# Memory Systems
## and Memory-Centric Computing Systems

## Part 3: Computation in Memory

Prof. Onur Mutlu

omutlu@gmail.com

https://people.inf.ethz.ch/omutlu

7 July 2019

SAMOS Tutorial

**SAFARI**　　**ETH**zürich　　**Carnegie Mellon**

# 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

# Three Key Systems Trends

## 1. Data access is a major bottleneck
- ❑ Applications are increasingly data hungry

## 2. Energy consumption is a key limiter

## 3. Data movement energy dominates compute
- ❑ Especially true for off-chip to on-chip movement

# Observation and Opportunity

- **High latency and high energy caused by data movement**
  - Long, energy-hungry interconnects
  - Energy-hungry electrical interfaces
  - Movement of large amounts of data

- Opportunity: Minimize data movement by performing computation directly (near) where the data resides
  - Processing in memory (PIM)
  - In-memory computation/processing
  - Near-data processing (NDP)
  - General concept applicable to any data storage & movement unit (caches, SSDs, main memory, network, controllers)
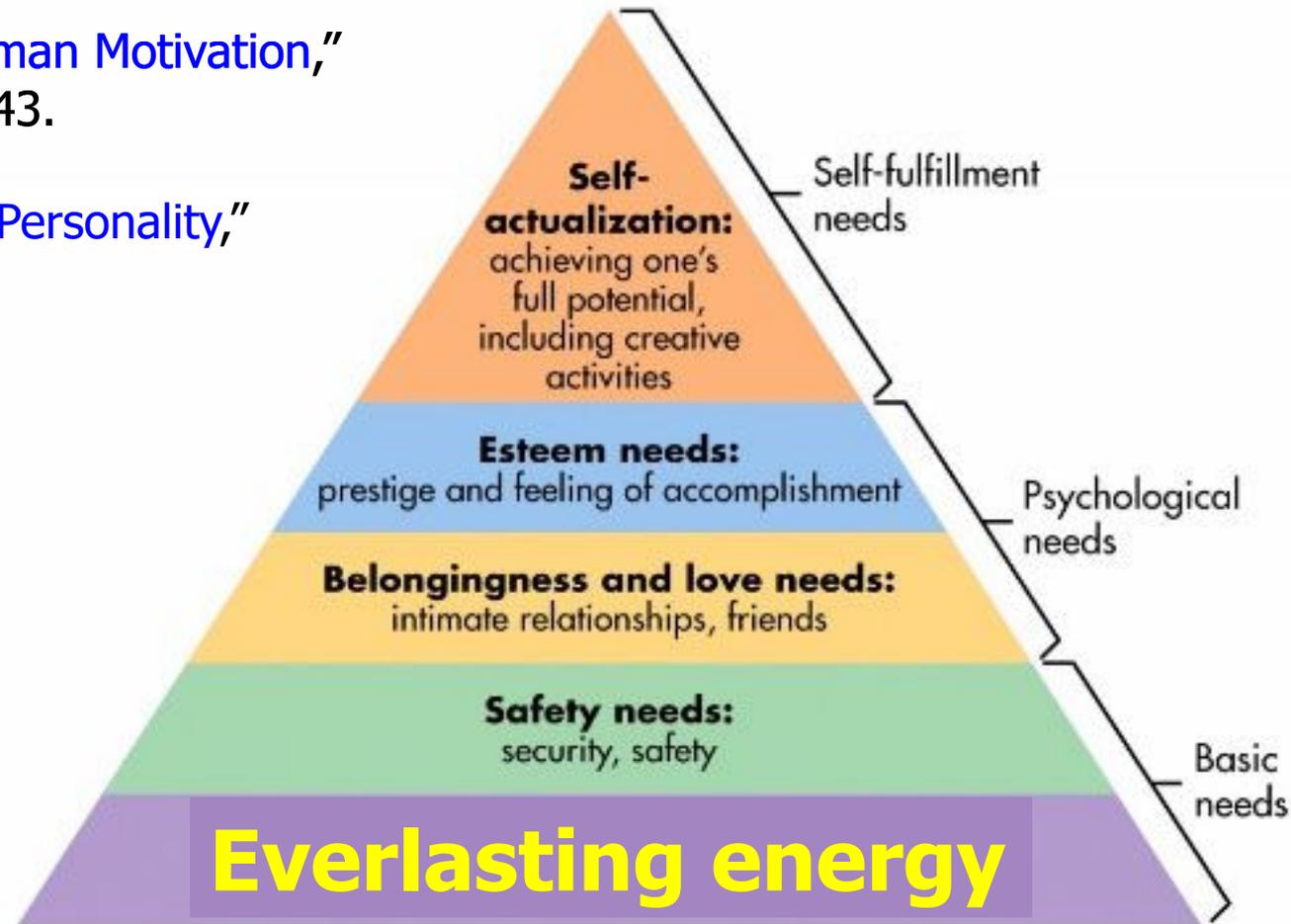
**SAFARI**

# Four Key Issues in Future Platforms

- Fundamentally Secure/Reliable/Safe Architectures

- Fundamentally Energy-Efficient Architectures
  - Memory-centric (Data-centric) Architectures

- Fundamentally Low-Latency Architectures

- Architectures for Genomics, Medicine, Health

**SAFARI**

# Maslow's (Human) Hierarchy of Needs, Revisited

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

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



Pyramid diagram of Maslow's Hierarchy of Needs:

- **Self-actualization:** achieving one's full potential, including creative activities — Self-fulfillment needs
- **Esteem needs:** prestige and feeling of accomplishment — Psychological needs
- **Belongingness and love needs:** intimate relationships, friends
- **Safety needs:** security, safety — Basic needs
- **Everlasting energy**

Source: https://www.simplypsychology.org/maslow.html

# Do We Want This?

Source: V. Milutinovic

# Or This?

**SAFARI** Source: V. Milutinovic

Challenge and Oppo

I expect that over the coming decade memory subsys-tem design will be the *only* important design issue for micro-processors.

# High Performance,
# Energy Efficient,
# Sustainable

# The Problem

Data access is the major performance and energy bottleneck

# Our current
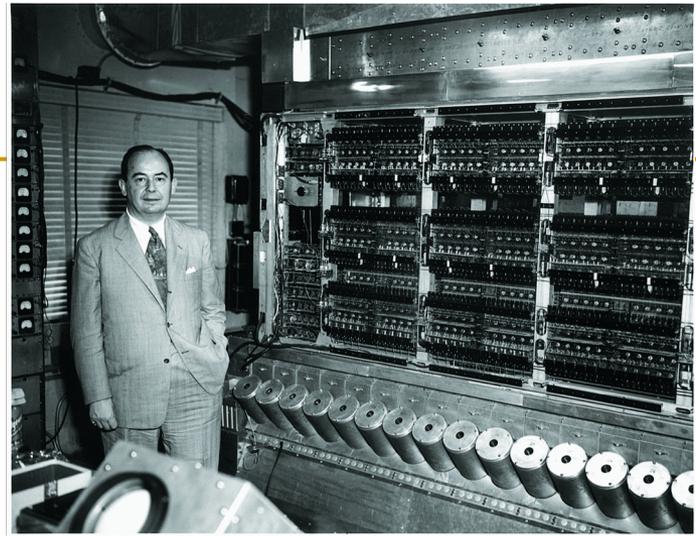# design principles
# cause great energy waste
# (and great performance loss)

**SAFARI**

# The Problem

## Processing of data
## is performed
## far away from the data

**SAFARI**

# A Computing System



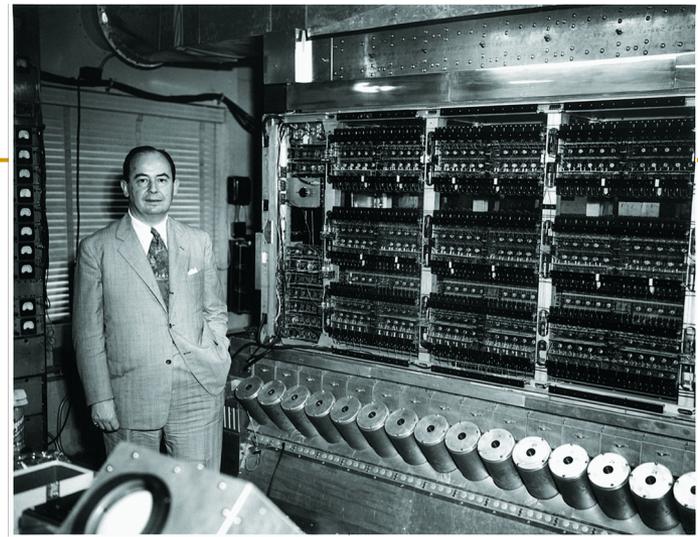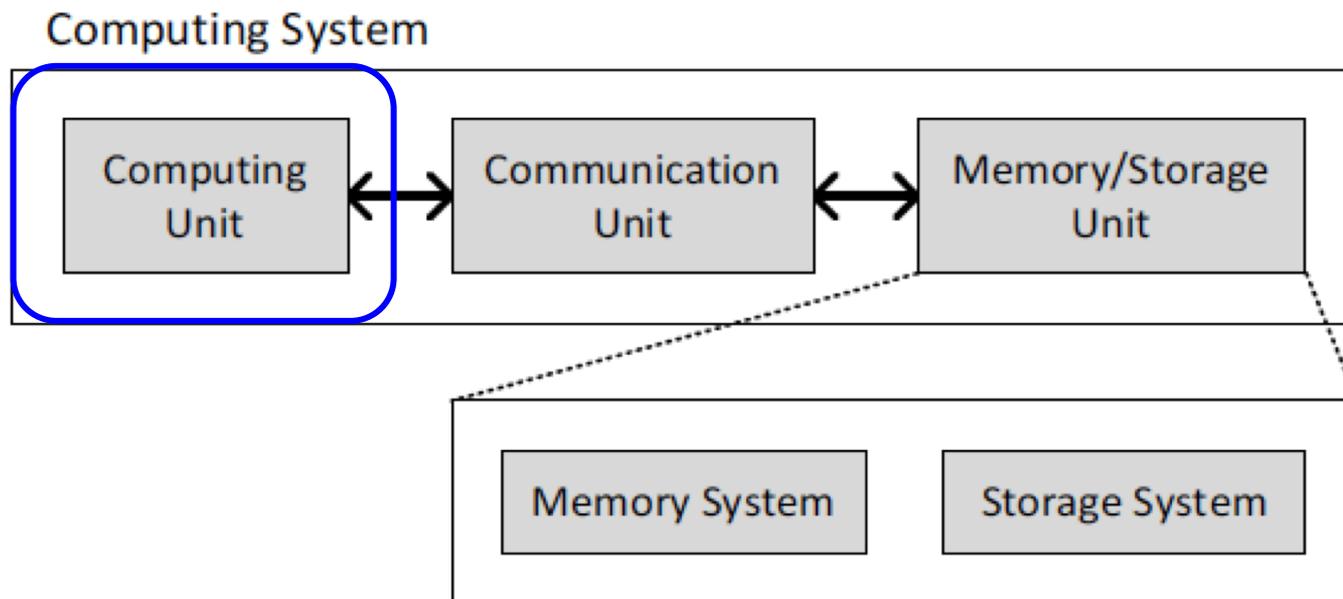- Three key components
- Computation
- Communication
- Storage/memory

Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.



Computing System

| Computing Unit | ↔ | Communication Unit | ↔ | Memory/Storage Unit |

Memory System          Storage System

# A Computing System



- Three key components
- Computation
- Communication
- Storage/memory

Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.



Computing System

Image source: https://lbsitbytes2010.wordpress.com/2013/03/29/john-von-neumann-roll-no-15/
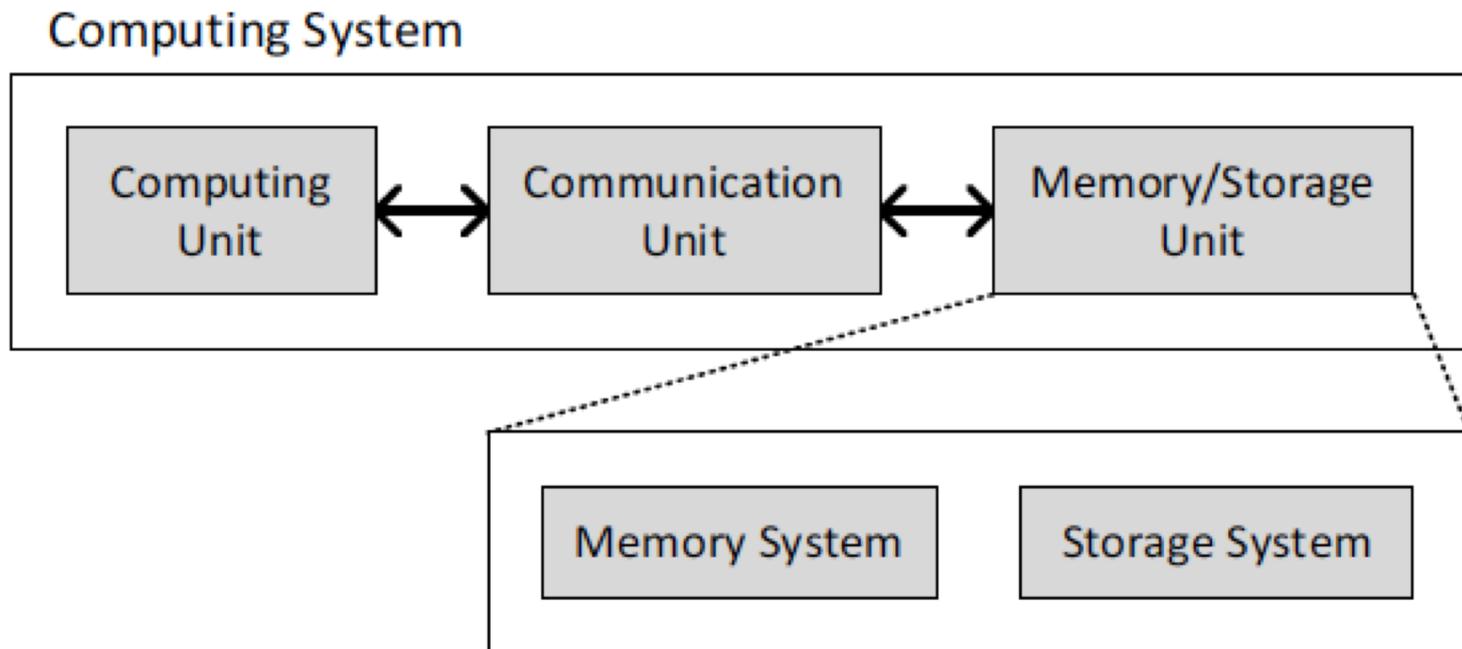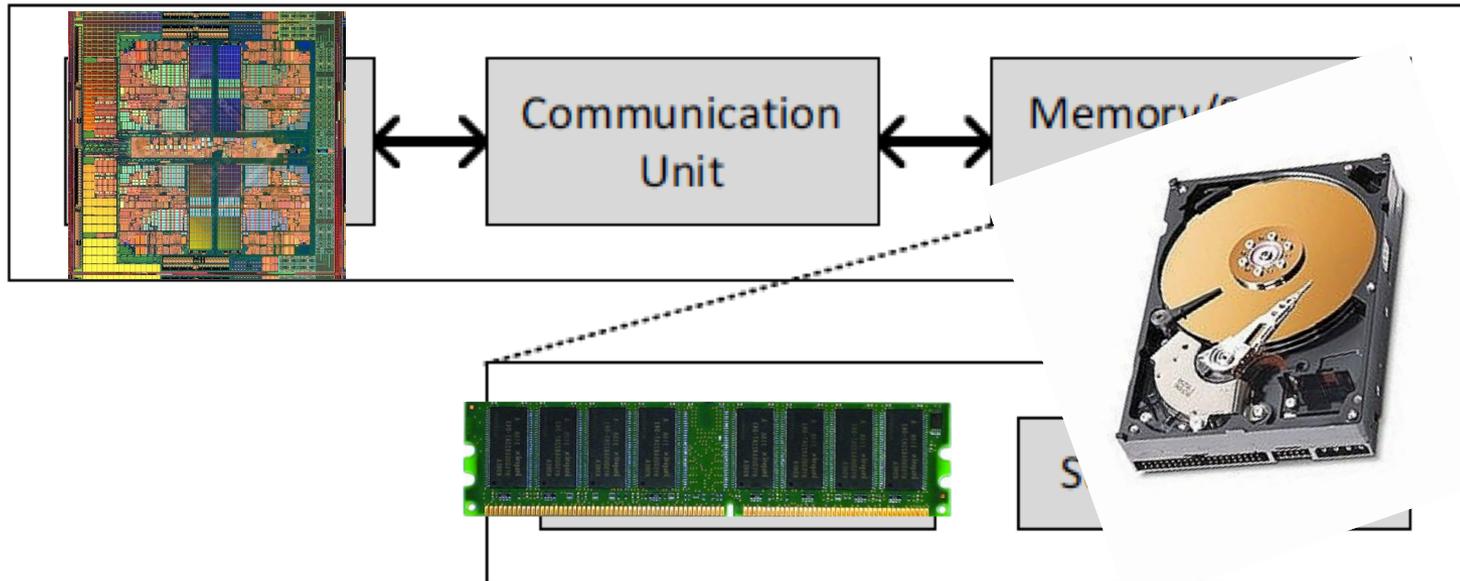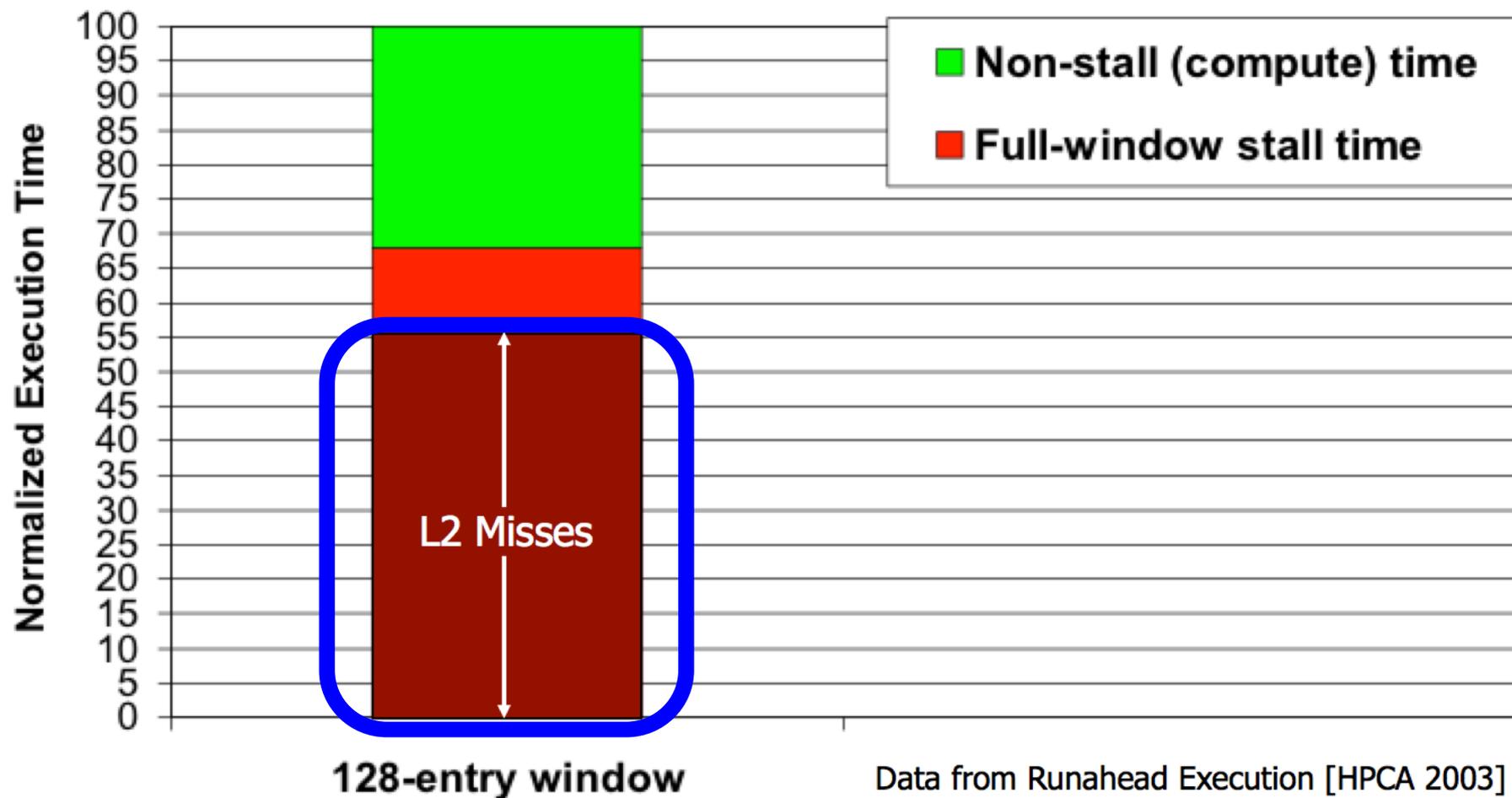
# Today's Computing Systems

- Are overwhelmingly processor centric
- All data processed in the processor → at great system cost
- Processor is heavily optimized and is considered the master
- Data storage units are dumb and are largely unoptimized (except for some that are on the processor die)

# Yet …

- **"It's the Memory, Stupid!"** (Richard Sites, MPR, 1996)



100 95 90 85 80 75 70 65 60 55 50 45 40 35 30 25 20 15 10 5 0

Normalized Execution Time

- Non-stall (compute) time
- Full-window stall time

L2 Misses

128-entry window

Data from Runahead Execution [HPCA 2003]

Mutlu+, "Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-Order Processors," HPCA 2003.

# The Performance Perspective

- Onur Mutlu, Jared Stark, Chris Wilkerson, and Yale N. Patt,
**"Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors"**
*Proceedings of the 9th International Symposium on High-Performance Computer Architecture* (**HPCA**), pages 129-140, Anaheim, CA, February 2003. Slides (pdf)

## Runahead Execution: An Alternative to Very Large Instruction Windows for Out-of-order Processors

Onur Mutlu §    Jared Stark †    Chris Wilkerson ‡    Yale N. Patt §

§ECE Department
The University of Texas at Austin
{onur,patt}@ece.utexas.edu

†Microprocessor Research
Intel Labs
jared.w.stark@intel.com

‡Desktop Platforms Group
Intel Corporation
chris.wilkerson@intel.com

# The Performance Perspective (Today)

- All of Google's Data Center Workloads (2015):



