Rethinking the Systems We Design

Onur Mutlu
onur@cmu.edu
September 19, 2014
Yale @ 75

Carnegie Mellon
SAFARI

Agenda

- Principled Computer Architecture/System Design
- How We Violate Those Principles Today
- Some Solution Approaches
- Concluding Remarks

First, Let's Start With …

- The Real Reason We Are Here Today
- Yale @ 35

Some Teachings of Yale Patt

THE UNIVERSITY OF TEXAS AT AUSTIN
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

EE 382N: Microarchitecture

Spring 2014

Welcome to EE 382N - Microarchitecture

The University of Texas at Austin

Spring 2002

Design Principles

- ***Critical path design***
- ***Bread and Butter design***
- ***Balanced design***

from Yale Patt's EE 382N lecture notes

(Micro)architecture Design Principles

- Bread and butter design
  - Spend time and resources on where it matters (i.e. improving what the machine is designed to do)
  - Common case vs. uncommon case
- Balanced design
  - Balance instruction/data flow through uarch components
  - Design to eliminate bottlenecks
- Critical path design
  - Find the maximum speed path and decrease it
    - Break a path into multiple cycles?

from my ECE 740 lecture notes

My Takeaways

- Quite reasonable principles
- Stated by other principled thinkers in similar or different ways
  - E.g., Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966
  - E.g., Gene M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," AFIPS Conference, April 1967.
  - E.g., Butler W. Lampson, "Hints for Computer System Design," SOSP 1983.
  - …
- Will take the liberty to generalize them in the rest of the talk

The Problem

- Systems designed today violate these principles

- Some system *components* individually might not (or might seem not to) violate the principles

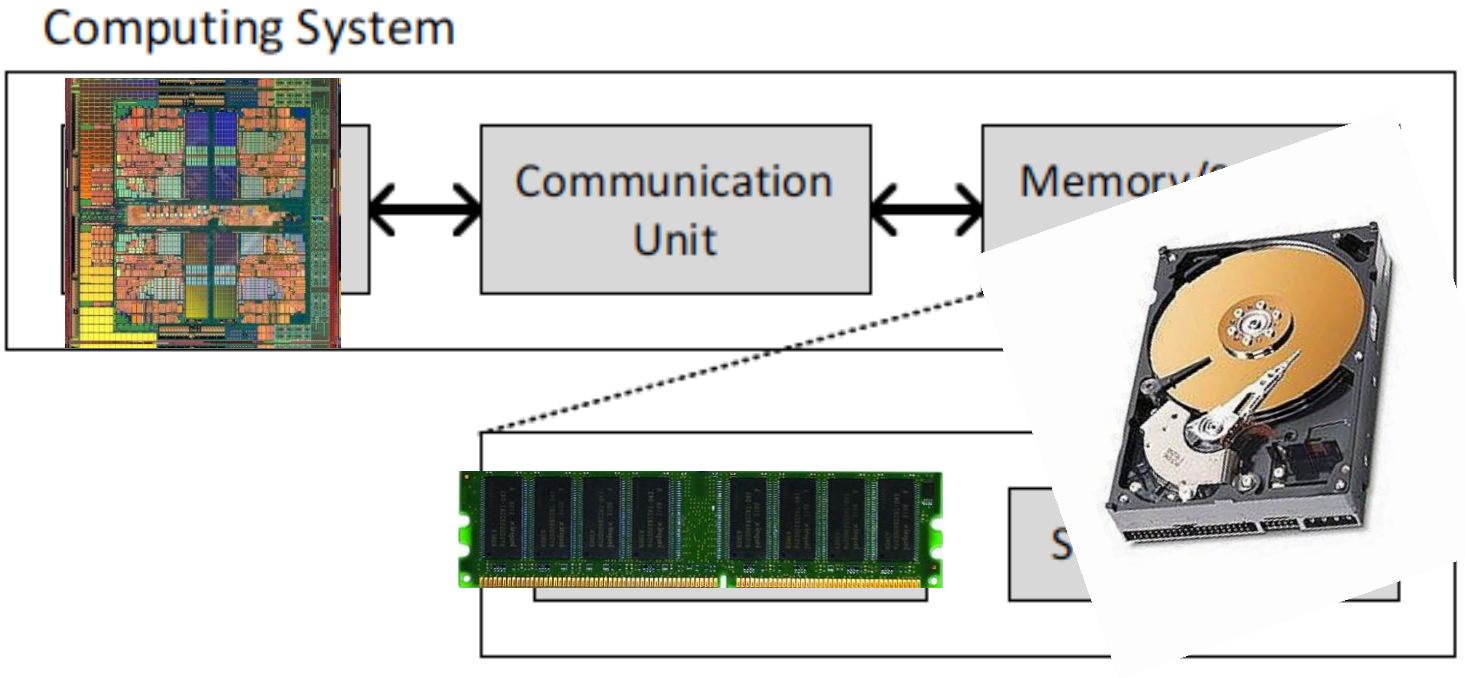
- But the overall system
  - Does not spend time or resources where it matters
  - Is grossly imbalanced
  - Does not optimize for the critical work/application

Agenda

- Principled Computer Architecture/System Design
- How We Violate Those Principles Today
- Some Solution Approaches
- Concluding Remarks

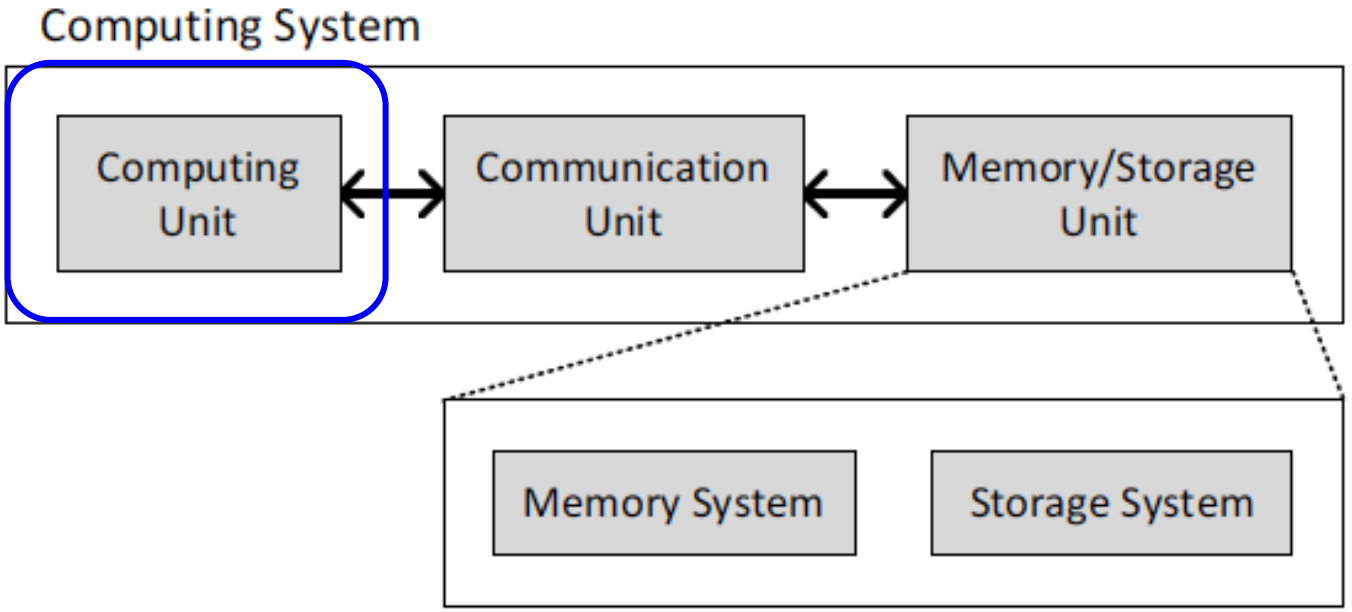
A Computing System

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



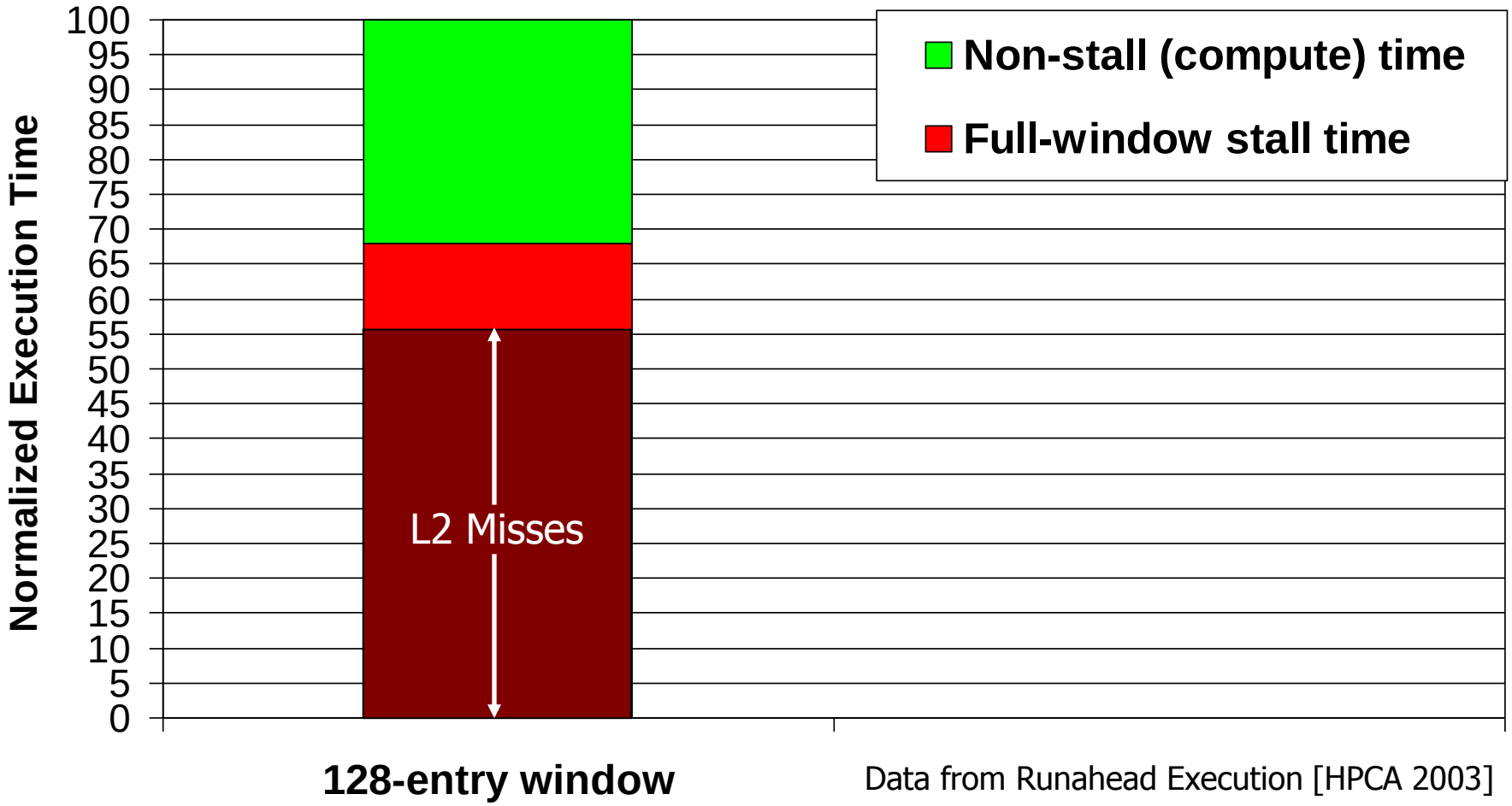
Today's Systems

- Are overwhelmingly processor centric
- Processor is heavily optimized and is considered the master
- Many system-level tradeoffs are constrained or dictated by the processor – all data processed in the processor
- Data storage units are dumb slaves and are largely unoptimized (except for some that are on the processor die)



