the impact of data locality

PEI Performance Delta: Large Data Sets

(Large Inputs, Baseline: Host-Only)

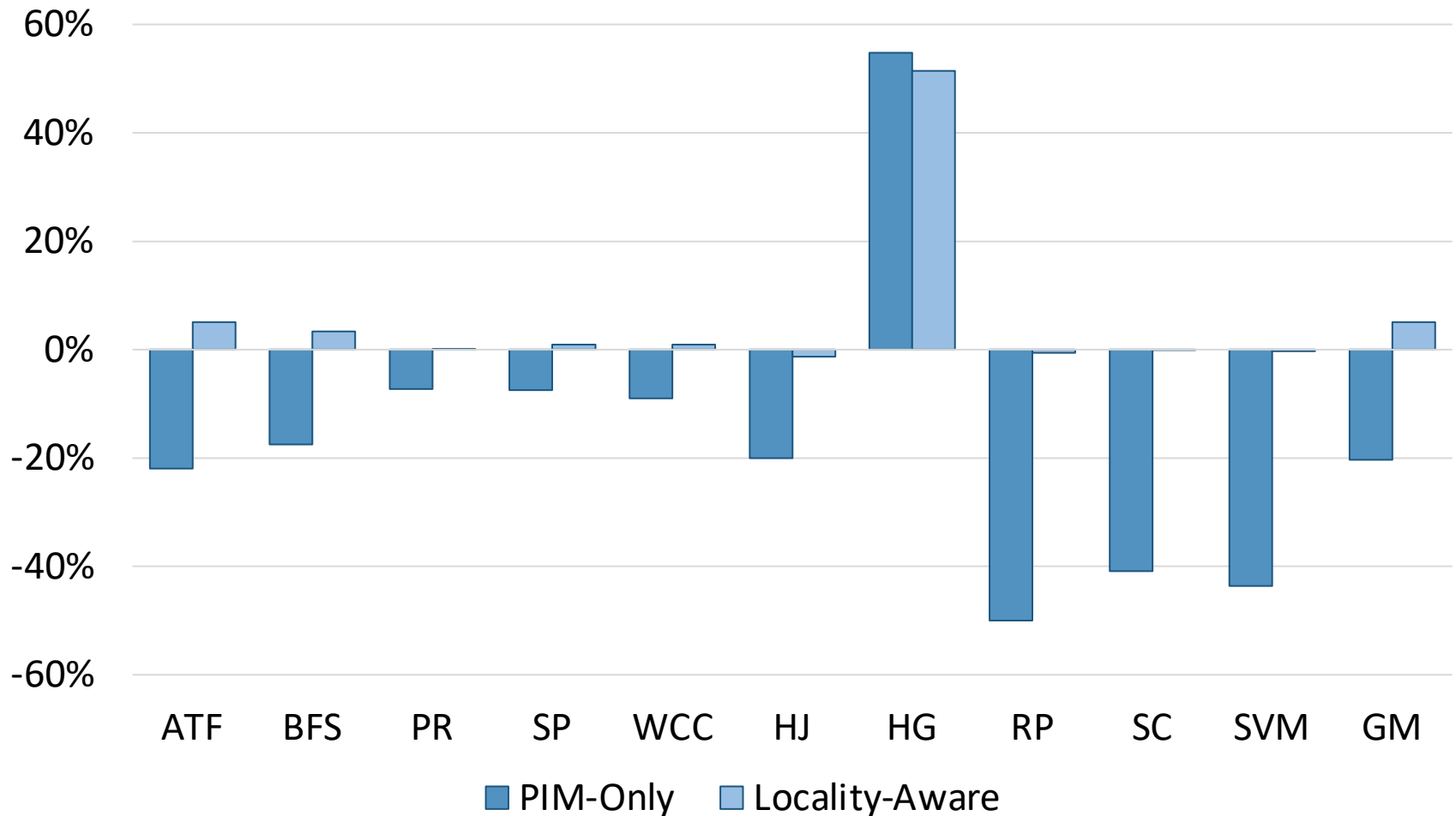


PEI Performance: Large Data Sets

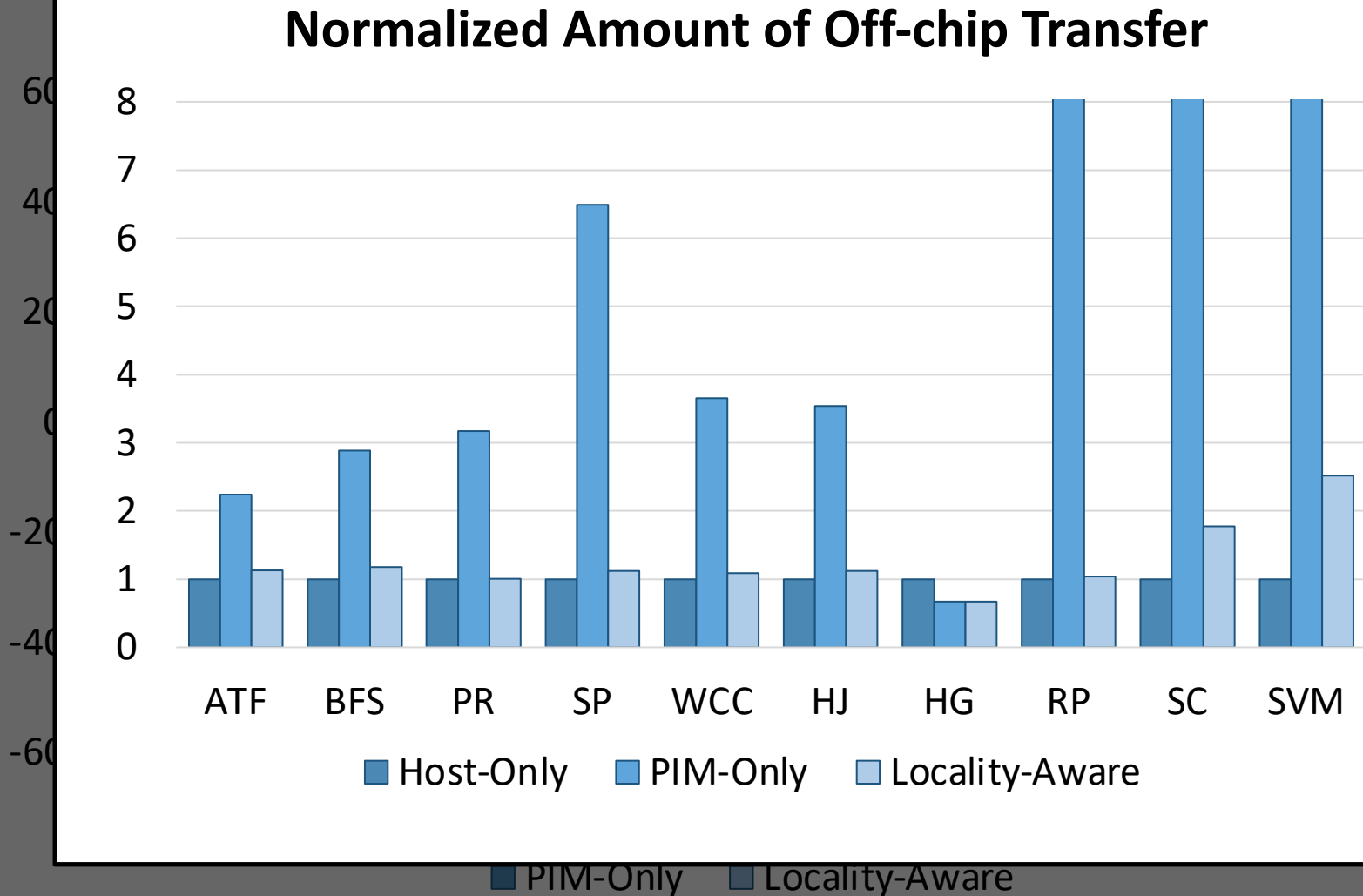


PEI Performance Delta: Small Data Sets

(Small Inputs, Baseline: Host-Only)

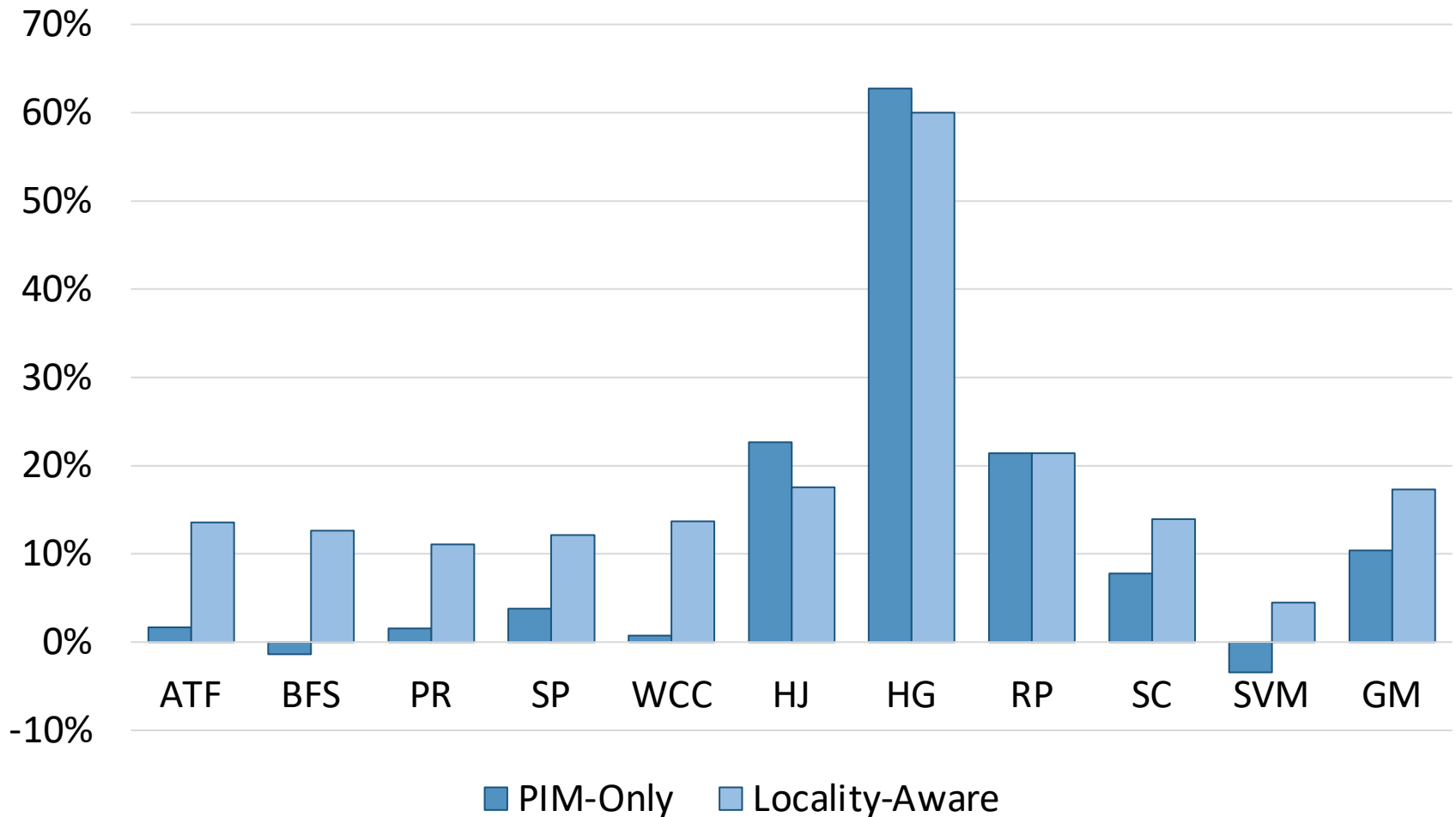


PEI Performance: Small Data Sets

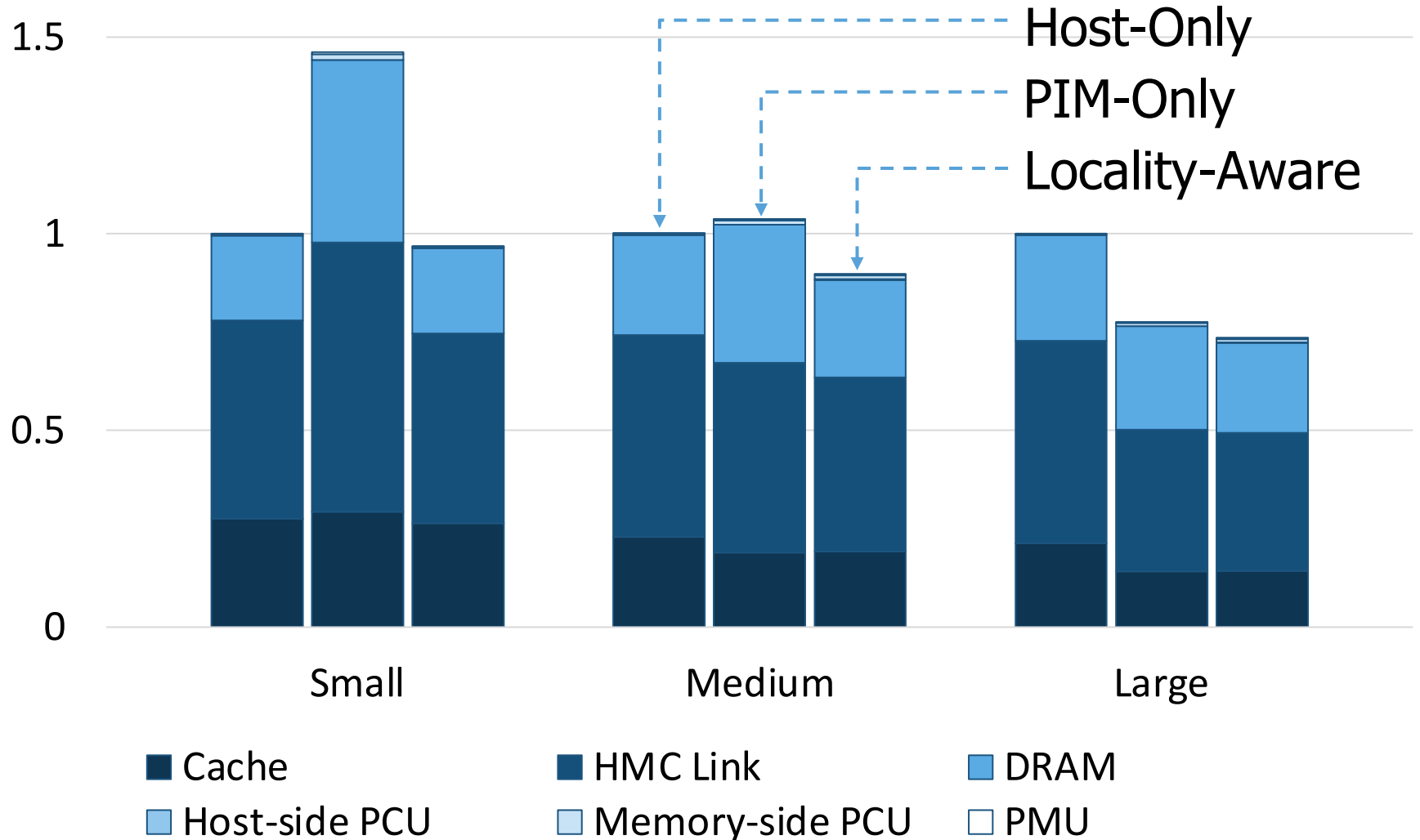


PEI Performance Delta: Medium Data Sets

(Medium Inputs, Baseline: Host-Only)



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

Simpler PIM: PIM-Enabled Instructions

- Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoungh 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[†] Kiyoungh Choi

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

Seoul National University

[†]Carnegie Mellon University

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

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
 - Processing using Memory
 - Processing near Memory
- How to Enable Adoption of Processing in Memory
- Conclusion

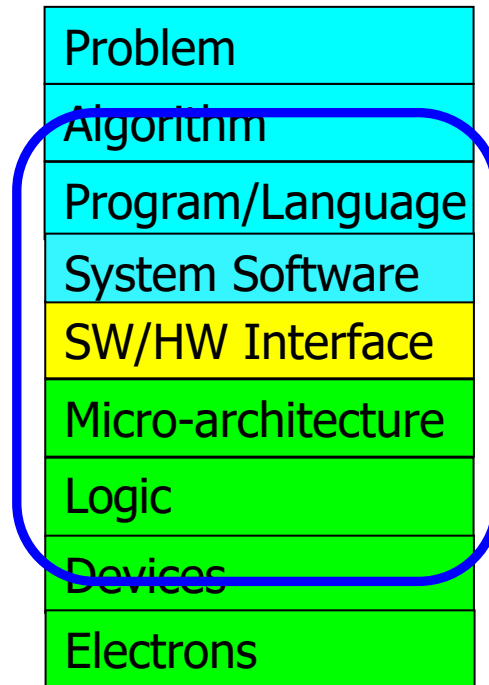
How to Enable Adoption of Processing in Memory

Potential Barriers to Adoption of PIM

1. **Applications & software** for PIM
2. Ease of **programming** (interfaces and compiler/HW support)
3. **System** and **security** support: coherence, synchronization, virtual memory, isolation, communication interfaces, ...
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



We can get there step by step

PIM Review and Open Problems

A Modern Primer on Processing in Memory

Onur Mutlu^{a,b}, Saugata Ghose^{b,c}, Juan Gómez-Luna^a, Rachata Ausavarungnirun^d

SAFARI Research Group

^a*ETH Zürich*

^b*Carnegie Mellon University*

^c*University of Illinois at Urbana-Champaign*

^d*King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,

"A Modern Primer on Processing in Memory"

*Invited Book Chapter in **Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann**, Springer, to be published in 2021.*

| | | |
|-----------|--|-----------|
| 1 | Introduction | 2 |
| 2 | Major Trends Affecting Main Memory | 4 |
| 3 | The Need for Intelligent Memory Controllers to Enhance Memory Scaling | 6 |
| 4 | Perils of Processor-Centric Design | 9 |
| 5 | Processing-in-Memory (PIM): Technology Enablers and Two Approaches | 11 |
| 5.1 | New Technology Enablers: 3D-Stacked Memory and Non-Volatile Memory . . . | 12 |
| 5.2 | Two Approaches: Processing Using Memory (PUM) vs. Processing Near Memory (PNM) | 13 |
| 6 | Processing Using Memory (PUM) | 14 |
| 6.1 | RowClone | 14 |
| 6.2 | Ambit | 15 |
| 6.3 | SIMDRAM | 17 |
| 6.4 | Gather-Scatter DRAM | 18 |
| 6.5 | In-DRAM Security Primitives | 18 |
| 7 | Processing Near Memory (PNM) | 20 |
| 7.1 | Tesseract: Coarse-Grained Application-Level PNM Acceleration of Graph Processing | 20 |
| 7.2 | Function-Level PNM Acceleration of Mobile Consumer Workloads | 21 |
| 7.3 | Programmer-Transparent Function-Level PNM Acceleration of GPU Applications | 22 |
| 7.4 | Instruction-Level PNM Acceleration with PIM-Enabled Instructions (PEI) . . | 23 |
| 7.5 | Function-Level PNM Acceleration of Genome Analysis Workloads | 24 |
| 7.6 | Application-Level PNM Acceleration of Time Series Analysis | 26 |
| 8 | Enabling the Adoption of PIM | 26 |
| 8.1 | Programming Models and Code Generation for PIM | 26 |
| 8.2 | PIM Runtime: Scheduling and Data Mapping | 27 |
| 8.3 | Memory Coherence | 29 |
| 8.4 | Virtual Memory Support | 30 |
| 8.5 | Data Structures for PIM | 30 |
| 8.6 | Benchmarks and Simulation Infrastructures | 31 |
| 8.7 | Real PIM Hardware Systems and Prototypes | 33 |
| 8.8 | Security Considerations | 36 |
| 9 | Other Resources on PIM | 37 |
| 10 | Conclusion and Future Outlook | 37 |

Main memory, built using the Dynamic Random Access Memory (DRAM) technology, is a major component in nearly all computing systems, including servers, cloud platforms, mobile/embedded devices, and sensor systems. Across all of these systems, the data working set sizes of modern applications are rapidly growing, while the need for fast analysis of such data is increasing. Thus, main memory is becoming an increasingly significant bottleneck across a wide variety of computing systems and applications [1–26]. Alleviating the main memory bottleneck requires the memory capacity, energy, cost, and performance to all scale in an efficient manner across technology generations. Unfortunately, it has become increasingly difficult in recent years, especially the past decade, to scale all of these dimensions [1, 2, 27–59], and thus the main memory bottleneck has been worsening.

A major reason for the main memory bottleneck is the high energy and latency cost associated with *data movement*. In modern computers, to perform any operation on data that resides in main memory, the processor must retrieve the data from main memory. This requires the memory controller to issue commands to a DRAM module across a relatively slow and power-hungry off-chip bus (known as the *memory channel*). The DRAM module sends the requested data across the memory channel, after which the data is placed in the caches and registers. The CPU can perform computation on the data once the data is in its registers. Data movement from the DRAM to the CPU incurs long latency and consumes a significant amount of energy [7–9, 60–64]. These costs are often exacerbated by the fact that much of the data brought into the caches is *not reused* by the CPU [62, 63, 65, 66], providing little benefit in return for the high latency and energy cost.

The cost of data movement is a fundamental issue with the *processor-centric* nature of contemporary computer systems. The CPU is considered to be the master in the system, and computation is performed only in the processor (and accelerators). In contrast, data storage and communication units, including the main memory, are treated as unintelligent workers that are incapable of computation. As a result of this processor-centric design paradigm, data moves a lot in the system between the computation units and communication/storage units so that computation can be done on it. With the increasingly *data-centric* nature of contemporary and emerging applications, the processor-centric design paradigm leads to great inefficiency in performance, energy and cost. For example, most of the real estate within a single compute

PIM Review and Open Problems (II)

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 (III)

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,

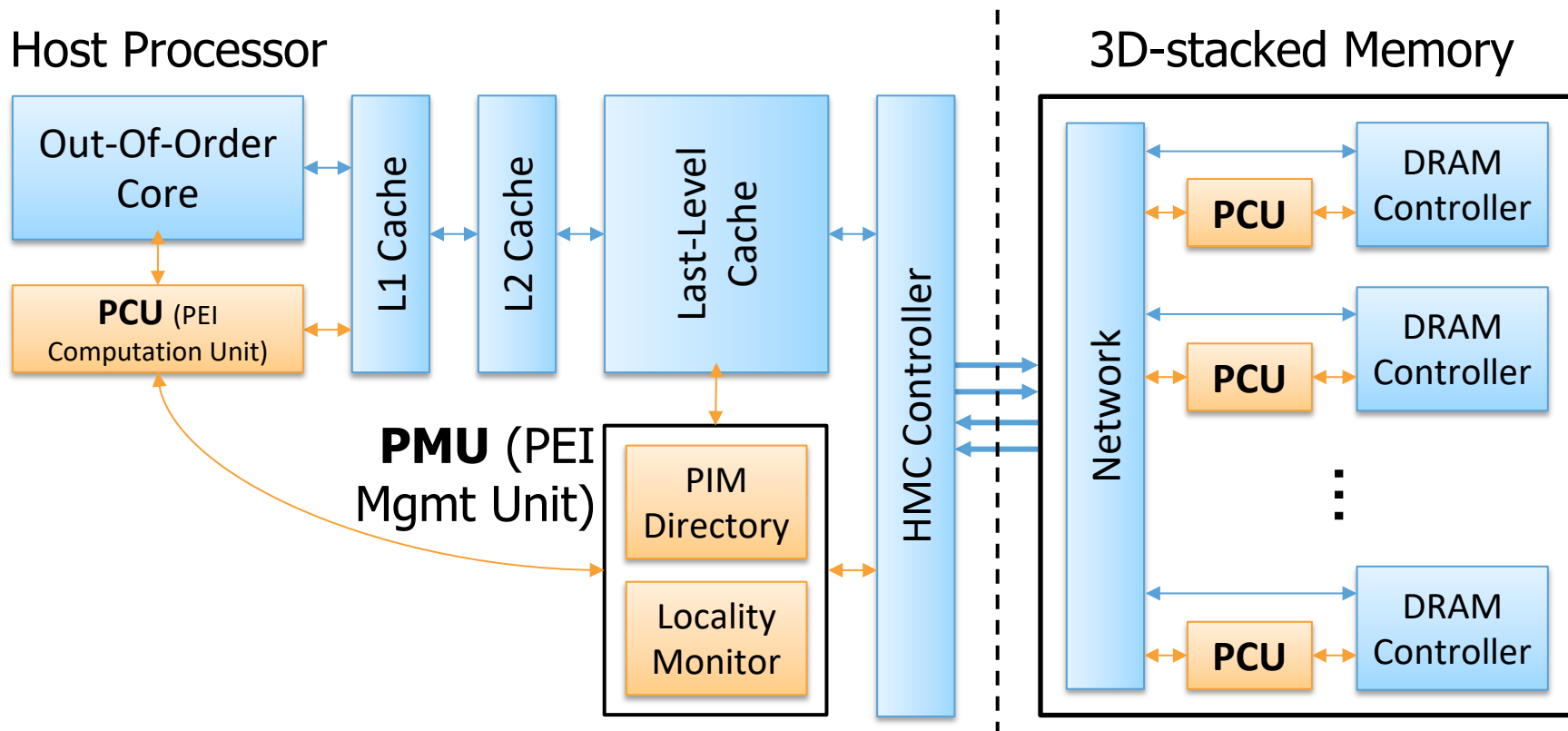
"Processing-in-Memory: A Workload-Driven Perspective"

Invited Article in IBM Journal of Research & Development, Special Issue on Hardware for Artificial Intelligence, to appear in November 2019.

