# Course Introduction: Operating Systems

Mainack Mondal and Saptarshi Ghosh

CS30002

Spring 2021-22

# Today’s class

- Resources and notes
- Introduction to operating systems

# Instructors

- Mainack Mondal
  - Research interests: system security, large scale system measurement, privacy
  - Office: CSE 316
  - Will also be in your lab

# Instructors

- Saptarshi Ghosh
  - Research interests: Social Networks, ML, NLP, IR
  - Office: CSE 207
  - Will also be in your lab

# Teaching assistants

- Soham Poddar (PhD)
- Abhishek Mukherjee (M.Tech.)
- Kirti Agarwal (M.Tech.)
- Adhikansh Singh (Dual)
- Ankit Bagde (Dual)
- M Kousshik Raj (Dual)
- Praagy Rastogi (Dual)
- Shrey Shrivastava  (Dual)
- Shivam Jha  (Dual)
- Udit Dharmin Desai  (Dual)
- Sanket Meshram  (Dual)

# Requirements

- Using computers and application software

# Requirements

- Using computers and application software

- How to write, compile and run C-programs

# Requirements

- Using computers and application software
- How to write, compile and run C-programs
- Computer Organization and architecture (prerequisite)

# Website / Books

- Website:
  - https://cse.iitkgp.ac.in/~mainack/courses/2021-spring/OS-course/index.html
- Textbooks / References:
  - Operating Systems Concepts, 9th ed. - A. Silverschatz, P.V. Galvin, and G. Gagne. Wiley
  - Some recent topics from research papers

# Course logistics

- There will be 2 or 3 exams and some assignments
    - Details and mode will be announced later

# Why should you attend class?

- Aside from the fact that slides are not complete and exam questions will come from class coverage?

# Why should you attend class?

- Aside from the fact that slides are not complete and exam questions will come from class coverage?
- Doubt clearing sessions

# Last but not the least

- Ask questions in the class
  - You need to know how Operating Systems (OS) as computer scientist
  - It is best done via class - interaction

# Last but not the least

- Ask questions in the class
  - You need to know how Operating Systems (OS) as computer scientist
  - It is best done via class - interaction

- Do lab assignments & practice problems religiously
  - This is a systems course
  - There is no other way of learning how systems work than actually trying to build systems

# Today’s class

- Resources and notes

- Introduction to operating systems

# Today's class

- Resources and notes

- Introduction to operating systems (OS)
  - What is an OS
  - What are the goals of an OS
  - Under the hood: the structure of OS
  - How does OS work?

# What is an Operating System?

# Let’s start with enumerating the actors

# Let’s start with enumerating the actors

Users

- might be one or multiple (using servers)

# Let’s start with enumerating the actors

Users

- might be one or multiple (using servers)

Logs in and start applications

- gcc, gedit, notepad, firefox, chrome, excel, …

# Let’s start with enumerating the actors

Users

- might be one or multiple (using servers)

Logs in and start applications

- gcc, gedit, notepad, firefox, chrome, excel, …

These applications ultimately need to use hardware

- processor, ram, display, mouse, keyboard ….

# Let’s start with enumerating the actors

Users

- might be one or multiple (using servers)

Logs in and start applications

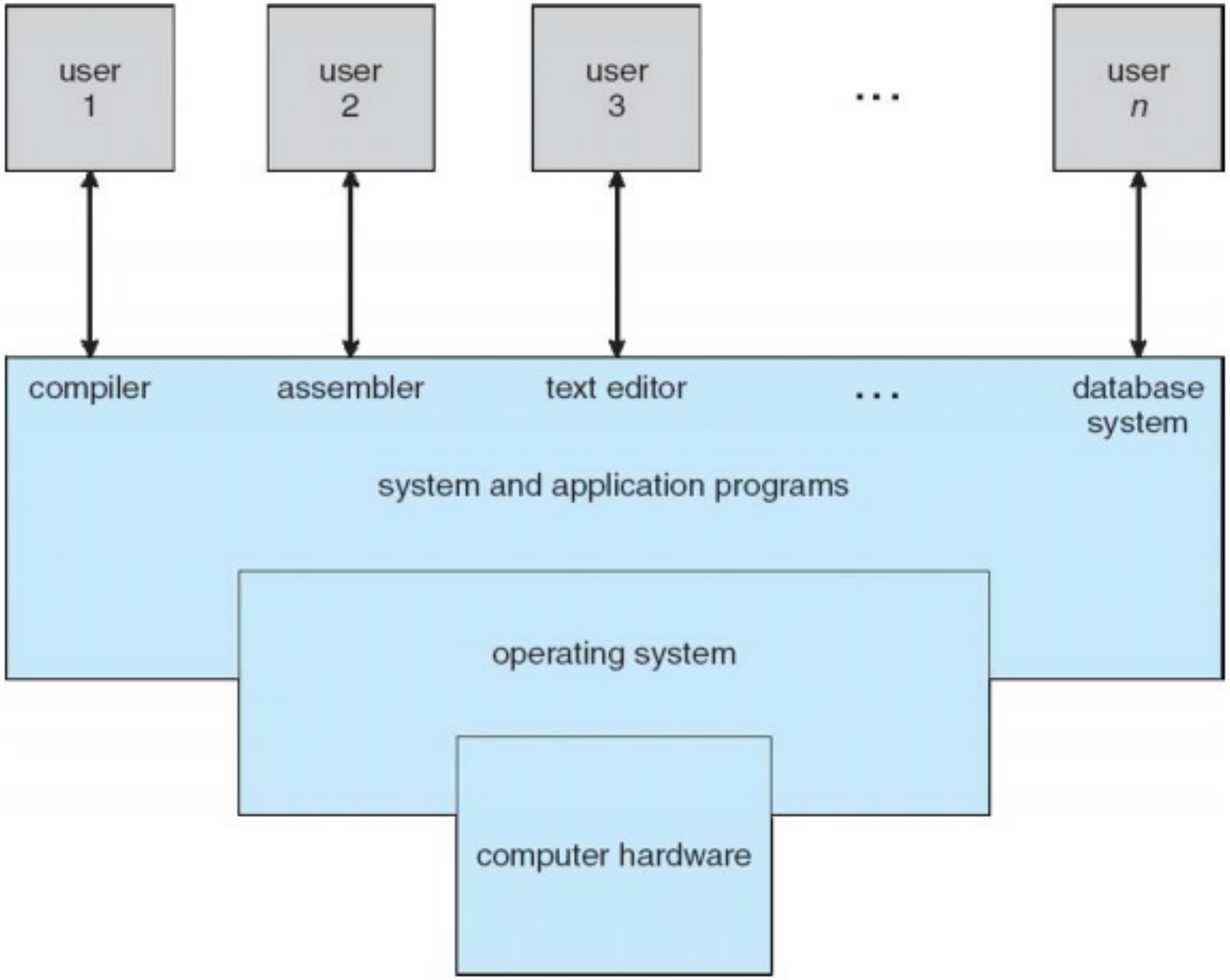
- gcc, gedit, notepad, firefox, chrome, excel, …

These applications ultimately need to use hardware

- processor, ram, display, mouse, keyboard ….

Operating system seats right in between applications and hardware

# Putting it all together



Source: Silberschatz, Galvin and Gagne ©2013

# OS: Definition

A program that acts as an intermediary between users of a computer system and hardware

# Today’s class

- Resources and notes

- Introduction to operating systems (OS)
  - What is an OS
  - What are the goals of an OS
  - Under the hood: the structure of OS
  - How does OS work?

