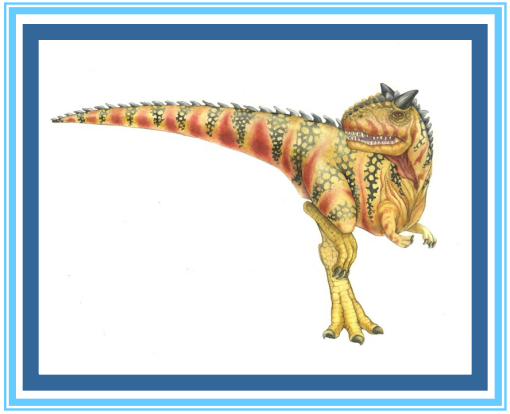
Lecture 1: Introduction



Today's Topics

- Course Introduction
- Class Policy
- Course Syllabus & Objective
- Introduction to Operating System

Course Introduction

- Instructor: Hossein Asadi
- Classes
  - Sat. & Mon.: 9:00AM ~ 10:30AM
    - Attend class on time
  - We may use vclass: https://vc.sharif.edu/ch/asadi
- Office Hours
  - Can be reached by email & skype or stop by my office
  - Email: asadi@sharif.edu
  - Skype account: hossein_asadi
    - Sat. through Wed.: 9AM ~ 6PM
  - Room # 610
- TA Classes
  - Mondays: 12:15PM ~ 1:15PM

Course Introduction (cont.)

- Course Webpage on CW
  - Check this webpage on regular basis
    - At least on Sun, Tue, Thur
    - Q&A only using CW forums
  - Everything will be posted on CW
    - Announcements, handouts, assignments, grades, quiz and exam notices, tutorials, simulators, …
  - Handouts
    - Will be posted a day before class
      - Print it & bring it to class
    - But I may update it after class
      - Check out submission date of handouts

Few Notes on Assignments

- Post All your Questions on CW Forums
  - Check forum history before posting any question
- Be Respectful to your Classmates and TAs
- Harsh Cheating Penalty

Teaching Assistants

TA Administration, Grades, CW, HW Schedules, Comments on TAs	• **Elham Adibi** (Chief TA, my PhD student)
Basic Concepts, Process Management & Process Coordination	• **TBD (Head TA)**
Memory and Storage Management & Misc. Topics	• **TBD (Head TA)** • •
Webpage Administration	• TBD

Textbook

- Operating System Concepts, 8th (or 9th) Edition, Wiley publishing
  - By A. Silberschatz, P. Galvin, & G. Gagne
- Other References:
  - Operating systems: design & implementation, by A. Tanenbaum and A. Woodhull, 3rd edition, 2006.
  - Operating systems: internals and design principles, by W. Stallings, 5th edition, 2005.

Syllabus

- Introduction to Operating Systems
  - OS structure
  - OS operations
- System Structure
  - OS services
  - System calls
  - OS design & implementation
  - Virtual machines

Syllabus (cont.)

- Process Management
  - Process concept
  - Interprocess communication
  - Multithread models
  - Threading libraries & issues
  - Process scheduling
  - Scheduling criteria & algorithms
  - Threading scheduling
  - Synchronization (HW & SW solutions)
  - Semaphores
  - Deadlocks (prevention, avoidance, detection, & recovery)

Syllabus (cont.)

- Memory Management
  - Swapping
  - Memory allocation
  - Paging
  - Page table
  - Segmentation

Syllabus (cont.)

- Virtual-Memory Management
  - Demand paging
  - Copy-on-write
  - Page replacement
  - Allocation of frames
  - Trashing
  - Allocating kernel memory
  - OS examples

Syllabus (cont.)

- Storage Management
  - File system
  - Implementing file systems
  - I/O systems
  - Structure of SSDs and HDDs

Syllabus (cont.)

- Protection & Security
- Real-Time OSs
  - Real-time scheduling
  - Real-time kernels

Class Policy

- Ask Questions Anytime
  - Don’t hesitate to ask even stupid questions!!!
- Cell Phones off or on Silent
- Absence
  - Three sessions allowed
- Food No, Drink yes!
- Feel Free to Pass Me Your Feedbacks
  - Anything related to this course

Class Policy (cont.)

- Assignments
  - Two late assignments will be accepted!
    - Only two days late!
    - Third late assignment (two-day late)
      - HW will be graded out of 50%
    - Forth and next late assignments will not be accepted!
  - Discussions encouraged!
  - But do your own handwriting!
  - Zero score for copied assignments!
    - Second time zero score for 30% share!

Objective

- Understand Major Components of OSs
- Describe Various Ways of Structuring an OS
- Describe Various Features of Processes including Scheduling, Creation, Termination, & Communication
- Understand Thread and its Issues
- Describe Various CPU-Scheduling Algorithms
- Good Understanding of Evaluation Criteria to Select a CPU-Scheduling Algorithm for a System
- Learn Concept of Atomic Transaction
- Learn How to Ensure Consistency of Shared Data

Objective (cont.)

- Learn How to Prevent Deadlocks in a OS
- Learn Various Ways of Organizing Memory
- Understand Various Memory Management Techniques
- Learn Virtual-Management Techniques
- Understand Functions and Interface of File Systems
- Be Able to Explain Structure of an OS I/O Subsystem
- Understand Principles of Protection in OS
- Be Able to Enumerate Security Threats in OS

Grading

- Midterm Exam: 25%
  - Aban16th
- Final Exam: 30% (as posted in EDU)
- Random (Unscheduled) Quizzes: 10%
- Assignments & Project: 40%
  - Bonus points for outstanding projects
- Exams Covers Topics of Lecture Classes and TA Classes

Copyright Notice

- Slides Mainly Adopted from Suggested Slides of Main Textbook.

Lecture 1: Introduction

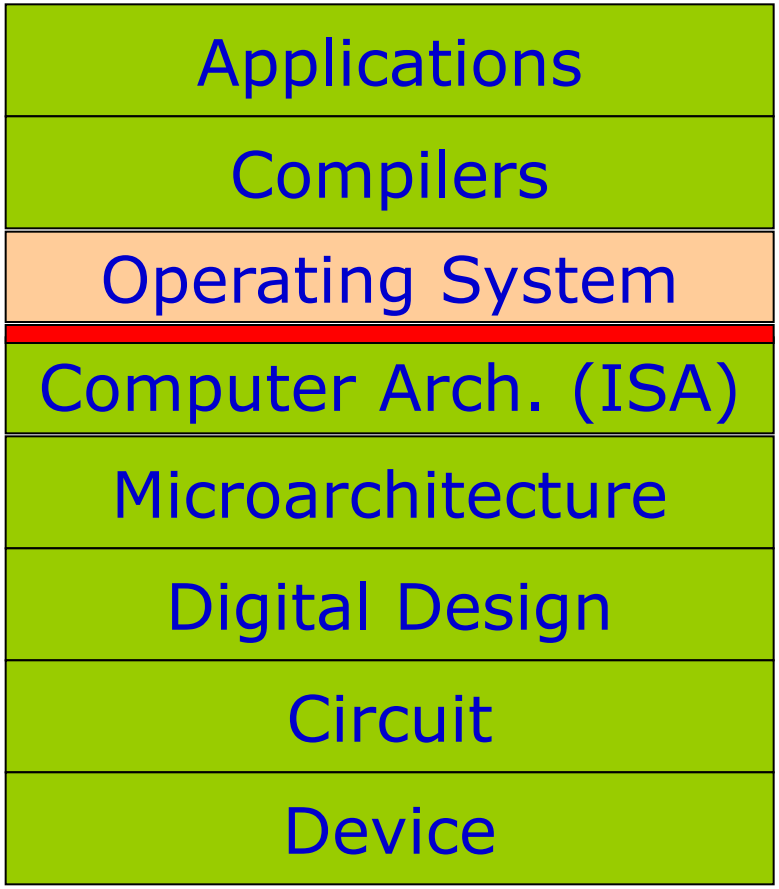
- Quick Overview and Review
  - What operating systems do
  - Computer-system organization
  - Computer-system architecture
  - Operating-system structure
  - Operating-system operations
  - Process management
  - Memory management
  - Storage management
  - Protection and security
  - Open-source operating systems

Objectives of this Lecture

- To Describe Basic Organization of Computer systems
- To Provide a Grand Tour of Major Components of Operating Systems
- To Review Open-Source Operating Systems

General Picture

- Computer Abstraction Levels



Almost Nearest SW Layer to ISA

What is an Operating System?

