CSC 252: Computer Organization
Spring 2018: Lecture 15

Instructor: Yuhao Zhu

Department of Computer Science
University of Rochester

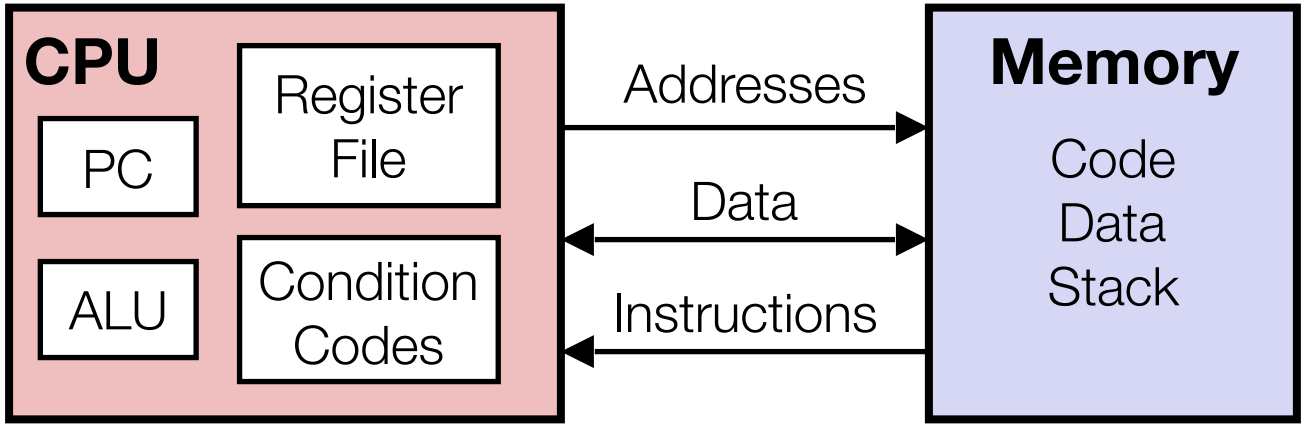
Action Items:

- **Grades for A2 are out**
- **Programming Assignment 4 is out**

Announcement

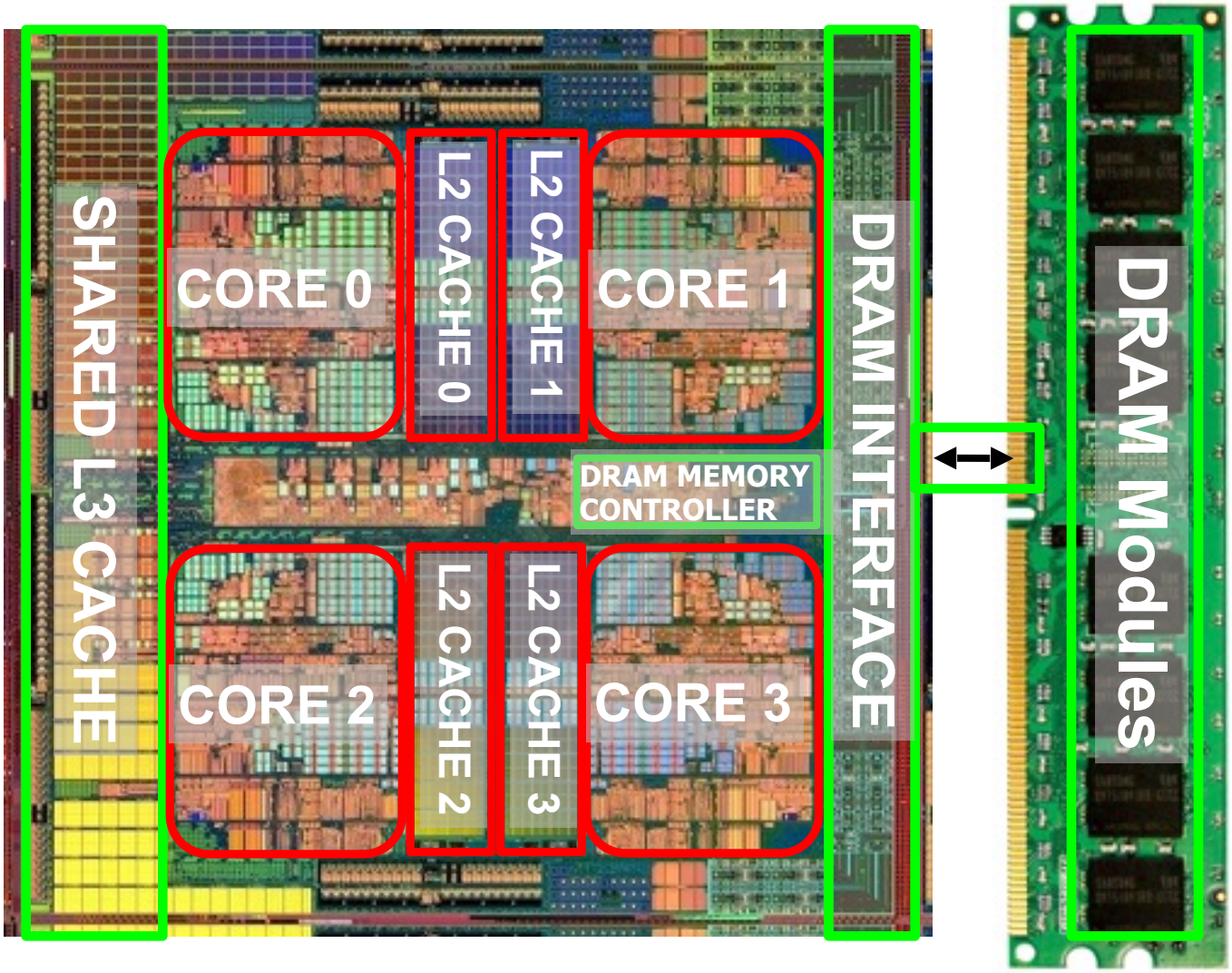
- A2 grades are out. See TAs if you have doubts.
- Programming Assignment 4 is out
  - Trivia due this Thursday, 12:00PM
  - Main assignment due on 11:59pm, **Monday, April 2**.

So far in 252…



- We have been discussing the CPU microarchitecture
  - Single Cycle, sequential implementation
  - Pipeline implementation
  - Resolving data dependency and control dependency
- What about memory?

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

The Problem

- Ideal memory's requirements oppose each other

- Bigger is slower

The Problem

- Ideal memory's requirements oppose each other

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

The Problem

- Ideal memory's requirements oppose each other
- Bigger is slower
  - Bigger → Takes longer to determine the location
- Faster is more expensive

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: Flip-flop vs. SRAM vs. DRAM vs. Disk vs. Tape

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: Flip-flop vs. SRAM vs. DRAM vs. Disk vs. Tape

- Higher bandwidth is more expensive

The Problem

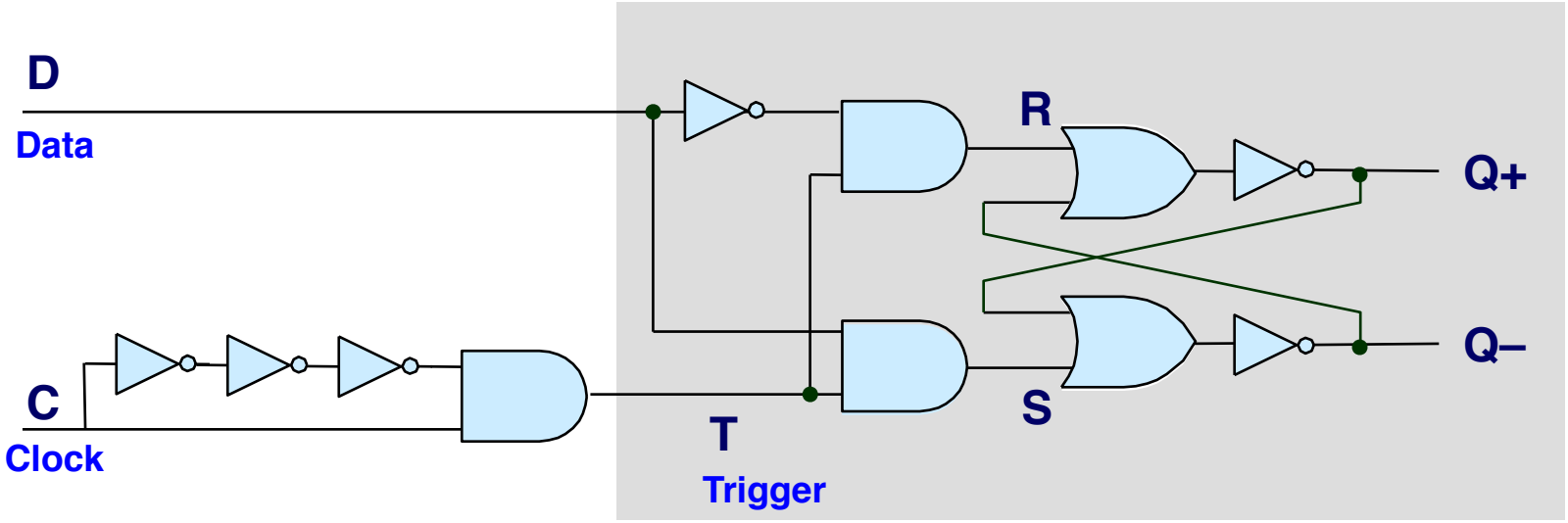
- Ideal memory's requirements oppose each other

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

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

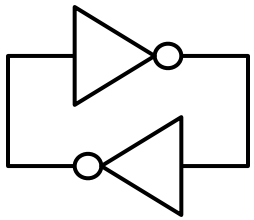
- Higher bandwidth is more expensive
  - Need more banks, more ports, higher frequency, or faster technology

Memory Technology: D Flip-Flop (DFF)



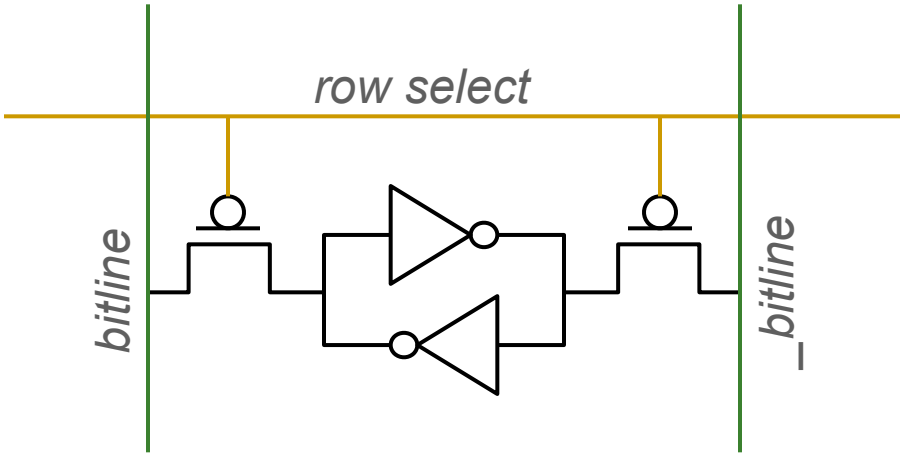
- Very fast
- Very expensive to build
  - 6 NOT gates (2 transistors / gate)
  - 3 AND gates (3 transistors / gate)
  - 2 OR gates (3 transistors / gate)
  - 27 transistors in total for just one bit!!

Memory Technology: SRAM



Memory Technology: SRAM

- Static random access memory
- Random access means you can supply an arbitrary address to the memory and get a value back
- 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
  - 6 transistors in total per bit

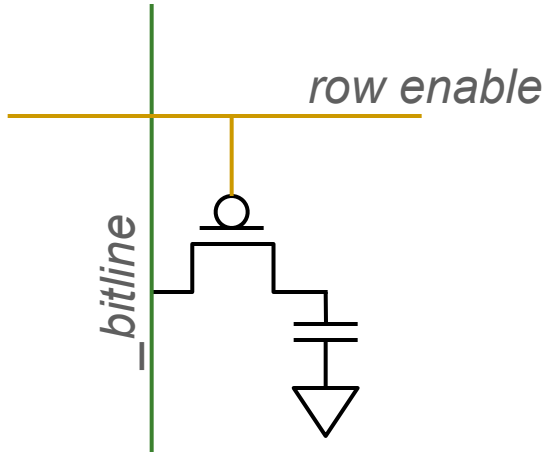


Memory Technology: DRAM

- 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

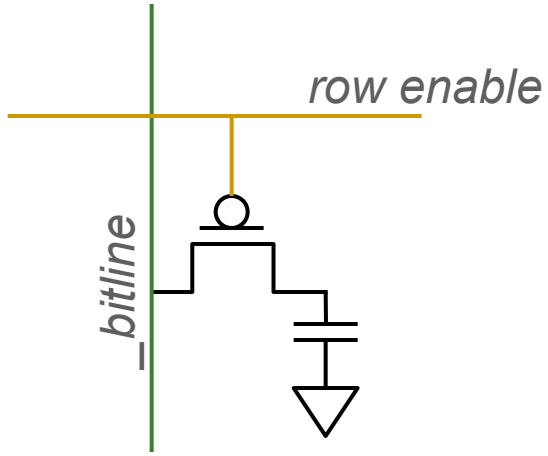
Memory Technology: DRAM

- 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
- Capacitors will leak!



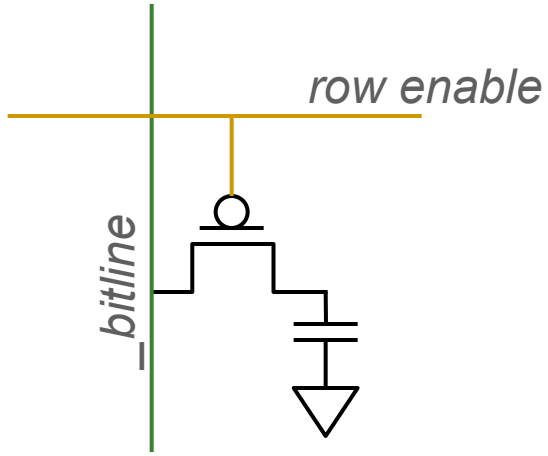
Memory Technology: DRAM

- 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
- Capacitors will leak!
  - DRAM cell loses charge over time



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
- Capacitors will leak!
  - DRAM cell loses charge over time
  - DRAM cell needs to be refreshed



Latch vs. DRAM vs. SRAM

- DFF
  - Fastest
  - Low density (27 transistors per bit)
  - High cost
- SRAM
  - Faster access (no capacitor)
  - Lower density (6 transistors per bit)
  - Higher cost
  - No need for refresh
  - Manufacturing compatible with logic process (no capacitor)
- DRAM
  - Slower access (capacitor)
  - Higher density (1 transistor + 1 capacitor per bit)
  - Lower cost
  - Requires refresh (power, performance, circuitry)
  - Manufacturing requires putting capacitor and logic together

Nonvolatile Memories

Nonvolatile Memories

- DFF, DRAM and SRAM are volatile memories
  - Lose information if powered off.

Nonvolatile Memories

- DFF, DRAM and SRAM are volatile memories
  - Lose information if powered off.
- Nonvolatile memories retain value even if powered off
  - Flash (~ 5 years)
  - Hard Disk (~ 5 years)
  - Tape (~ 15-30 years)
  - DNA (centuries)

Nonvolatile Memories

- DFF, DRAM and SRA
  - Lose information if p
- Nonvolatile memorie
  - Flash (~ 5 years)
  - Hard Disk (~ 5 years)
  - Tape (~ 15-30 years)
  - DNA (centuries)

Rewriting Life

Microsoft Has a Plan to Add DNA Data Storage to Its Cloud

Tech companies think biology may solve a looming data storage problem.

by Antonio Regalado May 22, 2017

Based on early research involving the storage of movies and documents in DNA, Microsoft is developing an apparatus that uses biology to replace tape drives, researchers at the company say.

Computer architects at Microsoft Research say the company has formalized a goal of having an operational storage system based on DNA

Nonvolatile Memories

- DFF, DRAM and SRAM are volatile memories
  - Lose information if powered off.
- Nonvolatile memories retain value even if powered off
  - Flash (~ 5 years)
  - Hard Disk (~ 5 years)
  - Tape (~ 15-30 years)
  - DNA (centuries)
- Uses for Nonvolatile Memories
  - Firmware (BIOS, controllers for disks, network cards, graphics accelerators, security subsystems,…)
  - Files in Smartphones, mp3 players, tablets, laptops
  - Backup

The Problem

- Bigger is slower
  - SRAM, 512 Bytes, sub-nanosec
  - SRAM, KByte~MByte, ~nanosec
  - DRAM, Gigabyte, ~50 nanosec
  - Hard Disk, Terabyte, ~10 millisec
- Faster is more expensive (dollars and chip area)
  - SRAM, < 10$ per Megabyte
  - DRAM, < 1$ per Megabyte
  - Hard Disk < 1$ per Gigabyte
- Other technologies have their place as well
  - PC-RAM, MRAM, RRAM

We want both fast and large Memory

- But we cannot achieve both with a single level of memory

- Idea: Memory Hierarchy
  - Have multiple levels of storage (progressively bigger and slower as the levels are farther from the processor)
  - ensure most of the data the processor needs in the near future is kept in the fast(er) level(s)

- Question: How do we know what kind of data processors would use in the near future?

