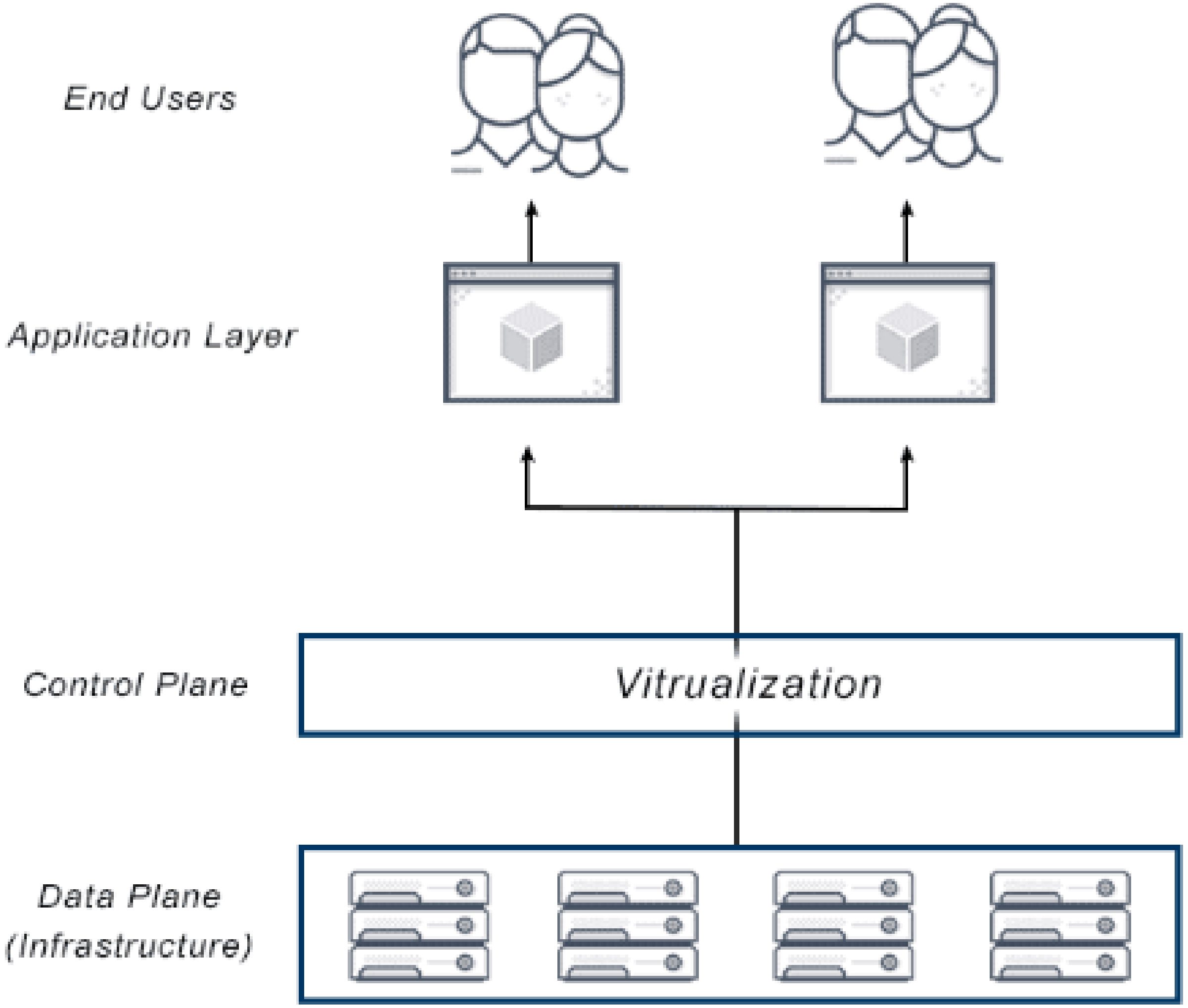
240-229: SDA (Operating Systems session)

Lecture 1: Introduction

Associate Professor Dr. Sangsuree Vasupongayya

Software defined architecture (SDA)

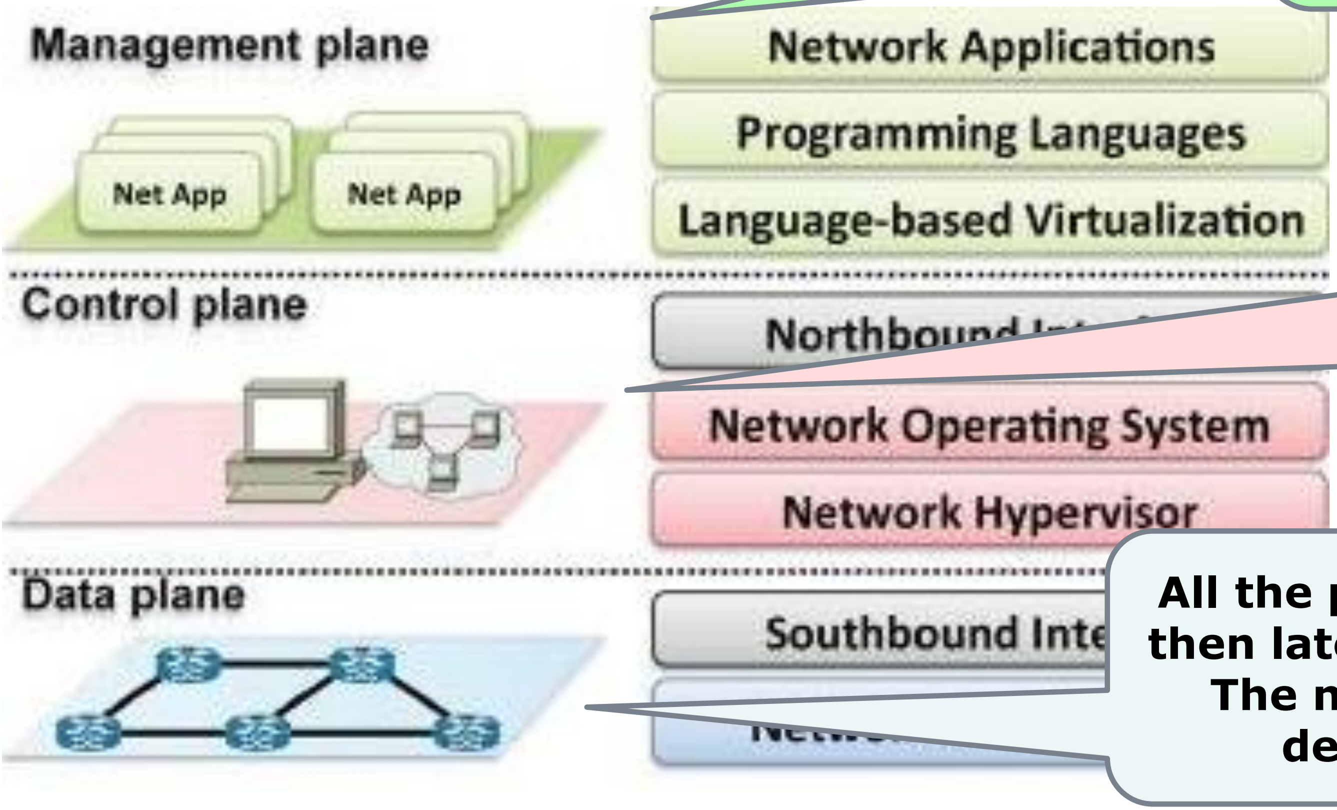
Software defined architecture (SDA) **provides a layer of virtualization between the software and its users**, which connects users to a simple dashboard that masks the complex systems operating in the background



https://avinetworks.com/wp-content/uploads/2019/06/software-defined-architecture-diagram.png

SDA

Software defined architecture (SDA) **provides a layer of virtualization between the software and its users**, which connects users to a simple dashboard that masks the complex systems operating in the background



The management plane looks after the network functions needed for the network to go on. These functions include functions such as firewall applications; load balancing, routing and monitoring of the data in the network

The logically abled controllers are set in this plane. This is considered as the "Network Brain". This is the most important part where all the logic is rested and forwarded to the entire network.

All the physical devices are placed in this layer and then later these are abstracted to the control plane. The network infrastructure has interconnected devices and comprises of the data plane.

Where is OS?

