

# CS 31: Intro to Systems Storage and Memory

Kevin Webb

Swarthmore College

March 17, 2015

# Transition

- First half of course: hardware focus
  - How the hardware is constructed
  - How the hardware works
  - How to interact with hardware / ISA
- Up next: performance and software systems
  - Memory performance
  - Operating systems
  - Standard libraries (strings, threads, etc.)

# Efficiency

- How to Efficiently Run Programs
- Good algorithm is critical...
- Many systems concerns to account for too!
  - The Memory Hierarchy and its effect on program performance
  - OS abstractions for running programs efficiently
  - Support for Parallel programming

# Efficiency

- How to Efficiently Run Programs
- Good algorithm is critical...
- Many systems concerns to account for too!
  - The Memory Hierarchy and its effect on program performance
  - OS abstractions for running programs efficiently
  - Support for Parallel programming

Suppose you're designing a new computer architecture. Which type of memory would you use? Why?

- A. low-capacity (~1 MB), fast, expensive
- B. medium-capacity (a few GB), medium-speed, moderate cost
- C. high-capacity (100's of GB), slow, cheap
- D. something else (it must exist)

# Classifying Memory

- Broadly, two types of memory:
  1. Random access: CPU instructions can access any location at any time (assuming OS permission)
  2. Secondary storage: CPU can't access this directly

# Random Access Memory (RAM)

- Any location can be accessed directly by CPU
  - Volatile Storage: lose power → lose contents
- Static RAM (SRAM)
  - Latch-Based Memory (e.g. RS latch), 1 bit per latch
  - Faster and more expensive than DRAM
    - “On chip”: Registers, Caches
- Dynamic RAM (DRAM)
  - Capacitor-Based Memory, 1 bit per capacitor
    - “Main memory”: Not part of CPU

# Memory Technologies

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, \$20 – \$75 per GB  
(Main memory, “RAM”)

We’ve talked a lot about registers (SRAM) and we’ll cover caches (SRAM) soon. Let’s look at main memory (DRAM) now.

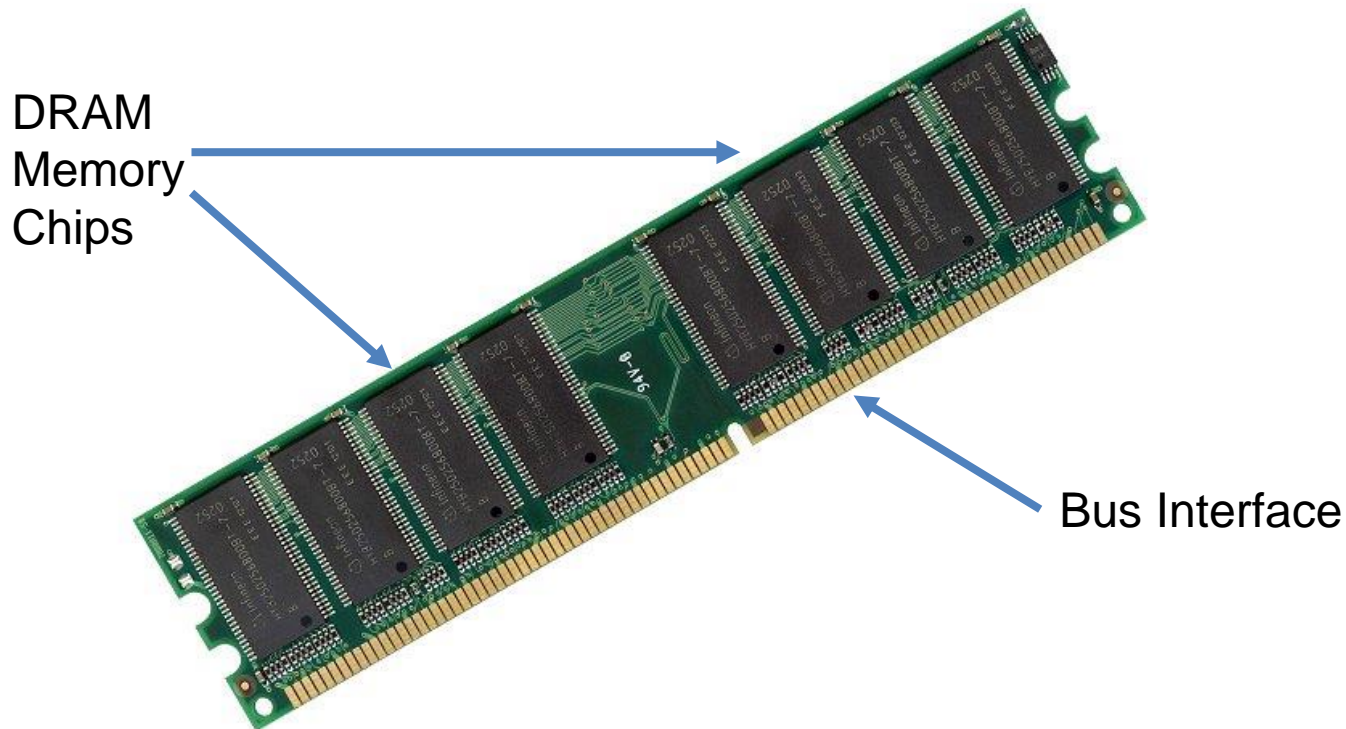


# Dynamic Random Access Memory (DRAM)

Capacitor based:

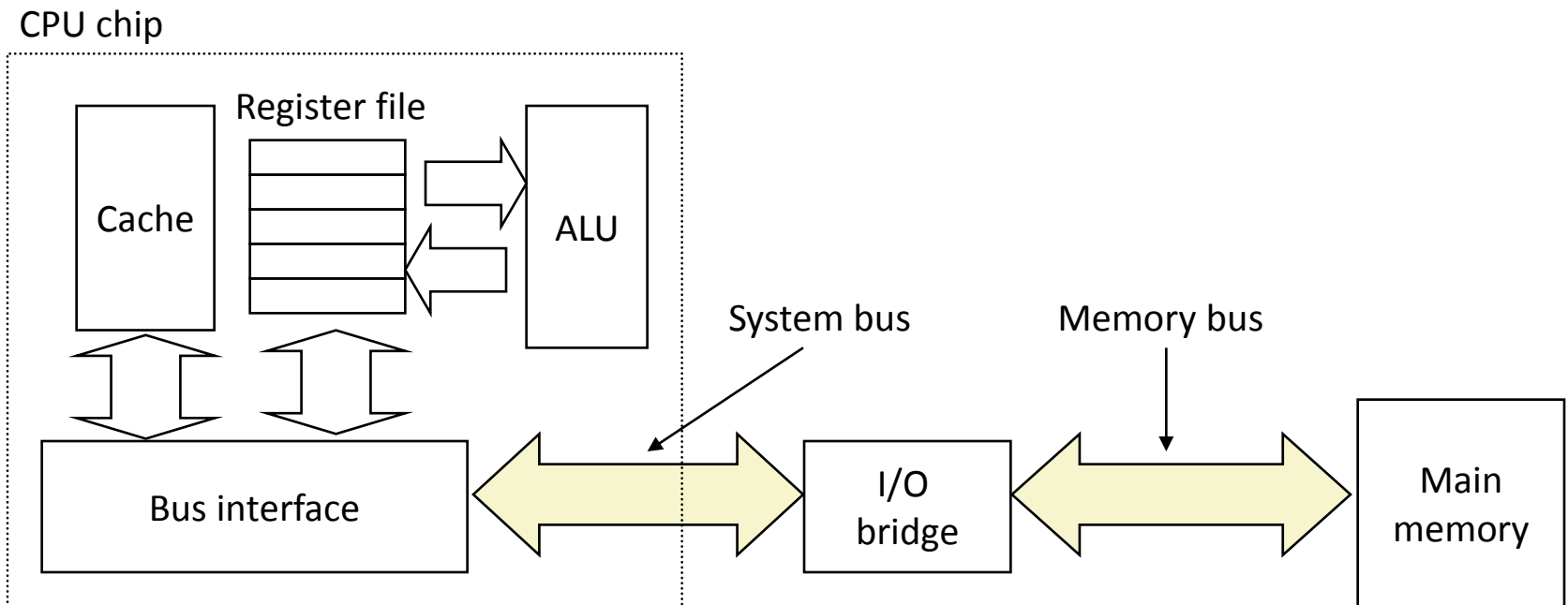
- cheaper and slower than SRAM
- capacitors are leaky (lose charge over time)
- Dynamic: value needs to be refreshed (every 10-100ms)

Example: DIMM (Dual In-line Memory Module):



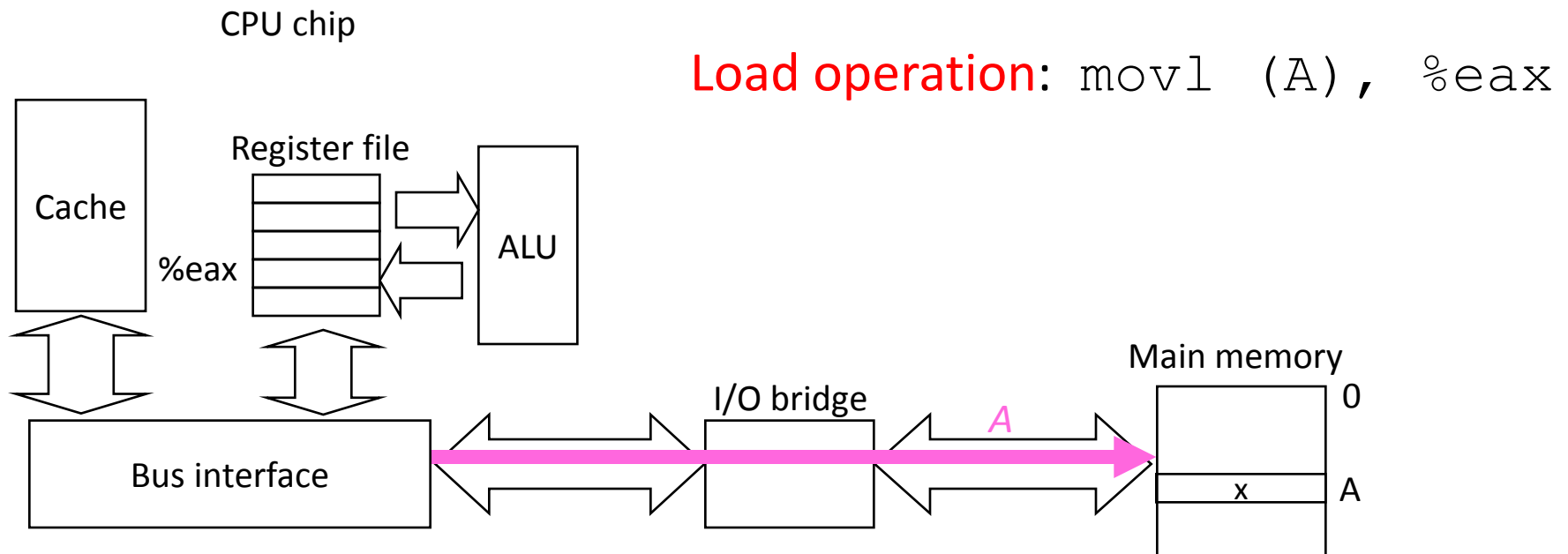
# Connecting CPU and Memory

- Components are connected by a **bus**:
  - A bus is a collection of parallel wires that carry address, data, and control signals.
  - Buses are typically shared by multiple devices.



