# Overview of Operating Systems

施吉昇

- Spring 2022
- National Taiwan Unive

國立臺灣大學

NEWS Lab
嵌入式系統暨無線網路實驗室

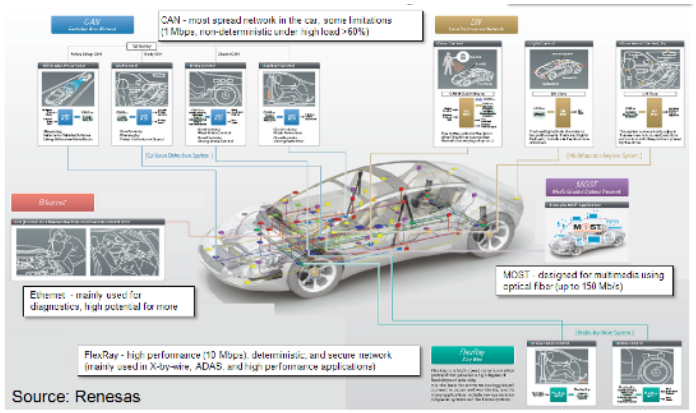
# Agenda

- Computing Systems and Computation Modes
- Computing System Architecture
- Type of Operating Systems
- Interface with Operating Systems
- Functional Components of Operating Systems

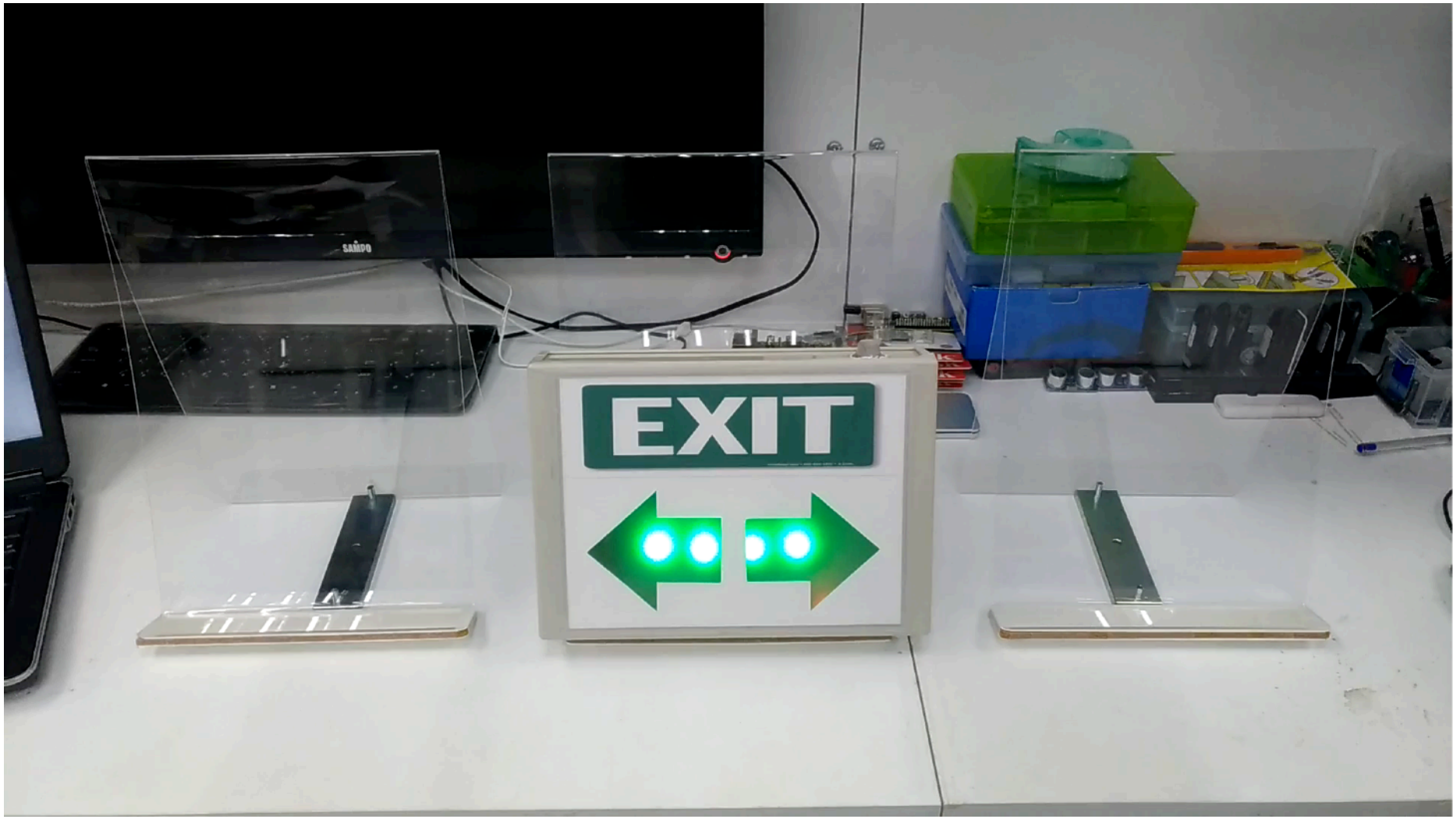
# Computing Systems and Computation Mode

# What can Computing Systems serve?

- You know
  - Laptop, Smartphone, workstation, etc
- How about these?
  - Robots
  - Self-driving Car
  - Modern airplane: Airbus A380, Boeing 777, etc.
  - Printer
  - Washing Machine
  - Toaster
  - Spaceship
  - Emergency Sign

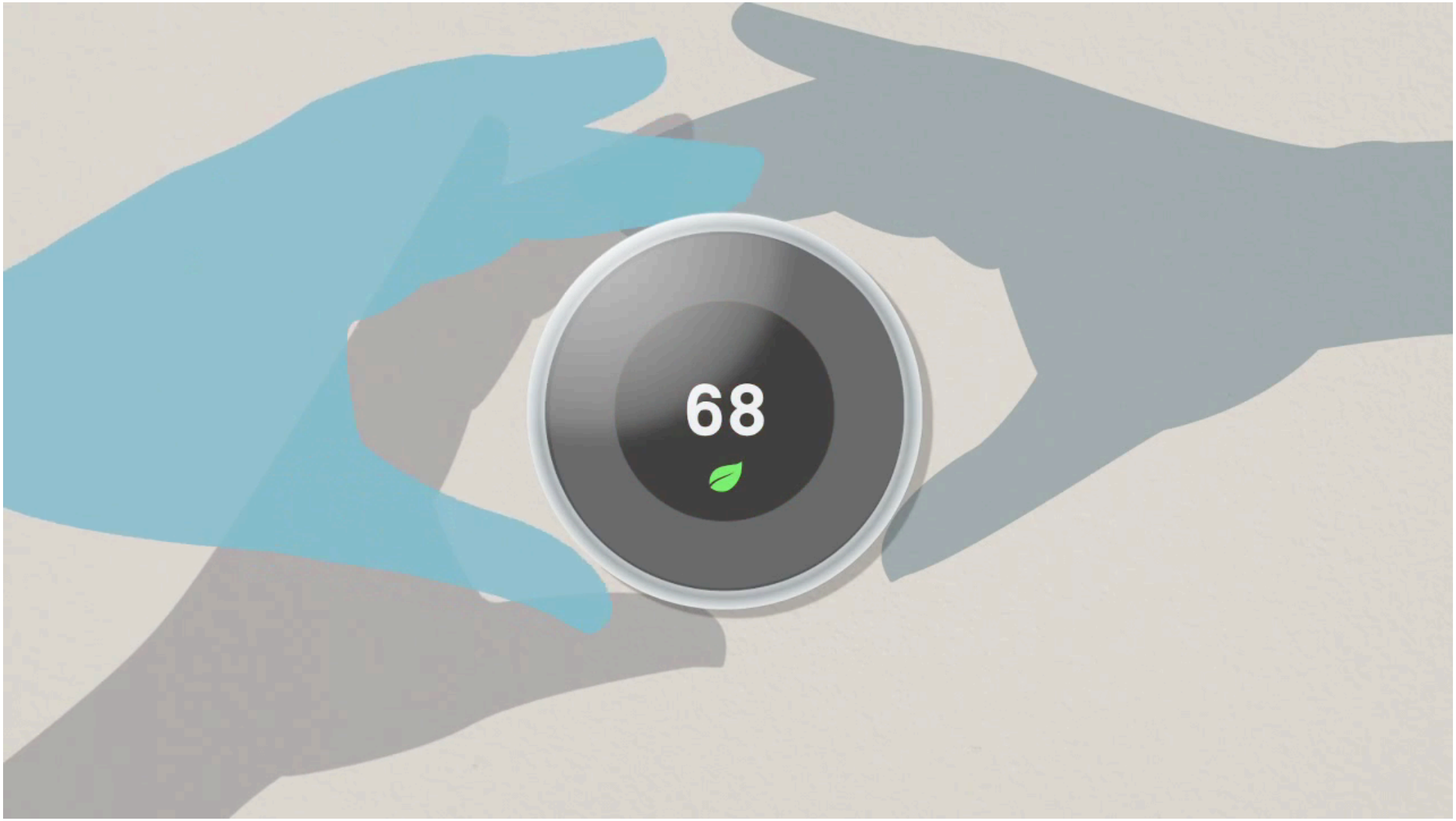


# Emergency Exit Sign



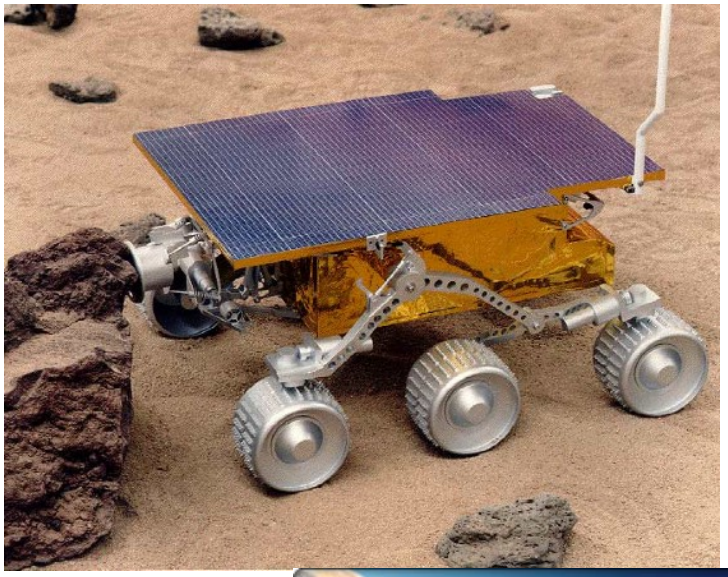
# Thermostat

# Thermostat

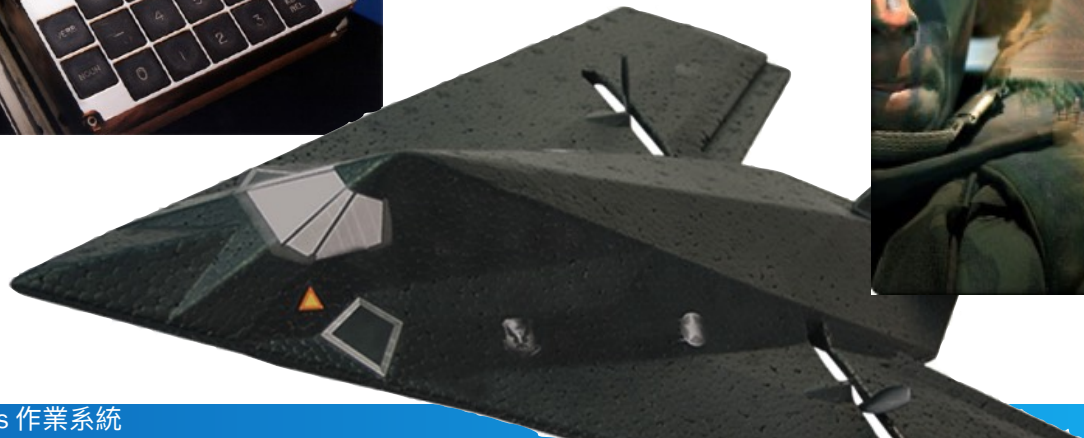


# Many Others


Millennium




週六 01
BARO 1025
ALT 2148
SET 8:3 p
H 37 | L 33
BATT 21d
GARMIN



# Computation Modes

Batch Mode

Online Mode

Real-Time Mode

# Do they have the same requirements?

- Users may interact with program.
- No time constraints involved.
- Example: Spreadsheet, games, browsing

- **Interactive Mode**: each process needs to take inputs and generate outputs from time to time.
  - Performance Metrics: throughput
  - Resource allocation policy: the operating systems need to allocate fair share of resources to each process in order to take inputs and generate outputs. No process should occupy the CPU/memory for long time.

**Responsiveness**

# Do they have the same requirements?

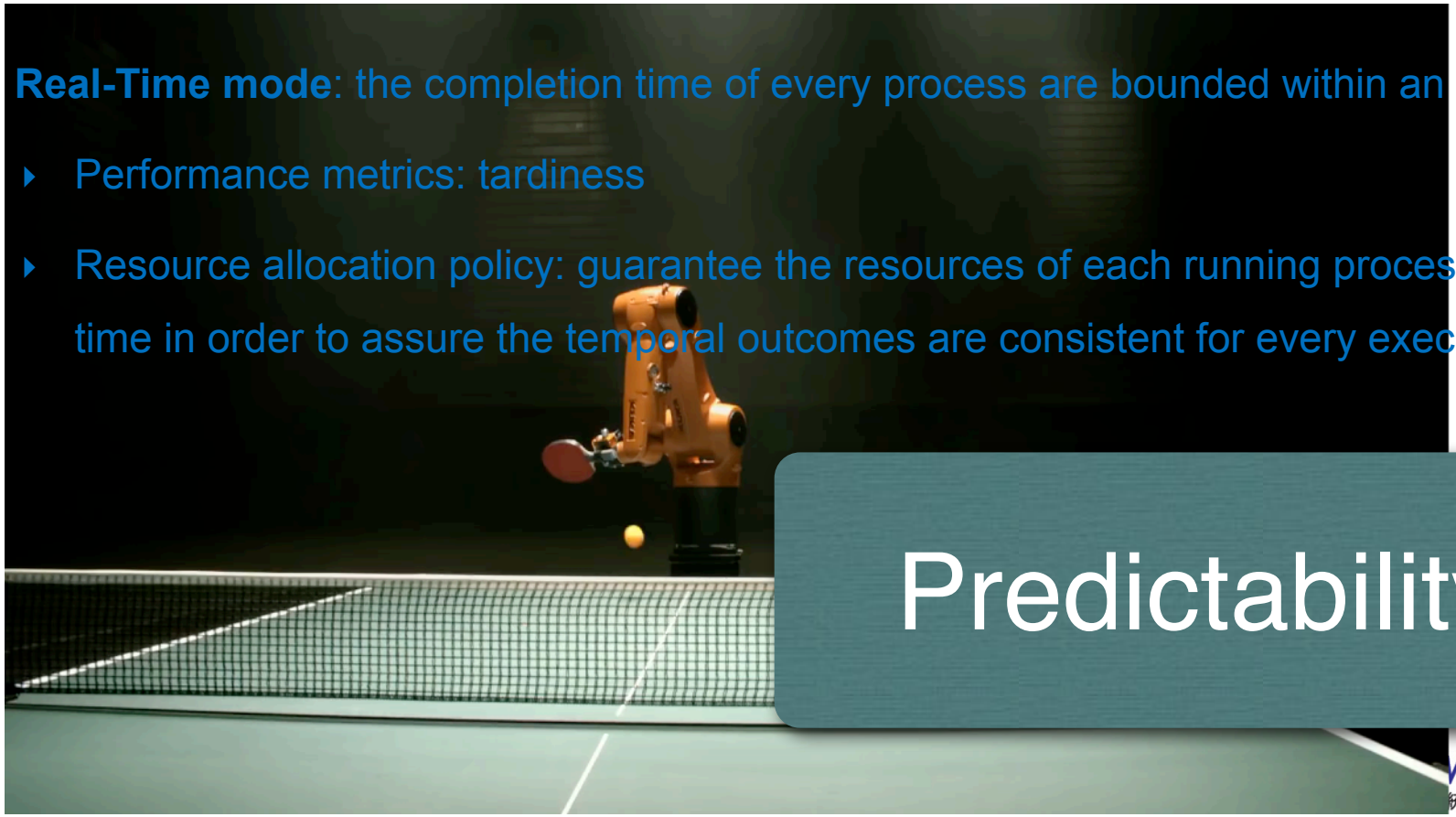
- Program runs independent of outside world.
- Results written to file or output device.
- Example: Compiler, Mining, ML

- **Batch mode**: one process needs to complete as soon as possible.
  - Performance metrics: completion time for one process (not each).
  - Resource allocation policy: allocate sufficient amount of resources before THE process starts in order to shorten the execution time and completion time.

