Memory Systems

and Memory-Centric Computing Systems

Lecture 4a: Processing-in-Memory II

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu

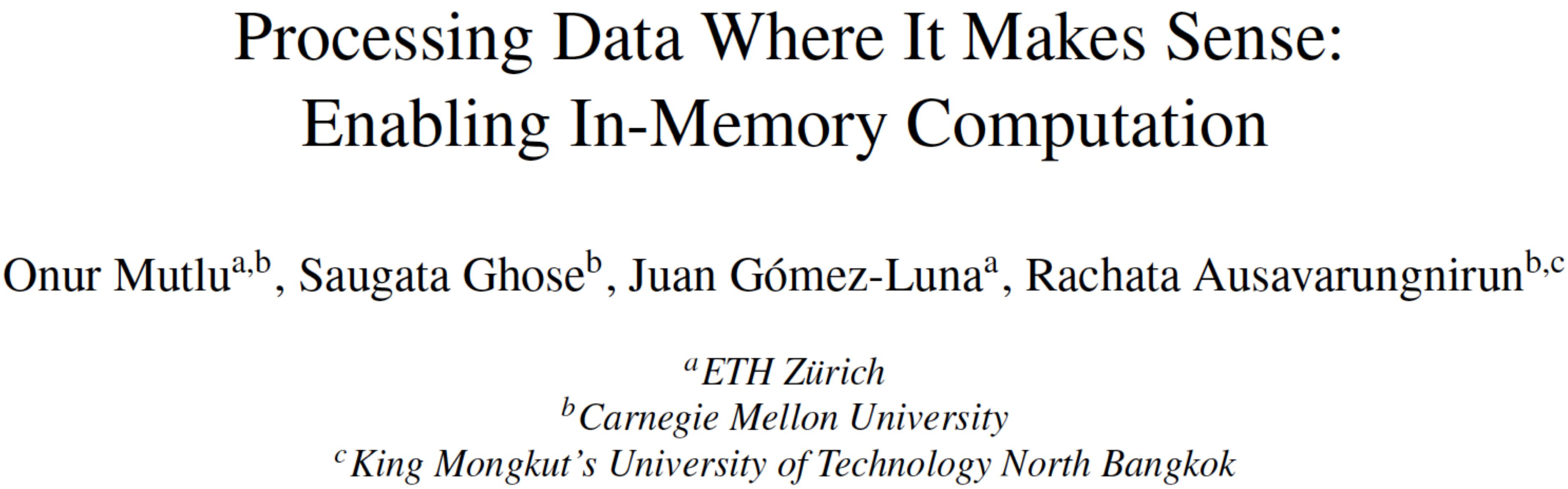
17 June 2019

TU Wien Fast Course 2019

SAFARI ETH zürich Carnegie Mellon

Required Review Papers (So Far)

Required Review Paper I



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

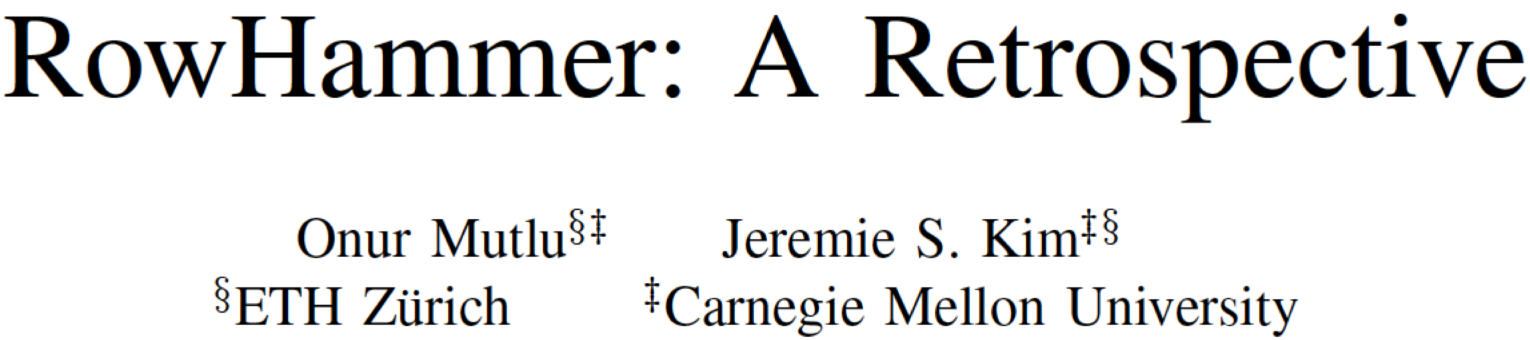
Onur Mutlu[a,b], Saugata Ghose[b], Juan Gómez-Luna[a], Rachata Ausavarungnirun[b,c]

[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]

Required Review Paper 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]

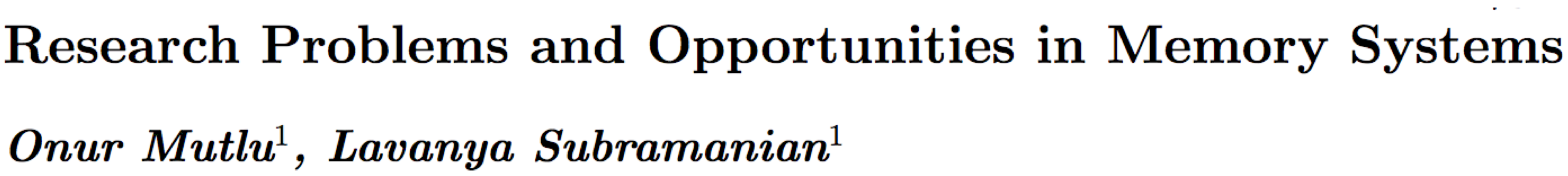


RowHammer: A Retrospective

Onur Mutlu[§‡] Jeremie S. Kim[‡§]
[§]ETH Zürich [‡]Carnegie Mellon University

Required Review Paper III

- Onur Mutlu and Lavanya Subramanian, **"Research Problems and Opportunities in Memory Systems"** *Invited Article in Supercomputing Frontiers and Innovations (**SUPERFRI**)*, 2014/2015.



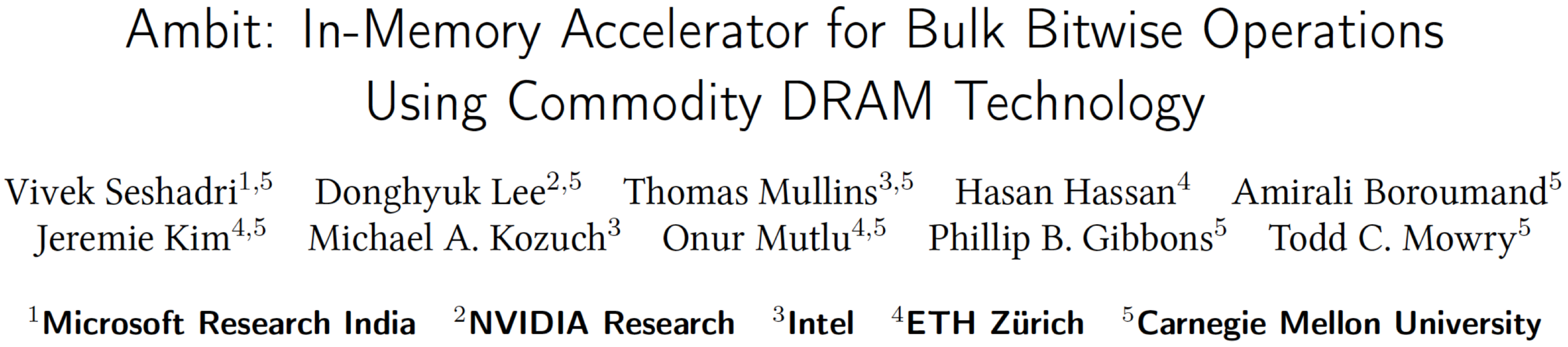
Research Problems and Opportunities in Memory Systems

Onur Mutlu[1], Lavanya Subramanian[1]

https://people.inf.ethz.ch/omutlu/pub/memory-systems-research_superfri14.pdf

Required Review Paper IV

- Vivek Seshadri et al., "**Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology**," MICRO 2017.



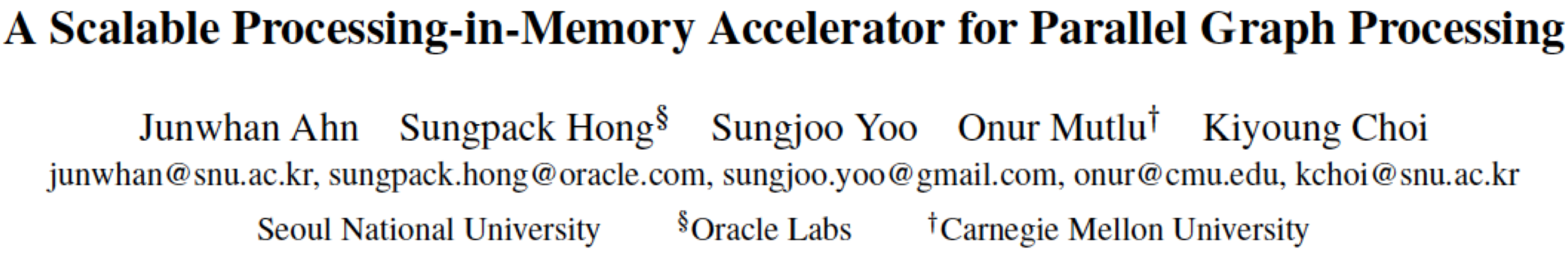
Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology

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**

Required Review Paper V

- 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)]

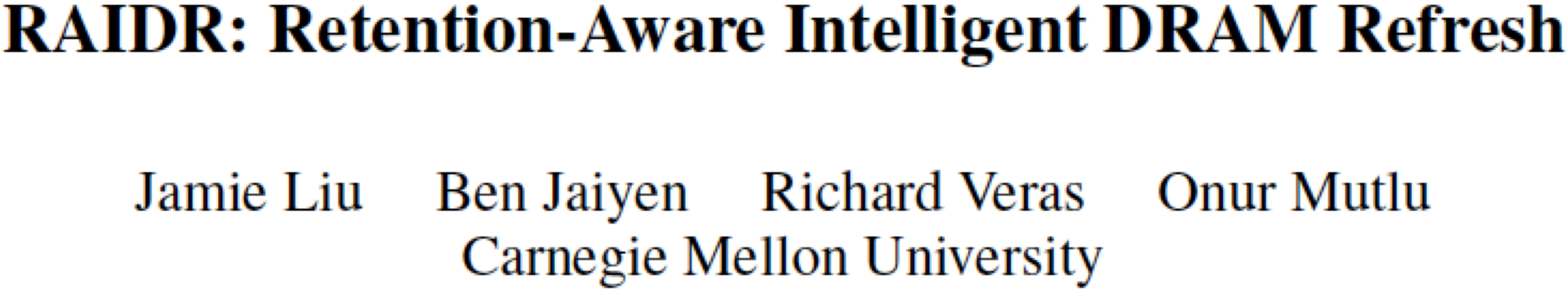


A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing

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

Required Review Paper VI

- Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu, **"RAIDR: Retention-Aware Intelligent DRAM Refresh"** *Proceedings of the 39th International Symposium on Computer Architecture* (***ISCA***), Portland, OR, June 2012. Slides (pdf)



RAIDR: Retention-Aware Intelligent DRAM Refresh

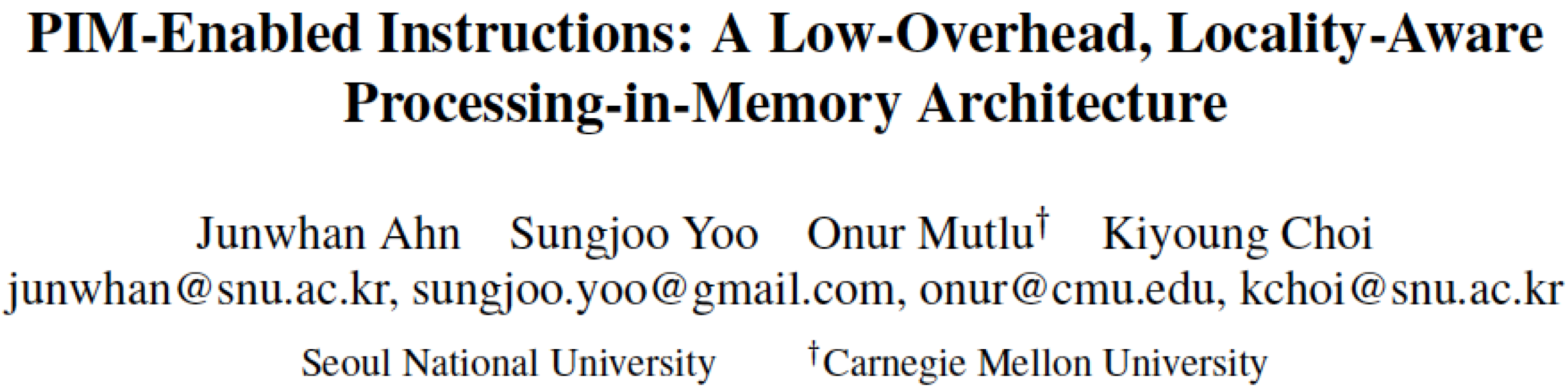
Jamie Liu   Ben Jaiyen   Richard Veras   Onur Mutlu
Carnegie Mellon University

End of Required Review Papers

Optional Review Papers (So Far)

Optional Review Paper I

- 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

Optional Review Paper II

- 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]

Optional Review Paper III

- 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]

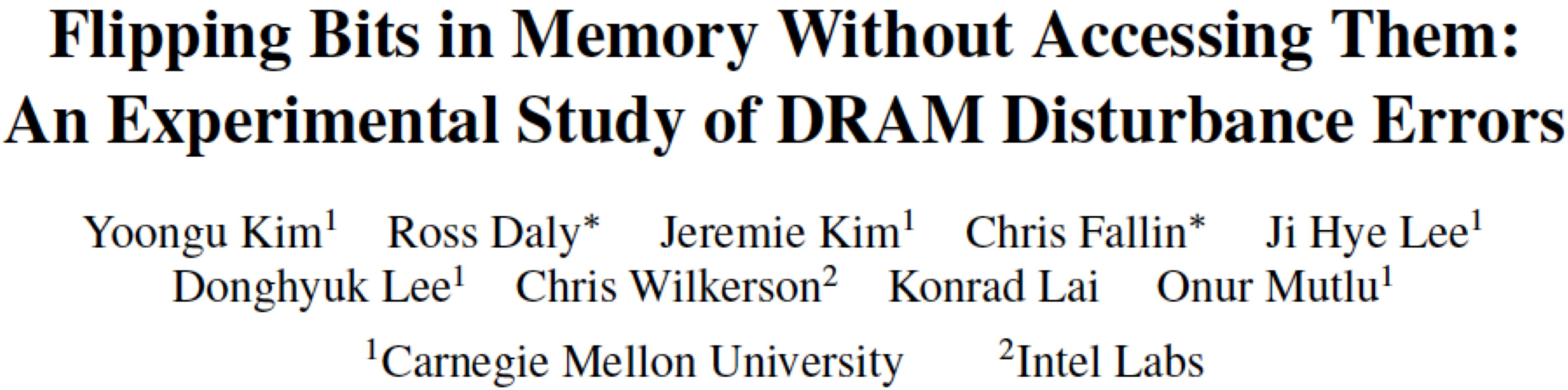
In-DRAM Bulk Bitwise Execution Engine

Vivek Seshadri
Microsoft Research India
visesha@microsoft.com

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

Optional Review Paper IV

- Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu, **"Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors"** *Proceedings of the 41st International Symposium on Computer Architecture* (***ISCA***), Minneapolis, MN, June 2014. [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Source Code and Data]



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

Yoongu Kim[1] Ross Daly* Jeremie Kim[1] Chris Fallin* Ji Hye Lee[1]
Donghyuk Lee[1] Chris Wilkerson[2] Konrad Lai Onur Mutlu[1]

[1]Carnegie Mellon University [2]Intel Labs

Optional Review Paper V

- Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana, **"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"** *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 39-50, Beijing, China, June 2008. Slides (pptx)

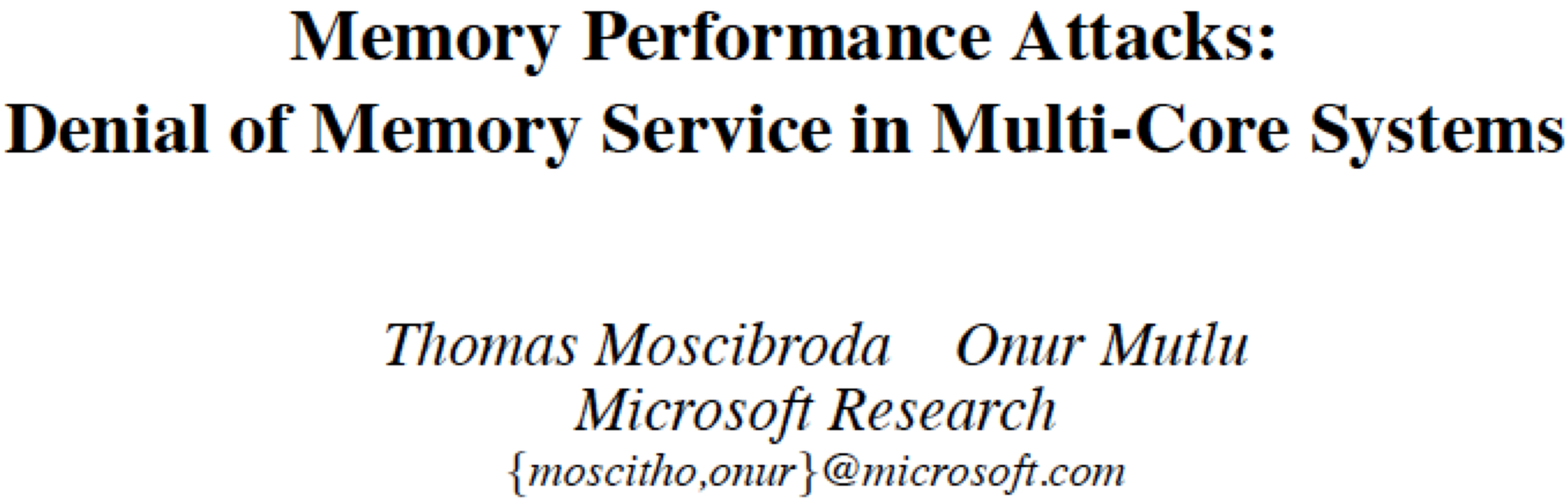
Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

Engin İpek[1,2] Onur Mutlu[2] José F. Martínez[1] Rich Caruana[1]

[1]Cornell University, Ithaca, NY 14850 USA
[2] Microsoft Research, Redmond, WA 98052 USA

Optional Review Paper VI

- Thomas Moscibroda and Onur Mutlu,
"Memory Performance Attacks: Denial of Memory Service in Multi-Core Systems"
Proceedings of the 16th USENIX Security Symposium (***USENIX SECURITY***), pages 257-274, Boston, MA, August 2007. Slides (ppt)

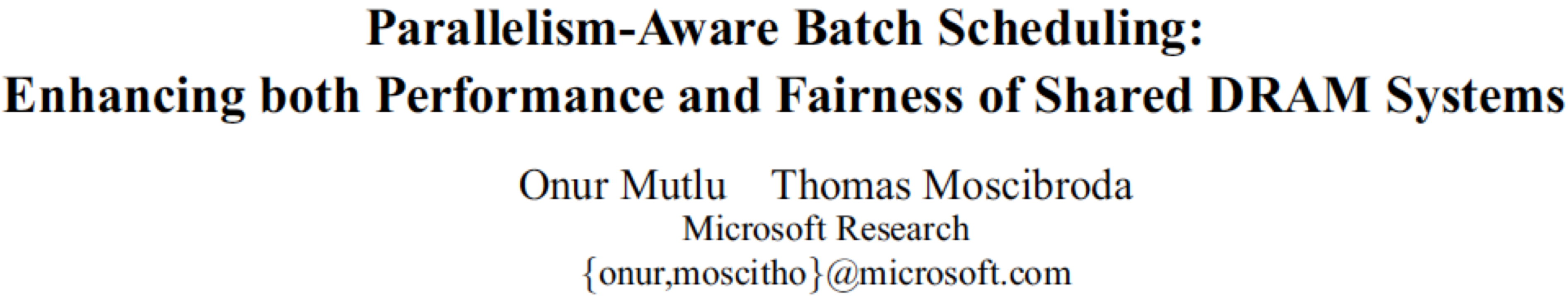


Memory Performance Attacks:
Denial of Memory Service in Multi-Core Systems

Thomas Moscibroda Onur Mutlu
Microsoft Research
{moscitho,onur}@microsoft.com

Optional Review Paper VII

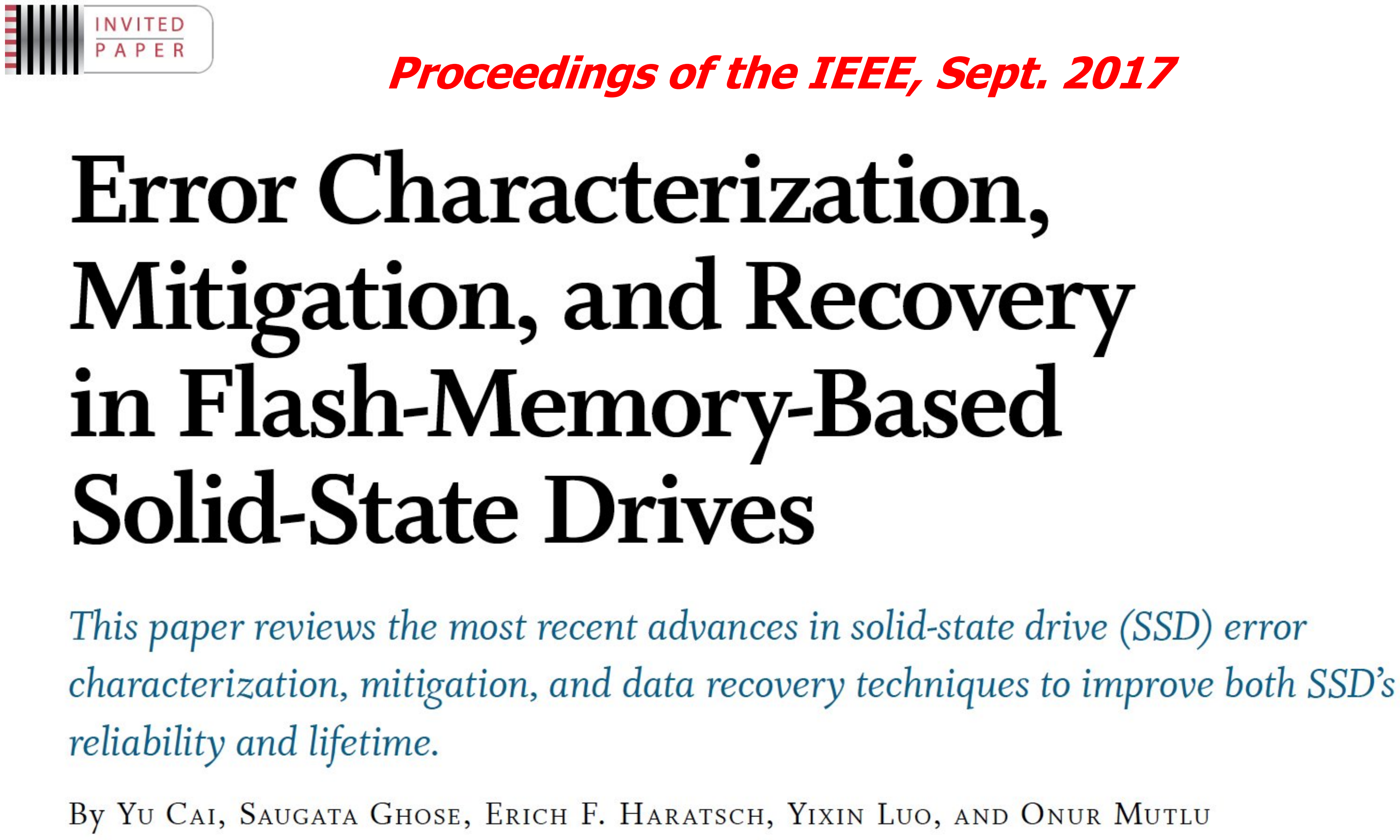
- Onur Mutlu and Thomas Moscibroda, **"Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems"** *Proceedings of the 35th International Symposium on Computer Architecture (**ISCA**)*, pages 63-74, Beijing, China, June 2008. [Summary] [Slides (ppt)]



Parallelism-Aware Batch Scheduling:
Enhancing both Performance and Fairness of Shared DRAM Systems

Onur Mutlu   Thomas Moscibroda
Microsoft Research
{onur,moscitho}@microsoft.com

Optional Review Paper VIII



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.

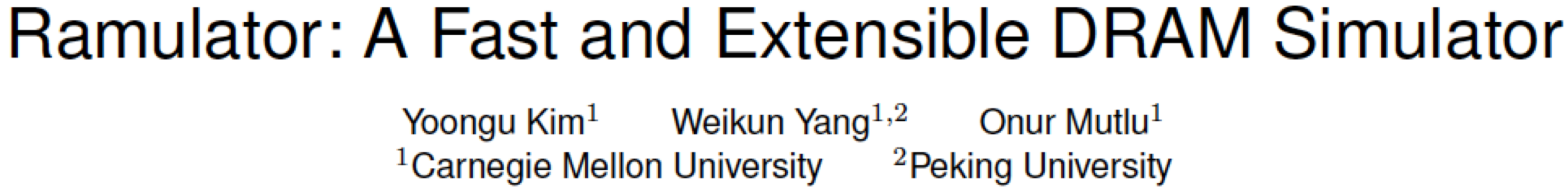
By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

https://arxiv.org/pdf/1706.08642

Optional Review Paper IX

- Yoongu Kim, Weikun Yang, and Onur Mutlu, **"Ramulator: A Fast and Extensible DRAM Simulator"** *IEEE Computer Architecture Letters* ***(CAL)***, March 2015. [Source Code]

- Source code is released under the liberal MIT License
  - https://github.com/CMU-SAFARI/ramulator

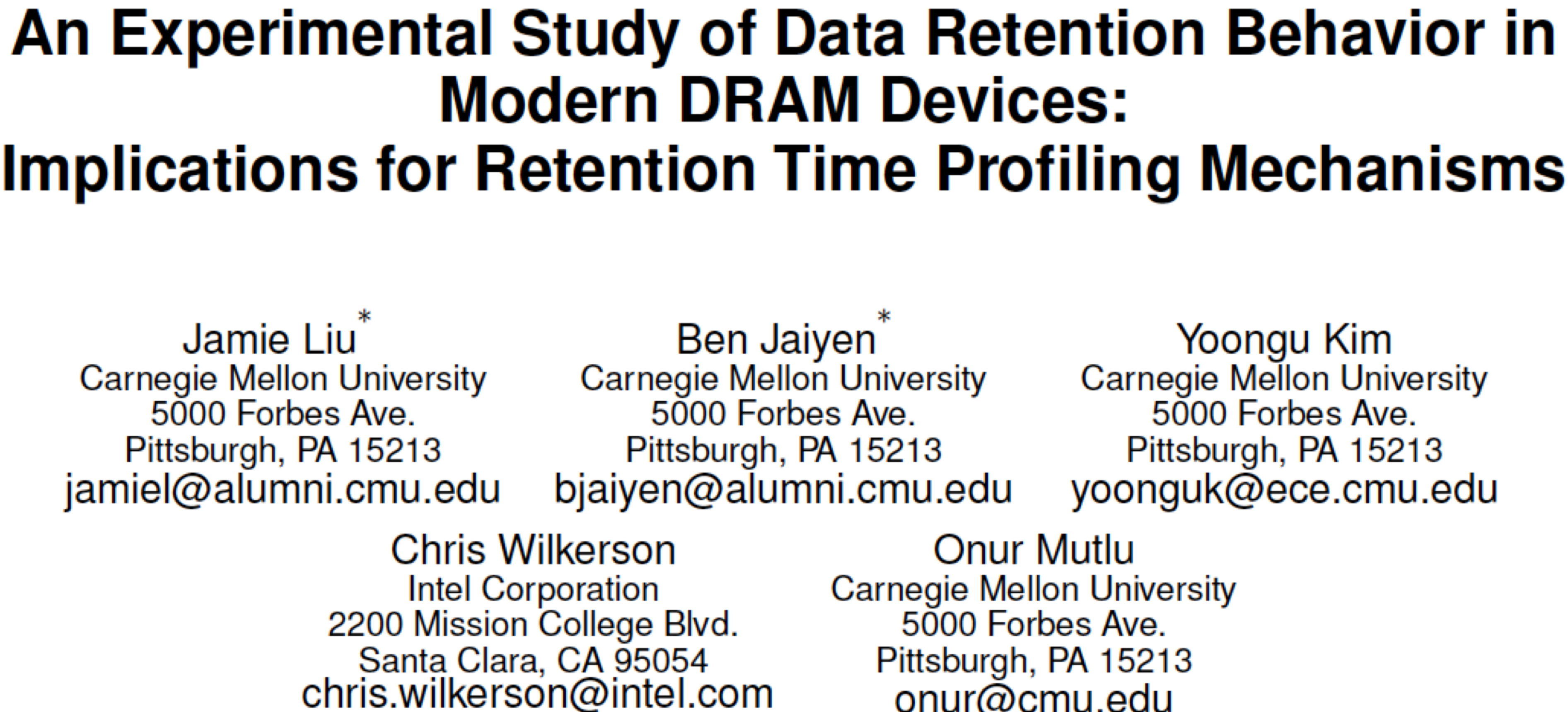


Ramulator: A Fast and Extensible DRAM Simulator

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

Optional Review Paper X

- Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu, **"An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms"** *Proceedings of the 40th International Symposium on Computer Architecture* (***ISCA***), Tel-Aviv, Israel, June 2013. Slides (ppt) Slides (pdf)



An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms

Jamie Liu*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
jamiel@alumni.cmu.edu

Ben Jaiyen*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
bjaiyen@alumni.cmu.edu

Yoongu Kim
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
yoonguk@ece.cmu.edu

Chris Wilkerson
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95054
chris.wilkerson@intel.com

Onur Mutlu
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
onur@cmu.edu

Optional Review Paper XI

- Minesh Patel, Jeremie S. Kim, and Onur Mutlu,
  "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions"
  *Proceedings of the 44th International Symposium on Computer Architecture (**ISCA**)*, Toronto, Canada, June 2017.
  [Slides (pptx) (pdf)]
  [Lightning Session Slides (pptx) (pdf)]

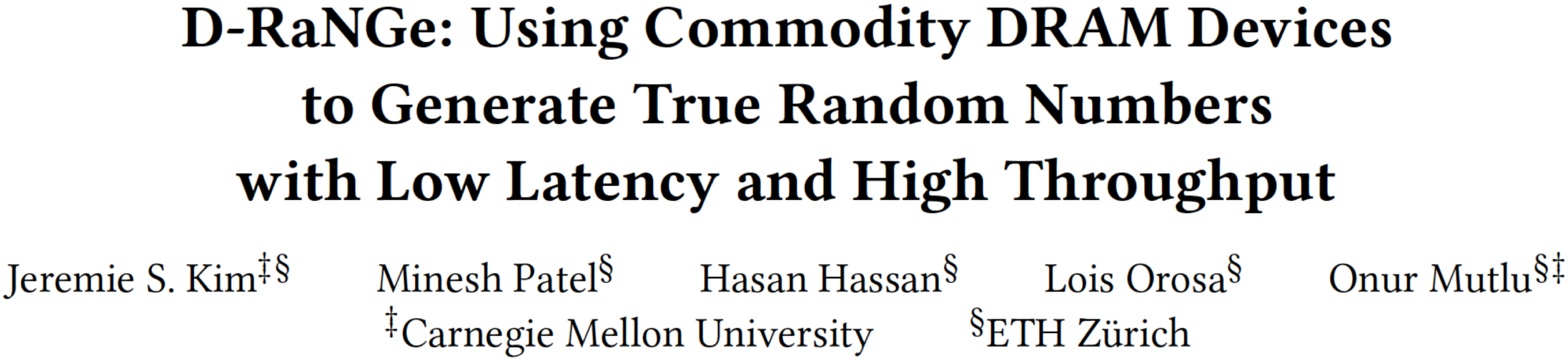
- First experimental analysis of (mobile) LPDDR4 chips
- Analyzes the complex tradeoff space of retention time profiling
- Idea: enable fast and robust profiling at higher refresh intervals & temperatures

The Reach Profiler (REAPER):
Enabling the Mitigation of DRAM Retention Failures
via Profiling at Aggressive Conditions

Minesh Patel[§‡] Jeremie S. Kim[‡§] Onur Mutlu[§‡]
[§]ETH Zürich [‡]Carnegie Mellon University

Optional Review Paper XII

- Jeremie S. Kim, Minesh Patel, Hasan Hassan, Lois Orosa, and Onur Mutlu, **"D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput"** *Proceedings of the 25th International Symposium on High-Performance Computer Architecture* (***HPCA***), Washington, DC, USA, February 2019.
  [Slides (pptx) (pdf)]
  [Full Talk Video (21 minutes)]



D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput

Jeremie S. Kim[‡§] Minesh Patel[§] Hasan Hassan[§] Lois Orosa[§] Onur Mutlu[§‡]
[‡]Carnegie Mellon University [§]ETH Zürich

Other Optional Reviews: Many

- There are too many to list…
  - What I provided is just a sampling…
- Many other important references are in lecture slides.
- I would recommend rigorously reading as many as possible.
- Recall that "Chance favors the prepared mind."
  - Critical rigorous analysis of key papers is great preparation
- You can use my website as a resource for papers & artifacts
  - https://people.inf.ethz.ch/omutlu/projects.htm

End of Optional Review Papers (So Far)

Challenge and Opportunity for Future

Computing Architectures

with

Minimal Data Movement

Agenda

- 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

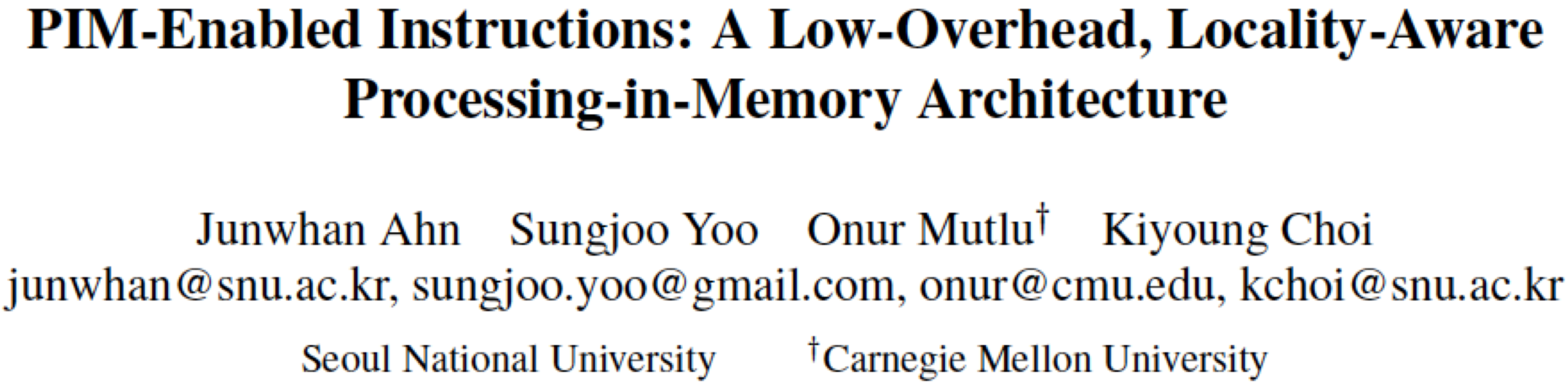
Several Questions in 3D-Stacked PIM

- 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 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