# How Memory Read Works

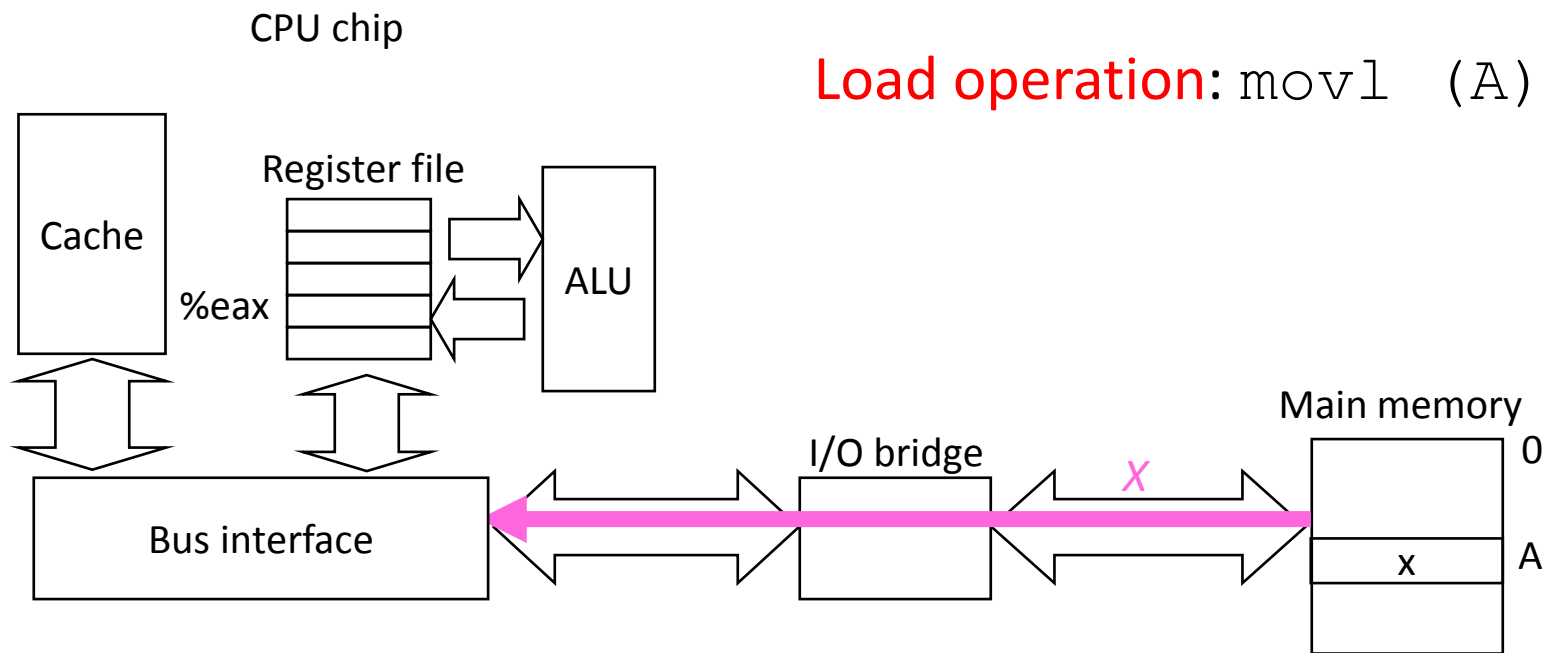
(1) CPU places address A on the memory bus.



# Read (cont.)

(2) Main Memory reads Address A from Memory Bus, fetches x at that address and puts it on the bus

