Computer Architecture

Lecture 15: Emerging Memory Technologies

Dr. Haiyu Mao
Prof. Onur Mutlu
ETH Zürich

Fall 2023

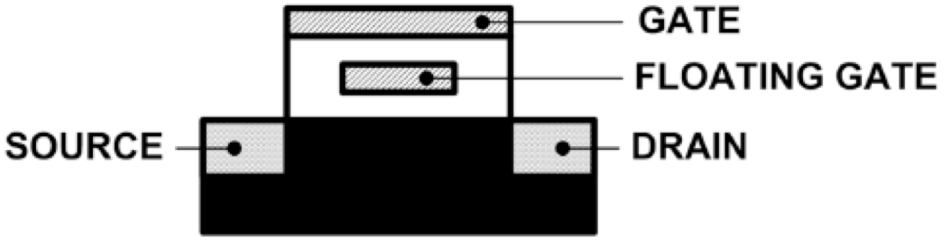
16 November 2023

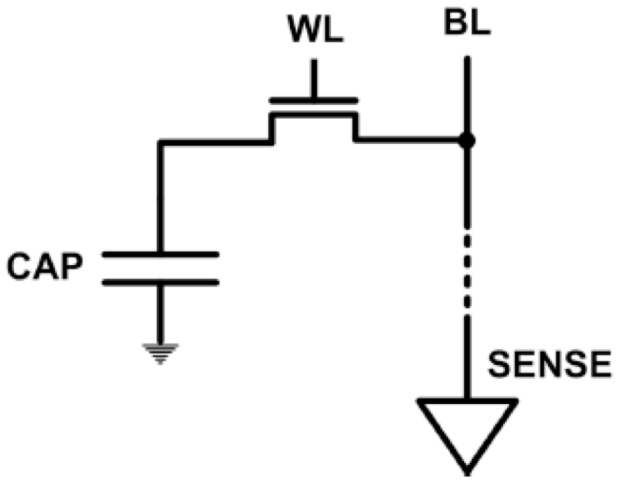
About Me

- Postdoctoral Researcher at SAFARI
  - Computer architecture
  - Algorithm-architecture co-design
  - Processing in memory (Non-volatile memory)
  - Processing in storage
  - Bioinformatics
  - Machine learning acceleration
- Group Associate at ETH Future Computing Laboratory
  - Cooperate with industry
- More info: https://hybol1993.github.io/

Limits of Charge Memory

- Difficult charge placement and control
  - Flash: floating gate charge
  - DRAM: capacitor charge, transistor leakage
- Reliable sensing becomes difficult as charge storage unit size reduces





Solution 1: New Memory Architectures

- Overcome memory shortcomings with
  - Memory-centric system design
  - Novel memory architectures, interfaces, functions
  - Better waste management (efficient utilization)

- Key issues to tackle
  - Enable reliability at low cost → high capacity
  - Reduce energy
  - Reduce latency
  - Improve bandwidth
  - Reduce waste (capacity, bandwidth, latency)
  - Enable computation close to data

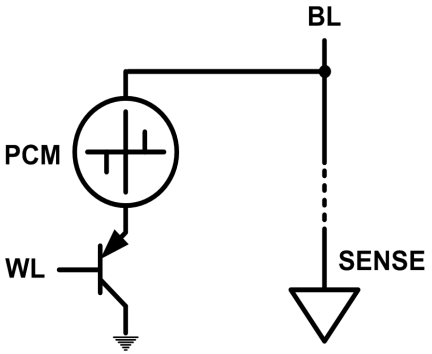
Solution 1: New Memory Architectures

- Liu+, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu+, "An Experimental Study of Data Retention Behavior in Modern DRAM Devices," ISCA 2013.
- Seshadri+, "RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data," MICRO 2013.
- Pekhimenko+, "Linearly Compressed Pages: A Main Memory Compression Framework," MICRO 2013.
- Chang+, "Improving DRAM Performance by Parallelizing Refreshes with Accesses," HPCA 2014.
- Khan+, "The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study," SIGMETRICS 2014.
- Luo+, "Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost," DSN 2014.
- Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.
- Lee+, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA 2015.
- Qureshi+, "AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems," DSN 2015.
- Meza+, "Revisiting Memory Errors in Large-Scale Production Data Centers: Analysis and Modeling of New Trends from the Field," DSN 2015.
- Kim+, "Ramulator: A Fast and Extensible DRAM Simulator," IEEE CAL 2015.
- Seshadri+, "Fast Bulk Bitwise AND and OR in DRAM," IEEE CAL 2015.
- Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing," ISCA 2015.
- Ahn+, "PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture," ISCA 2015.
- Lee+, "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," PACT 2015.
- Seshadri+, "Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses," MICRO 2015.
- Lee+, "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," TACO 2016.
- Hassan+, "ChargeCache: Reducing DRAM Latency by Exploiting Row Access Locality," HPCA 2016.
- Chang+, "Low-Cost Inter-Linked Subarrays (LISA): Enabling Fast Inter-Subarray Data Migration in DRAM," HPCA 2016.
- Chang+, "Understanding Latency Variation in Modern DRAM Chips Experimental Characterization, Analysis, and Optimization," SIGMETRICS 2016.
- Khan+, "PARBOR: An Efficient System-Level Technique to Detect Data Dependent Failures in DRAM," DSN 2016.
- Hsieh+, "Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems," ISCA 2016.
- Hashemi+, "Accelerating Dependent Cache Misses with an Enhanced Memory Controller," ISCA 2016.
- Boroumand+, "LazyPIM: An Efficient Cache Coherence Mechanism for Processing-in-Memory," IEEE CAL 2016.
- Pattnaik+, "Scheduling Techniques for GPU Architectures with Processing-In-Memory Capabilities," PACT 2016.
- Hsieh+, "Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation," ICCD 2016.
- Hashemi+, "Continuous Runahead: Transparent Hardware Acceleration for Memory Intensive Workloads," MICRO 2016.
- Khan+, "A Case for Memory Content-Based Detection and Mitigation of Data-Dependent Failures in DRAM"," IEEE CAL 2016.
- Hassan+, "SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies," HPCA 2017.
- Mutlu, "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser," DATE 2017.
- Lee+, "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms," SIGMETRICS 2017.
- Chang+, "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms," SIGMETRICS 2017.
- Patel+, "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," ISCA 2017.
- Seshadri and Mutlu, "Simple Operations in Memory to Reduce Data Movement," ADCOM 2017.
- Liu+, "Concurrent Data Structures for Near-Memory Computing," SPAA 2017.
- Khan+, "Detecting and Mitigating Data-Dependent DRAM Failures by Exploiting Current Memory Content," MICRO 2017.
- Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," MICRO 2017.
- Kim+, "GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies," BMC Genomics 2018.
- Kim+, "The DRAM Latency PUF: Quickly Evaluating Physical Unclonable Functions by Exploiting the Latency-Reliability Tradeoff in Modern DRAM Devices," HPCA 2018.
- Boroumand+, "Google Workloads for Consumer Devices: Mitigating Data Movement Bottlenecks," ASPLOS 2018.
- Das+, "VRL-DRAM: Improving DRAM Performance via Variable Refresh Latency," DAC 2018.
- Ghose+, "What Your DRAM Power Models Are Not Telling You: Lessons from a Detailed Experimental Study," SIGMETRICS 2018.
- Kim+, "Solar-DRAM: Reducing DRAM Access Latency by Exploiting the Variation in Local Bitlines," ICCD 2018.
- Wang+, "Reducing DRAM Latency via Charge-Level-Aware Look-Ahead Partial Restoration," MICRO 2018.
- Kim+, "D-RaNGe: Using Commodity DRAM Devices to Generate True Random Numbers with Low Latency and High Throughput," HPCA 2019.
- Singh+, "NAPEL: Near-Memory Computing Application Performance Prediction via Ensemble Learning," DAC 2019.
- Ghose+, "Demystifying Workload–DRAM Interactions: An Experimental Study," SIGMETRICS 2019.
- Patel+, "Understanding and Modeling On-Die Error Correction in Modern DRAM: An Experimental Study Using Real Devices," DSN 2019.
- Boroumand+, "CoNDA: Efficient Cache Coherence Support for Near-Data Accelerators," ISCA 2019.
- Hassan+, "CROW: A Low-Cost Substrate for Improving DRAM Performance, Energy Efficiency, and Reliability," ISCA 2019.
- Mutlu and Kim, "RowHammer: A Retrospective," TCAD 2019.
- Mutlu+, "Processing Data Where It Makes Sense: Enabling In-Memory Computation," MICPRO 2019.
- Seshadri and Mutlu, "In-DRAM Bulk Bitwise Execution Engine," ADCOM 2020.
- Koppula+, "EDEN: Energy-Efficient, High-Performance Neural Network Inference Using Approximate DRAM," MICRO 2019.
- Rezaei+, "NoM: Network-on-Memory for Inter-Bank Data Transfer in Highly-Banked Memories," CAL 2020.
- Frigo+, "TRRespass: Exploiting the Many Sides of Target Row Refresh," S&P 2020.
- Cojocar+, "Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers," S&P 2020.
- Luo+, "CLR-DRAM: A Low-Cost DRAM Architecture Enabling Dynamic Capacity-Latency Trade-Off," ISCA 2020.
- Kim+, "Revisiting RowHammer: An Experimental Analysis of Modern Devices and Mitigation Techniques," ISCA 2020.
- Salami+, "An Experimental Study of Reduced-Voltage Operation in Modern FPGAs for Neural Network Acceleration," DSN 2020.
- Fernando+, "NATSA: A Near-Data Processing Accelerator for Time Series Analysis," ICCD 2020.
- Wang+, "FIGARO: Improving System Performance via Fine-Grained In-DRAM Data Relocation and Caching," MICRO 2020.
- Patel+, "Bit-Exact ECC Recovery (BEER): Determining DRAM On-Die ECC Functions by Exploiting DRAM Data Retention Characteristics," MICRO 2020.
- Jafri+, "Refresh Triggered Computation: Improving the Energy Efficiency of Convolutional Neural Network Accelerators," TC 2020.
- Larimi+, "Understanding Power Consumption and Reliability of High-Bandwidth Memory with Voltage Underscaling," DATE 2021.
- Yaglikci+, "BlockHammer: Preventing RowHammer at Low Cost by Blacklisting Rapidly-Accessed DRAM Rows," HPCA 2021.
- Giannoula+, "SynCron: Efficient Synchronization Support for Near-Data-Processing Architectures," HPCA 2021.
- Mutlu+, "A Modern Primer on Processing in Memory," Invited Book Chapter 2021.
- Hajinazar+, "SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM," ASPLOS 2021.
- Orosa+, "CODIC: A Low-Cost Substrate for Enabling Custom In-DRAM Functionalities and Optimizations," ISCA 2021.
- Olgun+, "QUAC-TRNG: High-Throughput True Random Number Generation Using Quadruple Row Activation in Commodity DRAM Chips," ISCA 2021.
- Singh+, "FPGA-based Near-Memory Acceleration of Modern Data-Intensive Applications," IEEE Micro 2021.
- Oliveira+, "DAMOV: A New Methodology and Benchmark Suite for Evaluating Data Movement Bottlenecks," IEEE Access 2021.
- Gomez-Luna+, "Benchmarking a New Paradigm: An Experimental Analysis of a Real Processing-in-Memory Architecture," Arxiv 2021.
- Boroumand+, "Google Neural Network Models for Edge Devices: Analyzing and Mitigating Machine Learning Inference Bottlenecks," PACT 2021.
- Avoid DRAM:
  - Seshadri+, "The Evicted-Address Filter: A Unified Mechanism to Address Both Cache Pollution and Thrashing," PACT 2012.
  - Pekhimenko+, "Base-Delta-Immediate Compression: Practical Data Compression for On-Chip Caches," PACT 2012.
  - Seshadri+, "The Dirty-Block Index," ISCA 2014.
  - Pekhimenko+, "Exploiting Compressed Block Size as an Indicator of Future Reuse," HPCA 2015.
  - Vijaykumar+, "A Case for Core-Assisted Bottleneck Acceleration in GPUs: Enabling Flexible Data Compression with Assist Warps," ISCA 2015.
  - Pekhimenko+, "Toggle-Aware Bandwidth Compression for GPUs," HPCA 2016.

Solution 2: Emerging Memory Technologies

- Some emerging **resistive** memory technologies seem more scalable than DRAM (and they are non-volatile)

- Example: Phase Change Memory
  - Data stored by changing phase of material
  - Data read by detecting material's resistance
  - Expected to scale to 9nm (2022 [ITRS 2009])
  - Prototyped at 20nm (Raoux+, IBM JRD 2008)
  - Expected to be denser than DRAM: can store multiple bits/cell



- But, emerging technologies have (many) shortcomings
  - Can they be enabled to replace/augment/surpass DRAM?

Solution 2: Emerging Memory Technologies

- Lee+, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA'09, CACM'10, IEEE Micro'10.
- Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters 2012.
- Yoon, Meza+, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012.
- Kultursay+, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.
- Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.
- Lu+, "Loose Ordering Consistency for Persistent Memory," ICCD 2014.
- Zhao+, "FIRM: Fair and High-Performance Memory Control for Persistent Memory Systems," MICRO 2014.
- Yoon, Meza+, "Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories," TACO 2014.
- Ren+, "ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems," MICRO 2015.
- Chauhan+, "NVMove: Helping Programmers Move to Byte-Based Persistence," INFLOW 2016.
- Li+, "Utility-Based Hybrid Memory Management," CLUSTER 2017.
- Yu+, "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation," MICRO 2017.
- Tavakkol+, "MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices," FAST 2018.
- Tavakkol+, "FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives," ISCA 2018.
- Sadrosadati+. "LTRF: Enabling High-Capacity Register Files for GPUs via Hardware/Software Cooperative Register Prefetching," ASPLOS 2018.
- Salkhordeh+, "An Analytical Model for Performance and Lifetime Estimation of Hybrid DRAM-NVM Main Memories," TC 2019.
- Wang+, "Panthera: Holistic Memory Management for Big Data Processing over Hybrid Memories," PLDI 2019.
- Song+, "Enabling and Exploiting Partition-Level Parallelism (PALP) in Phase Change Memories," CASES 2019.
- Liu+, "Binary Star: Coordinated Reliability in Heterogeneous Memory Systems for High Performance and Scalability," MICRO'19.
- Song+, "Improving Phase Change Memory Performance with Data Content Aware Access," ISMM 2020.
- Yavits+, "WoLFRaM: Enhancing Wear-Leveling and Fault Tolerance in Resistive Memories using Programmable Address Decoders," ICCD 2020.
- Song+, "Aging-Aware Request Scheduling for Non-Volatile Main Memory," ASP-DAC 2021.

Intel Optane Persistent Memory (2019)

- Non-volatile main memory
- Based on 3D-XPoint Technology

https://www.storagereview.com/intel_optane_dc_persistent_memory_module_pmm

PCM as Main Memory: Idea in 2009

- Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger,
  "Architecting Phase Change Memory as a Scalable DRAM Alternative"
  *Proceedings of the 36th International Symposium on Computer Architecture (**ISCA**)*, pages 2-13, Austin, TX, June 2009. Slides (pdf)
  One of the 13 computer architecture papers of 2009 selected as Top Picks by IEEE Micro.
  Selected as a CACM Research Highlight.

Architecting Phase Change Memory as a Scalable DRAM Alternative

Benjamin C. Lee† Engin Ipek† Onur Mutlu‡ Doug Burger†

†Computer Architecture Group
Microsoft Research
Redmond, WA
{blee, ipek, dburger}@microsoft.com

‡Computer Architecture Laboratory
Carnegie Mellon University
Pittsburgh, PA
onur@cmu.edu

PCM as Main Memory: Idea in 2009

- Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger, **"Phase Change Technology and the Future of Main Memory"** *IEEE Micro, Special Issue: Micro's Top Picks from 2009 Computer Architecture Conferences (**MICRO TOP PICKS**)*, Vol. 30, No. 1, pages 60-70, January/February 2010.

Charge vs. Resistive Memories

- Charge Memory (e.g., DRAM, Flash)
  - Write data by capturing charge Q
  - Read data by detecting voltage V

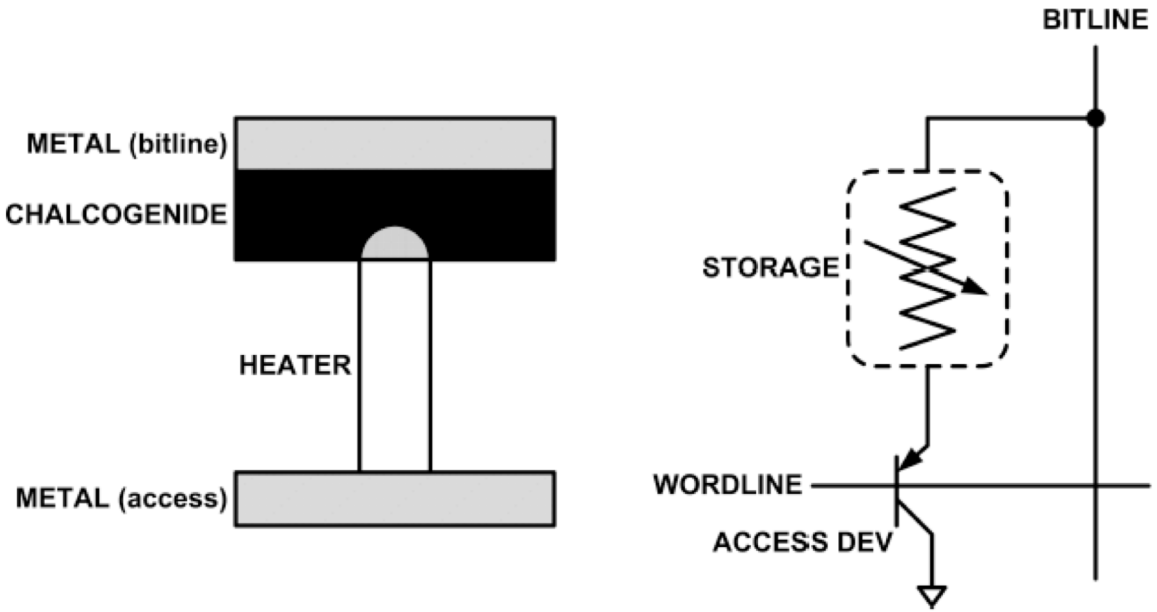
- Resistive Memory (e.g., PCM, STT-MRAM, memristors)
  - Write data by pulsing current dQ/dt
  - Read data by detecting resistance R

Promising Resistive Memory Technologies

- PCM
  - Inject current to change material phase
  - Resistance determined by phase
- STT-MRAM
  - Inject current to change magnet polarity
  - Resistance determined by polarity
- Memristors/RRAM/ReRAM
  - Inject current to change atomic structure
  - Resistance determined by atom distance

What is Phase Change Memory?

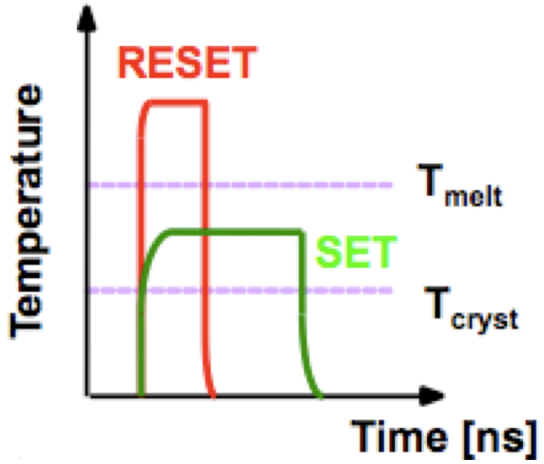
- Phase change material (chalcogenide glass) exists in two states:
  - Amorphous: Low optical reflexivity and high electrical resistivity
  - Crystalline: High optical reflexivity and low electrical resistivity

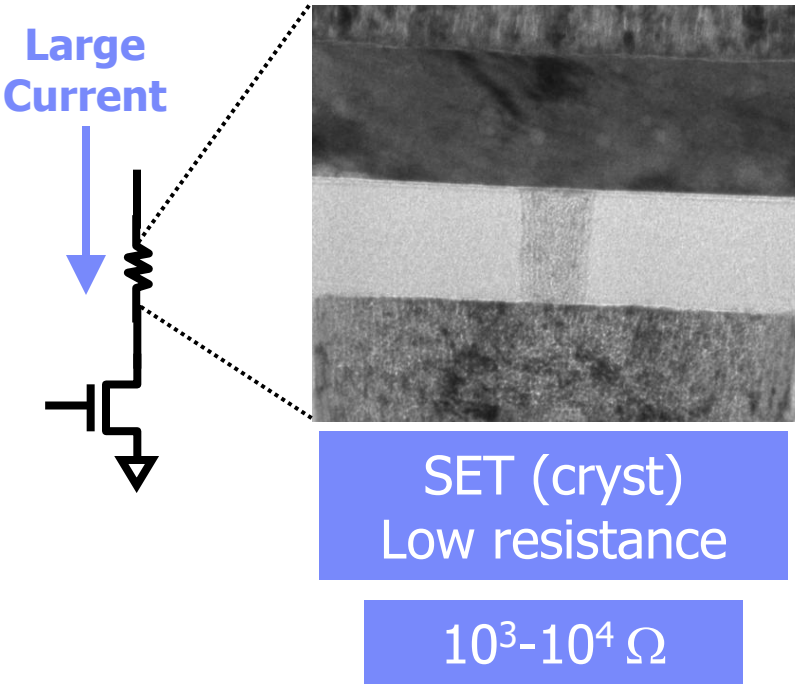


PCM is resistive memory: High resistance (0), Low resistance (1)
PCM cell can be switched between states reliably and quickly

How Does PCM Work?

- Write: change phase via current injection
  - SET: sustained current to heat cell above T*cryst*
  - RESET: cell heated above T*melt* and quenched
- Read: detect phase via material resistance
  - amorphous/crystalline





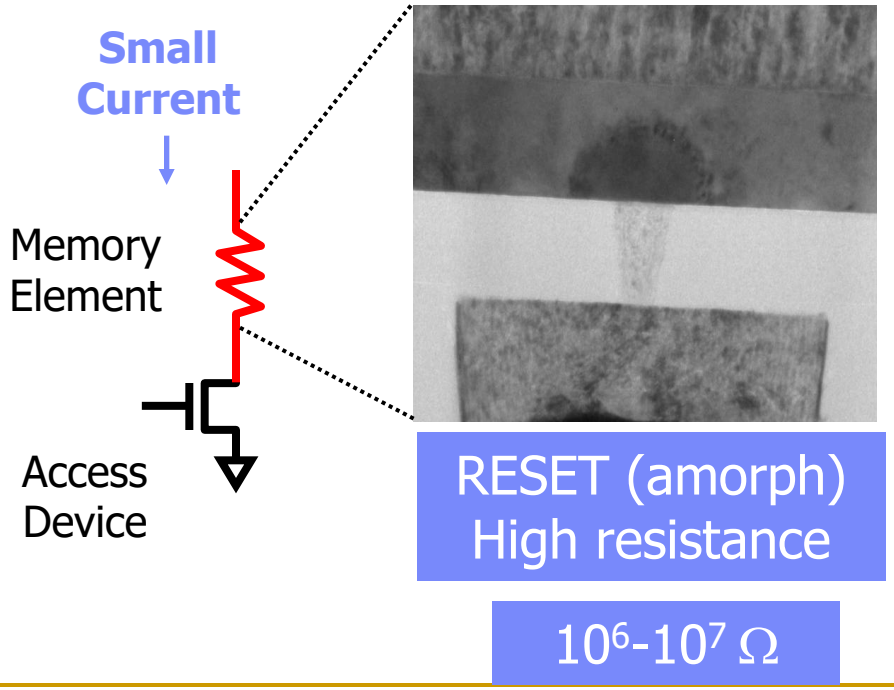
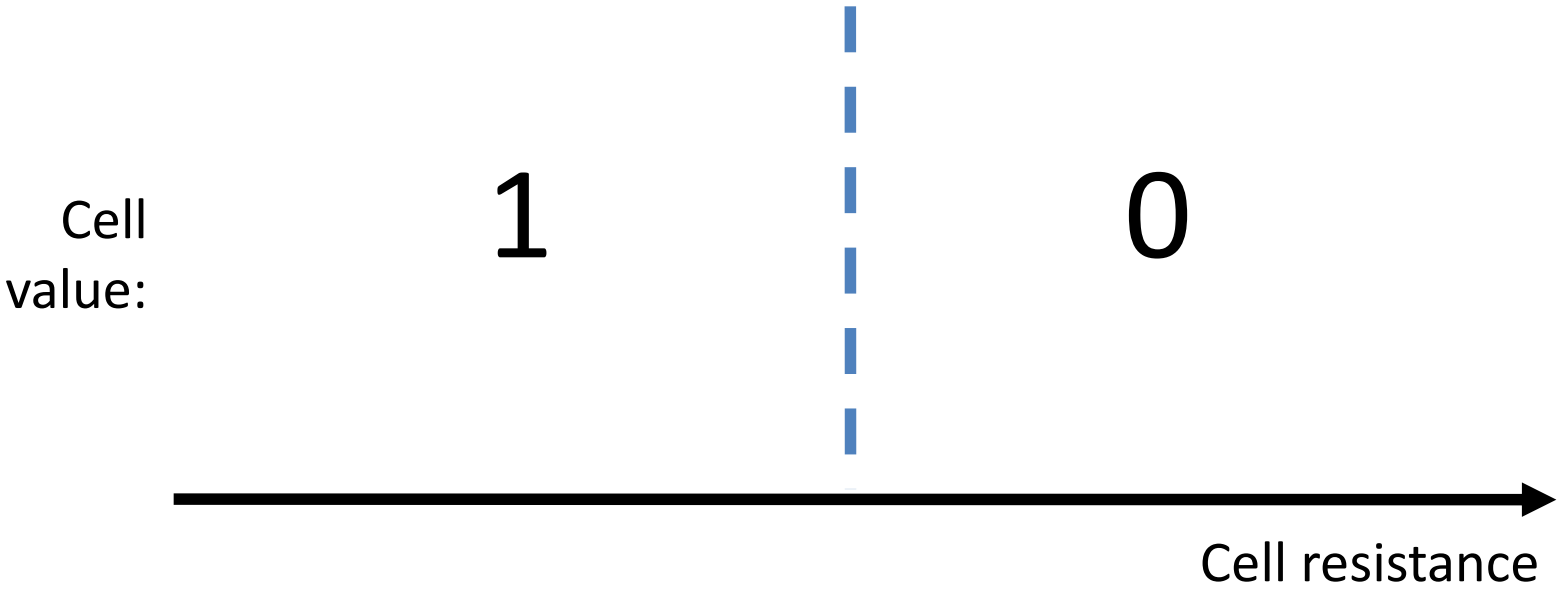


Photo Courtesy: Bipin Rajendran, IBM Slide Courtesy: Moinuddin Qureshi, IBM

Opportunity: PCM Advantages

- Scales better than DRAM, Flash
  - Requires current pulses, which scale linearly with feature size
  - Expected to scale to 9nm (2022 [ITRS])
  - Prototyped at 20nm (Raoux+, IBM JRD 2008)
- Can be denser than DRAM
  - Can store multiple bits per cell due to large resistance range
  - Prototypes with 2 bits/cell in ISSCC' 08, 4 bits/cell by 2012
- Non-volatile
  - Retain data for >10 years at 85C
- No refresh needed, low idle power

PCM Resistance → Value

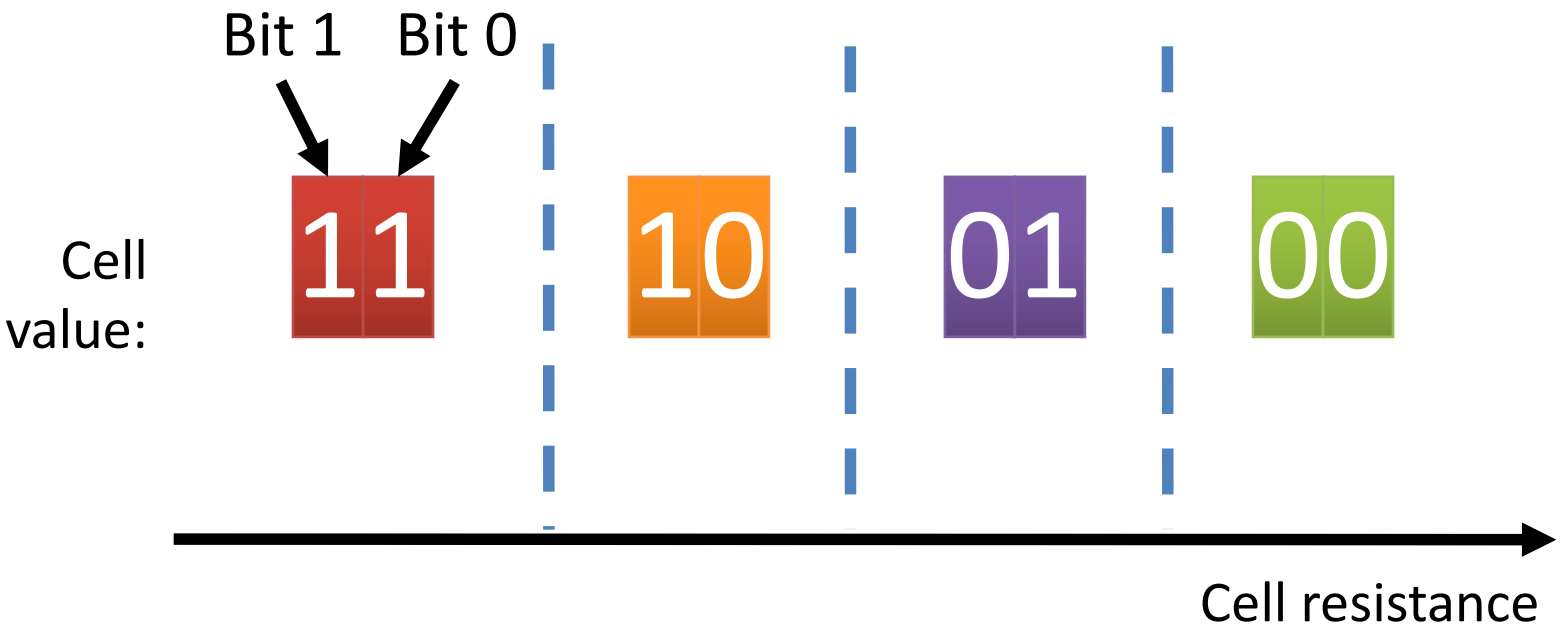


Multi-Level Cell PCM

- Multi-level cell: more than 1 bit per cell
  - Further increases density by 2 to 4x [Lee+,ISCA'09]

- But MLC-PCM also has drawbacks
  - Higher latency and energy than single-level cell PCM

MLC-PCM Resistance → Value

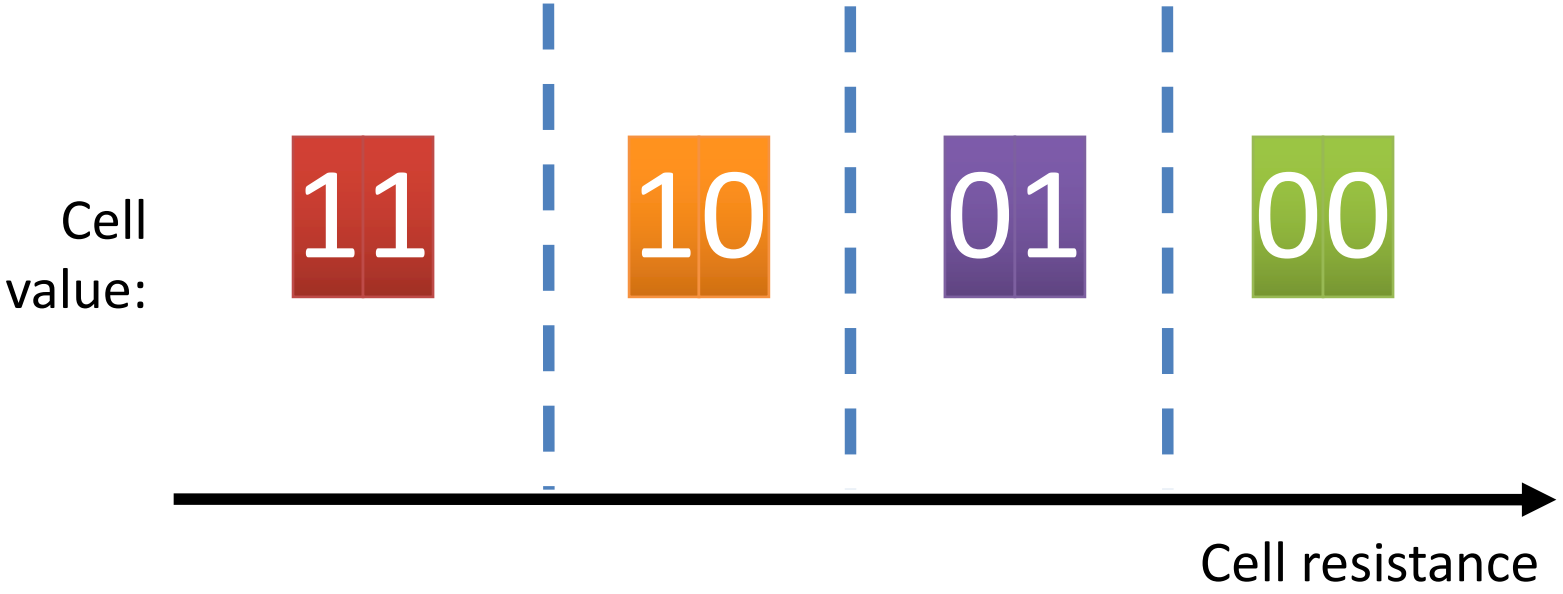


MLC-PCM Resistance → Value

Less margin between values

→ need more precise sensing/modification of cell contents

→ higher latency/energy (~2x for reads and 4x for writes)



Phase Change Memory Properties

- Surveyed prototypes from 2003-2008 (ITRS, IEDM, VLSI, ISSCC)
- Derived PCM parameters for F=90nm

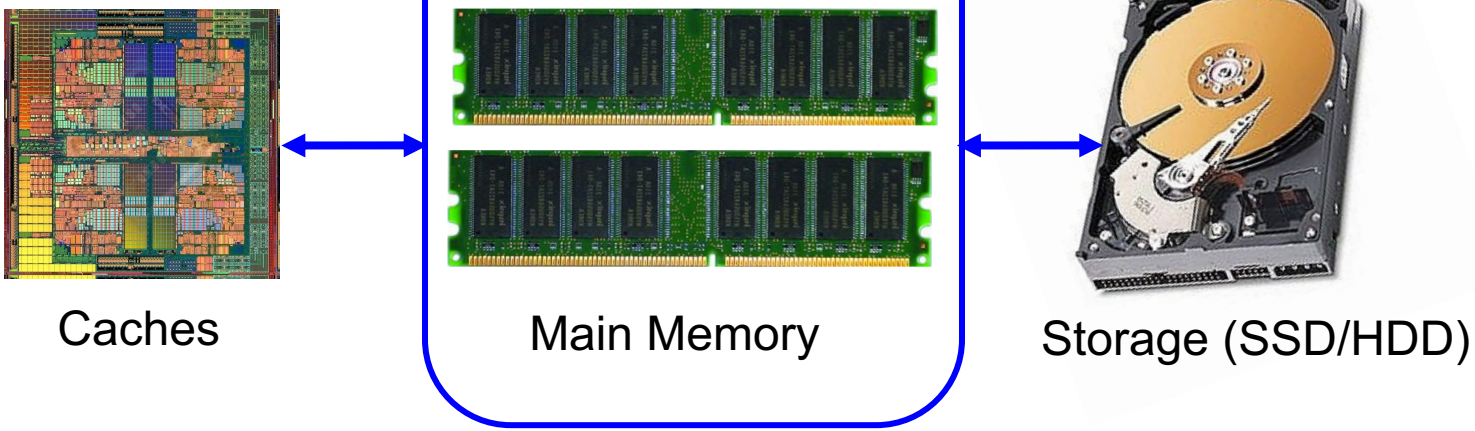
- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.
- Lee et al., "Phase Change Technology and the Future of Main Memory," IEEE Micro Top Picks 2010.

Table 1. Technology survey.