Yet …

- "It's the memory, stupid" (Anonymous DEC engineer)



Data from Runahead Execution [HPCA 2003]

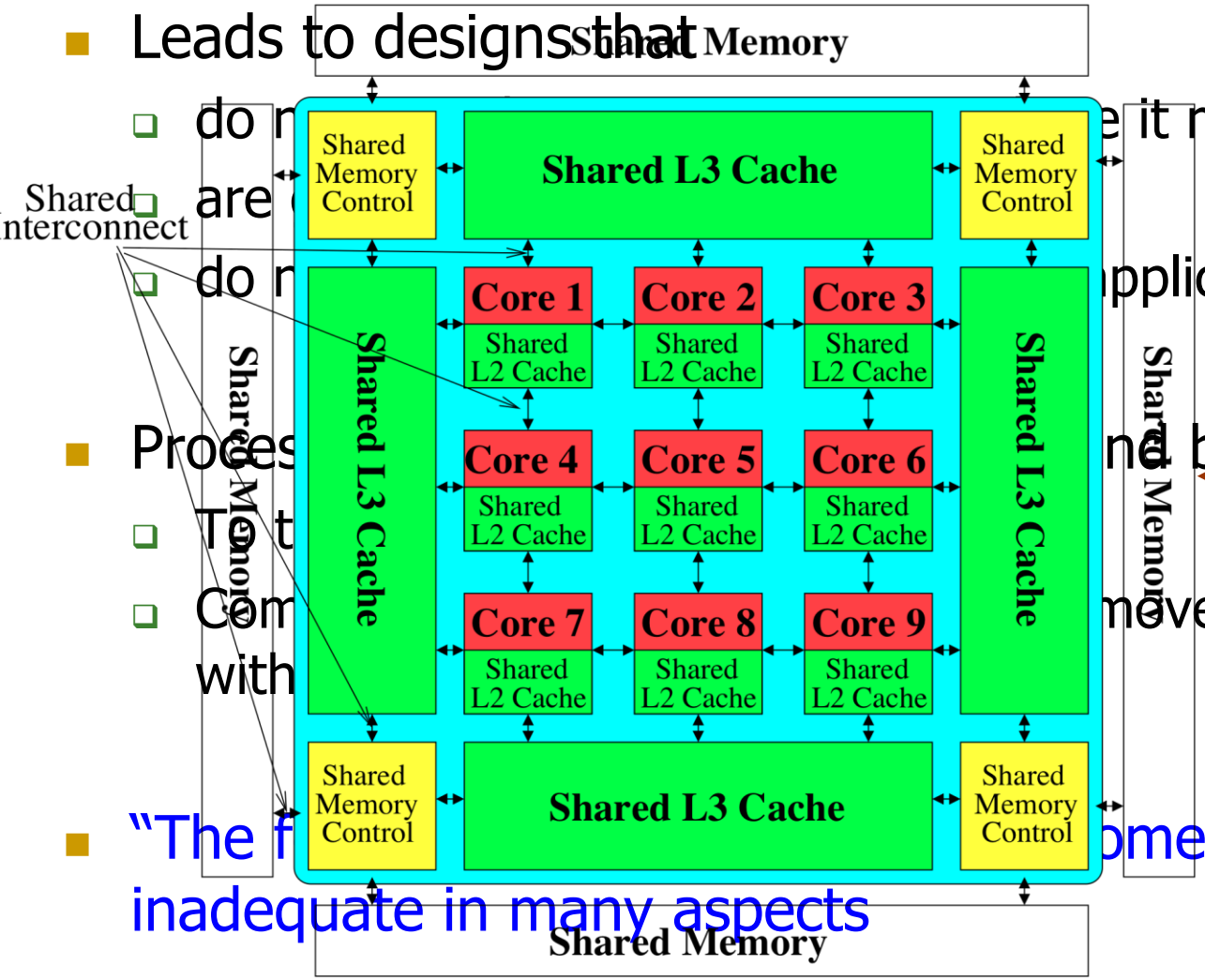
Yet …

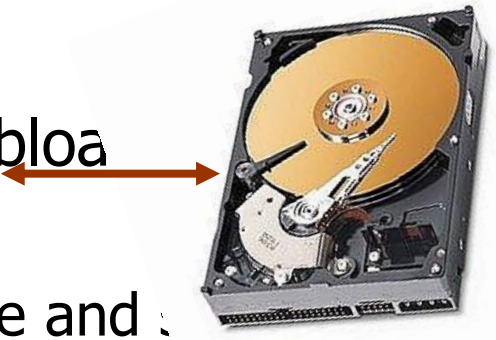
- Memory system is the major performance, energy, QoS/predictability and reliability bottleneck in many (most?) workloads

- And, it is becoming increasingly so
  - Increasing hunger for more data and its (fast) analysis
  - Demand to pack and consolidate more on-chip for efficiency
  - Memory bandwidth and capacity not scaling as fast as demand
  - Demand to guarantee SLAs, QoS, user satisfaction
  - DRAM technology is not scaling well to smaller feature sizes, exacerbating energy, reliability, capacity, bandwidth problems

This Processor-Memory Disparity

- Leads to designs that
  - do n… e it matters
  - are …
  - do n… pplication
- Proces… nd bloa
  - To t…
  - Com… move and …
    with
- "The f… omes dumb and inadequate in many aspects

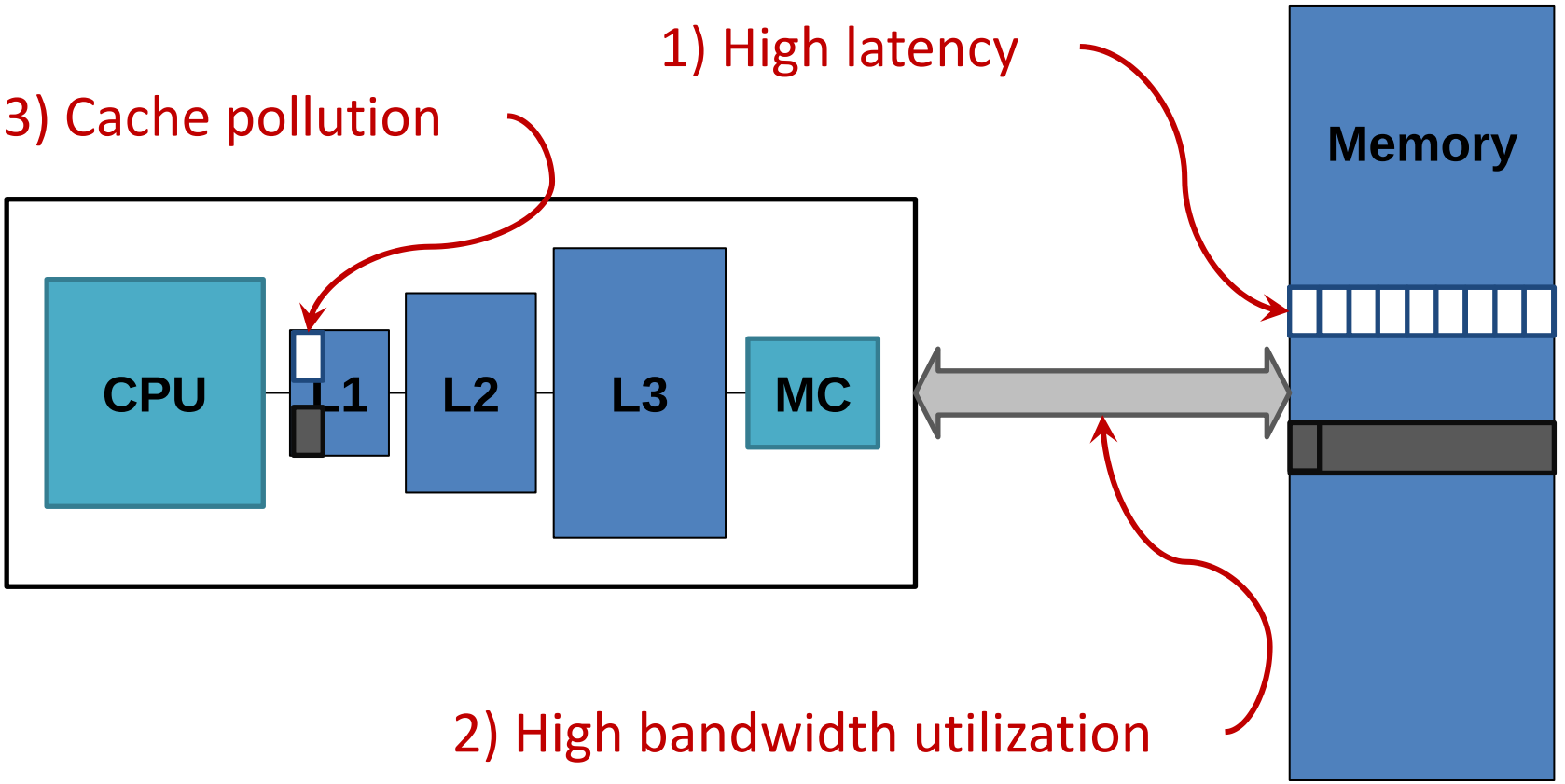




Several Examples

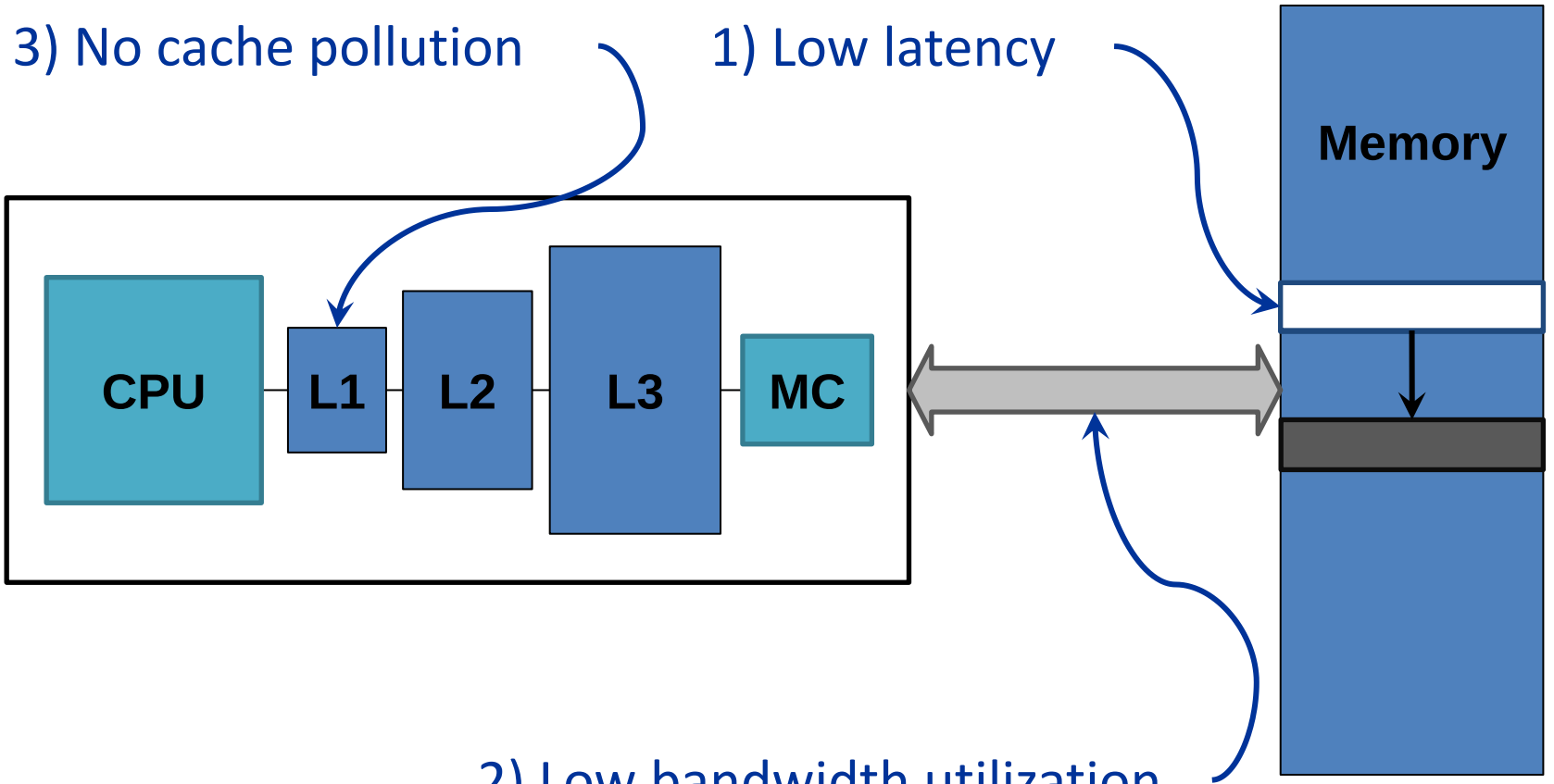
- Bulk data copy (and initialization)
- DRAM refresh
- Memory reliability
- Disparity of working memory and persistent storage
- Homogeneous memory
- Predictable performance and fairness in memory

