Memory Systems and Memory-Centric Computing Systems
Lecture 1: Memory Trends and Basics

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

9-13 July 2018

HiPEAC ACACES Summer School 2018

SAFARI

ETH zürich

Carnegie Mellon

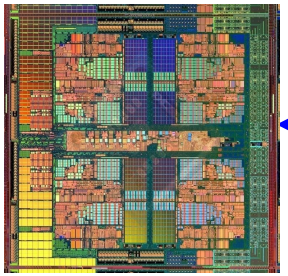
Brief Self Introduction

- Onur Mutlu
  - Full Professor @ ETH Zurich CS (EE), since September 2015
  - Strecker Professor @ Carnegie Mellon University ECE (CS), 2009-2016, 2016-…
  - PhD from UT-Austin, worked at Google, VMware, Microsoft Research, Intel, AMD
  - https://people.inf.ethz.ch/omutlu/
  - omutlu@gmail.com (Best way to reach me)
  - https://people.inf.ethz.ch/omutlu/projects.htm
- Research and Teaching in:
  - Computer architecture, hardware security, bioinformatics, computing platforms
  - Memory and storage systems
  - Hardware security, safety, predictability
  - Fault tolerance
  - Hardware/software cooperation
  - Architectures for bioinformatics, health, medicine
  - …

Current Research Focus Areas

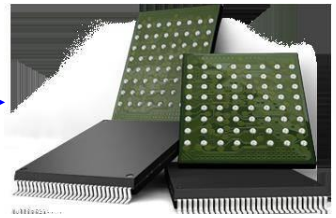
***Research Focus:* Computer architecture, HW/SW, bioinformatics, security**

- ***Memory and storage (DRAM, flash, emerging), interconnects***
- ***Heterogeneous & parallel systems, GPUs, systems for data analytics***
- ***System/architecture interaction, new execution models, new interfaces***
- ***Hardware security, energy efficiency, fault tolerance, performance***
- ***Genome sequence analysis & assembly algorithms and architectures***
- ***Biologically inspired systems & system design for bio/medicine***

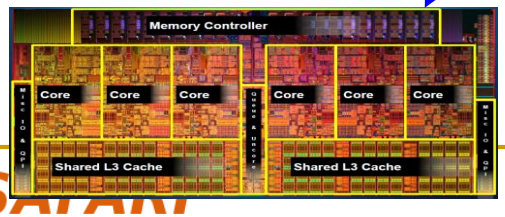


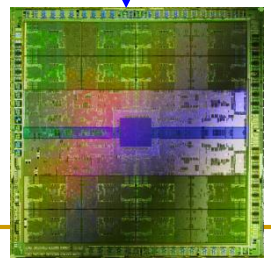
Heterogeneous Processors and Accelerators

Hybrid Main Memory



Persistent Memory/Storage





Graphics and Vision Processing

Broad research spanning apps, systems, logic with architecture at the center

Four Key Directions

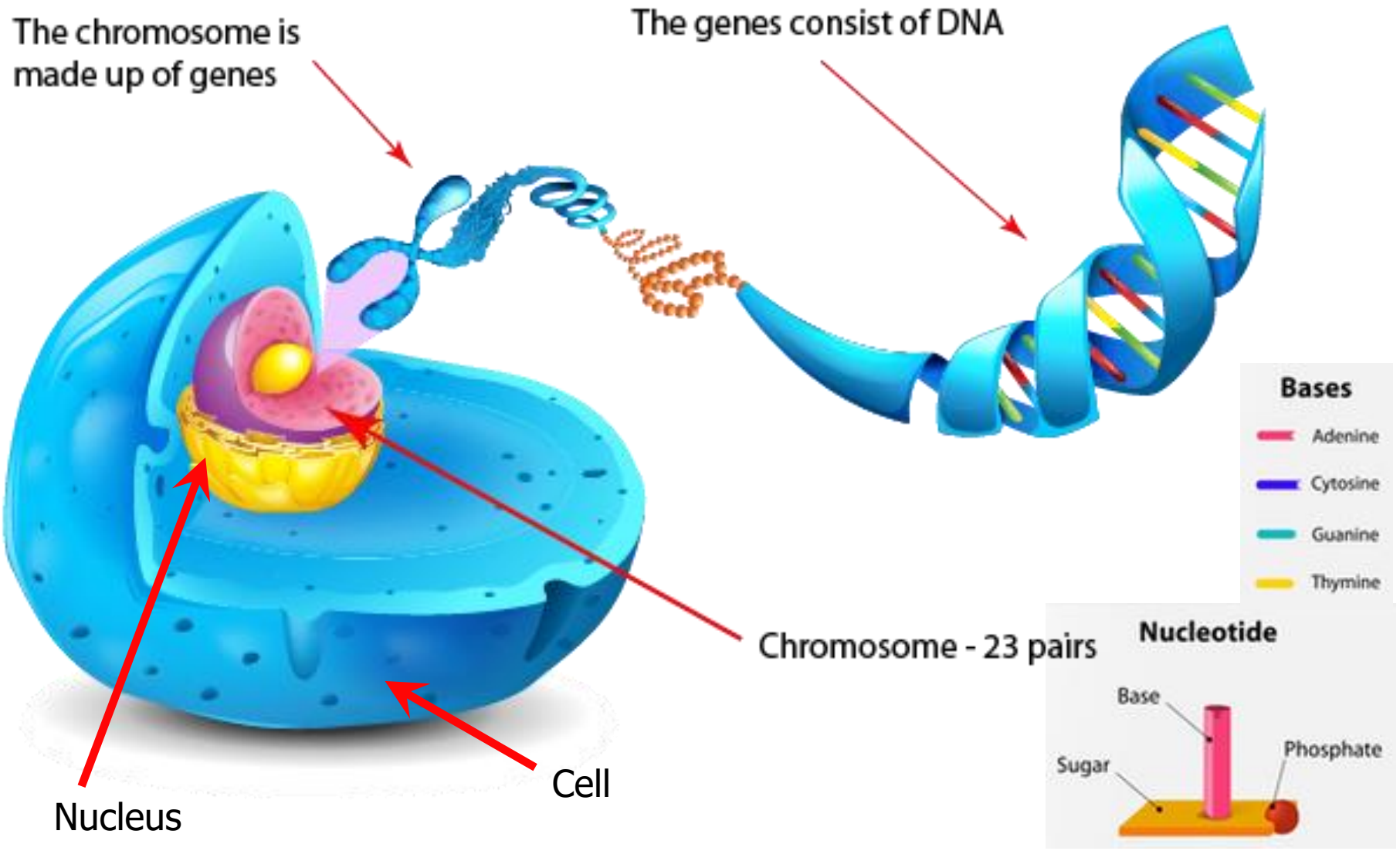
- Fundamentally Secure/Reliable/Safe Architectures
- Fundamentally Energy-Efficient Architectures
  - Memory-centric (Data-centric) Architectures
- Fundamentally Low-Latency Architectures
- Architectures for Genomics, Medicine, Health

A Motivating Detour: Genome Sequence Analysis

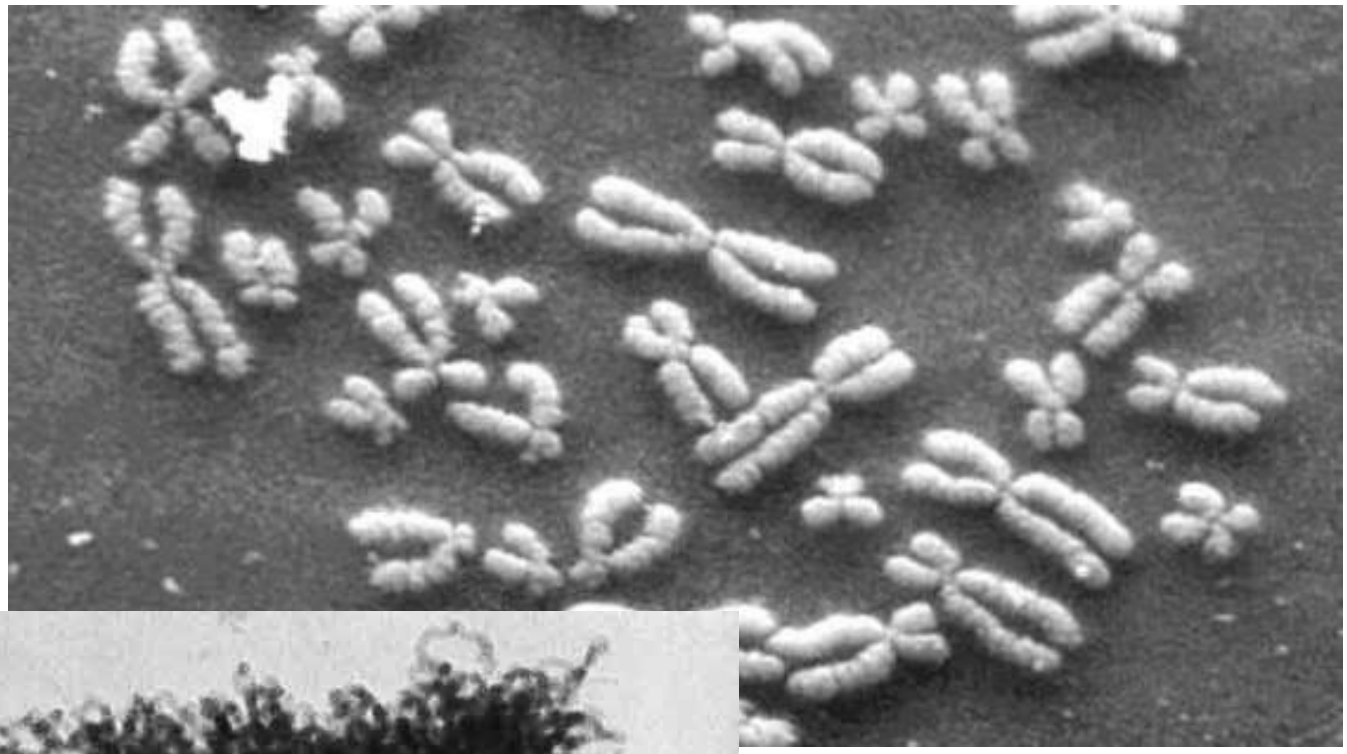
Our Dream

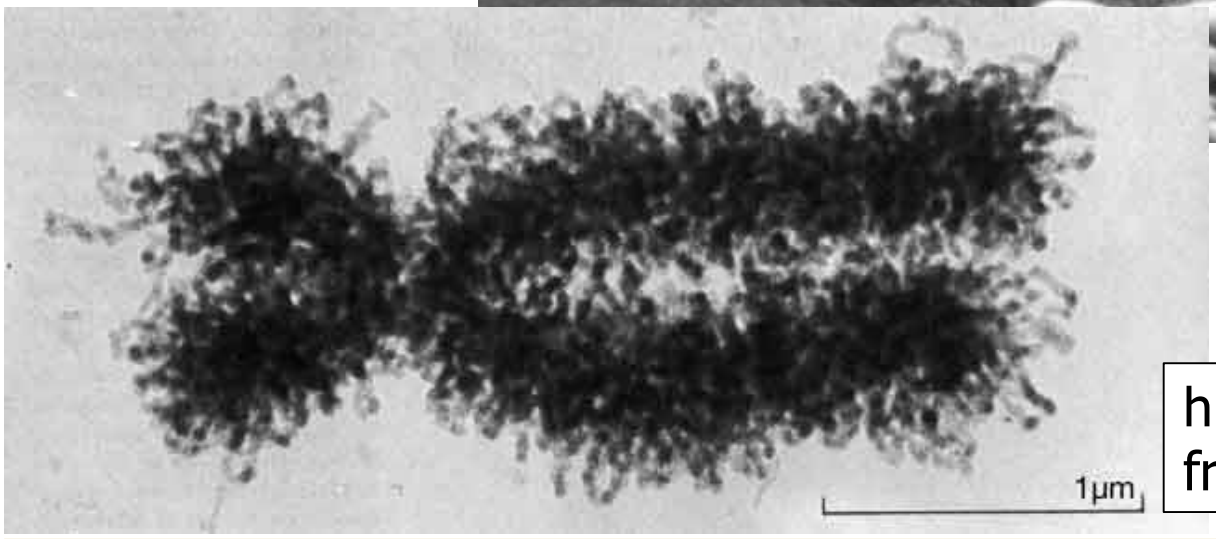
- Can we build devices that can analyze a genome within a minute?

What Is a Genome Made Of?



DNA Under Electron Microscope





human chromosome #12
from HeLa's cell

DNA Sequencing

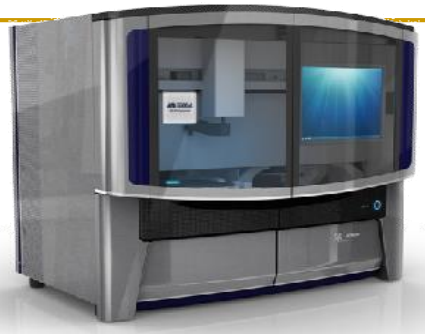
- Goal:
  - Find the complete sequence of A, C, G, T's in DNA.

- Challenge:
  - There is no machine that takes long DNA as an input, and gives the complete sequence as output
  - All sequencing machines chop DNA into pieces and identify relatively small pieces (but not how they fit together)

Untangling Yarn Balls & DNA Sequencing

Genome Sequencers

Roche/454



AB SOLiD

Illumina MiSeq

Complete Genomics

Oxford Nanopore MinION

Illumina HiSeq2000

Pacific Biosciences RS

Illumina NovaSeq 6000

Oxford Nanopore GridION

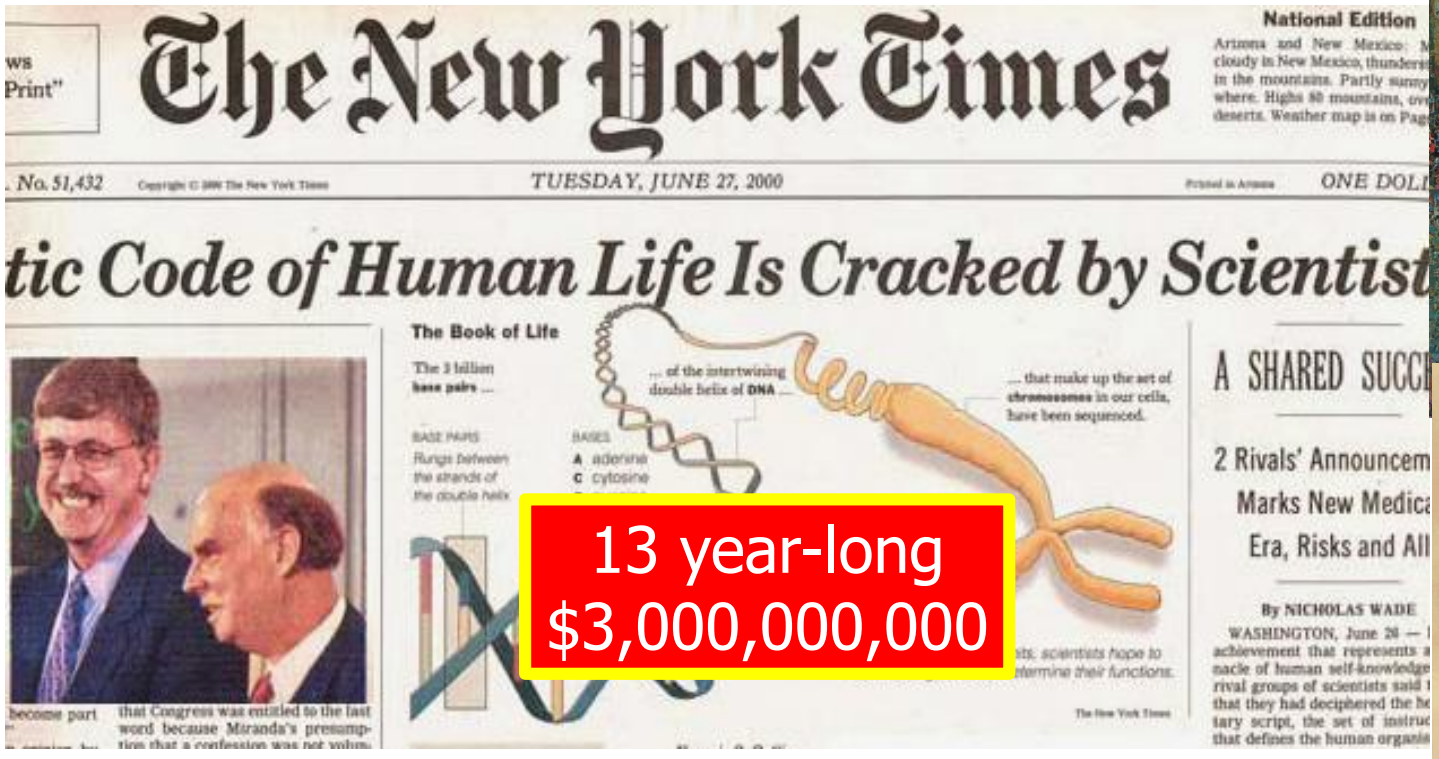
Ion Torrent PGM

Ion Torrent Proton

… and more! All produce data with different properties.

The Genomic Era

- 1990-2003: The Human Genome Project (HGP) provides a complete and accurate sequence of all **DNA base pairs** that make up the human genome and finds 20,000 to 25,000 human genes.


The New York Times

National Edition

TUESDAY, JUNE 27, 2000

tic Code of Human Life Is Cracked by Scientist

The Book of Life

A SHARED SUCC

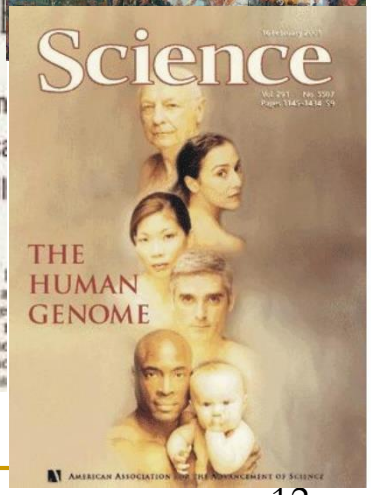
2 Rivals' Announcem

Marks New Medica

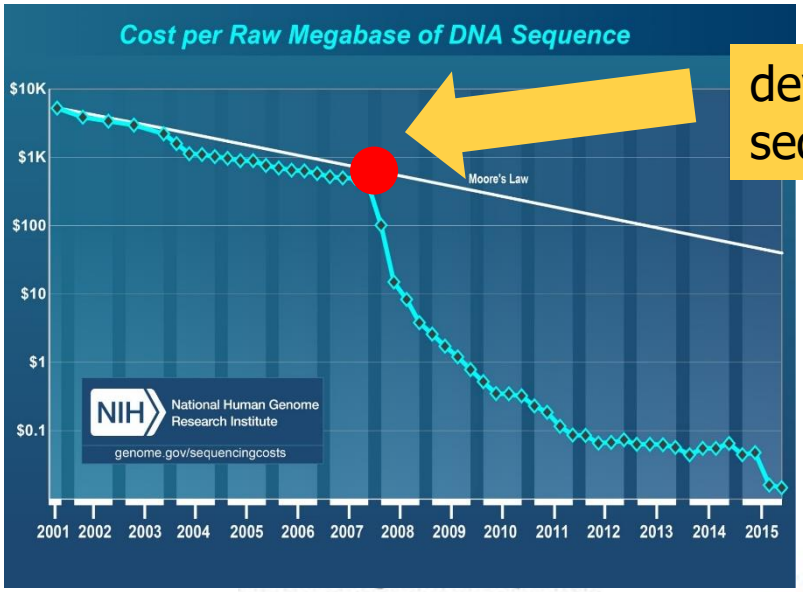
Era, Risks and All

By NICHOLAS WADE

13 year-long
$3,000,000,000

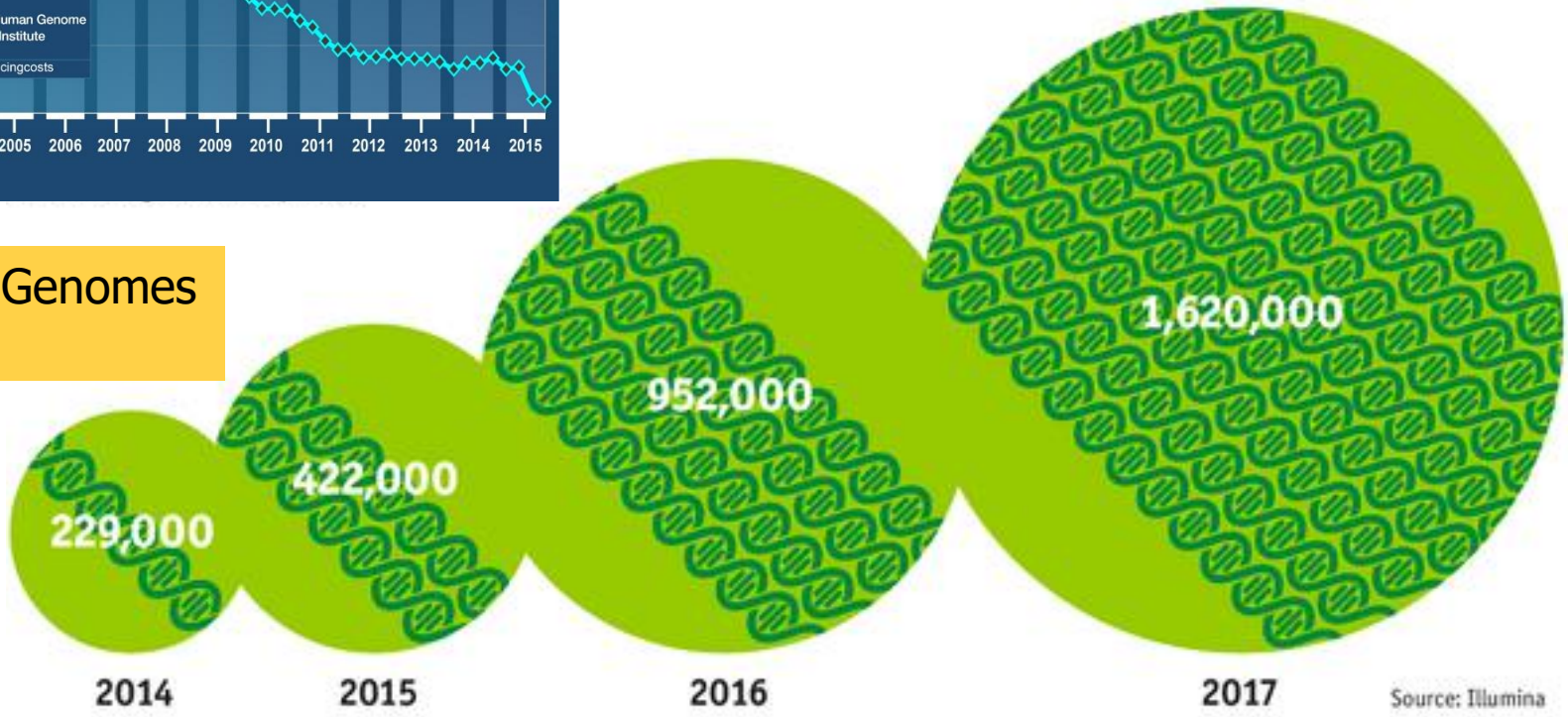


The Genomic Era (continued)



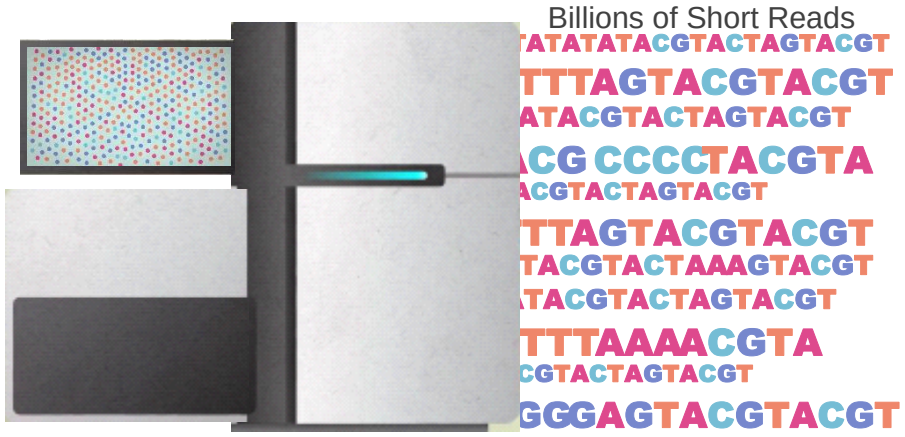
development of high-throughput sequencing (HTS) technologies

Number of Genomes Sequenced

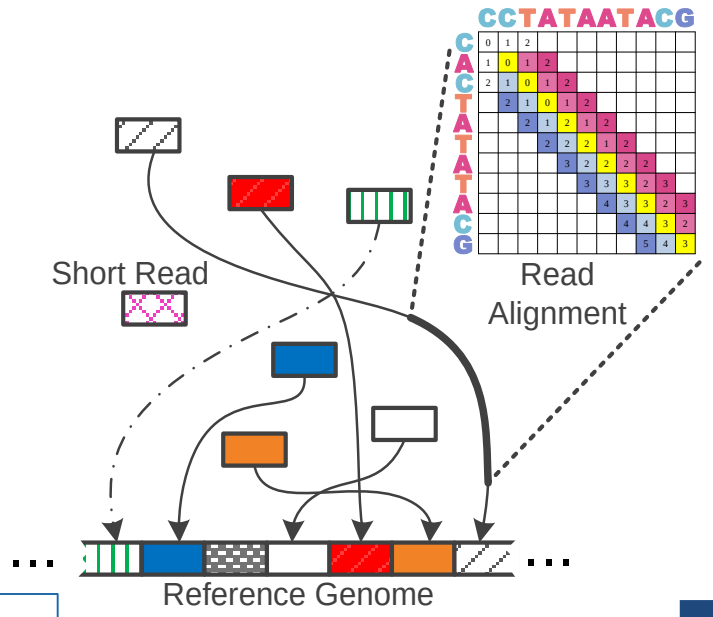


The Economist

http://www.economist.com/news/21631808-so-much-genetic-data-so-many-uses-genes-unzipped



1 Sequencing



Read Mapping 2

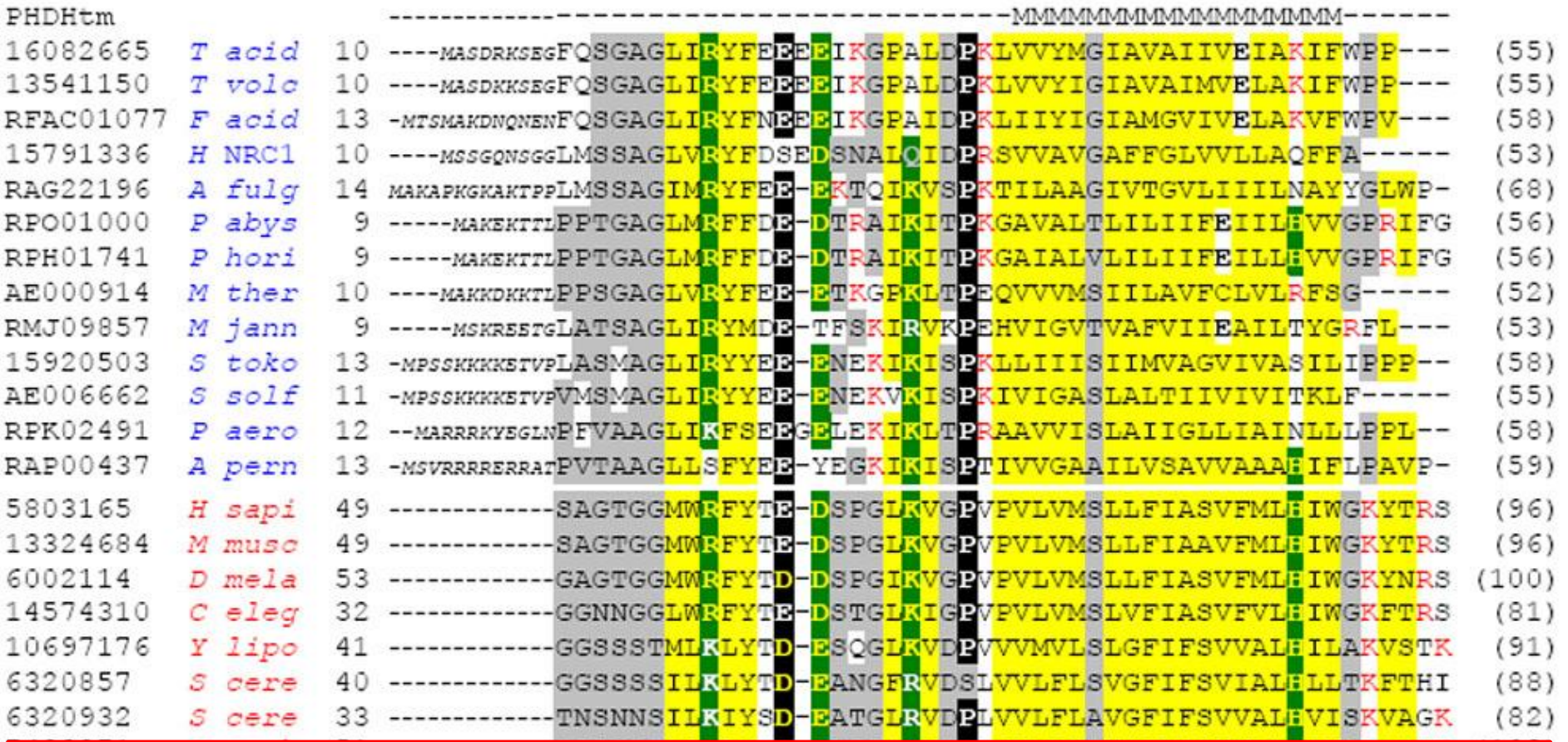
Genome Analysis

```
reference: TTTATCGCTTCCATGACGCAG
read1:       ATCGCATCC
read2:      TATCGCATC
read3:           CATCCATGA
read4:         CGCTTCCAT
read5:               CCATGACGC
read6:             TTCCATGAC
```

3 Variant Calling

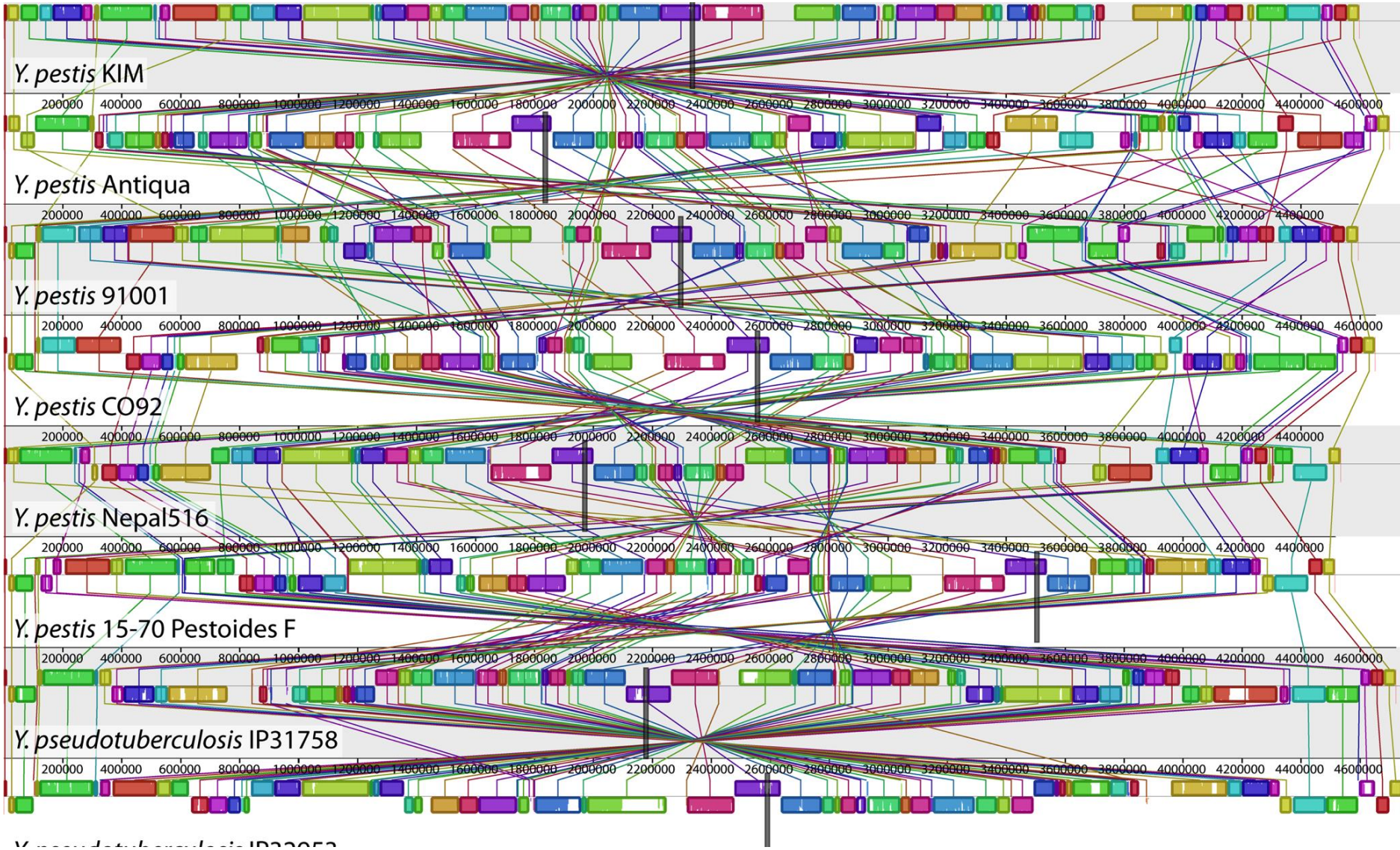
Scientific Discovery 4

Multiple sequence alignment



Example Question: If I give you a bunch of sequences, tell me where they are the same and where they are different.

Genome Sequence Alignment: Example



Source: By Aaron E. Darling, István Miklós, Mark A. Ragan - Figure 1 from Darling AE, Miklós I, Ragan MA (2008). "Dynamics of Genome Rearrangement in Bacterial Populations". PLOS Genetics. DOI:10.1371/journal.pgen.1000128., CC BY 2.5, https://commons.wikimedia.org/w/index.php?curid=30550950

The Genetic Similarity Between Species

Human ~ Human

99.9%

Human ~ Chimpanzee

96%

Human ~ Cat

90%

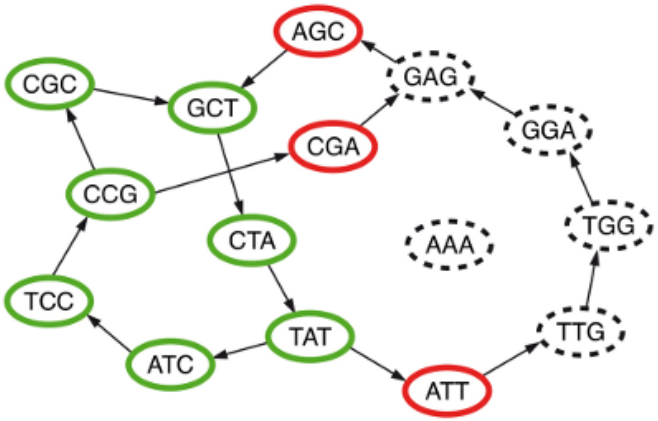
Human ~ Cow

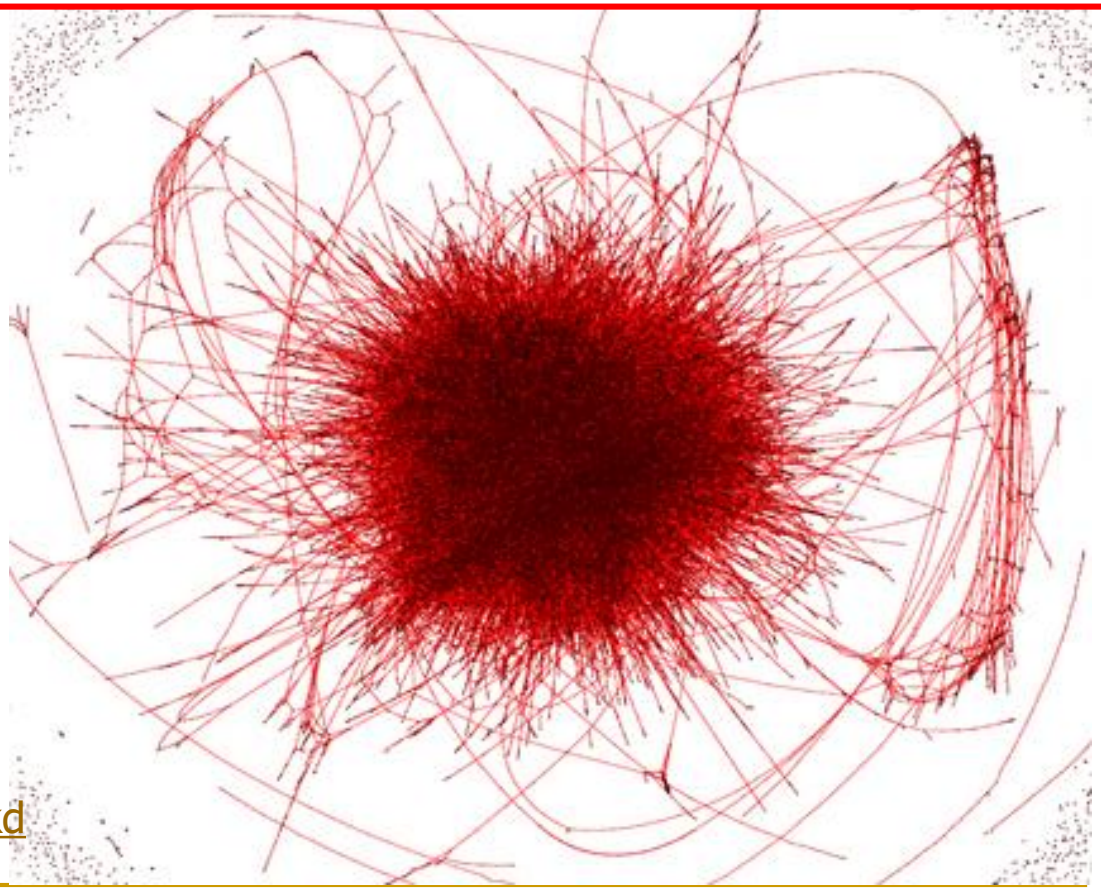
80%

Human ~ Banana

50-60%

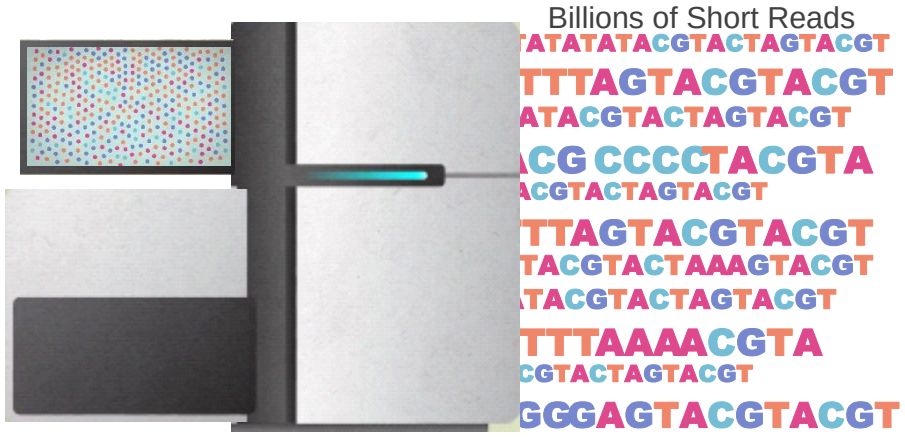
Question 2: Given a bunch of short sequences, Can you identify the approximate species cluster for genomically unknown organisms (bacteria)?



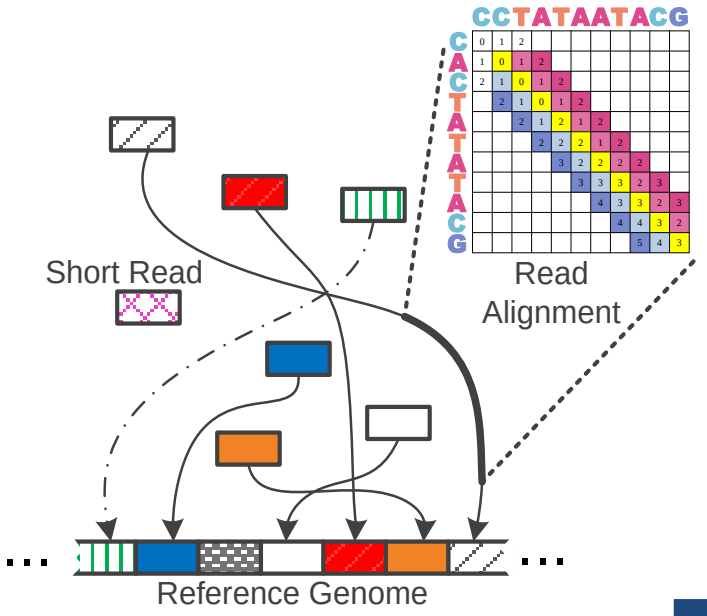


uncleaned de Bruijn graph

http://math.oregonstate.edu/~koslickd



1 Sequencing



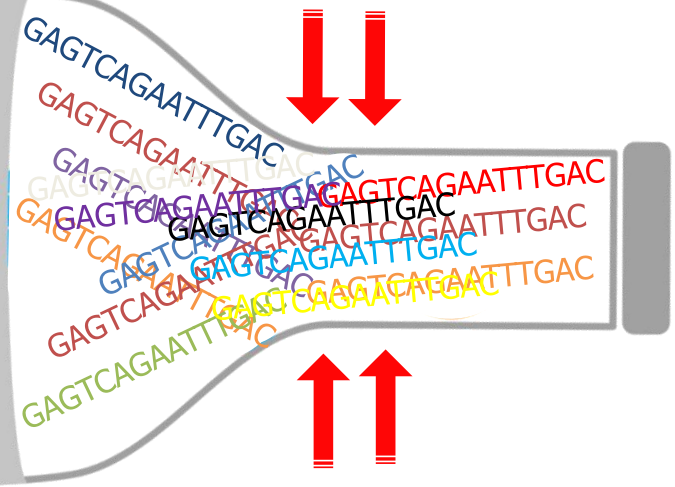
Read Mapping 2

Bottlenecked in Mapping!!

Illumina HiSeq4000

300 M

bases/min



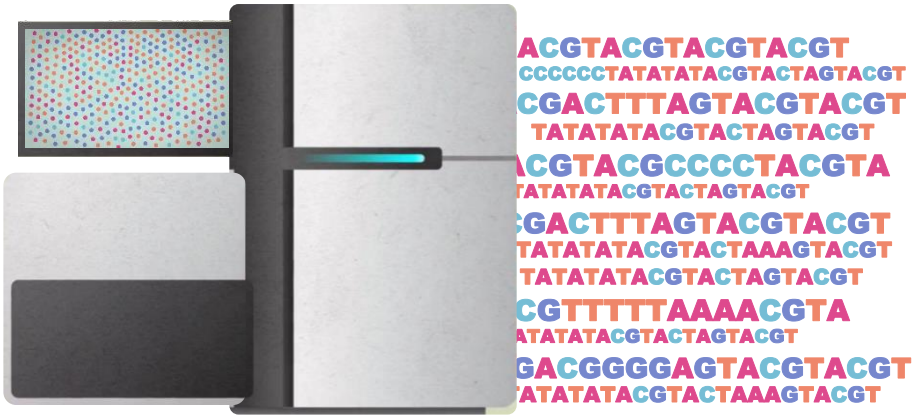
on average

2 M

bases/min

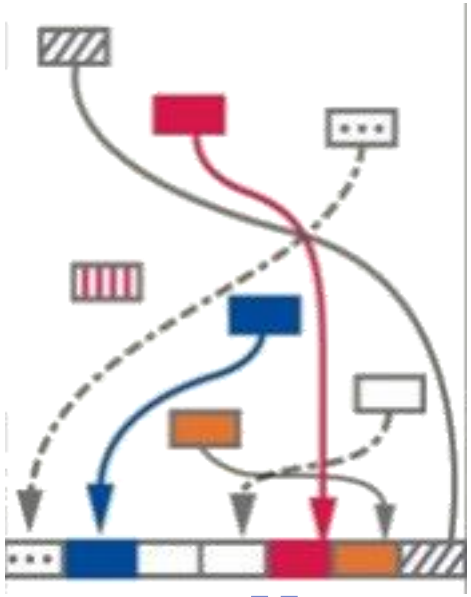
(0.6%)

The Read Mapping Bottleneck



Illumina HiSeq4000

300 Million bases/minute

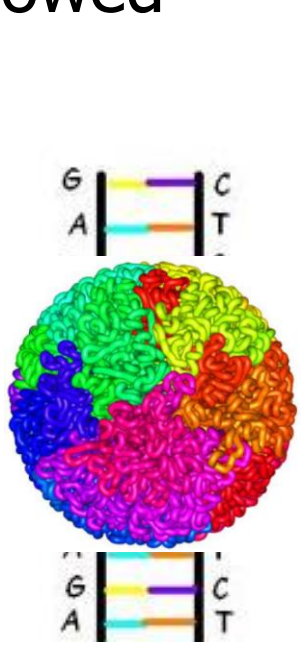


2 Million bases/minute

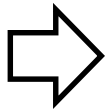
150X slower

Read Mapping

- Map many short DNA fragments (reads) to a known reference genome with some minor differences allowed



DNA, physically



Reads

Reference genome

Mapping short reads to reference genome is challenging (billions of 50-300 base pair reads)

Challenges in Read Mapping

- Need to find many mappings of each read
  - A short read may map to many locations, especially with High-Throughput DNA Sequencing technologies
  - How can we find all mappings efficiently?

