

Computer Architecture

Lecture 7: Computation in Memory II

Prof. Onur Mutlu

ETH Zürich

Fall 2019

10 October 2019

Sub-Agenda: In-Memory Computation

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

Processing in Memory: Two Approaches

1. Minimally changing memory chips
2. Exploiting 3D-stacked memory

Recall: 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

Recall: End-to-End System Design

Application

How to communicate occurrences of bulk copy/initialization across layers?

Operating System

How to ensure cache coherence?

ISA

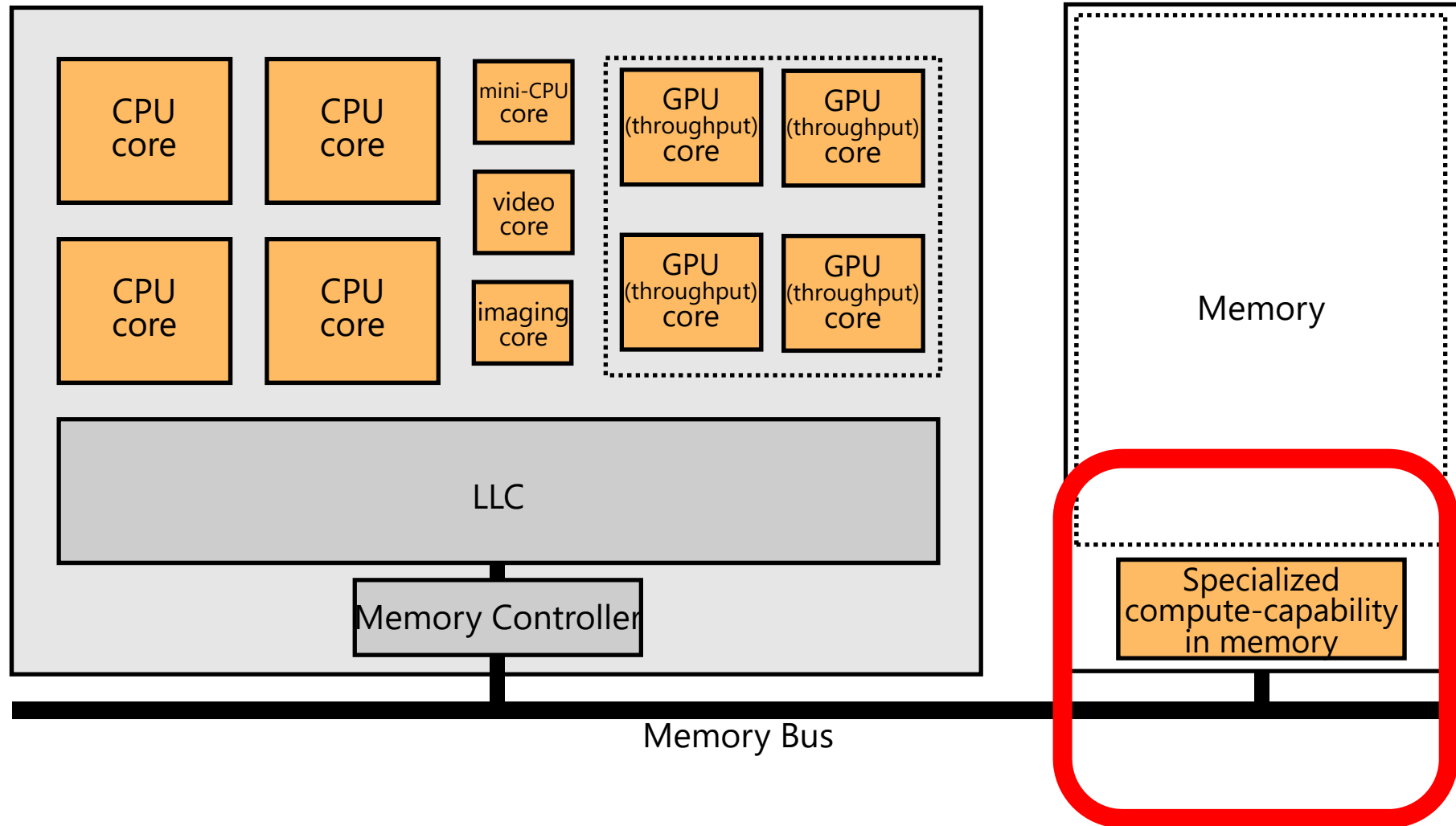
Microarchitecture

How to maximize latency and energy savings?

DRAM (RowClone)

How to handle data reuse?

Memory as an Accelerator

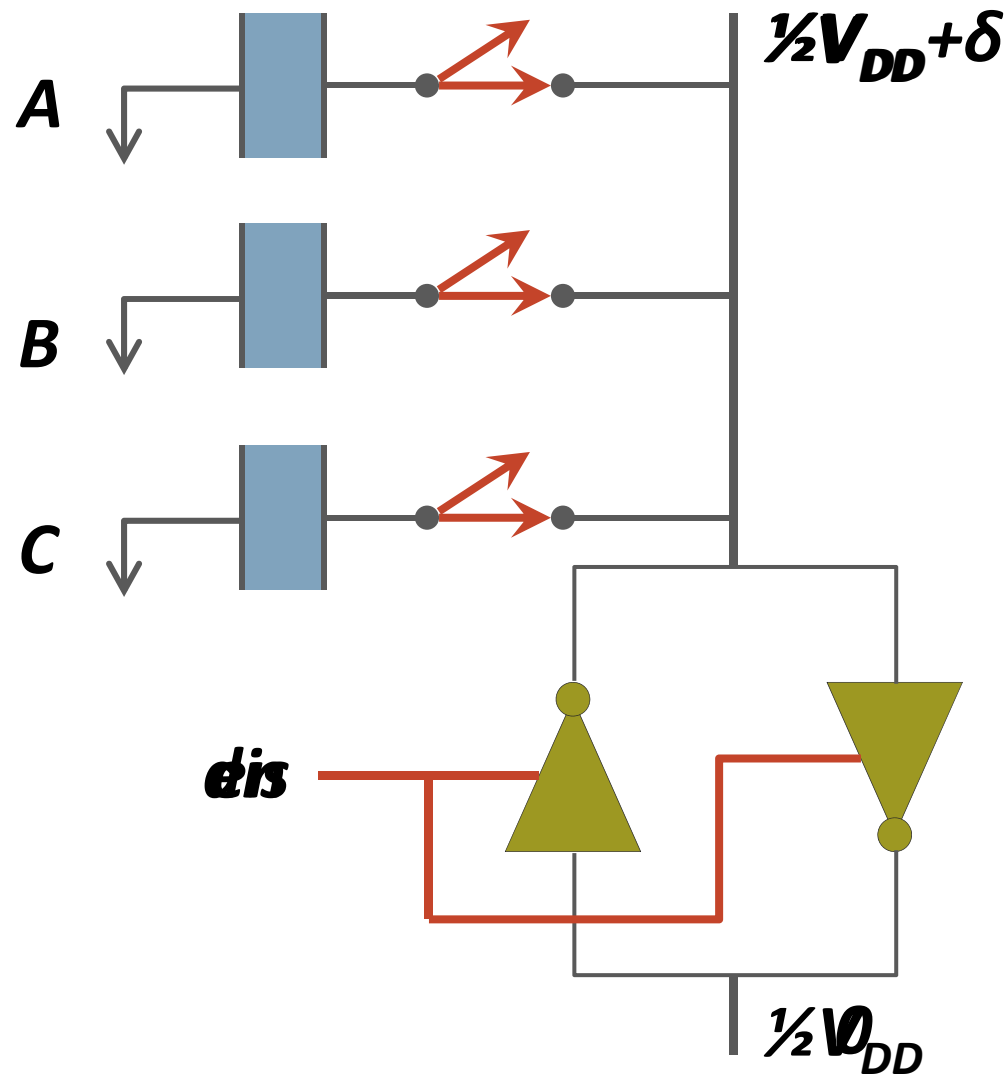


Memory similar to a "conventional" accelerator

In-Memory Bulk Bitwise Operations

- We can support in-DRAM COPY, ZERO, AND, OR, NOT, MAJ
- At low cost
- Using analog computation capability of DRAM
 - Idea: activating multiple rows performs computation
- 30-60X performance and energy improvement
 - Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," MICRO 2017.
- New memory technologies enable even more opportunities
 - Memristors, resistive RAM, phase change mem, STT-MRAM, ...
 - Can operate on data with minimal movement

In-DRAM AND/OR: Triple Row Activation



Final State
 $AB + BC + AC$

**$C(A + B) +$
 $\sim C(AB)$**

In-DRAM Bulk Bitwise AND/OR Operation

- **BULKAND A, B → C**
 - Semantics: Perform a bitwise AND of two rows A and B and store the result in row C
 - R0 – reserved zero row, R1 – reserved one row
 - D1, D2, D3 – Designated rows for triple activation
1. RowClone A into D1
 2. RowClone B into D2
 3. RowClone R0 into D3
 4. ACTIVATE D1,D2,D3
 5. RowClone Result into C

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

In-DRAM NOT: Dual Contact Cell

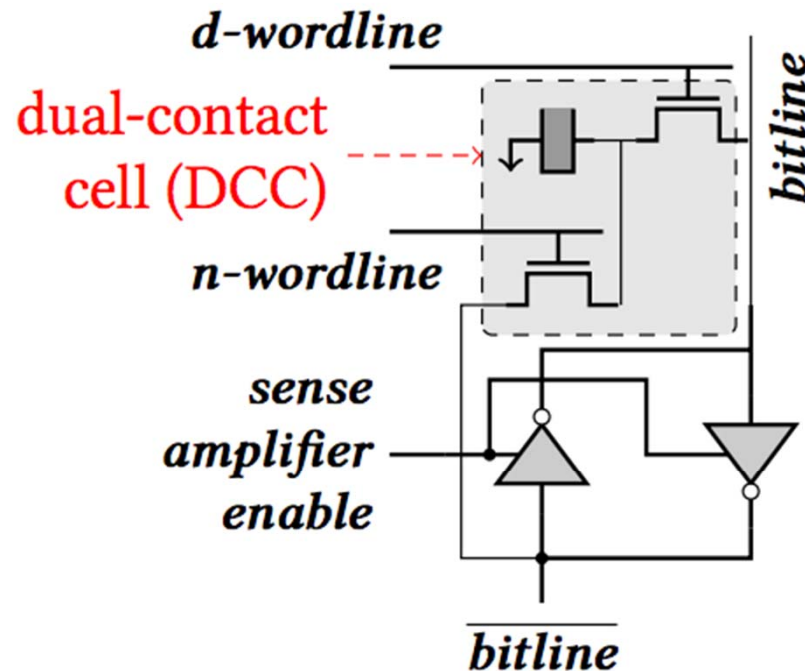


Figure 5: A dual-contact cell connected to both ends of a sense amplifier

Idea:
Feed the
negated value
in the sense amplifier
into a special row

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

In-DRAM NOT Operation

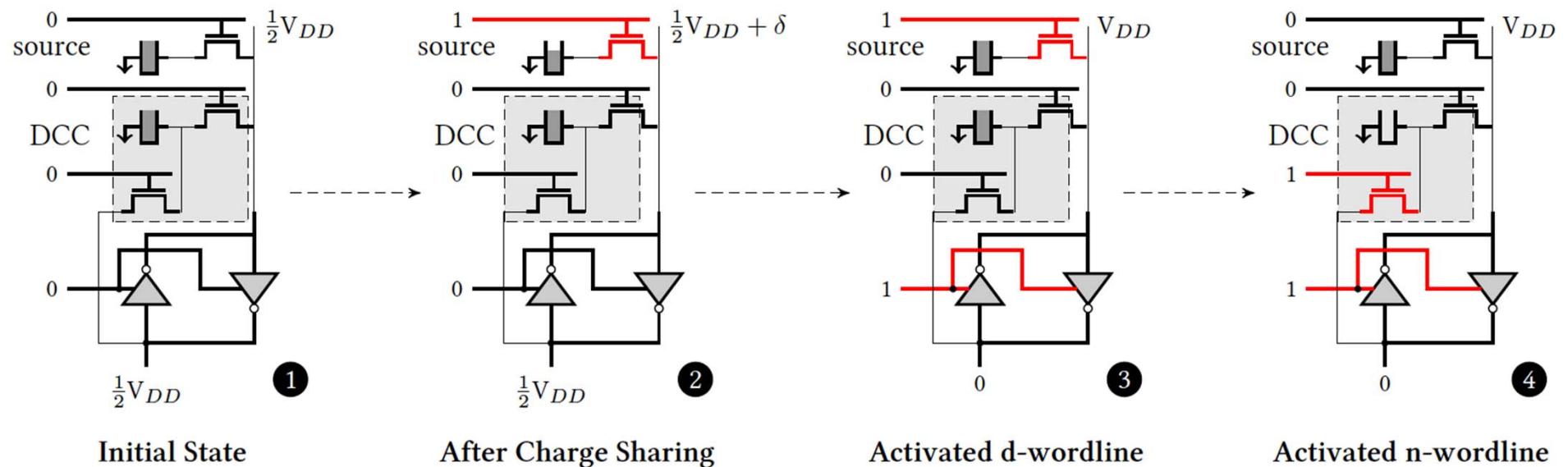


Figure 5: Bitwise NOT using a dual contact capacitor

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

Performance: In-DRAM Bitwise Operations

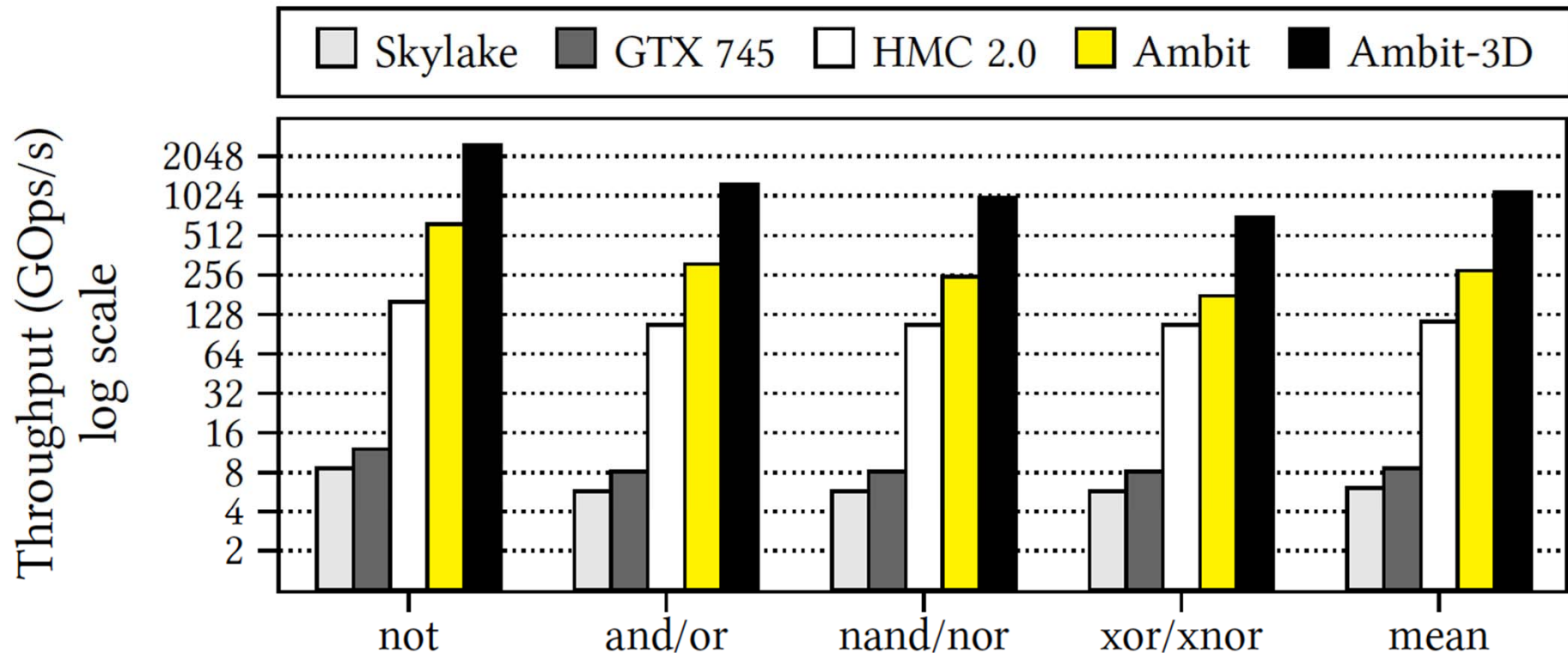


Figure 9: Throughput of bitwise operations on various systems.

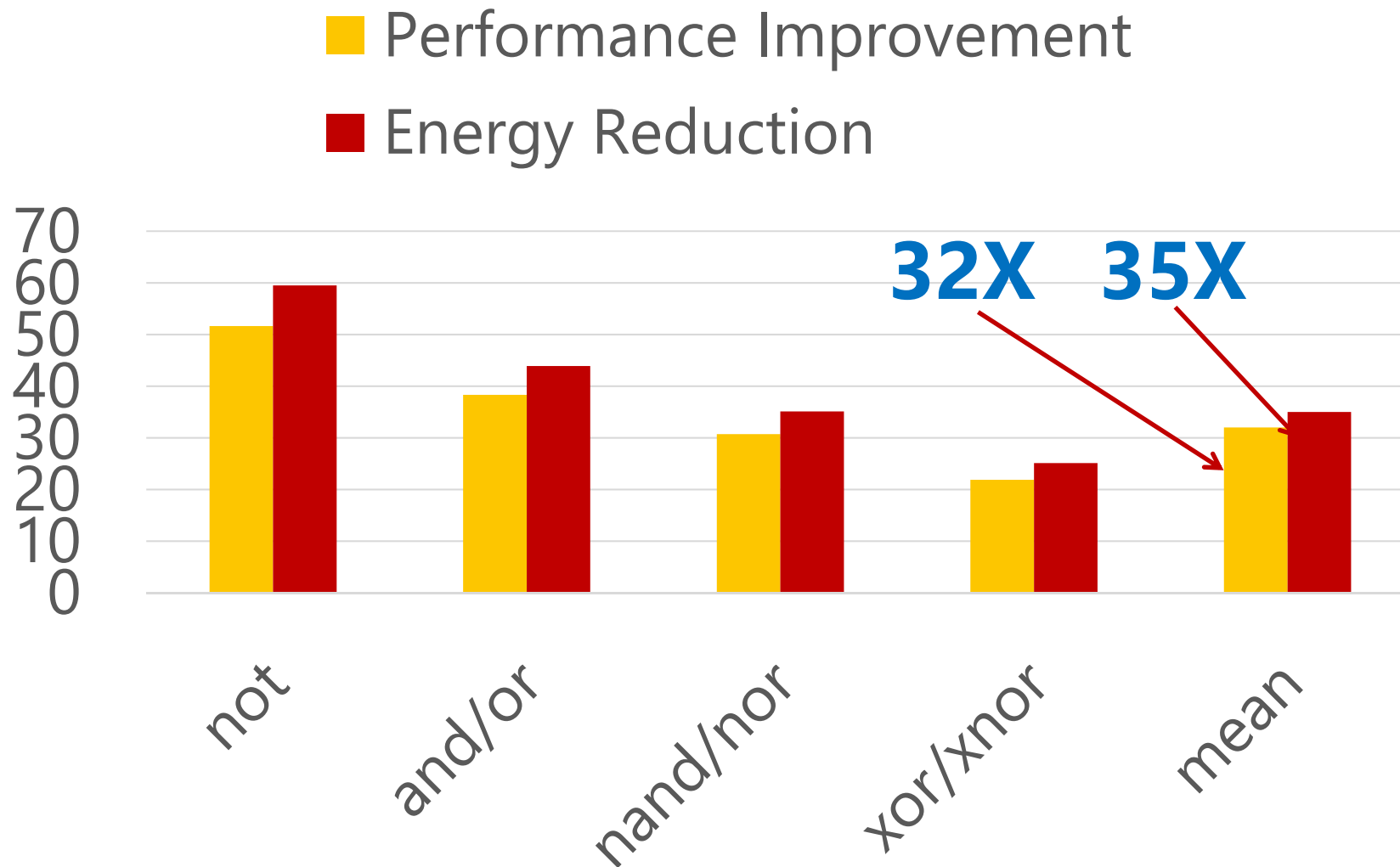
Energy of In-DRAM Bitwise Operations

	Design	not	and/or	nand/nor	xor/xnor
DRAM &	DDR3	93.7	137.9	137.9	137.9
Channel Energy	Ambit	1.6	3.2	4.0	5.5
(nJ/KB)	(↓)	59.5X	43.9X	35.1X	25.1X

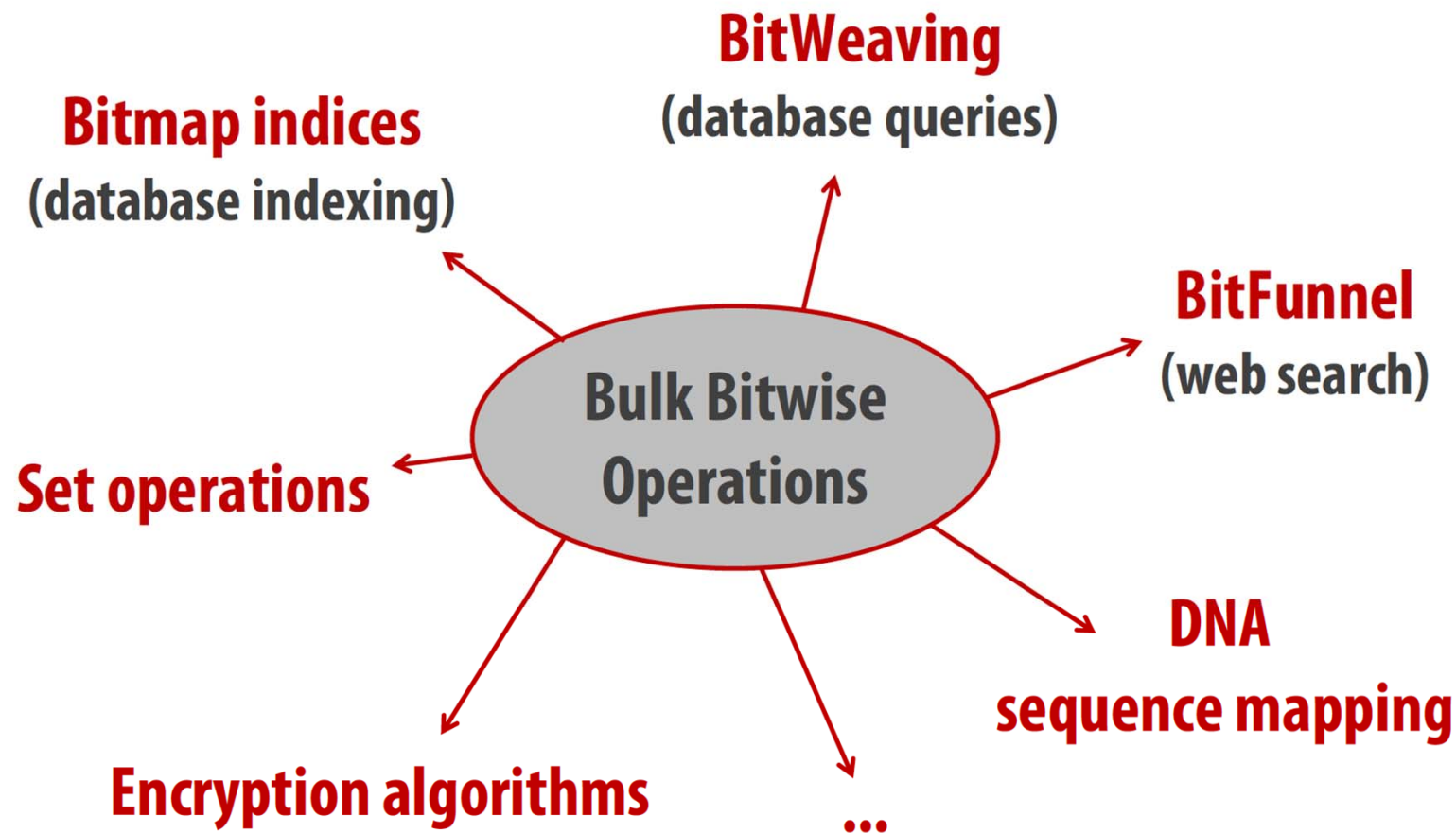
Table 3: Energy of bitwise operations. (↓) indicates energy reduction of Ambit over the traditional DDR3-based design.

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

Ambit vs. DDR3: Performance and Energy

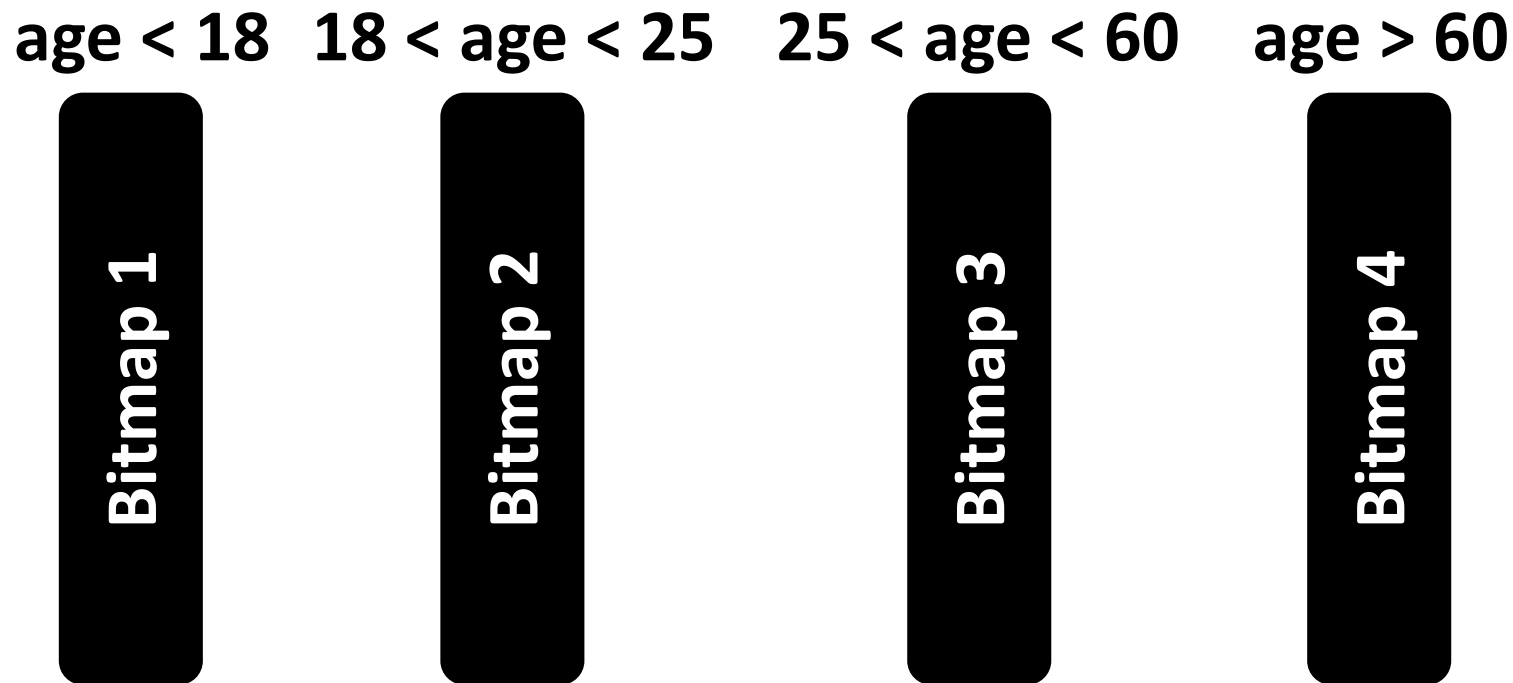


Bulk Bitwise Operations in Workloads



Example Data Structure: Bitmap Index

- Alternative to B-tree and its variants
- Efficient for performing *range queries* and *joins*
- **Many bitwise operations to perform a query**



Performance: Bitmap Index on Ambit

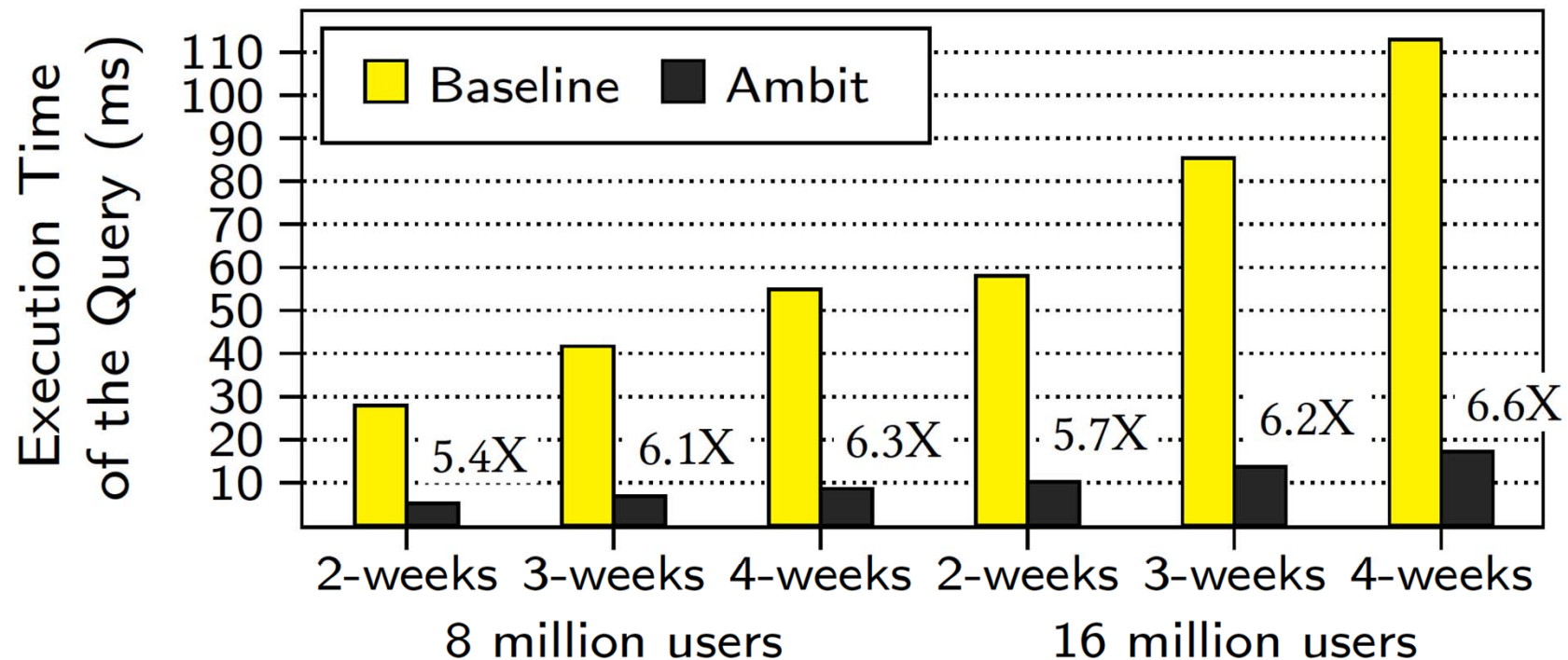


Figure 10: Bitmap index performance. The value above each bar indicates the reduction in execution time due to Ambit.