Parameter*	Published prototype									
	Horri[6]	Ahn[12]	Bedeschi[13]	Oh[14]	Pellizer[15]	Chen[5]	Kang[16]	Bedeschi[9]	Lee[10]	Lee[2]
Year	2003	2004	2004	2005	2006	2006	2006	2008	2008	**
Process, F (nm)	**	120	180	120	90	**	100	90	90	90
Array size (Mbytes)	**	64	8	64	**	**	256	256	512	**
Material	GST, N-d	GST, N-d	GST	GST	GST	GS, N-d	GST	GST	GST	GST, N-d
Cell size (μm^2)	**	0.290	0.290	**	0.097	60 nm^2	0.166	0.097	0.047	0.065 to 0.097
Cell size, F^2	**	20.1	9.0	**	12.0	**	16.6	12.0	5.8	9.0 to 12.0
Access device	**	**	BJT	FET	BJT	**	FET	BJT	Diode	BJT
Read time (ns)	**	70	48	68	**	**	62	**	55	48
Read current (μA)	**	**	40	**	**	**	**	**	**	40
Read voltage (V)	**	3.0	1.0	1.8	1.6	**	1.8	**	1.8	1.0
Read power (μW)	**	**	40	**	**	**	**	**	**	40
Read energy (pJ)	**	**	2.0	**	**	**	**	**	**	2.0
Set time (ns)	100	150	150	180	**	80	300	**	400	150
Set current (μA)	200	**	300	200	**	55	**	**	**	150
Set voltage (V)	**	**	2.0	**	**	1.25	**	**	**	1.2
Set power (μW)	**	**	300	**	**	34.4	**	**	**	90
Set energy (pJ)	**	**	45	**	**	2.8	**	**	**	13.5
Reset time (ns)	50	10	40	10	**	60	50	**	50	40
Reset current (μA)	600	600	600	600	400	90	600	300	600	300
Reset voltage (V)	**	**	2.7	**	1.8	1.6	**	1.6	**	1.6
Reset power (μW)	**	**	1620	**	**	80.4	**	**	**	480
Reset energy (pJ)	**	**	64.8	**	**	4.8	**	**	**	19.2
Write endurance (MLC)	10^7	10^9	10^6	**	10^8	10^4	**	10^5	10^5	10^8

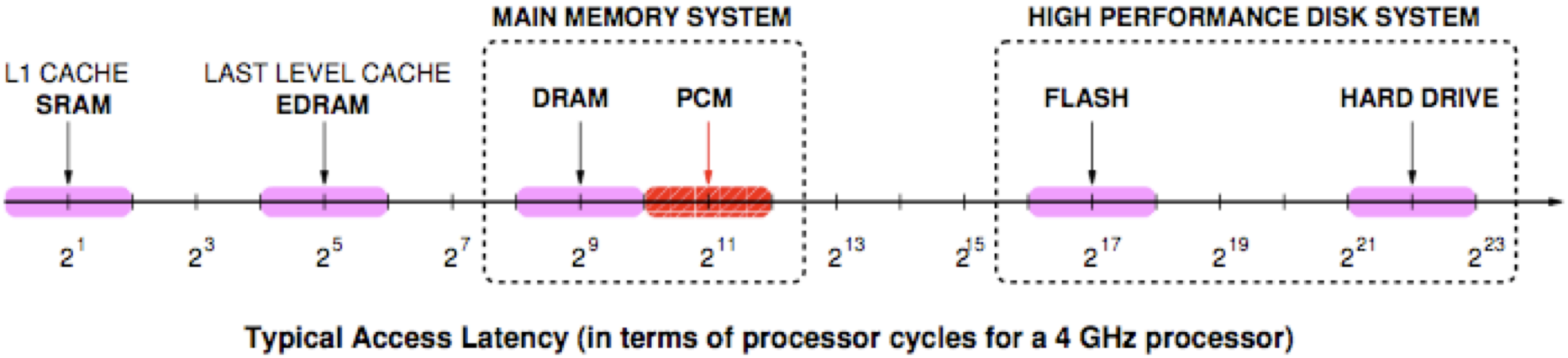
* BJT: bipolar junction transistor; FET: field-effect transistor; GST: $Ge_2Sb_2Te_5$; MLC: multilevel cells; N-d: nitrogen doped.
** This information is not available in the publication cited.

Where Can PCM Fit in the System?



Phase Change Memory Properties: Latency

- Latency comparable to, but slower than DRAM



Typical Access Latency (in terms of processor cycles for a 4 GHz processor)

- Read Latency
  - 50ns: 4x DRAM, 10^{-3}x NAND Flash
- Write Latency
  - 150ns: 12x DRAM
- Write Bandwidth
  - 5-10 MB/s: 0.1x DRAM, 1x NAND Flash

Qureshi+, "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Phase Change Memory Properties

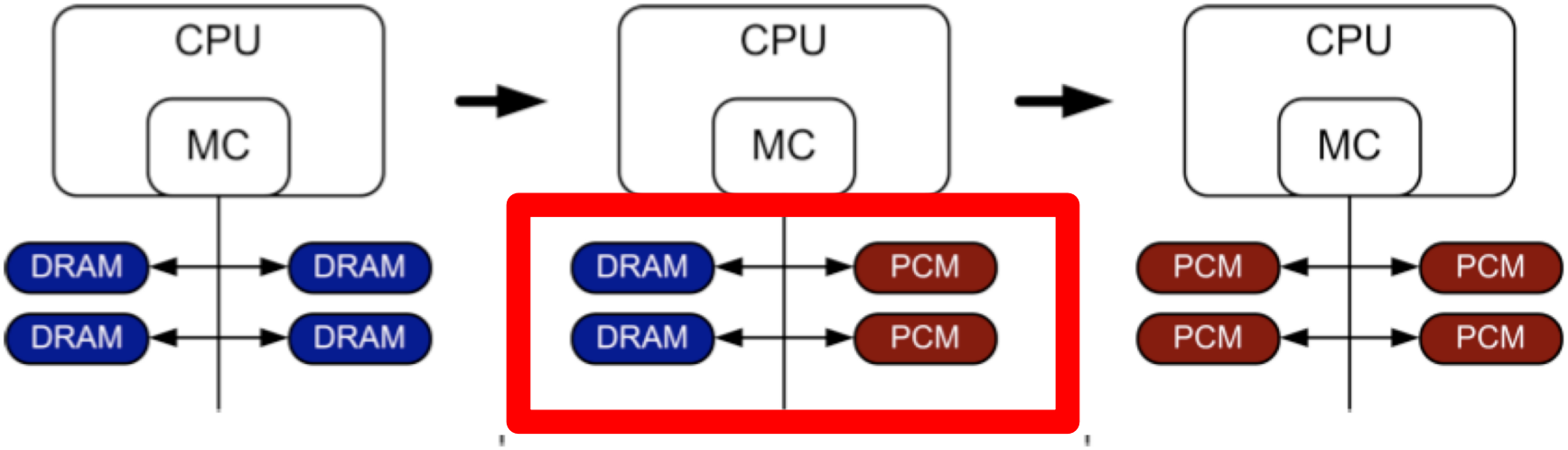
- Dynamic Energy
  - 40 uA Rd, 150 uA Wr
  - 2-43x DRAM, 1x NAND Flash
- Endurance
  - Writes induce phase change at 650C
  - Contacts degrade from thermal expansion/contraction
  - 10^8 writes per cell
  - 10^{-8}x DRAM, 10^3x NAND Flash
- Cell Size
  - 9-12F^2 using BJT, single-level cells
  - 1.5x DRAM, 2-3x NAND (will scale with feature size, MLC)

Phase Change Memory: Pros and Cons

- Pros over DRAM
  - Better technology scaling (capacity and cost)
  - Non volatile → Persistent
  - Low idle power (no refresh)
- Cons
  - Higher latencies: ~4-15x DRAM (especially write)
  - Higher active energy: ~2-50x DRAM (especially write)
  - Lower endurance (a cell dies after ~10^8 writes)
  - Reliability issues (resistance drift)
- Challenges in enabling PCM as DRAM replacement/helper:
  - Mitigate PCM shortcomings
  - Find the right way to place PCM in the system

PCM-based Main Memory (I)

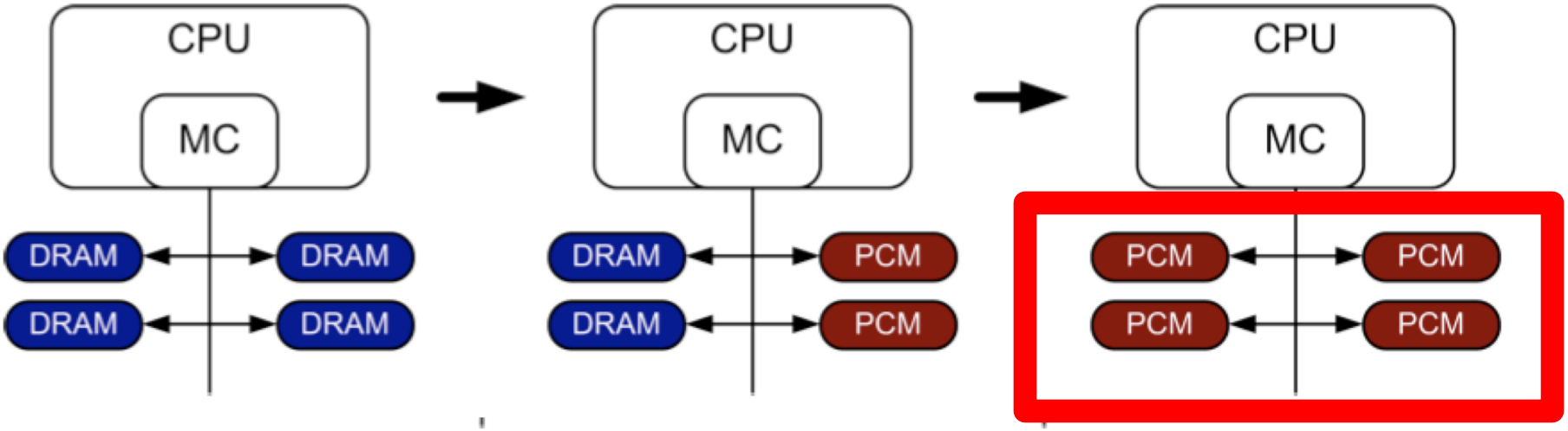
- How should PCM-based (main) memory be organized?



- Hybrid PCM+DRAM [Qureshi+ ISCA'09, Dhiman+ DAC'09]:
  - How to partition/migrate data between PCM and DRAM

PCM-based Main Memory (II)

- How should PCM-based (main) memory be organized?



- Pure PCM main memory [Lee et al., ISCA'09, Top Picks'10]:
  - How to redesign entire hierarchy (and cores) to overcome PCM shortcomings

An Initial Study: Replace DRAM with PCM

- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.
  - Surveyed prototypes from 2003-2008 (e.g. IEDM, VLSI, ISSCC)
  - Derived "average" PCM parameters for F=90nm

Density

- 9 - 12F^2 using BJT
- 1.5× DRAM

Latency

- 50ns Rd, 150ns Wr
- 4×, 12× DRAM

Endurance

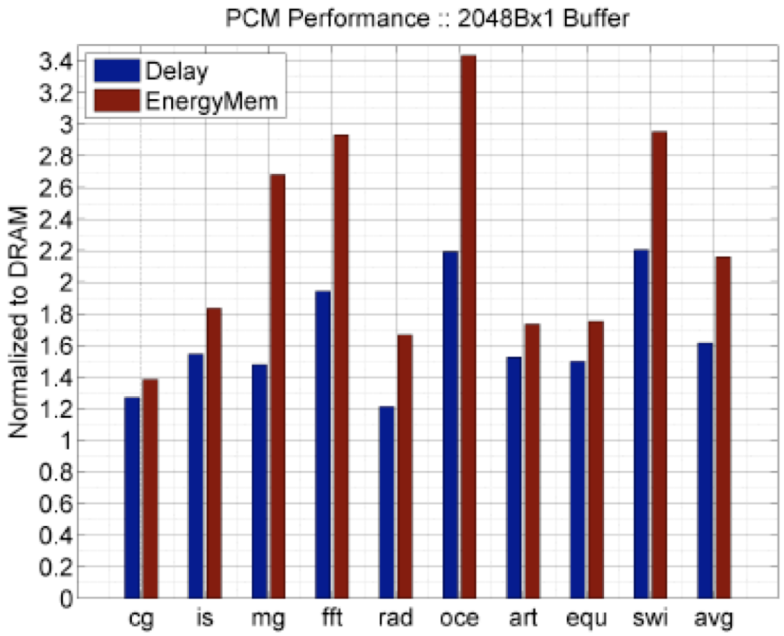
- 1E+08 writes
- 1E-08× DRAM

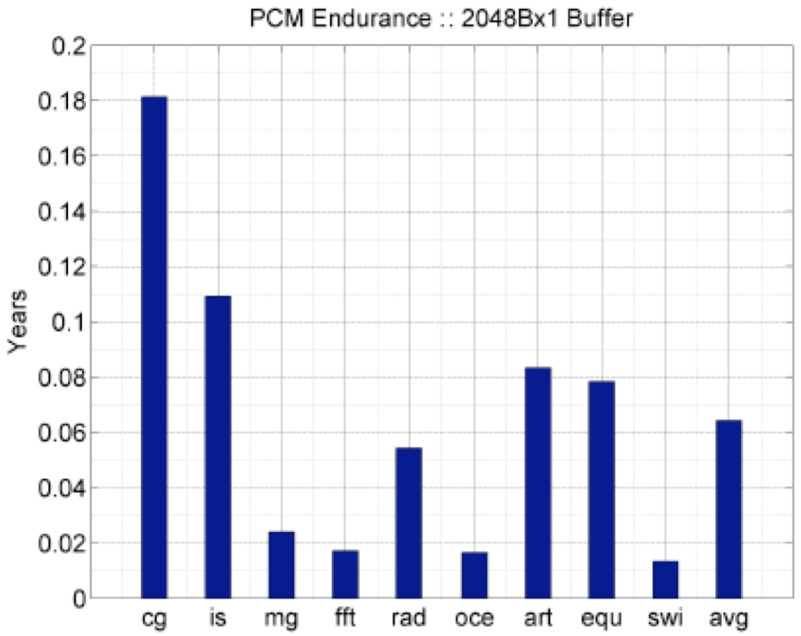
Energy

- 40μA Rd, 150μA Wr
- 2×, 43× DRAM

Results: Naïve Replacement of DRAM with PCM

- Replace DRAM with PCM in a 4-core, 4MB L2 system
- PCM organized the same as DRAM: row buffers, banks, peripherals
- 1.6x delay, 2.2x energy, 500-hour average lifetime



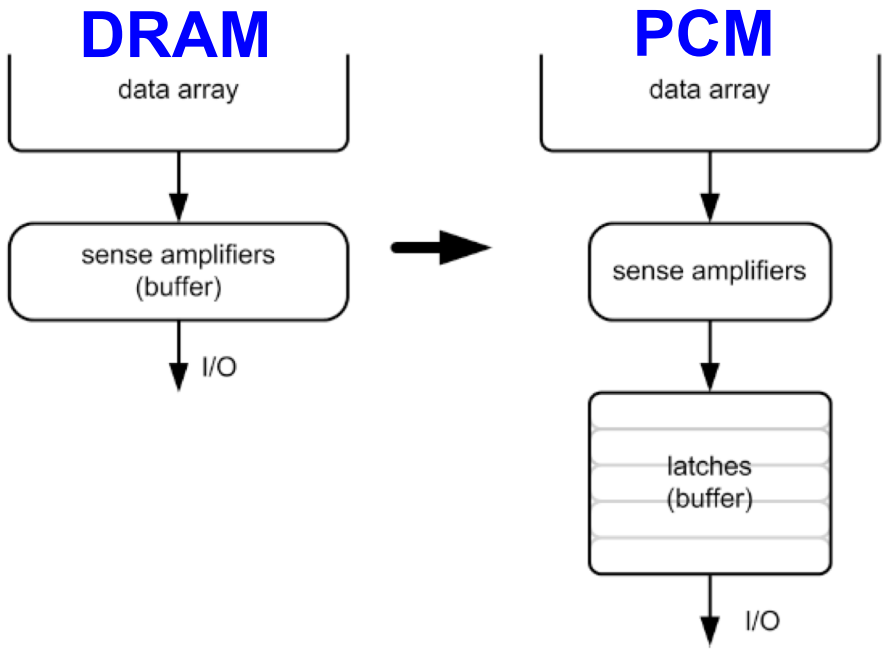


- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

Architecting PCM to Mitigate Shortcomings

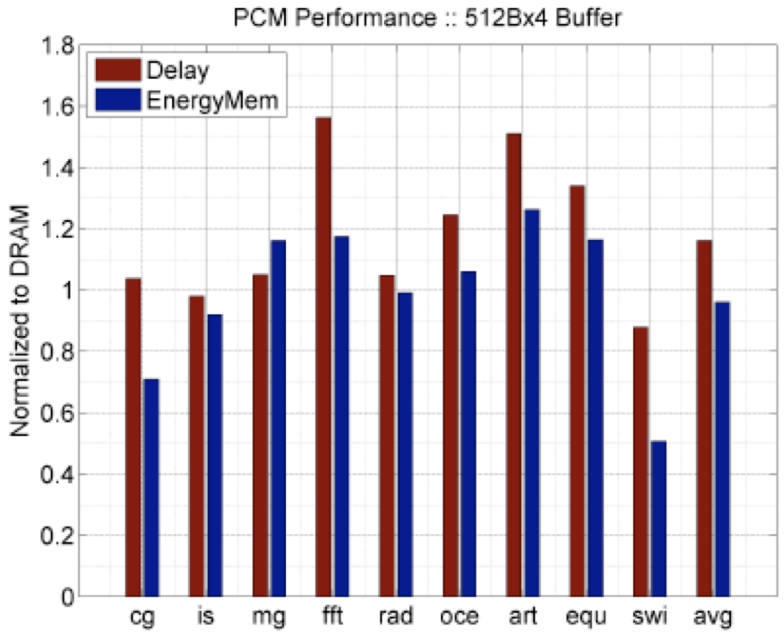
- Idea 1: Use multiple narrow row buffers in each PCM chip
  → Reduces array reads/writes → better endurance, latency, energy

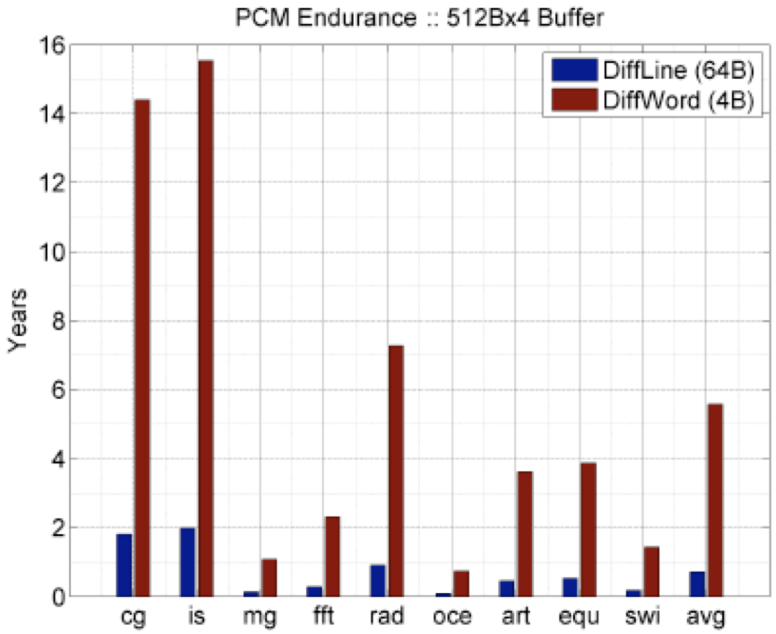
- Idea 2: Write into array at cache block or word granularity
  → Reduces unnecessary wear



Results: Architected PCM as Main Memory

- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density

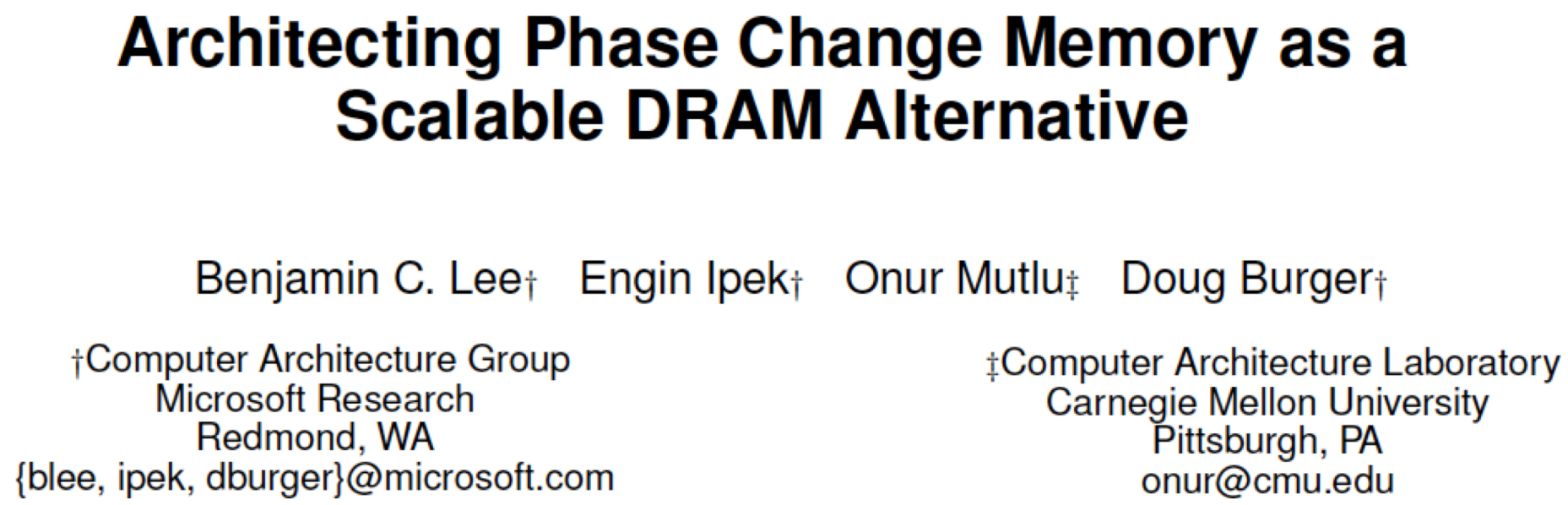




- Caveat 1: Worst-case lifetime is much shorter (no guarantees)
- Caveat 2: Intensive applications see large performance and energy hits
- Caveat 3: Optimistic PCM parameters?

PCM As Main Memory

- Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger,
  "Architecting Phase Change Memory as a Scalable DRAM Alternative"
  *Proceedings of the 36th International Symposium on Computer Architecture (**ISCA**)*, pages 2-13, Austin, TX, June 2009. Slides (pdf)
  One of the 13 computer architecture papers of 2009 selected as Top Picks by IEEE Micro.
  Selected as a CACM Research Highlight.



Architecting Phase Change Memory as a Scalable DRAM Alternative

Benjamin C. Lee† Engin Ipek† Onur Mutlu‡ Doug Burger†

†Computer Architecture Group
Microsoft Research
Redmond, WA
{blee, ipek, dburger}@microsoft.com

‡Computer Architecture Laboratory
Carnegie Mellon University
Pittsburgh, PA
onur@cmu.edu

More on PCM As Main Memory (II)

- Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger, **"Phase Change Technology and the Future of Main Memory"** *IEEE Micro, Special Issue: Micro's Top Picks from 2009 Computer Architecture Conferences (**MICRO TOP PICKS**)*, Vol. 30, No. 1, pages 60-70, January/February 2010.

PHASE-CHANGE TECHNOLOGY AND THE FUTURE OF MAIN MEMORY

Intel Optane Memory (Idea Realized in 2019)

- Non-volatile main memory
- Based on 3D-XPoint Technology

https://www.storagereview.com/intel_optane_dc_persistent_memory_module_pmm

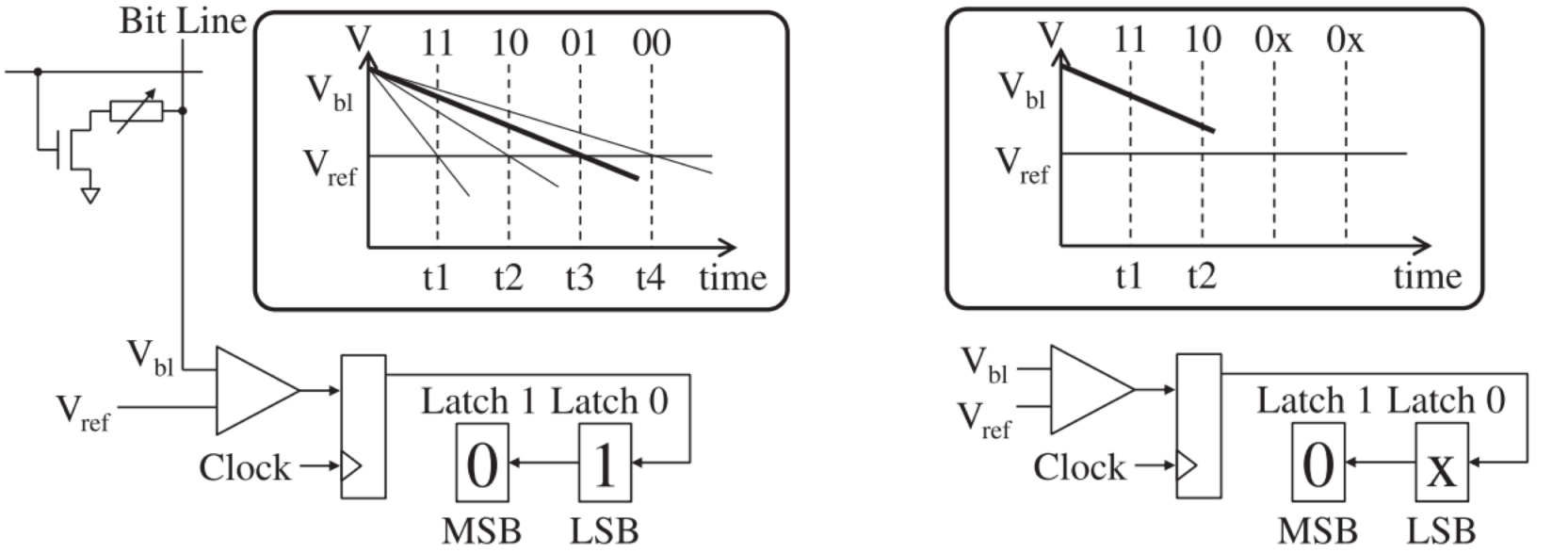
More on PCM Based Main Memory

HanBin Yoon, Justin Meza, Naveen Muralimanohar, Norman P. Jouppi, and Onur Mutlu, **"Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories"**
ACM Transactions on Architecture and Code Optimization (***TACO***), Vol. 11, No. 4, December 2014. [Slides (ppt) (pdf)]
Presented at the 10th HiPEAC Conference, Amsterdam, Netherlands, January 2015. [Slides (ppt) (pdf)]
Best (student) presentation award.

Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories

HANBIN YOON* and JUSTIN MEZA, Carnegie Mellon University
NAVEEN MURALIMANOHAR, Hewlett-Packard Labs
NORMAN P. JOUPPI**, Google Inc.
ONUR MUTLU, Carnegie Mellon University

Some PCM Bits Take Longer to Read…



(a) Sensing time is longer for higher cell resistances.

(b) One bit is determined before the other.

Fig. 3. MLC PCM cell read operation [Qureshi et al. 2010b].

