

CSC 2224: Parallel Computer Architecture and Programming Main Memory. DRAM.

Prof. Gennady Pekhimenko

University of Toronto

Fall 2022

*The content of this lecture is adapted from the slides of
Vivek Seshadri, Donghyuk Lee, Yoongu Kim,
and lectures of Onur Mutlu @ ETH and CMU*

Outline

1. What is DRAM?

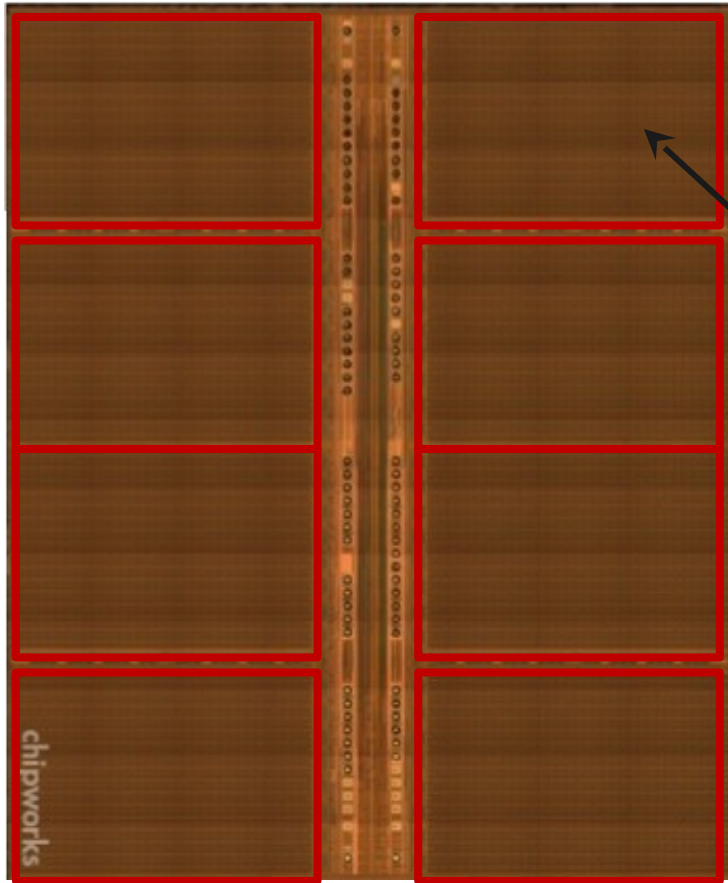
2. DRAM Internal Organization

- DRAM Cell
- DRAM Array
- DRAM Bank

3. Problems and Solutions

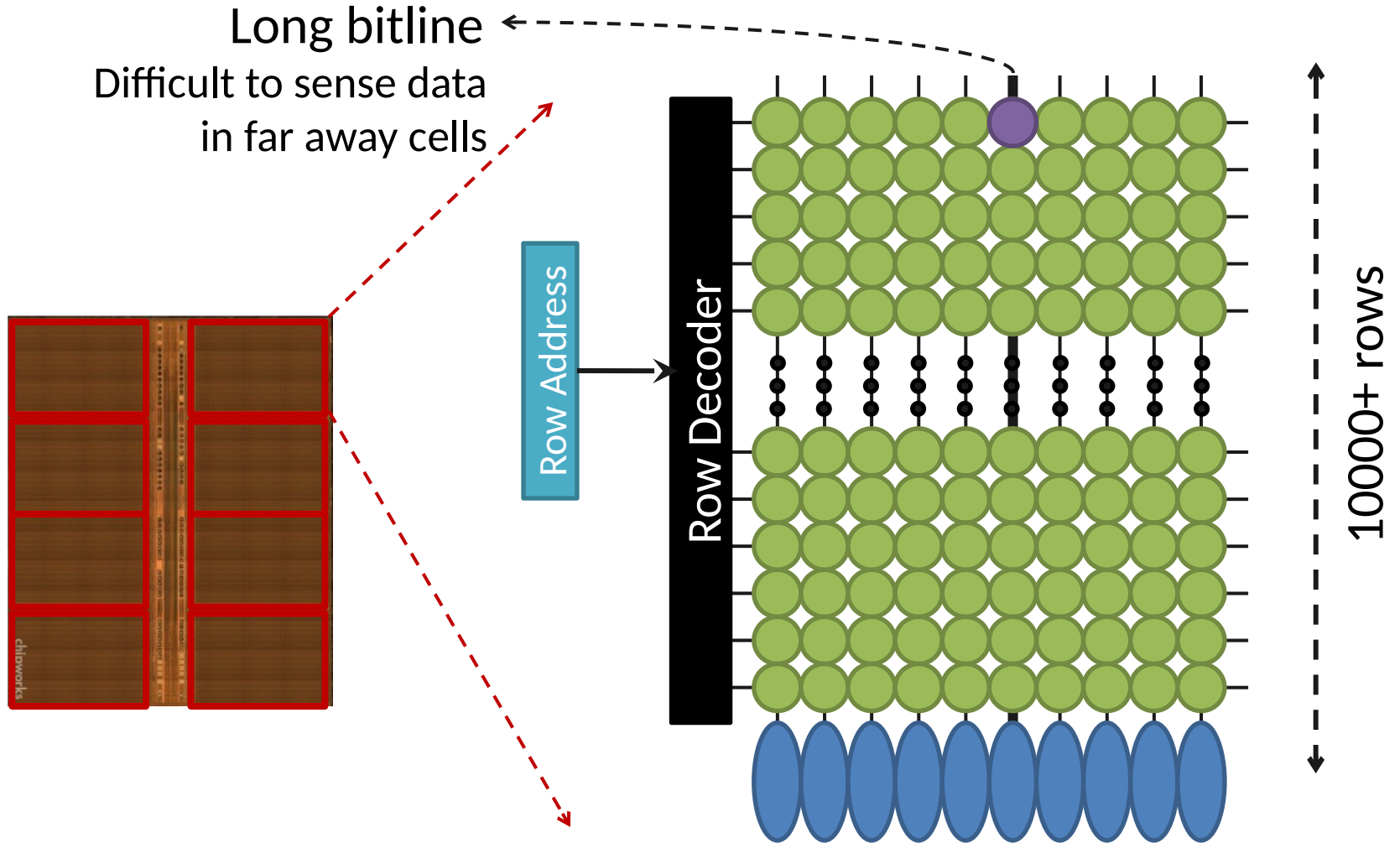
- Latency (Tiered-Latency DRAM, HPCA 2013 Adaptive-Latency DRAM, HPCA 2015)
- Parallelism (Subarray-level Parallelism, ISCA 2012)

DRAM Bank

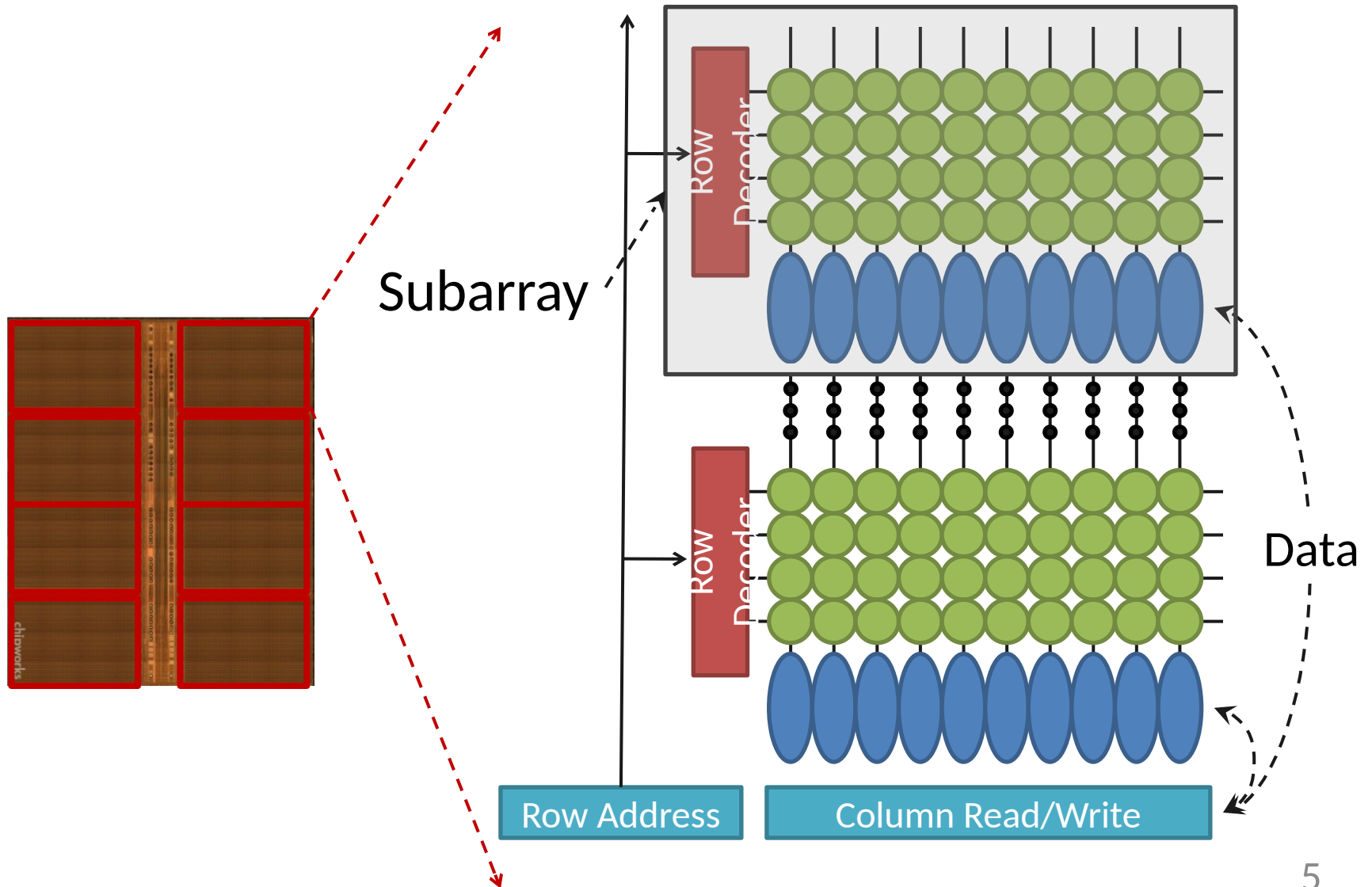


How to build a DRAM bank from a DRAM array?

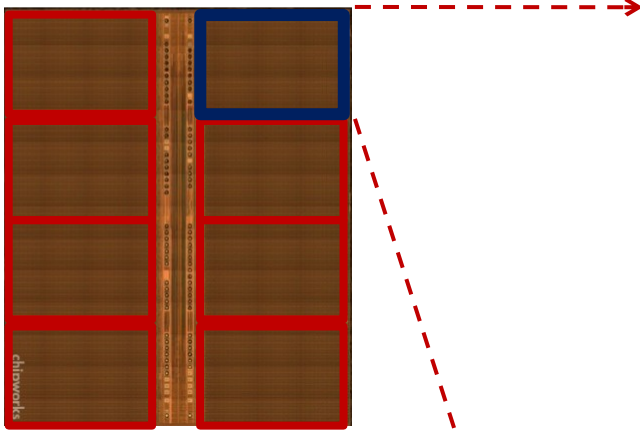
DRAM Bank: Single DRAM Array?



DRAM Bank: Collection of Arrays

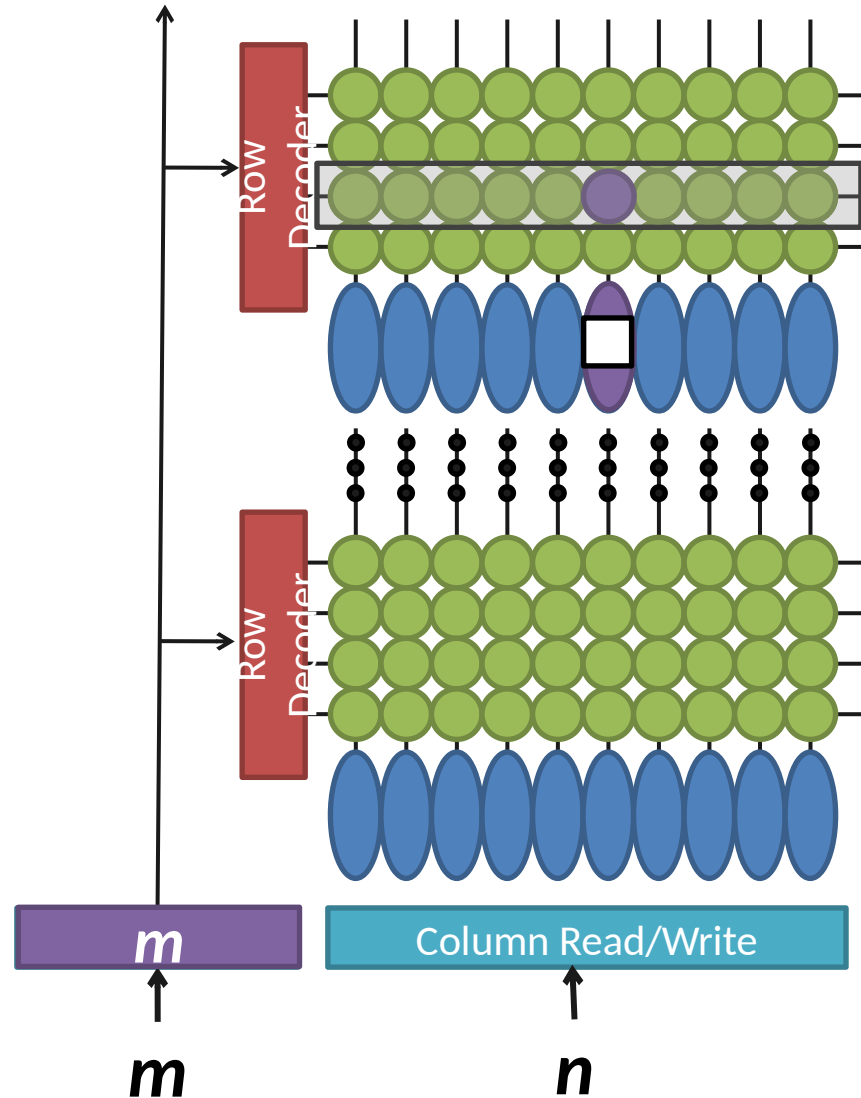


DRAM Operation: Summary



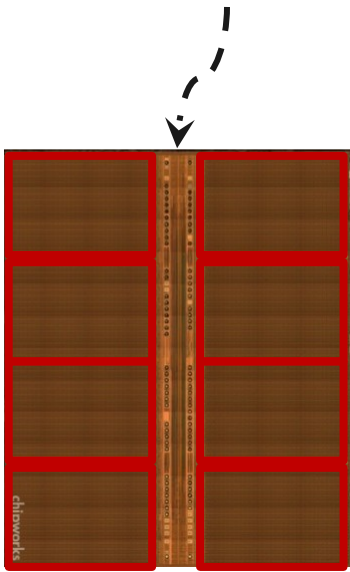
Row m , Col n

1. Enable row m
2. Access col n
3. Close row

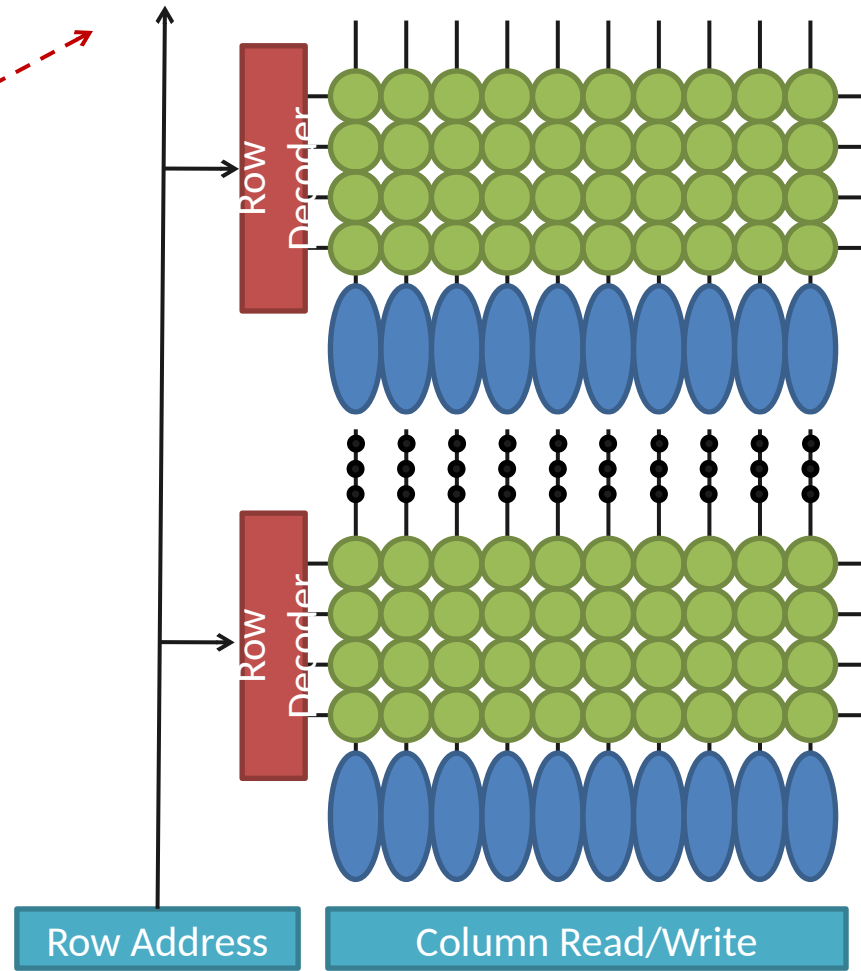


DRAM Chip Hierarchy

Collection of Banks



Collection of Subarrays



Outline

1. What is DRAM?

2. DRAM Internal Organization

3. Problems and Solutions

- Latency (Tiered-Latency DRAM, HPCA 2013; Adaptive-Latency DRAM, HPCA 2015)
- Parallelism (Subarray-level Parallelism, ISCA 2012)

Factors That Affect Performance

1. Latency

- How fast can DRAM serve a request?

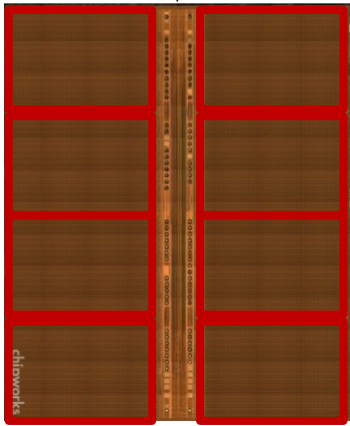
2. Parallelism

- How many requests can DRAM serve in parallel?

DRAM Chip Hierarchy

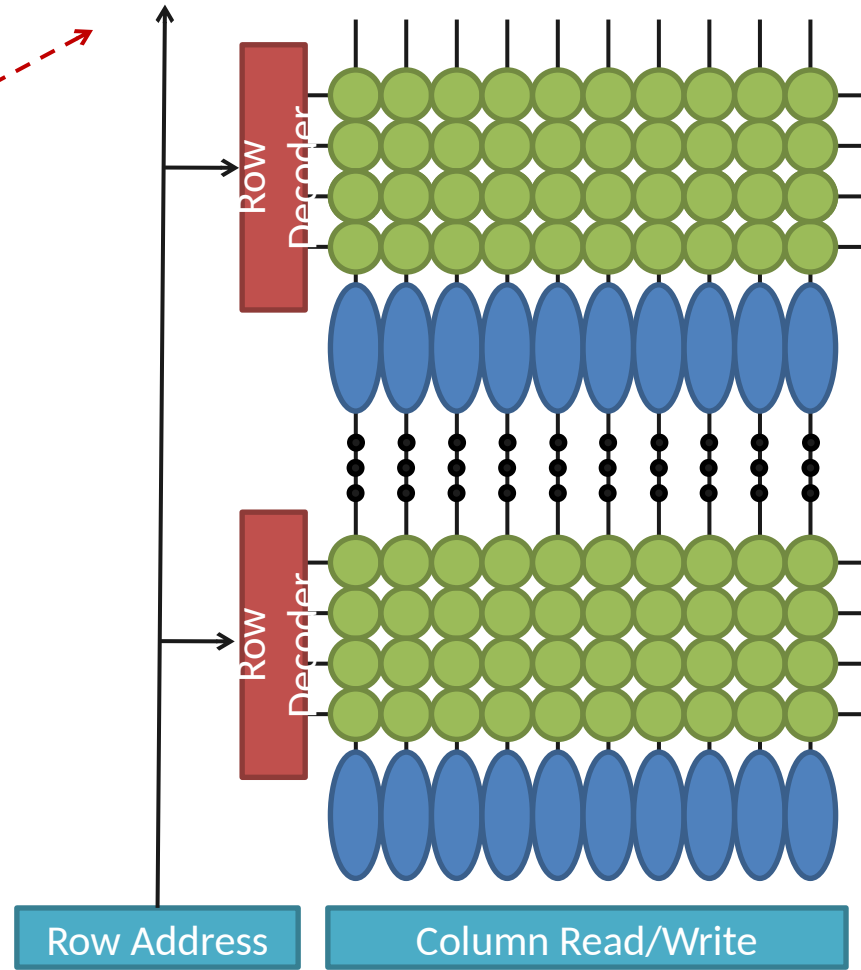
Collection of Banks

Parallelism



Latency

Collection of Subarrays



Outline

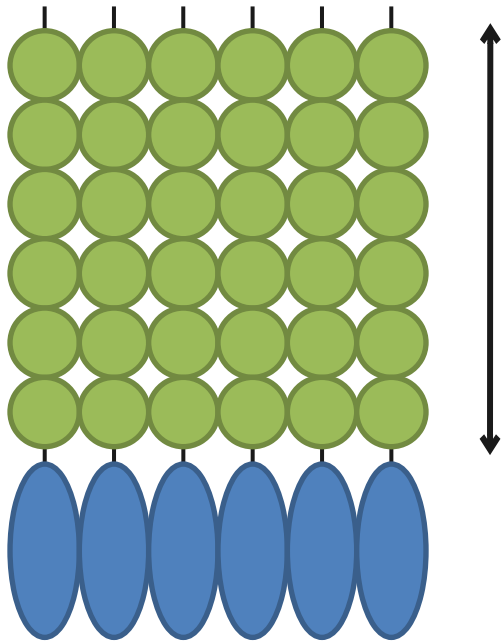
1. What is DRAM?

