# Memory Systems and Memory-Centric Computing Systems

## Part 5: Data-Driven & Data-Aware Arch.

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

3 September 2019

Perugia NiPS Summer School

## The Problem

Computing

is Bottlenecked by Data

## Data Overwhelms Modern Machines …

- Storage/memory capability
- Communication capability
- Computation capability
- Greatly impacts robustness, energy, performance, cost

## Data Movement Overwhelms Modern Machines

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

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

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

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

## Prediction

Future Innovations

Will Be Even More

Bottlenecked by Data

## Axiom

An Intelligent Architecture Handles Data Well

## How to Handle Data Well

- Ensure data does not overwhelm the components
  - via intelligent algorithms
  - via intelligent architectures
  - via whole system designs: algorithm-architecture-devices

- Take advantage of vast amounts of data and metadata
  - to improve architectural & system-level decisions

- Understand and exploit properties of (different) data
  - to improve algorithms & architectures in various metrics

## Corollaries: Architectures Today …

- Architectures are terrible at dealing with data
  - Designed to mainly store and move data vs. to compute
  - They are processor-centric as opposed to **data-centric**

- Architectures are terrible at taking advantage of vast amounts of data (and metadata) available to them
  - Designed to make simple decisions, ignoring lots of data
  - They make human-driven decisions vs. **data-driven** decisions

- Architectures are terrible at knowing and exploiting different properties of application data
  - Designed to treat all data as the same
  - They make component-aware decisions vs. **data-aware**

## Data-Centric (Memory-Centric) Architectures

# Data-Centric Architectures: Properties

- **Process data where it resides** (where it makes sense)
  - Processing in and near memory structures

- **Low-latency & low-energy data access**
  - Low latency memory
  - Low energy memory

- **Low-cost data storage & processing**
  - High capacity memory at low cost: hybrid memory, compression

- **Intelligent data management**
  - Intelligent controllers handling robustness, security, cost, scaling

## Challenge and Opportunity for Future

Computing Architectures
with
Minimal Data Movement

## Challenge and Opportunity for Future

Data-Centric

Computing Architectures

## Exploiting Data to Design Intelligent Architectures

## System Architecture Design Today

- Human-driven
  - Humans design the policies (how to do things)
- Many (too) simple, short-sighted policies all over the system
- No automatic data-driven policy learning
- (Almost) no learning: cannot take lessons from past actions

**Can we design fundamentally intelligent architectures?**

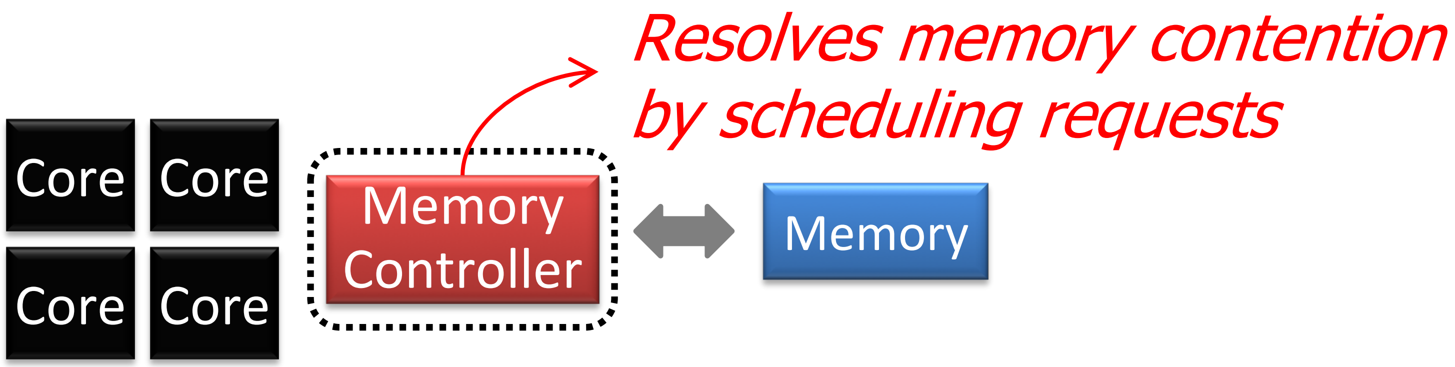
# An Intelligent Architecture

- Data-driven
  - Machine learns the "best" policies (how to do things)
- Sophisticated, workload-driven, changing, far-sighted policies
- Automatic data-driven policy learning
- All controllers are intelligent data-driven agents

**How do we start?**

## Self-Optimizing Memory Controllers

# Memory Controller

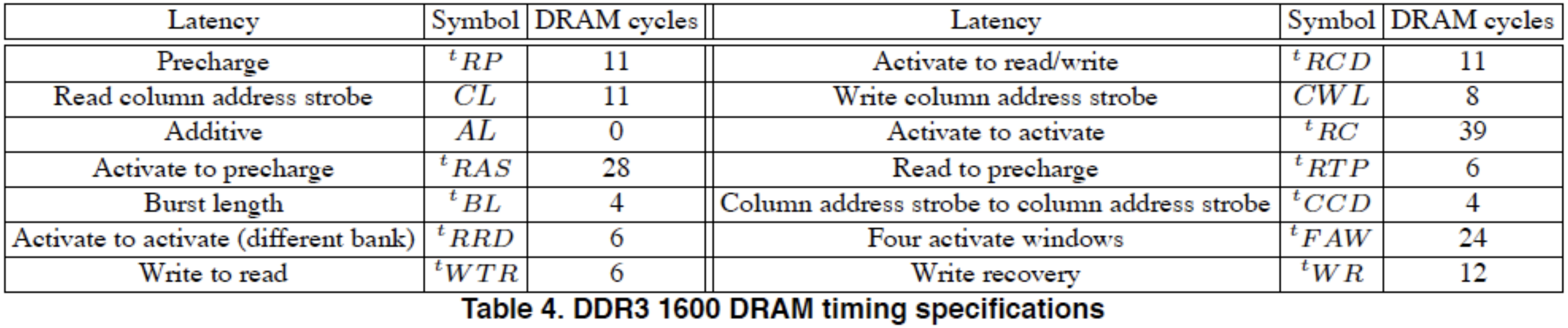


How to schedule requests to maximize system performance?

## Why are Memory Controllers Difficult to Design?

- Need to obey DRAM timing constraints for correctness
  - There are many (50+) timing constraints in DRAM
  - tWTR: Minimum number of cycles to wait before issuing a read command after a write command is issued
  - tRC: Minimum number of cycles between the issuing of two consecutive activate commands to the same bank
  - …
- Need to keep track of many resources to prevent conflicts
  - Channels, banks, ranks, data bus, address bus, row buffers, …
- Need to handle DRAM refresh
- Need to manage power consumption
- Need to optimize performance & QoS (in the presence of constraints)
  - Reordering is not simple
  - Fairness and QoS needs complicates the scheduling problem
- …

# Many Memory Timing Constraints



| Latency | Symbol | DRAM cycles | Latency | Symbol | DRAM cycles |
|---|---|---|---|---|---|
| Precharge | $^tRP$ | 11 | Activate to read/write | $^tRCD$ | 11 |
| Read column address strobe | $CL$ | 11 | Write column address strobe | $CWL$ | 8 |
| Additive | $AL$ | 0 | Activate to activate | $^tRC$ | 39 |
| Activate to precharge | $^tRAS$ | 28 | Read to precharge | $^tRTP$ | 6 |
| Burst length | $^tBL$ | 4 | Column address strobe to column address strobe | $^tCCD$ | 4 |
| Activate to activate (different bank) | $^tRRD$ | 6 | Four activate windows | $^tFAW$ | 24 |
| Write to read | $^tWTR$ | 6 | Write recovery | $^tWR$ | 12 |

**Table 4. DDR3 1600 DRAM timing specifications**

- From Lee et al., "DRAM-Aware Last-Level Cache Writeback: Reducing Write-Caused Interference in Memory Systems," HPS Technical Report, April 2010.