Today's Memory: Bulk Data Copy



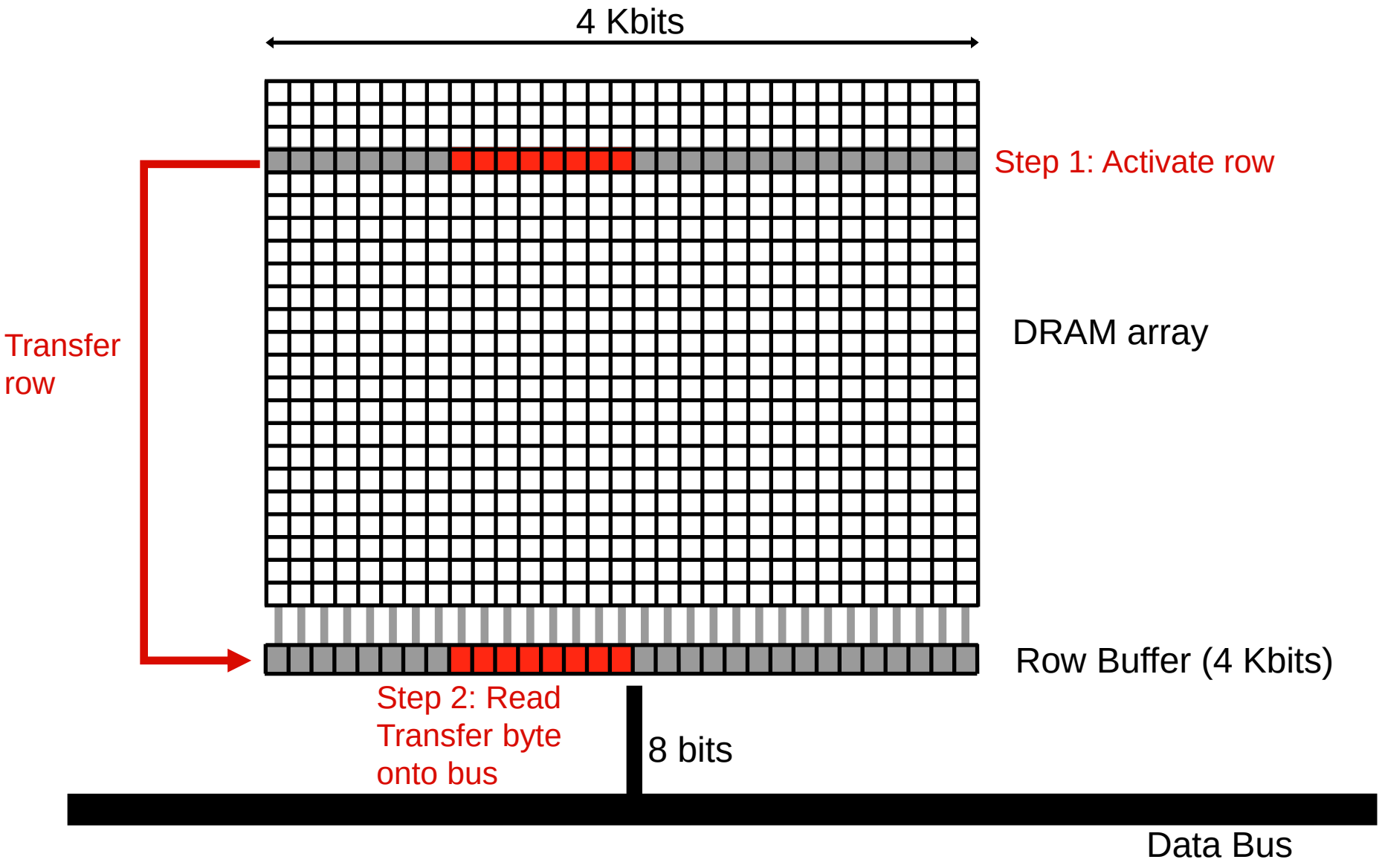
1046ns, 3.6uJ (for 4KB page copy via DMA)

Future: RowClone (In-Memory Copy)

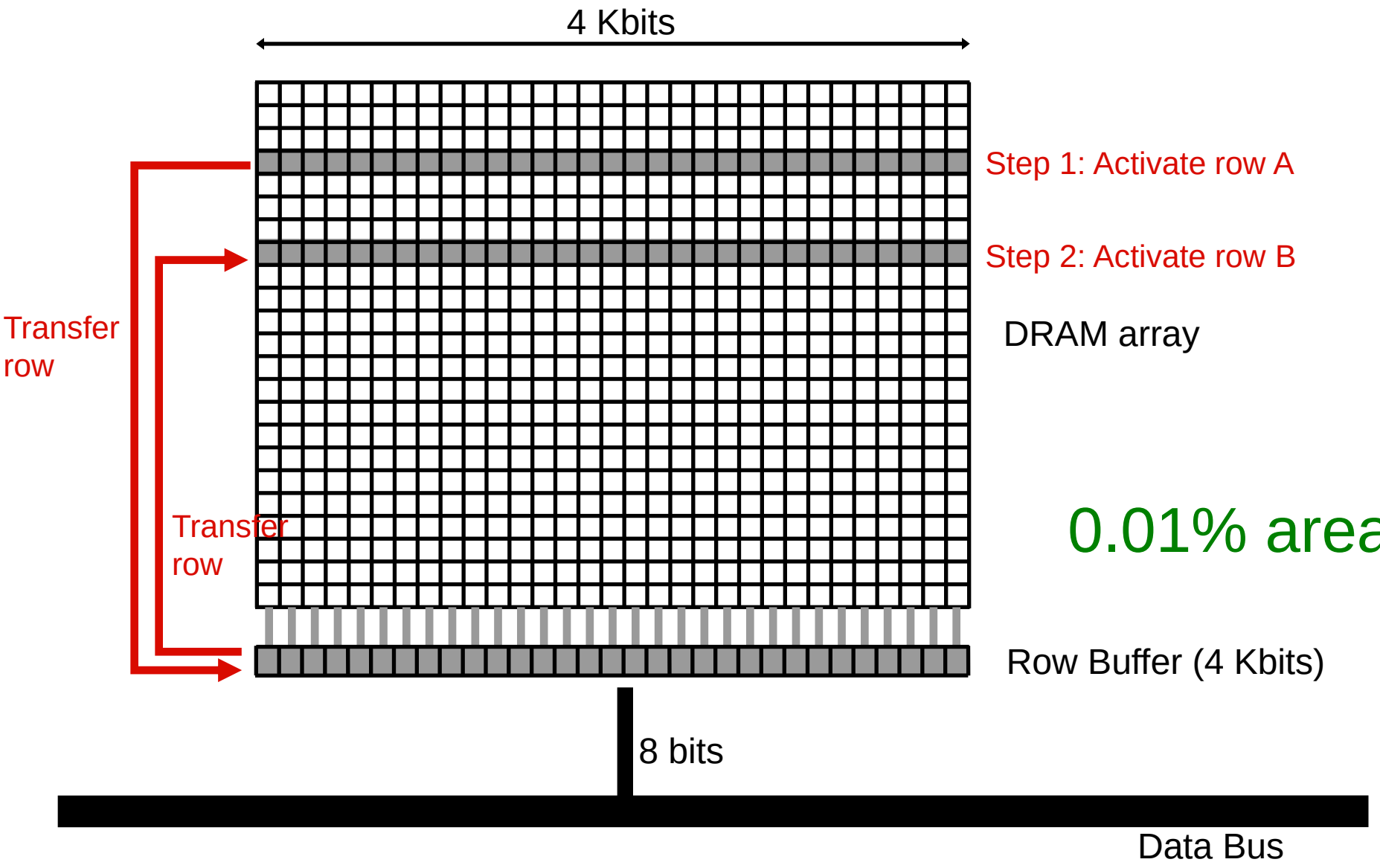


1046ns, 3.6uJ

DRAM Subarray Operation (load one byte)

