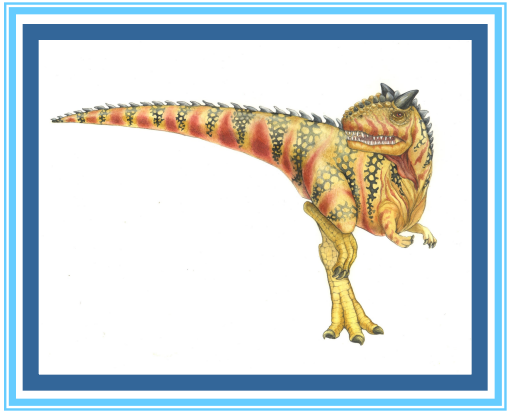
COMP 3511 Operating Systems
Spring 2022



Lectures and Labs/Tutorials

- **Lectures (Feb 7 – May 11, 2022):**
  - The university may make further announcements and we may switch back to face-to-face teaching later
  - L1 Tuesday/Thursday 1:30 pm – 2:50 pm, Room 4620 Lift 31-32

    ZOOM ID: 926 5903 5946 Passcode: 120517
  - L2 Monday/Wednesday 9:00 am – 10:20 am, Rm 2502, Lift 25-26

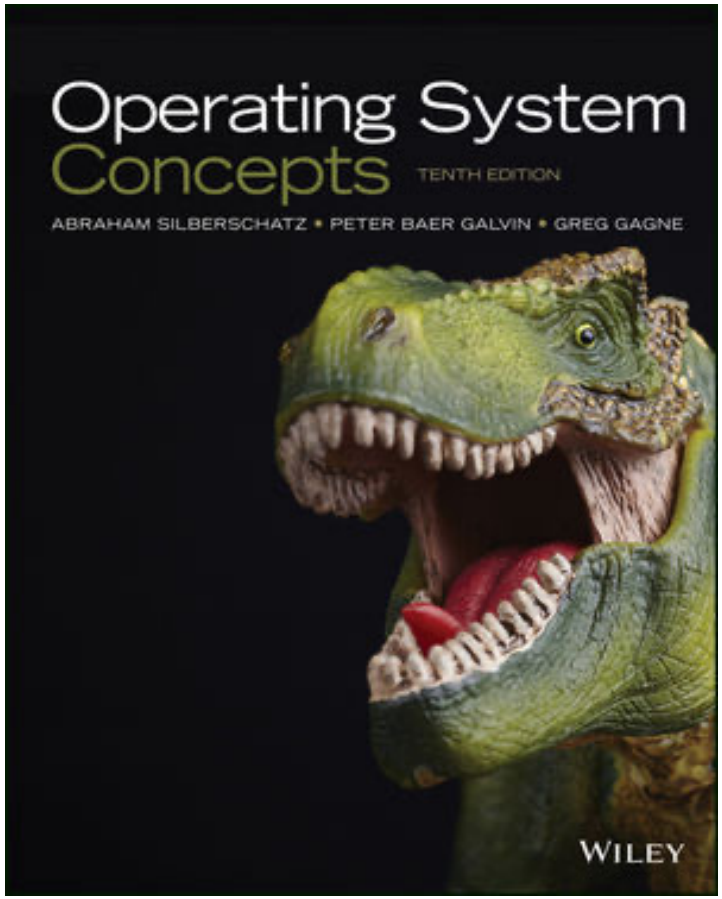
    ZOOM ID: 967 7039 7724 Passcode: 901069

- **Lab and Tutorials**
  - LA1 Friday 11:30 am - 1:20 pm,
  - LA2 Friday 4:00 pm – 5:50 pm,
  - LA3 Recording of the lab/tutorial

- **Course Website:** https://course.cse.ust.hk/comp3511/
- **Instructors:** Bo Li (L1) and Kai Chen (L2)

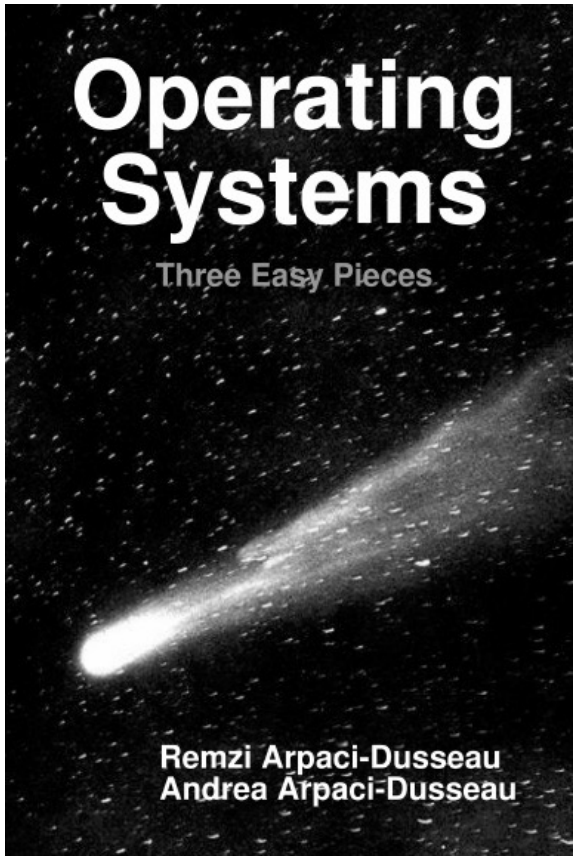
Textbook

- Operating System Concepts, A. Silberschatz, P. B. Galvin and G. Gagne, 10th Edition



Reference Book

- Operating Systems: Three Easy Pieces, Remzi H. Arpaci-Dusseau and Andrea C. Arpaci-Dusseau
- Online (free access): http://pages.cs.wisc.edu/~remzi/OSTEP/#book-chapters



Course Prerequisite

- **COMP 2611 or ELEC 2300 or ELEC 2350 (Computer Organization)**
  - Basic computer organization knowledge – von Neumann machine, CPU, caching, pipelining, memory hierarchy, I/O systems, interrupt, storage and hard drives
- **COMP 1029C or COMP 2011 or COMP 2012H (C programming)**
  - UNIX/Linux basic
  - Programming - C programming

Labs and Tutorials

- **9 Labs and Tutorials – rough schedule**
  - Lab #1 (week 2): Introduction to Linux
  - Lab #2 (week 3): C/C++ programming
  - Lab #3 (week 4): Linux process, pipe(), and Project #1
  - Lab #4 (week 6): review
  - Lab #5 (week 7): Project #2
  - Lab #6 (week 9): review
  - Lab #7 (week 10): Project #3
  - Lab #8 (week 11) review
  - Lab #9 (week 13): review

Grading Scheme

- **4 Homework - written assignments – 20% (5% each)**
  - HW #1 (week 2-4)
  - HW #2 (week 5-7)
  - HW #3 (week 8-10)
  - HW #4 (week 11-13)
- **3 Projects - programming assignments – 30%**
  - Project #1 (week 4-6) (10%)
  - Project #2 (week 7-9) (10%)
  - Project #3 (week 10-12) (10%)
- **Midterm Exam** (~week #8) - 20%
- **Final Exam** - 30%

Plagiarism Policy

- There are differences between collaborations, discussions and copy!

