Information
Technology
Institute

SWE Tracks’ renovation

In view of SWEBOK V3.0

Information Technology Institute

Introduction to Operating Systems

Lecture 2

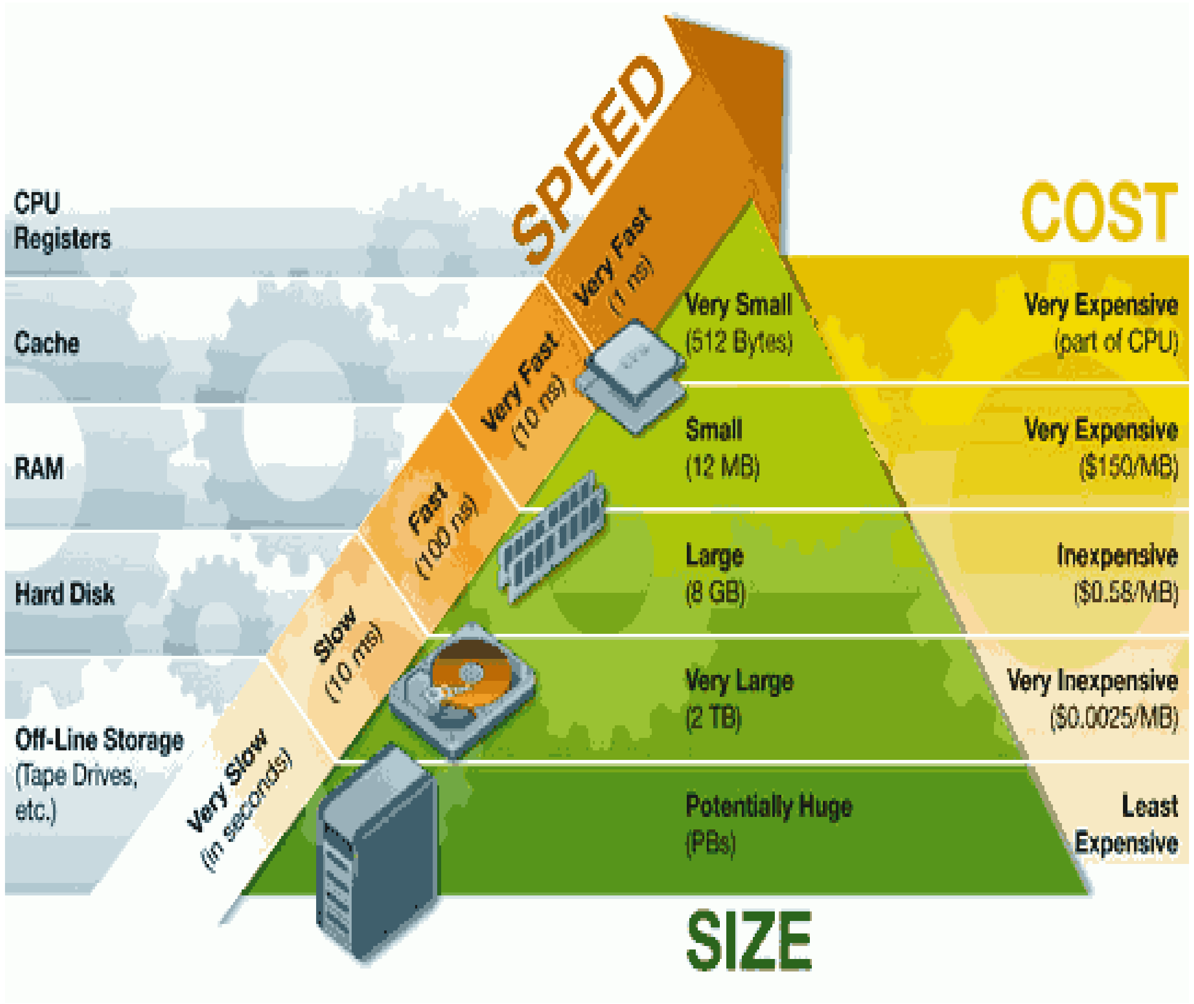
Memory Management

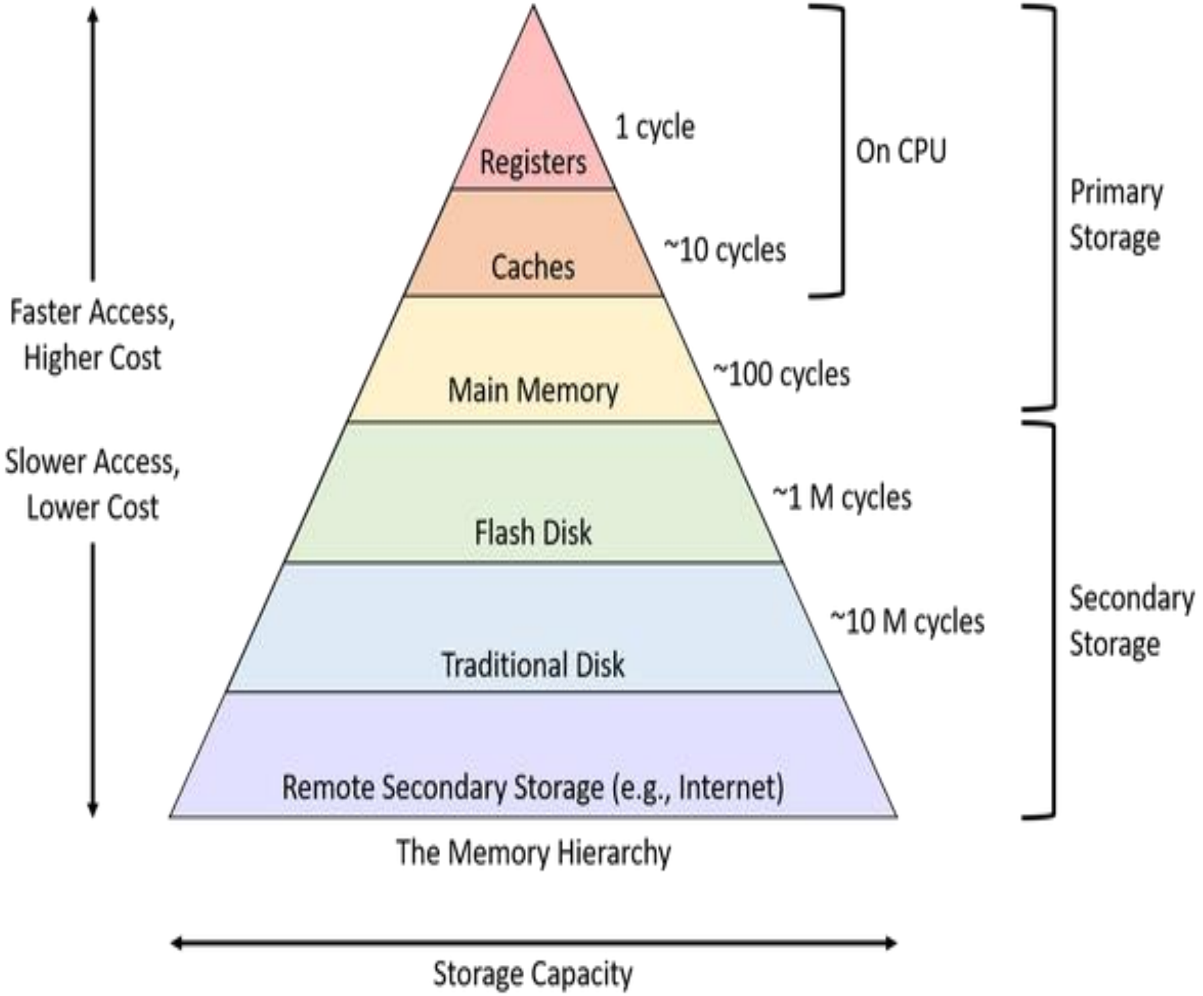
Content

- Need of Memory Management
- Address Binding
  - Logical Address Vs. Physical Address
- Inter-process Communication
  - Shared Memory Method
  - Message Passing Method

Course Outlines

Storage-Device (memory) Hierarchy

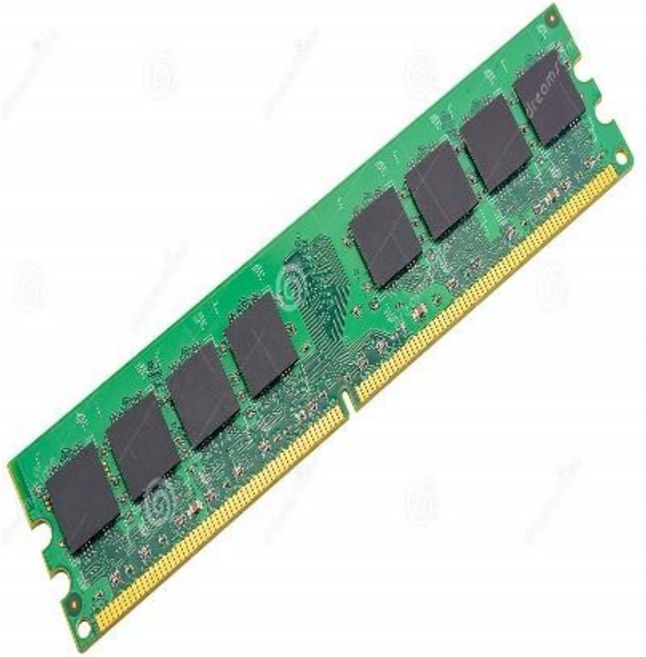




The Memory Hierarchy

RAM is the main memory in a computer.

- Array of memory words
- Each byte has an address
- Memory Address:
  - Physical
  - Logical



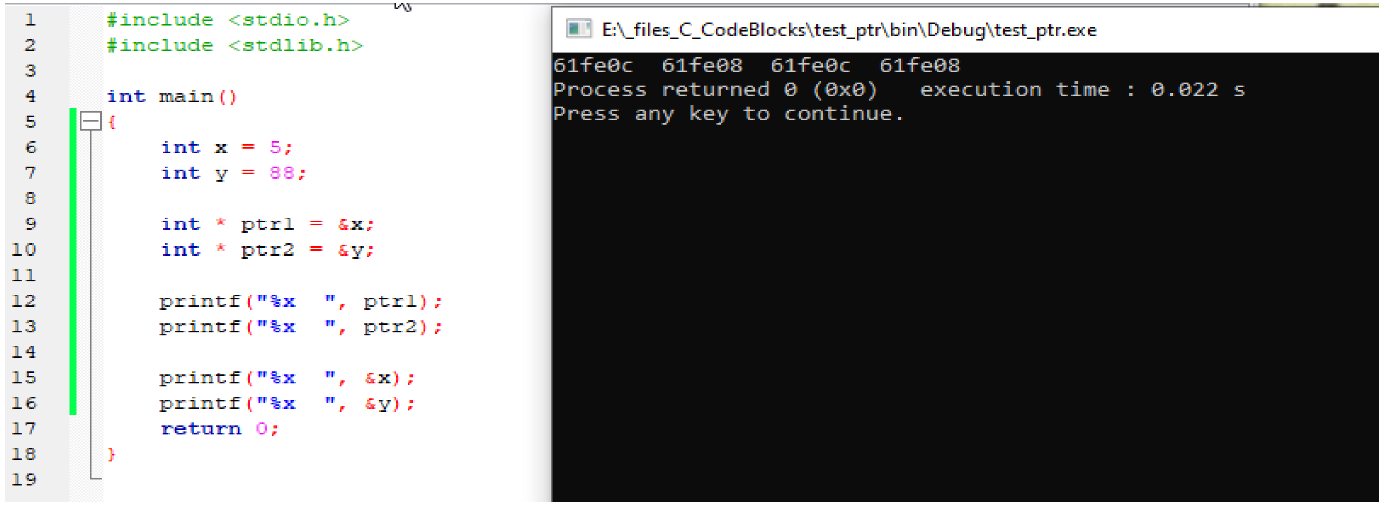
3.1 Need of Memory Management

- In systems with multiple programs running in parallel, there could be many processes in memory at the same time, and each process may have specific memory needs.
- Processes may need memory for various reasons:
  - **First**, the executable itself may need to be loaded into memory for execution. This is usually the *instructions or the code* that needs to be run.
  - The **second** item would be the data part of the executable. These could be *hardcoded* strings, text, and variables that are referenced by the process.
  - The **third** type of memory requirement could arise from runtime requests for memory. These could be needed from the *stack/heap* for the program to perform its execution.
- The OS and the kernel components may also need to be loaded in memory. Additionally, there may be a specific portion of memory needed for specific devices (Ex: printer spooling).

3.2 Address Binding

- The program each time executed it is loaded in a different memory location (according to the available spaces at time loading).
- And so, when the process in waiting state for I/O operation it may be swapped out from main memory to virtual memory (part from secondary storage), and when the waiting state changed to ready, the process must swapped in to main memory which – almost cases- another location in the main memory.
- That means the addresses of the variables which used in the program, may be changed many times at the runtime!!!
- To solve this problem, the common solution is to *map* the program's *compiled* addresses to the actual address in *physical* memory.

