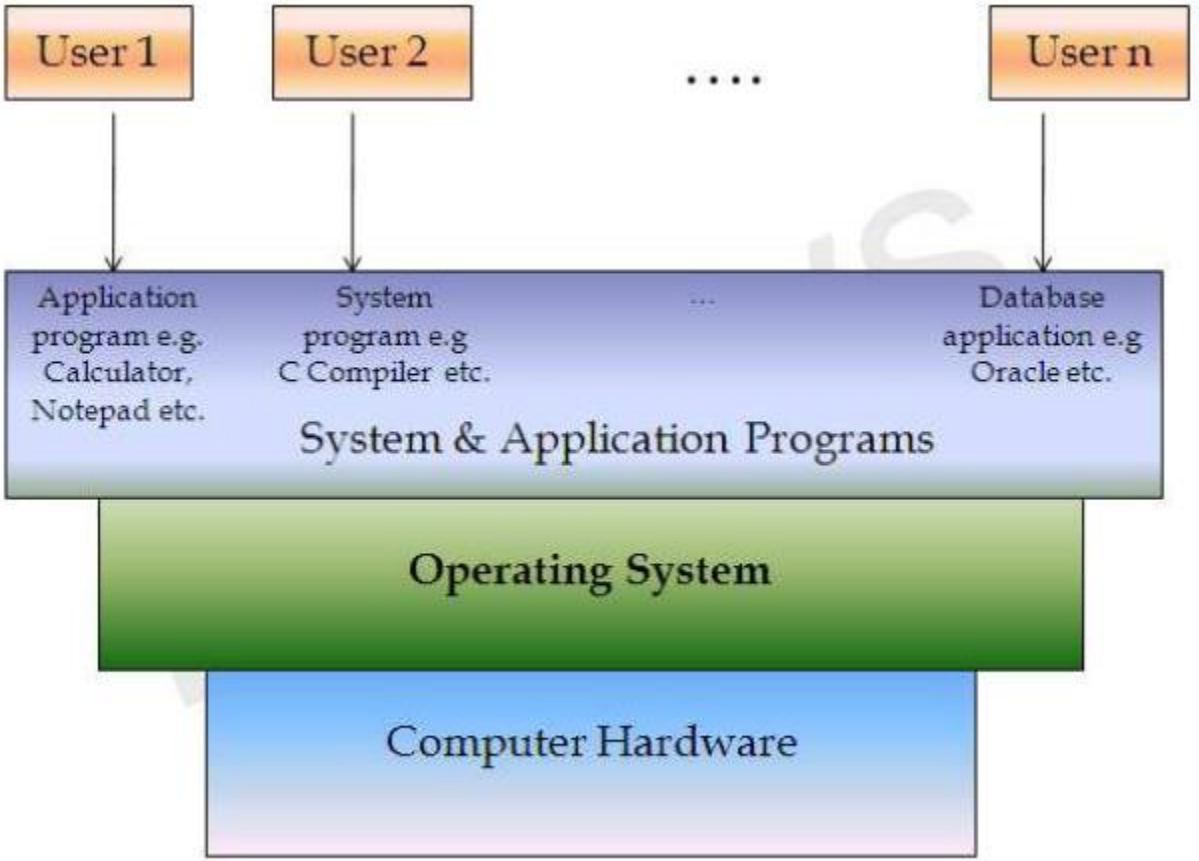
Introduction to Operating System

Topics To Cover

- **INTRODUCTION**
- **PROCESS**
- **SCHEDULING**
- **SYNCHRONIZATION**
- **DEADLOCK**
- **THREA*DS***

Abstract View of System Components



Computer System Structure

- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

Operating System Definition

- Operating system is a system program that provides hardware abstraction, manages system resources and controls the execution of application programs
- Os provides **hardware abstraction**
  - Turn hardware in a form that application can use
- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