# Many Memory Timing Constraints

- Kim et al., "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.
- Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

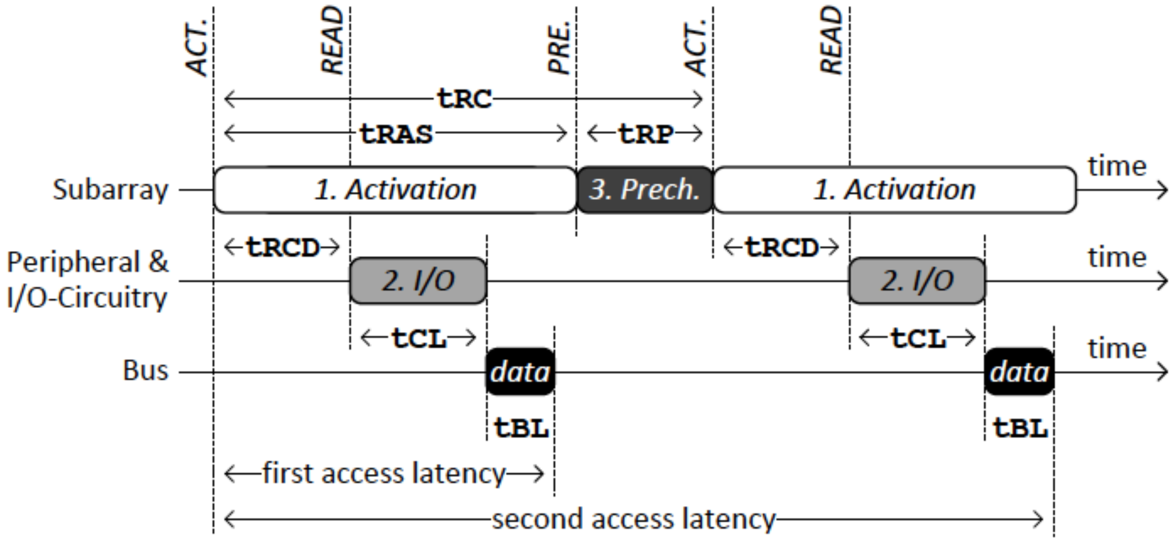


Figure 5. Three Phases of DRAM Access

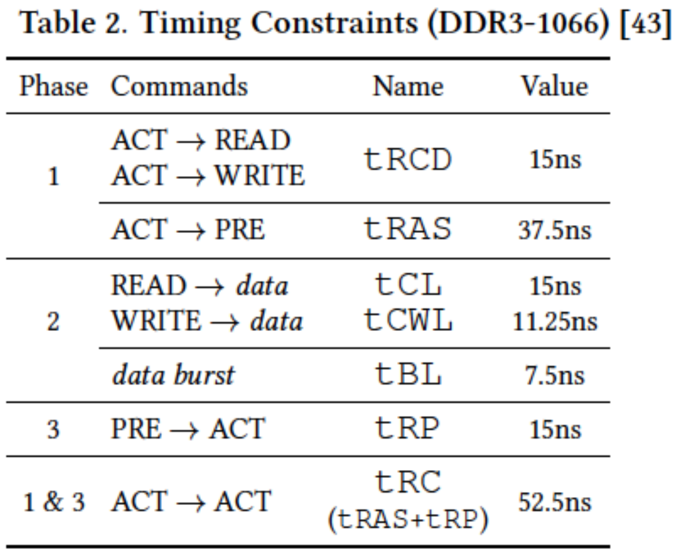


Table 2. Timing Constraints (DDR3-1066) [43]

| Phase | Commands | Name | Value |
|---|---|---|---|
| 1 | ACT → READ<br>ACT → WRITE | tRCD | 15ns |
|  | ACT → PRE | tRAS | 37.5ns |
| 2 | READ → *data*<br>WRITE → *data* | tCL<br>tCWL | 15ns<br>11.25ns |
|  | *data burst* | tBL | 7.5ns |
| 3 | PRE → ACT | tRP | 15ns |
| 1 & 3 | ACT → ACT | tRC<br>(tRAS+tRP) | 52.5ns |

# Why So Many Timing Constraints? (I)

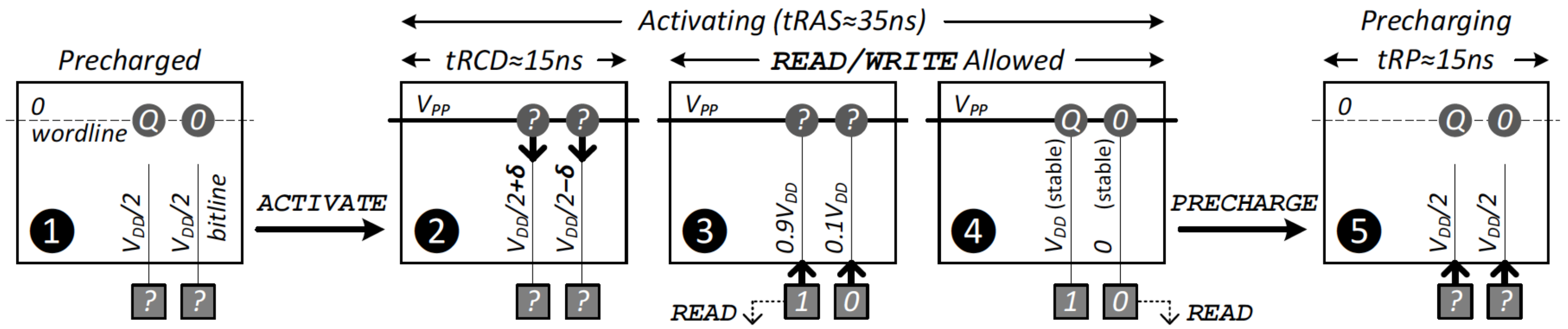


**Figure 4.** DRAM bank operation: Steps involved in serving a memory request [17] $(V_{PP} > V_{DD})$

| Category | RowCmd↔RowCmd | | | RowCmd↔ColCmd | | | ColCmd↔ColCmd | | | ColCmd→DATA | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Name | *tRC* | *tRAS* | *tRP* | *tRCD* | *tRTP* | *tWR** | *tCCD* | *tRTW*† | *tWTR** | *CL* | *CWL* |
| Commands | A→A | A→P | P→A | A→R/W | R→P | W*→P | R(W)→R(W) | R→W | W*→R | R→DATA | W→DATA |
| Scope | Bank | Bank | Bank | Bank | Bank | Bank | Channel | Rank | Rank | Bank | Bank |
| Value (ns) | **∼50** | ∼35 | 13-15 | 13-15 | ∼7.5 | **15** | 5-7.5 | 11-15 | ∼7.5 | 13-15 | 10-15 |

A: ACTIVATE– P: PRECHARGE– R: READ– W: WRITE

∗ Goes into effect after the last write *data*, not from the WRITE command

† Not explicitly specified by the JEDEC DDR3 standard [18]. Defined as a function of other timing constraints.

**Table 1.** Summary of DDR3-SDRAM timing constraints (derived from Micron's 2Gb DDR3-SDRAM datasheet [33])

Kim et al., "A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.

