Memory Systems
and Memory-Centric Computing Systems
Lecture 1c: Main Memory and DRAM Basics

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
12 June 2019
TU Wien Fast Course 2019
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
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)

Reading: PCM As Main Memory

Reading: More on PCM As Main Memory


Phase-Change Technology and the Future of Main Memory
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:
    - “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
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


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

  - [http://dl.acm.org/citation.cfm?id=366800](http://dl.acm.org/citation.cfm?id=366800)

Cache in 1962 (Bloom, Cohen, Porter)
A Modern Memory Hierarchy

Memory Abstraction

- **Register File**
  - 32 words, sub-nsec

- **L1 cache**
  - ~32 KB, ~nsec

- **L2 cache**
  - 512 KB ~ 1MB, many nsec

- **L3 cache**
  - ..... (not fully visible)

- **Main memory (DRAM)**
  - GB, ~100 nsec

- **Swap Disk**
  - 100 GB, ~10 msec

- **Manual/Compiler**
  - Register spilling

- **Automatic HW cache management**

- **Automatic demand paging**
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

A DRAM Chip consists of multiple of these banks, sharing Address, Data, and Command Buses.
Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)

Row decoder

Columns

Rows

Row 1

Row Buffer

CONFLICT!

Column mux

Data

Column address 85

Row address 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

Readings on DRAM

- **DRAM Organization and Operation Basics**

- **DRAM Refresh Basics**
The DRAM Subsystem
The Top Down View
DRAM Subsystem Organization

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column
The DRAM subsystem

“Channel”

DIMM (Dual in-line memory module)

Processor

Memory channel

Memory channel
Breaking down a DIMM

DIMM (Dual in-line memory module)

Side view

Front of DIMM

Back of DIMM

SIDE

4.00
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)

Addr/Cmd

CS <0:1>

Data <0:63>

Memory channel
Breaking down a Rank

Rank 0

Chip 0

Chip 1

... Chip 7

Data <0:63>
Breaking down a Chip
Breaking down a Bank

Diagram showing the layout of a bank with rows and columns, and a row buffer.
DRAM Subsystem Organization

- Channel
- DIMM
- Rank
- Chip
- Bank
- Row/Column
Example: Transferring a cache block

Physical memory space

0xFFFF...F

0x40

0x00

64B cache block

Mapped to

DIMM 0

Rank 0

Channel 0

0x00

0x40

0xFFFF...F
Example: Transferring a cache block

Physical memory space

0xFFFF...F

... 0x40 0x00

64B cache block

Chip 0

<0:7>

<8:15>

<56:63>

Rank 0

... Chip 1 Chip 7

Data <0:63>
Example: Transferring a cache block

Physical memory space

0xFFFF...F

64B cache block

0x00

0x40

Row 0
Col 0

Chip 0

Chip 1

Chip 7

Rank 0

<0:7>

<8:15>

<56:63>

Data <0:63>
Example: Transferring a cache block

Physical memory space

0xFFFF...F

0x00

0x40

0x00

64B cache block

Row 0
Col 0

Chip 0

Chip 1

Chip 7

Rank 0

Data <0:63>

8B

8B

8B
Example: Transferring a cache block

Physical memory space

Chip 0  Chip 1  Chip 7

Row 0  Col 1

<0:7>  <8:15>  <56:63>

Data <0:63>

64B cache block
Example: Transferring a cache block

Physical memory space

Rank 0

Chip 0

Chip 1

Chip 7

Data <0:63>

Row 0
Col 1

64B cache block

8B

8B

8B
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**
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

Before: No Overlapping
Assuming accesses to different DRAM rows

After: Overlapped Accesses
Assuming no bank conflicts
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

<table>
<thead>
<tr>
<th>Row (14 bits)</th>
<th>Bank (3 bits)</th>
<th>Column (11 bits)</th>
<th>Byte in bus (3 bits)</th>
</tr>
</thead>
</table>

- Accesses to consecutive cache blocks can be serviced in parallel

- Cache block interleaving
  - Consecutive cache block addresses in consecutive banks
  - 64 byte cache blocks

<table>
<thead>
<tr>
<th>Row (14 bits)</th>
<th>High Column</th>
<th>Bank (3 bits)</th>
<th>Low Col.</th>
<th>Byte in bus (3 bits)</th>
</tr>
</thead>
</table>