>5.4-6.6X Performance Improvement

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

Performance: BitWeaving on Ambit

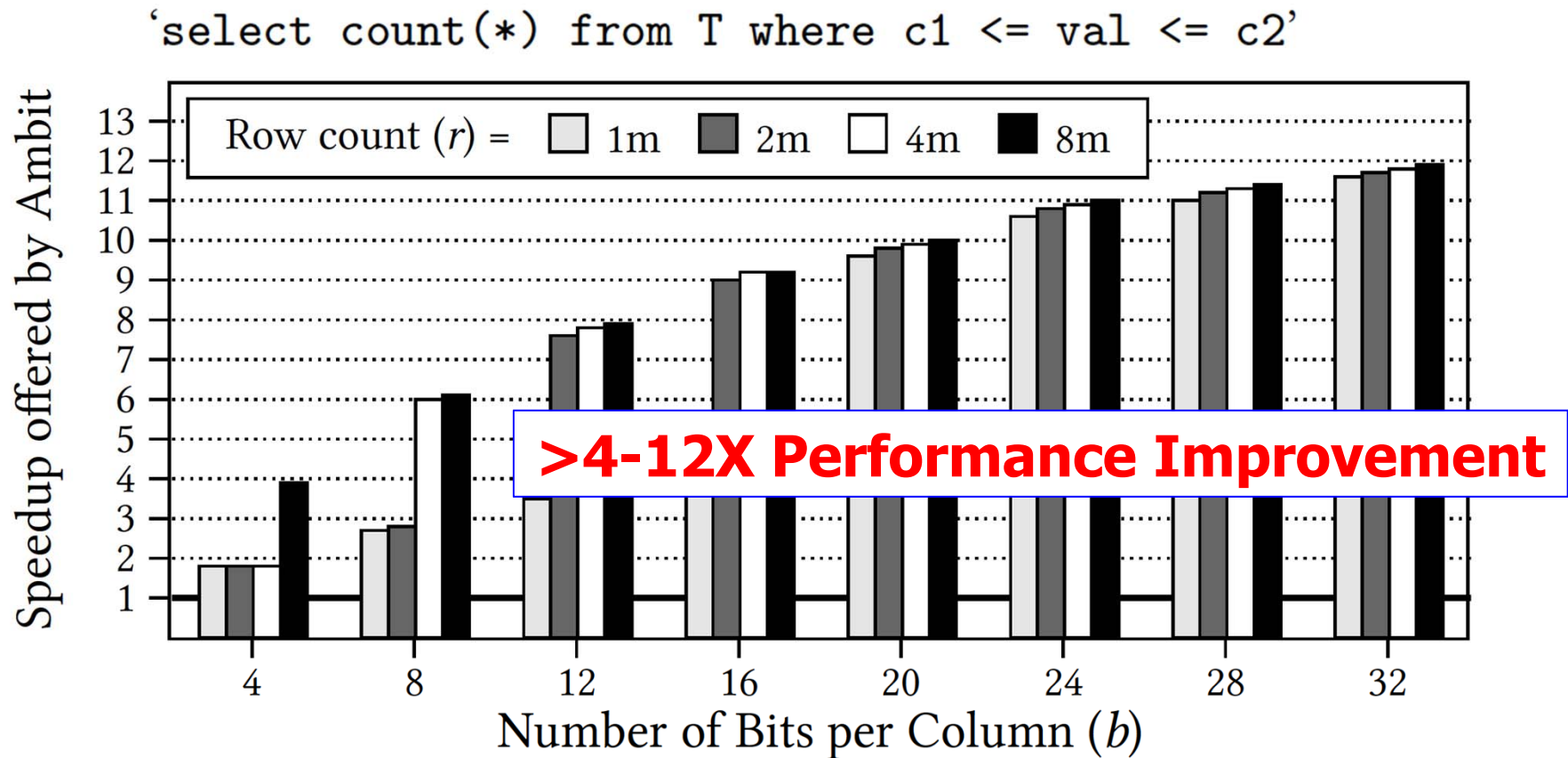


Figure 11: Speedup offered by Ambit over baseline CPU with SIMD for BitWeaving

Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations using Commodity DRAM Technology," MICRO 2017.

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

Challenge: Intelligent Memory Device

Does **memory**
have to be
dumb?

Challenge and Opportunity for Future

Computing Architectures with Minimal Data Movement

A Detour on the Review Process

Ambit Sounds Good, No?

Review from ISCA 2016

Paper summary

The paper proposes to extend DRAM to include bulk, bit-wise logical operations directly between rows within the DRAM.

Strengths

- Very clever/novel idea.
 - Great potential speedup and efficiency gains.
-

Weaknesses

- Probably won't ever be built. Not practical to assume DRAM manufacturers will change DRAM in this way.
-

Another Review

Another Review from ISCA 2016

Strengths

The proposed mechanisms effectively exploit the operation of the DRAM to perform efficient bitwise operations across entire rows of the DRAM.

Weaknesses

This requires a modification to the DRAM that will only help this type of bitwise operation. It seems unlikely that something like that will be adopted.

Yet Another Review

Yet Another Review from ISCA 2016

Weaknesses

The core novelty of Buddy RAM is almost all circuits-related (by exploiting sense amps). I do not find architectural innovation even though the circuits technique benefits architecturally by mitigating memory bandwidth and relieving cache resources within a subarray. The only related part is the new ISA support for bitwise operations at DRAM side and its induced issue on cache coherence.

The Reviewer Accountability Problem

Acknowledgments

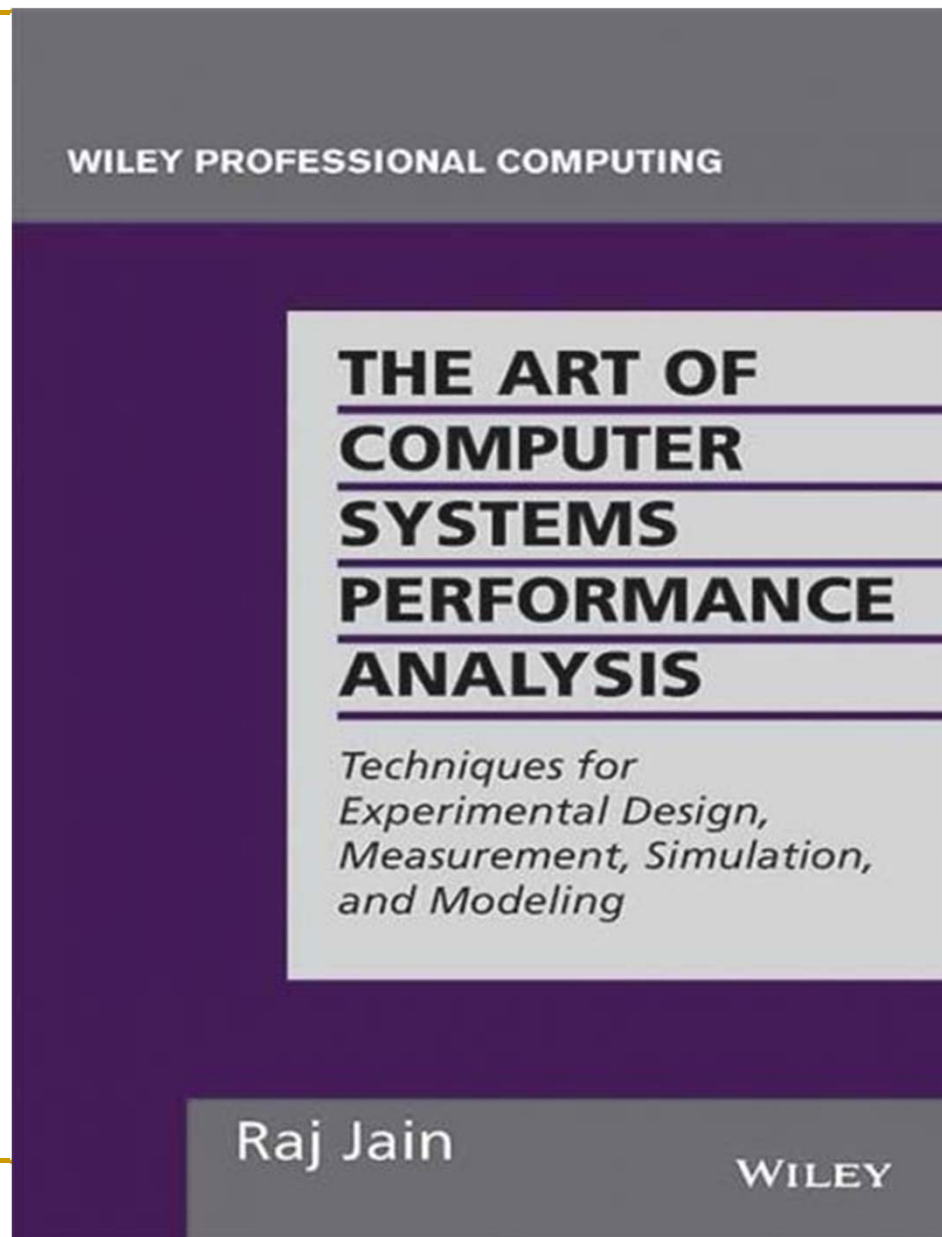
We thank the reviewers of ISCA 2016/2017, MICRO 2016/2017, and HPCA 2017 for their valuable comments. We

We Have a Mindset Issue...

- There are many other similar examples from reviews...
 - For many other papers...
- And, we are not even talking about JEDEC yet...
- How do we fix the mindset problem?
- By doing more research, education, implementation in alternative processing paradigms

We need to work on enabling the better future...

Aside: A Recommended Book



Raj Jain, "[The Art of Computer Systems Performance Analysis](#)," Wiley, 1991.

10.8 DECISION MAKER'S GAMES

Even if the performance analysis is correctly done and presented, it may not be enough to persuade your audience—the decision makers—to follow your recommendations. The list shown in Box 10.2 is a compilation of reasons for rejection heard at various performance analysis presentations. You can use the list by presenting it immediately and pointing out that the reason for rejection is not new and that the analysis deserves more consideration. Also, the list is helpful in getting the competing proposals rejected!

There is no clear end of an analysis. Any analysis can be rejected simply on the grounds that the problem needs more analysis. This is the first reason listed in Box 10.2. The second most common reason for rejection of an analysis and for endless debate is the workload. Since workloads are always based on the past measurements, their applicability to the current or future environment can always be questioned. Actually workload is one of the four areas of discussion that lead a performance presentation into an endless debate. These “rat holes” and their relative sizes in terms of time consumed are shown in Figure 10.26. Presenting this cartoon at the beginning of a presentation helps to avoid these areas.

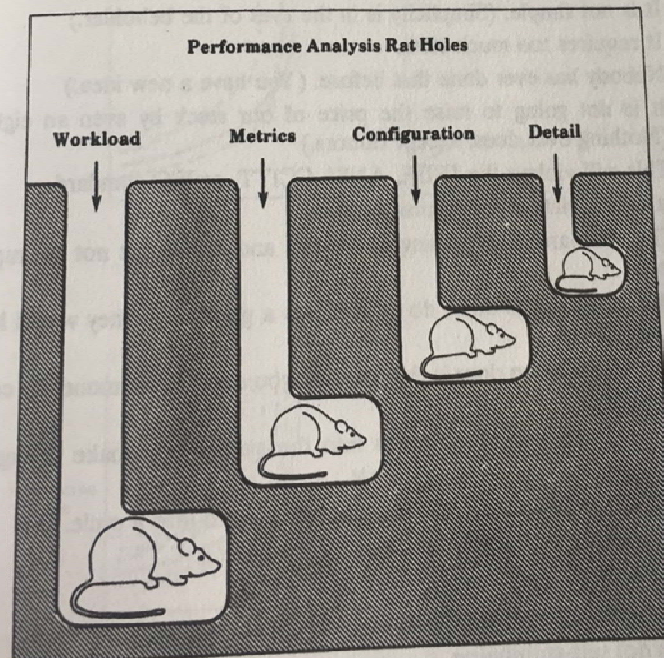


FIGURE 10.26 Four issues in performance presentations that commonly lead to endless discussion.

Raj Jain, "The Art of
Computer Systems
Performance Analysis,"
Wiley, 1991.

Box 10.2 Reasons for Not Accepting the Results of an Analysis

1. This needs more analysis.
2. You need a better understanding of the workload.
3. It improves performance only for long I/O's, packets, jobs, and files, and most of the I/O's, packets, jobs, and files are short.
4. It improves performance only for short I/O's, packets, jobs, and files, but who cares for the performance of short I/O's, packets, jobs, and files; its the long ones that impact the system.
5. It needs too much memory/CPU/bandwidth and memory/CPU/bandwidth isn't free.
6. It only saves us memory/CPU/bandwidth and memory/CPU/bandwidth is cheap.
7. There is no point in making the networks (similarly, CPUs/disks,...) faster; our CPUs/disks (any component other than the one being discussed) aren't fast enough to use them.
8. It improves the performance by a factor of x , but it doesn't really matter at the user level because everything else is so slow.
9. It is going to increase the complexity and cost.
10. Let us keep it simple stupid (and your idea is not stupid).
11. It is not simple. (Simplicity is in the eyes of the beholder.)
12. It requires too much state.
13. Nobody has ever done that before. (You have a new idea.)
14. It is not going to raise the price of our stock by even an eighth. (Nothing ever does, except rumors.)
15. This will violate the IEEE, ANSI, CCITT, or ISO standard.
16. It may violate some future standard.
17. The standard says nothing about this and so it must not be important.
18. Our competitors don't do it. If it was a good idea, they would have done it.
19. Our competition does it this way and you don't make money by copying others.
20. It will introduce randomness into the system and make debugging difficult.
21. It is too deterministic; it may lead the system into a cycle.
22. It's not interoperable.
23. This impacts hardware.
24. That's beyond today's technology.
25. It is not self-stabilizing.
26. Why change—it's working OK.

Raj Jain, "The Art of
Computer Systems
Performance Analysis,"
Wiley, 1991.

We Need to Fix the Reviewer Accountability Problem

Main Memory Needs Intelligent Controllers

Takeaway

Research Community
Needs

Accountable Reviewers

Suggestions to Reviewers

- Be fair; you do not know it all
- Be open-minded; you do not know it all
- Be accepting of diverse research methods: there is no single way of doing research
- Be constructive, not destructive
- Do not have double standards...

Do not block or delay scientific progress for non-reasons

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

Other Examples of “Why Change? It’s Working OK!”

Mindset Issues Are Everywhere

- They limit progress
- Examples of Bandwidth Waste in Real Life
- Examples of Latency and Queueing Delays in Real Life
- Example of Where to Build a Bridge on the Road

Another Example

Initial RowHammer Reviews

Disturbance Errors in DRAM: Demonstration, Characterization, and Prevention

Rejected (R2)



863kB

Friday 31 May 2013 2:00:53pm PDT

b9bf06021da54cddf4cd0b3565558a181868b972

You are an **author** of this paper.

+ **ABSTRACT**

+ **AUTHORS**

[Review #66A](#)
[Review #66B](#)
[Review #66C](#)
[Review #66D](#)
[Review #66E](#)
[Review #66F](#)

OveMer	Nov	WriQua	RevExp
1	4	4	4
5	4	5	3
2	3	5	4
1	2	3	4
4	4	4	3
2	4	4	3

Missing the Point **Reviews from Micro 2013**

PAPER WEAKNESSES

This is an excellent test methodology paper, but there is no micro-architectural or architectural content.

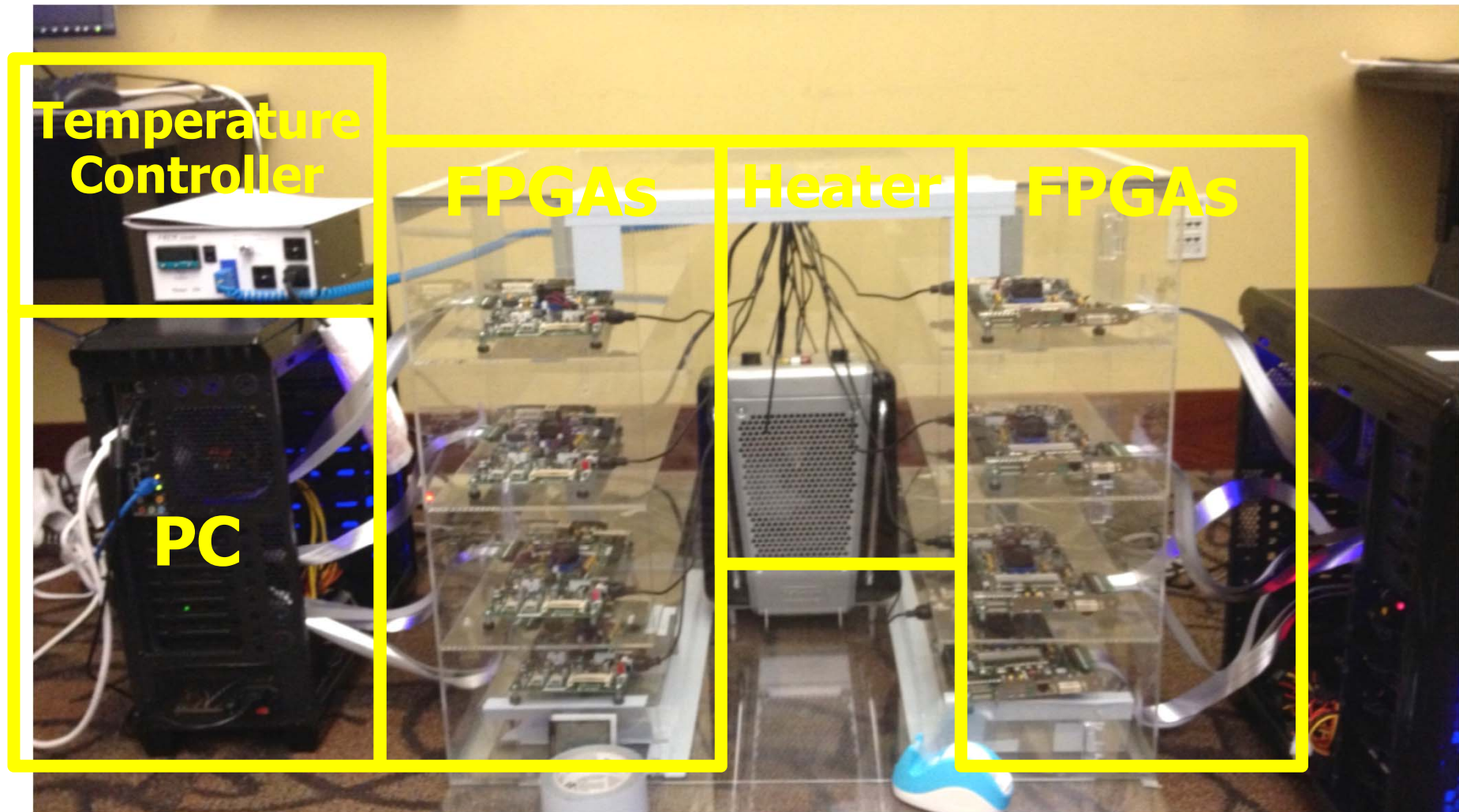
PAPER WEAKNESSES

- Whereas they show disturbance may happen in DRAM array authors don't show it can be an issue in realistic DRAM usage scenario
- Lacks architectural/microarchitectural impact on the DRAM disturbance analysis

PAPER WEAKNESSES

The mechanism investigated by the authors is one of many well known disturb mechanisms. The paper does not discuss the root causes to sufficient depth and the importance of this mechanism compared to others. Overall the length of the sections restating known information is much too long in relation to new work.

Experimental DRAM Testing Infrastructure



Tested DRAM Modules (129 total)

Manufacturer	Module	Date*	Timing†		Organization		Chip			Victims-per-Module			RL _{th} (ms)
		(yy-ww)	Freq (MT/s)	t _{RC} (ns)	Size (GB)	Chips	Size (Gb)‡	Pins	DieVersion§	Average	Minimum	Maximum	Min
A	A ₁	10-08	1066	50.625	0.5	4	1 ×16	B	0	0	0	–	
	A ₂	10-20	1066	50.625	1	8	1 ×8	F	0	0	0	–	
	A ₃₋₅	10-20	1066	50.625	0.5	4	1 ×16	B	0	0	0	–	
	A ₆₋₇	11-24	1066	49.125	1	4	2 ×16	D	7.8 × 10 ¹	5.2 × 10 ¹	1.0 × 10 ²	21.3	
	A ₈₋₁₂	11-26	1066	49.125	1	4	2 ×16	D	2.4 × 10 ²	5.4 × 10 ¹	4.4 × 10 ²	16.4	
	A ₁₃₋₁₄	11-50	1066	49.125	1	4	2 ×16	D	8.8 × 10 ¹	1.7 × 10 ¹	1.6 × 10 ²	26.2	
	A ₁₅₋₁₆	12-22	1600	50.625	1	4	2 ×16	D	9.5	9	1.0 × 10 ¹	34.4	
	A ₁₇₋₁₈	12-26	1600	49.125	2	8	2 ×8	M	1.2 × 10 ²	3.7 × 10 ¹	2.0 × 10 ²	21.3	
	A ₁₉₋₃₀	12-40	1600	48.125	2	8	2 ×8	K	8.6 × 10 ⁶	7.0 × 10 ⁶	1.0 × 10 ⁷	8.2	
	A ₃₁₋₃₄	13-02	1600	48.125	2	8	2 ×8	–	1.8 × 10 ⁶	1.0 × 10 ⁶	3.5 × 10 ⁶	11.5	
	A ₃₅₋₃₆	13-14	1600	48.125	2	8	2 ×8	–	4.0 × 10 ¹	1.9 × 10 ¹	6.1 × 10 ¹	21.3	
	A ₃₇₋₃₈	13-20	1600	48.125	2	8	2 ×8	K	1.7 × 10 ⁶	1.4 × 10 ⁶	2.0 × 10 ⁶	9.8	
	A ₃₉₋₄₀	13-28	1600	48.125	2	8	2 ×8	K	5.7 × 10 ⁴	5.4 × 10 ⁴	6.0 × 10 ⁴	16.4	
	A ₄₁	14-04	1600	49.125	2	8	2 ×8	–	2.7 × 10 ⁵	2.7 × 10 ⁵	2.7 × 10 ⁵	18.0	
	A ₄₂₋₄₃	14-04	1600	48.125	2	8	2 ×8	K	0.5	0	1	62.3	
B	B ₁	08-49	1066	50.625	1	8	1 ×8	D	0	0	0	–	
	B ₂	09-49	1066	50.625	1	8	1 ×8	E	0	0	0	–	
	B ₃	10-19	1066	50.625	1	8	1 ×8	F	0	0	0	–	
	B ₄	10-31	1333	49.125	2	8	2 ×8	C	0	0	0	–	
	B ₅	11-13	1333	49.125	2	8	2 ×8	C	0	0	0	–	
	B ₆	11-16	1066	50.625	1	8	1 ×8	F	0	0	0	–	
	B ₇	11-19	1066	50.625	1	8	1 ×8	F	0	0	0	–	
	B ₈	11-25	1333	49.125	2	8	2 ×8	C	0	0	0	–	
	B ₉	11-37	1333	49.125	2	8	2 ×8	D	1.9 × 10 ⁶	1.9 × 10 ⁶	1.9 × 10 ⁶	11.5	
	B ₁₀₋₁₂	11-46	1333	49.125	2	8	2 ×8	D	2.2 × 10 ⁶	1.5 × 10 ⁶	2.7 × 10 ⁶	11.5	
	B ₁₃	11-49	1333	49.125	2	8	2 ×8	C	0	0	0	–	
	B ₁₄	12-01	1866	47.125	2	8	2 ×8	D	9.1 × 10 ⁵	9.1 × 10 ⁵	9.1 × 10 ⁵	9.8	
	B ₁₅₋₃₁	12-10	1866	47.125	2	8	2 ×8	D	9.8 × 10 ⁵	7.8 × 10 ⁵	1.2 × 10 ⁶	11.5	
	B ₃₂	12-25	1600	48.125	2	8	2 ×8	E	7.4 × 10 ⁵	7.4 × 10 ⁵	7.4 × 10 ⁵	11.5	
	B ₃₃₋₄₂	12-28	1600	48.125	2	8	2 ×8	E	5.2 × 10 ⁵	1.9 × 10 ⁵	7.3 × 10 ⁵	11.5	
	B ₄₃₋₄₇	12-31	1600	48.125	2	8	2 ×8	E	4.0 × 10 ⁵	2.9 × 10 ⁵	5.5 × 10 ⁵	13.1	
C	B ₄₈₋₅₁	13-19	1600	48.125	2	8	2 ×8	E	1.1 × 10 ⁵	7.4 × 10 ⁴	1.4 × 10 ⁵	14.7	
	B ₅₂₋₅₃	13-40	1333	49.125	2	8	2 ×8	D	2.6 × 10 ⁴	2.3 × 10 ⁴	2.9 × 10 ⁴	21.3	
	B ₅₄	14-07	1333	49.125	2	8	2 ×8	D	7.5 × 10 ³	7.5 × 10 ³	7.5 × 10 ³	26.2	
	C ₁	10-18	1333	49.125	2	8	2 ×8	A	0	0	0	–	
	C ₂	10-20	1066	50.625	2	8	2 ×8	A	0	0	0	–	
	C ₃	10-22	1066	50.625	2	8	2 ×8	A	0	0	0	–	
	C ₄₋₅	10-26	1333	49.125	2	8	2 ×8	B	8.9 × 10 ²	6.0 × 10 ²	1.2 × 10 ³	29.5	
	C ₆	10-43	1333	49.125	1	8	1 ×8	T	0	0	0	–	
	C ₇	10-51	1333	49.125	2	8	2 ×8	B	4.0 × 10 ²	4.0 × 10 ²	4.0 × 10 ²	29.5	
	C ₈	11-12	1333	46.25	2	8	2 ×8	B	6.9 × 10 ²	6.9 × 10 ²	6.9 × 10 ²	21.3	
	C ₉	11-19	1333	46.25	2	8	2 ×8	B	9.2 × 10 ²	9.2 × 10 ²	9.2 × 10 ²	27.9	
	C ₁₀	11-31	1333	49.125	2	8	2 ×8	B	3	3	3	39.3	
	C ₁₁	11-42	1333	49.125	2	8	2 ×8	B	1.6 × 10 ²	1.6 × 10 ²	1.6 × 10 ²	39.3	
	C ₁₂	11-48	1600	48.125	2	8	2 ×8	C	7.1 × 10 ⁴	7.1 × 10 ⁴	7.1 × 10 ⁴	19.7	
	C ₁₃	12-08	1333	49.125	2	8	2 ×8	C	3.9 × 10 ⁴	3.9 × 10 ⁴	3.9 × 10 ⁴	21.3	
	C ₁₄₋₁₅	12-12	1333	49.125	2	8	2 ×8	C	3.7 × 10 ⁴	2.1 × 10 ⁴	5.4 × 10 ⁴	21.3	
	C ₁₆₋₁₈	12-20	1600	48.125	2	8	2 ×8	C	3.5 × 10 ³	1.2 × 10 ³	7.0 × 10 ³	27.9	
	C ₁₉	12-23	1600	48.125	2	8	2 ×8	E	1.4 × 10 ⁵	1.4 × 10 ⁵	1.4 × 10 ⁵	18.0	
	C ₂₀	12-24	1600	48.125	2	8	2 ×8	C	6.5 × 10 ⁴	6.5 × 10 ⁴	6.5 × 10 ⁴	21.3	
	C ₂₁	12-26	1600	48.125	2	8	2 ×8	C	2.3 × 10 ⁴	2.3 × 10 ⁴	2.3 × 10 ⁴	24.6	
	C ₂₂	12-32	1600	48.125	2	8	2 ×8	C	1.7 × 10 ⁴	1.7 × 10 ⁴	1.7 × 10 ⁴	22.9	
	C ₂₃₋₂₄	12-37	1600	48.125	2	8	2 ×8	C	2.3 × 10 ⁴	1.1 × 10 ⁴	3.4 × 10 ⁴	18.0	
	C ₂₅₋₃₀	12-41	1600	48.125	2	8	2 ×8	C	2.0 × 10 ⁴	1.1 × 10 ⁴	3.2 × 10 ⁴	19.7	
	C ₃₁	13-11	1600	48.125	2	8	2 ×8	C	3.3 × 10 ⁵	3.3 × 10 ⁵	3.3 × 10 ⁵	14.7	
	C ₃₂	13-35	1600	48.125	2	8	2 ×8	C	3.7 × 10 ⁴	3.7 × 10 ⁴	3.7 × 10 ⁴	21.3	

* We report the manufacture date marked on the chip packages, which is more accurate than other dates that can be gleaned from a module.

† We report timing constraints stored in the module's on-board ROM [33], which is read by the system BIOS to calibrate the memory controller.

‡ The maximum DRAM chip size supported by our testing platform is 2Gb.

§ We report DRAM die versions marked on the chip packages, which typically progress in the following manner: $\mathcal{M} \rightarrow \mathcal{A} \rightarrow \mathcal{B} \rightarrow \mathcal{C} \rightarrow \dots$.

Table 3. Sample population of 129 DDR3 DRAM modules, categorized by manufacturer and sorted by manufacture date

Fast Forward 6 Months

More Reviews... **Reviews from ISCA 2014**

PAPER WEAKNESSES

1) The disturbance error (a.k.a coupling or cross-talk noise induced error) is a known problem to the DRAM circuit community.

2) What you demonstrated in this paper is so called DRAM row hammering issue - you can even find a Youtube video showing this! - <http://www.youtube.com/watch?v=i3-gQSnBcdo>

2) The architectural contribution of this study is too insignificant.

PAPER WEAKNESSES

- Row Hammering appears to be well-known, and solutions have already been proposed by industry to address the issue.

- The paper only provides a qualitative analysis of solutions to the problem. A more robust evaluation is really needed to know whether the proposed solution is necessary.

Final RowHammer Reviews

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Accepted



639kB

21 Nov 2013 10:53:11pm CST |

f039be2735313b39304ae1c6296523867a485610

You are an **author** of this paper.

