CMPT 300 D100
Operating System I
1.2 OS Introduction

Dr. Hazra Imran

Spring 2022

Learning goals

- Describe the general organization of a computer system
- Describe the role of interrupts.

Computer System Structure

processes

Computer system can be divided into four components:

job

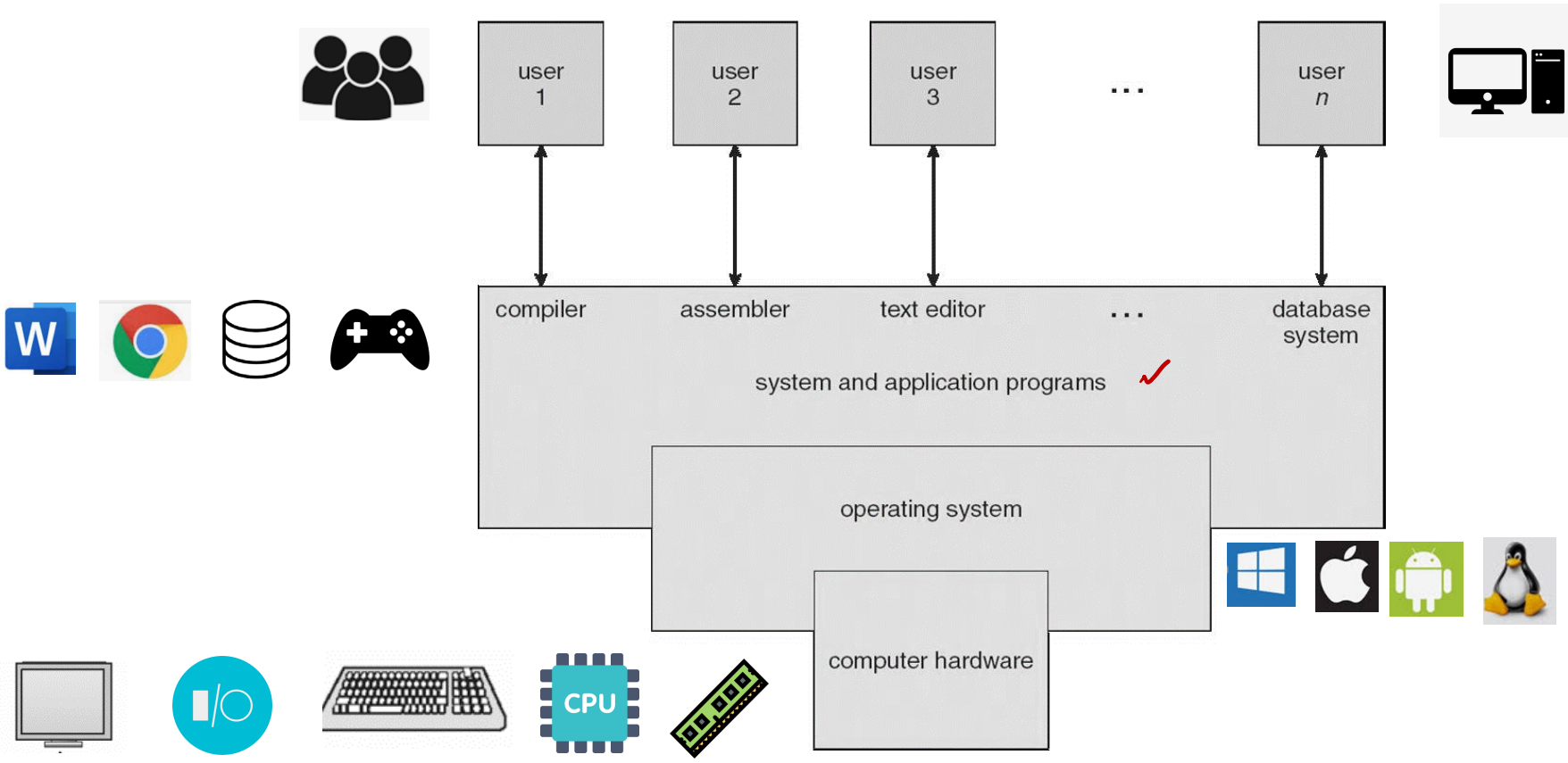
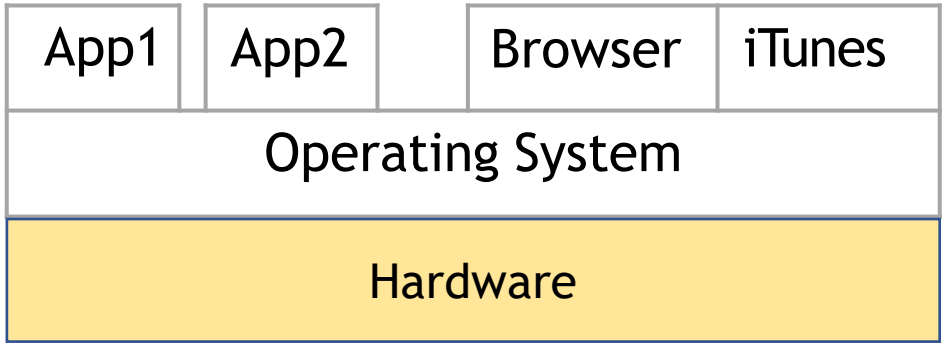


Image Credits: Google

Why Start With Hardware?



