Rethinking Memory System Design

(and the Platforms We Design Around It)

Onur Mutlu
onur.mutlu@inf.ethz.ch
https://people.inf.ethz.ch/omutlu

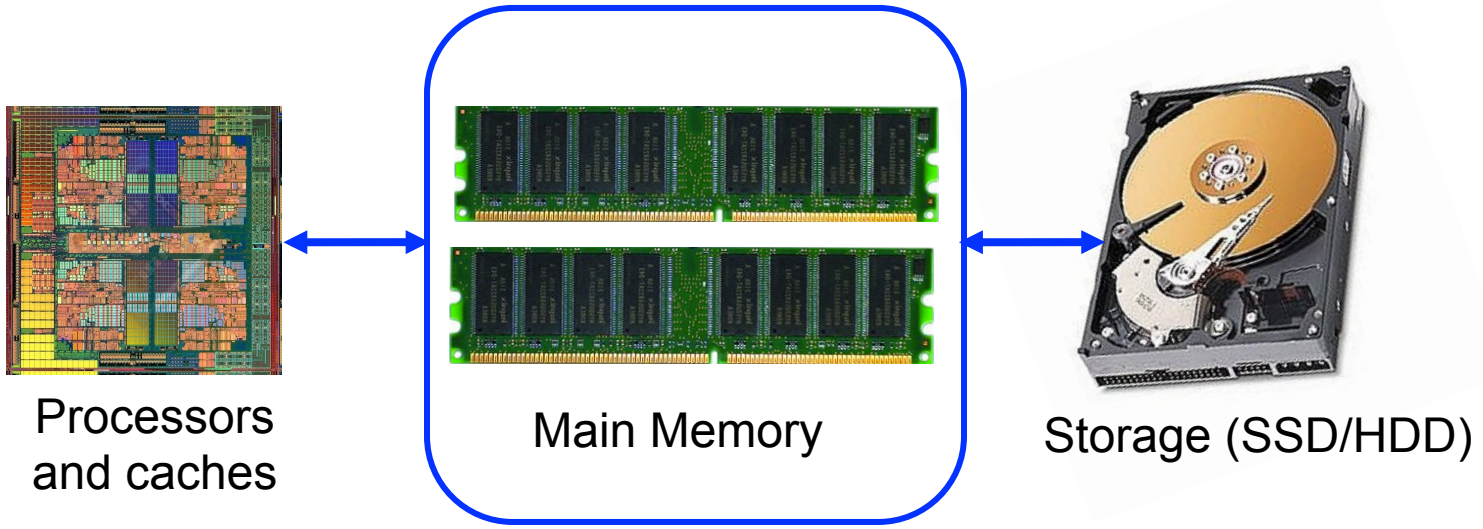
June 5, 2017

Technion Seiden Workshop: Beyond CMOS

SAFARI

ETH zürich

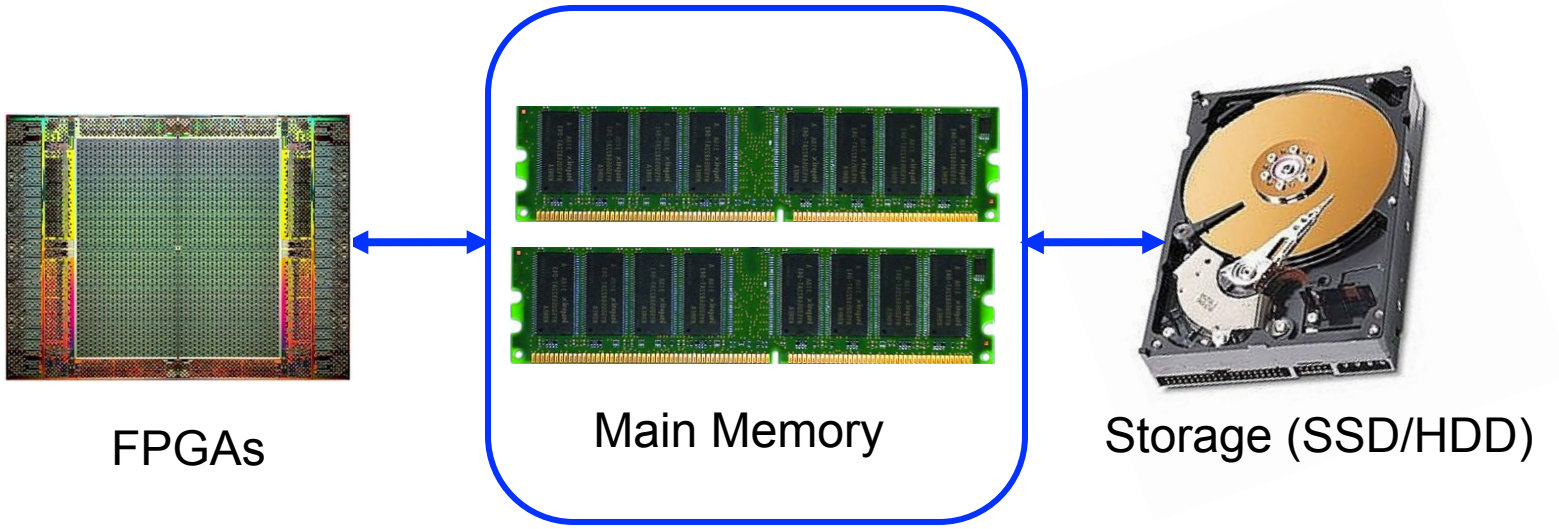
The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- Main memory system must scale (in *size*, *technology*, *efficiency*, *cost*, and *management algorithms*) to maintain performance growth and technology scaling benefits

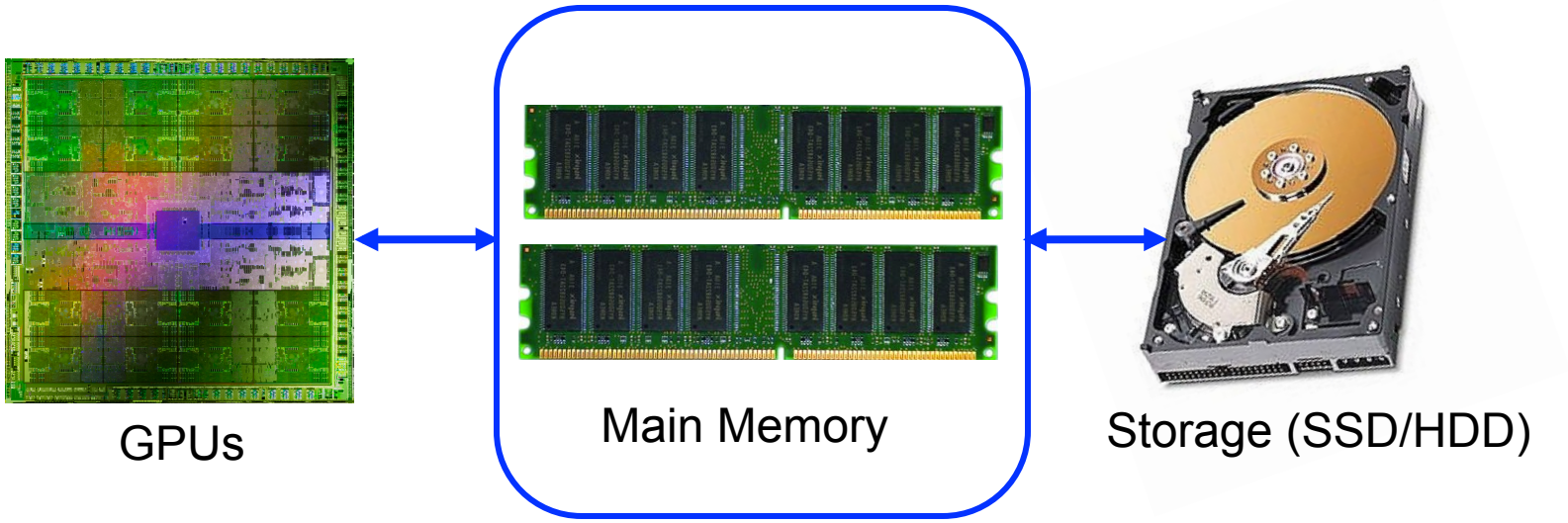
The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- Main memory system must scale (in *size*, *technology*, *efficiency*, *cost*, and *management algorithms*) to maintain performance growth and technology scaling benefits

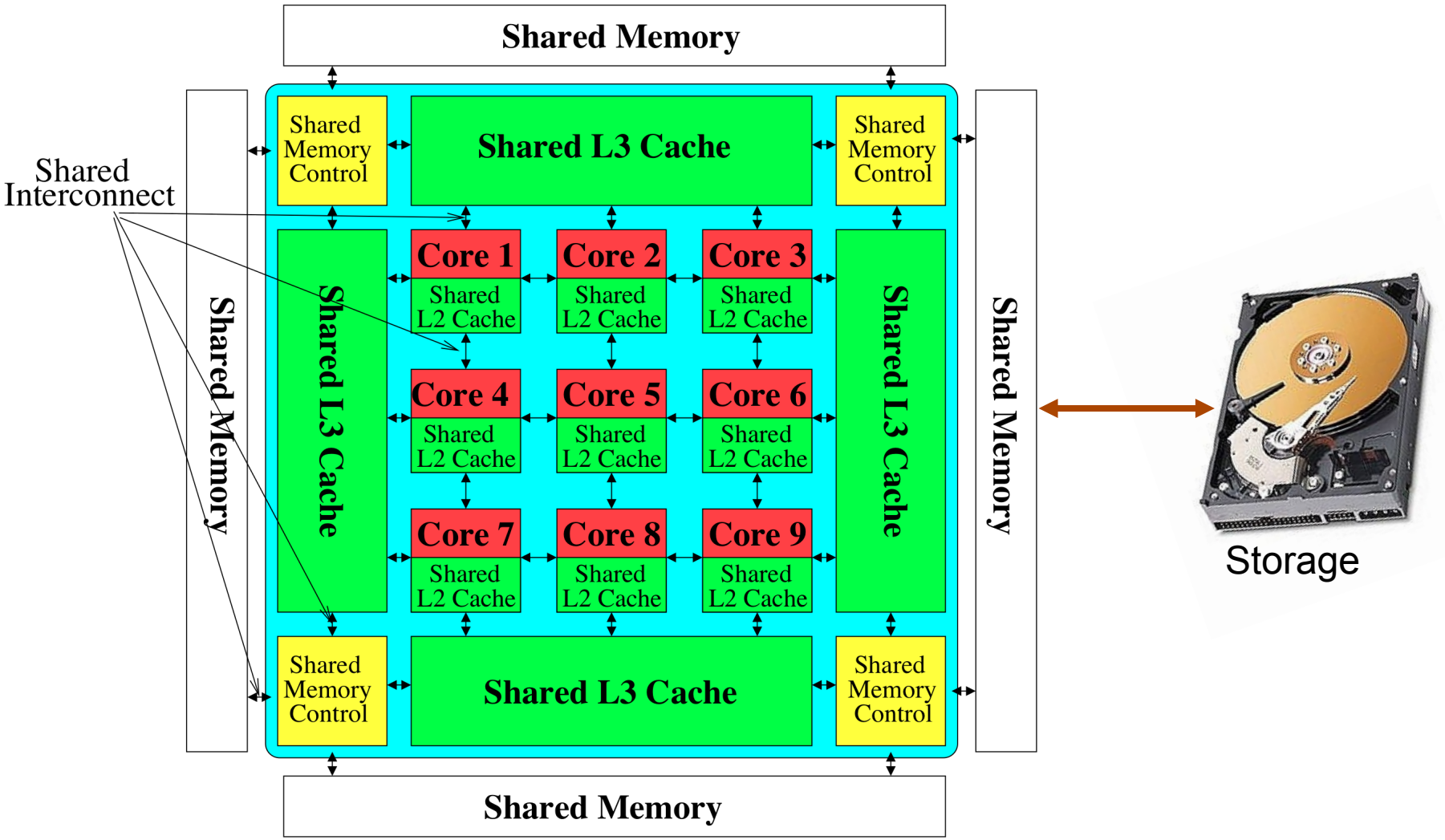
The Main Memory System



- Main memory is a critical component of all computing systems: server, mobile, embedded, desktop, sensor

- Main memory system must scale (in *size*, *technology*, *efficiency*, *cost*, and *management algorithms*) to maintain performance growth and technology scaling benefits

Memory System: A *Shared Resource* View



Most of the system is dedicated to storing and moving data

State of the Main Memory System

- Recent technology, architecture, and application trends
  - lead to new requirements
  - exacerbate old requirements

- DRAM and memory controllers, as we know them today, are (will be) unlikely to satisfy all requirements

- Some emerging non-volatile memory technologies (e.g., PCM) enable new opportunities: memory+storage merging

- We need to rethink the main memory system
  - to fix DRAM issues and enable emerging technologies
  - to satisfy all requirements

Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

Major Trends Affecting Main Memory (I)

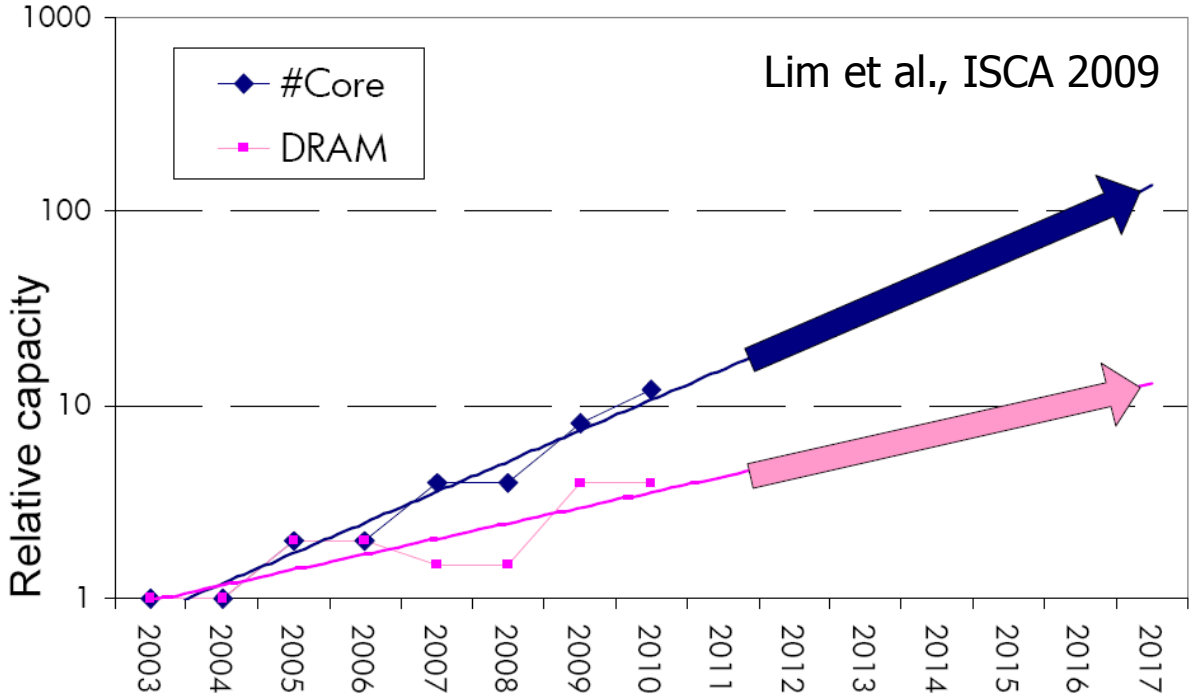
- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

Major Trends Affecting Main Memory (II)

- Need for main memory capacity, bandwidth, QoS increasing
  - Multi-core: increasing number of cores/agents
  - Data-intensive applications: increasing demand/hunger for data
  - Consolidation: cloud computing, GPUs, mobile, heterogeneity
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

Example: The Memory Capacity Gap

Core count doubling ~ every 2 years
DRAM DIMM capacity doubling ~ every 3 years



- *Memory capacity per core* expected to drop by 30% every two years
- Trends worse for *memory bandwidth per core*!

Major Trends Affecting Main Memory (III)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern
  - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer 2003]
  - DRAM consumes power even when not used (periodic refresh)

- DRAM technology scaling is ending

Major Trends Affecting Main Memory (IV)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern

- DRAM technology scaling is ending
  - ITRS projects DRAM will not scale easily below X nm
  - Scaling has provided many benefits:
    - higher capacity (density), lower cost, lower energy

Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

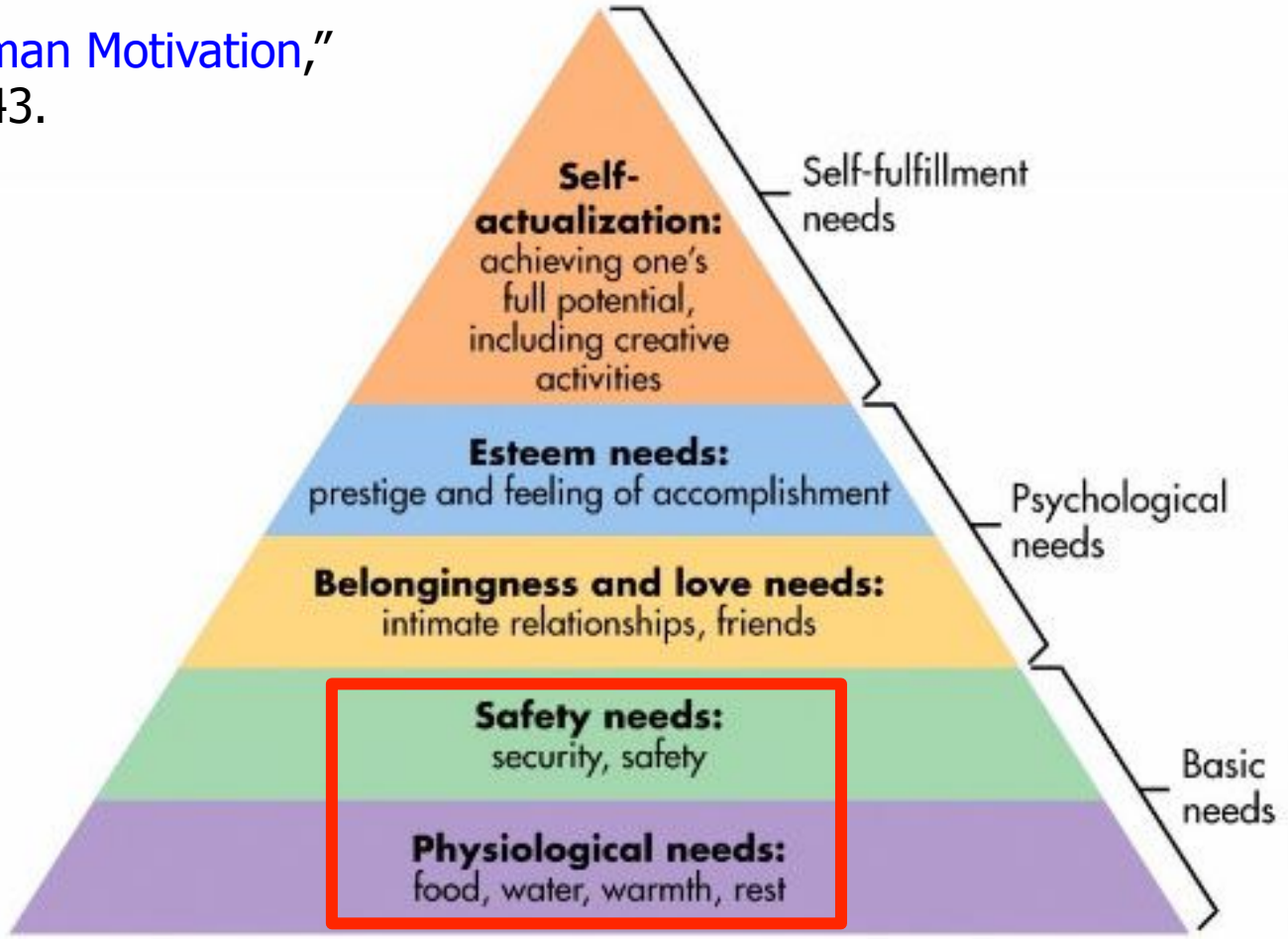
Two Key Issues in Future Platforms

- Fundamentally Secure/Reliable/Safe Architectures

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

Maslow's (Human) Hierarchy of Needs

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

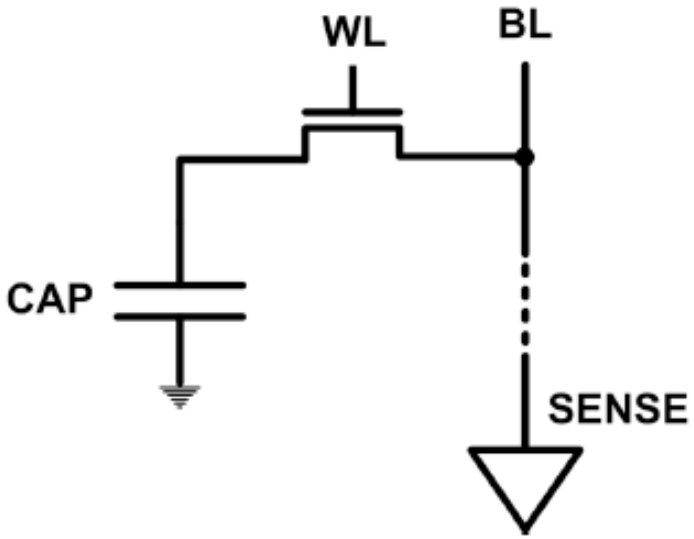


- We need to start with reliability and security…

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

The DRAM Scaling Problem

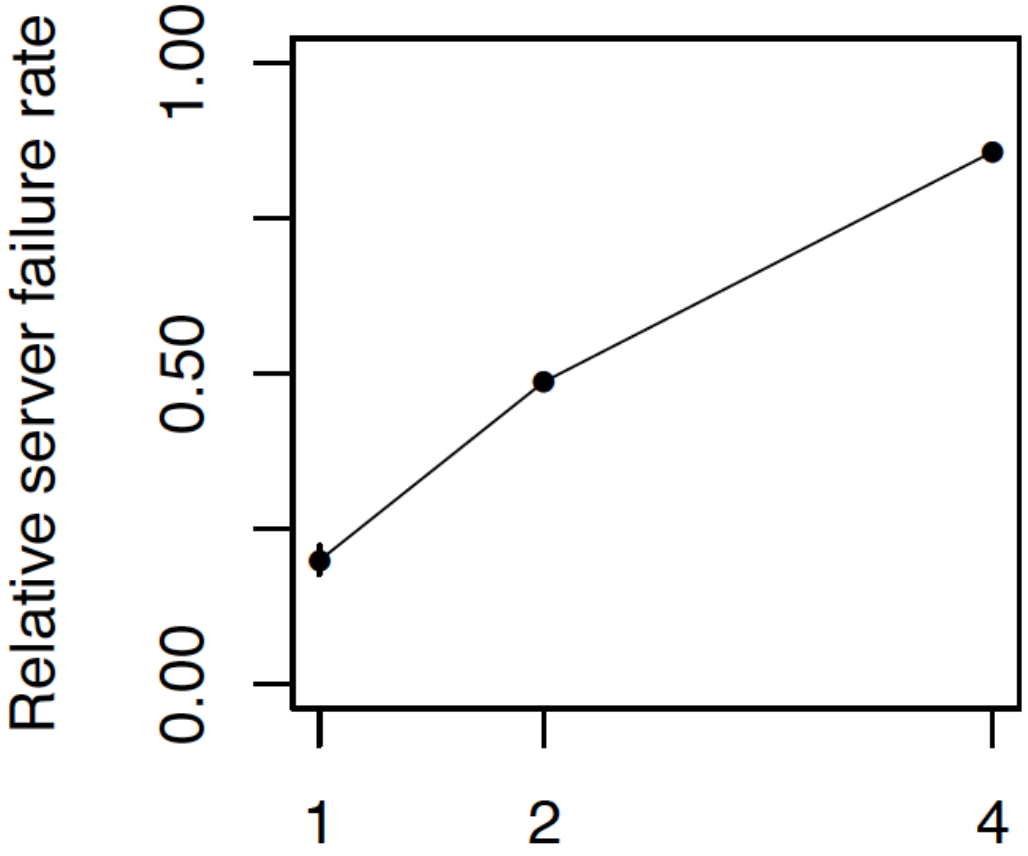
- DRAM stores charge in a capacitor (charge-based memory)
  - Capacitor must be large enough for reliable sensing
  - Access transistor should be large enough for low leakage and high retention time
  - Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]



- DRAM capacity, cost, and energy/power hard to scale

As Memory Scales, It Becomes Unreliable

- Data from all of Facebook's servers worldwide
- Meza+, "Revisiting Memory Errors in Large-Scale Production Data Centers," DSN'15.



Intuition: quadratic increase in capacity

Infrastructures to Understand Such Issues

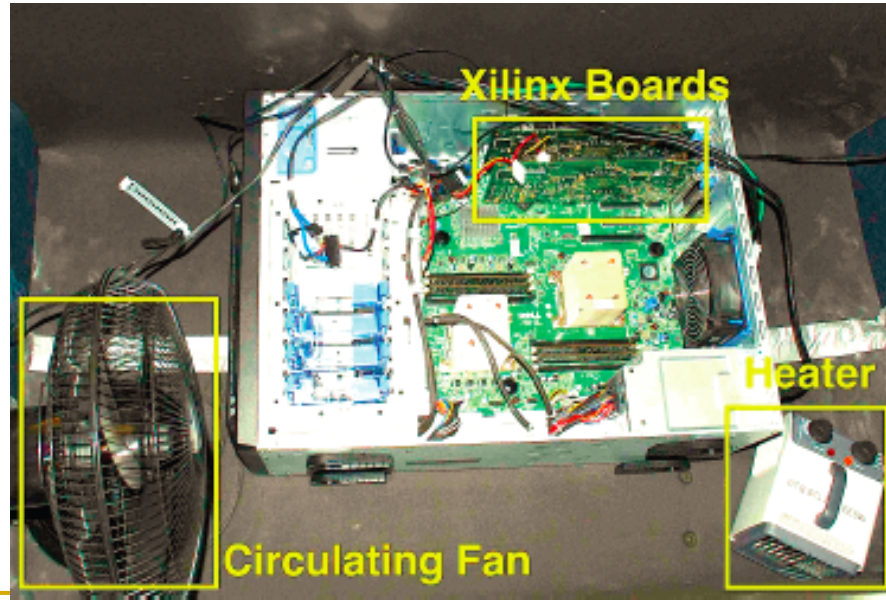
An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

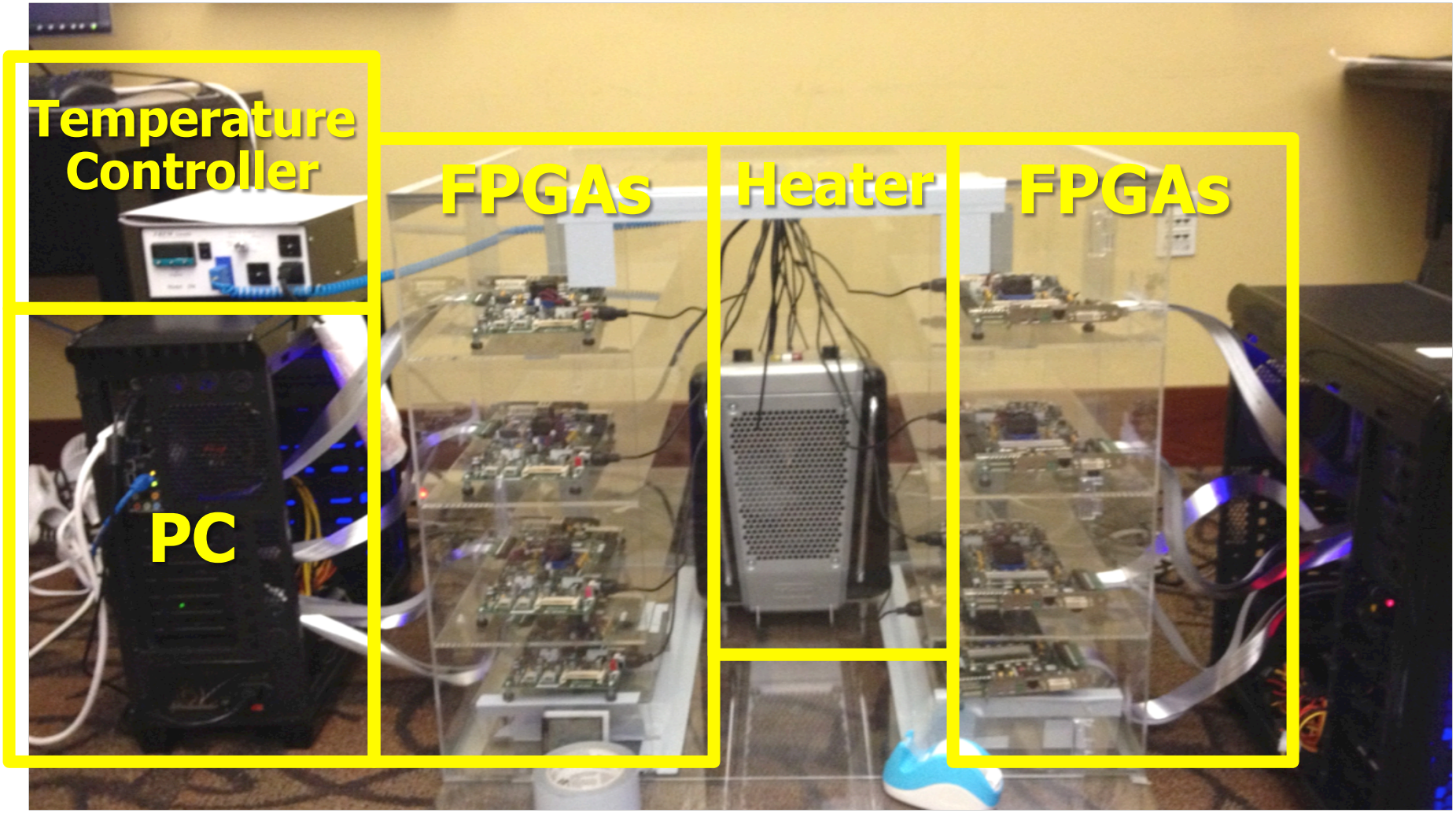
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)