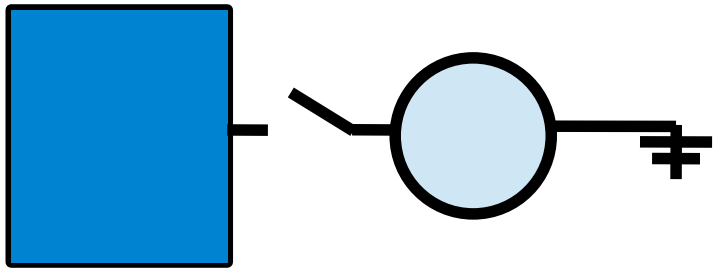
Observation 1: Read Asymmetry

- ***The read latency/energy of Bit 1 is lower than that of Bit 0***

- This is due to how MLC-PCM cells are read

Observation 1: Read Asymmetry

Simplified example

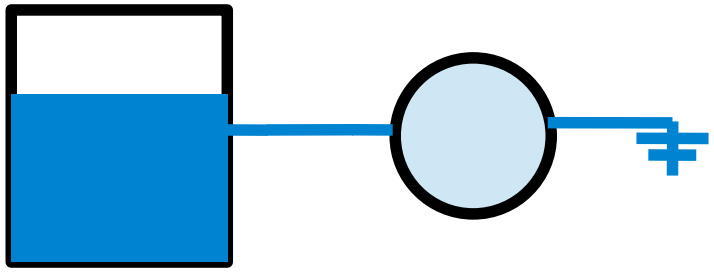


Capacitor filled with reference voltage

MLC-PCM cell with unknown resistance

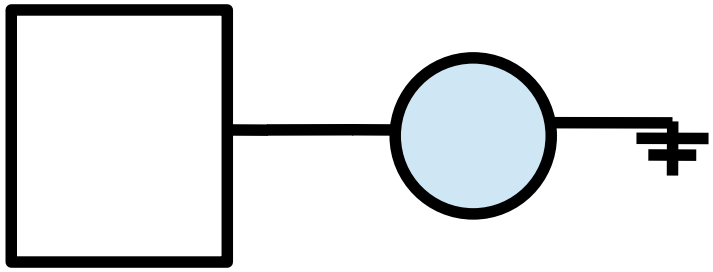
Observation 1: Read Asymmetry

Simplified example



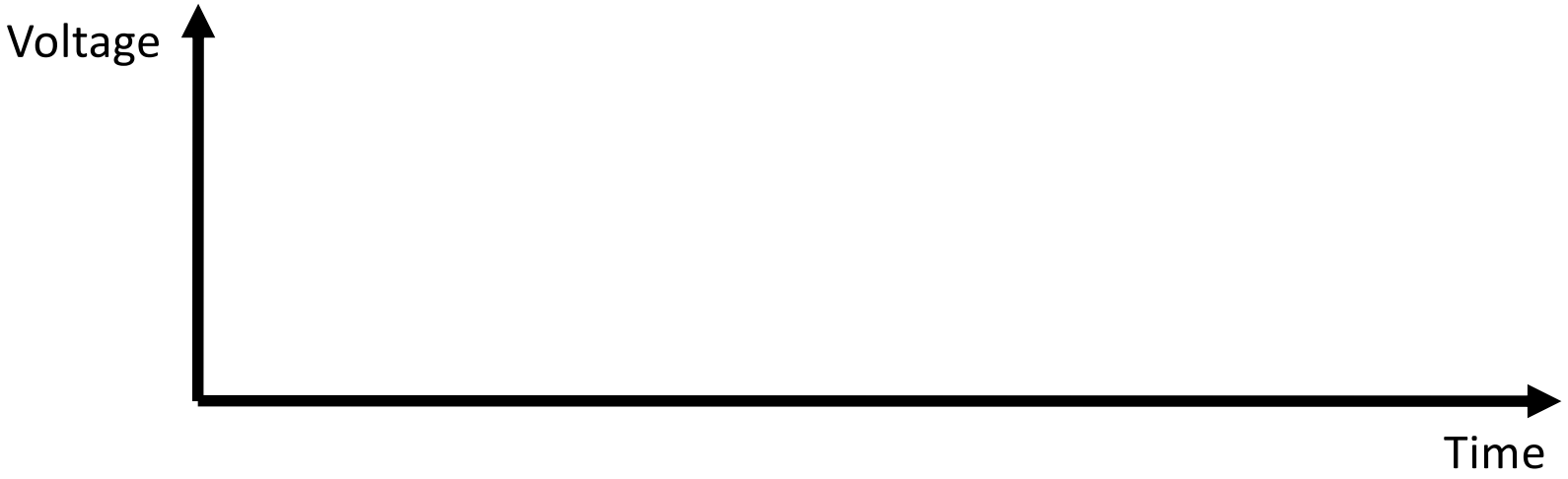
Observation 1: Read Asymmetry

Simplified example

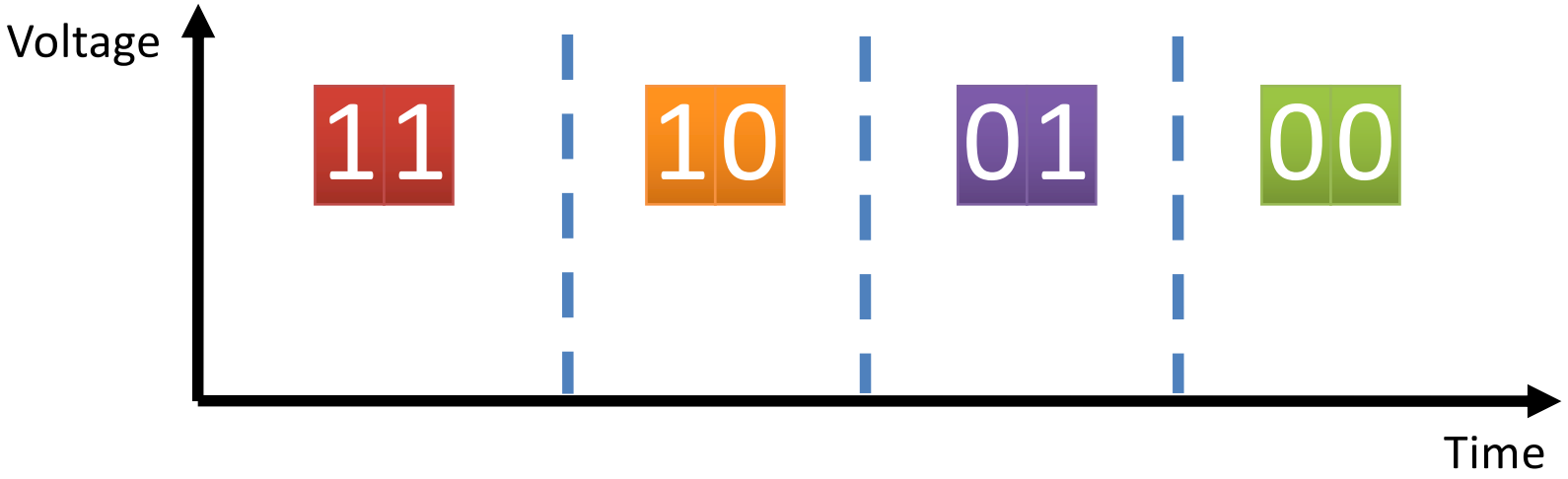


Infer data value

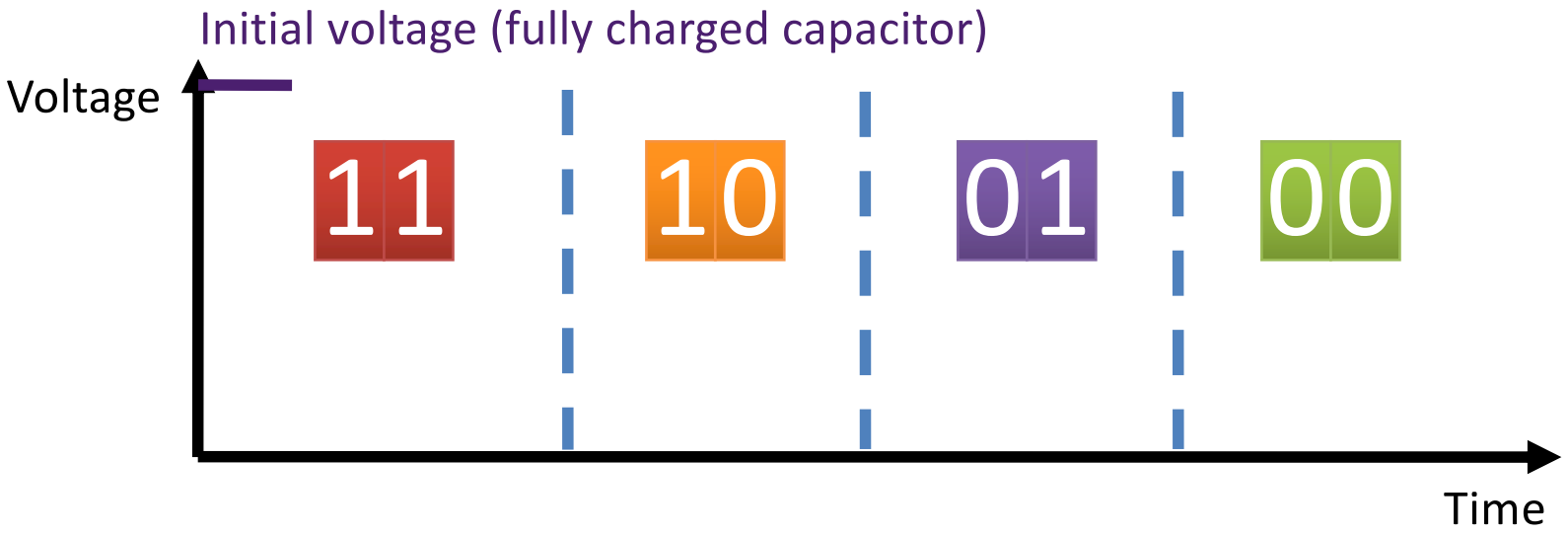
Observation 1: Read Asymmetry



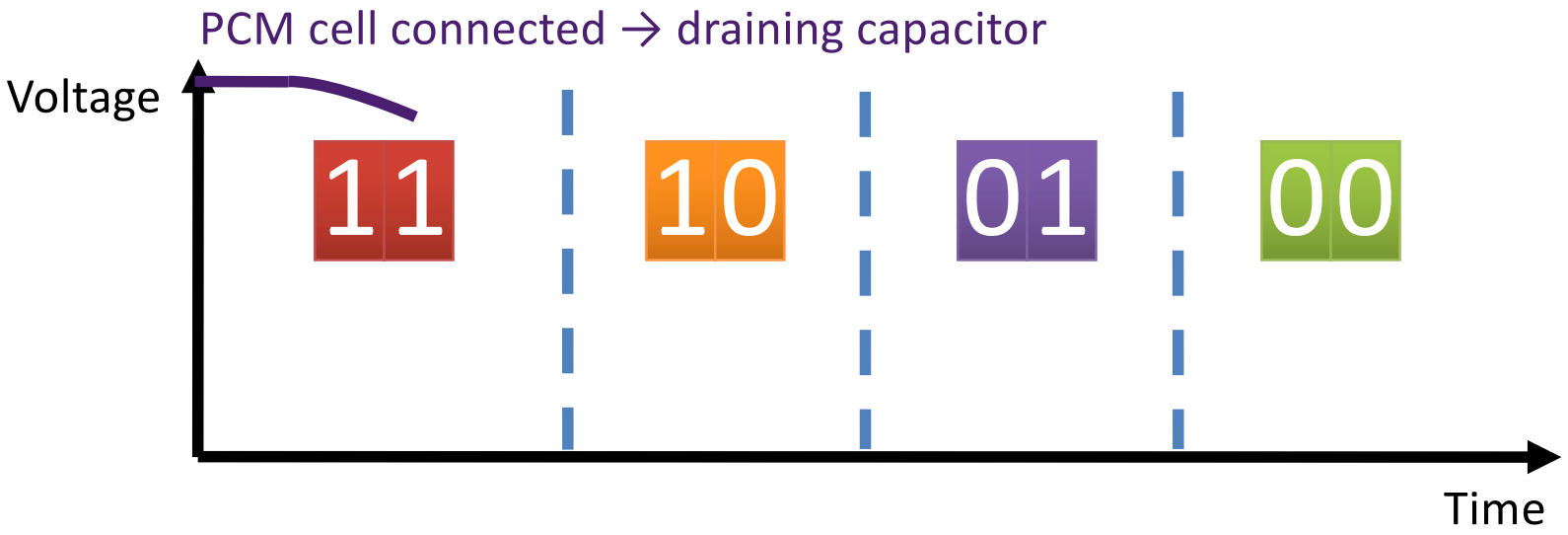
Observation 1: Read Asymmetry



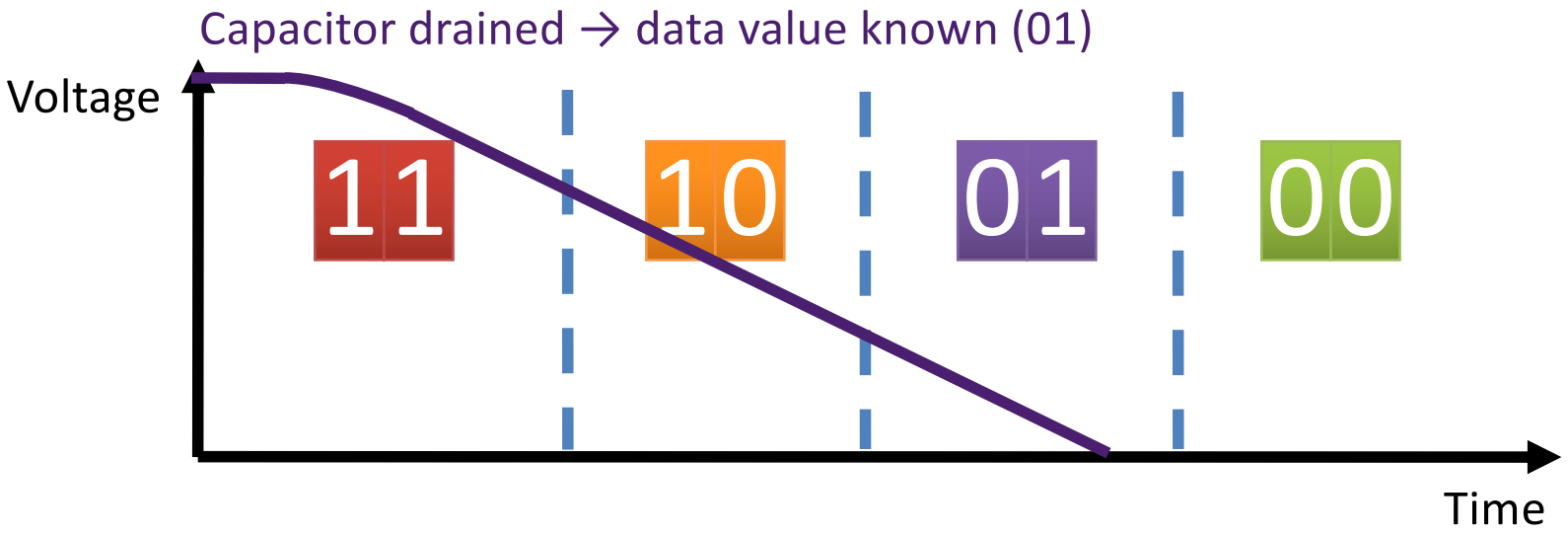
Observation 1: Read Asymmetry



Observation 1: Read Asymmetry



Observation 1: Read Asymmetry

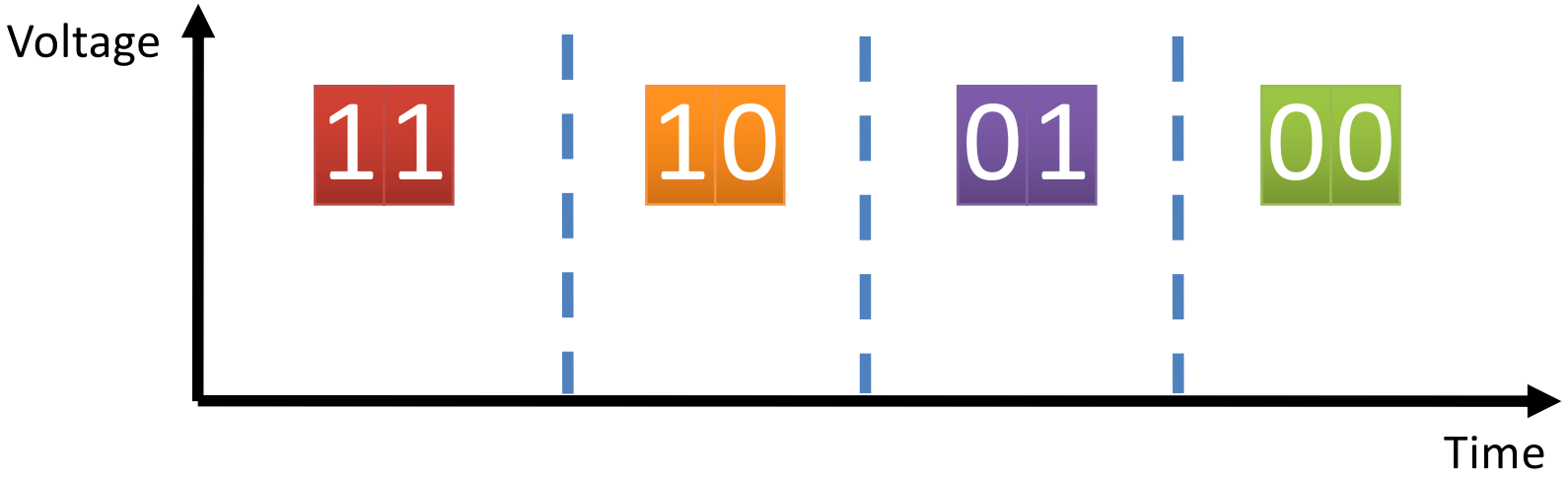


Observation 1: Read Asymmetry

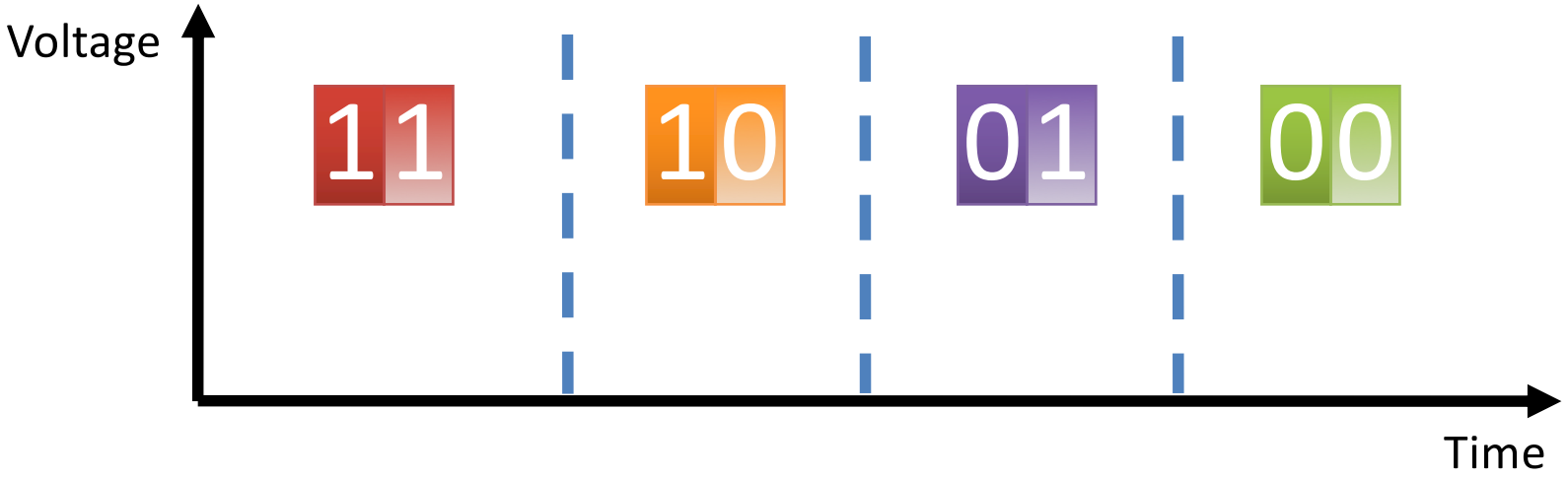
- In existing devices
  - Both MLC bits are read at the same time
  - Must wait ***maximum time*** to read both bits

- However, ***we can infer information about Bit 1 before this time***

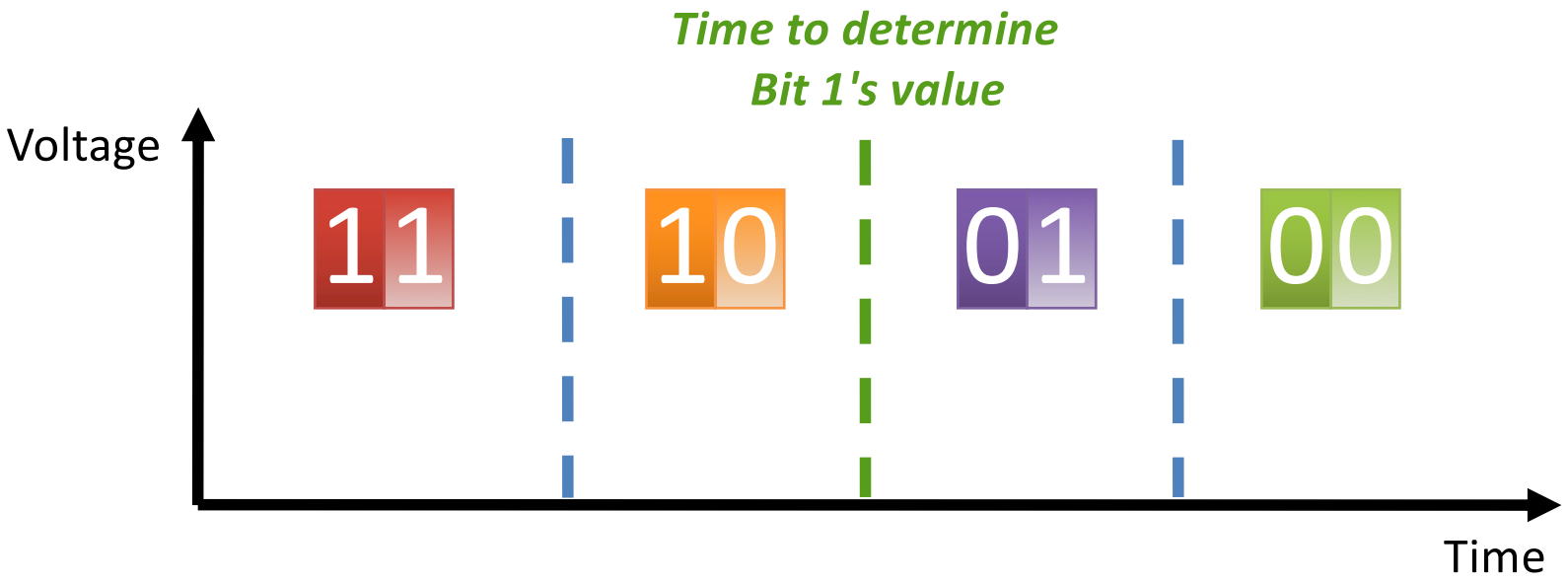
Observation 1: Read Asymmetry



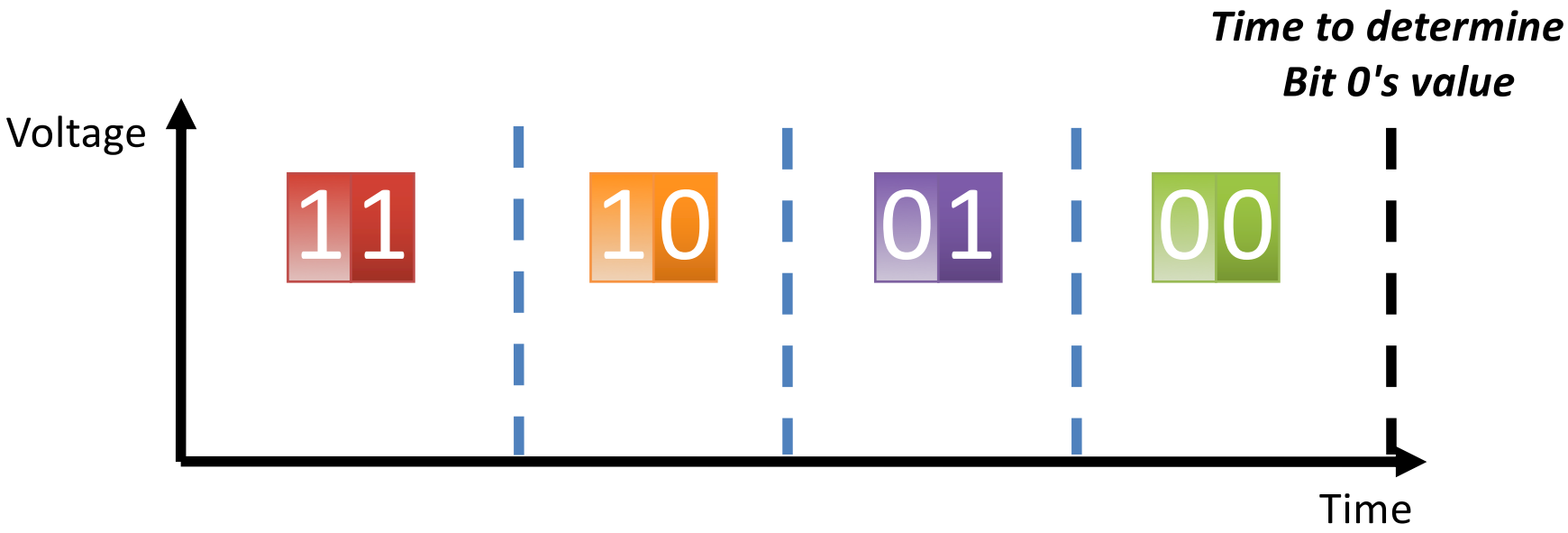
Observation 1: Read Asymmetry



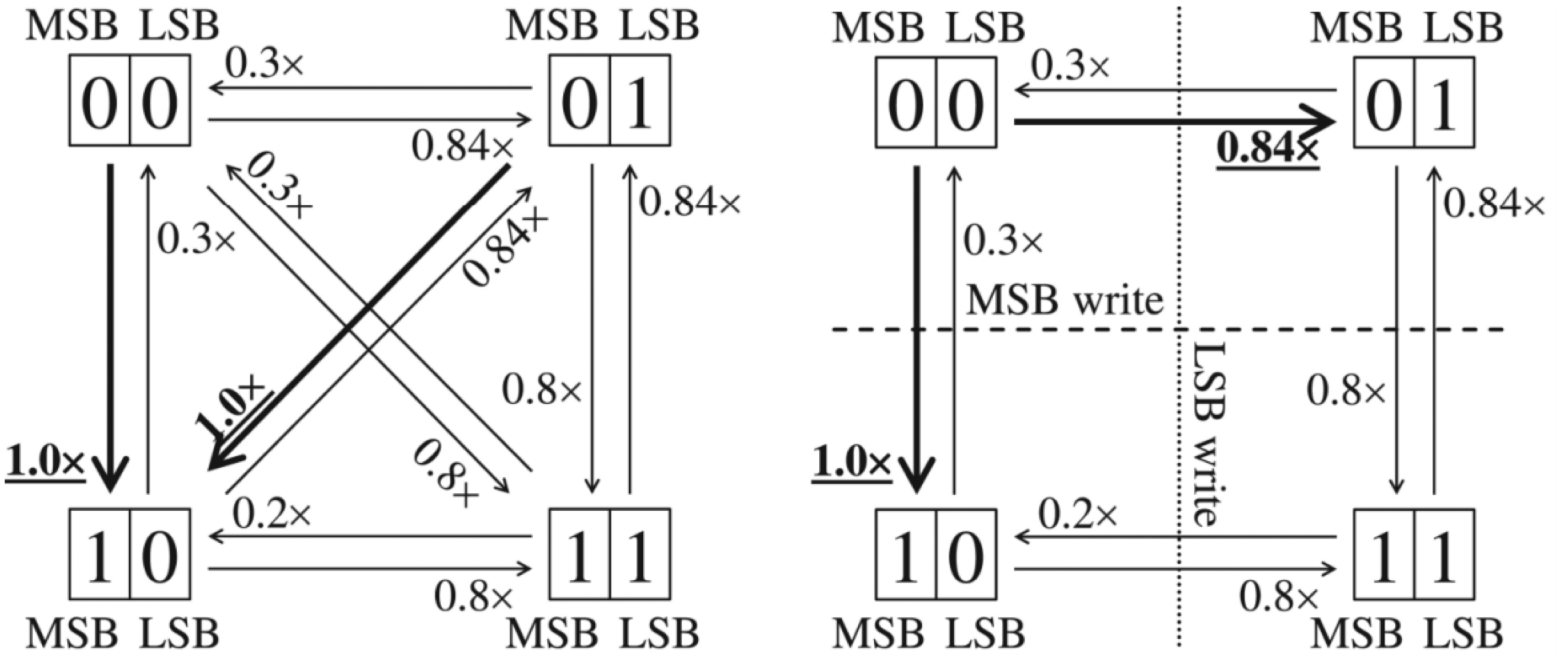
Observation 1: Read Asymmetry



Observation 1: Read Asymmetry



Some PCM Bits Take Longer to Write…



(a) All possible cell state transitions.

(b) Cell state transitions when modifying only the MSB *or* the LSB.

Fig. 4. MLC PCM cell write latencies [Joshi et al. 2011; Nirschl et al. 2007; Happ et al. 2006].

More on PCM Latencies and Exploiting Them

HanBin Yoon, Justin Meza, Naveen Muralimanohar, Norman P. Jouppi, and Onur Mutlu,
"Efficient Data Mapping and Buffering Techniques for Multi-Level Cell Phase-Change Memories"
ACM Transactions on Architecture and Code Optimization (***TACO***), Vol. 11, No. 4, December 2014. [Slides (ppt) (pdf)]
Presented at the 10th HiPEAC Conference, Amsterdam, Netherlands, January 2015. [Slides (ppt) (pdf)]
Best (student) presentation award.

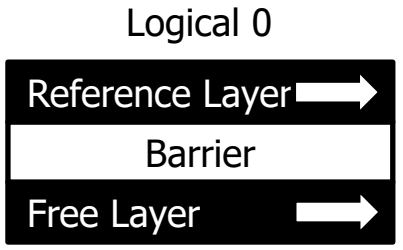
Efficient Data Mapping and Buffering Techniques for Multilevel Cell Phase-Change Memories

HANBIN YOON* and JUSTIN MEZA, Carnegie Mellon University
NAVEEN MURALIMANOHAR, Hewlett-Packard Labs
NORMAN P. JOUPPI**, Google Inc.
ONUR MUTLU, Carnegie Mellon University

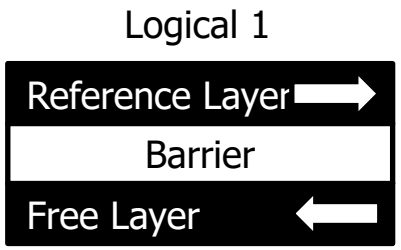
STT-RAM as Main Memory

STT-MRAM as Main Memory

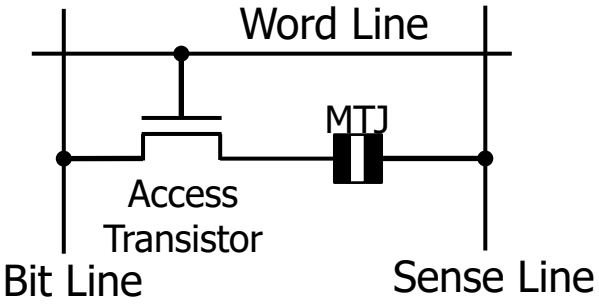
- Magnetic Tunnel Junction (MTJ) device
  - Reference layer: Fixed magnetic orientation
  - Free layer: Parallel or anti-parallel



- Magnetic orientation of the free layer determines logical state of device
  - High vs. low resistance



- Write: Push large current through MTJ to change orientation of free layer
- Read: Sense current flow



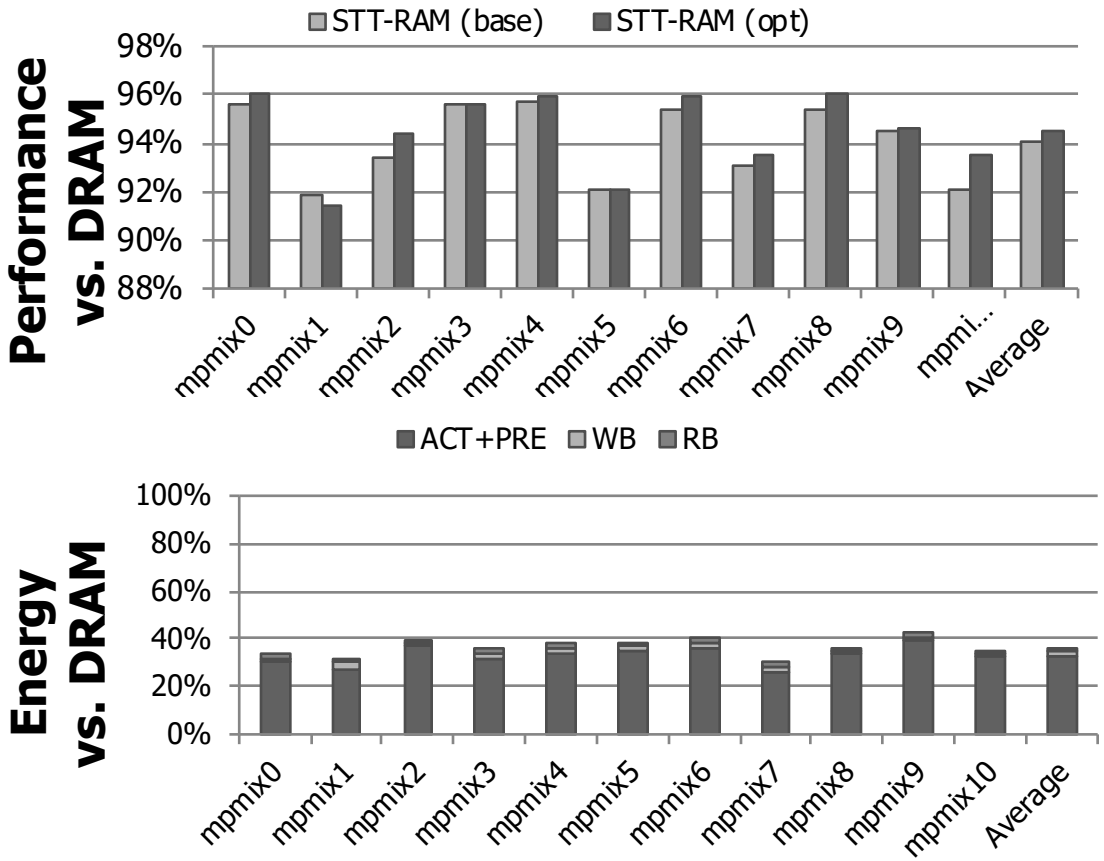
- Kultursay et al., "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

STT-MRAM: Pros and Cons

- Pros over DRAM
  - Better technology scaling (capacity and cost)
  - Non volatile → Persistent
  - Low idle power (no refresh)
- Cons
  - Higher write latency
  - Higher write energy
  - Poor density (currently)
  - Reliability?
- Another level of freedom
  - Can trade off non-volatility for lower write latency/energy (by reducing the size of the MTJ)

Architected STT-MRAM as Main Memory

- 4-core, 4GB main memory, multiprogrammed workloads
- ~6% performance loss, ~60% energy savings vs. DRAM



Kultursay+, "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

More on STT-MRAM as Main Memory

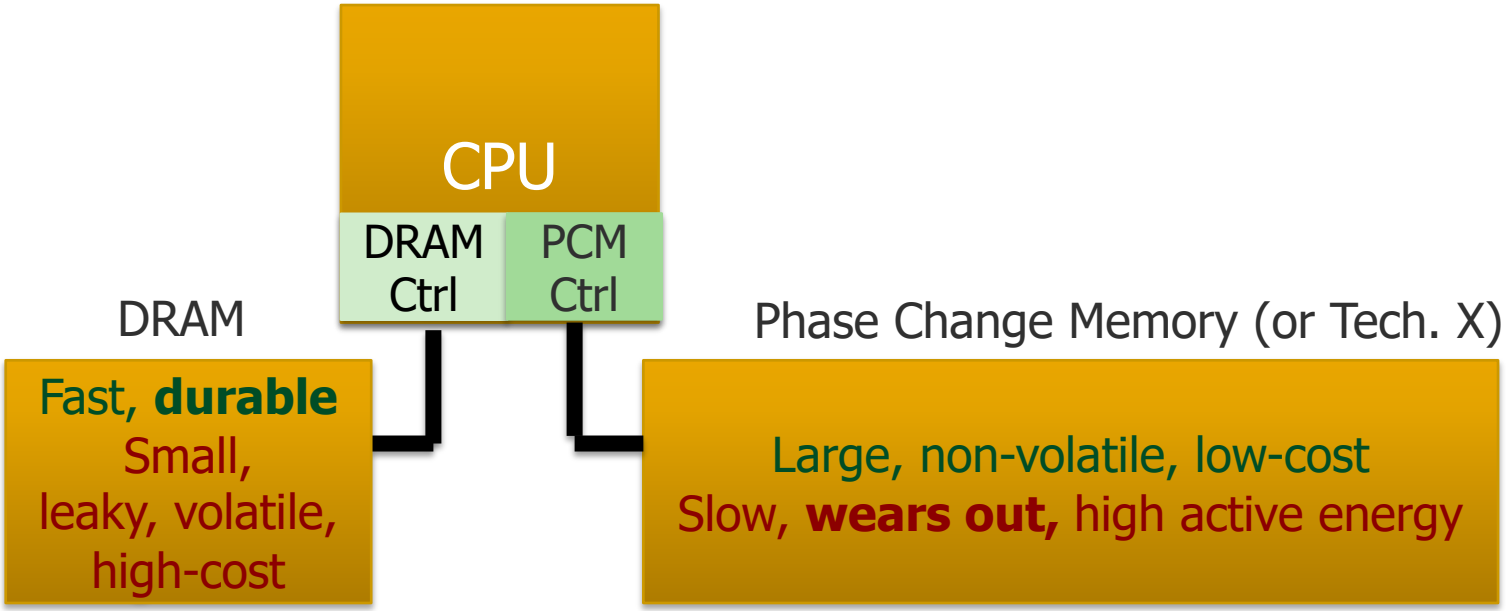
- Emre Kultursay, Mahmut Kandemir, Anand Sivasubramaniam, and Onur Mutlu, **"Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative"**
  *Proceedings of the 2013 IEEE International Symposium on Performance Analysis of Systems and Software (**ISPASS**)*, Austin, TX, April 2013. Slides (pptx) (pdf)

Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative

Emre Kültürsay*, Mahmut Kandemir*, Anand Sivasubramaniam*, and Onur Mutlu†
*The Pennsylvania State University and †Carnegie Mellon University

Hybrid Main Memory

A More Viable Approach: Hybrid Memory Systems



Hardware/software manage data allocation and movement
to achieve the best of multiple technologies

Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

Yoon+, "Row Buffer Locality Aware Caching Policies for Hybrid Memories," ICCD 2012 Best Paper Award.

Challenge and Opportunity

Providing the Best of Multiple Metrics

with

Multiple Memory Technologies

Challenge and Opportunity

Heterogeneous,
Configurable,
Programmable
Memory Systems

Hybrid Memory Systems: Issues

- Cache vs. Main Memory
- Granularity of Data Move/Manage-ment: Fine or Coarse
- Hardware vs. Software vs. HW/SW Cooperative
- When to migrate data?
- How to design a scalable and efficient large cache?
- …

One Option: DRAM as a Cache for PCM

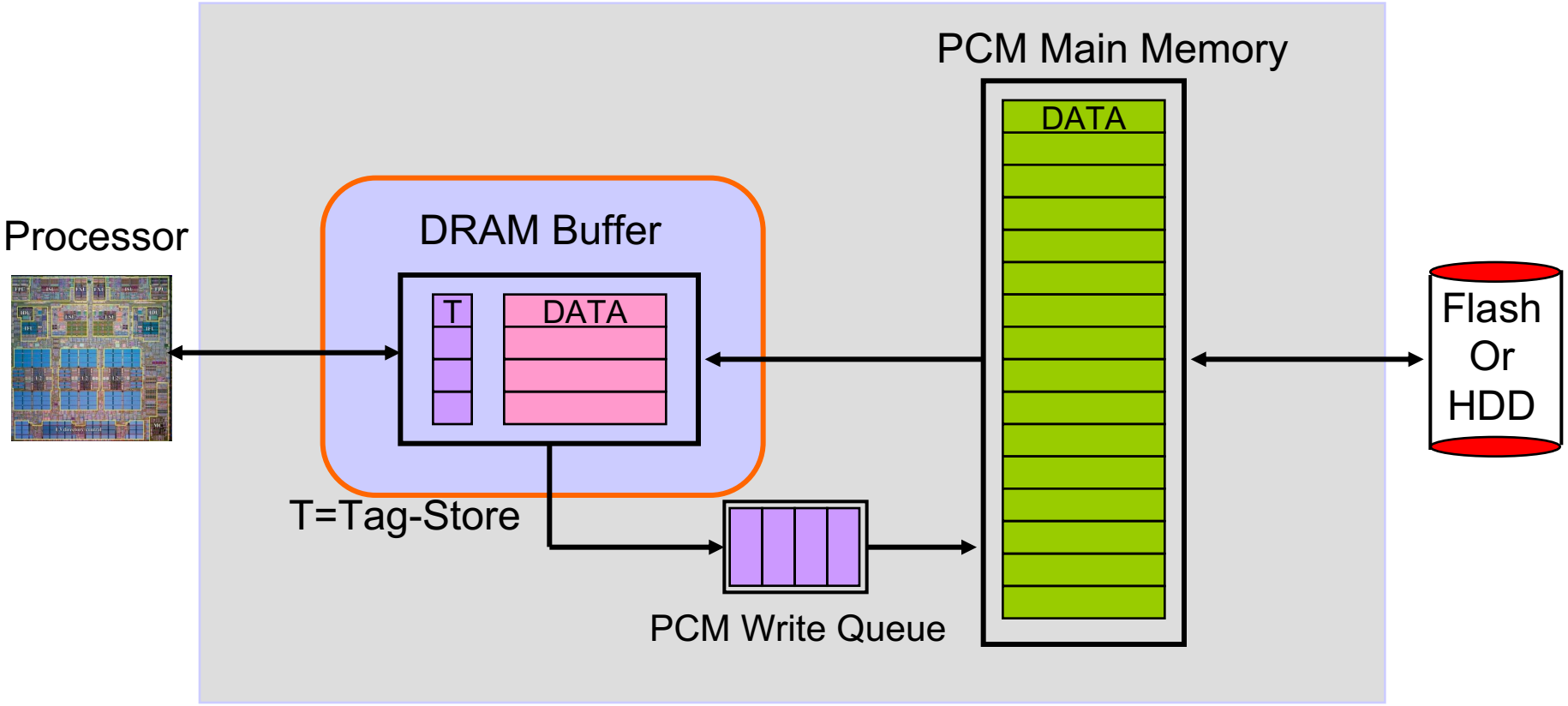
- PCM is main memory; DRAM caches memory rows/blocks
  - Benefits: Reduced latency on DRAM cache hit; write filtering
- Memory controller hardware manages the DRAM cache
  - Benefit: Eliminates system software overhead

- Three issues:
  - What data should be placed in DRAM versus kept in PCM?
  - What is the granularity of data movement?
  - How to design a low-cost hardware-managed DRAM cache?

- Two idea directions:
  - Locality-aware data placement **[Yoon+ , ICCD 2012]**
  - Cheap tag stores and dynamic granularity **[Meza+, IEEE CAL 2012]**

DRAM as a Cache for PCM

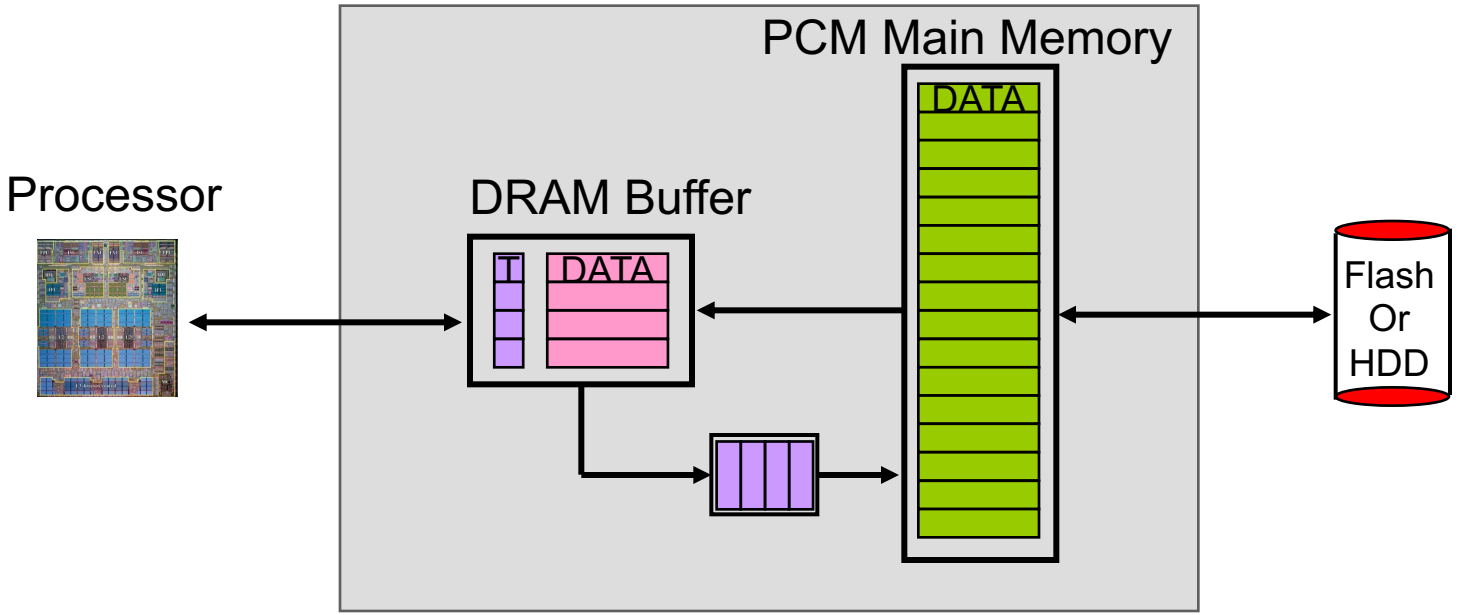
- Goal: Achieve the best of both DRAM and PCM/NVM
  - Minimize amount of DRAM w/o sacrificing performance, endurance
  - DRAM as cache to tolerate PCM latency and write bandwidth
  - PCM as main memory to provide large capacity at good cost and power



Qureshi+, "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Write Filtering Techniques

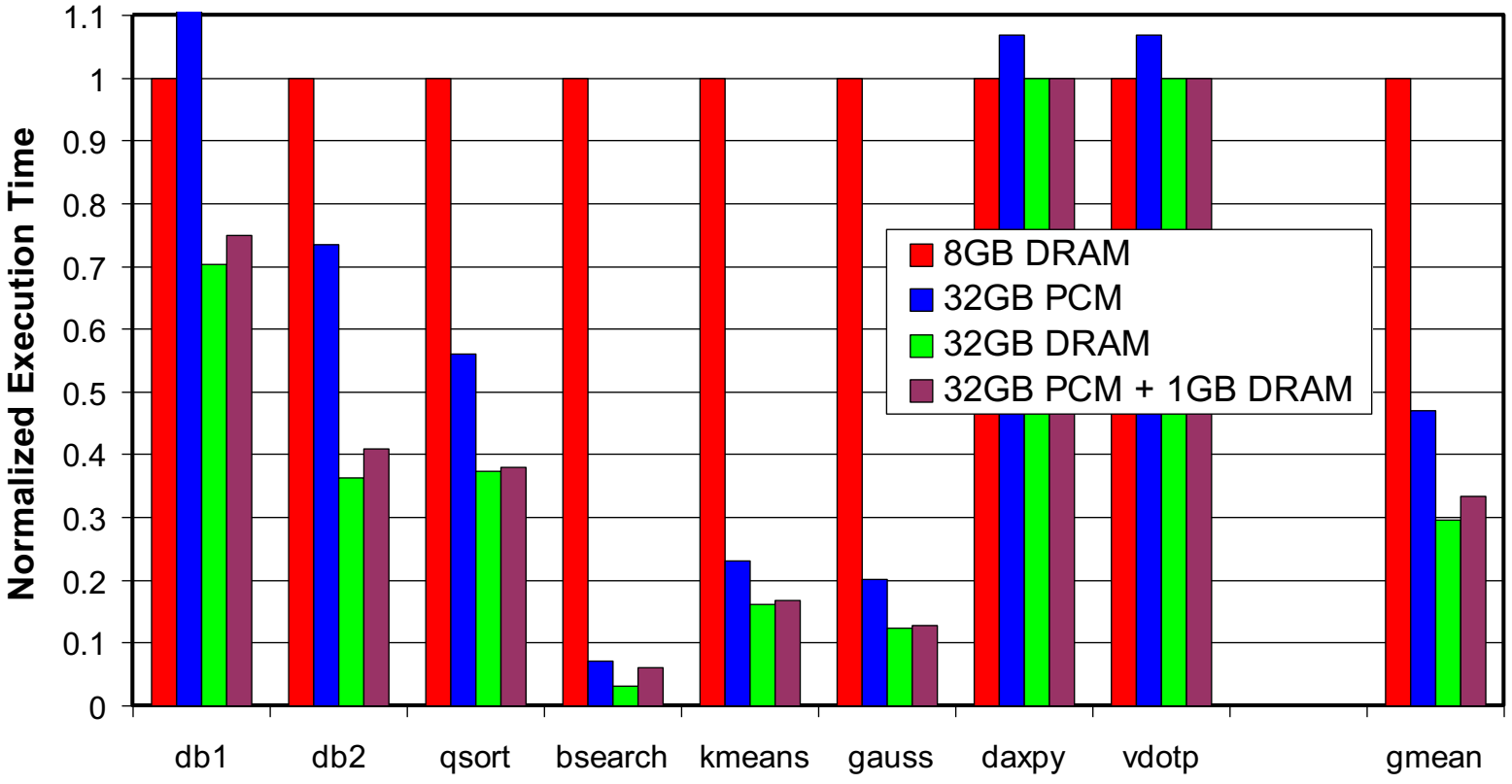
- Lazy Write: Pages from disk installed only in DRAM, not PCM
- Partial Writes: Only dirty lines from DRAM page written back
- Page Bypass: Discard pages with poor reuse on DRAM eviction



- Qureshi et al., "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Results: DRAM as PCM Cache (I)

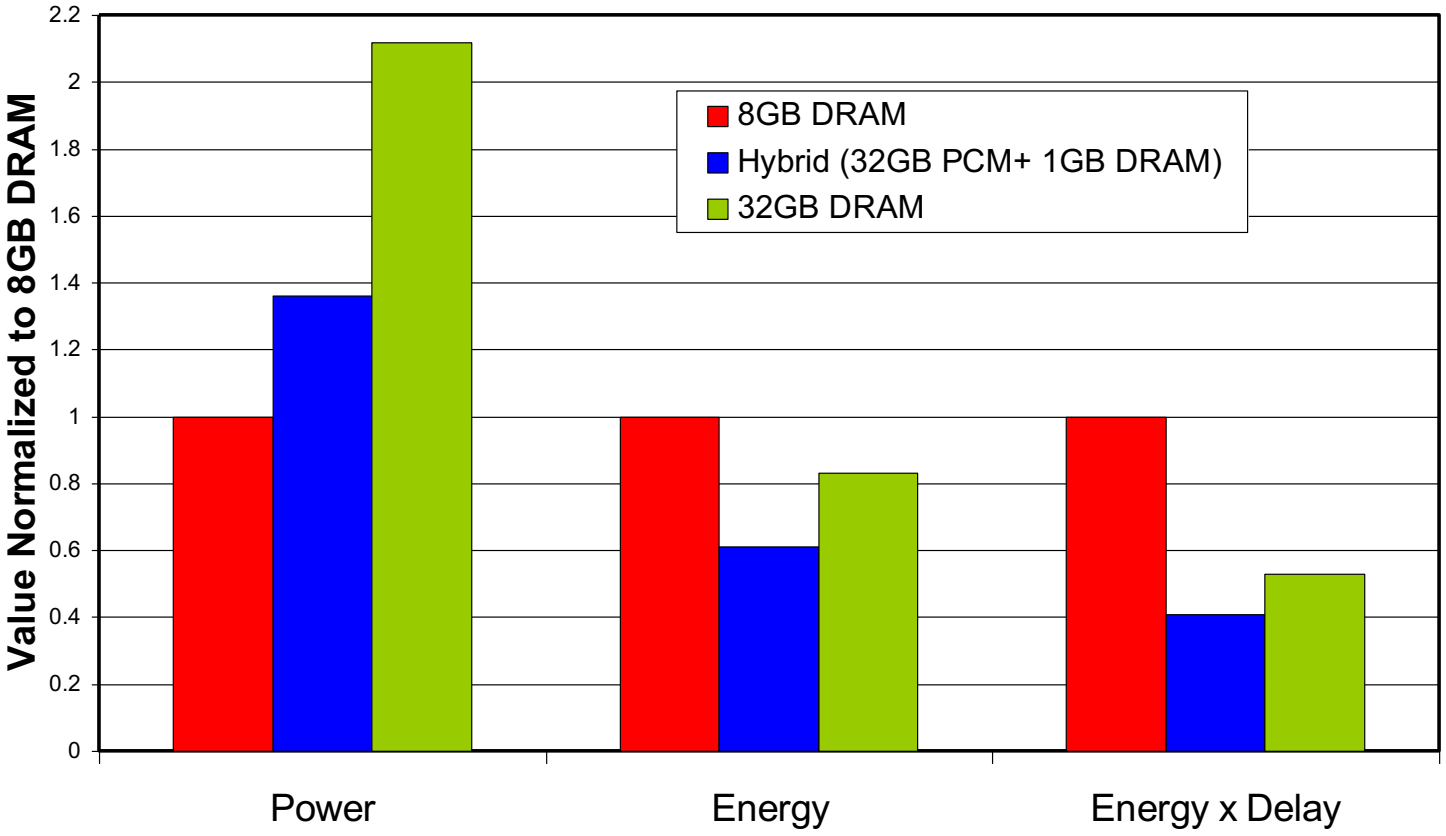
- Simulation of 16-core system, 8GB DRAM main-memory at 320 cycles, HDD (2 ms) with Flash (32 us) with Flash hit-rate of 99%
- Assumption: PCM 4x denser, 4x slower than DRAM
- DRAM block size = PCM page size (4kB)



Qureshi+, "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

Results: DRAM as PCM Cache (II)

- PCM-DRAM Hybrid performs similarly to similar-size DRAM
- Significant energy savings with PCM-DRAM Hybrid
- Average lifetime: 9.7 years (no guarantees)



Qureshi+, "Scalable high performance main memory system using phase-change memory technology," ISCA 2009.