- OS functionality depends upon the architectural features of H/W

enforce protection

resource sharing

- Architectural support can greatly simplify or complicate OS tasks

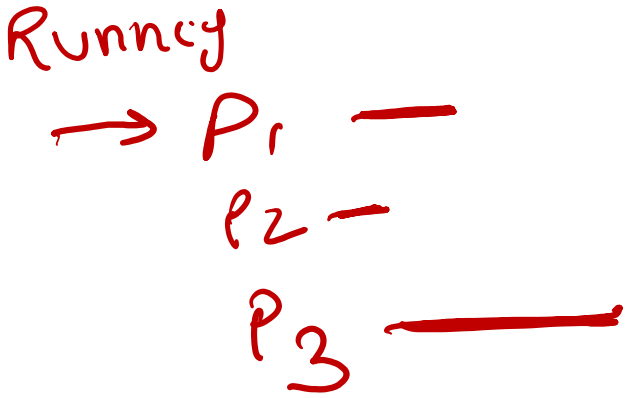
Role of Operating System

- OS is a **resource allocator**
  - Manages all hardware resources
  - Decides between conflicting requests for fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer

- OS provides **abstractions**
  - Hides the details of the hardware
  - Provides an interface that allows a consistent experience for application programs and users

Example of abstractions

- Allows user programs to deal with simpler, high-level concepts
- Hides complex and unreliable hardware
- Provides illusions like “only one application running” or “infinite memory available”



What if we read directly from the files?

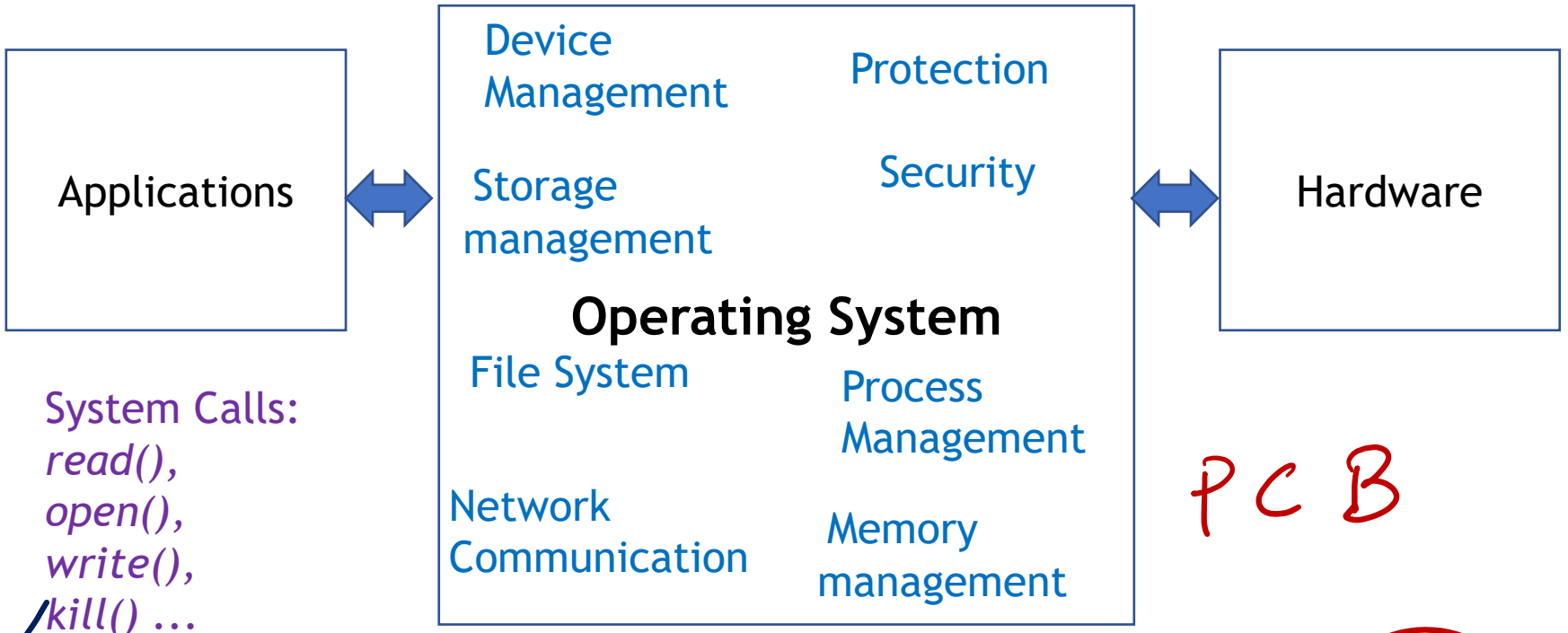
- Where is this file on disk? Which track and sectors?
- Code needs to change on a different system

