CS 1002 Programming Fundamentals
Lecture 03 29-August 2022

Muhammad Uzair Khan
Assosiate Professor & Director ORIC
Uzair.khan@nu.edu.pk

National University
Of Computer and Emerging Sciences

Computers and Programming

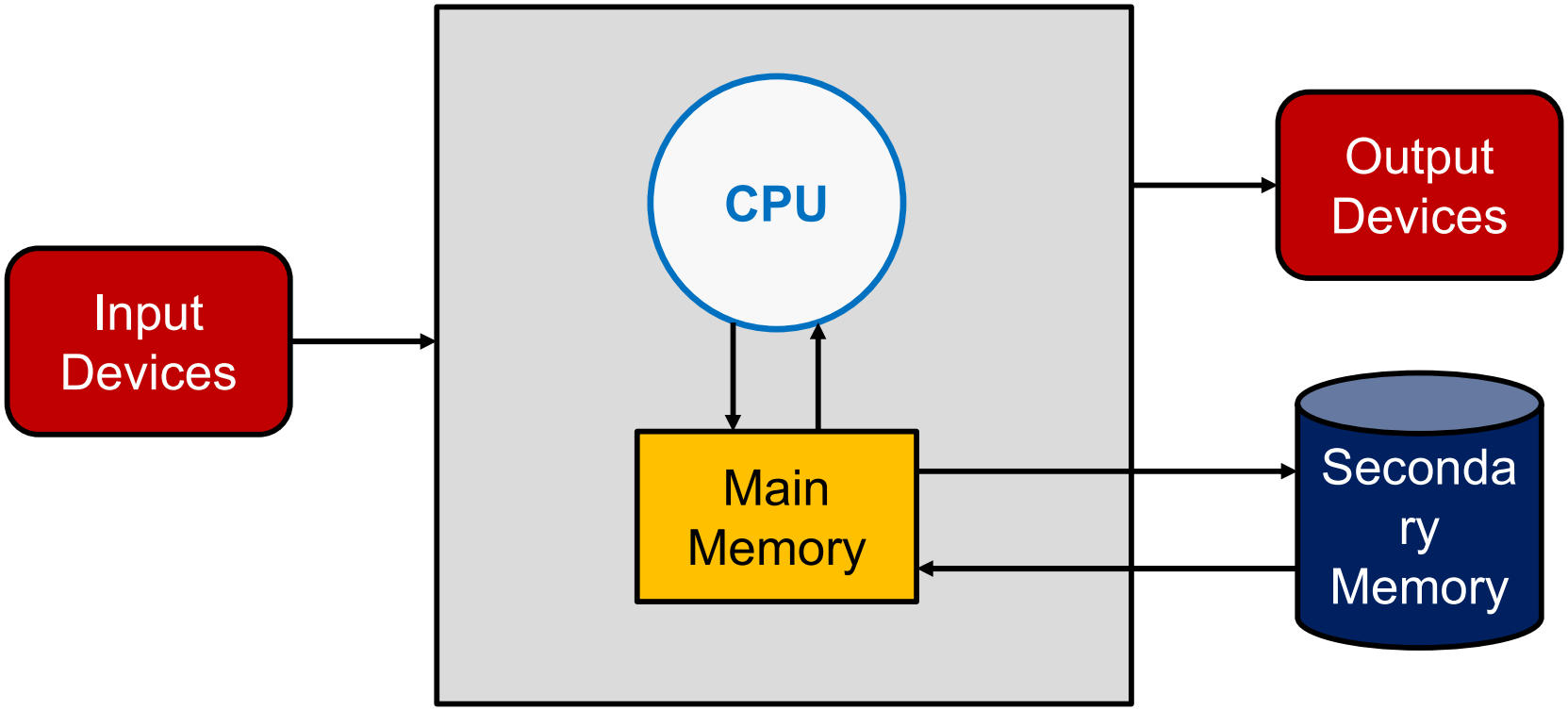
- A computer is just a machine (the *hardware*) for executing programs (the *software*)
  - Hence, the software rules the hardware!
- The process of creating software is called *programming*, and it is the focus of this course
- Virtually, anyone can learn how to program computers
  - It requires only some grit!

Hardware Basics

- To be a successful programmer, you need to know some details of how computers work

- For instance, understanding the basics of hardware will help you analyze the *performance* (or *efficiency*) of any of your programs
  - Will the data of your program fit in *memory*?
    - If not, how would that impact the performance of your program?
  - Is your program *CPU-bound* or *IO-Bound*?
    - If CPU-bound, how powerful is your CPU?
    - If IO-bound, how big is your disk or network bandwidth?