# Goals of an Operating System

# Primary goals

- Make the computer system convenient and easy to use (for the users)

- Ensure efficient utilization of resources (processor time, printer, RAM, ...)
    - Controls and co-ordinates the use of hardware resources among multiple users and applications

# These goals might be conflicting across computing systems

# These goals might be conflicting across computing systems

Mini computers and mainframes (1970 - 80)

Best utilization of resources

Cost: user experience (often no immediate result)

# These goals might be conflicting across computing systems

| Mini computers and mainframes (1970 - 80) | Workstations / terminals (1980 - now) |
|---|---|
| Best utilization of resources | Fast response time |
| Cost: user experience (often no immediate result) | Cost: Resource utilization is suboptimal |

# These goals might be conflicting across computing systems

| Mini computers and mainframes (1970 - 80) | Workstations / terminals (1980 - now) | Laptops |
|---|---|---|
| Best utilization of resources | Fast response time | Flexible UI, easy to use |
| Cost: user experience (often no immediate result) | Cost: Resource utilization is suboptimal | Cost: single user system |

# These goals might be conflicting across computing systems

| Mini computers and mainframes (1970 - 80) | Workstations / terminals (1980 - now) | Laptops | Mobile systems |
|---|---|---|---|
| Best utilization of resources | Fast response time | Flexible UI, easy to use | Optimized usability & battery life |
| Cost: user experience (often no immediate result) | Cost: Resource utilization is suboptimal | Cost: single user system | Cost: Severely constrained resources (your android cannot handle running full fledged game and browsing) |

# These goals might be conflicting across applications

Application A: I need 3 GB of memory, now!

# These goals might be conflicting across applications

Application A: I need 3 GB of memory, now!

Application B: I need 2 GB of memory, now!

# These goals might be conflicting across applications

Application A: I need 3 GB of memory, now!

Application B: I need 2 GB of memory, now!

Hardware: oops, total RAM is 4 GB

# OS: goal-oriented definition

- A resource allocator
  - Manages all resources (processor time, RAM, display, ...) to ensure they are shared in an efficient and fair manner
  - Decides which application gets how much resources and when

- A control program
  - Controls execution of other programs / applications to prevent errors and improve usability (e.g., by giving faster response to users)
  - A faulty application should not disrupt other applications (or the OS itself)

# OS: goal-oriented definition (contd.)

- Enabler of communication / coordination
  - One application may need to communicate to others, or share data / state / ...
- Enabler of easier development of applications
  - Offers a set of common services for applications (e.g., for I/O) - application developer does not need to worry about specifics of devices
  - Gives an illusion of infinite resources – dedicated processor, infinite memory, …

# Today’s class

- Resources and notes

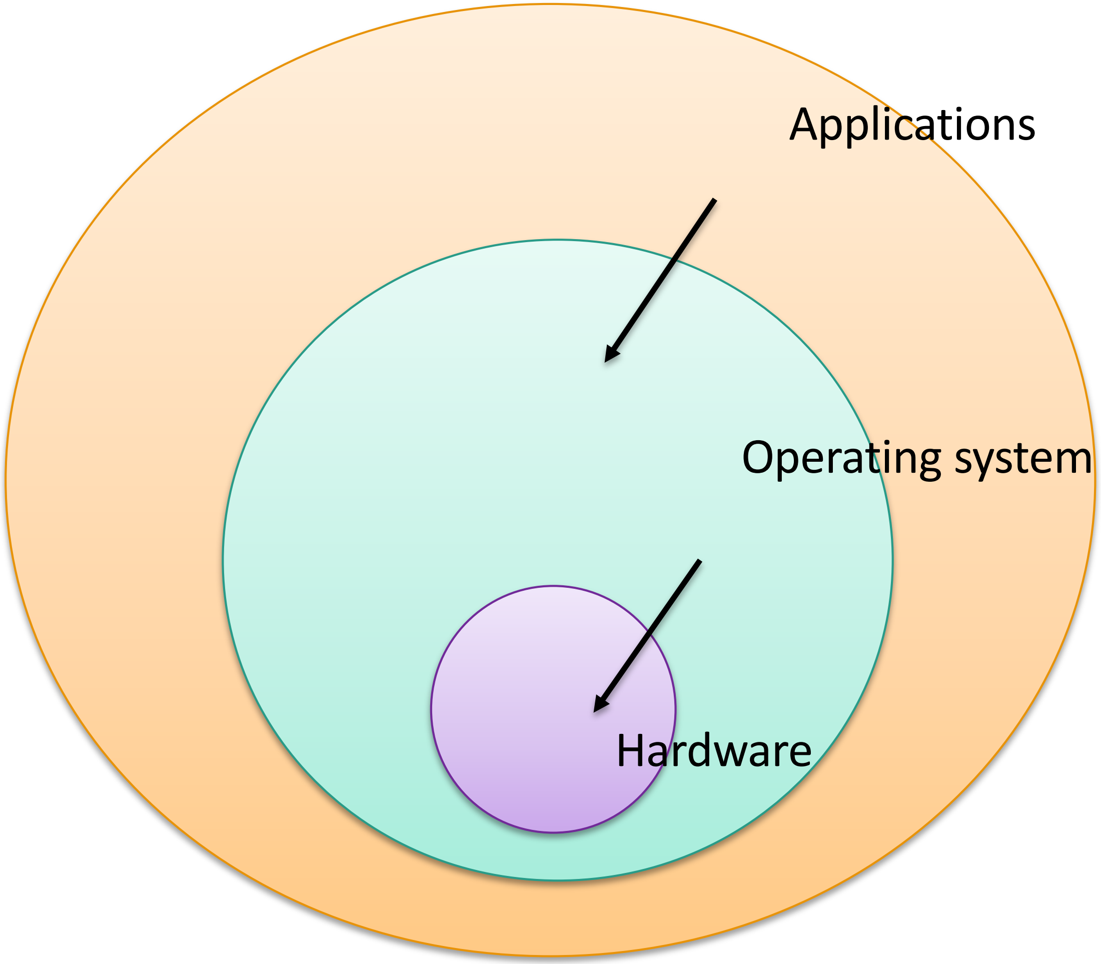
- Introduction to operating systems (OS)
    - What is an OS
    - What are the goals of an OS
    - Under the hood: the structure of OS
    - How does OS work?