Kanev+, "Profiling a Warehouse-Scale Computer," ISCA 2015.
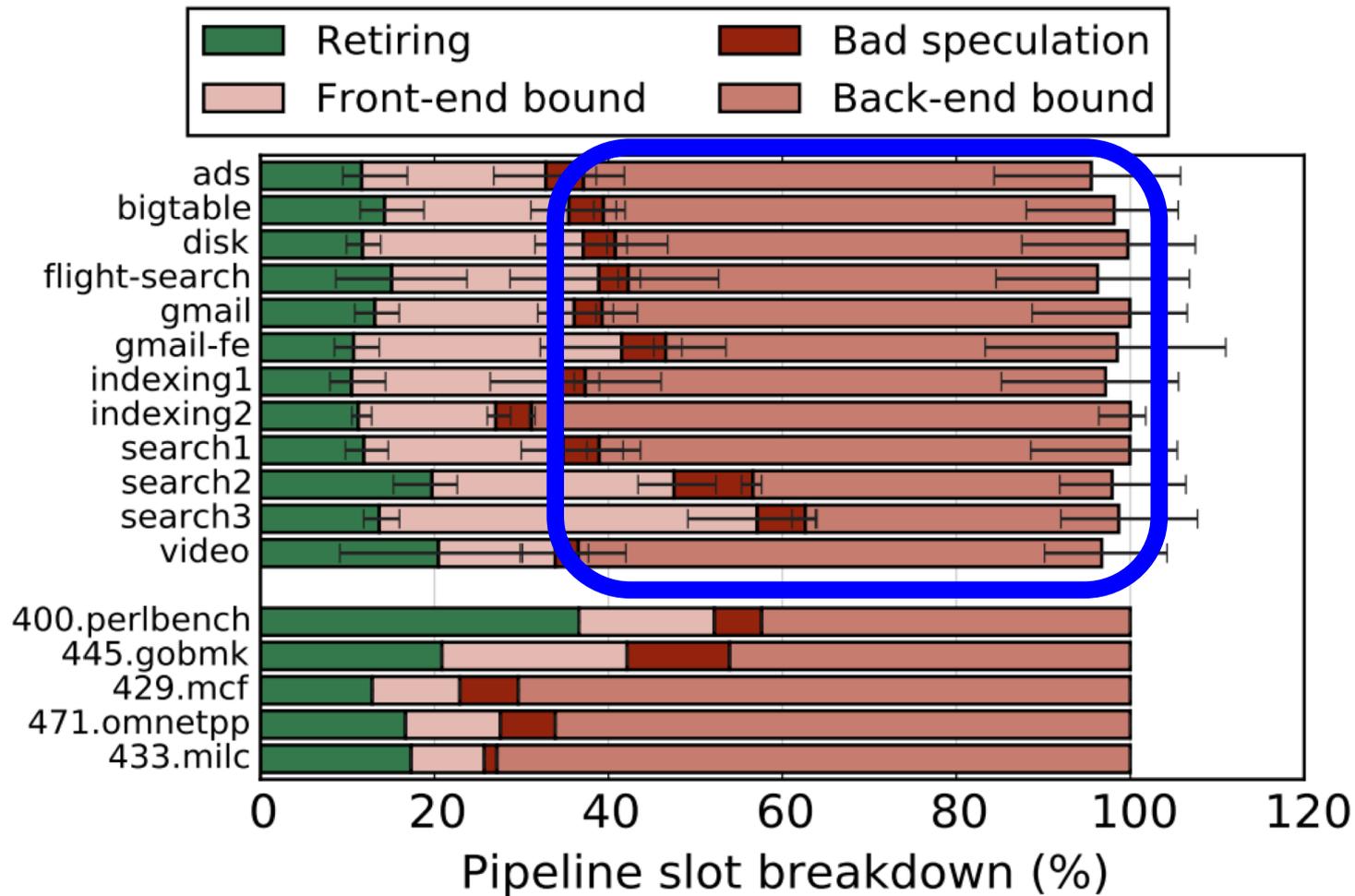
# The Performance Perspective (Today)
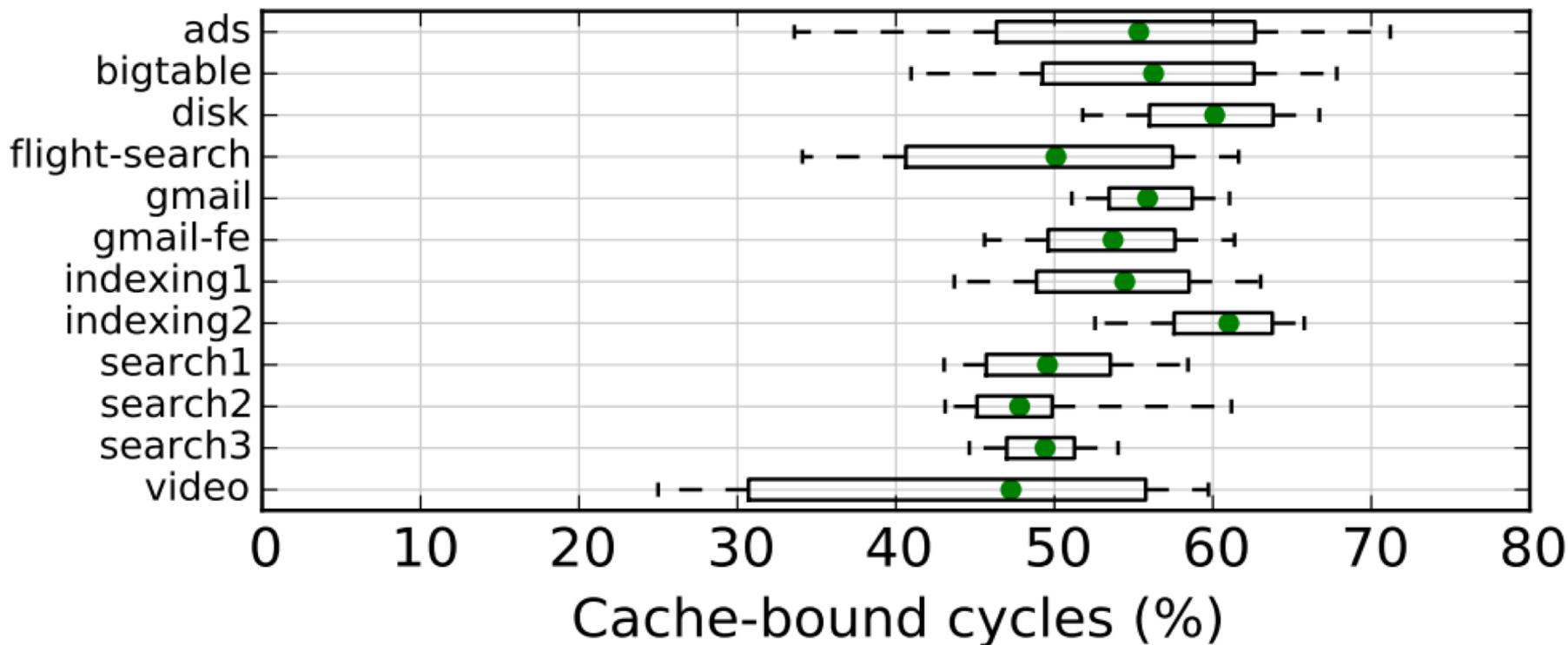
- All of Google's Data Center Workloads (2015):



Figure 11: Half of cycles are spent stalled on caches.

# Perils of Processor-Centric Design

- Grossly-imbalanced systems
    - Processing done only in **one place**
    - Everything else just stores and moves data: **data moves a lot**
    - → Energy inefficient
    - → Low performance
    - → Complex

- Overly complex and bloated processor (and accelerators)
    - To tolerate data access from memory
    - Complex hierarchies and mechanisms
    - → Energy inefficient
    - → Low performance
    - → Complex
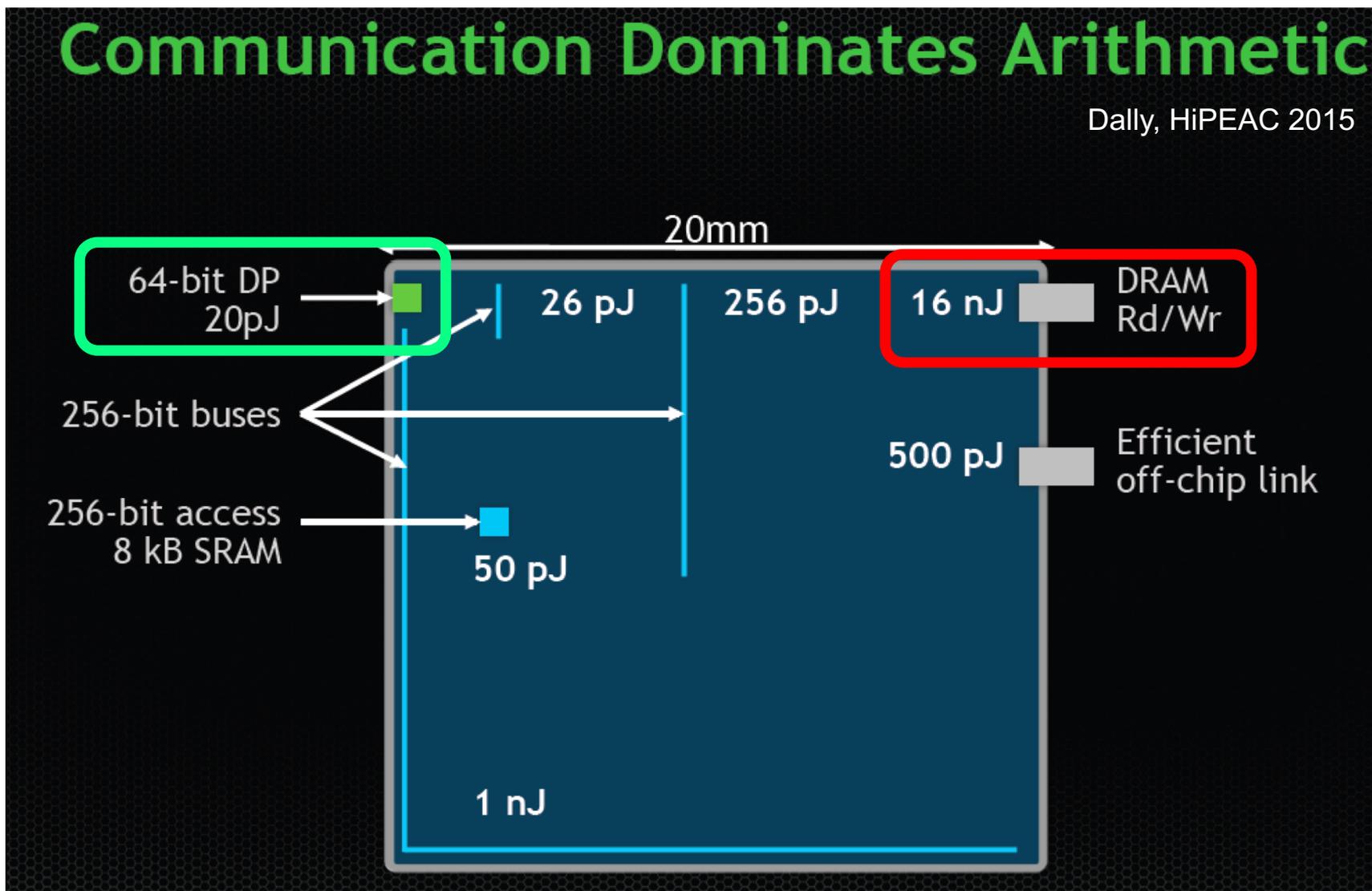
# Perils of Processor-Centric Design
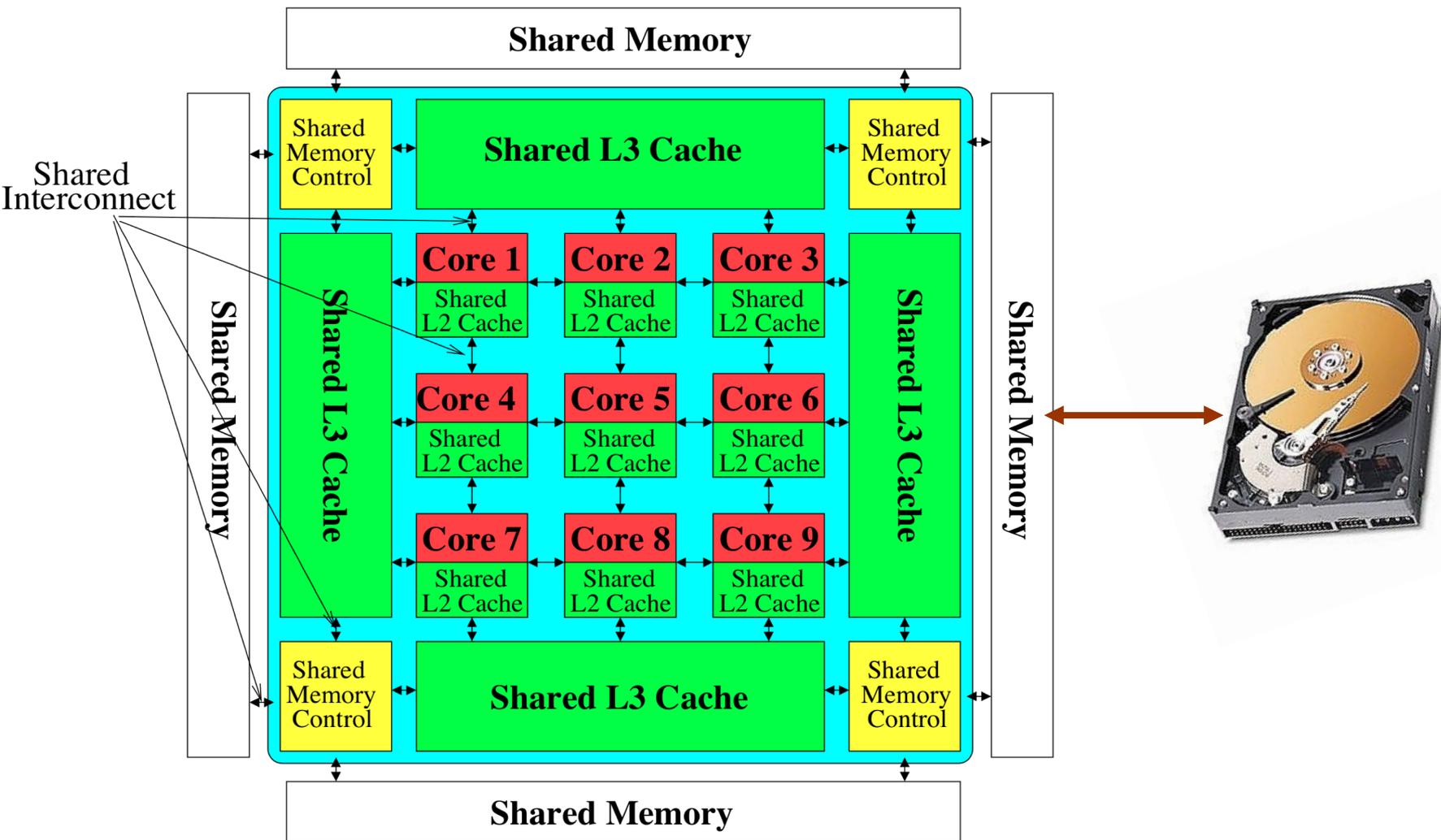


**Most of the system is dedicated to storing and moving data**

# The Energy Perspective
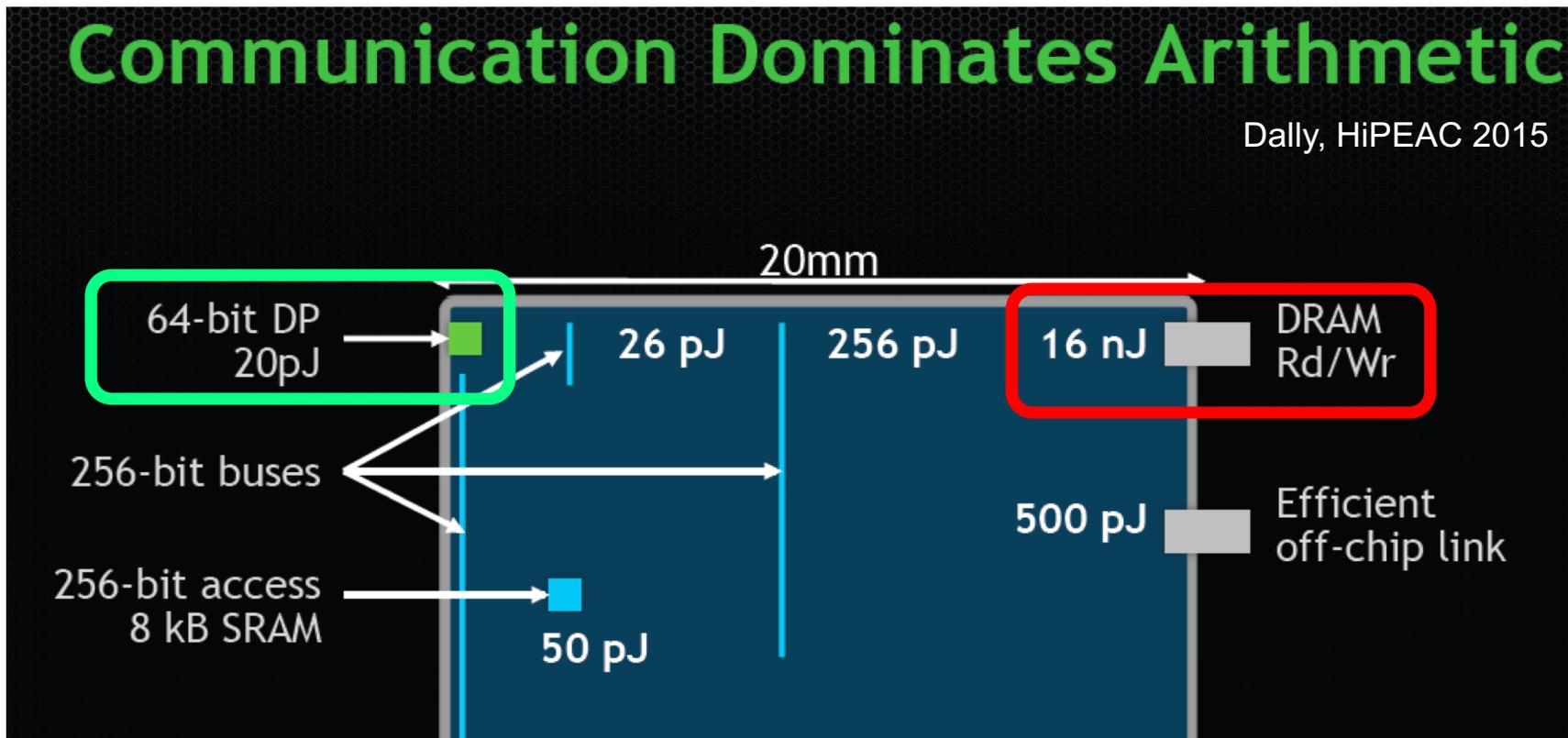


Communication Dominates Arithmetic

Dally, HiPEAC 2015

20mm

64-bit DP 20pJ — 26 pJ — 256 pJ — 16 nJ DRAM Rd/Wr

256-bit buses

256-bit access 8 kB SRAM — 50 pJ

500 pJ Efficient off-chip link

1 nJ

# Data Movement vs. Computation Energy

## Communication Dominates Arithmetic

Dally, HiPEAC 2015

20mm

64-bit DP
20pJ

26 pJ    256 pJ    16 nJ    DRAM Rd/Wr

256-bit buses

500 pJ    Efficient off-chip link

256-bit access
8 kB SRAM

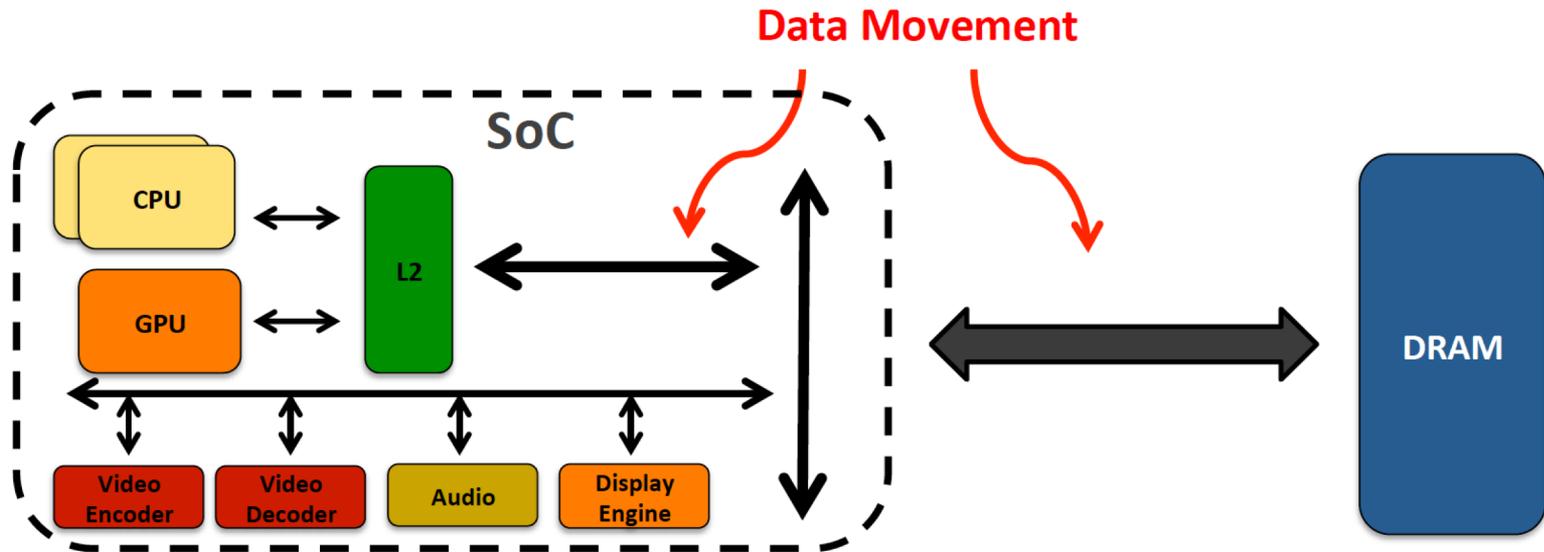50 pJ

A memory access consumes ~1000X the energy of a complex addition

# Data Movement vs. Computation Energy

- **Data movement** is a major system energy bottleneck
  - Comprises 41% of mobile system energy during web browsing [2]
  - Costs ~115 times as much energy as an ADD operation [1, 2]



[1]: Reducing data Movement Energy via Online Data Clustering and Encoding (MICRO'16)

[2]: Quantifying the energy cost of data movement for emerging smart phone workloads on mobile platforms (IISWC'14)

# Energy Waste in Mobile Devices

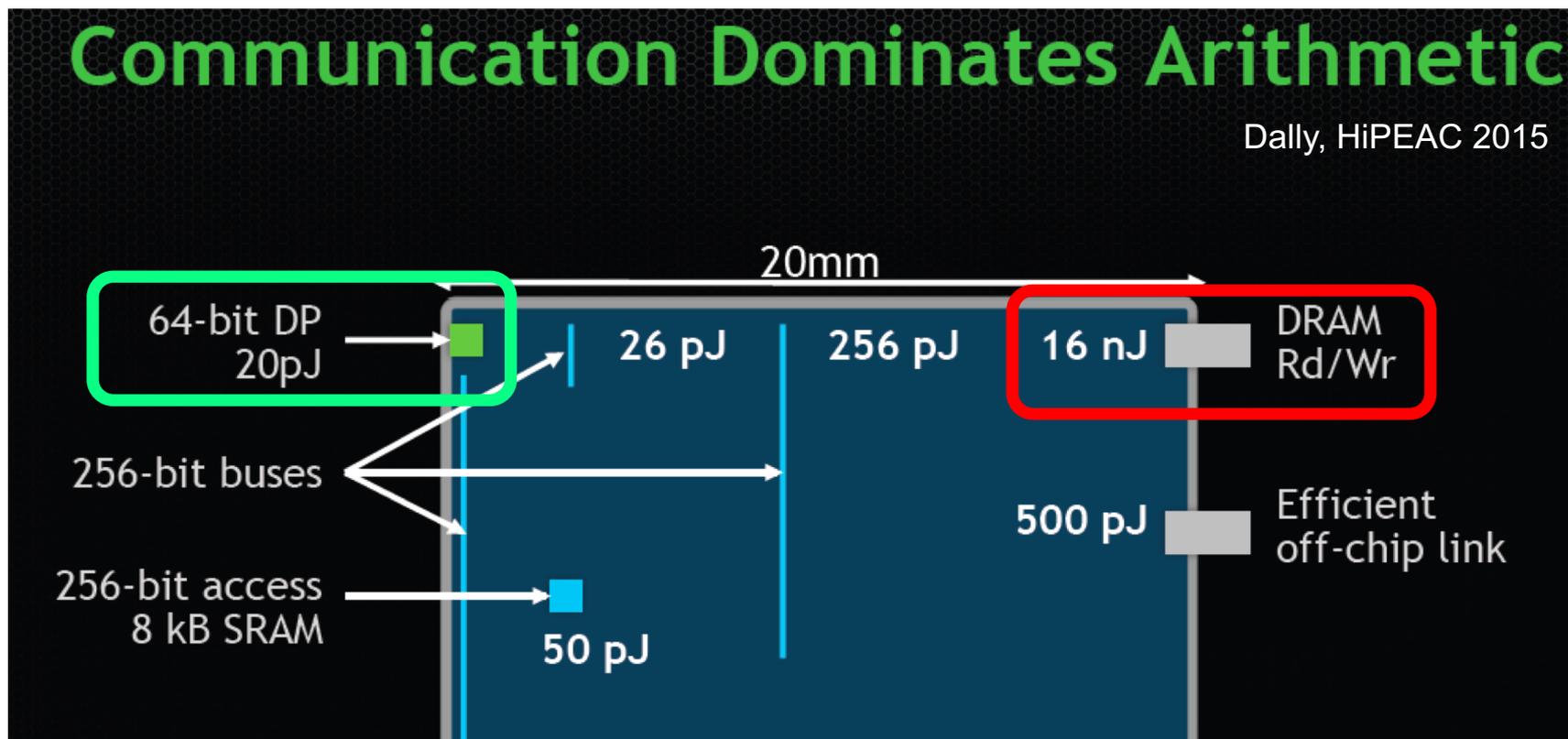- Amirali Boroumand, Saugata Ghose, Youngsok Kim, Rachata Ausavarungnirun, Eric Shiu, Rahul Thakur, Daehyun Kim, Aki Kuusela, Allan Knies, Parthasarathy Ranganathan, and Onur Mutlu,
**"Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks"**
*Proceedings of the 23rd International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

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

## Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

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

**SAFARI**

# We Do Not Want to Move Data!

## Communication Dominates Arithmetic

Dally, HiPEAC 2015

20mm

64-bit DP
20pJ

26 pJ    256 pJ    16 nJ    DRAM Rd/Wr

256-bit buses

256-bit access
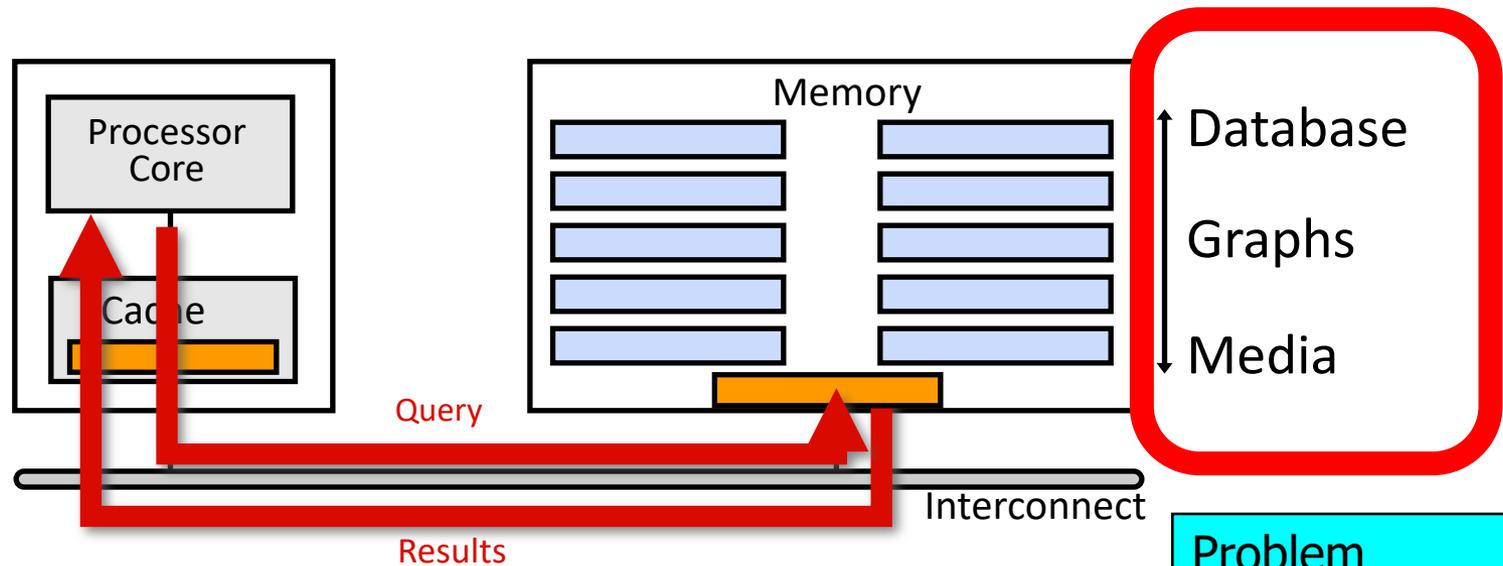8 kB SRAM

50 pJ

500 pJ    Efficient off-chip link

**A memory access consumes ~1000X the energy of a complex addition**

# We Need A Paradigm Shift To …

- Enable computation with minimal data movement

- Compute where it makes sense (where data resides)

- Make computing architectures more data-centric
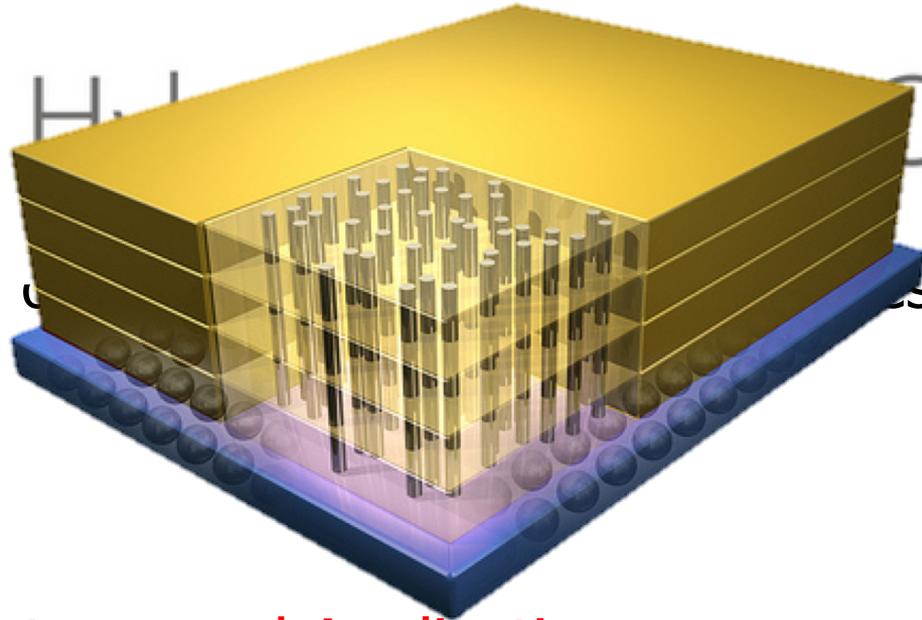
# Goal: Processing Inside Memory



- Many questions … How do we design the:
  - compute-capable memory & controllers?
  - processor chip and in-memory units?
  - software and hardware interfaces?
  - system software and languages?
  - algorithms?