2. DRAM Internal Organization

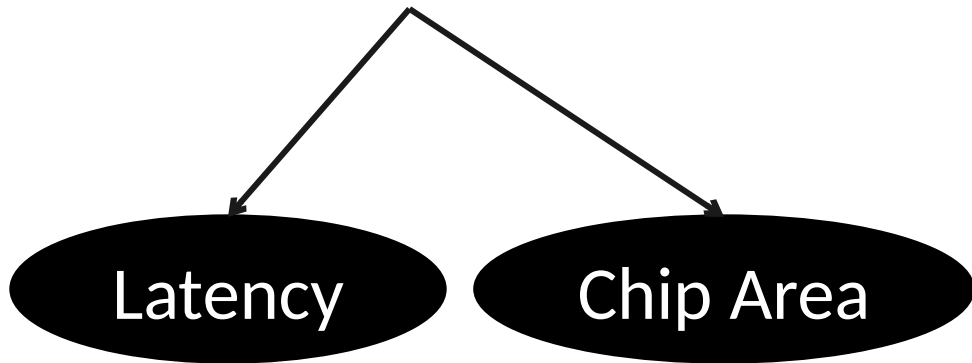
3. Problems and Solutions

- Latency (Tiered-Latency DRAM, HPCA 2013; Adaptive-Latency DRAM, HPCA 2015)
- Parallelism (Subarray-level Parallelism, ISCA 2012)

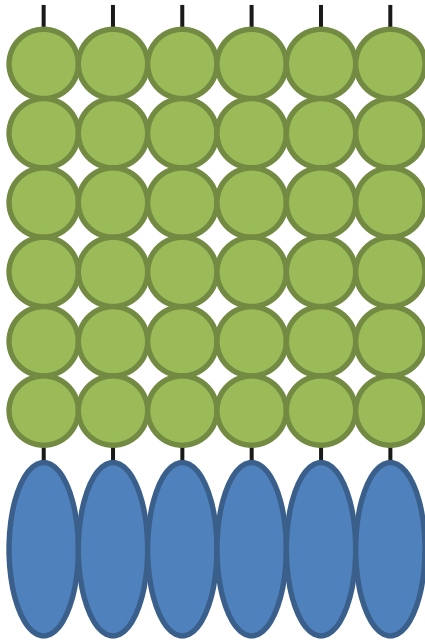
Subarray Size: Rows/Subarray



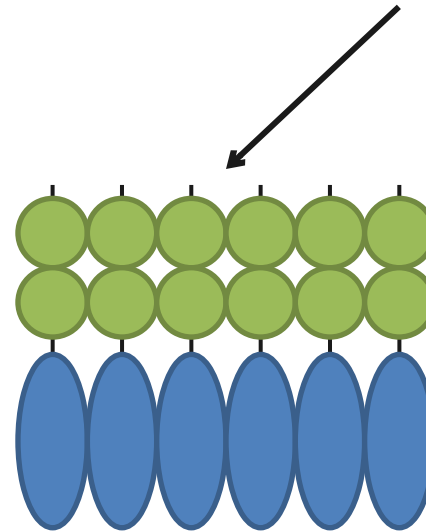
Number of rows in subarray



Subarray Size vs. Access Latency



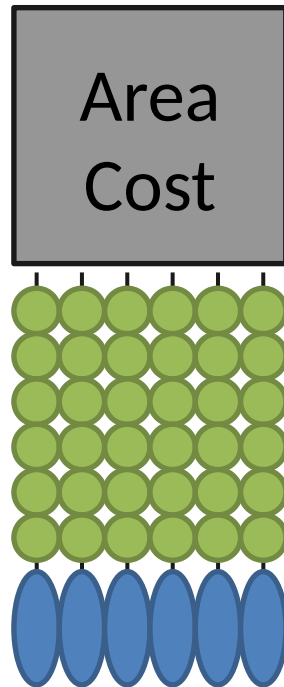
Shorter Bitlines => Faster access



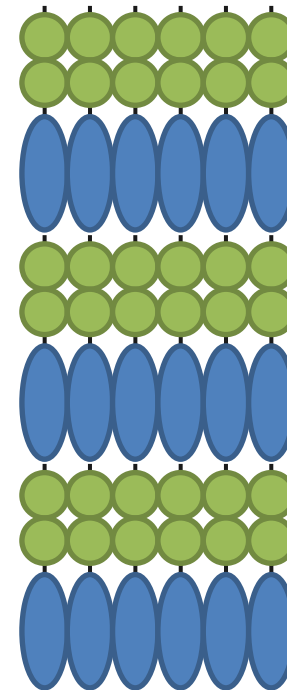
Smaller subarrays => lower access latency

Subarray Size vs. Chip Area

Large Subarray

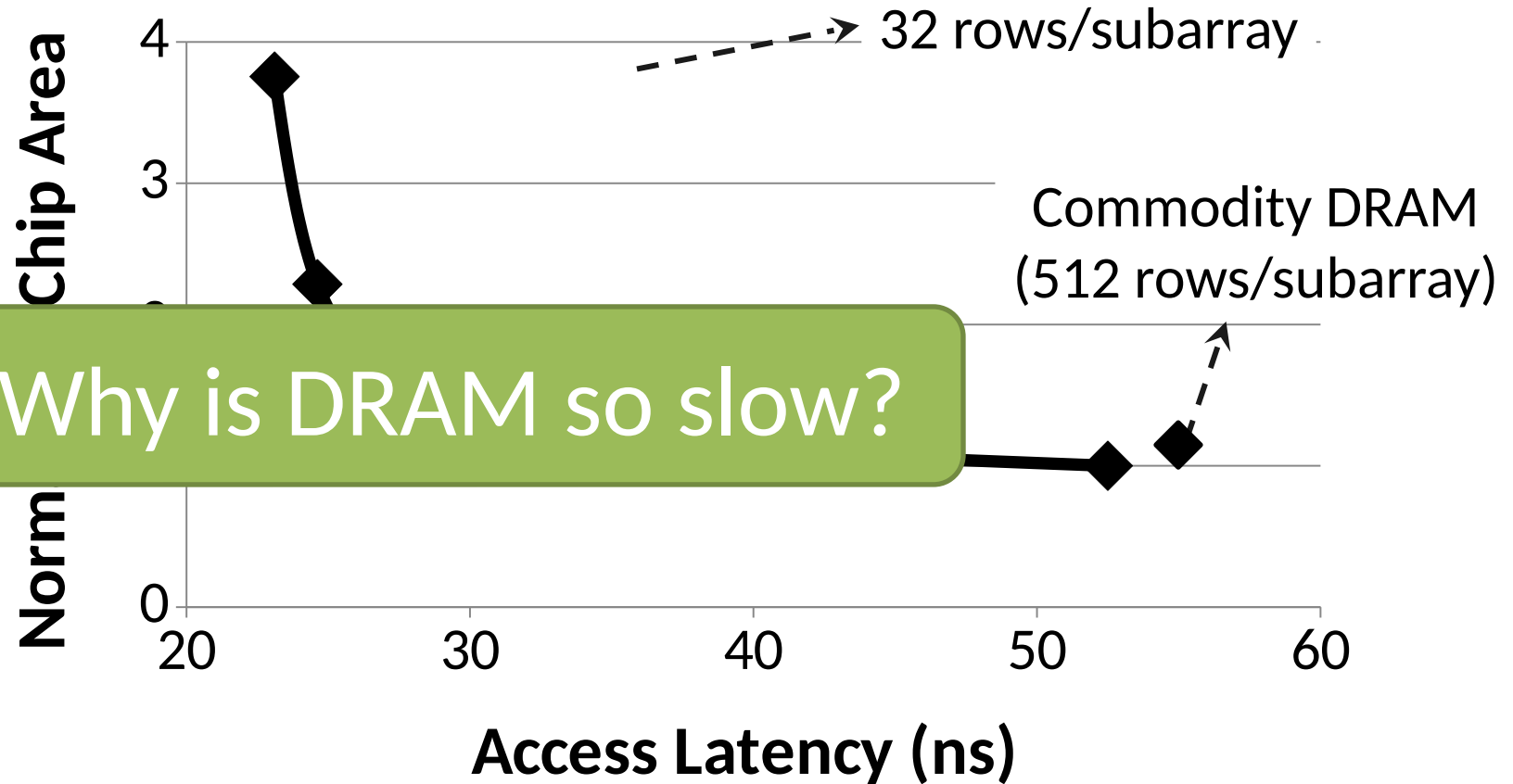


Smaller Subarrays

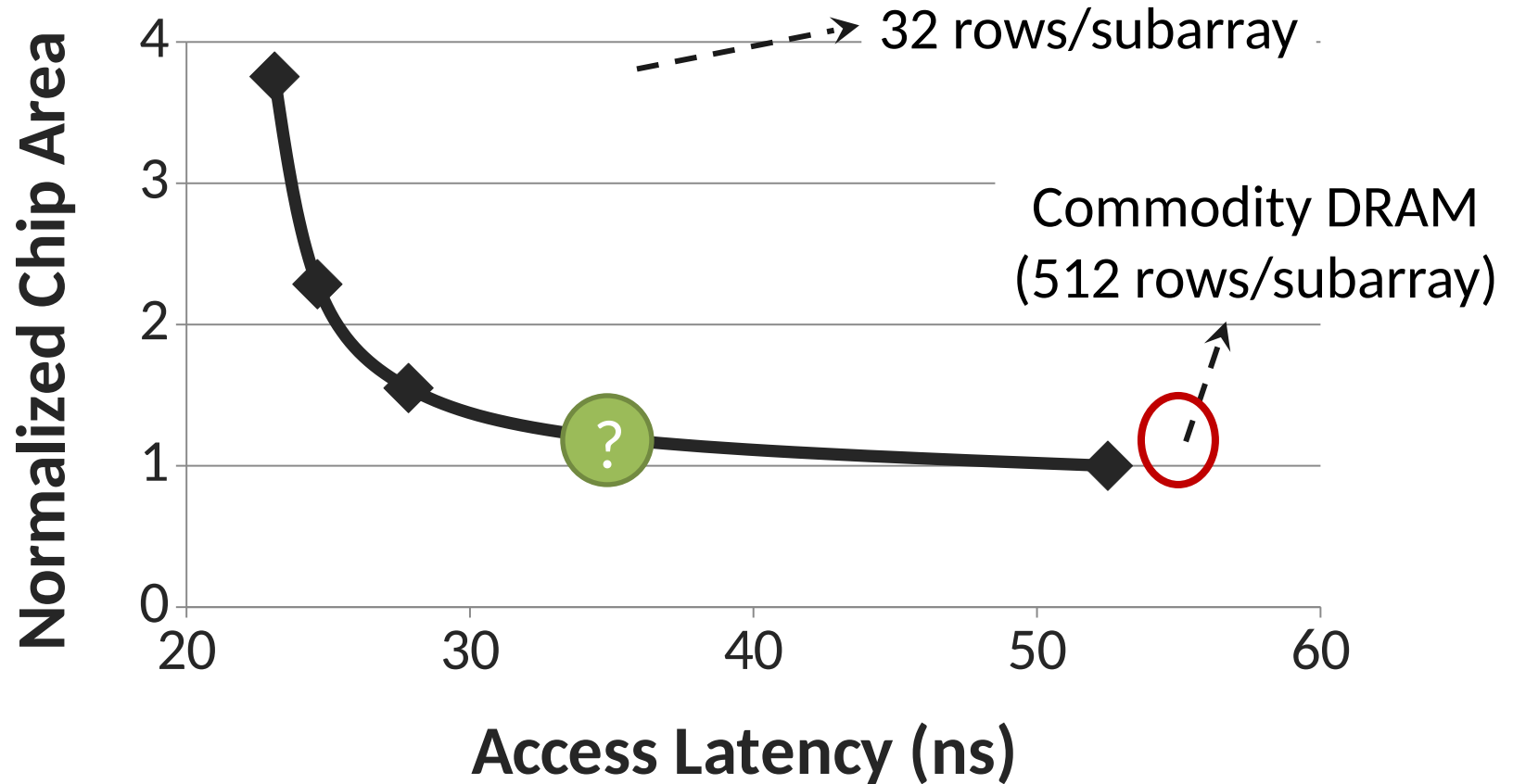


Smaller subarrays => larger chip area

Chip Area vs. Access Latency

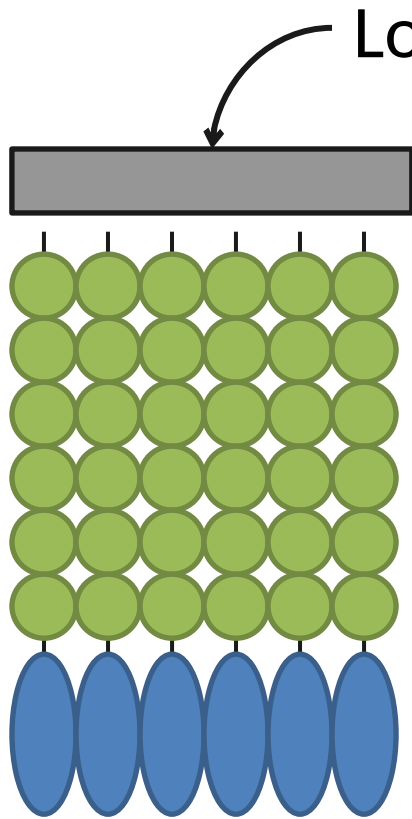


Chip Area vs. Access Latency



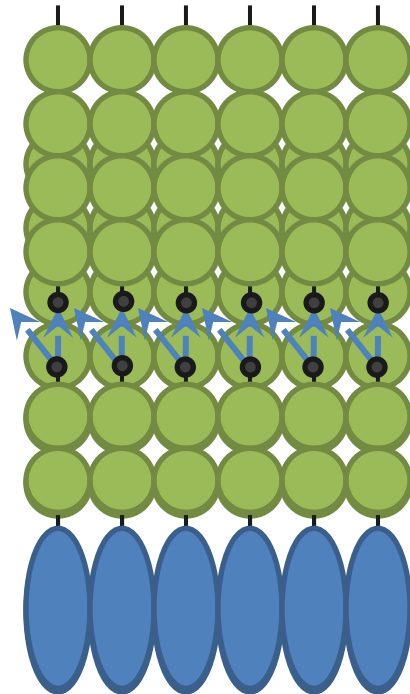
How to enable low latency without high area overhead?

New Proposal

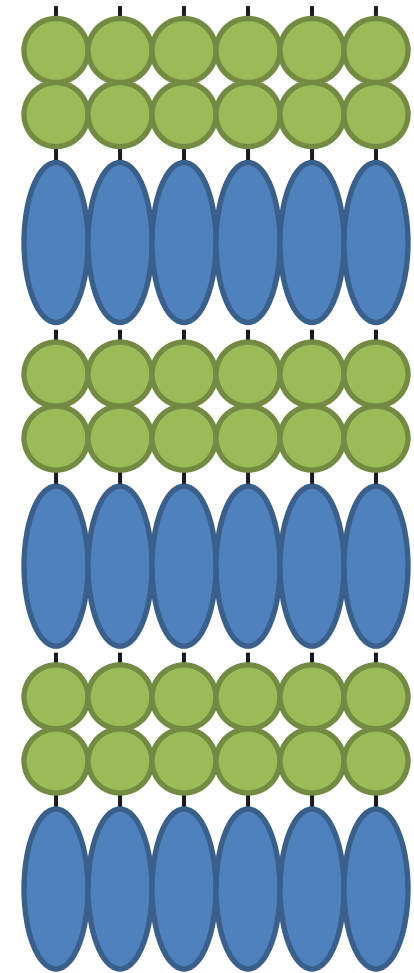


Large Subarray

Low area cost

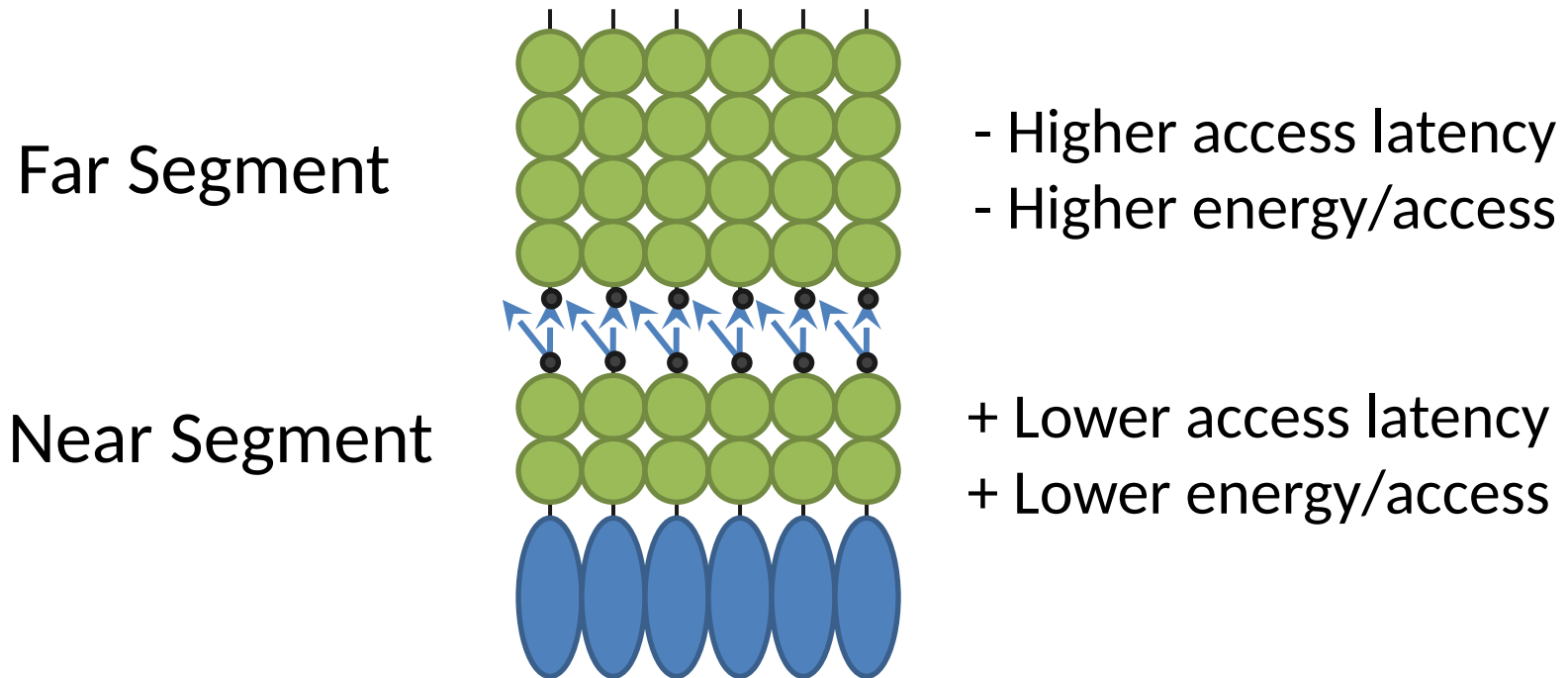


Our Proposal



Small Subarray

Tiered-Latency DRAM

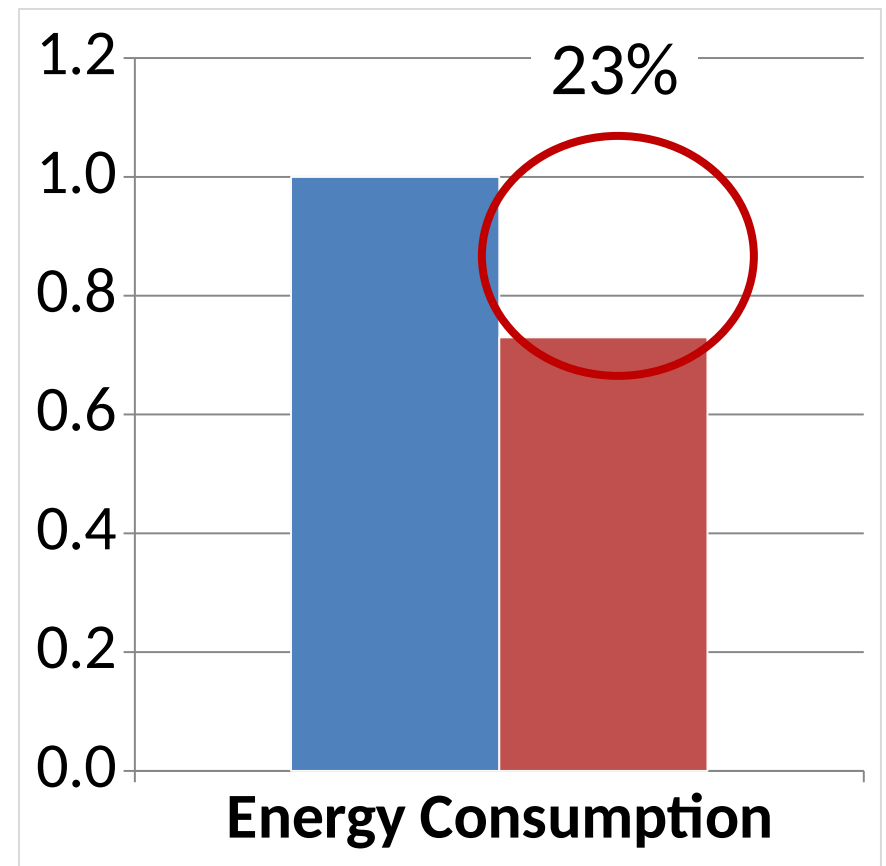
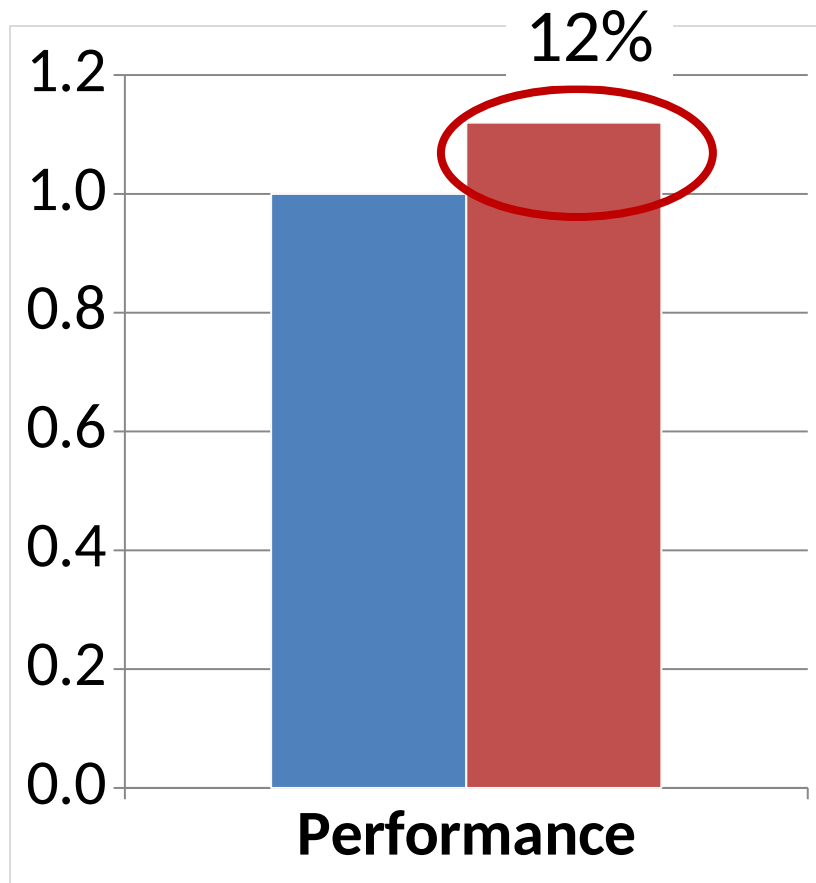


Map frequently accessed data to near segment

Results Summary

■ Commodity DRAM

■ Tiered-Latency DRAM



Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture

Donghyuk Lee, Yoongu Kim, Vivek Seshadri,
Jamie Liu, Lavanya Subramanian, Onur Mutlu

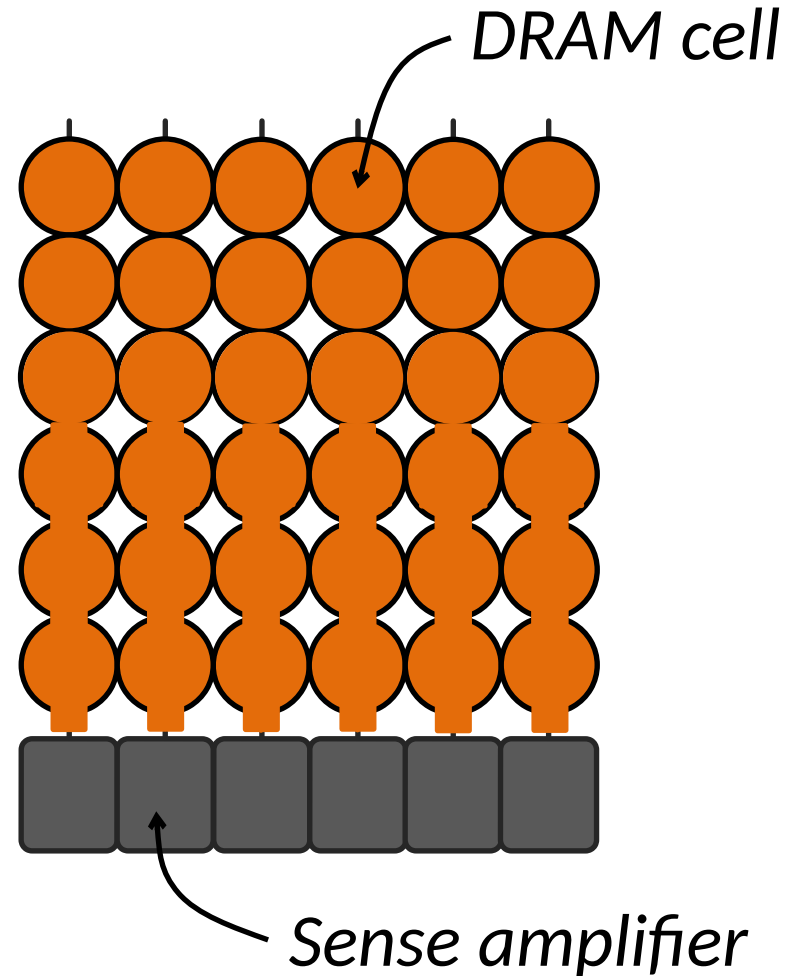
Published in the proceedings of 19th IEEE International
Symposium on

High Performance Computer Architecture 2013

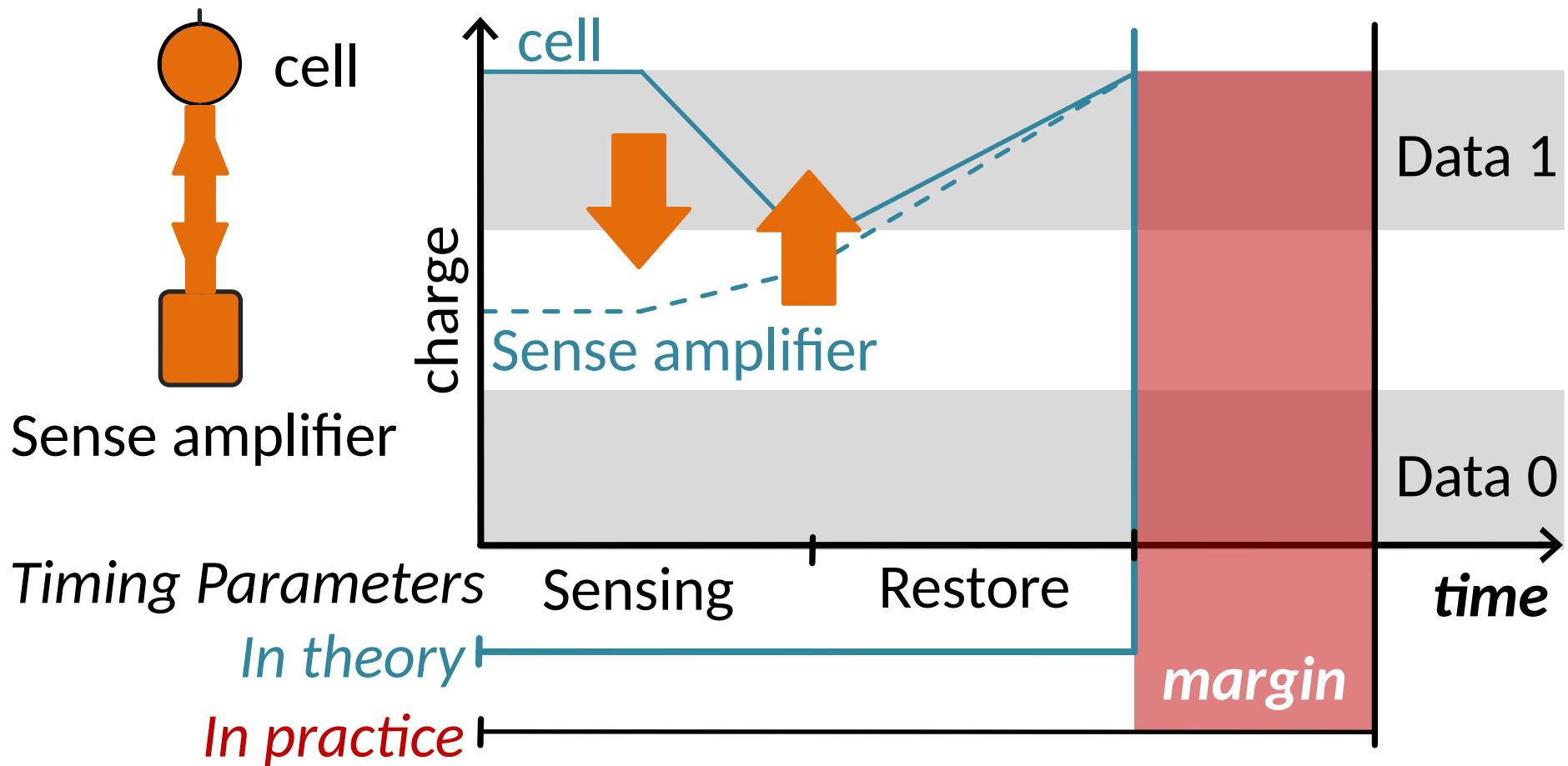
DRAM Stores Data as Charge

Three steps of charge movement

1. Sensing
2. Restore
3. Precharge



DRAM Charge over Time



Why does DRAM need the extra timing margin?