- Need to tolerate small variances/errors in each read
  - Each individual is different: Subject's DNA may slightly differ from the reference (Mismatches, insertions, deletions)
  - How can we efficiently map each read with up to *e* errors present?

- Need to map each read very fast (i.e., performance is important)
  - Human DNA is 3.2 billion base pairs long → Millions to billions of reads (State-of-the-art mappers take weeks to map a human's DNA)
  - How can we design a much higher performance read mapper?

Our First Step: Comprehensive Mapping

- + Guaranteed to find *all* mappings → sensitive
- + Can tolerate up to *e* errors

nature genetics

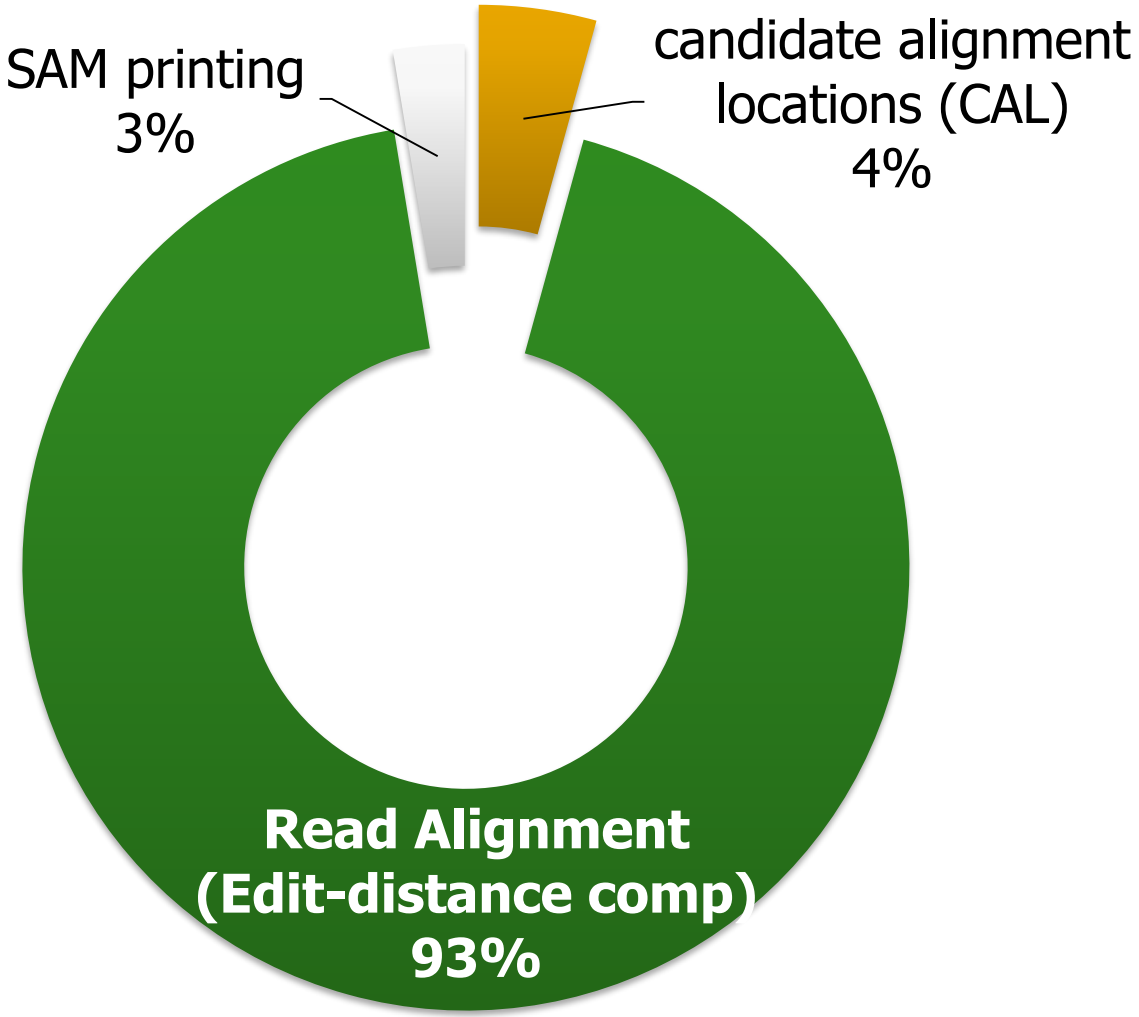
http://mrfast.sourceforge.net/

Personalized copy number and segmental duplication maps using next-generation sequencing

Can Alkan[1,2], Jeffrey M Kidd[1], Tomas Marques-Bonet[1,3], Gozde Aksay[1], Francesca Antonacci[1], Fereydoun Hormozdiari[4], Jacob O Kitzman[1], Carl Baker[1], Maika Malig[1], Onur Mutlu[5], S Cenk Sahinalp[4], Richard A Gibbs[6] & Evan E Eichler[1,2]

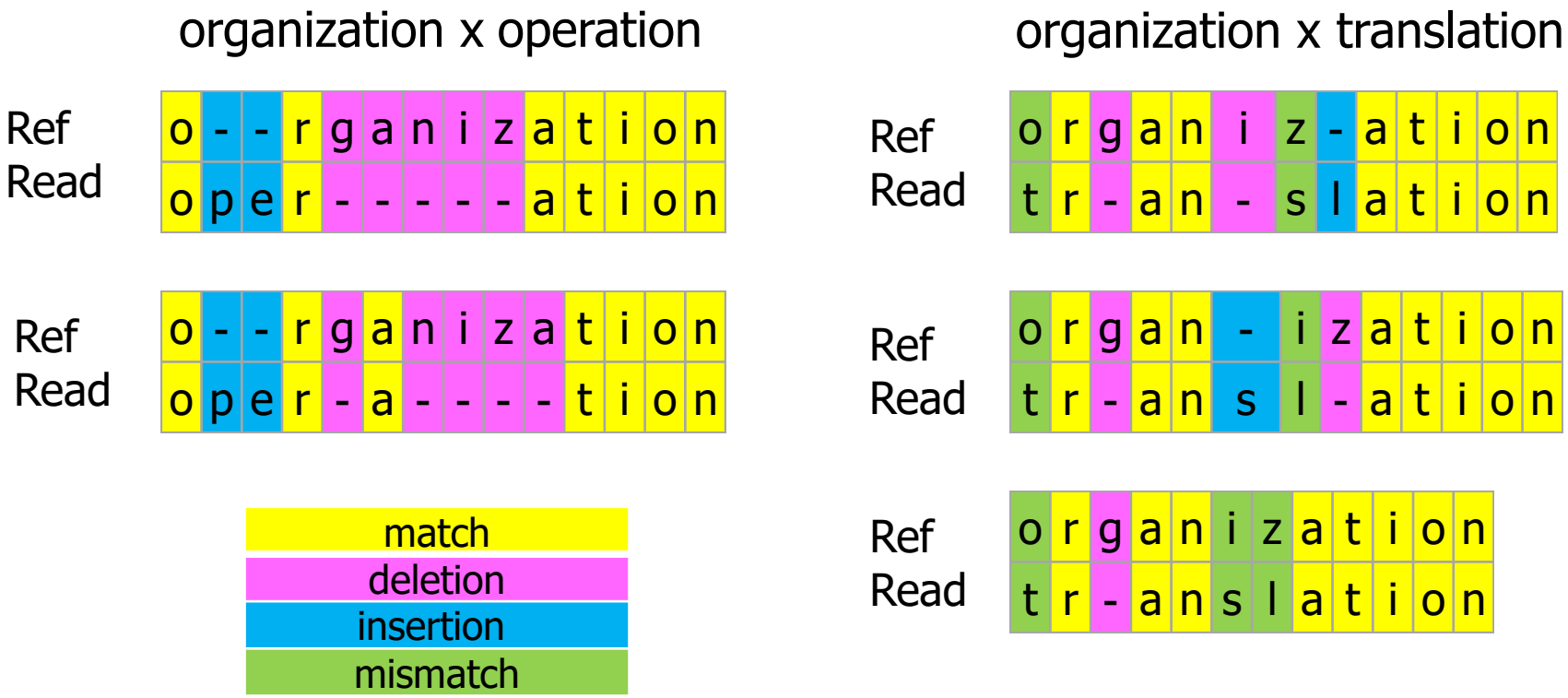
Alkan+, **"Personalized copy number and segmental duplication maps using next-generation sequencing"**, Nature Genetics 2009.

Read Mapping Execution Time Breakdown



Read Alignment/Verification

- **Edit distance** is defined as the minimum number of edits (i.e. insertions, deletions, or substitutions) needed to make the read exactly match the reference segment.



Idea

Filter fast before you align

Minimize costly
"approximate string comparisons"

Our First Filter: Pure Software Approach

- Download source code and try for yourself
  - Download link to FastHASH

Xin *et al. BMC Genomics* 2013, **14**(Suppl 1):S13
http://www.biomedcentral.com/1471-2164/14/S1/S13

BMC Genomics

PROCEEDINGS Open Access

Accelerating read mapping with FastHASH

Hongyi Xin[1], Donghyuk Lee[1], Farhad Hormozdiari[2], Samihan Yedkar[1], Onur Mutlu[1*], Can Alkan[3*]

From The Eleventh Asia Pacific Bioinformatics Conference (APBC 2013)
Vancouver, Canada. 21-24 January 2013

Next Step: SIMD Acceleration (New Algorithm)

Bioinformatics, 31(10), 2015, 1553–1560
doi: 10.1093/bioinformatics/btu856
Advance Access Publication Date: 10 January 2015
Original Paper

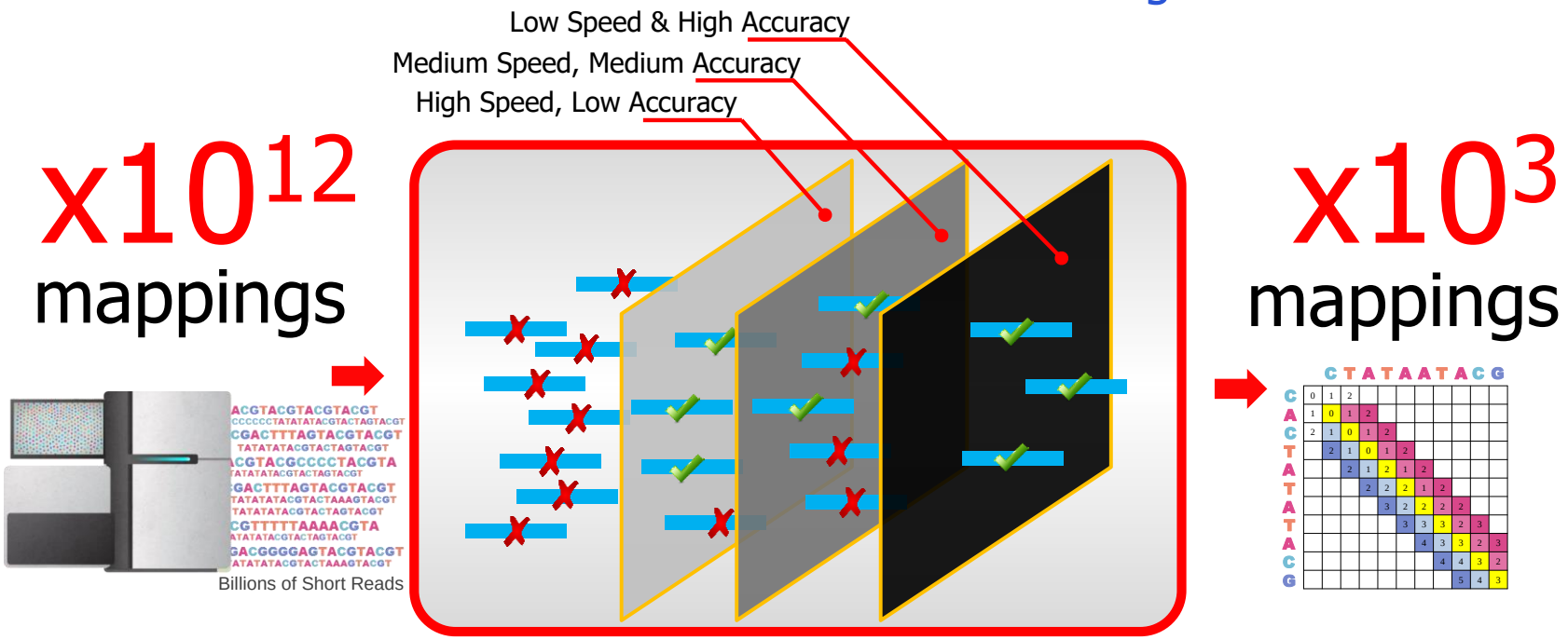
Sequence analysis

Shifted Hamming distance: a fast and accurate SIMD-friendly filter to accelerate alignment verification in read mapping

Hongyi Xin[1,*], John Greth[2], John Emmons[2], Gennady Pekhimenko[1], Carl Kingsford[3], Can Alkan[4,*] and Onur Mutlu[2,*]

Xin+, **"Shifted Hamming Distance: A Fast and Accurate SIMD-friendly Filter to Accelerate Alignment Verification in Read Mapping", Bioinformatics 2015.**

Our FPGA-based Filter: GateKeeper



1 **High throughput DNA sequencing (HTS) technologies**

2 **Read Pre-Alignment Filtering**
Fast & Low False Positive Rate

3 **Read Alignment**
Slow & Zero False Positives

FPGA-Based Alignment Filtering

- Mohammed Alser, Hasan Hassan, Hongyi Xin, Oguz Ergin, Onur Mutlu, and Can Alkan
  "GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping"
  Bioinformatics, [published online, May 31], 2017.
  [Source Code]
  [Online link at Bioinformatics Journal]

GateKeeper: a new hardware architecture for accelerating pre-alignment in DNA short read mapping

Mohammed Alser ✉, Hasan Hassan, Hongyi Xin, Oğuz Ergin, Onur Mutlu ✉, Can Alkan ✉

Bioinformatics, Volume 33, Issue 21, 1 November 2017, Pages 3355–3363,
https://doi.org/10.1093/bioinformatics/btx342

Published: 31 May 2017 **Article history** ▾

DNA Read Mapping & Filtering

- Problem: **Heavily bottlenecked by Data Movement**
- GateKeeper FPGA performance limited by DRAM bandwidth [Alser+, Bioinformatics 2017]
- Ditto for SHD on SIMD [Xin+, Bioinformatics 2015]
- Solution: Processing-in-memory can alleviate the bottleneck
- However, we need to design mapping & filtering algorithms to fit processing-in-memory

In-Memory DNA Sequence Analysis

- Jeremie S. Kim, Damla Senol Cali, Hongyi Xin, Donghyuk Lee, Saugata Ghose, Mohammed Alser, Hasan Hassan, Oguz Ergin, Can Alkan, and Onur Mutlu, **"GRIM-Filter: Fast Seed Location Filtering in DNA Read Mapping Using Processing-in-Memory Technologies"**
  BMC Genomics, 2018.
  *Proceedings of the 16th Asia Pacific Bioinformatics Conference (**APBC**)*, Yokohama, Japan, January 2018.
  arxiv.org Version (pdf)

GRIM-Filter: Fast seed location filtering in DNA read mapping using processing-in-memory technologies

Jeremie S. Kim[1,6*], Damla Senol Cali[1], Hongyi Xin[2], Donghyuk Lee[3], Saugata Ghose[1], Mohammed Alser[4], Hasan Hassan[6], Oguz Ergin[5], Can Alkan[4*] and Onur Mutlu[6,1*]

From The Sixteenth Asia Pacific Bioinformatics Conference 2018
Yokohama, Japan. 15-17 January 2018

Key Principles and Results

- Two key principles:
  - Exploit the structure of the genome to minimize computation
  - Morph and exploit the structure of the underlying hardware to maximize performance and efficiency

- **Algorithm-architecture co-design** for DNA read mapping
  - **Speeds up** read mapping by **~200X (sometimes more)**
  - **Improves accuracy** of read mapping in the presence of errors

Xin et al., "Accelerating Read Mapping with FastHASH," BMC Genomics 2013.

Xin et al., "Shifted Hamming Distance: A Fast and Accurate SIMD-friendly Filter to Accelerate Alignment Verification in Read Mapping," Bioinformatics 2015.

Alser et al., "GateKeeper: A New Hardware Architecture for Accelerating Pre-Alignment in DNA Short Read Mapping," Bioinformatics 2017.

Kim et al., "Genome Read In-Memory (GRIM) Filter," BMC Genomics 2018.

New Genome Sequencing Technologies

Nanopore sequencing technology and tools for genome assembly: computational analysis of the current state, bottlenecks and future directions

Damla Senol Cali ✉, Jeremie S Kim, Saugata Ghose, Can Alkan, Onur Mutlu

Briefings in Bioinformatics, bby017, https://doi.org/10.1093/bib/bby017
Published: 02 April 2018 **Article history** ▾

Oxford Nanopore MinION

Senol Cali+, "**Nanopore Sequencing Technology and Tools for Genome Assembly: Computational Analysis of the Current State, Bottlenecks and Future Directions**," *Briefings in Bioinformatics* (***BIB***), 2018.
[Open arxiv.org version]

Nanopore Genome Assembly Pipeline

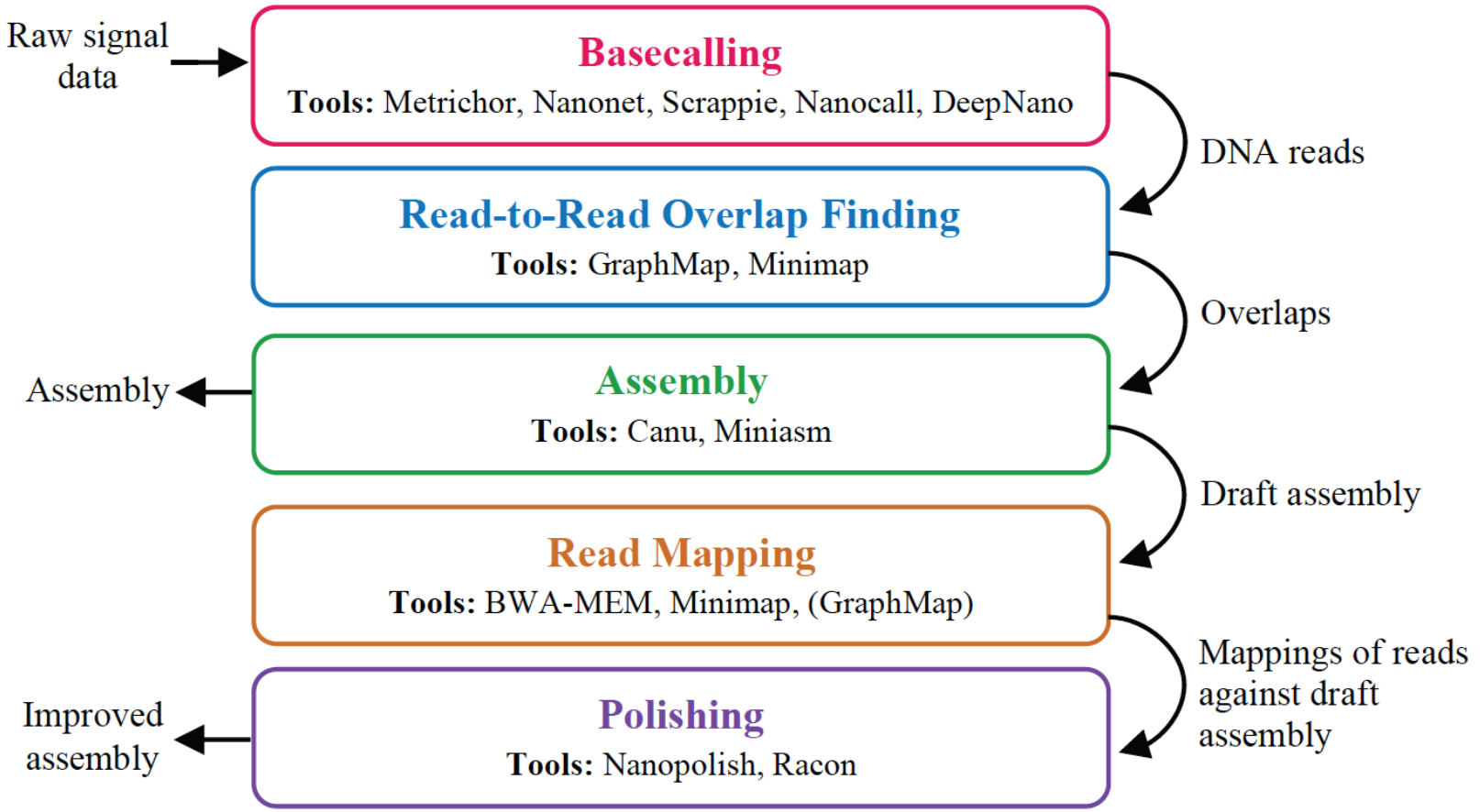


Figure 1. The analyzed genome assembly pipeline using nanopore sequence data, with its five steps and the associated tools for each step.

Senol Cali+, "**Nanopore Sequencing Technology and Tools for Genome Assembly**", Briefings in Bioinformatics, 2018. [Open arxiv.org version]

More on Genome Analysis: Another Talk

Accelerating Genome Analysis

A Primer on an Ongoing Journey

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

May 21, 2018

HiCOMB-17 Keynote Talk

SAFARI

ETH zürich

Recall Our Dream

- Can we build devices that can analyze a genome within a minute?

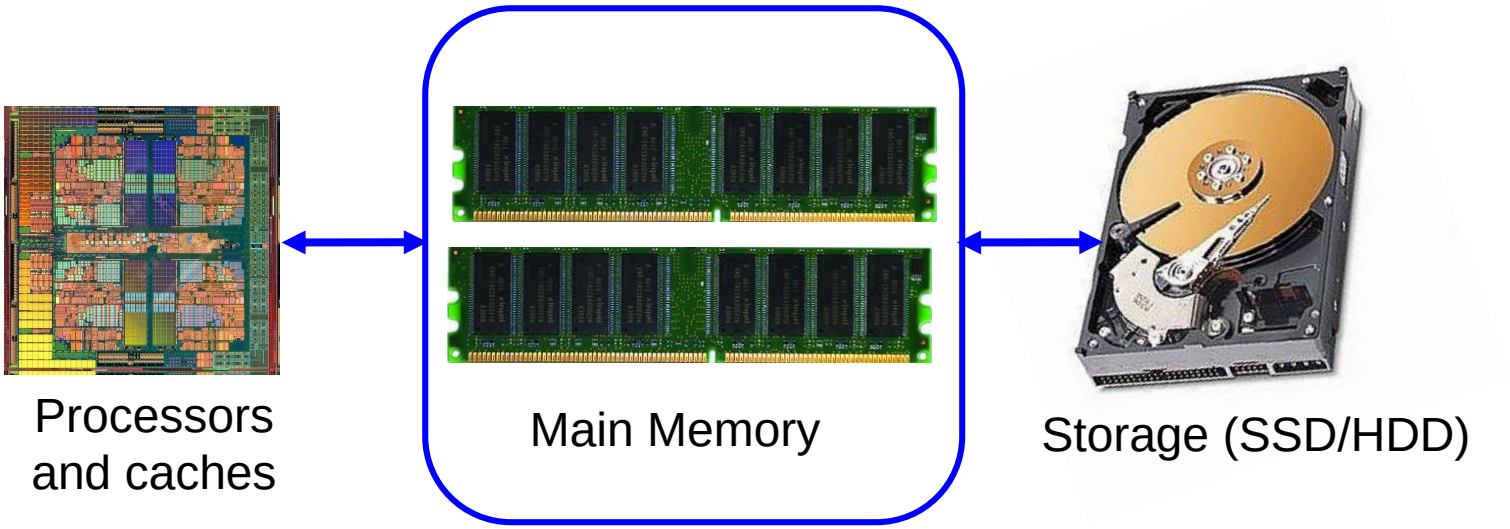
- Still a long ways to go
  - Energy efficiency
  - Performance (latency)
  - Security
  - Huge memory bottleneck

Four Key Directions

- Fundamentally Secure/Reliable/Safe Architectures
- Fundamentally Energy-Efficient Architectures
  - Memory-centric (Data-centric) Architectures
- Fundamentally Low-Latency Architectures
- Architectures for Genomics, Medicine, Health

Memory & Storage

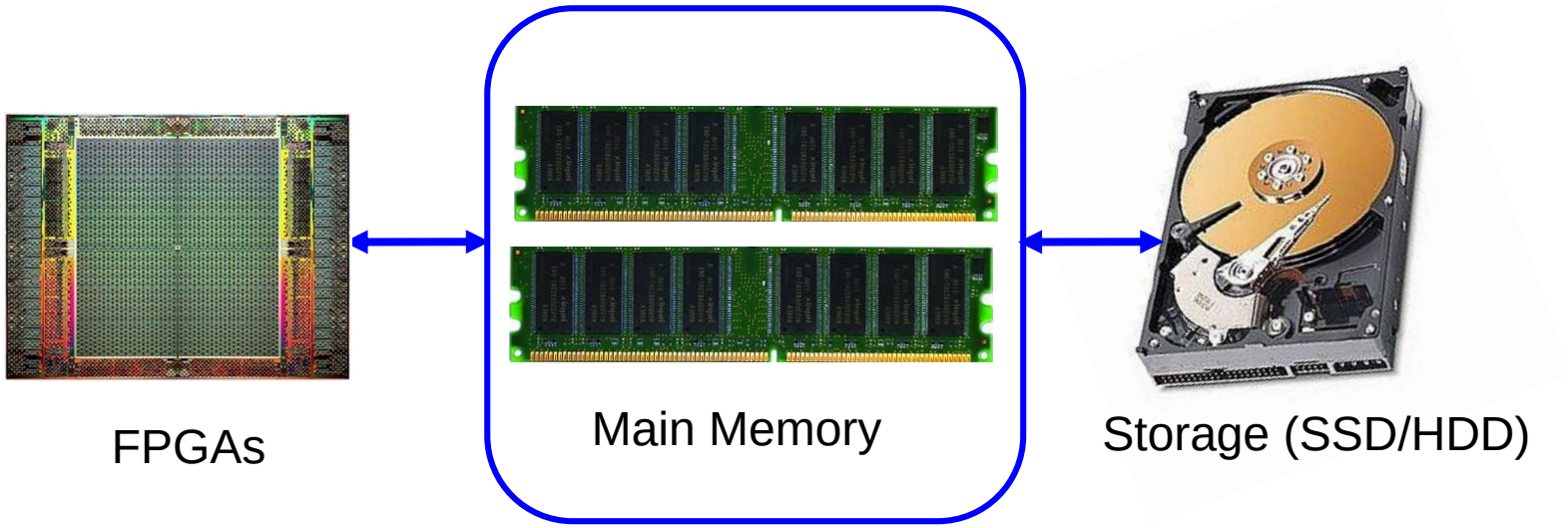
The Main Memory System



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

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

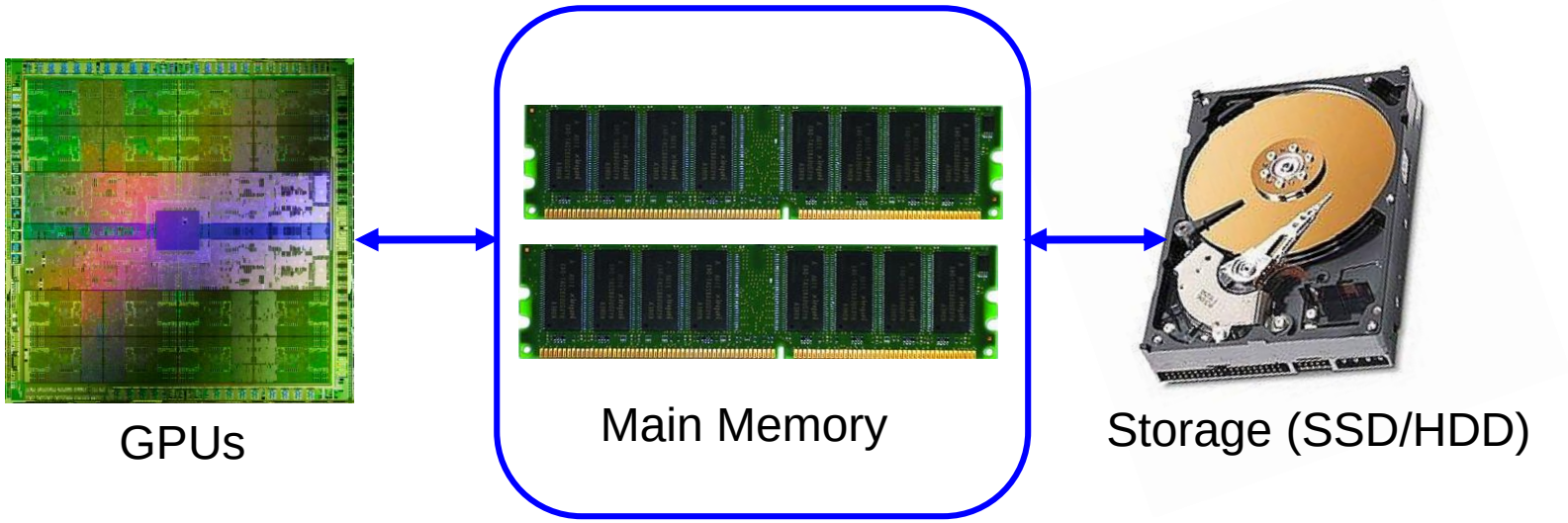
The Main Memory System



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

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

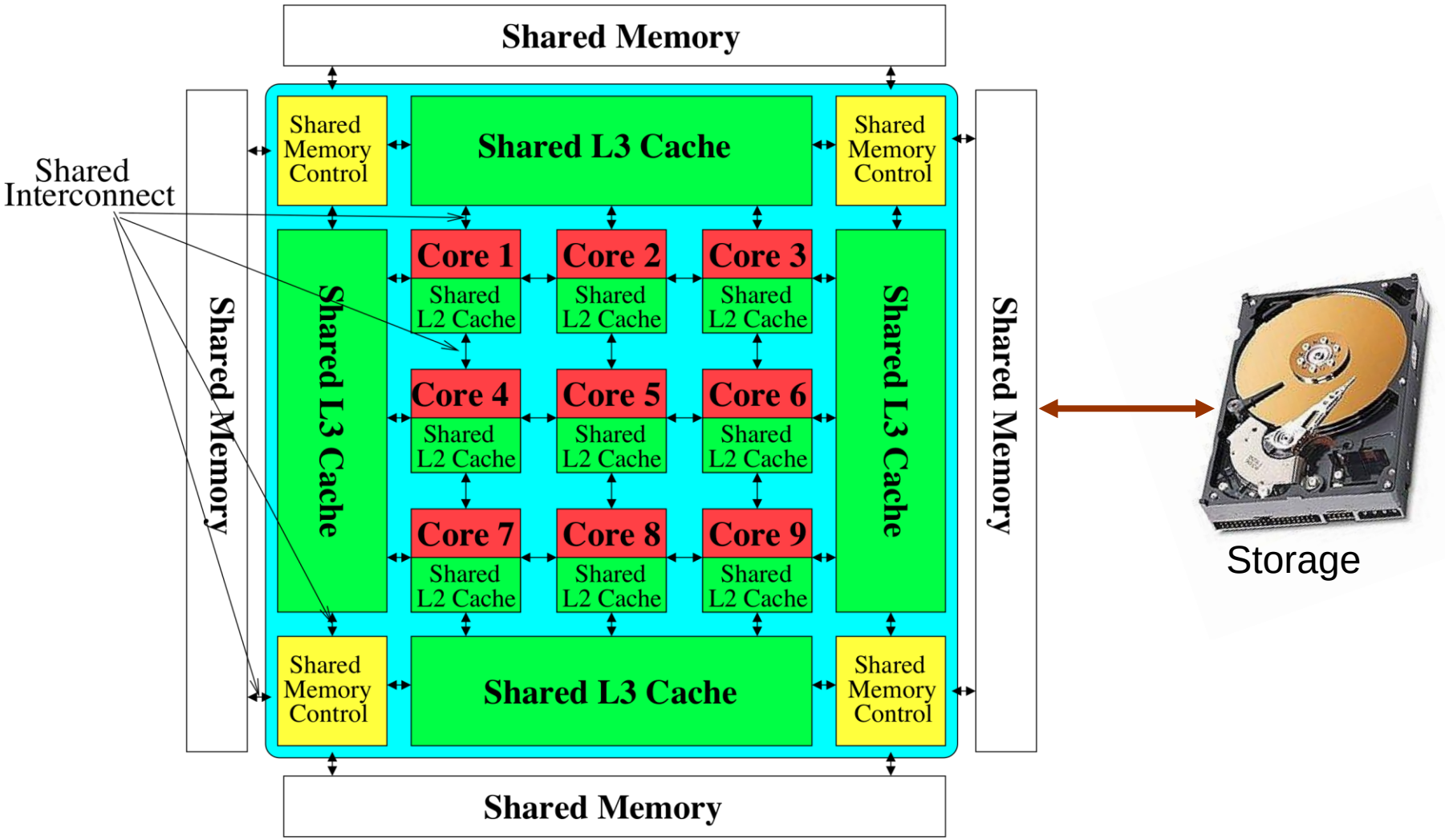
The Main Memory System



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

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

Memory System: A *Shared Resource* View



Most of the system is dedicated to storing and moving data

State of the Main Memory System

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

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

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

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

Major Trends Affecting Main Memory (I)

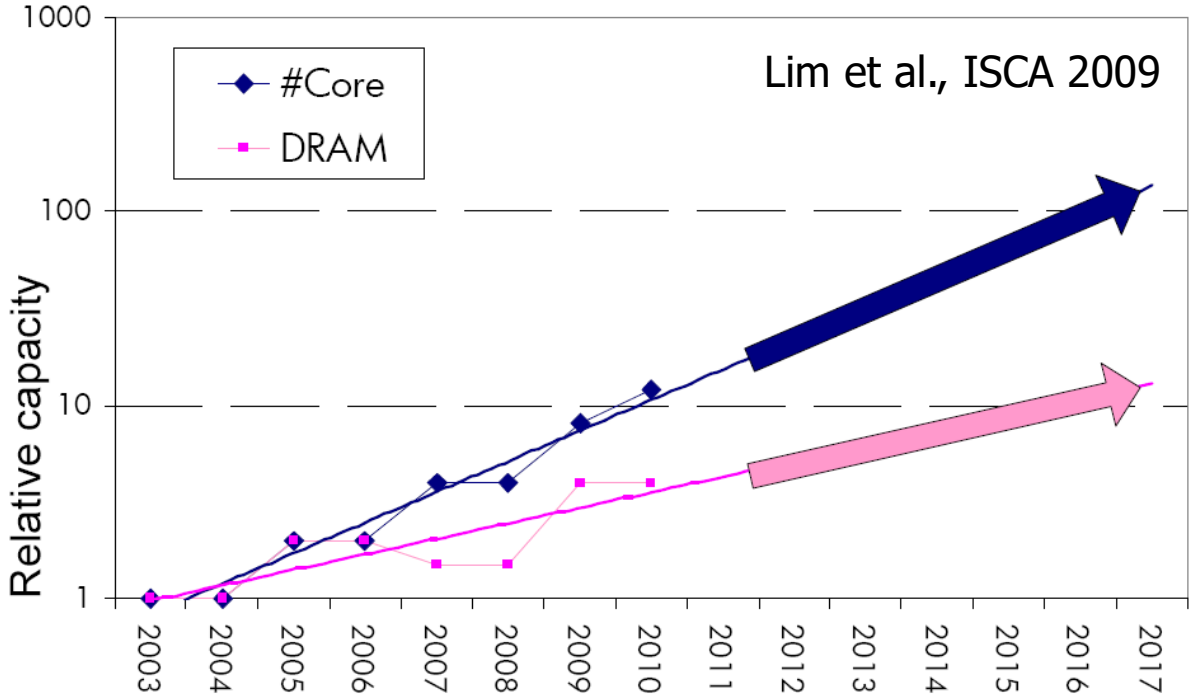
- Need for main memory capacity, bandwidth, QoS increasing
- Main memory energy/power is a key system design concern
- DRAM technology scaling is ending

Major Trends Affecting Main Memory (II)

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

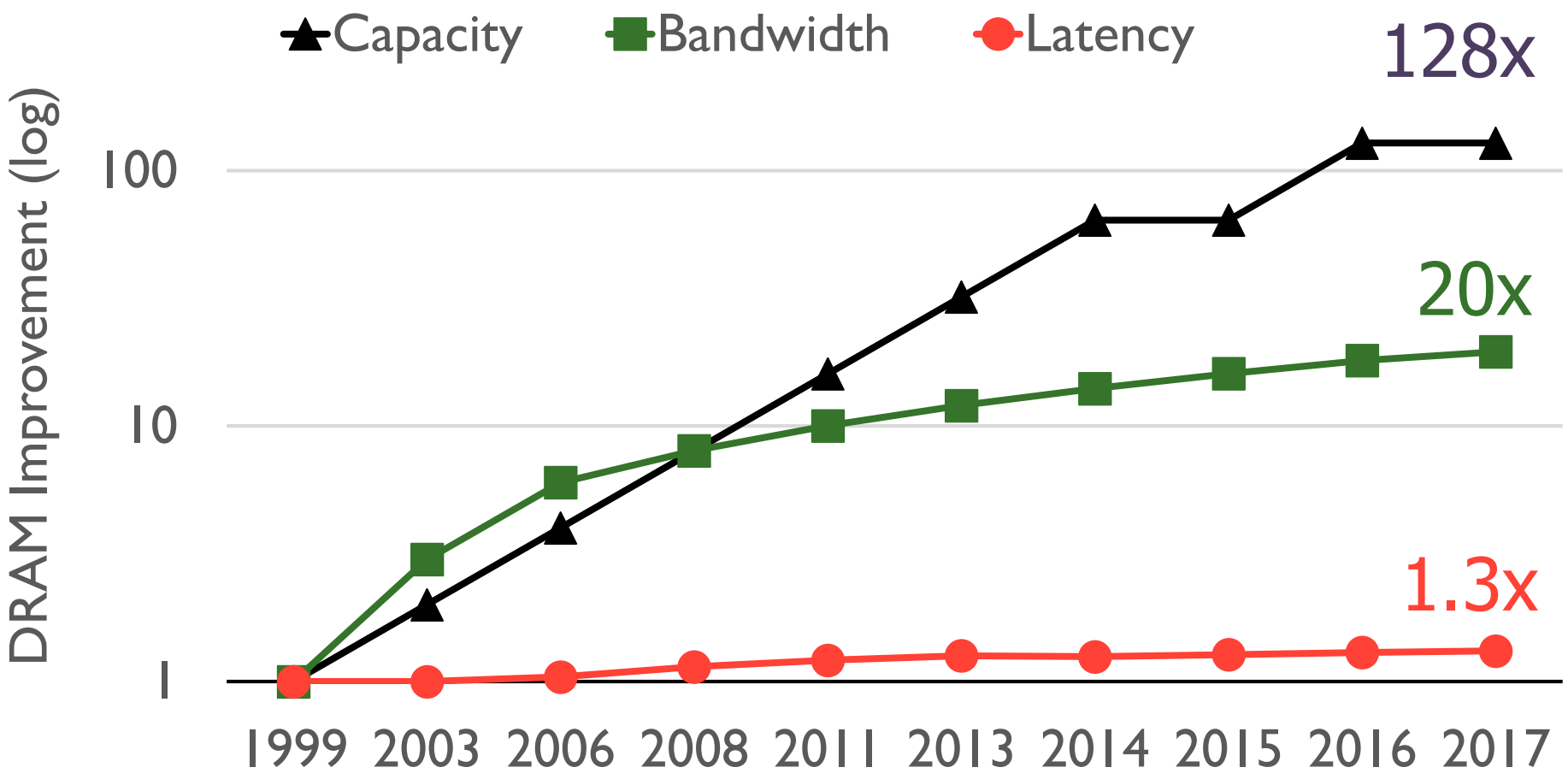
Consequence: The Memory Capacity Gap

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



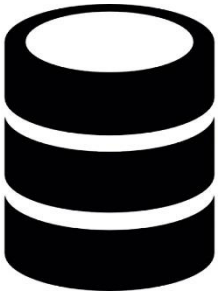
- *Memory capacity per core* expected to drop by 30% every two years
- Trends worse for *memory bandwidth per core*!