# Why In-Memory Computation Today?



Dally, HiPEAC 2

→ Industry

- **Pull from Systems and Applications**
  - ❑ Data access is a major system and application bottleneck
  - ❑ Systems are energy limited
  - ❑ Data movement much more energy-hungry than computation

# Sub-Agenda: In-Memory Computation

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

**SAFARI**

# Processing in Memory: Two Approaches

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

# Approach 1: Minimally Changing DRAM

- DRAM has great capability to perform bulk data movement and computation internally with small changes
  - Can exploit internal connectivity to move data
  - Can exploit analog computation capability
  - …

- Examples: RowClone, In-DRAM AND/OR, Gather/Scatter DRAM
  - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
  - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
  - Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses (Seshadri et al., MICRO 2015)
  - "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology" (Seshadri et al., MICRO 2017)

**SAFARI**

# Starting Simple: Data Copy and Initialization

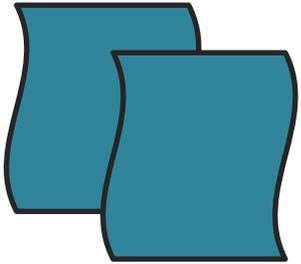**Bulk Data Copy**

src - - - -> dst
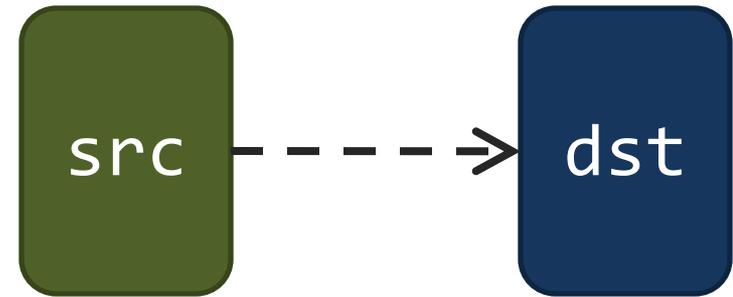
**Bulk Data Initialization**

val - - - - -> dst
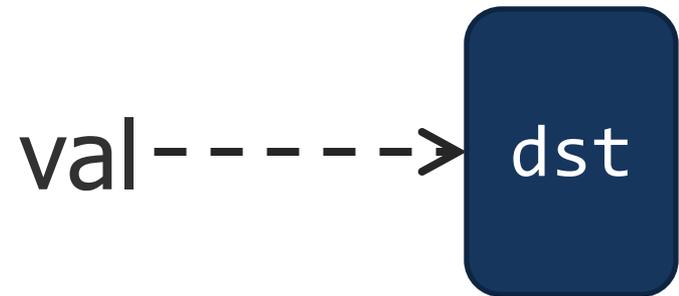
# Starting Simple: Data Copy and Initialization

*memmove & memcpy:* 5% cycles in Google's datacenter [Kanev+ ISCA'15]

**Forking**

00000
00000
00000

**Zero initialization**
**(e.g., security)**
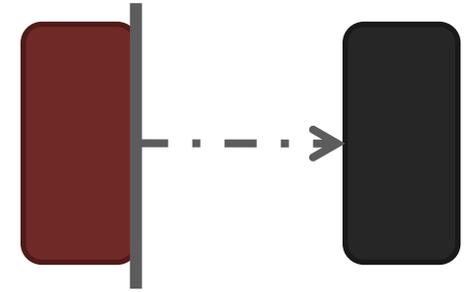
**Checkpointing**

**VM Cloning**
**Deduplication**
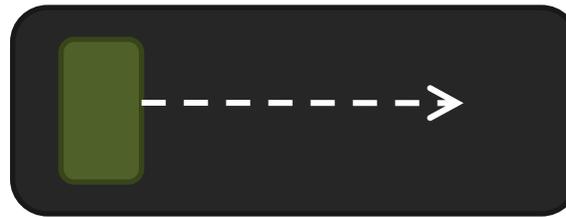
**Page Migration**

• • •
Many more

# Today's Systems: Bulk Data Copy

1) High latency

3) Cache pollution

**Memory**

**CPU**   **L1**   **L2**   **L3**   **MC**

2) High bandwidth utilization

4) Unwanted data movement
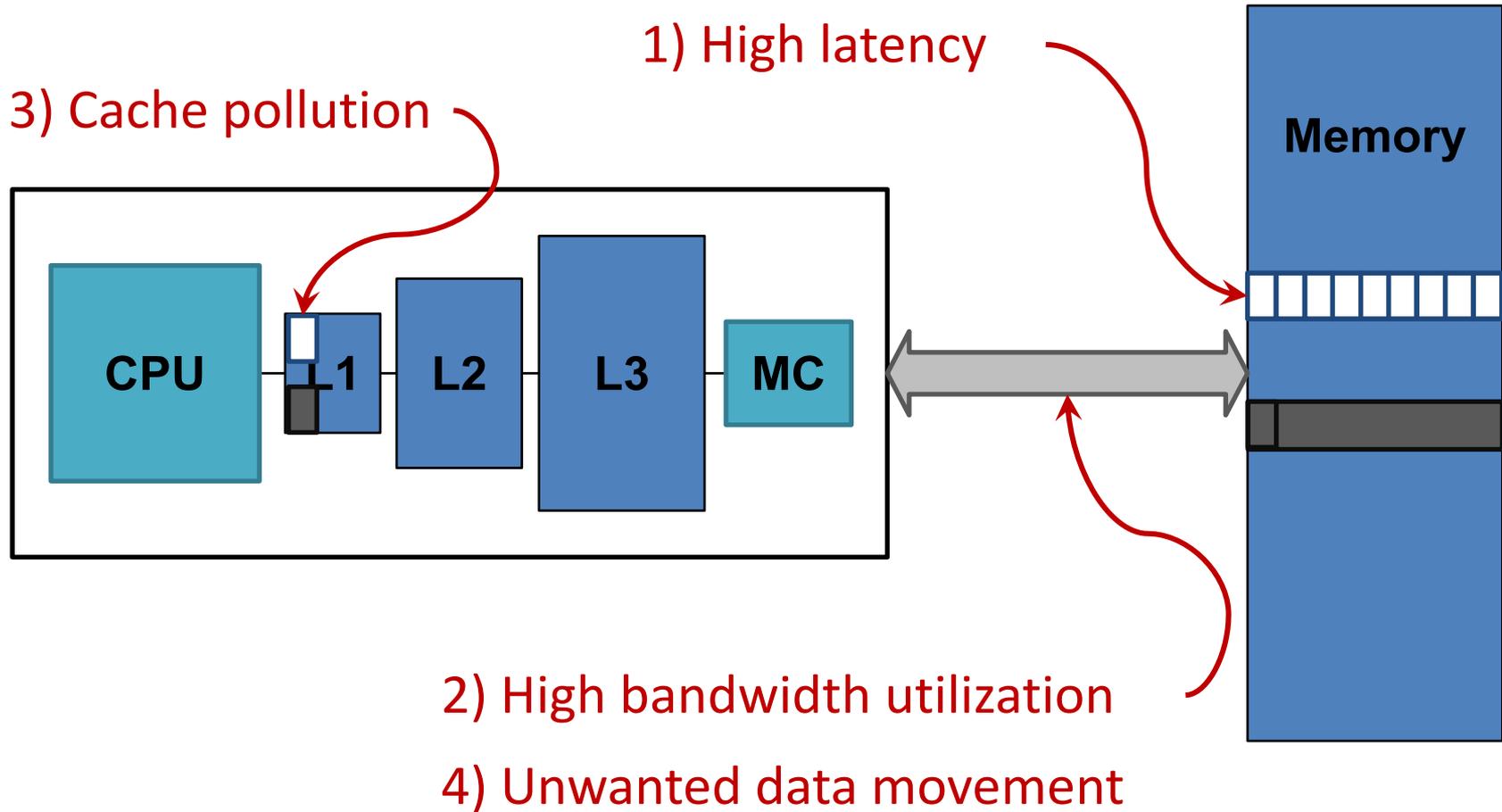
1046ns, 3.6uJ   (for 4KB page copy via DMA)

# Future Systems: In-Memory Copy

3) No cache pollution

1) Low latency

**Memory**

**CPU**  **L1**  **L2**  **L3**  **MC**

2) Low bandwidth utilization

4) No unwanted data movement
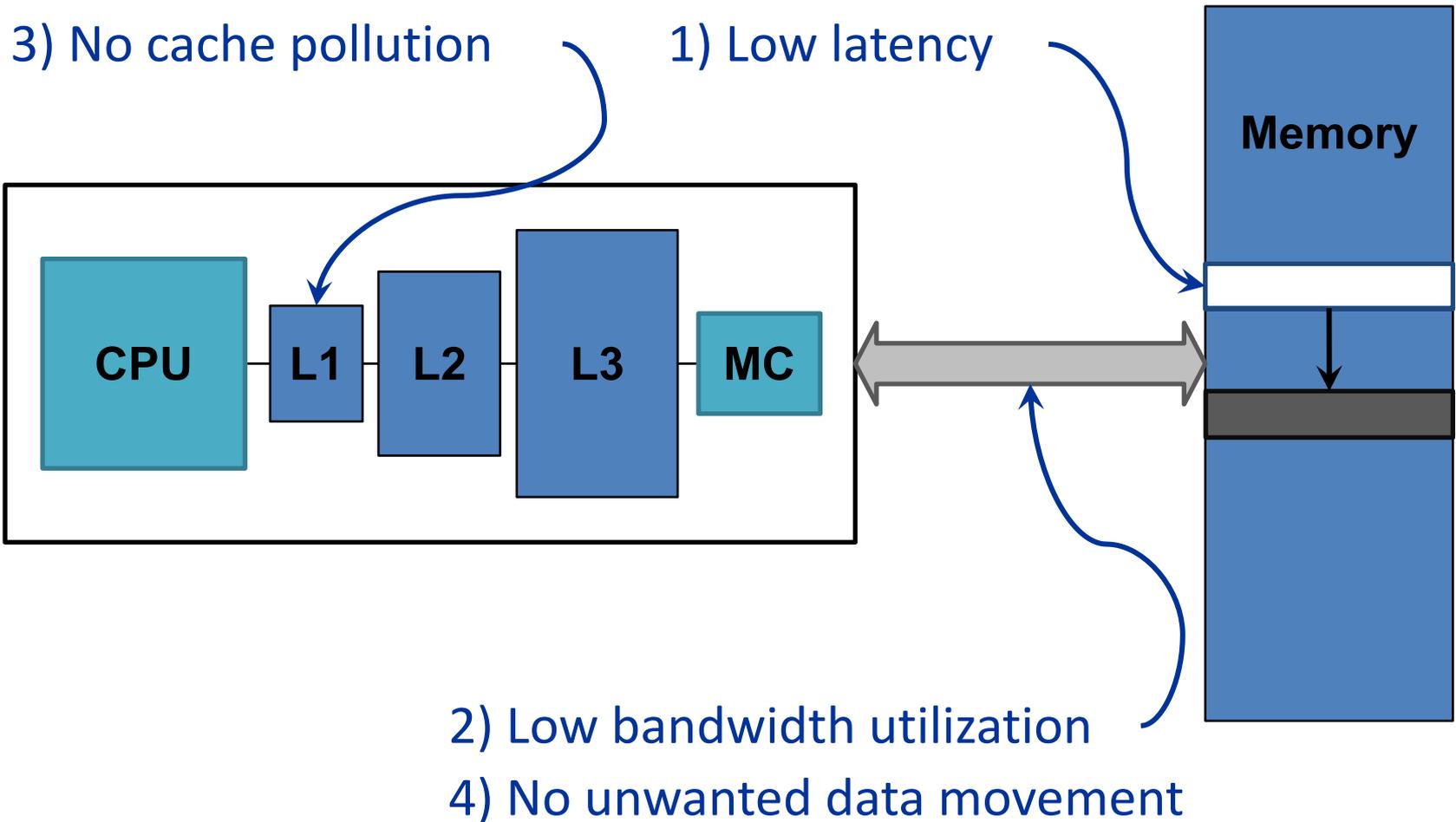
1046ns, 3.6uJ  →  90ns, 0.04uJ

# RowClone: In-DRAM Row Copy

**Idea: Two consecutive ACTivates**
**Negligible HW cost**

4 Kbytes

Step 1: Activate row A

Step 2: Activate row B

DRAM subarray

Transfer row

Transfer row

Row Buffer (4 Kbytes)

8 bits
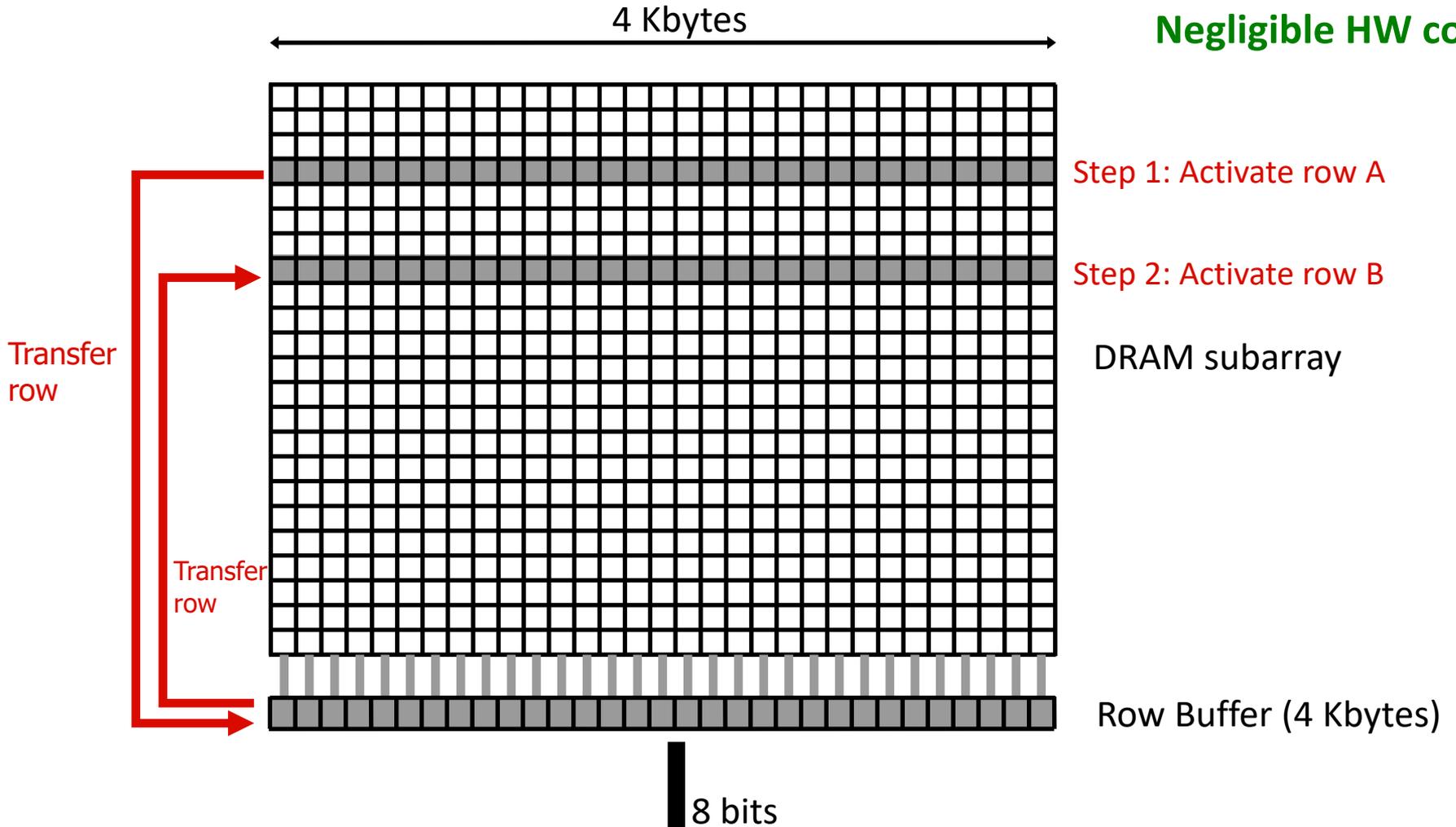
**11.6X** latency reduction, **74X** energy reduction

# RowClone: Inter-Bank
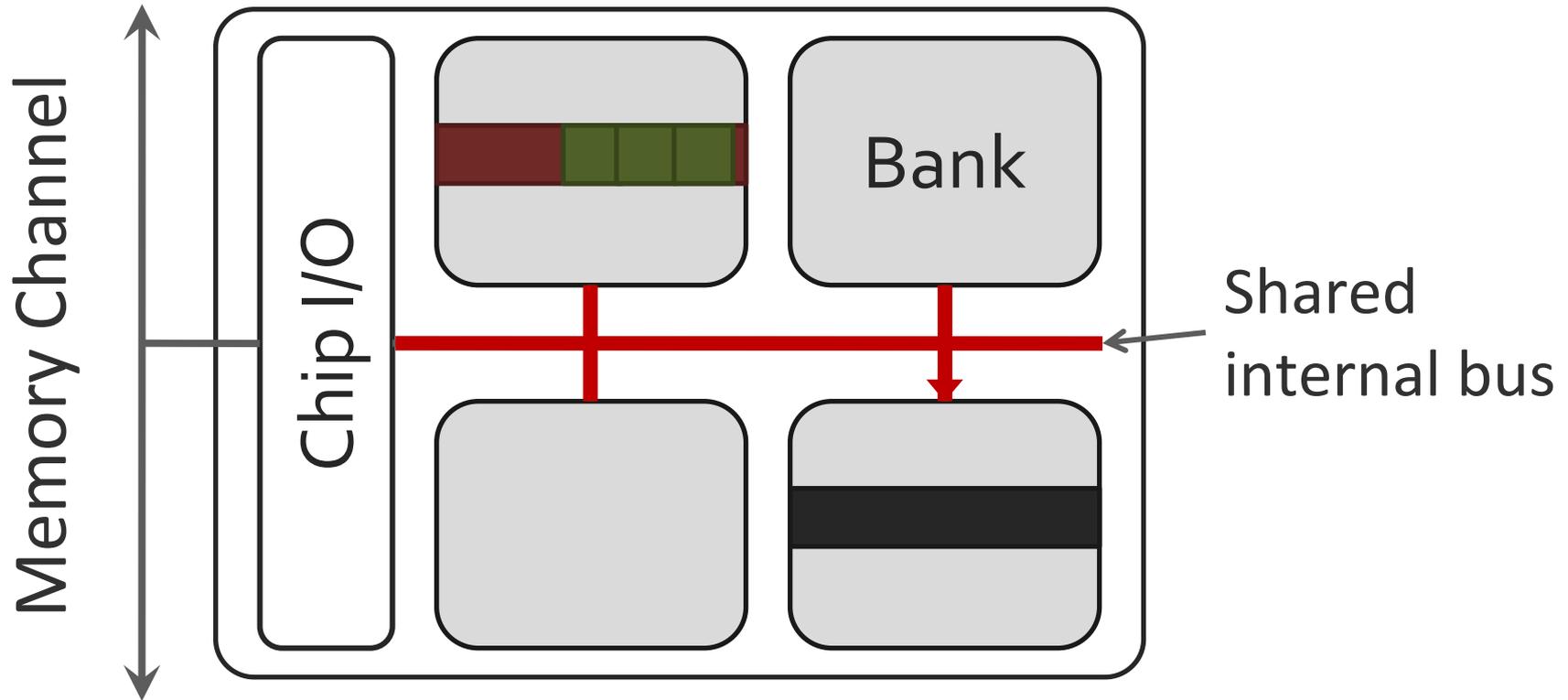


**Overlap the latency of the read and the write**
**1.9X latency reduction, 3.2X energy reduction**

# Generalized RowClone

**0.01% area cost**

Inter Subarray Copy
(Use Inter-Bank Copy Twice)

Subarray

Bank I/O

Memory Channel

Chip I/O

Bank

Inter Bank Copy
(Pipelined
Internal RD/WR)