More on DRAM-PCM Hybrid Memory

- **Scalable High-Performance Main Memory System Using Phase-Change Memory Technology**
  Moinuddin K. Qureshi, Viji Srinivasan, and Jude A. Rivers
  Appears in the International Symposium on Computer Architecture (ISCA) 2009.

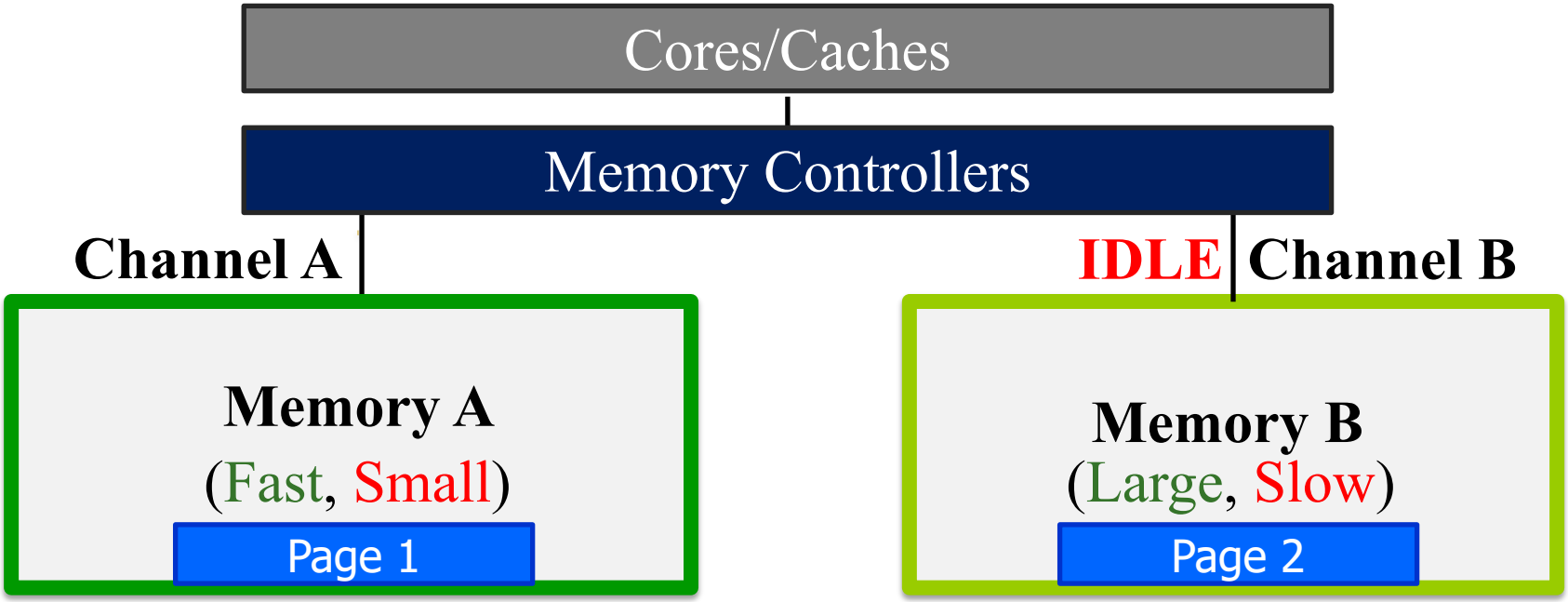
Scalable High Performance Main Memory System Using Phase-Change Memory Technology

Moinuddin K. Qureshi Vijayalakshmi Srinivasan Jude A. Rivers

IBM Research
T. J. Watson Research Center, Yorktown Heights NY 10598

{mkquresh, viji, jarivers}@us.ibm.com

Data Placement in Hybrid Memory



Which memory do we place each page in, to **maximize system performance**?

- Memory A is fast, but small
- Load should be balanced on both channels?
- Page migrations have performance and energy overhead

Data Placement Between DRAM and PCM

- Idea: Characterize data access patterns and guide data placement in hybrid memory
- Streaming accesses: As fast in PCM as in DRAM
- Random accesses: Much faster in DRAM
- Idea: Place random access data with some reuse in DRAM; streaming data in PCM
- Yoon+, "Row Buffer Locality-Aware Data Placement in Hybrid Memories," ICCD 2012 Best Paper Award.

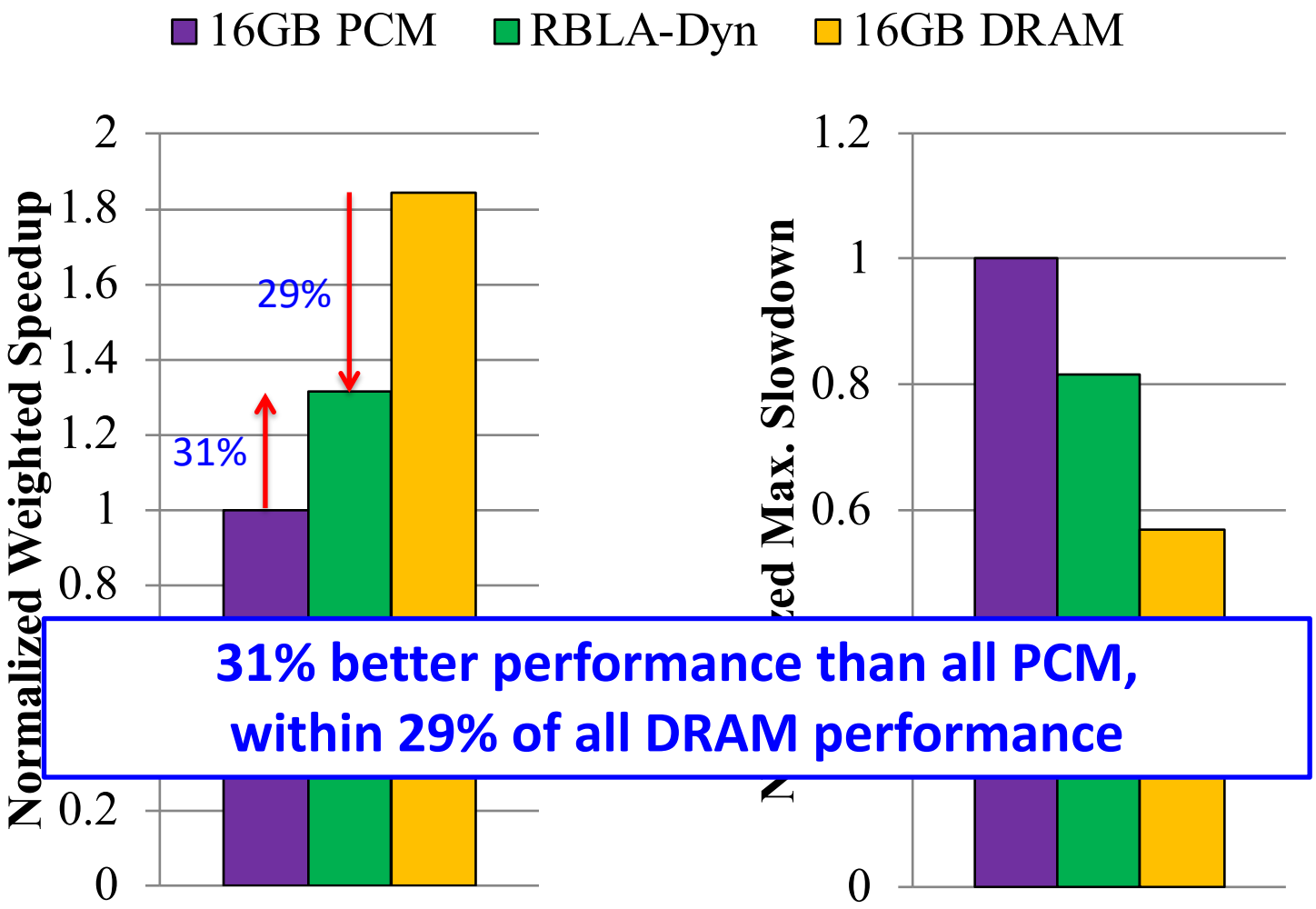
Key Observation & Idea

- Row buffers exist in both DRAM and PCM
  - Row hit latency **similar** in DRAM & PCM [Lee+ ISCA'09]
  - Row miss latency **small** in DRAM, **large** in PCM

- Place data in DRAM which
  - is likely to miss in the row buffer (low row buffer locality)→ miss penalty is smaller in DRAM

    AND
  - is reused many times → cache only the data worth the movement cost and DRAM space

Hybrid vs. All-PCM/DRAM [ICCD'12]



Yoon+, "Row Buffer Locality-Aware Data Placement in Hybrid Memories," ICCD 2012 Best Paper Award.

More on Hybrid Memory Data Placement

- HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael Harding, and Onur Mutlu,
  "Row Buffer Locality Aware Caching Policies for Hybrid Memories"
  *Proceedings of the 30th IEEE International Conference on Computer Design (**ICCD**)*, Montreal, Quebec, Canada, September 2012. Slides (pptx) (pdf)
  Best paper award (in Computer Systems and Applications track).

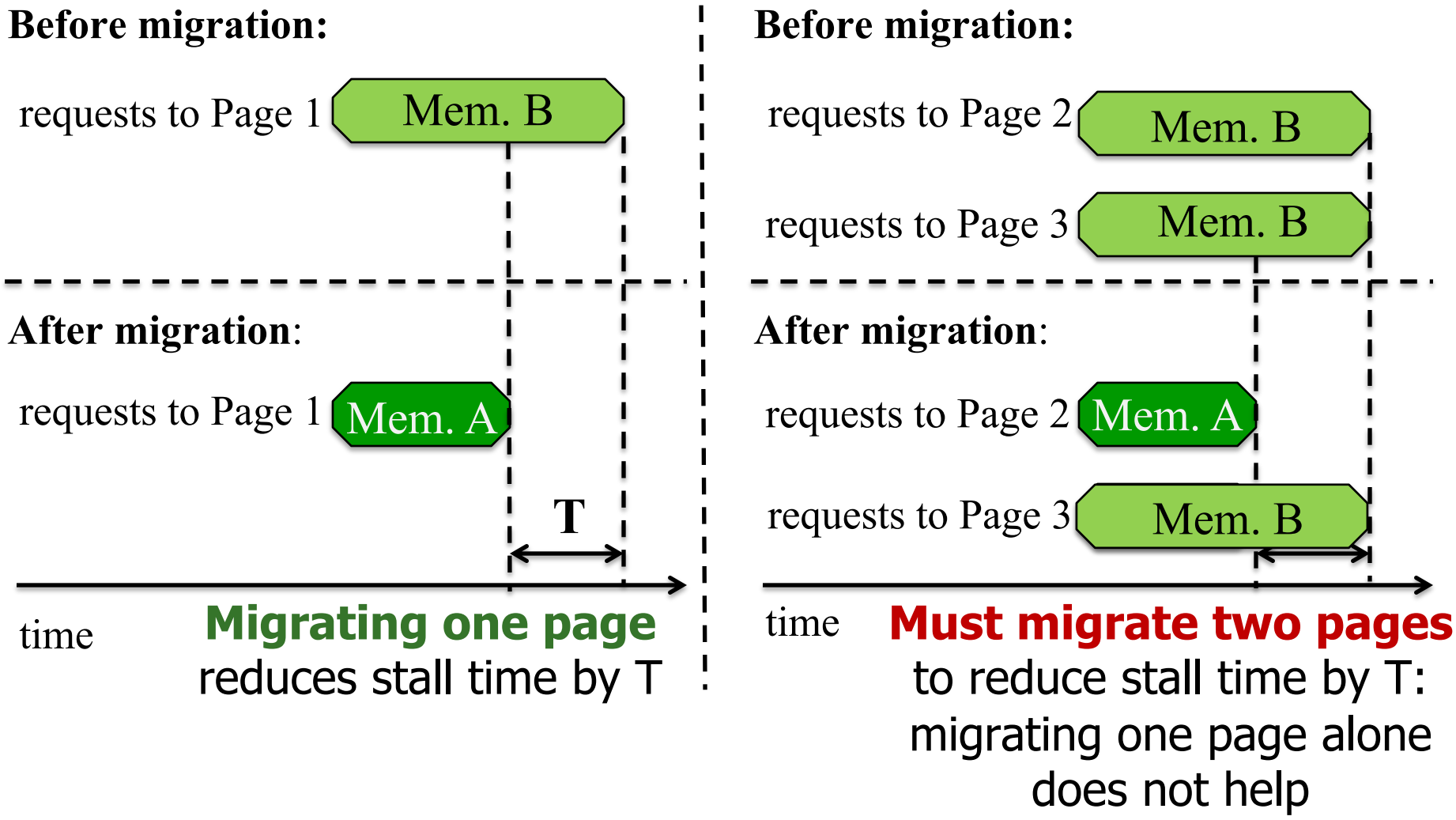
Row Buffer Locality Aware Caching Policies for Hybrid Memories

HanBin Yoon, Justin Meza, Rachata Ausavarungnirun, Rachael A. Harding and Onur Mutlu
Carnegie Mellon University
{hanbinyoon,meza,rachata,onur}@cmu.edu, rhardin@mit.edu

Weaknesses of Existing Solutions

- They are all heuristics that consider only a ***limited part*** **of memory access behavior**
- **Do not *directly* capture the overall system performance impact** of data placement decisions
- Example: None capture **memory-level parallelism** (MLP)
  - Number of ***concurrent*** **memory requests** from the same application when a page is accessed
  - Affects how much page migration helps performance

Importance of Memory-Level Parallelism



Page migration decisions **need to consider MLP**

Our Goal [CLUSTER 2017]

A **generalized** mechanism that

1. Directly estimates the **performance benefit of migrating a page** between **any two types of memory**

2. Places **only** the **performance-critical data** in the fast memory

Utility-Based Hybrid Memory Management

- A memory manager that works for ***any*** hybrid memory
  - e.g., DRAM-NVM, DRAM-RLDRAM

- **Key Idea**
  - For each page, use **comprehensive** characteristics to calculate estimated ***utility*** (i.e., performance impact) of migrating page from one memory to the other in the system
  - **Migrate only pages with the highest utility** (i.e., pages that improve system performance the most when migrated)

- Li+, "Utility-Based Hybrid Memory Management", CLUSTER 2017.

Key Mechanisms of UH-MEM

- For each page, estimate **utility** using a **performance model**
  - **Application stall time reduction**

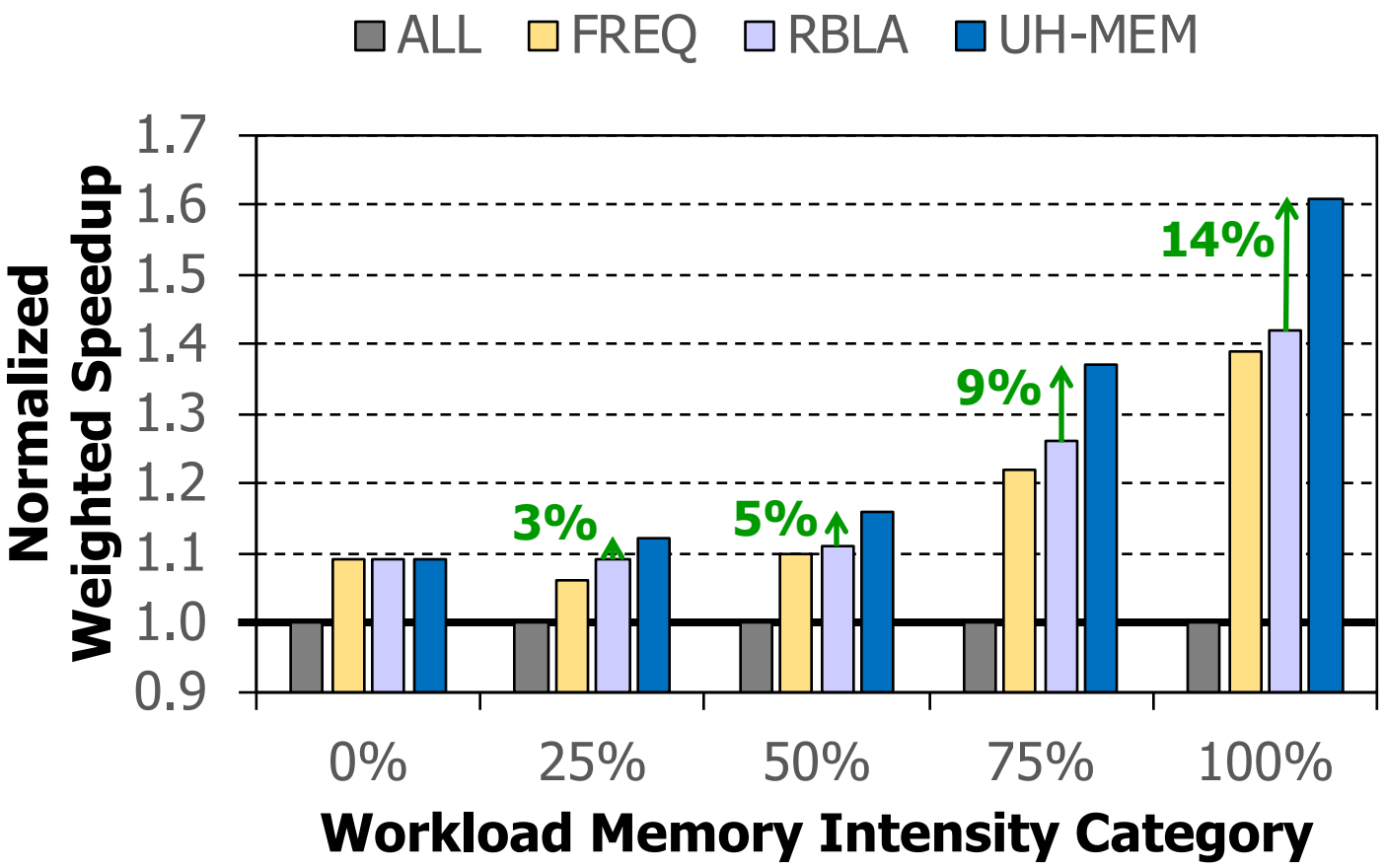
    How much would migrating a page benefit the performance of the application that the page belongs to?
  - **Application performance sensitivity**

    How much does the improvement of a single application's performance increase the *overall* system performance?

$$Utility = \Delta StallTime_i \times Sensitivity_i$$

- **Migrate** only pages whose utility exceed the migration threshold from slow memory to fast memory
- Periodically **adjust migration threshold**

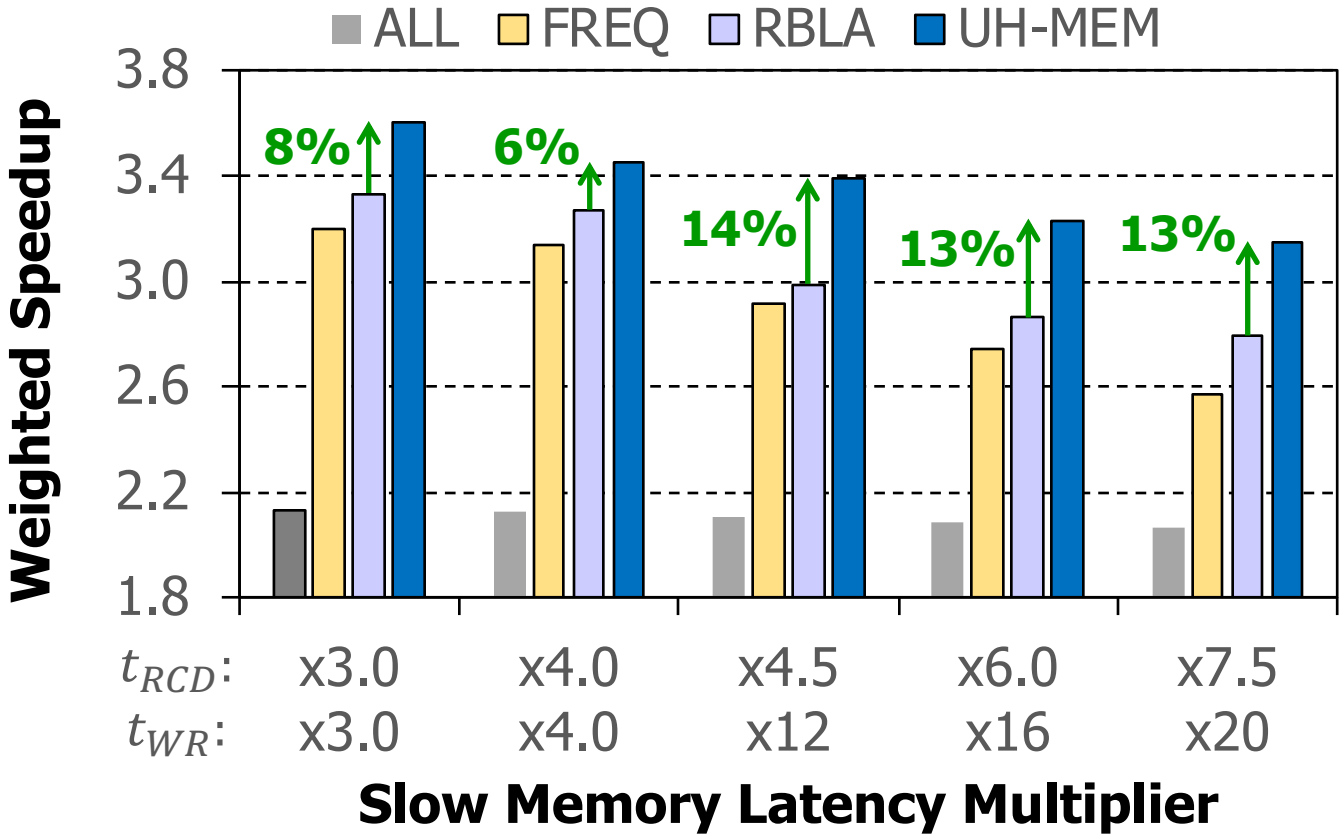
Results: System Performance



UH-MEM improves system performance
over the best state-of-the-art hybrid memory manager

Results: Sensitivity to Slow Memory Latency

- We vary t_{RCD} and t_{WR} of the slow memory



UH-MEM improves system performance for a wide variety of hybrid memory systems

More on UH-MEM

- Yang Li, Saugata Ghose, Jongmoo Choi, Jin Sun, Hui Wang, and Onur Mutlu,
  "Utility-Based Hybrid Memory Management"
  Proceedings of the 19th IEEE Cluster Conference (***CLUSTER***), Honolulu, Hawaii, USA, September 2017.
  [Slides (pptx) (pdf)]

Utility-Based Hybrid Memory Management

Yang Li† Saugata Ghose† Jongmoo Choi‡ Jin Sun† Hui Wang⋆ Onur Mutlu†‡
†*Carnegie Mellon University* ‡*Dankook University* ⋆*Beihang University* †*ETH Zürich*

Challenge and Opportunity

Enabling
an Emerging Technology
to Augment DRAM

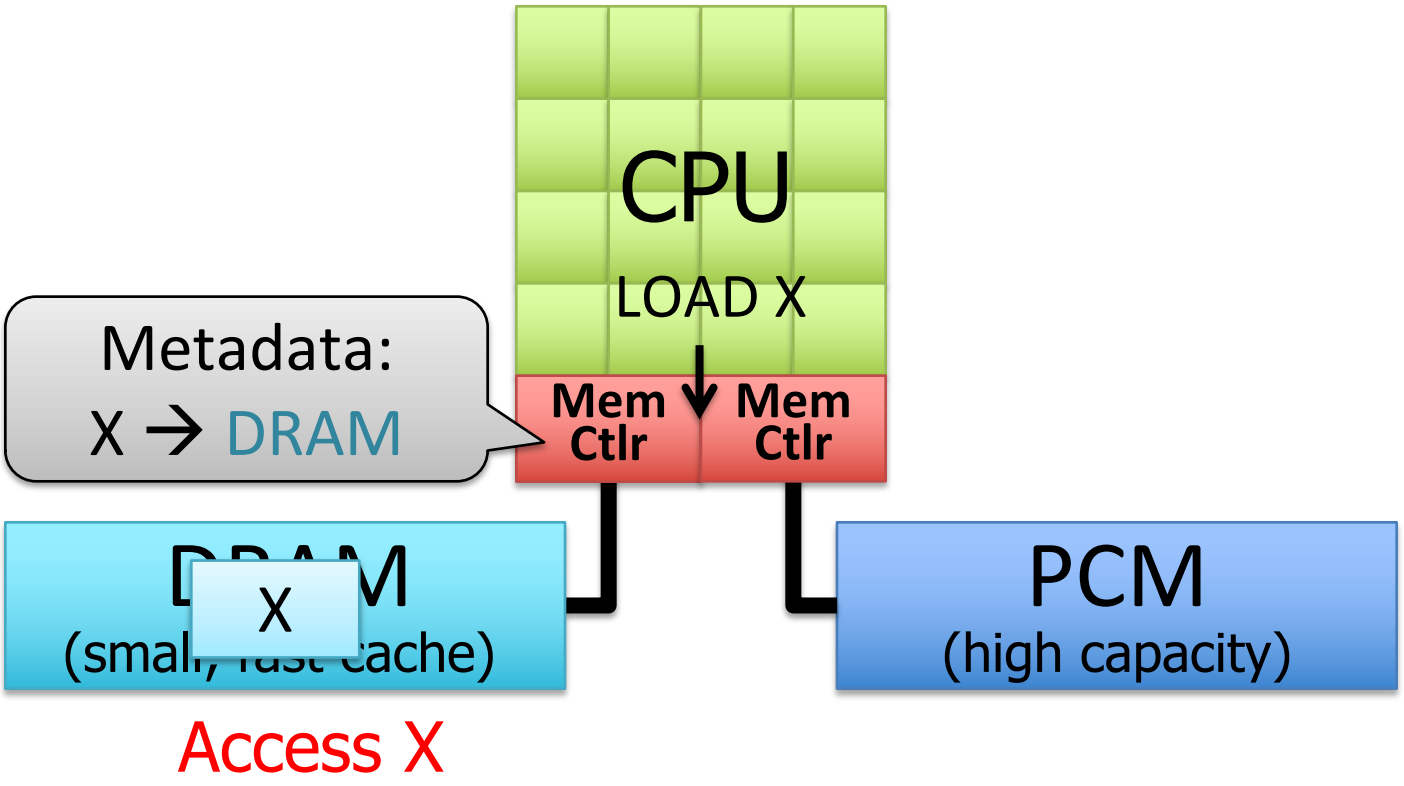
Managing Hybrid Memories

Another Challenge

Designing Effective Large (DRAM) Caches

One Problem with Large DRAM Caches

- A large DRAM cache requires a large metadata (tag + block-based information) store
- How do we design an efficient DRAM cache?



Idea 1: Tags in Memory

- Store tags in the same row as data in DRAM
  - Store metadata in same row as their data
  - Data and metadata can be accessed together

- Benefit: No on-chip tag storage overhead
- Downsides:
  - Cache hit determined only after a DRAM access
  - Cache hit requires two DRAM accesses

Idea 2: Cache Tags in SRAM

- Recall Idea 1: Store all metadata in DRAM
  - To reduce metadata storage overhead

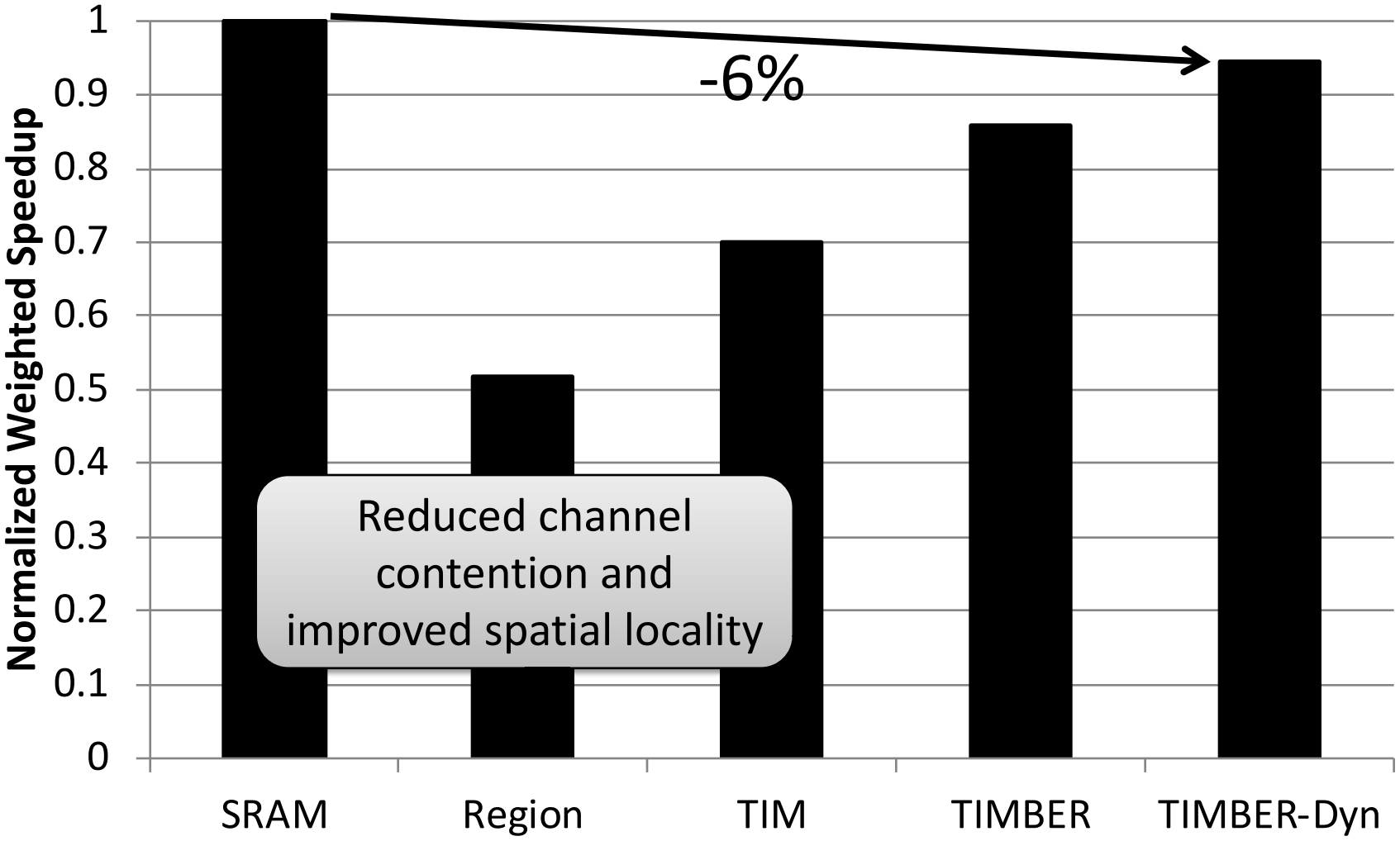
- Idea 2: Cache in on-chip SRAM frequently-accessed metadata
  - Cache only a small amount to keep SRAM size small

Idea 3: Dynamic Data Transfer Granularity

- Some applications benefit from caching more data
  - They have good spatial locality
- Others do not
  - Large granularity wastes bandwidth and reduces cache utilization

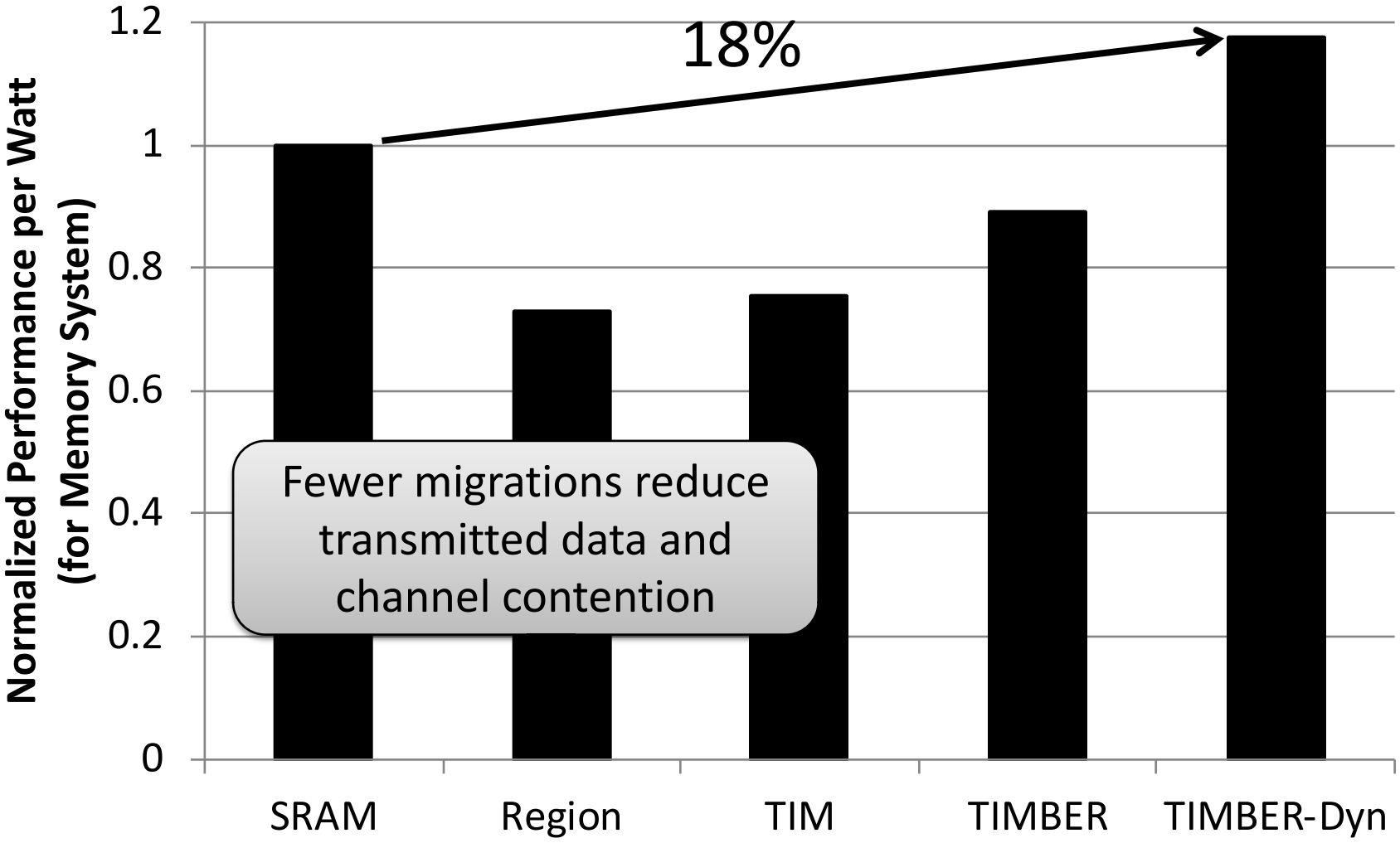
- Idea 3: Simple dynamic caching granularity policy
  - Cost-benefit analysis to determine best DRAM cache block size
  - Group main memory into sets of rows
  - Different sampled row sets follow different fixed caching granularities
  - The rest of main memory follows the best granularity
    - Cost–benefit analysis: access latency versus number of cachings
    - Performed every quantum

TIMBER Performance



Meza, Chang, Yoon, Mutlu, Ranganathan, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

TIMBER Energy Efficiency



Meza, Chang, Yoon, Mutlu, Ranganathan, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

On Large DRAM Cache Design

- Justin Meza, Jichuan Chang, HanBin Yoon, Onur Mutlu, and Parthasarathy Ranganathan,
  "Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management"
  IEEE Computer Architecture Letters (***CAL***), February 2012.