- First time: all involved get ZERO marks, and will be reported to ARR
- Second time: need to terminate (Fail grade)
- Any cheating in midterm or final exam results in automatic Fail grade

Lecture Format

- Lectures:
  - Lecture notes are made available before lectures
- Tutorials and Labs
  - Unix environment, editor (vim), compile and run programs, Makefile
  - C++ and C programming basic
  - Tutorials on programming assignments
  - C programming APIs and interfaces
  - Supplement materials with more examples and exercises
- Reading the corresponding materials in the textbook and reference book
  - Lecture notes do not and can not cover everything
- Chapter Summaries
  - Comprehensive summary at the end of each chapter

Assignments

- Written assignments
  - Due by time specified
  - Contact the TA for any disputes on the grading
  - Regrading requests be granted within **two weeks** after the homework grades are released
  - Late policy: 10% reduction, only one day delay is allowed
- Programming assignments - **individual project**
  - Due by time specified
  - Run on a CS Lab 2 Linux machine
  - Submit it using Canvas
  - Regrading requests be granted within **two weeks** after the grades are released
  - Late policy : 10% reduction, only one day delay is allowed

Midterm and Final Examinations

- Midterm Exam
  - TBD
- Final Exam
  - TBD

- **All exams are open-book and open-notes**

- **No make-up exams will be given unless**
  - under special circumstances, e.g., sickness, with prior letters of proof
  - The instructor must be informed before the exam

Tips for Learning

- Attend lectures and lab tutorials
  - Download lecture/lab notes prior to lectures
  - Important concepts are explained, with examples
- Complete homework and projects independently
  - This is to test your knowledge and how much you comprehend
- **Spend 30 minutes or so each week to review the content**
  - Chapter summary helps
  - This can save you lots of time later when you prepare for exams
  - You can not expect to learn everything 2-3 days before exams
  - Knowledge is accumulated incrementally
- Start your project earlier
  - Have a plan for the project
- Raise questions during or after lectures !
  - Do not delay your questions until close to the exams

What you are suppose to learn

- Define the fundamental principles, strategies and algorithms used in the design and implementation of operating systems
- Analyze and evaluate operating system functions
- Understand the basic structure of an operating system kernel, and identify the relationship between the various subsystems
- Identify the typical events, alerts, and symptoms indicating potential operating system problems
- Design and implement programs for basic operating system functions and algorithms
- **Advanced OS course** – COMP 4511 System and Kernel Programming in Linux

Course Outline

- **Overview** (3 lectures)
  - Basic OS concept (2 lectures)
  - System architecture (1 lectures)
- **Process and Thread** (12 lectures)
  - Process and thread (4 lectures)
  - CPU scheduling (4 lectures)
  - Synchronization (2 lectures)
  - Deadlock (2 lectures)
- **Memory and storage** (8 lectures)
  - Memory management (2 lectures)
  - Virtual memory (3 lectures)
  - Secondary storage (1 lectures)
  - File systems and implementation (2 lectures)
- **Security and protection** (2 lectures)
  - Security (1 lecture)
  - Protection (1 lecture)

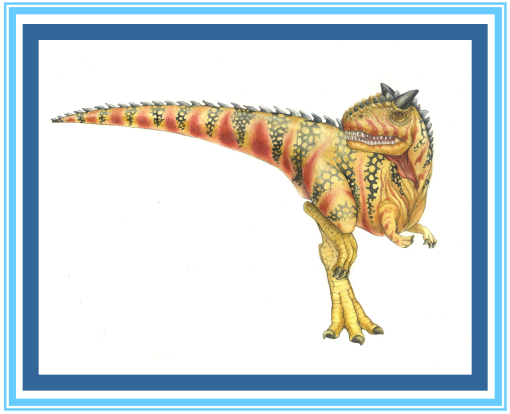
Course Coverage

- **Overview**
  - Chapter 1 – high-level description of OS, basic components in computer systems including multi-processor systems, virtualization
  - Chapter 2 – OS services including APIs and system calls, and common OS design approaches (monolithic, layered, microkernel, modular)
- **Process and Thread**
  - Chapter 3 (Process) – concept of a process capturing a program execution, creating and terminating a process, IPC
  - Chapter 4 (Thread) – concept of a thread and multi-threaded process for concurrent execution of a program
  - Chapter 5 (CPU scheduling) – CPU scheduling algorithms including real-time scheduling, and issues with multiprocessor scheduling and thread scheduling
  - Chapter 6-7 (Synchronization) – critical section problem, synchronization tools (hardware and software), and synchronization examples
  - Chapter 8 (Deadlock) – deadlock characterization, resource allocation graph, deadlock prevention, avoidance and detection algorithms

Course Coverage (Cont.)

- **Memory and storage**
  - Chapter 9 (Memory) - contiguous memory allocation, segmentation, paging including hierarchical paging
  - Chapter 10 (Virtual memory) – virtual vs. physical memory, demand paging, page replacement algorithm, thrashing and frame allocation
  - Chapter 11 (Secondary storage) – hard drive or disk structure, disk scheduling algorithms and RAID (disk array) structure
  - Chapter 13-14 (File systems) – file access methods, directory structure and implementation, basic file system data structure (on-disk and in-memory), disk space management including disk block allocation
- **Security and protection**
  - Chapter 16 (Security) – security issues, program, system and network threats, and user authentication
  - Chapter 17 (Protection) – basic protection principles, protection rings, protection domain and implementation (access matrix)

Chapter 1: Introduction



Chapter 1: Introduction

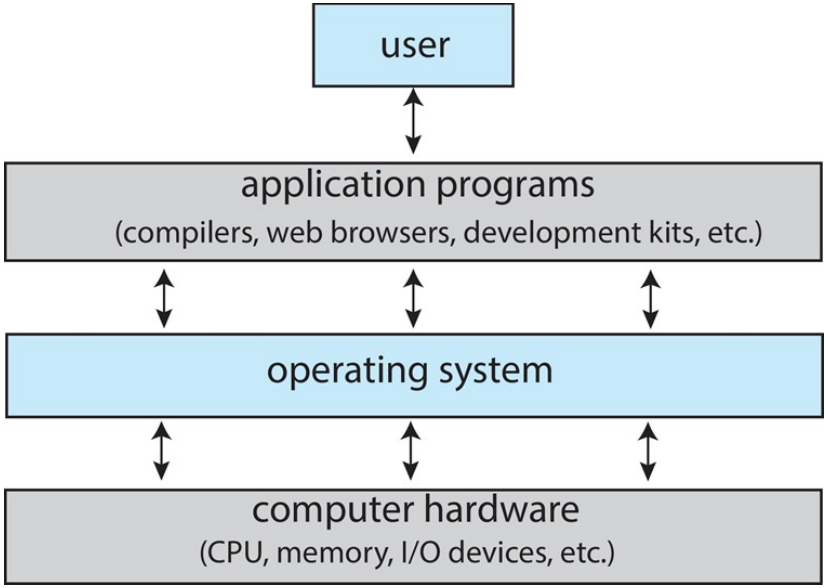
- What Operating Systems Do
- Computer System Organization and Architecture
- Multiprocessor and Parallel Systems
- Definition of Operating Systems
- Virtualization and Cloud Computing
- Free and Open-Source Operating Systems

Objectives

- Describe the general organization of a computer system and the role of interrupts.
- Describe the components in a modern multiprocessor computer system.
- Discuss how operating systems are used in various computing environments
- Provide examples of free and open-source operating systems

What is an Operating System?

- **Hardware** – basic computing resources, CPU, memory, I/O devices
- **Operating system** – controls and coordinates use of hardware among various applications and among different users
- **Application programs** – define the ways how system resources are used to solve user computing problems
  - Editors, compilers, web browsers, database, video games, and etc.
- **Users** - people, machines, other computers or devices



What is an Operating System?

- OS is a program that acts as an intermediary between users and computer hardware
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Manage and use the computer hardware in an efficient manner
- User view
  - Want convenience, ease of use, good performance and security
  - Users do not care about resource utilization, efficiency
- System view
  - OS as a resource allocator and a control program

What Operating Systems Do

- It depends on the point of view and the target devices
- Shared computers such as mainframe or minicomputer
  - OS needs to try to keep all users satisfied – performance vs. fairness
- Users of individual systems such as workstations have dedicated resources, also frequently use shared resources from servers
- Mobile devices (e.g., smartphones and handheld devices) are resource constrained
  - must be optimized for usability and battery life
  - target specific user interfaces such as touch screen, voice control such as Apple's Siri
- Computers or computing devices with little or no user interface
  - Embedded systems - present within home devices (e.g., AC, toasters), automobiles, ships, spacecraft.
  - Designed to run primarily without user intervention – may have numeric keypads and indicator lights to show status