# Why So Many Timing Constraints? (II)

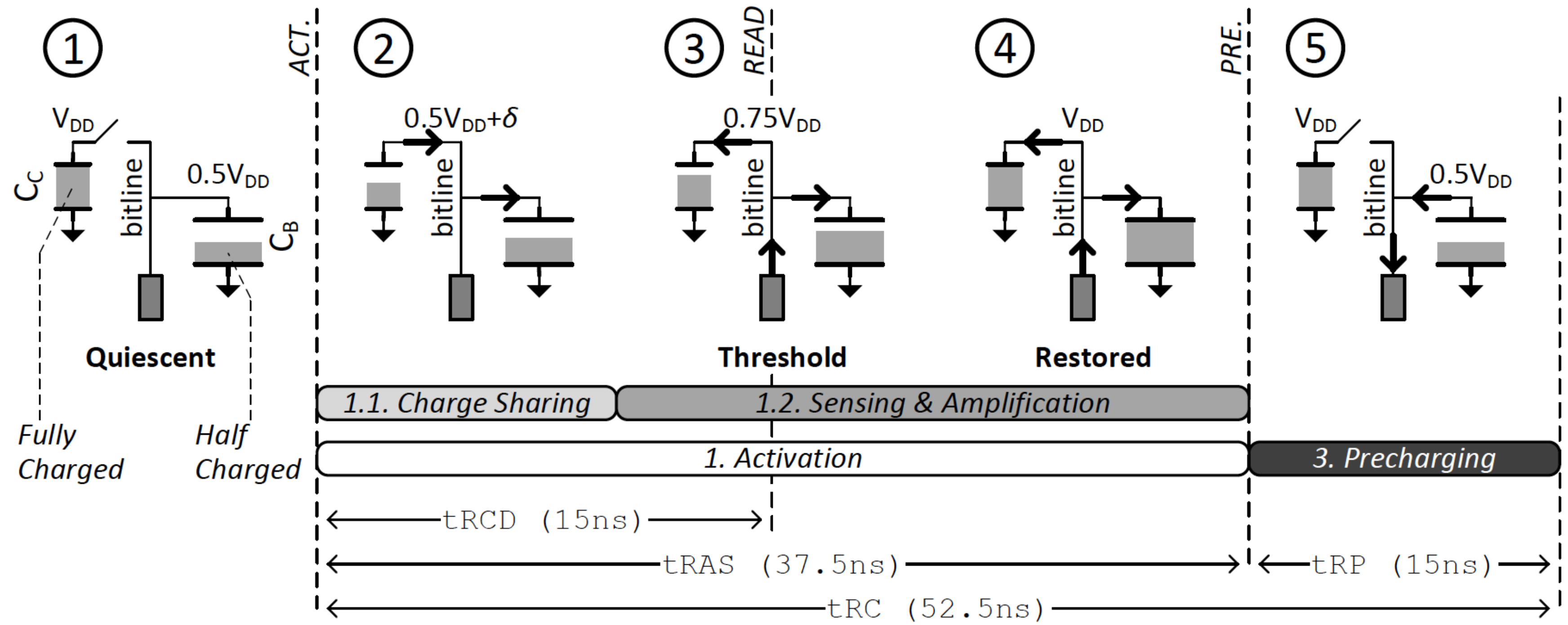


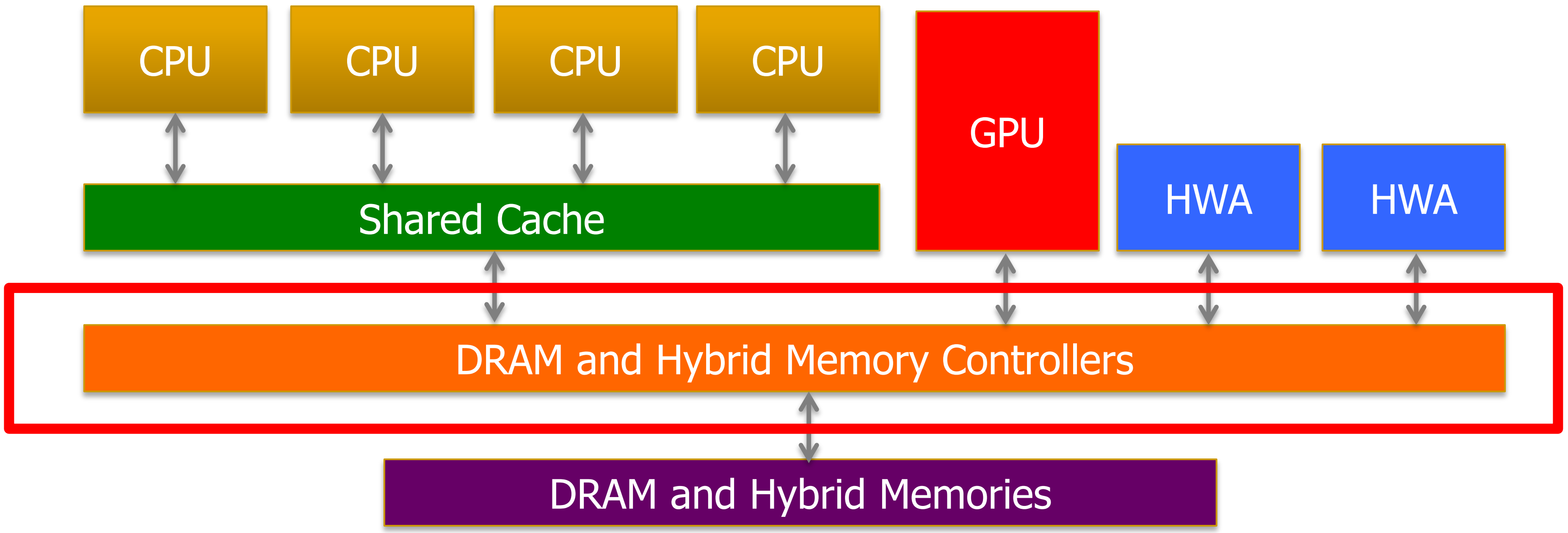
Figure 6. Charge Flow Between the Cell Capacitor ($C_C$), Bitline Parasitic Capacitor ($C_B$), and the Sense-Amplifier ($C_B \approx 3.5C_C$ [39])

Table 2. Timing Constraints (DDR3-1066) [43]

| Phase | Commands | Name | Value |
|---|---|---|---|
| 1 | ACT → READ<br>ACT → WRITE | tRCD | 15ns |
| | ACT → PRE | tRAS | 37.5ns |
| 2 | READ → *data*<br>WRITE → *data* | tCL<br>tCWL | 15ns<br>11.25ns |
| | *data burst* | tBL | 7.5ns |
| 3 | PRE → ACT | tRP | 15ns |
| 1 & 3 | ACT → ACT | tRC<br>(tRAS+tRP) | 52.5ns |

Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.

# Memory Controller Design Is Becoming More Difficult



- Heterogeneous agents: CPUs, GPUs, and HWAs
- Main memory interference between CPUs, GPUs, HWAs
- Many timing constraints for various memory types
- Many goals at the same time: performance, fairness, QoS, energy efficiency, …

## Reality and Dream

- Reality: It difficult to design a policy that maximizes performance, QoS, energy-efficiency, …
  - Too many things to think about
  - Continuously changing workload and system behavior

- Dream: Wouldn't it be nice if the DRAM controller automatically found a good scheduling policy on its own?

## Self-Optimizing DRAM Controllers

- Problem: DRAM controllers are difficult to design
  - It is difficult for human designers to design a policy that can adapt itself very well to different workloads and different system conditions

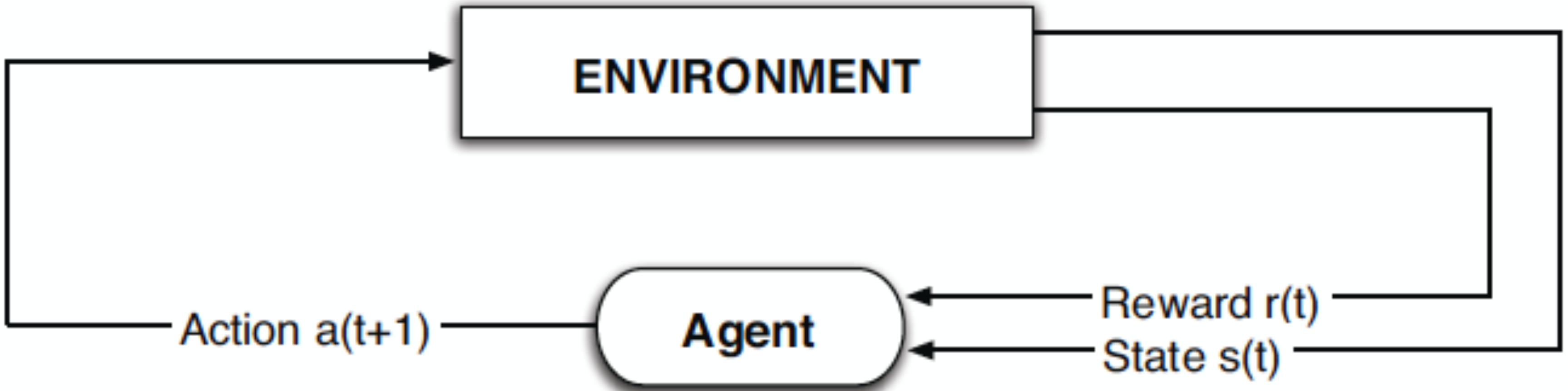
- Idea: A memory controller that adapts its scheduling policy to workload behavior and system conditions using machine learning.

- Observation: Reinforcement learning maps nicely to memory control.

- Design: Memory controller is a reinforcement learning agent
  - It dynamically and continuously learns and employs the best scheduling policy to maximize long-term performance.

Ipek+, "Self Optimizing Memory Controllers: A Reinforcement Learning Approach," ISCA 2008.