Abstraction ….

```
file = open ("test.txt")
...
close (file);
```

detail / commands.

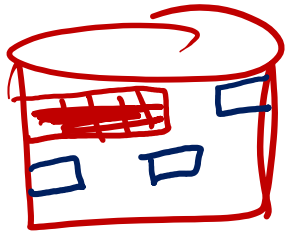
Example of abstraction



System Calls:
read(),
open(),
write(),
kill() ...

PCB

P1
P2



Try out Activity - 1

load store add

Hardware component	Hardware capabilities	User needs
CPU	Machine instructions - perform operations on contents of registers and memory locations.	User thinks in term of • arrays high level • list ds
Main memory	Physical memory is a linear sequence of addressable bytes or words that hold programs and data.	• var - heap • executables lib
Secondary storage	Are multi-dimensional structures, which require complex sequences of low-level operations to store and access data organized in discrete blocks.	programs to manipulate / access data
I/O Devices	are operated by reading and writing registers of the device controllers.	Interfaces

Instruction Cycle

The CPU repetitively performs the instruction cycle:

Fetch

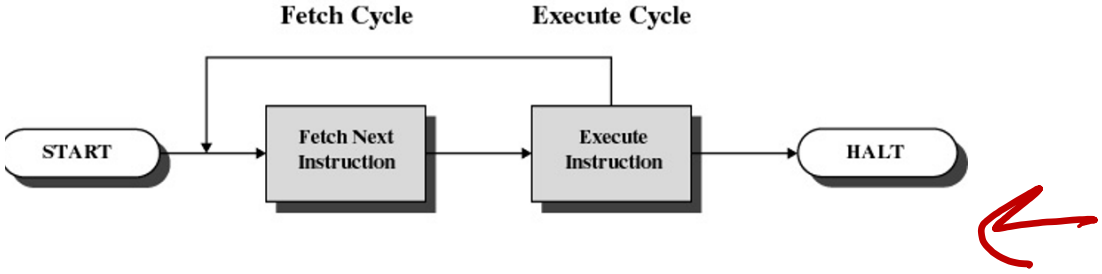
- The program counter (PC) holds the address in memory of the next instruction to execute
- The address from memory is fetched and stored in the IR
- The PC is incremented to fetch the next instruction (unless told otherwise)

Decode

- The CPU determines what instruction is in the IR

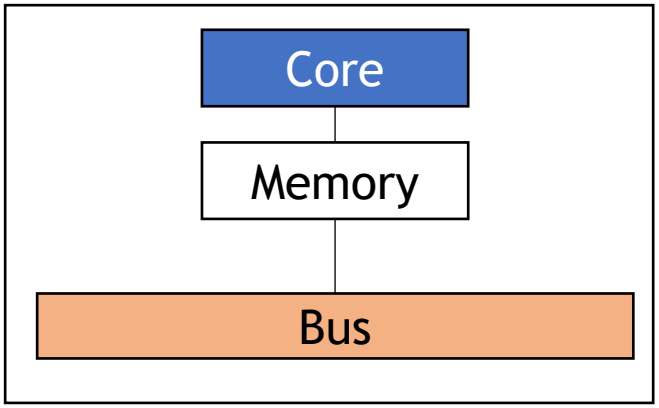
Execute

- Circuitry interprets the opcode and executes the instruction
- Moving data, performing an operation in the ALU, etc.
- May need to fetch operands from memory or store data back to memory