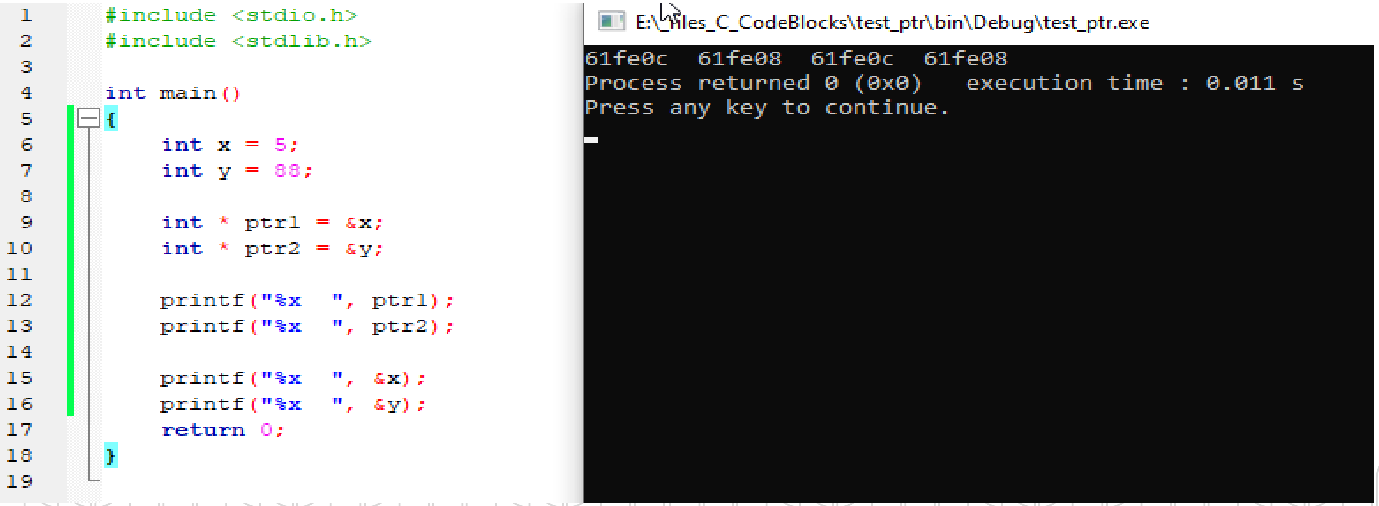
3.2.1 Logical Address Vs. Physical Address



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x = 5;
    int y = 88;

    int * ptr1 = &x;
    int * ptr2 = &y;

   printf("%x  ", ptr1);
   printf("%x  ", ptr2);

   printf("%x  ", &x);
   printf("%x  ", &y);
    return 0;
}
```

```
61fe0c  61fe08  61fe0c  61fe08
Process returned 0 (0x0)   execution time : 0.022 s
Press any key to continue.
```

If we run this program many times, the addresses of x and y variable will be the same !!!



```c
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int x = 5;
    int y = 88;

    int * ptr1 = &x;
    int * ptr2 = &y;

   printf("%x  ", ptr1);
   printf("%x  ", ptr2);