- Web Applications
  - Growing trend toward using applications that run on remote Internet servers instead of local PCs.

    e.g., Google docs, gmail, Wikipedia, Facebook
  - It’s all about “The Cloud”

Client Machines

Data Center

Categories of Software

Compilers and translator programs

- Enable programmers to create other software

Software applications

- Serve as productivity tools to help users solve problems

System software

- Coordinates hardware operations
- The Hardware-Software Connection
- Class of software that includes **the operating system** and utility programs
- Handles low-level details and hundreds of other tasks behind the scenes
- User does not need to be concerned about details

What the OS does

Every computer depends on an operating system to:

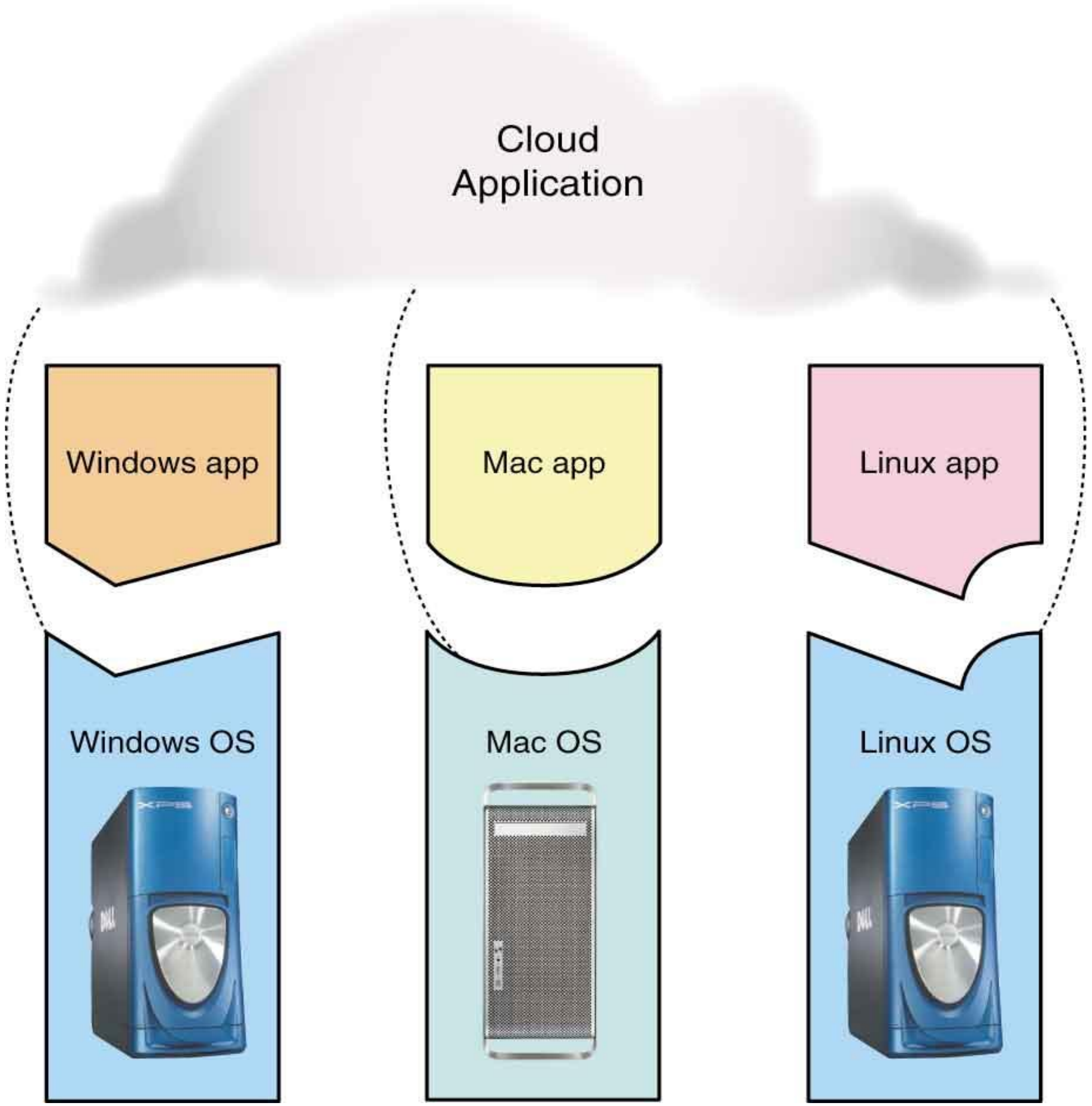
- Keep hardware running efficiently
- Maintains file system
- Supports multitasking
- Manages virtual memory

Operating system runs continuously when computer is on

What is an Operating System?

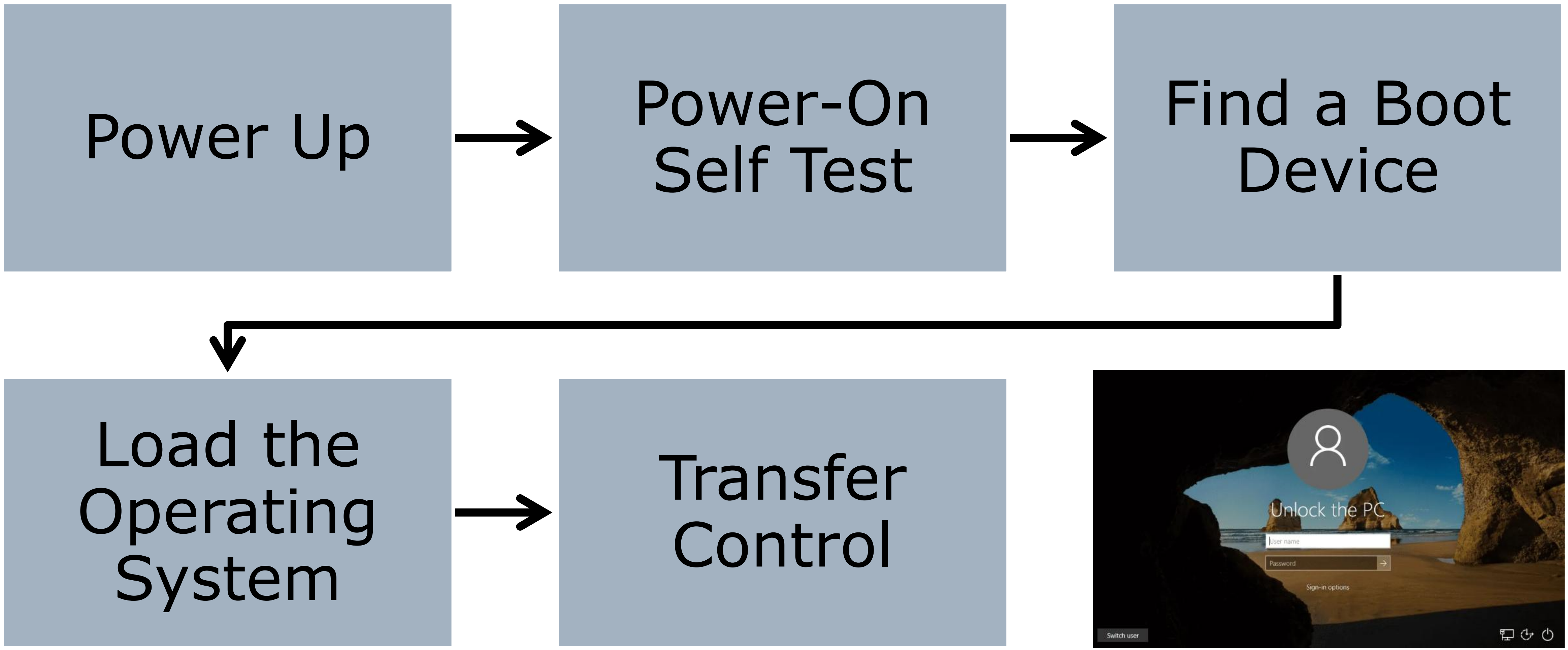
- □ A program that acts as an intermediary between a user of a computer and the computer hardware.
- □ Operating system goals:
  - Execute user programs and make solving user problems easier.
  - Make the computer system convenient to use.
- □ Use the computer hardware in an efficient manner.

