Computer Architecture

Lecture 6: Low-Latency DRAM and Processing In Memory

Prof. Onur Mutlu
ETH Zürich

Fall 2017

5 October 2017

High-Level Summary of Last Lecture

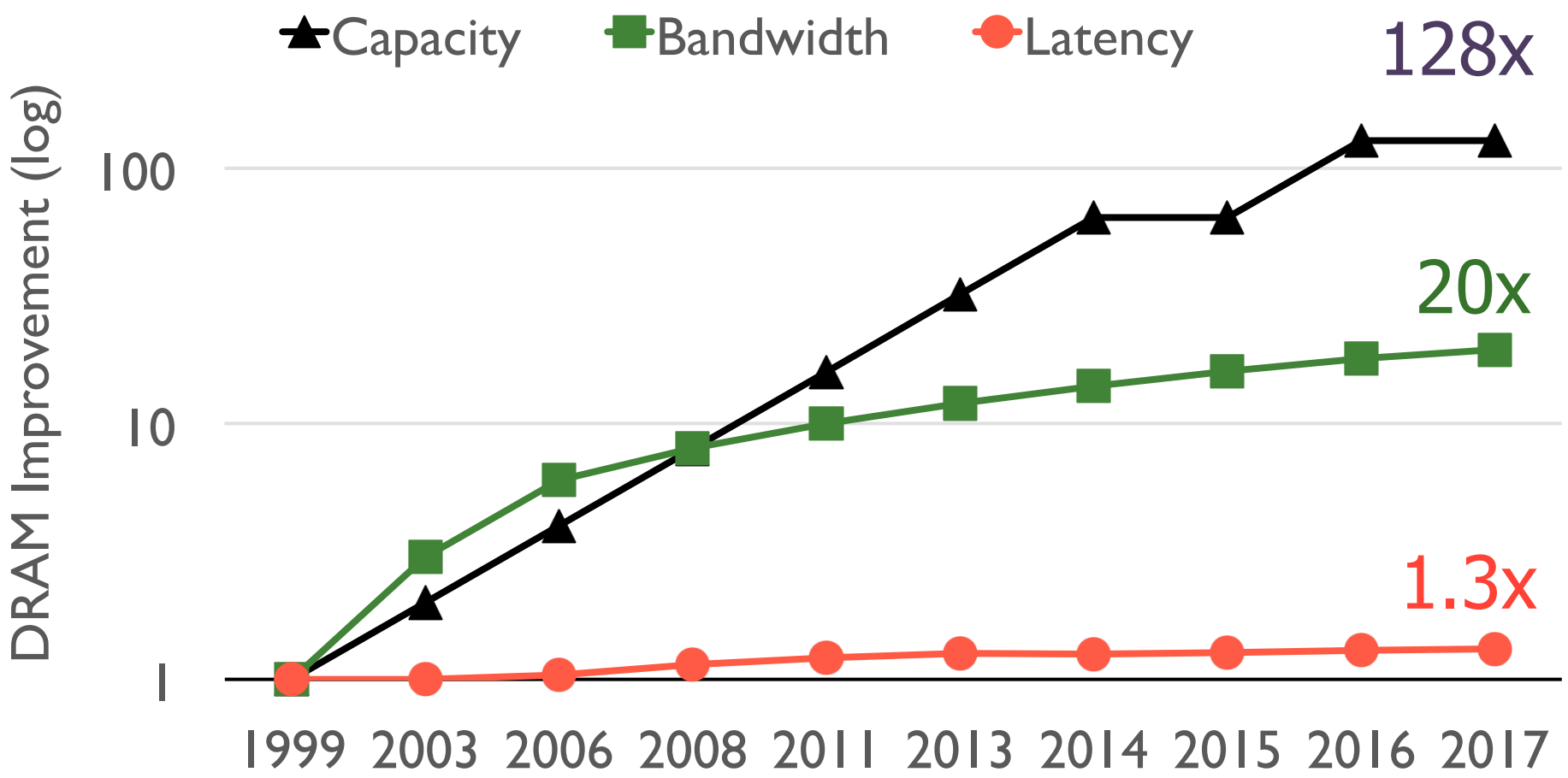
- DRAM Operation Continued
- Memory Controllers
- Memory Latency
  - Tiered-Latency DRAM

Agenda for Today

- Memory Latency Continued
- Memory Latency-Voltage-Reliability Relationship
- Processing In Memory

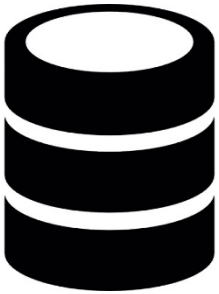
Memory Latency: Fundamental Tradeoffs

Review: Memory Latency Lags Behind

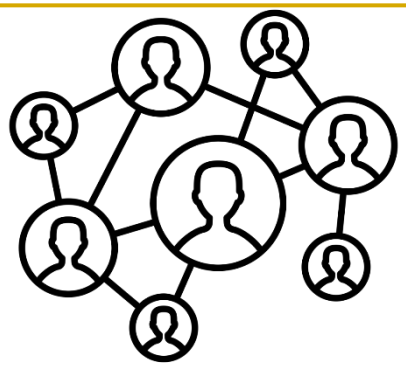


Memory latency remains almost constant

DRAM Latency Is Critical for Performance

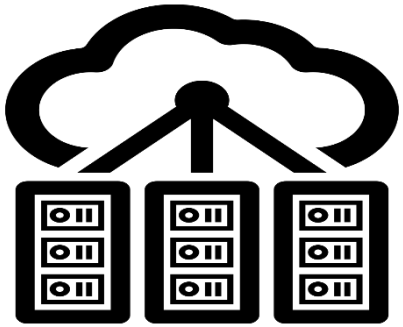


In-memory Databases
[Mao+, EuroSys'12;
Clapp+ (**Intel**), IISWC'15]



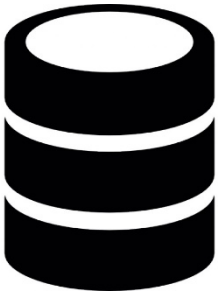
Graph/Tree Processing
[Xu+, IISWC'12; Umuroglu+, FPL'15]

In-Memory Data Analytics
[Clapp+ (**Intel**), IISWC'15;
Awan+, BDCloud'15]

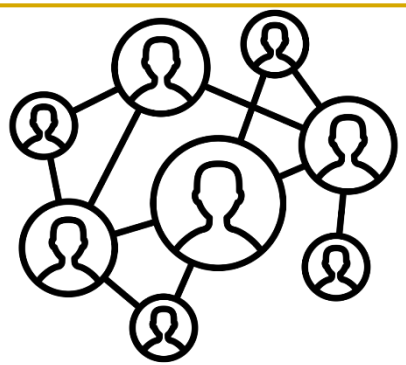


Datacenter Workloads
[Kanev+ (**Google**), ISCA'15]

DRAM Latency Is Critical for Performance



In-memory Databases

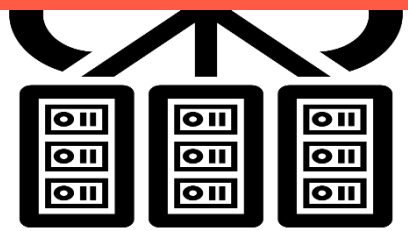


Graph/Tree Processing

Long memory latency → performance bottleneck

In-Memory Data Analytics

[Clapp+ (**Intel**), IISWC'15;
Awan+, BDCloud'15]



Datacenter Workloads

[Kanev+ (**Google**), ISCA'15]

What Causes the Long DRAM Latency?

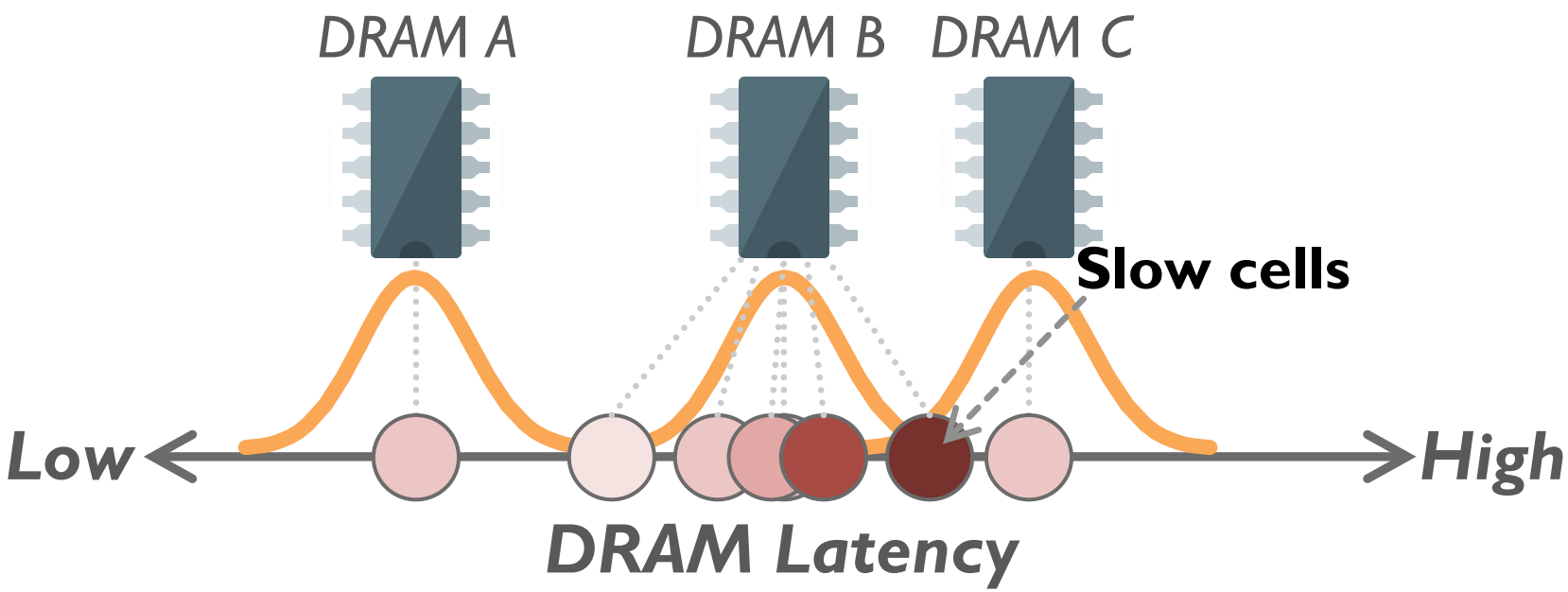
Why the Long Latency?

- Reason 1: Design of DRAM Micro-architecture
  - Goal: Maximize capacity/area, not minimize latency

- Reason 2: "One size fits all" approach to latency specification
  - Same latency parameters for all temperatures
  - Same latency parameters for all DRAM chips (e.g., rows)
  - Same latency parameters for all parts of a DRAM chip
  - Same latency parameters for all supply voltage levels
  - Same latency parameters for all application data
  - …

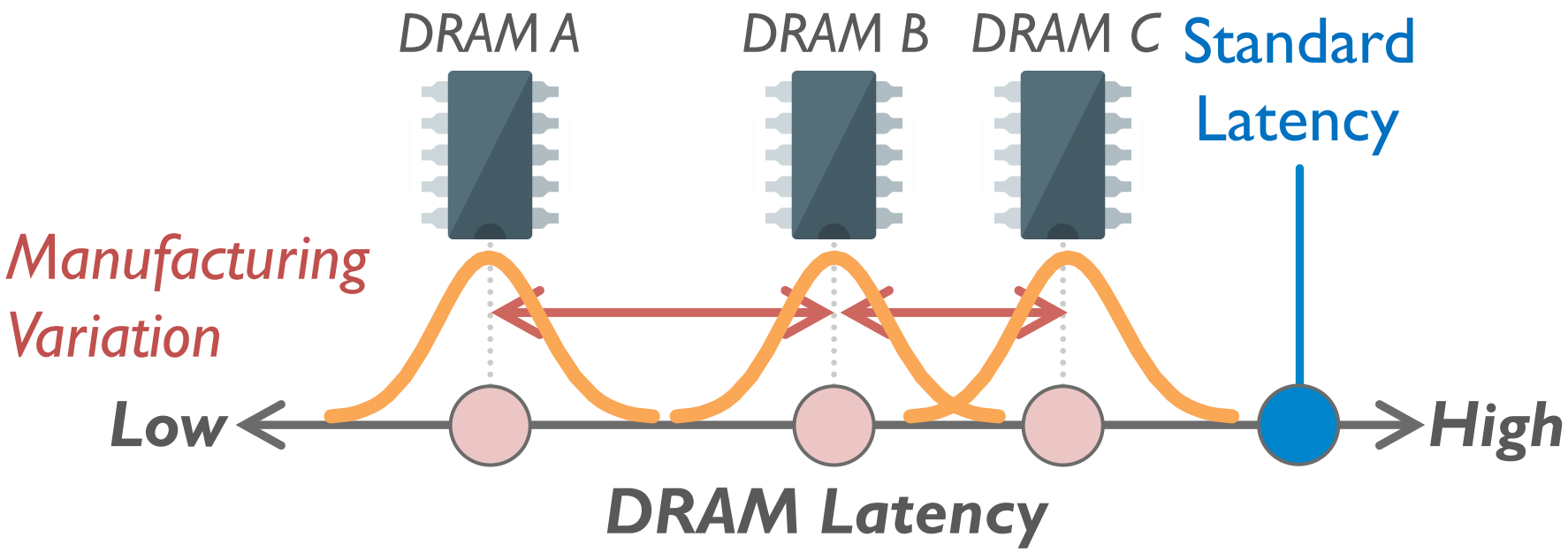
Latency Variation in Memory Chips

Heterogeneous manufacturing & operating conditions →
latency variation in timing parameters



Why is Latency High?

- DRAM latency: Delay as specified in DRAM standards
  - Doesn't reflect true DRAM device latency
- Imperfect manufacturing process → latency variation
- **High standard latency** chosen to increase yield

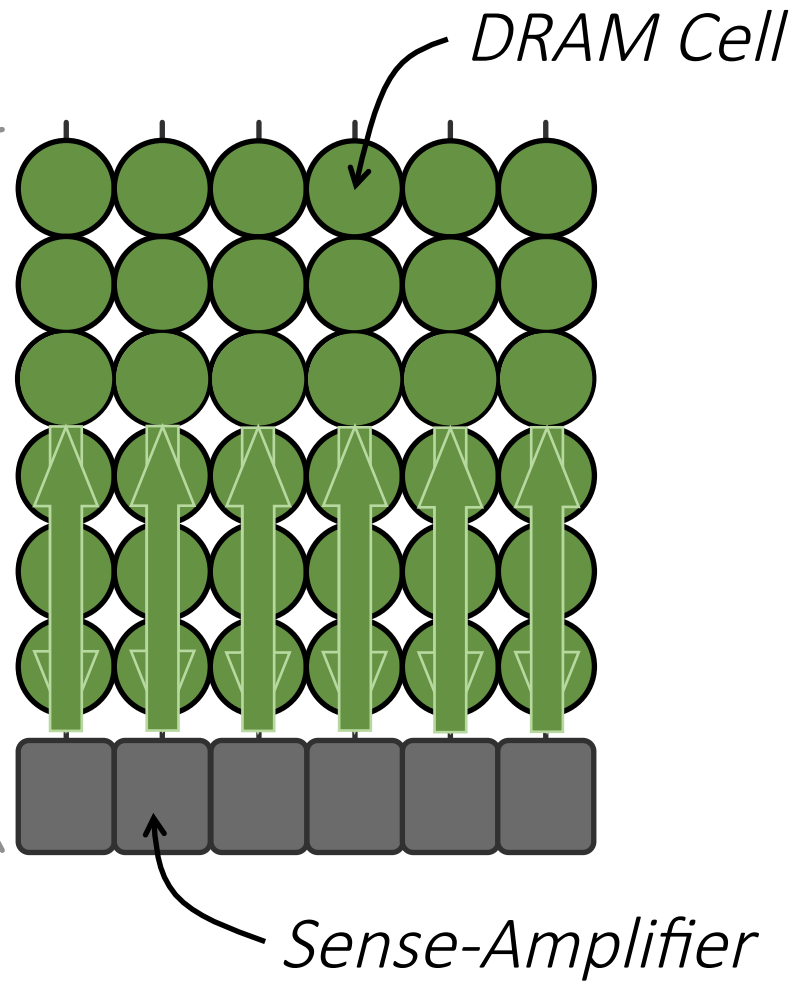


What Causes the Long Memory Latency?

- **Conservative timing margins!**
- DRAM timing parameters are set to cover the worst case
- Worst-case temperatures
  - 85 degrees vs. common-case
  - to enable a wide range of operating conditions
- Worst-case devices
  - DRAM cell with smallest charge across any acceptable device
  - to tolerate process variation at acceptable yield
- This leads to large timing margins for the common case

Understanding and Exploiting Variation in DRAM Latency

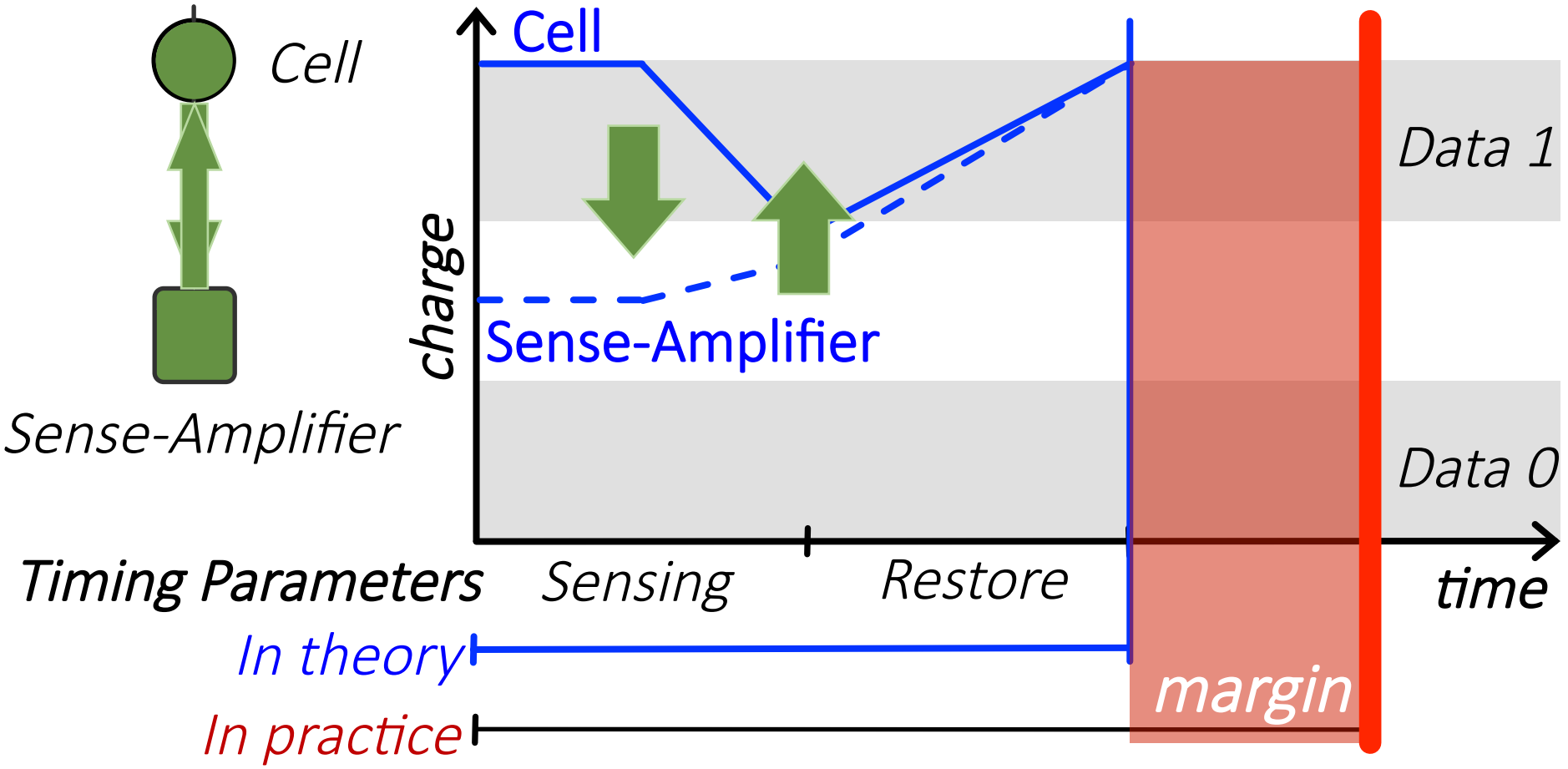
DRAM Stores Data as Charge



Three steps of charge movement

1. Sensing
2. Restore
3. Precharge

DRAM Charge over Time



Why does DRAM need the extra timing margin?

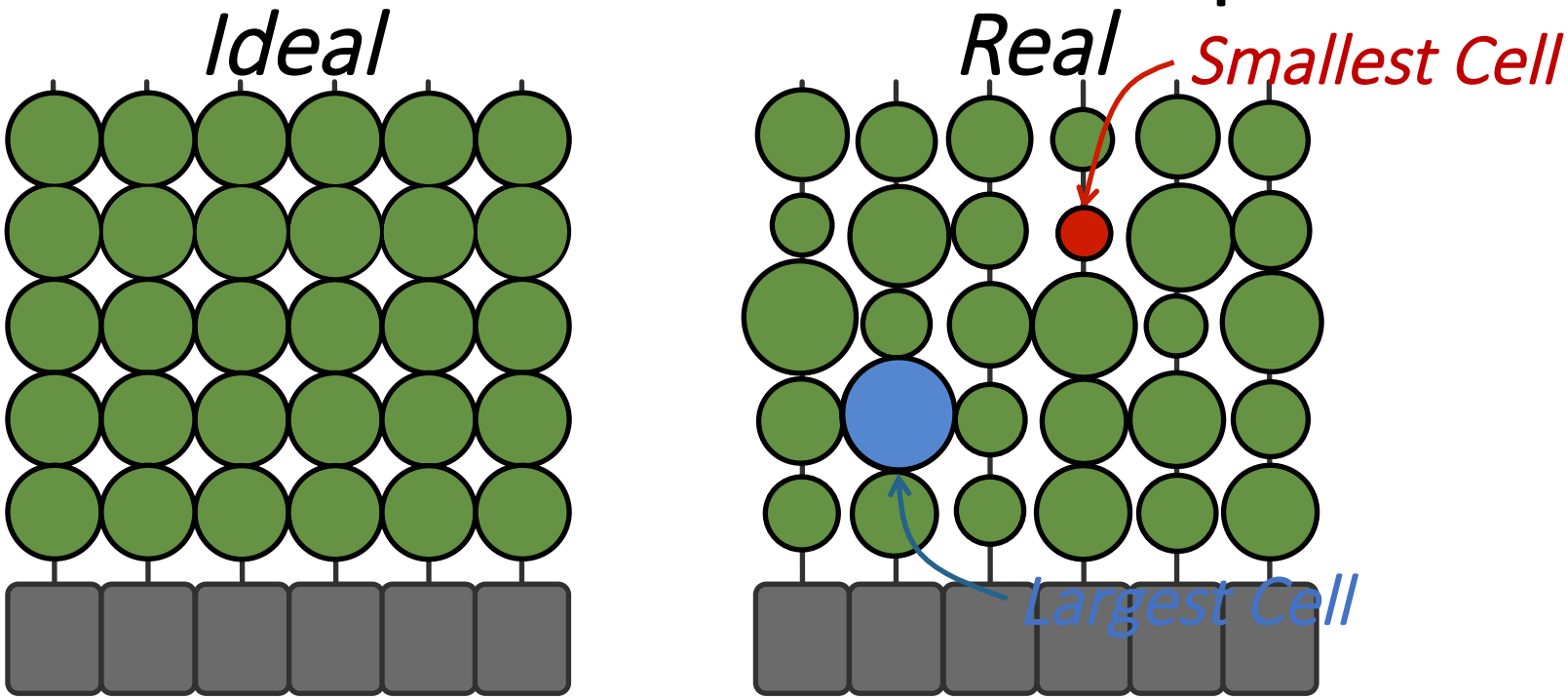
Two Reasons for Timing Margin

1. Process Variation

- DRAM cells are not equal
- Leads to extra timing margin for a cell that can store a large amount of charge

2. Temperature Dependence

DRAM Cells are Not Equal



Same Size ➔
Same Charge ➔
Same Latency

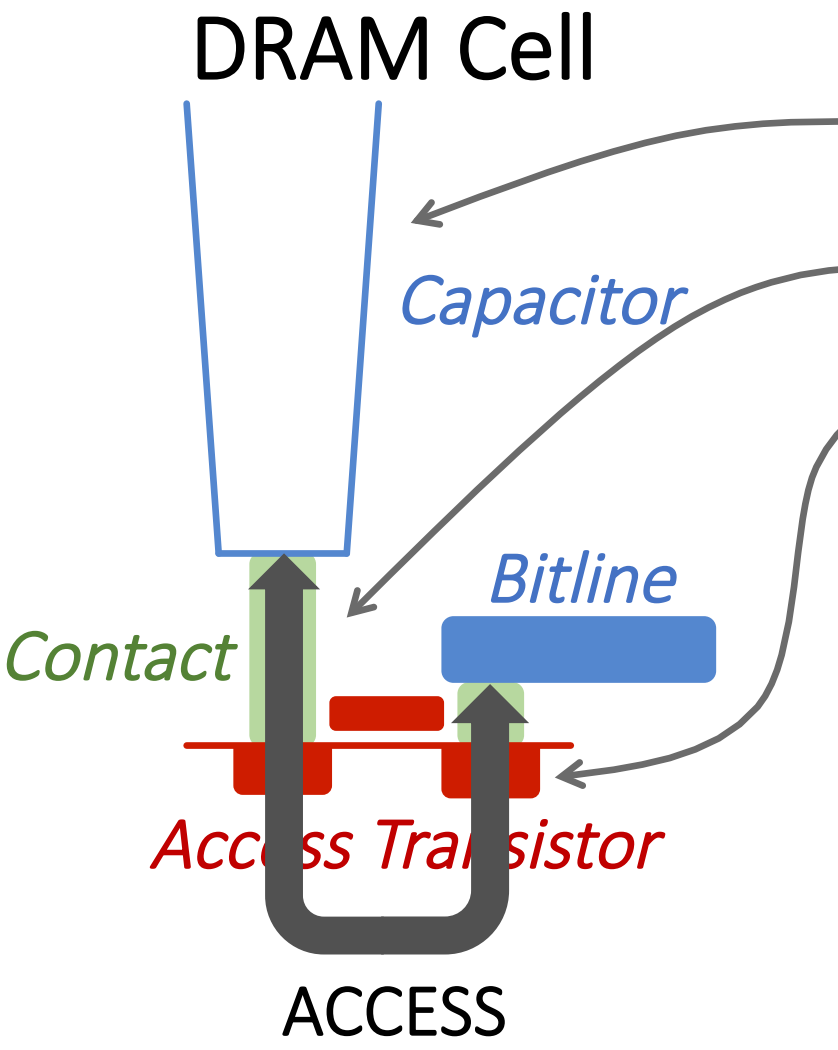
Large variation in cell size ➔

Large variation in charge ➔

Different Size ➔
Different Charge ➔
Different Latency

Large variation in access latency

Process Variation



❶ Cell Capacitance

❷ Contact Resistance

❸ Transistor Performance

Small cell can store small charge

- *Small cell capacitance*
- *High contact resistance*
- *Slow access transistor*

➔ *High access latency*

Two Reasons for Timing Margin

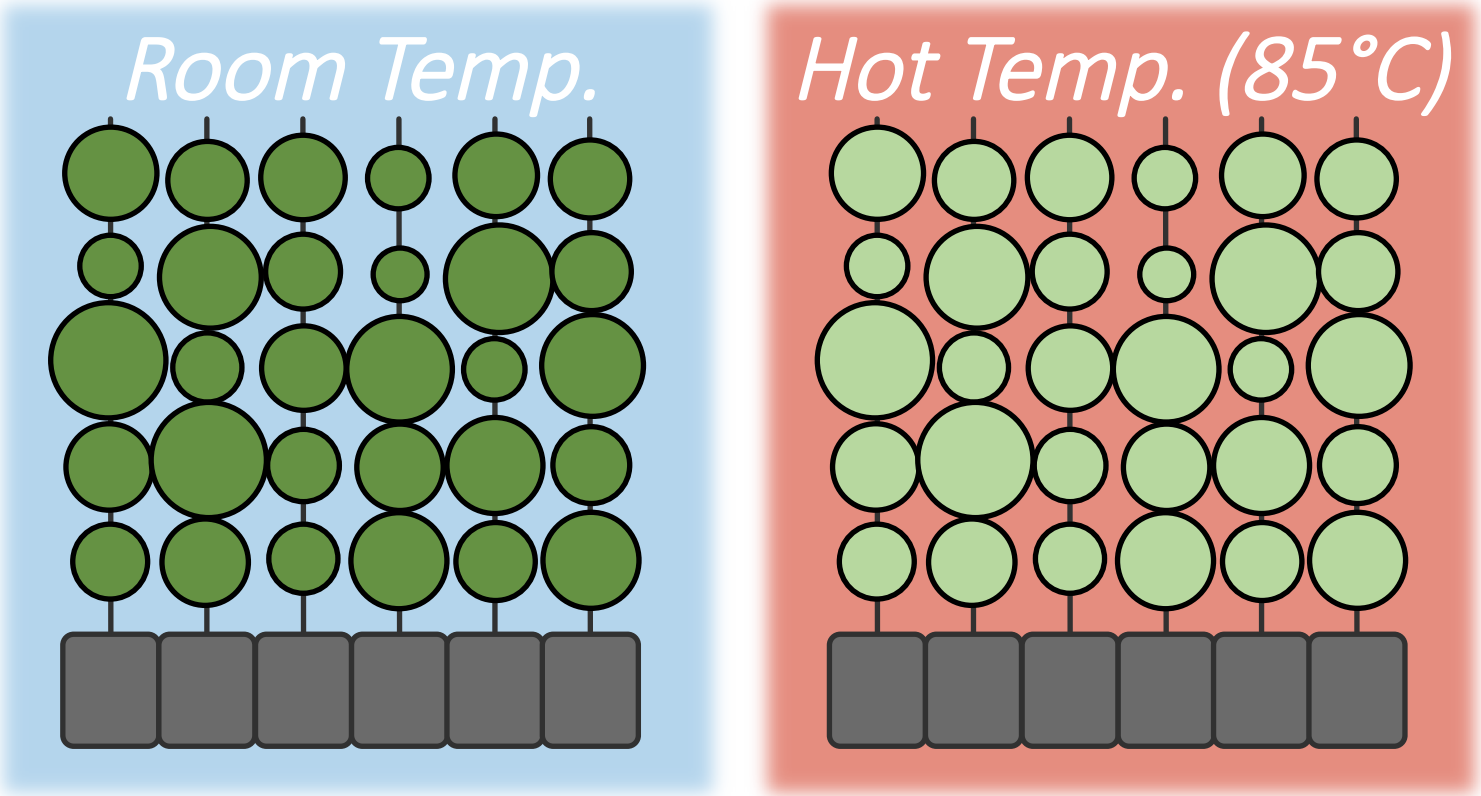
1. Process Variation

- DRAM cells are not equal
- Leads to extra timing margin for a cell that can store a large amount of charge

2. Temperature Dependence

- DRAM leaks more charge at higher temperature
- Leads to extra timing margin for cells that operate at low temperature

Charge Leakage Temperature



Cells store small charge at high temperature and large charge at low temperature

→ Large variation in access latency

DRAM Timing Parameters

- *DRAM timing parameters are dictated by* ***the worst-case***
  - The smallest cell with the smallest charge **in all DRAM products**
  - Operating at **the highest temperature**

- *Large timing margin for the common-case*

Adaptive-Latency DRAM [HPCA 2015]

- Idea: Optimize DRAM timing for the common case
  - Current temperature
  - Current DRAM module

- Why would this reduce latency?

  - A DRAM cell can store much more charge in the common case (low temperature, strong cell) than in the worst case

  - More charge in a DRAM cell
    → Faster sensing, charge restoration, precharging
    → Faster access (read, write, refresh, …)

Extra Charge → Reduced Latency

1. Sensing

   Sense **cells with extra charge** faster

   → Lower sensing latency

2. Restore

   No need to fully restore **cells with extra charge**

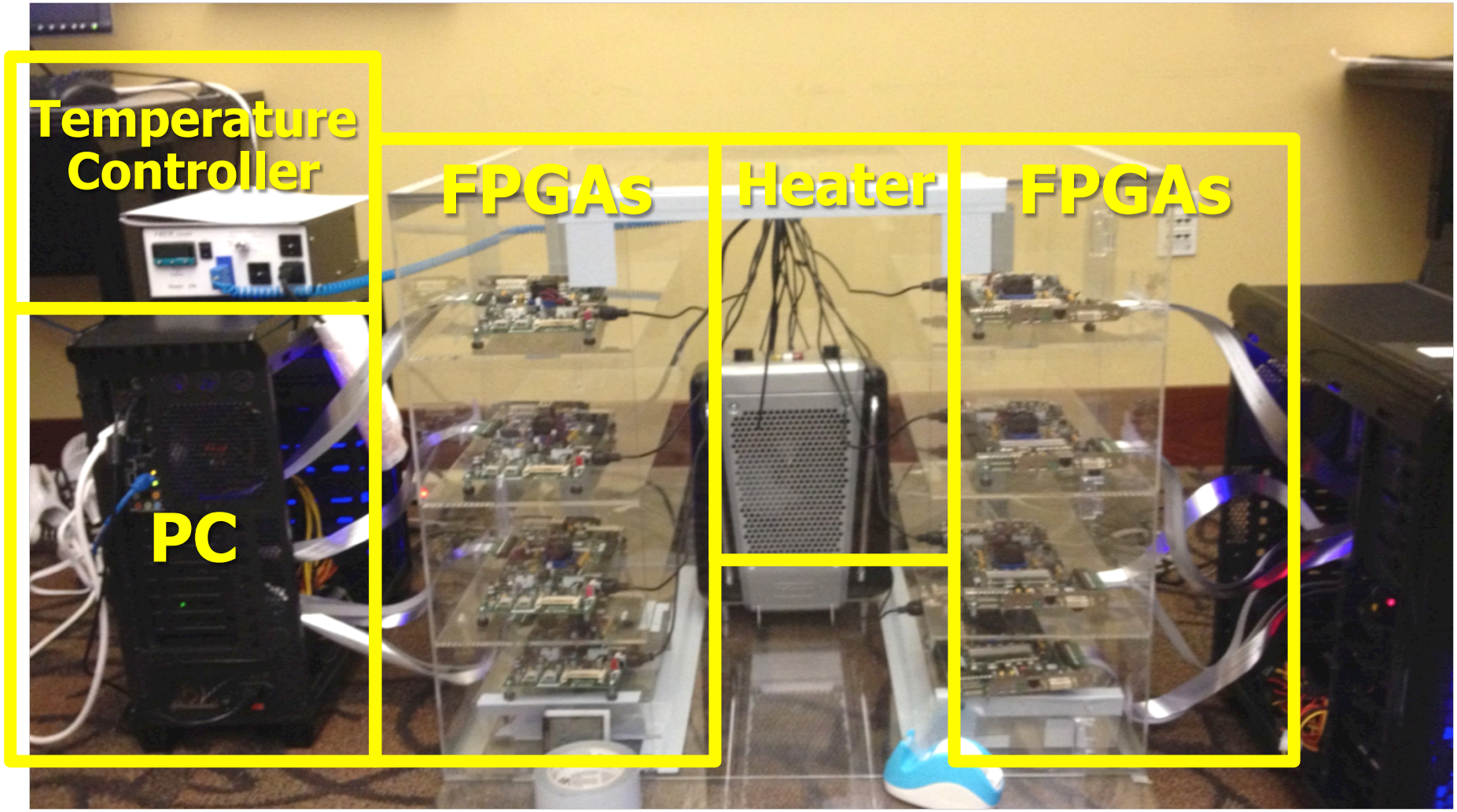
   → Lower restoration latency

3. Precharge

   No need to fully precharge bitlines for **cells with extra charge**

   → Lower precharge latency

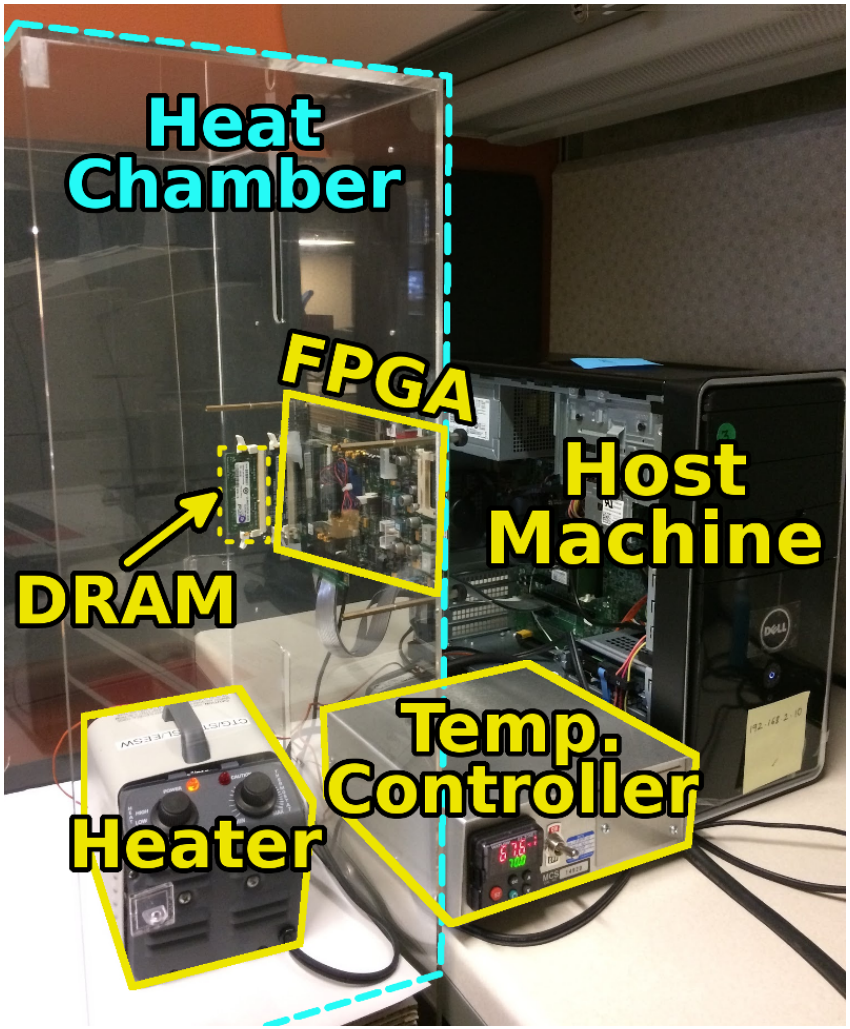
DRAM Characterization Infrastructure



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

DRAM Characterization 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

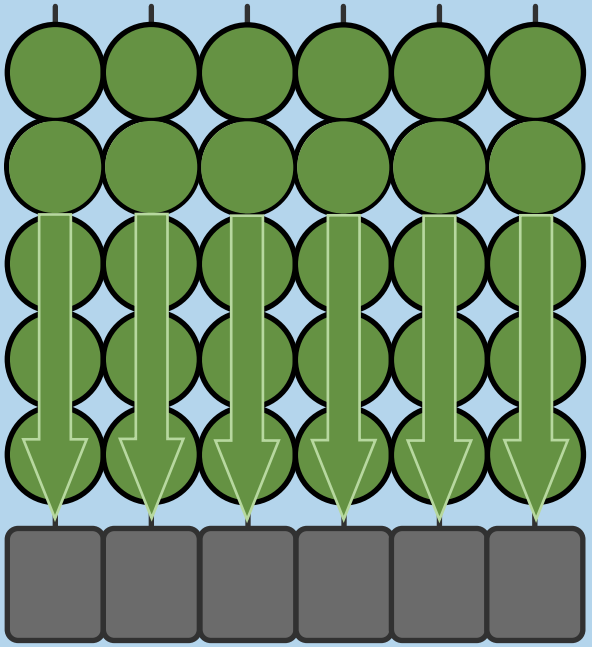
SoftMC: A Flexible and Practical Open-Source Infrastructure for Enabling Experimental DRAM Studies

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*

Observation 1. Faster Sensing

Typical DIMM at Low Temperature



More Charge

Strong Charge Flow

Faster Sensing

115 DIMM Characterization

Timing (tRCD)

17% ↓

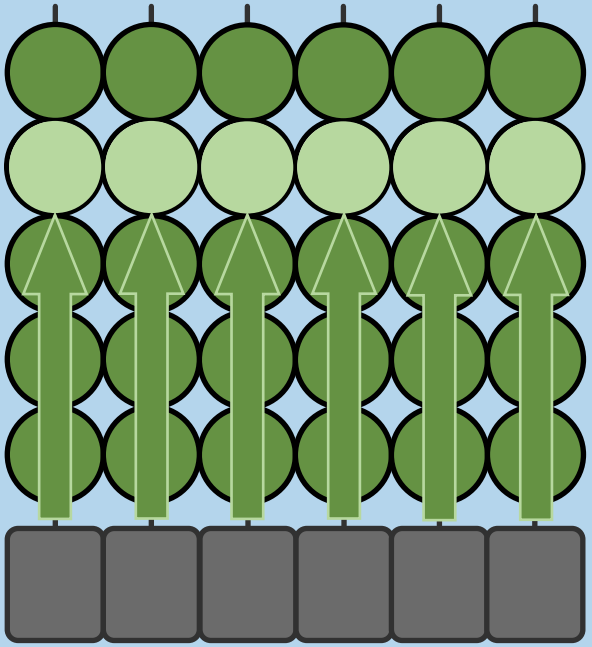
No Errors

Typical DIMM at Low Temperature

➔ *More charge* ➔ *Faster sensing*

Observation 2. Reducing Restore Time

Typical DIMM at Low Temperature



Less Leakage ➔ Extra Charge

No Need to Fully Restore Charge

115 DIMM Characterization

Read (tRAS)

37% ↓

Write (tWR)

54% ↓

No Errors

Typical DIMM at lower temperature

➔ *More charge* ➔ *Restore time reduction*

AL-DRAM

- *Key idea*
  - **Optimize DRAM timing parameters online**
- *Two components*
  - DRAM manufacturer provides multiple sets of reliable DRAM timing parameters at different temperatures for each DIMM
  - System monitors DRAM temperature & uses appropriate DRAM timing parameters

DRAM Temperature

- *DRAM temperature measurement*
  - Server cluster: Operates at under 34°C
  - Desktop: Operates at under 50°C
  - *DRAM standard optimized for 85°C*

DRAM operates at low temperatures in the common-case

- *Previous works – Maintain low DRAM temperature*
  - David+ ICAC 2011
  - Lin+ ISCA 2007
  - Zhu+ ITHERM 2008

Latency Reduction Summary of 115 DIMMs

- *Latency reduction for read & write (55°C)*
  - *Read Latency: **32.7%***
  - *Write Latency: **55.1%***
- *Latency reduction for each timing parameter (55°C)*
  - *Sensing: **17.3%***
  - *Restore: **37.3%** (read), **54.8%** (write)*
  - *Precharge: **35.2%***

Lee+, "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case," HPCA 2015.

