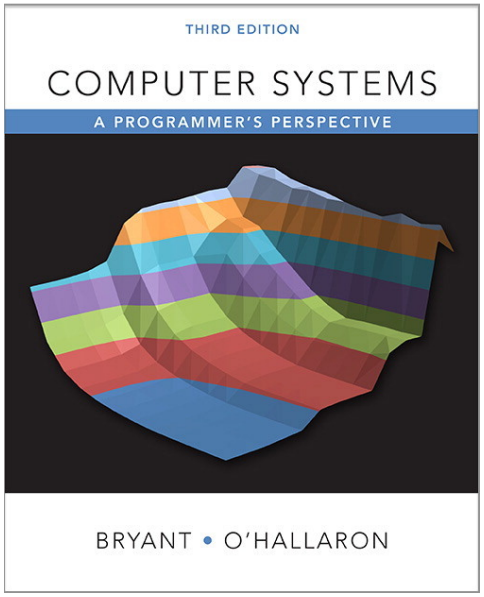
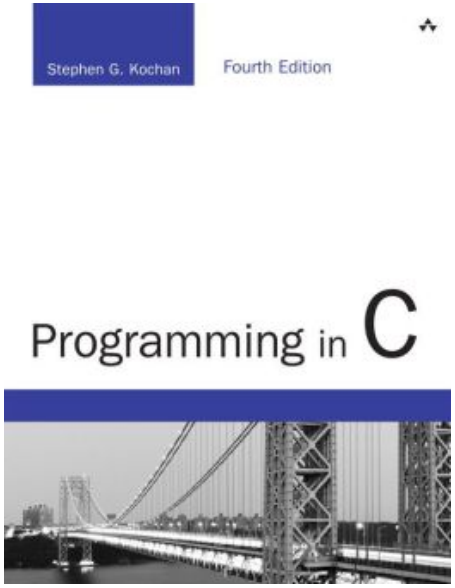
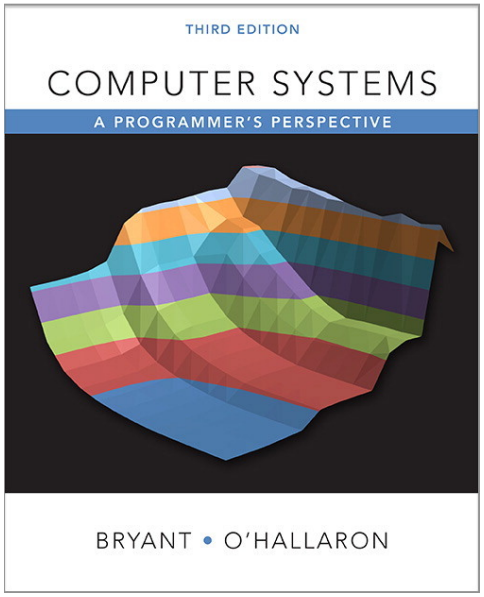
CSC 373 Sections 501,510 Winter 2020
Computer Systems I





Note: You must have the 3rd edition of Bryant and O'Hallaron text The Specific C text is unimportant. There are 100s of good Intro to C texts.

CSC 374 Computer Systems II
Follow-up to this course



Same text (Bryant and O'Hallaron)

Brief Outline of CSC 373/374

CSC 373: Computer Systems I

- Emphasis on **computer architecture**
  - Brief overview of CPU (Central Processing Unit) architecture and memory organization(Ch. 1)
  - Intro to C and Linux (C text)
  - Representation of data and programs (Ch. 2)
  - Assembly language (Ch. 3)
  - Compilation (i.e., how a C program maps to assembly language) (Ch. 3)
  - Security vulnerabilities (Ch. 3)
  - Organization of computer memory
    - Cache memory and caching (Ch. 6)

Brief Outline of CSC 373/374

CSC 374: Computer Systems II

- Emphasis on **operating systems**
- Program optimization
- Linking
- Processes
- Heap memory organization
- Virtual memory
- System i/o

Overall CSC Course Sequence

- **CSC 241-242 (Python)**
  - Python has a less steep learning curve than many other programming languages, such as Java or C/C++
  - Python has a wide range of easy-to-use tools
  - Enables us to show a wider variety of applications in an introductory course (e.g., GUIs, Web crawling, Database connectivity)

CSC Course Sequence

- **CSC 300-301 (Java):**
  - Java is the more commonly used language in industry; see for example the Tiobe report.
  - We want to expose you to different kinds of programing languages
  - Java is **pure object-oriented**, and a more strongly **typed** language
  - Compromise language between efficiency and programmability

- **CSC 373/374 (C/Linux):**
  - C is a language which exposes the programmer to low-level interaction with the architecture of a computer
  - For example, C has pointer types, bitwise operations, explicit memory management, etc. These are all features of the language that are not found in e.g., Java
  - C compiles into executable code, unlike many other languages which require an interpreter (e.g., Java byte code, or Python).
  - Therefore, debugging some C programs requires that the programmer know more about the architecture (i.e., assembly language) of a computer

CSC Course Sequence

- **CSC 373/374 (C/Linux):**
  - Similarly, Linux interacts with a programming language like C more easily that other operating systems such as Windows or Mac OS
  - C++ is a superset of C, but the additional portions of C++ (classes, objects, etc.) are not as relevant to systems courses.

C/Linux Programming Environment Options

- **Recommended**: cdmlinux.cdm.depaul.edu is a 64-bit Linux environment with gnu gcc and gdb tools
- You may prefer to install Ubuntu or some other Linux OS installed on your own computer
- Macs already have a variant of Linux, but may not be able to run assignments during the second half of the quarter

Comparison of technologies

- Several professional surveys are available, in which developers self-report on various aspects of their jobs
  - Example: the Stack Overflow survey
  - in particular,
    - most popular technologies
    - salary information

Our CSC sequence can be though of as a history of computer science

- **241-242 Python** language is “highest-level”; allows programmer to write code often without knowing the details of what’s going on. Example: list, set
- **300-301 Java** was developed earlier (1990s) and allows programmer to specify more of a program’s details. Example: LinkedList, ArrayList, HashSet, SortedSet

The C Programming Language

- **373-374 -- C**
- requires the user to manage much more of what is going on in a computer system (sometimes this is a good thing): pointers, register variables, memory management, bitwise operators. Much closer to what's actually going on under the hood.

Assembly Language

- **We will spare you the great pain of writing assembly language**. But you will learn how to understand it