PEI: PIM-Enabled Instructions (Ideas)

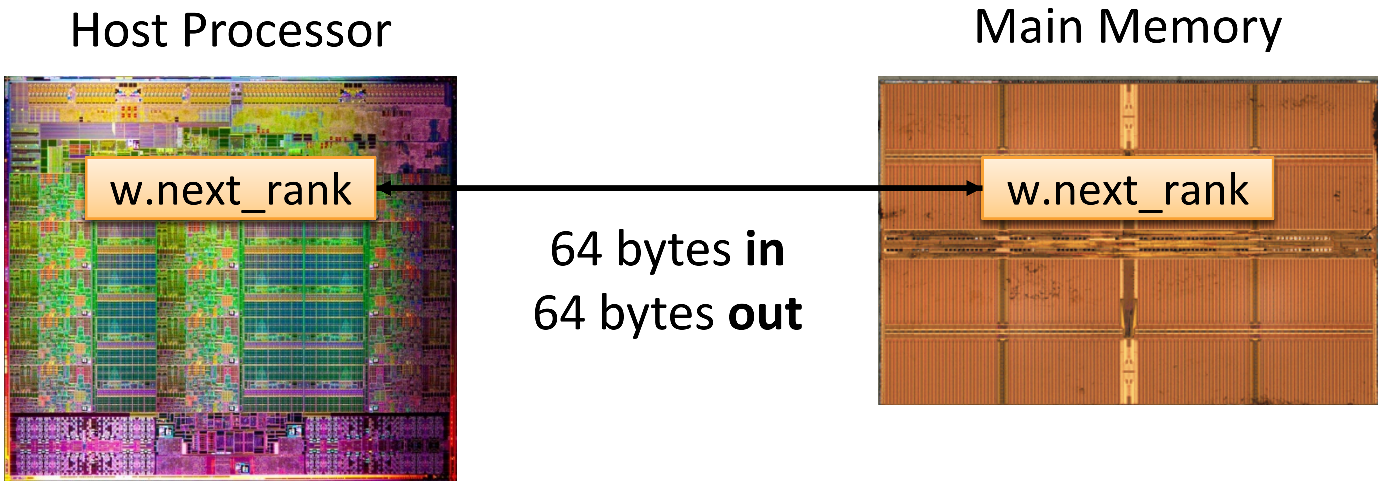
- Goal: Develop mechanisms to get the most out of near-data processing with minimal cost, minimal changes to the system, no changes to the programming model

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

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

Simple PIM Operations as ISA Extensions (II)

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



Host Processor

Main Memory

w.next_rank

w.next_rank

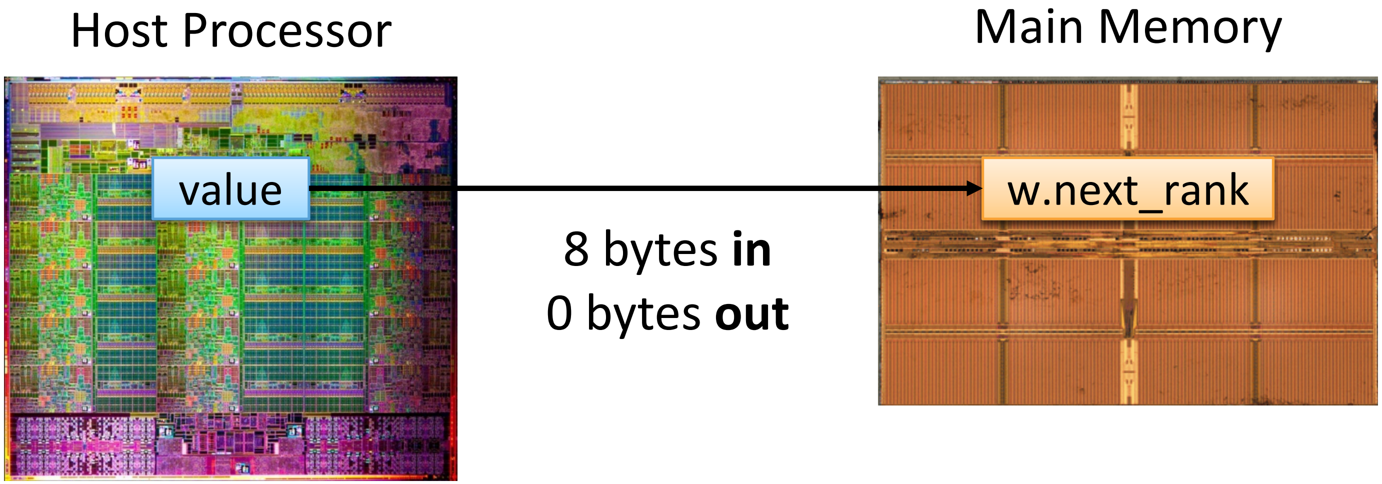
64 bytes **in**
64 bytes **out**

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

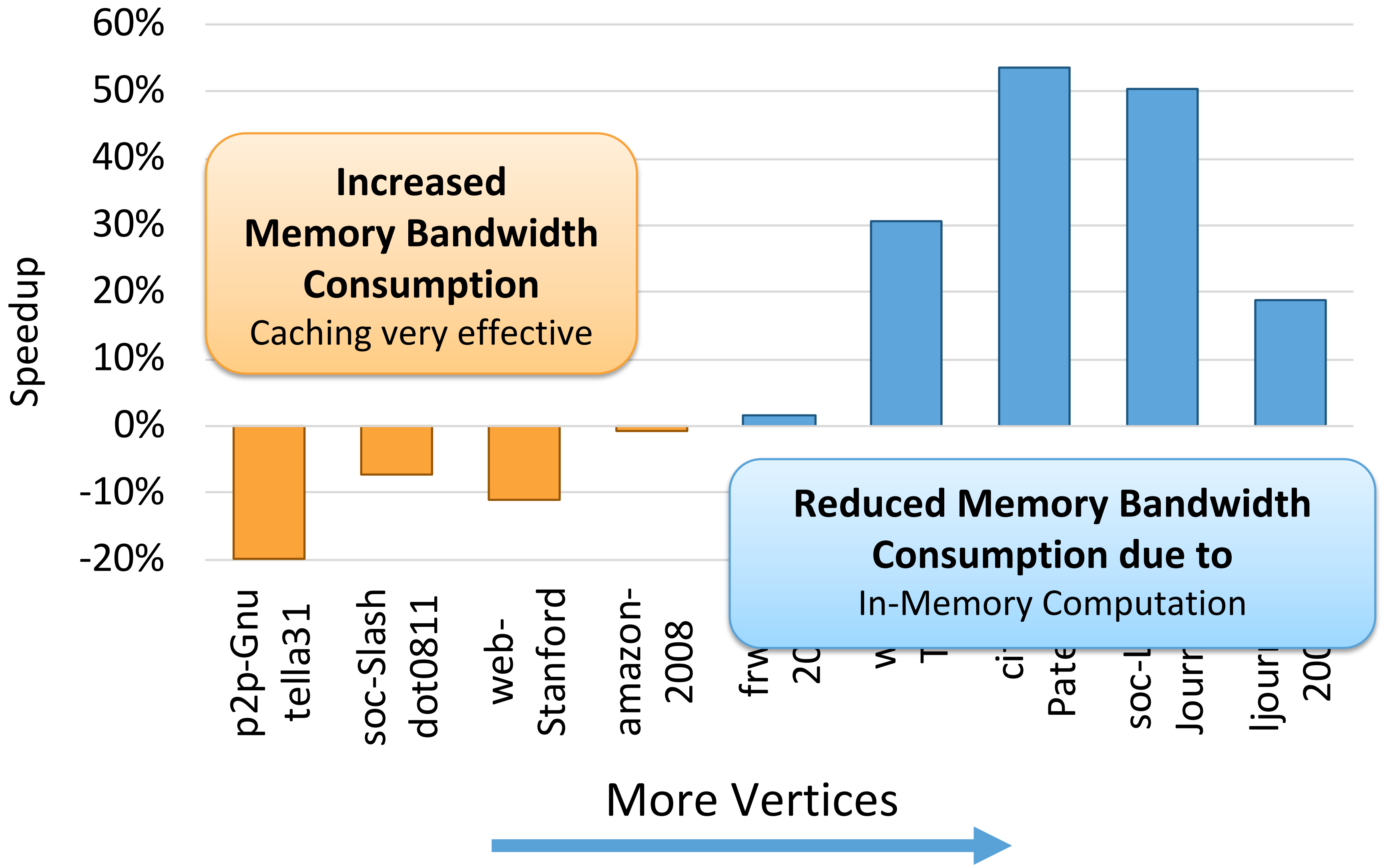
w.next_rank

8 bytes **in**

0 bytes **out**

In-Memory Addition

Always Executing in Memory? Not A Good Idea



Increased Memory Bandwidth Consumption

Caching very effective

Reduced Memory Bandwidth Consumption due to

In-Memory Computation

More Vertices

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

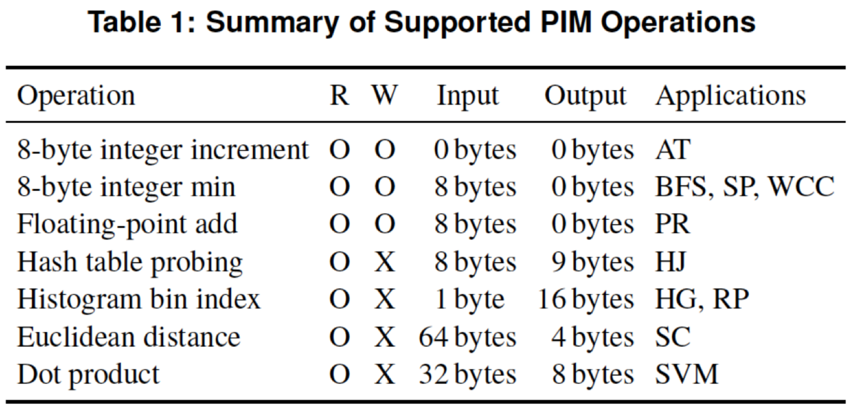


Table 1: Summary of Supported PIM Operations

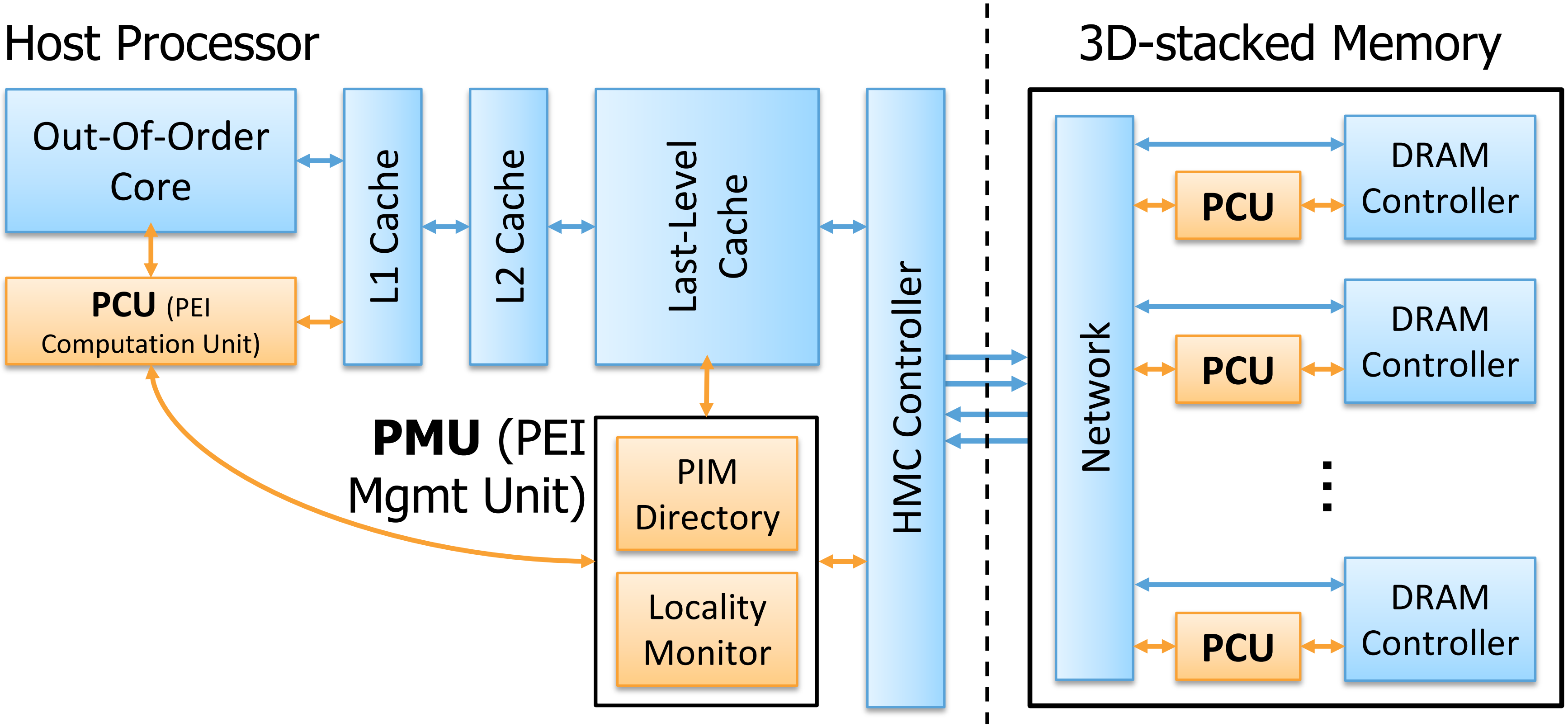
Operation	R	W	Input	Output	Applications
8-byte integer increment	O	O	0 bytes	0 bytes	AT
8-byte integer min	O	O	8 bytes	0 bytes	BFS, SP, WCC
Floating-point add	O	O	8 bytes	0 bytes	PR
Hash table probing	O	X	8 bytes	9 bytes	HJ
Histogram bin index	O	X	1 byte	16 bytes	HG, RP
Euclidean distance	O	X	64 bytes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

Example (Abstract) PEI uArchitecture



Example PEI uArchitecture

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

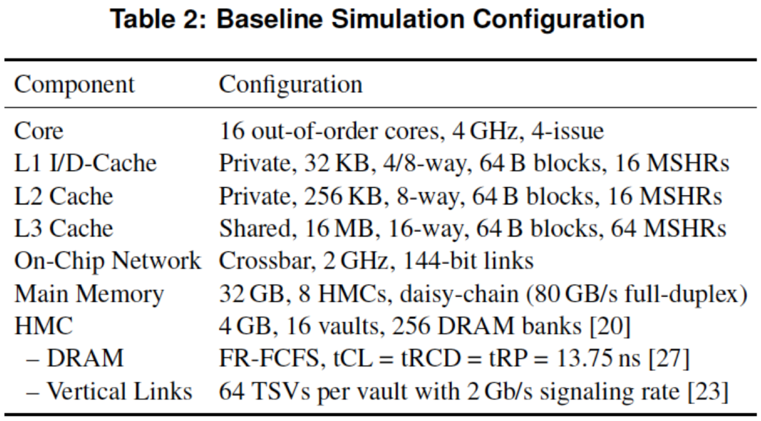


Table 2: Baseline Simulation Configuration

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

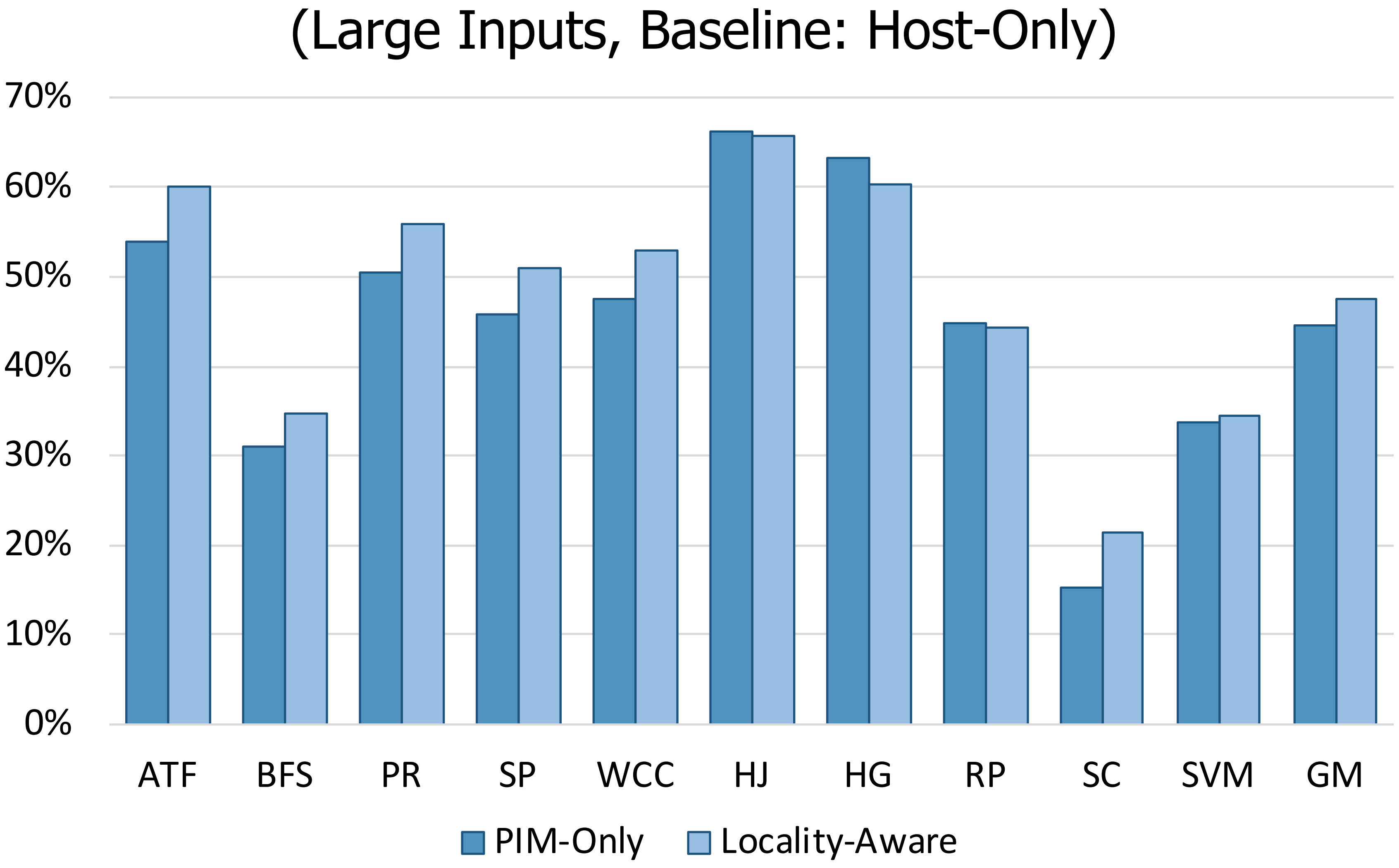
- **Performance Improvement and Energy Reduction:**
  - 47% average speedup with large input data sets
  - 32% speedup with small input data sets
  - 25% avg. energy reduction in a single node with large input data sets

Evaluated Data-Intensive Applications

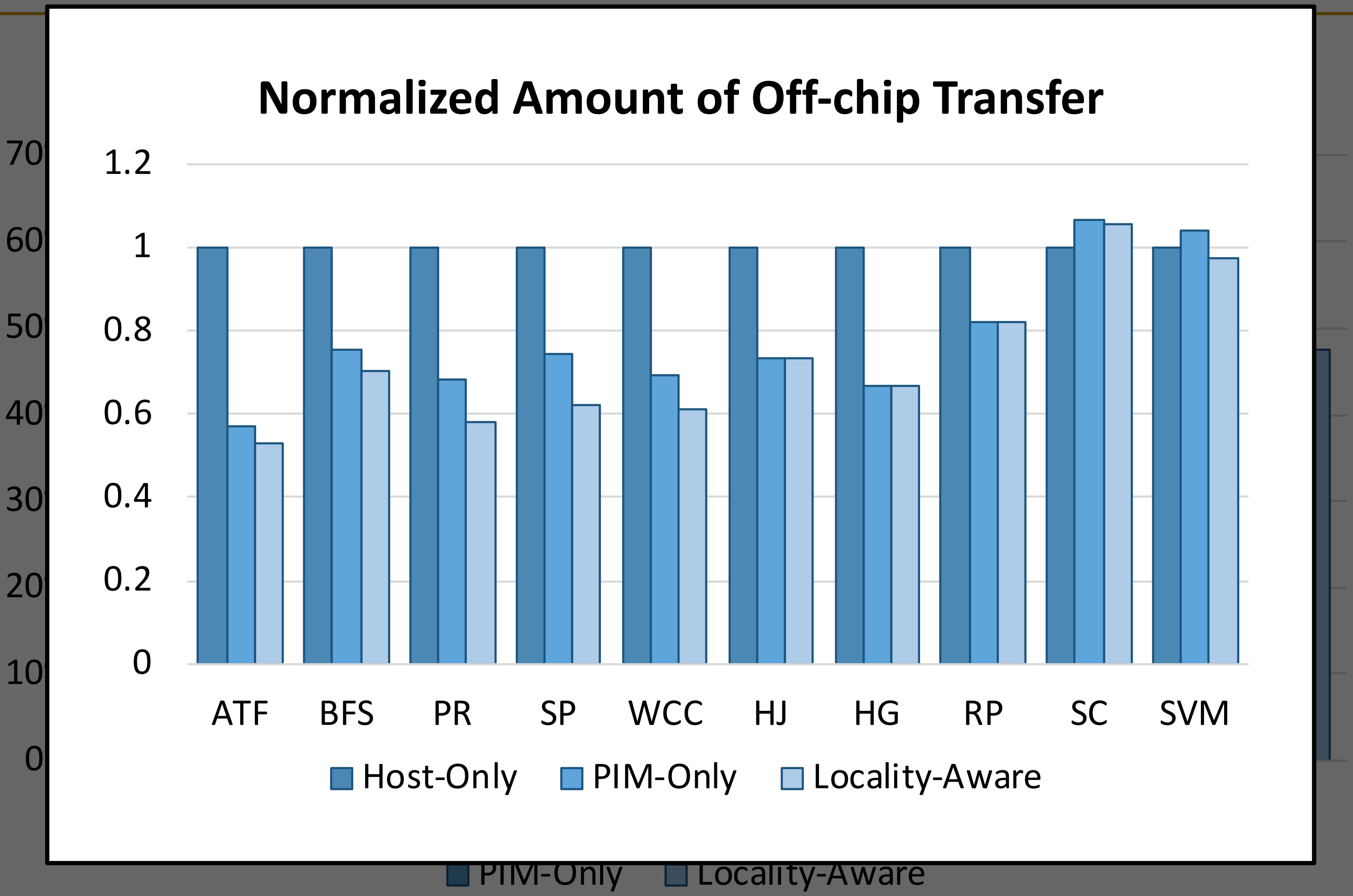
- Ten emerging data-intensive workloads
  - Large-scale graph processing
    - Average teenage follower, BFS, PageRank, single-source shortest path, weakly connected components
  - In-memory data analytics
    - Hash join, histogram, radix partitioning
  - Machine learning and data mining
    - Streamcluster,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