Example: Capacity, Bandwidth & Latency

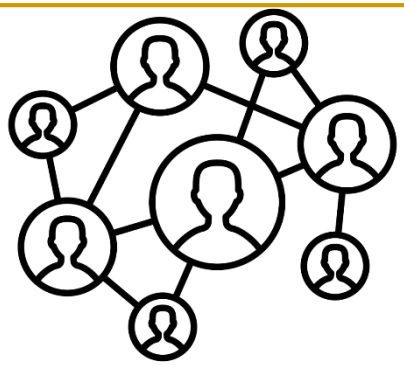


Memory latency remains almost constant

DRAM Latency Is Critical for Performance

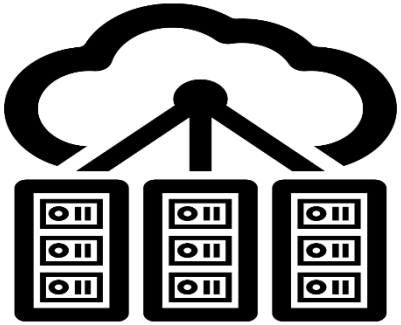


In-memory Databases
[Mao+, EuroSys'12;
Clapp+ (**Intel**), IISWC'15]



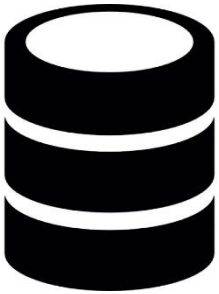
Graph/Tree Processing
[Xu+, IISWC'12; Umuroglu+, FPL'15]

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

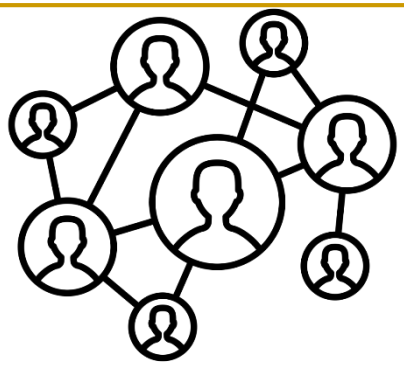


Datacenter Workloads
[Kanev+ (**Google**), ISCA'15]

DRAM Latency Is Critical for Performance



In-memory Databases

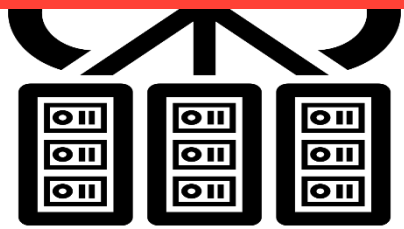


Graph/Tree Processing

Long memory latency → performance bottleneck

In-Memory Data Analytics

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



Datacenter Workloads

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

Major Trends Affecting Main Memory (III)

- Need for main memory capacity, bandwidth, QoS increasing

- Main memory energy/power is a key system design concern
  - ~40-50% energy spent in off-chip memory hierarchy [Lefurgy, IEEE Computer'03] >40% power in DRAM [Ware, HPCA'10][Paul,ISCA'15]
  - DRAM consumes power even when not used (periodic refresh)

- DRAM technology scaling is ending

Major Trends Affecting Main Memory (IV)

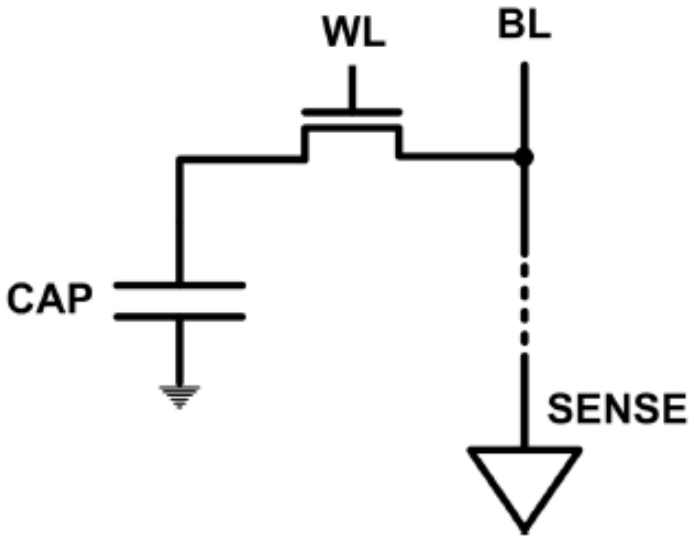
- Need for main memory capacity, bandwidth, QoS increasing

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

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

The DRAM Scaling Problem

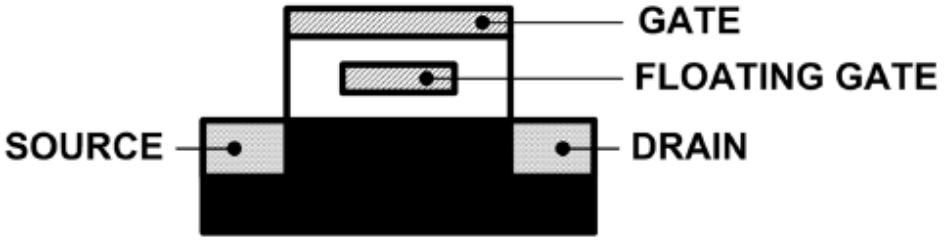
- DRAM stores charge in a capacitor (charge-based memory)
  - Capacitor must be large enough for reliable sensing
  - Access transistor should be large enough for low leakage and high retention time
  - Scaling beyond 40-35nm (2013) is challenging [ITRS, 2009]

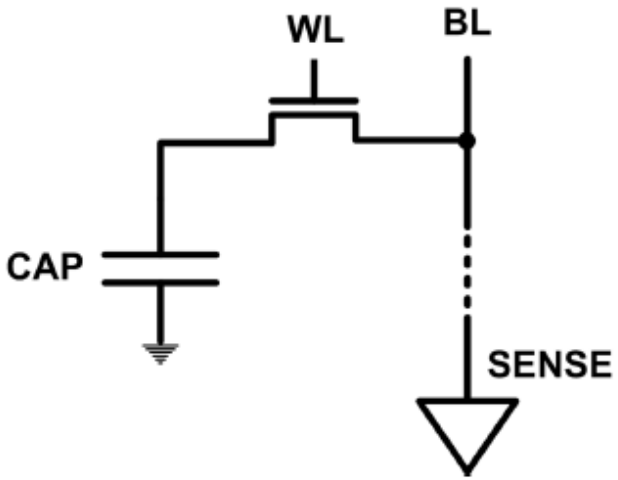


- DRAM capacity, cost, and energy/power hard to scale

Limits of Charge Memory

- Difficult charge placement and control
  - Flash: floating gate charge
  - DRAM: capacitor charge, transistor leakage
- Reliable sensing becomes difficult as charge storage unit size reduces





Major Trends Affecting Main Memory (V)

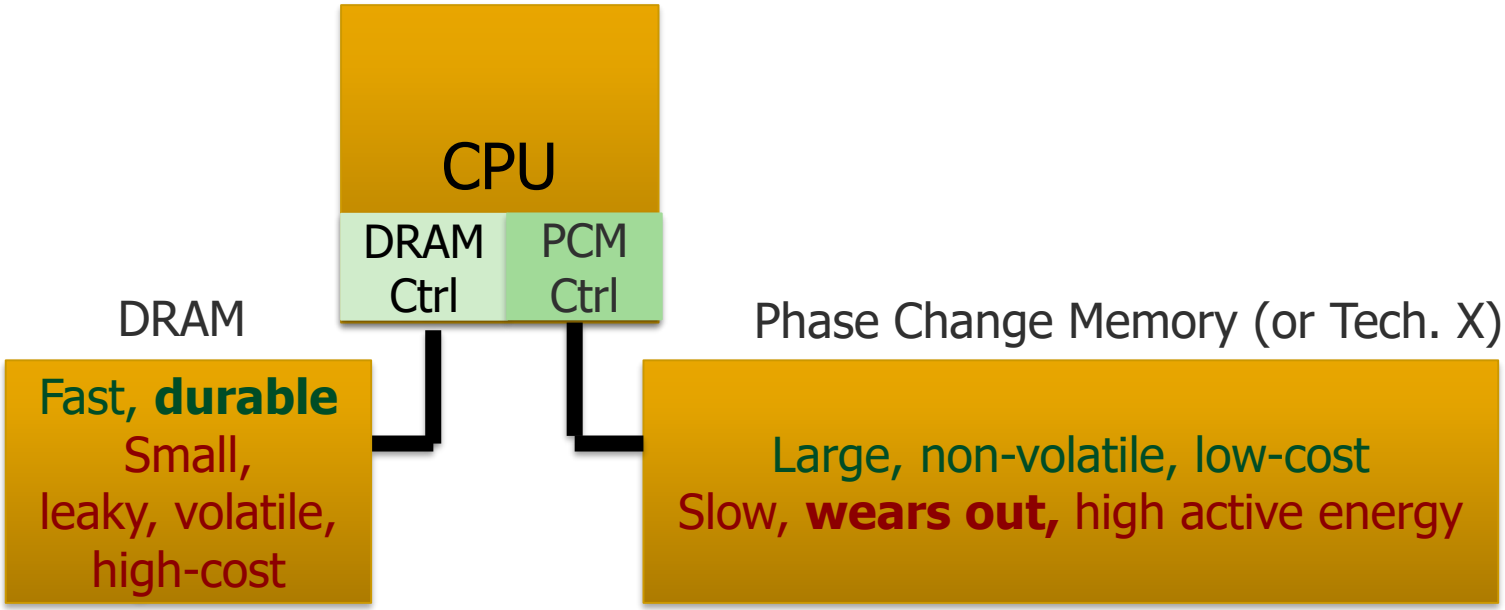
- DRAM scaling has already become increasingly difficult
  - Increasing cell leakage current, reduced cell reliability, increasing manufacturing difficulties [Kim+ ISCA 2014], [Liu+ ISCA 2013], [Mutlu IMW 2013], [Mutlu DATE 2017]
  - **Difficult to significantly improve capacity, energy**
- **Emerging memory technologies** are promising

Major Trends Affecting Main Memory (V)

- DRAM scaling has already become increasingly difficult
  - Increasing cell leakage current, reduced cell reliability, increasing manufacturing difficulties [Kim+ ISCA 2014], [Liu+ ISCA 2013], [Mutlu IMW 2013], [Mutlu DATE 2017]
  - **Difficult to significantly improve capacity, energy**
- **Emerging memory technologies** are promising

3D-Stacked DRAM	higher bandwidth	smaller capacity
Reduced-Latency DRAM (e.g., RL/TL-DRAM, FLY-RAM)	lower latency	higher cost
Low-Power DRAM (e.g., LPDDR3, LPDDR4, Voltron)	lower power	higher latency higher cost
Non-Volatile Memory (NVM) (e.g., PCM, STTRAM, ReRAM, 3D Xpoint)	larger capacity	higher latency higher dynamic power lower endurance

Major Trend: Hybrid Main Memory



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.

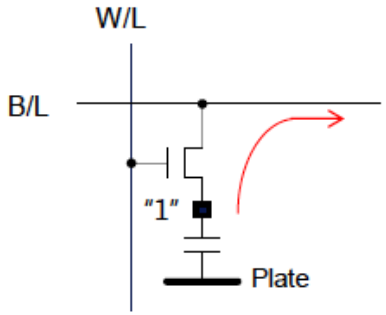
One Foreshadowing

Main Memory Needs Intelligent Controllers

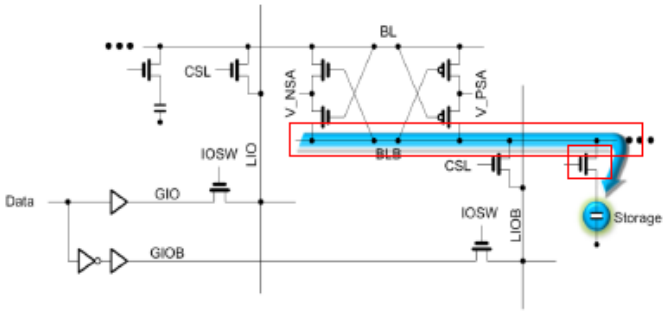
Industry Is Writing Papers About It, Too

DRAM Process Scaling Challenges

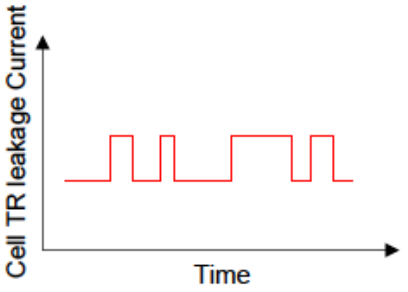
- **Refresh**
  - Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance
  - Leakage current of cell access transistors increasing
- **tWR**
  - Contact resistance between the cell capacitor and access transistor increasing
  - On-current of the cell access transistor decreasing
  - Bit-line resistance increasing
- **VRT**
  - Occurring more frequently with cell capacitance decreasing



Refresh



tWR



VRT

Call for Intelligent Memory Controllers

DRAM Process Scaling Challenges

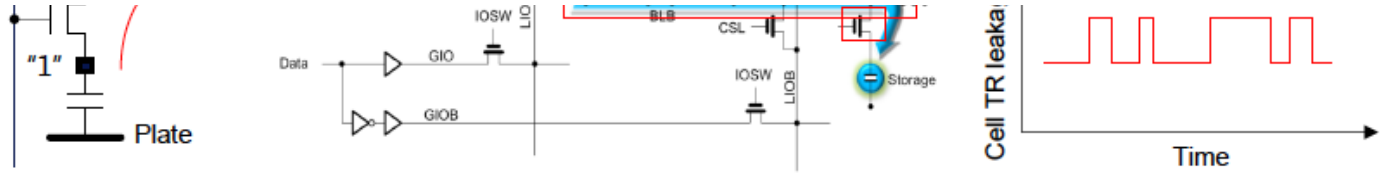
- **Refresh**
  - Difficult to build high-aspect ratio cell capacitors decreasing cell capacitance

THE MEMORY FORUM 2014

Co-Architecting Controllers and DRAM to Enhance DRAM Process Scaling

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

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



Refresh **tWR** **VRT**

Agenda

- Brief Introduction
- A Motivating Example
- Memory System Trends
- What Will You Learn In This Course
  - And, how to make the best of it…
- Memory Fundamentals
- Key Memory Challenges and Solution Directions
  - Security, Reliability, Safety
  - Energy and Performance: Data-Centric Systems
  - Latency and Latency-Reliability Tradeoffs
- Summary and Future Lookout

Course Logistics

What Will You Learn in This Course?

- Memory Systems and Memory-Centric Computing Systems
  - July 9-13, 2018

- Topic 1: Main Memory Trends and Basics
- Topic 2: Memory Reliability & Security: RowHammer and Beyond
- Topic 3: In-memory Computation
- Topic 4: Low-Latency and Low-Energy Memory
- Topic 5 (unlikely): Enabling and Exploiting Non-Volatile Memory
- Topic 6 (unlikely): Flash Memory and SSD Scaling

- Major Overview Reading:
  - Mutlu and Subramanian, "Research Problems and Opportunities in Memory Systems," SUPERFRI 2014.

This Course

- Will cover many problems and potential solutions related to the design of memory systems in the many core era

- The design of the memory system poses many
  - Difficult research and engineering problems
  - Important fundamental problems
  - Industry-relevant problems
  - **Problems whose solutions can revolutionize the world**

- Many creative and insightful solutions are needed to solve these problems

- Goal: Acquire the basics to develop such solutions (by covering fundamentals and cutting edge research)

Course Information

- My Contact Information
  - Onur Mutlu
  - omutlu@gmail.com
  - https://people.inf.ethz.ch/omutlu
  - +41-79-572-1444 (my cell phone)
  - Find me during breaks and/or email any time.

- Website for Course Slides, Papers, Updates
  - https://people.inf.ethz.ch/omutlu/acaces2018.html

- For the curious:
  - ACACES 2013 Course: Scalable Memory Systems
  - https://people.inf.ethz.ch/omutlu/acaces2013-memory.html

How To Make the Best Out of This Course

- Be alert during lectures – they will be fast paced
- Do the readings (and explore even more)
  - I will provide many references
- Go back and reinforce fundamentals (as needed)
  - I will provide pointers to basic computer architecture materials (lecture videos, slides, readings, exams, …)

- Remember "Chance favors the prepared mind." (Pasteur)

Unfortunately, No Time For:

- Memory Interference and QoS
- Predictable Performance
  - QoS-aware Memory Controllers
- Emerging Memory Technologies and Hybrid Memories
- Cache Management
- Interconnects
- You can find many materials on these at my online lectures
  - https://people.inf.ethz.ch/omutlu/teaching.html

Readings, Videos, Reference Materials

Reference Overview Paper I

Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions

SAUGATA GHOSE, KEVIN HSIEH, AMIRALI BOROUMAND,
RACHATA AUSAVARUNGNIRUN
Carnegie Mellon University

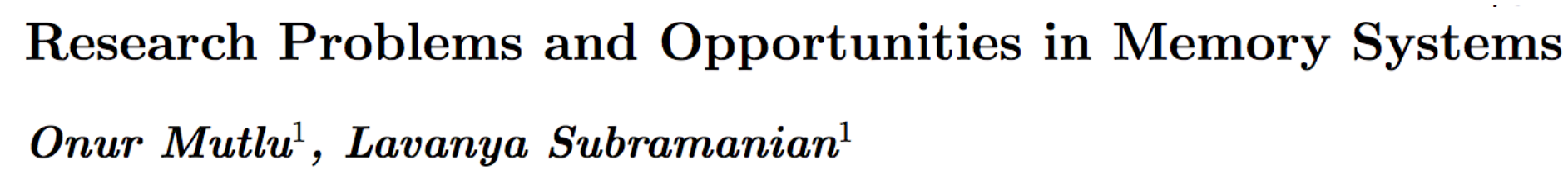
ONUR MUTLU
ETH Zürich and Carnegie Mellon University

Saugata Ghose, Kevin Hsieh, Amirali Boroumand, Rachata Ausavarungnirun, Onur Mutlu, **"Enabling the Adoption of Processing-in-Memory: Challenges, Mechanisms, Future Research Directions"**
Invited Book Chapter, to appear in 2018.
[Preliminary arxiv.org version]

https://arxiv.org/pdf/1802.00320.pdf

Reference Overview Paper II

- Onur Mutlu and Lavanya Subramanian,
  "Research Problems and Opportunities in Memory Systems"
  *Invited Article in Supercomputing Frontiers and Innovations (**SUPERFRI**)*, 2014/2015.

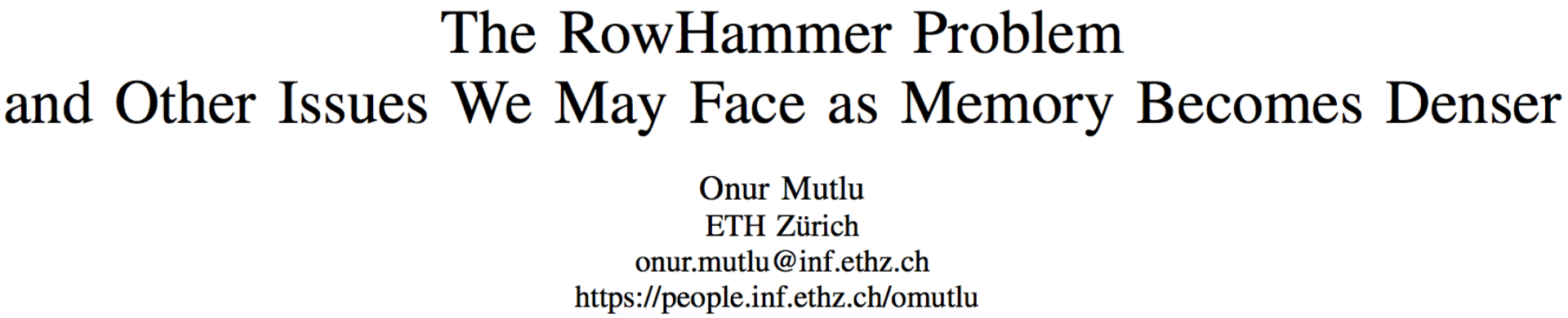

Research Problems and Opportunities in Memory Systems

Onur Mutlu[1], *Lavanya Subramanian*[1]

https://people.inf.ethz.ch/omutlu/pub/memory-systems-research_superfri14.pdf

Reference Overview Paper III

- Onur Mutlu,
  "The RowHammer Problem and Other Issues We May Face as Memory Becomes Denser"
  *Invited Paper in Proceedings of the Design, Automation, and Test in Europe Conference (**DATE**)*, Lausanne, Switzerland, March 2017.
  [Slides (pptx) (pdf)]

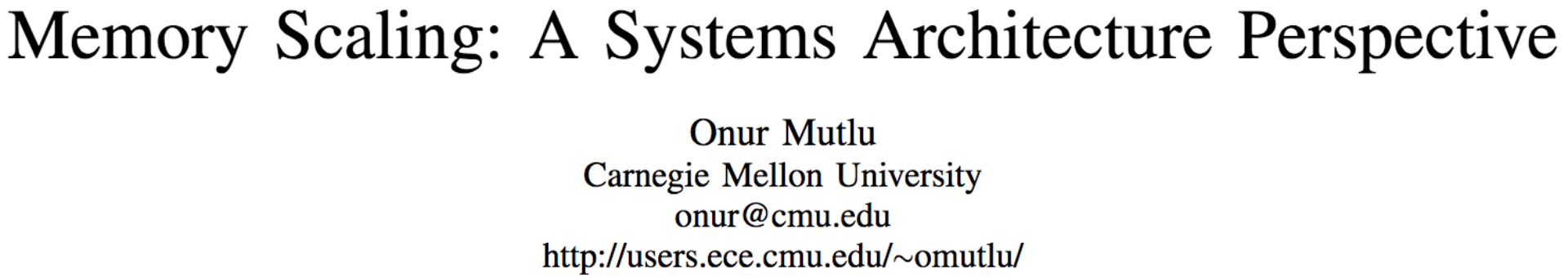

The RowHammer Problem
and Other Issues We May Face as Memory Becomes Denser

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

https://people.inf.ethz.ch/omutlu/pub/rowhammer-and-other-memory-issues_date17.pdf

Reference Overview Paper IV

- Onur Mutlu,
  "Memory Scaling: A Systems Architecture Perspective"
  *Technical talk at MemCon 2013 (**MEMCON**)*, Santa Clara, CA, August 2013. [Slides (pptx) (pdf)]
  [Video] [Coverage on StorageSearch]



Memory Scaling: A Systems Architecture Perspective

Onur Mutlu
Carnegie Mellon University
onur@cmu.edu
http://users.ece.cmu.edu/~omutlu/

https://people.inf.ethz.ch/omutlu/pub/memory-scaling_memcon13.pdf

Reference Overview Paper V

Proceedings of the IEEE, Sept. 2017

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD's reliability and lifetime.

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

https://arxiv.org/pdf/1706.08642

Related Videos and Course Materials (I)

- Undergraduate Computer Architecture Course Lecture Videos (2015, 2014, 2013)
- Undergraduate Computer Architecture Course Materials (2015, 2014, 2013)

- Graduate Computer Architecture Course Lecture Videos (2017, 2015, 2013)
- Graduate Computer Architecture Course Materials (2017, 2015, 2013)

- Parallel Computer Architecture Course Materials (Lecture Videos)

Related Videos and Course Materials (II)

- **Freshman Digital Circuits and Computer Architecture Course Lecture Videos (2018, 2017)**
- **Freshman Digital Circuits and Computer Architecture Course Materials (2018)**

- **Memory Systems Short Course Materials**
  (Lecture Video on Main Memory and DRAM Basics)

Some Open Source Tools (I)

- Rowhammer – Program to Induce RowHammer Errors
  - https://github.com/CMU-SAFARI/rowhammer
- Ramulator – Fast and Extensible DRAM Simulator
  - https://github.com/CMU-SAFARI/ramulator
- MemSim – Simple Memory Simulator
  - https://github.com/CMU-SAFARI/memsim
- NOCulator – Flexible Network-on-Chip Simulator
  - https://github.com/CMU-SAFARI/NOCulator
- SoftMC – FPGA-Based DRAM Testing Infrastructure
  - https://github.com/CMU-SAFARI/SoftMC
- Other open-source software from my group
  - **https://github.com/CMU-SAFARI/**
  - **http://www.ece.cmu.edu/~safari/tools.html**

Some Open Source Tools (II)

- MQSim – A Fast Modern SSD Simulator
  - https://github.com/CMU-SAFARI/MQSim
- Mosaic – GPU Simulator Supporting Concurrent Applications
  - https://github.com/CMU-SAFARI/Mosaic
- IMPICA – Processing in 3D-Stacked Memory Simulator
  - https://github.com/CMU-SAFARI/IMPICA
- SMLA – Detailed 3D-Stacked Memory Simulator
  - https://github.com/CMU-SAFARI/SMLA
- HWASim – Simulator for Heterogeneous CPU-HWA Systems
  - https://github.com/CMU-SAFARI/HWASim

- Other open-source software from my group
  - **https://github.com/CMU-SAFARI/**
  - **http://www.ece.cmu.edu/~safari/tools.html**

More Open Source Tools (III)

- A lot more open-source software from my group
  - https://github.com/CMU-SAFARI/
  - http://www.ece.cmu.edu/~safari/tools.html

Referenced Papers

- All are available at

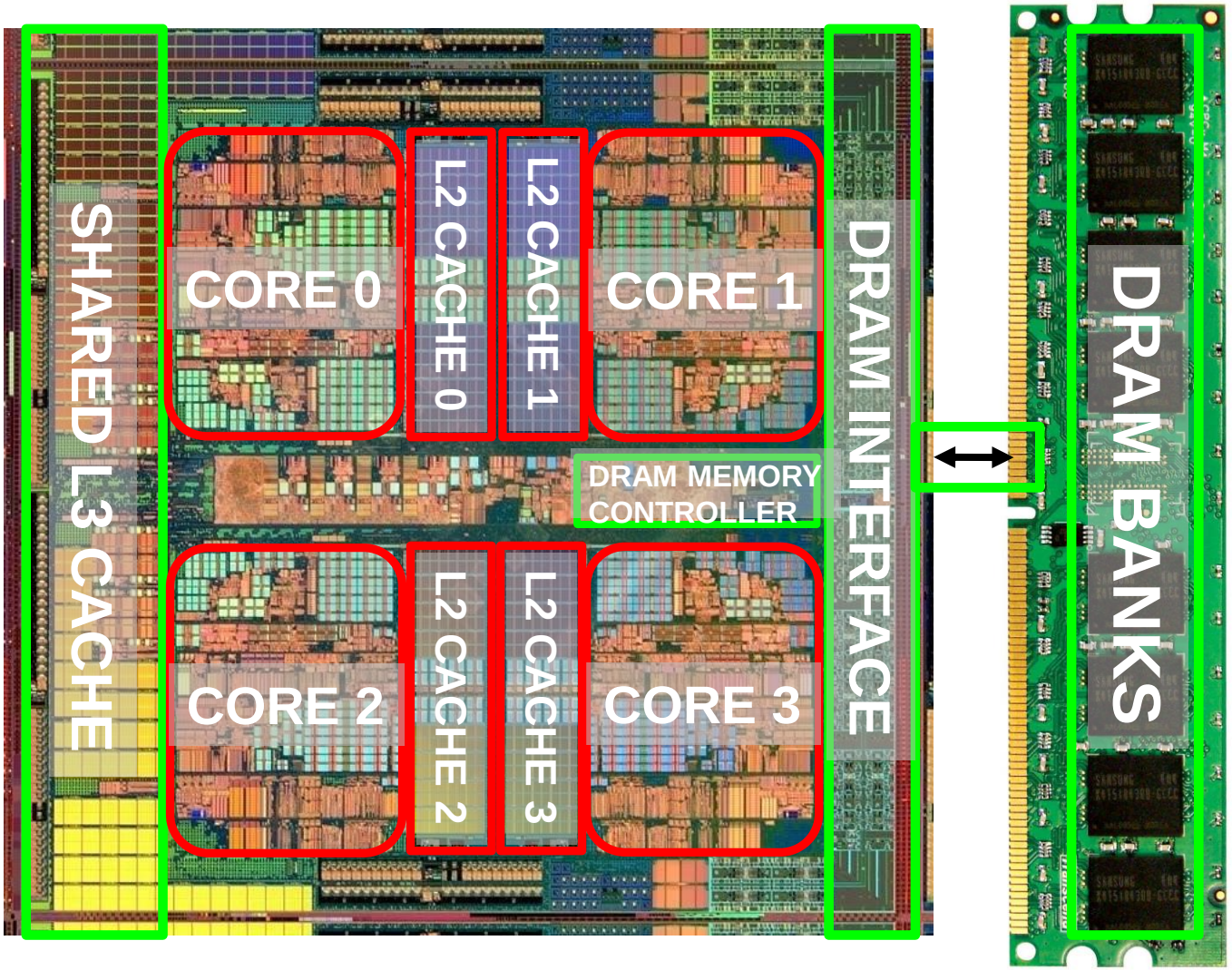
  https://people.inf.ethz.ch/omutlu/projects.htm

  http://scholar.google.com/citations?user=7XyGUGkAAAAJ&hl=en

  https://people.inf.ethz.ch/omutlu/acaces2018.html

Memory Fundamentals

Memory in a Modern System



Ideal Memory

- Zero access time (latency)
- Infinite capacity
- Zero cost
- Infinite bandwidth (to support multiple accesses in parallel)

The Problem

- Ideal memory's requirements oppose each other

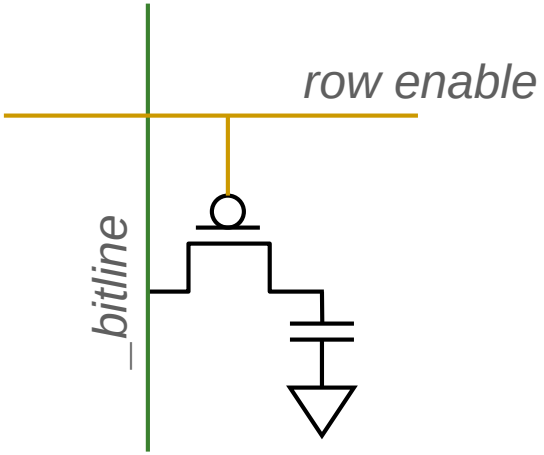
- Bigger is slower
  - Bigger → Takes longer to determine the location

- Faster is more expensive
  - Memory technology: SRAM vs. DRAM vs. Disk vs. Tape

- Higher bandwidth is more expensive
  - Need more banks, more ports, higher frequency, or faster technology

Memory Technology: DRAM

- Dynamic random access memory
- Capacitor charge state indicates stored value
  - Whether the capacitor is charged or discharged indicates storage of 1 or 0
  - 1 capacitor
  - 1 access transistor

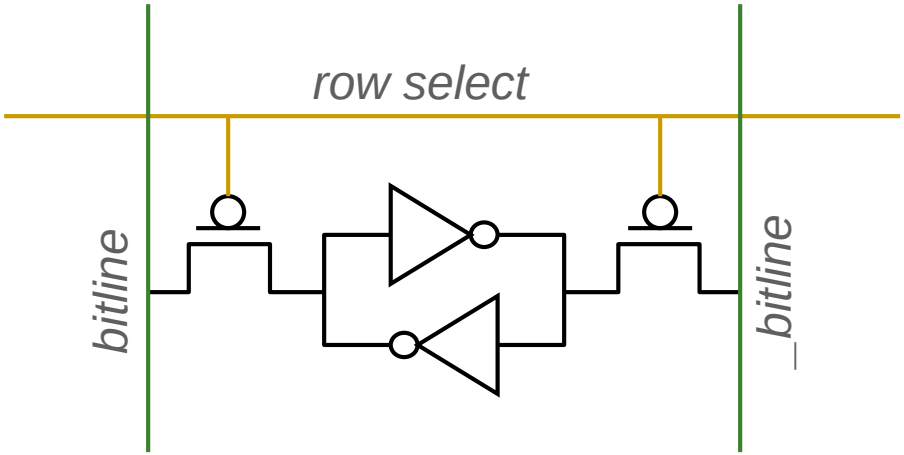


- Capacitor leaks through the RC path
  - DRAM cell loses charge over time
  - DRAM cell needs to be refreshed

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

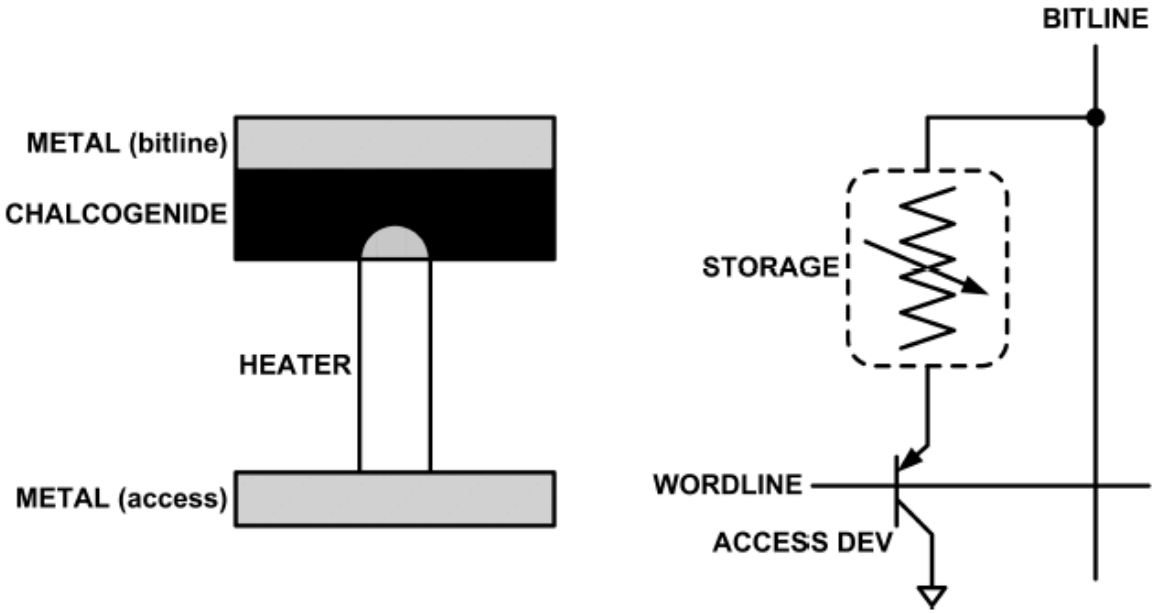
Memory Technology: SRAM

- Static random access memory
- Two cross coupled inverters store a single bit
  - Feedback path enables the stored value to persist in the "cell"
  - 4 transistors for storage
  - 2 transistors for access



An Aside: Phase Change Memory

- Phase change material (chalcogenide glass) exists in two states:
  - Amorphous: Low optical reflexivity and high electrical resistivity
  - Crystalline: High optical reflexivity and low electrical resistivity

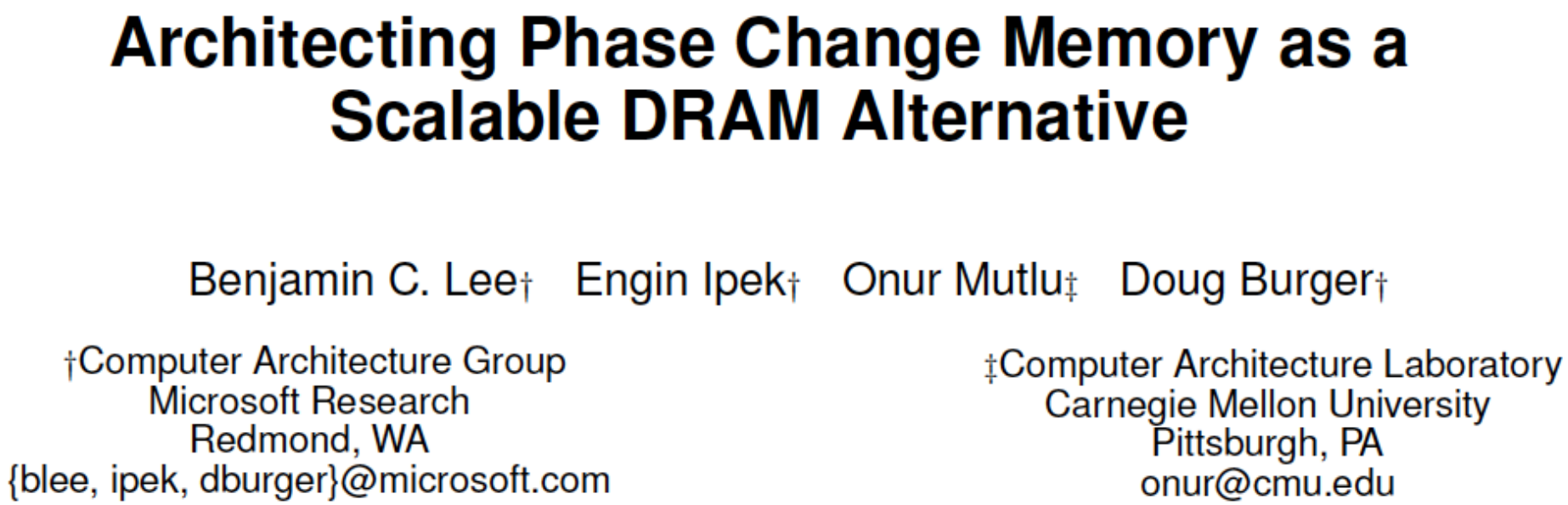


PCM is resistive memory: High resistance (0), Low resistance (1)

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

Reading: PCM As Main Memory

- Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger, **"Architecting Phase Change Memory as a Scalable DRAM Alternative"**
  *Proceedings of the 36th International Symposium on Computer Architecture (**ISCA**)*, pages 2-13, Austin, TX, June 2009. Slides (pdf)



Architecting Phase Change Memory as a Scalable DRAM Alternative

Benjamin C. Lee† Engin Ipek† Onur Mutlu‡ Doug Burger†

†Computer Architecture Group
Microsoft Research
Redmond, WA
{blee, ipek, dburger}@microsoft.com

‡Computer Architecture Laboratory
Carnegie Mellon University
Pittsburgh, PA
onur@cmu.edu

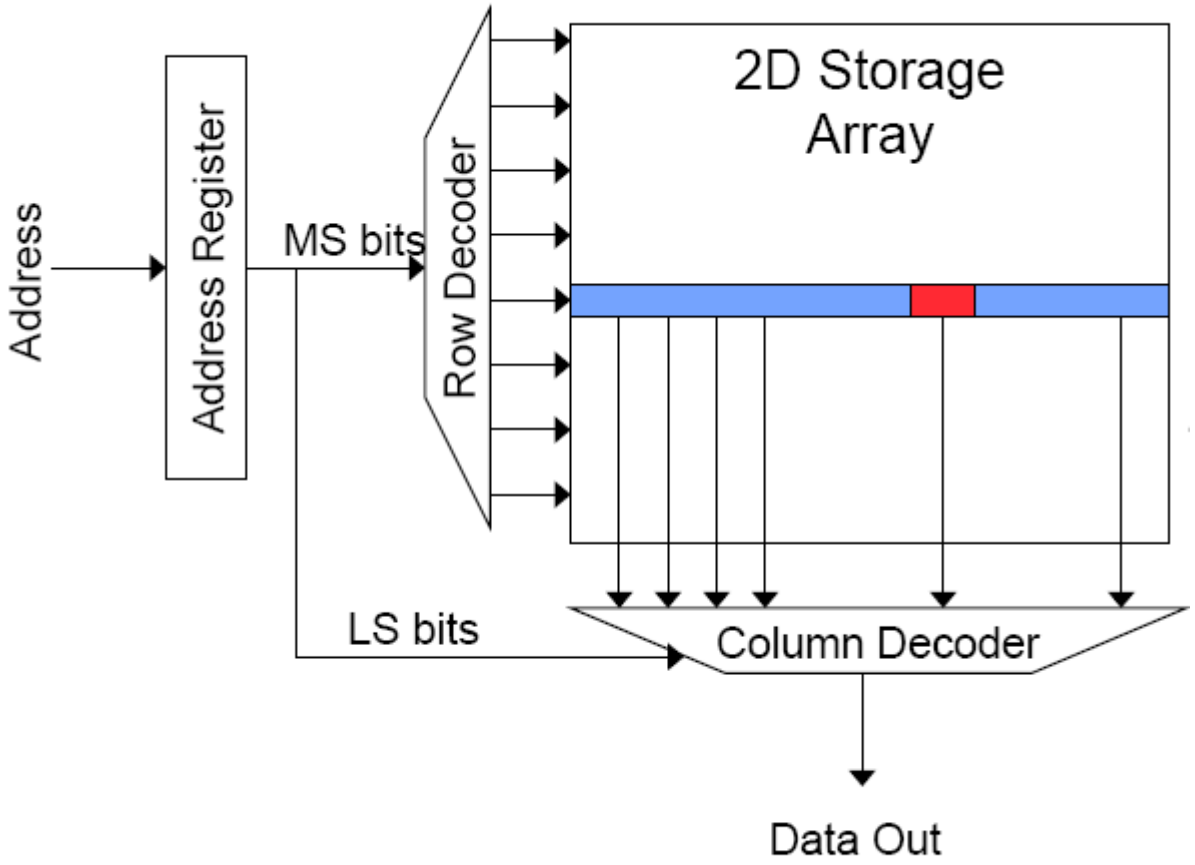
Reading: More on PCM As Main Memory

- Benjamin C. Lee, Ping Zhou, Jun Yang, Youtao Zhang, Bo Zhao, Engin Ipek, Onur Mutlu, and Doug Burger, **"Phase Change Technology and the Future of Main Memory"** *IEEE Micro*, *Special Issue: Micro's Top Picks from 2009 Computer Architecture Conferences (**MICRO TOP PICKS**)*, Vol. 30, No. 1, pages 60-70, January/February 2010.

Memory Bank: A Fundamental Concept

- Interleaving (banking)
  - Problem: a single monolithic memory array takes long to access and does not enable multiple accesses in parallel
  - Goal: Reduce the latency of memory array access and enable multiple accesses in parallel
  - Idea: Divide the array into multiple banks that can be accessed independently (in the same cycle or in consecutive cycles)
    - Each bank is smaller than the entire memory storage
    - Accesses to different banks can be overlapped
  - An issue: How do you map data to different banks? (i.e., how do you interleave data across banks?)

Memory Bank Organization and Operation



- Read access sequence:

1. Decode row address & drive word-lines

2. Selected bits drive bit-lines
   - Entire row read

3. Amplify row data

4. Decode column address & select subset of row
   - Send to output

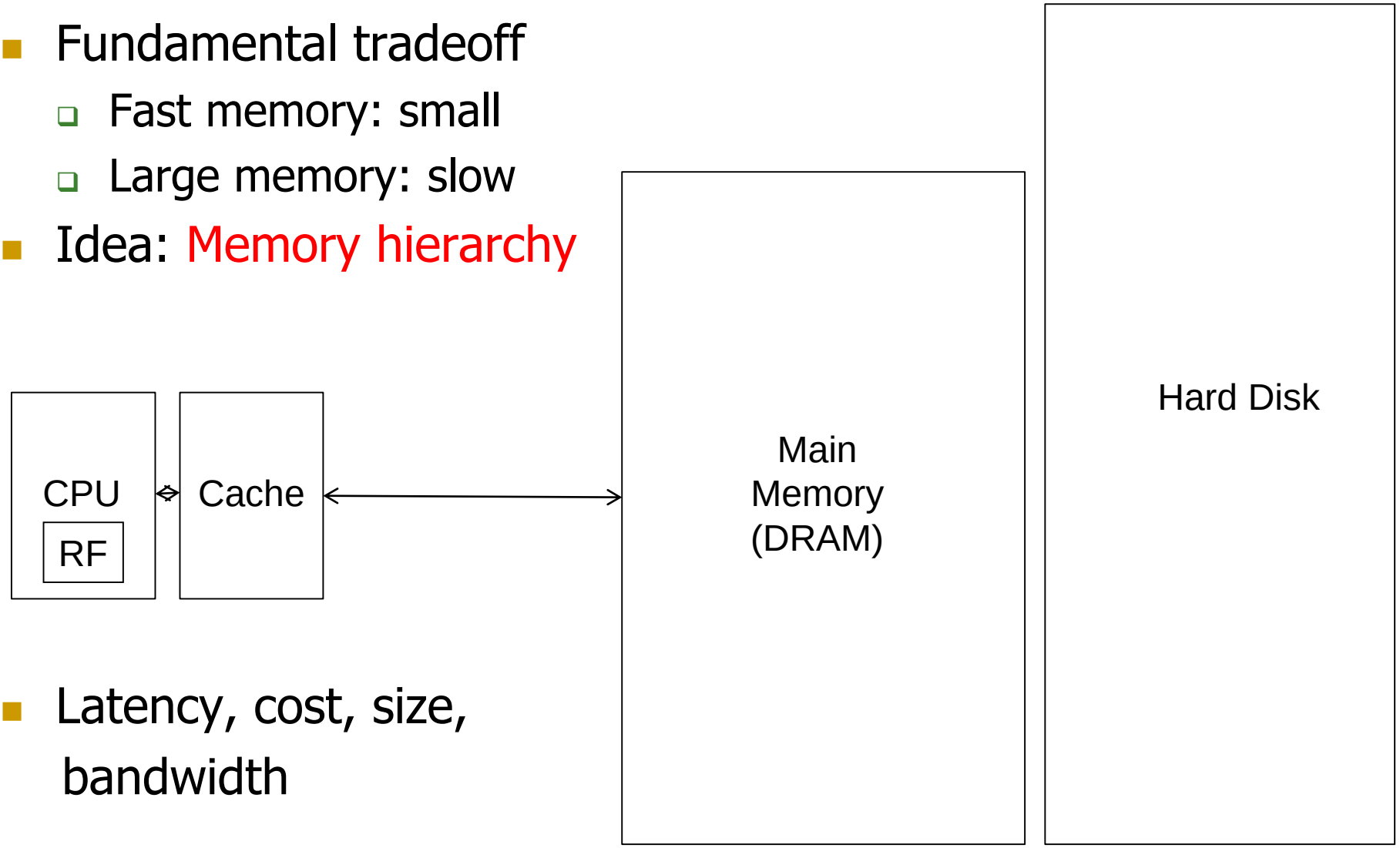
5. Precharge bit-lines
   - For next access

Why Memory Hierarchy?

- We want both fast and large
- But we cannot achieve both with a single level of memory
- Idea: Have multiple levels of storage (progressively bigger and slower as the levels are farther from the processor) and ensure most of the data the processor needs is kept in the fast(er) level(s)

Memory Hierarchy

- Fundamental tradeoff
  - Fast memory: small
  - Large memory: slow
- Idea: Memory hierarchy



- Latency, cost, size, bandwidth

Caching Basics: Exploit Temporal Locality

- Idea: Store recently accessed data in automatically managed fast memory (called cache)
- Anticipation: the data will be accessed again soon

- Temporal locality principle
  - Recently accessed data will be again accessed in the near future
  - This is what Maurice Wilkes had in mind:
    - Wilkes, "Slave Memories and Dynamic Storage Allocation," IEEE Trans. On Electronic Computers, 1965.
    - "The use is discussed of a fast core memory of, say 32000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast memory than that of the slow memory."