Machine Code

- **The actual 0's and 1's that machines translate as instructions or data**

Some early computers

ENIAC: 1946

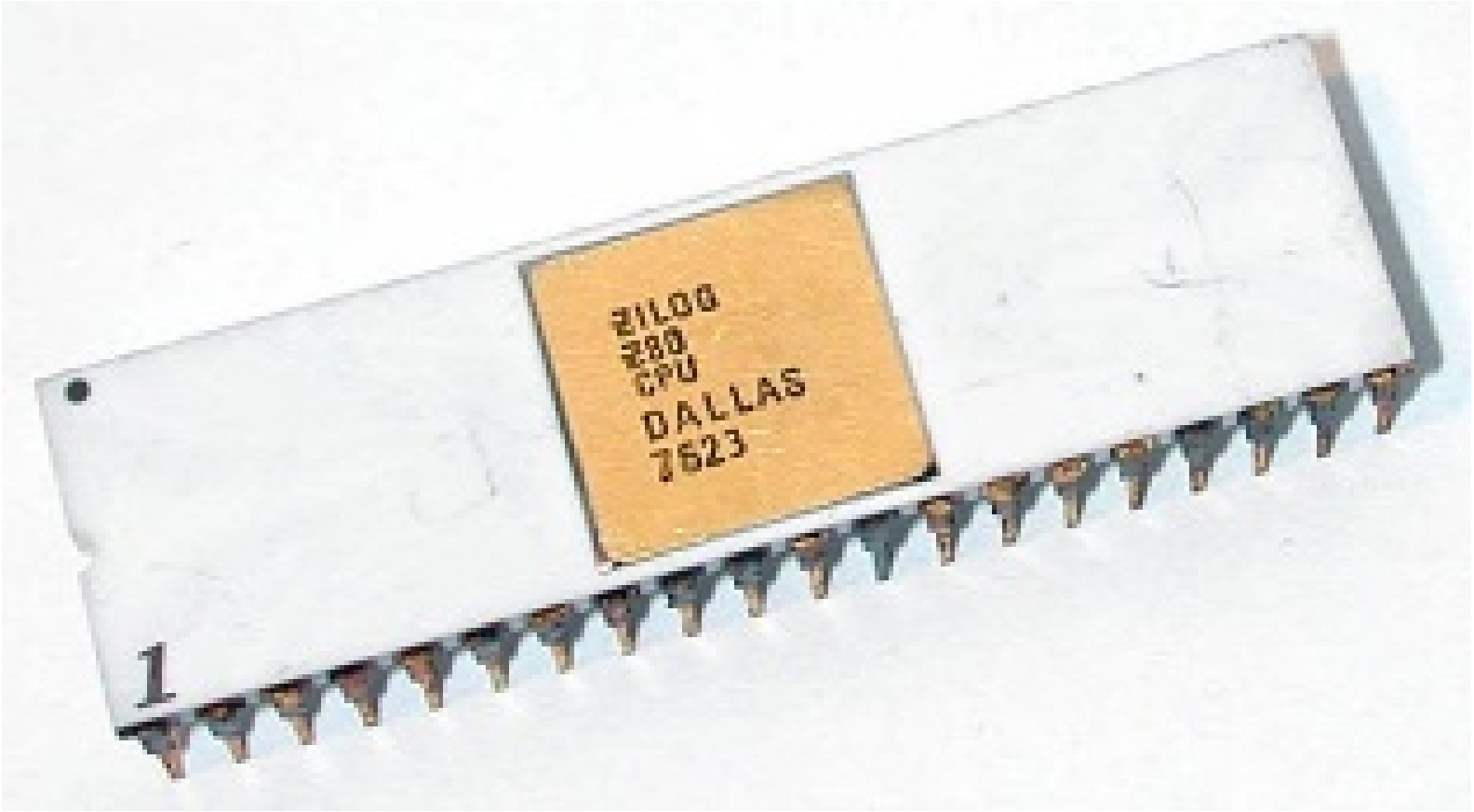
Some early computers

Univac I (1951)
Delivered to US Census bureau

Some early computers: IBM 360

1964 – first commercial successful computer to use integrated circuits. Also, transferred from punch cards to other input devices

Intel 8080 chip: 1976

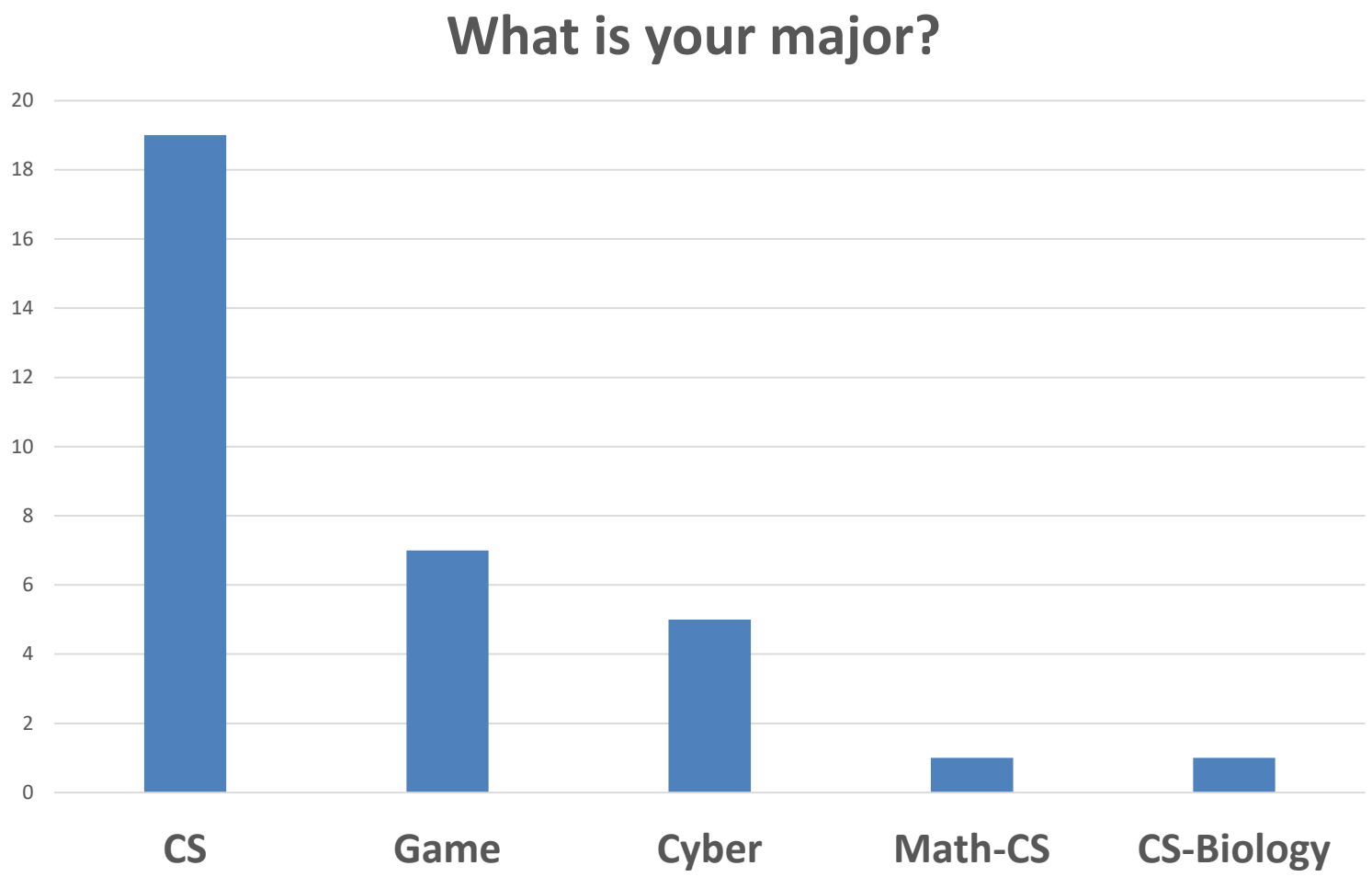


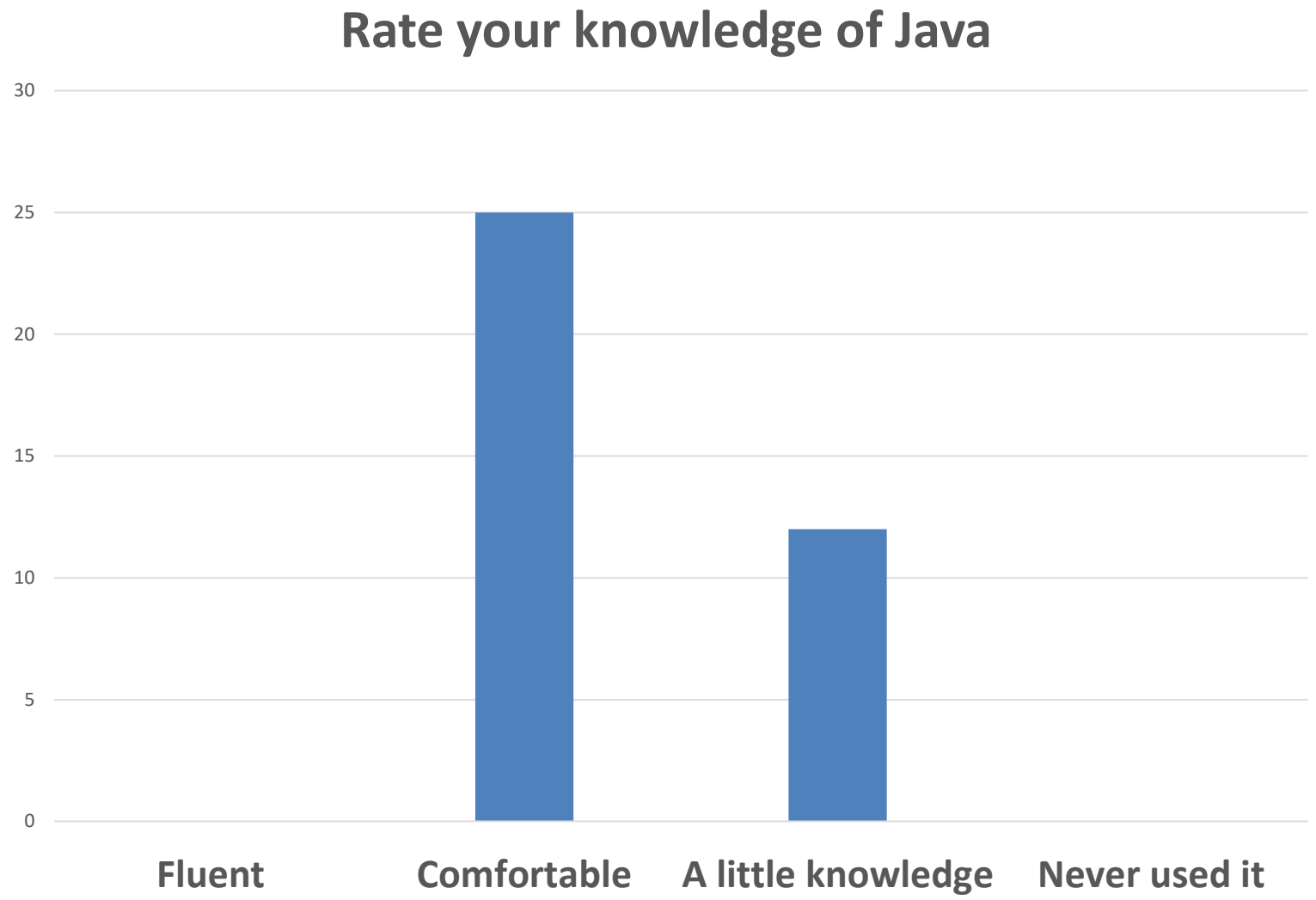
Could address 64 kilobytes of memory!