Efficiency

# Do they have the same requirements?

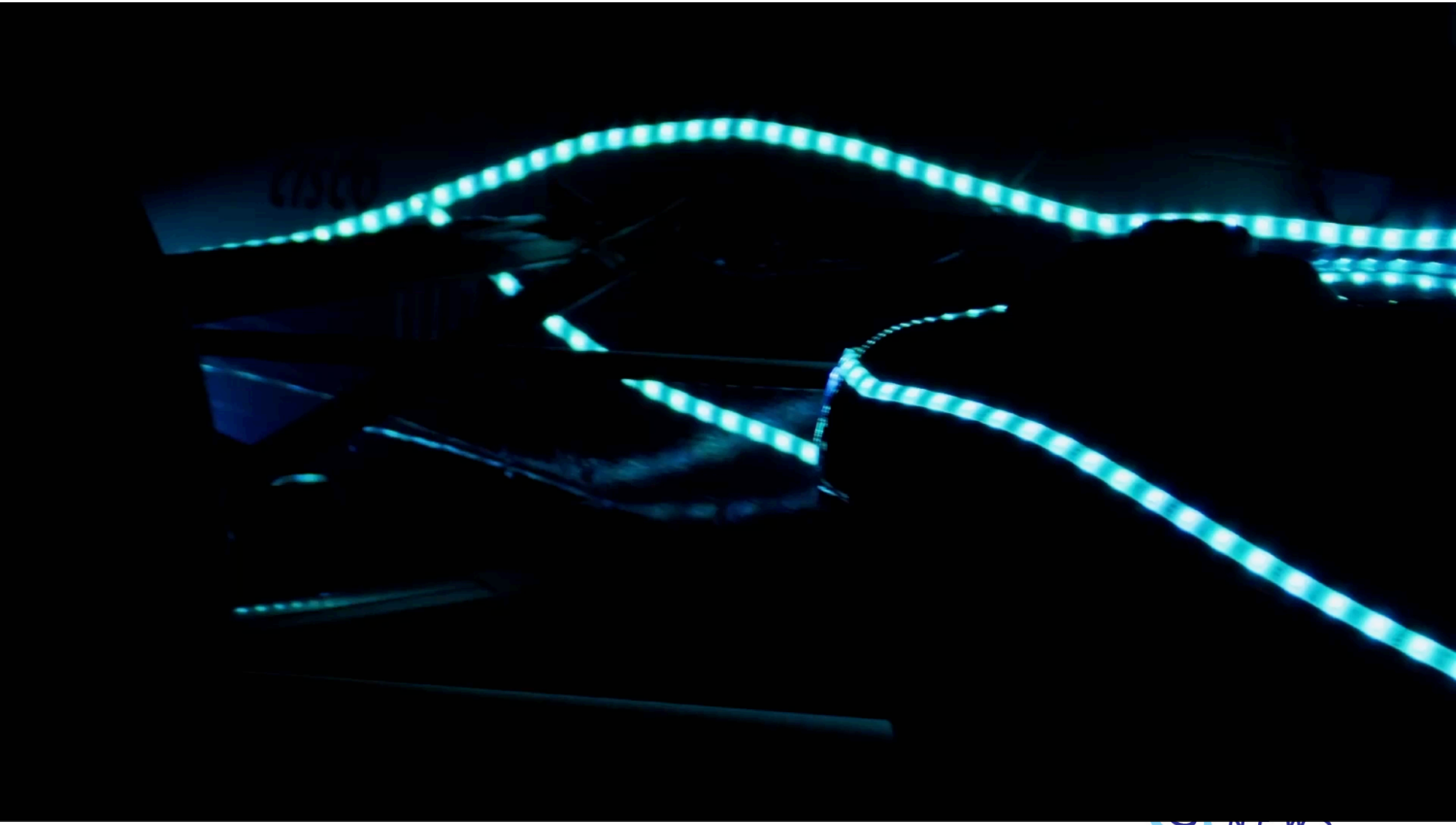
- Program interacts strongly with environment.
- Time constraints imposed.
- Example: robot control, self-driving car



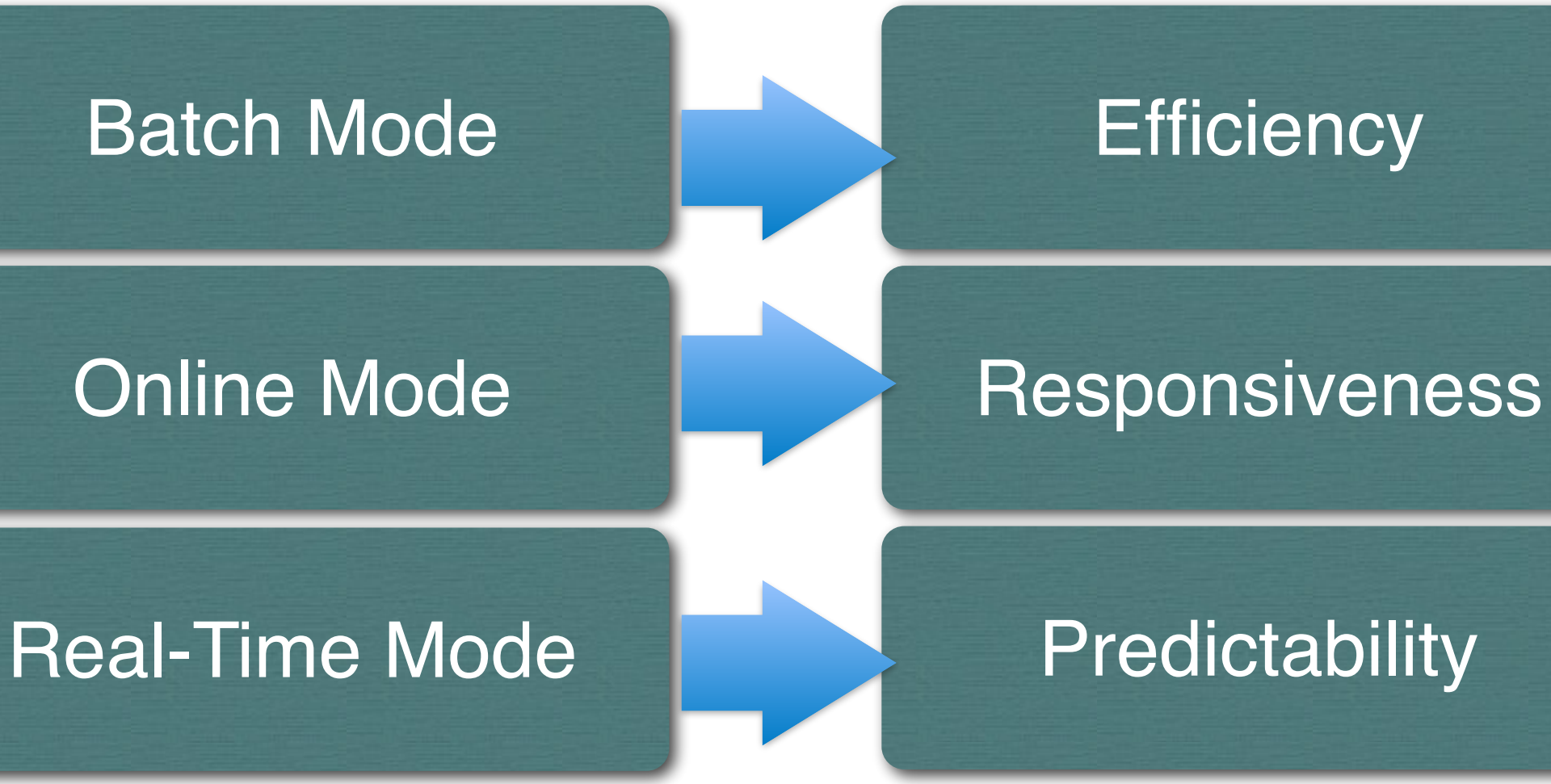
- **Real-Time mode**: the completion time of every process are bounded within an interval.
  - Performance metrics: tardiness
  - Resource allocation policy: guarantee the resources of each running process at ALL time in order to assure the temporal outcomes are consistent for every execution.

**Predictability**

# Another Challenge

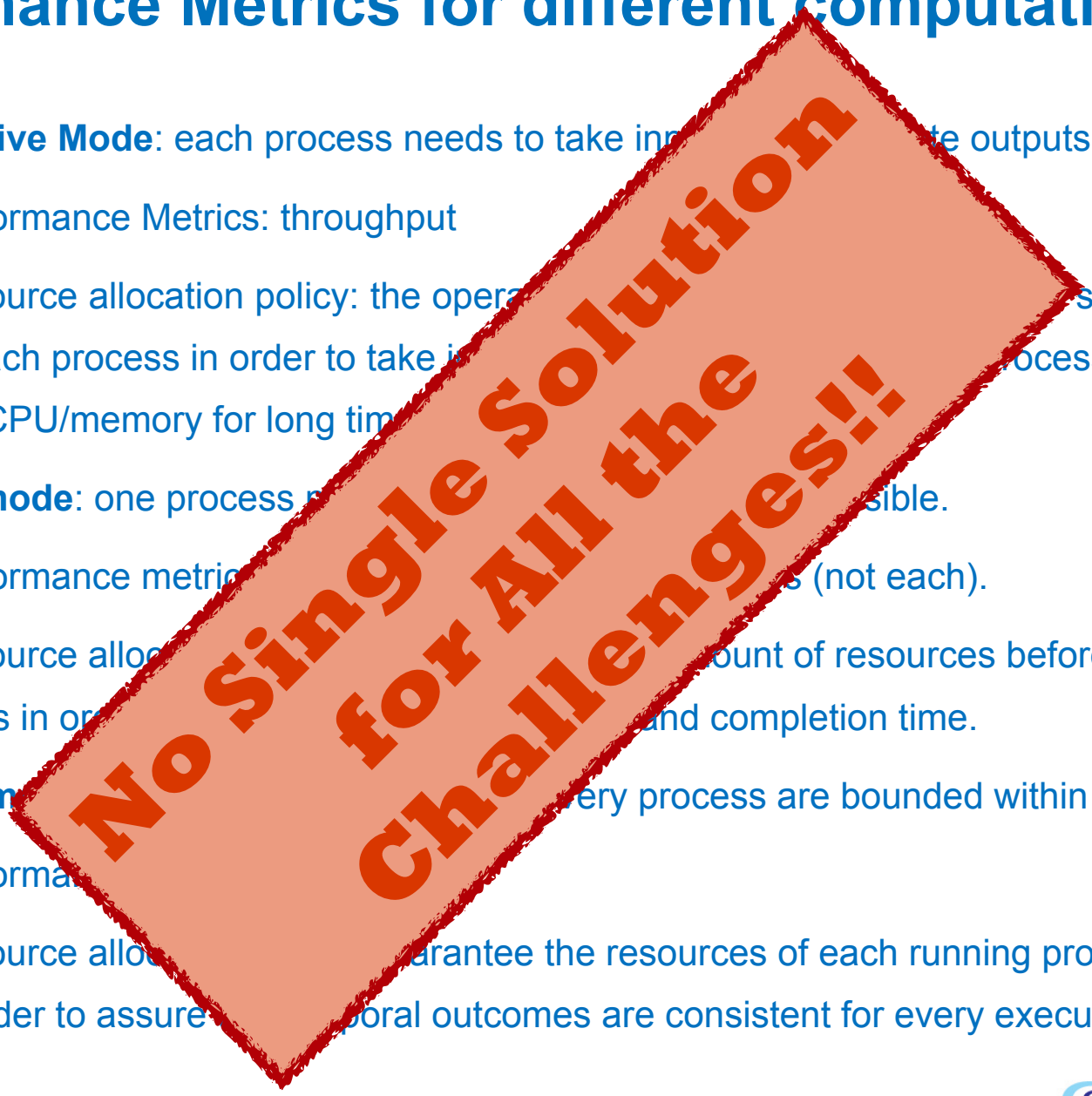


# Do they have the same requirements?



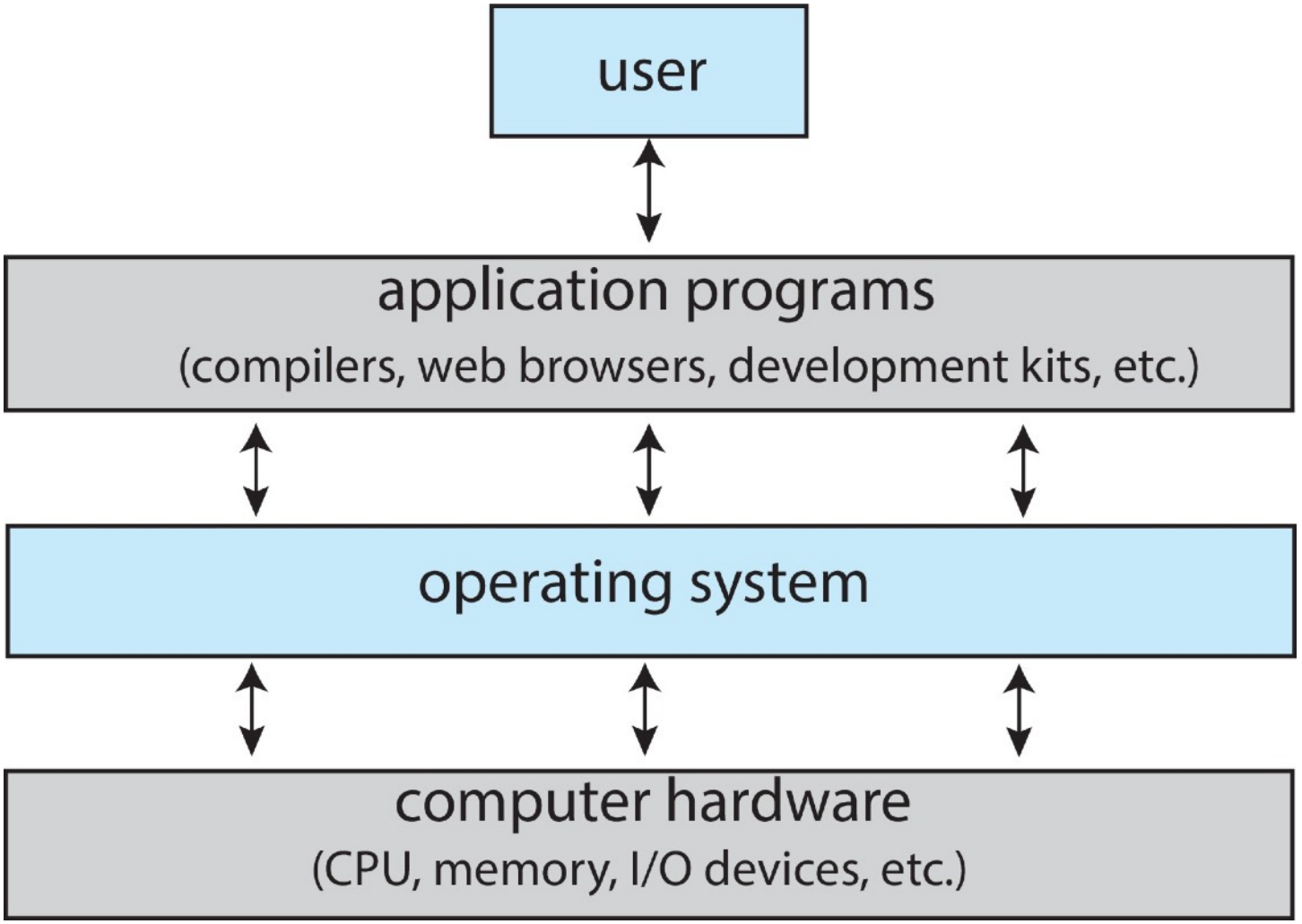
# Performance Metrics for different computation mode

- **Interactive Mode**: each process needs to take in… …te outputs from time to time.
  - Performance Metrics: throughput
  - Resource allocation policy: the opera… …share of resources to each process in order to take i… …ocess should occupy the CPU/memory for long tim…
- **Batch mode**: one process … …sible.
  - Performance metric… …s (not each).
  - Resource alloc… …ount of resources before THE process starts in or… …nd completion time.
- **Real-Tim**… …ery process are bounded within an interval.
  - Performa…
  - Resource allo… …arantee the resources of each running process at ALL time in order to assure … …poral outcomes are consistent for every execution.