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

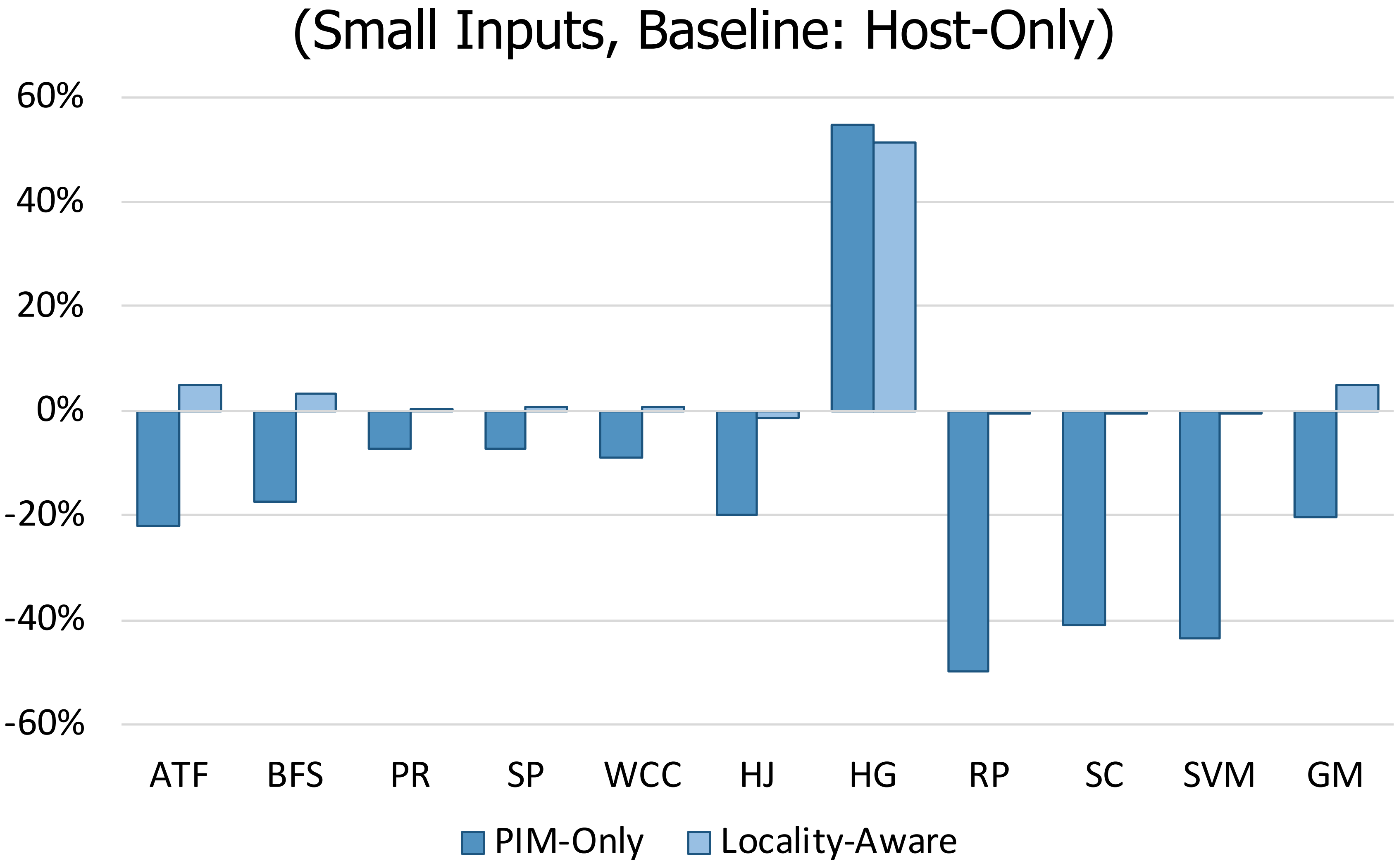
PEI Performance Delta: Large Data Sets



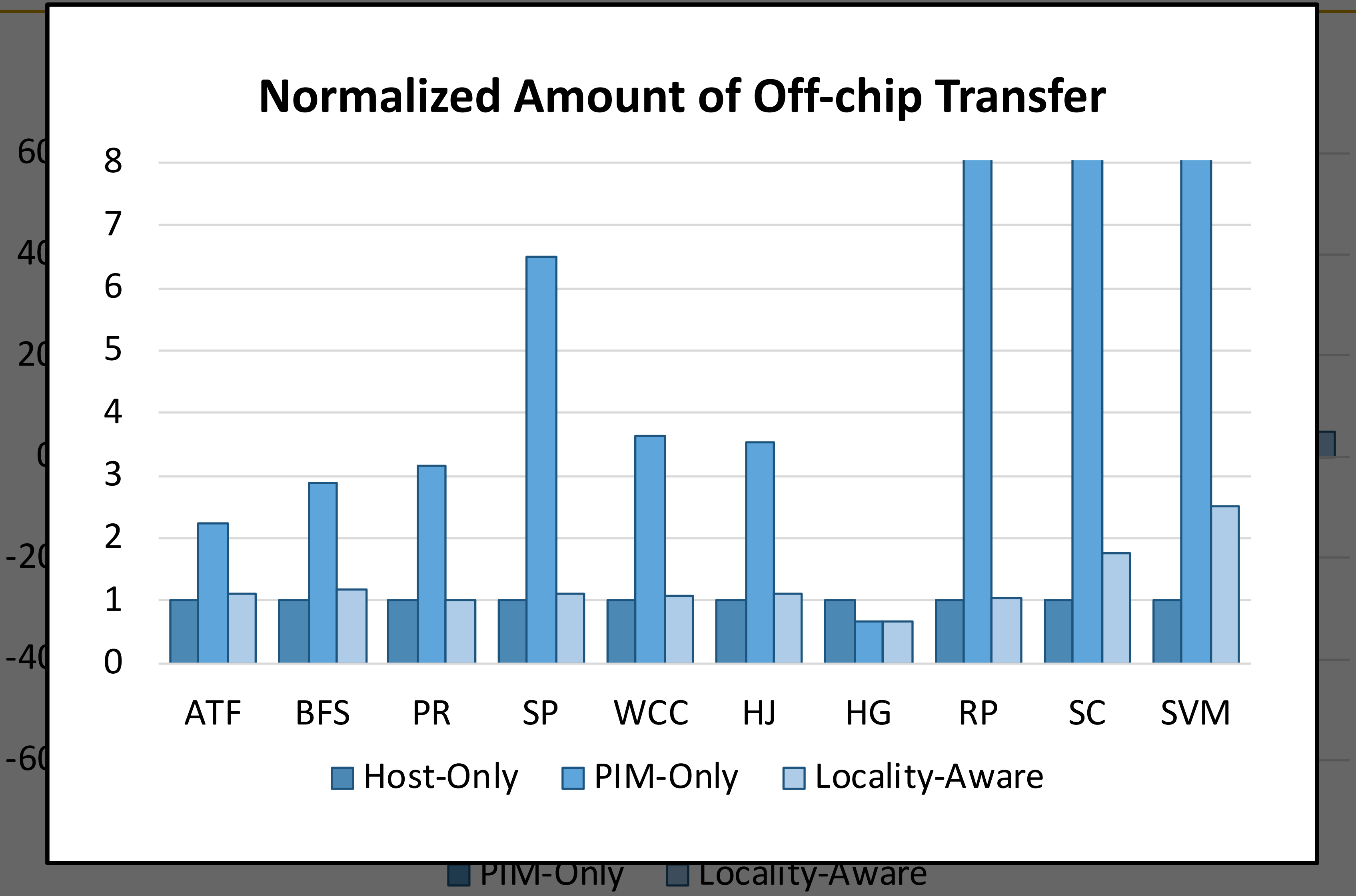
PEI Performance: Large Data Sets



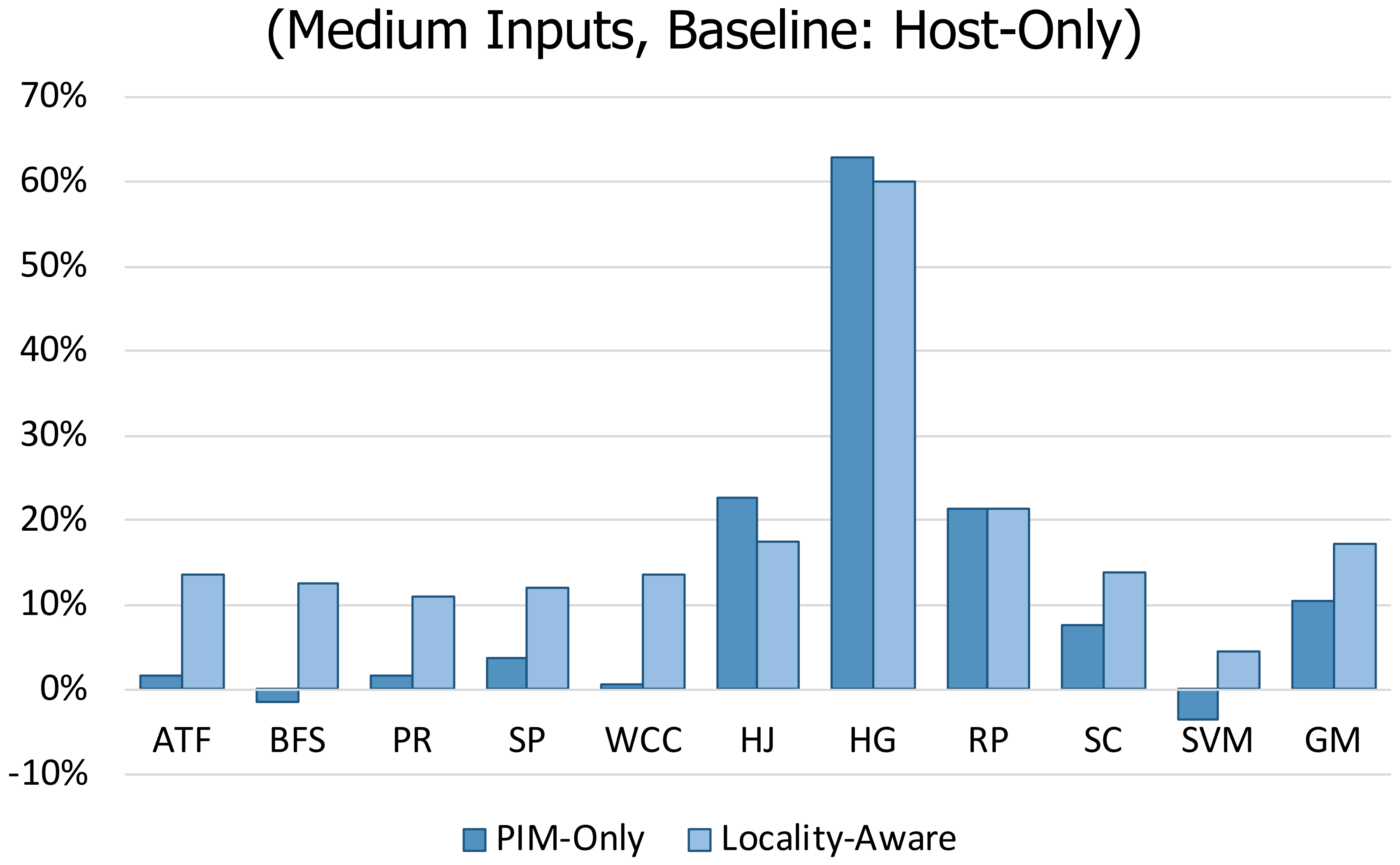
PEI Performance Delta: Small Data Sets



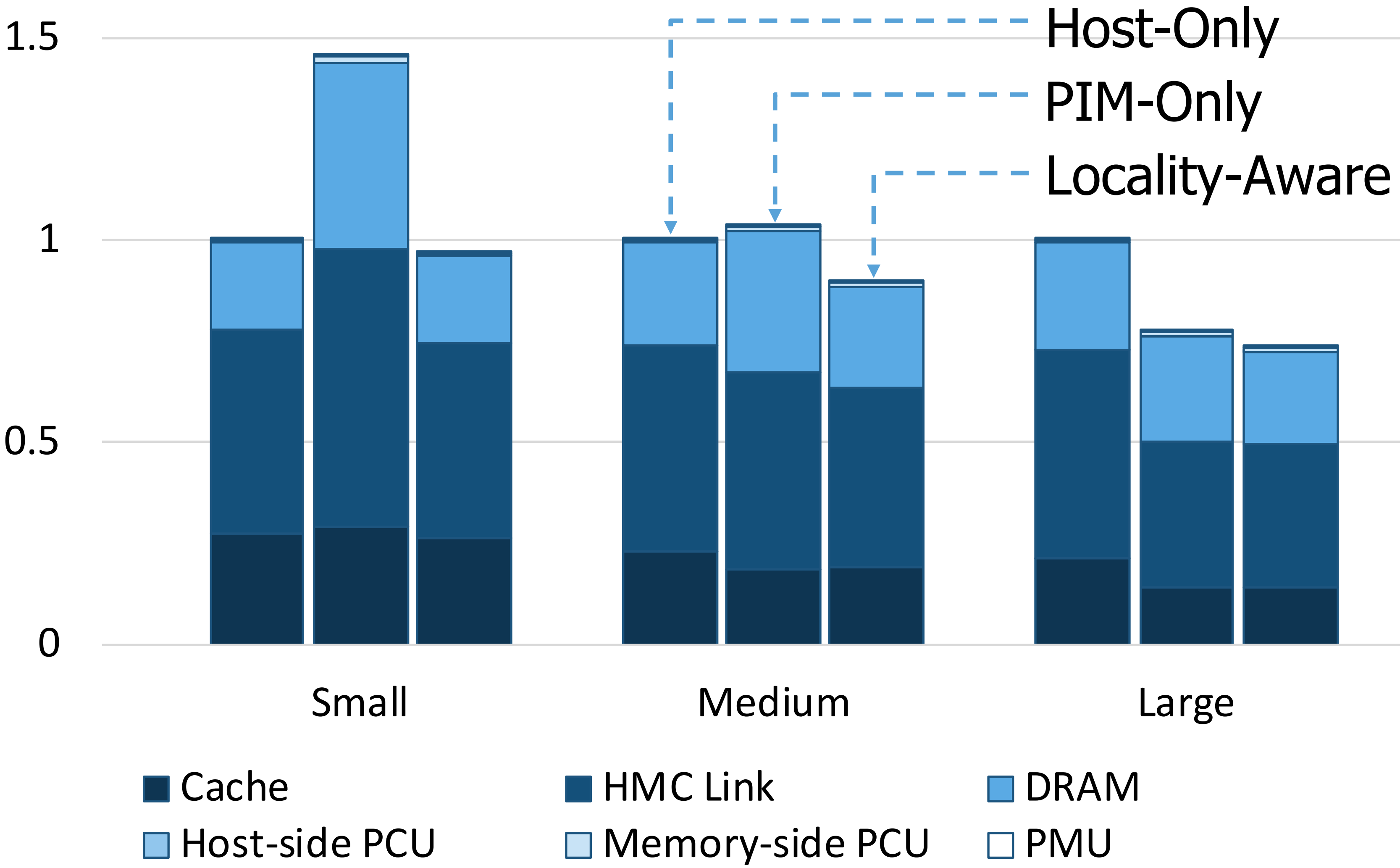
PEI Performance: Small Data Sets



PEI Performance Delta: Medium Data Sets



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

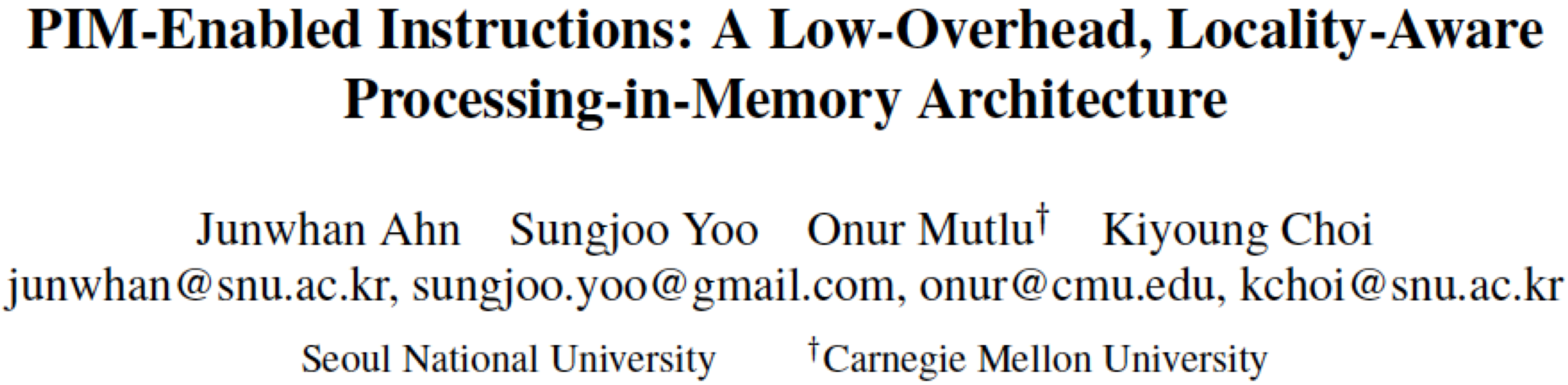
- Disadvantages

  - 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)]

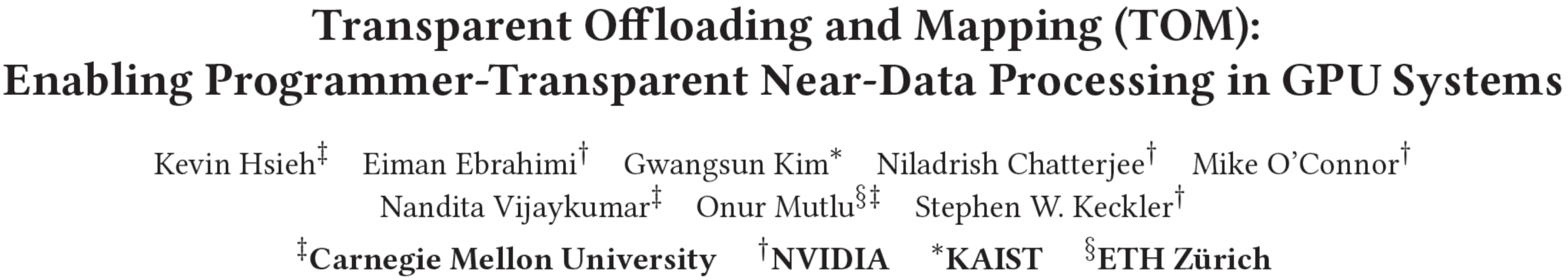


PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture

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

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)]

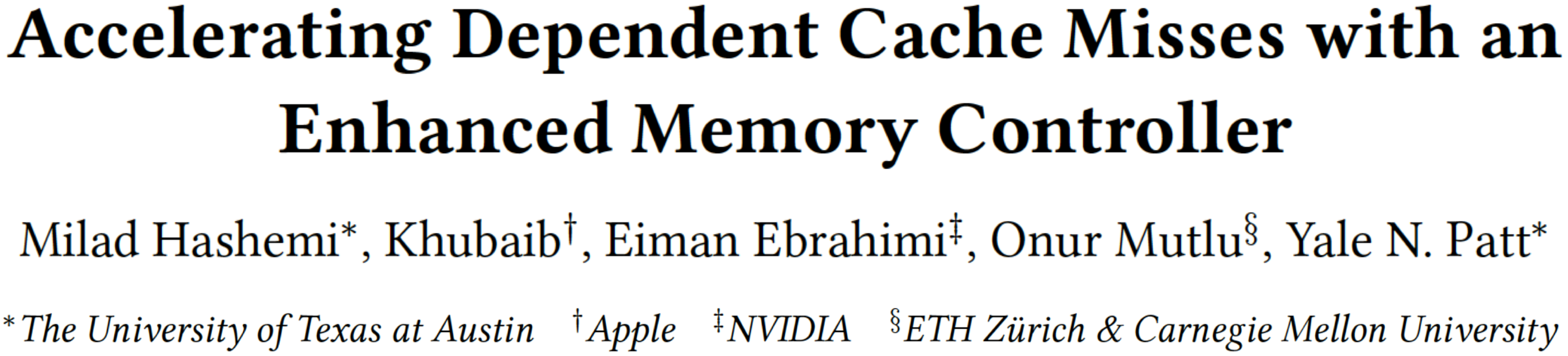


Transparent Offloading and Mapping (TOM):
Enabling Programmer-Transparent Near-Data Processing in GPU Systems

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)]

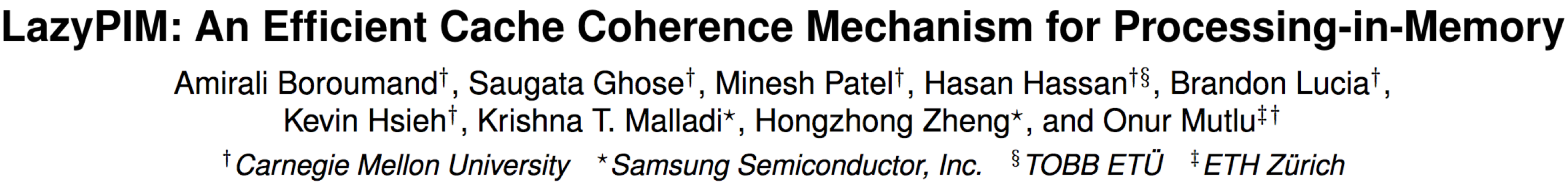
Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads

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.



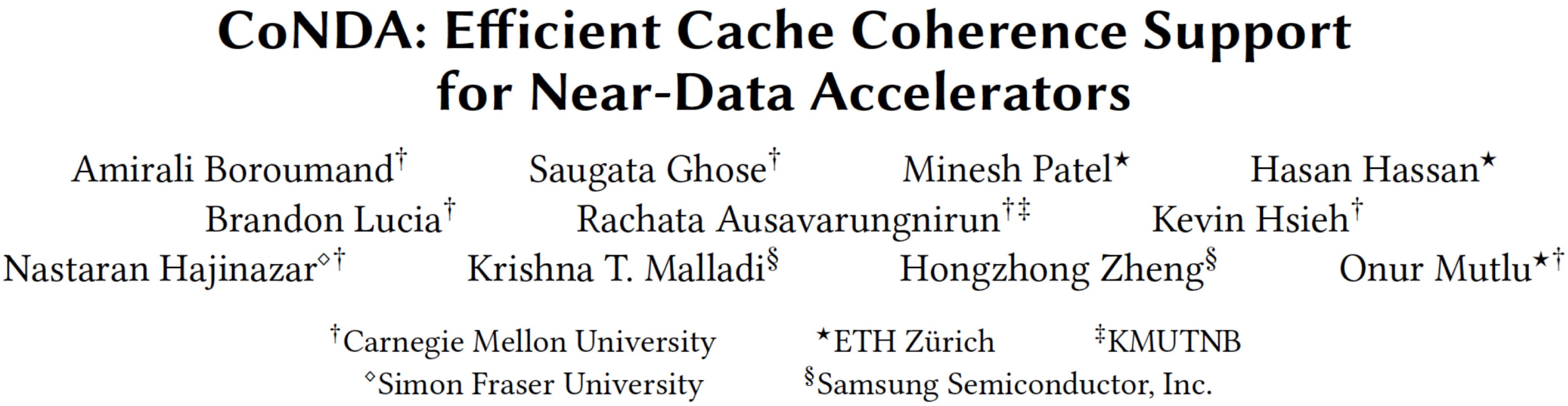
LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory

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.

Challenge and Opportunity for Future

Fundamentally

Energy-Efficient

(Data-Centric)

Computing Architectures

Challenge and Opportunity for Future

Fundamentally

High-Performance

(Data-Centric)

Computing Architectures

Challenge and Opportunity for Future

Computing Architectures

with

Minimal Data Movement

Agenda

- 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 for PIM

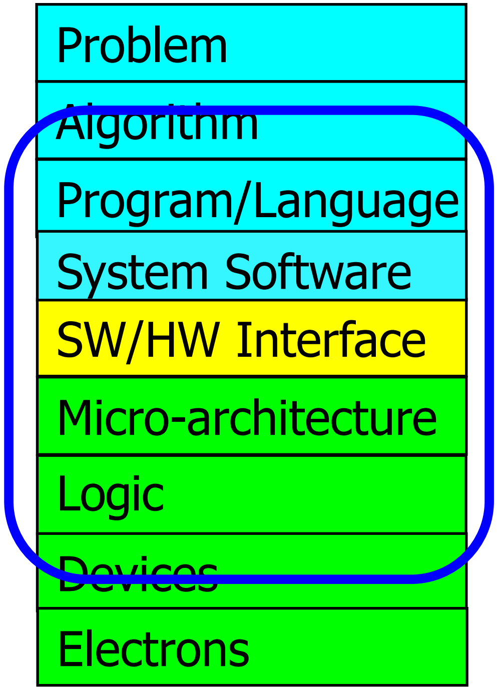
2. Ease of programming (interfaces and compiler/HW support)

3. System support: coherence & virtual memory

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

5. Infrastructures to assess benefits and feasibility

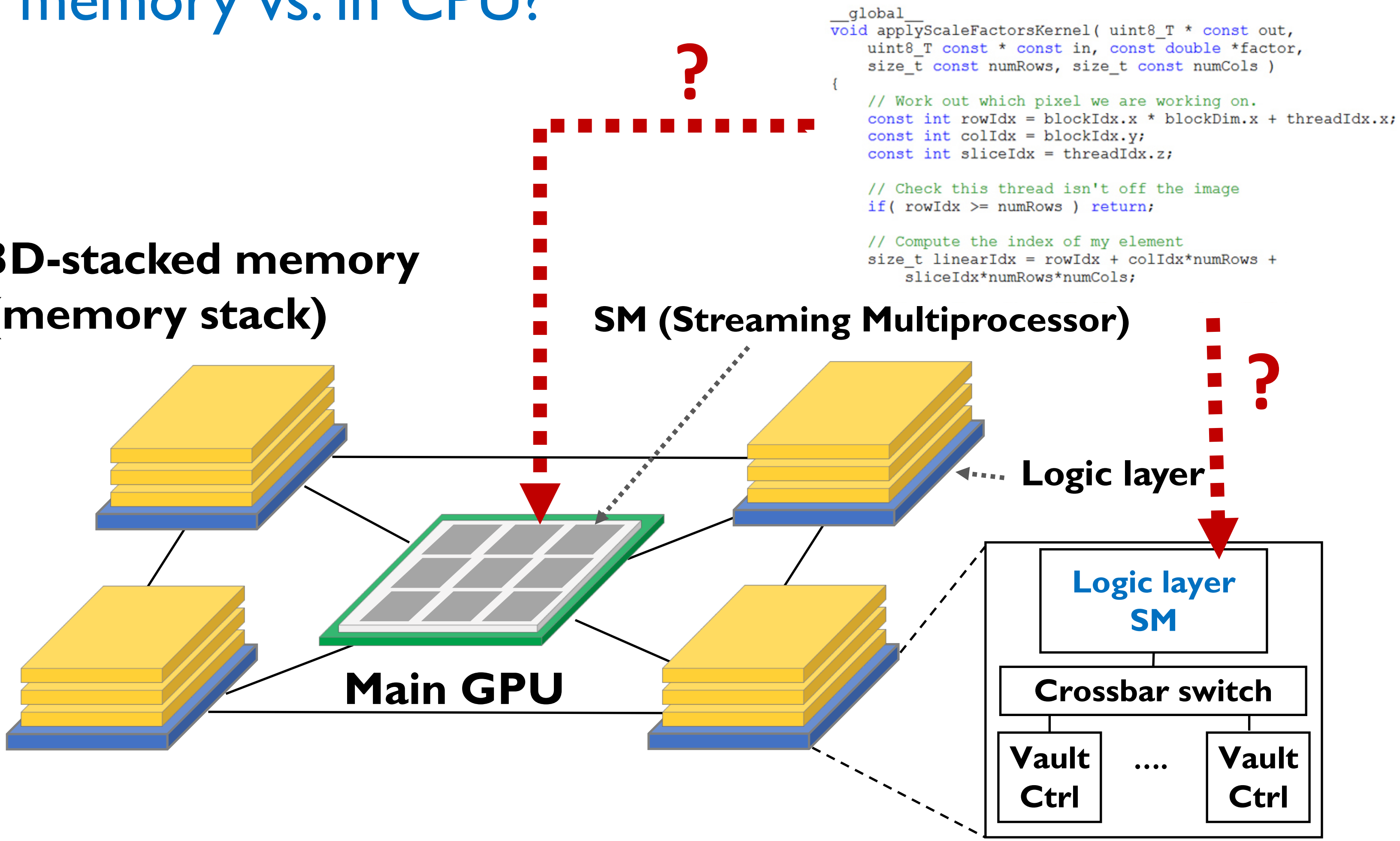
We Need to Revisit the Entire Stack



We can get there step by step

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

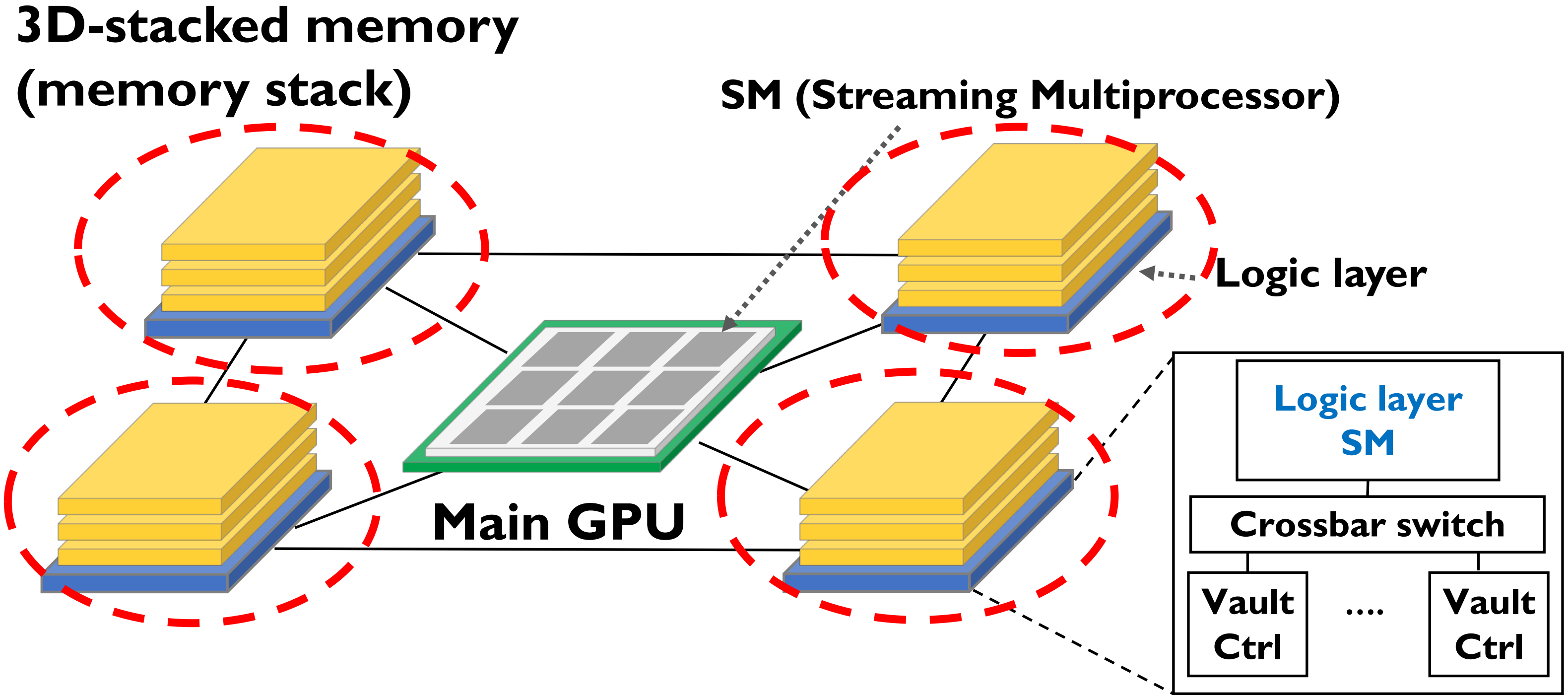
Vault Ctrl

....

Vault Ctrl

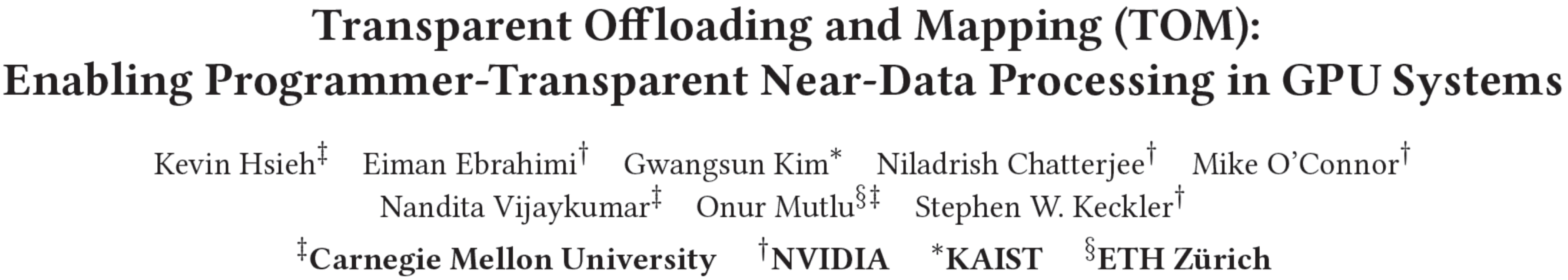
Key Challenge 2: Data Mapping

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



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)]

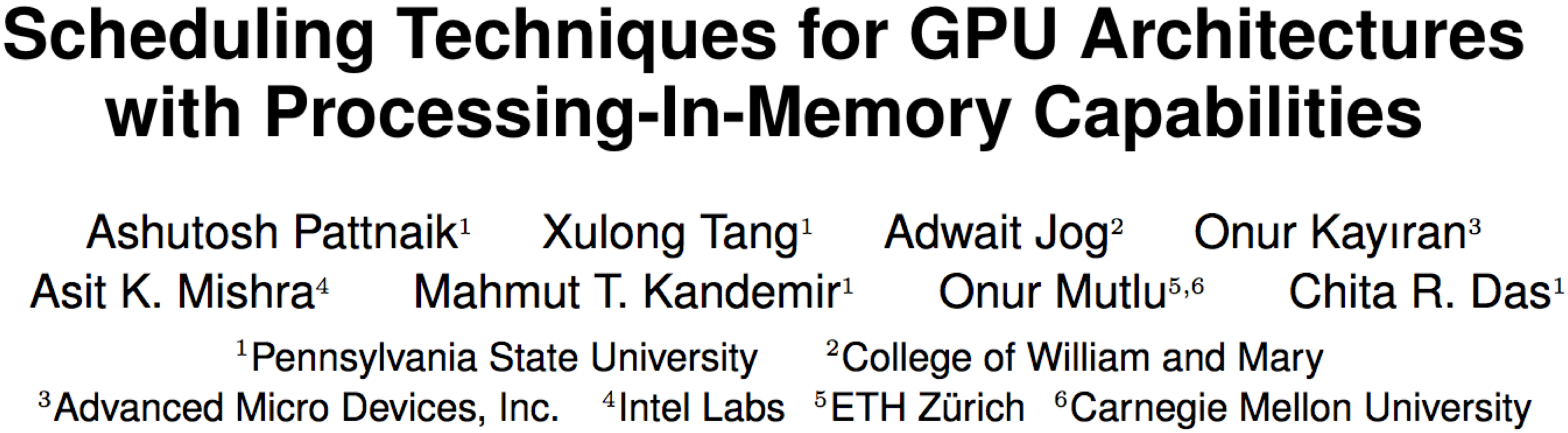


Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems

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?

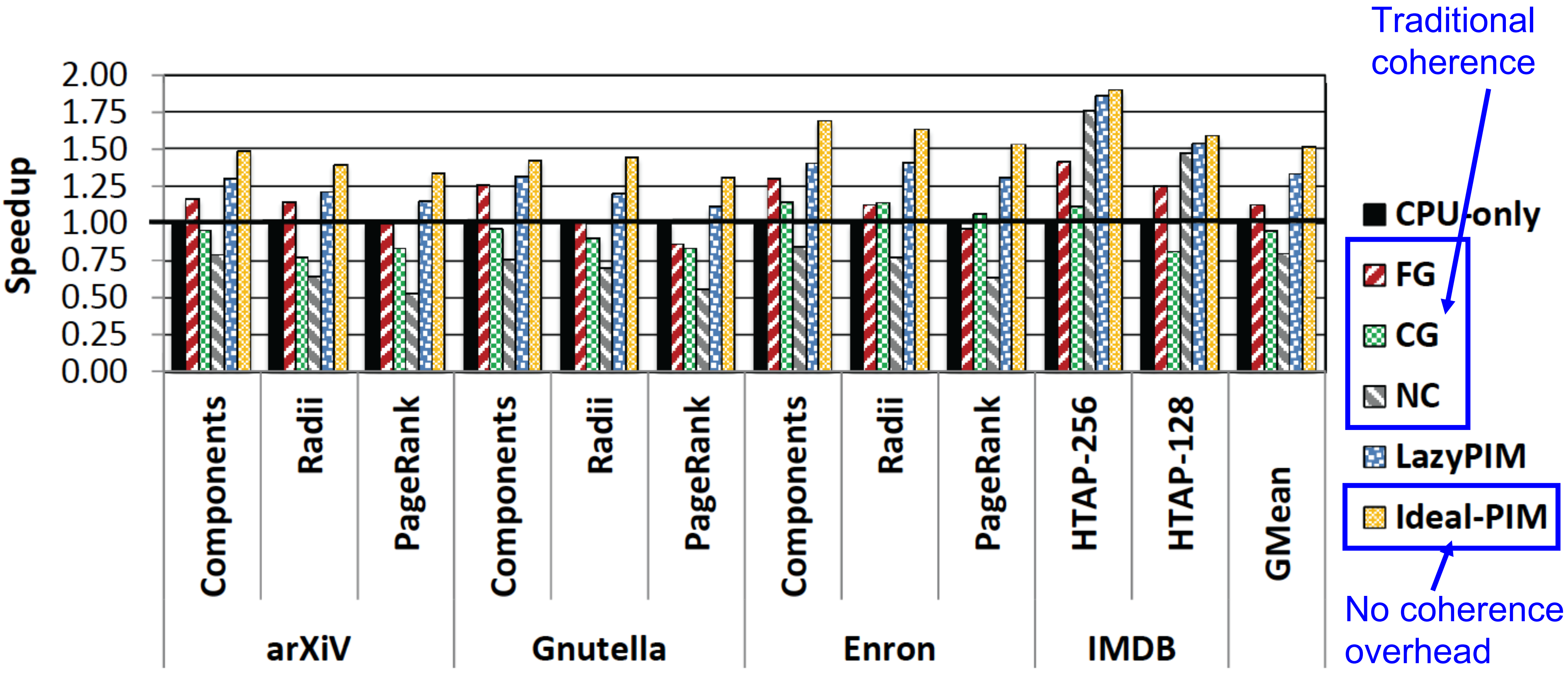
- Ashutosh Pattnaik, Xulong Tang, Adwait Jog, Onur Kayiran, Asit K. Mishra, Mahmut T. Kandemir, Onur Mutlu, and Chita R. Das, **"Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities"**
  *Proceedings of the 25th International Conference on Parallel Architectures and Compilation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

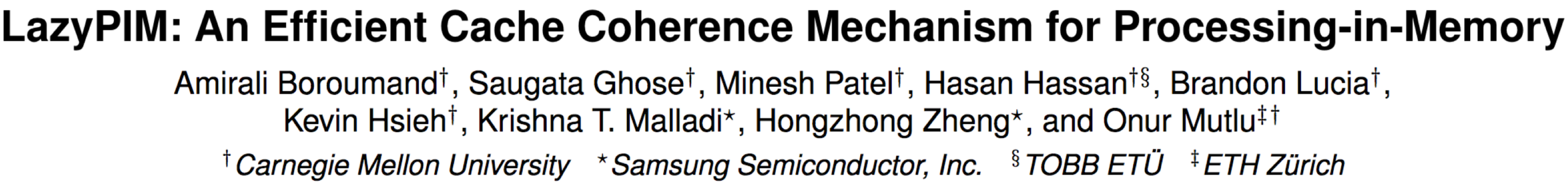


Traditional coherence

No coherence overhead

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.

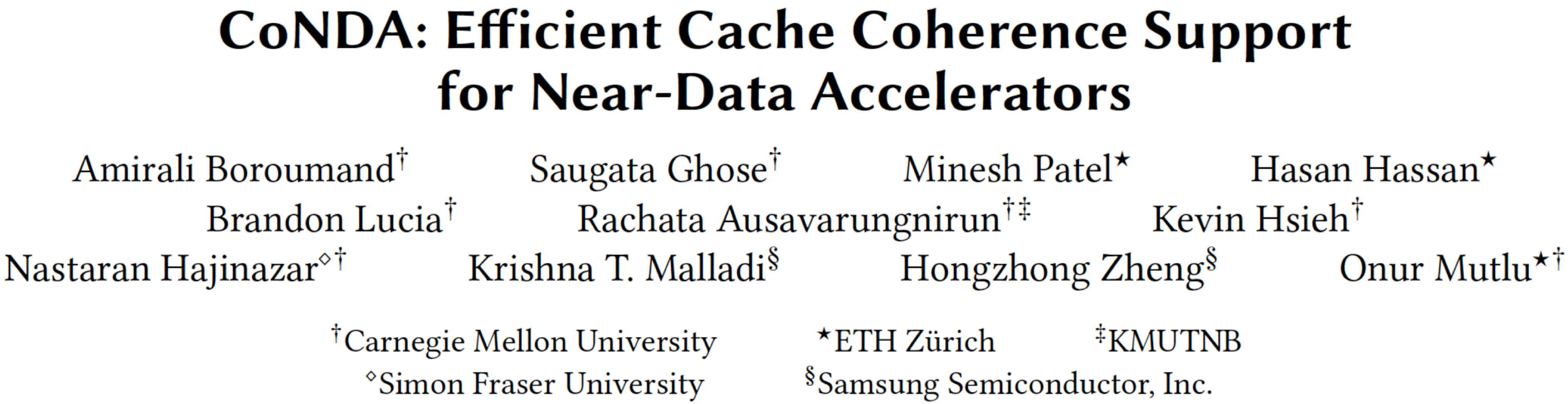


LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory

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?

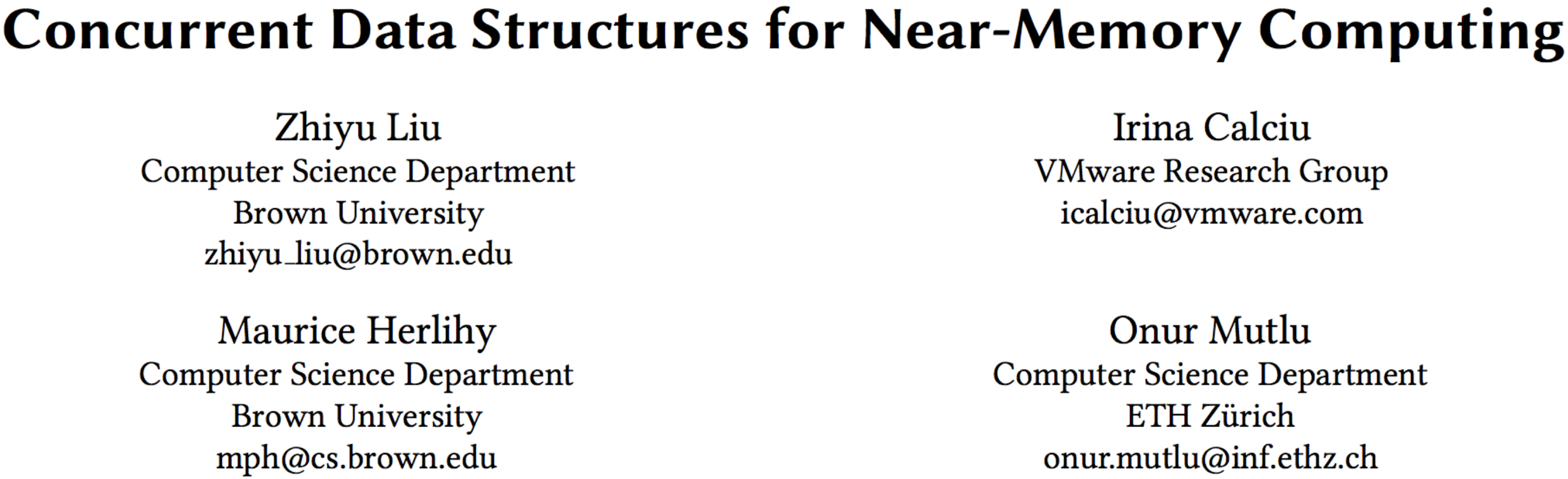
- Kevin Hsieh, Samira Khan, Nandita Vijaykumar, Kevin K. Chang, Amirali Boroumand, Saugata Ghose, and Onur Mutlu, **"Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation"** *Proceedings of the 34th IEEE International Conference on Computer Design (**ICCD**)*, Phoenix,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

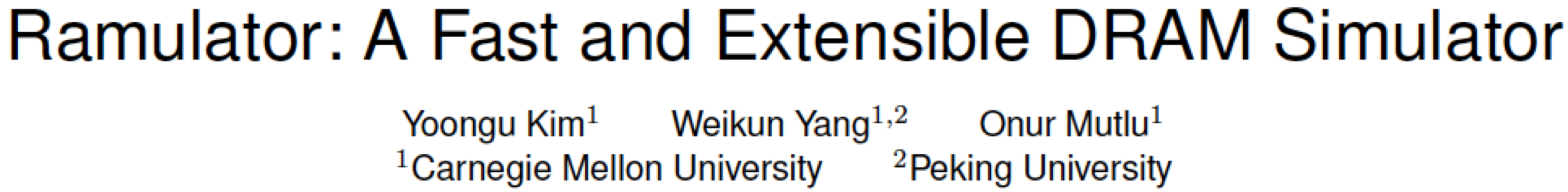
Irina Calciu
VMware Research Group
icalciu@vmware.com

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]

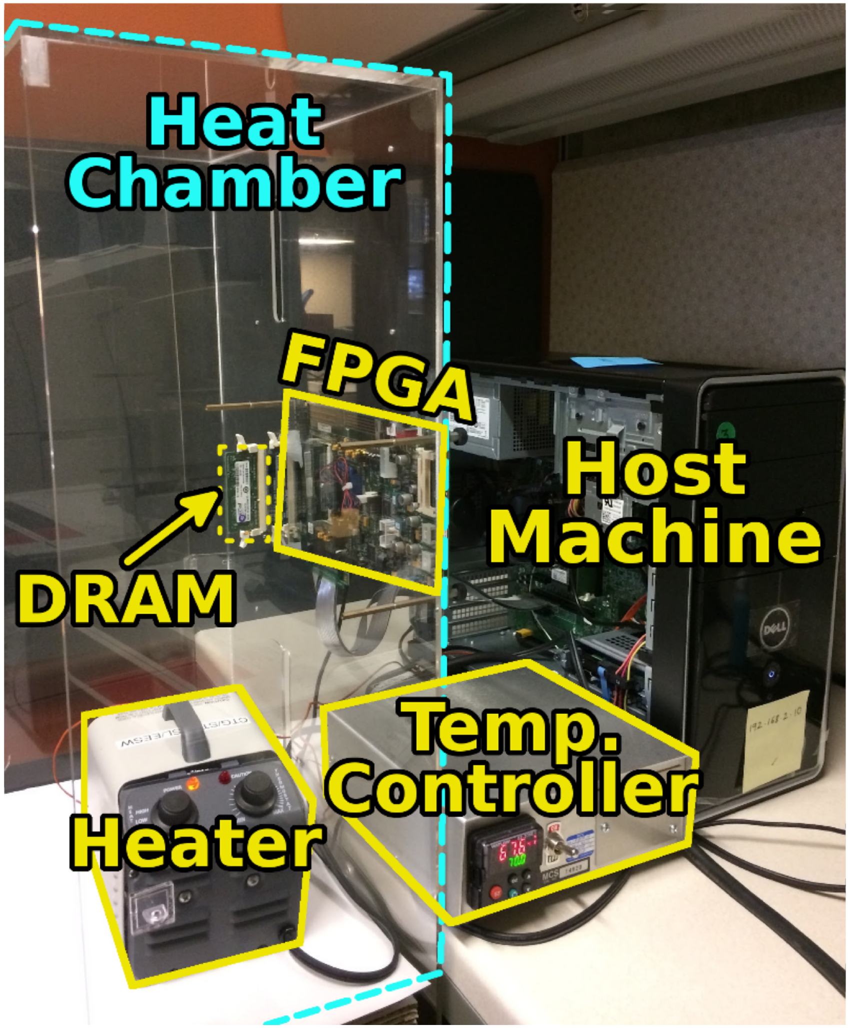
[1]Carnegie Mellon University [2]Peking University

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.