Compatibility Issues



- Operating systems are designed to run on particular hardware platforms.
- Applications are designed to run on particular operating systems.
- Most cloud applications are designed to run on multiple platforms.

the Boot Sequence



Power Up → Power-On Self Test → Find a Boot Device → Load the Operating System → Transfer Control

Computer Startup

Operating system must be made available to hardware so hardware can start it

- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EEPROM, generally known as **firmware**
  - Initializes all aspects of the system
  - Loads operating system kernel and starts execution

Operating System Functions

OS is a **resource allocator**

- Manages all resources
- Decides between conflicting requests for efficient and fair resource use

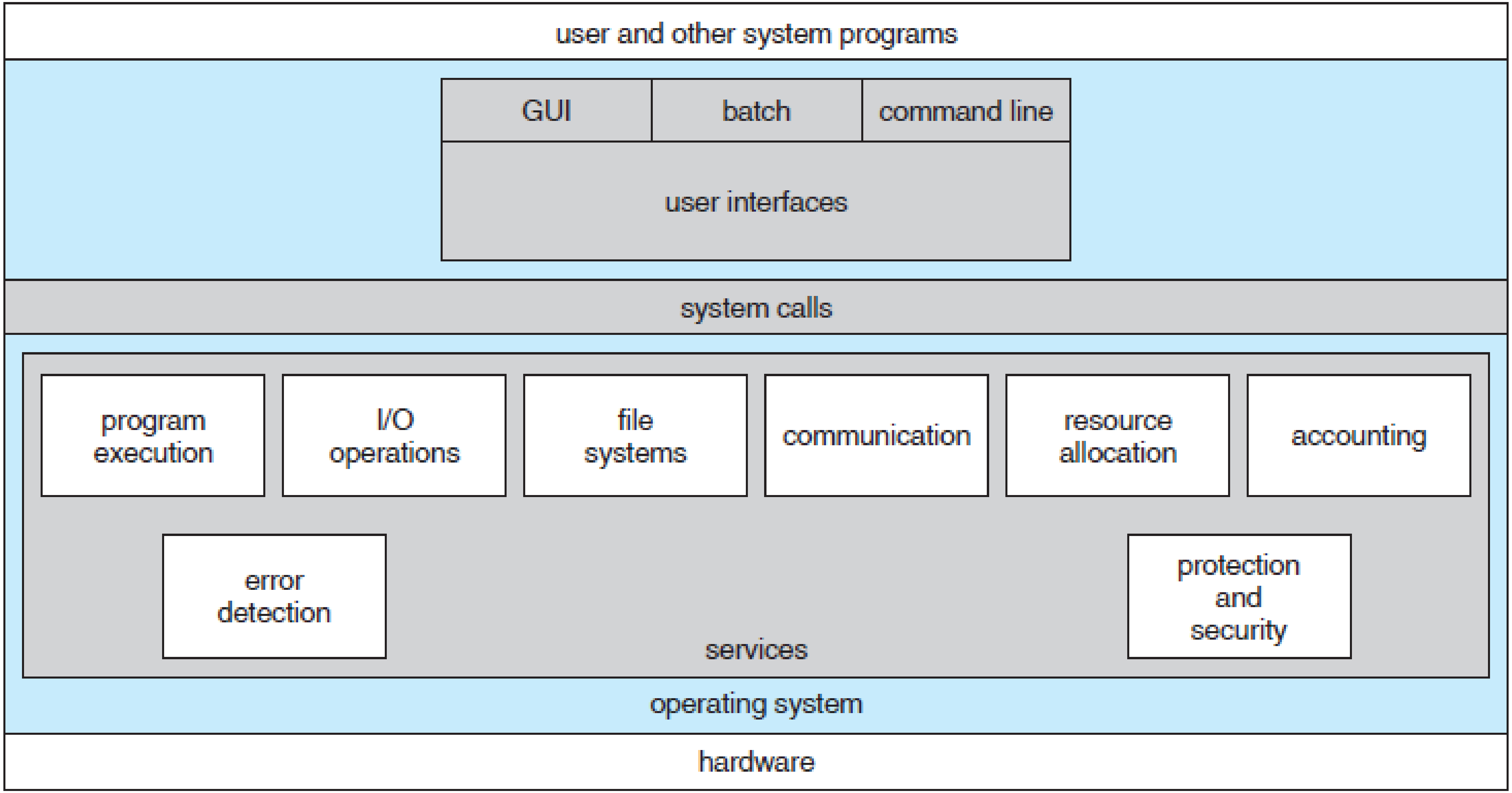
OS **controls** how programs execute

- Controls execution of programs to prevent errors and improper use of the computer

Operating System Functions

Processor management	Process / Thread Scheduling
Main memory management	Main memory Virtual memory
Device management	
File management	
Network management	
User Interface	
Cooperation issues	Synchronization / Deadlock

Services of Operating systems



Manage: I/O devices

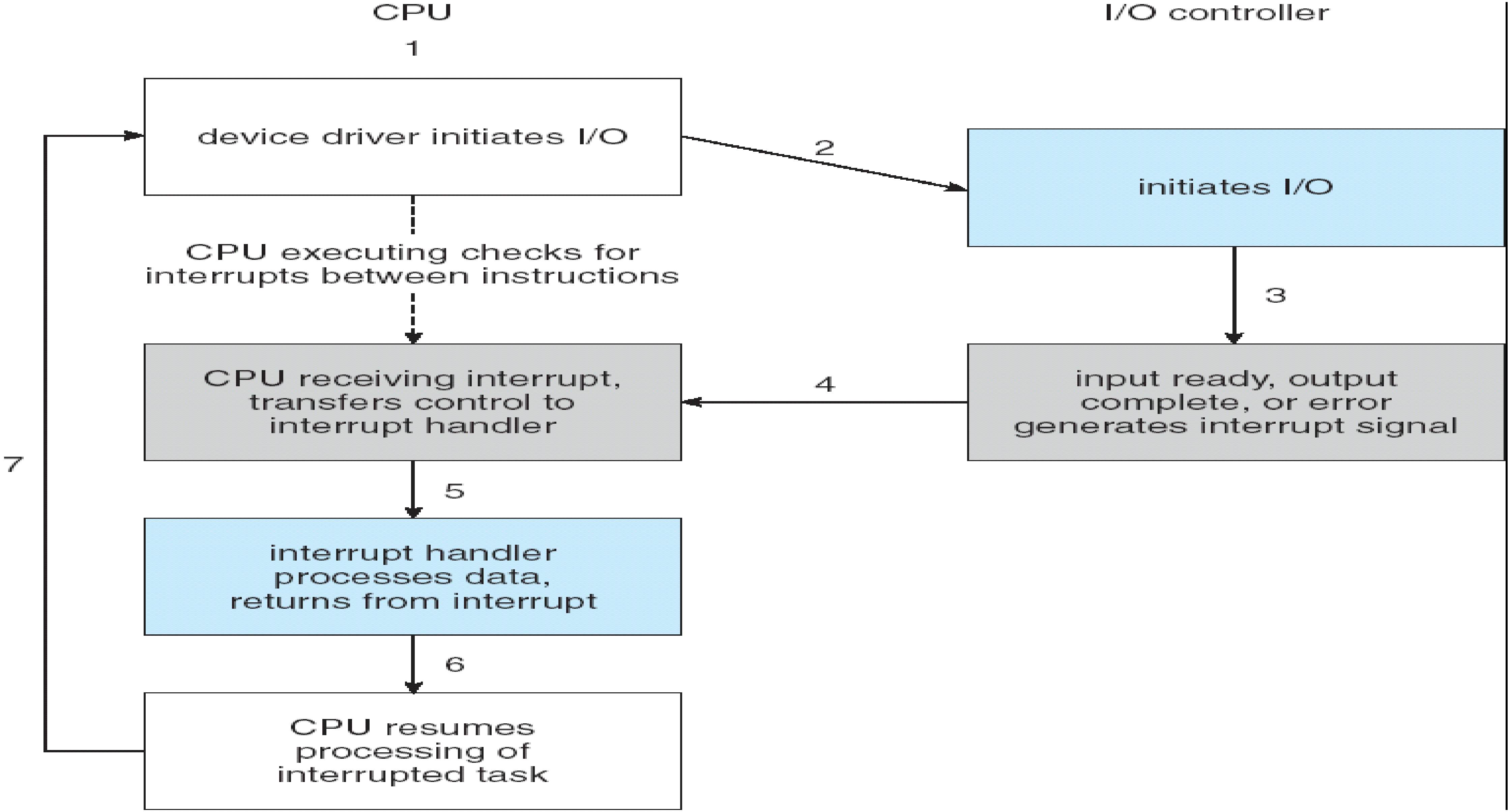
I/O devices and the CPU can execute concurrently.