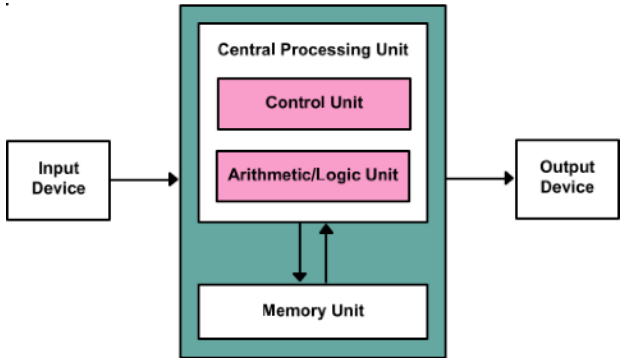
# Computing System Architecture

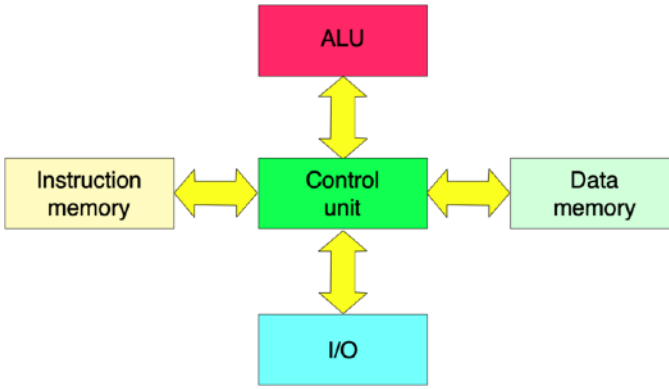
# Abstract View of Components of Computer



# von Neumann Architecture

- The von Neumann architecture—also known as the von Neumann model or Princeton architecture—is a computer architecture based on a 1945 description by John von Neumann and others in the First Draft of a Report on the EDVAC.
- The Harvard architecture is a computer architecture with separate storage and signal pathways for instructions and data.
- Both of them are stored-program computers, which originates from Turing Machine.
- The major difference is if the bus is shared between data and instruction.
  - Most modern processors appear to be von Neumann machines. However,
  - internally, caches for instruction and data are separated as a modified Harvard architecture.



# Challenge

## Don’t take everything on textbook for granted

- Are von Neumann and Harvard good enough for modern/future computing?
  - In both architectures, processors are placed at the center of the computer. In other words, processors have the most heavy loading and complex workload. All other components work together to support processors.
  - Is it still true?
- Modern GPUs can execute single instruction on thousands of GPU cores simultaneously in one or two cycles. However, when using deep neural networks,
  - moving the results, which are data, for next instruction can take tens, hundreds, or thousands of cycles.

# Definition of Operating Systems

## No universally accepted definition

- Conceptually, Operating systems overlook the hardware components to serve the applications.
- Everything else is either
  - A ***system program*** (ships with the operating system, but not part of the kernel) , or
  - An ***application program***, all programs not associated with the operating system
- Today's OSs for general purpose and mobile computing also include ***middleware*** – a set of software frameworks that provide addition services to application developers such as *dockers*, databases, multimedia, graphics

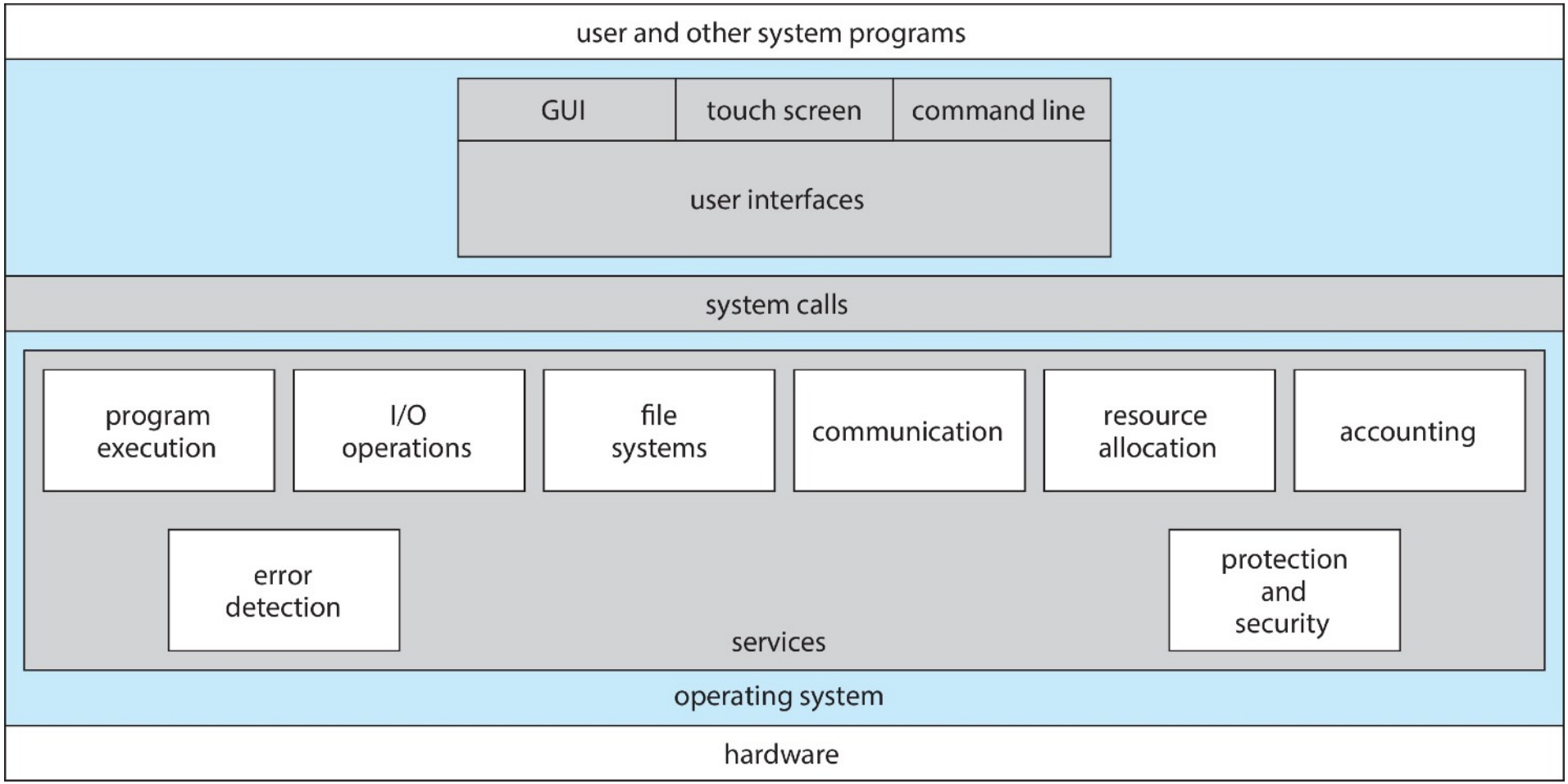
# Type of Operating Systems

# Stand Alone (Uni-Processor) Operating Systems

- An OS acts as a resource manager or an arbitrator
  - Manages CPU, I/O devices, memory
- OS provides a virtual interface that is easier to use than hardware

Remember what we did last week to write a ‘Hello World’ without Operating Systems?

# A View of Operating System Services



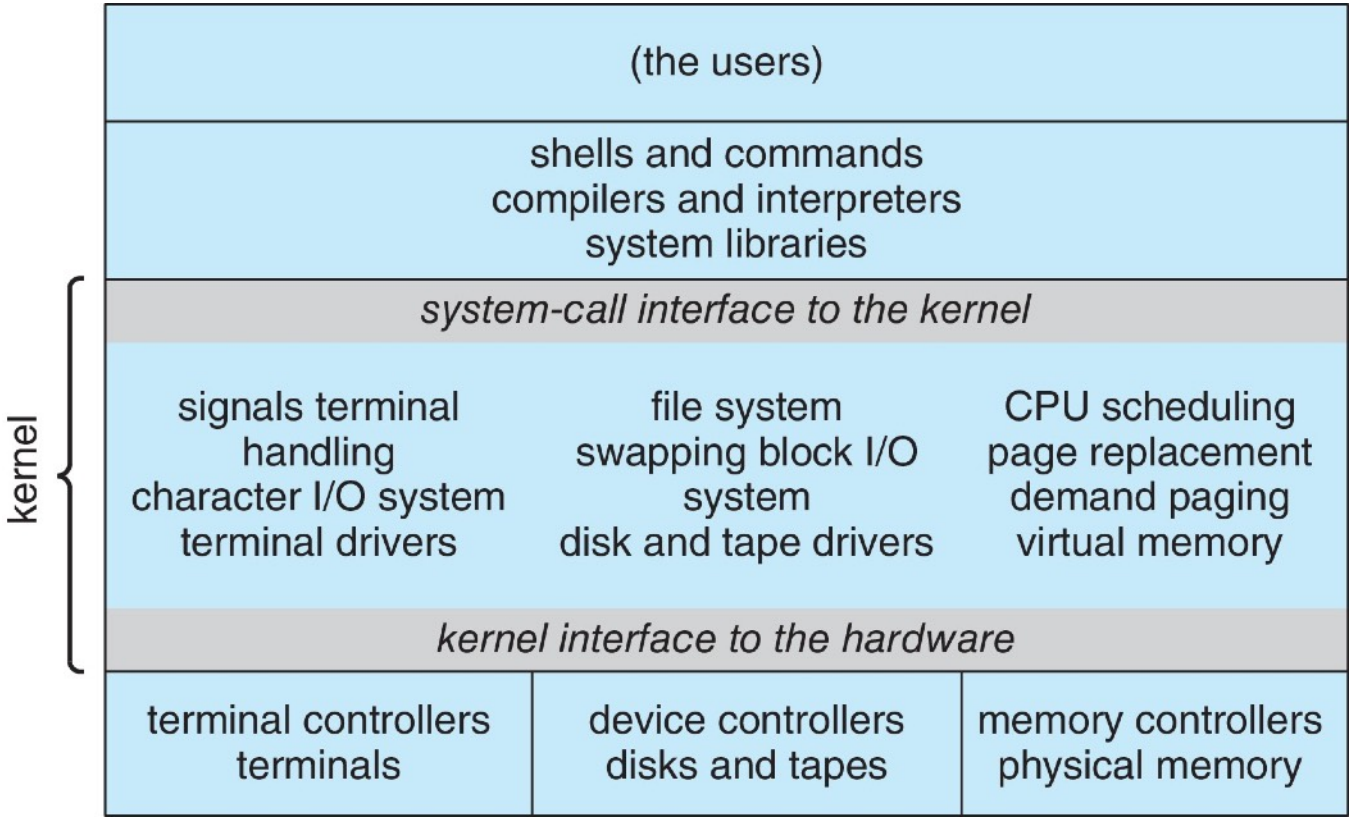
# Stand Alone (Uni-Processor) Operating Systems

- An OS acts as a resource manager or an arbitrator
  - Manages CPU, I/O devices, memory
- OS provides a virtual interface that is easier to use than hardware
- Structure of uniprocessor operating systems
  - Monolithic (e.g., MS-DOS, early UNIX)
    - One large kernel that handles everything
  - Layered design
    - Functionality is decomposed into N layers
    - Each layer uses services of layer N-1 and implements new service(s) for layer N+1

# Monolithic Structure – Original UNIX

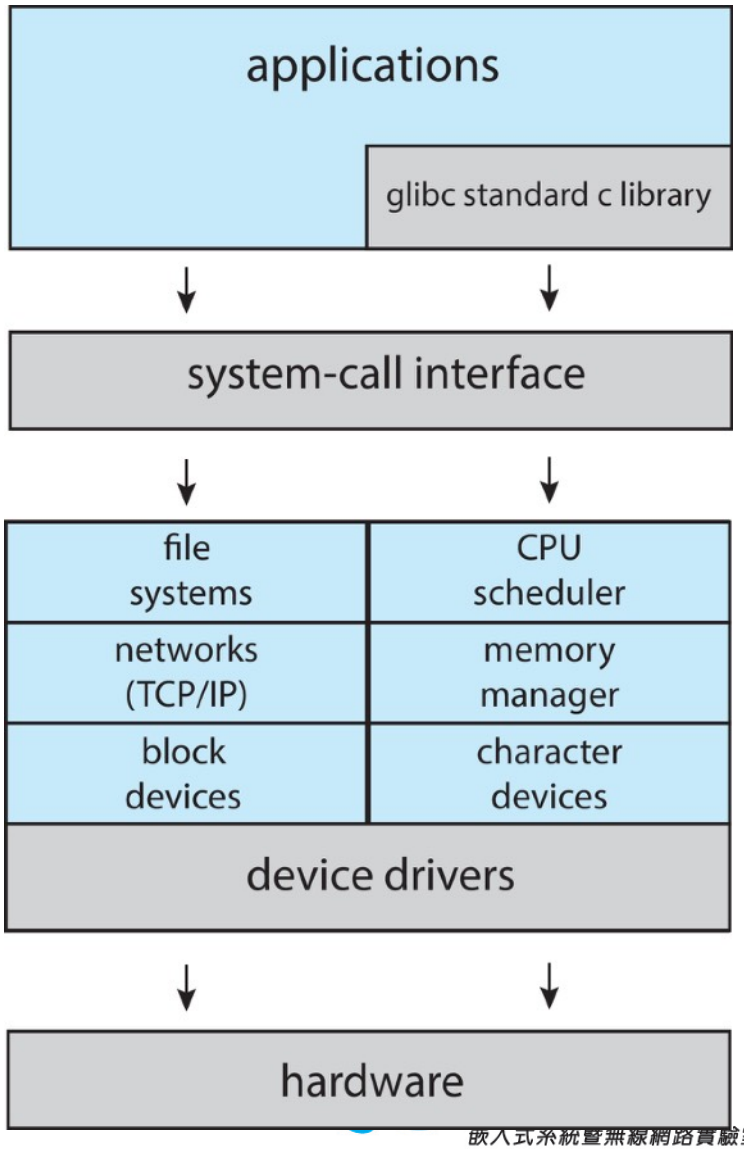
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel
    - Consists of everything below the system-call interface and above the physical hardware
    - Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

# Traditional UNIX System Structure



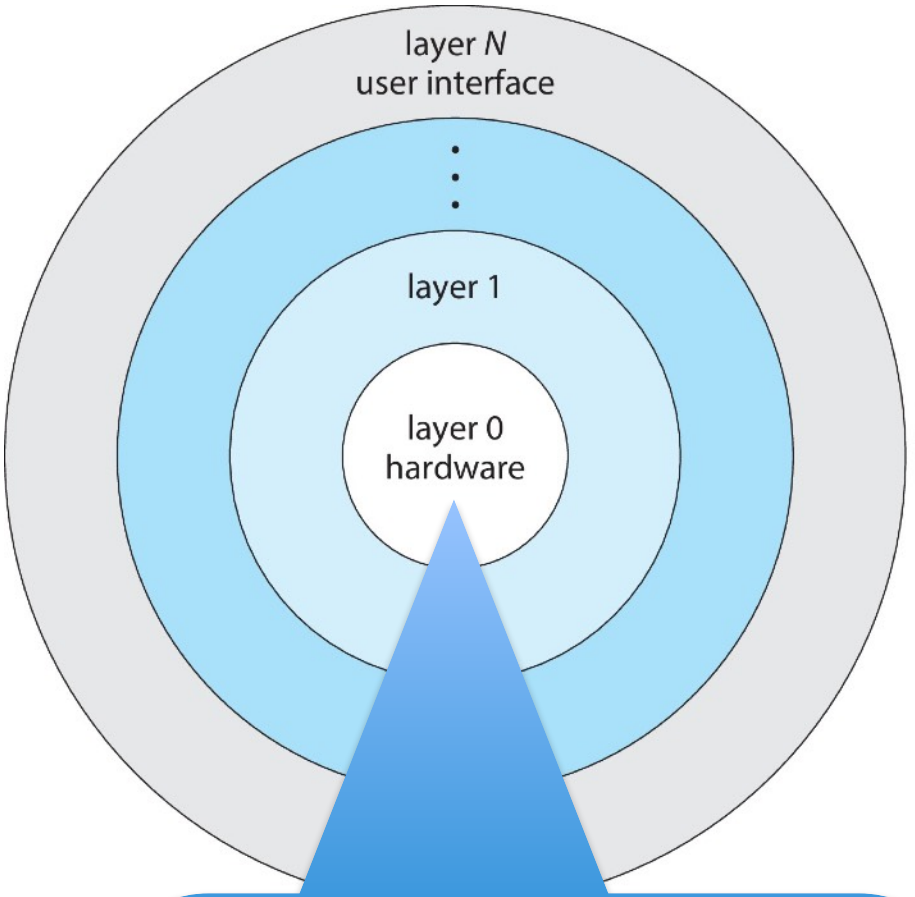
# Linux System Structure

- The Linux kernel is monolithic in that it runs entirely in kernel mode in a single address space
  - It does have a modular design that allows the kernel to be modified during run time
- Applications typically use the glibc standard C library when communicating with the system call interface to the kernel.