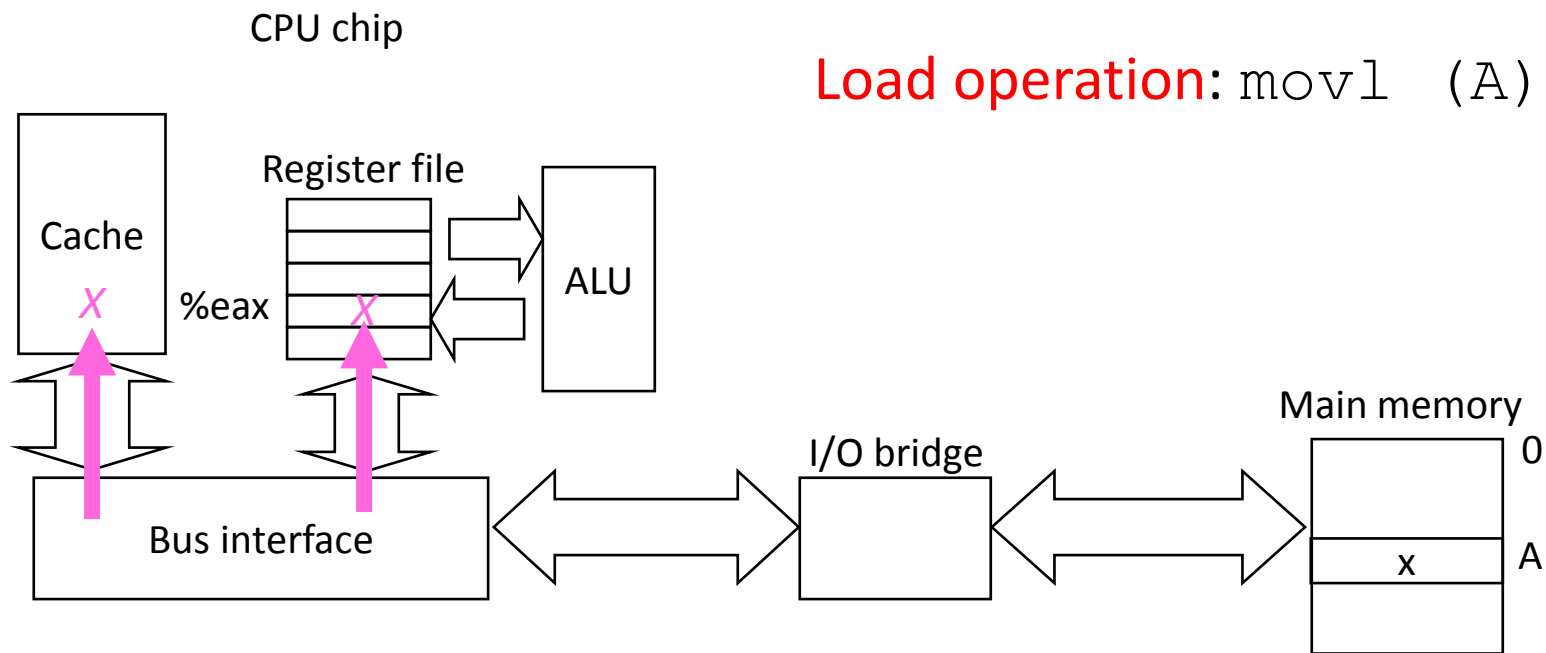
**Load operation:** `movl (A), %eax`



# Read (cont.)

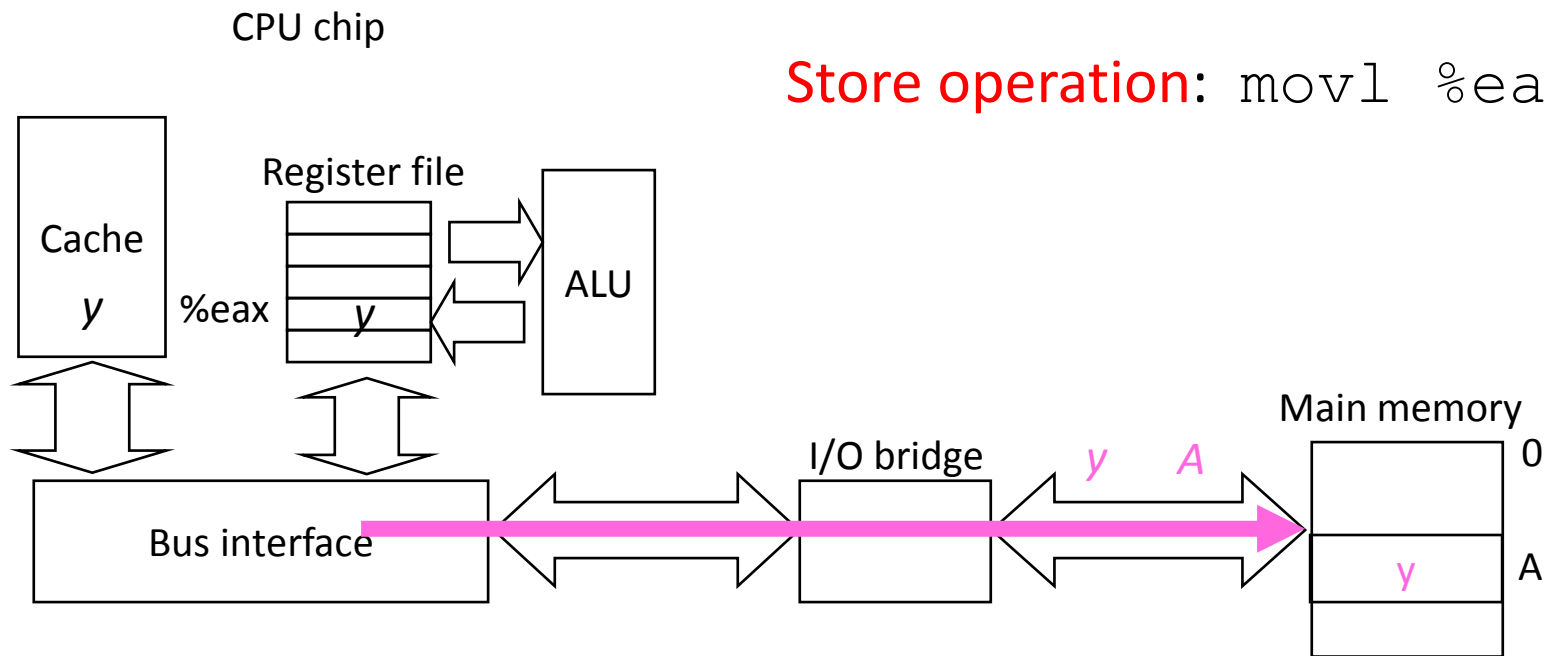
(3) CPU reads  $X$  from the bus, and copies it into register `%eax`, a copy also goes into the on-chip cache memory

**Load operation:** `movl (A), %eax`



# Write

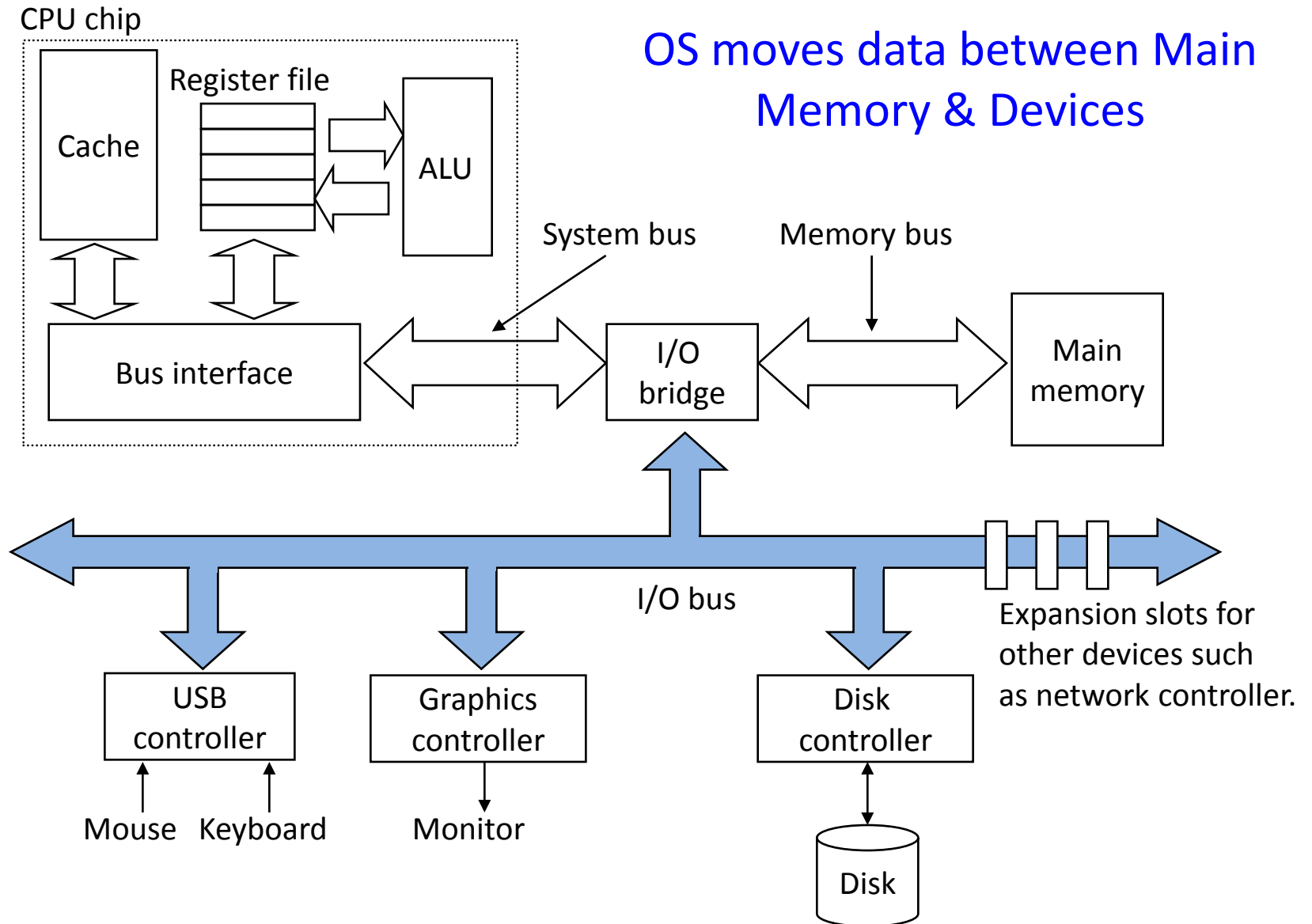
1. CPU writes  $A$  to bus, Memory Reads it
2. CPU writes  $y$  to bus, Memory Reads it
3. Memory stores read value,  $y$ , at address  $A$



# Secondary Storage

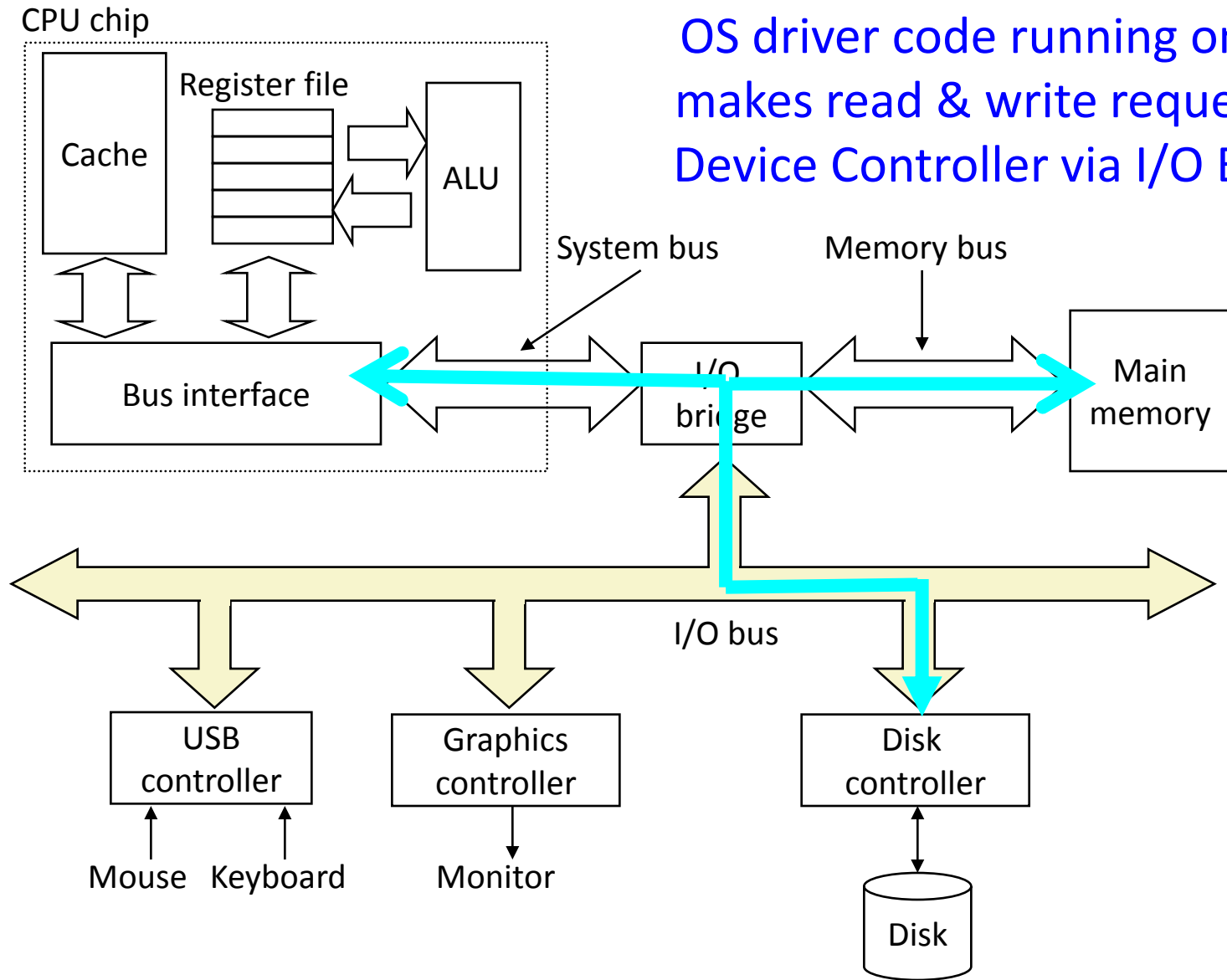
- Disk, Tape Drives, Flash Solid State Drives, ...
- Non-volatile: retains data without a charge
- Instructions CANNOT directly access data on secondary storage
  - No way to specify a disk location in an instruction
  - Operating System moves data to/from memory