Operating System Definition

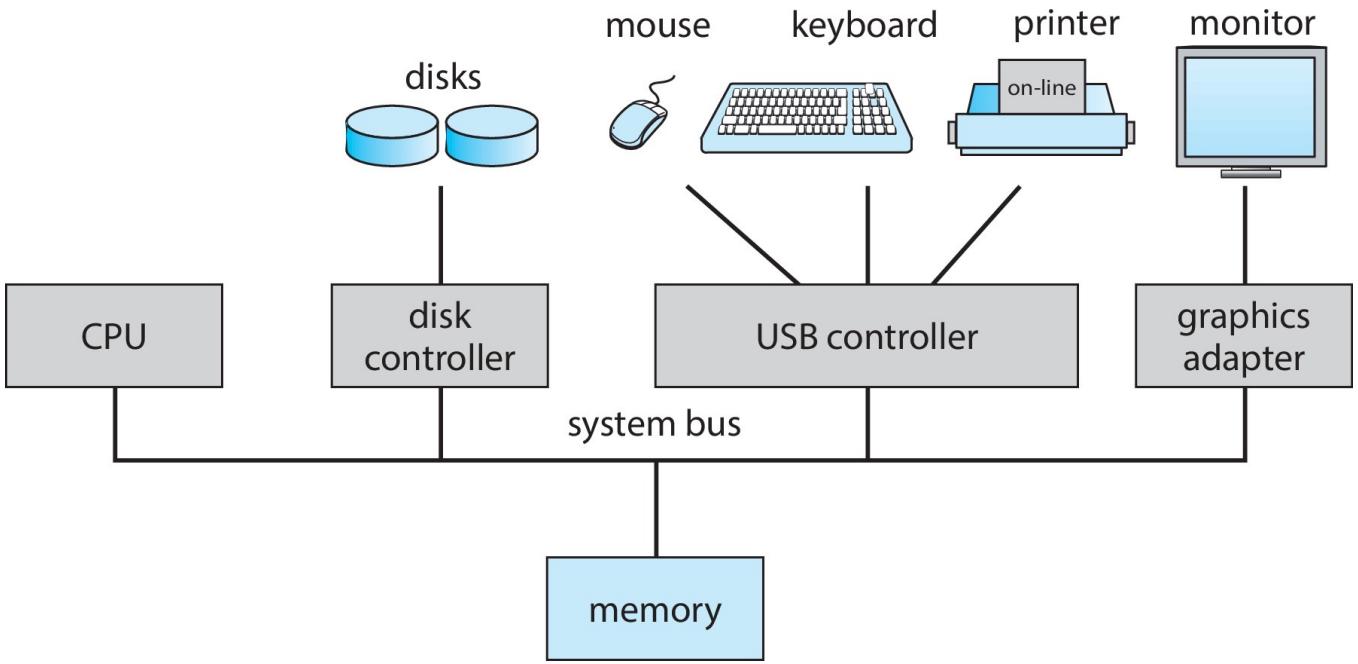
- **Moore's Law** - the number of transistors on an integrated circuit would double every ~18 months. Computers gained in functionality and shrank in size, leading to a vast number of uses and a variety of operating systems
- There is no universally accepted definition on OS
  - "Everything a vendor ships when you order an operating system" is a good approximation, but it varies wildly
- OS is a resource allocator
  - Manages all resources – hardware and software
  - Decides between conflicting requests for efficient and fair resource use
- OS is a control program
  - Controls execution of programs, prevent errors and improper use of the computer
- In a nutshell, OS manages and controls hardware, and helps to facilitate programs to run on computers.

Operating System Definition

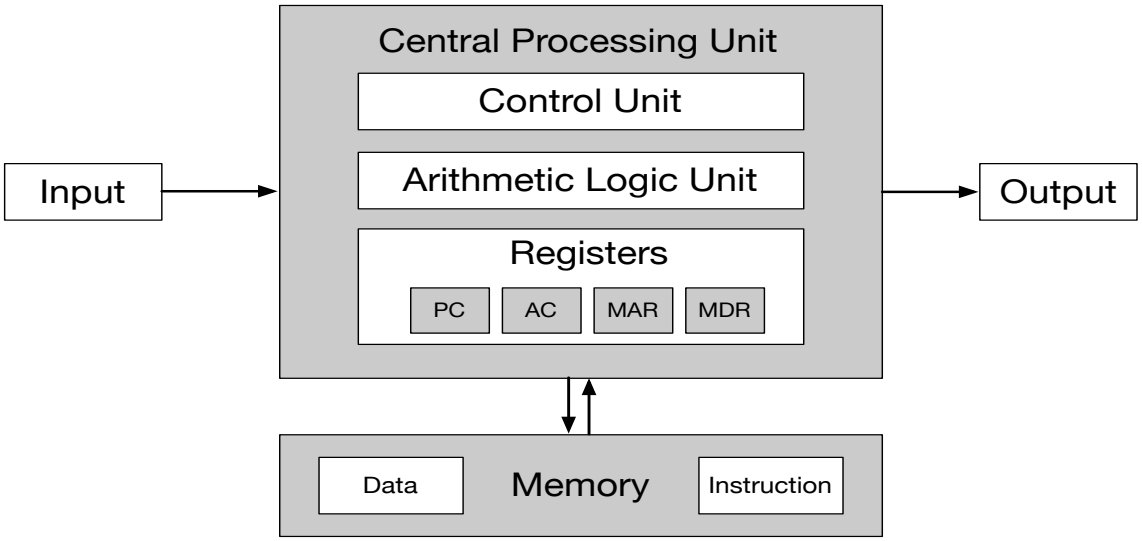
- **Kernel**
  - "The one program running at all times on the computer"
  - The essential functionalities that will be discussed in this course
- **Middleware**
  - A set of software frameworks that provide additional services to application developers such as databases, multimedia, graphics
  - Popular in mobile OSes - Apple's iOS and Google's Android
- Everything else
  - **System programs** (ships with the operating system, but not part of the kernel), such as word processors, browsers
  - Application programs, not associated with the operating system – apps
- OS includes the always running kernel, middleware frameworks that ease application development and provide additional features, and system programs that aid in managing the system while it is running

Computer System Organization

- Computer-system operation
  - One or more CPU cores, device controllers connected through common bus providing access to shared memory
  - **Concurrent** execution of CPUs and devices - competing for memory cycles through shared bus

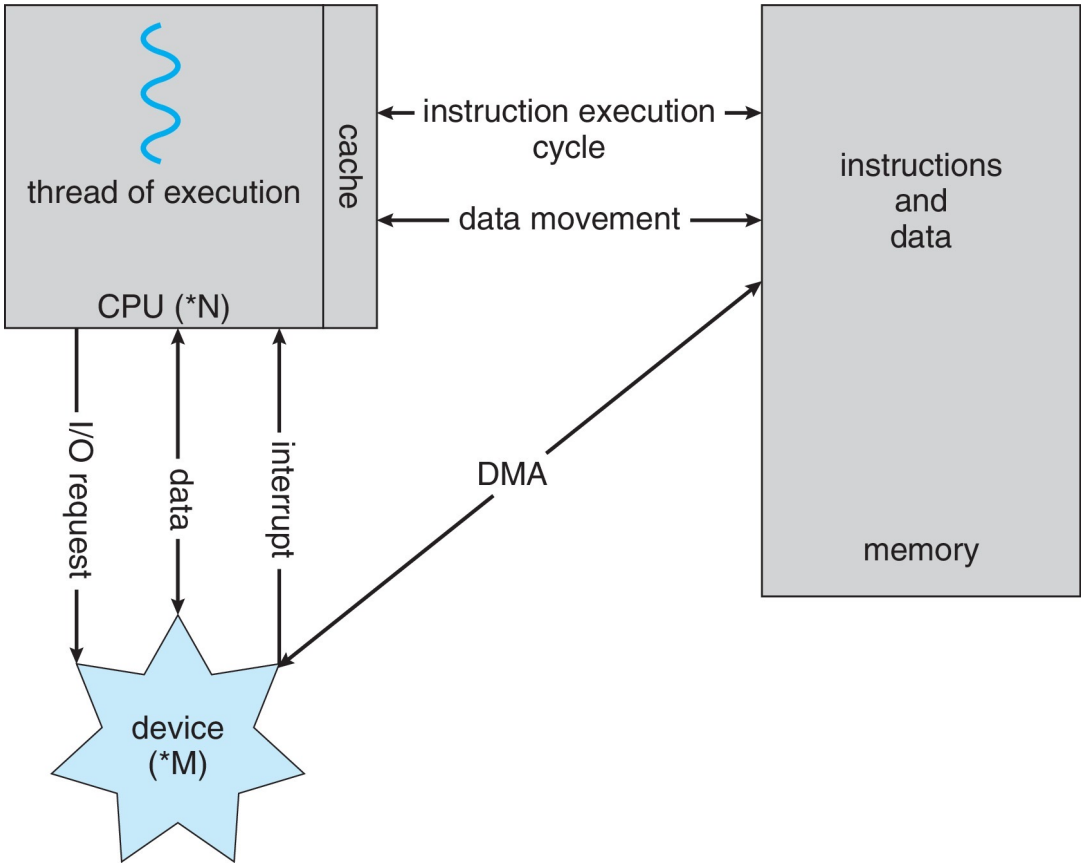


A Von Neumann Architecture



- A processing unit that contains an arithmetic logic unit (ALU) and processor registers
- A control unit that contains an instruction register (IR) and program counter (PC)
- Memory stores data and instructions – along with caches
- External mass storage (not shown in the figure)
- Input and output mechanisms