A simple program

A Simple Program

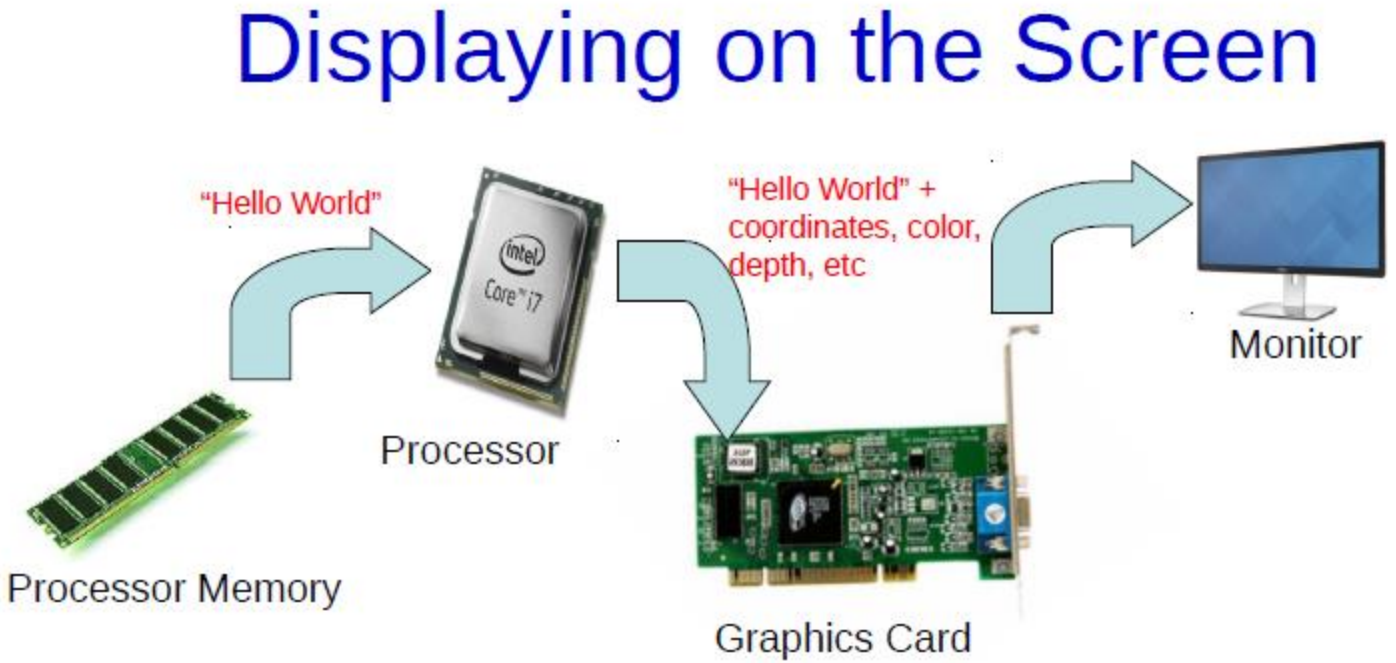
What is the output of the following program?

```
#include <stdio.h>

int main(){
  char str[] = "Hello World\n";
  printf("%s", str);
}
```

How is the string displayed on the screen?

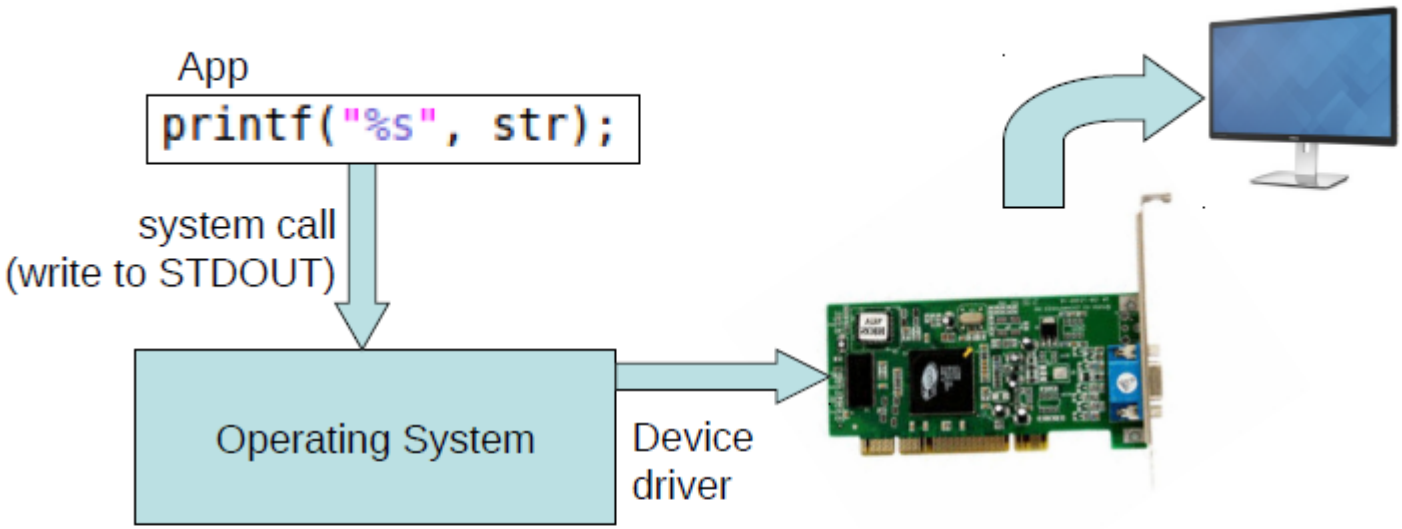
Hardware Abstraction



- Can be complex and tedious
- Hardware dependent

Without an OS, all programs need to take care of every nitty gritty detail

Hardware Abstraction



- **Easy to program** apps
  - No more nitty gritty details for programmers
- **Reusable functionality**
  - Apps can reuse the OS functionality
- **Portable**
  - OS interfaces are consistent. The app does not change when hardware changes

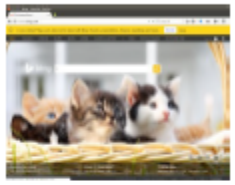
•The **functions** which change the execution mode of the program from user mode to **kernel** mode are known as system calls