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*.

Interrupt-Driven I/O Cycle


CPU
I/O controller
1
device driver initiates I/O
2
initiates I/O
CPU executing checks for interrupts between instructions
3
CPU receiving interrupt, transfers control to interrupt handler
4
input ready, output complete, or error generates interrupt signal
5
interrupt handler processes data, returns from interrupt
6
CPU resumes processing of interrupted task
7

Interrupts

- CPU **Interrupt-request line** triggered by I/O device
- **Interrupt handler** receives interrupts
- **Maskable** to ignore or delay some interrupts
- Interrupt vector to dispatch interrupt to correct handler
  - Based on priority
  - Some **nonmaskable**
- Interrupt mechanism also used for exceptions

Operating-System Operations

- Interrupt driven by hardware
- Software error or request creates **exception** or **trap**
  - Division by zero, request for operating system service
- Other process problems include infinite loop, processes modifying each other or the operating system

Manage: Storage

Main memory – only large storage media that the CPU can access directly.

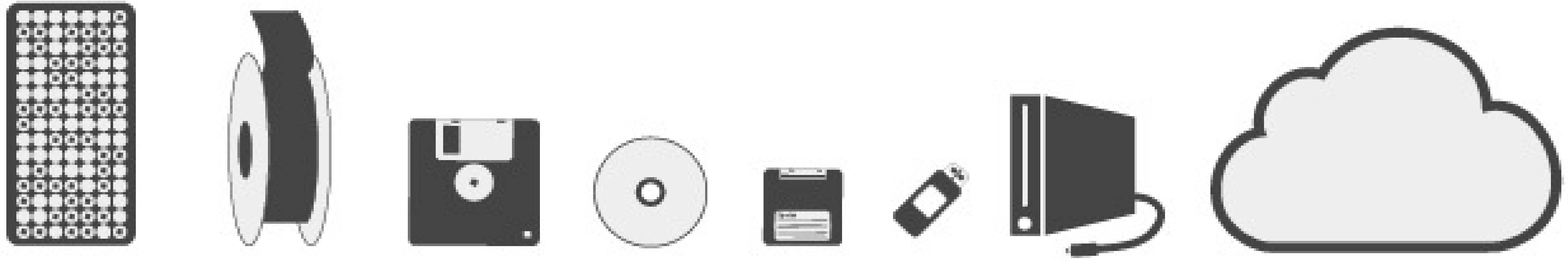
Secondary storage – extension of main memory that provides large nonvolatile storage capacity.

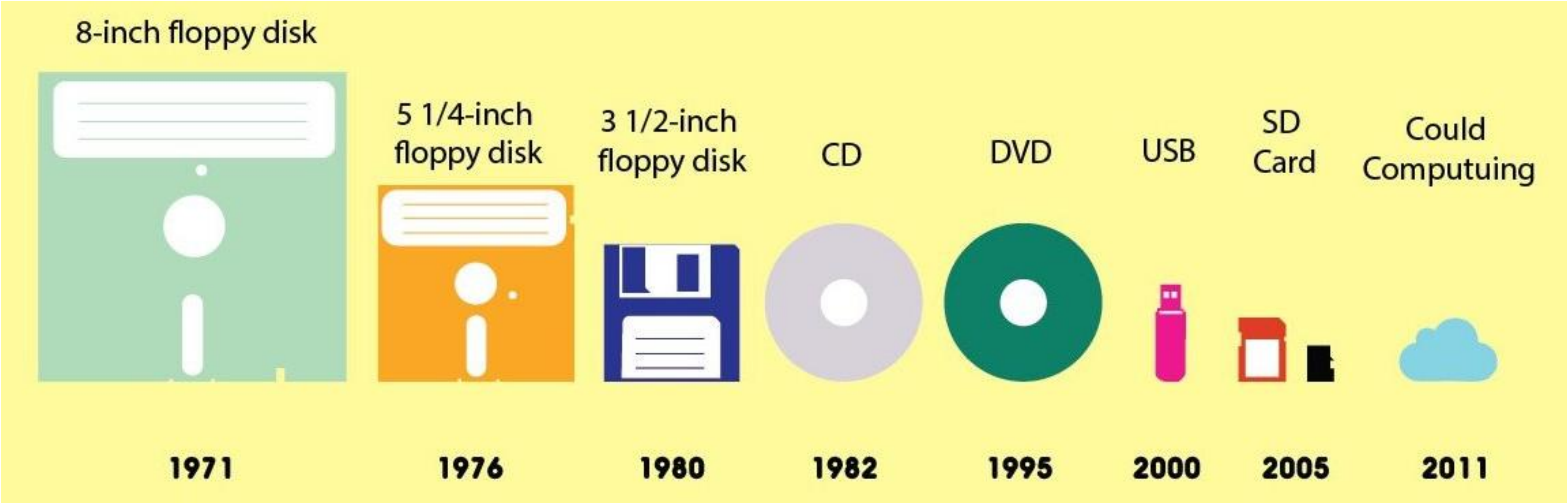
Magnetic disks – rigid metal or glass platters covered with magnetic recording material

Disk surface is logically divided into *tracks*, which are subdivided into *sectors*.

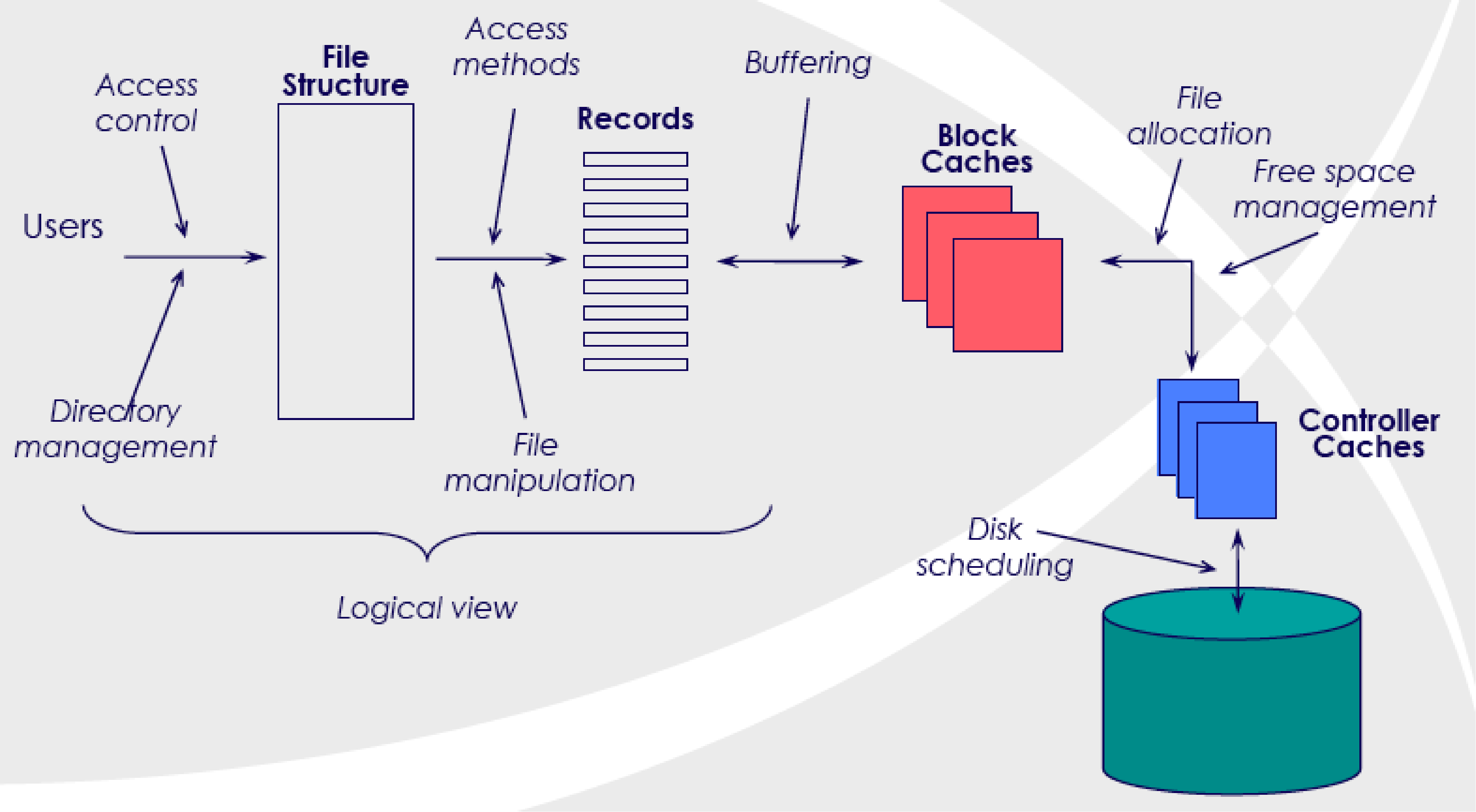
The *disk controller* determines the logical interaction between the device and the computer.