How a Modern Computer Works



Steps in executing an instruction:

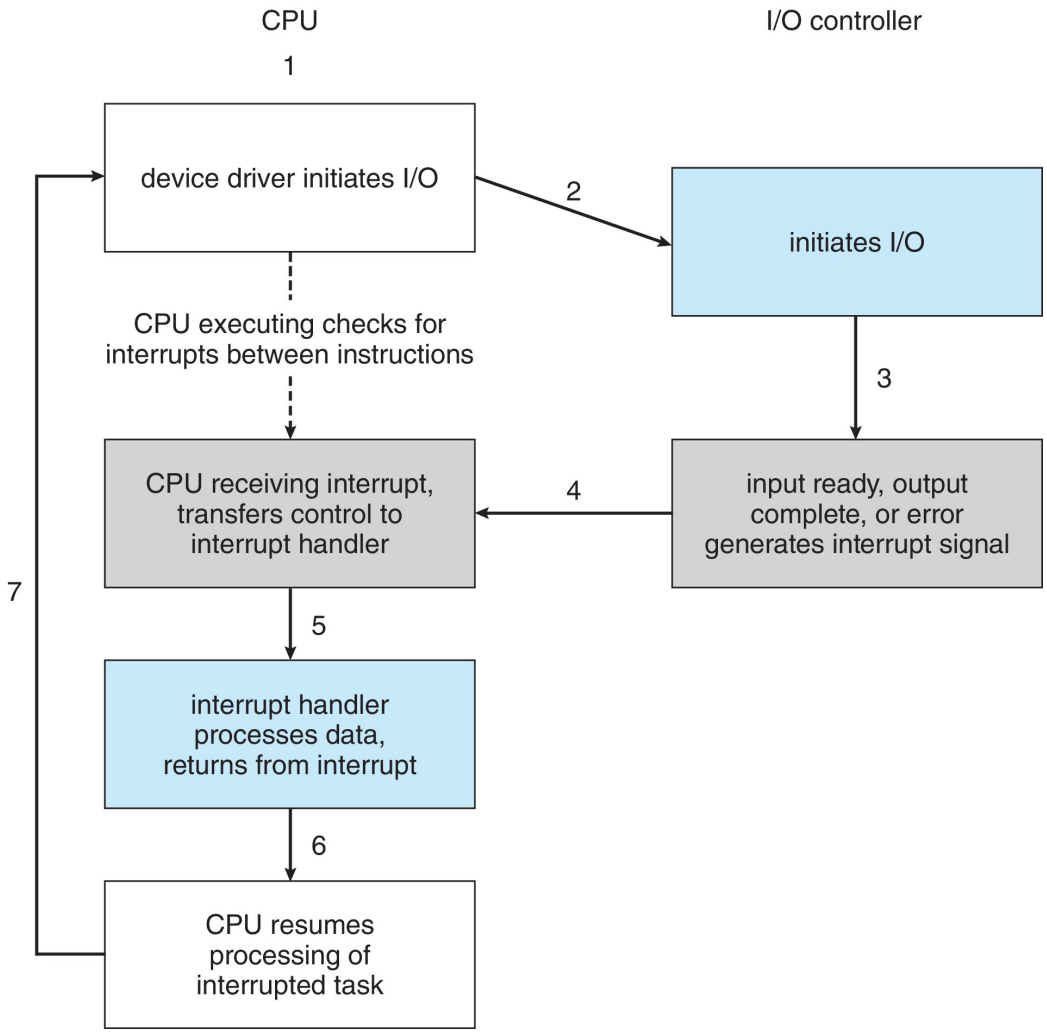
- Fetch instruction
- Decode instruction
- Fetch data
- Execute instruction
- Write back if any

A von Neumann architecture

Computer-System Operation

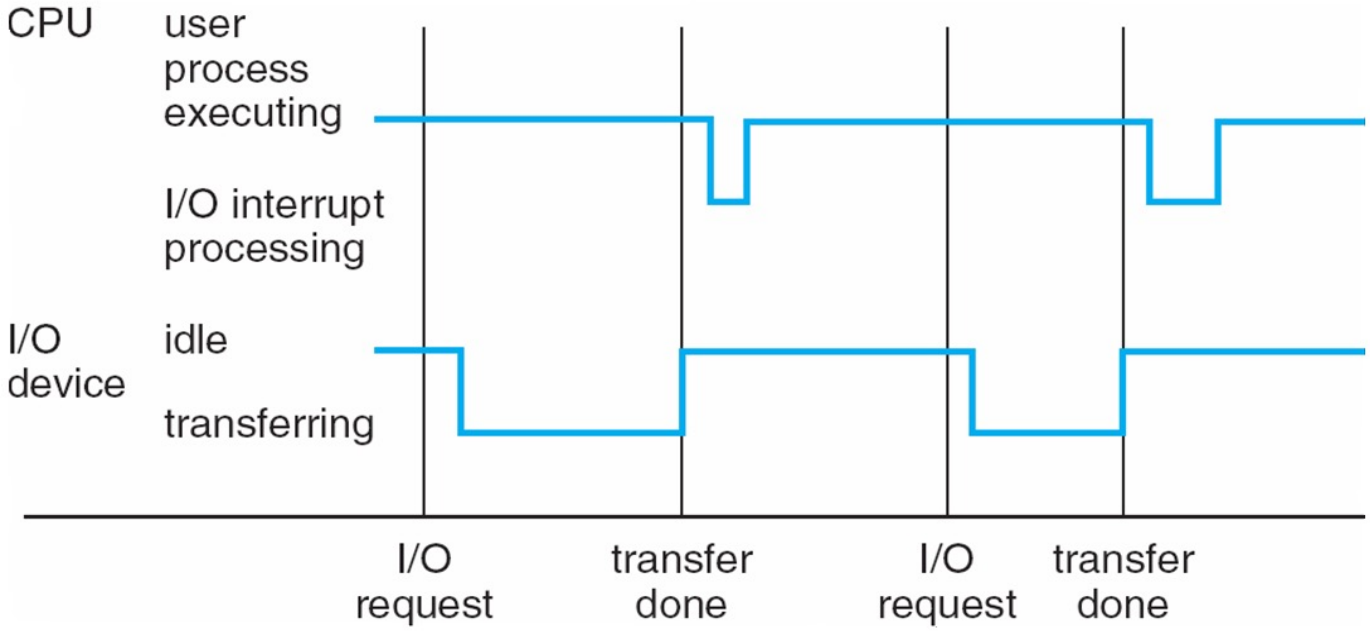
- I/O devices and the CPU can execute concurrently
- Each device controller is in charge of a particular device
- Each device controller has a local buffer
- The device controller is responsible for moving the data between the peripheral devices that it controls and its local buffer storage
  - **I/O operations** are from the device to local buffer of the controller
- CPU moves data from/to main memory to/from local buffers, typically for slow devices such as keyboard and mouse. **DMA** controller is used for move the data for fast devices like disks
- The device controller informs CPU that it has finished an operation by causing an **interrupt**
  - For input devices, this implies that data is available in local buffer
  - Interrupts are widely used in modern operating systems to handle asynchronous events - device controllers and hardware faults