# Ring-based/Layer-Designed Operating Systems

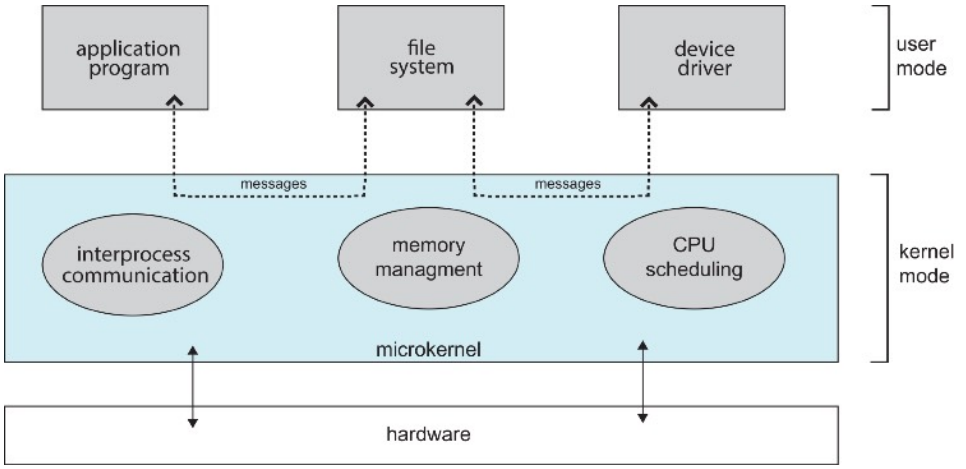
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (Ring 0), is the hardware; the highest layer (Ring N) is the user interface.
  - There are ring -1 ~ -3 on some processors.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

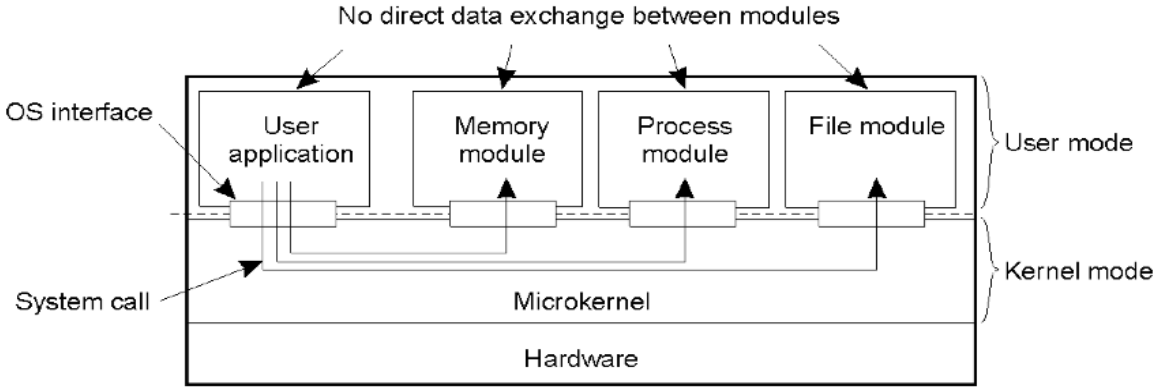



Intel and AMD have introduced a new privilege level: The hypervisor can now run at "Ring -1"; so the guest operating systems can run in Ring 0.

# Micro-Kernel Architecture

- Microkernel architecture
  - Small kernel
  - User-level servers implement additional functionality



# Microkernels

- Moves as much from the kernel into user space
- **Mach** is an example of **microkernel**
  - Mac OS X kernel (**Darwin**) partly based on Mach
- Communication takes place between user modules using **message passing**
- Benefits:
  - Easier to extend a microkernel
  - Easier to port the operating system to new architectures
  - More reliable (less code is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user space to kernel space communication

# Modules

- Many modern operating systems implement **loadable kernel modules** (**LKMs**)
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
  - Linux, Solaris, etc.

# Hybrid Systems

- Most modern operating systems are not one pure model
  - Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

# Distributed Operating Systems

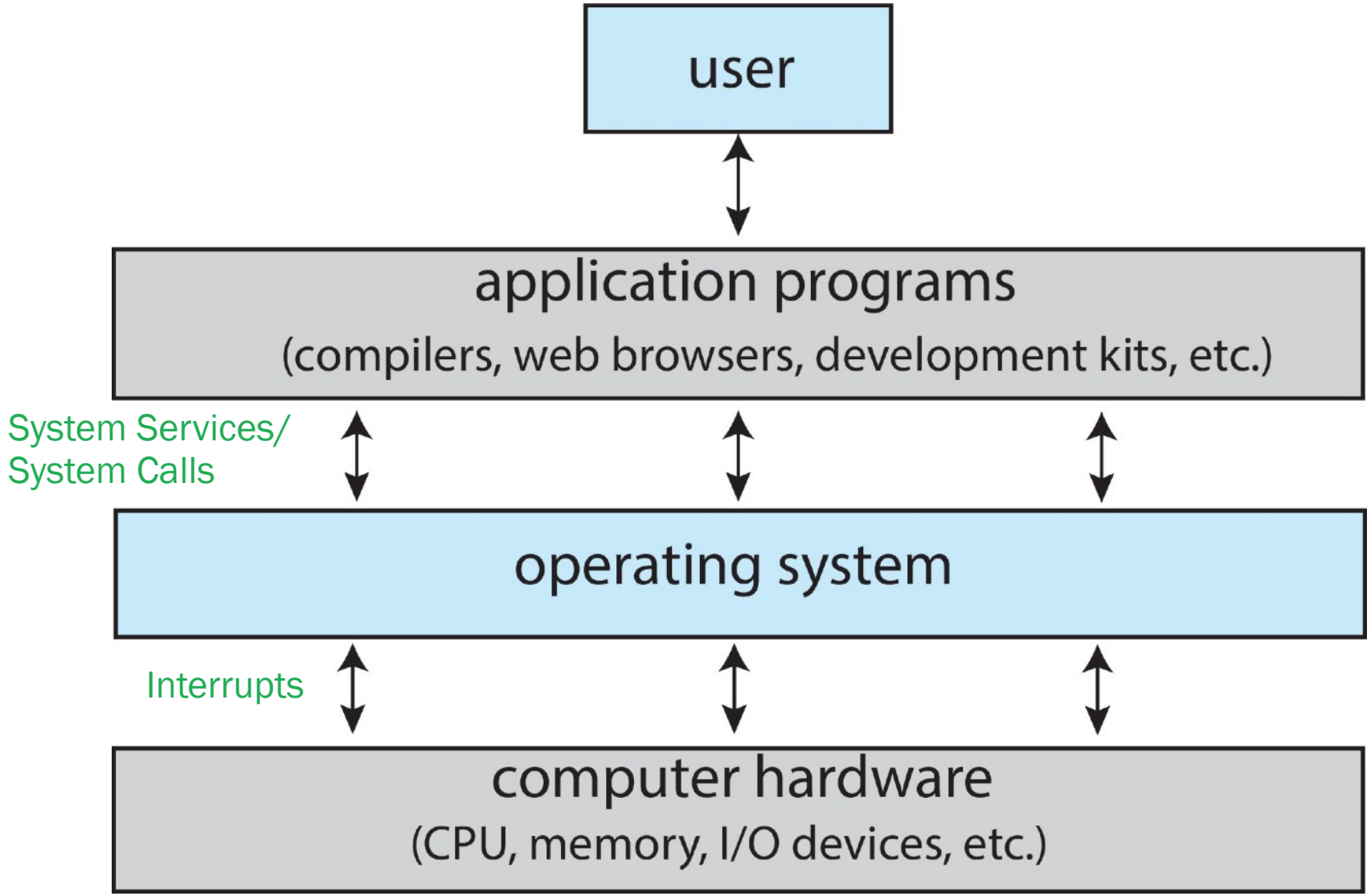
- Distributed systems: number of computers or processors are connected via network to increase the computation capacity.
  - **Example**: Google search services use thousands of computers, not a single one, to serve the world. Not one computer for each request. Multiple computers for each search request.
- The operating system for distributed computing systems can be
  - Network operating systems,
  - Distributed operating systems, or
  - Middleware

# Types of OSs for distributed systems

| System | Description | Main Goal |
|---|---|---|
| DOS | Tightly-coupled operating system for multi-processors and homogeneous multicomputers | Hide and manage hardware resources |
| NOS | Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN) | Offer local services to remote clients |
| Middleware | Additional layer atop of NOS implementing general-purpose services | Provide distribution transparency |

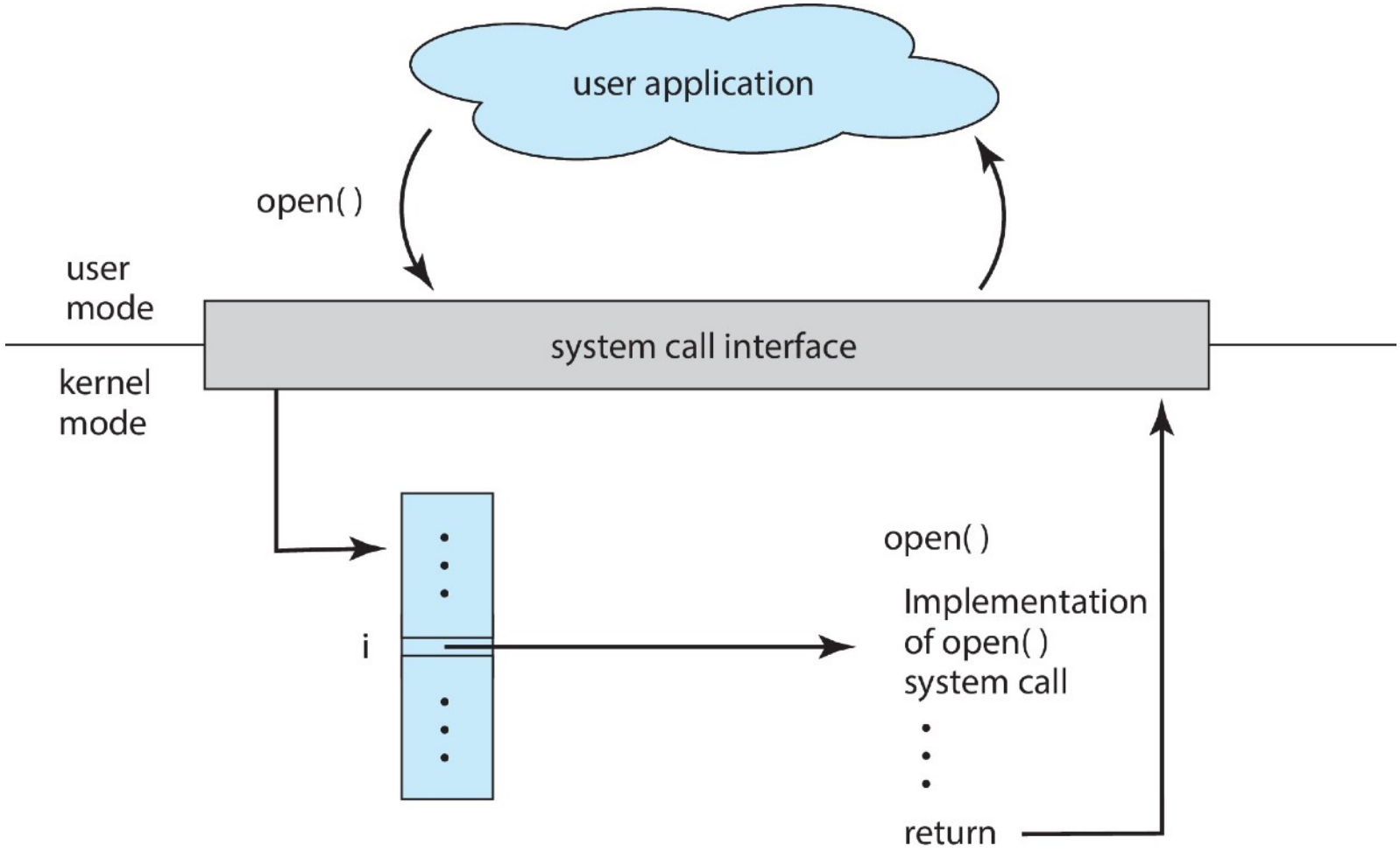
# Interface with Operating Systems

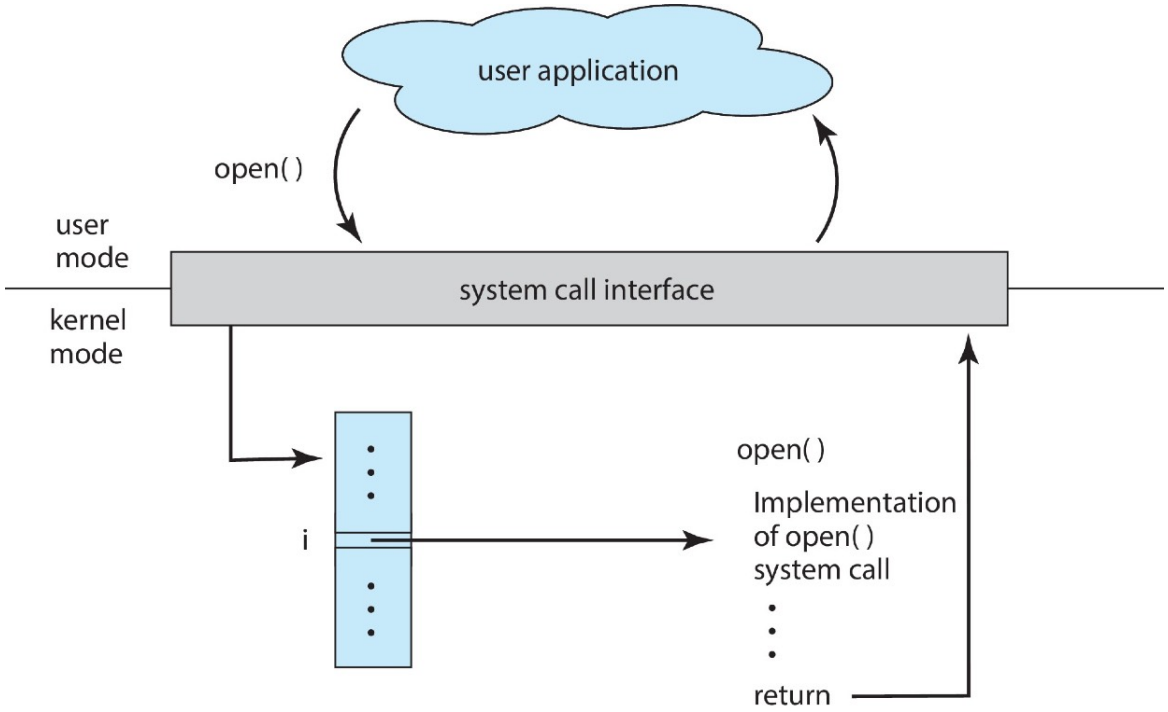
# Interface with Operating Systems



# System Services/System Calls

# API – System Call – OS Relationship






- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use
- Three most common APIs are
  - Win32 API for Windows,