Intra Subarray
Copy (2 ACTs)
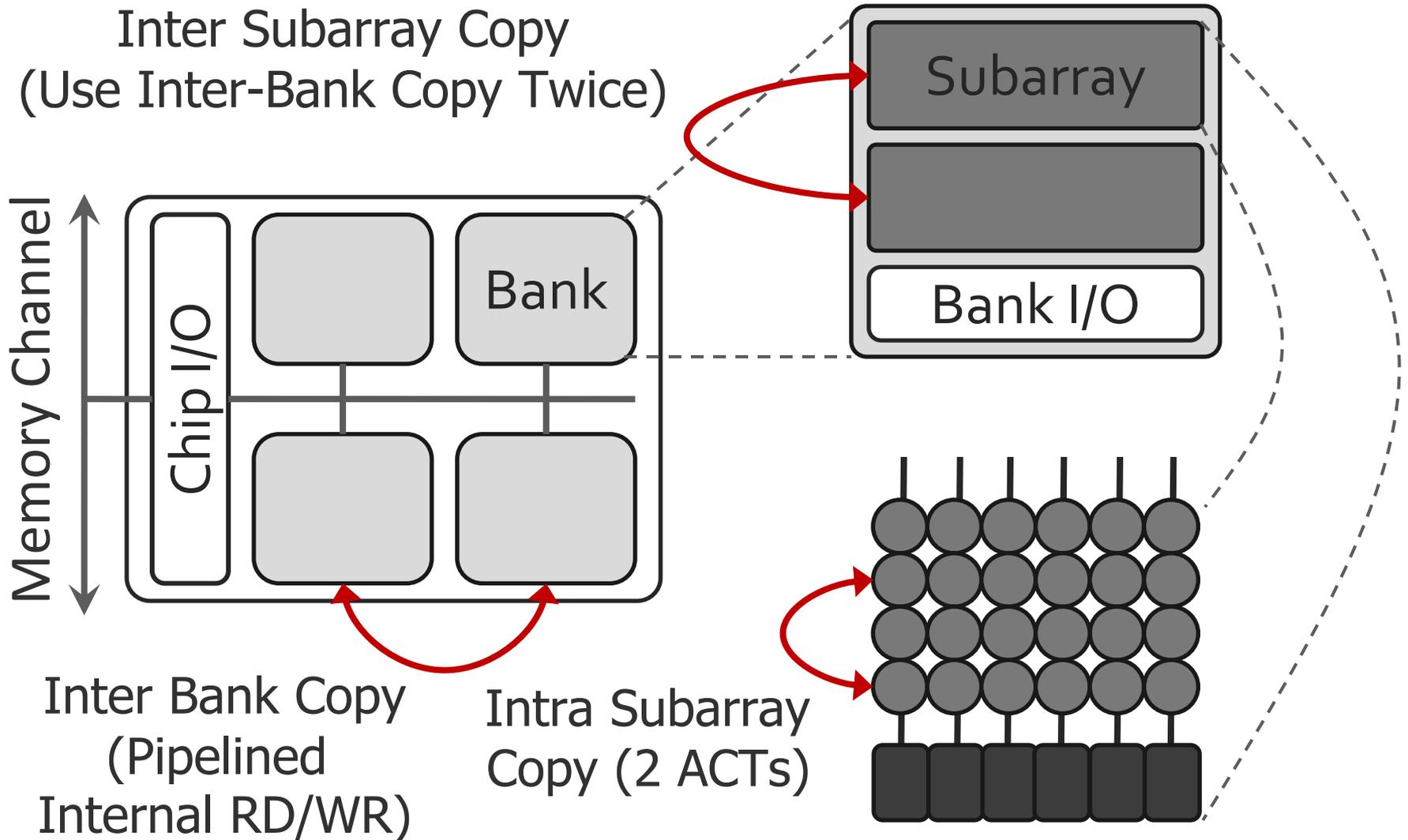
# RowClone: Latency and Energy Savings
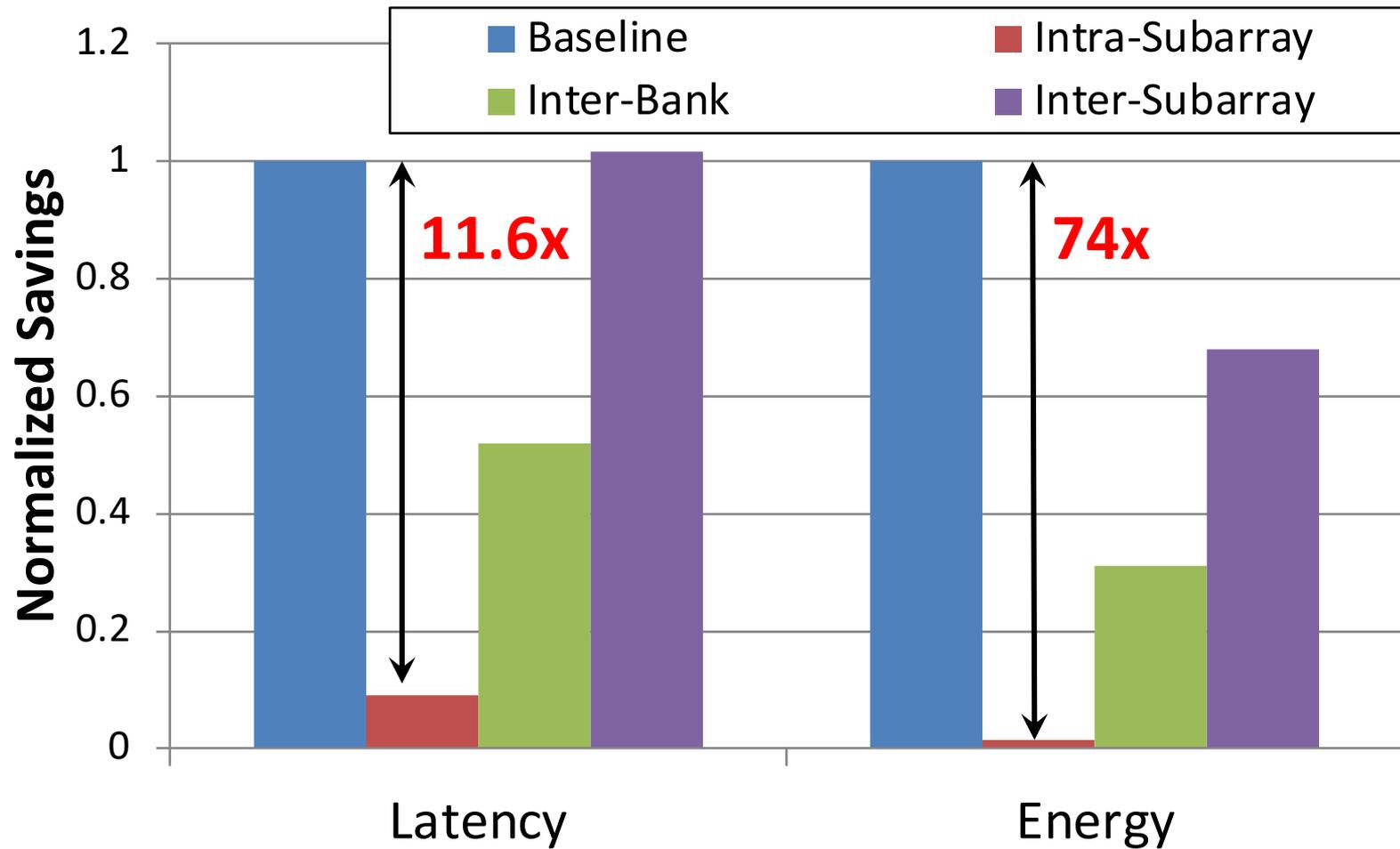


Seshadri et al., "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.

# 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
vseshadr@cs.cmu.edu

Yoongu Kim
yoongukim@cmu.edu

Chris Fallin*
cfallin@c1f.net

Donghyuk Lee
donghyuk1@cmu.edu

Rachata Ausavarungnirun
rachata@cmu.edu

Gennady Pekhimenko
gpekhime@cs.cmu.edu

Yixin Luo
yixinluo@andrew.cmu.edu

Onur Mutlu
onur@cmu.edu

Phillip B. Gibbons†
phillip.b.gibbons@intel.com

Michael A. Kozuch†
michael.a.kozuch@intel.com

Todd C. Mowry
tcm@cs.cmu.edu

Carnegie Mellon University    †Intel Pittsburgh

# 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



A

B

C

$\frac{1}{2}V_{DD}+\delta$

dis

$\frac{1}{2}V_{DD}$

**Final State**
*AB + BC + AC*

*C(A + B) +
~C(AB)*

Seshadri+, "Fast Bulk Bitwise AND and OR in DRAM", IEEE CAL 2015.

# 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

**SAFARI**

# In-DRAM NOT: Dual Contact Cell



d-wordline

dual-contact
cell (DCC)

n-wordline

bitline

sense
amplifier
enable

bitline

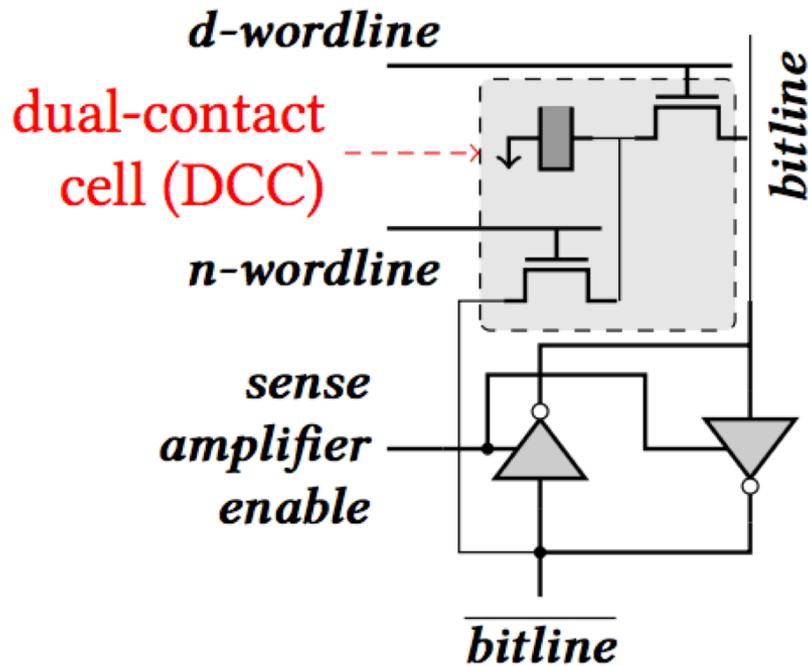**Figure 5: A dual-contact cell connected to both ends of a sense amplifier**

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.

# Performance: In-DRAM Bitwise Operations
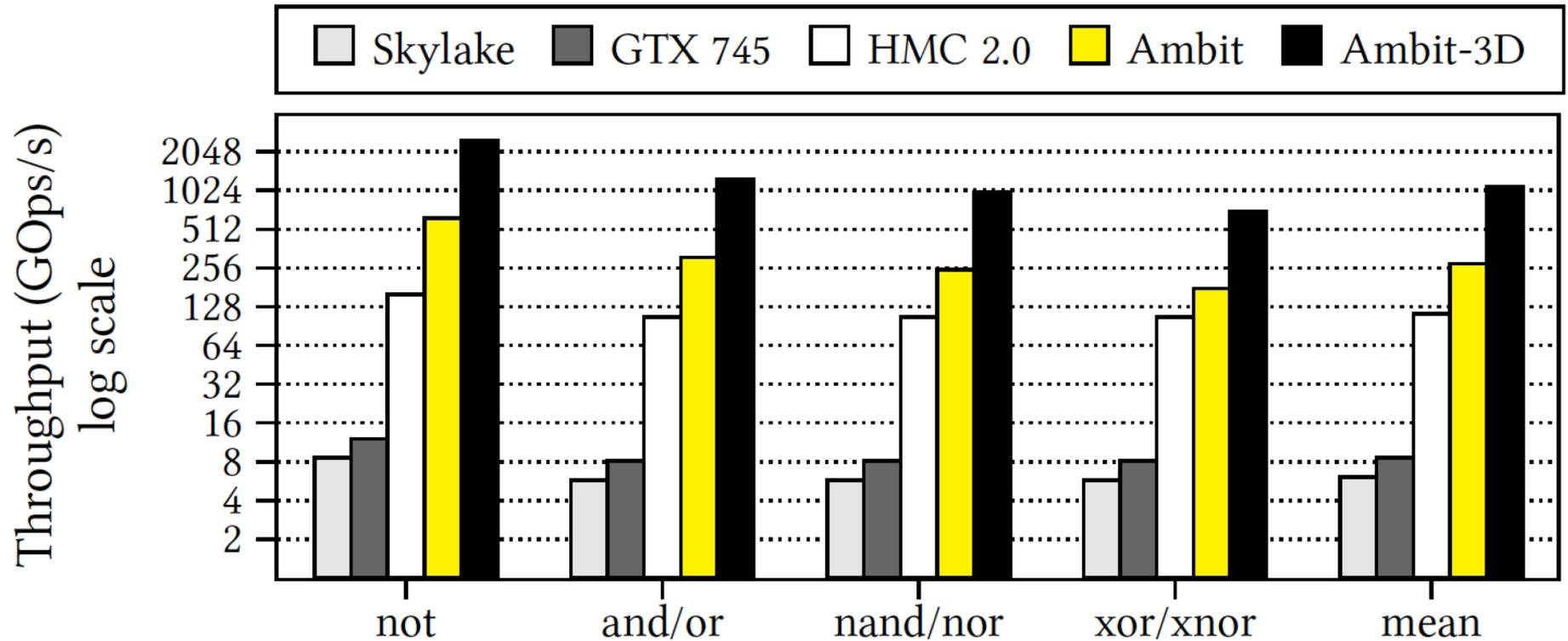


**Figure 9: Throughput of bitwise operations on various systems.**

# 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.

**SAFARI**

# Ambit vs. DDR3: Performance and Energy



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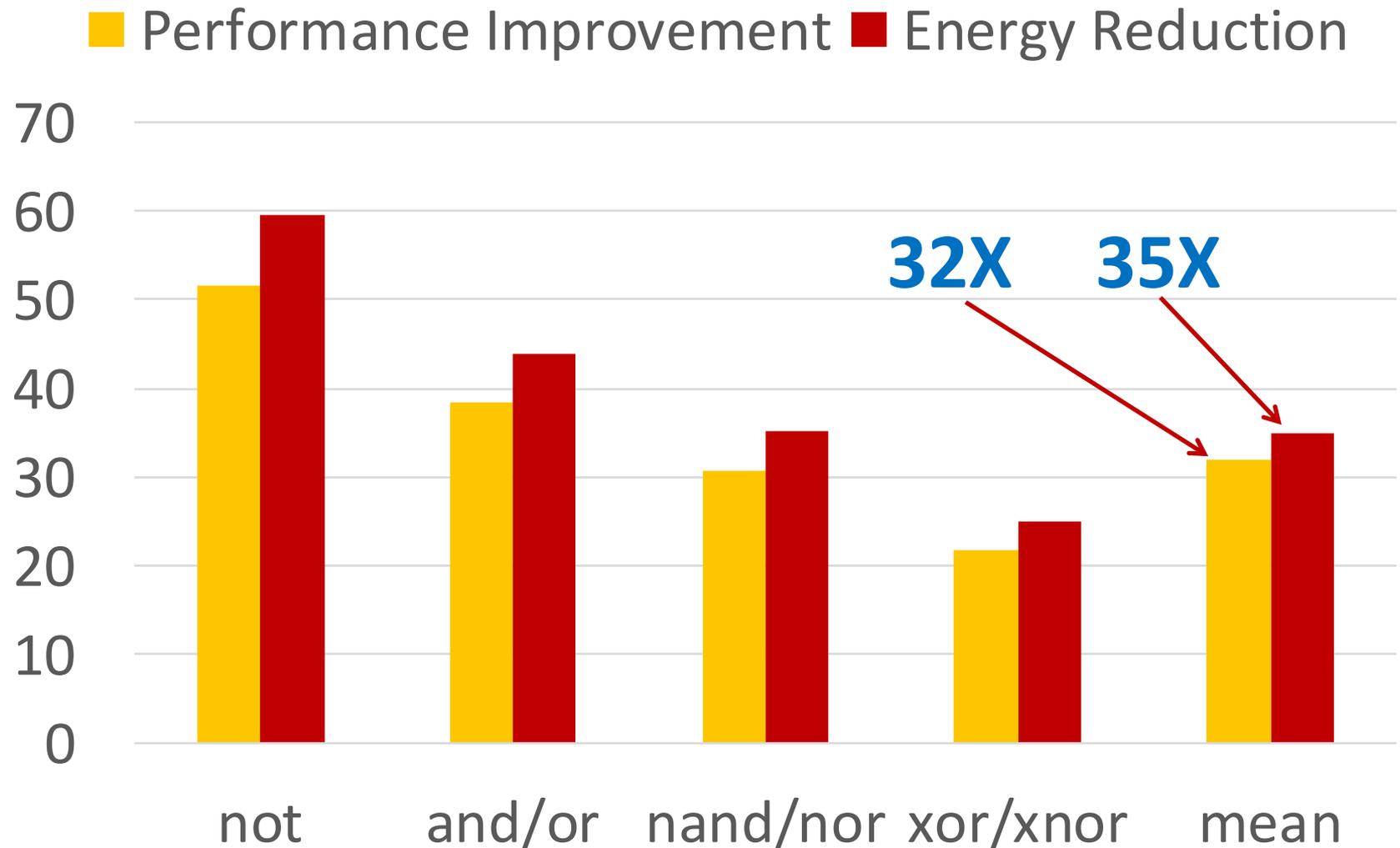
# Bulk Bitwise Operations in Workloads



**BitWeaving**
(database queries)

**Bitmap indices**
(database indexing)

**BitFunnel**
(web search)

**Bulk Bitwise Operations**

**Set operations**

**DNA sequence mapping**

**Encryption algorithms**

...

[1] Li and Patel, BitWeaving, SIGMOD 2013
[2] Goodwin+, BitFunnel, SIGIR 2017
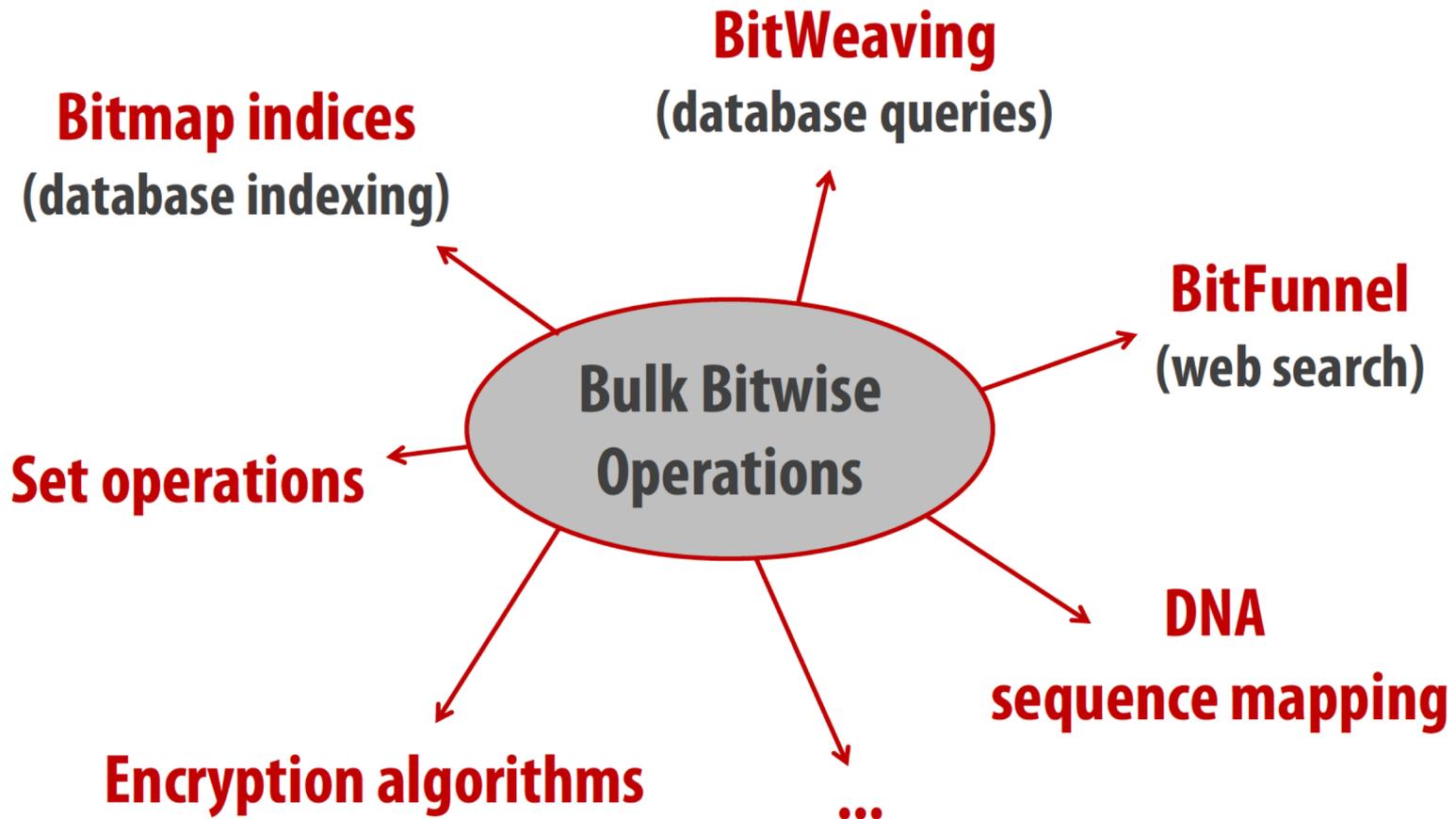
# Example Data Structure: Bitmap Index

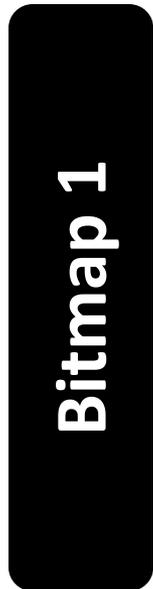- Alternative to B-tree and its variants
- Efficient for performing *range queries* and *joins*
- **Many bitwise operations to perform a query**

**age < 18    18 < age < 25    25 < age < 60    age > 60**

**Bitmap 1**    **Bitmap 2**    **Bitmap 3**    **Bitmap 4**

# Performance: Bitmap Index on Ambit
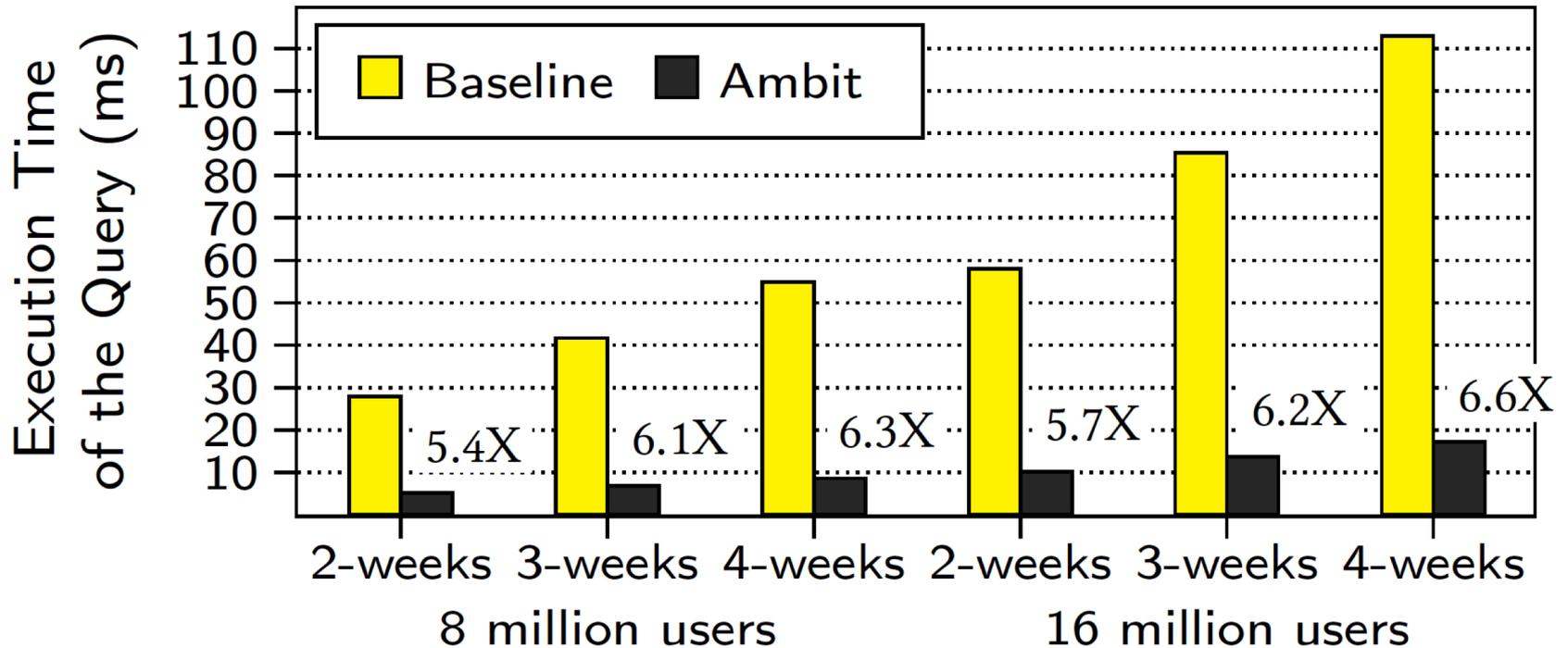


Figure 10: Bitmap index performance. The value above each bar indicates the reduction in execution time due to Ambit.

>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



'select count(*) from T where c1 <= val <= c2'

**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

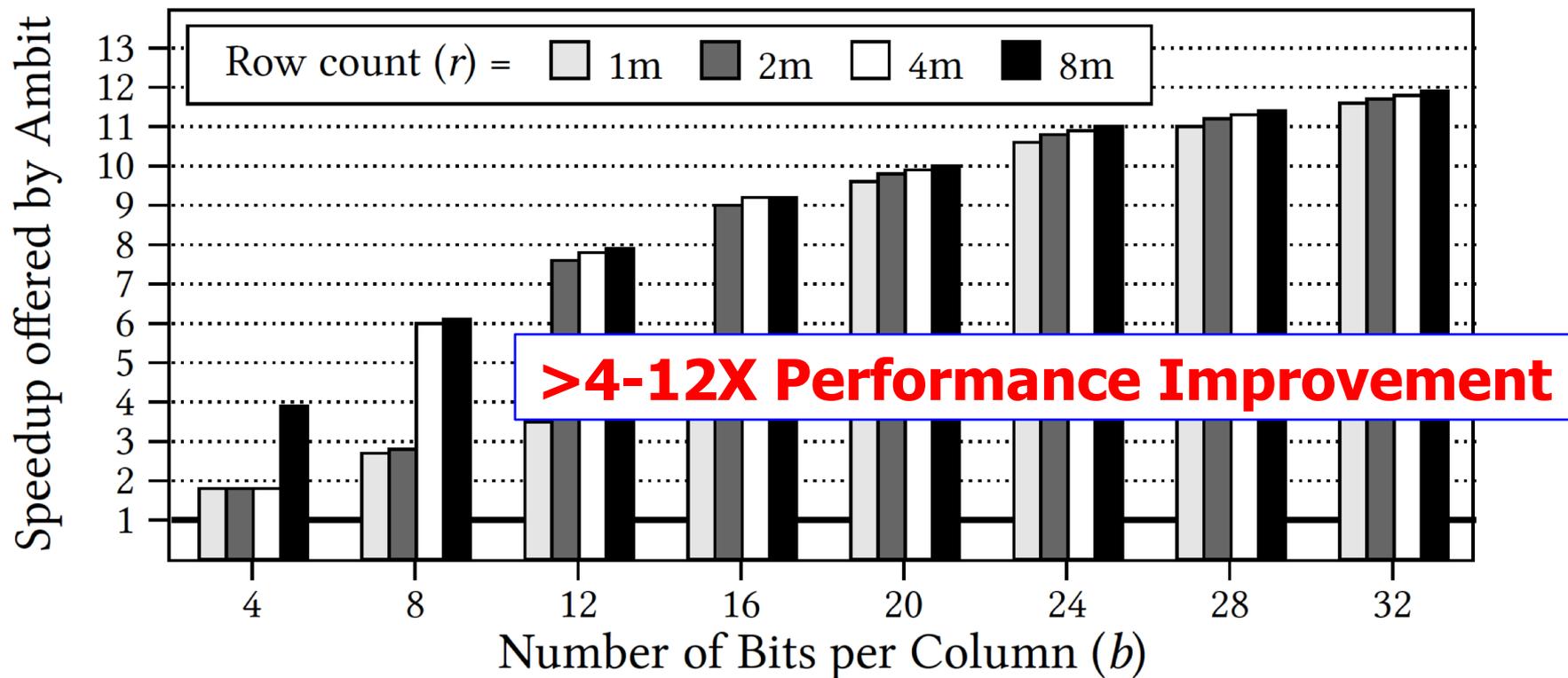- Vivek Seshadri, Kevin Hsieh, Amirali Boroumand, Donghyuk Lee, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  **"Fast Bulk Bitwise AND and OR in DRAM"**
  *IEEE Computer Architecture Letters* (**CAL**), April 2015.