	OveMer	Nov	WriQua	RevConAnd
Review #41A	8	4	5	3
Review #41B	7	4	4	3
Review #41C	6	4	4	3
Review #41D	2	2	5	4
Review #41E	3	2	3	3
Review #41F	7	4	4	3

RowHammer: Hindsight & Impact (I)

Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors

Abstract. Memory isolation is a key property of a reliable and secure computing system — an access to one memory address should not have unintended side effects on data stored in other addresses. However, as DRAM process technology

Project Zero

[Flipping Bits in Memory Without Accessing Them:
An Experimental Study of DRAM Disturbance Errors](#)
(Kim et al., ISCA 2014)

News and updates from the Project Zero team at Google

[Exploiting the DRAM rowhammer bug to
gain kernel privileges](#) (Seaborn, 2015)

Monday, March 9, 2015

Exploiting the DRAM rowhammer bug to gain kernel privileges

RowHammer: Hindsight & Impact (II)

- Onur Mutlu and Jeremie Kim,
"RowHammer: A Retrospective"
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD) Special Issue on Top Picks in Hardware and Embedded Security, 2019.
[[Preliminary arXiv version](#)]

RowHammer: A Retrospective

Onur Mutlu^{§‡} Jeremie S. Kim^{‡§}
§ETH Zürich ‡Carnegie Mellon University

Suggestion to Researchers: Principle: Passion

Follow Your Passion
(Do not get derailed
by naysayers)

Suggestion to Researchers: Principle: Resilience

Be Resilient

Principle: Learning and Scholarship

Focus on
learning and scholarship

Principle: Learning and Scholarship

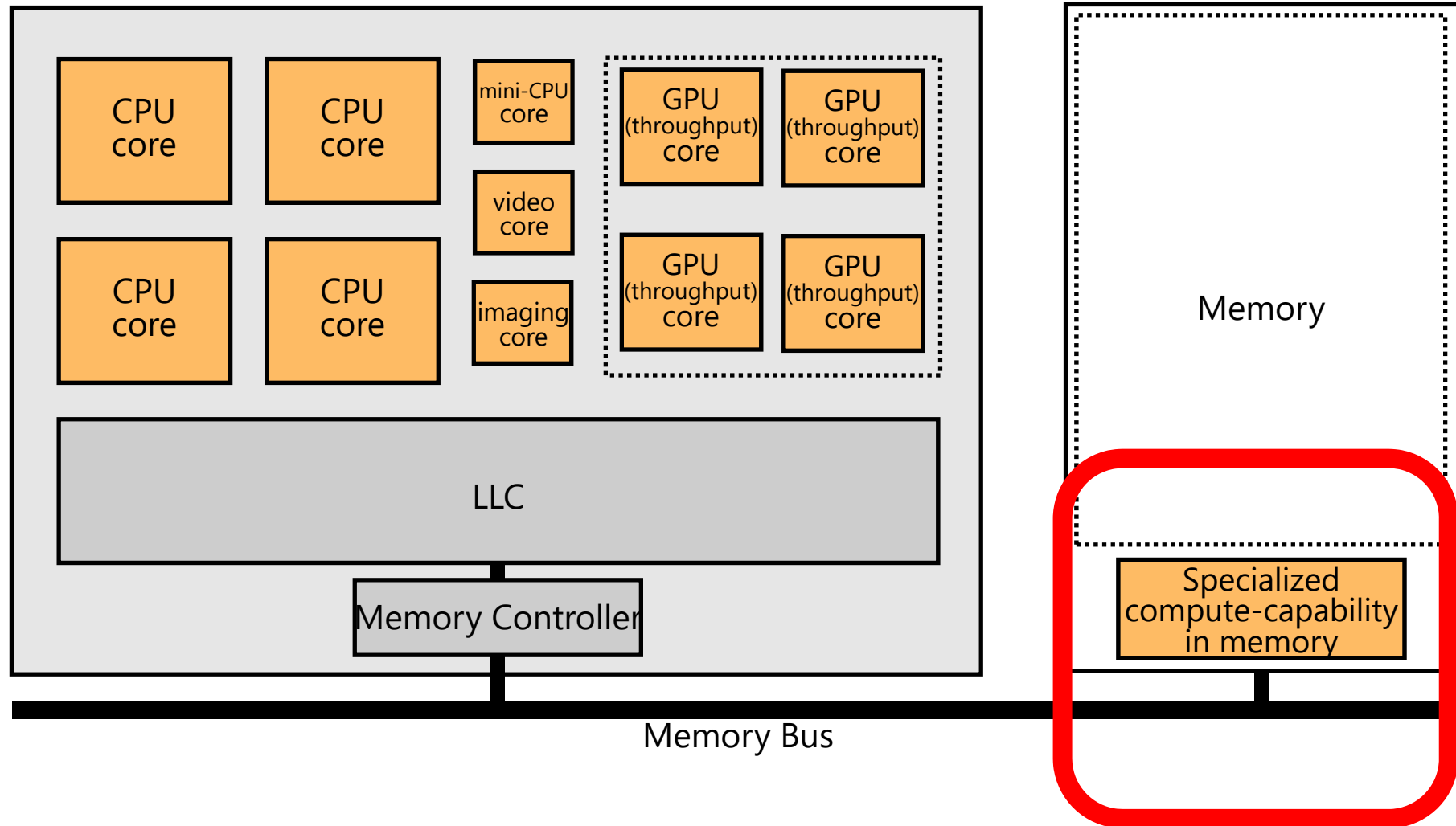
The quality of your work
defines your impact

Sub-Agenda: In-Memory Computation

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

We Need to Think Differently
from the Past Approaches

Memory as an Accelerator



Memory similar to a "conventional" accelerator

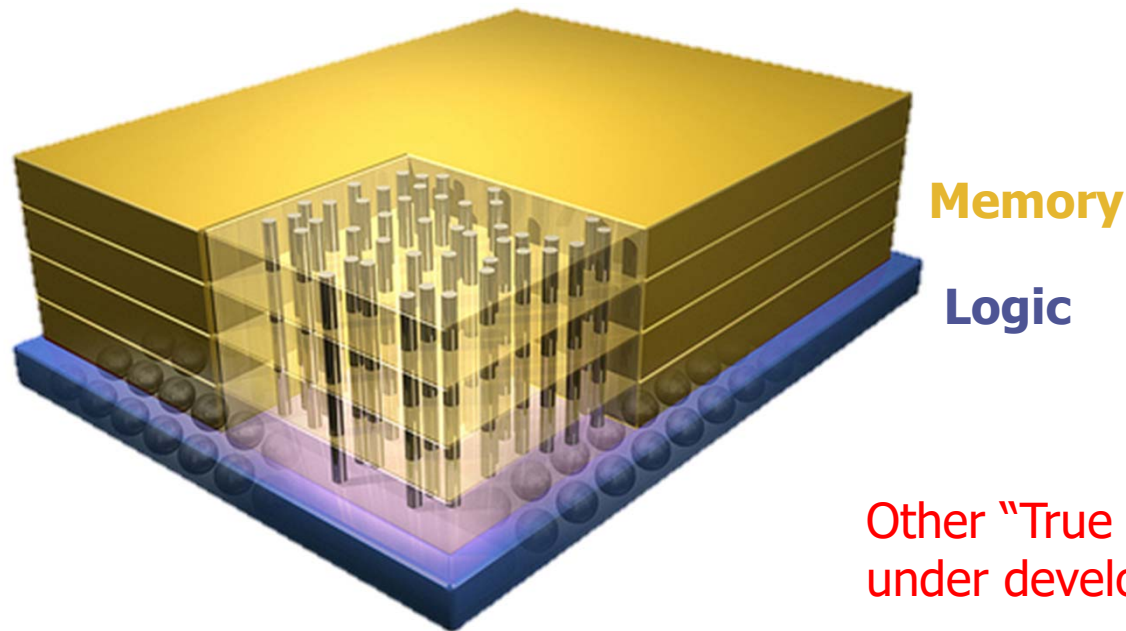
Processing in Memory: Two Approaches

1. Minimally changing memory chips
2. Exploiting 3D-stacked memory

Opportunity: 3D-Stacked Logic+Memory



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



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

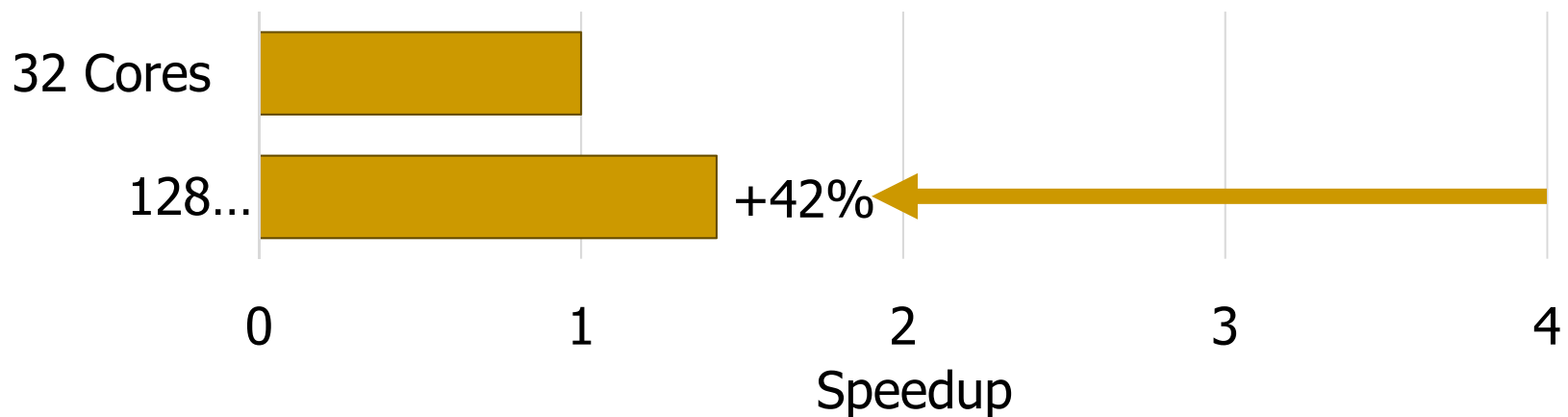


300 Million
Twitter Users



30 Billion
Instagram Photos

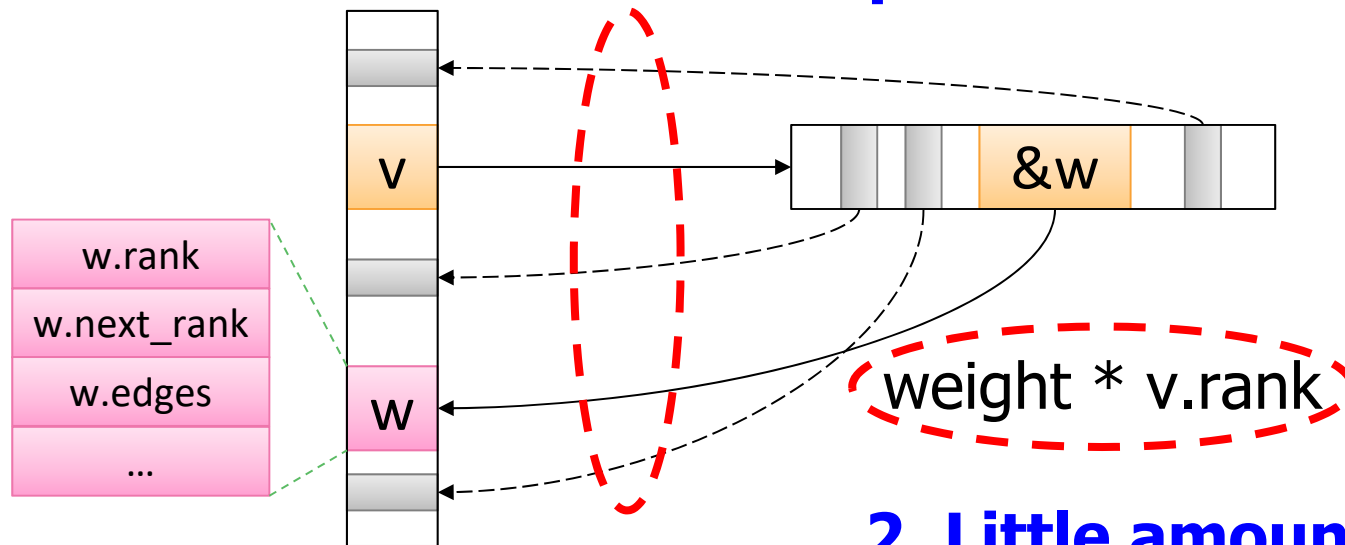
- Scalable large-scale graph processing is challenging



Key Bottlenecks in Graph Processing

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

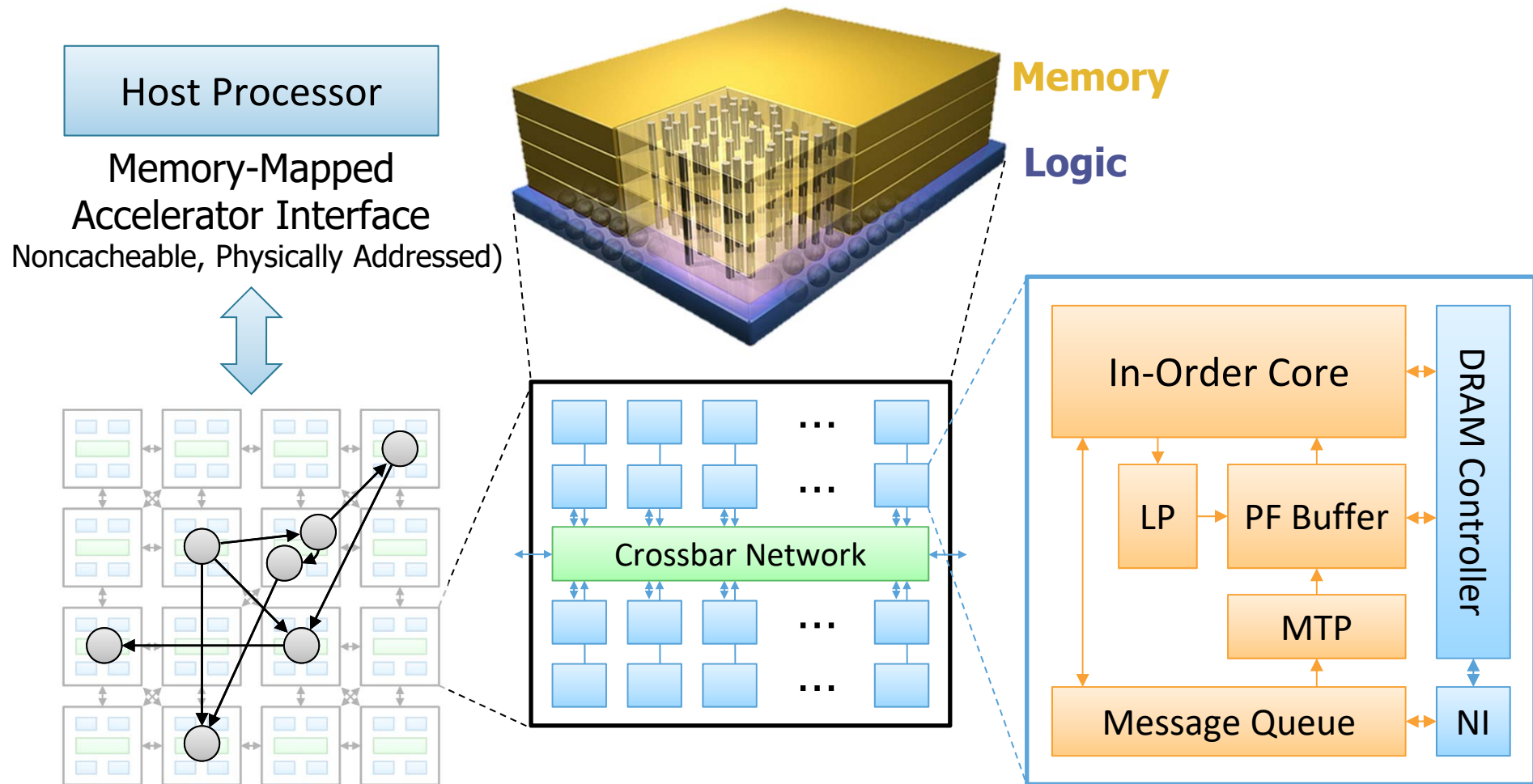
1. Frequent random memory accesses



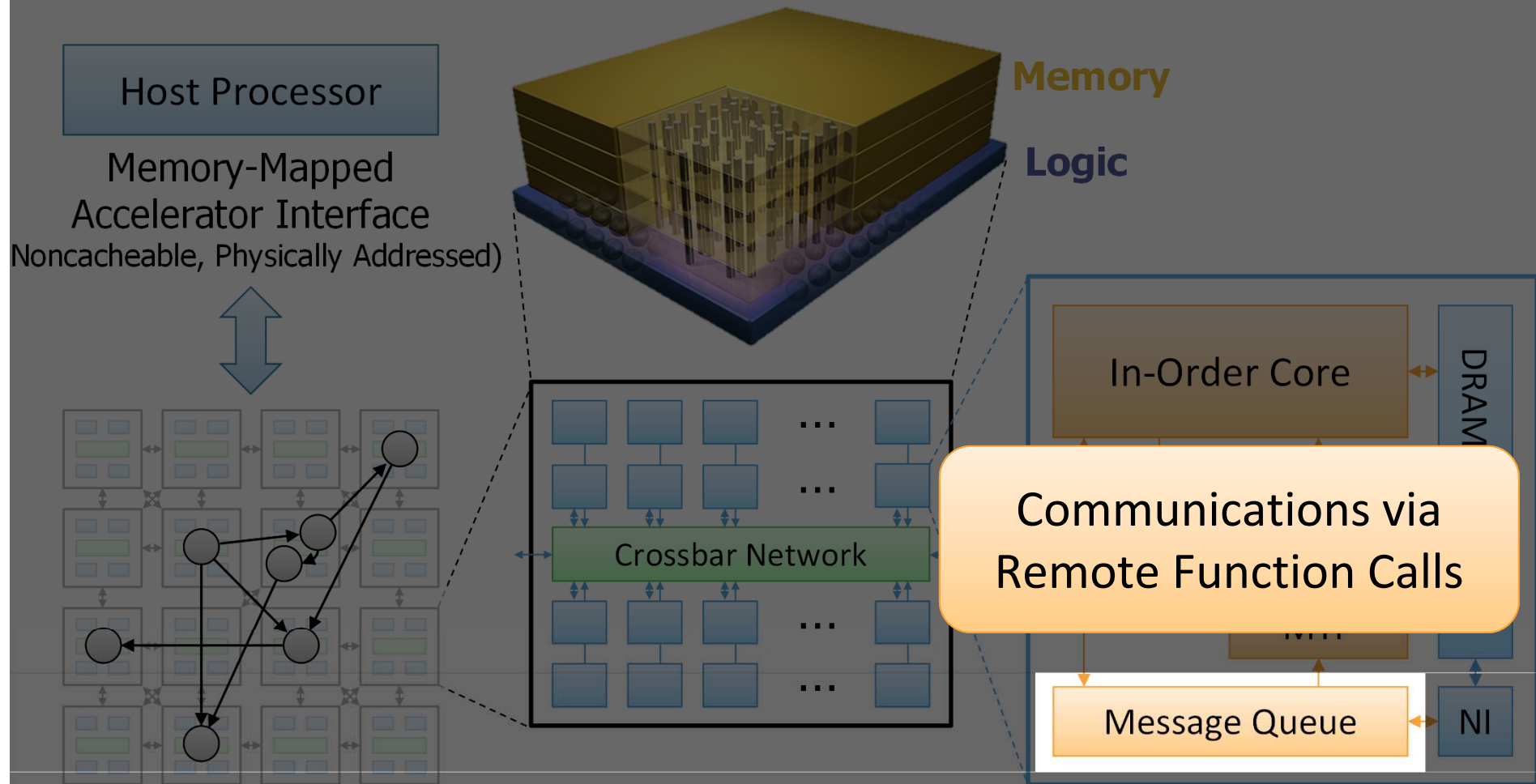
2. Little amount of computation

Tesseract System for Graph Processing

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

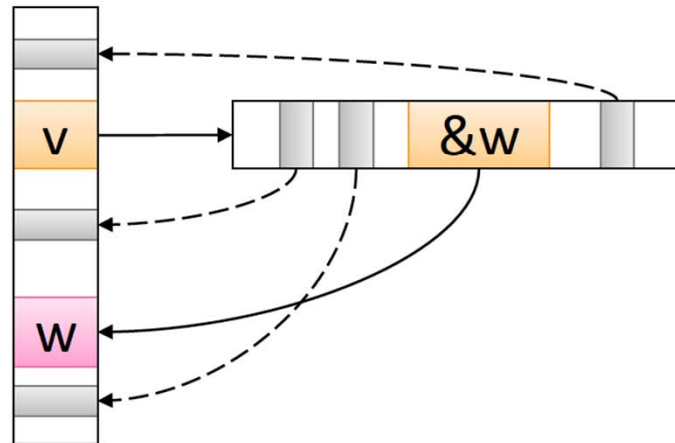


Tesseract System for Graph Processing



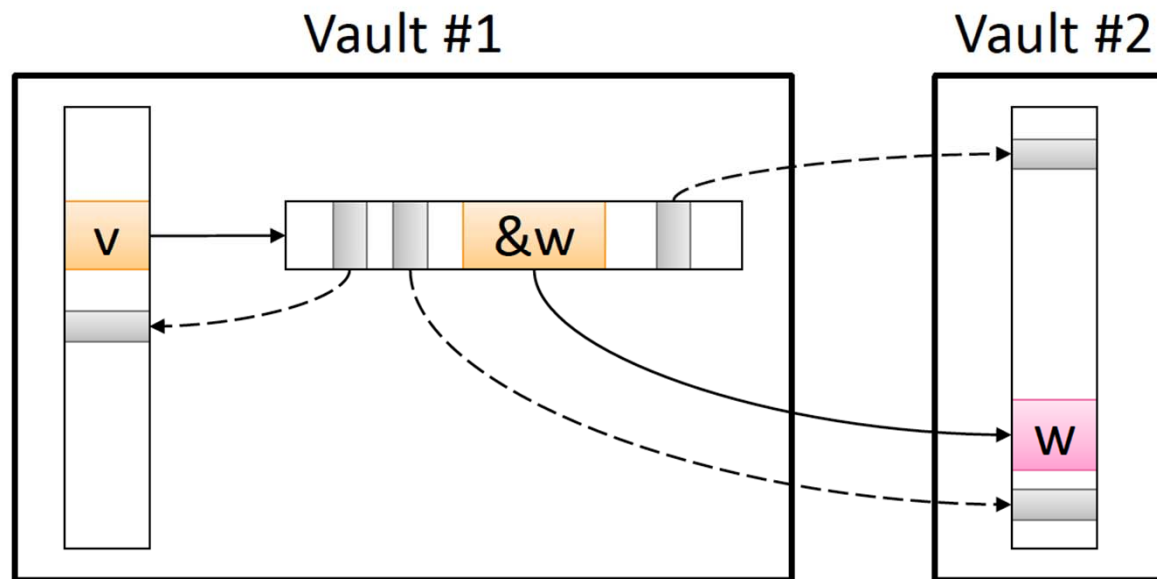
Communications In Tesseract (I)

```
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;  
  }  
}
```

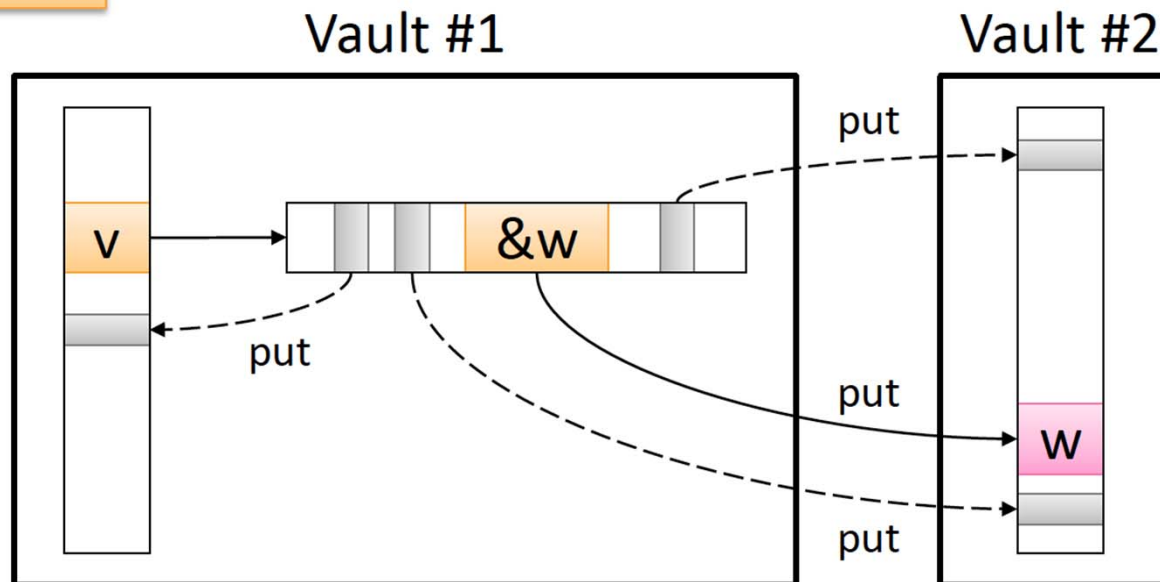


Communications In Tesseract (III)

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

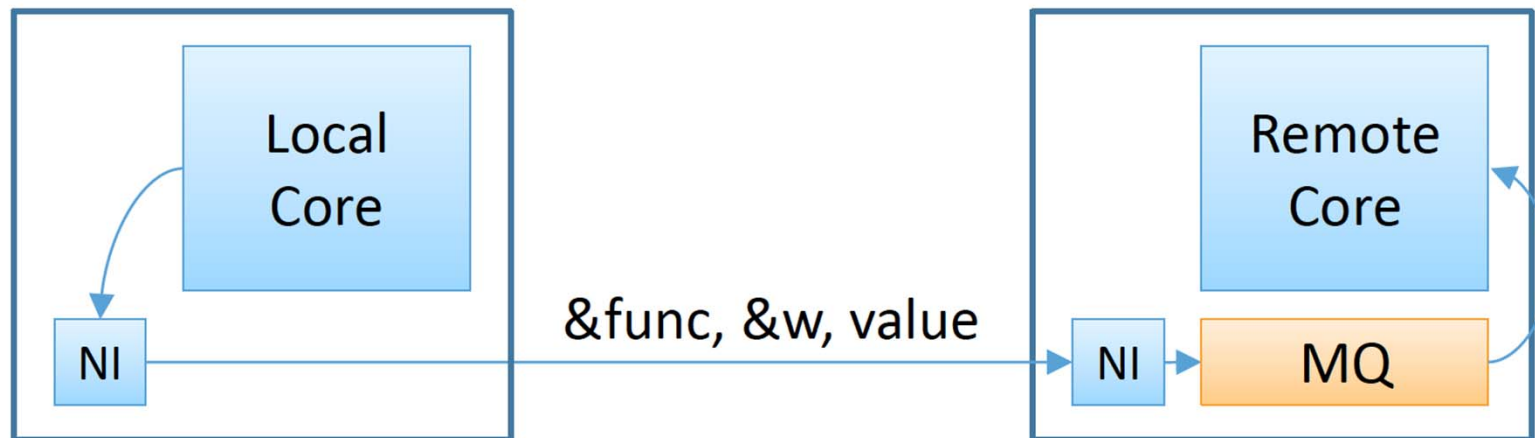
Non-blocking Remote Function Call

Can be **delayed** until the nearest barrier



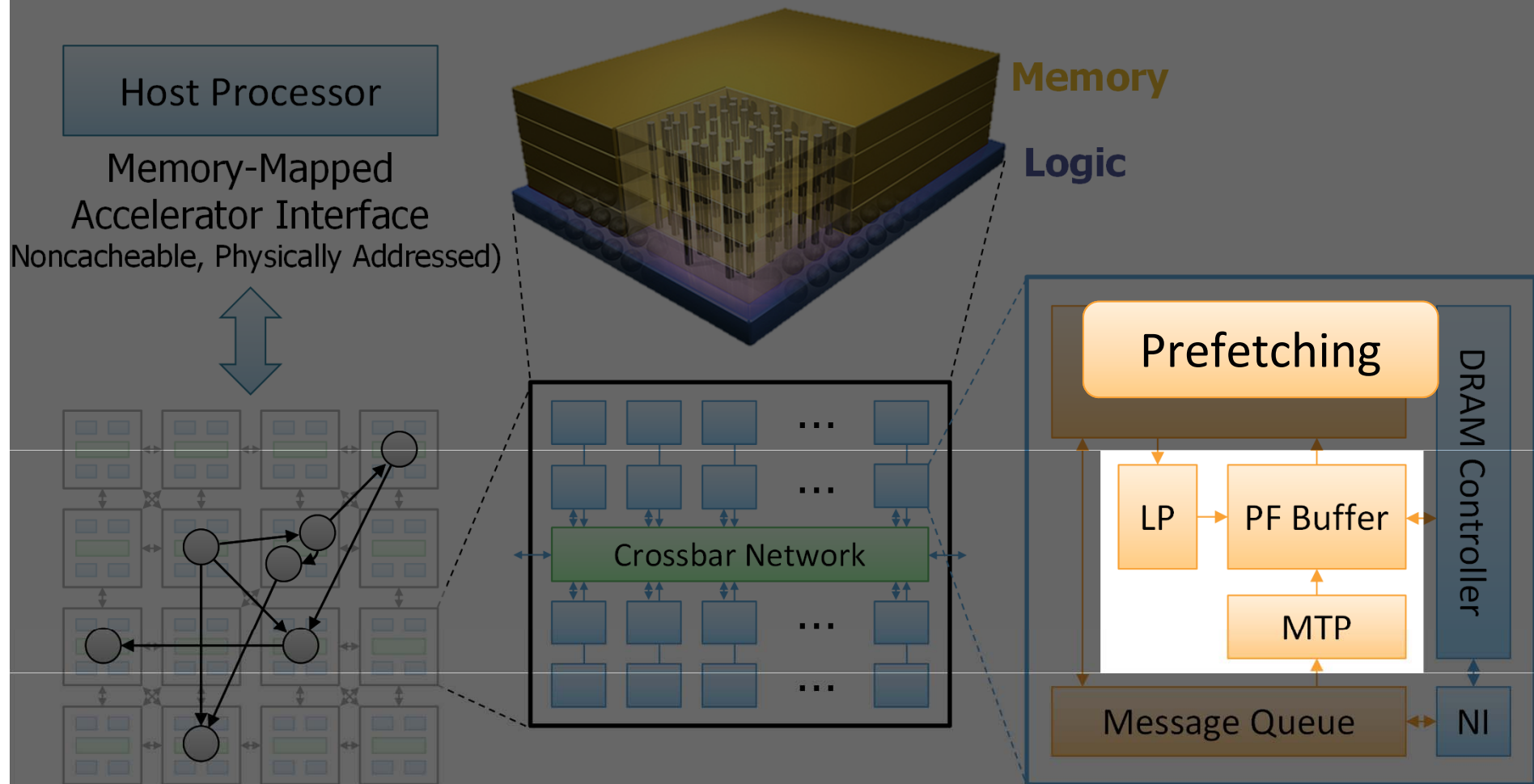
Remote Function Call (Non-Blocking)

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



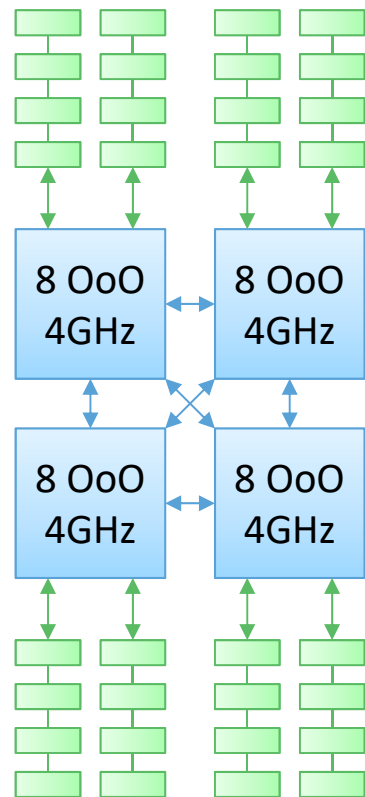
```
put(w.id, function() { w.next_rank += value; })
```