- A Program that Acts as an Intermediary between a User of a Computer and Computer HW
  - Acts as a governor
- Operating System Goals:

Execute user programs and make solving user problems easier	**Allow multiple programs to simultaneously be executed in a shared HW**
Use computer HW in an efficient manner	**Make computer system convenient to use**

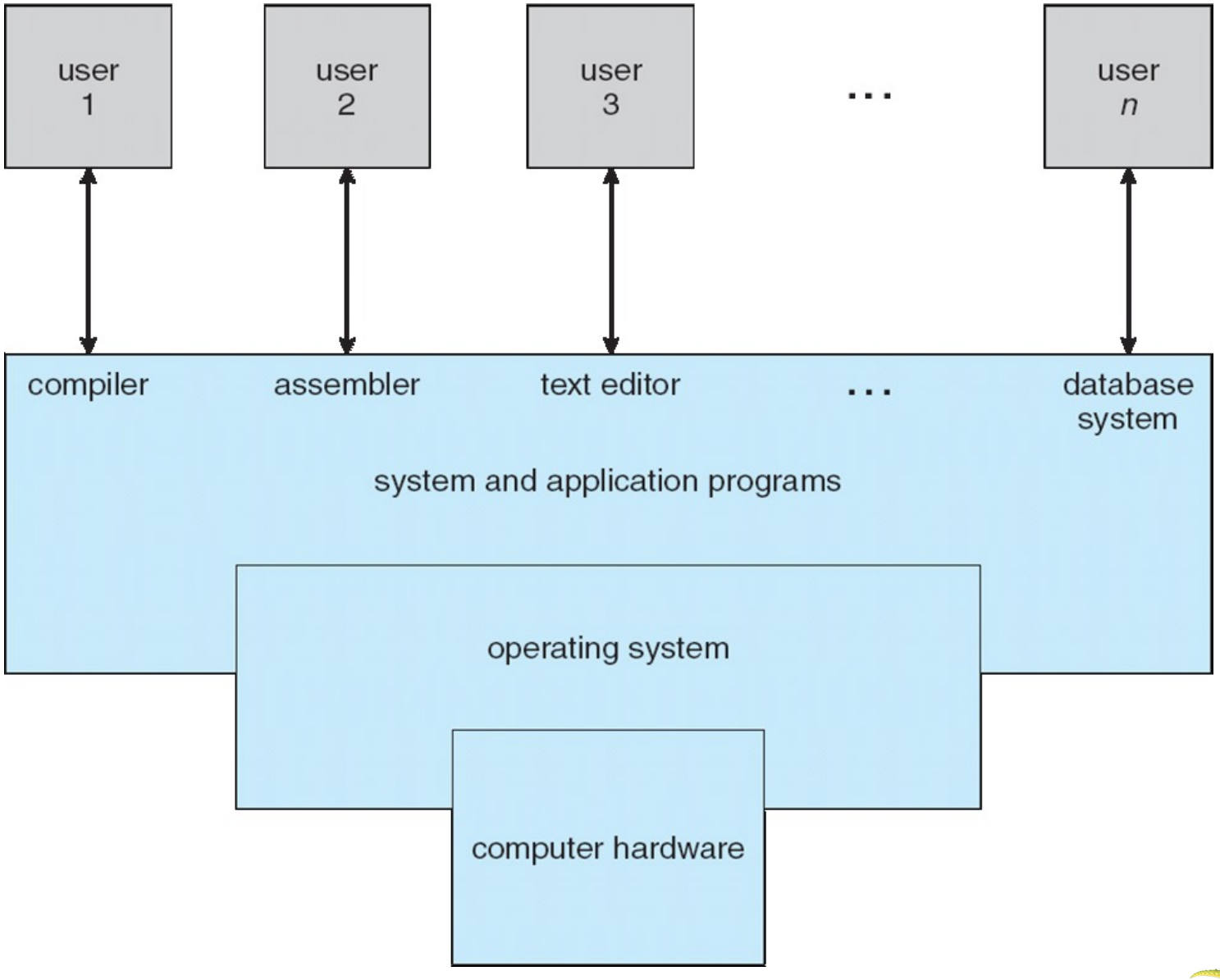
OS: Mandatory or Optional?

- Important Question:
  - Can a computer run without an operating system?
- Answer:
  - Yes, earliest computers didn't have any OS
- Another Question:
  - What does a computer without an OS look like?
- Answer:
  - Machines tasked with one program at a time
    - Cannot look at a doc and keep a clock running on your desktop
  - Each program has a lot of work to do
    - Where to load a program
    - IO access

Computer System Structure

- HW
  - Provides basic computing resources
  - CPU, memory, storage, and I/O devices
- OS
  - Controls and coordinates use of HW among various applications and users
- Application & System Programs
  - Define ways in which system resources used to solve computing problems of users
  - Word processors, compilers, web browsers, database systems, video games
- Users
  - People, machines, other computers

Components of a Computer System



The Role of Operating Systems

- Let's Start Designing & Implementing an Operating System
- Important Question:
  - Major parameters in which we would like to optimize OS towards to?
- To Answer this Question:
  - We need to take a look at the role of OS from two perspectives

The Role of Operating Systems (cont.)

- **Depends on the Point of View**
  - Users view or system view
- Users want Convenience, **Ease of Use**
  - **Throughput** not important
  - **Latency** is, however, of importance
  - **Don't care** about **resource utilization**
- Shared Computer such as **Mainframe** or **Minicomputer** must keep all users happy
  - Resource utilization
  - Fair share between all users

The Role of Operating Systems (cont.)

- **Also Depends on Type of System Usage**
  - Embedded, desktop, high-performance computing
    - Objectives: power, performance, ease of use, utilization, real-time, reliability, etc
  - Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
    ➔ Compromise between individual usability & resource utilization
  - Handheld computers are resource poor, optimized for usability and battery life
  - Some have Little or No User Interface
    - E.g. embedded computers in devices & automobiles

The Role of OS: System View

- OS is a **Resource Allocator**
  - Possible resources include CPU time, memory space, file-storage space, I/O devices, etc.
  - Manages all resources
  - How, when, and for how long allocate resources to programs and users
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **Control Program**
  - Controls execution of programs to prevent errors and improper use of computer

Operating System Story

- Fundamental Goal of a Computer System
  - Execute user programs and to make solving user problems easier
- ➔ Computer HW Constructed
  - HW alone not easy to use
  - ➔ Application programs developed
  - Application programs require certain common operations such as those controlling I/O devices
- Finally:
  - Common functions of controlling and allocating resources brought together into one piece ➔ called OS

Operating System Definition

- No Universally Accepted Definition
- "Everything a Vendor Ships when you Order an OS" is a Good Approximation
  - But varies wildly (from 1MB to GBs)
- "The one Program Running at All Times on the computer" is **Kernel**.
- Everything else is either
  - A system program (ships with OS)
    - File management programs, loader, linker, compiler
  - An application program