# I/O Bus: connects Devices & Memory



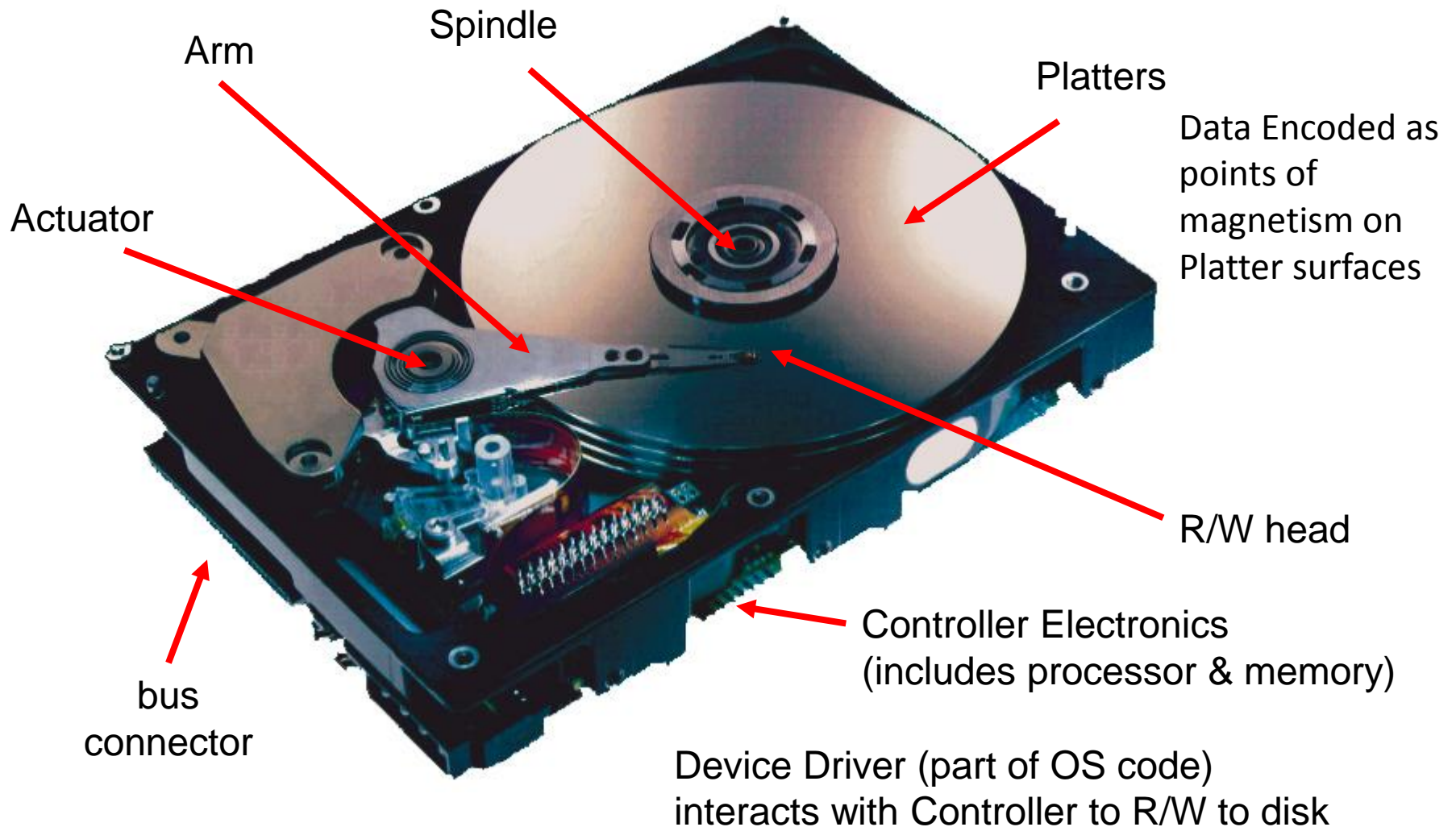


# Device Driver: OS device-specific code



OS driver code running on CPU makes read & write requests to Device Controller via I/O Bridge

# What's Inside A Disk Drive?

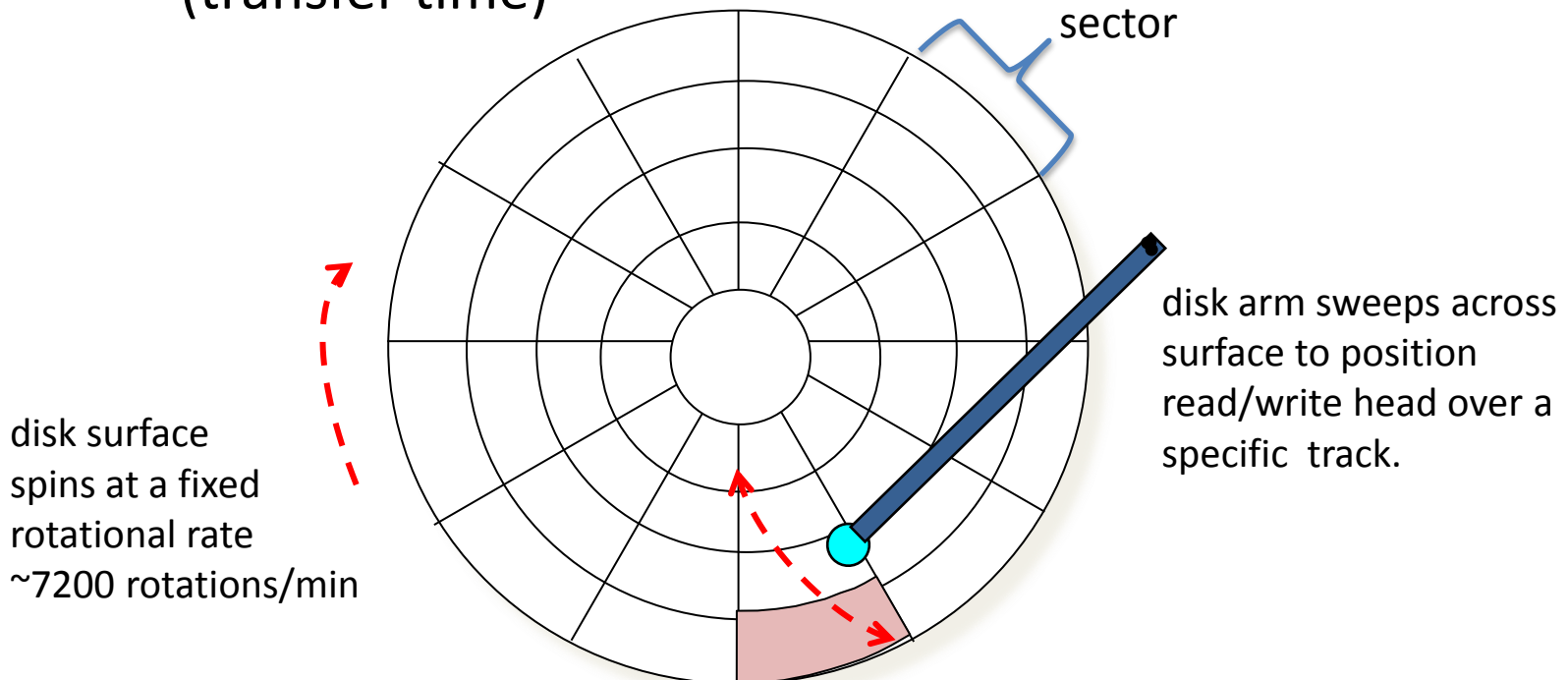


*Image from Seagate Technology*

# Reading and Writing to Disk

Data blocks located in some **Sector** of some **Track** on some **Surface**

1. Disk Arm moves to correct **track** (seek time)
2. Wait for **sector** spins under R/W head (rotational latency)
3. As sector spins under head, data are Read or Written (transfer time)



# Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, \$2000 – \$5000 per GB

Like walking:

Down the hall


- Dynamic RAM (DRAM)
  - 50ns – 70ns, \$20 – \$75 per GB

Across campus

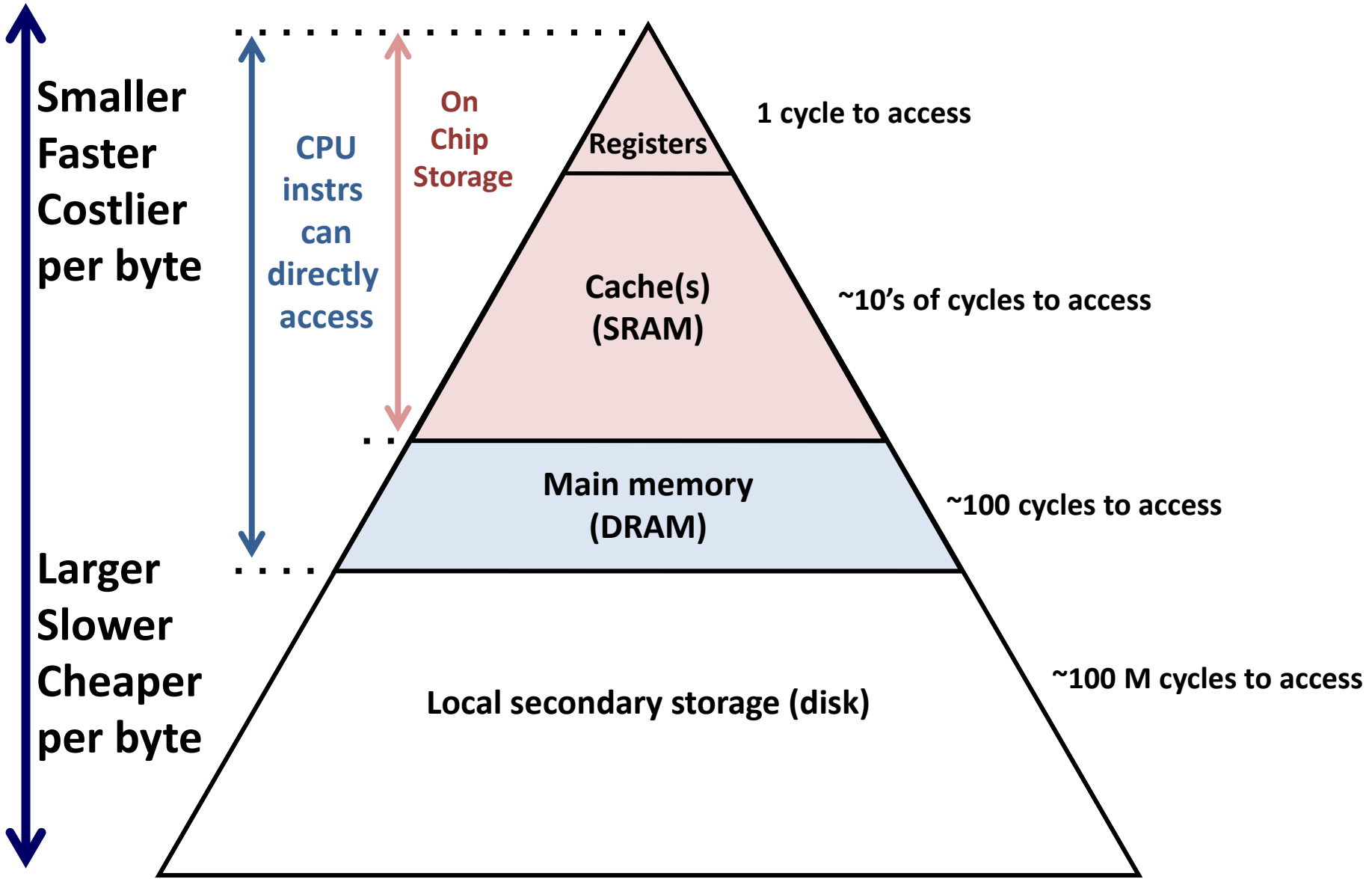
Solid-state disks (flash): 100 us – 1 ms, \$2 - \$10 per GB