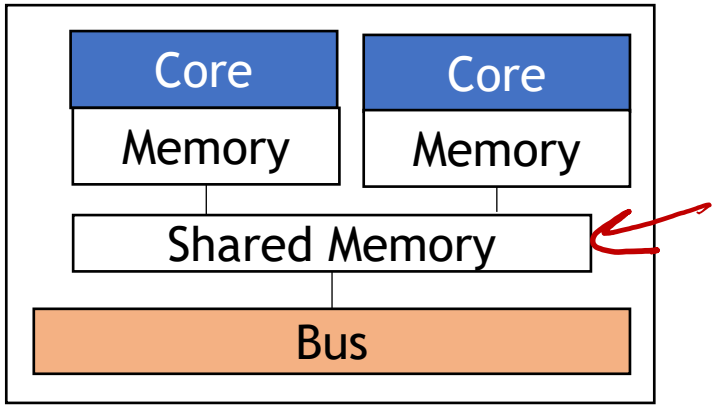


Single-core processor vs Multiple-core processor

Single-Core Processor



Multi-Core Processor

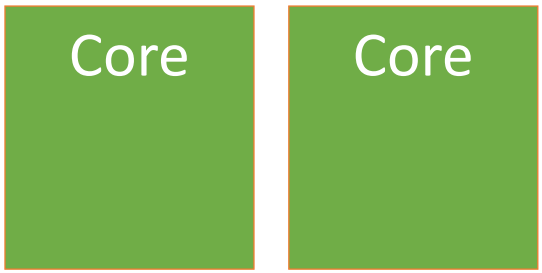


- ☹ High power consumption
- ☹ Heat generation
- ☹ 75% of CPU time is used waiting for memory access results.
- ☹ Limited parallelism

Massive parallelism offered by

Hyper threading

- Each core may execute more than one thread
  - Hyper-threading
  - Share certain physical resources (e.g., caches, registers)

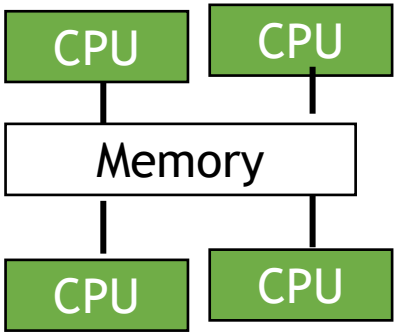


Multi-core and multi-socket

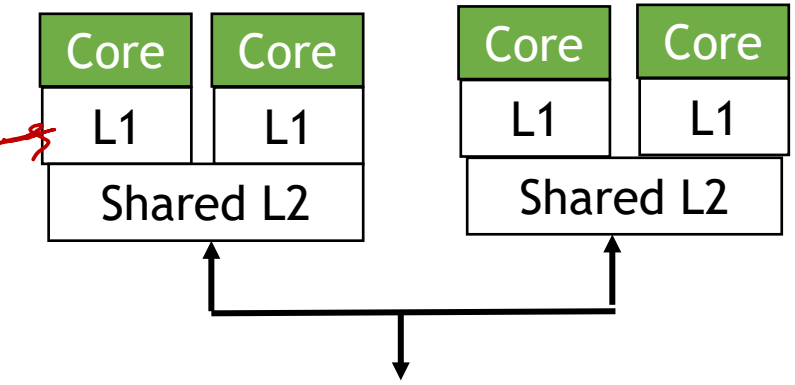
- Socket to place a multicore processor
- Usually 4-32 cores per processor/socket

Shared memory systems

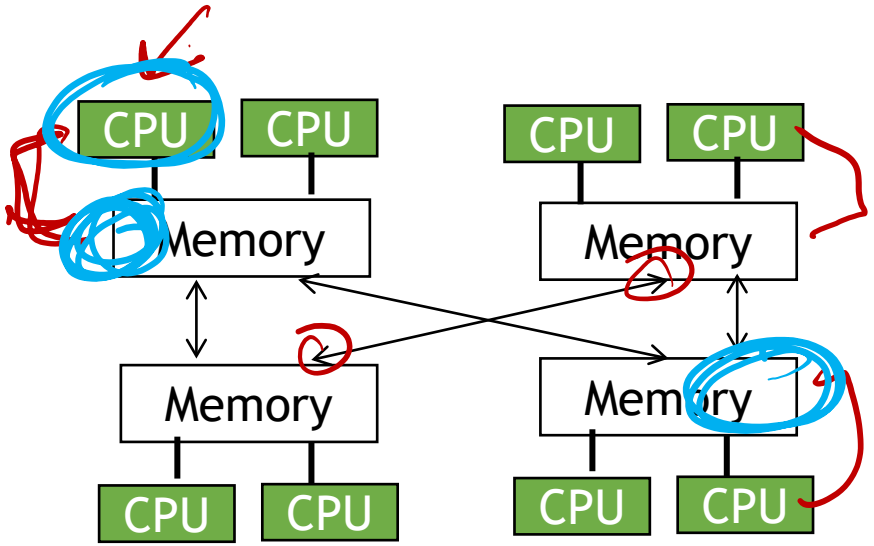
Centralized Shared Memory Architecture
aka.. Symmetric Multi-Processors (SMP)