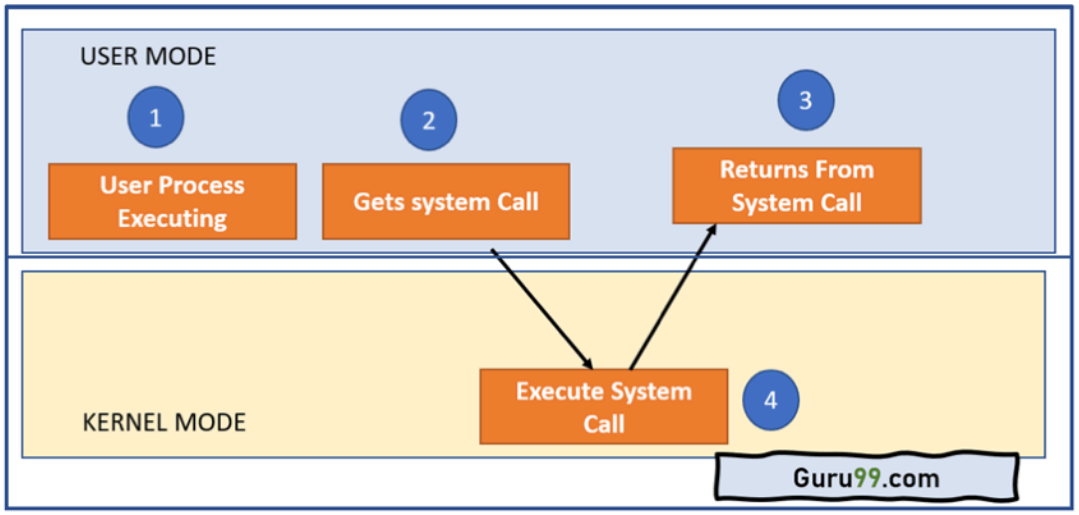
# System Calls

- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level Application Programming Interface (API) rather than direct system call use
- Three most common APIs are
  - Win32 API for Windows,
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X), and
  - Java API for the Java virtual machine (JVM)

Note that the system-call names used throughout this text are generic

# System Call Implementation

- Typically, a number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers
  - i386: http://asm.sourceforge.net/syscall.html (190)
  - x86: arch/x86/entry/syscalls/syscall_64.tbl (334)
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values



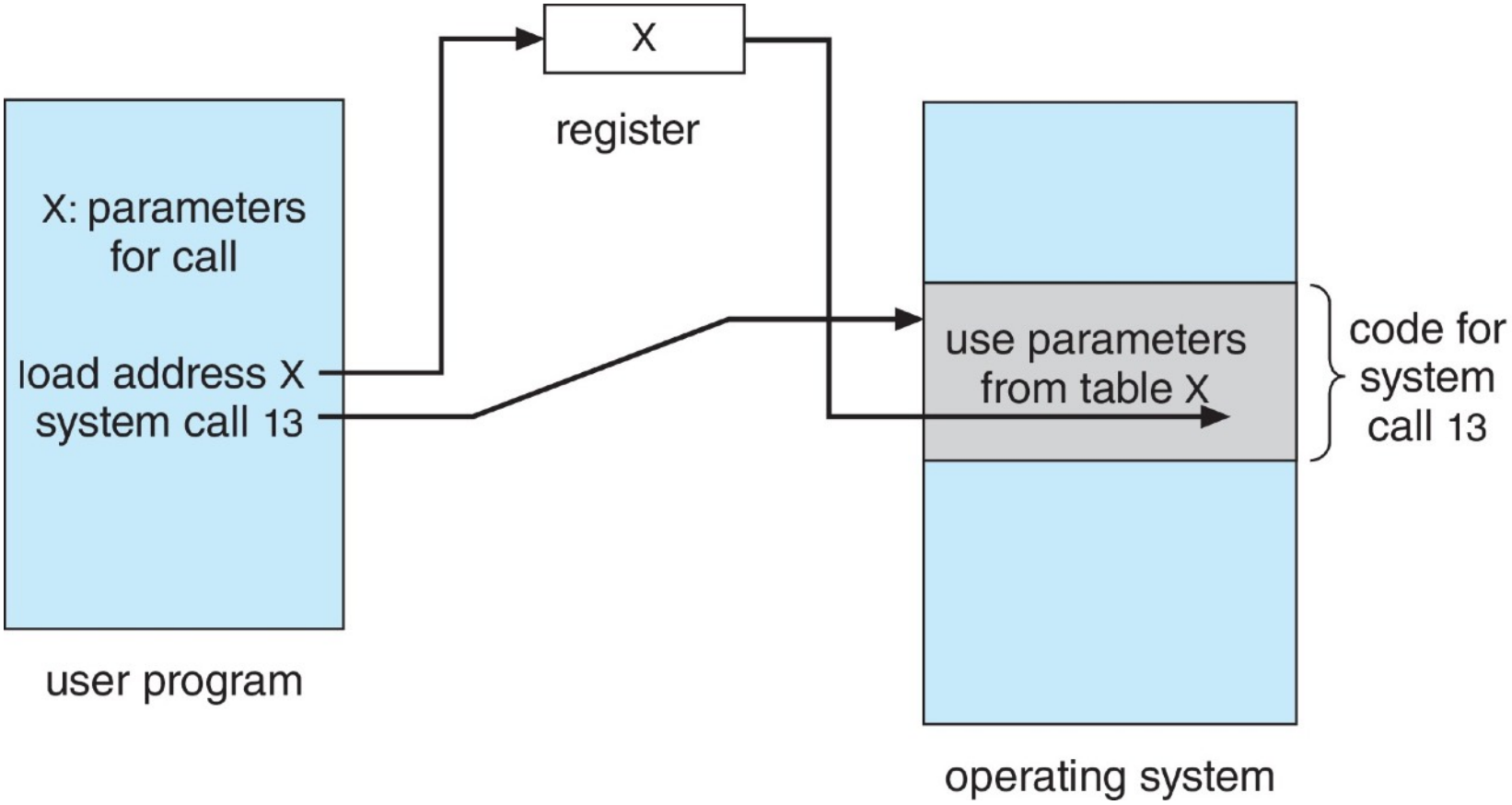
# System Call Implementation

- Typically, a number is associated with each system call
  - System-call interface maintains a table indexed according to these numbers
  - i386: http://asm.sourceforge.net/syscall.html (190)
  - x86: arch/x86/entry/syscalls/syscall_64.tbl (334)
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented
  - Just needs to obey API and understand what OS will do as a result call
  - Most details of OS interface hidden from programmer by API
    - Managed by run-time support library (set of functions built into libraries included with compiler)

# System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers
    - In some cases, may be more parameters than registers
  - Parameters stored in a block, or table, in memory, and address of block passed as a parameter in a register
    - This approach taken by Linux and Solaris
  - Parameters placed, or pushed, onto the stack by the program and popped off the stack by the operating system
- Block and stack methods do not limit the number or length of parameters being passed

# Parameter Passing via Table



# System Calls to Serve

- Process control
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - Debugger for determining bugs, single step execution
  - Locks for managing access to shared data between processes

# Types of System Calls (Cont.)

- File management
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- Device management
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices

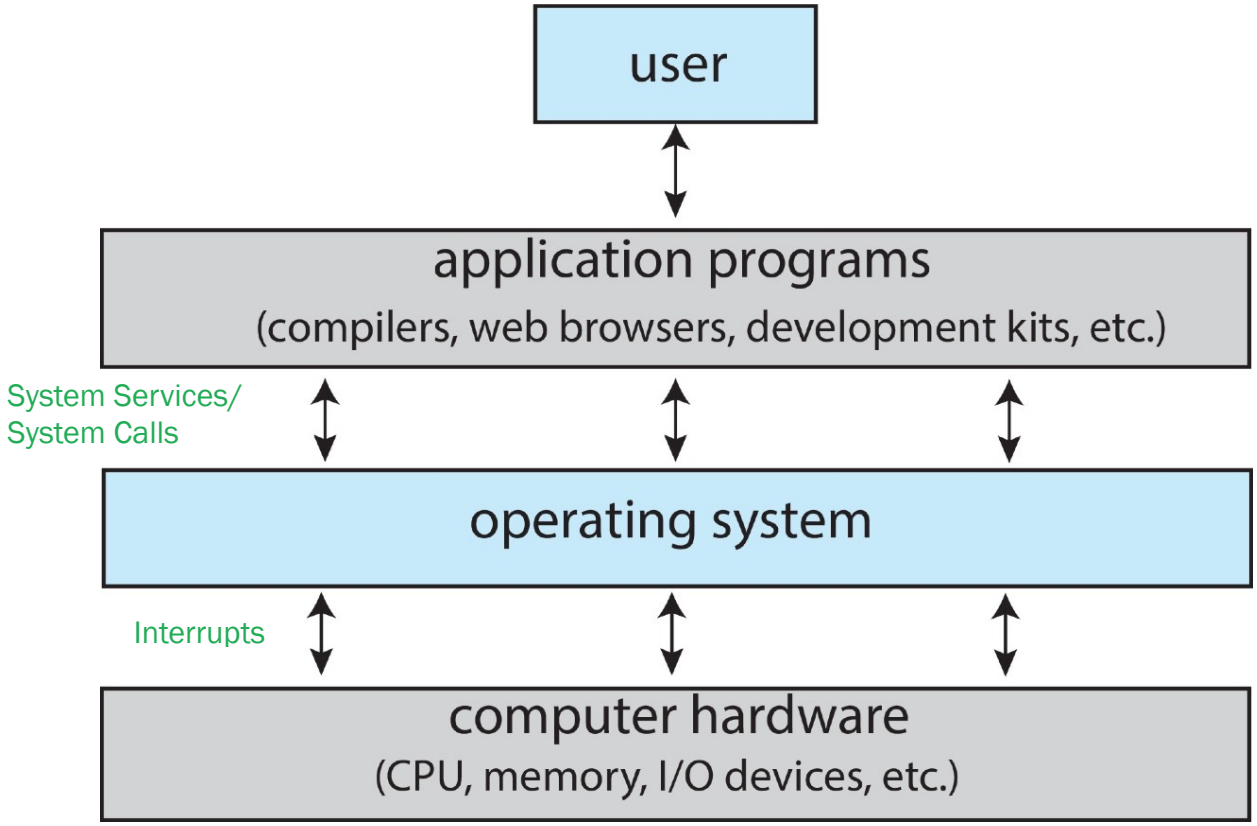
# Types of System Calls (Cont.)

- Information maintenance
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- Communications
  - create, delete communication connection
  - send, receive messages if message passing model to host name or process name
    - From client to server
  - Shared-memory model create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

# Types of System Calls (Cont.)

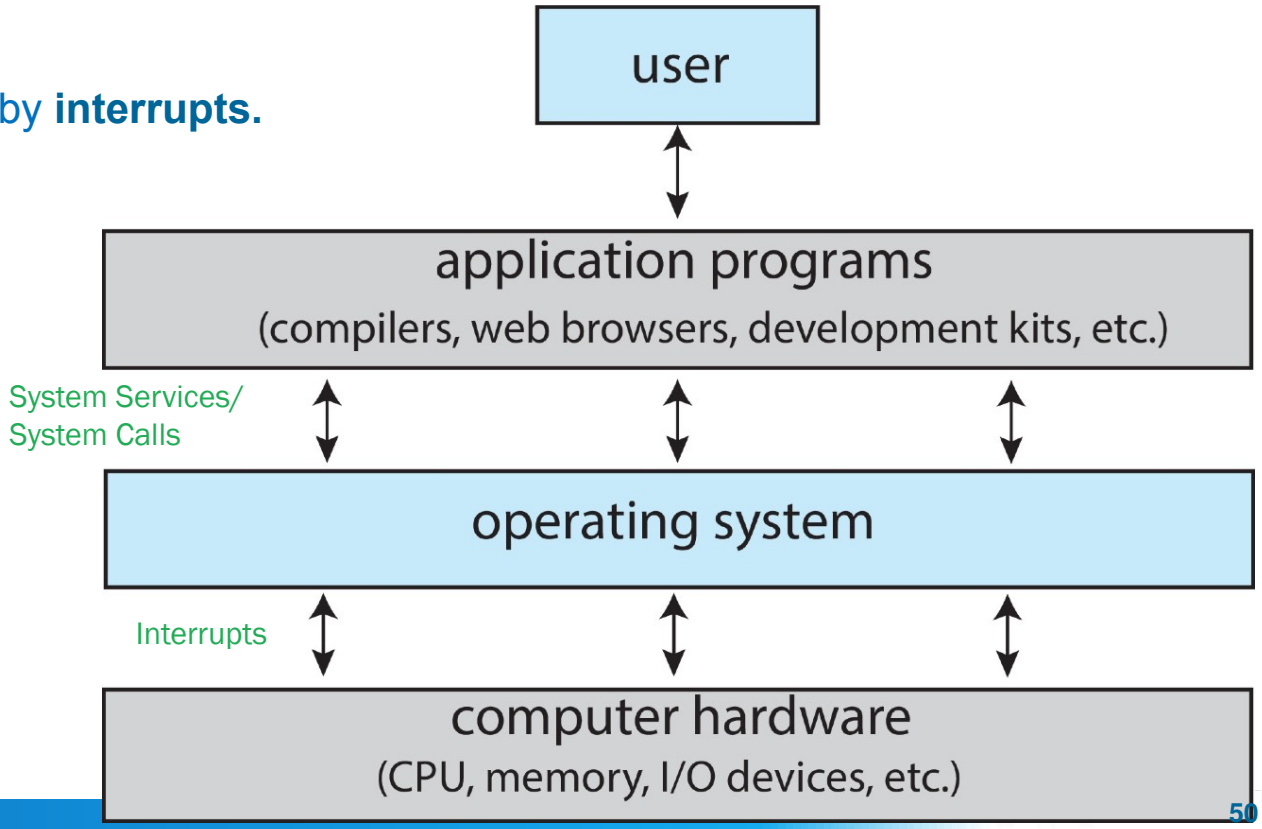
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

# Interrupts

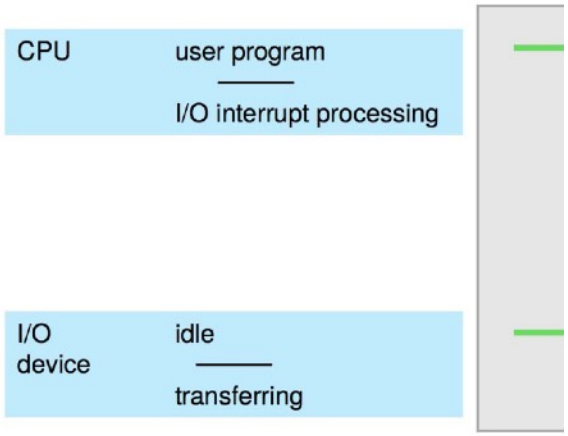

user
application programs
(compilers, web browsers, development kits, etc.)
System Services/
System Calls
operating system
Interrupts
computer hardware
(CPU, memory, I/O devices, etc.)