- Magnetic disk
  - 5ms – 20ms, \$0.20 – \$2 per GB

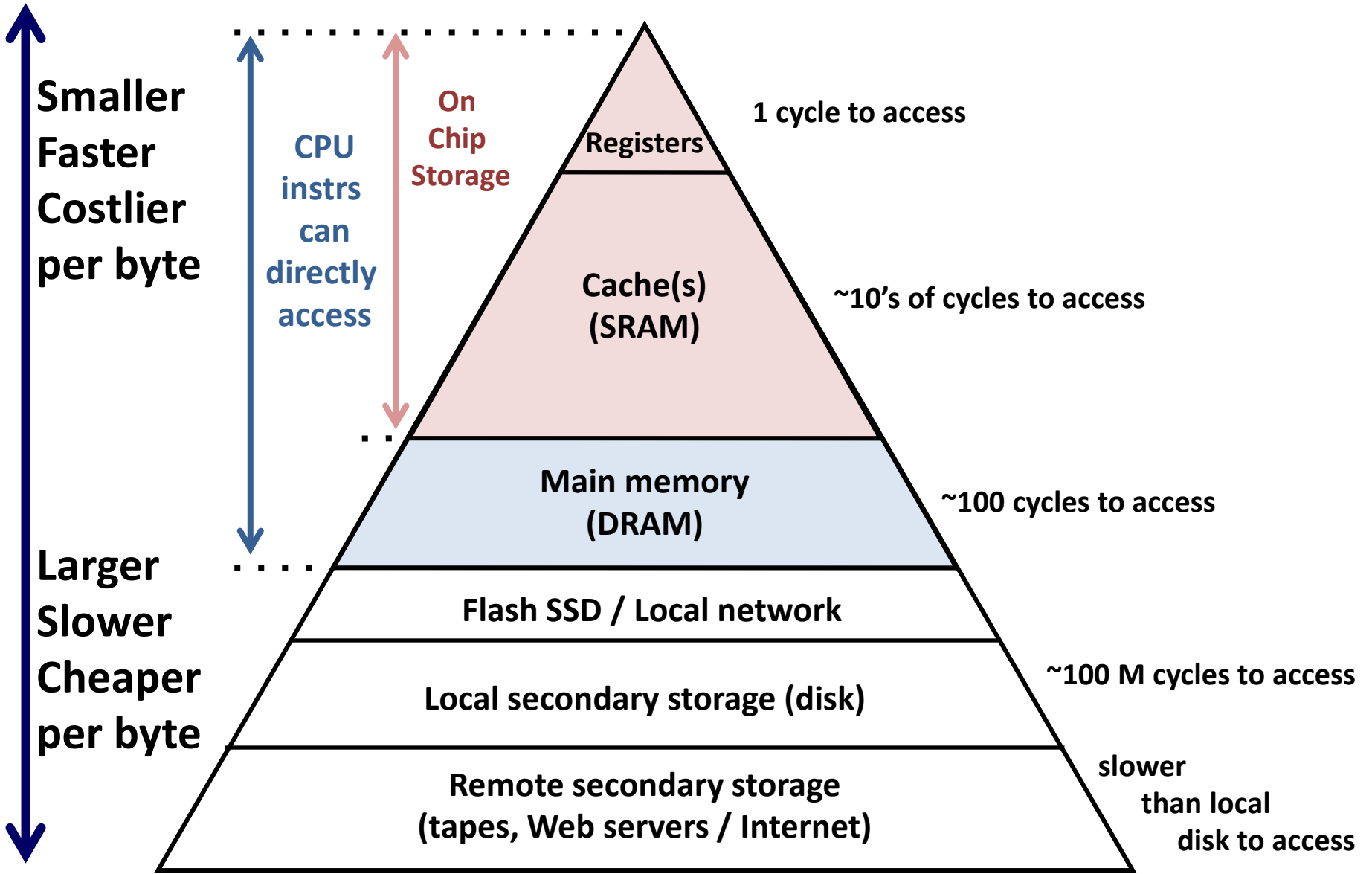
To Alaska

 1 ms == 1,000,000 ns

# The Memory Hierarchy



# The Memory Hierarchy



# Abstraction Goal

- Reality: There is no one type of memory to rule them all!
- Abstraction: hide the complex/undesirable details of reality.
- Illusion: We have the speed of SRAM, with the capacity of disk, at reasonable cost.

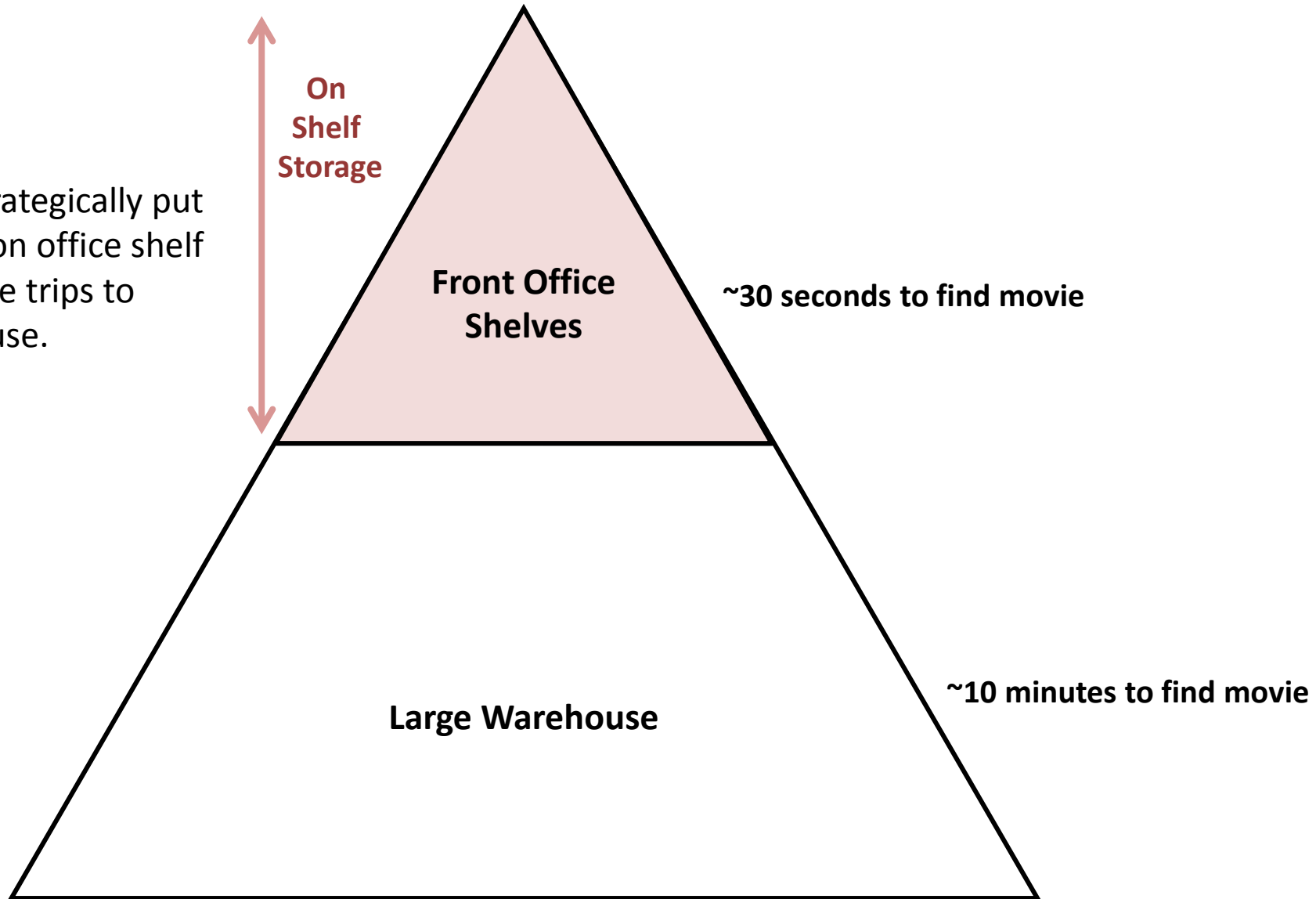
# Motivating Story / Analogy

- You work at a video rental store (remember Blockbuster?)
- You have a huge warehouse of movies
  - 10-15 minutes to find movie, bring to customer
  - Customers don't like waiting...
- You have a small office in the front with shelves, you choose what goes on shelves
  - < 30 seconds to find movie on shelf



# The Video Store Hierarchy

**Goal:** strategically put movies on office shelf to reduce trips to warehouse.



Quick vote: Which movie should we place on the shelf for tonight?