Computer Startup

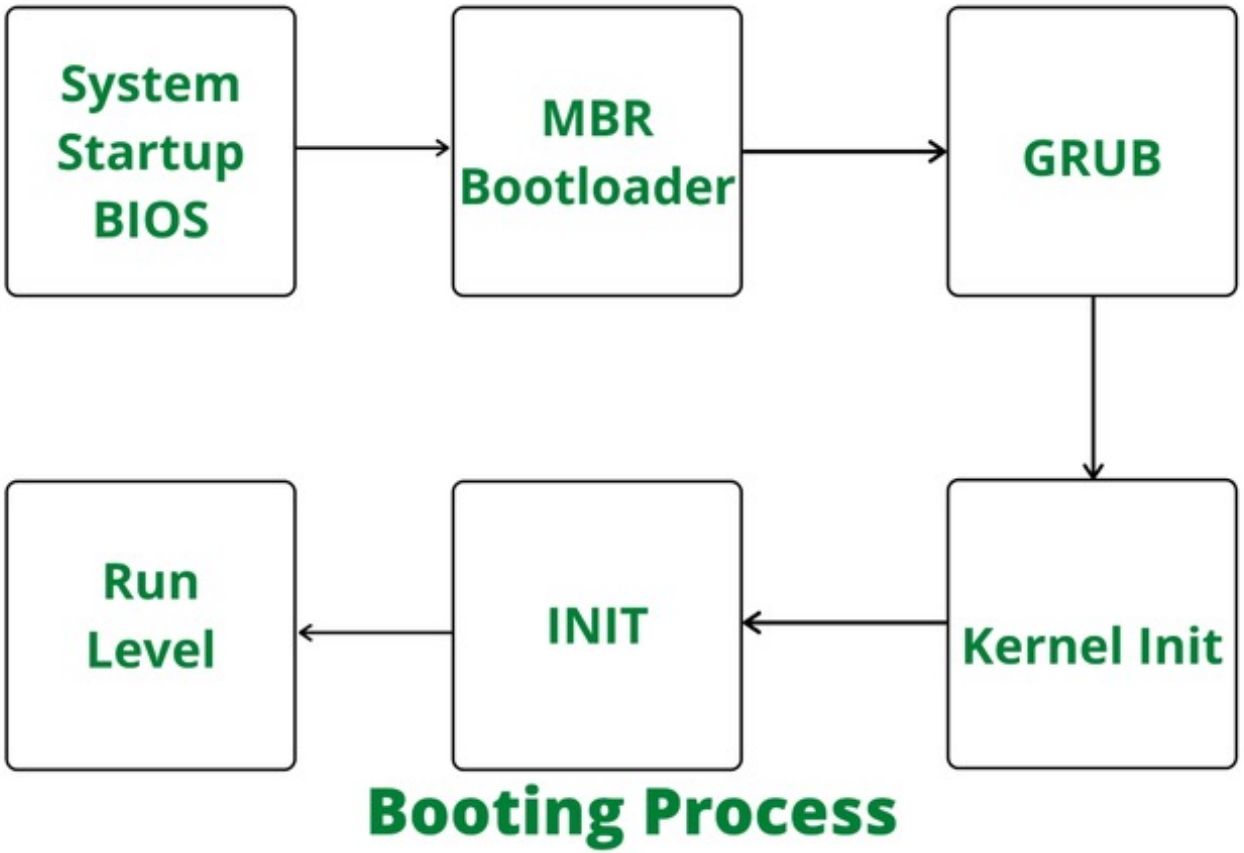
- **Question:**
  - Can we think of any SW layer lower than OS?
- **Another Question:**
  - OS resides in disk. How it can be executed?
  - Remember stored program concept

Computer Startup (cont.)

- **Bootstrap Program**
  - Loaded at Power-Up/Reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of a system
    - Such as CPU registers, device controllers
  - Loads OS kernel and starts execution
  - OS then starts executing first process

Computer Startup (cont.)

Bootstrap Program (Linux Example)



© https://www.geeksforgeeks.org/how-linux-kernel-boots/

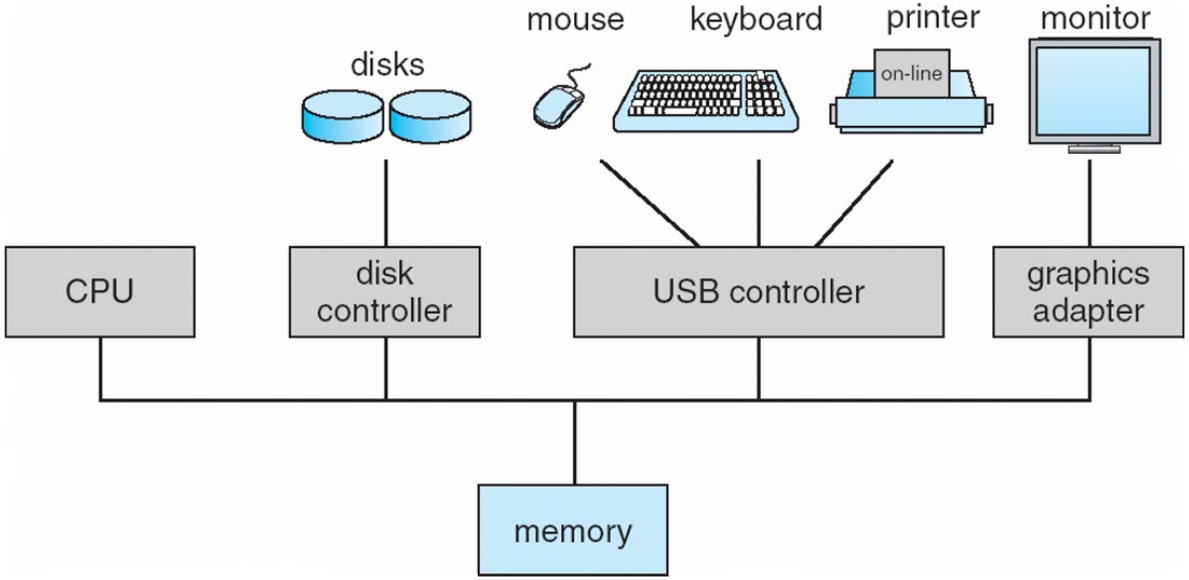
The Rest of This Lecture

- An Overview on Computer Structure
  - Computer system organization
  - Interrupt
  - Storage structure
  - Caching
  - DMA
  - BMC
  - Computer system architecture
    - Multi-processors

Computer System Organization

Computer-System Operation

- One or more CPUs, device controllers connect through common bus providing access to shared memory
- Concurrent execution of CPUs and devices competing for memory cycles



Computer-System Operation

- Each Device Controller is in Charge of a Particular Device Type (e.g., disk drives, audio devices)
- Each Device Controller has a Local Buffer
- I/O: Device ⟷ Local Buffer of Controller
- Device Controller Informs CPU
  - That it has finished its operation by causing an interrupt
- CPU Moves Data
  - Main memory ⟷ Local buffers