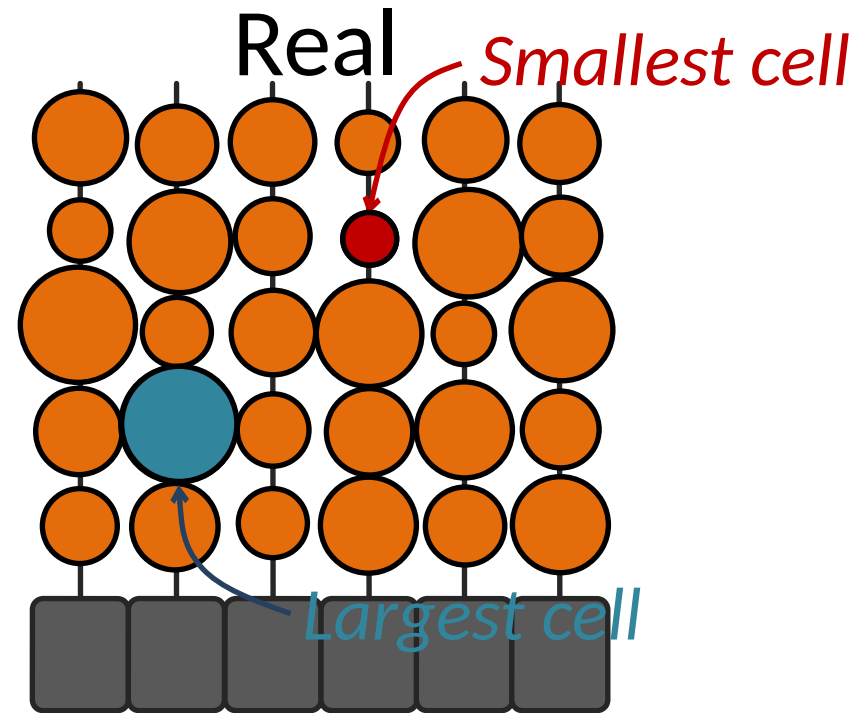
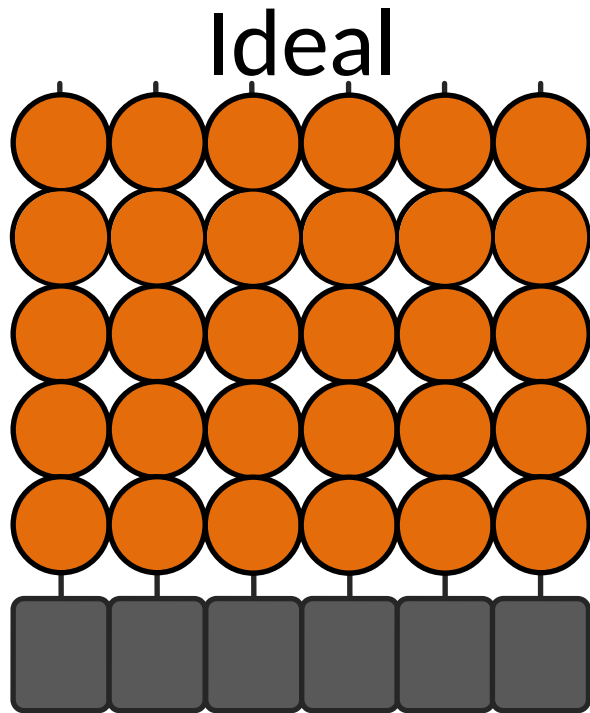
Two Reasons for Timing Margin

1. Process Variation

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

2. Temperature Dependence

DRAM Cells are Not Equal



Same size	□	Different size	□
Same charge	□	Different charge	□
Same latency	□	Different latency	□
Large variation in cell size	□		
Large variation in charge	□		
Large variation in access latency	□		

Two Reasons for Timing Margin

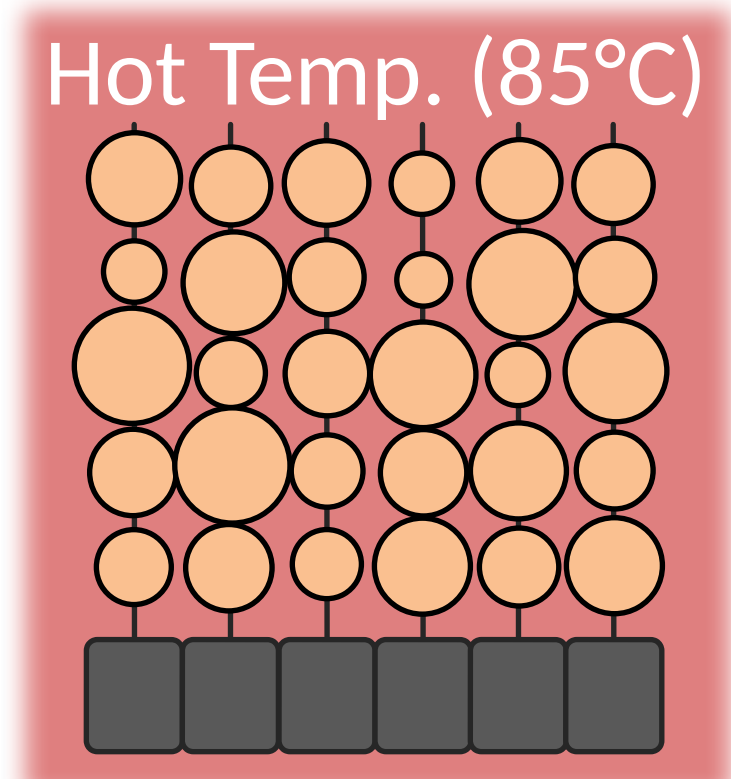
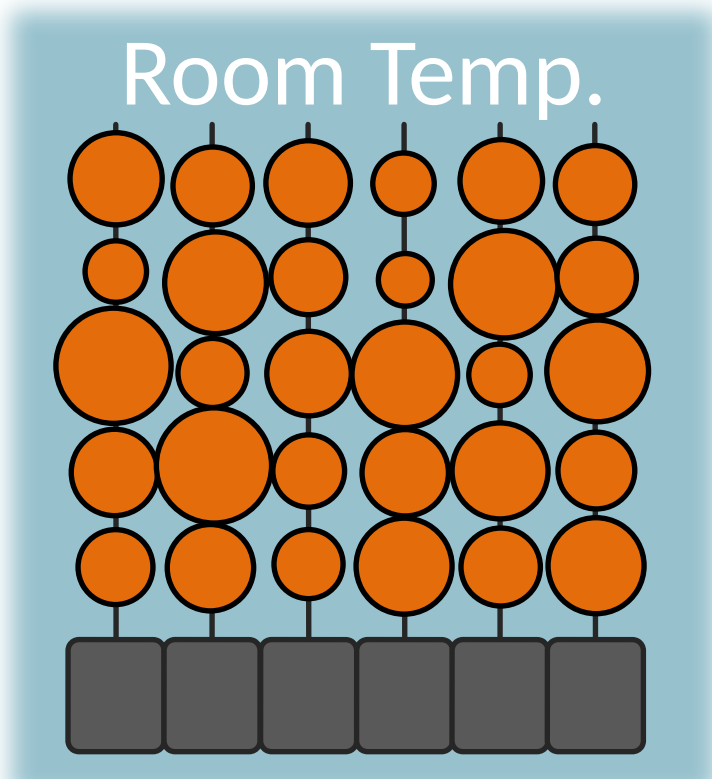
1. Process Variation

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

2. Temperature Dependence

- DRAM leaks more charge at higher temperature
- Leads to extra timing margin when operating at low temperature

Charge Leakage Temperature



Cells store small charge at high temperature **Small leakage**
and large charge at low temperature **Large leakage**
□ 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
 - Can lower latency for the common case

DRAM Testing Infrastructure

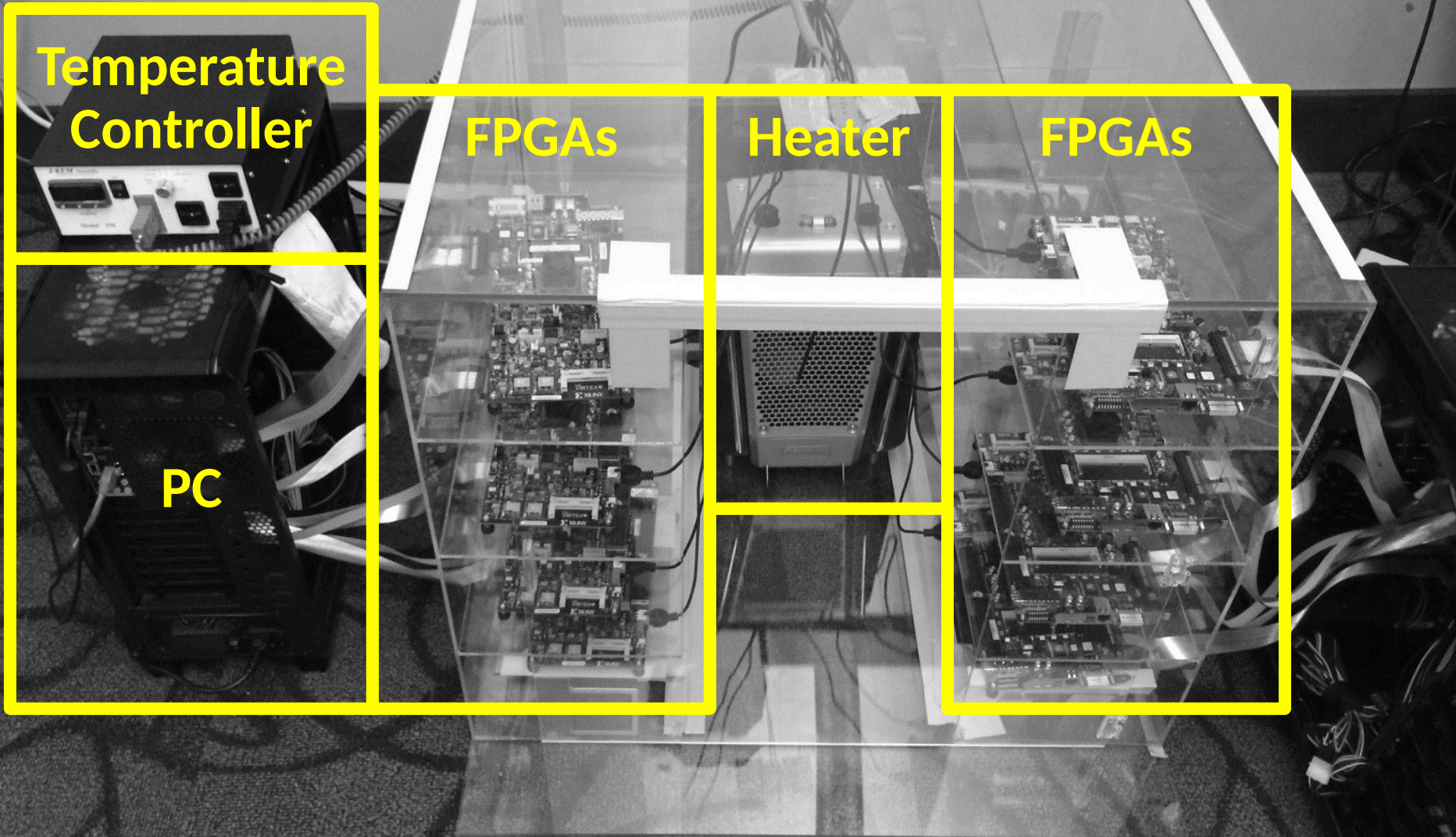
Temperature
Controller

FPGAs

Heater

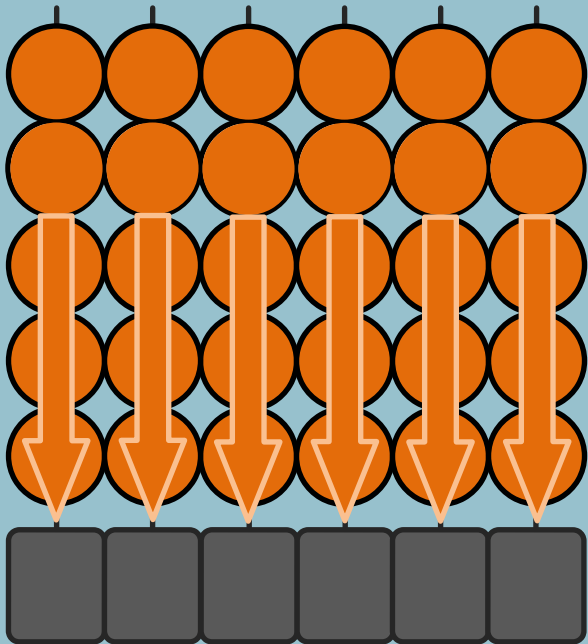
FPGAs

PC



Obs 1. Faster Sensing

Typical DIMM at Low Temperature



More charge

Strong charge flow

Faster sensing

115 DIMM characterization

Timing
(t_{RCD})

17% ↓

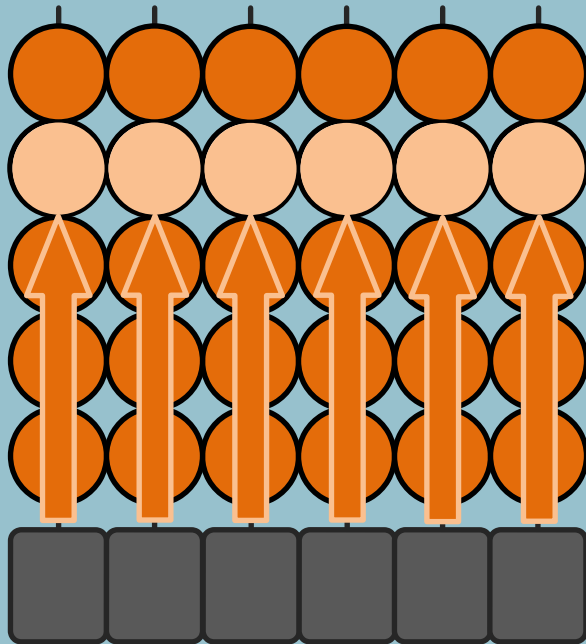
No Errors

Typical DIMM at Low Temperature

□ *More charge* □ *Faster sensing*

Obs 2. Reducing Restore Time

Typical DIMM at Low Temperature



Larger cell &
Less leakage □
Extra charge

No need to fully
restore charge

115 DIMM
characterization

Read (t_{RAS})

37% ↓

Write (t_{WR})

54% ↓

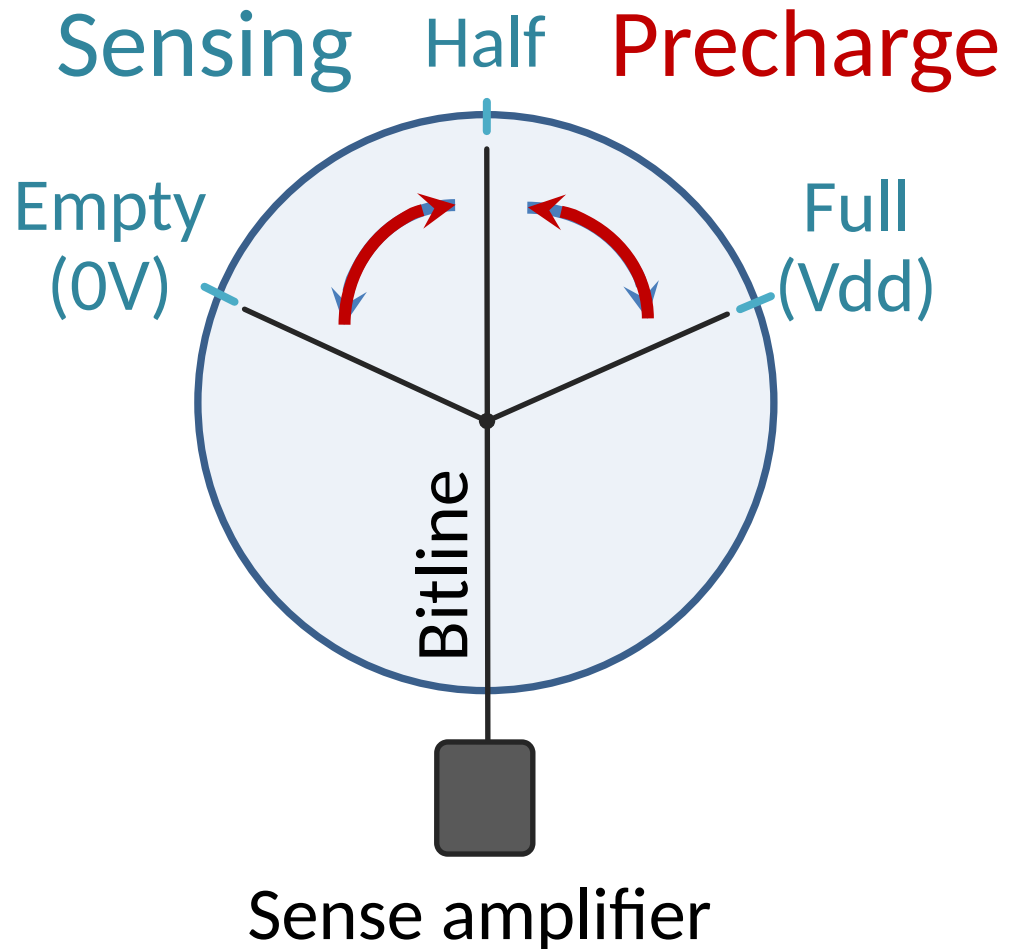
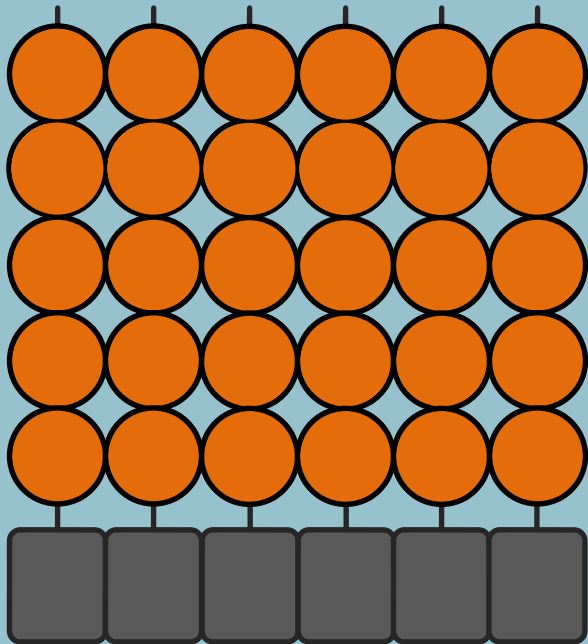
No Errors

Typical DIMM at lower temperature

□ More charge □ Restore time reduction

Obs 3. Reducing Precharge Time

Typical DIMM at Low Temperature



Precharge ? - Setting bitline to half-full charge

Obs 3. Reducing Precharge Time

Access empty cell

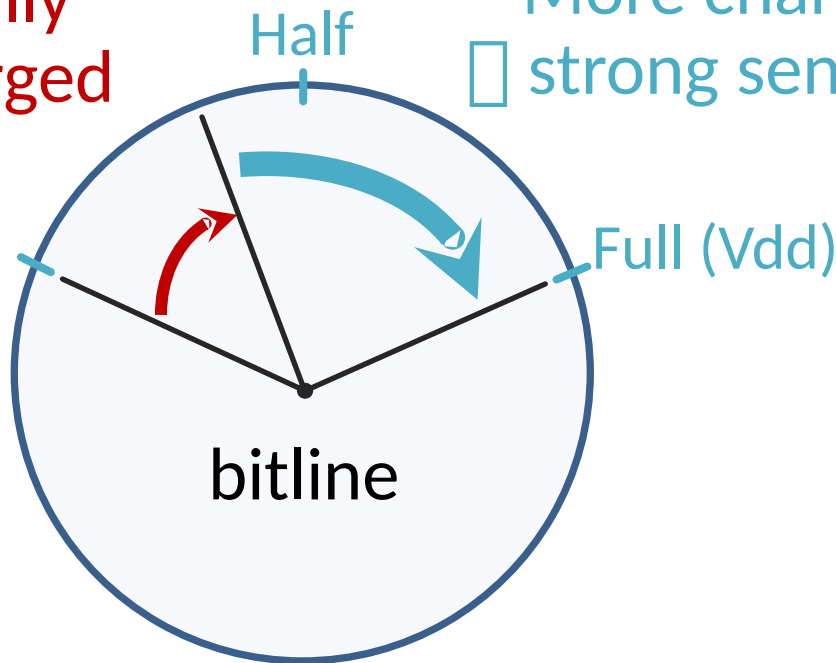
Access full cell

115 DIMM
characterization

Not fully
precharged

More charge
□ strong sensing

Empty
(0V)



Timing
(t_{RP})

35% ↓

No Errors

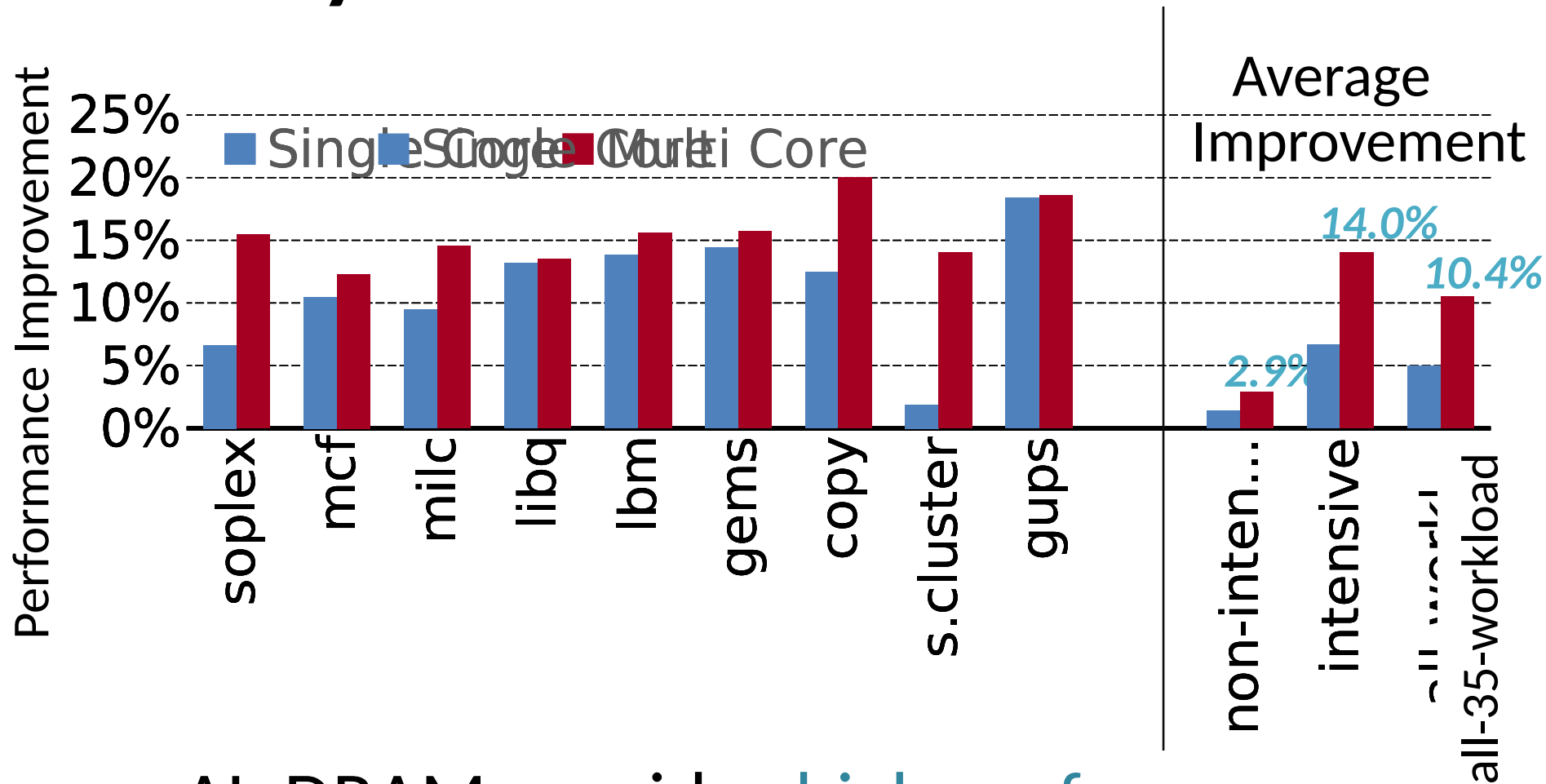
Typical DIMM at Lower Temperature

□ More charge □ Precharge time reduction

Adaptive-Latency DRAM

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

Real System Evaluation



AL-DRAM provides high performance improvement, greater for multi-core workloads

Summary: AL-DRAM

- Observation
 - DRAM timing parameters are dictated by the worst-case cell (smallest cell at highest temperature)
- Our Approach: *Adaptive-Latency DRAM (AL-DRAM)*
 - Optimizes DRAM timing parameters for *the common case* (typical DIMM operating at low temperatures)
- Analysis: Characterization of 115 DIMMs
 - Great potential to *lower DRAM timing parameters (17 – 54%)* without any errors
- Real System Performance Evaluation
 - Significant *performance improvement (14%* for memory-intensive workloads) without errors (*33* days)

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

Donghyuk Lee, Yoongu Kim,

Gennady Pekhimenko, Samira Khan, Vivek
Seshadri, Kevin Chang, and Onur Mutlu

Published in the proceedings of 21st

**International Symposium on High Performance
Computer Architecture 2015**

Outline

1. What is DRAM?

2. DRAM Internal Organization

3. Problems and Solutions

– Latency (Tiered-Latency DRAM, HPCA 2013;
Adaptive-Latency DRAM, HPCA 2015)

– Parallelism (Subarray-level Parallelism, ISCA 2012)

Parallelism: Demand vs. Supply

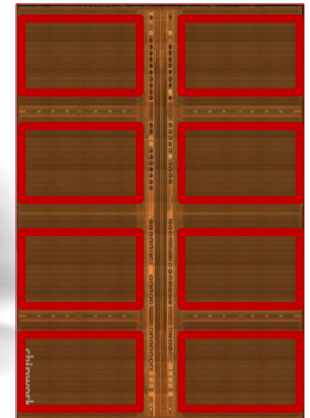
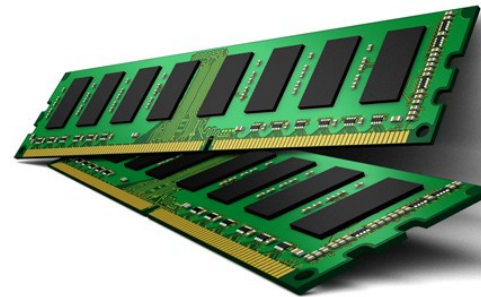
Demand

Supply

Out-of-order
Execution

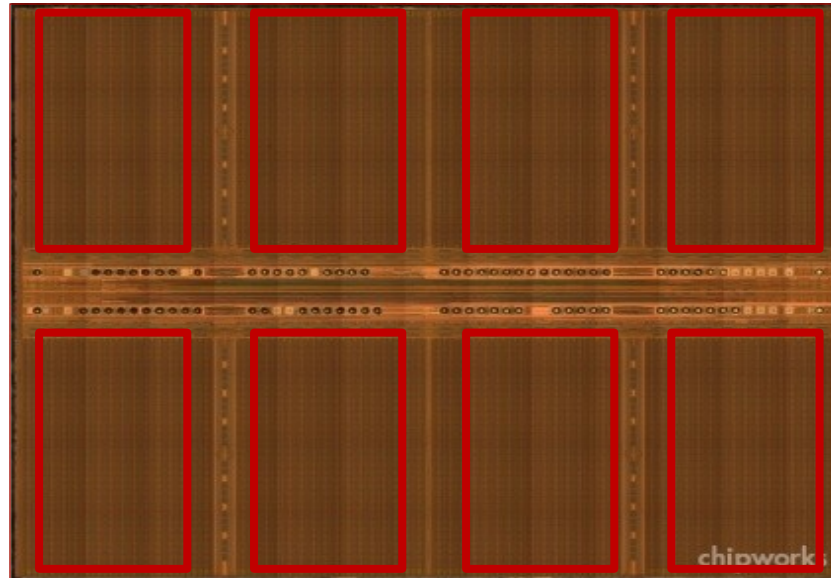
Multi-cores

Prefetchers



Multiple
Banks

Increasing Number of Banks?