•In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on

Os as Resource Manager

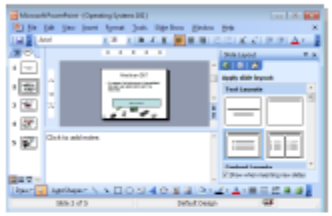
- Multiple apps but limited hardware

Apps



```
#include <stdio.h>

int main(){
  char str[] = "Hello World\n";
  printf("%s", str);
}
```



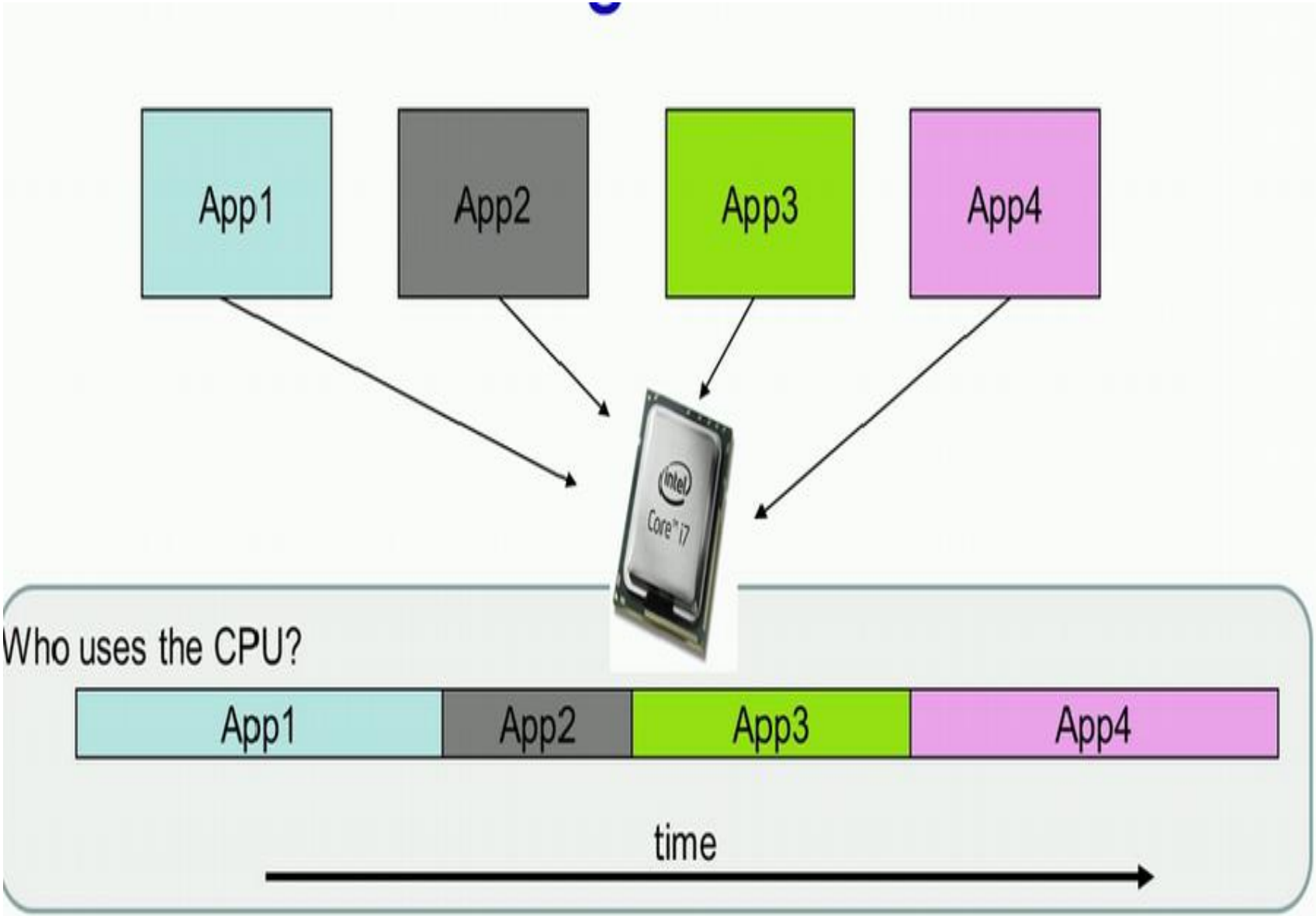
Operating Systems

A few processors

Os as Resource Manager

- Os must manage CPU, memory, file, secondary storage (hard disk), etc.
- Resource Management
  - Allows multiple apps to share resources
  - Protect apps from each other
  - Improves performances by efficient utilization of resources