[Preliminary arXiv version]

PIM Runtime: Scheduling and Data Mapping

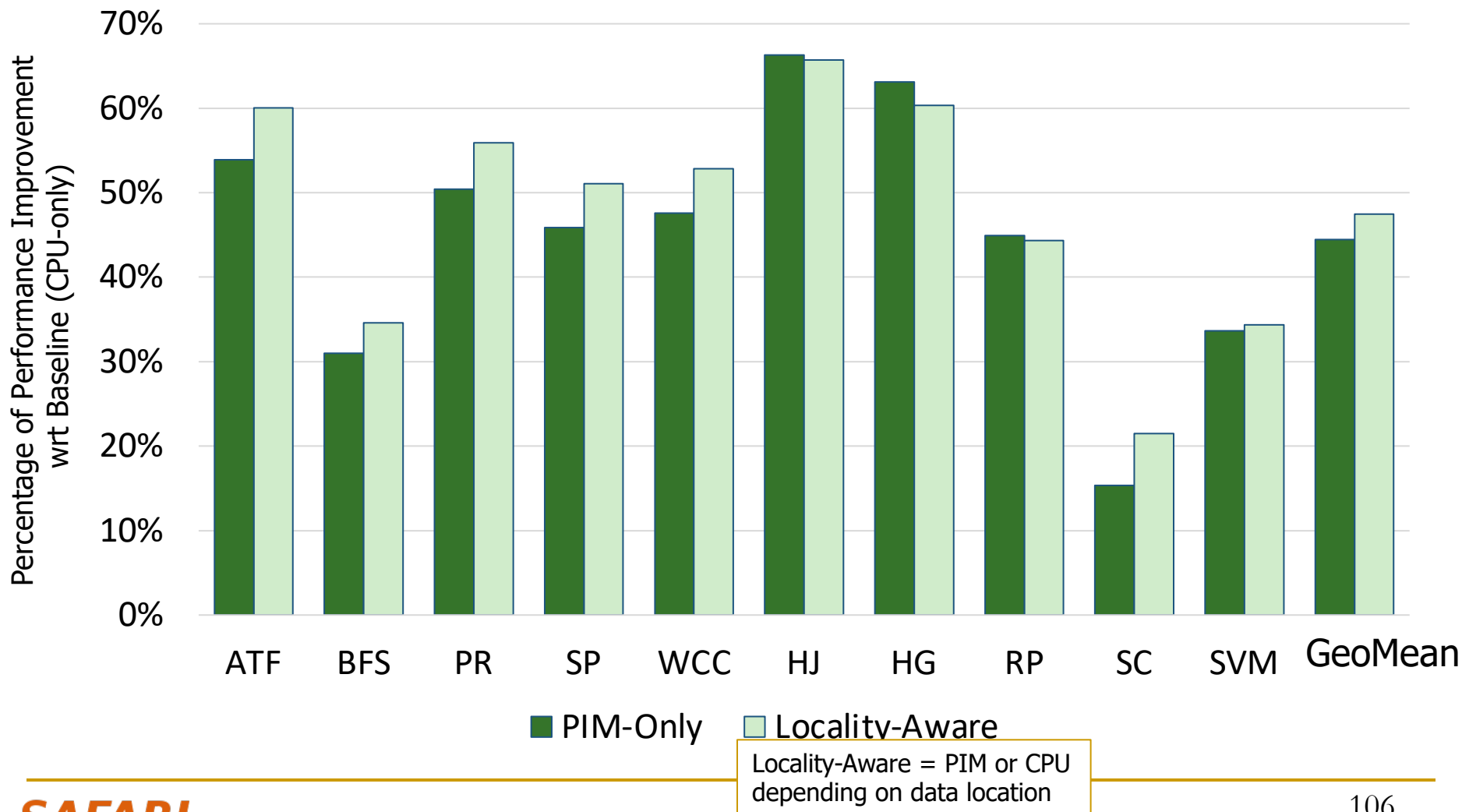
Example PEI Microarchitecture



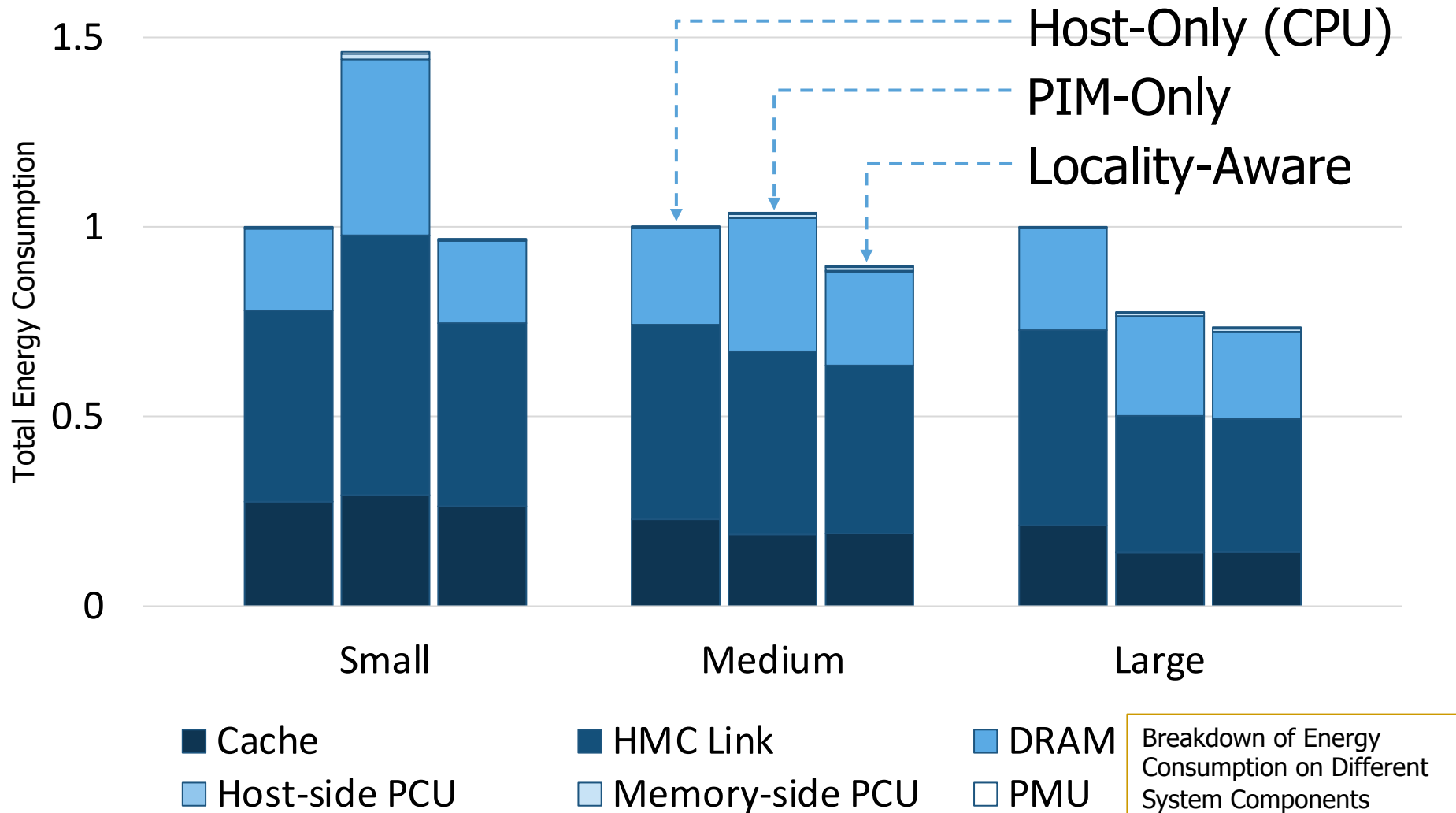
Example PEI uArchitecture

PEI Performance Delta: Large Data Sets

(Large Inputs, Baseline: CPU-Only)



PEI Energy Consumption



More on PIM-Enabled Instructions

- Junwhan Ahn, Sungjoo Yoo, Onur Mutlu, and Kiyoun 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[†] Kiyoun Choi

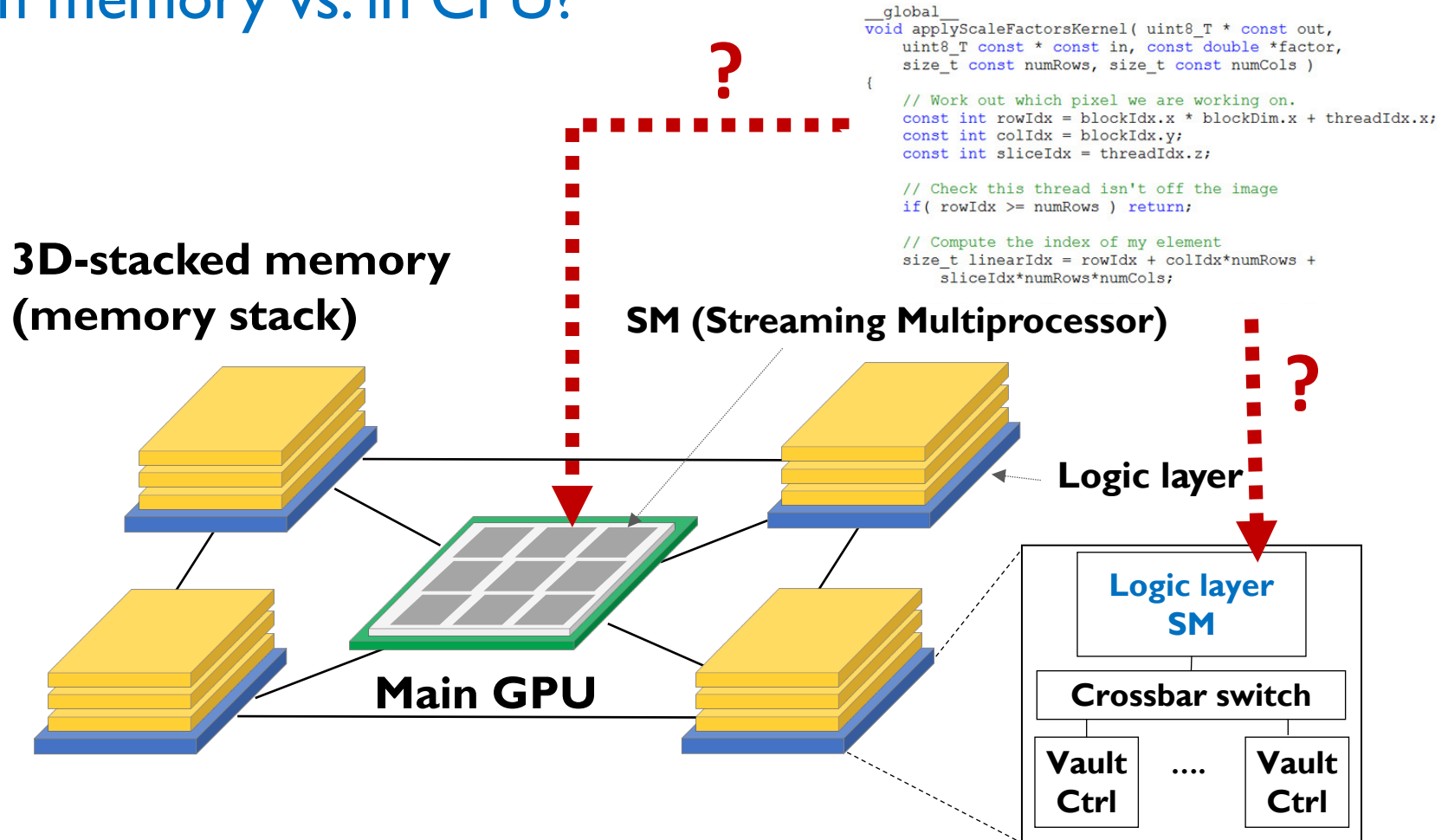
junwhan@snu.ac.kr, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr

Seoul National University

[†]Carnegie Mellon University

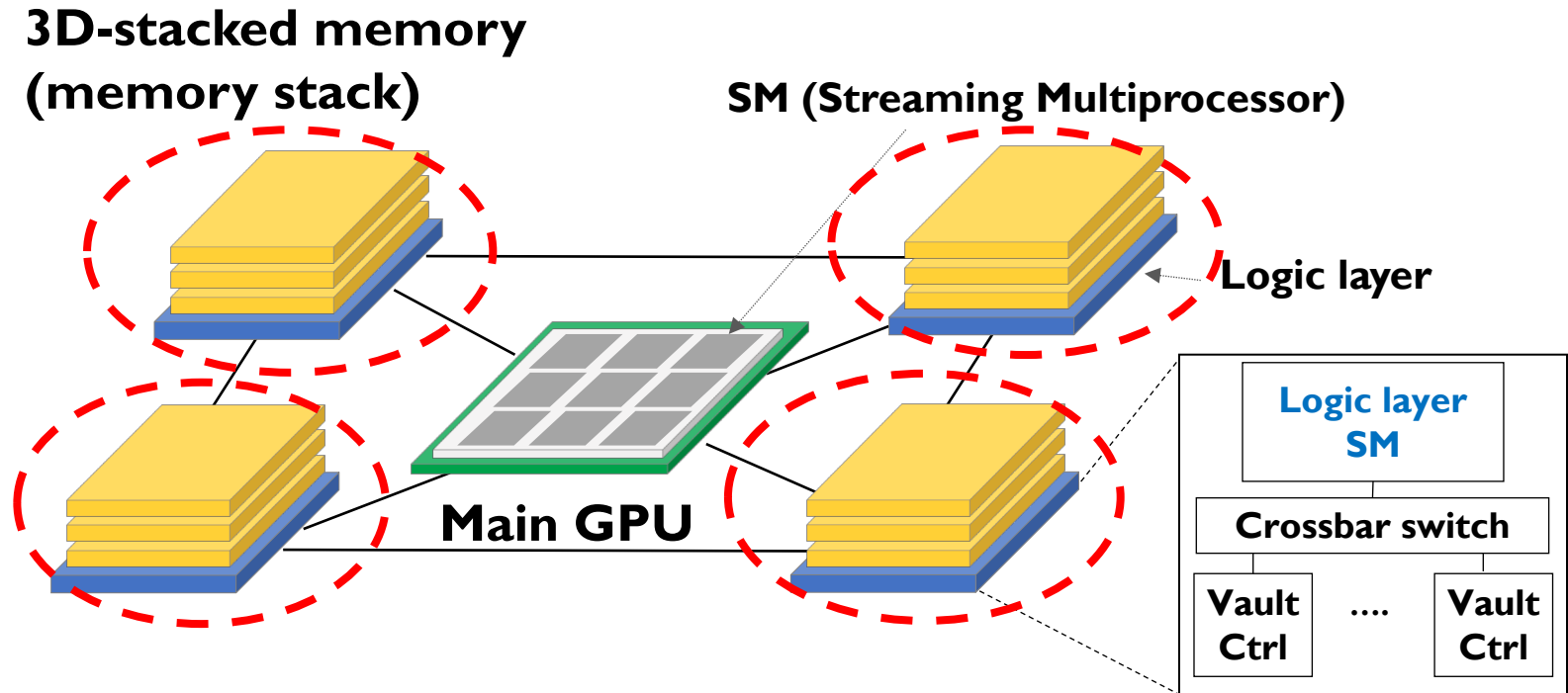
Key Challenge 1: Code Mapping

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



Key Challenge 2: Data Mapping

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



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 Pattnaik¹ Xulong Tang¹ Adwait Jog² Onur Kayiran³
Asit K. Mishra⁴ Mahmut T. Kandemir¹ Onur Mutlu^{5,6} Chita R. Das¹
¹Pennsylvania State University ²College of William and Mary
³Advanced Micro Devices, Inc. ⁴Intel Labs ⁵ETH Zürich ⁶Carnegie 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](#)]

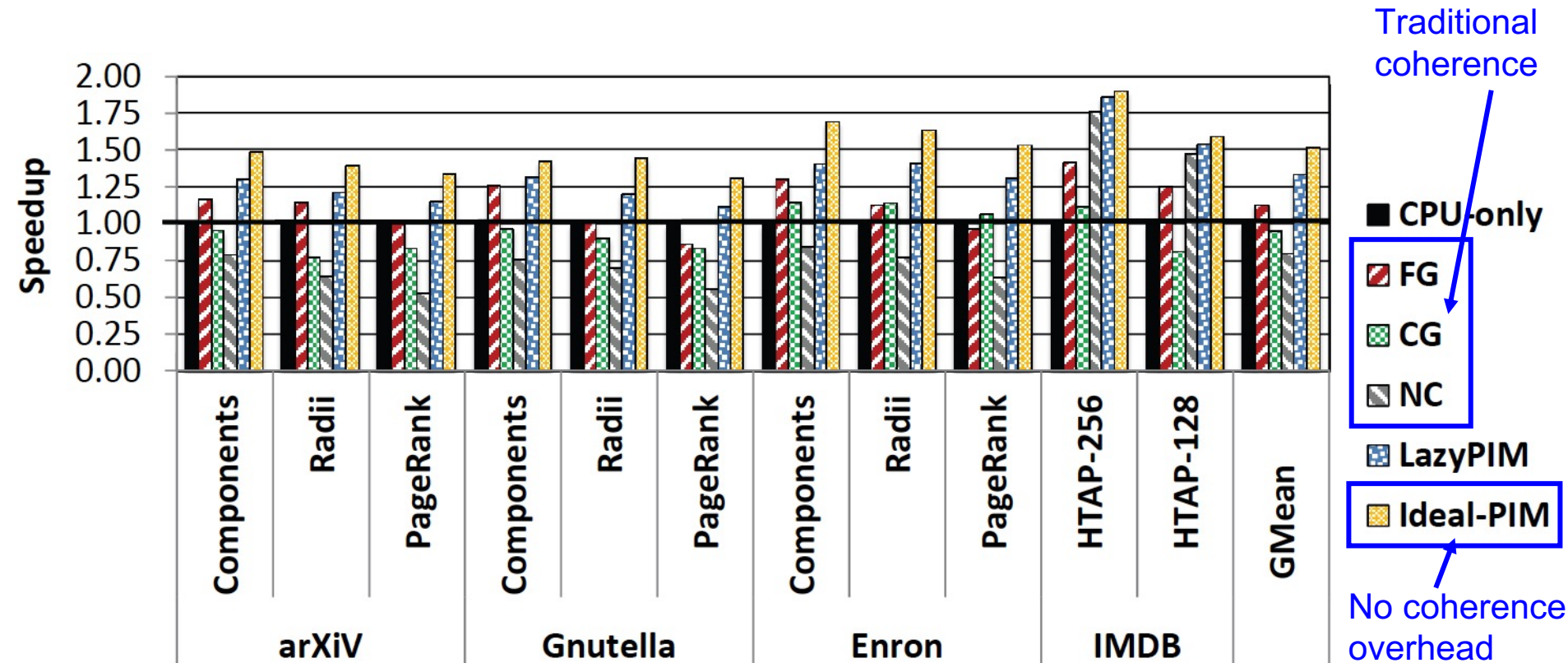
Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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

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

Memory Coherence

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

SAFARI



Carnegie Mellon



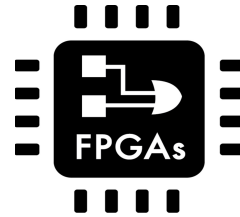
ETH zürich

Specialized Accelerators

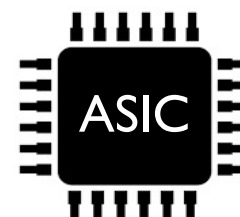
Specialized accelerators are now everywhere!