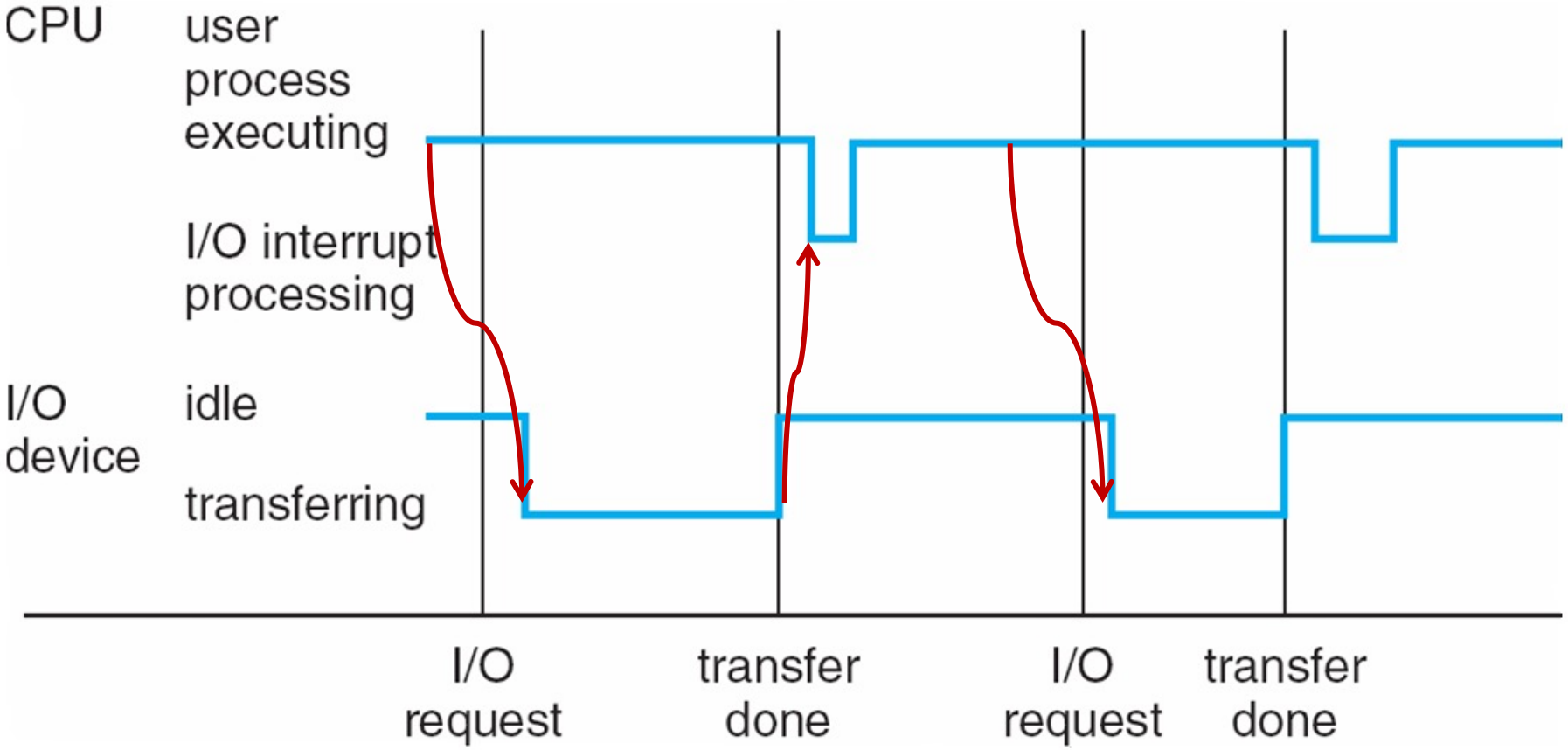
Common Functions of Interrupts

- Interrupt Transfers Control to Interrupt Service Routine
  - Generally, through **interrupt vector**, which contains addresses of all service routines
- Must Save Address of Interrupted Instruction
- A **Trap** or **Exception**
  - A SW-generated interrupt caused either by an error or a User Request (aka, system call)

Interrupt Handling

- OS Preserves State of CPU by Storing Registers and Program Counter (PC)
- Determines which Type of Interrupt has Occurred
- Different Segments of Code Determine what Action Should be Taken for Each Type of Interrupt

Interrupt Timeline



Storage Structure

- **Main Memory** – only large storage media that CPU can access **directly**
  - **Random access** and typically **volatile**
- Secondary Storage
  - Extension of main memory that provides large **nonvolatile** storage capacity
  - Hard Disk Drives (HDDs)
  - **Solid-State Drives (SSD)**
    - Faster than HDDs, non-volatile
- Tertiary Storage
  - Optical disk
  - Magnetic tape

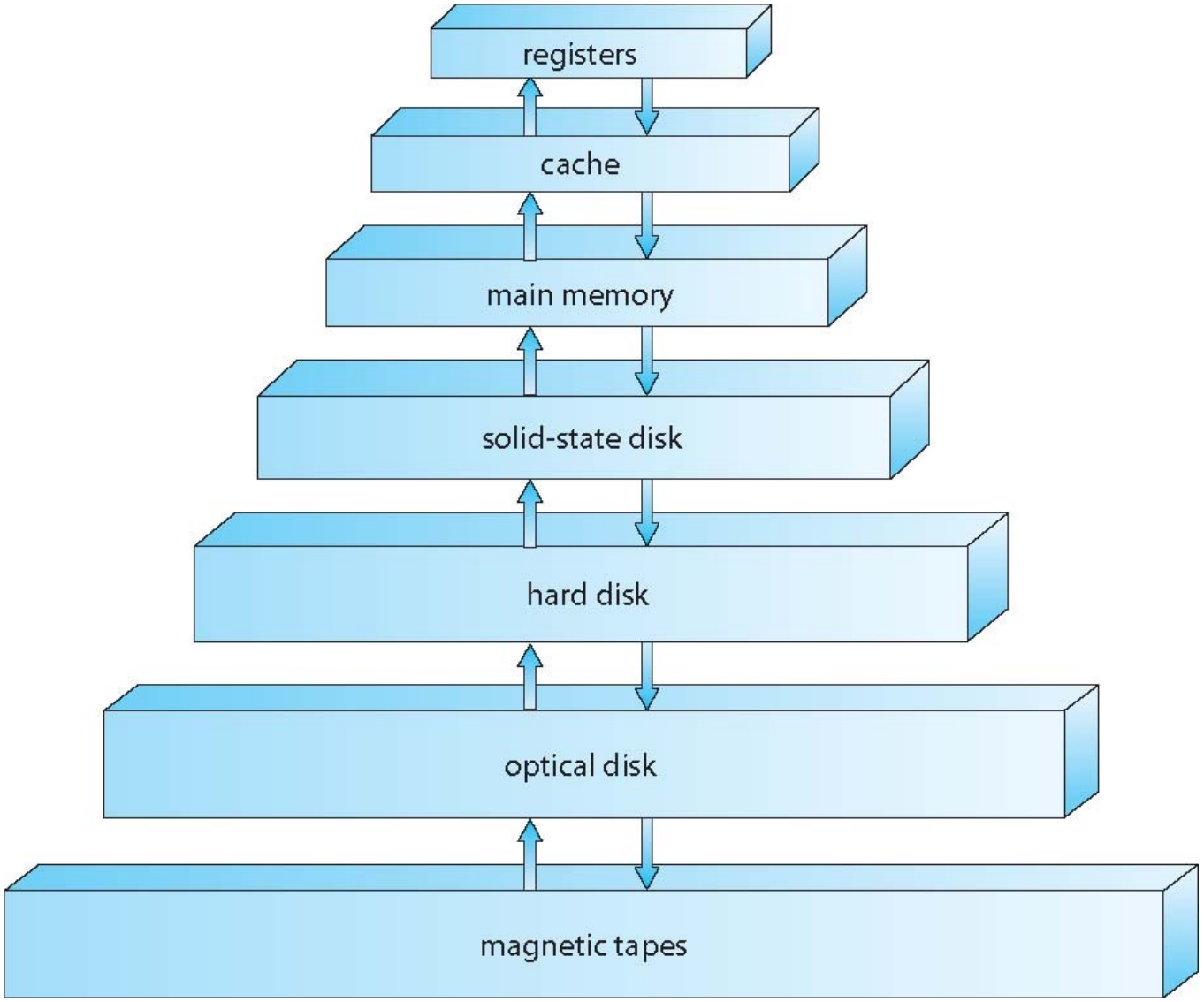
Storage Hierarchy

- Storage Systems Organized in Hierarchy
  - Ideal storage ➔ inexpensive, fast, non-volatile
  - Ranked based on Speed, cost, & volatility
- **Caching**
  - Information in use copied from slower to faster storage temporarily
  - Performed at various levels: HW, OS, & application
  - Example: main memory as a cache for secondary storage

Caching

- Faster Storage (cache) Checked First
  - To determine if information is there
  - **If it is**, information used directly from cache (fast)
  - **If not**, data copied to cache and used there
- Cache is Smaller than Storage being Cached ➔ Issues?
  - Cache management important design problem
  - Cache size and replacement policy