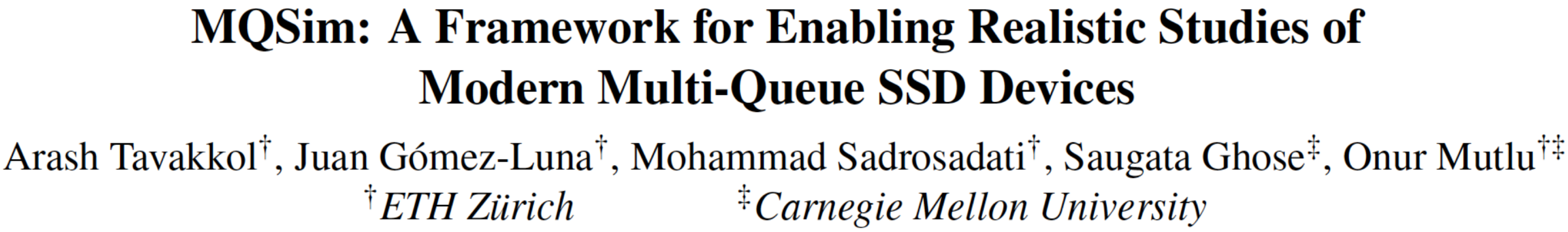
- **Flexible**
- **Easy to Use (C++ API)**
- **Open-source**

  github.com/CMU-SAFARI/SoftMC



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]

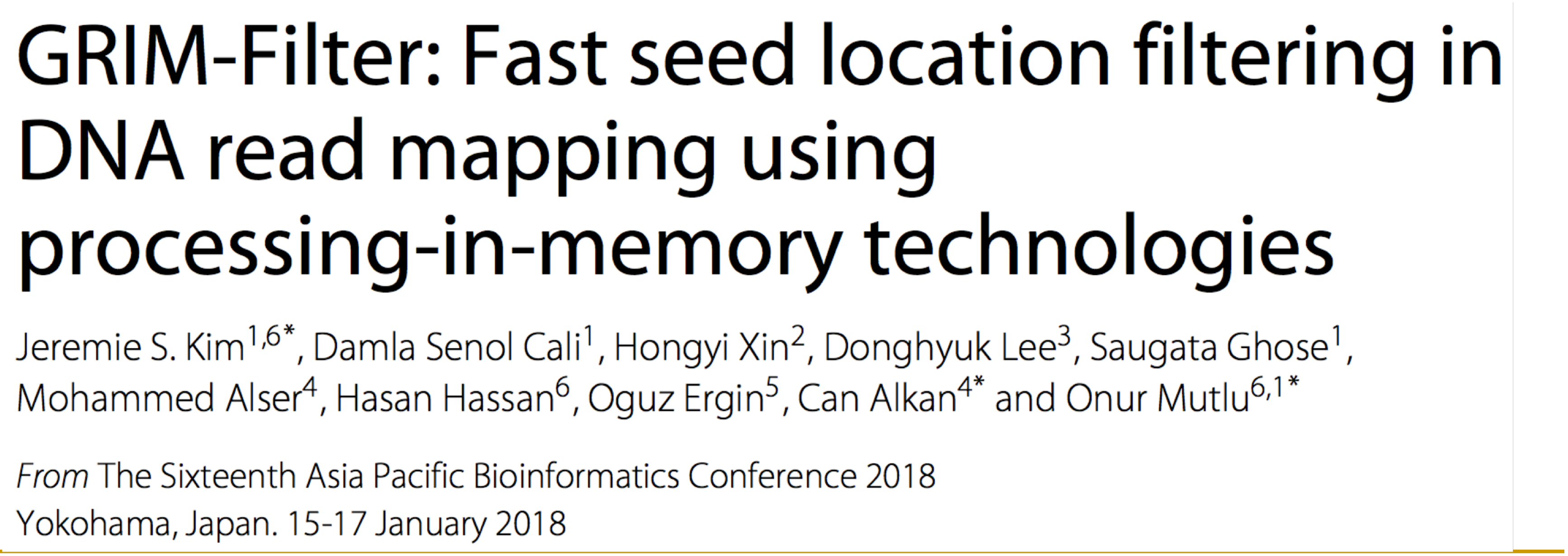


MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices

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*]

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

Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks

Amirali Boroumand

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

SAFARI Carnegie Mellon Google

SAMSUNG SEOUL NATIONAL UNIVERSITY ETH Zürich

SAFARI

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

İhsan Doğramacı Bilkent University 1984

TOBB UNIVERSITY OF ECONOMICS AND TECHNOLOGY

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**.

GRIM-Filter in 3D-stacked DRAM

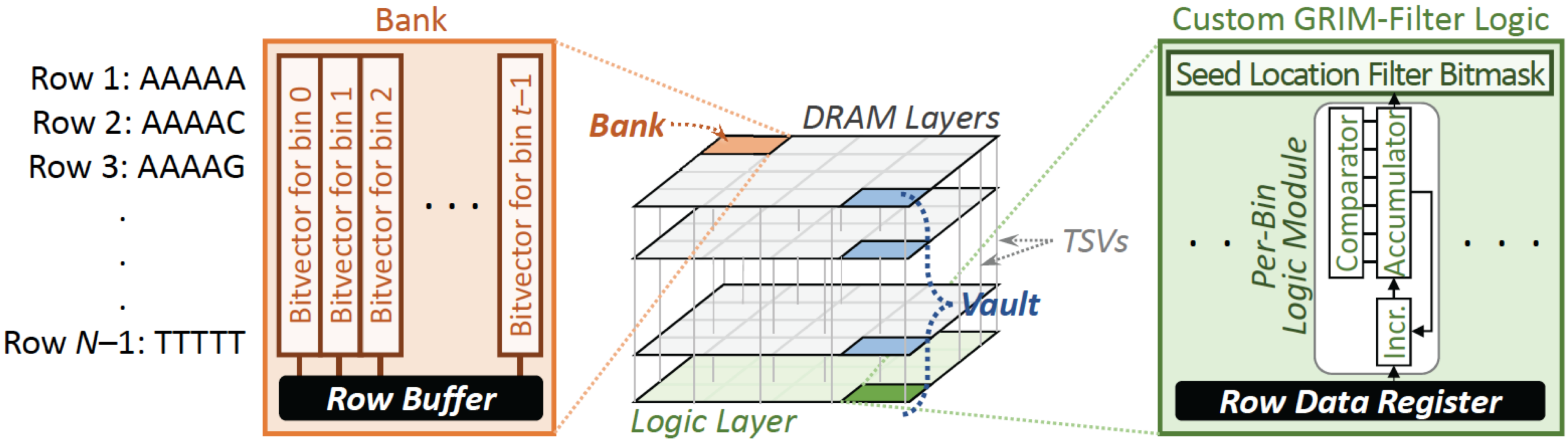
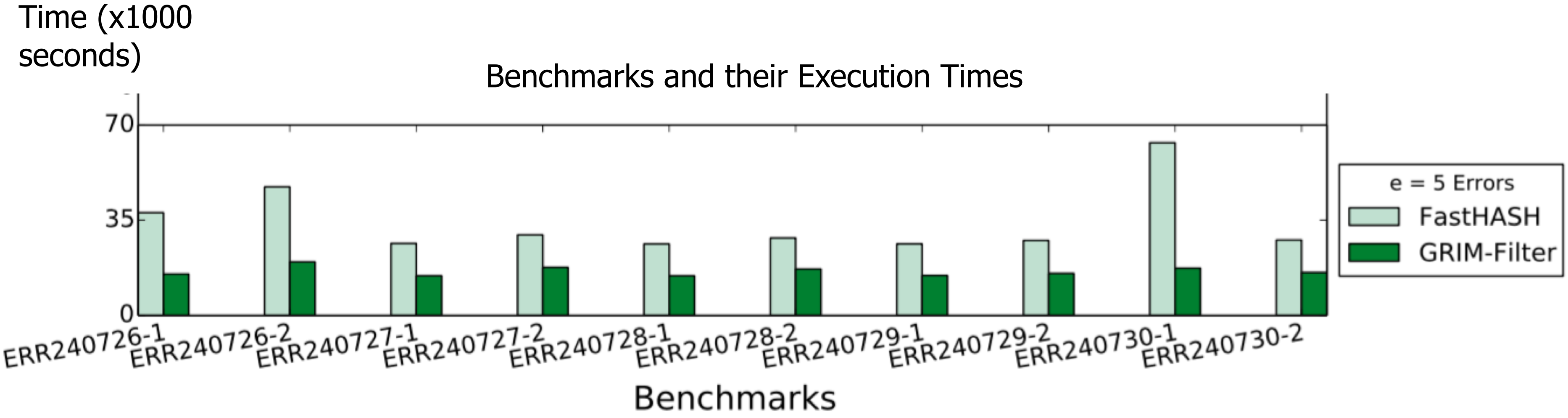


Figure 7: *Left block:* GRIM-Filter bitvector layout within a DRAM bank. *Center block:* 3D-stacked DRAM with tightly integrated logic layer stacked underneath with TSVs for a high intra-DRAM data transfer bandwidth. *Right block:* Custom GRIM-Filter logic placed in the logic layer.

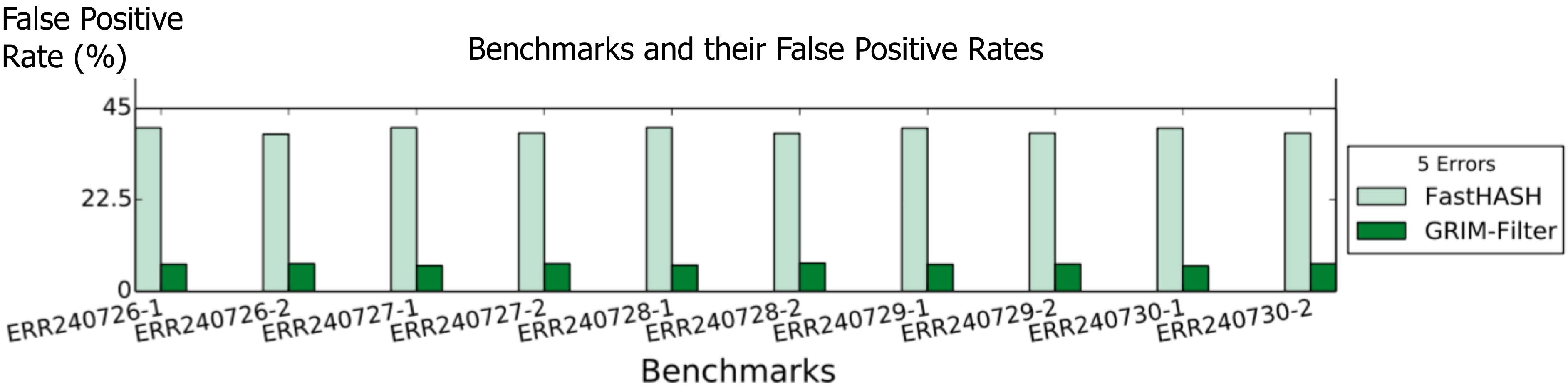
- The layout of bit vectors in a bank enables filtering many bins in parallel
- Customized logic for accumulation and comparison per genome segment
  - Low area overhead, simple implementation

GRIM-Filter Performance



1.8x-3.7x performance benefit across real data sets

GRIM-Filter False Positive Rate



5.6x-6.4x False Positive reduction across real data sets

Conclusions

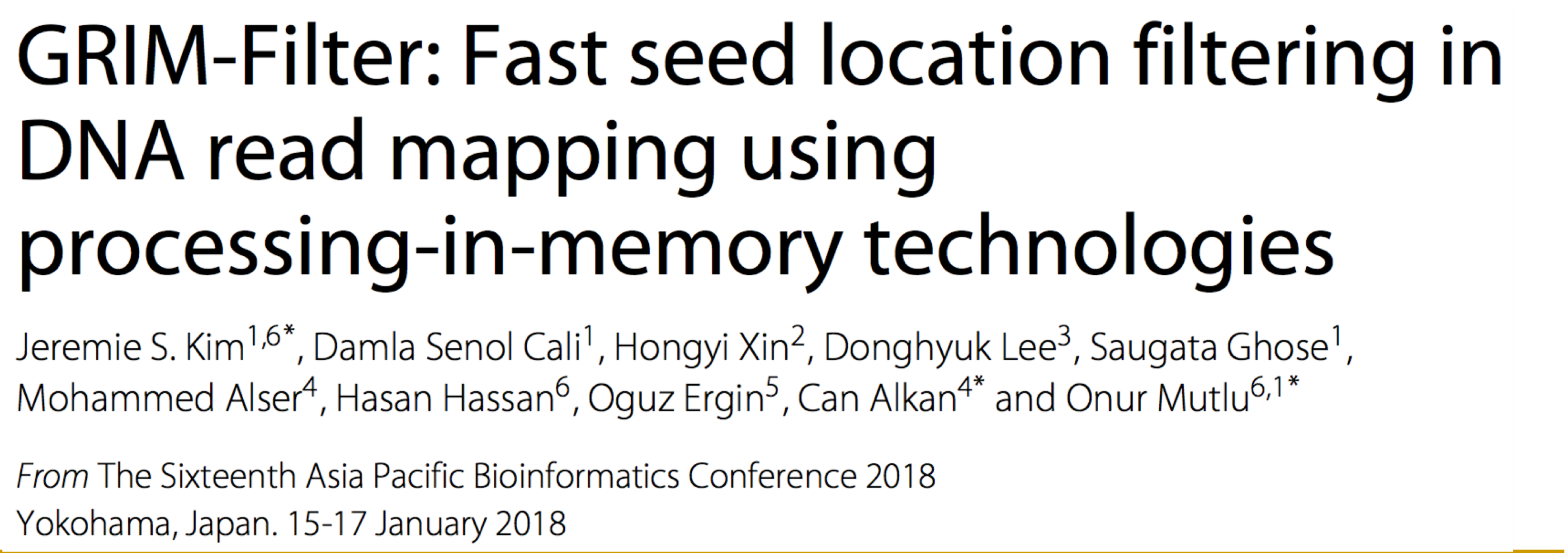
- We propose an in memory filter algorithm to accelerate end-to-end genome read mapping by reducing the number of required alignments

- Compared to the previous best filter
  - We observed 1.8x-3.7x speedup
  - We observed 5.6x-6.4x fewer false positives

- GRIM-Filter is a universal filter that can be applied to any genome read mapper

In-Memory DNA Sequence Analysis

- 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)

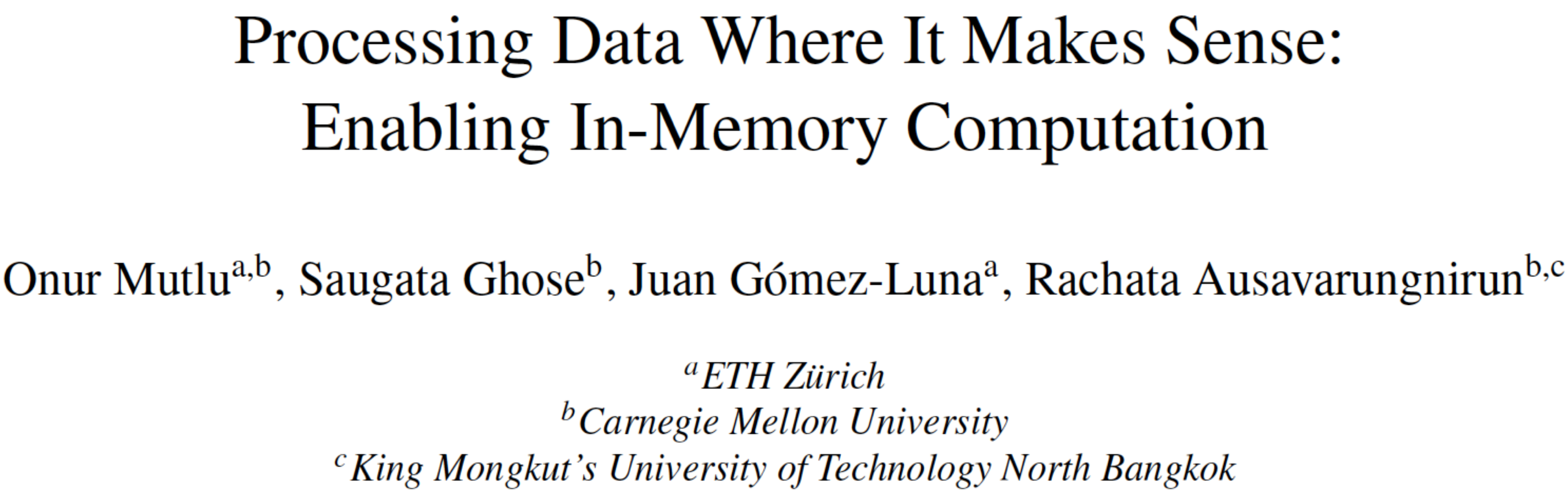


GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies

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*]

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

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

Enabling the Paradigm Shift

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

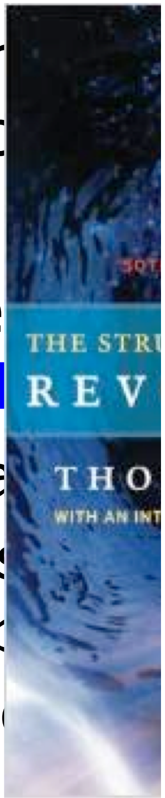
- Recommended book: Thomas Kuhn, "The Structure of Scientific Revolutions" (1962)
  - Pre-paradigm science: no clear consensus in the field
  - Normal science: dominant theory used to explain/improve things (business as usual); exceptions considered anomalies
  - Revolutionary science: underlying assumptions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

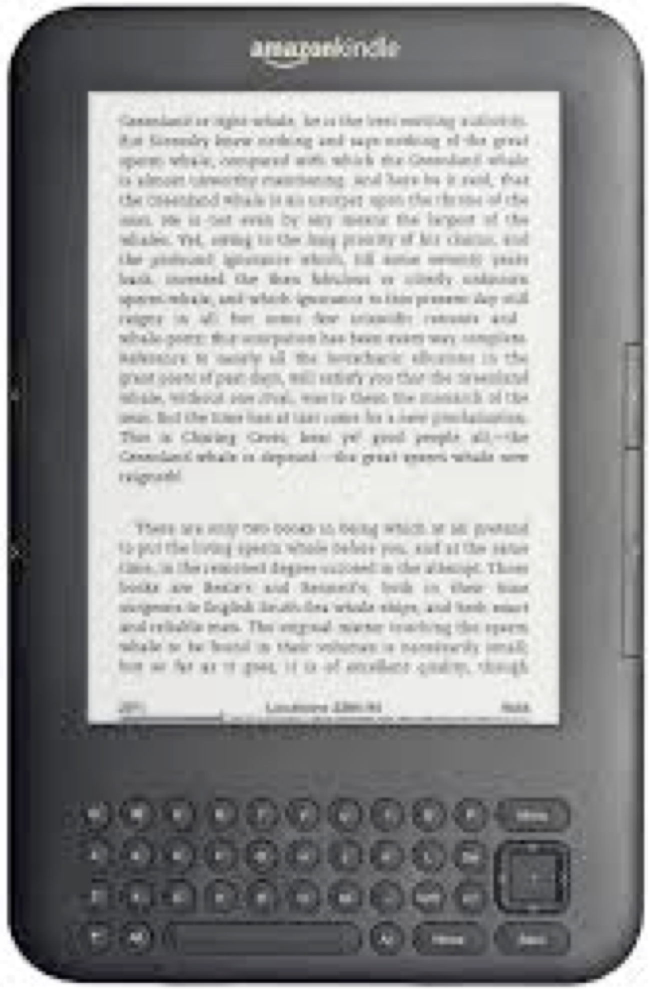
Computer Architecture Today

- You can revolutionize the way computers are built, if you understand bot oftware (and change each ac

- You can ir communic

- Recomme ure of Scientific R
  - Pre-para ield
  - Normal s improve things (b anomalies
  - Revoluti examined





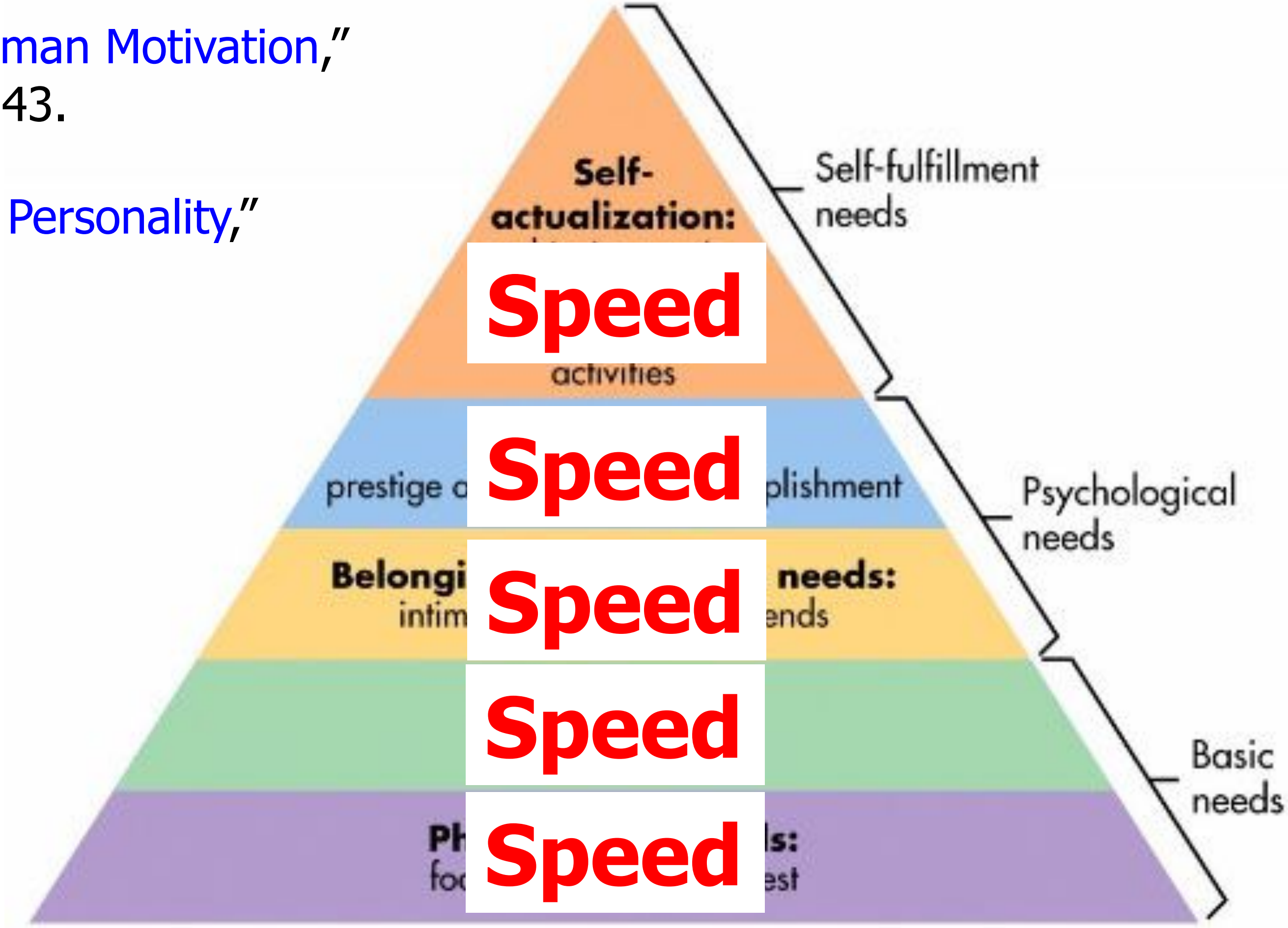
Agenda

- 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.



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

Challenge and Opportunity for Future

Fundamentally

High-Performance

(Data-Centric)

Computing Architectures

Challenge and Opportunity for Future

Fundamentally

Energy-Efficient

(Data-Centric)

Computing Architectures

Challenge and Opportunity for Future

Fundamentally

Low-Latency

(Data-Centric)

Computing Architectures

Challenge and Opportunity for Future

Computing Architectures

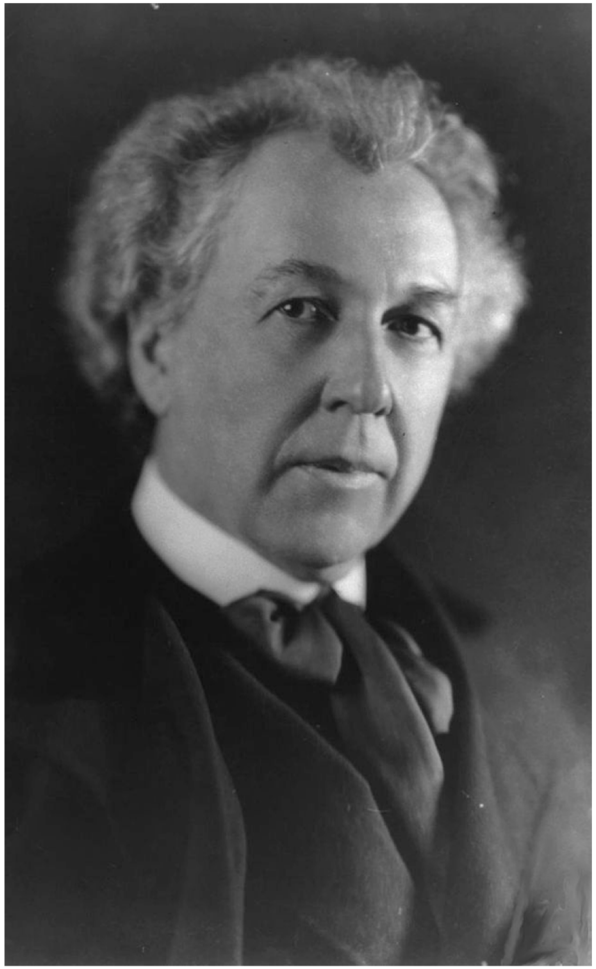
with

Minimal Data Movement

PIM: Concluding Remarks

A Quote from A Famous Architect

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



Precedent-Based Design?

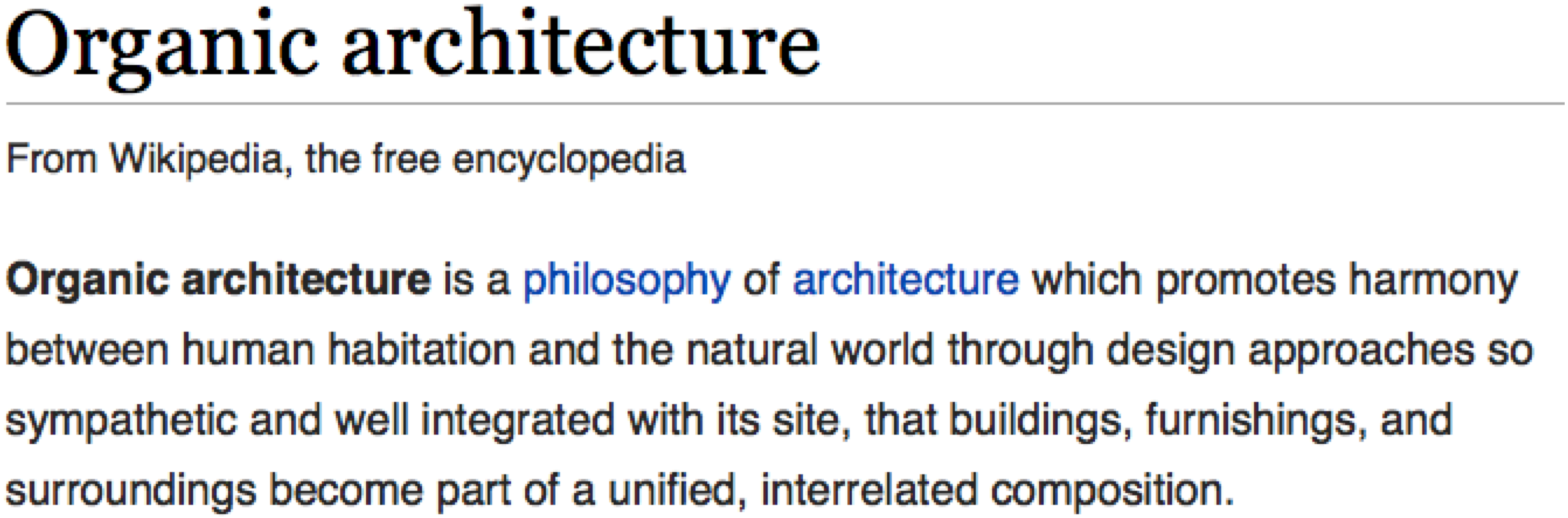
- “architecture […] based upon principle, and not upon precedent”

Principled Design

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

www.GreatBuildings.com

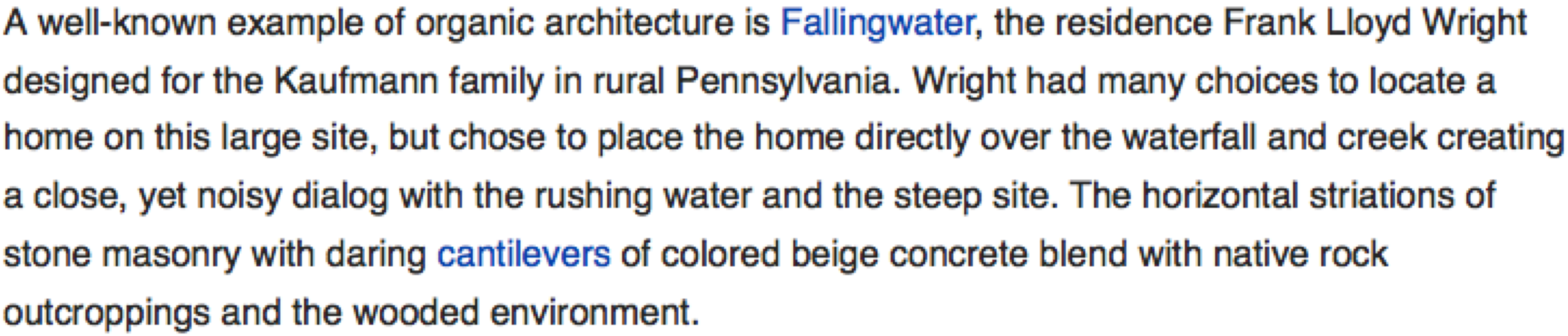
The Overarching Principle



Organic architecture

From Wikipedia, the free encyclopedia

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



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

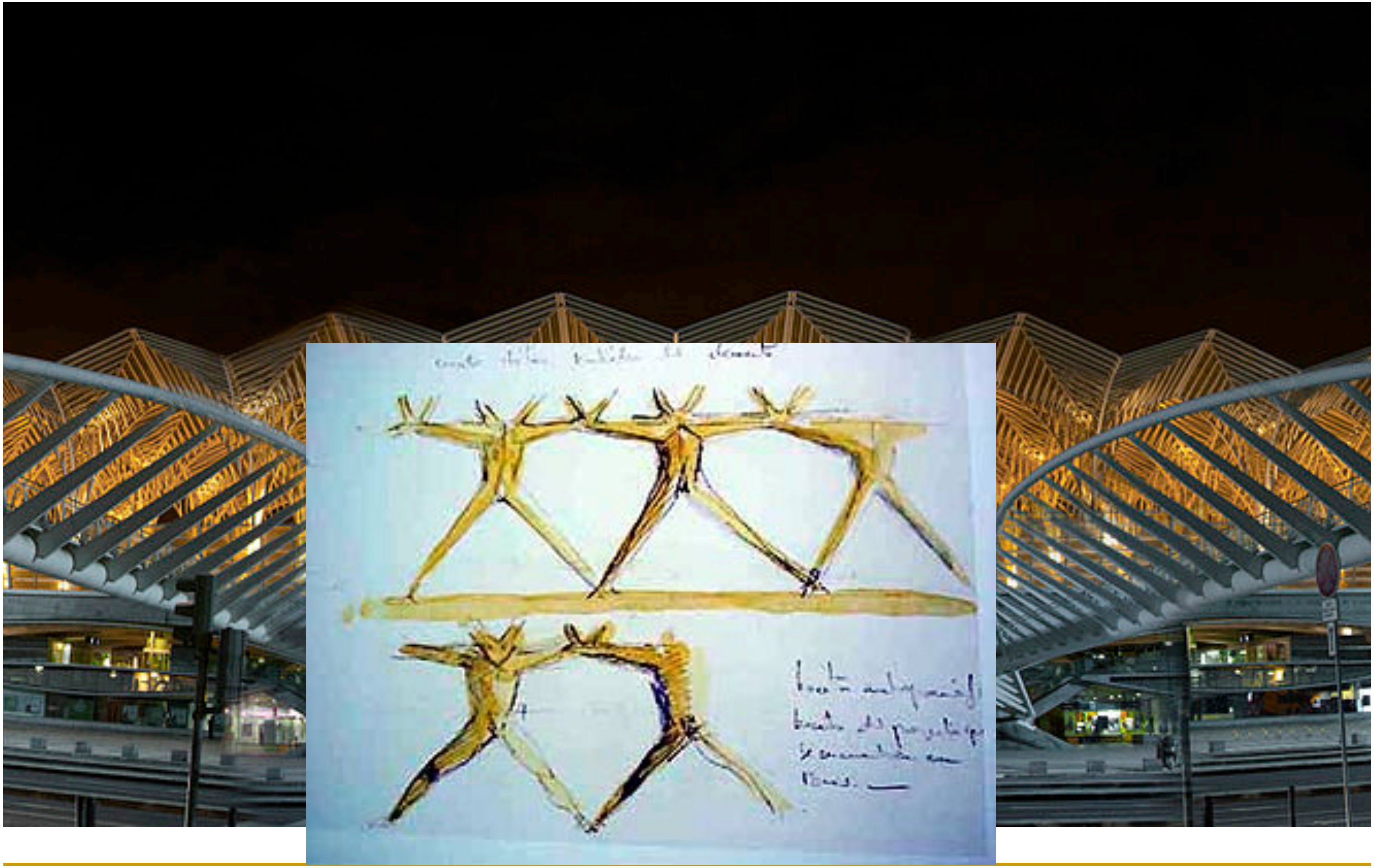
Another Example: Precedent-Based Design

Source: http://cookiemagik.deviantart.com/art/Train-station-207266944

Principled Design

Source: By Toni_V, CC BY-SA 2.0, https://commons.wikimedia.org/w/index.php?curid=4087256