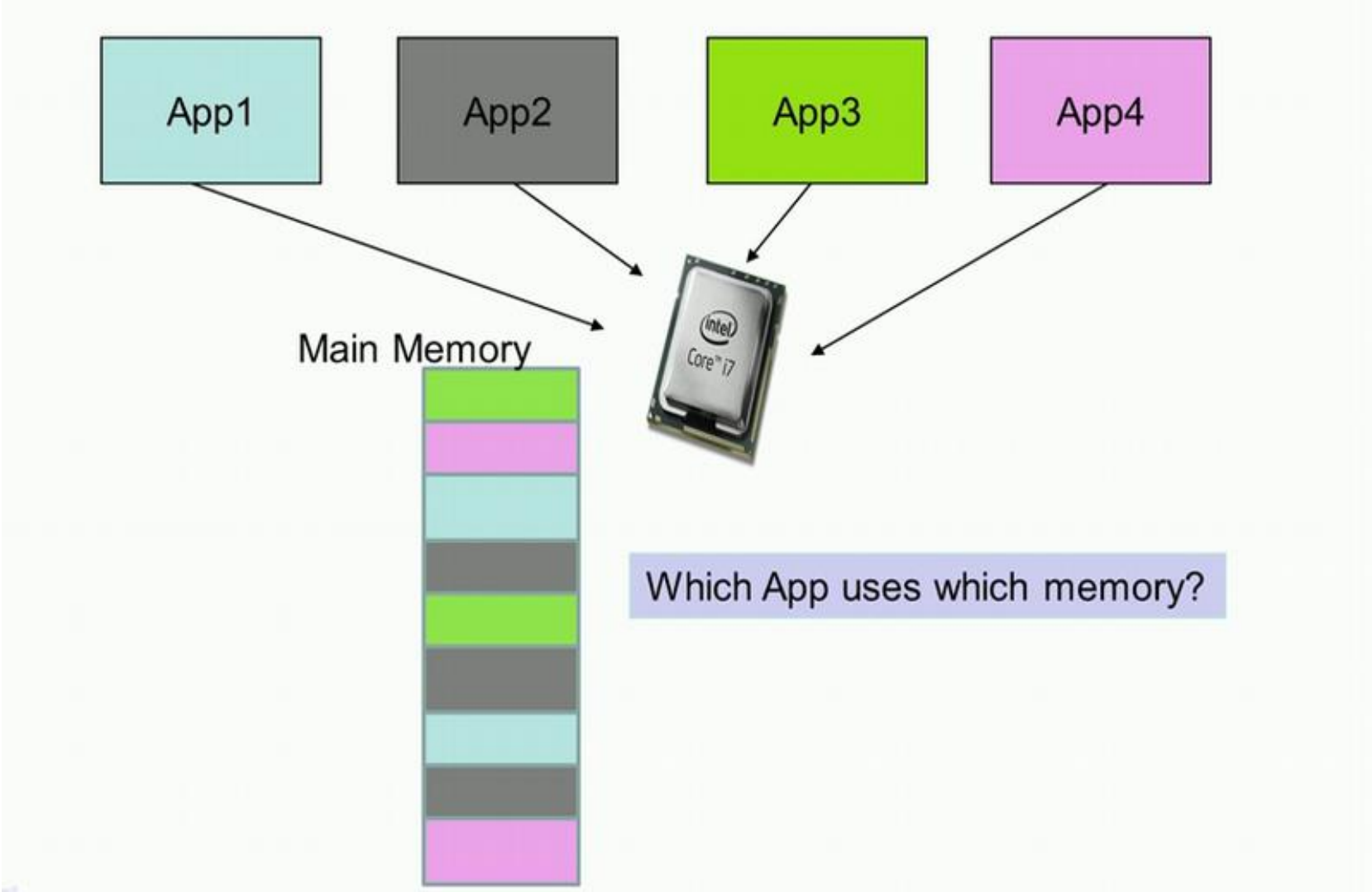
Sharing the CPU



Sharing the CPU

- operating systems around the late 70's and early 80's was to allow one application to execute on the CPU till it completes and then, start the next application.
- Now, this scheduling of the various applications on the processor is managed by the OS.

