Agenda

Memory hierarchy

Memory characteristics

Storage devices

A mistake to avoid

Locality

Memory Hierarchy

Problem: There's no single type of memory that is both high capacity and high performance

The **memory hierarchy** arranges memory in terms of performance and capacity

- This layered arrangement helps leverage the advantages of different types of memory while maintaining performance

Programmers generally do not need to be aware of the memory hierarchy; however, there are some key mistakes to avoid (will talk about later)

Memory Hierarchy

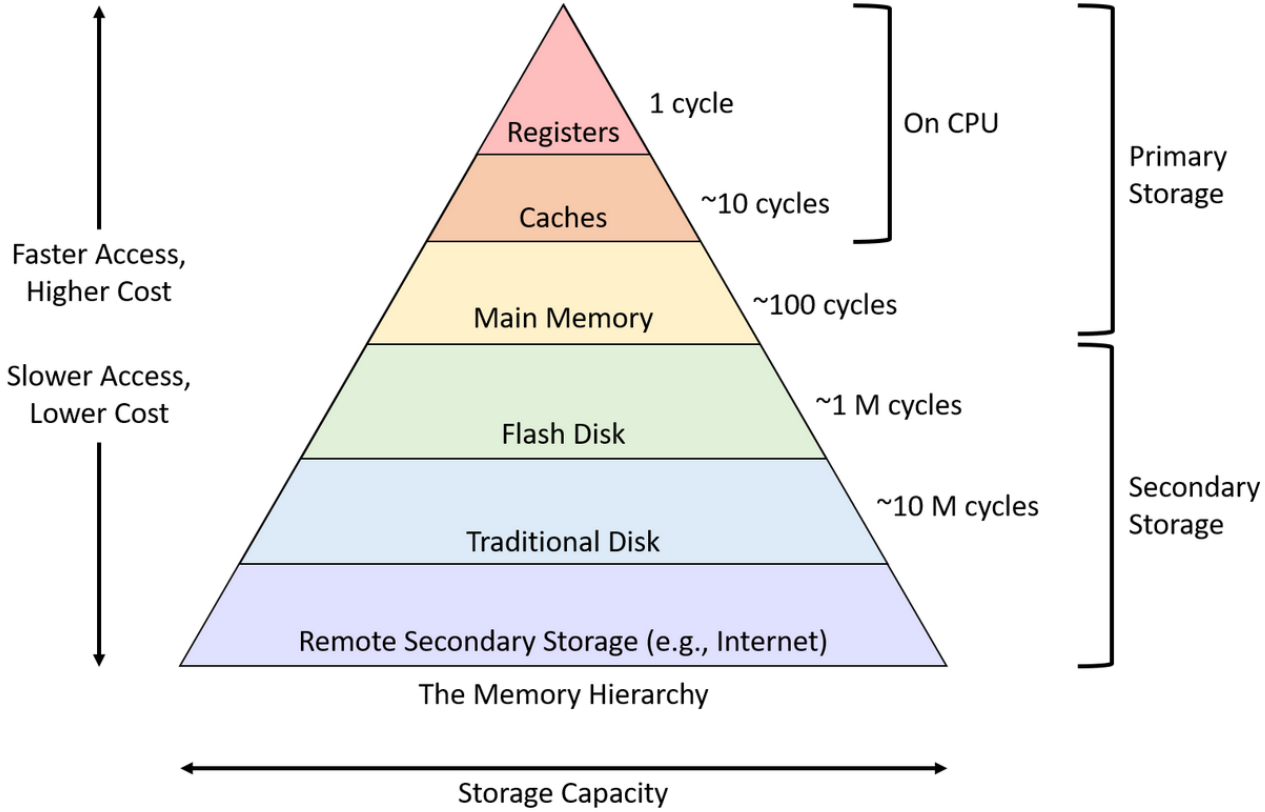

Figure 1. The memory hierarchy

Memory Hierarchy: Analogy with books

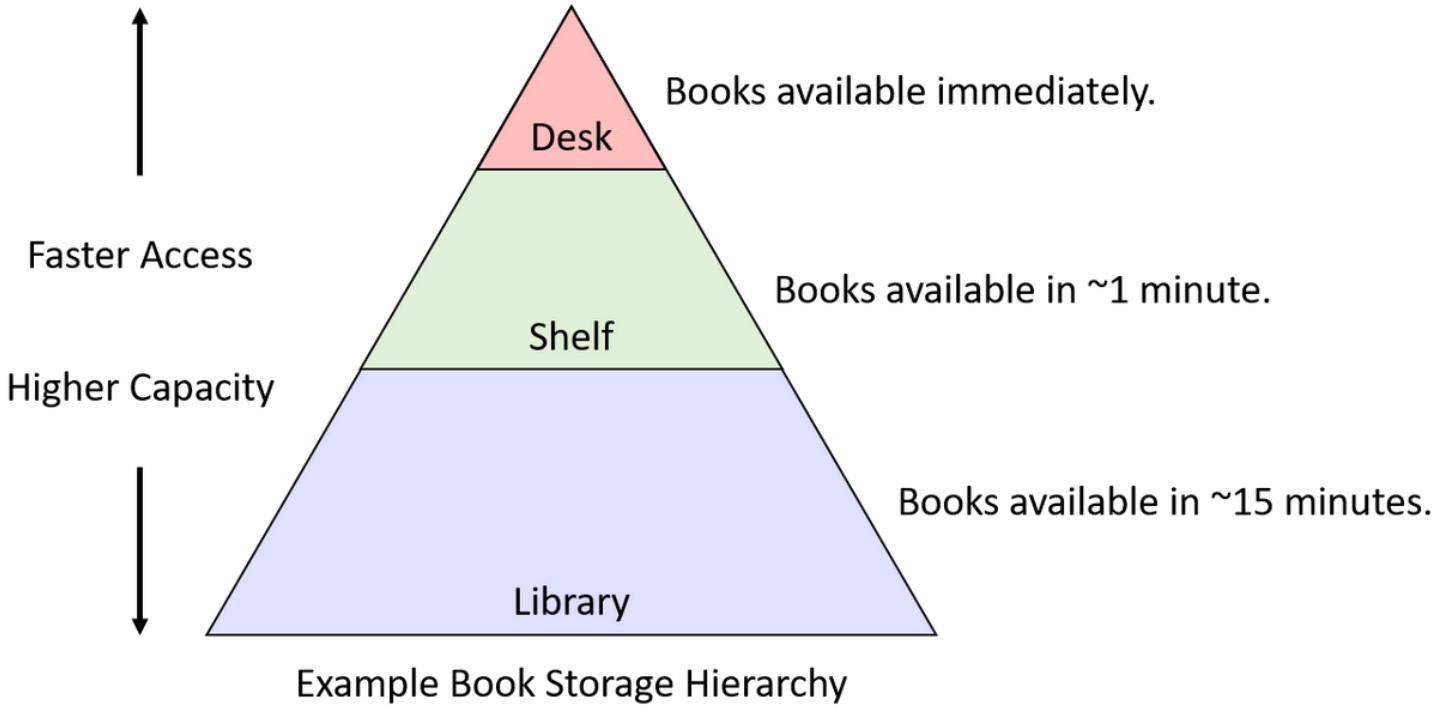

Figure 1. A hypothetical book storage hierarchy

Memory characteristics

Capacity How much data can be stored?

Latency How fast can we get the data?

Transfer rate How much data can be moved per second?

Storage devices

primary storage can be accessed directly by the CPU

- needs power to hold data
- Less capacity
- Faster access

secondary storage must be copied into primary storage before it can be used

- persistent, keeps data even when turned off
- More capacity
- Slower access

Primary storage

random access memory (RAM)

accessing data does not depend on the data's location on the device

Two types:

***static* RAM (SRAM)**

***dynamic* RAM(DRAM)**

Primary storage

Table 1. Primary Storage Device Characteristics of a Typical 2020 Workstation

Device	Capacity	Approx. latency	RAM type
Register	4 - 8 bytes	< 1 ns	SRAM
CPU cache	1 - 32 megabytes	5 ns	SRAM
Main memory	4 - 64 gigabytes	100 ns	DRAM