## 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

SAFARI

# 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 Hassan[4]   Amirali Boroumand[5]
Jeremie Kim[4,5]   Michael A. Kozuch[3]   Onur Mutlu[4,5]   Phillip B. Gibbons[5]   Todd C. Mowry[5]

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

# Does memory have to be dumb?

**SAFARI**

# Computing Architectures with

# Minimal Data Movement

# Sub-Agenda: In-Memory Computation

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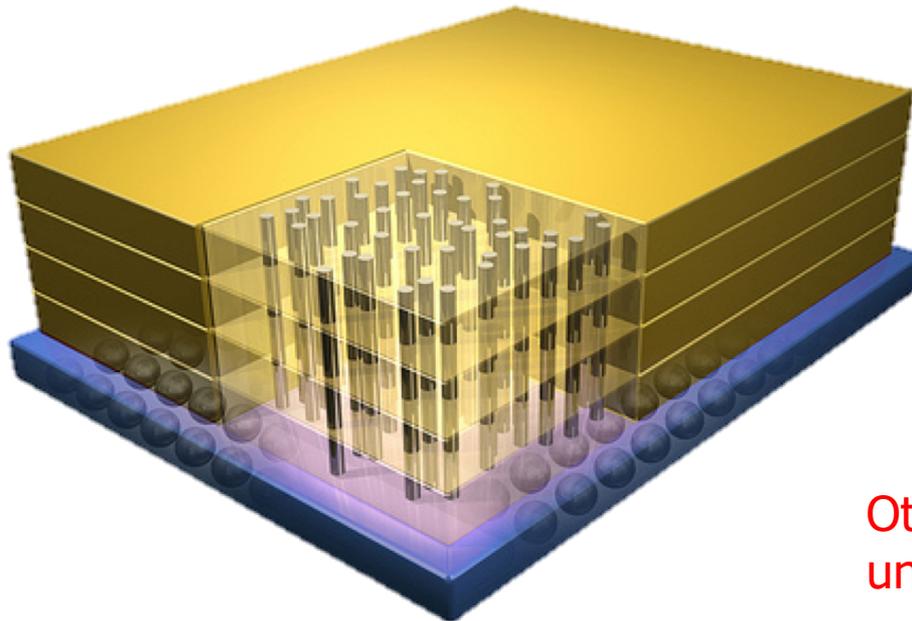- Conclusion

**SAFARI**

# Processing in Memory: Two Approaches

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

# Opportunity: 3D-Stacked Logic+Memory

**Memory**

**Logic**

Other "True 3D" technologies under development

# DRAM Landscape (circa 2015)

| Segment | DRAM Standards & Architectures |
|---|---|
| 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

**32 Cores**

**128...** +42%

0      1      2      3      4

Speedup
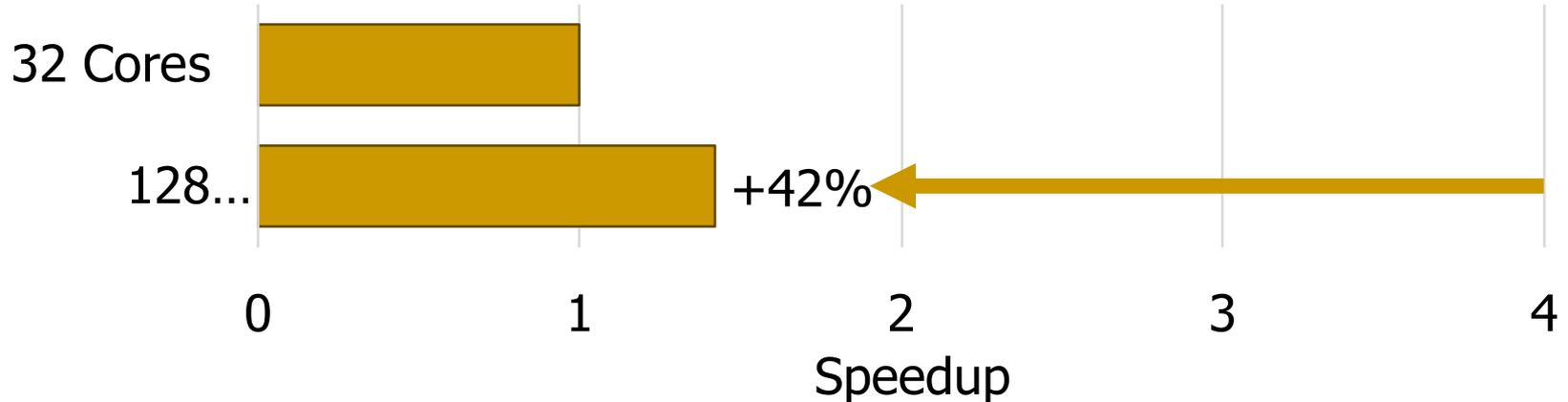
# Key Bottlenecks in Graph Processing

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

**1. Frequent random memory accesses**

w.rank

w.next_rank

w.edges

...

v

&w

W

weight * v.rank
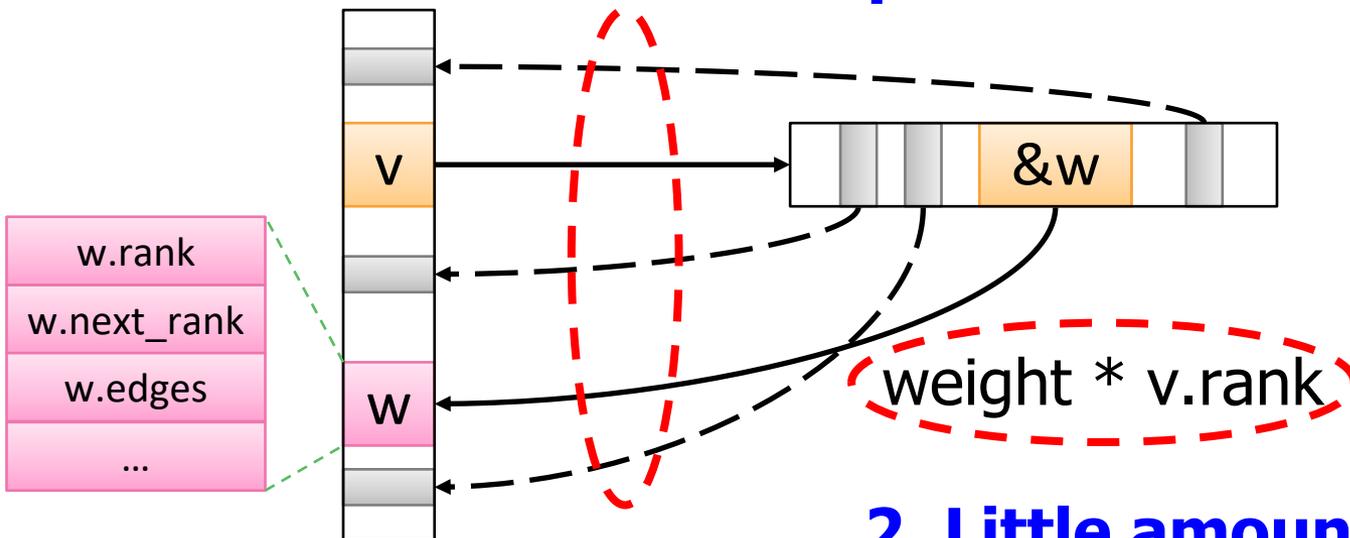
**2. Little amount of computation**

**SAFARI**

# Tesseract System for Graph Processing

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



Host Processor

Memory-Mapped
Accelerator Interface
(Noncacheable, Physically Addressed)

Memory

Logic

Crossbar Network

In-Order Core

DRAM Controller

LP

PF Buffer

MTP

Message Queue

NI

Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing" ISCA 2015.

# Tesseract System for Graph Processing



Host Processor

Memory-Mapped
Accelerator Interface
(Noncacheable, Physically Addressed)