GPU

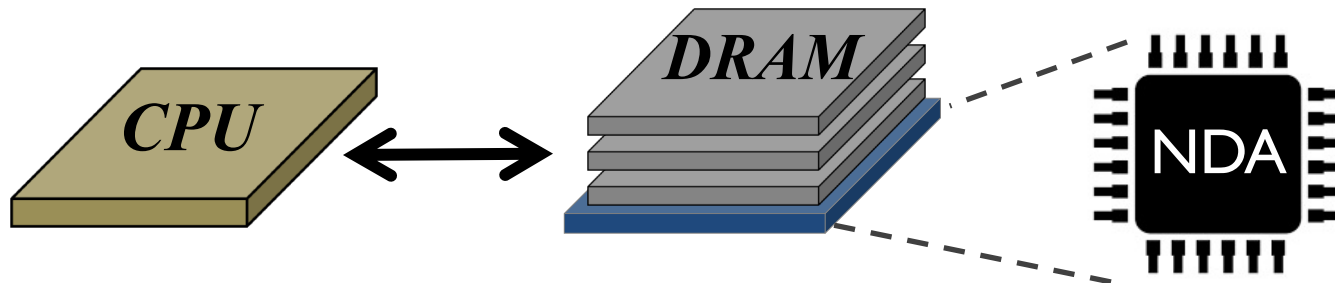


FPGA



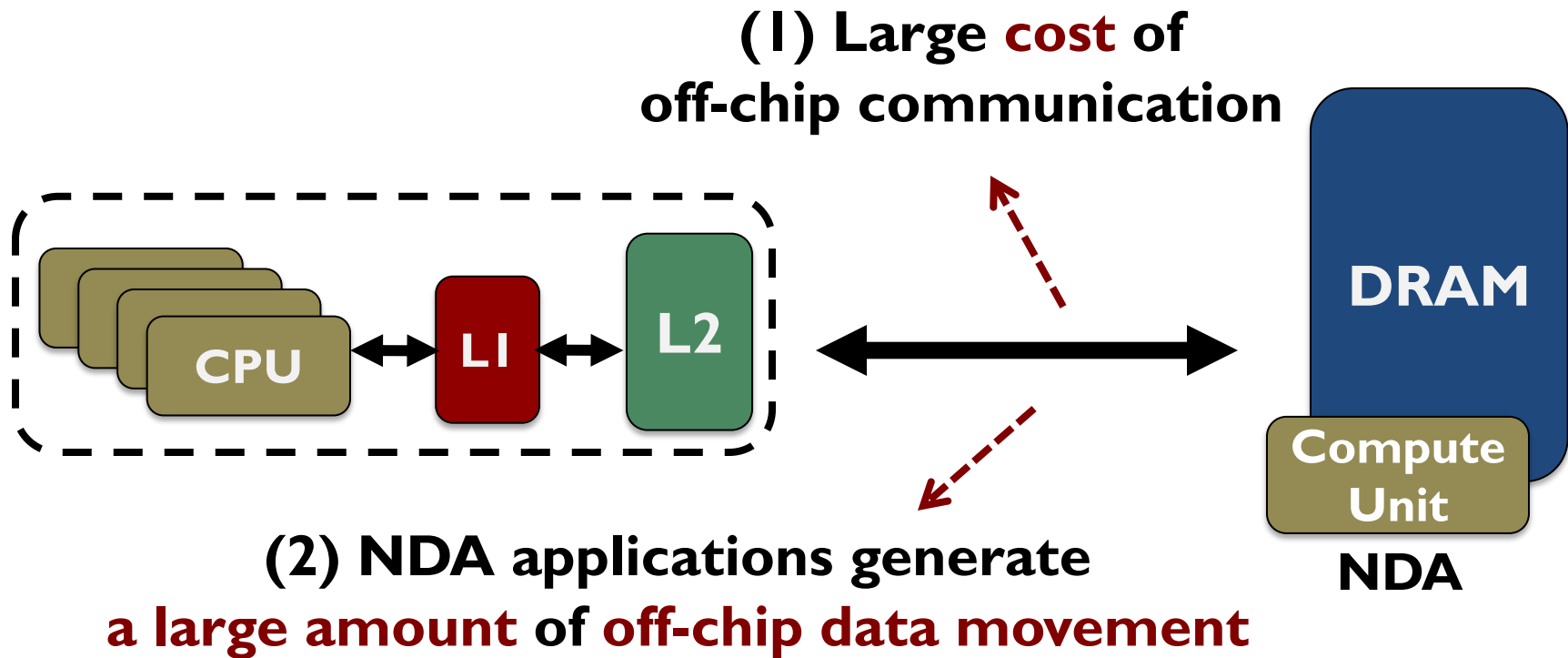
ASIC

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



Coherence For NDAs

Challenge: Coherence between NDAs and CPUs



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**

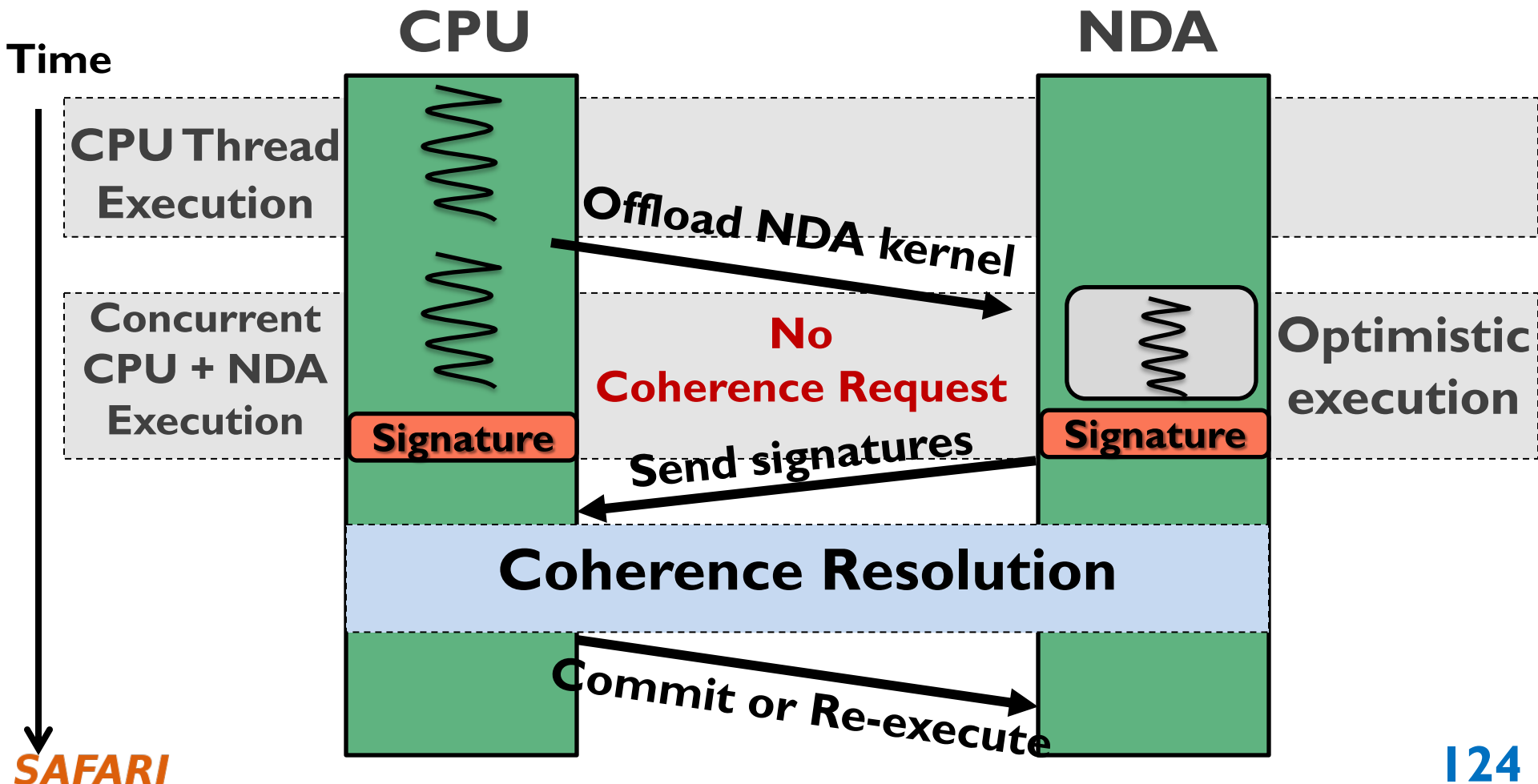
An Optimistic Approach

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

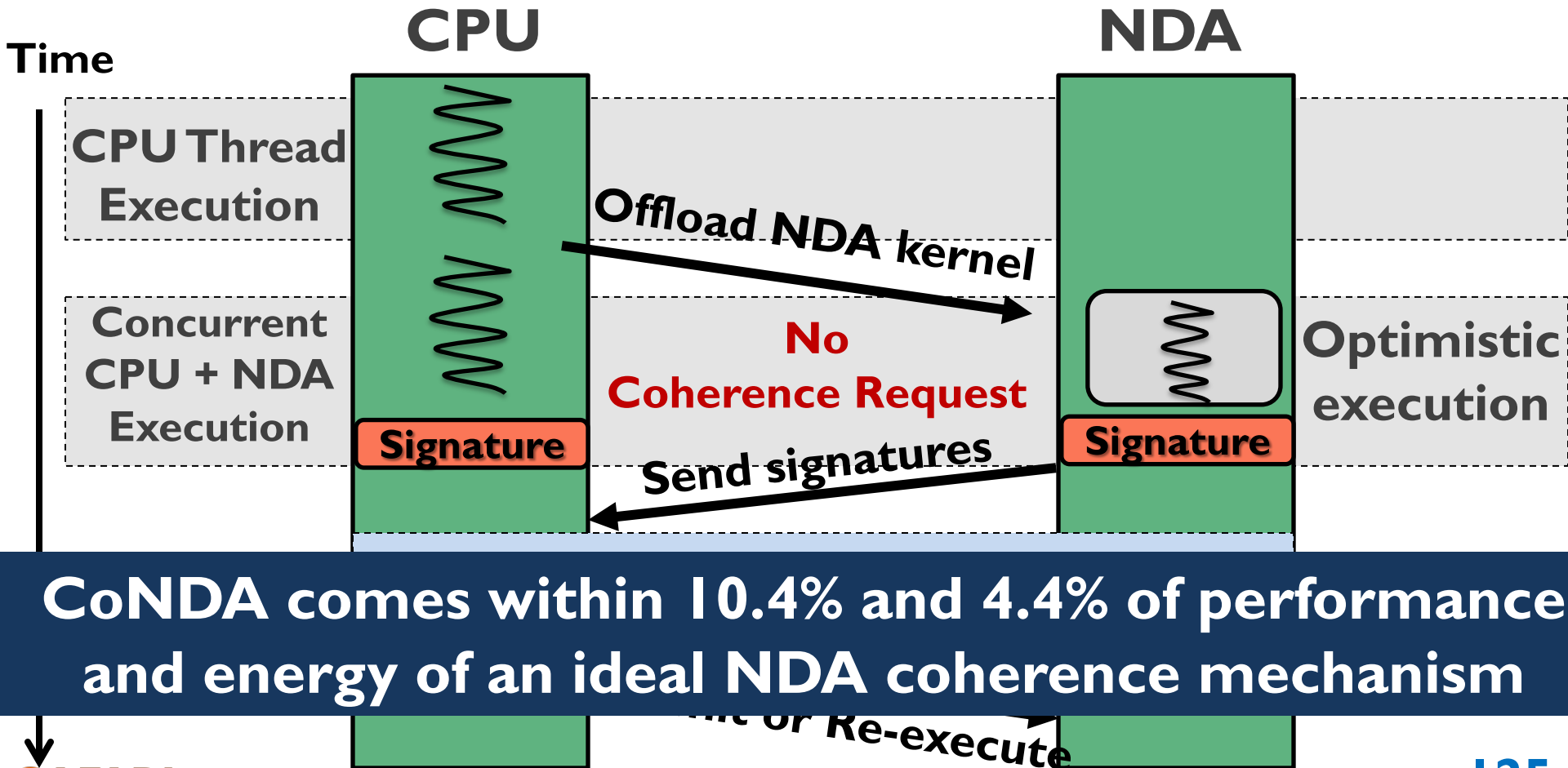
CoNDA

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



CoNDA

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



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

SAFARI



Carnegie Mellon



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.

Synchronization Support

How to Support Synchronization?

- Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, Onur Mutlu, **"SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures"**
Proceedings of the 27th International Symposium on High-Performance Computer Architecture (HPCA), Virtual, February-March 2021.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Short Talk Slides \(pptx\)](#)] [[pdf](#)]
[[Talk Video](#) (21 minutes)]
[[Short Talk Video](#) (7 minutes)]

SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures

Christina Giannoula^{†‡} Nandita Vijaykumar^{*‡} Nikela Papadopoulou[†] Vasileios Karakostas[†] Ivan Fernandez^{§‡}
Juan Gómez-Luna[‡] Lois Orosa[‡] Nectarios Koziris[†] Georgios Goumas[†] Onur Mutlu[‡]
[†]*National Technical University of Athens* [‡]*ETH Zürich* ^{*}*University of Toronto* [§]*University of Malaga*

SynCron

Efficient Synchronization Support for Near-Data-Processing Architectures



Christina Giannoula

Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu

SAFARI



ETH zürich



Executive Summary

Problem:

Synchronization support is **challenging** for NDP systems

Prior schemes are **not suitable** or **efficient** for NDP systems

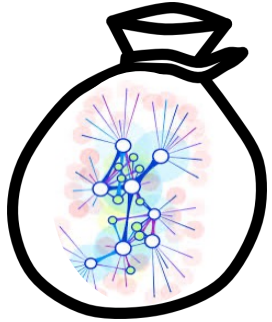
Contribution:

SynCron: the **first end-to-end** synchronization solution for NDP architectures

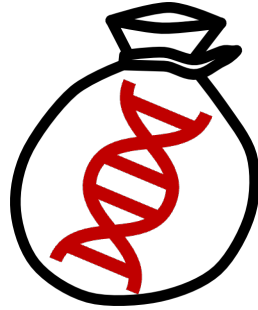
Key Results:

SynCron comes within **9.5%** and **6.2%** of performance and energy of an **Ideal** zero-overhead synchronization scheme

Synchronization is Necessary



Graph Analytics



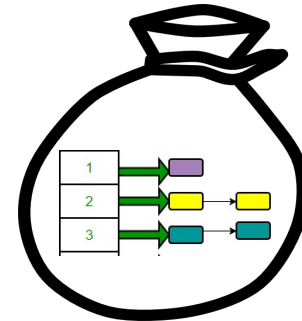
Bioinformatics



Databases



Image Processing



Concurrent Data Structures

Single Source Shortest Path (SSSP)

```
for v in Graph:
  for u in neighbors[v]:
    if distance[v] + edge_weight[v, u] < distance[u]
      lock_acquire(u)
      if distance[v] + edge_weight[v, u] < distance[u]
        distance[u] = distance[v] + edge_weight[v, u]
      lock_release(u)
```

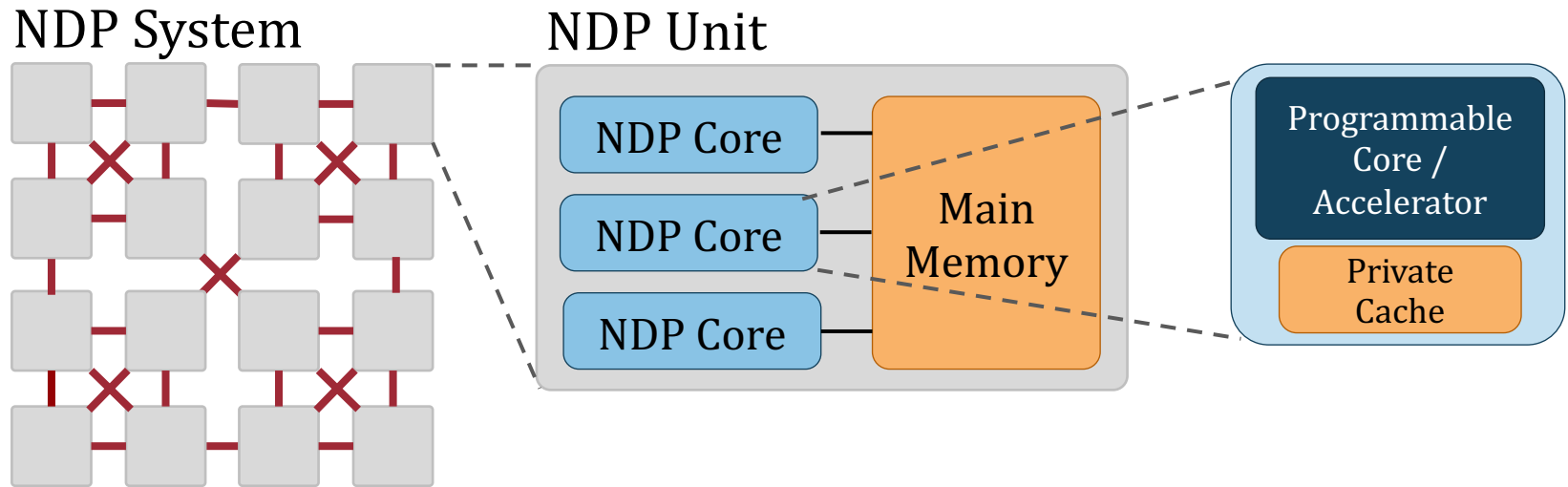
Locks



Barriers



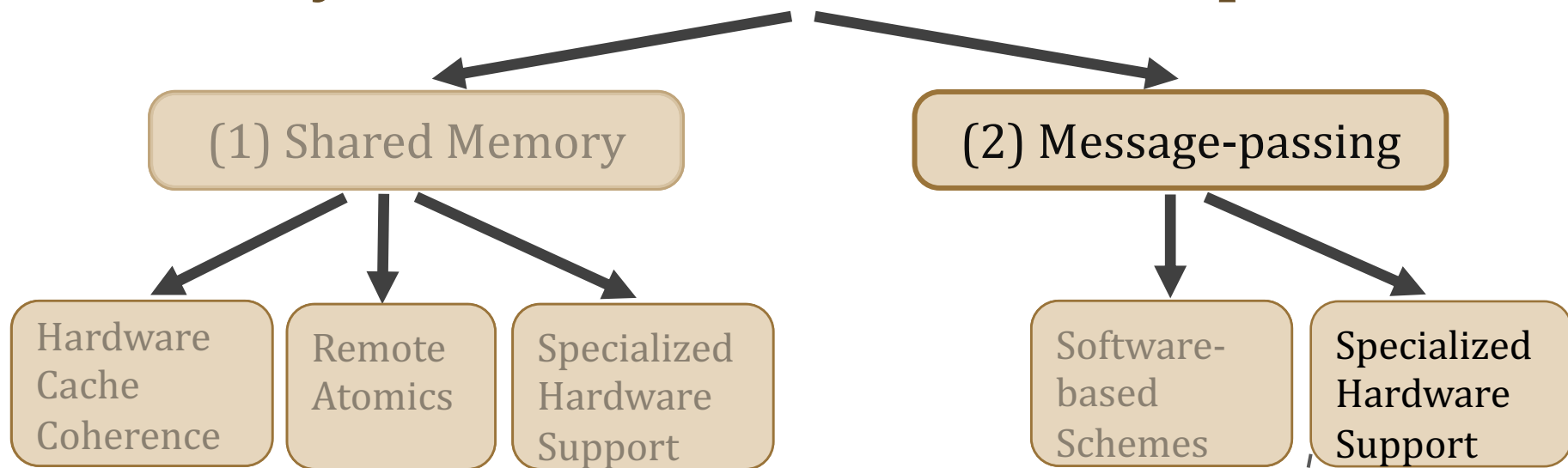
Baseline NDP Architecture



Synchronization **challenges** in NDP systems:

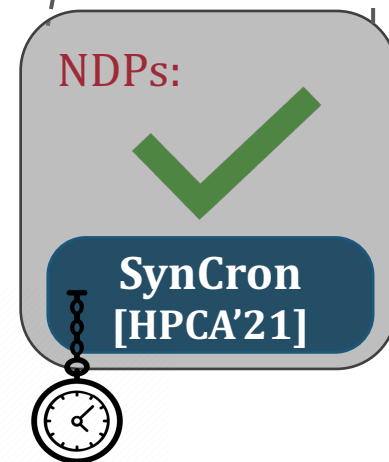
- (1) Lack of hardware cache coherence support
- (2) Expensive communication across NDP units
- (3) Lack of a shared level of cache memory

NDP Synchronization Solution Space

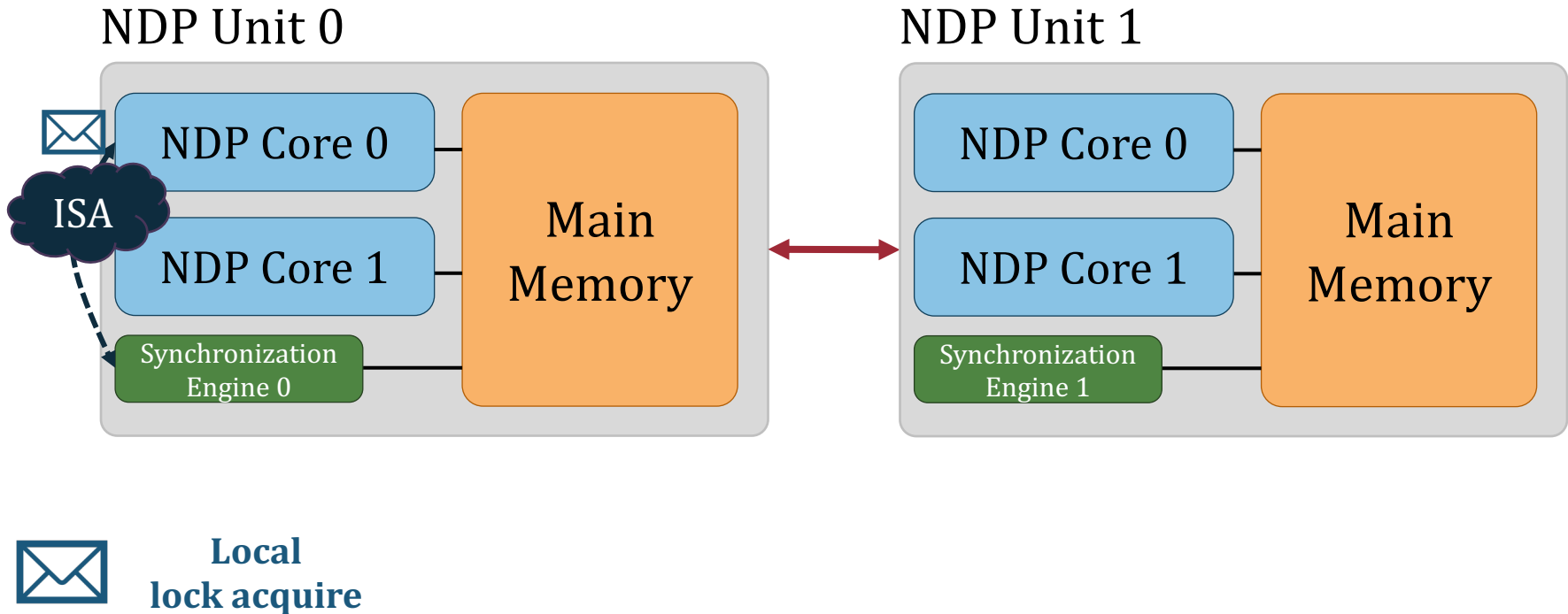


SynCron's Key Techniques:

1. **Hardware support** for synchronization acceleration
2. **Direct buffering** of synchronization variables
3. **Hierarchical** message-passing **communication**
4. Integrated hardware-only **overflow management**

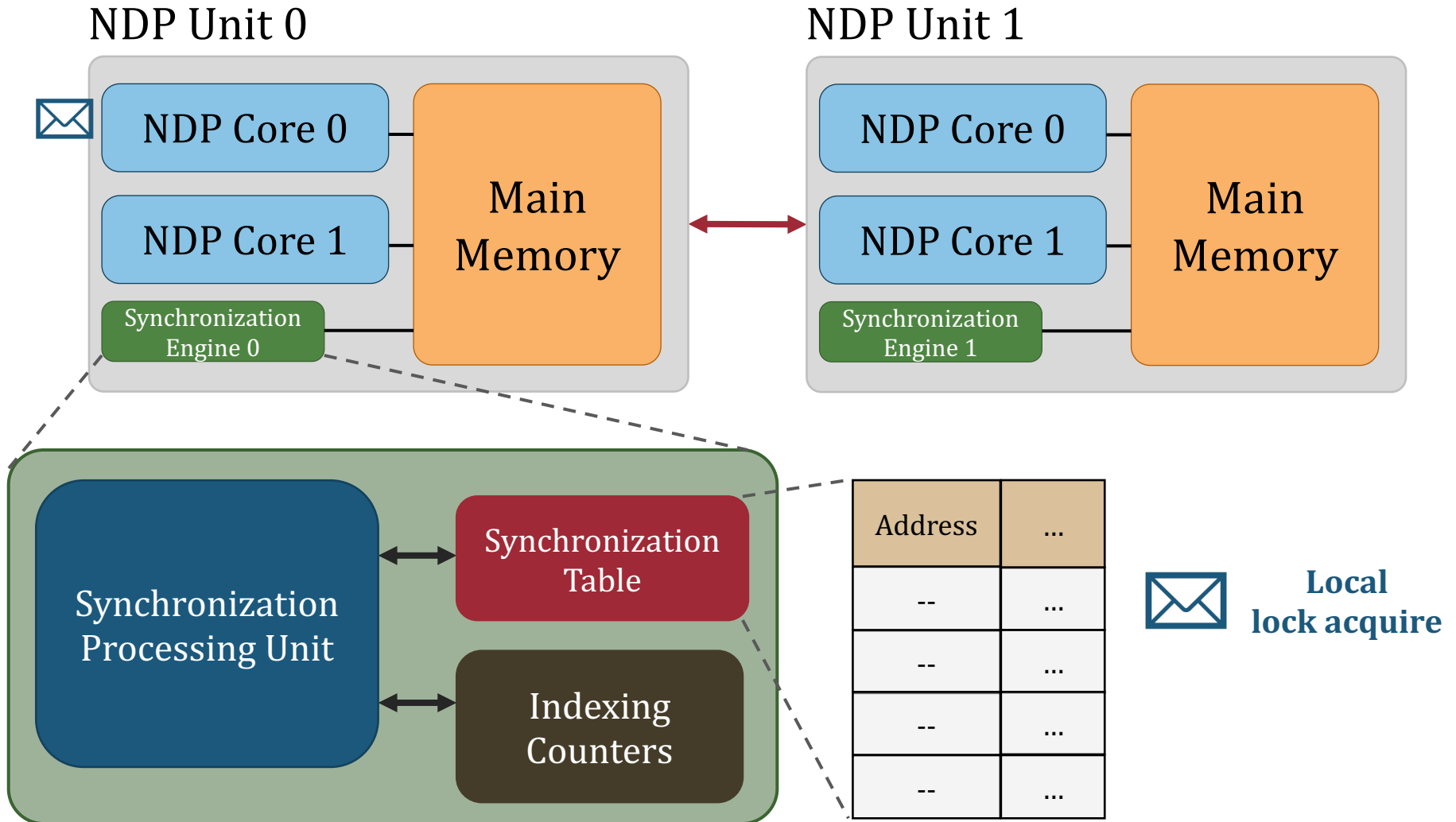


1. Hardware Synchronization Support



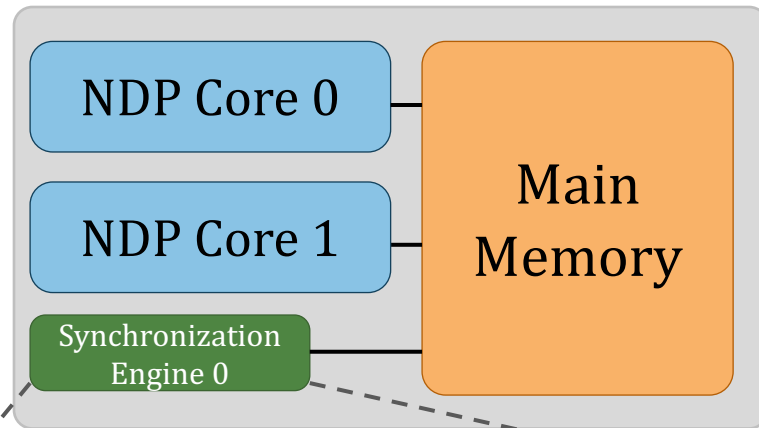
- ✓ No Complex Cache Coherence Protocols
- ✓ No Expensive Atomic Operations
- ✓ Low Hardware Cost

2. Direct Buffering of Variables

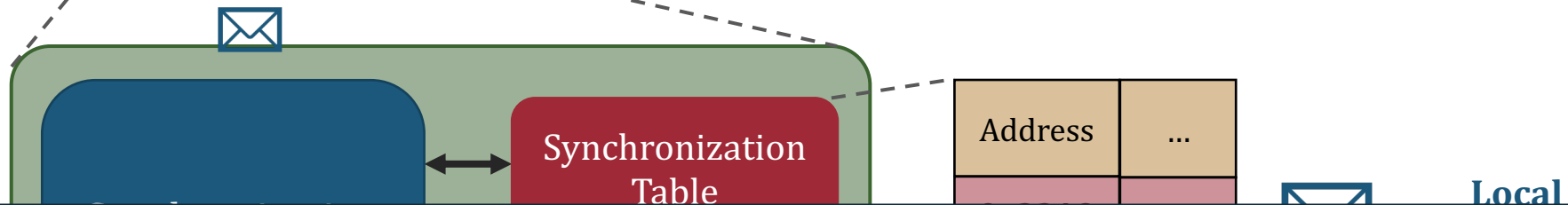
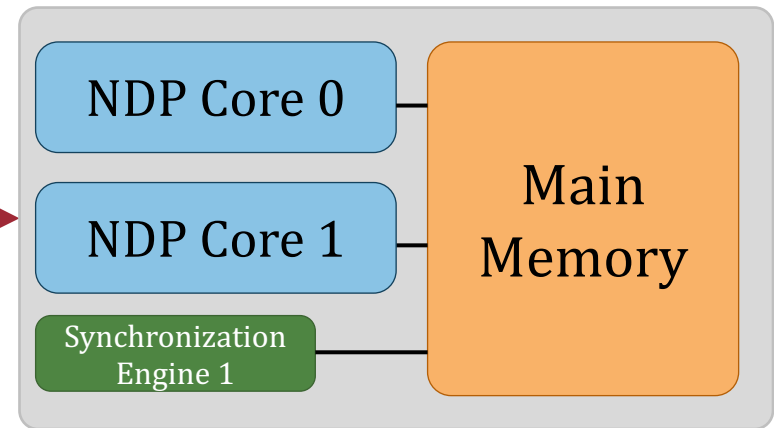