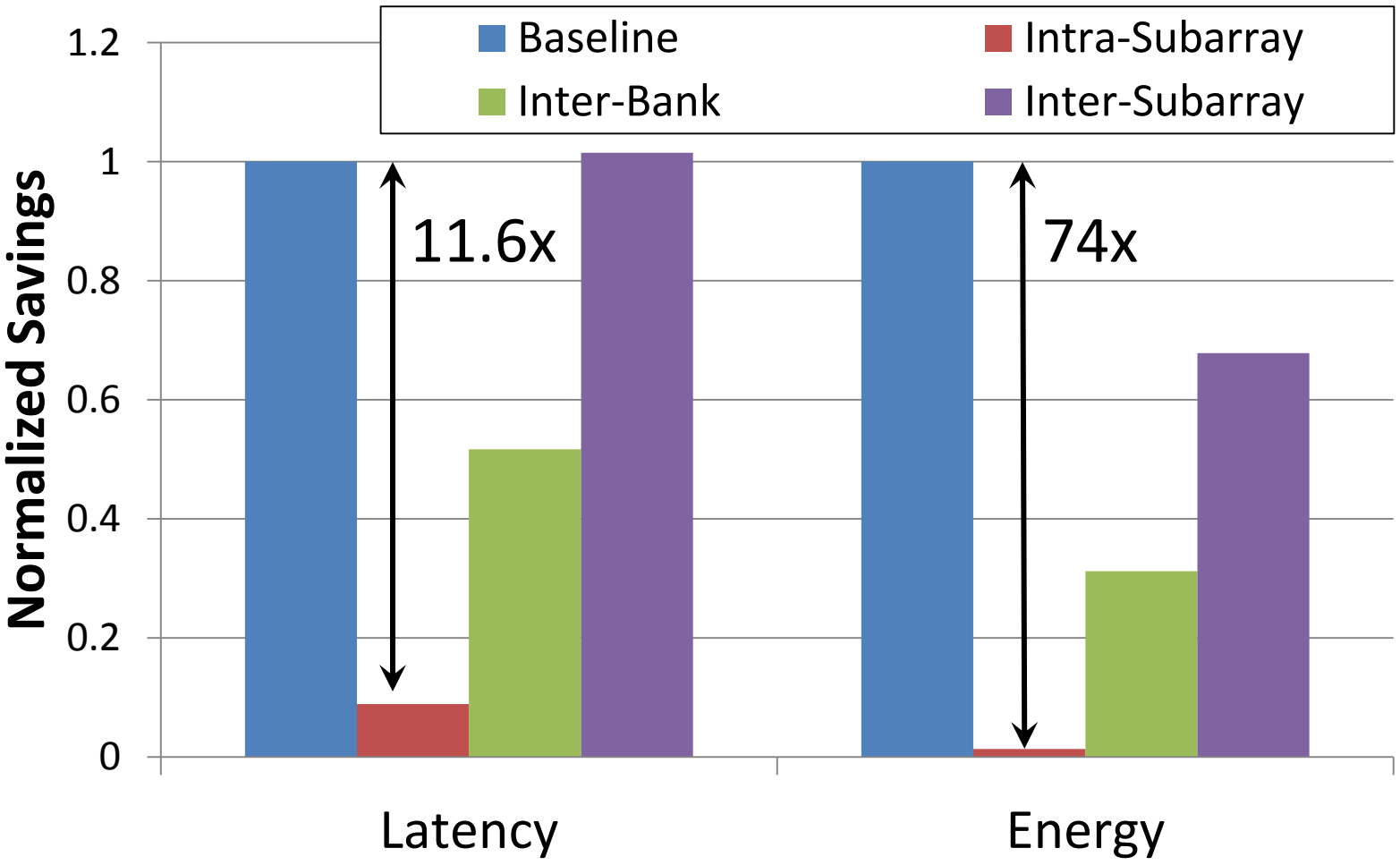


RowClone: In-DRAM Row Copy



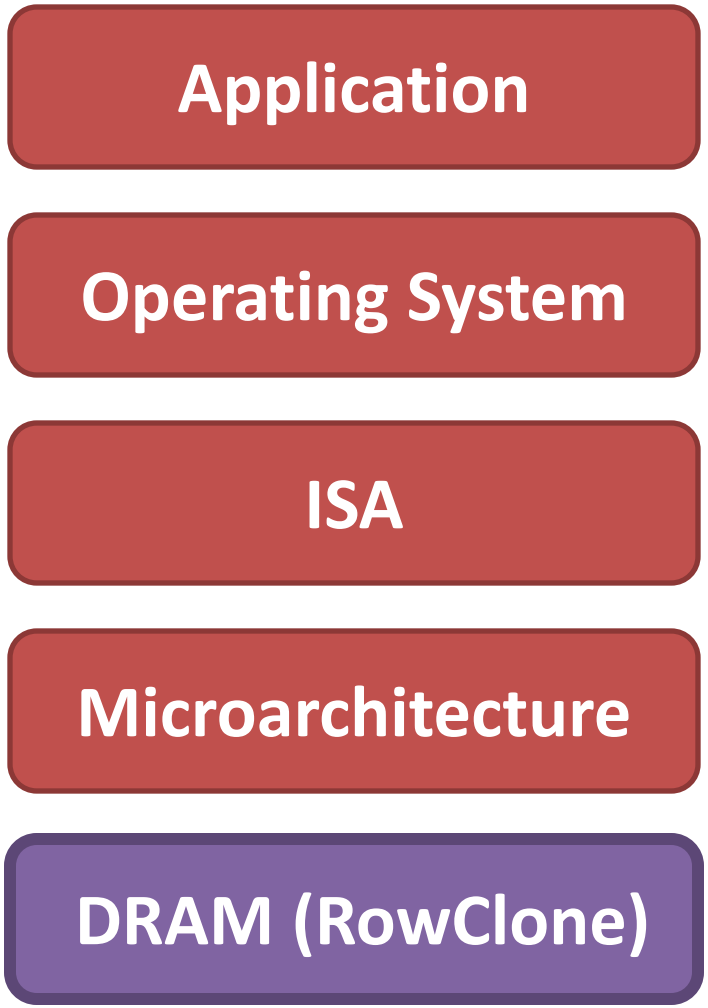
0.01% area

RowClone: Latency and Energy Savings



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

End-to-End System Design



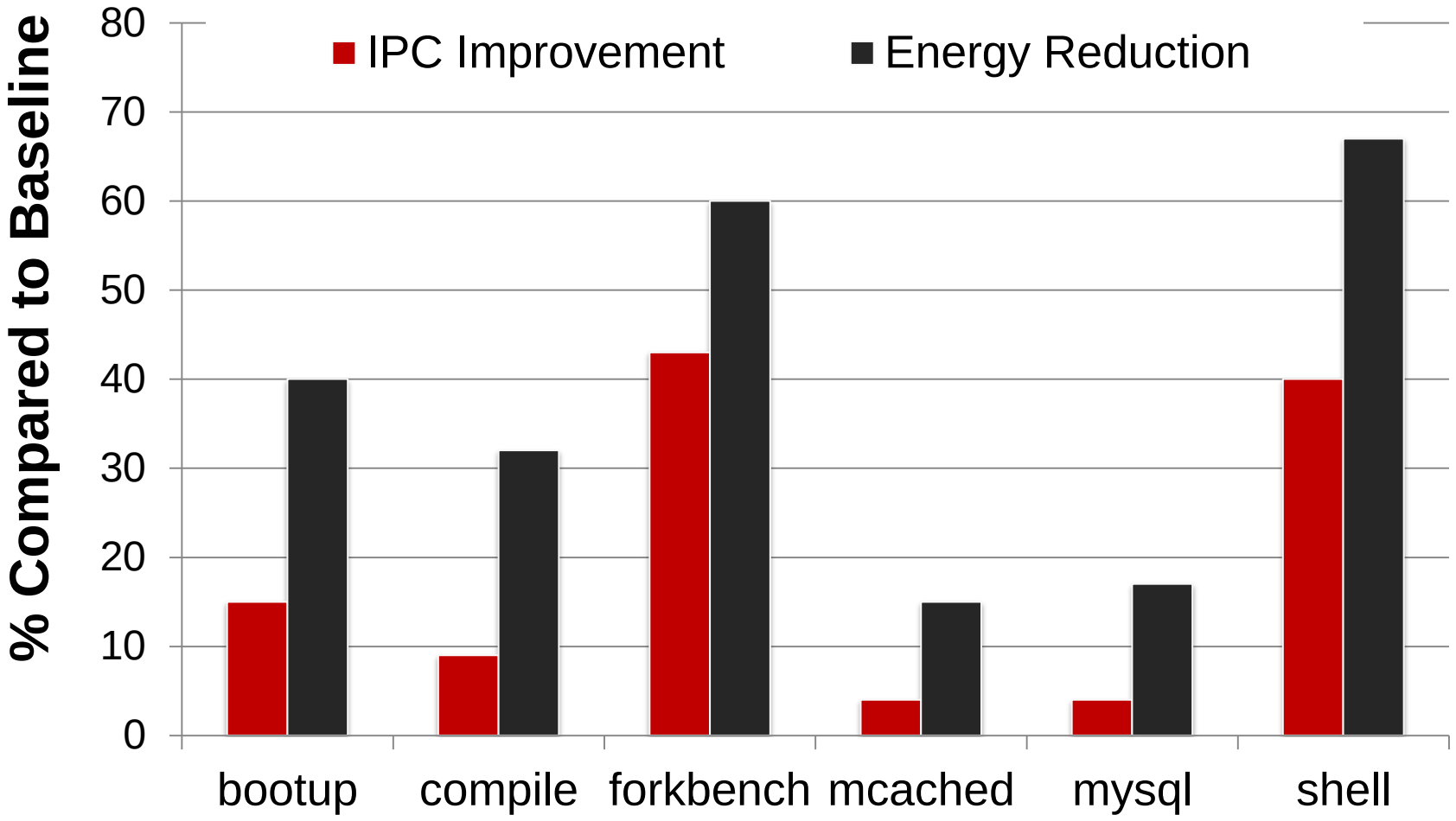
How does the software communicate occurrences of bulk copy/initialization to hardware?

How to ensure data coherence?

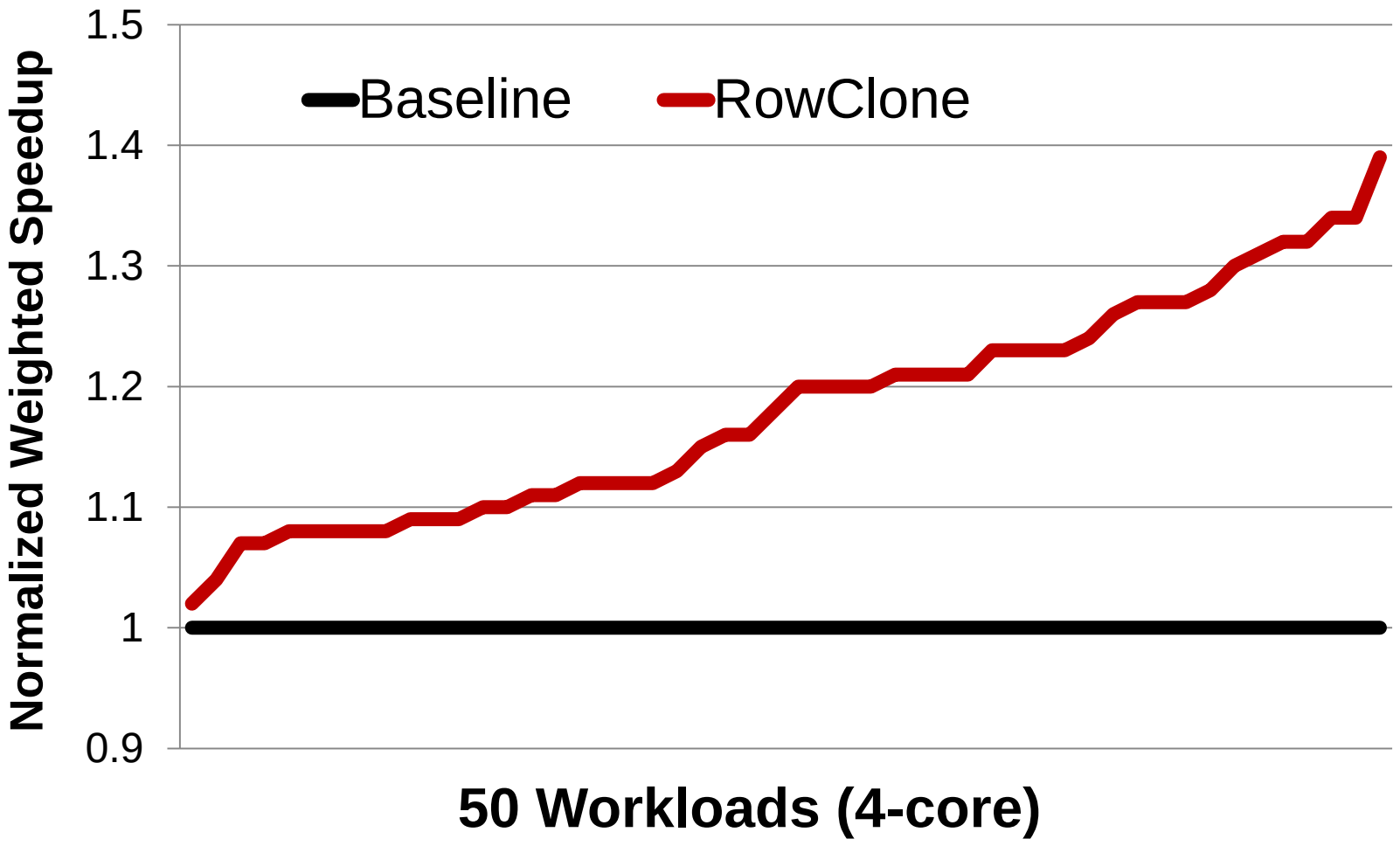
How to maximize latency and energy savings?

How to handle data reuse?