Another Principled Design



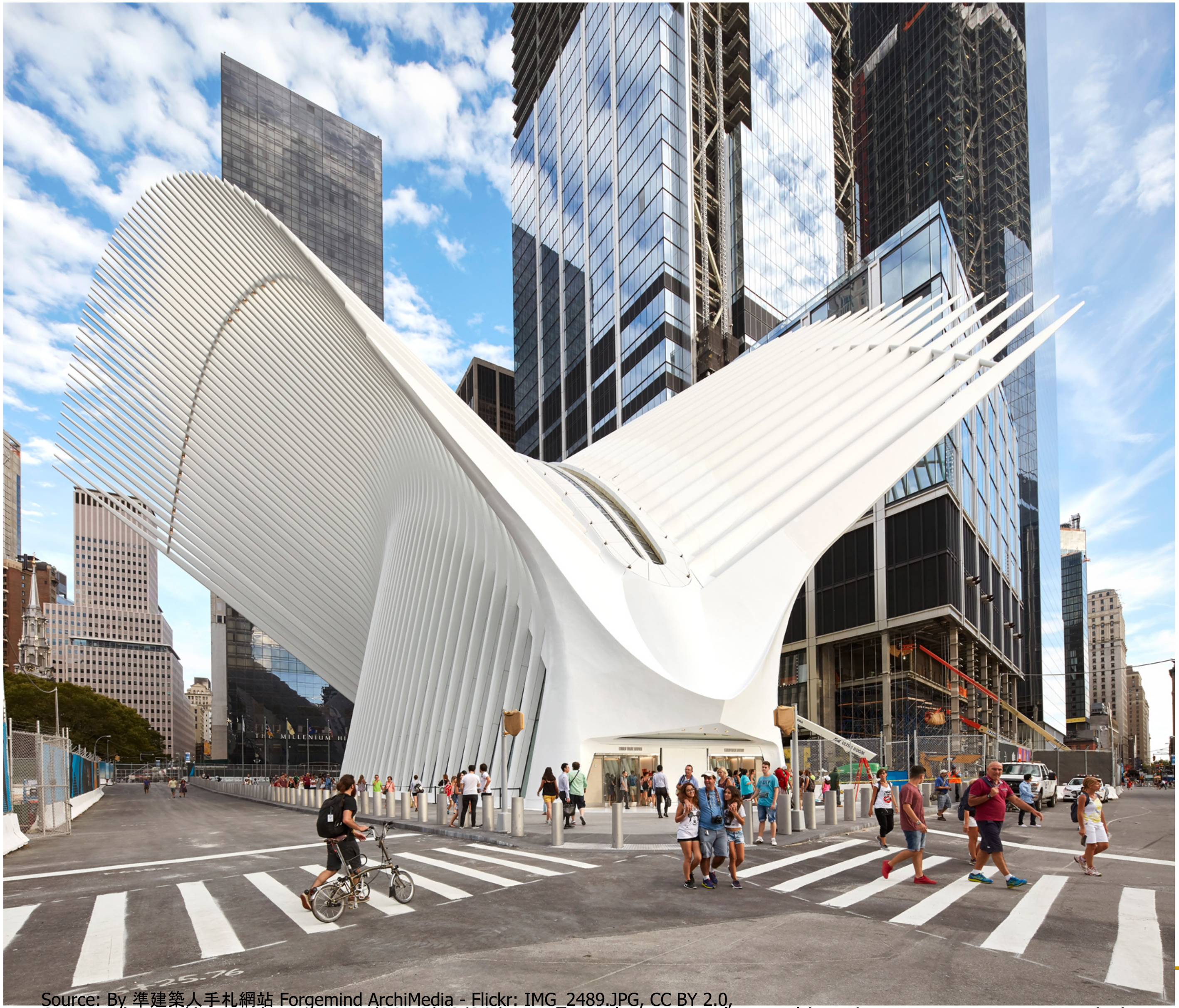
Source: By Martín Gómez Tagle - Lisbon, Portugal, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=13764903

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

Another Principled Design

Source: CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=172107

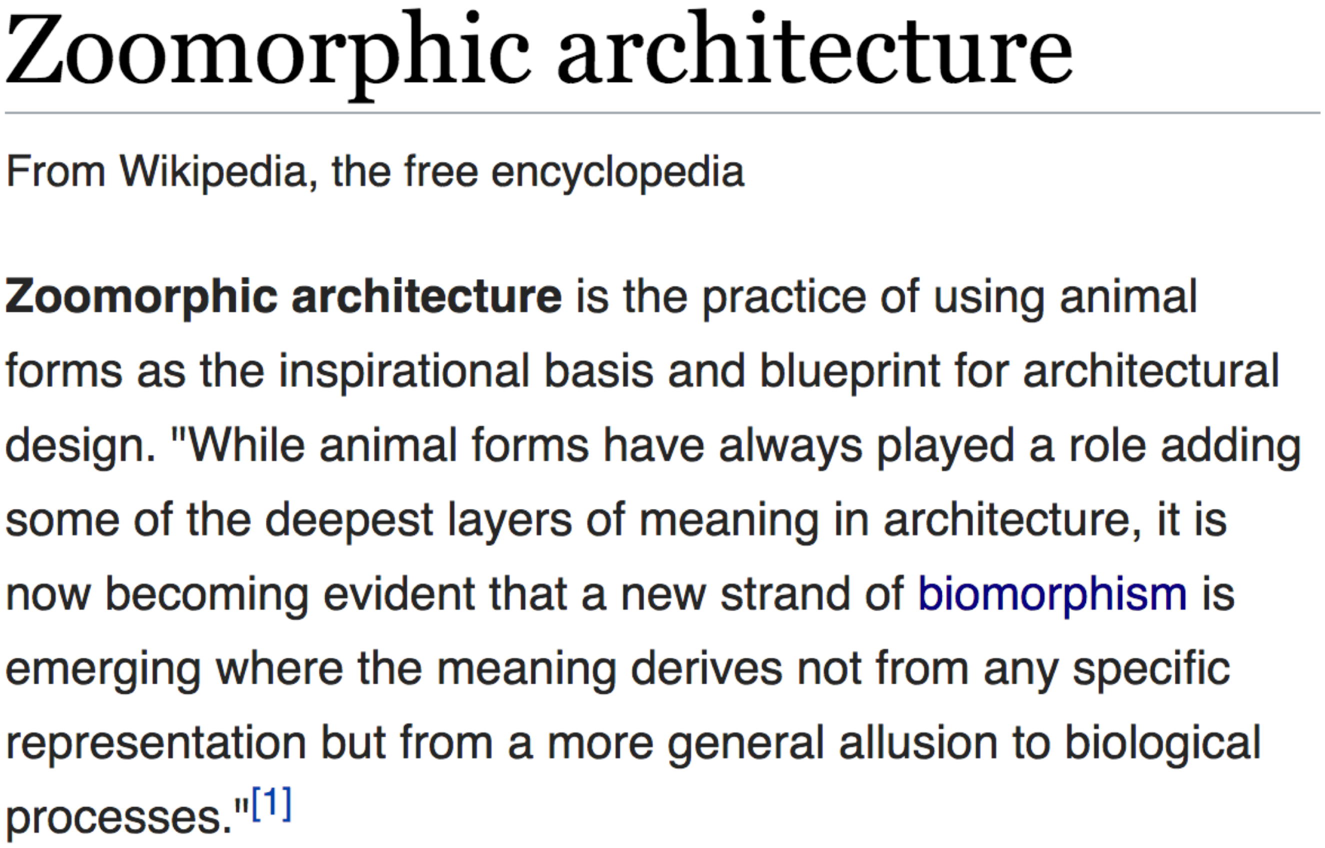
Principle Applied to Another Structure



Source: By 準建築人手札網站 Forgemind ArchiMedia - Flickr: IMG_2489.JPG, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=51493356, https://en.wikipedia.org/wiki/Santiago_Calatrava

Source: https://www.dezeen.com/2016/08/29/santiago-calatrava-oculus-world-trade-center-transportation-hub-new-york-photographs-hufton-crow/

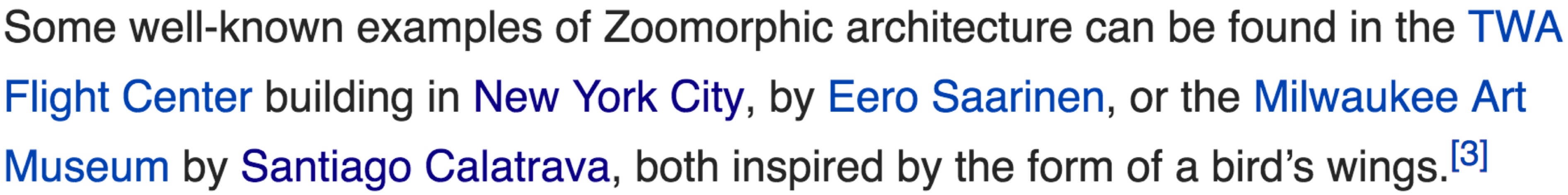
The Overarching Principle



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**
  - **…**

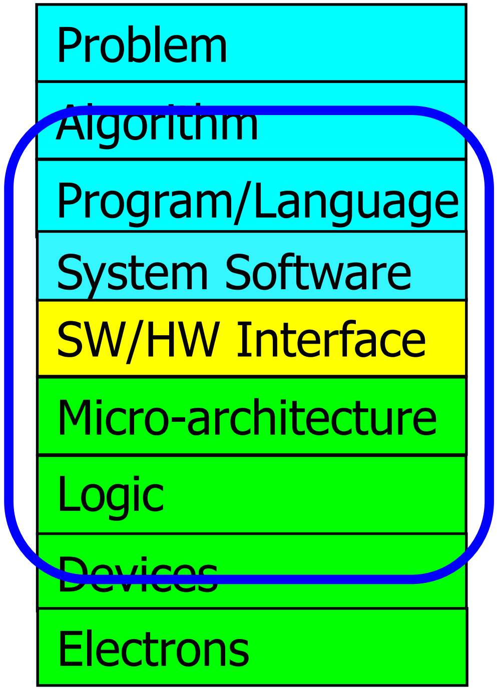
The Future of Processing in Memory is Bright

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

Can enable:

- Orders of magnitude improvements

- New applications and computing systems



Problem

Algorithm

Program/Language

System Software

SW/HW Interface

Micro-architecture

Logic

Devices

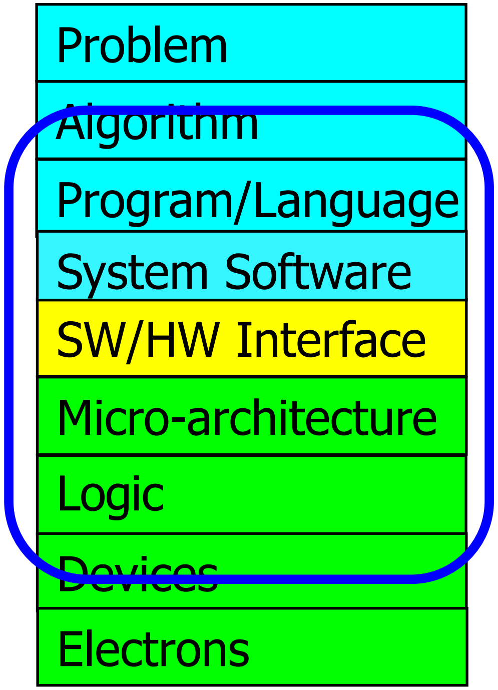
Electrons

Yet, we have to

- Think across the stack

- Design enabling systems

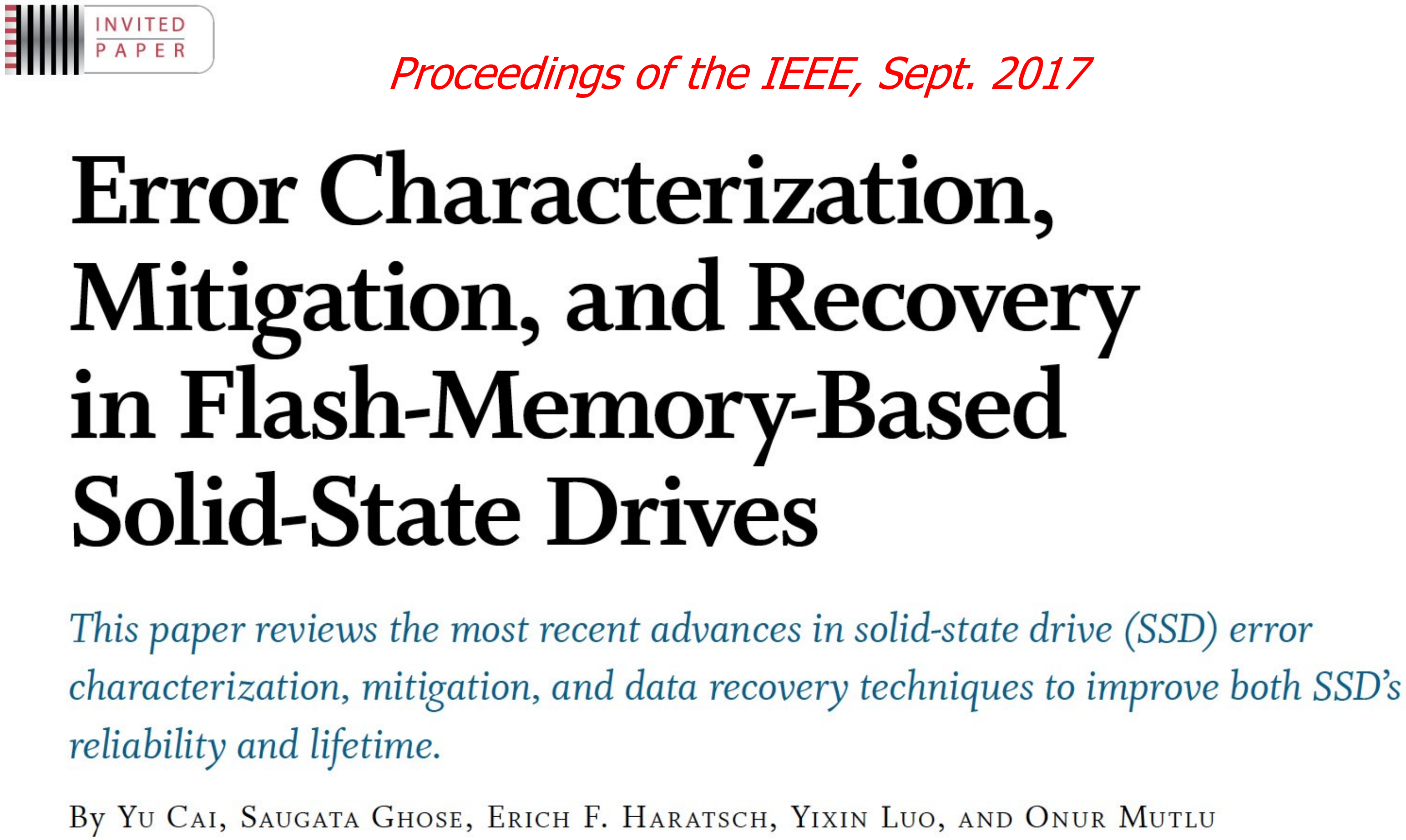
We Need to Revisit the Entire Stack



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

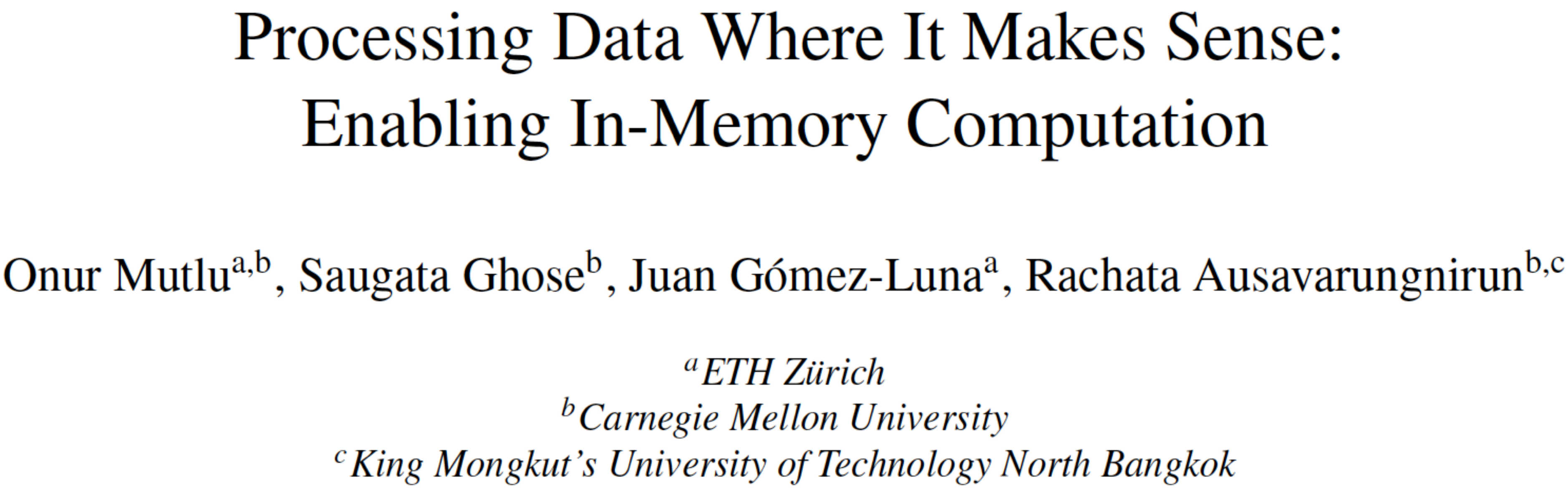
Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

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

Lecture 4a: Processing-in-Memory II

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu

17 June 2019

TU Wien Fast Course 2019

SAFARI ETH zürich Carnegie Mellon

Backup Slides

GRIM-Filter:

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

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

Systems@ETH zürich

SAFARI

Carnegie Mellon

İhsan Doğramacı Bilkent University

TOBB
UNIVERSITY OF
ECONOMICS AND TECHNOLOGY

ETH zürich

Executive Summary

- **Genome Read Mapping** is a very important problem and is the first step in genome analysis

- 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 algorithm called **GRIM-Filter**
  - Accelerates read mapping by reducing the number of required alignments
  - GRIM-Filter can be accelerated using **processing-in-memory**
    - Adds simple logic into **3D-Stacked memory**
    - Uses high internal memory bandwidth to perform parallel filtering

- GRIM-Filter with processing-in-memory delivers a **3.7x speedup**

GRIM-Filter Outline

1. **Motivation and Goal**

2. Background Read Mappers

   a. Hash Table Based

   b. Hash Table Based with Filter

3. Our Proposal: GRIM-Filter

4. Mapping GRIM-Filter to 3D-Stacked Memory

5. Results

6. Conclusion

Motivation and Goal

- **Sequencing**: determine the [A,C,G,T] series in DNA strand

- Today’s machines sequence short strands (**reads**)
  - Reads are on the order of 100 – 20k base pairs (**bp**)
  - The human genome is approximately 3 billion bp

- Therefore genomes are cut into reads, which are sequenced independently, and then reconstructed
  - **Read mapping** is the first step in analyzing someone’s genome to detect predispositions to diseases, personalize medicine, etc.

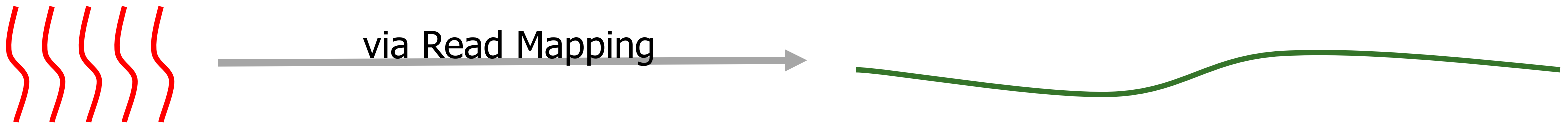
- **Goal**: We want to **accelerate** end-to-end performance of **read mapping**

GRIM-Filter Outline

1. Motivation and Goal
2. **Background: Read Mappers**
   a. Hash Table Based
   b. Hash Table Based with Filter
3. Our Proposal: GRIM-Filter
4. Mapping GRIM-Filter to 3D-Stacked Memory
5. Results
6. Conclusion

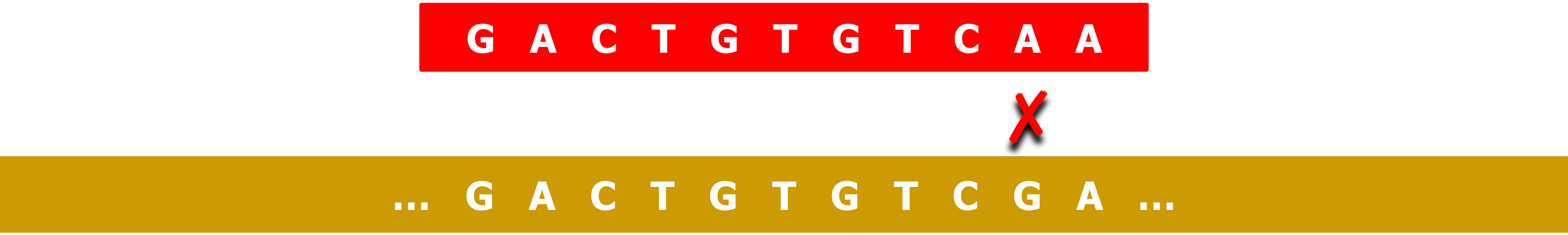
Background: Read Mappers

We now have **sequenced reads** and want a **full genome**



We map **reads** to a known **reference genome** (>99.9% similarity across humans) with some minor errors allowed

Because of high similarity, long sequences in **reads** perfectly match in the **reference genome**



G A C T G T G T C A A

… G A C T G T G T C G A …

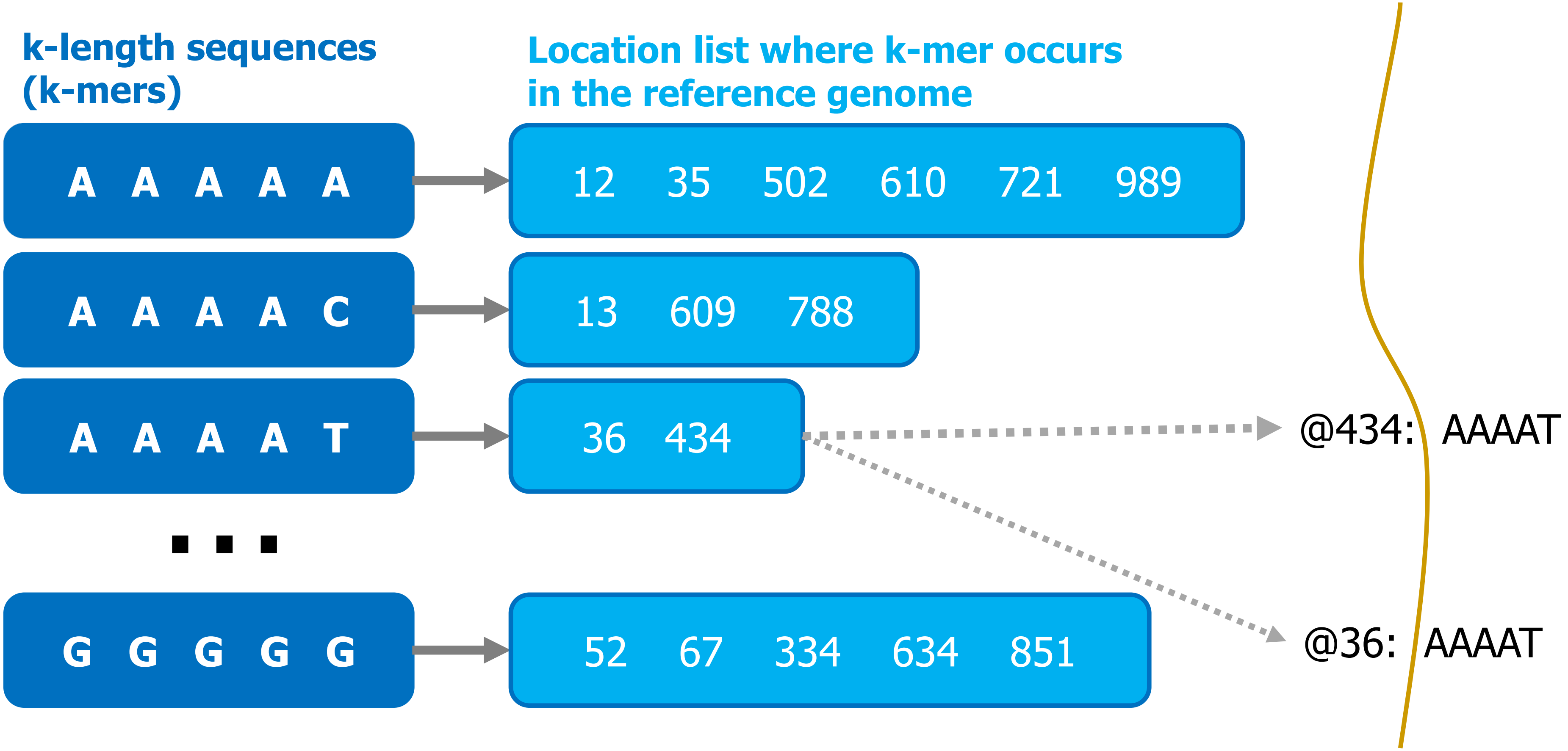
We can use a hash table to help quickly map the reads!

GRIM-Filter Outline

1. Motivation and Goal
2. **Background: Read Mappers**
   a. **Hash Table Based**
   b. Hash Table Based with Filter
3. Our Proposal: GRIM-Filter
4. Mapping GRIM-Filter to 3D-Stacked Memory
5. Results
6. Conclusion

Generating Hash Tables

To map any reads, generate a **hash table** per **reference genome.**



We can query the table with substrings from reads to quickly find a list of possible mapping locations

Hash Tables in Read Mapping

Read Sequence (100 bp)

99.9% of locations result in a mismatch

Hash Table

Reference Genome

We want to filter these out so we do not waste time trying to align them

Location Filtering

- **Alignment** is expensive and requires the use of O(n^2) dynamic programming algorithm
  - We need to align millions to billions of reads

Our goal is to accelerate read mapping by improving the filtering step

out mismatches quickly)

- Both methods are used by mappers today, but filtering has replaced alignment as the bottleneck **[Xin+, BMC Genomics 2013]**

GRIM-Filter Outline

1. Motivation and Goal
2. **Background: Read Mappers**
   - a. Hash Table Based
   - **b. Hash Table Based with Filter**
3. Our Proposal: GRIM-Filter
4. Mapping GRIM-Filter to 3D-Stacked Memory
5. Results
6. Conclusion

GRIM-Filter Outline

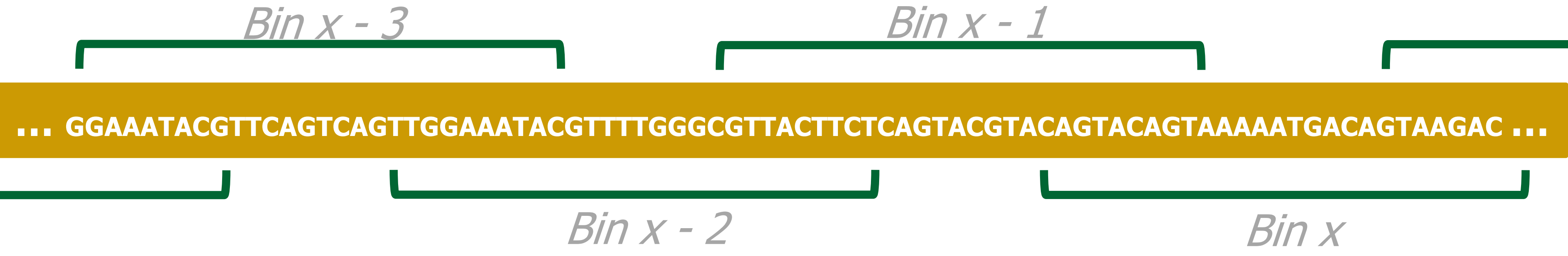
1. Motivation and Goal
2. Background: Read Mappers
   - a. Hash Table Based
   - b. Hash Table Based with Filter
3. **Our Proposal: GRIM-Filter**
4. Mapping GRIM-Filter to 3D-Stacked Memory
5. Results
6. Conclusion

Our Proposal: GRIM-Filter

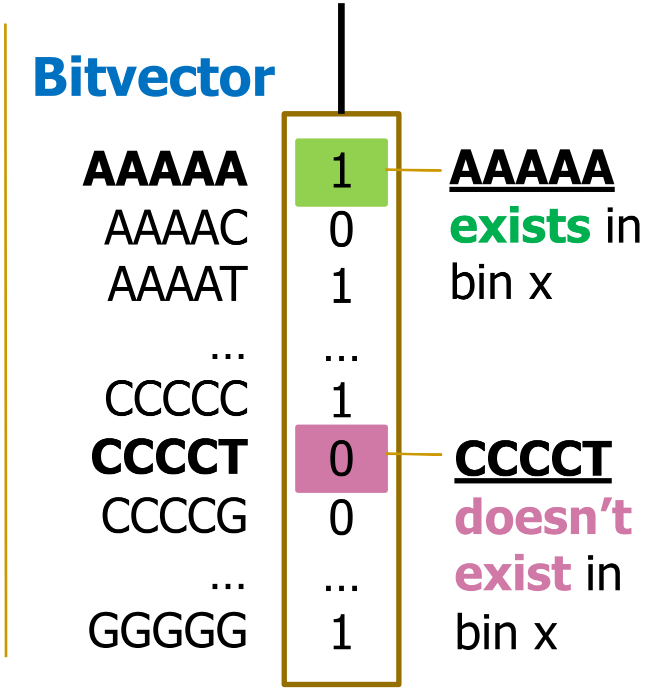
1. **Data Structures: Bins & Bitvectors**
2. Checking a Bin
3. Integrating GRIM-Filter into a Mapper

GRIM-Filter: Bins

- We partition the genome into large sequences (**bins**).


Bin x - 3
Bin x - 1
… GGAAATACGTTCAGTCAGTTGGAAATACGTTTTGGGCGTTACTTCTCAGTACGTACAGTACAGTAAAAATGACAGTAAGAC …
Bin x - 2
Bin x

- Represent each bin with a **bitvector** that holds the occurrence of all permutations of a small string (**token**) in the bin
- To account for matches that straddle bins, we employ overlapping bins
  - A read will now always completely fall within a single bin



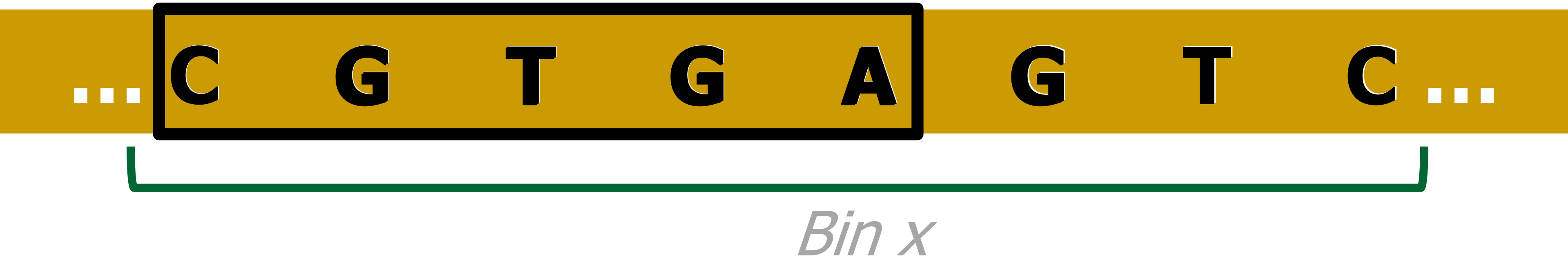
Bitvector

AAAAA	1
AAAAC	0
AAAAT	1
…	…
CCCCC	1
CCCCT	0
CCCCG	0
…	…
GGGGG	1

AAAAA **exists** in bin x

CCCCT **doesn't exist** in bin x

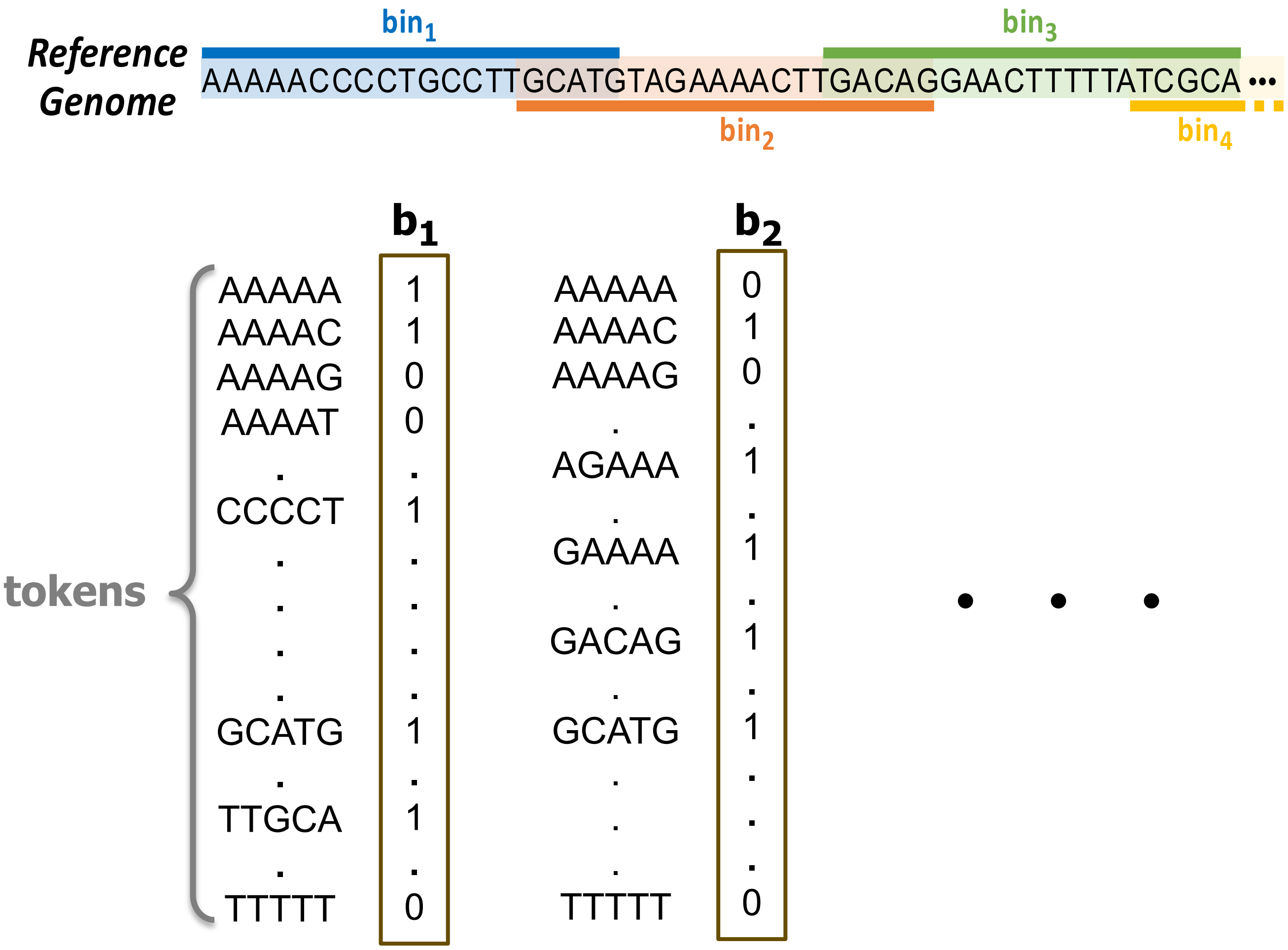
GRIM-Filter: Bitvectors



Bin x Bitvector

AAAAA	0
...	...
CGTGA	0
...	...
TGAGT	0
...	...
GAGTC	0
...	...
GTGAG	0
...	...