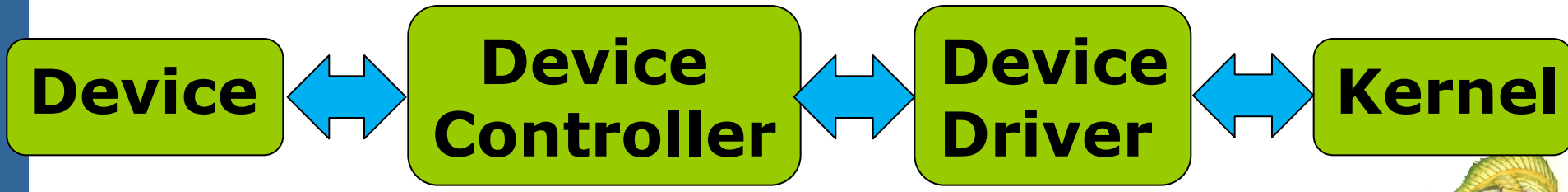
Storage-Device Hierarchy



I/O & Device Driver

Device Driver

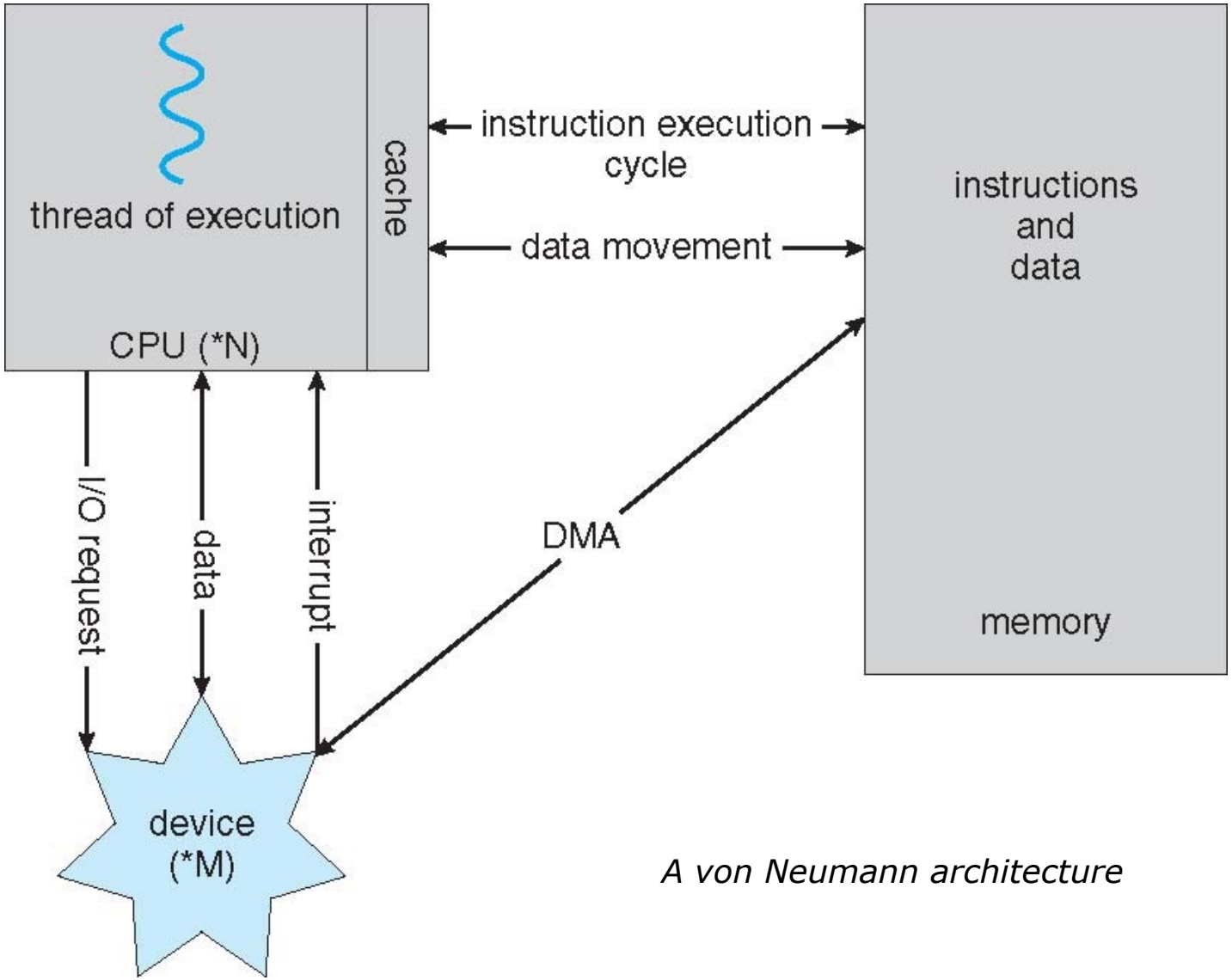
- Alleviates complexity of I/O devices from Kernel
- For each device controller to manage I/O
- Provides uniform interface (controller ⇔ kernel)
- Device ⇐⇒ Device controller (local buffer)
- Device controller ⇐⇒ Device driver
  - Device driver is informed by an interrupt
- Device driver ⇐⇒ Kernel



Direct Memory Access Structure

- Used for High-Speed I/O Devices
  - Able to transmit info at close to memory speeds
- Device Controller Transfers Data Blocks
  - From buffer storage directly to main memory without CPU intervention
- Only one Interrupt is Generated per Block
  - Rather than one interrupt per byte

How a Modern Computer Works

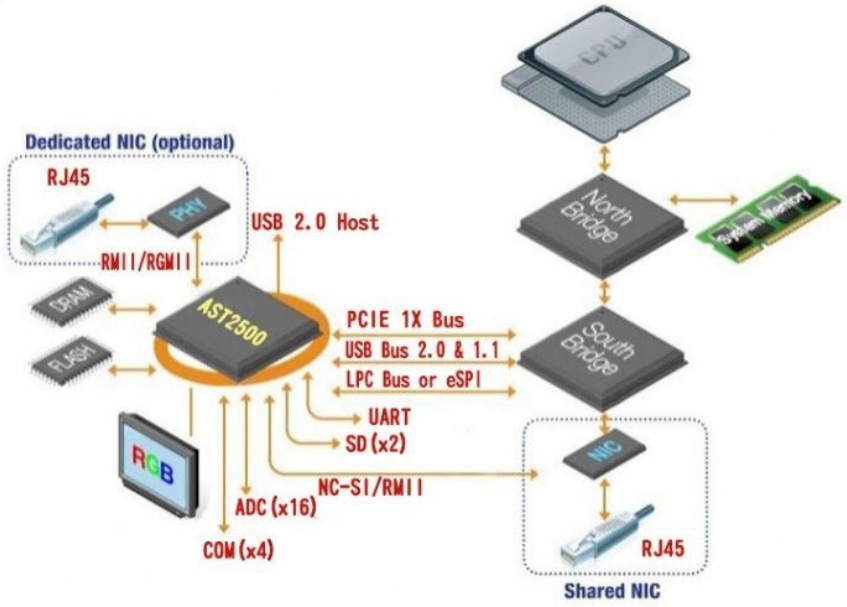


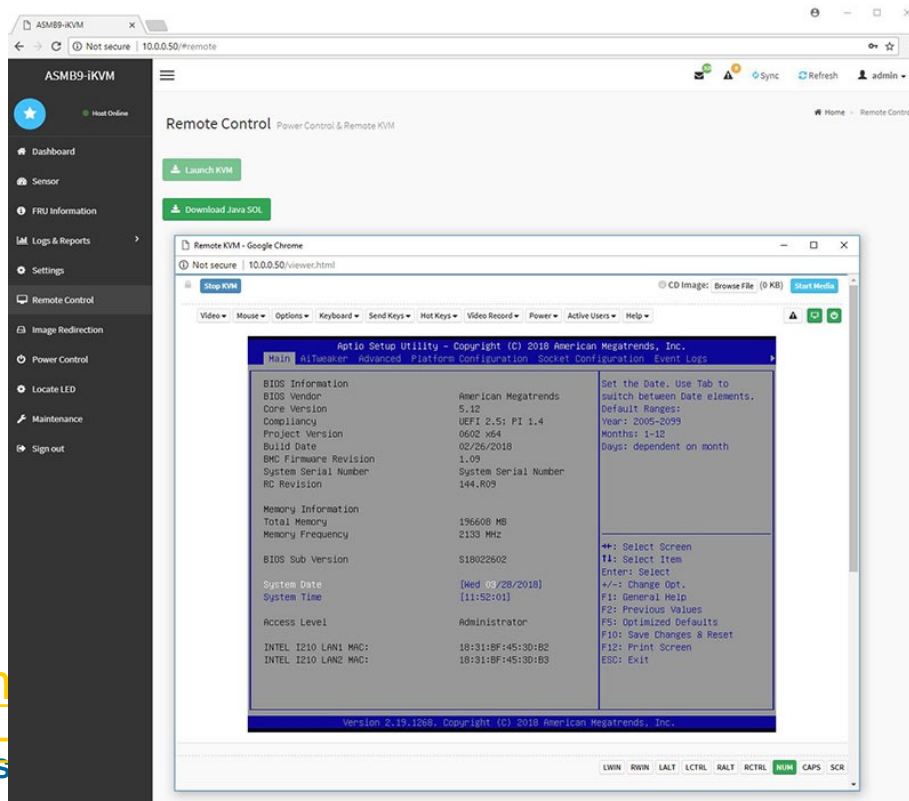
A von Neumann architecture

Baseboard Management Controller (BMC)