Enabling Efficient and Scalable Hybrid Memories Using Fine-Granularity DRAM Cache Management

Justin Meza* Jichuan Chang† HanBin Yoon* Onur Mutlu* Parthasarathy Ranganathan†
*Carnegie Mellon University †Hewlett-Packard Labs
{meza,hanbinyoon,onur}@cmu.edu {jichuan.chang,partha.ranganathan}@hp.com

DRAM Caches: Many Recent Options

Table 1: Summary of Operational Characteristics of Different State-of-the-Art DRAM Cache Designs – We assume perfect way prediction for Unison Cache. Latency is relative to the access time of the off-package DRAM (see Section 6 for baseline latencies). We use different colors to indicate the high (dark red), medium (white), and low (light green) overhead of a characteristic.

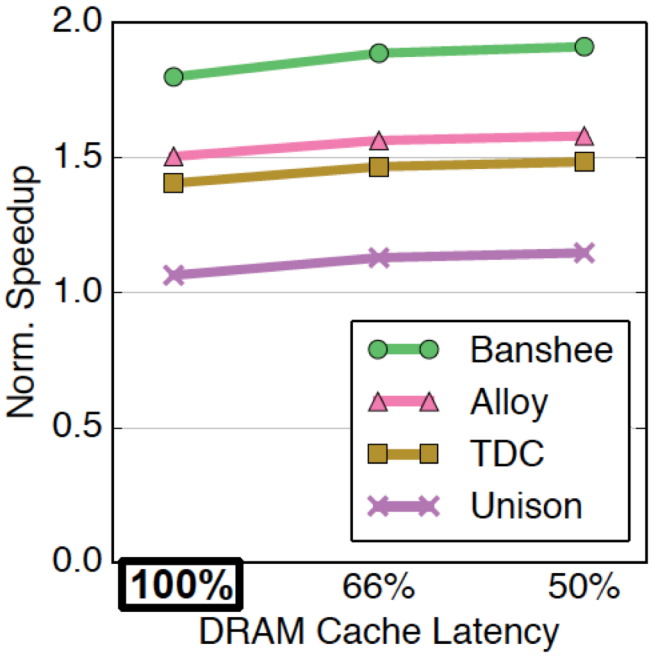
Scheme	DRAM Cache Hit	DRAM Cache Miss	Replacement Traffic	Replacement Decision	Large Page Caching
Unison [32]	In-package traffic: 128 B (data + tag read and update) Latency: ~1x	In-package traffic: 96 B (spec. data + tag read) Latency: ~2x	On every miss Footprint size [31]	Hardware managed, set-associative, LRU	Yes
Alloy [50]	In-package traffic: 96 B (data + tag read) Latency: ~1x	In-package traffic: 96 B (spec. data + tag read) Latency: ~2x	On some misses Cacheline size (64 B)	Hardware managed, direct-mapped, stochastic [20]	Yes
TDC [38]	In-package traffic: 64 B Latency: ~1x TLB coherence	In-package traffic: 0 B Latency: ~1x TLB coherence	On every miss Footprint size [28]	Hardware managed, fully-associative, FIFO	No
HMA [44]	In-package traffic: 64 B Latency: ~1x	In-package traffic: 0 B Latency: ~1x	Software managed, high replacement cost		Yes
Banshee (This work)	In-package traffic: 64 B Latency: ~1x	In-package traffic: 0 B Latency: ~1x	Only for hot pages Page size (4 KB)	Hardware managed, set-associative, frequency based	Yes

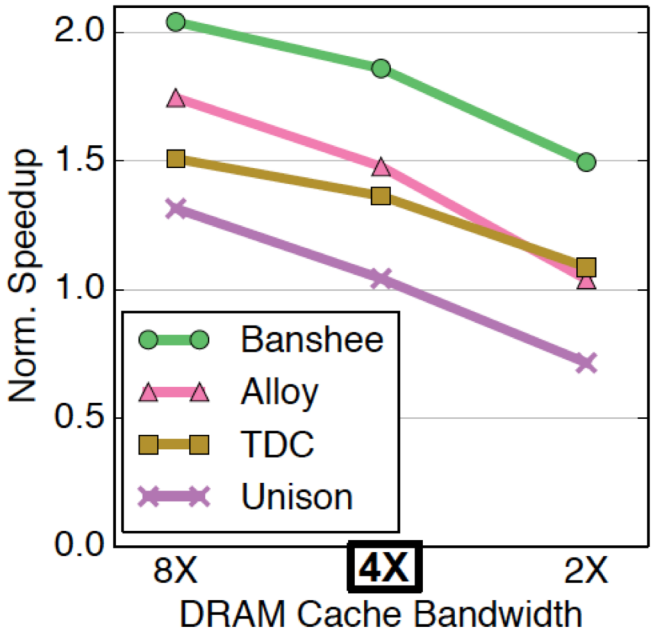
Yu+, "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation," MICRO 2017.

Banshee [MICRO 2017]

- Tracks presence in cache using TLB and Page Table
  - No tag store needed for DRAM cache
  - Enabled by a new lightweight lazy TLB coherence protocol

- New bandwidth-aware frequency-based replacement policy





More on Banshee

- Xiangyao Yu, Christopher J. Hughes, Nadathur Satish, Onur Mutlu, and Srinivas Devadas,
  "Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation"
  *Proceedings of the 50th International Symposium on Microarchitecture (**MICRO**)*, Boston, MA, USA, October 2017.

Banshee: Bandwidth-Efficient DRAM Caching via Software/Hardware Cooperation

Xiangyao Yu[1] Christopher J. Hughes[2] Nadathur Satish[2] Onur Mutlu[3] Srinivas Devadas[1]
[1]MIT [2]Intel Labs [3]ETH Zürich

Other Opportunities with Emerging Memory Technologies

Other Opportunities with Emerging Technologies

- Merging of memory and storage
  - e.g., a single interface to manage all data
- New applications
  - e.g., ultra-fast checkpoint and restore
- More robust system design
  - e.g., reducing data loss
- Processing tightly-coupled with memory
  - e.g., enabling efficient search and filtering

Recall: Processing Using Memory

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 Bulk Bitwise 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

Ambit: Bulk-Bitwise in-DRAM Computation

- Vivek Seshadri, Donghyuk Lee, Thomas Mullins, Hasan Hassan, Amirali Boroumand, Jeremie Kim, Michael A. Kozuch, Onur Mutlu, Phillip B. Gibbons, and Todd C. Mowry,
  "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology"
  *Proceedings of the 50th International Symposium on Microarchitecture (**MICRO**)*, Boston, MA, USA, October 2017.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]

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

In-DRAM Bulk Bitwise Execution Paradigm

- 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

SIMDRAM Framework for in-DRAM Computing

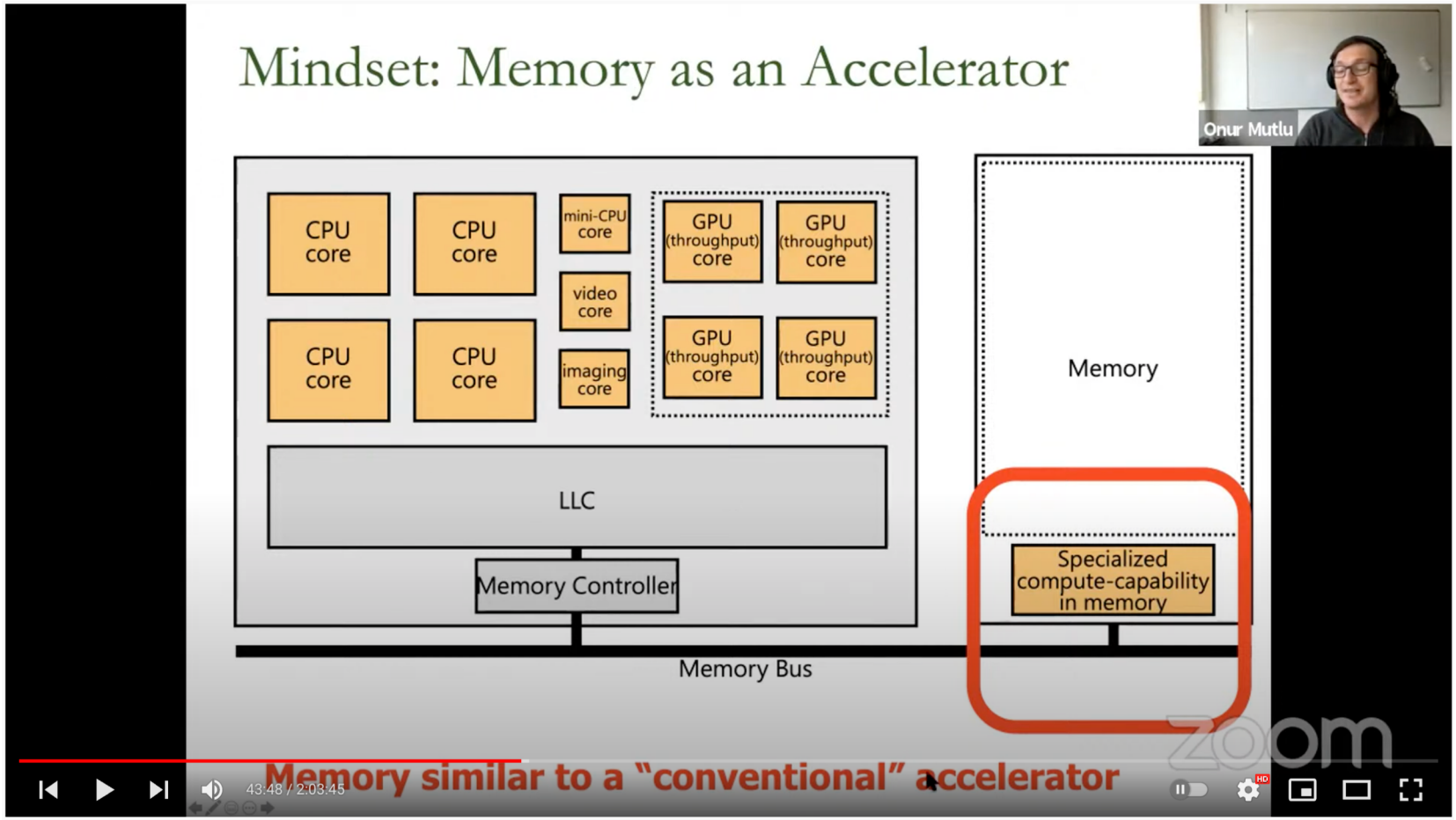
- Nastaran Hajinazar, Geraldo F. Oliveira, Sven Gregorio, Joao Dinis Ferreira, Nika Mansouri Ghiasi, Minesh Patel, Mohammed Alser, Saugata Ghose, Juan Gomez-Luna, and Onur Mutlu, **"SIMDRAM: An End-to-End Framework for Bit-Serial SIMD Computing in DRAM"** *Proceedings of the 26th International Conference on Architectural Support for Programming Languages and Operating Systems (**ASPLOS**)*, Virtual, March-April 2021.
  [2-page Extended Abstract]
  [Short Talk Slides (pptx) (pdf)]
  [Talk Slides (pptx) (pdf)]
  [Short Talk Video (5 mins)]
  [Full Talk Video (27 mins)]

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

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

[1]ETH Zürich [2]Simon Fraser University [3]University of Illinois at Urbana–Champaign

Lecture on RowClone & Processing using DRAM



DEPARTMENT OF INFORMATION TECHNOLOGY AND ELECTRICAL ENGINEERING (D-ITET)

Seminar in Computer Arch. - Meeting 3: RowClone: In-Memory Data Copy and Initialization (Fall 2021)

292 views • Streamed live on Oct 7, 2021

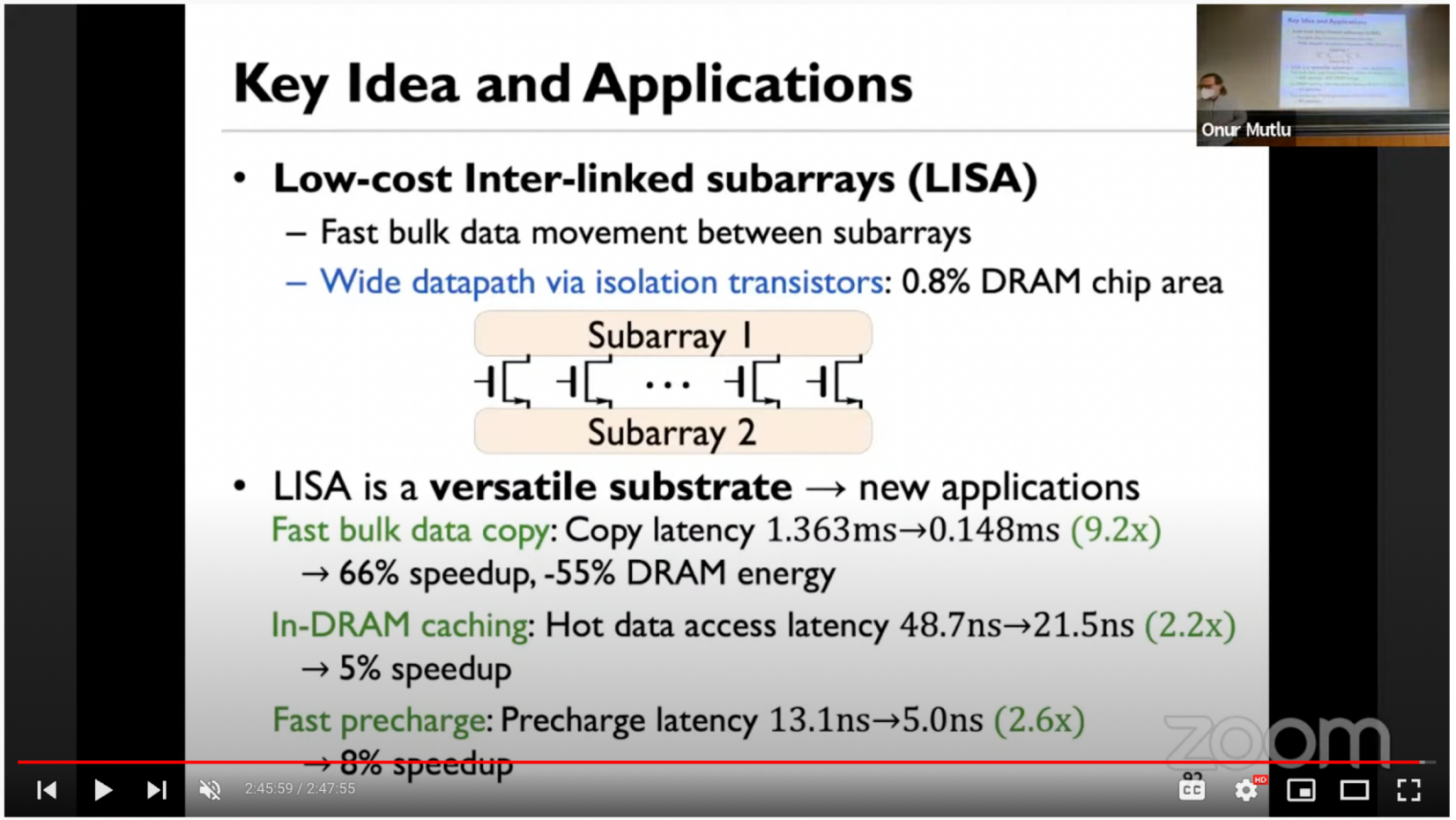
21 0 SHARE SAVE ...

Onur Mutlu Lectures
19.1K subscribers

SUBSCRIBED

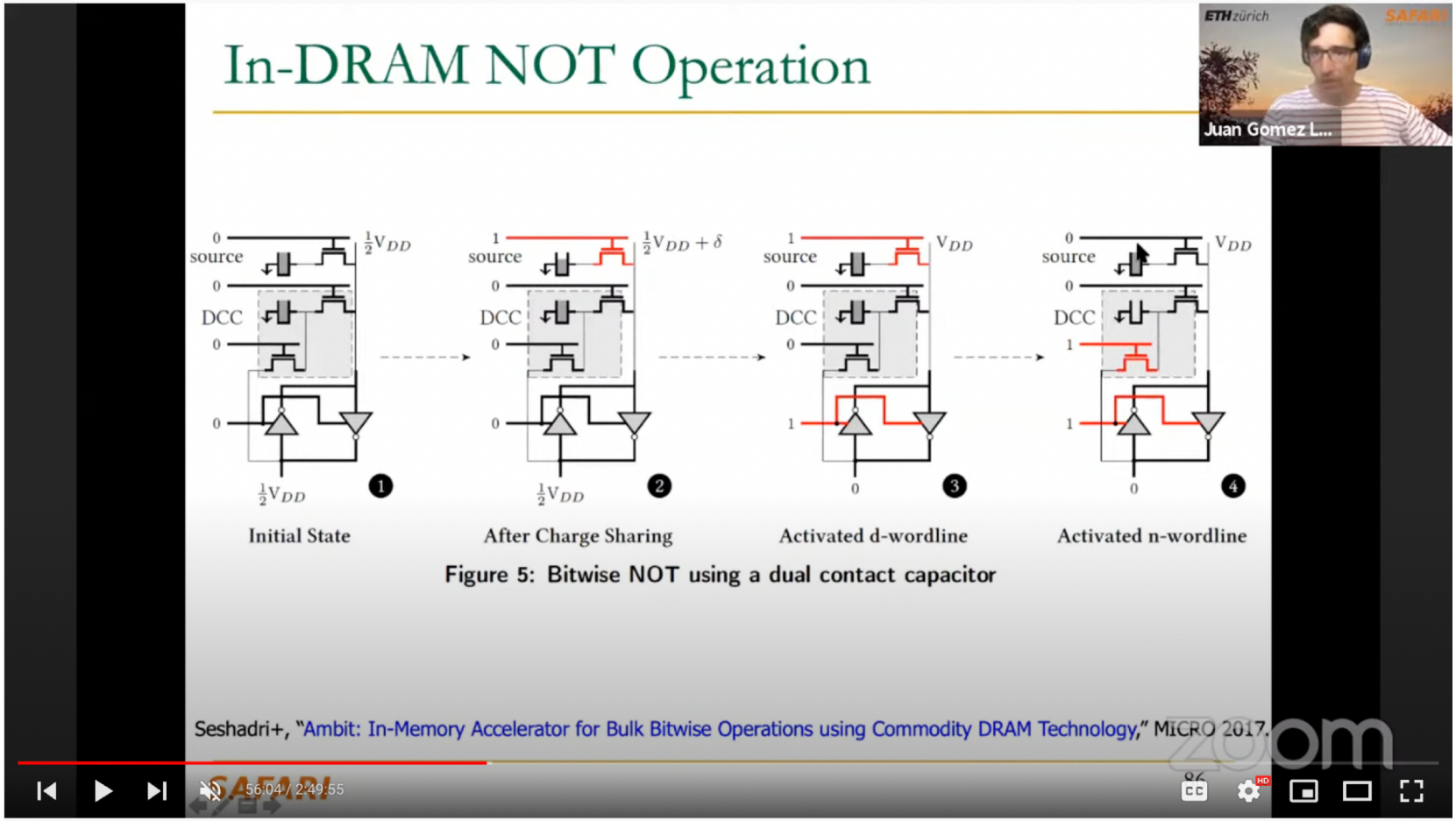
https://www.youtube.com/watch?v=n6Pwg1qax_E&list=PL5Q2soXY2Zi_7UBNmC9B8Yr5JSwTG9yH4&index=4

Lecture on Processing using Memory (I)



https://www.youtube.com/watch?v=HNd4skQrt6I&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF&index=6

Lecture on Processing using Memory (II)



https://www.youtube.com/watch?v=k56x2qcaXWY&list=PL5Q2soXY2Zi-Mnk1PxjEIG32HAGILkTOF&index=7

Pinatubo: RowClone and Bitwise Ops in PCM

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

Shuangchen Li[1]*, Cong Xu[2], Qiaosha Zou[1,5], Jishen Zhao[3], Yu Lu[4], and Yuan Xie[1]

University of California, Santa Barbara[1], Hewlett Packard Labs[2]
University of California, Santa Cruz[3], Qualcomm Inc.[4], Huawei Technologies Inc.[5]
{shuangchenli, yuanxie}ece.ucsb.edu[1]

https://cseweb.ucsd.edu/~jzhao/files/Pinatubo-dac2016.pdf

Pinatubo: RowClone and Bitwise Ops in PCM

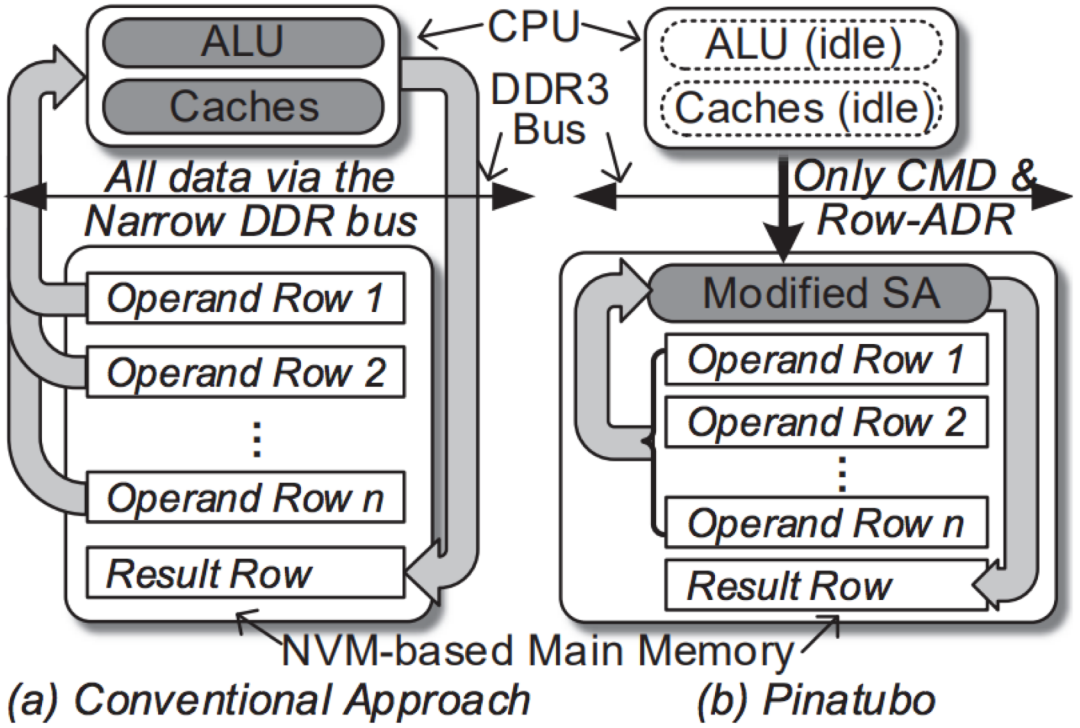


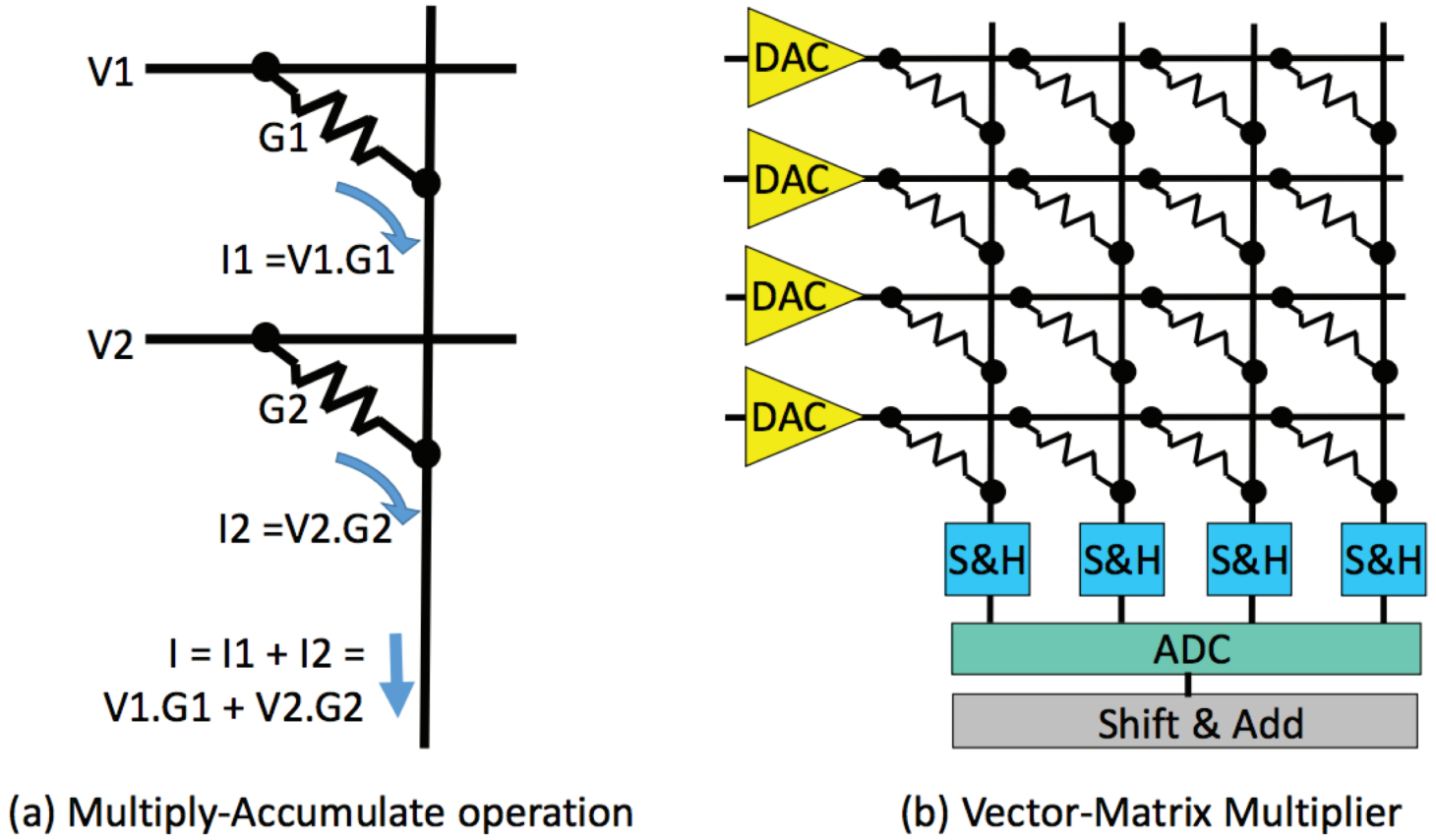
Figure 2: Overview: (a) Computing-centric approach, moving tons of data to CPU and write back. (b) The proposed Pinatubo architecture, performs n-row bitwise operations inside NVM in one step.

New: In-Memory Crossbar Array Operations

In-Memory Crossbar Array Operations

- Some emerging NVM technologies have crossbar array structure
  - Memristors, resistive RAM, phase change mem, STT-MRAM, …
- Crossbar arrays can be used to perform dot product operations using "analog computation capability"
  - Can operate on multiple pieces of data using Kirchoff's laws
    - Bitline current is a sum of products of wordline V x (1 / cell R)
  - Computation is in analog domain inside the crossbar array
- Need peripheral circuitry for D->A and A->D conversion of inputs and outputs

In-Memory Crossbar Computation

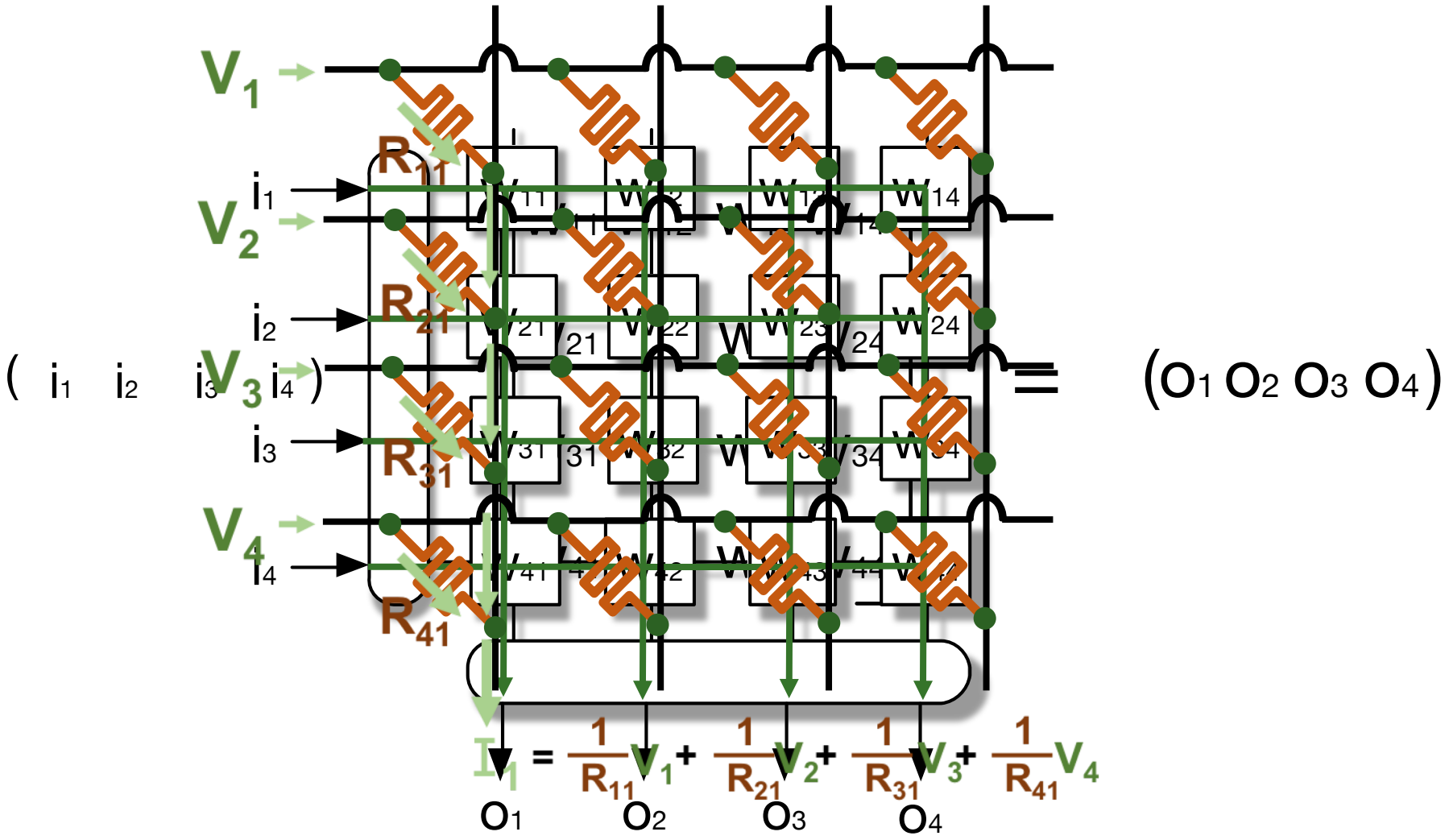


(a) Multiply-Accumulate operation

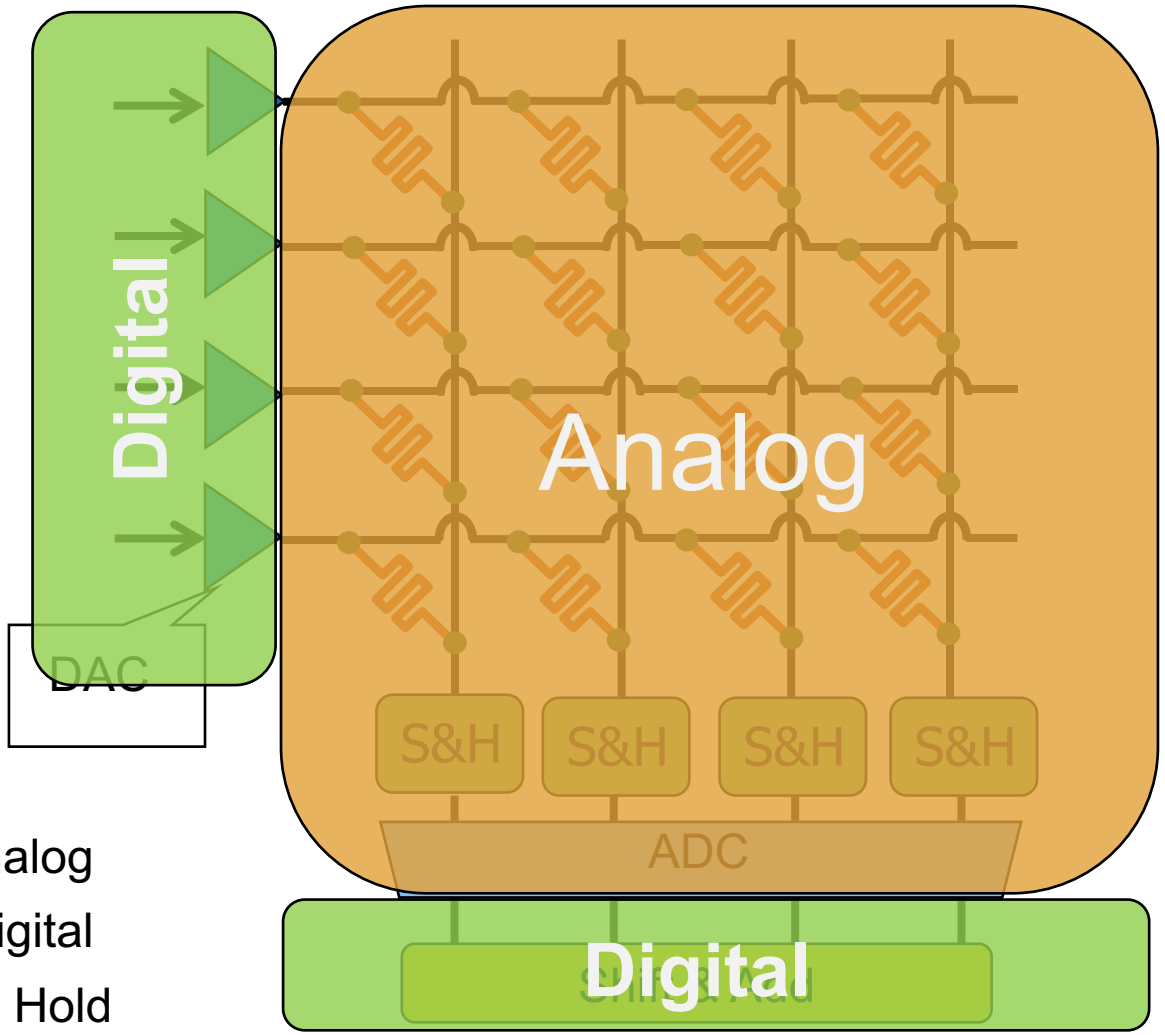
(b) Vector-Matrix Multiplier

Fig. 1. (a) Using a bitline to perform an analog sum of products operation. (b) A memristor crossbar used as a vector-matrix multiplier.