Memory

Logic

Crossbar Network

In-Order Core

DRAM

Communications via
Remote Function Calls

Message Queue

NI

**SAFARI**

# Communications In Tesseract (I)

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

**SAFARI**

# Communications In Tesseract (II)

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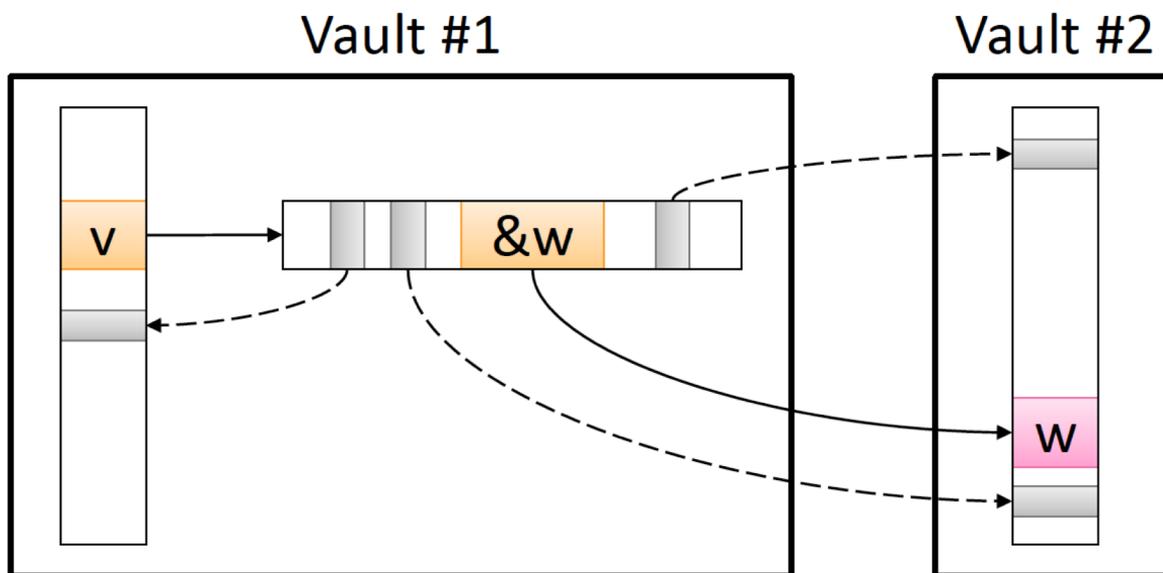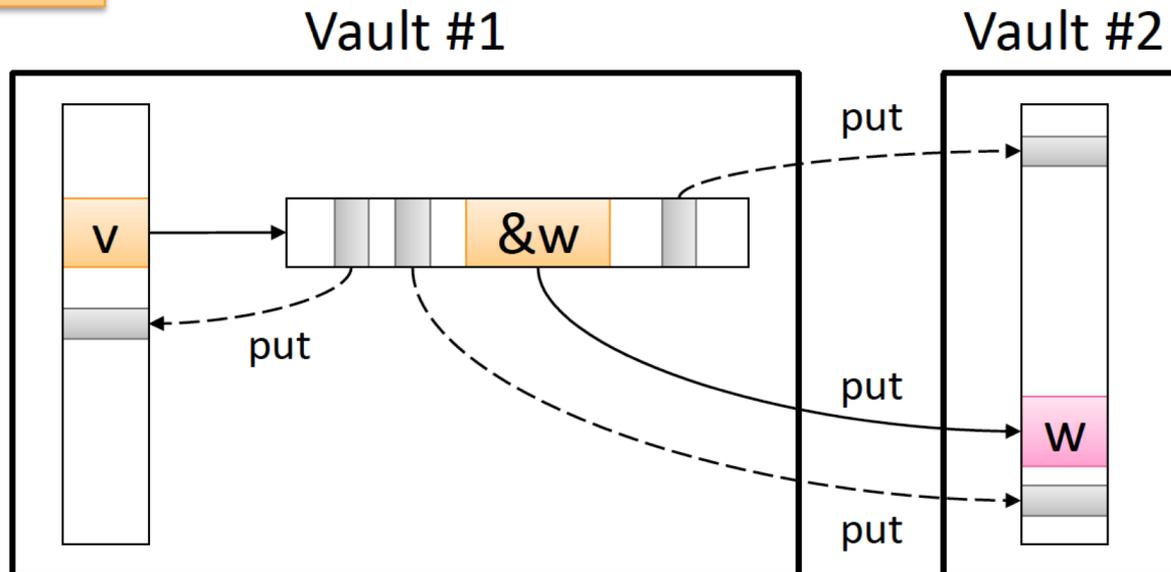
# Communications In Tesseract (III)

```
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
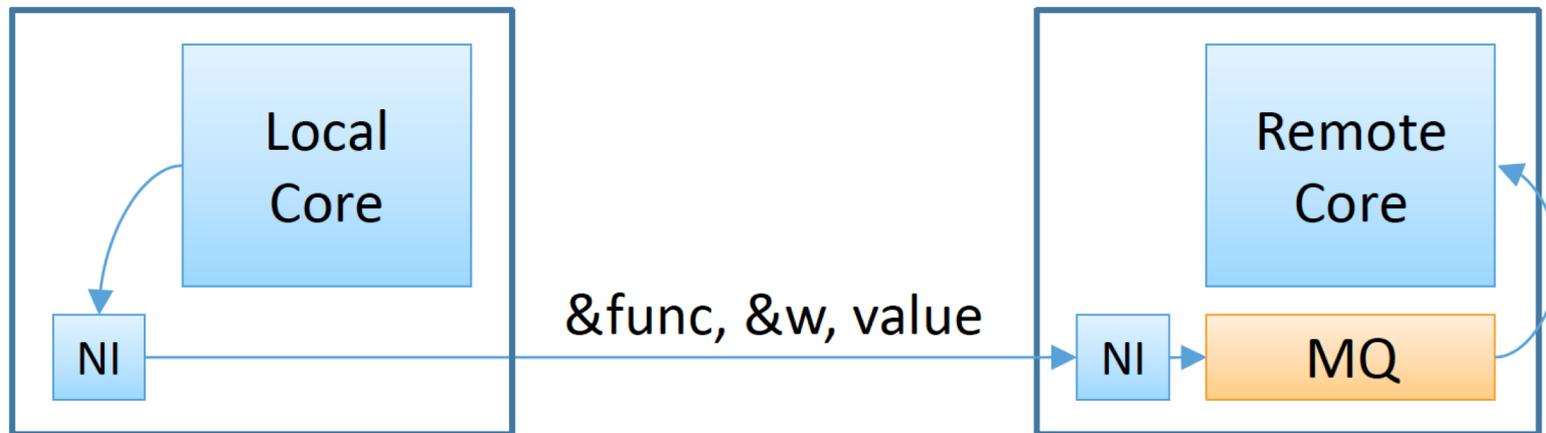


put(w.id, function() { w.next_rank += value; })

# Tesseract System for Graph Processing
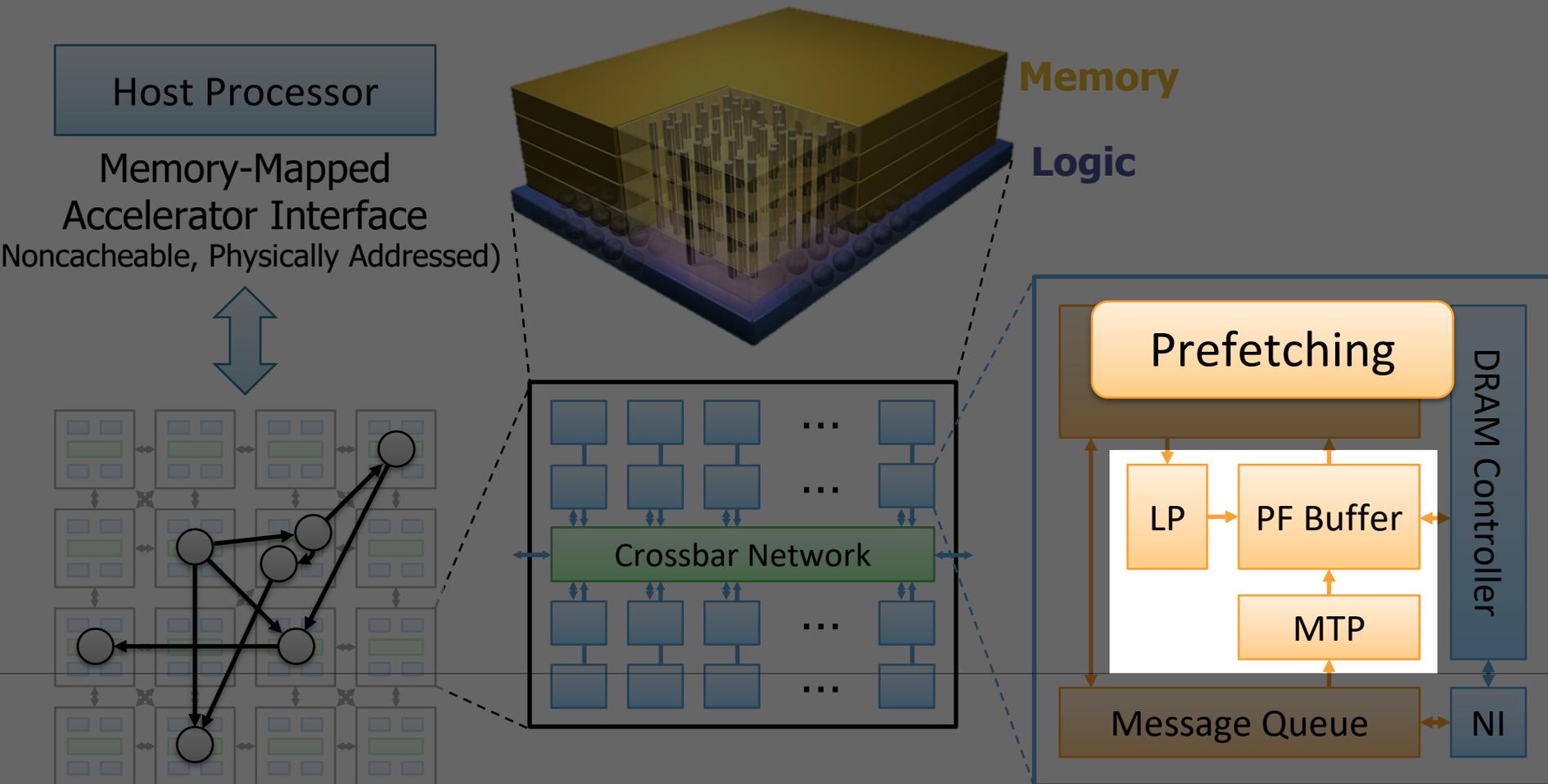


Host Processor

Memory-Mapped
Accelerator Interface
(Noncacheable, Physically Addressed)

**Memory**

**Logic**

Crossbar Network

...
...
...
...

Prefetching

LP | PF Buffer

MTP

DRAM Controller

Message Queue | NI
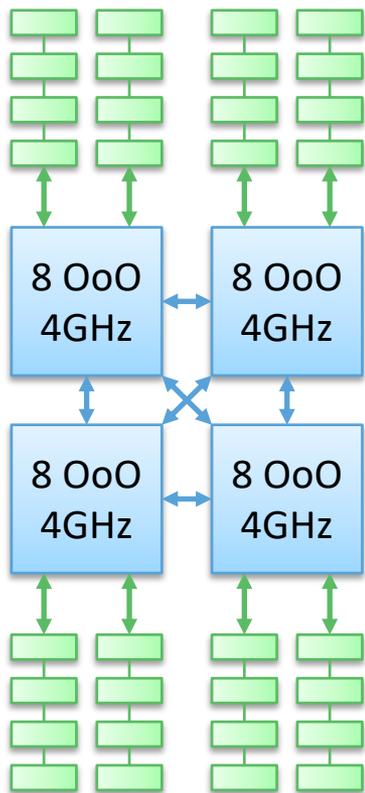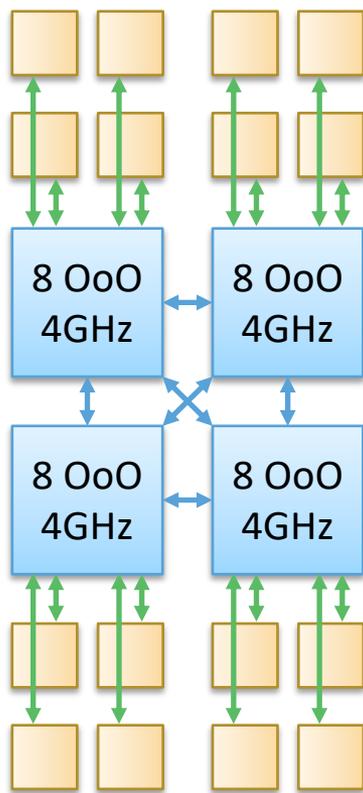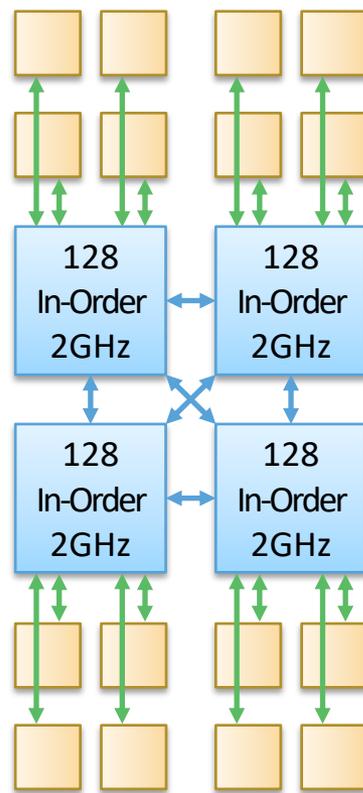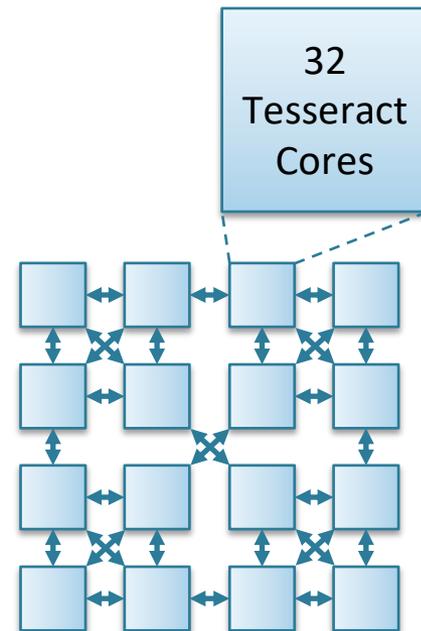
# Evaluated Systems

# Tesseract Graph Processing Performance



**>13X Performance Improvement**

On five graph processing algorithms

13.8x

11.6x

9.0x

+56%

+25%

Speedup

16 14 12 10 8 6 4 2 0

DDR3-OoO  HMC-OoO  HMC-MC  Tesseract  Tesseract-LP  Tesseract-LP-MTP

# Tesseract Graph Processing Performance



**Memory Bandwidth Consumption**

Memory Bandwidth (TB/s)

- 80GB/s — DDR3-OoO
- 190GB/s — HMC-OoO
- 243GB/s — HMC-MC
- 1.3TB/s — Tesseract
- 2.2TB/s — Tesseract-LP
- 2.9TB/s — Tesseract-LP-MTP
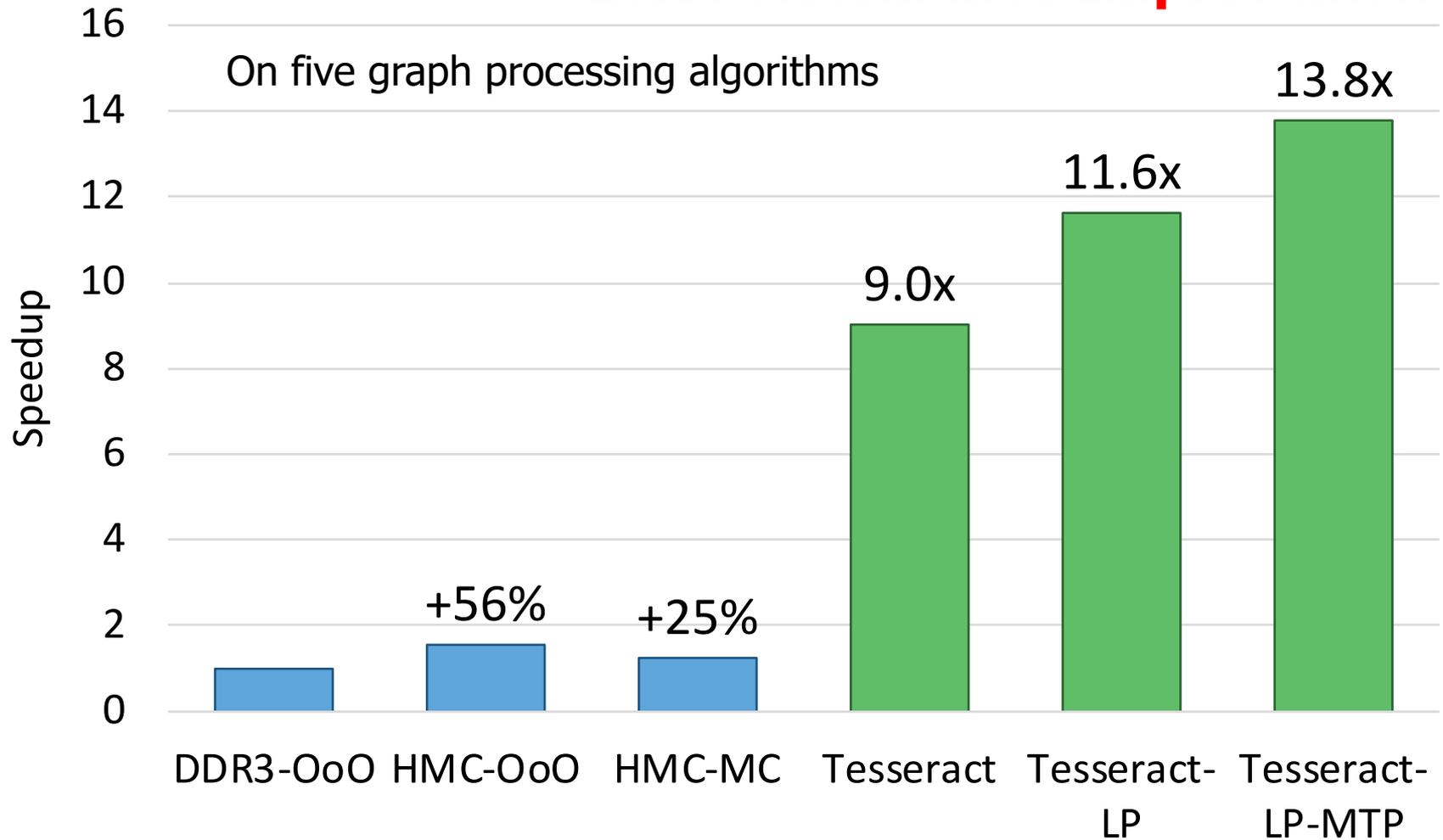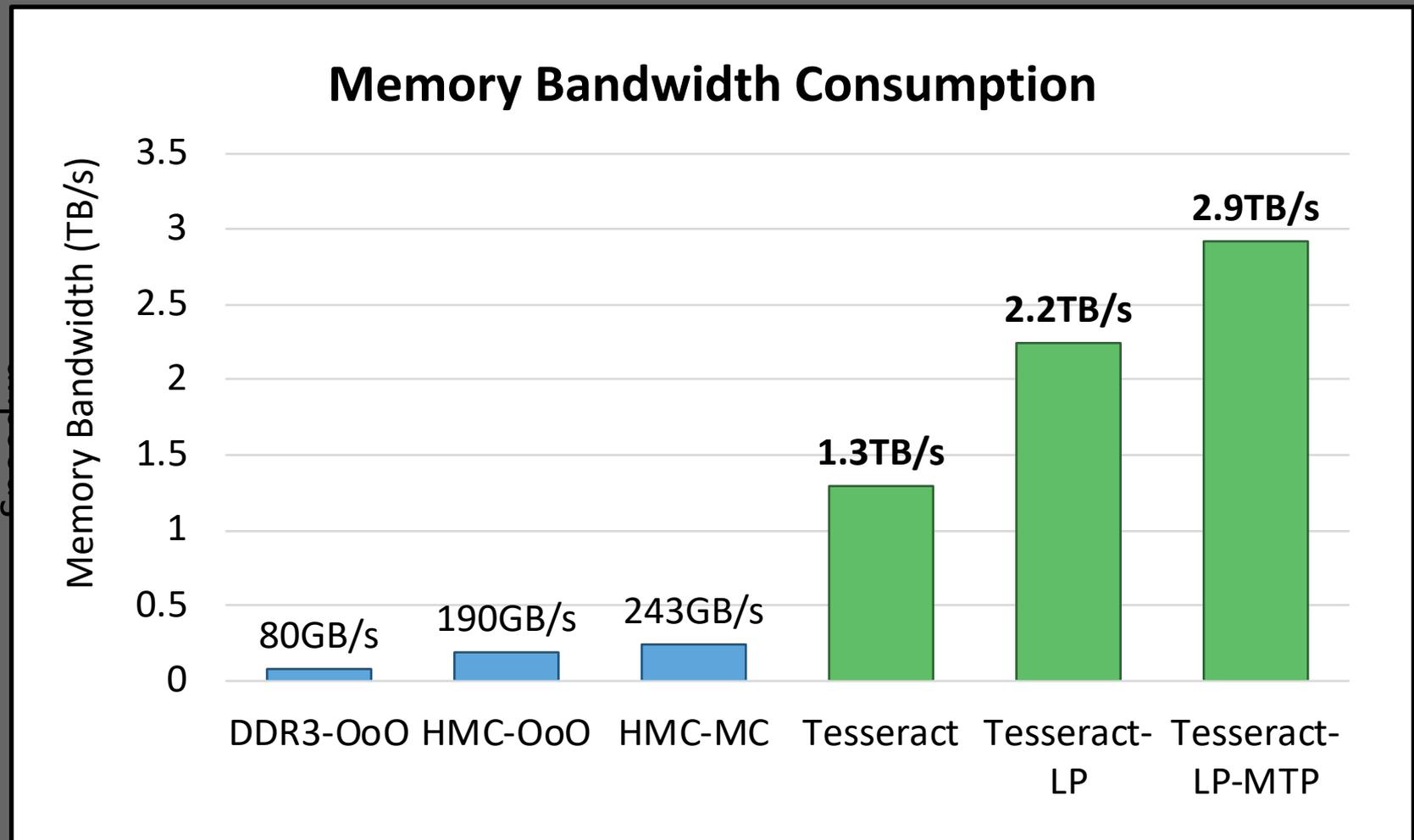
# Effect of Bandwidth & Programming Model
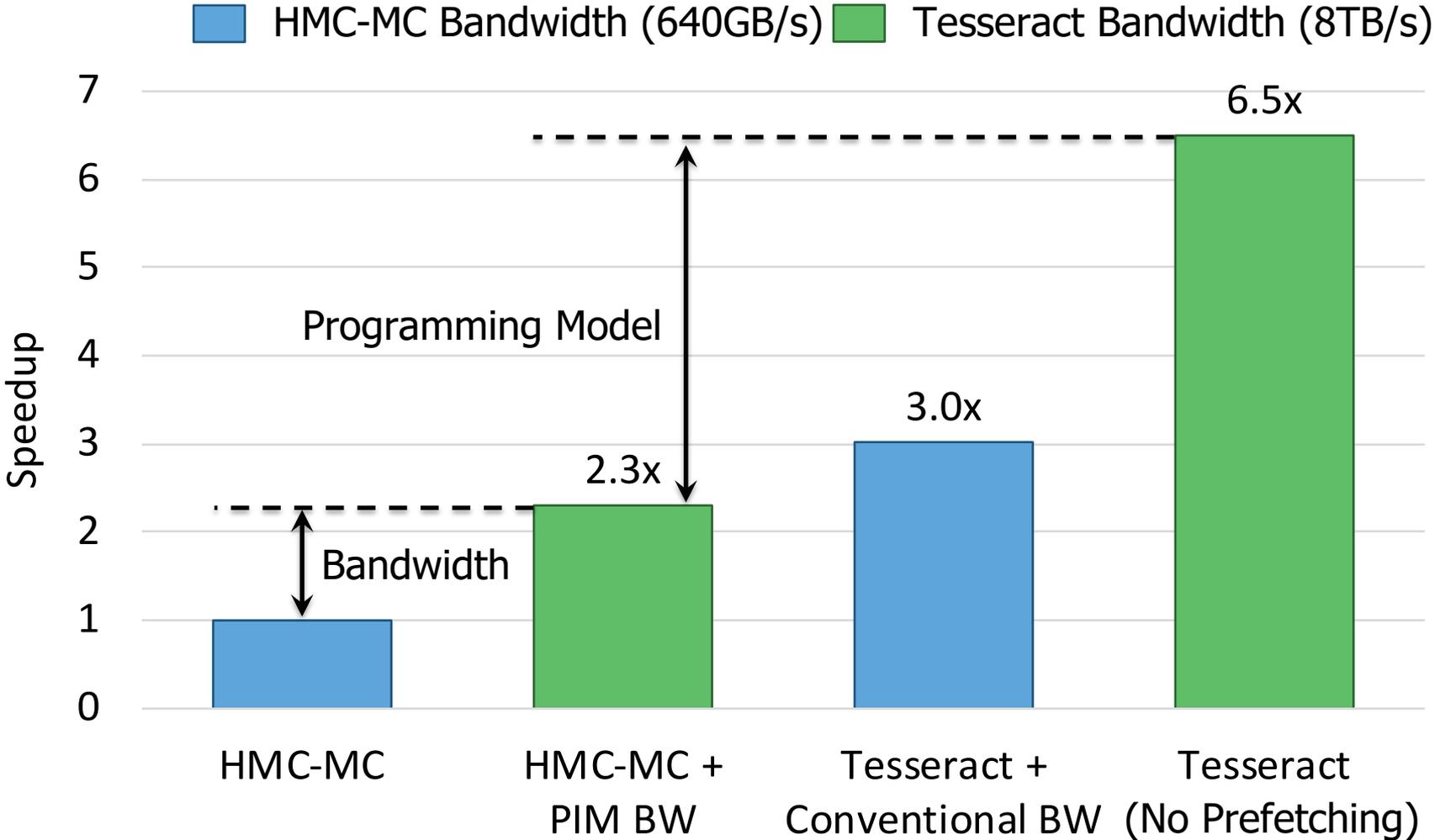
# Tesseract Graph Processing System Energy



Legend: ■ Memory Layers  ■ Logic Layers  □ Cores

Y-axis: 0, 0.2, 0.4, 0.6, 0.8, 1, 1.2

> 8X Energy Reduction

X-axis categories: HMC-OoO, Tesseract with Prefetching
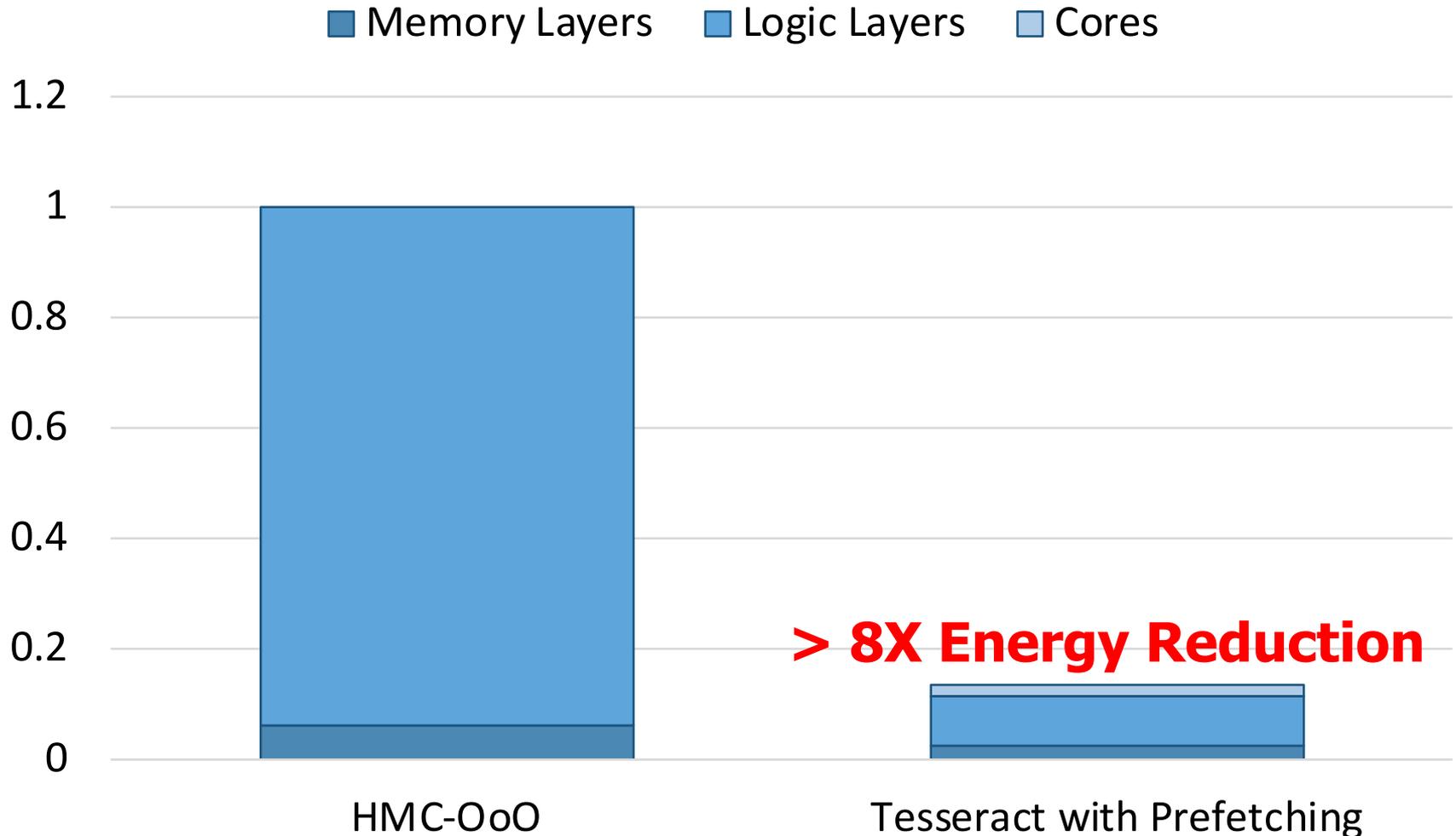
# More on Tesseract

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

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

Junwhan Ahn    Sungpack Hong[§]    Sungjoo Yoo    Onur Mutlu[†]    Kiyoung Choi
junwhan@snu.ac.kr, sungpack.hong@oracle.com, sungjoo.yoo@gmail.com, onur@cmu.edu, kchoi@snu.ac.kr
Seoul National University    [§]Oracle Labs    [†]Carnegie Mellon University

# Several Questions in 3D-Stacked PIM

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

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

# 3D-Stacked PIM on Mobile Devices

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

## Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

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

# 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**

# Consumer Devices

**Consumer devices are everywhere!**

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

# Popular Google Consumer 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

80

# Energy Cost of Data Movement

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

**Data Movement**

**SoC**

**CPU** ⟷ **L1** ⟷ **L2** ⟷ **DRAM**

**Compute Unit**

**Processing-In-Memory (PIM)**
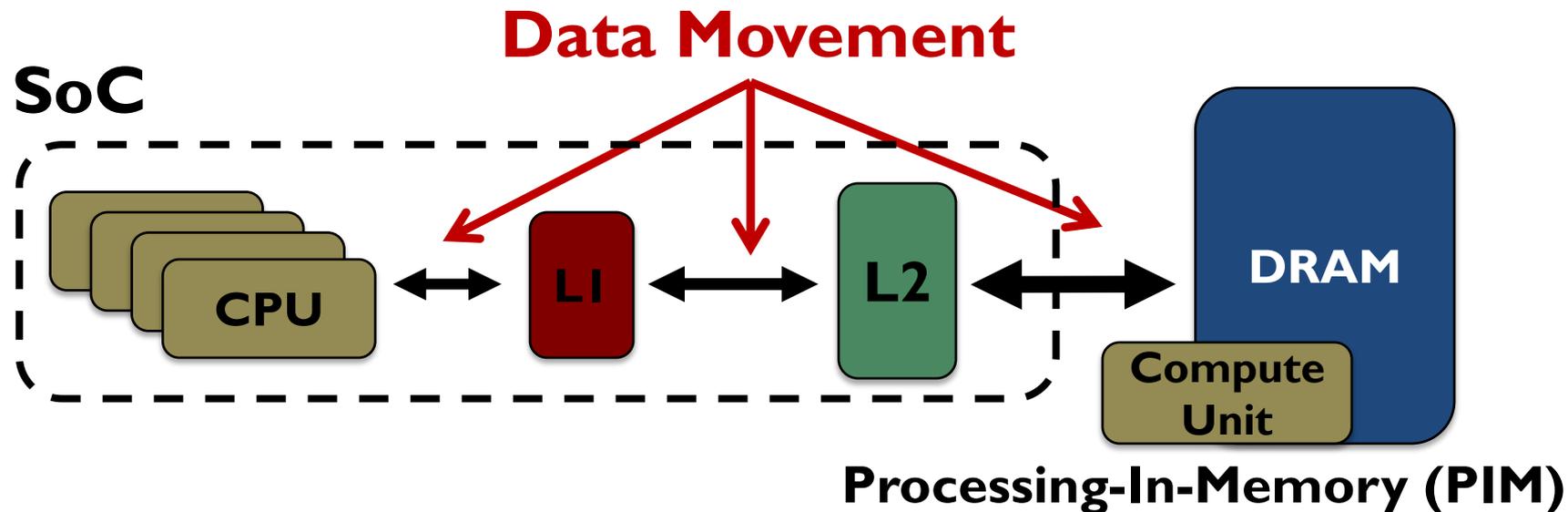
**Potential solution: move computation close to data**

**Challenge: limited area and energy budget**

# Using PIM to Reduce Data Movement

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

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

**Small embedded low-power core**

**Small fixed-function accelerators**

**PIM Core**

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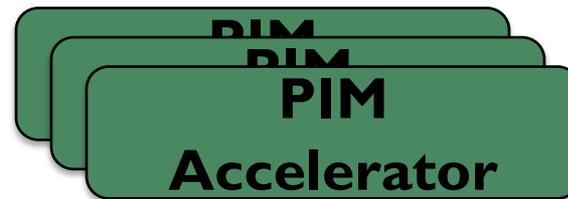
**Offloading to PIM logic reduces energy and improves performance, on average, by 55.4% and 54.2%**

# Goals

**1**    **Understand the data movement related bottlenecks in modern consumer workloads**

**2**    **Analyze opportunities to mitigate data movement by using processing-in-memory (PIM)**

**3**    **Design PIM logic that can maximize energy efficiency given the limited area and energy budget in consumer devices**

# 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**

# TensorFlow Mobile



**Inference** → **Prediction**
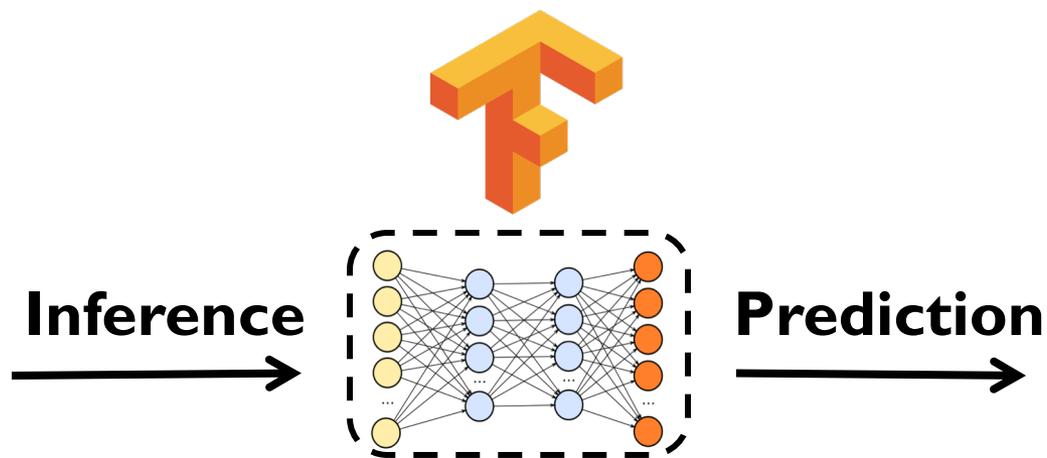
**57.3%** of the inference energy is spent on
**data movement**

↓

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

# Packing

Matrix → **Packing** → Packed Matrix

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

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

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

**A simple data reorganization process that requires simple arithmetic**

# Quantization

floating point → **Quantization** → integer

**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**

# Quantization

floating point → **Quantization** → integer

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

**Based on our analysis, we conclude that:**
- Both functions are good candidates for PIM execution
- It is feasible to implement them in PIM logic

inference **execution time**

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

# Evaluation Methodology

- **System Configuration (gem5 Simulator)**
  - **SoC:** 4 OoO cores, 8-wide issue, 64 kB L1 cache, 2MB L2 cache

  - **PIM Core:** 1 core per vault, 1-wide issue, 4-wide SIMD, 32kB L1 cache

  - **3D-Stacked Memory:** 2GB cube, 16 vaults per cube
    - Internal Bandwidth: 256GB/S
    - Off-Chip Channel Bandwidth: 32 GB/s

  - **Baseline Memory:** LPDDR3, 2GB, FR-FCFS scheduler

- **We study each target in isolation and emulate each separately and run them in our simulator**

# Normalized Energy
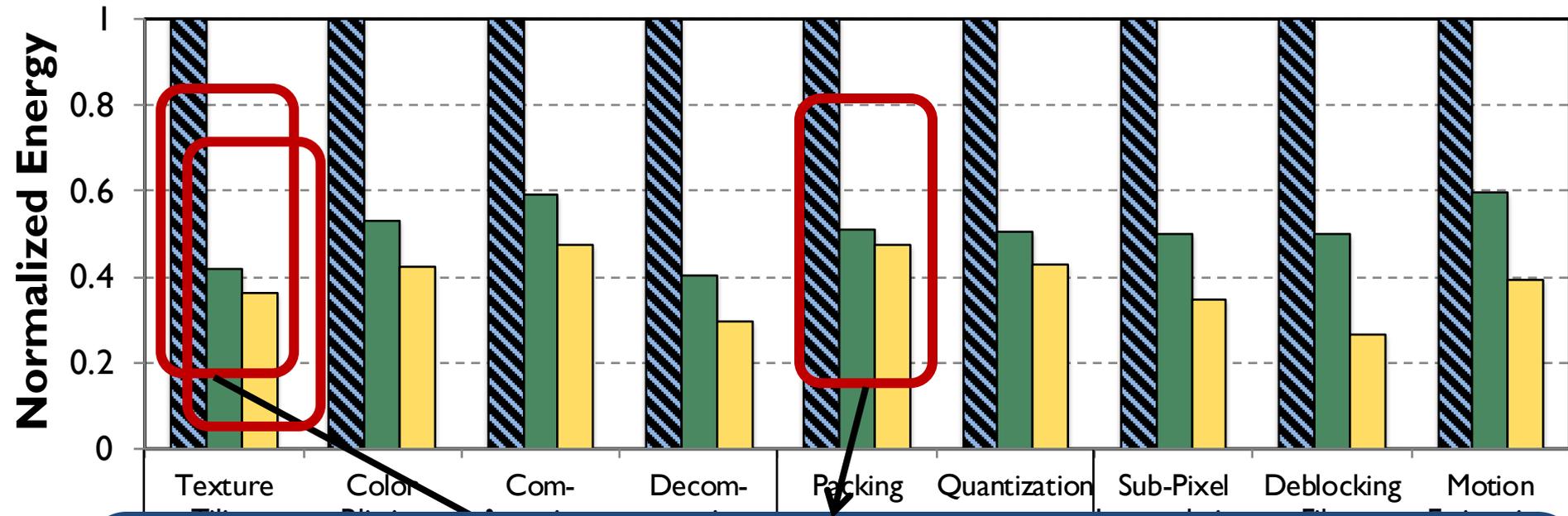


**77.7%** and **82.6%** of energy reduction for **texture tiling** and **packing** comes from eliminating **data movement**

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

# Normalized Runtime



**CPU-Only** **PIM-Core** **PIM-Acc**

*Chrome Browser:* Texture Tiling, Color Blitting, Comp-ression, Decomp-ression

*Video Playback and Capture:* Sub-Pixel Interpolation, Deblocking Filter, Motion Estimation

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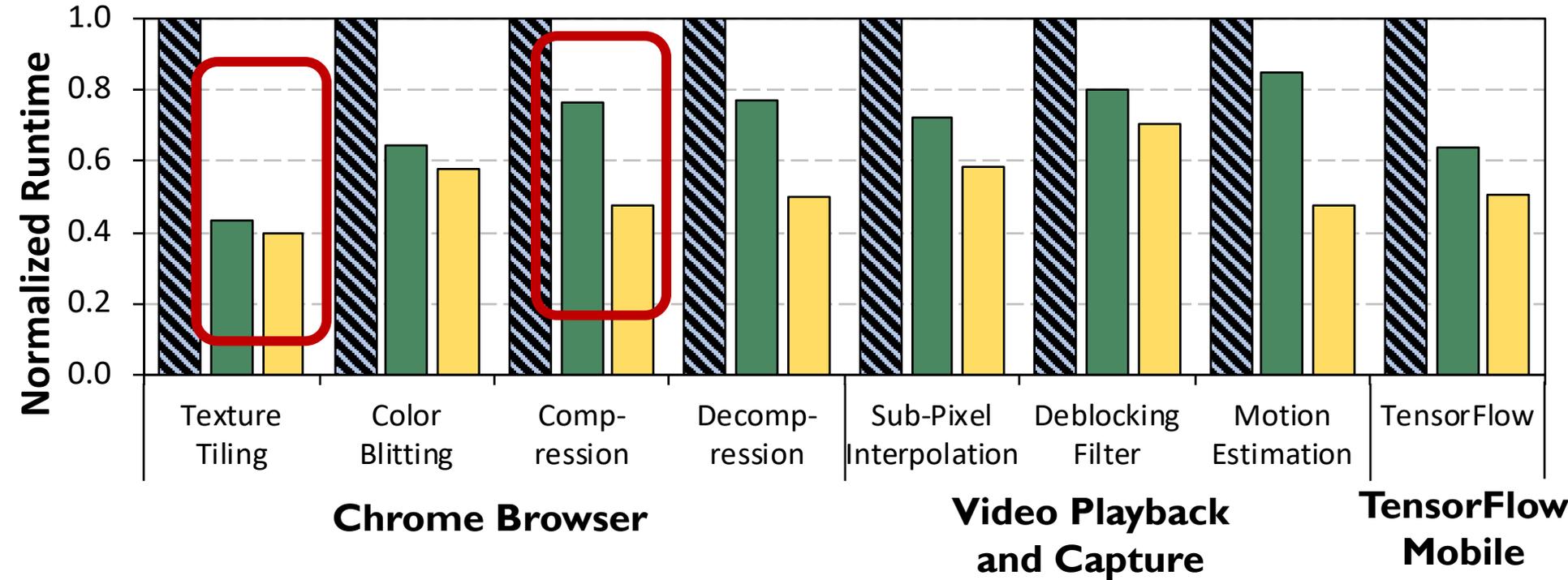