Evolution of computer storage

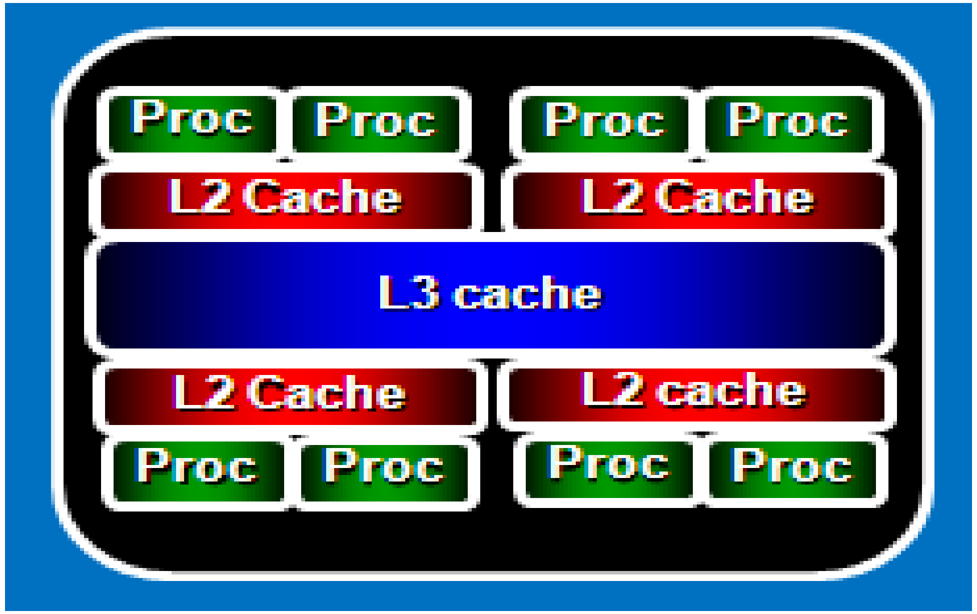


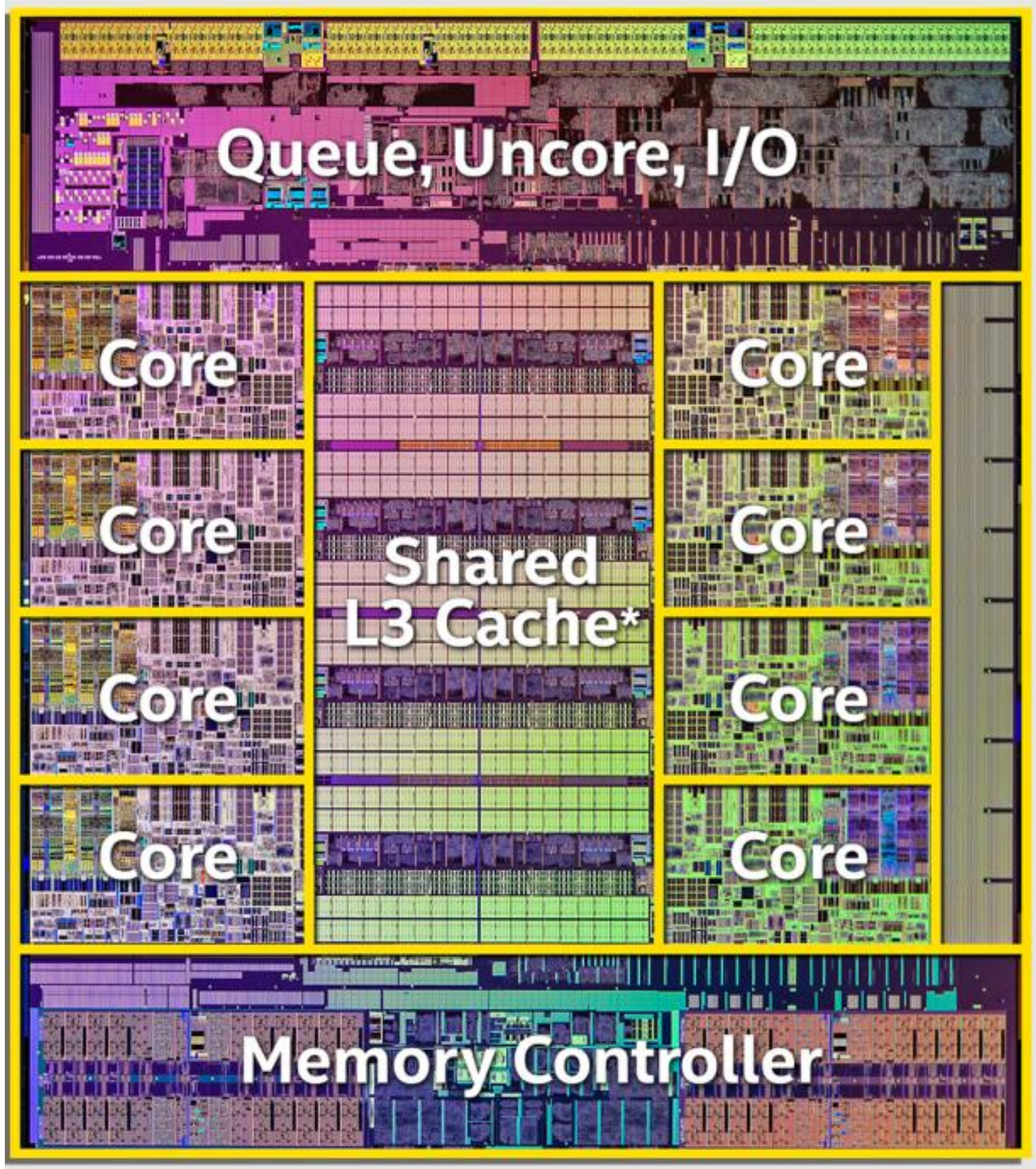


Storage Management

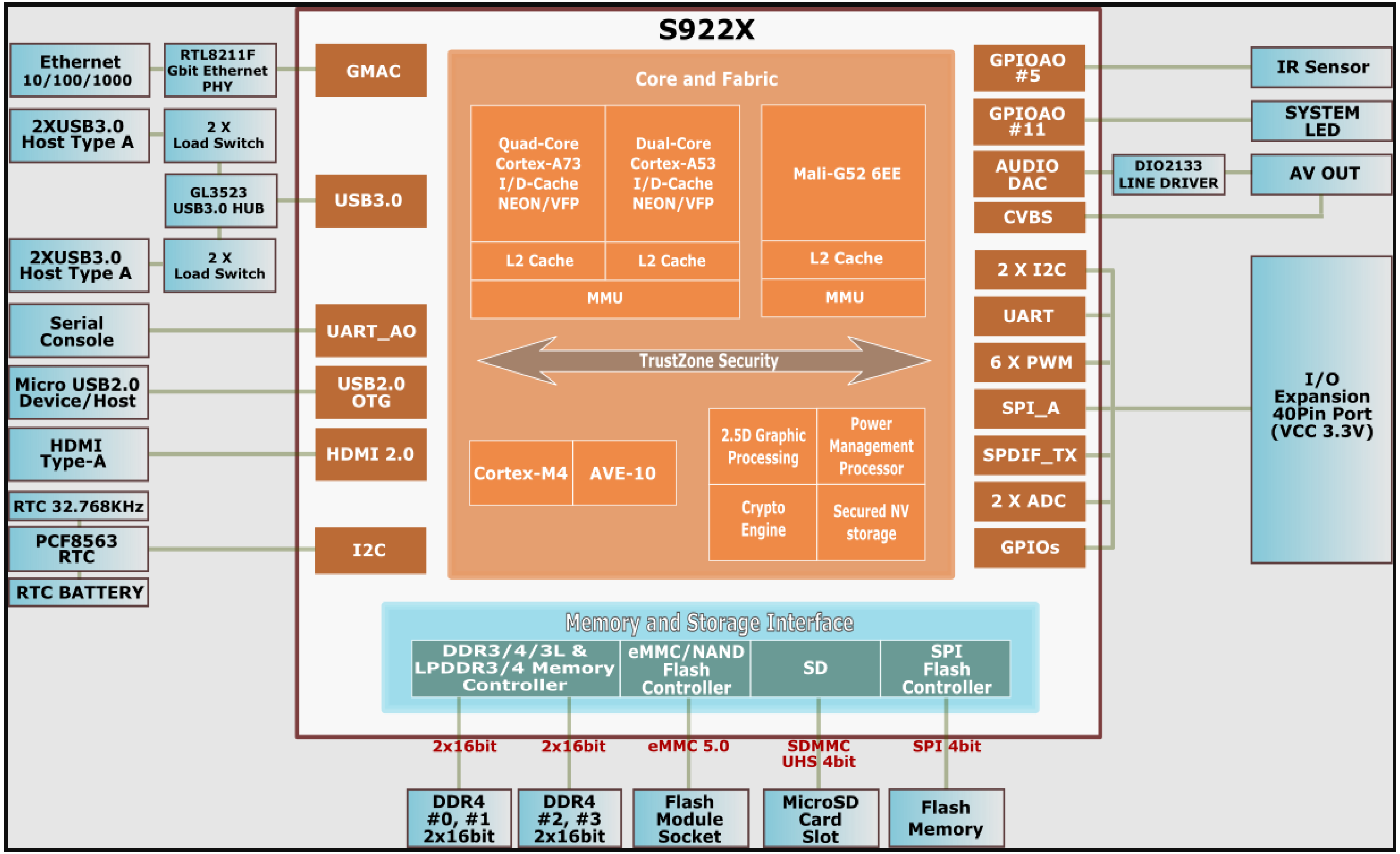


Intel Core i7 Processor

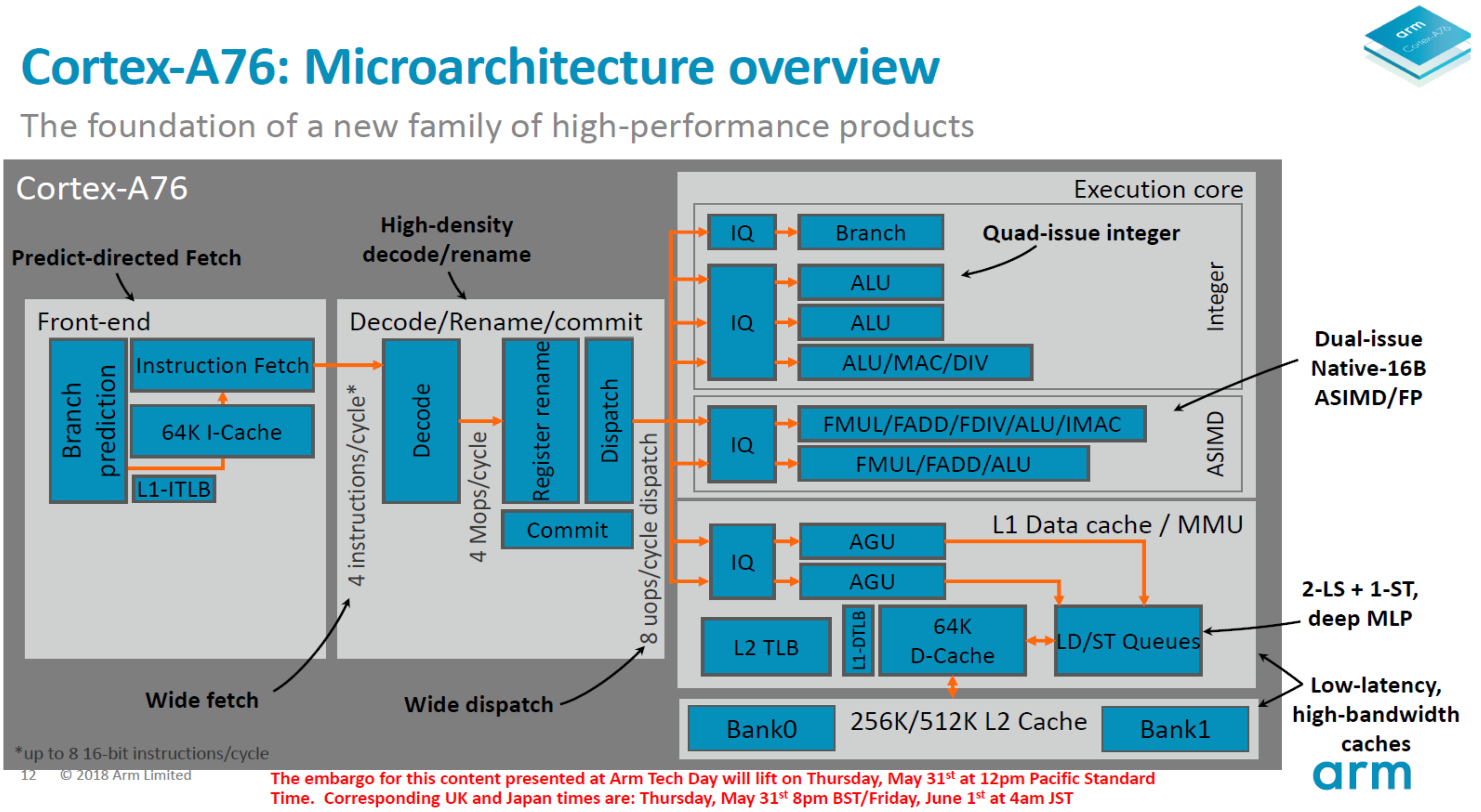




ODROID-N2 Architecture



Raspberry Pi 5



Cortex-A76: Microarchitecture overview

The foundation of a new family of high-performance products

Predict-directed Fetch

High-density decode/rename

Quad-issue integer

Dual-issue Native-16B ASIMD/FP

2-LS + 1-ST, deep MLP

Low-latency, high-bandwidth caches

Wide fetch

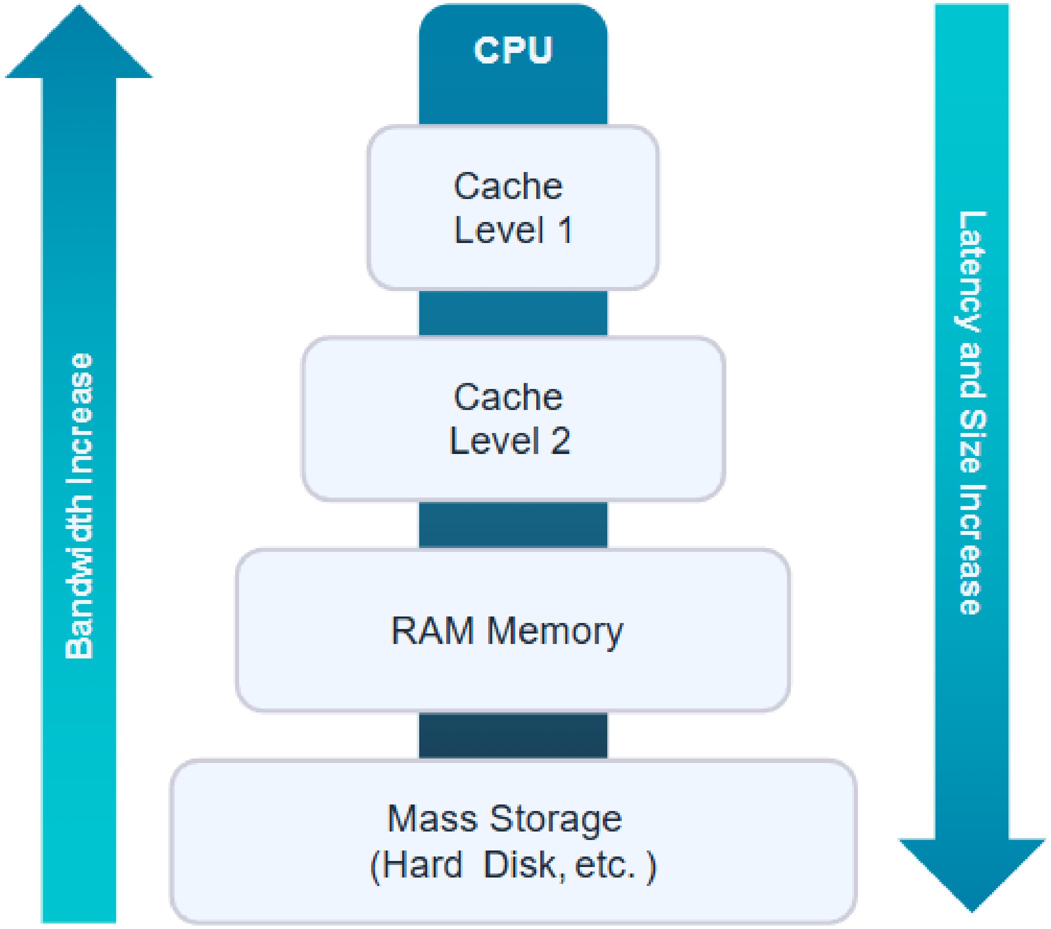
Wide dispatch

*up to 8 16-bit instructions/cycle

12 © 2018 Arm Limited