## Self-Optimizing DRAM Controllers

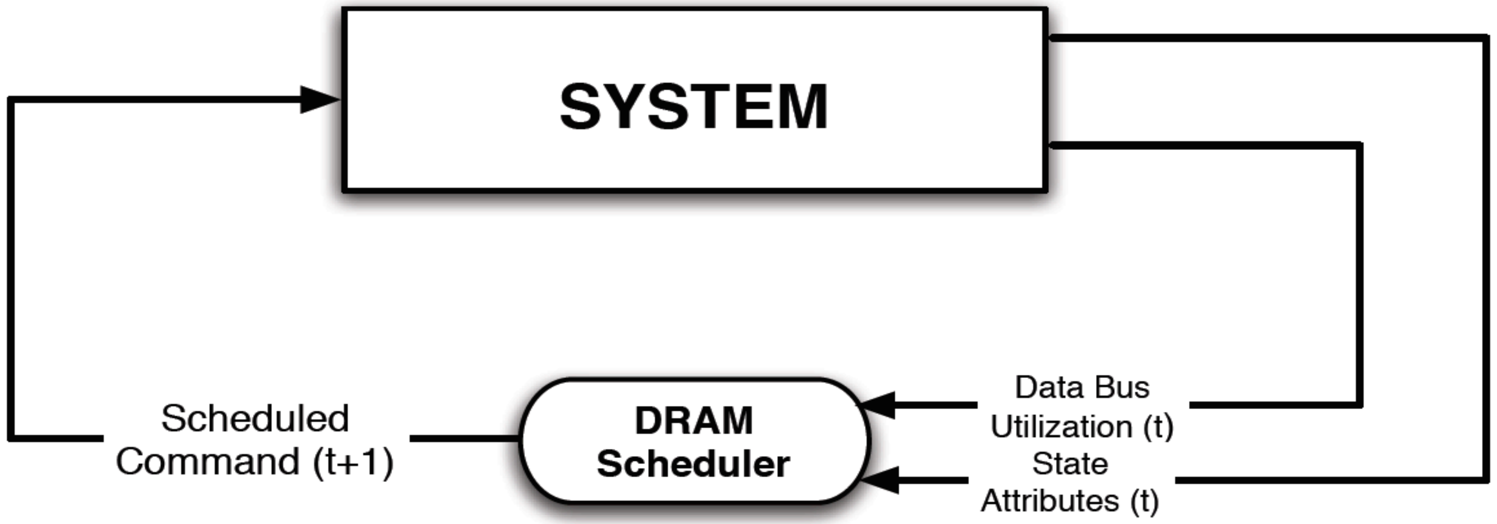


Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \dots$ ( $0 \leq \gamma < 1$)

**Figure 2:** (a) Intelligent agent based on reinforcement learning principles;

# Self-Optimizing DRAM Controllers

- Dynamically adapt the memory scheduling policy via interaction with the system at runtime
    - Associate system states and actions (commands) with long term reward values: each action at a given state leads to a learned reward
    - Schedule command with highest estimated long-term reward value in each state
    - Continuously update reward values for <state, action> pairs based on feedback from system



# Self-Optimizing DRAM Controllers

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

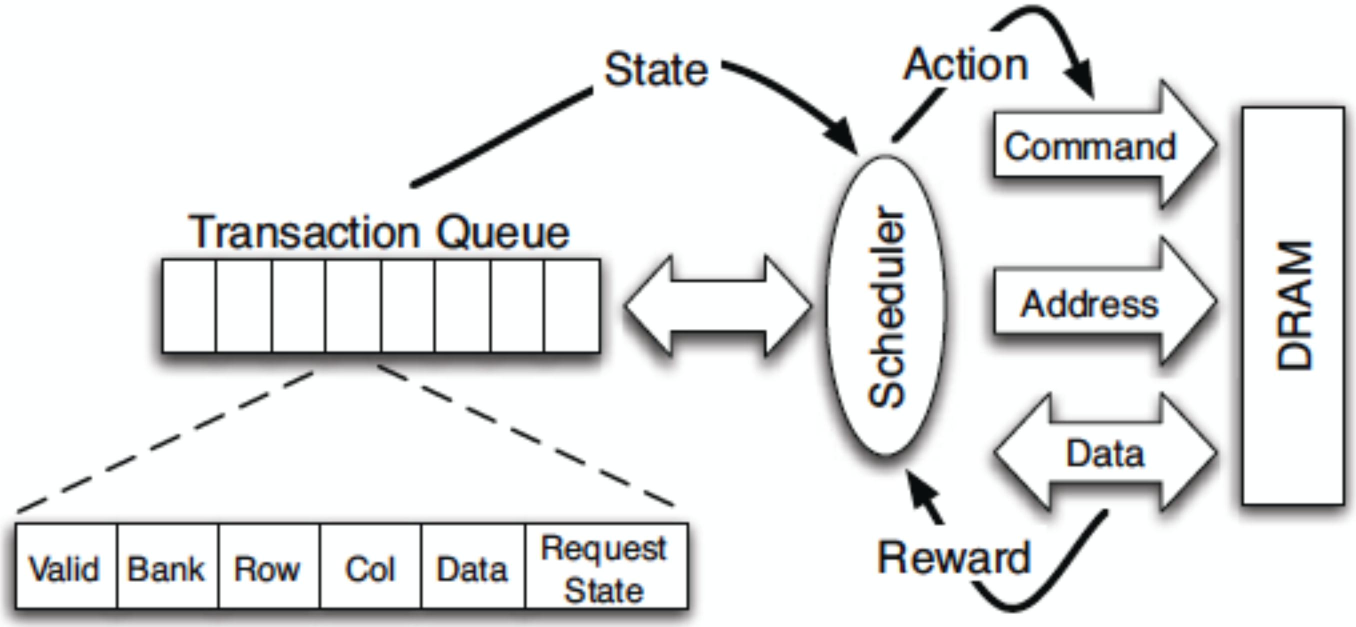


Figure 4: High-level overview of an RL-based scheduler.

## States, Actions, Rewards

- Reward function
  - +1 for scheduling Read and Write commands
  - 0 at all other times

  Goal is to maximize long-term data bus utilization

- State attributes
  - Number of reads, writes, and load misses in transaction queue
  - Number of pending writes and ROB heads waiting for referenced row
  - Request's relative ROB order

- Actions
  - Activate
  - Write
  - Read - load miss
  - Read - store miss
  - Precharge - pending
  - Precharge - preemptive
  - NOP

## Performance Results

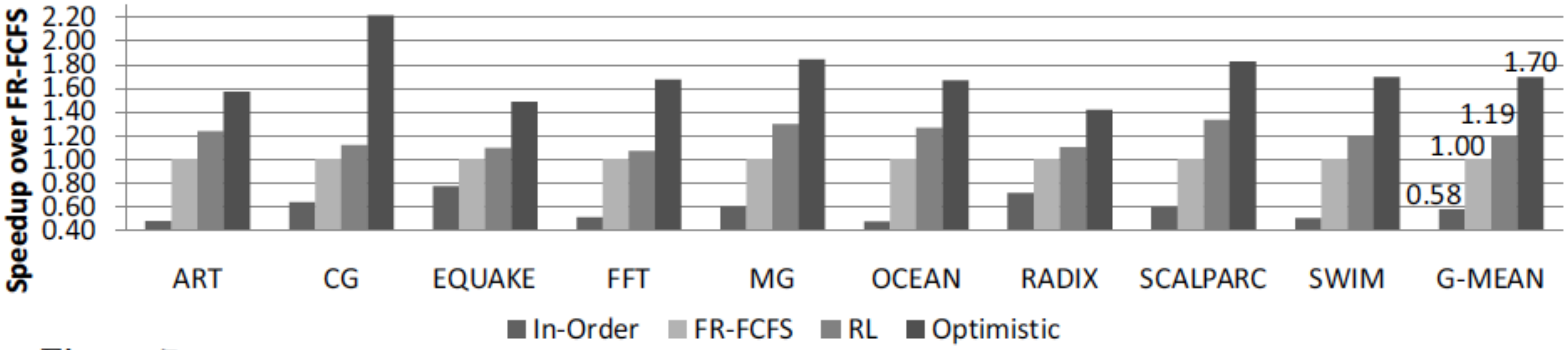


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

**Large, robust performance improvements over many human-designed policies**

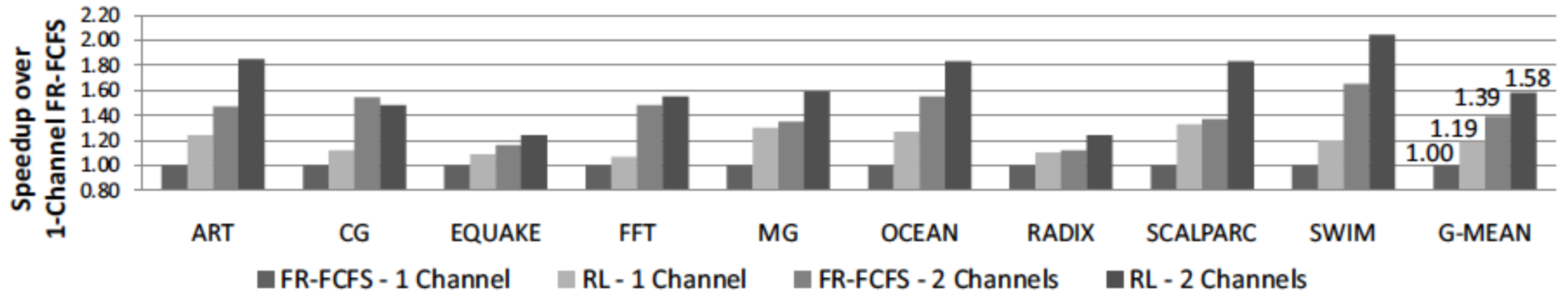


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

# Self Optimizing DRAM Controllers

+ Continuous learning in the presence of changing environment

+ Reduced designer burden in finding a good scheduling policy. Designer specifies:

    1) What system variables might be useful

    2) What target to optimize, but not how to optimize it

-- How to specify different objectives? (e.g., fairness, QoS, …)

-- Hardware complexity?