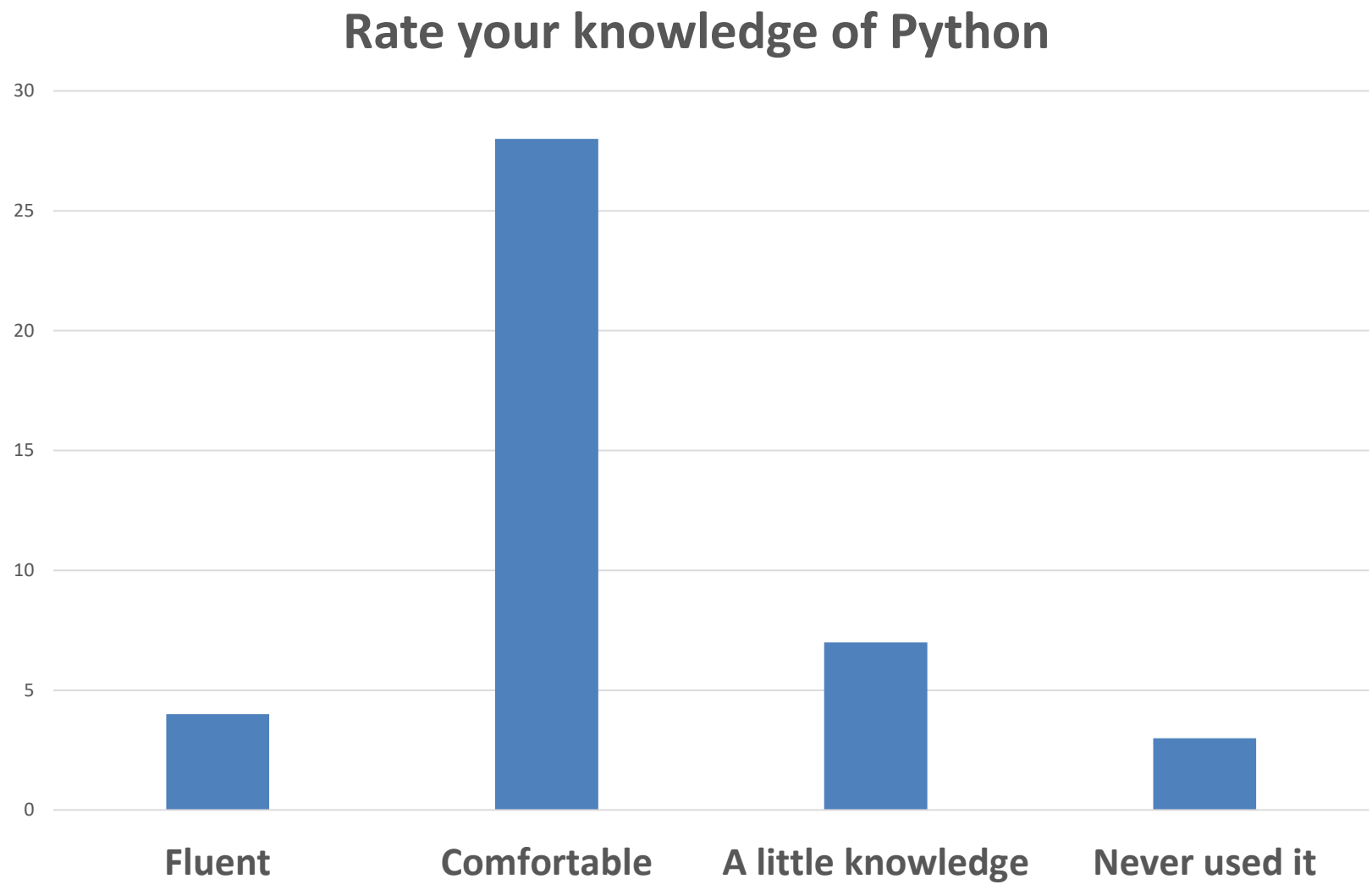
Since then…

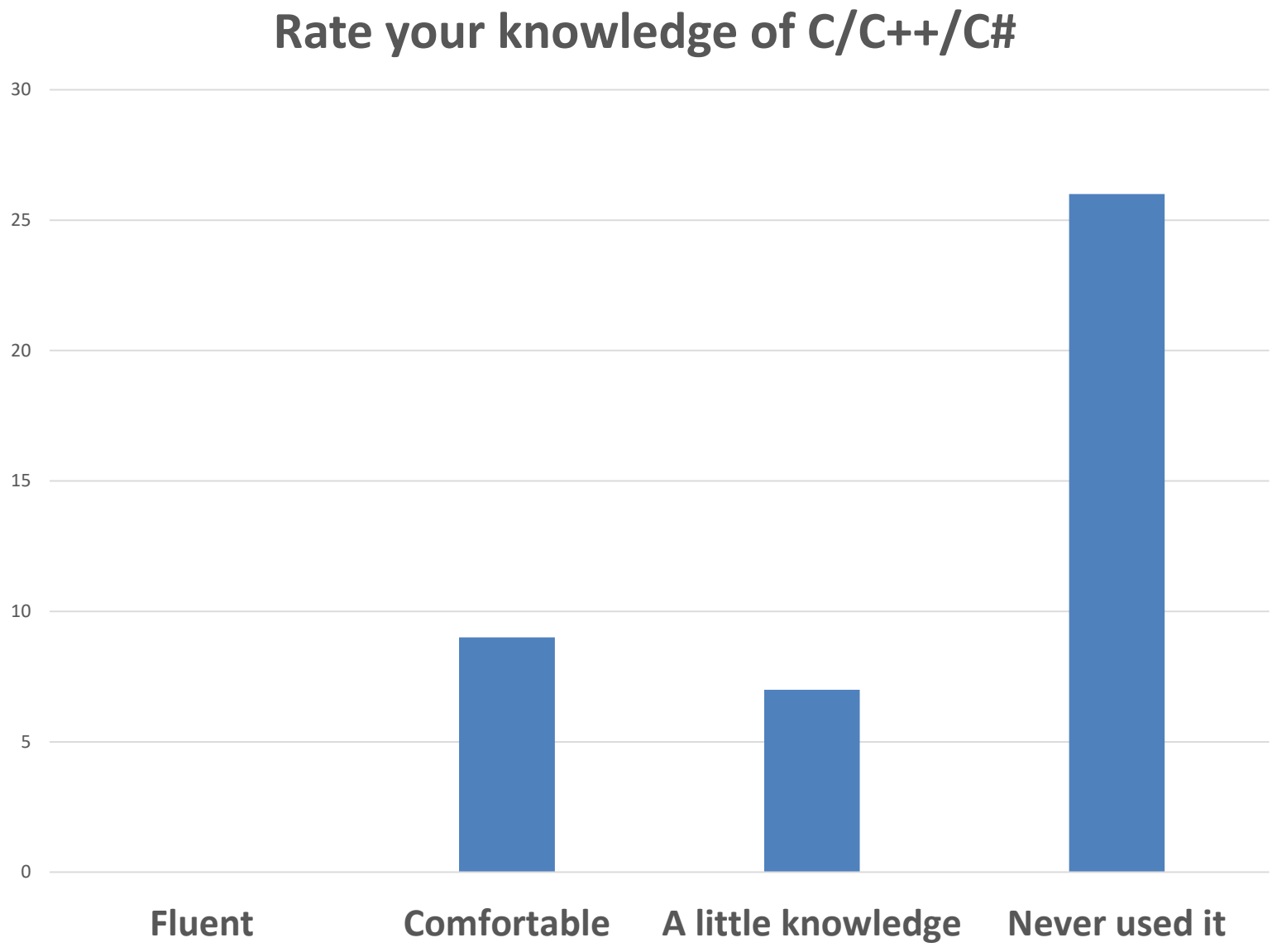
- There have been dramatic improvements in processor speed, types of memory, the scale of memory, peripheral devices, …
- But much of the core of a computer remains as it was 50 years ago.
- We will study the modern-age core of an Intel computer in Chapter 3 of Bryant and O'Hallaron. (even Macs use Intel-style CPUs)