2. Direct Buffering of Variables

NDP Unit 0



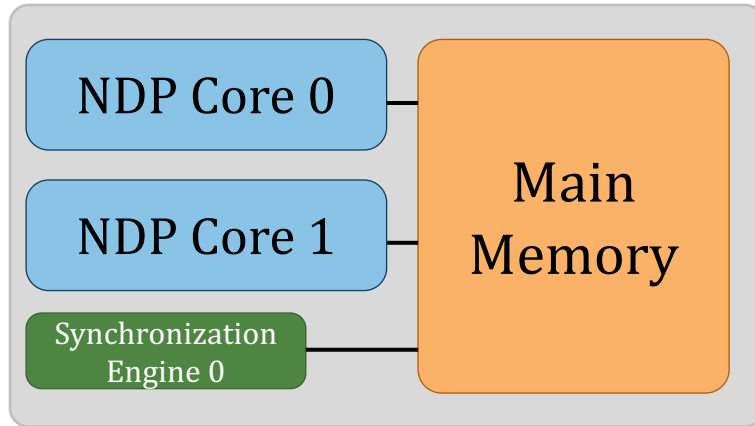
NDP Unit 1



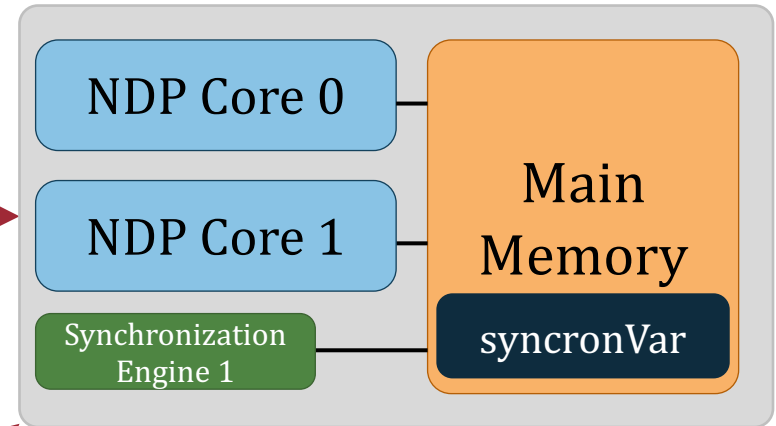
- ✓ No Costly Memory Accesses
- ✓ Low Latency

3. Hierarchical Communication

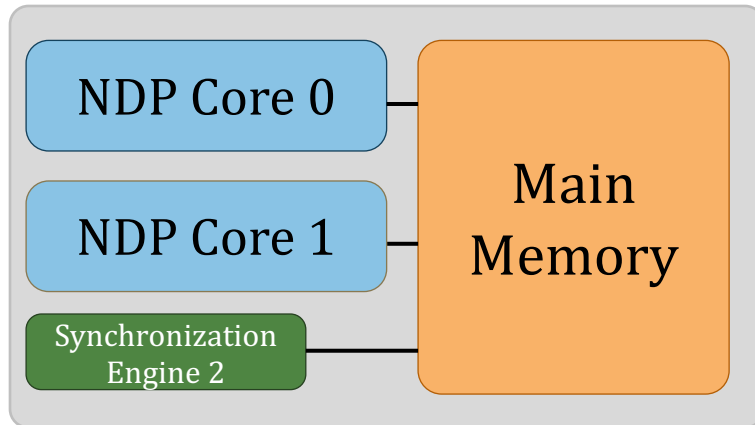
NDP Unit 0



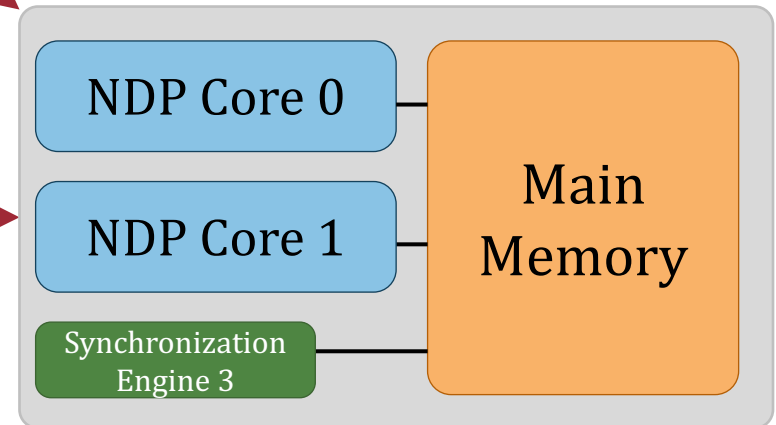
NDP Unit 1



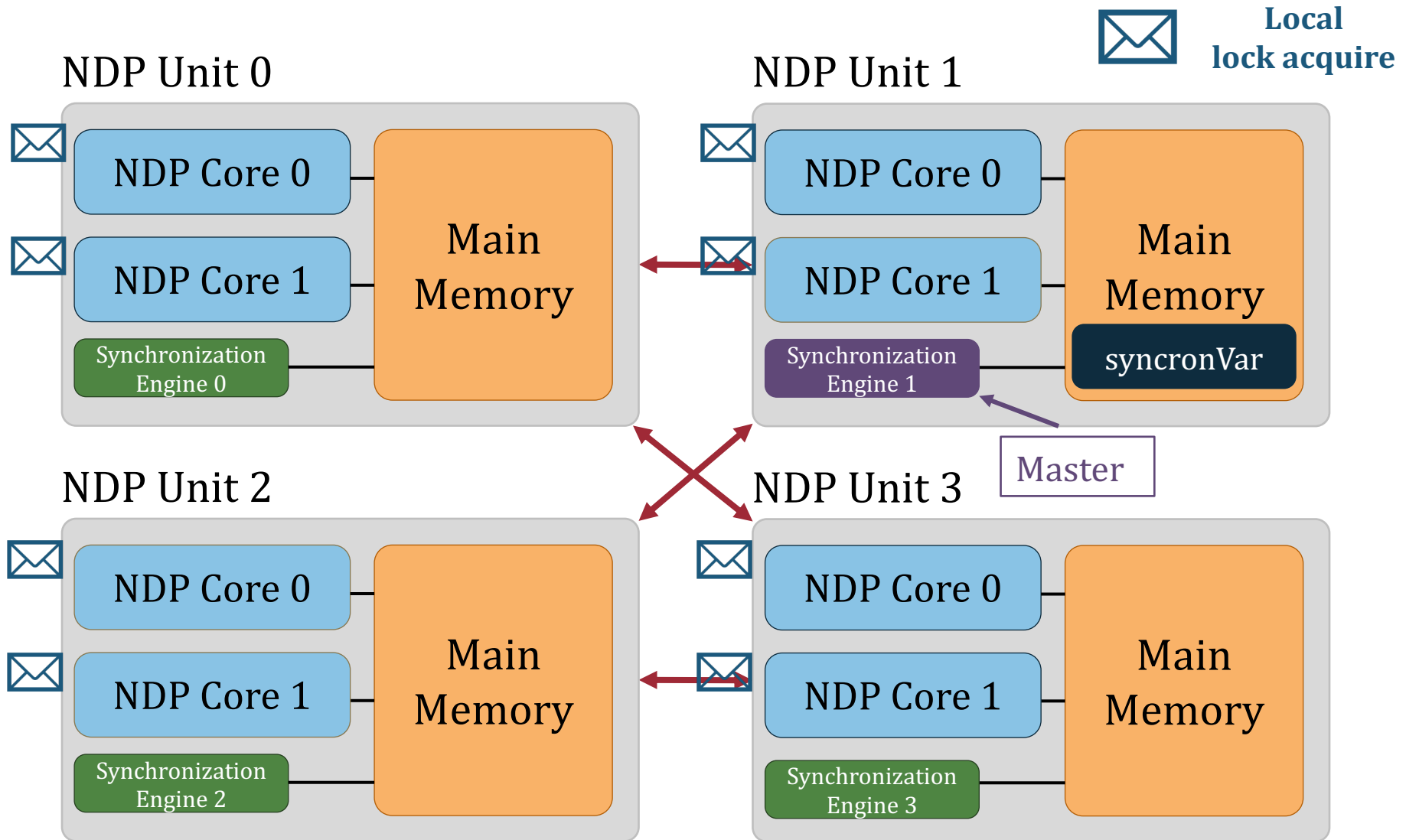
NDP Unit 2



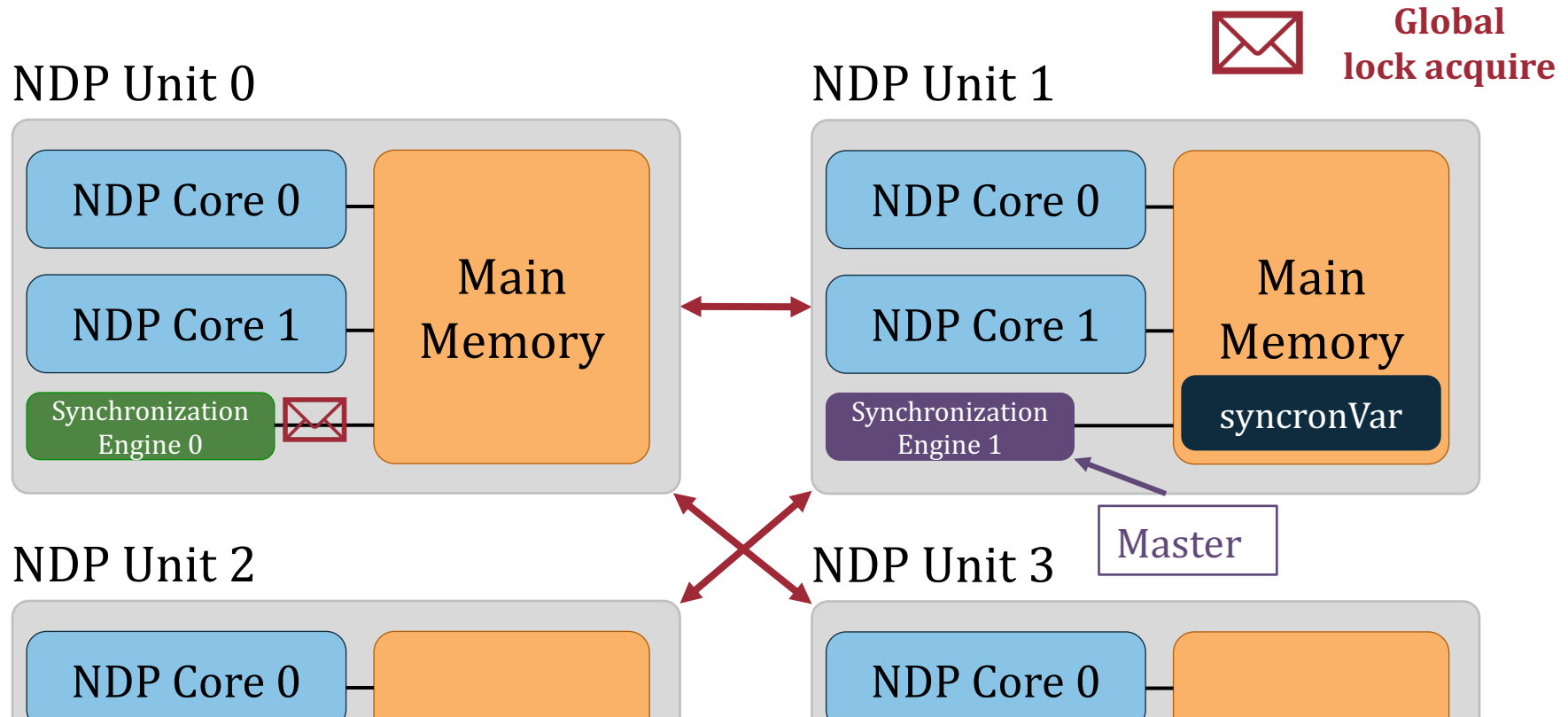
NDP Unit 3



3. Hierarchical Communication



3. Hierarchical Communication



✓ **Minimize Expensive Traffic**

SynCron

The first end-to-end synchronization solution for NDP architectures

SynCron's Benefits:

1. High System Performance
2. Low Hardware Cost

SynCron comes within 9.5% and 6.2% of performance and energy of Ideal zero-overhead synchronization

SynCron

Efficient Synchronization Support for Near-Data-Processing Architectures



Christina Giannoula

Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas
Ivan Fernandez, Juan Gómez Luna, Lois Orosa
Nectarios Koziris, Georgios Goumas, Onur Mutlu

SAFARI



ETH zürich



How to Support Synchronization?

- Christina Giannoula, Nandita Vijaykumar, Nikela Papadopoulou, Vasileios Karakostas, Ivan Fernandez, Juan Gómez-Luna, Lois Orosa, Nectarios Koziris, Georgios Goumas, Onur Mutlu, **"SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures"**
Proceedings of the 27th International Symposium on High-Performance Computer Architecture (HPCA), Virtual, February-March 2021.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Short Talk Slides \(pptx\)](#)] [[pdf](#)]
[[Talk Video](#) (21 minutes)]
[[Short Talk Video](#) (7 minutes)]

SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures

Christina Giannoula^{†‡} Nandita Vijaykumar^{*‡} Nikela Papadopoulou[†] Vasileios Karakostas[†] Ivan Fernandez^{§‡}
Juan Gómez-Luna[‡] Lois Orosa[‡] Nectarios Koziris[†] Georgios Goumas[†] Onur Mutlu[‡]
[†]*National Technical University of Athens* [‡]*ETH Zürich* ^{*}*University of Toronto* [§]*University of Malaga*

Lecture on Synchronization Support for PIM

1. Hardware Synchronization Support

NDP Unit 0

NDP Unit 1

ISA

NDP Core 0

NDP Core 1

Main Memory

Synchronization Engine 0

Synchronization Engine 1

Synchronization Table

- ✓ No Complex Cache Coherence Protocols
- ✓ No Expensive Atomic Operations
- ✓ Low Hardware Cost

Processing in Memory Course: Meeting 11: Synchronization Support for PIM Architectures - Fall'21

360 views • Streamed live on Dec 14, 2021

20 DISLIKE SHARE SAVE ...

Onur Mutlu Lectures
20.9K subscribers

SUBSCRIBED

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

Virtual Memory Support

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

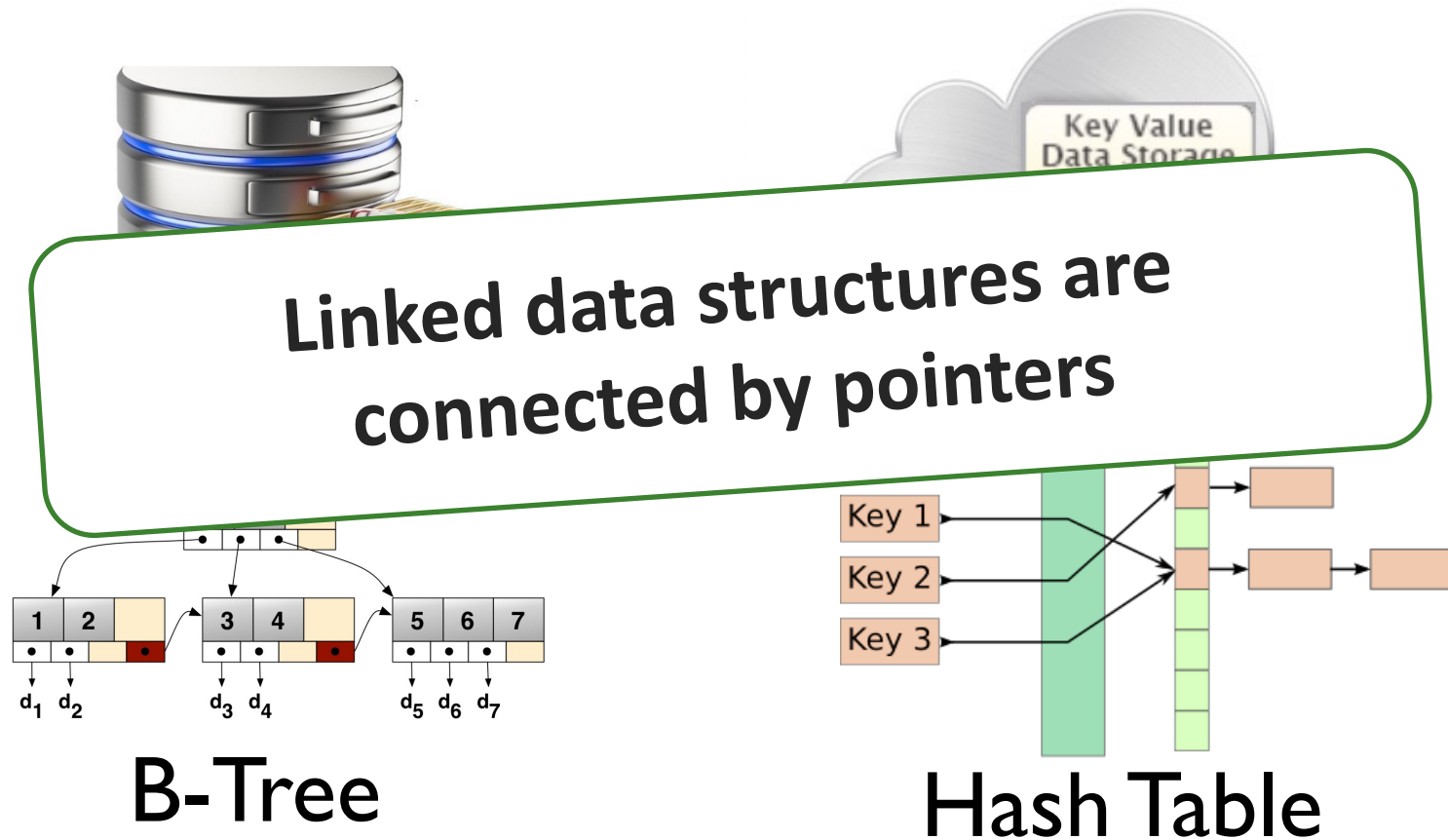
Kevin Hsieh[†] Samira Khan[‡] Nandita Vijaykumar[†]
Kevin K. Chang[†] Amirali Boroumand[†] Saugata Ghose[†] Onur Mutlu^{§†}
[†]*Carnegie Mellon University* [‡]*University of Virginia* [§]*ETH Zürich*

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

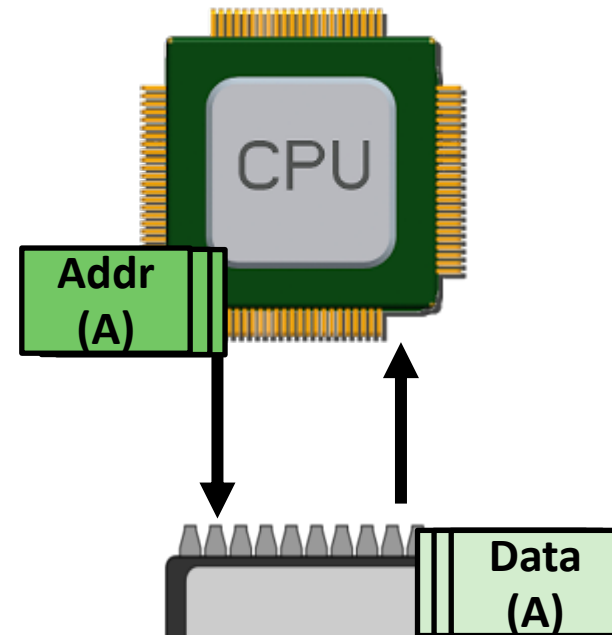
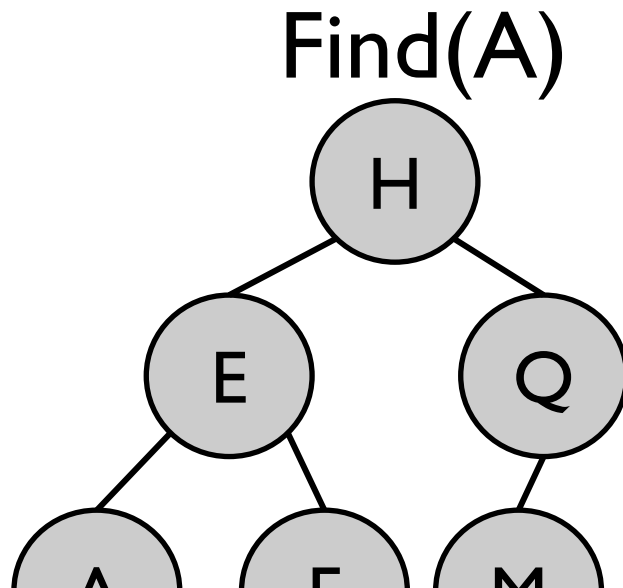
Linked Data Structures

- Linked data structures are widely used in many important applications



The Problem: Pointer Chasing

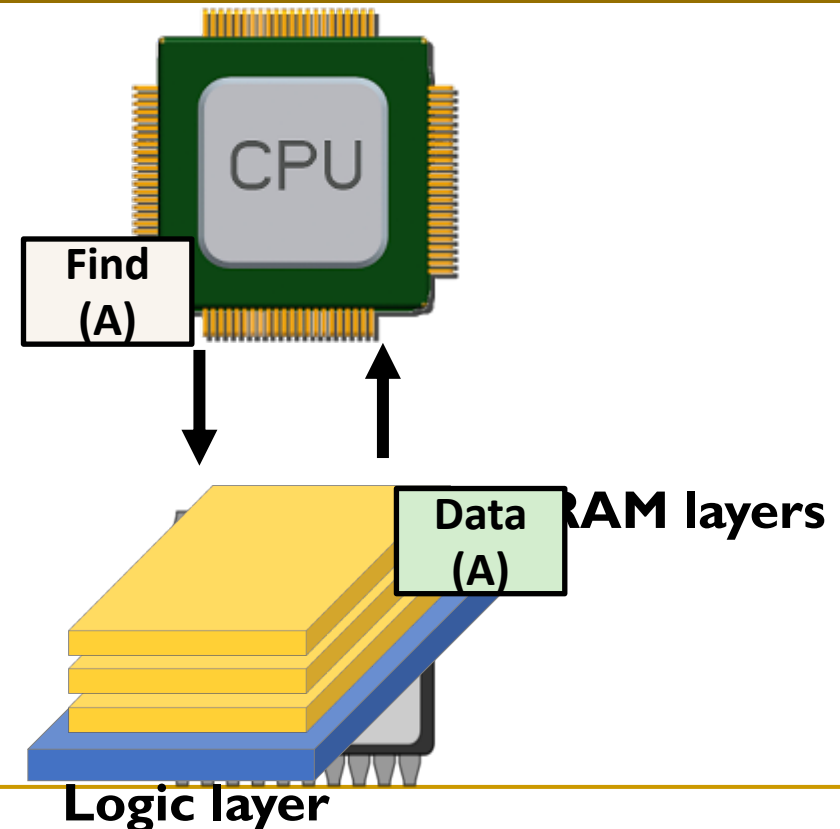
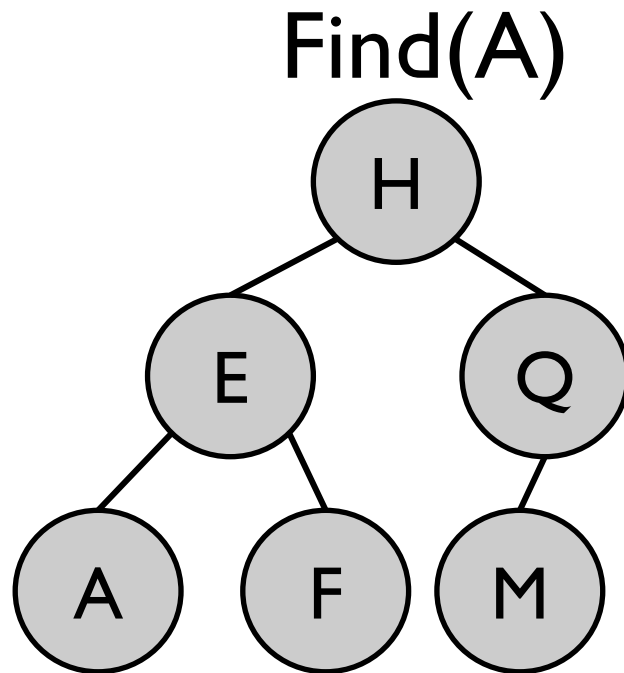
- Traversing linked data structures requires chasing pointers



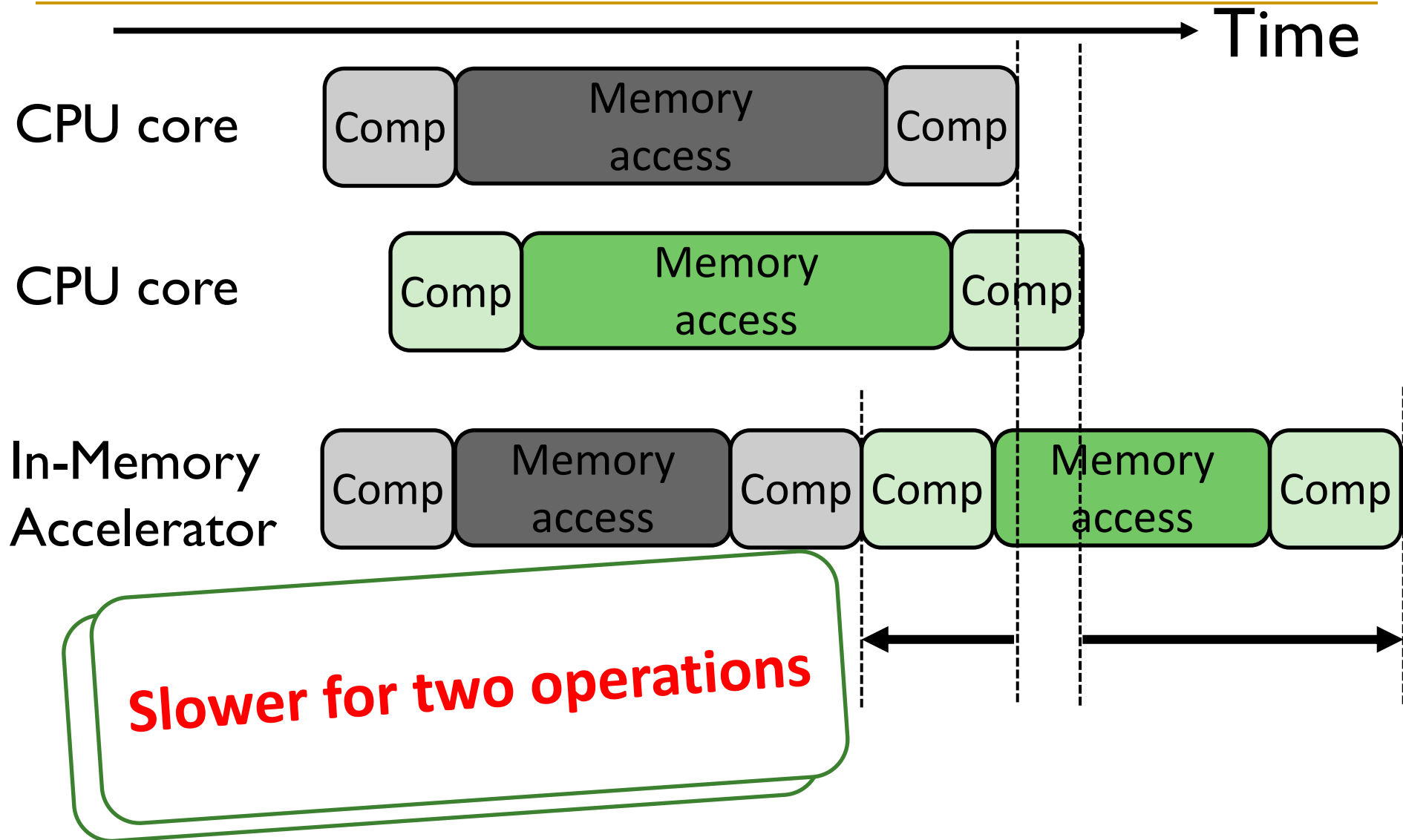
**Serialized and irregular access pattern
6X cycles per instruction in real workloads**