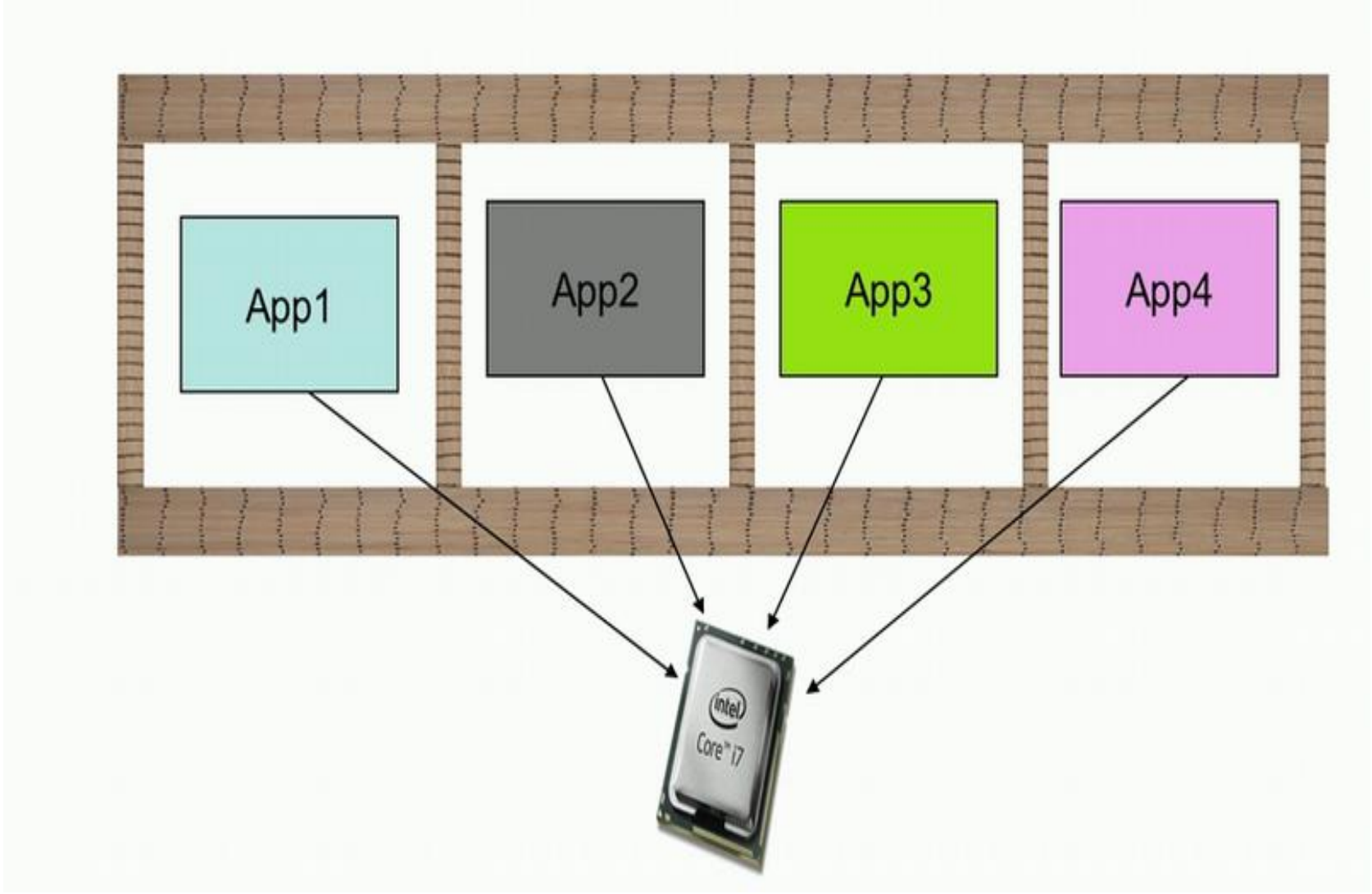
Sharing the Memory



Sharing the Memory

- In order to execute each of these applications needs to be present in the Main Memory that is the RAM of the system. The Operating System needs to ensure how this limited RAM resource is shared among the various applications executing on the system

Sharing but Isolate



Sharing but Isolate

- In spite of sharing the limited hardware resources among the various applications which are executing on the computer system, the sharing should be done in such a way so that applications are isolated from each other.
- Each application the OS should ensure runs in a sand boxed environment that is Application1 should not know anything about Application 2

Operating System Types

- Application Specific
  - Embedded OS
    - eg. Contiki OS, for extremely memory constraint environments
  - Mobile OS
    - Android, iOS, Ubuntu Touch, Windows Touch
  - RTOS
    - QNX, VxWorks, RTLinux
  - Secure Environments
    - SeLinux, SeL4
  - For Servers
    - Redhat, Ubuntu, Windows Server
  - Desktops
    - Mac OS, Windows, Ubuntu

Embedded Operating System

- Embedded operating system designed for memory constraint environments, such as wireless sensor nodes that are used in the internet of things
- Embedded operating system are designed with the power consumption kept in mind. So, these operating systems manage the various resources and also abstract hardware in such a way that the power consumed by the entire system is kept minimum

Mobile Operating System

- Embedded operating systems are designed, so that the battery charge of your mobile phone is extended for the maximum amount of time. So, the mobile OS's like the once we mentioned over here has quite similarities in this aspect with the embedded operating systems like the Contiki OS
- Mobile operating systems unlike the Embedded OS's also need to take care of certain special devices. For instance the monitors or the LCD screens and key pads
- The mobile OS also has user interaction which typically is not in the Embedded OS.

Real Time Operating System

- Several machine critical applications like rockets, automobiles or nuclear plants, a delay in doing a particular operation by the computer system would be catastrophic.
- A real-time operating system (RTOS) is the one which deals with real-time application requests
- The input data must be processed without any delays.
- The processing time is shorter and the waiting time is minimum
- RTOS's are designed in such a way that every critical operation on the system is guaranteed to complete within the specified time.
- If deadline has to be met deterministically (hard deadline), it is a hard RTOS.
- If an RTOS meets a deadline approximately, it is known as a soft RTOS

Real Time Operating System

- The events that an RTOS may have to respond to can be either periodic (occurring at regular intervals) or aperiodic (occurring at irregular intervals). Further, if there are n periodic events and an event i occurs with period P_i and requires T_i seconds of CPU time, then the load can be handled if

- An RTOS that meets this criteria is said to be schedulable.

$$\sum_{i=1}^{n} (Ti / Pi) <= 1$$

Hard Real Time System

- **Hard real time system** guarantees that critical tasks complete on time. This type of system can never miss its deadline. Missing the deadline may have disastrous consequences. Example: Flight controller system
- In a hard real time system if a deadline for a task is not met we will say that the system has failed, once an event occurs the required response could not be obtained before the stipulated time then we will say that the system has failed.
- The time restrictions that we give to the task are the range of microseconds or milliseconds.