Infrastructures to Understand Such Issues



Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

A Curious Discovery [Kim et al., ISCA 2014]

One can

predictably induce errors

in most DRAM memory chips

DRAM RowHammer

A simple hardware failure mechanism
can create a widespread
system security vulnerability

WIRED

Forget Software—Now Hackers Are Exploiting Physics

BUSINESS | CULTURE | DESIGN | GEAR | SCIENCE

SHARE

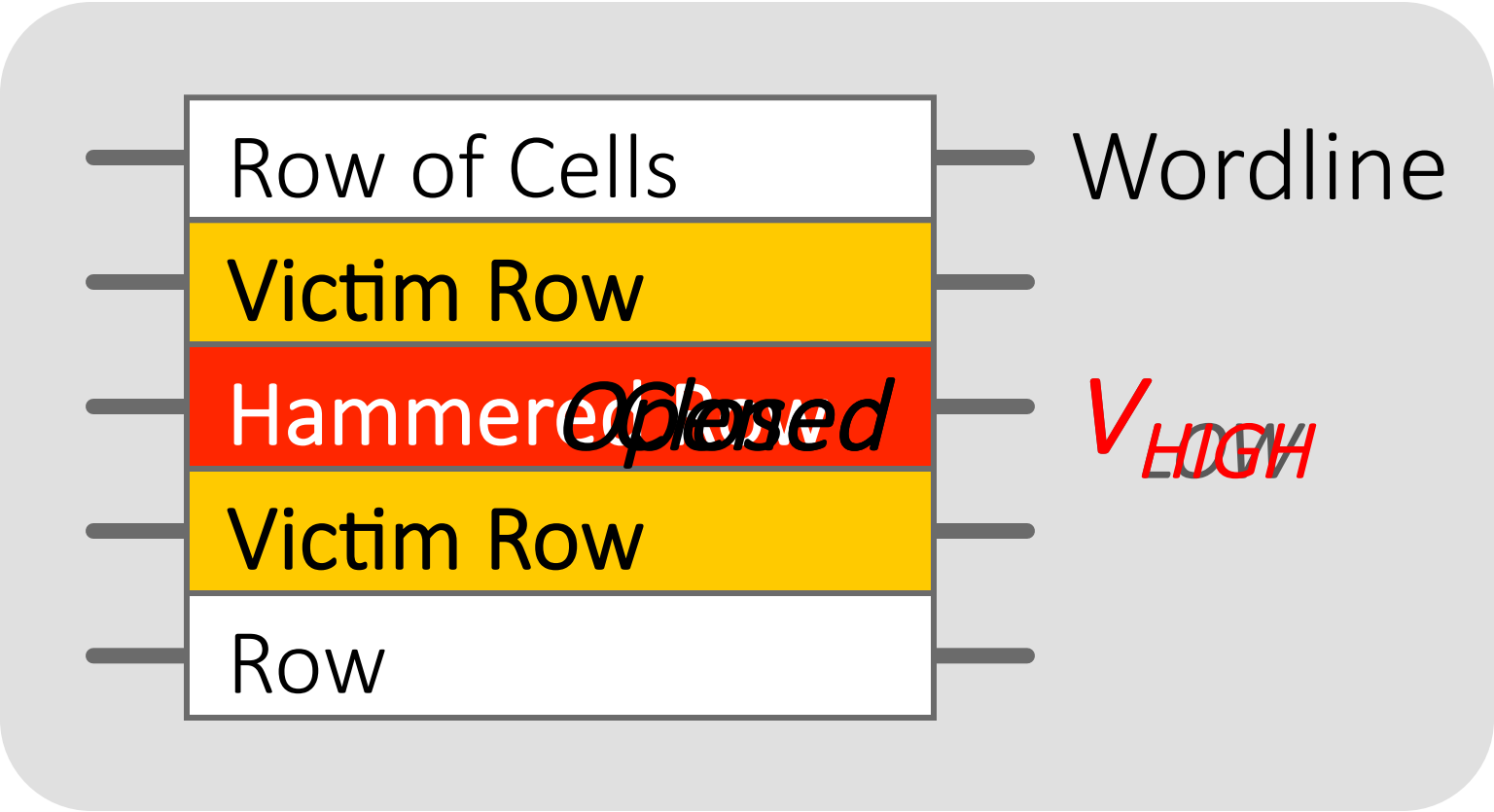
SHARE
18276

TWEET

ANDY GREENBERG SECURITY 08.31.16 7:00 AM

FORGET SOFTWARE—NOW HACKERS ARE EXPLOITING PHYSICS

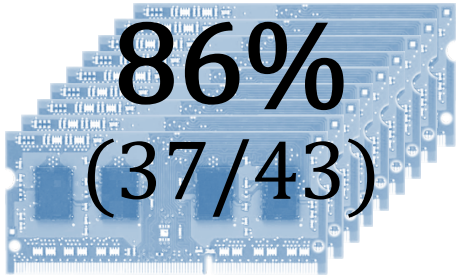
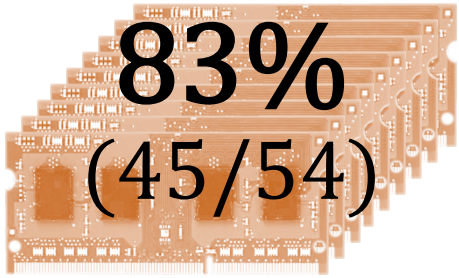
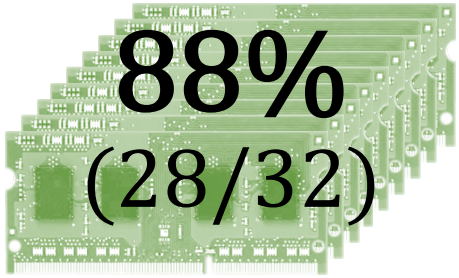
Modern DRAM is Prone to Disturbance Errors



Repeatedly reading a row enough times (before memory gets refreshed) induces disturbance errors in **adjacent rows** in most real DRAM chips you can buy today

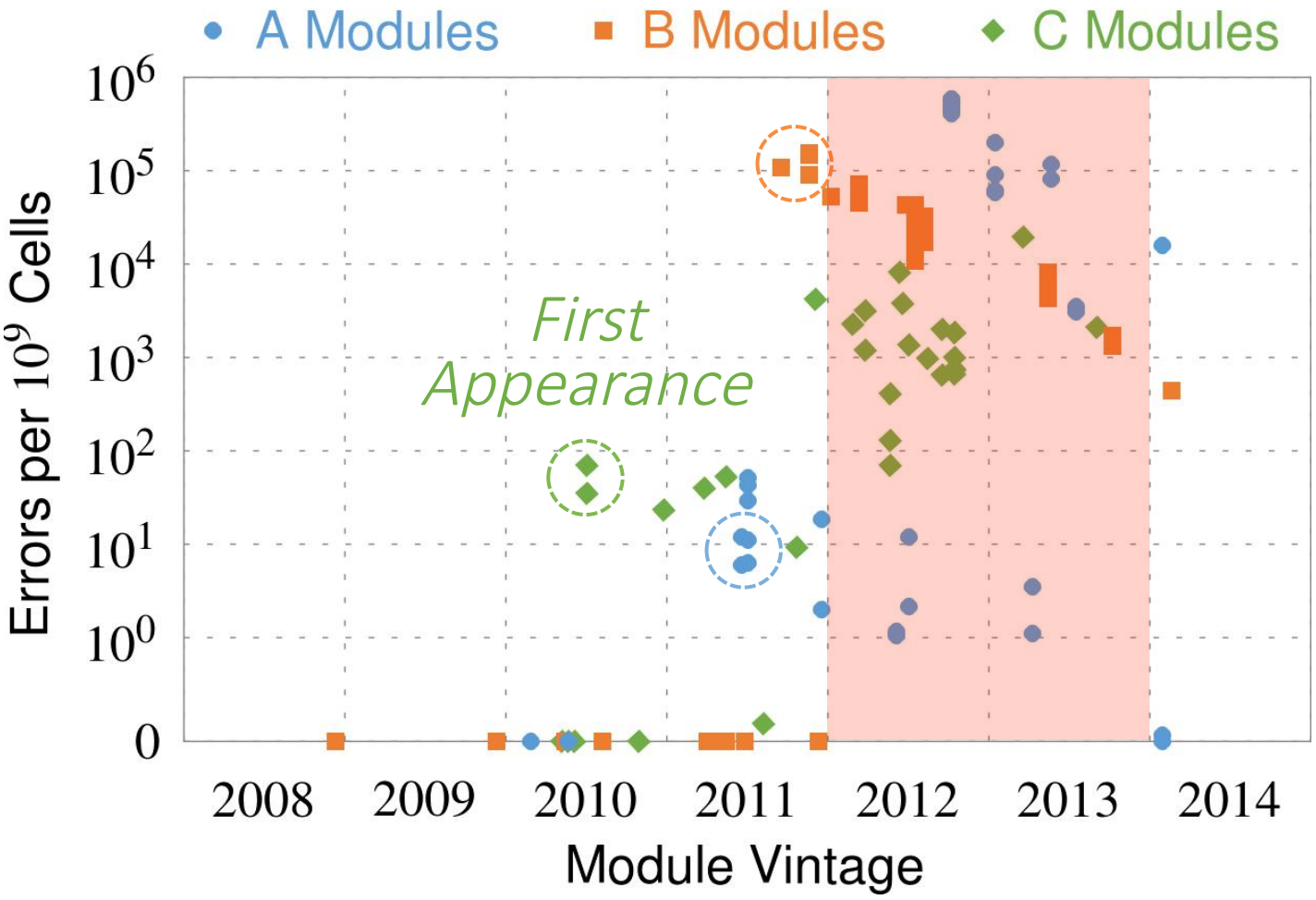
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors, (Kim et al., ISCA 2014)

Most DRAM Modules Are at Risk

A company	B company	C company
 86% (37/43)	 83% (45/54)	 88% (28/32)
Up to 1.0×10^{7} errors	Up to 2.7×10^{6} errors	Up to 3.3×10^{5} errors

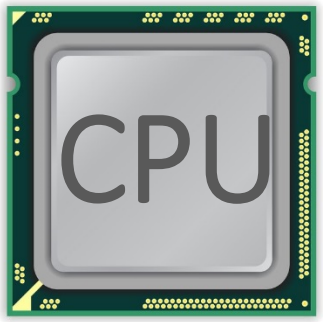
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors, (Kim et al., ISCA 2014)

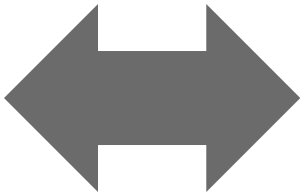
Recent DRAM Is More Vulnerable

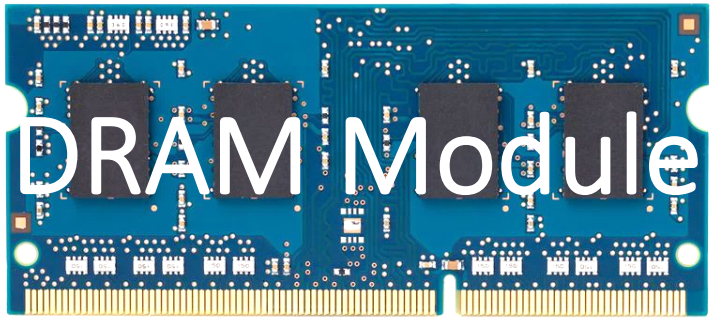


All modules from 2012–2013 are vulnerable

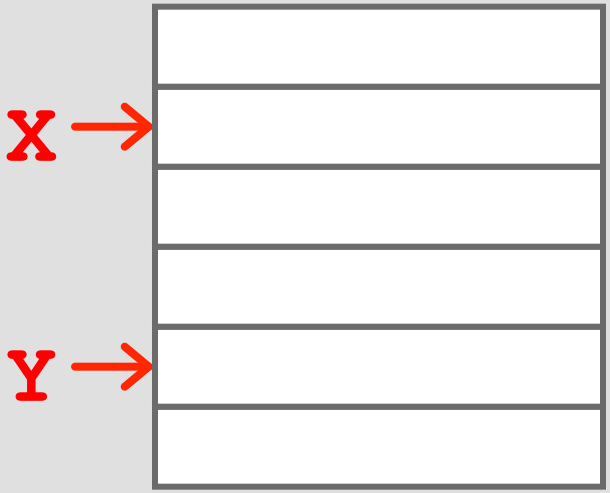
A Simple Program Can Induce Many Errors





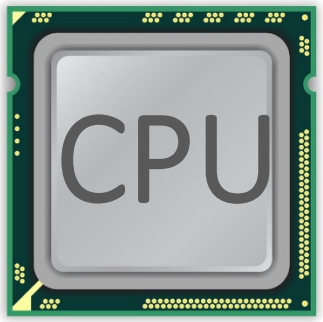


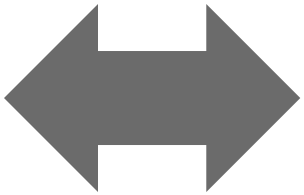

```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

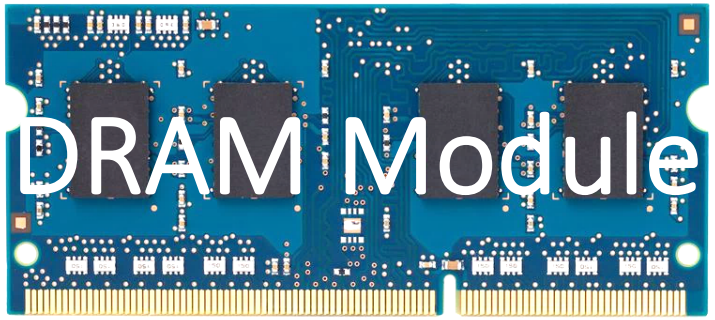


Download from: https://github.com/CMU-SAFARI/rowhammer

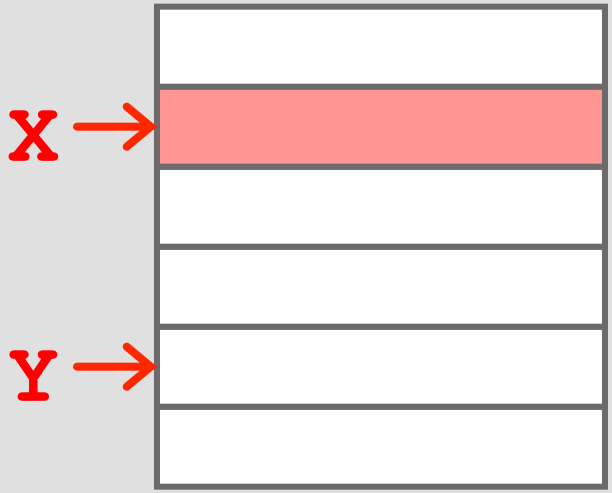
A Simple Program Can Induce Many Errors





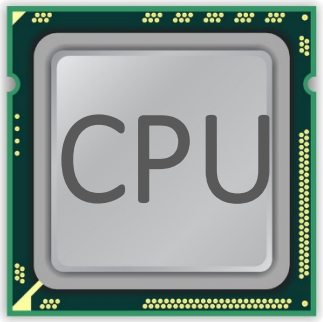


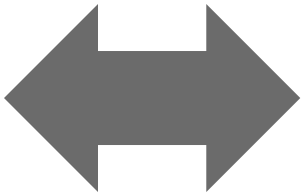

```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

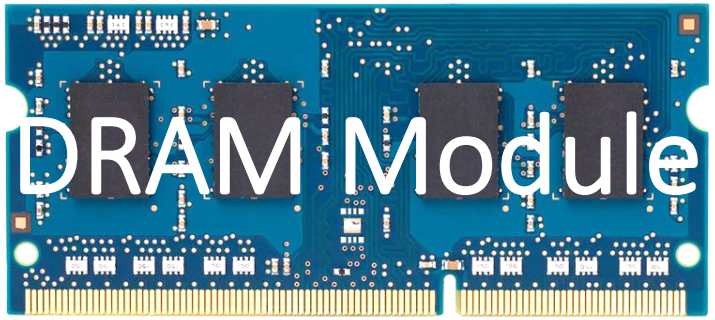


Download from: https://github.com/CMU-SAFARI/rowhammer

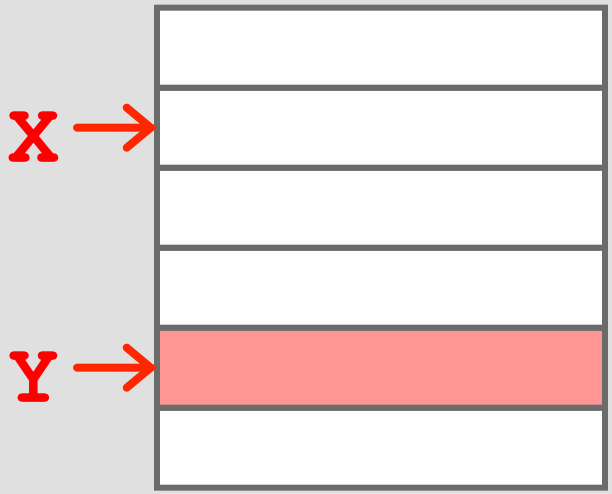
A Simple Program Can Induce Many Errors





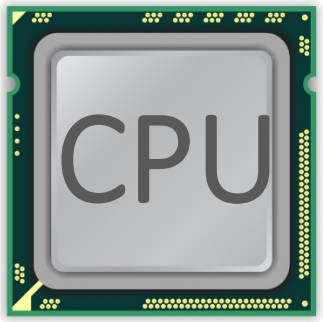


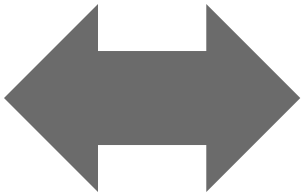

```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

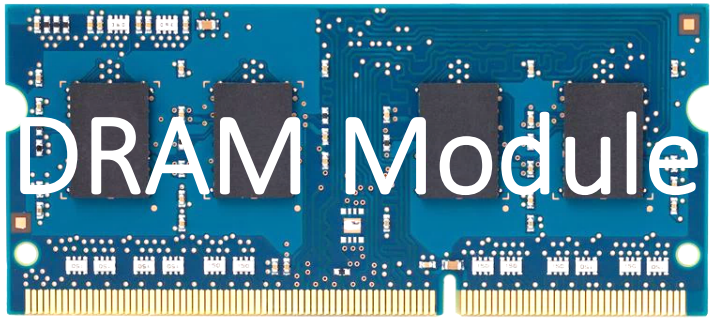


Download from: https://github.com/CMU-SAFARI/rowhammer

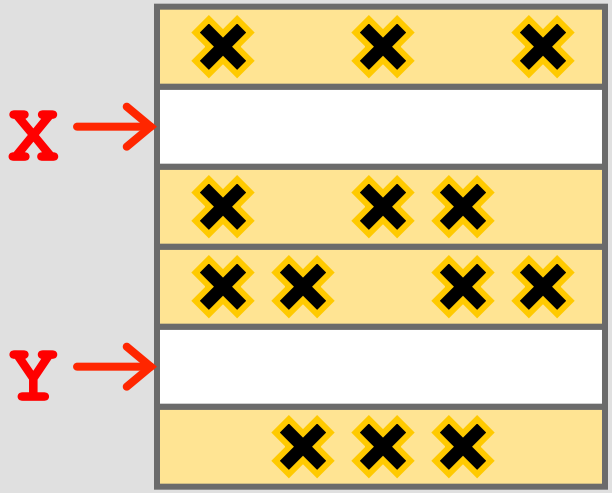
A Simple Program Can Induce Many Errors








```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



Download from: https://github.com/CMU-SAFARI/rowhammer

Observed Errors in Real Systems

CPU Architecture	Errors	Access-Rate
Intel Haswell (2013)	22.9K	12.3M/sec
Intel Ivy Bridge (2012)	20.7K	11.7M/sec
Intel Sandy Bridge (2011)	16.1K	11.6M/sec
AMD Piledriver (2012)	59	6.1M/sec

A real reliability & security issue

Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

One Can Take Over an Otherwise-Secure System

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

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

Project Zero

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

News and updates from the Project Zero team at Google

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

Monday, March 9, 2015

Exploiting the DRAM rowhammer bug to gain kernel privileges

RowHammer Security Attack Example

- "Rowhammer" is a problem with some recent DRAM devices in which repeatedly accessing a row of memory can cause bit flips in adjacent rows (Kim et al., ISCA 2014).
  - Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)
- We tested a selection of laptops and found that a subset of them exhibited the problem.
- We built two working privilege escalation exploits that use this effect.
  - Exploiting the DRAM rowhammer bug to gain kernel privileges (Seaborn, 2015)
- One exploit uses rowhammer-induced bit flips to gain kernel privileges on x86-64 Linux when run as an unprivileged userland process.
- When run on a machine vulnerable to the rowhammer problem, the process was able to induce bit flips in page table entries (PTEs).
- It was able to use this to gain write access to its own page table, and hence gain read-write access to all of physical memory.

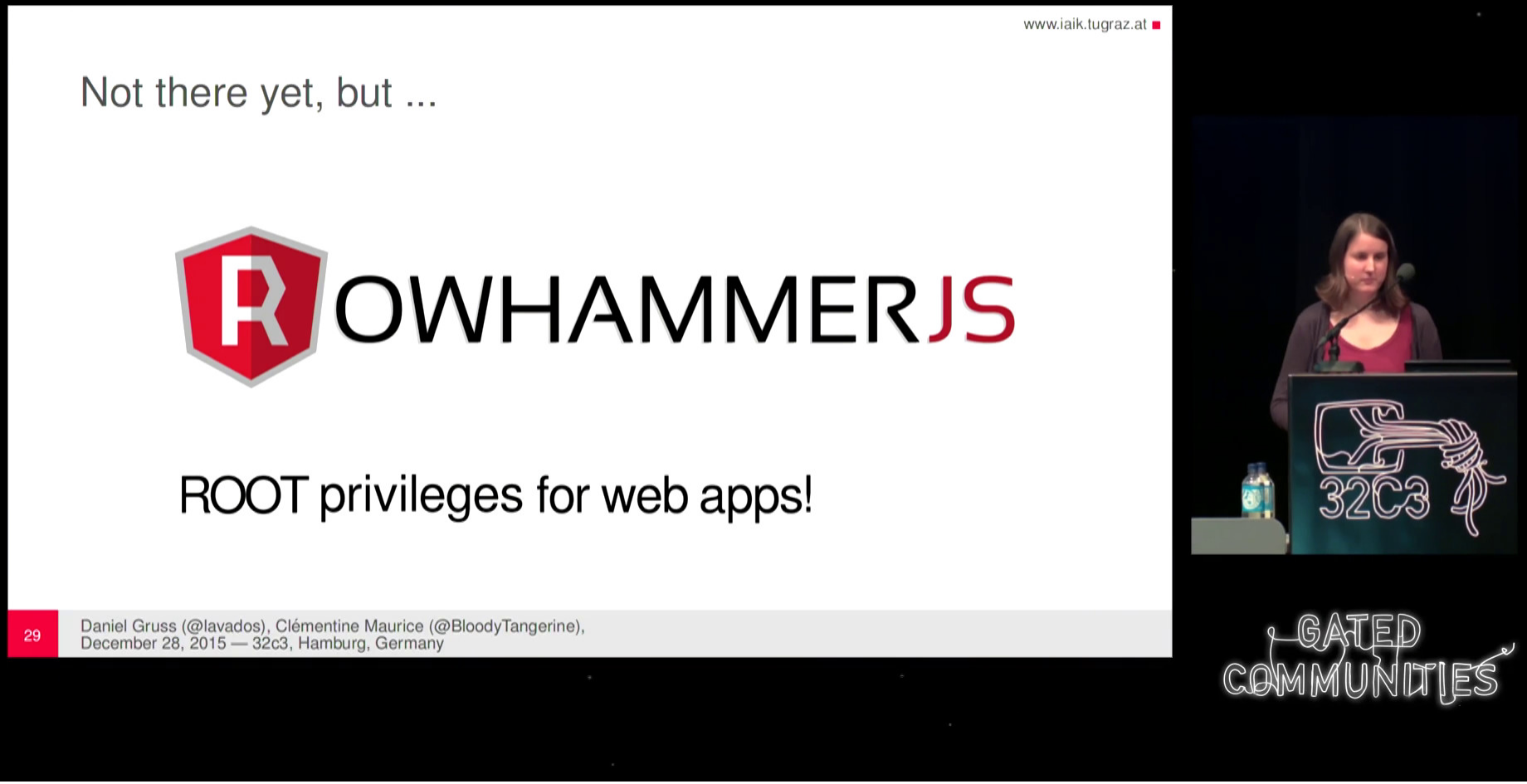
Security Implications

Security Implications

It's like breaking into an apartment by repeatedly slamming a neighbor's door until the vibrations open the door you were after

More Security Implications

"We can gain unrestricted access to systems of website visitors."



Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript (DIMVA'16)

Source: https://lab.dsst.io/32c3-slides/7197.html

More Security Implications

"Can gain control of a smart phone deterministically"

Drammer: Deterministic Rowhammer Attacks on Mobile Platforms, CCS'16

Source: https://fossbytes.com/drammer-rowhammer-attack-android-root-devices/

More Security Implications?

Apple's Patch for RowHammer

- https://support.apple.com/en-gb/HT204934

Available for: OS X Mountain Lion v10.8.5, OS X Mavericks v10.9.5

Impact: A malicious application may induce memory corruption to escalate privileges

Description: A disturbance error, also known as Rowhammer, exists with some DDR3 RAM that could have led to memory corruption. This issue was mitigated by increasing memory refresh rates.

CVE-ID

CVE-2015-3693 : Mark Seaborn and Thomas Dullien of Google, working from original research by Yoongu Kim et al (2014)

HP, Lenovo, and other vendors released similar patches

Better Solution Directions: Principled Designs

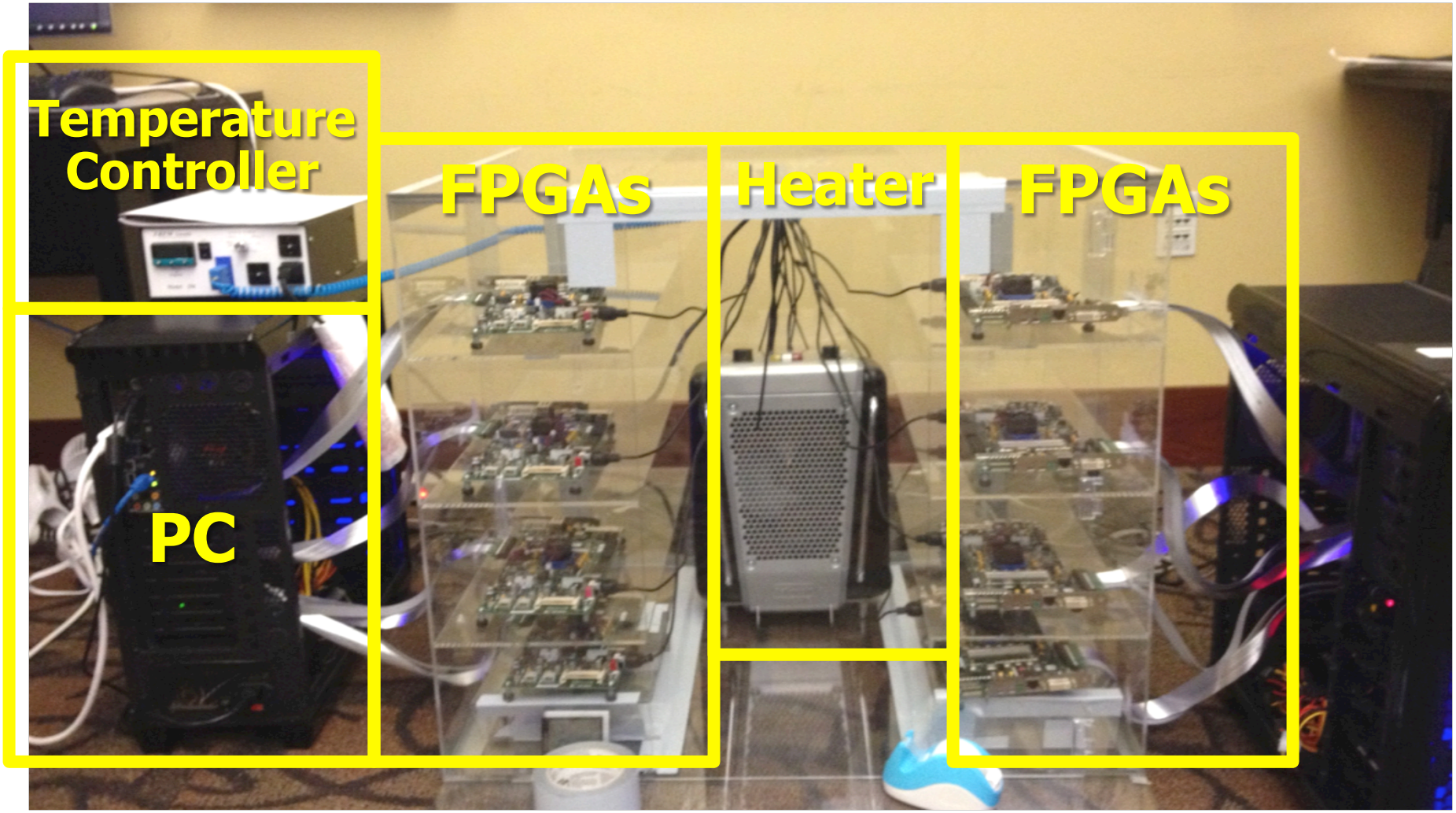
Design fundamentally secure computing architectures

Predict and prevent such safety issues

How Do We Keep Memory Secure?