Tesseract System for Graph Processing



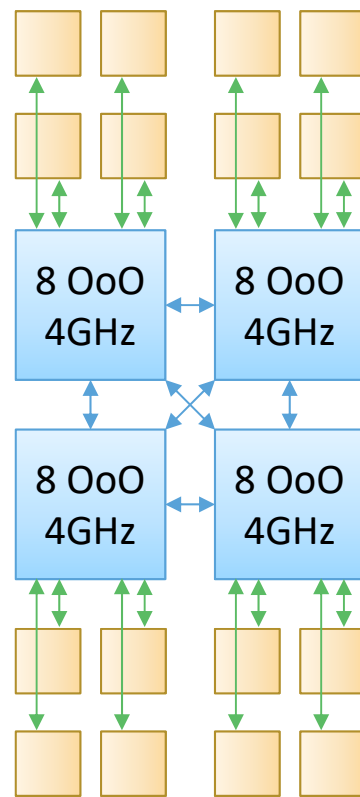
Evaluated Systems

DDR3-OoO



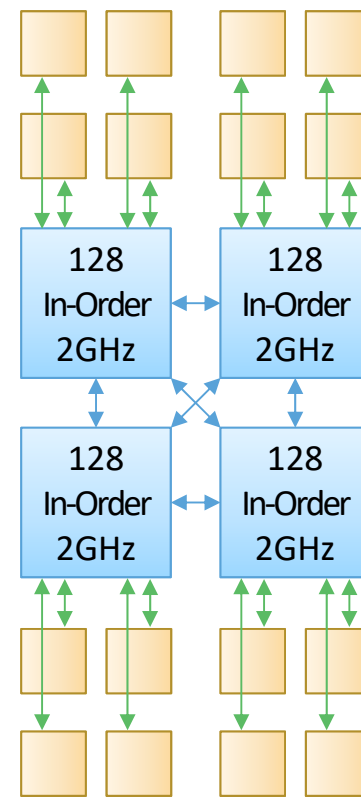
102.4GB/s

HMC-OoO



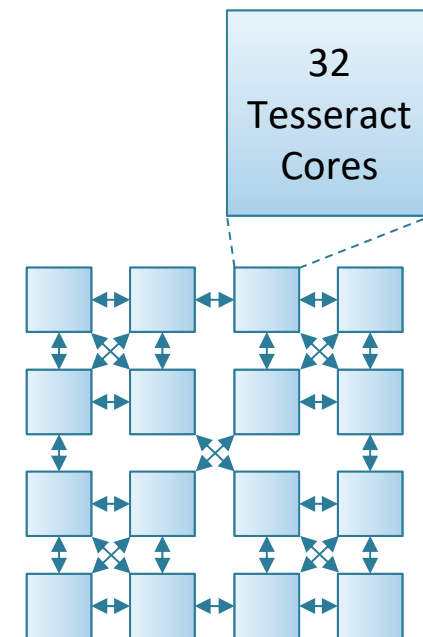
640GB/s

HMC-MC



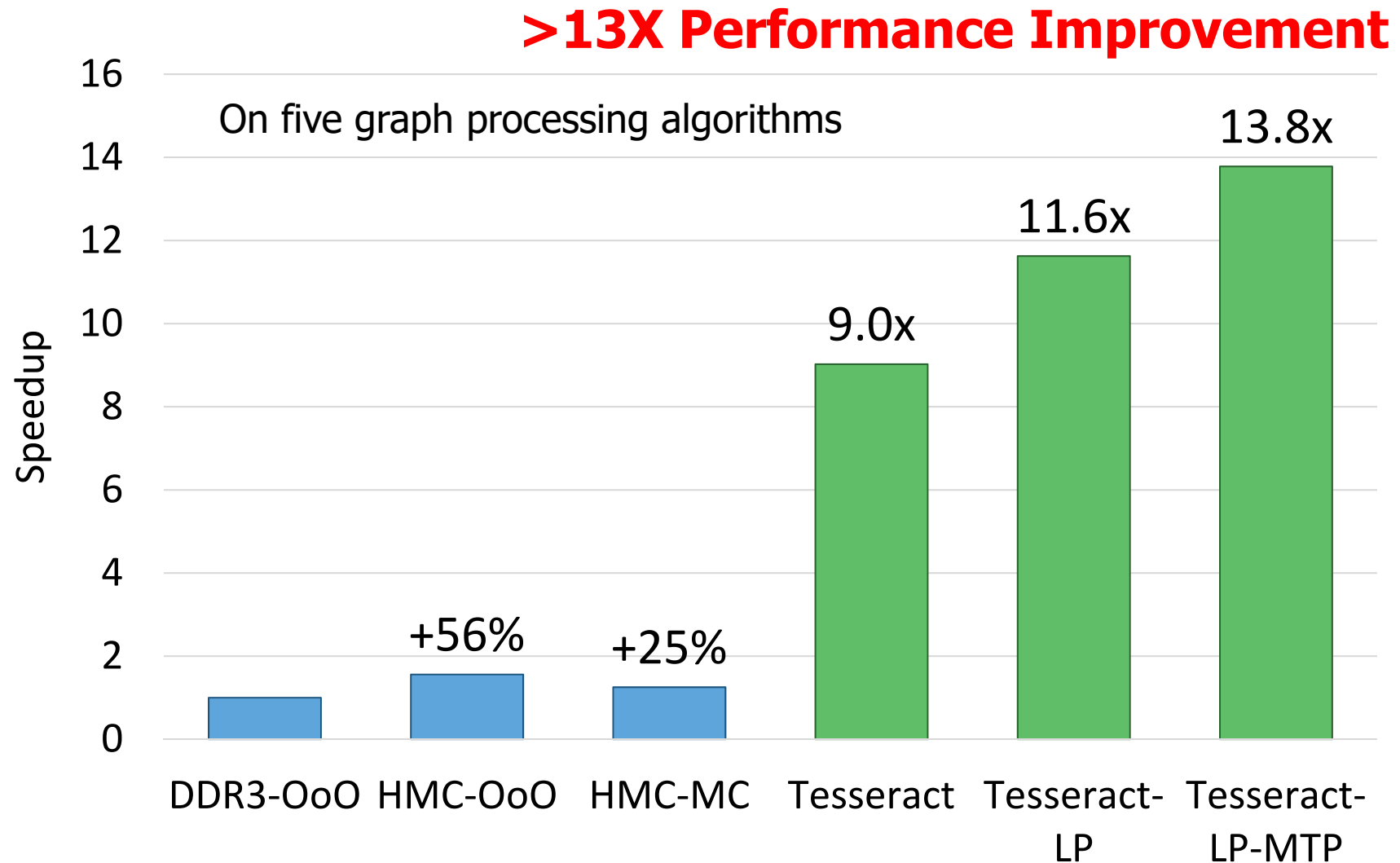
640GB/s

Tesseract

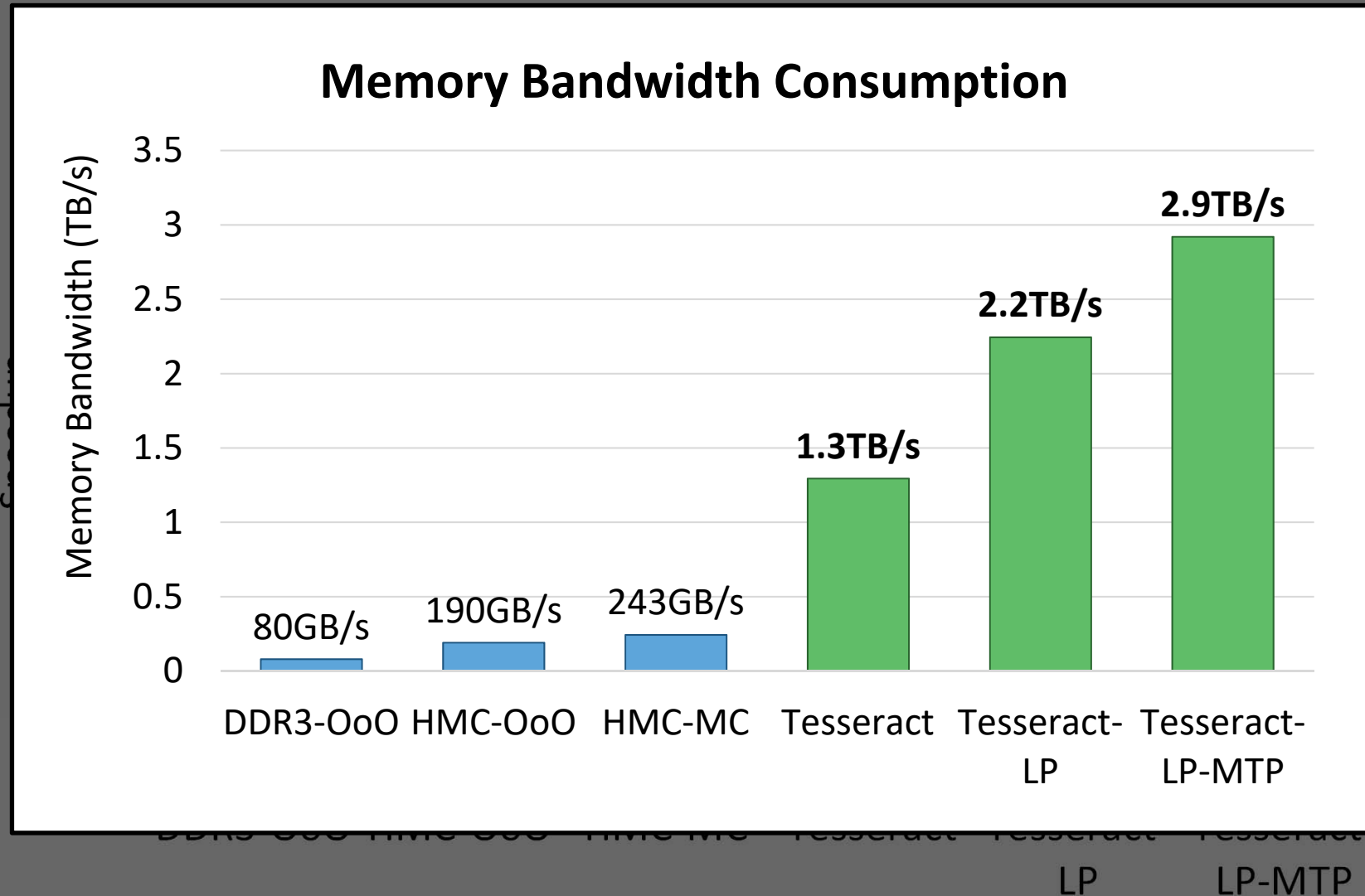


8TB/s

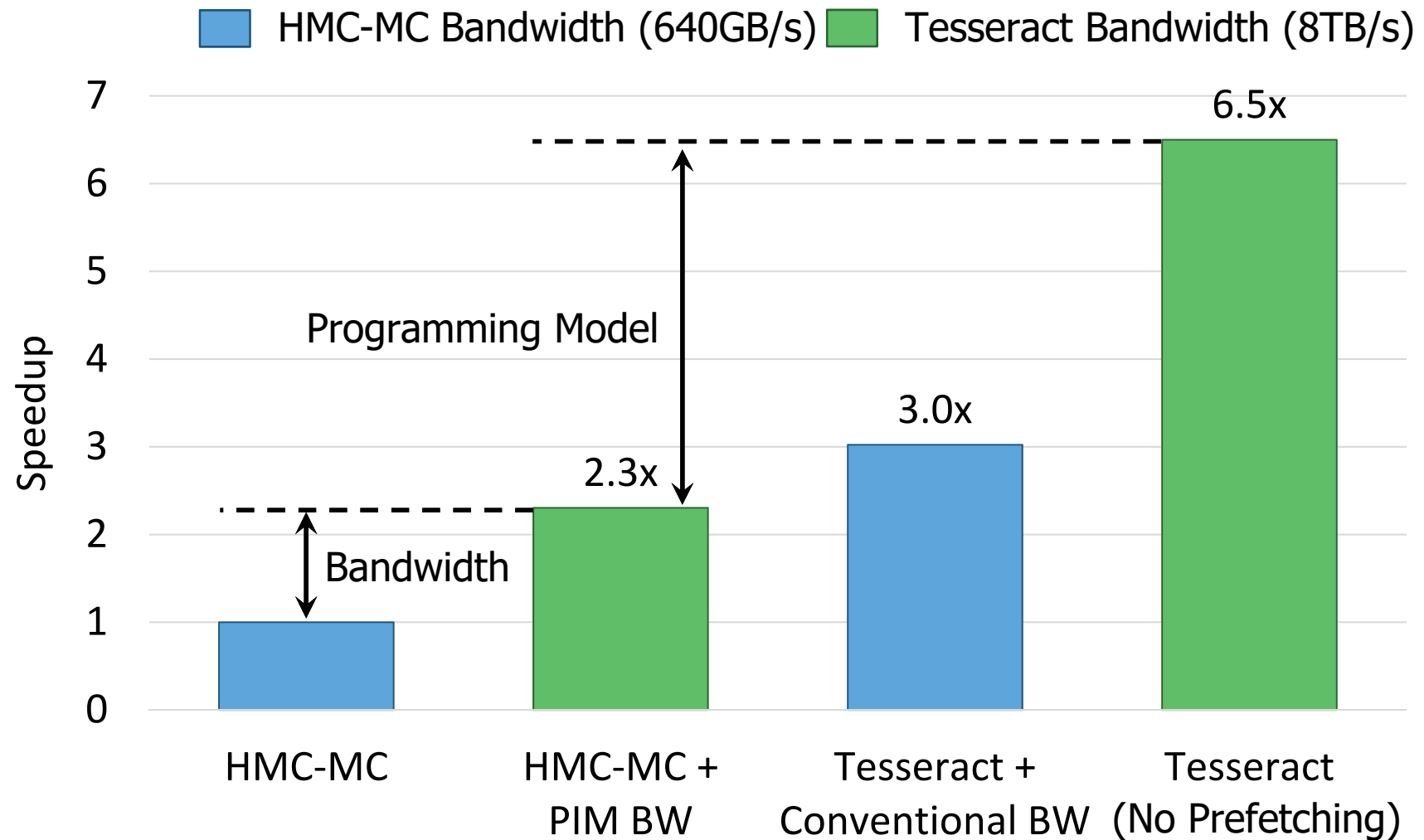
Tesseract Graph Processing Performance



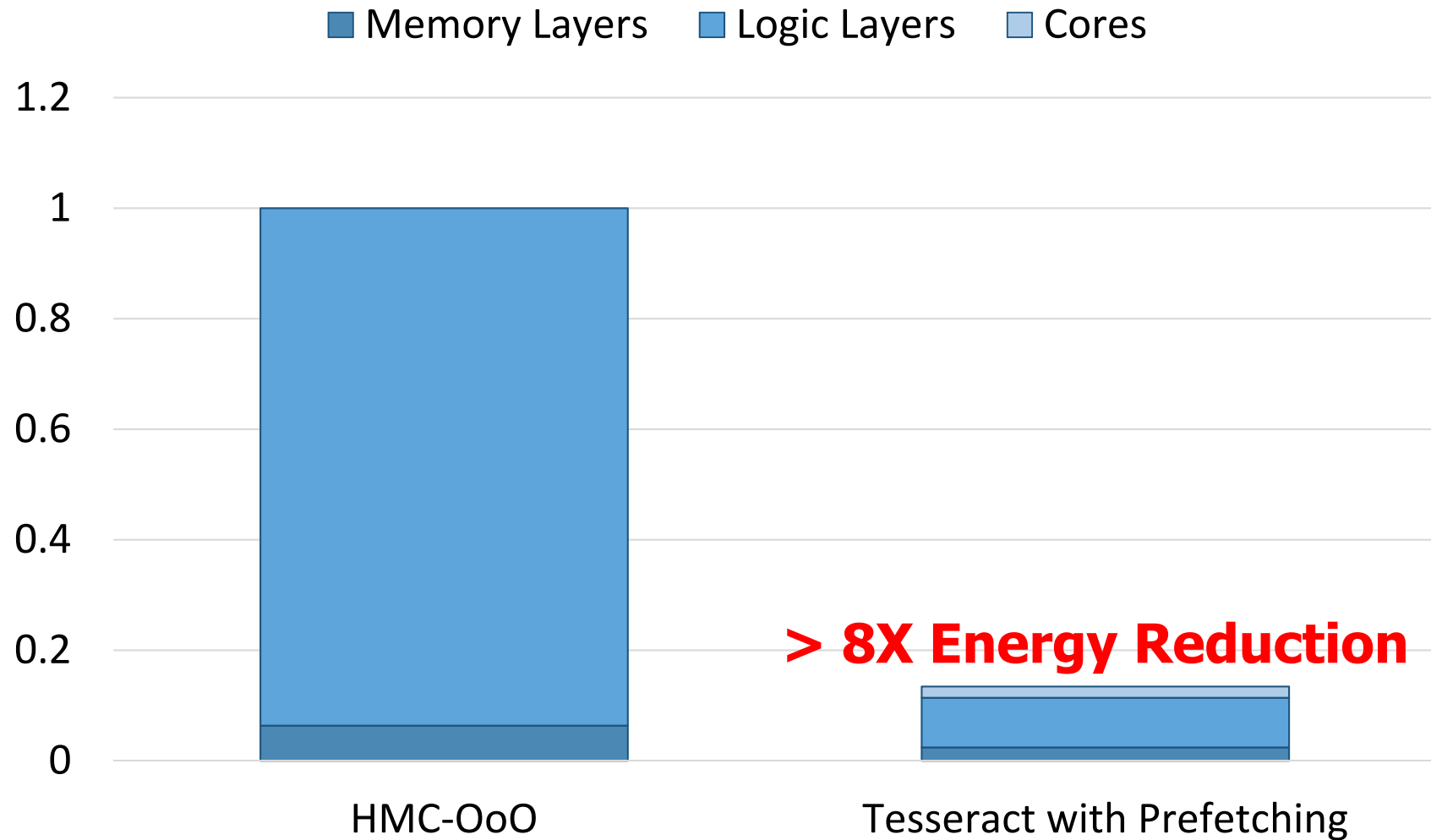
Tesseract Graph Processing Performance



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 \(pdf\)](#)] [[Lightning Session Slides \(pdf\)](#)]

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

Several Questions in 3D-Stacked PIM

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

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

3D-Stacked PIM on Mobile Devices

- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu, **"Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"**

*Proceedings of the 23rd International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**), Williamsburg, VA, USA, March 2018.*

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

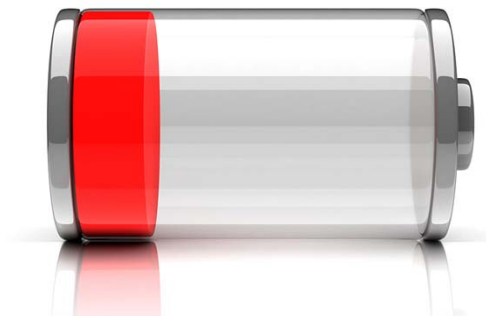
Amirali Boroumand ¹	Saugata Ghose ¹	Youngsok Kim ²	
Rachata Ausavarungnirun ¹	Eric Shiu ³	Rahul Thakur ³	Daehyun Kim ^{4,3}
Aki Kuusela ³	Allan Knies ³	Parthasarathy Ranganathan ³	Onur Mutlu ^{5,1}

Consumer Devices



Consumer devices are everywhere!

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



SAFARI

Four Important Workloads



Chrome

Google's web browser



TensorFlow Mobile

Google's machine learning framework



Video Playback

Google's **video codec**



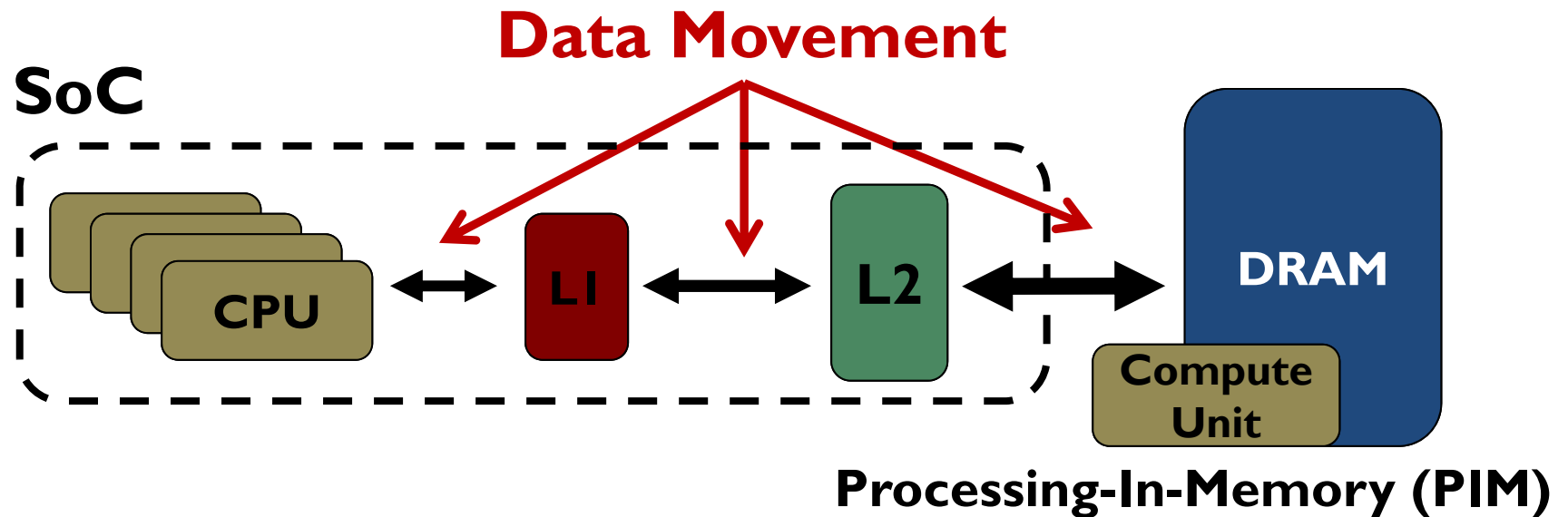
Video Capture

Google's **video codec**

SAFARI

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**

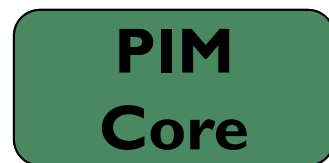
Challenge: limited area and energy budget

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



Video Playback

Google's **video codec**

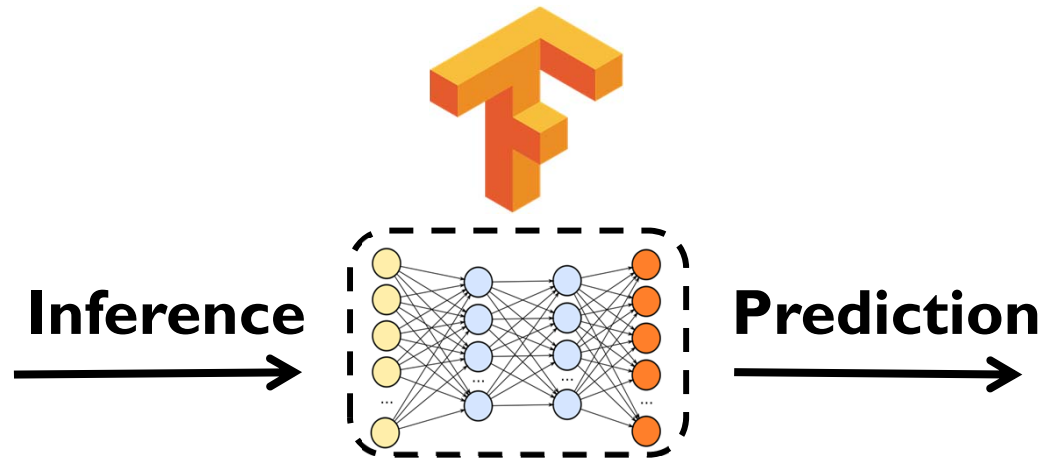


Video Capture

Google's **video codec**

SAFARI

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



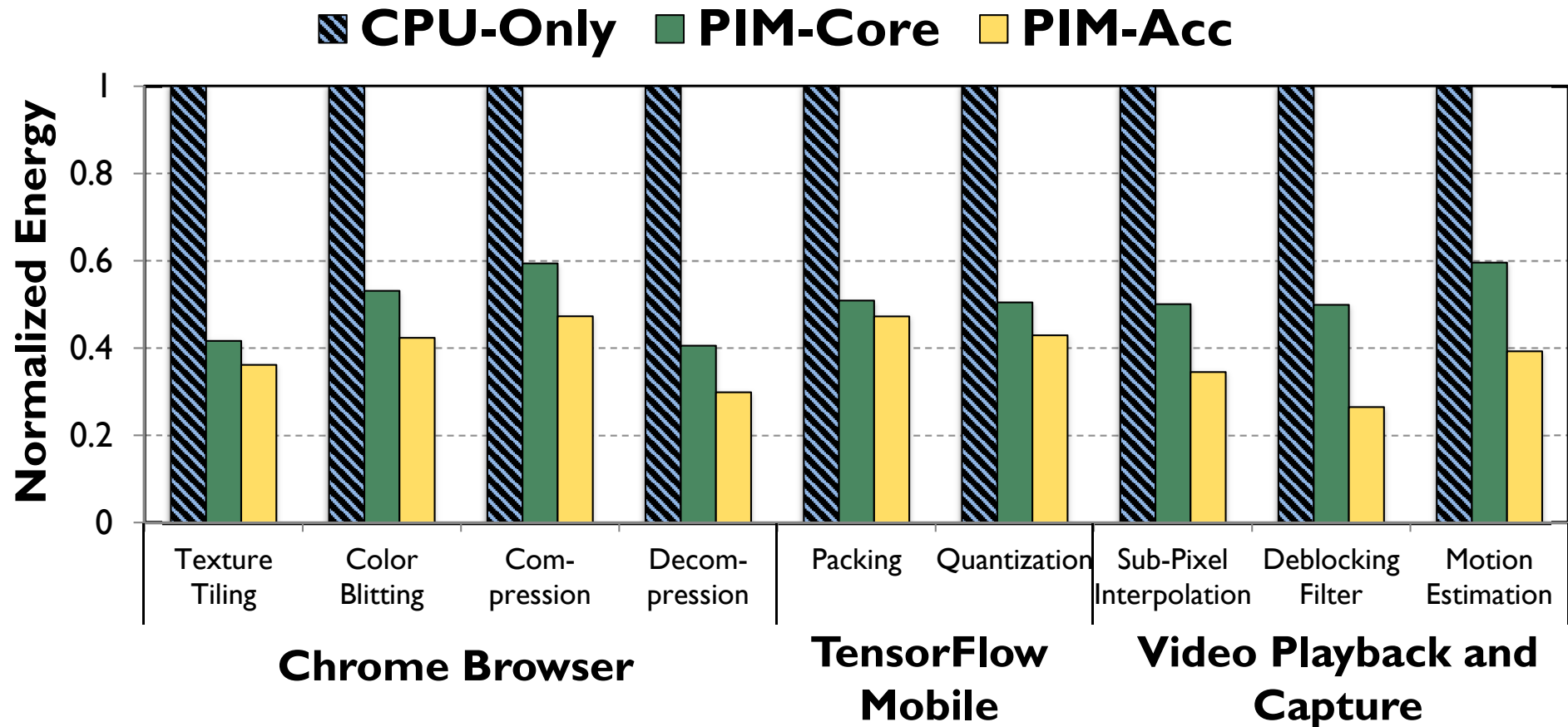
Up to **16.8%** of the inference **energy** and **16.1%** of inference **execution time**