Our Goal

Accelerating pointer chasing inside main memory

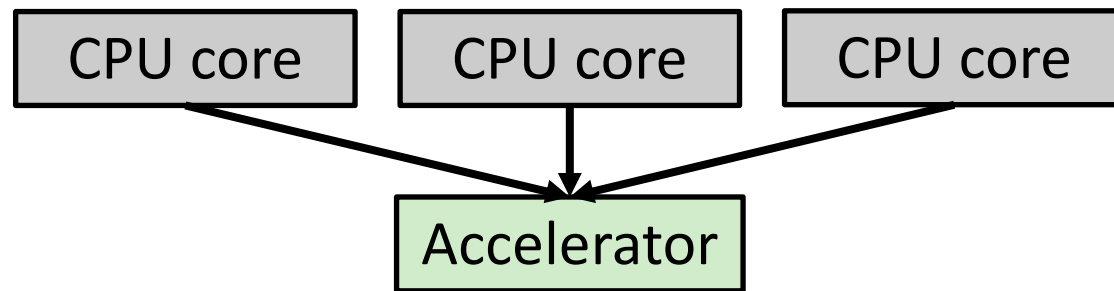


Parallelism Challenge

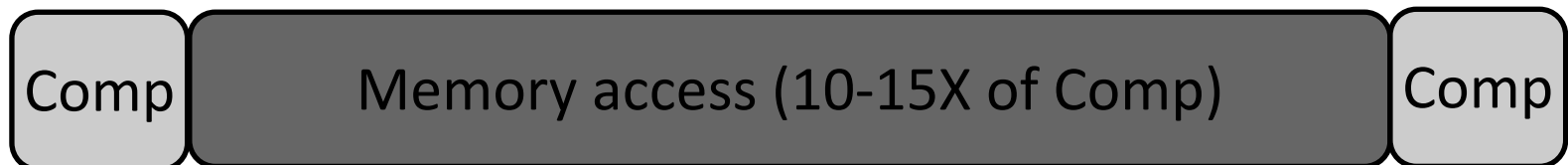


Parallelism Challenge and Opportunity

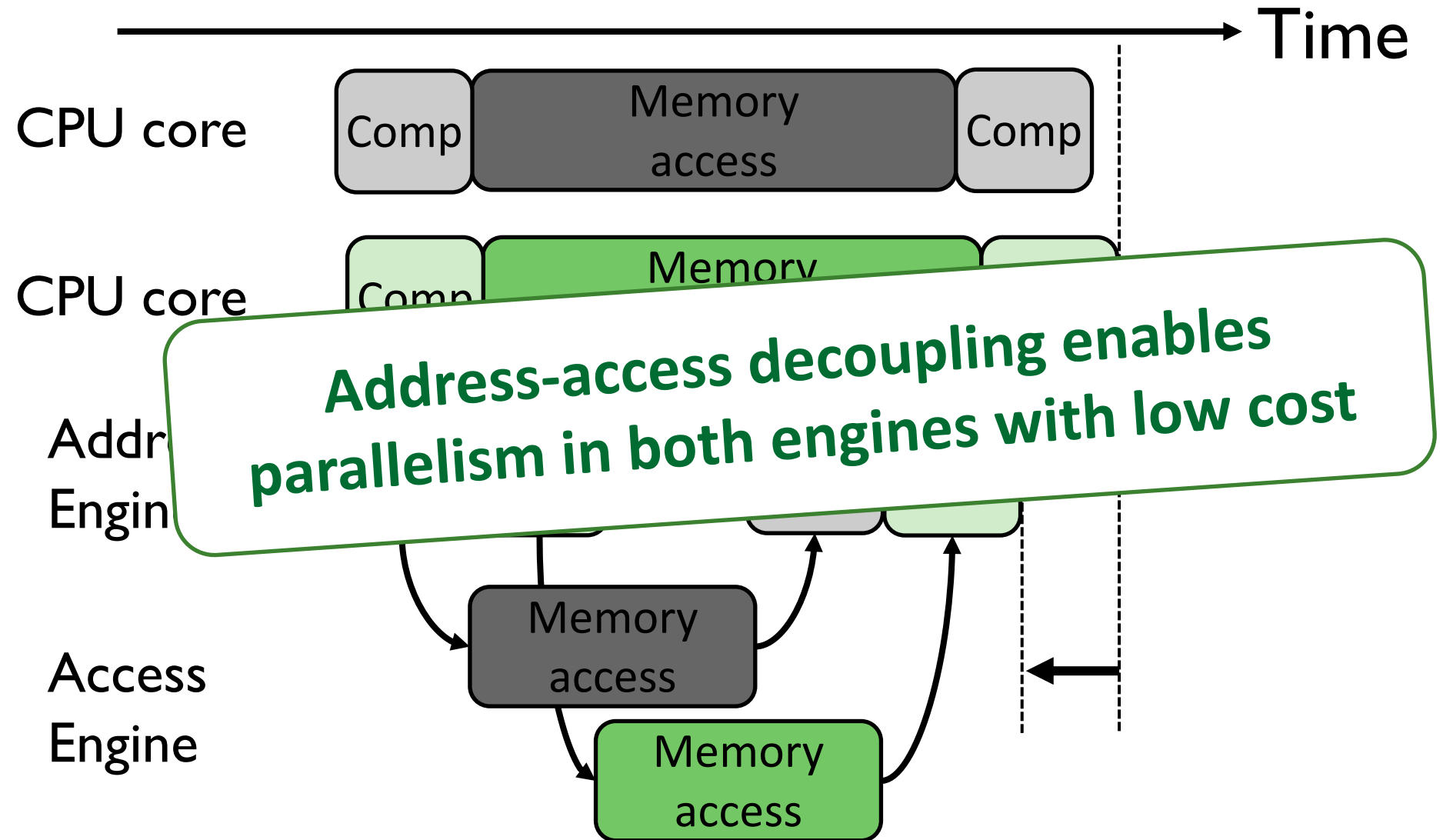
- A simple in-memory accelerator can still be **slower** than multiple CPU cores



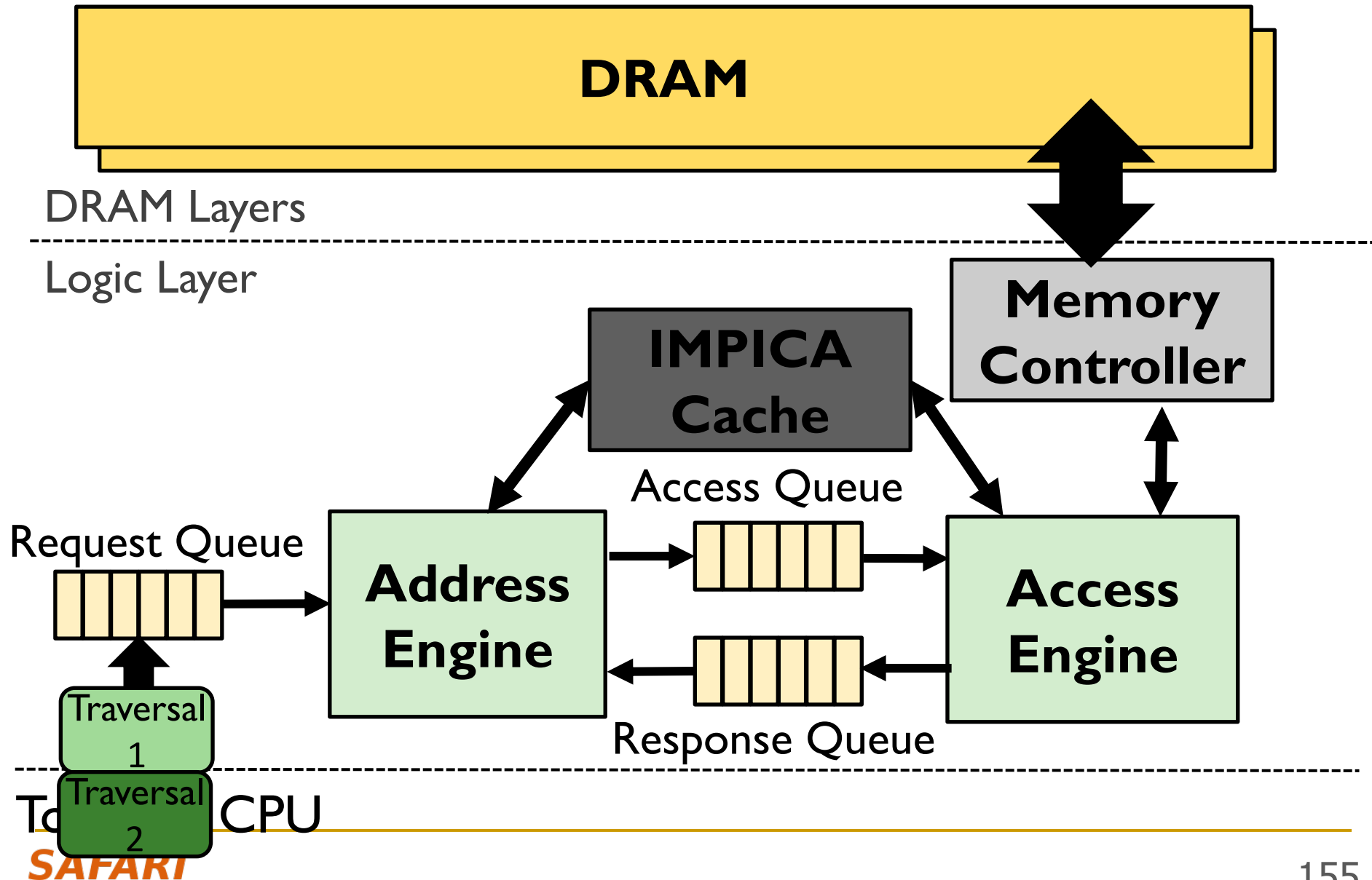
- **Opportunity:** a pointer-chasing accelerator spends a long time **waiting for memory**



Our Solution: Address-Access Decoupling



IMPICA Core Architecture

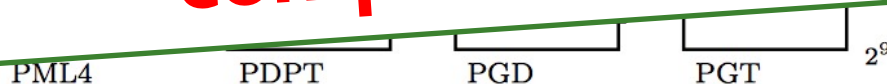


Address Translation Challenge

The page table walk requires multiple memory accesses



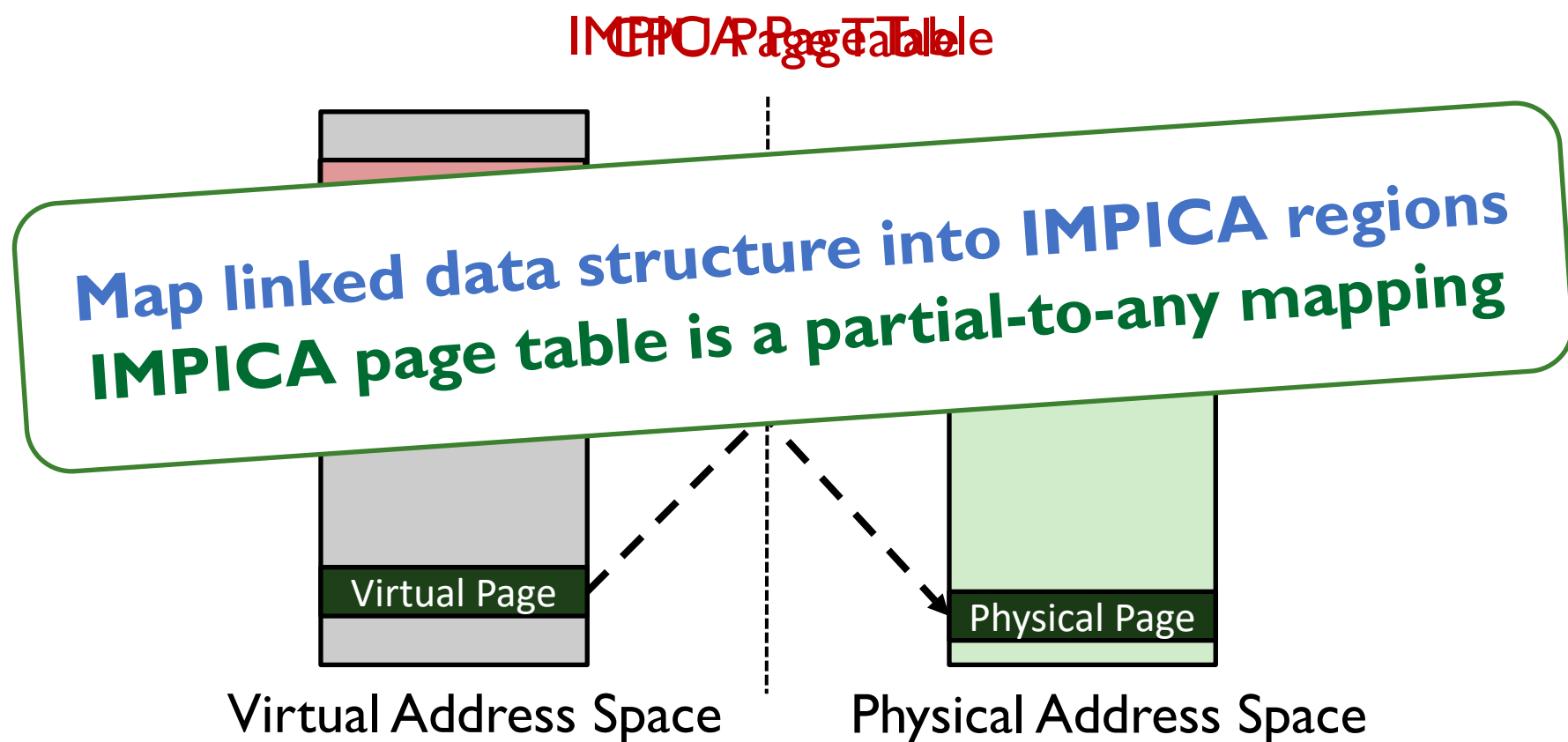
No TLB/MMU on the memory side
Duplicating it is costly and creates compatibility issue



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: Mechanism

Virtual Address

Bit [47:4]

Bit [11:0]

**Flat page table
saves one memory access**

Region Table

+

**Tiny region table is almost
always in the cache**

Flat Page Table
(2MB)

Small Page Table
(4KB)

+

Physical Address

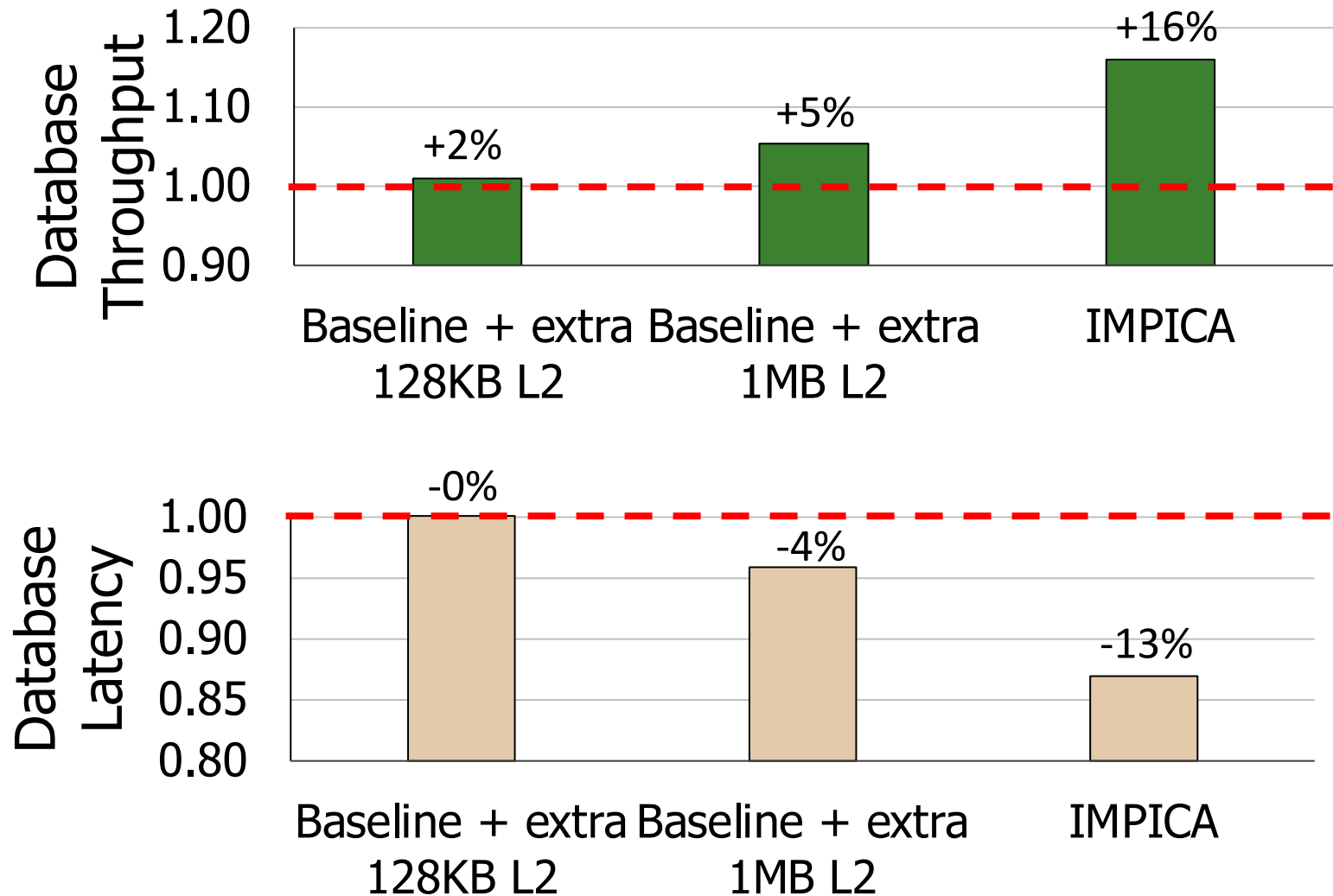
Evaluation Methodology

- 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>

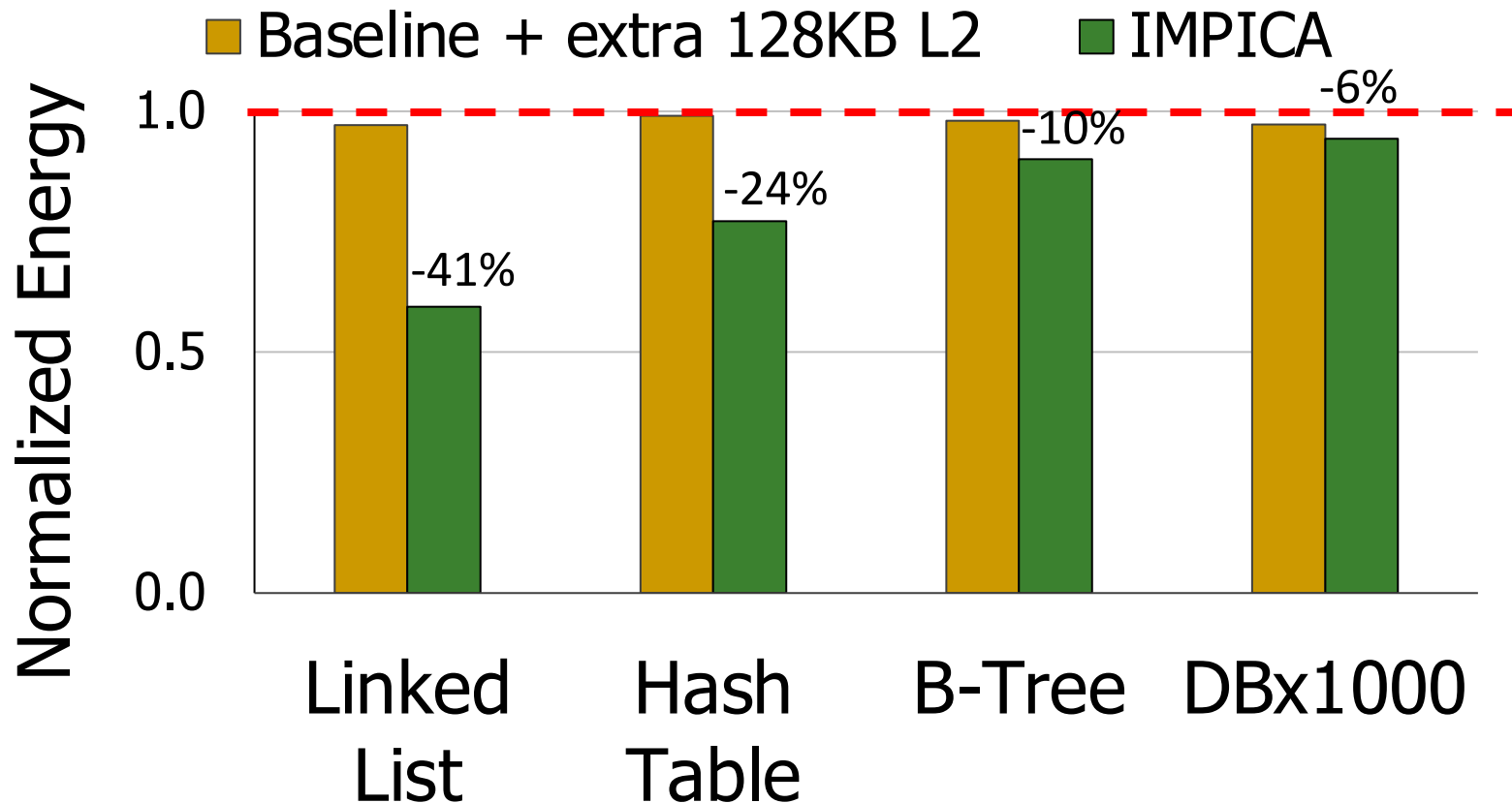
Result – Microbenchmark Performance



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%

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*

Rethinking Virtual Memory

Nastaran Hajinazar, Pratyush Patel, Minesh Patel, Konstantinos Kanellopoulos, Saugata Ghose, Rachata Ausavarungnirun, Geraldo Francisco de Oliveira Jr., Jonathan Appavoo, Vivek Seshadri, and Onur Mutlu, **"The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework"**

Proceedings of the 47th International Symposium on Computer Architecture (ISCA), Virtual, June 2020.