# Typical structure of OS (unix)

- Note: By structure we simply meant control flow

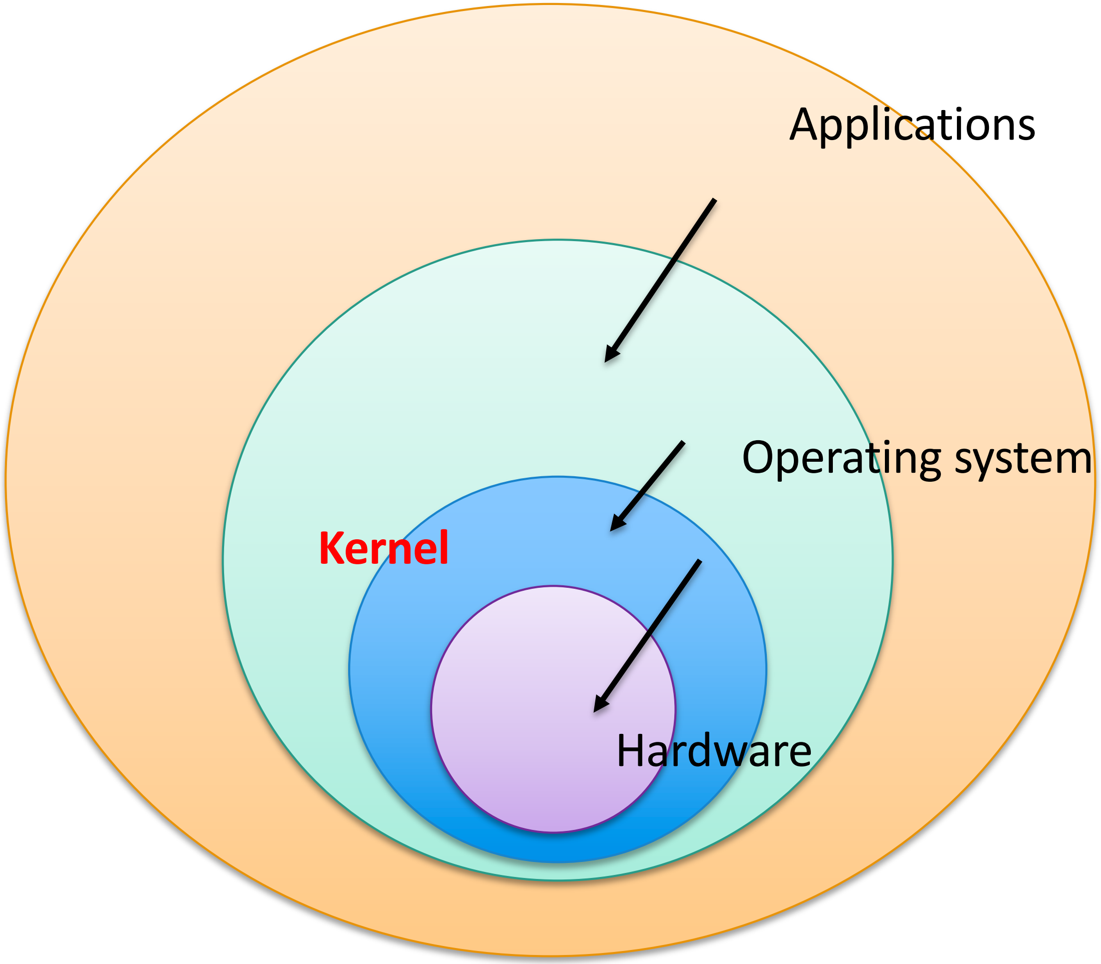
# Typical structure of OS (unix)

- Note: By structure we simply meant control flow



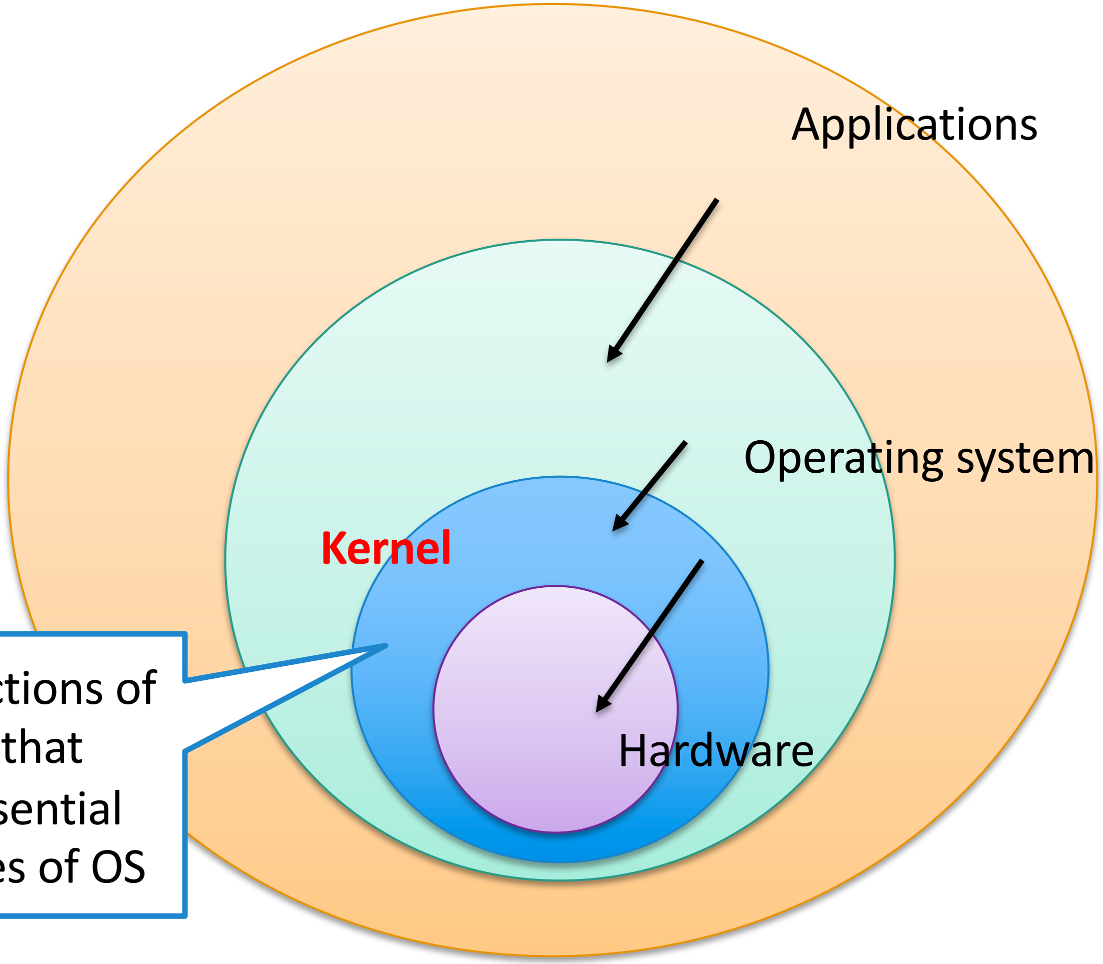
# Typical structure of OS (unix)