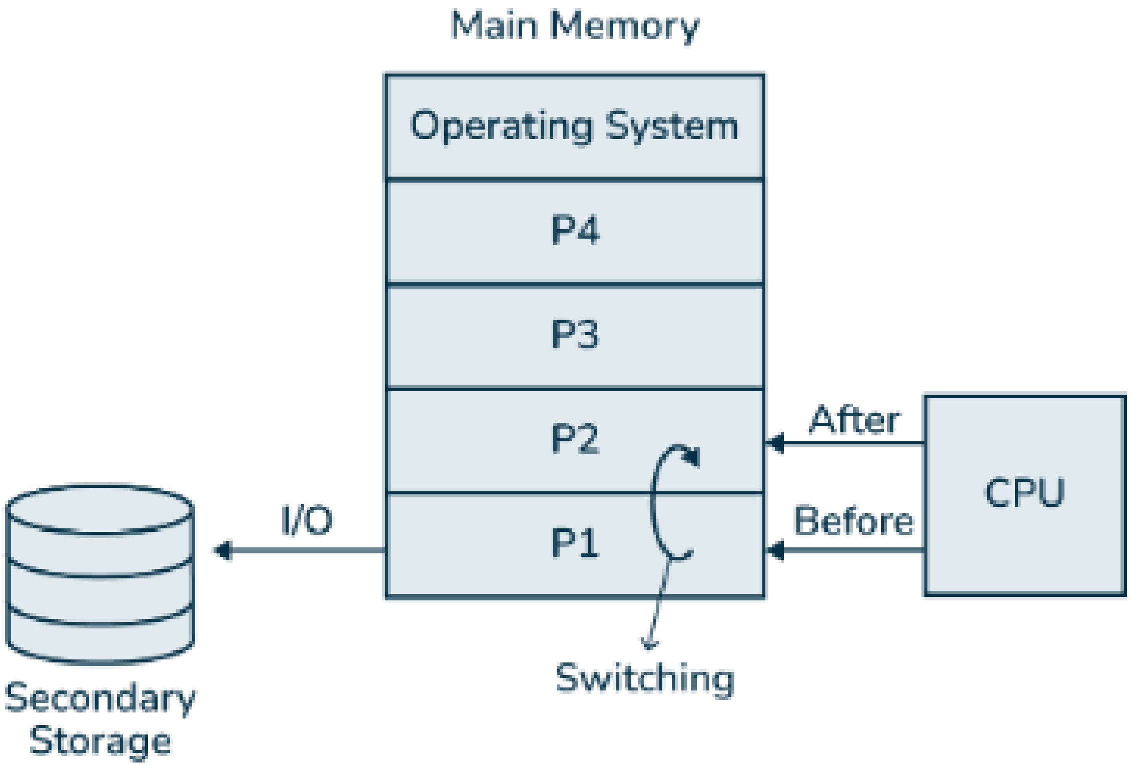
The embargo for this content presented at Arm Tech Day will lift on Thursday, May 31st at 12pm Pacific Standard Time. Corresponding UK and Japan times are: Thursday, May 31st 8pm BST/Friday, June 1st at 4am JST

Caching



- Important principle, performed at many levels in a computer (in hardware, operating system, software)
- Information in use copied from slower to faster storage temporarily
- Faster storage (cache) checked first to determine if information is there
  - If it is, information used directly from the cache (fast)
  - If not, data copied to cache and used there
- Cache smaller than storage being cached
  - Cache management important design problem