  8 bits

  3 bits
Bank Mapping Randomization

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

- Reading:
### Address Mapping (Multiple Channels)

<table>
<thead>
<tr>
<th>C</th>
<th>Row (14 bits)</th>
<th>Bank (3 bits)</th>
<th>Column (11 bits)</th>
<th>Byte in bus (3 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Row (14 bits)</td>
<td>C</td>
<td>Bank (3 bits)</td>
<td>Column (11 bits)</td>
</tr>
<tr>
<td></td>
<td>Row (14 bits)</td>
<td>Bank (3 bits)</td>
<td>C</td>
<td>Column (11 bits)</td>
</tr>
<tr>
<td></td>
<td>Row (14 bits)</td>
<td>Bank (3 bits)</td>
<td>Column (11 bits)</td>
<td>C</td>
</tr>
</tbody>
</table>

### Where are consecutive cache blocks?

<table>
<thead>
<tr>
<th>C</th>
<th>Row (14 bits)</th>
<th>High Column</th>
<th>Bank (3 bits)</th>
<th>Low Col.</th>
<th>Byte in bus (3 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Row (14 bits)</td>
<td>C</td>
<td>High Column</td>
<td>Bank (3 bits)</td>
<td>Low Col.</td>
</tr>
<tr>
<td></td>
<td>Row (14 bits)</td>
<td>High Column</td>
<td>C</td>
<td>Bank (3 bits)</td>
<td>Low Col.</td>
</tr>
<tr>
<td></td>
<td>Row (14 bits)</td>
<td>High Column</td>
<td>Bank (3 bits)</td>
<td>C</td>
<td>Low Col.</td>
</tr>
<tr>
<td></td>
<td>Row (14 bits)</td>
<td>High Column</td>
<td>Bank (3 bits)</td>
<td>Low Col.</td>
<td>C</td>
</tr>
</tbody>
</table>
Interaction with Virtual→Physical Mapping

- Operating System influences where an address maps to in DRAM

<table>
<thead>
<tr>
<th>Virtual Page number (52 bits)</th>
<th>Physical Frame number (19 bits)</th>
<th>Row (14 bits)</th>
<th>Bank (3 bits)</th>
<th>Column (11 bits)</th>
<th>Byte in bus (3 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page offset (12 bits)</td>
<td>Page offset (12 bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 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"
Slides (pptx)
More on Reducing Bank Conflicts

- Read Sections 1 through 4 of:

![Diagram of DRAM bank organization]

**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)
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
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

  Slides (pdf)

RAIDR: Retention-Aware Intelligent DRAM Refresh

Jamie Liu  Ben Jaiyen  Richard Veras  Onur Mutlu
Carnegie Mellon University
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)
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

- 64-128ms
- >256ms
- 128-256ms
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"


[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 Systems
and Memory-Centric Computing Systems
Lecture 1c: Main Memory and DRAM Basics

Prof. Onur Mutlu
omutlu@gmail.com
https://people.inf.ethz.ch/omutlu
12 June 2019
TU Wien Fast Course 2019
Backup Slides
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, ...

Another View of the SSD Controller

![SSD System Architecture Diagram](https://arxiv.org/pdf/1711.11427.pdf)

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


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"


[Slides (pptx) (pdf)]
[Source Code]
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"


[Slides (pptx) (pdf)] [Lightning Talk Slides (pptx) (pdf)] 
[Lightning Talk Video]
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)

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDRAM3 (2011) [29]</td>
</tr>
</tbody>
</table>

Table 1. Landscape of DRAM-based memory

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
Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)
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 and act accordingly
## Open vs. Closed Row Policies