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

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

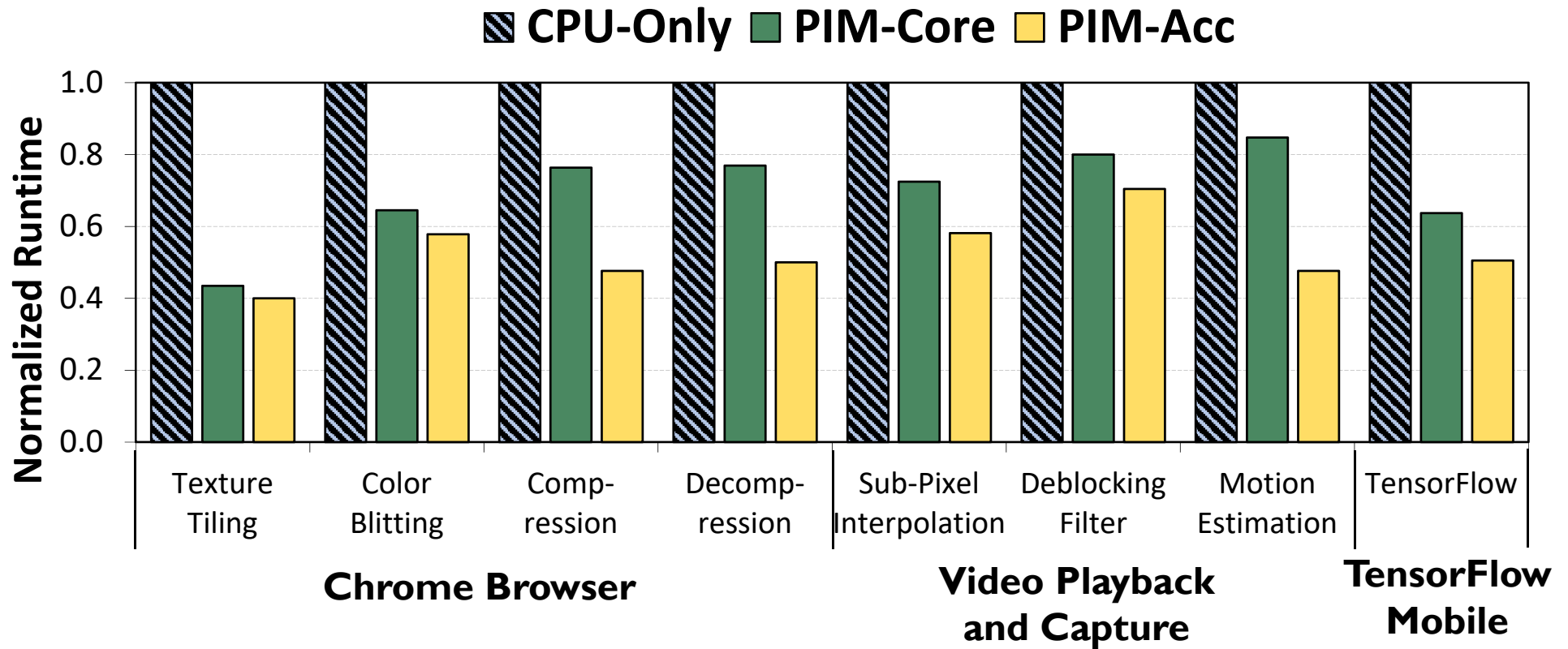
Normalized Energy



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

SAFARI

Normalized Runtime



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

Workload Analysis



Chrome

Google's web browser



TensorFlow

Google's machine learning
framework



Video Playback

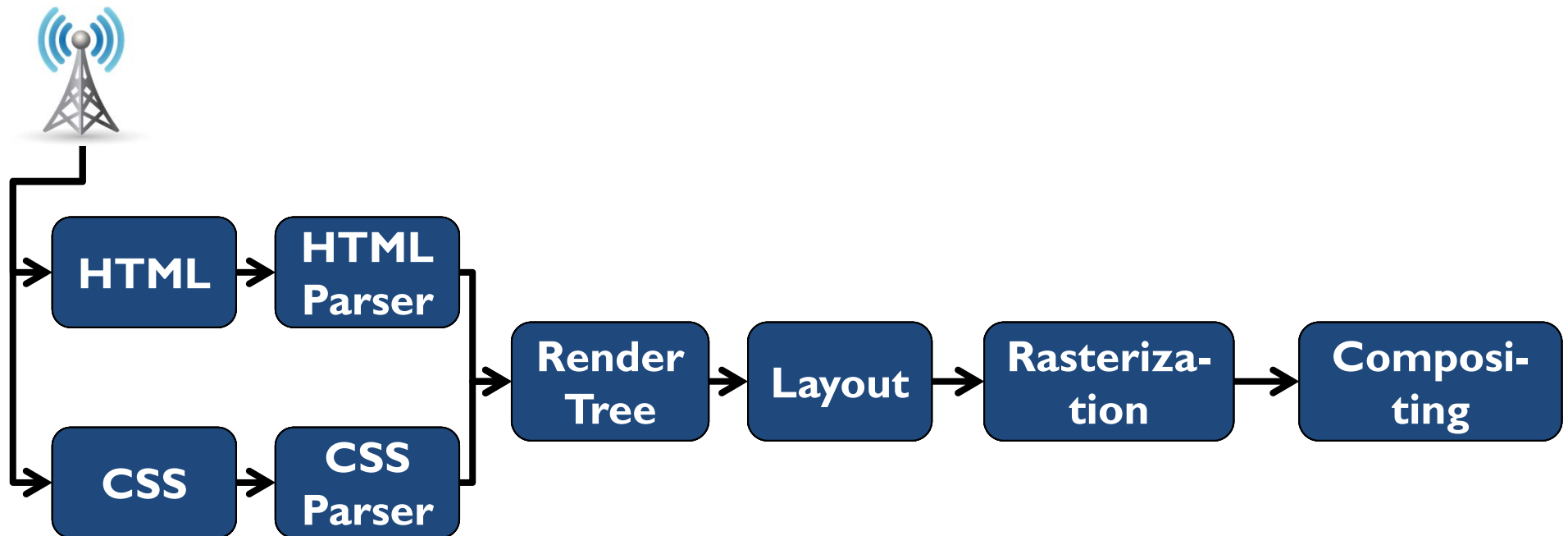
Google's video codec



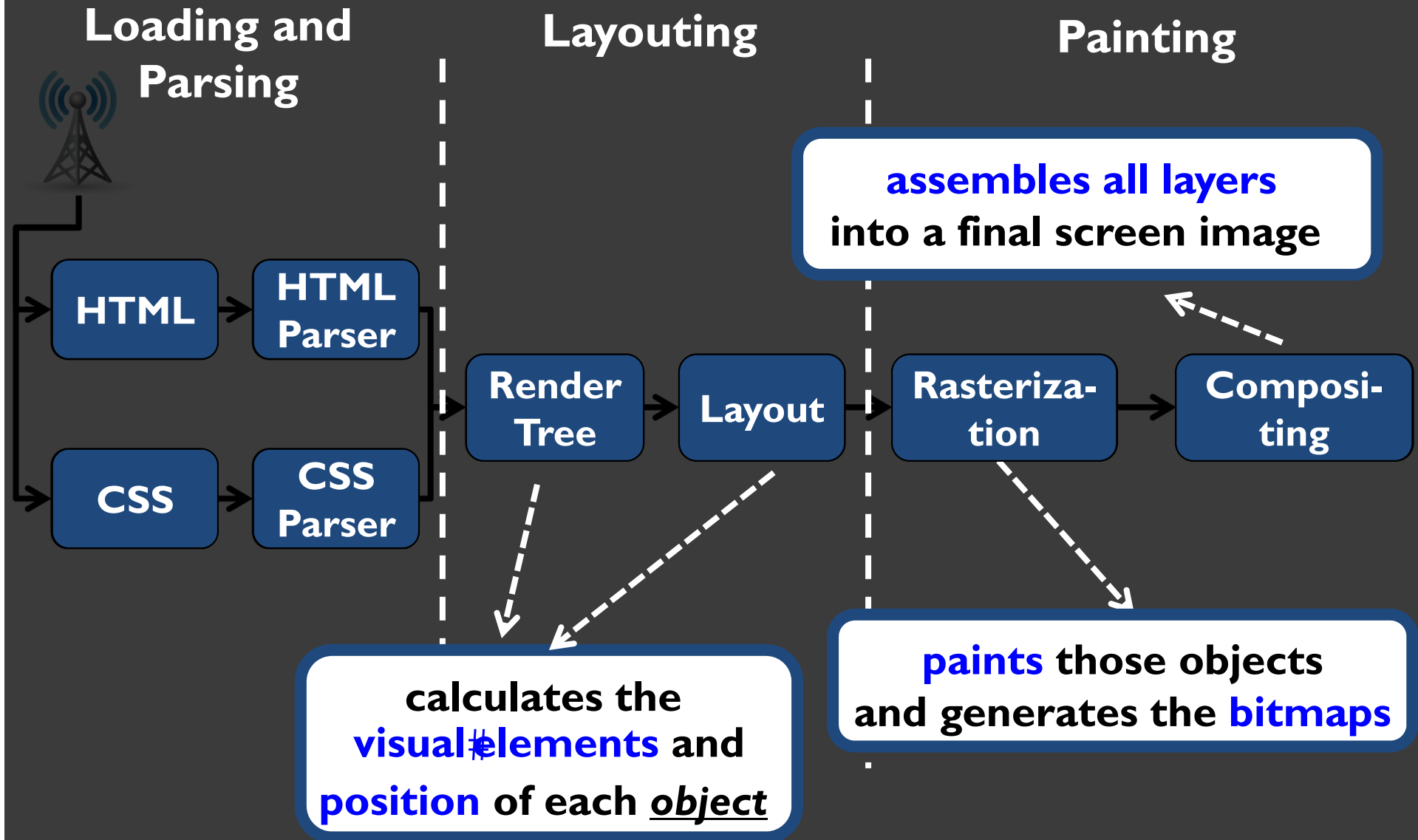
Video Capture

Google's video codec

How Chrome Renders a Web Page



How Chrome Renders a Web Page



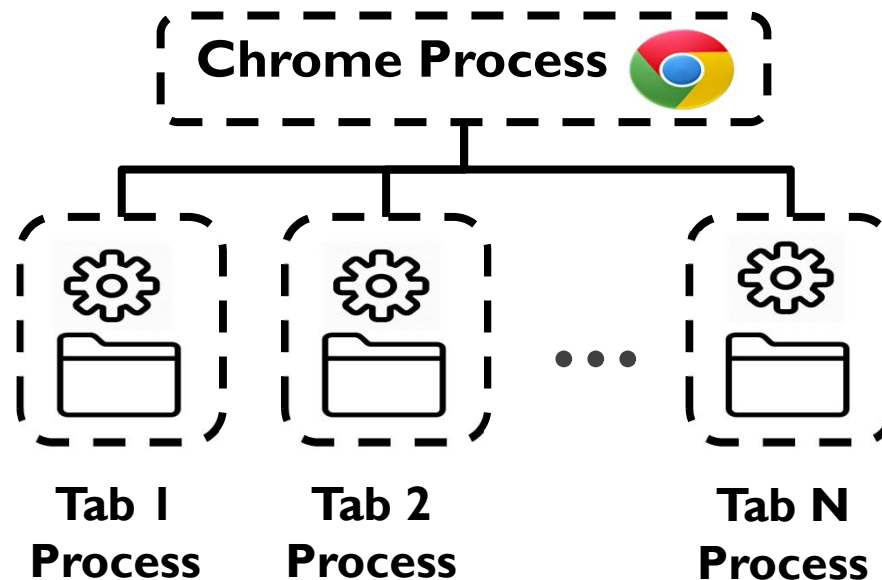
Browser Analysis

- 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

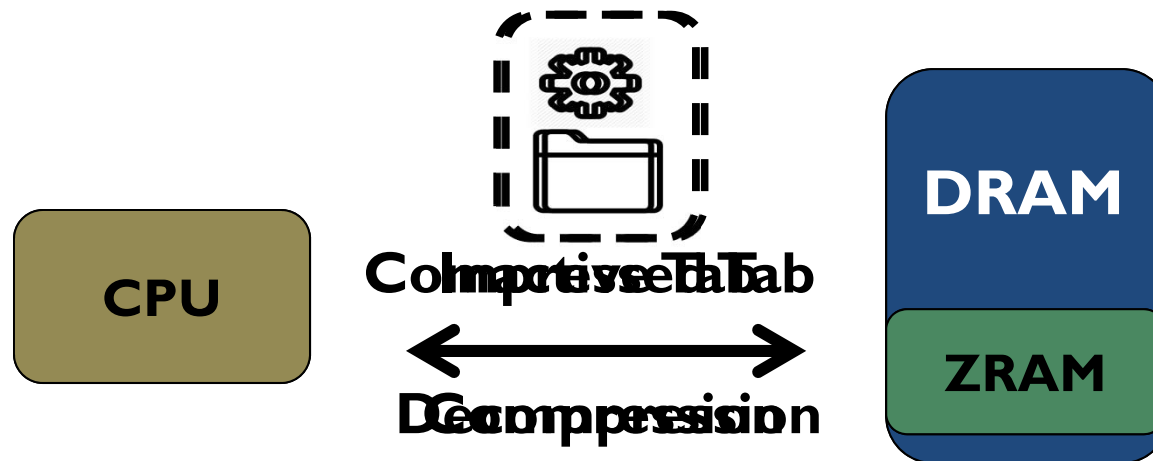


- Main operations during **tab switching**:
 - Context switch
 - Load the new page

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

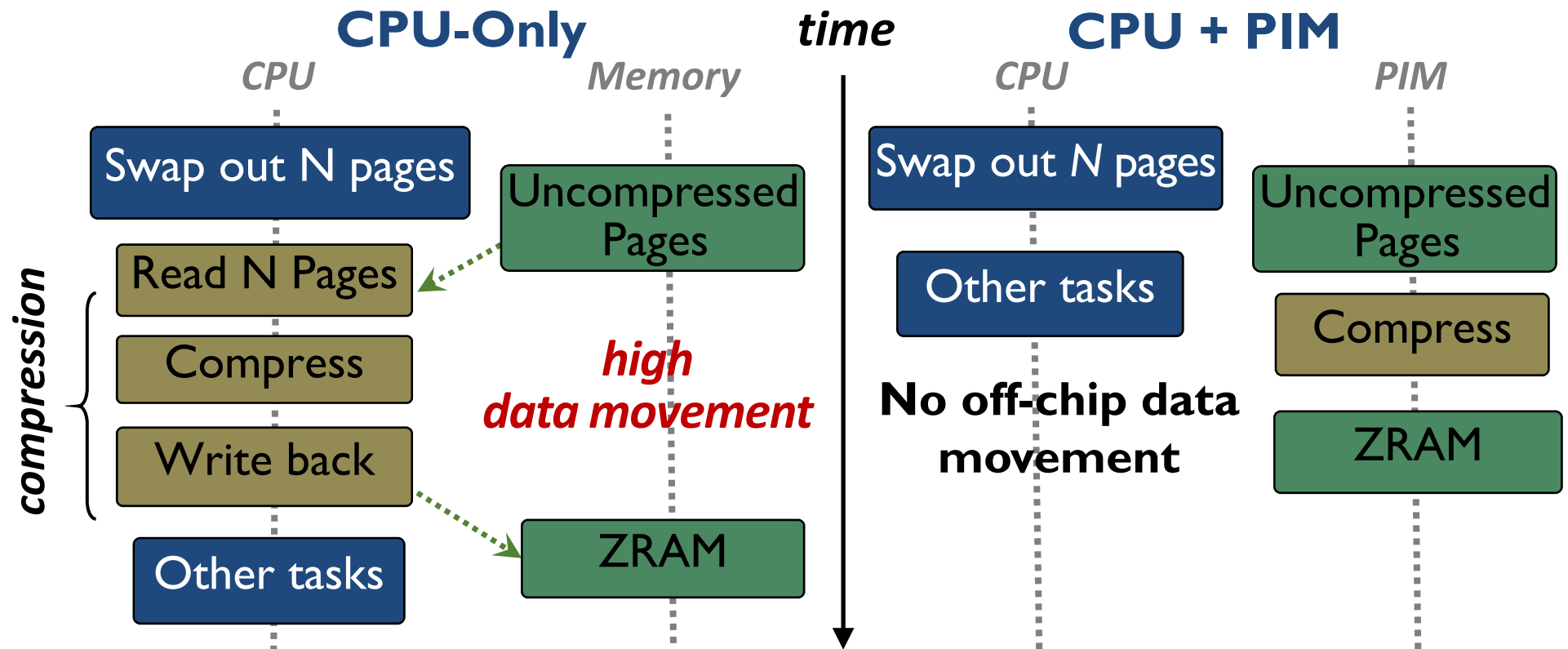
- To study **data movement** during tab switching, we emulate a user switching through 50 tabs

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

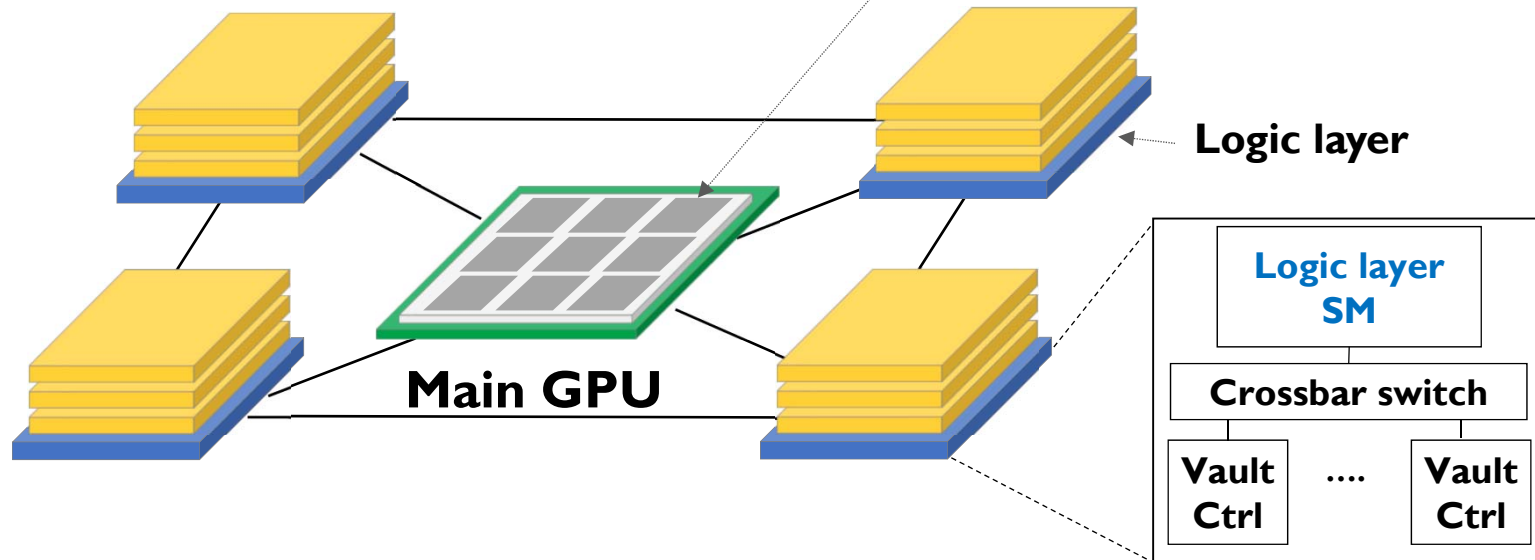
Amirali Boroumand¹ Saugata Ghose¹ Youngsok Kim²
Rachata Ausavarungnirun¹ Eric Shiu³ Rahul Thakur³ Daehyun Kim^{4,3}
Aki Kuusela³ Allan Knies³ Parthasarathy Ranganathan³ Onur Mutlu^{5,1}

SAFARI

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*

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

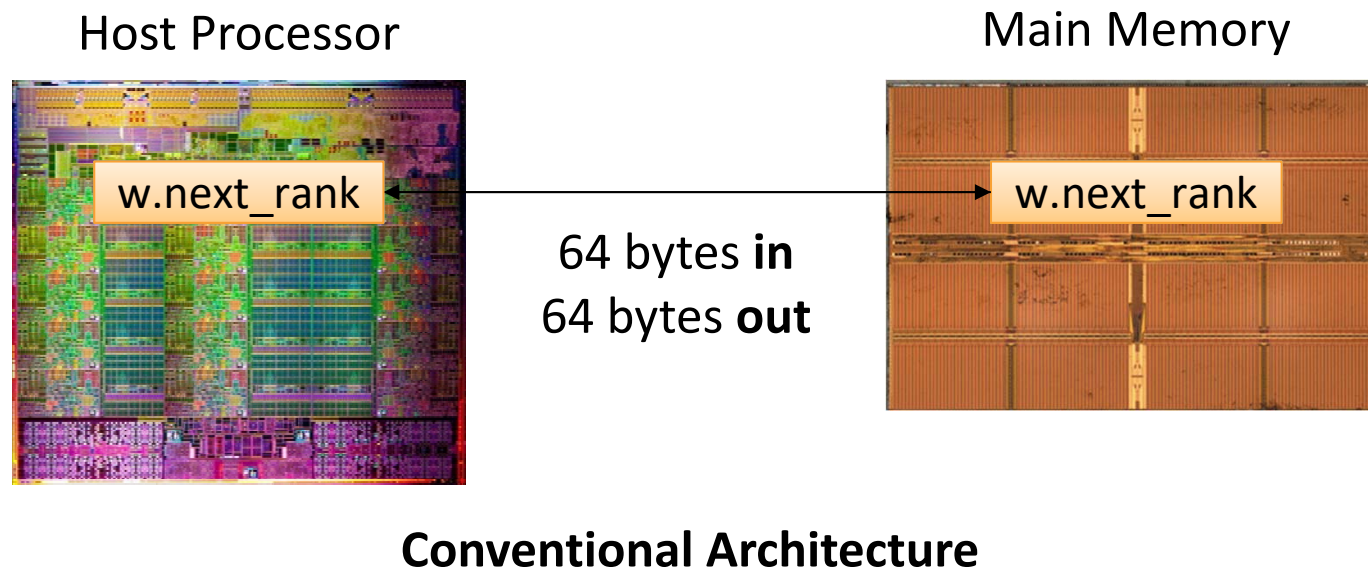
SAFARI

PEI: PIM-Enabled Instructions (Ideas)

- **Goal:** Develop mechanisms to get the most out of near-data processing with minimal cost, minimal changes to the system, no changes to the programming model
- **Key Idea 1:** Expose each PIM operation as a **cache-coherent, virtually-addressed host processor instruction** (called PEI) that operates on **only a single cache block**
 - e.g., `__pim_add(&w.next_rank, value) → pim.add r1, (r2)`
 - No changes sequential execution/programming model
 - No changes to virtual memory
 - Minimal changes to cache coherence
 - No need for data mapping: Each PEI restricted to a single memory module
- **Key Idea 2:** **Dynamically decide where to execute a PEI** (i.e., the host processor or PIM accelerator) based on simple locality characteristics and simple hardware predictors
 - Execute each operation at the location that provides the best performance

Simple PIM Operations as ISA Extensions (II)

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

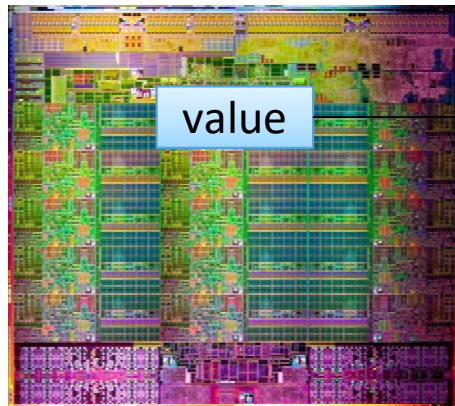


Simple PIM Operations as ISA Extensions (III)

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

pim.add r1, (r2)