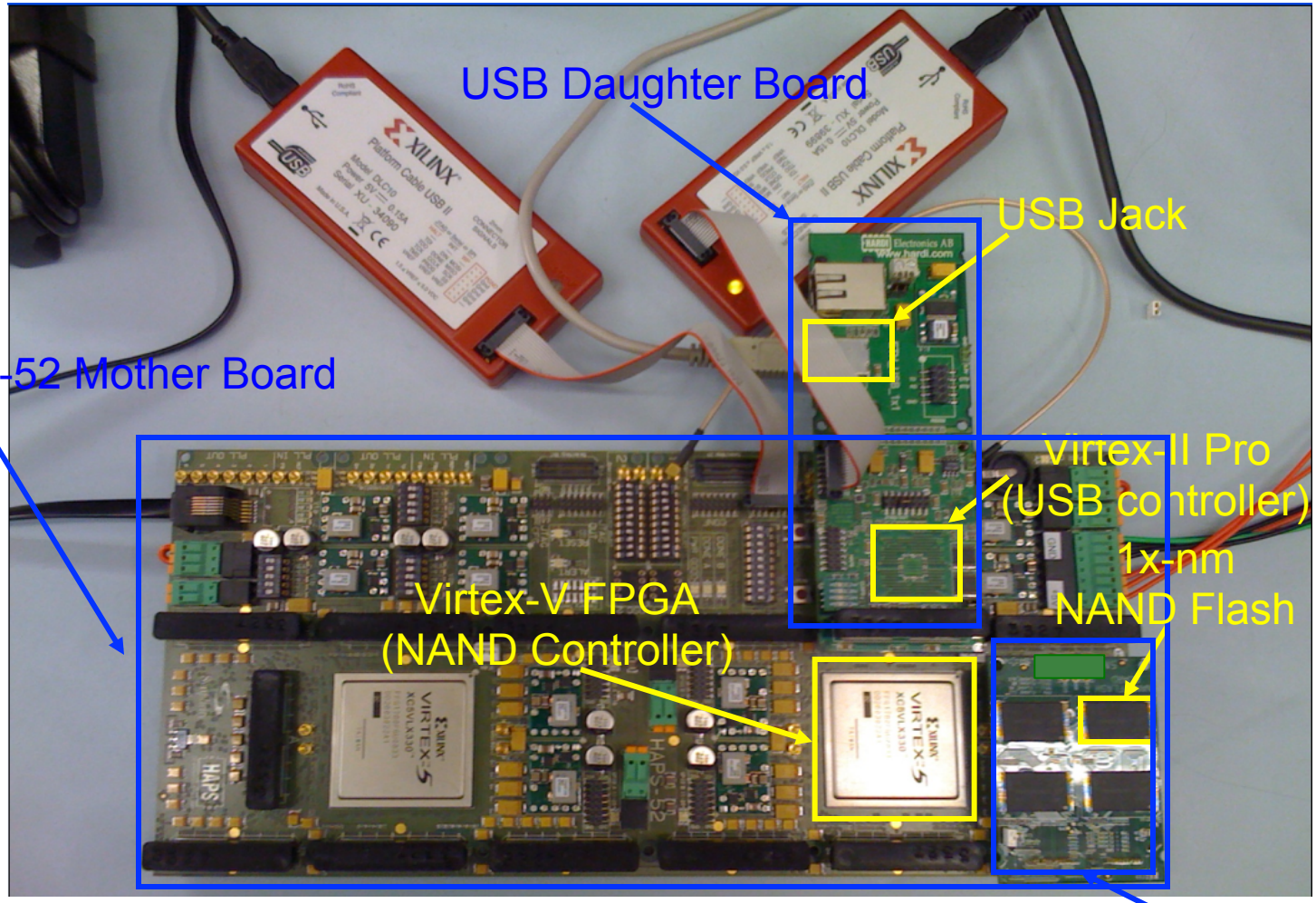
- **Understand**: Methodologies for failure modeling and discovery
  - Modeling and prediction based on real (device) data
- **Architect**: Principled co-architecting of system and memory
  - Good partitioning of duties across the stack
- **Design & Test**: Principled design, automation, testing
  - High coverage and good interaction with system reliability methods

Understand and Model with Experiments (DRAM)



Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

Understand and Model with Experiments (Flash)



[DATE 2012, ICCD 2012, DATE 2013, ITJ 2013, ICCD 2013, SIGMETRICS 2014, HPCA 2015, DSN 2015, MSST 2015, JSAC 2016, HPCA 2017, DFRWS 2017]

Cai+, "Error Characterization, Mitigation, and Recovery in Flash Memory Based Solid State Drives," Proc. IEEE 2017.

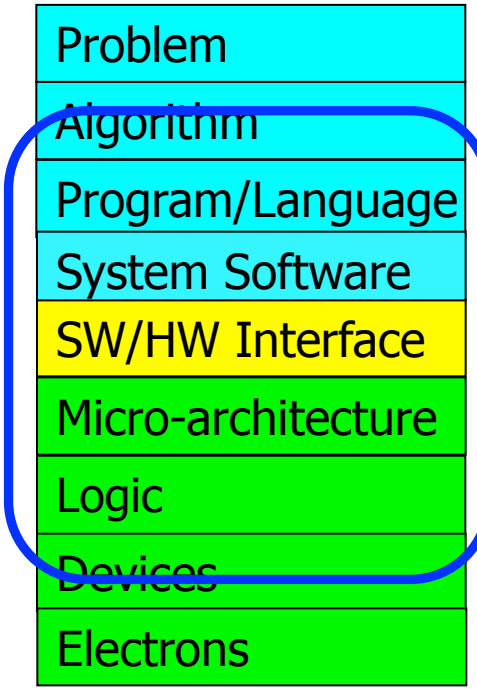
There are Two Other Solutions

- **New Technologies:** Replace or (more likely) augment DRAM with a different technology
  - Non-volatile memories

- **Embracing Un-reliability:**

  Design memories with different reliability and store data intelligently across them

- …



Fundamental solutions to security require co-design across the hierarchy

Challenge and Opportunity for Future

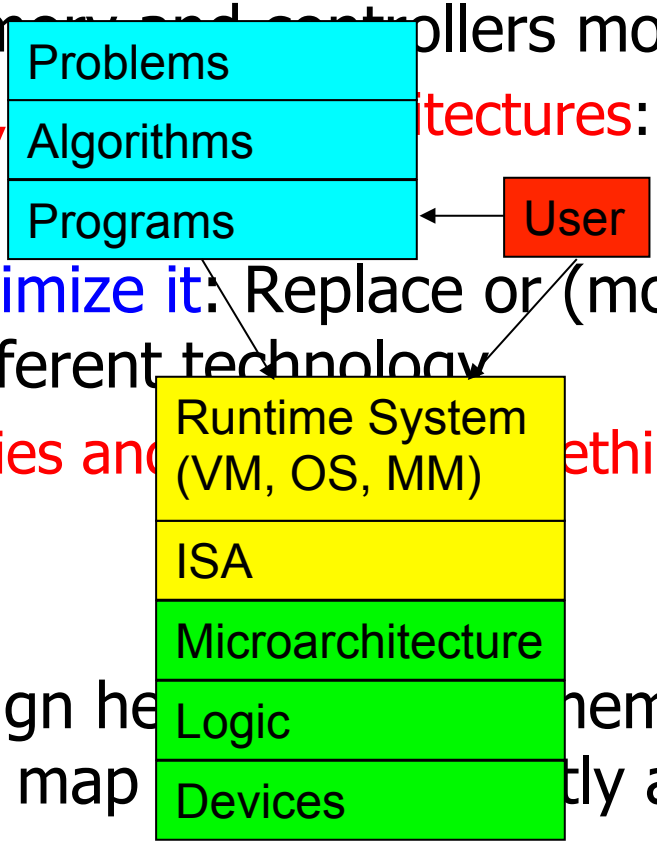
Fundamentally
Secure, Reliable, Safe
Computing Architectures

Solving the Memory Scaling Problem

- Fix it: Make memory and controllers more intelligent
  - New interfaces, functions, architectures: system-mem codesign
- Eliminate or minimize it: Replace or (more likely) augment DRAM with a different technology
  - New technologies and system-wide rethinking of memory & storage
- Embrace it: Design heterogeneous memories (none of which are perfect) and map data intelligently across them
  - New models for data management and maybe usage
- …

Solving the Memory Scaling Problem

- Fix it: Make mem[illegible]llers more intelligent
  - New interfaces, [illegible]itectures: system-mem codesign
- Eliminate or minimize it: Replace or (more likely) augment DRAM with a different technology
  - New technologies an[illegible]ethinking of memory & storage
- Embrace it: Design he[illegible]nemories (none of which are perfect) and map [illegible]tly across them
  - New models for data management and maybe usage



Solutions (to memory scaling) require software/hardware/device cooperation

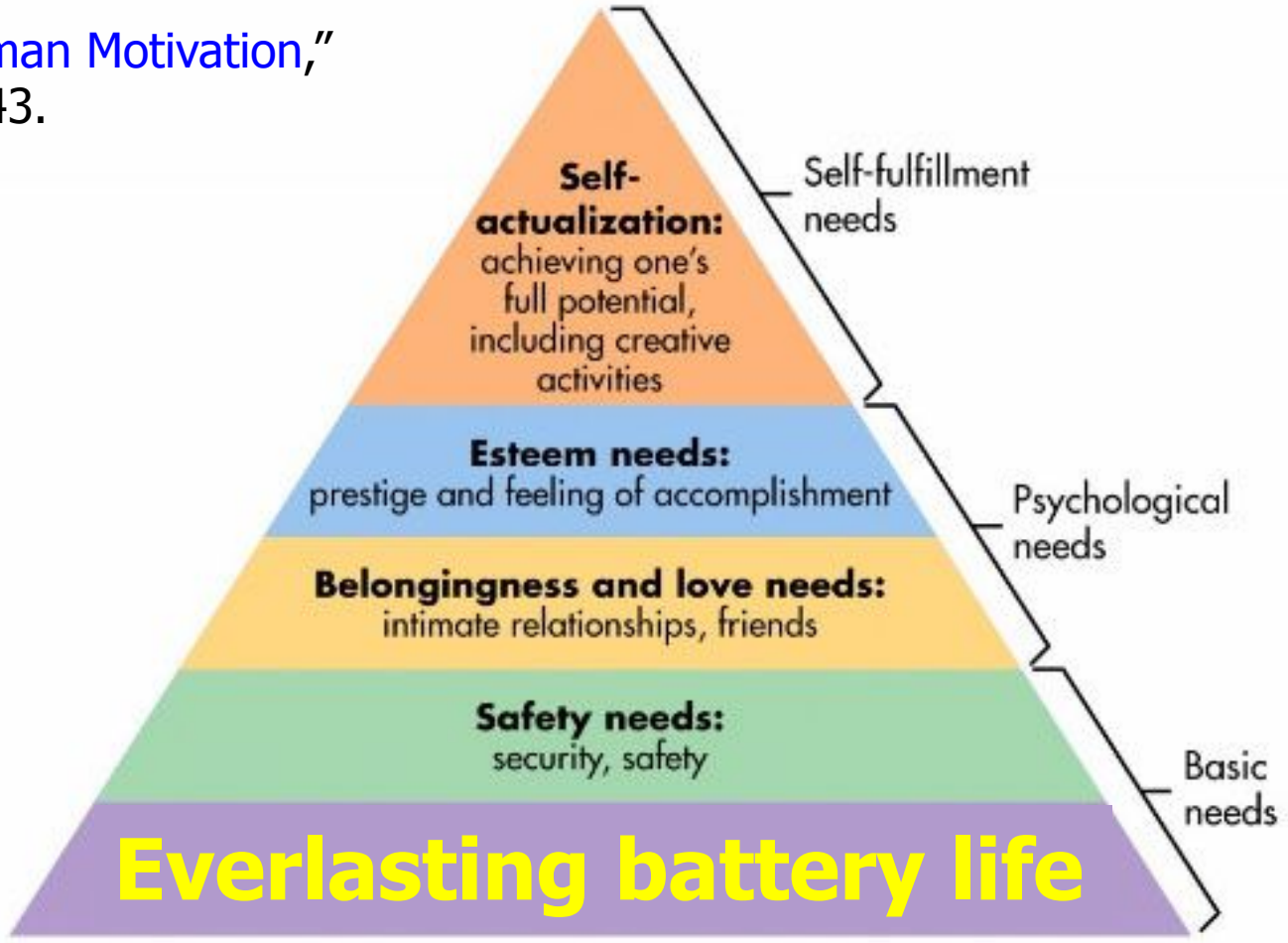
Two Key Issues in Future Platforms

- Fundamentally Secure/Reliable/Safe Architectures

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

Maslow's (Human) Hierarchy of Needs, Revisited

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



Challenge and Opportunity for Future

Sustainable

and

Energy Efficient

The Problem

Data access is the major performance and energy bottleneck

Our current

design principles

cause great energy waste

The Problem

Processing of data

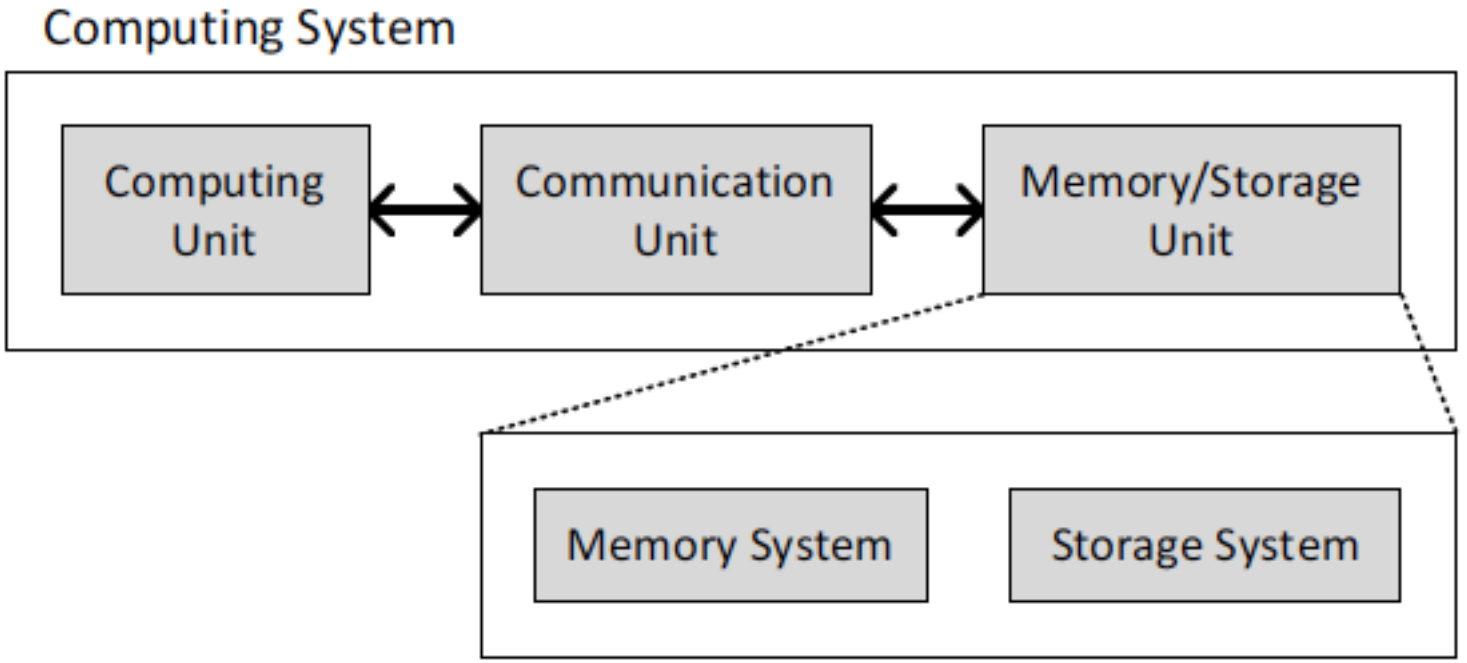
is performed

far away from the data

A Computing System

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

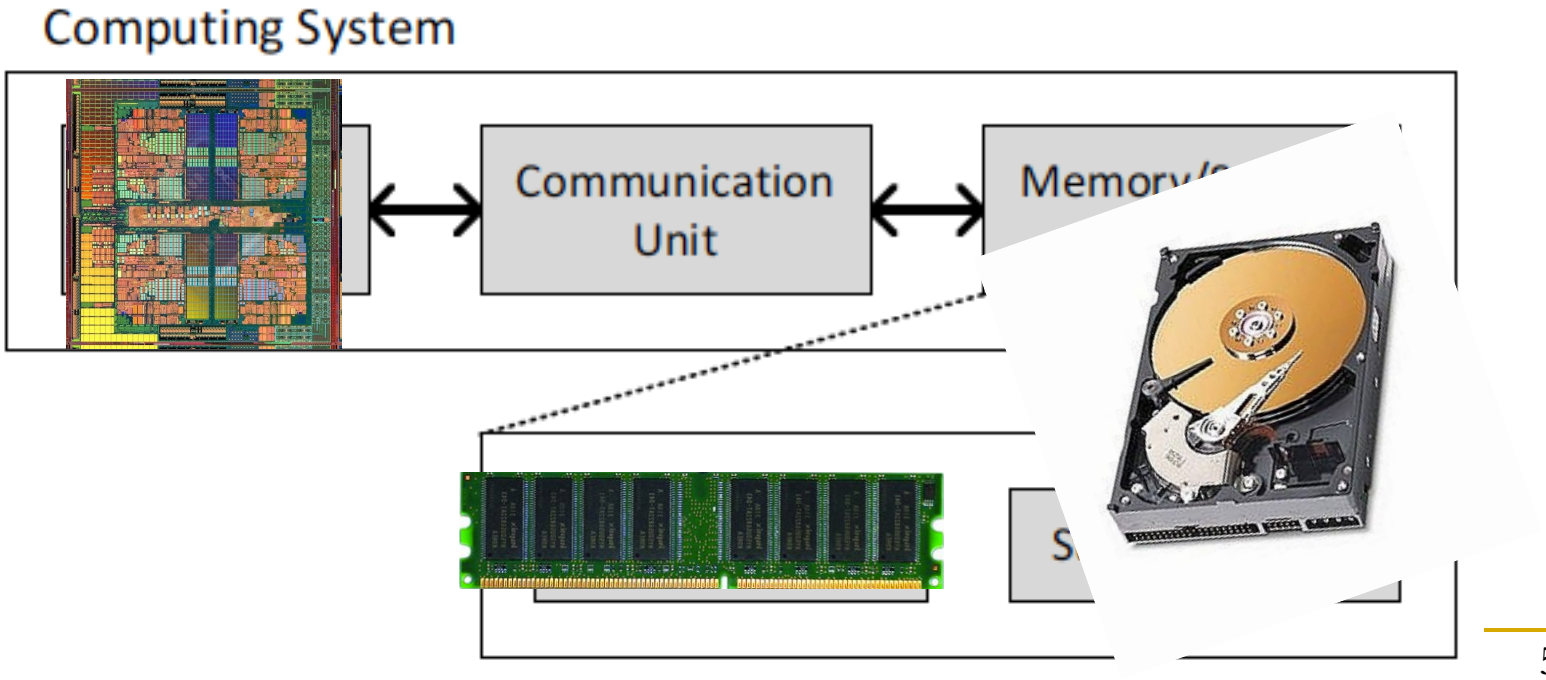
Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.



A Computing System

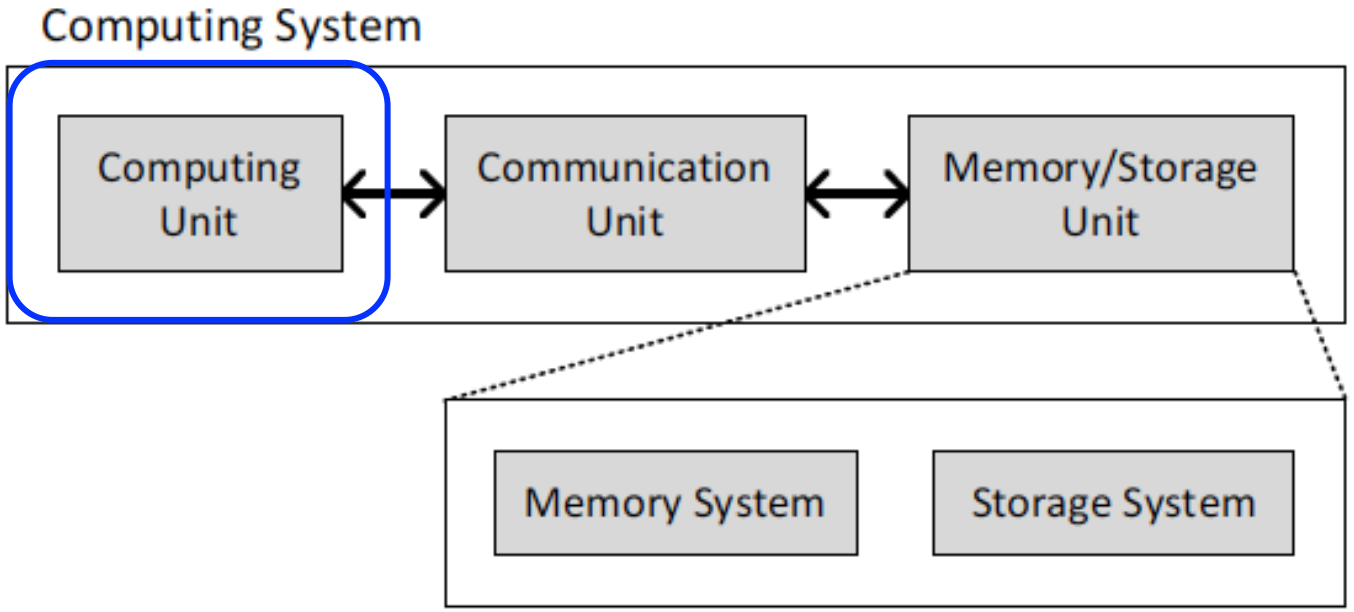
- Three key components
- Computation
- Communication
- Storage/memory

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



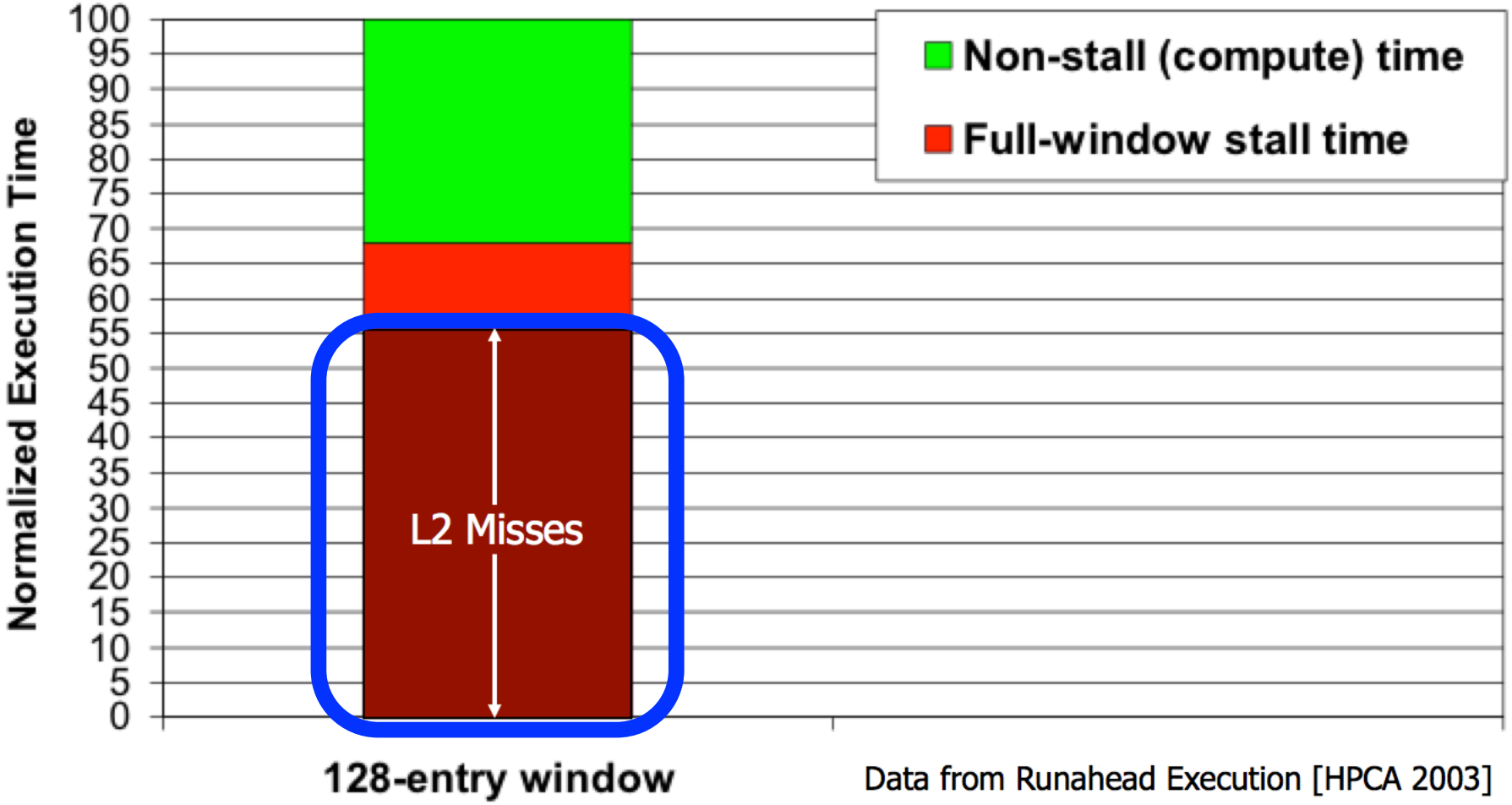
Today's Computing Systems

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



Yet …

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



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

Perils of Processor-Centric Design

- Grossly-imbalanced systems
  - Processing done only in **one place**
  - Everything else just stores and moves data: **data moves a lot**

  → Energy inefficient

  → Low performance

  → Complex

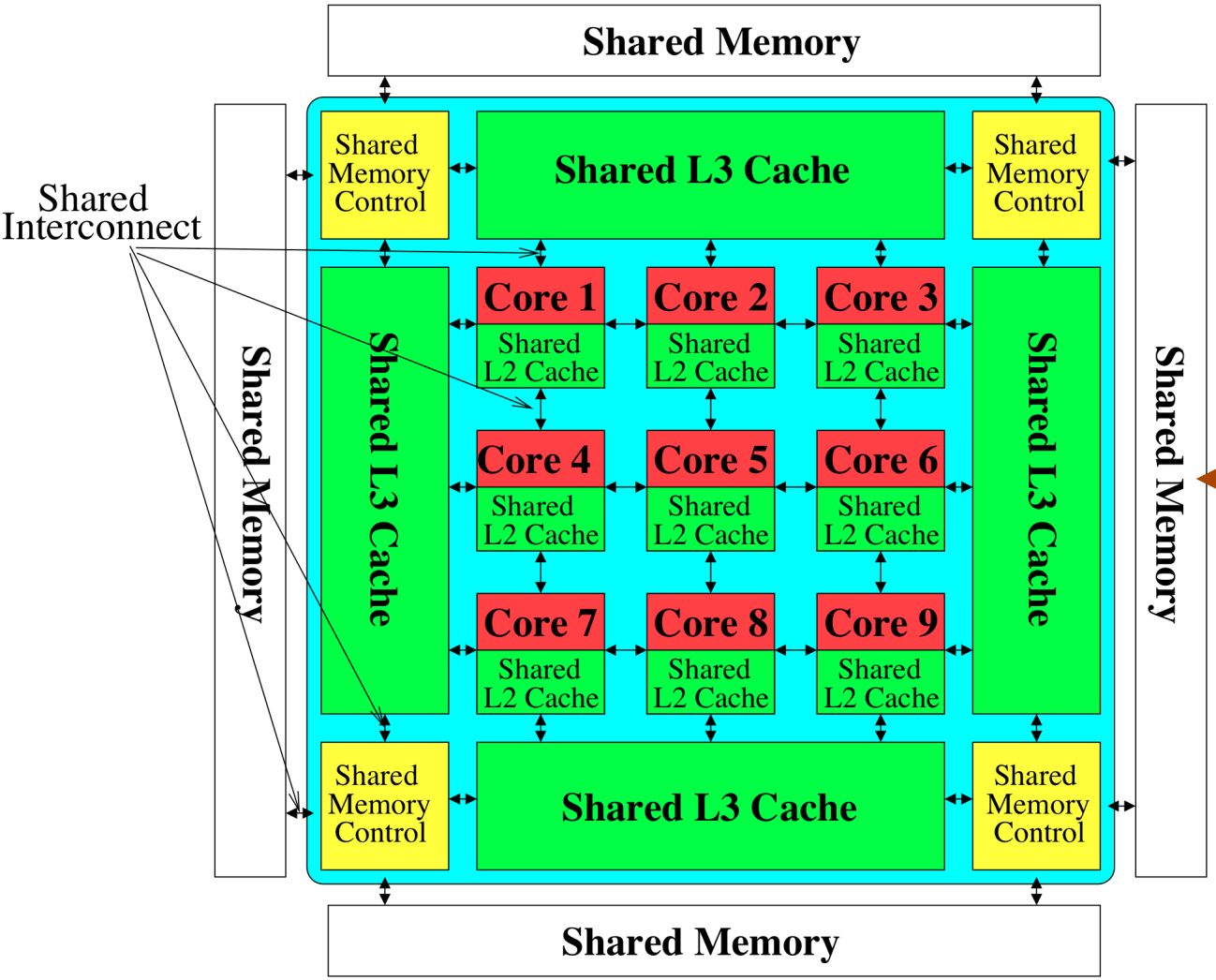
- Overly complex and bloated processor (and accelerators)
  - To tolerate data access from memory
  - Complex hierarchies and mechanisms