Memory bandwidth per processor is limited
Typical size: 2-32 processors



Distributed Shared Memory Architecture
aka.. Non-Uniform Memory Architectures (NUMA)



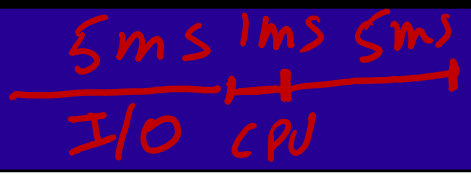
Memory local to a core is faster to access than non-local memory

Some useful system monitoring tools

- **htop -** highlights the processes and information using some color
- nmon - lets you monitor system resources and processes.
- CoreFreq - a CPU monitoring program

- https://linoxide.com/monitoring-2/10-tools-monitor-cpu-performance-usage-linux-command-line/

Computer System Operation



- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device type
- Each device controller has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an **interrupt**

ISR

Interrupts

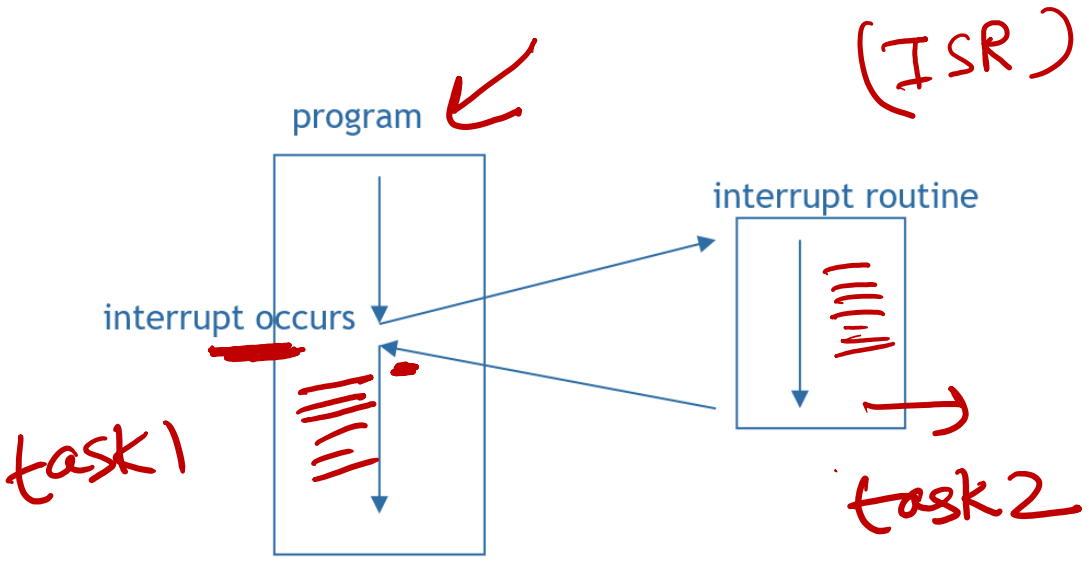
- An interrupt is a signal that causes the computer to alter its normal flow of instruction execution.
- Allows the processor to execute other instructions while an I/O operation is in progress
- Improves processing efficiency
- A suspension of a process caused by an event external to that process and performed in such a way that the process can be resumed