AL-DRAM: Real System Evaluation

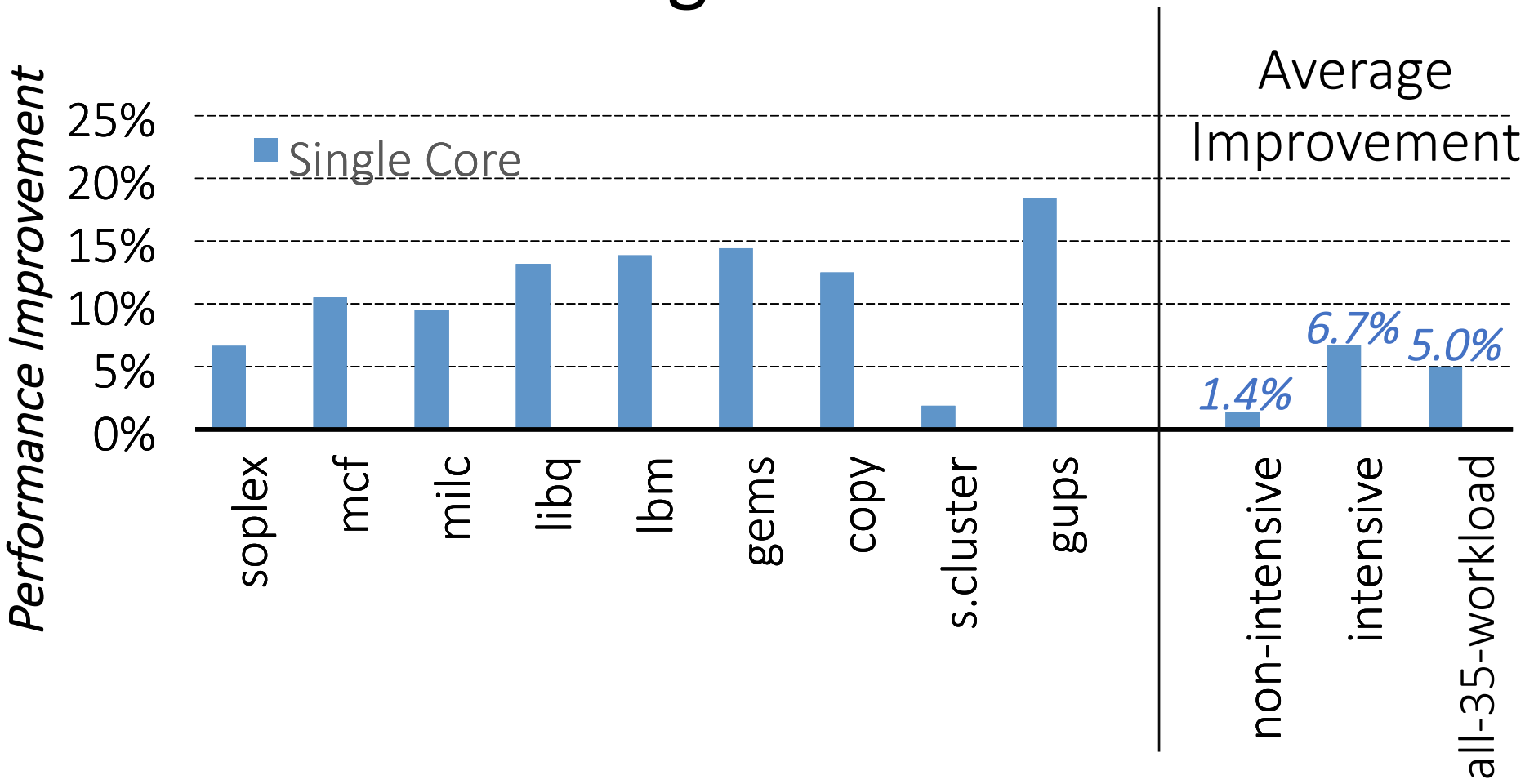
- *System*
  - *CPU: AMD 4386 (8 Cores, 3.1GHz, 8MB LLC)*

D18F2x200_dct[0]_mp[1:0] DDR3 DRAM Timing 0

Reset: 0F05_0505h. See 2.9.3 [DCT Configuration Registers].

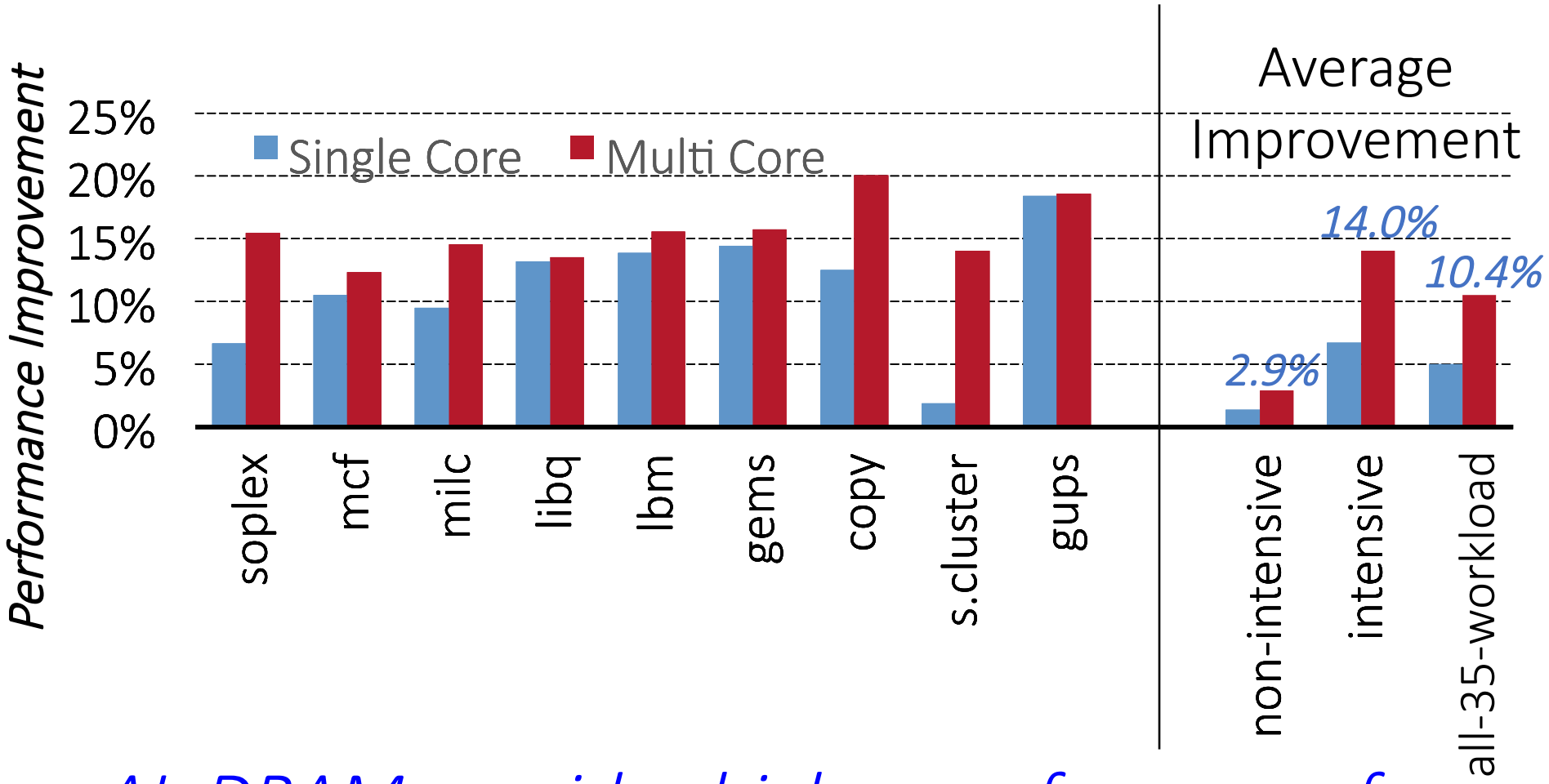
Bits	Description
31:30	Reserved.
29:24	**Tras: row active strobe**. Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration]. Specifies the minimum time in memory clock cycles from an activate command to a precharge command, both to the same chip select bank. Bits — Description 07h-00h — Reserved 2Ah-08h — <Tras> clocks 3Fh-2Bh — Reserved
23:21	Reserved.
20:16	**Trp: row precharge time**. Read-write. BIOS: See 2.9.7.5 [SPD ROM-Based Configuration]. Specifies the minimum time in memory clock cycles from a precharge command to an activate command or auto refresh command, both to the same bank.

AL-DRAM: Single-Core Evaluation



*AL-DRAM **improves performance** on a real system*

AL-DRAM: Multi-Core Evaluation



AL-DRAM provides higher performance for

multi-programmed & multi-threaded workloads

Reducing Latency Also Reduces Energy

- AL-DRAM reduces DRAM power consumption by 5.8%
- Major reason: reduction in row activation time

AL-DRAM: Advantages & Disadvantages

- Advantages
  - + Simple mechanism to reduce latency
  - + Significant system performance and energy benefits
    - + Benefits higher at low temperature
  - + Low cost, low complexity

- Disadvantages
  - - Need to determine reliable operating latencies for different temperatures and different DIMMs → higher testing cost
    - (might not be that difficult for low temperatures)

More on AL-DRAM

- Donghyuk Lee, Yoongu Kim, Gennady Pekhimenko, Samira Khan, Vivek Seshadri, Kevin Chang, and Onur Mutlu,
  "Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case"
  Proceedings of the
  21st International Symposium on High-Performance Computer Architecture (***HPCA***), Bay Area, CA, February 2015.
  [Slides (pptx) (pdf)] [Full data sets]

Adaptive-Latency DRAM: Optimizing DRAM Timing for the Common-Case

Donghyuk Lee Yoongu Kim Gennady Pekhimenko
Samira Khan Vivek Seshadri Kevin Chang Onur Mutlu

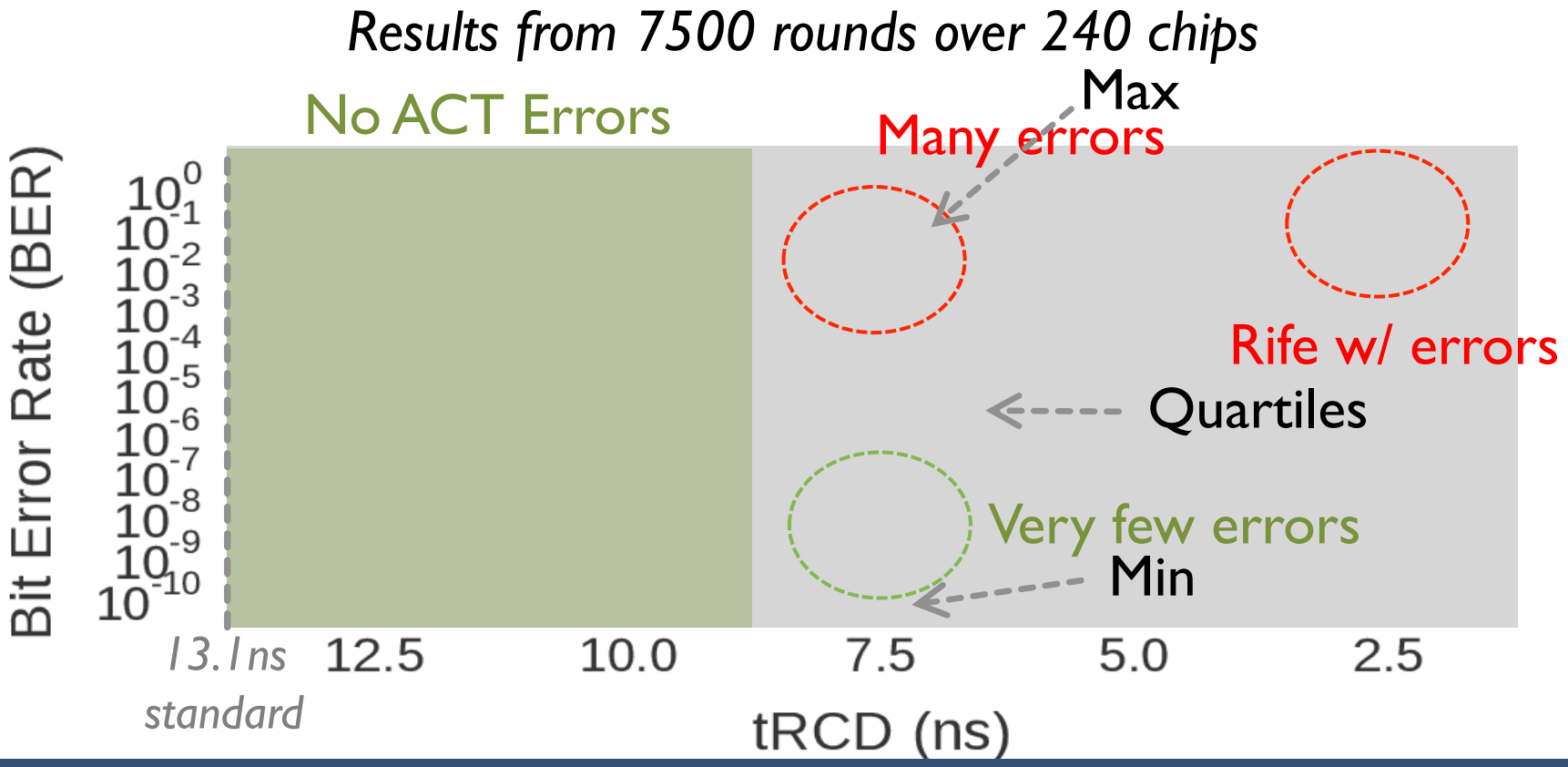
Carnegie Mellon University

Different Types of Latency Variation

- AL-DRAM exploits latency variation
  - Across time (different temperatures)
  - Across chips

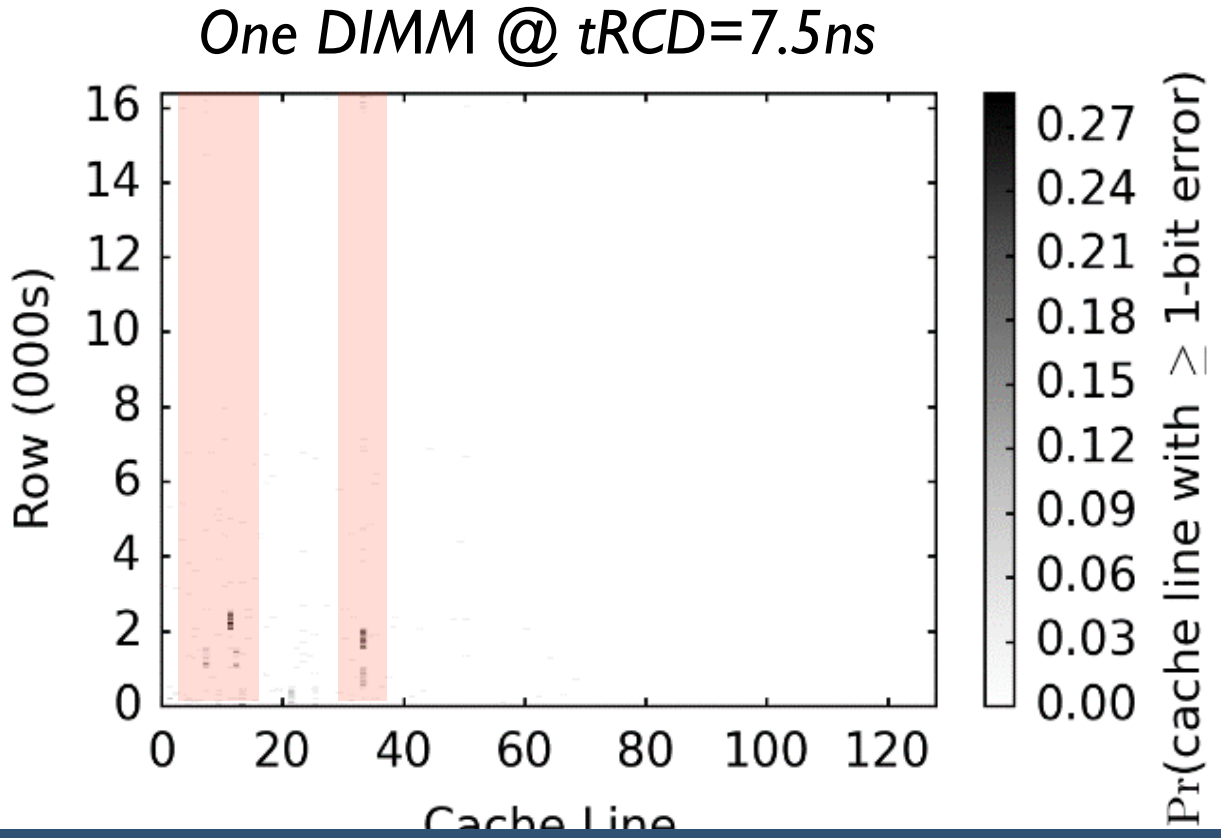
- Is there also latency variation within a chip?
  - Across different parts of a chip

Variation in Activation Errors



Modern DRAM chips exhibit significant variation in activation latency

Spatial Locality of Activation Errors



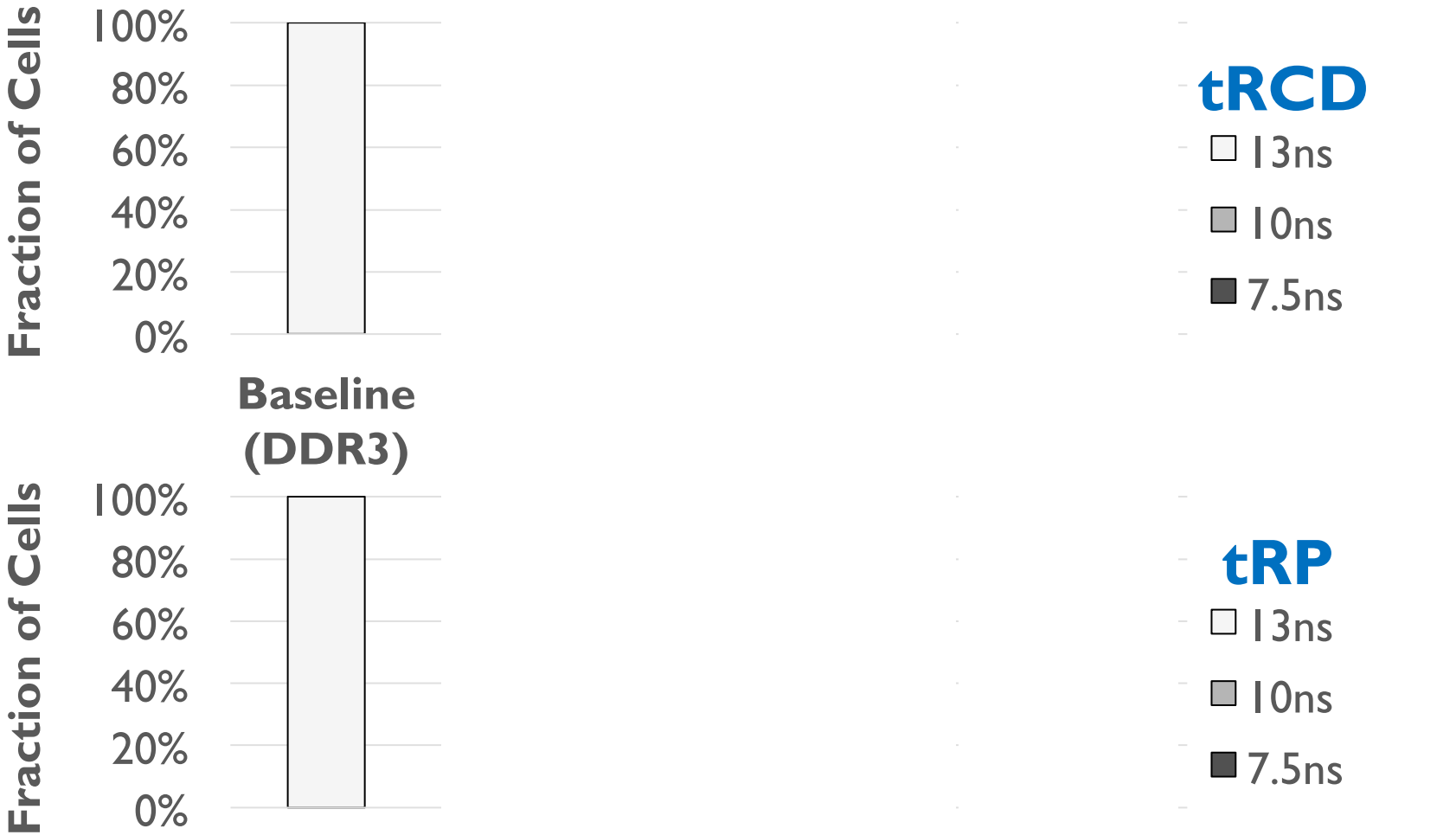
Activation errors are concentrated at certain columns of cells

Mechanism to Reduce DRAM Latency

- **Observation:** DRAM timing errors (slow DRAM cells) are concentrated on certain regions
- **Flexible-LatencY (FLY) DRAM**
  - A software-transparent design that reduces latency
- **Key idea**:
  1) Divide memory into regions of different latencies
  2) *Memory controller:* Use lower latency for regions without slow cells; higher latency for other regions

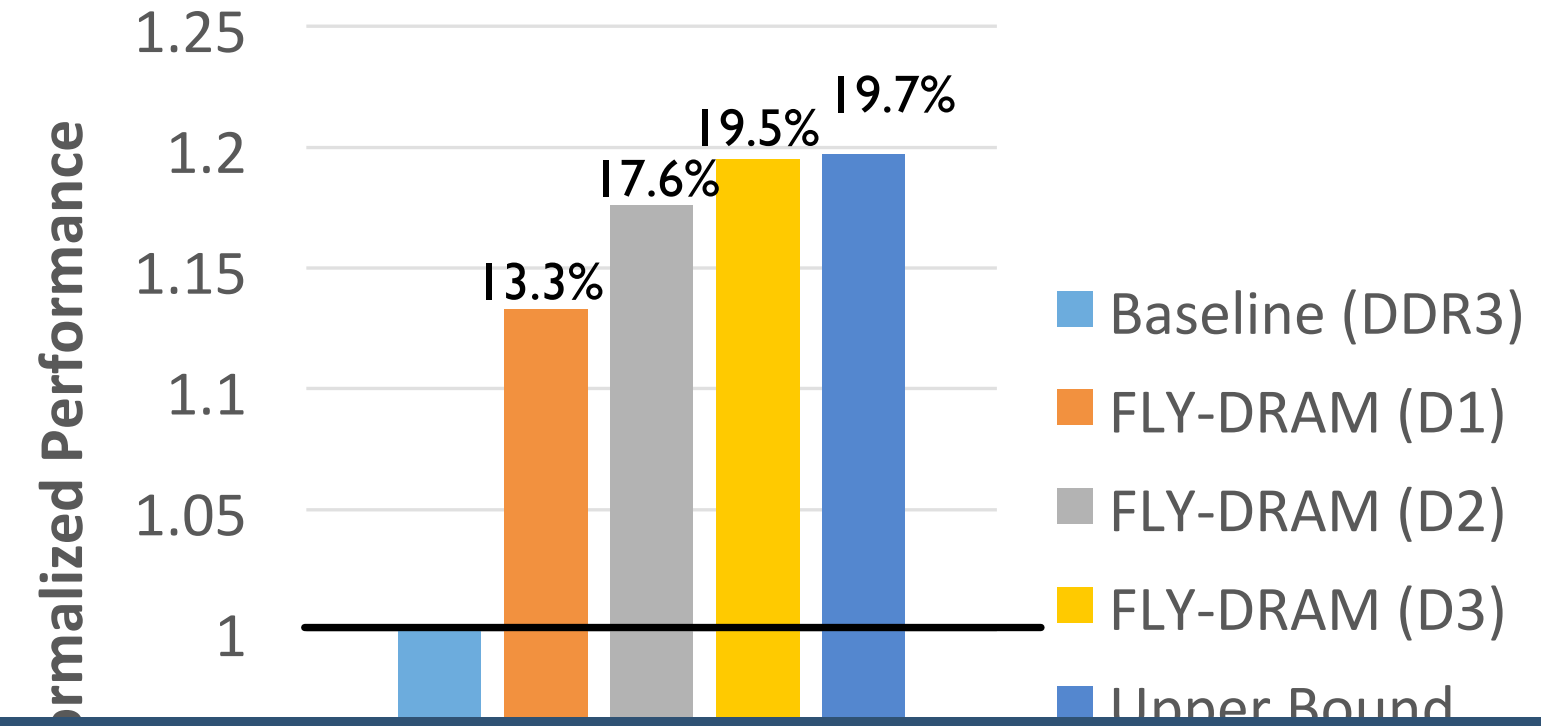
Chang+, "**Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization"**," SIGMETRICS 2016.

FLY-DRAM Configurations



Chang+, "**Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization"**," SIGMETRICS 2016.

Results



FLY-DRAM improves performance by exploiting spatial latency variation in DRAM

Chang+, "**Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization**," SIGMETRICS 2016.

FLY-DRAM: Advantages & Disadvantages

- Advantages

  + Reduces latency significantly

    + Exploits significant within-chip latency variation

- Disadvantages

  - Need to determine reliable operating latencies for different parts of a chip → higher testing cost

  - Slightly more complicated controller

Analysis of Latency Variation in DRAM Chips

- Kevin Chang, Abhijith Kashyap, Hasan Hassan, Samira Khan, Kevin Hsieh, Donghyuk Lee, Saugata Ghose, Gennady Pekhimenko, Tianshi Li, and Onur Mutlu,
  "Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization"
  Proceedings of the
  ACM International Conference on Measurement and Modeling of Computer Systems (***SIGMETRICS***), Antibes Juan-Les-Pins, France, June 2016.
  [Slides (pptx) (pdf)]
  [Source Code]

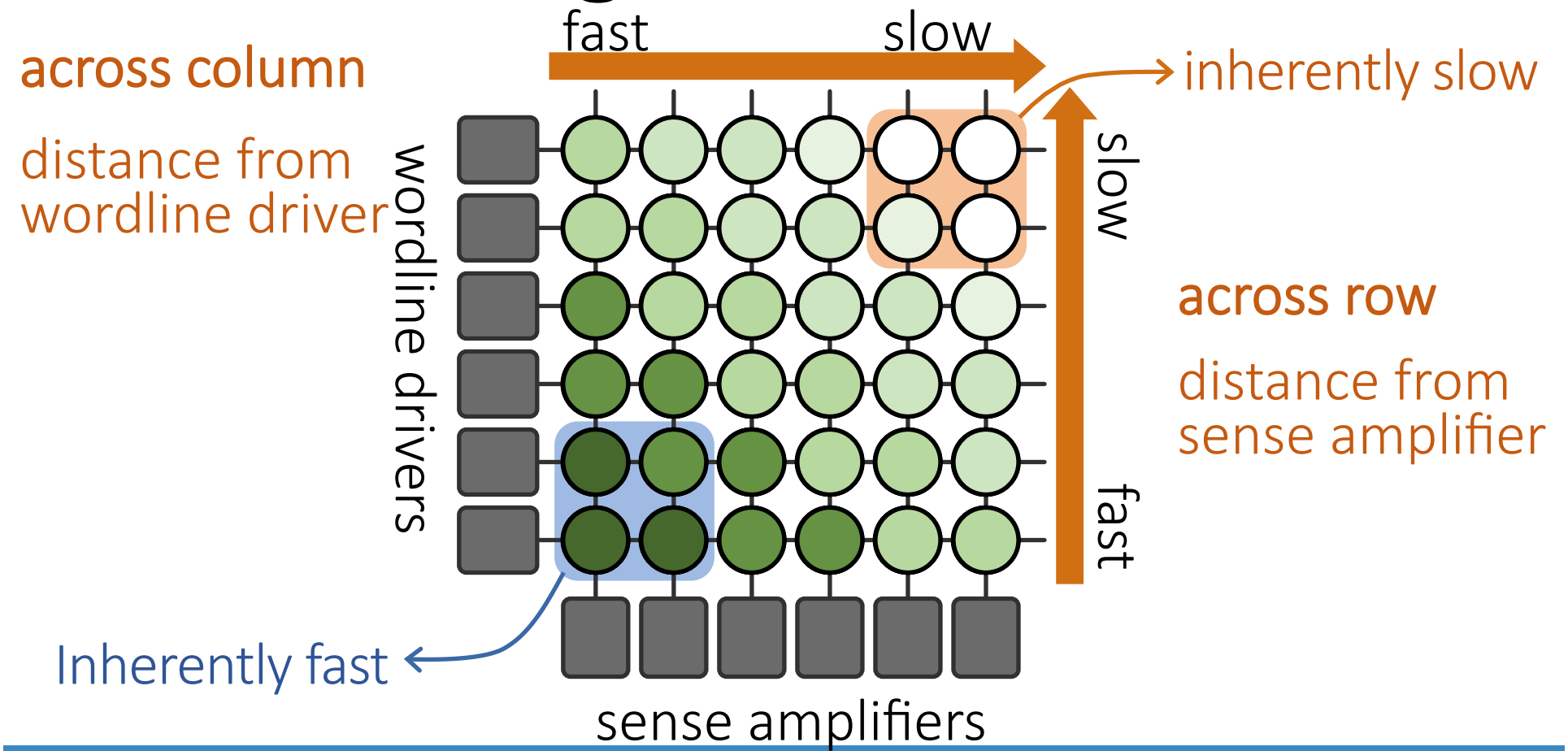
Understanding Latency Variation in Modern DRAM Chips: Experimental Characterization, Analysis, and Optimization

Kevin K. Chang[1] Abhijith Kashyap[1] Hasan Hassan[1,2]
Saugata Ghose[1] Kevin Hsieh[1] Donghyuk Lee[1] Tianshi Li[1,3]
Gennady Pekhimenko[1] Samira Khan[4] Onur Mutlu[5,1]

[1]Carnegie Mellon University [2]TOBB ETÜ [3]Peking University [4]University of Virginia [5]ETH Zürich

Why Is There Spatial Latency Variation Within a Chip?

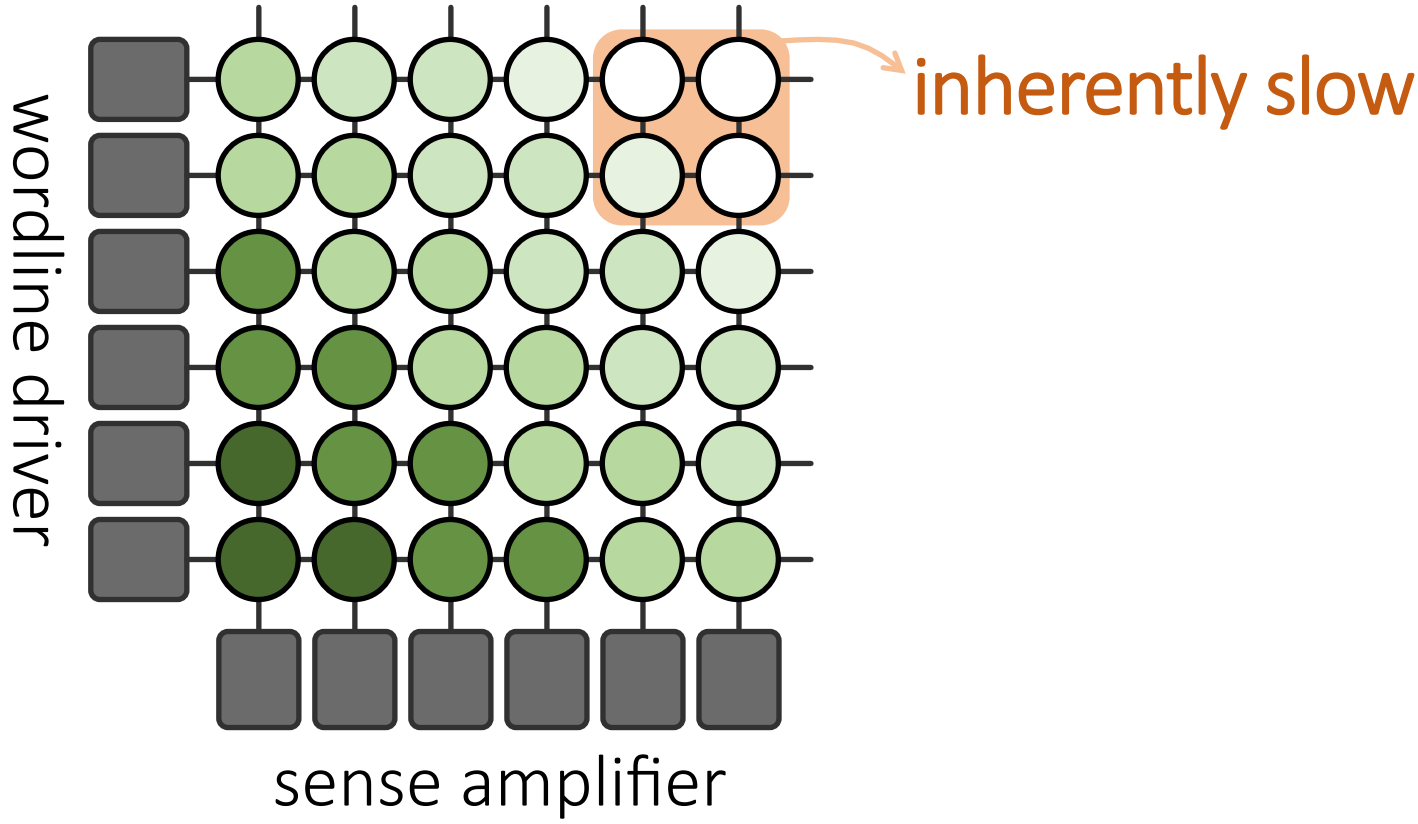
What Is Design-Induced Variation?



Systematic variation in cell access times caused by the ***physical organization*** of DRAM

DIVA Online **Profiling**

Design-**I**nduced-**V**ariation-**A**ware

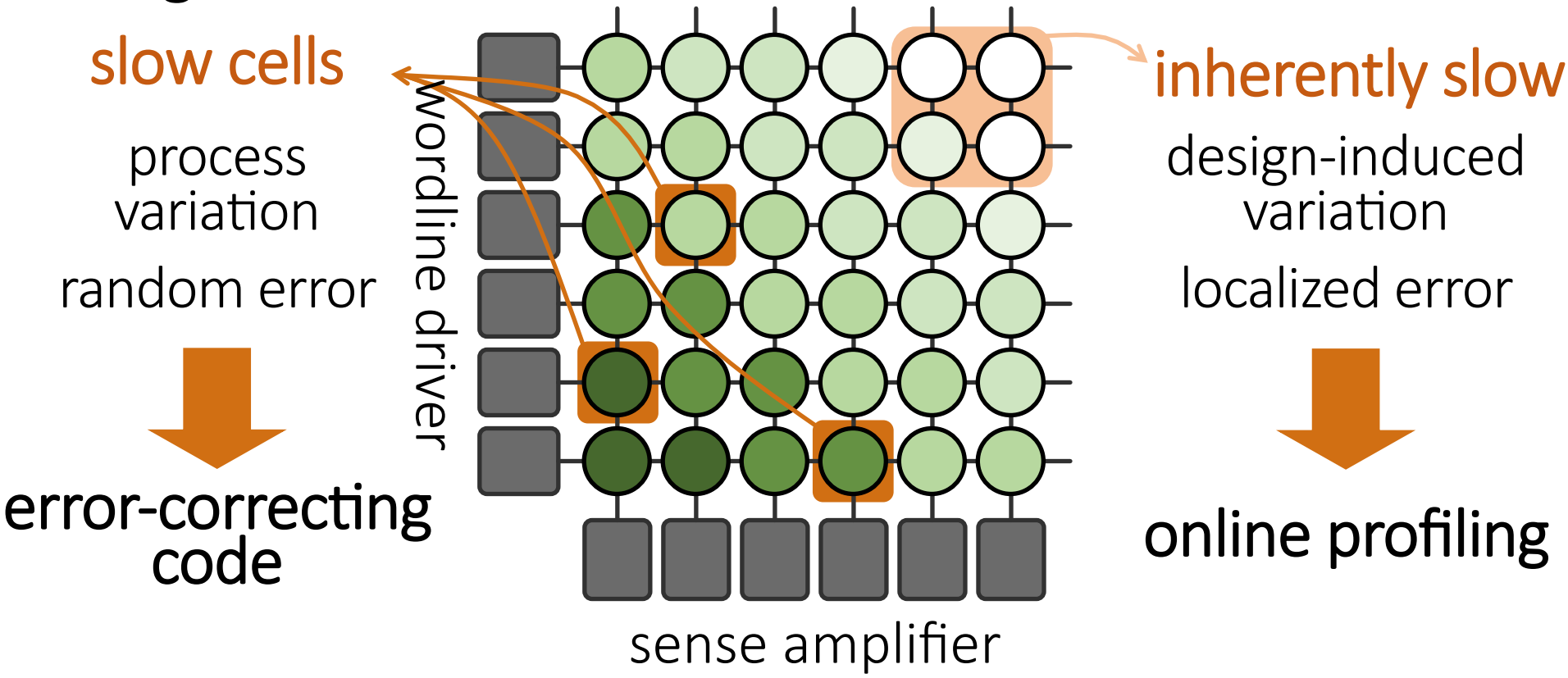


Profile *only slow regions* to determine min. latency

→ *Dynamic* & *low cost* latency optimization

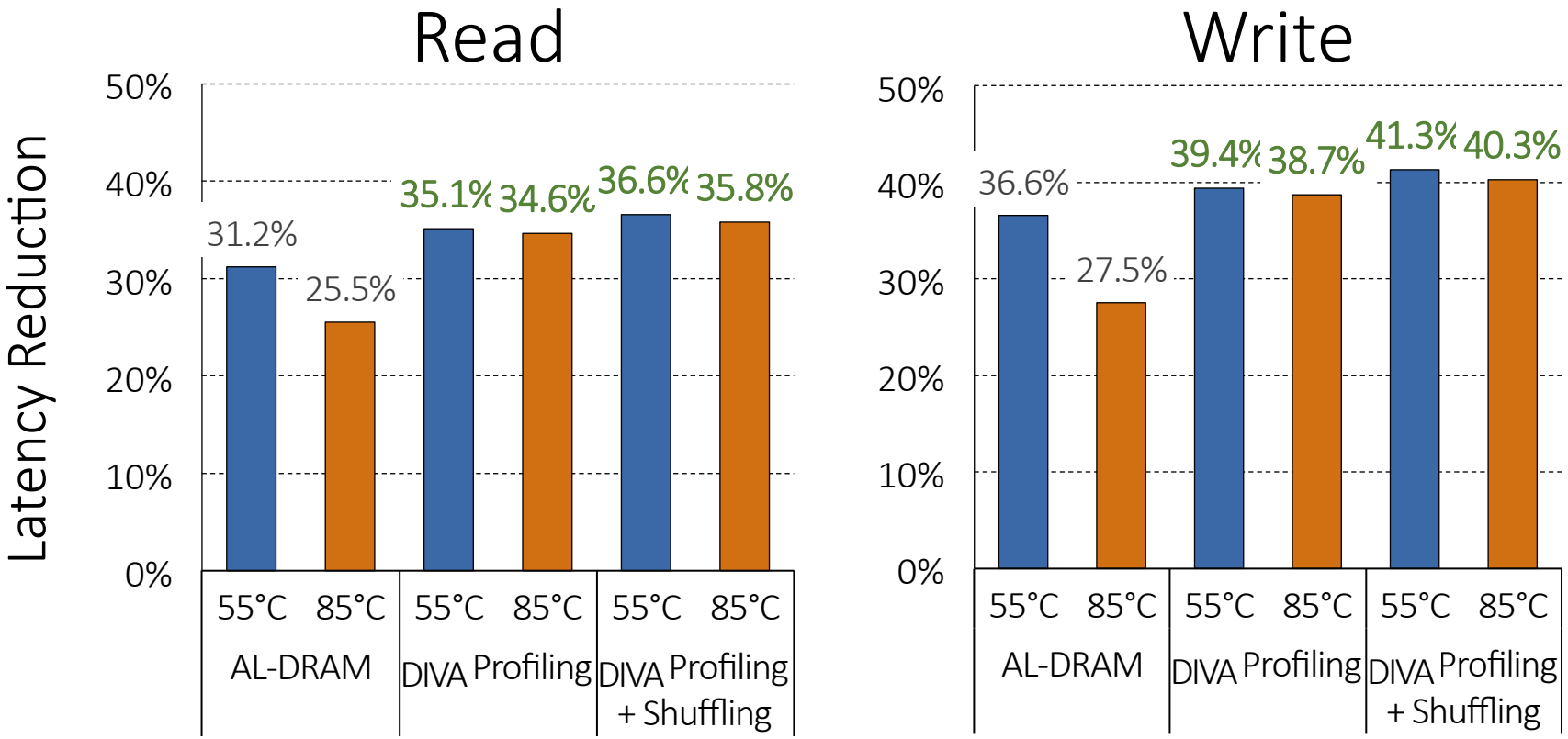
DIVA Online **Profiling**

Design-**I**nduced-**V**ariation-**A**ware



Combine error-correcting codes & online profiling
→ Reliably reduce DRAM latency

DIVA-DRAM Reduces Latency



DIVA-DRAM ***reduces latency more aggressively*** and uses ECC to correct random slow cells

DIVA-DRAM: Advantages & Disadvantages

- Advantages

++ Automatically finds the lowest reliable operating latency at system runtime (lower production-time testing cost)

+ Reduces latency more than prior methods (w/ ECC)

+ Reduces latency at high temperatures as well

- Disadvantages

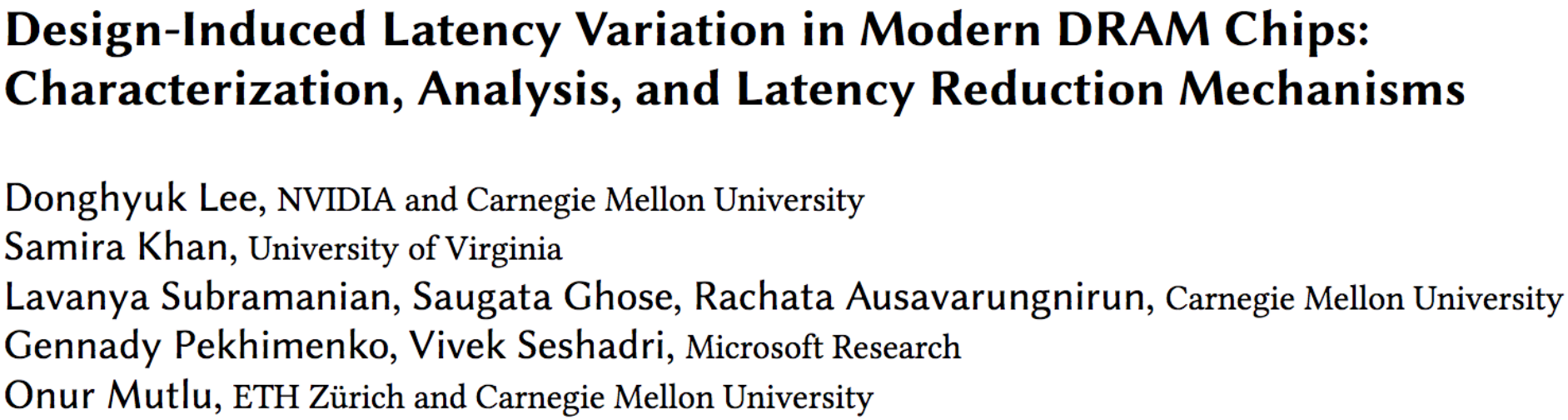
- Requires knowledge of inherently-slow regions

- Requires ECC (Error Correcting Codes)

- Imposes overhead during runtime profiling

Design-Induced Latency Variation in DRAM

- Donghyuk Lee, Samira Khan, Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Gennady Pekhimenko, Vivek Seshadri, and Onur Mutlu,
  "Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms"
  Proceedings of the
  *ACM International Conference on Measurement and Modeling of Computer Systems (**SIGMETRICS**)*, Urbana-Champaign, IL, USA, June 2017.



Design-Induced Latency Variation in Modern DRAM Chips: Characterization, Analysis, and Latency Reduction Mechanisms

Donghyuk Lee, NVIDIA and Carnegie Mellon University
Samira Khan, University of Virginia
Lavanya Subramanian, Saugata Ghose, Rachata Ausavarungnirun, Carnegie Mellon University
Gennady Pekhimenko, Vivek Seshadri, Microsoft Research
Onur Mutlu, ETH Zürich and Carnegie Mellon University

Understanding & Exploiting the Voltage-Latency-Reliability Relationship

High DRAM Power Consumption

- Problem: High DRAM (memory) power in today's systems

>40% in POWER7 (Ware+, HPCA'10)

>40% in GPU (Paul+, ISCA'15)

Low-Voltage Memory

- Existing DRAM designs to help reduce DRAM power by lowering supply voltage conservatively
  - $Power \propto Voltage^2$
- DDR3L (low-voltage) reduces voltage from 1.5V to 1.35V (**-10%**)
- LPDDR4 (low-power) employs low-power I/O interface with 1.2V (lower bandwidth)

Can we reduce DRAM power and energy by *further* reducing supply voltage?