Primary storage

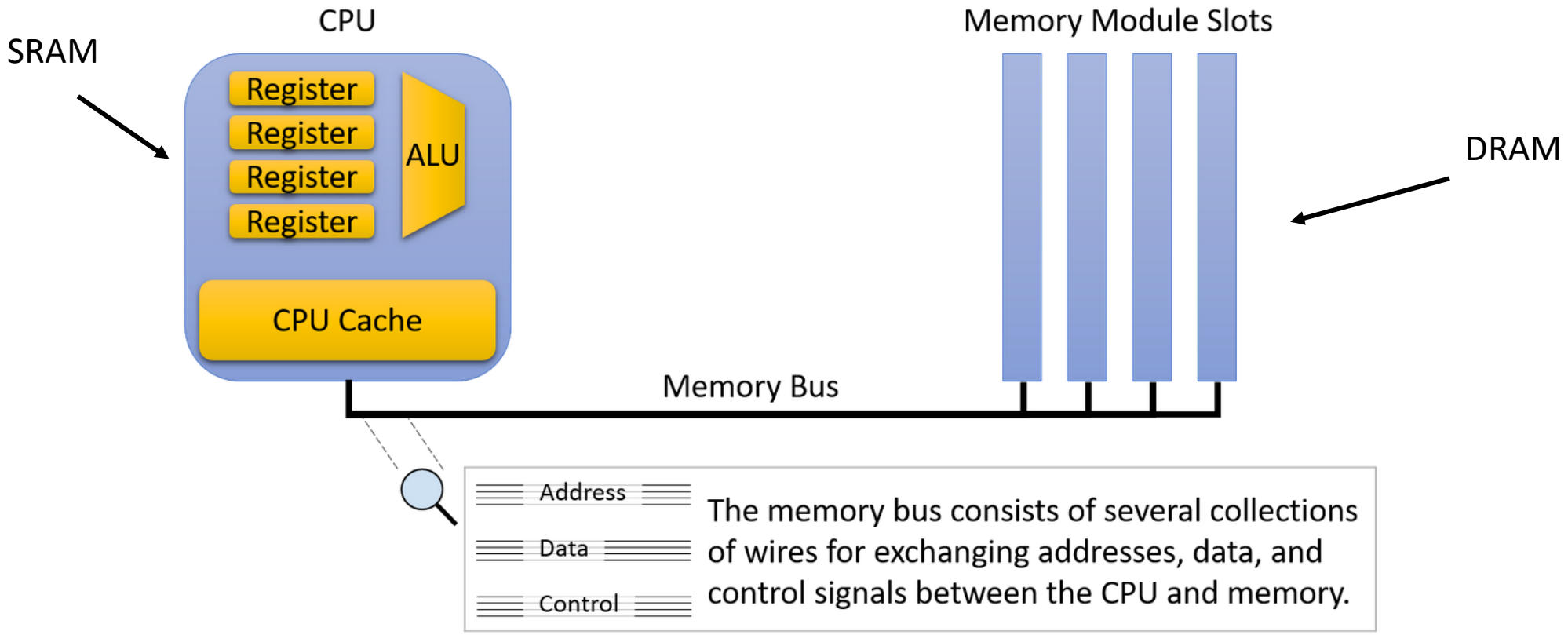

Figure 1. Primary storage and memory bus architecture

Secondary storage

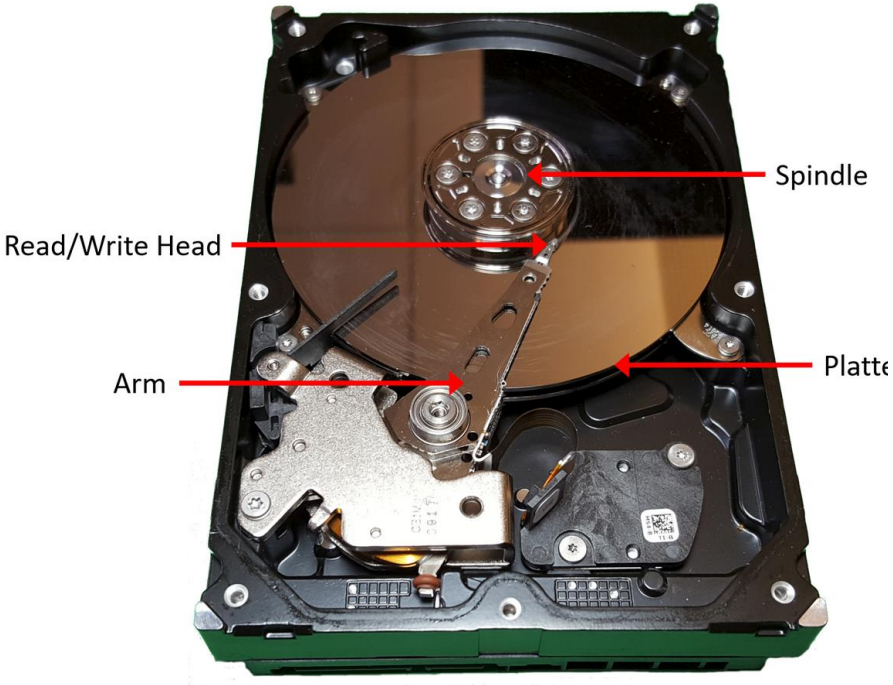

Figure 4. The major components of a hard disk drive

(a) (b) (c)

Figure 2. Example photos of (a) a punch card, (b) a magnetic tape spool, and (c) a variety of floppy disk sizes. Images from Wikipedia.

Secondary storage

Modern Secondary Storage

Table 2. Secondary Storage Device Characteristics of a Typical 2020 Workstation

Device	Capacity	Latency	Transfer rate
Flash disk	0.5 - 2 terabytes	0.1 - 1 ms	200 - 3,000 megabytes / second
Traditional hard disk	0.5 - 10 terabytes	5 - 10 ms	100 - 200 megabytes / second
Remote network server	Varies considerably	20 - 200 ms	Varies considerably

Secondary storage

Table 1. Primary Storage Device Characteristics of a Typical 2020 Workstation

Device	Capacity	Approx. latency	RAM type
Register	4 - 8 bytes	< 1 ns	SRAM
CPU cache	1 - 32 megabytes	5 ns	SRAM
Main memory	4 - 64 gigabytes	100 ns	DRAM

Modern Secondary Storage

Table 2. Secondary Storage Device Characteristics of a Typical 2020 Workstation

Device	Capacity	Latency	Transfer rate
Flash disk	0.5 - 2 terabytes	0.1 - 1 ms	200 - 3,000 megabytes / second
Traditional hard disk	0.5 - 10 terabytes	5 - 10 ms	100 - 200 megabytes / second
Remote network server	Varies considerably	20 - 200 ms	Varies considerably