Host Processor



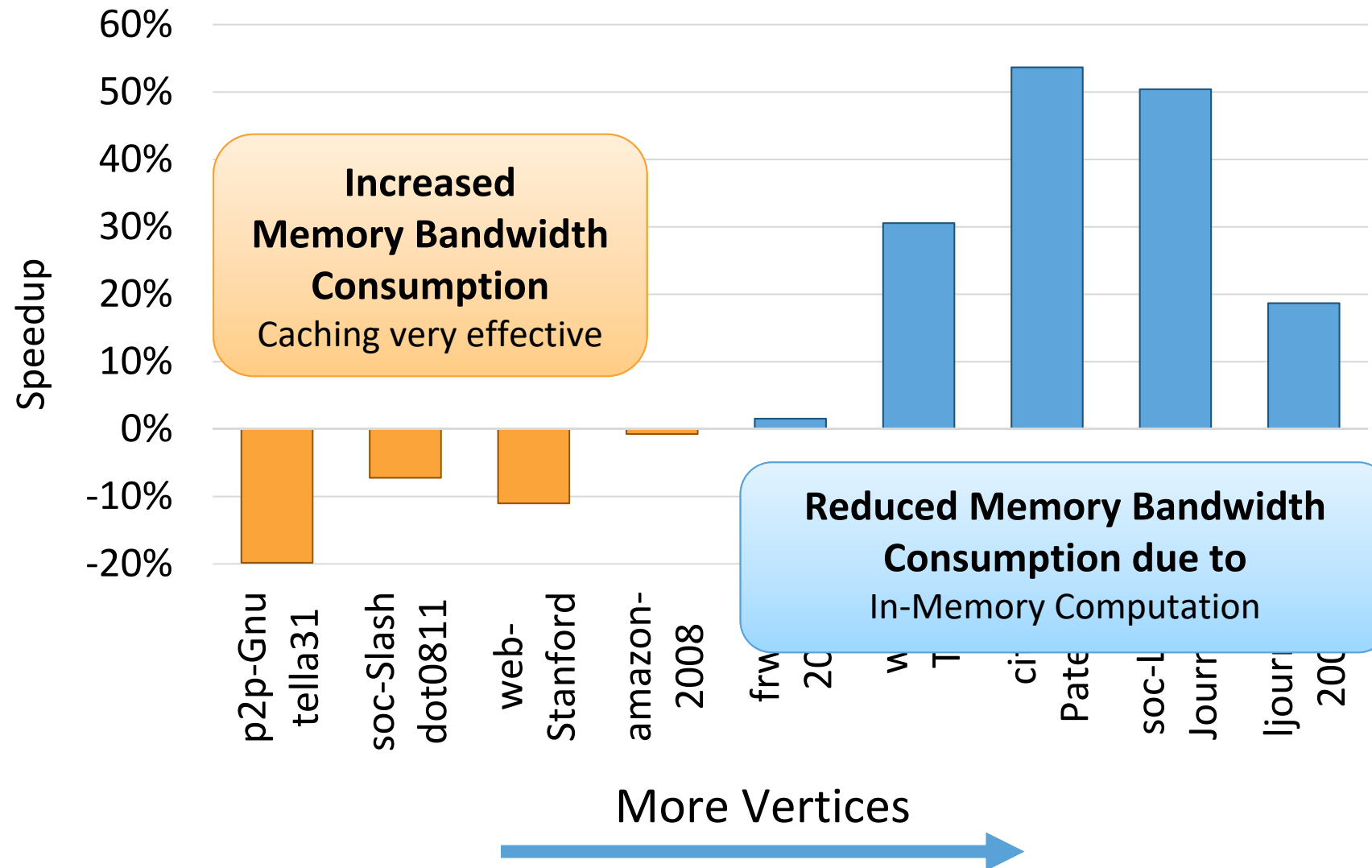
Main Memory



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(&w.next_rank, value);  
    }  
}
```

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

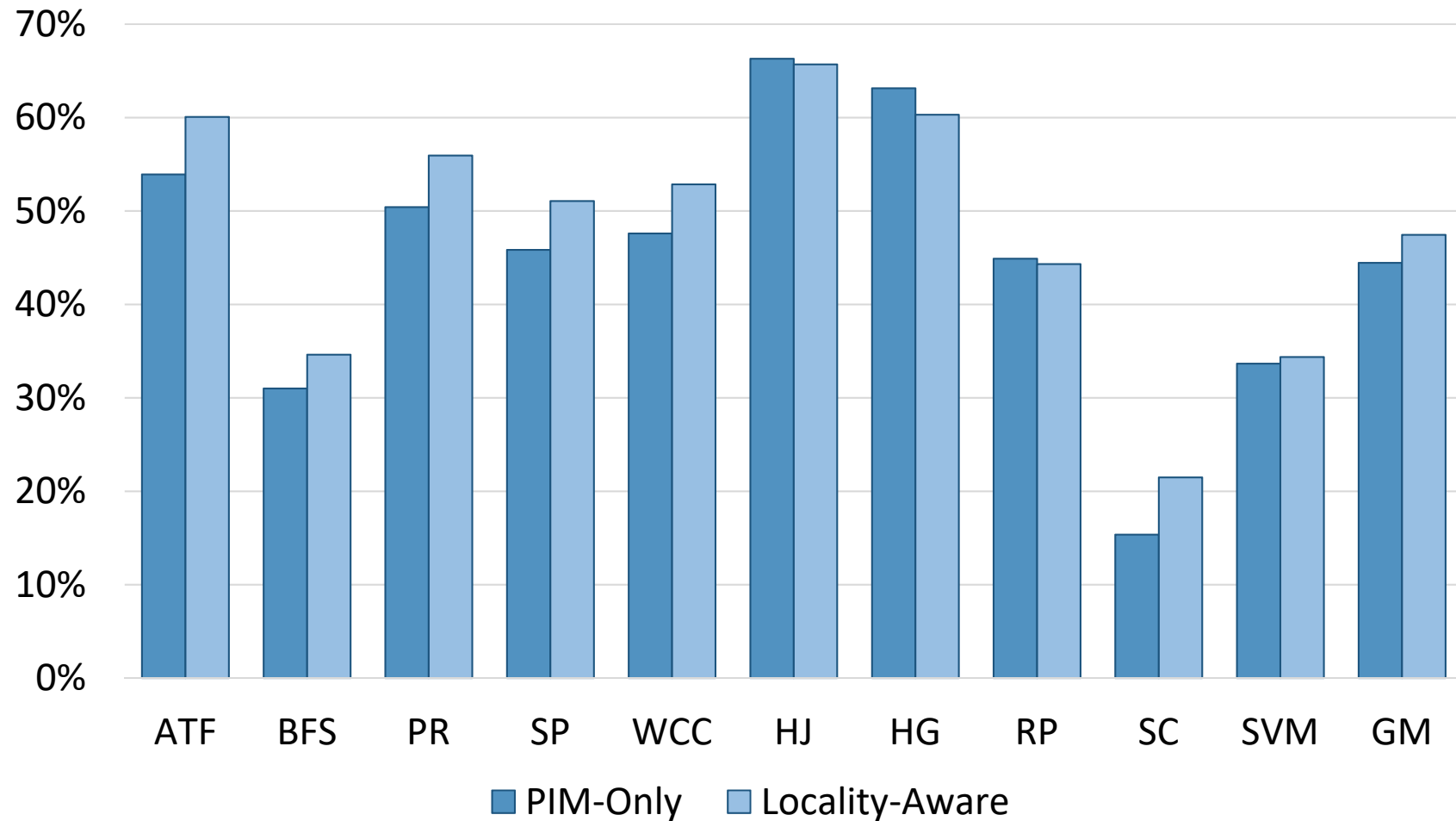
Component	Configuration
Core	16 out-of-order cores, 4 GHz, 4-issue
L1 I/D-Cache	Private, 32 KB, 4/8-way, 64 B blocks, 16 MSHRs
L2 Cache	Private, 256 KB, 8-way, 64 B blocks, 16 MSHRs
L3 Cache	Shared, 16 MB, 16-way, 64 B blocks, 64 MSHRs
On-Chip Network	Crossbar, 2 GHz, 144-bit links
Main Memory	32 GB, 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

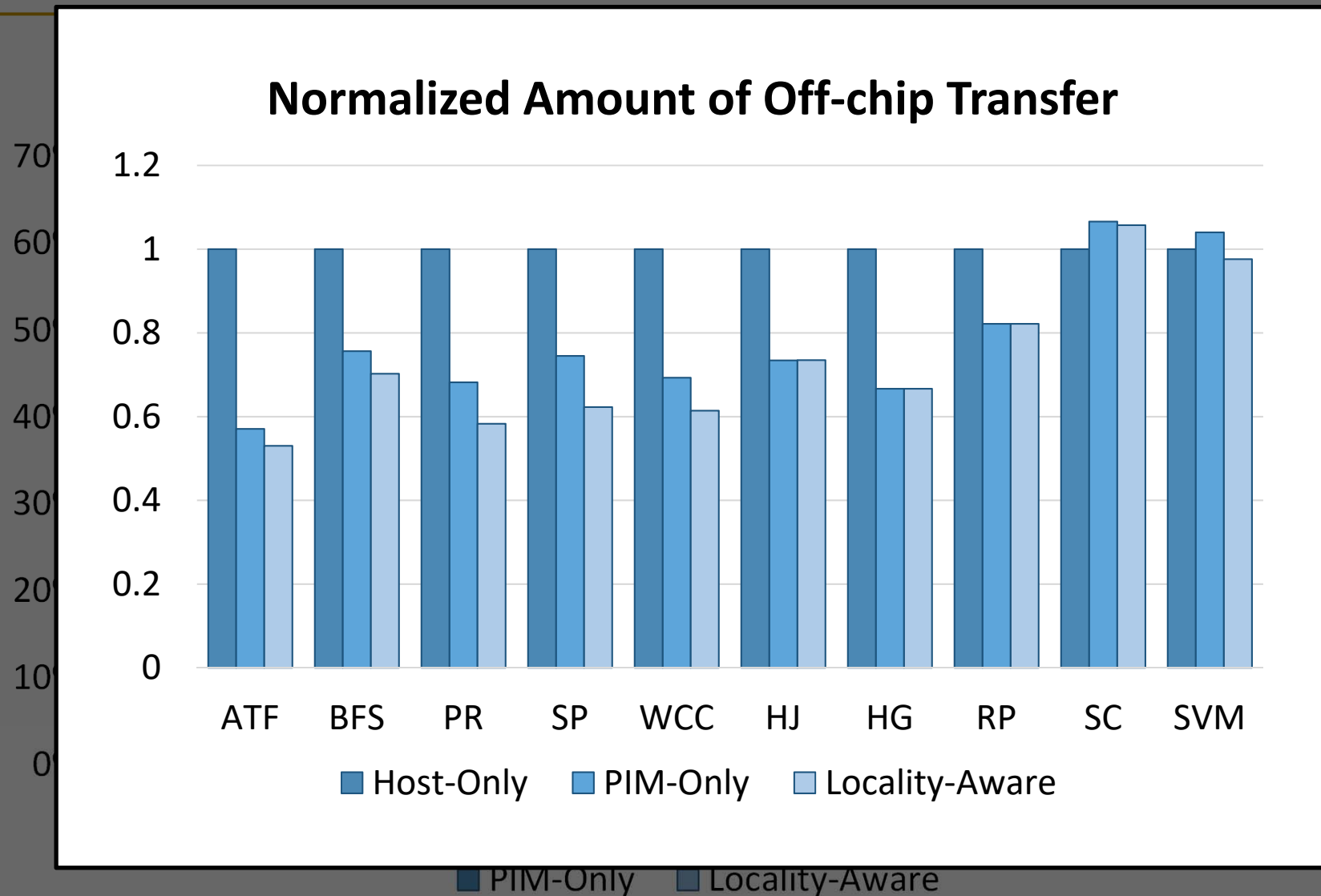
- Ten emerging data-intensive workloads
 - Large-scale graph processing
 - Average teenage follower, BFS, PageRank, single-source shortest path, weakly connected components
 - In-memory data analytics
 - Hash join, histogram, radix partitioning
 - Machine learning and data mining
 - Streamcluster, SVM-RFE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

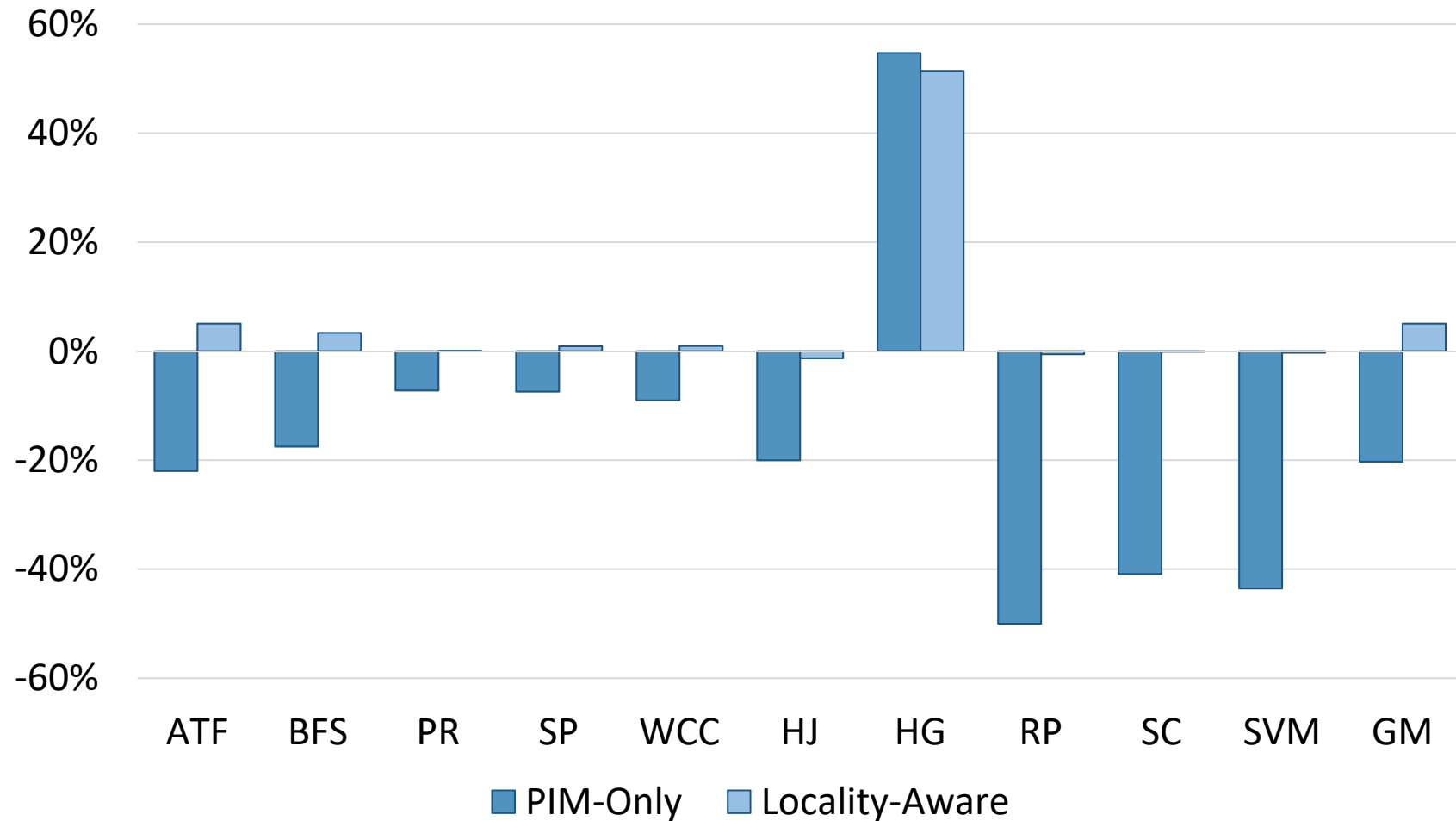


PEI Performance: Large Data Sets

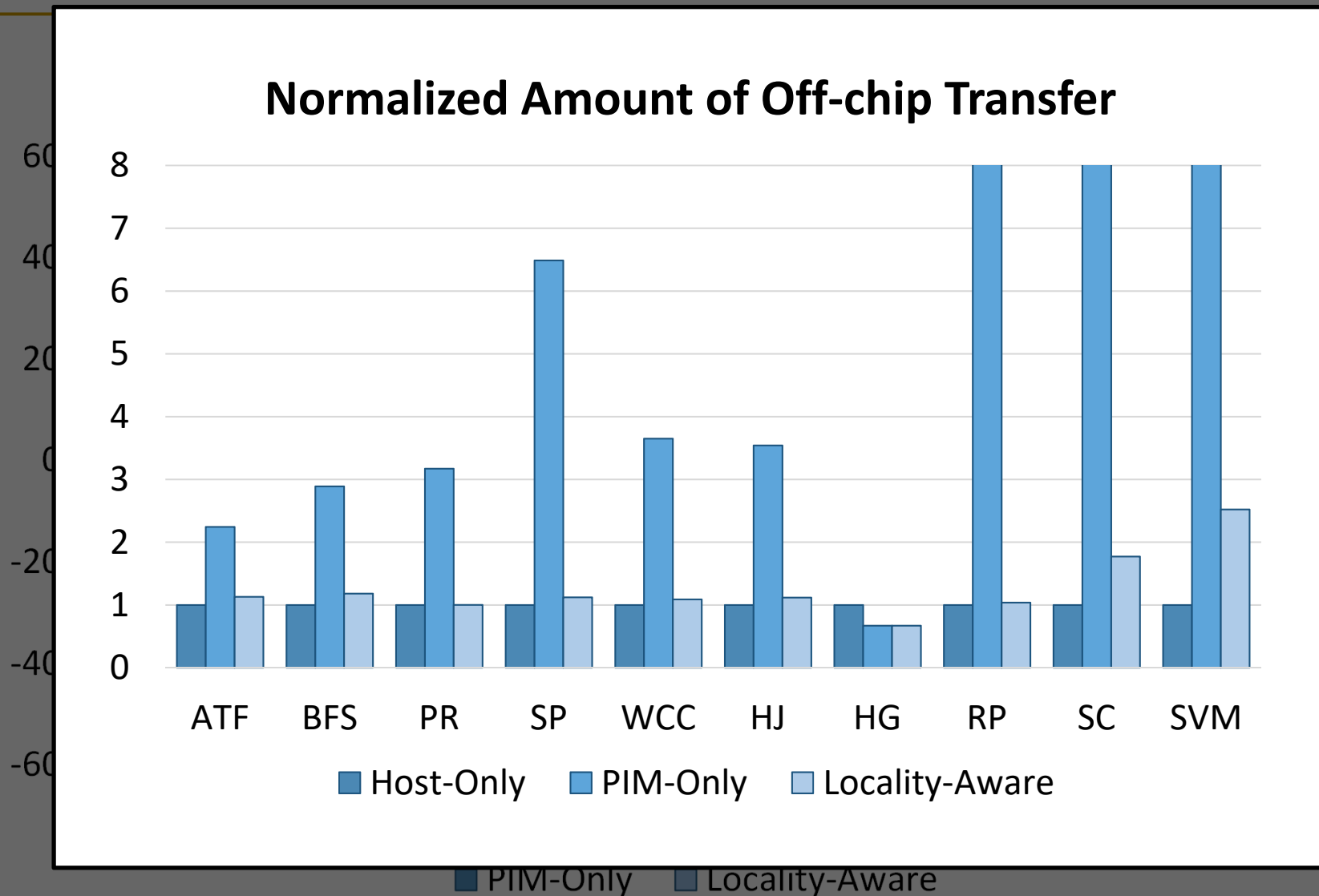


PEI Performance Delta: Small Data Sets

(Small Inputs, Baseline: Host-Only)

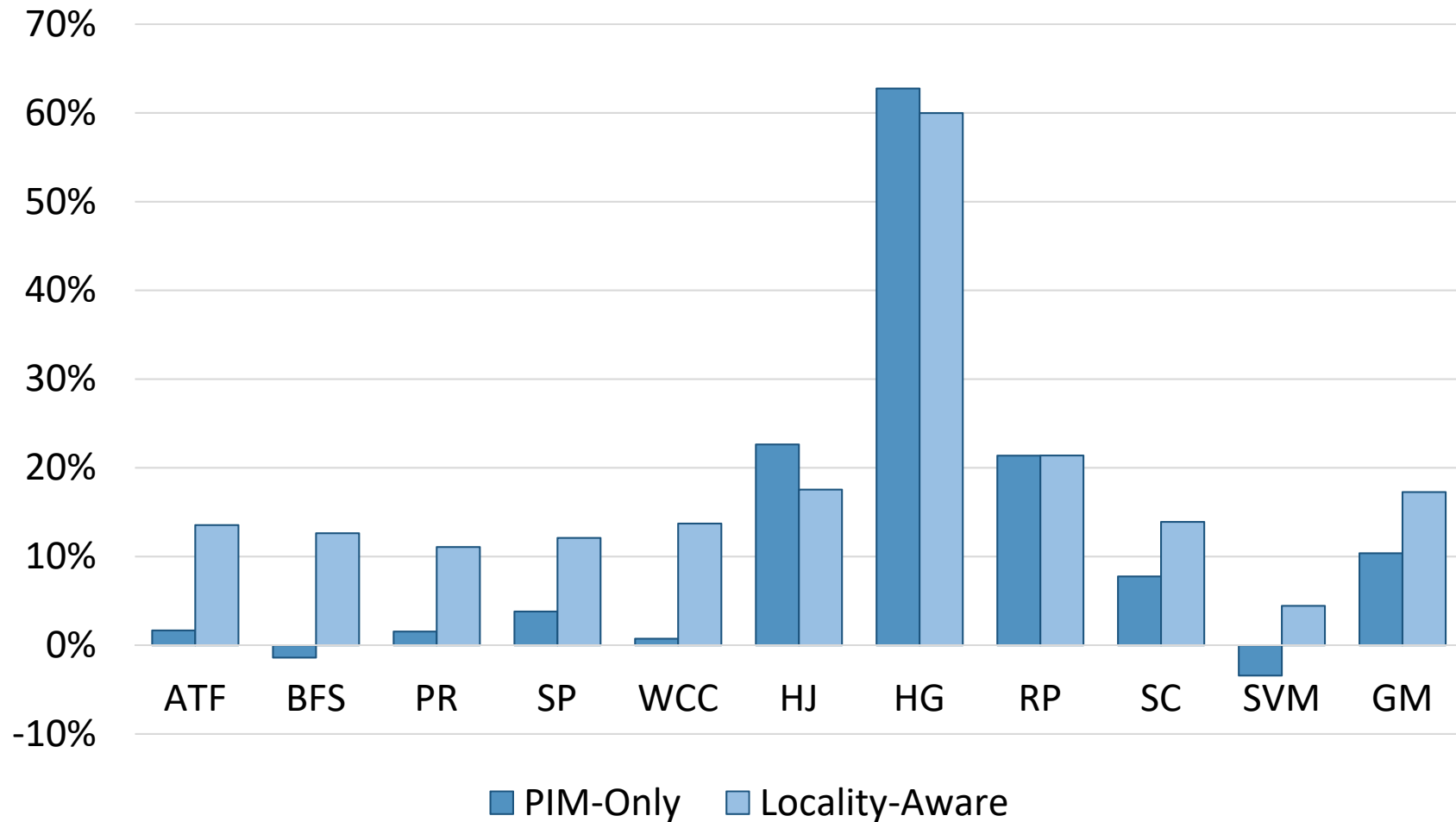


PEI Performance: Small Data Sets

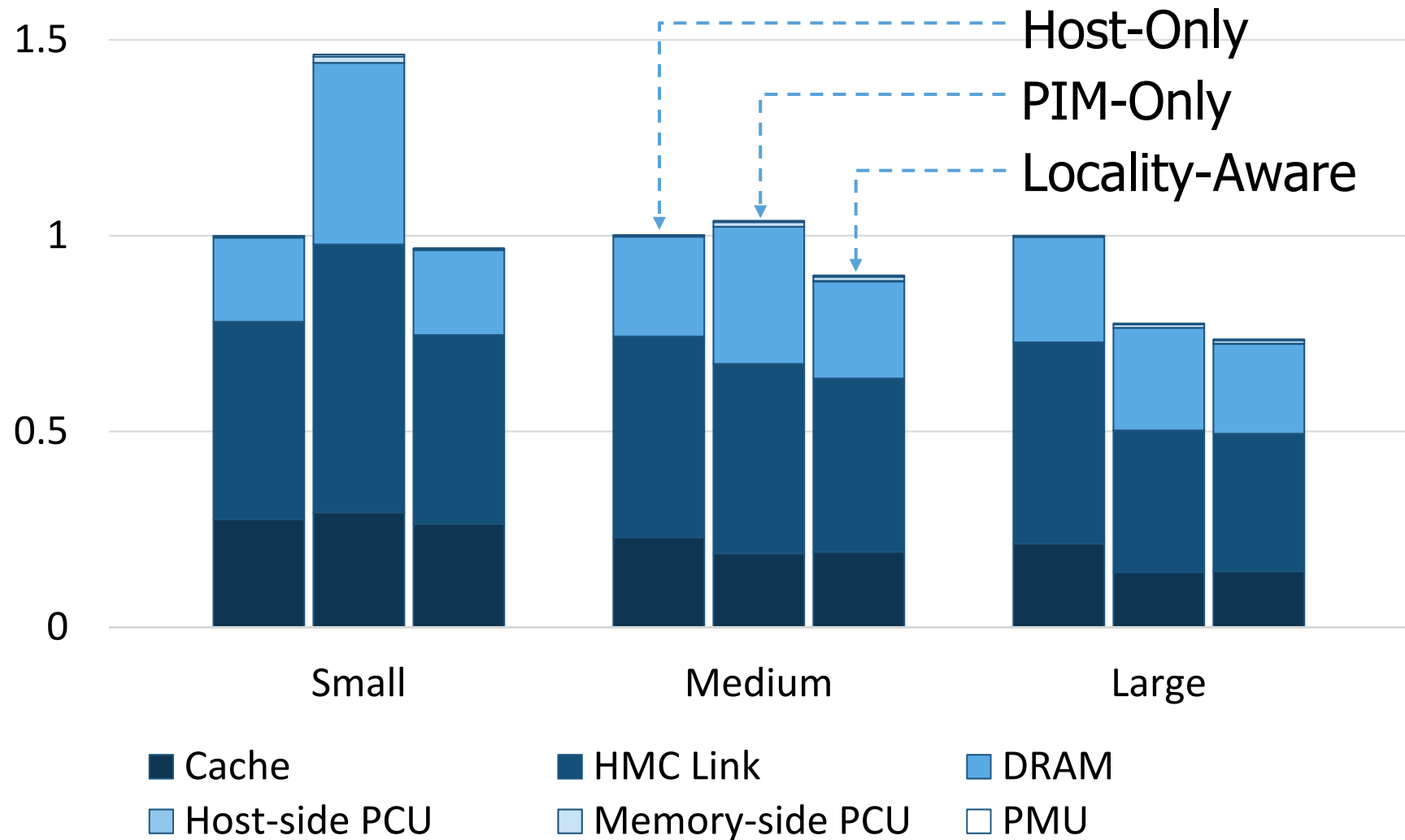


PEI Performance Delta: Medium Data Sets

(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

SAFARI

Automatic Code and Data Mapping

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

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

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

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

Automatic Offloading of Critical Code

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

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

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

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

Automatic Offloading of Prefetch Mechanisms

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

Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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

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

Efficient Automatic Data Coherence Support

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

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

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

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

Efficient Automatic Data Coherence Support

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

CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

Amirali Boroumand [†]	Saugata Ghose [†]	Minesh Patel [*]	Hasan Hassan [*]
Brandon Lucia [†]	Rachata Ausavarungnirun ^{†‡}	Kevin Hsieh [†]	
Nastaran Hajinazar ^{◊†}	Krishna T. Malladi [§]	Hongzhong Zheng [§]	Onur Mutlu ^{★†}
[†] Carnegie Mellon University		[*] ETH Zürich	[‡] KMUTNB
[◊] Simon Fraser University		[§] Samsung Semiconductor, Inc.	

Fundamentally
Energy-Efficient
(Data-Centric)
Computing Architectures

Fundamentally
High-Performance
(Data-Centric)
Computing Architectures

Computing Architectures with Minimal Data Movement

Sub-Agenda: In-Memory Computation

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

Eliminating the Adoption Barriers

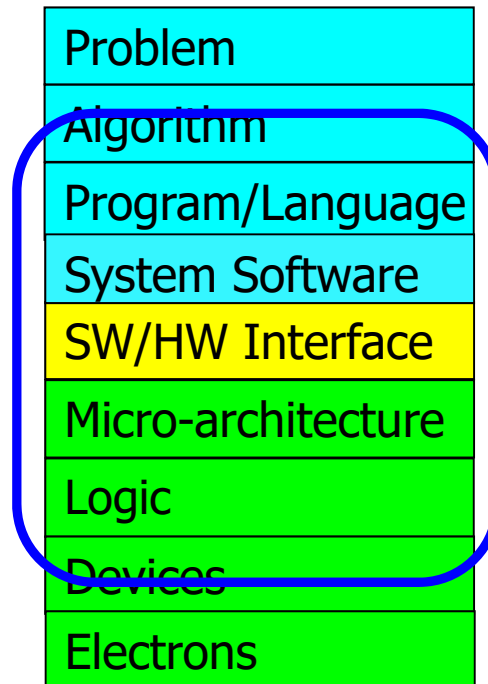
How to Enable Adoption of Processing in Memory

Barriers to Adoption of PIM

1. Functionality of and applications & software for PIM
2. Ease of programming (interfaces and compiler/HW support)
3. System support: coherence & virtual memory
4. Runtime and compilation systems for adaptive scheduling, data mapping, access/sharing control
5. Infrastructures to assess benefits and feasibility

All can be solved with change of mindset

We Need to Revisit the Entire Stack



We can get there step by step

PIM Review and Open Problems

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

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

^a*ETH Zürich*

^b*Carnegie Mellon University*

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

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
**"Processing Data Where It Makes Sense: Enabling In-Memory
Computation"**

*Invited paper in Microprocessors and Microsystems (**MICPRO**), June 2019.*

[[arXiv version](#)]

PIM Review and Open Problems (II)

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

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

[†]*Carnegie Mellon University*

[§]*ETH Zürich*

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,

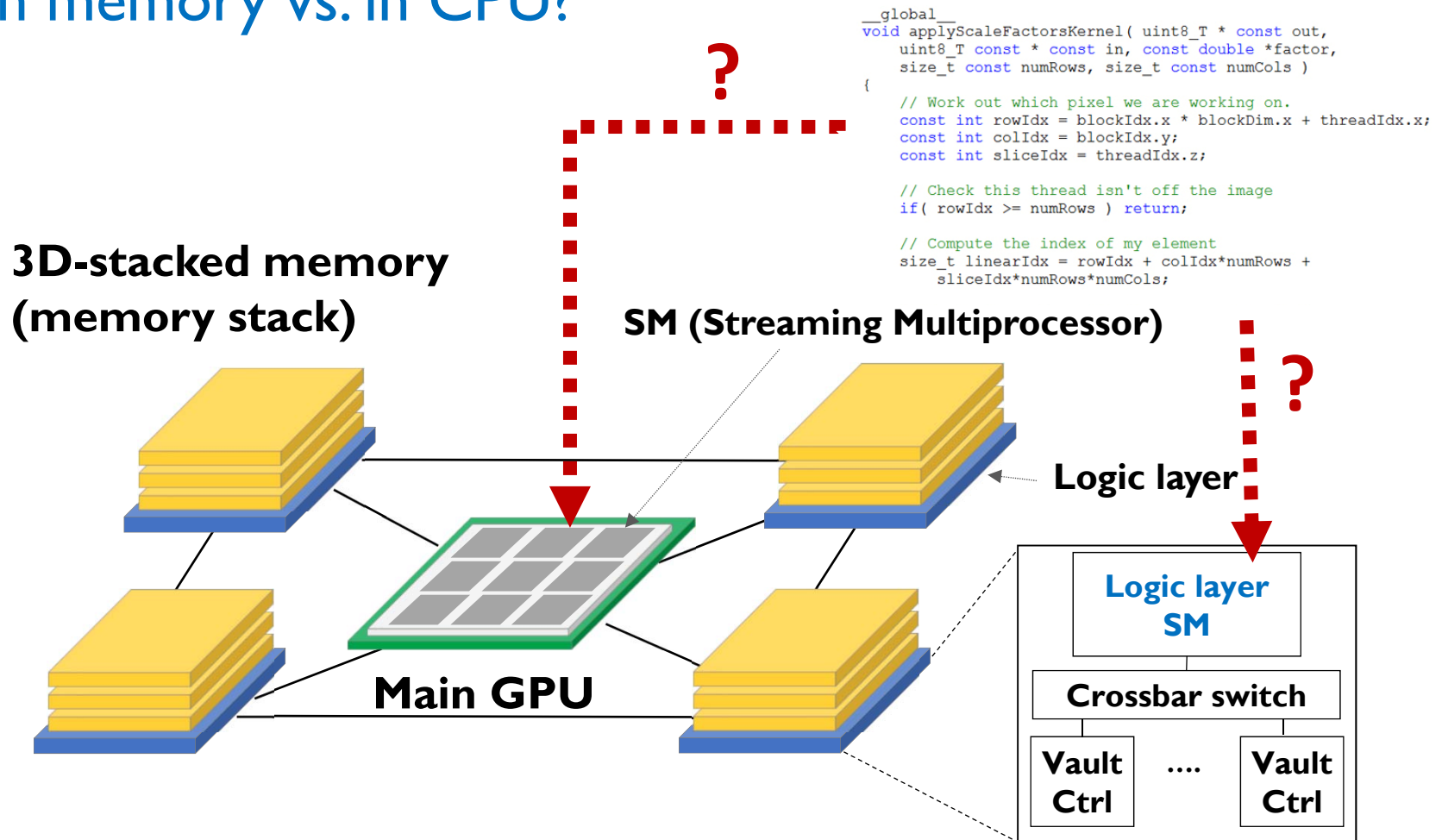
"Processing-in-Memory: A Workload-Driven Perspective"

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