# Common Functions of Interrupts

- Interrupt transfers control to the interrupt service routine generally, through the **interrupt vector**, which contains the addresses of all the service routines
- Interrupt architecture must save the address of the interrupted instruction
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request
- Operating system is driven by **interrupts.**

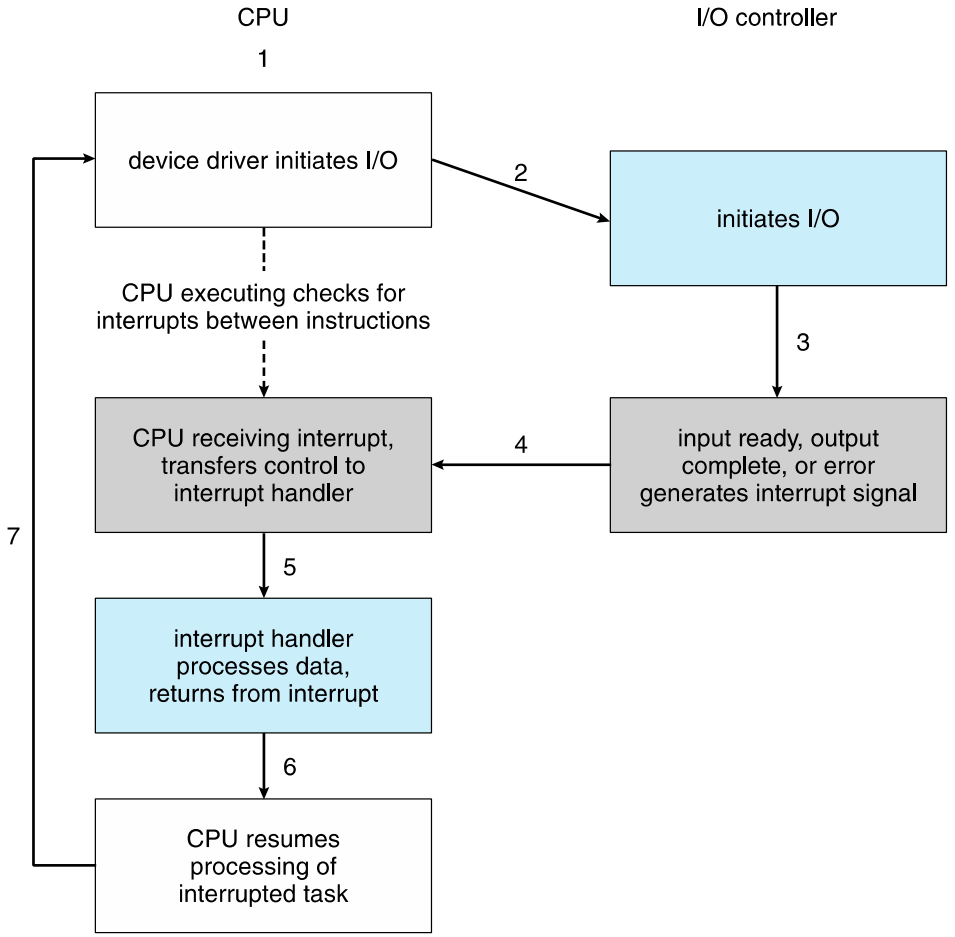


# Interrupt Timeline



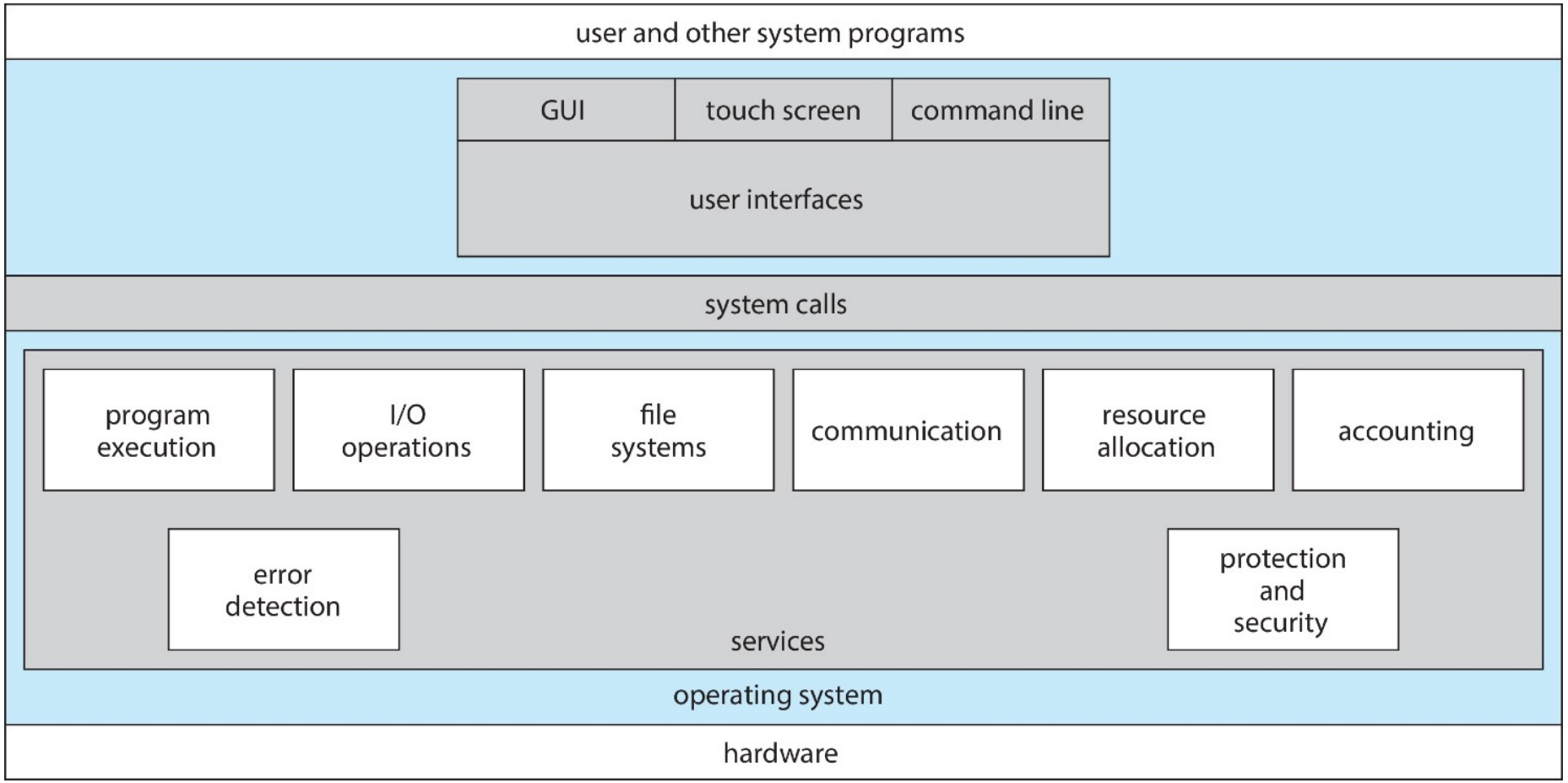
# Interrupt Handling

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

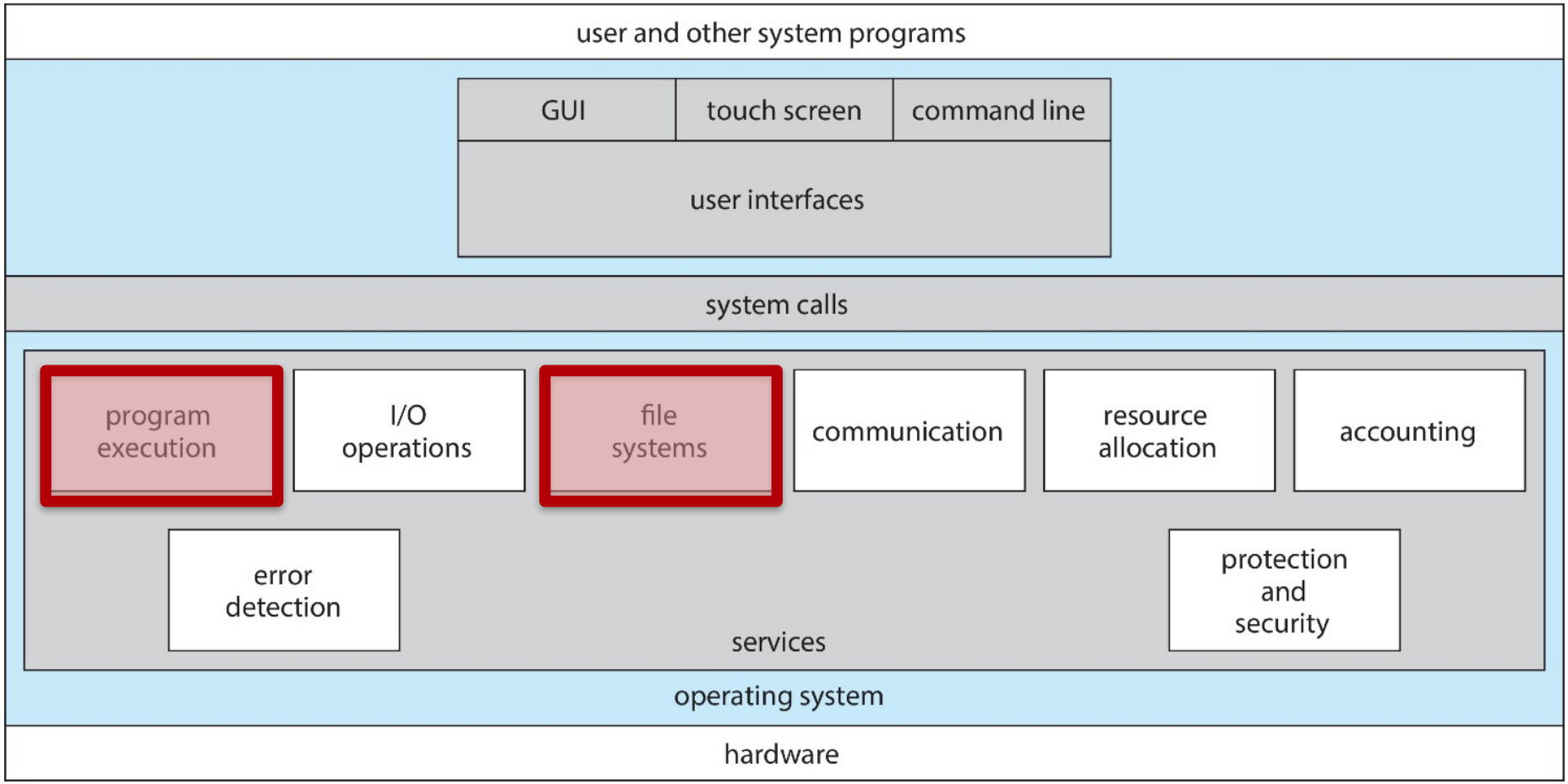


# Functional Components in Operating Systems

# Functional Components in Operating Systems

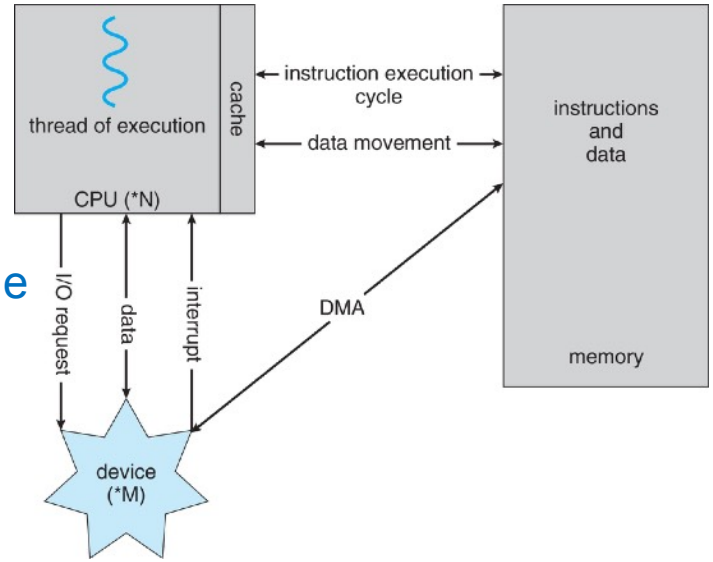


# Functional Components in Operating Systems

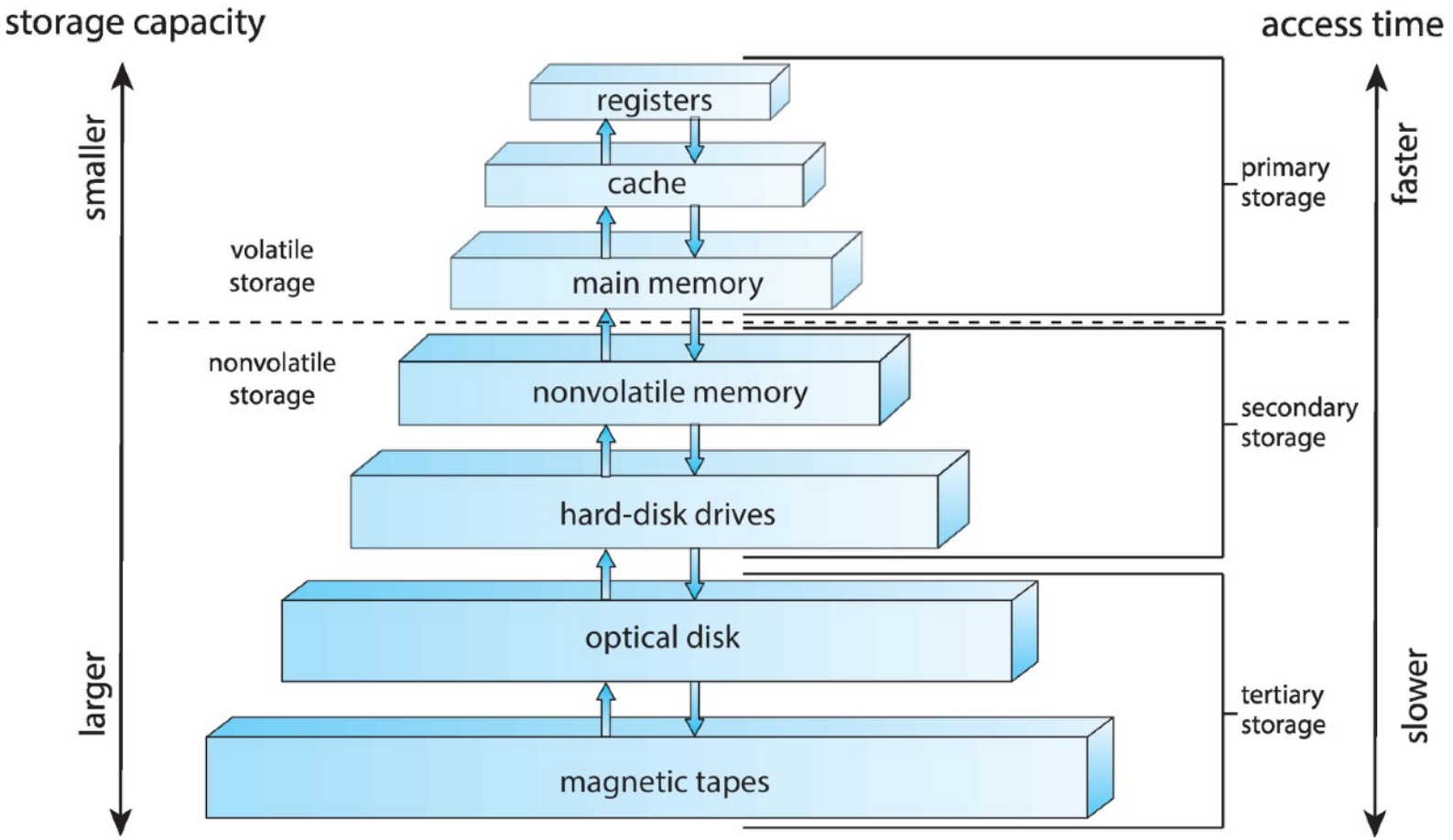


# Storage Sub-Systems

- Storage sub-systems serve the processors to store programs and data.
- Due to the impedance mismatch, multiple types of storage devices are deployed to provide storage services and form the storage sub-systems:
  - Primary storage (main memory) – only large storage media that the CPU can access directly
    - Random access and volatile
    - Typically random-access memory in the form of Dynamic Random-access Memory (DRAM)
  - Secondary storage - Extension of main memory that provides large nonvolatile storage capacity