[[Slides \(pptx\)](#) ([pdf](#))]

[[Lightning Talk Slides \(pptx\)](#) ([pdf](#))]

[[ARM Research Summit Poster \(pptx\)](#) ([pdf](#))]

[[Talk Video](#) (26 minutes)]

[[Lightning Talk Video](#) (3 minutes)]

[[Lecture Video](#) (43 minutes)]

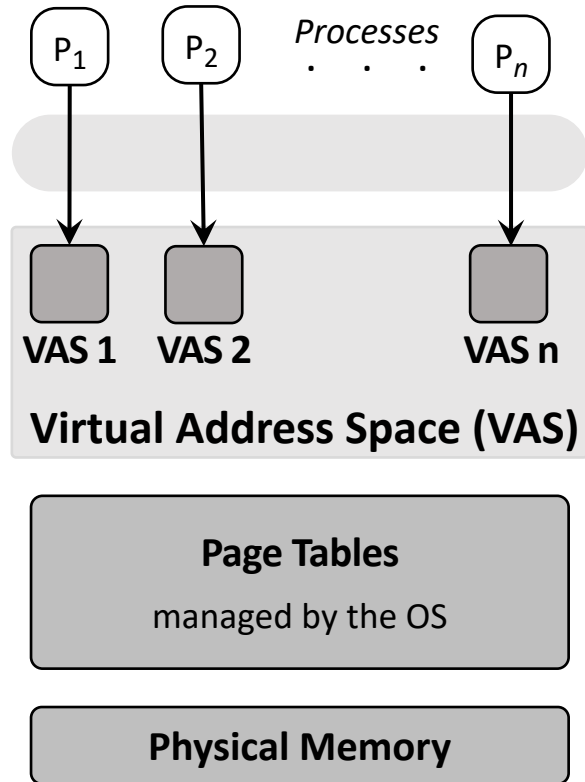
The Virtual Block Interface: A Flexible Alternative to the Conventional Virtual Memory Framework

Nastaran Hajinazar^{*†} Pratyush Patel[⌘] Minesh Patel^{*} Konstantinos Kanellopoulos^{*} Saugata Ghose[‡]
Rachata Ausavarungnirun[⊙] Geraldo F. Oliveira^{*} Jonathan Appavoo[◇] Vivek Seshadri[▽] Onur Mutlu^{*‡}

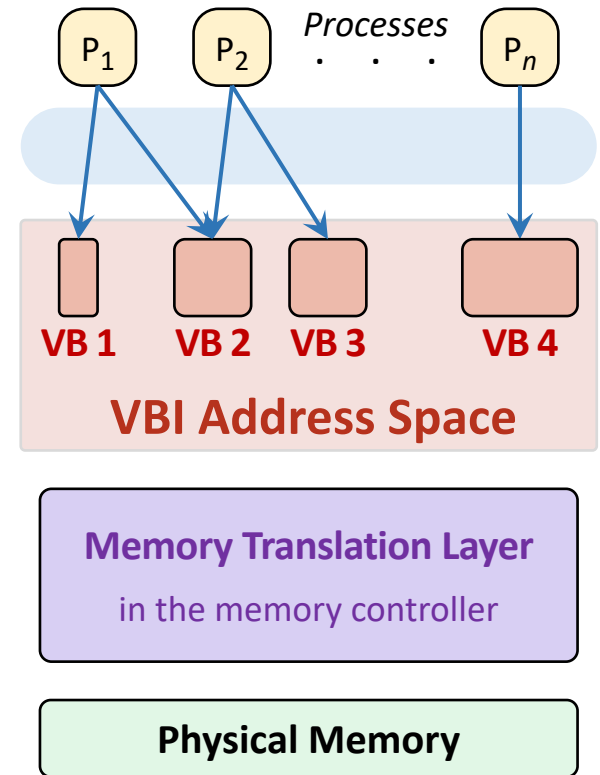
^{*}ETH Zürich [†]Simon Fraser University [⌘]University of Washington [‡]Carnegie Mellon University

[⊙]King Mongkut's University of Technology North Bangkok [◇]Boston University [▽]Microsoft Research India

VBI: Overview



Conventional Virtual Memory

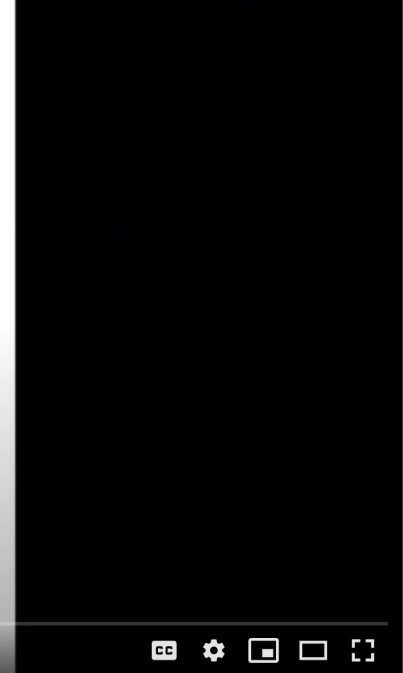


VBI

Lecture on Virtual Block Interface

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**



SAFARI 9:22 / 42:44

12



ETH ZÜRICH HAUPTGEBÄUDE

Computer Architecture - Lecture 12c: The Virtual Block Interface (ETH Zürich, Fall 2020)

726 views • Oct 31, 2020

16 0 SHARE SAVE ...



Onur Mutlu Lectures
16.5K subscribers

ANALYTICS

EDIT VIDEO

Benchmarks and Simulation Infrastructures

DAMOV Analysis Methodology & Workloads

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

GERALDO F. OLIVEIRA, ETH Zürich, Switzerland

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland

LOIS OROSA, ETH Zürich, Switzerland

SAUGATA GHOSE, University of Illinois at Urbana–Champaign, USA

NANDITA VIJAYKUMAR, University of Toronto, Canada

IVAN FERNANDEZ, University of Malaga, Spain & ETH Zürich, Switzerland

MOHAMMAD SADROSADATI, Institute for Research in Fundamental Sciences (IPM), Iran & ETH Zürich, Switzerland

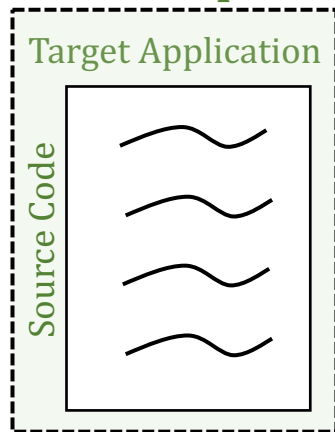
ONUR MUTLU, ETH Zürich, Switzerland

Data movement between the CPU and main memory is a first-order obstacle against improving performance, scalability, and energy efficiency in modern systems. Computer systems employ a range of techniques to reduce overheads tied to data movement, spanning from traditional mechanisms (e.g., deep multi-level cache hierarchies, aggressive hardware prefetchers) to emerging techniques such as Near-Data Processing (NDP), where some computation is moved close to memory. Prior NDP works investigate the root causes of data movement bottlenecks using different profiling methodologies and tools. However, there is still a lack of understanding about the key metrics that can identify different data movement bottlenecks and their relation to traditional and emerging data movement mitigation mechanisms. Our goal is to methodically identify potential sources of data movement over a broad set of applications and to comprehensively compare traditional compute-centric data movement mitigation techniques (e.g., caching and prefetching) to more memory-centric techniques (e.g., NDP), thereby developing a rigorous understanding of the best techniques to mitigate each source of data movement.

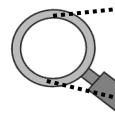
With this goal in mind, we perform the first large-scale characterization of a wide variety of applications, across a wide range of application domains, to identify fundamental program properties that lead to data movement to/from main memory. We develop the first systematic methodology to classify applications based on the sources contributing to data movement bottlenecks. From our large-scale characterization of 77K functions across 345 applications, we select 144 functions to form the first open-source benchmark suite (DAMOV) for main memory data movement studies. We select a diverse range of functions that (1) represent different types of data movement bottlenecks, and (2) come from a wide range of application domains. Using NDP as a case study, we identify new insights about the different data movement bottlenecks and use these insights to determine the most suitable data movement mitigation mechanism for a particular application. We open-source DAMOV and the complete source code for our new characterization methodology at <https://github.com/CMU-SAFARI/DAMOV>.

Methodology Overview

User Input



Step 1 Application Profiling



Profiler

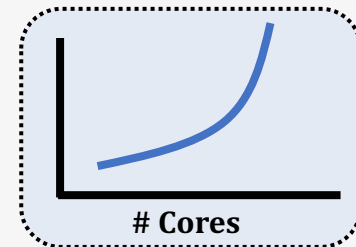
roi_begin

roi_end

DAMOV-SIM Simulator

```
ld 0xFF
st 0xAF
ld 0xFF
st 0xAF
ld 0xFF
```

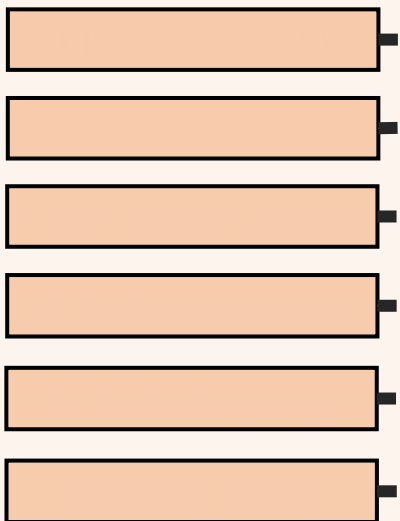
Memory Traces



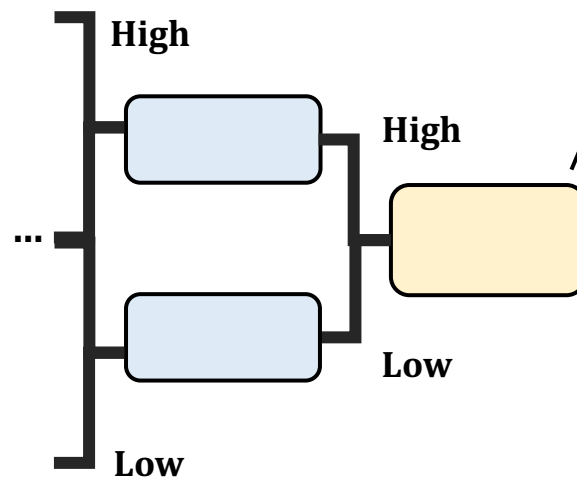
Scalability Analysis

Methodology Output

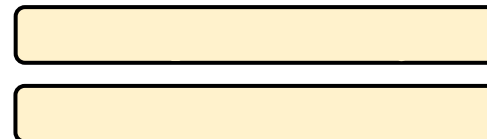
Memory Bottleneck Classes



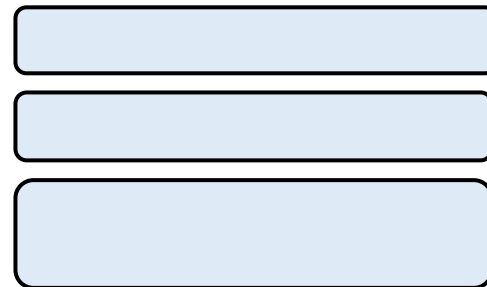
SAFARI



Step 2 Locality-based Clustering



Step 3 Memory Bottleneck Class.



16

More on DAMOV

- Geraldo F. Oliveira, Juan Gomez-Luna, Lois Orosa, Saugata Ghose, Nandita Vijaykumar, Ivan fernandez, Mohammad Sadrosadati, and Onur Mutlu,
"DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks"
Preprint in [arXiv](#), 8 May 2021.
[[arXiv preprint](#)]
[[DAMOV Suite and Simulator Source Code](#)]
[[SAFARI Live Seminar Video](#) (2 hrs 40 mins)]
[[Short Talk Video](#) (21 minutes)]

DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks

GERALDO F. OLIVEIRA, ETH Zürich, Switzerland

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland

LOIS OROSA, ETH Zürich, Switzerland

SAUGATA GHOSE, University of Illinois at Urbana–Champaign, USA

NANDITA VIJAYKUMAR, University of Toronto, Canada

IVAN FERNANDEZ, University of Malaga, Spain & ETH Zürich, Switzerland

MOHAMMAD SADROSADATI, ETH Zürich, Switzerland

ONUR MUTLU, ETH Zürich, Switzerland

Lecture on DAMOV

Step 3: Memory Bottleneck Classification (2/2)

- **Goal:** identify the specific sources of data movement bottlenecks

DAMOV-SIM Simulator

Scalability Analysis

Integrated ZSim and Ramulator

Configuration 1: Host CPU System

CPU → L1 → L2 → L3 → Off-chip link → DRAM

Configuration 2: NDP System

Off-chip link → ... → CPU → L1 → Logic Layer → DRAM

- **Scalability Analysis:**
 - 1, 4, 16, 64, and 256 out-of-order/in-order host and NDP CPU cores
 - 3D-stacked memory as main memory

24:08 / 1:00:03

DAMOV-SIM: <https://github.com/CMU-SAFARI/DAMOV-SIM>

Processing-in-Memory Course: Lecture 5: How to Evaluate Data Movement Bottlenecks - Spring 2022

346 views • Streamed live on Apr 7, 2022

12 DISLIKE SHARE CLIP SAVE ...



Onur Mutlu Lectures

25.9K subscribers

SUBSCRIBED



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

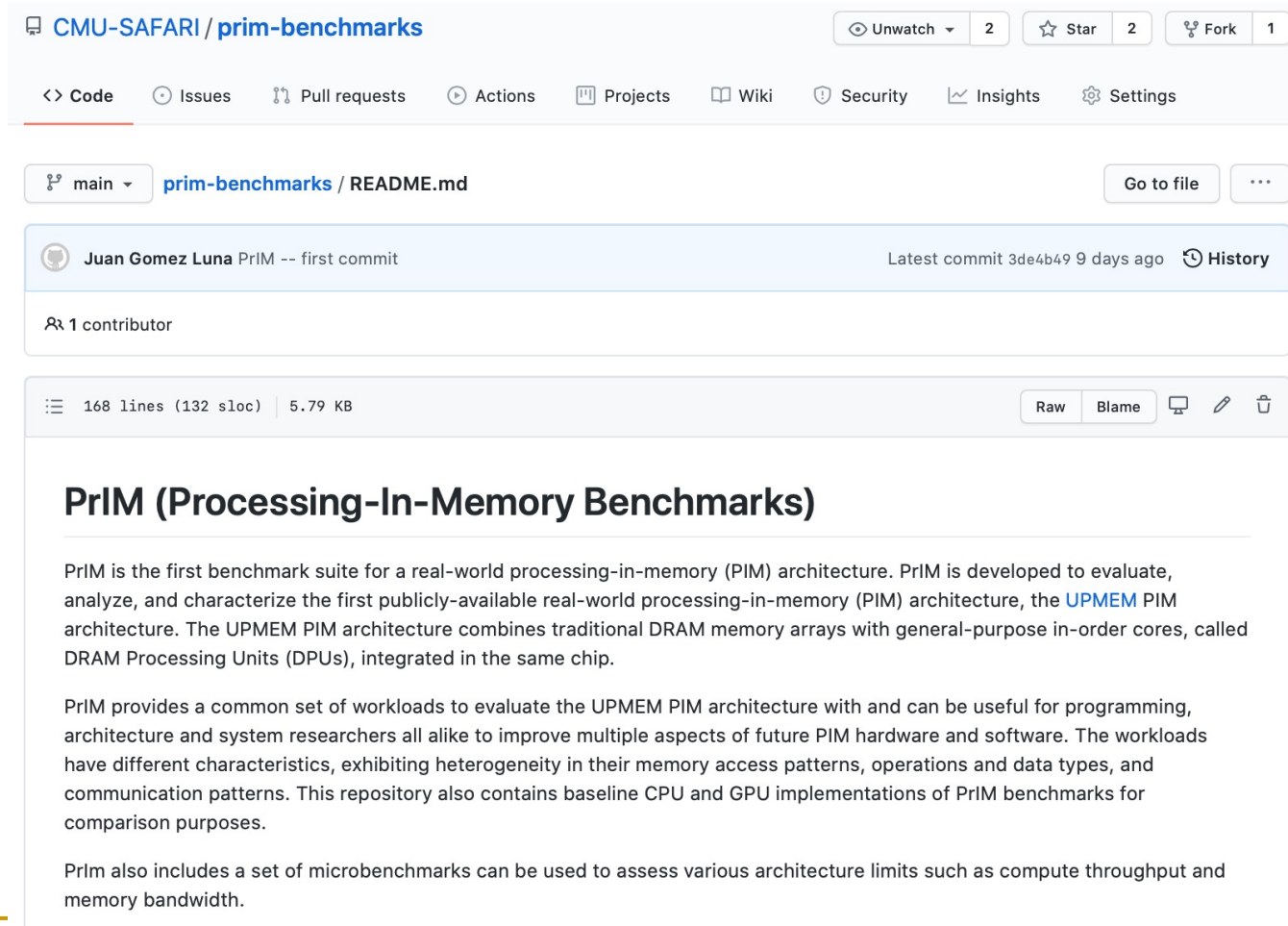
Yoongu Kim¹ Weikun Yang^{1,2} Onur Mutlu¹
¹Carnegie Mellon University ²Peking University

PrIM Benchmarks: Application Domains

| Domain | Benchmark | Short name |
|-----------------------|-------------------------------|------------|
| Dense linear algebra | Vector Addition | VA |
| | Matrix-Vector Multiply | GEMV |
| Sparse linear algebra | Sparse Matrix-Vector Multiply | SpMV |
| Databases | Select | SEL |
| | Unique | UNI |
| Data analytics | Binary Search | BS |
| | Time Series Analysis | TS |
| Graph processing | Breadth-First Search | BFS |
| Neural networks | Multilayer Perceptron | MLP |
| Bioinformatics | Needleman-Wunsch | NW |
| Image processing | Image histogram (short) | HST-S |
| | Image histogram (large) | HST-L |
| Parallel primitives | Reduction | RED |
| | Prefix sum (scan-scan-add) | SCAN-SSA |
| | Prefix sum (reduce-scan-scan) | SCAN-RSS |
| | Matrix transposition | TRNS |

PrIM Benchmarks are Open Source

- All microbenchmarks, benchmarks, and scripts
- <https://github.com/CMU-SAFARI/prim-benchmarks>



The screenshot shows the GitHub repository page for `CMU-SAFARI/prim-benchmarks`. At the top, there are navigation tabs for `<> Code`, `Issues`, `Pull requests`, `Actions`, `Projects`, `Wiki`, `Security`, `Insights`, and `Settings`. Below these, the repository name `prim-benchmarks` is shown with a dropdown menu set to `main` and a link to `README.md`. The commit history section shows a single commit by `Juan Gomez Luna` titled `PrIM -- first commit`, with the latest commit hash `3de4b49` and a timestamp of `9 days ago`. The file details section indicates the file is `168 lines (132 sloc)` and `5.79 KB` in size. The main content area displays the title `PrIM (Processing-In-Memory Benchmarks)` followed by a detailed description of the benchmark suite and its purpose.

CMU-SAFARI / `prim-benchmarks` Unwatch 2 Star 2 Fork 1

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

main `prim-benchmarks` / `README.md` Go to file ...

Juan Gomez Luna `PrIM -- first commit` Latest commit `3de4b49` 9 days ago History

1 contributor

168 lines (132 sloc) 5.79 KB Raw Blame

PrIM (Processing-In-Memory Benchmarks)

PrIM is the first benchmark suite for a real-world processing-in-memory (PIM) architecture. PrIM is developed to evaluate, analyze, and characterize the first publicly-available real-world processing-in-memory (PIM) architecture, the [UPMEM](#) PIM architecture. The UPMEM PIM architecture combines traditional DRAM memory arrays with general-purpose in-order cores, called DRAM Processing Units (DPUs), integrated in the same chip.

PrIM provides a common set of workloads to evaluate the UPMEM PIM architecture with and can be useful for programming, architecture and system researchers all alike to improve multiple aspects of future PIM hardware and software. The workloads have different characteristics, exhibiting heterogeneity in their memory access patterns, operations and data types, and communication patterns. This repository also contains baseline CPU and GPU implementations of PrIM benchmarks for comparison purposes.

PrIM also includes a set of microbenchmarks can be used to assess various architecture limits such as compute throughput and memory bandwidth.

Lecture on PrIM Benchmarks

Strong Scaling: 1 DPU (VI)

The amount of time spent on CPU-DPU and DPU-CPU transfers is low compared to the time spent on DPU execution

TRNS performs step 1 of the matrix transposition via the CPU-DPU transfer.
Using small transfers (8 elements) does not exploit full CPU-DPU bandwidth

KEY OBSERVATION 13
Transferring large data chunks from/to the host CPU is preferred for input data and output results due to higher sustained CPU-DPU/DPU-CPU bandwidths.

17:57 / 34:35 • Strong Scaling: 1 DPU (V) >

Processing-in-Memory Course: Lecture 8: Benchmarking and Workload Suitability on PIM - Spring 2022

213 views • Premiered Apr 28, 2022

5 DISLIKE SHARE CLIP SAVE ...



Onur Mutlu Lectures

25.9K subscribers

SUBSCRIBED



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

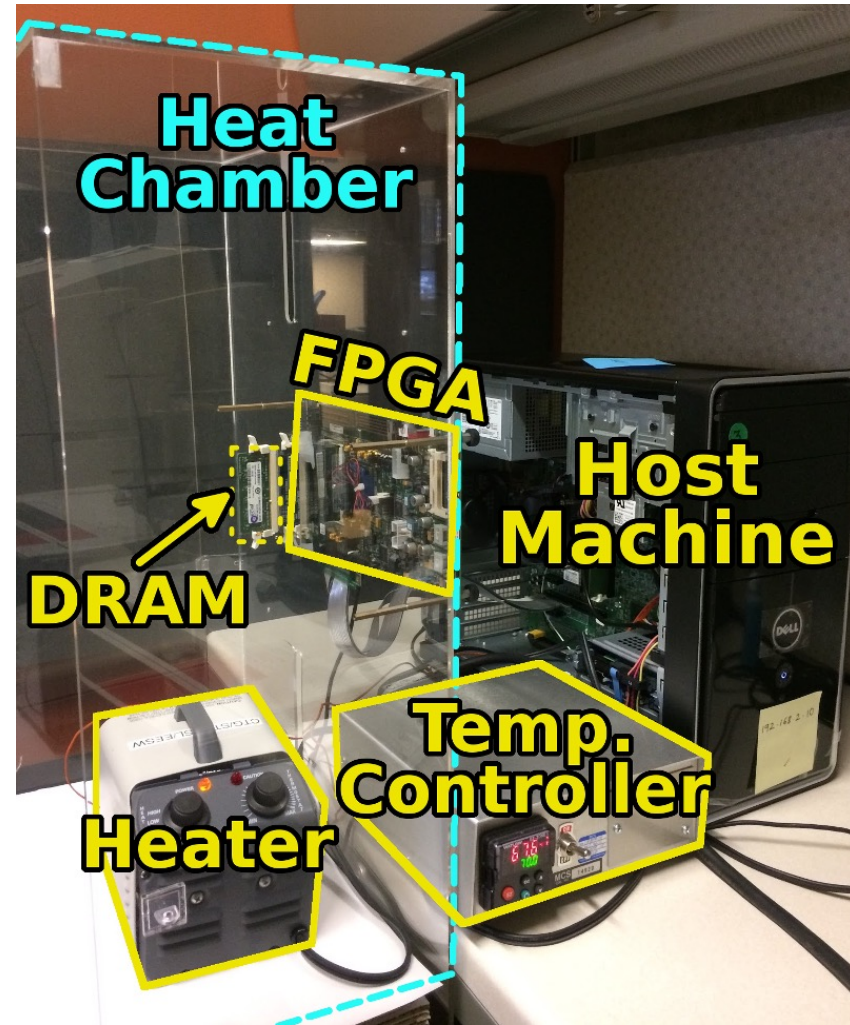
^aEindhoven University of Technology

^bETH Zürich

^cIBM 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



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*

Applications that Benefit from PIM

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 Cali¹, Hongyi Xin², Donghyuk Lee³, Saugata Ghose¹, Mohammed Alser⁴, Hasan Hassan⁶, Oguz Ergin⁵, Can Alkan^{4*} and Onur Mutlu^{6,1*}

From The Sixteenth Asia Pacific Bioinformatics Conference 2018
Yokohama, Japan. 15-17 January 2018

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

Carnegie Mellon



ETH zürich

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**.

Accelerating Approximate String Matching

- Damla Senol Cali, Gurpreet S. Kalsi, Zülal Bingöl, Can Firtina, Lavanya Subramanian, Jeremie S. Kim, Rachata Ausavarungnirun, Mohammed Alser, Juan Gomez-Luna, Amirali Boroumand, Anant Nori, Allison Scibisz, Sreenivas Subramoney, Can Alkan, Saugata Ghose, and Onur Mutlu, **"GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis"**
Proceedings of the 53rd International Symposium on Microarchitecture (MICRO), Virtual, October 2020.
[[Lightning Talk Video](#) (1.5 minutes)]
[[Lightning Talk Slides \(pptx\)](#) ([pdf](#))]
[[Talk Video](#) (18 minutes)]
[[Slides \(pptx\)](#) ([pdf](#))]

GenASM: A High-Performance, Low-Power Approximate String Matching Acceleration Framework for Genome Sequence Analysis

Damla Senol Cali^{†⌘} Gurpreet S. Kalsi[⌘] Zülal Bingöl[▽] Can Firtina[◇] Lavanya Subramanian[‡] Jeremie S. Kim^{◇†}
Rachata Ausavarungnirun[○] Mohammed Alser[◇] Juan Gomez-Luna[◇] Amirali Boroumand[†] Anant Nori[⌘]
Allison Scibisz[†] Sreenivas Subramoney[⌘] Can Alkan[▽] Saugata Ghose^{*†} Onur Mutlu^{◇†▽}
[†]Carnegie Mellon University [⌘]Processor Architecture Research Lab, Intel Labs [▽]Bilkent University [◇]ETH Zürich
[‡]Facebook [○]King Mongkut's University of Technology North Bangkok ^{*}University of Illinois at Urbana-Champaign

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



SEOUL
NATIONAL
UNIVERSITY

ETH zürich

Accelerating Climate Modeling

- Gagandeep Singh, Dionysios Diamantopoulos, Christoph Hagleitner, Juan Gómez-Luna, Sander Stuijk, Onur Mutlu, and Henk Corporaal,
"NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling"
Proceedings of the 30th International Conference on Field-Programmable Logic and Applications (FPL), Gothenburg, Sweden, September 2020.
[[Slides \(pptx\)](#)] [[pdf](#)]
[[Lightning Talk Slides \(pptx\)](#)] [[pdf](#)]
[[Talk Video](#) (23 minutes)]
Nominated for the Stamatis Vassiliadis Memorial Award.

NERO: A Near High-Bandwidth Memory Stencil Accelerator for Weather Prediction Modeling

Gagandeep Singh^{a,b,c} Dionysios Diamantopoulos^c Christoph Hagleitner^c Juan Gómez-Luna^b
Sander Stuijk^a Onur Mutlu^b Henk Corporaal^a
^aEindhoven University of Technology ^bETH Zürich ^cIBM Research Europe, Zurich

Accelerating Time Series Analysis

- Ivan Fernandez, Ricardo Quisiant, Christina Giannoula, Mohammed Alser, Juan Gómez-Luna, Eladio Gutiérrez, Oscar Plata, and Onur Mutlu,
"NATSA: A Near-Data Processing Accelerator for Time Series Analysis"
Proceedings of the 38th IEEE International Conference on Computer Design (ICCD), Virtual, October 2020.