Soft Real Time System

- **Soft real time system**, a critical real time task gets priority over other tasks and retains that priority until it completes. This type of system can miss its deadline occasionally with some acceptably low probability. Missing the deadline have no disastrous consequences.
- On the other hand a soft real time system if there is a deadline missed the system does not fail and the only thing is that it just the result becomes less valuable and if a large number of deadlines fail then we say that the system performance has degraded.
- The examples are railway reservation system, web browsing and in fact, all interactive applications where we expect the result to be produced within few seconds 20 seconds, 30 second and then we say that the system is performing well. If it takes minutes several minutes and so on then we say that the performance has degraded.

Secure Environment OS

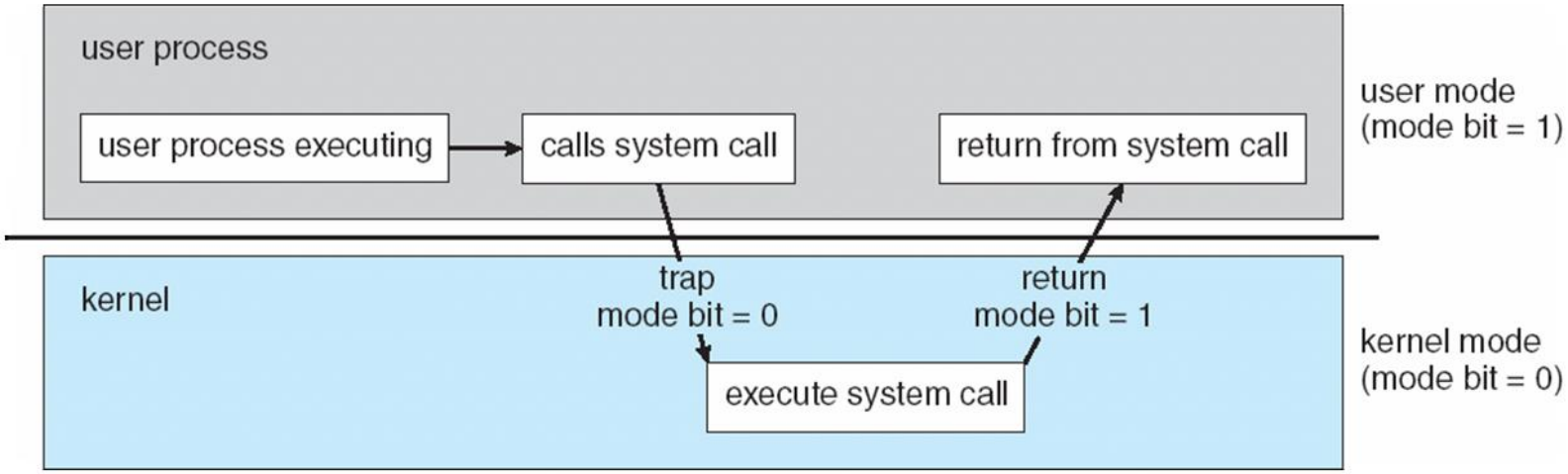
- These operating systems are used in providing Secure Environments. These operating systems are especially utilized for applications where security is extremely critical. So, these for example could be for web servers that host banking software

Operating System Operations are Event Driven

operating systems are event driven that is whenever an event occurs only then the operating system executes.