  → Energy inefficient

  → Low performance

  → Complex

Perils of Processor-Centric Design



Most of the system is dedicated to storing and moving data

Three Key Systems Trends

1. Data access is a major bottleneck

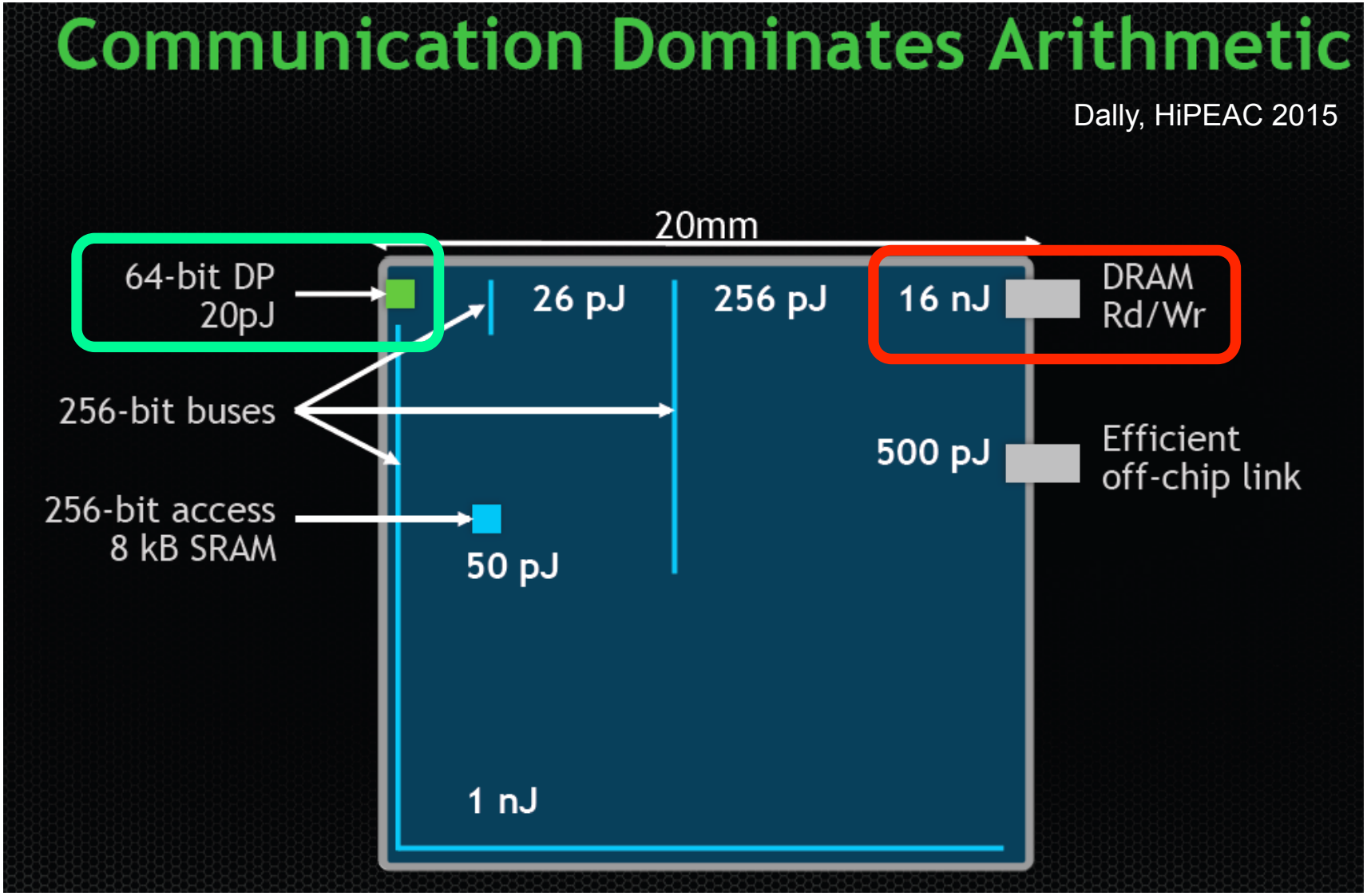
- Applications are increasingly data hungry

2. Energy consumption is a key limiter

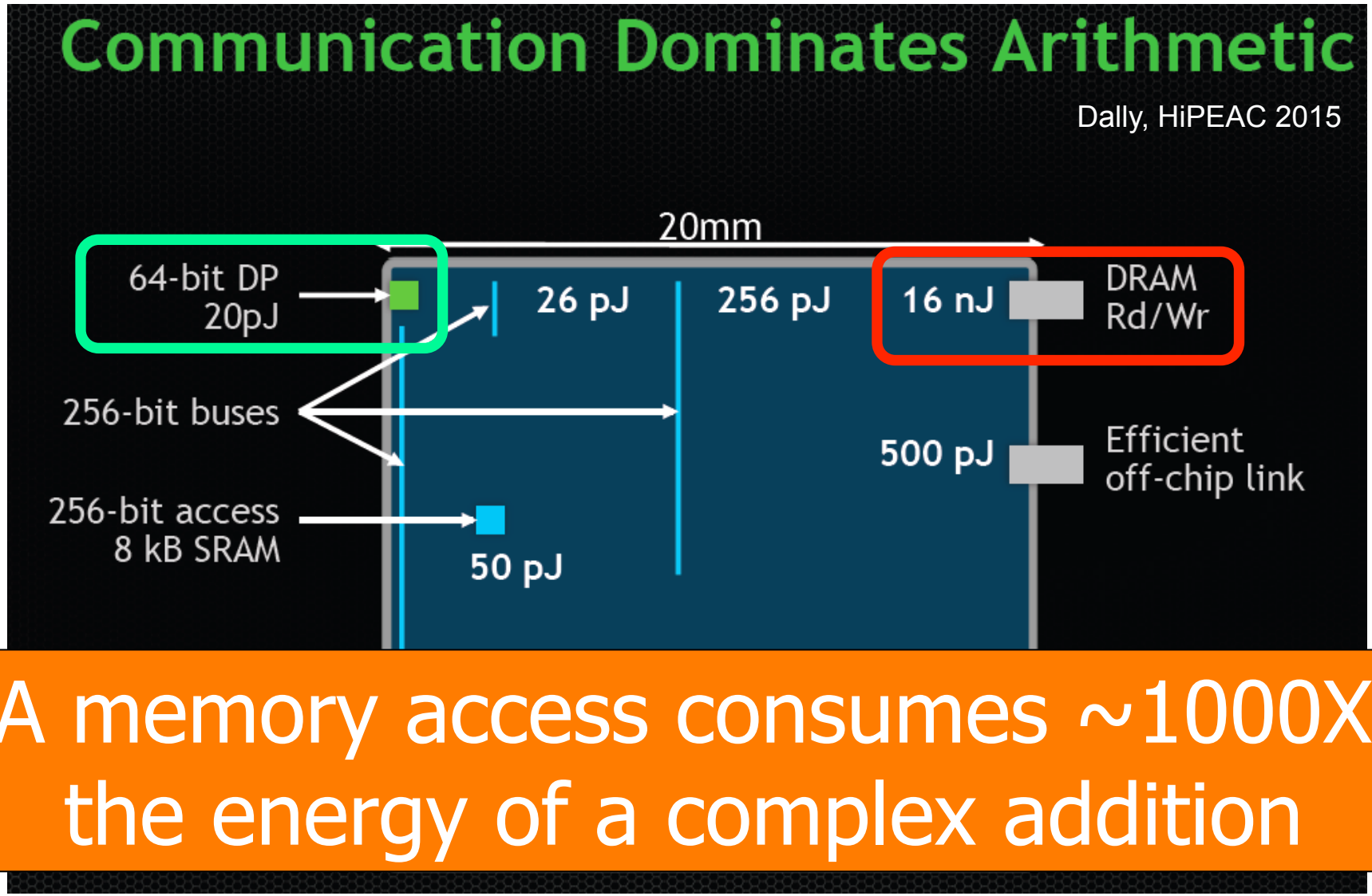
3. Data movement energy dominates compute

- Especially true for off-chip to on-chip movement

Data Movement vs. Computation Energy



Data Movement vs. Computation Energy

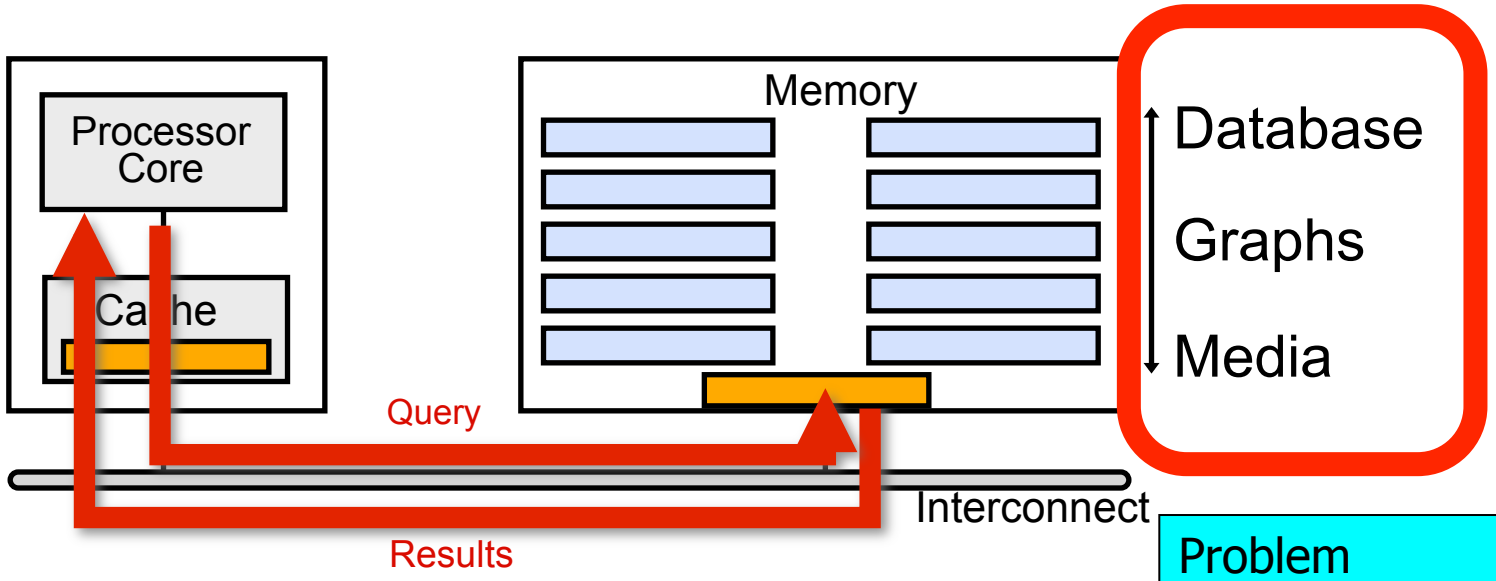


A memory access consumes ~1000X the energy of a complex addition

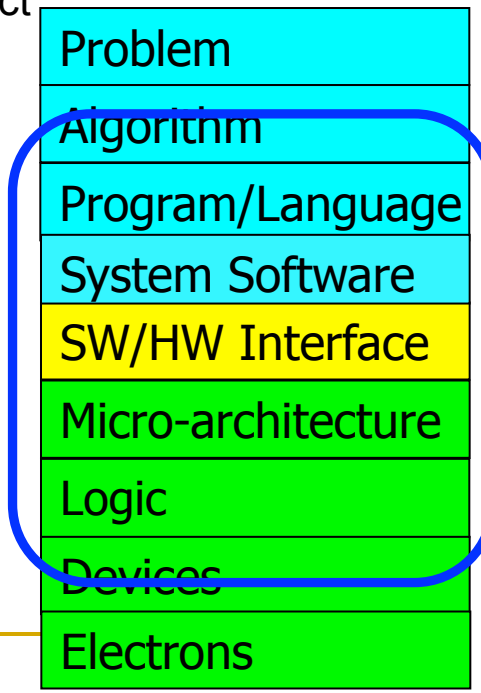
We Need A Paradigm Shift To …

- Enable computation with minimal data movement
- Compute where it makes sense (where data resides)
- Make computing architectures more data-centric

Goal: In-Memory Computation Engine

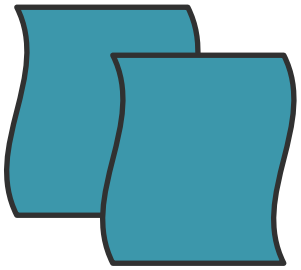


- Many questions … How do we design the:
  - compute-capable memory?
  - processor chip?
  - software interface?
  - system software and languages?
  - algorithms?

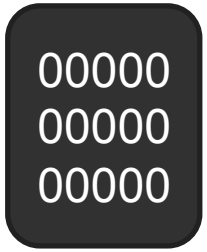


Starting Simple: Data Copy and Initialization

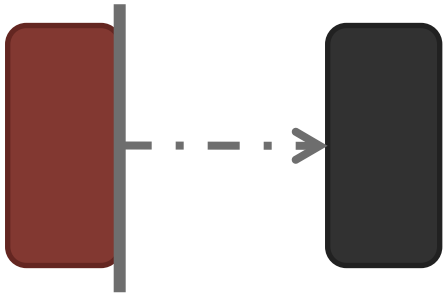
memmove & memcpy: 5% cycles in Google's datacenter [Kanev+ ISCA'15]



Forking

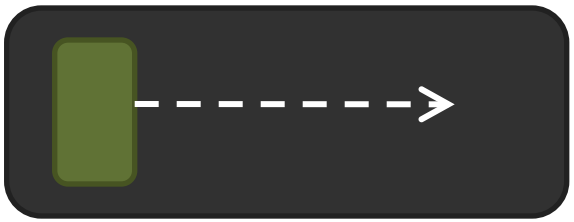


Zero initialization (e.g., security)



Checkpointing

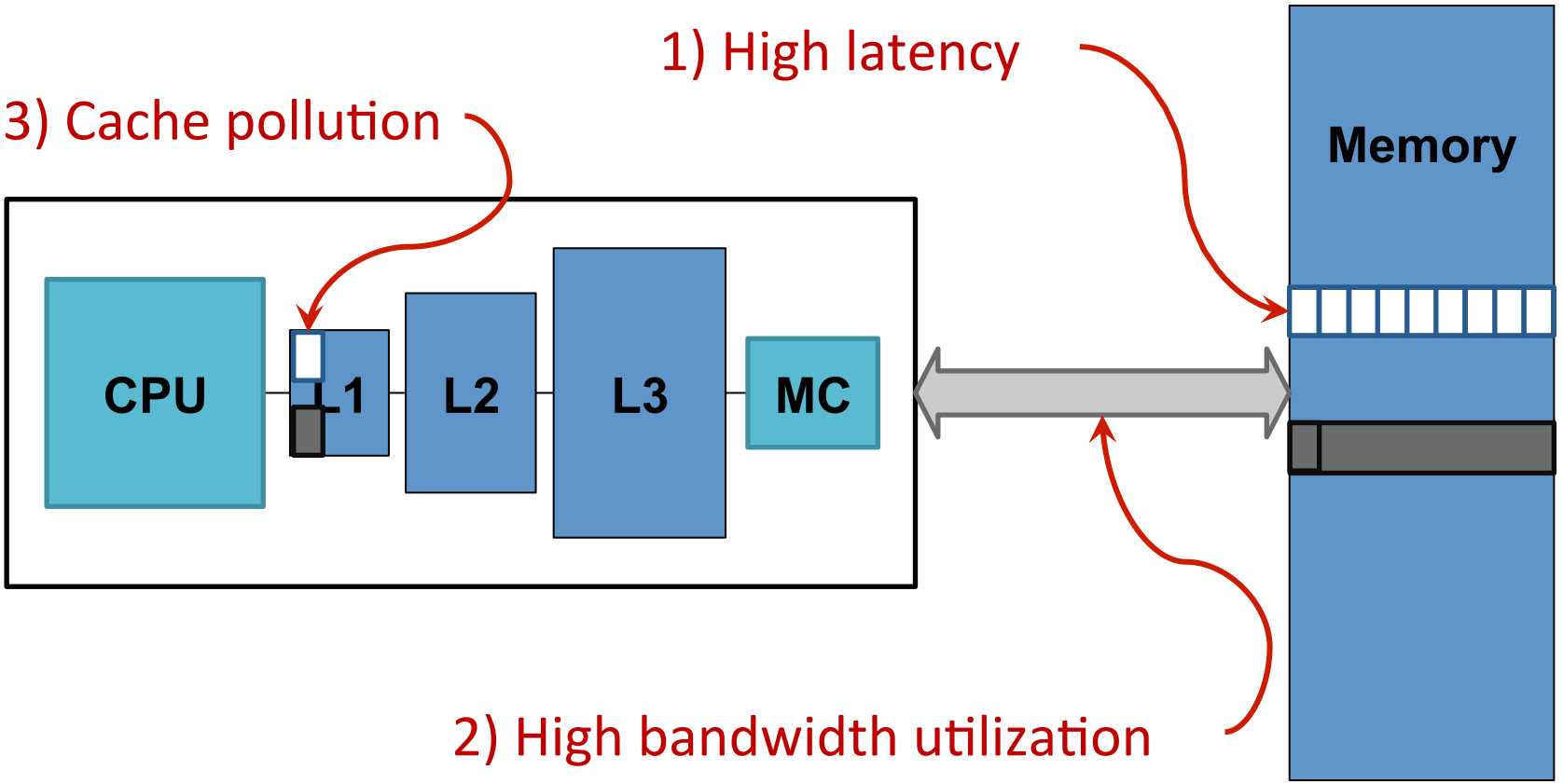
VM Cloning
Deduplication



Page Migration

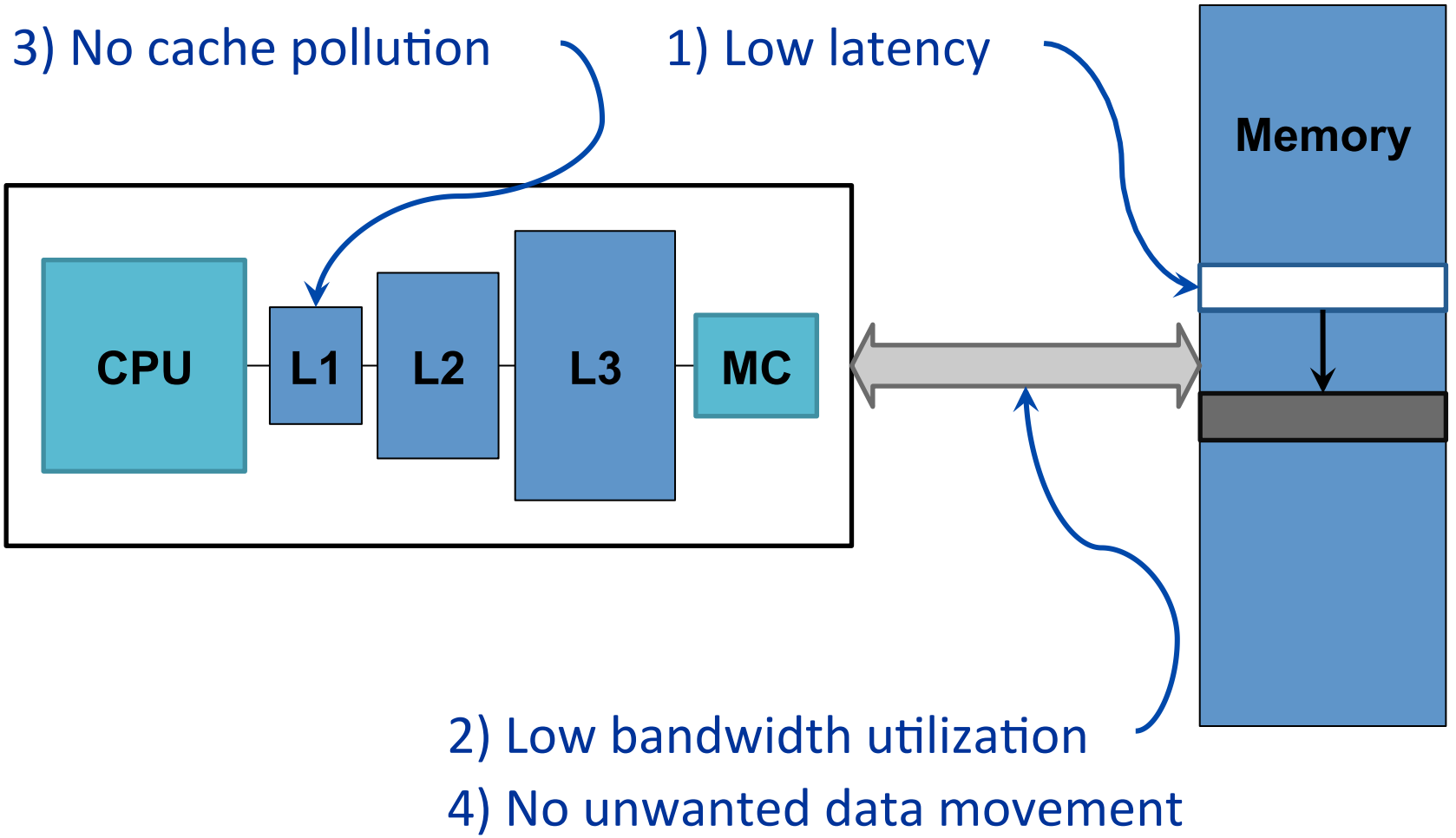
• • •
Many more

Today's Systems: Bulk Data Copy



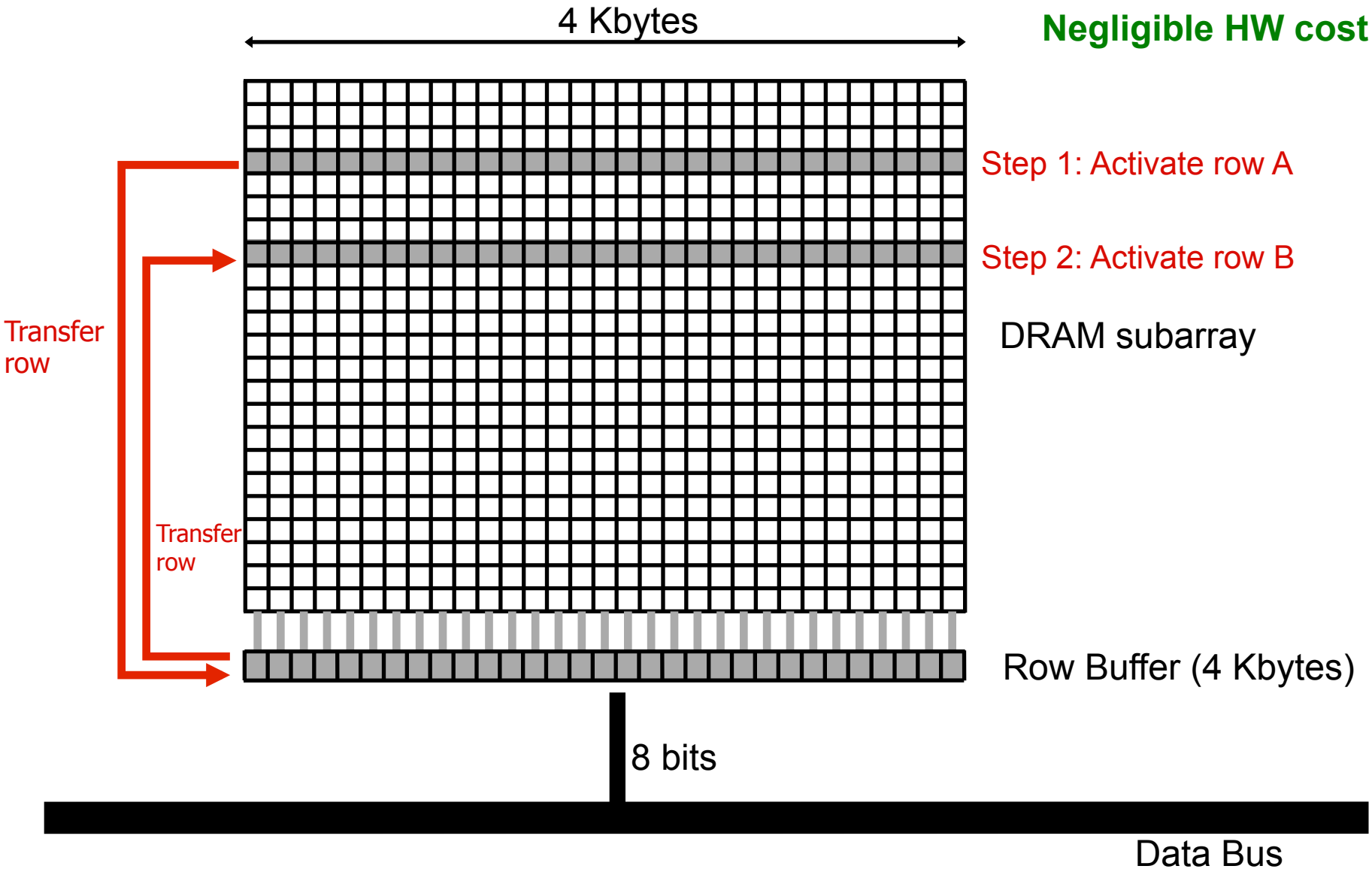
1046ns, 3.6uJ (for 4KB page copy via DMA)

Future Systems: In-Memory Copy

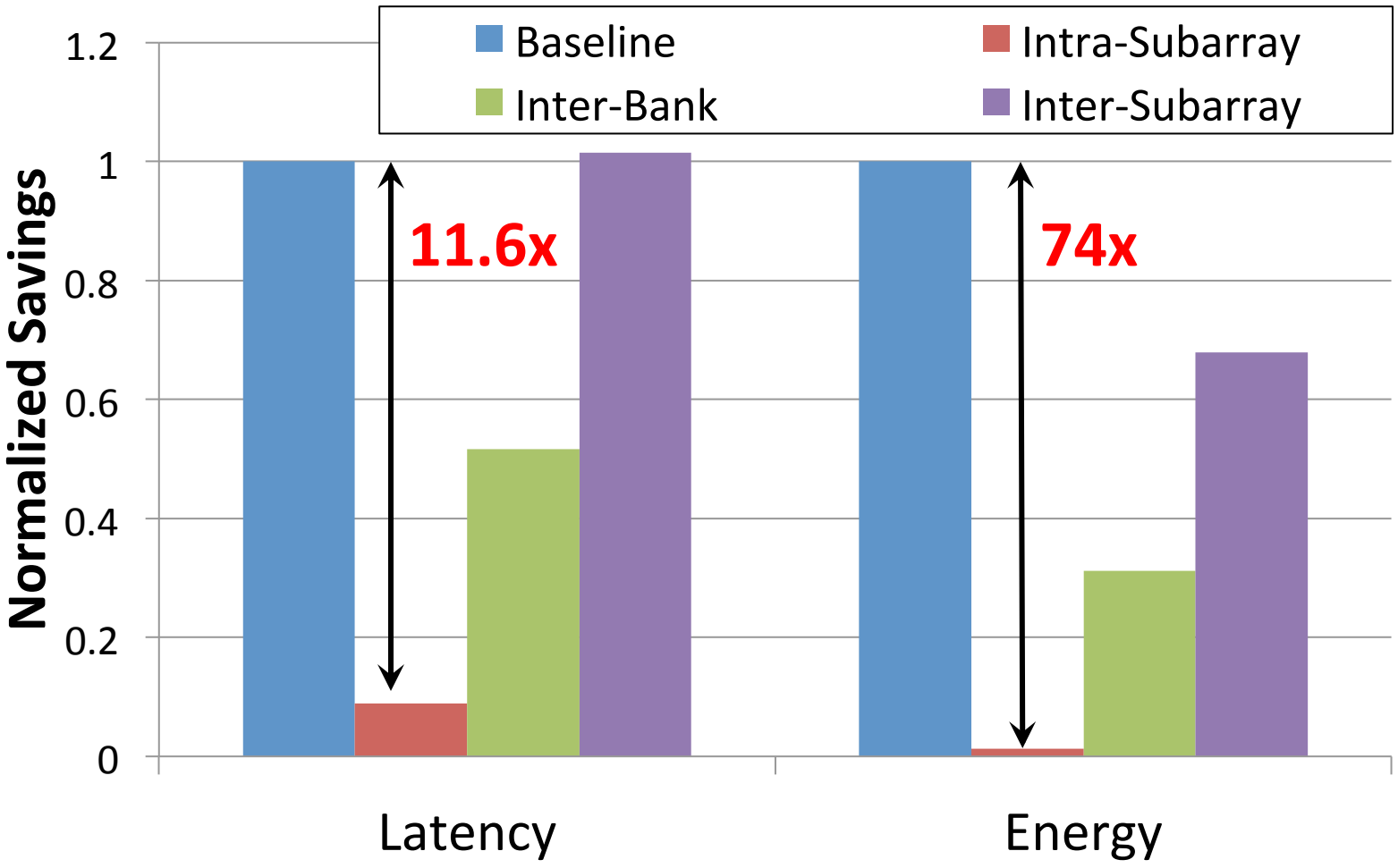


RowClone: In-DRAM Row Copy

Idea: Two consecutive ACTivates
Negligible HW cost



RowClone: Latency and Energy Savings



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

(Truly) In-Memory Computation

- Similarly, we can support in-DRAM AND, OR, NOT, MAJ
- At low cost
- Using analog behavior of memory
- 30-60X performance and energy improvement
  - Seshadri+, "In-DRAM Bulk Bitwise AND and OR," CAL 2016.
  - Seshadri+, "Buddy-RAM: Improving the Performance and Efficiency of Bulk Bitwise Operations Using DRAM," arxiv 2016.