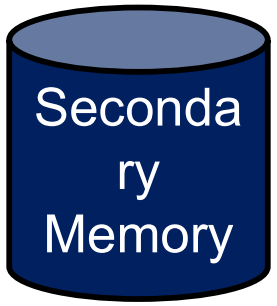
Functional View of a Computer



Functional View of a Computer

- The secondary memory is where your *saved program and data* reside
- It is a *non-volatile* storage
  - I.e., when the power is turned off, your program and data will NOT be lost

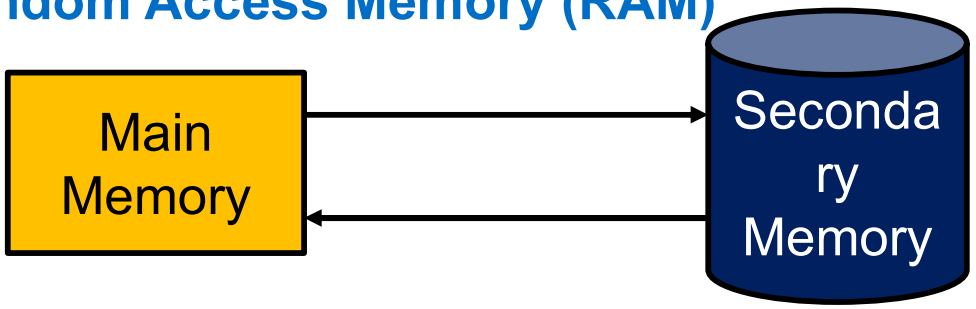
E.g., Hard Disk



Functional View of a Computer

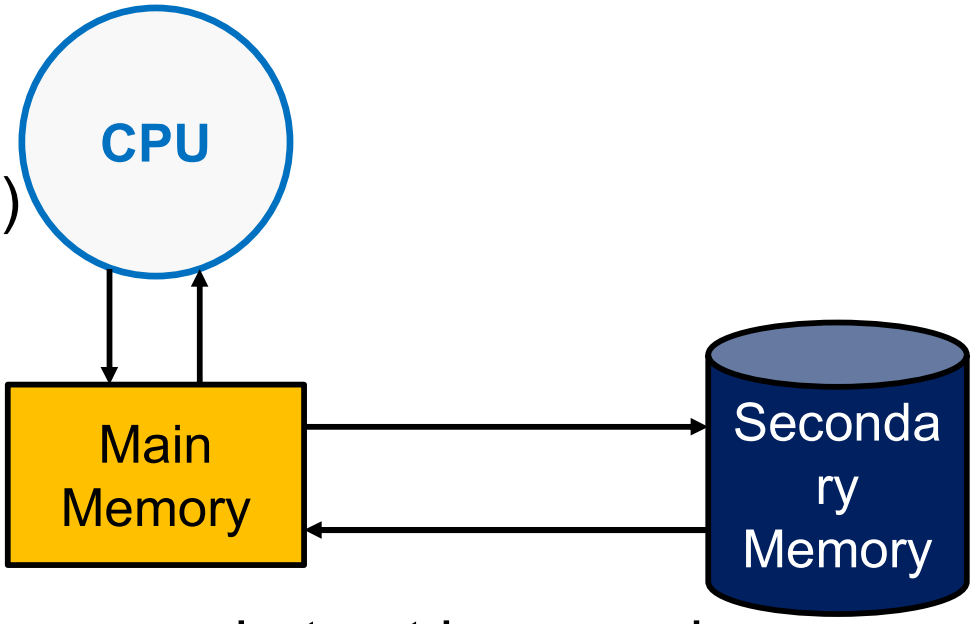
- The main memory is much faster (but more expensive) than the secondary one, however, it is *volatile*
- Your program and data are *copied* from secondary memory to main memory for efficiency reasons

E.g., Random Access Memory (RAM)