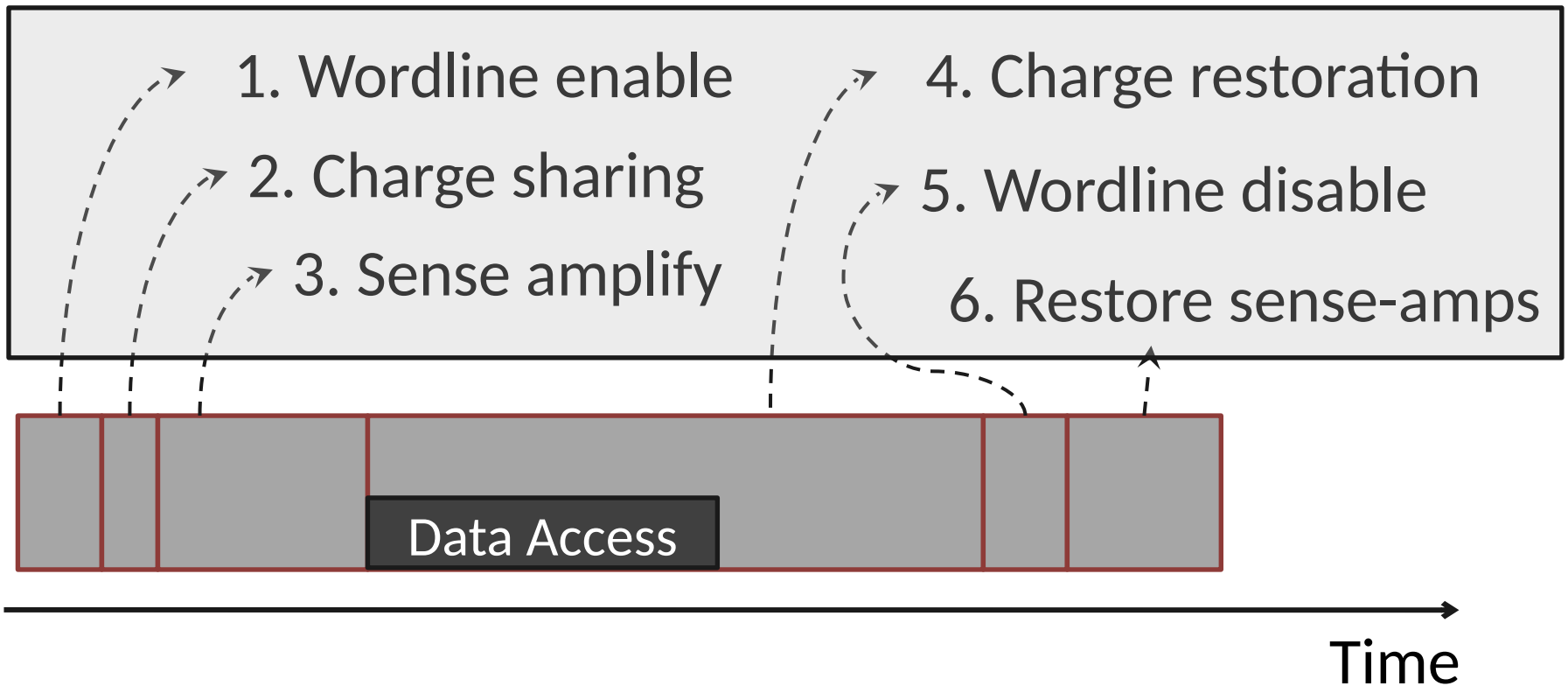
Adding more banks → Replication of shared structures

Replication → Cost

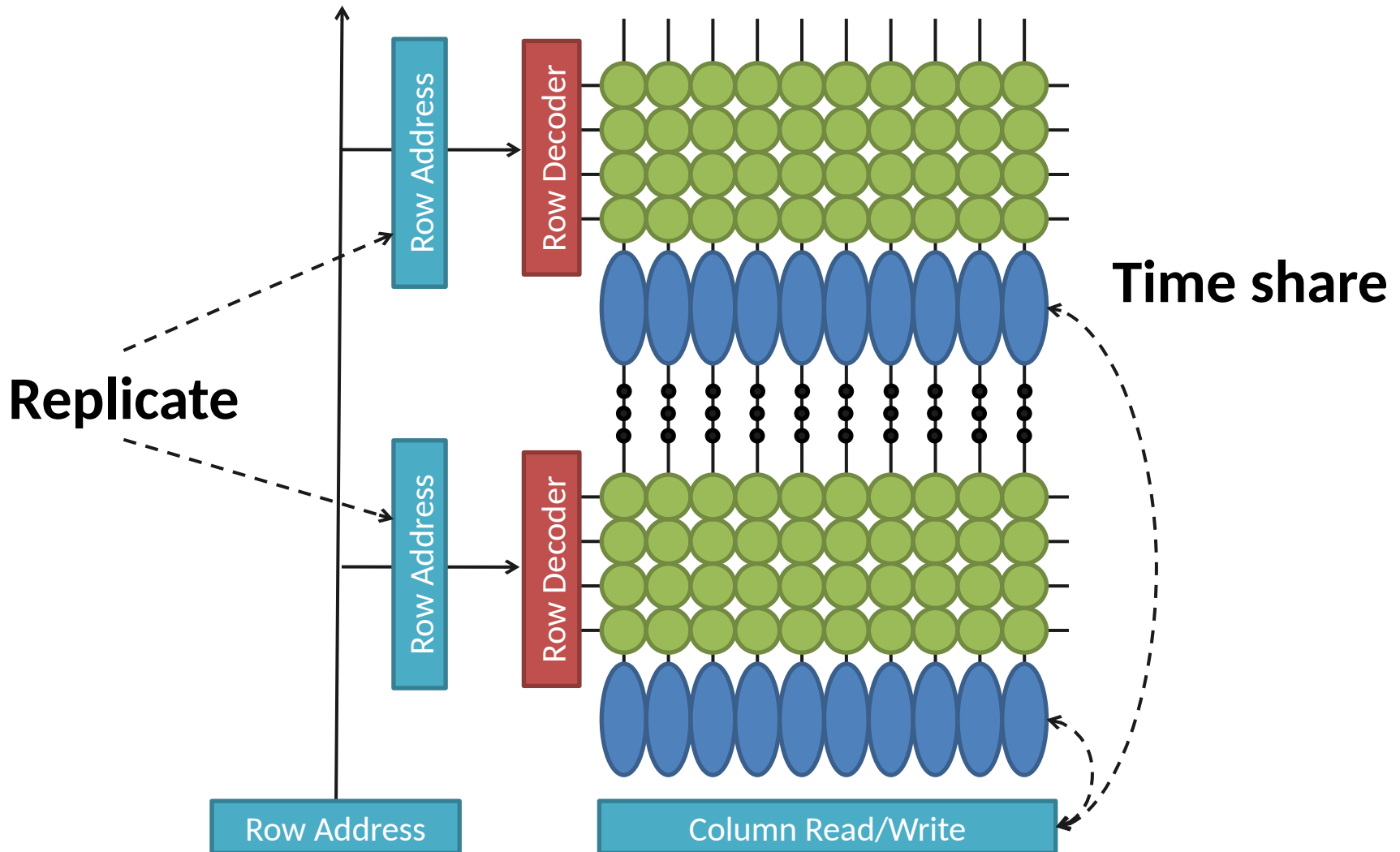
How to improve available parallelism within DRAM?

Our Observation

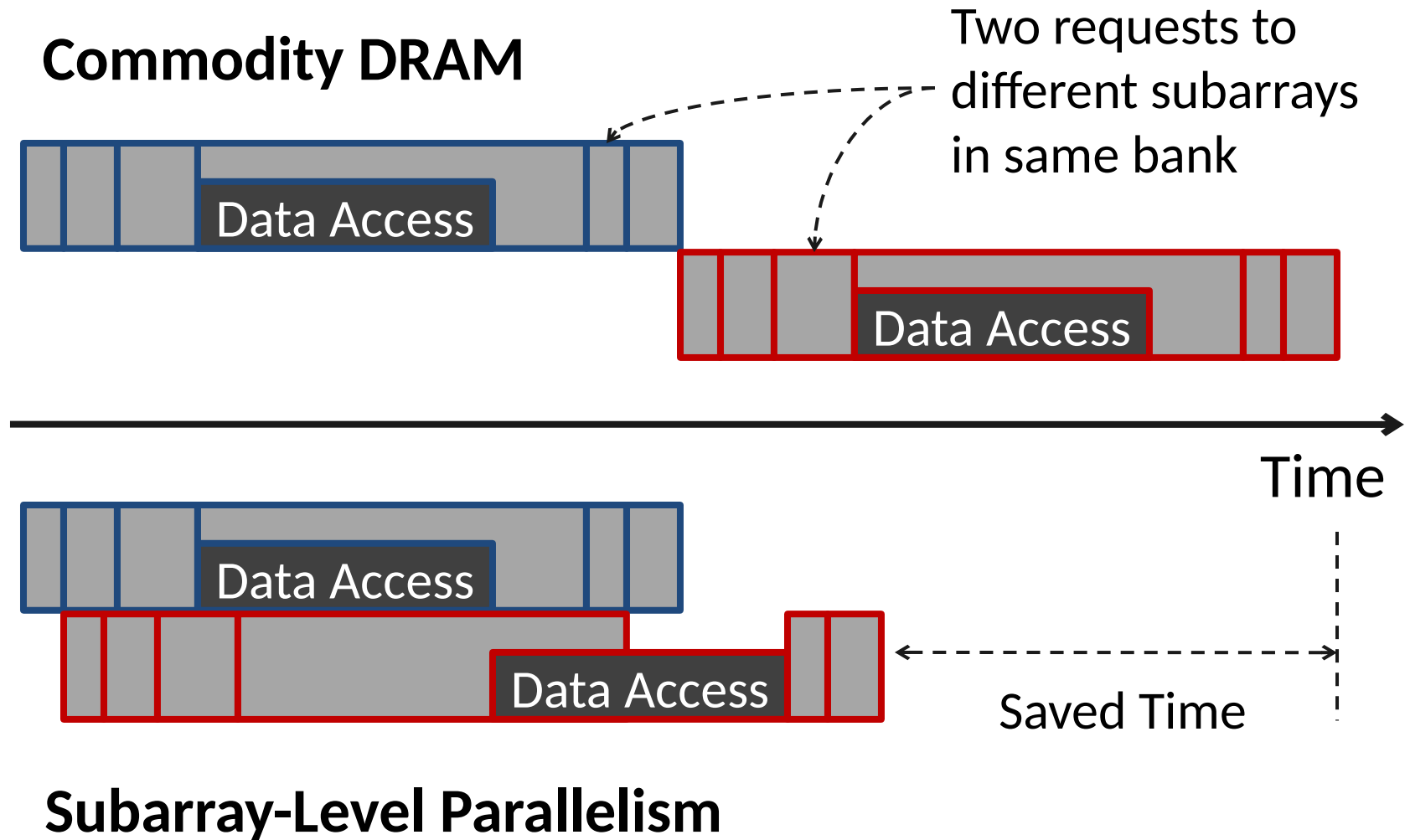
Local to a subarray



Subarray-Level Parallelism



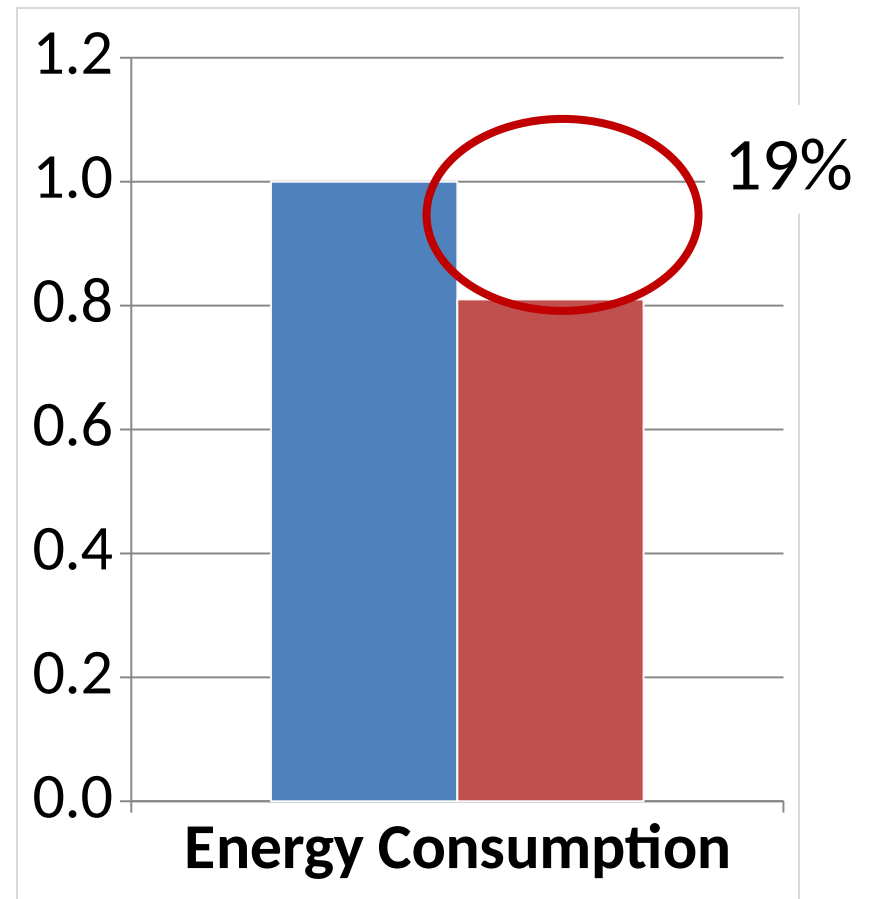
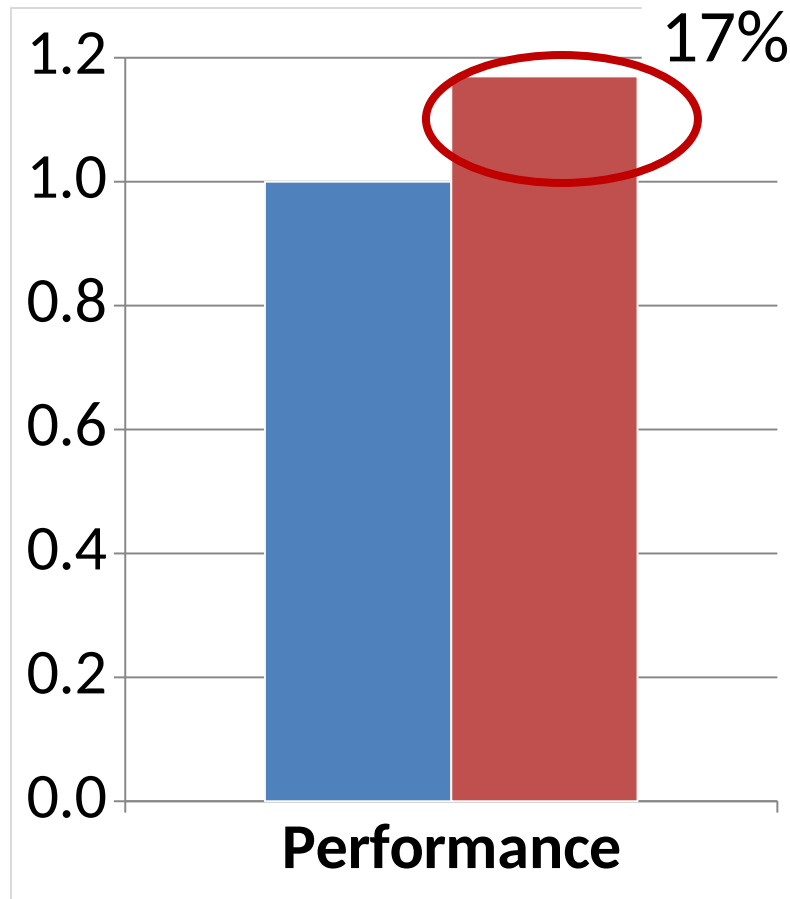
Subarray-Level Parallelism: Benefits



Results Summary

■ Commodity DRAM

■ Subarray-Level Parallelism



A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM

Yoongu Kim, Vivek Seshadri, Donghyuk Lee,
Jamie Liu, Onur Mutlu

Published in the proceedings of 39th

**International Symposium on Computer Architecture
2012**

CSC 2224: Parallel Computer Architecture and Programming Main Memory Fundamentals

Prof. Gennady Pekhimenko

University of Toronto

Fall 2022

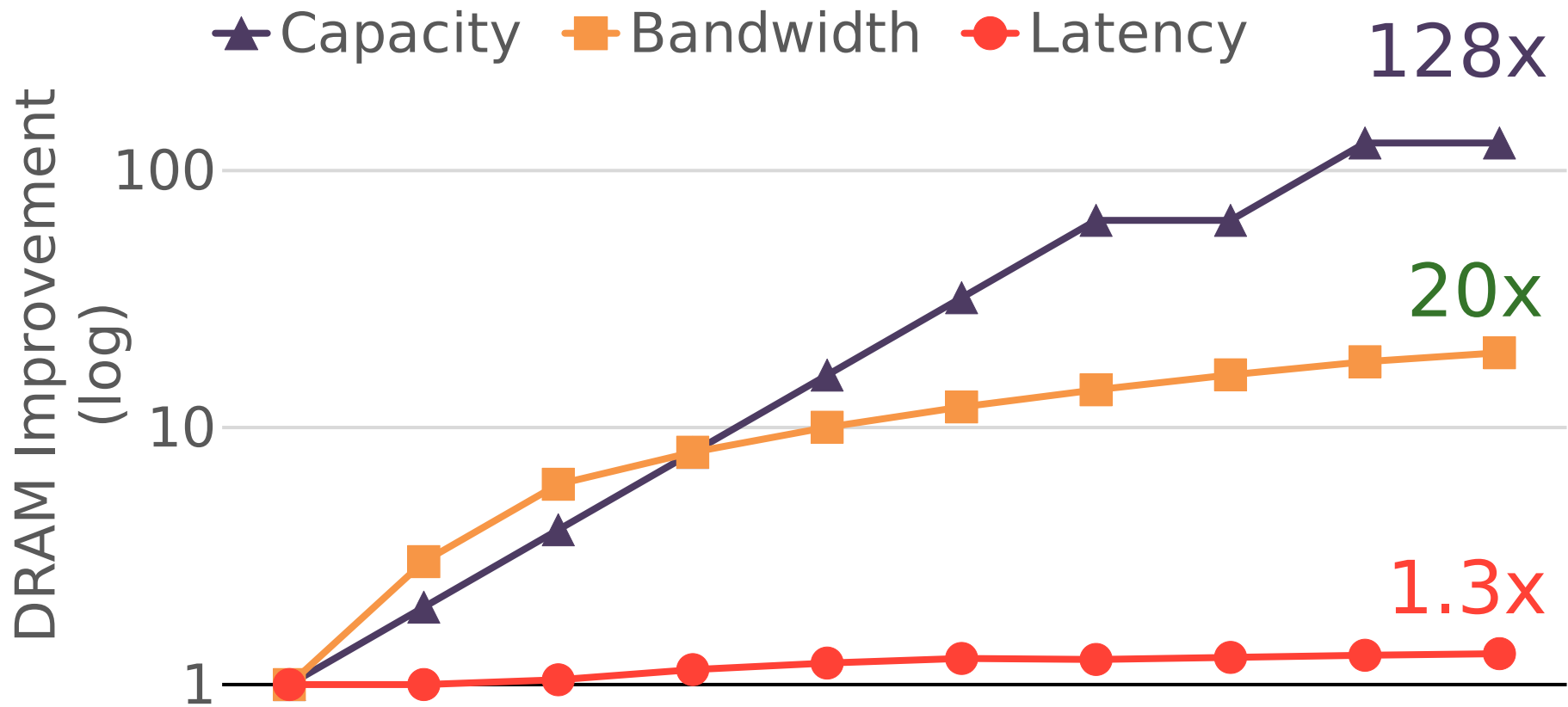
*The content of this lecture is adapted from the slides of
Vivek Seshadri, Donghyuk Lee, Yoongu Kim,
and lectures of Onur Mutlu @ ETH and CMU*

Review #5

Flipping Bits in Memory Without Accessing Them

Yoongu Kim et al., *ISCA 2014*

Review: Memory Latency Lags Behind

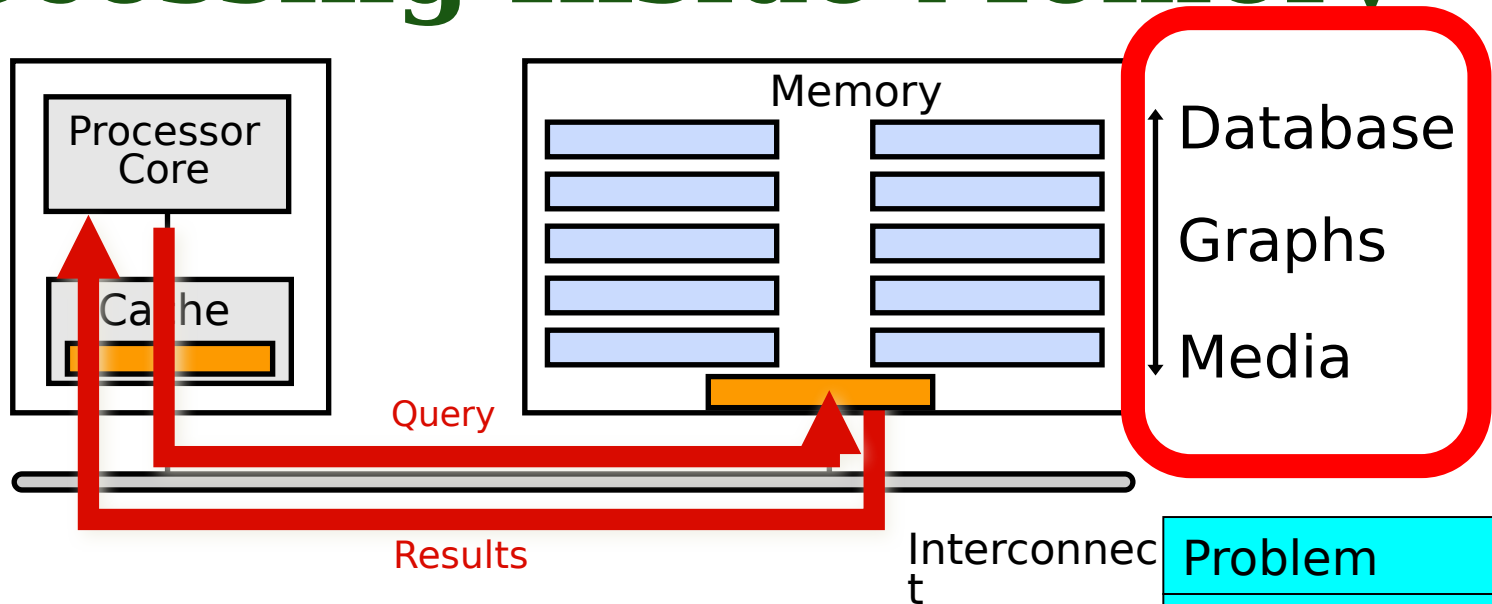


Memory latency remains almost constant

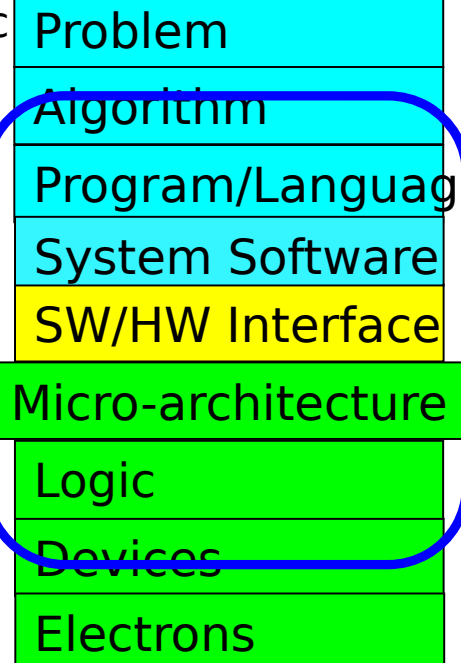
We Need A Paradigm Shift To ...