[Preliminary arXiv version]

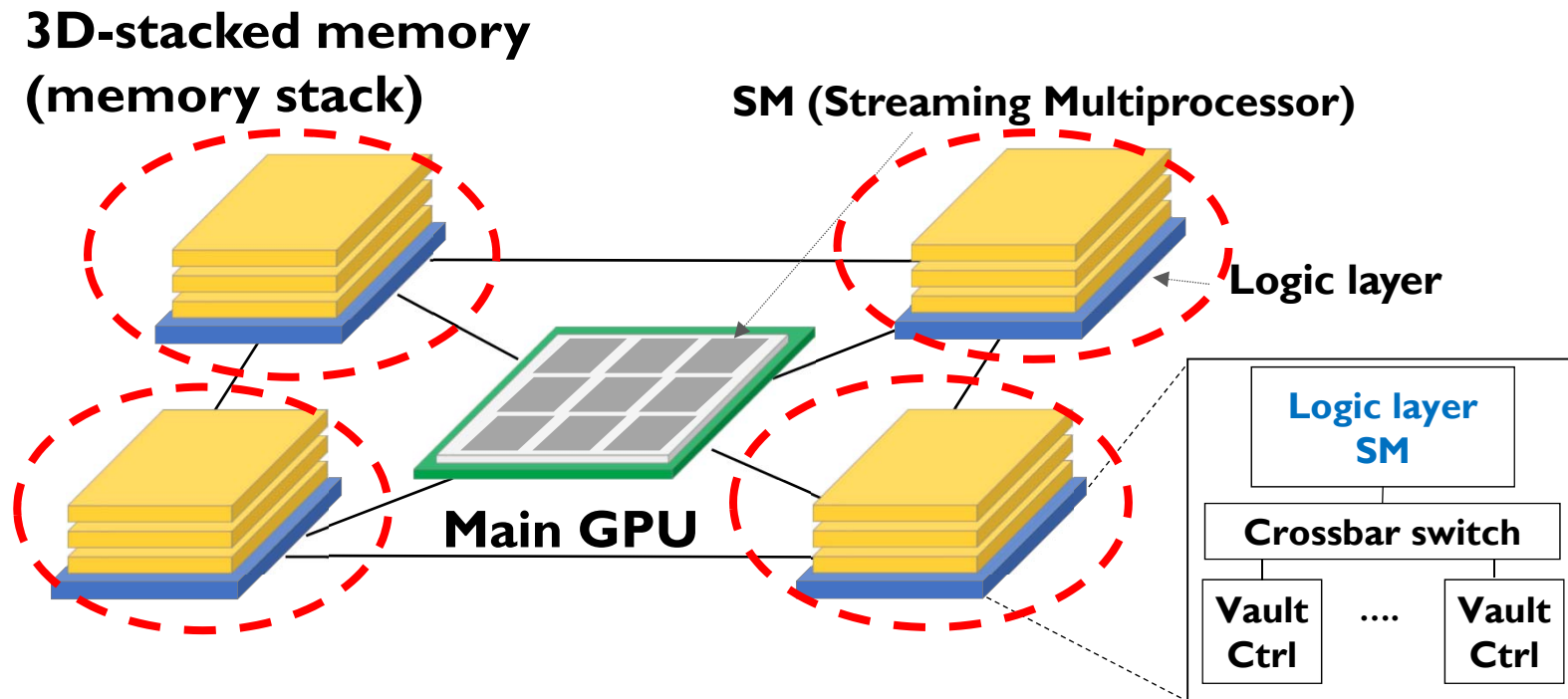
Key Challenge 1: Code Mapping

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



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)

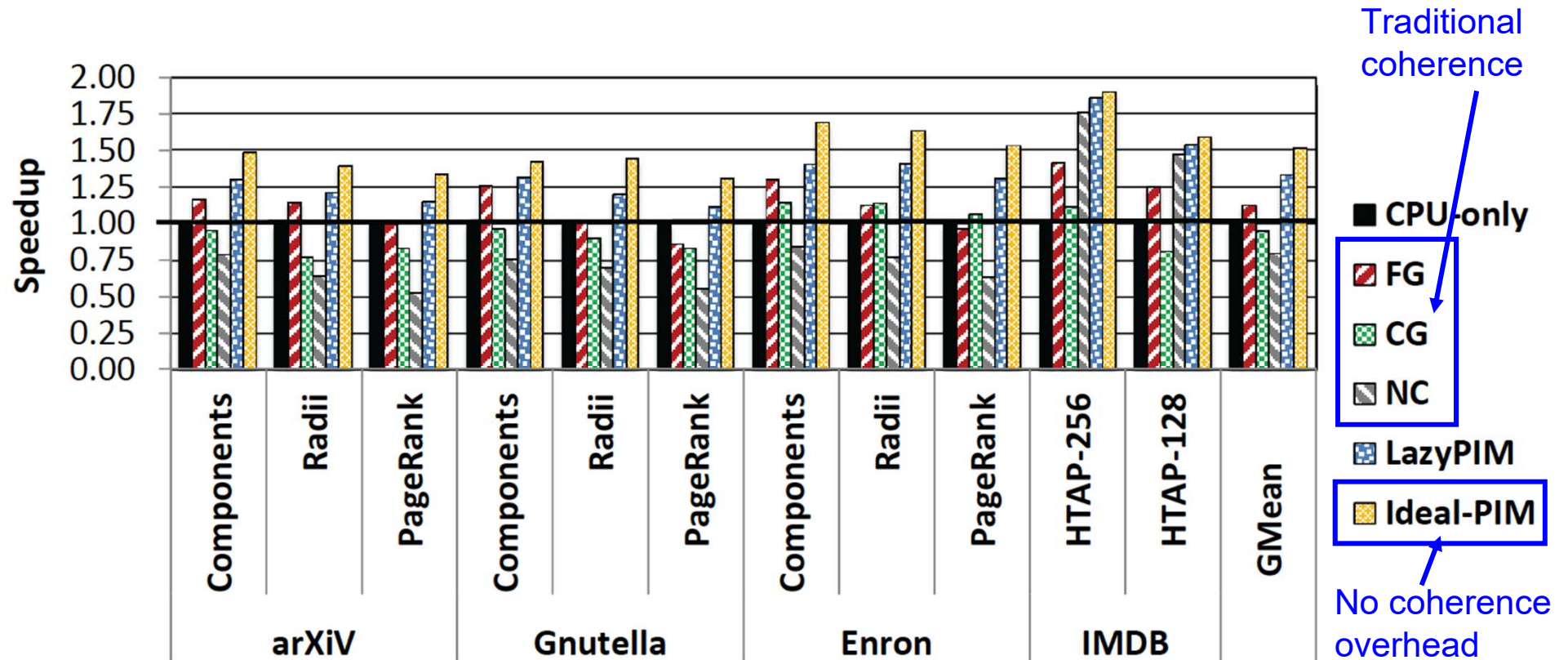
- Milad Hashemi, Onur Mutlu, and Yale N. Patt,
"Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads"
Proceedings of the 49th International Symposium on Microarchitecture (MICRO), Taipei, Taiwan, October 2016.
[[Slides \(pptx\)](#)] [[pdf](#)] [[Lightning Session Slides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

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.	

How to Support Virtual Memory?

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

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

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

How to Design Data Structures for PIM?

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

Concurrent Data Structures for Near-Memory Computing

Zhiyu Liu

Computer Science Department
Brown University
zhiyu.liu@brown.edu

Irina Calciu

VMware Research Group
icalciu@vmware.com

Maurice Herlihy

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

Onur Mutlu

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

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

Performance & Energy Models for PIM

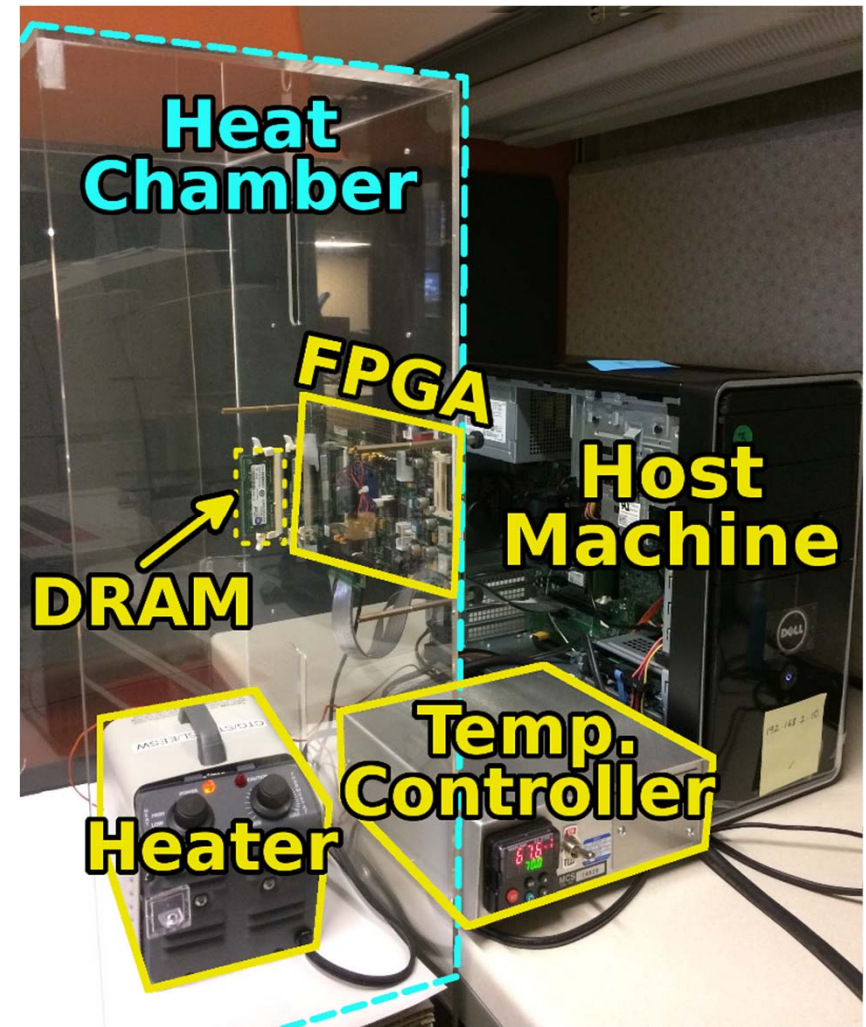
- Gagandeep Singh, Juan Gomez-Luna, Giovanni Mariani, Geraldo F. Oliveira, Stefano Corda, Sander Stujik, Onur Mutlu, and Henk Corporaal, **"NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning"**
Proceedings of the 56th Design Automation Conference (DAC), Las Vegas, NV,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

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

Gagandeep Singh ^{a,c}	Juan Gómez-Luna ^b	Giovanni Mariani ^c	Geraldo F. Oliveira ^b
Stefano Corda ^{a,c}	Sander Stuijk ^a	Onur Mutlu ^b	Henk Corporaal ^a
^a Eindhoven University of Technology		^b ETH Zürich	^c IBM Research - Zurich

An FPGA-based Test-bed for PIM?

- Hasan Hassan et al., **SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies** HPCA 2017.
- Flexible
- Easy to Use (C++ API)
- Open-source
github.com/CMU-SAFARI/SoftMC



Simulation Infrastructures for PIM (in SSDs)

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

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

Arash Tavakkol[†], Juan Gómez-Luna[†], Mohammad Sadrosadati[†], Saugata Ghose[‡], Onur Mutlu^{†‡}
[†]*ETH Zürich* [‡]*Carnegie Mellon University*

New Applications and Use Cases for PIM

- Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu, **"GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies"** ***BMC Genomics***, 2018.
Proceedings of the 16th Asia Pacific Bioinformatics Conference (APBC), Yokohama, Japan, January 2018.
[arxiv.org Version \(pdf\)](https://arxiv.org/abs/1801.00000)

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**.

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

PIM Review and Open Problems

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

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

^a*ETH Zürich*

^b*Carnegie Mellon University*

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

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
**"Processing Data Where It Makes Sense: Enabling In-Memory
Computation"**

*Invited paper in Microprocessors and Microsystems (**MICPRO**), June 2019.*

[arXiv version]

PIM Review and Open Problems (II)

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

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

[†]*Carnegie Mellon University*

[§]*ETH Zürich*

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,

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

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

[Preliminary arXiv version]

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures

Fundamentally
High-Performance
(Data-Centric)
Computing Architectures

Challenge and Opportunity for Future

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

- Recommendation

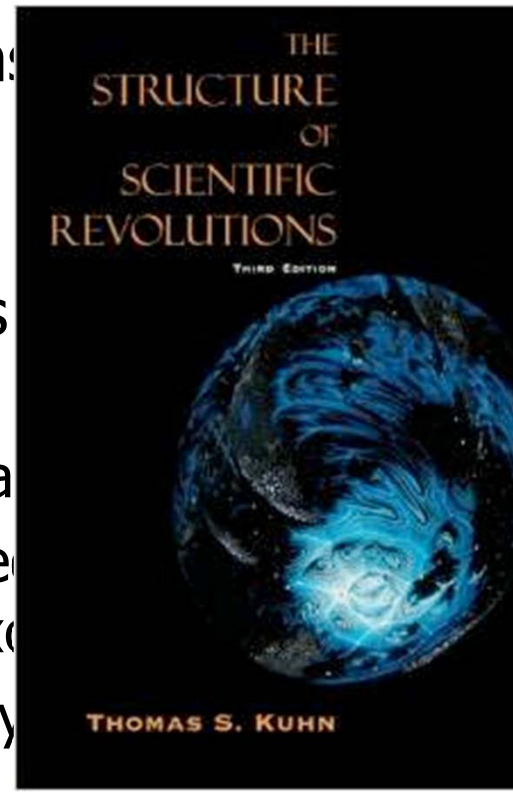
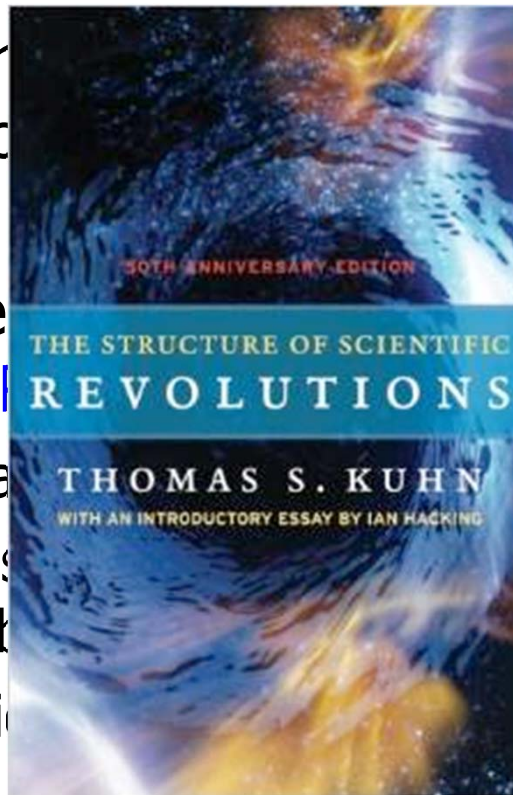
Scientific

Pre-para

Normal s

things (k

Revoluti



ure of

eld

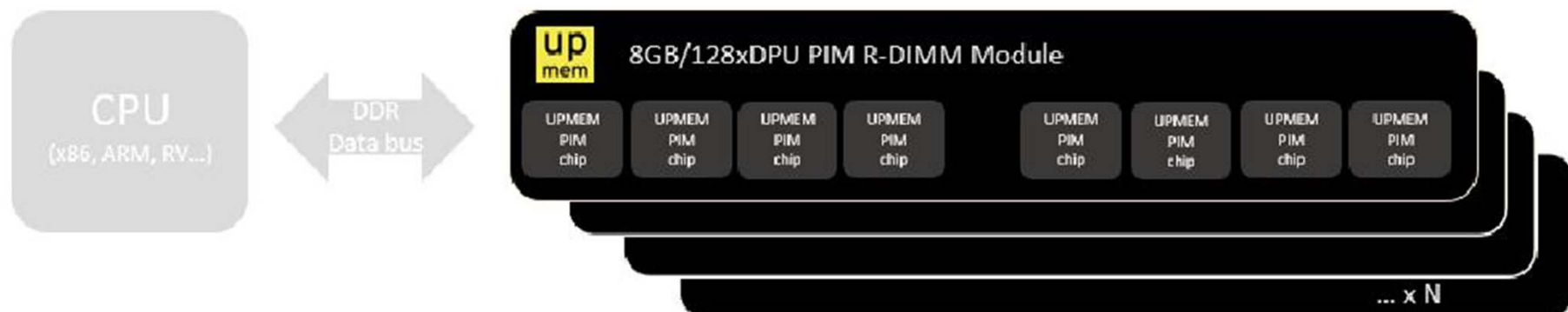
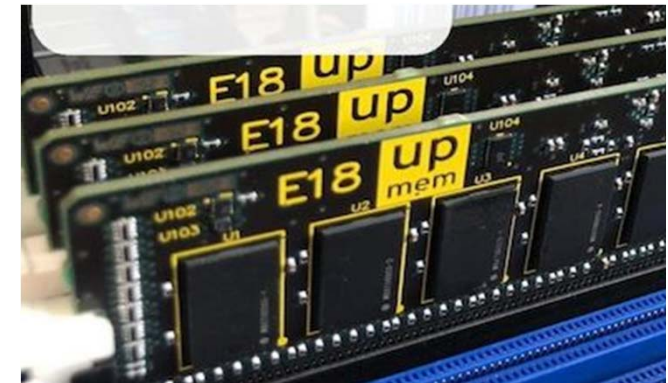
improve

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



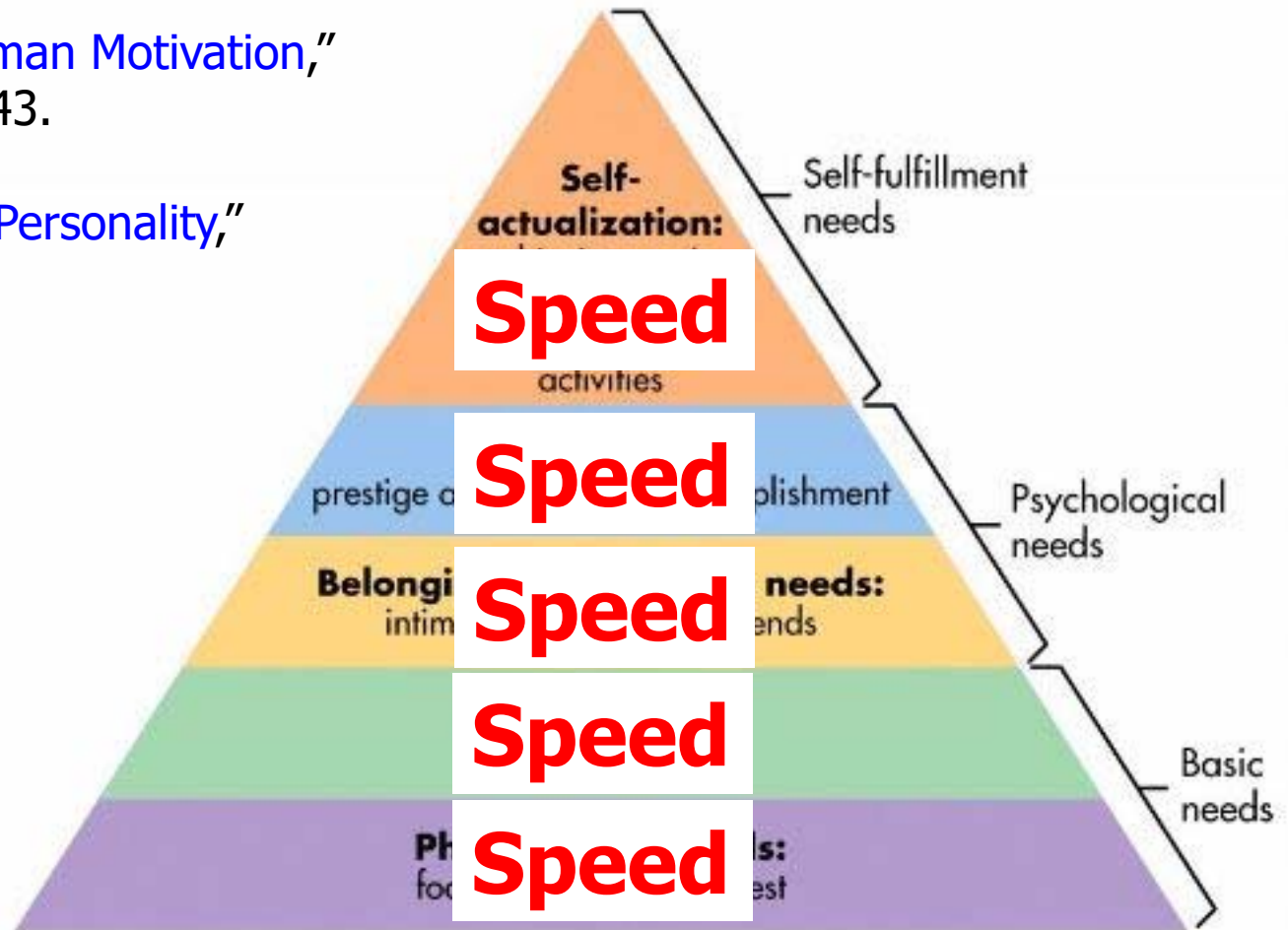
Sub-Agenda: In-Memory Computation

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

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

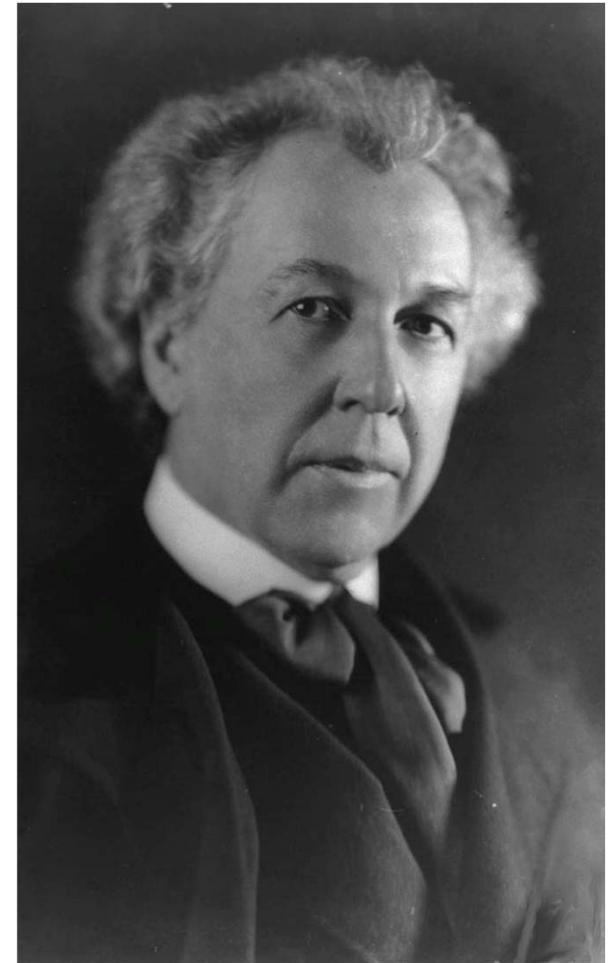
Fundamentally Low-Latency (Data-Centric) Computing Architectures

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.