   printf("%x  ", &x);
   printf("%x  ", &y);
    return 0;
}
```

```
61fe0c  61fe08  61fe0c  61fe08
Process returned 0 (0x0)   execution time : 0.011 s
Press any key to continue.
_
```

Because these addresses are logical addresses not physical addresses on the main memory

3.2.1 Logical Address Vs. Physical Address

- A program will have variables, instructions, and references that are included as part of the source code. The references to these are usually referred to as the symbolic addresses. When the same program gets compiled, the compiler translates these addresses into relative addresses (Logical Address).
- This is important for the OS to then load the program in memory with a given base address and then use the relative address from that base to refer to different parts of the program.
- In general, there is not enough physical memory to host all programs at the same time. This leads to the concept of virtual memory that can be mapped to physical memory.
- The ***memory management unit*** is responsible for translating virtual addresses or logical addresses to physical addresses.

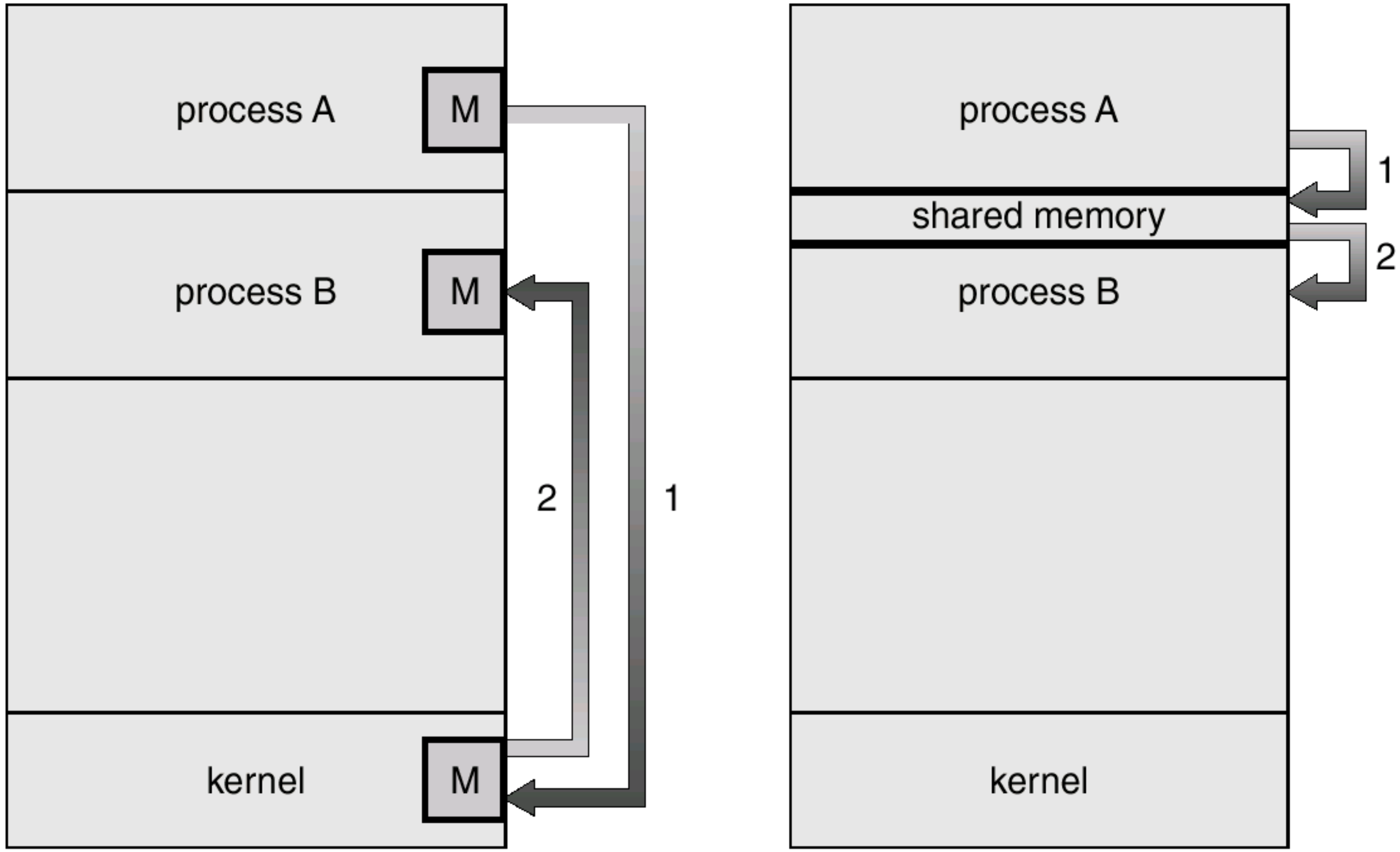
3.3 Inter-process Communication

- It is often desirable to have processes communicate with each other to coordinate work, for instance. In such cases, the OS provides one or more mechanisms to enable such process-to-process communication.
- These mechanisms are broadly classified as inter-process communication (IPC). The two common ways are explained in the following, which involve:
  - Shared memory and
  - Message passing.

Inter-process Communication

- The communication between these processes can be seen as a method of co-operation between them. Processes can communicate with each other through both:

- Shared Memory
- Message passing



3.3.1 Shared Memory Method

- When two or more processes need to communicate with each other, they may create a shared memory area that is accessible by both processes.
- Then, one of the processes may act as the *producer of data*, while the other could act as the *consumer of data*.
- The ***memory acts as the communication buffer*** between these two processes.
- This is a very common mechanism to communicate between processes.
- Note: this method need a way of management when the two processes need to save in the shared memory at the same time, it is called ***Synchronization***

3.3.2 Message Passing Method

- The other method is called message passing where the two processes have a predefined communication link that could be a ***file system***, ***socket***, ***named pipe***, and so on and a protocol-based messaging mechanism that they use to communicate.
- Typically, the first step would be to establish the communication channel itself.
- For example, in the case of a ***TCP/IP communication***, one of the processes could act as the ***server*** waiting on a specific port. The other process could register as a ***client*** and connect to that port. The next step could involve sharing of messages between the client and server using predefined protocols leveraging Send and Receive commands. The processes must agree on the communication parameters and flow for this to be successful.
- ***Note: Some OSs have system calls (APIs) for sending and receiving the data among processes.***

I/O Management

Content

- Need of I/O Management
- I/O Subsystem
- I/O Devices Categories:
  - Block Devices
  - Character Devices
- Synchronization and Critical Sections
  - Mutex
  - Semaphore
- Deadlocks

Course Outlines

1.2.5 Input Devices

- Which are peripherals used to provide data and control signals to a computer.
- Input devices allow us to enter raw data for processing. For Example:
  - Keyboard (default input device)
  - Microphone
  - Scanner (2D and 3D)
  - Mouse
  - Trackball
  - Touchpad
  - Barcode and QR Code readers
  - Digital Camera

1.2.6 Output Devices

- Which are pieces of computer hardware used to communicate the results of data processing performed by a computer.
- The objective of output devices is to turn computer information into a human friendly/readable form. For Example:
  - Screen (Monitor or Console) (default output device) – LED or LCD
  - Data Projectors
  - Speaker and Headphones
  - Printer (2D and 3D) – inkjet or laser
  - Plotter (wide format printer)
  - Cutter (2D or 3D)

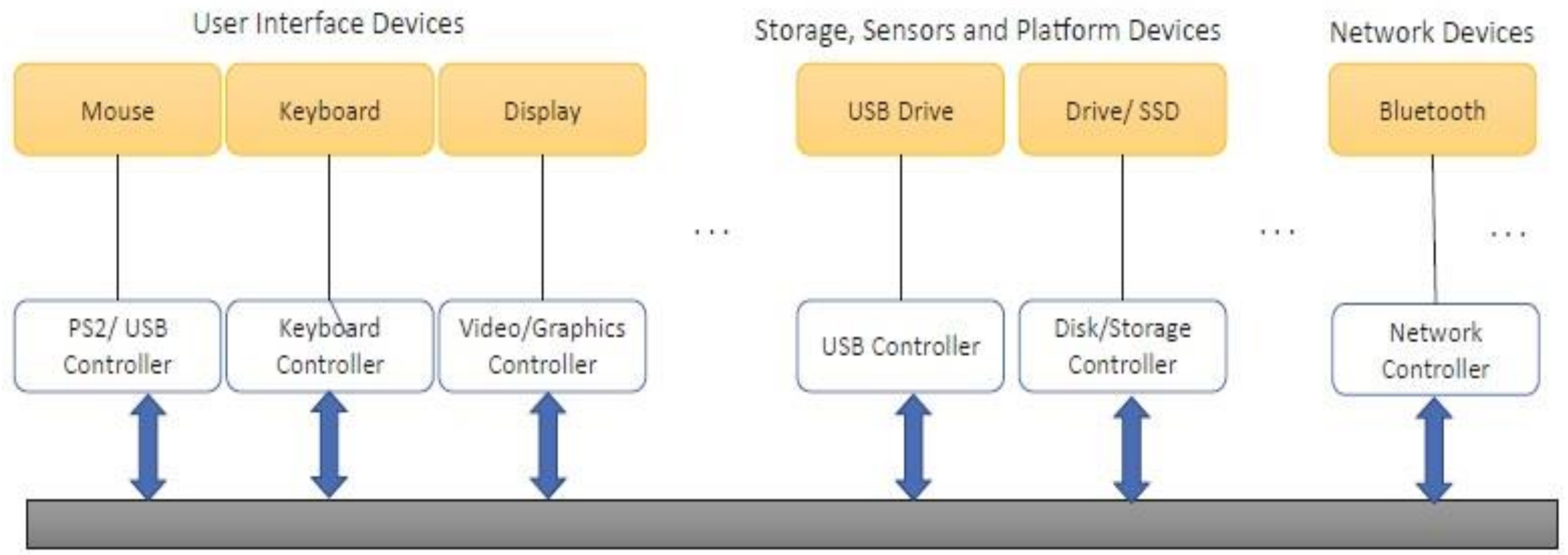
1.2.7 Input/Output Devices

- Some devices work as Input and Output devices like:
  - Touch Screen
  - Network
  - Most of I/O Ports (Both of serial and parallel)
  - New types of ports, like USB and Type-C ports
- All of the secondary storages used as I/O devices for permeant storage, like:
  - Hard disk
  - Floppy disk
  - Flash memory
  - CD and DVD

4.1 Need for I/O Management

- As part of the system, there could be multiple devices that are connected and perform different input-output functions.
- These I/O devices could be used for human interaction such as display panel, touch panels, keyboard, mouse, and track pads, to name a few.
- Another I/O devices could be to connect the system to storage devices, sensors, and so on. There could also be I/O devices for networking needs that implement certain parts of the networking stack. These could be Wi-Fi, Ethernet, and Bluetooth devices and so on.
- They vary from one to another in the form of protocols they use to communicate such as the data format, speed at which they operate, error reporting mechanisms, and more.
- The OS presents a unified I/O system that abstracts the complexity from applications. The OS handles this by establishing protocols and interfaces with each I/O controller. However, *the I/O subsystem usually forms the complex part of the operating system due to the dynamics and the wide variety of I/Os involved*.

4.1 Need for I/O Management



Device controller & device driver

- – A **device driver** **is a code** inside the OS that allows to be empowered with the specific commands needed to operate the associated device.

The code is implemented by the device manufacturer which allows the device to communicate with the computer's OS. Without device drivers, the computer won't be able to able to communicate properly with the hardware devices.

- **Device controller**, on the other hand, is like a bridge between the device and the operating system. It is an **electronic component** consisting of chips which control the device.

http://www.differencebetween.net/technology/difference-between-device-driver-and-device-controller/

I/O Subsystem

- Input/output devices that are connected to the computer are called ***peripheral*** devices.

- It is communicated with the system through the busses: ***Data bus***: to transfer data, ***Address bus***: used to specify address locations, and ***Control bus***: to control a device.

I/O Subsystem (Buses)

- Buses are the means by which data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory.
- A standard CPU system bus is comprised of a control bus, data bus and address bus.

Address Bus	**Carries the addresses of data (but not the data) between the processor and memory**
Data Bus	**Carries data between the processor, the memory unit and the input/output devices**
Control Bus	**Carries control signals/commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer**

I/O Subsystem