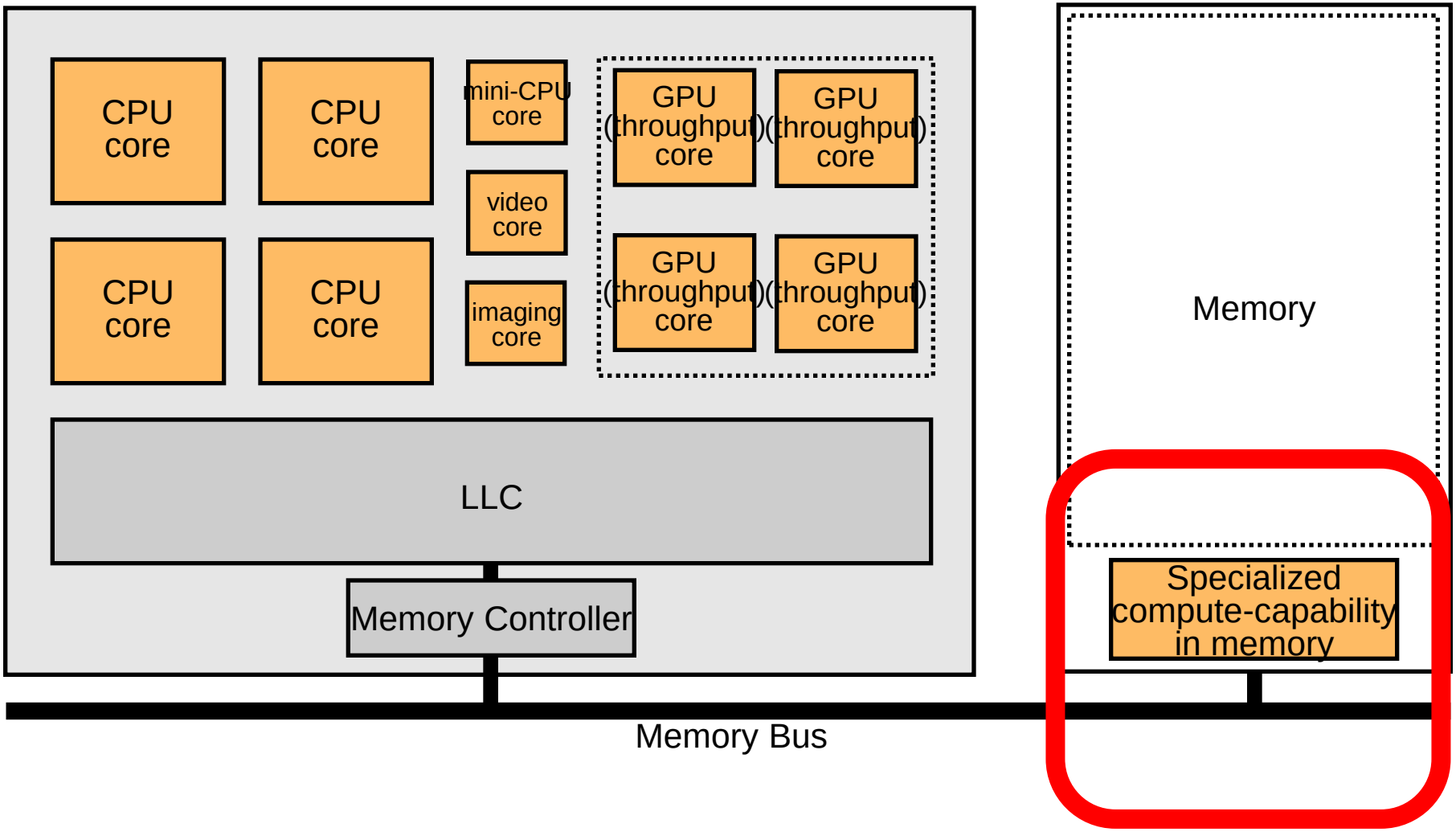
RowClone: Overall Performance



RowClone: Multi-Core Performance

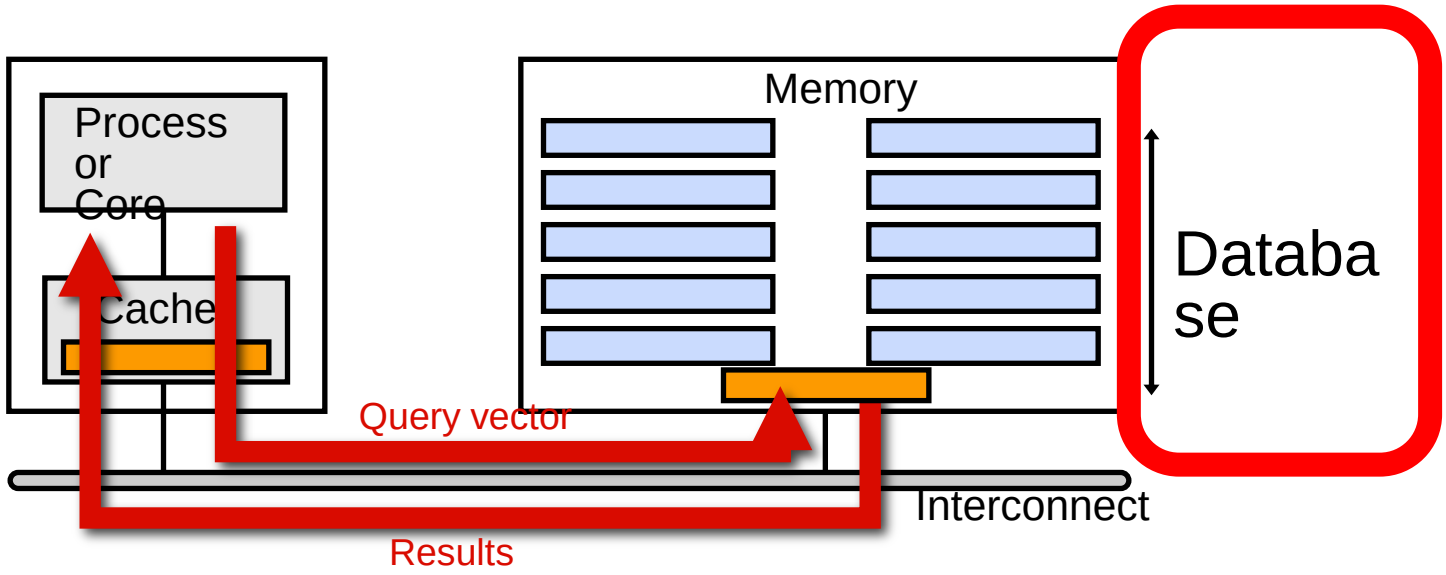


Goal: Ultra-Efficient Processing Near Data



Memory similar to a "conventional" accelera

Enabling Ultra-Efficient Search

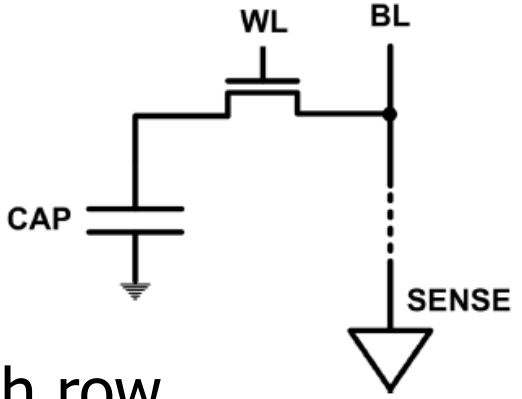


- What is the right partitioning of computation capability?
- What is the right low-cost memory substrate?
- What memory technologies are the best enablers?
- How do we rethink/ease (visual) search

Several Examples

- Bulk data copy (and initialization)
- DRAM refresh
- Memory reliability
- Disparity of working memory and persistent storage
- Homogeneous memory
- Memory QoS and predictable performance

DRAM Refresh



- DRAM capacitor charge leaks over time

- The memory controller needs to refresh each row periodically to restore charge
  - Activate each row every N ms
  - Typical N = 64 ms

- Downsides of refresh

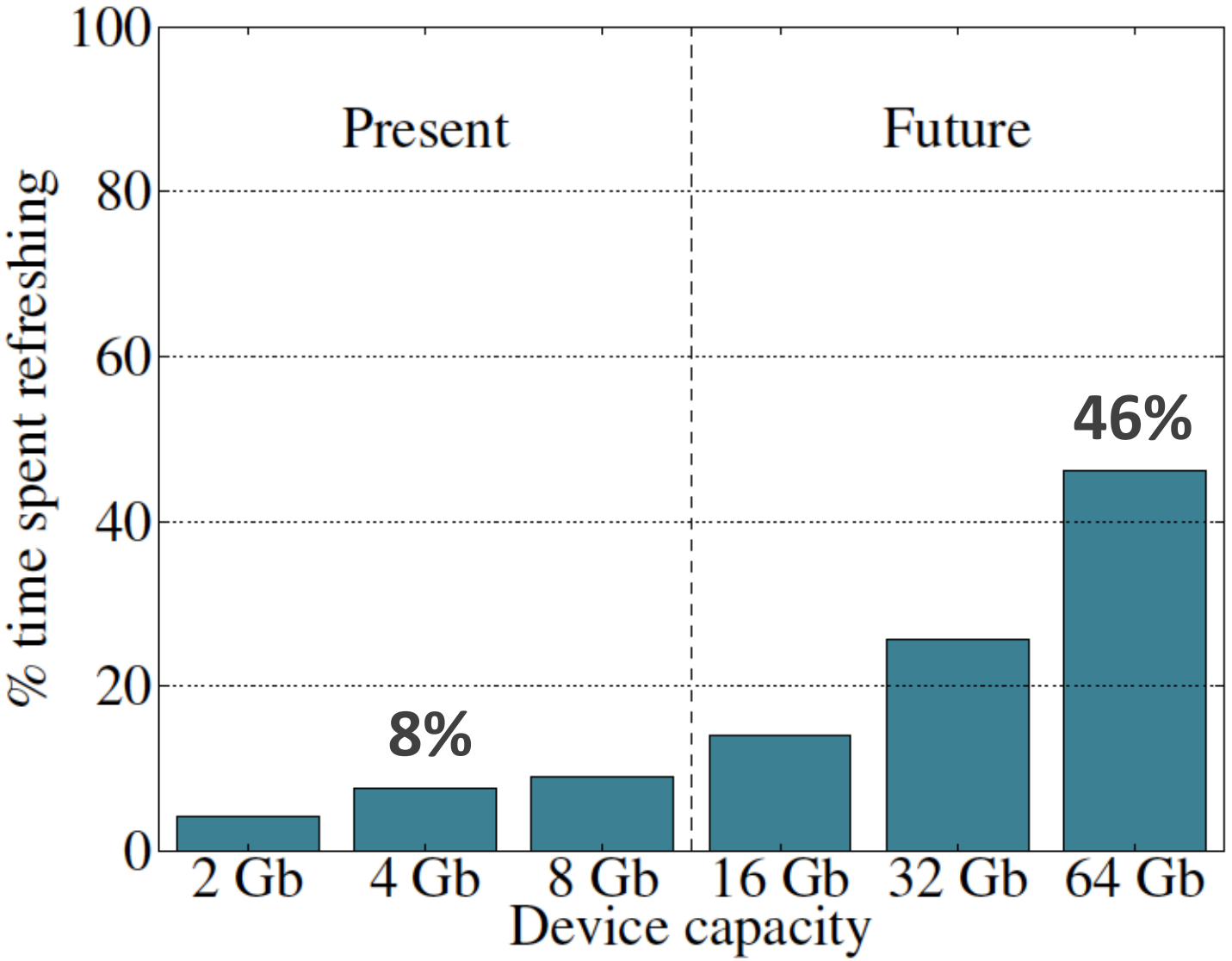
  -- Energy consumption: Each refresh consumes energy