GRIM-Filter: Bitvectors



tokens	b_1
AAAAA	1
AAAAC	1
AAAAG	0
AAAAT	0
.	.
CCCCT	1
.	.
.	.
.	.
.	.
GCATG	1
.	.
TTGCA	1
.	.
TTTTT	0

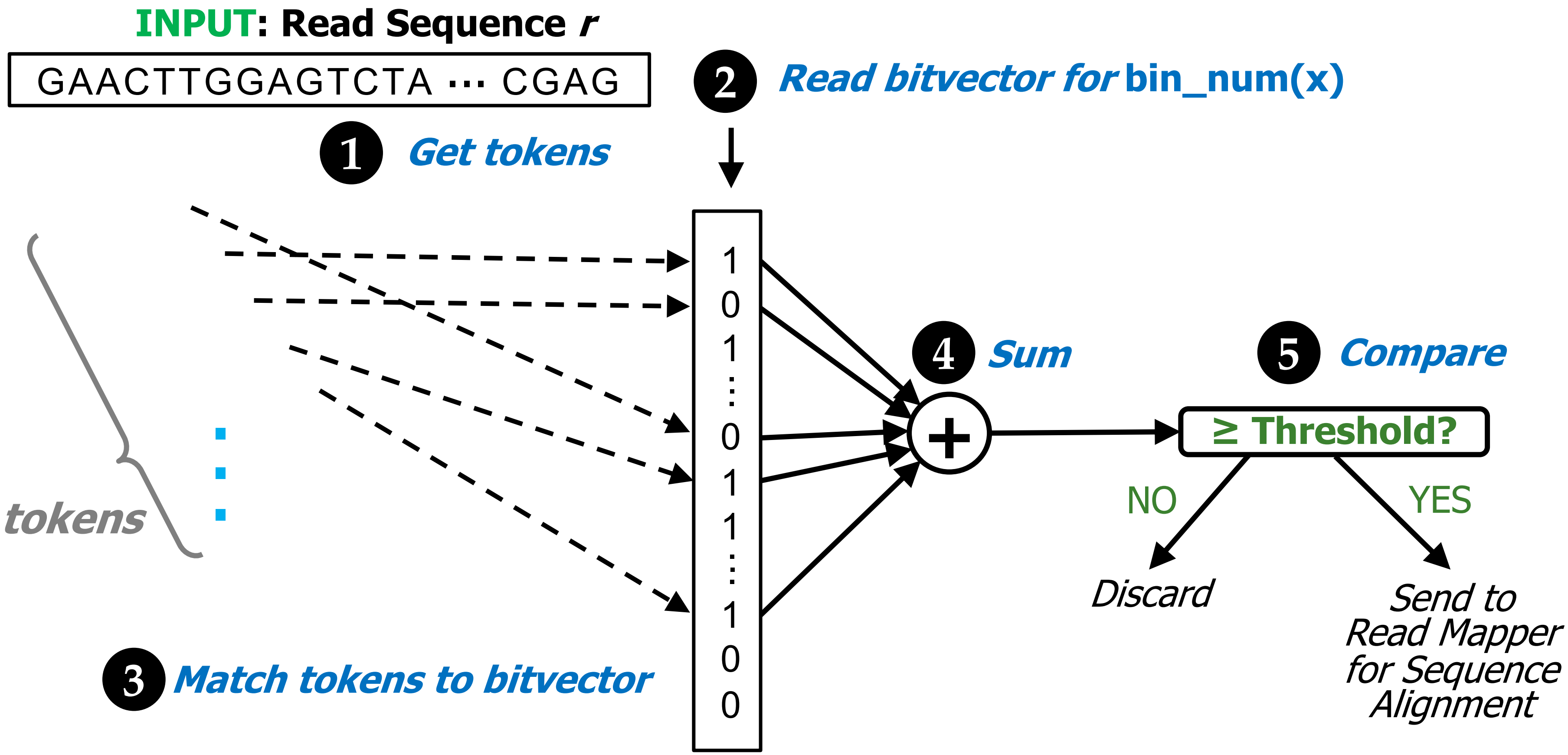
tokens	b_2
AAAAA	0
AAAAC	1
AAAAG	0
.	.
AGAAA	1
.	.
GAAAA	1
.	.
GACAG	1
.	.
GCATG	1
.	.
.	.
.	.
TTTTT	0

Storing all bitvectors requires $4^n * t$ bits in memory, where t = number of bins.

For **bin size** ~200, and **n** = 5, **memory footprint** ~3.8 GB

GRIM-Filter: Checking a Bin

How GRIM-Filter determines whether to **discard** potential match locations in a given bin **prior** to alignment



Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors

2. Checking a Bin

3. Integrating GRIM-Filter into a Mapper

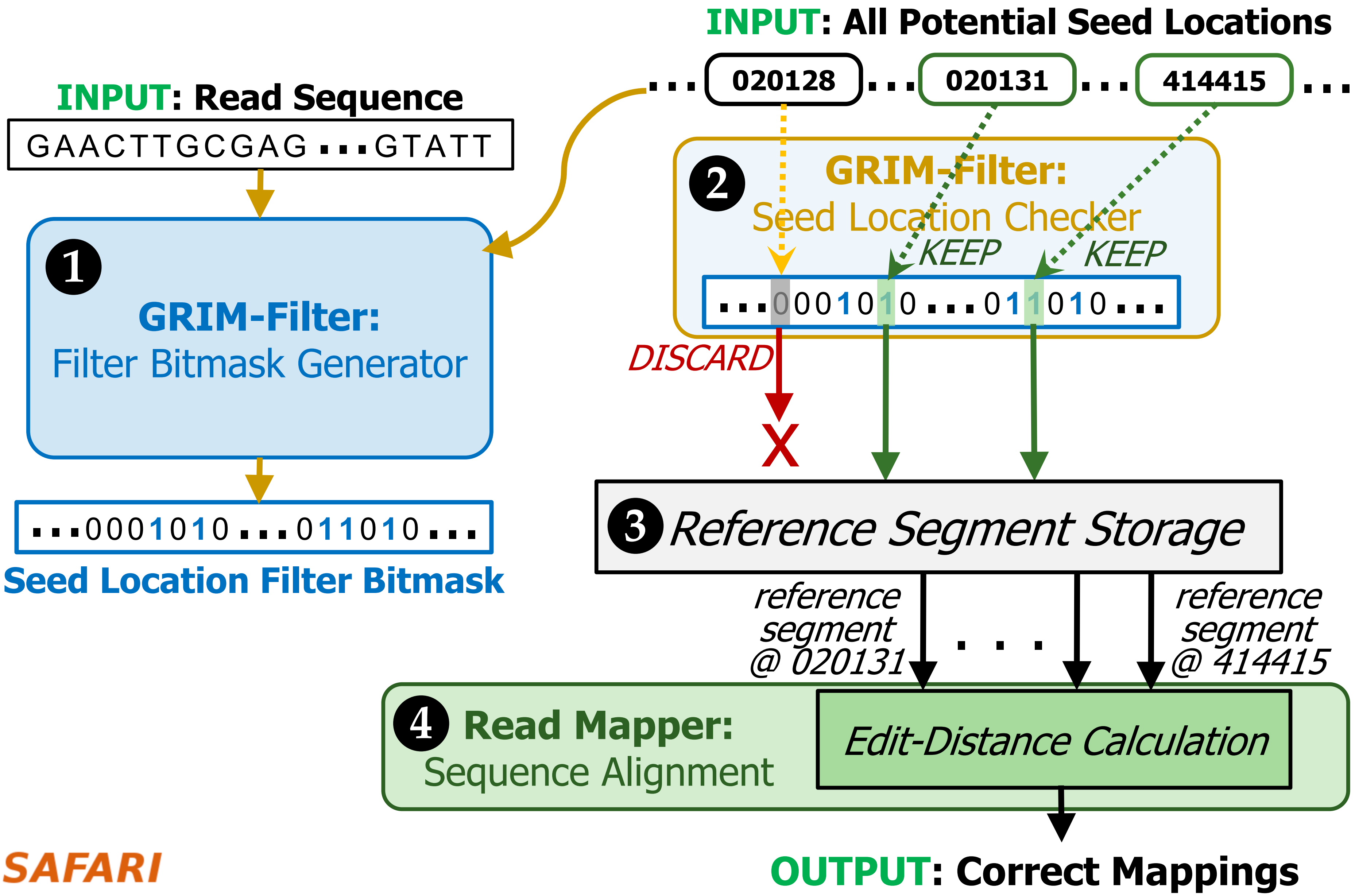
Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors
2. **Checking a Bin**
3. Integrating GRIM-Filter into a Mapper

Our Proposal: GRIM-Filter

1. Data Structures: Bins & Bitvectors
2. Checking a Bin
3. **Integrating GRIM-Filter into a Mapper**

Integrating GRIM-Filter into a Read Mapper



INPUT: Read Sequence

GAACTTGCGAG…GTATT

❶ **GRIM-Filter:**
Filter Bitmask Generator

…0001010…011010…

Seed Location Filter Bitmask

INPUT: All Potential Seed Locations

… 020128 … 020131 … 414415 …

❷ **GRIM-Filter:**
Seed Location Checker

KEEP *KEEP*

…0001010…011010…

DISCARD

X

❸ *Reference Segment Storage*

reference segment @ 020131

. . .

reference segment @ 414415

❹ **Read Mapper:**
Sequence Alignment

Edit-Distance Calculation

OUTPUT: Correct Mappings

GRIM-Filter Outline

1. Motivation and Goal
2. Background: Read Mappers
    - a. Hash Table Based
    - b. Hash Table Based with Filter
3. Our Proposal: GRIM-Filter
4. Mapping GRIM-Filter to 3D-Stacked Memory
5. Results
6. Conclusion

Key Properties of GRIM-Filter

1. **Simple Operations**:
   - To check a given bin, find the **sum** of all bits corresponding to each token in the read
   - **Compare** against threshold to determine whether to align

2. **Highly Parallel**: Each bin is operated on independently and there are many many bins

3. **Memory Bound**: Given the frequent accesses to the large bitvectors, we find that GRIM-Filter is memory bound

These properties together make GRIM-Filter a good algorithm to be run in 3D-Stacked DRAM

Hash Tables in Read Mapping

Read Sequence (100 bp)

Aligning…

Match!

Mismatch

Aligning…

False Negative

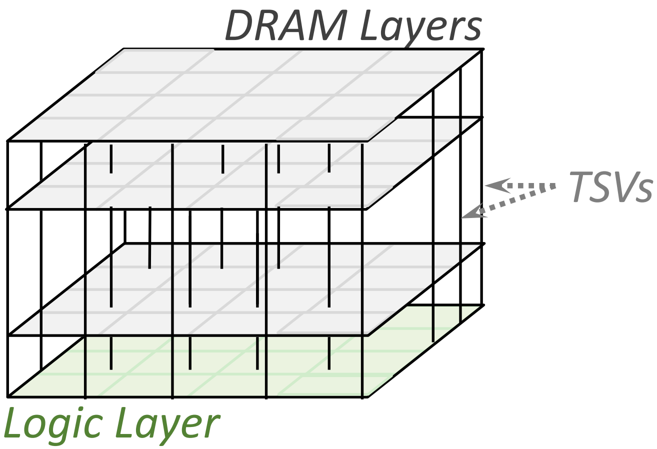
Hash Table

37 140
894 1203
1564

Reference Genome

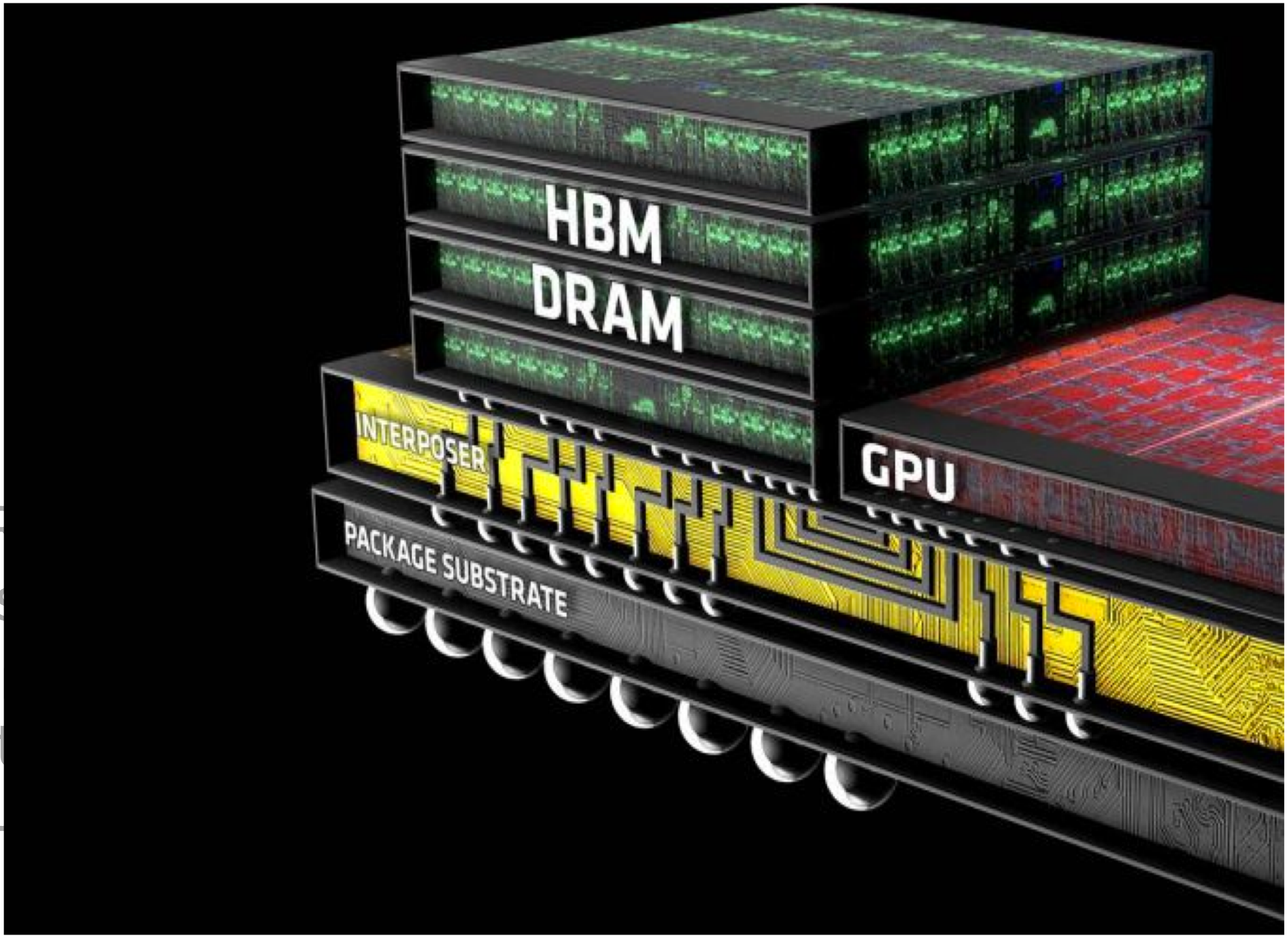
Filter

3D-Stacked Memory



- 3D-Stacked DRAM architecture has **extremely high bandwidth** as well as a stacked customizable logic layer
  - Logic Layer enables **Processing-in-Memory**, offloading computation to this layer and alleviating the memory bus
  - Embed GRIM-Filter operations into **DRAM logic layer** and appropriately distribute bitvectors throughout memory

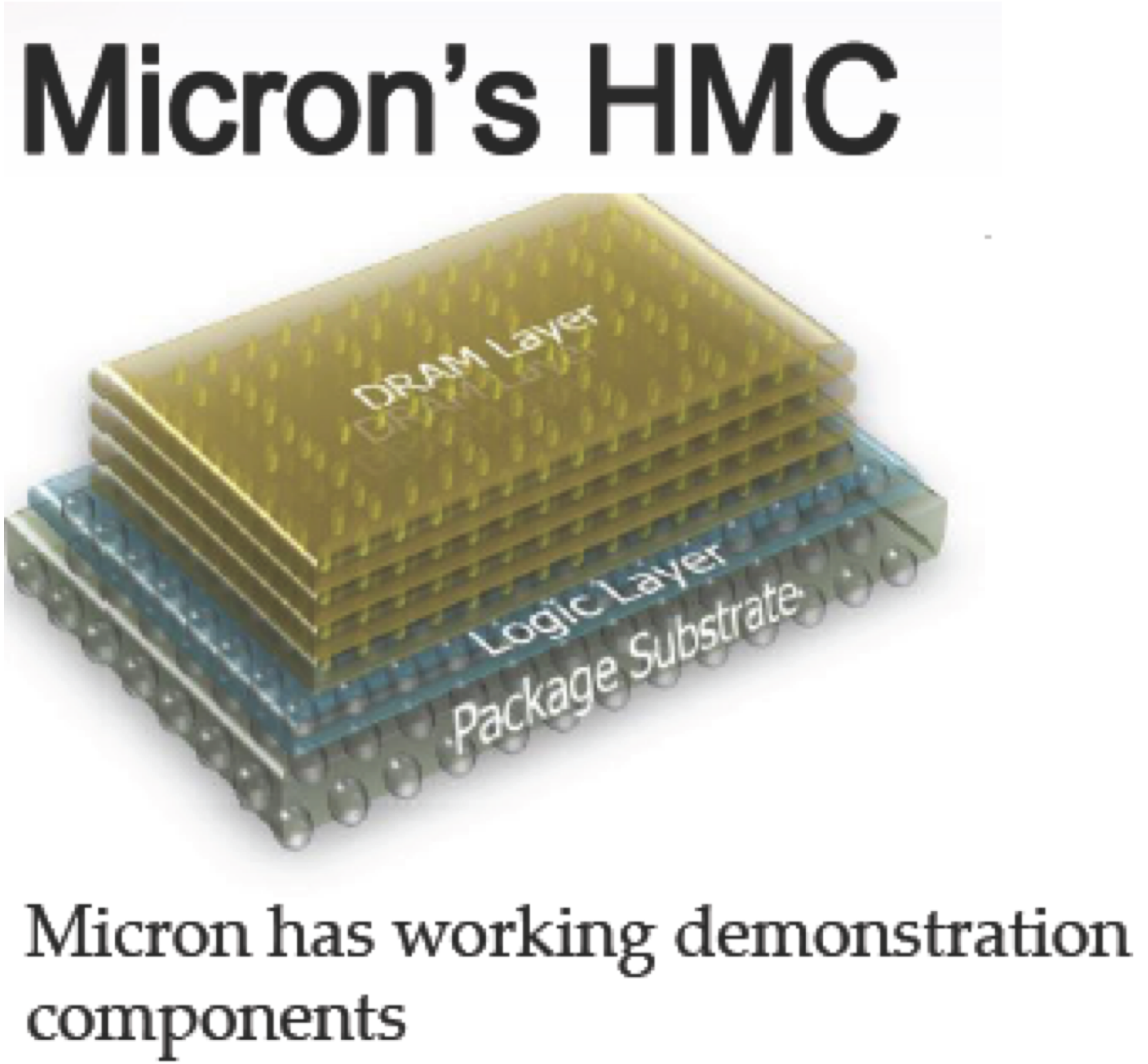
3D-Stacked Memory



- 3D-Stacked DF
  bandwidth as
  - Logic Layer e
    computation t
  - Embed GRIM-
    appropriately

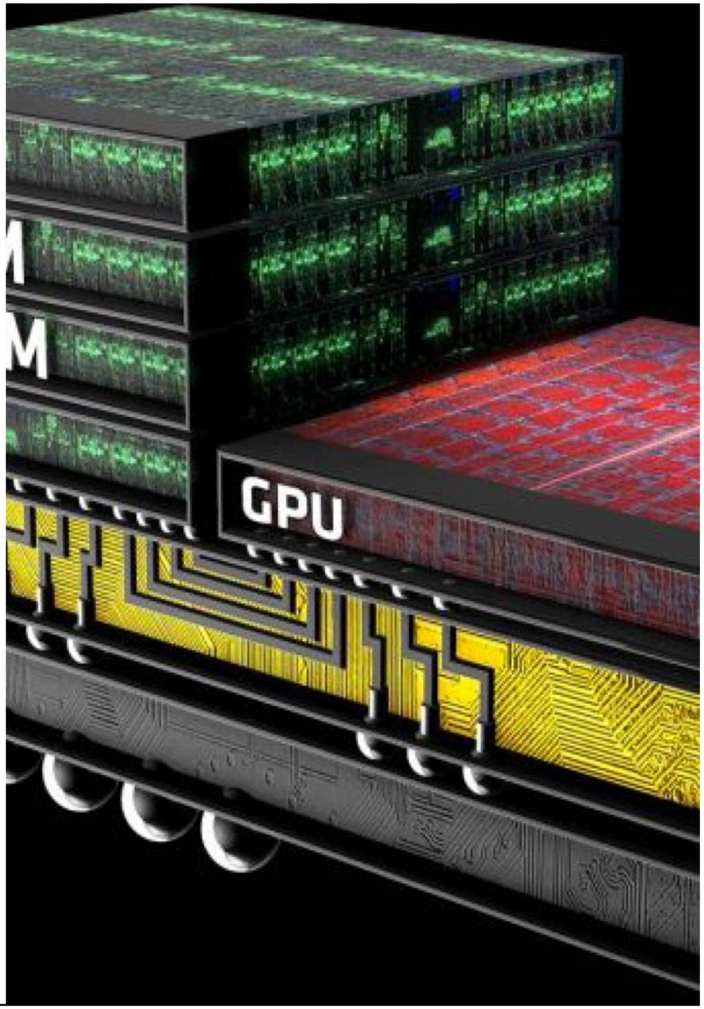
http://i1-news.softpedia-static.com/images/news2/Micron-and-Samsung-Join-Force-to-Create-Next-Gen-Hybrid-Memory-2.png

3D-Stacked Memory



Micron's HMC

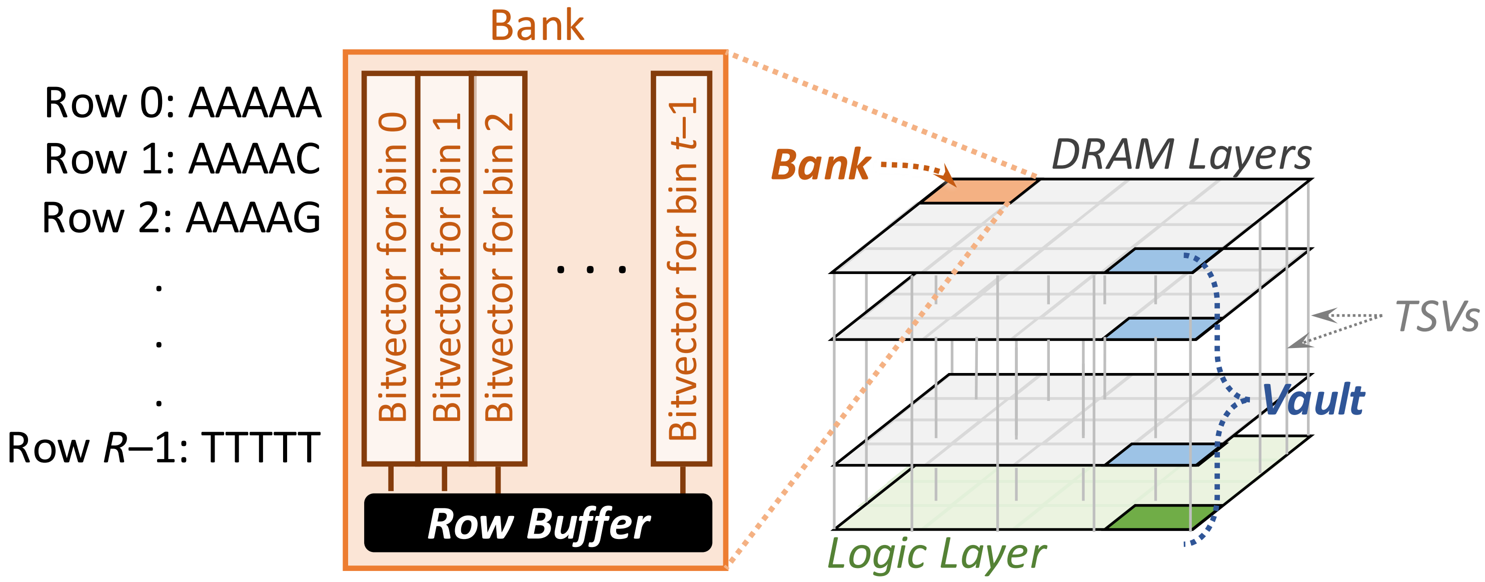
Micron has working demonstration components



http://images.anandtech.com/doci/9266/HBMCar_678x452.jpg

http://i1-news.softpedia-static.com/images/news2/Micron-and-Samsung-Join-Force-to-Create-Next-Gen-Hybrid-Memory-2.png

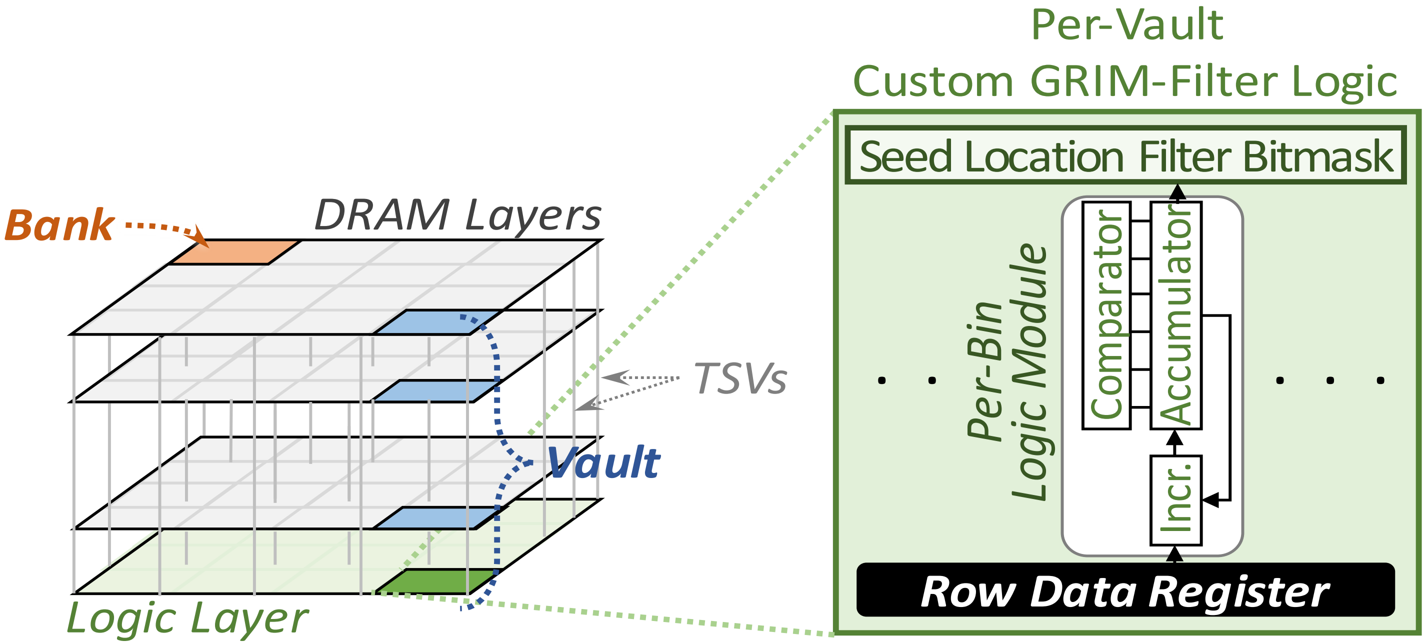
GRIM-Filter in 3D-Stacked DRAM



- Each DRAM layer is organized as an array of **banks**
    - A **bank** is an array of cells with a row buffer to transfer data

- The layout of bitvectors in a bank enables filtering many bins in parallel

GRIM-Filter in 3D-Stacked DRAM



- Customized logic for accumulation and comparison per genome segment
  - Low area overhead, simple implementation
  - For HBM2, we use 4096 incrementer LUTs, 7-bit counters, and comparators in logic layer

Details are in the paper

GRIM-Filter Outline

1. Motivation and Goal
2. Background: Read Mappers
   - a. Hash Table Based
   - b. Hash Table Based with Filter
3. Our Proposal: GRIM-Filter
4. Mapping GRIM-Filter to 3D-Stacked Memory
5. **Results**
6. Conclusion

Methodology

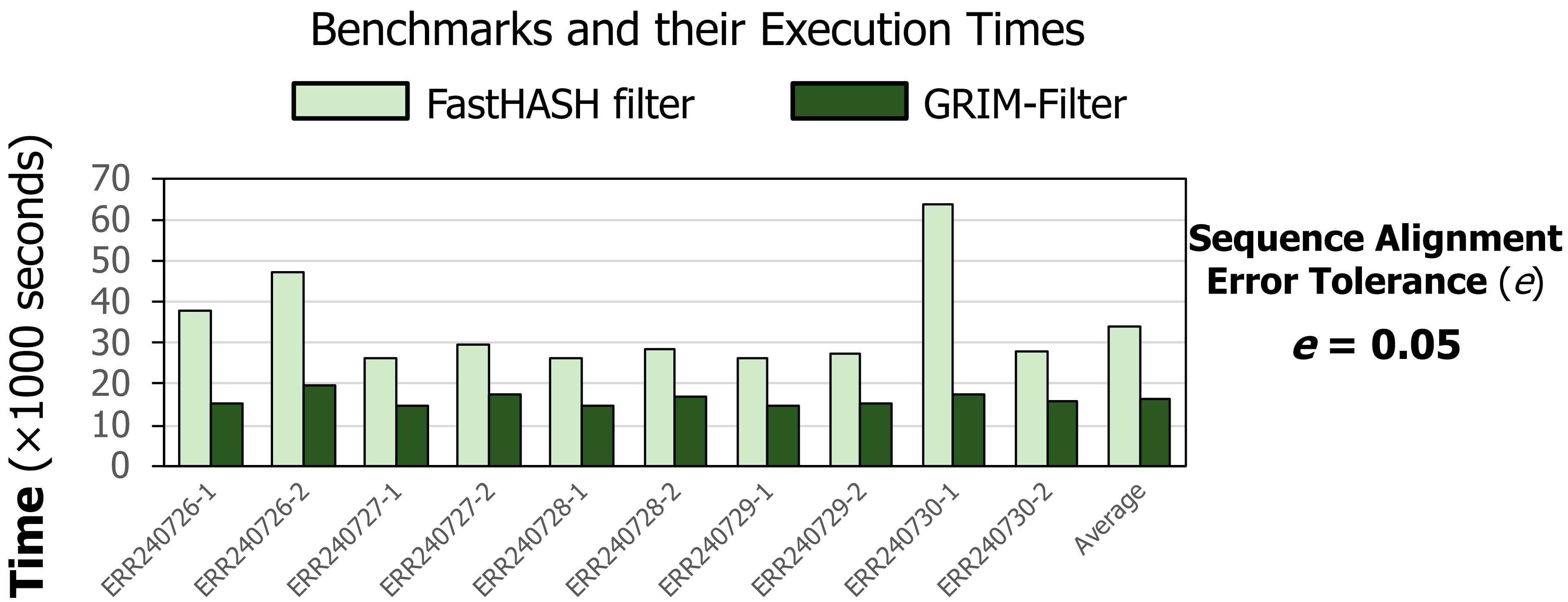
- Performance simulated using an in-house 3D-Stacked DRAM simulator

- Evaluate 10 real read data sets (From the 1000 Genomes Project)
  - Each data set consists of 4 million reads of length 100

- Evaluate two key metrics
  - Performance
  - False negative rate
    - The fraction of locations that pass the filter but result in a mismatch

- Compare against a state-of-the-art filter, FastHASH **[Xin+, BMC Genomics 2013]** when using mrFAST, but **GRIM-Filter can be used with ANY read mapper**