Interrupt-Driven I/O Cycle



Interrupt Timeline

- CPU and devices execute concurrently
- An I/O device may trigger an interrupt by sending a signal to the CPU
- CPU handles the interrupt, and then returns to the interrupted instruction



Common Functions of Interrupts

- All modern operating systems are interrupt-driven
- Interrupt transfers control to an interrupt service routine or interrupt handler – part of kernel code, which runs an OS code to handle a specific interrupt
- The interrupt mechanism also implements a system of interrupt priority levels, making it possible for a high-priority interrupt to preempt the execution of a low-priority interrupt
- A trap or exception is a software-generated interrupt caused either by an error or a user request (e.g. a system call requesting OS services – to be discussed)

Storage Definitions and Notation Review

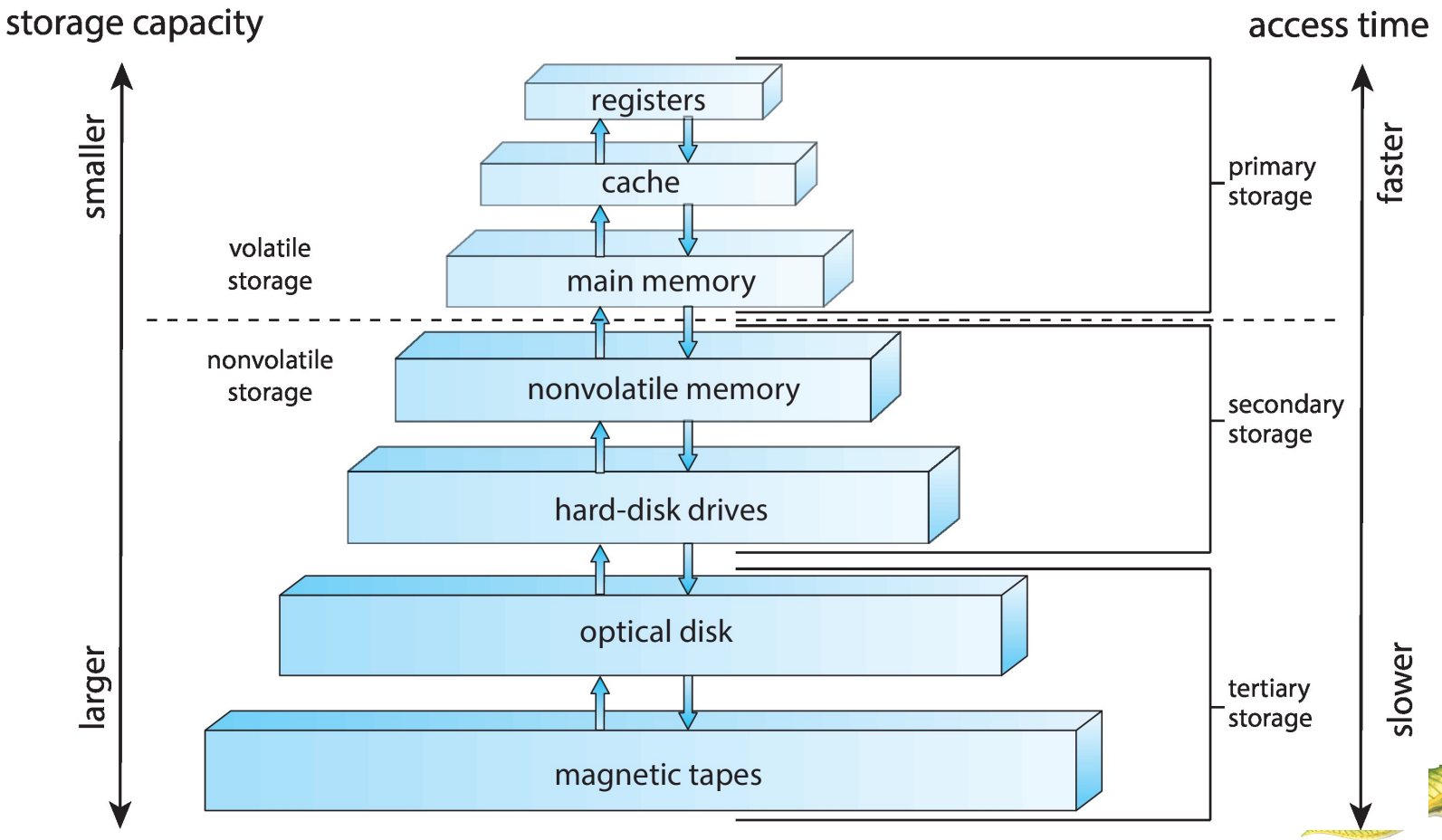
The basic unit of computer storage is the **bit**. A bit can contain one of two values, 0 and 1. All other storage in a computer is based on collections of bits. Given enough bits, it is amazing how many things a computer can represent: numbers, letters, images, movies, sounds, documents, and programs, to name a few. A **byte** is 8 bits, and on most computers it is the smallest convenient chunk of storage. For example, most computers don't have an instruction to move a bit but do have one to move a byte. A less common term is **word**, which is a given computer architecture's native unit of data. A word is made up of one or more bytes. For example, a computer that has 64-bit registers and 64-bit memory addressing typically has 64-bit (8-byte) words. A computer executes many operations in its native word size rather than a byte at a time.

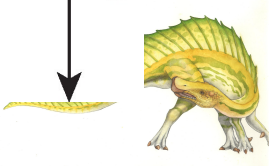
Computer storage, along with most computer throughput, is generally measured and manipulated in bytes and collections of bytes.
A **kilobyte**, or **KB**, is 1,024 bytes
a **megabyte**, or **MB**, is $1,024^2$ bytes
a **gigabyte**, or **GB**, is $1,024^3$ bytes
a **terabyte**, or **TB**, is $1,024^4$ bytes
a **petabyte**, or **PB**, is $1,024^5$ bytes

Computer manufacturers often round off these numbers and say that a megabyte is 1 million bytes and a gigabyte is 1 billion bytes. Networking measurements are an exception to this general rule; they are given in bits (because networks move data a bit at a time).

Storage Hierarchy

- Storage systems organized in **hierarchy**, varied with speed, cost per unit, capacity (size) and volatility (non-volatile disk vs. volatile memory)





Memory

- Main memory – the only large storage media that CPU can access directly
  - Volatile, and typically random-access memory in the form of Dynamic Random-Access Memory (DRAM)
  - The basic operations **load** and **store** instructions to specific memory addresses – byte addressable, an address refers to one byte in memory
- Computers use other forms of memory as well. For example, the first program to run on computer power-on is a bootstrap program, which is stored on electrically erasable programmable read-only memory (EEPROM) and other forms of firmware —storage infrequently written to and is non-volatile

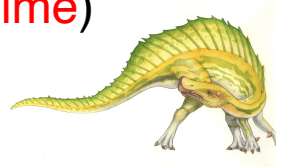
Second Storage

- The secondary storage – extension of main memory providing large **non-volatile** storage capacity, which can hold large quantities of data permanently.
- The most common secondary-storage devices are hard-disk drives (HDDs) and nonvolatile memory (NVM) devices, which provide storage for both programs and data.
  - Non-volatile memory (NVM) devices – faster than hard disks (HDDs)
- There are generally two types of secondary storage
  - **Mechanical**, such as HDDs, optical disks, holographic storage, and magnetic tape
  - **Electrical**, such as flash memory, FRAM, NRAM, and SSD. Electrical storage will be referred to as NVM
- Mechanical storage is generally larger and less expensive per byte than electrical storage. Conversely, electrical storage is typically costly, smaller, more reliable, and faster than mechanical storage.