Caching Basics: Exploit Spatial Locality

- Idea: Store addresses adjacent to the recently accessed one in automatically managed fast memory
  - Logically divide memory into equal size blocks
  - Fetch to cache the accessed block in its entirety
- Anticipation: nearby data will be accessed soon

- Spatial locality principle
  - Nearby data in memory will be accessed in the near future
    - E.g., sequential instruction access, array traversal
  - This is what IBM 360/85 implemented
    - 16 Kbyte cache with 64 byte blocks
    - Liptay, "Structural aspects of the System/360 Model 85 II: the cache," IBM Systems Journal, 1968.

A Note on Manual vs. Automatic Management

- Manual: Programmer manages data movement across levels

  -- too painful for programmers on substantial programs

  - "core" vs "drum" memory in the 50's
  - still done in some embedded processors (on-chip scratch pad SRAM in lieu of a cache)

- Automatic: Hardware manages data movement across levels, transparently to the programmer

  ++ programmer's life is easier

  - simple heuristic: keep most recently used items in cache
  - the average programmer doesn't need to know about it
    - You don't need to know how big the cache is and how it works to write a "correct" program! (What if you want a "fast" program?)

Automatic Management in Memory Hierarchy

- Wilkes, "Slave Memories and Dynamic Storage Allocation," IEEE Trans. On Electronic Computers, 1965.

Slave Memories and Dynamic Storage Allocation

M. V. WILKES

SUMMARY

The use is discussed of a fast core memory of, say, 32 000 words as a slave to a slower core memory of, say, one million words in such a way that in practical cases the effective access time is nearer that of the fast memory than that of the slow memory.

- "By a slave memory I mean one which automatically accumulates to itself words that come from a slower main memory, and keeps them available for subsequent use without it being necessary for the penalty of main memory access to be incurred again."

Historical Aside: Other Cache Papers

- Fotheringham, "Dynamic Storage Allocation in the Atlas Computer, Including an Automatic Use of a Backing Store," CACM 1961.
  - http://dl.acm.org/citation.cfm?id=366800

- Bloom, Cohen, Porter, "Considerations in the Design of a Computer with High Logic-to-Memory Speed Ratio," AIEE Gigacycle Computing Systems Winter Meeting, Jan. 1962.

Cache in 1962 (Bloom, Cohen, Porter)

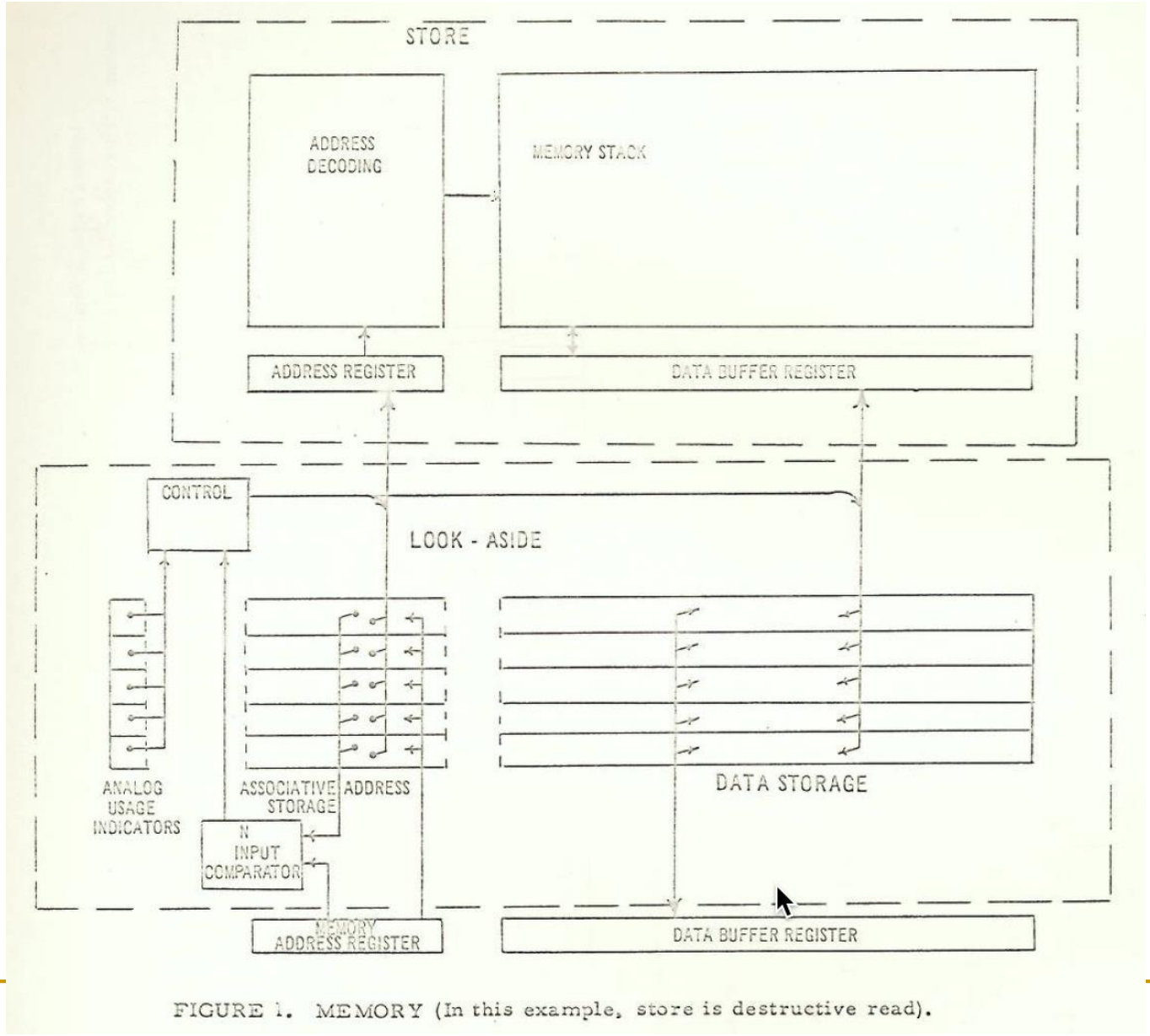
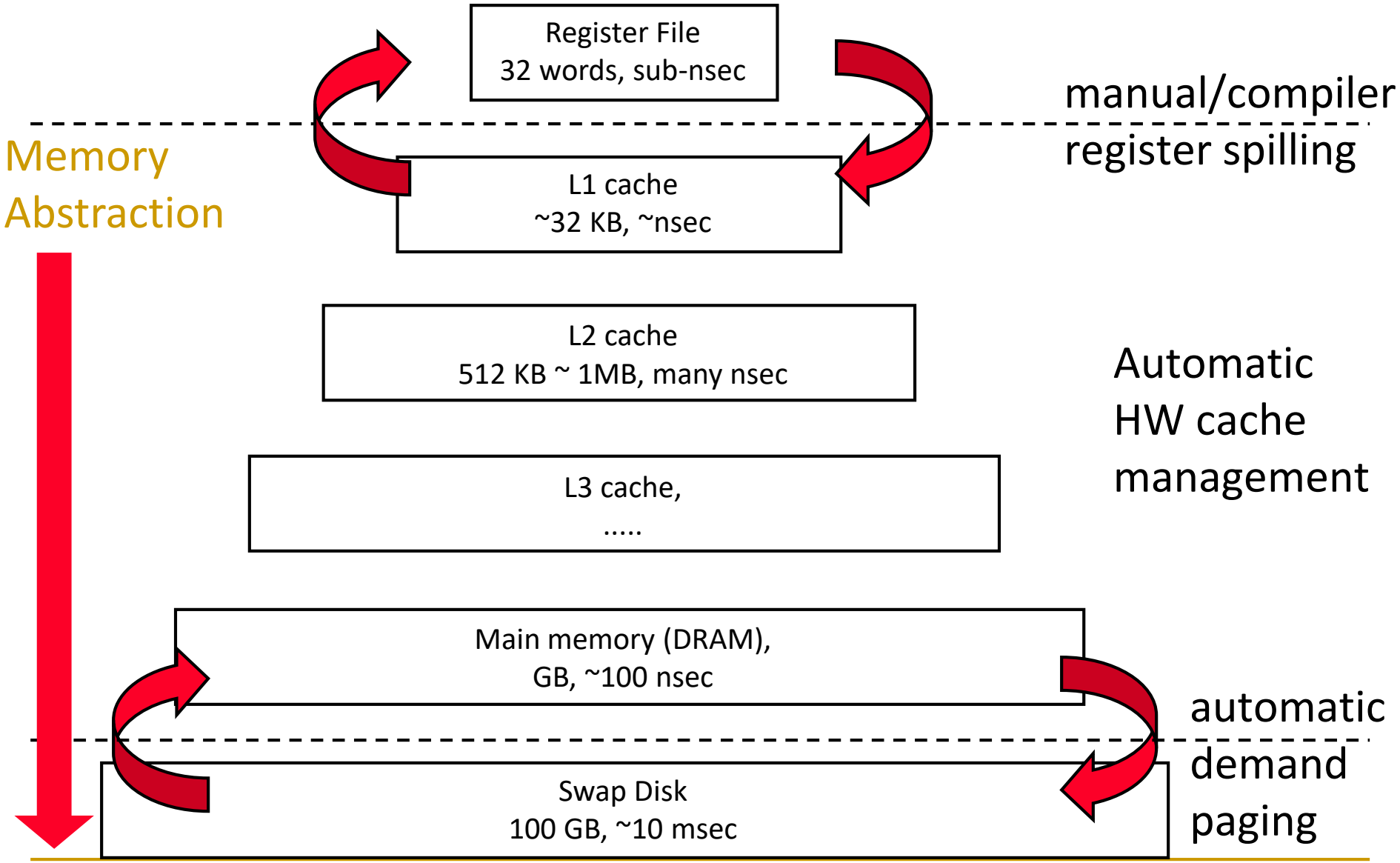


FIGURE 1. MEMORY (In this example, store is destructive read).

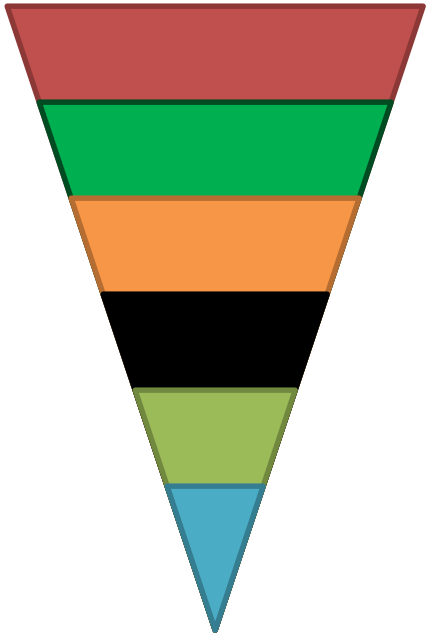
A Modern Memory Hierarchy



The DRAM Subsystem

DRAM Subsystem Organization

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column

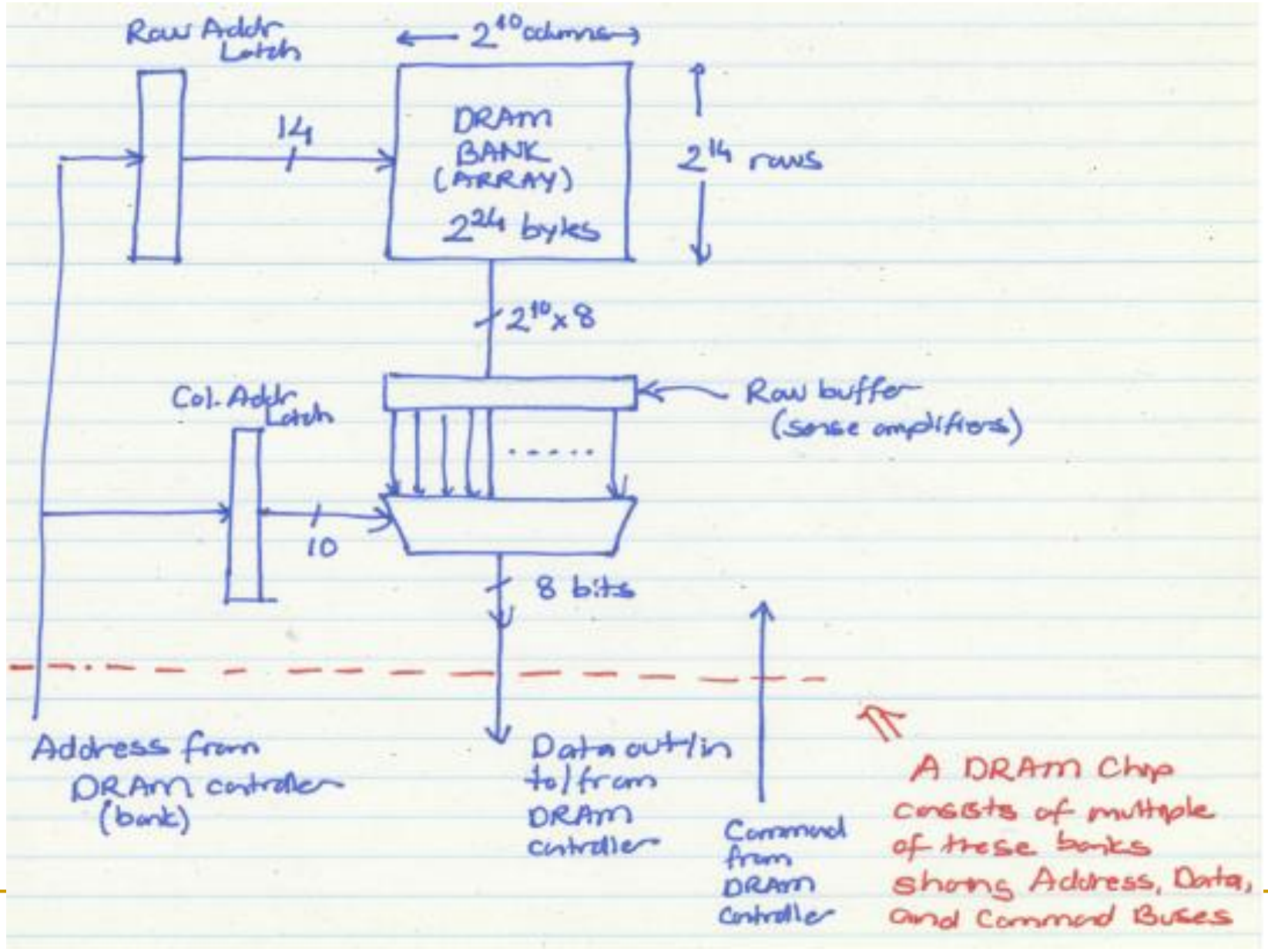


Page Mode DRAM

- A DRAM bank is a 2D array of cells: rows x columns
- A "DRAM row" is also called a "DRAM page"
- "Sense amplifiers" also called "row buffer"

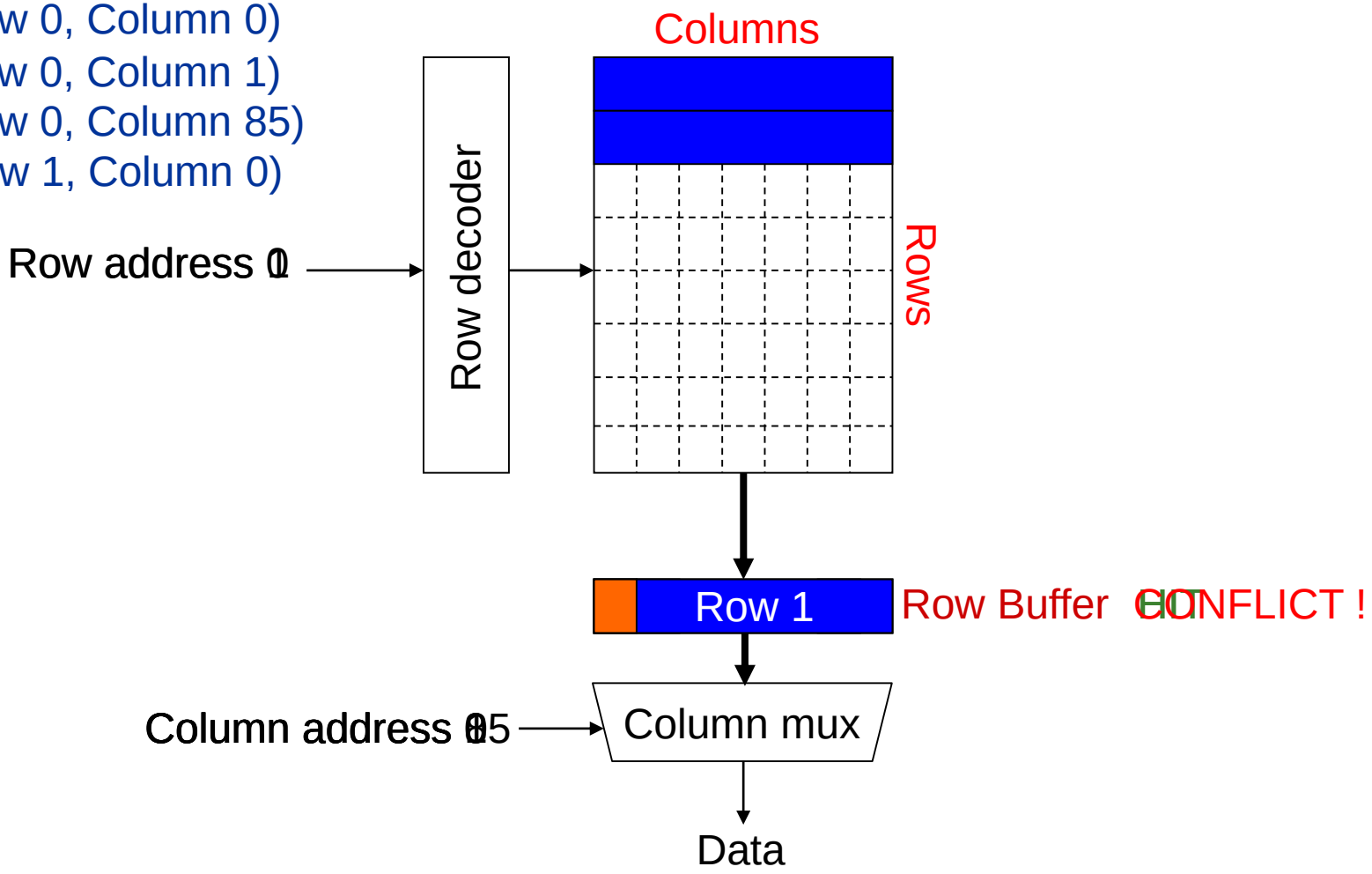
- Each address is a <row,column> pair
- Access to a "closed row"
  - Activate command opens row (placed into row buffer)
  - Read/write command reads/writes column in the row buffer
  - Precharge command closes the row and prepares the bank for next access
- Access to an "open row"
  - No need for activate command

The DRAM Bank Structure



DRAM Bank Operation

Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

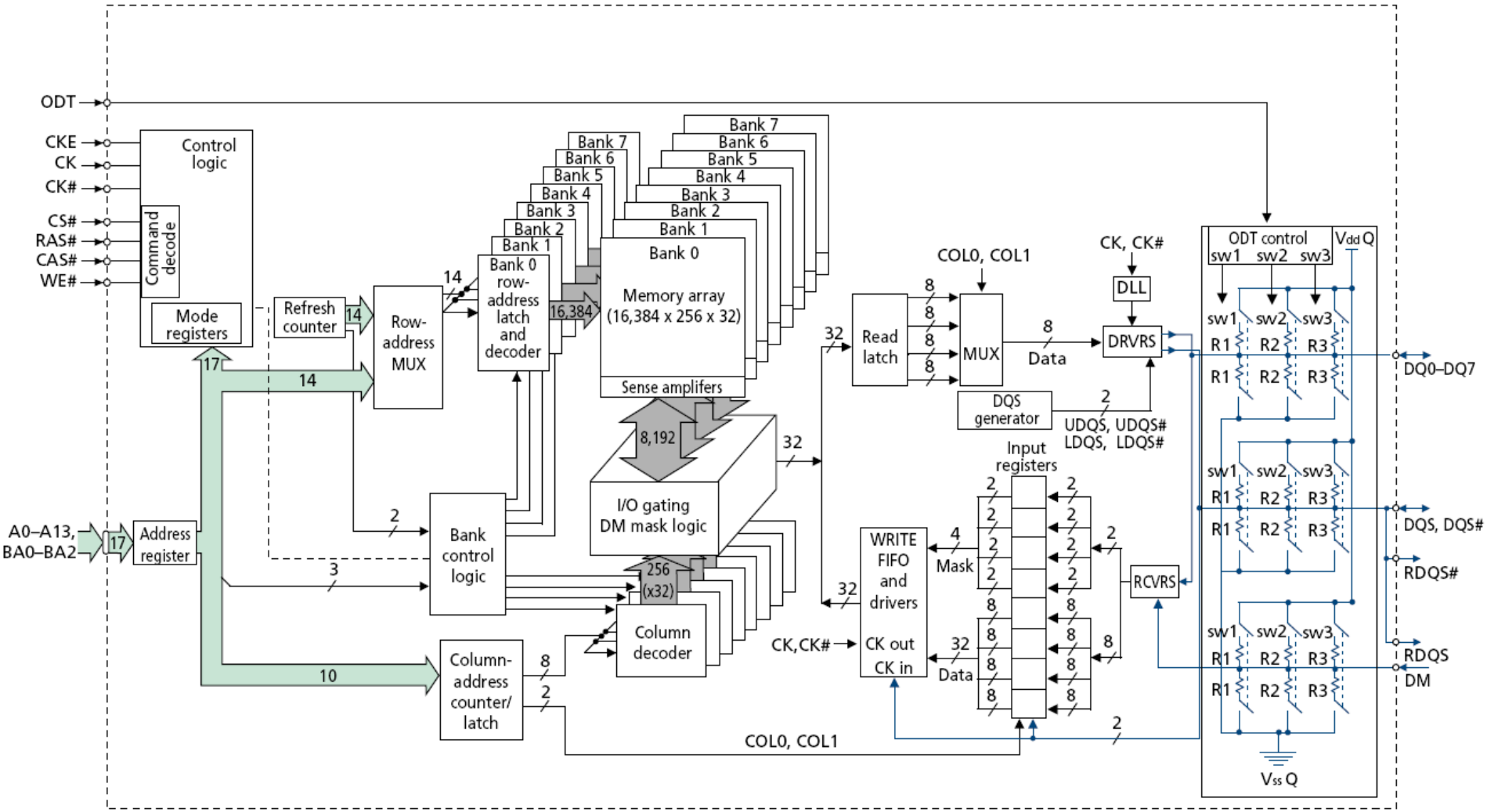


The DRAM Chip

- Consists of multiple banks (8 is a common number today)
- Banks share command/address/data buses
- The chip itself has a narrow interface (4-16 bits per read)

- Changing the number of banks, size of the interface (pins), whether or not command/address/data buses are shared has significant impact on DRAM system cost

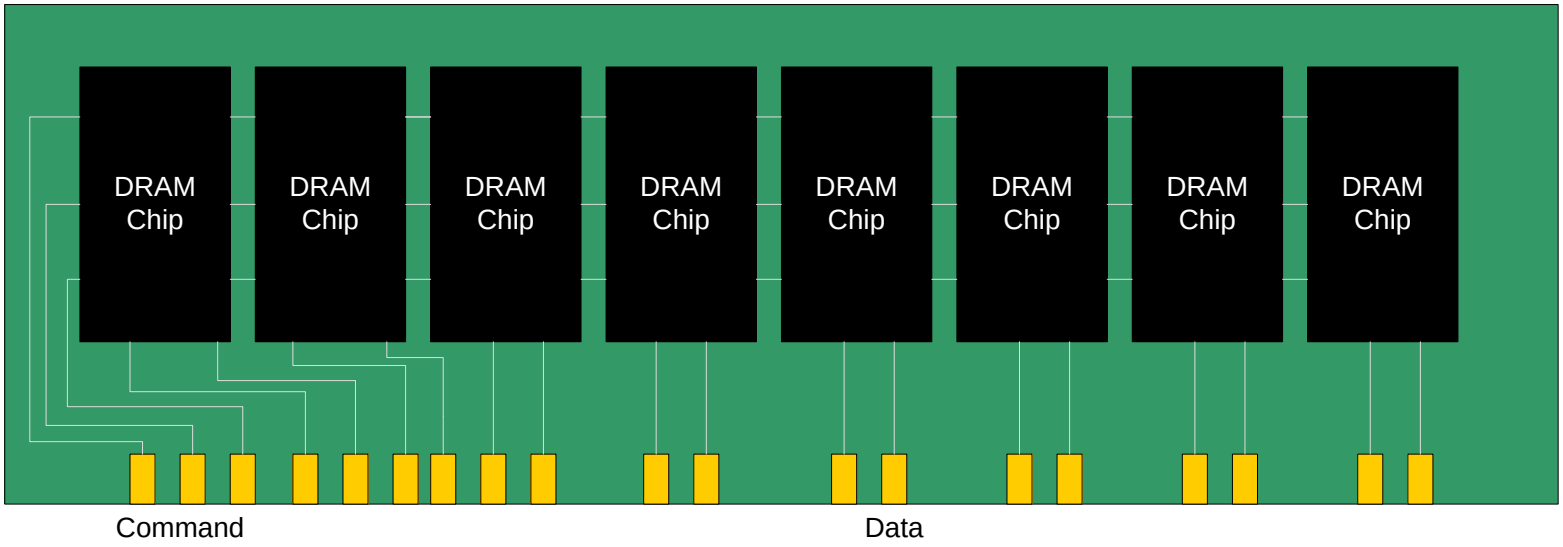
128M x 8-bit DRAM Chip



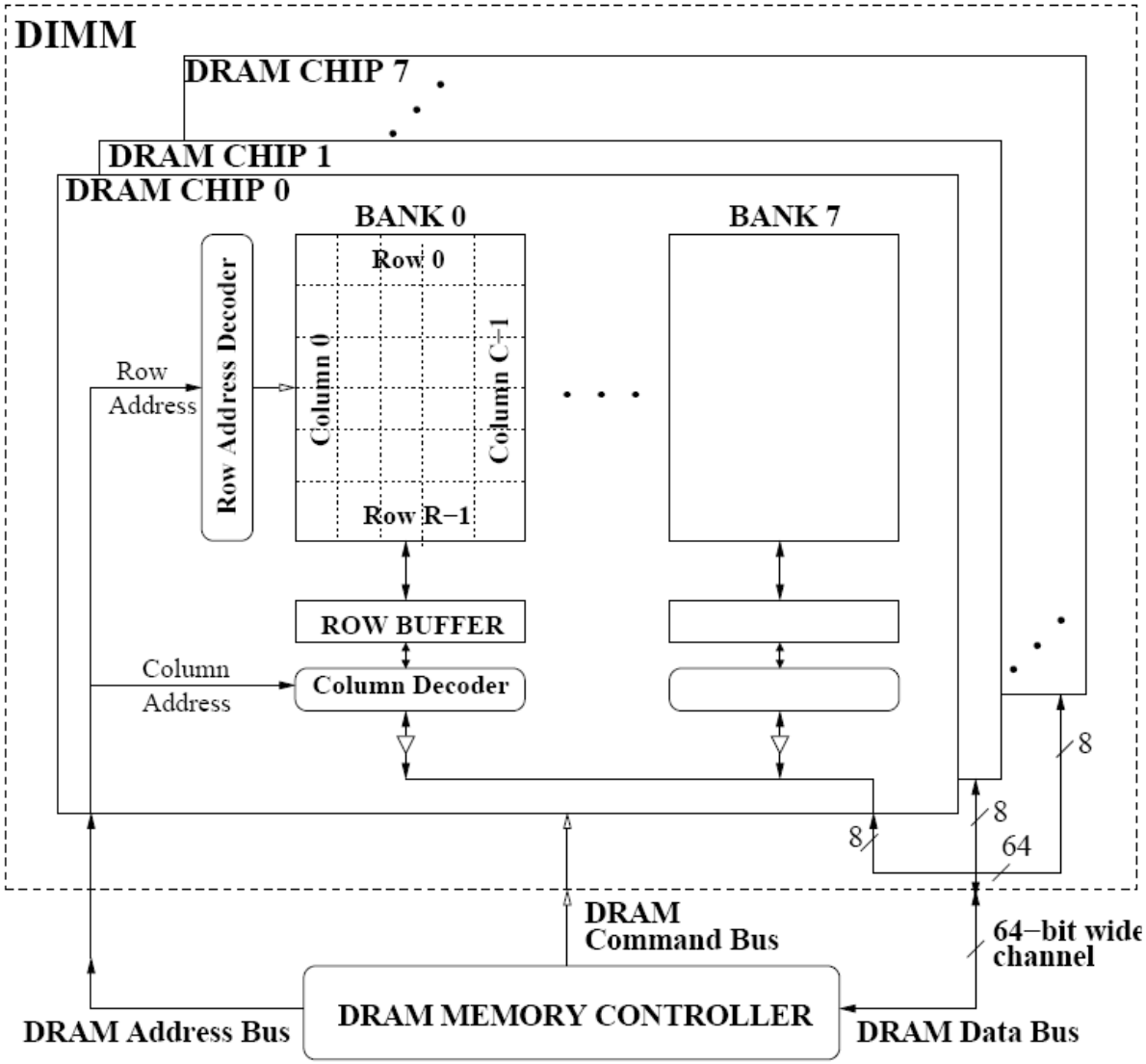
DRAM Rank and Module

- Rank: Multiple chips operated together to form a wide interface
- All chips comprising a rank are controlled at the same time
  - Respond to a single command
  - Share address and command buses, but provide different data
- A DRAM module consists of one or more ranks
  - E.g., DIMM (dual inline memory module)
  - This is what you plug into your motherboard
- If we have chips with 8-bit interface, to read 8 bytes in a single access, use 8 chips in a DIMM

A 64-bit Wide DIMM (One Rank)

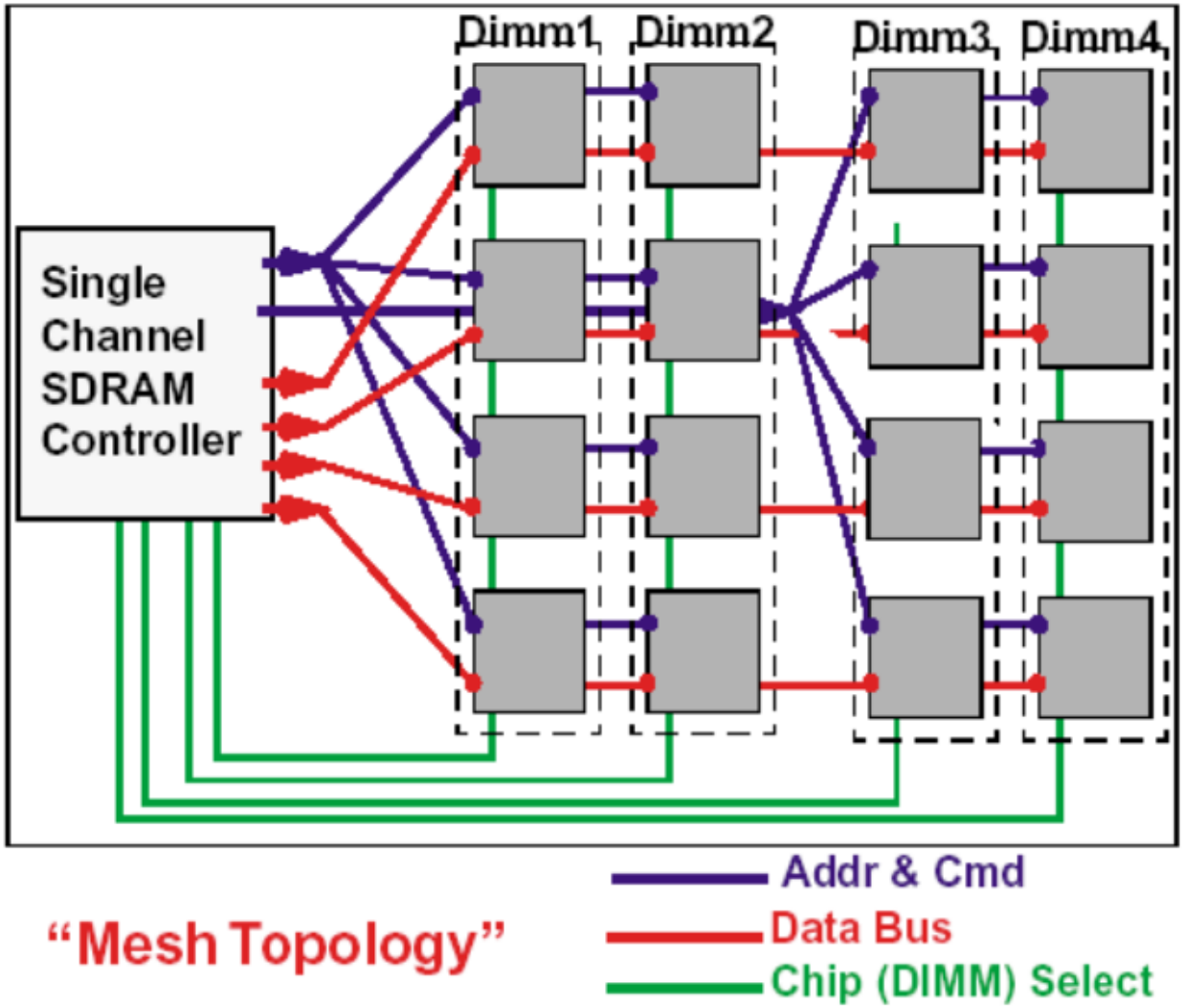


A 64-bit Wide DIMM (One Rank)



- Advantages:
  - Acts like a high-capacity DRAM chip with a wide interface
  - Flexibility: memory controller does not need to deal with individual chips
- Disadvantages:
  - Granularity: Accesses cannot be smaller than the interface width

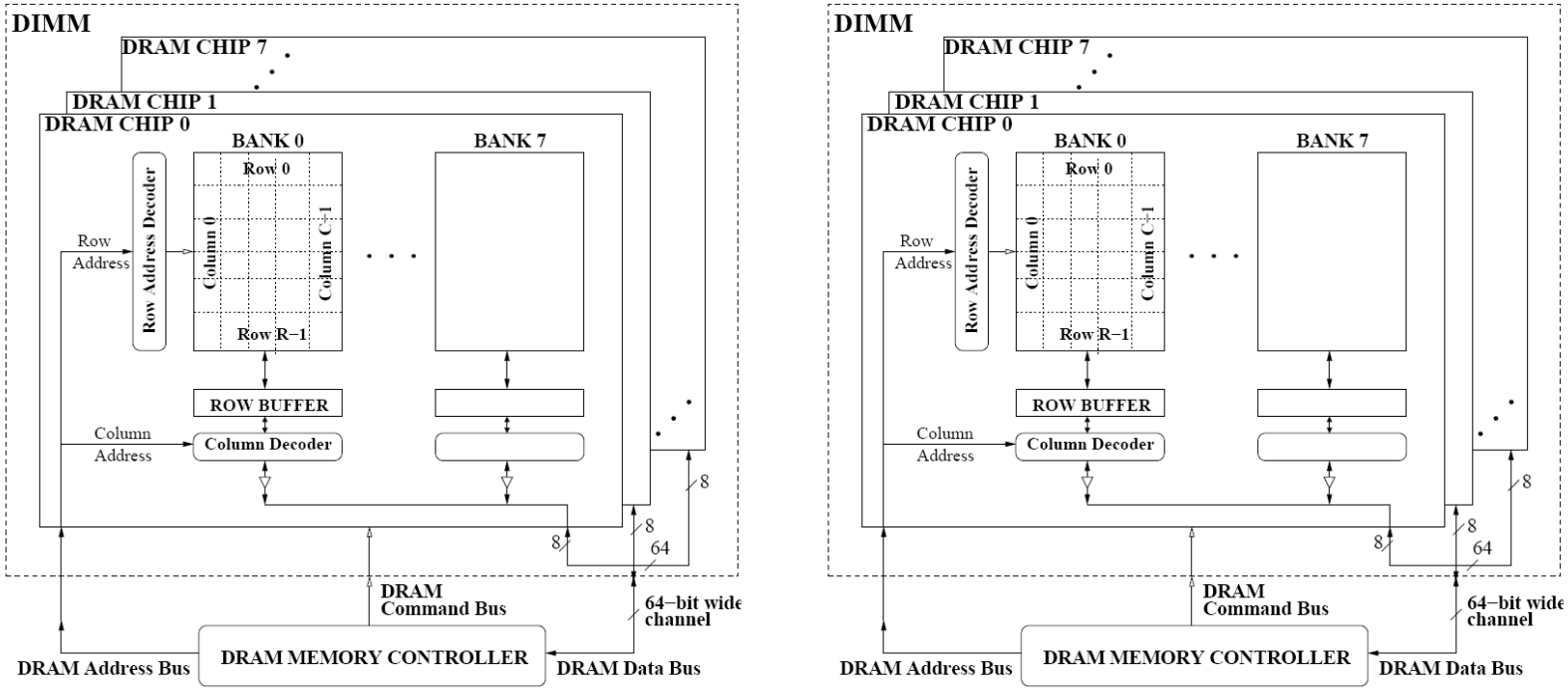
Multiple DIMMs



- Advantages:
  - Enables even higher capacity
- Disadvantages:
  - Interconnect complexity and energy consumption can be high

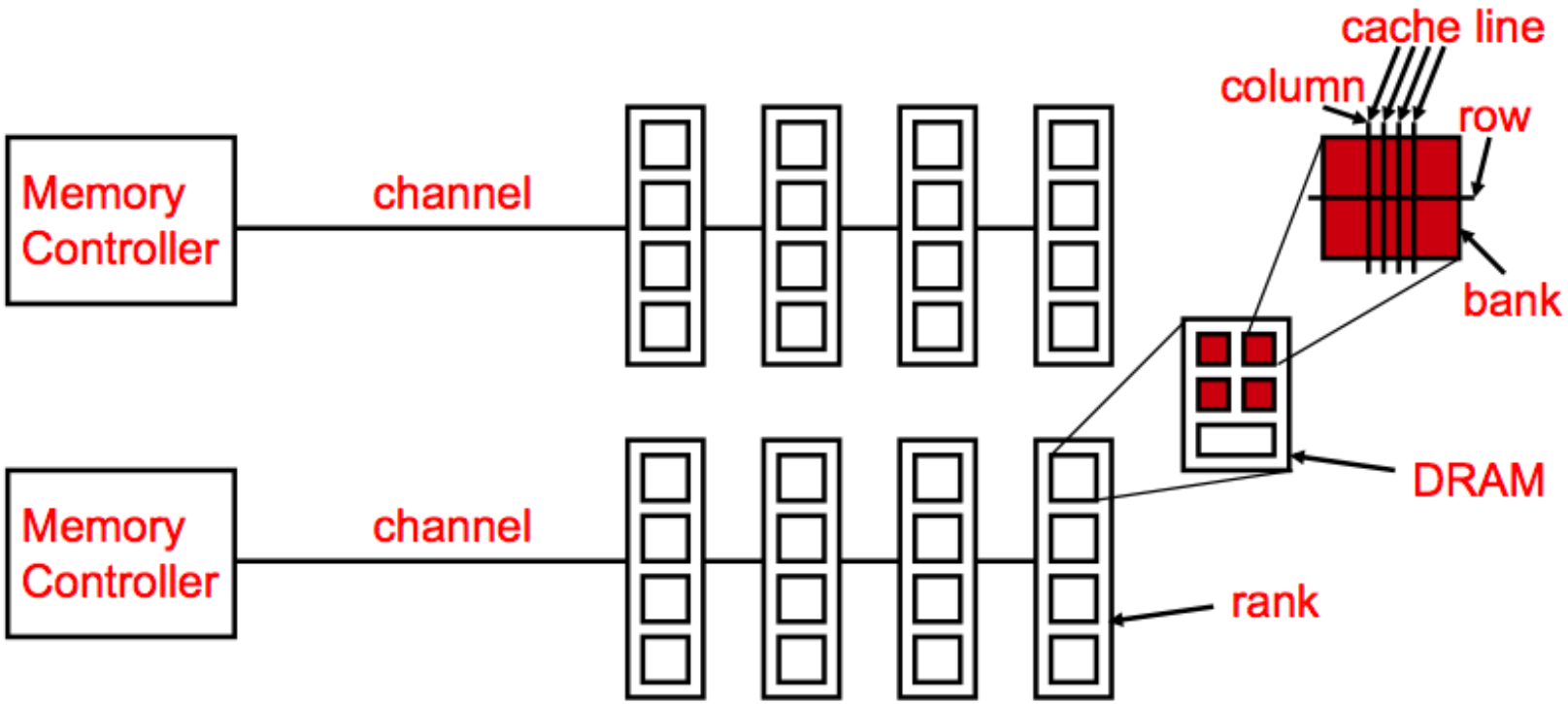
  → Scalability is limited by this

DRAM Channels

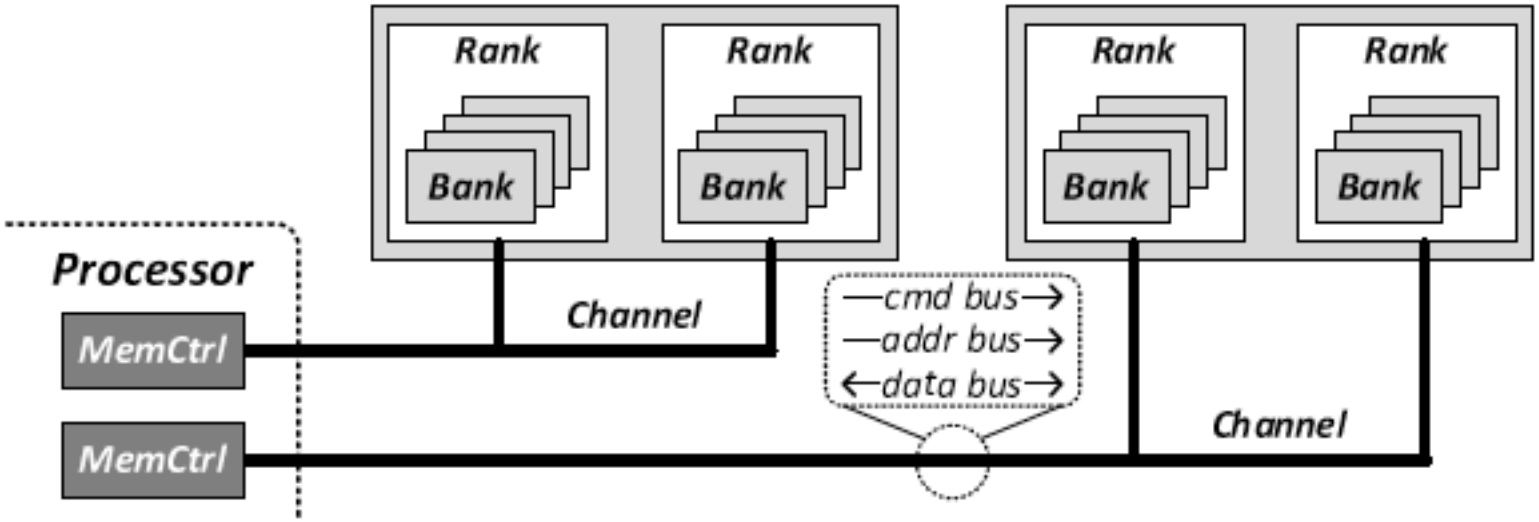


- 2 Independent Channels: 2 Memory Controllers (Above)
- 2 Dependent/Lockstep Channels: 1 Memory Controller with wide interface (Not shown above)

Generalized Memory Structure



Generalized Memory Structure



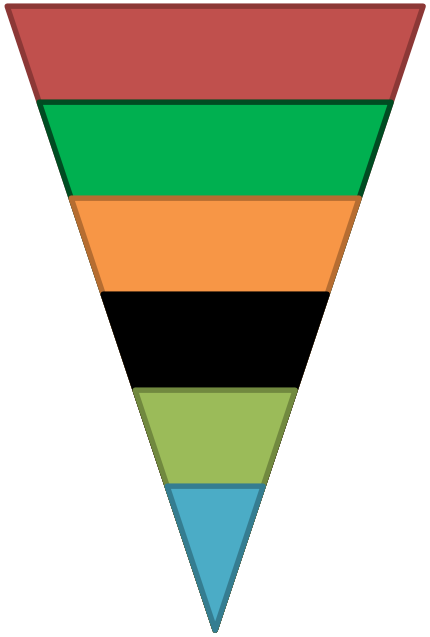
Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

Lee+, "Decoupled Direct Memory Access," PACT 2015.