BMC Role

- Small computer that sits on every server MB
- Typically ARM-based SoC with GUI and built-in logic





© https://www.servethehome.com
management-controller-or-bmc-in-

Computer-System Architecture

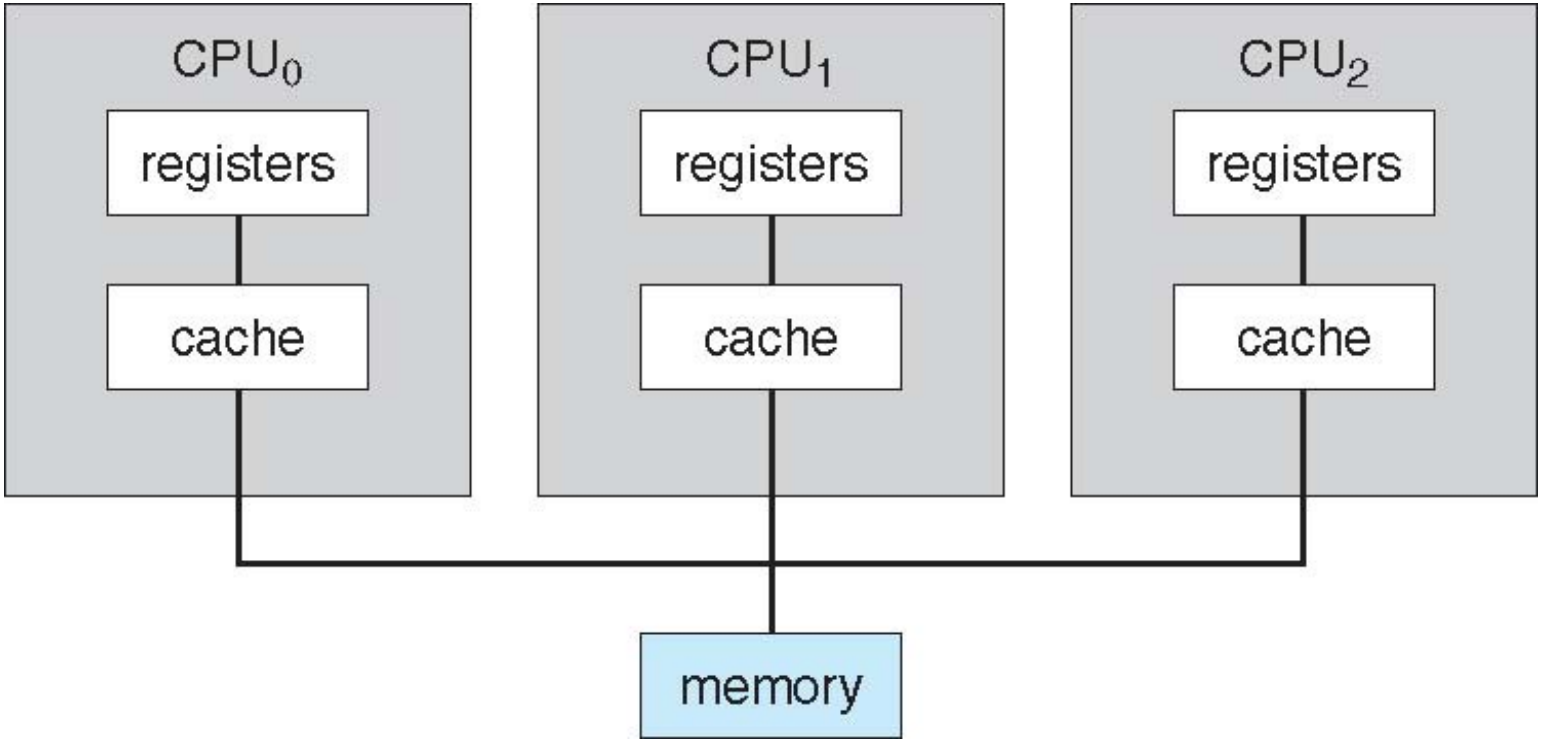
- Single General-Purpose Processor
  - Commonly used in embedded and thin clients
  - Most systems have special-purpose processors as well
- **Multiprocessors** Systems
  - Growing in use and importance
  - Mainly used in server-class systems
  - A.k.a **parallel systems**, **tightly-coupled systems**
  - Advantages include:
    1. **Increased throughput**
    2. **Economy of scale**
    3. **Increased reliability** – graceful degradation or fault tolerance

Computer-System Architecture (cont.)

Multiprocessors Systems

- **Asymmetric Multiprocessing**
  - Each processor is assigned a specific task
  - Master/slave model
  - Master: schedules and allocates work to slave processors
- **Symmetric Multiprocessing (SMP)**
  - Each processor performs all tasks
  - All processors are peers
  - No Master-slave relationship

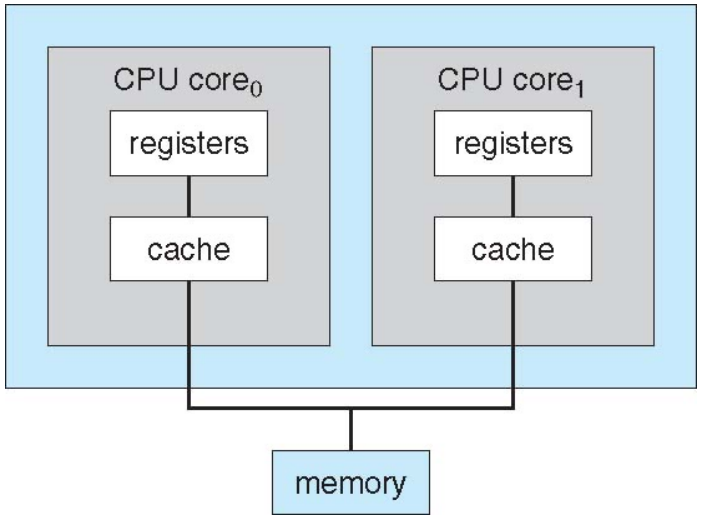
Symmetric Multiprocessing Architecture



- N CPUs ➔ N concurrent processes
- Share physical memory

A Dual-Core Design

- **Multi-chip** vs. **Multicore**
- **Early Trend: Multi-Chip Processors**
- **Current Trend: Multicore**
  - **Faster communications between cores**
  - **More energy efficiency**
  - **No difference from OS perspective**



Back to OS Concepts

- Multi-Programming vs. Multi-Tasking
- Interrupt Driven OS
- Dual Mode in OS
- Process Hogging (Infinite Loop)

Operating System Structure

- **Multiprogramming (Batch system)**
  - Aimed at improving system efficiency
  - 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
  - Job pool: all jobs waiting to be executed reside either in disk or memory
    - A subset of total jobs in system is kept in memory
  - When it has to wait (for I/O, e.g.), OS switches to another job
    - Non-multiprogrammed system ➔ CPU sits idle

Operating System Structure (cont.)

- **Multiprogrammed System**
  - Provides environment to effectively utilize resources
  - Does not provide user interaction
- **Timesharing (Multitasking)**
  - Logical extension of multi-programmed
  - CPU switches jobs so frequently that users can interact with each job ➔ creating **interactive** computing
  - **Response time** should be < 1 second
    - Each user is given impression that entire computer system is dedicated to his/her use

Operating System Structure (cont.)