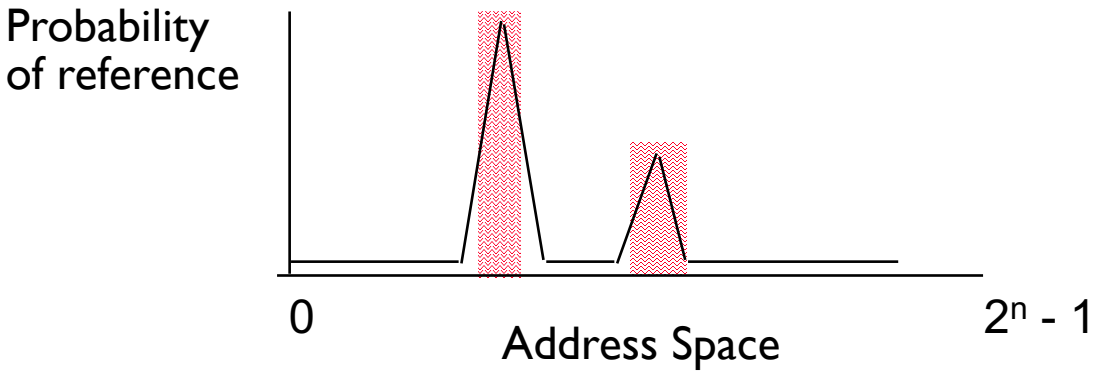
Caching

- **Important principle** - performed at many levels in computers
  - Cache for memory, address translation, file blocks, file names (frequent used), file directories, network routes, and etc.
- A subset of information copied from slower to faster storage temporarily
  - Make frequently used case faster and less frequent case less dominant
- Cache first checks to determine if information is inside the cache
  - Hit: if it is, information used directly from the cache (fast)
  - Miss: if not, data copied from slower storage to cache and used there
- Cache usually much smaller than storage (e.g. memory) being cached
  - Cache management: cache size and replacement policy
  - Major criteria – cache hit ratio, percentage content found in cache
- Important measurement

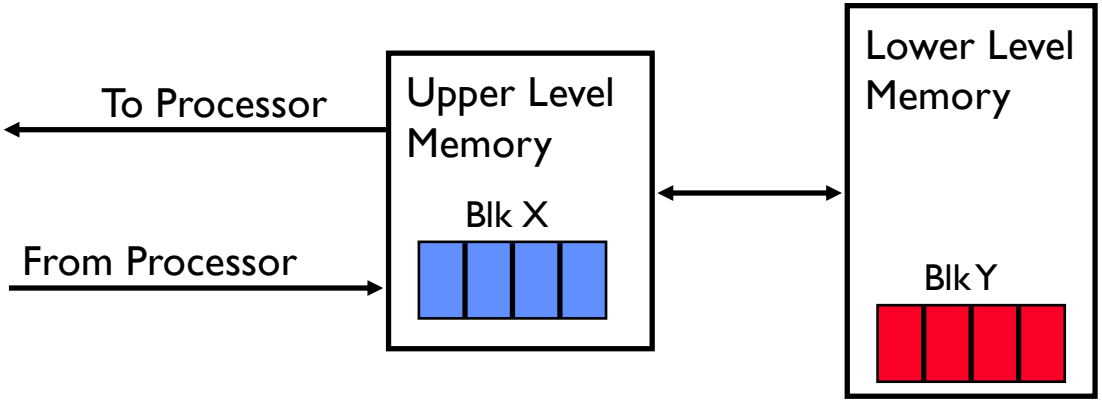
  Average Access time = (Hit Rate x Hit Time) + (Miss Rate x Miss Time)



Why Does Caching Work? - Locality



- Temporal locality (locality in Time)
  - Keep the recently accessed data items closer to processor
- Spatial locality (locality in Space)
  - Move contiguous blocks to the upper levels (caches)



Characteristics of Various Types of Storage

Level	1	2	3	4	5
Name	registers	cache	main memory	solid-state disk	magnetic disk
Typical size	< 1 KB	< 16MB	< 64GB	< 1 TB	< 10 TB
Implementation technology	custom memory with multiple ports CMOS	on-chip or off-chip CMOS SRAM	CMOS SRAM	flash memory	magnetic disk
Access time (ns)	0.25-0.5	0.5-25	80-250	25,000-50,000	5,000,000
Bandwidth (MB/sec)	20,000-100,000	5,000-10,000	1,000-5,000	500	20-150
Managed by	compiler	hardware	operating system	operating system	operating system
Backed by	cache	main memory	disk	disk	disk or tape

Movement between levels of storage hierarchy can be explicit or implicit

I/O Subsystem

- OS needs to accommodate a wide variety of devices, each with different capabilities, control-bit definitions, and protocols for interacting with host
- OS enables I/O devices to be treated in a standard, uniform way – that involves abstraction, encapsulation, and software layering, like for any complex software engineering problems
- I/O system calls encapsulate device behaviors in a few generic classes, each is accessed through a standardized set of functions - an interface
- One purpose of OS is to hide peculiarities of hardware devices from users
- 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

Direct Memory Access

- To avoid **programmed I/O** (move one byte at a time such as keyboard and mouse) for large data movement
- Requires **DMA** controller - bypasses CPU to transfer data between I/O device and memory directly
- OS writes DMA command block into memory
  - Source and destination addresses
  - Read or write mode
  - Number of bytes to be transferred
  - Writes location of command block to DMA controller
  - Bus mastering of DMA controller – grabs bus from CPU
  - When done, send interrupt to CPU for signaling completion

Single-Processor Systems

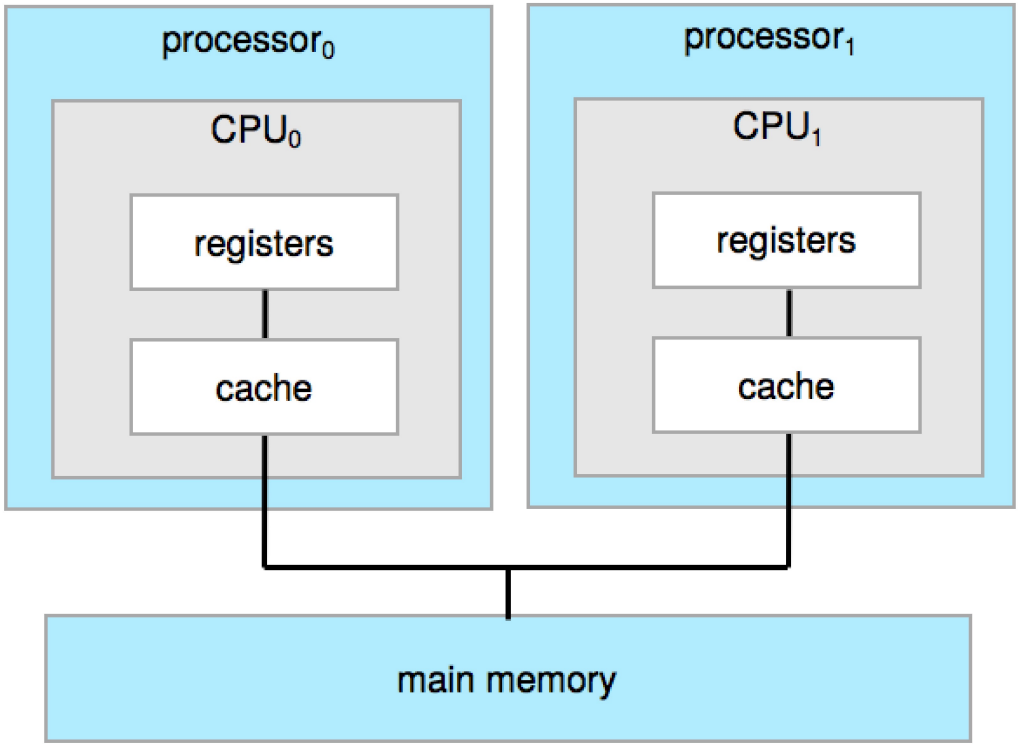
- In the past, most computer systems used a single processor containing one CPU with a single processing core
  - The core executes instructions and registers for storing data locally.
  - The processing core or CPU core is capable of executing a general-purpose instruction set, including instructions from processes.
- Such systems usually have other special-purpose processors - device-specific processors, such as disk, keyboard, and graphics controllers (GPU).
  - They run a limited instruction set, do not execute instructions from processes