  - Cache size and replacement policy

Support: Multiprogramming



- Single user cannot keep CPU and I/O devices busy at all times
- Multiprogramming organizes jobs (code and data) so CPU always has one to execute
- A subset of total jobs in system is kept in memory
- One job selected and run via **job scheduling**
- When it has to wait (for I/O for example), OS switches to another job

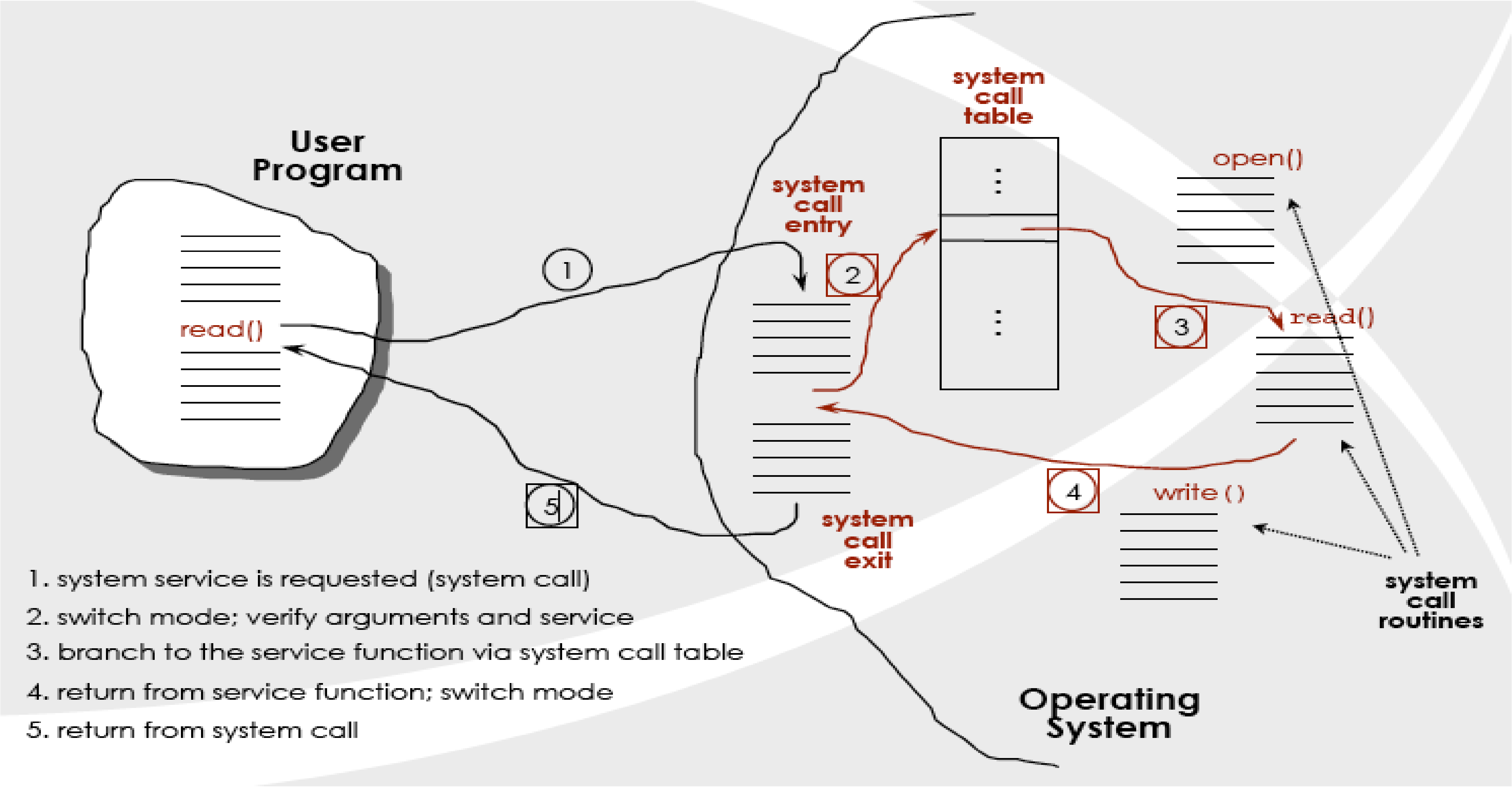
Protection and Security

- **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

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 Program 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)
- Why use APIs rather than system calls?