GRIM-Filter Performance

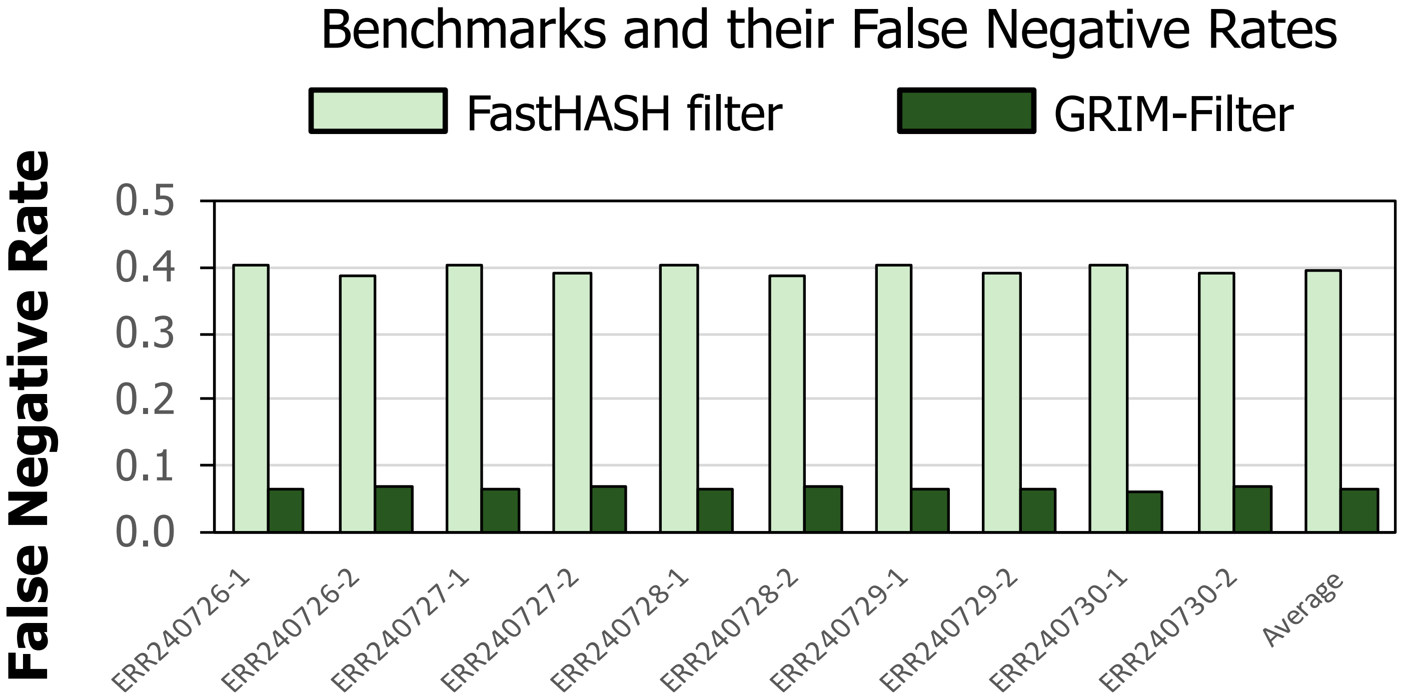


1.8x-3.7x performance benefit across real data sets

2.1x average performance benefit

GRIM-Filter gets performance due to its hardware-software co-design

GRIM-Filter False Negative Rate



Sequence Alignment Error Tolerance (*e*)

***e* = 0.05**

5.6x-6.4x False Negative reduction across real data sets

6.0x average reduction in False Negative Rate

GRIM-Filter utilizes more information available in the read to filter

Other Results in the Paper

- Sensitivity of execution time and false negative rates to error tolerance of string matching

- Read mapper execution time breakdown

- Sensitivity studies on the filter
  - Token Size
  - Bin Size
  - Error Tolerance

GRIM-Filter Outline

1. Motivation and Goal
2. Background: Read Mappers
   - a. Hash Table Based
   - b. Hash Table Based with Filter
3. Our Proposal: GRIM-Filter
4. Mapping GRIM-Filter to 3D-Stacked Memory
5. Results
6. **Conclusion**

Conclusion

We propose an in-memory filtering algorithm to accelerate end-to-end read mapping by reducing the number of required alignments

Key ideas:

- Introduce a **new representation** of coarse-grained segments of the reference genome
- Use **massively-parallel in-memory operations** to identify read presence within each coarse-grained segment

Key contributions and results:

- Customized filtering algorithm for 3D-Stacked DRAM
- Compared to the previous best filter
  - We observed 1.8x-3.7x read mapping speedup
  - We observed 5.6x-6.4x fewer false negatives

GRIM-Filter is a universal filter that can be applied to any read mapper

GRIM-Filter:

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

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

Systems@ETH zürich

SAFARI

Carnegie Mellon

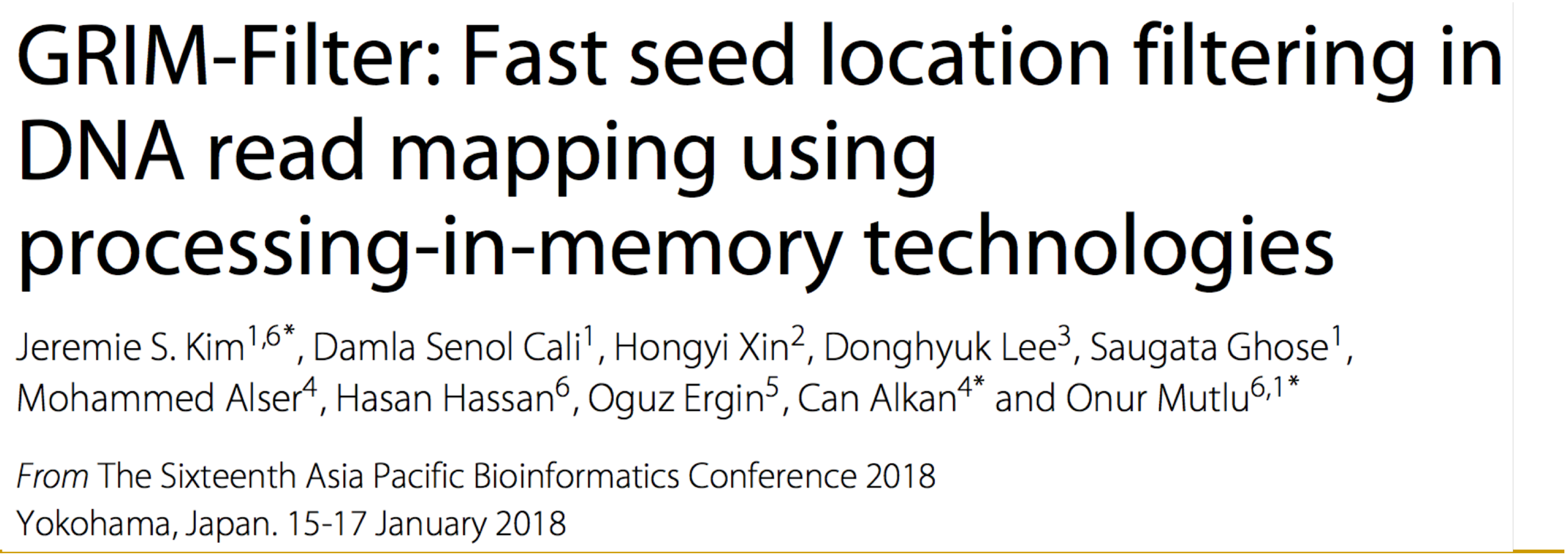
İhsan Doğramacı Bilkent University

TOBB
UNIVERSITY OF
ECONOMICS AND TECHNOLOGY

ETH zürich

In-Memory DNA Sequence Analysis

- 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*]

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

LazyPIM

An Efficient Cache Coherence Mechanism for Processing In Memory

Amirali Boroumand

"LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory",
IEEE CAL 2016. (Preliminary version)

SAFARI

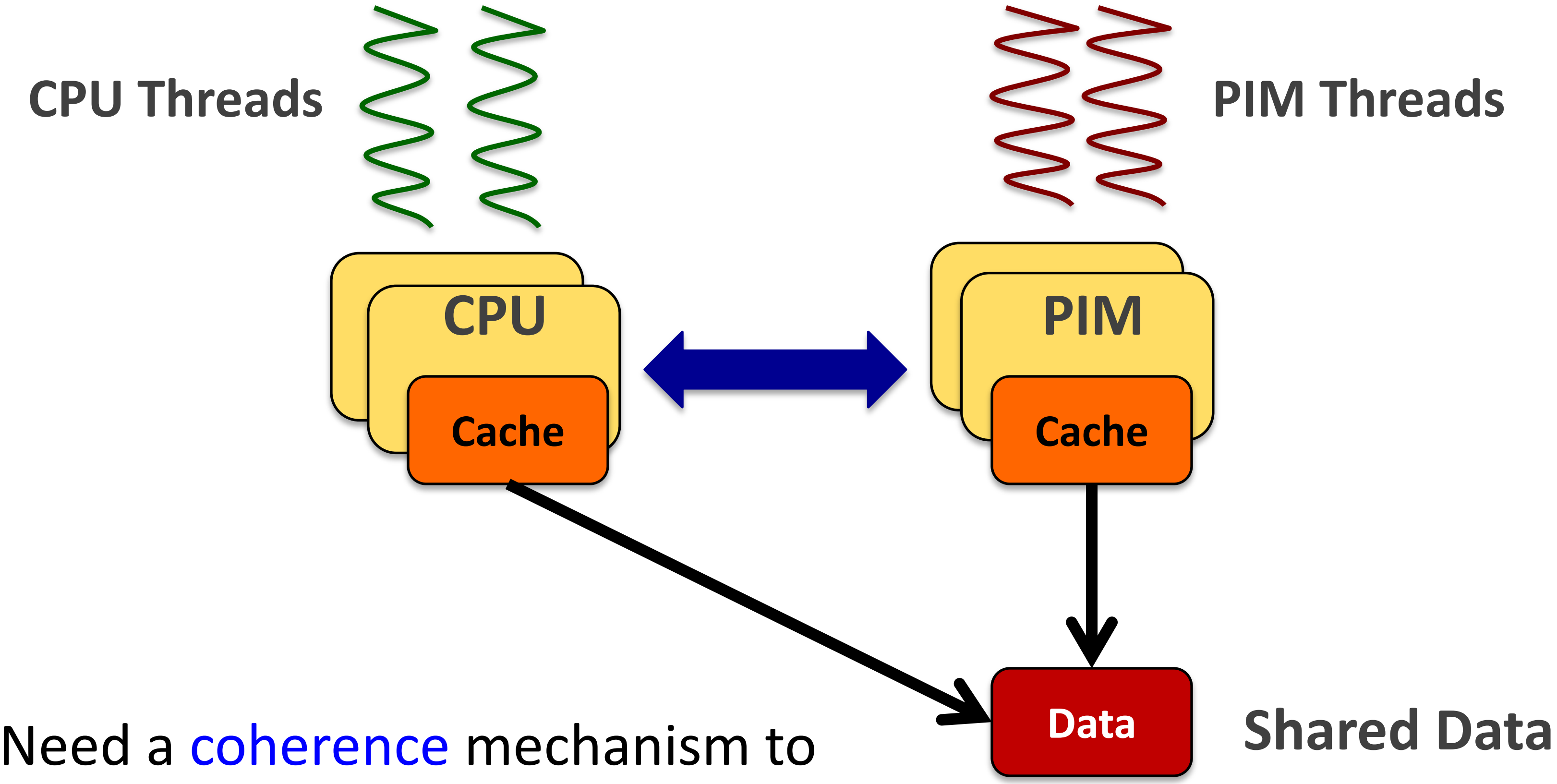
Carnegie Mellon

LazyPIM Summary

- Cache Coherence is a major system challenge for PIM
  - Conventional cache coherence makes PIM programming easy but loses a significant portion of PIM benefits
- Observation:
  - Significant amount of sharing between PIM cores and CPU cores in many important data-intensive applications
  - Efficient handling of coherence is critical to retain PIM benefits
- LazyPIM
  - Key idea: use speculation to avoid coherence lookups during PIM core execution and compressed signatures to verify correctness after PIM core is done
  - Improves performance by 19.8% and energy by 18% vs. best previous
  - Comes within 4.4% and 9.8% of ideal PIM *energy* and *performance*
- We believe LazyPIM can enable new applications that benefit from fine-grained sharing between CPU and PIM

PIM Coherence

- **A Major System Challenge for PIM: Coherence**

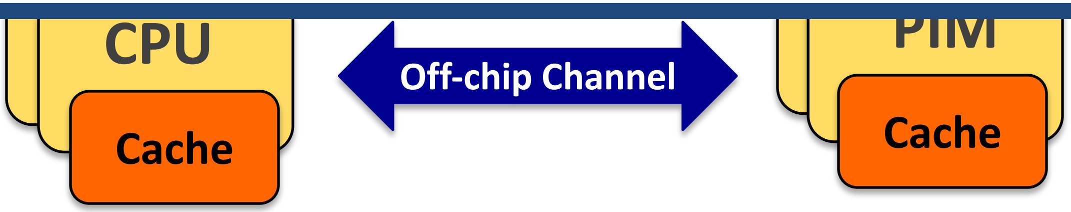


Need a coherence mechanism to ensure correctness!

PIM Coherence

- **Potential solution: Conventional coherence protocols**
  - We can treat PIM cores as **additional independent cores**
  - Use **conventional coherence protocol** to make them coherent with the CPUs

Conventional coherence is impractical: large number of coherence messages over off-chip channel



✓ Simplifies PIM programming model

✗ Generates a large amount of off-chip coherence traffic

✗ Eliminates on average 72.4% of Ideal PIM energy improvement

Goal and Key Idea

- **Our goal is to develop a cache coherence mechanism that:**

  1) Maintains the **logical behavior** of conventional cache coherence protocols to simplify **PIM programming model**

  2) **Retains** the large **performance and energy benefits** of PIM

- **Our key idea is**

  1) Avoid *coherence lookups* during PIM core execution

  2) **Batch lookups** in compressed signatures and use them to **verify correctness** after PIM core finishes

Background

Prior Approaches to PIM Coherence

Prior Approaches to PIM Coherence

- **There are many recent proposals on PIM**
  - Primarily focus on the design of compute unit within the logic layer

- **Prior works employ other approaches than conventional coherence protocol**
  - Marking PIM-data as **Non-cacheable**
    - They no longer need to deal with coherence
  - **Coarse-grained** coherence
    - Tracks coherence at a larger granularity than a single cache line
    - Does not transfer permission while PIM is working
    - **No concurrent access** from the CPU and PIM

Prior Approaches to PIM Coherence

- **Prior works proposed coherence mechanisms assuming:**
  - Entire application could be offloaded to PIM core → **Almost zero sharing** between PIM and CPU
  - Only **limited** communication happens between CPU and PIM

Observation: These assumptions *do not hold* for many important data-intensive applications that benefit from PIM

Motivation

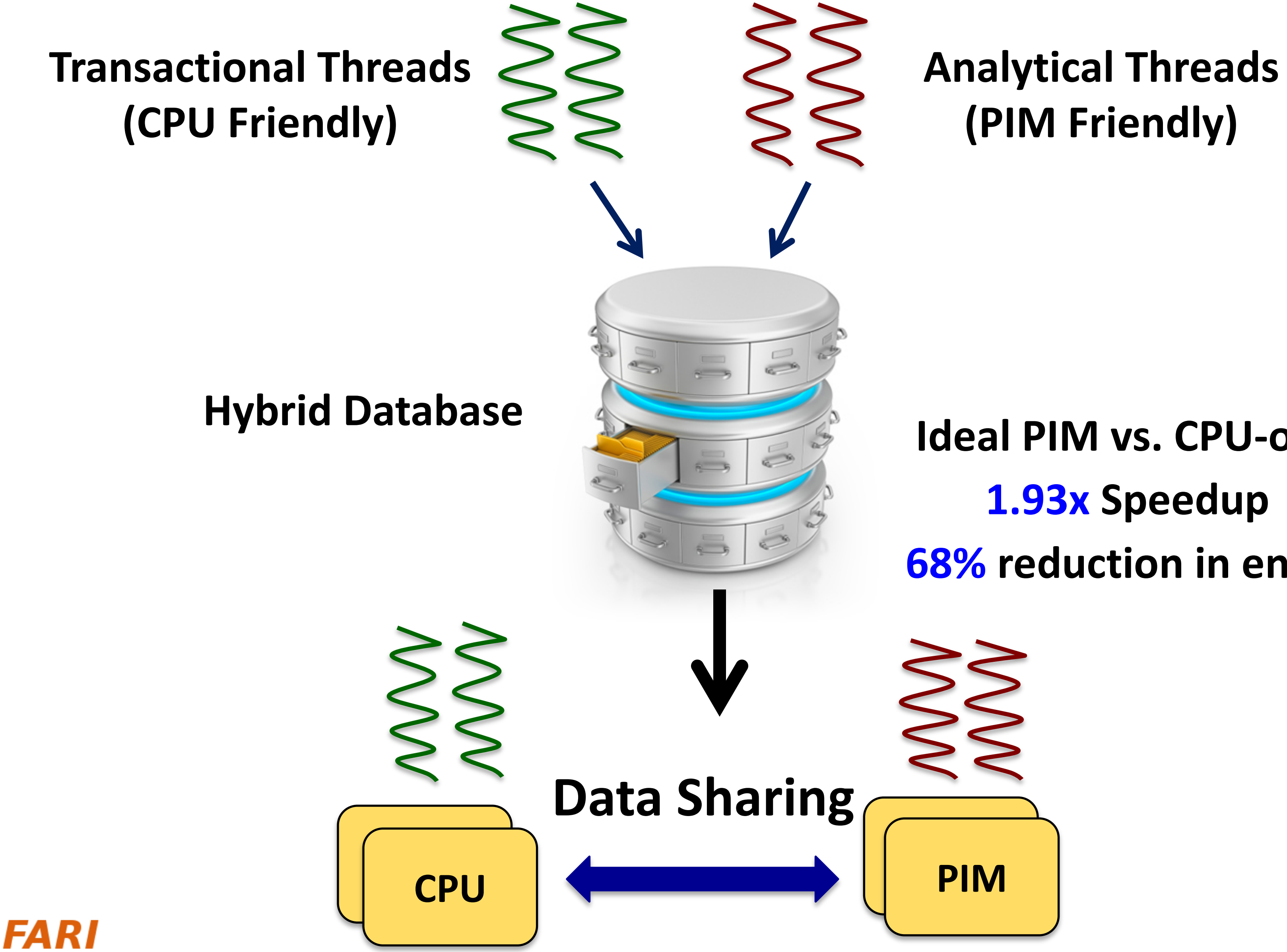
Applications with Data Sharing

Application Analysis for PIM

- **An application benefits from PIM when we offload its memory-intensive parts that:**
  - Generate a lot of data movement
  - Have poor cache locality
  - Contribute to a large portion of execution time

- **Parts of the application that are compute-intensive or cache friendly should remain on the CPU**
  - To benefit from larger and sophisticated cores with larger caches

Example: Hybrid In-Memory Database



Transactional Threads (CPU Friendly)

Analytical Threads (PIM Friendly)

Hybrid Database

Data Sharing

CPU

PIM

Ideal PIM vs. CPU-only:

1.93x Speedup

68% reduction in energy

Applications with High Data Sharing

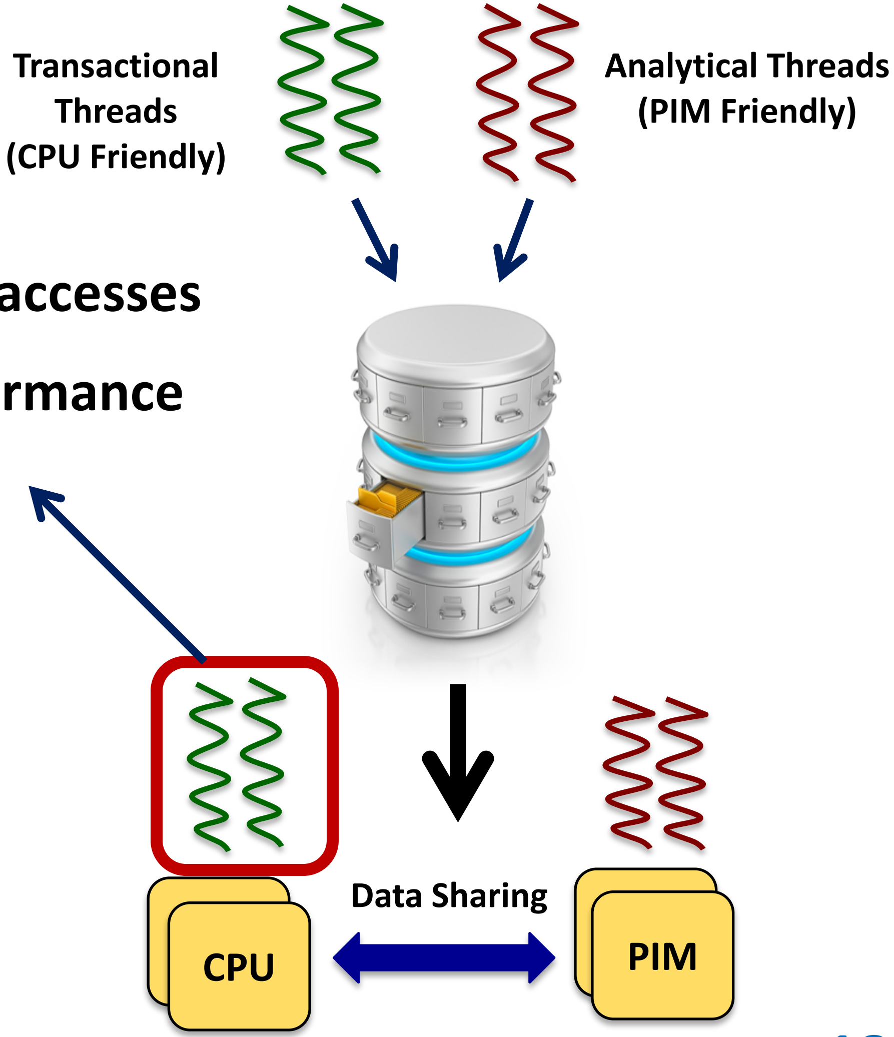
- Our application analysis shows that:
  - Some portions of the applications perform better on CPUs
  - These portions often access the same region of data as the PIM cores

- Based on this observation, we can conclude that:
  - There are important data-intensive applications that have strong potential for PIM and show significant data sharing between the CPU and PIM

Let’s see how prior approaches work for these applications

Non-Cacheable

- ✗ Generates **a large number of off-chip** accesses
- ✗ Significantly hurts **CPU threads'** performance


Transactional Threads (CPU Friendly)
Analytical Threads (PIM Friendly)
Data Sharing
CPU
PIM

Motivation: Summary

- Conventional cache coherence loses a significant portion of PIM benefits

- Prior works use other approaches to avoid those costs
  - Their assumption: Zero or a limited amount of sharing

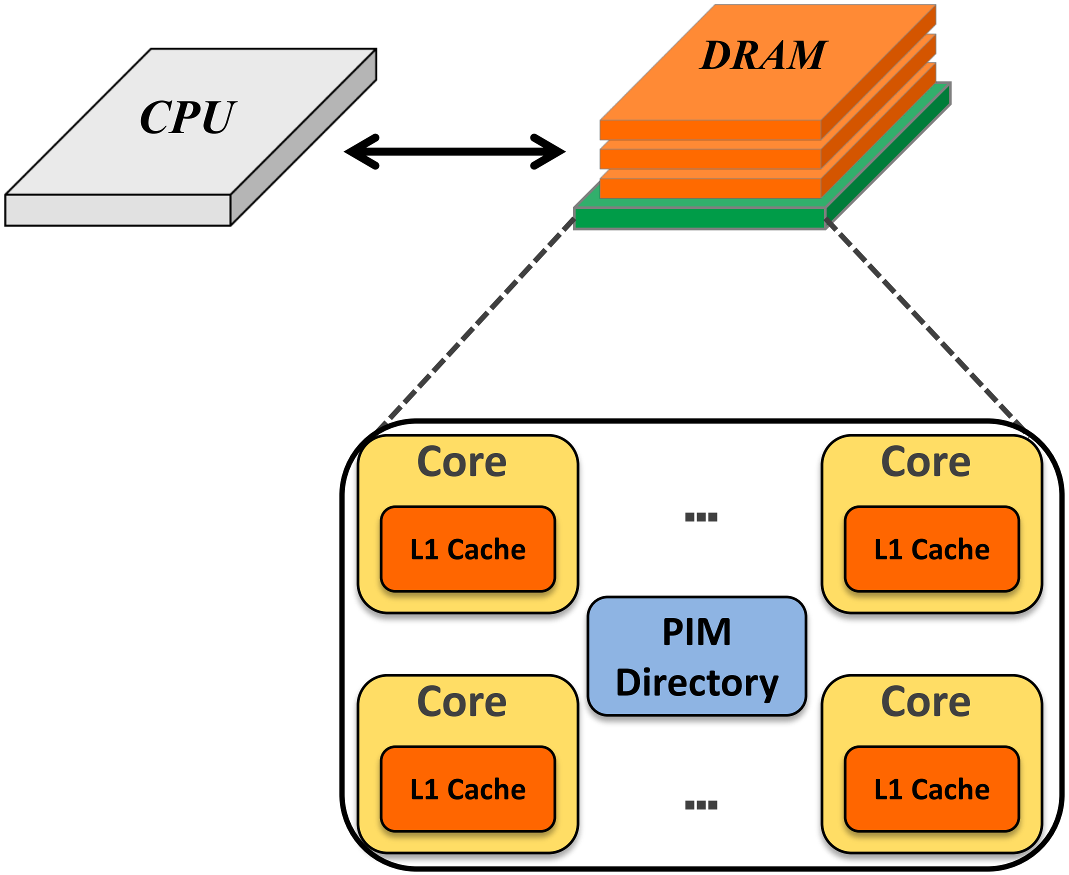
- We observe that those assumptions do not hold for a number of important data-intensive applications
  - Using prior approaches eliminates a significant portion of PIM benefits

- We want to get the best of both worlds

  1) Maintain the logical behavior of conventional cache coherence
  2) Retain the large performance and energy benefits of PIM

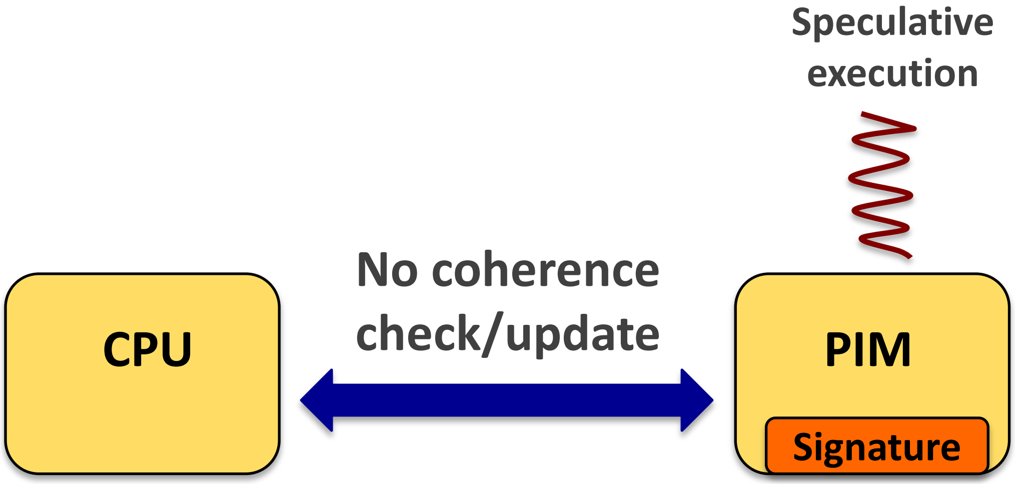
LazyPIM

Baseline PIM Architecture



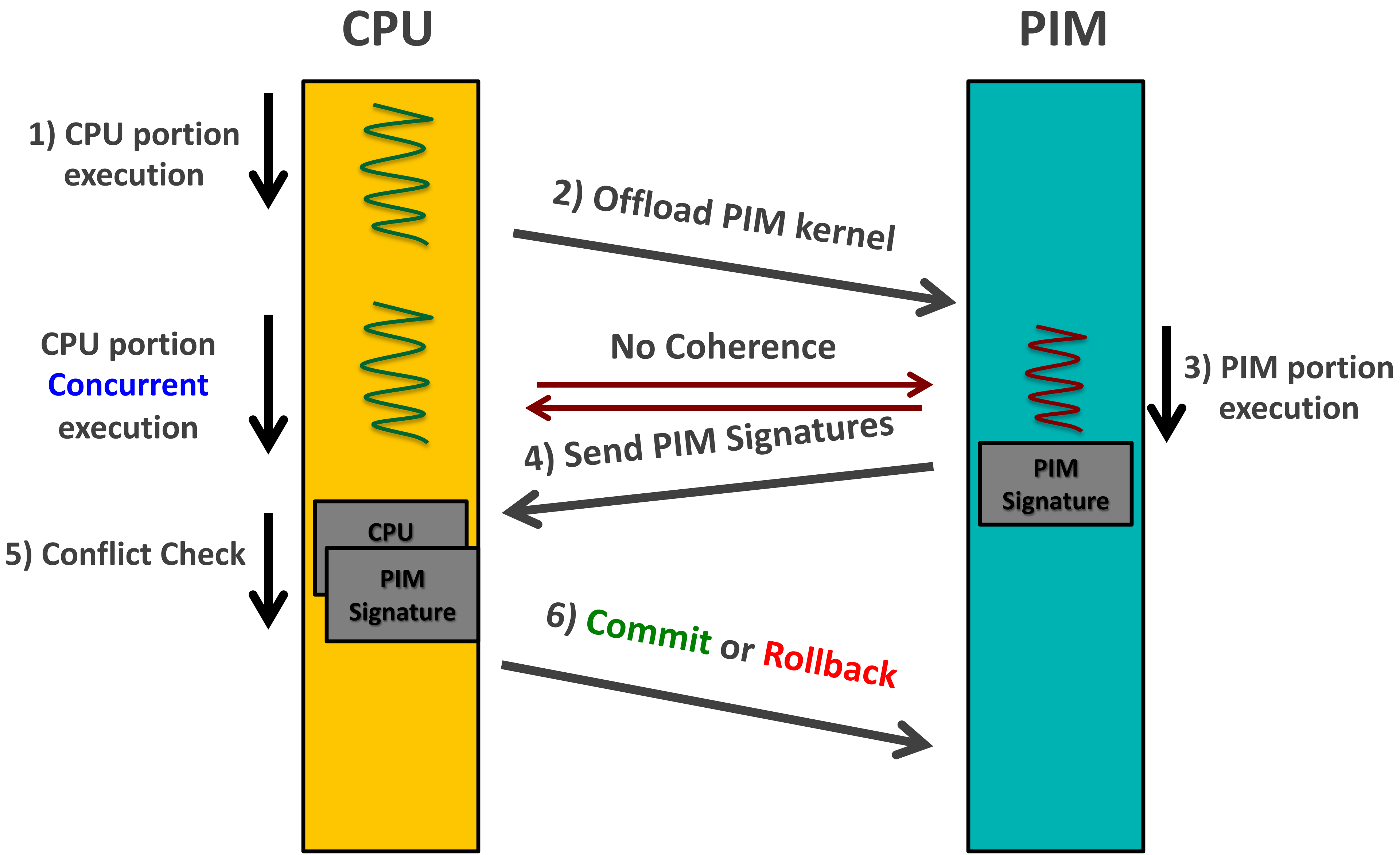
Our Proposal

- **LazyPIM:**
  - Lets PIM cores use ***speculation** to **avoid*** coherence lookups *during execution*
  - Uses **compressed *signatures*** to batch the lookups and verify correctness *after* the PIM core completes



Verify Correctness

LazyPIM High-level Operation



CPU

PIM

1) CPU portion execution

2) Offload PIM kernel

3) PIM portion execution

No Coherence

CPU portion Concurrent execution

4) Send PIM Signatures

PIM Signature

5) Conflict Check

CPU

PIM Signature

6) Commit or Rollback

How LazyPIM Avoids Pitfalls of Prior Approaches

- **Conventional Coherence (Fine-grained)**

✗ Generates a large amount of off-chip coherence traffic *for every miss*

✓ LazyPIM only sends a compressed signature after PIM cores finishes

- **Coarse-grained Coherence**

✗ Unnecessarily flushes a large amount of data

✓ LazyPIM performs only the necessary flushes

✗ Causes Thread Serialization

✓ LazyPIM enables concurrent execution of the CPUs and PIM cores

- **Non-Cacheable**

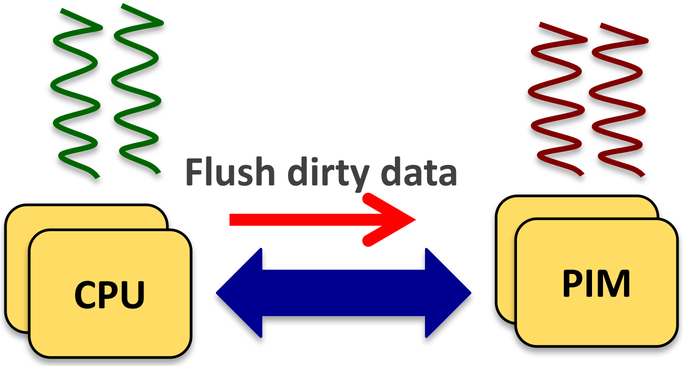
✗ A large number of off-chip accesses hurting CPU threads' performance

✓ LazyPIM allows CPU threads to use caches

Coarse-Grained Coherence

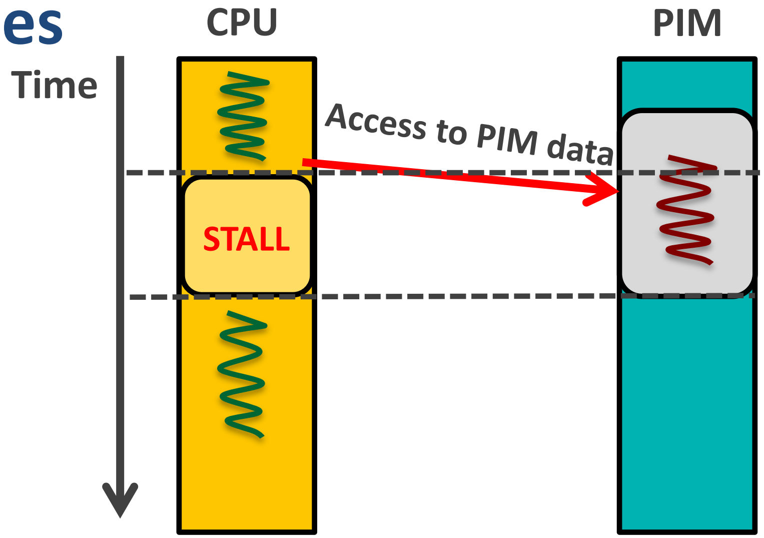
- Need to get coherence permission for the entire region
  - Needs to flush every dirty data within that region to transfer permission

❌ Unnecessarily flushes a large amount of data in pointer-based data structure



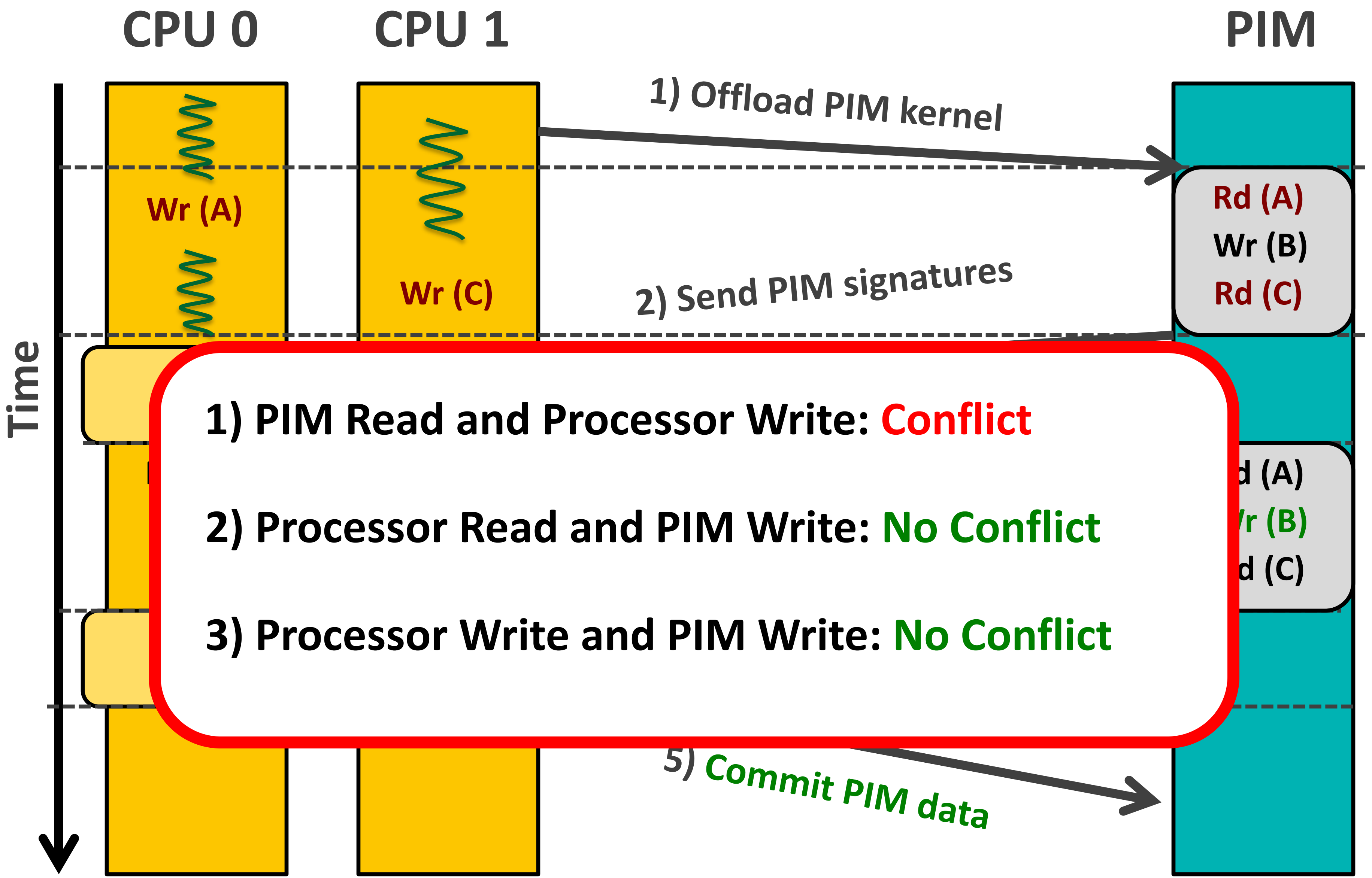
- Does not allow concurrent accesses
  - Blocks CPUs accessing PIM-data during PIM execution

❌ Coarse-grained locks frequently cause thread serialization



How we define conflicts in LazyPIM?