- Note: By structure we simply meant control flow



# Typical structure of OS (unix)

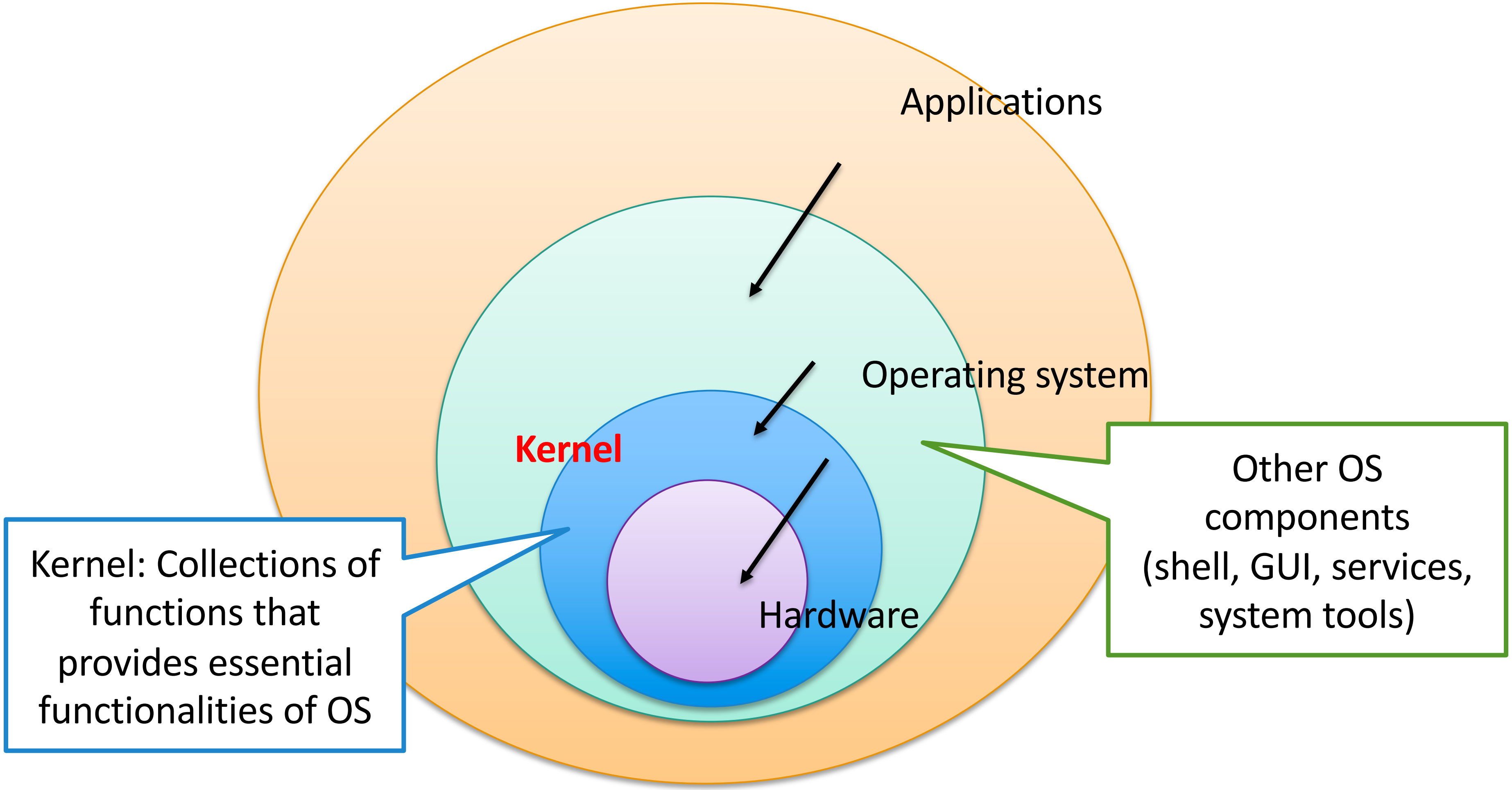
- Note: By structure we simply meant control flow



Kernel: Collections of functions that provides essential functionalities of OS

# Typical structure of OS (unix)

- Note: By structure we simply meant control flow



Kernel: Collections of functions that provides essential functionalities of OS

Other OS components (shell, GUI, services, system tools)

# Function of Kernel

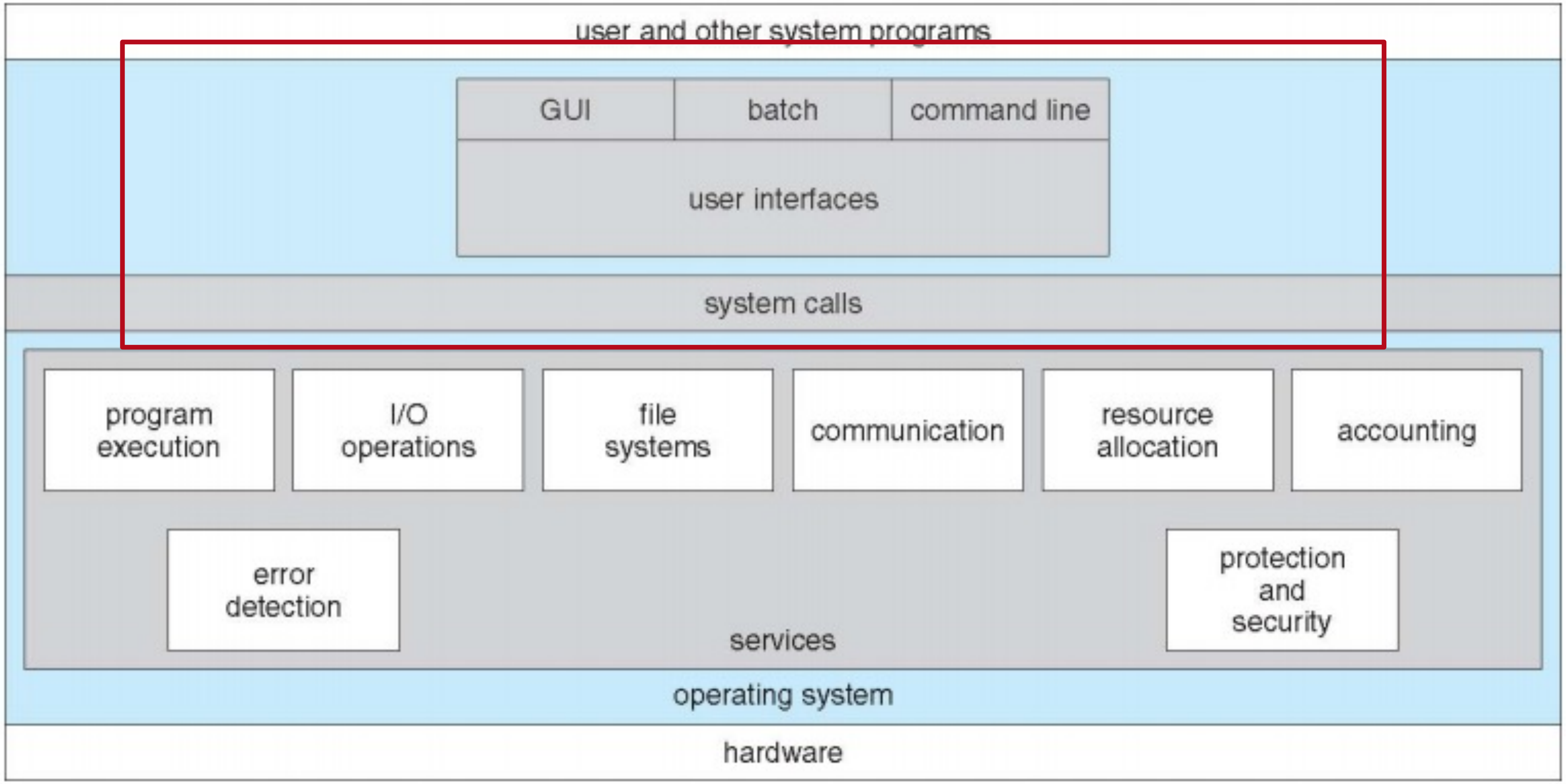
- Part of OS closest to the hardware, handles important functionalities
  - Managing memory, network, file, processes, system calls …

# Function of Kernel

- Part of OS closest to the hardware, handles important functionalities
  - Managing memory, network, file, processes, system calls …

- Other parts of OS (e.g., the shell) and application programs can interact with kernel whenever they require these functionalities
  - E.g., need to read from keyboard (*scanf*), show something on display (*printf*), create directory (*mkdir*)

# What other services?



Source: Silberschatz, Galvin and Gagne ©2013

# Food for thought

- A user might ask for *printf* any time
    - Does “Kernel run all the times”?