Goals

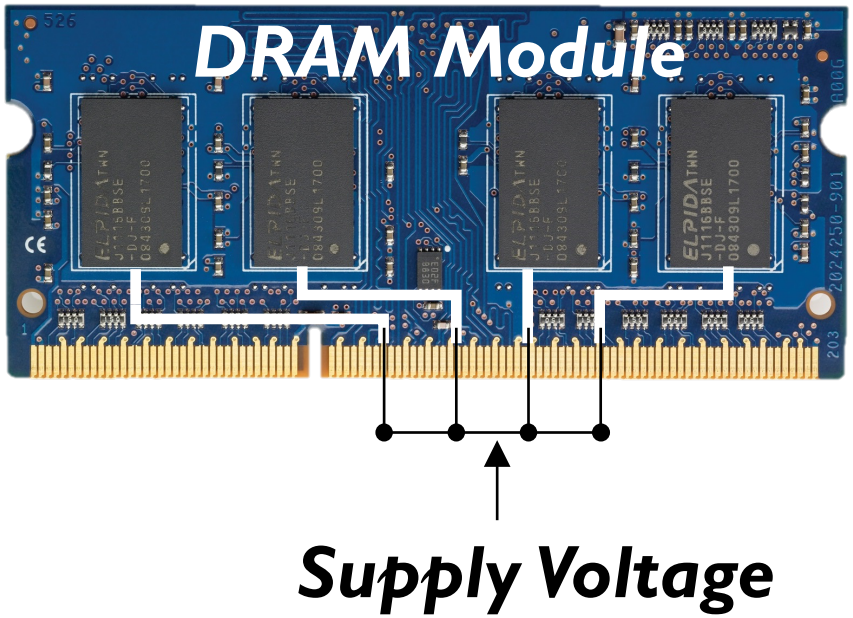
1 Understand and characterize the various characteristics of DRAM under **reduced voltage**

2 Develop a mechanism that reduces DRAM energy by **lowering voltage** while keeping performance loss within a target

Key Questions

- How does reducing voltage affect ***reliability*** (errors)?
- How does reducing voltage affect ***DRAM latency***?
- How do we design a new DRAM energy reduction mechanism?

Supply Voltage Control on DRAM



Adjust the *supply voltage* to every chip on the same module

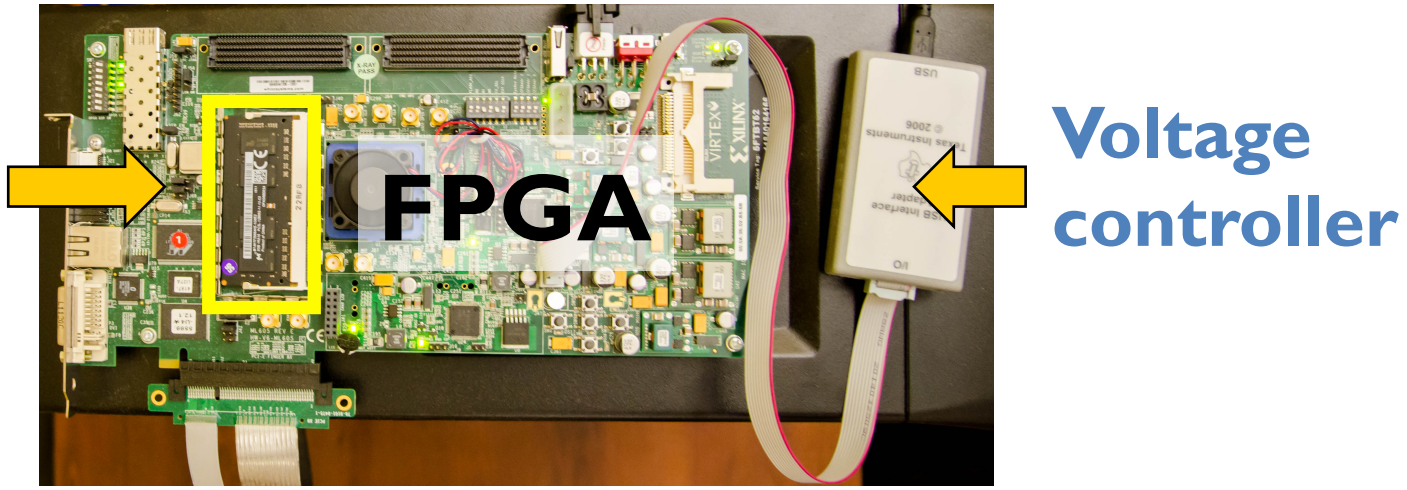
Custom Testing Platform

SoftMC [Hassan+, HPCA'17]: FPGA testing platform to

1) Adjust supply voltage to DRAM modules

2) Schedule DRAM commands to DRAM modules

Existing systems: DRAM commands not exposed to users



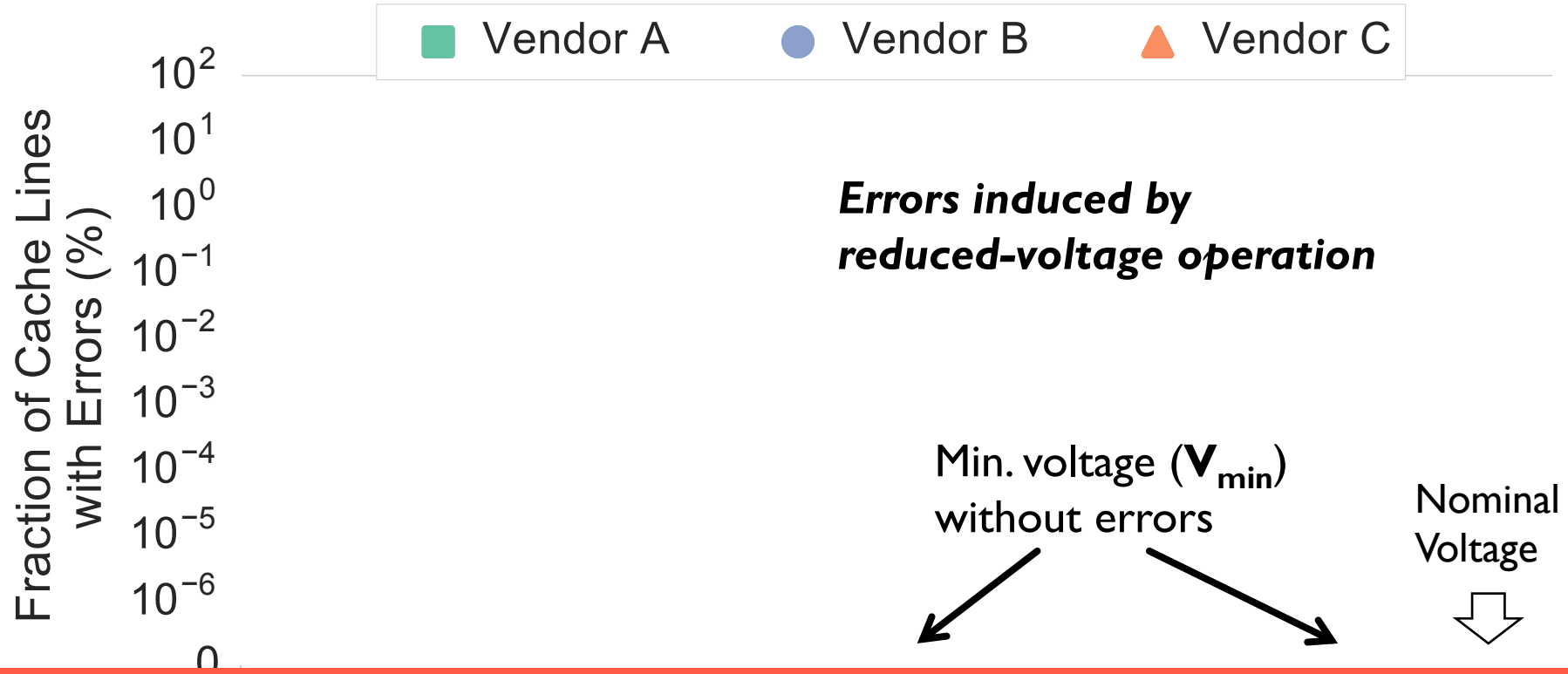
https://github.com/CMU-SAFARI/DRAM-Voltage-Study

Tested DRAM Modules

- **124 DDR3L** (low-voltage) DRAM chips
  - **31 SO-DIMMs**
  - **1.35V** (DDR3 uses 1.5V)
  - Density: 4Gb per chip
  - Three major vendors/manufacturers
  - Manufacturing dates: 2014-2016

- Iteratively read every bit in each 4Gb chip under a wide range of supply voltage levels: 1.35V to 1.0V (**-26%**)

Reliability Worsens with Lower Voltage

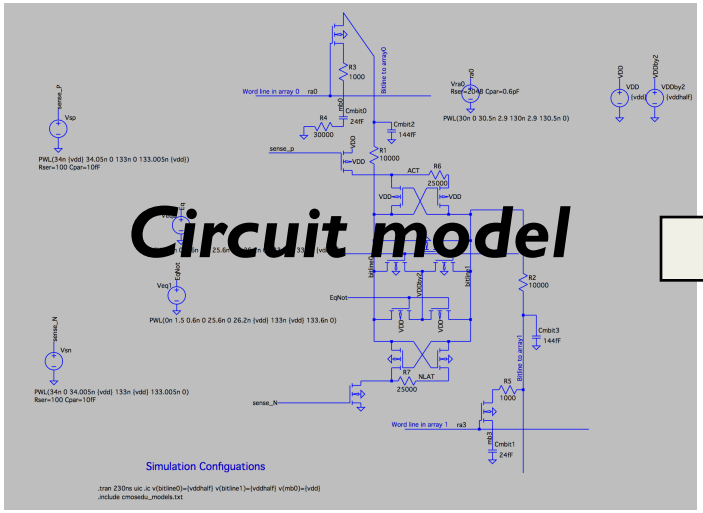


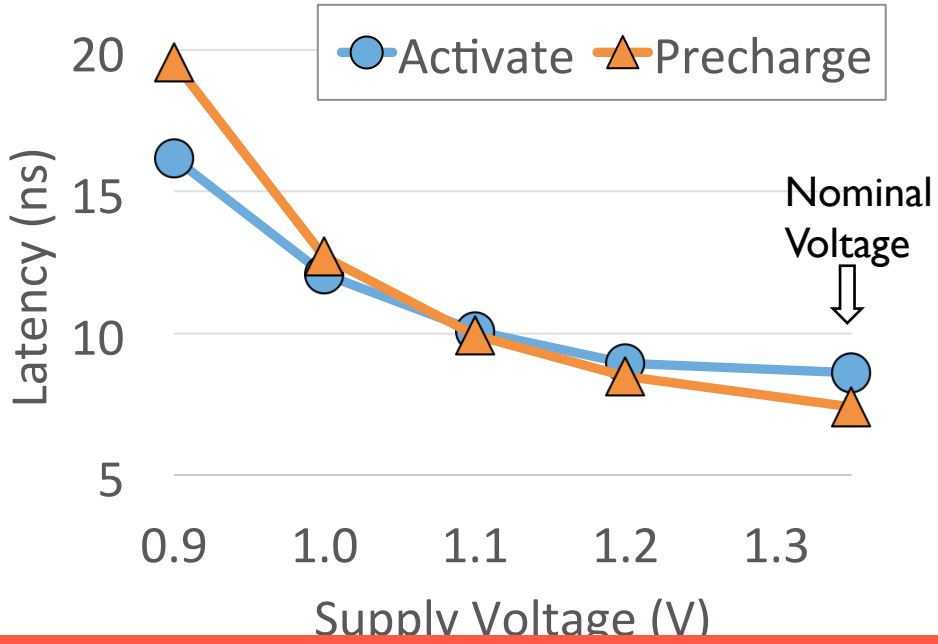
Reducing voltage below V_{min} causes an increasing number of errors

Source of Errors

Detailed circuit simulations (SPICE) of a DRAM cell array to model the behavior of DRAM operations

https://github.com/CMU-SAFARI/DRAM-Voltage-Study

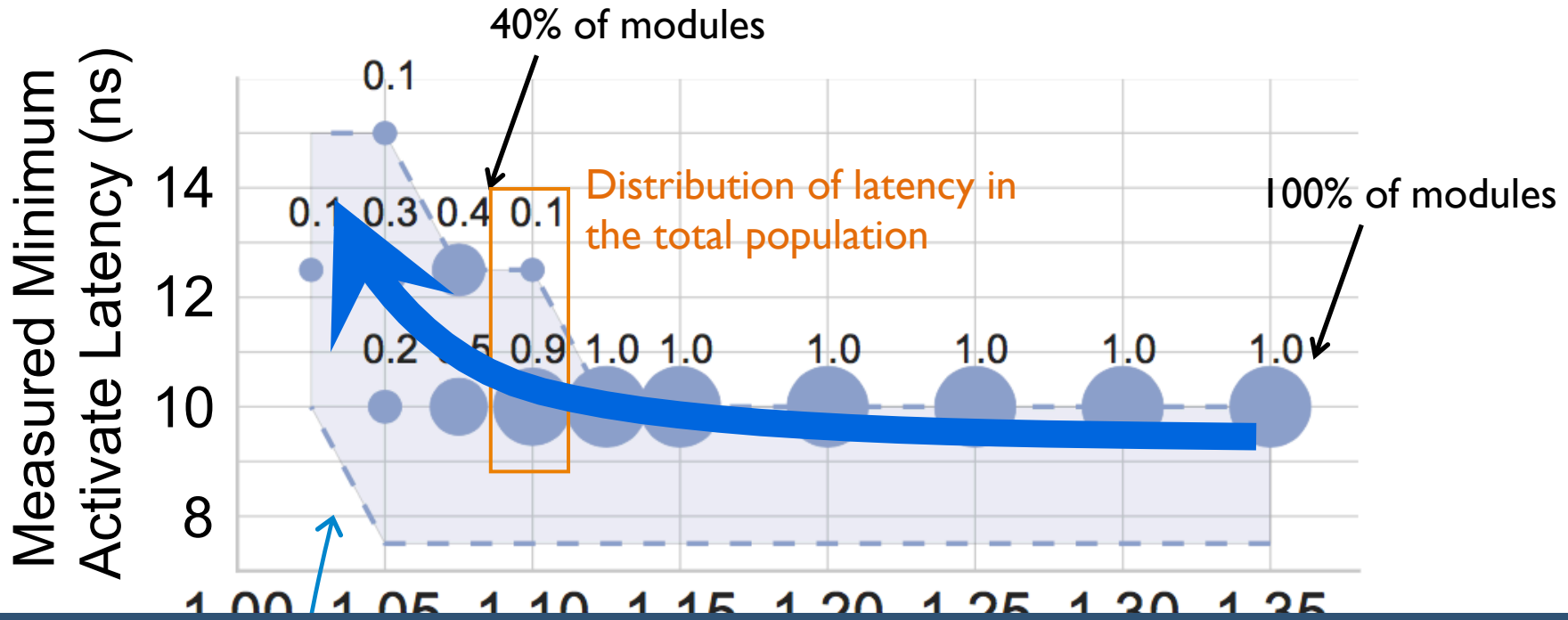




Reliable low-voltage operation requires higher latency

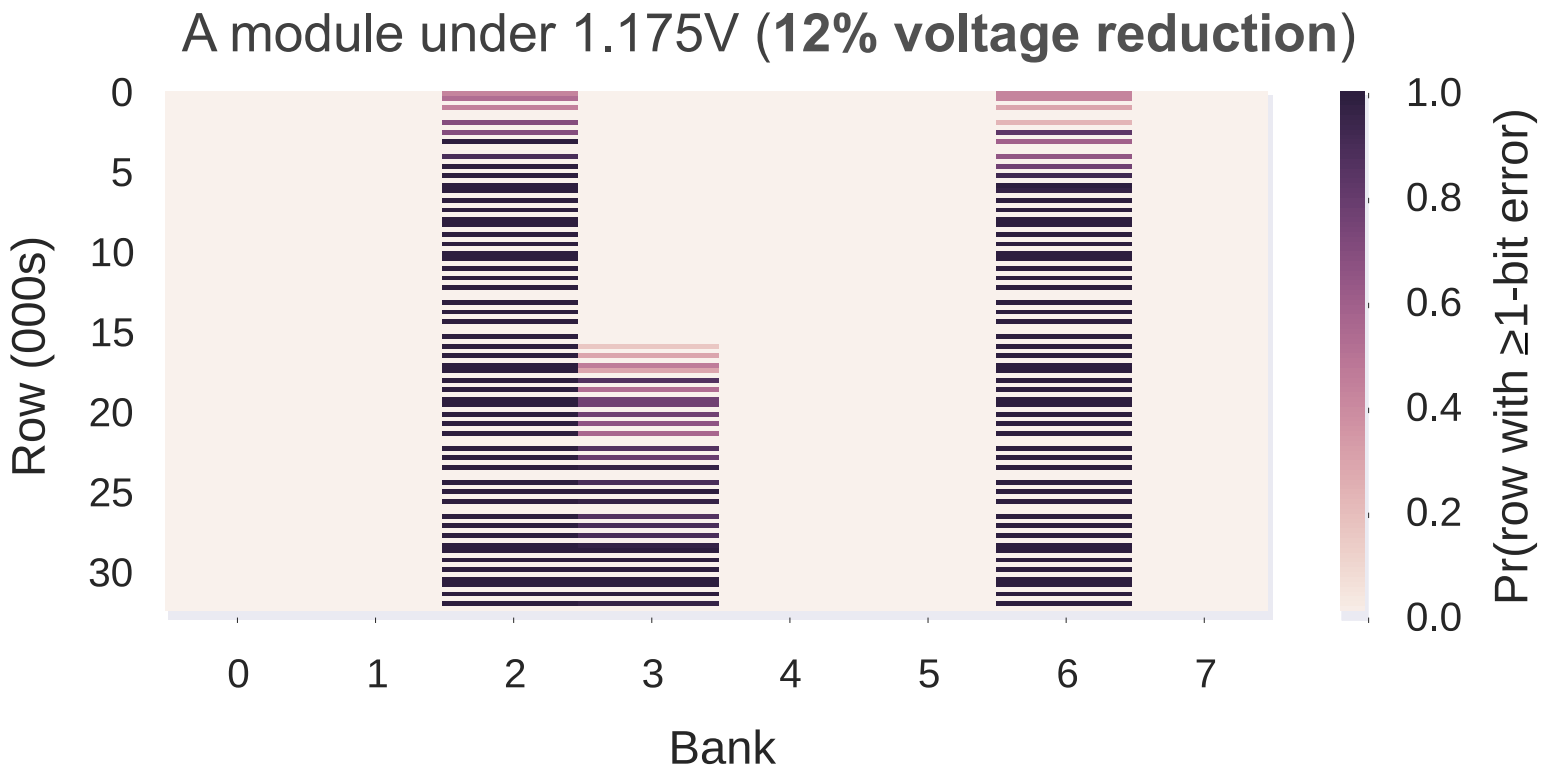
DIMMs Operating at Higher Latency

Measured minimum latency that *does **not** cause errors* in DRAM modules



DRAM requires longer latency to access data without errors at lower voltage

Spatial Locality of Errors



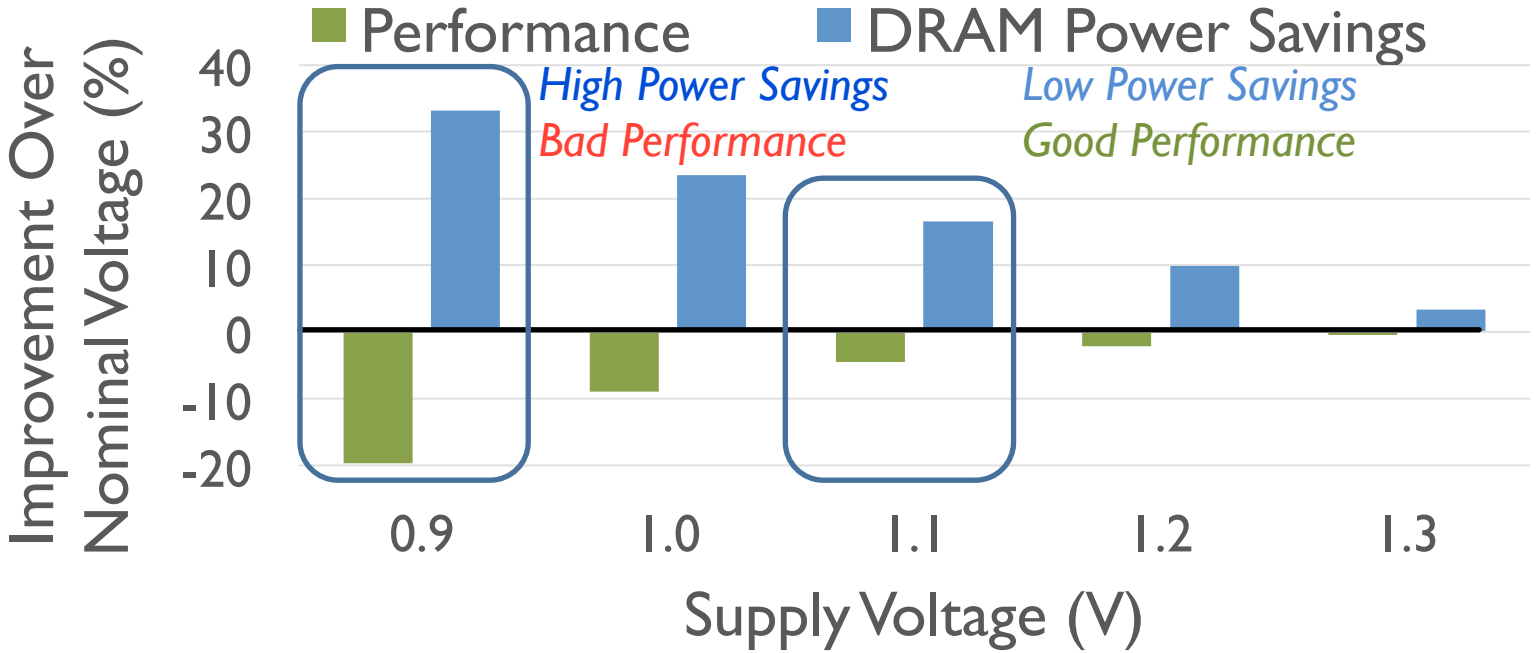
Errors concentrate in certain regions

Summary of Key Experimental Observations

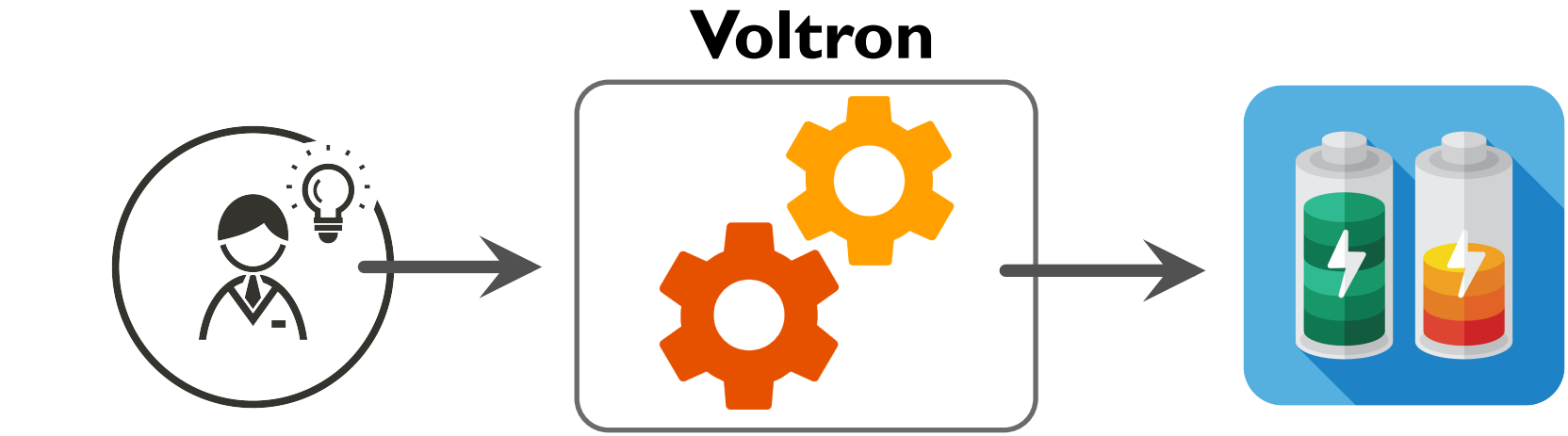
- Voltage-induced errors increase as voltage reduces further below V_{min}
- Errors exhibit spatial locality
- Increasing the latency of DRAM operations mitigates voltage-induced errors

DRAM Voltage Adjustment to Reduce Energy

- Goal: Exploit the trade-off between voltage and latency to reduce energy consumption
- Approach: Reduce DRAM voltage **reliably**
  - Performance loss due to increased latency at lower voltage



Voltron Overview

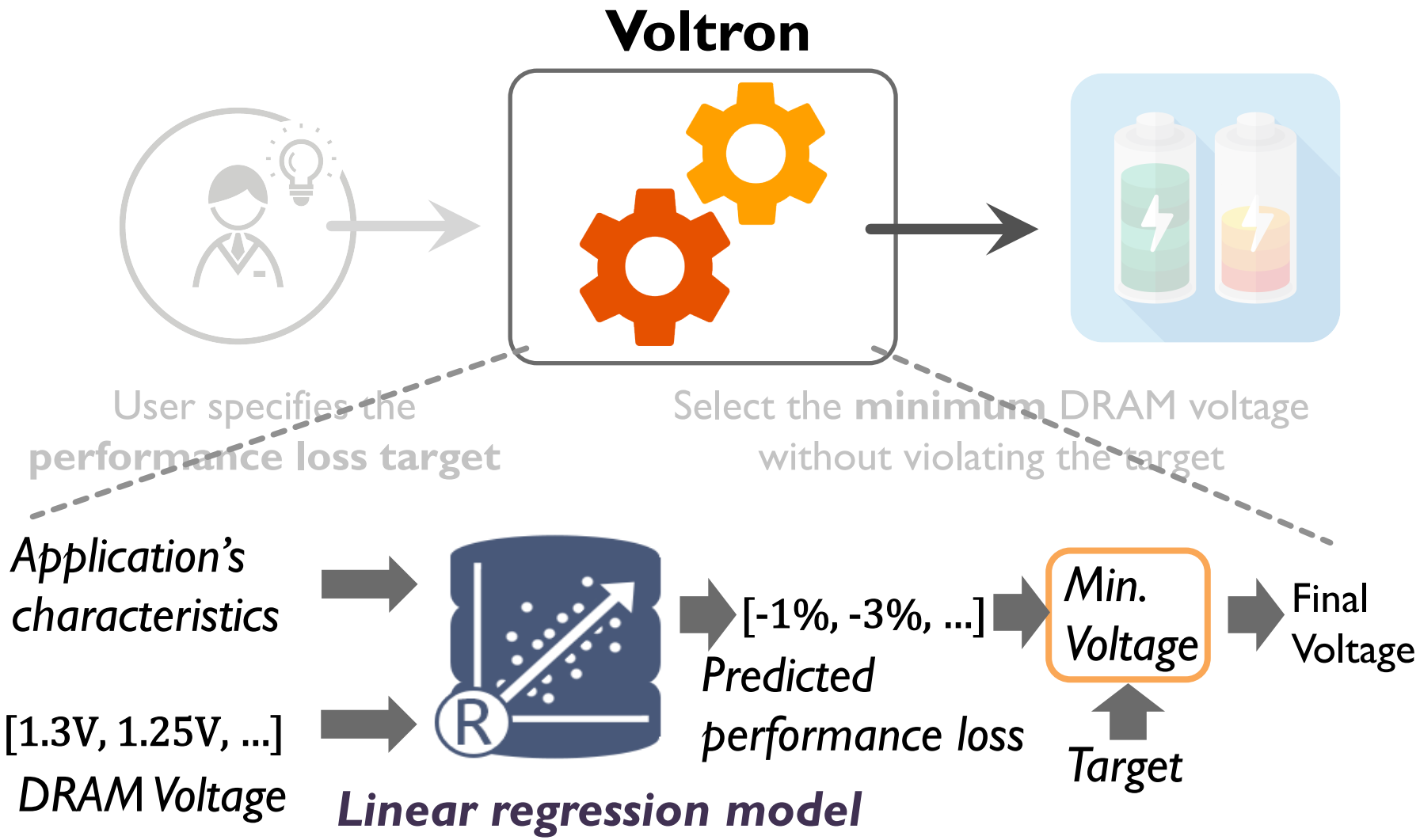


User specifies the **performance loss target**

Select the **minimum** DRAM voltage without violating the target

How do we predict performance loss due to increased latency under low DRAM voltage?

Linear Model to Predict Performance



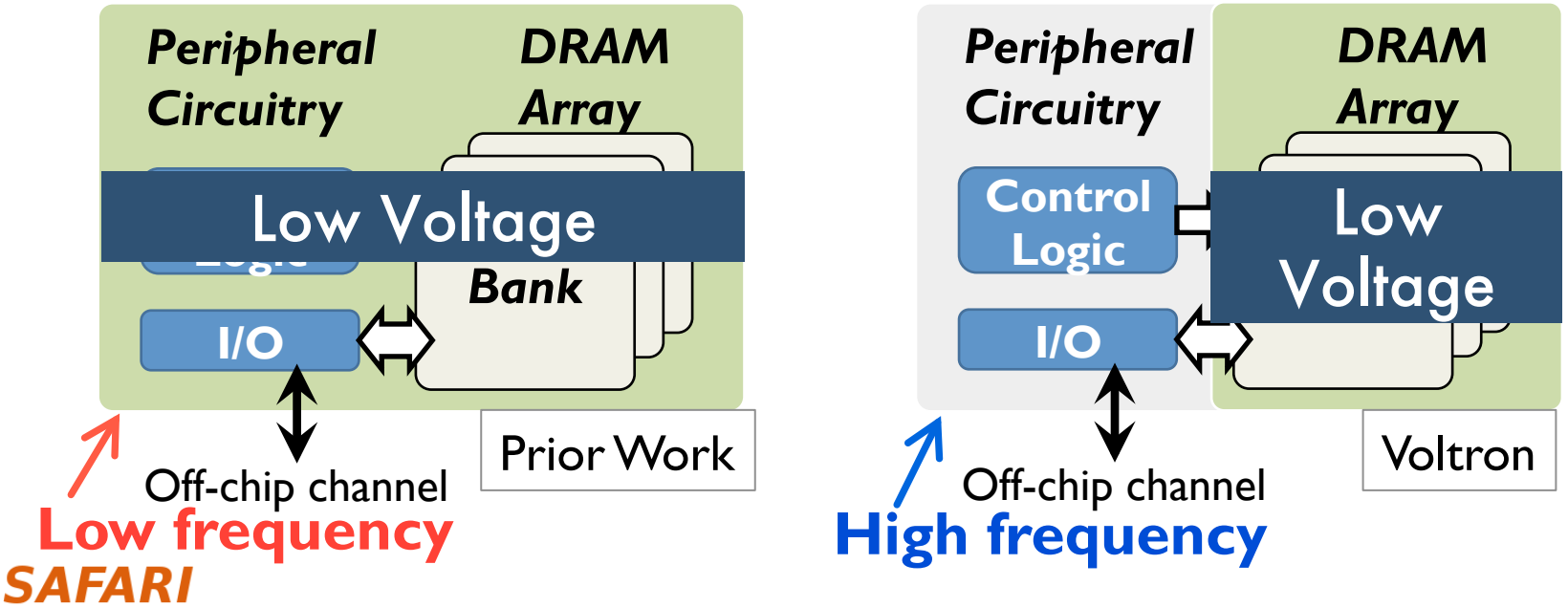
Regression Model to Predict Performance

- Application's characteristics for the model:
  - ***Memory intensity***: Frequency of last-level cache misses
  - ***Memory stall time***: Amount of time memory requests stall commit inside CPU

- Handling multiple applications:
  - Predict a performance loss for each application
  - Select the minimum voltage that satisfies the performance target for all applications

Comparison to Prior Work

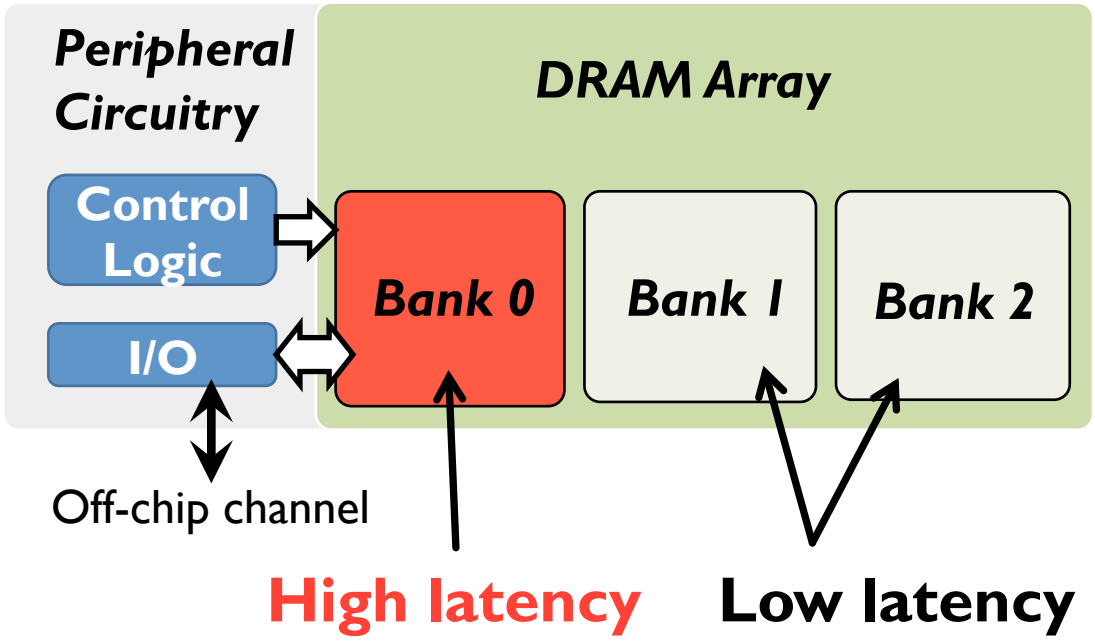
- Prior work: Dynamically scale *frequency and voltage* of the entire DRAM based on bandwidth demand [David+, ICAC'11]
  - Problem: Lowering voltage on the peripheral circuitry decreases channel frequency (memory data throughput)
- Voltron: Reduce voltage to only DRAM array without changing the voltage to peripheral circuitry



Exploiting Spatial Locality of Errors

Key idea: Increase the latency only for DRAM banks that observe errors under low voltage

- Benefit: Higher performance

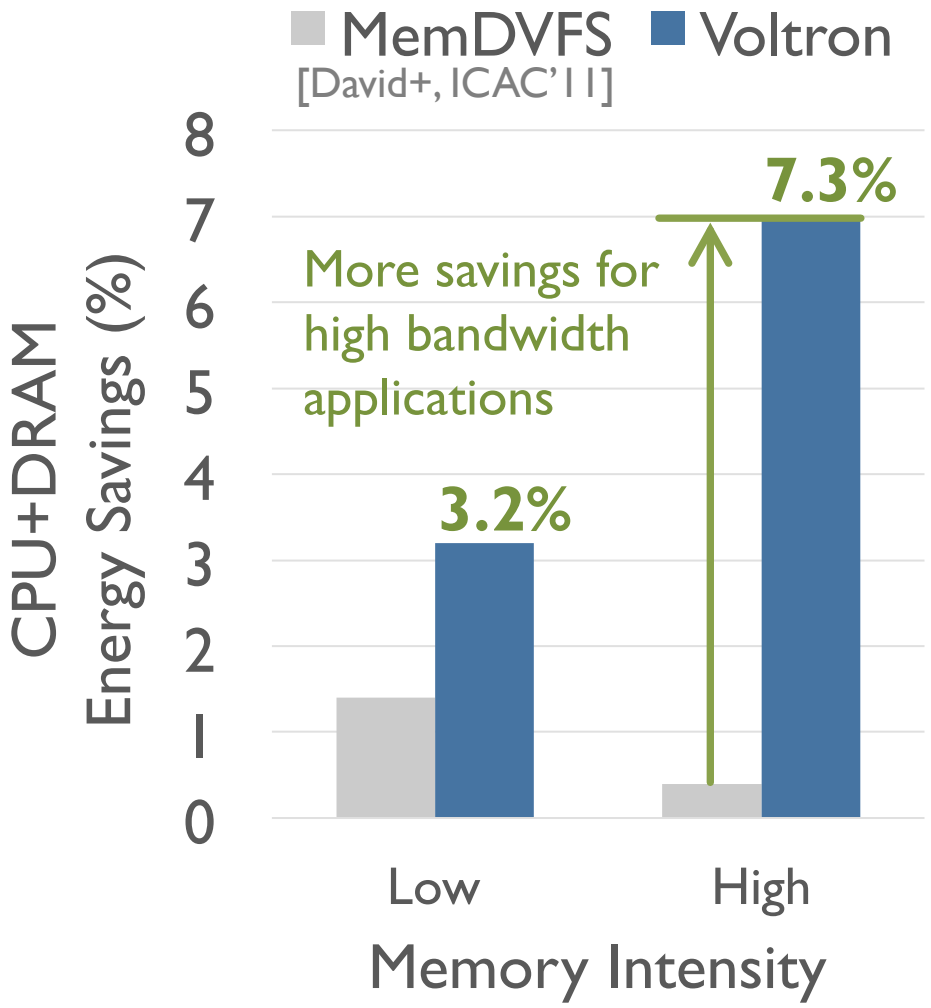


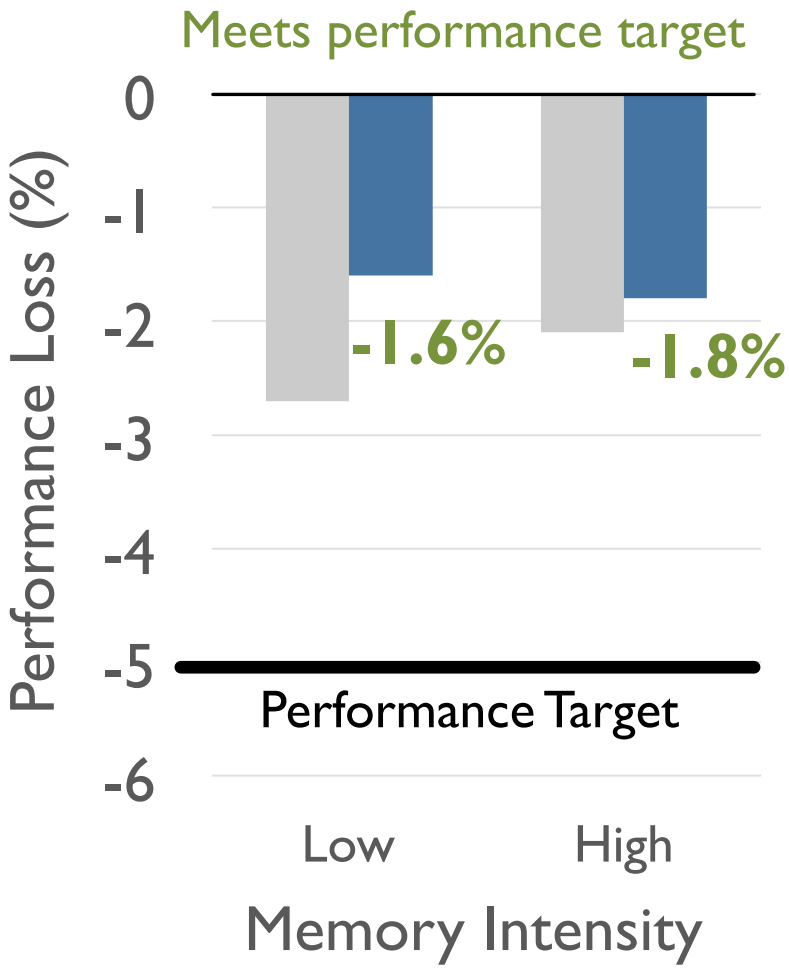
Voltron Evaluation Methodology

- **Cycle-level simulator**: Ramulator [CAL'15]
  - **McPAT** and **DRAMPower** for energy measurement

  https://github.com/CMU-SAFARI/ramulator
- **4-core** system with DDR3L memory
- **Benchmarks**: SPEC2006, YCSB
- Comparison to prior work: **MemDVFS** [David+, ICAC'11]
  - Dynamic DRAM frequency and voltage scaling
  - Scaling based on the *memory bandwidth consumption*

Energy Savings with Bounded Performance





Voltron: Advantages & Disadvantages

- Advantages

  + Can trade-off between voltage and latency to improve energy or performance

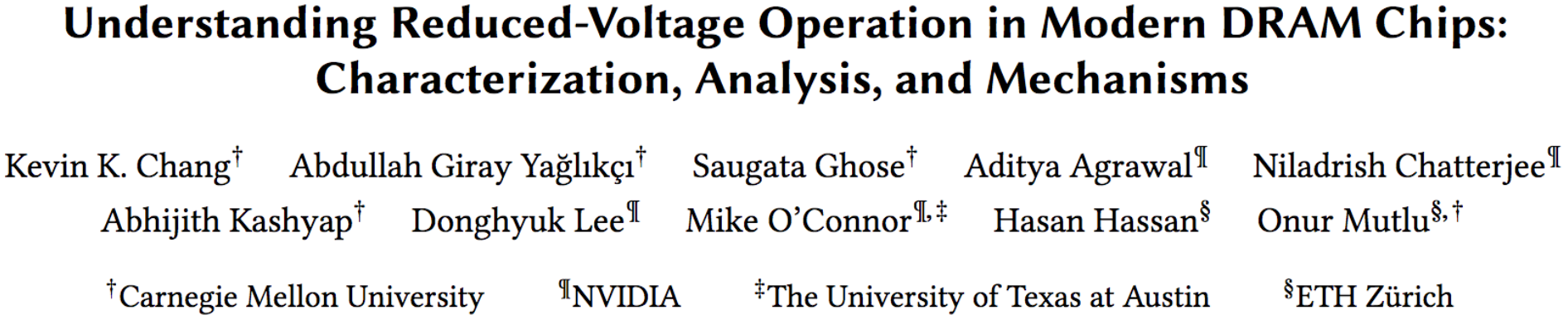
  + Can exploit the high voltage margin present in DRAM

- Disadvantages

  - Requires finding the reliable operating voltage for each chip → higher testing cost

Analysis of Latency-Voltage in DRAM Chips

- Kevin Chang, A. Giray Yaglikci, Saugata Ghose, Aditya Agrawal, Niladrish Chatterjee, Abhijith Kashyap, Donghyuk Lee, Mike O'Connor, Hasan Hassan, and Onur Mutlu,
  "Understanding Reduced-Voltage Operation in Modern DRAM Devices: Experimental Characterization, Analysis, and Mechanisms"
  Proceedings of the
  ACM International Conference on Measurement and Modeling of Computer Systems (***SIGMETRICS***), Urbana-Champaign, IL, USA, June 2017.



Understanding Reduced-Voltage Operation in Modern DRAM Chips: Characterization, Analysis, and Mechanisms

Kevin K. Chang[†] Abdullah Giray Yağlıkçı[†] Saugata Ghose[†] Aditya Agrawal[¶] Niladrish Chatterjee[¶]
Abhijith Kashyap[†] Donghyuk Lee[¶] Mike O'Connor[¶,‡] Hasan Hassan[§] Onur Mutlu[§,†]

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

And, What If …

- … we can sacrifice reliability of some data to access it with even lower latency?