<table>
<thead>
<tr>
<th>Policy</th>
<th>First access</th>
<th>Next access</th>
<th>Commands needed for next access</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open row</td>
<td>Row 0</td>
<td>Row 0 (row hit)</td>
<td>Read</td>
</tr>
<tr>
<td>Open row</td>
<td>Row 0</td>
<td>Row 1 (row conflict)</td>
<td>Precharge + Activate Row 1 + Read</td>
</tr>
<tr>
<td>Closed row</td>
<td>Row 0</td>
<td>Row 0 – access in request buffer (row hit)</td>
<td>Read</td>
</tr>
<tr>
<td>Closed row</td>
<td>Row 0</td>
<td>Row 0 – access not in request buffer (row closed)</td>
<td>Activate Row 0 + Read + Precharge</td>
</tr>
<tr>
<td>Closed row</td>
<td>Row 0</td>
<td>Row 1 (row closed)</td>
<td>Activate Row 1 + Read + Precharge</td>
</tr>
</tbody>
</table>
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
  - \( t_{WTR} \): Minimum number of cycles to wait before issuing a read command after a write command is issued
  - \( t_{RC} \): 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

More on DRAM Operation


![Diagram showing three phases of DRAM access](image)

**Figure 5. Three Phases of DRAM Access**

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

<table>
<thead>
<tr>
<th>Phase</th>
<th>Commands</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACT $\rightarrow$ READ</td>
<td>tRCD</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>ACT $\rightarrow$ WRITE</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>ACT $\rightarrow$ PRE</td>
<td>tRAS</td>
<td>37.5ns</td>
</tr>
<tr>
<td>2</td>
<td>READ $\rightarrow$ data</td>
<td>tCL</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>WRITE $\rightarrow$ data</td>
<td>tCWL</td>
<td>11.25ns</td>
</tr>
<tr>
<td>2</td>
<td>data burst</td>
<td>tBL</td>
<td>7.5ns</td>
</tr>
<tr>
<td>3</td>
<td>PRE $\rightarrow$ ACT</td>
<td>tRP</td>
<td>15ns</td>
</tr>
<tr>
<td>1 &amp; 3</td>
<td>ACT $\rightarrow$ ACT</td>
<td>tRC</td>
<td>52.5ns</td>
</tr>
<tr>
<td></td>
<td>(tRAS+tRP)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Why So Many Timing Constraints? (I)

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

<table>
<thead>
<tr>
<th>Category</th>
<th>RowCmd↔RowCmd</th>
<th>RowCmd↔ColCmd</th>
<th>ColCmd↔ColCmd</th>
<th>ColCmd→DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>(tRC)</td>
<td>(tRAS)</td>
<td>(tRP)</td>
<td>(tCCD)</td>
</tr>
<tr>
<td>Commands</td>
<td>A→A</td>
<td>A→P</td>
<td>P→A</td>
<td>R(W)→R(W)</td>
</tr>
<tr>
<td>Scope</td>
<td>Bank</td>
<td>Bank</td>
<td>Bank</td>
<td>Channel</td>
</tr>
<tr>
<td>Value (ns)</td>
<td>(~50)</td>
<td>(~35)</td>
<td>13-15</td>
<td>5-7.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>11-15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>~7.5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>13-15</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10-15</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Phase</th>
<th>Commands</th>
<th>Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ACT → READ, ACT → WRITE</td>
<td>tRCD</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td>ACT → PRE</td>
<td>tRAS</td>
<td>37.5ns</td>
</tr>
<tr>
<td>2</td>
<td>READ → data, WRITE → data</td>
<td>tCL</td>
<td>15ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>tCWL</td>
<td>11.25ns</td>
</tr>
<tr>
<td></td>
<td>data burst</td>
<td>tBL</td>
<td>7.5ns</td>
</tr>
<tr>
<td>3</td>
<td>PRE → ACT</td>
<td>tRP</td>
<td>15ns</td>
</tr>
<tr>
<td>1 &amp; 3</td>
<td>ACT → ACT</td>
<td>tRC</td>
<td>52.5ns</td>
</tr>
<tr>
<td></td>
<td></td>
<td>(tRAS+tRP)</td>
<td></td>
</tr>
</tbody>
</table>
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 design a policy that maximizes performance, QoS, energy-efficiency, ...
  - Too many things to think about
  - Continuously changing workload and system behavior

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

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

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

- **Observation:** Reinforcement learning maps nicely to memory control.

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

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


Figure 4: High-level overview of an RL-based scheduler.
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
Large, robust performance improvements over many human-designed policies
Self Optimizing DRAM Controllers

+ Continuous learning in the presence of changing environment

+ Reduced designer burden in finding a good scheduling policy.