  -- Performance degradation: DRAM rank/bank unavailable while refreshed

  -- QoS/predictability impact: (Long) pause times during refresh

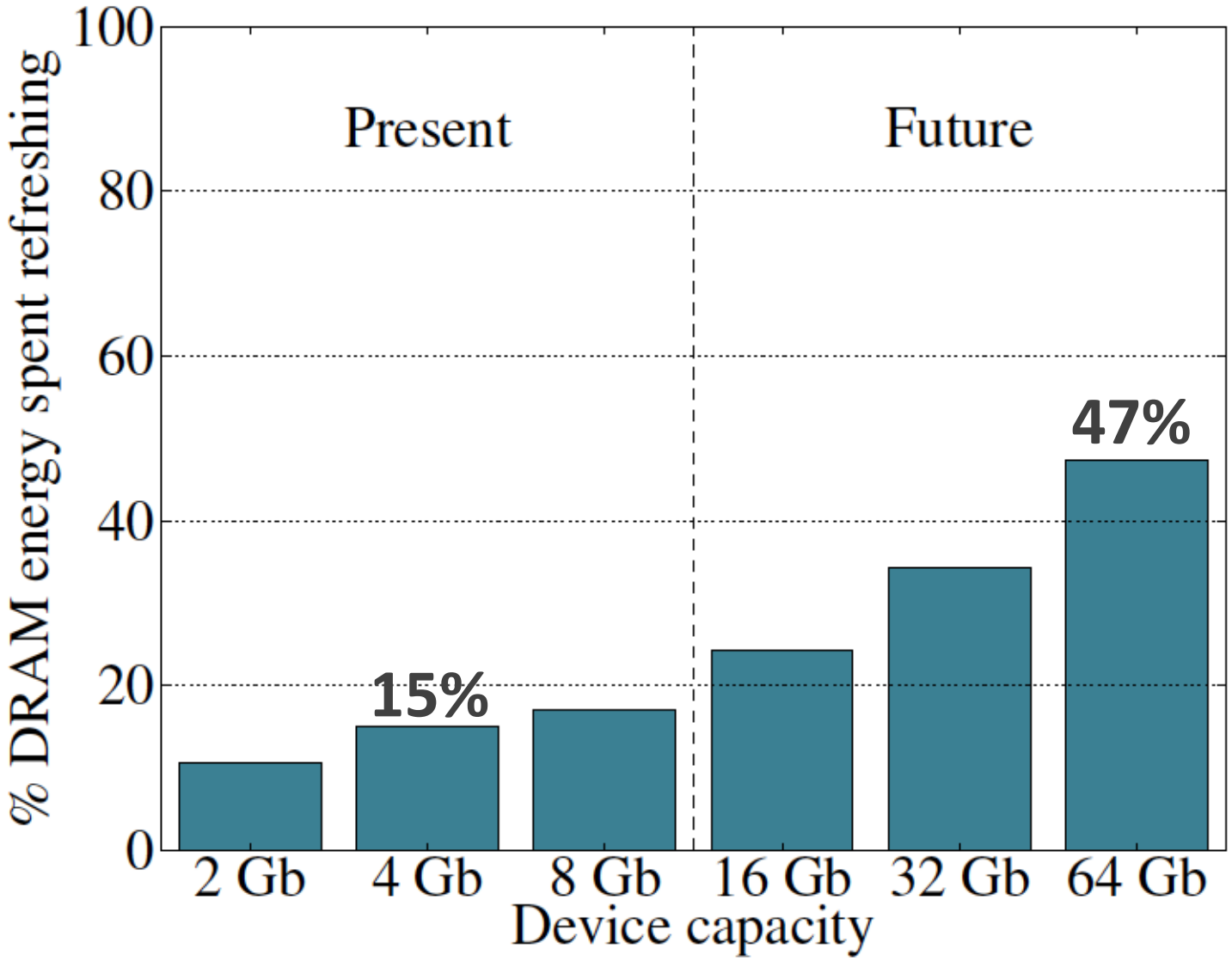
  -- Refresh rate limits DRAM capacity scaling

Refresh Overhead: Performance



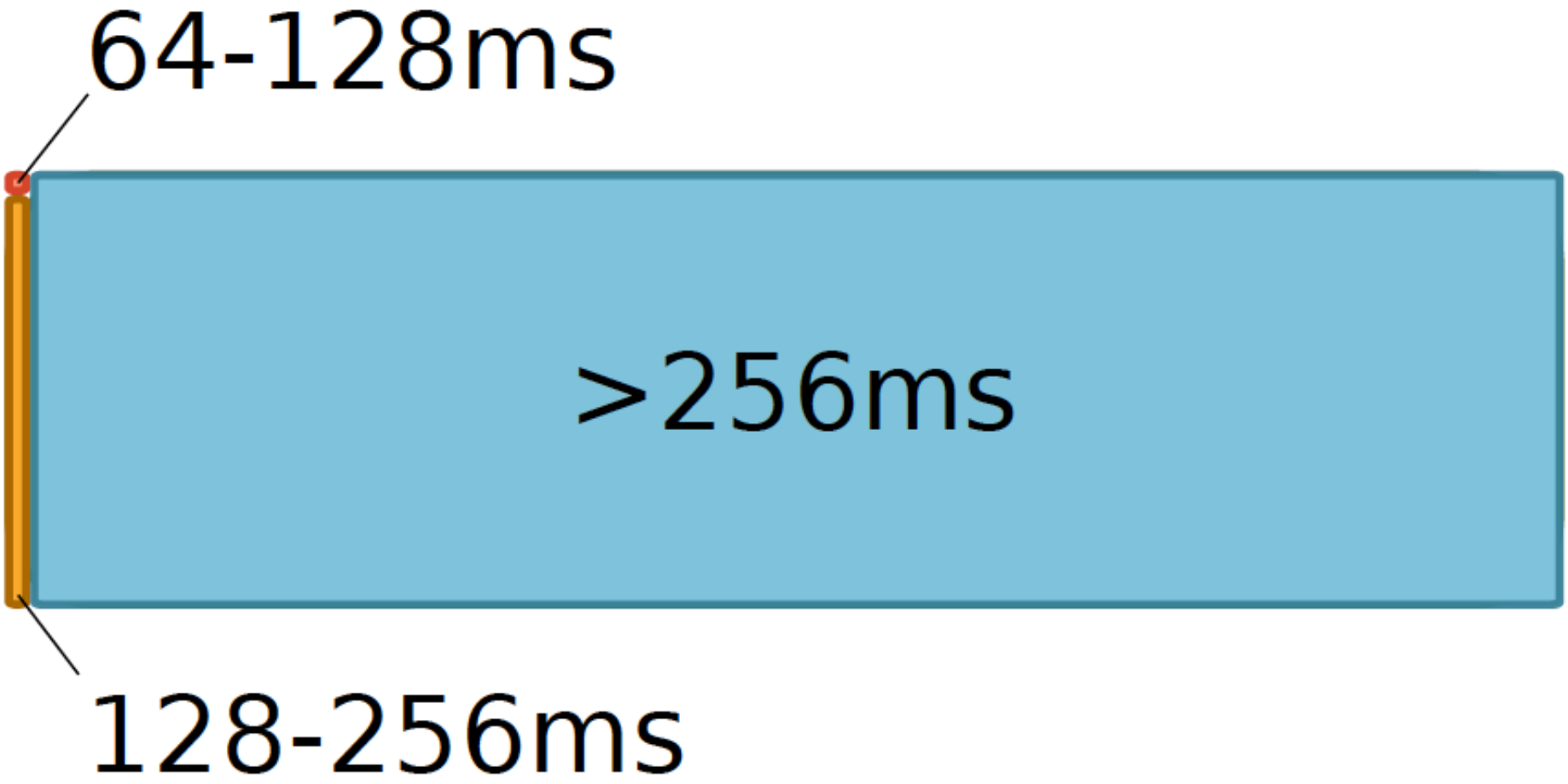
Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

Refresh Overhead: Energy



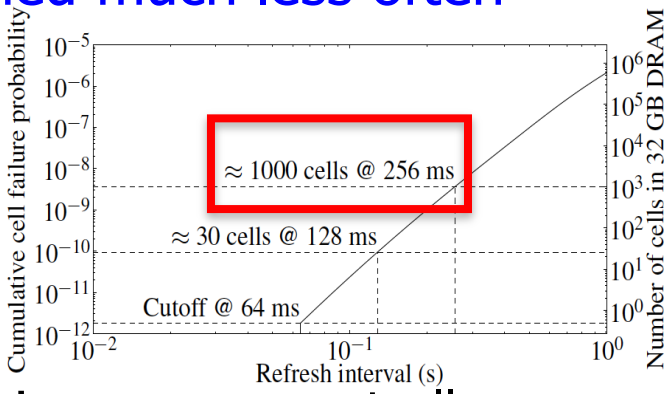
Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

Retention Time Profile of DRAM



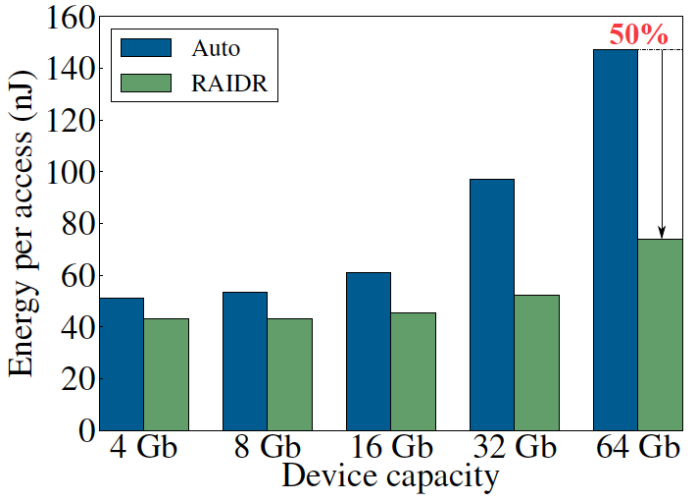
RAIDR: Eliminating Unnecessary Refreshes

- Observation: Most DRAM rows can be refreshed much less often without losing data **[Kim+, EDL'09][Liu+ ISCA'13]**



- Key idea: Refresh rows containing weak cells more frequently, other rows less frequently
  1. Profiling: Profile retention time of all rows
  2. Binning: Store rows into bins by retention time in memory controller
     Efficient storage with Bloom Filters (only 1.25KB for 32GB memory)
  3. Refreshing: Memory controller refreshes rows in different bins at different rates

- Results: 8-core, 32GB, SPEC, TPC-C, TPC-H
  - 74.6% refresh reduction @ 1.25KB storage
  - ~16%/20% DRAM dynamic/idle power reduction
  - ~9% performance improvement
  - Benefits increase with DRAM capacity



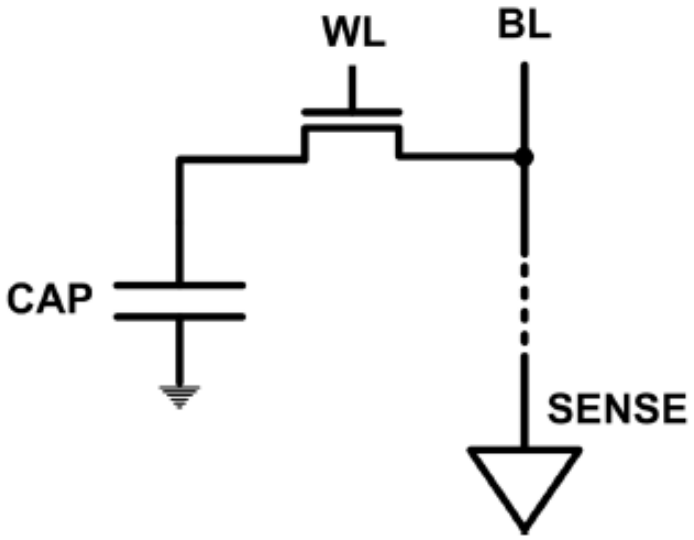
Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.