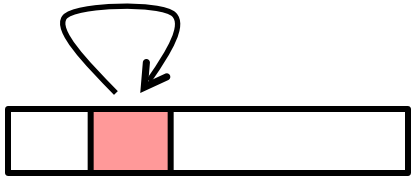
Locality

- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

Locality

- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

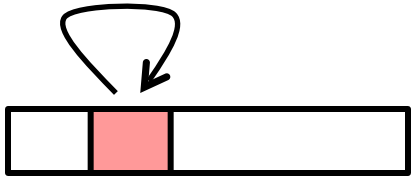
- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



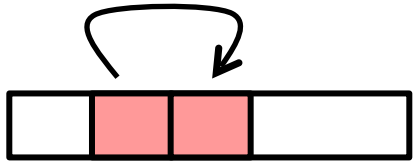
Locality

- Principle of Locality: Programs tend to use data and instructions with addresses near or equal to those they have used recently

- Temporal locality:
  - Recently referenced items are likely to be referenced again in the near future



- Spatial locality:
  - Items with nearby addresses tend to be referenced close together in time



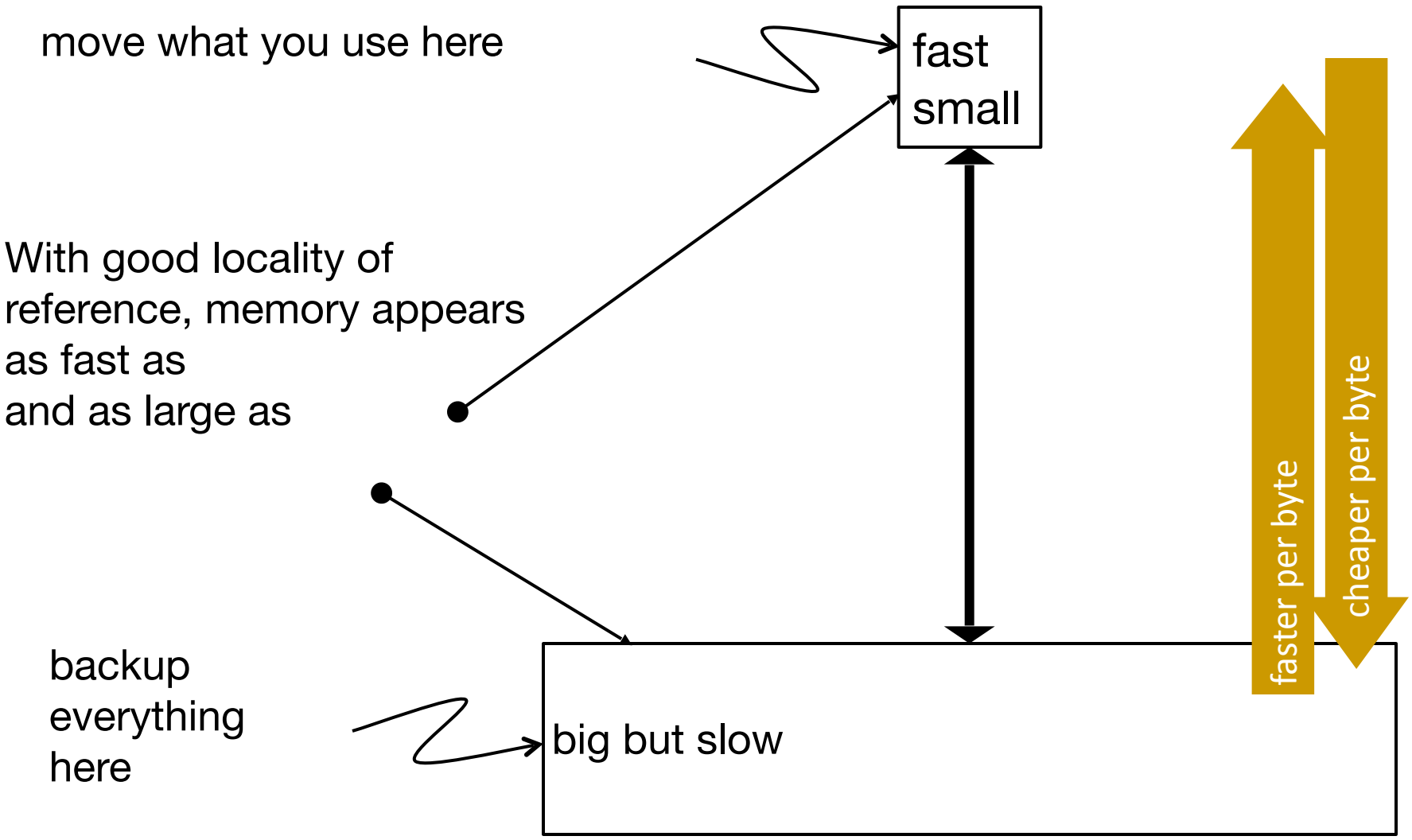
Locality Example

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

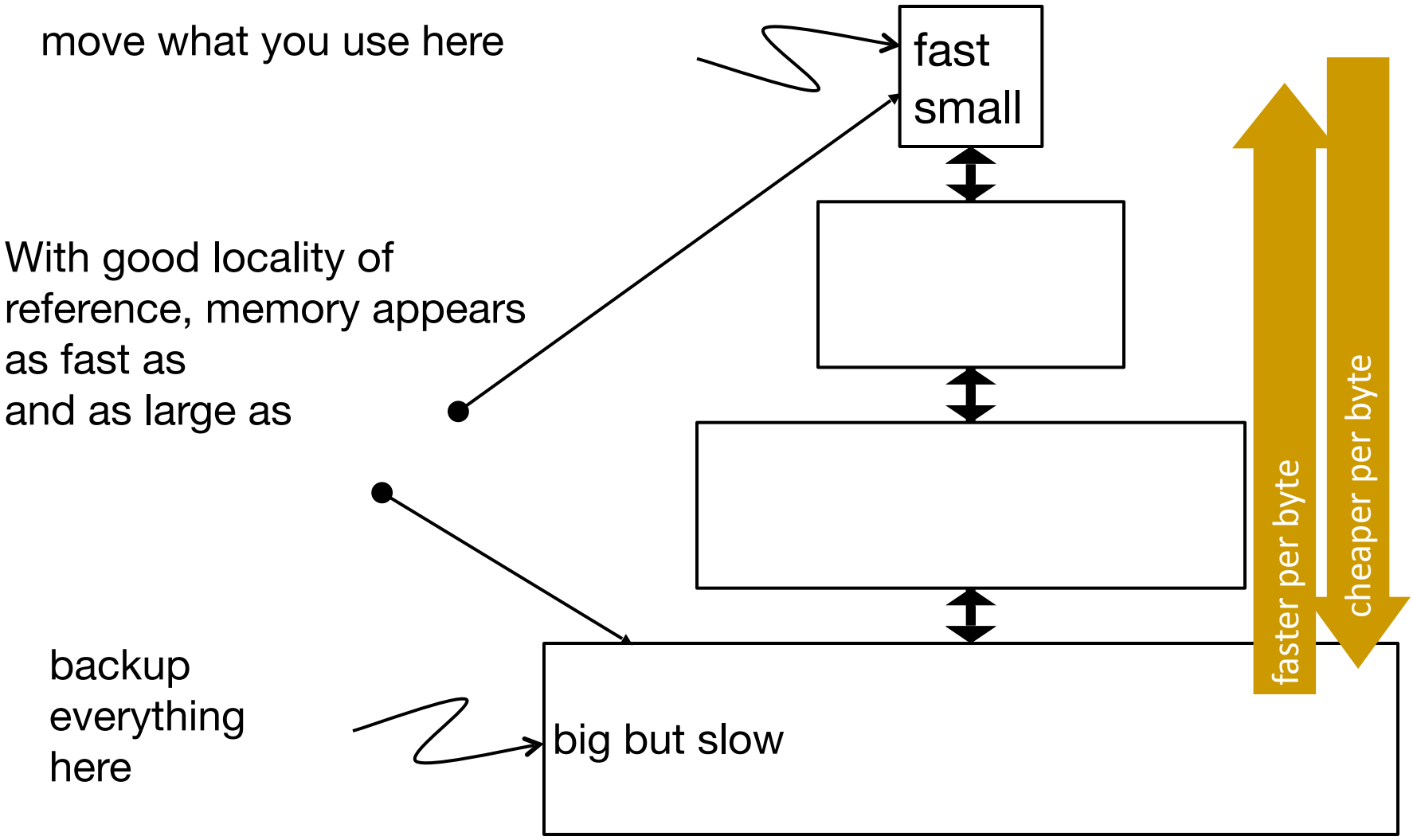
- Data references
  - **Spatial** Locality: Reference array elements in succession (stride-1 reference pattern)
  - **Temporal** Locality: Reference variable sum each iteration.

- Instruction references
  - **Spatial** Locality: Reference instructions in sequence.
  - **Temporal** Locality: Cycle through loop repeatedly.

Memory Hierarchy

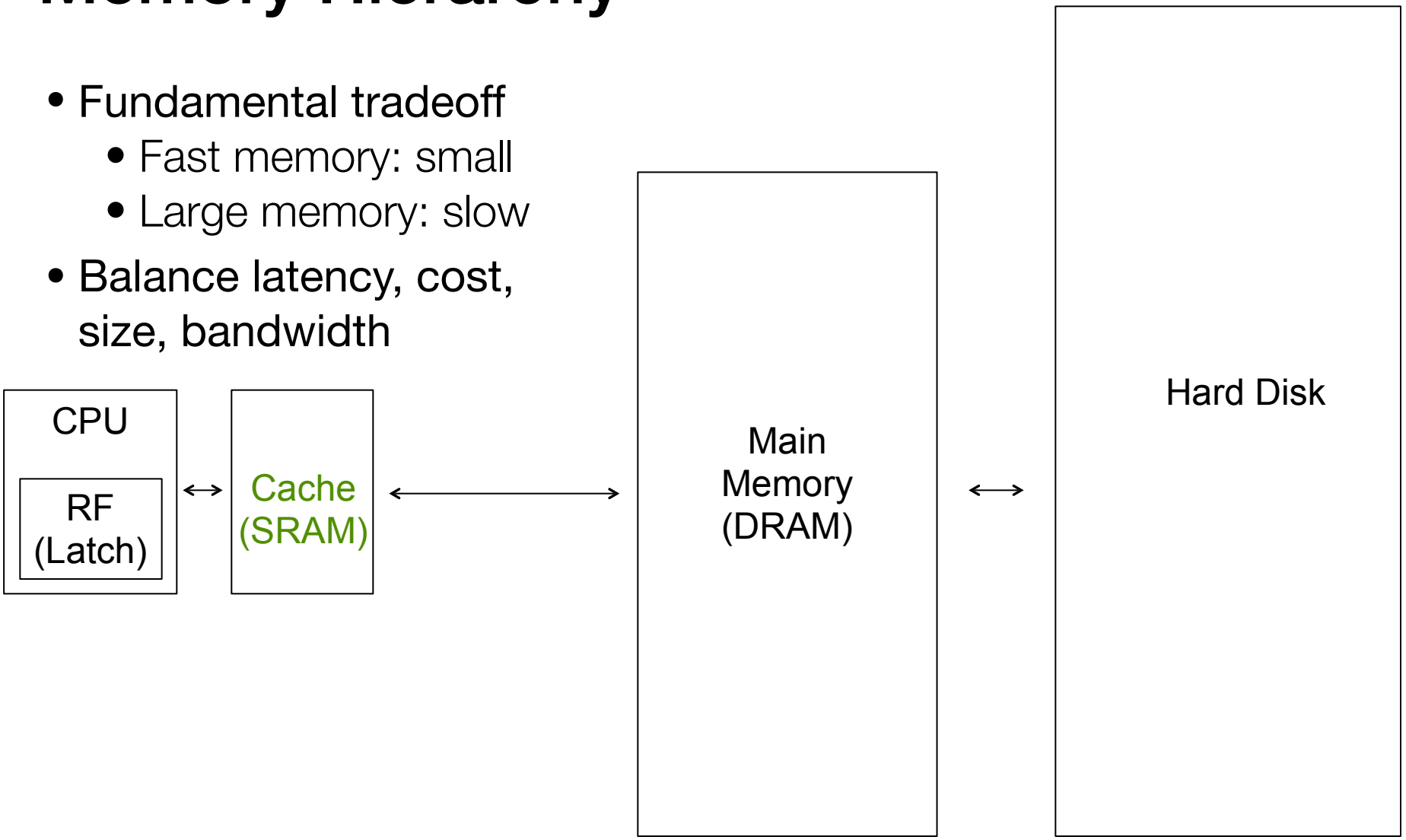


Memory Hierarchy



Memory Hierarchy

- Fundamental tradeoff
  - Fast memory: small
  - Large memory: slow
- Balance latency, cost, size, bandwidth



Caching Basics: Exploit Temporal Locality

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

Caching Basics: Exploit Temporal Locality

- Idea: Store recently accessed data in 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

Caching Basics: Exploit Temporal Locality

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

Caching Basics: Exploit Spatial Locality

- Idea: Store addresses adjacent to the recently accessed one in fast memory (cache)
  - 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

Caching Basics: Exploit Spatial Locality

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

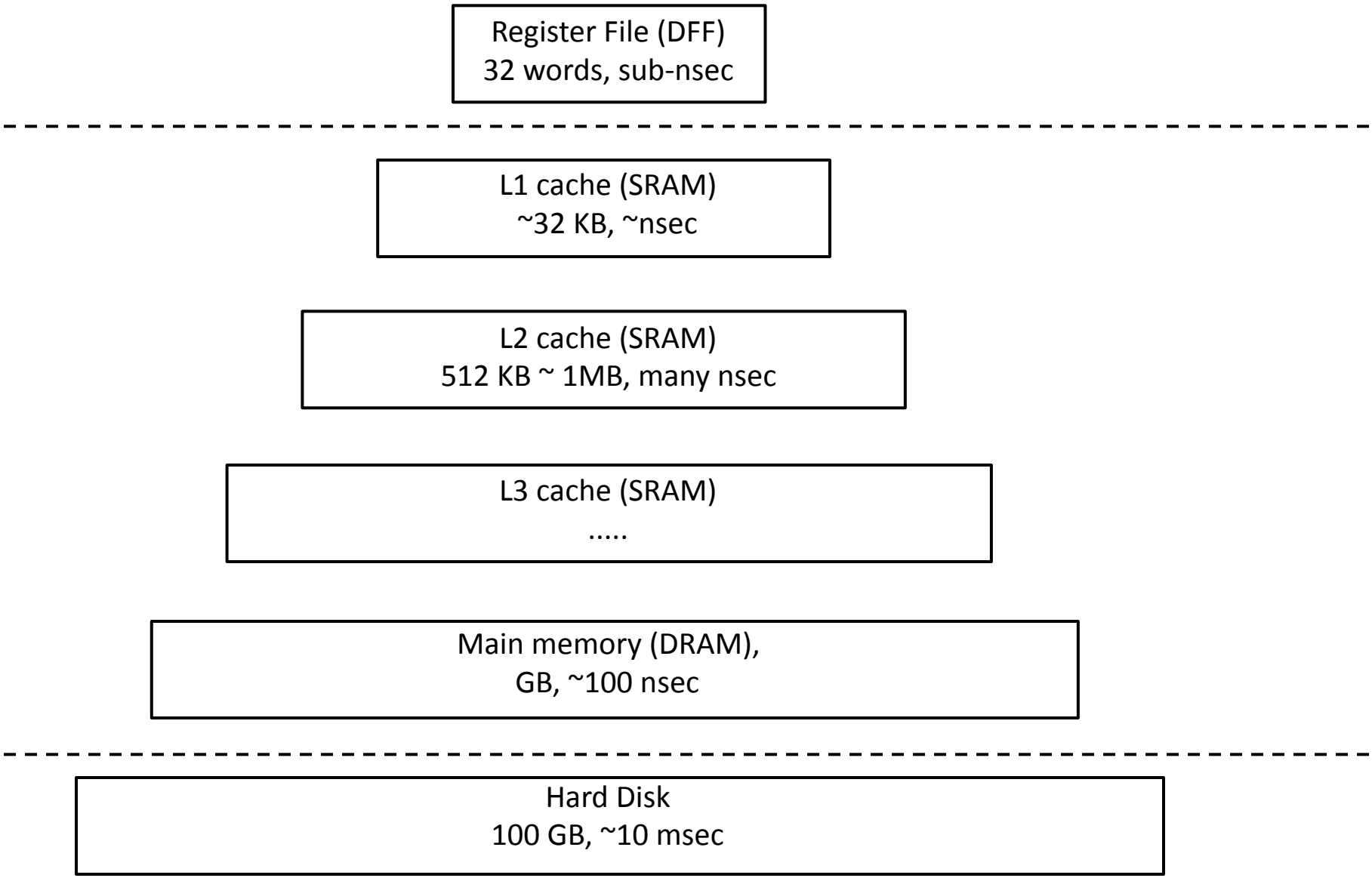
- Spatial locality principle
  - Nearby data in memory will be accessed in the near future
    - E.g., sequential instruction access, array traversal
  - This is what IBM 360/85 implemented
    - 16 Kbyte cache with 64 byte blocks
    - Liptay, "Structural aspects of the System/360 Model 85 II: the cache,"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

The Bookshelf Analogy

- Book in your hand
- Desk
- Bookshelf
- Boxes at home
- Boxes in storage

- Recently-used books tend to stay on desk
  - Comp Org. books, books for classes you are currently taking
  - Until the desk gets full
- Adjacent books in the shelf needed around the same time

A Modern Memory Hierarchy

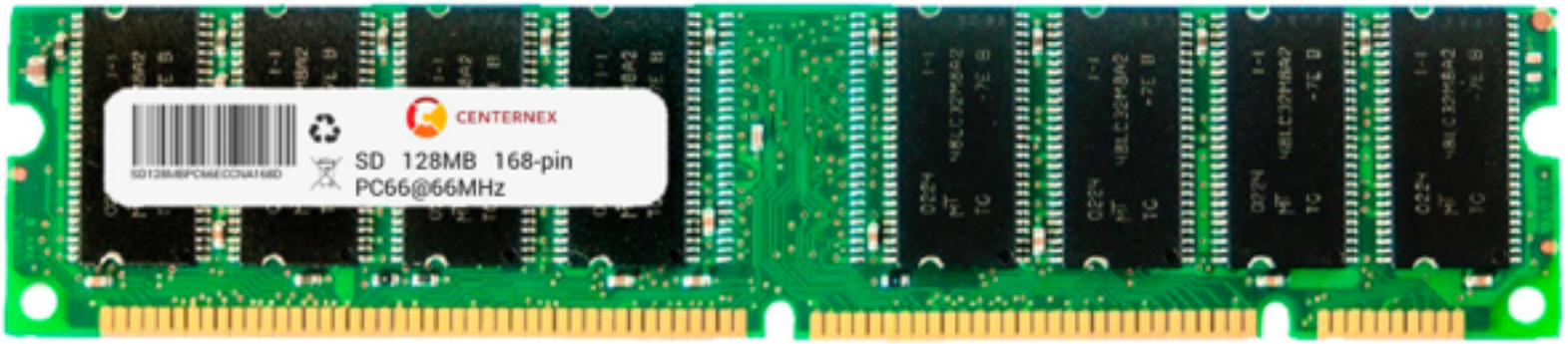


This Module (3 Lectures)