Interrupt Handler

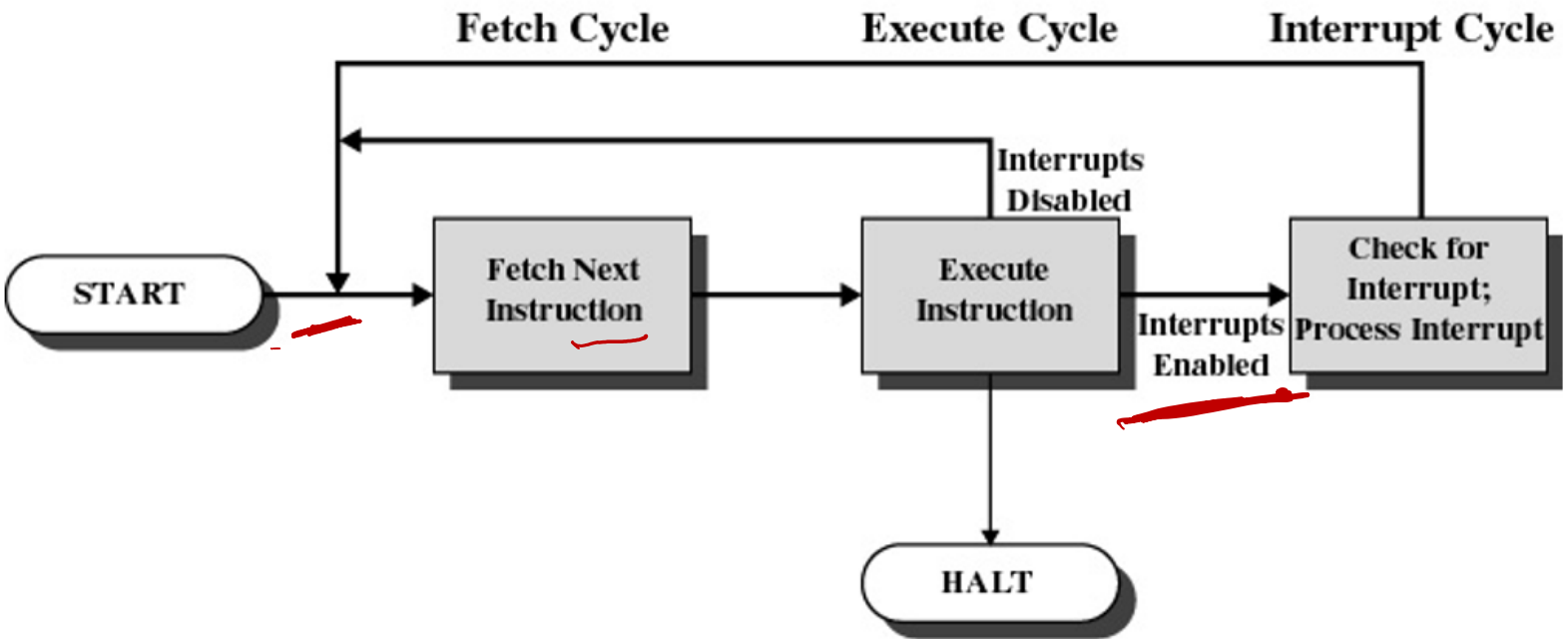
- A program that determines nature of the interrupt and performs whatever actions are needed
- Control is transferred to this program
- part of the operating system

Classes of Interrupts

- Program
  - arithmetic overflow
  - division by zero
  - execute illegal instruction
  - reference outside user's memory space
- Timer
- I/O
- Hardware failure



Interrupt Cycle



- Processor checks for interrupts
- If no interrupts fetch the next instruction for the current program
- If an interrupt is pending, suspend execution of the current program, and execute the **interrupt handler**