NATSA: A Near-Data Processing Accelerator for Time Series Analysis

| | | | |
|--|---|----------------------------------|-----------------------------|
| Ivan Fernandez [§] | Ricardo Quisiant [§] | Christina Giannoula [†] | Mohammed Alser [‡] |
| Juan Gómez-Luna [‡] | Eladio Gutiérrez [§] | Oscar Plata [§] | Onur Mutlu [‡] |
| [§] <i>University of Malaga</i> | [†] <i>National Technical University of Athens</i> | [‡] <i>ETH Zürich</i> | |

PIM Review and Open Problems

A Modern Primer on Processing in Memory

Onur Mutlu^{a,b}, Saugata Ghose^{b,c}, Juan Gómez-Luna^a, Rachata Ausavarungnirun^d

SAFARI Research Group

^a*ETH Zürich*

^b*Carnegie Mellon University*

^c*University of Illinois at Urbana-Champaign*

^d*King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
"A Modern Primer on Processing in Memory"
*Invited Book Chapter in **Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann**, Springer, to be published in 2021.*

PIM Review and Open Problems (II)

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 (III)

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

Fundamentally High-Performance (Data-Centric) Computing Architectures

Computing Architectures with Minimal Data Movement

One Important Takeaway

Main Memory Needs
Intelligent Controllers

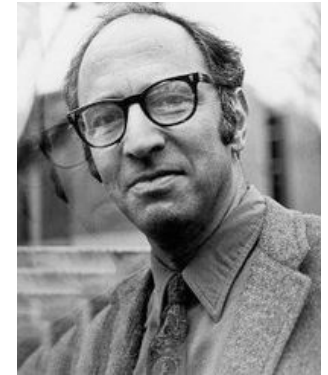
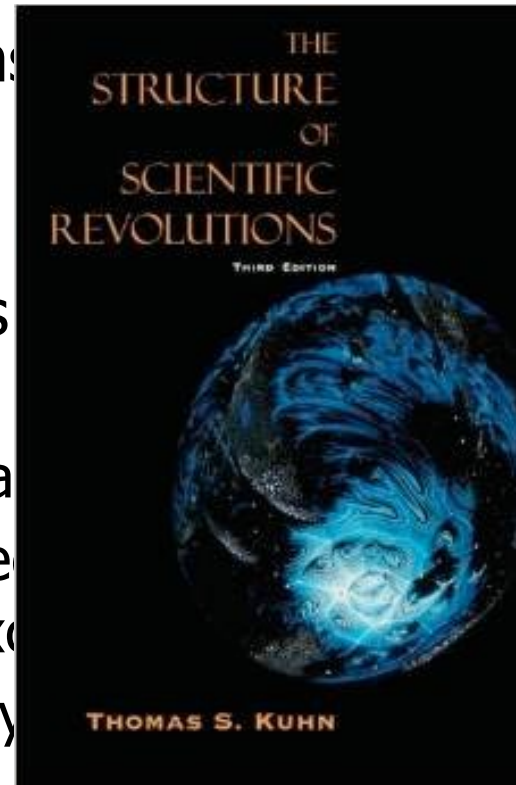
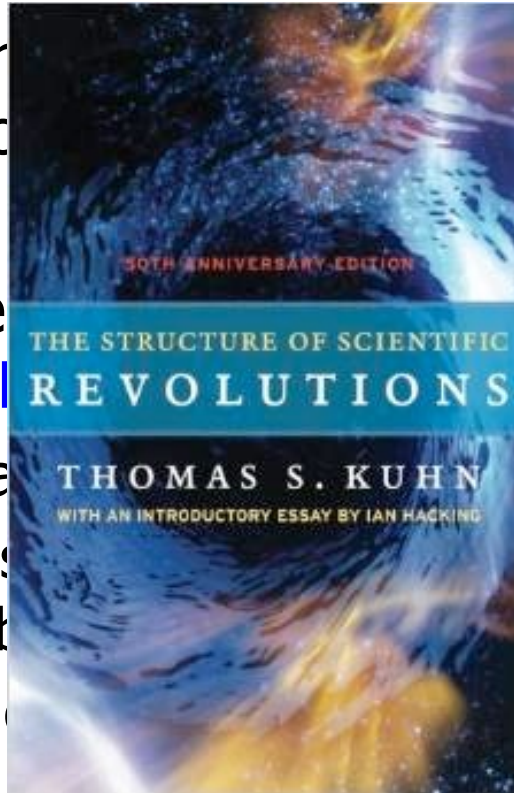
Enabling the Paradigm Shift

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 improve communication
- Recommended reading:
 - **Scientific Revolutions**
 - Pre-para
 - Normal s
 - things (b
 - Revolution

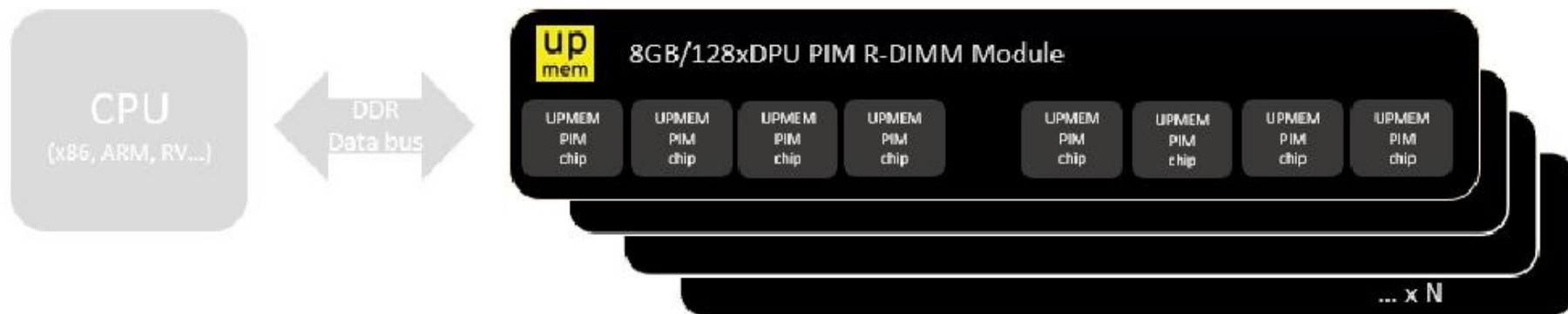


ure of

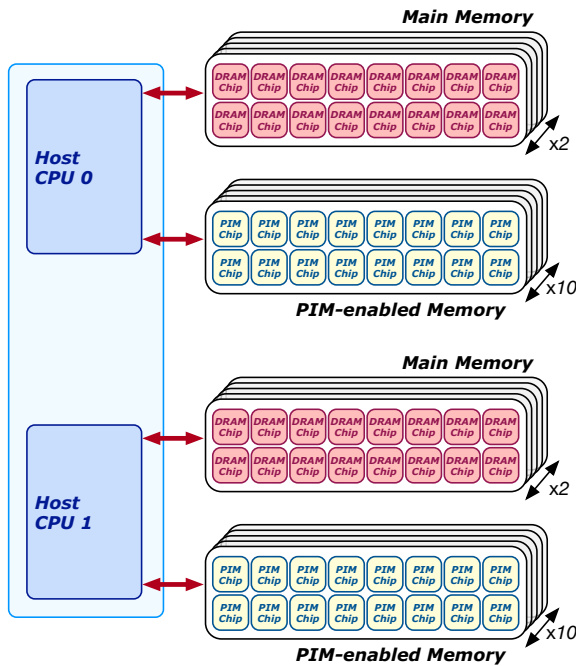
eld
improve
anomalies
examined

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



2,560-DPU Processing-in-Memory System



Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture

JUAN GÓMEZ-LUNA, ETH Zürich, Switzerland
 IZZAT EL HAJJ, American University of Beirut, Lebanon
 IVAN FERNANDEZ, ETH Zürich, Switzerland and University of Malaga, Spain
 CHRISTINA GIANNOULA, ETH Zürich, Switzerland and NTUA, Greece
 GERALDO F. OLIVEIRA, ETH Zürich, Switzerland
 ONUR MUTLU, ETH Zürich, Switzerland

Many modern workloads, such as neural networks, databases, and graph processing, are fundamentally memory-bound. For such workloads, the data movement between main memory and CPU cores imposes a significant overhead in terms of both latency and energy. A major reason is that this communication happens through a narrow bus with high latency and limited bandwidth, and the low data reuse in memory-bound workloads is insufficient to amortize the cost of main memory access. Fundamentally addressing this *data movement bottleneck* requires a paradigm where the memory system assumes an active role in computing by integrating processing capabilities. This paradigm is known as *processing-in-memory (PIM)*.

Recent research explores different forms of PIM architectures, motivated by the emergence of new 3D-stacked memory technologies that integrate memory with a logic layer where processing elements can be easily placed. Past works evaluate these architectures in simulation or, at best, with simplified hardware prototypes. In contrast, the UPMEM company has designed and manufactured the first publicly-available real-world PIM architecture. The UPMEM PIM architecture combines traditional DRAM memory arrays with general-purpose in-order cores, called *DRAM Processing Units (DPUs)*, integrated in the same chip.

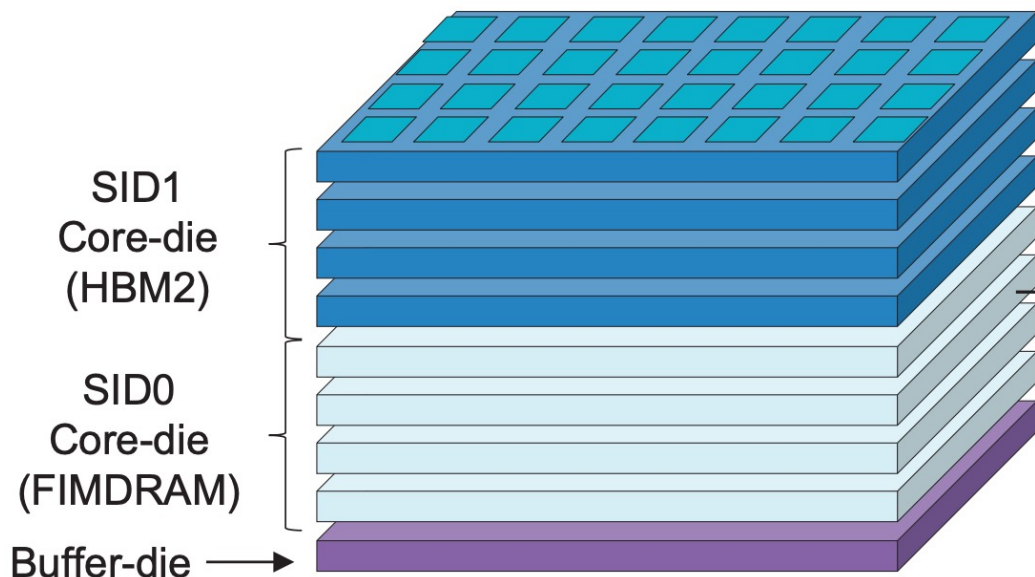
This paper provides the first comprehensive analysis of the first publicly-available real-world PIM architecture. We make two key contributions. First, we conduct an experimental characterization of the UPMEM-based PIM system using microbenchmarks to assess various architecture limits such as compute throughput and memory bandwidth, yielding new insights. Second, we present *PrIM (Processing-In-Memory benchmarks)*, a benchmark suite of 16 workloads from different application domains (e.g., dense/sparse linear algebra, databases, data analytics, graph processing, neural networks, bioinformatics, image processing), which we identify as memory-bound. We evaluate the performance and scaling characteristics of PrIM benchmarks on the UPMEM PIM architecture, and compare their performance and energy consumption to their state-of-the-art CPU and GPU counterparts. Our extensive evaluation conducted on two real UPMEM-based PIM systems with 640 and 2,560 DPUs provides new insights about suitability of different workloads to the PIM system, programming recommendations for software designers, and suggestions and hints for hardware and architecture designers of future PIM systems.



<https://arxiv.org/pdf/2105.03814.pdf>

Samsung Function-in-Memory DRAM (2021)

■ FIMDRAM based on HBM2



[3D Chip Structure of HBM with FIMDRAM]

Chip Specification

128DQ / 8CH / 16 banks / BL4

32 PCU blocks (1 FIM block/2 banks)

1.2 TFLOPS (4H)

**FP16 ADD /
Multiply (MUL) /
Multiply-Accumulate (MAC) /
Multiply-and- Add (MAD)**

ISSCC 2021 / SESSION 25 / DRAM / 25.4

25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications

Young-Cheon Kwon¹, Suk Han Lee¹, Jaehoon Lee¹, Sang-Hyuk Kwon¹, Je Min Ryu¹, Jong-Pil Son¹, Seongil O¹, Hak-Soo Yu¹, Haesuk Lee¹, Soo Young Kim¹, Youngmin Cho¹, Jin Guk Kim¹, Jongyoon Choi¹, Hyun-Sung Shin¹, Jin Kim¹, BengSeng Phuah¹, HyoungMin Kim¹, Myeong Jun Song¹, Ahn Choi¹, Daeho Kim¹, SooYoung Kim¹, Eun-Bong Kim¹, David Wang², Shinhaeng Kang¹, Yuhwan Ro³, Seungwoo Seo³, JoonHo Song³, Jaeyoun Youn¹, Kyomin Sohn¹, Nam Sung Kim¹

¹Samsung Electronics, Hwaseong, Korea

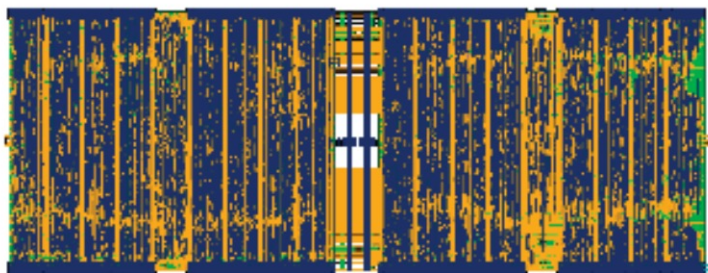
²Samsung Electronics, San Jose, CA

³Samsung Electronics, Suwon, Korea

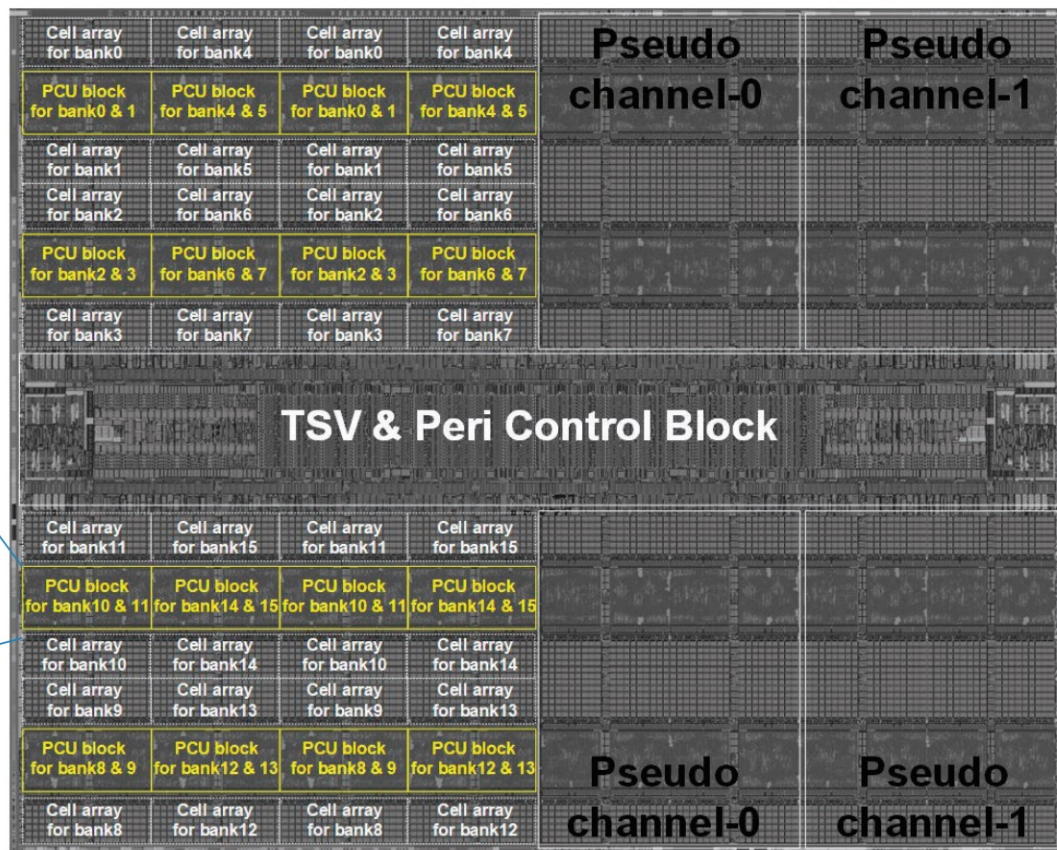
Samsung Function-in-Memory DRAM (2021)

Chip Implementation

- Mixed design methodology to implement FIMDRAM
 - Full-custom + Digital RTL



[Digital RTL design for PCU block]



ISSCC 2021 / SESSION 25 / DRAM / 25.4

25.4 A 20nm 6GB Function-In-Memory DRAM, Based on HBM2 with a 1.2TFLOPS Programmable Computing Unit Using Bank-Level Parallelism, for Machine Learning Applications

Young-Cheon Kwon¹, Suk Han Lee¹, Jaehoon Lee¹, Sang-Hyuk Kwon¹, Je Min Ryu¹, Jong-Pil Son¹, Seongil O¹, Hak-Soo Yu¹, Haesuk Lee¹, Soo Young Kim¹, Youngmin Cho¹, Jin Guk Kim¹, Jongyeon Choi¹, Hyun-Sung Shim¹, Jin Kim¹, BengSeng Phuah¹, HyounMin Kim¹, Myeong Jun Song¹, Ahn Chai¹, Daeho Kim¹, SooYoung Kim¹, Eun-Bong Kim¹, David Wang², Shintae Kang³, Yulwan Ro¹, Seungwoo Seo¹, JoonHo Song¹, Jaeyoun Yoon¹, Kyomin Sohn¹, Nam Sung Kim¹

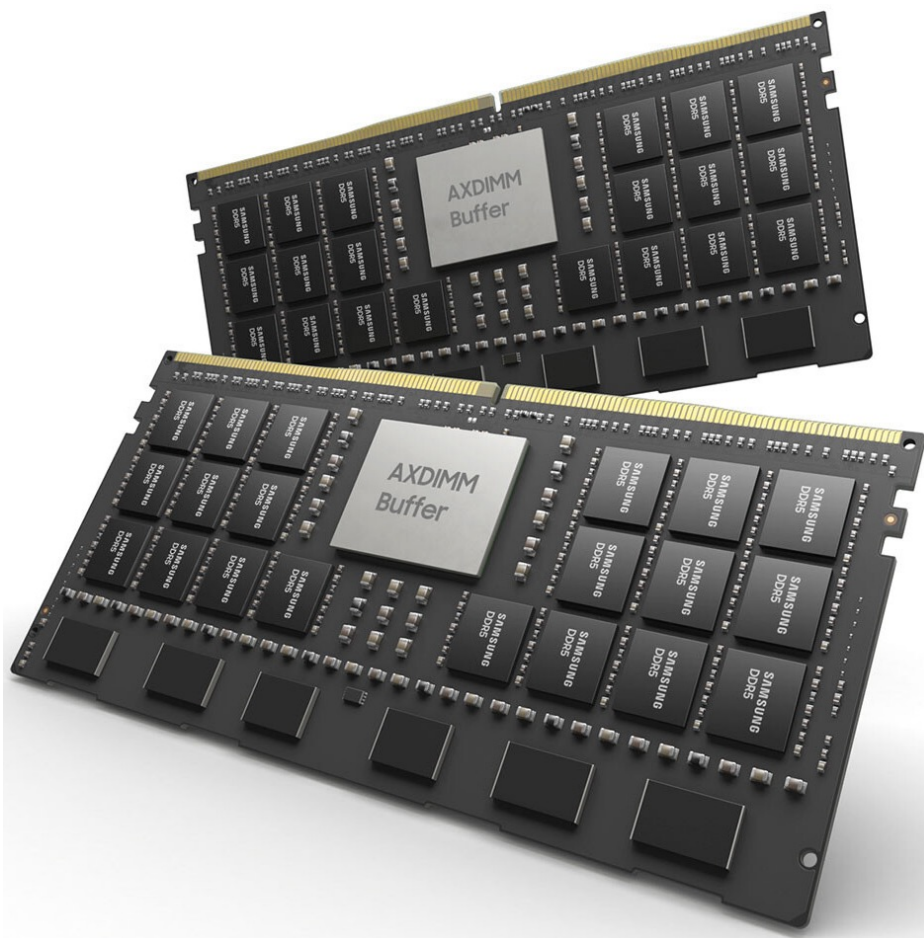
¹Samsung Electronics, Hwaseong, Korea

²Samsung Electronics, San Jose, CA

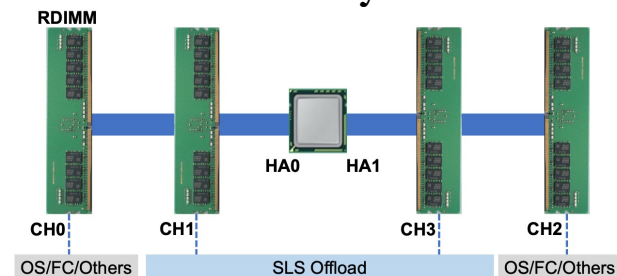
³Samsung Electronics, Suwon, Korea

Samsung AxDIMM (2021)

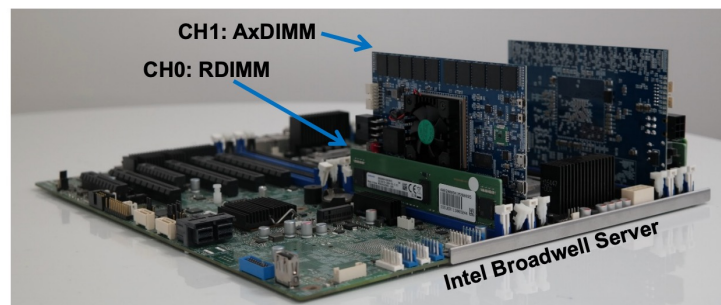
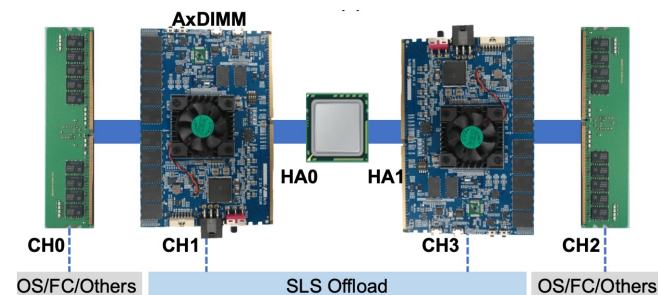
- DIMM-based PIM
 - DLRM recommendation system



Baseline System



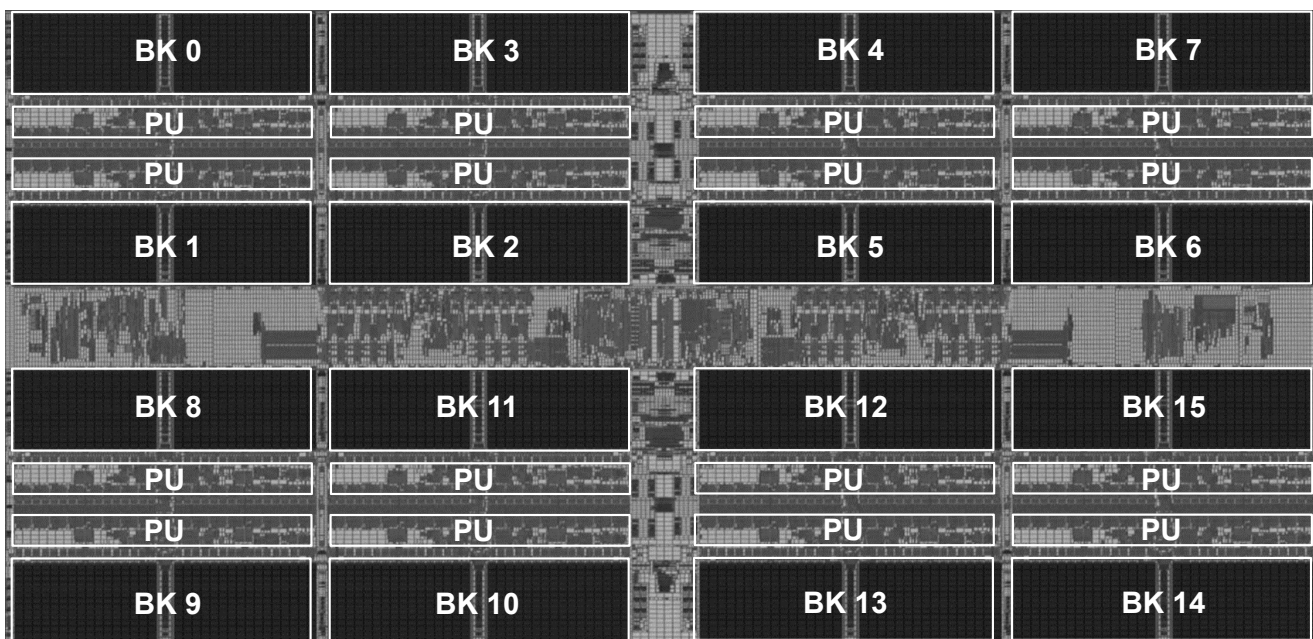
AxDIMM System



SK Hynix AiM: Chip Implementation (2022)