- A. Eternal Sunshine of the Spotless Mind
- B. The Godfather
- C. Pulp Fiction
- D. Rocky V
- E. There's no way for us to know.

# Problem: Prediction

- We can't know the future...
- So... are we out of luck?  
What might we look at to help us decide?
- The past is often a pretty good predictor...

# Repeat Customer: Bob

- Has rented “Eternal Sunshine of the Spotless Mind” ten times in the last two weeks.
- You talk to him:
  - He just broke up with his girlfriend
  - Swears it will be the last time he rents the movie (he’s said this the last six times)

Quick vote: Which movie should we place on the shelf for tonight?

- A. Eternal Sunshine of the Spotless Mind
- B. The Godfather
- C. Pulp Fiction
- D. Rocky V
- E. There's no way for us to know.

# Repeat Customer: Alice

- Alice rented Rocky a month ago
- You talk to her:
  - She's got a thing for Sylvester Stalone
- Over the next few weeks she rented:
  - Rocky II, Rocky III, Rocky IV

Quick vote: Which movie should we place on the shelf for tonight?

- A. Eternal Sunshine of the Spotless Mind
- B. The Godfather
- C. Pulp Fiction
- D. Rocky V
- E. There's no way for us to know.

# Critical Concept: Locality

- Locality: we tend to repeatedly access recently accessed items, or those that are nearby.
- Temporal locality: An item accessed recently is likely to be accessed again soon. (Bob)
- Spatial locality: We're likely to access an item that's nearby others we just accessed. (Alice)



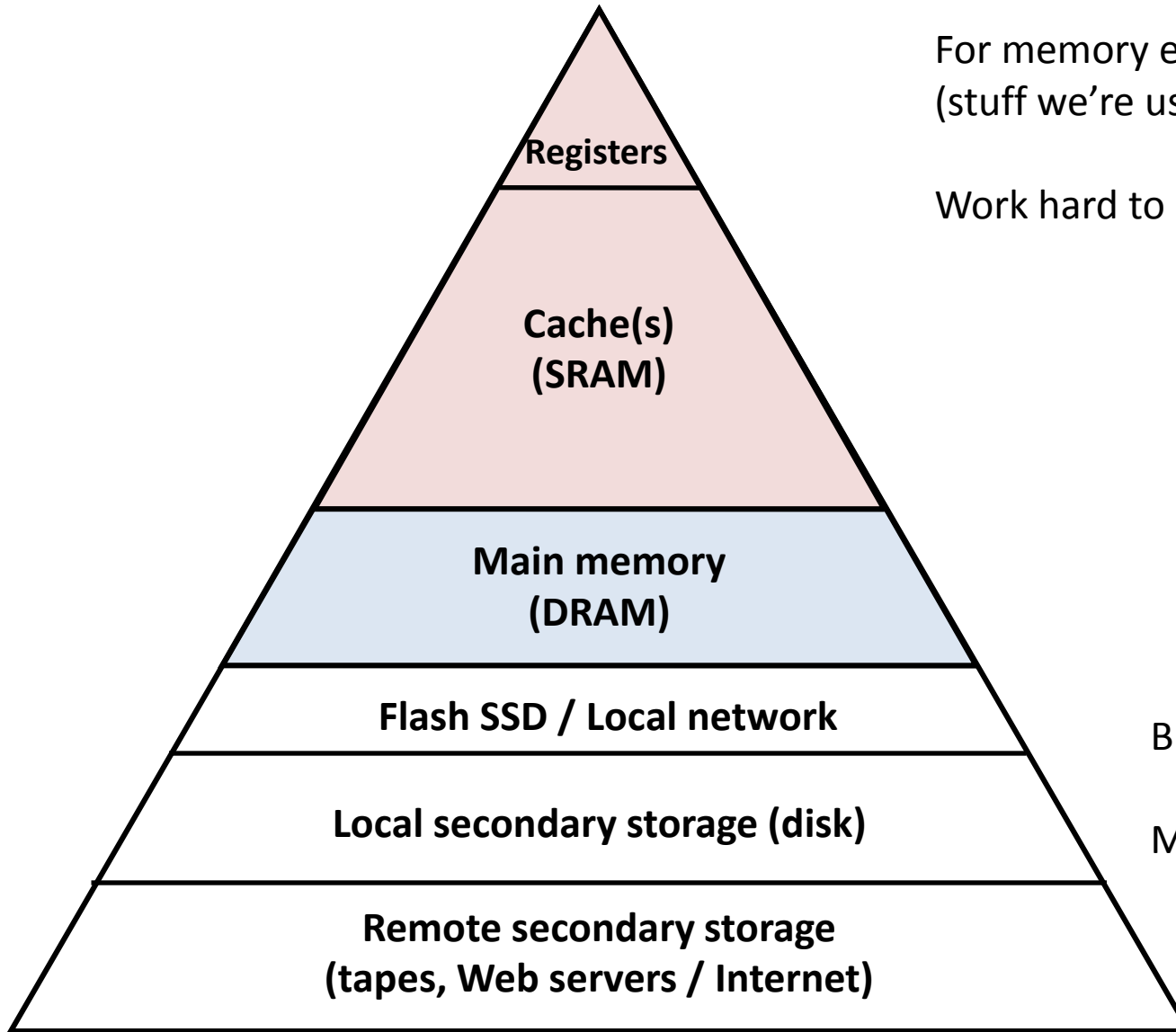
In the following code, how many examples are there of temporal / spatial locality?

Where are they?

```
int i;  
int num = read_int_from_user();  
int *array = create_random_array(num);  
for (i = 0; i < num; i++) {  
    printf("At index %d, value: %d", i, array[i]);  
}
```

- A. 1 temporal, 1 spatial
- B. 1 temporal, 2 spatial
- C. 2 temporal, 1 spatial
- D. 2 temporal, 2 spatial
- E. Some other number

# Big Picture



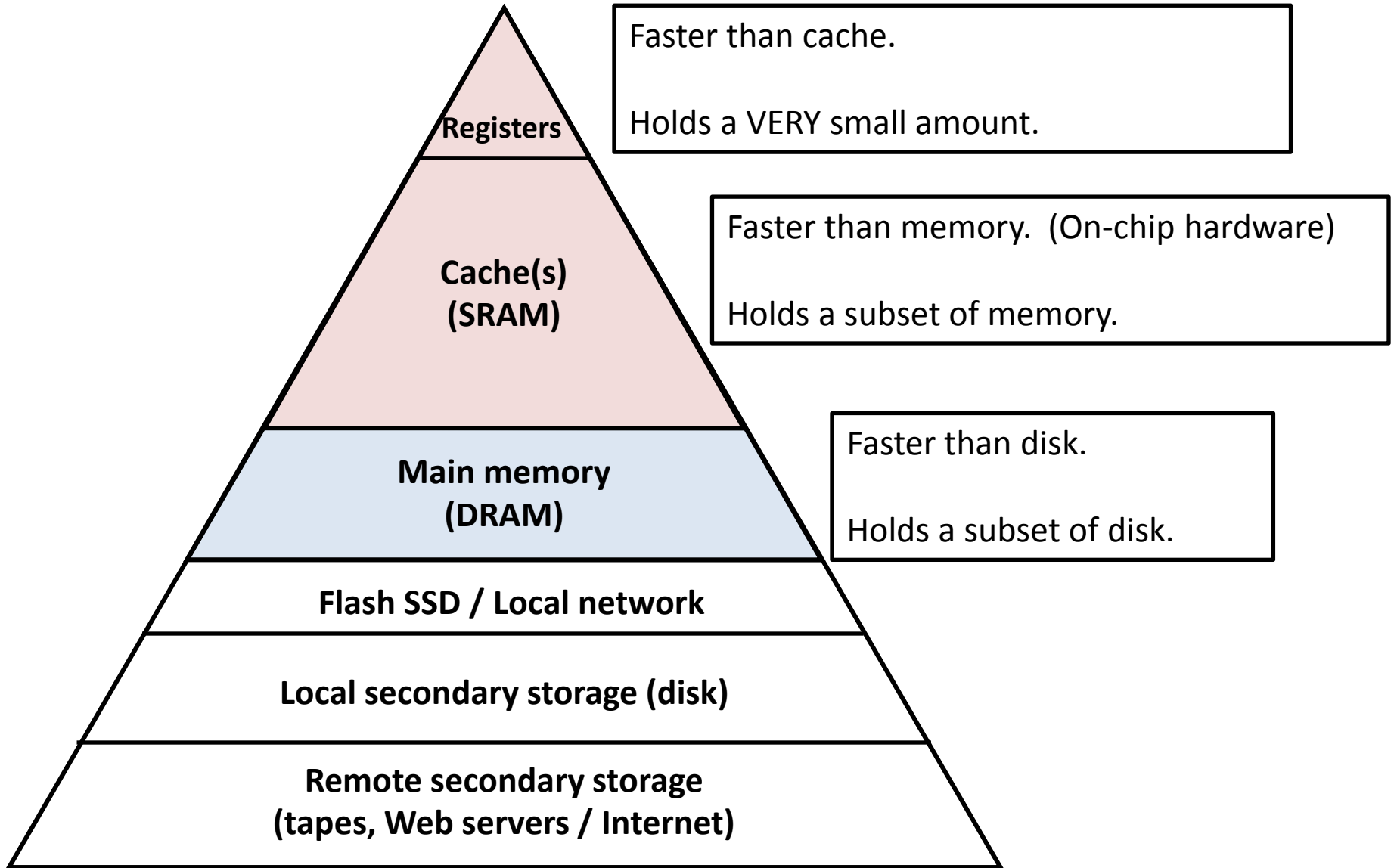
For memory exhibiting locality  
(stuff we're using / likely to use):

Work hard to keep them up here!