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

Processing Inside Memory



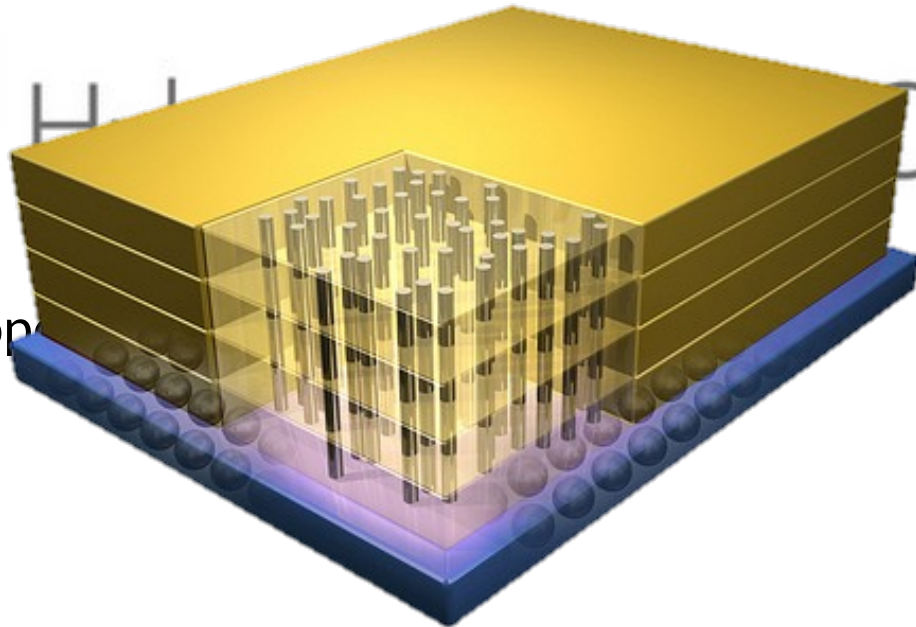
- Many questions ... How do we design the:
 - compute-capable memory & controllers?
 - processor chip?
 - Software and hardware interfaces?
 - system software and languages?
 - algorithms?



Why In-Memory Computation Today?



□ Industry op



- 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

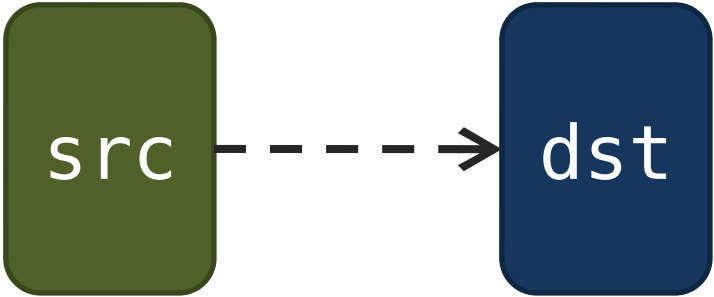
- 1. **Minimally change DRAM** to enable simple yet powerful computation primitives
 - [RowClone: Fast and Efficient In-DRAM Copy and Initialization of Bulk Data](#) (Seshadri et al., MICRO 2013)
 - [Fast Bulk Bitwise AND and OR in DRAM](#) (Seshadri et al., IEEE CAL 2015)
 - [Gather-Scatter DRAM: In-DRAM 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

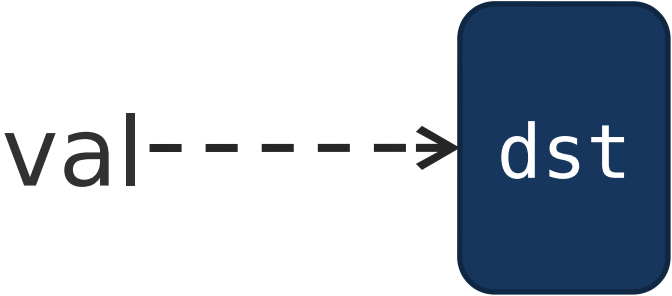
- DRAM has great capability to perform **bulk data movement and computation** internally with small changes
 - Can exploit internal bandwidth to move data
 - Can exploit analog computation capability
 - ...
- Examples: RowClone, In-DRAM AND/OR, Gather/Scatter DRAM
 - RowClone: Fast and Efficient In-DRAM 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

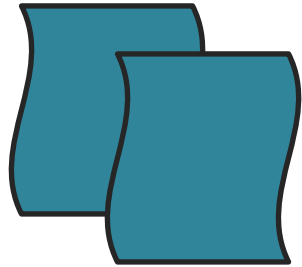
Architecture Support for Improving Bulk Memory Copying and Initialization Performance

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

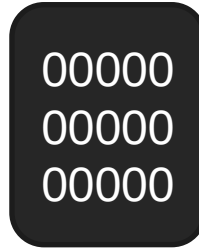
Li Zhao, Ravishankar Iyer
Intel Labs
Intel Corporation
Hillsboro, USA

Bulk Data Copy and Initialization

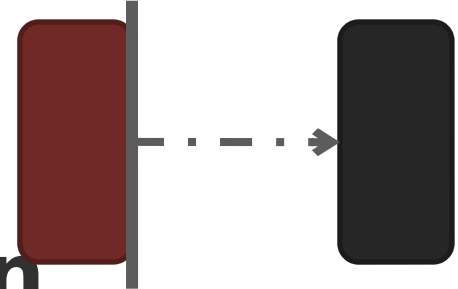
memmove & memcpy: 5% cycles in Google's datacenter [ISCA'15]



Forking



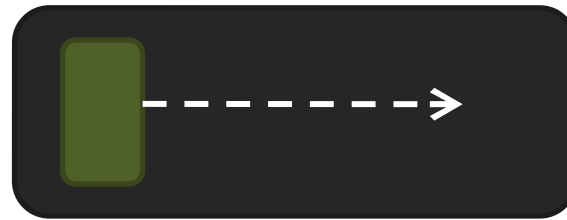
Zero initialization
(e.g., security)



Checkpointing



VM Cloning
Deduplication



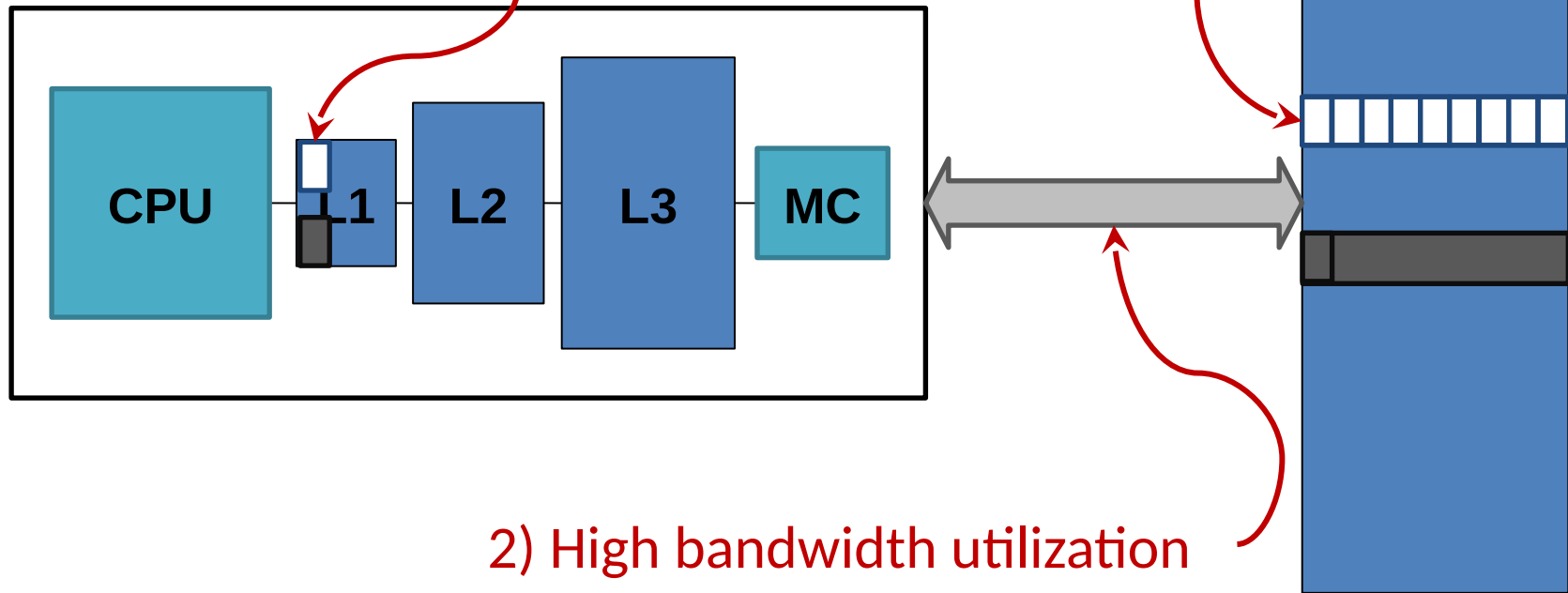
Page Migration

...
Many more

Today's Systems: Bulk Data Copy

1) High latency

3) Cache pollution



2) High bandwidth utilization

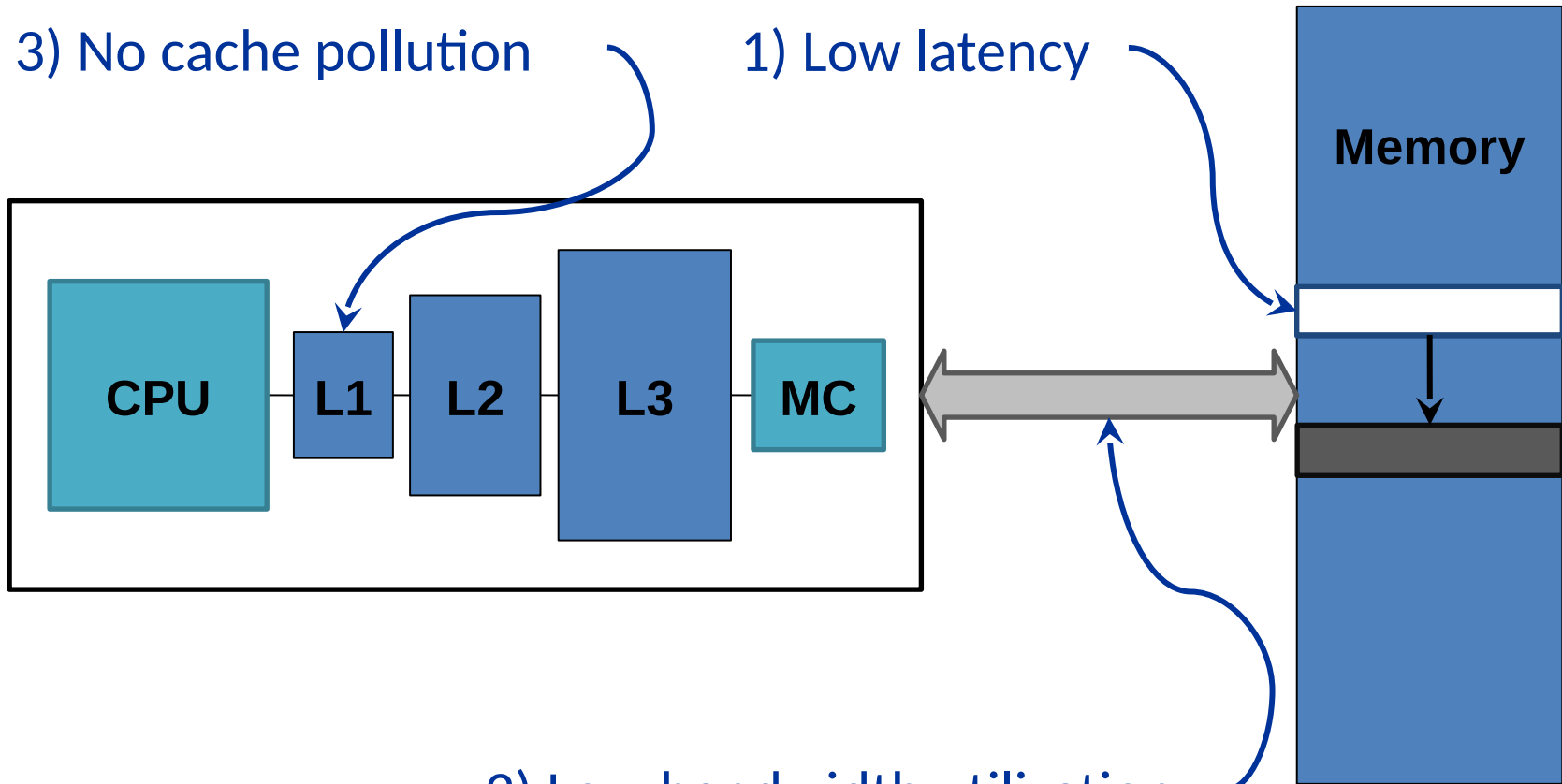
4) Unwanted data movement

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

Future Systems: In-Memory Copy

3) No cache pollution

1) Low latency



2) Low bandwidth utilization

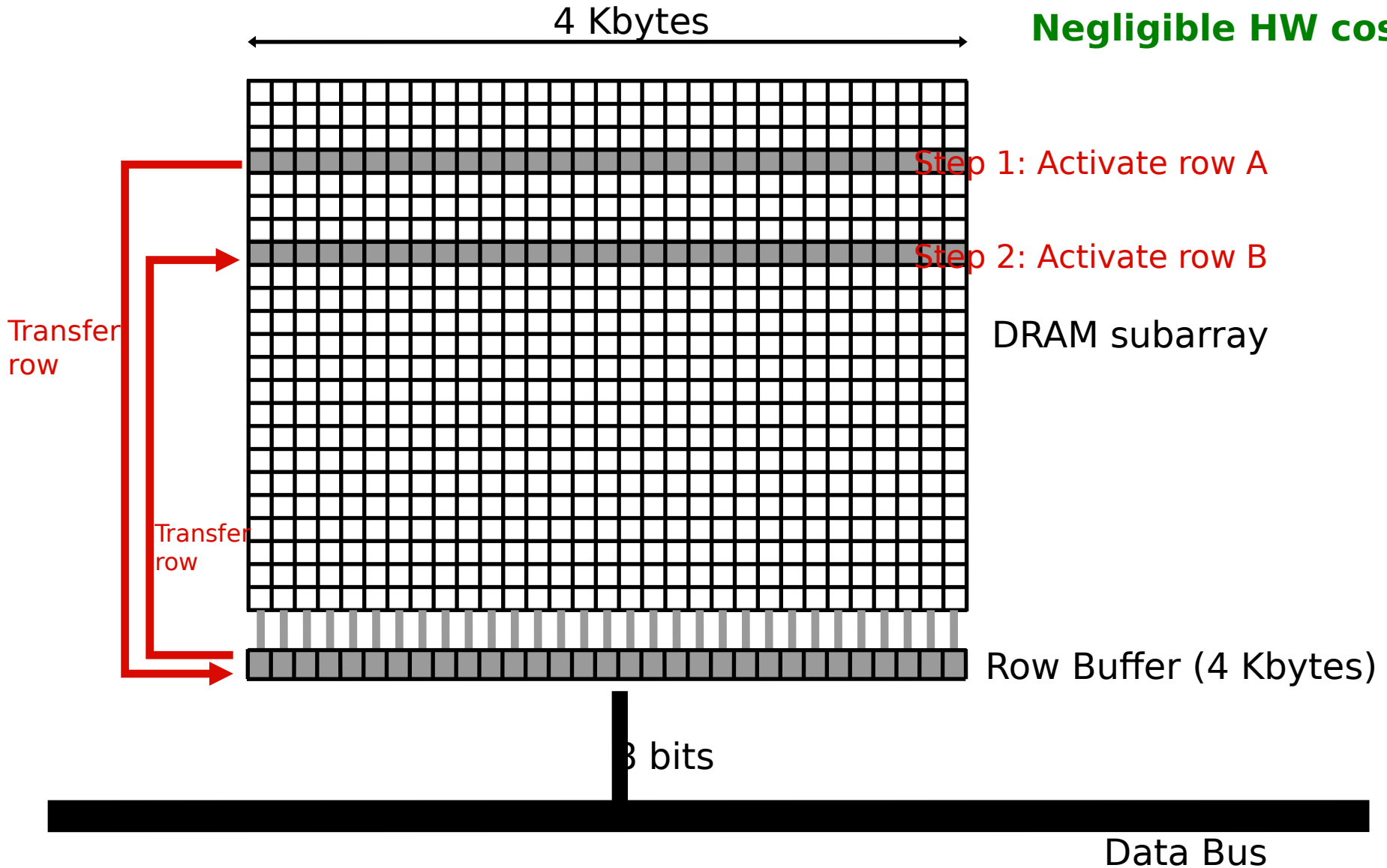
4) No unwanted data movement

1046ns, 3.6uJ

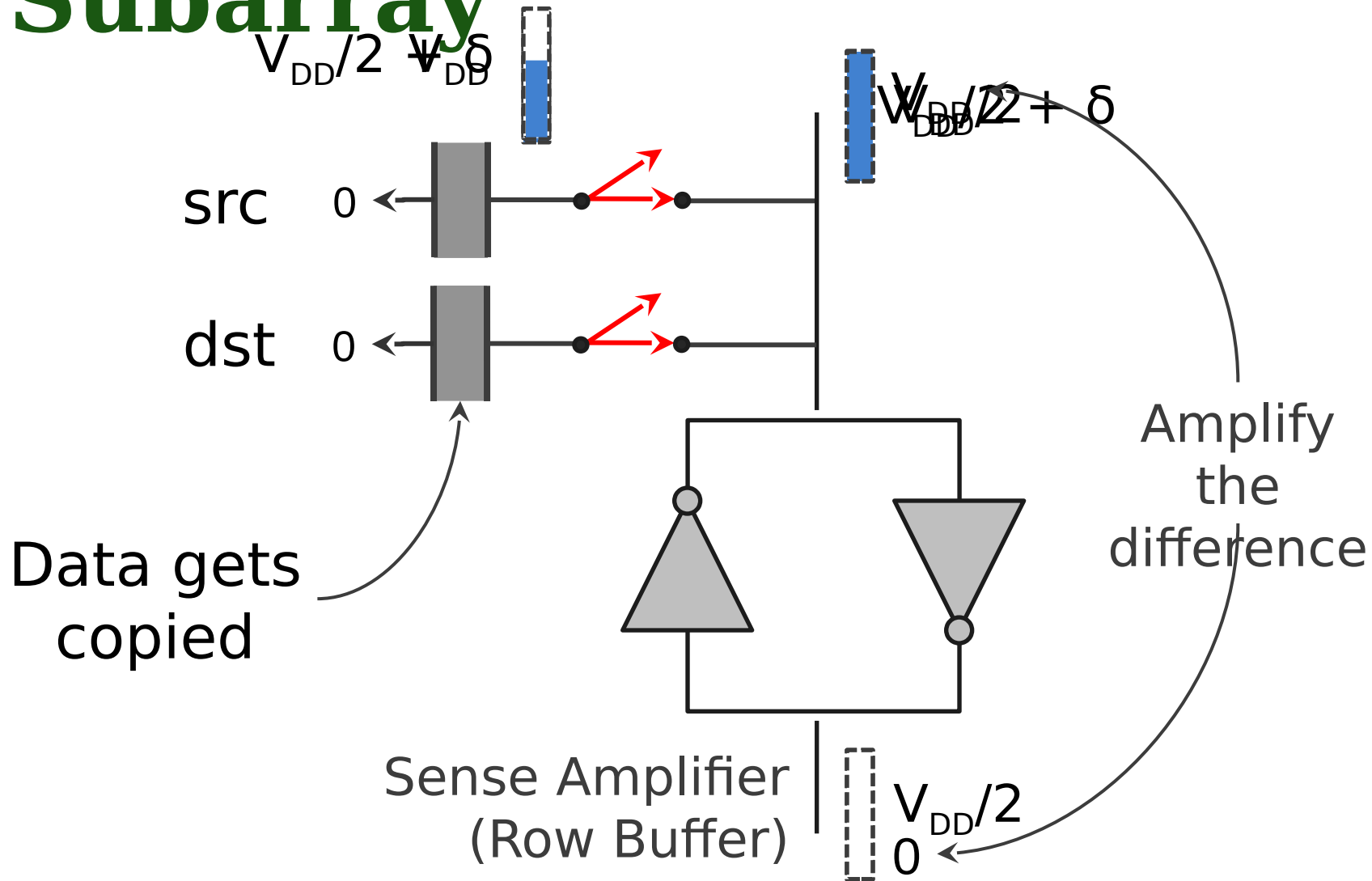
□ 90ns, 0.04uJ

RowClone: In-DRAM Row Copy

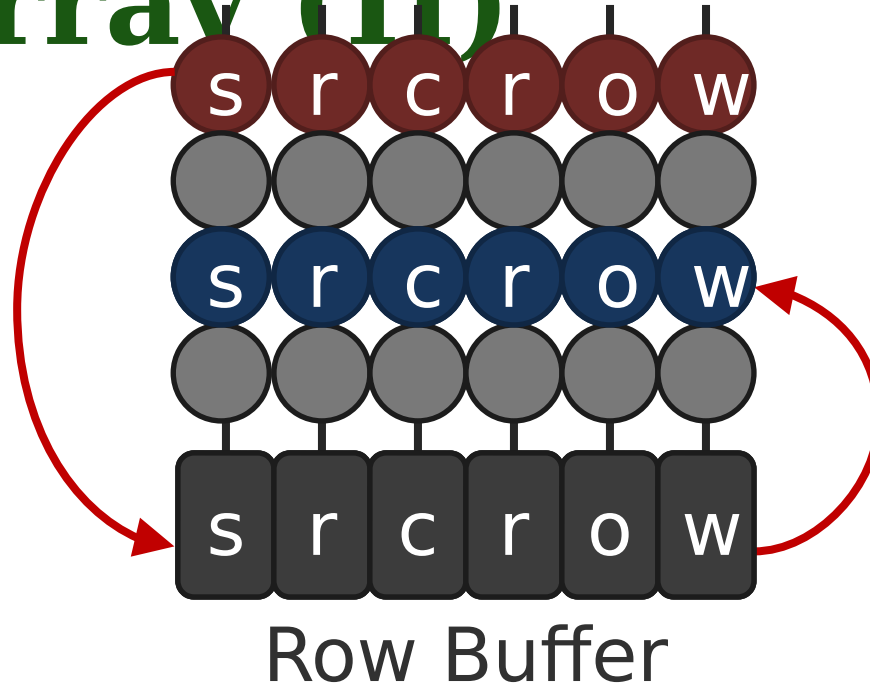
Idea: Two consecutive ACTivates
Negligible HW cost



RowClone: Intra-Subarray

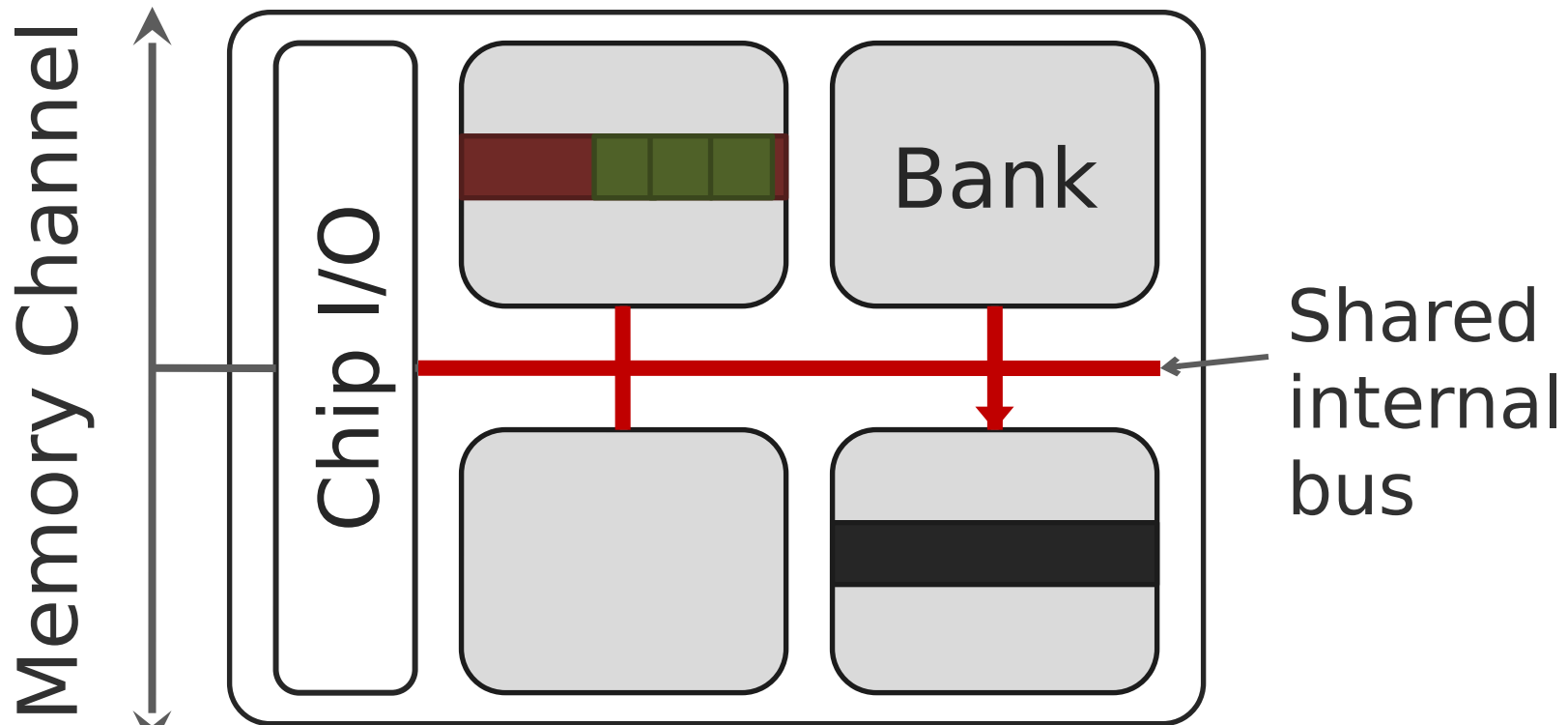


RowClone: Intra-Subarray (II)