# Storage-Device Hierarchy



# Memory or Not Memory?

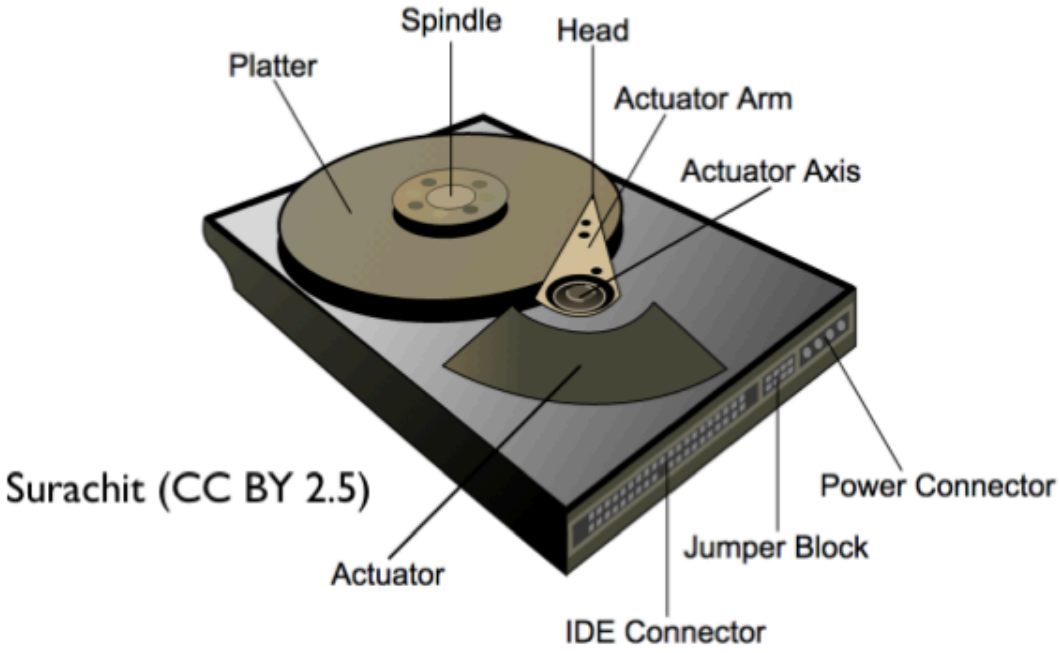
| | Volatile | | Non-Volatile | | | | | |
|---|---|---|---|---|---|---|---|---|
| | **SRAM** | **DRAM** | **HDD** | **NAND flash** | **STT-RAM** | **ReRAM** | **PCM** | **FeRAM** |
| **Cell size (*F*2)** | 120–200 | 60–100 | N/A | 4–6 | 6–50 | 4–10 | 4–12 | 6–40 |
| **Write endurance** | $10^{16}$ | > $10^{15}$ | >$10^{15}$ (mechanical parts) | $10^{4}$–$10^{5}$ | $10^{12}$–$10^{15}$ | $10^{8}$–$10^{11}$ | $10^{8}$–$10^{9}$ | $10^{14}$–$10^{15}$ |
| **Read latency** | ~ 0.2–2 ns | ~ 10 ns | 3–5 ms | 15–35 $\mu$s | 2–35 ns | ~ 10 ns | 20–60 ns | 20–90 ns |
| **Write latency** | ~ 0.2–2 ns | ~ 10 ns | 3–5 ms | 200–500 $\mu$s | 3–50 ns | ~ 50 ns | 20–150 ns | 50–75 ns |
| **Leakage power** | high | medium | (mechanical parts) | low | low | low | low | low |
| **Dynamic energy (Read/Write)** | low | medium | (mechanical parts) | low | low/high | low/high | medium/high | low/high |
| **Maturity** | mature | mature | mature | mature | prototypes | prototypes | prototypes | industrialized |

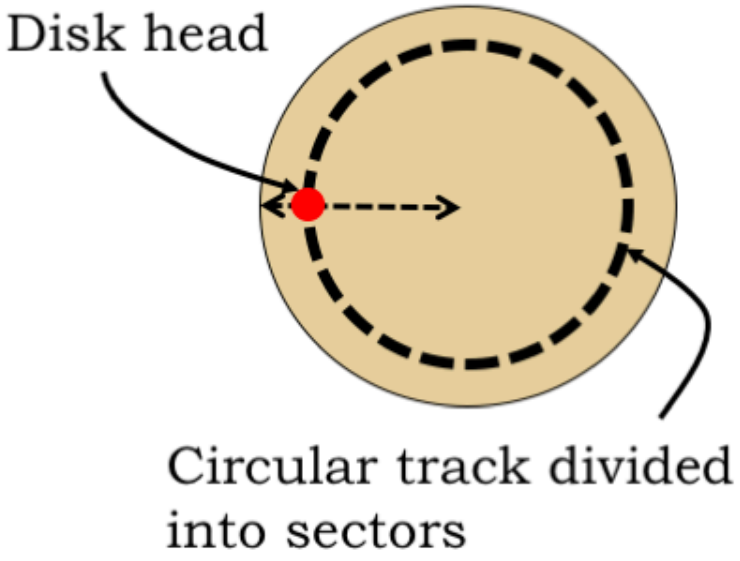
# Storage Structure - Secondary Storage

Extension of main memory that provides large nonvolatile storage capacity

- Secondary storage
  - Hard Disk Drives (HDD) – rigid metal or glass platters covered with magnetic recording material
    - Disk surface is logically divided into **tracks**, which are subdivided into **sectors**
    - The **disk controller** determines the logical interaction between the device and the computer
  - **Non-volatile memory** (**NVM**) devices– faster than hard disks, nonvolatile
    - Various technologies
    - Becoming more popular as capacity and performance increases, price drops

# Non-Volatile Storage: Hard Disk



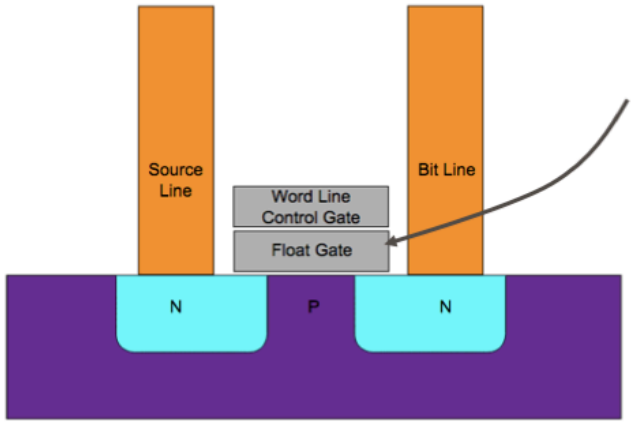




Hard Disk: Rotating magnetic platters + read/write head

- Extremely slow (~10ms): Mechanically move head to position, wait for data to pass underneath head
- ~100MB/s for sequential read/writes
- ~100KB/s for random read/writes
- Cheap

# Non-Volatile Storage: flash memory

- 1960': Dawon Kahng and Simon Sze (施敏，NTU BS '57, Stanford PhD '63) proposed the first floating gate device in 1967 at Bell Lab, around the same time when IBM invented dynamic random access memory (DRAM) cells.
- 1980: Dr. Fujio Masuoka from Toshiba defined the word 'flash' in 1984, which caught the attention of Intel.
  - Intel introduced the first commercial NOR type flash chip in 1988.
  - NAND flash from Samsung and Toshiba followed in 1989.

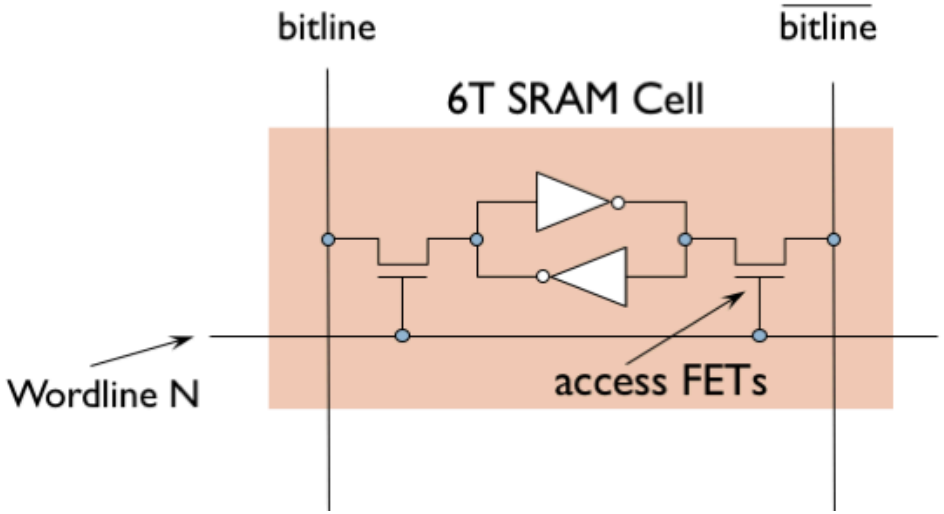


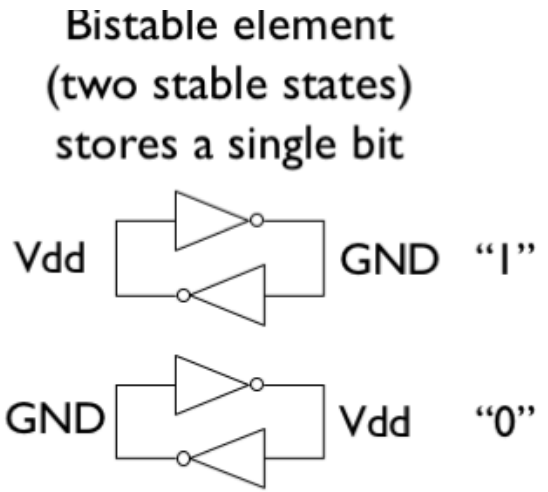

Electrons here diminish strength of field from control gate ⇒ no inversion ⇒ NFET stays off even when word line is high.

Cyferz (CC BY 2.5)

# Static Random Access Memory (SRAM)

- Static random-access memory (static RAM or SRAM) is a type of random-access memory (RAM) that uses latching circuitry (flip-flop) to store each bit.
- The data remains as long as the power is supplied.
- 6-MOSFET (6T) Cell as an example:
  - Two CMOS inverters (4 MOSFETs) forming a bistable element.
  - Two access transistors.

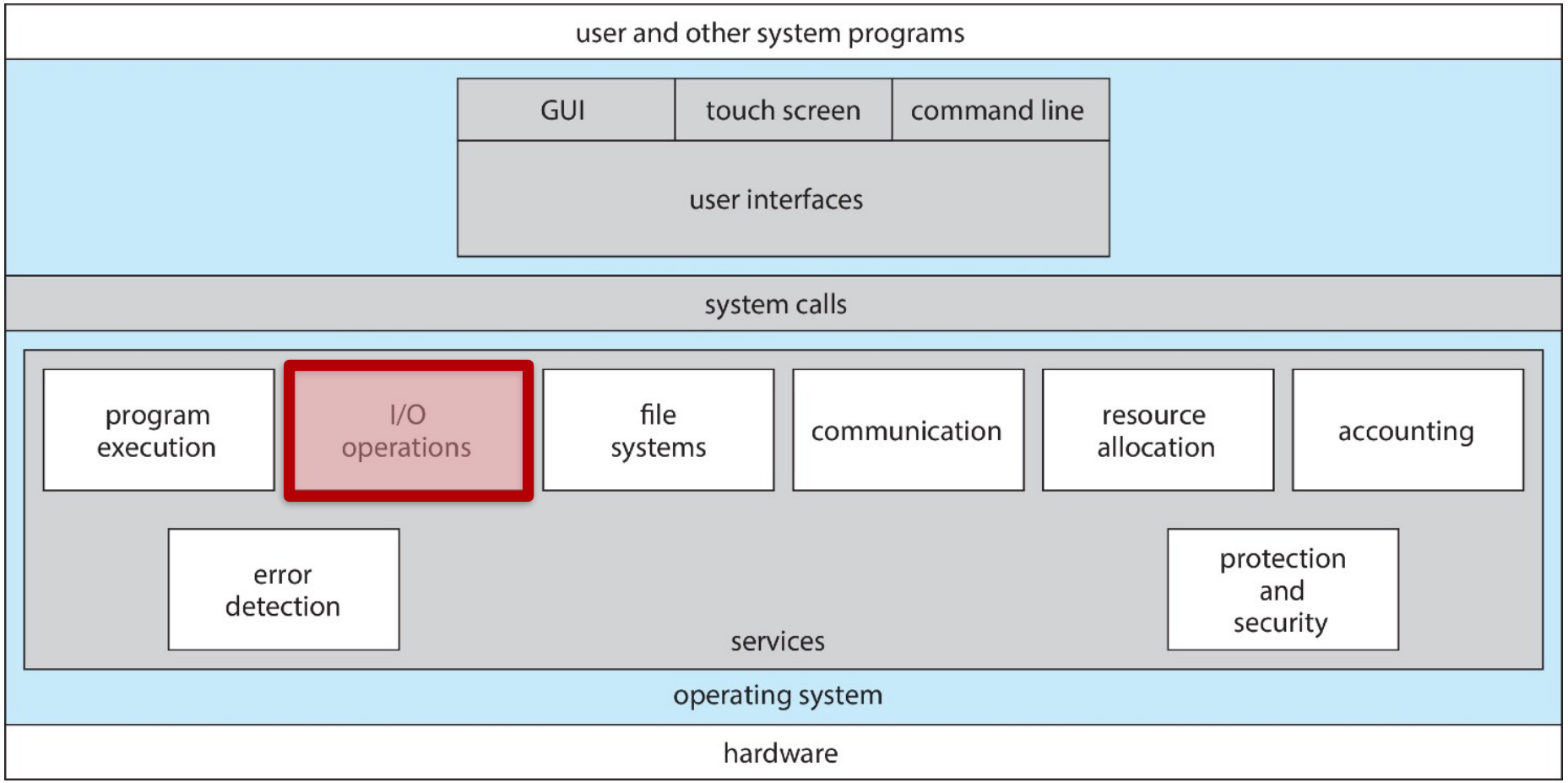


# Dynamic Random Access Memory (DRAM)

- DRAM is a type of random-access semiconductor memory that stores each bit of data in a memory cell consisting of a tiny capacitor and a transistor, both typically based on metal-oxide-semiconductor (MOS) technology.
- The capacitor can either be charged or discharged: conventionally called 0 and 1.
- The electric charge on the capacitors slowly leaks off, so without intervention the data on the chip would soon be lost. To prevent this, DRAM requires an external memory refresh circuit which periodically rewrites the data in the capacitors, restoring them to their original charge.