- There ***could be different buses or device protocols*** that an operating system may support. The most common protocols include:
  - Peripheral Component Interconnect Express (PCIe) protocol,
  - Inter-Integrated Circuit (I2C), and
  - Advanced Configuration and Power Interface (ACPI)
- ***A device*** can be connected over ***one or more*** of these ***interfaces***.
- Consider the need to send a request to read the temperature of a specific device that is connected via ACPI. In this case, the operating system sends a request to the ACPI subsystem, targeting the device that handles the request and returns the data. This is then passed back to the application.

I/O Subsystem

- Typically, there is a software component in kernel mode called as the "***device driver***" that handles all interfaces with a device.
- It helps with communicating between the device and the OS and abstracts the device specifics.
- Similarly, there could be a driver at the bus level usually referred to as ***the bus driver***. (Ex: USB Bus driver)
- Most OSs include an inbox driver that implements the bus driver.
- There is usually a driver for each controller and each device.

I/O Devices Categories

- The I/O devices can be broadly divided into two categories called:

block devices and ***character*** devices.

Character Devices

- Another class of devices are character devices, the subtle difference is that the communication happens by sending and receiving single characters, which is usually a byte or an octet.
- Many serial port devices like ***keyboards***, some ***sensor devices***, and microcontrollers follow this mechanism.

Block Devices

- These are devices with which the I/O device controller communicates by sending blocks of data.
- A block is referred to as a group of bytes that are referred together for Read/Write purposes.
- Example: ***flash memory***, ***digital camera***
- The device driver would access by specifying the size of Read/Writes which is varying from device to another.

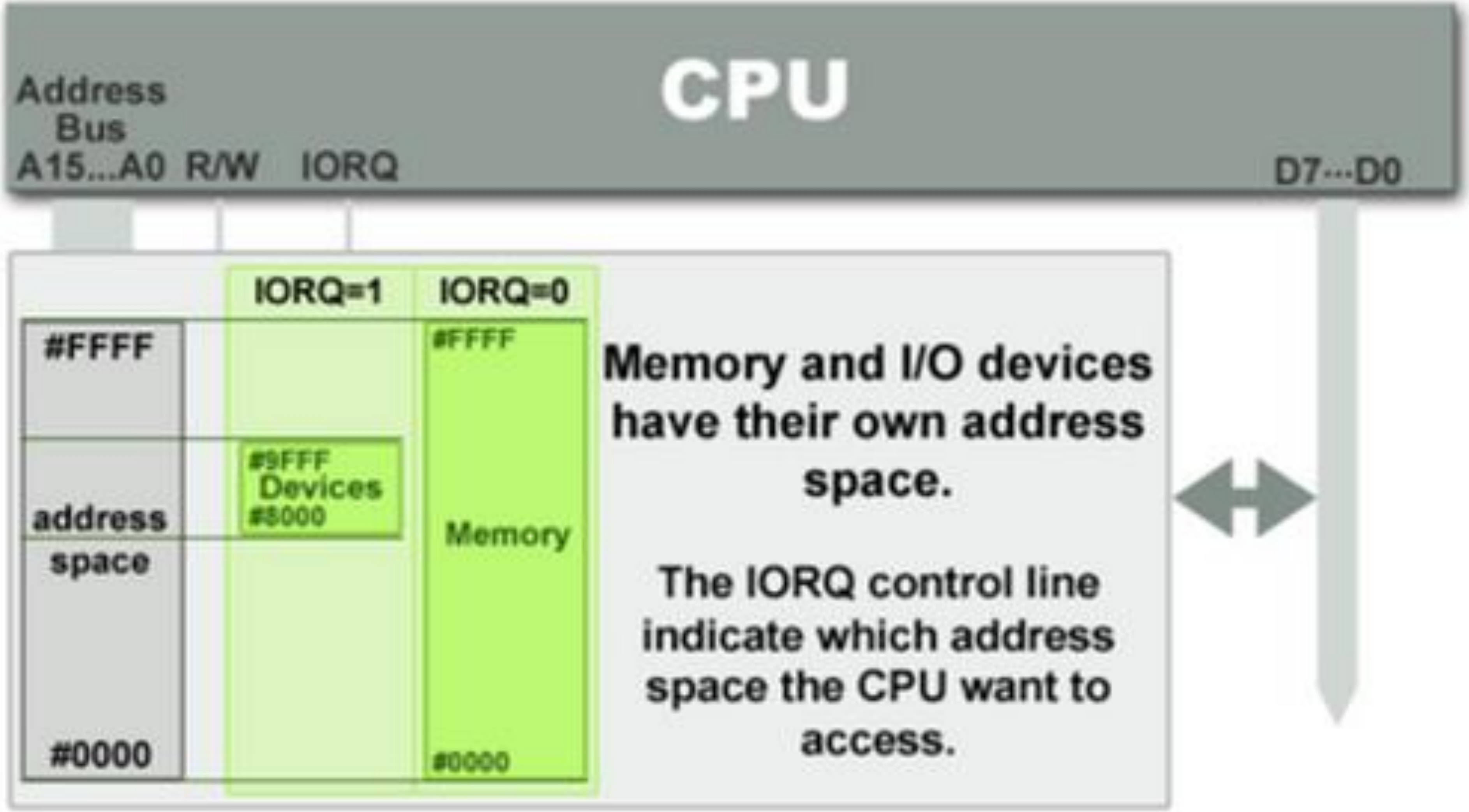
4.4 I/O Protocols Categories

- The protocols used by the different devices (block devices or character devices) could vary from one to another. There are three main categories of I/O protocols that are used:
  - Special Instruction I/O
  - Memory-Mapped I/O
  - Direct Memory Access (DMA)

4.4.1 Special Instruction I/O

- Special Instruction I/O or Port-mapped IO (PMIO)
- There could be specific CPU instructions that are custom developed for communicating with and controlling the I/O devices.
- **Each device has a unique I/O address**
- For example, there could be a CPU-specific protocol to communicate with the embedded controller.
- This may be needed for faster and efficient communication.
- However, such type of I/Os are special and smaller in number.

4.4.1 Special Instruction I/O



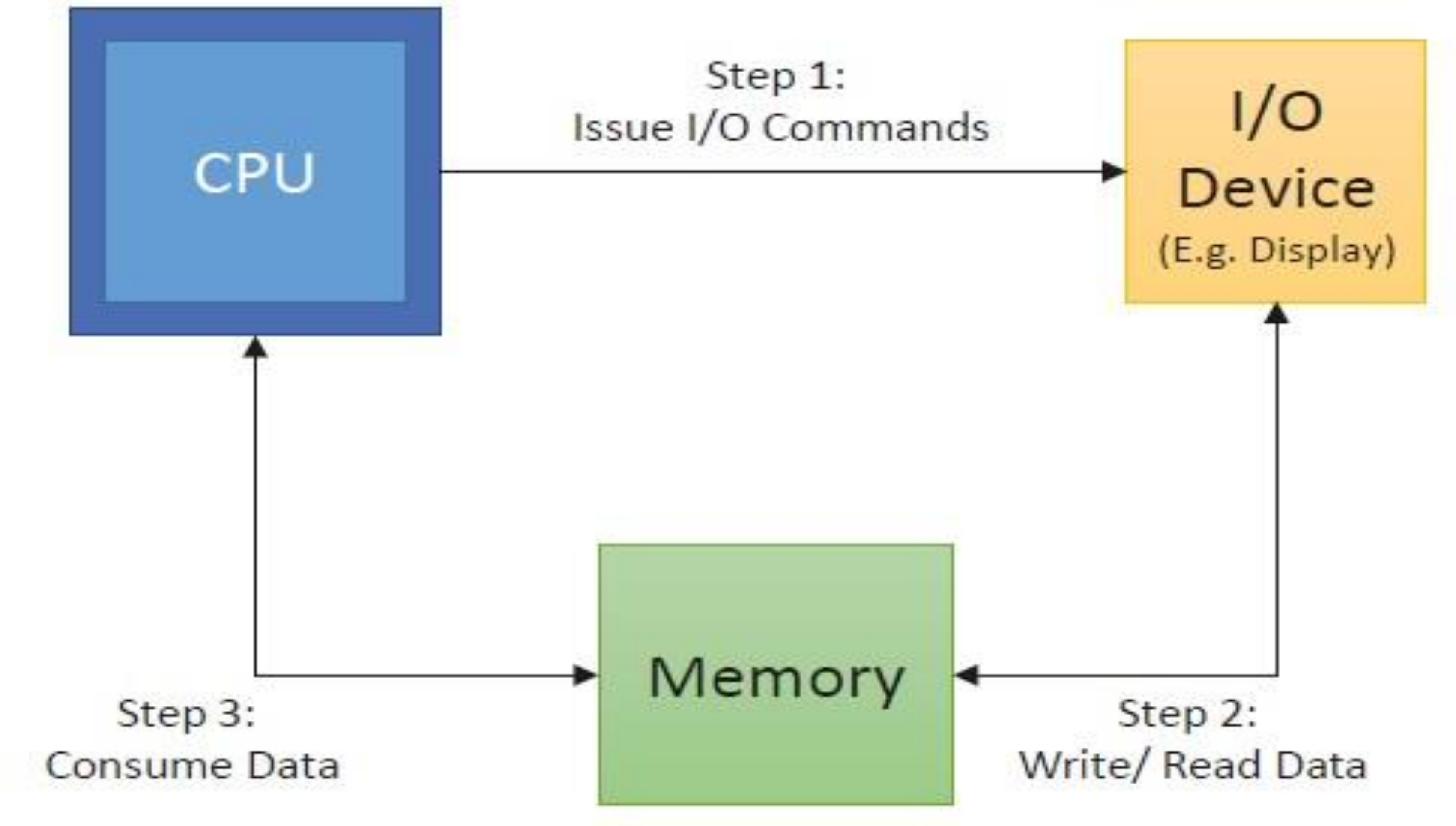
Memory and I/O devices have their own address space.

The IORQ control line indicate which address space the CPU want to access.

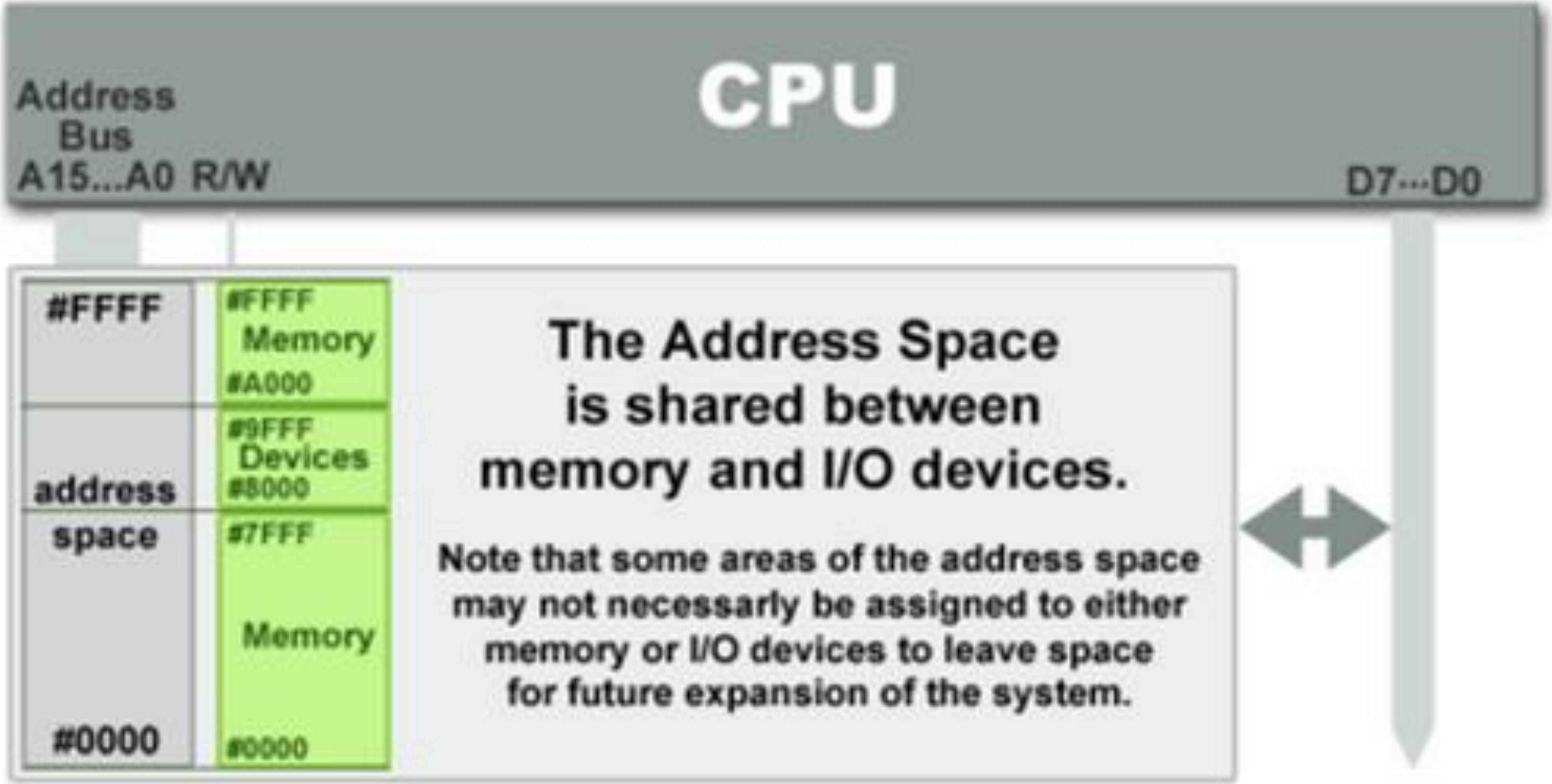
4.4.2 Memory-Mapped I/O

- The most common form of I/O protocol is memory-mapped I/O (MMIO).
- The device and OS agree on a common address range carved out by the OS, and the I/O device makes reads and writes from/to this space to communicate to the OS.
- OS components such as drivers will communicate using this interface to talk to the device.
- MMIO is also an effective mechanism for data transfer that can be implemented without using up precious CPU cycles.
- Hence, it is used to enable high-speed communication for network and graphics devices that require high data transfer rates due to the volume of data being passed.

4.4.2 Memory-Mapped I/O



4.4.2 Memory-Mapped I/O



The Address Space is shared between memory and I/O devices.

Note that some areas of the address space may not necessarly be assigned to either memory or I/O devices to leave space for future expansion of the system.

4.4.3 Direct Memory Access (DMA)

- Direct memory access (DMA) is a method that allows an input/output (I/O) device to send or receive data directly to or from the main memory, bypassing the CPU to speed up memory operations.

- The process is managed by a chip known as a DMA controller (DMAC).

- DMA Controller is a hardware device that allows I/O devices to directly access memory with less participation of the processor.

Direct Memory Access Structure

- A DMA channel enables a device to transfer data without exposing the CPU to a work overload. Without the DMA channels, the CPU copies every piece of data using a peripheral bus from the I/O device.
- Using a peripheral bus occupies the CPU during the read/write process and does not allow other work to be performed until the operation is completed.
- With DMA, the CPU can process other tasks while data transfer is being performed. The transfer of data is first initiated by the CPU. The data block can be transferred to and from memory by the DMAC in three ways.

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds.
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention.
- Only one interrupt is generated per block, rather than the one interrupt per byte.

4.4.3 Direct Memory Access (DMA)

- There could be devices that run at a slower speed than supported by the CPU or the bus it is connected on. In this case, the device can leverage DMA.
- Here, the OS *grants authority to another controller*, usually referred to as the *direct memory access controller*, to *interrupt the CPU after a specific data transfer is complete*.
- The devices running at a smaller rate can communicate back to the DMA controller after completing its operation.
- Note: Most OSs also handle additional specific device classes, blocking and nonblocking I/Os, and other I/O controls.
  - *As a programmer, you could be interacting with devices that may perform caching (an intermediate layer that acts as a buffer to report data faster) and have different error reporting mechanisms, protocols, and so on.*

What is an Interrupt?

An interrupt is a signal which is sent from a device or from software to the operating system.

The interrupt signal causes the operating system to temporarily stop what it is doing and ‘service’ the interrupt.

What is an Interrupt?

- **Definition:** It is referred to as an input signal that has the highest priority for hardware or software events that requires immediate processing of an event.
- A new mechanism was introduced to overcome this complicated process. In this mechanism, hardware or software will send the signal to a processor, rather than a processor checking for any signal from hardware or software (**pooling**).
- The signal alerts the processor with the highest priority and suspends the current activities by saving its present state and function, and processes the interrupt immediately, this is known as ISR. As it doesn’t last long, the processor restarts normal activities as soon as it is processed.

Difference between Interrupt and Polling

In the case of an interrupt, the system informs the CPU that it needs attention While in polling, the CPU constantly inspects the status of the system to find whether it needs attention.

In Polling

you pick up the phone every few seconds to check whether you are getting a phone

In interrupt:

do whatever you should do and pick up the phone every when it is rings.

Difference between Interrupt and Polling

In Polling