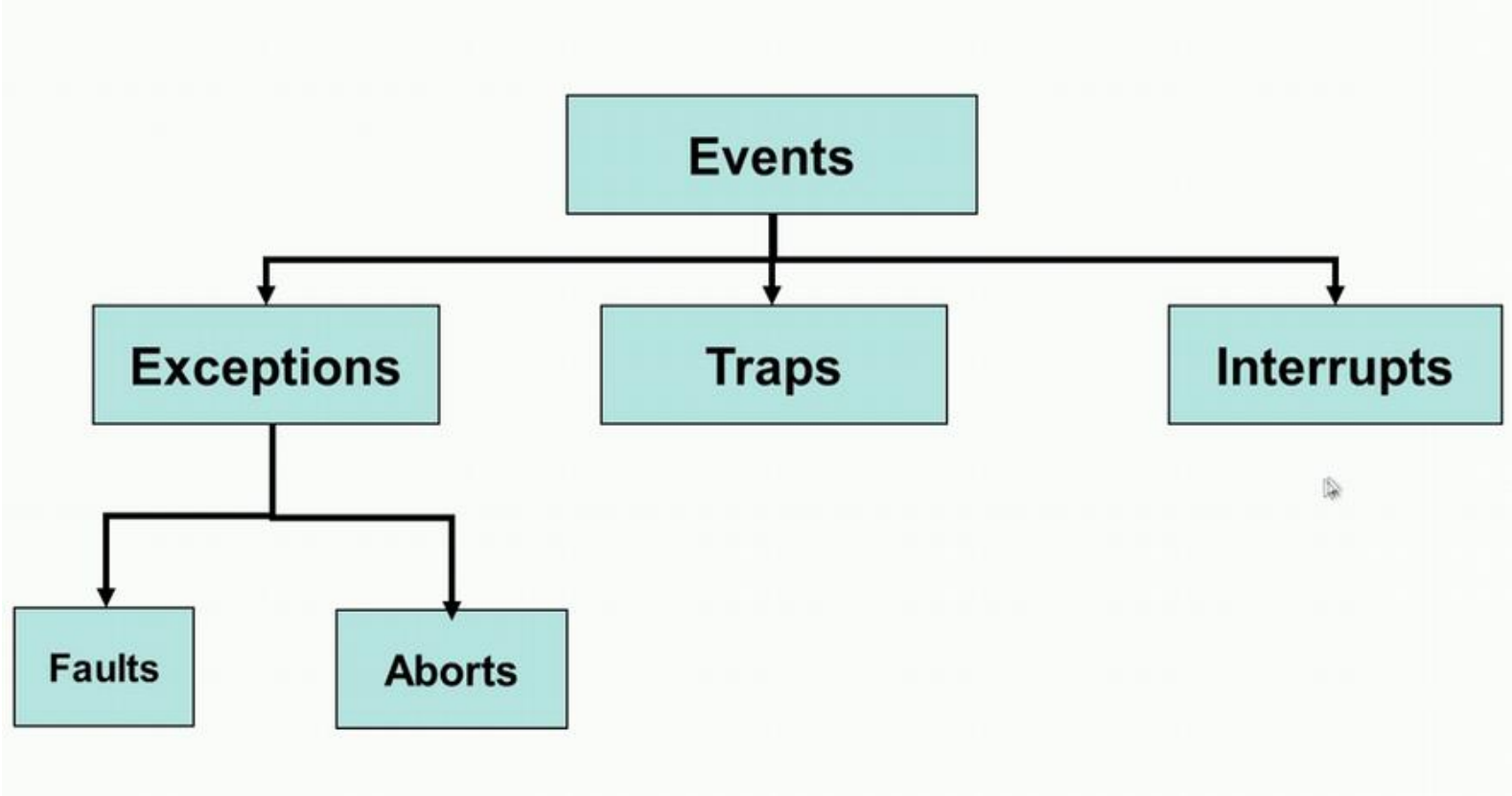
✓User process continues to execute on the processor until an event occurs
✓When the event occurs it would trigger the operating system to execute
✓This triggering of the operating system will also result in a change in the privilege level.
✓The system would no longer be executing in the user space, but rather it will be in privilege level 0 or that is executing in the Kernel space
✓The operating system would essentially execute and service this particular event and at the end of that execution, the control is fed back to user space and the process will continue to execute

Transition from User to Kernel Mode



Event Types

Events are events are classified into 3 different types; these are ***Hardware Interrupts***, ***Traps*** *and* ***Exceptions***



What is a Hardware interrupt?