Multiprocessor Systems

- On modern computers, from mobile devices to servers, multiprocessors systems now dominate the landscape of computing
  - Traditionally, such systems have two (or more) processors, each with a single-core CPU
  - The speed-up ratio with N processors is less than N, because of overhead, e.g., contention for shared resources (bus or memory)
- Multiprocessors systems growing in use and importance, **advantages** are
  - Increased throughput – more computing capability
  - Economy of scale – share other devices such as I/O devices
  - Increased reliability – graceful degradation or fault tolerance
- Two types of multiprocessor systems
  - Asymmetric Multiprocessing –often master-slave manner, the master processor assign specific tasks to slaves, and the master handles I/O
  - Symmetric Multiprocessing – each processor performs all tasks, including operating-system functions and user processes

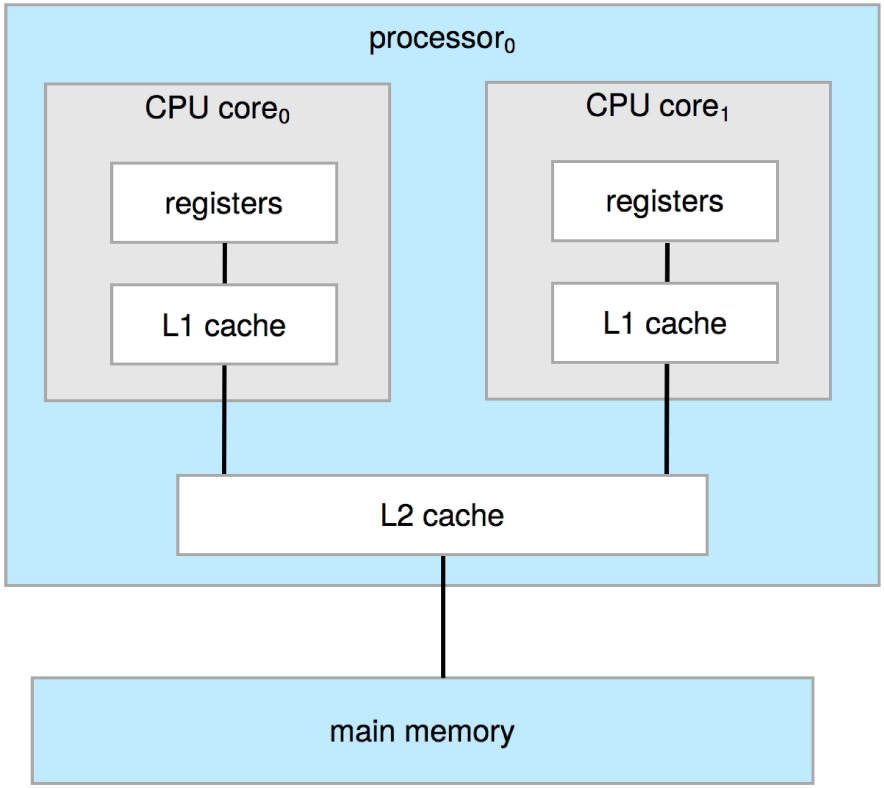
Symmetric Multiprocessor Systems

- Symmetric Multiprocessing or SMP – each CPU processor has its own set of registers, as well as a private or local cache. However, all processors share physical memory over the system bus.



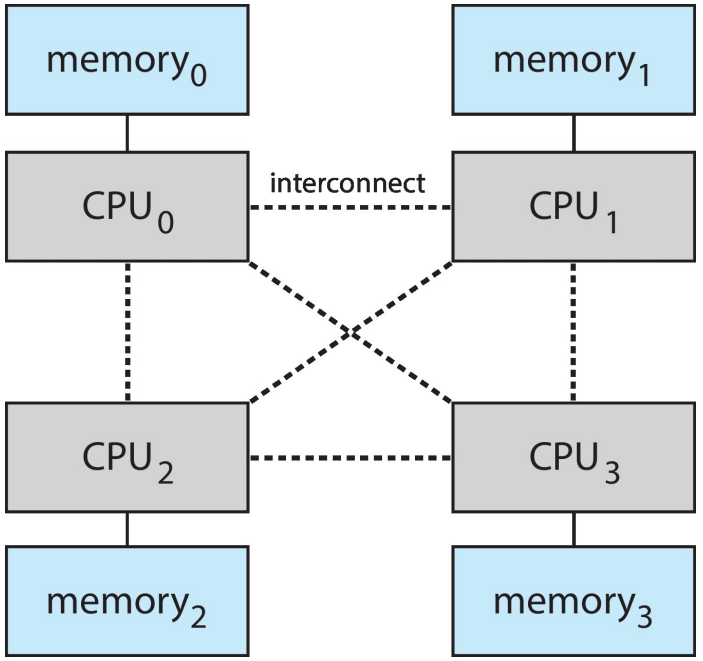
A Multi-Core Design

- The **multicore** system, multiple computing cores reside on **a single chip**
  - Faster on-chip communication than between-chip communication
  - Uses significantly less power - important for mobile devices and laptops



Non-Uniform Memory Access (NUMA)

- Adding more CPUs to a multiprocessor system may not scale, due to the contention for system bus, which can become a bottleneck
- An alternative is to provide each CPU (or group of CPUs) with its own local memory that is accessed via a small, fast local bus.
- The CPUs are connected by a shared system interconnect, and all CPUs share one physical address space.
- This approach—known as non-uniform memory access or NUMA
- The potential drawback with a NUMA system is increased latency when a CPU must access remote memory across the system interconnect – scheduling and memory management implication



Computer System Component

- **CPU** - The hardware that executes instructions
- **Processor** - A physical chip that contains one or more CPUs
- **Core** – The basic computation unit of the CPU or the component that executes instructions and registers for storing data locally
- **Multicore** – Including multiple computing cores on a single physical processor chip
- **Multiprocessor system**– including multiple processors
  - Traditionally, such systems have two (or more) processors, each with a single-core CPU
  - This evolves with multiple computing cores reside on a single processor chip
- A multicore system is a multiprocessor system, but a multiprocessor system is not necessarily a multicore system