- New memory technologies enable even more opportunities
  - Memristors, resistive RAM, phase change mem, STT-MRAM, …
  - Can operate on data with minimal movement

Another Example: In-Memory Graph Processing

- Large graphs are everywhere (circa 2015)

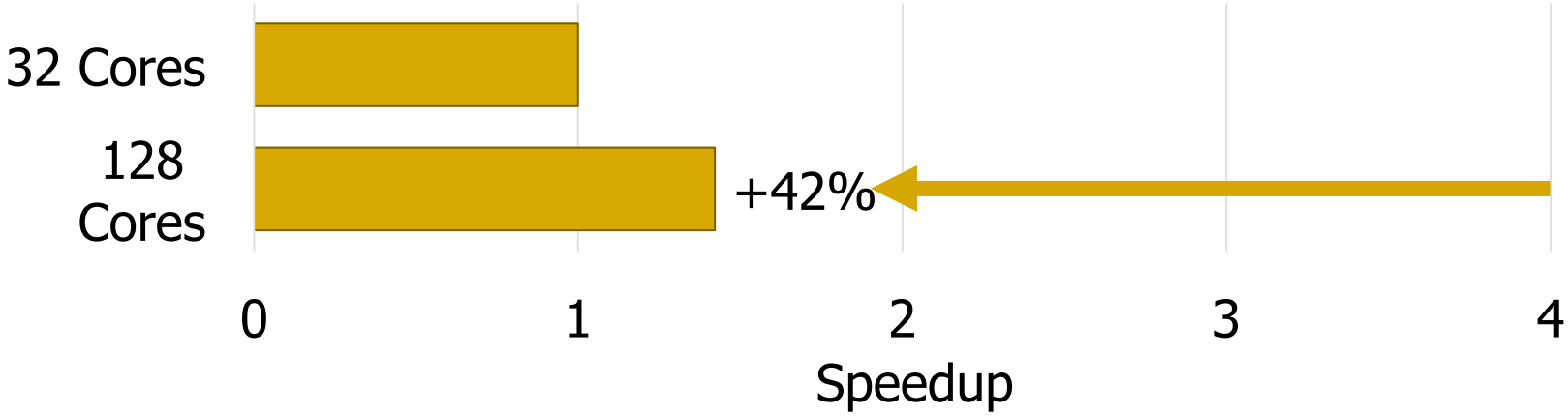
36 Million
Wikipedia Pages

1.4 Billion
Facebook Users

300 Million
Twitter Users

30 Billion
Instagram Photos

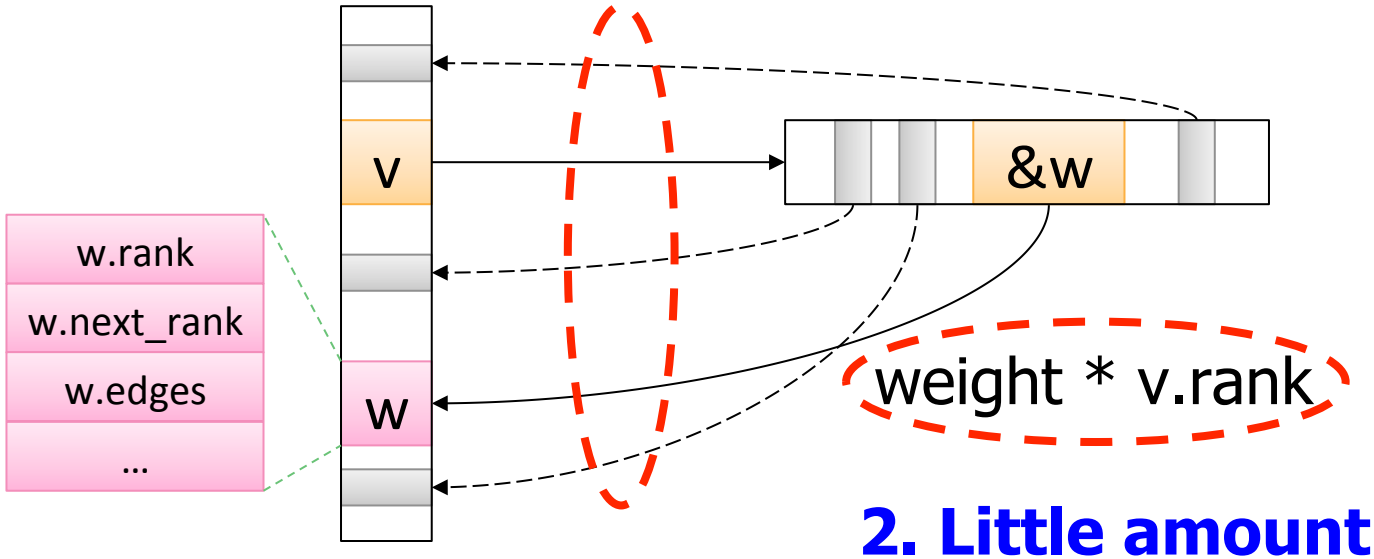
- Scalable large-scale graph processing is challenging



Key Bottlenecks in Graph Processing

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

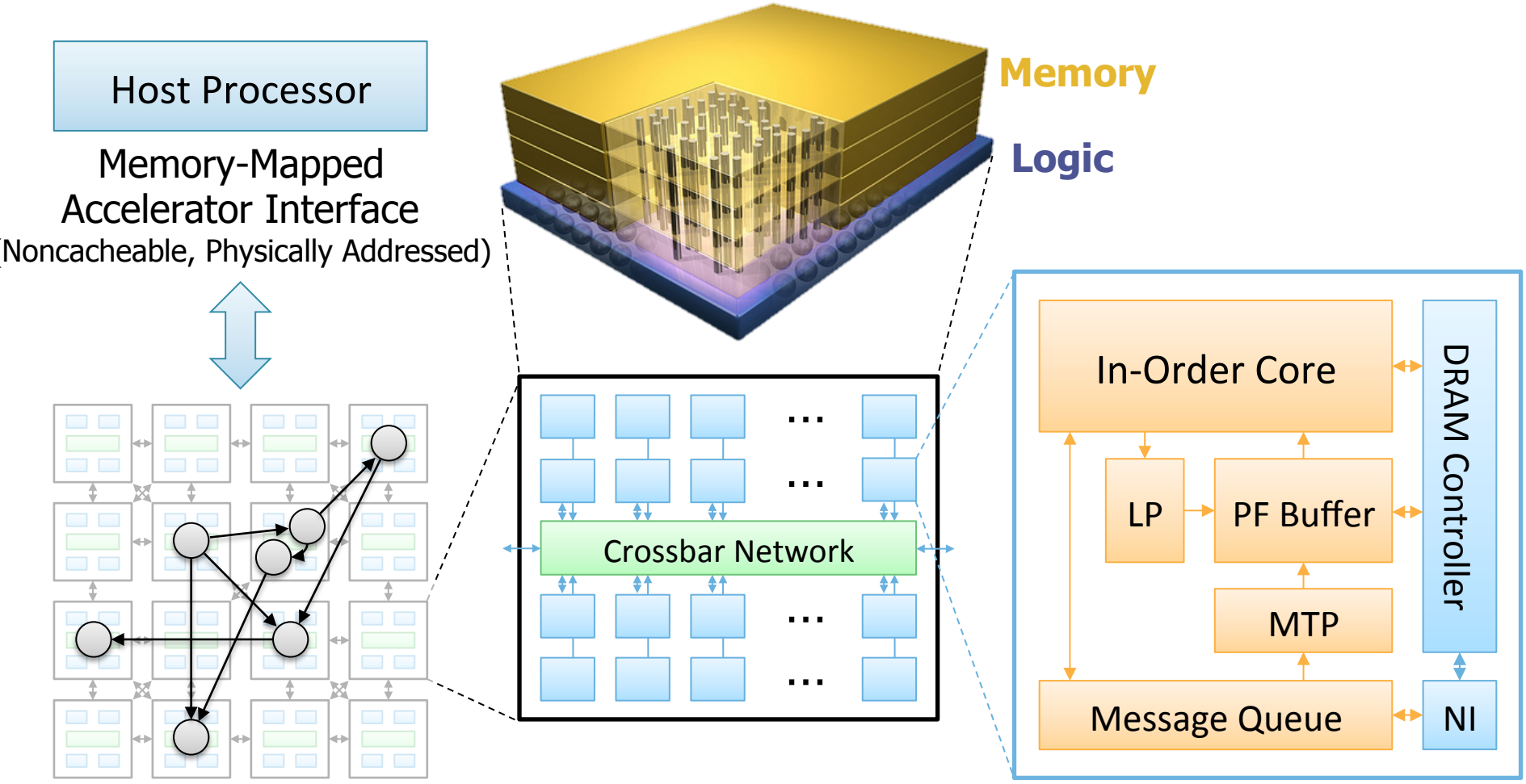
1. Frequent random memory accesses



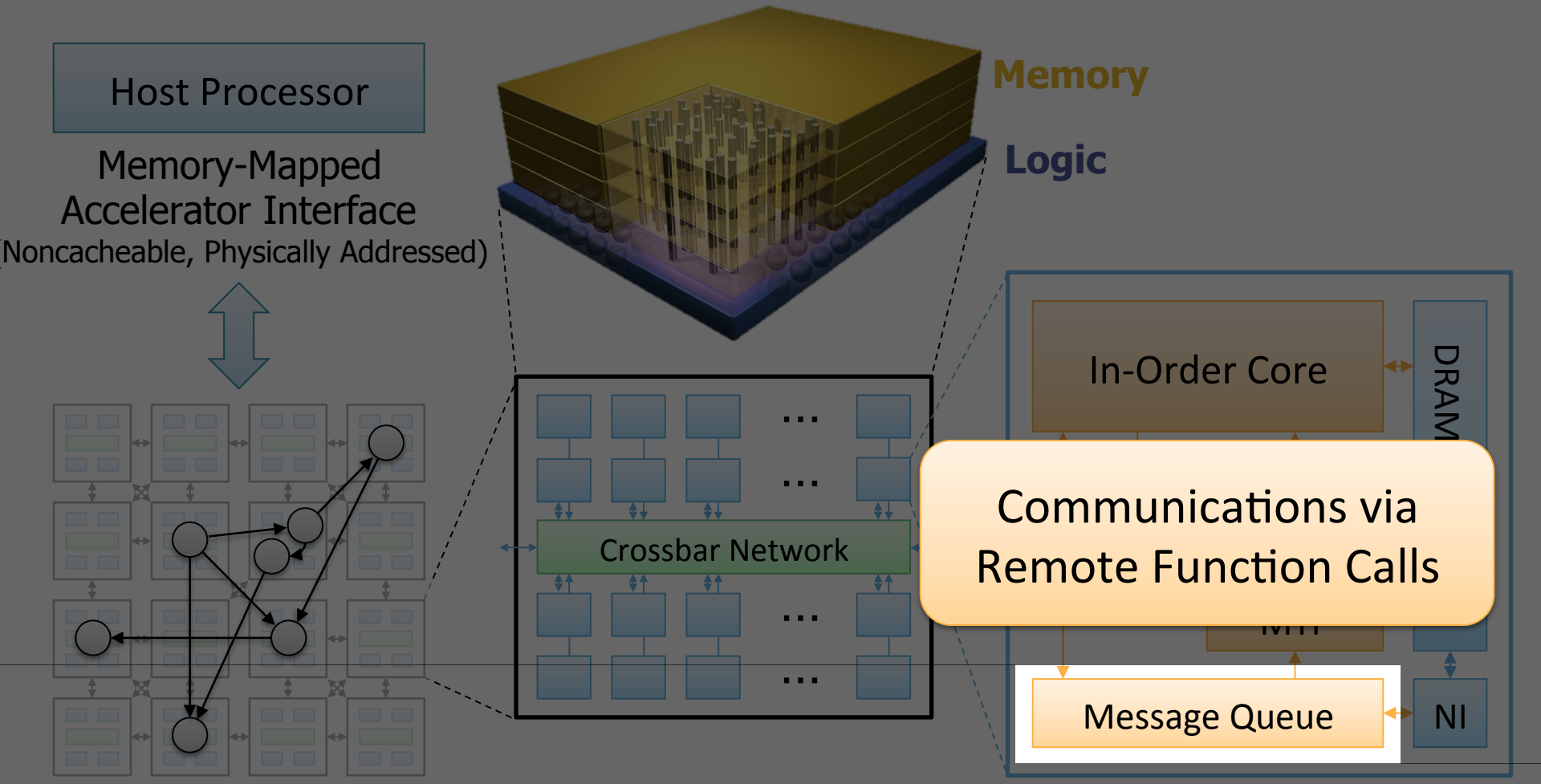
2. Little amount of computation

Tesseract System for Graph Processing

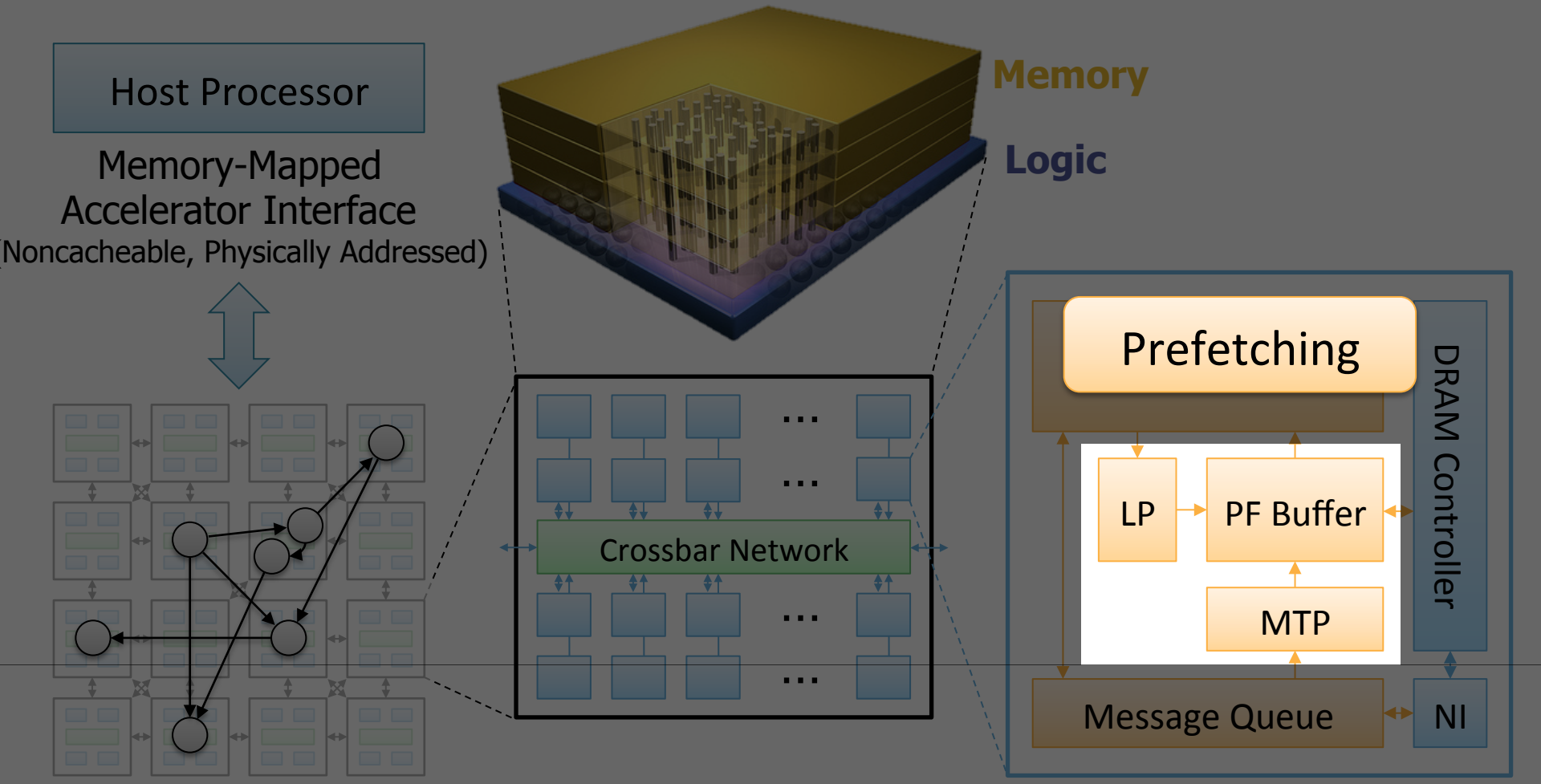
Interconnected set of 3D-stacked memory+logic chips with simple cores