In-Memory Crossbar Computation



Required Peripheral Circuitry



DAC: Digital to Analog

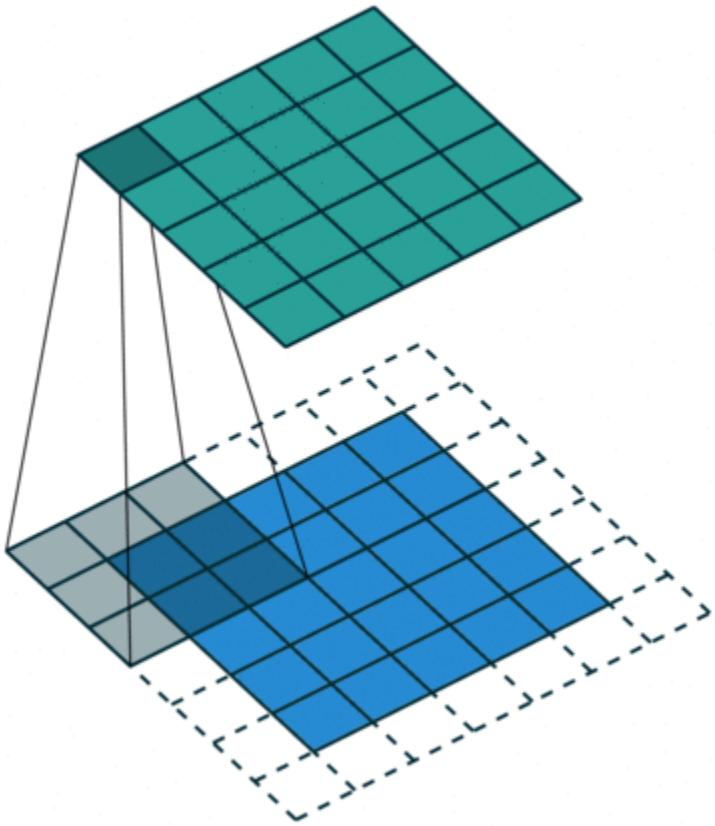
ADC: Analog to Digital

S&H: Sample and Hold

Shift and add: used to summarize the final output

An Example of 2D Convolution

Output feature map



Input feature map

Structure information

Input: 5*5 (blue)

Kernel (filter): 3*3 (grey)

Output: 5*5 (green)

Computation information

Stride: 1

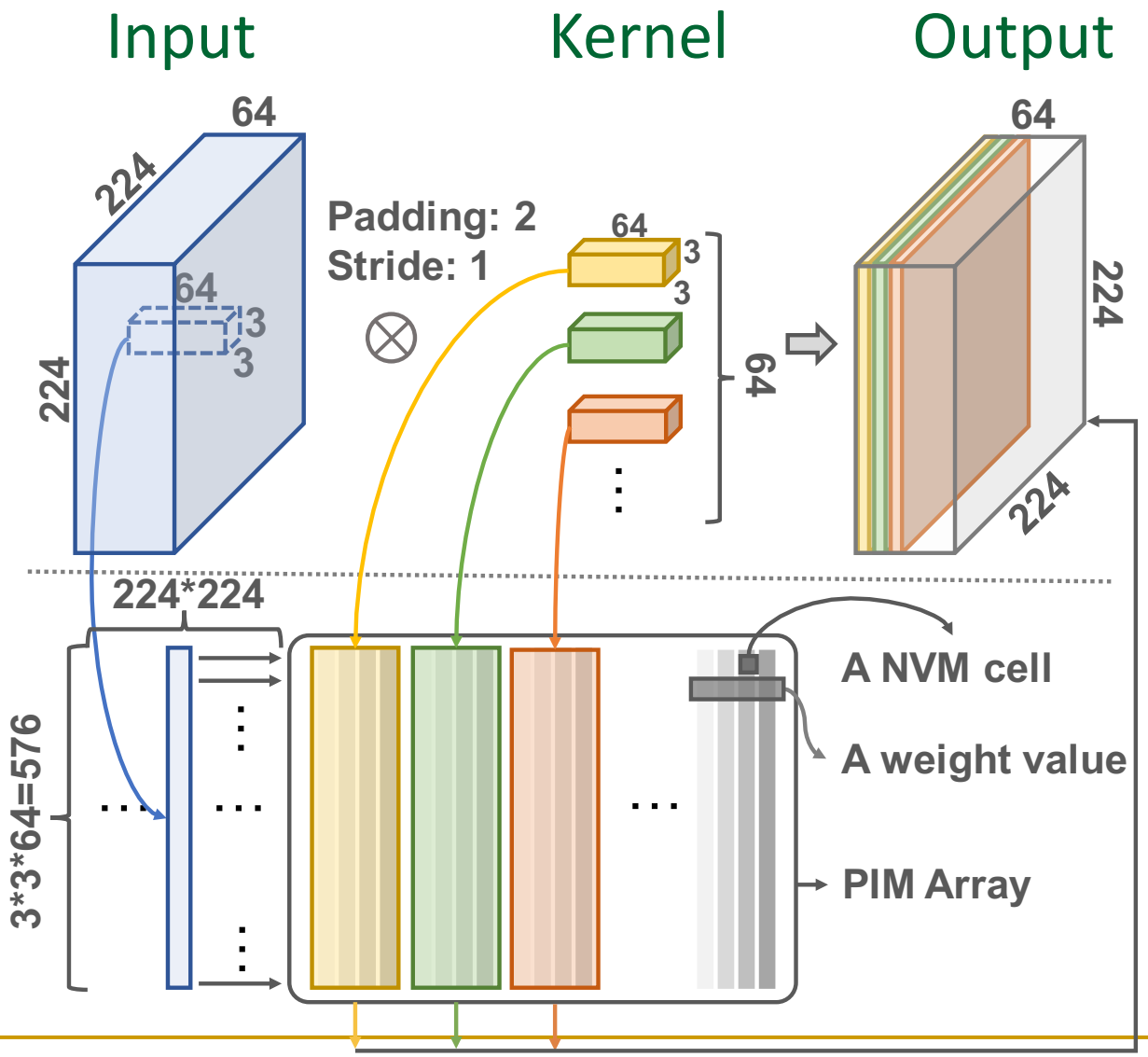
Padding: 1 (white)

Output Dim = (Input + 2*Padding - Kernel) / Stride + 1

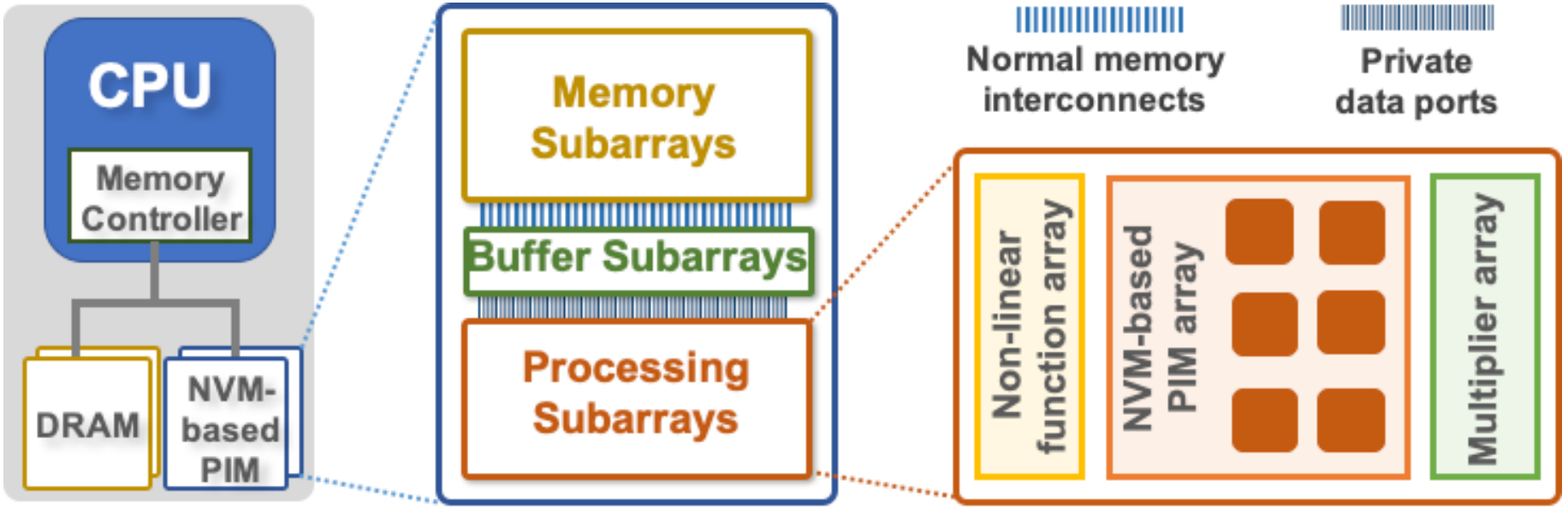
Mapping Computation onto the Crossbar

A convolution operation in neural network application

An NVM-based PIM array



An Overview of NVM-Based PIM System



NVM-based PIM array:

core processing unit for vector-matrix multiplication

Non-linear function array:

processing unit for non-linear functions (e.g., ReLU operations in neural networks)

Multiplier array:

handles element-wise operations

NVM-based PIM used in Genome Analysis

- Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina, Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, and Onur Mutlu,
  "GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping"
  Proceedings of the 55th International Symposium on Microarchitecture (***MICRO***),
  Chicago, IL, USA, October 2022.
  [Slides (pptx) (pdf)]
  [Longer Lecture Slides (pptx) (pdf)]
  [Lecture Video (25 minutes)]
  [arXiv version]

GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

Haiyu Mao[1] Mohammed Alser[1] Mohammad Sadrosadati[1] Can Firtina[1] Akanksha Baranwal[1]
Damla Senol Cali[2] Aditya Manglik[1] Nour Almadhoun Alserr[1] Onur Mutlu[1]
[1]*ETH Zürich* [2]*Bionano Genomics*

NVM-based PIM used in Genome Analysis

- Taha Shahroodi, Gagandeep Singh, Mahdi Zahedi, Haiyu Mao, Joel Lindegger, Can Firtina, Stephan Wong, Onur Mutlu, and Said Hamdioui,
  "Swordfish: A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-In-Memory with Non-Ideal Memristors"
  Proceedings of the 56th International Symposium on Microarchitecture (***MICRO***),
  Toronto, ON, Canada, November 2023.
  [Slides (pptx) (pdf)]
  [arXiv version]

Swordfish: A Framework for Evaluating Deep Neural Network-based Basecalling using Computation-In-Memory with Non-Ideal Memristors

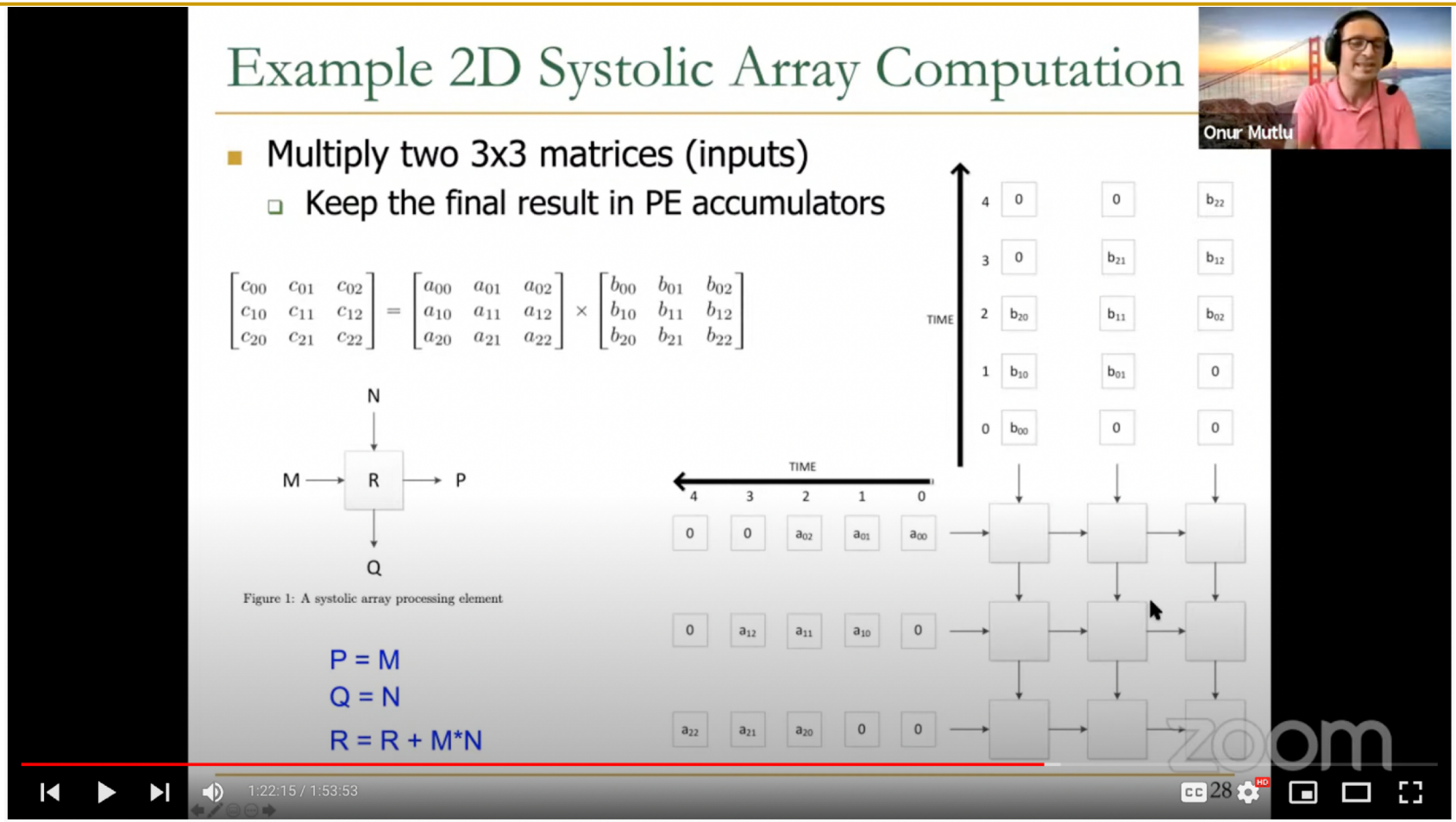
Taha Shahroodi[1] Gagandeep Singh[2,3] Mahdi Zahedi[1] Haiyu Mao[3] Joel Lindegger[3] Can Firtina[3]
Stephan Wong[1] Onur Mutlu[3] Said Hamdioui[1]

[1]TU Delft [2]AMD Research [3]ETH Zürich

Example Readings on NVM-Based PIM

- Shafiee+, "ISAAC: A Convolutional Neural Network Accelerator with In-Situ Analog Arithmetic in Crossbars", ISCA 2016.
- Chi+, "PRIME: A Novel Processing-in-memory Architecture for Neural Network Computation in ReRAM-based Main Memory", ISCA 2016.
- Prezioso+, "Training and Operation of an Integrated Neuromorphic Network based on Metal-Oxide Memristors", Nature 2015
- Ambrogio+, "Equivalent-accuracy accelerated neural-network training using analogue memory", Nature 2018.

Lecture on Systolic Arrays & Convolutions



Digital Design & Computer Arch. - Lecture 19: VLIW, Systolic Arrays, DAE (ETH Zürich, Spring 2021)

2,727 views • Streamed live on May 7, 2021

63 3 SHARE SAVE ...

Onur Mutlu Lectures
20.2K subscribers

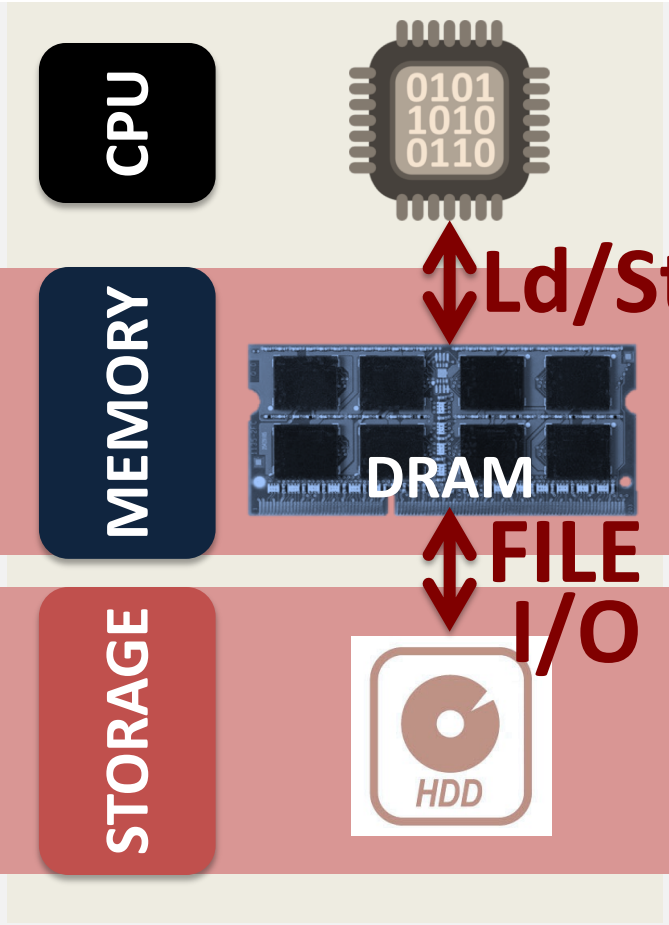
SUBSCRIBED

https://www.youtube.com/watch?v=UtLy4Yagdys&list=PL5Q2soXY2Zi_uej3aY39YB5pfW4SJ7LlN&index=21

Other Opportunities with Emerging Technologies

- Merging of memory and storage
  - e.g., a single interface to manage all data
- New applications
  - e.g., ultra-fast checkpoint and restore
- More robust system design
  - e.g., reducing data loss
- Processing tightly-coupled with memory
  - e.g., enabling efficient search and filtering

TWO-LEVEL STORAGE MODEL



VOLATILE

FAST

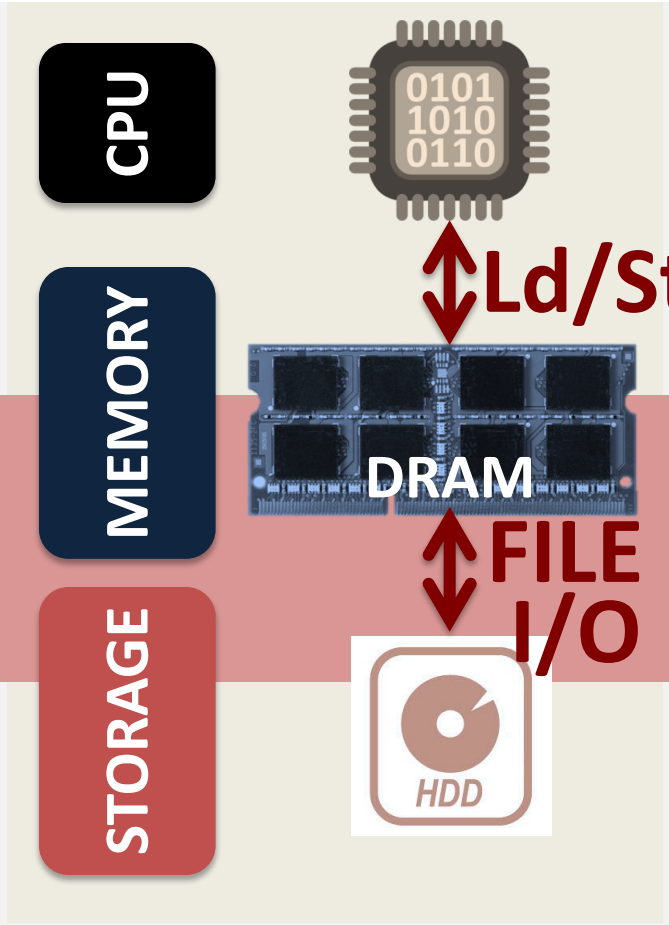
BYTE ADDR

NONVOLATILE

SLOW

BLOCK ADDR

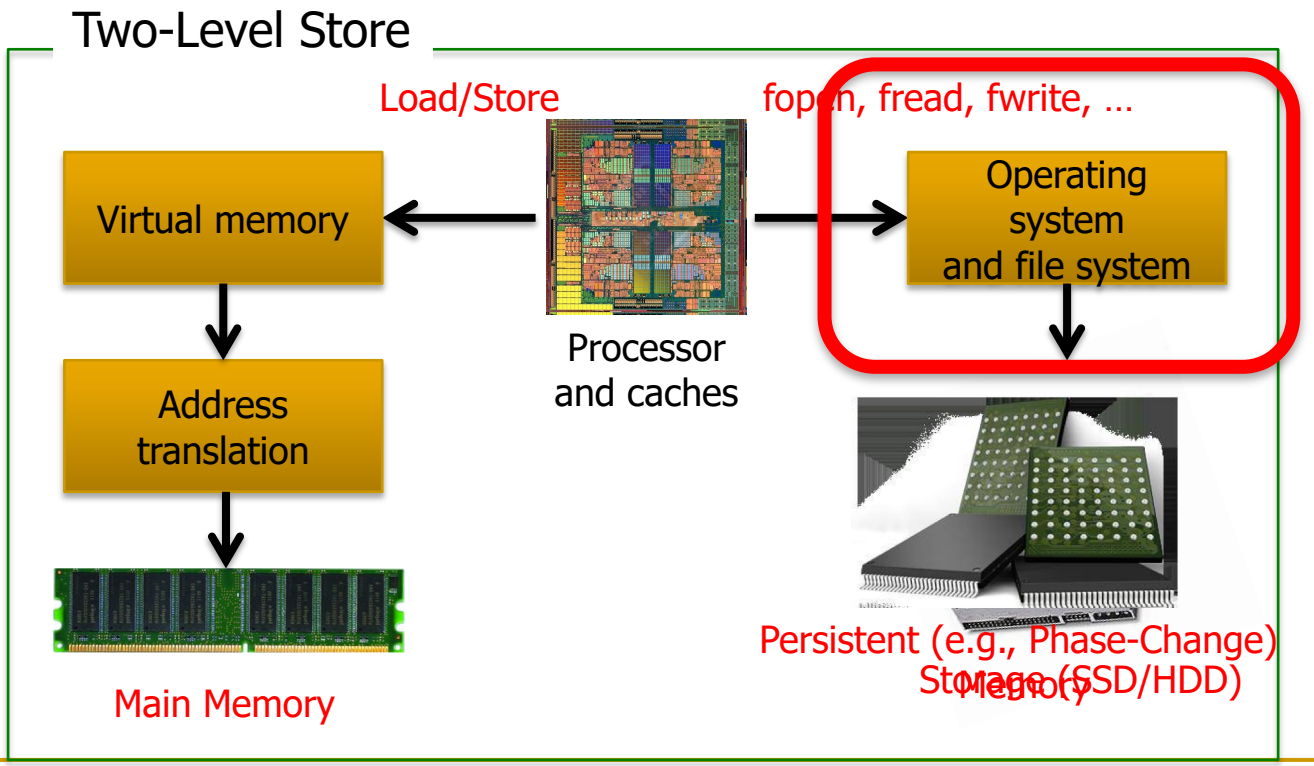
TWO-LEVEL STORAGE MODEL



Non-volatile memories combine characteristics of memory and storage

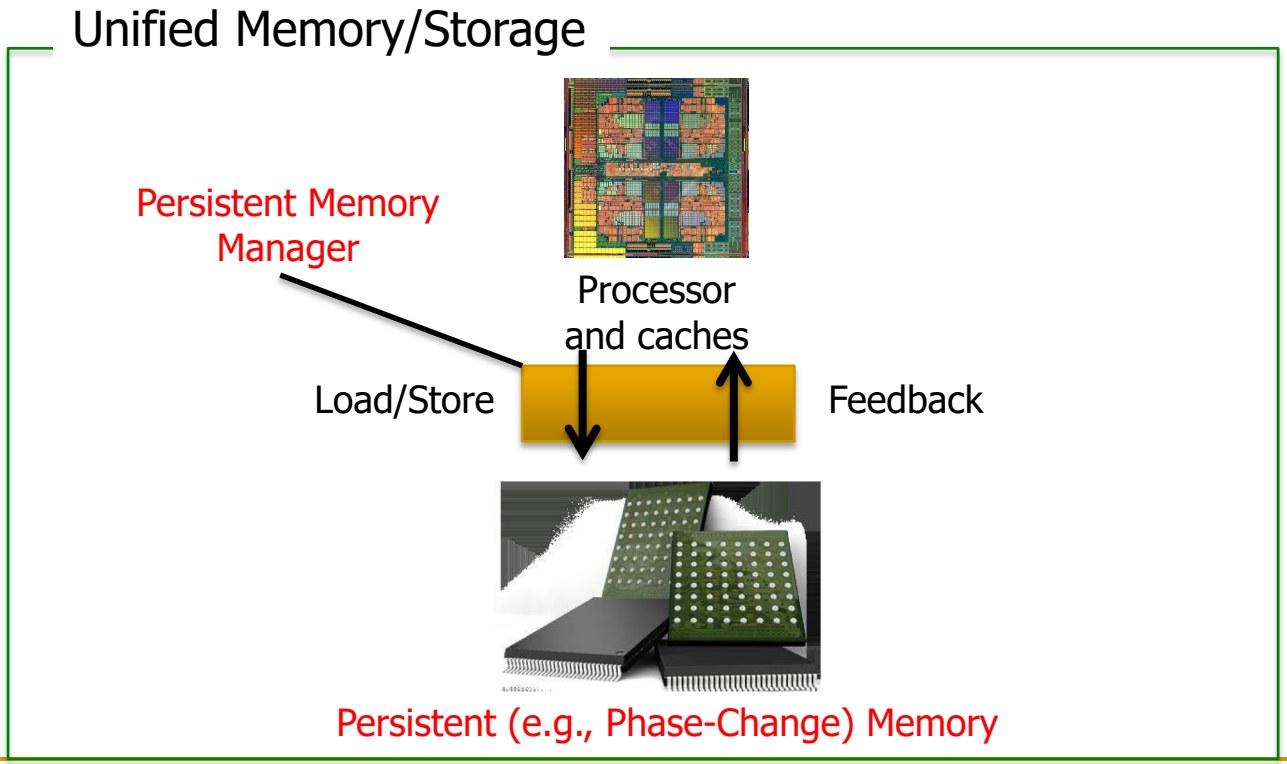
Two-Level Memory/Storage Model

- The traditional two-level storage model is a bottleneck with NVM
  - **Volatile** data in memory → a **load/store** interface
  - **Persistent** data in storage → a **file system** interface
  - Problem: Operating system (OS) and file system (FS) code to locate, translate, buffer data become performance and energy bottlenecks with fast NVM stores



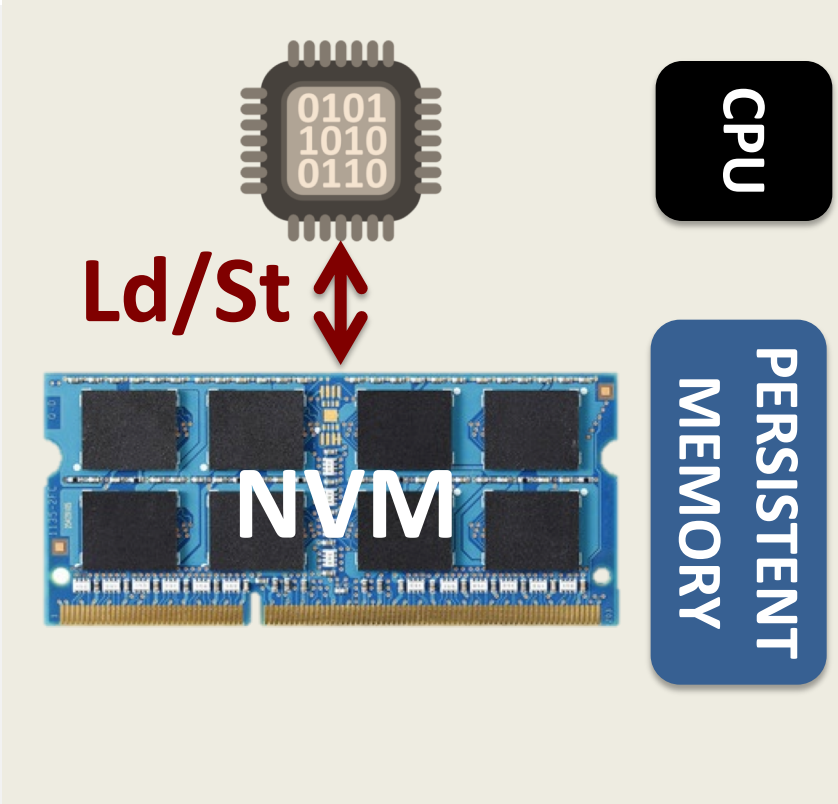
Unified Memory and Storage with NVM

- Goal: Unify memory and storage management in a single unit to eliminate wasted work to locate, transfer, and translate data
  - Improves both energy and performance
  - Simplifies programming model as well



Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

PERSISTENT MEMORY

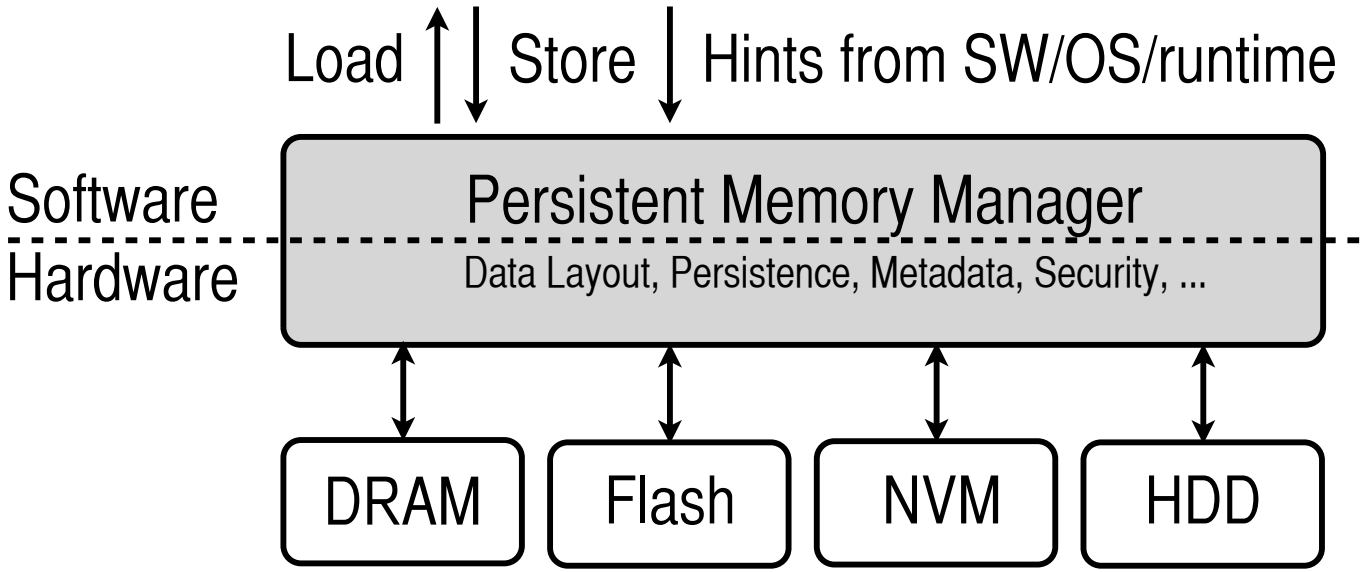


Provides an opportunity to manipulate persistent data directly

The Persistent Memory Manager (PMM)

```
1  int main(void) {
2    // data in file.dat is persistent
3    FILE myData = "file.dat";
4    myData = new int[64];
5  }
6  void updateValue(int n, int value) {
7    FILE myData = "file.dat";
8    myData[n] = value; // value is persistent
9  }
```

Persistent objects



PMM uses access and hint information to allocate, locate, migrate and access data in the heterogeneous array of devices

The Persistent Memory Manager (PMM)

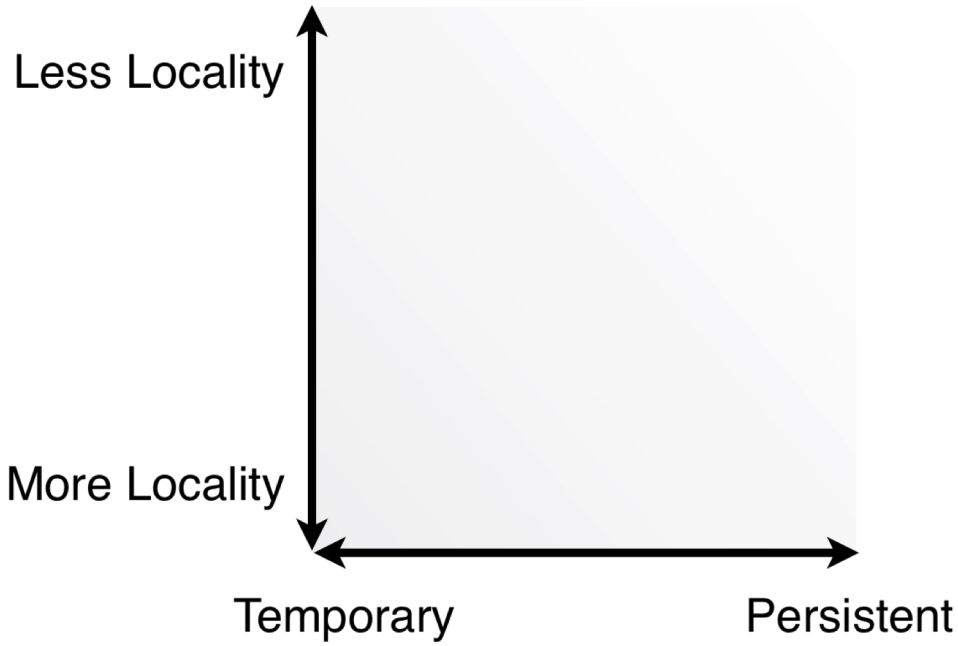
- Exposes a load/store interface to access persistent data
  - Applications can directly access persistent memory → no conversion, translation, location overhead for persistent data
- Manages data placement, location, persistence, security
  - To get the best of multiple forms of storage
- Manages metadata storage and retrieval
  - This can lead to overheads that need to be managed
- Exposes hooks and interfaces for system software
  - To enable better data placement and management decisions
- Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

Efficient Data Mapping among Heterogeneous Devices

- A persistent memory exposes a large, persistent address space
  - But it may use many different devices to satisfy this goal
  - From fast, low-capacity volatile DRAM to slow, high-capacity non-volatile HDD or Flash
  - And other NVM devices in between

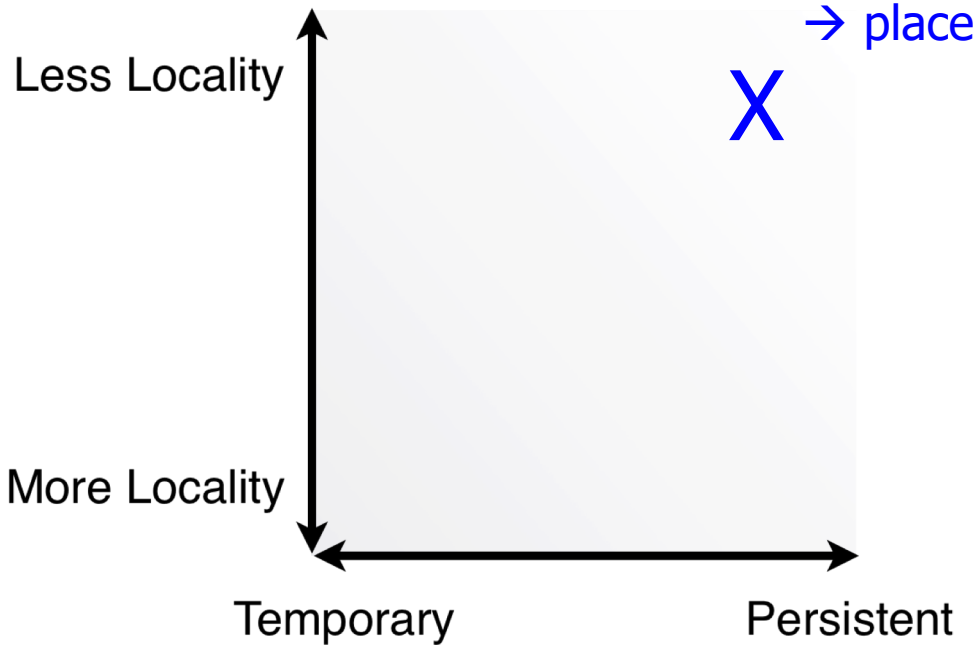
- Performance and energy can benefit from good placement of data among these devices
  - Utilizing the strengths of each device and avoiding their weaknesses, if possible
  - For example, consider two important application characteristics: locality and persistence