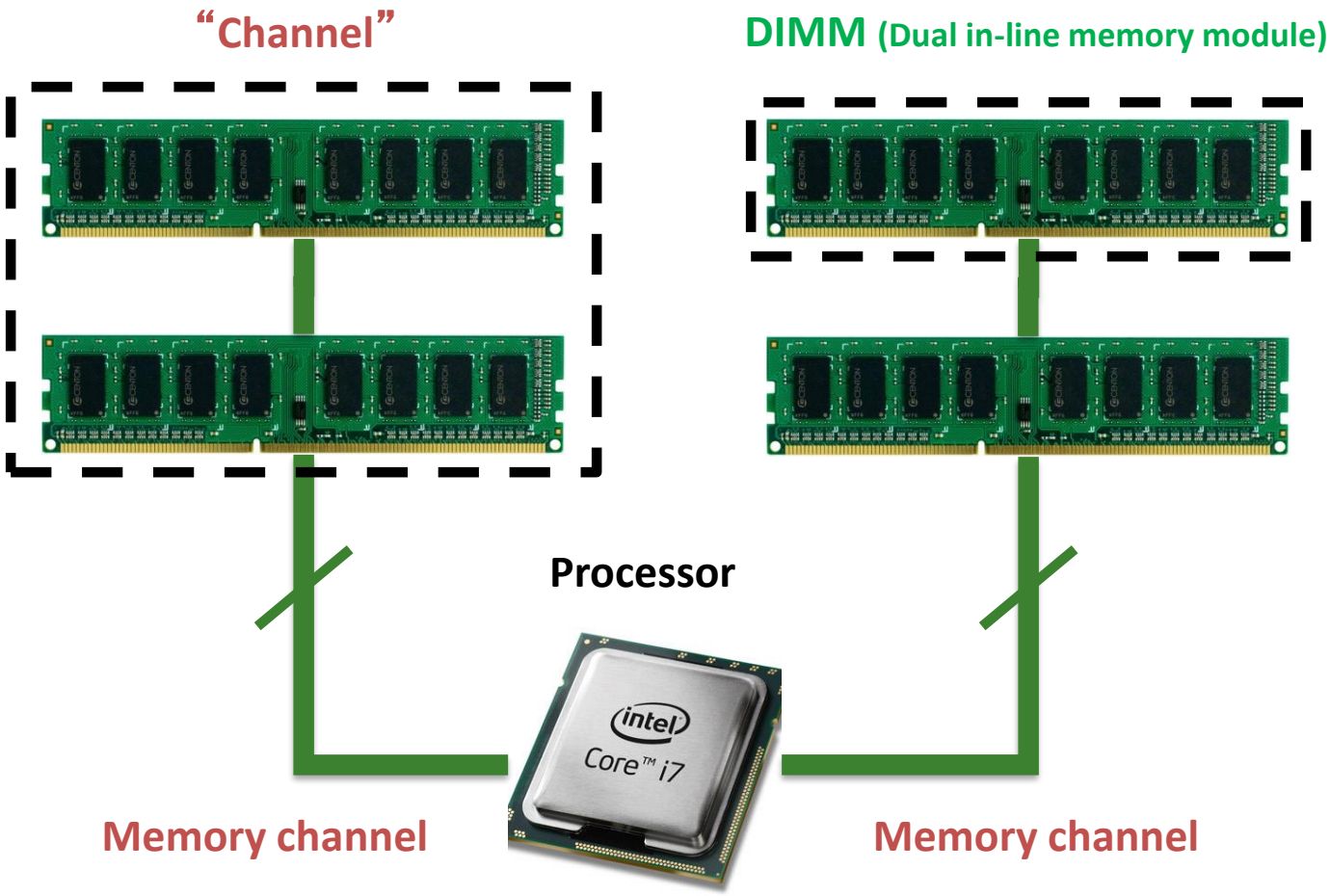
The DRAM Subsystem
The Top Down View

DRAM Subsystem Organization

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column



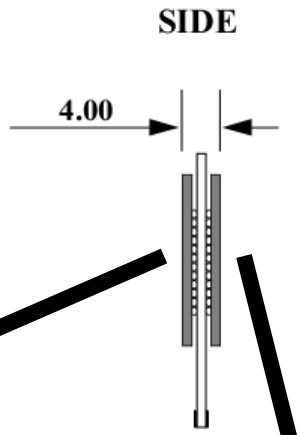
The DRAM subsystem



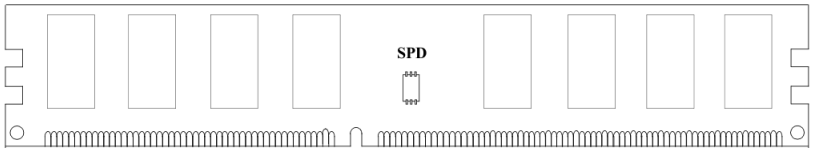
Breaking down a DIMM

DIMM (Dual in-line memory module)

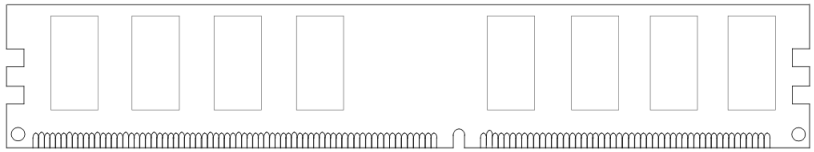
Side view



Front of DIMM



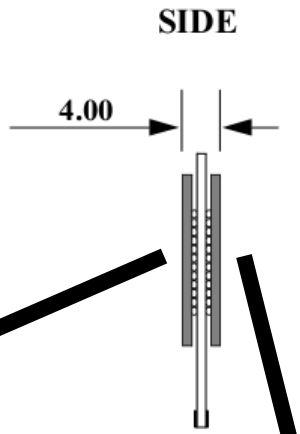
Back of DIMM



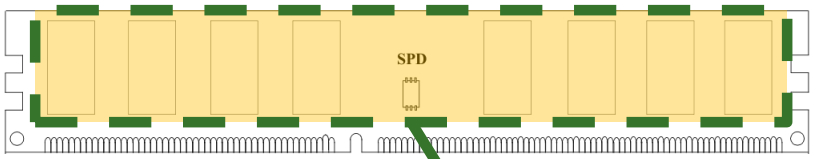
Breaking down a DIMM

DIMM (Dual in-line memory module)

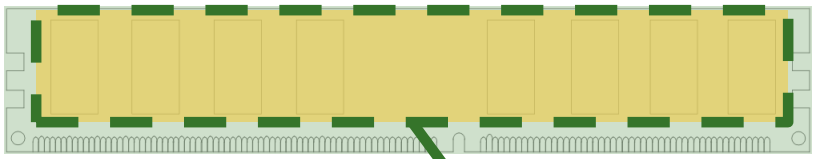
Side view



Front of DIMM



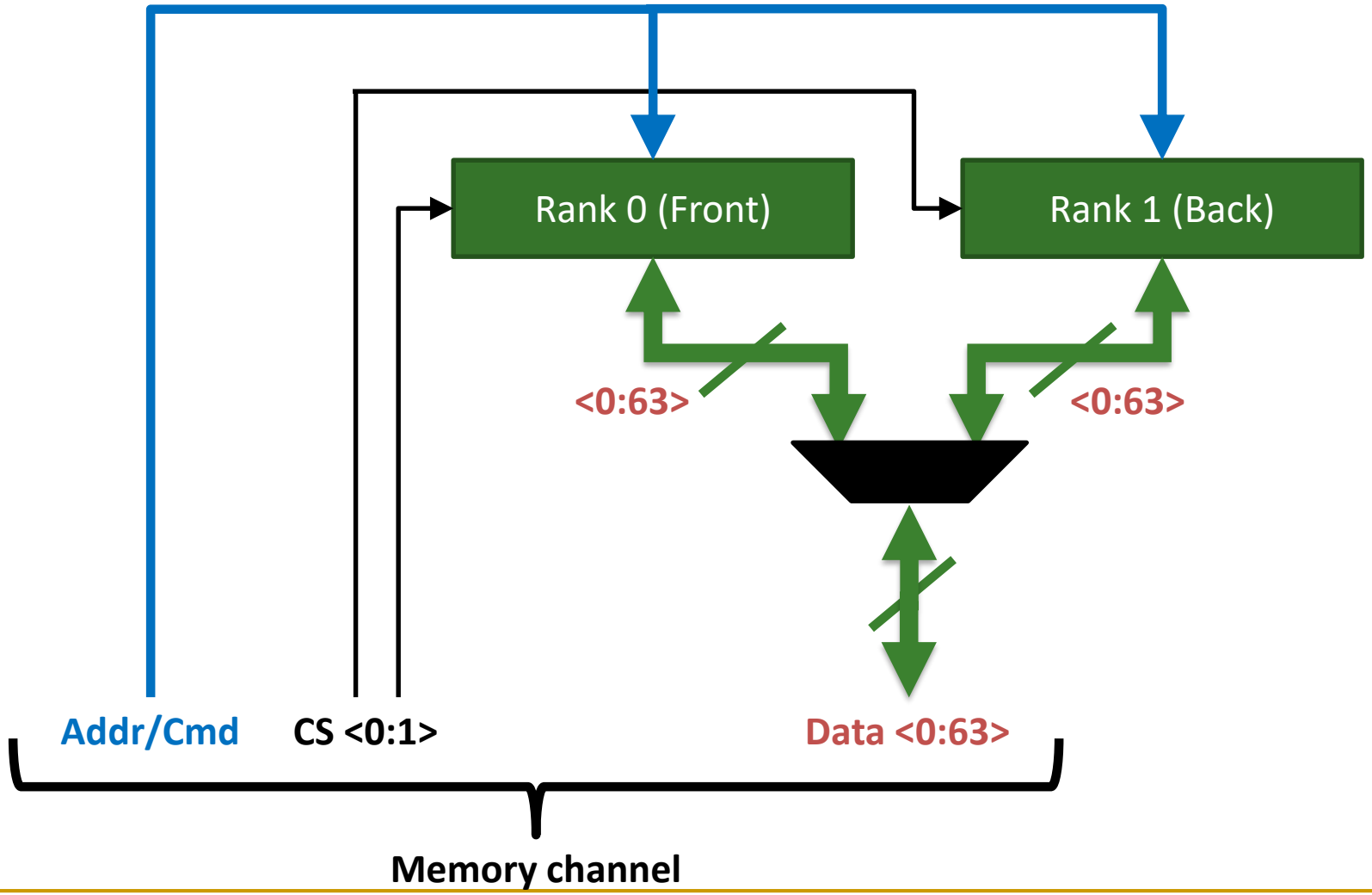
Back of DIMM



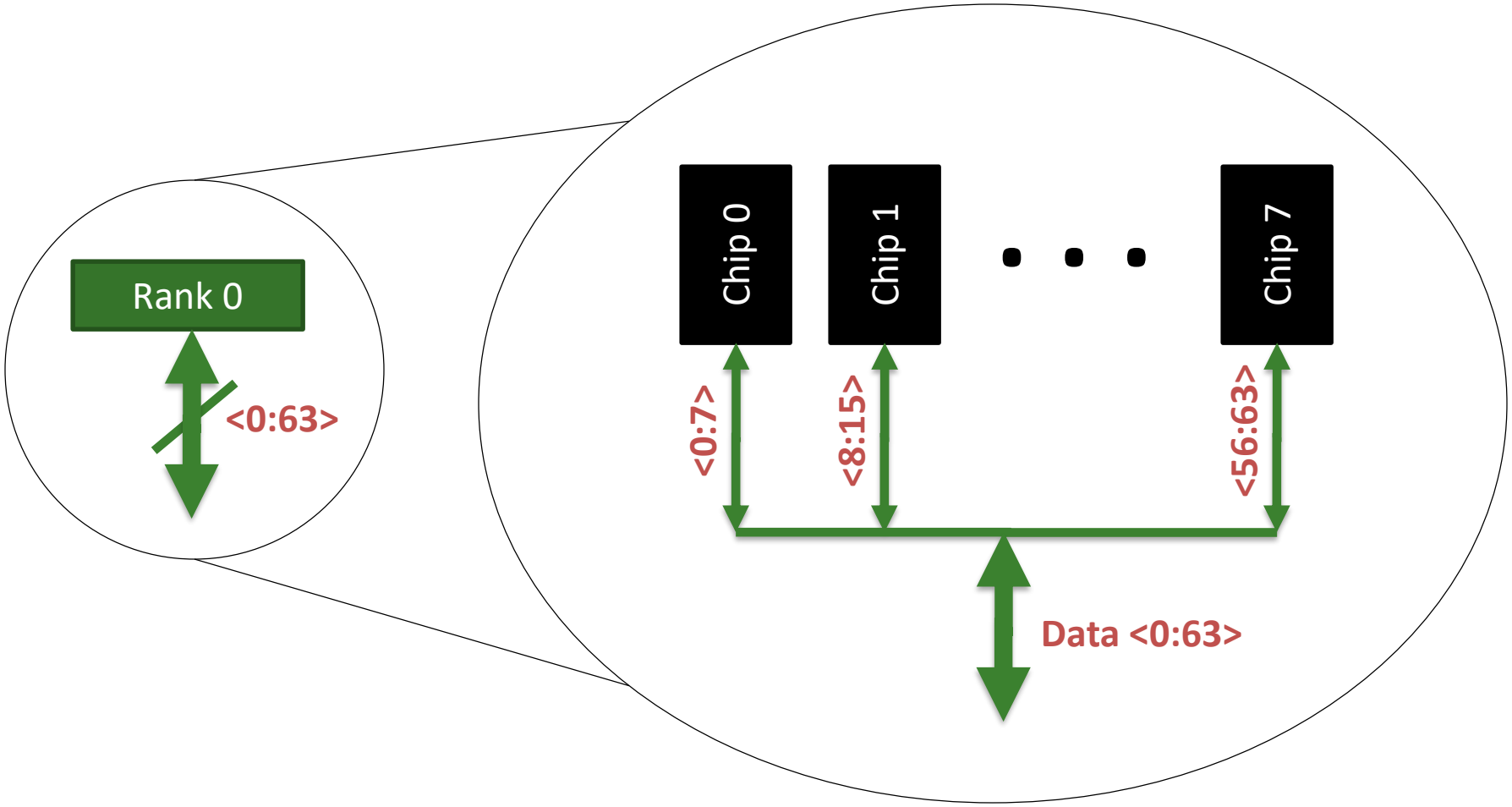
Rank 0: collection of 8 chips

Rank 1

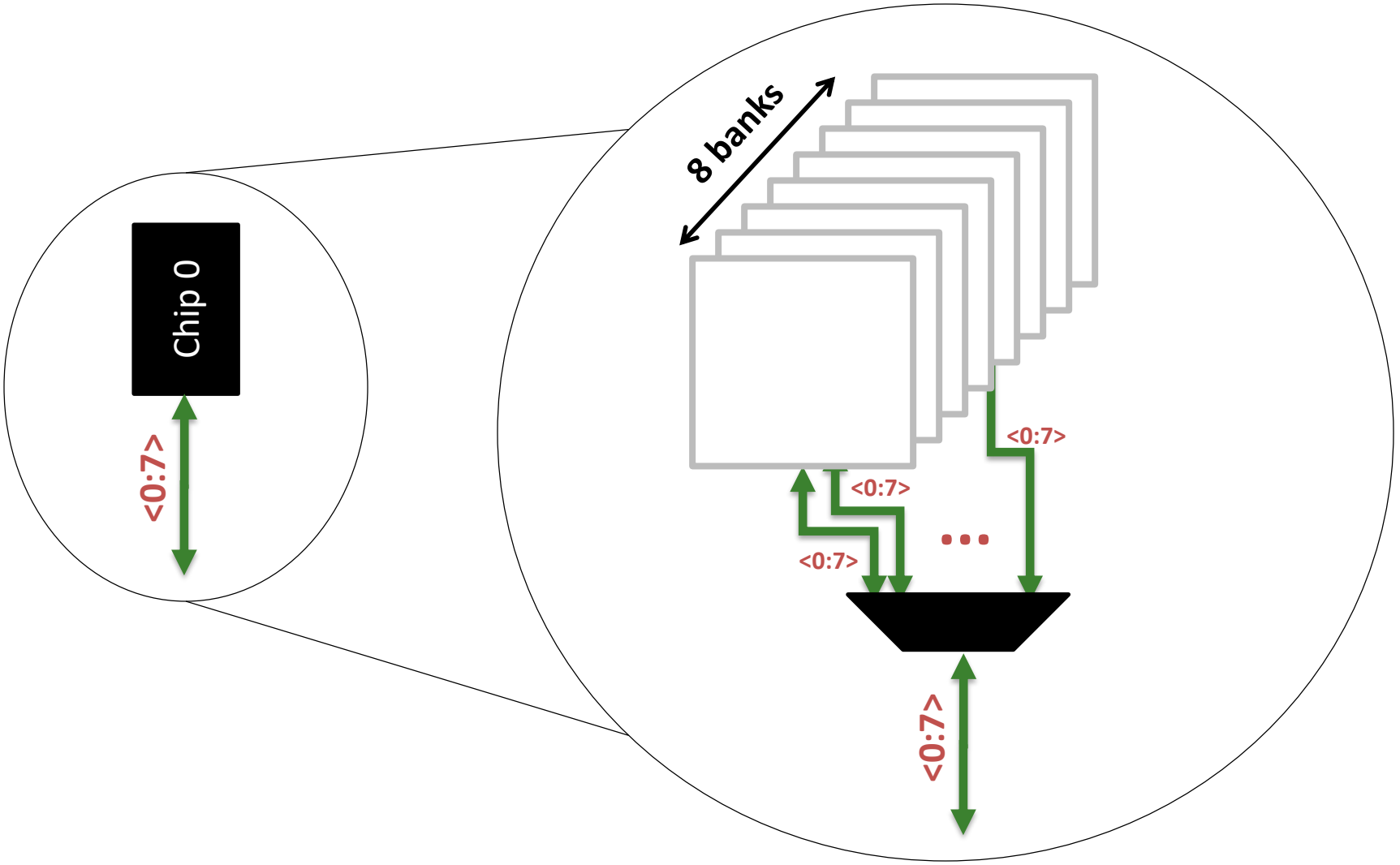
Rank


Rank 0 (Front)
Rank 1 (Back)
<0:63>
<0:63>
Addr/Cmd
CS <0:1>
Data <0:63>
Memory channel

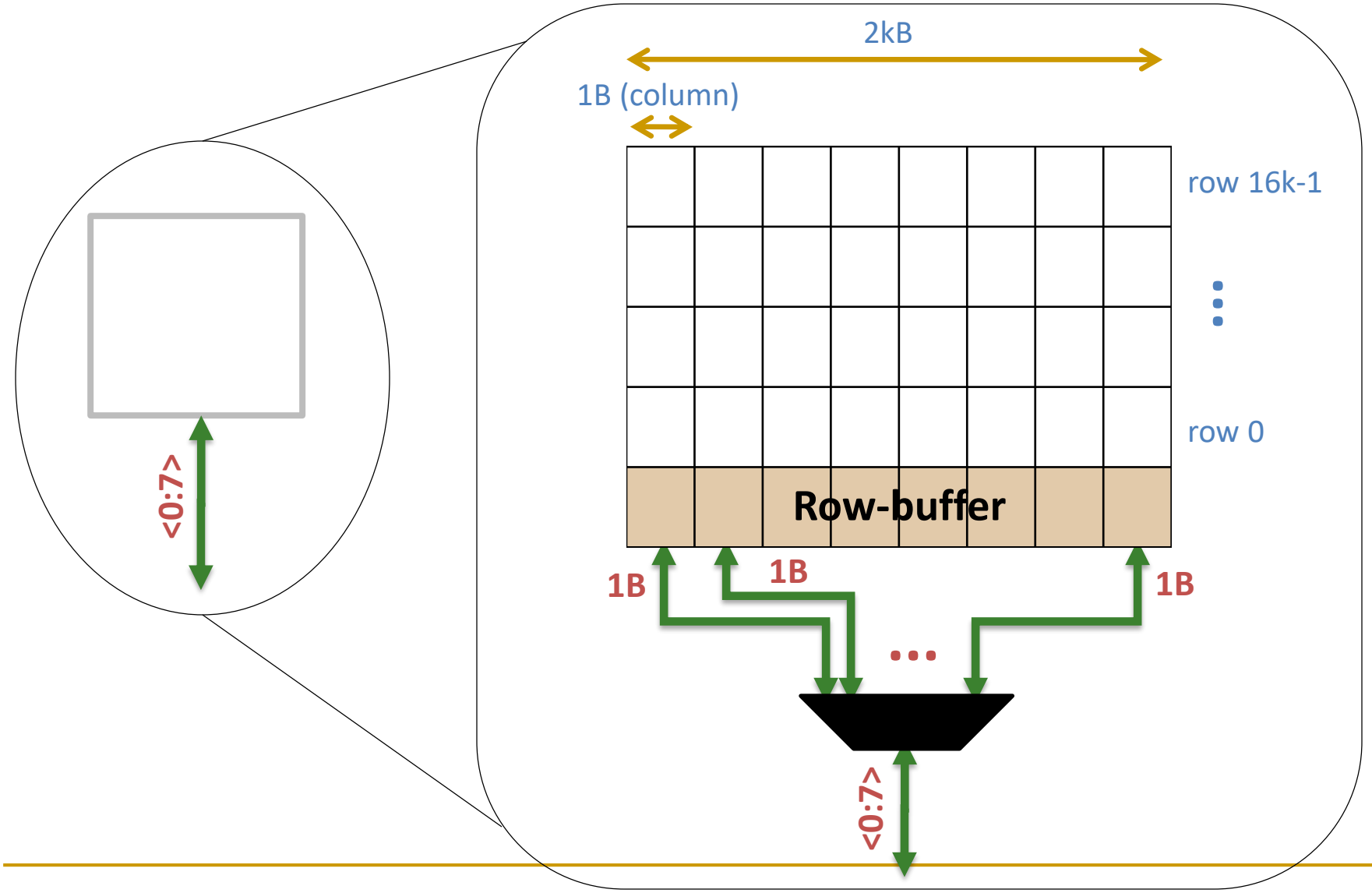
Breaking down a Rank



Breaking down a Chip

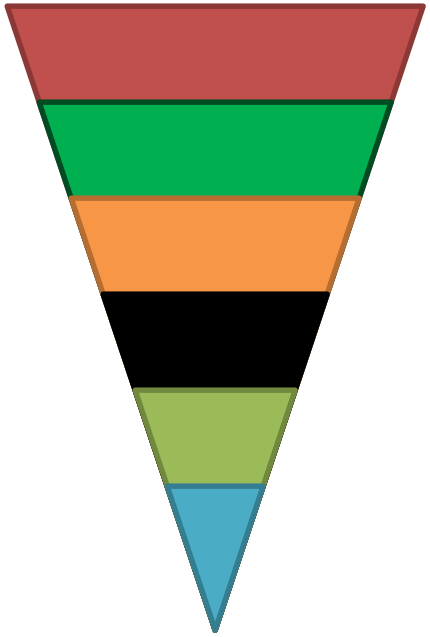


Breaking down a Bank

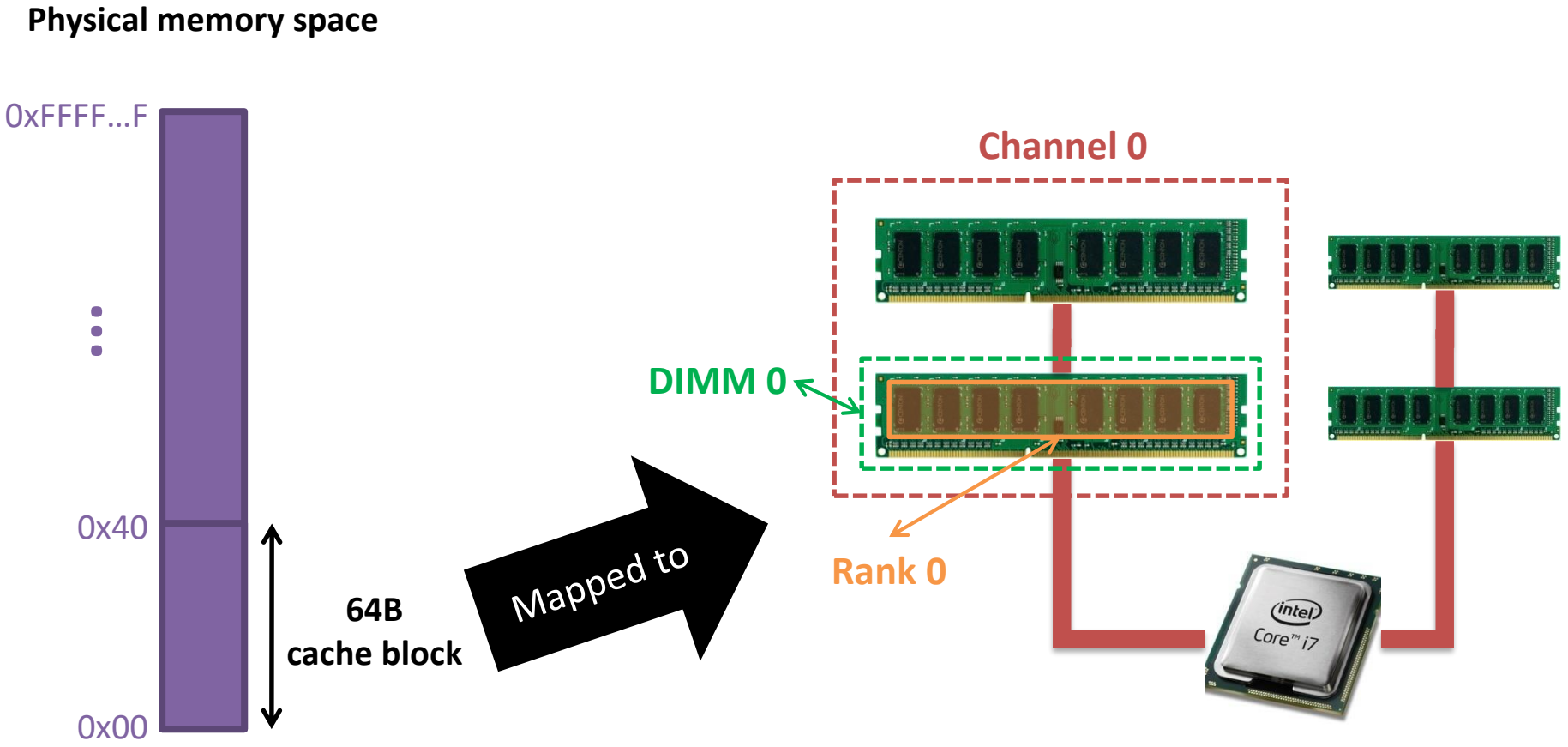


DRAM Subsystem Organization

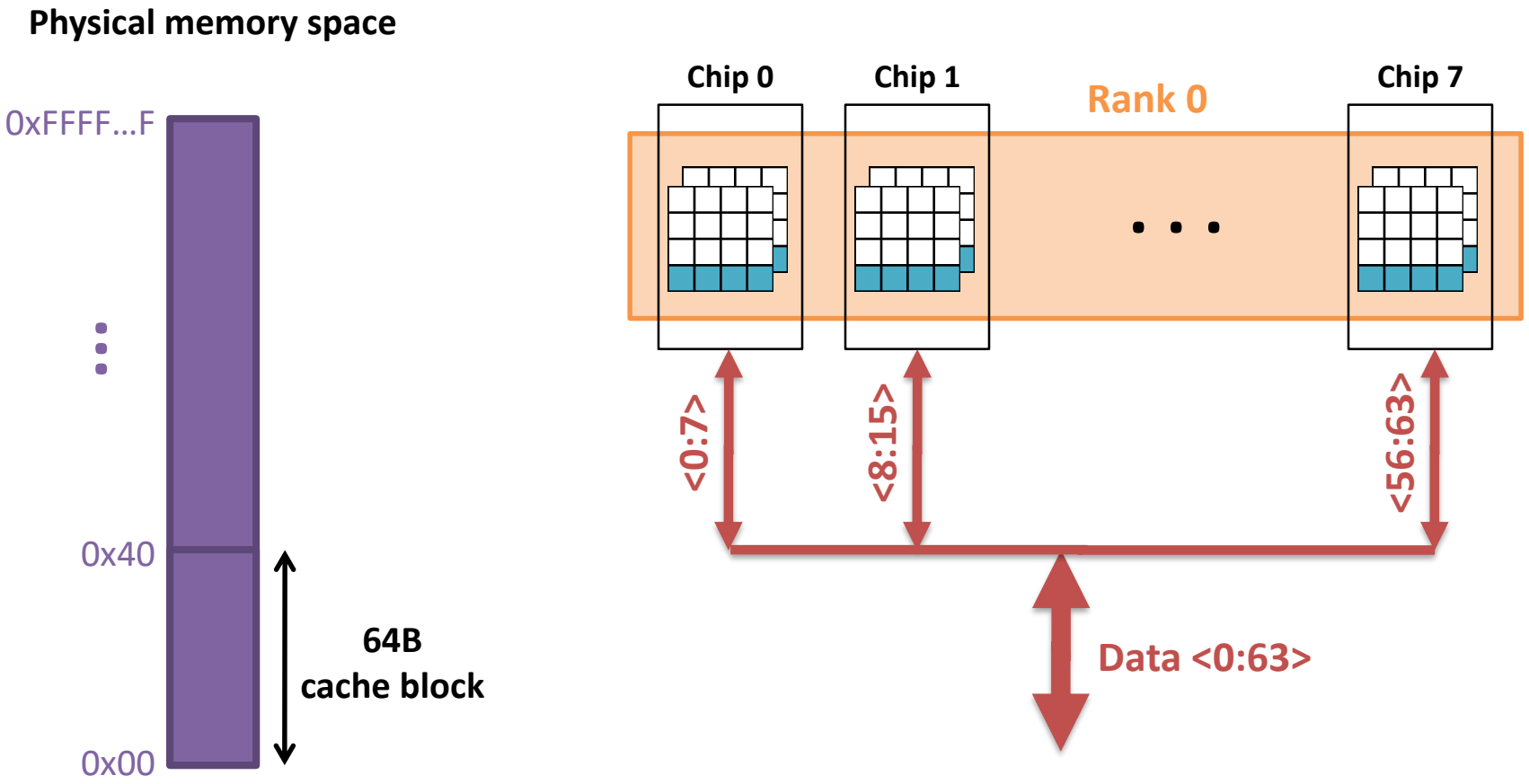
- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column



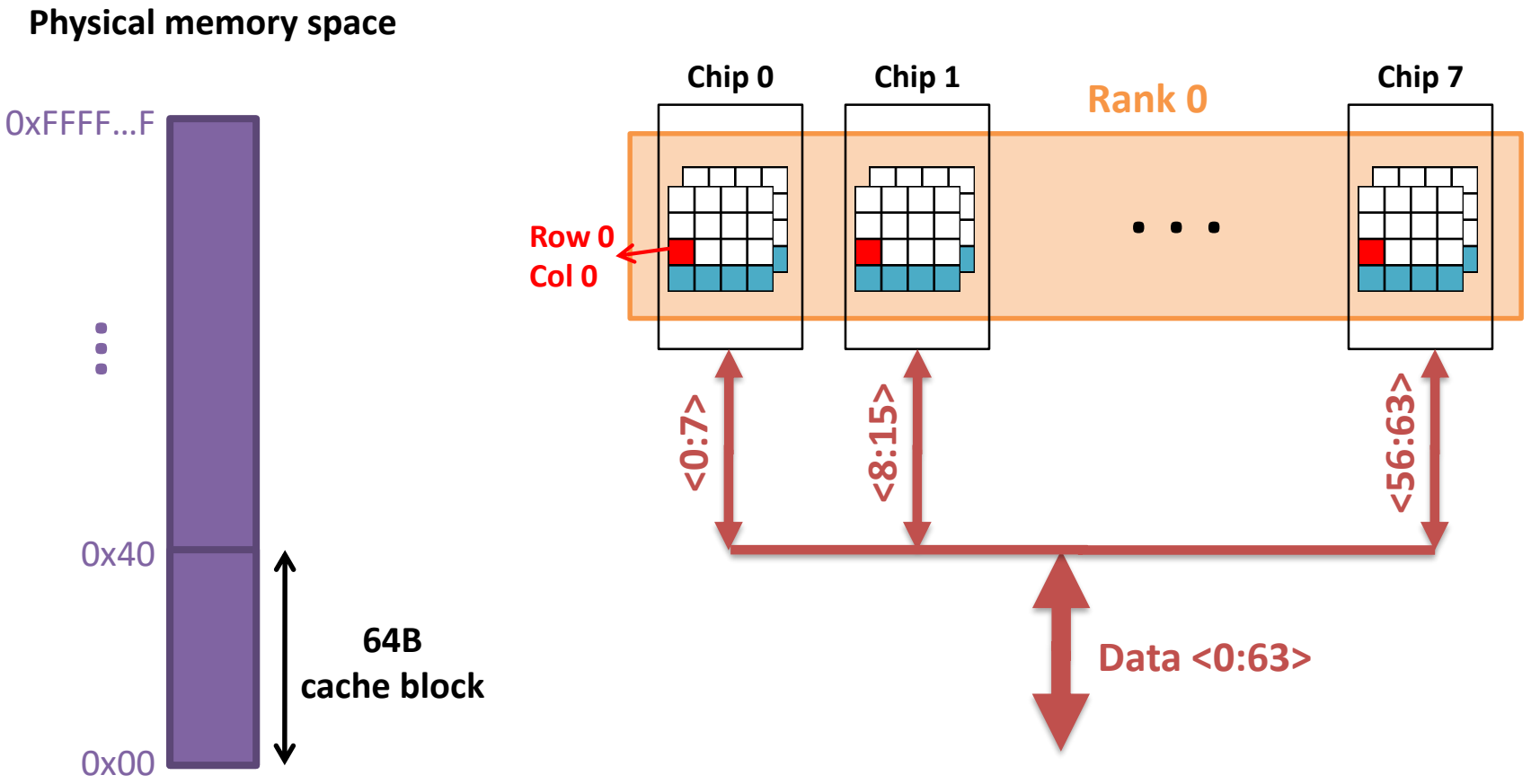
Example: Transferring a cache block



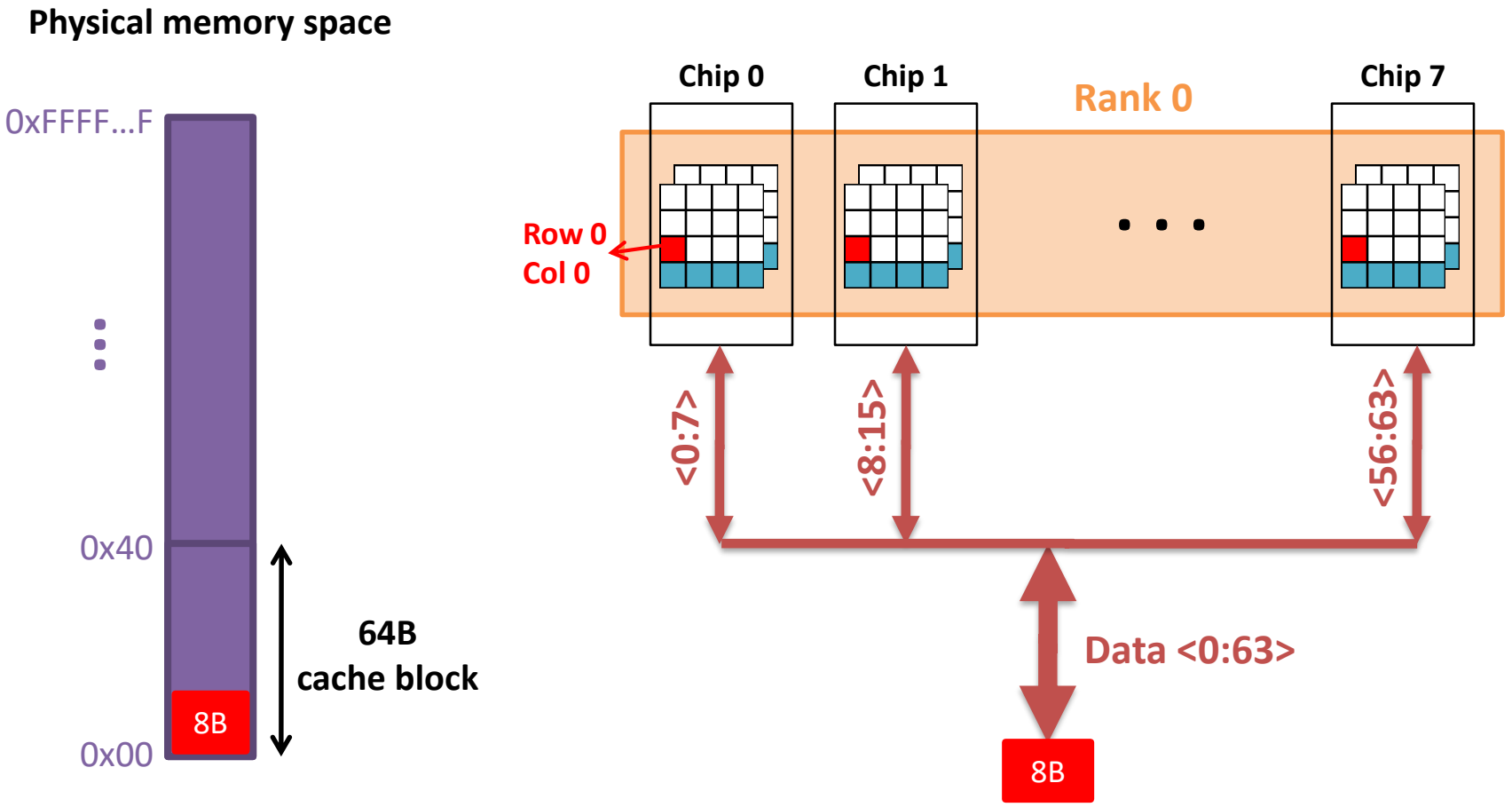
Example: Transferring a cache block



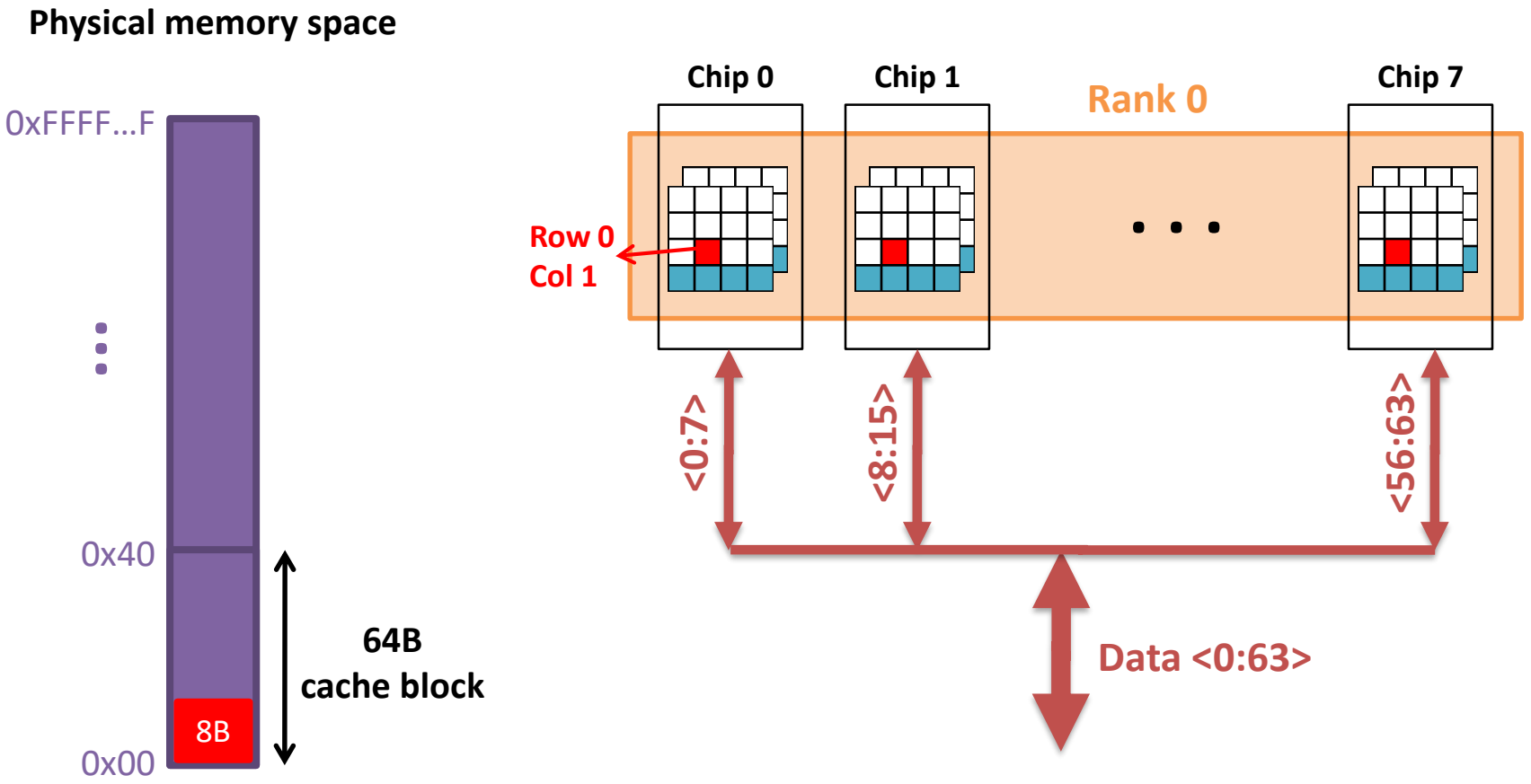
Example: Transferring a cache block



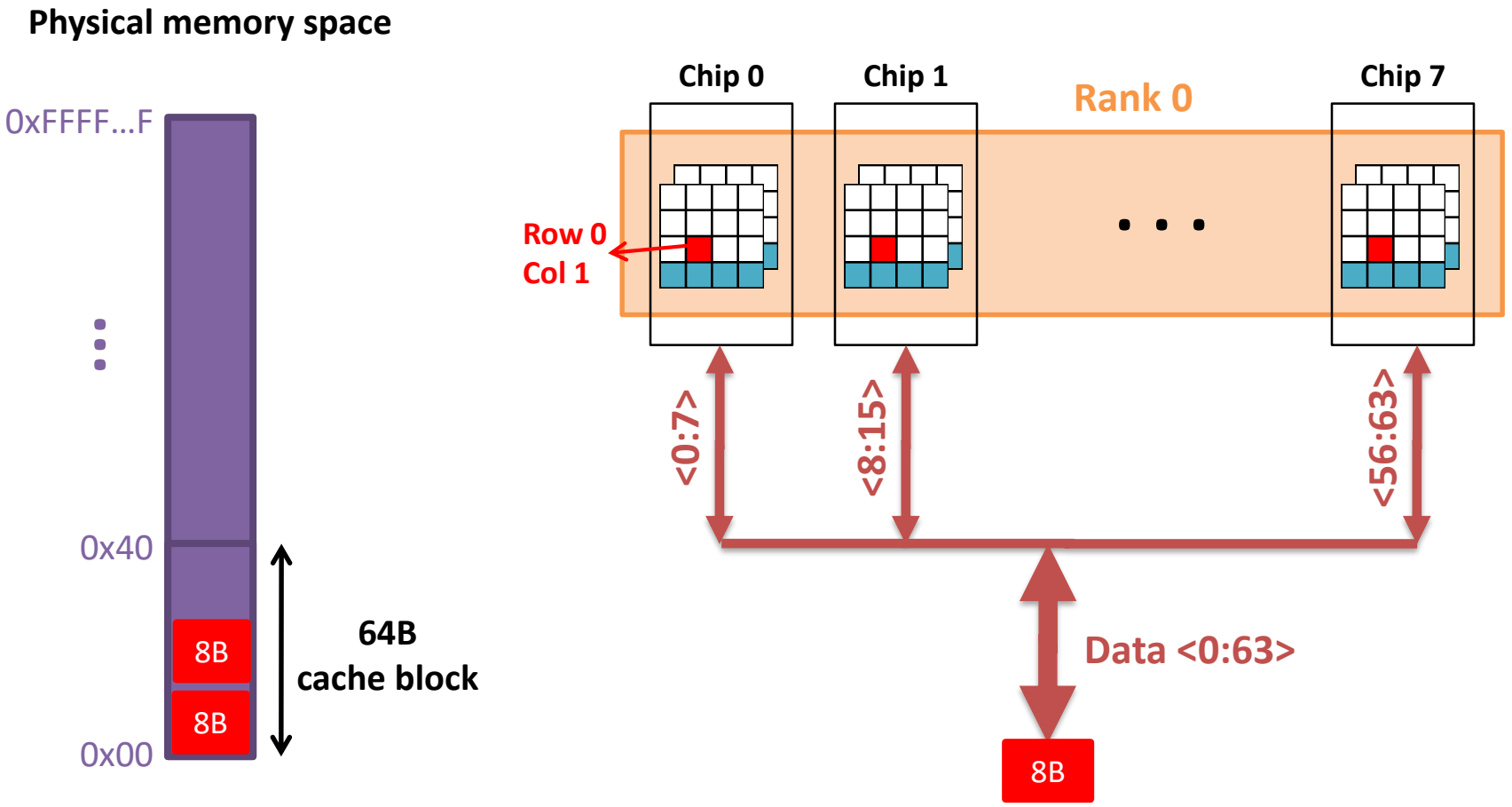
Example: Transferring a cache block



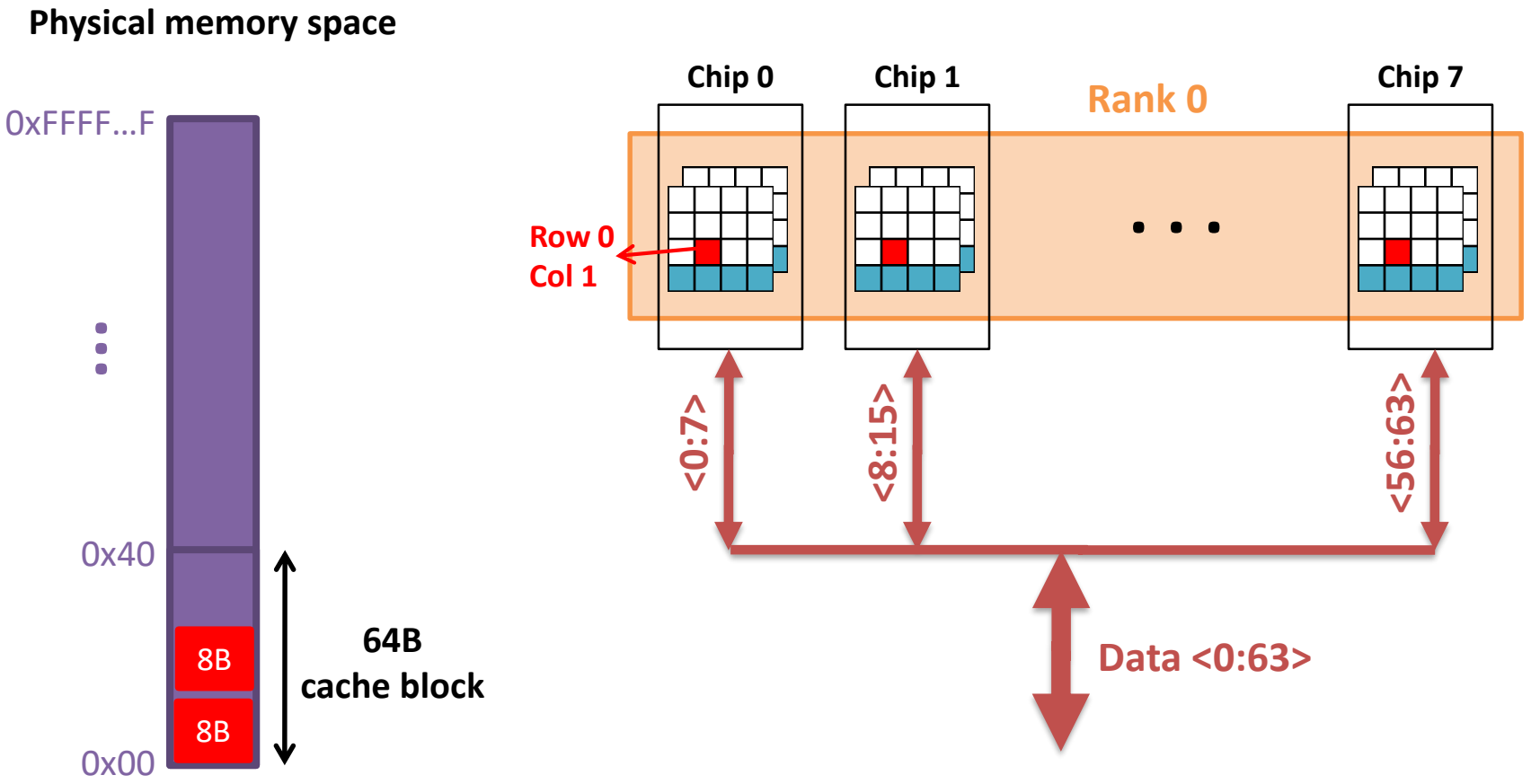
Example: Transferring a cache block



Example: Transferring a cache block



Example: Transferring a cache block



A 64B cache block takes 8 I/O cycles to transfer.