# Food for thought

- A user might ask for *printf* any time
  - Does “Kernel run all the times”?
- Possible problem
  - Kernel needs resources to run

# Food for thought

- A user might ask for *printf* any time
  - Does “Kernel run all the times”?
- Possible problem
  - Kernel needs resources to run
- Insight
  - Kernel “only” needs to run when any of its functionalities are required

# Today’s class

- Resources and notes

- Introduction to operating systems (OS)
  - What is an OS
  - What are the goals of an OS
  - Under the hood: the structure of OS
  - How does OS work?

# Today's class

- Resources and notes

- Introduction to operating systems (OS)
    - What is an OS
    - What are the goals of an OS
    - Under the hood: the structure of OS
    - How does ~~OS~~ work?

Kernel

# Kernel is “interrupt” driven

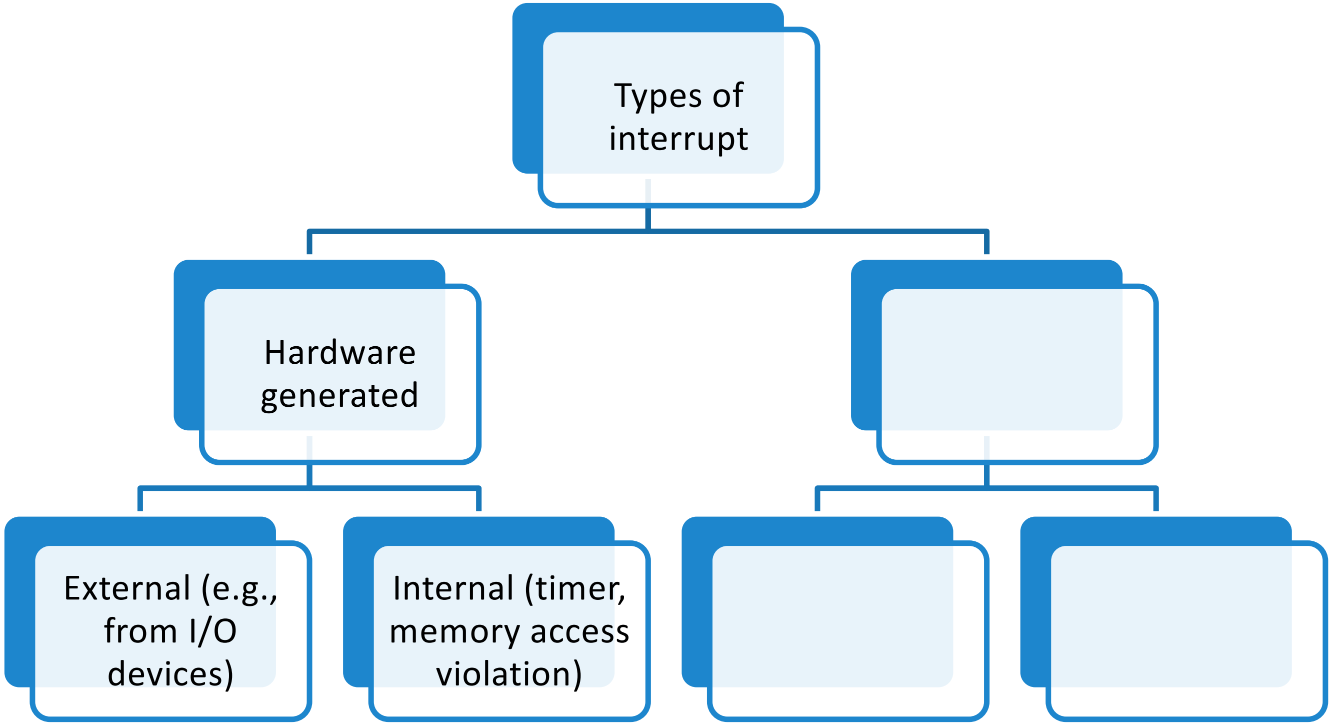
- Think of interrupt as the “wake up call” to kernel
  - When interrupt comes, some function in Kernel is invoked

# Kernel is “interrupt” driven

- Think of interrupt as the “wake up call” to kernel
    - When interrupt comes, some function in Kernel is invoked

- More technically
    - Interrupt is a signal (instruction), generated by hardware/software
    - The interrupt in turn activates appropriate kernel routine(s) depending on specific category of the interrupt

# Types of interrupt



Types of interrupt

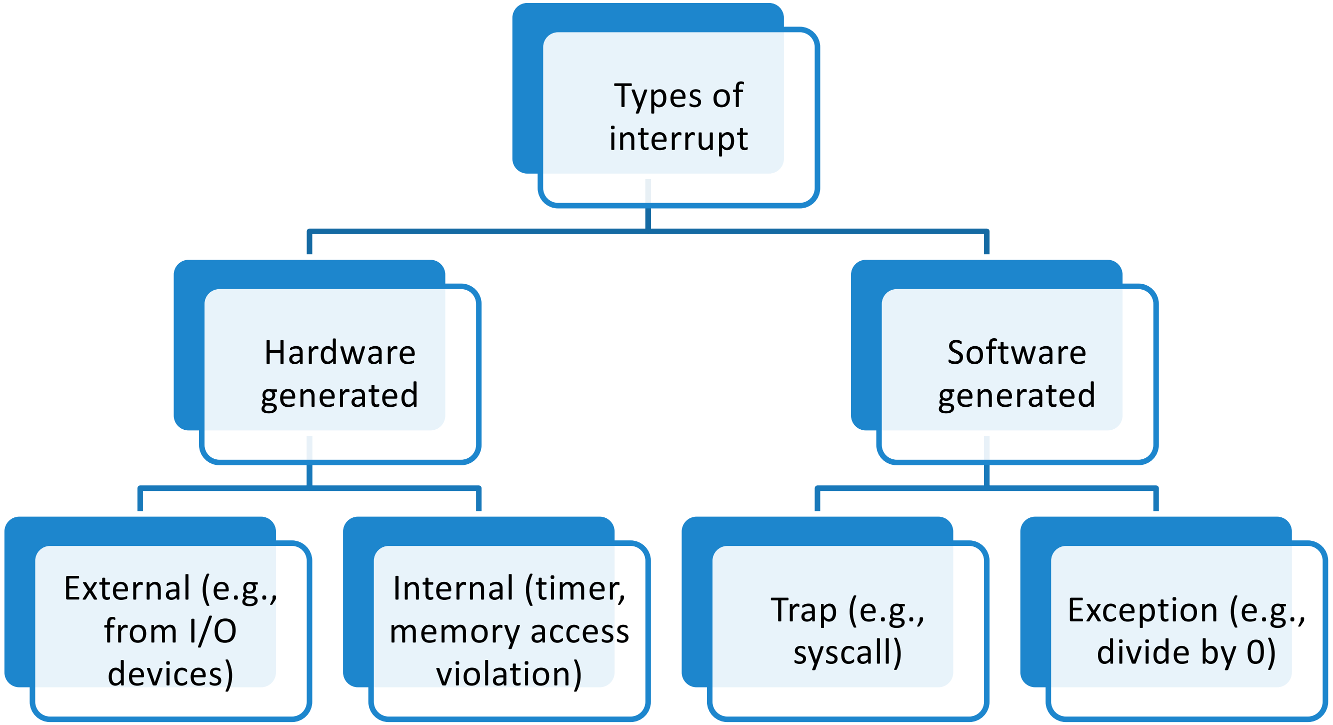
Hardware generated

External (e.g., from I/O devices)