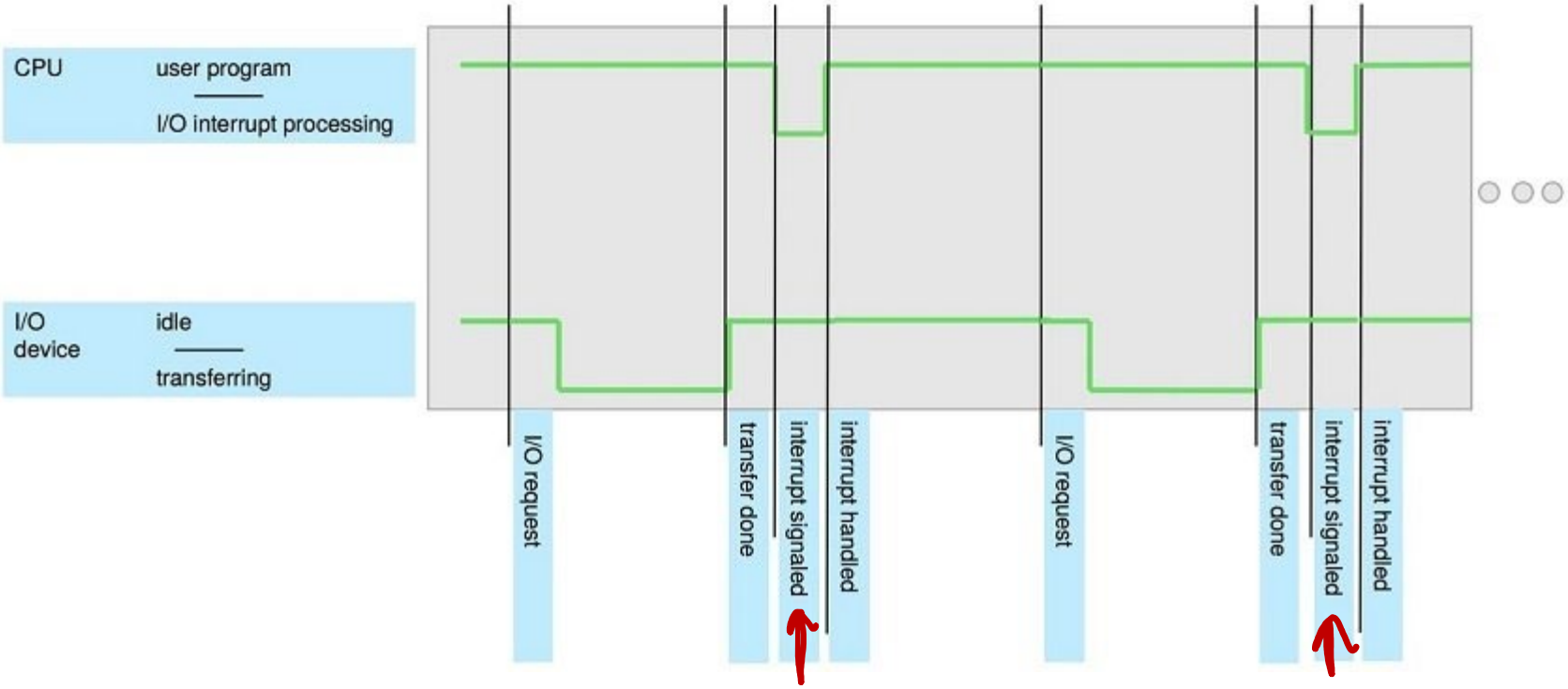
Interrupt Handling

- The operating system preserves the state of the CPU by storing registers and the program counter
- Determines which type of interrupt has occurred:
  - **polling**
  - **vectored** interrupt system
- Separate segments of code determine what action should be taken for each type of interrupt

" Polling is like opening the door every few seconds to see if guests have arrived…"

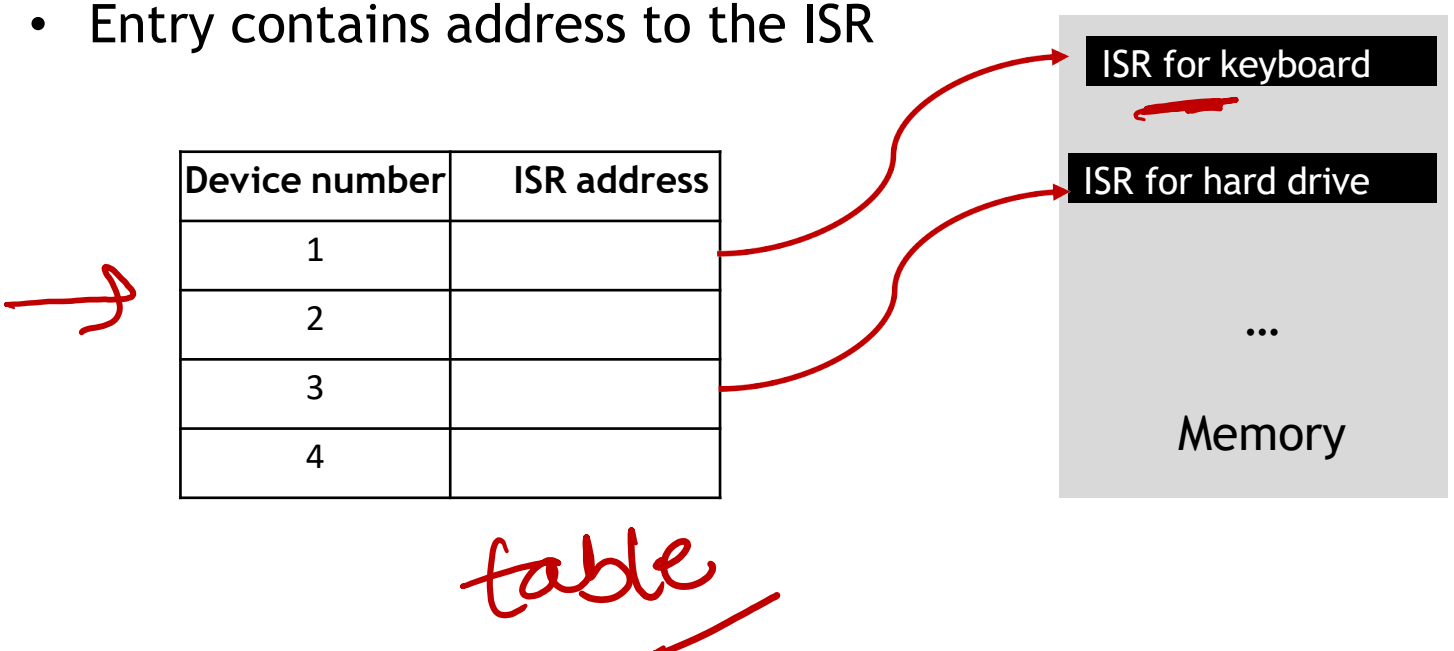
Interrupts are like waiting for the door bell to ring.

Interrupt Timeline



How to Find the "Corresponding ISR"?