During the process, 8 columns are read sequentially.

Latency Components: Basic DRAM Operation

- CPU → controller transfer time
- Controller latency
  - Queuing & scheduling delay at the controller
  - Access converted to basic commands
- Controller → DRAM transfer time
- DRAM bank latency
  - Simple CAS (column address strobe) if row is "open" OR
  - RAS (row address strobe) + CAS if array precharged OR
  - PRE + RAS + CAS (worst case)
- DRAM → Controller transfer time
  - Bus latency (BL)
- Controller to CPU transfer time

Memory Systems and Memory-Centric Computing Systems
Lecture 1: Memory Trends and Basics

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

9-13 July 2018

HiPEAC ACACES Summer School 2018

SAFARI

ETH zürich

Carnegie Mellon

We did not cover the remaining slides in Lecture 1.

The remaining slides are useful for more background.

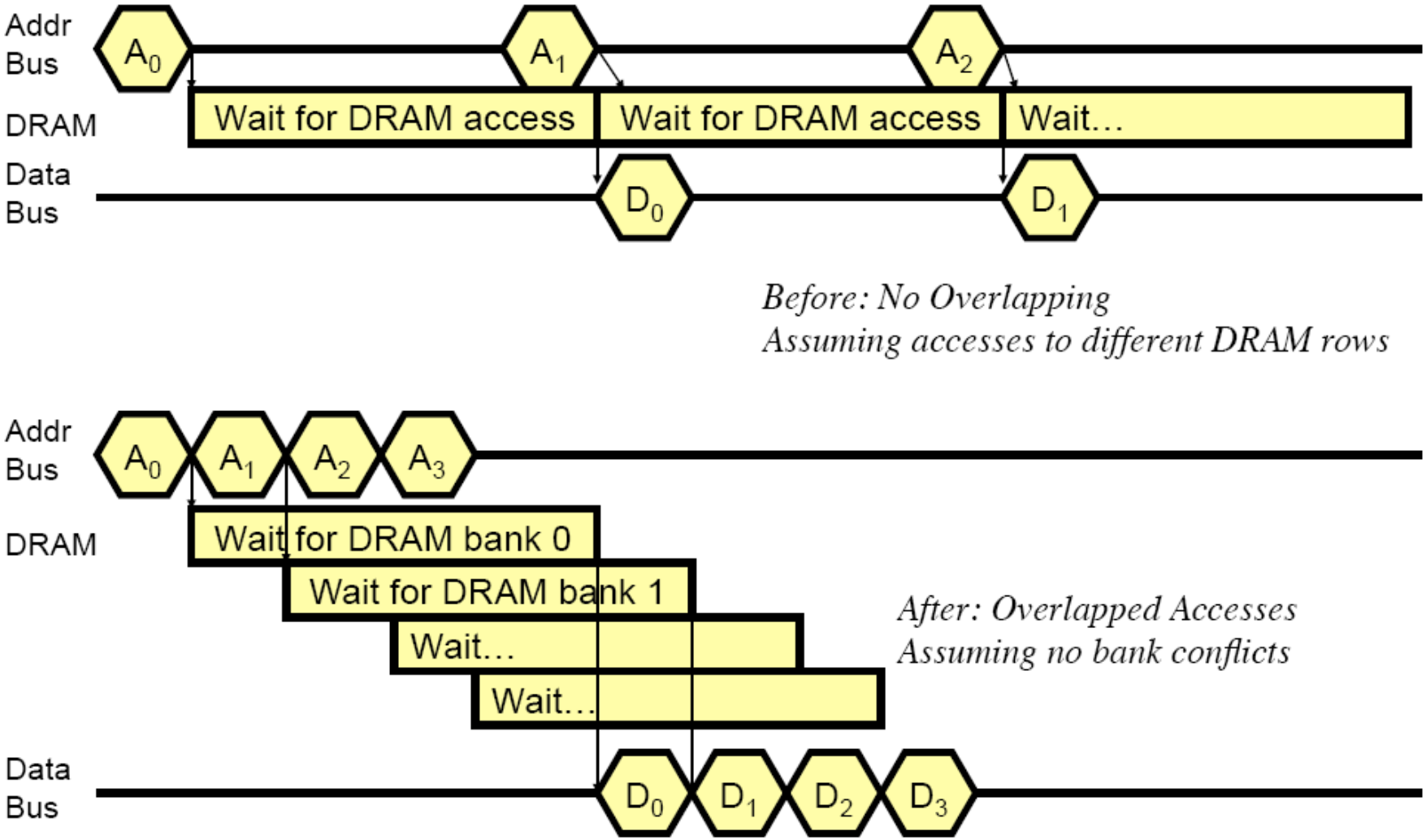
We may cover some (but not all) of them in the rest of the course.

Multiple Banks (Interleaving) and Channels

- Multiple banks
  - Enable concurrent DRAM accesses
  - Bits in address determine which bank an address resides in
- Multiple independent channels serve the same purpose
  - But they are even better because they have separate data buses
  - Increased bus bandwidth

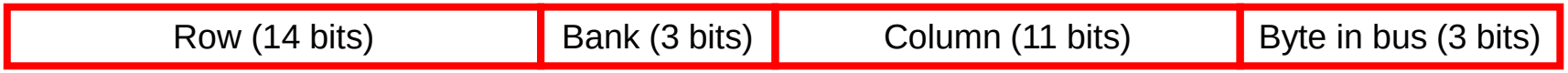
- Enabling more concurrency requires reducing
  - Bank conflicts
  - Channel conflicts
- How to select/randomize bank/channel indices in address?
  - Lower order bits have more entropy
  - Randomizing hash functions (XOR of different address bits)

How Multiple Banks Help

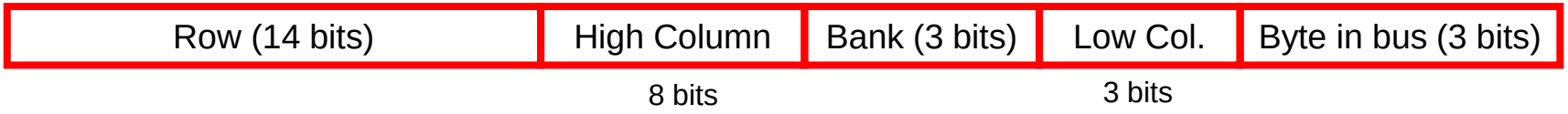


Address Mapping (Single Channel)

- Single-channel system with 8-byte memory bus
  - 2GB memory, 8 banks, 16K rows & 2K columns per bank
- Row interleaving
  - Consecutive rows of memory in consecutive banks



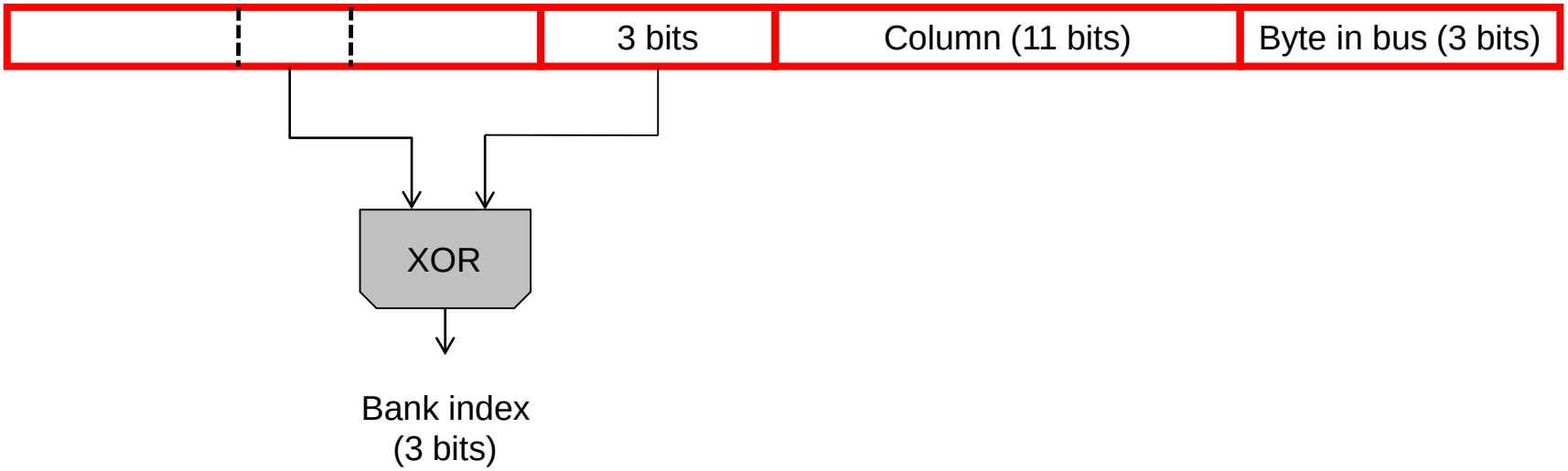
  - Accesses to consecutive cache blocks serviced in a pipelined manner
- Cache block interleaving
  - Consecutive cache block addresses in consecutive banks
  - 64 byte cache blocks



  - Accesses to consecutive cache blocks can be serviced in parallel

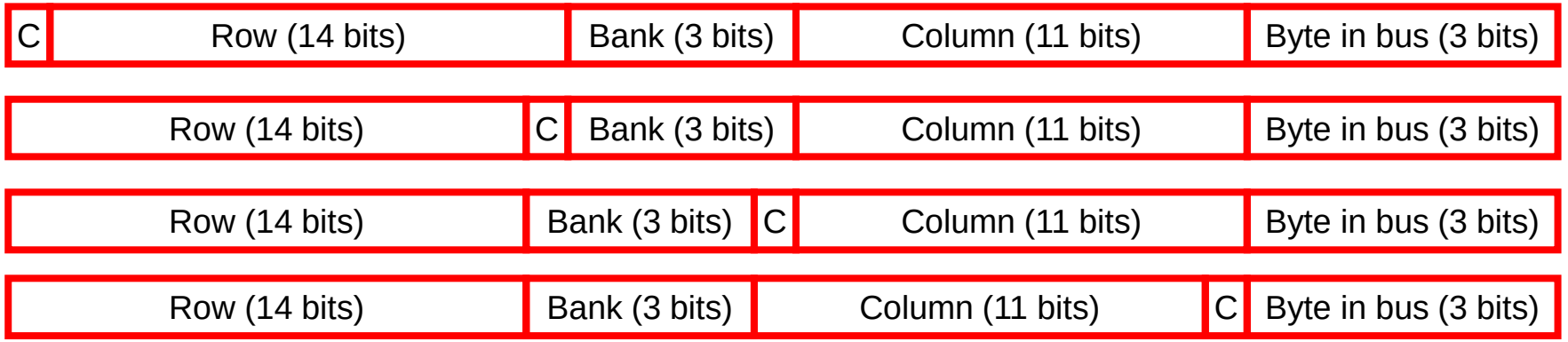
Bank Mapping Randomization

- DRAM controller can randomize the address mapping to banks so that bank conflicts are less likely

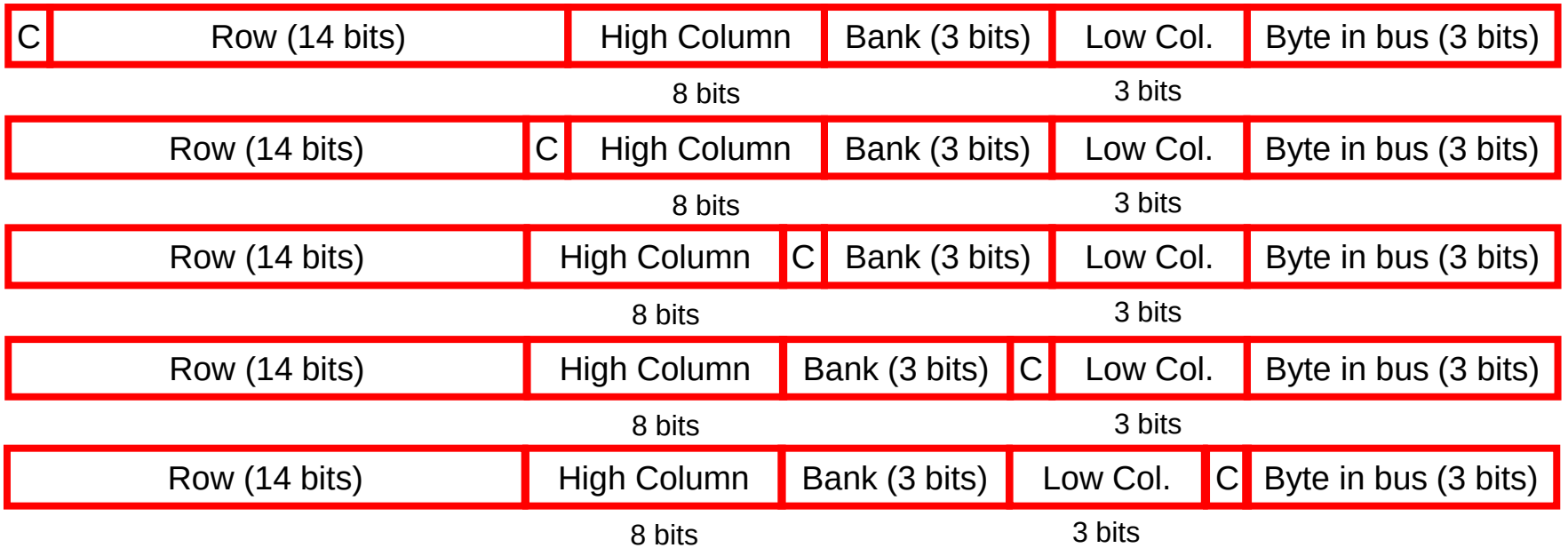


- Reading:
  - Rau, "Pseudo-randomly Interleaved Memory," ISCA 1991.

Address Mapping (Multiple Channels)

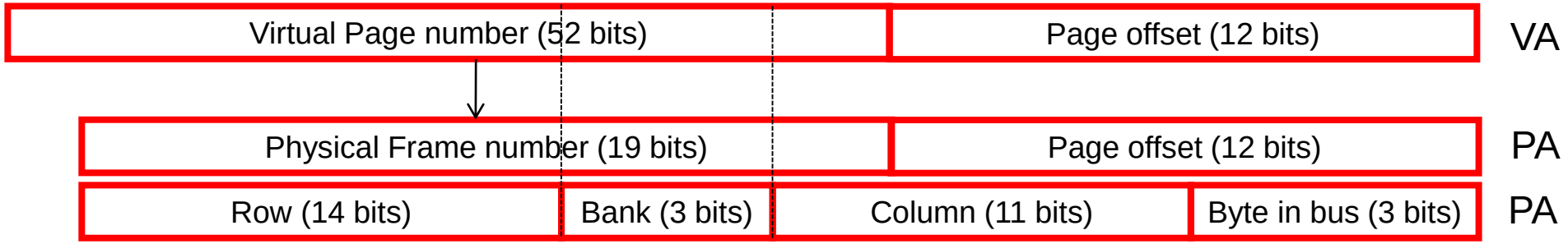


- Where are consecutive cache blocks?



Interaction with Virtual→Physical Mapping

- Operating System influences where an address maps to in DRAM



- Operating system can influence which bank/channel/rank a virtual page is mapped to.

- It can perform page coloring to
  - Minimize bank conflicts
  - Minimize inter-application interference **[Muralidhara+ MICRO'11]**
  - Minimize latency in the network **[Das+ HPCA'13]**

Memory Channel Partitioning

- Sai Prashanth Muralidhara, Lavanya Subramanian, Onur Mutlu, Mahmut Kandemir, and Thomas Moscibroda, **"Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning"** *Proceedings of the 44th International Symposium on Microarchitecture (**MICRO**)*, Porto Alegre, Brazil, December 2011. Slides (pptx)

Reducing Memory Interference in Multicore Systems via Application-Aware Memory Channel Partitioning

Sai Prashanth Muralidhara
Pennsylvania State University
smuralid@cse.psu.edu

Lavanya Subramanian
Carnegie Mellon University
lsubrama@ece.cmu.edu

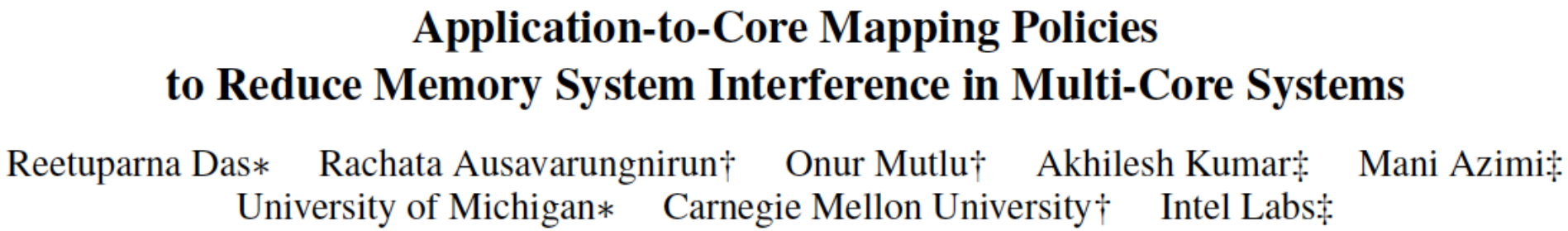
Onur Mutlu
Carnegie Mellon University
onur@cmu.edu

Mahmut Kandemir
Pennsylvania State University
kandemir@cse.psu.edu

Thomas Moscibroda
Microsoft Research Asia
moscitho@microsoft.com

Application-to-Core Mapping

- Reetuparna Das, Rachata Ausavarungnirun, Onur Mutlu, Akhilesh Kumar, and Mani Azimi,
  "Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems"
  *Proceedings of the 19th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Shenzhen, China, February 2013.
  Slides (pptx)



Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems

Reetuparna Das* Rachata Ausavarungnirun† Onur Mutlu† Akhilesh Kumar‡ Mani Azimi‡
University of Michigan* Carnegie Mellon University† Intel Labs‡

More on Reducing Bank Conflicts

- Read Sections 1 through 4 of:
  - Kim et al., "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

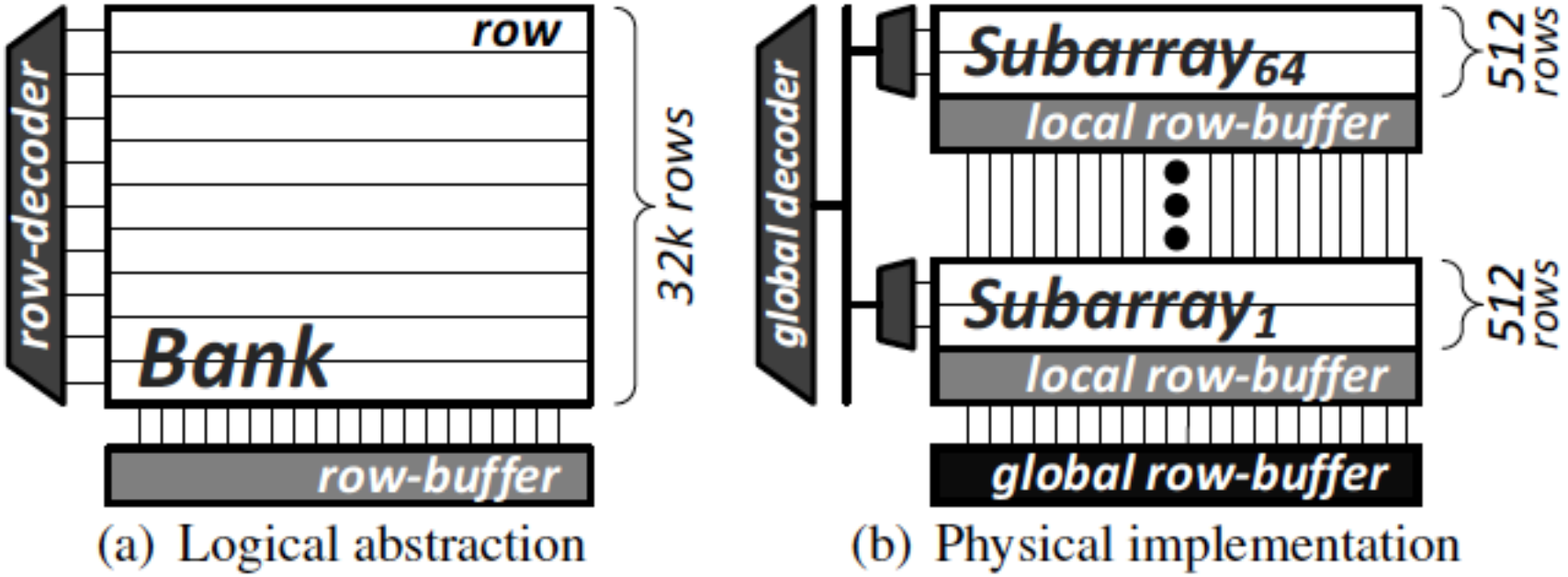


Figure 1. DRAM bank organization

Subarray Level Parallelism

- Yoongu Kim, Vivek Seshadri, Donghyuk Lee, Jamie Liu, and Onur Mutlu, **"A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM"**
  *Proceedings of the 39th International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2012. Slides (pptx)

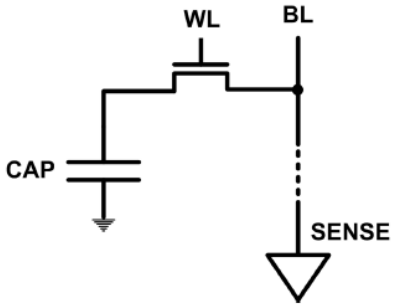
A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM

Yoongu Kim Vivek Seshadri Donghyuk Lee Jamie Liu Onur Mutlu

Carnegie Mellon University

DRAM Refresh (I)

- DRAM capacitor charge leaks over time
- The memory controller needs to read each row periodically to restore the charge
  - Activate + precharge each row every N ms
  - Typical N = 64 ms



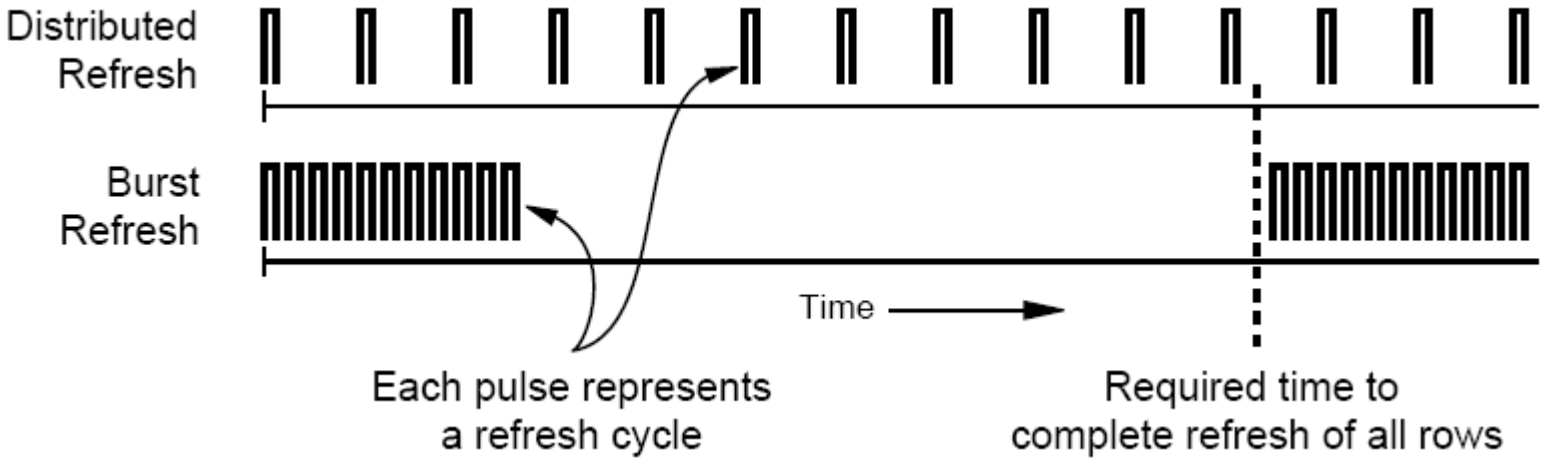
- Implications on performance?

  -- DRAM bank unavailable while refreshed

  -- Long pause times: If we refresh all rows in burst, every 64ms the DRAM will be unavailable until refresh ends

- Burst refresh: All rows refreshed immediately after one another
- Distributed refresh: Each row refreshed at a different time, at regular intervals

DRAM Refresh (II)



- Distributed refresh eliminates long pause times
- How else we can reduce the effect of refresh on performance?
  - Can we reduce the number of refreshes?

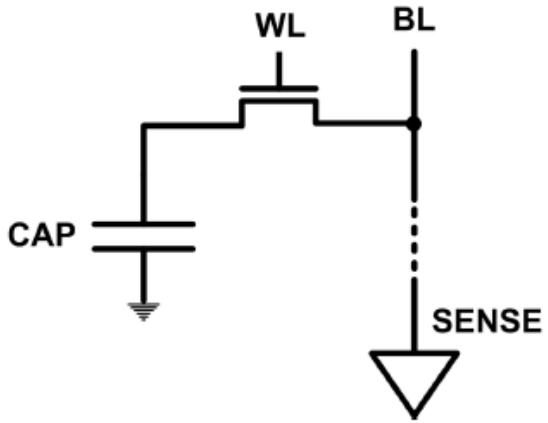
Downsides of DRAM Refresh

-- 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 density scaling



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

More on DRAM Refresh (I)

- Jamie Liu, Ben Jaiyen, Richard Veras, and Onur Mutlu, **"RAIDR: Retention-Aware Intelligent DRAM Refresh"** *Proceedings of the 39th International Symposium on Computer Architecture (**ISCA**)*, Portland, OR, June 2012. Slides (pdf)

RAIDR: Retention-Aware Intelligent DRAM Refresh

Jamie Liu Ben Jaiyen Richard Veras Onur Mutlu
Carnegie Mellon University

DRAM Retention Analysis

- Jamie Liu, Ben Jaiyen, Yoongu Kim, Chris Wilkerson, and Onur Mutlu, **"An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms"** *Proceedings of the 40th International Symposium on Computer Architecture (**ISCA**)*, Tel-Aviv, Israel, June 2013. Slides (ppt) Slides (pdf)

An Experimental Study of Data Retention Behavior in Modern DRAM Devices: Implications for Retention Time Profiling Mechanisms

Jamie Liu*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
jamiel@alumni.cmu.edu

Ben Jaiyen*
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
bjaiyen@alumni.cmu.edu

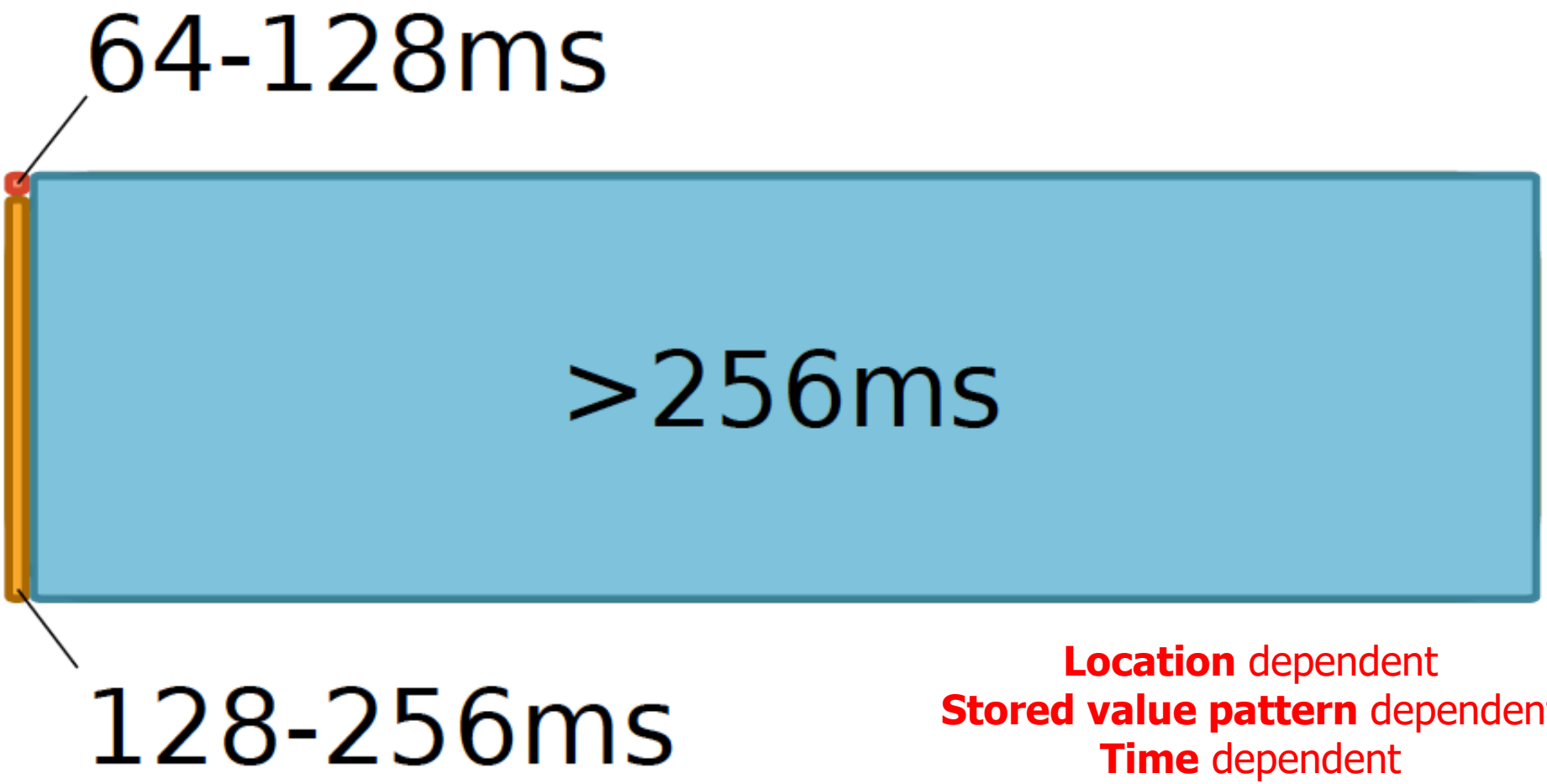
Yoongu Kim
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
yoonguk@ece.cmu.edu

Chris Wilkerson
Intel Corporation
2200 Mission College Blvd.
Santa Clara, CA 95054
chris.wilkerson@intel.com

Onur Mutlu
Carnegie Mellon University
5000 Forbes Ave.
Pittsburgh, PA 15213
onur@cmu.edu

Data Retention in Memory [Liu et al., ISCA 2013]

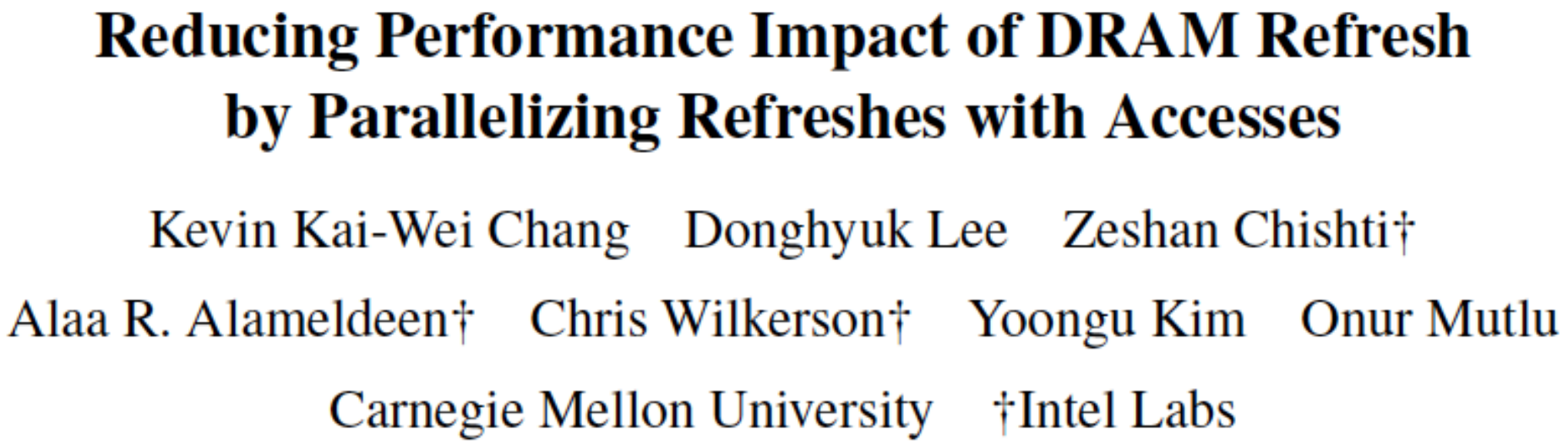
- Data Retention Time Profile of DRAM looks like this:



Location dependent
Stored value pattern dependent
Time dependent

DRAM Refresh-Access Parallelization

- Kevin Chang, Donghyuk Lee, Zeshan Chishti, Alaa Alameldeen, Chris Wilkerson, Yoongu Kim, and Onur Mutlu,
  "Improving DRAM Performance by Parallelizing Refreshes with Accesses"
  *Proceedings of the 20th International Symposium on High-Performance Computer Architecture (**HPCA**)*, Orlando, FL, February 2014.
  [Summary] [Slides (pptx) (pdf)]



Reducing Performance Impact of DRAM Refresh by Parallelizing Refreshes with Accesses

Kevin Kai-Wei Chang Donghyuk Lee Zeshan Chishti†

Alaa R. Alameldeen† Chris Wilkerson† Yoongu Kim Onur Mutlu

Carnegie Mellon University †Intel Labs

Memory Controllers

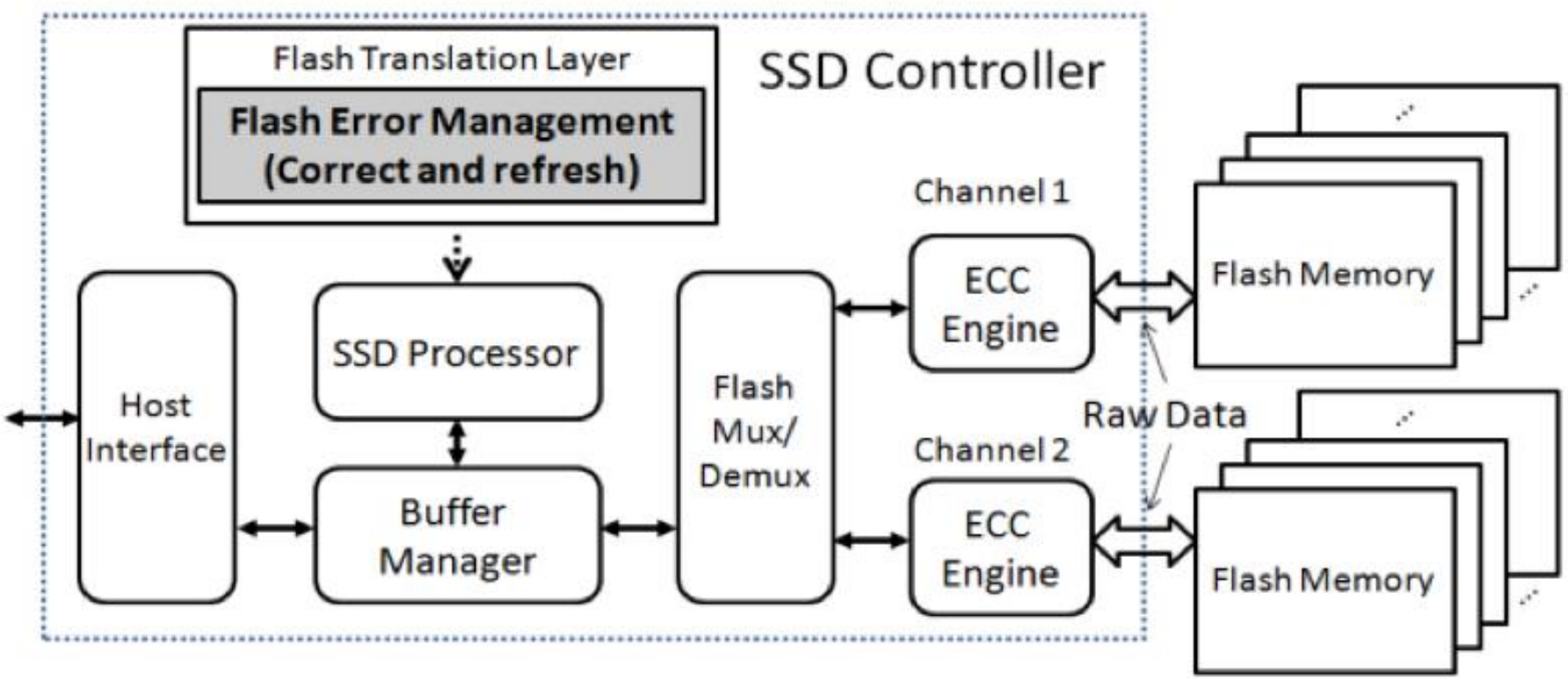
DRAM versus Other Types of Memories

- Long latency memories have similar characteristics that need to be controlled.

- The following discussion will use DRAM as an example, but many scheduling and control issues are similar in the design of controllers for other types of memories
  - Flash memory
  - Other emerging memory technologies
    - Phase Change Memory
    - Spin-Transfer Torque Magnetic Memory
  - These other technologies can place other demands on the controller

Flash Memory (SSD) Controllers

- Similar to DRAM memory controllers, except:
  - They are flash memory specific
  - They do much more: error correction, garbage collection, page remapping, …



Cai+, "Flash Correct-and-Refresh: Retention-Aware Error Management for Increased Flash Memory Lifetime", ICCD 2012.

Another View of the SSD Controller

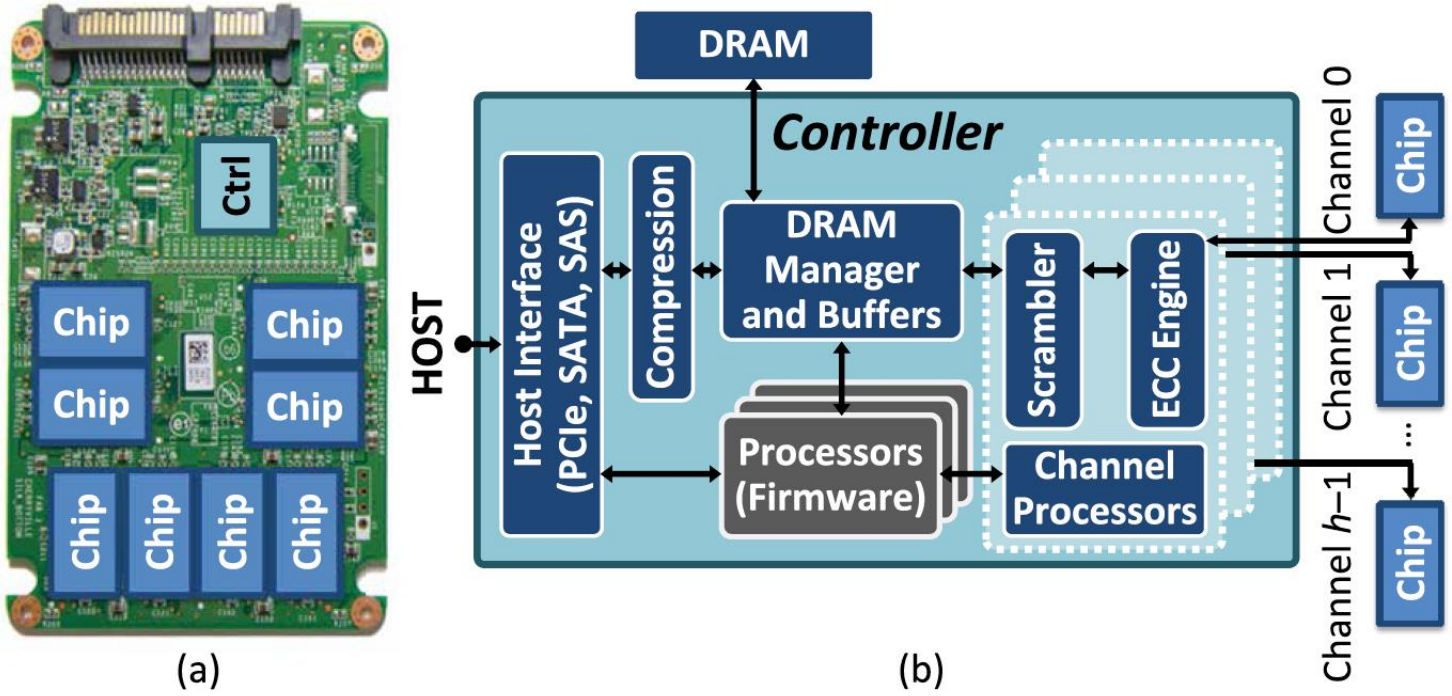


Fig. 1. ***(a) SSD system architecture, showing controller (Ctrl) and chips. (b) Detailed view of connections between controller components and chips.***

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

https://arxiv.org/pdf/1711.11427.pdf

On Modern SSD Controllers (I)

Proceedings of the IEEE, Sept. 2017

Error Characterization, Mitigation, and Recovery in Flash-Memory-Based Solid-State Drives

This paper reviews the most recent advances in solid-state drive (SSD) error characterization, mitigation, and data recovery techniques to improve both SSD's reliability and lifetime.