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



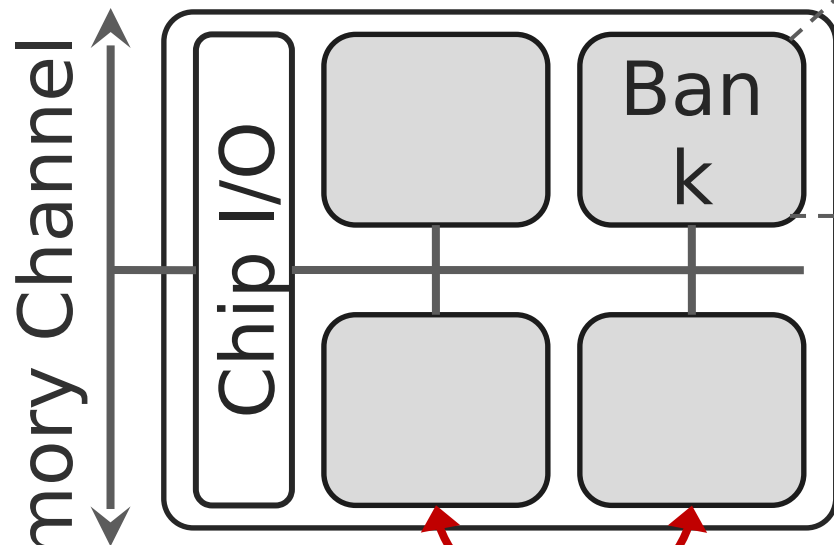
Overlap the latency of the read and the write

1.9X latency reduction, 3.2X energy reduction

Generalized RowClone

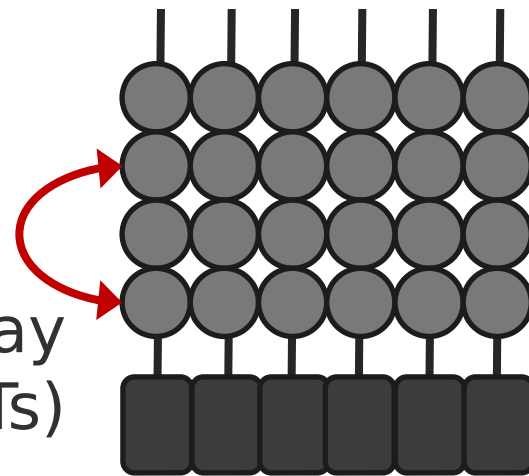
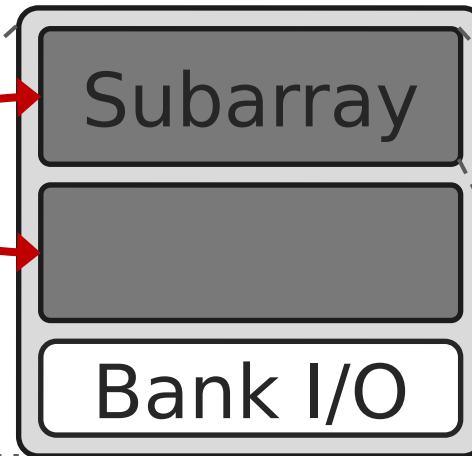
0.01% area cost

Inter Subarray Copy
(Use Inter-Bank Copy Twice)

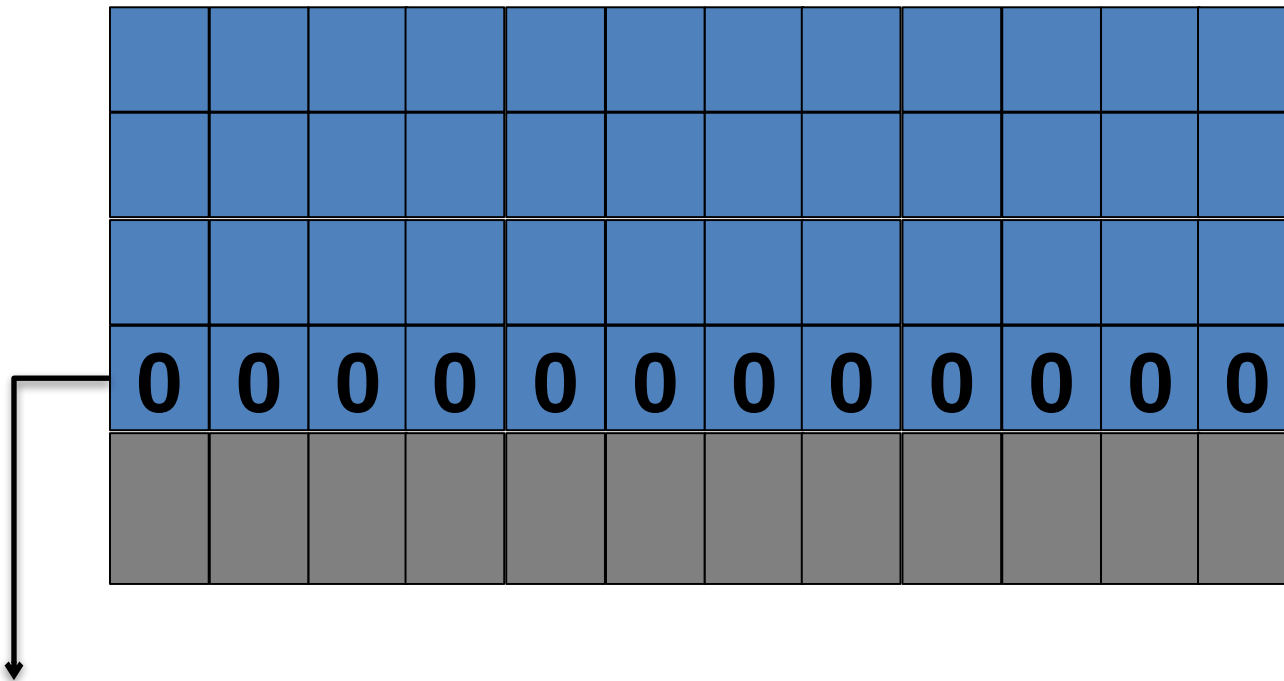


Inter Bank Copy
(Pipelined
Internal RD/WR)

Intra Subarray
Copy (2 ACTs)



RowClone: Fast Row Initialization

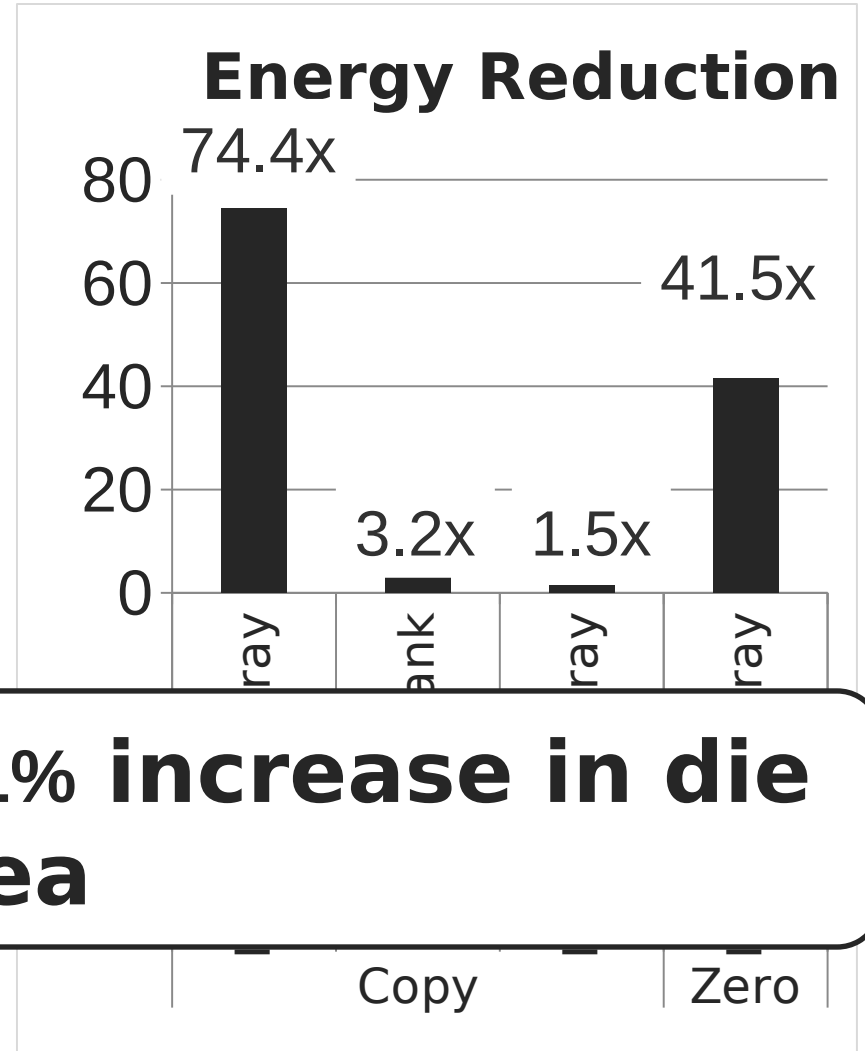
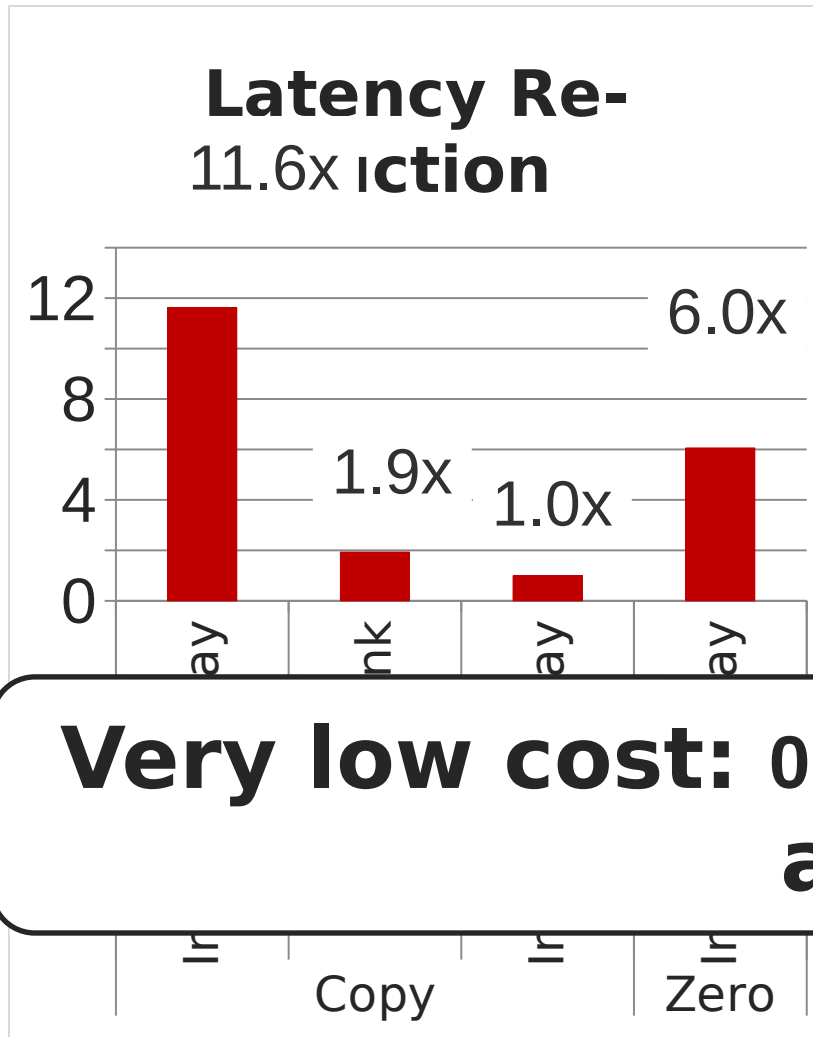


Fix a row at Zero
(0.5% loss in capacity)

RowClone: Bulk Initialization

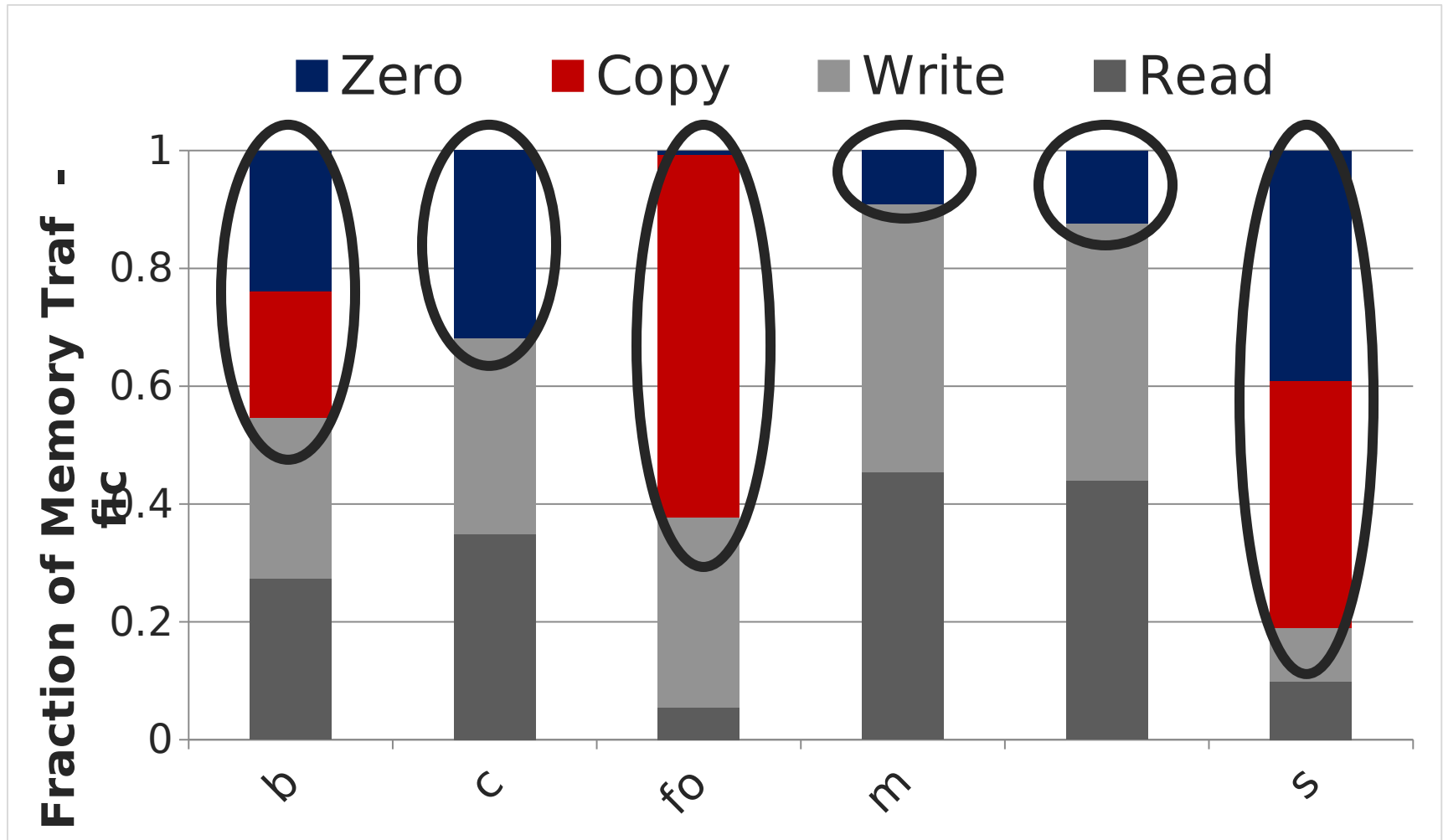
- Initialization with arbitrary data
 - Initialize one row
 - Copy the data to other rows
- Zero initialization (most common)
 - Reserve a row in each subarray (always zero)
 - Copy data from reserved row (FPM mode)
 - **6.0X** lower latency, **41.5X** lower DRAM energy
 - 0.2% loss in capacity

RowClone: Latency & Energy Benefits

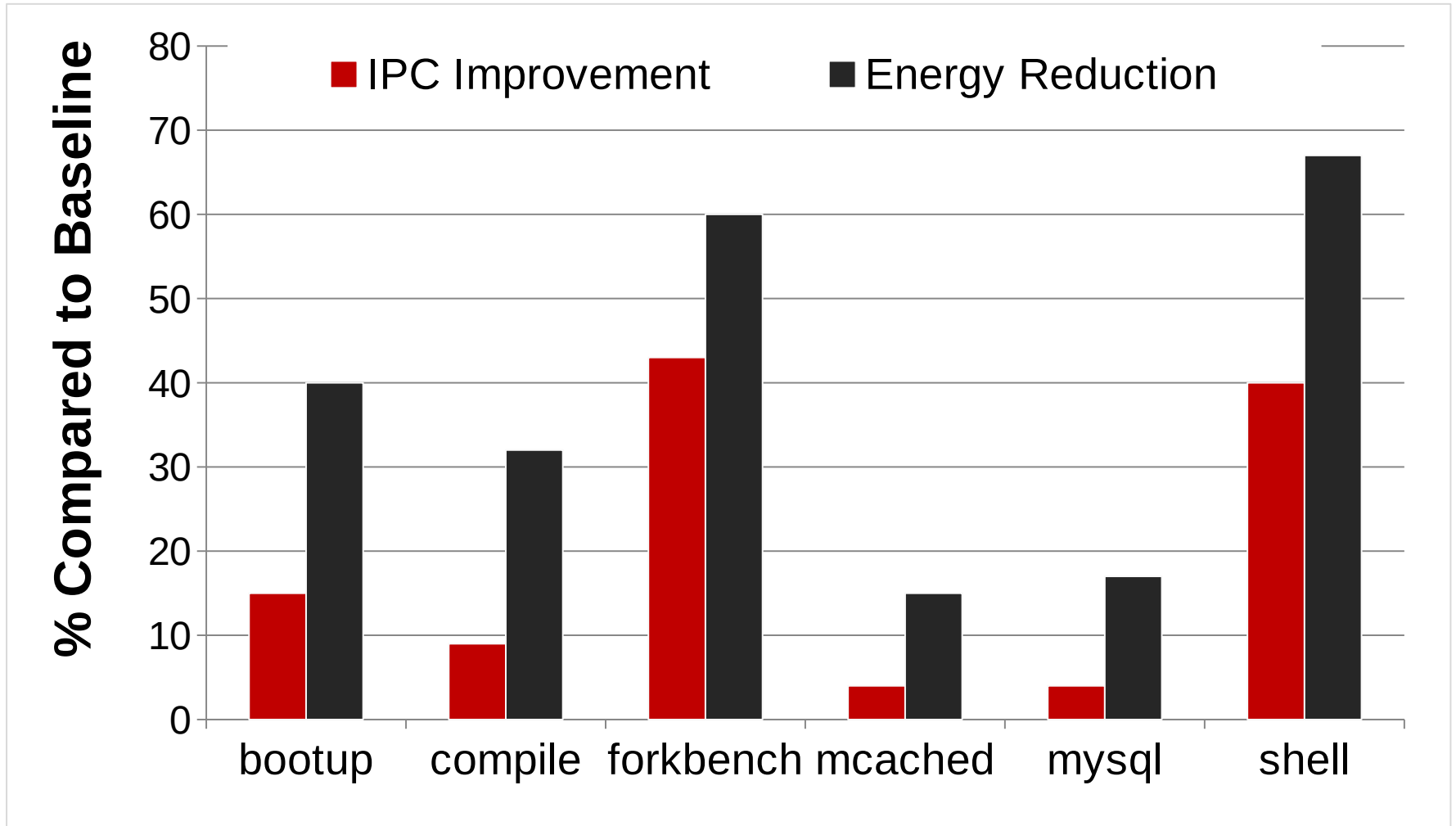


Very low cost: 0.01% increase in die area

Copy and Initialization in Workloads



RowClone: Application Performance



End-to-End System Design

Application

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

Operating System

How to ensure cache coherence?

ISA

Microarchitecture

How to maximize latency and energy savings?

DRAM (RowClone)

How to handle data reuse?

Ambit

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

Vivek Seshadri

Donghyuk Lee, Thomas Mullins, Hasan Hassan,
Amirali Boroumand, Jeremie Kim, Michael A.
Kozuch, Onur Mutlu, Phillip B. Gibbons, Todd C.
Mowry

SAFARI

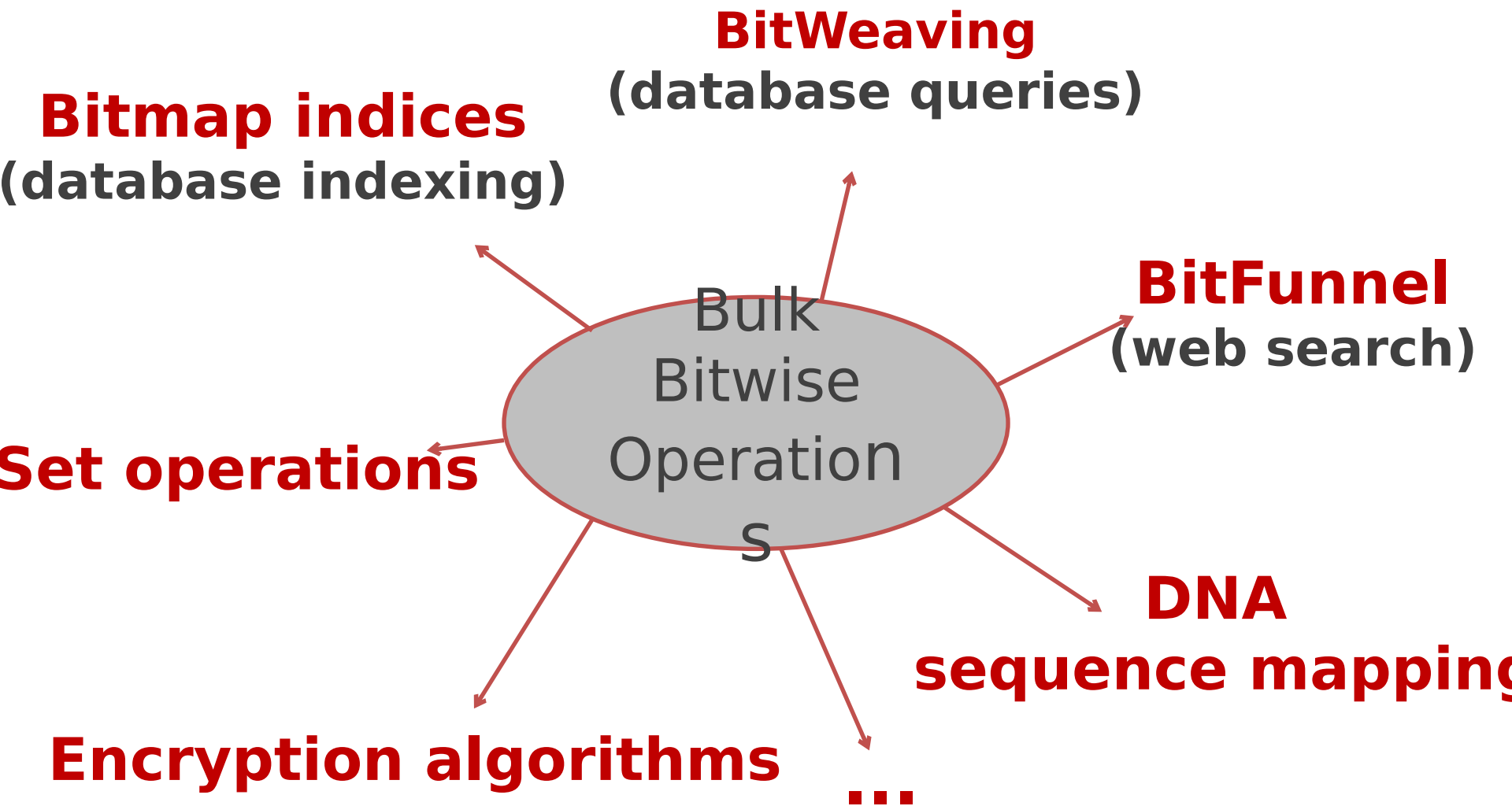
Carnegie Mellon



ETH zürich

Executive Summary

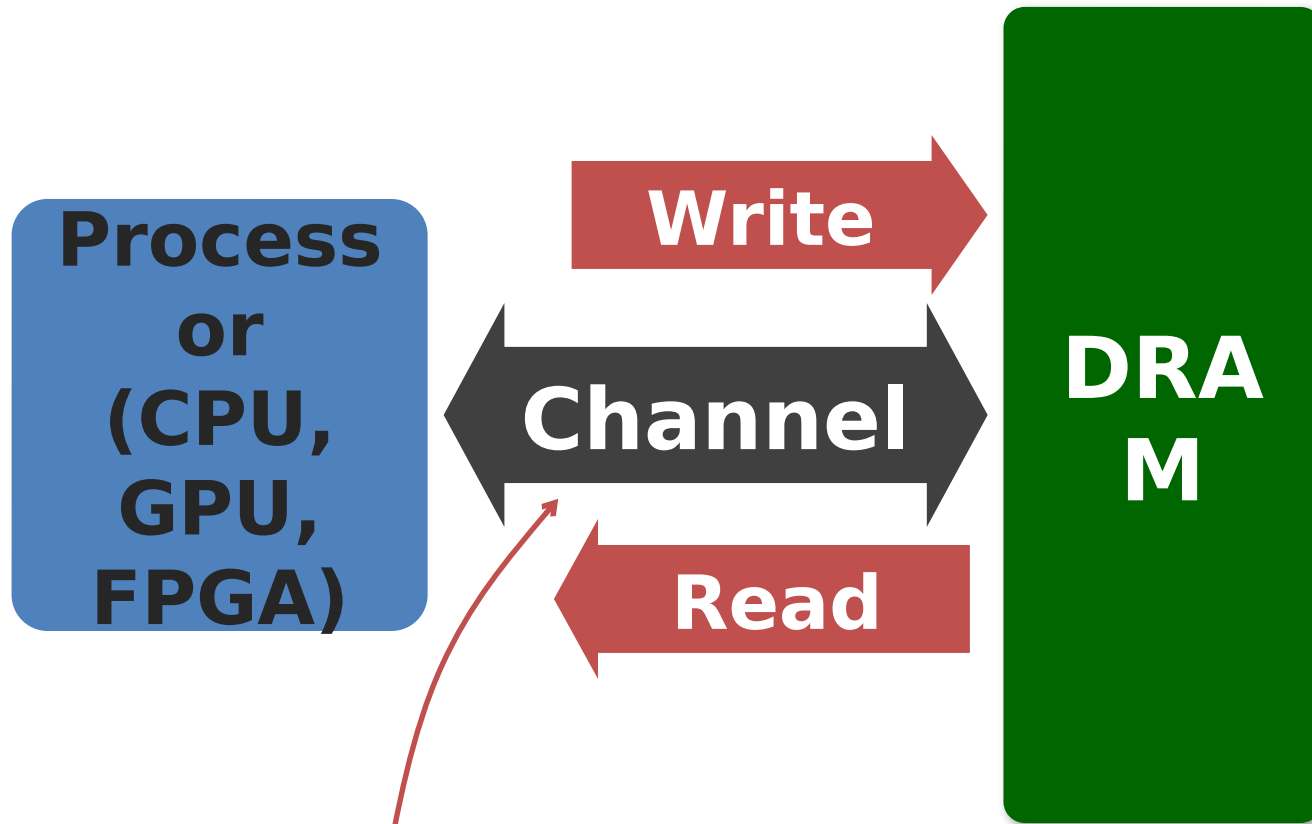
- **Problem: Bulk bitwise operations**
 - present in many applications, e.g., databases, search filters
 - existing systems are memory bandwidth limited
- **Our Proposal: Ambit**
 - perform bulk bitwise operations **completely inside DRAM**
 - **bulk bitwise AND/OR**: simultaneous activation of three rows
 - **bulk bitwise NOT**: inverters already in sense amplifiers
 - less than 1% area overhead over existing DRAM chips
- **Results compared to state-of-the-art baseline**
 - average across seven bulk bitwise operations
 - 32X performance improvement, 35X energy reduction
 - 3X-7X performance for real-world data-intensive applications