Challenge and Opportunity for Future

Fundamentally

Low Latency

Computing Architectures

More Fundamentally Reducing Latency and Energy

Processing In Memory

Observation and Opportunity

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

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

The Problem

Data access is the major performance and energy bottleneck

Our current

design principles

cause great energy waste

The Problem

Processing of data

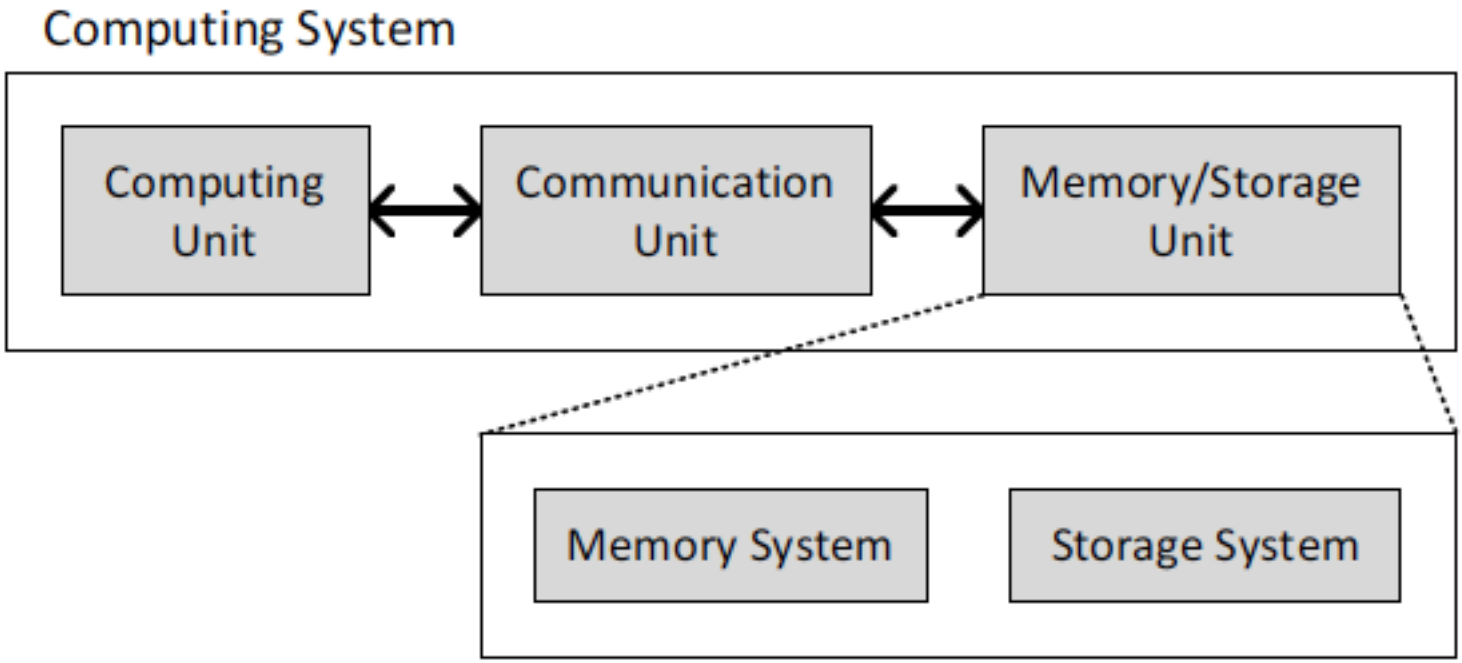
is performed

far away from the data

A Computing System

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

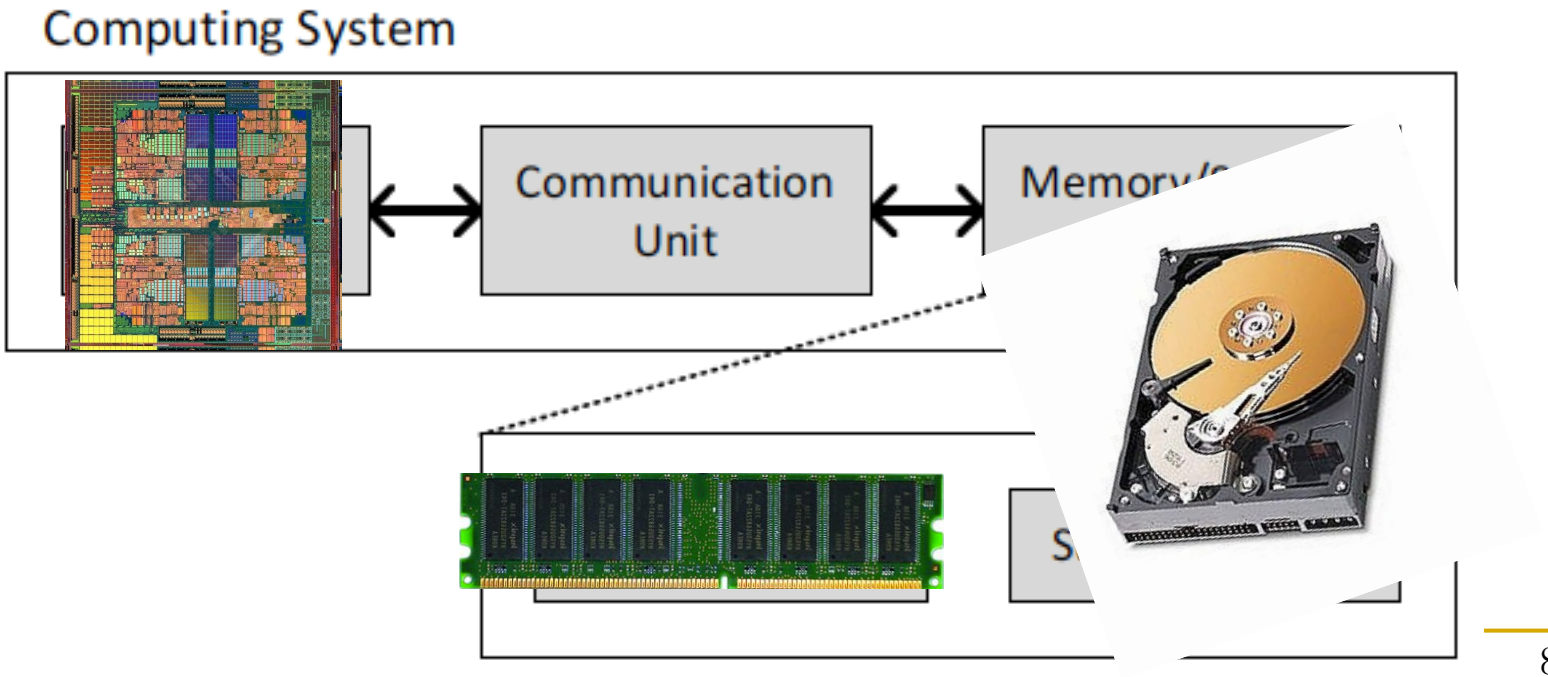
Burks, Goldstein, von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," 1946.



A Computing System

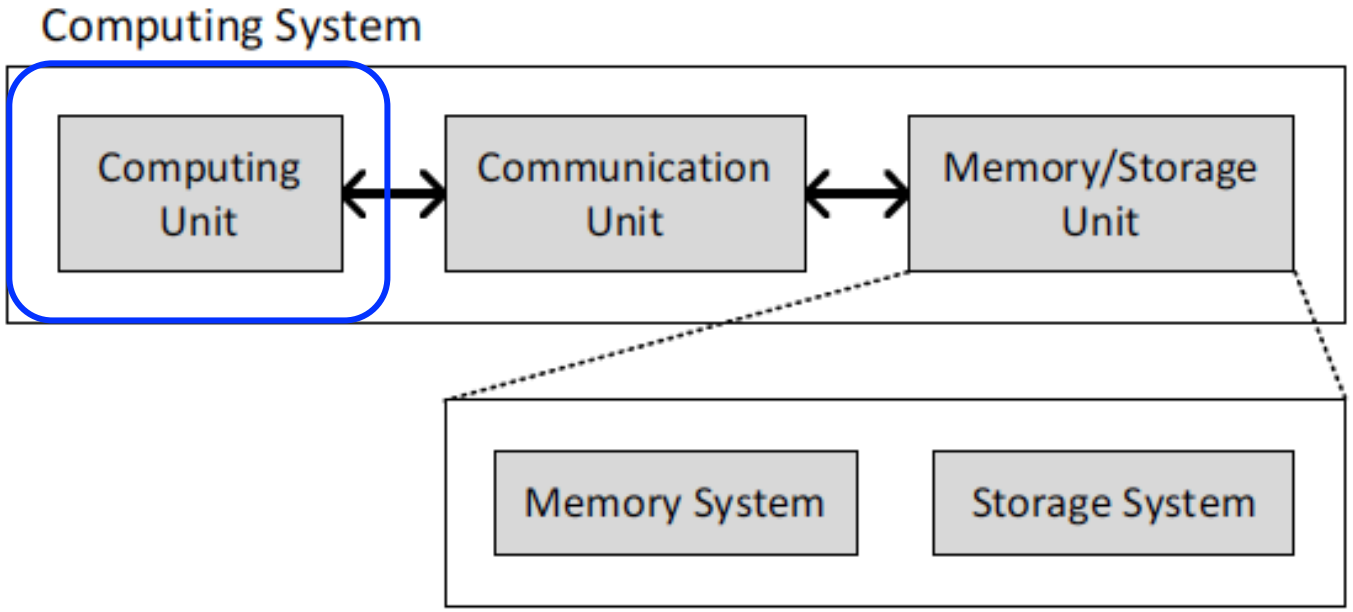
- Three key components
- Computation
- Communication
- Storage/memory

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



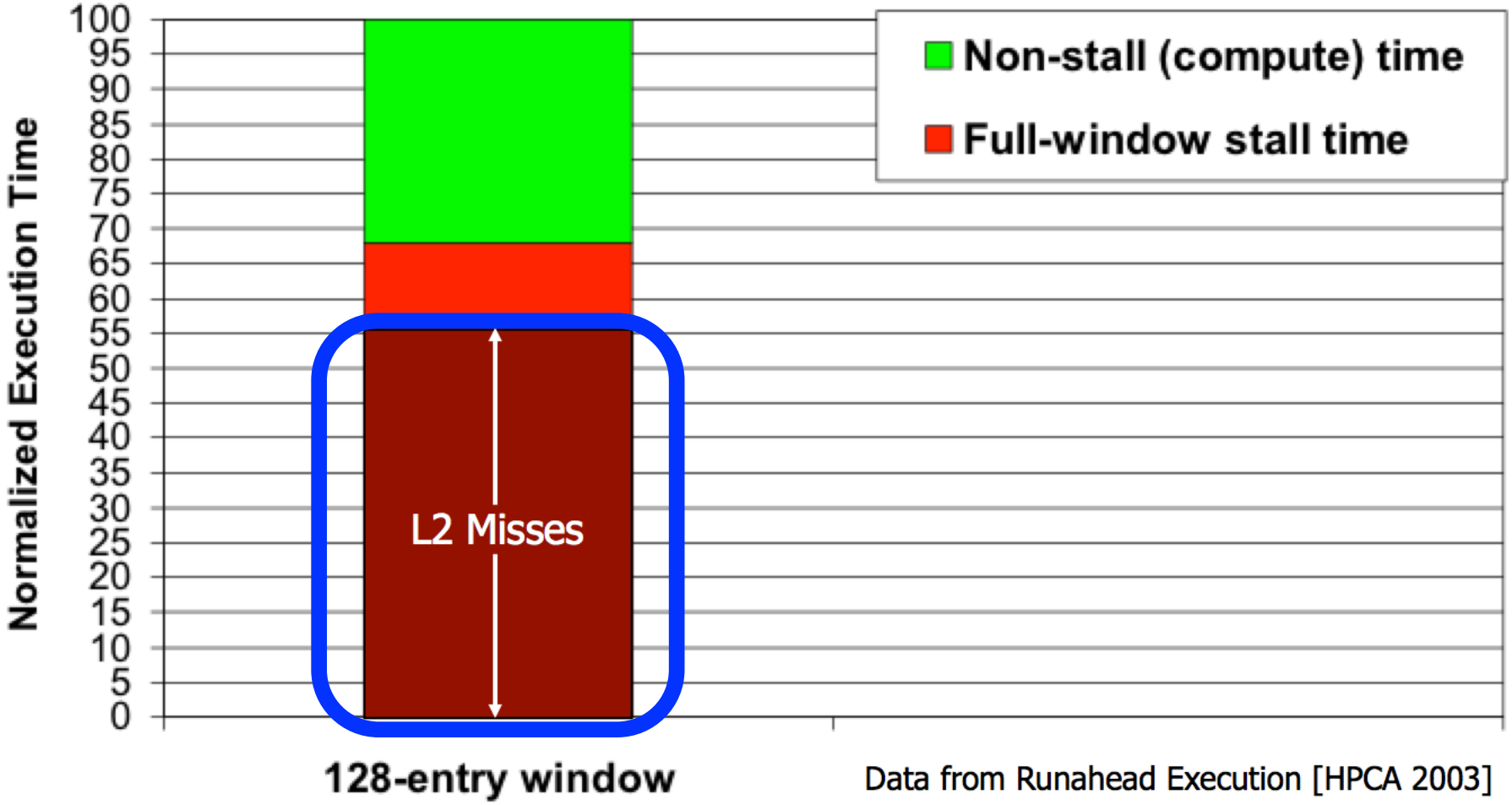
Today's Computing Systems

- 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 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

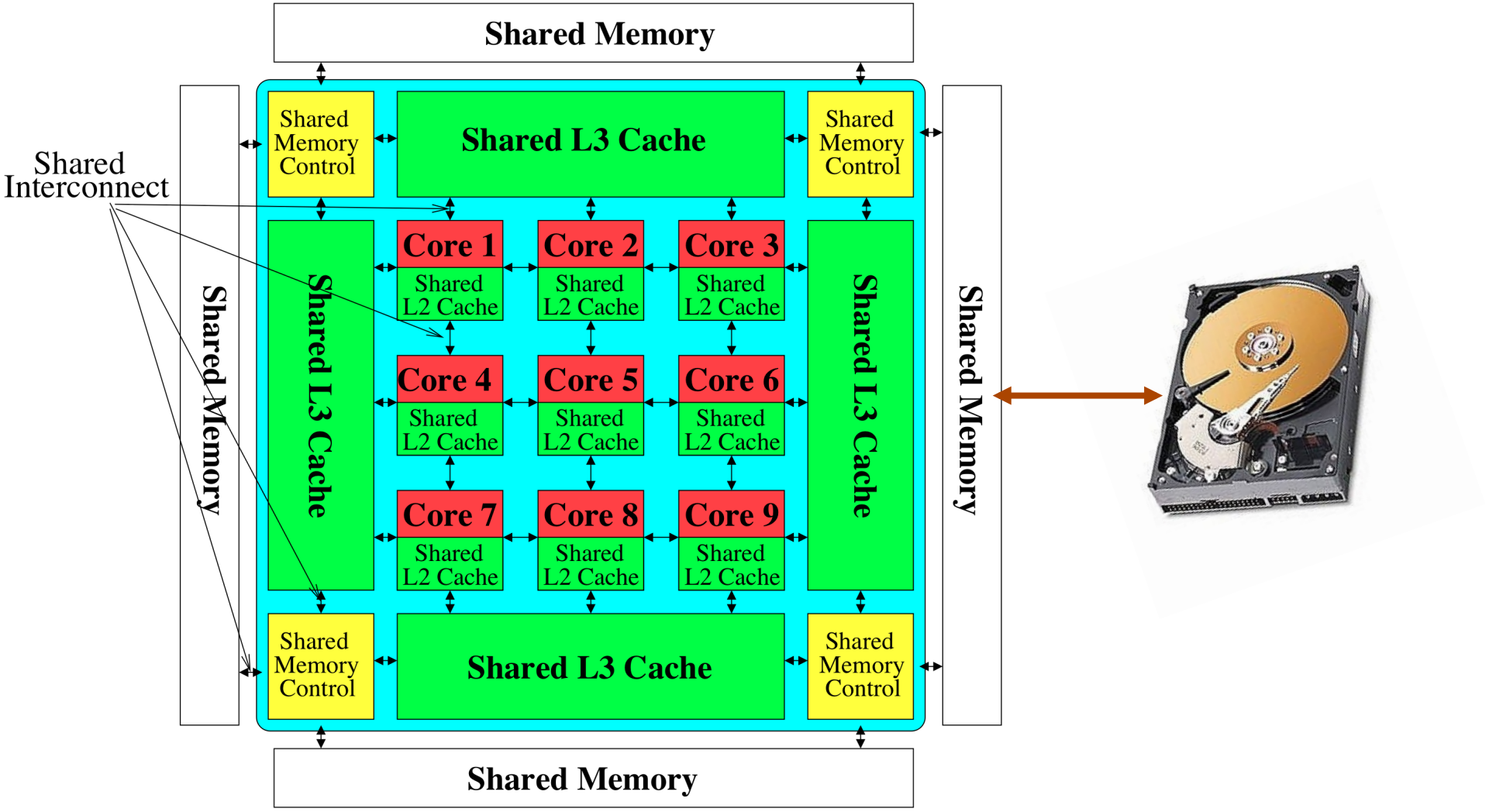
- Overly complex and bloated processor (and accelerators)
  - To tolerate data access from memory
  - Complex hierarchies and mechanisms

  → 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

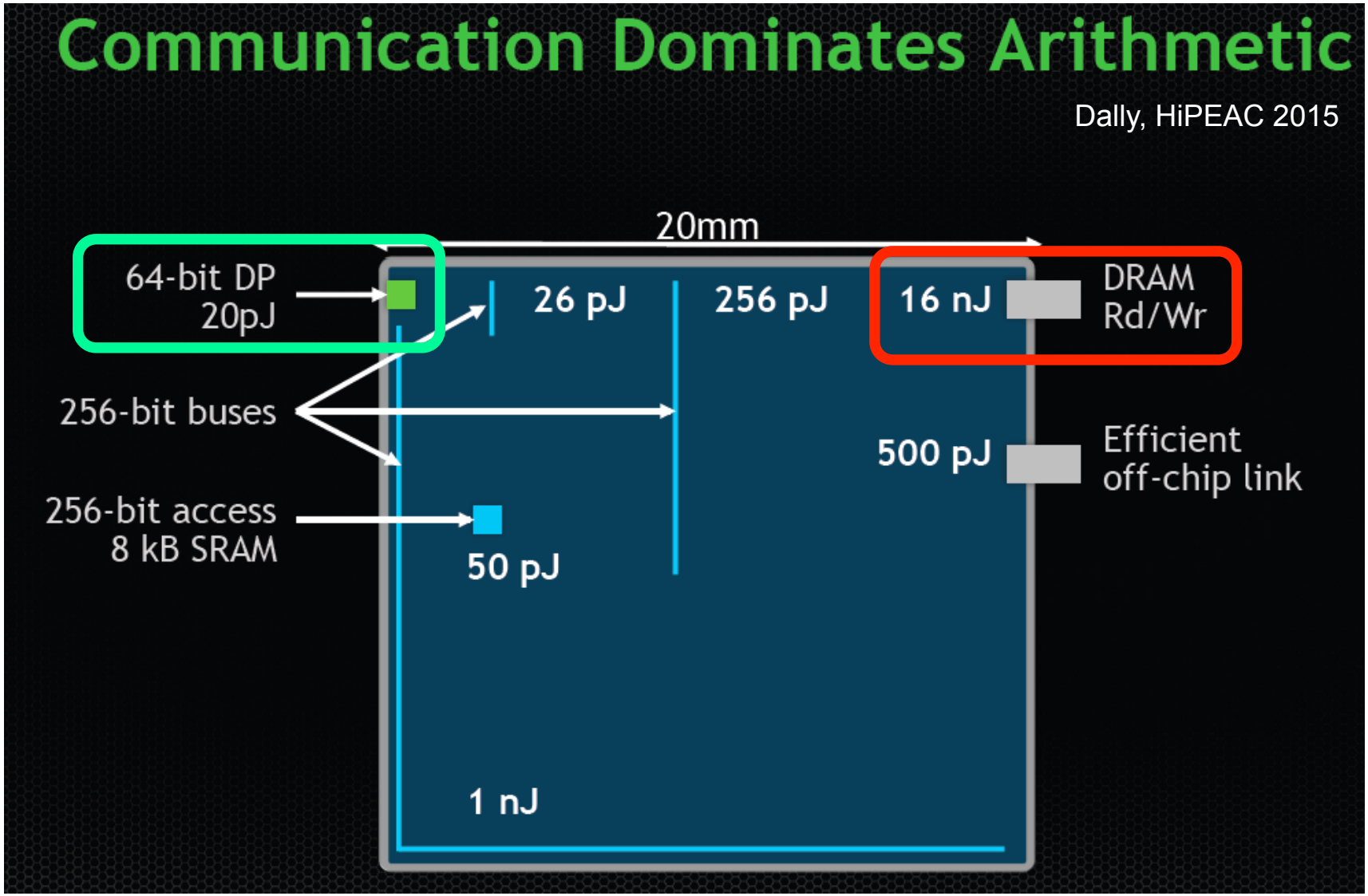
- Applications are increasingly data hungry

2. Energy consumption is a key limiter

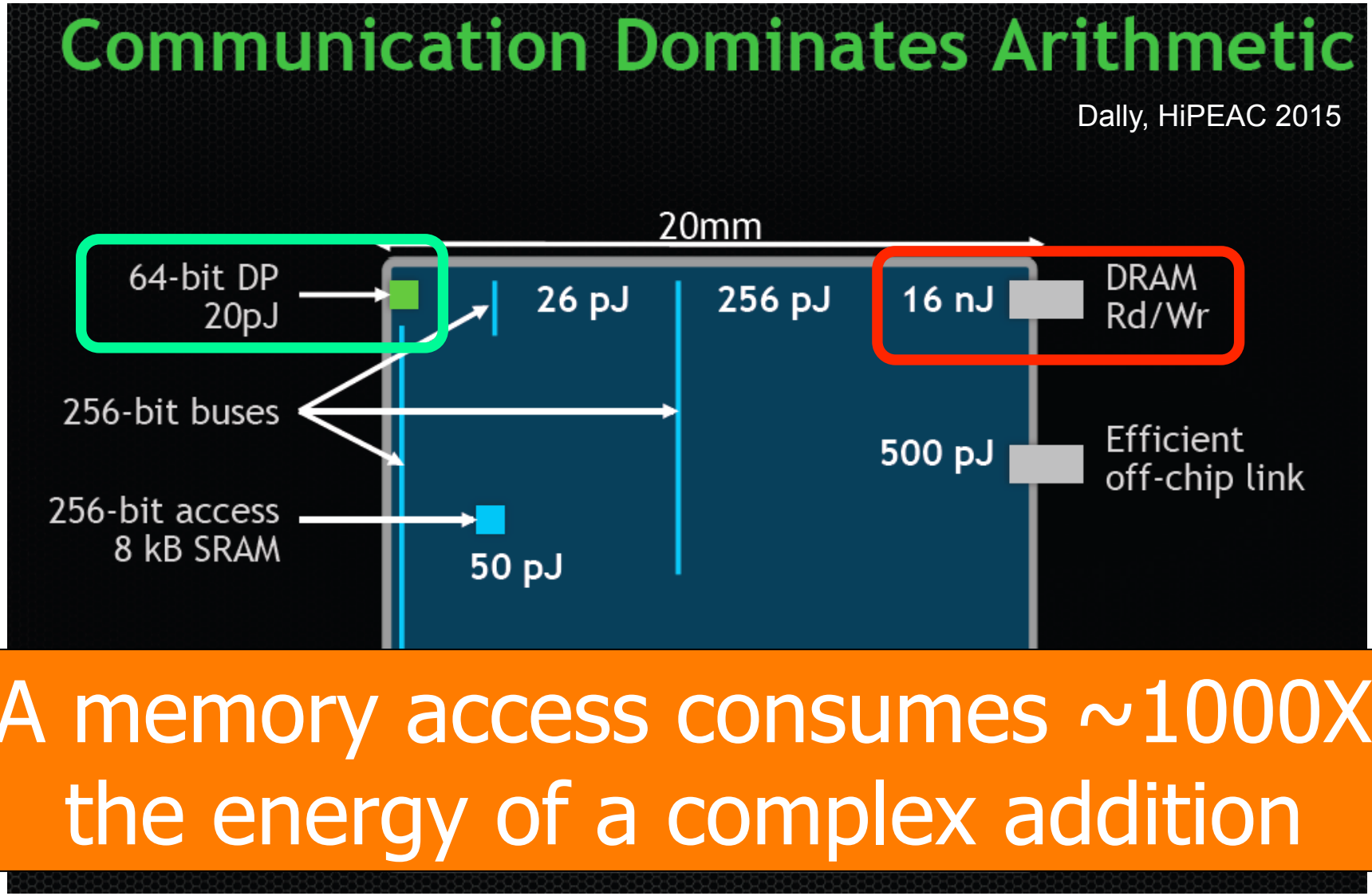
3. Data movement energy dominates compute

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

Data Movement vs. Computation Energy



Data Movement vs. Computation Energy

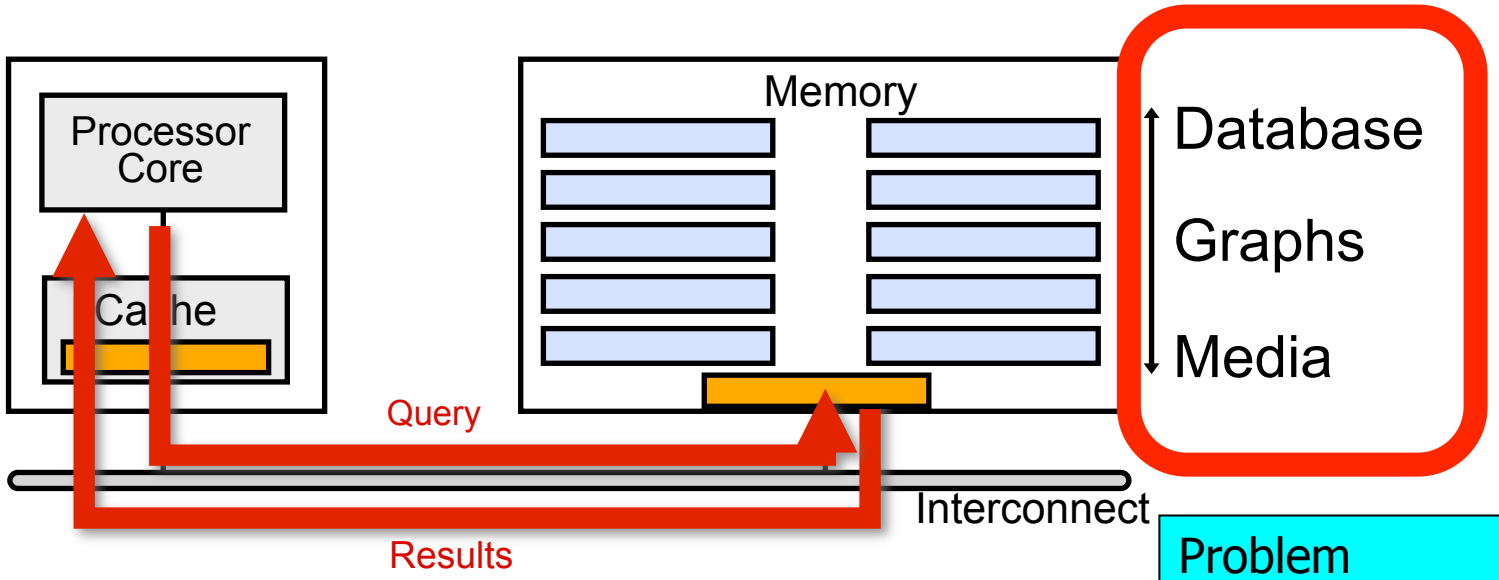


A memory access consumes ~1000X the energy of a complex addition

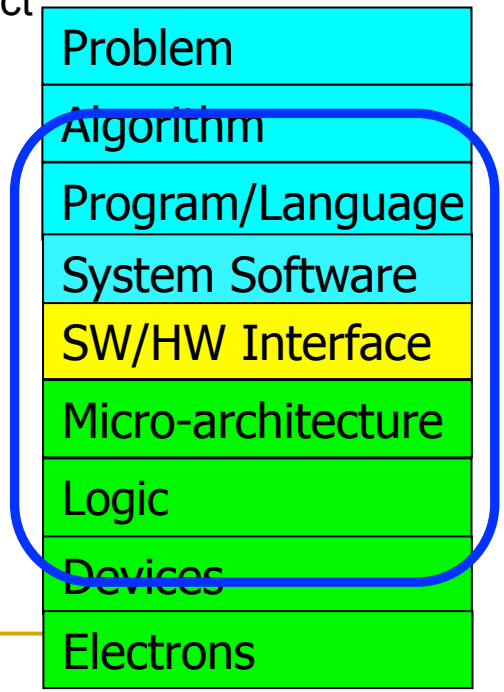
We Need A Paradigm Shift To …

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

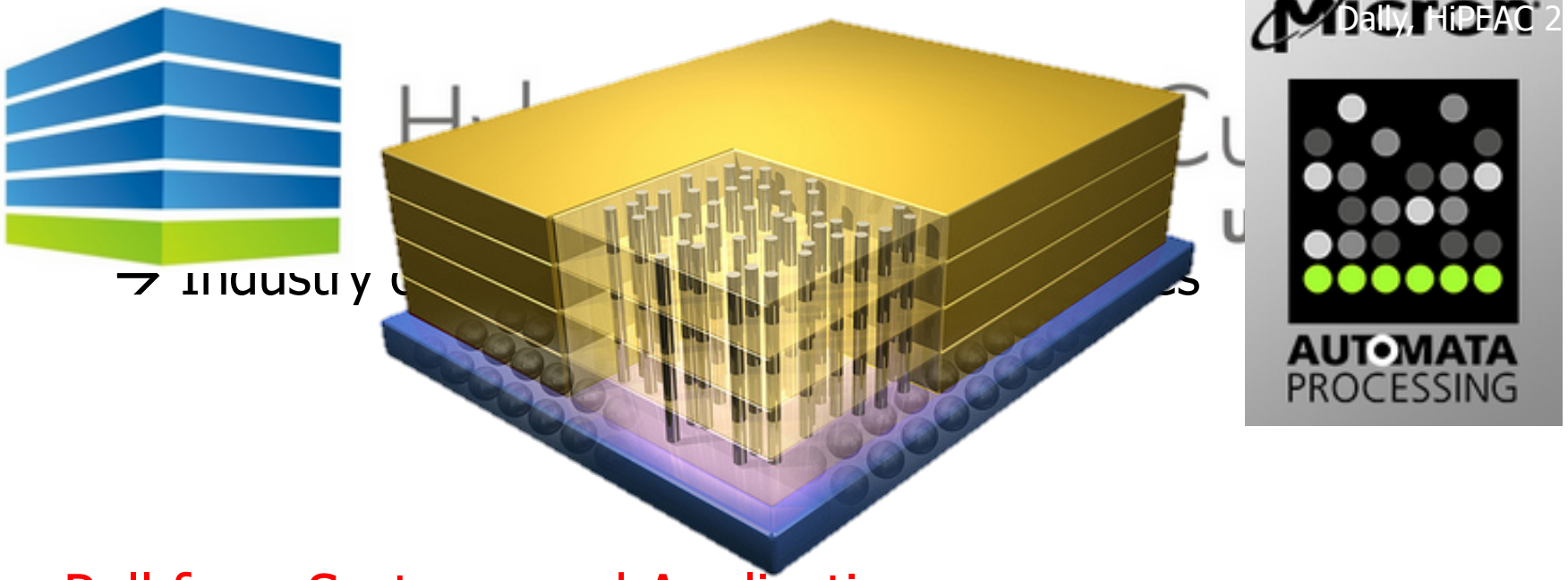
Goal: Processing Inside Memory



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



Why In-Memory Computation Today?



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

Two Approaches to In-Memory Processing

- 1. Minimally change DRAM to enable simple yet powerful computation primitives
  - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
  - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
  - Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses (Seshadri et al., MICRO 2015)

- 2. Exploit the control logic in 3D-stacked memory to enable more comprehensive computation near memory
  - PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture (Ahn et al., ISCA 2015)
  - A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing (Ahn et al., ISCA 2015)
  - Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation (Hsieh et al., ICCD 2016)

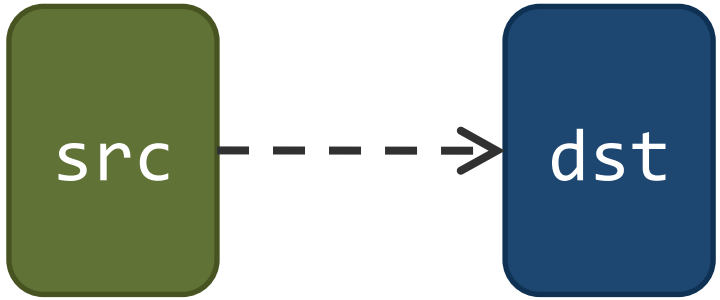
Approach 1: Minimally Changing DRAM

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

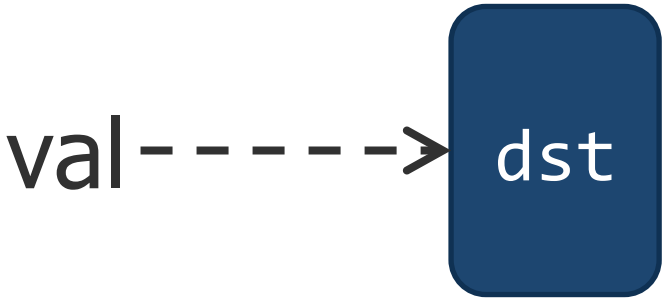
- Examples: RowClone, In-DRAM AND/OR, Gather/Scatter DRAM
  - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
  - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
  - Gather-Scatter DRAM: In-DRAM Address Translation to Improve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

Starting Simple: Data Copy and Initialization

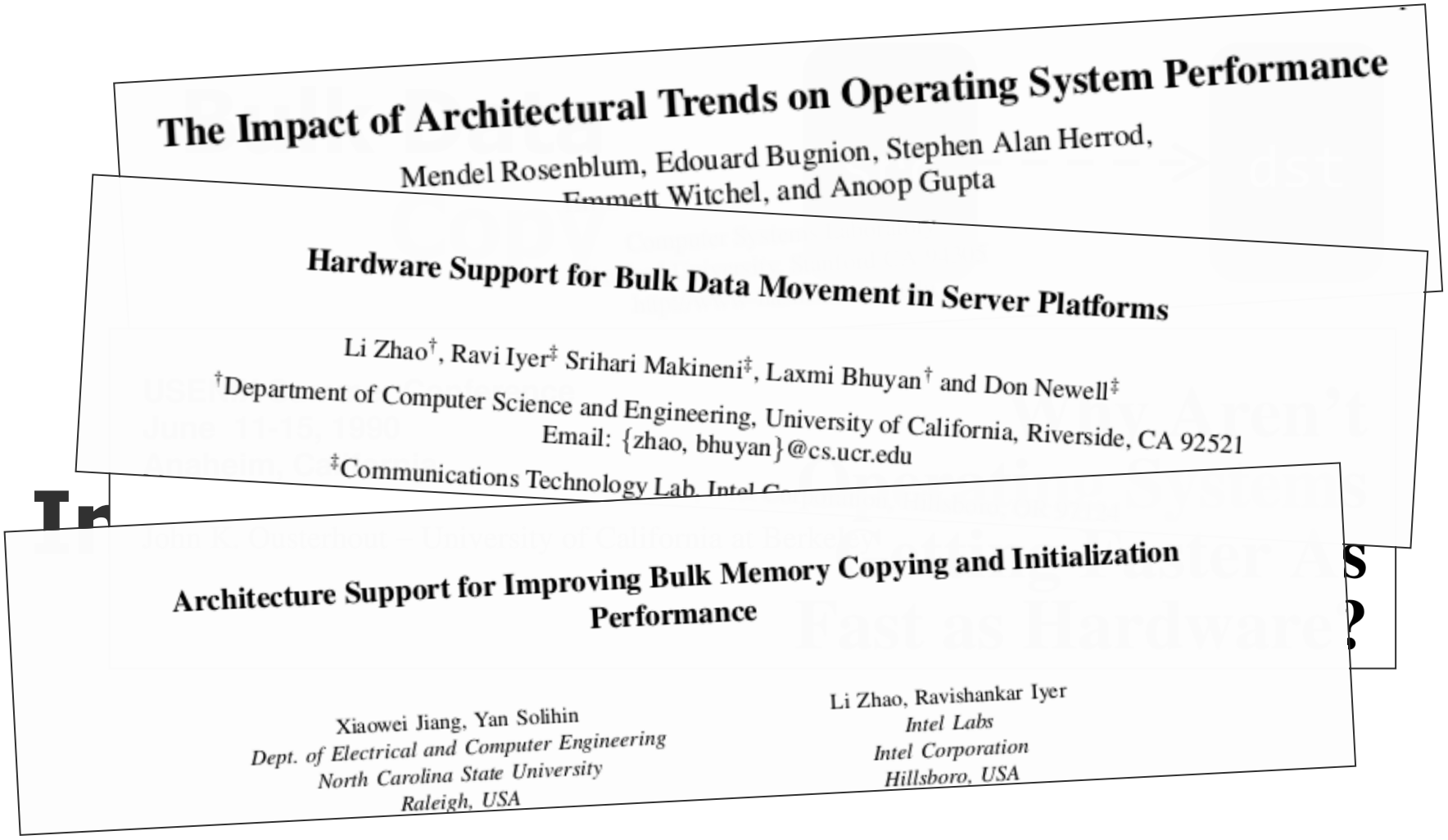
Bulk Data Copy



Bulk Data Initialization



Bulk Data Copy and Initialization



The Impact of Architectural Trends on Operating System Performance

Mendel Rosenblum, Edouard Bugnion, Stephen Alan Herrod,
Emmett Witchel, and Anoop Gupta

Hardware Support for Bulk Data Movement in Server Platforms

Li Zhao[†], Ravi Iyer[‡] Srihari Makineni[‡], Laxmi Bhuyan[†] and Don Newell[‡]

[†]Department of Computer Science and Engineering, University of California, Riverside, CA 92521
Email: {zhao, bhuyan}@cs.ucr.edu

[‡]Communications Technology Lab, Intel C

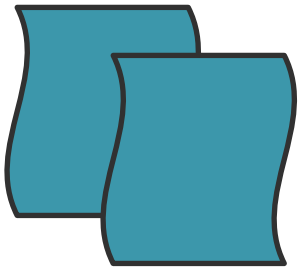
Architecture Support for Improving Bulk Memory Copying and Initialization Performance

Xiaowei Jiang, Yan Solihin
Dept. of Electrical and Computer Engineering
North Carolina State University
Raleigh, USA

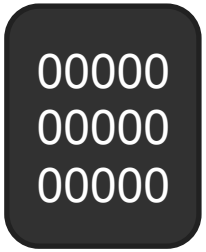
Li Zhao, Ravishankar Iyer
Intel Labs
Intel Corporation
Hillsboro, USA

Bulk Data Copy and Initialization

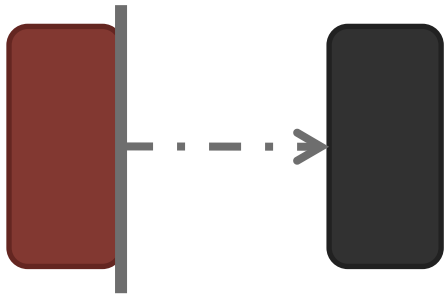
memmove & memcpy: 5% cycles in Google's datacenter [Kanev+ ISCA'15]



Forking

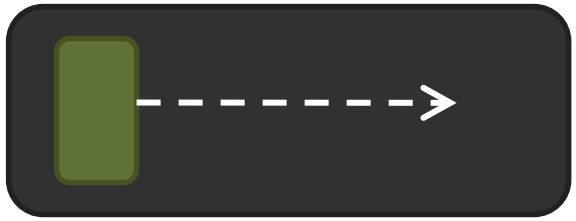


Zero initialization (e.g., security)