By Yu Cai, Saugata Ghose, Erich F. Haratsch, Yixin Luo, and Onur Mutlu

https://arxiv.org/pdf/1706.08642

On Modern SSD Controllers (II)

- Arash Tavakkol, Juan Gomez-Luna, Mohammad Sadrosadati, Saugata Ghose, and Onur Mutlu,
  "MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices"
  *Proceedings of the 16th USENIX Conference on File and Storage Technologies (**FAST**)*, Oakland, CA, USA, February 2018.
  [Slides (pptx) (pdf)]
  [Source Code]

MQSim: A Framework for Enabling Realistic Studies of Modern Multi-Queue SSD Devices

Arash Tavakkol[†], Juan Gómez-Luna[†], Mohammad Sadrosadati[†], Saugata Ghose[‡], Onur Mutlu[†‡]
[†]*ETH Zürich* [‡]*Carnegie Mellon University*

On Modern SSD Controllers (III)

- Arash Tavakkol, Mohammad Sadrosadati, Saugata Ghose, Jeremie Kim, Yixin Luo, Yaohua Wang, Nika Mansouri Ghiasi, Lois Orosa, Juan G. Luna and Onur Mutlu,
  "FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives"
  *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]

FLIN: Enabling Fairness and Enhancing Performance in Modern NVMe Solid State Drives

Arash Tavakkol[†] Mohammad Sadrosadati[†] Saugata Ghose[‡] Jeremie S. Kim[‡†] Yixin Luo[‡]
Yaohua Wang[†§] Nika Mansouri Ghiasi[†] Lois Orosa[†*] Juan Gómez-Luna[†] Onur Mutlu[†‡]
[†]*ETH Zürich* [‡]*Carnegie Mellon University* [§]*NUDT* [*]*Unicamp*

DRAM Types

- DRAM has different types with different interfaces optimized for different purposes
  - Commodity: DDR, DDR2, DDR3, DDR4, …
  - Low power (for mobile): LPDDR1, …, LPDDR5, …
  - High bandwidth (for graphics): GDDR2, …, GDDR5, …
  - Low latency: eDRAM, RLDRAM, …
  - 3D stacked: WIO, HBM, HMC, …
  - …
- Underlying microarchitecture is fundamentally the same
- A flexible memory controller can support various DRAM types
- This complicates the memory controller
  - Difficult to support all types (and upgrades)

DRAM Types (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]

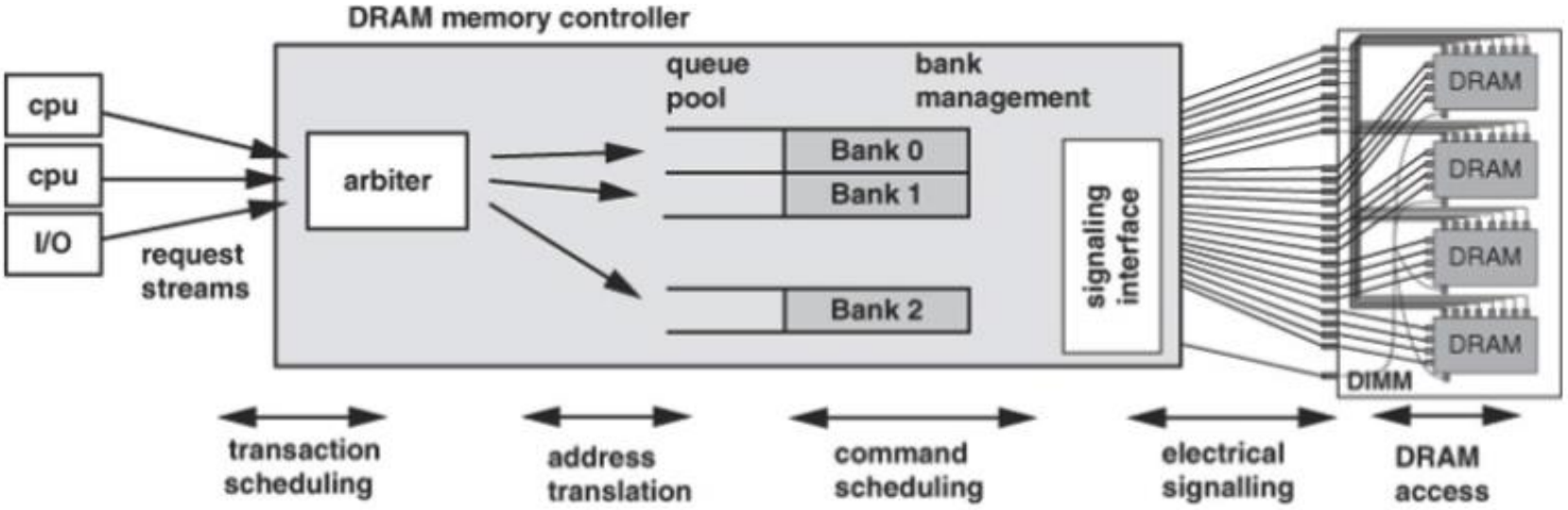
Table 1. Landscape of DRAM-based memory

Kim et al., "Ramulator: A Fast and Extensible DRAM Simulator," IEEE Comp Arch Letters 2015.

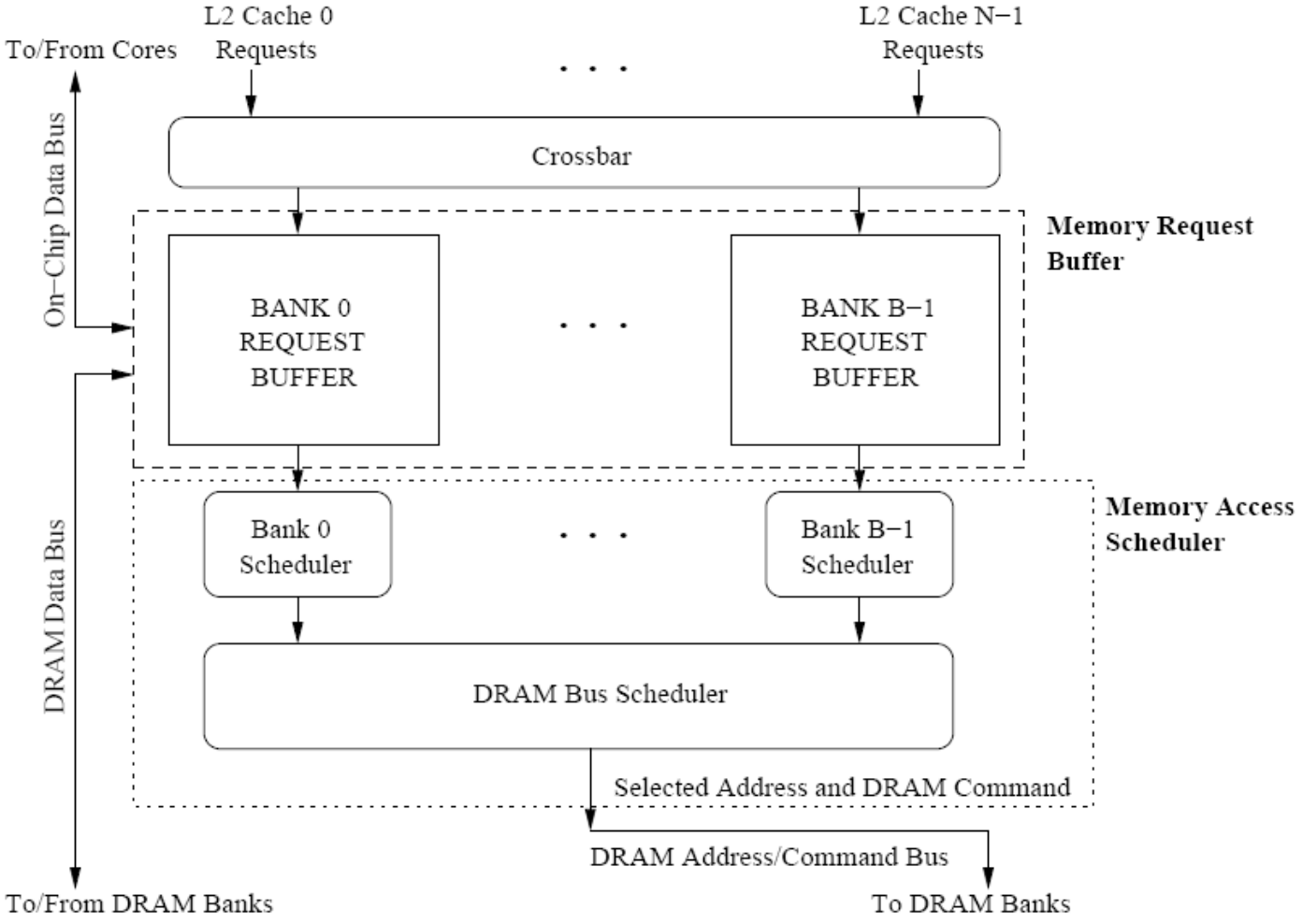
DRAM Controller: Functions

- Ensure correct operation of DRAM (refresh and timing)
- Service DRAM requests while obeying timing constraints of DRAM chips
  - Constraints: resource conflicts (bank, bus, channel), minimum write-to-read delays
  - Translate requests to DRAM command sequences
- Buffer and schedule requests to for high performance + QoS
  - Reordering, row-buffer, bank, rank, bus management
- Manage power consumption and thermals in DRAM
  - Turn on/off DRAM chips, manage power modes

A Modern DRAM Controller (I)



A Modern DRAM Controller



DRAM Scheduling Policies (I)

- FCFS (first come first served)
  - Oldest request first

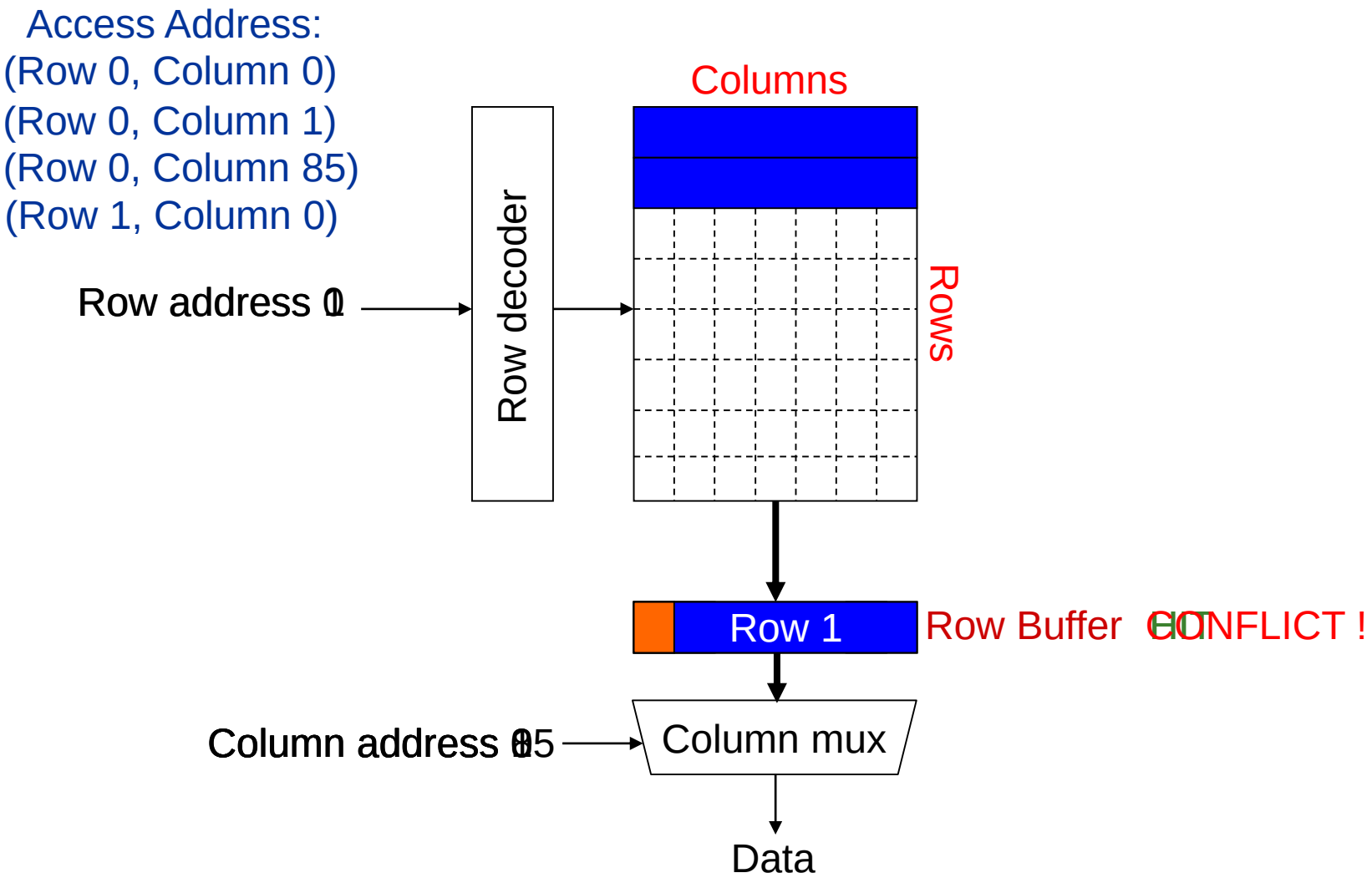
- FR-FCFS (first ready, first come first served)

  1. Row-hit first
  2. Oldest first

  Goal: Maximize row buffer hit rate → maximize DRAM throughput

  - Actually, scheduling is done at the command level
    - Column commands (read/write) prioritized over row commands (activate/precharge)
    - Within each group, older commands prioritized over younger ones

Review: DRAM Bank Operation



DRAM Scheduling Policies (II)

- A scheduling policy is a request prioritization order

- Prioritization can be based on
  - Request age
  - Row buffer hit/miss status
  - Request type (prefetch, read, write)
  - Requestor type (load miss or store miss)
  - Request criticality
    - Oldest miss in the core?
    - How many instructions in core are dependent on it?
    - Will it stall the processor?
  - Interference caused to other cores
  - ...

Row Buffer Management Policies

- Open row
  - Keep the row open after an access

  + Next access might need the same row → row hit

  -- Next access might need a different row → row conflict, wasted energy

- Closed row
  - Close the row after an access (if no other requests already in the request buffer need the same row)

  + Next access might need a different row → avoid a row conflict

  -- Next access might need the same row → extra activate latency

- Adaptive policies
  - Predict whether or not the next access to the bank will be to the same row

Open vs. Closed Row Policies

Policy	First access	Next access	Commands needed for next access
Open row	Row 0	Row 0 (row hit)	Read
Open row	Row 0	Row 1 (row conflict)	Precharge + Activate Row 1 + Read
Closed row	Row 0	Row 0 – access in request buffer (row hit)	Read
Closed row	Row 0	Row 0 – access not in request buffer (row closed)	Activate Row 0 + Read + Precharge
Closed row	Row 0	Row 1 (row closed)	Activate Row 1 + Read + Precharge

DRAM Power Management

- DRAM chips have power modes
- Idea: When not accessing a chip power it down

- Power states
  - Active (highest power)
  - All banks idle
  - Power-down
  - Self-refresh (lowest power)

- Tradeoff: State transitions incur latency during which the chip cannot be accessed

Difficulty of DRAM Control

Why are DRAM 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 DRAM 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.

More on DRAM Operation

- 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.

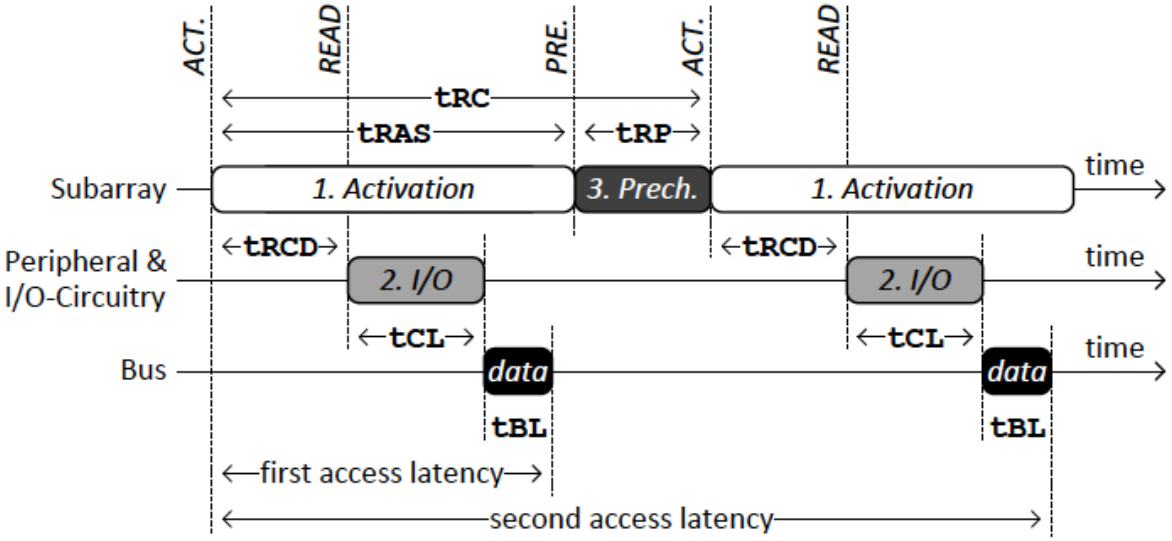


Figure 5. Three Phases of DRAM Access

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

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

Why So Many Timing Constraints? (I)

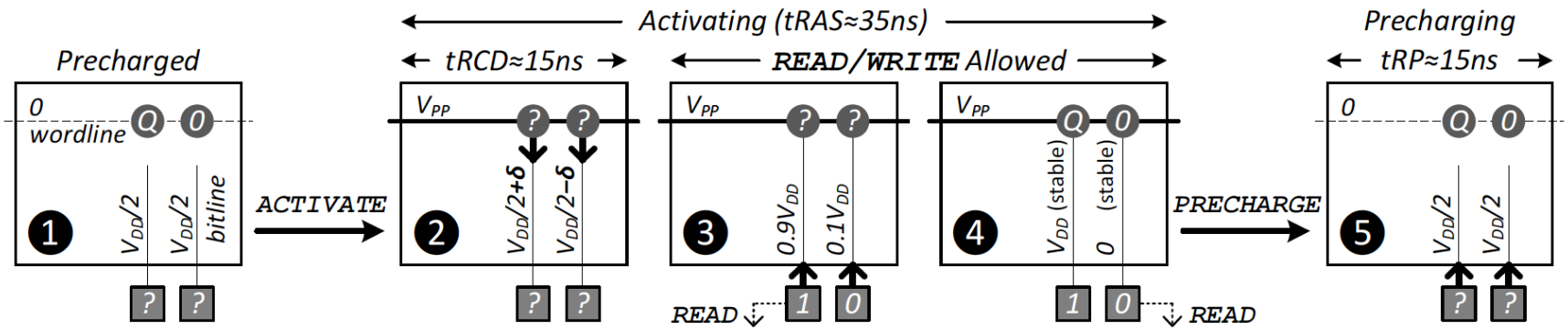


Figure 4. DRAM bank operation: Steps involved in serving a memory request [17] ($V_{PP} > V_{DD}$)

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)

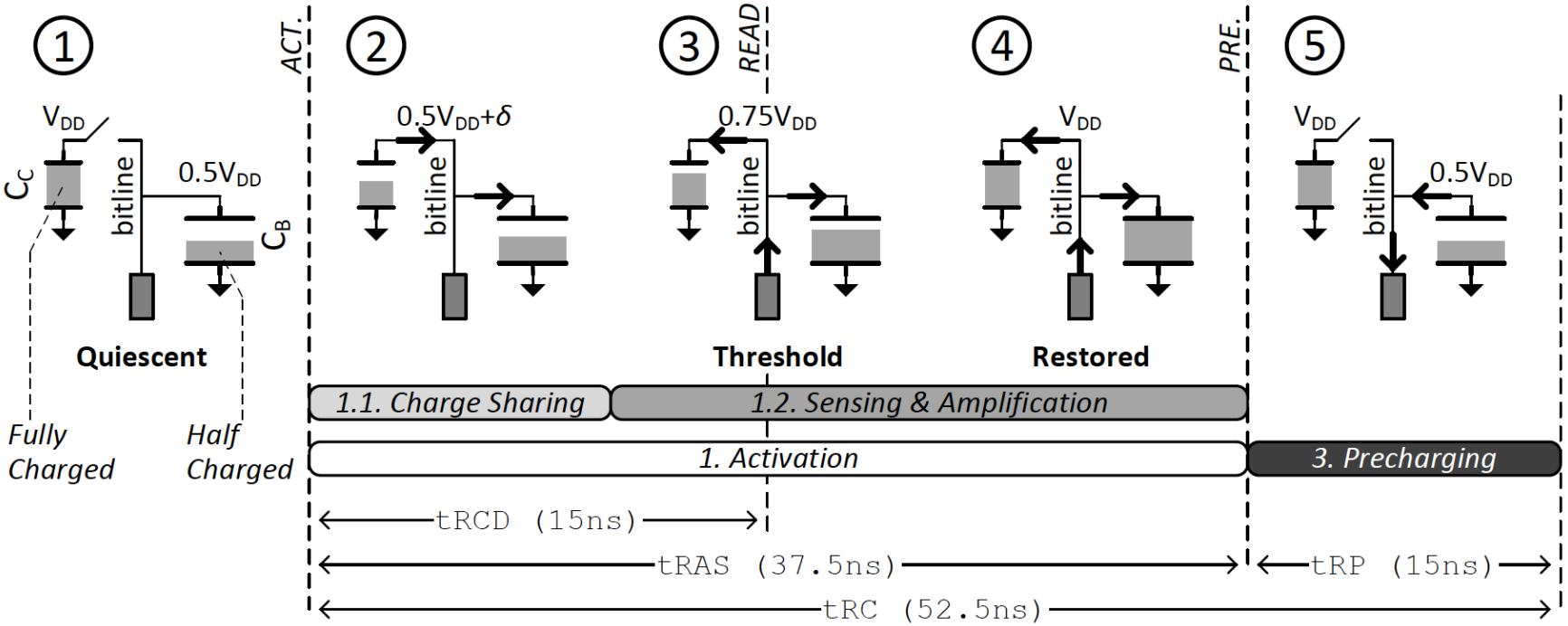


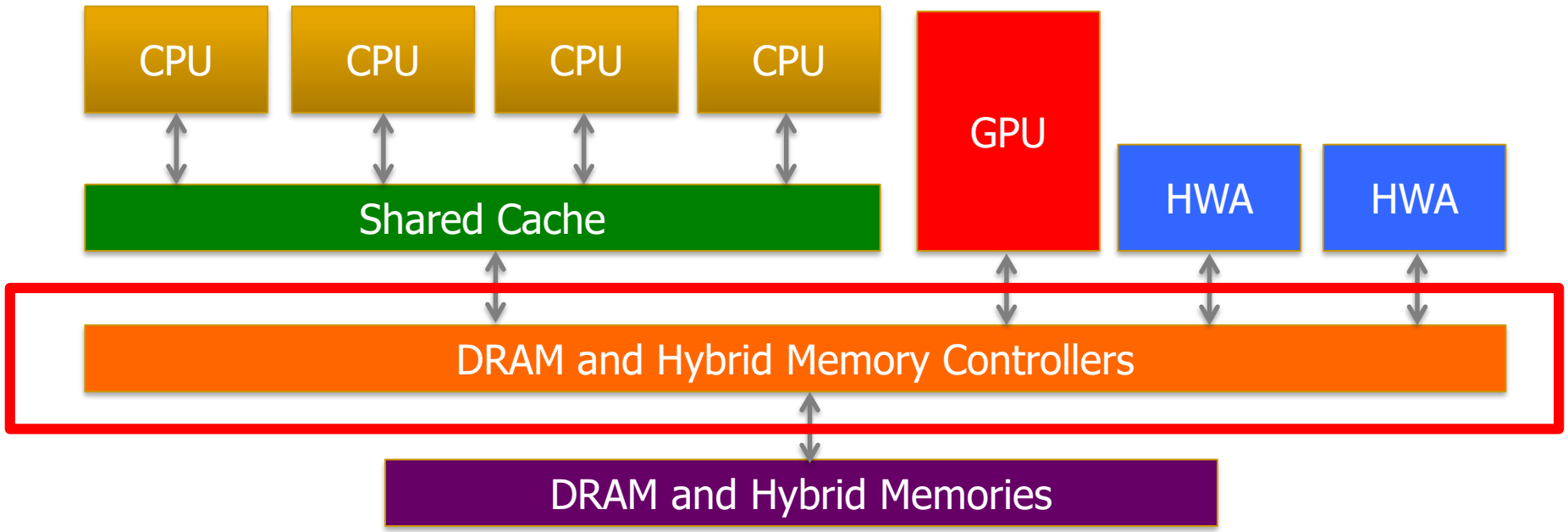
Figure 6. Charge Flow Between the Cell Capacitor (C_C), Bitline Parasitic Capacitor (C_B), and the Sense-Amplifier ($C_B \approx 3.5C_C$ [39])

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

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

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

DRAM 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

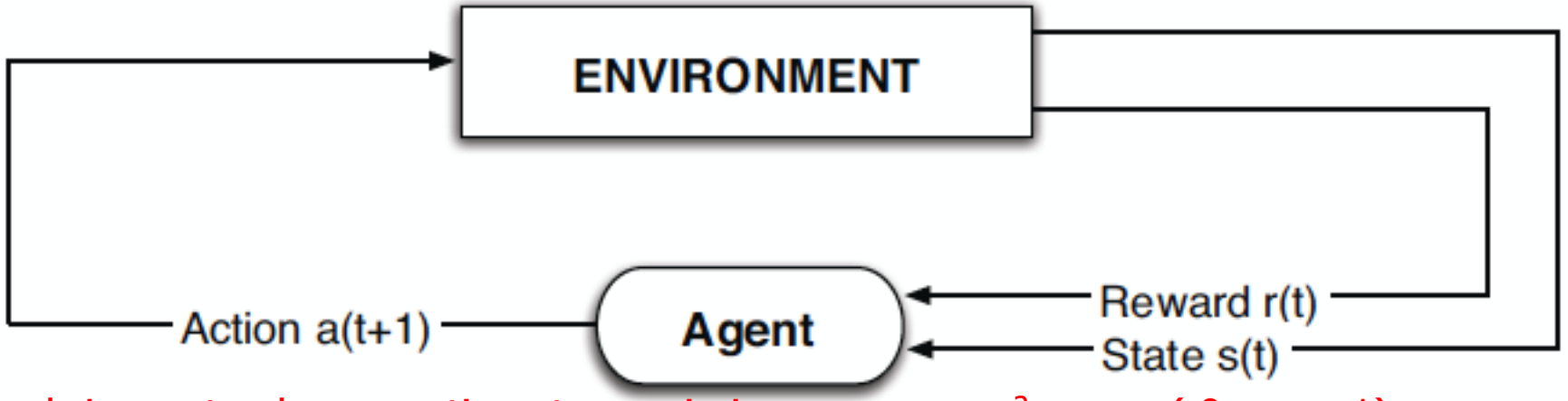
- Reality: It difficult to optimize all these different constraints while maximizing performance, QoS, energy-efficiency, …
- Dream: Wouldn't it be nice if the DRAM controller automatically found a good scheduling policy on its own?

Self-Optimizing DRAM Controllers

- Problem: DRAM controllers 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: Design a memory controller that adapts its scheduling policy decisions 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 that dynamically and continuously learns and employs the best scheduling policy.

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

Self-Optimizing DRAM Controllers

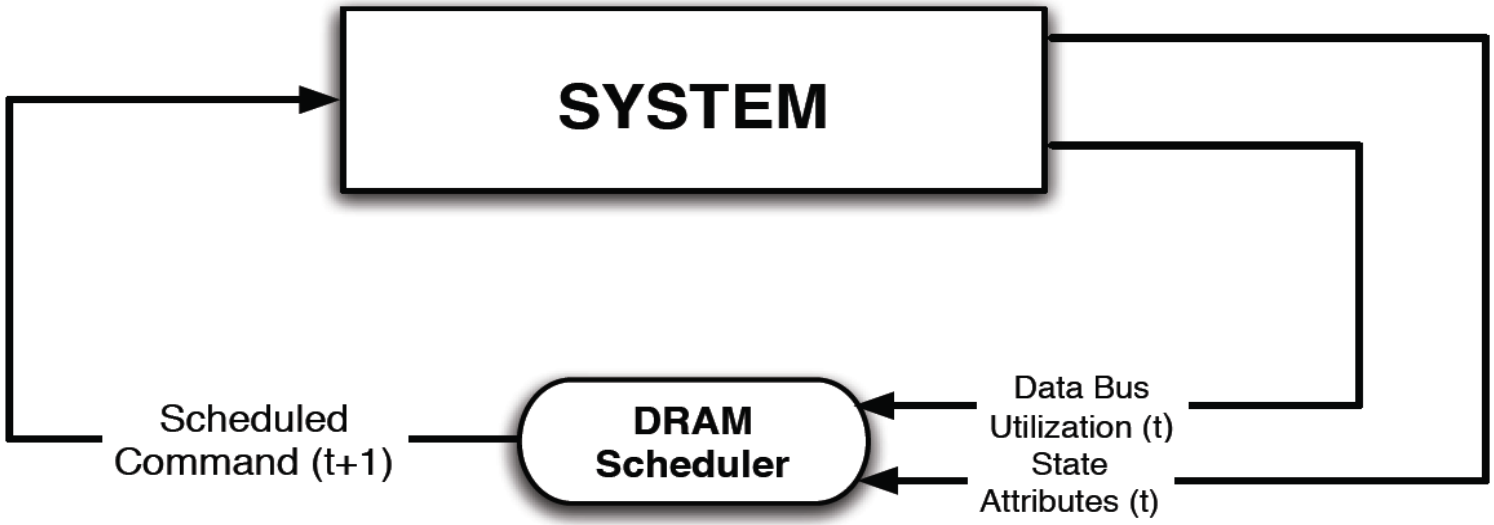


Goal: Learn to choose actions to maximize $r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots$ ($0 \leq \gamma < 1$)

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.

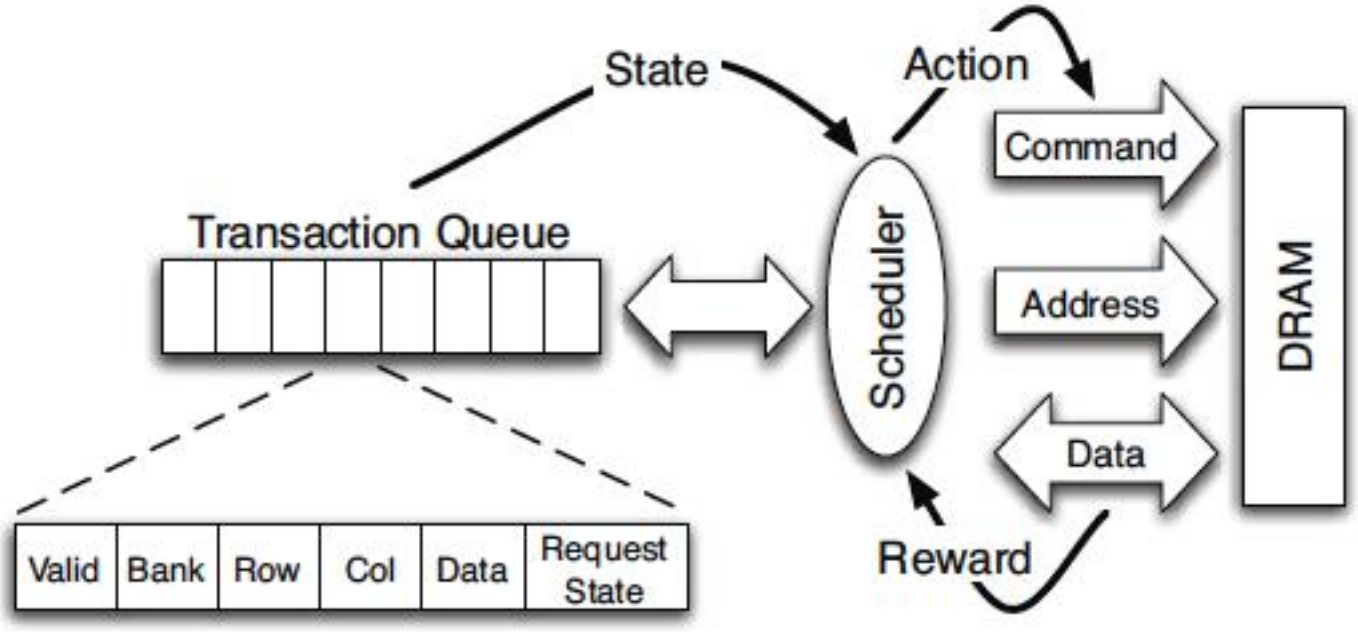


Figure 4: High-level overview of an RL-based scheduler.

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

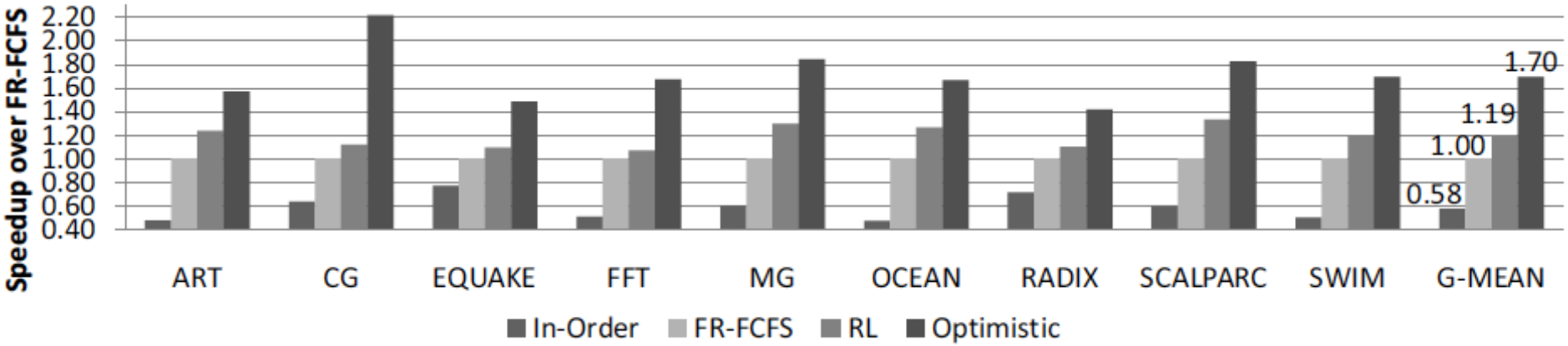


Figure 7: Performance comparison of in-order, FR-FCFS, RL-based, and optimistic memory controllers

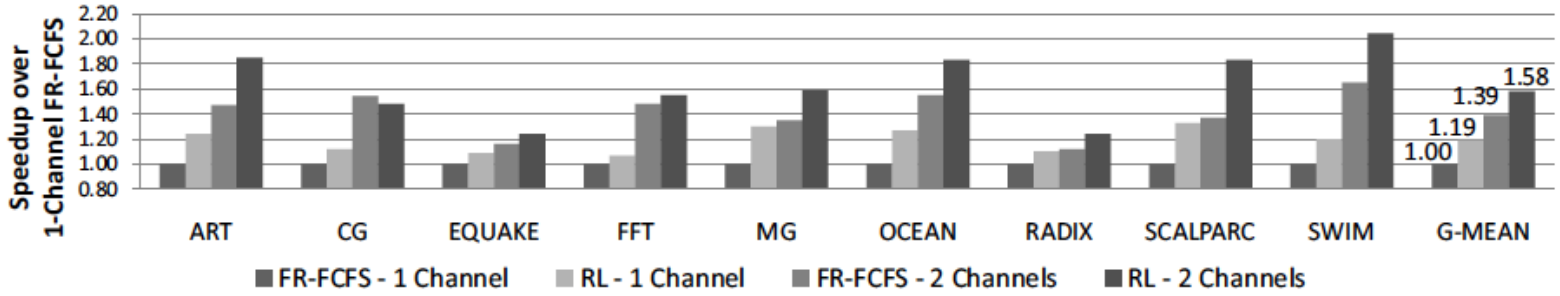


Figure 15: Performance comparison of FR-FCFS and RL-based memory controllers on systems with 6.4GB/s and 12.8GB/s peak DRAM bandwidth

Self Optimizing DRAM Controllers

- Advantages

  + Adapts the scheduling policy dynamically to changing workload behavior and to maximize a long-term target

  + Reduces the designer's 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

- Disadvantages and Limitations

  -- Black box: designer much less likely to implement what she cannot easily reason about

  -- How to specify different reward functions that can achieve different objectives? (e.g., fairness, QoS)

  -- Hardware complexity?