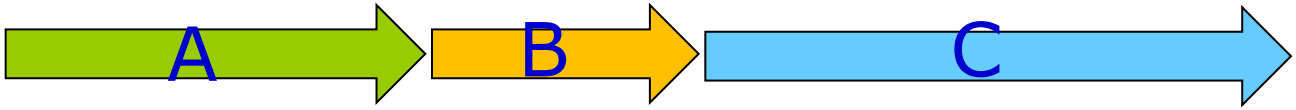
- **Timesharing (Multitasking)**
  - **Process:** A program loaded into memory and executing
    - A process executes for only a short time before it either finishes or needs to perform I/O
- Fact:
  - Time sharing and multiprogramming requires several jobs be kept in main memory
  - Job scheduling: Decides which job to be brought to memory (Lec. 5)
  - CPU scheduling: Decides which job to be executed

Operating System Structure (cont.)

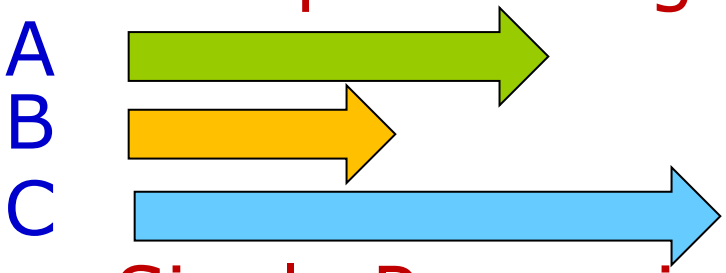
- **Timesharing (Multitasking)**
  - If processes don't fit in memory, **swapping** moves them in and out to run (Lec. 8)
  - **Virtual memory** allows execution of processes not completely in memory (Lec. 9)
    - Separates logical memory (as viewed by users) from physical memory

Multiprogramming vs. Multitasking vs. Multiprocessing

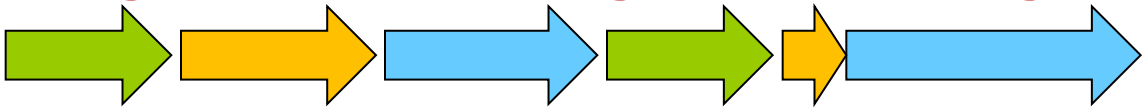
Single Processing Non-Multi-programmed



Multiprocessing

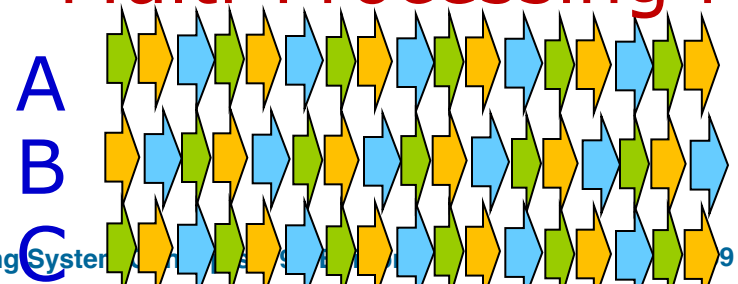


Single Processing Multi-Programming

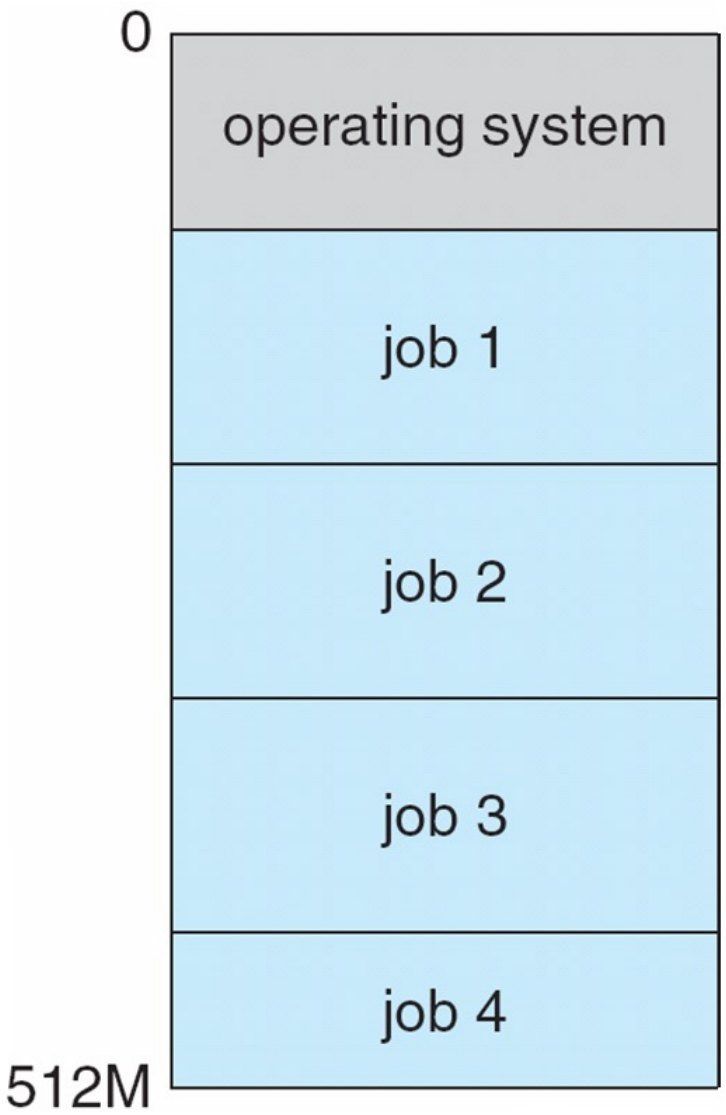


Single Processing Multi-Tasking

Multi-Processing Multi-Tasking



Memory Layout for Multiprogrammed System



Operating-System Operations

- **Modern OS are Interrupt Driven**
  - No process to execute ➔ No I/O devices to service and no users to whom to respond ➔ OS will sit **idle** and wait for sth to happen, i.e., interrupt
  - Events are signaled by occurrence of an interrupt
- **Interrupts**
  - Hardware driven
  - Software driven

Operating-System Operations (cont.)

- Hardware Interrupt by One of Devices
  - Disk drive, NIC, printer, and others
- Software Interrupt (**Exception** or **Trap**)
  - Software error
    - Division by zero
    - Invalid memory access
    - Access to kernel address space
  - Request for OS service from a user program
  - Other process problems
    - Infinite loop
    - Processes modifying each other or OS

Operating-System Operations (cont.)

- **Dual-Mode** Operation Allows OS to Protect itself and Other System Components
  - **User mode** and **kernel mode**
    - **Kernel mode: aka, supervisor, system, privileged mode**
- **Mode Bit**
  - Provided by HW: kernel (0) or user (1)
  - Provides ability to distinguish when system is running user code or kernel code
- System call
  - Changes mode to kernel, return from call resets to user
  - At system boot time: HW starts in kernel mode

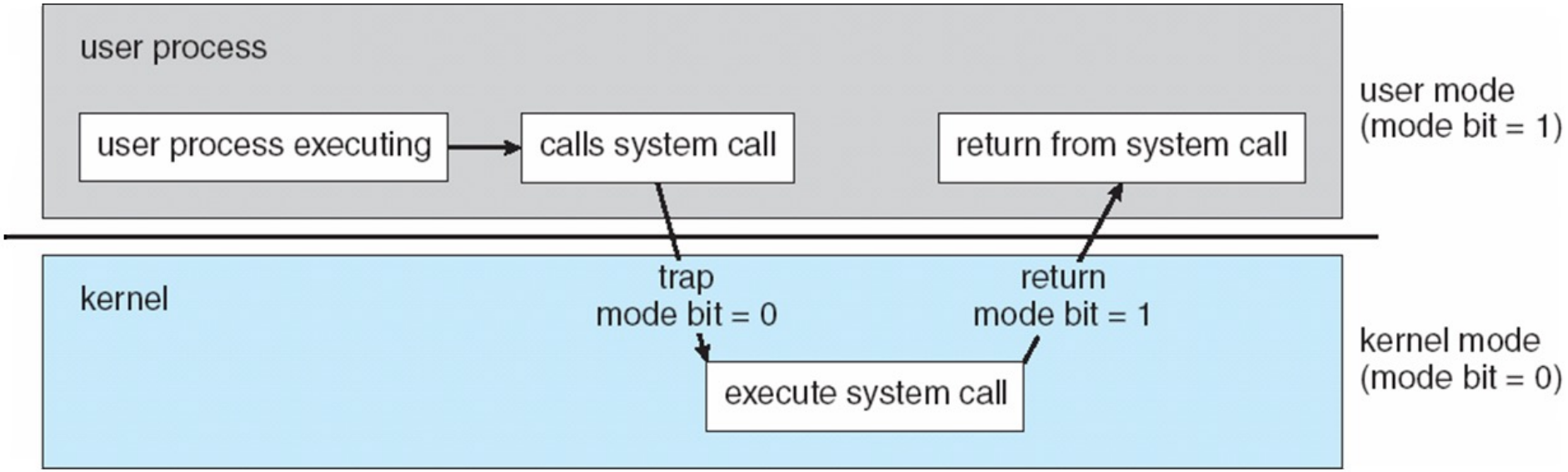
Operating-System Operations (cont.)