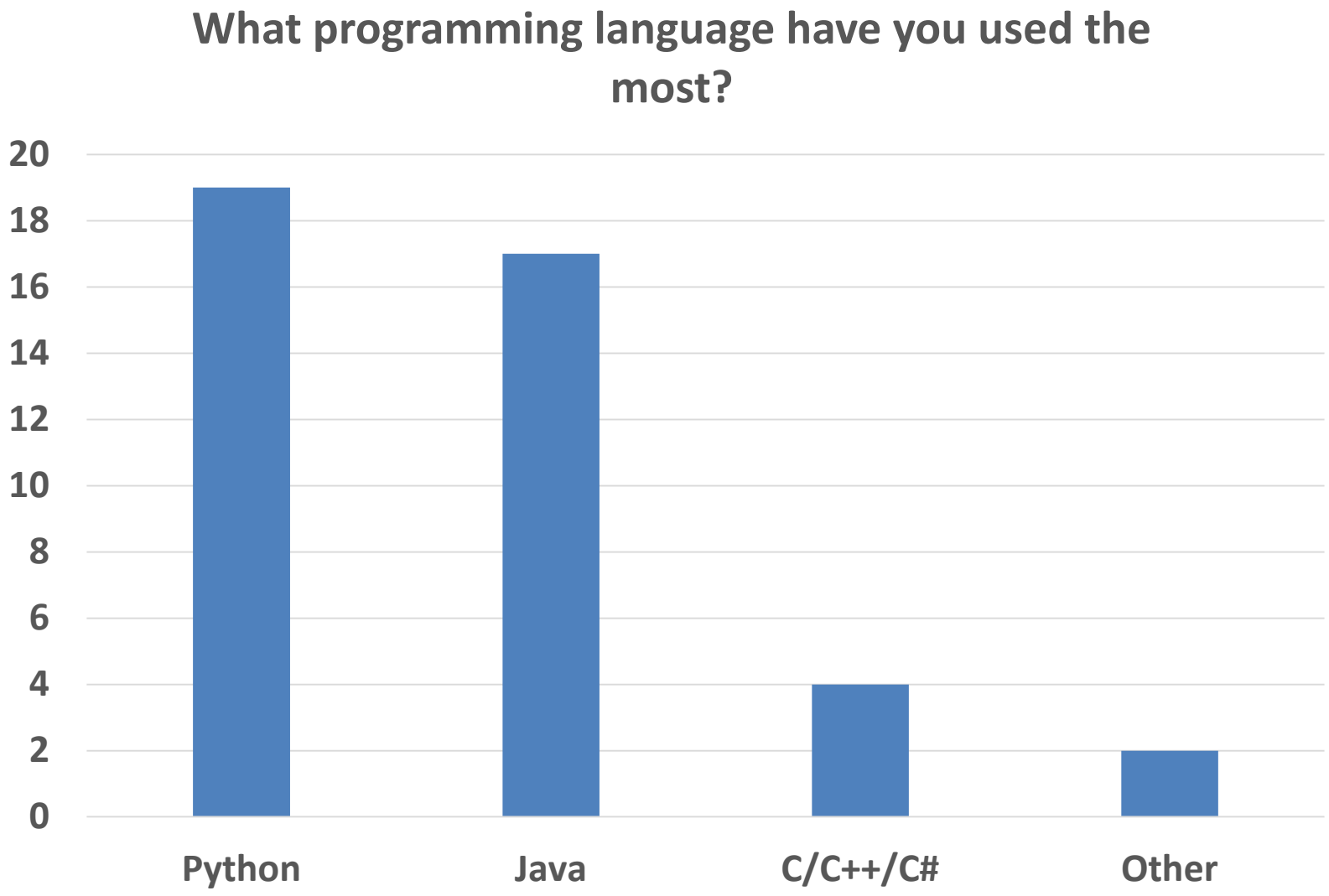
Survey Results

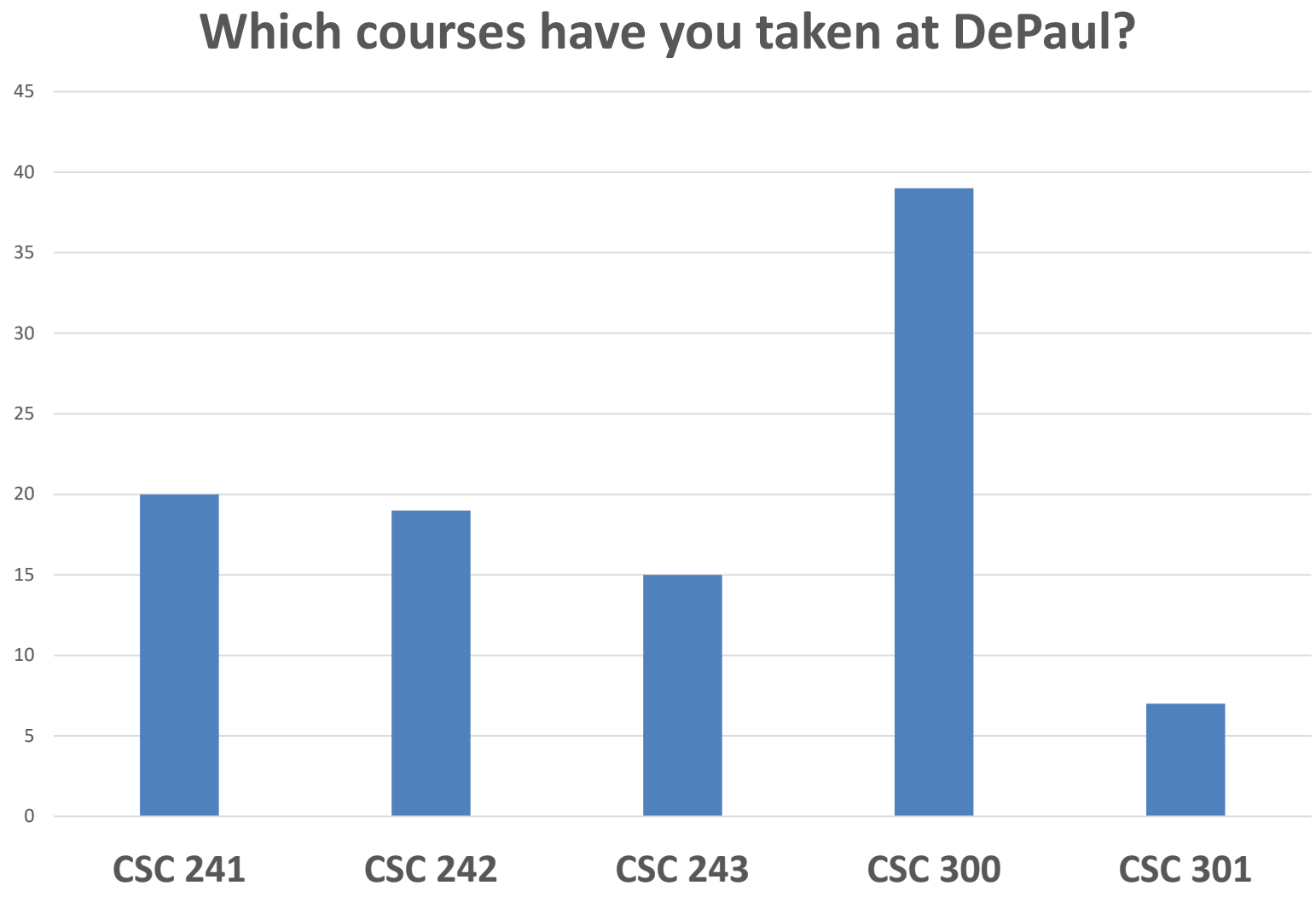


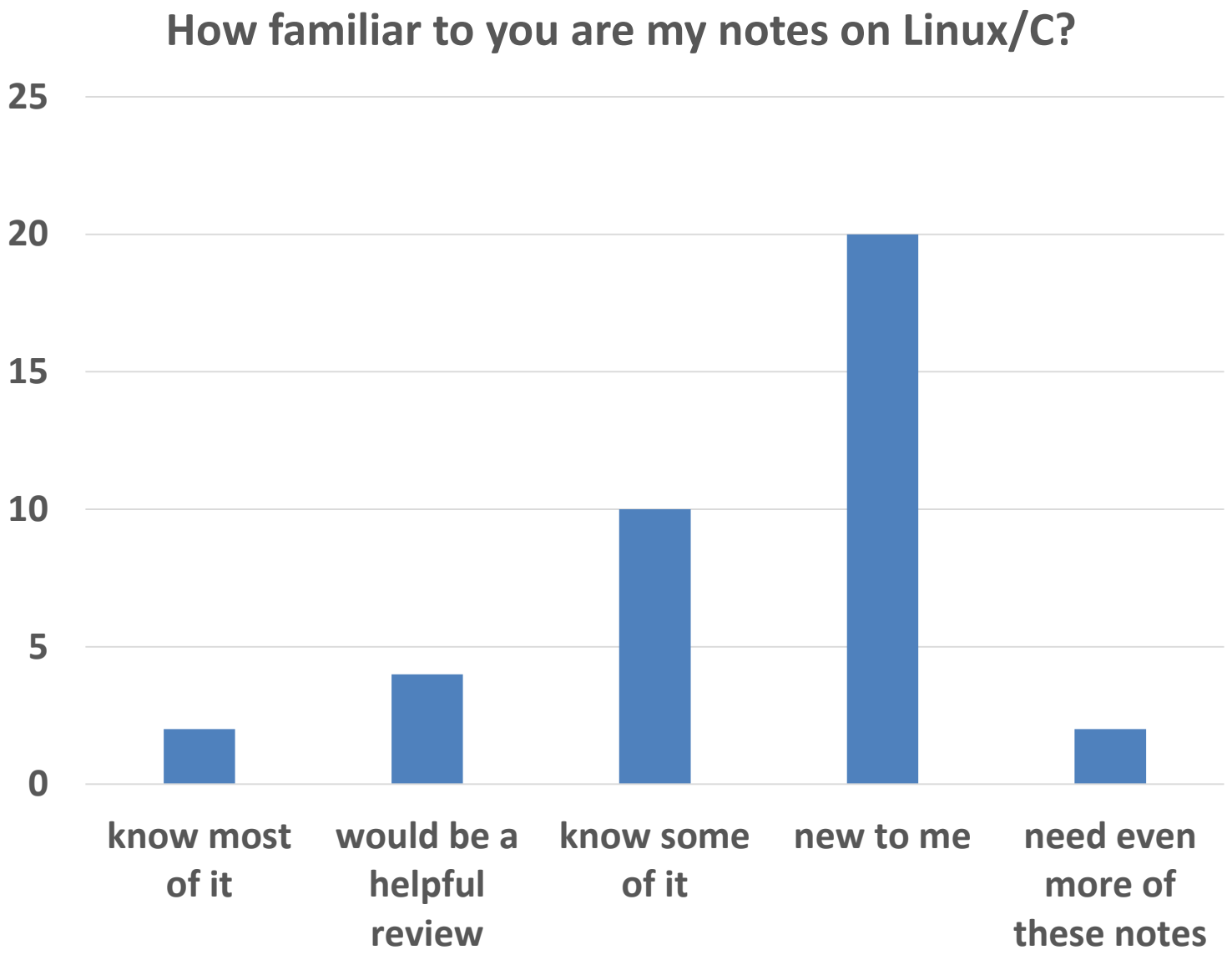

Rate your knowledge of Java
30
25
20
15
10
5
0
Fluent
Comfortable
A little knowledge
Never used it

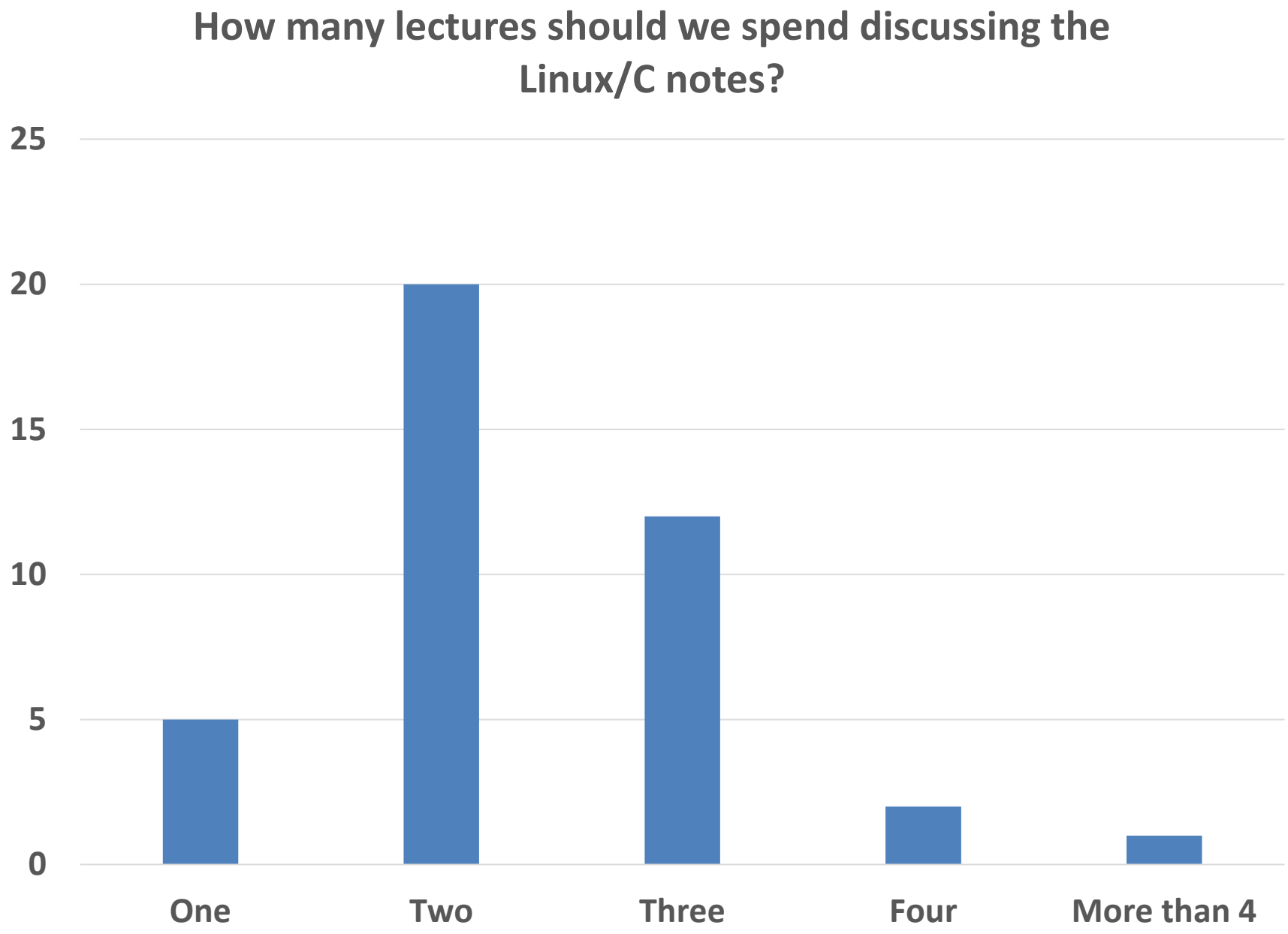

Rate your knowledge of Python
30
25
20
15
10
5
0
Fluent
Comfortable
A little knowledge
Never used it


Rate your knowledge of C/C++/C#
30
25
20
15
10
5
0
Fluent
Comfortable
A little knowledge
Never used it


What programming language have you used the most?
20
18
16
14
12
10
8
6
4
2
0
Python
Java
C/C++/C#
Other


Which courses have you taken at DePaul?
45
40
35
30
25
20
15
10
5
0
CSC 241
CSC 242
CSC 243
CSC 300
CSC 301


How familiar to you are my notes on Linux/C?
25
20
15
10
5
0
know most of it
would be a helpful review
know some of it
new to me
need even more of these notes


How many lectures should we spend discussing the Linux/C notes?
25
20
15
10
5
0
One
Two
Three
Four
More than 4

Introductory Topics

The lifetime of hello.c

To motivate the different concepts we will study in this course, we will trace the lifetime of the simplest C program, **hello.c**:

```
#include <stdio.h>
int main()
{
  printf("hello, world\n");
}
```