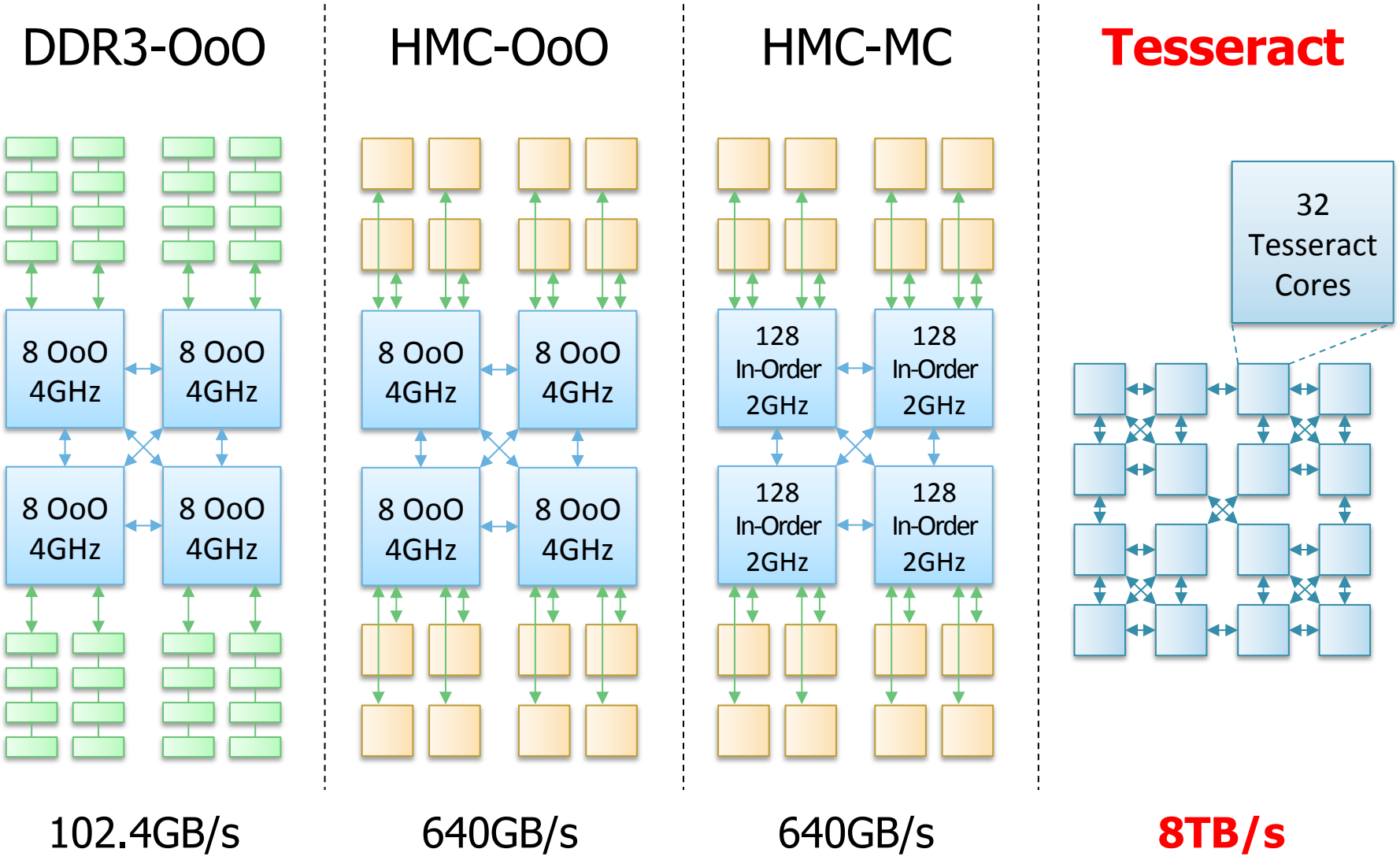
Tesseract System for Graph Processing



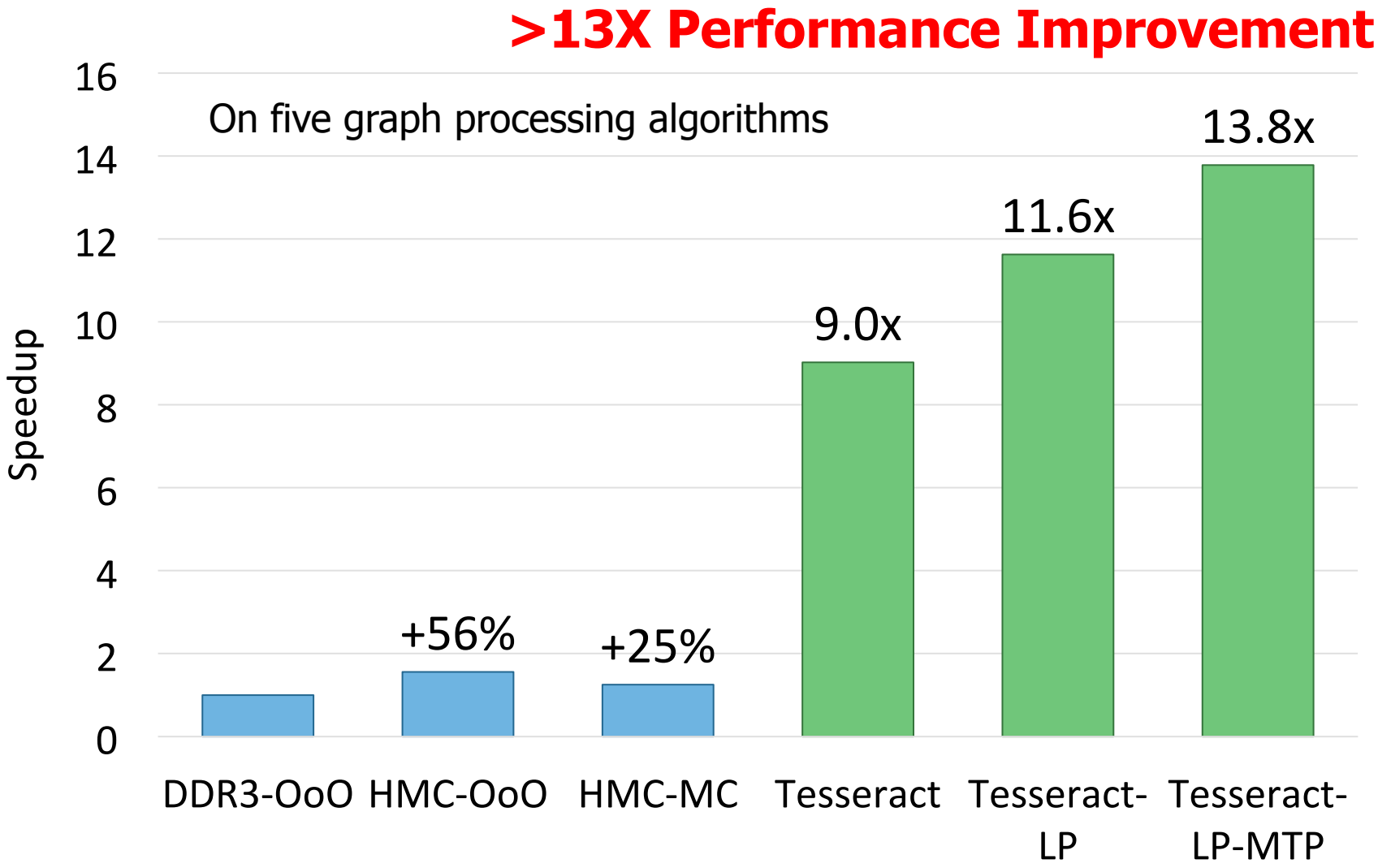
Tesseract System for Graph Processing



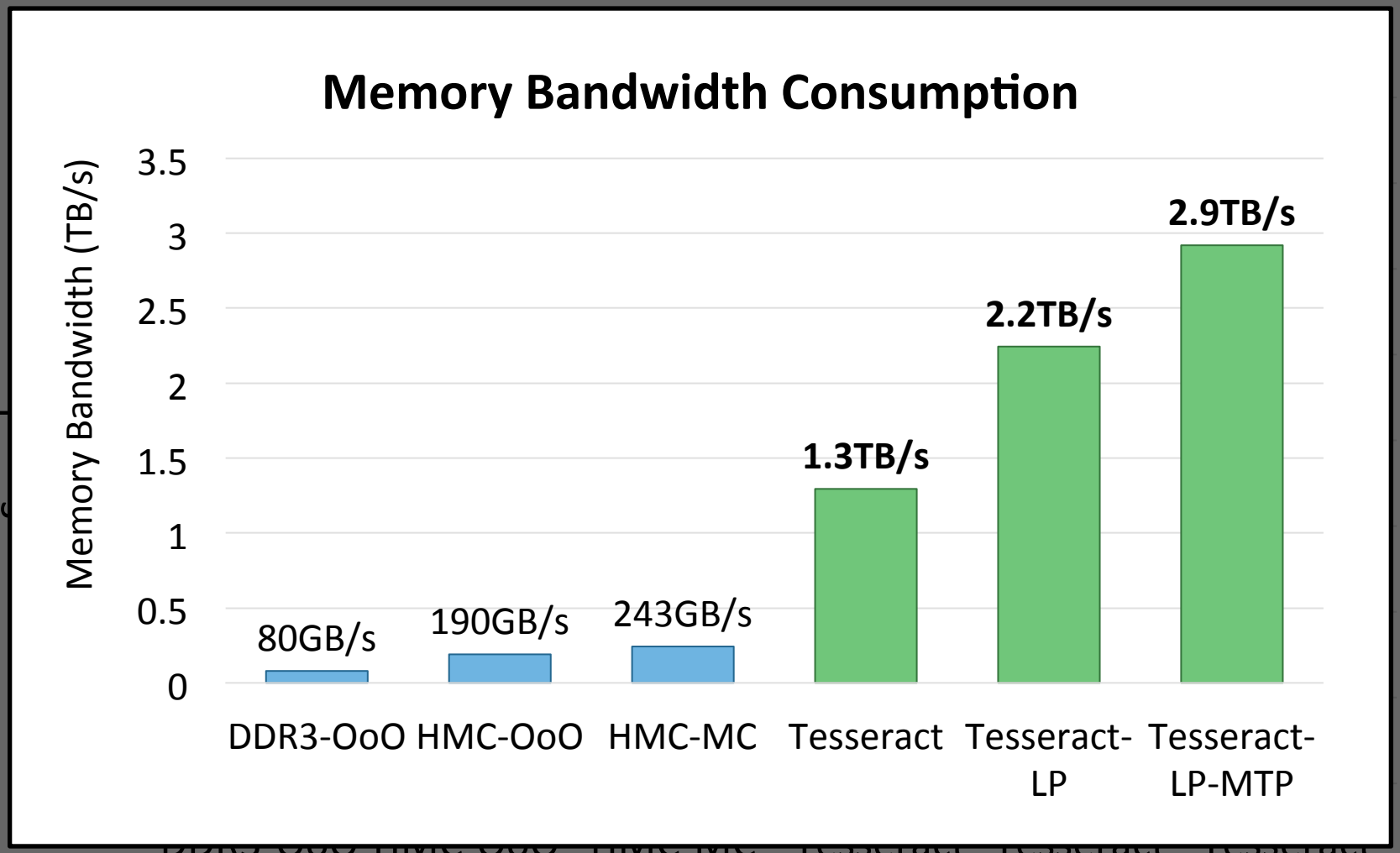
Evaluated Systems



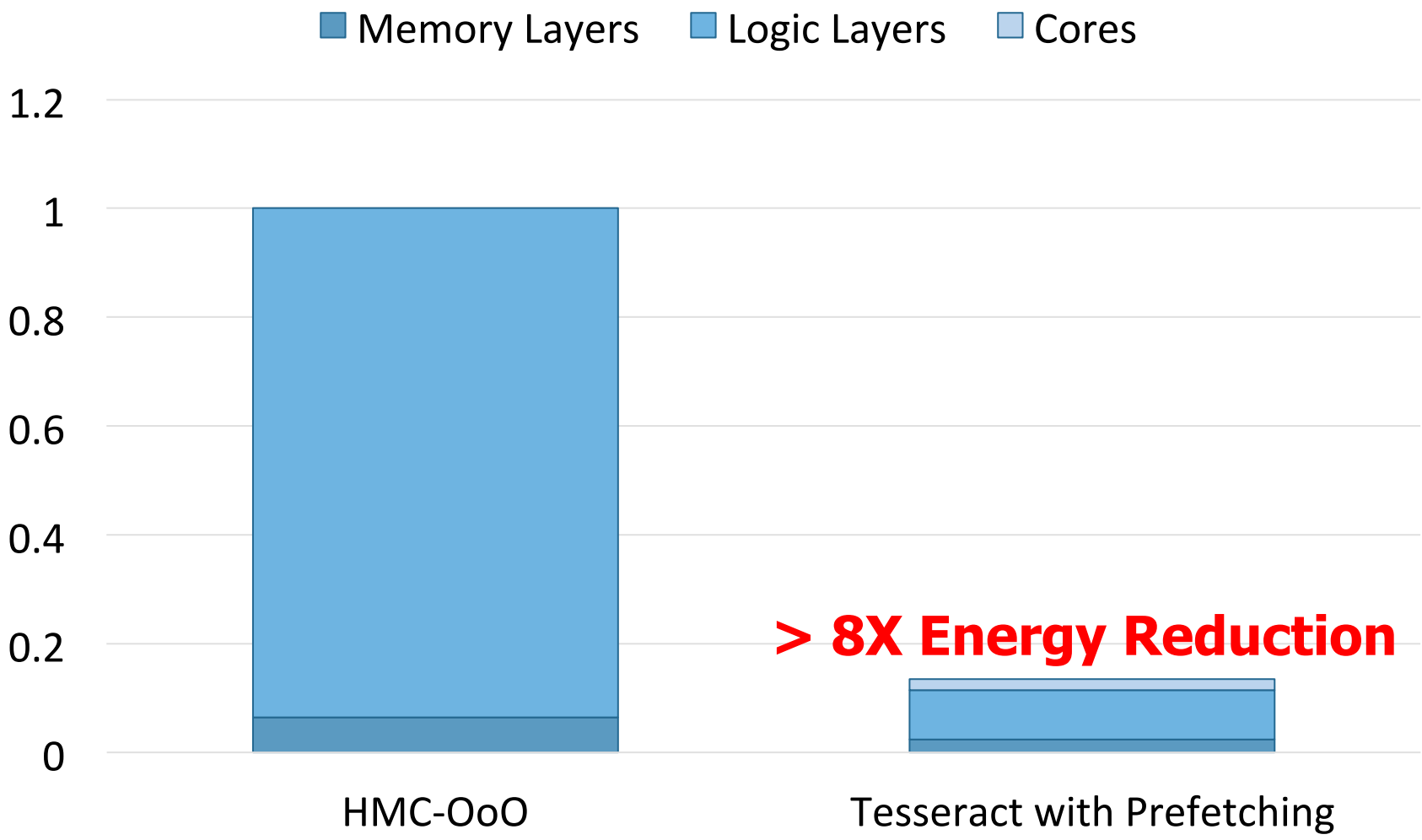
Tesseract Graph Processing Performance



Tesseract Graph Processing Performance



Tesseract Graph Processing Energy



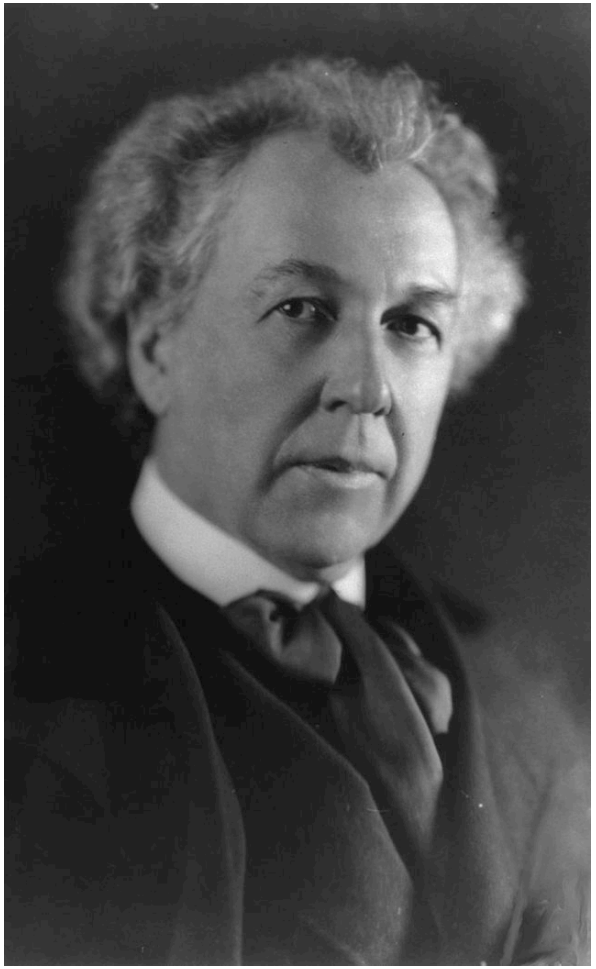
Challenge and Opportunity for Future

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures

Concluding Remarks

A Quote from A Famous Architect

- "architecture [...] based upon principle, and not upon precedent"



Precedent-Based Design?

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

Principled Design

- "architecture [...] based upon principle, and not upon precedent"

Another Example: Precedent-Based Design

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

Principled Design

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

Principle Applied to Another Structure

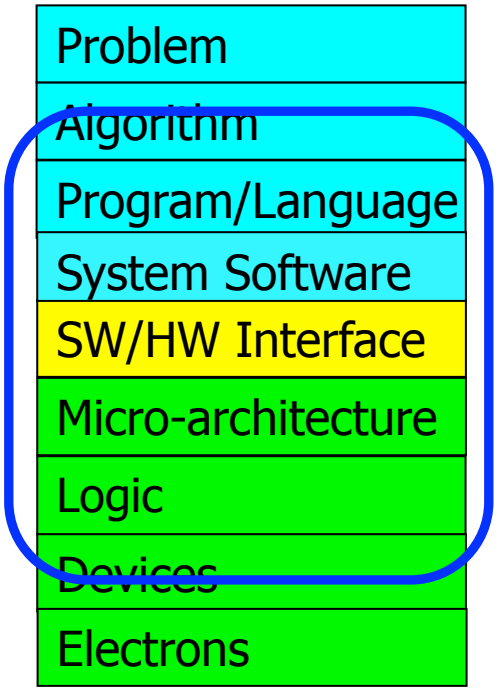
Source: By 準建築人手札網站 Forgemind ArchiMedia - Flickr: IMG_2489.JPG, CC BY 2.0, https://commons.wikimedia.org/w/index.php?curid=[illegible], https://en.wikipedia.org/wiki/Santiago_Calatrava

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

Concluding Remarks

- It is time to design principled system architectures to solve the memory scaling problem
- Discover design principles for fundamentally secure and reliable computer architectures
- Design complete systems to be balanced and energy-efficient, i.e., data-centric (or memory-centric)
- Enable new and emerging memory architectures
- This can
  - Lead to **orders-of-magnitude** improvements
  - **Enable new applications & computing platforms**
  - **…**

Concluding Remarks



Rethinking Memory System Design

(and the Platforms We Design Around It)

Onur Mutlu
onur.mutlu@inf.ethz.ch
https://people.inf.ethz.ch/omutlu

June 5, 2017

Technion Seiden Workshop: Beyond CMOS

SAFARI

ETH zürich

Acknowledgments

- My current and past students and postdocs
  - Rachata Ausavarungnirun, Abhishek Bhowmick, Amirali Boroumand, Rui Cai, Yu Cai, Kevin Chang, Saugata Ghose, Kevin Hsieh, Tyler Huberty, Ben Jaiyen, Samira Khan, Jeremie Kim, Yoongu Kim, Yang Li, Jamie Liu, Lavanya Subramanian, Donghyuk Lee, Yixin Luo, Justin Meza, Gennady Pekhimenko, Vivek Seshadri, Lavanya Subramanian, Nandita Vijaykumar, HanBin Yoon, Jishen Zhao, …
- My collaborators
  - Can Alkan, Chita Das, Phil Gibbons, Sriram Govindan, Norm Jouppi, Mahmut Kandemir, Mike Kozuch, Konrad Lai, Ken Mai, Todd Mowry, Yale Patt, Moinuddin Qureshi, Partha Ranganathan, Bikash Sharma, Kushagra Vaid, Chris Wilkerson, …

Funding Acknowledgments

- NSF
- GSRC
- SRC
- CyLab
- AMD, Google, Facebook, HP Labs, Huawei, IBM, Intel, Microsoft, Nvidia, Oracle, Qualcomm, Rambus, Samsung, Seagate, VMware

Slides Not Covered But Could Be Useful

Recap: The DRAM Scaling Problem

DRAM Process Scaling Challenges

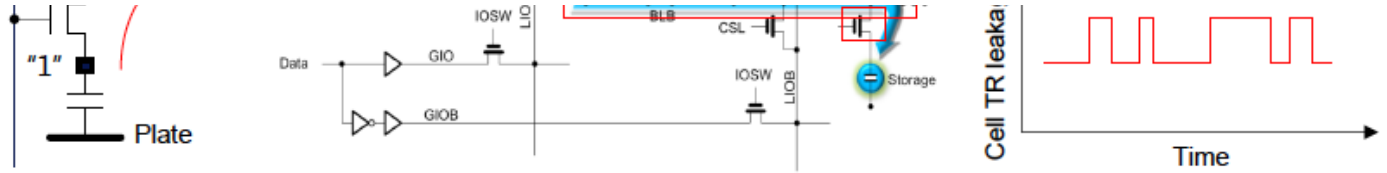
- **Refresh**
  - Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance

THE MEMORY FORUM 2014

Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling

Uksong Kang, Hak-soo Yu, Churoo Park, *Hongzhong Zheng,
**John Halbert, **Kuljit Bains, SeongJin Jang, and Joo Sun Choi

*Samsung Electronics, Hwasung, Korea / *Samsung Electronics, San Jose / **Intel*



Refresh **tWR** **VRT**

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
  - Reduce energy
  - Improve latency and bandwidth
  - Reduce waste (capacity, bandwidth, latency)
  - Enable computation close to data

Solution 1: New Memory Architectures

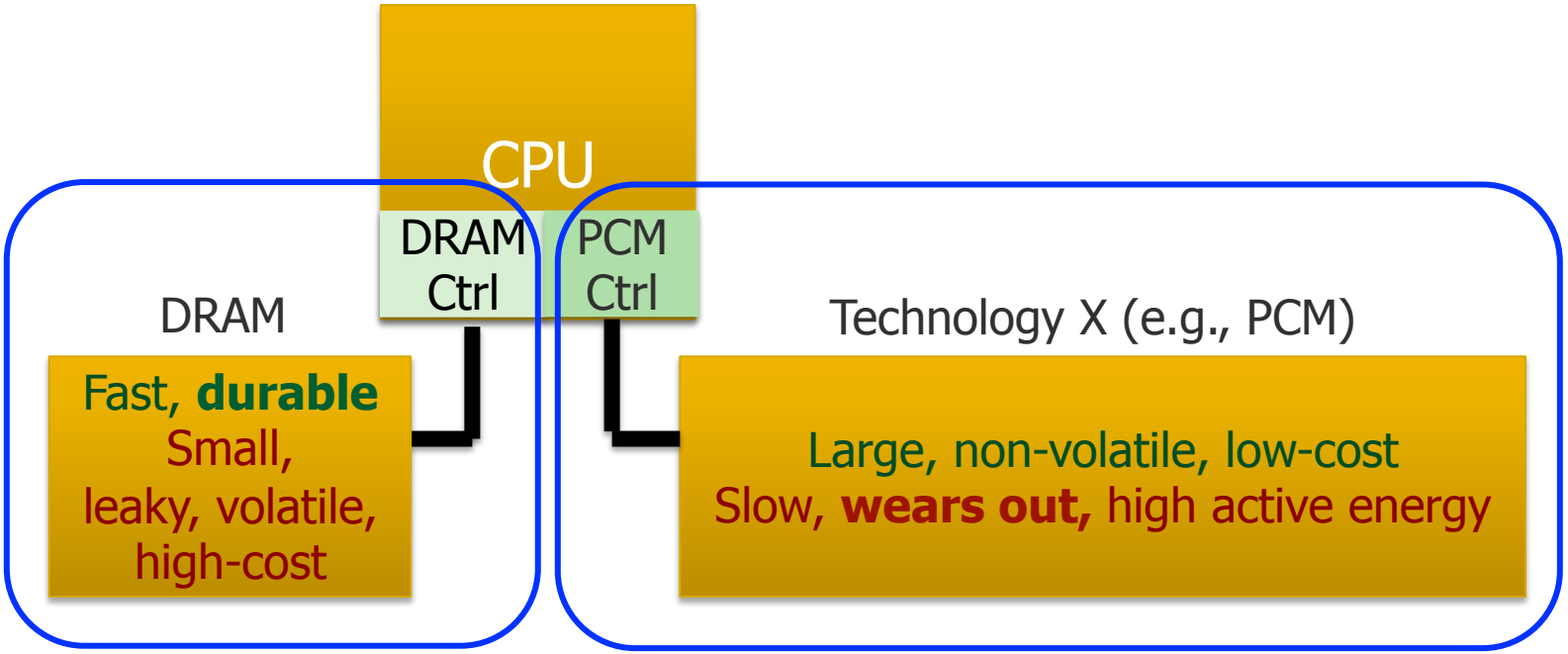
- Liu+, "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
- Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.
- Lee+, "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
- Liu+, "An Experimental Study of Data Retention Behavior in Modern 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.
- 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
  - Expected to scale to 9nm (2022 [ITRS])
  - Expected to be denser than DRAM: can store multiple bits/cell
- But, emerging technologies have shortcomings as well
  - Can they be enabled to replace/augment/surpass DRAM?

- 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.

Solution 3: Hybrid Memory Systems

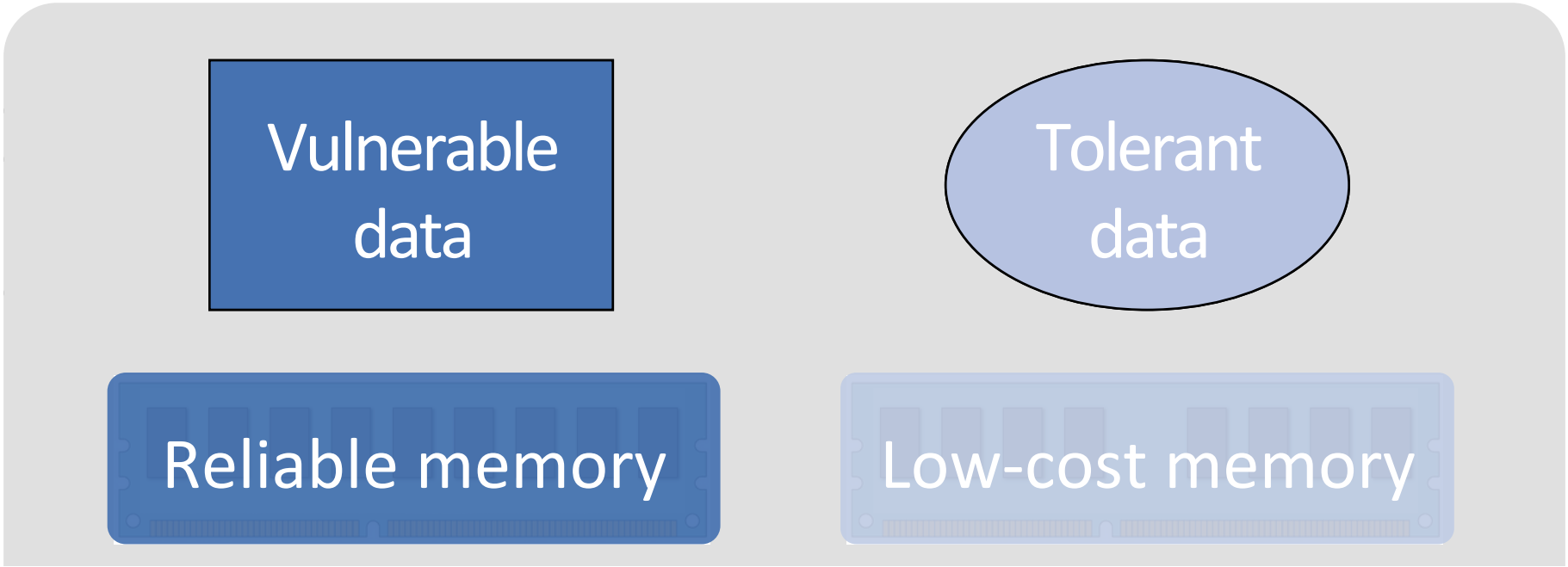


Hardware/software manage data allocation and movement
to achieve the best of multiple technologies

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

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

Exploiting Memory Error Tolerance with Hybrid Memory Systems



On Microsoft's Web Search workload

Reduces server hardware cost by 4.7 %

Achieves single server availability target of 99.90 %

Heterogeneous-**R**eliability **M**emory [DSN 2014]

Challenge and Opportunity

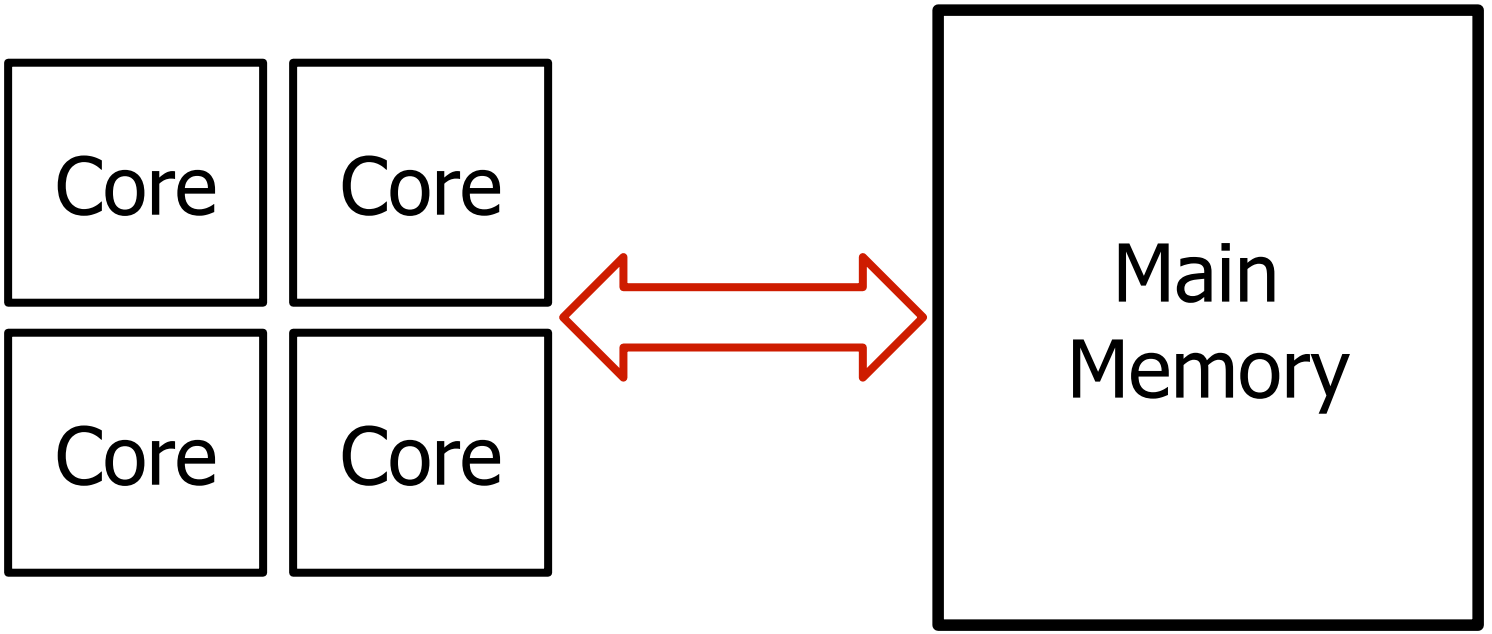
Providing the Best of Multiple Metrics

Departing From "Business as Usual"

Heterogeneous Memory Systems

Configurable Memory Systems

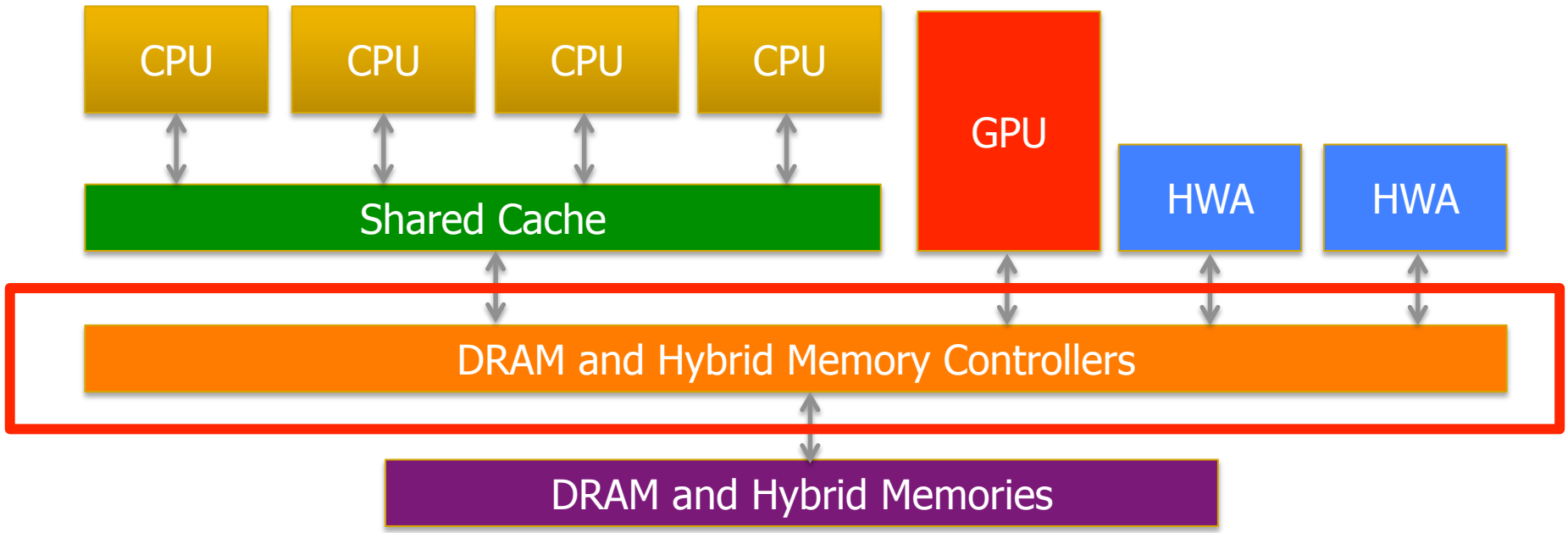
An Orthogonal Issue: Memory Interference



Cores' interfere with each other when accessing shared main memory

This is uncontrolled today → Unpredictable, uncontrollable system

Goal: Predictable Performance in Complex Systems



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs

How to allocate resources to heterogeneous agents to mitigate interference and provide predictable performance?

QoS-Aware Memory Systems

- Solution: QoS-Aware Memory Systems
- Hardware provides a configurable QoS substrate
  - Application-aware memory scheduling, partitioning, throttling
- Software configures the substrate to satisfy various QoS goals
- QoS-aware memory systems provide predictable performance and higher efficiency

Subramanian et al., "MISE: Providing Performance Predictability and Improving Fairness in Shared Main Memory Systems," HPCA 2013.

Subramanian et al., "The Application Slowdown Model," MICRO 2015.

Challenge and Opportunity

Strong
Memory Service
Guarantees

Departing From "Business as Usual"

Predictable Memory Management

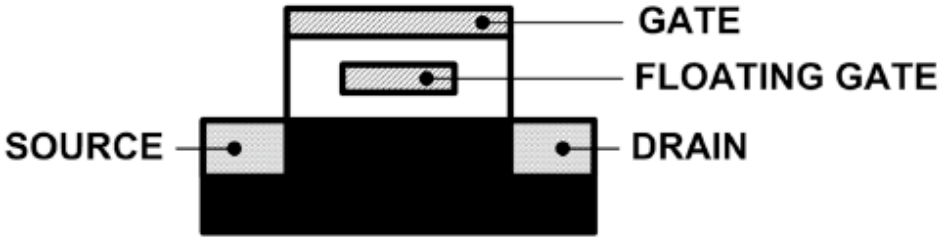
Programmable Memory Systems

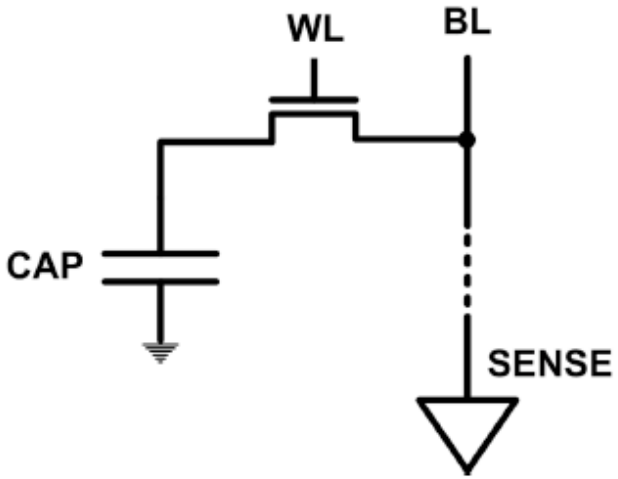
Some Promising Directions

- New memory architectures
  - Memory-centric system design
- Enabling and exploiting emerging NVM technologies
  - Hybrid memory systems
  - Unified interface to all data
- System-level QoS and predictability
  - Predictable systems with configurable QoS

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





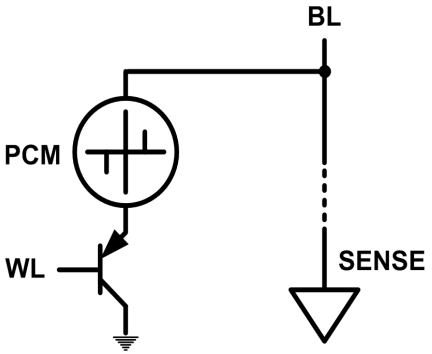
Promising Resistive Memory Technologies

- 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

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])
  - Prototyped at 20nm (Raoux+, IBM JRD 2008)
  - Expected to be denser than DRAM: can store multiple bits/cell



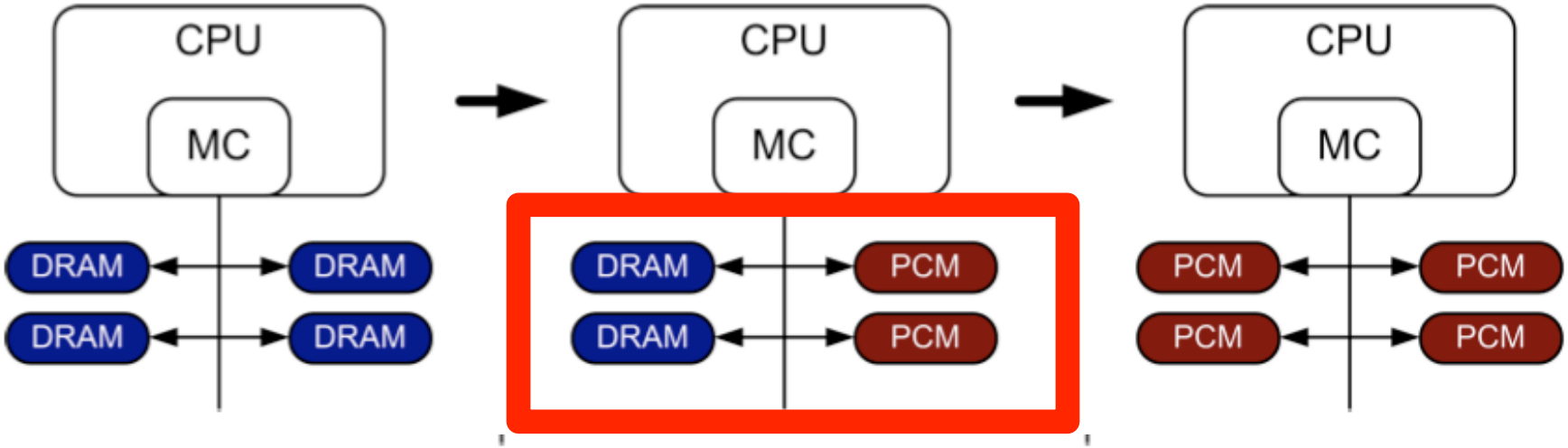
- But, emerging technologies have (many) shortcomings
  - Can they be enabled to replace/augment/surpass DRAM?