Designer specifies:

1) What system variables might be useful
2) What target to optimize, but not how to optimize it

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

-- Hardware complexity?

-- Design mindset and flow
More on Self-Optimizing DRAM Controllers

Engin Ipek, Onur Mutlu, José F. Martínez, and Rich Caruana,
"Self Optimizing Memory Controllers: A Reinforcement Learning Approach"
Challenge and Opportunity for Future Self-Optimizing (Data-Driven) Computing Architectures
System Architecture Design Today

- Human-driven
  - Humans design the policies (how to do things)

- Many (too) simple, short-sighted policies all over the system

- No automatic data-driven policy learning

- (Almost) no learning: cannot take lessons from past actions

Can we design fundamentally intelligent architectures?
An Intelligent Architecture

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

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

- Automatic data-driven policy learning

- All controllers are intelligent data-driven agents

We need to rethink design (of all controllers)
Simulating Memory
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

<table>
<thead>
<tr>
<th>Segment</th>
<th>DRAM Standards &amp; Architectures</th>
</tr>
</thead>
<tbody>
<tr>
<td>Commodity</td>
<td>DDR3 (2007) [14]; DDR4 (2012) [18]</td>
</tr>
<tr>
<td>Performance</td>
<td>eDRAM [28], [32]; RLDRAM3 (2011) [29]</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Simulator (clang -O3)</th>
<th>Cycles (10^6)</th>
<th>Runtime (sec.)</th>
<th>Req/sec (10^3)</th>
<th>Memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Random</td>
<td>Stream</td>
<td>Random</td>
<td>Stream</td>
</tr>
<tr>
<td>Ramulator</td>
<td>652</td>
<td>411</td>
<td>752</td>
<td>249</td>
</tr>
<tr>
<td>DRAMSim2</td>
<td>645</td>
<td>413</td>
<td>2,030</td>
<td>876</td>
</tr>
<tr>
<td>USIMM</td>
<td>661</td>
<td>409</td>
<td>1,880</td>
<td>750</td>
</tr>
<tr>
<td>DrSim</td>
<td>647</td>
<td>406</td>
<td>18,109</td>
<td>12,984</td>
</tr>
<tr>
<td>NVMain</td>
<td>666</td>
<td>413</td>
<td>6,881</td>
<td>5,023</td>
</tr>
</tbody>
</table>

Table 3. Comparison of five simulators using two traces
Case Study: Comparison of DRAM Standards

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

Across 22 workloads, simple CPU model
Ramulator Paper and Source Code


- Source code is released under the liberal MIT License
  - https://github.com/CMU-SAFARI/ramulator
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 **will** help you get into **memory systems research**
Some More Suggested Readings
Some Key Readings on DRAM (I)

- DRAM Organization and Operation
Some Key Readings on DRAM (II)

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

[Source Code]
https://people.inf.ethz.ch/omutlu/pub/parbs_isca08.pdf


https://people.inf.ethz.ch/omutlu/pub/rlmc_isca08.pdf


Lee et al., “Decoupled Direct Memory Access: Isolating CPU and IO Traffic by Leveraging a Dual-Data-Port DRAM,” PACT 2015.
More Readings

- To come as we cover the future topics

- Search for “DRAM” or “Memory” in:
  - https://people.inf.ethz.ch/omutlu/projects.htm
Inside A DRAM Chip
DRAM Module and Chip
Goals

• Cost
• Latency
• Bandwidth
• Parallelism
• Power
• Energy
• Reliability
• ...

...
DRAM Chip
Sense Amplifier

enable

top

bottom

Inverter
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

\[ \frac{1}{2} V_{DD} + \delta \]

\[ \frac{1}{2} V_{DD} \]
DRAM Subarray – Building Block for DRAM Chip

Row Decoder

Cell Array

Array of Sense Amplifiers (Row Buffer) 8Kb

Cell Array
DRAM Bank

Address

Row Decoder

Array of Sense Amplifiers (8Kb)

Cell Array

Cell Array

Cell Array

Array of Sense Amplifiers

Cell Array

Bank I/O (64b)

Address

Data
DRAM Chip

Shared internal bus

Memory channel - 8bits
DRAM Operation

1. ACTIVATE Row
2. READ/WRITE Column
3. PRECHARGE
End of Backup Slides