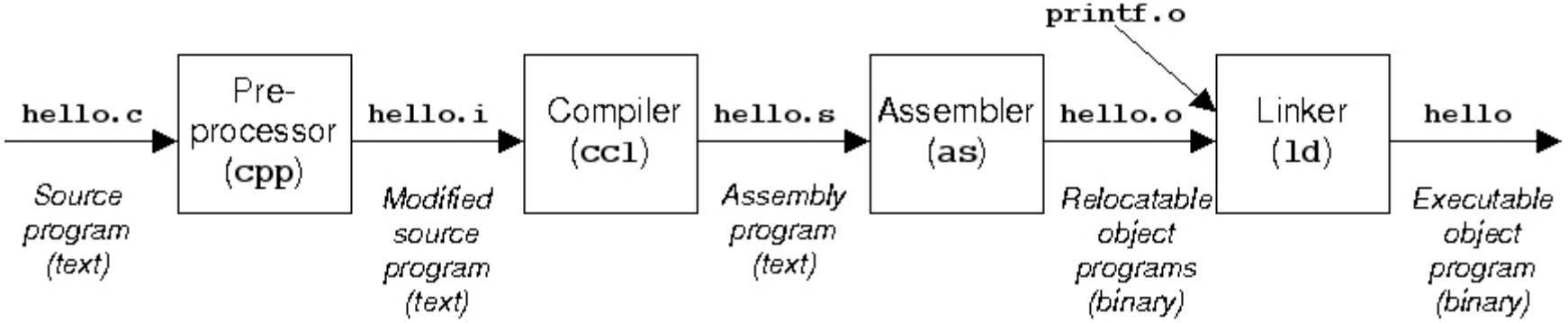
The lifetime of hello.c

- The program is initially created using an editor and then saved as a text file.
  - So far, the same format as any text file
  - Either way the file is just a sequence of 0s and 1s
  - Characters represented using ASCII standard
  - All information is stored as a bunch of bits: context is used to differentiate programs from data
  - We will study information storage (chapter 2) in week 2 or 3

From source to executable

```
$ gcc -o hello hello.c
```

- The compilation system performs the translation of the source file **hello.c** to an executable **hello** which consists of a sequence of low-level machine language



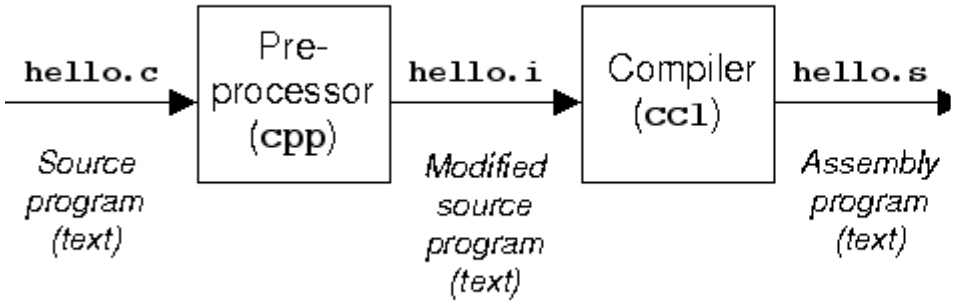

```
$gcc -o hello.i -E hello.c
```

- To obtain the assembly program hello.s:
  ```
  $gcc -o hello.s -S hello.c
  ```
- You can also create an object file, using
- ```
  $gcc –o hello.o –c hello.c
  ```
  ```

From source to executable

- It is a little more interesting to look at the intermediate files generated by **hello.c**:

```
$ gcc -o hello hello.s –S –O2
```



Why does a programmer need to know how compilation is done?

- To optimize program performance
  - Understand what parts of c programs are platform-dependent
  - Understand how compilers translate C into machine code (chapter 3; CSC 373)
  - Understand compiler optimization (CSC 373-374)
  - If time, understand memory caching and the impact of caching on the efficiency of programs (chapter 6)

- To avoid security holes (chapter 3; CSC 373)
  - We will study the buffer overflow bugs that are the cause of many security holes in networks and the Internet, and how to avoid them by being aware of the stack discipline that compilers use to generate code for functions (part of chapter 3)

Running the executable

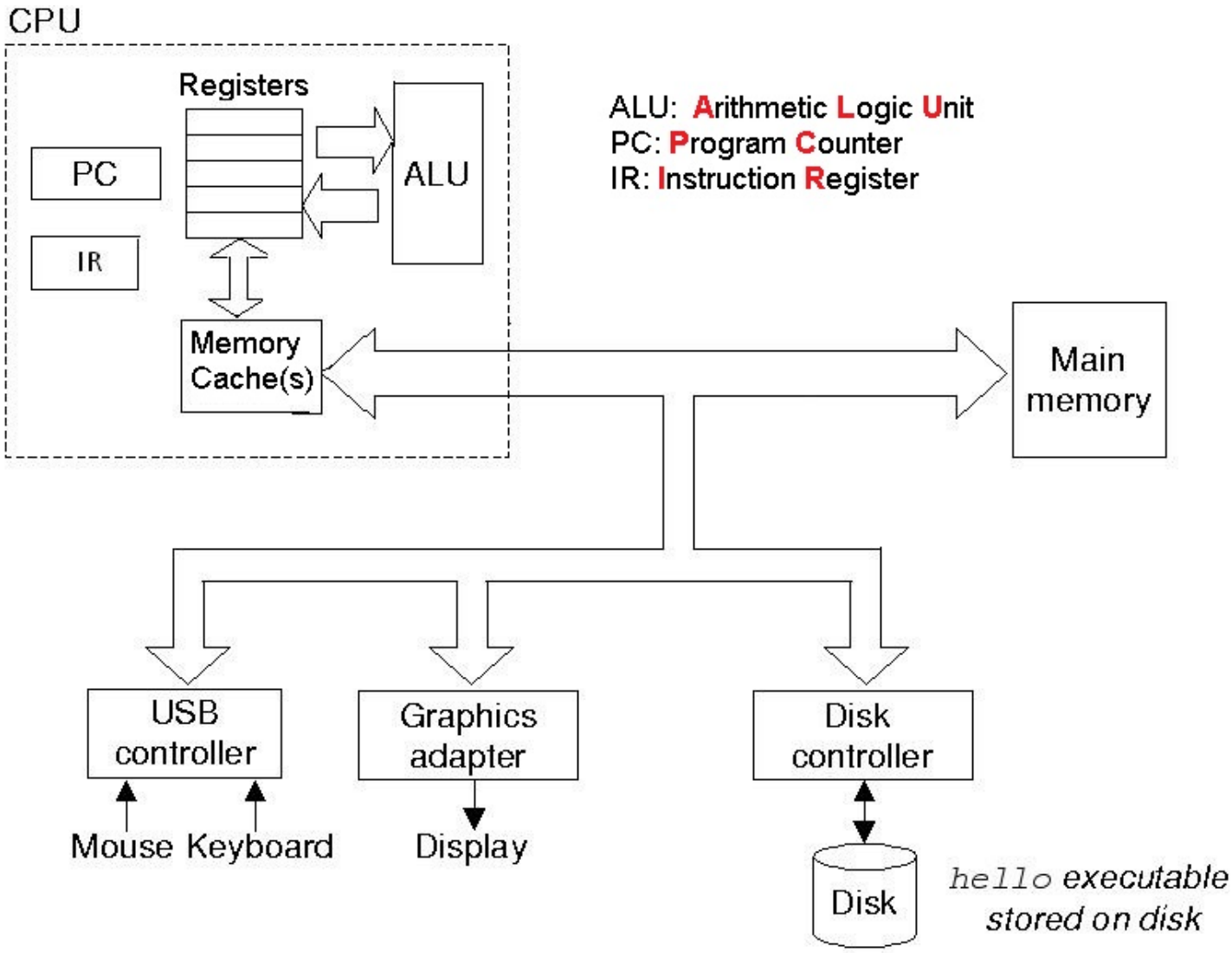
- We run the executable hello as follows