Secondary storage

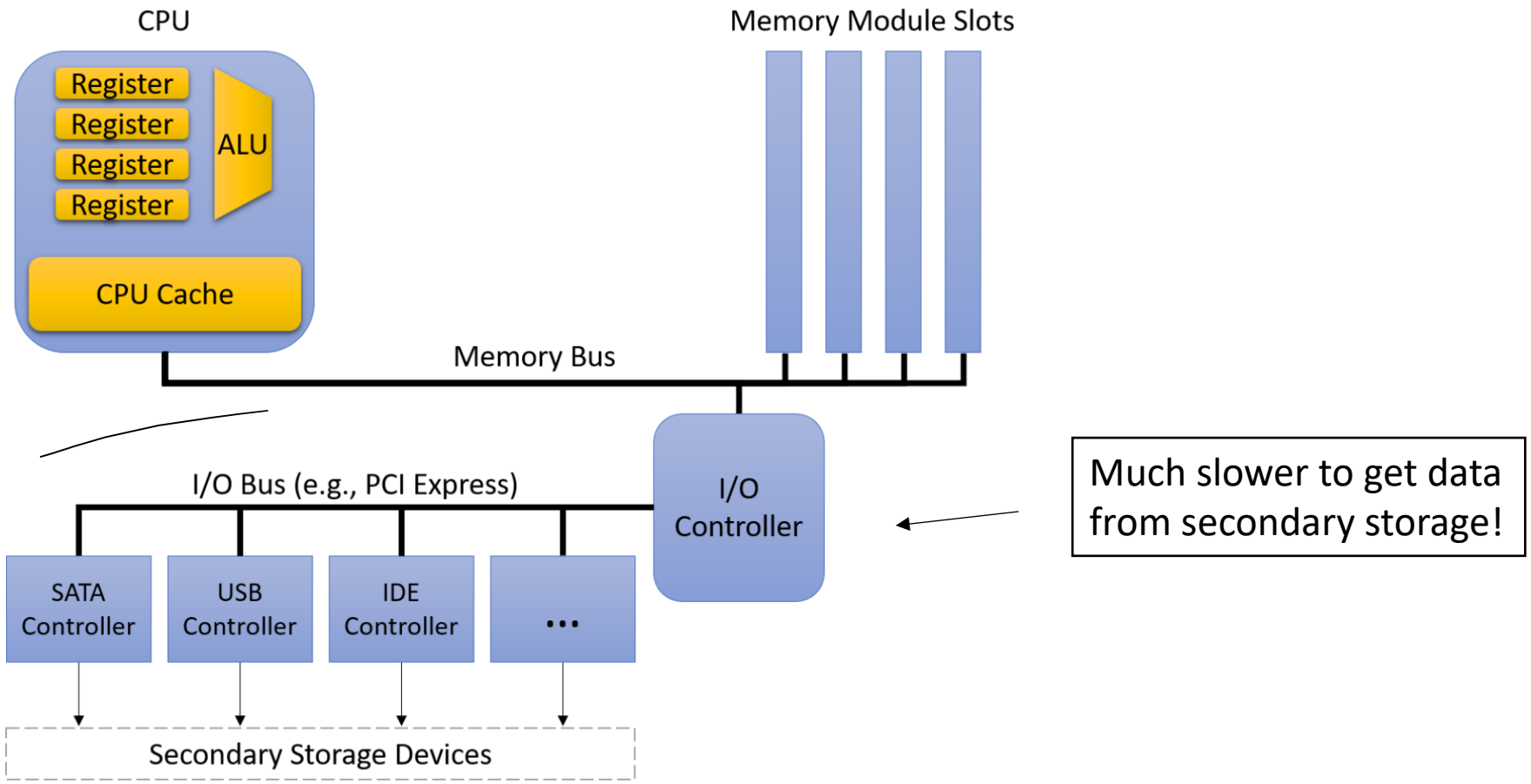

Figure 3. Secondary storage and I/O bus architecture

Secondary storage

Hard disk drives (HDD)

- Data is stored on actual disks (magnetic)
- Best when accessed sequentially

Solid-state drives (SSD)

- Consist of “cells”
- Same amount of time to access data regardless of the location on the drive

Implication of the Memory Hierarchy

Two algorithms with the same runtime performance can perform very differently in practice!

Example: What is the running time (order-N) of the following function?

```
float averageMat(int **mat, int n) {
  int i, j, total = 0;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      total += mat[i][j];
    }
  }

  return (float) total / (n * n);
}
```

Example: Average Matrix

Exercise: `matrix.c` creates a random matrix and then computes the average of its elements.

Run `matrix` with different sizes. Which version of `averageMatrix` is slower? How are the two functions different?

Example: Average Matrix

Program A

```
float averageMat_v1(int **mat, int n) {
  int i, j, total = 0;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      total += mat[i][j];
    }
  }

  return (float) total / (n * n);
}
```

Program B

```
float averageMat_v2(int **mat, int n) {
  int i, j, total = 0;
  for (i = 0; i < n; i++) {
    for (j = 0; j < n; j++) {
      total += mat[j][i];
    }
  }

  return (float) total / (n * n);
}
```

Which program is better and why?

Review: Row vs Column major order

Locality

Programs typically have common memory access patterns

Hardware optimized based on these patterns

Temporal locality

- Idea: Recently-used memory is likely to be used again, so keep frequently used memory close to the CPU

Spatial locality

- Idea: Memory tends to be accessed in sequence, so fetch memory in blocks, not one at a time

Programs that leverage locality run faster than ones that don’t

Locality: Example

```
/* Sum up the elements in an integer array of length len. */
int sum_array(int *array, int len) {
    int i;
    int sum = 0;

    for (i = 0; i < len; i++) {
        sum += array[i];
    }

    return sum;
}
```