[1] Li and Patel, BitWeaving, SIGMOD 2013

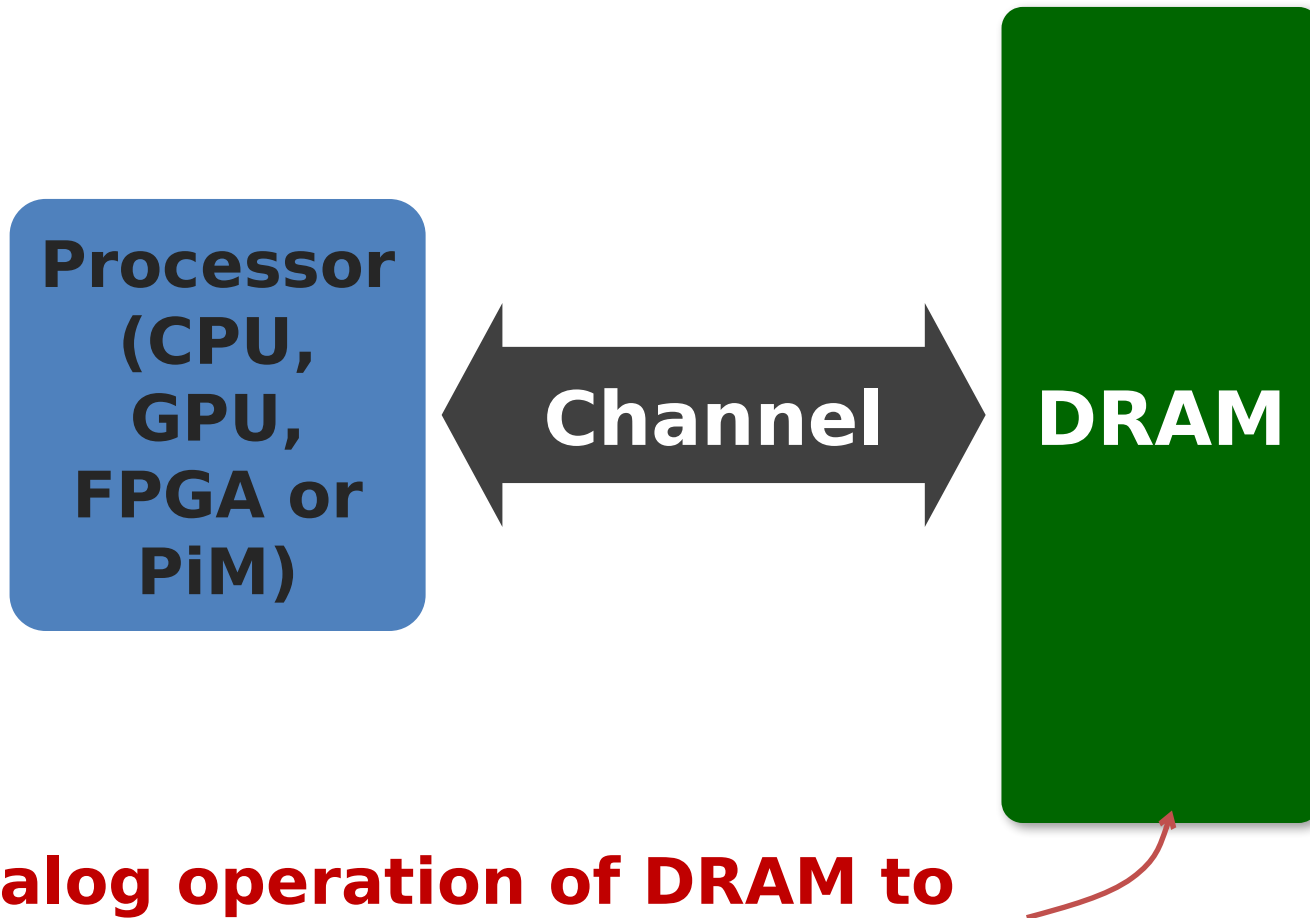
[2] Goodwin+, BitFunnel, SIGIR 2017

Today, DRAM is just a storage device!



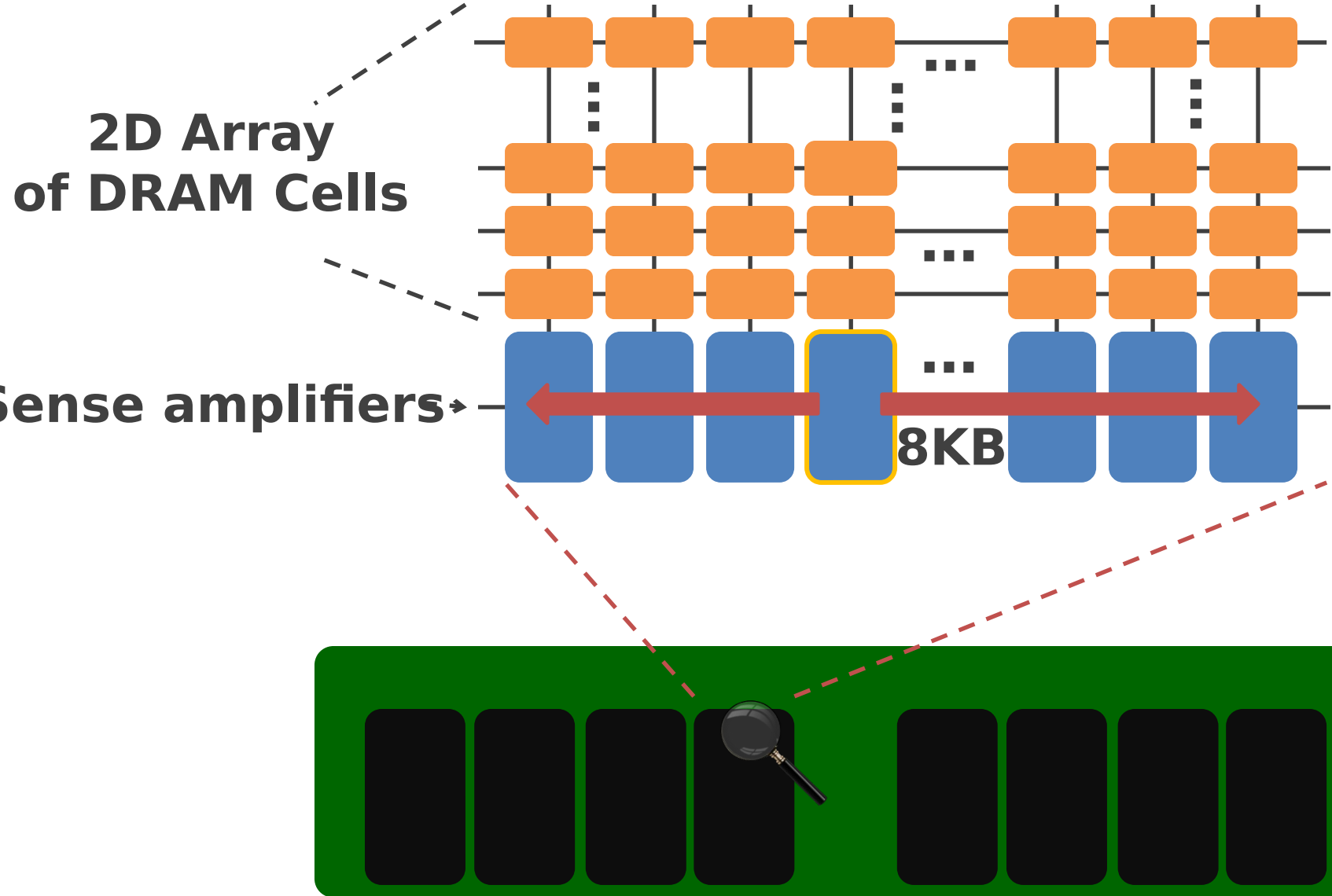
Throughput of bulk bitwise operations limited by available memory bandwidth

Our Approach

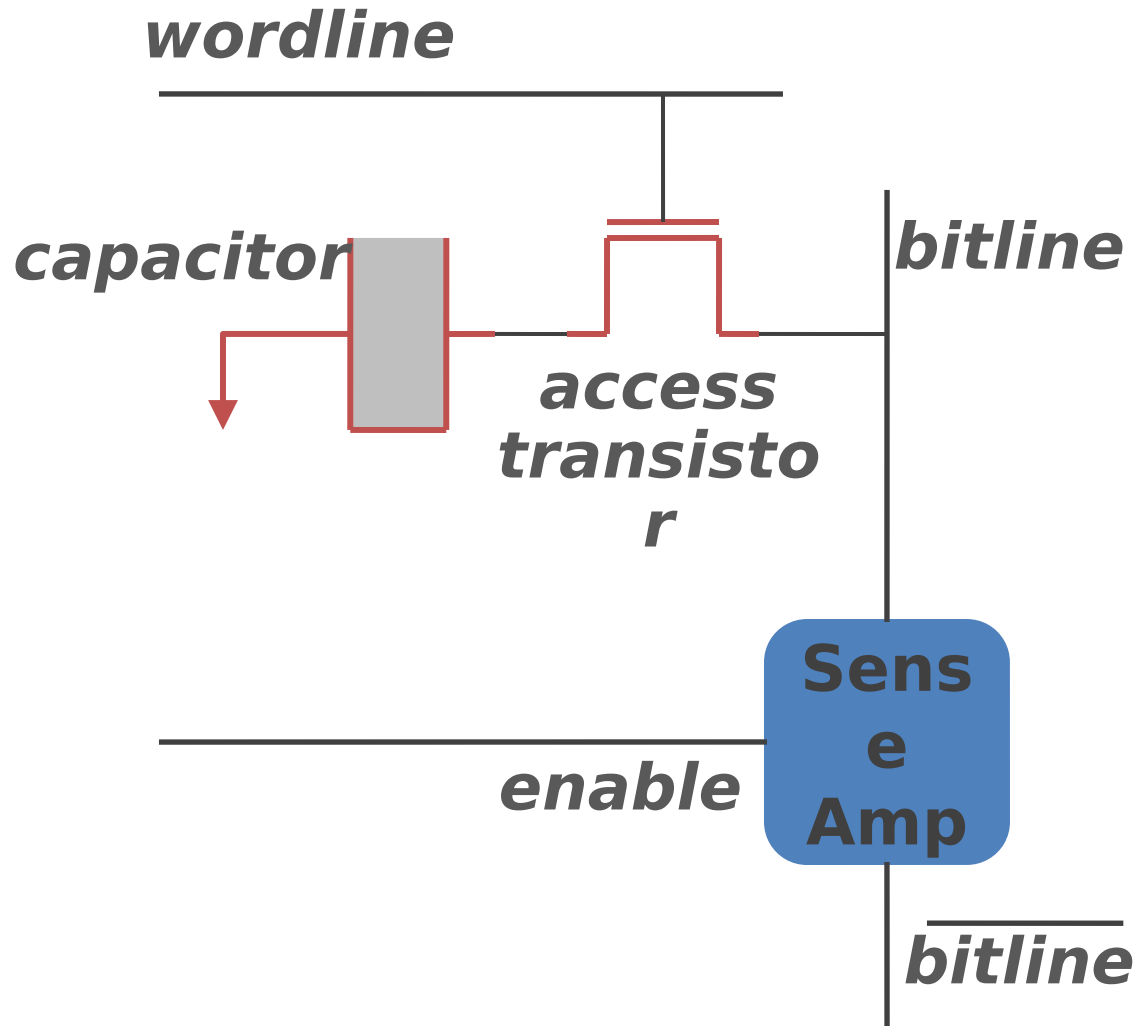


Use analog operation of DRAM to perform bitwise operations completely inside memory!

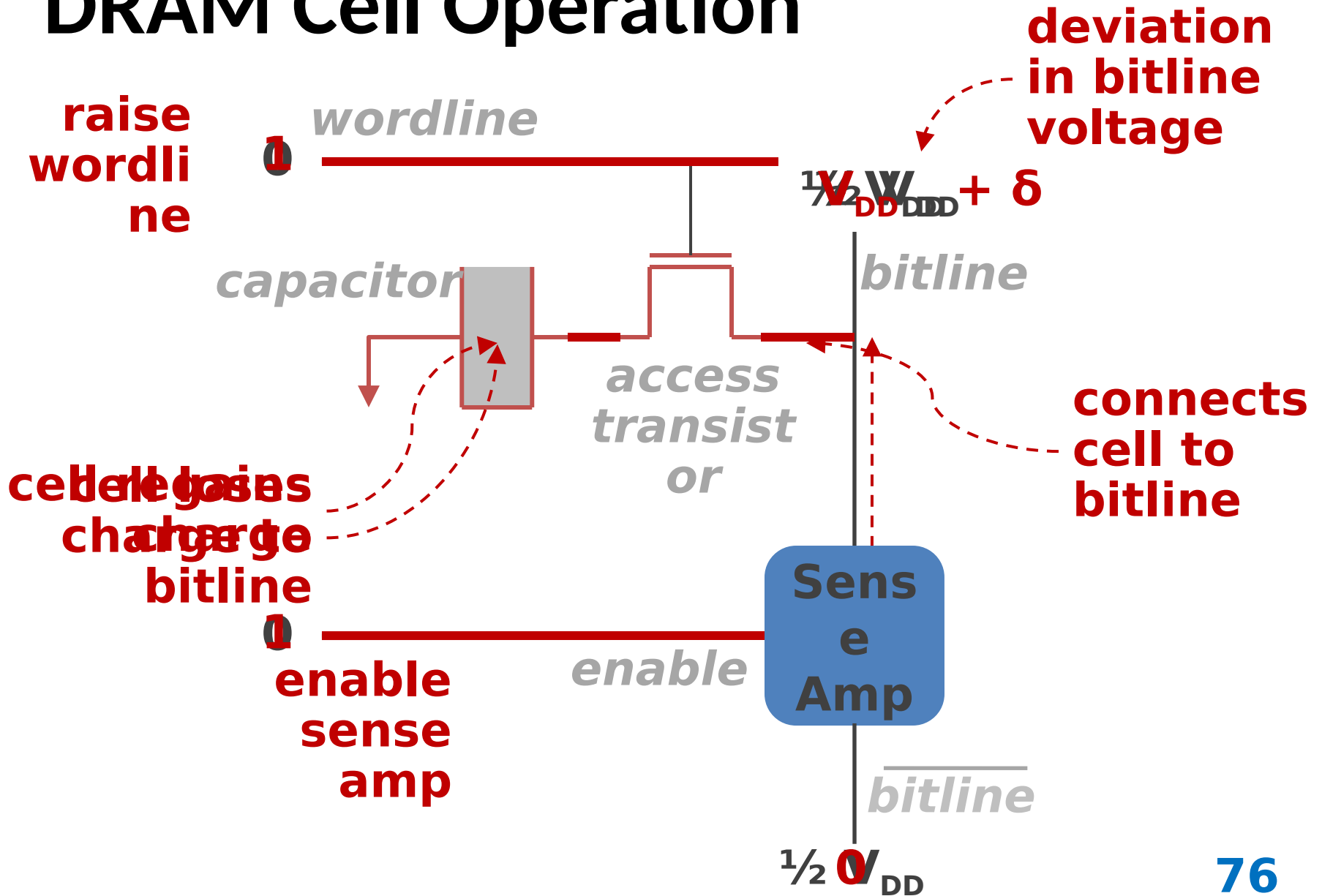
Inside a DRAM Chip



DRAM Cell Operation

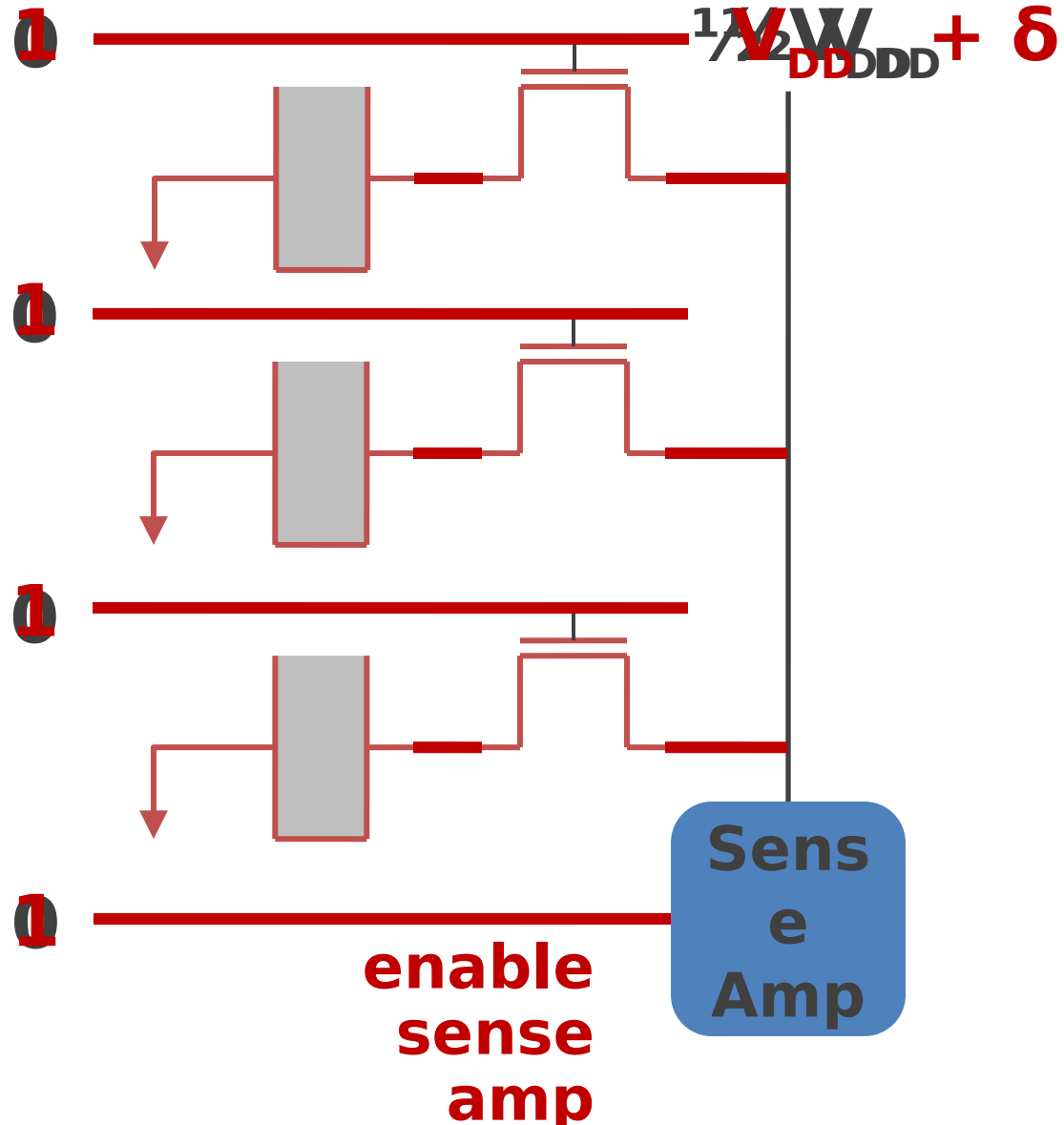


DRAM Cell Operation

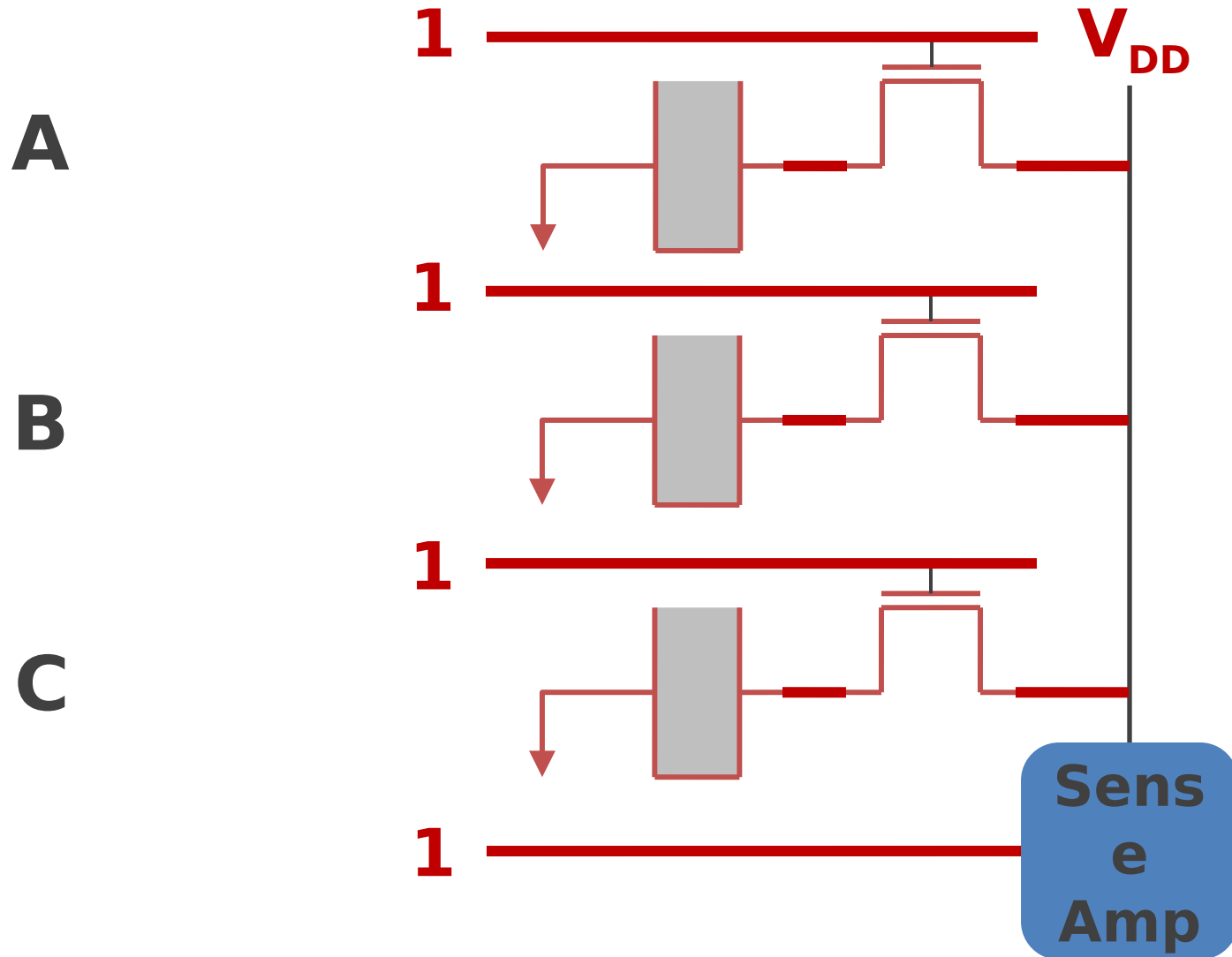


Triple-Row Activation: Majority Function

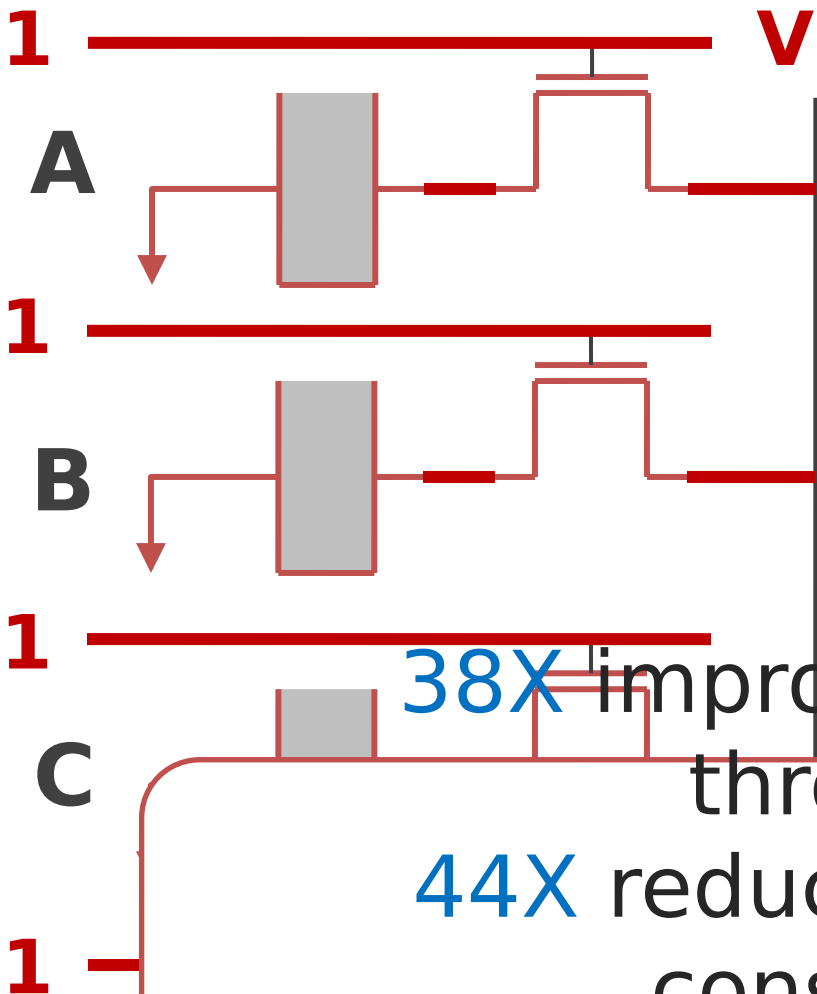
activate
all
three
rows



Bitwise AND/OR Using Triple-Row Activation



Bitwise AND/OR Using Triple-Row Activation



$$\text{Output} = AB + BC + CA$$

$$= C(A \text{ OR } B) + \sim C(A \text{ AND } B)$$

Control the value of C to perform bitwise

38X improvement in raw throughput
 44X reduction in energy consumption
 for bulk bitwise AND/OR operations

Bulk Bitwise AND/OR in DRAM

typically reserve three designated rows **t1**, **t2**, and **t3**

It = row A AND/OR row B

1. Copy data of row A to row t1

2. Copy data of row B to row t2

3.