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)

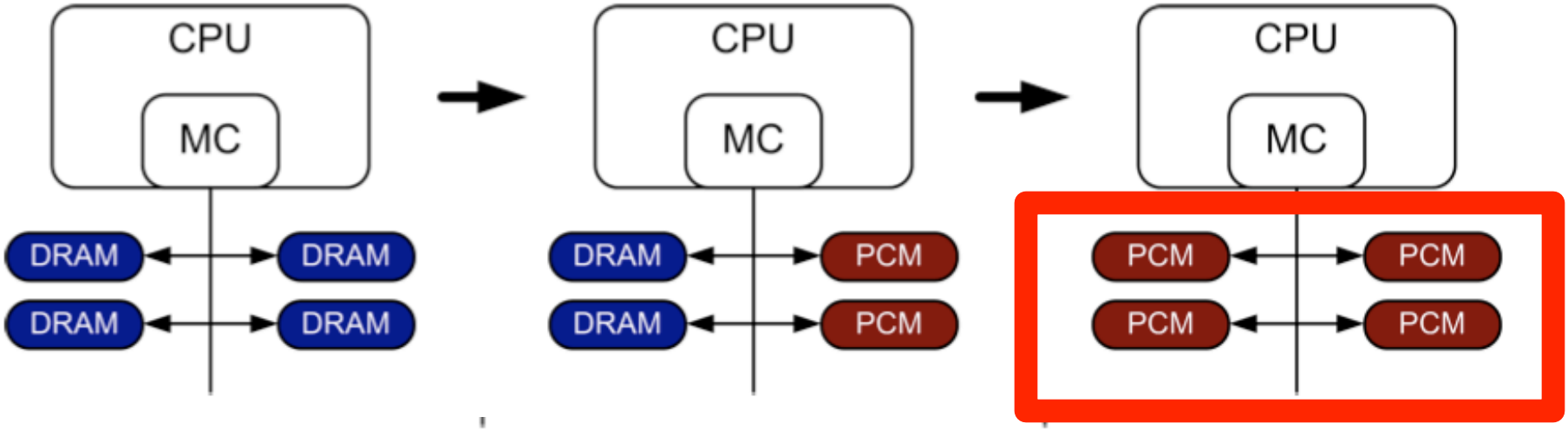
- How should PCM-based (main) memory be organized?



- 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

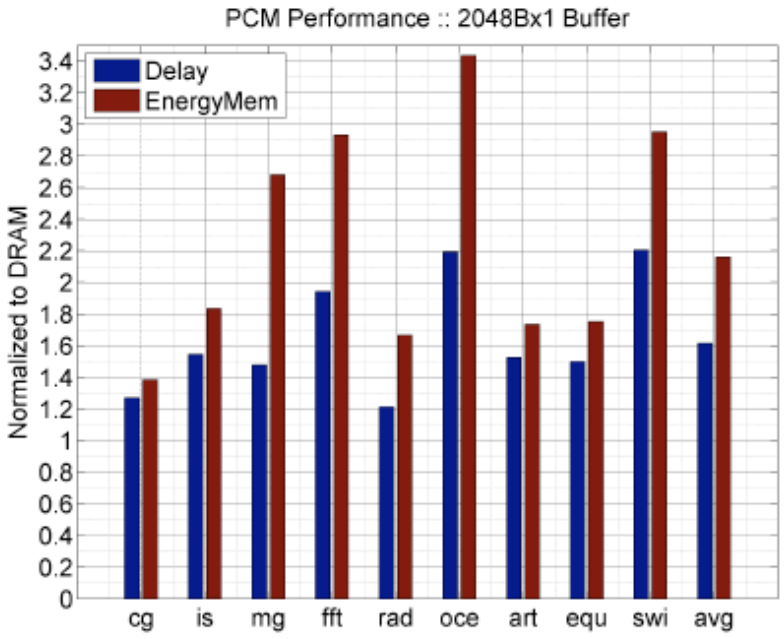
- 1E+08 writes
- 1E-08× DRAM

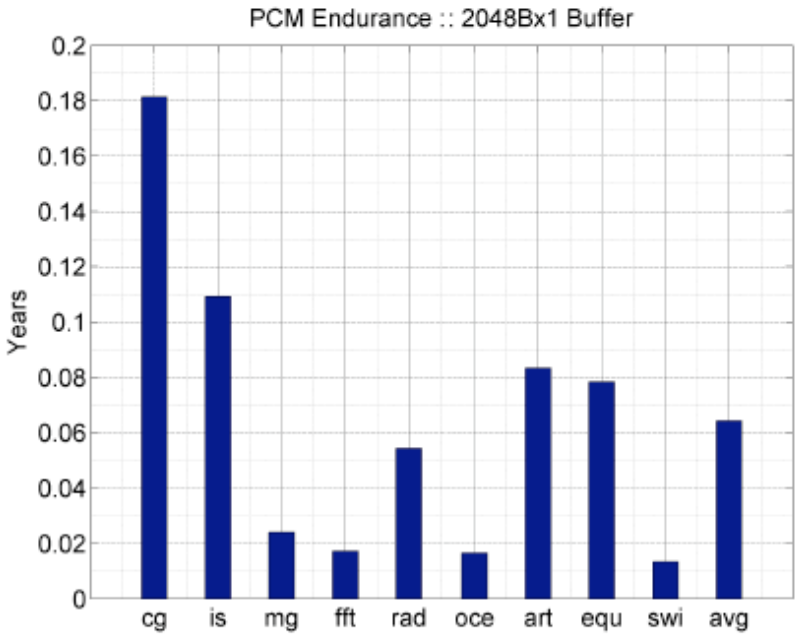
Energy

- 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

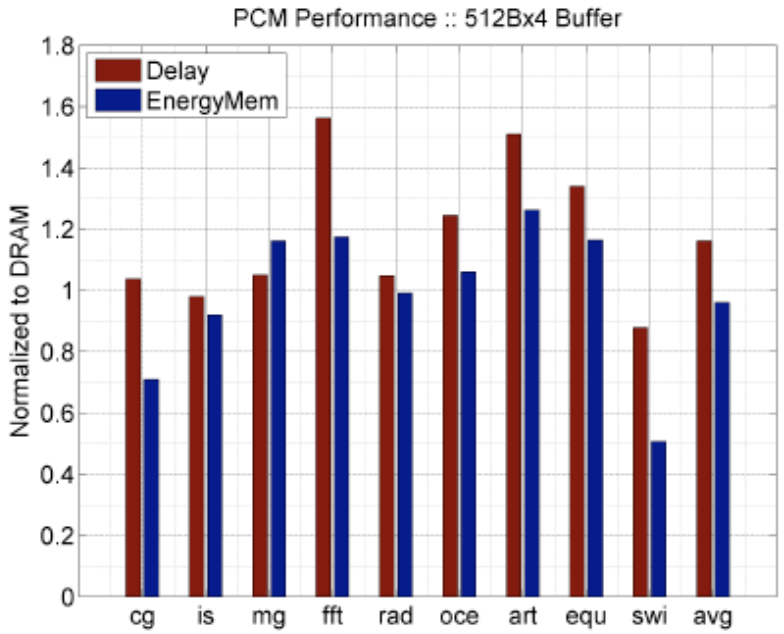


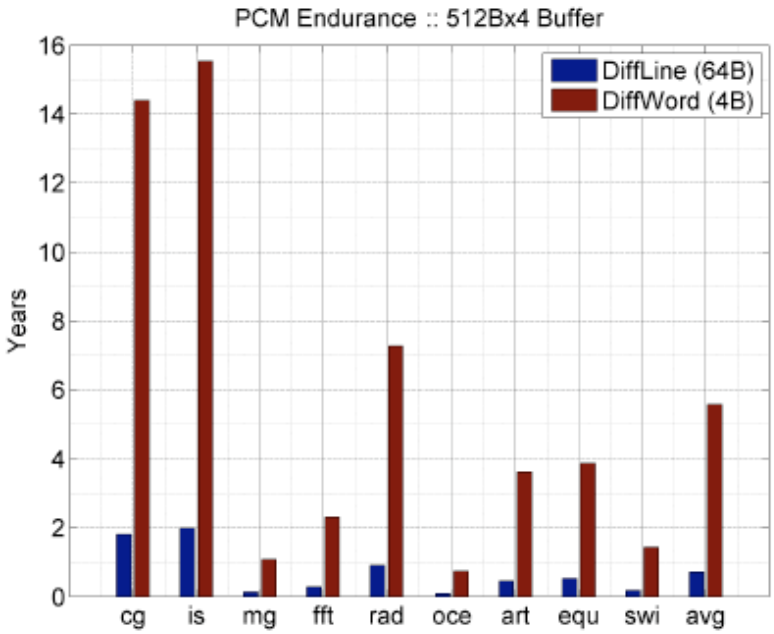


- Lee, Ipek, Mutlu, Burger, "Architecting Phase Change Memory as a Scalable DRAM Alternative," ISCA 2009.

Results: Architected PCM as Main Memory

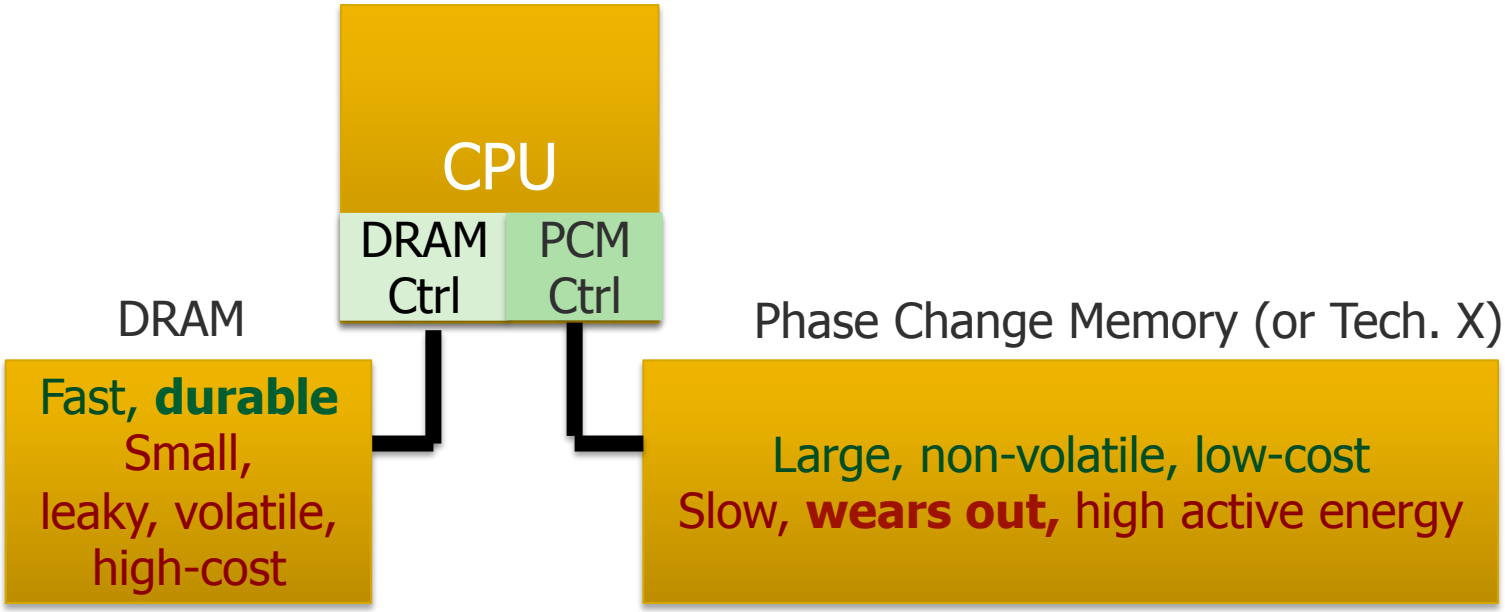
- 1.2x delay, 1.0x energy, 5.6-year average lifetime
- Scaling improves energy, endurance, density





- 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?

A More Viable Approach: Hybrid Memory Systems



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

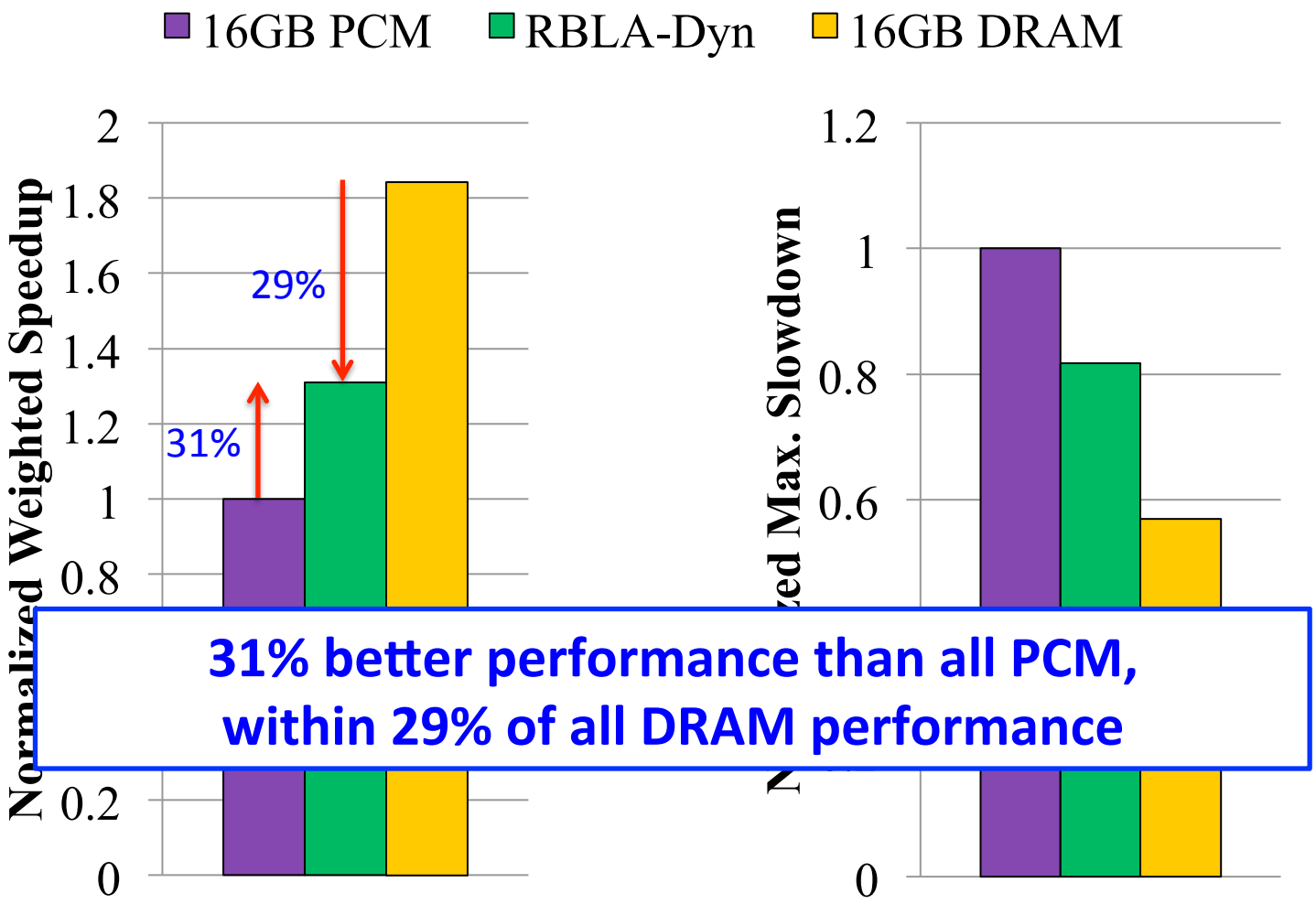
Meza+, "Enabling Efficient and Scalable Hybrid Memories," IEEE Comp. Arch. Letters, 2012.

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

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.

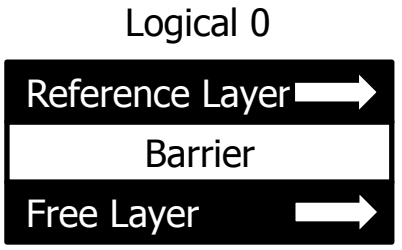
Hybrid vs. All-PCM/DRAM [ICCD'12]



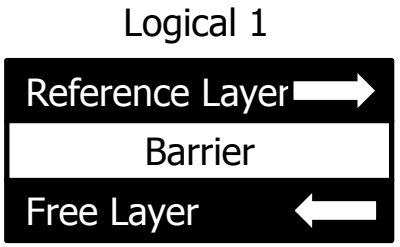
Yoon+, "Row Buffer Locality-Aware Data Placement in Hybrid Memories," ICCD 2012 Best Paper Award.

STT-MRAM as Main Memory

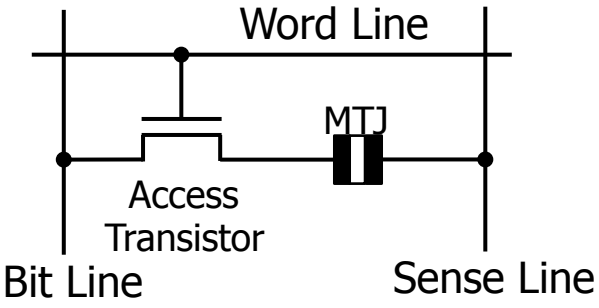
- Magnetic Tunnel Junction (MTJ) device
  - Reference layer: Fixed magnetic orientation
  - Free layer: Parallel or anti-parallel



- 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



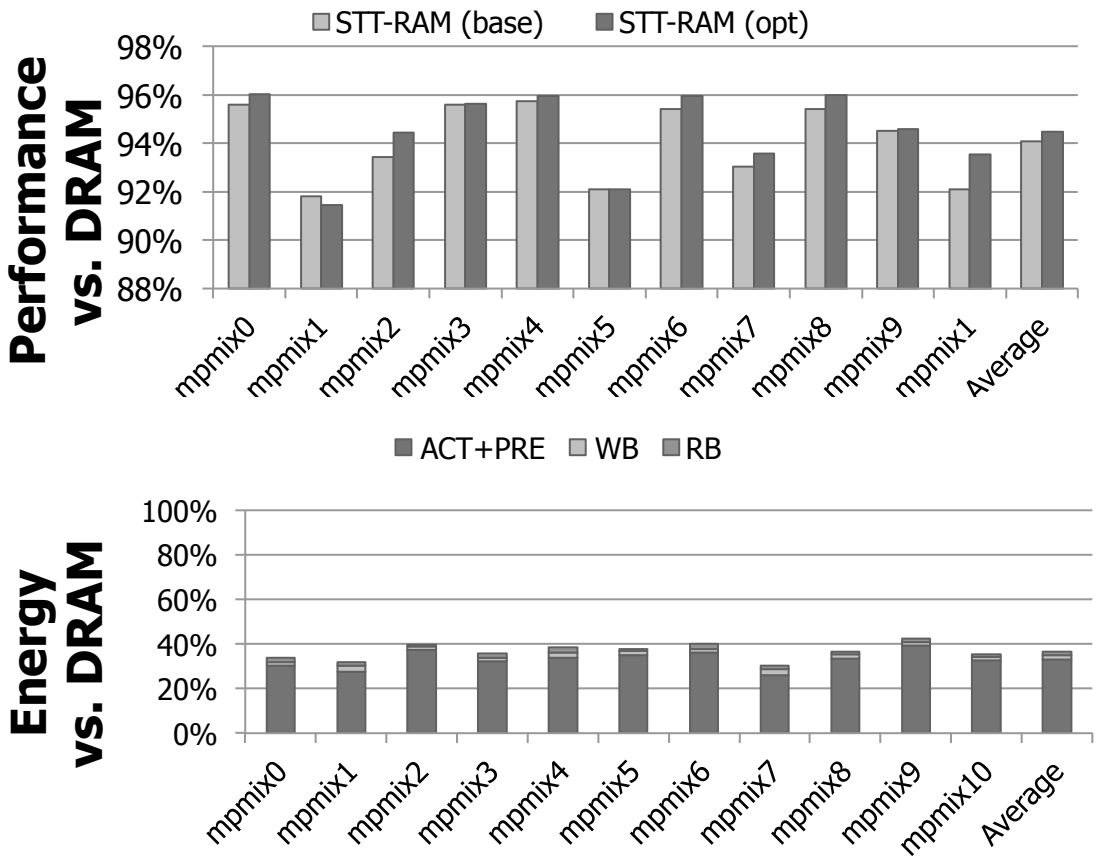
- Kultursay et al., "Evaluating STT-RAM as an Energy-Efficient Main Memory Alternative," ISPASS 2013.

STT-MRAM: Pros and Cons

- Pros over DRAM
  - Better technology scaling
  - Non volatility
  - Low idle power (no refresh)
- Cons
  - Higher write latency
  - Higher write energy
  - 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.

Challenge and Opportunity

Enabling an Emerging Technology to Replace DRAM

Departing From Business As Usual

Hybrid Memory

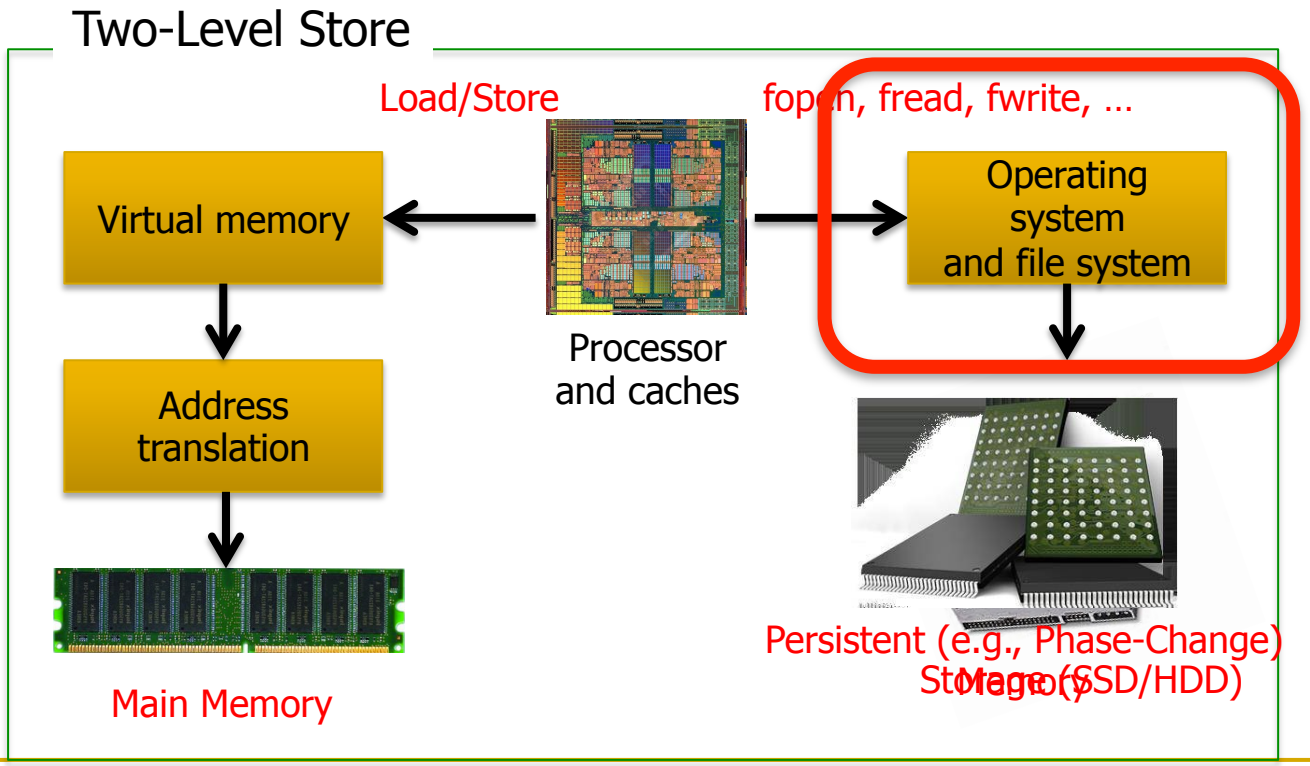
Persistent Memory

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

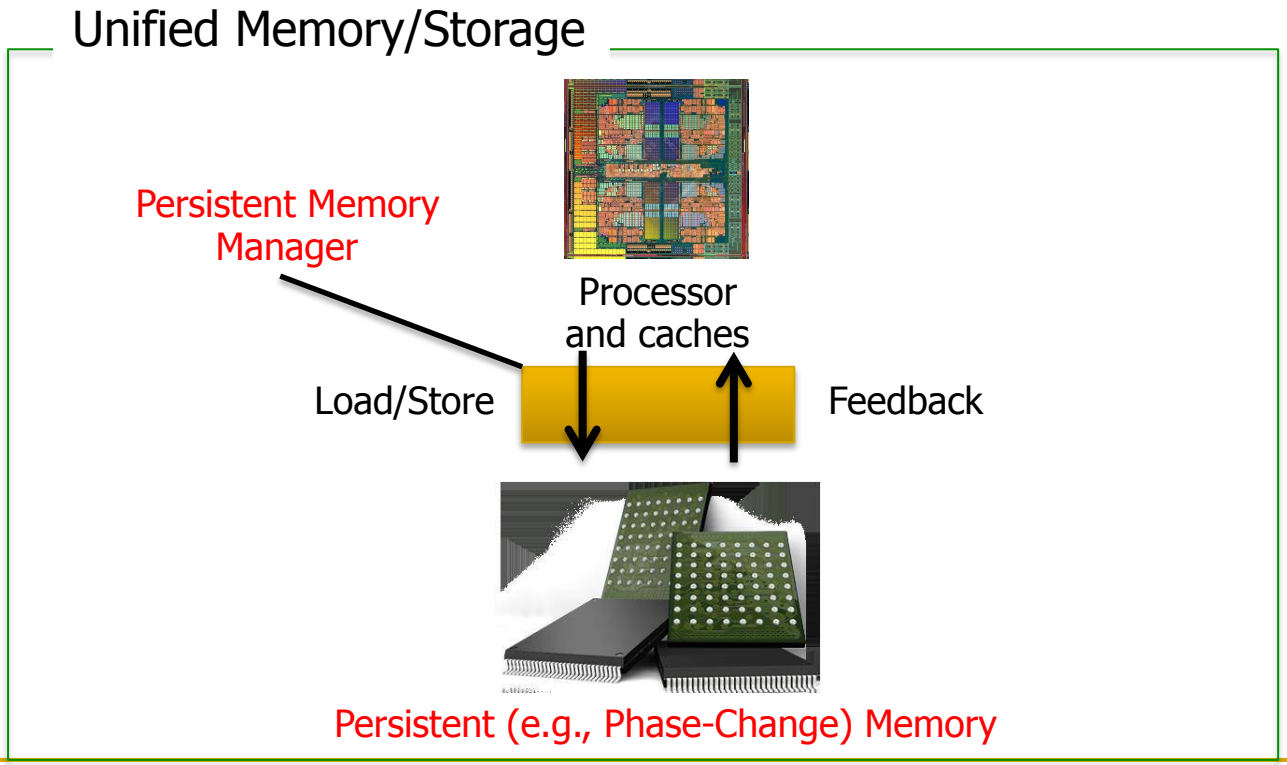
Coordinated Memory and Storage with NVM (I)

- 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



Coordinated Memory and Storage with NVM (II)

- 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

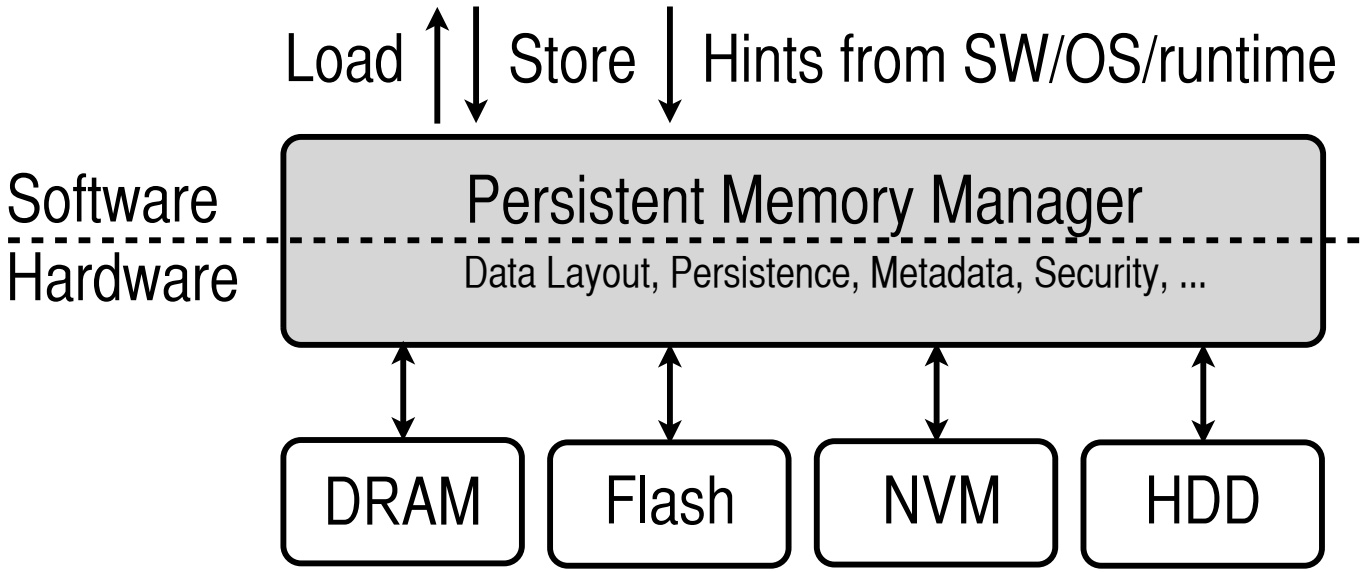


Meza+, "A Case for Efficient Hardware-Software Cooperative Management of Storage and Memory," WEED 2013.