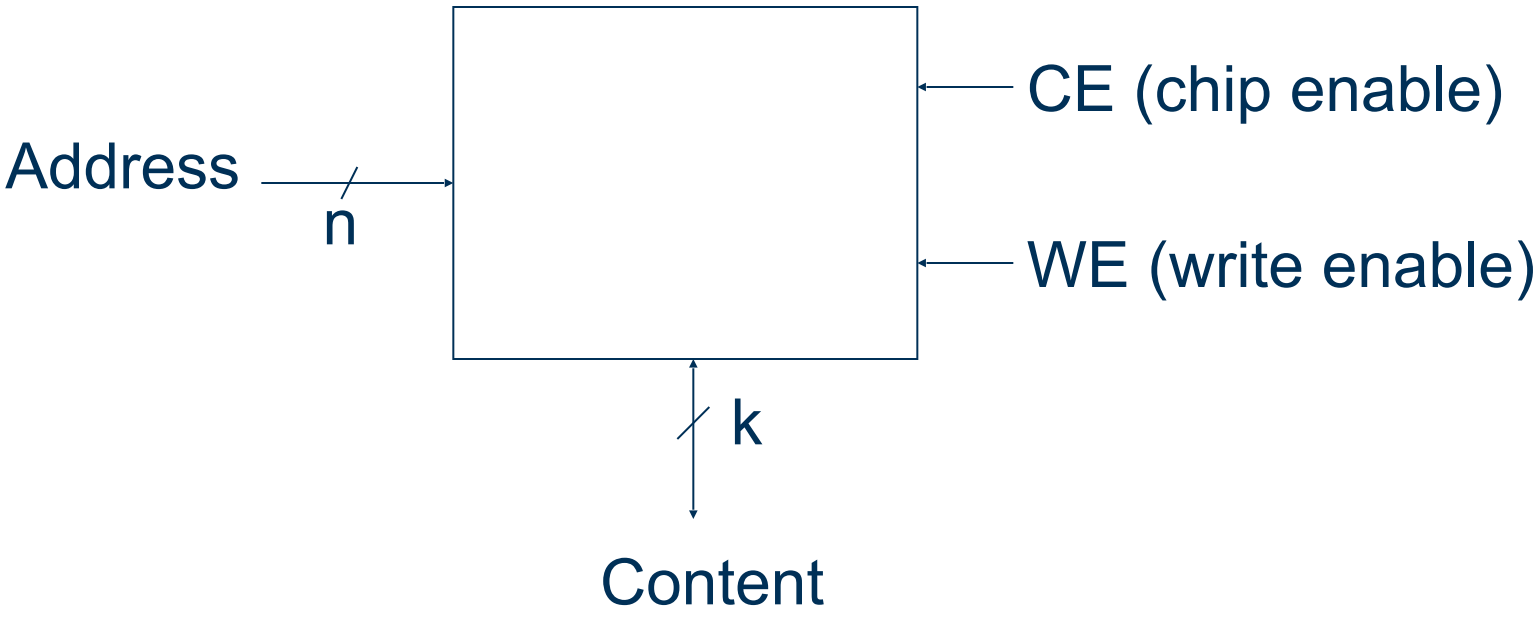
- Memory Hierarchy Overview
  - Trade-offs between different memory technologies
  - Exploiting locality to get the best of all worlds
- SRAM/DRAM Hardware Basics
- Cache
- Memory-oriented Program Optimizations

Random-Access Memory (RAM)

- Key features
  - Basic storage unit is normally a **cell** (**one bit per cell**).
  - A supercell is a collection of cells (e.g., 8, 16)
  - RAM is packaged as a chip, which is a collection of supercells
  - Multiple RAM chips form a memory module
- RAM comes in two varieties:
  - SRAM (Static RAM): implements cache
  - DRAM (Dynamic RAM): implement main memory

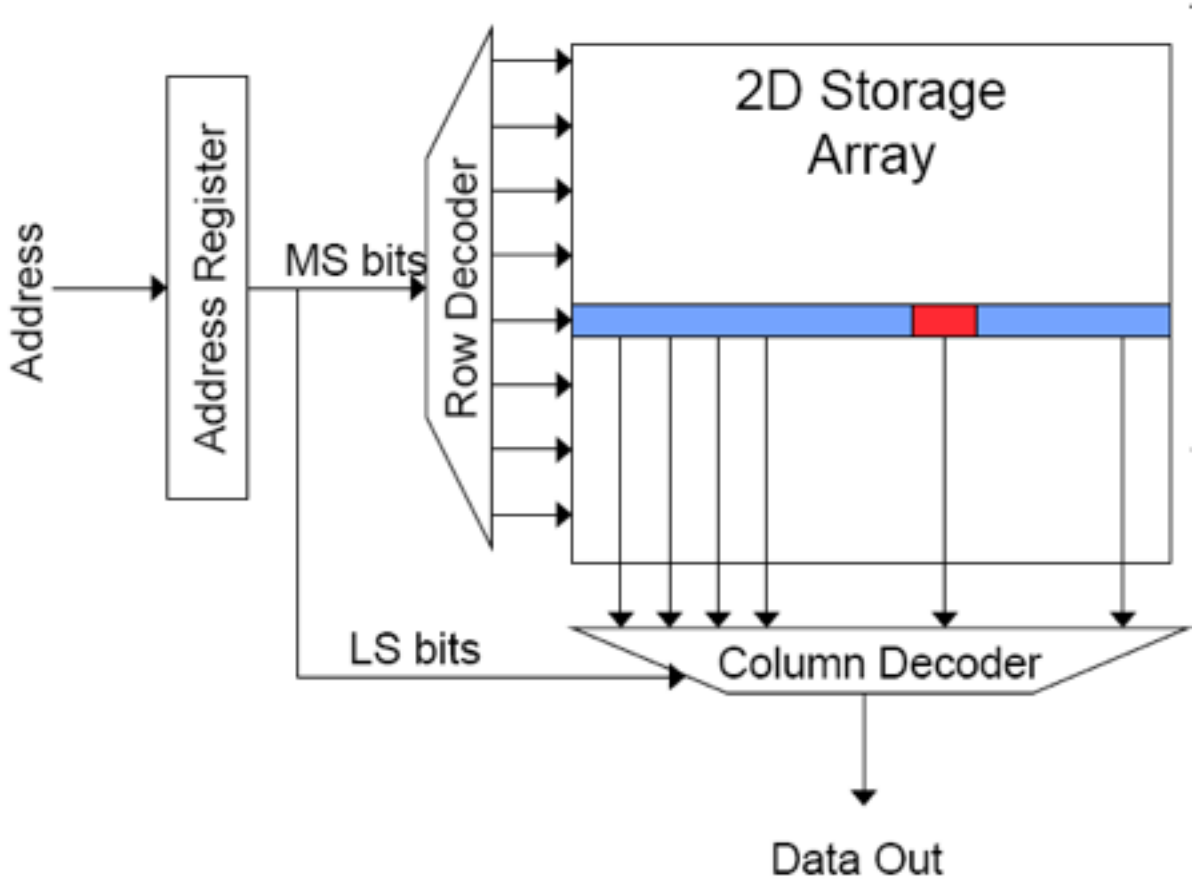


Abstract View of RAM



Hardware Organization of RAM

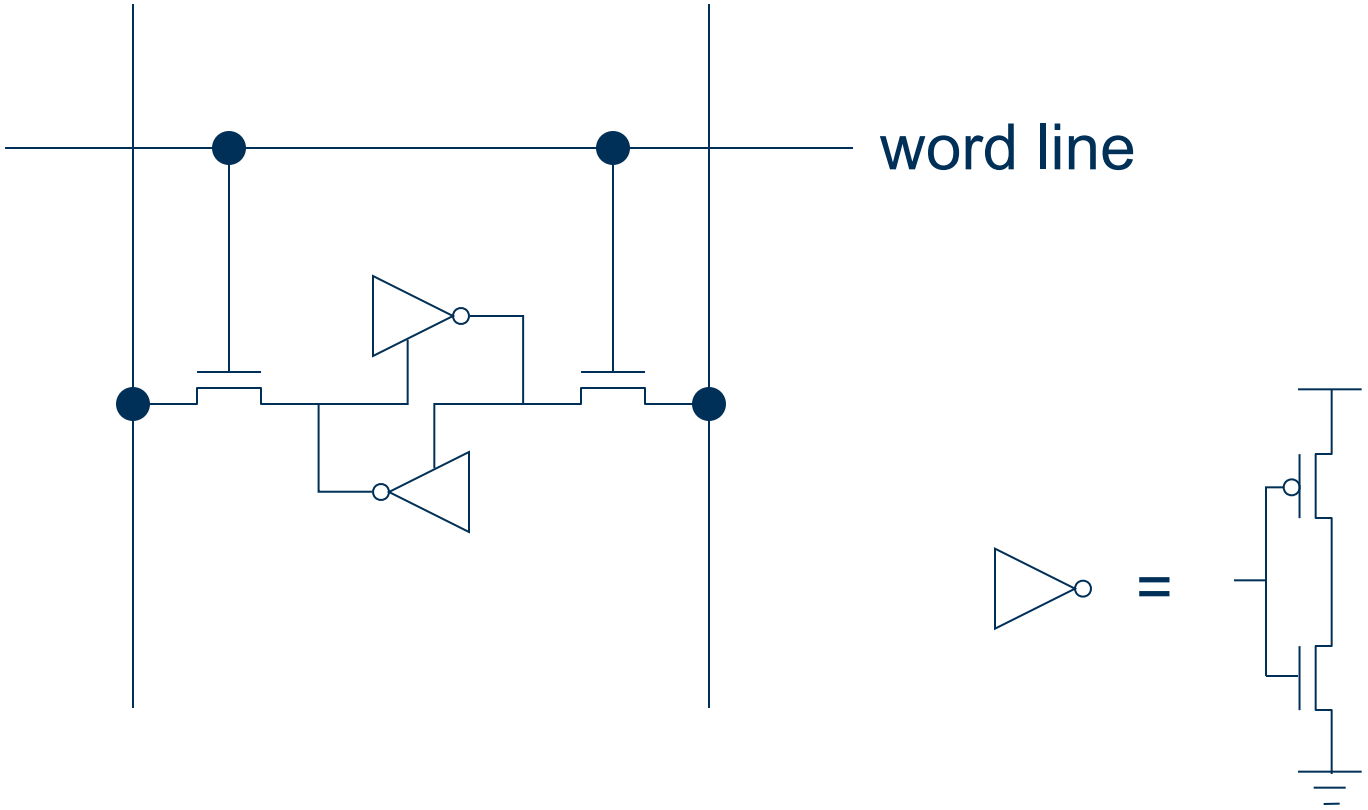
- A RAM is organized as a 2D array
- Address split into two: row address and column address



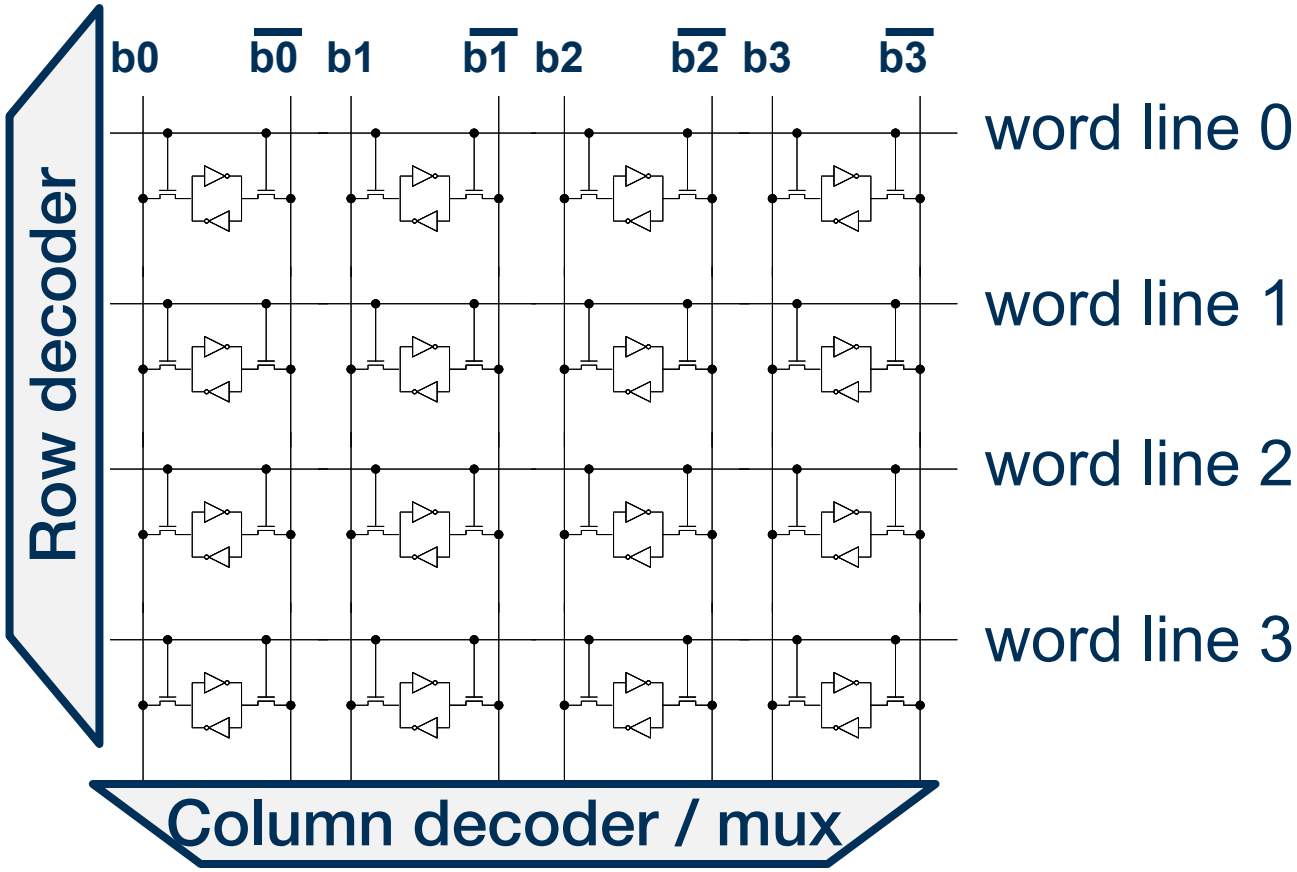
SRAM Cell

SRAM Cell

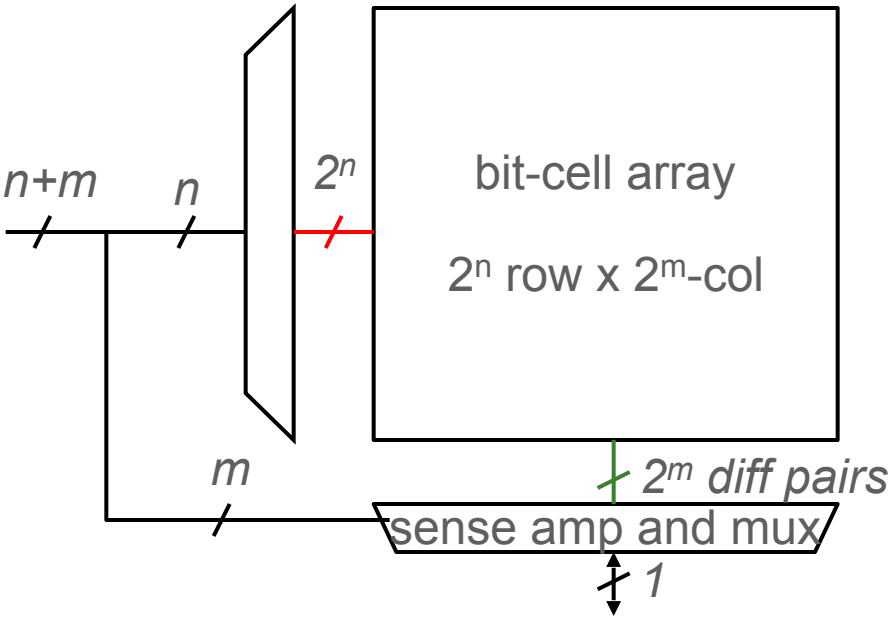
- SRAM
  - Static RAM (holds value while power is on)
  - Densest logic-only memory
  - 6 carefully laid-out transistors



SRAM Array


b0
b0
b1
b1
b2
b2
b3
b3
Row decoder
word line 0
word line 1
word line 2
word line 3
Column decoder / mux

SRAM Array Access



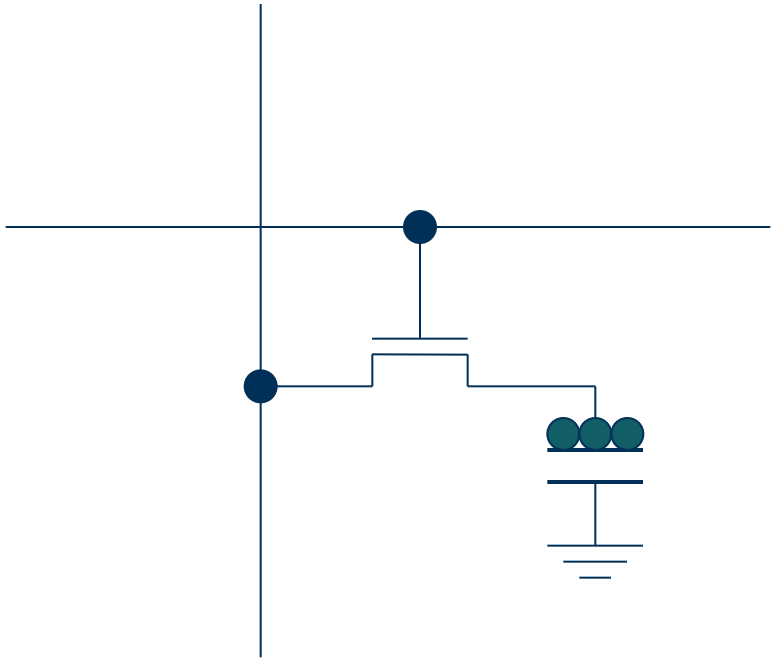
SRAM notes

- SRAMs are very carefully laid out – amortize costs over many bits (8Kb or more usually)

- In large arrays, the latency of access is dominated by the latency of the wires!
  - Therefore, the larger the memory, the slower it is (generally)