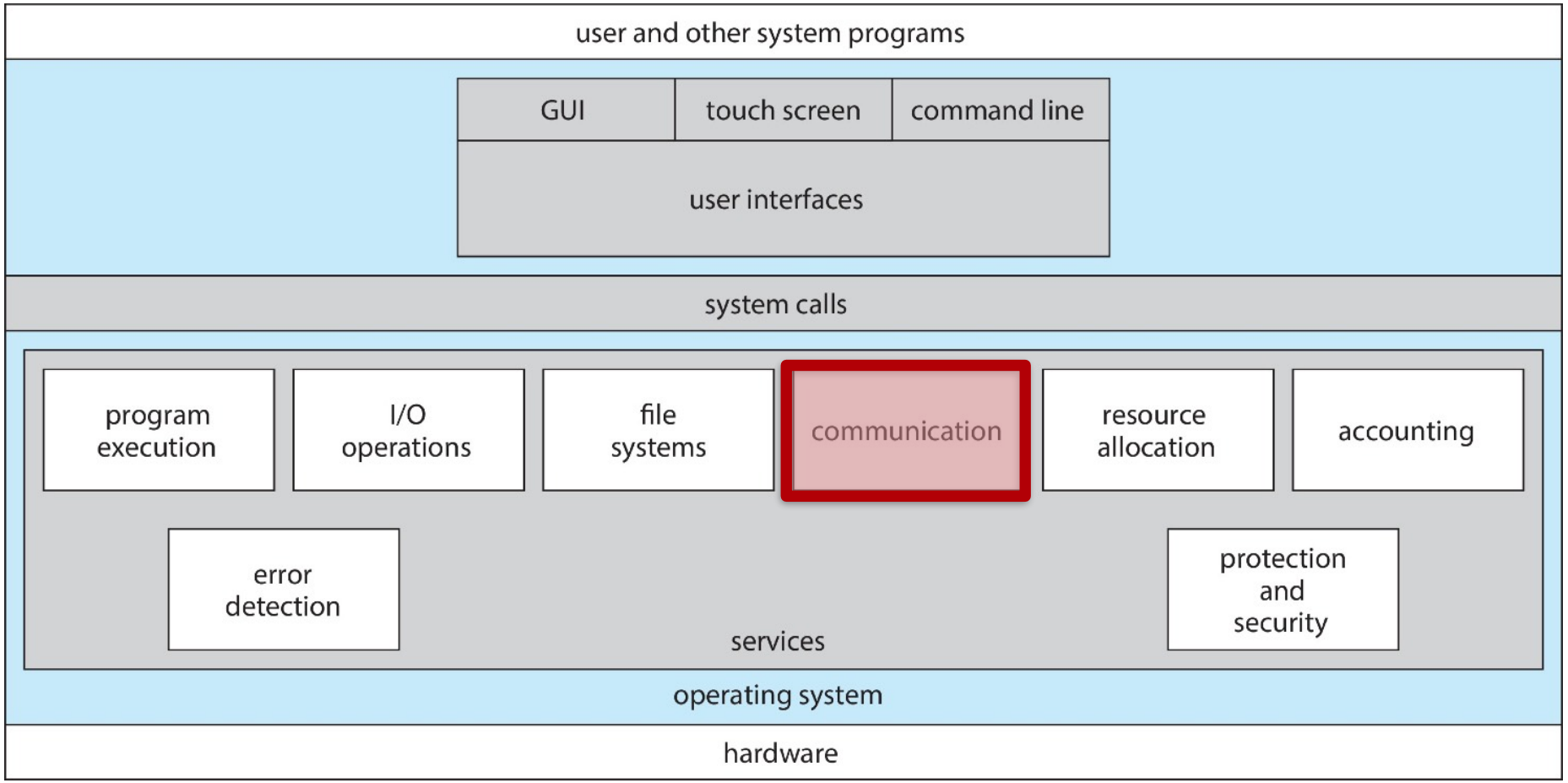
# Functional Components in Operating Systems



# I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including buffering (storing data temporarily while it is being transferred), caching (storing parts of data in faster storage for performance), spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

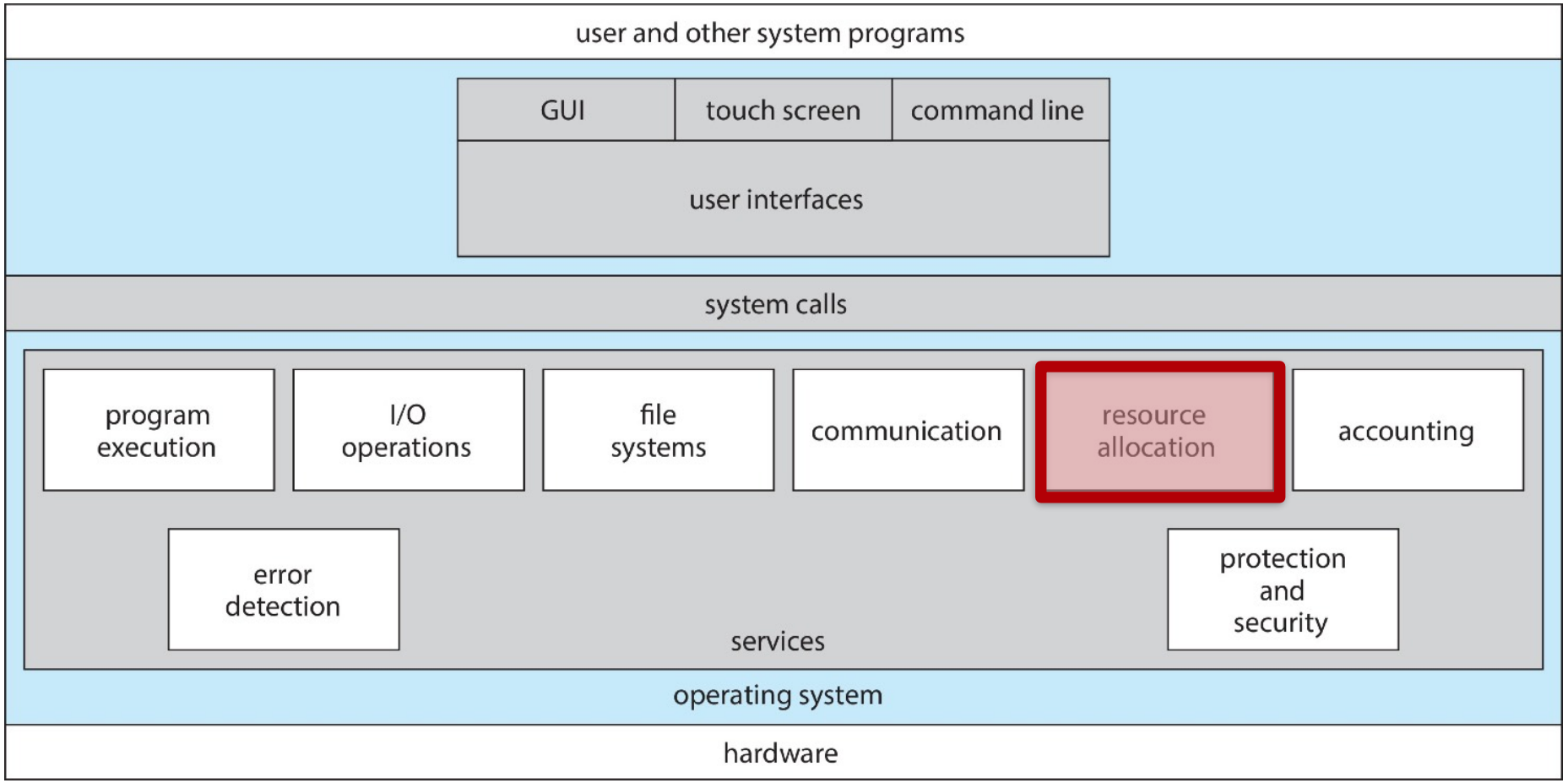
# Functional Components in Operating Systems



# Communications

- Communications - Provide the mechanism for creating virtual connections among processes, users, and computer systems
- Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another
- Operations:
  - create, delete communication connection
  - send, receive messages if message passing model to host name or process name
    - From client to server
- Shared-memory model create and gain access to memory regions
- Transfer status information
- Attach and detach remote devices

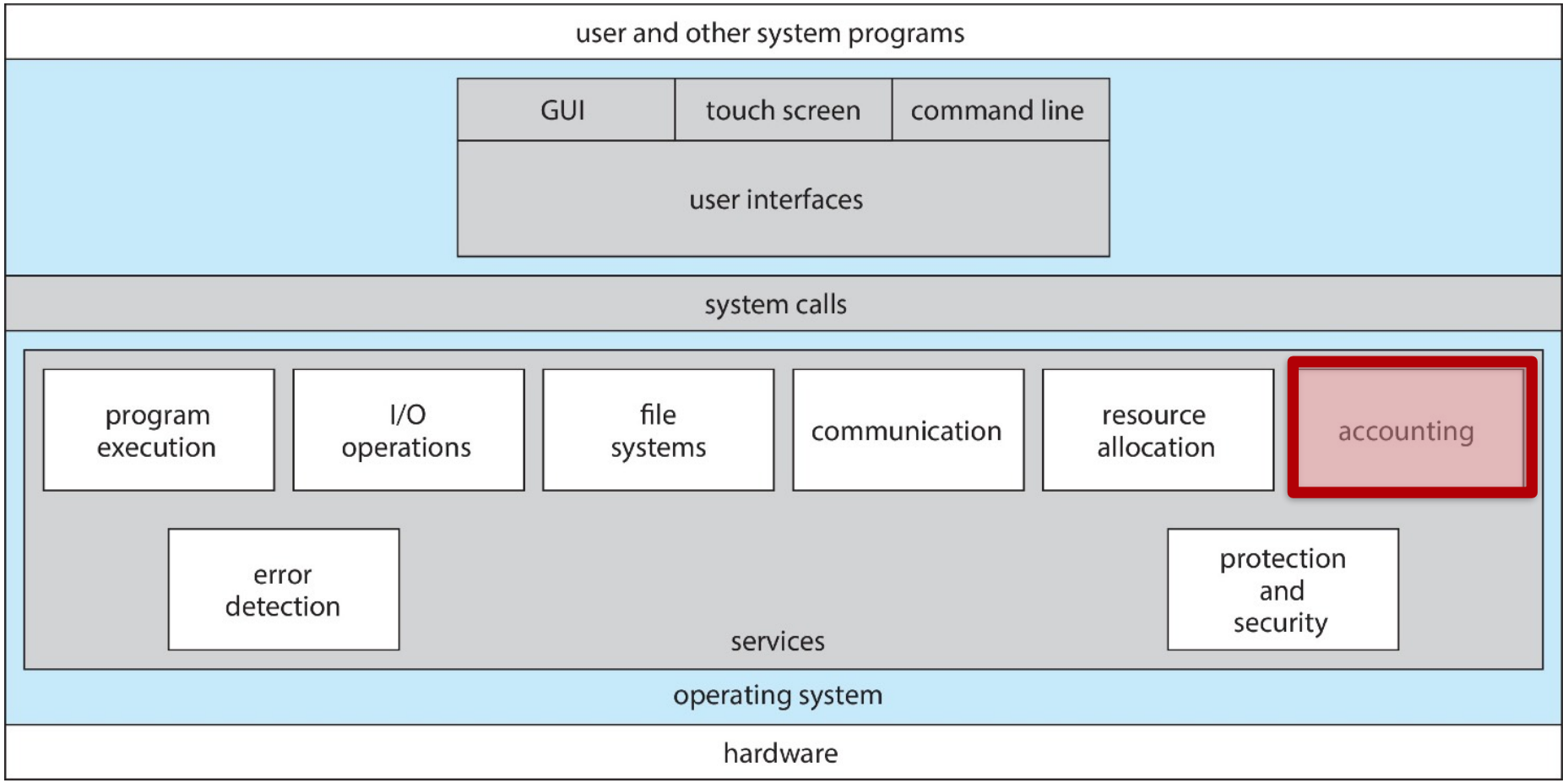
# Functional Components in Operating Systems



# Resource Allocation

- When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
  - Many types of resources:
    - Physical Resources: file storage, I/O devices, etc.
    - Logical Resources: - CPU cycles, main memory, process ID, user ID, file locks, etc.
- Resources can be assigned mostly temporarily or permanently.
  - Memory pages and CPU cycles are assigned temporarily.
  - Secondary storage can be assigned permanently.
- NTU Parking Permission is an allocation but does not have a parking space assignment.

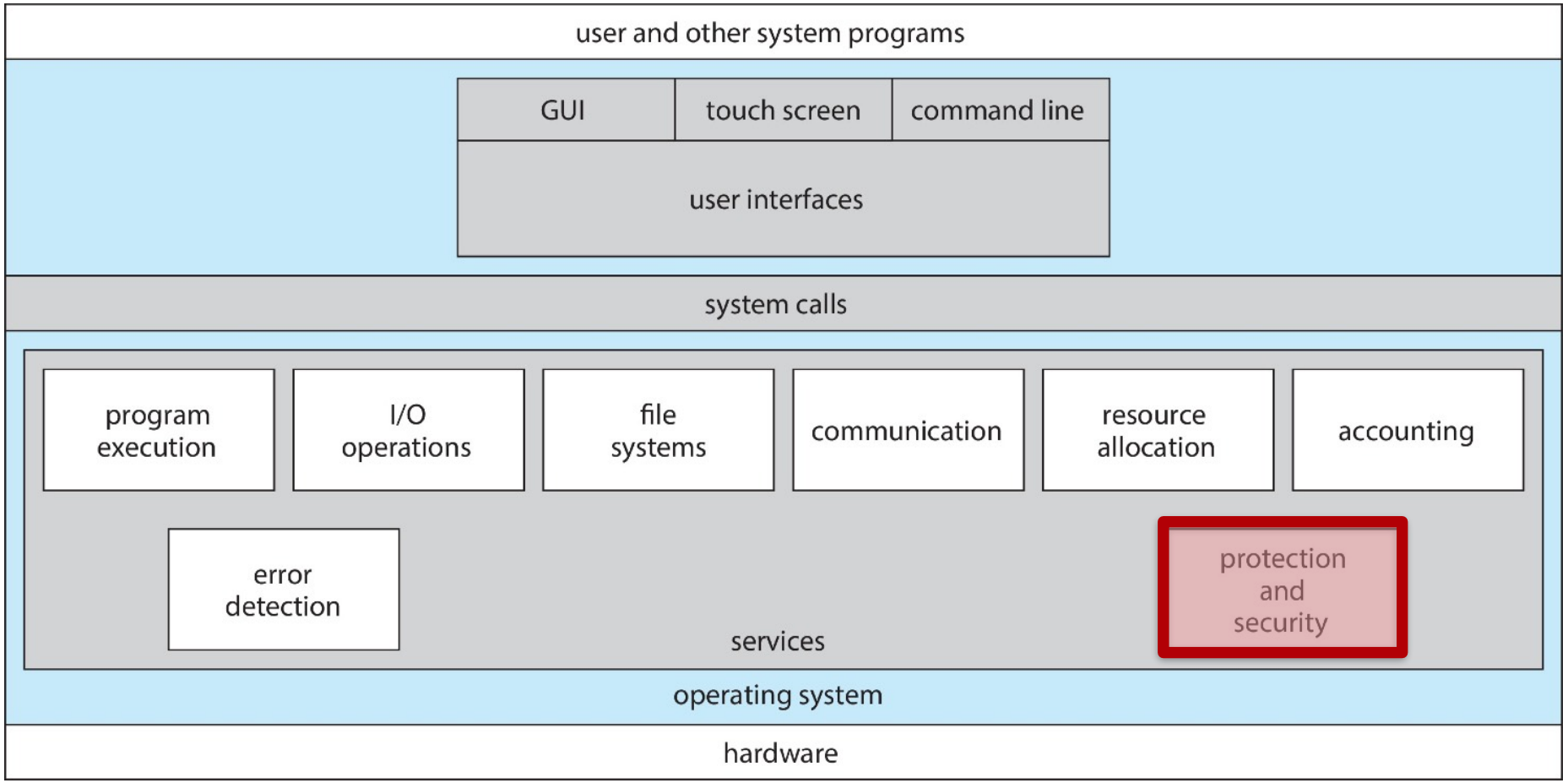
# Functional Components in Operating Systems



# Accounting

- Accounting: The Operating System tracks an account of all the functionalities taking place in the computer system at a time. All the details such as the types of errors occurred are recorded by the Operating System.
- Purpose of accounting:
  - Control and limit who can use the computers.
  - Track and control how much resources are used.
  - Log the errors if any
  - Charges based on the usage: Computing as a Utility
    - Virtual machine on Amazon, Microsoft Azure, or 國家高速電腦中心 are charged based on the resources and time.
    - The computers can be shared by users and accounts.
    - Users do not need to pay for the whole computer.

# Functional Components in Operating Systems



# Protection and Security

- Protection – any mechanism for controlling access of processes or users to resources defined by the OS
- Security – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (user IDs, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (group ID) allows set of users to be defined and controls managed, then also associated with each process, file
  - Privilege escalation allows user to change to effective ID with more rights

# Any Questions?

# See You
# Next Class