Internal (timer, memory access violation)

Hardware interrupts: generated by external device (e.g., a printer) OR internal hardware unit

# Types of interrupt

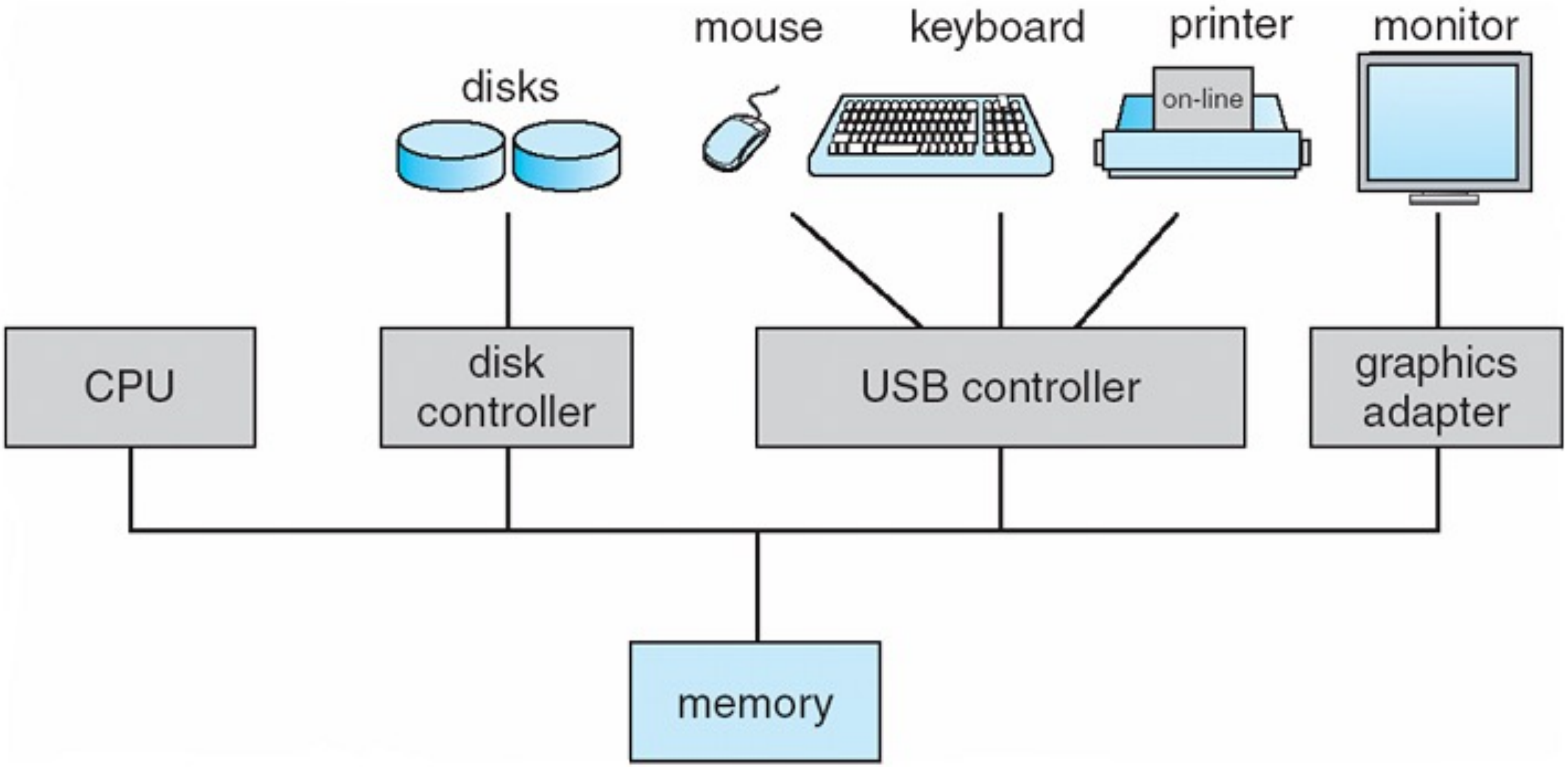

Types of interrupt
Hardware generated
Software generated
External (e.g., from I/O devices)
Internal (timer, memory access violation)
Trap (e.g., syscall)
Exception (e.g., divide by 0)

Hardware interrupts: generated by external device (e.g., a printer) OR internal hardware unit

Software interrupts: Generated by a running program

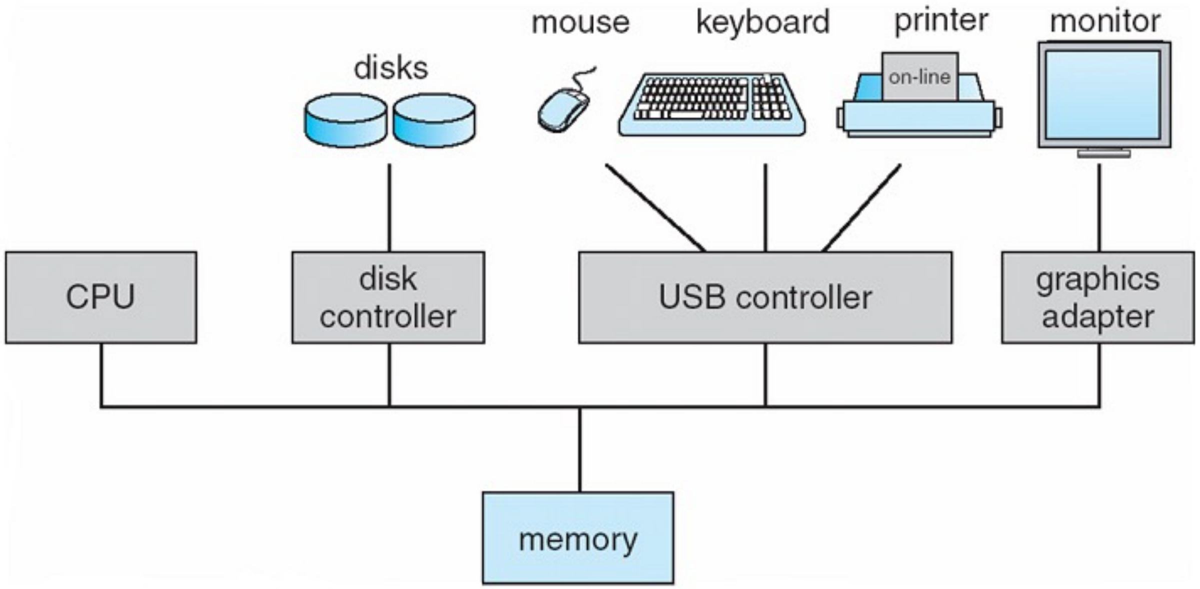
# Case study: Handling Input/Output (I/O) requests