```
int main()
{
  while(1)
  {
     int count = 1;
     while(count < 500000)
       count++;
     toggle(PIN_5);
  }
  return 0;
}
```

Polling is an affair that informs the CPU that a system needs its attention. It is a constant action to find out whether the system is working efficiently and properly.

In interrupt:

```
int main()
{
  WFI();
  return 0;
}
```

interrupt is an affair that indicates the CPU to take prompt action. In other words, we can say that the device notifies the CPU that it needs its attention. When an interrupt happens, the CPU usually holds its current task and starts executing the corresponding interrupt handler. After completing this task it resumes the old task which was paused.

Difference between Interrupt and Polling

S.No	Interrupt	Polling
1.	When it comes to an interrupt, the device informs the CPU that it needs its attention.	When it comes to polling, the CPU keeps on checking if the device needs attention.
2.	It is a hardware mechanism, not a protocol.	It is a protocol and not a hardware mechanism.
3.	It can occur at any time and moment.	Here the CPU polls the system at some particular intervals.
4.	In interrupts, processor is simply disturbed once any device interrupts it.	On the opposite hand, in polling, processor waste countless processor cycles by repeatedly checking the command-ready little bit of each device.

Types of Interrupt (Hardware / Software Interrupts)

1- Hardware Interrupts

An electronic signal sent from an external device or hardware to communicate with the processor indicating that it requires immediate attention. For example, strokes from a keyboard or an action from a mouse invoke hardware interrupts causing the CPU to read and process it. So it arrives asynchronously and during any point of time while executing an instruction.

- **For example (Key press by the user, e.g. CTRL ALT DEL)**

Hardware interrupts are classified into two types

- **Maskable Interrupts**
- **Non-maskable Interrupts (NMI)**

Hardware Interrupts (Maskable and Non-Maskable)

1. **Maskable Interrupt :**
   An Interrupt that **can be disabled or ignored** by the instructions of CPU are called as Maskable Interrupt.

1. **Non-Maskable Interrupt :**
   An interrupt that **cannot be disabled or ignored** by the instructions of CPU are called as Non-Maskable Interrupt. A Non-maskable interrupt is often used when response time is critical or when an interrupt should never be disable during normal system operation. **Simply is the highest priority activities that need to be processed immediately and under any situation.**

Types of Interrupt (Hardware / Software Interrupts)

2- Software Interrupts

- The processor itself requests a software interrupt after executing certain instructions or if particular conditions are met.

- These can be a specific instruction that triggers an interrupt such as subroutine calls and can be triggered unexpectedly because of program execution errors, known as exceptions or traps.

  A trap is a software generated interrupt.

  An operating system is interrupt driven.

Examples

- division by zero or invalid memory access
- A cout (or) cin statement in C++ would generate a software interrupt because it would make a system call to print something

Interrupt Handling

1. CPU is interrupted
2. CPU stops current process
3. CPU transfers execution to a fixed location
4. CPU executes interrupt service routine
5. CPU resumes process

- Notes:
  - Interrupts must be handled quickly
  - Interrupted process information must be stored

Interrupt Handling Mechanisms

The interrupt handler is the part of the operating system which is responsible for dealing with interrupt signals.

The interrupt handler prioritizes interruptions as they are received, placing them into a queue as necessary.

For every interruption, the current task needs to be stopped, with it’s status saved (so it can resume later).

4.5 Interrupt Handling Mechanisms

- Polling or programmed I/O:
  - In this mechanism, each period of time examine interrupt information and calls a specific routine (driver)
- Interrupt-Driven I/O:
  - In This mechanism, there is an ***interrupt vector*** which contains the addresses for all ***Interrupt Service Routines*** (ISR) –***drivers***- for all I/O devices, according the ***Interrupt Request Number*** (IRQ) –which ***unique*** for each device- the OS acquire the address of the address of the correct service routine
  - Most of OSs using this mechanism

4.5 Interrupt Handling Mechanisms

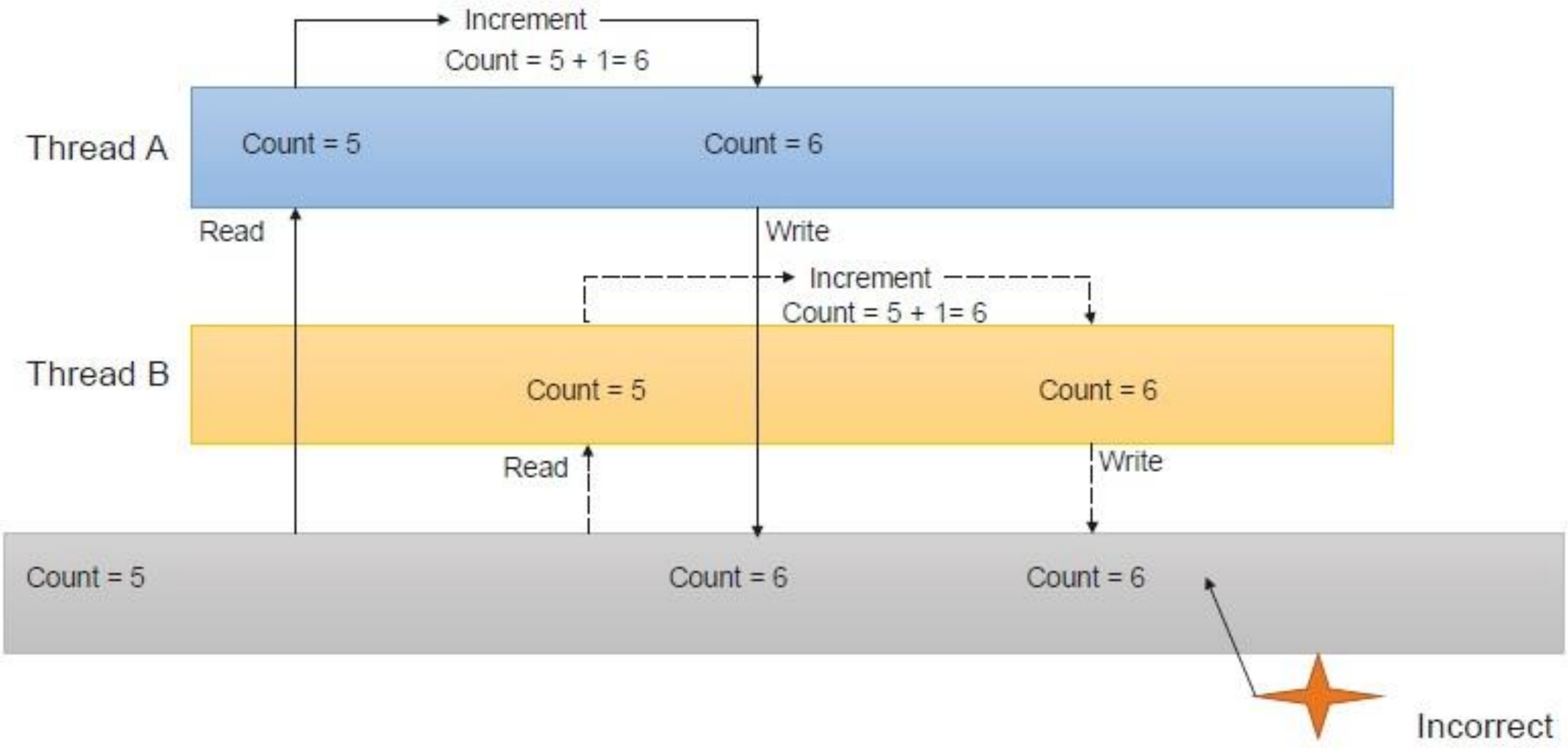
- Synchronous I/O: Example: Printing Operation
  - Process request I/O operation
  - I/O operation is started
  - I/O Operation is complete
  - Control is returned to the user process
- Asynchronous I/O: Example: Using Networking sending and receiving using threads
  - Process Request I/O operation
  - I/O operation is started
  - Control is returned immediately to the user process
  - I/O continues while system operations occur

4.7 Synchronization and Critical Sections

- In multi-threaded applications, if one thread tries to change the value of shared data at the same time as another thread tries to read the value, there could be a race condition across threads.
- In this case, the result can be unpredictable.
- The access to such shared variables via shared memory, files, ports, and other I/O resources needs to be synchronized to protect it from being corrupted.

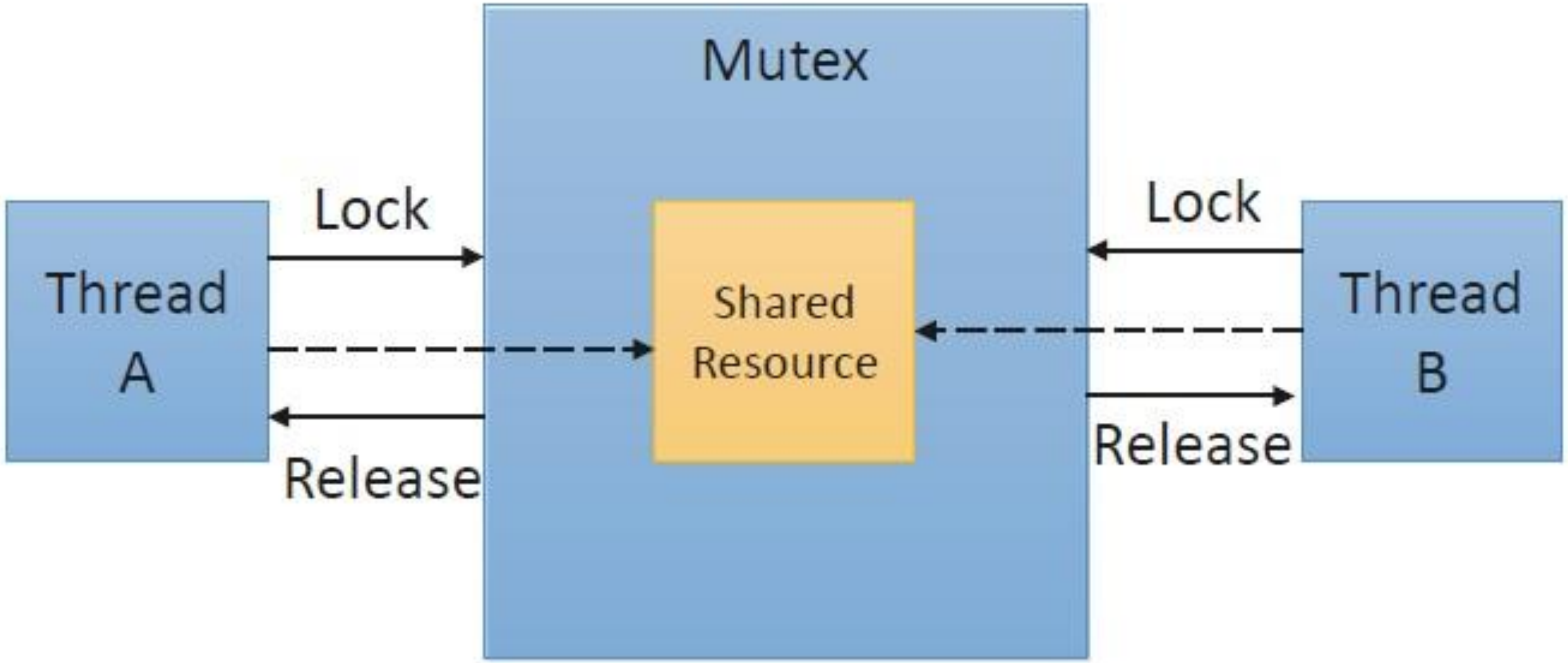
- order to support this, the operating system provides mutexes and semaphores to coordinate access to these shared resources.