-- Design **mindset** and flow

## More on Self-Optimizing DRAM Controllers

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

Self-Optimizing Memory Controllers: A Reinforcement Learning Approach

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

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

# An Intelligent Architecture

- Data-driven
  - Machine learns the "best" policies (how to do things)

- Sophisticated, workload-driven, changing, far-sighted policies

- Automatic data-driven policy learning

- All controllers are intelligent data-driven agents

**We need to rethink design**
**(of all controllers)**

## Challenge and Opportunity for Future

Self-Optimizing

(Data-Driven)

Computing Architectures

# Corollaries: Architectures Today …

- Architectures are terrible at dealing with data
  - Designed to mainly store and move data vs. to compute
  - They are processor-centric as opposed to **data-centric**

- Architectures are terrible at taking advantage of vast amounts of data (and metadata) available to them
  - Designed to make simple decisions, ignoring lots of data
  - They make human-driven decisions vs. **data-driven** decisions

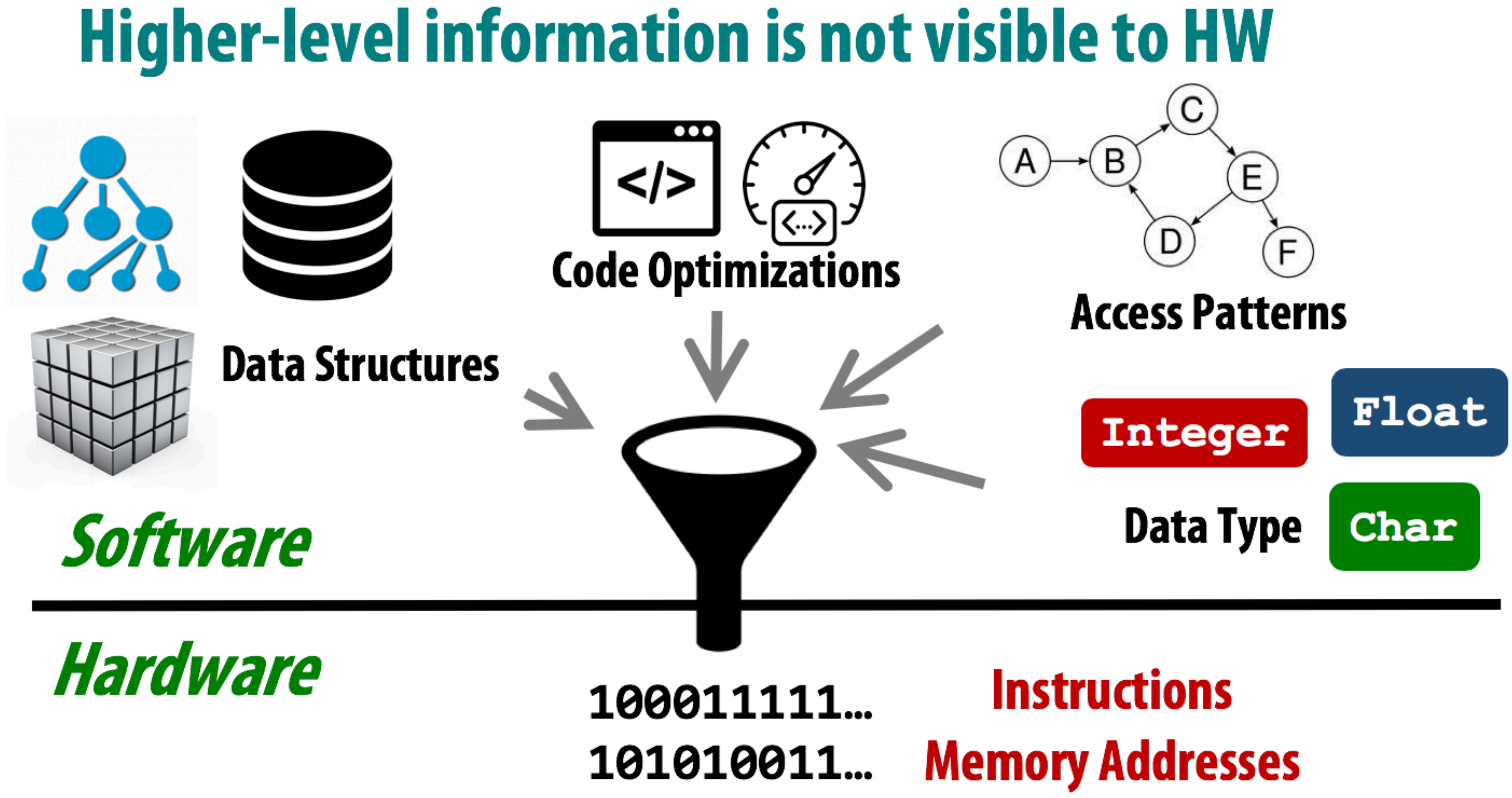
- Architectures are terrible at knowing and exploiting different properties of application data
  - Designed to treat all data as the same
  - They make component-aware decisions vs. **data-aware**

## Data-Aware Architectures

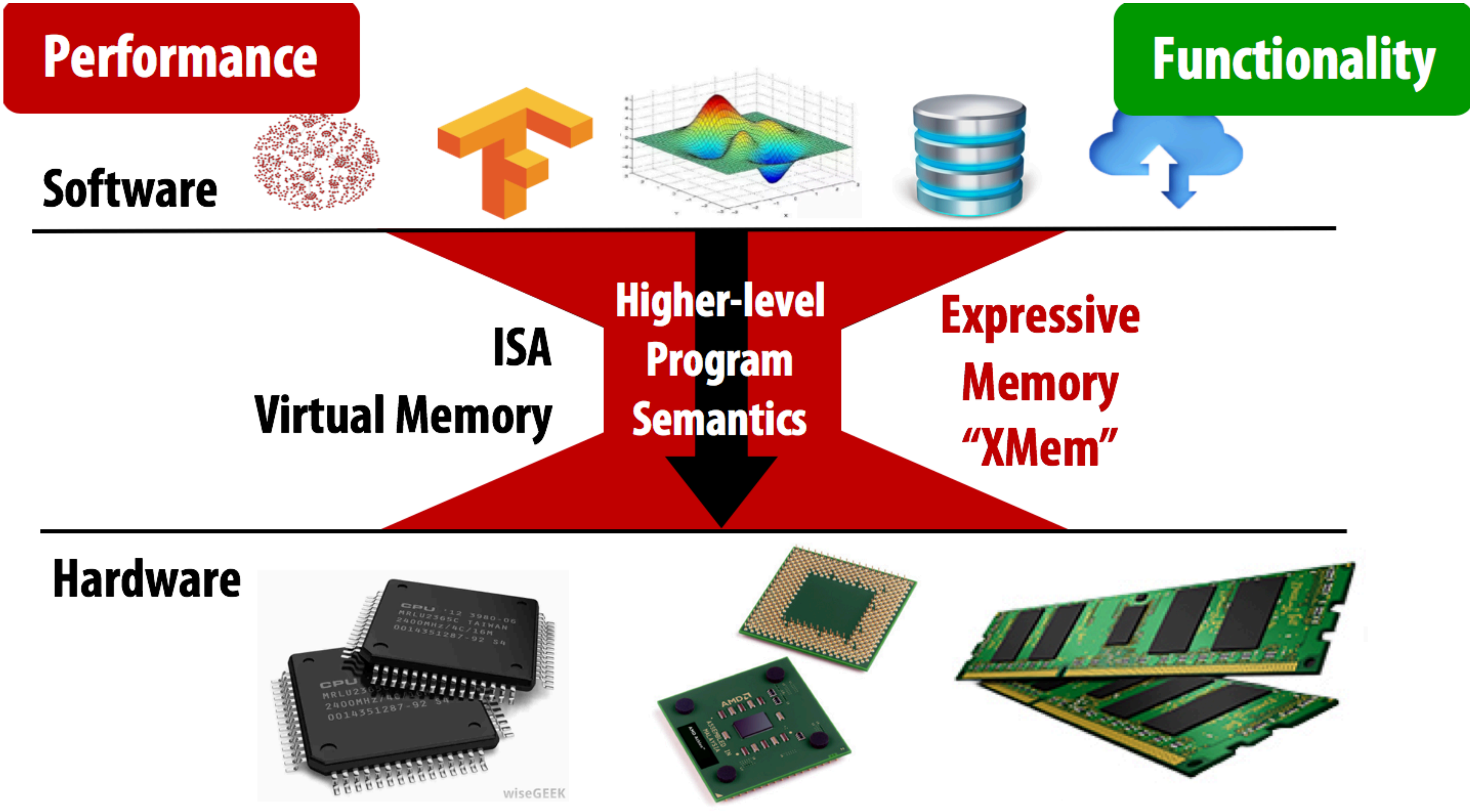
- A data-aware architecture understands what it can do with and to each piece of data