```
$ ./hello
hello, world
```

- The shell is the command line interpreter that prints the prompt (varies from machine to machine, but we'll assume the prompt is $), waits for you to type a command, and then performs the command.

  To understand what happens to **hello** during the execution, we need to understand a little about the hardware organization of a computer system.

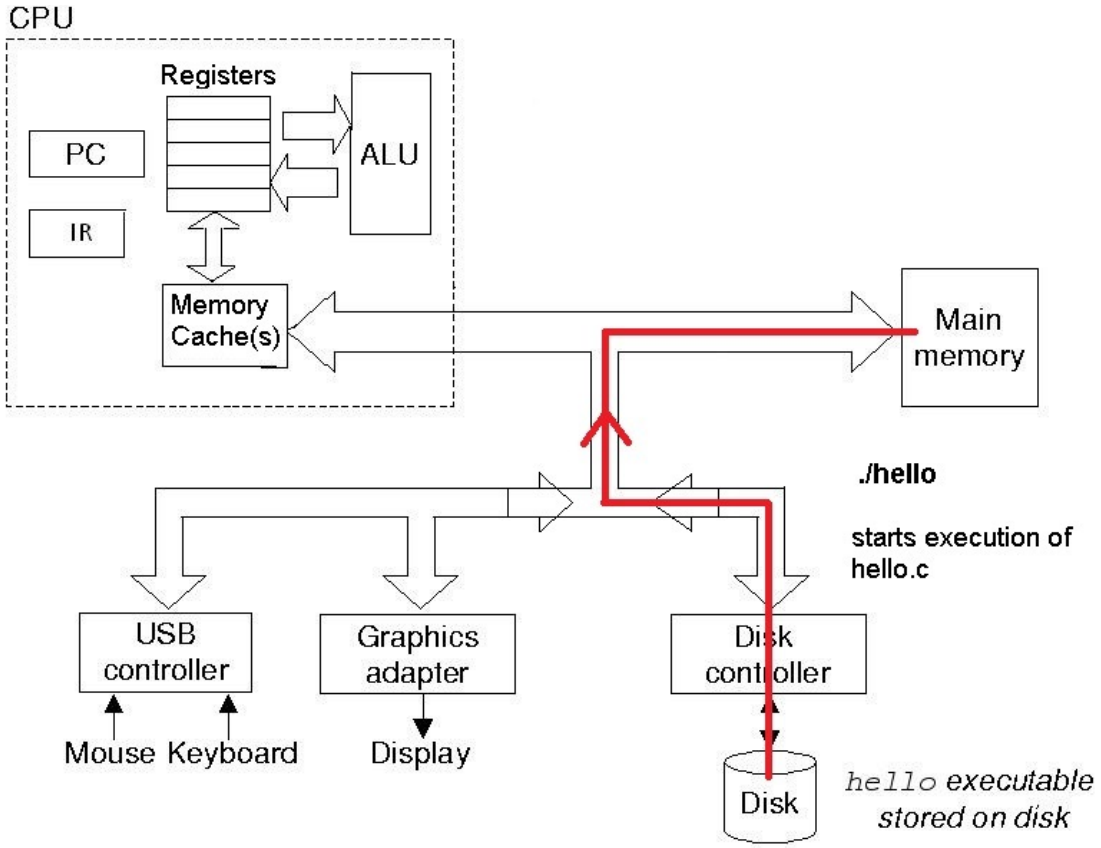
Hardware organization of a computer system



Beginning program execution

- All processes on a 64-bit machine have 2^{64} addressable locations in their **address space**
- Each location is 1 **byte** (or 8 bits).
- Therefore, a process thinks it has 2^{64} bytes of memory

  (= 18,446,744,073,709,551,616 bytes)
- Typical computers have **much less** actual RAM (more like 16 Gbytes = 16 billion bytes), or about 1 billionth as many bytes
- Also, typically many processes are running concurrently, and must share RAM
- Therefore, computer operating systems support **virtual memory** to give a process the illusion that it has 2^{64} bytes of memory
- Executable files are stored on disk in **elf** format; therefore the executable files are much smaller than 2^{64} bytes
- Upon start of execution, a portion of the address space is constructed for the process and loaded into RAM



- A typical minimum amount of data to copy to main memory is 4096 bytes (the **page** or **sector** size). One or more pages is copied from the disk to main memory, depending on the size of the program and the amount of RAM available.

- Since typically several processes are running concurrently, only a fraction of the total RAM on a computer is available to any individual process.

During program execution

- Portions of the pages in main memory built from `hello` are copied to the cache(s) as needed.

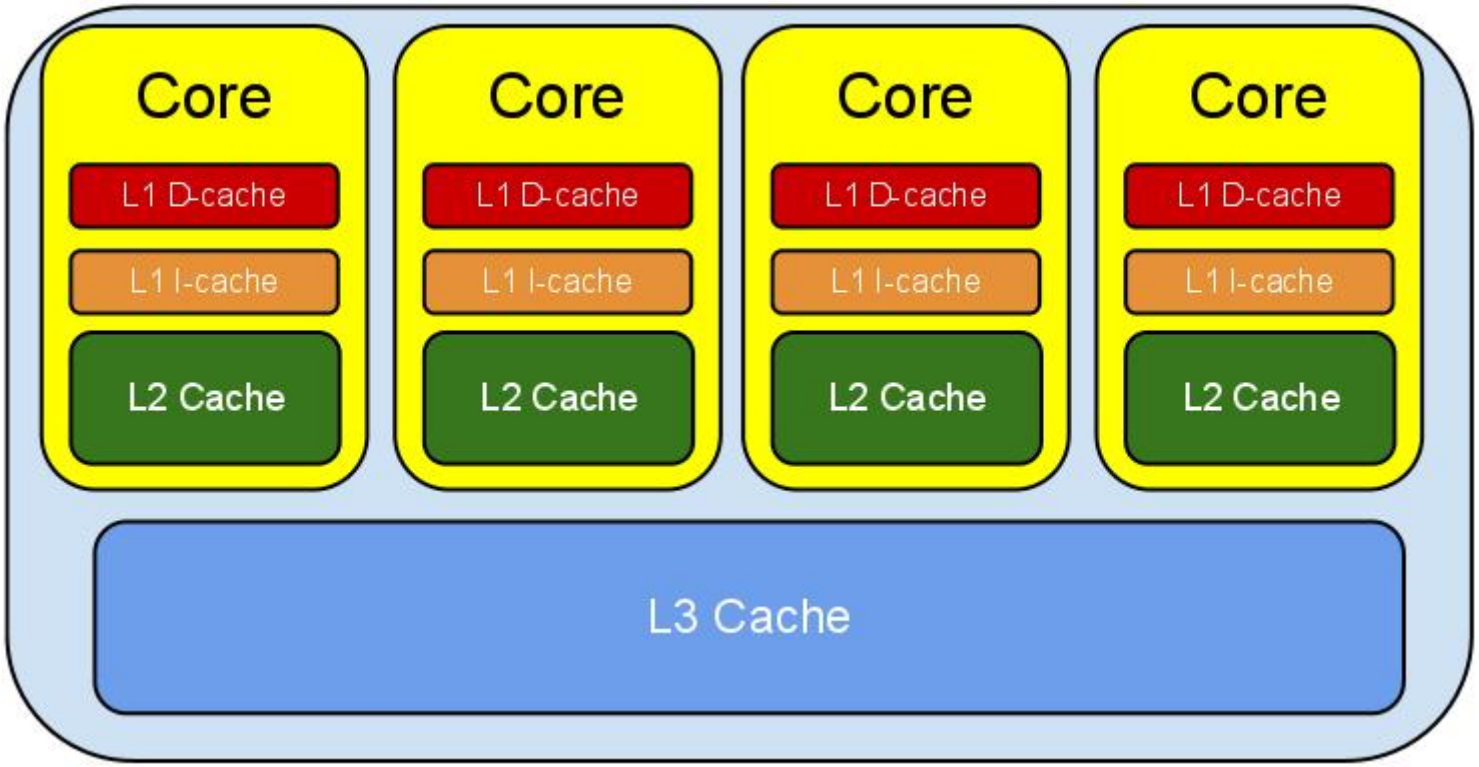
- These are called **blocks**, and are smaller than pages. Data is written back to main memory during this process as it is changed (or perhaps only as needed).

- From the cache, data (some are instructions) is copied between the cache and the registers.

Multiple kinds of caches

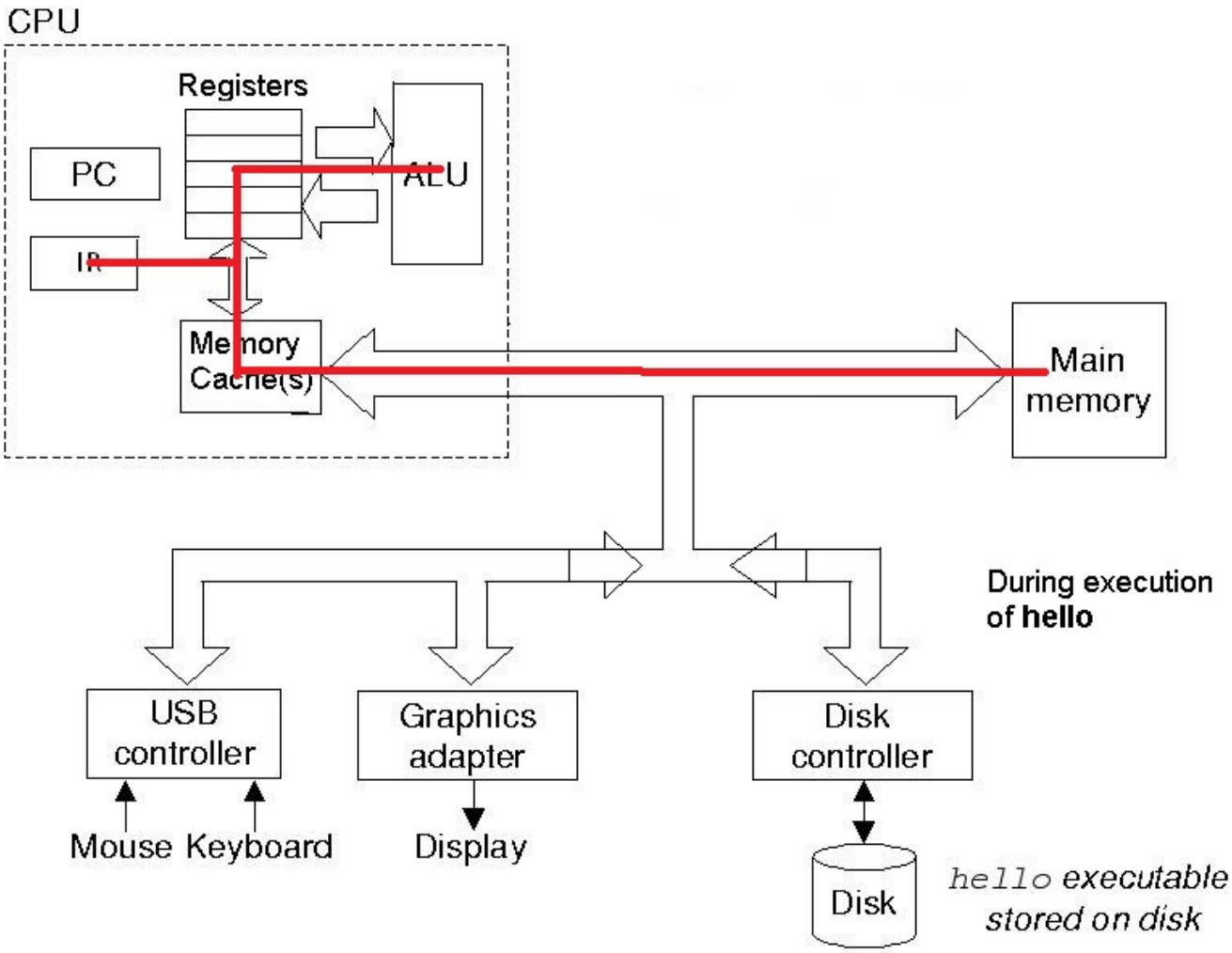
- The fastest kind of cache is called the **L1 cache.**
- Typically, computers now have more than one CPU (also called **core**). There are usually 2-8 cores.