4.7 Synchronization and Critical Sections



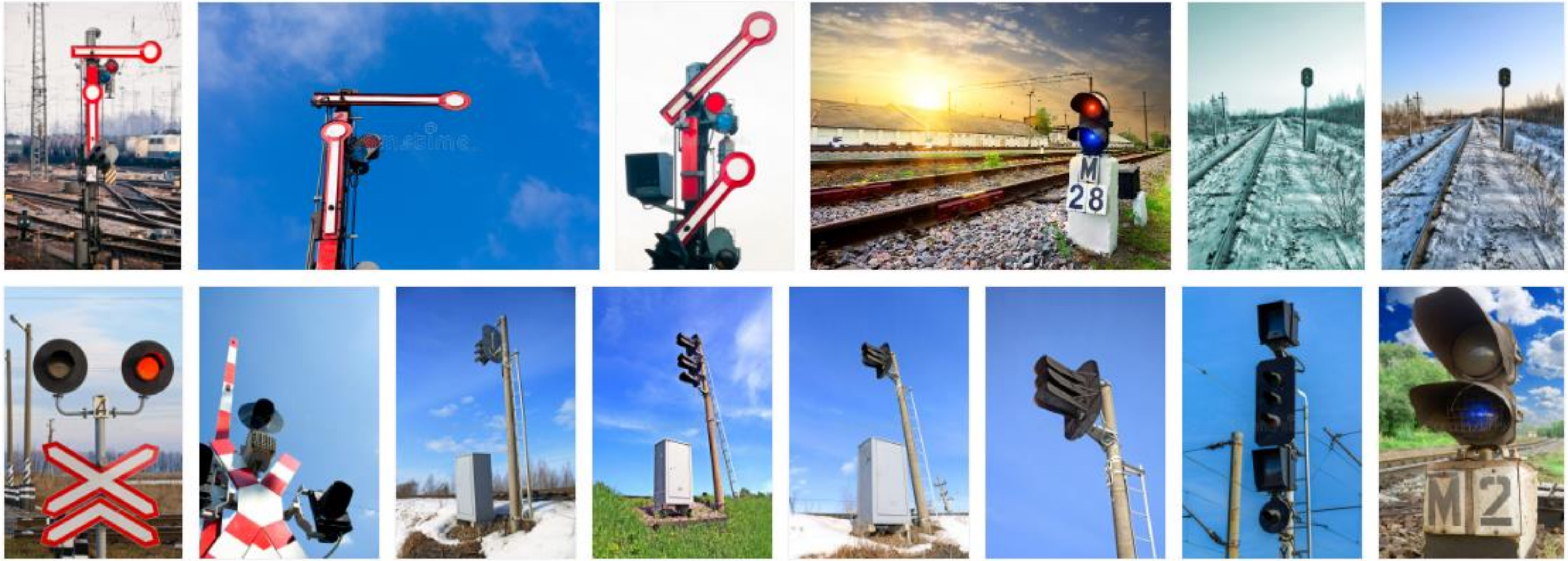
4.7.1 Mutex

- A mutex is used for implementing ***mutual exclusion***: either of the participating processes or threads can have the key (mutex) and proceed with their work.
- The other one would have to wait until the one holding the mutex finishes.



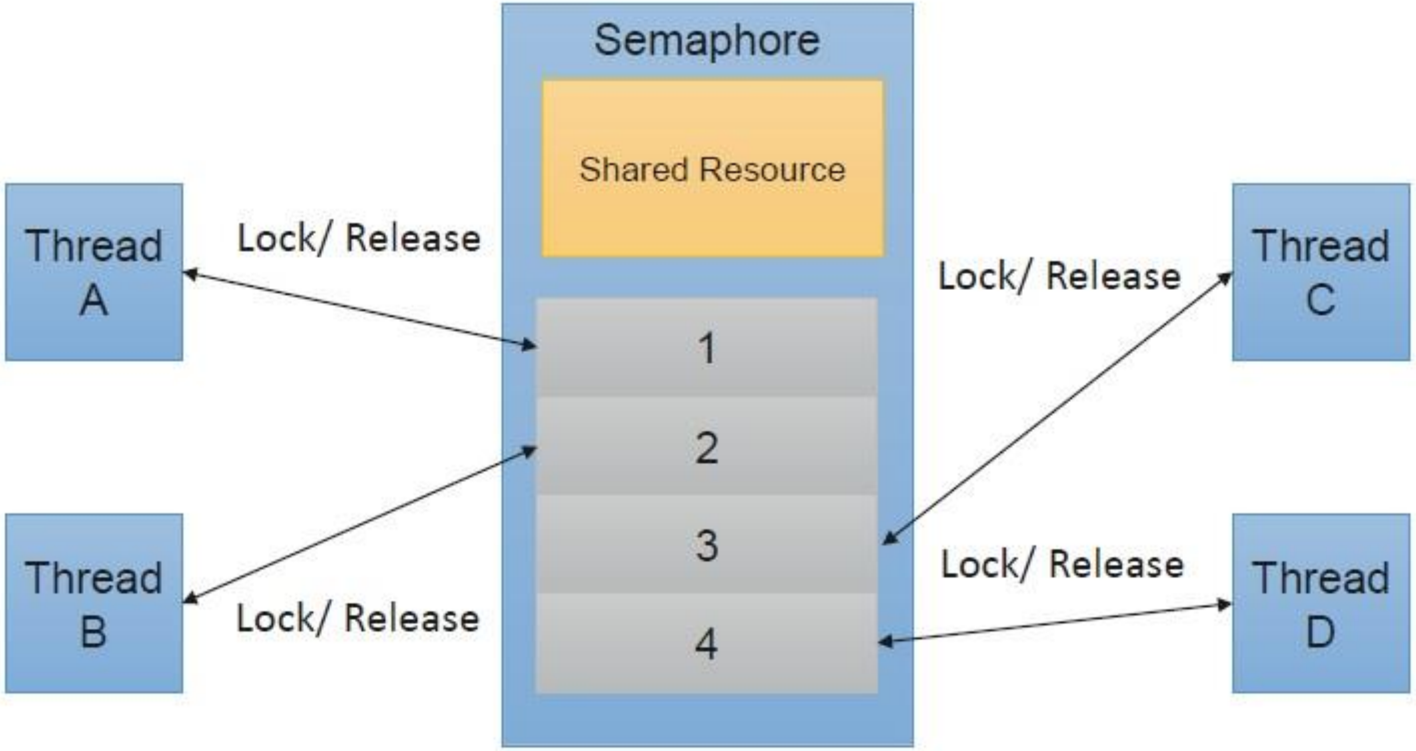
4.7.2 Semaphore

- A semaphore is a generalized mutex. A binary semaphore can assume a value of 0/1 and can be used to perform locks to certain critical sections.



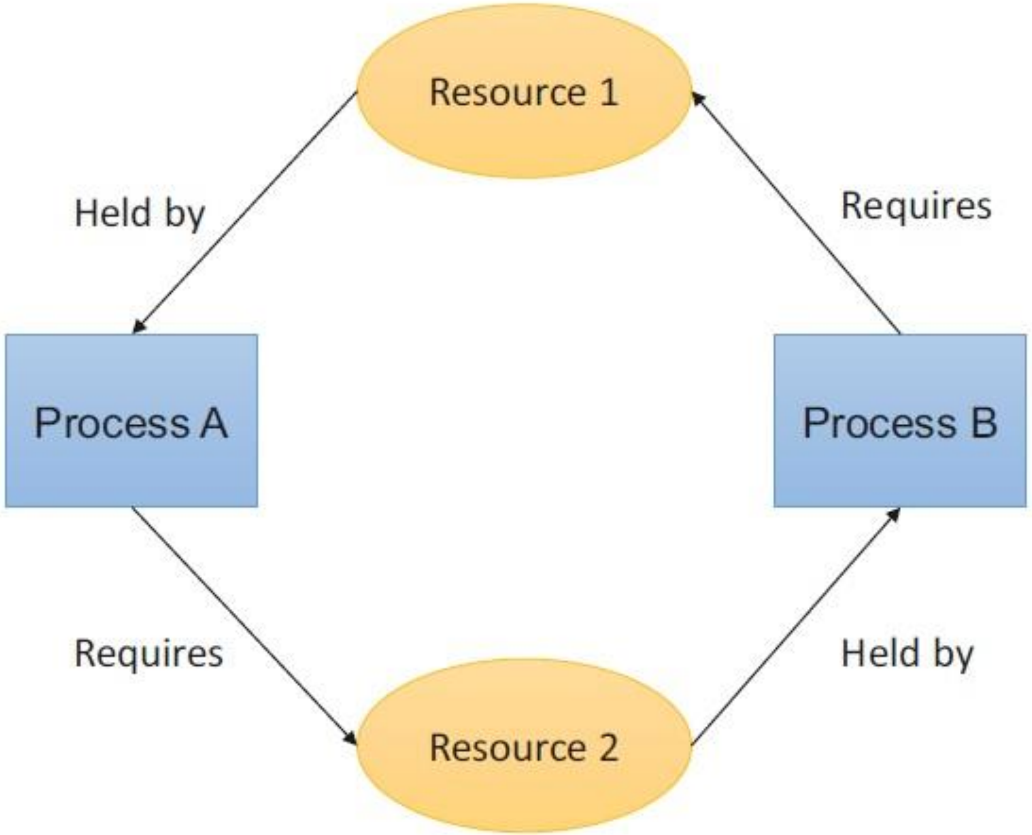
4.7.2 Semaphore

- A semaphore is a generalized mutex. A binary semaphore can assume a value of 0/1 and can be used to perform locks to certain critical sections.



4.8 Deadlocks

- When a set of processes become blocked because each process is holding a resource and waiting for another resource acquired by some other process. This is called as a ***deadlock***.
- Process A holds Resource 1 and requires Resource 2. However, Process B already is holding Resource 2, but requires Resource 1. Unless either of them releases their resource, neither of the processes may be able to move forward with the execution.



4.8 Deadlocks

- A deadlock can arise if the following four conditions hold:
  - **Mutual Exclusion**: There is at least one resource on the system that is not shareable. This means that only one process can access this at any point in time. In the preceding example, Resources 1 and 2 can be accessed by only one process at any time.
  - **Hold and Wait**: A process is holding at least one resource and is waiting for other resources to proceed with its action. In the preceding example, both Processes A and B are holding at least one resource.
  - **No Preemption**: A resource cannot be forcefully taken from a process unless released automatically.
  - **Circular Wait**: A set of processes are waiting for each other in circular form.

File Systems

Content

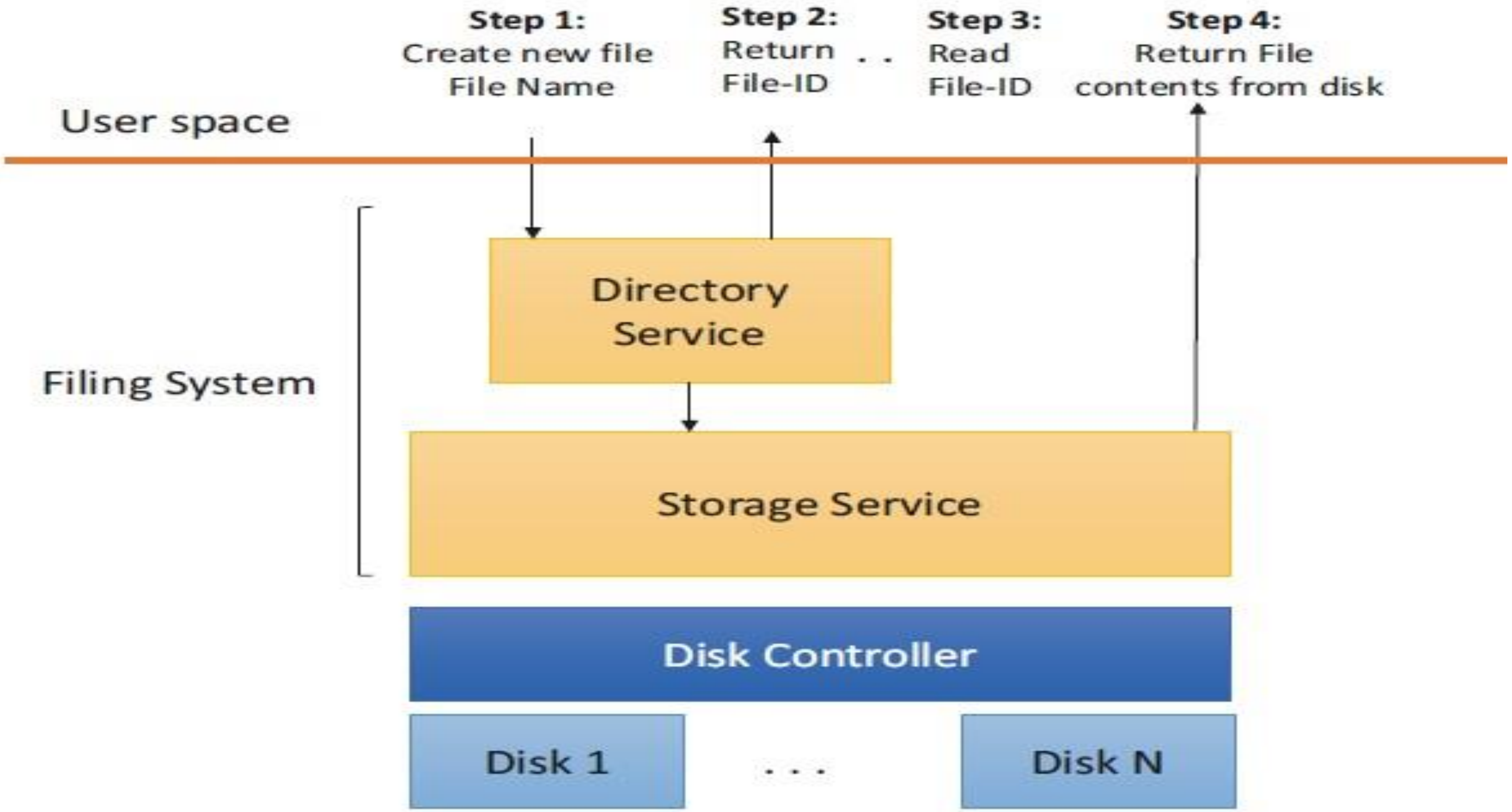
- Need for File Systems
- File Concept
- Directory Name Space
- Access Control
- Concurrency

Course Outlines

5.1 Need for File Systems

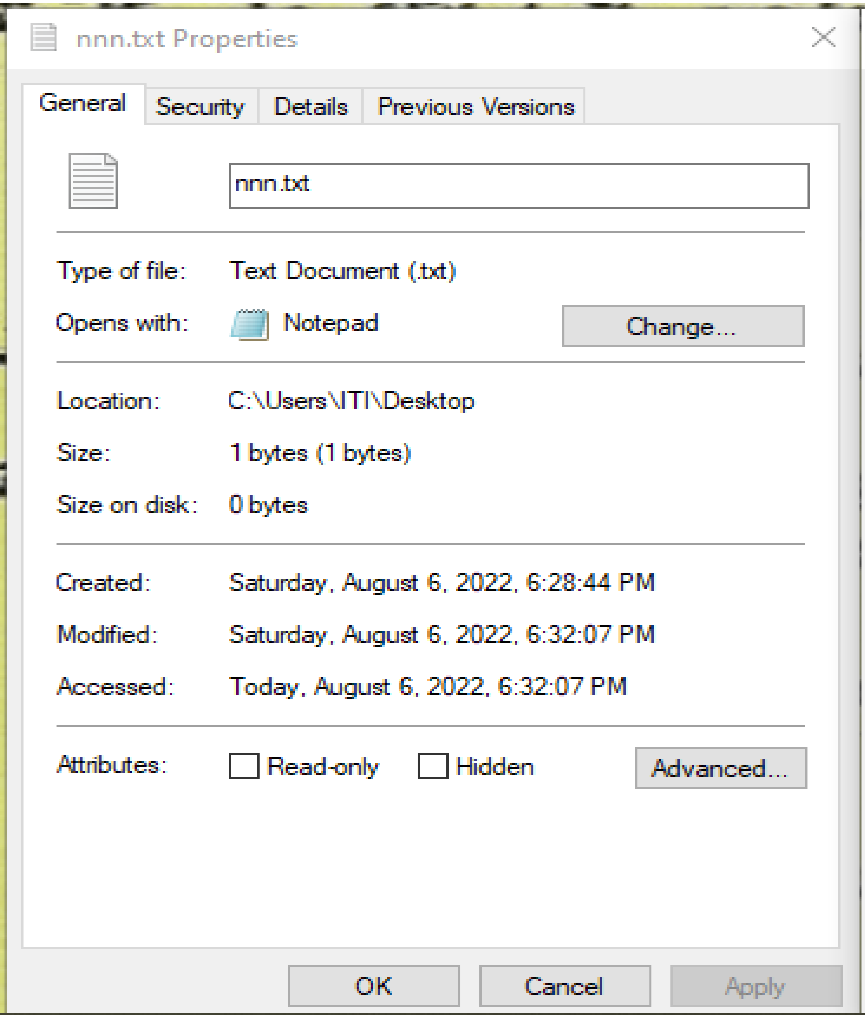
- Applications often need to read and write files to achieve their goals. We leverage the OS to create, read, and write such files on the system. We depend on the OS to maintain and manage files on the system.
- OS file systems have two main components to facilitate file management:
  - ***Directory Service***: There is a need to uniquely manage files in a structured manner, manage access, and provide **Read-Write-Edit controls** on the file system. This is taken care by a layer called as the directory service.
  - ***Storage Service***: There is a need to **communicate to** the underlying **hardware such as the disk**. This is managed by a storage service that abstracts different types of storage devices on the system.

5.1 Need for File Systems



5.2 File Concept

Example on different formats (txt file, word file and PDF file) with the same content (one letter “a”) with different sizes.



nnn.txt Properties

nnn.txt

Type of file: Text Document (.txt)

Opens with: Notepad

Location: C:\Users\ITI\Desktop

Size: 1 bytes (1 bytes)

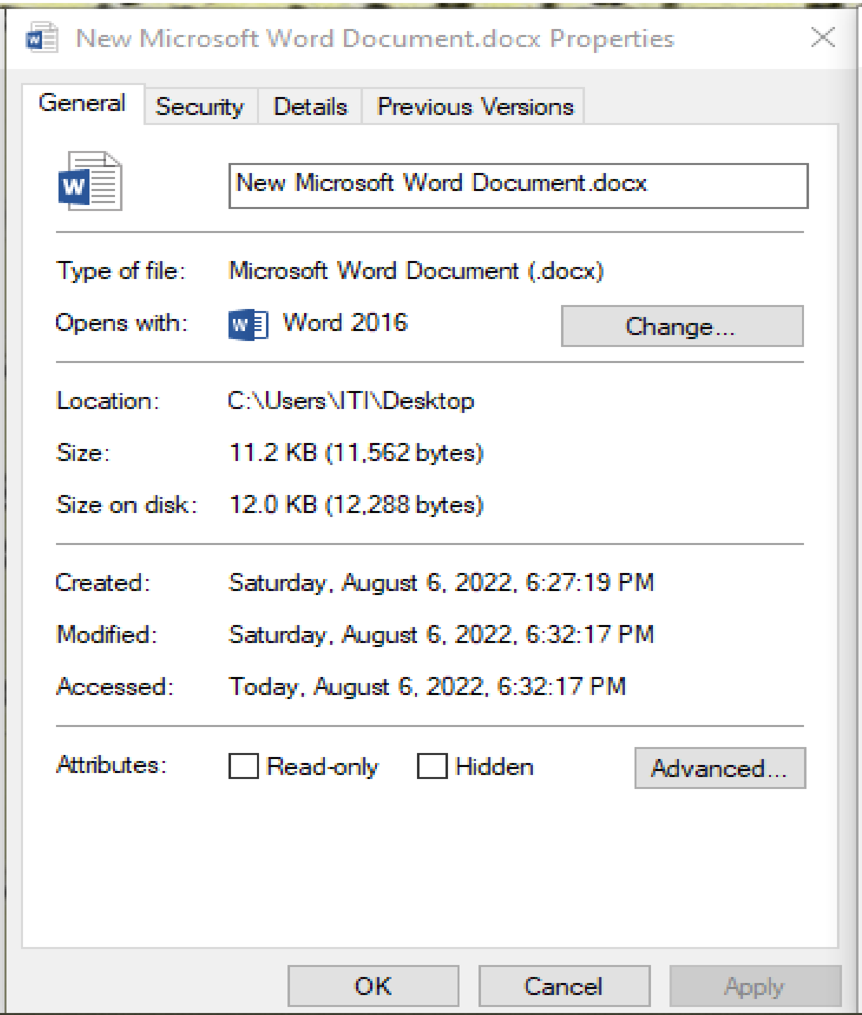
Size on disk: 0 bytes

Created: Saturday, August 6, 2022, 6:28:44 PM

Modified: Saturday, August 6, 2022, 6:32:07 PM

Accessed: Today, August 6, 2022, 6:32:07 PM

Attributes: □ Read-only □ Hidden



New Microsoft Word Document.docx Properties

New Microsoft Word Document.docx

Type of file: Microsoft Word Document (.docx)

Opens with: Word 2016

Location: C:\Users\ITI\Desktop

Size: 11.2 KB (11,562 bytes)

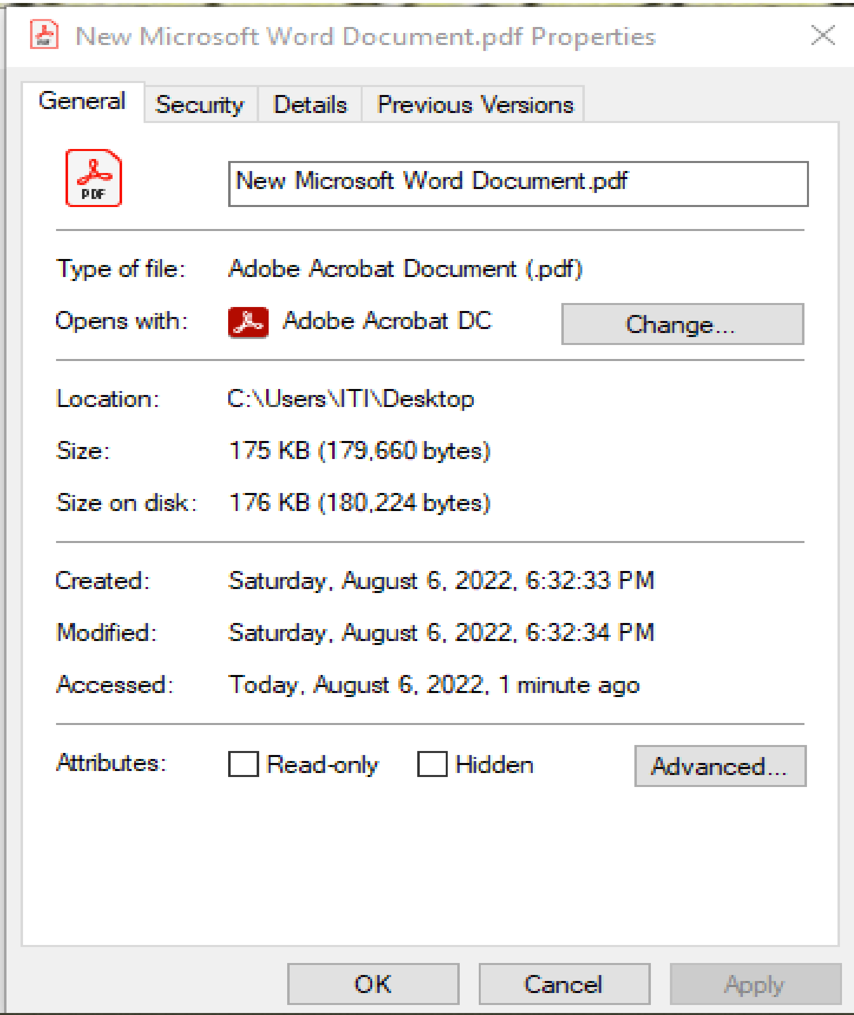
Size on disk: 12.0 KB (12,288 bytes)

Created: Saturday, August 6, 2022, 6:27:19 PM

Modified: Saturday, August 6, 2022, 6:32:17 PM

Accessed: Today, August 6, 2022, 6:32:17 PM

Attributes: □ Read-only □ Hidden



New Microsoft Word Document.pdf Properties

New Microsoft Word Document.pdf

Type of file: Adobe Acrobat Document (.pdf)

Opens with: Adobe Acrobat DC

Location: C:\Users\ITI\Desktop

Size: 175 KB (179,660 bytes)

Size on disk: 176 KB (180,224 bytes)

Created: Saturday, August 6, 2022, 6:32:33 PM

Modified: Saturday, August 6, 2022, 6:32:34 PM

Accessed: Today, August 6, 2022, 1 minute ago

Attributes: □ Read-only □ Hidden

5.2 File Concept

- From the perspective of the user, a file is a collection of related data that is stored together and can be accessed using a unique file ID usually referred as the file name.
- These files can be represented internally by different methods. For example, there could be **.bin** files in Windows, which only represent a sequence of bytes.
- There are also many **application-specific files**, with their own formats. It is up to the programmer to define and identify if they require a custom file format for their application or if they can leverage a standard or common file format such as the JavaScript Object Notation (**JSON**) or the Extensible Markup Language (**XML**).

Ref: https://www.uspto.gov/ebc/portal/infofilesize.htm

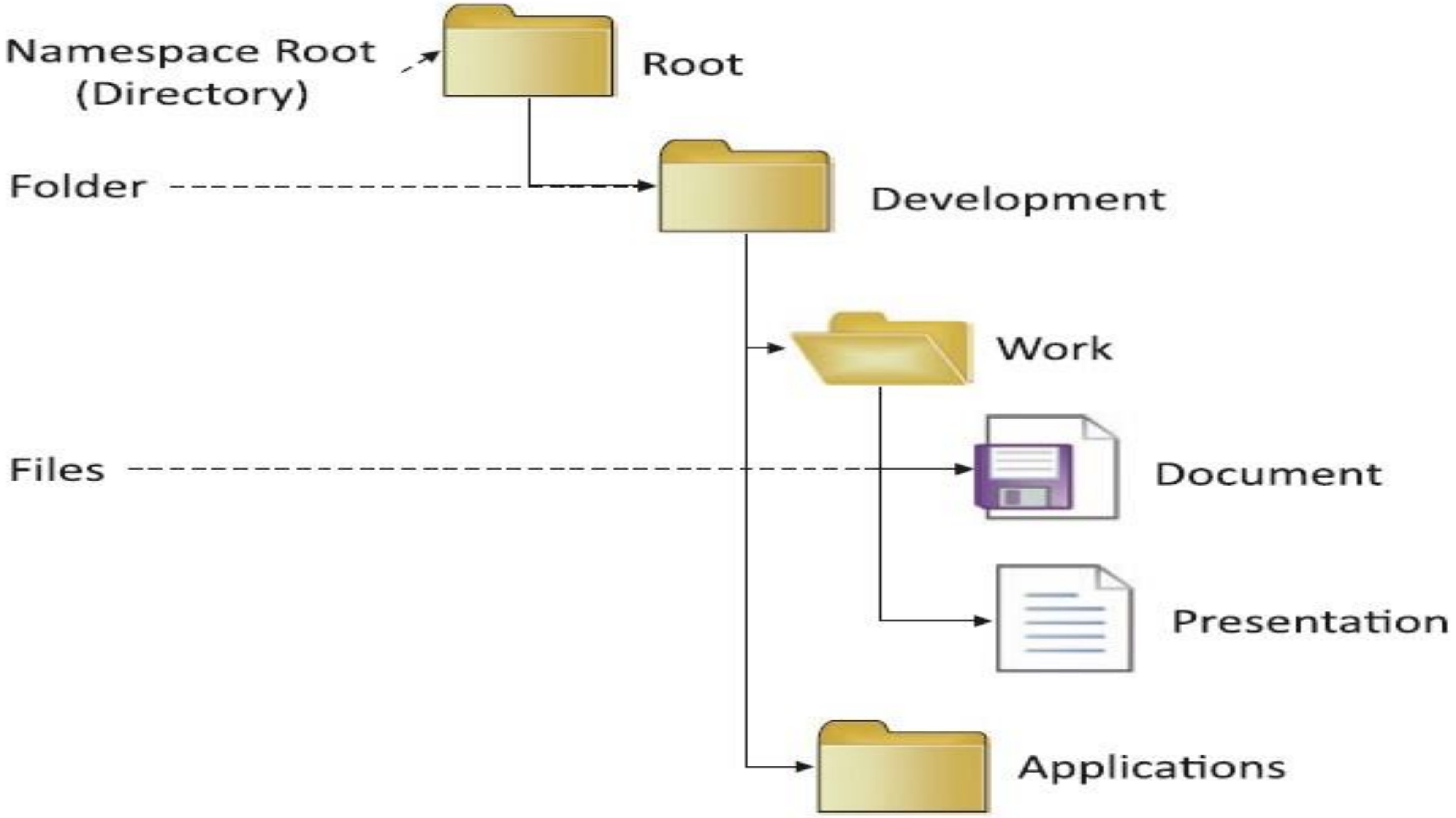
5.2 File Concept

- As a programmer, it may be important to know the **attributes** of the file before accessing it.
- The common attributes of any file include the ***location*** of the file, file ***extension***, ***size***, ***access controls***, and some ***history of operations*** done on the file, to name a few.
- Some of these are part of the so-called ***file control block***, which a user has access to via the OS.
- Most OSs expose APIs using which the programmer can access the details in the file control block.