Checkpointing

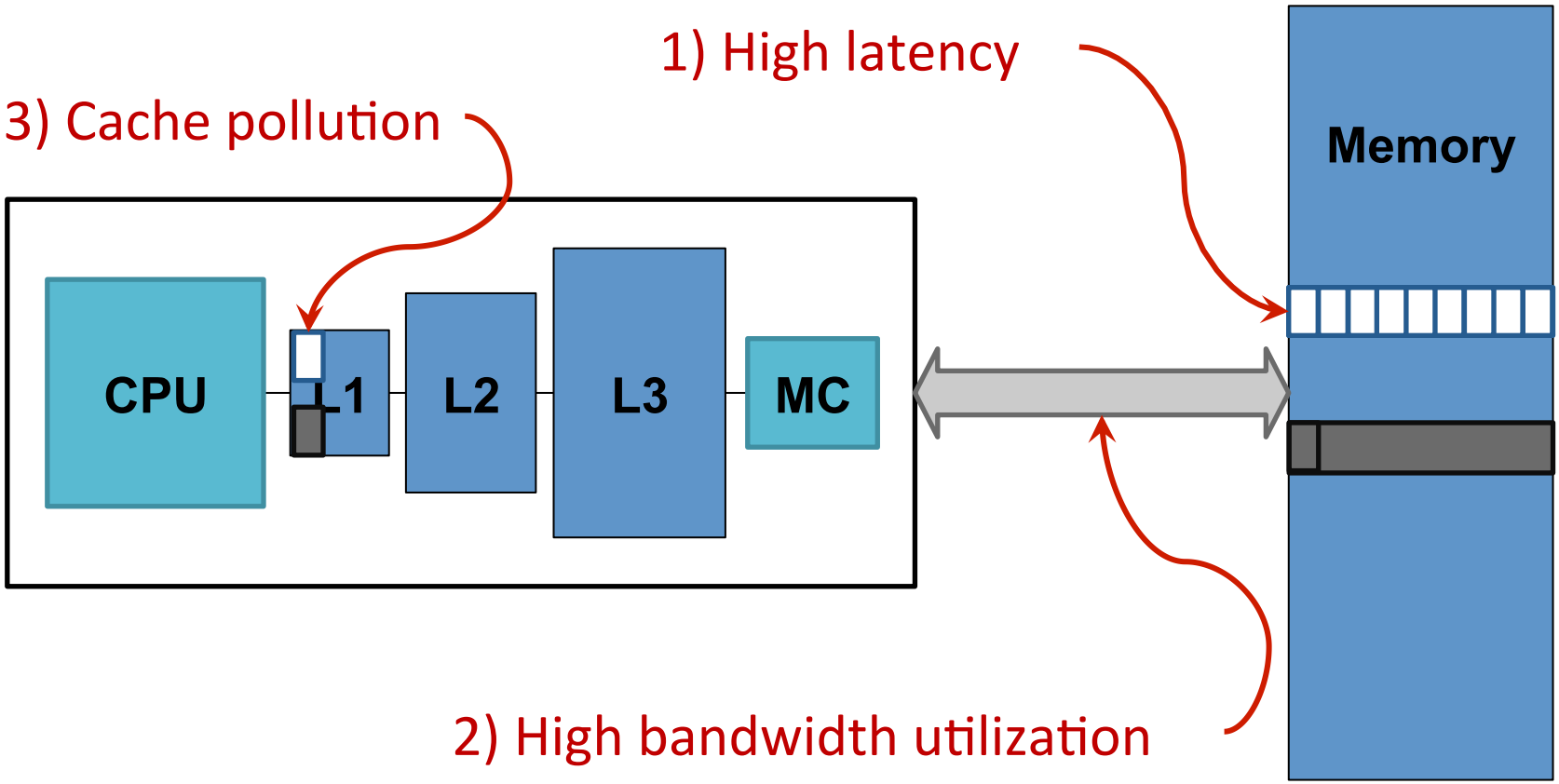
VM Cloning
Deduplication



Page Migration

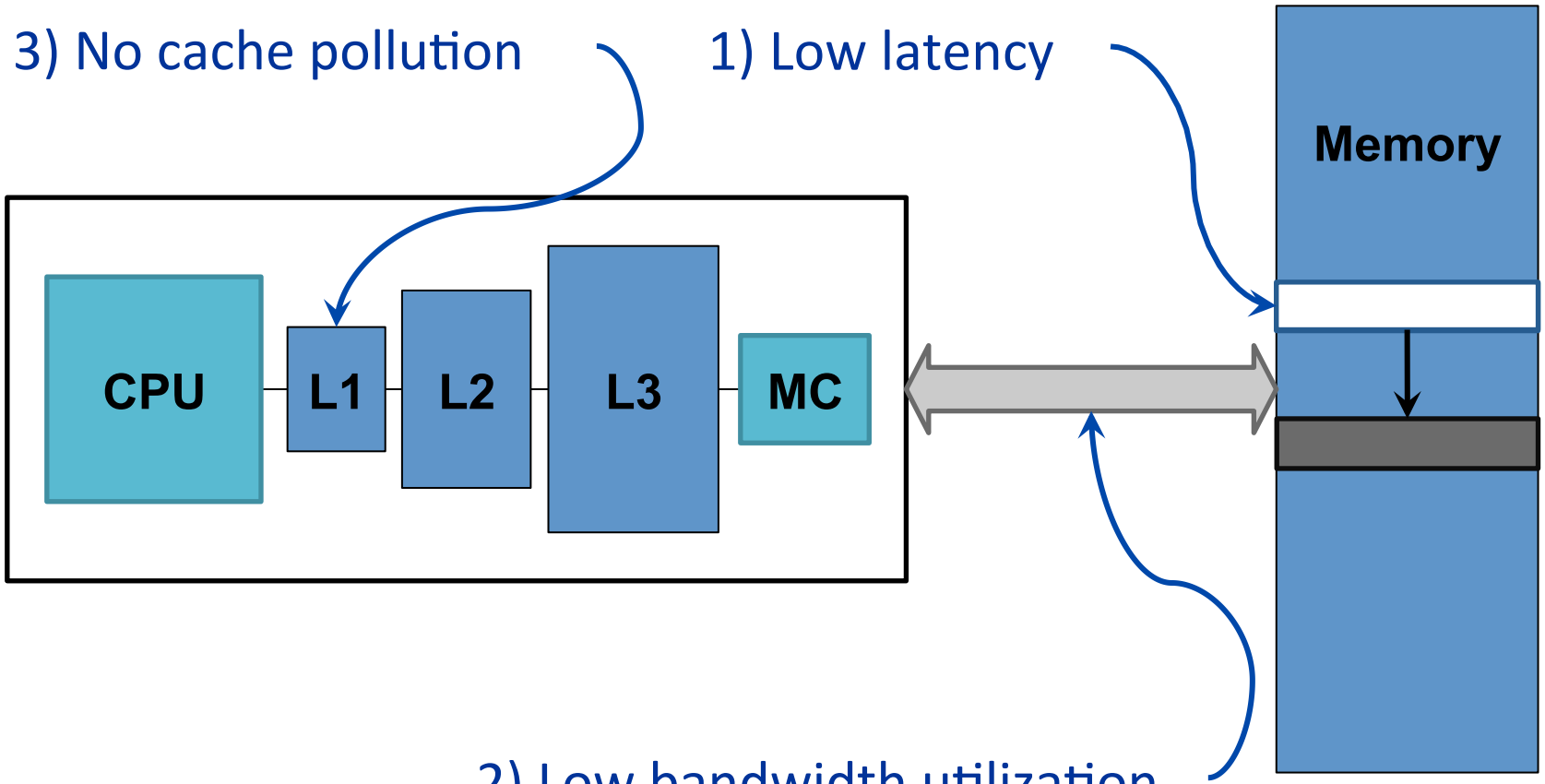
• • •
Many more

Today's Systems: Bulk Data Copy



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

Future Systems: In-Memory Copy

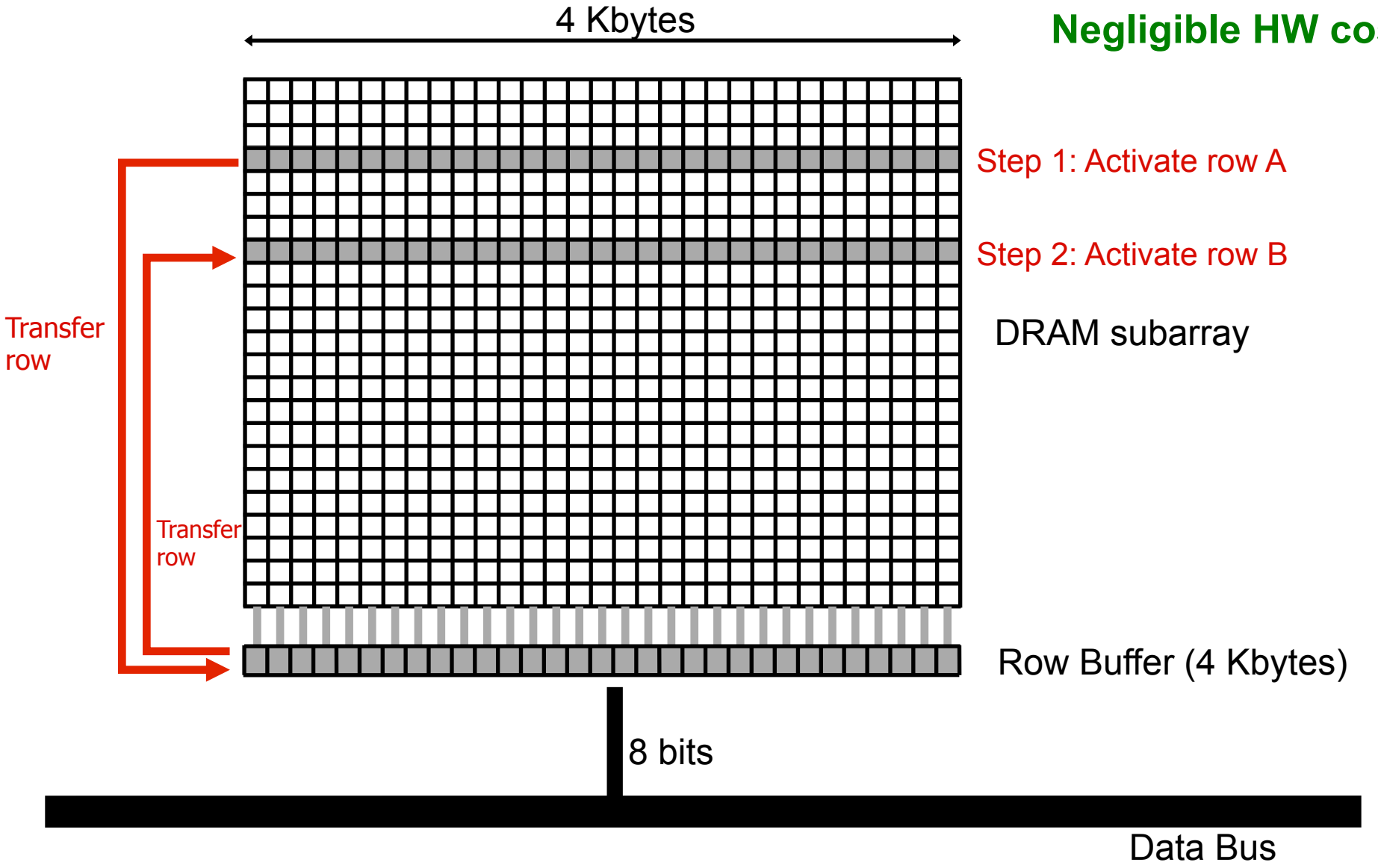


1046ns, 3.6uJ → 90ns, 0.04uJ

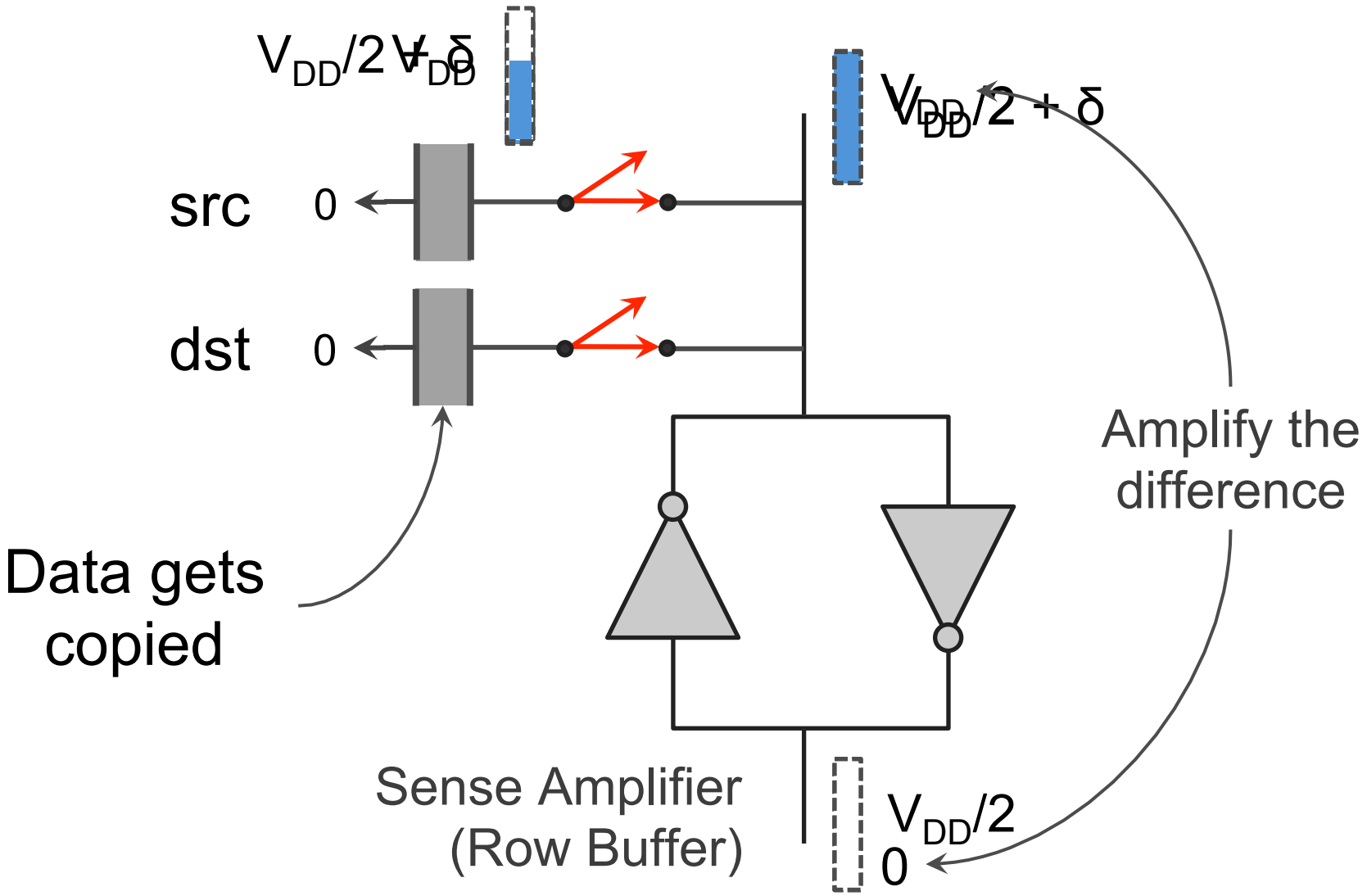
RowClone: In-DRAM Row Copy

Idea: Two consecutive ACTivates

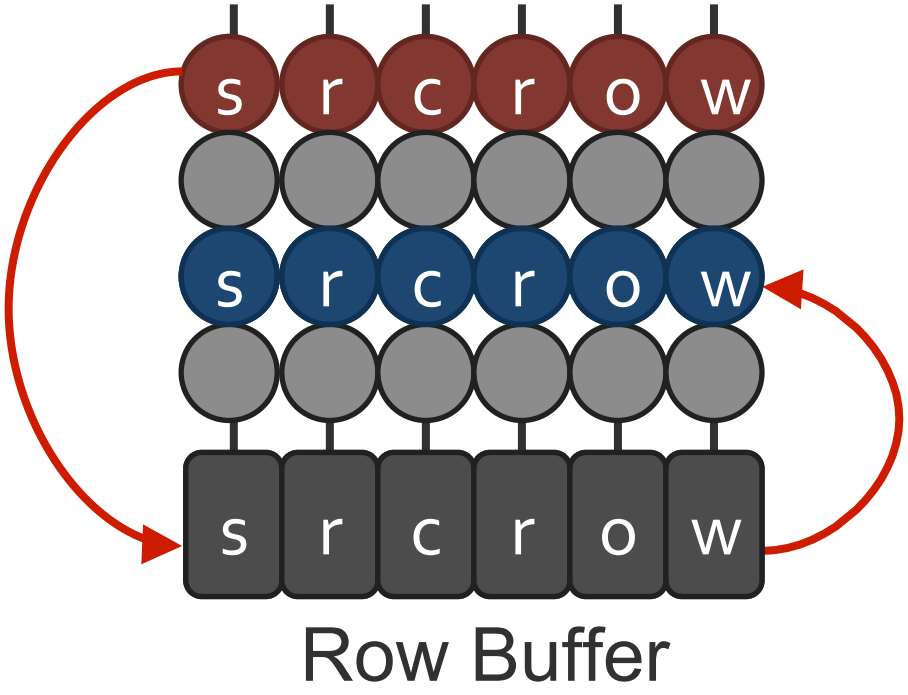
Negligible HW cost



RowClone: Intra-Subarray



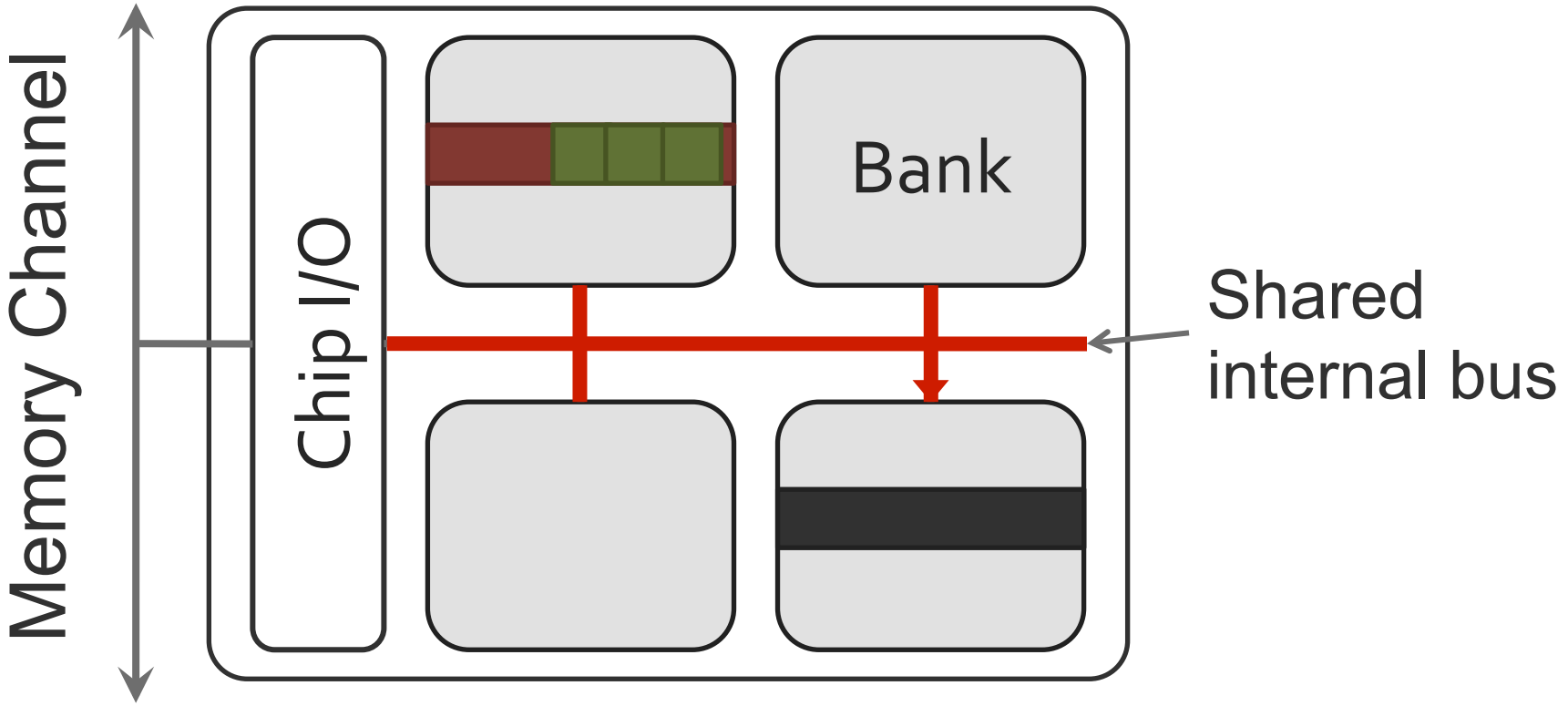
RowClone: Intra-Subarray (II)



1. **Activate** src row (copy data from src to row buffer)

2. **Activate** dst row (disconnect src from row buffer, connect dst – copy data from row buffer to dst)

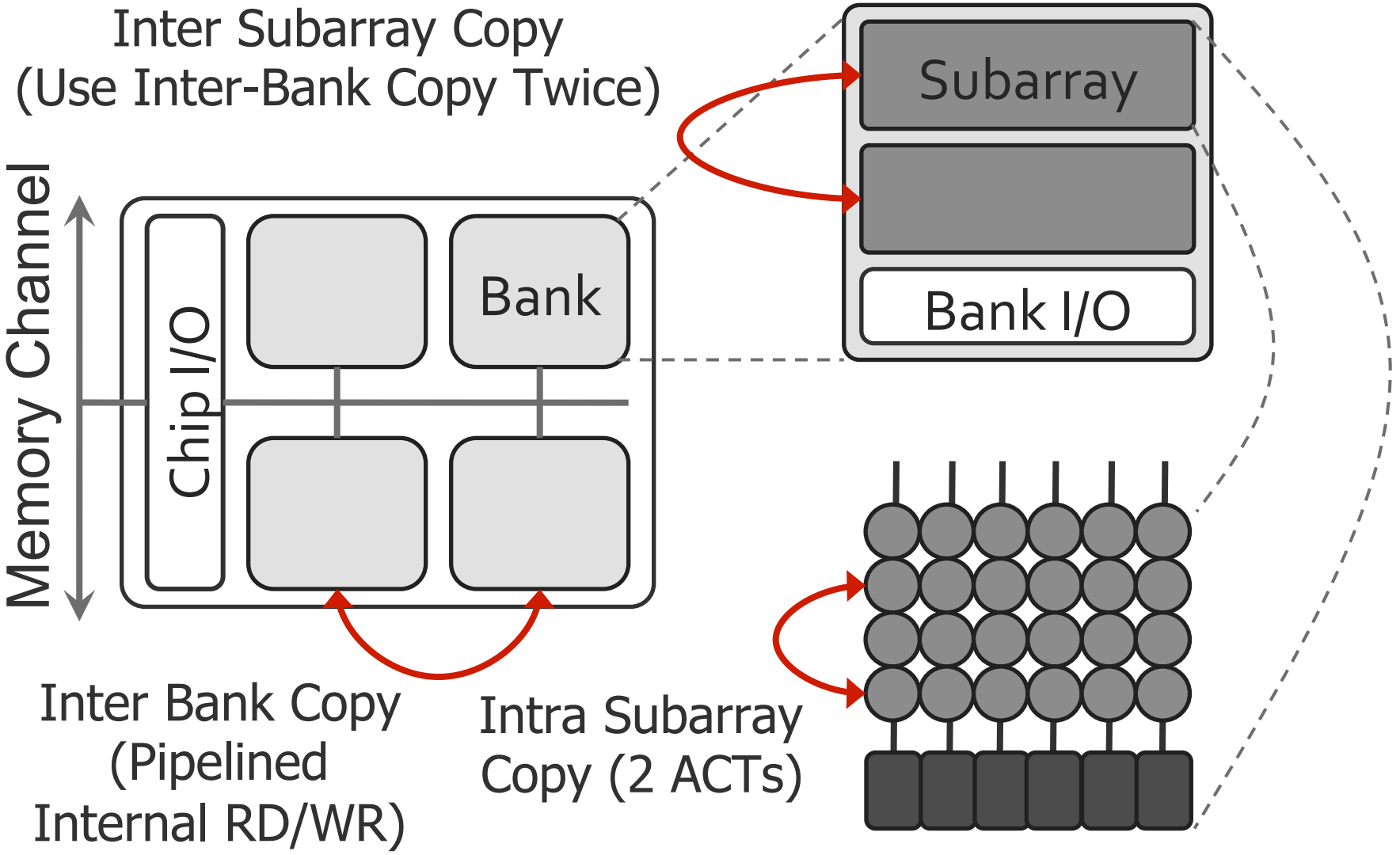
RowClone: Inter-Bank



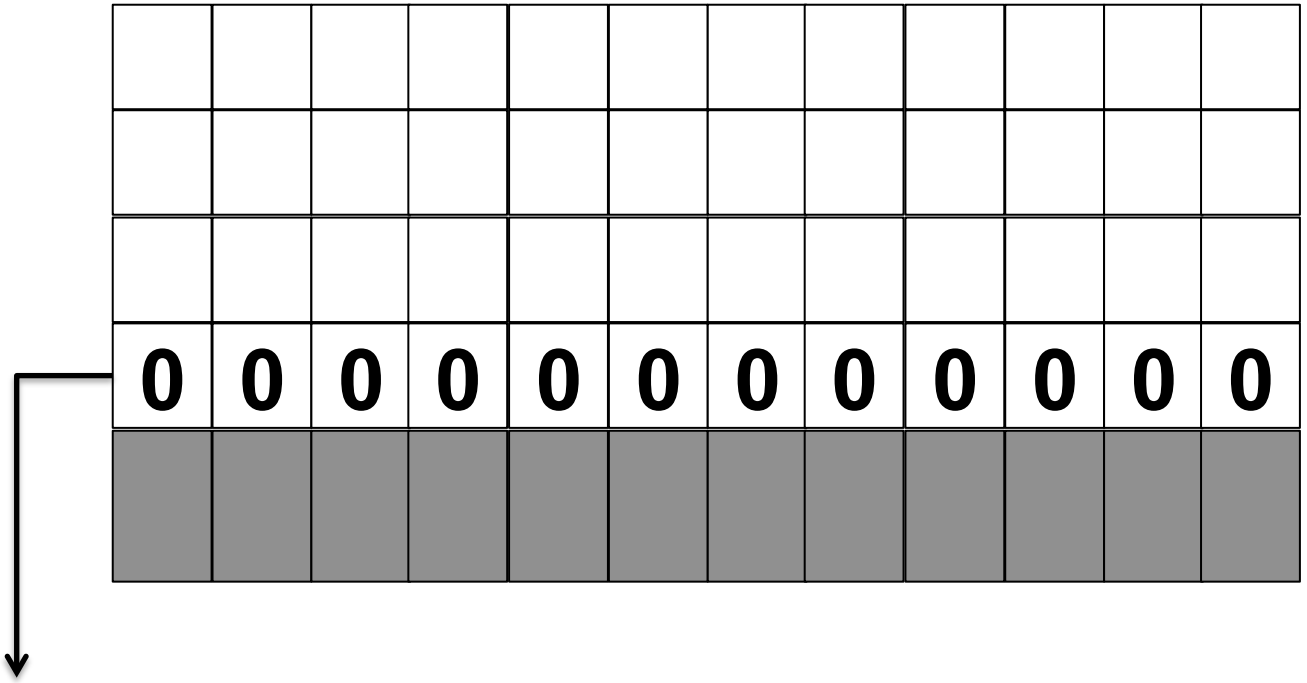
Overlap the latency of the read and the write
1.9X latency reduction, 3.2X energy reduction

Generalized RowClone

0.01% area cost



RowClone: Fast Row Initialization



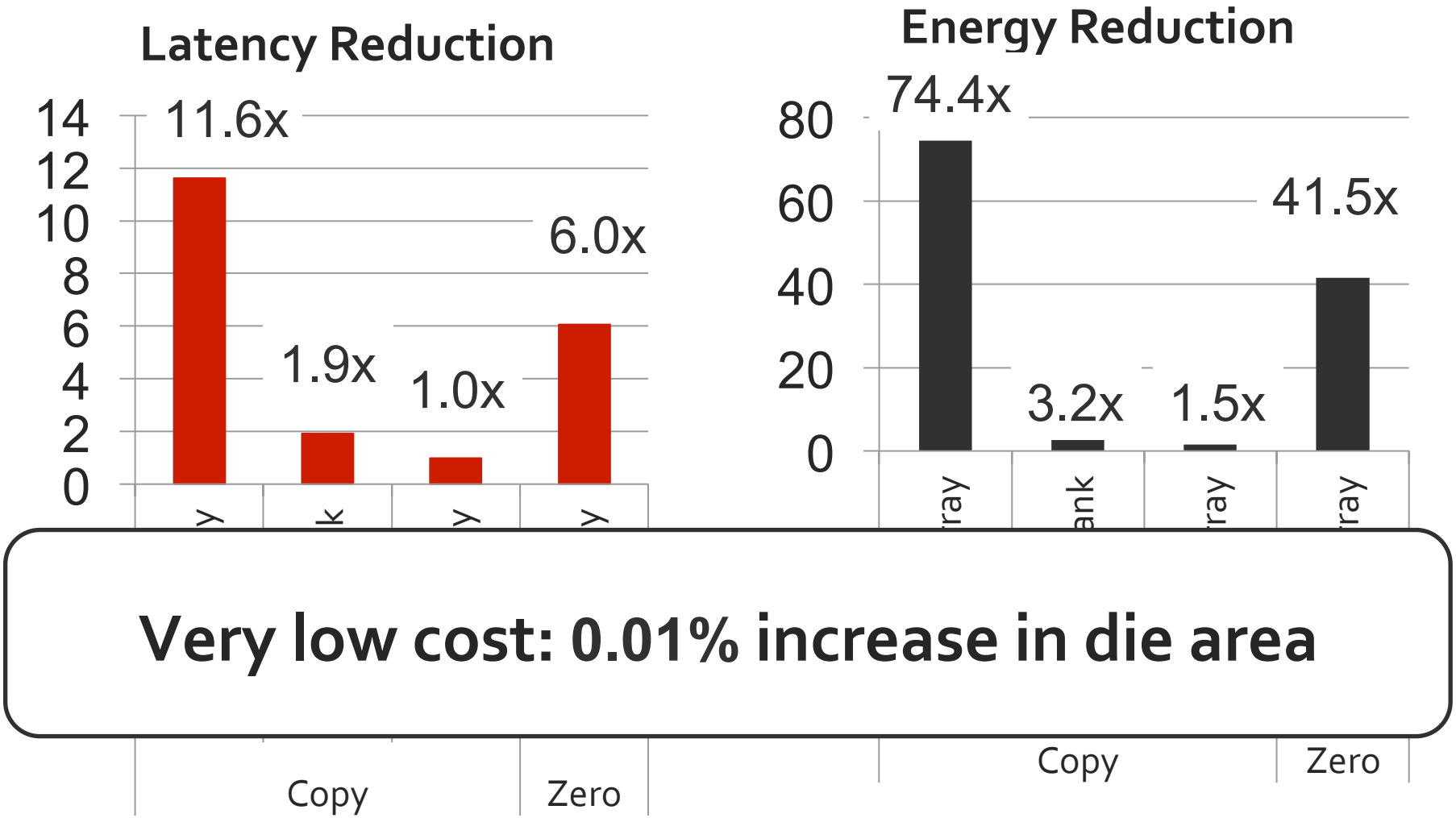
Fix a row at Zero
(0.5% loss in capacity)

RowClone: Bulk Initialization

- Initialization with arbitrary data
  - Initialize one row
  - Copy the data to other rows

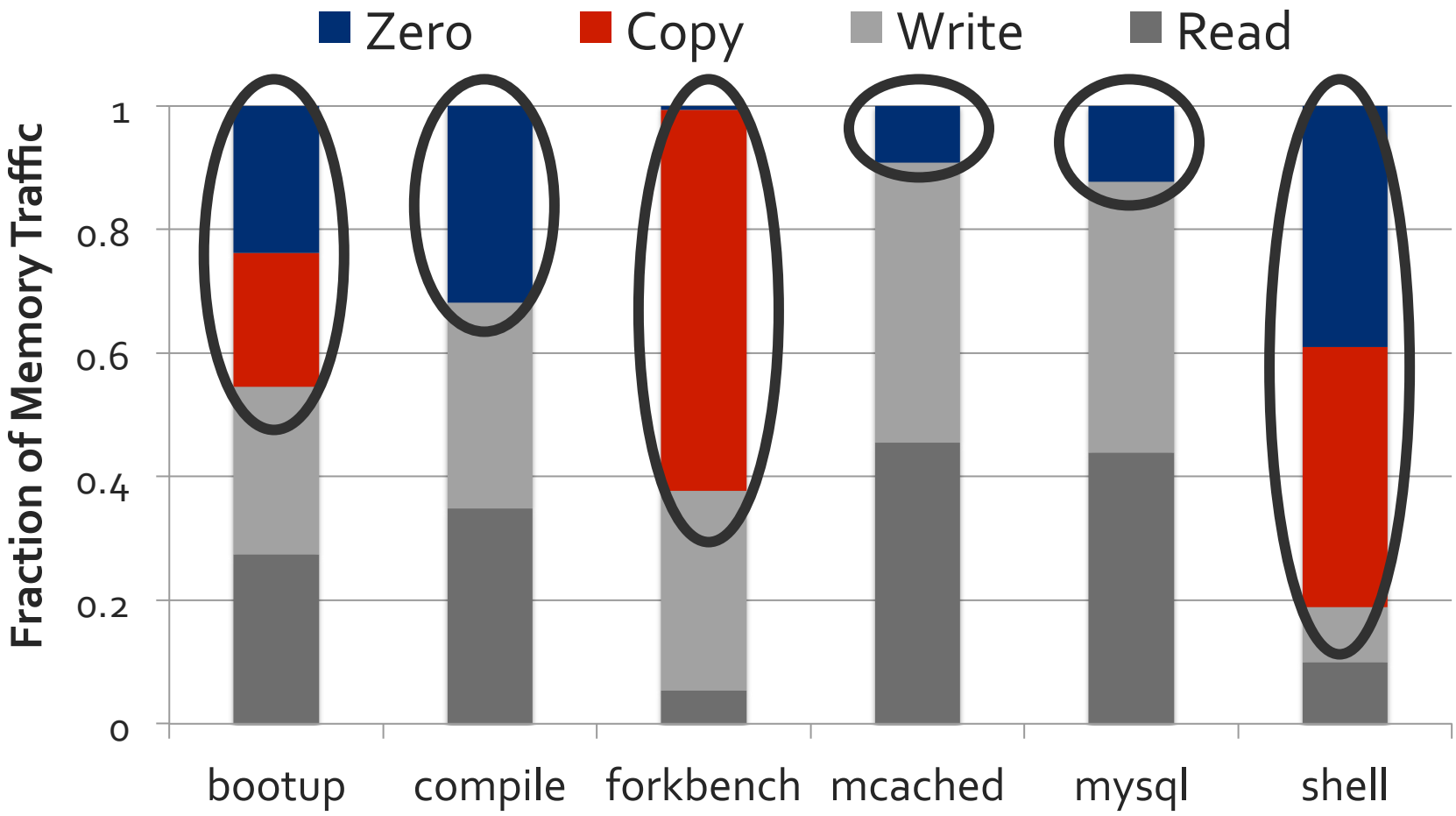
- Zero initialization (most common)
  - Reserve a row in each subarray (always zero)
  - Copy data from reserved row (FPM mode)
  - **6.0X** lower latency, **41.5X** lower DRAM energy
  - 0.2% loss in capacity

RowClone: Latency & Energy Benefits

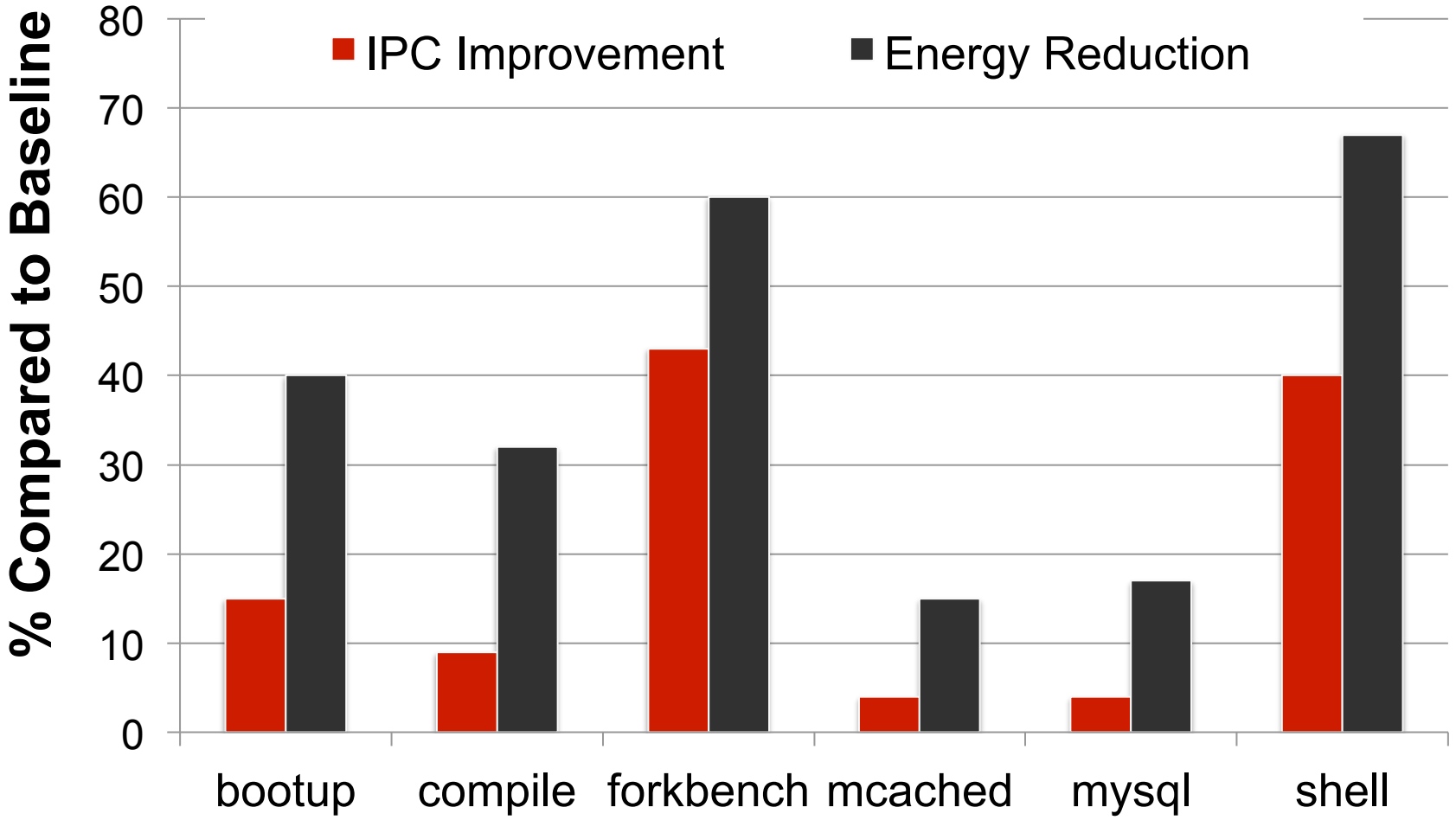


Very low cost: 0.01% increase in die area

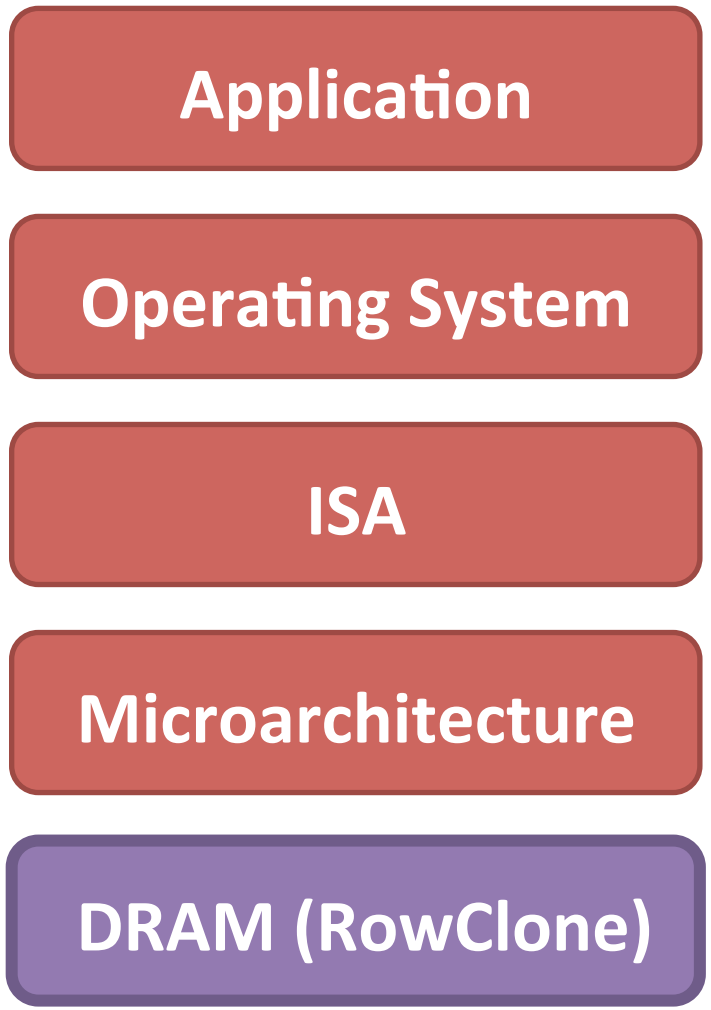
Copy and Initialization in Workloads



RowClone: Application Performance



End-to-End System Design



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

How to ensure cache coherence?

How to maximize latency and energy savings?

How to handle data reuse?

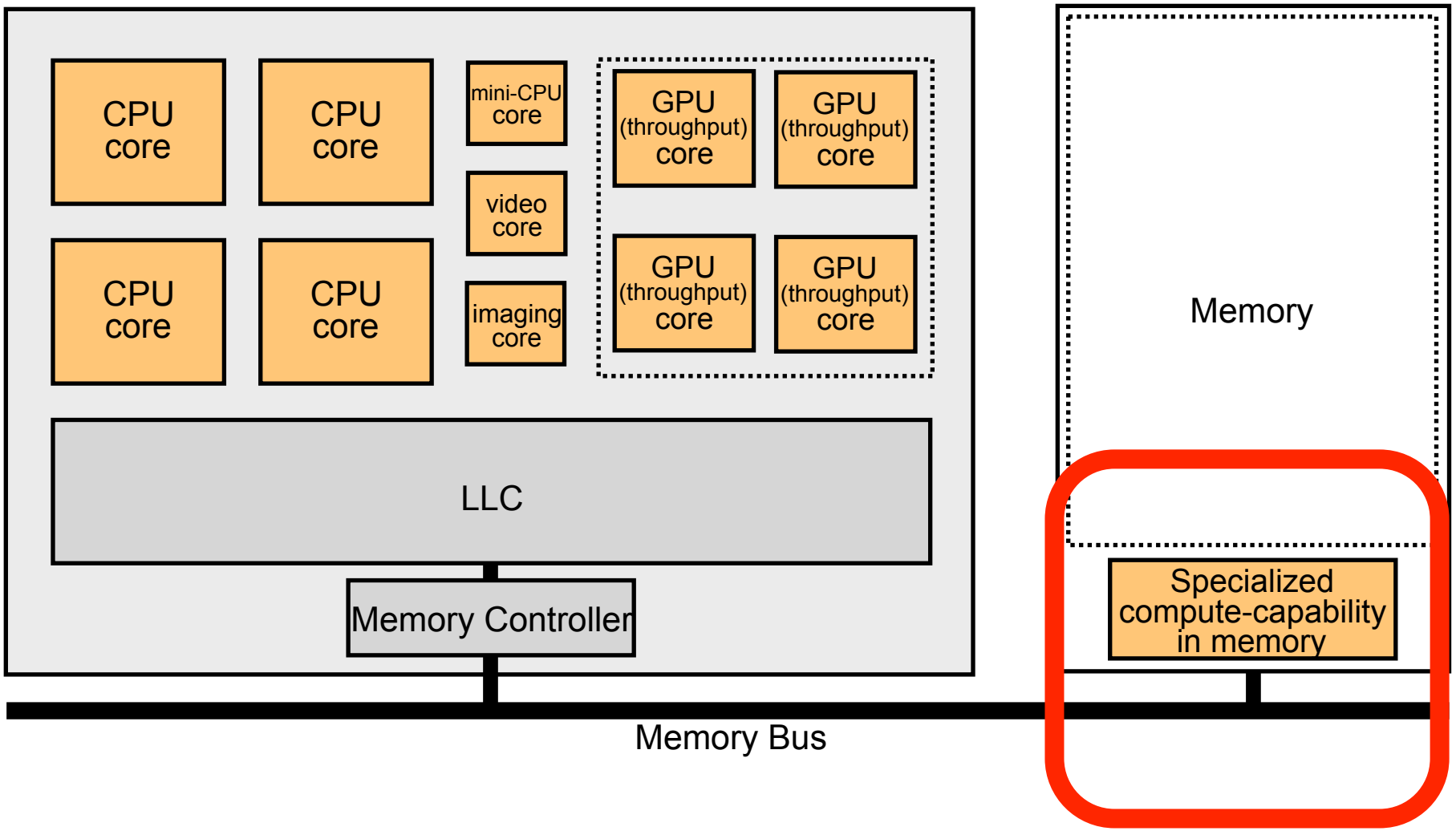
More on RowClone

- Vivek Seshadri, Yoongu Kim, Chris Fallin, Donghyuk Lee, Rachata Ausavarungnirun, Gennady Pekhimenko, Yixin Luo, Onur Mutlu, Michael A. Kozuch, Phillip B. Gibbons, and Todd C. Mowry,
  "RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization"
  *Proceedings of the 46th International Symposium on Microarchitecture (**MICRO**)*, Davis, CA, December 2013. [Slides (pptx) (pdf)] [Lightning Session Slides (pptx) (pdf)] [Poster (pptx) (pdf)]

RowClone: Fast and Energy-Efficient In-DRAM Bulk Data Copy and Initialization

Vivek Seshadri vseshadr@cs.cmu.edu
Yoongu Kim yoongukim@cmu.edu
Chris Fallin* cfallin@c1f.net
Donghyuk Lee donghyuk1@cmu.edu
Rachata Ausavarungnirun rachata@cmu.edu
Gennady Pekhimenko gpekhime@cs.cmu.edu
Yixin Luo yixinluo@andrew.cmu.edu
Onur Mutlu onur@cmu.edu
Phillip B. Gibbons† phillip.b.gibbons@intel.com
Michael A. Kozuch† michael.a.kozuch@intel.com
Todd C. Mowry tcm@cs.cmu.edu
Carnegie Mellon University †Intel Pittsburgh

Memory as an Accelerator



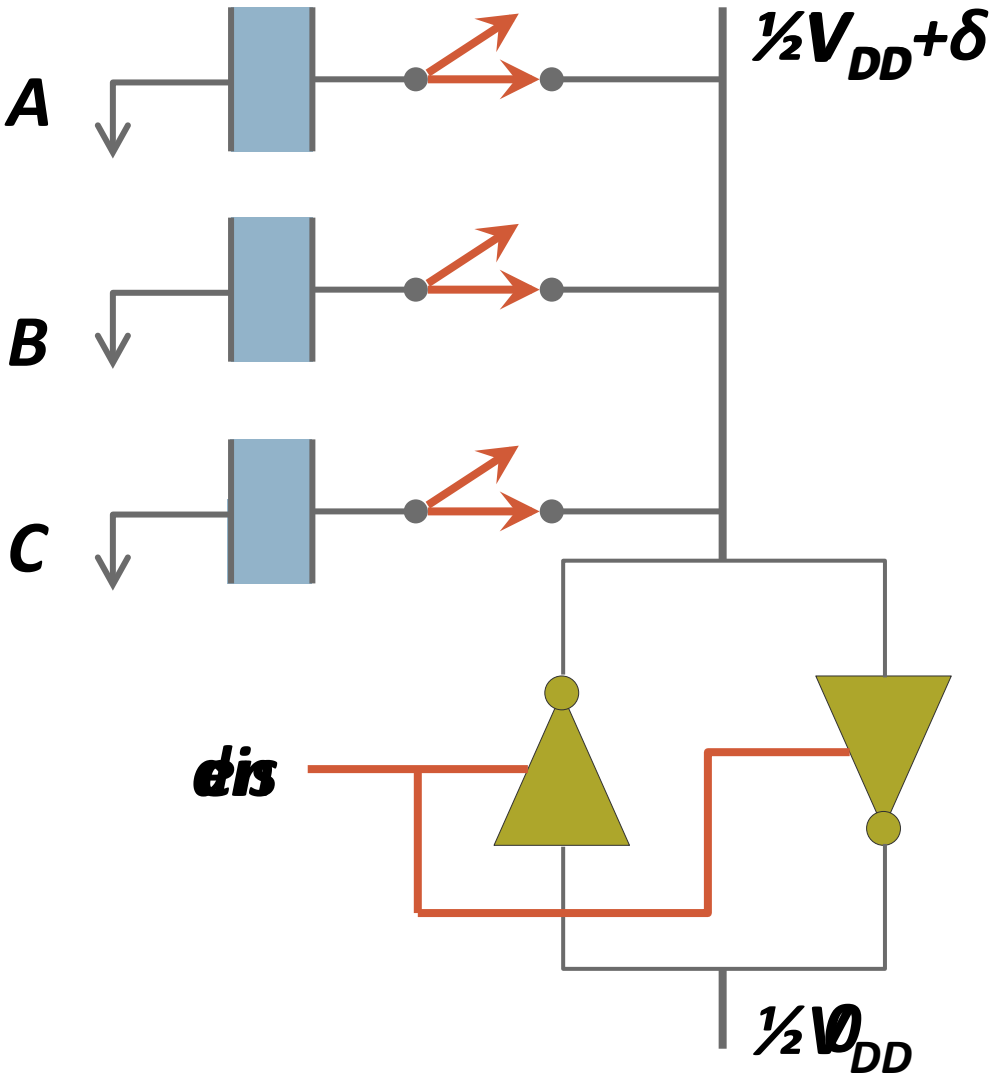
Memory similar to a "conventional" accelerator

In-Memory Bulk Bitwise Operations

- We can support in-DRAM COPY, ZERO, AND, OR, NOT, MAJ
- At low cost
- Using analog computation capability of DRAM
  - Idea: activating multiple rows performs computation
- 30-60X performance and energy improvement
  - Seshadri+, "Ambit: In-Memory Accelerator for Bulk Bitwise Operations Using Commodity DRAM Technology," MICRO 2017.