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

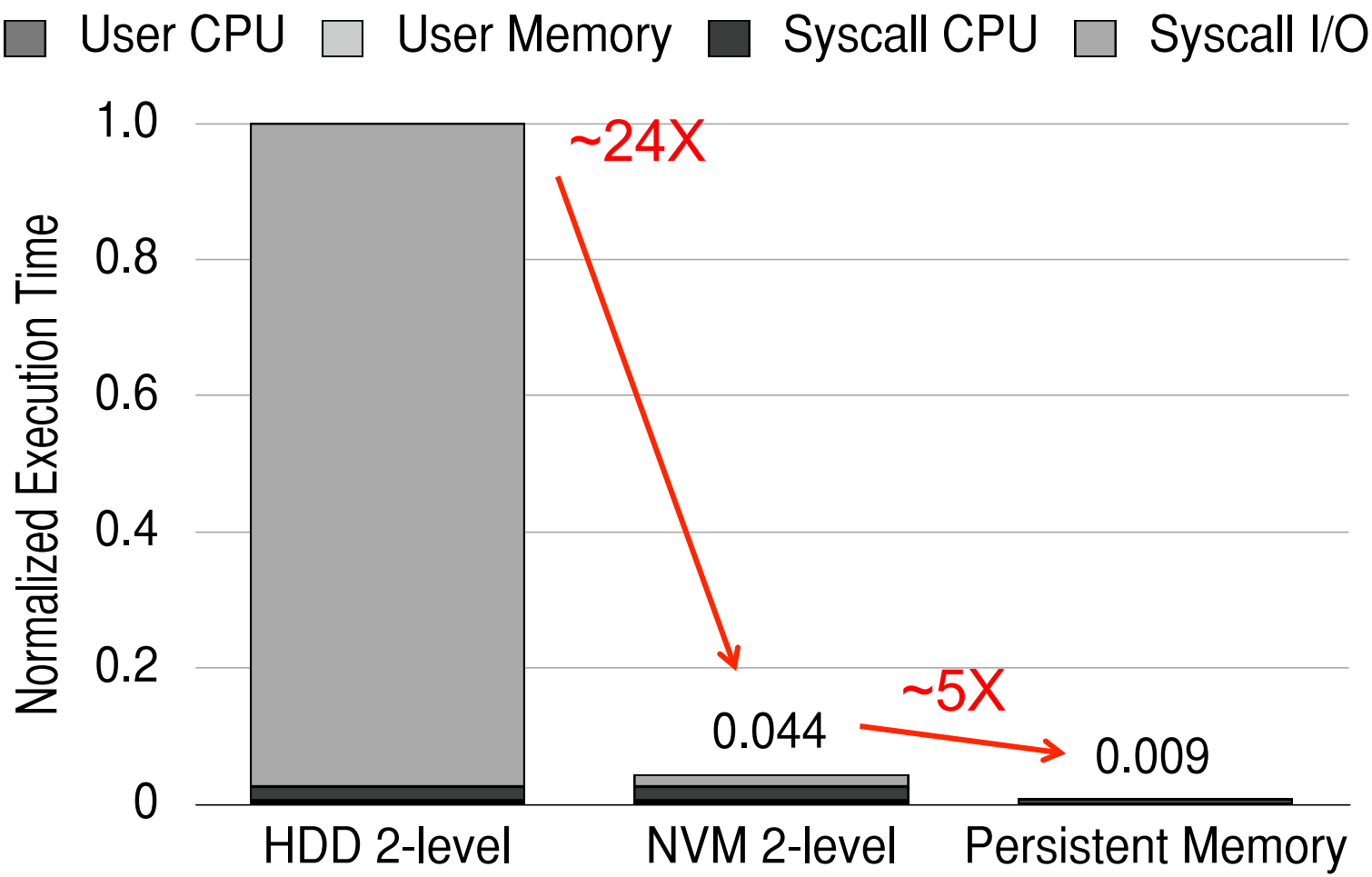


PMM uses access and hint information to allocate, locate, migrate and access data in the heterogeneous array of devices

The Persistent Memory Manager (PMM)

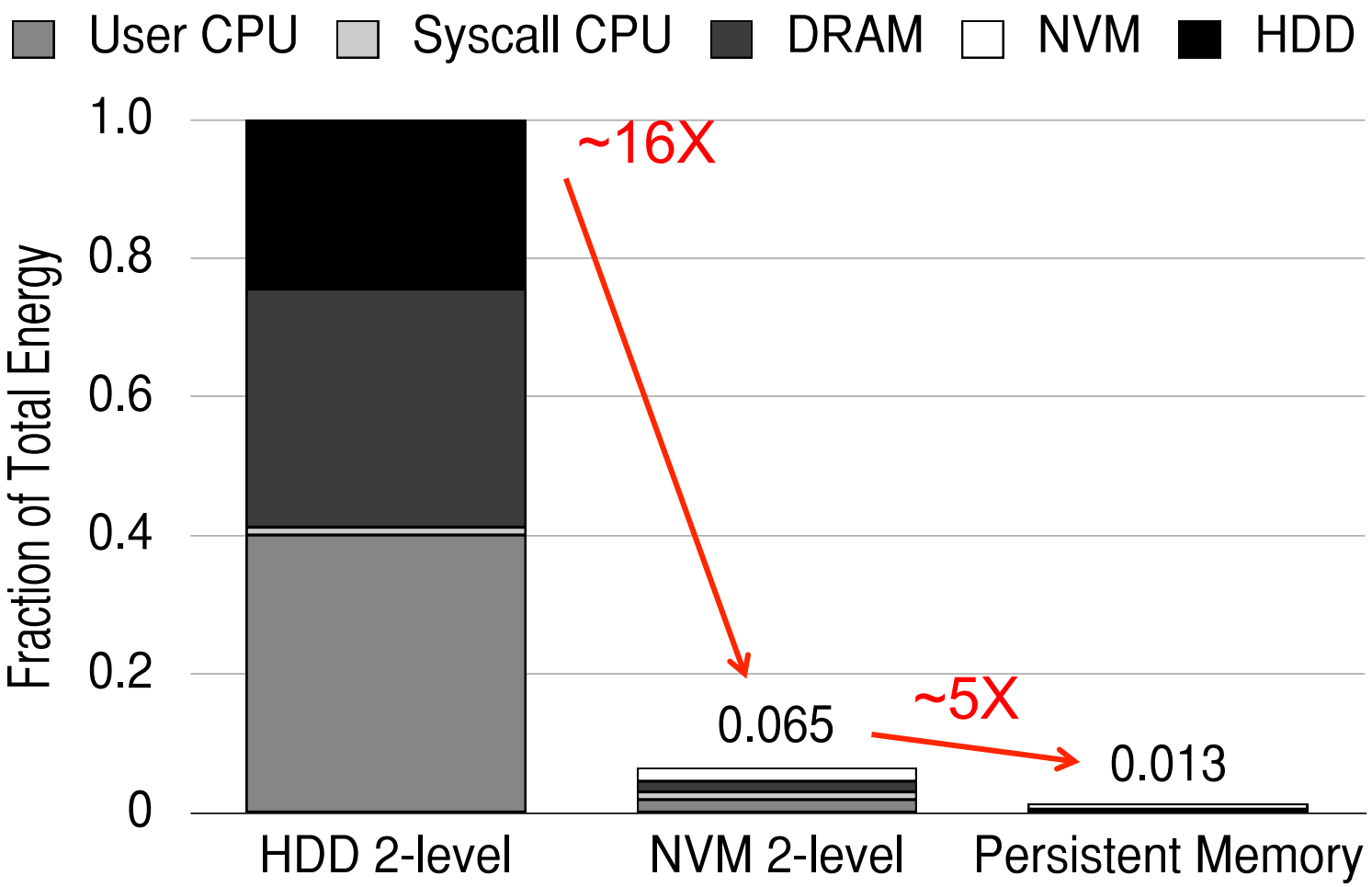
- Exposes a load/store interface to access persistent data
  - Applications can directly access persistent memory → no conversion, translation, location overhead for persistent data
- Manages data placement, location, persistence, security
  - To get the best of multiple forms of storage
- Manages metadata 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.

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.

Challenge and Opportunity

Combined

Memory & Storage

Departing From "Business as Usual"

A Unified Interface to **All Data**

Agenda

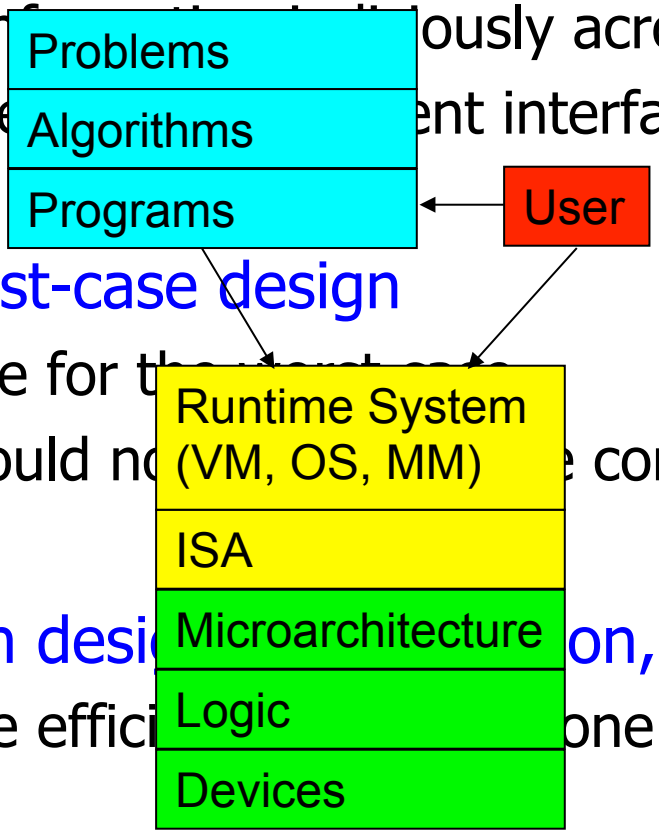
- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

Principles (So Far)

- Better interfaces between layers of the system stack
  - Expose more information judiciously across the system stack
  - Design more flexible and efficient interfaces
- Better-than-worst-case design
  - Do not optimize for the worst case
  - Worst case should not determine the common case
- Heterogeneity in design (specialization, asymmetry)
  - Enables a more efficient design (No one size fits all)
- These principles are coupled (and require broad thinking)

Principles (So Far)

- Better interfaces between layers of the system stack
  - Expose more in ously across the system stack
  - Design more fle ent interfaces
- Better-than-worst-case design
  - Do not optimize for t
  - Worst case should nc common case
- Heterogeneity in desi on, asymmetry)
  - Enables a more effici one size fits all)
- These principles are coupled (and require broad thinking)



Agenda

- Major Trends Affecting Main Memory
- The Memory Scaling Problem and Solution Directions
  - New Memory Architectures
  - Enabling Emerging Technologies
- Cross-Cutting Principles
- Summary

Summary

Business as Usual	Opportunity
RowHammer	Memory controller anticipates and fixes errors
Fixed, frequent refreshes	Heterogeneous refresh rate across memory
Fixed, high latency	Heterogeneous latency in time and space
Slow page copy & initialization	Exploit internal connectivity in memory to move data
Fixed reliability mechanisms	Heterogeneous reliability across time and space
Memory as a dumb device	Memory as an accelerator and autonomous agent
DRAM-only main memory	Emerging memory technologies and hybrid memories
Two-level data storage model	Unified interface to all data
Large timing and error margins	Online adaptation of timing and error margins
Poor performance guarantees	Strong service guarantees and configurable QoS
Fixed policies in controllers	Configurable and programmable memory controllers
…	…

Summary

- Memory problems are a critical bottleneck for system performance, efficiency, and usability
- New memory architectures
  - Compute capable and autonomous memory
- Enabling emerging NVM technologies
  - Persistent and hybrid memory
- System-level memory/storage QoS
  - Predictable systems with configurable QoS
- **Many opportunities and challenges that will change the systems and software we design**

Some Open Source Tools

- Rowhammer
  - https://github.com/CMU-SAFARI/rowhammer
- Ramulator – Fast and Extensible DRAM Simulator
  - https://github.com/CMU-SAFARI/ramulator
- MemSim
  - https://github.com/CMU-SAFARI/memsim
- NOCulator
  - https://github.com/CMU-SAFARI/NOCulator
- DRAM Error Model
  - http://www.ece.cmu.edu/~safari/tools/memerr/index.html
- Other open-source software from my group
  - https://github.com/CMU-SAFARI/
  - http://www.ece.cmu.edu/~safari/tools.html

Referenced Papers

- All are available at
  http://users.ece.cmu.edu/~omutlu/projects.htm
  http://scholar.google.com/citations?user=7XyGUGkAAAAJ&hl=en

- A detailed accompanying overview paper

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

Related Videos and Course Materials

- Undergraduate Computer Architecture Course Lecture Videos (2013, 2014, 2015)
- Undergraduate Computer Architecture Course Materials (2013, 2014, 2015)
- Graduate Computer Architecture Lecture Videos (2013, 2015)
- Graduate Computer Architecture Course Materials (2013, 2015)
- Parallel Computer Architecture Course Materials (Lecture Videos)
- Memory Systems Short Course Materials (Lecture Video on Main Memory and DRAM Basics)

Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters'15]

Ramulator Motivation

- DRAM and Memory Controller landscape is changing
- Many new and upcoming standards
- Many new controller designs
- A fast and easy-to-extend simulator is very much needed

Segment	*DRAM Standards & Architectures*
Commodity	DDR3 (2007) [14]; DDR4 (2012) [18]
Low-Power	LPDDR3 (2012) [17]; LPDDR4 (2014) [20]
Graphics	GDDR5 (2009) [15]
Performance	eDRAM [28], [32]; RLDRAM3 (2011) [29]
3D-Stacked	WIO (2011) [16]; WIO2 (2014) [21]; MCDRAM (2015) [13]; HBM (2013) [19]; HMC1.0 (2013) [10]; HMC1.1 (2014) [11]
Academic	SBA/SSA (2010) [38]; Staged Reads (2012) [8]; RAIDR (2012) [27]; SALP (2012) [24]; TL-DRAM (2013) [26]; RowClone (2013) [37]; Half-DRAM (2014) [39]; Row-Buffer Decoupling (2014) [33]; SARP (2014) [6]; AL-DRAM (2015) [25]

Table 1. Landscape of DRAM-based memory

Ramulator

- Provides out-of-the box support for many DRAM standards:
  - DDR3/4, LPDDR3/4, GDDR5, WIO1/2, HBM, plus new proposals (SALP, AL-DRAM, TLDRAM, RowClone, and SARP)
- ~2.5X faster than fastest open-source simulator
- Modular and extensible to different standards

Simulator (clang -O3)	*Cycles (10^6)*		*Runtime (sec.)*		*Req/sec (10^3)*		*Memory (MB)*
	Random	*Stream*	*Random*	*Stream*	*Random*	*Stream*	
Ramulator	652	411	752	249	133	402	2.1
DRAMSim2	645	413	2,030	876	49	114	1.2
USIMM	661	409	1,880	750	53	133	4.5
DrSim	647	406	18,109	12,984	6	8	1.6
NVMain	666	413	6,881	5,023	15	20	4,230.0

Table 3. Comparison of five simulators using two traces

Case Study: Comparison of DRAM Standards

Standard	*Rate (MT/s)*	*Timing (CL-RCD-RP)*	*Data-Bus (Width×Chan.)*	*Rank-per-Chan*	*BW (GB/s)*
DDR3	1,600	11-11-11	64-bit × 1	1	11.9
DDR4	2,400	16-16-16	64-bit × 1	1	17.9
SALP†	1,600	11-11-11	64-bit × 1	1	11.9
LPDDR3	1,600	12-15-15	64-bit × 1	1	11.9
LPDDR4	2,400	22-22-22	32-bit × 2*	1	17.9
GDDR5 [12]	6,000	18-18-18	64-bit × 1	1	44.7
HBM	1,000	7-7-7	128-bit × 8*	1	119.2
WIO	266	7-7-7	128-bit × 4*	1	15.9
WIO2	1,066	9-10-10	128-bit × 8*	1	127.2

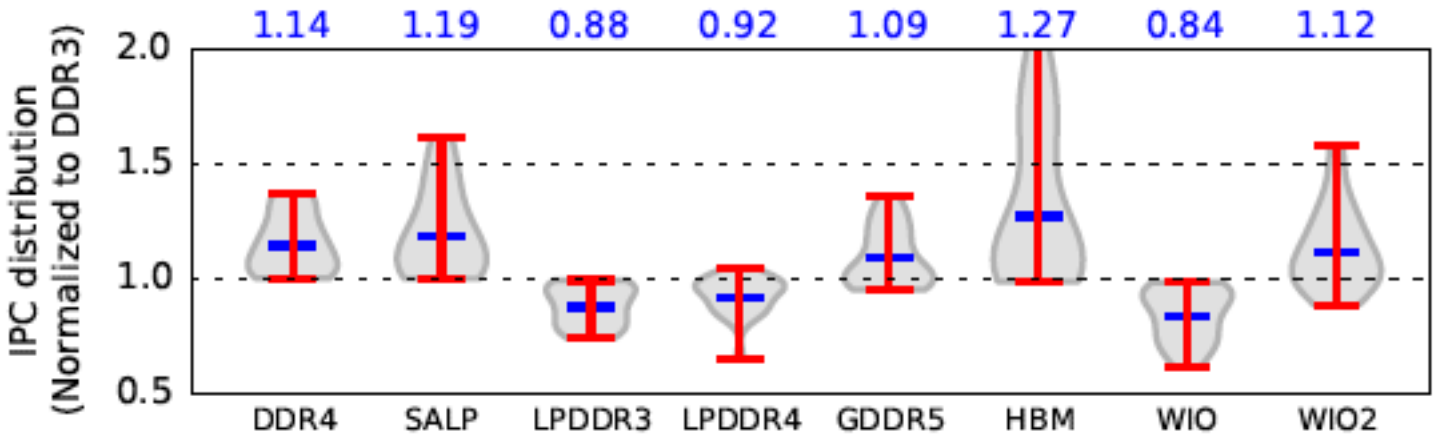


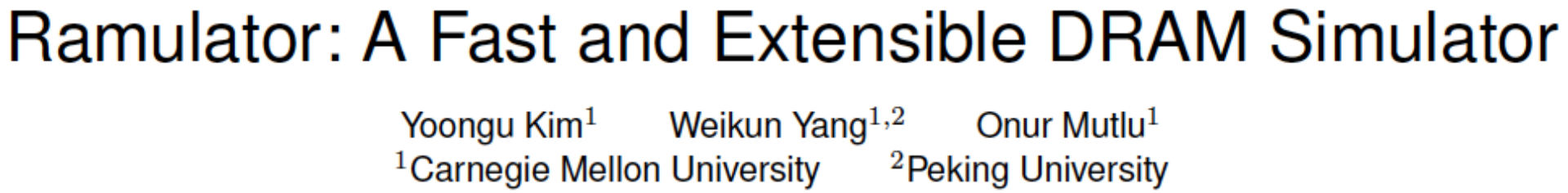
Figure 2. Performance comparison of DRAM standards

Across 22 workloads, simple CPU model

Ramulator Paper and Source Code

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

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



DRAM Infrastructure

Experimental DRAM Testing Infrastructure

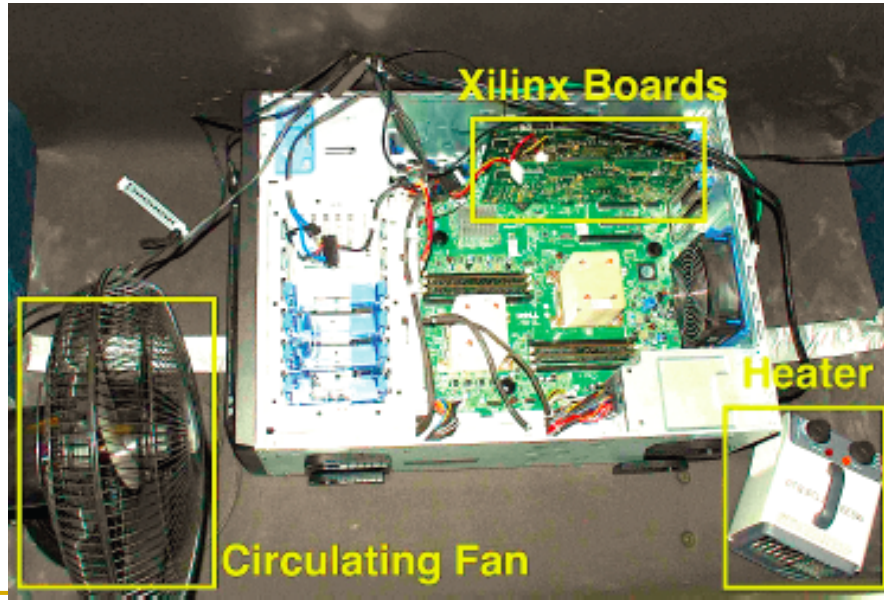
An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms (Liu et al., ISCA 2013)

The Efficacy of Error Mitigation Techniques for DRAM Retention Failures: A Comparative Experimental Study (Khan et al., SIGMETRICS 2014)

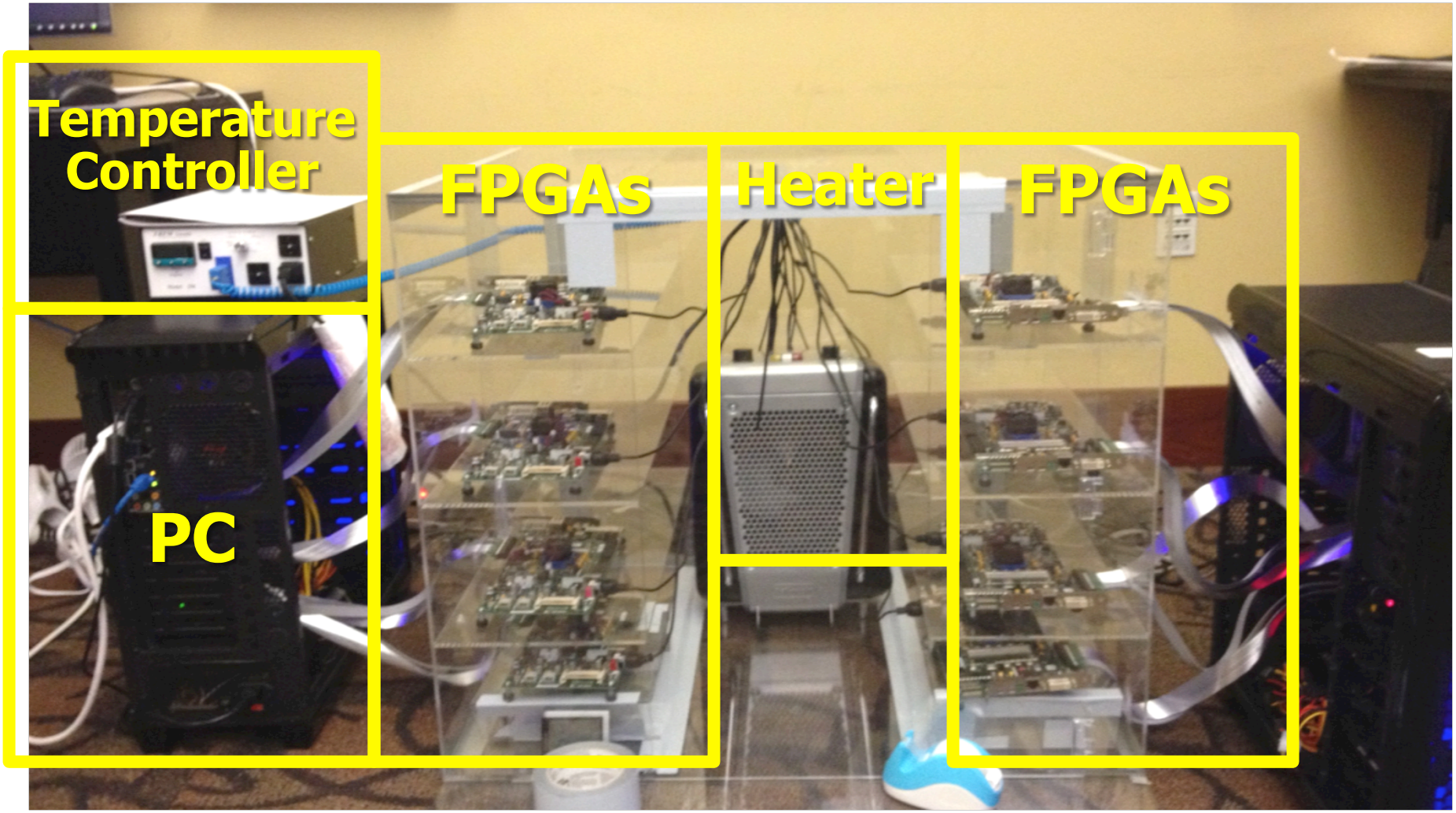
Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors (Kim et al., ISCA 2014)

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case (Lee et al., HPCA 2015)

AVATAR: A Variable-Retention-Time (VRT) Aware Refresh for DRAM Systems (Qureshi et al., DSN 2015)



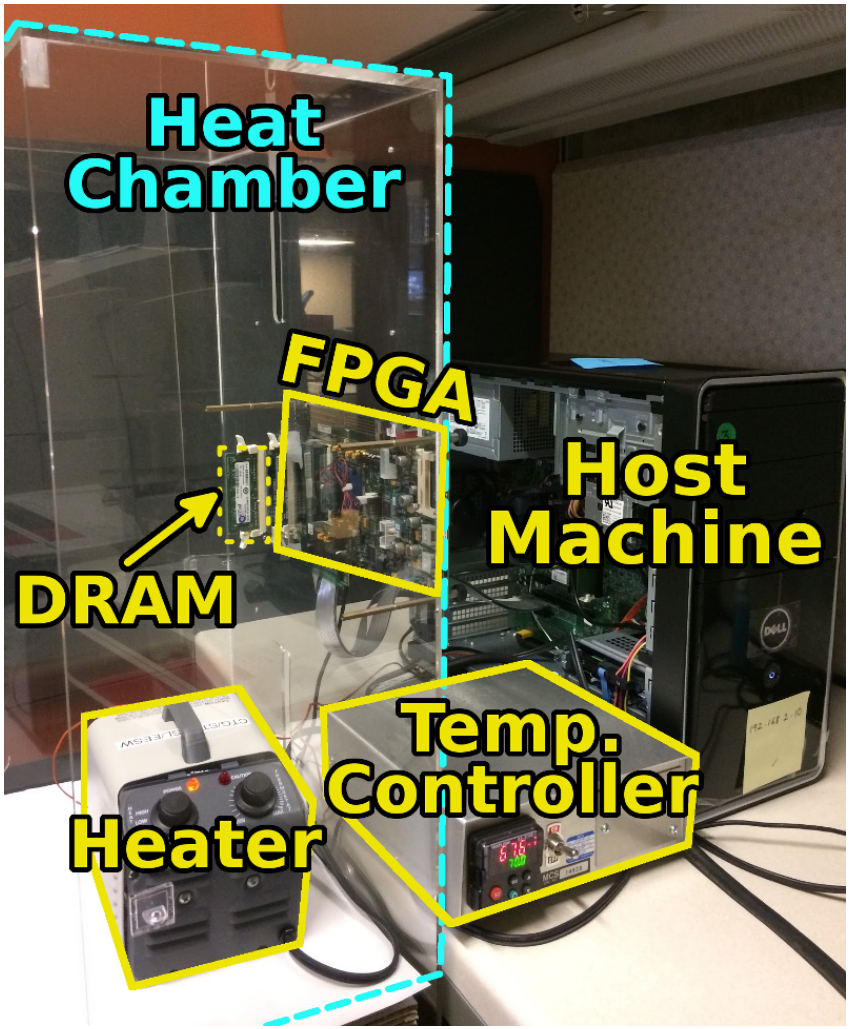
Experimental DRAM Testing Infrastructure



Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

SoftMC: Open Source DRAM Infrastructure

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



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

  github.com/CMU-SAFARI/SoftMC

SoftMC: Open Source DRAM Infrastructure

- https://github.com/CMU-SAFARI/SoftMC

SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies

Hasan Hassan[1,2,3] Nandita Vijaykumar[3] Samira Khan[4,3] Saugata Ghose[3] Kevin Chang[3]
Gennady Pekhimenko[5,3] Donghyuk Lee[6,3] Oguz Ergin[2] Onur Mutlu[1,3]

[1]*ETH Zürich* [2]*TOBB University of Economics & Technology* [3]*Carnegie Mellon University*
[4]*University of Virginia* [5]*Microsoft Research* [6]*NVIDIA Research*