- It makes use of different properties of data to improve performance, efficiency and other metrics
  - Compressibility
  - Approximability
  - Locality
  - Sparsity
  - Criticality for Computation X
  - Access Semantics
  - …

# One Problem: Limited Interfaces

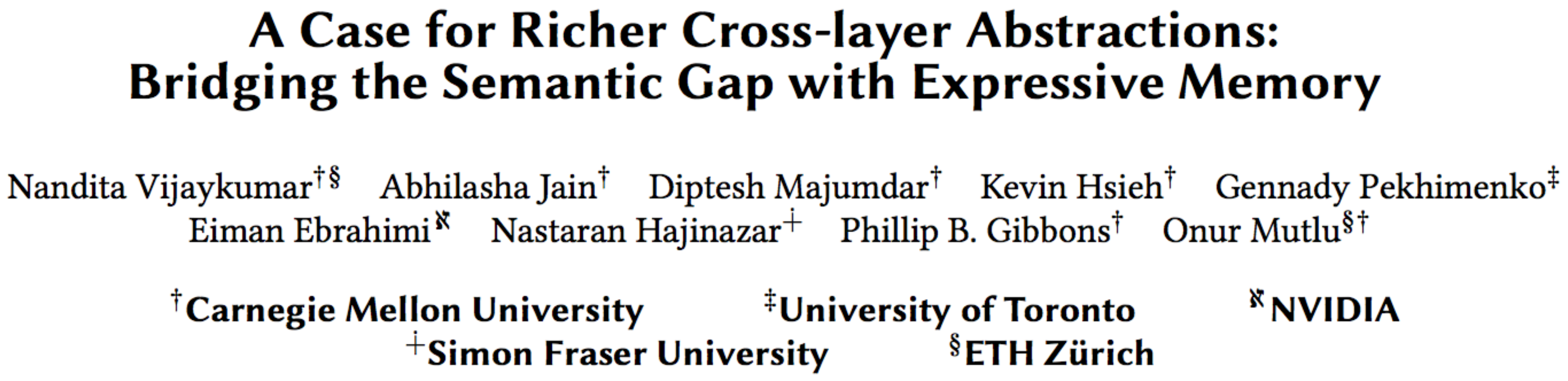


# A Solution: More Expressive Interfaces



# Expressive (Memory) Interfaces

- Nandita Vijaykumar, Abhilasha Jain, Diptesh Majumdar, Kevin Hsieh, Gennady Pekhimenko, Eiman Ebrahimi, Nastaran Hajinazar, Phillip B. Gibbons and Onur Mutlu,
  **"A Case for Richer Cross-layer Abstractions: Bridging the Semantic Gap with Expressive Memory"**
  *Proceedings of the 45th International Symposium on Computer Architecture* (***ISCA***), Los Angeles, CA, USA, June 2018.
  [Slides (pptx) (pdf)] [Lightning Talk Slides (pptx) (pdf)]
  [Lightning Talk Video]



**A Case for Richer Cross-layer Abstractions: Bridging the Semantic Gap with Expressive Memory**

Nandita Vijaykumar[†§] Abhilasha Jain[†] Diptesh Majumdar[†] Kevin Hsieh[†] Gennady Pekhimenko[‡]
Eiman Ebrahimi[ℵ] Nastaran Hajinazar[+] Phillip B. Gibbons[†] Onur Mutlu[§†]

[†]Carnegie Mellon University [‡]University of Toronto [ℵ]NVIDIA
[+]Simon Fraser University [§]ETH Zürich

# X-MeM Aids Many Optimizations

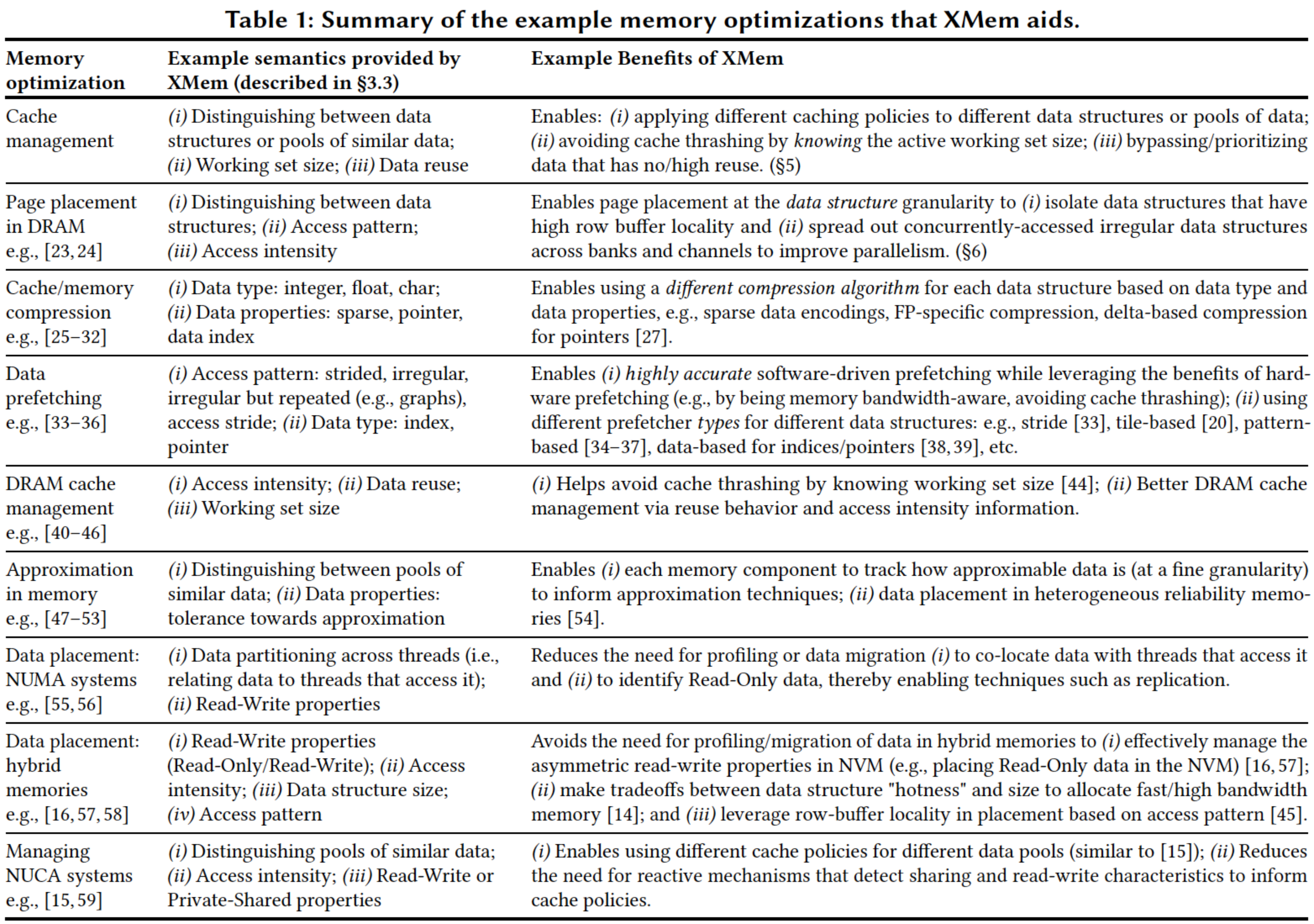
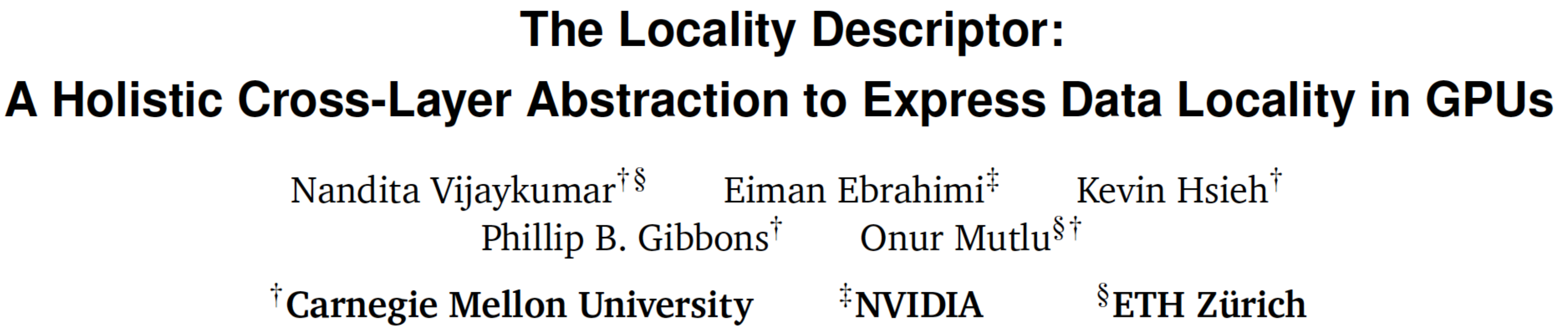