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

In-DRAM AND/OR: Triple Row Activation



Final State
AB + BC + AC

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

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

In-DRAM Bulk Bitwise AND/OR Operation

- BULKAND A, B → C
- Semantics: Perform a bitwise AND of two rows A and B and store the result in row C

- R0 – reserved zero row, R1 – reserved one row
- D1, D2, D3 – Designated rows for triple activation

1. RowClone A into D1
2. RowClone B into D2
3. RowClone R0 into D3
4. ACTIVATE D1,D2,D3
5. RowClone Result into C

More on In-DRAM Bulk AND/OR

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

Fast Bulk Bitwise AND and OR in DRAM

Vivek Seshadri*, Kevin Hsieh*, Amirali Boroumand*, Donghyuk Lee*,
Michael A. Kozuch†, Onur Mutlu*, Phillip B. Gibbons†, Todd C. Mowry*

*Carnegie Mellon University †Intel Pittsburgh

In-DRAM NOT: Dual Contact Cell

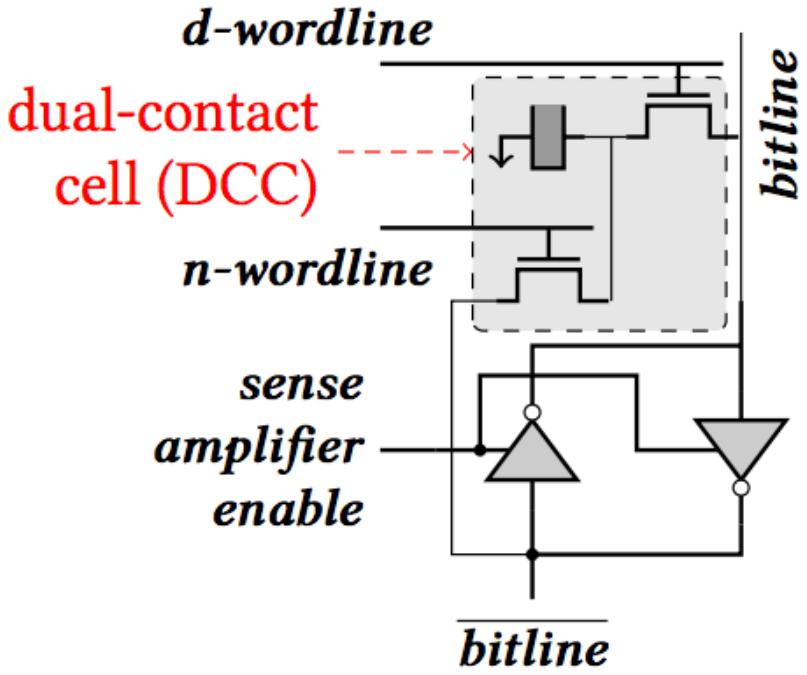


Figure 5: A dual-contact cell connected to both ends of a sense amplifier

Idea:

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

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

In-DRAM NOT Operation

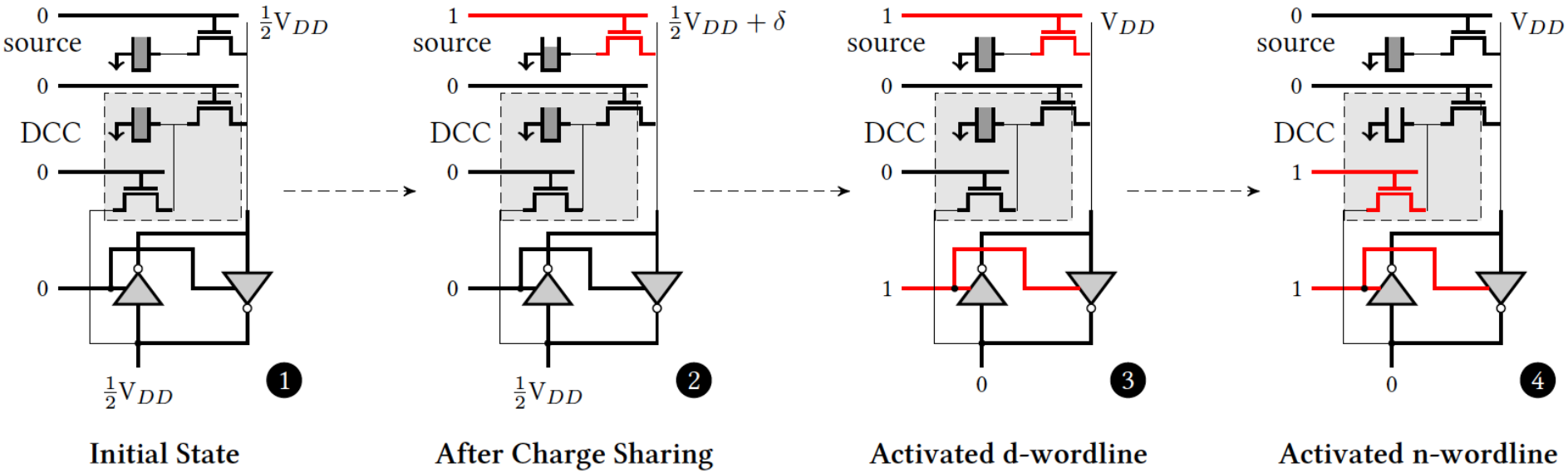


Figure 5: Bitwise NOT using a dual contact capacitor

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

Performance: In-DRAM Bitwise Operations

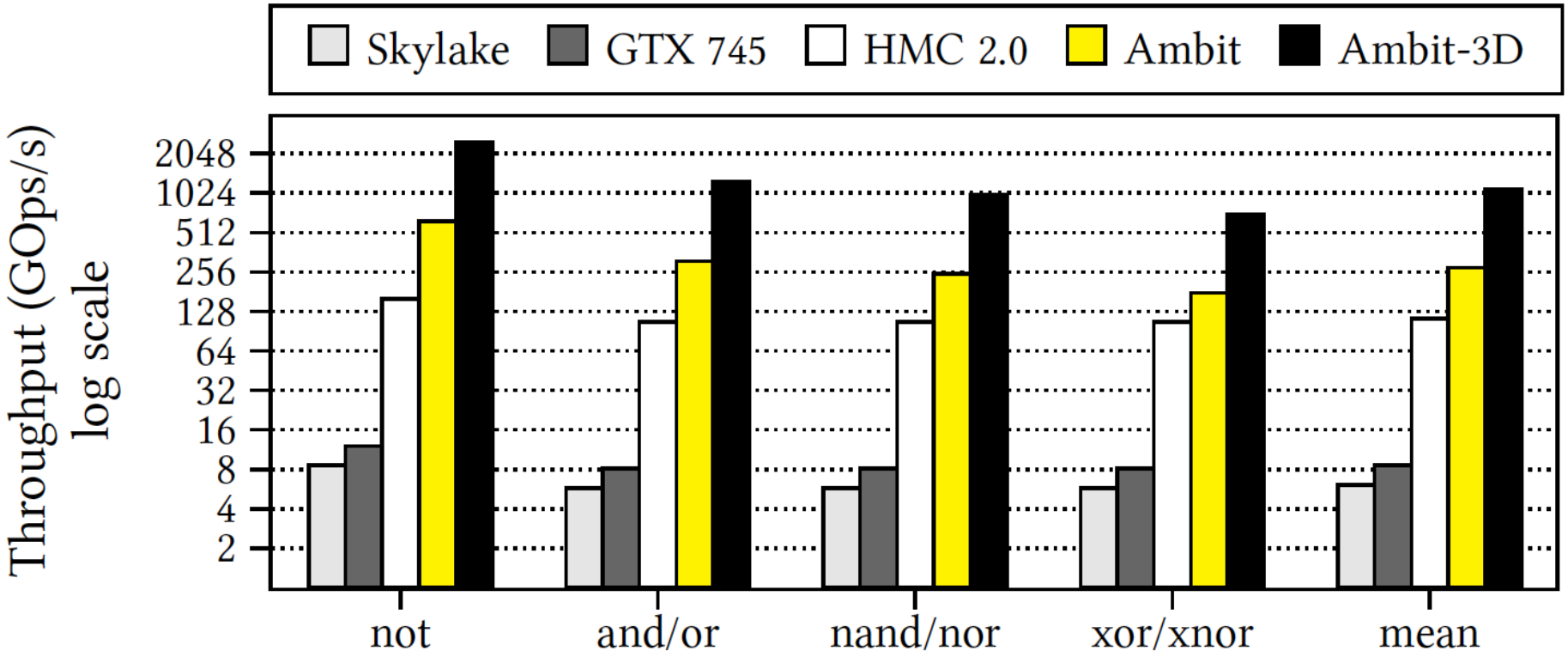


Figure 9: Throughput of bitwise operations on various systems.

Energy of In-DRAM Bitwise Operations

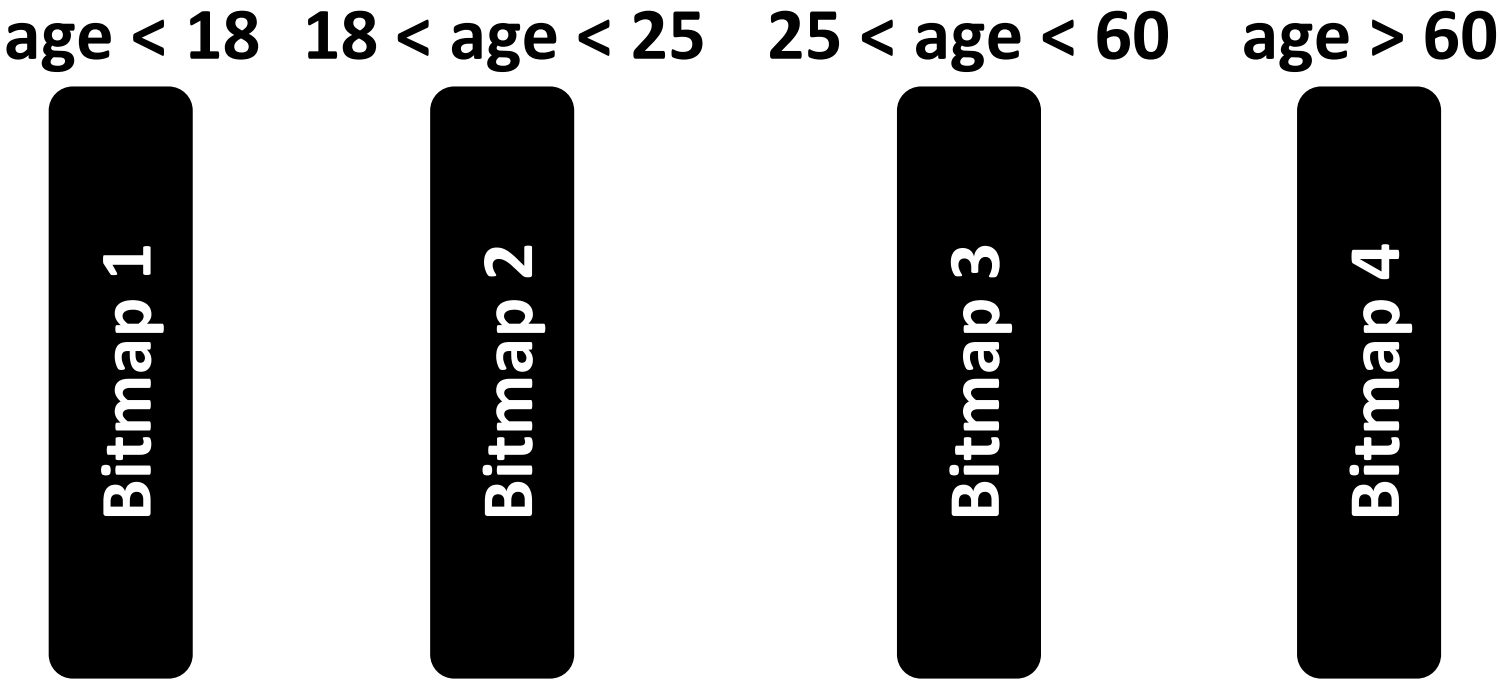
	Design	not	and/or	nand/nor	xor/xnor
DRAM &	**DDR3**	93.7	137.9	137.9	137.9
Channel Energy	**Ambit**	1.6	3.2	4.0	5.5
(nJ/KB)	(↓)	59.5X	43.9X	35.1X	25.1X

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

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

Example Data Structure: Bitmap Index

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



Performance: Bitmap Index on Ambit

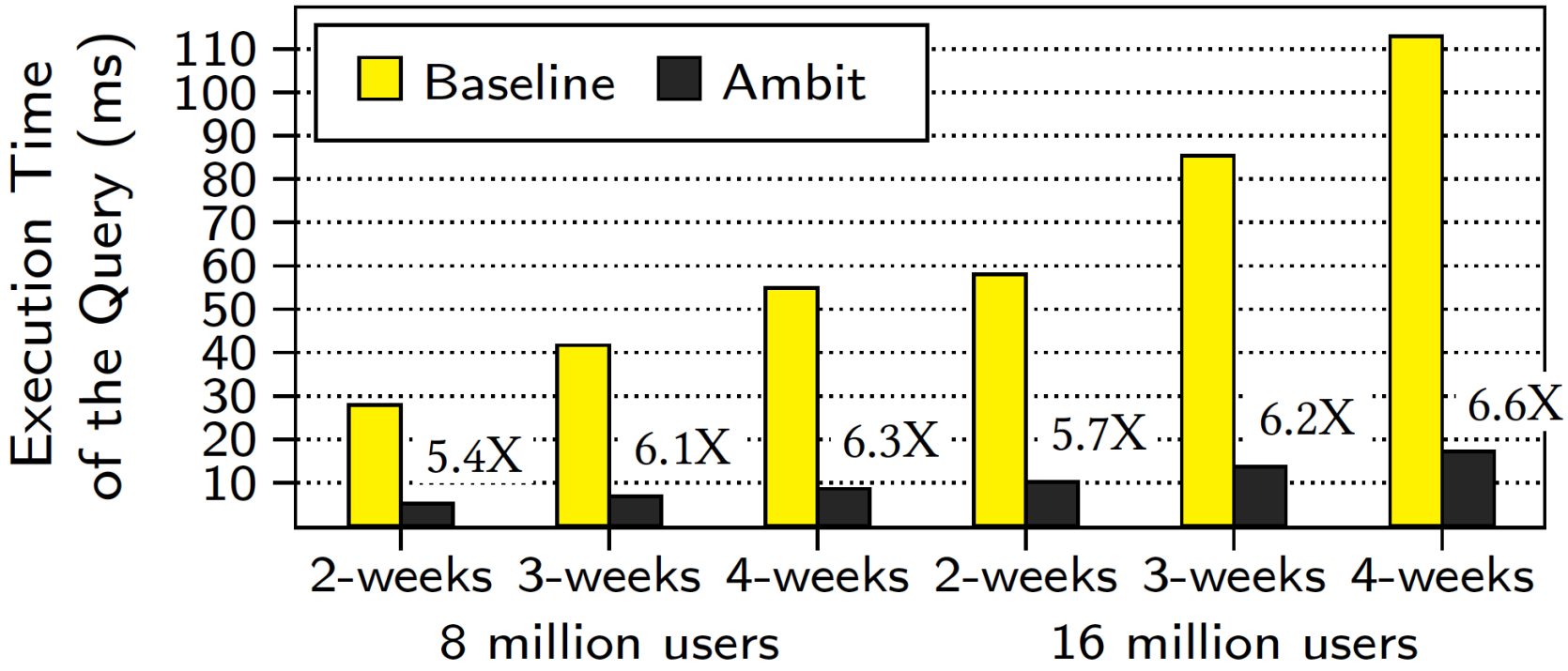


Figure 10: Bitmap index performance. The value above each bar indicates the reduction in execution time due to Ambit.

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

Performance: BitWeaving on Ambit

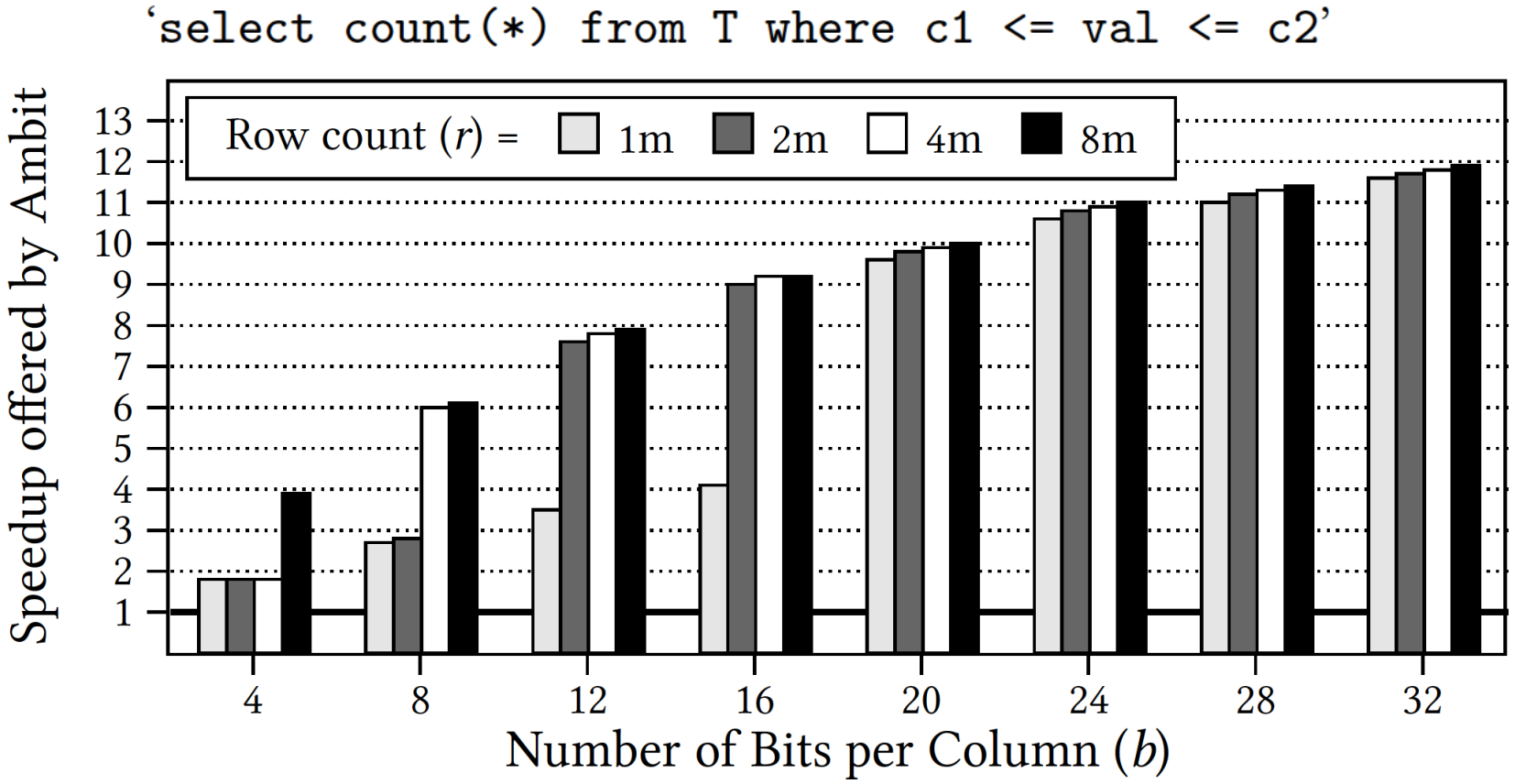


Figure 11: Speedup offered by Ambit over baseline CPU with SIMD for BitWeaving

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

Required Reading: Ambit

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

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

Vivek Seshadri[1,5] Donghyuk Lee[2,5] Thomas Mullins[3,5] Hasan Hassan[4] Amirali Boroumand[5]
Jeremie Kim[4,5] Michael A. Kozuch[3] Onur Mutlu[4,5] Phillip B. Gibbons[5] Todd C. Mowry[5]

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

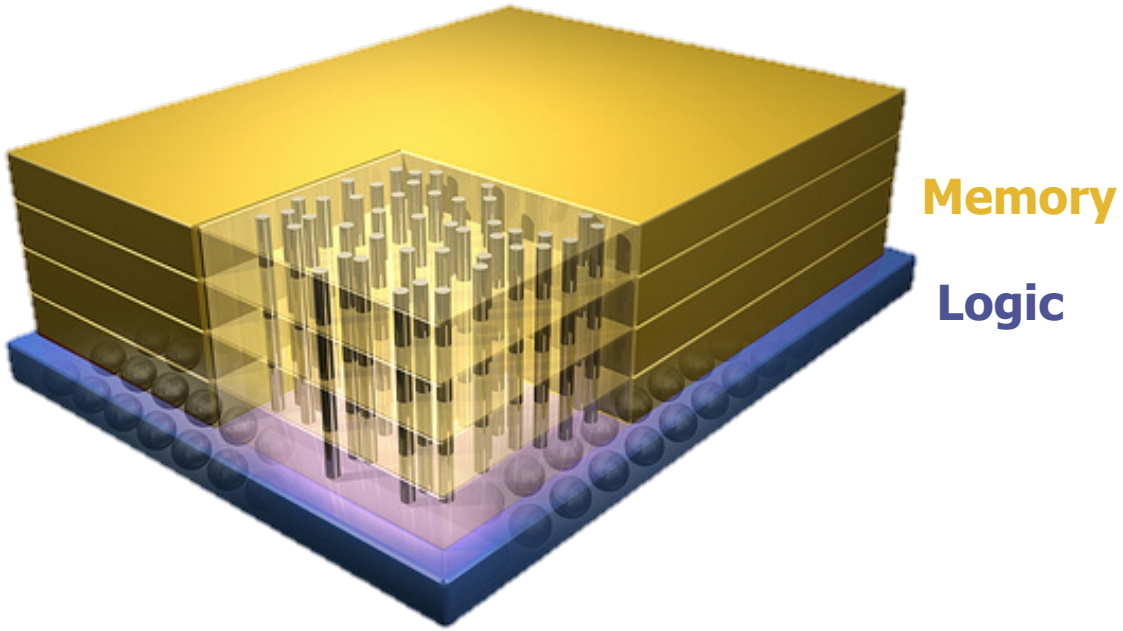
Two Approaches to In-Memory Processing

- 1. Minimally change DRAM to enable simple yet powerful computation primitives
  - RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data (Seshadri et al., MICRO 2013)
  - Fast Bulk Bitwise AND and OR in DRAM (Seshadri et al., IEEE CAL 2015)
  - Gather-Scatter DRAM: In-DRAM Address Translation to Improve the Spatial Locality of Non-unit Strided Accesses (Seshadri et al., MICRO 2015)

- 2. Exploit the control logic in 3D-stacked memory to enable more comprehensive computation near memory
  - PIM-Enabled Instructions: A Low-Overhead, Locality-Aware Processing-in-Memory Architecture (Ahn et al., ISCA 2015)
  - A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing (Ahn et al., ISCA 2015)
  - Accelerating Pointer Chasing in 3D-Stacked Memory: Challenges, Mechanisms, Evaluation (Hsieh et al., ICCD 2016)

Opportunity: 3D-Stacked Logic+Memory

Hybrid Memory Cube
CONSORTIUM



DRAM Landscape (circa 2015)

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

Table 1. Landscape of DRAM-based memory

Kim+, "Ramulator: A Flexible and Extensible DRAM Simulator", IEEE CAL 2015.

Two Key Questions in 3D Stacked PIM

- What is the minimal processing-in-memory support we can provide ?
  - without changing the system significantly
  - while achieving significant benefits of processing in 3D-stacked memory

- How can we accelerate important applications if we use 3D-stacked memory as a coarse-grained accelerator?
  - what is the architecture and programming model?
  - what are the mechanisms for acceleration?

Graph Processing

- Large graphs are everywhere (circa 2015)

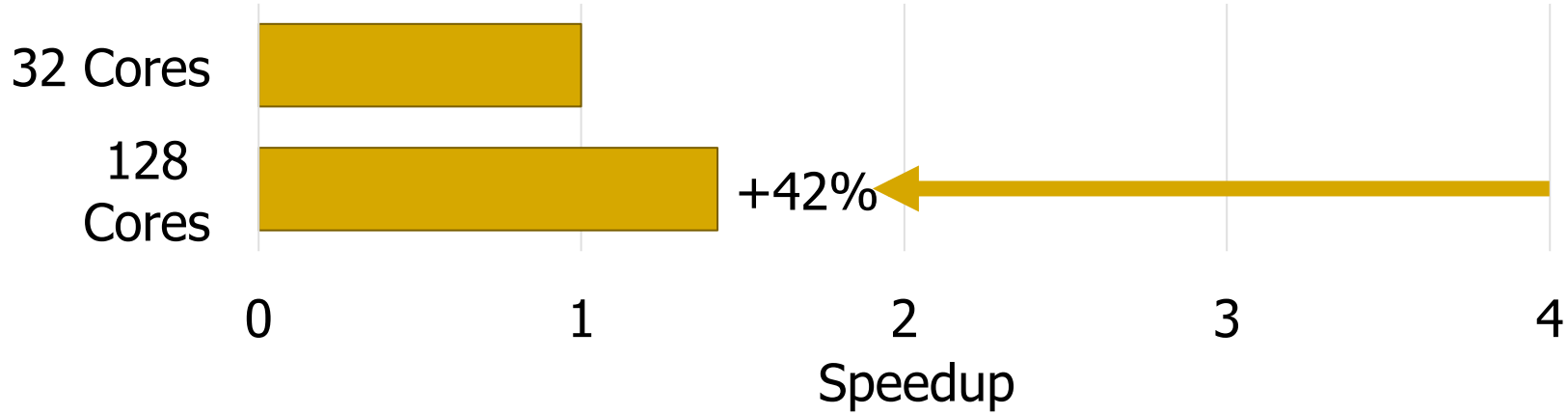
36 Million
Wikipedia Pages

1.4 Billion
Facebook Users

300 Million
Twitter Users

30 Billion
Instagram Photos

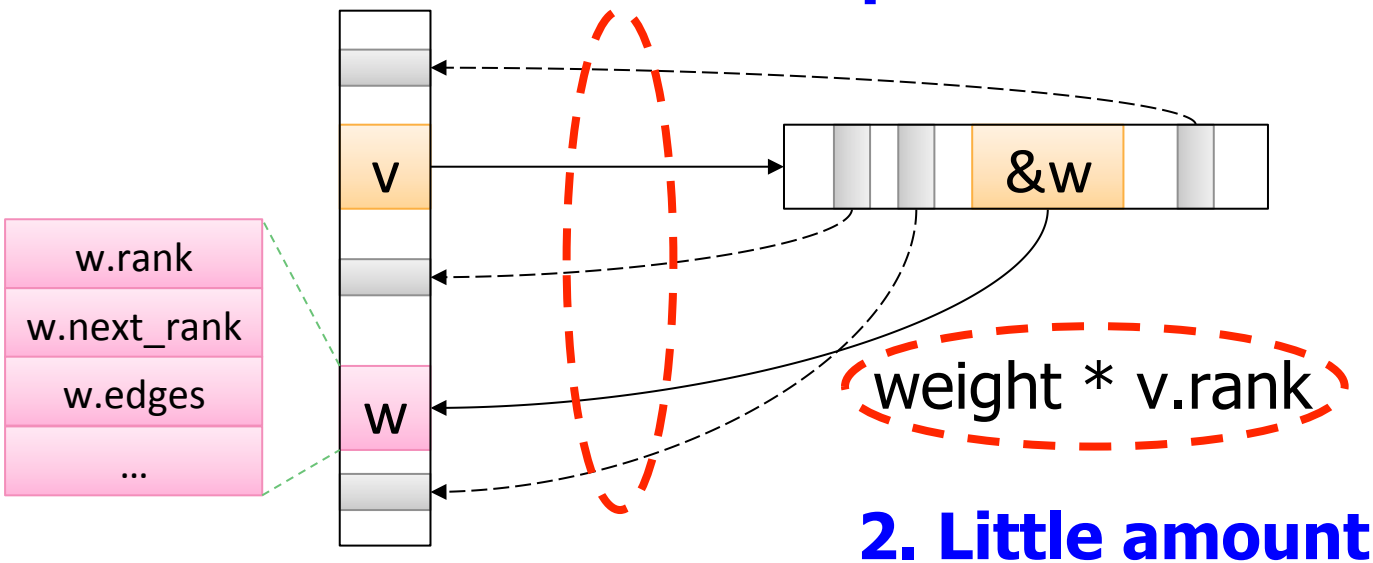
- Scalable large-scale graph processing is challenging



Key Bottlenecks in Graph Processing

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

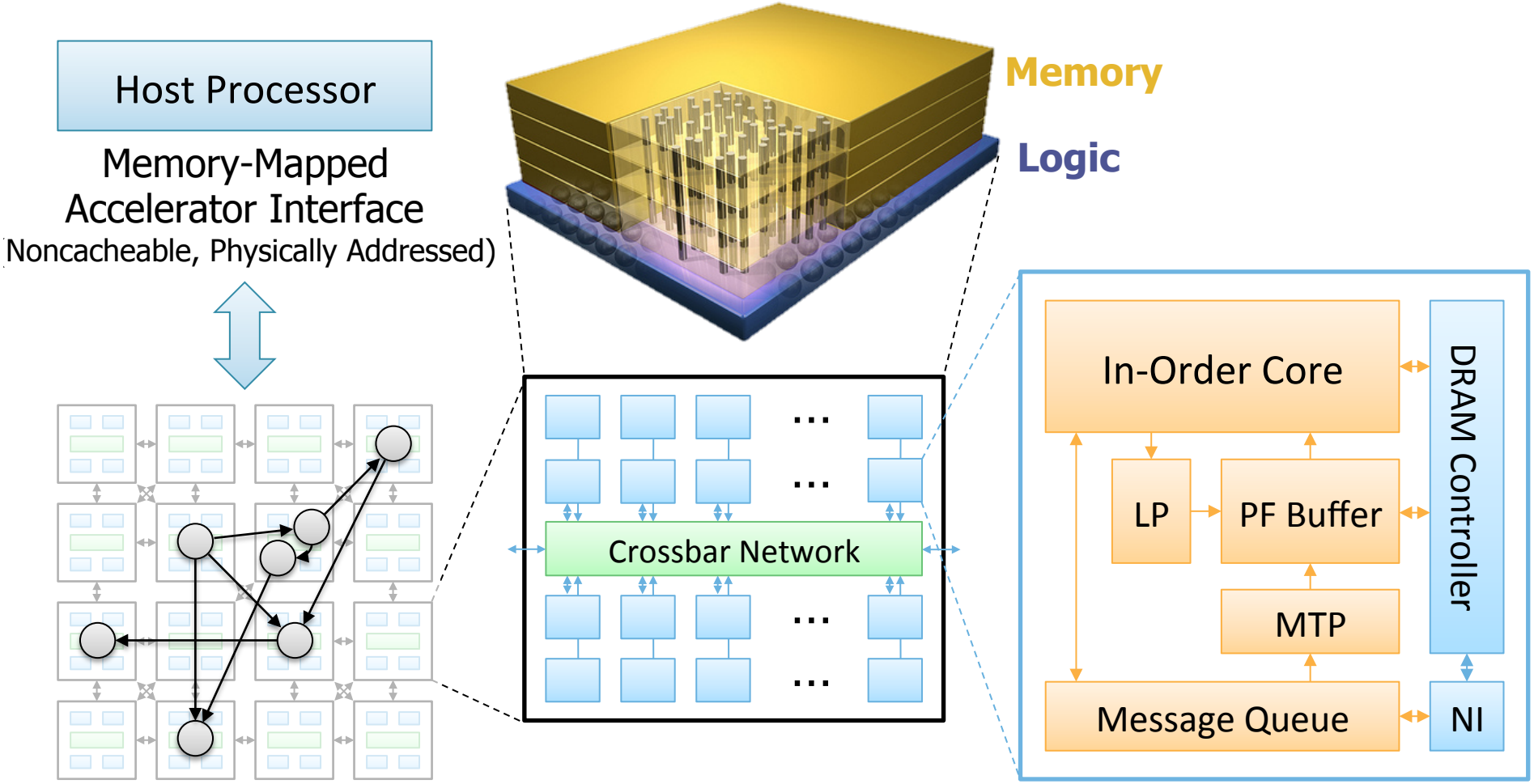
1. Frequent random memory accesses



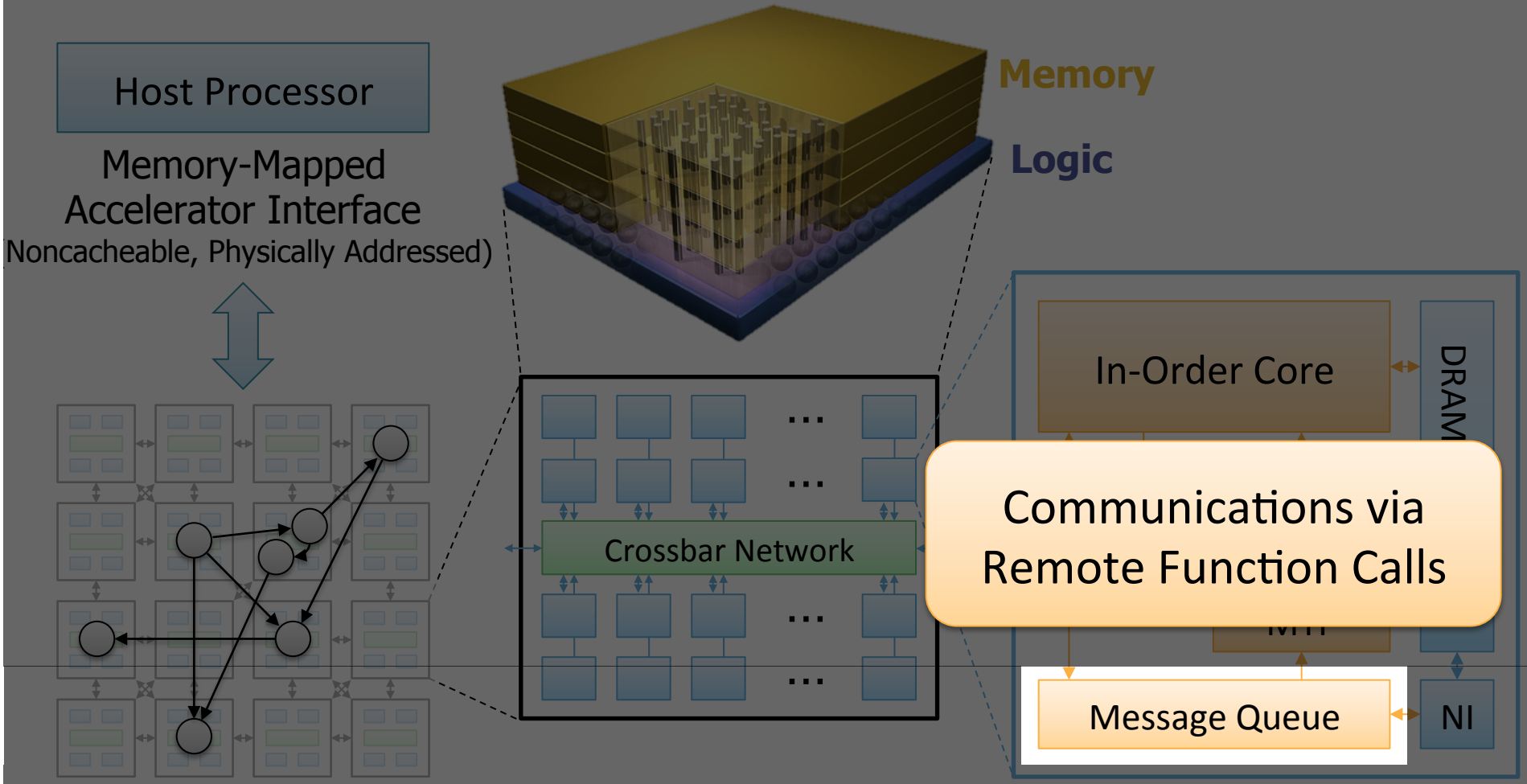
2. Little amount of computation

Tesseract System for Graph Processing

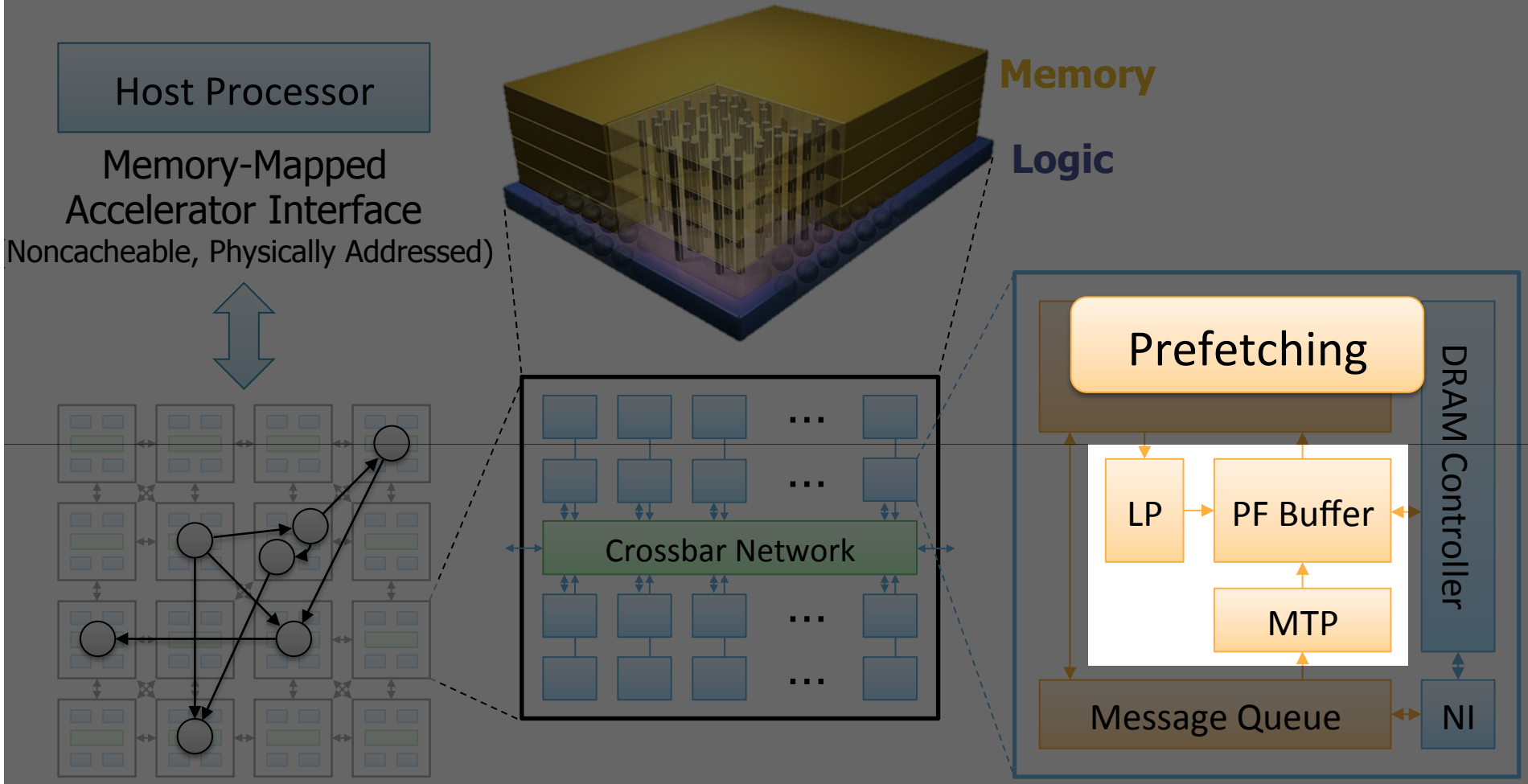
Interconnected set of 3D-stacked memory+logic chips with simple cores