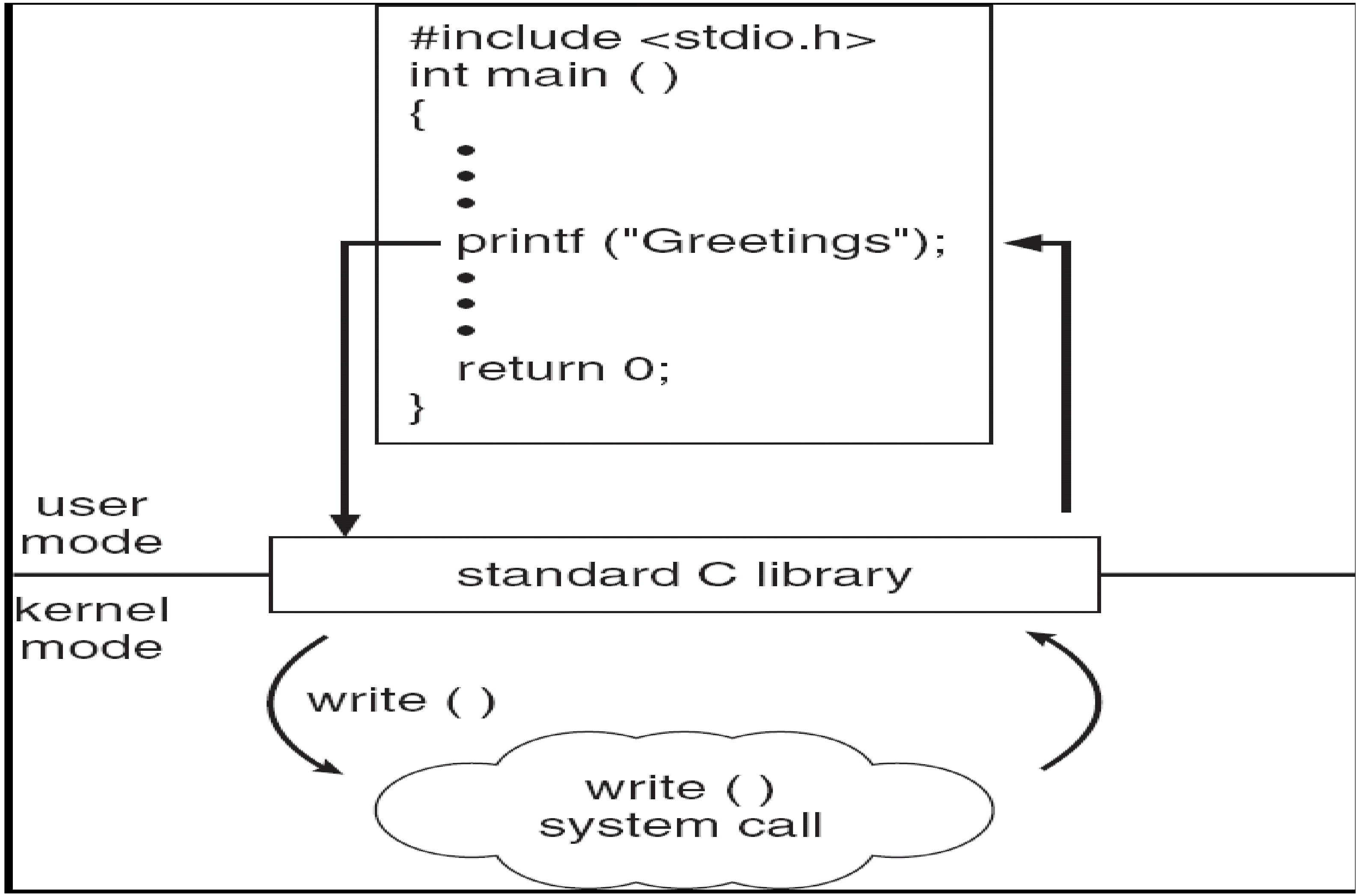
System calls



1. system service is requested (system call)
2. switch mode; verify arguments and service
3. branch to the service function via system call table
4. return from service function; switch mode
5. return from system call

Standard C Library Example

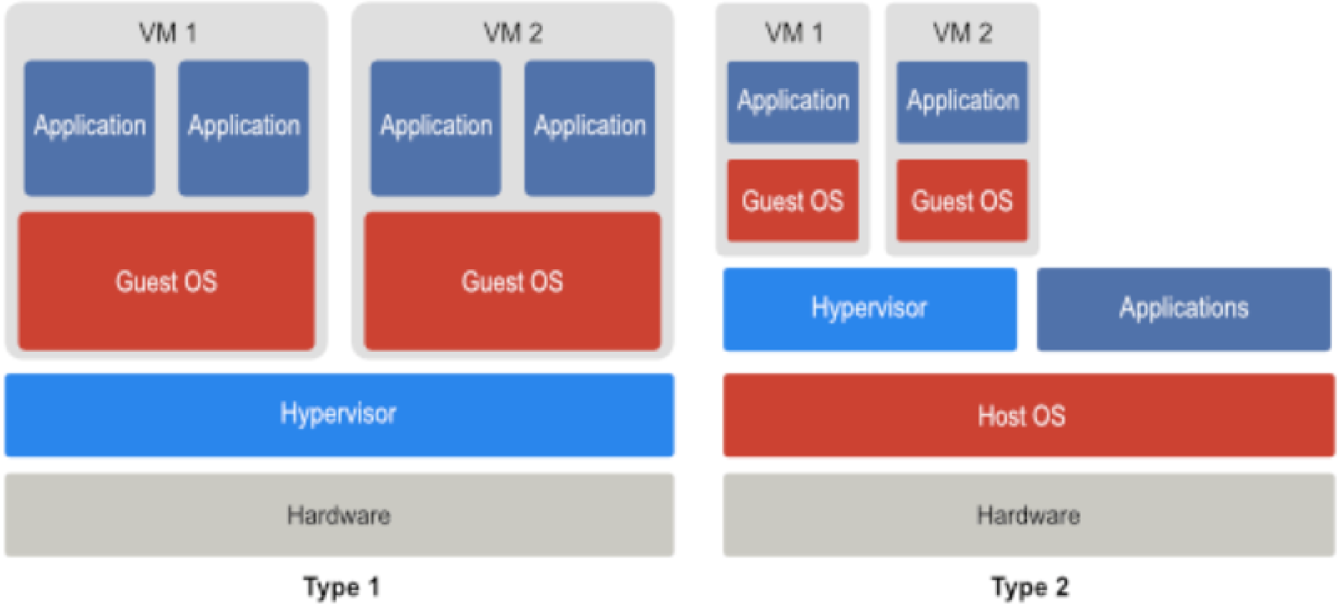
C program invoking printf() library call, which calls write() system call



```
#include <stdio.h>
int main ( )
{
   •
   •
   •
   printf ("Greetings");
   •
   •
   •
   return 0;
}
```

Hypervisor

- A hypervisor, also known as a virtual machine monitor or VMM, is **software that creates and runs virtual machines (VMs)**. A hypervisor allows one host computer to support multiple guest VMs by virtually sharing its resources, such as memory and processing.
- Type 1 Hypervisor that works directly on the host machine’s hardware resources.
- Type 2 Hypervisor. A hypervisor creates the virtualization layer that separates the guest machine from the underlying operating system.



Lab 1: Unix command

- ssh studentID@t.coe.psu.ac.th
  - Register with PSU passport at https://user.coe.psu.ac.th
- Basic Unix command
  - Make/Delete/Change directory
  - Copy/Move/Delete files
  - Change mode
  - gcc compiler

Lab 2: C programming review

- Compile a C program
  - gcc filename.c
- Run the program
  - ./a.exe
- Basic C programming
  - main, function
  - Variables, array, structure
  - printf, scanf
  - switch, if-else statement
  - for loop, do-while, while loop
  - comment

Lab 3: JAVA programming review

- cmd for window users
  - Using command-line
- Compile a JAVA program
  - javac filename.java
- Run the program
  - java filename
- Basic JAVA programming