- For the user, these are exposed on the ***graphical user interface via built-in tools*** shipped with the OS.

5.3 Directory Name Space

- The operating system defines a logical ordering of different files on the system based on the usage and underlying storage services. One of the criteria most OSs adopt is to structure their directory service to locate files efficiently.
- Most OSs ***organize*** their files in a ***hierarchical*** form with files organized inside ***folders***.
- Each folder in this case is a directory. This structure is called as the ***directory namespace***.
- The directory service and namespace have additional capabilities such as searches by size, type, access levels, and so on.
- The directory namespaces can be ***multileveled*** and adaptive in modern OSs as we can see in the following folder structure with folders created inside another folder

5.3 Directory Name Space



5.4 Access Control

- There are different access levels that can be applied at file and directory levels.
- The OS provides different access control ***IDs*** and ***permissions*** to different users on the system.
- Also, each file may also have ***different levels of permissions*** to Read, Write, Modify, and so on.
- For example, there may be specific files that we may want anyone to be able to access and Read but not Write and Modify.
- The file system provides and manages the controls to all files when accessed at runtime.
- These may also be ***helpful*** when ***more than one user*** is using the same system.

5.5 Concurrency and Cleanup Control

- There are many cases when the OS needs to ***ensure that a file is not moved or deleted when it is in use***.
- For example, if a user is making changes to a file, the OS needs to ensure that the same file cannot be moved or deleted by another application or process. In this case, the OS would cause the attempt to move or delete the file to fail with an appropriate error code.
- *As a programmer, it is appropriate to access a file with the required* ***access level*** *and* ***mode*** *(Read/Write). This also helps to be in line with the concurrency needs of the OS and guards against inconsistent updates.*

5.5 Concurrency and Cleanup Control

- The OS needs to be able to ***periodically clear temporarily created files*** that may no longer be required for the functioning of the system.
- This is typically done using a ***garbage collector on the system***.
- Many OSs ***mark unused files over a period of time*** and have additional settings that are exposed, which the user can set to clean up files from specified locations automatically.

Access and Protection

Content

- Access and Protection
  - User Mode and Kernel Mode (Rings)

Course Outlines

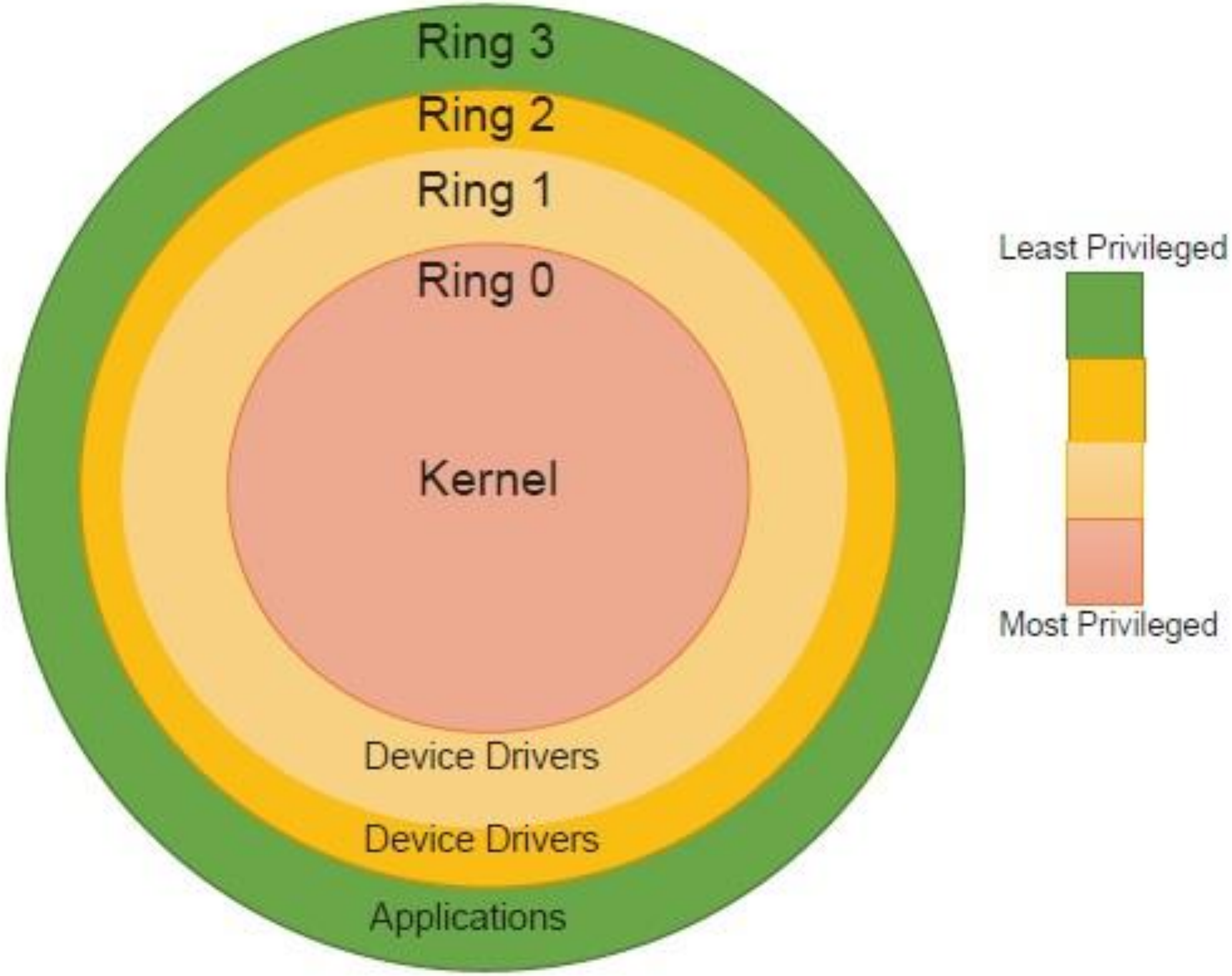
6.1 Access and Protection

- If we have a system that is used by only one user without any access, networked or otherwise, to other systems, there may still not be assurance that the contents in the system are protected.
- There is still a need to protect the ***program resources from other applications***.
- Also, there may be a need to protect ***critical devices*** on the system.
- There is always a need to connect and ***share resources and data between systems***.
- Hence, it is important to protect these resources accordingly.
- The OS provides APIs that help with access control and protection.

6.2 User Mode and Kernel Mode (Rings)

- One of the reasons the separation between user mode and kernel mode is implemented by most OSs is that it ensures ***different privilege levels are granted to programs, based on which mode they run in***.
- OS divides the program execution privileges into ***different rings***.
- Internally, programs running in ***specific rings are associated with specific access levels and privileges***.
- For example, ***applications and user-mode*** services running in *Ring 3* would ***not be able to access the hardware directly***. The ***drivers*** running on the *Ring 0* level would have the ***highest privileges and access to the hardware*** on the system.
- In practice, most OSs only leverage two rings, which are Ring 0 and Ring 3.

6.2 User Mode and Kernel Mode (Rings)



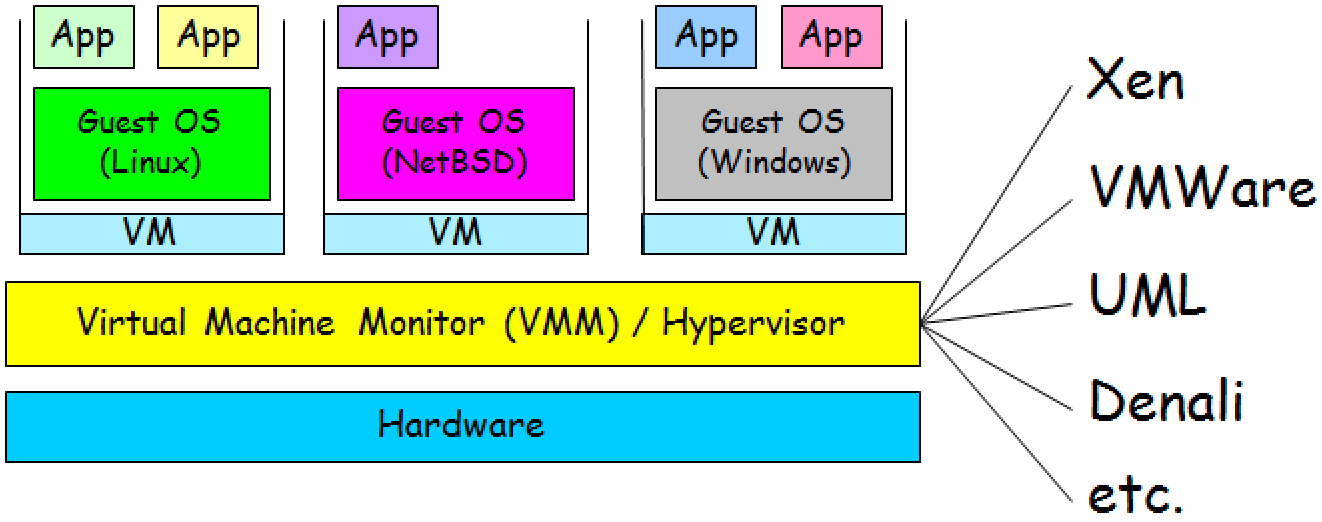
Virtualization and User Interface

7.1 Virtualization

- Operating systems and modern hardware provide a feature called virtualization that ***virtualizes the hardware*** such that each calling environment believes it has the dedicated access it needs to function.
- Virtualization is delivered via so-called ***virtual machines*** (VMs).
- A VM has its own ***guest OS***, which may be the same as or different from the underlying ***host OS***.
- A user can launch a VM, much like running any other program, and log into the guest OS.
- The ***host OS*** provides a ***hypervisor***, which manages the access to the hardware.
- The guest OS is usually unaware of the internals and passes any resource/hardware requests to the host OS.
- The user can completely ***customize their VM and perform all their actions*** on this VM ***without affecting the host OS*** or any other VM on the system.

Virtualization

- VM technology allows multiple virtual machines to run on a single physical machine



7.1 Virtualization

- At a high level, VMs help effectively utilize the hardware resources and are used heavily in server and cloud deployments.
- Advantages of virtualization:
  - Run operating systems where the ***physical hardware is unavailable***.
  - Easier to create ***new machines***, ***backup machines***, etc.