# The setup



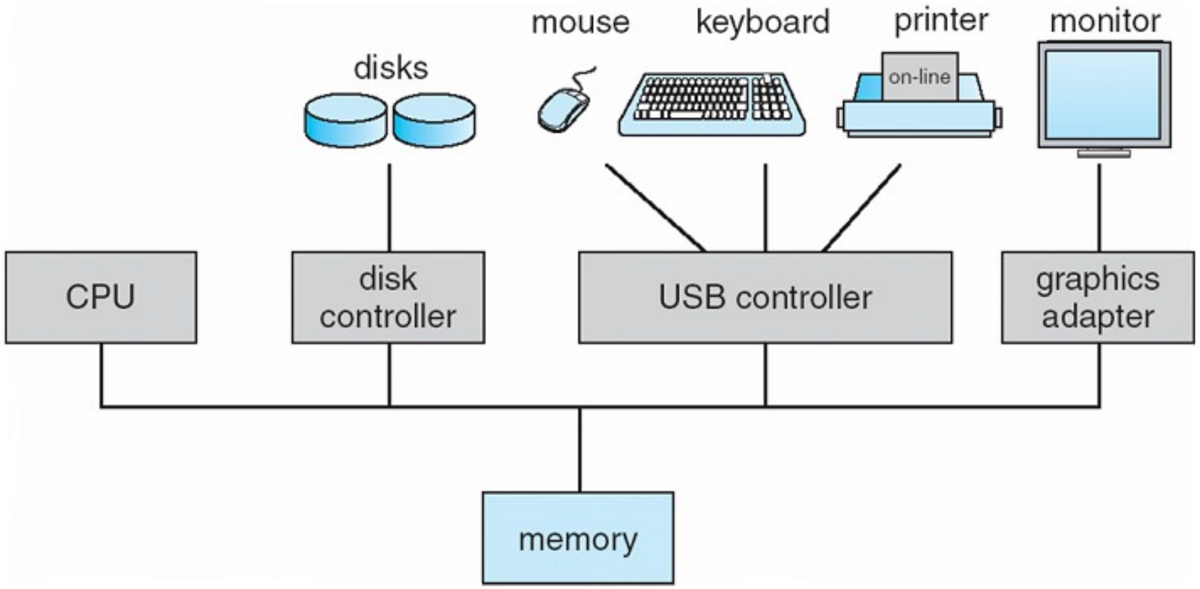
Source: Silberschatz, Galvin and Gagne ©2013

# The setup (contd.)



- I/O devices and CPU can execute in parallel
    - Each device controller is in charge of a particular device type
    - Each device controller has a local buffer
    - Data transfers between local buffer and main memory
    - Device controller sends an interrupt to the CPU to indicate I/O completion

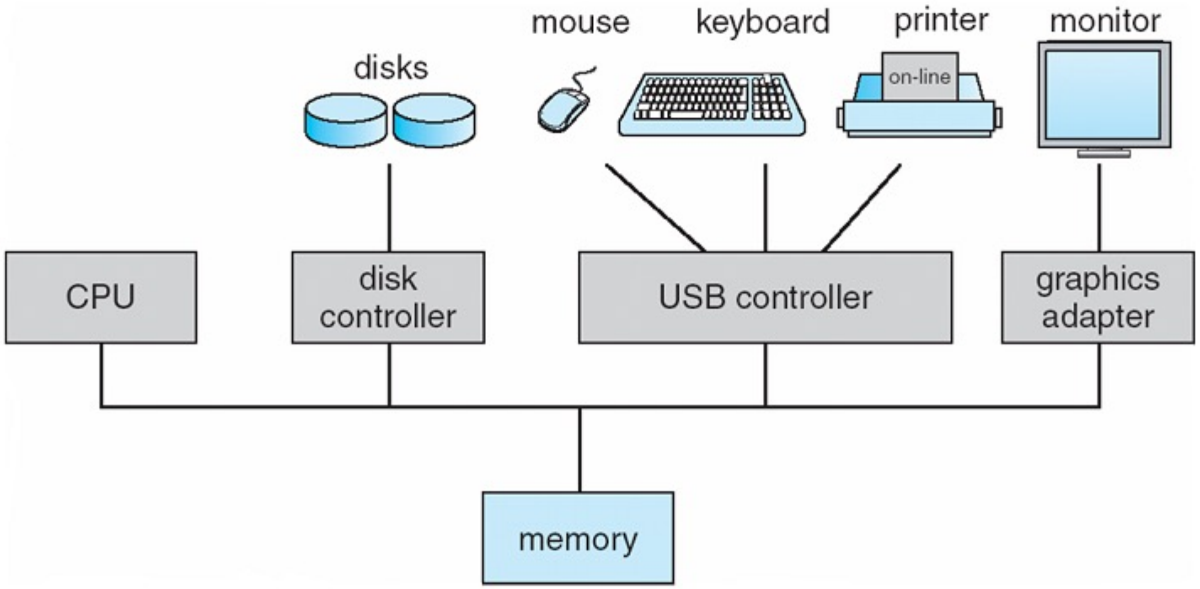
# The setup (contd.)



Question: How would an I/O request be handled in this setup?

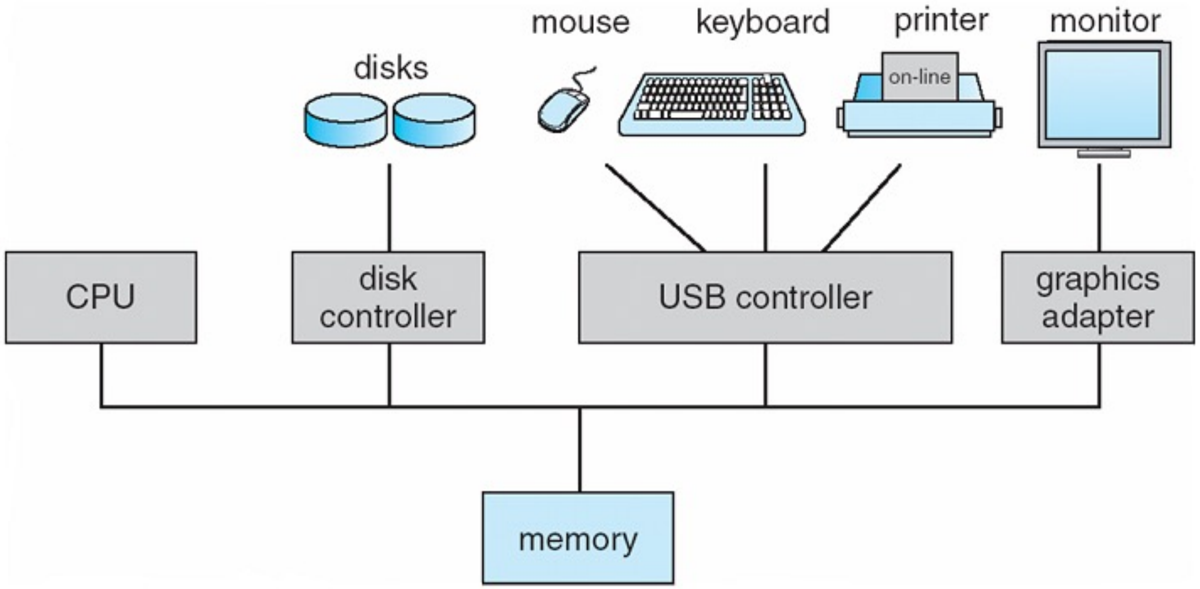
- Each device controller has a local buffer
- Data transfers between local buffer and main memory
- Device controller sends an interrupt to the CPU to indicate I/O completion