**Offloading these kernels to PIM core and PIM accelerator improves performance on average by 44.6% and 54.2%**

# 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

## ASPLOS 2018

# More on PIM for Mobile Devices

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

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

## Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

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

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

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

# Truly Distributed GPU Processing with PIM?

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

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

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

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

**SM (Streaming Multiprocessor)**



**Logic layer**

**Main GPU**

**Logic layer
SM**

**Crossbar switch**

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

# Accelerating GPU Execution with PIM (I)

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

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

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

# Accelerating GPU Execution with PIM (II)

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

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

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

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

# 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

# Several Questions in 3D-Stacked PIM

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

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

**SAFARI**

# PIM-Enabled Instructions

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

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

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

**SAFARI**

# PEI: PIM-Enabled Instructions (Ideas)

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

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

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

**SAFARI**

# 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

w.next_rank

w.next_rank

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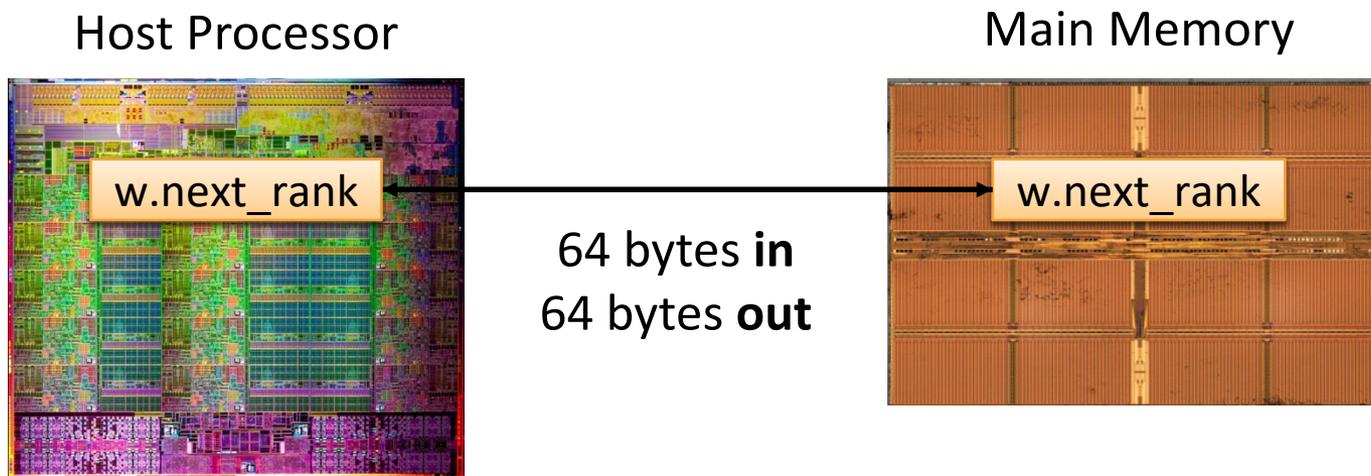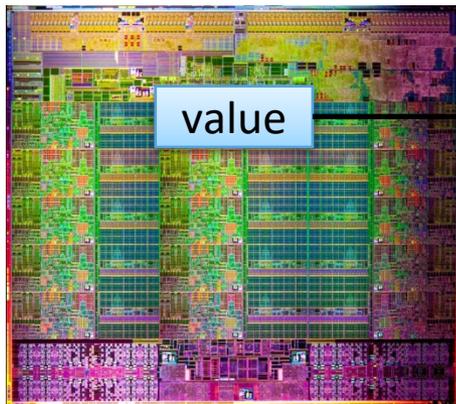
**Conventional Architecture**

# Simple PIM Operations as ISA Extensions (III)

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

pim.add r1, (r2)
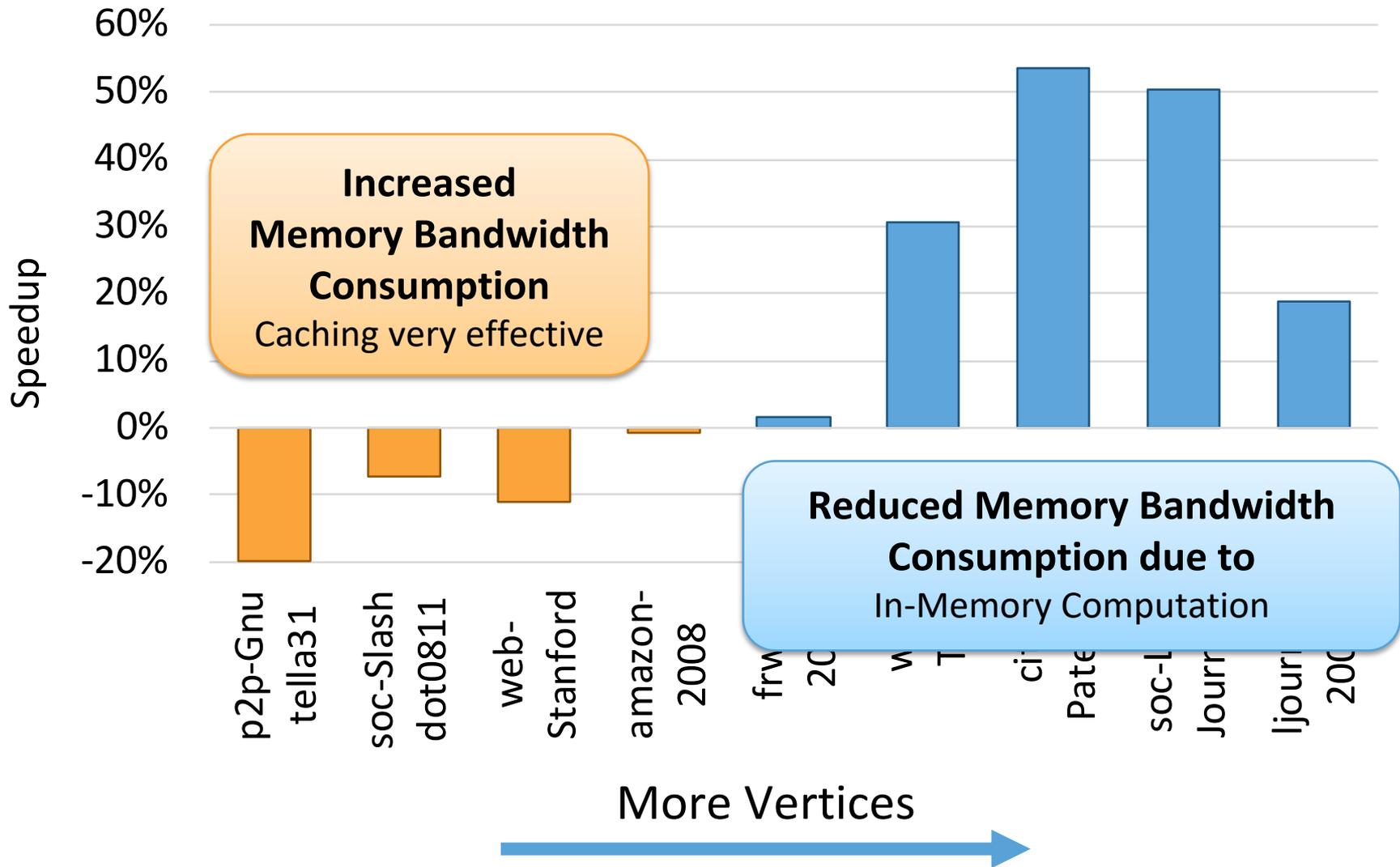
Host Processor

Main Memory

value → w.next_rank

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

**In-Memory Addition**

# Always Executing in Memory? Not A Good Idea

# PEI: PIM-Enabled Instructions (Example)

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

pim.add r1, (r2)

pfence

**Table 1: Summary of Supported PIM Operations**

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

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

# PIM-Enabled Instructions

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

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

# PEI: Initial Evaluation Results

- Initial evaluations with 10 emerging data-intensive workloads

  - Large-scale graph processing

  - In-memory data analytics

  - Machine learning and data mining

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

**Table 2: Baseline Simulation Configuration**

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

- Pin-based cycle-level x86-64 simulation

- **Performance Improvement and Energy Reduction:**

  - 47% average speedup with large input data sets

  - 32% speedup with small input data sets

  - 25% avg. energy reduction in a single node with large input data sets

**SAFARI**

# Evaluated Data-Intensive Applications

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

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

**SAFARI**

# PEI Performance Delta: Large Data Sets



(Large Inputs, Baseline: Host-Only)

**SAFARI**
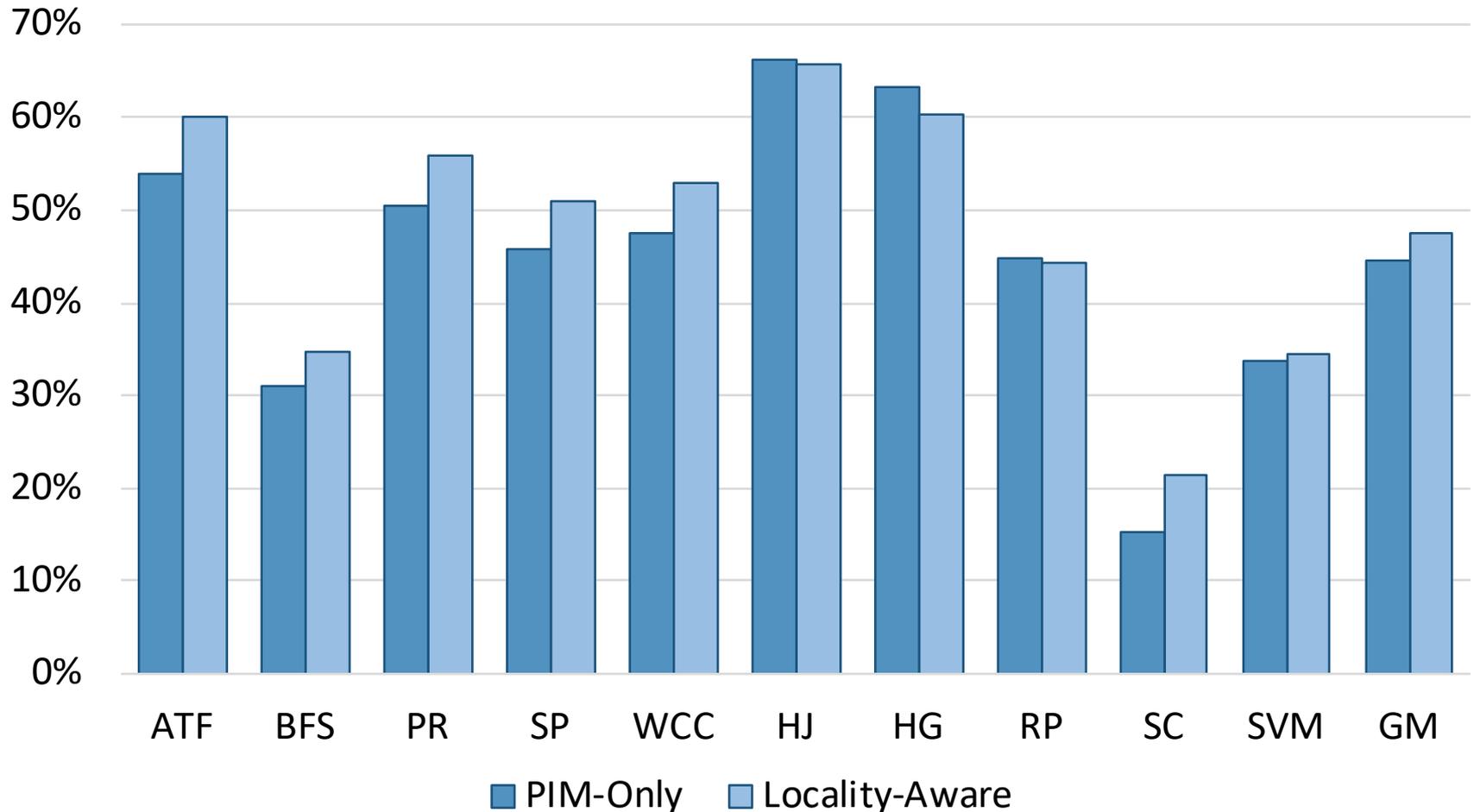
# PEI Performance: Large Data Sets



**Normalized Amount of Off-chip Transfer**

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

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

**SAFARI**
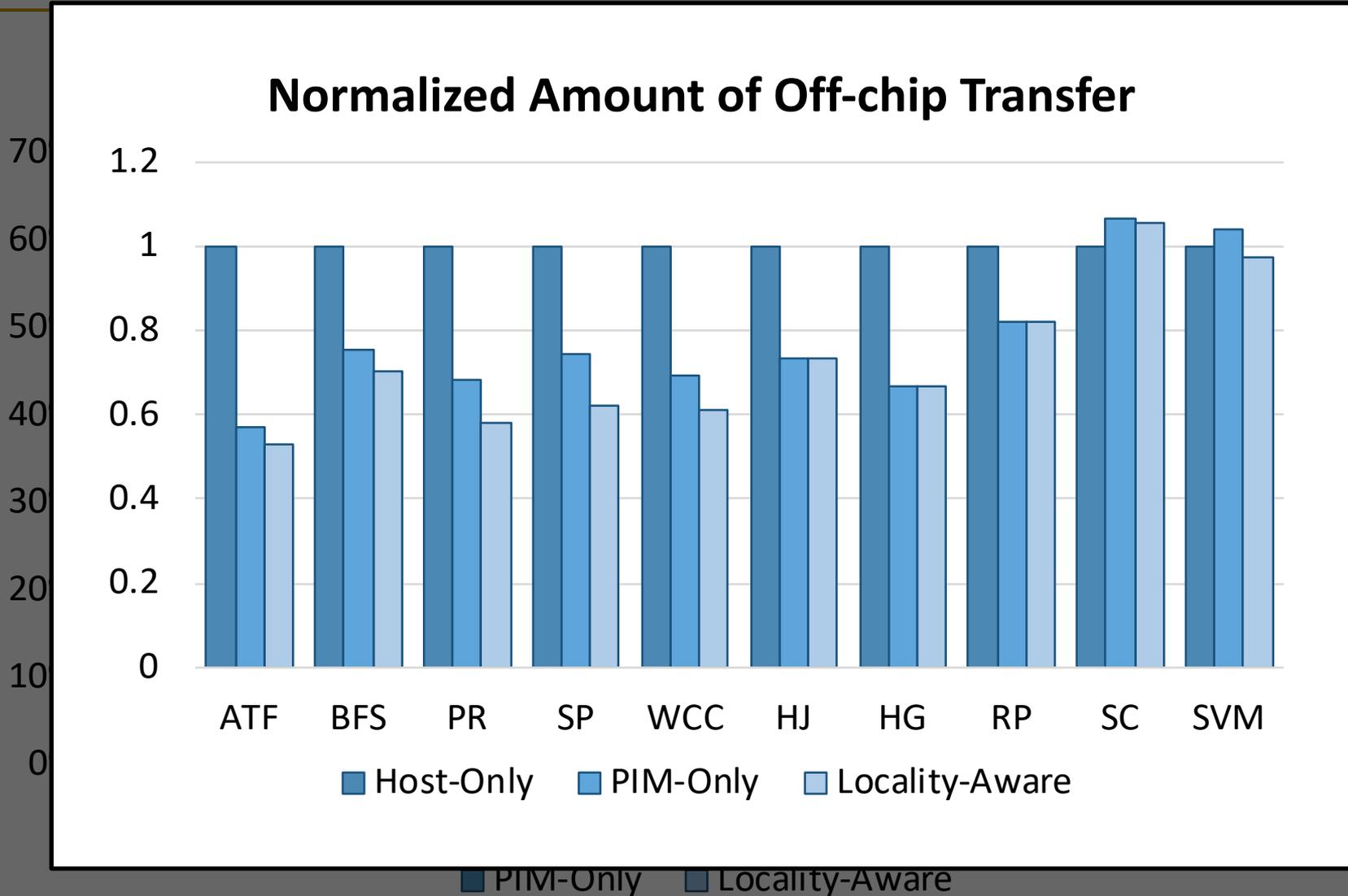
# PEI Performance Delta: Small Data Sets



(Small Inputs, Baseline: Host-Only)

Legend: PIM-Only, Locality-Aware

Categories: ATF, BFS, PR, SP, WCC, HJ, HG, RP, SC, SVM, GM
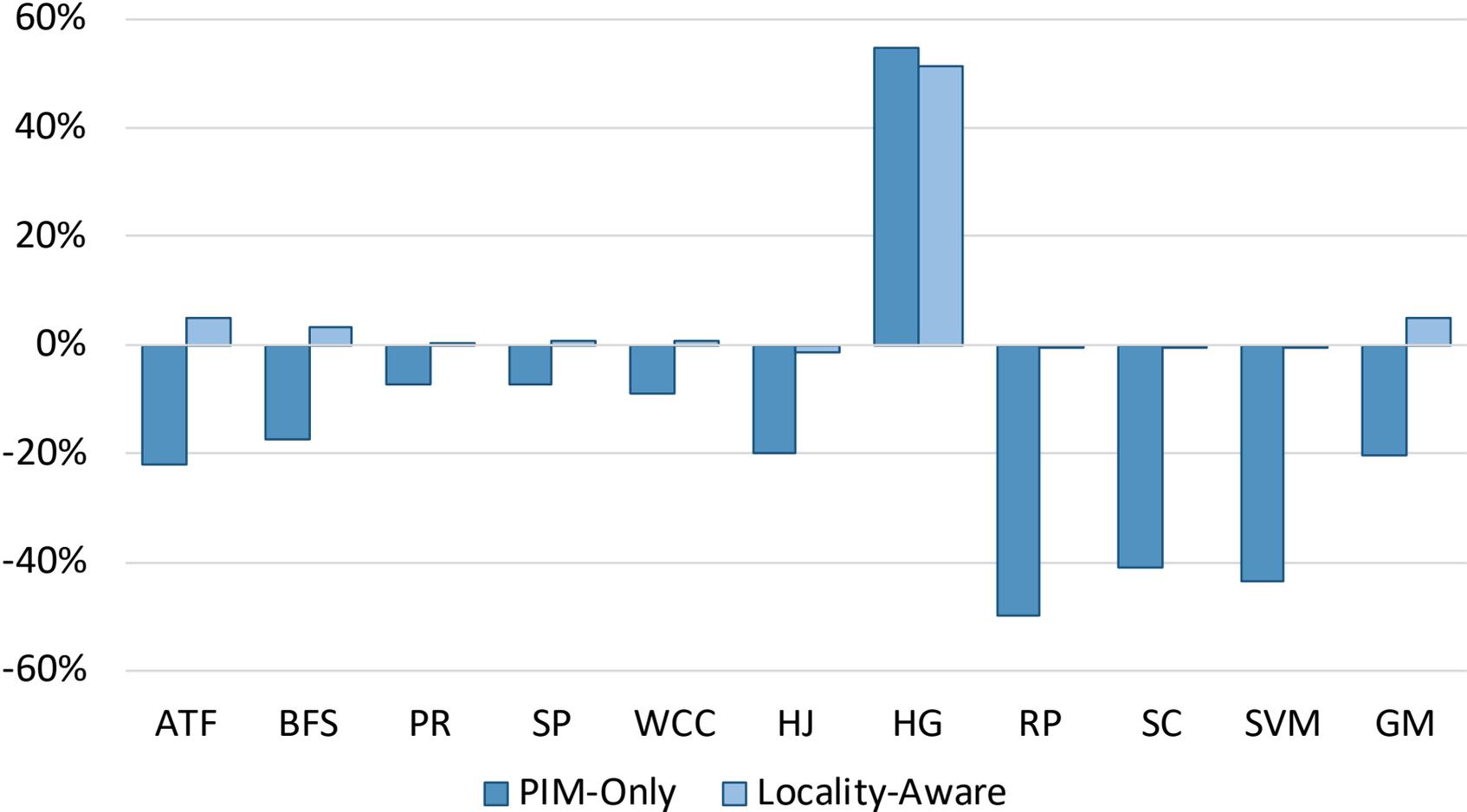
# PEI Performance: Small Data Sets



**Normalized Amount of Off-chip Transfer**

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

Legend: ■ Host-Only  ■ PIM-Only  □ Locality-Aware

**SAFARI**
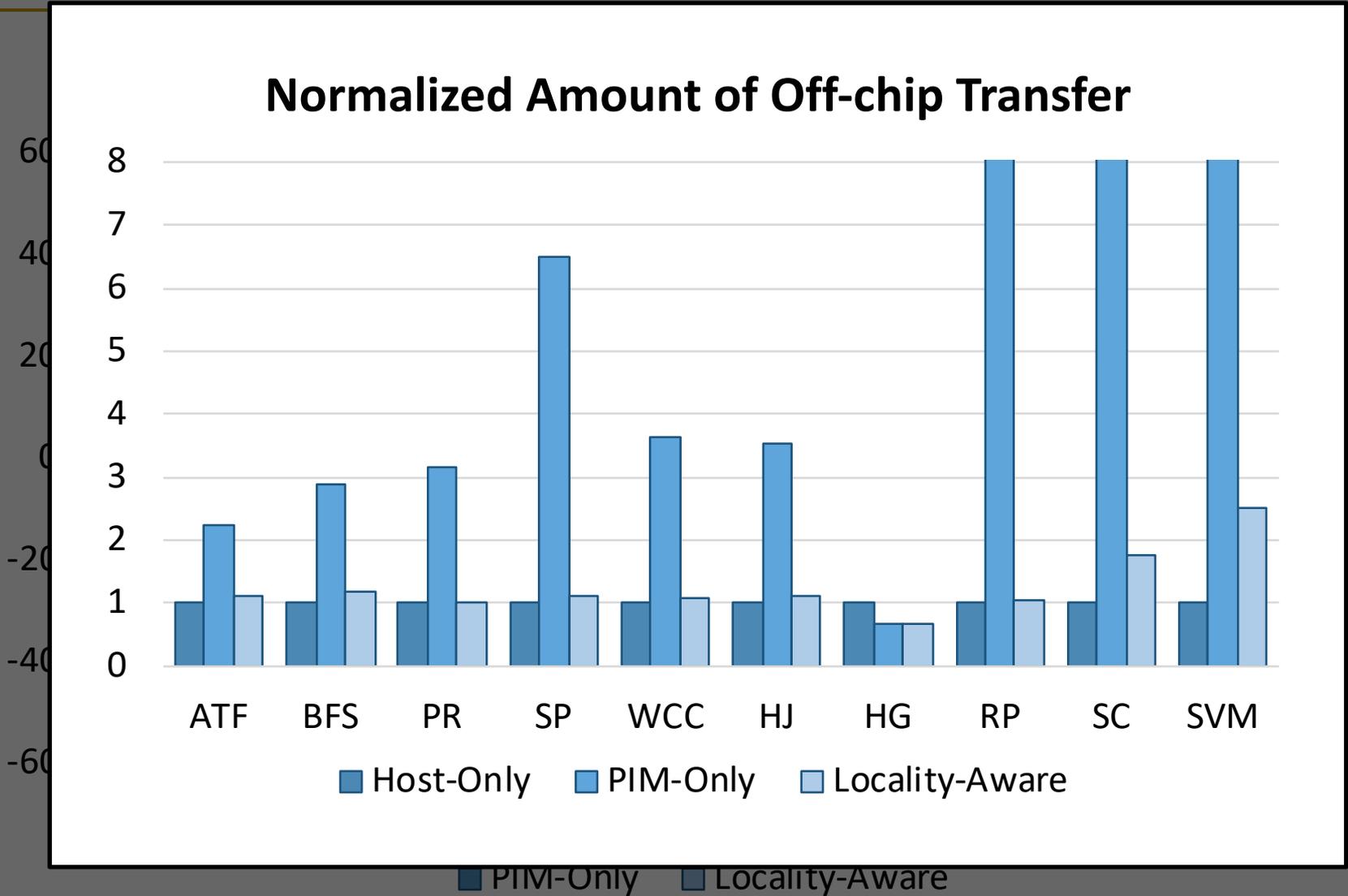
# PEI Performance Delta: Medium Data Sets



(Medium Inputs, Baseline: Host-Only)

Legend: PIM-Only, Locality-Aware
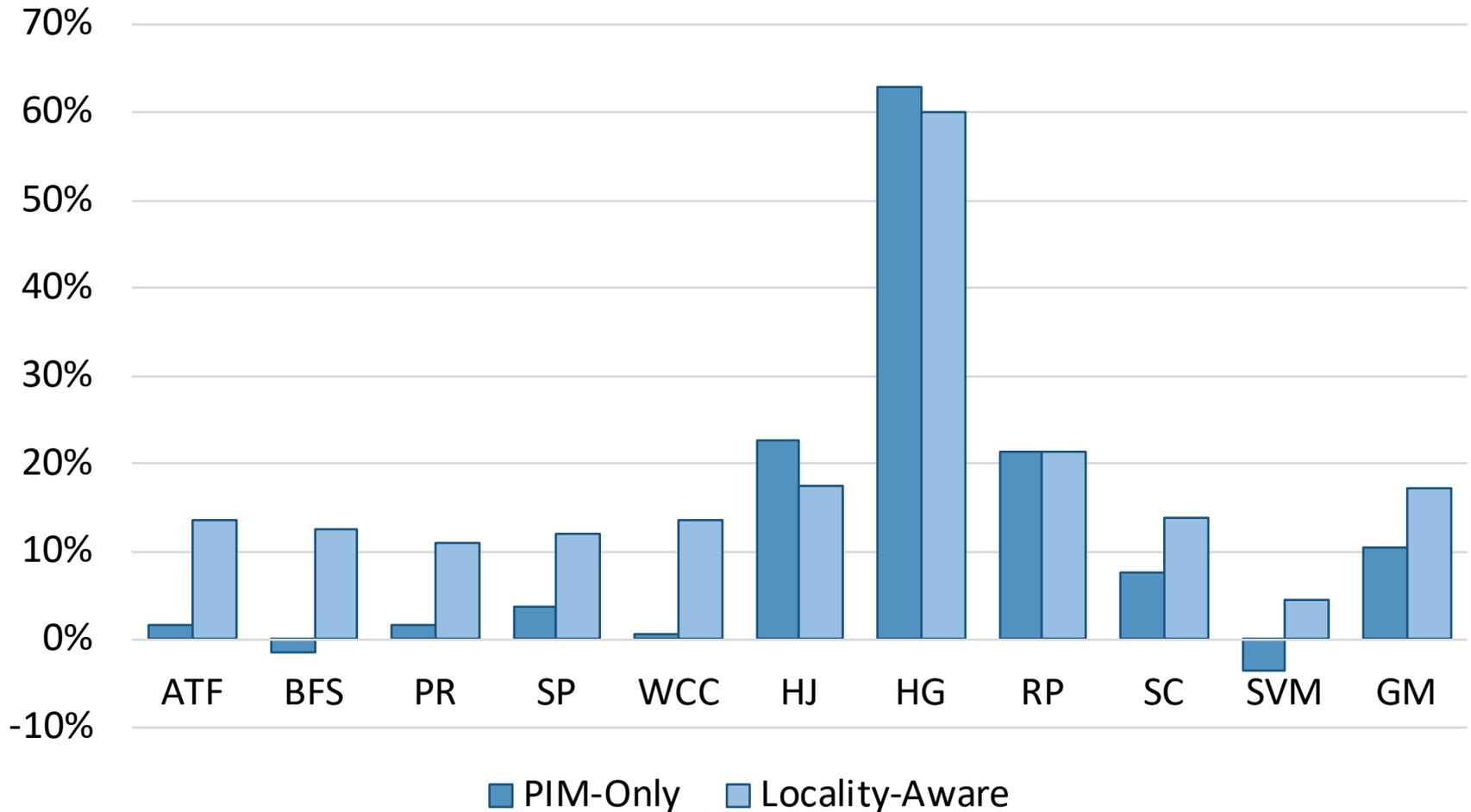
# PEI Energy Consumption
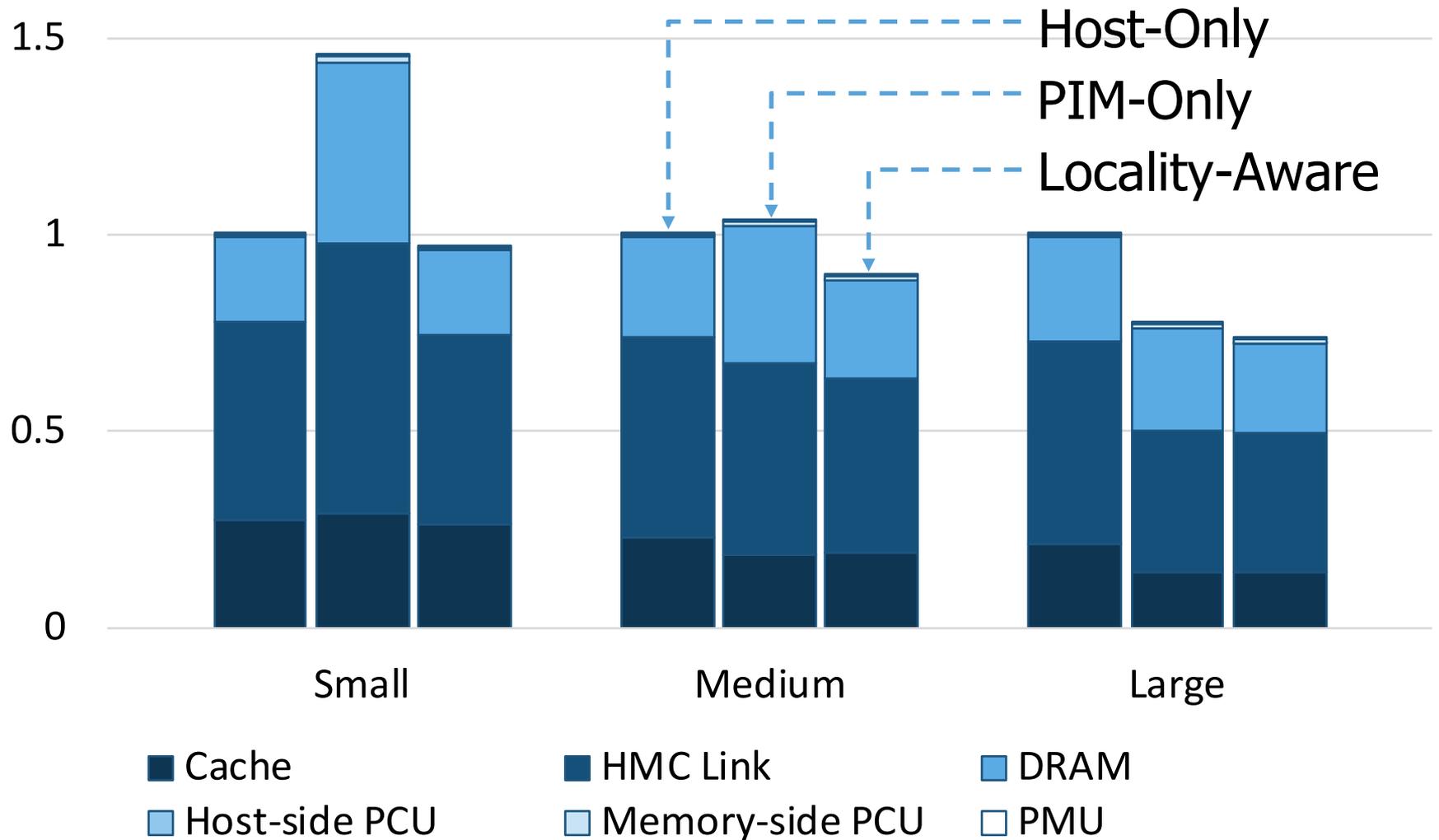
# PEI: Advantages & Disadvantages

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

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

Junwhan Ahn    Sungjoo Yoo    Onur Mutlu[†]    Kiyoung Choi

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

Seoul National University    [†]Carnegie Mellon University

SAFARI

# Automatic Code and Data Mapping

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

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

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

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

# Automatic Offloading of Critical Code

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

# Accelerating Dependent Cache Misses with an Enhanced Memory Controller

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

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

# Automatic Offloading of Prefetch Mechanisms

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

## Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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

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

# Efficient Automatic Data Coherence Support

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