- Each core has a devoted L1 cache
- There are commonly two other caches, called **L2** and **L3** caches. They are slower, and may be shared between cores.
- The cache may also be divided between D-cache (data) and I-cache (instructions). This is because instructions and data are found in different portions of the address space.

Multiple kinds of caches



(https://wiki.csiro.au/display/ASC/Understanding+the+CPU+Cache)

During program execution



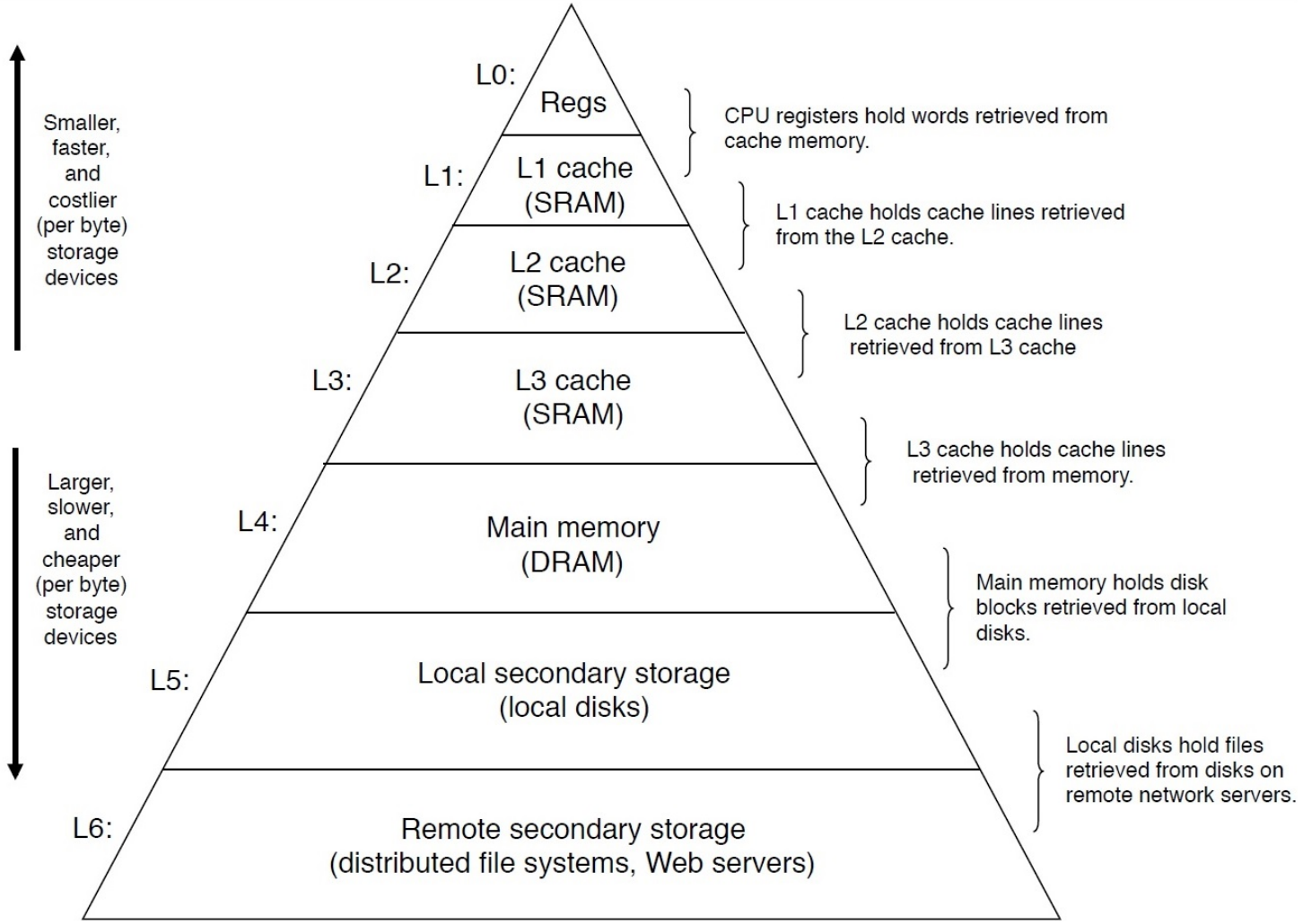
Cost vs. speed

- There are several different types of computer memory
- In general, the faster the memory, the more expensive it is
- If money were no object, we could build a computer with only the fastest kind
- But since it is, computers have a lot of cheaper memory, and a little of the expensive memory.
- The fastest kinds of memory are *registers* and *caches*

The memory hierarchy

- Operating systems spend a lot of time moving information from one place to another
  - When a program executes, first it's copied from disk to main memory, then portions are copied to the cache(s). Eventually a subset of a program's instructions are copied into the L1 I-cache, and a subset of the program's data are copied into the L1 D-cache
  - Instructions are then copied from the L1 cache to the IR; data may be copied to the registers, depending on the instructions.
  - Output copied from disk and/or main memory to the output device
- Goal: minimize information transfer time while at the same time minimizing cost
- Answer: intermediate, smaller, faster storage (caches)

The memory hierarchy



Typical specs

- For about $1000-$1200 (not including peripherals), you get a computer with
  - 2-3 GHz CPU: 2-3 billion cycles per second. This is a measure of how many instructions can be executed by the CPU per second (although not a direct measure)
  - Quad-core (4 CPUs), so 4 processes or threads can be running at the same time
  - 8-12 MB "shared" cache (8-12 million bytes). Specs do not detail how much of each type of cache. Some is shared among the cores.
  - 16 GB RAM (16 billion bytes; main memory)
  - 1 TByte Disk (1 trillion bytes)
    - Solid state disks are faster and therefore more expensive / smaller

Comparisons of speed

Memory Type	Latency (how many times slower)
Registers	1
L1 Cache	4
L2 Cache	10
L3 Cache	50
RAM	200
Disk	100,000