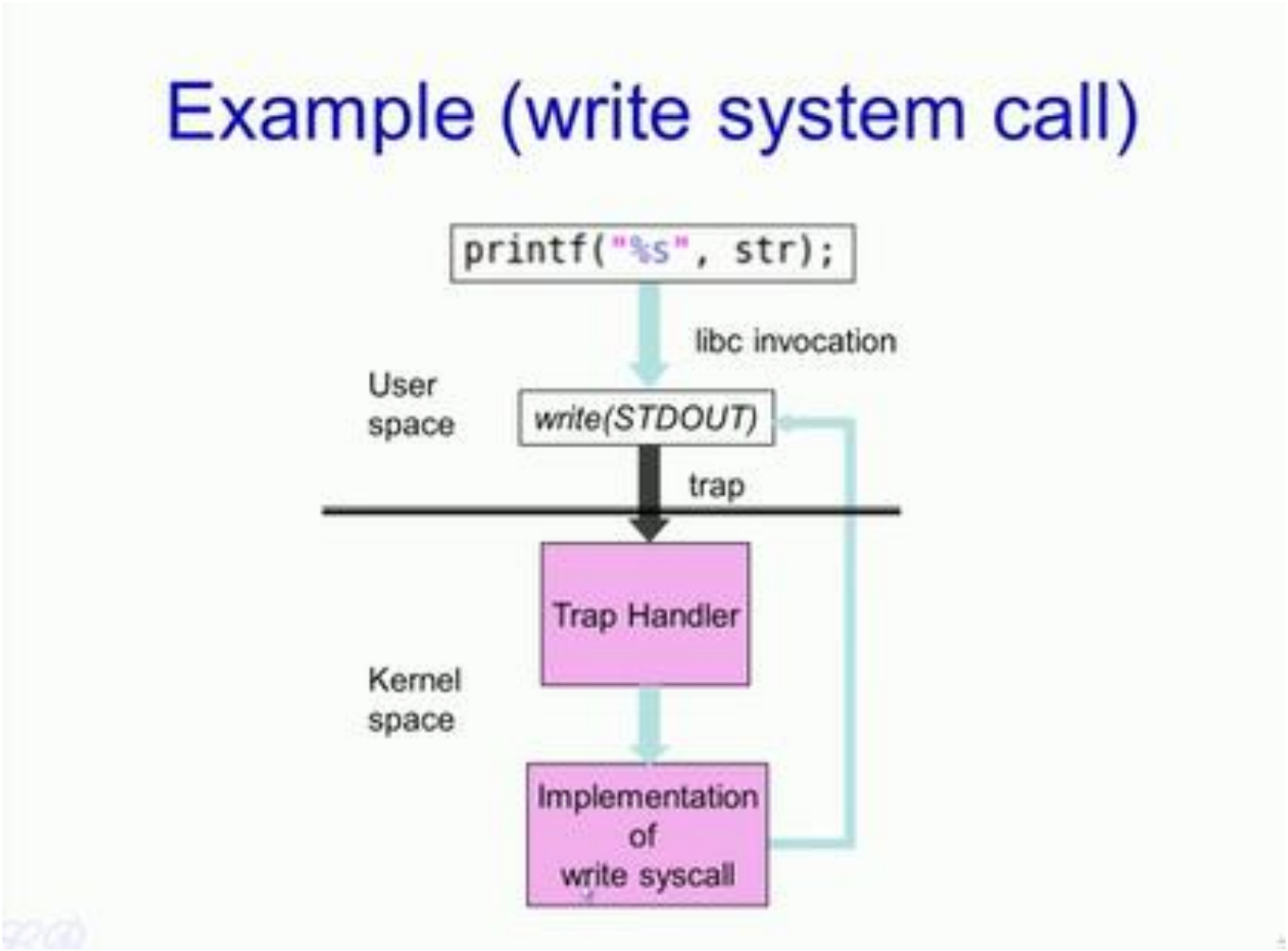
- In system programming, an interrupt is a signal to the processor emitted by hardware or software indicating an event that needs immediate attention.
- Interrupt alerts the processor to a high-priority condition requiring the interruption of the current code the processor is executing.
- For example, pressing a key on the keyboard. The network card when it receives a packet could possibly raise an interrupt or a USB device when plugged in could raise in hardware interrupt.
- When in an executing program an event occurs (a key is pressed) an interrupt generated to the processor and that would result in a switch in the processor to what is known as the Interrupt Handler Routine
- Next, context is switched back to the program which was originally being run

Trap /Software interrupt

- Traps are sometimes known as software interrupts, they are raised by user programs in order to invoke some operating system functionality

- printf is a function present in the library libc and it would cause the libc function to be invoked. Now in the libc function there is a call to the write system call with the specifier STDOUT i.e write(STDOUT).

How system call works

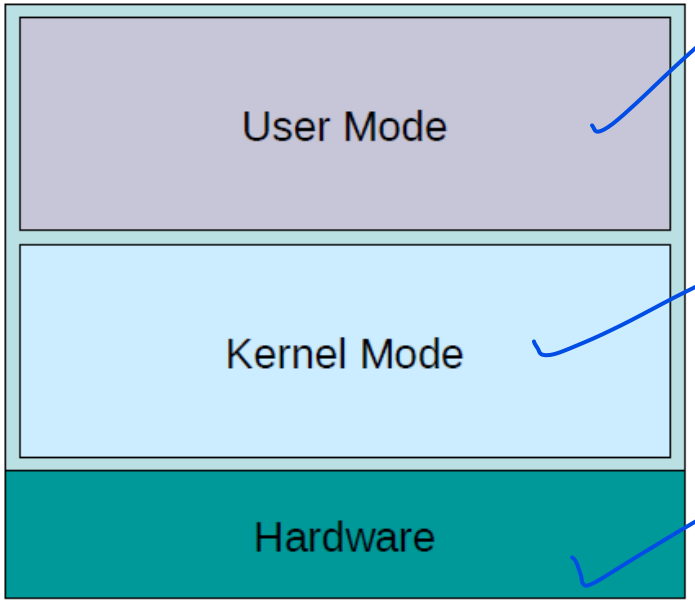


Exceptions

- These events are generated automatically by the processor itself as a result of an illegal instruction
- There are 2 types of exceptions these are *Faults and Aborts*
- Faults are essentially exceptions from which the processor could recover
- When a divide by 0 exception occurs in a program, typically the program would be terminated. Essentially the operating system has no way to recover from such exception.

Operating Modes

- User Mode
  - Where processes run
  - Restricted access to resources
  - Restricted capabilities
- Kernel mode a.k.a. Privileged mode
  - Where the OS runs
  - Privileged (can do anything)



Communicating with the OS (System Calls)

- System call invokes a function in the kernel using a Trap
- This causes
  - Processor to shift from user mode to privileged mode kernel mode
- On completion of the system call, the execution gets transferred back to the user mode process