# Handling I/O in the setup



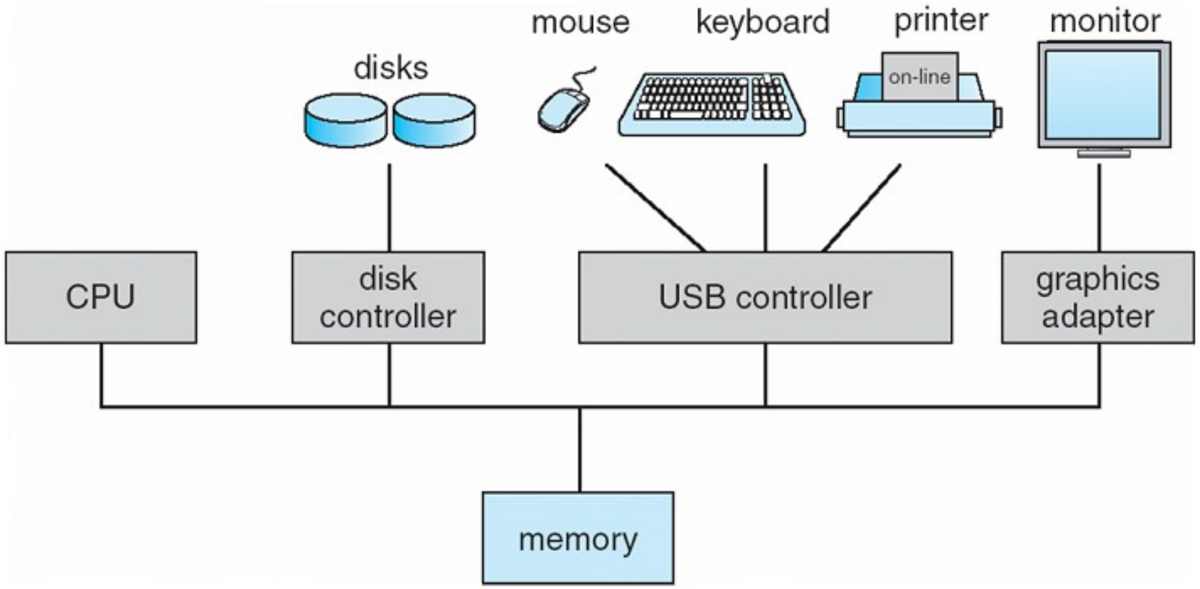
- Option 1
  - User program makes I/O request - transfers data from memory to buffer of device controller and initiates I/O
  - CPU remains idle as long as device controller handles request
  - Upon I/O completion, device controller sends interrupt, then execution of user program resumes
  - "Busy waiting" - under-utilization of CPU

# Handling I/O in the setup



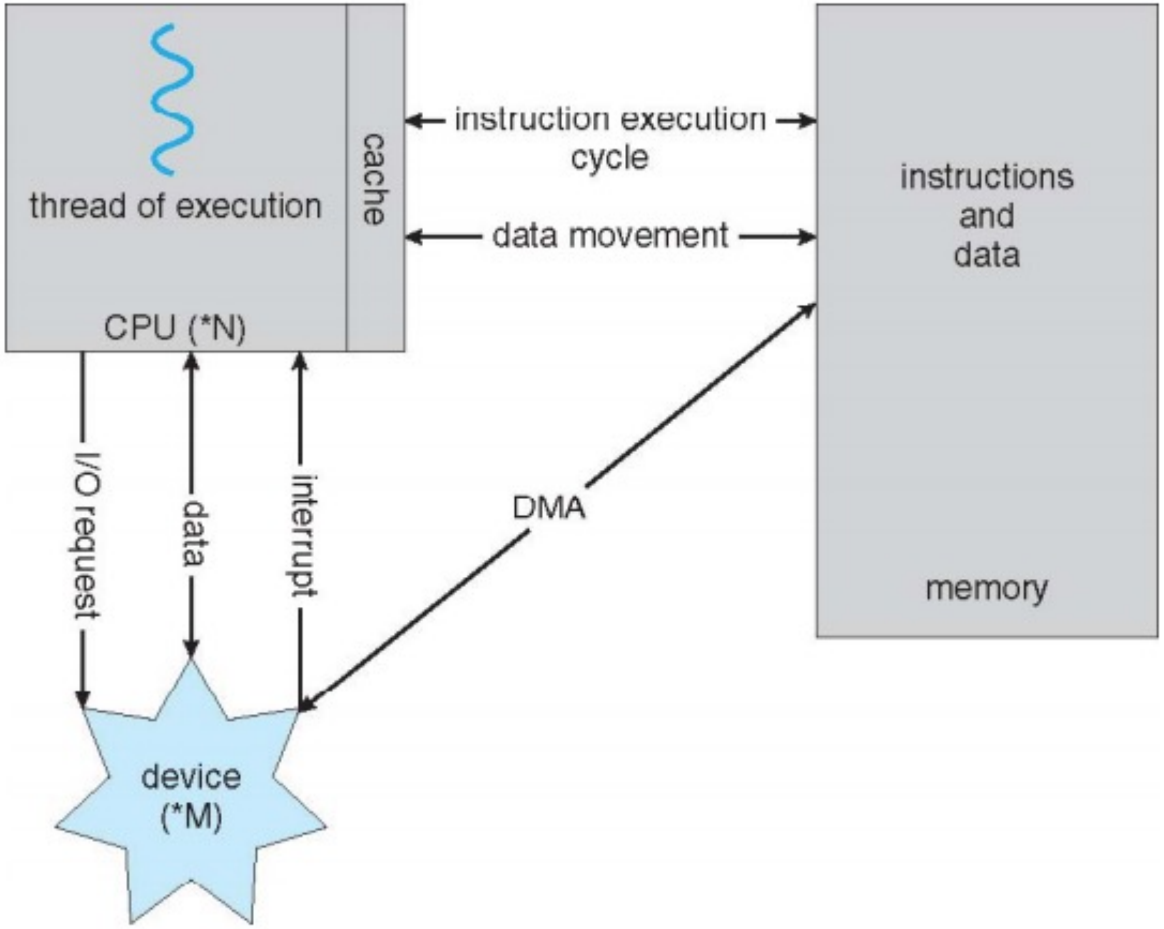
- Option 2: use OS
  - User program makes I/O request through a "system call"
  - OS transfers data from memory to buffer of device controller and initiates I/O
  - OS can allocate CPU to other programs during I/O of one program
  - Upon I/O completion, device controller sends interrupt, then OS may resume this particular user program (according to some scheduling policy)
  - "Multiprogramming" - much better utilization of CPU

# Handling I/O in the setup



- Previous scheme of interrupt-driven I/O is fine for small amounts of data, but high overhead for bulk data transfer such as disk I/O
- Hence option 3 – Direct Memory Access (DMA)
  - Once OS initiates I/O, DMA manages transfer of data between memory and device controller with no intervention by the CPU

# I/O handling in von Neumann architecture



Source: Silberschatz, Galvin and Gagne ©2013