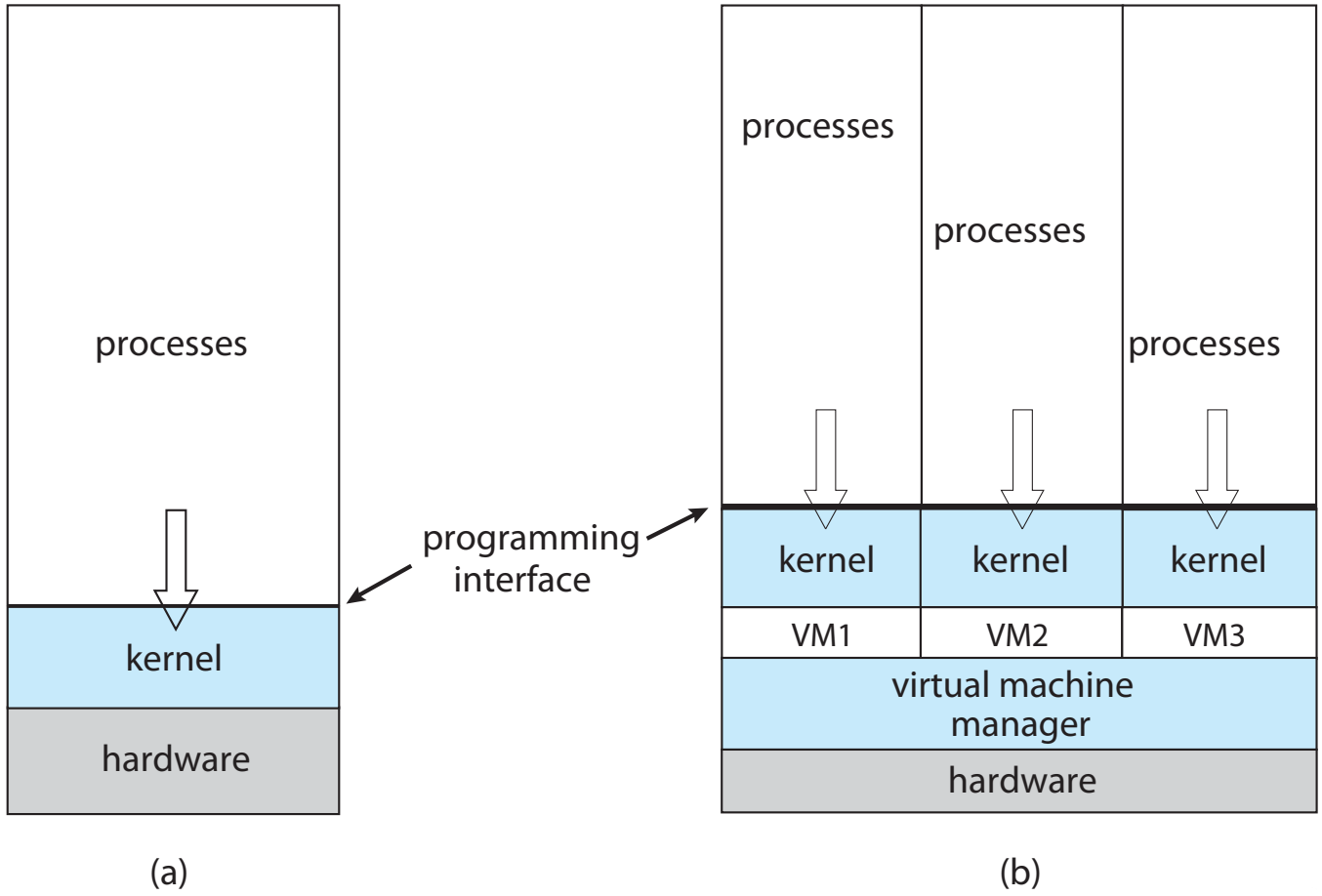
Operating System Structure

- Multiprogramming is needed for efficiency
  - Single user cannot keep CPU and I/O devices busy at all times – all modern computer systems are multi-programmed
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute – in mainframe computers
  - A subset of total jobs in system is kept in memory
  - One job selected and run via job scheduling – load into the memory
  - When it has to wait (for I/O for example), OS switches to another job
- Timesharing (multitasking) is logical extension in which CPU switches jobs frequently that users can interact with each job while it is running, creating interactive computing
  - Response time should be < 1 second
  - Each user has at least one program executing in memory ⇨process
  - If several jobs ready to run at the same time ⇨ CPU scheduling
  - If processes donot fit in memory, swapping moves them in and out to run
  - Virtual memory allows execution of processes not completely in memory

Virtualization

- Virtualization abstracts the hardware of a single computer into different execution environments - creating an illusion that each is running on its own “private computer”
  - The layered design creates a virtual system (virtual machine or VM) on which operation systems or applications can run
  - It allows operating systems to run as applications within other operating system(s) - vast and growing industry
- Several components
  - **Host** – underlying hardware system
  - **Virtual machine manager** (**VMM**) or **hypervisor** – creates and runs virtual machines by providing interface that is identical to the host
  - **Guest** – process provided with virtual copy of the host, usually an operating system
- Single physical machine can run multiple operating systems concurrently, each in its own virtual machine

Virtualization – System Models



Virtual Machine Implementation

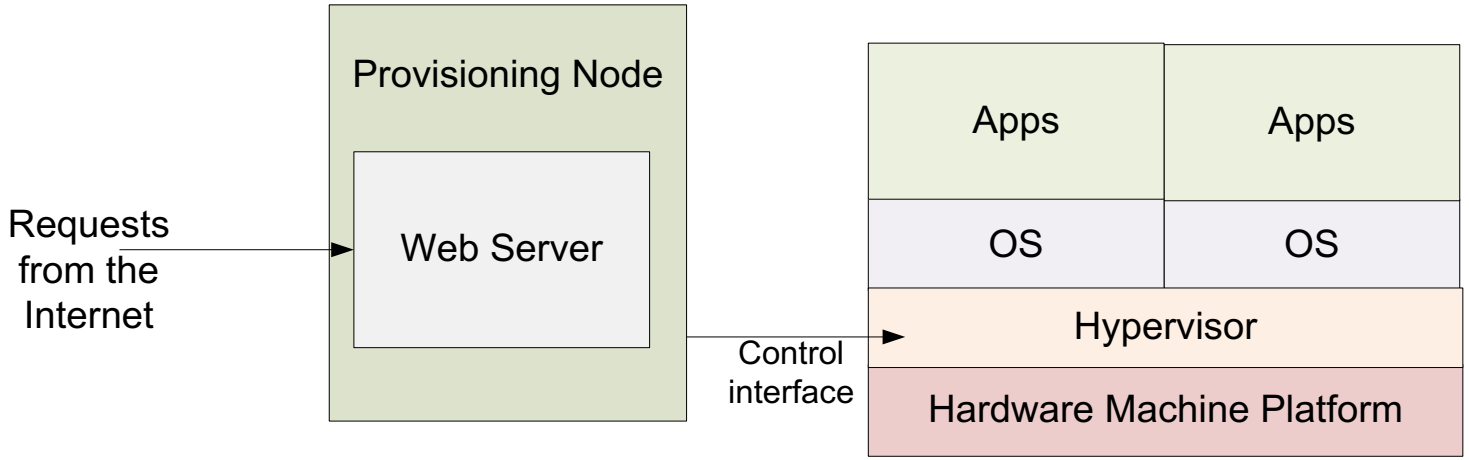
- VMM implementations vary greatly
  - **Type 0 hypervisors** - Hardware-based solutions that provide support for virtual machine creation and management via firmware such as IBM LPARs and Oracle LDOMs are examples
  - **Type 1 hypervisors** - Operating-system-like software built to provide virtualization like VMware ESX, Joyent SmartOS, and Citrix XenServer
  - **Type 2 hypervisors** - Applications that run on standard operating systems but provide VMM features to guest operating systems including VMware Workstation and Fusion, Parallels Desktop, and Oracle VirtualBox
- Other options such as **emulation** that simulates computer hardware in software - used when source CPU type different from target CPU type (i.e. IBM PowerPC to Intel x86)
  - Every machine-level instruction on source system translated to the equivalent function on the target system - generally slow

Virtualization – a bit history

- **Virtualization** – OS natively compiled for CPU, running guest OSes
  - Virtualization originally designed in IBM mainframes (1972) to allow multiple users to run tasks concurrently in a system designed for a single user or share a batch-oriented system
  - VMware runs one or more guest copies of Windows, each running its own applications, on Intel x86 CPU
  - A VMM provides an environment for programs that is essentially identical to the original machine (interface)
  - Programs running within the environment show only minor performance decreases
  - The VMM is in complete control of system resources
- In late 1990s Intel CPUs fast enough for virtualization on general purpose PCs
  - **Xen** and **VMware** created technologies, still used today
  - Virtualization has expanded to many OSes, CPUs, VMMs

Cloud Computing and Virtualization

- Delivers computing, storage, and apps as a service over a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon EC2 has millions of servers, tens of millions of VMs, petabytes of storage available across the Internet, pay based on usage



Cloud Computing Types

- Many types of clouds
  - Public cloud – available via Internet to anyone willing to pay
  - Private cloud – run by a company for the company's own use
  - Hybrid cloud – includes both public and private cloud components
  - Software as a Service (SaaS) – one or more applications available via the Internet (i.e., word processor)
  - Platform as a Service (PaaS) – software stack ready for application use via the Internet (i.e., a database server)
  - Infrastructure as a Service (IaaS) – servers or storage available over Internet (i.e., storage available for backup use)

Free and Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary closed-source and proprietary
  - Microsoft Windows is a well-known example of the closed-source approach.
- Started by Free Software Foundation (FSF), which has "copyleft" GNU Public License (GPL)
  - Free software and open-source software are two different ideas
    - http://gnu.org/philosophy/open-source-misses-the-point.html/
  - Free software not only makes source code available but also is licensed to allow no-cost use, redistribution, and modification. Open-source software does not necessarily offer such licensing
- Popular examples include GNU/Linux, FreeBSD UNIX (including core of Mac OS X - Darwin), and Solaris
- Open-source code is arguably more secure, allowing more programmers to contribute, and is certainly a better learning tool

End of Chapter 1