Several Examples

- Bulk data copy (and initialization)
- DRAM refresh
- Memory reliability
- Disparity of working memory and persistent storage
- Homogeneous memory
- Memory QoS and predictable performance

The DRAM Scaling Problem

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

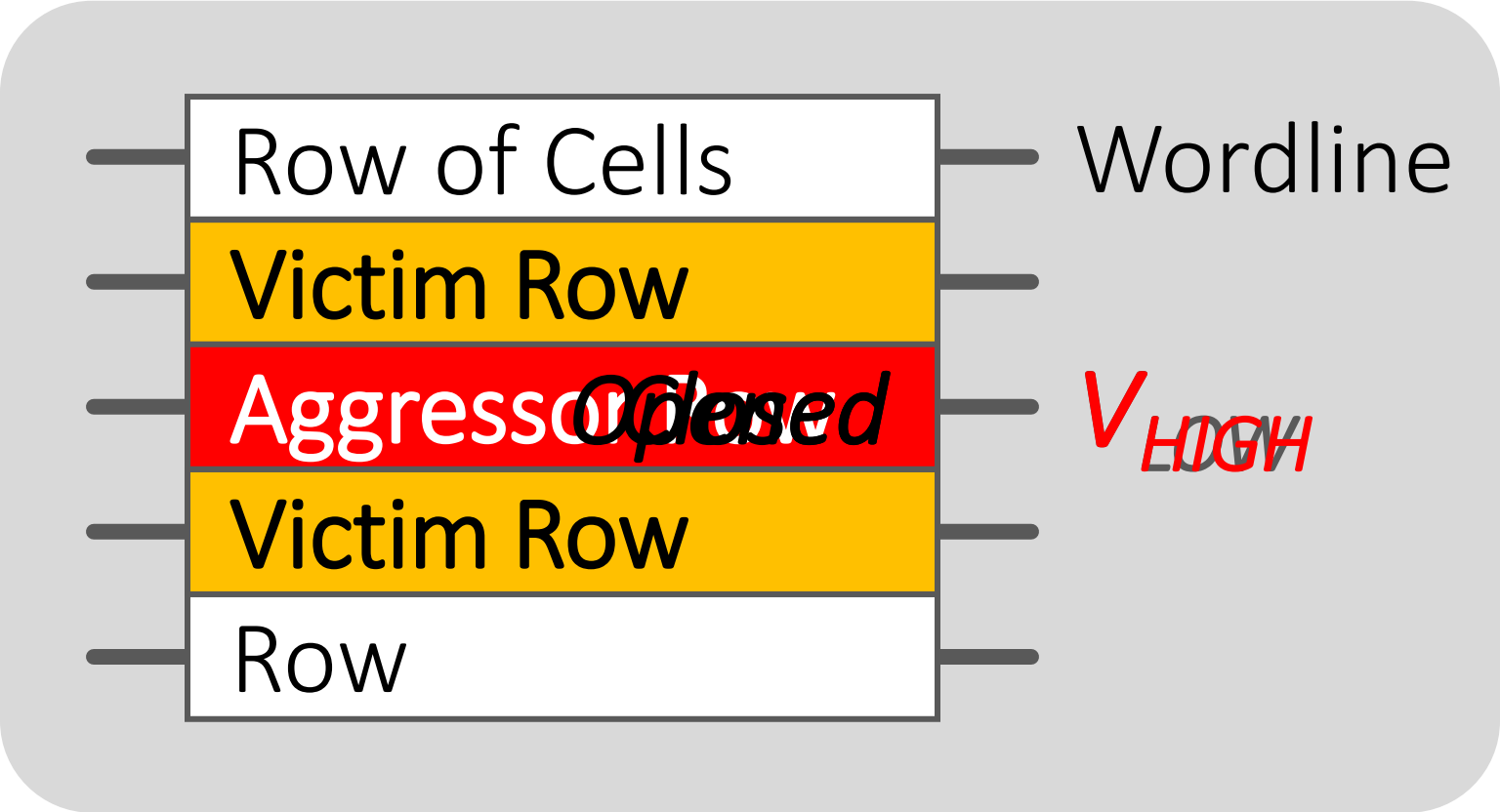


- DRAM capacity, cost, and energy/power hard to scale

The DRAM Scaling Problem

- DRAM scaling has become a real problem the system should be concerned about
  - And, maybe embrace

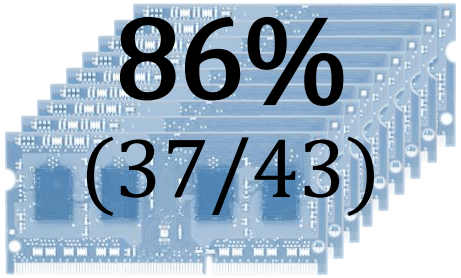
An Example of The Scaling Problem



*Repeatedly opening and closing a row induces **disturbance errors** in adjacent rows in **most real DRAM chips** [Kim+ ISCA 2014]*

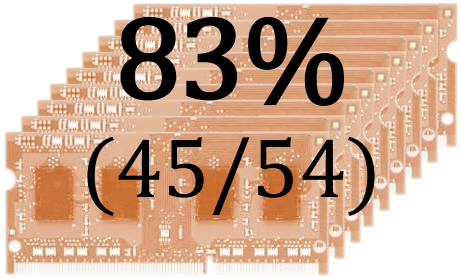
Most DRAM Modules Are at Risk

A company



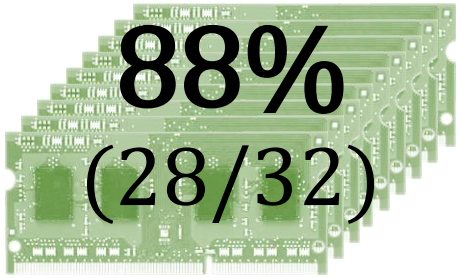
Up to
1.0×10^7
errors

B company



Up to
2.7×10^6
errors

C company

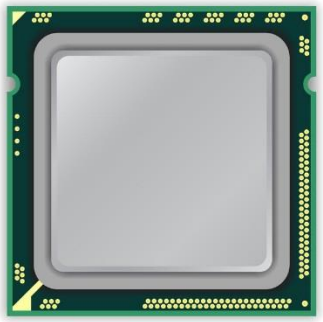


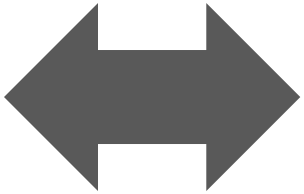
Up to
3.3×10^5
errors

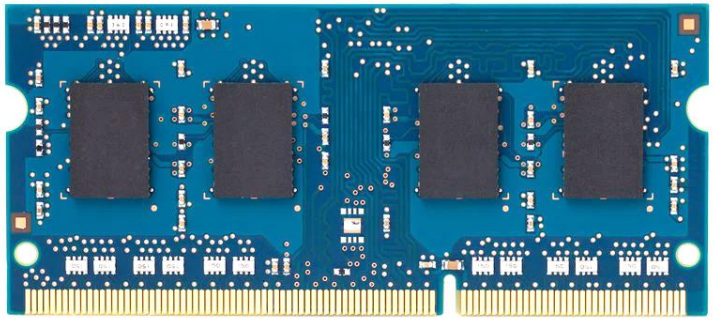
Kim+, "Flipping Bits in Memory Without Accessing Them: An Experimental Study of DRAM Disturbance Errors," ISCA 2014.

x86 CPU

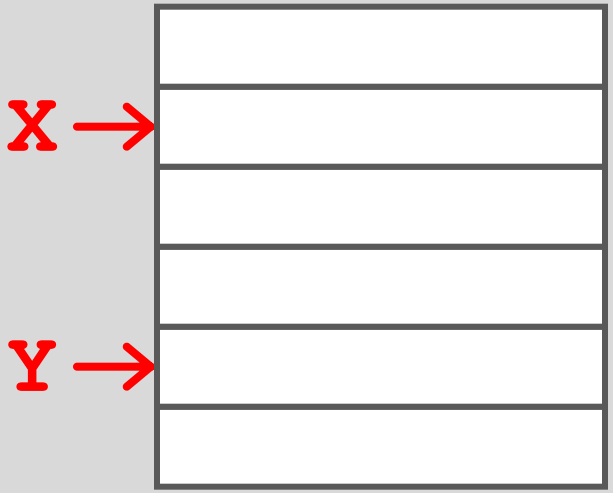
DRAM Module



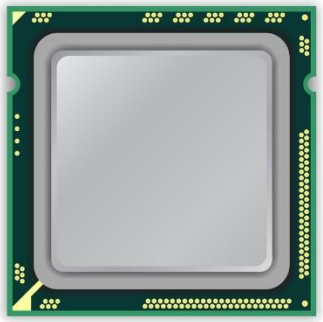




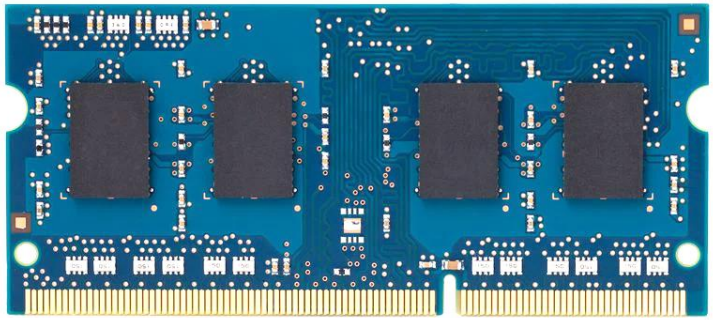
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



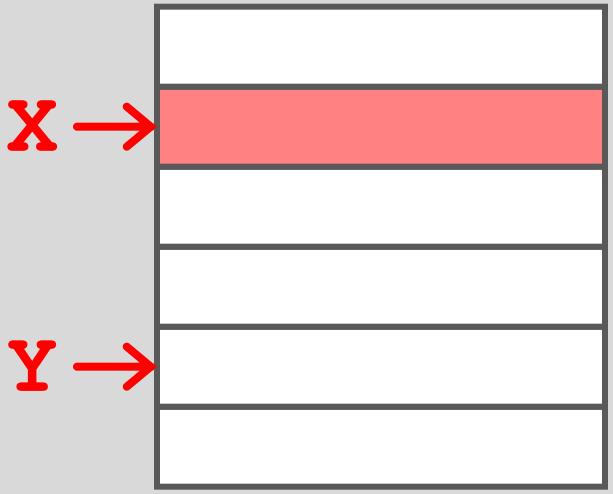
x86 CPU



DRAM Module

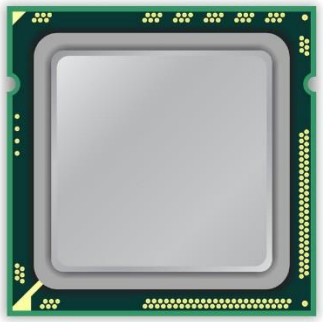


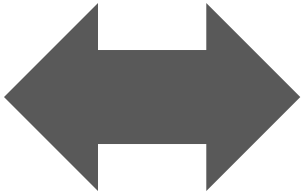
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```