**Table 1: Summary of the example memory optimizations that XMem aids.**

| **Memory optimization** | **Example semantics provided by XMem (described in §3.3)** | **Example Benefits of XMem** |
|---|---|---|
| Cache management | *(i)* Distinguishing between data structures or pools of similar data; *(ii)* Working set size; *(iii)* Data reuse | Enables: *(i)* applying different caching policies to different data structures or pools of data; *(ii)* avoiding cache thrashing by *knowing* the active working set size; *(iii)* bypassing/prioritizing data that has no/high reuse. (§5) |
| Page placement in DRAM e.g., [23, 24] | *(i)* Distinguishing between data structures; *(ii)* Access pattern; *(iii)* Access intensity | Enables page placement at the *data structure* granularity to *(i)* isolate data structures that have high row buffer locality and *(ii)* spread out concurrently-accessed irregular data structures across banks and channels to improve parallelism. (§6) |
| Cache/memory compression e.g., [25–32] | *(i)* Data type: integer, float, char; *(ii)* Data properties: sparse, pointer, data index | Enables using a *different compression algorithm* for each data structure based on data type and data properties, e.g., sparse data encodings, FP-specific compression, delta-based compression for pointers [27]. |
| Data prefetching e.g., [33–36] | *(i)* Access pattern: strided, irregular, irregular but repeated (e.g., graphs), access stride; *(ii)* Data type: index, pointer | Enables *(i) highly accurate* software-driven prefetching while leveraging the benefits of hardware prefetching (e.g., by being memory bandwidth-aware, avoiding cache thrashing); *(ii)* using different prefetcher *types* for different data structures: e.g., stride [33], tile-based [20], pattern-based [34–37], data-based for indices/pointers [38, 39], etc. |
| DRAM cache management e.g., [40–46] | *(i)* Access intensity; *(ii)* Data reuse; *(iii)* Working set size | *(i)* Helps avoid cache thrashing by knowing working set size [44]; *(ii)* Better DRAM cache management via reuse behavior and access intensity information. |
| Approximation in memory e.g., [47–53] | *(i)* Distinguishing between pools of similar data; *(ii)* Data properties: tolerance towards approximation | Enables *(i)* each memory component to track how approximable data is (at a fine granularity) to inform approximation techniques; *(ii)* data placement in heterogeneous reliability memories [54]. |
| Data placement: NUMA systems e.g., [55, 56] | *(i)* Data partitioning across threads (i.e., relating data to threads that access it); *(ii)* Read-Write properties | Reduces the need for profiling or data migration *(i)* to co-locate data with threads that access it and *(ii)* to identify Read-Only data, thereby enabling techniques such as replication. |
| Data placement: hybrid memories e.g., [16, 57, 58] | *(i)* Read-Write properties (Read-Only/Read-Write); *(ii)* Access intensity; *(iii)* Data structure size; *(iv)* Access pattern | Avoids the need for profiling/migration of data in hybrid memories to *(i)* effectively manage the asymmetric read-write properties in NVM (e.g., placing Read-Only data in the NVM) [16, 57]; *(ii)* make tradeoffs between data structure "hotness" and size to allocate fast/high bandwidth memory [14]; and *(iii)* leverage row-buffer locality in placement based on access pattern [45]. |
| Managing NUCA systems e.g., [15, 59] | *(i)* Distinguishing pools of similar data; *(ii)* Access intensity; *(iii)* Read-Write or Private-Shared properties | *(i)* Enables using different cache policies for different data pools (similar to [15]); *(ii)* Reduces the need for reactive mechanisms that detect sharing and read-write characteristics to inform cache policies. |

# Expressive (Memory) Interfaces for GPUs

- Nandita Vijaykumar, Eiman Ebrahimi, Kevin Hsieh, Phillip B. Gibbons and Onur Mutlu,
  **"The Locality Descriptor: A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs"**
  *Proceedings of the 45th International Symposium on Computer Architecture* (***ISCA***), Los Angeles, CA, USA, June 2018.
  [Slides (pptx) (pdf)] [Lightning Talk Slides (pptx) (pdf)]
  [Lightning Talk Video]

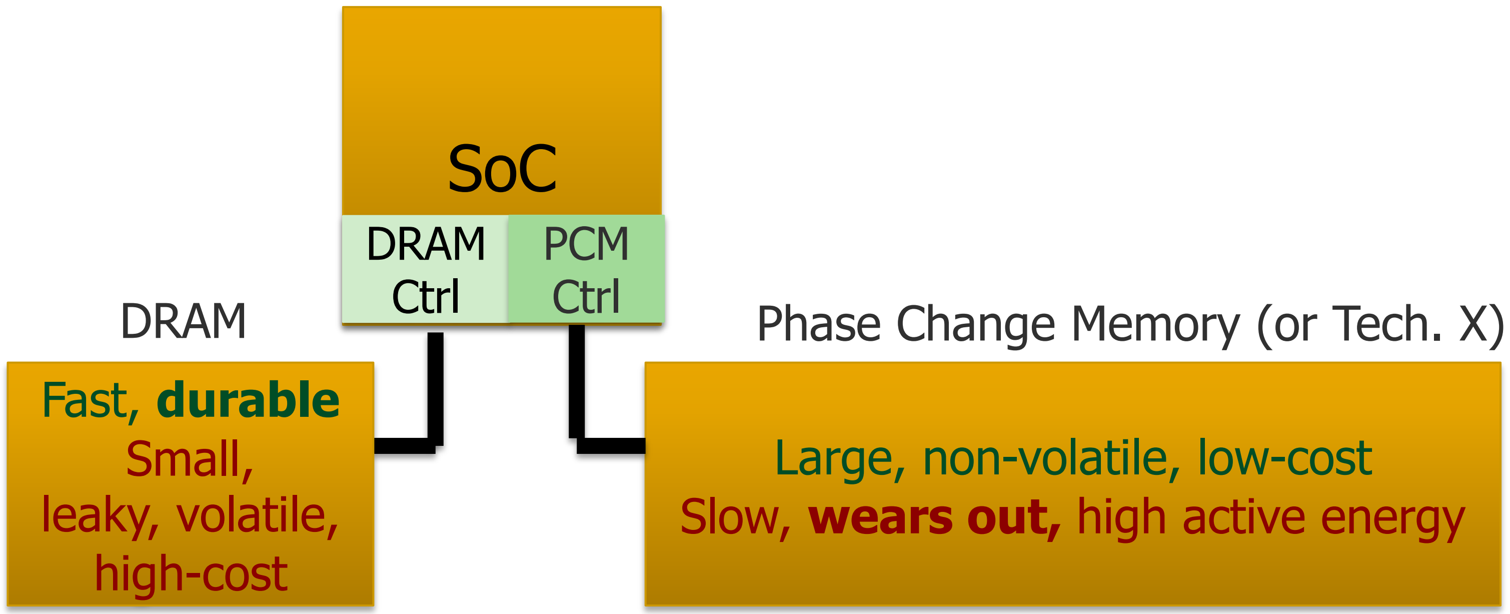


**The Locality Descriptor:**
**A Holistic Cross-Layer Abstraction to Express Data Locality in GPUs**

Nandita Vijaykumar[†§] Eiman Ebrahimi[‡] Kevin Hsieh[†]
Phillip B. Gibbons[†] Onur Mutlu[§†]