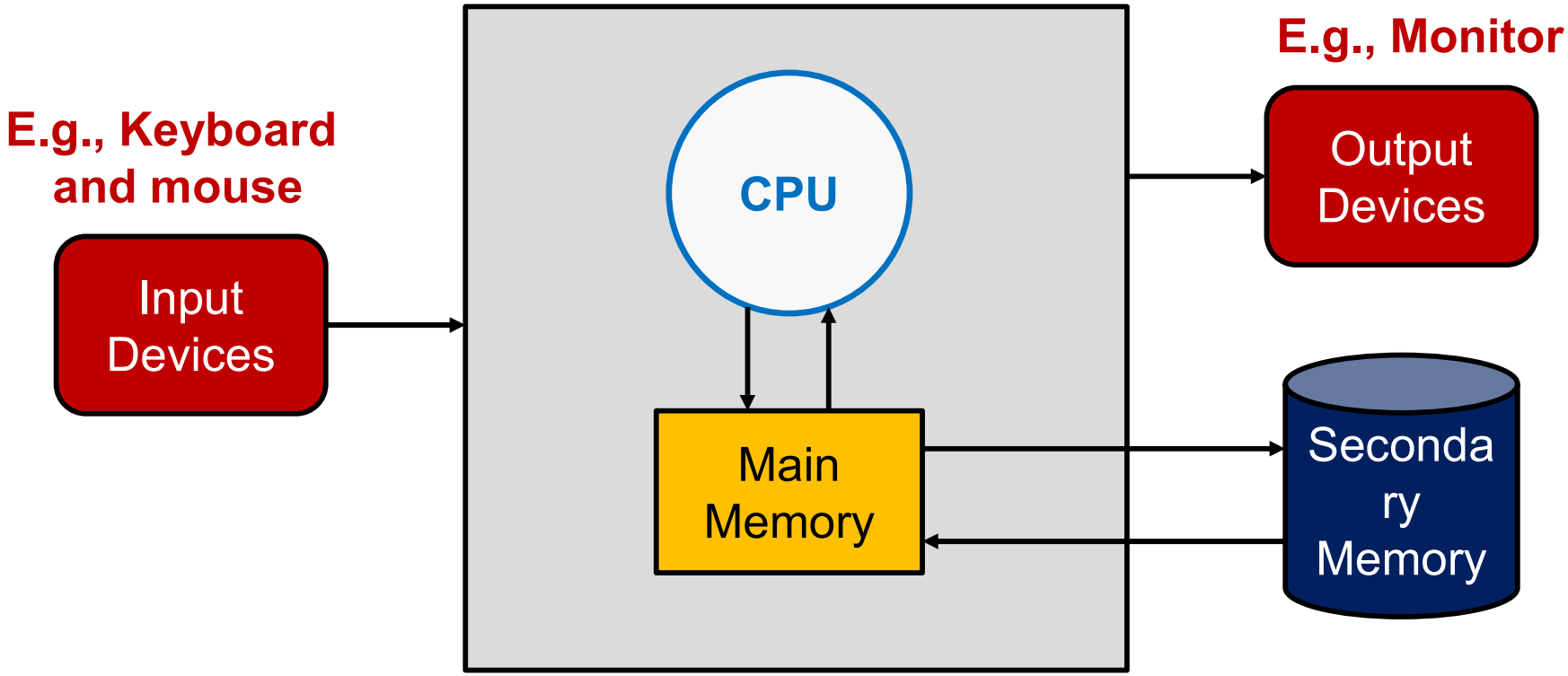
Functional View of a Computer

- The *Central Processing Unit* (CPU) is the “brain” of the computer
- It can at least perform:
  - Arithmetic operations (e.g., adding 2 numbers)
  - Logical operations (e.g., test if 2 numbers are equal)



- It can *directly* access information stored in main memory but not in secondary memory

Functional View of a Computer



Functional View of a Computer

- Humans interact with computers via Input and Output (IO) devices
- Information from Input devices are processed by the CPU and may be shuffled off to the main or secondary memory
- When information need to be displayed, the CPU sends them to one or more Output devices

E.g., Keyboard and mouse

Input Devices

E.g., Monitor

Output Devices

Programming Languages

- A *program* is just a sequence of instructions telling the computer what to do

- Obviously, we need to provide these instructions in a language that computers can understand
  - We refer to this kind of a language as a *programming language*
  - Python, Java, C and C++ are examples of programming languages

- Every structure in a programming language has an exact form (i.e., *syntax*) and a precise meaning (i.e., *semantic*)

Machine Languages

- Python, Java, C, and C++ are, indeed, examples of *high-level* languages

- Strictly speaking, computer hardware can only understand a very *low-level* language known as *machine language*

- If you want a computer to add two numbers, the instructions that the CPU will carry out might be something like this:

```
Load the number from memory location 2001 into the CPU
Load the number from memory location 2002 into the CPU
Add the two numbers in the CPU
Store the result into location 2003
```

A Lot of Work!

High-Level to Low-Level Languages

- In a high-level language like C++, the addition of two numbers can be expressed more naturally:

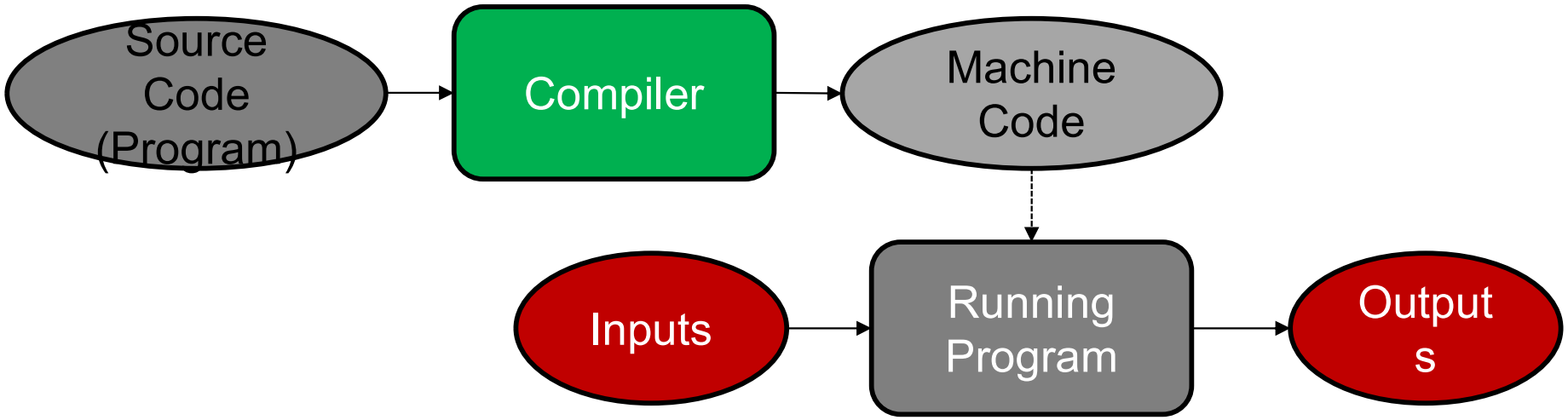
```
c = a + b
```

Much Easier!

- But, we need a way to translate the high-level language into a machine language that a computer can execute
  - To this end, high-level language can either be *compiled* or *interpreted*

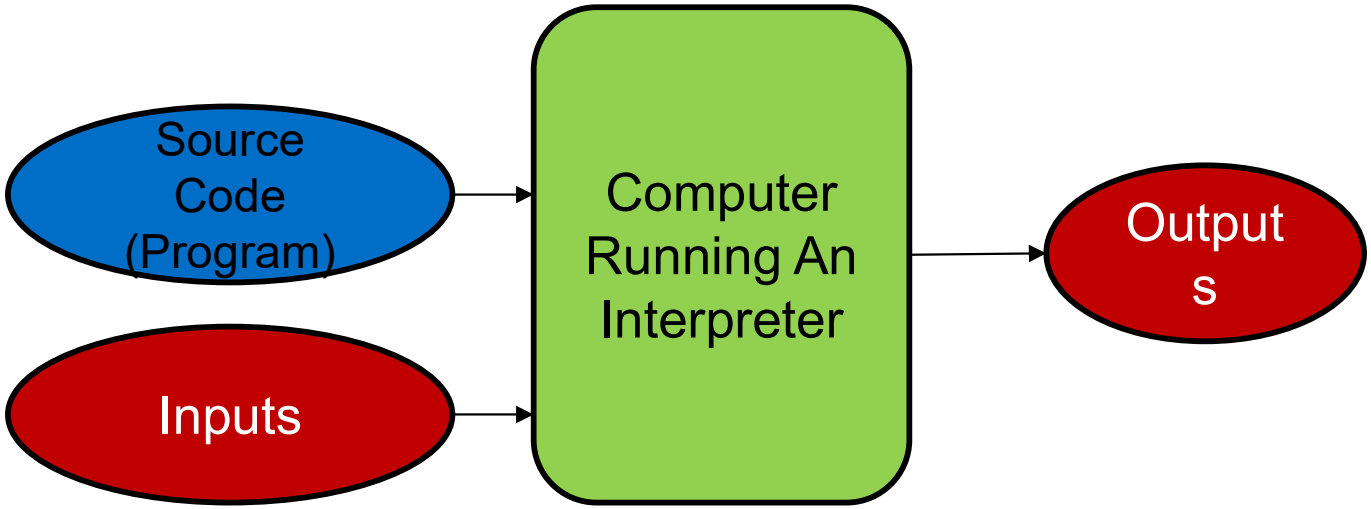
Compiling a High-Level Language

- A *compiler* is a complex software that takes a program written in a high-level language and *translates* it into an equivalent program in the machine language of some computer



Interpreting a High-Level Language

- An *interpreter* is a software that analyzes and executes the source code instruction-by-instruction (*on-the-fly*) as necessary



- E.g., Python is an interpreted language
- C++ is a compiled language
- Java is hybrid !!!

Compiling vs. Interpreting

- Compiling is a static (i.e., pre-execution), one-shot translation
  - Once a program is compiled, it may be run over and over again without further need for the compiler or the source code
- Interpreting is dynamic (i.e., happens during execution)
  - The interpreter and the source code are needed every time the program runs
- Compiled programs tend to be faster, while interpreted ones lend themselves to a more flexible programming environments (*they can be developed and run interactively*)