  - ***Software testing*** using “clean” installs of operating systems and software.
  - ***Emulate more machines*** than are physically available.
  - ***Debug problems*** (suspend and resume the problem machine).
  - ***Easy migration of virtual machines*** (shutdown needed or not).
  - Run ***legacy systems***

7.3 User Interface

- Although the ***user interface*** (UI) is not part of the OS kernel itself.

- There can be multiple user interfaces for the OS all being implemented either as **command line** interface or a ***graphical-based*** interface

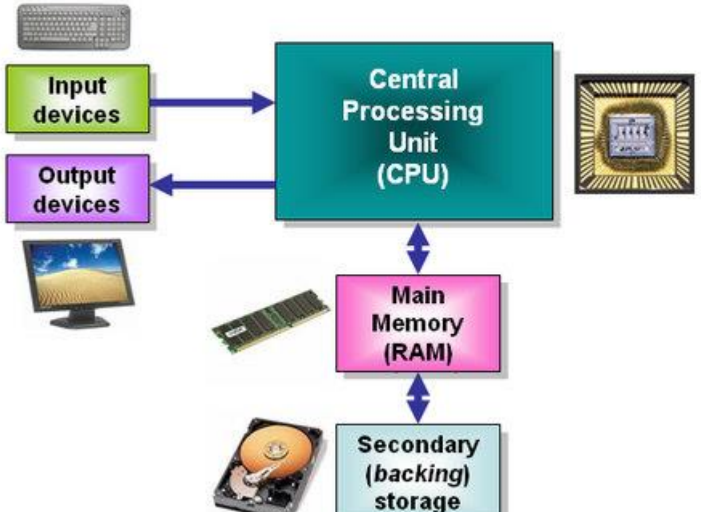
- The **graphical user interface** is the rich set of graphical front-end interfaces and functionalities provided by the OS for the user to interact with the computer.

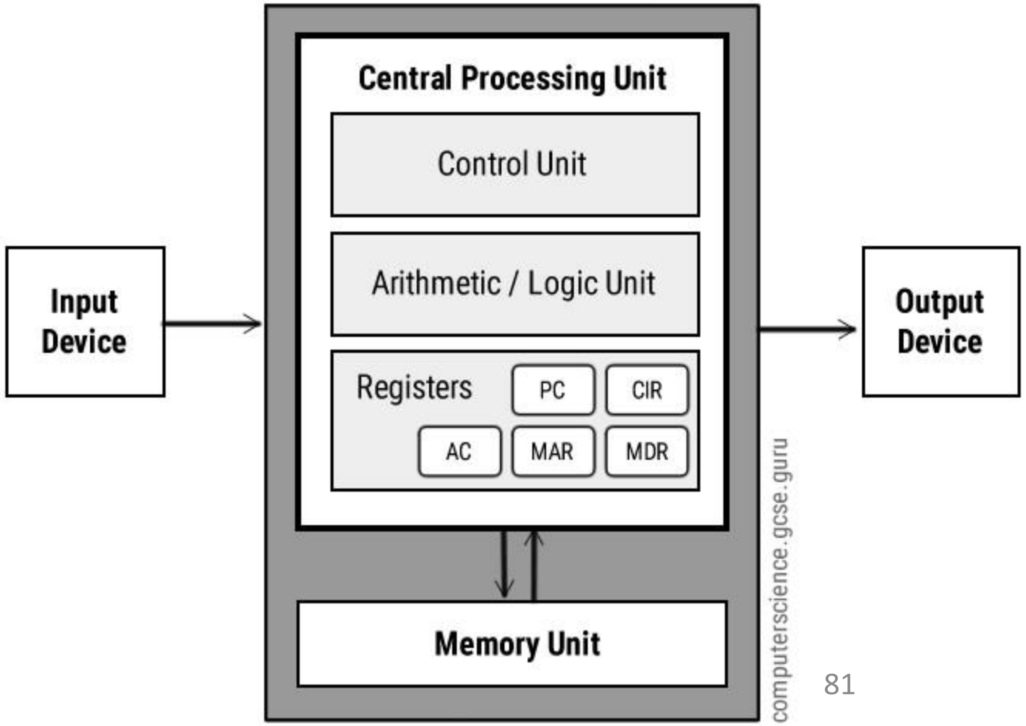
- There could be an alternate simpler interface through a **command line** that most OSs also provide for communication. This is a text-based interface.

- *It is common for programmers to use the shell interface instead of the GUI for quickly traversing through the file system and interacting with the OS.*

Introduction to Computer Architecture

- Von Neumann architecture was first published by John von Neumann in 1945.
- His computer architecture design consists of a Control Unit (CU), Arithmetic and Logic Unit (ALU), Memory Unit, Registers and Inputs/Outputs.
- Von Neumann architecture is based on the stored-program computer concept, where instruction data and program data are stored in the same memory. This design is still used in most computers produced today.





computerscience.gcse.guru

1.2.1 Central Processing Unit (CPU)

- The Central Processing Unit (CPU) is the electronic circuit responsible for executing the instructions of a computer program.
- It is sometimes referred to as the microprocessor or processor.
- The CPU contains the ALU, CU and a variety of registers.
- **1.2.1.1 Registers**
  - Registers are high speed storage areas in the CPU. All data must be stored in a register before it can be processed.

MAR	**Memory Address Register**	**Holds the memory location of data that needs to be accessed**
MDR	**Memory Data Register**	**Holds data that is being transferred to or from memory**
AC	**Accumulator**	**Where intermediate arithmetic and logic results are stored**
PC	**Program Counter**	**Contains the address of the next instruction to be executed**
CIR	**Current Instruction Register**	**Contains the current instruction during processing**

1.2.1 Central Processing Unit (CPU)

- **1.2.1.2 Arithmetic and Logic Unit (ALU)**
  - The ALU allows arithmetic (add, subtract etc) and logic (AND, OR, NOT etc) operations to be carried out.
- **1.2.1.3 Control Unit (CU)**
  - The control unit controls the operation of the computer's ALU, memory and input/output devices, telling them how to respond to the program instructions it has just read and interpreted from the memory unit.
  - The control unit also provides the timing and control signals required by other computer components.

1.2.3 Buses

- Buses are the means by which data is transmitted from one part of a computer to another, connecting all major internal components to the CPU and memory.
- A standard CPU system bus is comprised of a control bus, data bus and address bus.

Address Bus	**Carries the addresses of data (but not the data) between the processor and memory**
Data Bus	**Carries data between the processor, the memory unit and the input/output devices**
Control Bus	**Carries control signals/commands from the CPU (and status signals from other devices) in order to control and coordinate all the activities within the computer**

1.2.4 Memory Unit

- The memory unit consists of RAM (Random Access Memory) and ROM (Read Only Memory), sometimes referred to as primary or main memory .
- Unlike a hard drive (secondary memory), this memory is fast and also directly accessible by the CPU.
- RAM is split into partitions (bytes). Each partition consists of an address and its contents (both in binary form).
- The address will uniquely identify every location (byte) in the memory.
- Loading data from permanent memory (secondary storage or hard drive), into the faster and directly accessible temporary memory (RAM), allows the CPU to operate much quicker.

1.2.5 Input Devices

- Which are peripherals used to provide data and control signals to a computer.
- Input devices allow us to enter raw data for processing. For Example:
  - Keyboard (default input device)
  - Microphone
  - Scanner (2D and 3D)
  - Mouse
  - Trackball
  - Touchpad
  - Barcode and QR Code readers
  - Digital Camera

1.2.6 Output Devices

- Which are pieces of computer hardware used to communicate the results of data processing performed by a computer.
- The objective of output devices is to turn computer information into a human friendly/readable form. For Example:
  - Screen (Monitor or Console) (default output device) – LED or LCD
  - Data Projectors
  - Speaker and Headphones
  - Printer (2D and 3D) – inkjet or laser
  - Plotter (wide format printer)
  - Cutter (2D or 3D)

1.2.7 Input/Output Devices

- Some devices work as Input and Output devices like:
  - Touch Screen
  - Network
  - Most of I/O Ports (Both of serial and parallel)
  - New types of ports, like USB and Type-C ports
- All of the secondary storages used as I/O devices for permeant storage, like:
  - Hard disk
  - Floppy disk
  - Flash memory
  - CD and DVD

References

- “Essential Computer Science: A Programmer’s Guide to Foundational Concepts”: Paul D. Crutcher, Neeraj Kumar Singh, Peter Tiegs: Apress, 2021

Course Outlines

Thank You