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

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

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

# Efficient Automatic Data Coherence Support

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

## CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators

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

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

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

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

# Fundamentally Energy-Efficient (Data-Centric) Computing Architectures

# Fundamentally High-Performance (Data-Centric) Computing Architectures

# Computing Architectures with Minimal Data Movement

# Sub-Agenda: In-Memory Computation

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

**SAFARI**

# How to Enable Adoption of Processing in Memory

# Barriers to Adoption of PIM

1. Functionality of and applications & software for PIM

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

3. System support: coherence & virtual memory

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

5. Infrastructures to assess benefits and feasibility

**All can be solved with change of mindset**

# We Need to Revisit the Entire Stack

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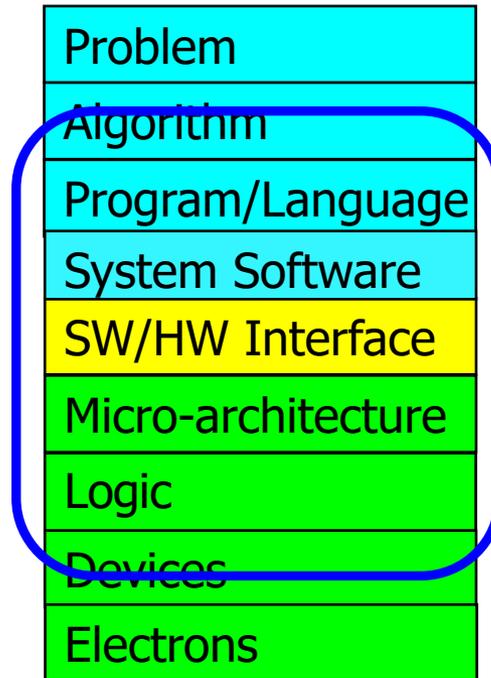
**We can get there step by step**

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

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

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

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

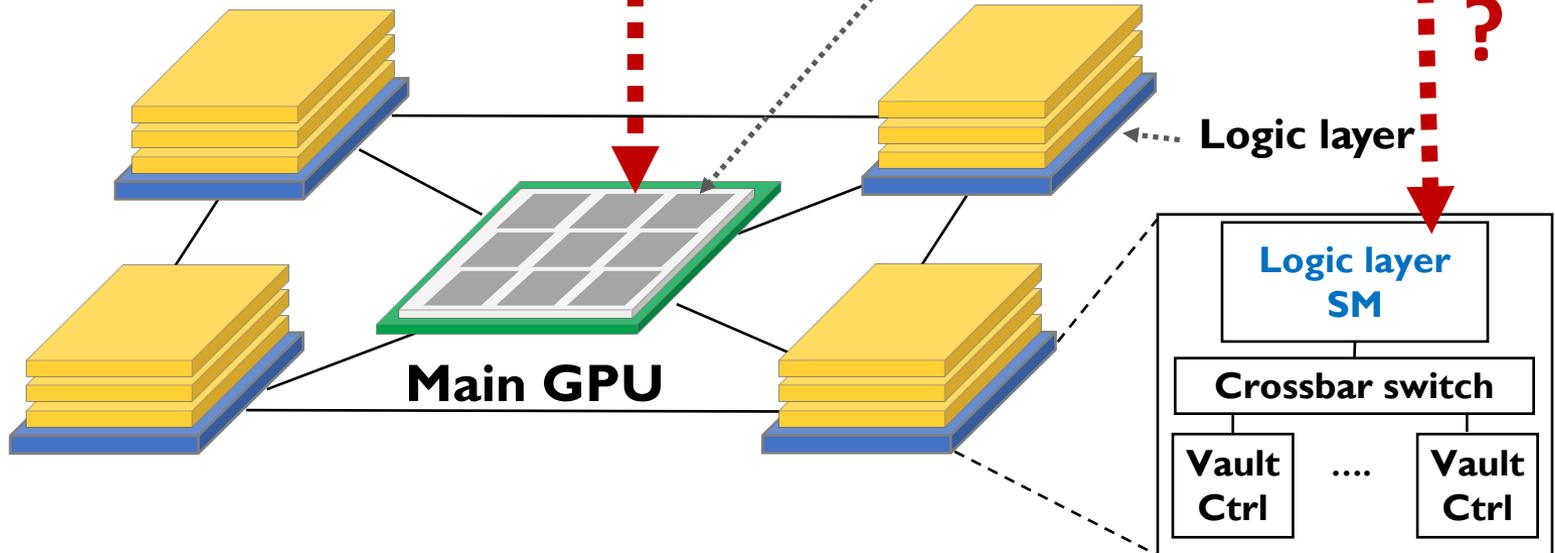**https://arxiv.org/pdf/1903.03988.pdf**

# Key Challenge 1: Code Mapping

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

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

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

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



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

**SM (Streaming Multiprocessor)**

**Logic layer**

**Main GPU**

**Logic layer SM**

**Crossbar switch**

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

# Key Challenge 2: Data Mapping

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



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

**SM (Streaming Multiprocessor)**

**Logic layer**

**Main GPU**

Logic layer SM

Crossbar switch

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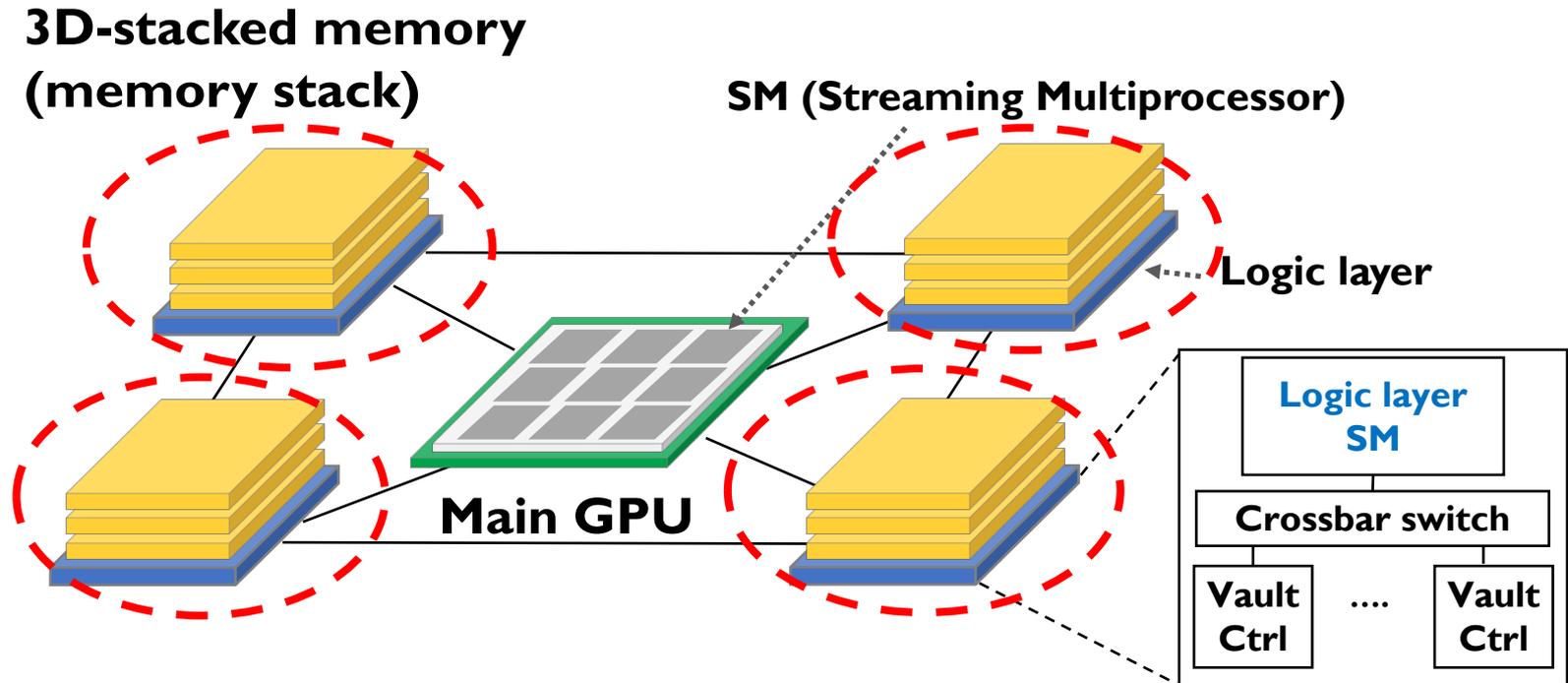
# How to Do the Code and Data Mapping?

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

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

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

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

# How to Schedule Code?

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

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

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

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

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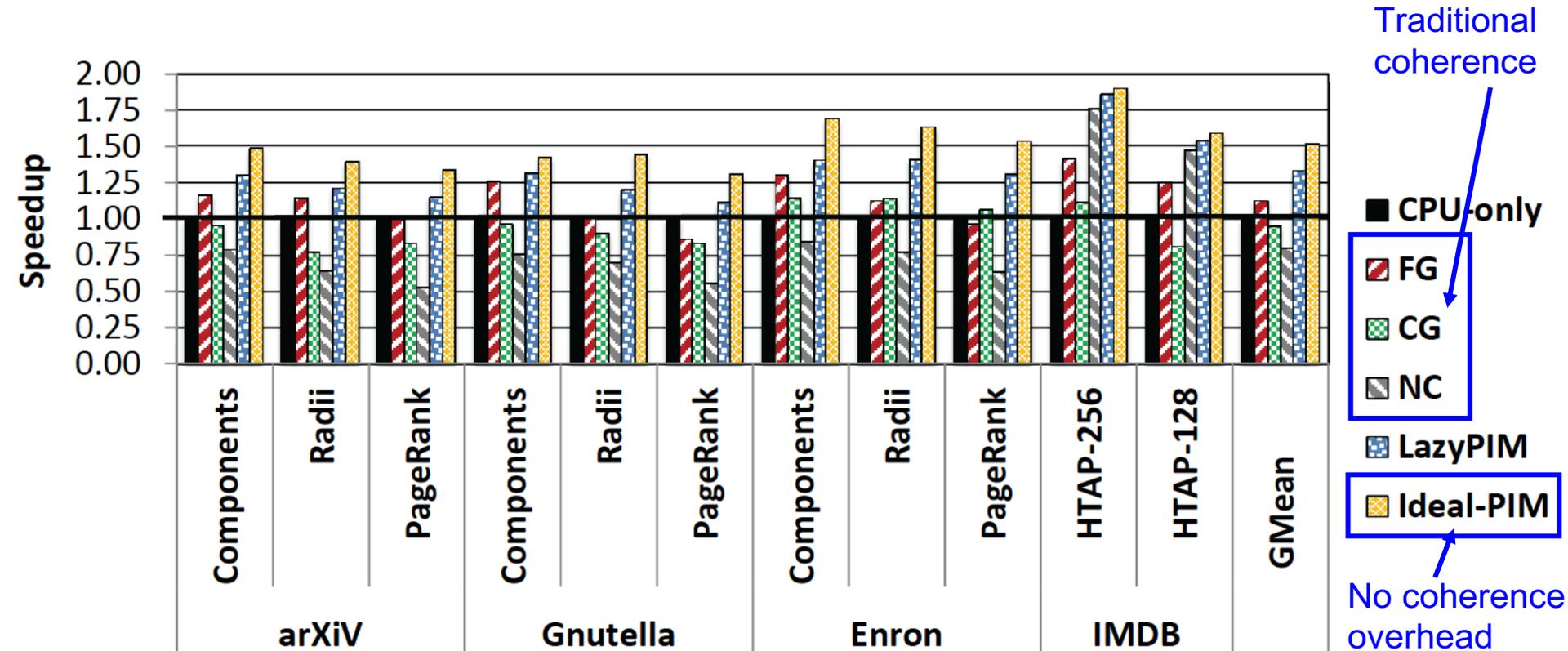
# How to Maintain Coherence? (I)

- 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
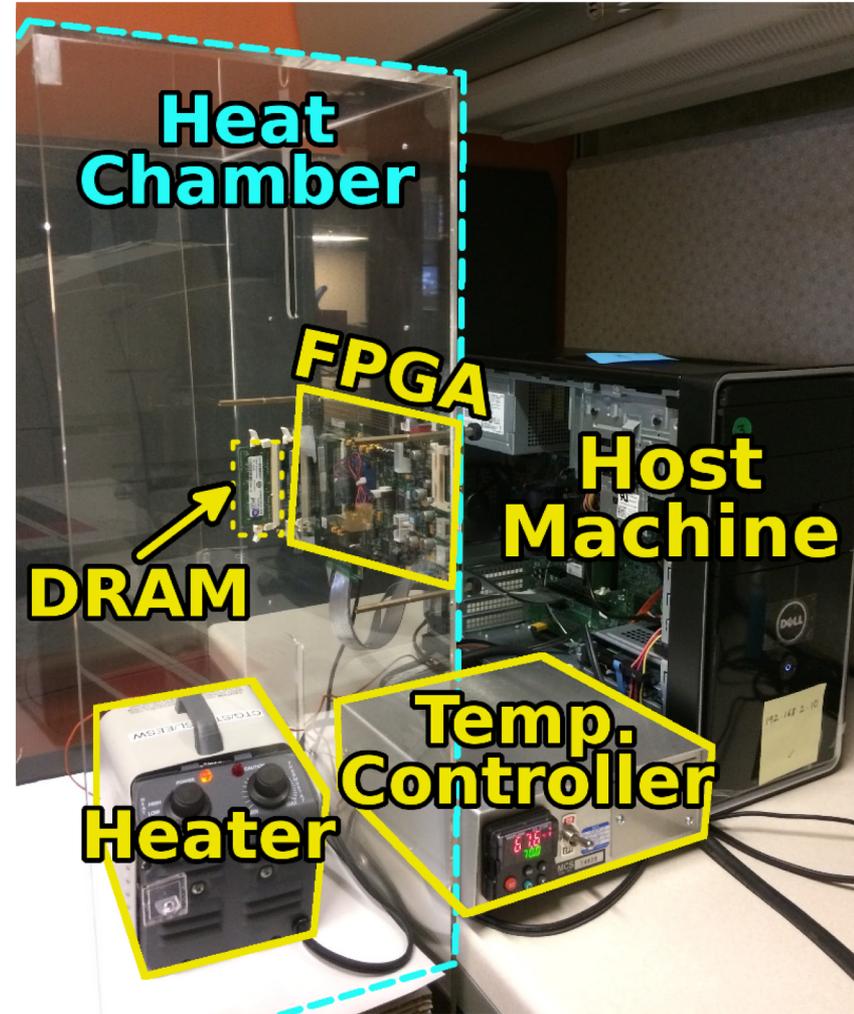
# Simulation Infrastructures for PIM

- **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

# Ramulator: A Fast and Extensible DRAM Simulator

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

# An FPGA-based Test-bed for PIM?

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


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

  *github.com/CMU-SAFARI/SoftMC*

**SAFARI**

# Simulation Infrastructures for PIM (in SSDs)

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

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

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

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

# New Applications and Use Cases for PIM

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

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

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

# 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**

# 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]

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

# 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

# Sub-Agenda: In-Memory Computation

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

**SAFARI**

# PIM: Concluding Remarks

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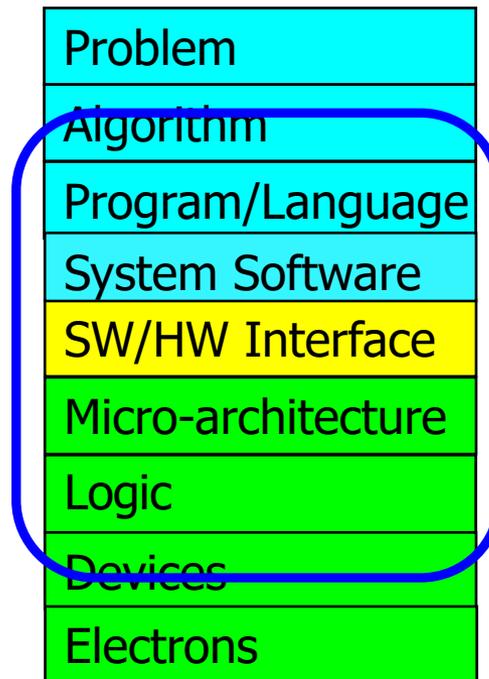
- It is time to design principled system architectures to solve the memory problem

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

- Enable computation capability inside and close to memory

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

# The Future of Processing in Memory is Bright

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

Can enable:

- Orders of magnitude improvements

- New applications and computing systems

| Problem |
| Algorithm |
| Program/Language |
| System Software |
| SW/HW Interface |
| Micro-architecture |
| Logic |
| Devices |
| Electrons |

Yet, we have to

- Think across the stack

- Design enabling systems

# We Need to Revisit the Entire Stack

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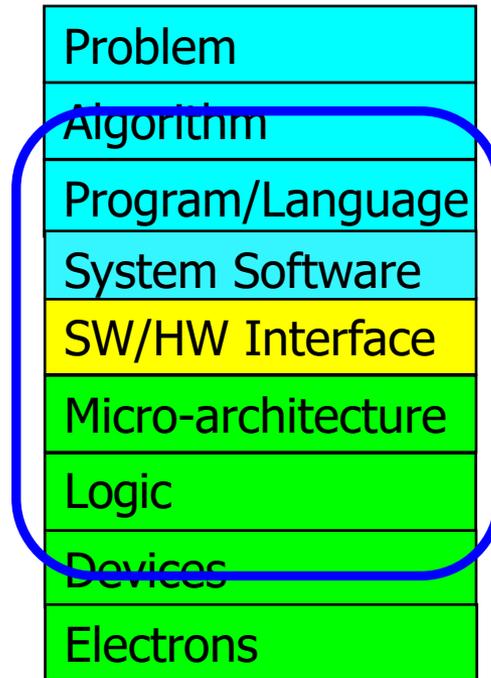
**We can get there step by step**

# If In Doubt, See Other Doubtful Technologies

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

INVITED PAPER

*Proceedings of the IEEE, Sept. 2017*

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

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

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

https://arxiv.org/pdf/1706.08642

# 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]

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

# Memory Systems
## and Memory-Centric Computing Systems

## Part 3: Computation in Memory

Prof. Onur Mutlu

omutlu@gmail.com

https://people.inf.ethz.ch/omutlu

7 July 2019

SAMOS Tutorial

**SAFARI**   **ETH** *zürich*   **Carnegie Mellon**