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

Simulating Memory

Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters'15]

Ramulator Motivation

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

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

Table 1. Landscape of DRAM-based memory

Ramulator

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

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

Table 3. Comparison of five simulators using two traces

Case Study: Comparison of DRAM Standards

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

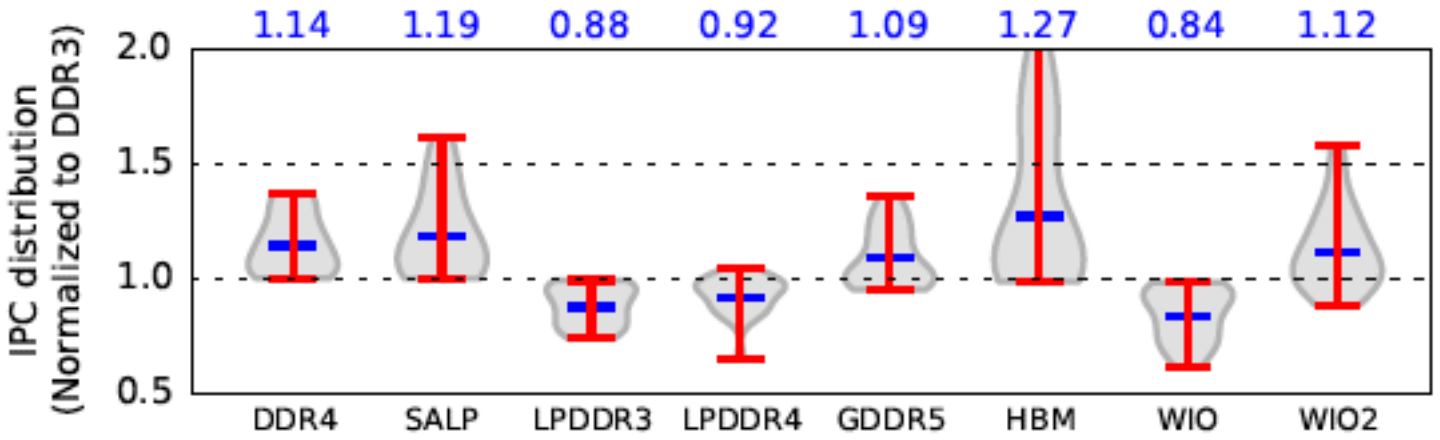


Figure 2. Performance comparison of DRAM standards

Across 22 workloads, simple CPU model

Ramulator Paper and Source Code

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

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

Ramulator: A Fast and Extensible DRAM Simulator

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

Optional Assignment

- Review the Ramulator paper
  - Email me your review (omutlu@gmail.com)
- Download and run Ramulator
  - Compare DDR3, DDR4, SALP, HBM for the libquantum benchmark (provided in Ramulator repository)
  - Email me your report (omutlu@gmail.com)
- This may help you get into memory systems research quickly

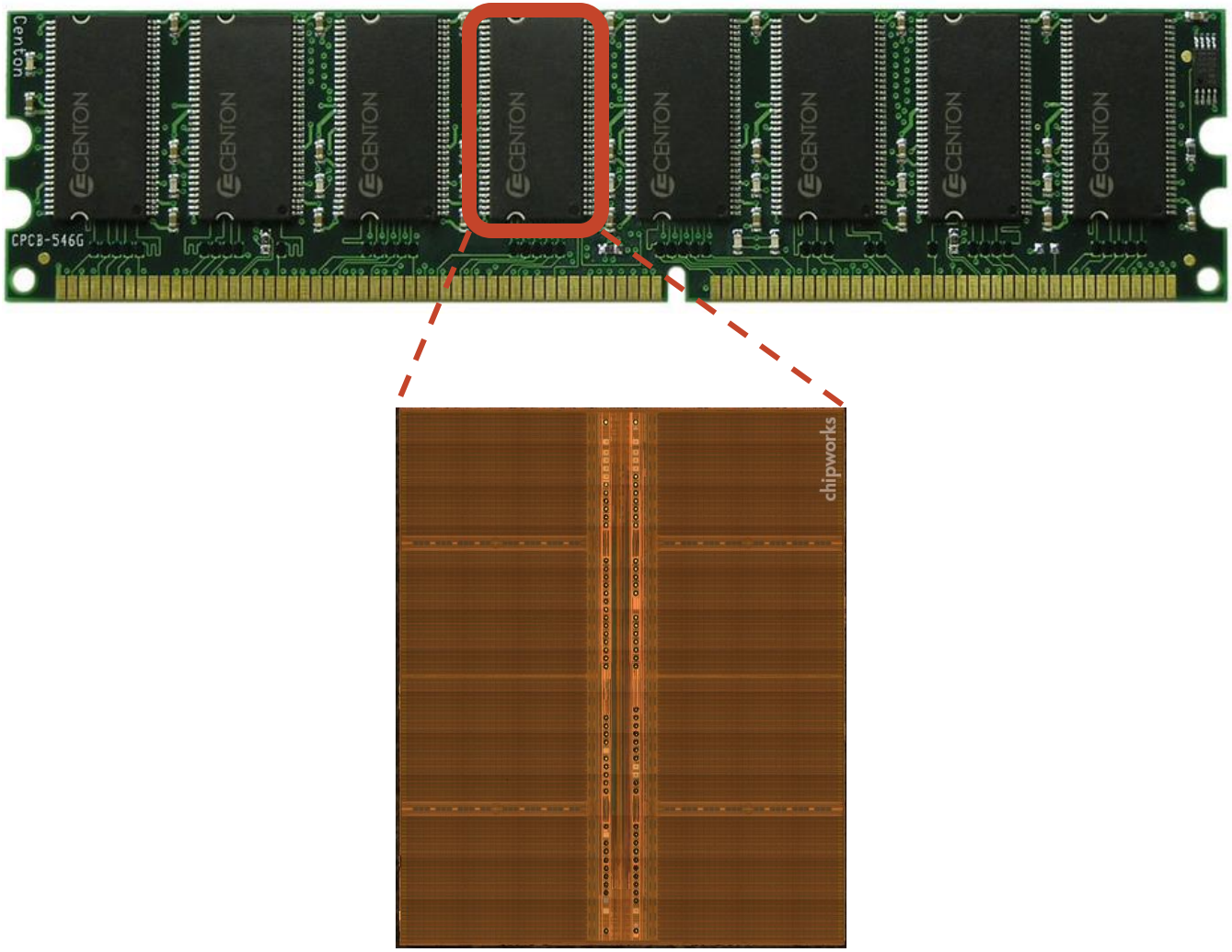
Topics We Will Not Cover

No Time, Unfortunately, for:

- Memory Interference and QoS
- Predictable Performance
  - QoS-aware Memory Controllers
- Emerging Memory Technologies and Hybrid Memories
- Cache Management
- Interconnects
- You can find many materials on these at my online lectures
  - https://people.inf.ethz.ch/omutlu/teaching.html

Inside A DRAM Chip

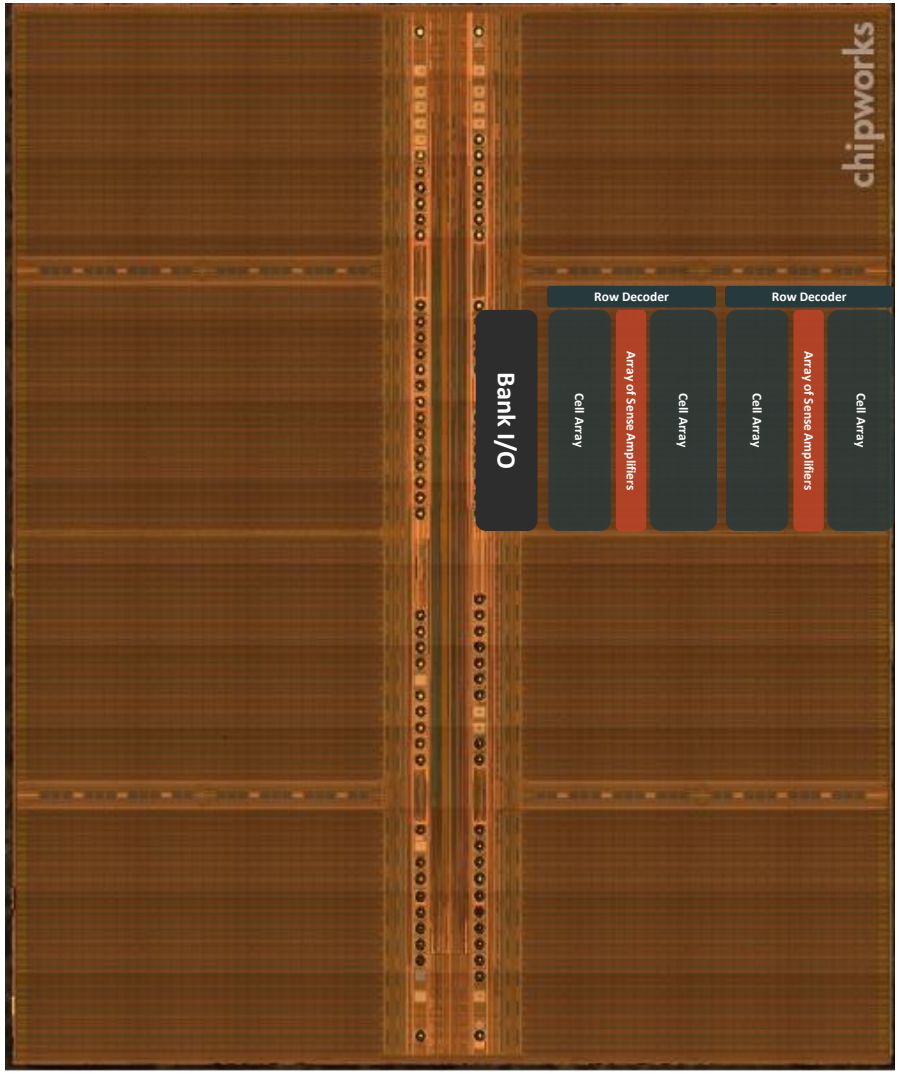
DRAM Module and Chip



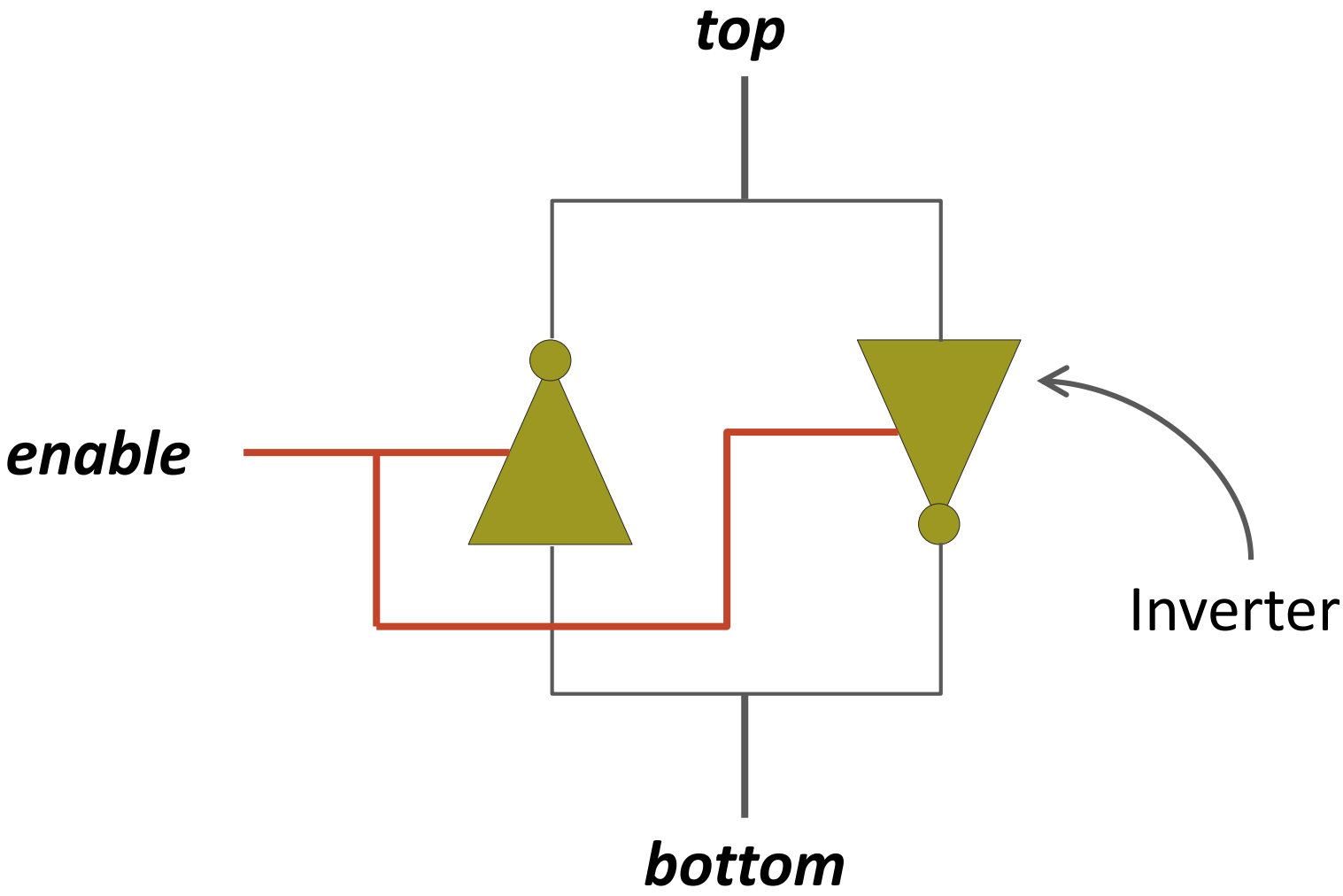
Goals

- Cost
- Latency
- Bandwidth
- Parallelism
- Power
- Energy
- Reliability
- …

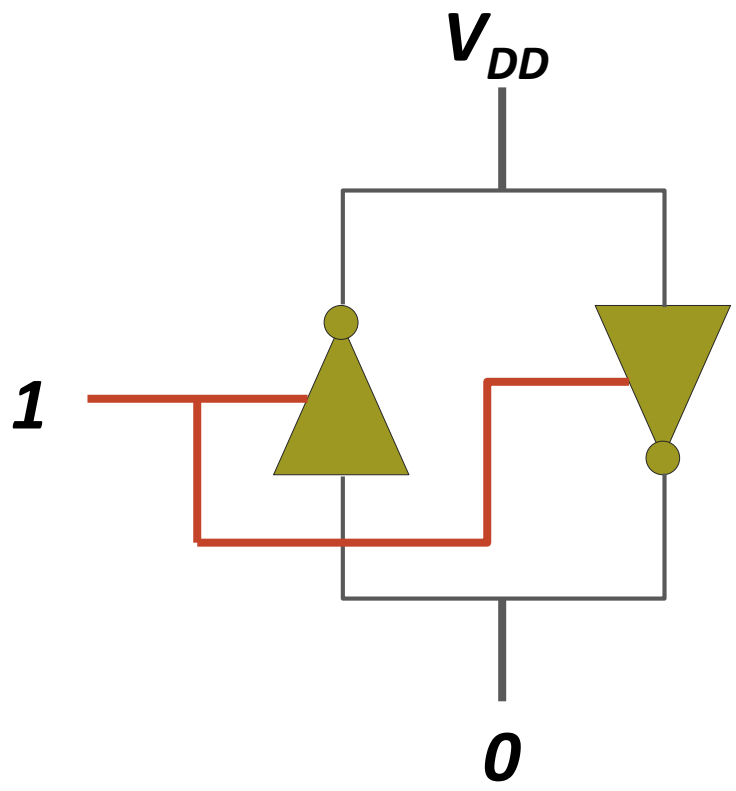
DRAM Chip


chipworks
Row Decoder
Row Decoder
Bank I/O
Cell Array
Array of Sense Amplifiers
Cell Array
Cell Array
Array of Sense Amplifiers
Cell Array

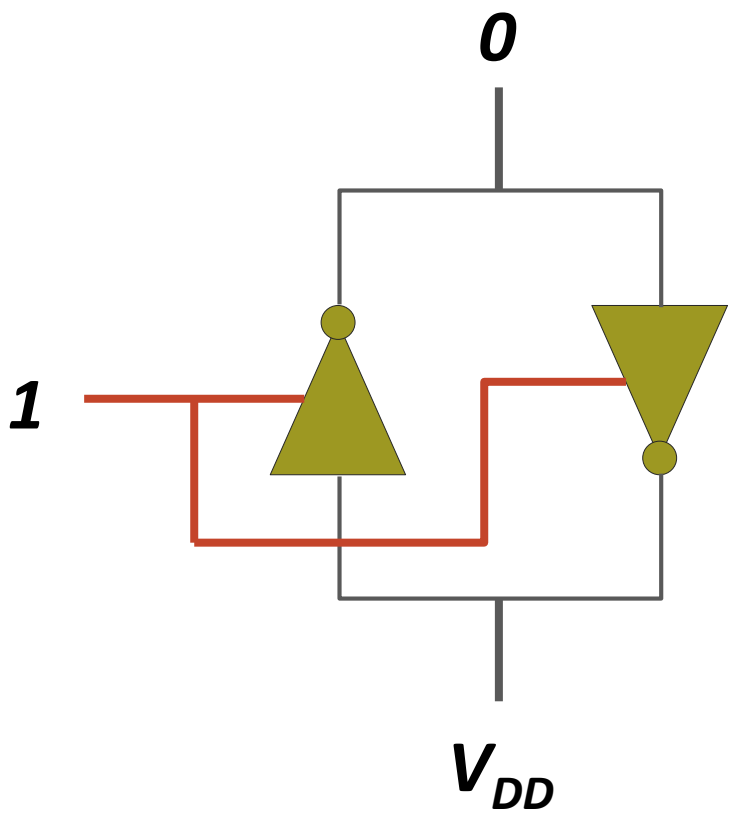
Sense Amplifier



Sense Amplifier – Two Stable States

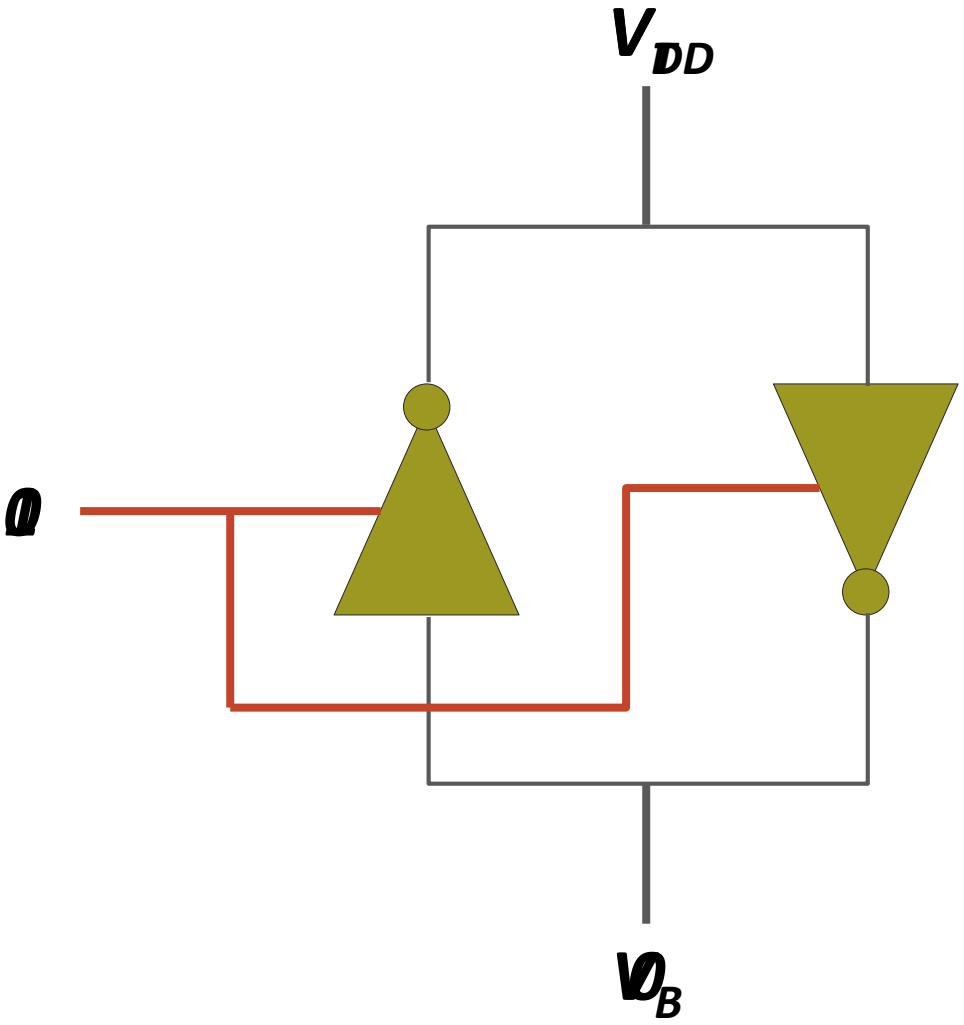


Logical “1”



Logical “0”

Sense Amplifier Operation

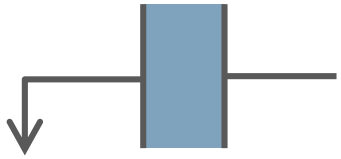


$V_T > V_B$

DRAM Cell – Capacitor

Empty State

Logical “0”

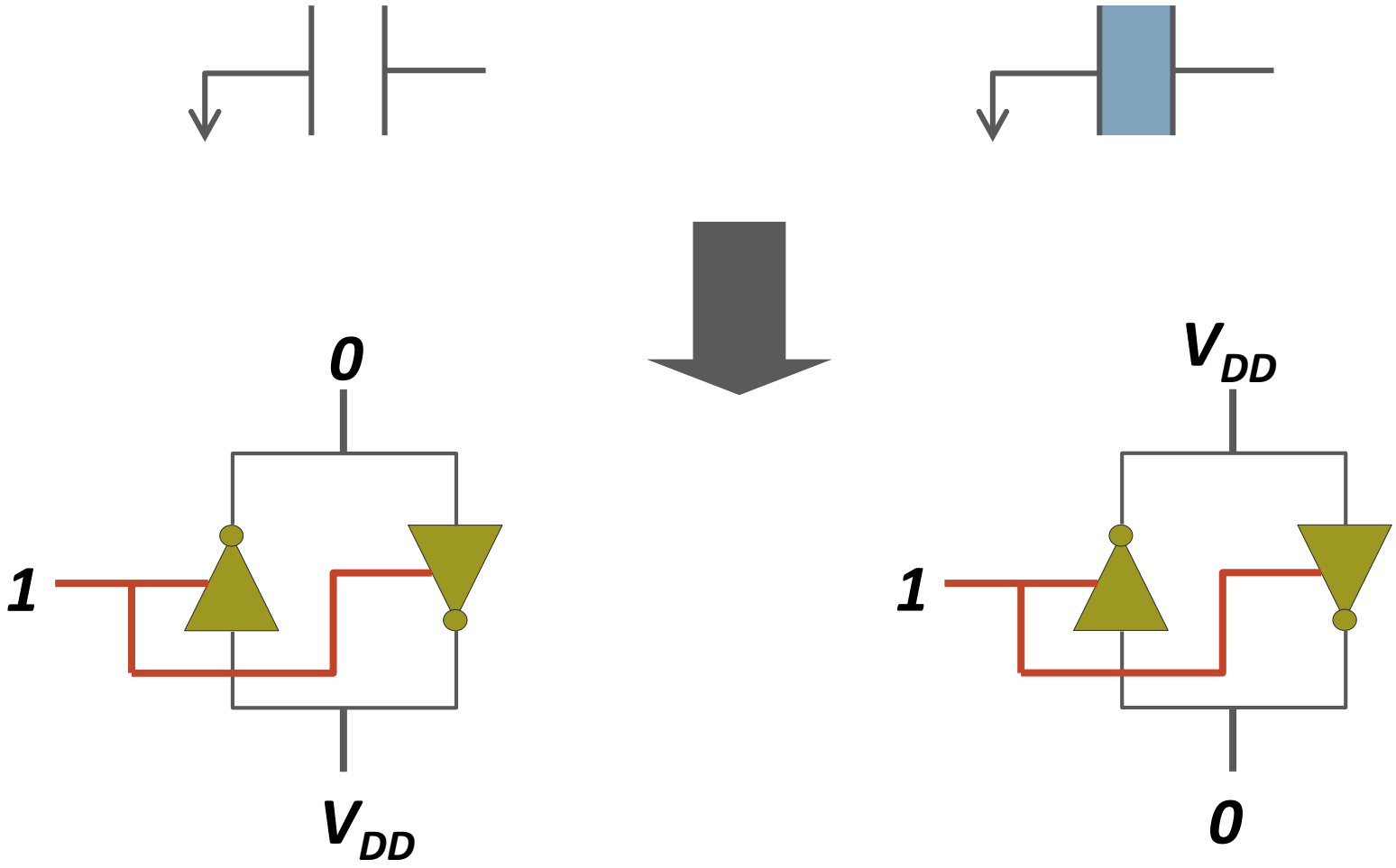


Fully Charged State

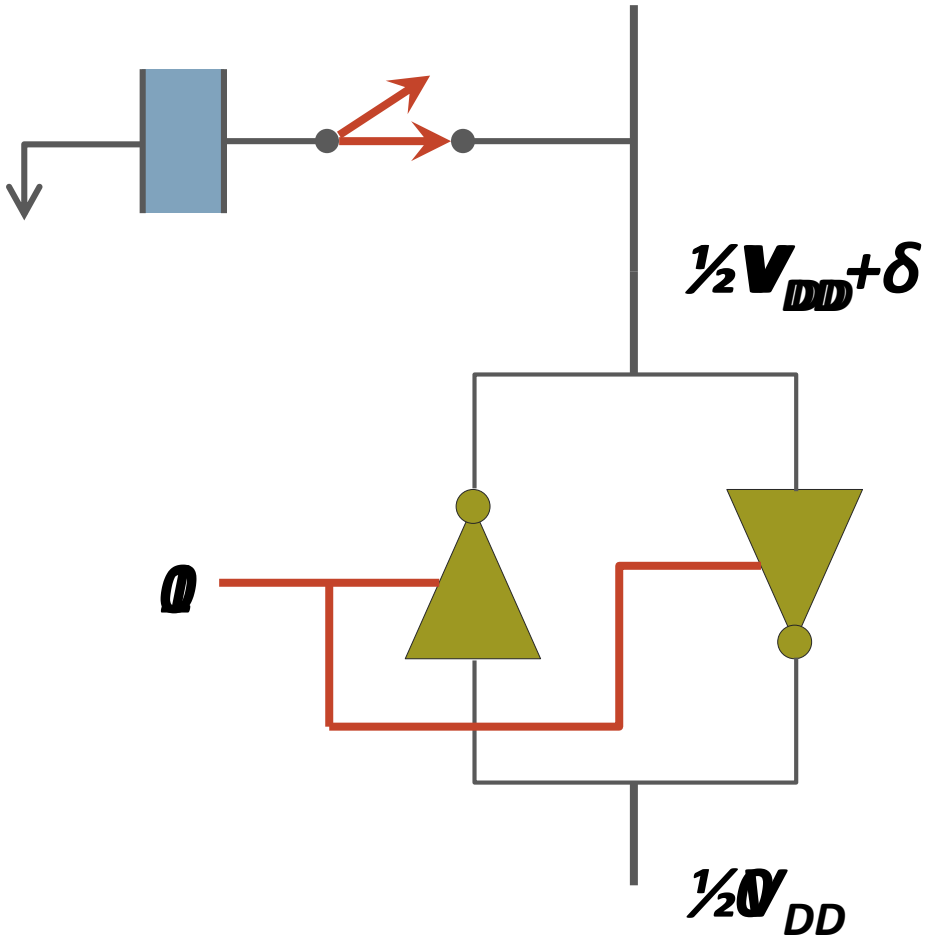
Logical “1”

1. Small – Cannot drive circuits
2. Reading destroys the state

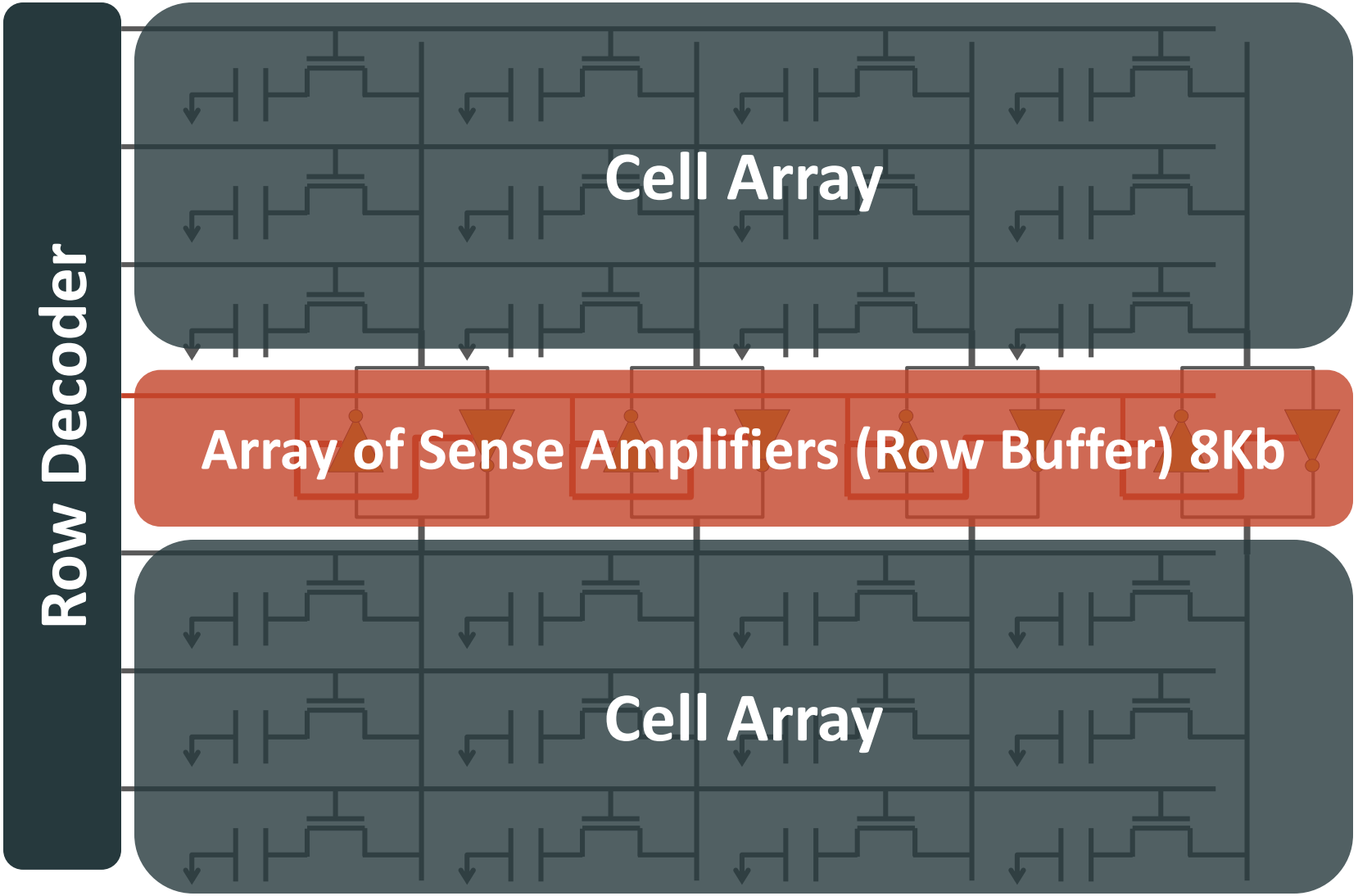
Capacitor to Sense Amplifier



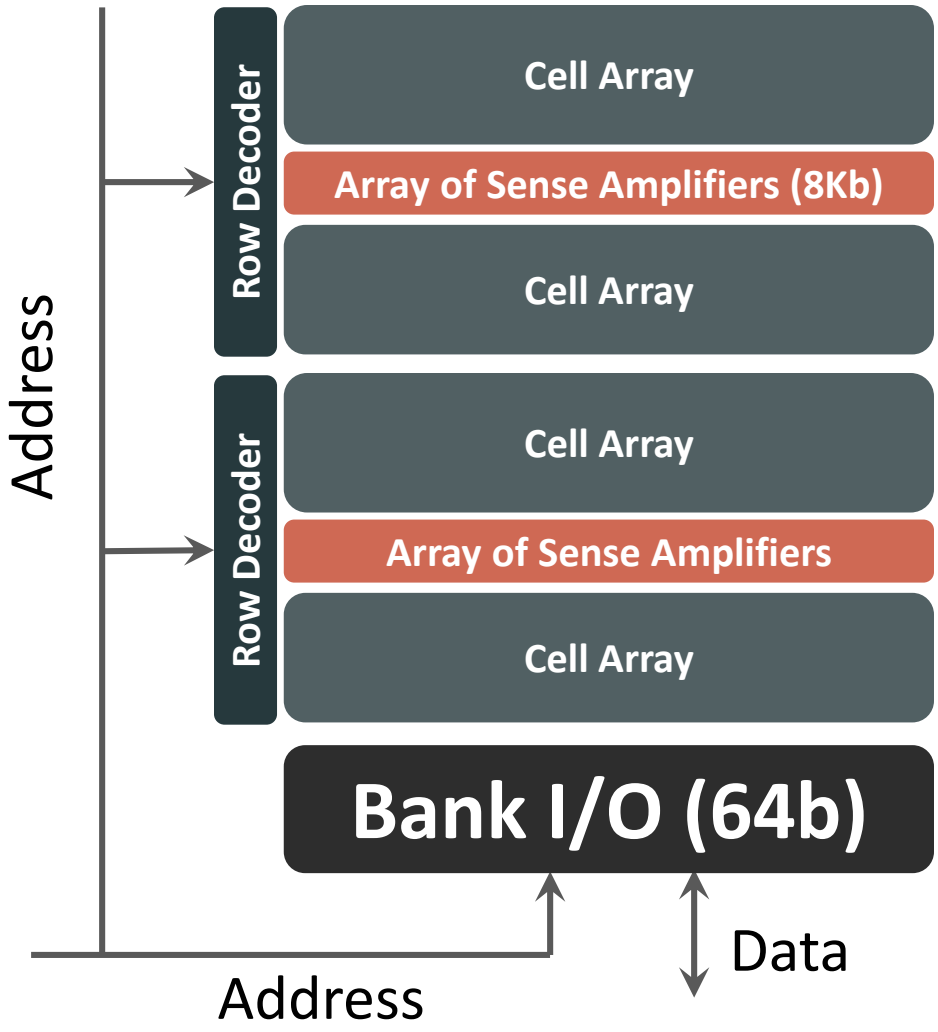
DRAM Cell Operation



DRAM Subarray – Building Block for DRAM Chip

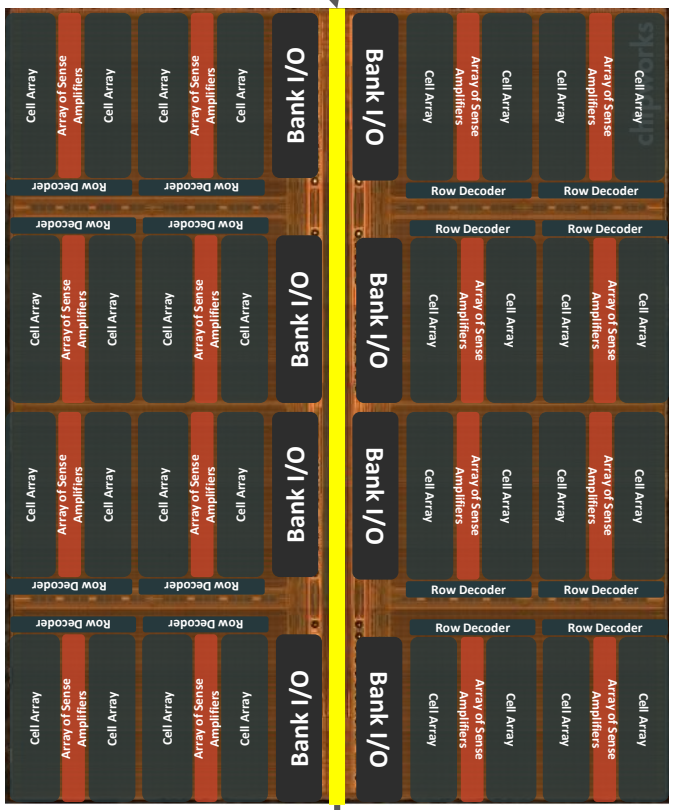


DRAM Bank


Row Decoder
Cell Array
Array of Sense Amplifiers (8Kb)
Cell Array
Row Decoder
Cell Array
Array of Sense Amplifiers
Cell Array
Address
Bank I/O (64b)
Data
Address

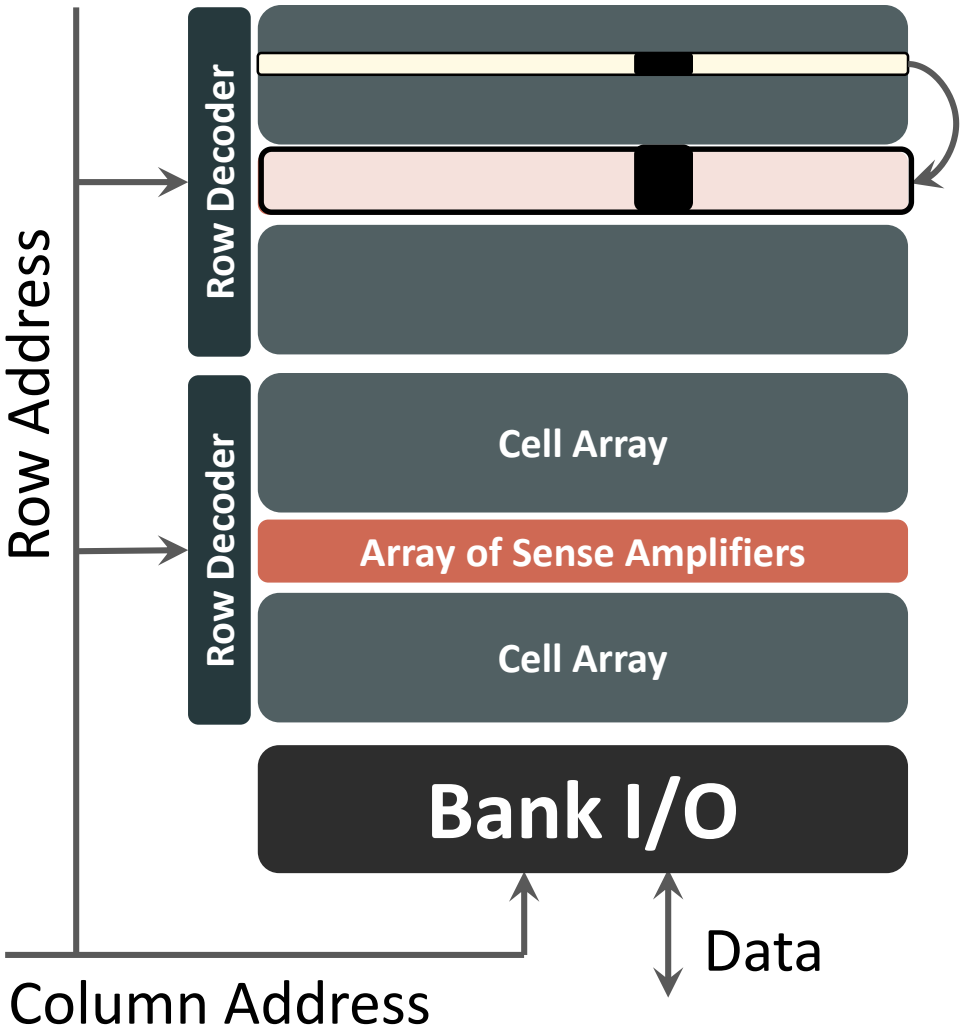
DRAM Chip

Shared internal bus



Memory channel - 8bits

DRAM Operation



1. ACTIVATE Row
2. READ/WRITE Column
3. PRECHARGE

Some More Suggested Readings

Some Key Readings on DRAM (I)

- DRAM Organization and Operation
  - Lee et al., "Tiered-Latency DRAM: A Low Latency and Low Cost DRAM Architecture," HPCA 2013.
    https://people.inf.ethz.ch/omutlu/pub/tldram_hpca13.pdf
  - Kim et al., "A Case for Subarray-Level Parallelism (SALP) in DRAM," ISCA 2012.
    https://people.inf.ethz.ch/omutlu/pub/salp-dram_isca12.pdf
  - Lee et al., "Simultaneous Multi-Layer Access: Improving 3D-Stacked Memory Bandwidth at Low Cost," ACM TACO 2016.
    https://people.inf.ethz.ch/omutlu/pub/smla_high-bandwidth-3d-stacked-memory_taco16.pdf

Some Key Readings on DRAM (II)

- DRAM Refresh
  - Liu et al., "RAIDR: Retention-Aware Intelligent DRAM Refresh," ISCA 2012.
    https://people.inf.ethz.ch/omutlu/pub/raidr-dram-refresh_isca12.pdf
  - Chang et al., "Improving DRAM Performance by Parallelizing Refreshes with Accesses," HPCA 2014.
    https://people.inf.ethz.ch/omutlu/pub/dram-access-refresh-parallelization_hpca14.pdf
  - Patel et al., "The Reach Profiler (REAPER): Enabling the Mitigation of DRAM Retention Failures via Profiling at Aggressive Conditions," ISCA 2017.
    https://people.inf.ethz.ch/omutlu/pub/reaper-dram-retention-profiling-lpddr4_isca17.pdf

Reading on Simulating Main Memory

- How to evaluate future main memory systems?
- An open-source simulator and its brief description

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

Some Key Readings on Memory Control 1

- Mutlu+, "Parallelism-Aware Batch Scheduling: Enhancing both Performance and Fairness of Shared DRAM Systems," ISCA 2008.
  https://people.inf.ethz.ch/omutlu/pub/parbs_isca08.pdf

- Kim et al., "Thread Cluster Memory Scheduling: Exploiting Differences in Memory Access Behavior," MICRO 2010.
  https://people.inf.ethz.ch/omutlu/pub/tcm_micro10.pdf

- Subramanian et al., "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," TPDS 2016.
  https://people.inf.ethz.ch/omutlu/pub/bliss-memory-scheduler_ieee-tpds16.pdf

- Usui et al., "DASH: Deadline-Aware High-Performance Memory Scheduler for Heterogeneous Systems with Hardware Accelerators," TACO 2016.
  https://people.inf.ethz.ch/omutlu/pub/dash_deadline-aware-heterogeneous-memory-scheduler_taco16.pdf

Some Key Readings on Memory Control 2

- Ipek+, "Self Optimizing Memory Controllers: A Reinforcement Learning Approach," ISCA 2008.
  https://people.inf.ethz.ch/omutlu/pub/rlmc_isca08.pdf

- Ebrahimi et al., "Fairness via Source Throttling: A Configurable and High-Performance Fairness Substrate for Multi-Core Memory Systems," ASPLOS 2010.
  https://people.inf.ethz.ch/omutlu/pub/fst_asplos10.pdf

- Subramanian et al., "The Application Slowdown Model: Quantifying and Controlling the Impact of Inter-Application Interference at Shared Caches and Main Memory," MICRO 2015.
  https://people.inf.ethz.ch/omutlu/pub/application-slowdown-model_micro15.pdf

- Lee et al., "Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM," PACT 2015.
  https://people.inf.ethz.ch/omutlu/pub/decoupled-dma_pact15.pdf

More Readings

- To come as we cover the future topics

- Search for "DRAM" or "Memory" in:
  - https://people.inf.ethz.ch/omutlu/projects.htm

Evaluating New Ideas for New (Memory) Architectures

Potential Evaluation Methods

- How do we assess an idea will improve a target metric X?
- A variety of evaluation methods are available:
  - Theoretical proof
  - Analytical modeling/estimation
  - Simulation (at varying degrees of abstraction and accuracy)
  - Prototyping with a real system (e.g., FPGAs)
  - Real implementation

The Difficulty in Architectural Evaluation

- The answer is usually workload dependent
  - E.g., think caching
  - E.g., think pipelining
  - E.g., think any idea we talked about (RAIDR, Mem. Sched., …)
- Workloads change
- System has many design choices and parameters
  - Architect needs to decide many ideas and many parameters for a design
  - Not easy to evaluate all possible combinations!
- System parameters may change

Simulation: The Field of Dreams

Dreaming and Reality

- An architect is in part a dreamer, a creator
- Simulation is a key tool of the architect
- Simulation enables
  - The exploration of many dreams
  - A reality check of the dreams
  - Deciding which dream is better
- Simulation also enables
  - The ability to fool yourself with false dreams

Why High-Level Simulation?

- Problem: RTL simulation is intractable for design space exploration → too time consuming to design and evaluate
  - Especially over a large number of workloads
  - Especially if you want to predict the performance of a good chunk of a workload on a particular design
  - Especially if you want to consider many design choices
    - Cache size, associativity, block size, algorithms
    - Memory control and scheduling algorithms
    - In-order vs. out-of-order execution
    - Reservation station sizes, ld/st queue size, register file size, …
    - …
- Goal: Explore design choices quickly to see their impact on the workloads we are designing the platform for

Different Goals in Simulation

- Explore the design space quickly and see what you want to
  - potentially implement in a next-generation platform
  - propose as the next big idea to advance the state of the art
  - the goal is mainly to see relative effects of design decisions
- Match the behavior of an existing system so that you can
  - debug and verify it at cycle-level accuracy
  - propose small tweaks to the design that can make a difference in performance or energy
  - the goal is very high accuracy
- Other goals in-between:
  - Refine the explored design space without going into a full detailed, cycle-accurate design
  - Gain confidence in your design decisions made by higher-level design space exploration

Tradeoffs in Simulation

- Three metrics to evaluate a simulator
  - Speed
  - Flexibility
  - Accuracy

- Speed: How fast the simulator runs (xIPS, xCPS, slowdown)
- Flexibility: How quickly one can modify the simulator to evaluate different algorithms and design choices?
- Accuracy: How accurate the performance (energy) numbers the simulator generates are vs. a real design (Simulation error)

- The relative importance of these metrics varies depending on where you are in the design process (what your goal is)

Trading Off Speed, Flexibility, Accuracy

- Speed & flexibility affect:
  - How quickly you can make design tradeoffs
- Accuracy affects:
  - How good your design tradeoffs may end up being
  - How fast you can build your simulator (simulator design time)
- Flexibility also affects:
  - How much human effort you need to spend modifying the simulator
- You can trade off between the three to achieve design exploration and decision goals

High-Level Simulation

- Key Idea: Raise the abstraction level of modeling to give up some accuracy to enable speed & flexibility (and quick simulator design)

- Advantage

  + Can still make the right tradeoffs, and can do it quickly

    + All you need is modeling the key high-level factors, you can omit corner case conditions

    + All you need is to get the "relative trends" accurately, not exact performance numbers

- Disadvantage

  -- Opens up the possibility of potentially wrong decisions

    -- How do you ensure you get the "relative trends" accurately?

Simulation as Progressive Refinement

- High-level models (Abstract, C)
- …
- Medium-level models (Less abstract)
- …
- Low-level models (RTL with everything modeled)
- …
- Real design

- As you refine (go down the above list)
  - Abstraction level reduces
  - Accuracy (hopefully) increases (not necessarily, if not careful)
  - Flexibility reduces; Speed likely reduces except for real design
  - You can loop back and fix higher-level models

Making The Best of Architecture

- A good architect is comfortable at all levels of refinement
  - Including the extremes
- A good architect knows when to use what type of simulation
  - And, more generally, what type of evaluation method
- Recall: A variety of evaluation methods are available:
  - Theoretical proof
  - Analytical modeling
  - Simulation (at varying degrees of abstraction and accuracy)
  - Prototyping with a real system (e.g., FPGAs)
  - Real implementation

Ramulator: A Fast and Extensible DRAM Simulator

[IEEE Comp Arch Letters'15]

Ramulator Motivation

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

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

Table 1. Landscape of DRAM-based memory

Ramulator

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

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

Table 3. Comparison of five simulators using two traces

Case Study: Comparison of DRAM Standards

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

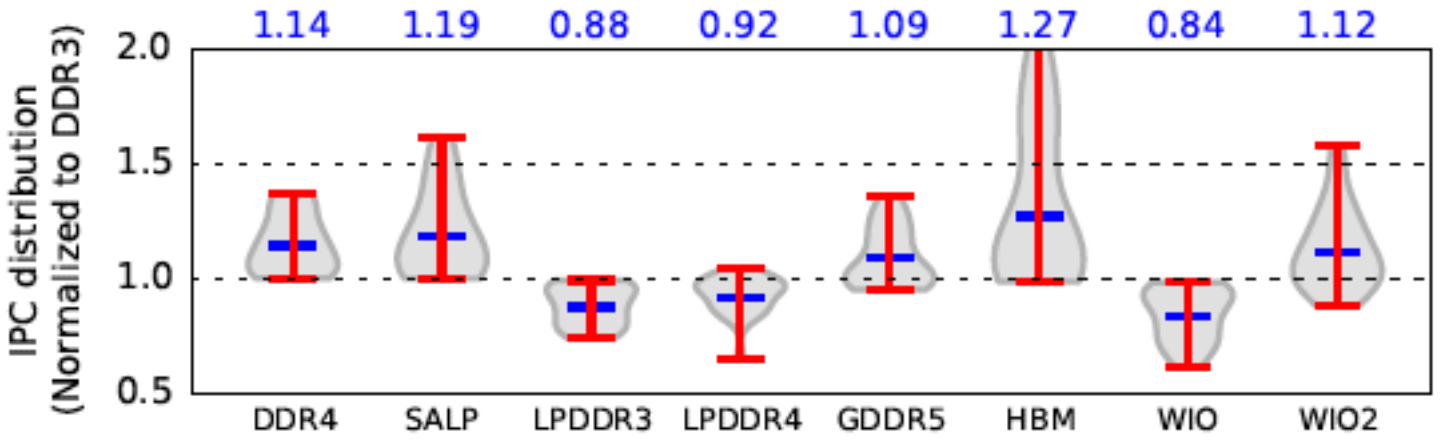


Figure 2. Performance comparison of DRAM standards

Across 22 workloads, simple CPU model

Ramulator Paper and Source Code

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

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

Ramulator: A Fast and Extensible DRAM Simulator

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

Optional Assignment

- Review the Ramulator paper
  - Email me your review
- Download and run Ramulator
  - Compare DDR3, DDR4, SALP, HBM for the libquantum benchmark (provided in Ramulator repository)
  - Email me your report
- This may help you get into memory systems research quickly

End of Backup Slides