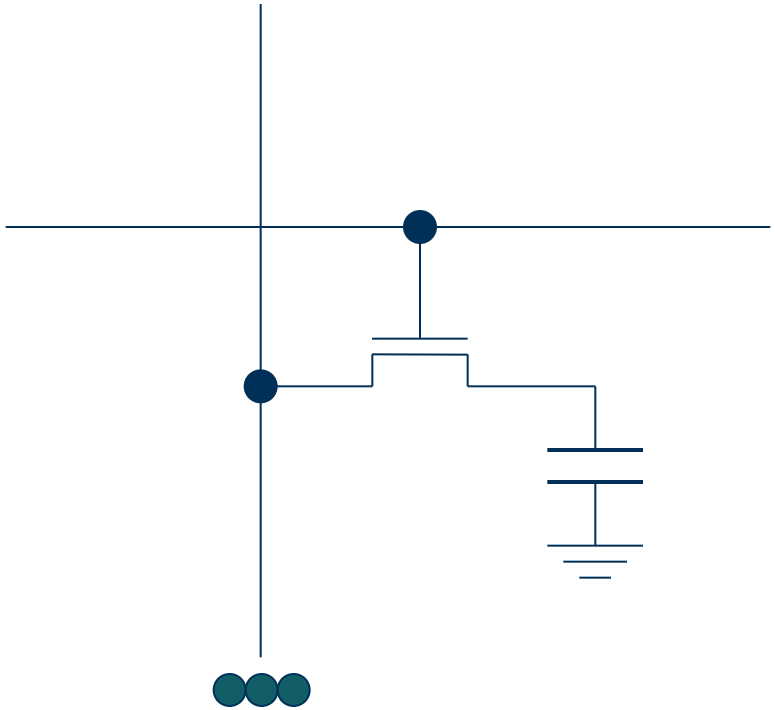
DRAM Cell

- Capacitor holding value leaks, eventually you will lose information (everything turns to 0)
- How do you maintain the values in DRAM?
  - Refresh periodically
  - A major source for power consumption in DRAM



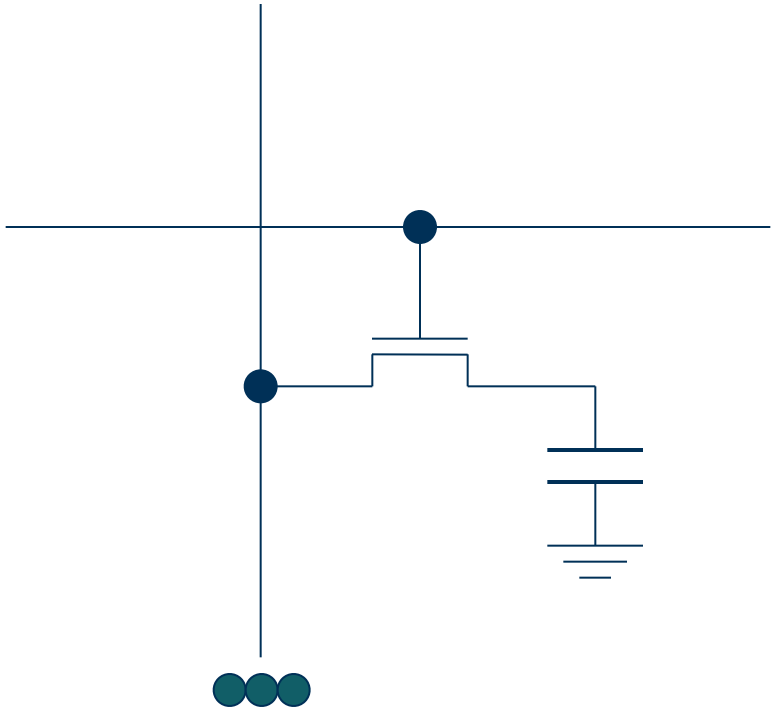
DRAM Cell

- Capacitor holding value leaks, eventually you will lose information (everything turns to 0)
- How do you maintain the values in DRAM?
  - Refresh periodically
  - A major source for power consumption in DRAM

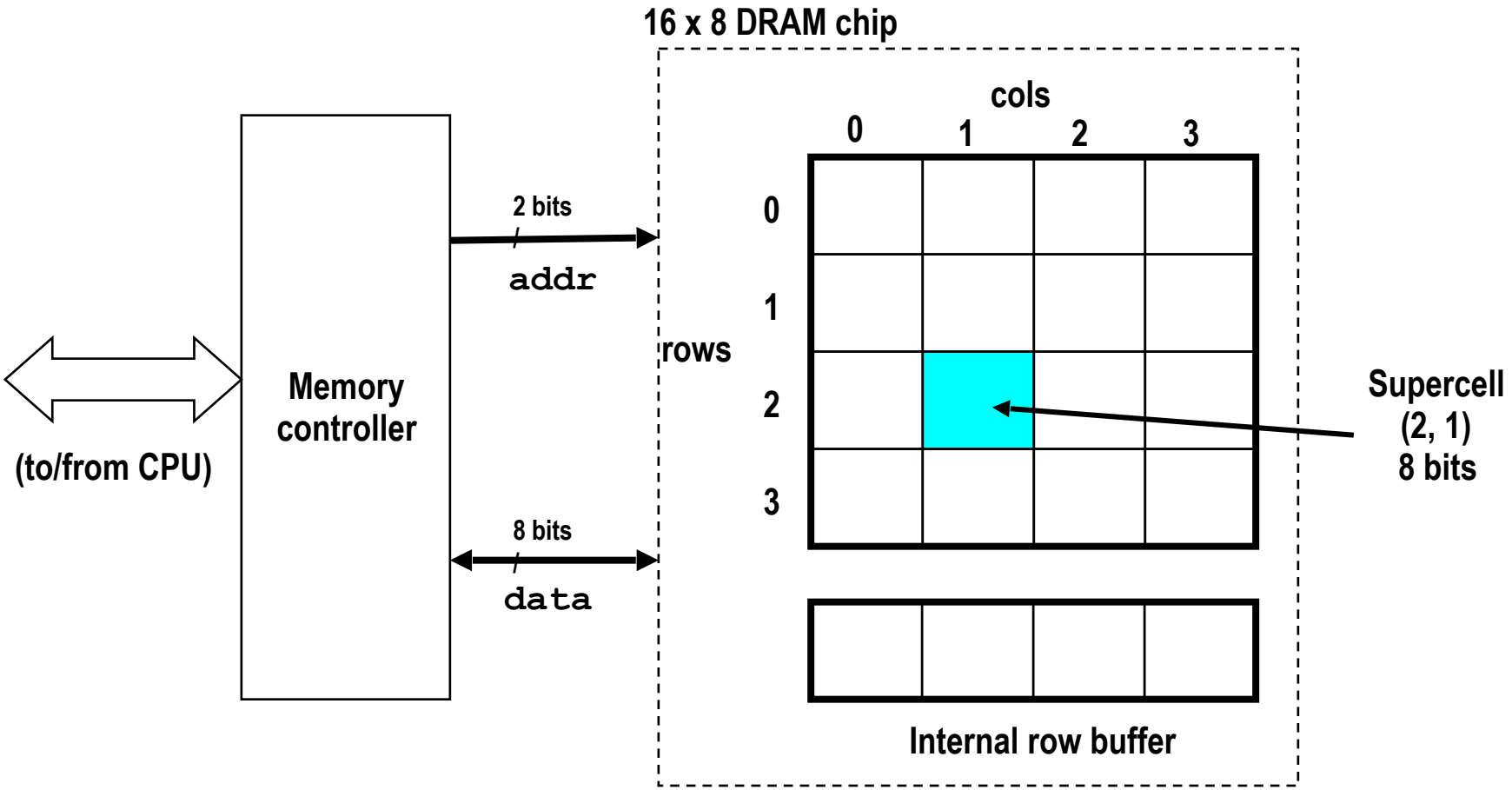


DRAM Cell

- Capacitor holding value leaks, eventually you will lose information (everything turns to 0)
- How do you maintain the values in DRAM?
  - Refresh periodically
  - A major source for power consumption in DRAM



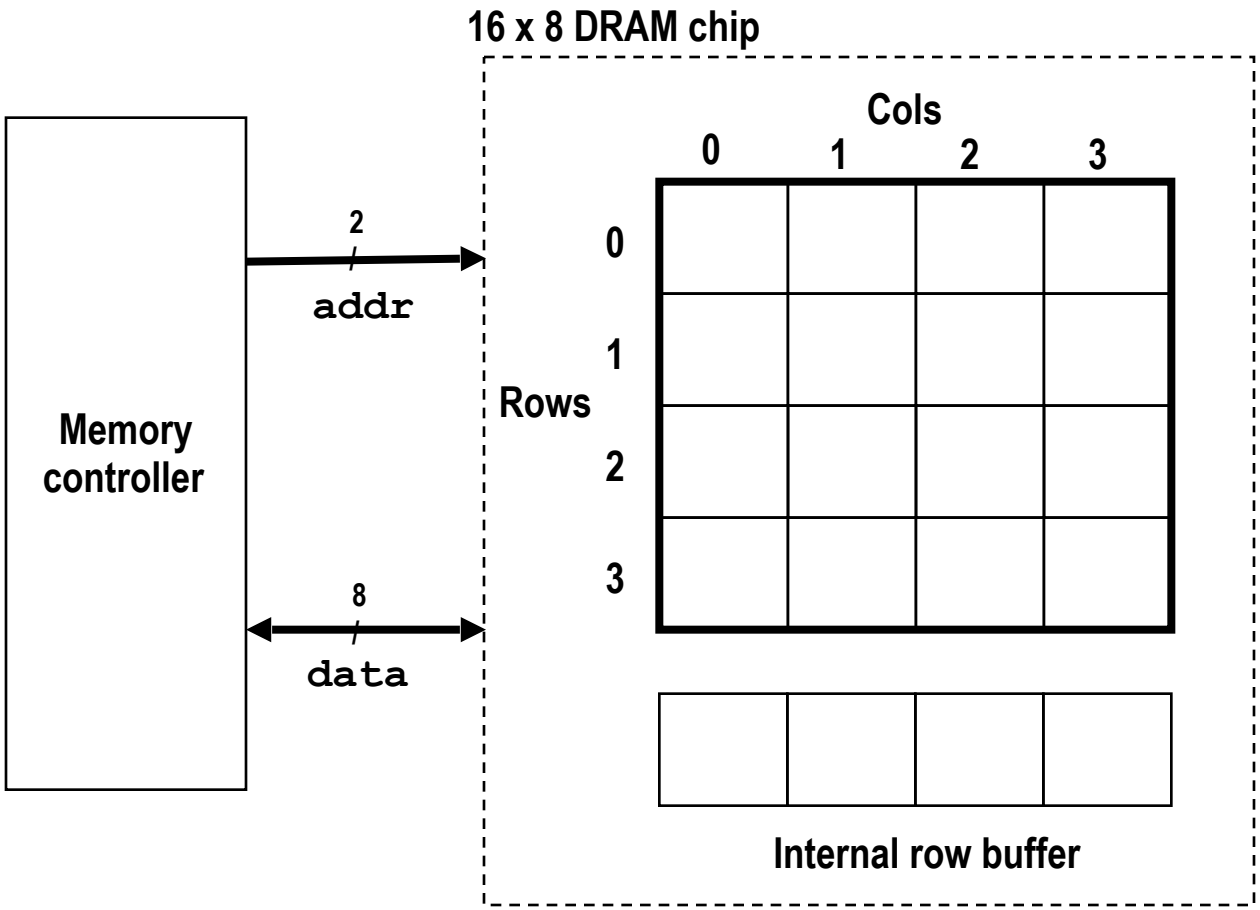
DRAM Chip Organization



Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

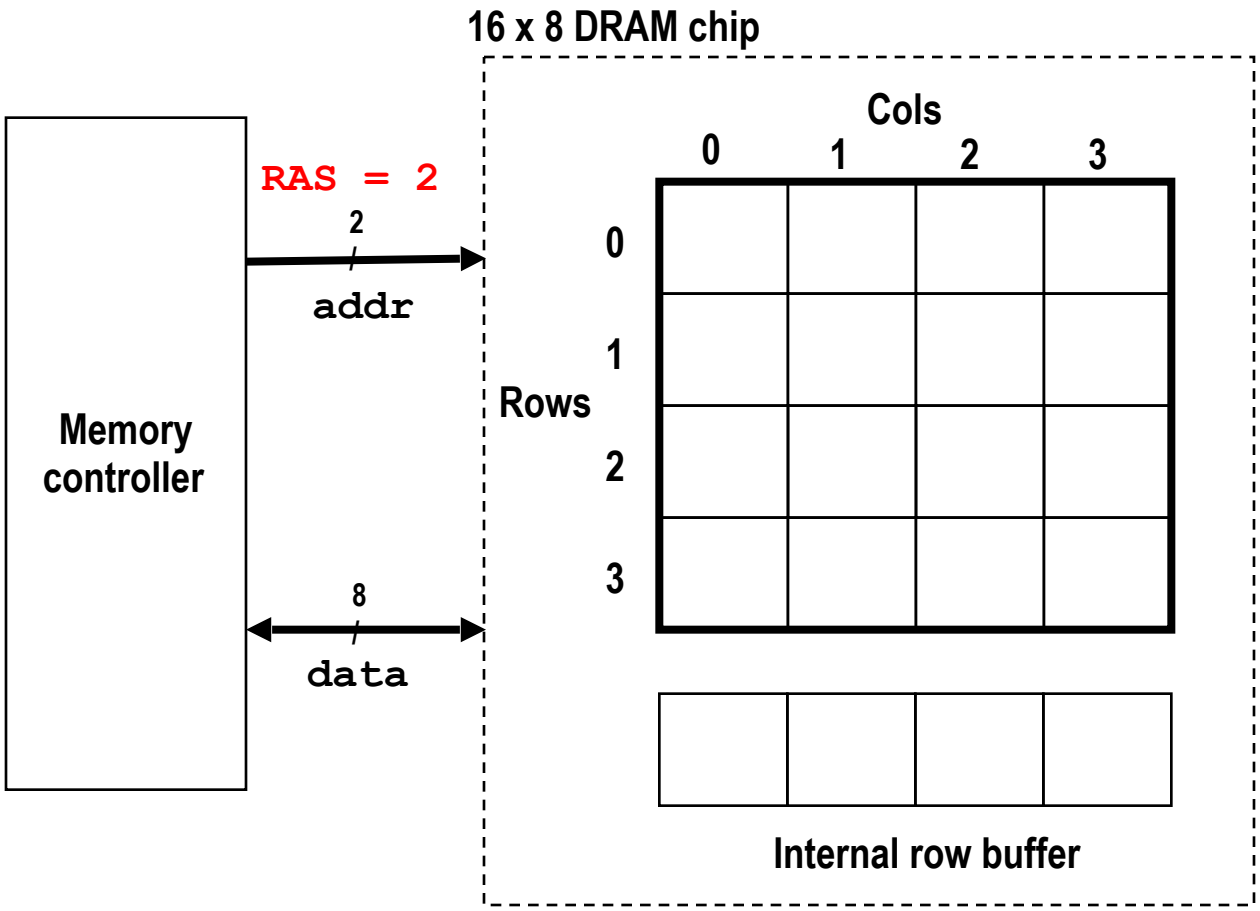
Step 1(b): Row 2 copied from DRAM array to row buffer.



Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

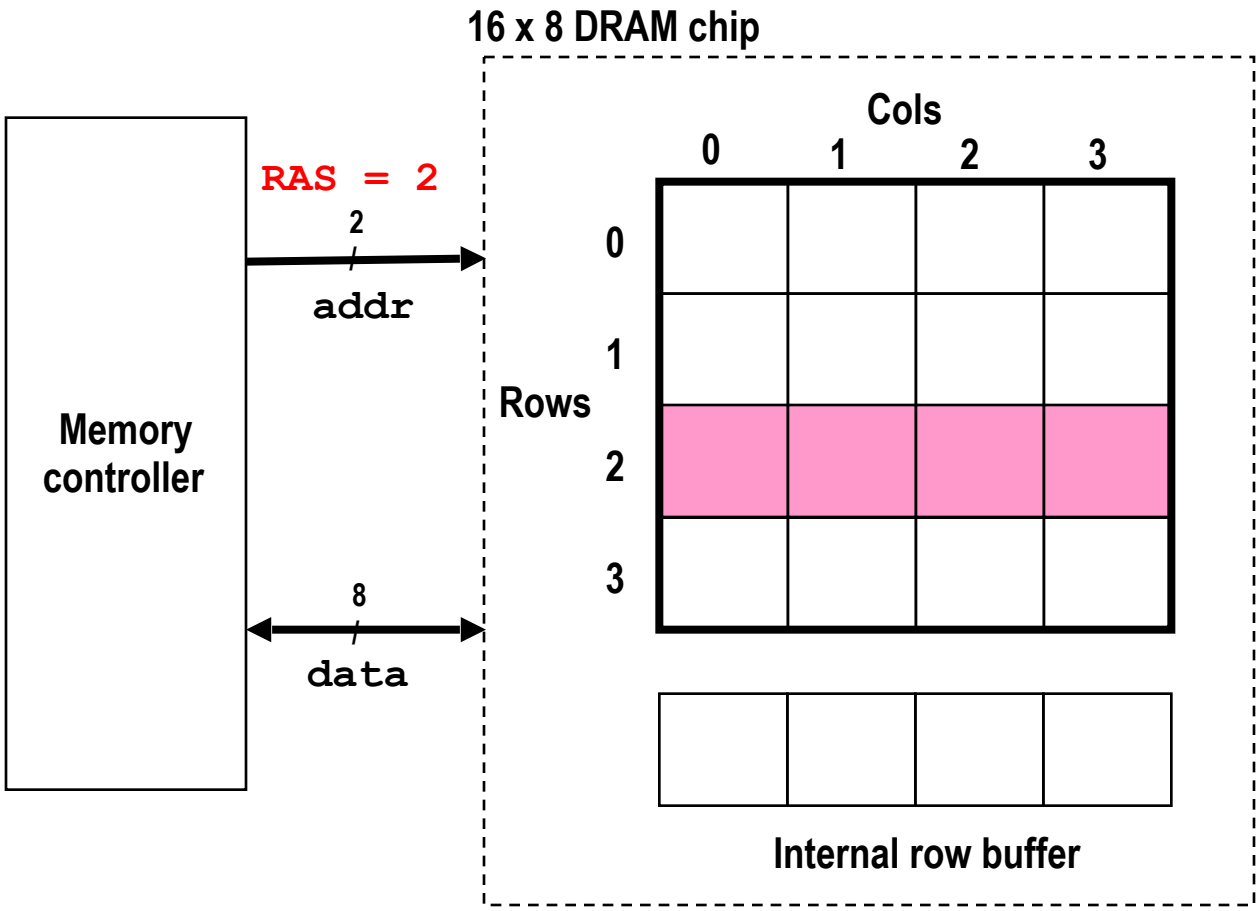
Step 1(b): Row 2 copied from DRAM array to row buffer.



Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

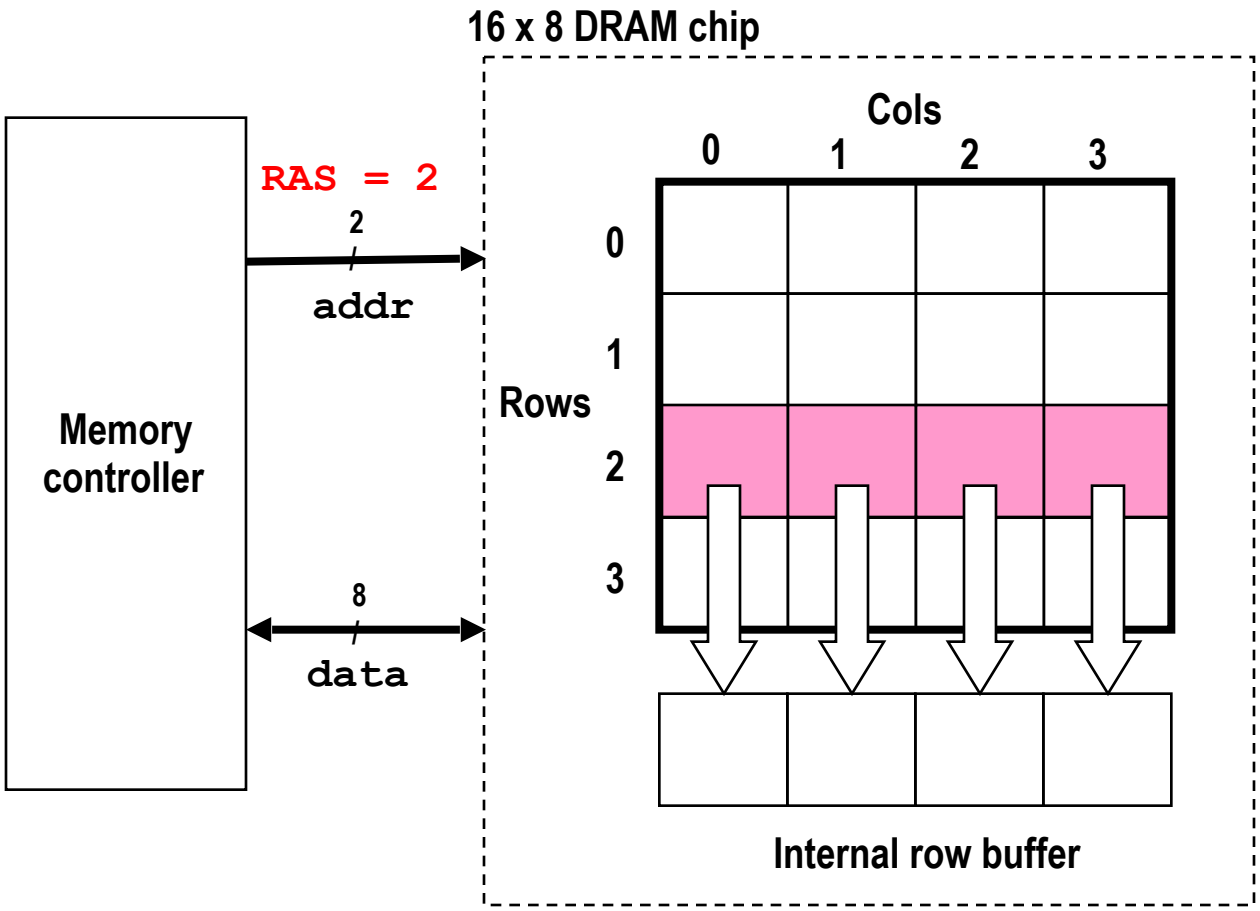
Step 1(b): Row 2 copied from DRAM array to row buffer.



Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

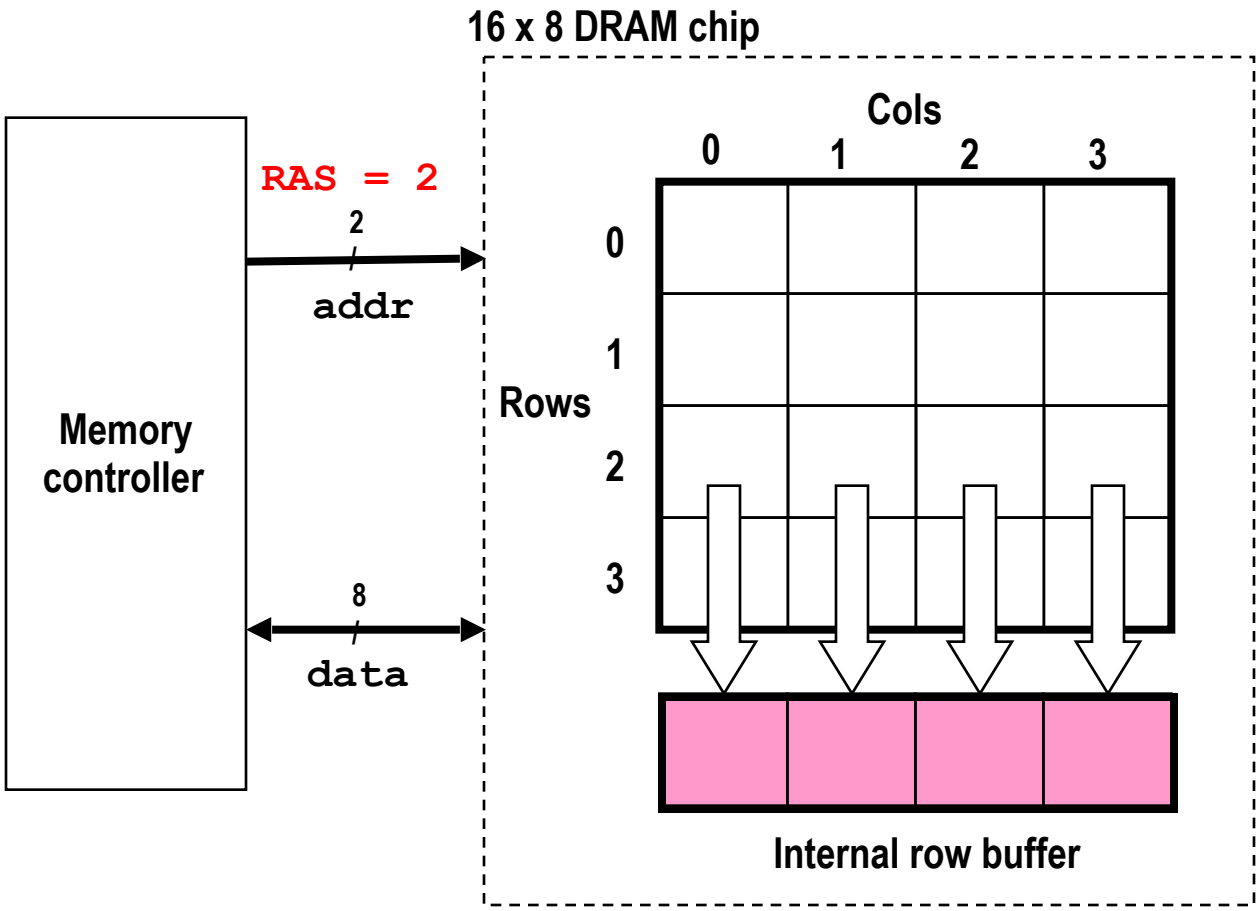
Step 1(b): Row 2 copied from DRAM array to row buffer.



Reading DRAM Supercell (2,1)

Step 1(a): Row access strobe (RAS) selects row 2.

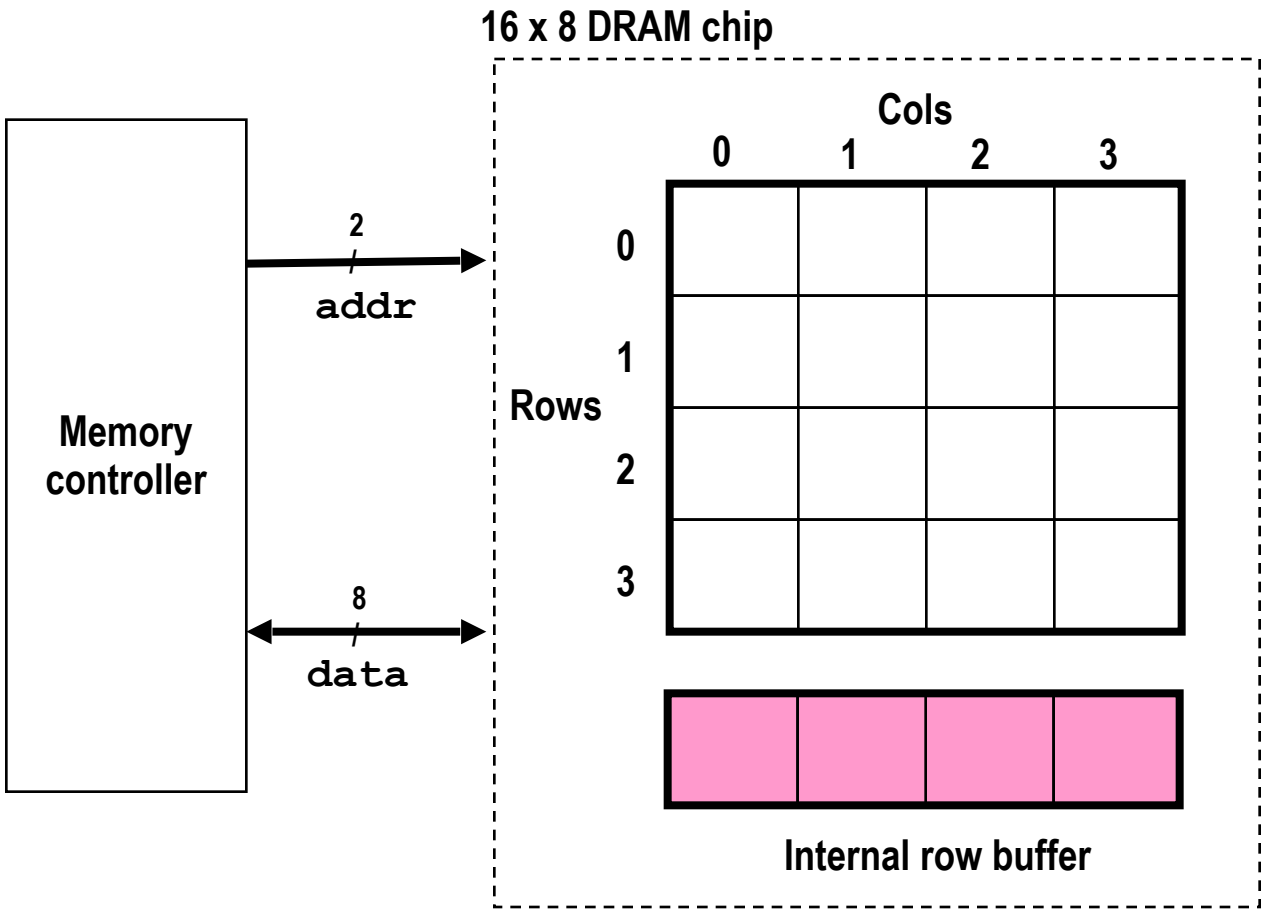
Step 1(b): Row 2 copied from DRAM array to row buffer.



Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

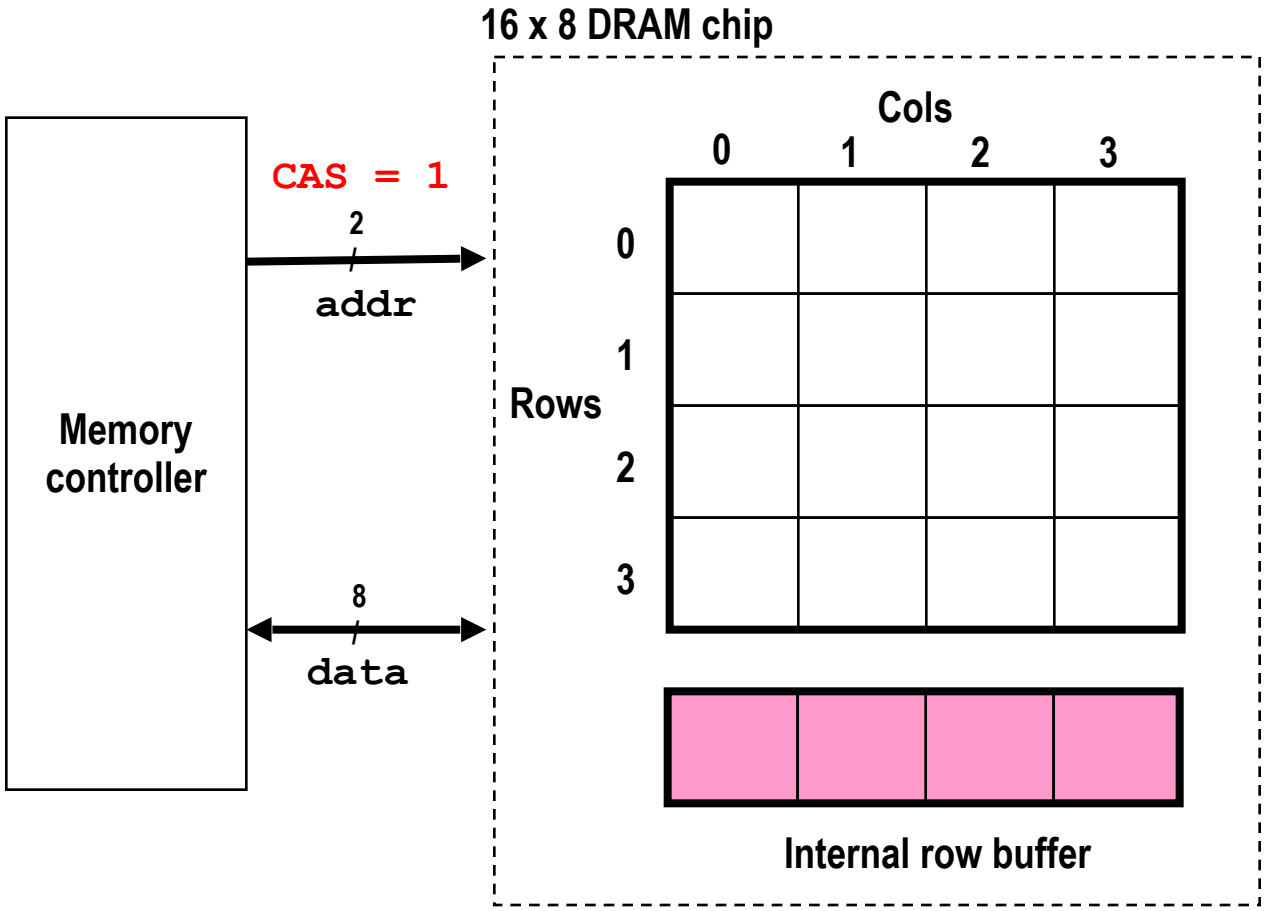
Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

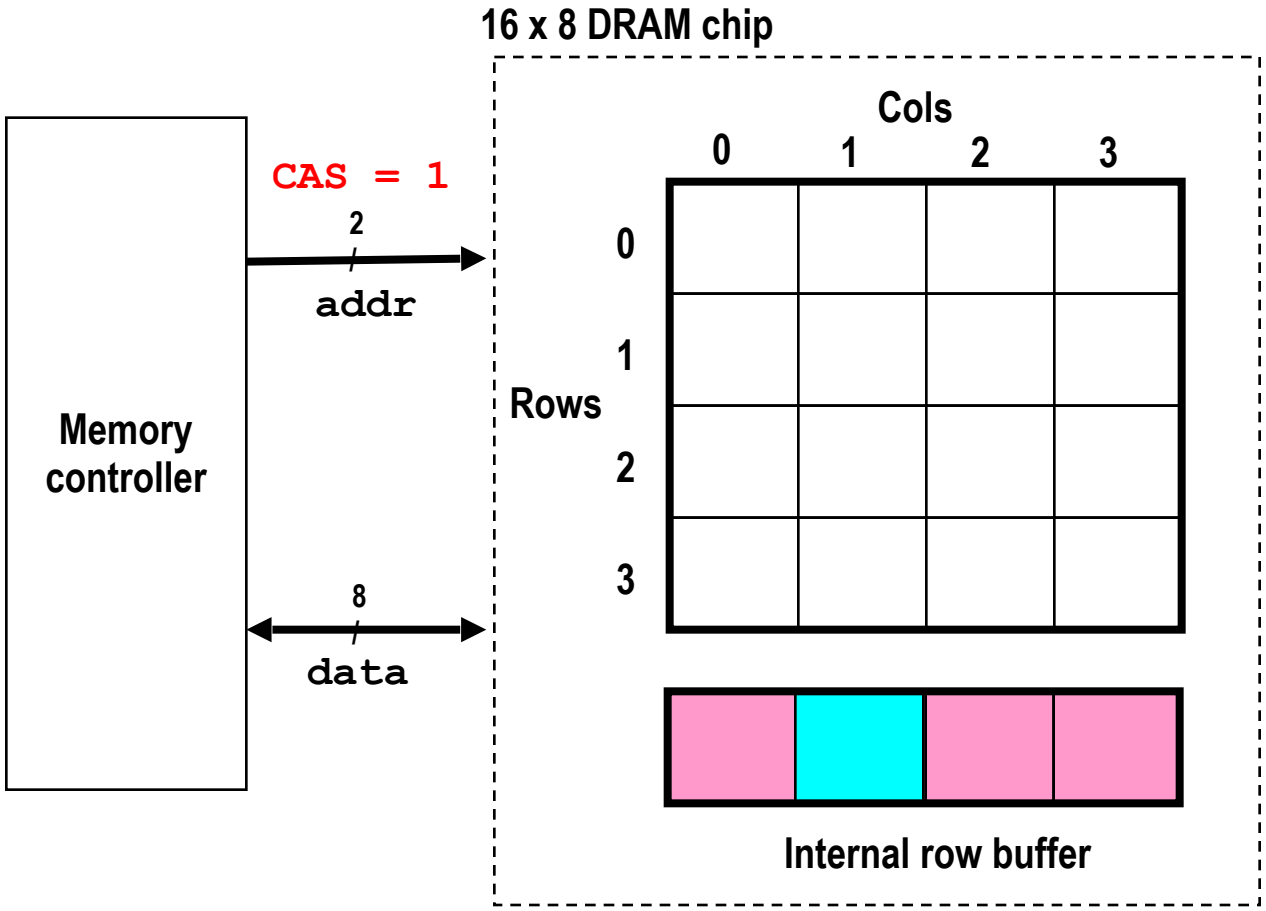
Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