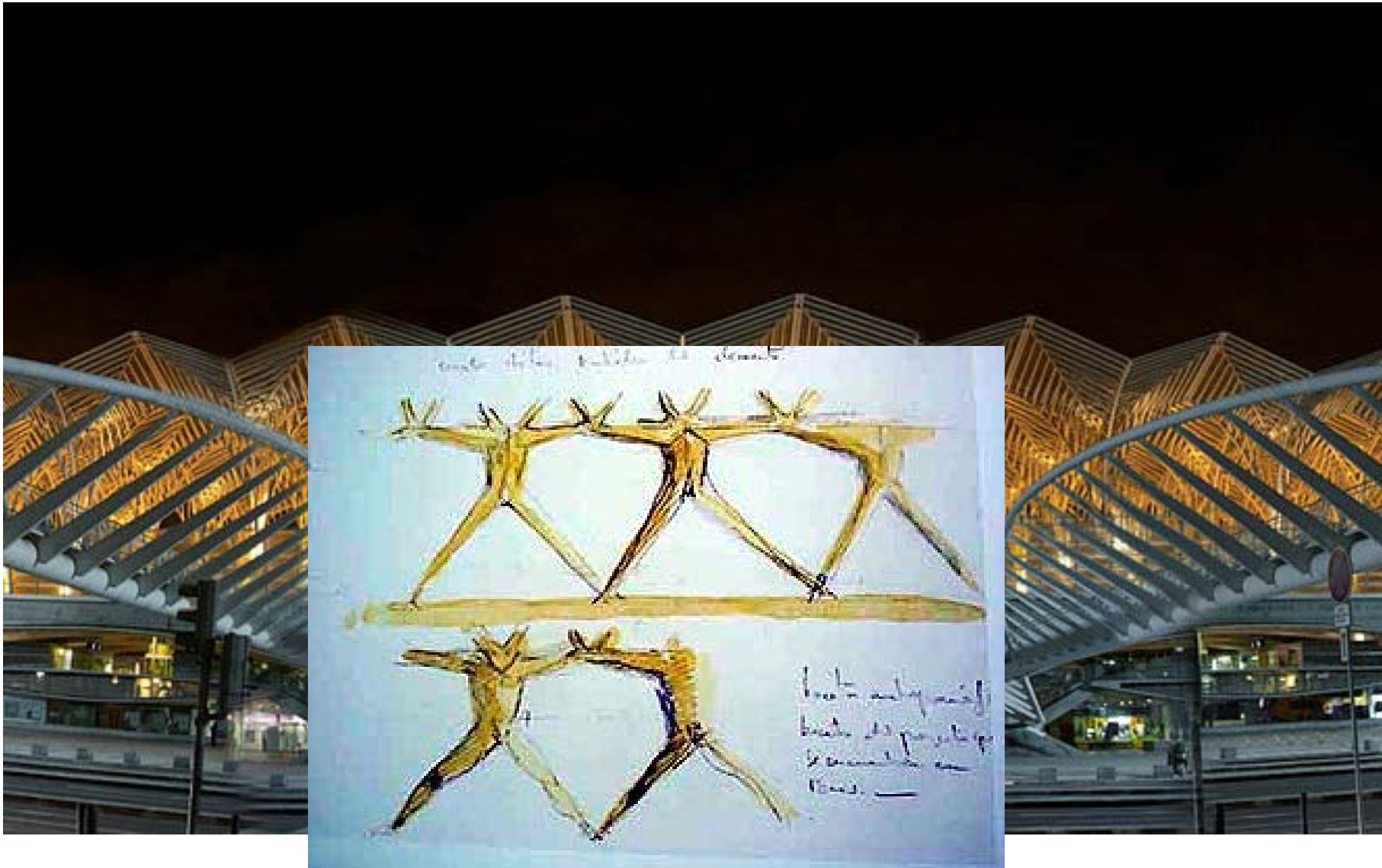
Another Example: Precedent-Based Design



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 準建築人手机網站 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/>

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?



Source: <http://spectrum.ieee.org/image/MjYzMzAyMg.jpeg>

Concluding Remarks

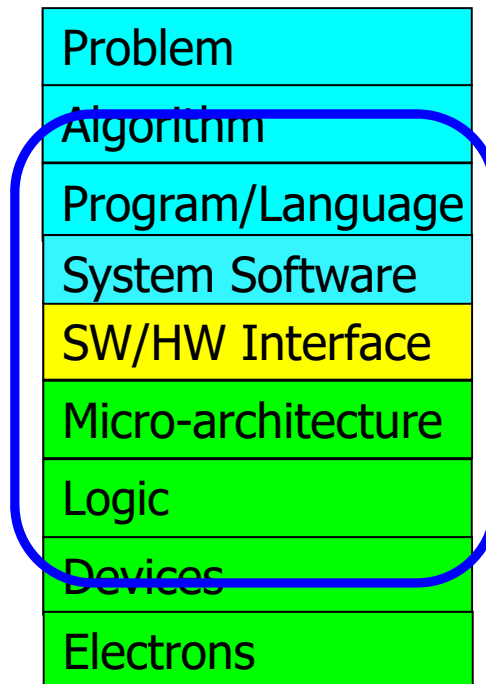
- It is time to design **principled system architectures** to solve the **memory problem**
- Design complete systems to be balanced, high-performance, and energy-efficient, i.e., **data-centric** (or **memory-centric**)
- Enable computation capability inside and close to memory
- **This** can
 - Lead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

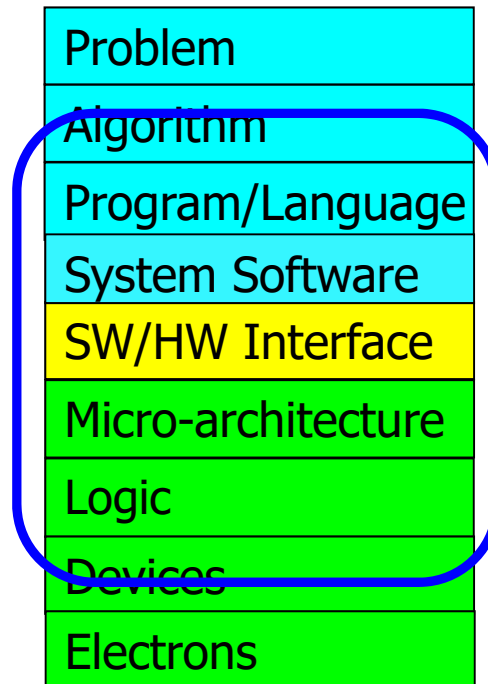
- Orders of magnitude improvements
- New applications and computing systems



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

If In Doubt, See Other Doubtful Technologies

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



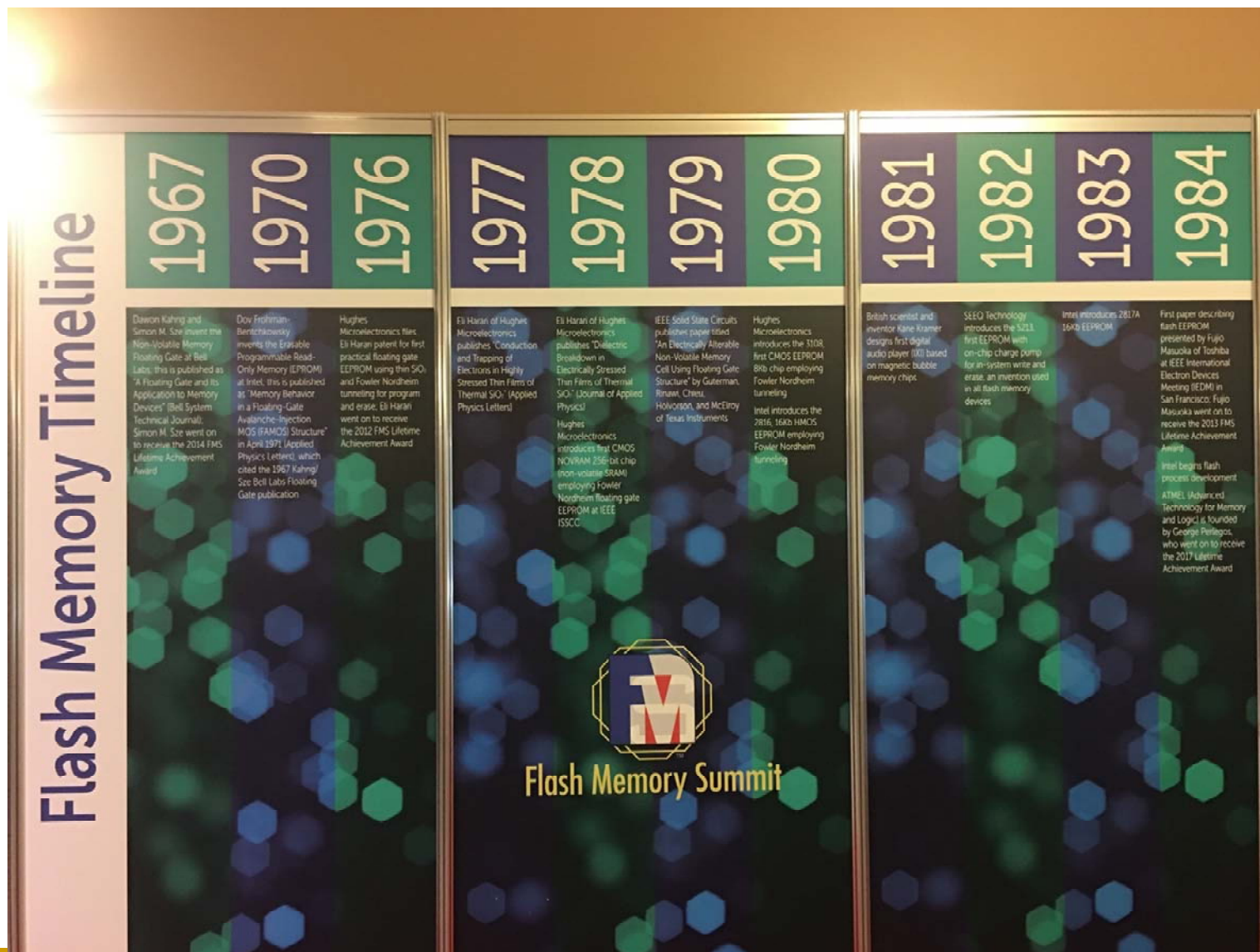
Proceedings of the IEEE, Sept. 2017

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



SAFARI



PIM Review and Open Problems

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

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

^a*ETH Zürich*

^b*Carnegie Mellon University*

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

Onur Mutlu, Saugata Ghose, Juan Gomez-Luna, and Rachata Ausavarungnirun,
**"Processing Data Where It Makes Sense: Enabling In-Memory
Computation"**

*Invited paper in Microprocessors and Microsystems (**MICPRO**), June 2019.*

[[arXiv version](#)]

PIM Review and Open Problems (II)

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

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

[†]*Carnegie Mellon University*

[§]*ETH Zürich*

Saugata Ghose, Amirali Boroumand, Jeremie S. Kim, Juan Gomez-Luna, and Onur Mutlu,

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

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

[Preliminary arXiv version]

Computer Architecture

Lecture 7: Computation in Memory II

Prof. Onur Mutlu

ETH Zürich

Fall 2019

10 October 2019

Accelerating Linked Data Structures

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

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

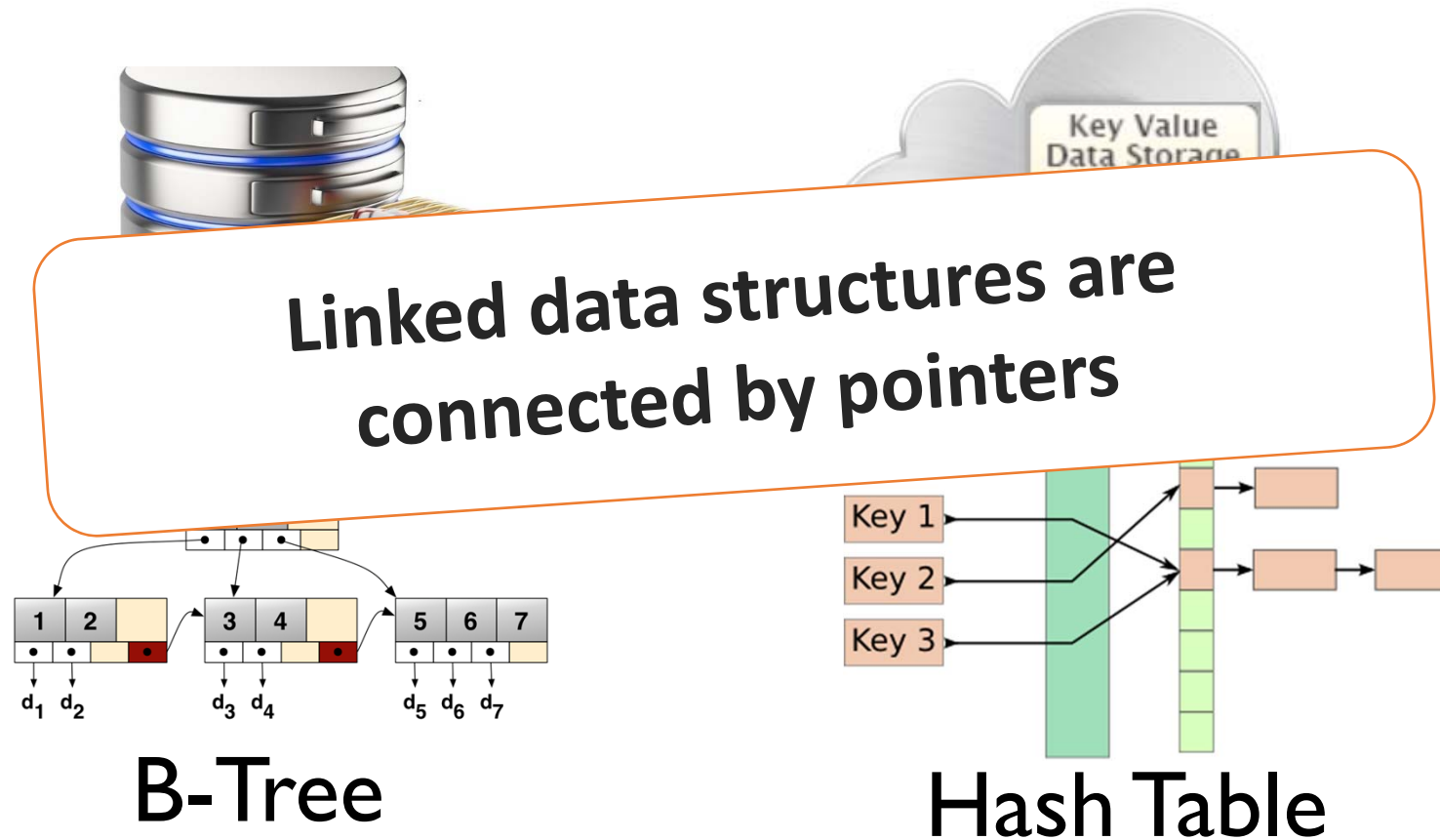
Kevin Hsieh[†] Samira Khan[‡] Nandita Vijaykumar[†]
Kevin K. Chang[†] Amirali Boroumand[†] Saugata Ghose[†] Onur Mutlu^{§†}
[†]*Carnegie Mellon University* [‡]*University of Virginia* [§]*ETH Zürich*

Executive Summary

- **Our Goal:** Accelerating pointer chasing inside main memory
- **Challenges:** Parallelism challenge and Address translation challenge
- **Our Solution:** In-Memory Pointer Chasing Accelerator (IMPICA)
 - Address-access decoupling: enabling parallelism in the accelerator with low cost
 - IMPICA page table: low cost page table in logic layer
- **Key Results:**
 - 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
 - 6% - 41% reduction in energy consumption

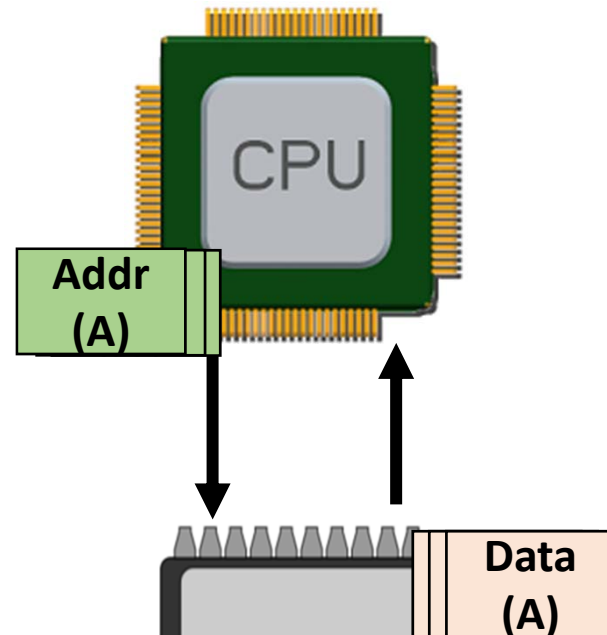
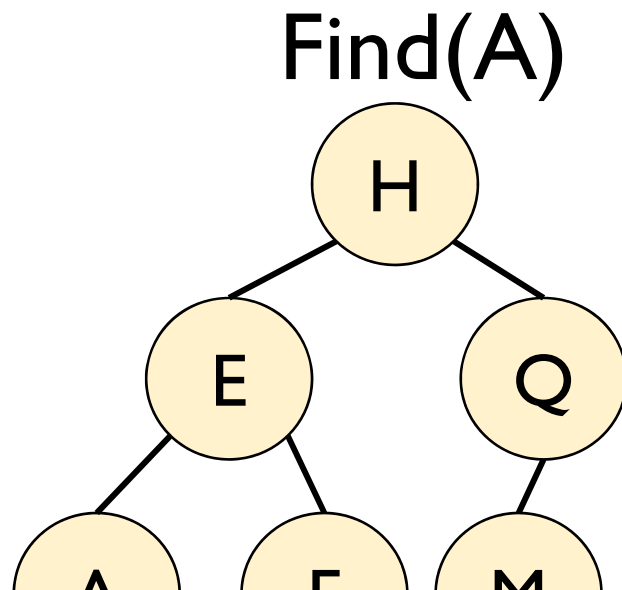
Linked Data Structures

- Linked data structures are widely used in many important applications



The Problem: Pointer Chasing

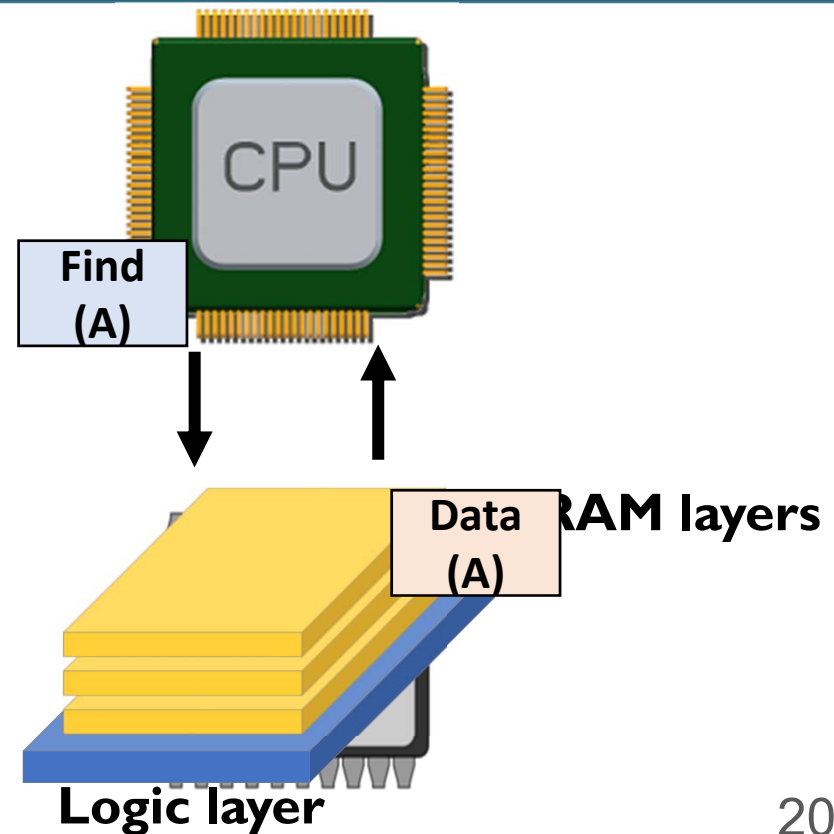
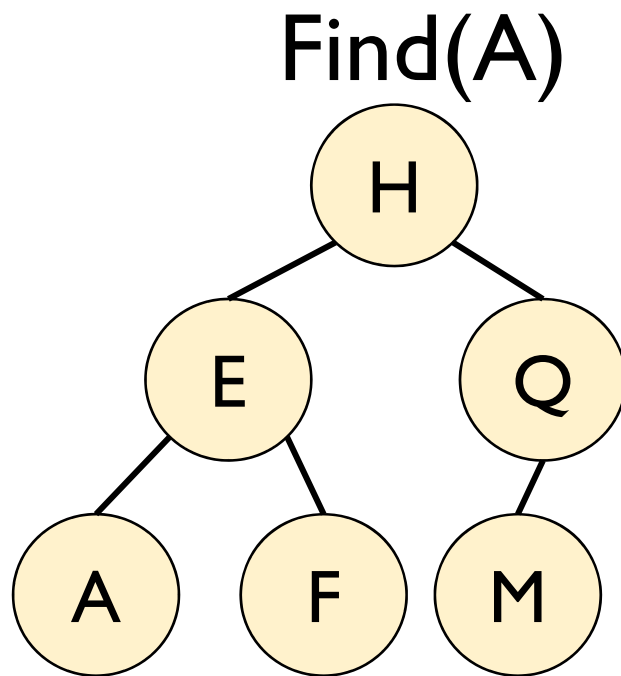
- Traversing linked data structures requires chasing pointers



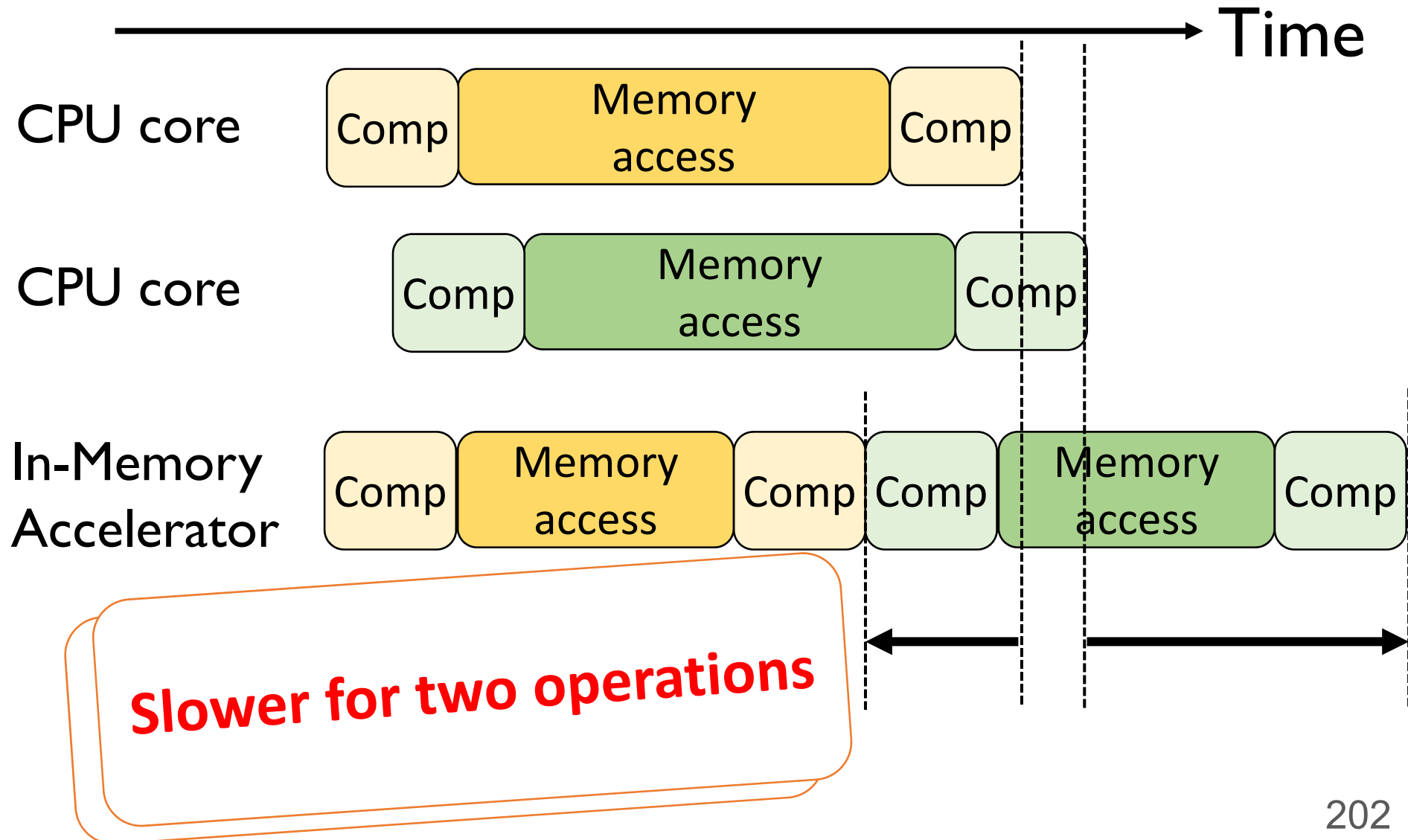
**Serialized and irregular access pattern
6X cycles per instruction in real workloads**

Our Goal

Accelerating pointer chasing inside main memory

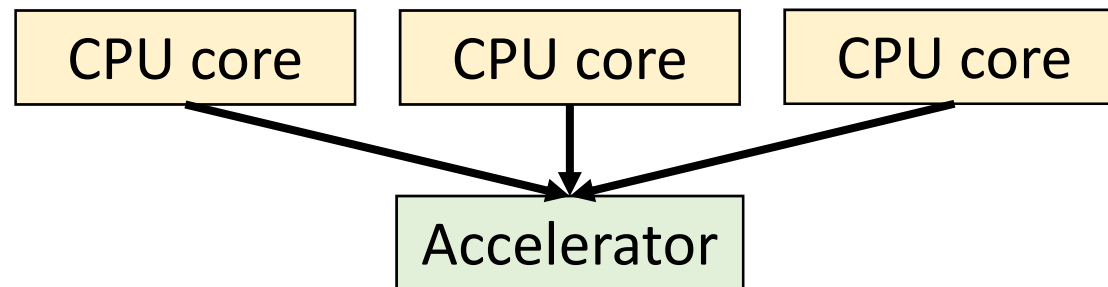


Parallelism Challenge

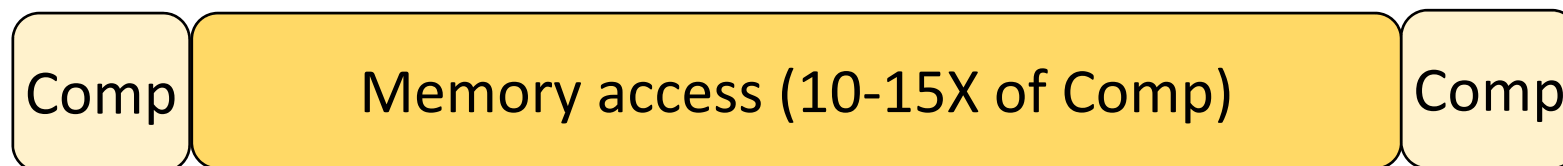


Parallelism Challenge and Opportunity

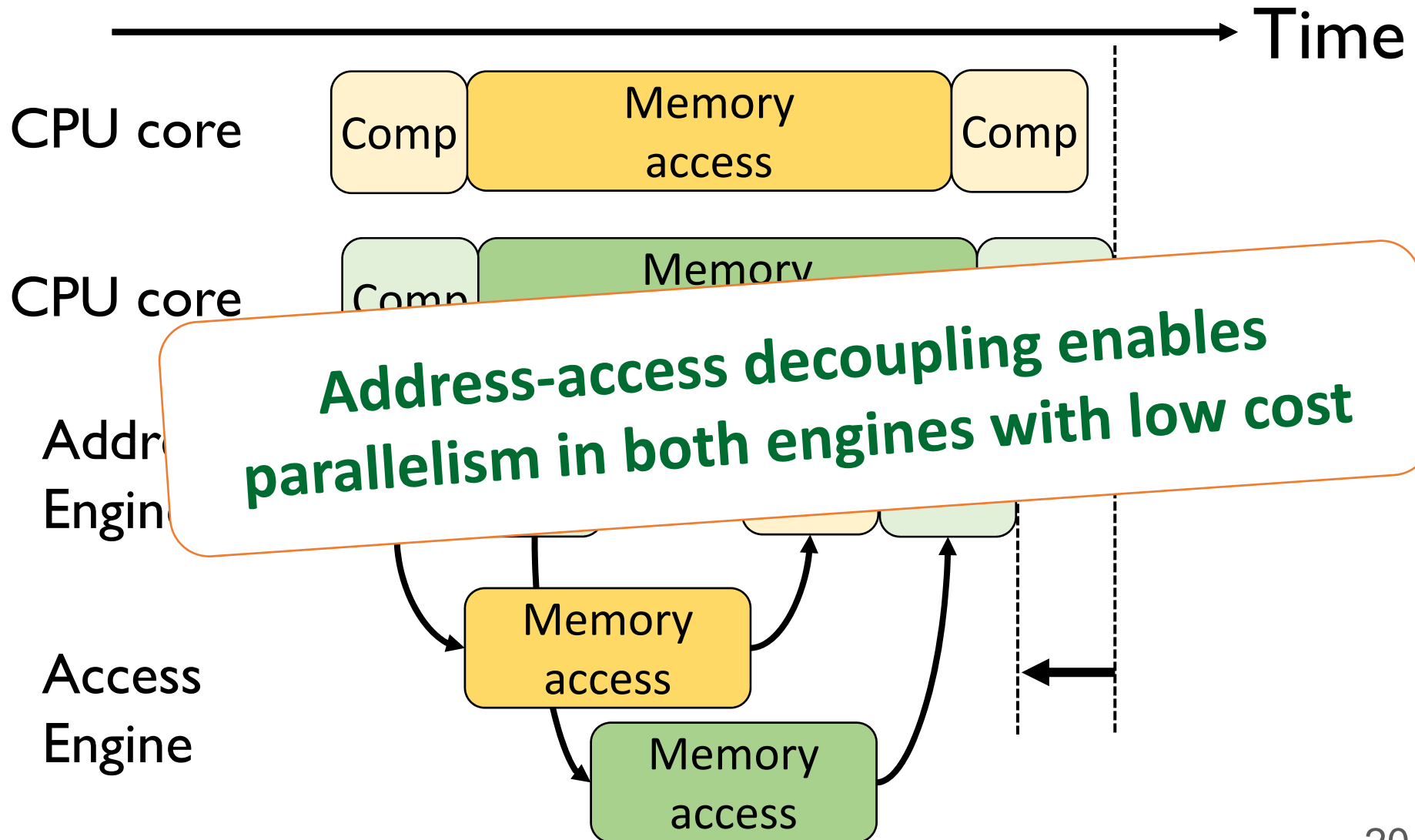
- A simple in-memory accelerator can still be **slower** than multiple CPU cores



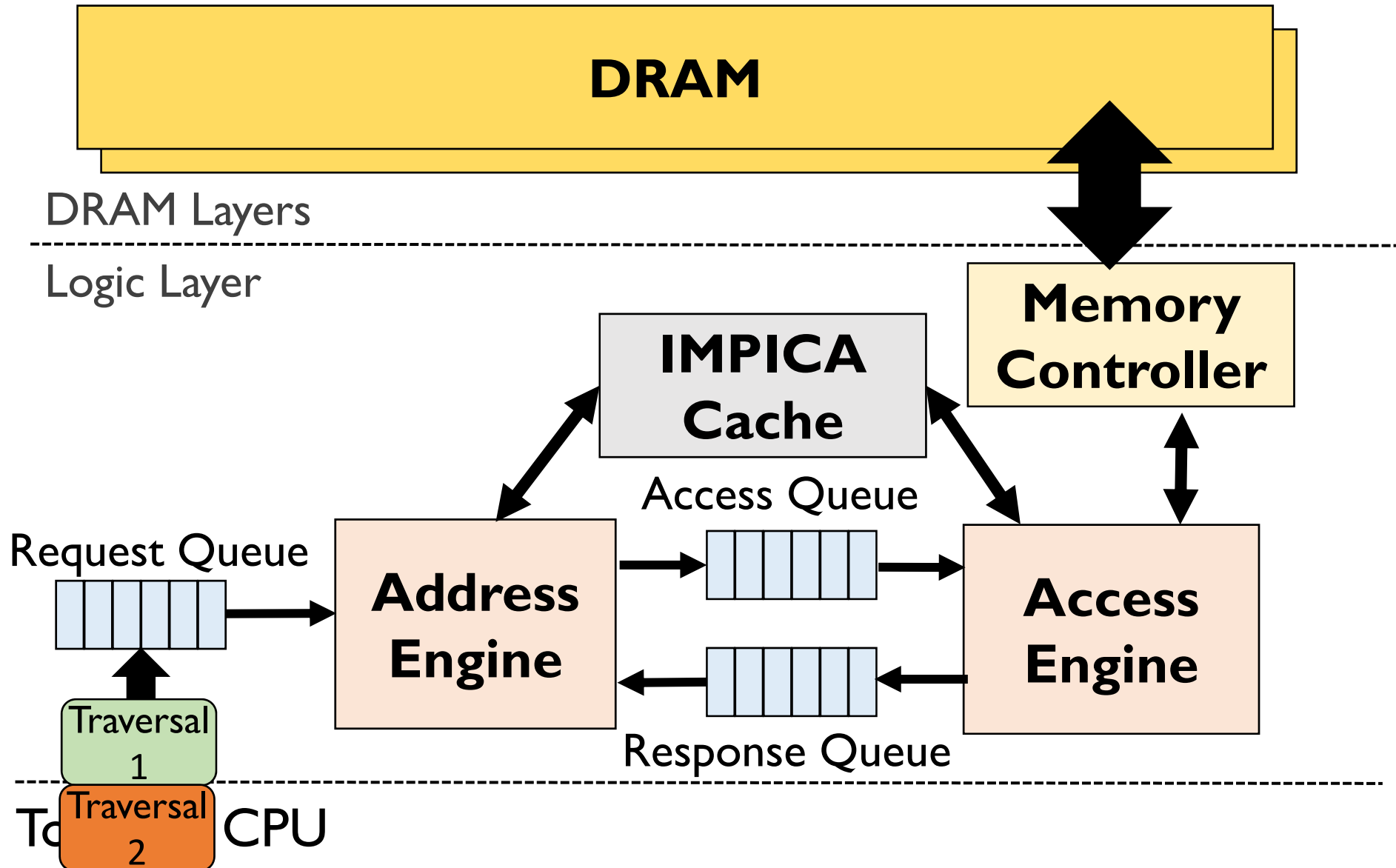
- **Opportunity:** a pointer-chasing accelerator spends a long time **waiting for memory**



Our Solution: Address-Access Decoupling

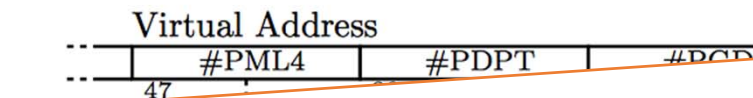


IMPICA Core Architecture



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

PML4

PDPT

PGD

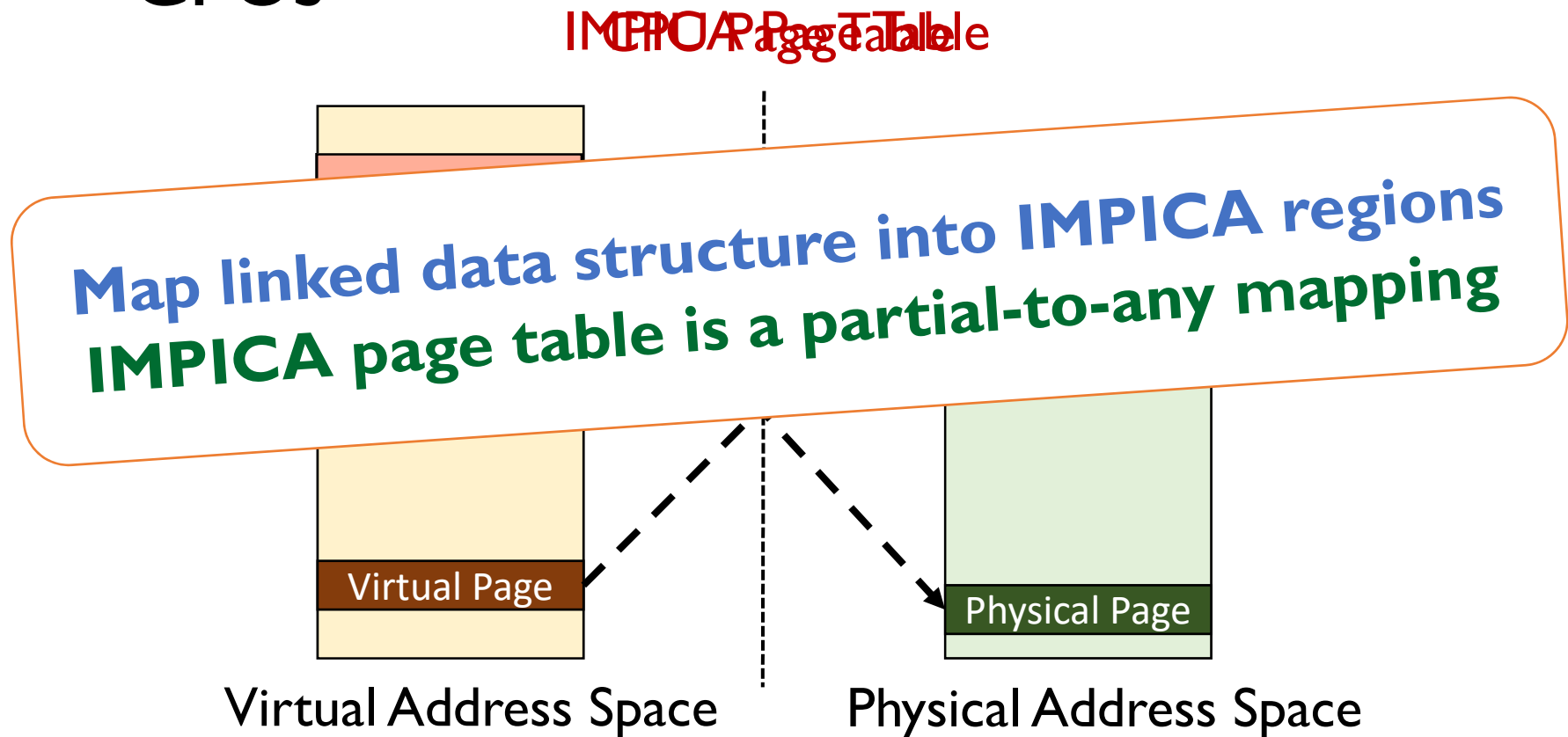
PGT

2^9

Page table walk

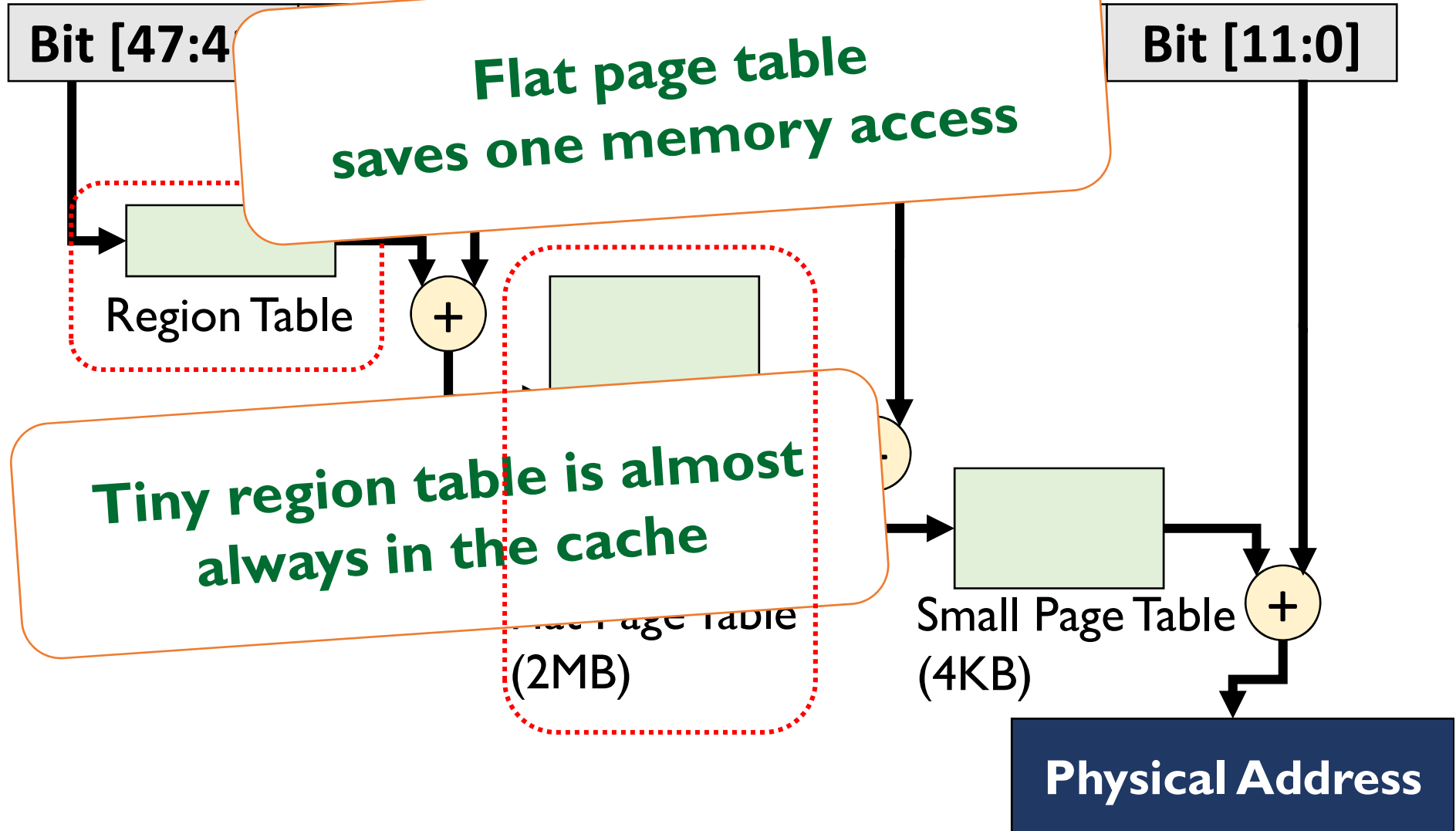
Our Solution: IMPICA Page Table

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



IMPICA Page Table: Mechanism

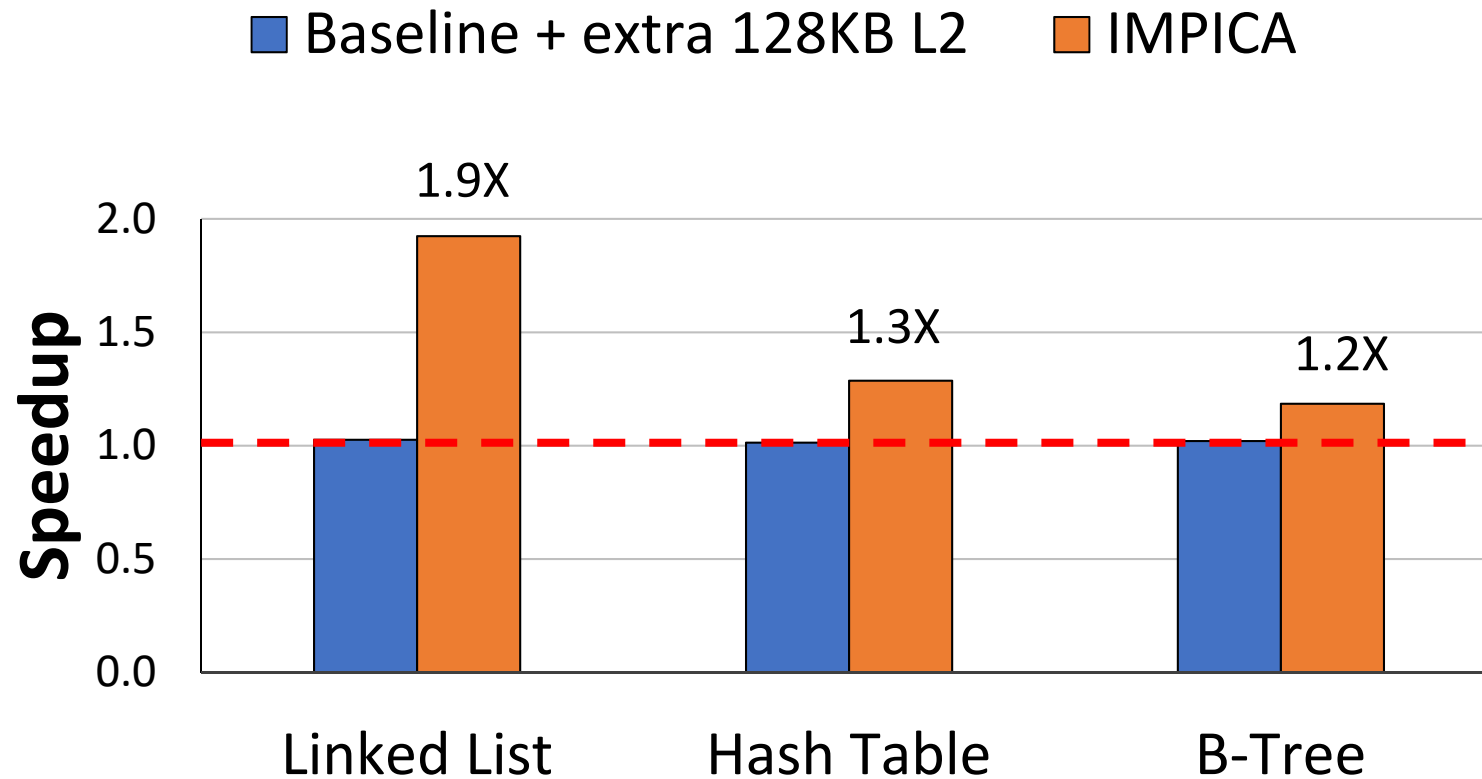
Virtual Address



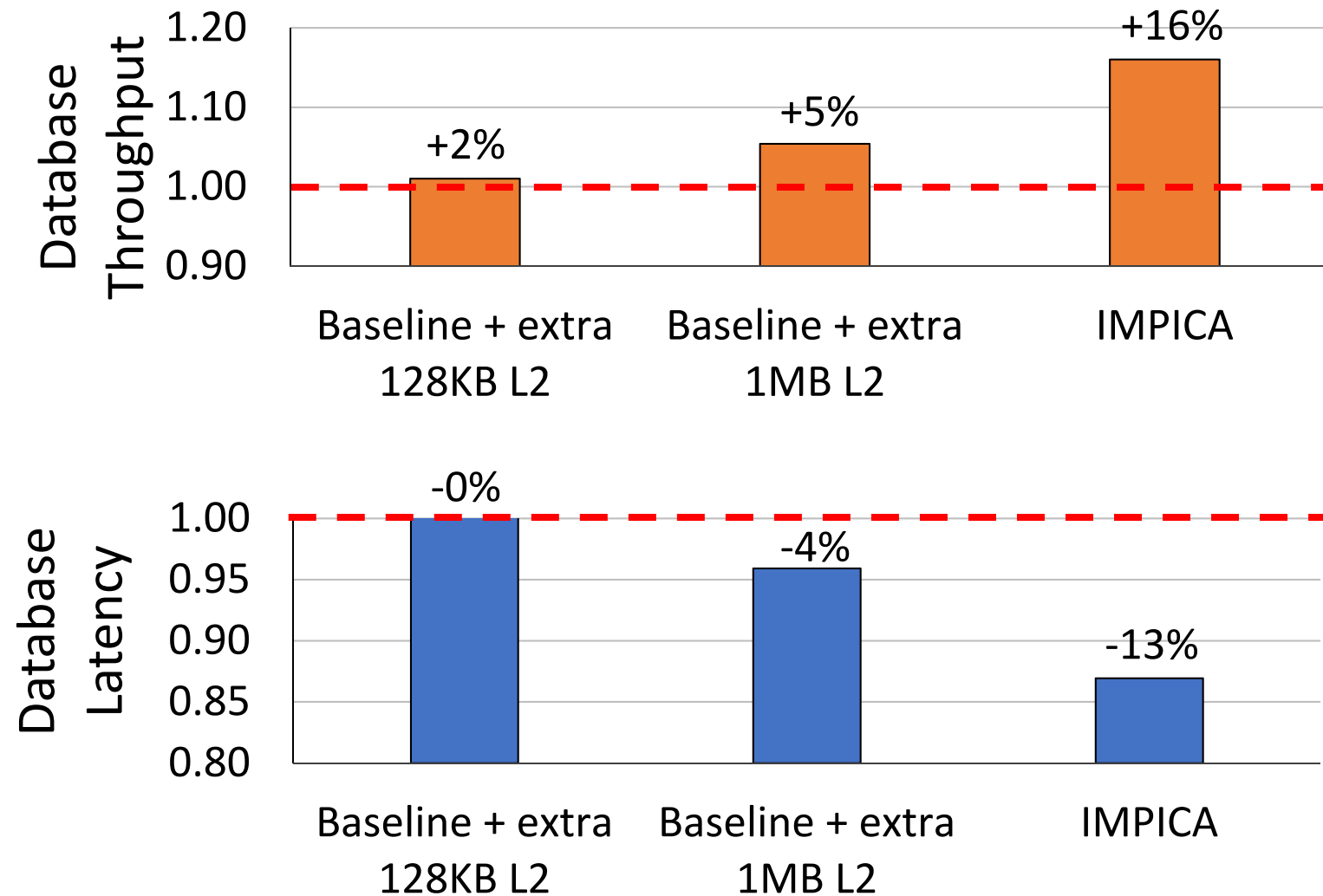
Evaluation Methodology

- 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>

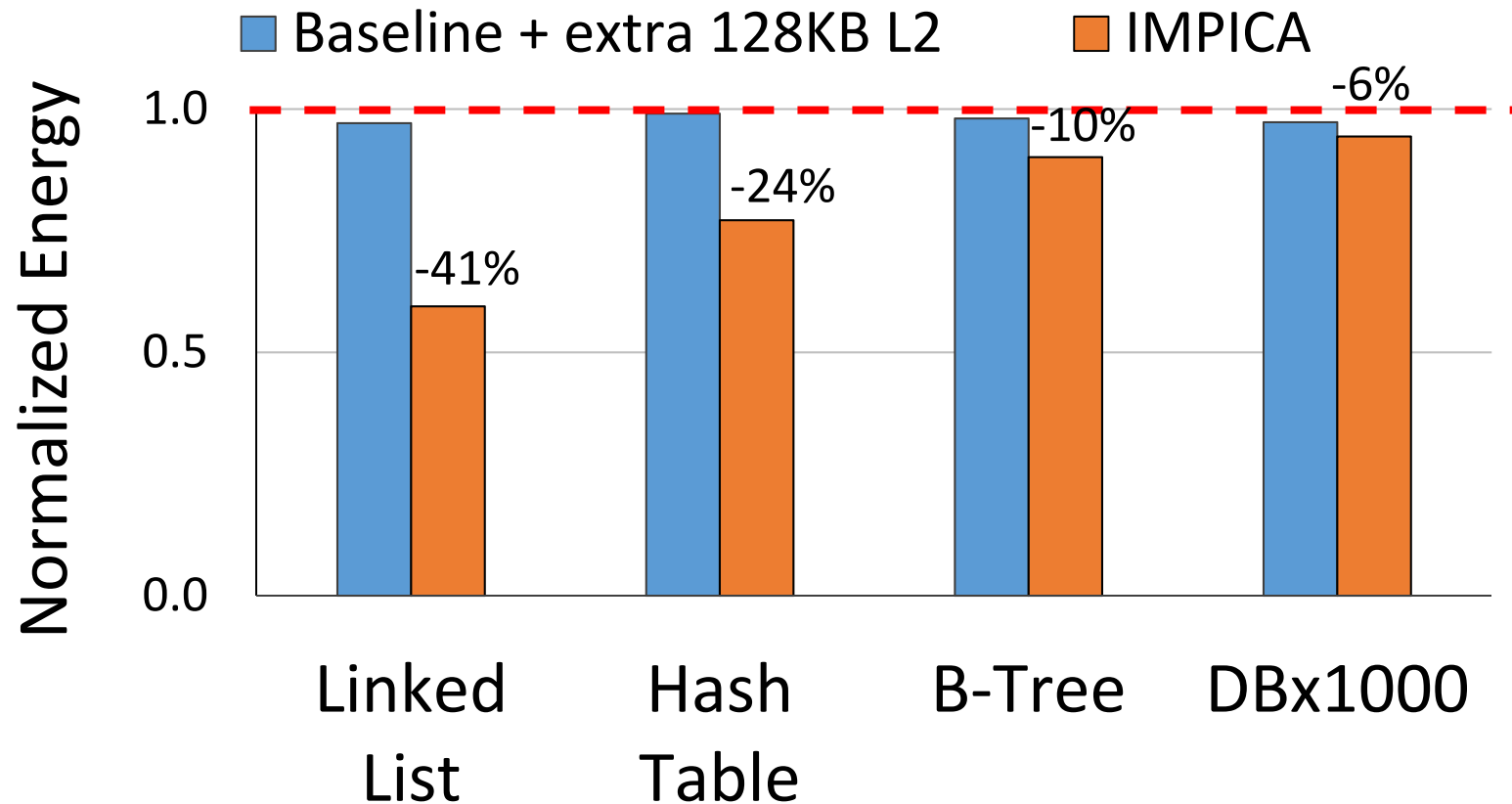
Result – Microbenchmark Performance



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%