MICRO 2013

4.

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

5.

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

Bulk Bitwise AND/OR in DRAM

typically reserve three designated rows **t1**, **t2**, and **t3**

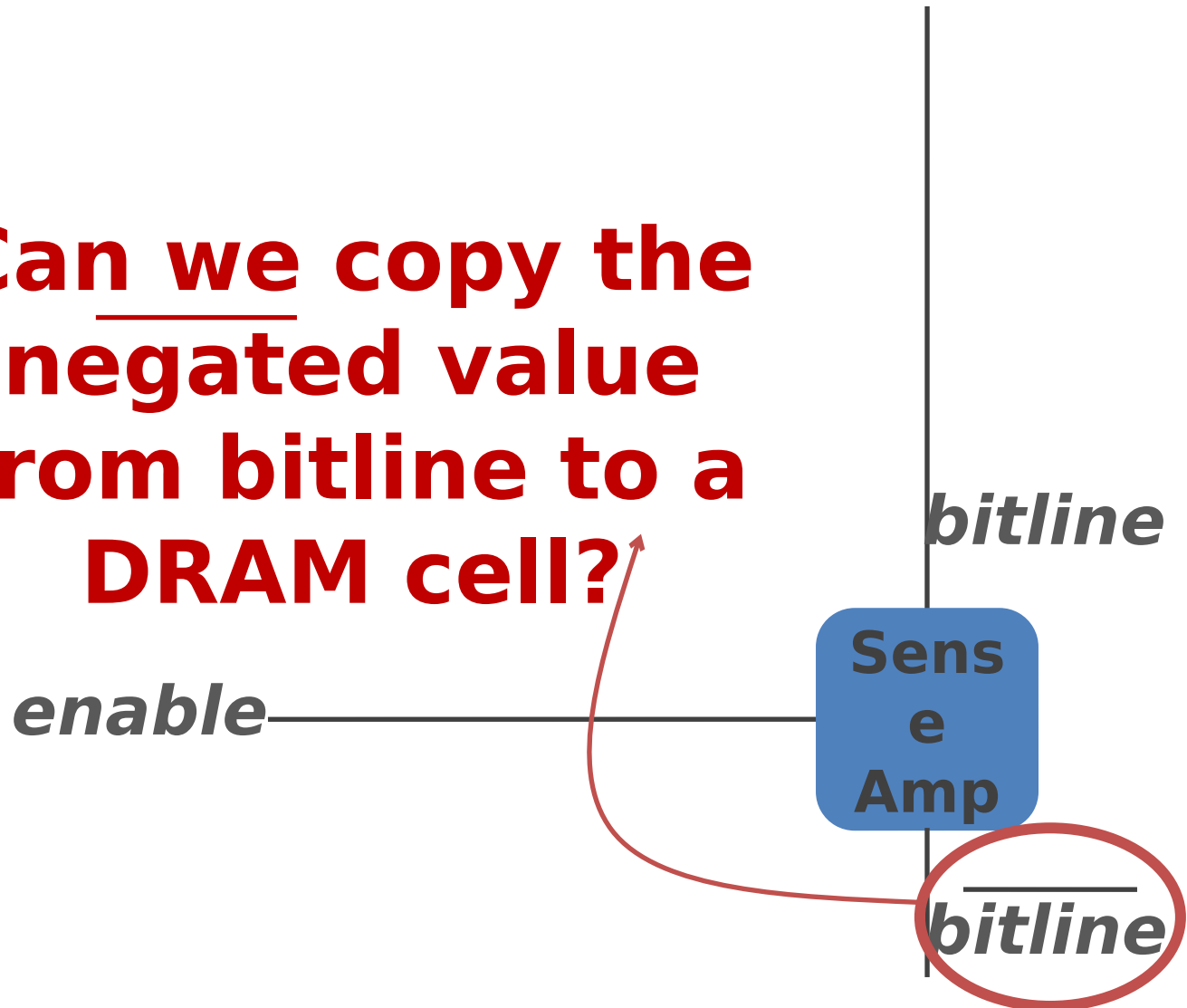
It = row A **AND/OR** row B

1. **Copy** data of row A to row **t1**
2. **Copy** data of row B to row **t2**
3. **Initialize** data of row **t3** to **0/1**
4. **Activate** rows **t1/t2/t3** simultaneously
5. **Copy** data of row **t1/t2/t3** to **Result** row

Use **RowClone** to perform **copy** and **initialization** operations completely in **DRAM!**

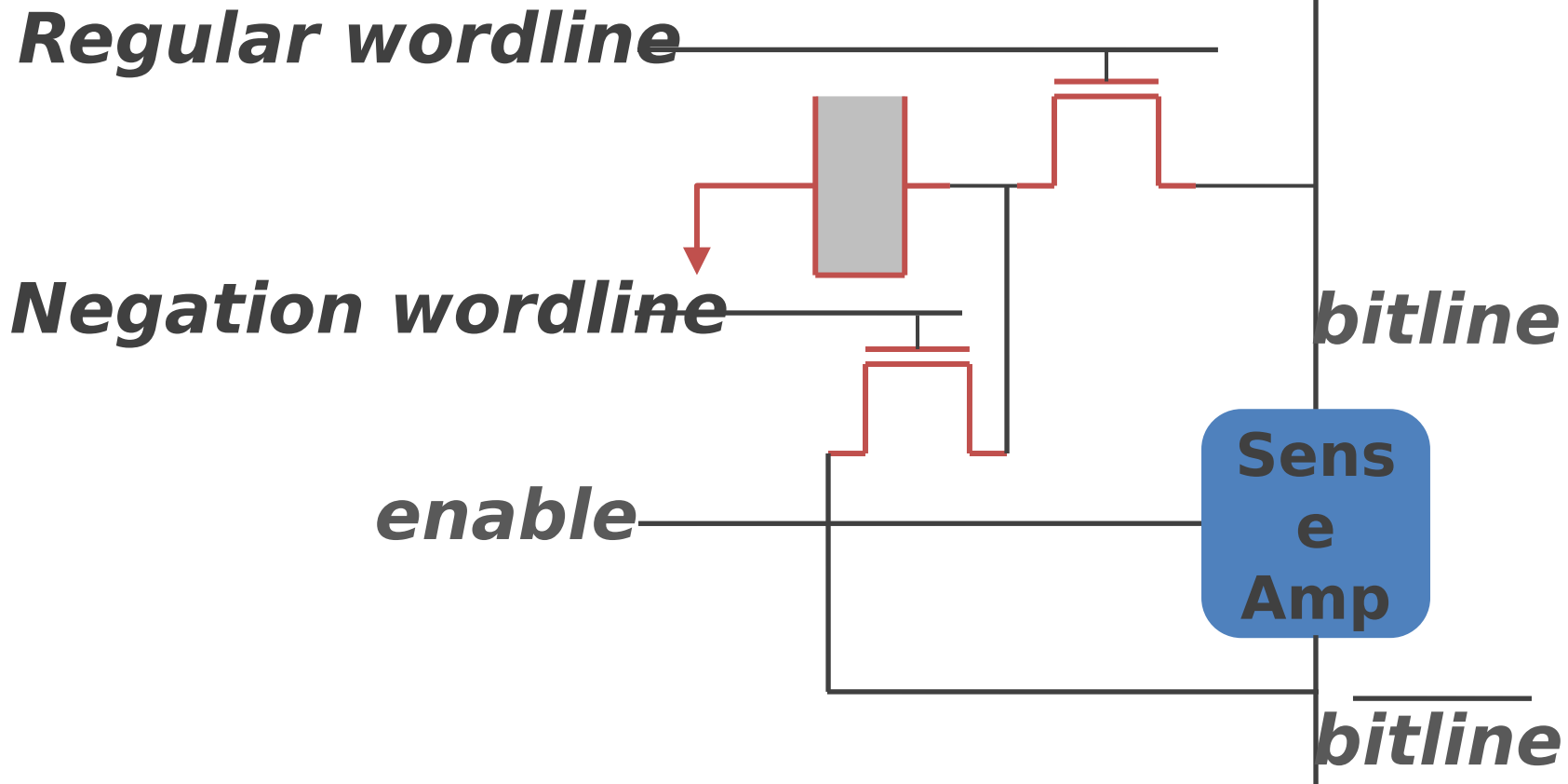
Negation Using the Sense Amplifier

Can we copy the negated value from bitline to a DRAM cell?

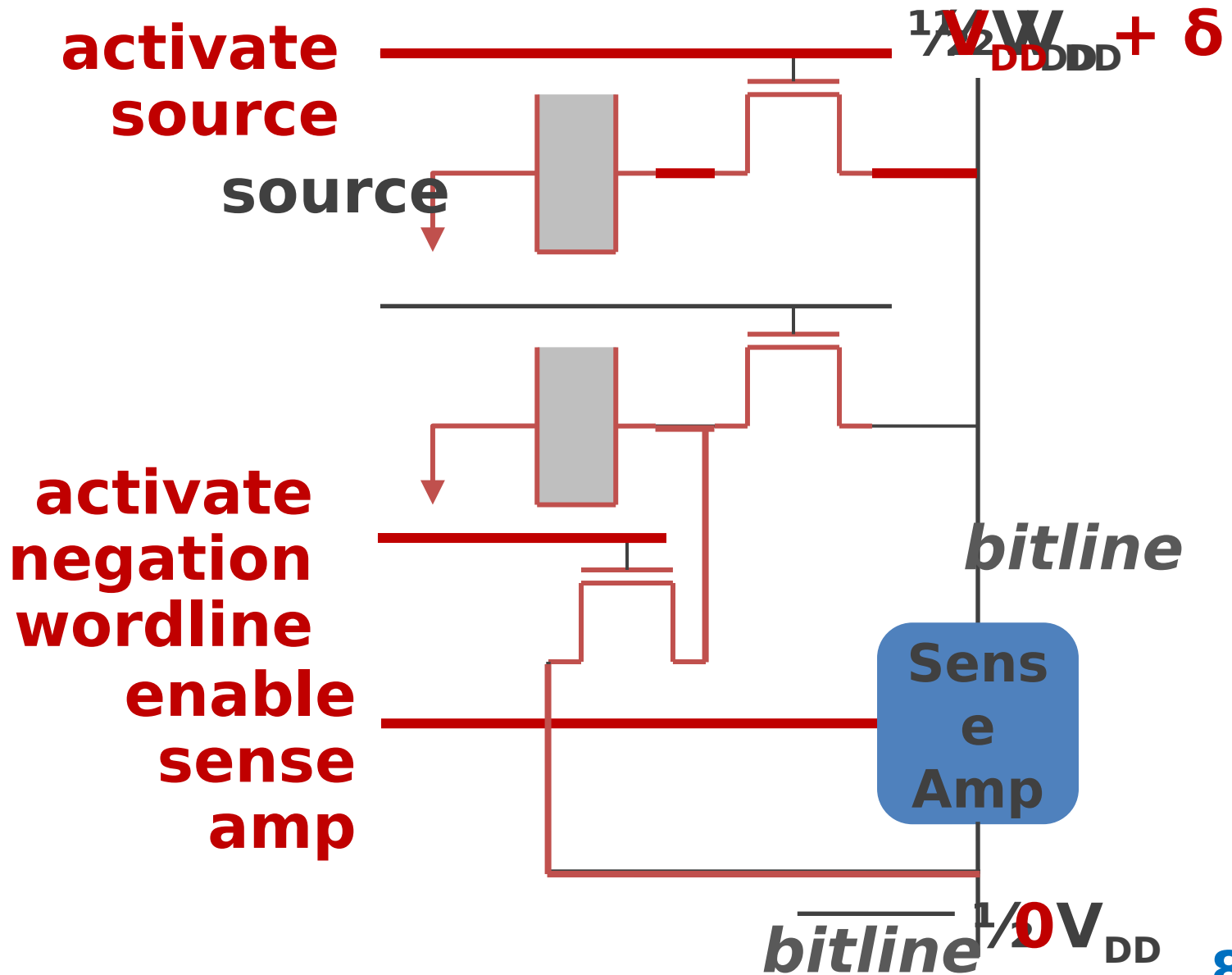


Negation Using the Sense Amplifier

Dual Contact Cell

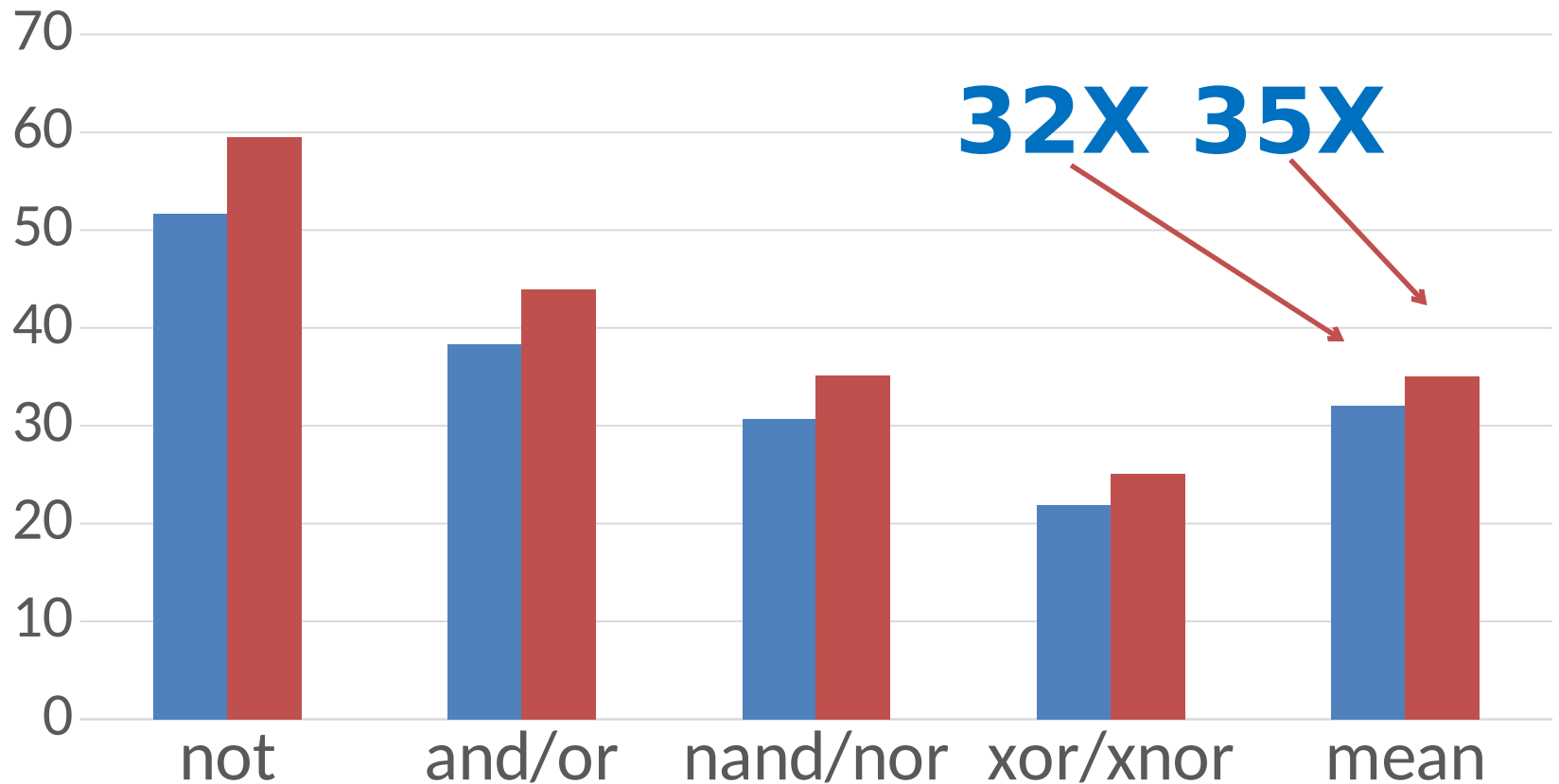


Negation Using the Sense Amplifier



Ambit vs. DDR3: Performance and Energy

- Performance Improvement
- Energy Reduction



Integrating Ambit with the System

1. PCIe device

- Similar to other accelerators (e.g., GPU)

2. System memory bus

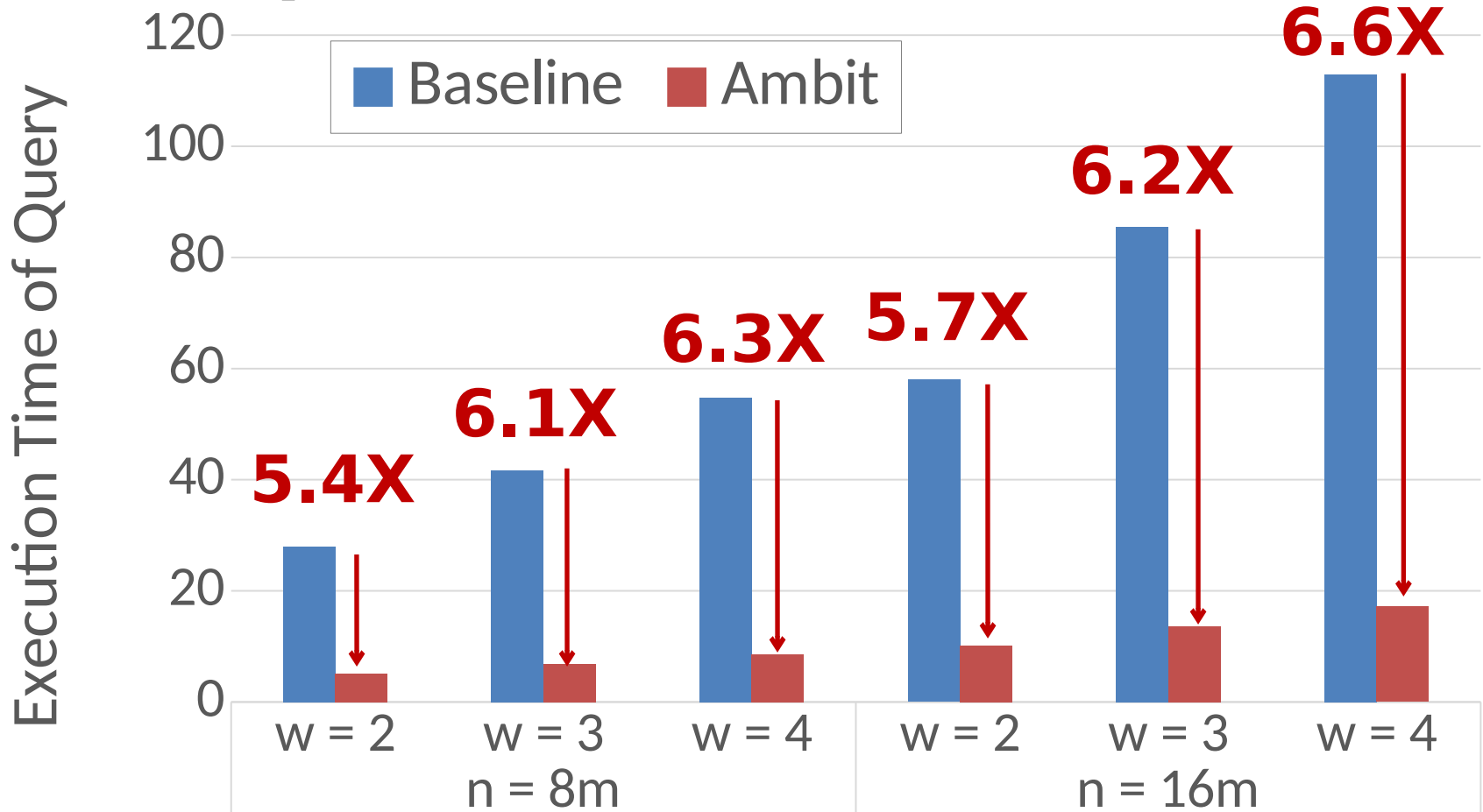
- Ambit uses the same DRAM command/address interface

Pros and cons discussed in paper
(Section 5.4)

Real-world Applications

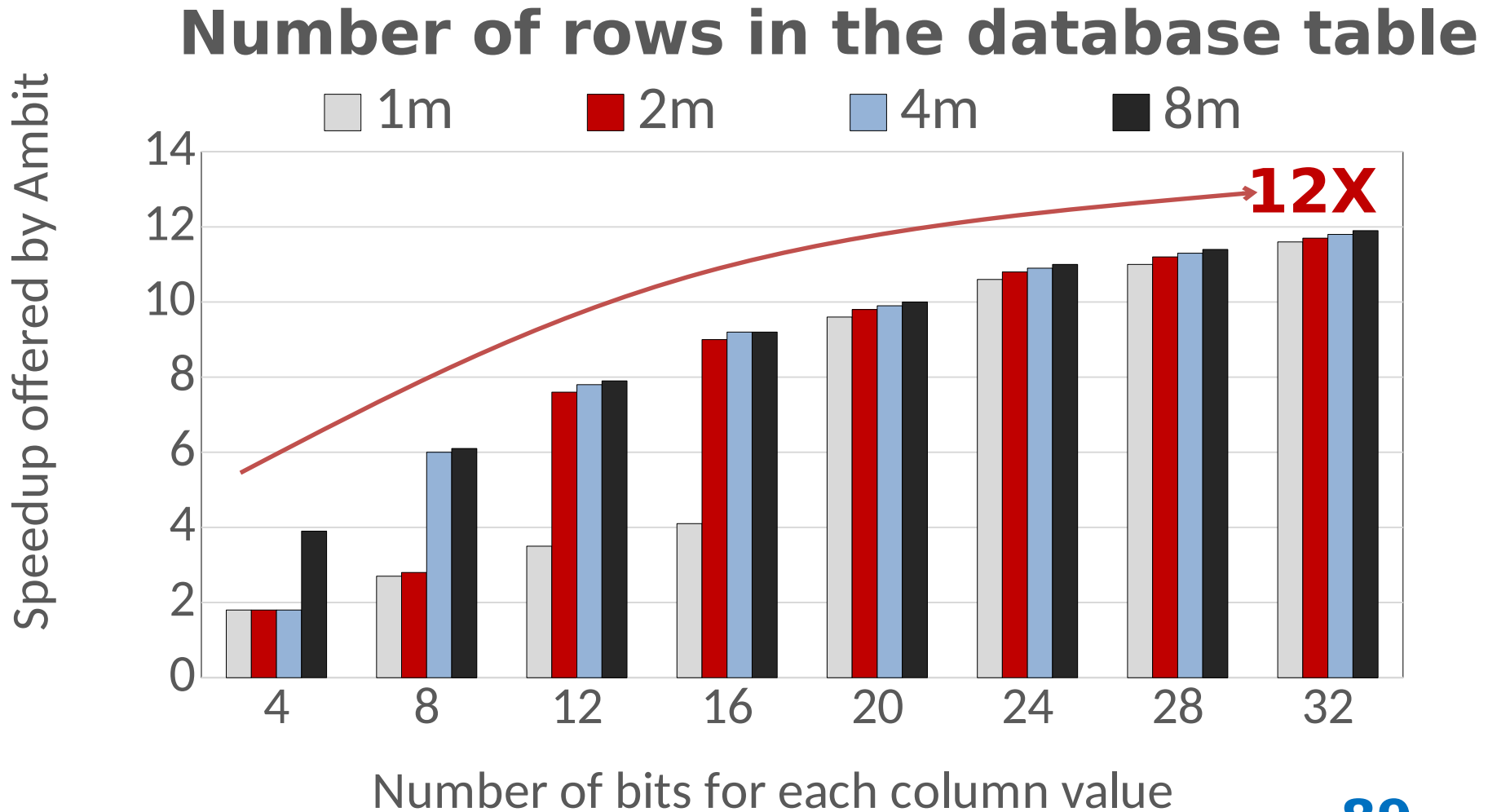
- **Methodology** (Gem5 simulator)
 - Processor: x86, 4 GHz, out-of-order, 64-entry instruction queue
 - L1 cache: 32 KB D-cache and 32 KB I-cache, LRU policy
 - L2 cache: 2 MB, LRU policy
 - Memory controller: FR-FCFS, 8 KB row size
 - Main memory: DDR4-2400, 1 channel, 1 rank, 8 bank
- **Workloads**
 - **Database bitmap indices**
 - **BitWeaving** - column scans using bulk bitwise operations
 - **Set operations** - comparing bitvectors with red-black trees

Bitmap Indices: Performance



consistent reduction in execution time. 6X on average

Speedup offered by Ambit for BitWeaving `select count(*) where c1 < field < c2`



Review #5

Flipping Bits in Memory Without Accessing Them

Yoongu Kim et al., *ISCA 2014*

CSC 2224: Parallel Computer Architecture and Programming Advanced Memory

Prof. Gennady Pekhimenko

University of Toronto

Fall 2022

*The content of this lecture is adapted from the slides of
Vivek Seshadri, Yoongu Kim,
and lectures of Onur Mutlu @ ETH and CMU*