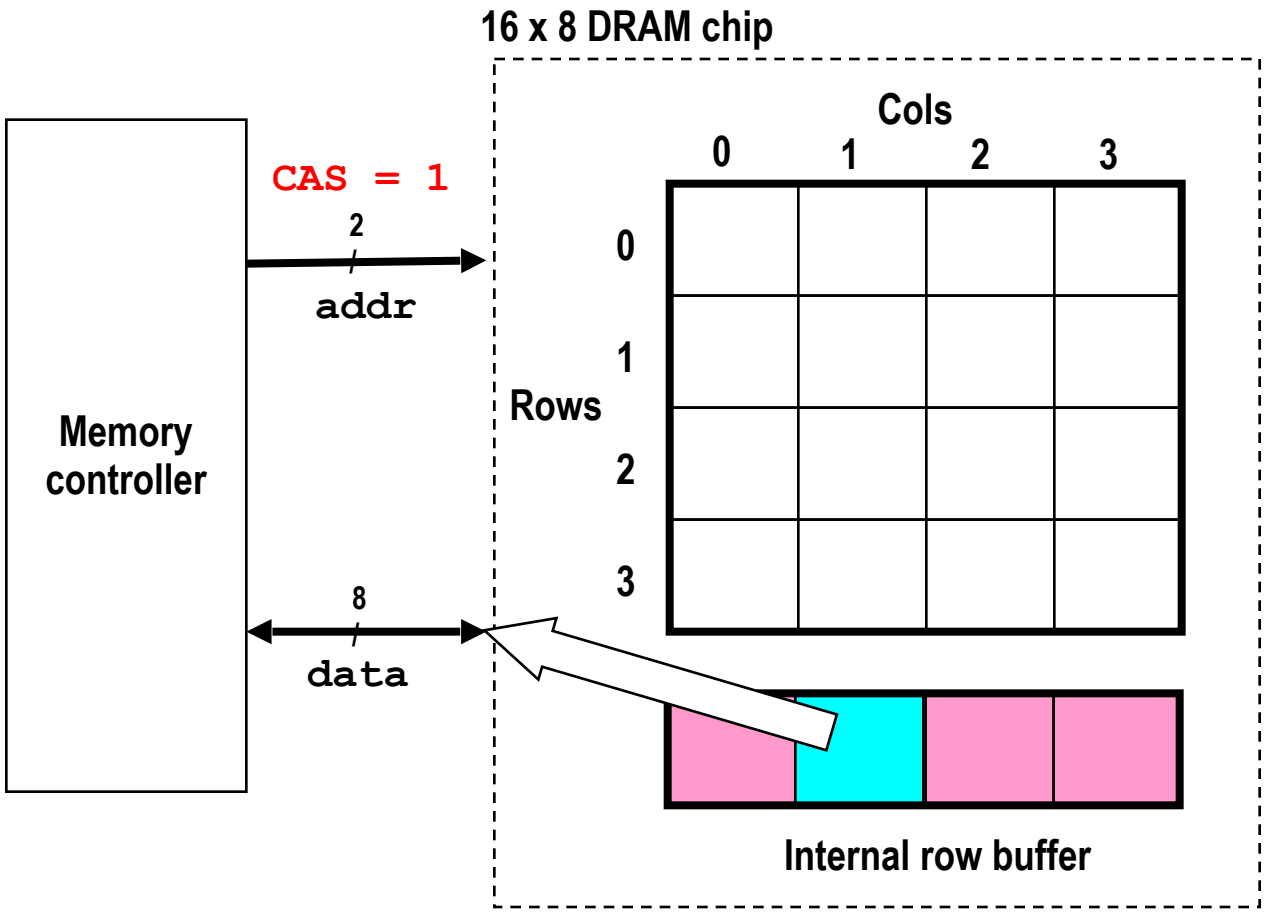
Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

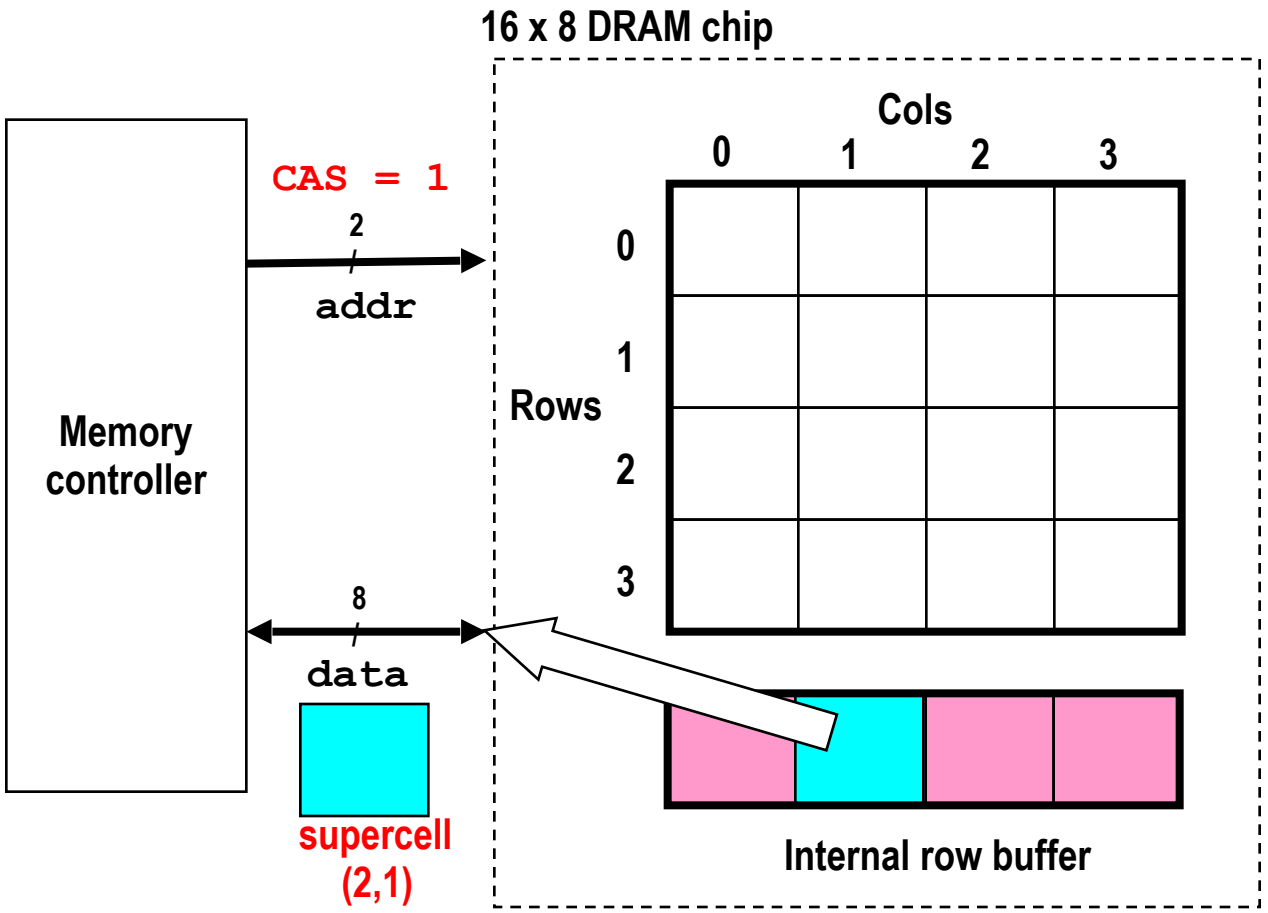
Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



Reading DRAM Supercell (2,1)

Step 2(a): Column access strobe (CAS) selects column 1.

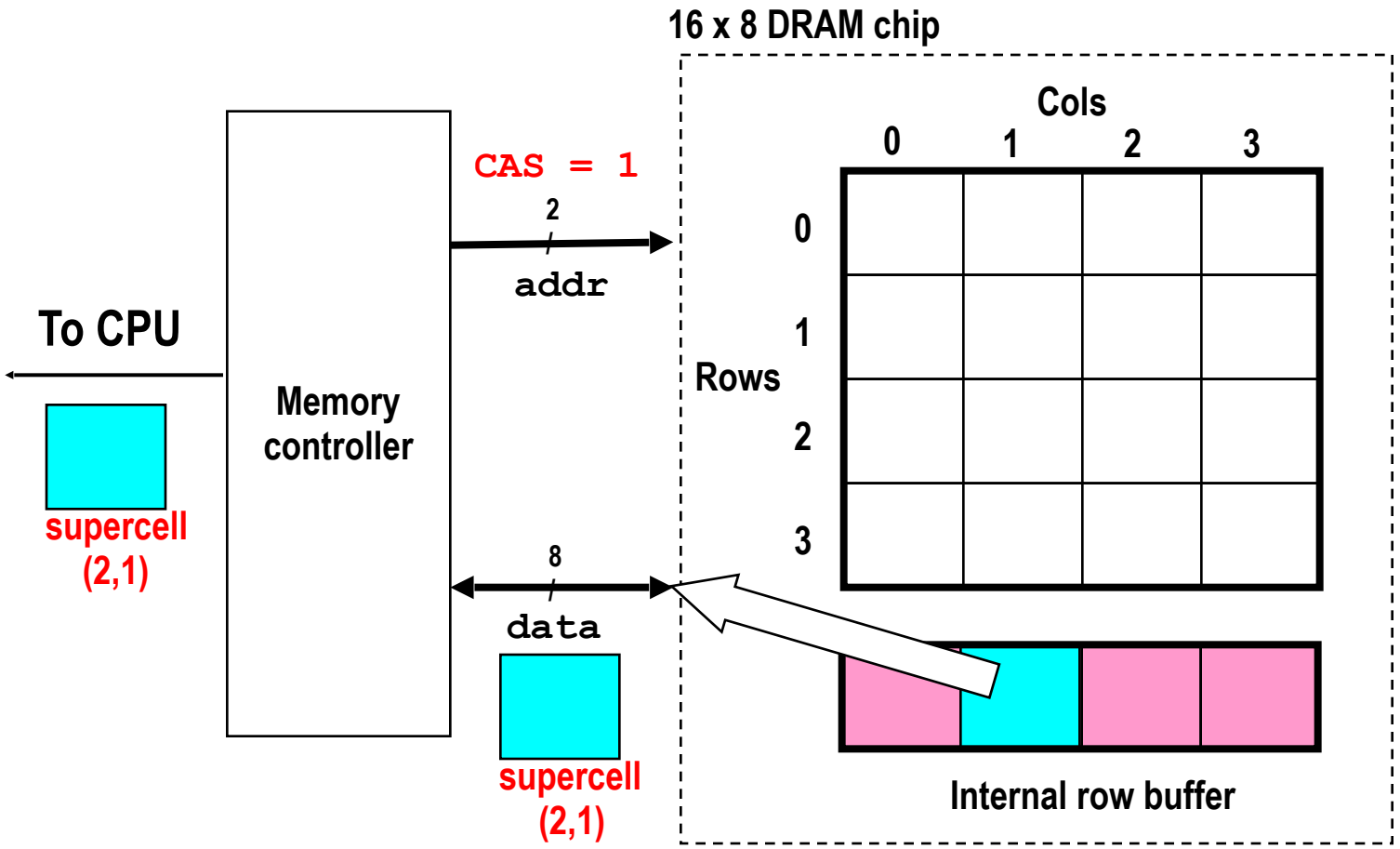
Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



Reading DRAM Supercell (2,1)

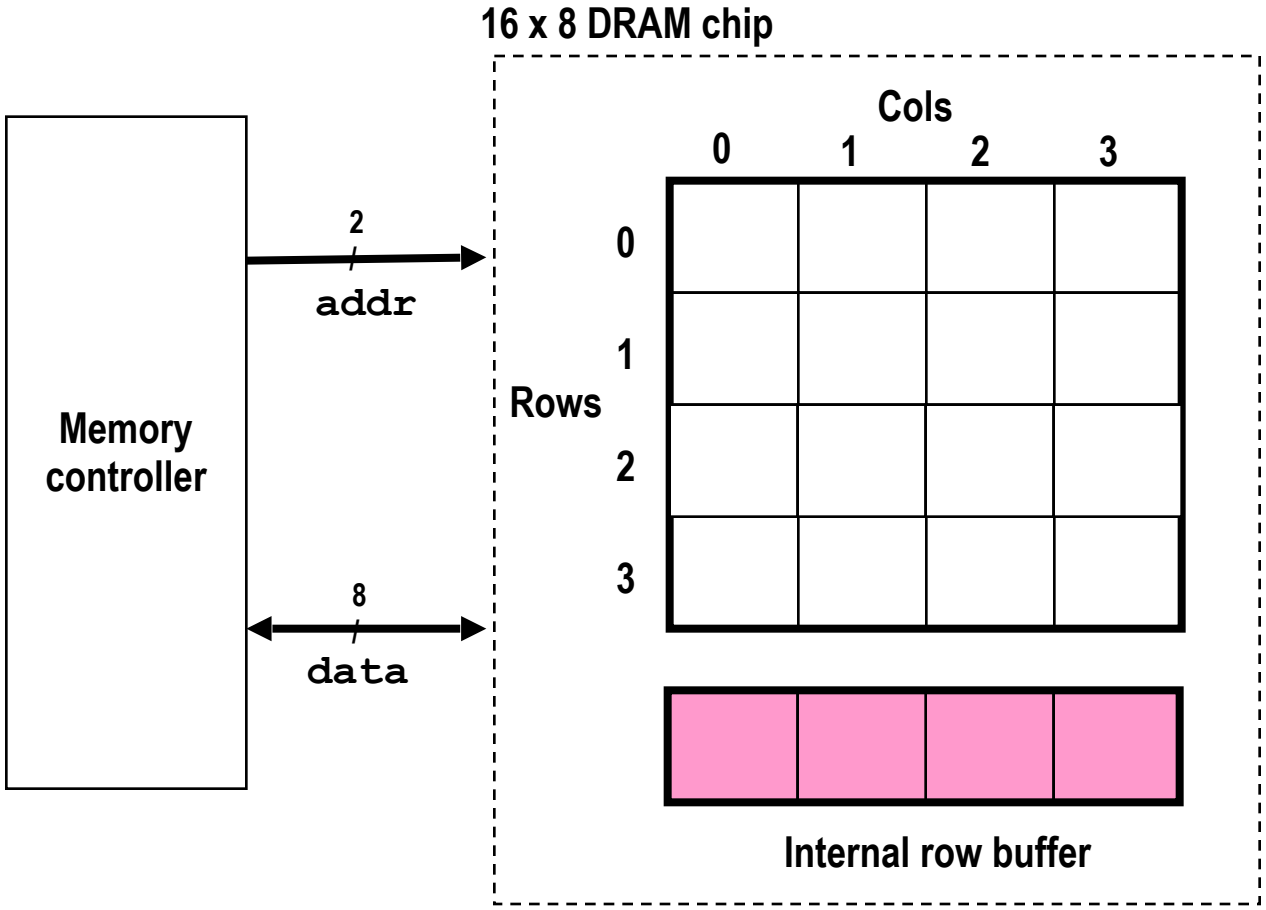
Step 2(a): Column access strobe (CAS) selects column 1.

Step 2(b): Supercell (2,1) copied from buffer to data lines, and eventually back to the CPU.



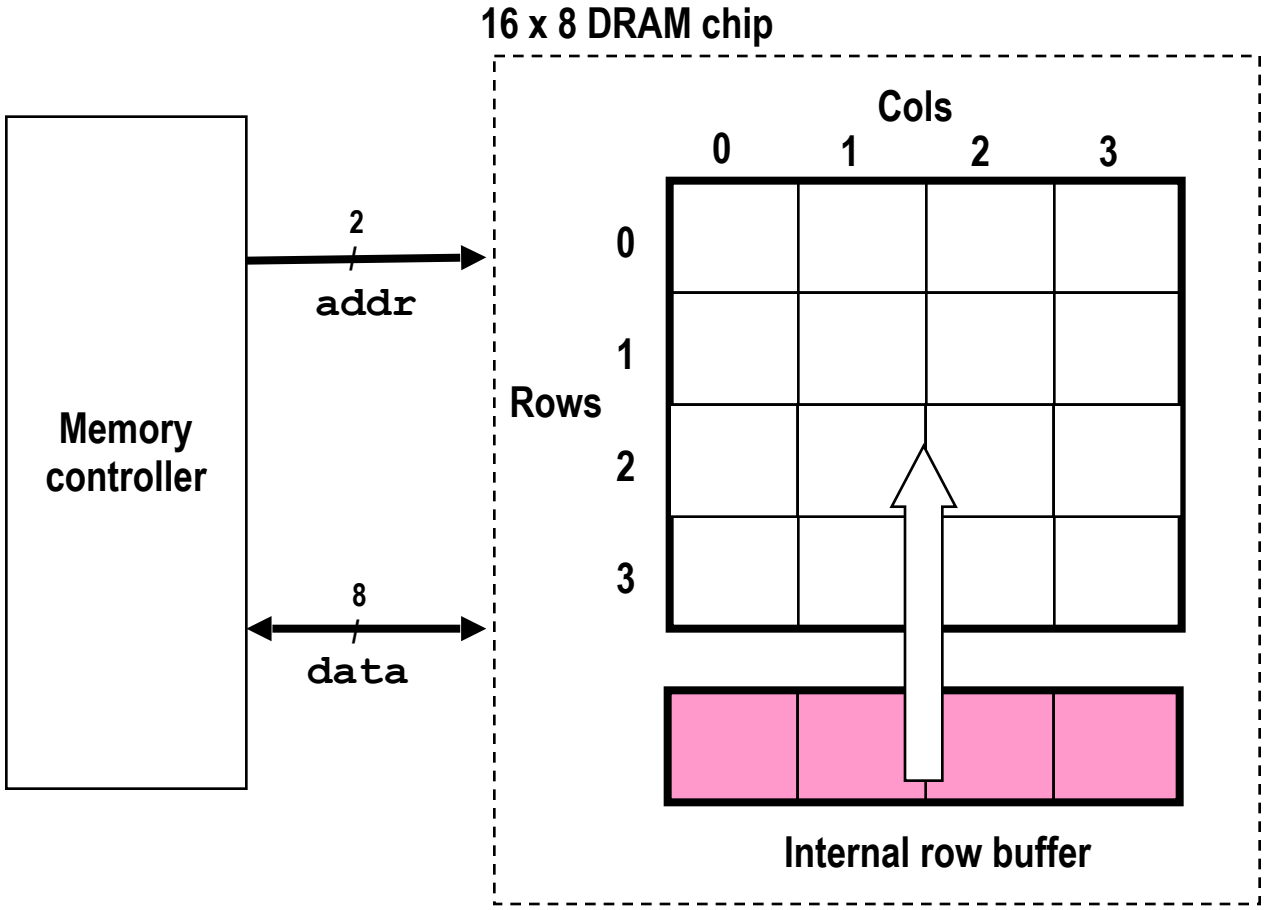
Reading DRAM Supercell (2,1)

Step 3: A sense amplifier amplifies and regenerates the bitline and refresh the cells. A DRAM controller must periodically read each row within the allowed refresh time (10s of ms) to restore charge.