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

# An Example: Hybrid Memory Management


SoC
DRAM Ctrl
PCM Ctrl
DRAM
Fast, **durable**
Small, leaky, volatile, high-cost
Phase Change Memory (or Tech. X)
Large, non-volatile, low-cost
Slow, **wears out,** high active energy

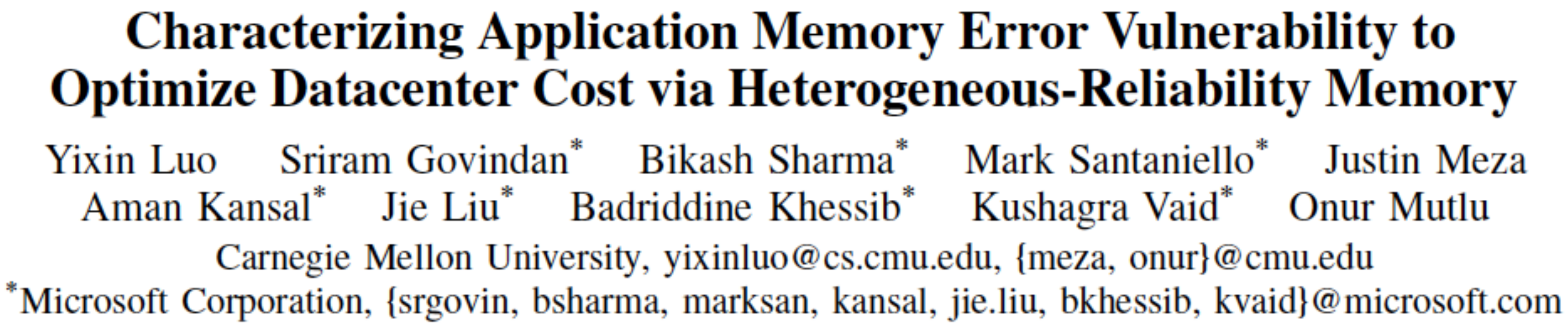
Hardware/software manage data allocation and movement
to achieve the best of multiple technologies

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.

## An Example: Heterogeneous-Reliability Memory

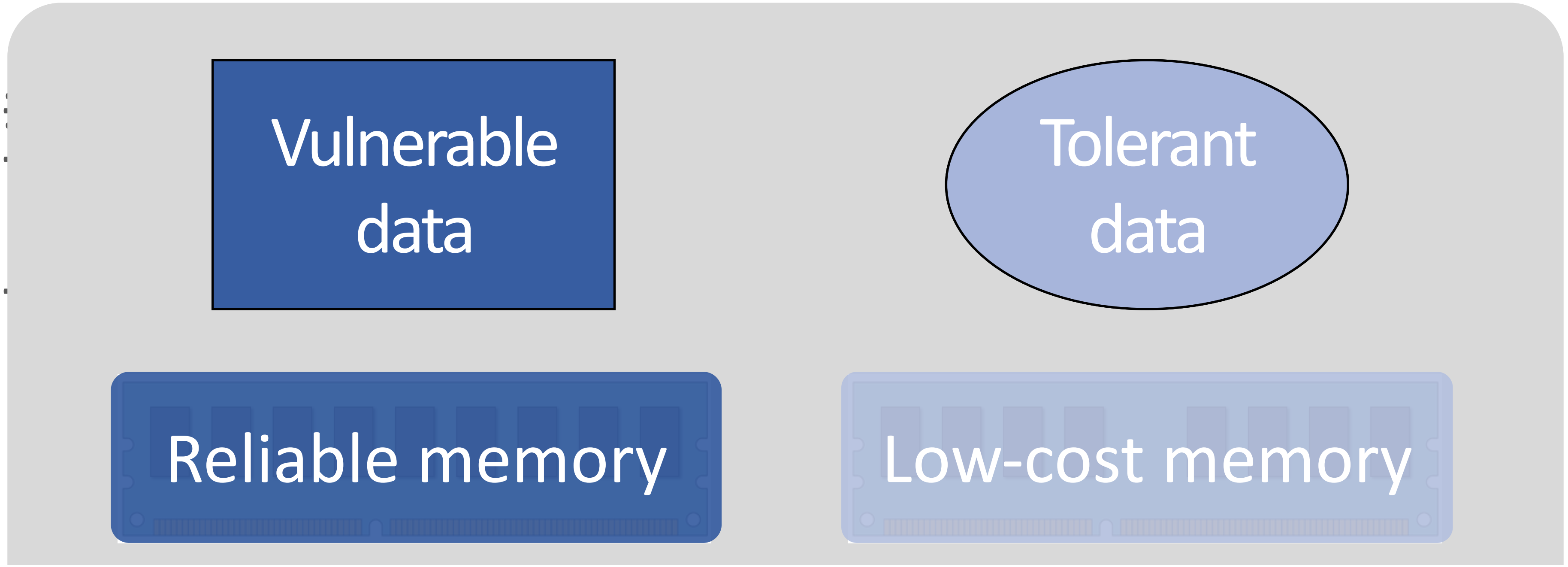
- Yixin Luo, Sriram Govindan, Bikash Sharma, Mark Santaniello, Justin Meza, Aman Kansal, Jie Liu, Badriddine Khessib, Kushagra Vaid, and Onur Mutlu, **"Characterizing Application Memory Error Vulnerability to Optimize Data Center Cost via Heterogeneous-Reliability Memory"** *Proceedings of the 44th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**)*, Atlanta, GA, June 2014. [Summary] [Slides (pptx) (pdf)] [Coverage on ZDNet]



**Characterizing Application Memory Error Vulnerability to Optimize Datacenter Cost via Heterogeneous-Reliability Memory**

Yixin Luo Sriram Govindan* Bikash Sharma* Mark Santaniello* Justin Meza
Aman Kansal* Jie Liu* Badriddine Khessib* Kushagra Vaid* Onur Mutlu
Carnegie Mellon University, yixinluo@cs.cmu.edu, {meza, onur}@cmu.edu
*Microsoft Corporation, {srgovin, bsharma, marksan, kansal, jie.liu, bkhessib, kvaid}@microsoft.com

# 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 %

**H**eterogeneous-**R**eliability **M**emory [DSN 2014]

## Challenge and Opportunity for Future

Data-Aware

(Expressive)

Computing Architectures

## Recap: Corollaries: Architectures Today

- Architectures are terrible at dealing with data
  - Designed to mainly store and move data vs. to compute
  - They are processor-centric as opposed to data-centric

- Architectures are terrible at taking advantage of vast amounts of data (and metadata) available to them
  - Designed to make simple decisions, ignoring lots of data
  - They make human-driven decisions vs. data-driven decisions

- Architectures are terrible at knowing and exploiting different properties of application data
  - Designed to treat all data as the same
  - They make component-aware decisions vs. data-aware

## Architectures for Intelligent Machines

**Data-centric**

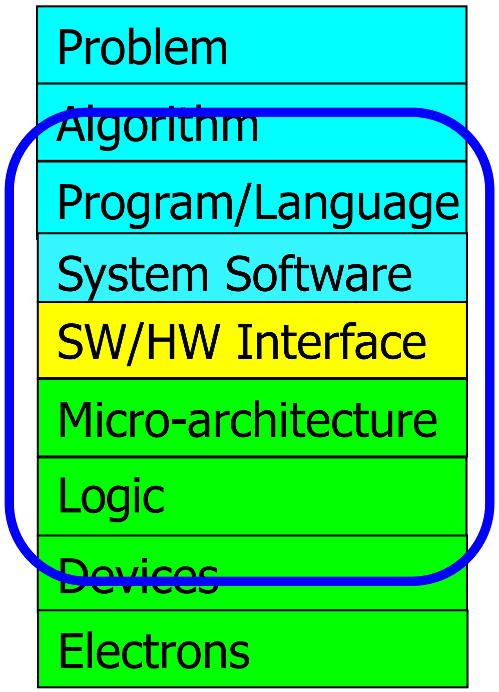
**Data-driven**

**Data-aware**

SAFARI

Source: http://spectrum.ieee.org/image/MjYzMzAyMg.jpeg

## We Need to Think Across the Entire Stack



**We can get there step by step**

# Memory Systems and Memory-Centric Computing Systems

## Part 5: Data-Driven & Data-Aware Arch.

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

3 September 2019

Perugia NiPS Summer School