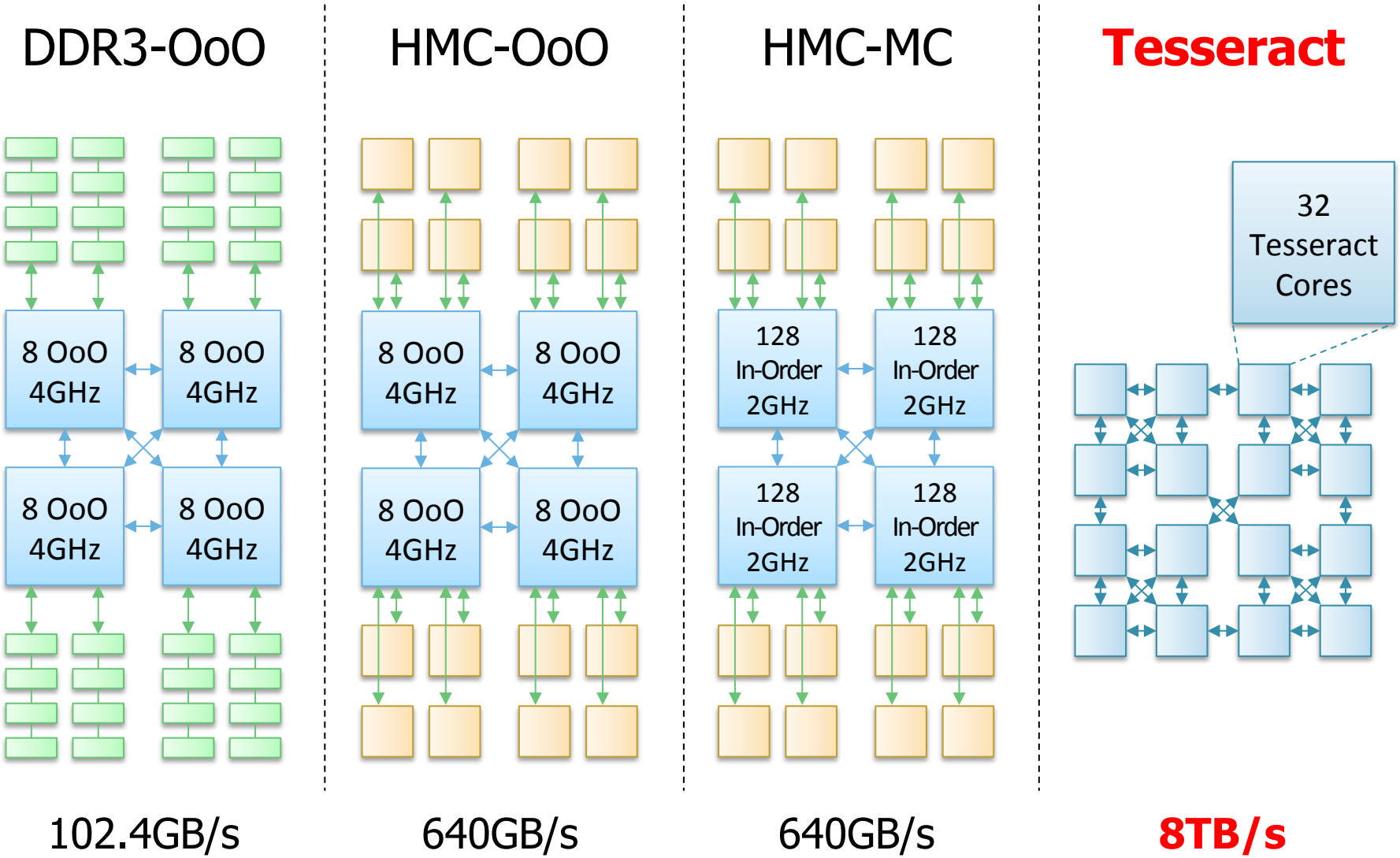
Tesseract System for Graph Processing



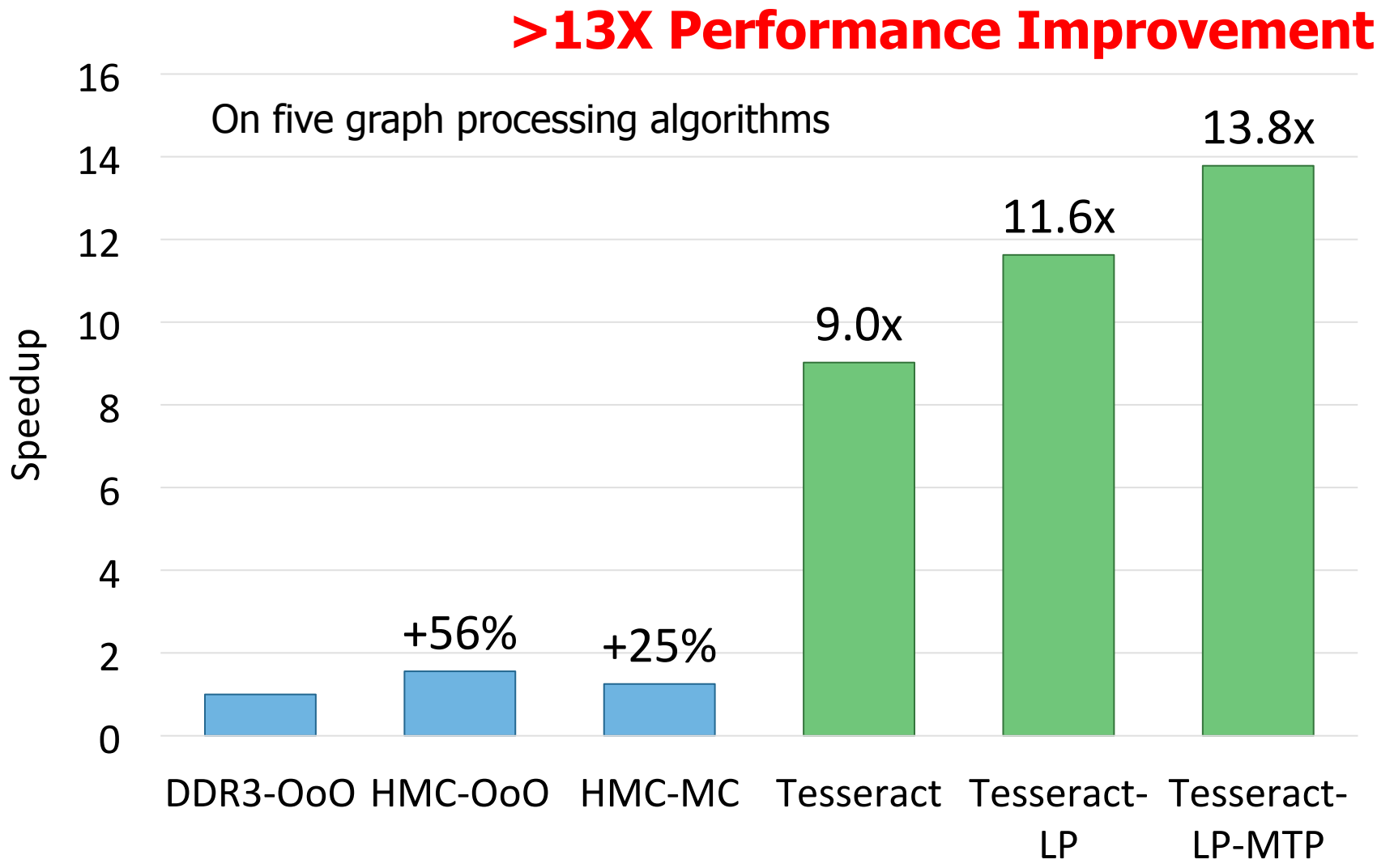
Tesseract System for Graph Processing



Evaluated Systems

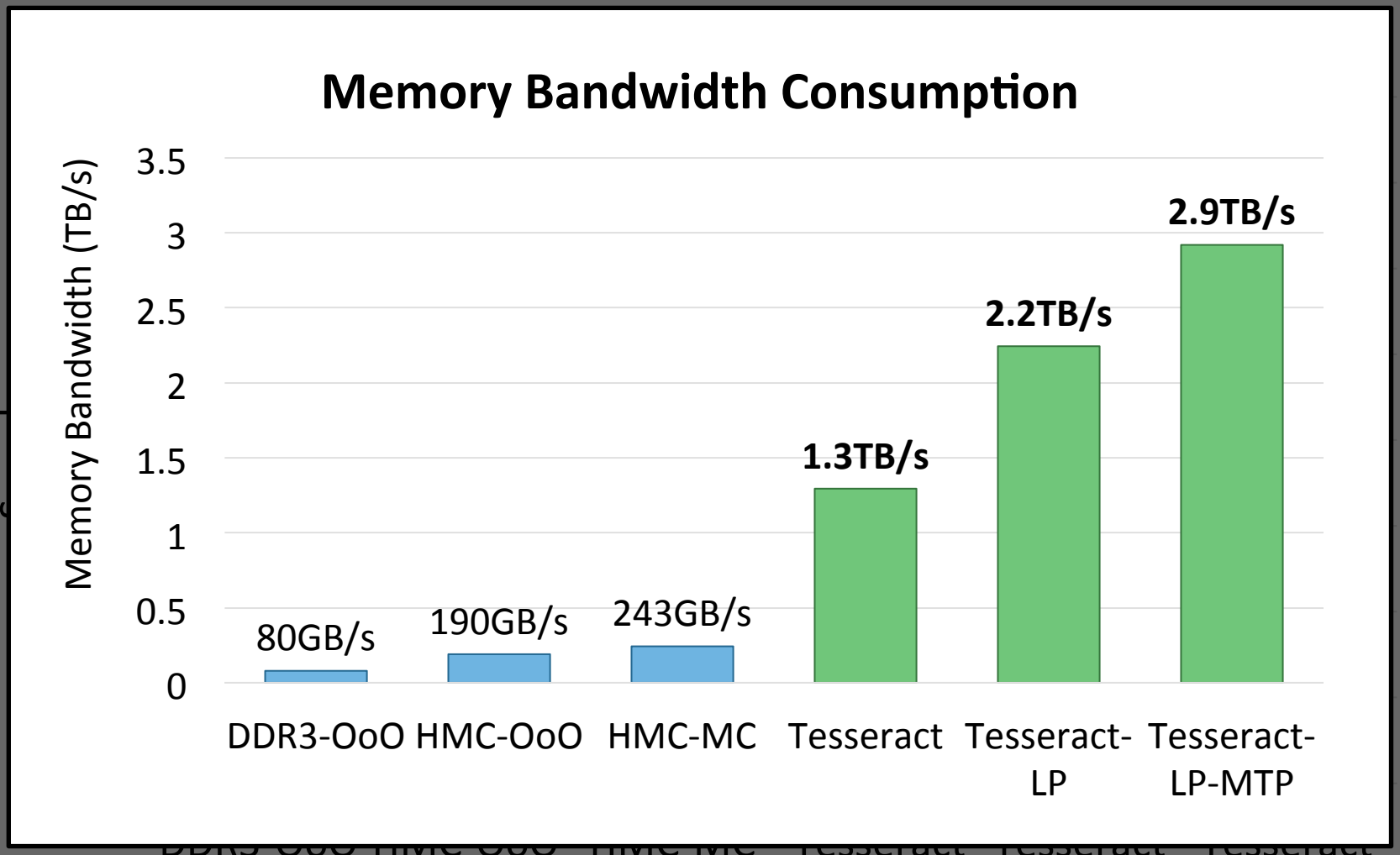


Tesseract Graph Processing Performance

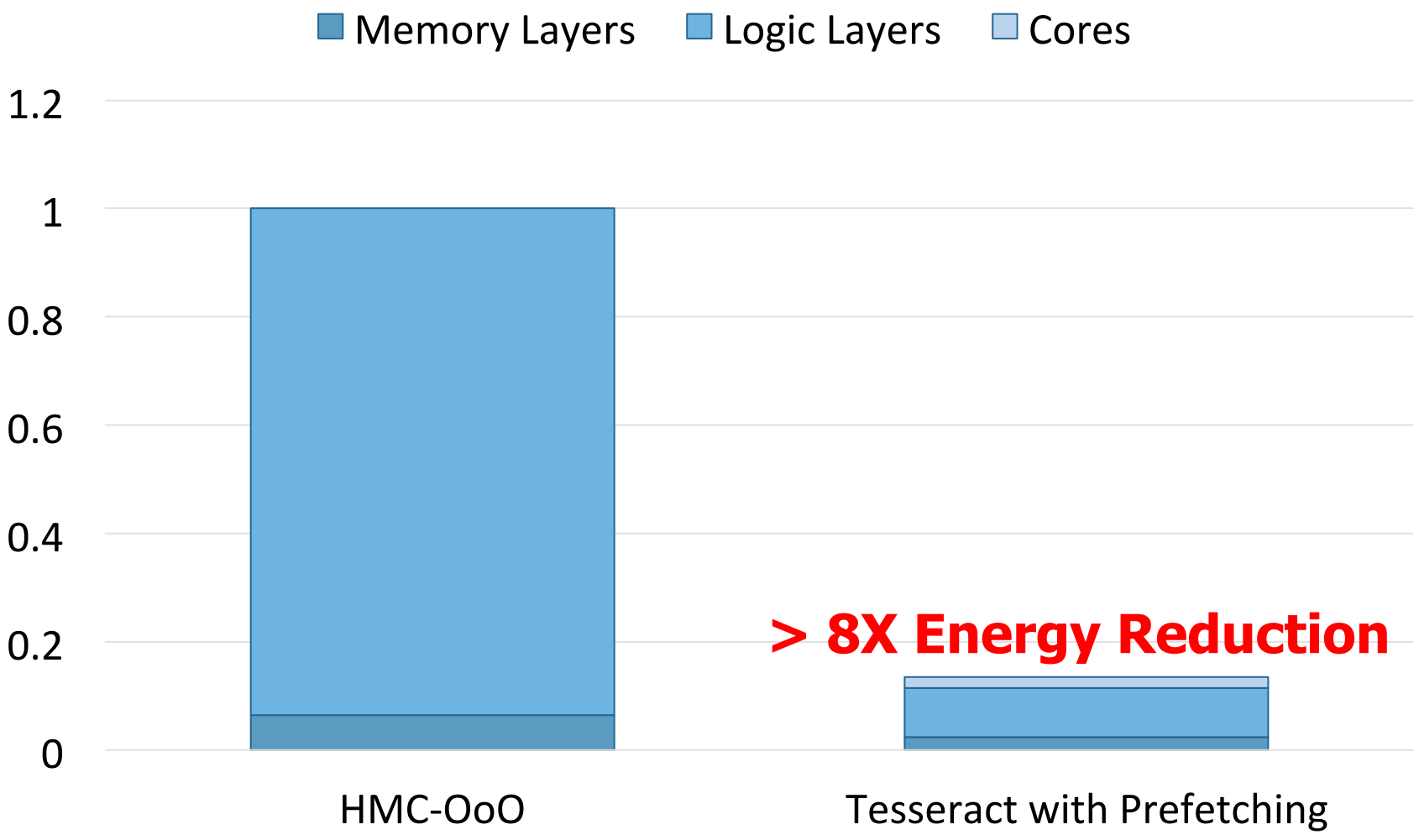


 Ahn+, "A Scalable Processing-in-Memory Accelerator for Parallel Graph Processing" ISCA 2015.

Tesseract Graph Processing Performance



Tesseract Graph Processing System Energy



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

More on Tesseract

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

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

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

We did not cover the following slides in lecture.
These are for your preparation for the next lecture.

Computer Architecture

Lecture 6: Low-Latency DRAM and Processing In Memory

Prof. Onur Mutlu
ETH Zürich

Fall 2017

5 October 2017

Two Key Questions in 3D Stacked PIM

- What is the minimal processing-in-memory support we can provide ?
  - without changing the system significantly
  - while achieving significant benefits of processing in 3D-stacked memory

- How can we accelerate important applications if we use 3D-stacked memory as a coarse-grained accelerator?
  - what is the architecture and programming model?
  - what are the mechanisms for acceleration?

PEI: PIM-Enabled Instructions (Ideas)

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

PEI: PIM-Enabled Instructions (Example)

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

pim.add r1, (r2)

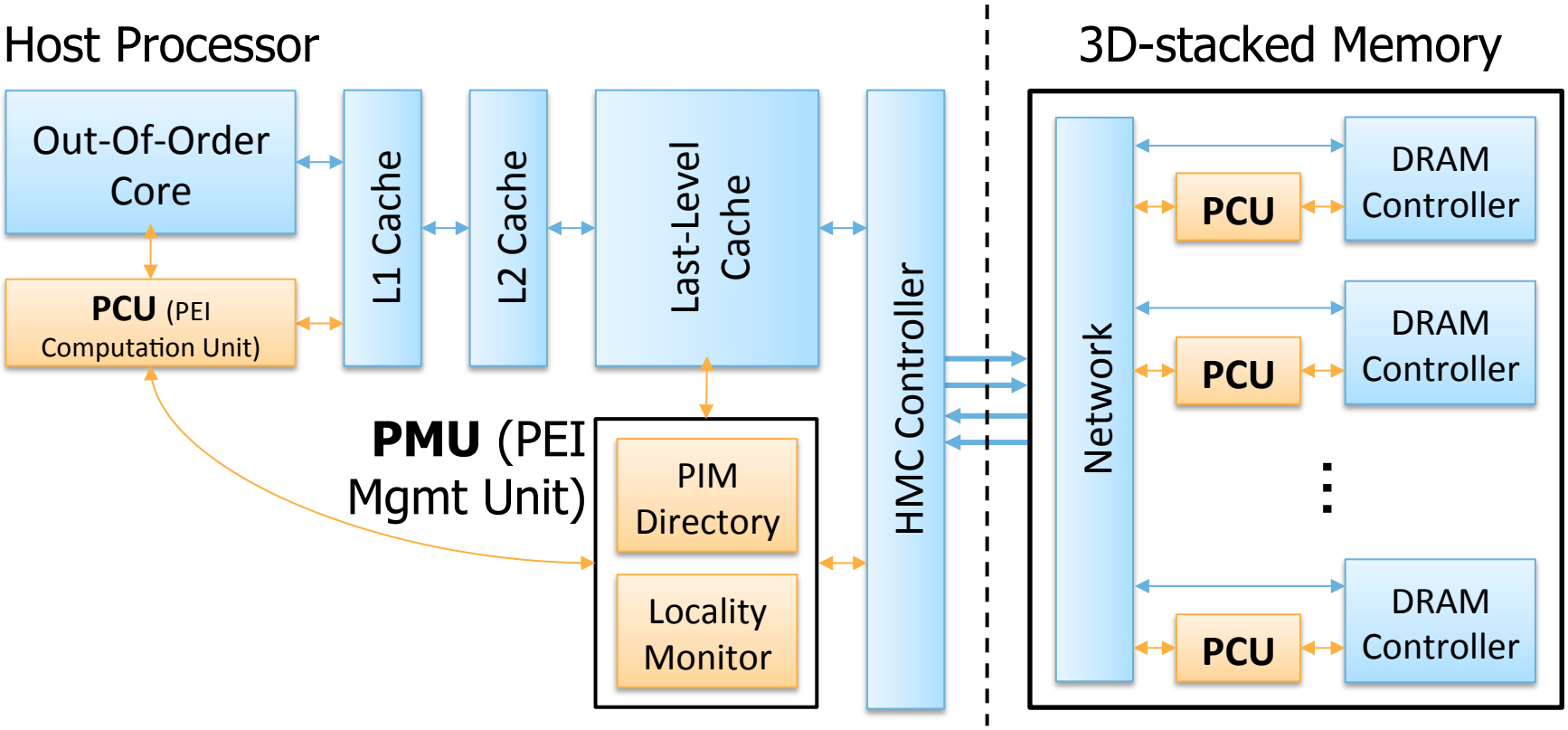
pfence

Table 1: Summary of Supported PIM Operations

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

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

Example (Abstract) PEI uArchitecture



Example PEI uArchitecture

PEI: Initial Evaluation Results

- Initial evaluations with 10 emerging data-intensive workloads
  - Large-scale graph processing
  - In-memory data analytics
  - Machine learning and data mining
  - Three input sets (small, medium, large) for each workload to analyze the impact of data locality

Table 2: Baseline Simulation Configuration

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

- Pin-based cycle-level x86-64 simulation

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

More on PIM-Enabled Instructions

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

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

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

More on PIM Design: 3D-Stacked GPU I

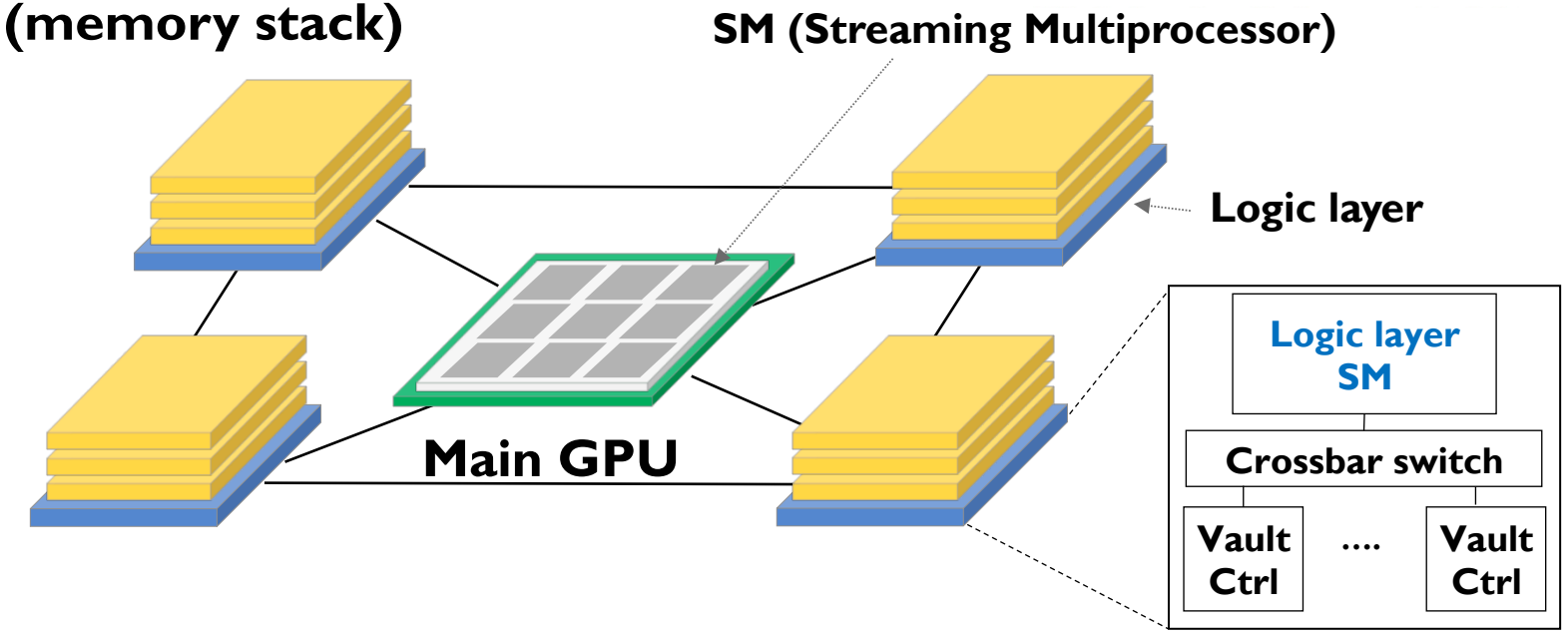
- Kevin Hsieh, Eiman Ebrahimi, Gwangsun Kim, Niladrish Chatterjee, Mike O'Connor, Nandita Vijaykumar, Onur Mutlu, and Stephen W. Keckler, **"Transparent Offloading and Mapping (TOM): Enabling Programmer-Transparent Near-Data Processing in GPU Systems"**
  Proceedings of the
  43rd International Symposium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

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

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

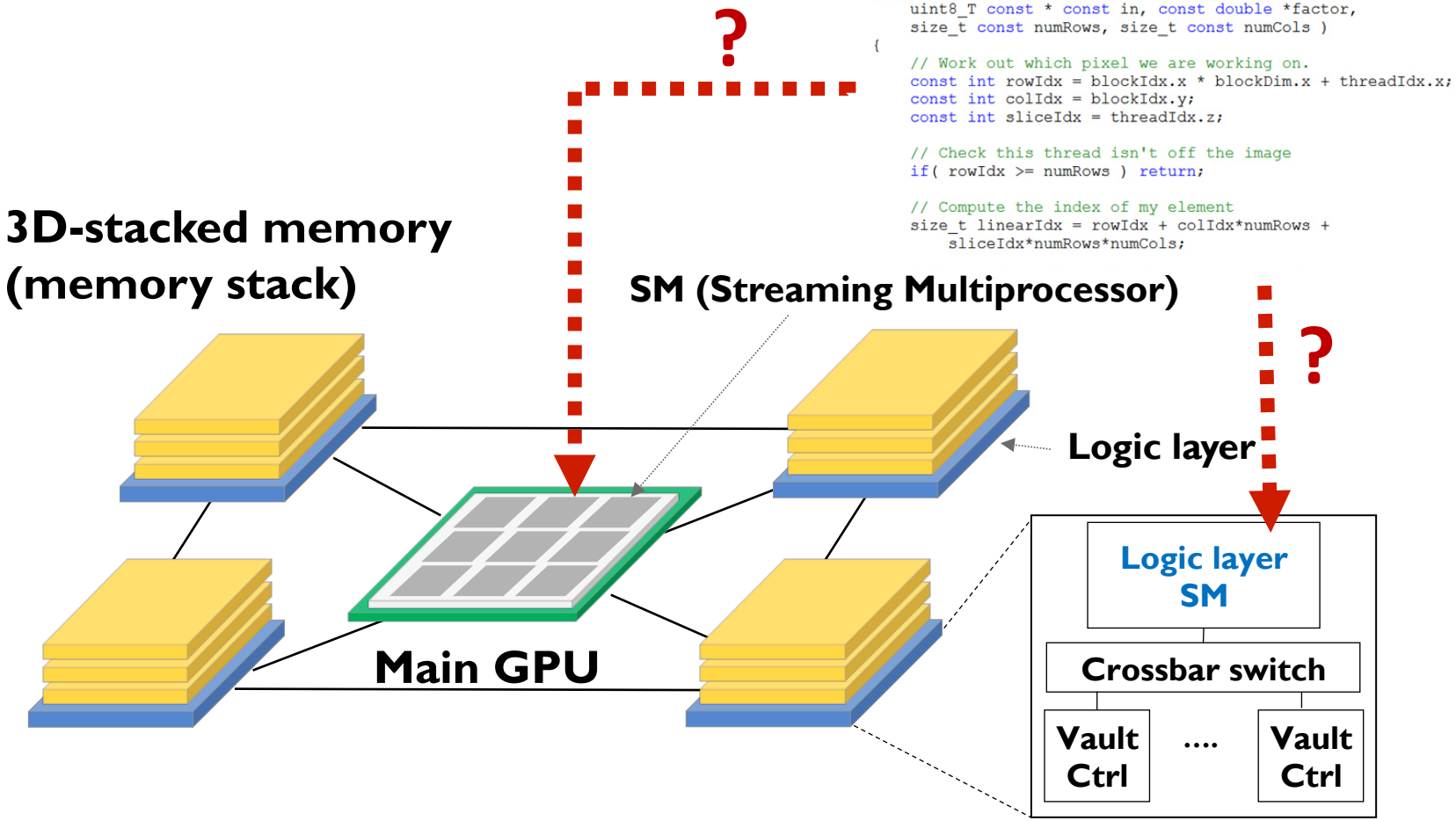
Key Challenge 1

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

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

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



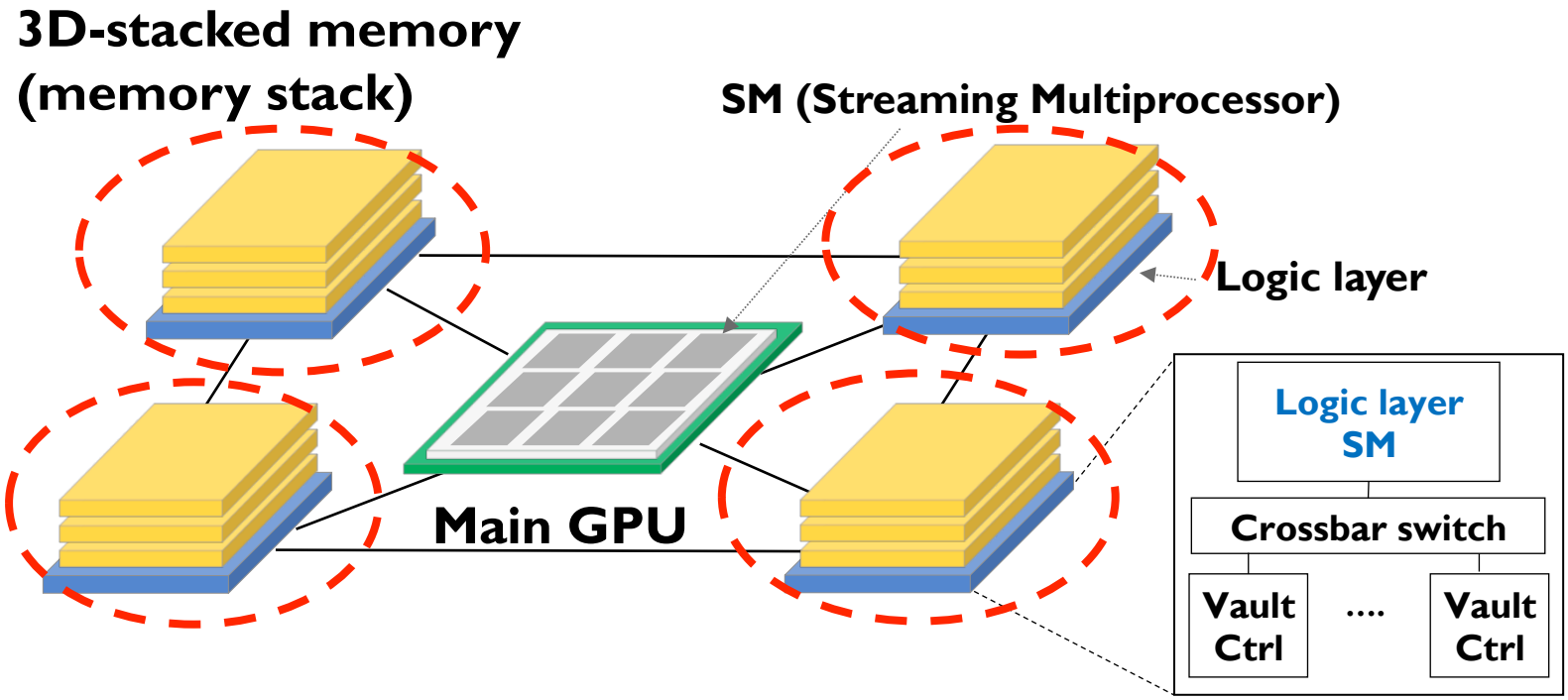
Key Challenge 1

- **Challenge 1:** Which operations should be executed on the logic layer SMs?



Key Challenge 2

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



More on PIM Design: 3D-Stacked GPU II

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

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

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

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

More on PIM: Linked Data Structures

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

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

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

Executive Summary

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

More on PIM Design: Dependent Misses

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

Accelerating Dependent Cache Misses with an Enhanced Memory Controller

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

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

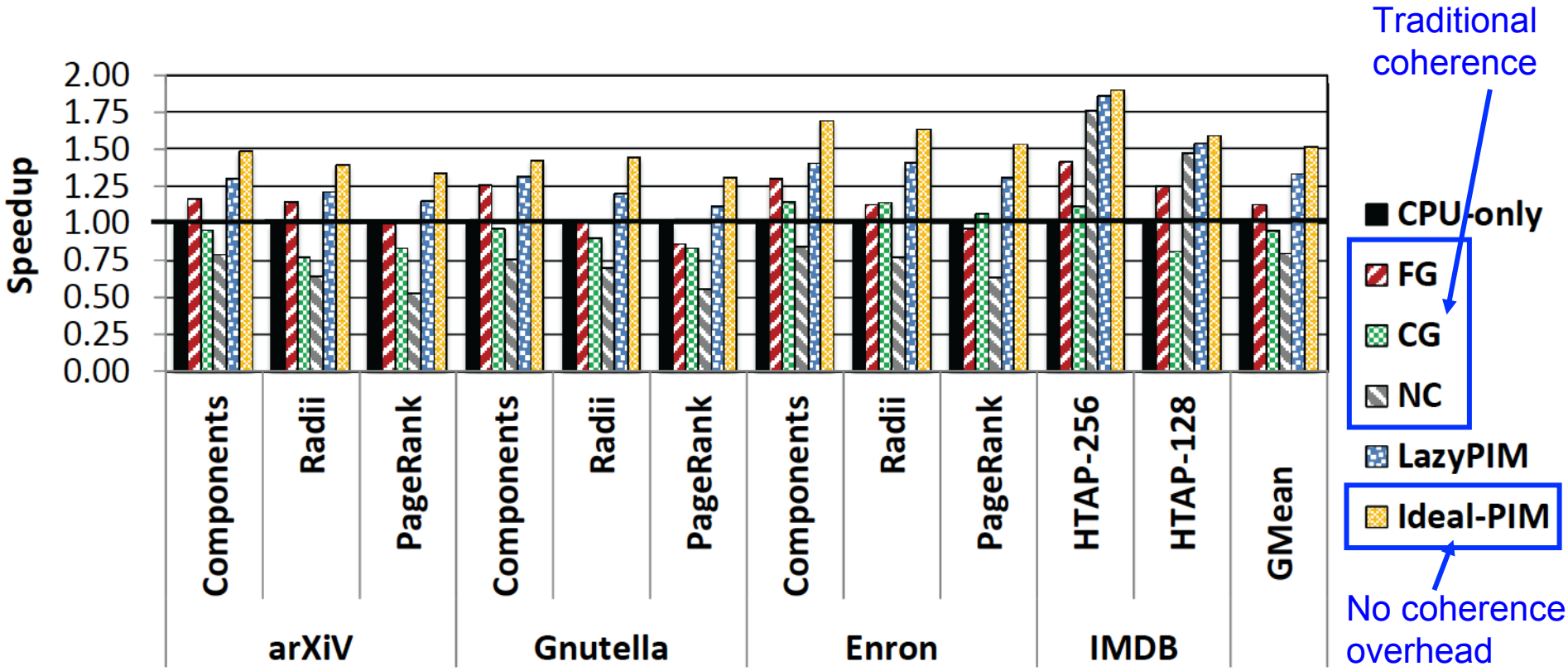
More on PIM Design: Coherence

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

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

Amirali Boroumand[†], Saugata Ghose[†], Minesh Patel[†], Hasan Hassan[†§], Brandon Lucia[†],
Kevin Hsieh[†], Krishna T. Malladi[⋆], Hongzhong Zheng[⋆], and Onur Mutlu[‡†]
[†]*Carnegie Mellon University* [⋆]*Samsung Semiconductor, Inc.* [§]*TOBB ETÜ* [‡]*ETH Zürich*

Traditional Coherence Approaches Do Not Work



More on PIM Design: Data Structures

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

Concurrent Data Structures for Near-Memory Computing

Zhiyu Liu
Computer Science Department
Brown University
zhiyu_liu@brown.edu

Irina Calciu
VMware Research Group
icalciu@vmware.com

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

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

Simulation Infrastructures for PIM

- Ramulator extended for PIM
  - Flexible and extensible DRAM simulator
  - Can model many different memory standards and proposals
  - Kim+, "**Ramulator: A Flexible and Extensible DRAM Simulator**", IEEE CAL 2015.
  - https://github.com/CMU-SAFARI/ramulator

Ramulator: A Fast and Extensible DRAM Simulator

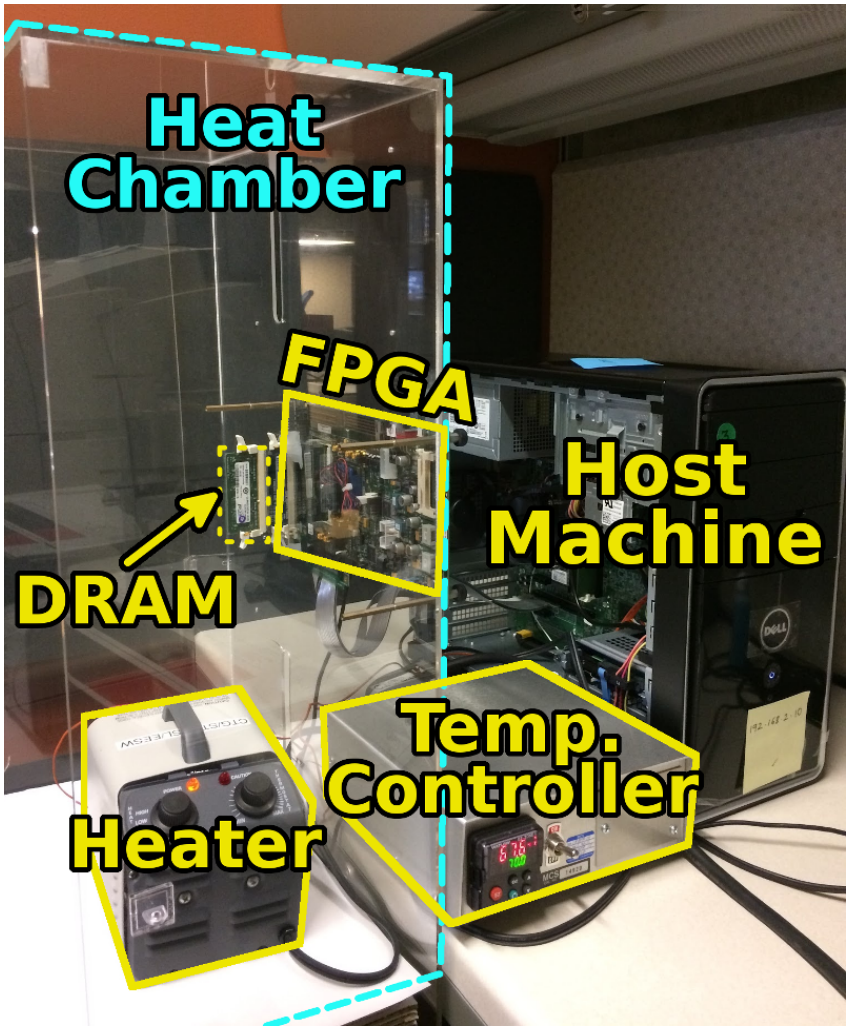
Yoongu Kim[1] Weikun Yang[1,2] Onur Mutlu[1]
[1]Carnegie Mellon University [2]Peking University

An FPGA-based Test-bed for PIM?

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

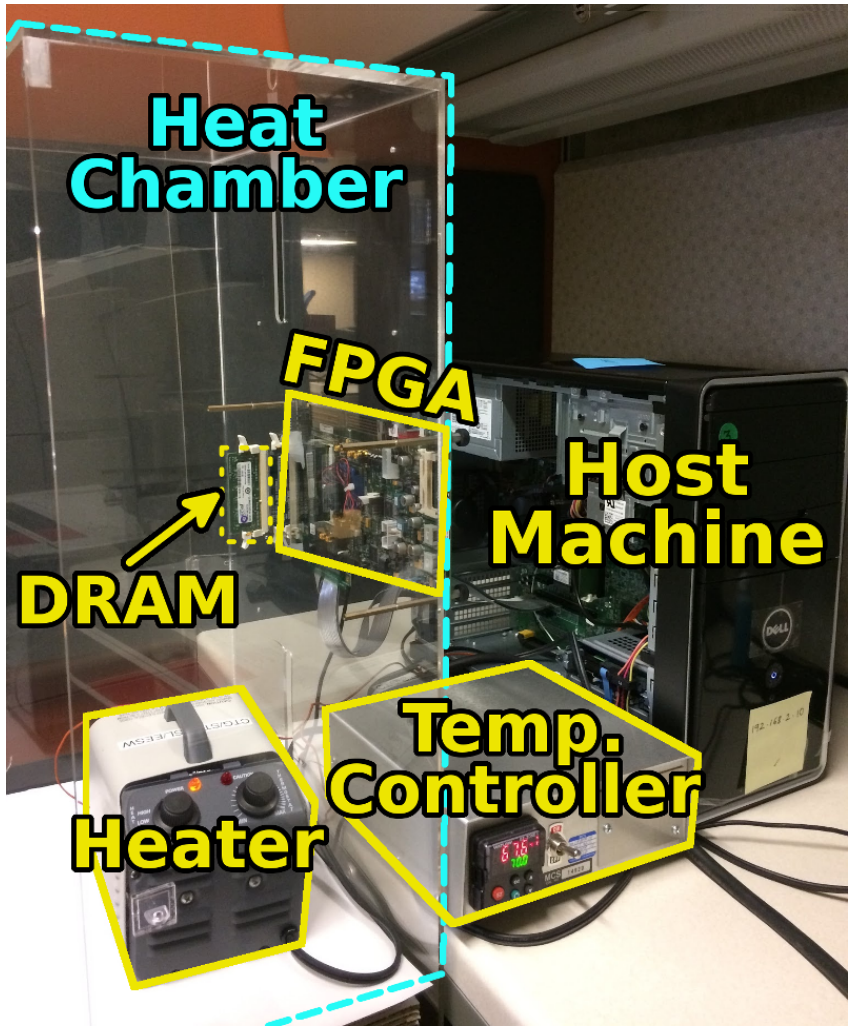
- **Flexible**
- **Easy to Use (C++ API)**
- **Open-source**

  github.com/CMU-SAFARI/SoftMC



An FPGA-based Test-bed for PIM

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



- **Flexible**
- **Easy to Use (C++ API)**
- **Open-source**
  github.com/CMU-SAFARI/SoftMC

Challenge and Opportunity for Future

Fundamentally Energy-Efficient (Data-Centric) Computing Architectures

Challenge and Opportunity for Future

Fundamentally Low-Latency (Data-Centric) Computing Architectures

Barriers to Adoption of PIM

1. Functionality of and applications for PIM

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

3. System support: coherence & virtual memory

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

5. Infrastructures to assess benefits and feasibility

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

Kevin Hsieh

Samira Khan, Nandita Vijaykumar, Kevin K. Chang,

Amirali Boroumand, Saugata Ghose, Onur Mutlu

Carnegie Mellon University

University of Virginia

ETH zürich

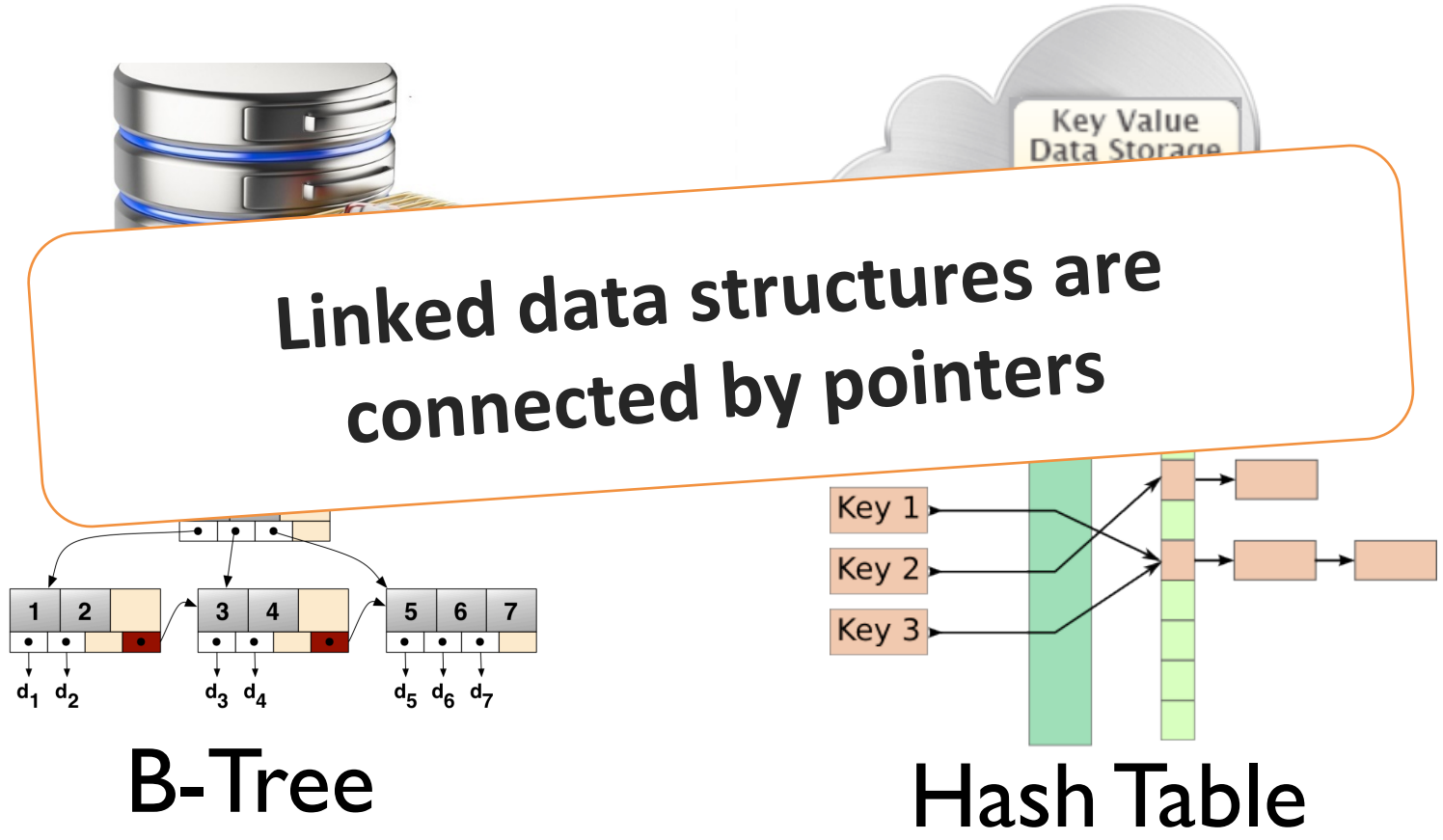
SAFARI

Executive Summary

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

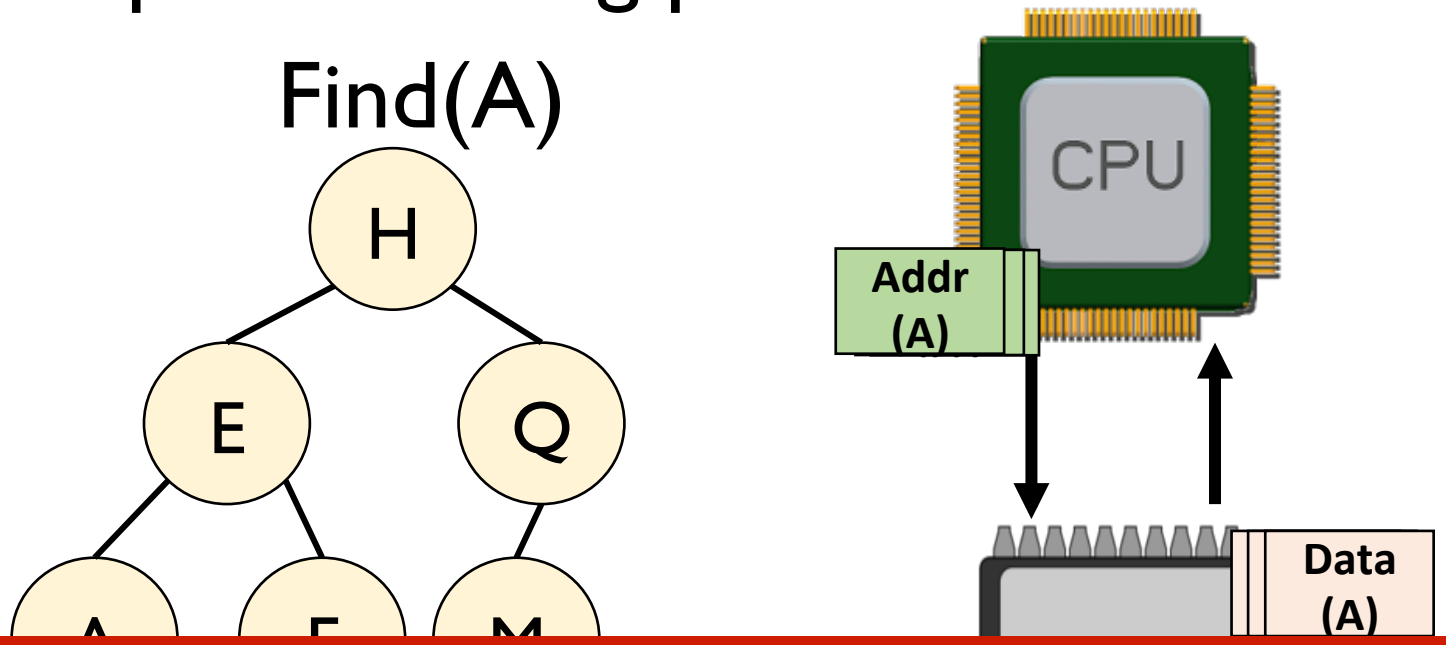
Linked Data Structures

- Linked data structures are widely used in many important applications



The Problem: Pointer Chasing

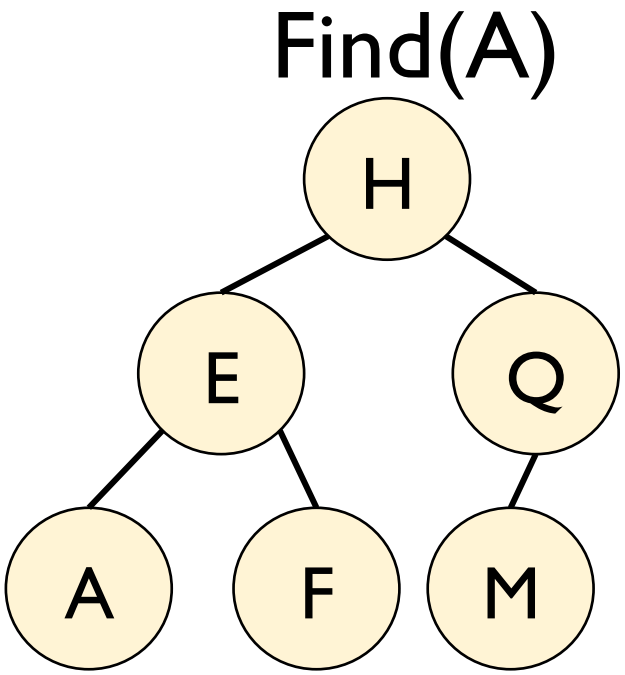
- Traversing linked data structures requires chasing pointers