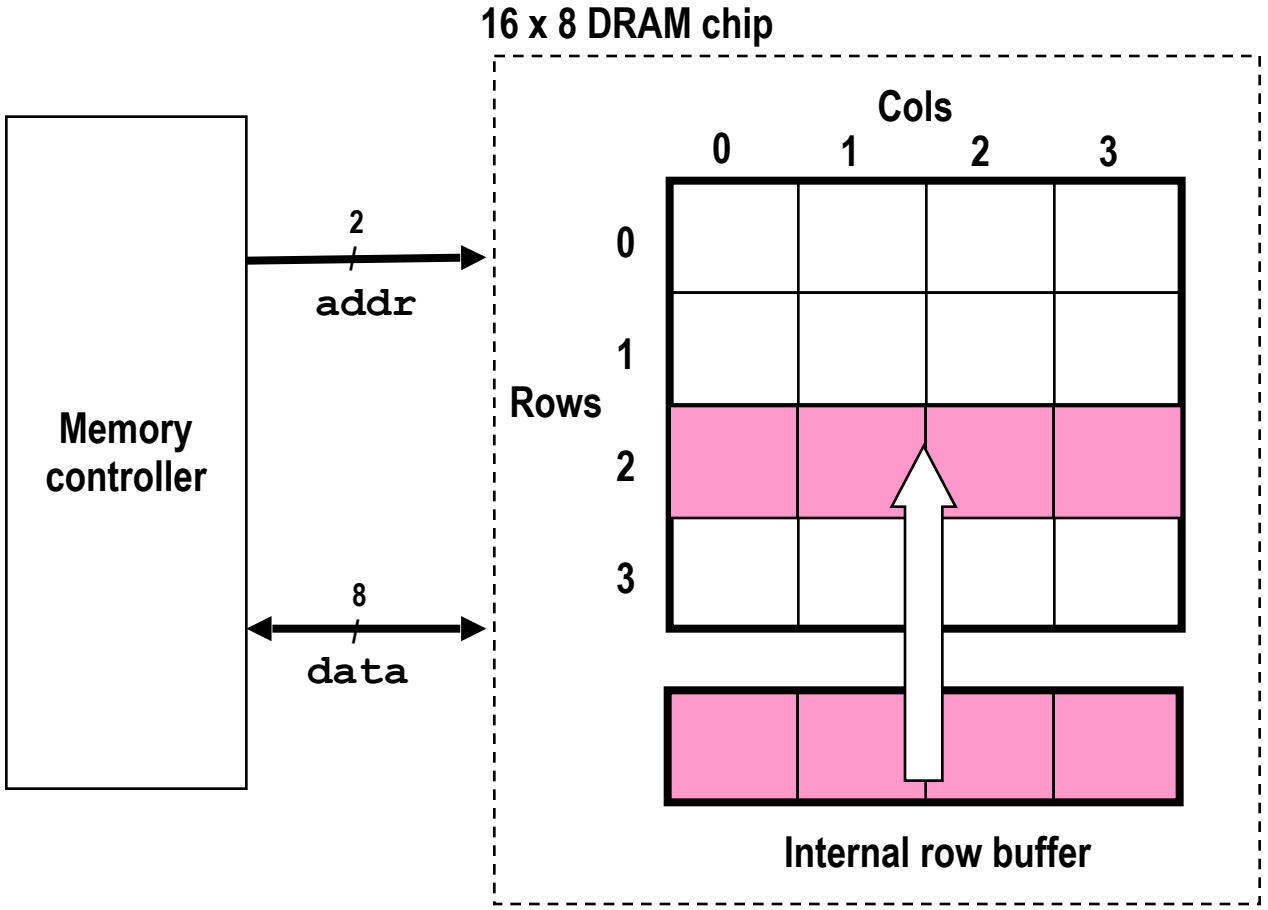
Reading DRAM Supercell (2,1)

Step 3: A sense amplifier amplifies and regenerates the bitline and refresh the cells. A DRAM controller must periodically read each row within the allowed refresh time (10s of ms) to restore charge.

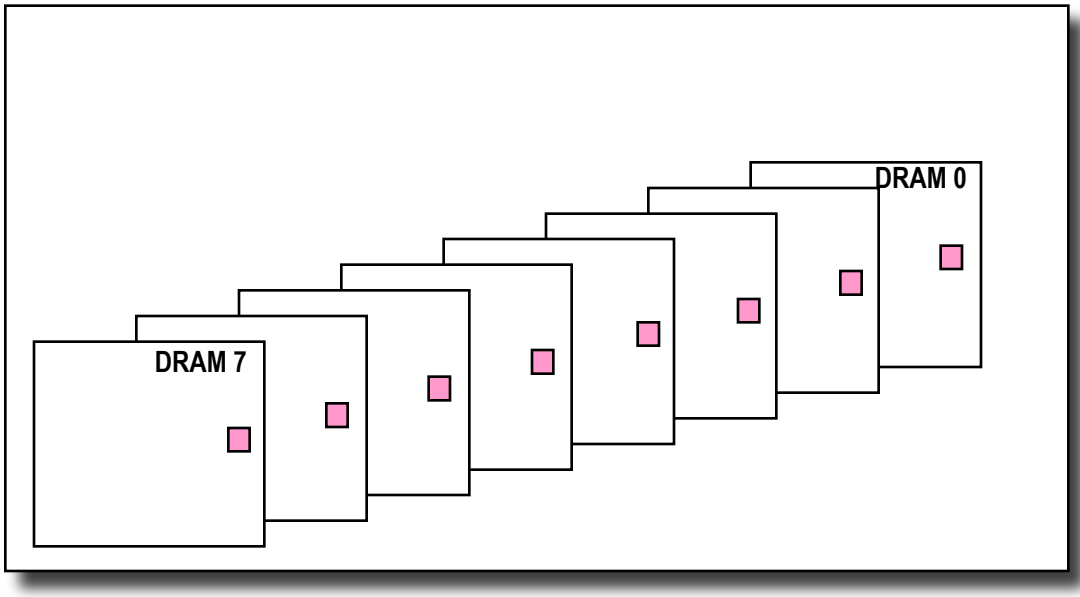


Reading DRAM Supercell (2,1)

Step 3: A sense amplifier amplifies and regenerates the bitline and refresh the cells. A DRAM controller must periodically read each row within the allowed refresh time (10s of ms) to restore charge.



Memory Modules

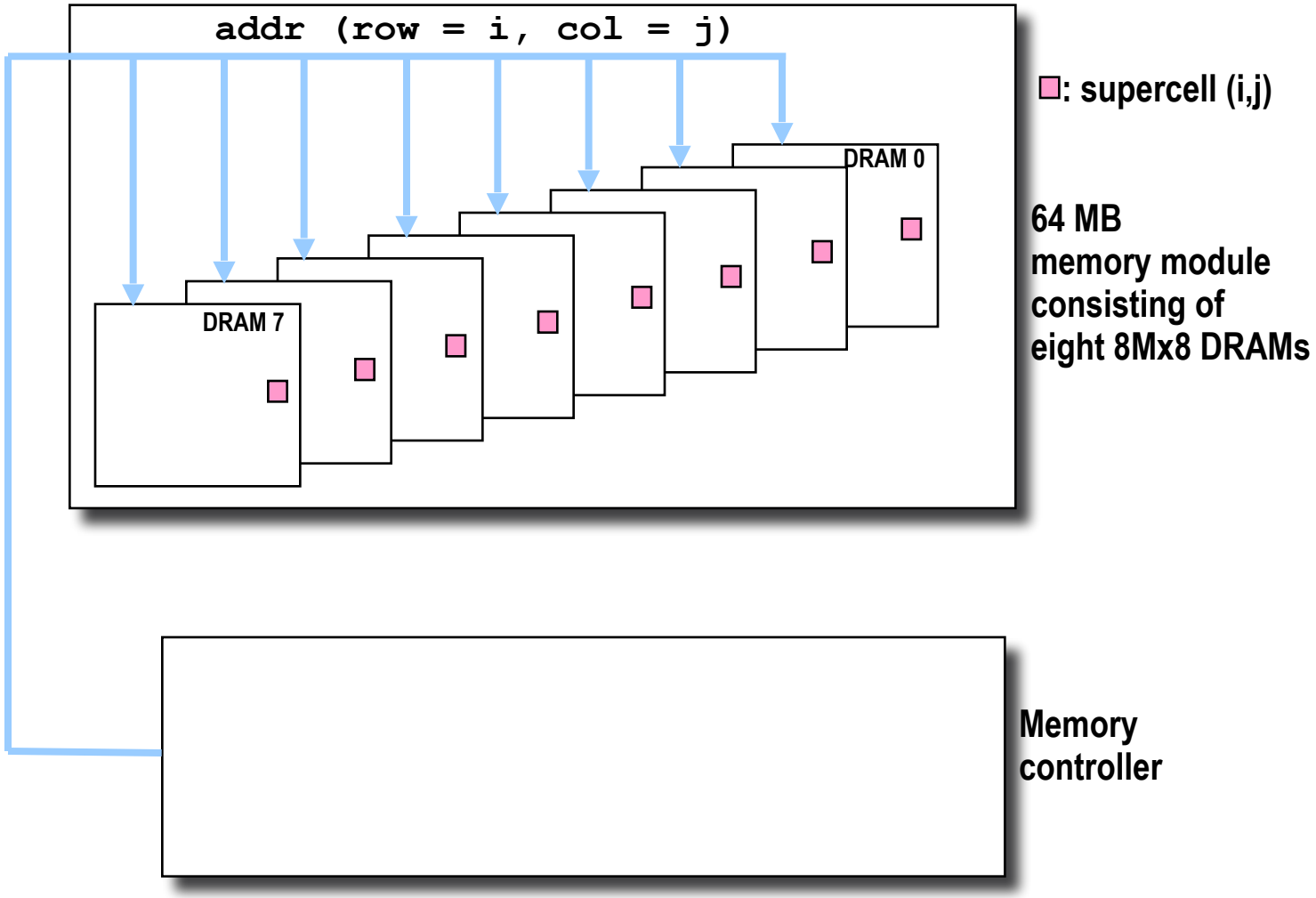


: supercell (i,j)

64 MB
memory module
consisting of
eight 8Mx8 DRAMs

Memory
controller

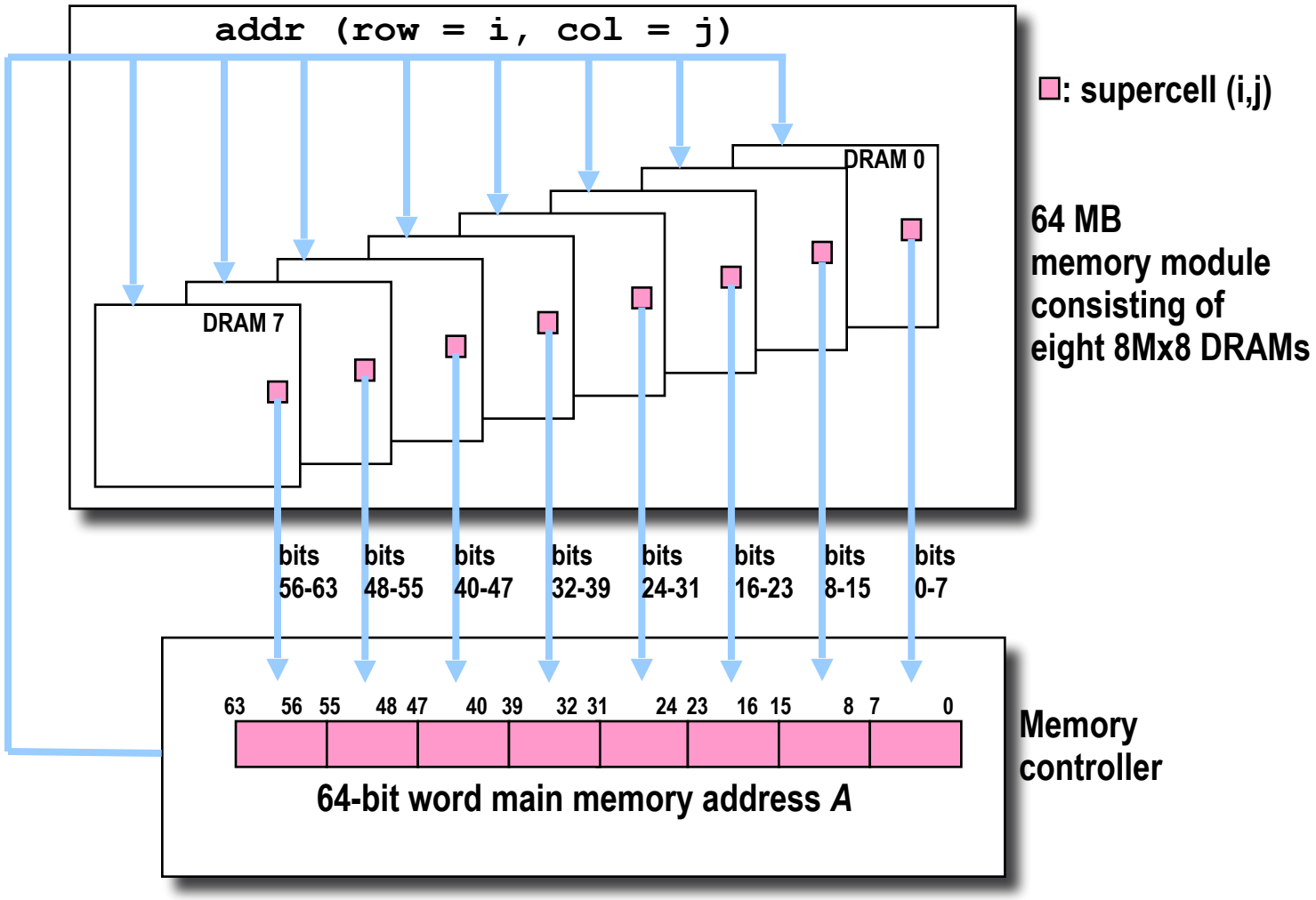
Memory Modules



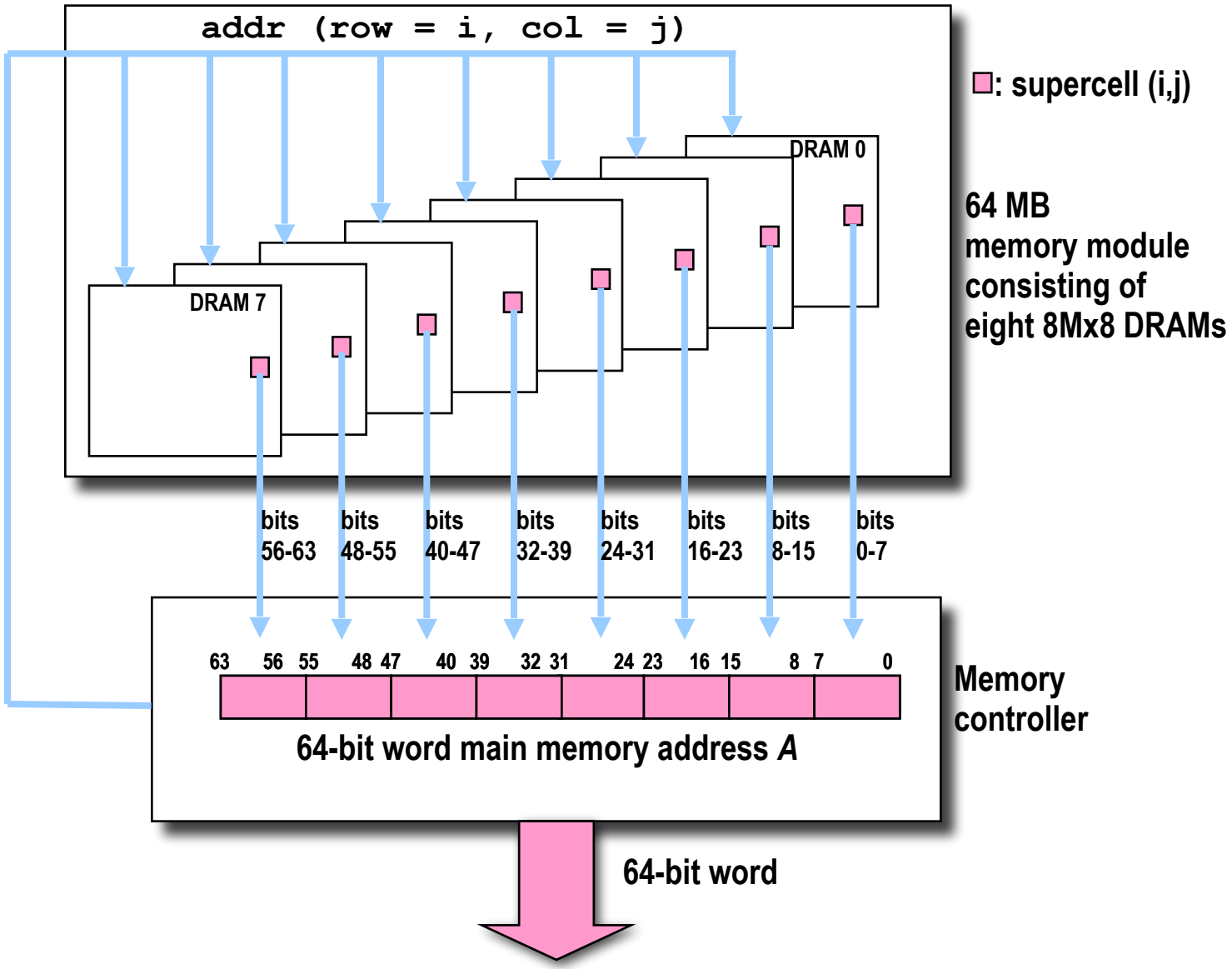
■: supercell (i,j)