Bulk storage down here.

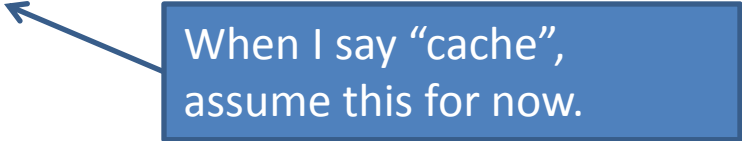
Move this up on demand.

# Big Picture



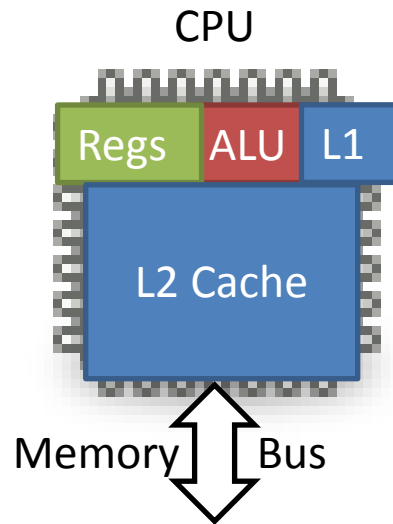
# Cache

- In general: a storage location that holds a subset of a larger memory, faster to access
- CPU cache: an SRAM on-chip storage location that holds a subset of DRAM main memory (10-50x faster to access)
- Goal: choose the right subset, based on past locality, to achieve our abstraction

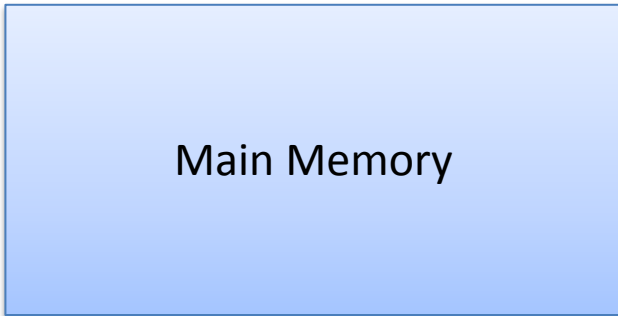


When I say “cache”,  
assume this for now.

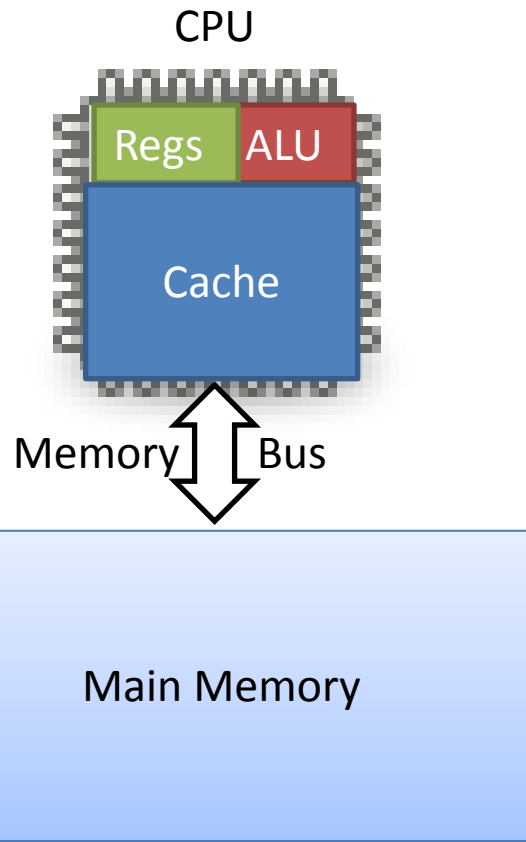
# Cache Basics



- CPU real estate dedicated to cache
- Usually two levels:
  - L1: smallest, fastest
  - L2: larger, slower
- Same rules apply:
  - L1 subset of L2



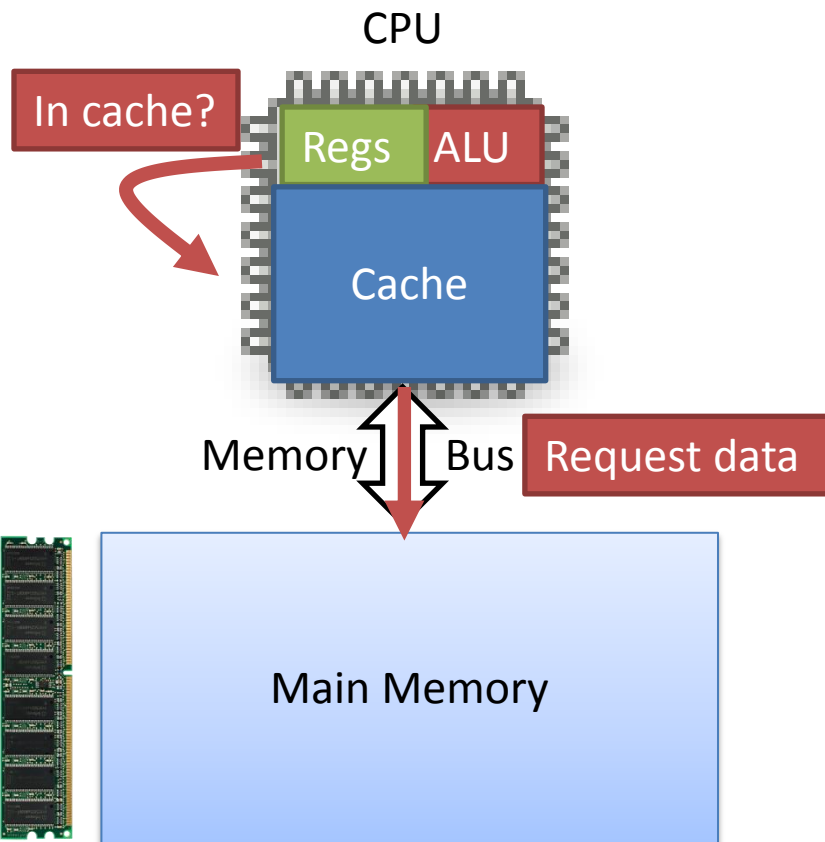
# Cache Basics



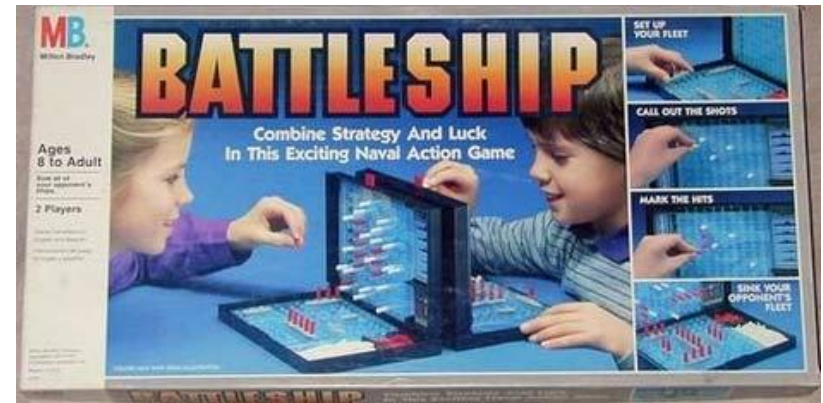
- CPU real estate dedicated to cache
- Usually two levels:
  - L1: smallest, fastest
  - L2: larger, slower
- We'll assume one cache (same principles)

Cache is a subset of main memory.  
(Not to scale, memory much bigger!)