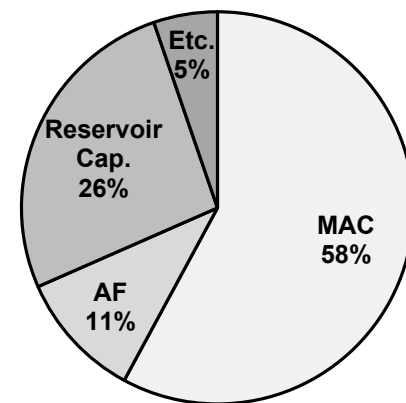
- 4 Gb AiM die with 16 processing units (PUs)

AiM Die Photograph



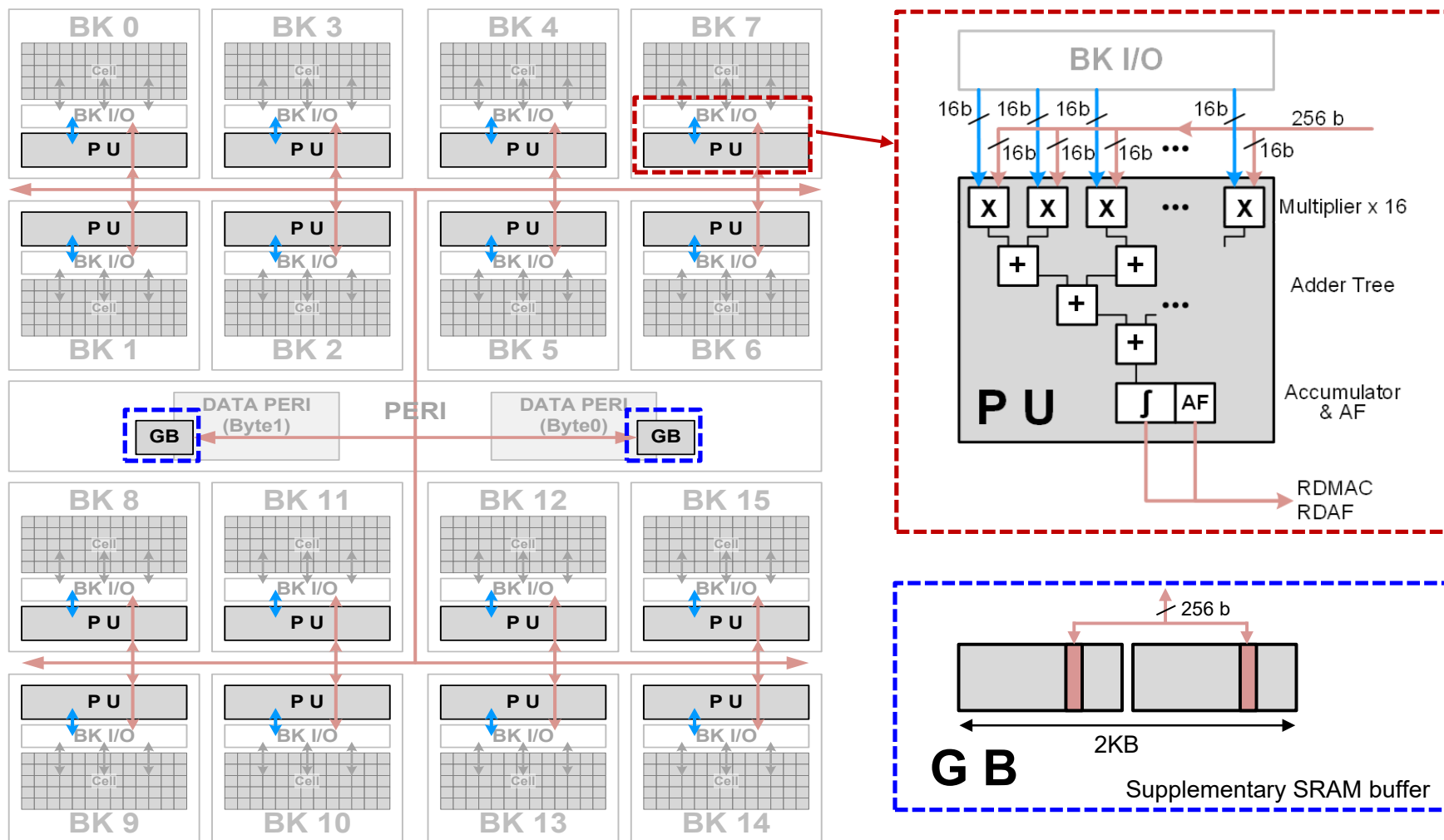
1 Process Unit (PU) Area

| | |
|--------------------------|---------------------|
| Total | 0.19mm ² |
| MAC | 0.11mm ² |
| Activation Function (AF) | 0.02mm ² |
| Reservoir Cap. | 0.05mm ² |
| Etc. | 0.01mm ² |



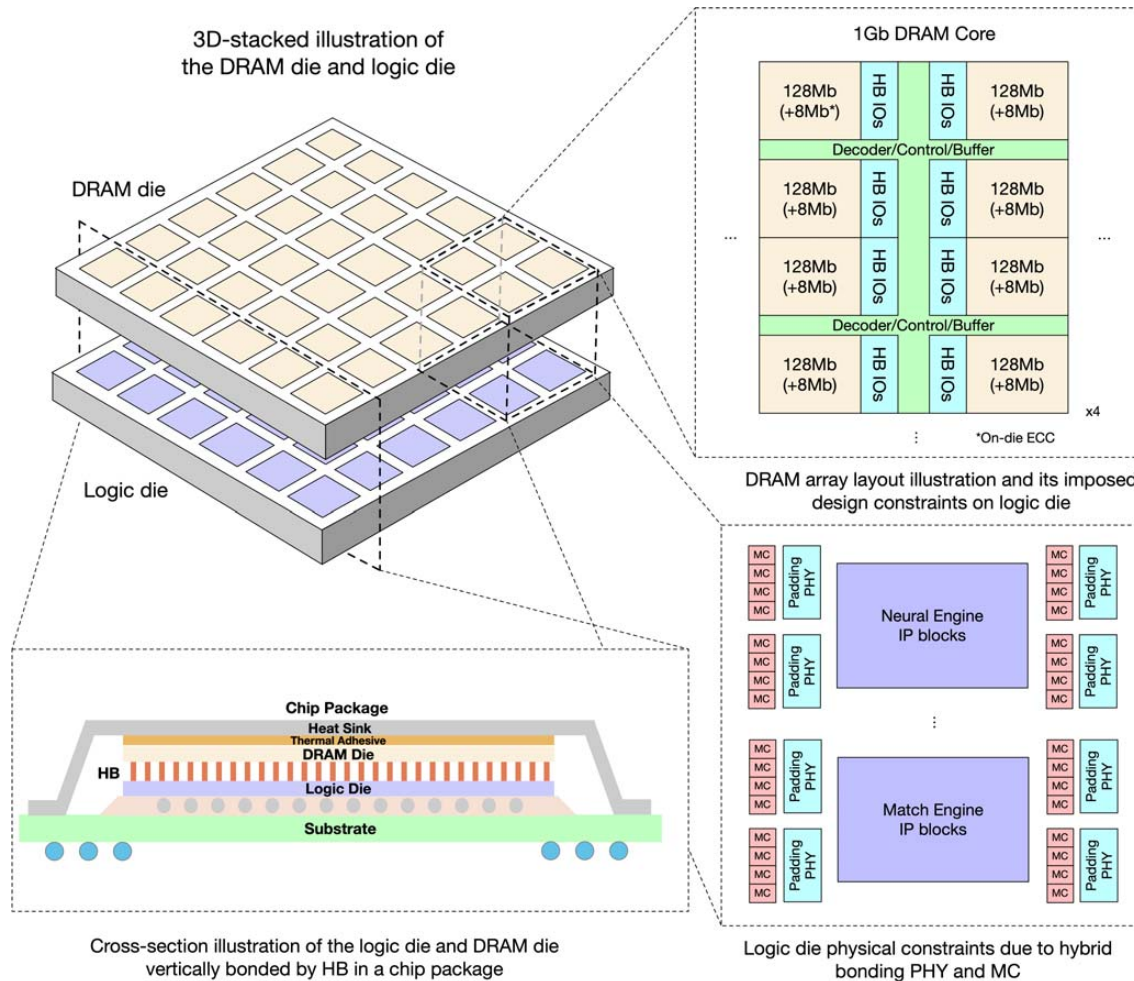
SK Hynix AiM: System Organization (2022)

■ GDDR6-based AiM architecture

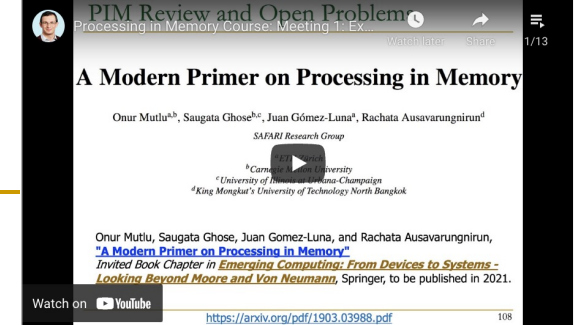


Alibaba HB-PNM: Overall Architecture (2022)

- 3D-stacked logic die and DRAM die vertically bonded by hybrid bonding (HB)



PIM Course (Spring 2022)



Spring 2022 Meetings/Schedule

| Week | Date | Livestream | Meeting | Learning Materials | Assignments |
|------|--------------------------|------------------|---|---|-------------|
| W1 | 10.03 Thu. | YouTube Live | M1: P&S PIM Course Presentation (PDF) (PPT) | Required Materials Recommended Materials | HW 0 Out |
| W2 | 15.03 Tue. 17.03 Thu. | YouTube Premiere | M2: Real-world PIM: UPMEM PIM (PDF) (PPT) | | |
| W3 | 24.03 Thu. | YouTube Live | M3: Real-world PIM: Microbenchmarking of UPMEM PIM (PDF) (PPT) | | |
| W4 | 31.03 Thu. | YouTube Live | M4: Real-world PIM: Samsung HBM-PIM (PDF) (PPT) | | |
| W5 | 07.04 Thu. | YouTube Live | M5: How to Evaluate Data Movement Bottlenecks (PDF) (PPT) | | |
| W6 | 14.04 Thu. | YouTube Live | M6: Real-world PIM: SK Hynix AIM (PDF) (PPT) | | |
| W7 | 21.04 Thu. | YouTube Premiere | M7: Programming PIM Architectures (PDF) (PPT) | | |
| W8 | 28.04 Thu. | YouTube Premiere | M8: Benchmarking and Workload Suitability on PIM (PDF) (PPT) | | |
| W9 | 05.05 Thu. | YouTube Premiere | M9: Real-world PIM: Samsung AxoDIMM (PDF) (PPT) | | |
| W10 | 12.05 Thu. | YouTube Premiere | M10: Real-world PIM: Alibaba HB-PNM (PDF) (PPT) | | |
| W11 | 19.05 Thu. | YouTube Live | M11: SpMV on a Real PIM Architecture (PDF) (PPT) | | |
| W12 | 26.05 Thu. | YouTube Live | M12: End-to-End Framework for Processing-using-Memory (PDF) (PPT) | | |
| W13 | 02.06 Thu. | YouTube Live | M13: Bit-Serial SIMD Processing using DRAM (PDF) (PPT) | | |
| W14 | 09.06 Thu. | YouTube Live | M14: Analyzing and Mitigating ML Inference Bottlenecks (PDF) (PPT) | | |
| W15 | 15.06 Thu. | YouTube Live | M15: In-Memory HTAP Databases with HW/SW Co-design (PDF) (PPT) | | |
| W16 | 23.06 Thu. | YouTube Live | M16: In-Storage Processing for Genome Analysis (PDF) (PPT) | | |
| W17 | 18.07 Mon. | YouTube Premiere | M17: How to Enable the Adoption of PIM? (PDF) (PPT) | | |
| W18 | 09.08 Tue. | YouTube Premiere | SS1: ISVLSI 2022 Special Session on PIM (PDF & PPT) | | |

Spring 2022 Edition:

- https://safari.ethz.ch/projects_and_seminars/spring2022/doku.php?id=processing_in_memory

Youtube Livestream:

- <https://www.youtube.com/watch?v=9e4Chnwdovo&list=PL5Q2soXY2Zi-841fUYYUK9EsXKhQKRPyX>

Project course

- Taken by Bachelor's/Master's students
- Processing-in-Memory lectures
- Hands-on research exploration
- Many research readings

<https://www.youtube.com/onurmutlulectures>

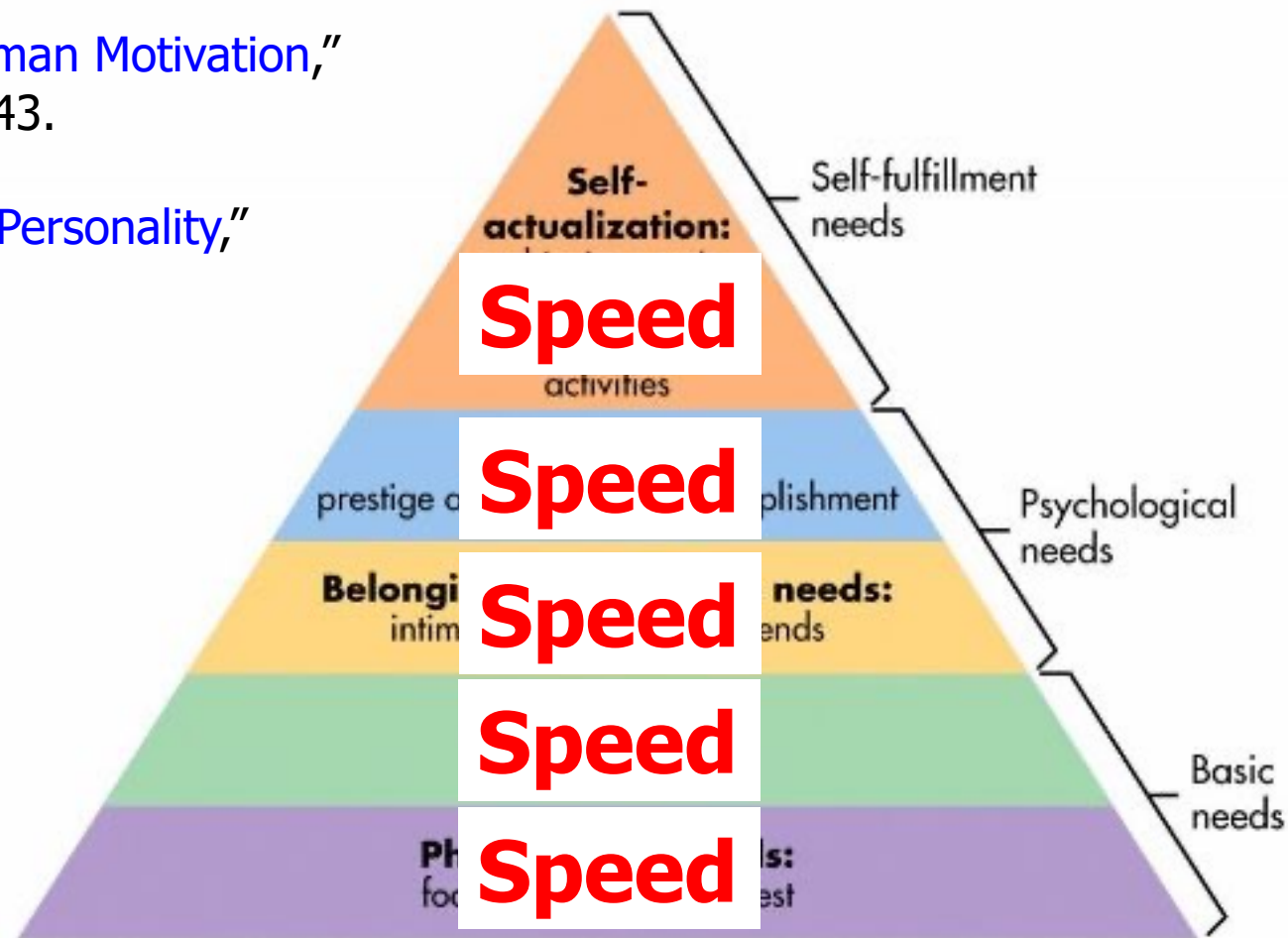
Sub-Agenda: In-Memory Computation

- 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
 - Processing using Memory
 - Processing near Memory
- How to Enable Adoption of Processing in Memory
- Conclusion

Maslow's Hierarchy of Needs, A Third Time

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

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



Fundamentally High-Performance (Data-Centric) Computing Architectures

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures

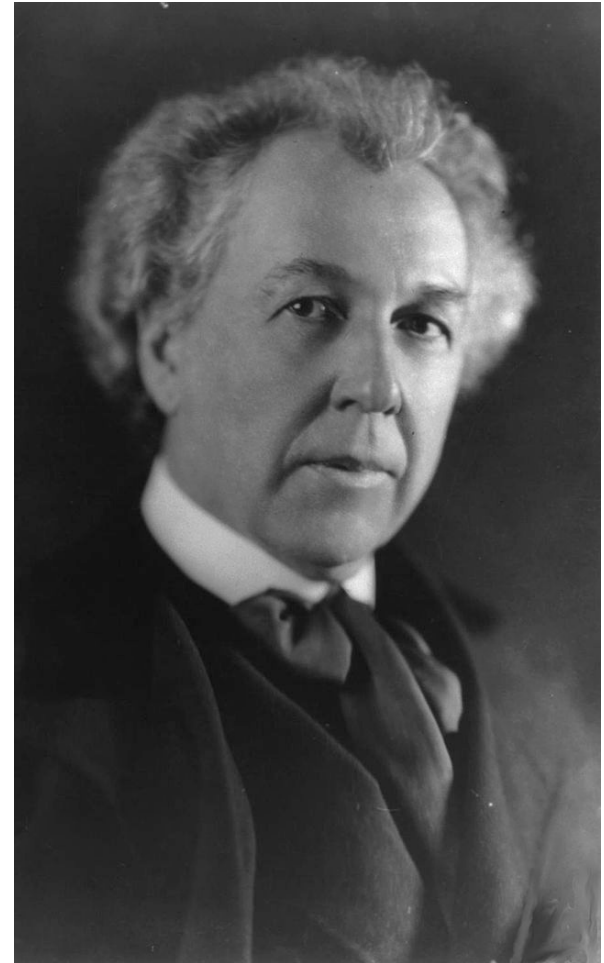
Fundamentally Low-Latency (Data-Centric) Computing Architectures

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**”





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: By Martín Gómez Tagle - Lisbon, Portugal, CC BY-SA 3.0, <https://commons.wikimedia.org/w/index.php?curid=13764903>

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

Another Principled Design



Principle Applied to Another Structure



Source: By 準建築人手札網站 Forgemind ArchiMedia - Flickr: IMG_2489.JPG, CC BY 2.0

Source: <https://www.dezeen.com/2016/08/29/santiago-calatrava-oculus-world-trade-center-transportation-hub-new-york-photographs-hufton-crow/>
<https://commons.wikimedia.org/wiki/index.php?curid=91498396>, https://en.wikipedia.org/wiki/Santiago_Calatrava

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

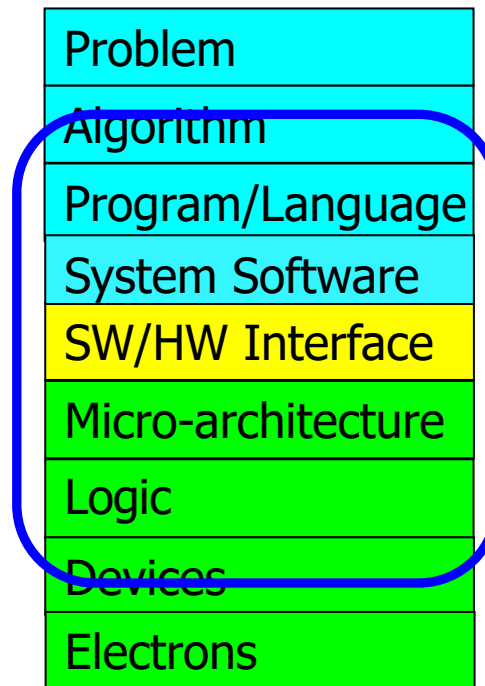
- It is time to design **principled system architectures** to solve the **memory problem**
- Design complete systems to be balanced, high-performance, and energy-efficient, i.e., **data-centric (or memory-centric)**
- Enable computation capability inside and close to memory
- **This** can
 - ❑ Lead to **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:

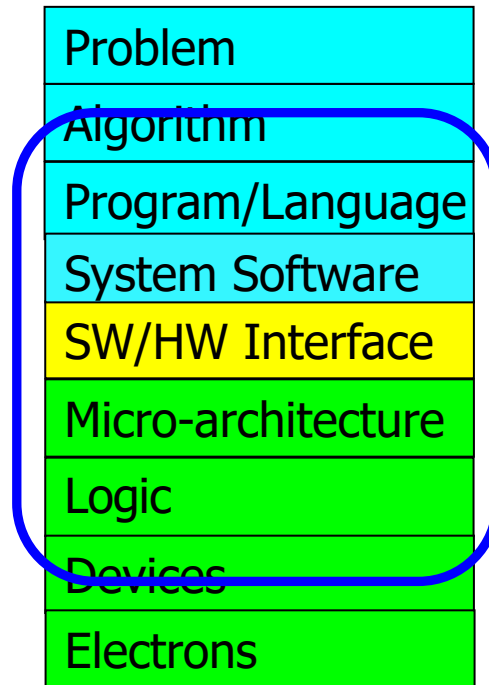
- Orders of magnitude improvements
- New applications and computing systems



Yet, we have to

- Think across the stack
- Design enabling systems

We Need to Revisit the Entire Stack



We can get there step by step

We Need to Exploit Good Principles

- Data-centric system design
- All components intelligent
- Better cross-layer communication, better interfaces
- Better-than-worst-case design
- Heterogeneity
- Flexibility, adaptability

Open minds

If In Doubt, See Other Doubtful Technologies

- A very “doubtful” emerging technology
 - for at least two decades



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

Flash Memory Timeline

Flash Memory Timeline

1967

Dawon Kahng and Simon M. Sze invent the Non-Volatile Memory Floating Gate at Bell Labs; this is published as "A Floating Gate and Its Application to Memory Devices" (Bell System Technical Journal). Simon M. Sze went on to receive the 2014 FMS Lifetime Achievement Award

1970

Dov Frohman-Bentchkowsky invents the Erasable Programmable Read-Only Memory (EPROM) at Intel; this is published as "Memory Behavior in a Floating-Gate Avalanche-Injection MOS (FAMOS) Structure" in April 1971 (Applied Physics Letters), which cited the 1967 Kahng/Sze Bell Labs Floating Gate publication

1976

Hughes Microelectronics files Eli Harari patent for first practical floating gate EEPROM using thin SiO₂ and Fowler Nordheim tunneling for program and erase. Eli Harari went on to receive the 2012 FMS Lifetime Achievement Award

1977

Eli Harari of Hughes Microelectronics publishes "Conduction and Trapping of Electrons in Highly Stressed Thin Films of Thermal SiO₂" (Applied Physics Letters)

1978

Eli Harari of Hughes Microelectronics publishes "Dielectric Breakdown in Electrically Stressed Thin Films of Thermal SiO₂" (Journal of Applied Physics)

1979

Hughes Microelectronics introduces first CMOS NOR-VRAM 256-bit chip (non-volatile SRAM) employing Fowler Nordheim floating gate EEPROM at IEEE ISSCC

1980

IEEE Solid State Circuits publishes paper titled "An Electrically Alterable Non-Volatile Memory Cell Using Floating Gate Structure" by Gulterman, Rinawi, Chieu, Holvorson, and McEvoy of Texas Instruments

Hughes Microelectronics introduces the 3108, first CMOS EPROM, 8Kb chip employing Fowler Nordheim tunneling

Intel introduces the 2816, 16Kb HMOS EPROM employing Fowler Nordheim tunneling

1981

British scientist and inventor Kane Kramer designs first digital audio player (IXI) based on magnetic bubble memory chips

1982

SEEO Technology introduces the 5213, first EEPROM with on-chip charge pump for in-system write and erase, an invention used in all flash memory devices

1983

Intel introduces 2817A 16Kb EEPROM

1984

First paper describing flash EEPROM presented by Fujio Masuoka of Toshiba at IEEE International Electron Devices Meeting (IEDM) in San Francisco. Fujio Masuoka went on to receive the 2013 FMS Lifetime Achievement Award

Intel begins flash process development. AT&T (Advanced Technology for Memory and Logic) is founded by George Perlegos, who went on to receive the 2017 Lifetime Achievement Award



Flash Memory Summit

Flash Memory Timeline



PIM Review and Open Problems

A Modern Primer on Processing in Memory

Onur Mutlu^{a,b}, Saugata Ghose^{b,c}, Juan Gómez-Luna^a, Rachata Ausavarungnirun^d

SAFARI Research Group

^a*ETH Zürich*

^b*Carnegie Mellon University*

^c*University of Illinois at Urbana-Champaign*

^d*King Mongkut's University of Technology North Bangkok*

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,

"A Modern Primer on Processing in Memory"

*Invited Book Chapter in **Emerging Computing: From Devices to Systems - Looking Beyond Moore and Von Neumann**, Springer, to be published in 2021.*

PIM Review and Open Problems (II)

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 (III)

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]

Computer Architecture

Lecture 4: Processing near Memory

Prof. Onur Mutlu

ETH Zürich

Fall 2022

7 October 2022