64 MB memory module consisting of eight 8Mx8 DRAMs

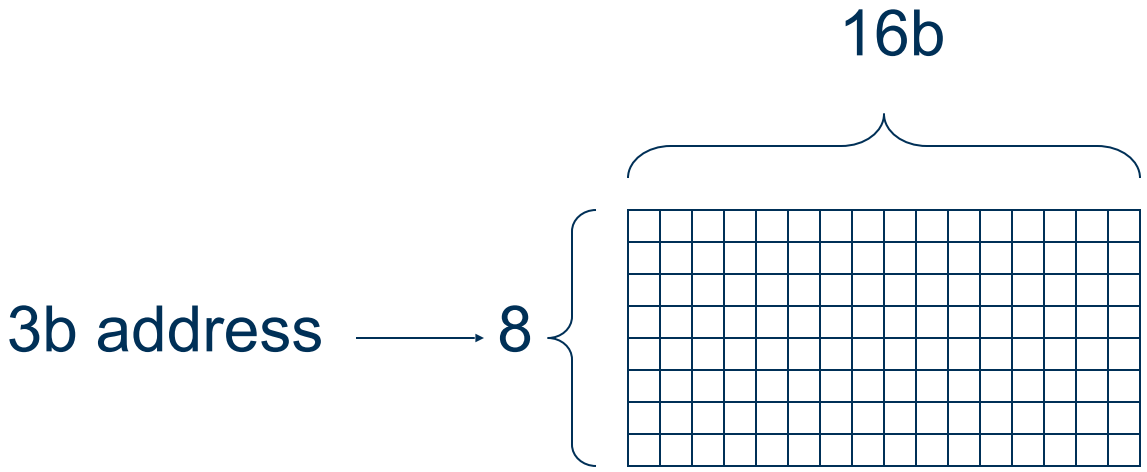
Memory Modules



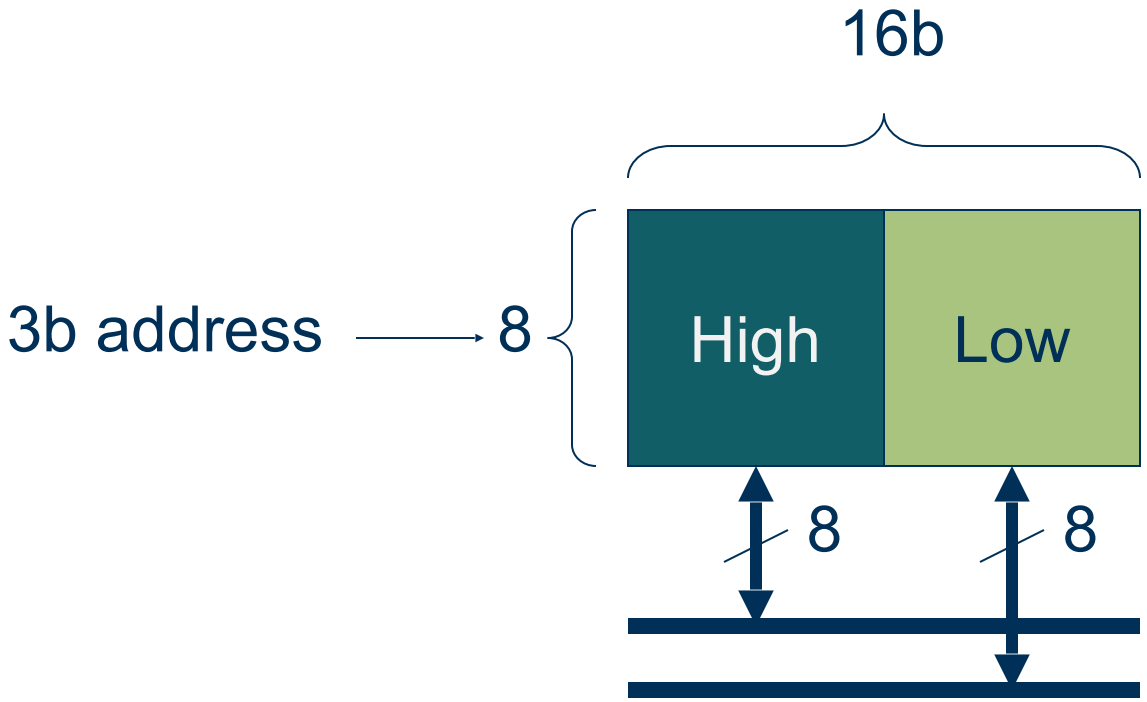
Memory Modules



Memory Layout Across Two Chips

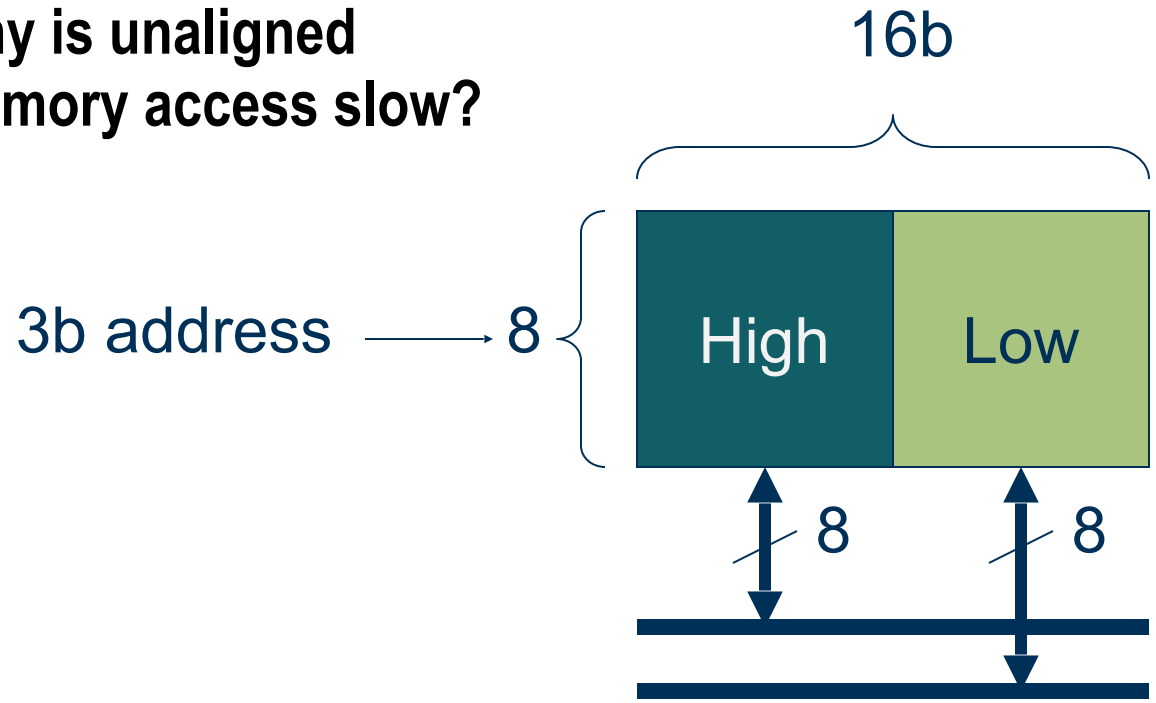


Memory Layout Across Two Chips



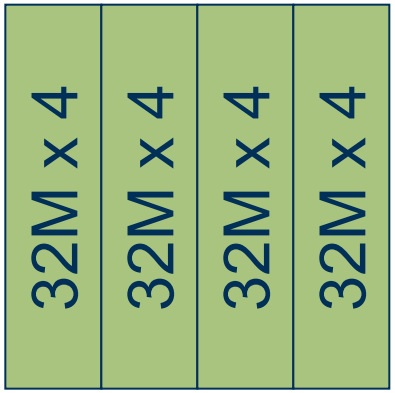
Memory Layout Across Two Chips

Why is unaligned memory access slow?



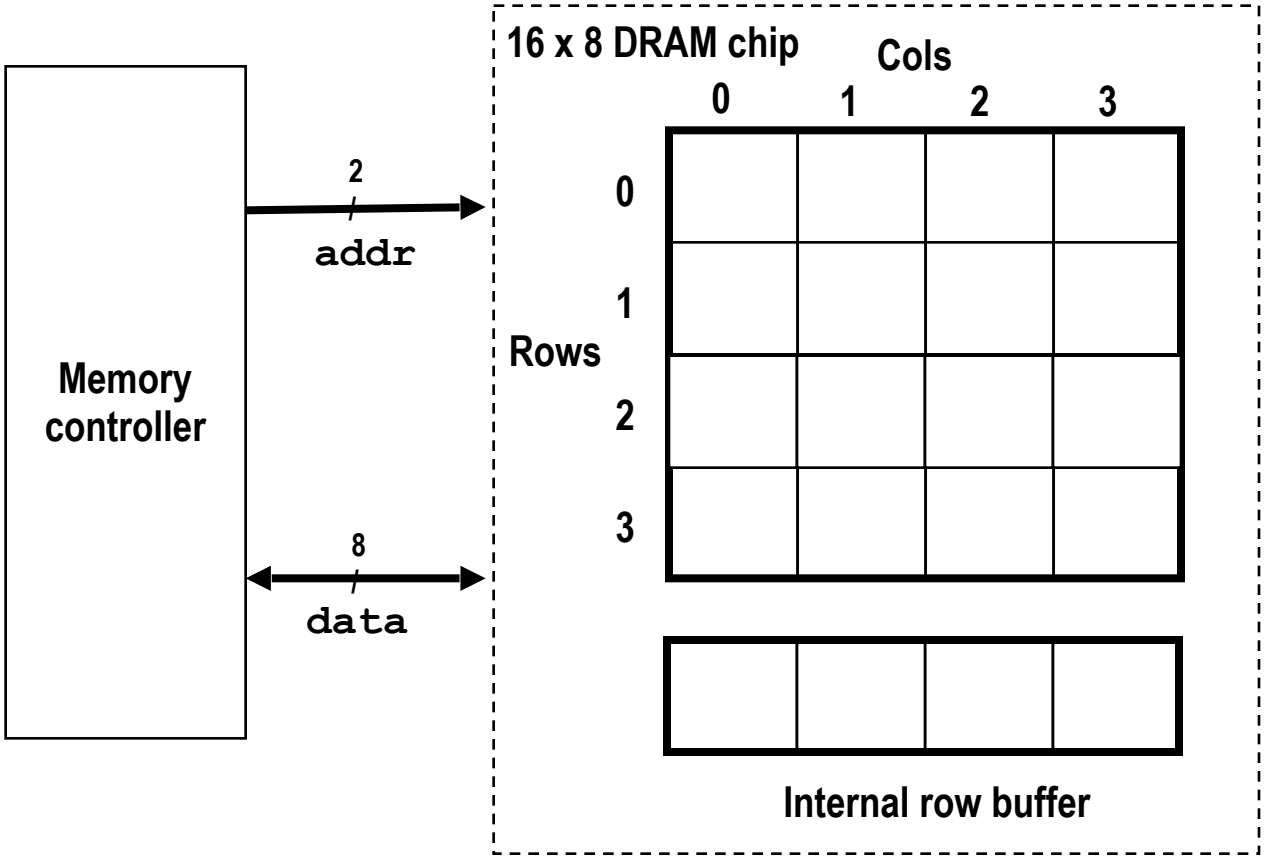
Narrower RAMs Enable Greater Capacity

- Given Constant Total Width (pins)
- Multiple smaller chips are more reliable than one big chip
  - Yield rate issue

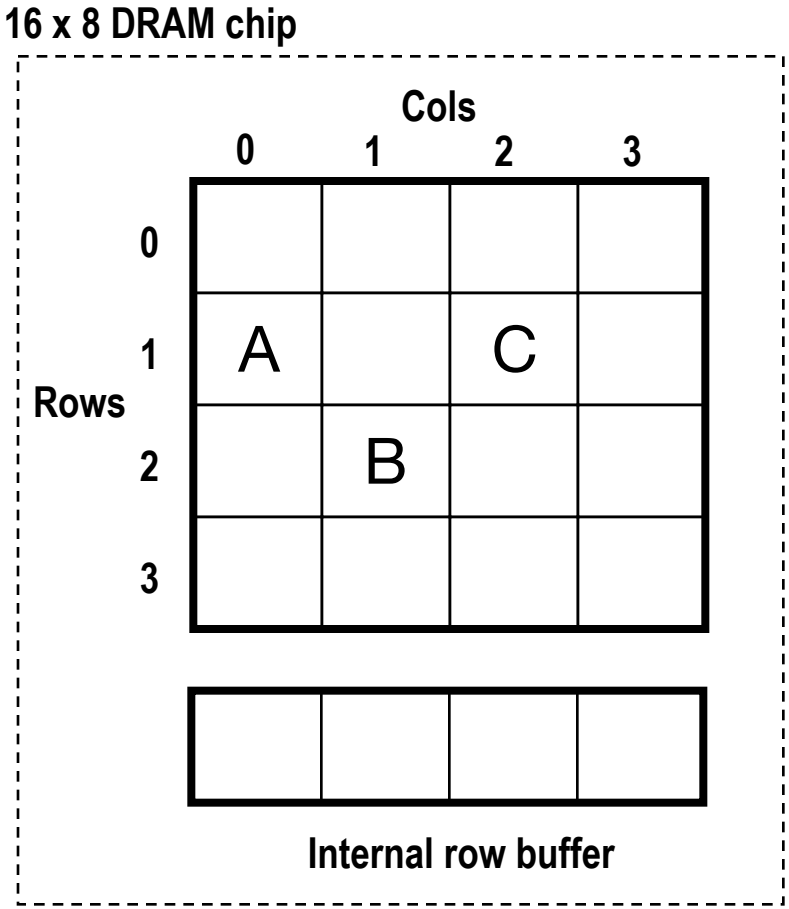


Why Split Address into Row and Column?

- +: Reduce the number of address pins
- +: Also allow reading multiple columns within the same row
- -: Send address in two steps, increase latency

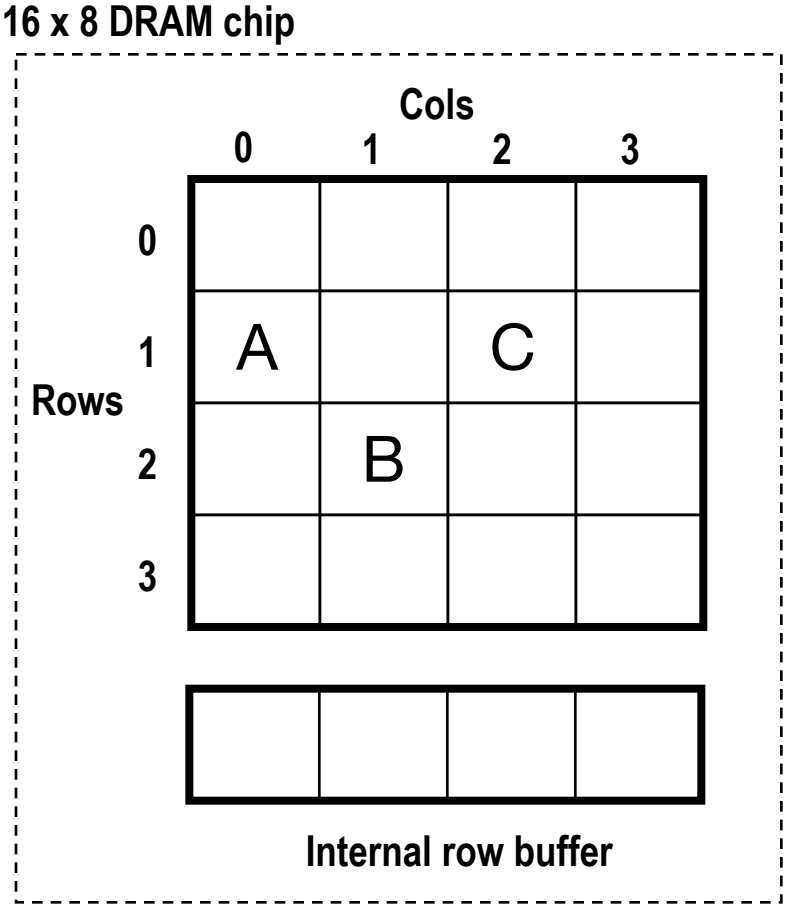


Memory Scheduling



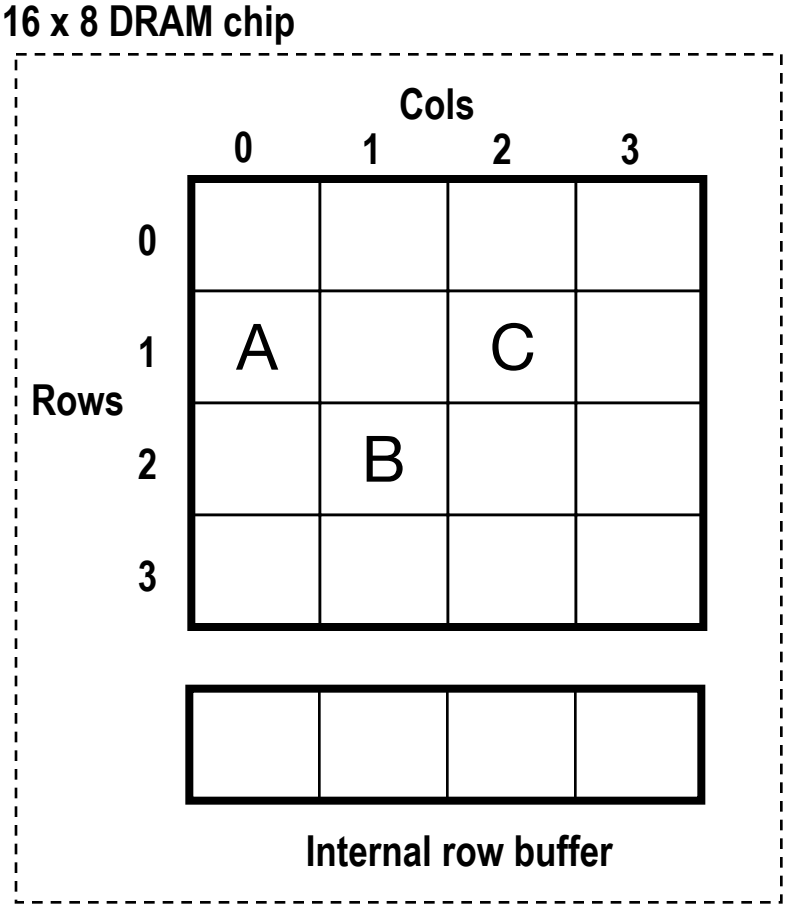
Memory Scheduling

- Assume the following memory accesses: A, B, C



Memory Scheduling

- Assume the following memory accesses: A, B, C
- Which one is faster?
  - A —> B —> C
  - A —> C —> B



Memory Scheduling

- Assume the following memory accesses: A, B, C
- Which one is faster?
  - A —> B —> C
  - A —> C —> B
- Most common memory scheduling policy: FR-FCFS
  - First-ready, first-come-first-serve
  - Prioritize addresses to data that is already in the row buffer; otherwise first-come-first-serve