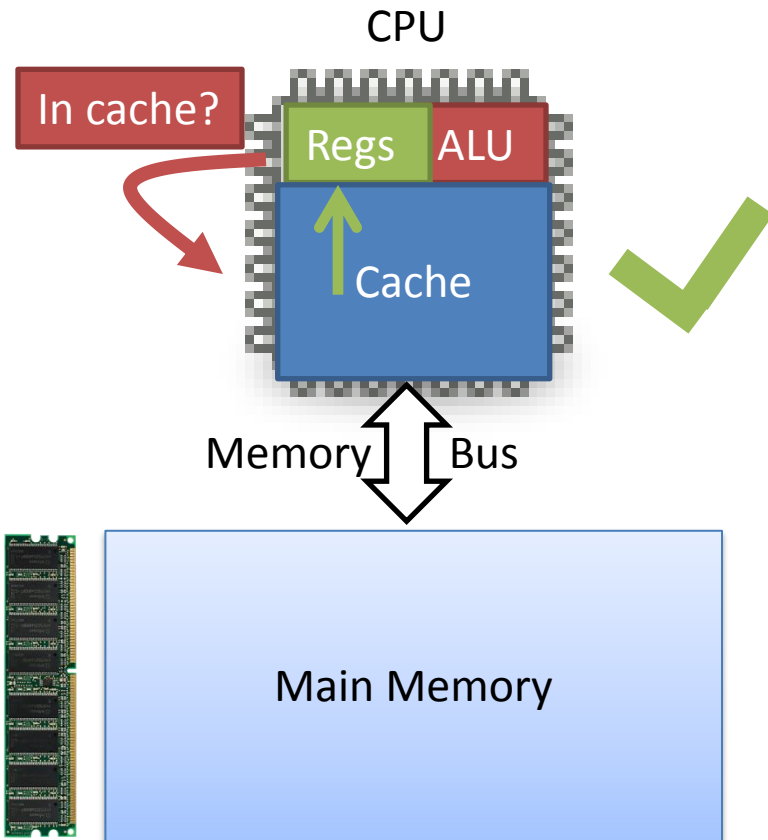
# Cache Basics: Read from memory



- In parallel:
  - Issue read to memory
  - Check cache



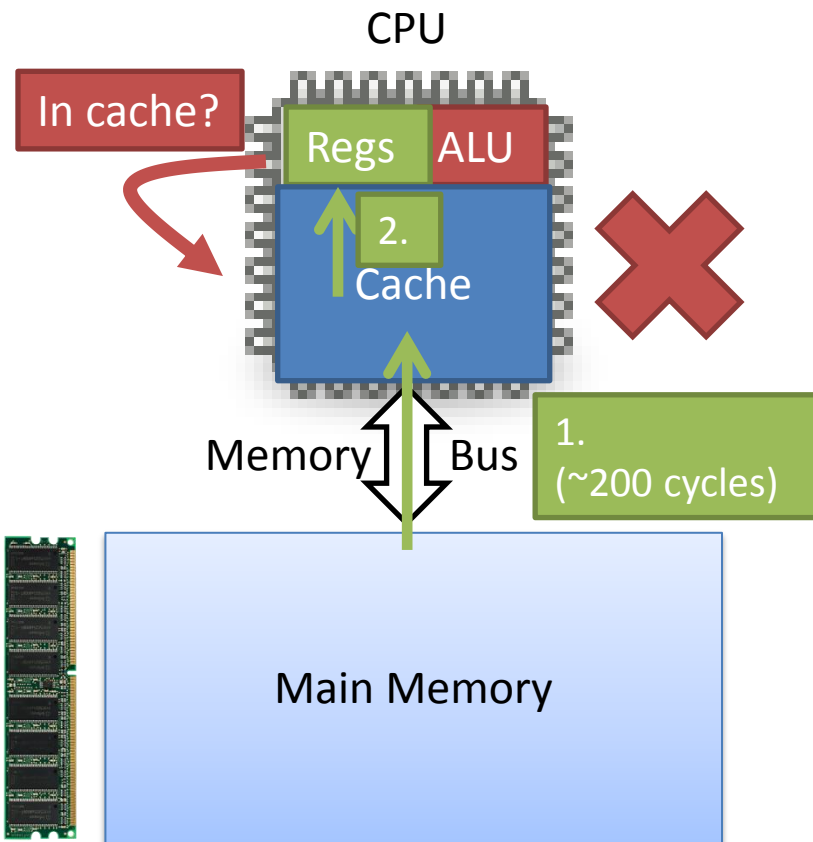
# Cache Basics: Read from memory



- In parallel:
  - Issue read to memory
  - Check cache
- Data in cache (hit):
  - Good, send to register
  - Cancel/ignore memory

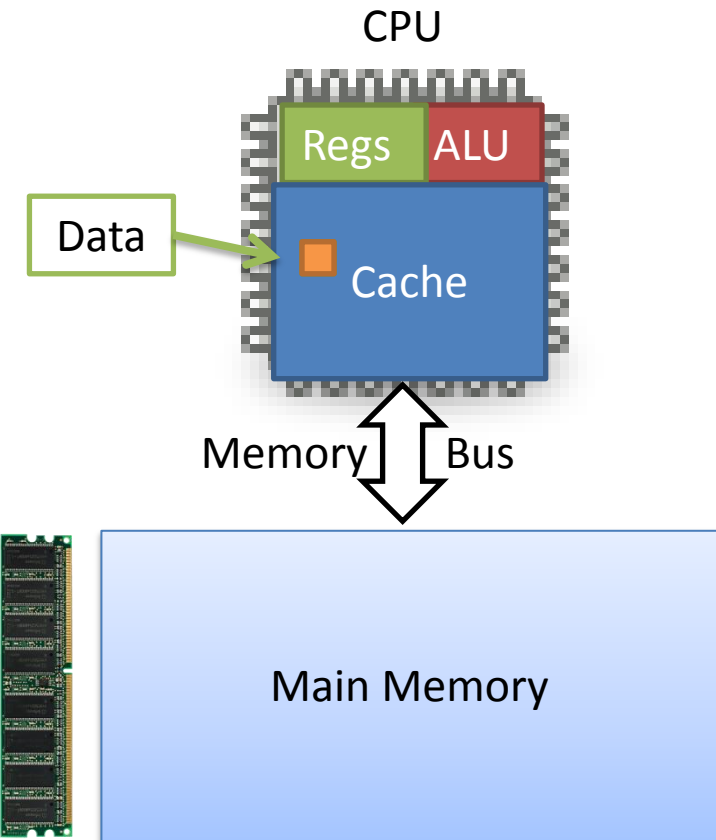


# Cache Basics: Read from memory



- In parallel:
  - Issue read to memory
  - Check cache
- Data in cache (hit):
  - Good, send to register
  - Cancel/ignore memory
- Data not in cache (miss):
  1. Load cache from memory (might need to evict data)
  2. Send to register

# Cache Basics: Write to memory



- Assume data already cached
    - Otherwise, bring it in like read
1. Update cached copy.
  2. Update memory?

# When should we copy the written data from cache to memory? Why?

- A. Immediately update the data in memory when we update the cache.
- B. Update the data in memory when we evict the data from the cache.
- C. Update the data in memory if the data is needed elsewhere (e.g., another core).
- D. Update the data in memory at some other time. (When?)

# When should we copy the written data from cache to memory? Why?

- A. Immediately update the data in memory when we update the cache. (“Write-through”)
- B. Update the data in memory when we evict the data from the cache. (“Write-back”)
- C. Update the data in memory if the data is needed elsewhere (e.g., another core).
- D. Update the data in memory at some other time. (When?)

# Cache Basics: Write to memory

- Both options (write-through, write-back) viable
- write-through: write to memory immediately
  - simpler, accesses memory more often (slower)
- write-back: only write to memory on eviction
  - complex (cache inconsistent with memory)
  - potentially reduces memory accesses (faster)

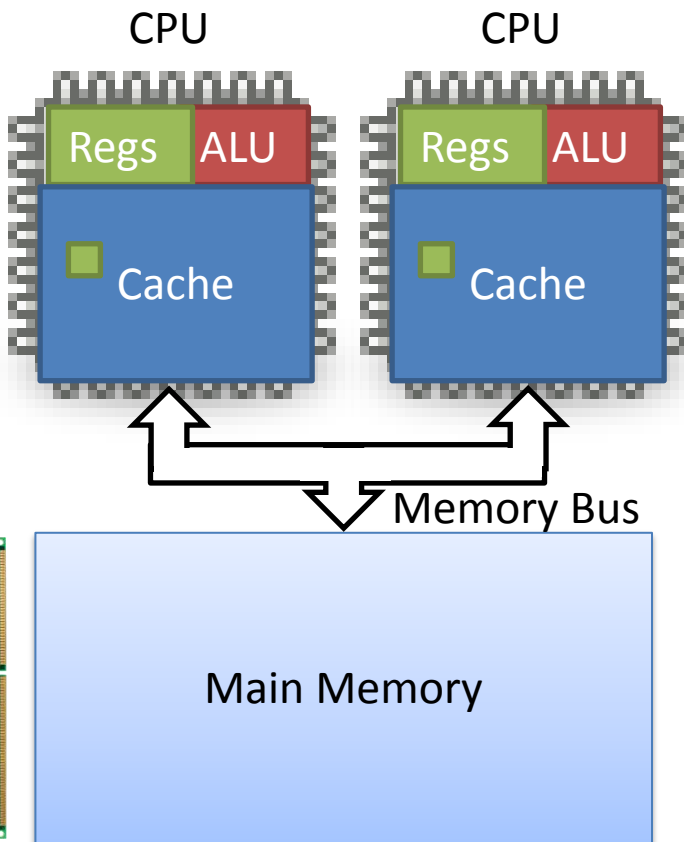
# Cache Basics: Write to memory

- Both options (write-through, write-back) viable
- write-through: write to memory immediately
  - simpler, accesses memory more often (slower)
- write-back: only write to memory on eviction
  - complex (cache inconsistent with memory)
  - potentially reduces memory accesses (faster)

Sells better.  
Servers/Desktops/Laptops

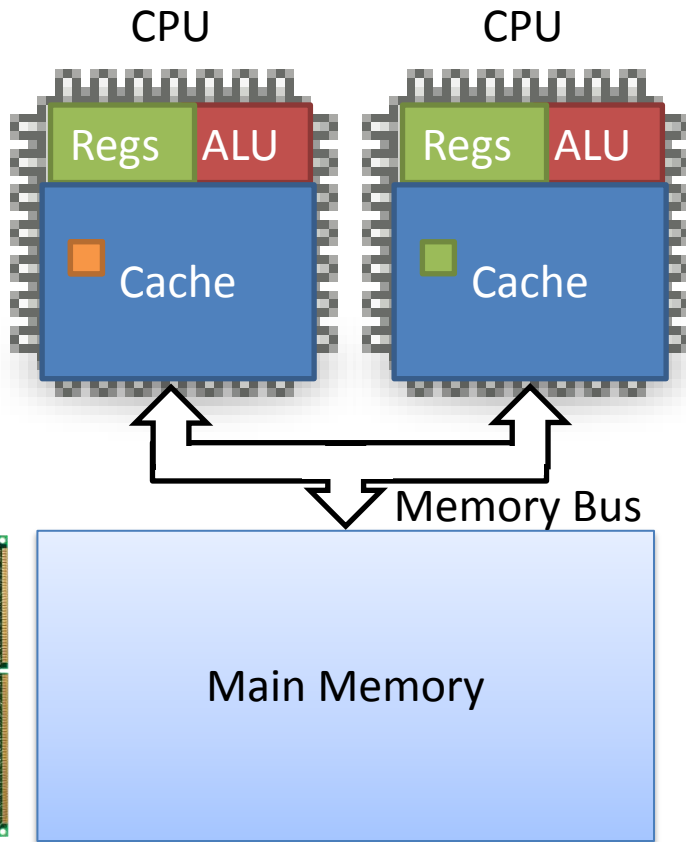


# Cache Coherence



- Keeping multiple cores' memory consistent

# Cache Coherence



- Keeping multiple cores' memory consistent
- If one core updates data
  - Copy data directly from one cache to the other.
  - Avoid (slower) memory
- Lots of HW complexity here. We might discuss towards end of semester.



# Up next:

- Cache details
- How cache is organized
  - finding data
  - storing data
- How cached subset is chosen (eviction)