Efficient Data Mapping among Heterogeneous Devices

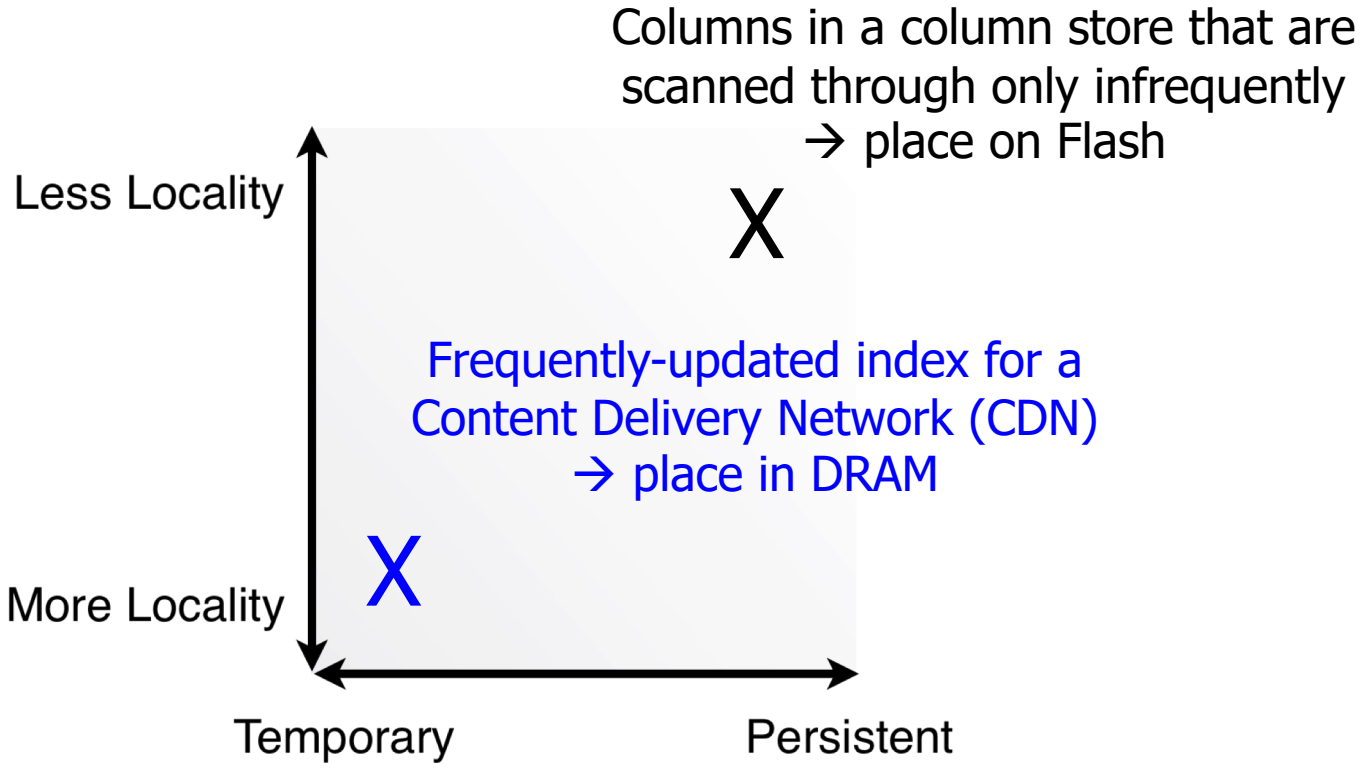


Efficient Data Mapping among Heterogeneous Devices

Columns in a column store that are scanned through only infrequently → place on Flash



Efficient Data Mapping among Heterogeneous Devices

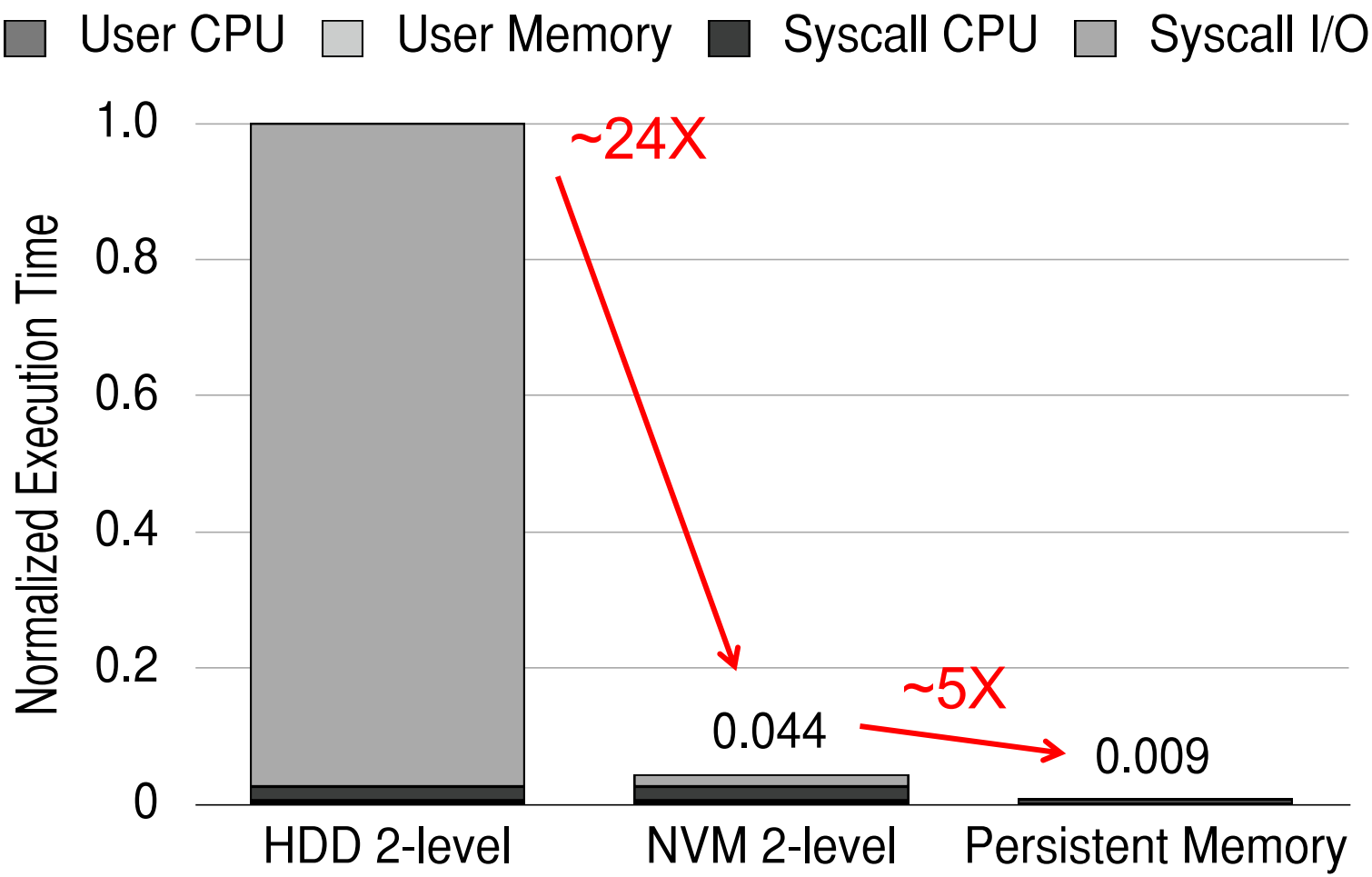


Applications or system software can provide hints for data placement

Evaluated Systems

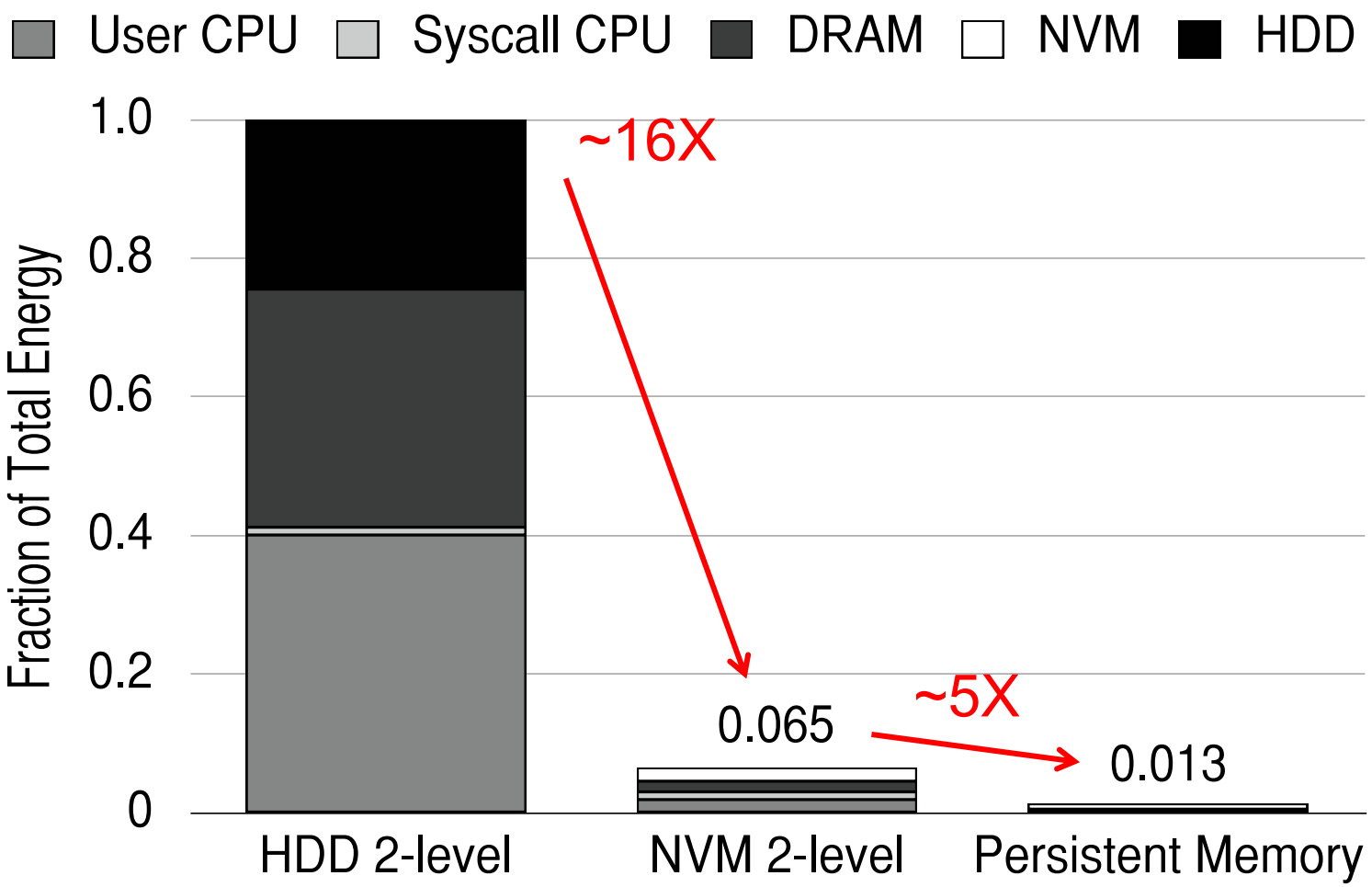
- HDD Baseline
  - Traditional system with volatile DRAM memory and persistent HDD storage
  - Overheads of operating system and file system code and buffering
- NVM Baseline (NB)
  - Same as HDD Baseline, but HDD is replaced with NVM
  - Still has OS/FS overheads of the two-level storage model
- Persistent Memory (PM)
  - Uses only NVM (no DRAM) to ensure full-system persistence
  - All data accessed using loads and stores
  - Does not waste time on system calls
  - Data is manipulated directly on the NVM device

Performance Benefits of a Single-Level Store



Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

Energy Benefits of a Single-Level Store



Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

On Persistent Memory Benefits & Challenges

- Justin Meza, Yixin Luo, Samira Khan, Jishen Zhao, Yuan Xie, and Onur Mutlu,
  "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory"
  *Proceedings of the 5th Workshop on Energy-Efficient Design (**WEED**)*, Tel-Aviv, Israel, June 2013. Slides (pptx) Slides (pdf)

A Case for Efficient Hardware/Software Cooperative Management of Storage and Memory

Justin Meza* Yixin Luo* Samira Khan*‡ Jishen Zhao† Yuan Xie†§ Onur Mutlu*
*Carnegie Mellon University †Pennsylvania State University ‡Intel Labs §AMD Research

Challenge and Opportunity

Combined
Memory & Storage

Challenge and Opportunity

A Unified Interface to
All Data

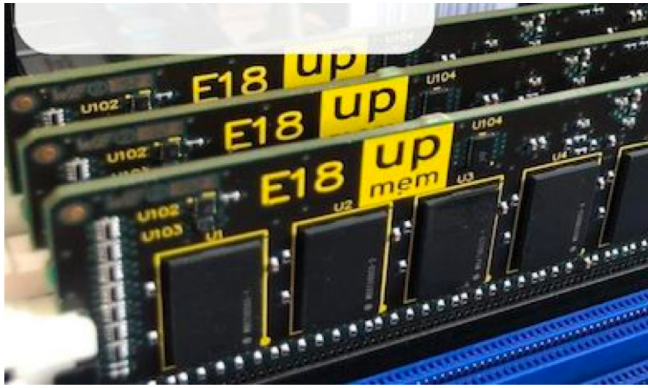
Intel Optane Persistent Memory (2019)

- Non-volatile main memory
- Based on 3D-XPoint Technology

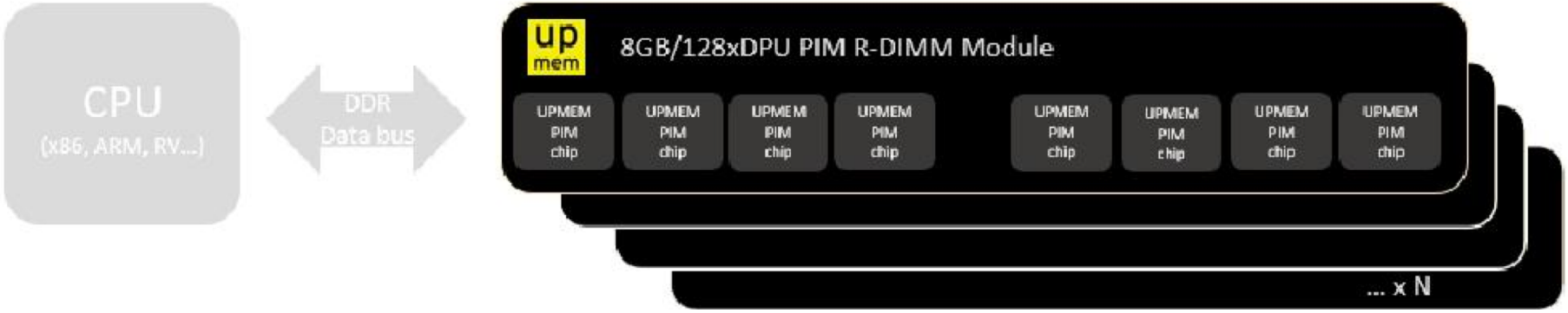
https://www.storagereview.com/intel_optane_dc_persistent_memory_module_pmm

UPMEM Processing-in-DRAM Engine (2019)

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



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



https://www.anandtech.com/show/14750/hot-chips-31-analysis-inmemory-processing-by-upmem
https://www.upmem.com/video-upmem-presenting-its-true-processing-in-memory-solution-hot-chips-2019/

One Key Challenge in Persistent Memory

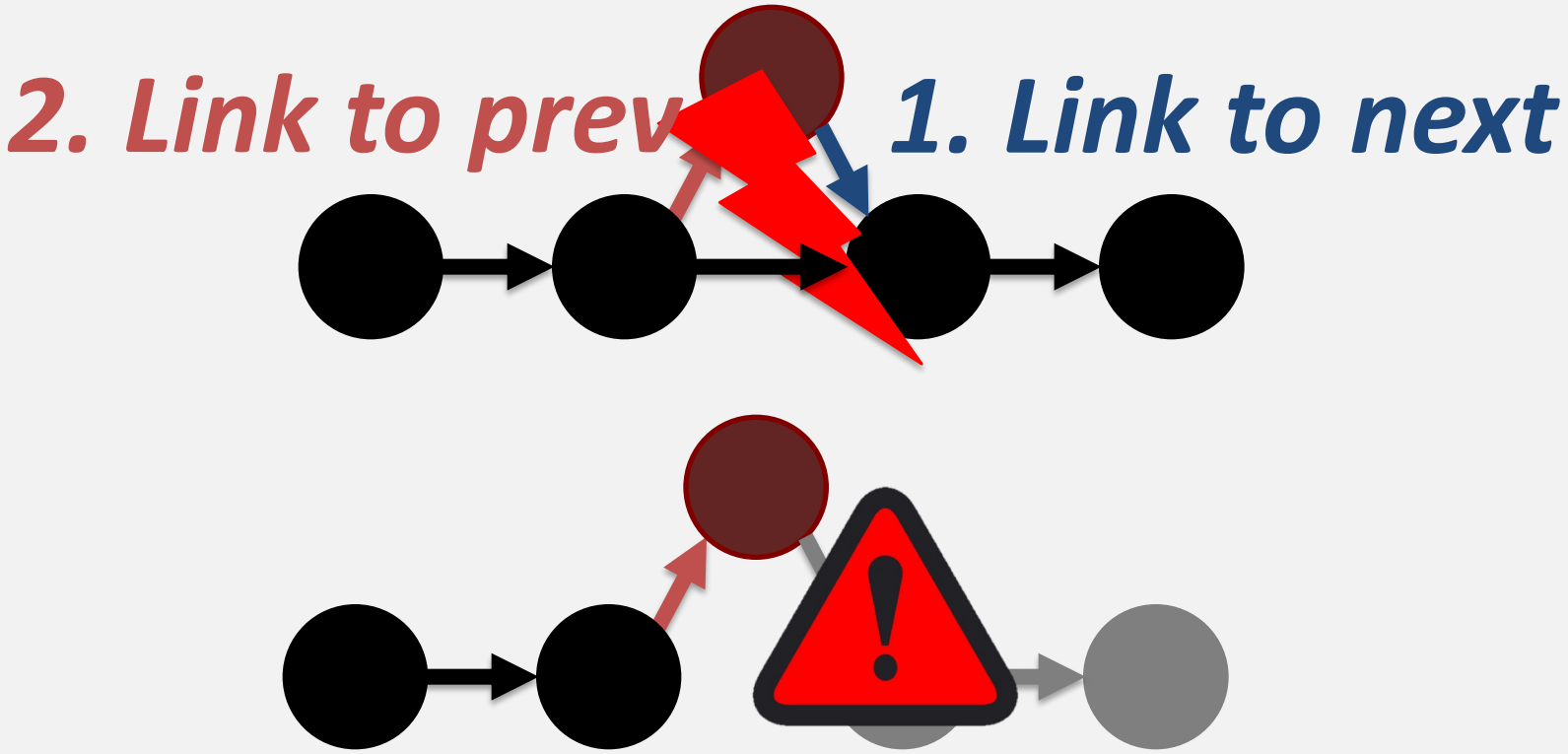
- How to ensure consistency of system/data if all memory is persistent?

- Two extremes
  - Programmer transparent: Let the system handle it
  - Programmer only: Let the programmer handle it

- Many alternatives in-between…

CRASH CONSISTENCY PROBLEM

Add a node to a linked list



System crash can result in inconsistent memory state

CURRENT SOLUTIONS

Explicit interfaces to manage consistency

– **NV-Heaps** [ASPLOS'11], **BPFS** [SOSP'09], **Mnemosyne** [ASPLOS'11]

```
AtomicBegin {
      Insert a new node;
} AtomicEnd;
```

Limits adoption of NVM

Have to rewrite code with clear partition between volatile and non-volatile data

Burden on the programmers

CURRENT SOLUTIONS

Explicit interfaces to manage consistency

– **NV-Heaps** [ASPLOS'11], **BPFS** [SOSP'09], **Mnemosyne** [ASPLOS'11]

Example Code

update a node in a persistent hash table

```
void hashtable_update(hashtable_t* ht,
                      void *key, void *data)
{
    list_t* chain = get_chain(ht, key);
    pair_t* pair;
    pair_t updatePair;
    updatePair.first = key;
    pair = (pair_t*) list_find(chain,
                               &updatePair);
    pair->second = data;
}
```

CURRENT SOLUTIONS

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key,
void*data){
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*) TMLIST_FIND(chain,
                               &updatePair);
  pair->second = data;
}
```

CURRENT SOLUTIONS

Manual declaration of persistent components

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key,
void*data){
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*) TMLIST_FIND(chain,
                               &updatePair);
  pair->second = data;
}
```

CURRENT SOLUTIONS

Manual declaration of persistent components

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key,
void*data){
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*) TMLIST_FIND(chain,
                               &updatePair);
  pair->second = data;
}
```

Need a new implementation

CURRENT SOLUTIONS

Manual declaration of persistent components

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key,
void*data){
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*) TMLIST_FIND(chain,
                               &updatePair);
  pair->second = data;
}
```

Need a new implementation

Third party code can be inconsistent

CURRENT SOLUTIONS

Manual declaration of persistent components

```
void TMhashtable_update(TMARCGDECL
hashtable_t* ht, void *key,
void*data){
  list_t* chain = get_chain(ht, key);
  pair_t* pair;
  pair_t updatePair;
  updatePair.first = key;
  pair = (pair_t*)TMLIST_FIND(chain,
                              &updatePair);
  pair->second = data;
}
```

Need a new implementation

Prohibited Operation `=` **Third party code can be inconsistent**

Burden on the programmers

OUR APPROACH: ThyNVM

Goal:
Software transparent consistency in persistent memory systems

Key Idea:
Periodically checkpoint state; recover to previous checkpt on crash

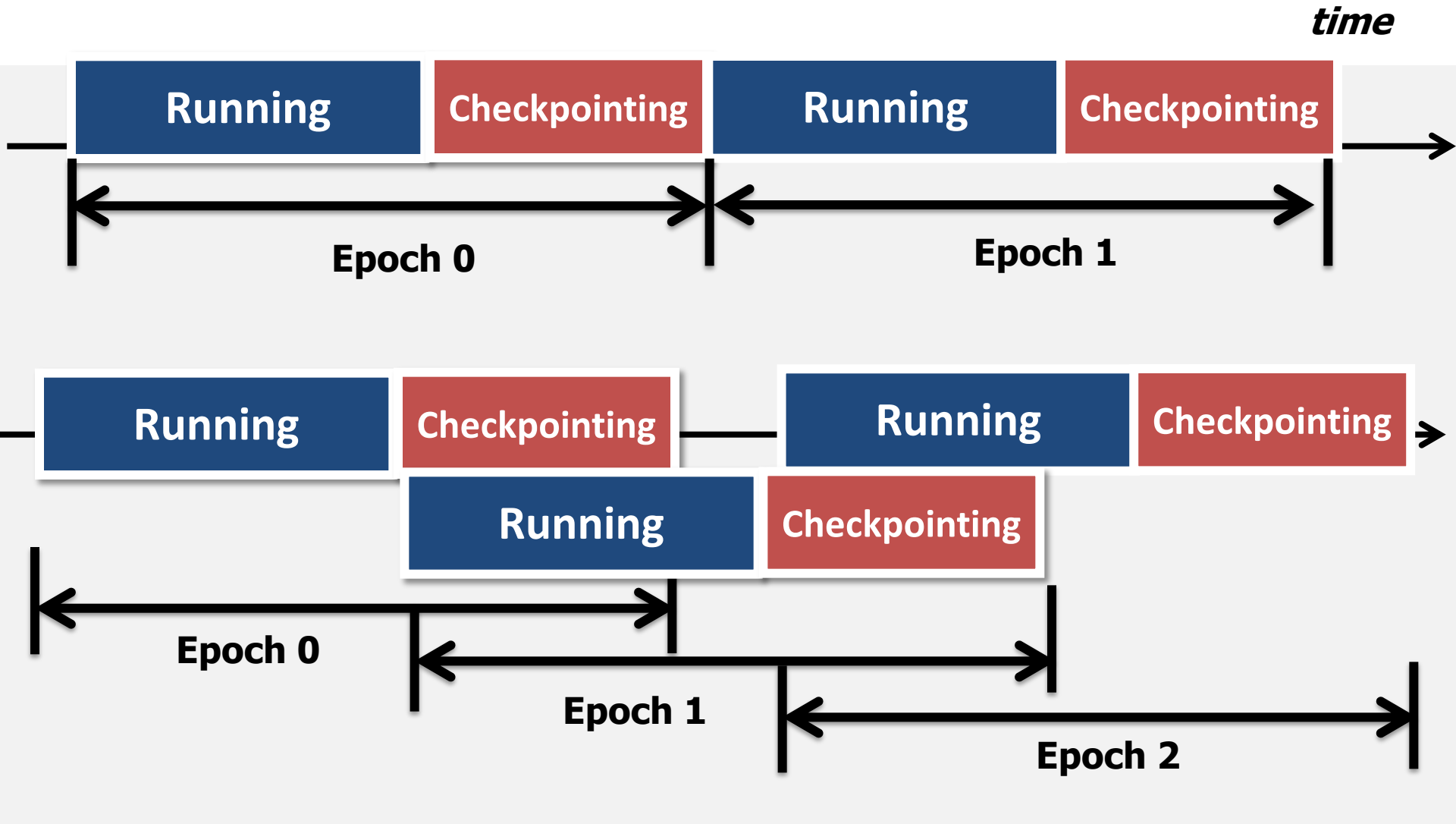
ThyNVM: Summary

A new hardware-based checkpointing mechanism

- **Checkpoints** at *multiple granularities* to reduce both checkpointing latency and metadata overhead
- **Overlaps** *checkpointing* and *execution to* reduce checkpointing latency
- **Adapts** to *DRAM and NVM* characteristics

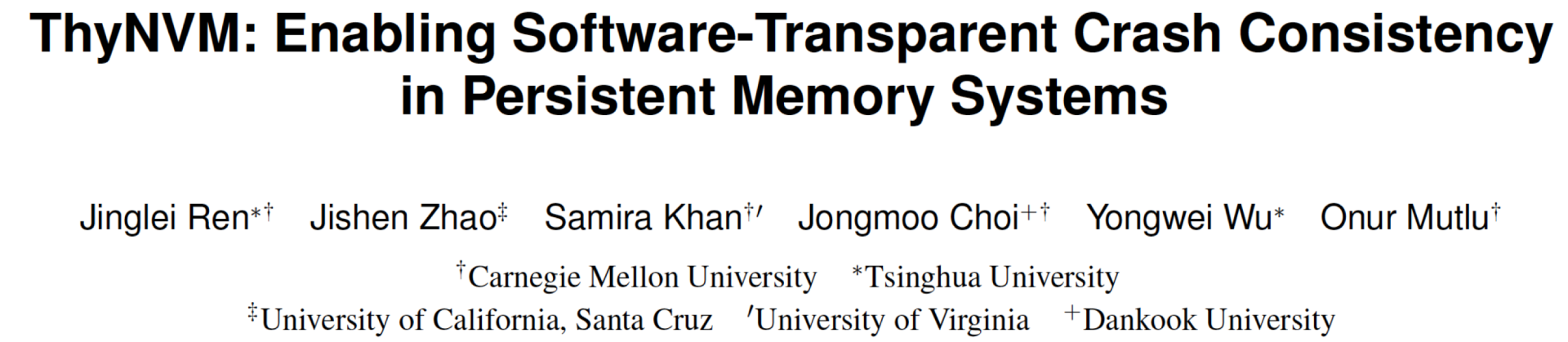
Performs within 4.9% of an *idealized DRAM* with zero cost consistency

2. OVERLAPPING CHECKPOINTING AND EXECUTION



More About ThyNVM

- Jinglei Ren, Jishen Zhao, Samira Khan, Jongmoo Choi, Yongwei Wu, and Onur Mutlu,
  "ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems"
  *Proceedings of the 48th International Symposium on Microarchitecture (**MICRO**)*, Waikiki, Hawaii, USA, December 2015.
  [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]
  [Source Code]



ThyNVM: Enabling Software-Transparent Crash Consistency in Persistent Memory Systems

Jinglei Ren*† Jishen Zhao‡ Samira Khan†′ Jongmoo Choi+† Yongwei Wu* Onur Mutlu†

†Carnegie Mellon University *Tsinghua University
‡University of California, Santa Cruz ′University of Virginia +Dankook University

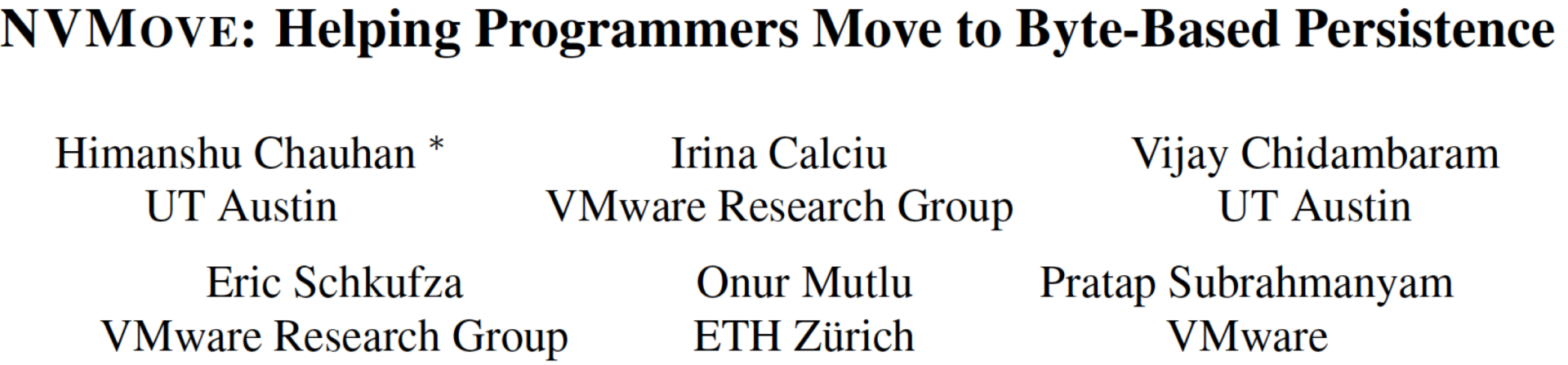
Another Key Challenge in Persistent Memory

Programming Ease

to Exploit Persistence

Tools/Libraries to Help Programmers

- Himanshu Chauhan, Irina Calciu, Vijay Chidambaram, Eric Schkufza, Onur Mutlu, and Pratap Subrahmanyam, **"NVMove: Helping Programmers Move to Byte-Based Persistence"**
  *Proceedings of the 4th Workshop on Interactions of NVM/Flash with Operating Systems and Workloads (**INFLOW**)*, Savannah, GA, USA, November 2016.
  [Slides (pptx) (pdf)]



NVMove: Helping Programmers Move to Byte-Based Persistence

Himanshu Chauhan * — UT Austin
Irina Calciu — VMware Research Group
Vijay Chidambaram — UT Austin
Eric Schkufza — VMware Research Group
Onur Mutlu — ETH Zürich
Pratap Subrahmanyam — VMware

Consistency Support for Persistent Memory

- Youyou Lu, Jiwu Shu, Long Sun, and Onur Mutlu,
  "Loose-Ordering Consistency for Persistent Memory"
  *Proceedings of the 32nd IEEE International Conference on Computer Design (**ICCD**)*, Seoul, South Korea, October 2014.
  [Slides (pptx) (pdf)]
  [Erratum]

Loose-Ordering Consistency for Persistent Memory

Youyou Lu †, Jiwu Shu † §, Long Sun † and Onur Mutlu ‡
†Department of Computer Science and Technology, Tsinghua University, Beijing, China
§State Key Laboratory of Computer Architecture, Institute of Computing Technology,
Chinese Academy of Sciences, Beijing, China
‡Computer Architecture Laboratory, Carnegie Mellon University, Pittsburgh, PA, USA
luyy09@mails.tsinghua.edu.cn, shujw@tsinghua.edu.cn, sun-l12@mails.tsinghua.edu.cn, onur@cmu.edu

Another Key Challenge in Persistent Memory

Security and Data Privacy Issues

Security and Privacy Issues of NVM

- Endurance problems → Wearout attacks
- Hybrid memories → Performance attacks
- Data not erased after power-off → Privacy breaches

Conclusion

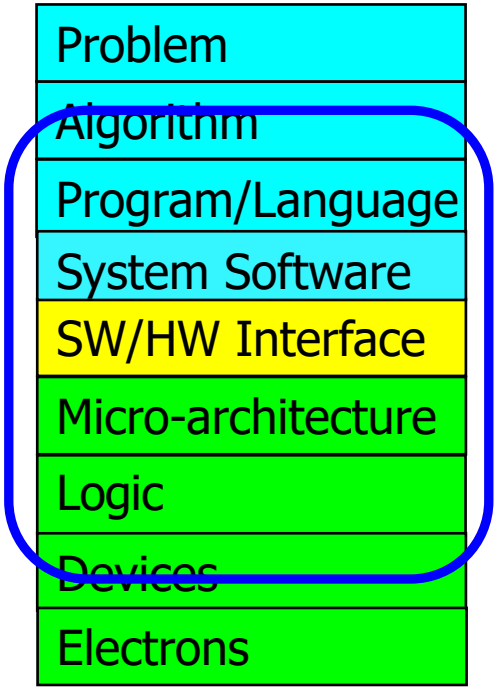
The Future of Emerging Technologies is Bright

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

Can enable:

- Orders of magnitude improvements

- New applications and computing systems



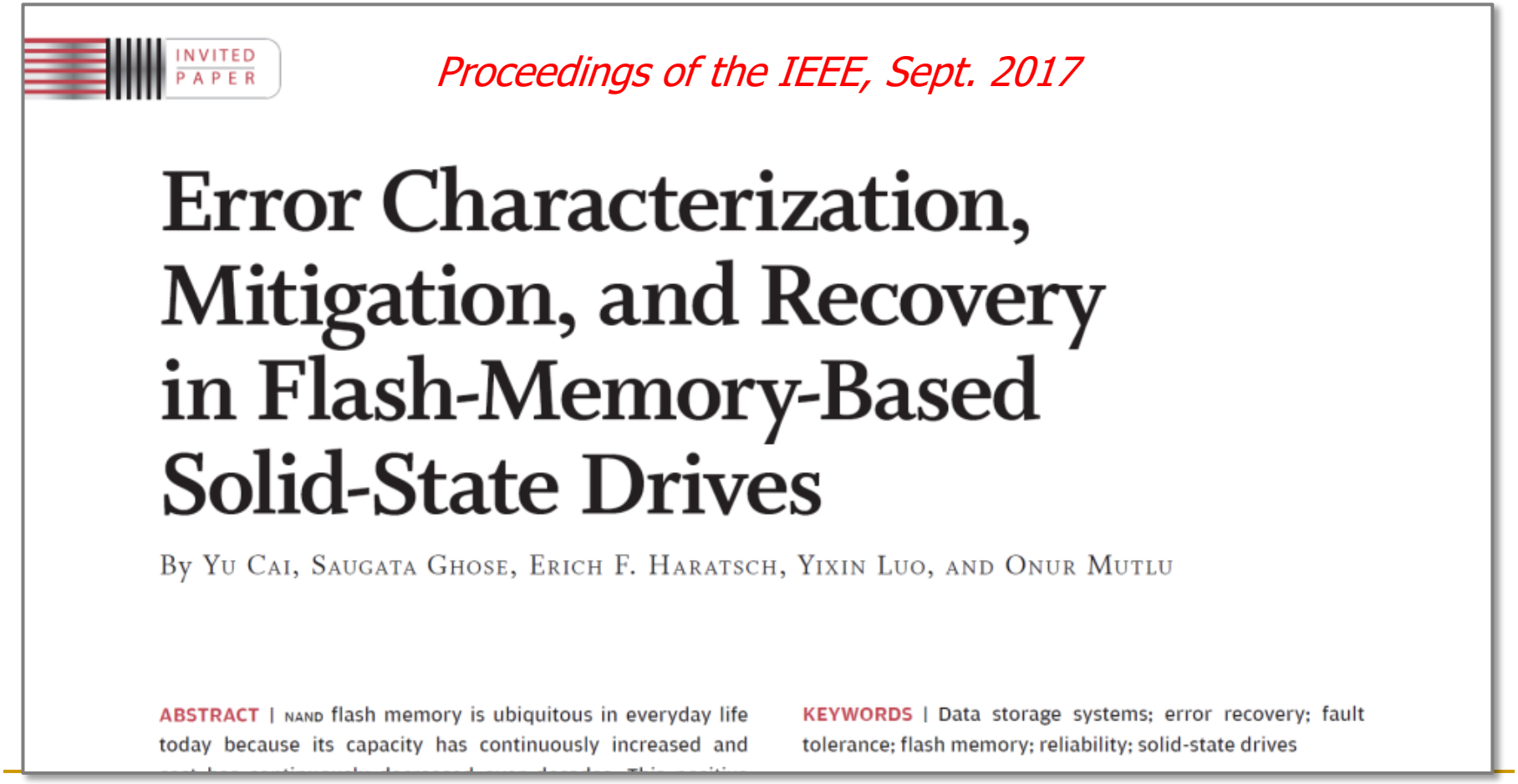
Yet, we have to

- Think across the stack

- Design enabling systems

If In Doubt, Refer to Flash Memory

- 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

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

ABSTRACT | NAND flash memory is ubiquitous in everyday life today because its capacity has continuously increased and

KEYWORDS | Data storage systems; error recovery; fault tolerance; flash memory; reliability; solid-state drives

https://arxiv.org/pdf/1706.08642

Many Research & Design Opportunities

- Enabling completely persistent memory
- Computation in/using NVM based memories
- Hybrid memory systems
- Security and privacy issues in persistent memory
- Reliability and endurance related problems
- Virtual memory systems for NVM (e.g., virtual block interface)
- ...