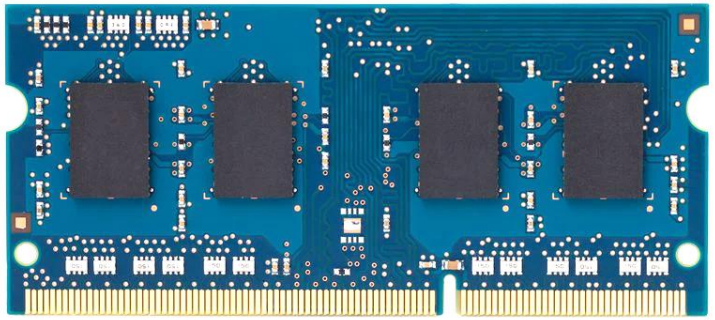


x86 CPU

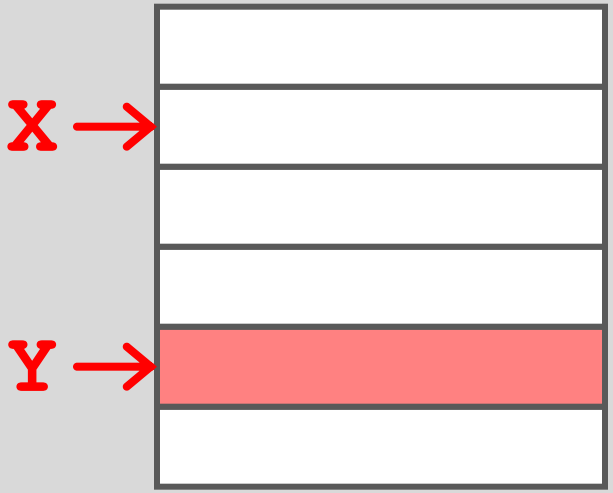
DRAM Module



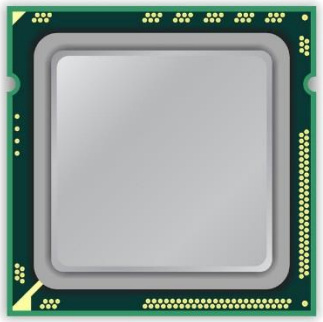




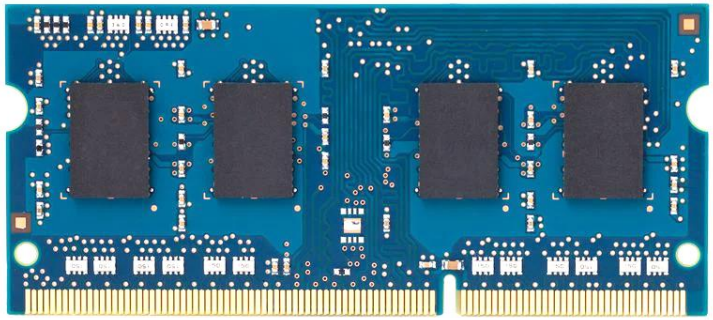
```
loop:
   mov (X), %eax
   mov (Y), %ebx
   clflush (X)
   clflush (Y)
   mfence
   jmp loop
```



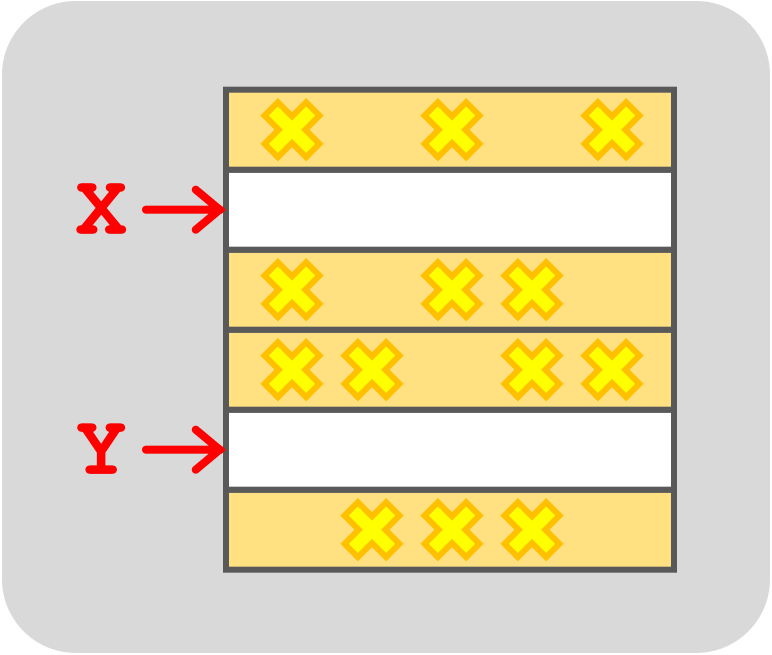
x86 CPU



DRAM Module



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



Observed Errors in Real Systems

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

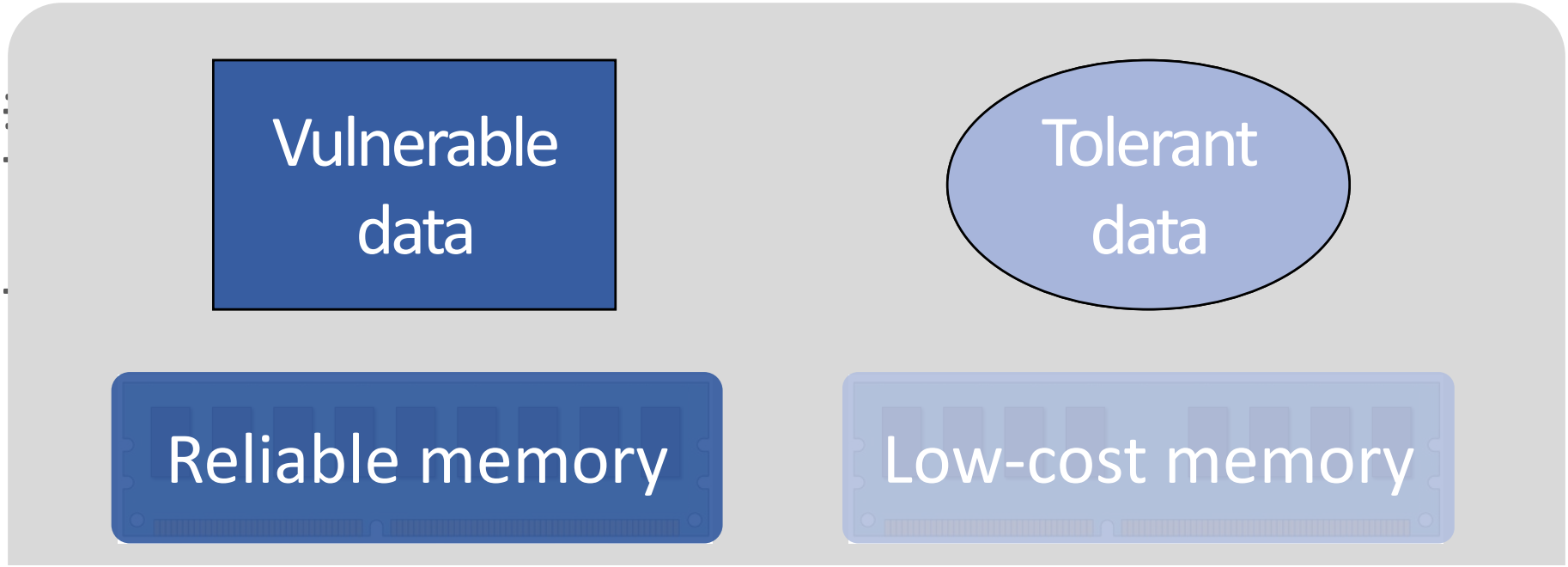
- *In a more controlled environment, we can induce as many as ten million disturbance errors*
- *Disturbance errors are a serious reliability issue*

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

How Do We Solve The Problem?

- Tolerate it: Make DRAM and controllers more intelligent
  - Just like flash memory and hard disks
- Eliminate or minimize it: Replace or (more likely) augment DRAM with a different technology
- Embrace it: Design heterogeneous-reliability memories that map error-tolerant data to less reliable portions
- …

Exploiting Memory Error Tolerance



On Microsoft's Web Search application

Reduces server hardware cost by 4.7 %

Achieves single server availability target of 99.90 %

Heterogeneous-**R**eliability **M**emory

Several Examples

- Bulk data copy (and initialization)
- DRAM refresh
- DRAM reliability
- Disparity of working memory and persistent storage
- Homogeneous memory
- Memory QoS and predictable performance

Agenda

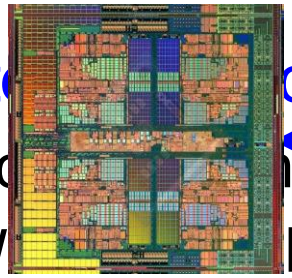
- Principled Computer Architecture/System Design
- How We Violate Those Principles Today
- Some Solution Approaches
- Concluding Remarks

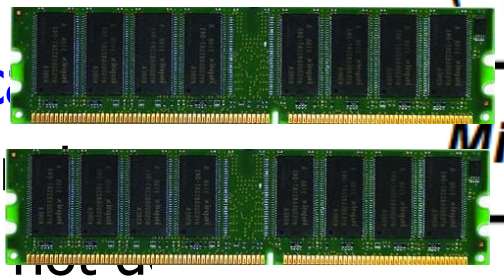
Some Directions for the Future

- We need to rethink the entire memory/storage system
  - Satisfy data-intensive workloads
  - Fix many DRAM issues (energy, reliability, …)
  - Enable emerging technologies
  - Enable a better overall system design
- We need to find a better balance between moving data versus moving computation
  - Minimize system energy and bandwidth
  - Maximize system performance and efficiency
- We need to enable system-level memory/storage QoS
  - Provide predictable performance
  - Build controllable and robust systems

Some Solution Principles (So Far)

- More data-centric system
  - Do not center everything
- Better cooperation across
  - Careful co-design of com
  - More flexible interfaces
- Bett orst-c
  - Do ize for
  - W hould not d
- Heterogeneity in design (
  - Enables a more efficient





Agenda

- Principled Computer Architecture/System Design
- How We Violate Those Principles Today
- Some Solution Approaches
- Concluding Remarks

Role of the Architect

Role of the Architect

-- Look Backward (Examine old code)

-- Look forward (Listen to the dreamers)

-- Look Up (Nature of the problems)

-- Look Down (Predict the future of technology)

from Yale Patt's EE 382N lecture notes

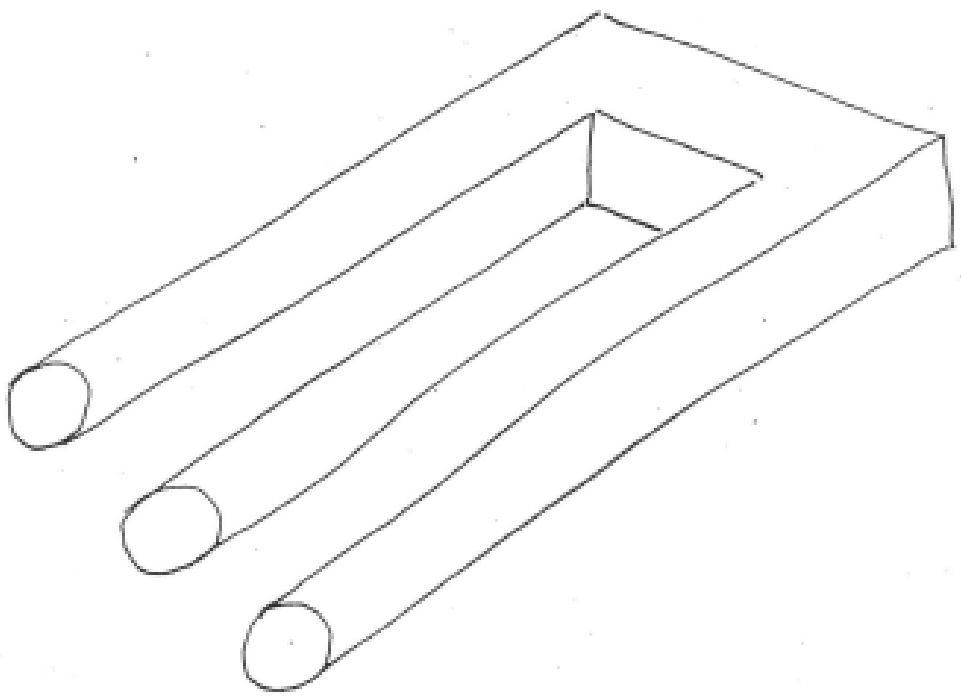
A Quote from Another Famous Architect

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

Concluding Remarks

- It is time to design systems to be more balanced, i.e., more memory-centric
- It is time to make memory/storage a priority in system design and optimize it & integrate it better into the system
- It is time to design systems to be more focused on critical pieces of work
- Future systems will/should be data-centric and memory-centric, with appropriate attention to principles

***Finally*, people are always telling you:**
Think outside the box



from Yale Patt's EE 382N lecture notes

I prefer: Expand the box

from Yale Patt's EE 382N lecture notes

Rethinking the Systems We Design

Onur Mutlu
onur@cmu.edu
September 19, 2014
Yale @ 75

Carnegie Mellon

SAFARI