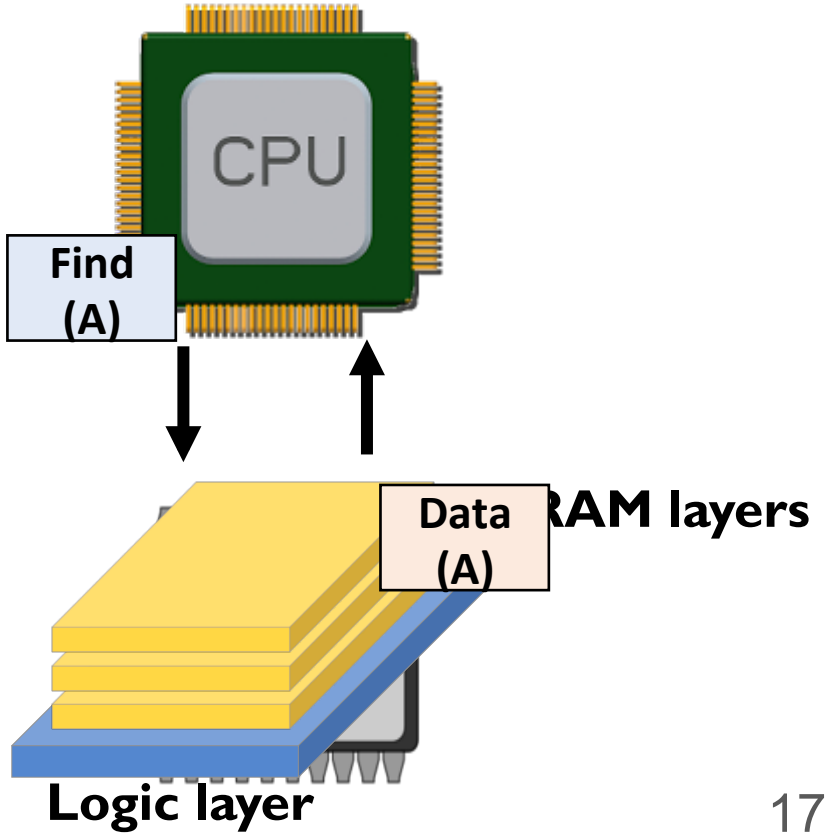


Serialized and irregular access pattern
6X cycles per instruction in real workloads

Our Goal

Accelerating pointer chasing inside main memory

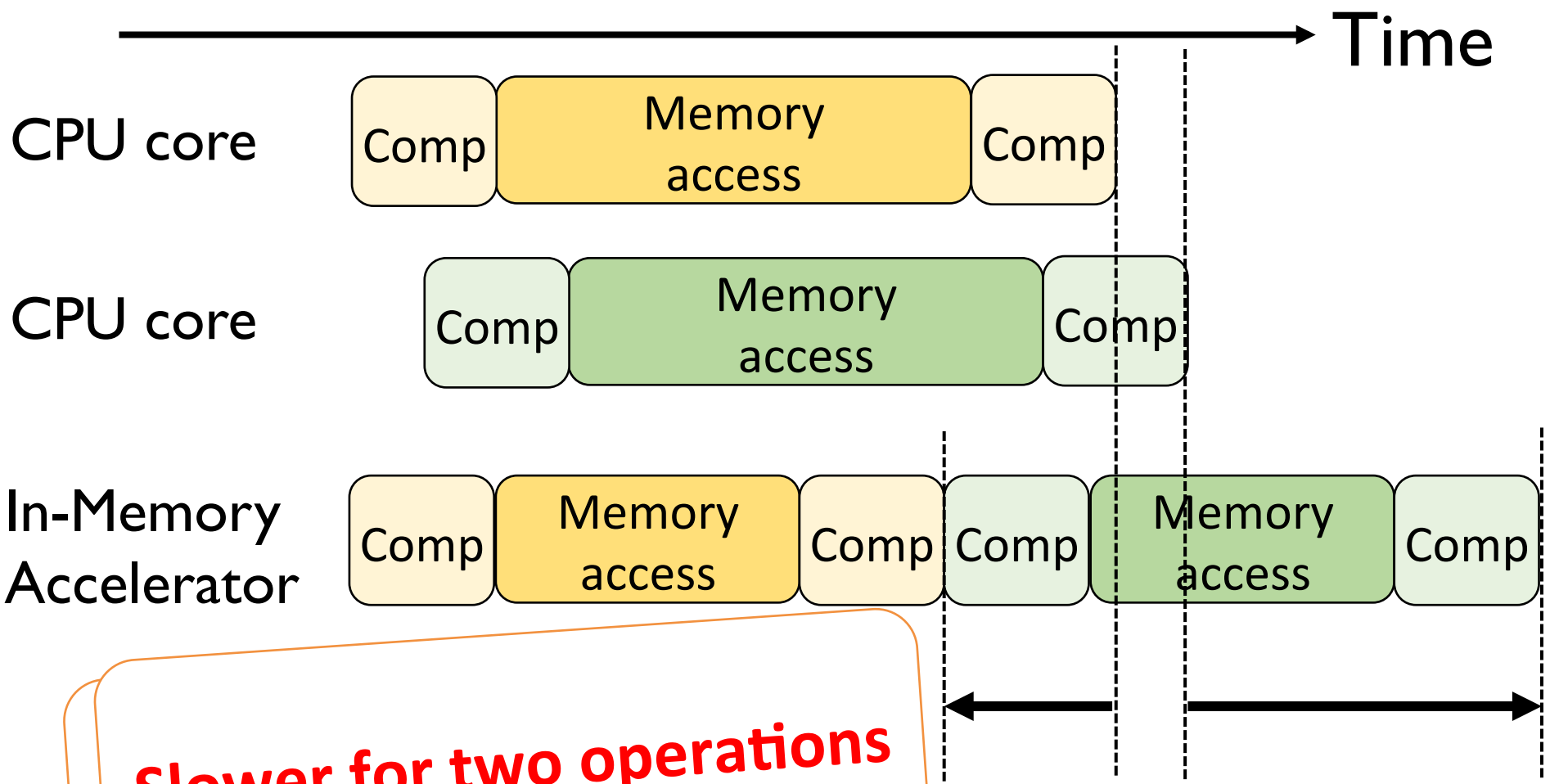




Outline

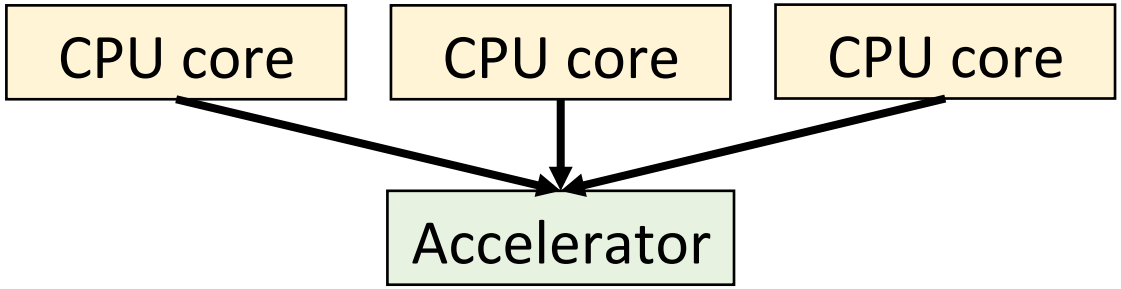
- Motivation and Our Approach
- Parallelism Challenge
- IMPICA Core Architecture
- Address Translation Challenge
- IMPICA Page Table
- Evaluation
- Conclusion

Parallelism Challenge

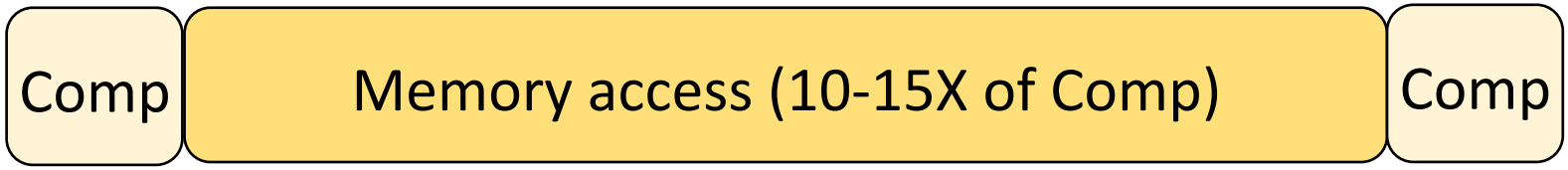


Parallelism Challenge and Opportunity

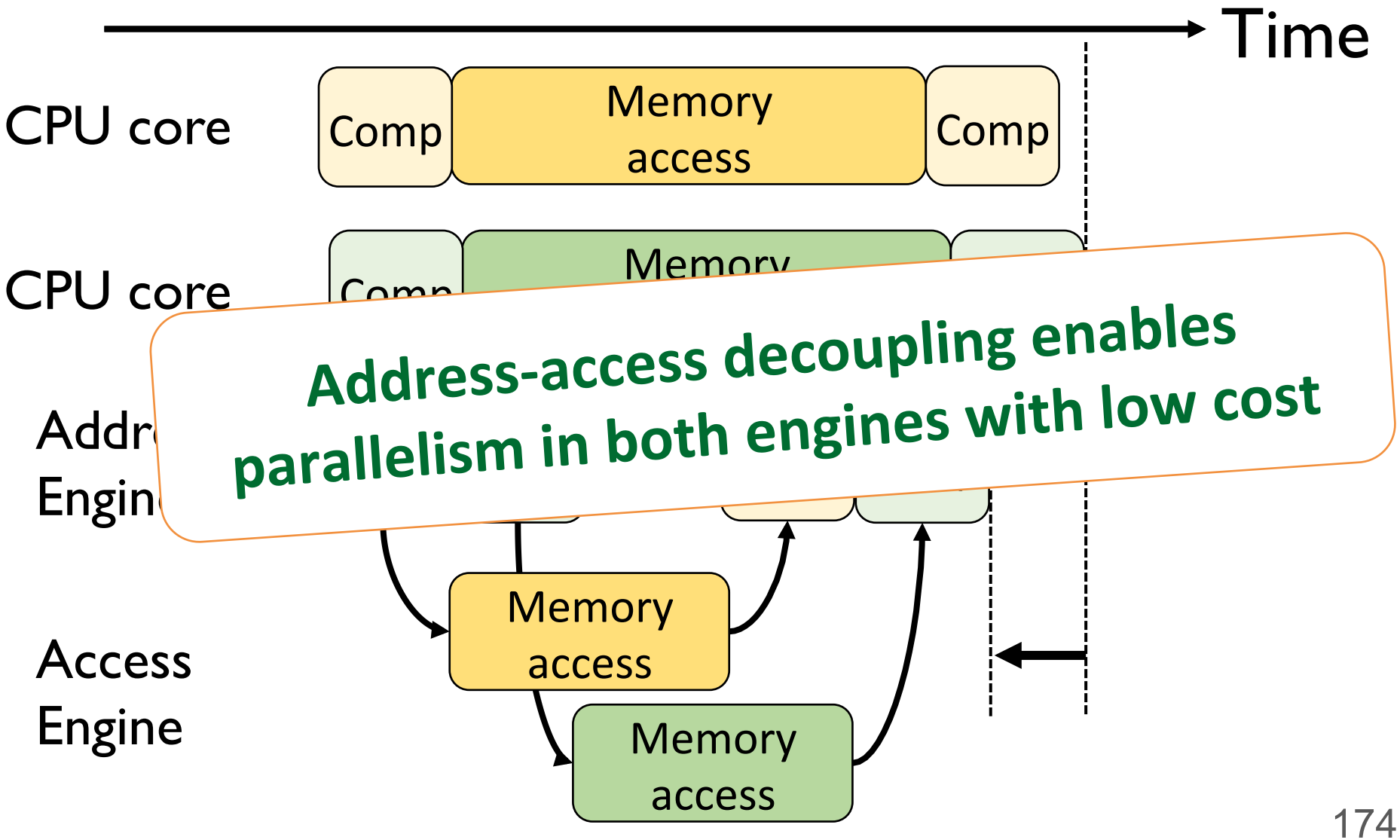
- A simple in-memory accelerator can still be slower than multiple CPU cores



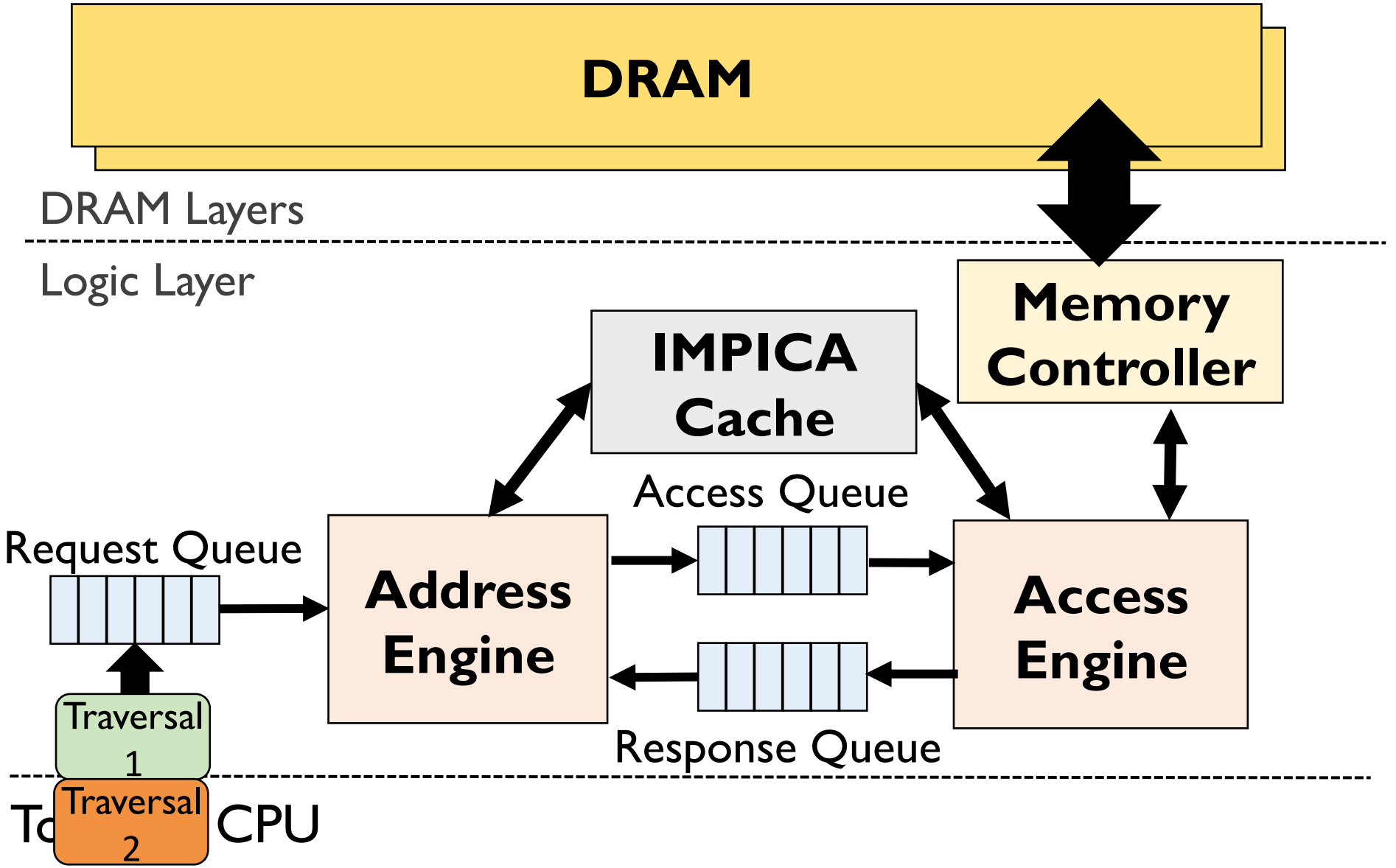
- Opportunity: a pointer-chasing accelerator spends a long time waiting for memory



Our Solution: Address-Access Decoupling



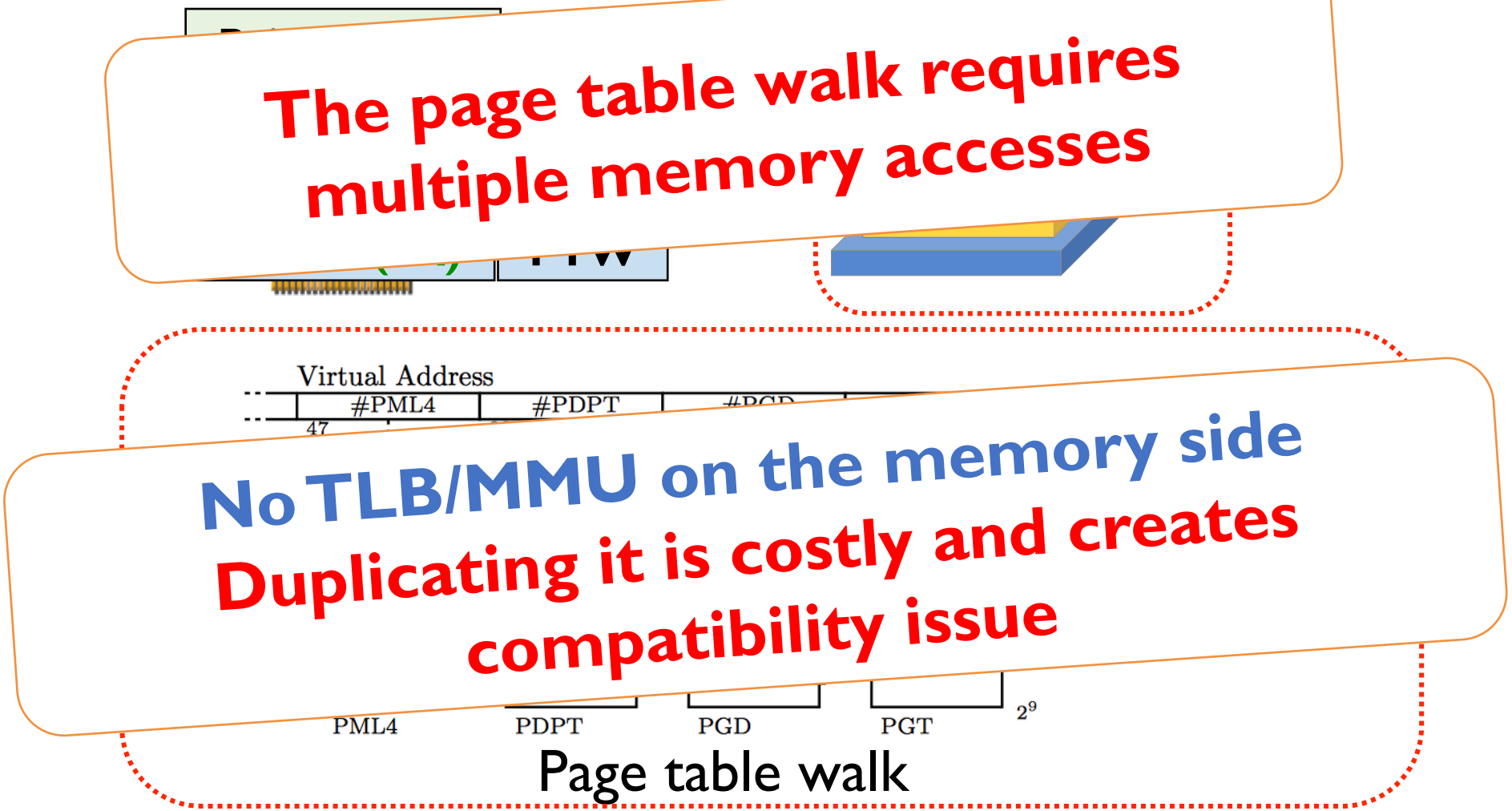
IMPICA Core Architecture



Outline

- Motivation and Our Approach
- Parallelism Challenge
- IMPICA Core Architecture
- Address Translation Challenge
- IMPICA Page Table
- Evaluation
- Conclusion

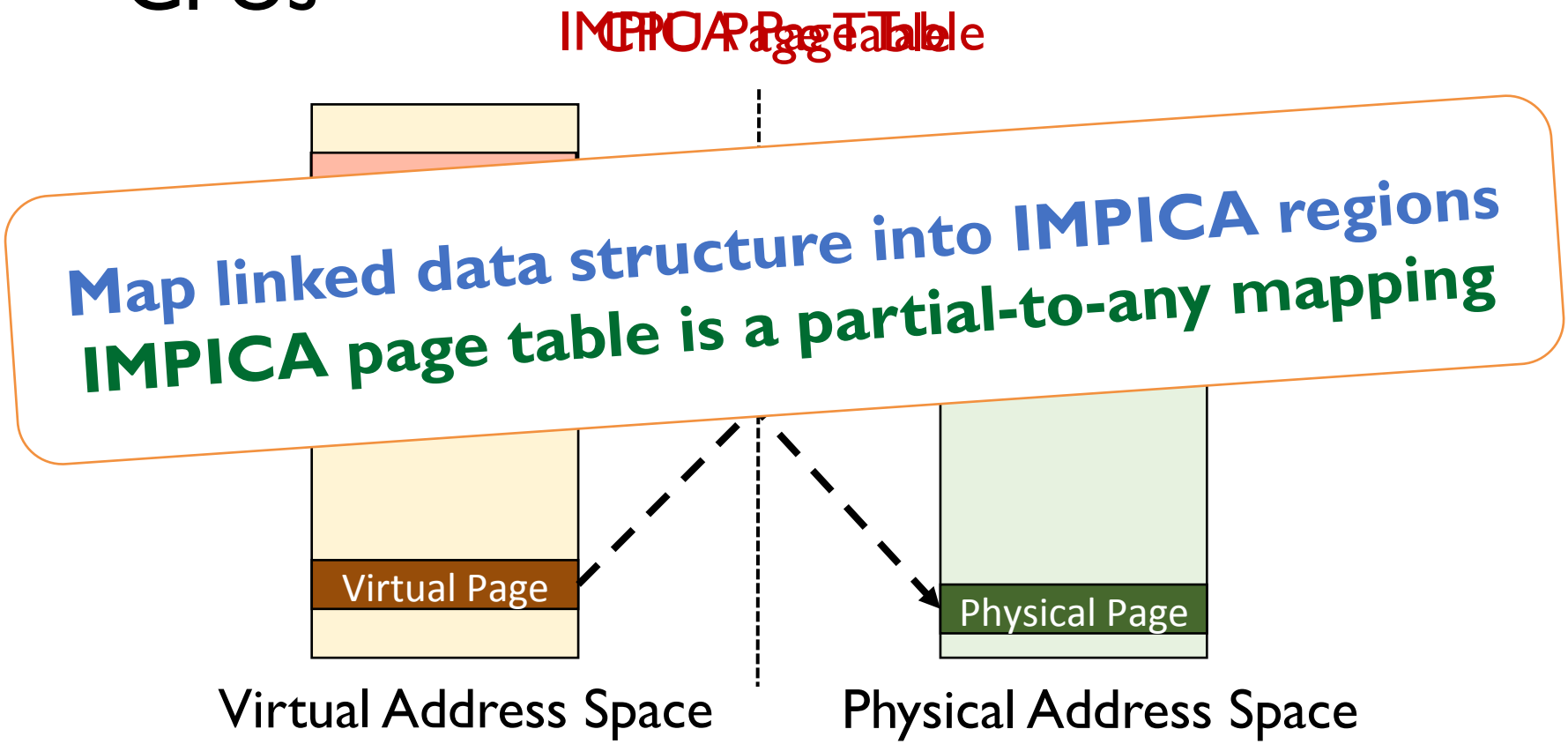
Address Translation Challenge



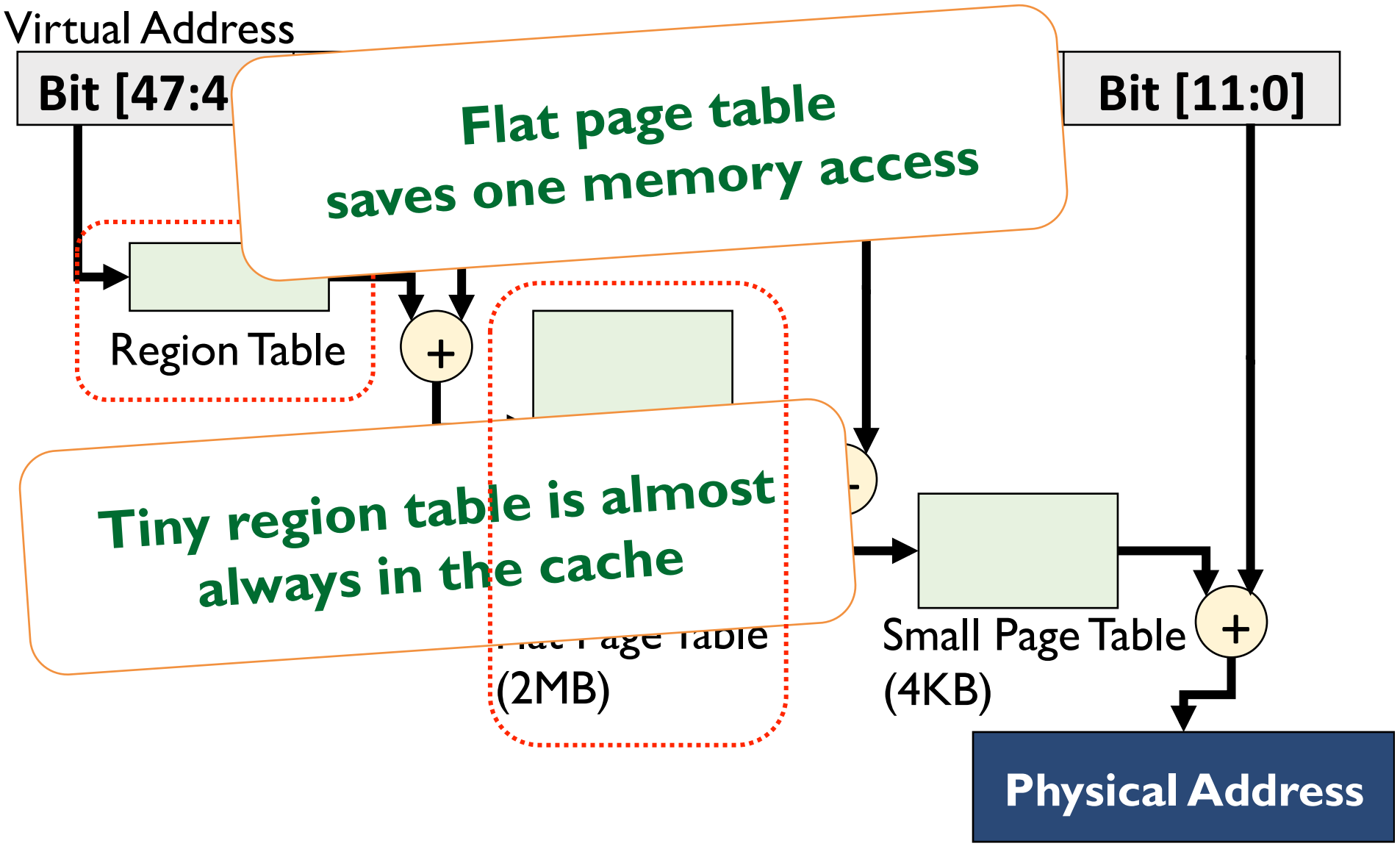
Page table walk

Our Solution: IMPICA Page Table

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



IMPICA Page Table: Mechanism



Outline

- Motivation and Our Approach
- Parallelism Challenge
- IMPICA Core Architecture
- Address Translation Challenge
- IMPICA Page Table
- Evaluation
- Conclusion

Evaluated Workloads

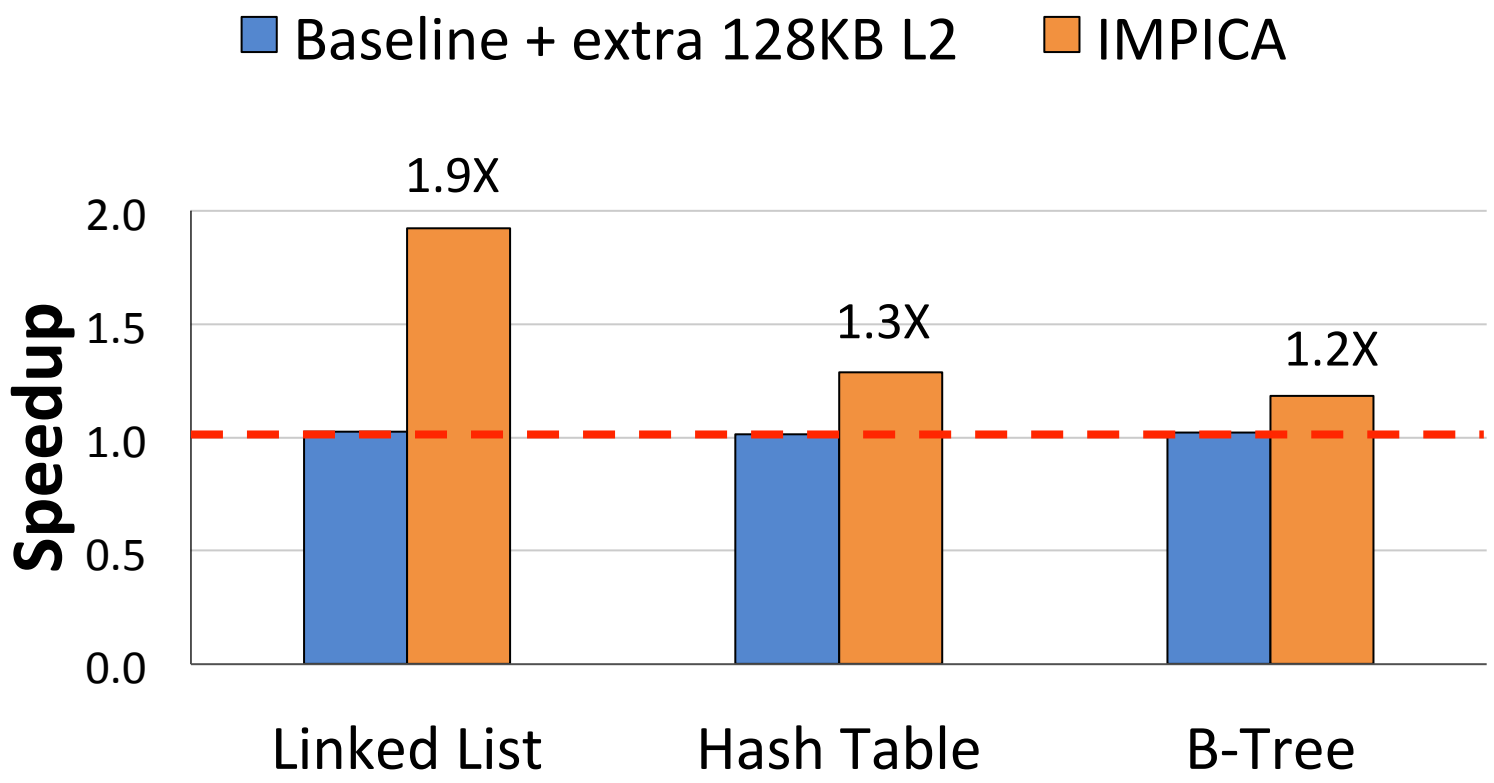
- Microbenchmarks
  - Linked list (from Olden benchmark)
  - Hash table (from Memcached)
  - B-tree (from DBx1000)

- Application
  - DBx1000 (with TPC-C benchmark)

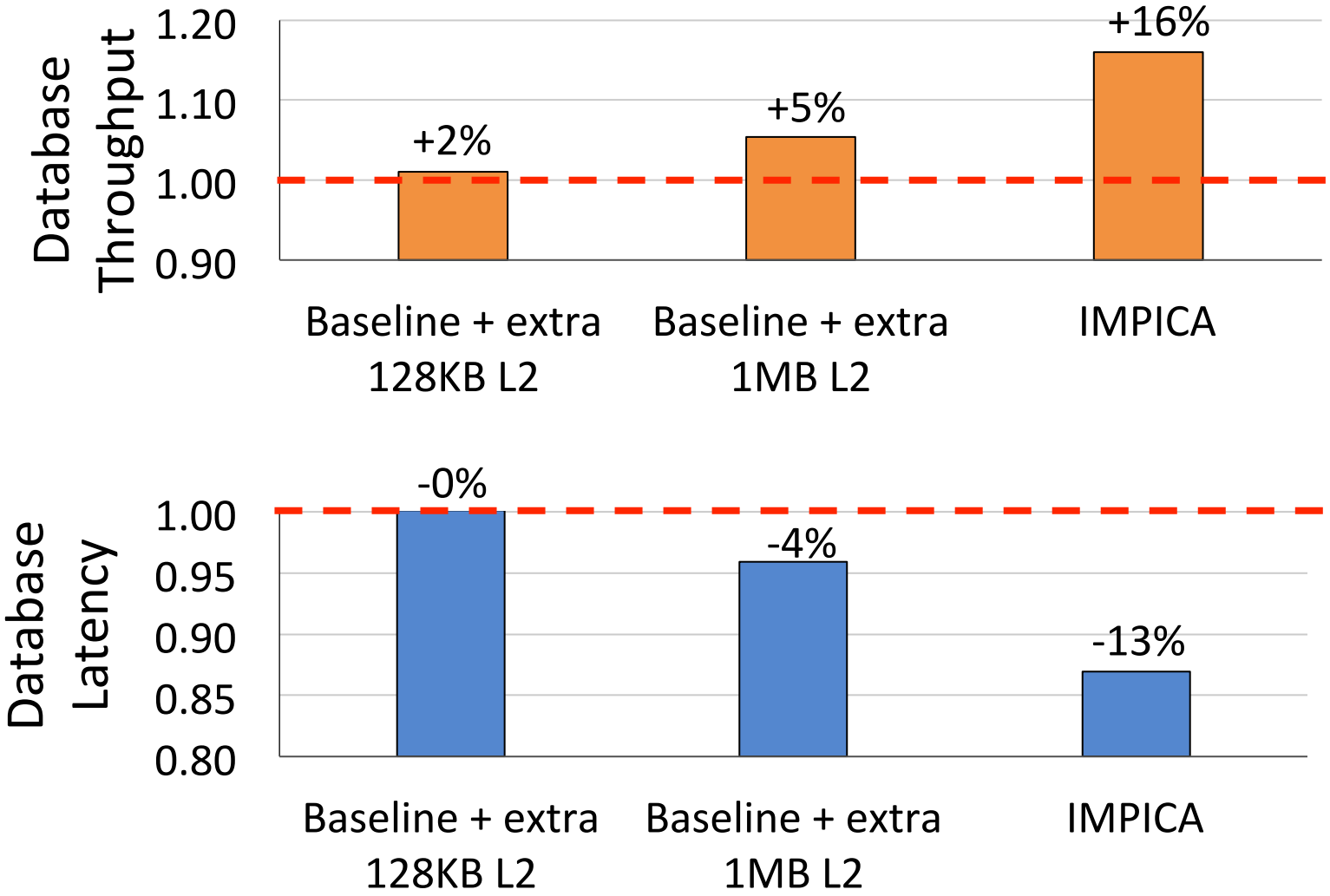
Evaluation Methodology

- Simulator: gem5
- System Configuration
  - CPU
    - 4 OoO cores, 2GHz
    - Cache: 32KB L1, 1MB L2
  - IMPICA
    - 1 core, 500MHz, 32KB Cache
  - Memory Bandwidth
    - 12.8 GB/s for CPU, 51.2 GB/s for IMPICA
- Our simulator code is open source
  - https://github.com/CMU-SAFARI/IMPICA

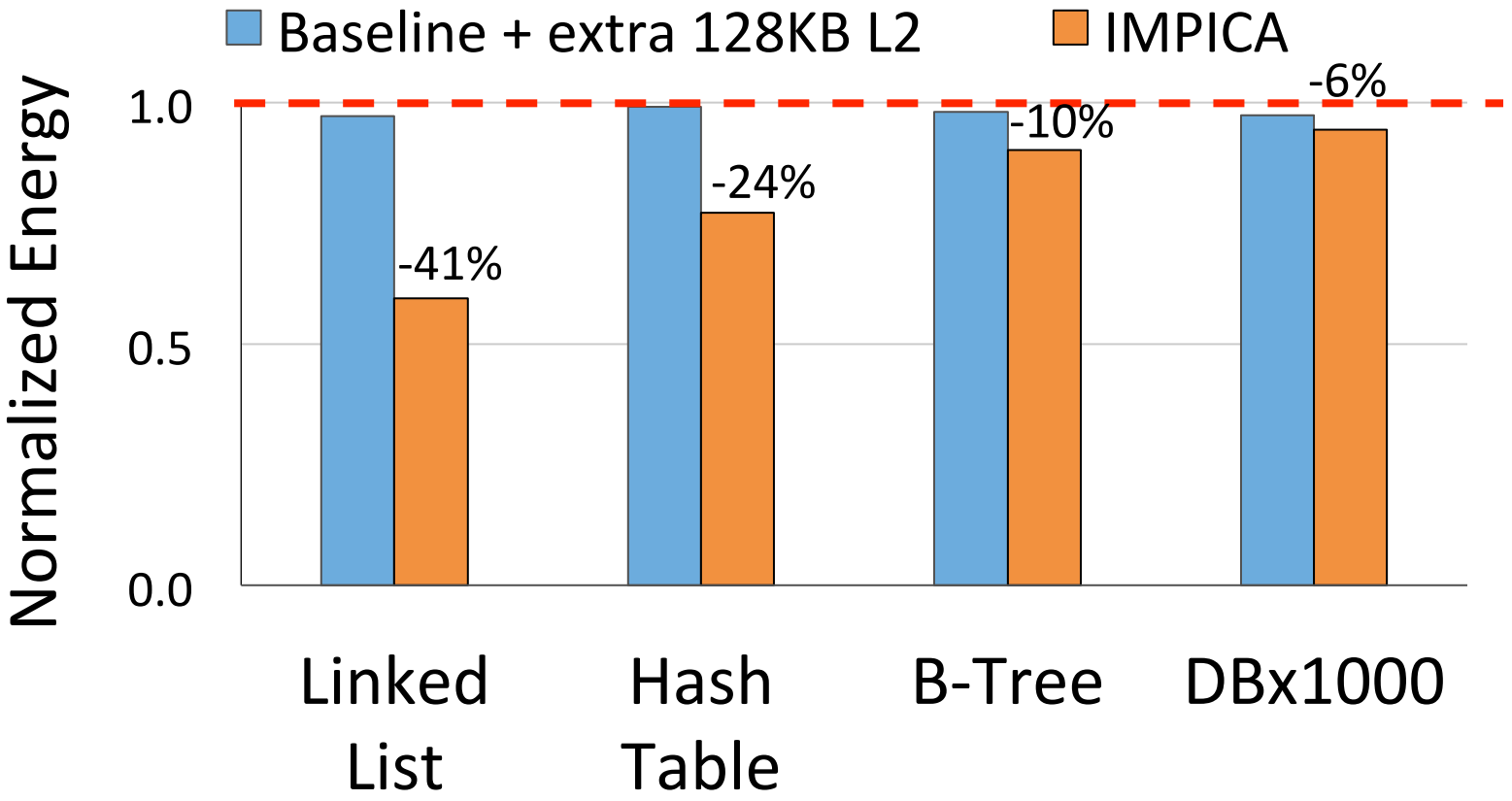
Result – Microbenchmark Performance



Result – Database Performance



System Energy Consumption



Area and Power Overhead

CPU (Cortex-A57)	5.85 mm^2 per core
L2 Cache	5 mm^2 per MB
Memory Controller	10 mm^2
IMPICA (+32KB cache)	0.45 mm^2

- Power overhead: average power increases by 5.6%

More in the Paper

- Interface and design considerations
  - CPU interface and programming model
  - Page table management
  - Cache coherence
- Area and power overhead analysis
- Sensitivity to IMPICA page table design

Conclusion

- Performing pointer-chasing inside main memory can greatly speed up the traversal of linked data structures
- **Challenges:** Parallelism challenge and Address translation challenge
- **Our Solution:** In-Memory PoInter Chasing Accelerator
  - Address-access decoupling: enabling parallelism with low cost
  - IMPICA page table: low cost page table structure
- **Key Results**:
  - 1.2X – 1.9X speedup for pointer chasing operations, +16% database throughput
  - 6% - 41% reduction in energy consumption
- Our solution can be applied to a broad class of in-memory accelerators

Current Investigations

- More efficient address translation and protection mechanisms for PIM

- More concurrent data structures for PIM

More Info on IMPICA (Current Status)

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


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

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

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

Kevin Hsieh

Samira Khan, Nandita Vijaykumar, Kevin K. Chang,

Amirali Boroumand, Saugata Ghose, Onur Mutlu

Carnegie Mellon University

University of Virginia

ETH zürich

SAFARI

We did not cover the following slides in lecture.
These are for your preparation for the next lecture.

Computer Architecture

Lecture 6: Low-Latency DRAM and Processing In Memory

Prof. Onur Mutlu
ETH Zürich

Fall 2017

5 October 2017