GenPIP

In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

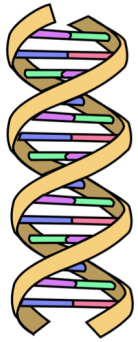
Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina,
Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, Onur Mutlu

SAFARI

ETH zürich

Overview: Genome Analysis

- **Genome analysis:** Enables us to determine the order of the DNA sequence in an organism's genome
  - Plays an important role in
    - Personalized medicine
    - Outbreak tracing
    - Understanding of evolution
    - …



- Modern genome sequencing machines extract smaller randomized fragments of the original DNA sequence, known as ***reads***
  - **Oxford Nanopore Technologies (ONT)**:

    A widely-used sequencing technology
    - Portable sequencing devices
    - High-throughput
    - Cheap

ONT sequencing device
[forbes.com]

Overview: Two Limitations

Multiple steps in genome analysis

Large data movement between multiple steps

A lot of **wasted computation** done on data that is later discovered to be **useless**

Overview: GenPIP

- **GenPIP:** A fast and energy-efficient **in-memory** acceleration system for the Genome analysis PIPeline via **tight integration of genome analysis steps**

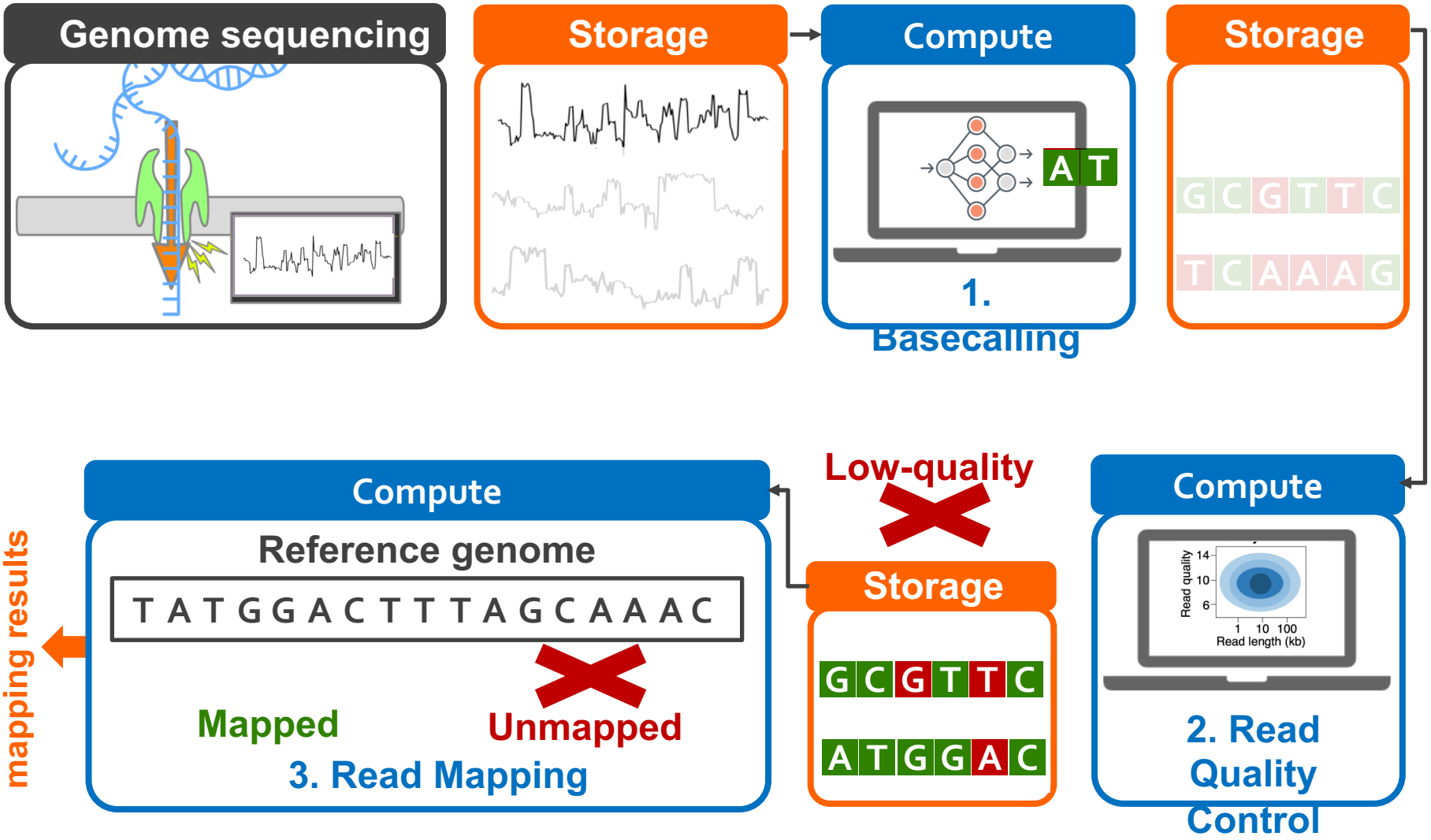
- **GenPIP** has two key techniques
  - **Chunk-based pipeline (CP)**
    - **Provides fine-grained collaboration** of genome analysis steps
  - **Early rejection (ER)**
    - **Timely stops the execution on useless data** by predicting which reads will not be useful

- **GenPIP** outperforms state-of-the-art software & hardware solutions using **CPU**, **GPU**, and **optimistic PIM** by **41.6×**, **8.4x**, and **1.4x**, respectively.

Outline

- Background and Motivation
- GenPIP: Tight Integration of Genome Analysis Steps
  - Chunk-based Pipeline (CP)
  - Early Rejection (ER)
- GenPIP Implementation
- Evaluation
- Conclusion

Genome Analysis Pipeline



Limitation 1: Large Data Movement

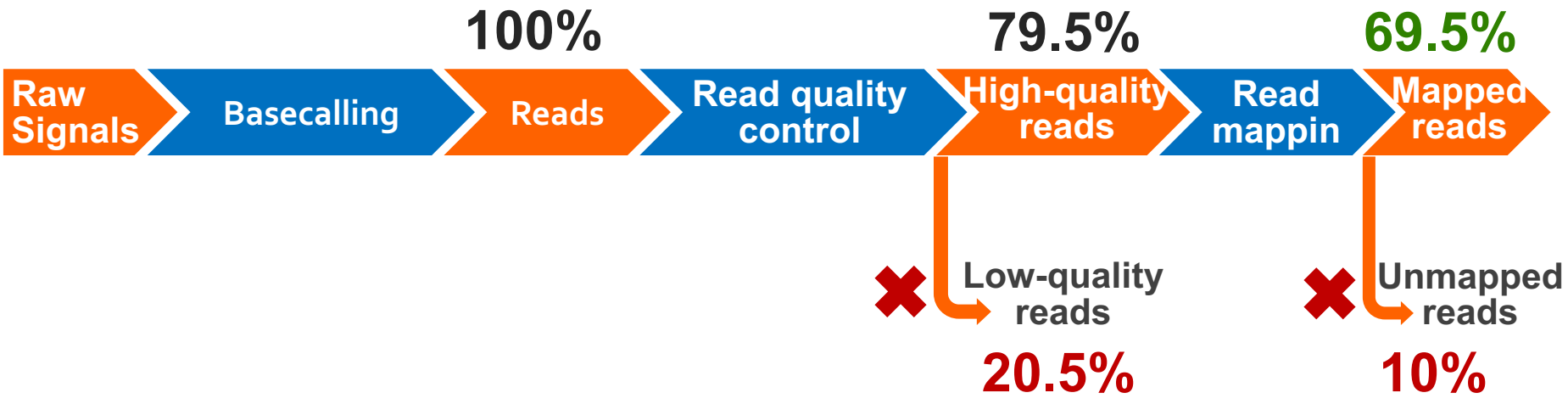
- Using a human dataset in [NC'19] as an example:

Large data movement between genome analysis steps

[NC'19] Rory Bowden, Robert W Davies, Andreas Heger, Alistair T Pagnamenta, Mariateresa de Cesare, Laura E Oikkonen, Duncan Parkes, Colin Freeman, Fatima Dhalla, Smita Y Patel, et al. Sequencing of human genomes with nanopore technology. Nature Communications, 2019.

Limitation 2: Wasted Computation

- Using a human dataset in [NC'19] as an example:



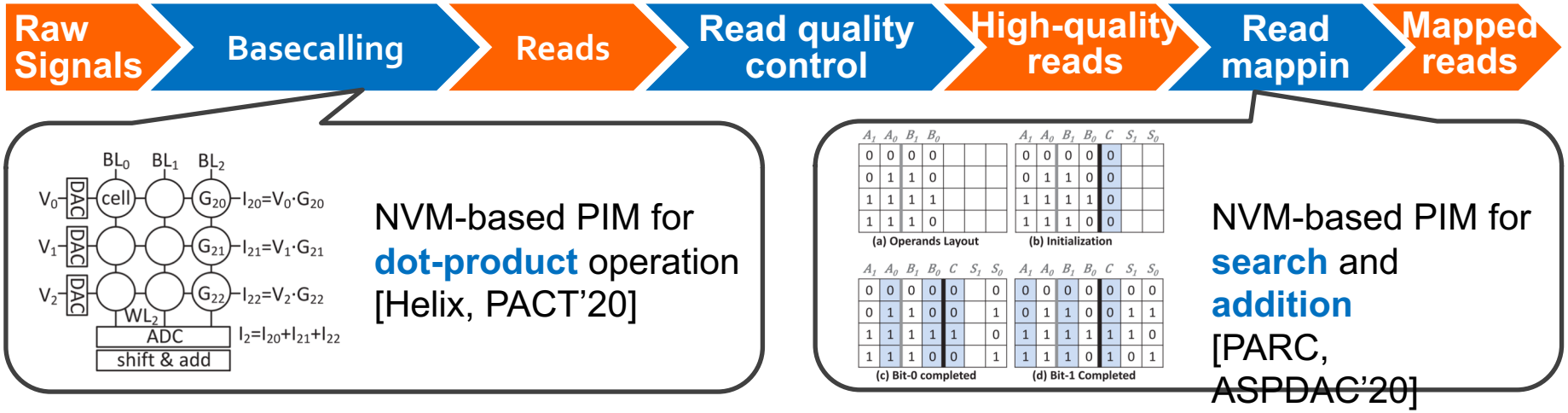
A considerable amount of computation on useless data due to

- **Low-quality reads**
- **Unmapped reads**

[NC'19] Rory Bowden, Robert W Davies, Andreas Heger, Alistair T Pagnamenta, Mariateresa de Cesare, Laura E Oikkonen, Duncan Parkes, Colin Freeman, Fatima Dhalla, Smita Y Patel, et al. Sequencing of human genomes with nanopore technology. Nature Communications, 2019.

State-of-the-art Works

- NVM-based PIM is an efficient technique to reduce data movement by processing data using or near memory



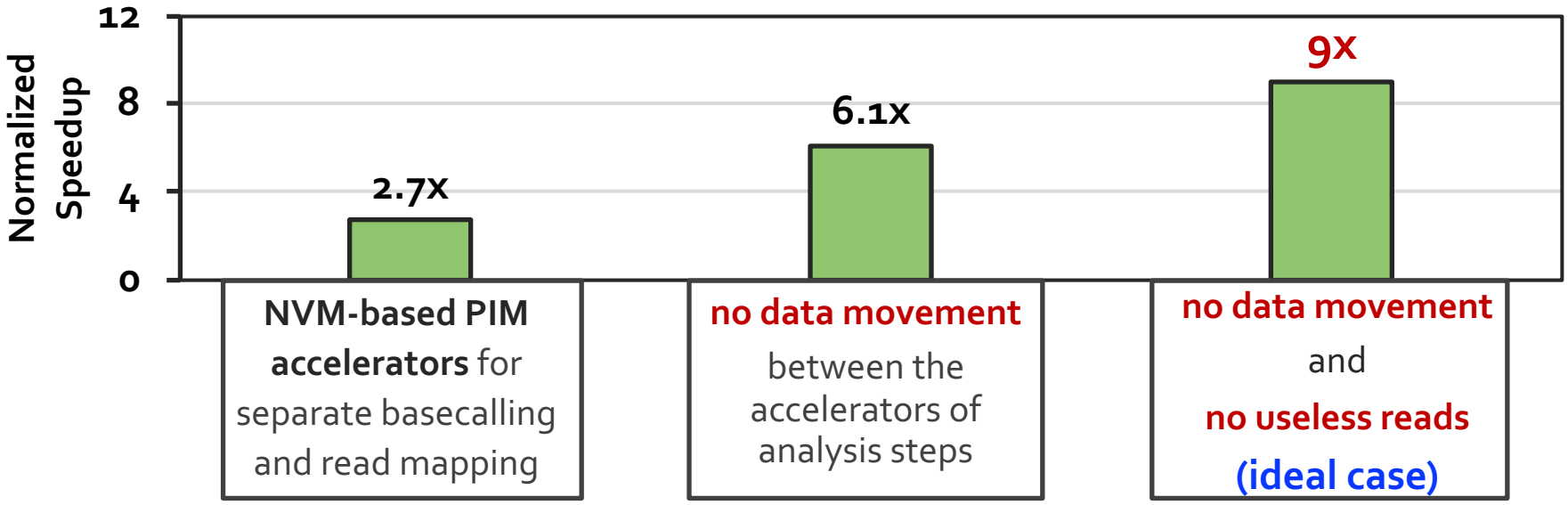
- Reduce the data movement in a single genome analysis step
- Exacerbate the data movement overhead between analysis steps

No prior work tackles data movement between analysis steps and reduces useless computation

Goal and Opportunities

Goal: Efficiently accelerate the entire genome analysis pipeline while **minimizing data movement and useless computation**

- We perform a study to quantify potential performance benefits
  - Results are normalized to the performance of GPU



Outline

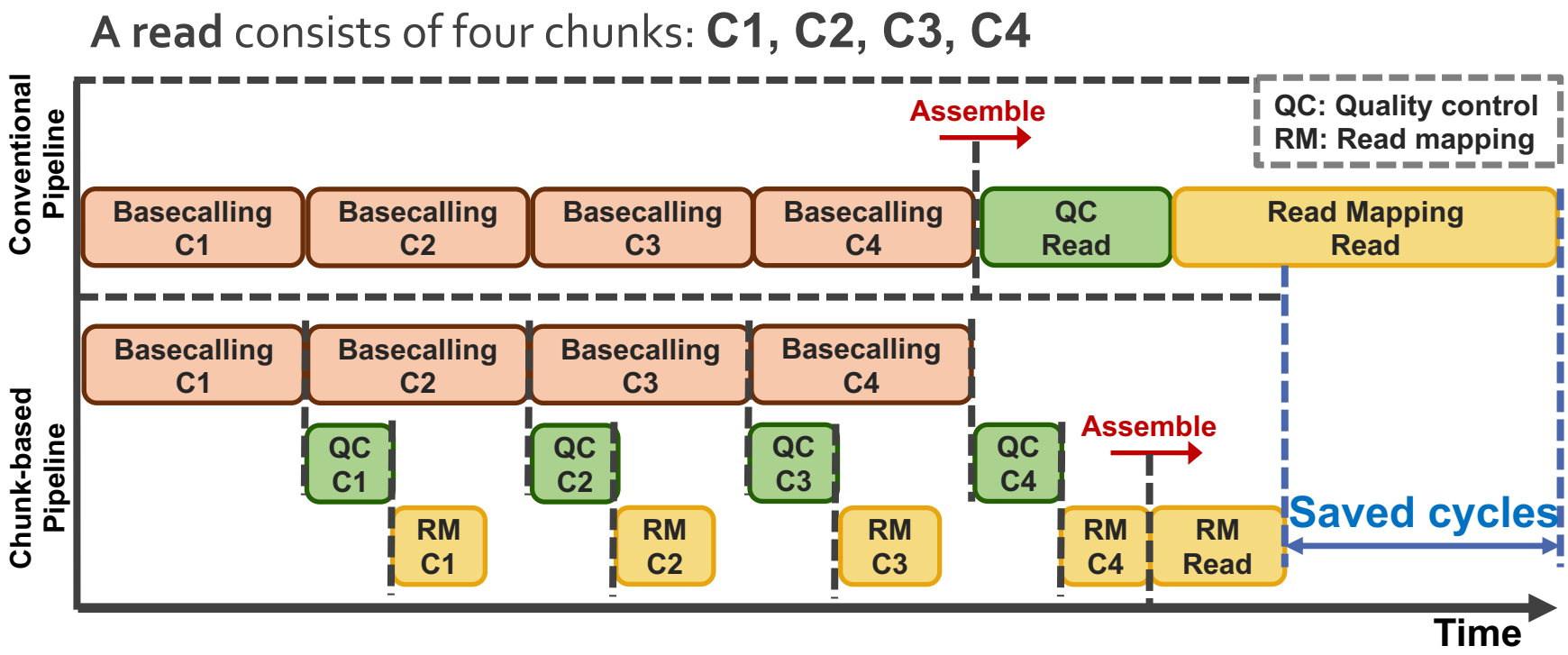
- Background and Motivation
- **GenPIP: Tight Integration of Genome Analysis Steps**
  - **Chunk-based Pipeline (CP)**
  - **Early Rejection (ER)**
- GenPIP Implementation
- Evaluation
- Conclusion

GenPIP

- ***First* holistic in-memory accelerator for the genome analysis pipeline**, including basecalling, read quality control, and read mapping steps
- **GenPIP** has two key techniques
  - **Chunk-based Pipeline (CP)**
    - **Enables fine-grained pipelining** of genome analysis steps
    - Processes reads at **chunk** granularity (i.e., a subsequence; 300 bases)
  - **Early Rejection (ER)**

Chunk-based Pipeline (CP)

- CP **increases parallelism** by overlapping the execution of different steps at chuck granularity
- CP **reduces intermediate data** by computing on data as soon as data is generated
- CP **provides opportunities for ER** by analyzing a read at chunk granularity

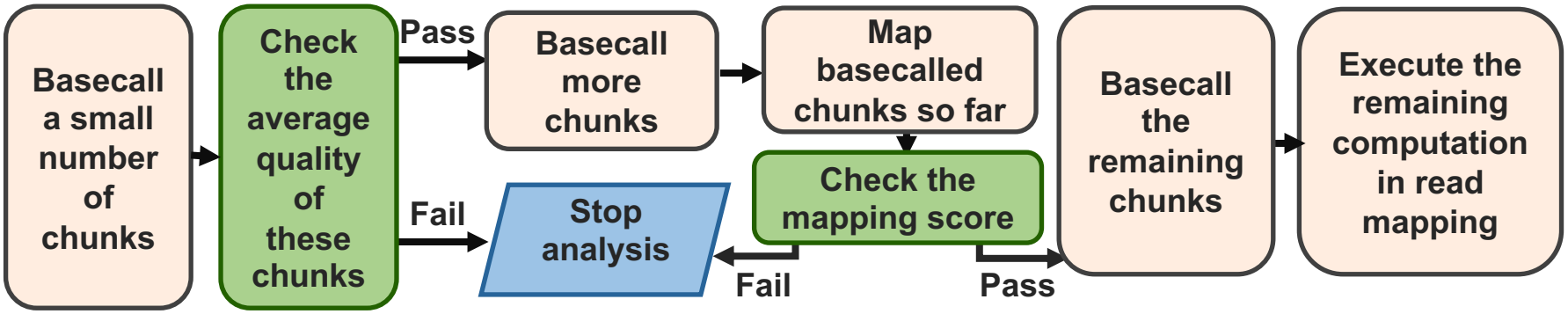


GenPIP

- ***First* holistic in-memory accelerator for the genome analysis pipeline**, including basecalling, read quality control, and read mapping steps
- **GenPIP** has two key techniques
  - Chunk-based Pipeline (CP)
    - **Enables fine-grained collaboration** of genome analysis steps by processing reads at chunk granularity (i.e., a subsequence of a read, e.g., 300 bases)
  - **Early Rejection (ER)**
    - **Stops the execution on useless reads as early as possible** by using a small number of chunks to predict the usefulness of a read

Early Rejection (ER)

- **Predict and eliminate** low-quality and unmapped reads from the genome analysis pipeline **as early as possible**



- **Early-Rejection based on chunk quality scores (ER-QSR)**
  - Predict **low-quality** reads using chunk quality scores

- **Early-Rejection based on chunk mapping scores (ER-CMR)**
  - Predict **unmapped reads** using chunk mapping scores

Implementation of CP and ER

CP and ER can be applied on different systems, e.g., CPU, GPU, and PIM

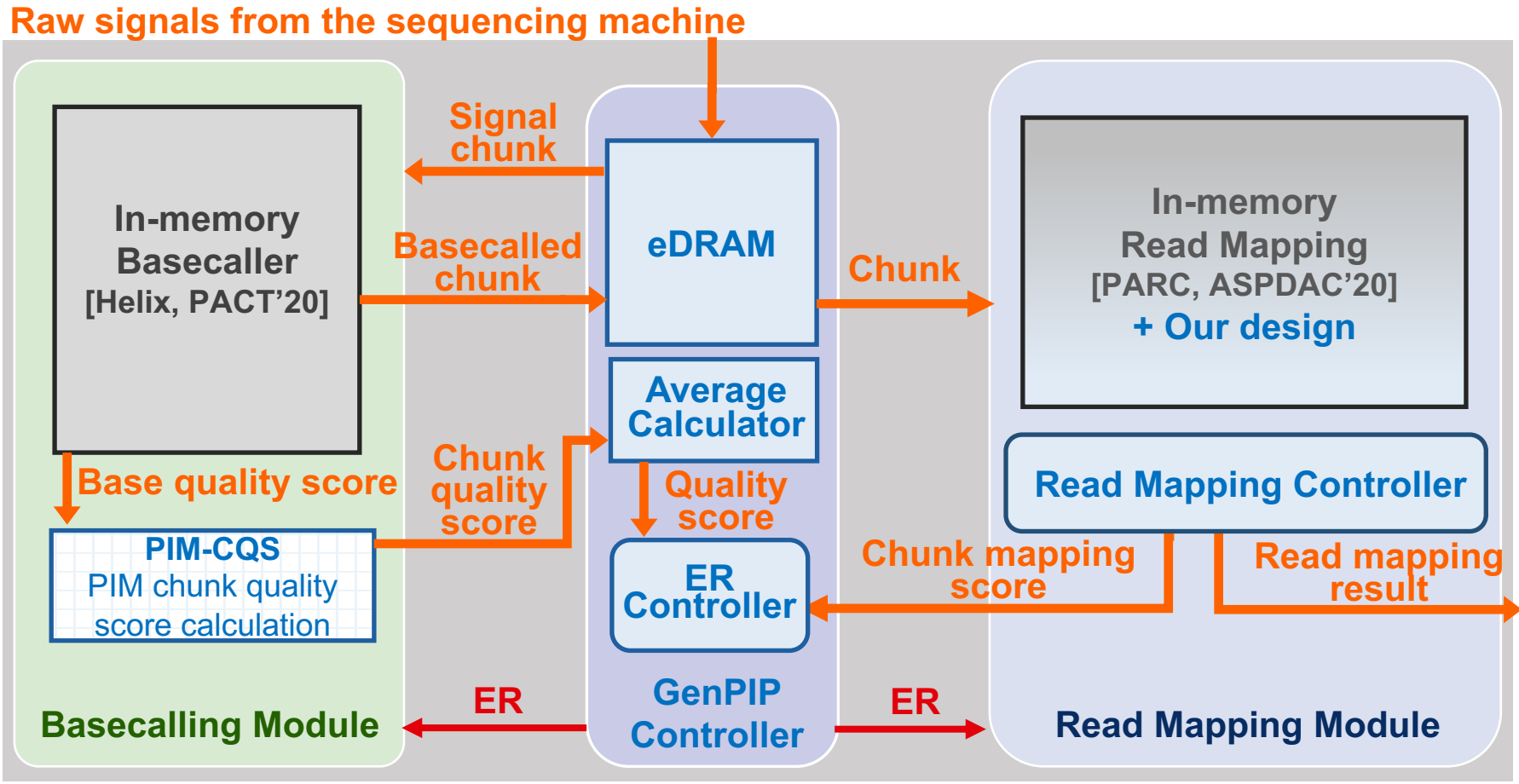
We implement CP and ER using PIM since PIM is more efficient to reduce the data movement between genome analysis steps

We also apply CP and ER on CPU and GPU baselines and observe speedup and energy savings

Outline

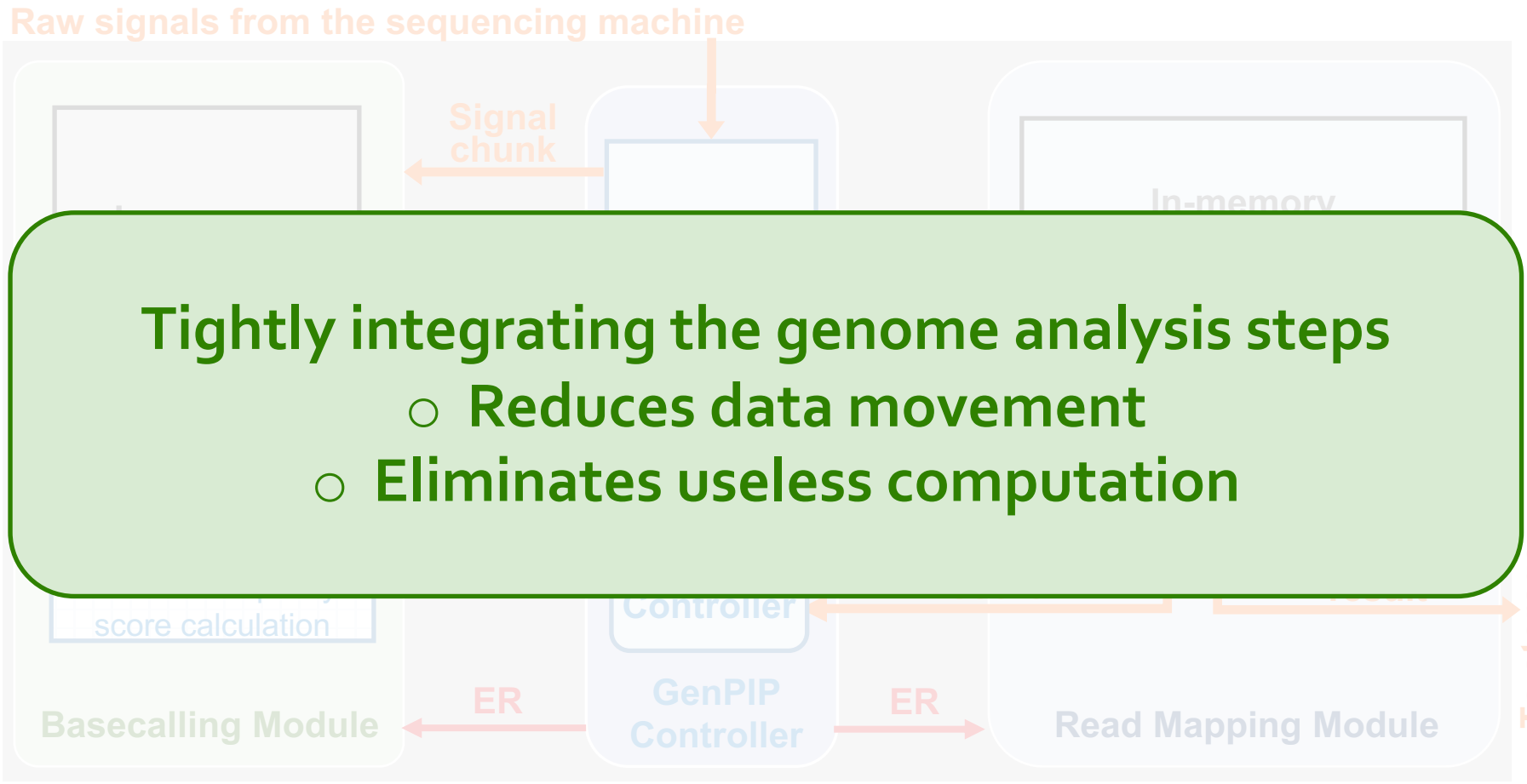
- Background and Motivation
- GenPIP: Tight Integration of Genome Analysis Steps
  - Chunk-based Pipeline (CP)
  - Early Rejection (ER)
- **GenPIP Implementation**
- Evaluation
- Conclusion

GenPIP Implementation



https://arxiv.org/pdf/2209.08600.pdf

GenPIP Implementation



Tightly integrating the genome analysis steps

- **Reduces data movement**
- **Eliminates useless computation**

Outline

- Background and Motivation
- GenPIP: Tight Integration of Genome Analysis Steps
  - Chunk-based Pipeline (CP)
  - Early Rejection (ER)
- GenPIP Implementation
- **Evaluation**
- Conclusion

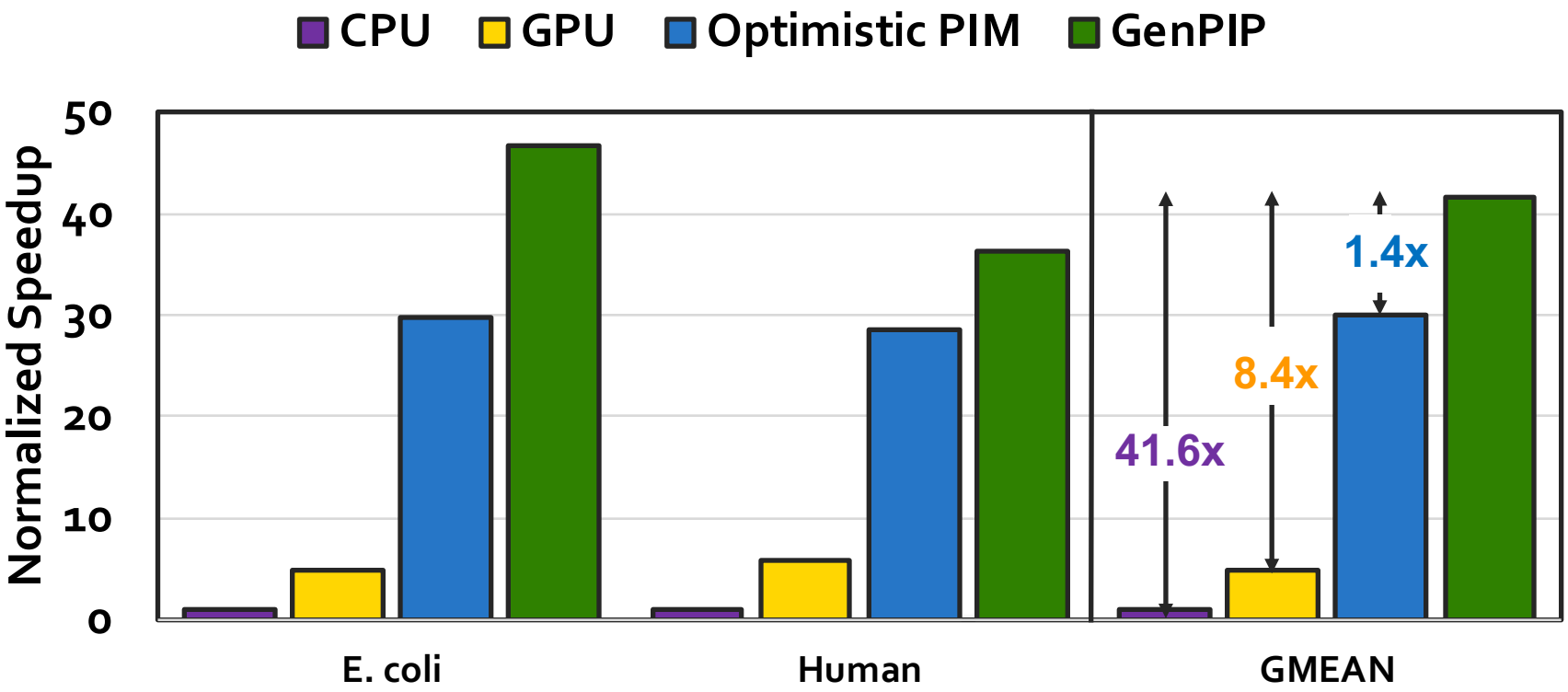
Evaluation Methodology

- **Performance, Area and Power Analysis:**
  - Simulation via Verilog HDL, NVSim [TCAD'12], and CACTI 6.5 [MICRO'07]
  - See methodology in the paper for more

- **Baselines:**
  - **CPU** (Intel Xeon Gold 5118 CPU)
  - **GPU** (NVIDIA GeForce RTX 2080 Ti GPU)
  - **Optimistic integration of two PIM accelerators (Helix [PACT'20] and PARC [ASP-DAC'20])**
    - Assumes **no data movement** between steps
    - Assumes intermediate data causes no overhead

- **Datasets:**
  - **E. coli** (http://lab.loman.net/2016/07/30/nano pore- r9- data- release/)
  - **Human** (https://www.ebi.ac.uk/ena/browser/view/PRJEB30620)

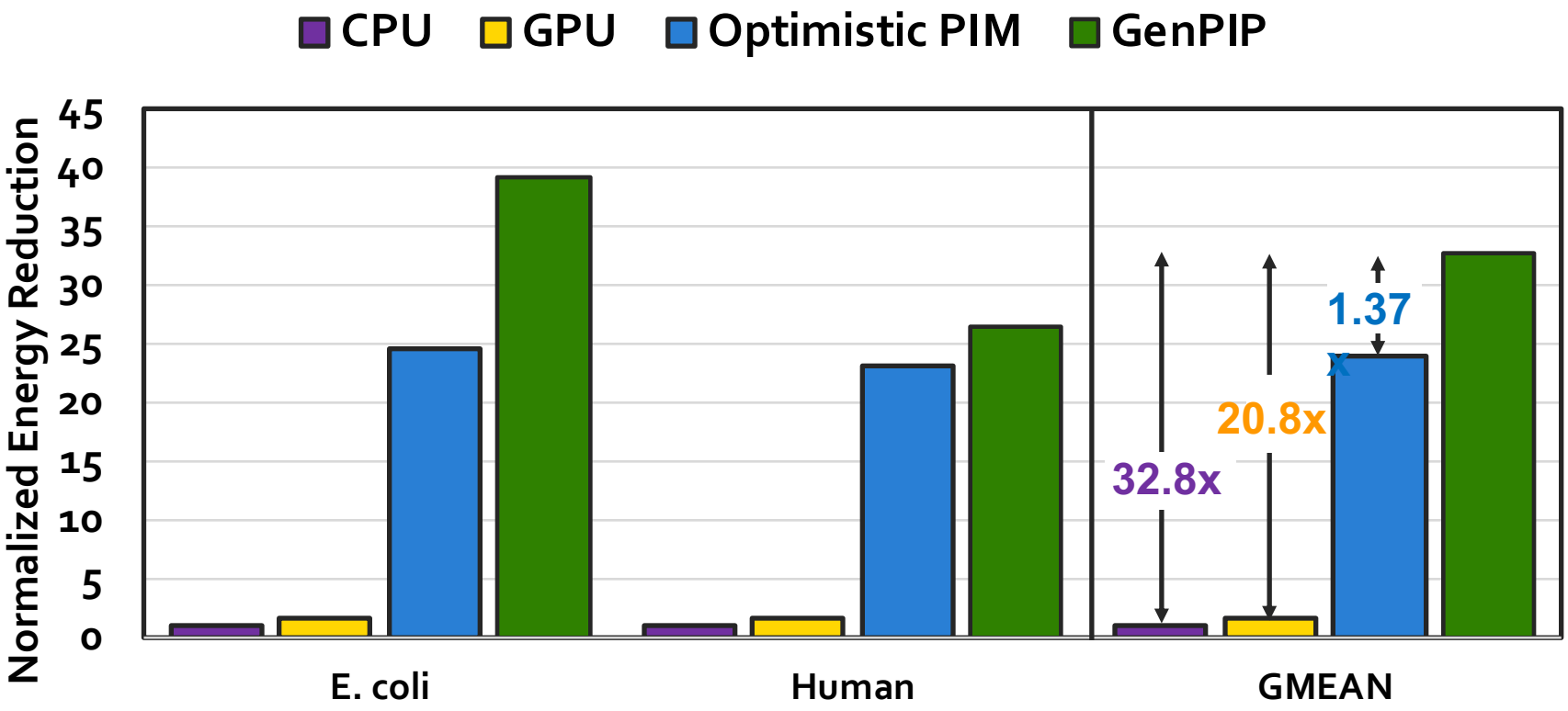
Key Results – Performance



GenPIP provides 41.6x, 8.4x, and 1.4x speedup over CPU, GPU, and optimistic PIM

Both CP and ER are critical to the speedup

Key Results – Energy Efficiency



GenPIP provides 32.8x, 20.8x, and 1.37x energy savings over CPU, GPU, and optimistic PIM

ER is especially critical to the energy efficiency

More in the Paper

GenPIP: In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

Haiyu Mao[1] Mohammed Alser[1] Mohammad Sadrosadati[1] Can Firtina[1] Akanksha Baranwal[1]
Damla Senol Cali[2] Aditya Manglik[1] Nour Almadhoun Alserr[1] Onur Mutlu[1]
[1]*ETH Zürich* [2]*Bionano Genomics*

https://arxiv.org/pdf/2209.08600.pdf

More in the Paper

- Details of **CP and ER**
- Detailed **GenPIP** implementation
  - GenPIP controller
  - Early rejection implementation
  - In-memory seeding accelerator
- Results of **applying CP and ER in CPU and GPU**
- **Sensitivity analysis** on the number of sampled chunks used for ER
- **Area and power** analysis

Outline

- Background and Motivation
- GenPIP: Tight Integration of Genome Analysis Steps
  - Chunk-based Pipeline (CP)
  - Early Rejection (ER)
- GenPIP Implementation
- Evaluation
- **Conclusion**

Conclusion

- **Problem:** The genome analysis pipeline has **large data movement** between genome analysis steps and a significant amount of **wasted computation on useless data**

- **Goal**: **Tightly integrate genome analysis steps** to reduce the data movement between steps and eliminate computation on useless data

- **GenPIP**: **The *first* in-memory genome analysis accelerator** that **tightly integrates** genome analysis steps
- **GenPIP** has two key techniques
  - **A chunk-based pipeline**
  - **A new early-rejection technique**

- **GenPIP outperforms** state-of-the-art software & hardware solutions using **CPU**, **GPU**, and **optimistic PIM** by **41.6×**, **8.4x**, and **1.4x**, respectively.

GenPIP

In-Memory Acceleration of Genome Analysis via Tight Integration of Basecalling and Read Mapping

Haiyu Mao, Mohammed Alser, Mohammad Sadrosadati, Can Firtina,
Akanksha Baranwal, Damla Senol Cali, Aditya Manglik, Nour Almadhoun Alserr, Onur Mutlu

SAFARI

ETH zürich

Computer Architecture

Lecture 15: Emerging Memory Technologies

Dr. Haiyu Mao
Prof. Onur Mutlu
ETH Zürich

Fall 2023

16 November 2023