- Interrupts must be handled very quickly
- The system can support multiple interrupts
- Need multiple different ISRs
- Interrupt vector
- Table indexed by device number
- Entry contains address to the ISR

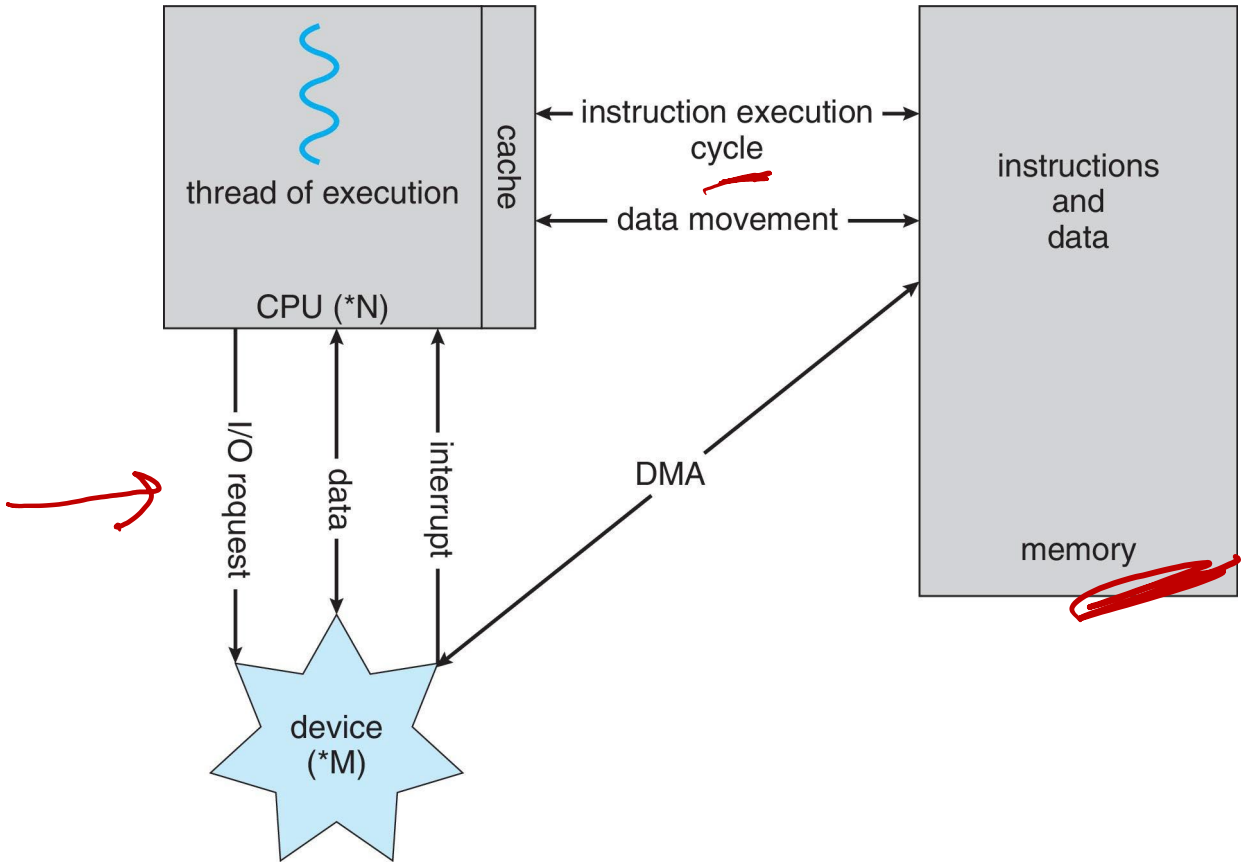


I/O Structure

- After I/O starts, control returns to user program only upon I/O completion
  - Wait instruction idles the CPU until the next interrupt
  - Wait loop (contention for memory access)
  - At most one I/O request is outstanding at a time, no simultaneous I/O processing

- After I/O starts, control returns to user program without waiting for I/O completion
  - **System call** – request to the OS to allow user to wait for I/O completion
  - **Device-status table** contains entry for each I/O device indicating its type, address, and state
  - OS indexes into I/O device table to determine device status and to modify table entry to include interrupt

Summary: How Modern Computer Works



Try out activity

Program 1 and 2 run concurrently. Whenever a timeout interrupt occurs, the kernel switches control between the programs. Show the order of instruction execution, assuming program 1 is currently running.

Application 1	Application 2
... instruction i (timeout interrupt) instruction i+1 ... instruction k (timeout interrupt) instruction k+1 ...	instruction 0 ... instruction j (timeout interrupt) instruction j+1 ...