Conflicts



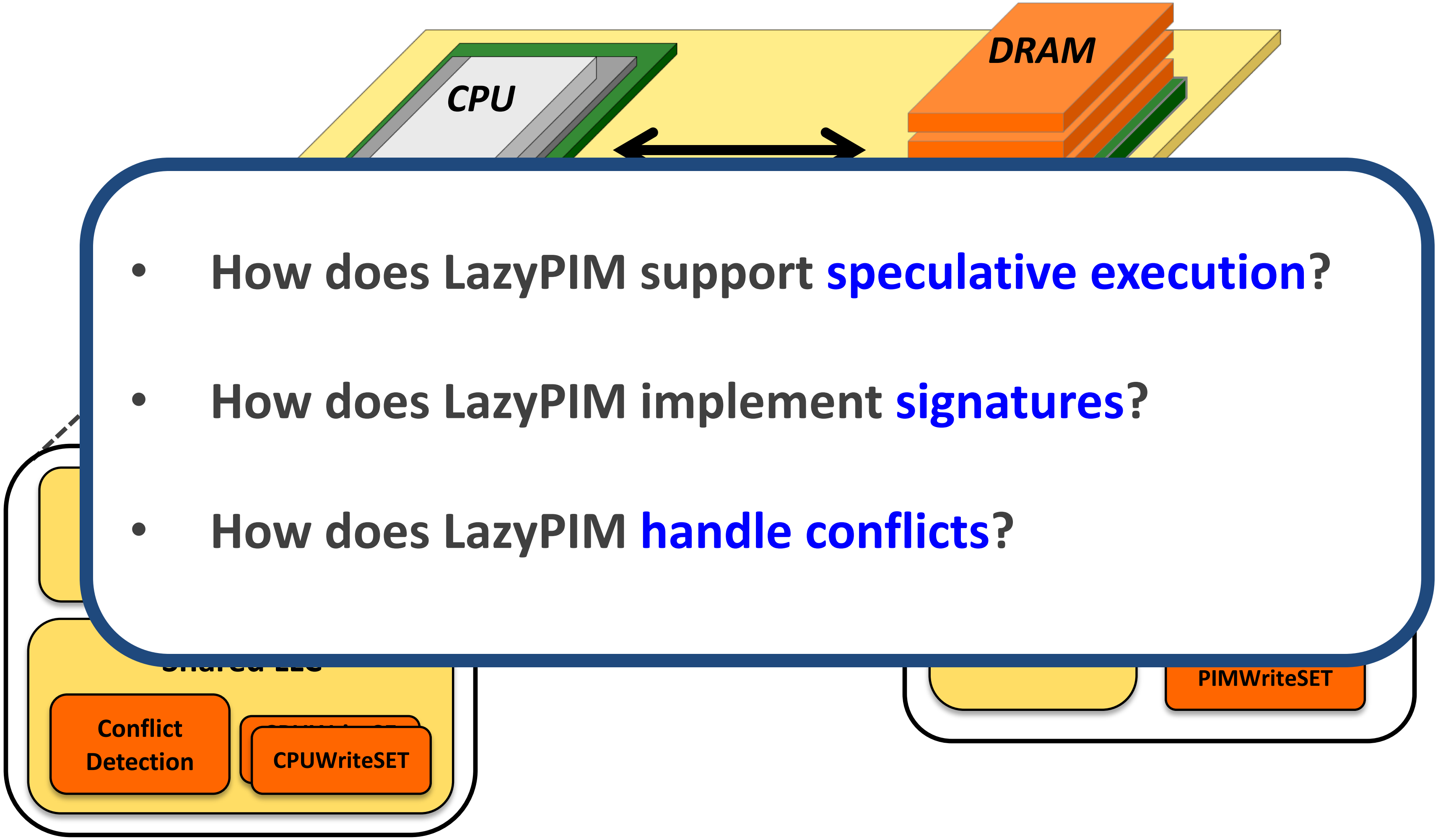
1) PIM Read and Processor Write: Conflict

2) Processor Read and PIM Write: No Conflict

3) Processor Write and PIM Write: No Conflict

Architecture Support

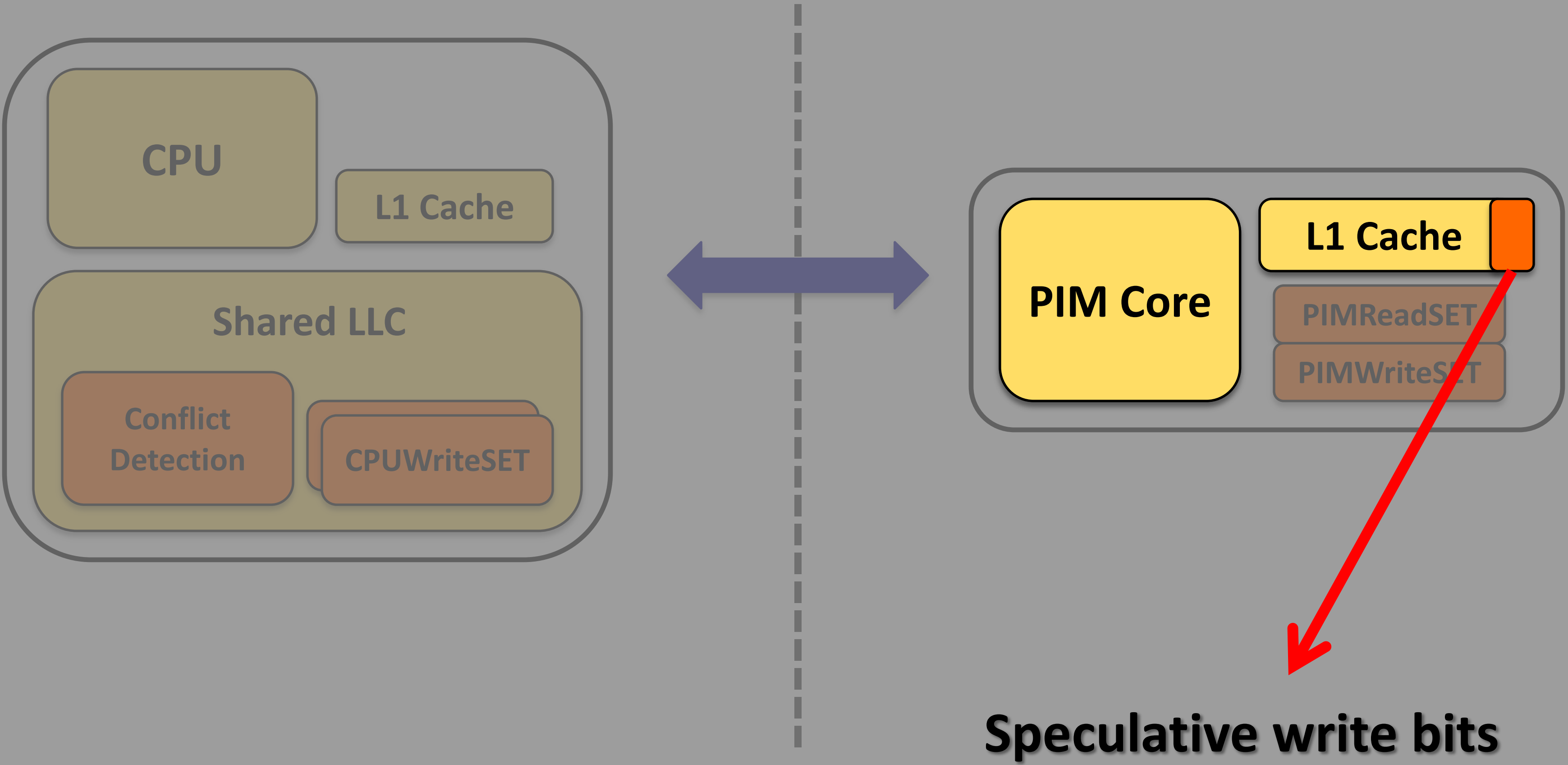
LazyPIM Architecture



- How does LazyPIM support **speculative execution**?
- How does LazyPIM implement **signatures**?
- How does LazyPIM **handle conflicts**?

Tracking speculative updates

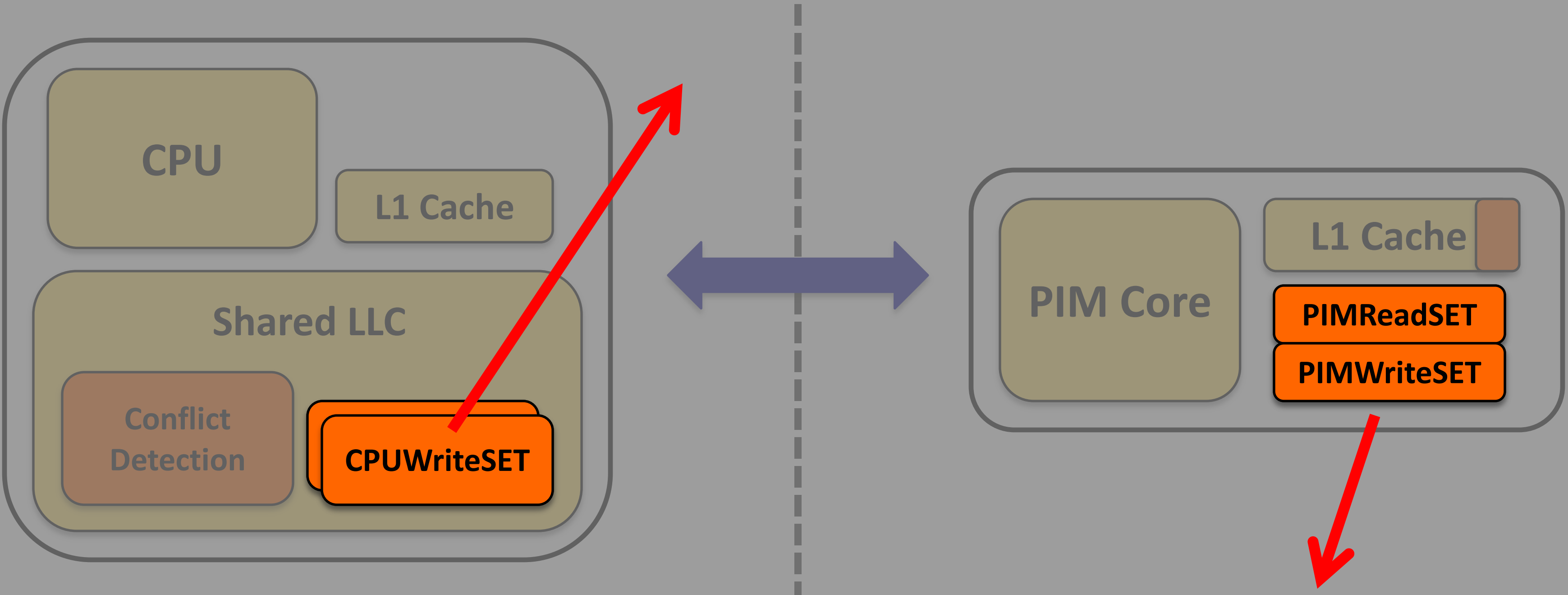
- One-bit flag per cache line to mark all data updates as speculative



Specula

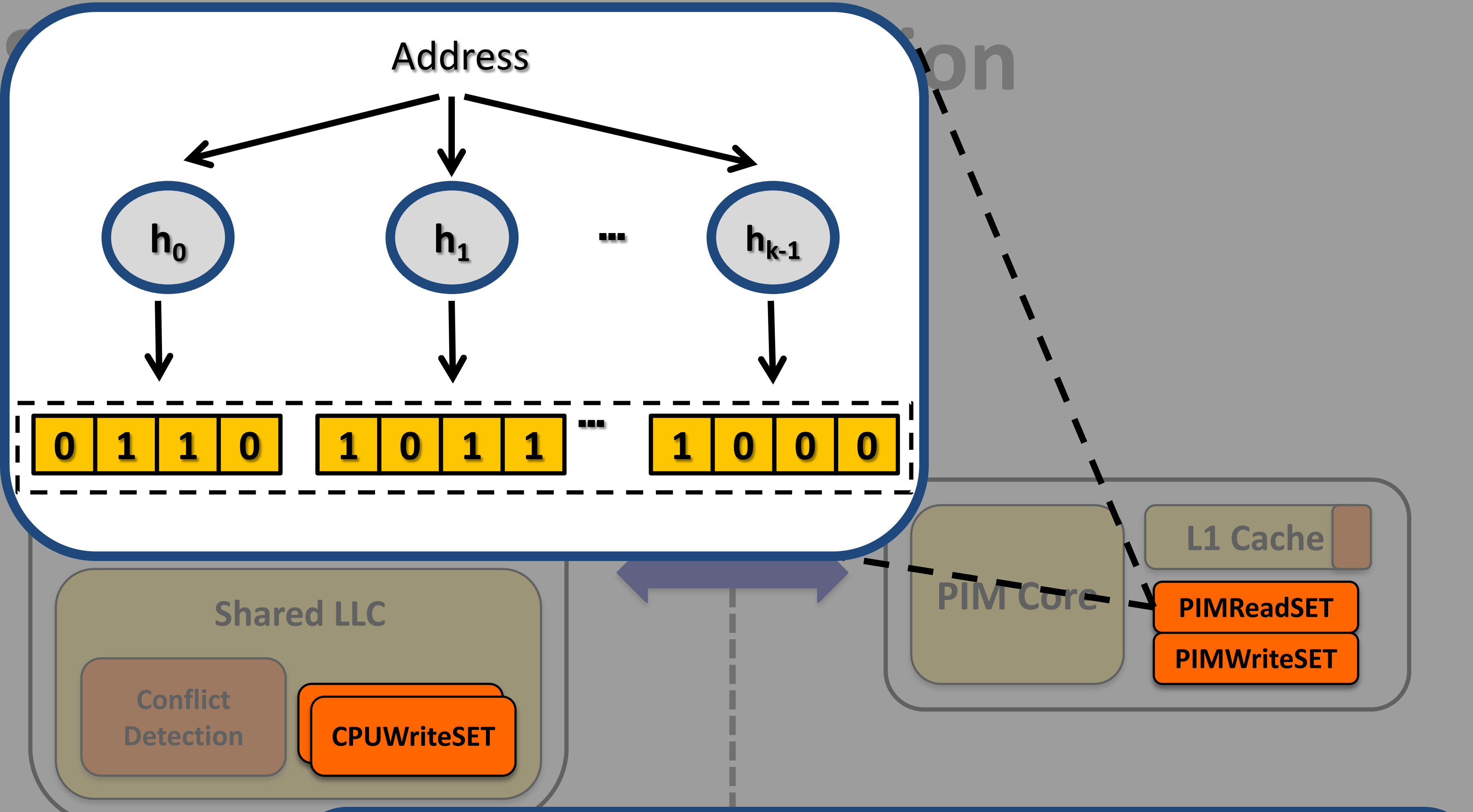
Tracking potential conflicts

- The CPU records all dirty cache lines and writes in the PIM data region in the CPUWriteSet



Tracking memory accesses

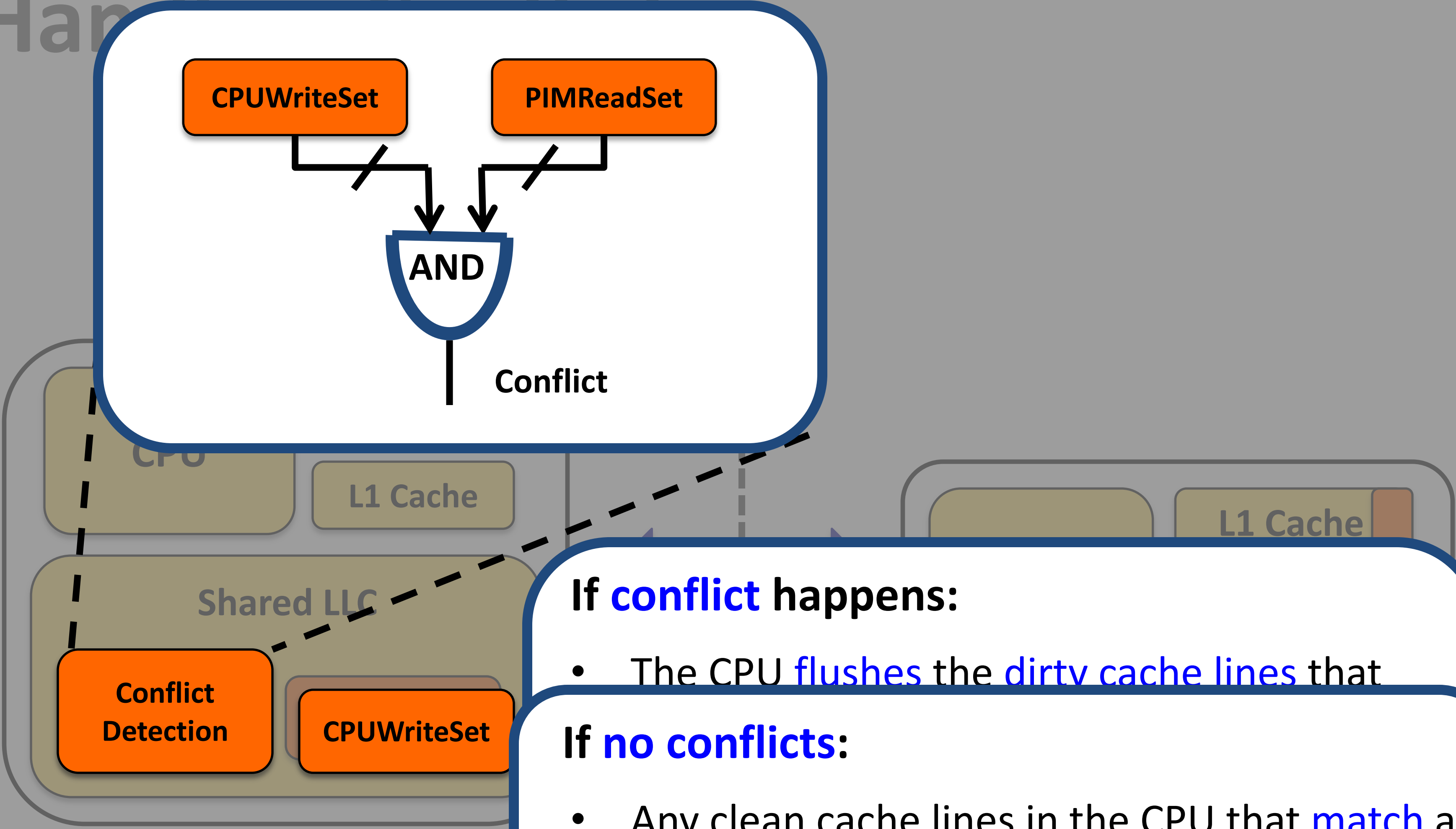
- The PIMReadSet and PIMWriteSet are updated for every read and write by the PIM core



Bloom filter based signature has two major benefits:

- Allows us to easily perform conflict detection
- Allows for a large number of addresses to be stored within a fixed-length register

Handling Conflicts



If conflict happens:

- The CPU flushes the dirty cache lines that

If no conflicts:

- Any clean cache lines in the CPU that match an address in the PIMWriteSet are invalidated
- PIM core commits speculative updates

Evaluation

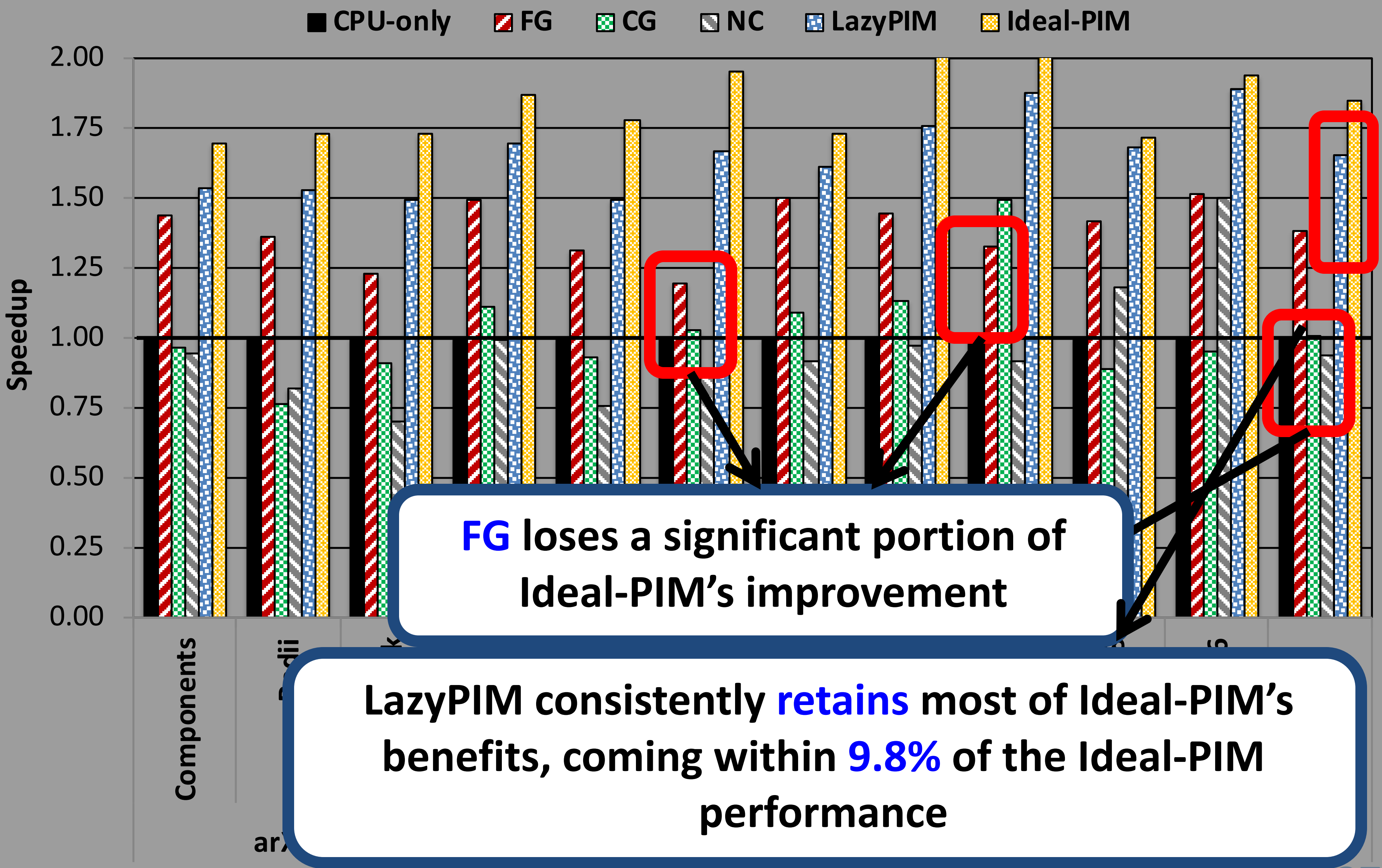
Evaluation Methodology

- **Simulator**
  - Gem5 full system simulator
- **System Configuration:**
  - **Processor**
    - 4-16 Cores, 8 wide issue, 2GHz Frequency
    - L1 I/D Cache: 64KB private, 4-way associative, 64B Block
    - L2 Cache: 2MB shared, 8-way associative, 64B Blocks
    - Cache Coherence Protocol: MESI
  - **PIM**
    - 4-16 Cores, 1 wide issue, 2GHz Frequency
    - L1 I/D Cache: 64KB private, 4-way associative, 64B Block
    - Cache Coherence Protocol: MESI
  - **3D-stacked Memory**
    - One 4GB Cube, 16 Vaults per cube

Applications

- **Ligra**
  - Lightweight multithreaded graph processing for shared memory system
  - We used three Ligra graph applications
    - PageRank
    - Radii
    - Connected Components
  - Input graphs constructed from real-world network datasets:
    - arXiV General Relativity (5K nodes, 14K edges)
    - peer-to- peer Gnutella25 (22K nodes, 54K edges).
    - Enron email communication network (36K nodes, 183K edges)
- **IMDB**
  - In-house prototype of an in-memory database (IMDB)
  - Capable of running both **transactional** queries and **analytical** queries on the **same** database tables (**HTAP workload**)
  - 32K transactions, 128/256 analytical queries

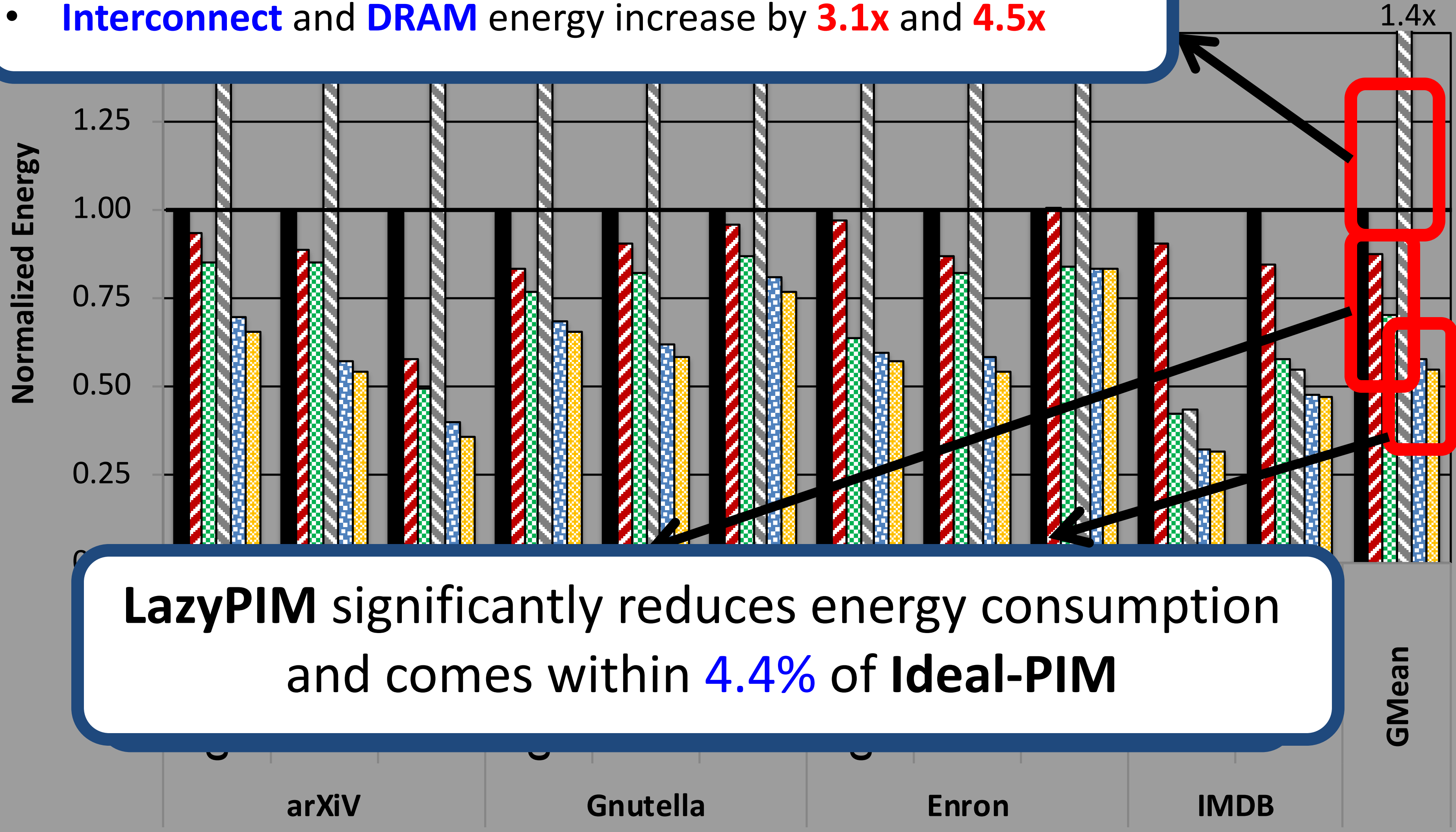
Speedup with 16 Threads



FG loses a significant portion of Ideal-PIM's improvement

LazyPIM consistently **retains** most of Ideal-PIM's benefits, coming within **9.8%** of the Ideal-PIM performance

Energy with 16 threads



- **NC** suffers greatly from the *large number of accesses to DRAM*
- **Interconnect** and **DRAM** energy increase by **3.1x** and **4.5x**

LazyPIM significantly reduces energy consumption and comes within 4.4% of **Ideal-PIM**

Conclusion

Conclusion

- Cache Coherence is a **major system challenge** for PIM
  - Conventional cache coherence makes PIM programming easy but **loses a significant portion of PIM benefits**
- Observation:
  - **Significant amount of sharing** between **PIM cores** and **CPU cores** in many important data-intensive applications
  - Efficient **handling of coherence** is **critical** to retain PIM benefits
- LazyPIM
  - **Key idea**: use **speculation** to **avoid coherence lookups** during PIM core execution and **compressed signatures** to verify correctness after PIM core is done
  - Improves performance by **19.8%** and energy by **18% vs. best previous**
  - Comes within **4.4%** and **9.8%** of **ideal PIM** *energy* and *performance*
- **We believe LazyPIM can enable new applications that benefit from fine-grained sharing between CPU and PIM**

LazyPIM

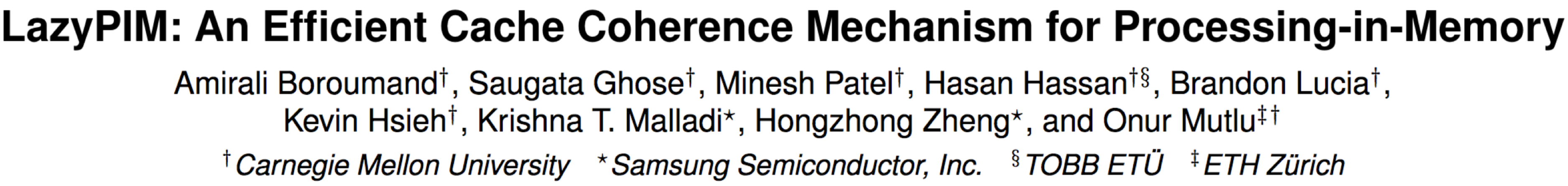
An Efficient Cache Coherence Mechanism for Processing In Memory

Amirali Boroumand

"LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory",
IEEE CAL 2016. (Preliminary version)

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.

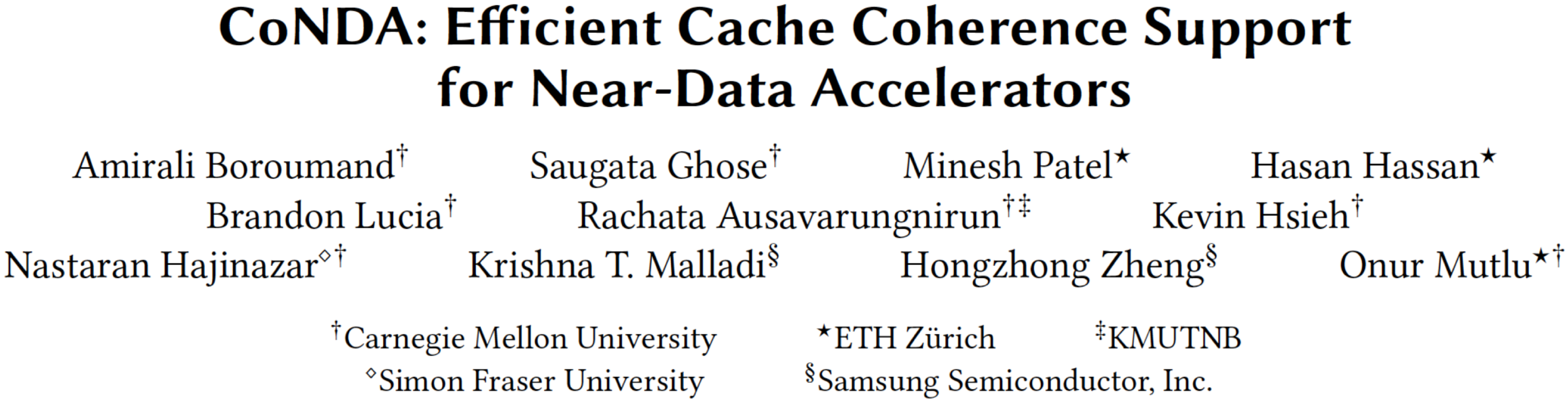


LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory

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*

How to Maintain Coherence in PIM?

- 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.

End of Backup Slides