- **Main Aim of Dual-Mode Operation**
  - Protecting OS from errant users
  - Some instructions may cause harm ➔ designate such instructions as privileged instructions
    - Only executable in kernel mode
  - Privileged instructions cannot be executed in user mode
    - Treated as illegal and traps it to OS interrupts
  - Examples of privileged instructions
    - ❑Switching to kernel mode
    - ❑I/O control
    - ❑Setting the timer
    - ❑Interrupt management
    - ❑Clearing memory
    - ❑Remove process from memory

Operating-System Operations (cont.)

- Flow of a System Call
  - Treated as a SW interrupt
  - ➔ Control passes through interrupt vector to a service routing in the OS
  - ➔ Mode bit = 0 (kernel mode)
  - ➔ Kernel examines interrupting instruction to determine type of interrupt
  - ➔ Additional information may be found in RF or stack
  - ➔ Kernel verifies passed parameters
  - ➔ System call is executed
  - ➔ Control passed to instruction following system call
  - ➔ Mode bit =1 (user mode)

Operating-System Operations (cont.)



- Questions:
  - **Is a system call instruction to switch to kernel mode a privileged or non-privileged instruction?**
  - **Is a direct instruction to switch to kernel mode a privileged or non-privileged instruction?**

(c) https://www.codingninjas.com/codestudio/library/privileged-and-non-privileged-instructions

Operating-System Operations (cont.)

- Lack of HW-Supported Mode
  - ➔ Can cause serious shortcomings in OS
  - MS-DOS written for 8088 both has no mode bit and no dual mode
- Possible Scenarios in Absence of Dual-Mode
  - User program can wipe out OS
  - Multiple programs can write to a device at same time
- Recent CPUs and OSs Support Dual-Mode
  - Linux, Unix, Solaris, Win-XP, Win-Vista
- Increasingly CPUs Support Multi-Mode Operations
  - **Virtual Machine Manager (VMM)** mode for guest **VMs**

Transition from User to Kernel Mode

- Timer to Prevent Infinite Loop / Process Hogging Resources
  - Timer is set to interrupt computer after some time period
  - Keep a counter that is decremented by physical clock
  - OS sets counter (privileged instruction)
  - Counter = 0 ➔ Generates an interrupt
  - Set up timer before scheduling process to regain control or terminate program that exceeds allotted time

A Quick Preview of Concepts

- Process Management
- Memory Management
- Storage Management
- Protection & Security
- Open Source OSes

Process Management

- A Process is a Program in Execution
  - Unit of work within a system
  - ***Active entity*** vs. program which is ***passive entity***
- Process Needs Resources to Accomplish its Task
  - CPU, memory, I/O, files
  - Initialization data
- Process Termination Requires Reclaiming any Reusable Resources

Process Management Activities

- OS Responsible for following Activities
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling

Memory Management

- Main Memory
  - Repository of quickly accessible data shared by CPU and I/O devices
  - The only large storage device that CPU is able to address and access directly
- To Execute a Program
  - All (or part) of Instructions must be in Memory
  - All (or part) of Data needed by program must be in memory
- To Improve Utilization of CPU
  - Several programs must be kept in memory ➔ needs memory management

Memory Management (cont.)

- Memory Management
  - Determines what is in memory and when
  - Optimizing CPU utilization and computer response to users
- Management Activities
  - Keeping track of which parts of memory are currently being used and by whom
  - Deciding which processes (or parts thereof) and data to move into and out of memory
  - Allocating and deallocating memory space as needed

Storage Management

- OS Provides Uniform, Logical View of Information Storage
  - To make computer system convenient for users
  - Abstracts physical properties to logical storage unit, aka, **file**
  - Each medium is controlled by a device driver (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)

Mass-Storage Management

- Why Mass Storage Needed?
  - Main memory too small to accommodate all data/programs
  - Main memory is volatile ➔ data will be lost upon power outage
- Disks usually Used to Store Data
  - That does not fit in main memory or data that must be kept for a “long” period of time
- Proper Management is of Central Importance
- **Entire Speed** of Computer Operation Hinges on Disk Subsystem and its Algorithms

Mass-Storage Management (cont.)

- OS Activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some Storage Need Not be Fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed – by OS or applications
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

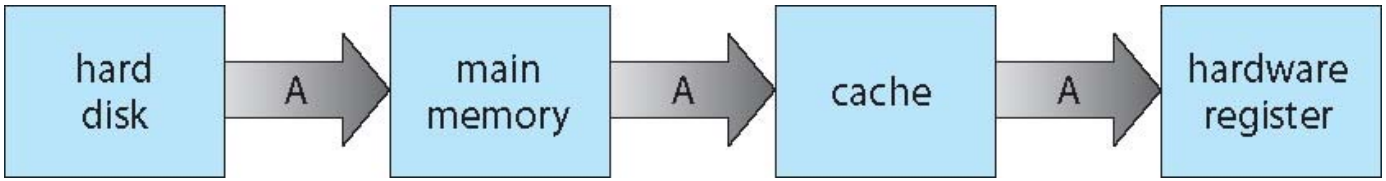
Performance of Various Levels of Storage

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~~ DRAM	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

Migration of data “A” from Disk to Register

- Multitasking environments must be careful to use most recent value, no matter where it is stored in storage hierarchy



- Multiprocessor environment must provide **cache coherency** in HW such that all CPUs have the most recent value in their cache
- Distributed environment situation even more complex ➔ Need OS-level data coherency
  - Several copies of a datum can exist
  - Various solutions exist

Open-Source vs. Closed-Source OS

- Close-Source OS
  - Available as compiled binary code
  - Example include Microsoft Windows
  - Very hard to update or extend by users
    - Needs reverse engineering which is illegal in many countries
    - Reverse engineering is a lot of work
    - Comments and structure of SW would not be available by reverse engineering
  - Typically more prone to bugs
    - Limited number of code reviewers

Open-Source vs. Closed-Source OS (cont.)

- Open-Source OS
  - Available in source-code format
  - A user can modify, compile, and run updated code
  - Large community of interested programmers who contribute to code (debug/analyze/changes)
  - More secure than close-source code as many more eyes reviewing the code
  - Revenue achieved by support contracts and sale of HW
    - Red-Hat, SUSE, Sun

Open-Source vs. Closed-Source OS (cont.)

- Open-Source OS
  - Counter to **copy protection** and **Digital Rights Management (DRM)** movement
  - Started by **Free Software Foundation (FSF)**, which has "copyleft" **GNU Public License (GPL)**
    - By Richard Stallman, 1983
    - GPL requires any changes made to source code must be released under the same GPL license
  - Examples include **GNU/Linux** and **BSD UNIX** (including core of **Mac OS X**), and many more

Sample Open-Source OS

- Linux
  - Initiated by a student in Finland
  - Updates released once a week
  - Different distributions vary in function, utility, installed applications, HW support, user interface, and purpose
    - RedHat enterprise: large commercial use
    - PCLinuxOS: can be booted from a CD
    - Other distributions: Fedora, Debian, SUSE, Slackware, and Ubuntu

Sample Open-Source OS (cont.)

- BSD UNIX
  - Started in 1978 as a derivative of AT&T UNIX
  - Fully functional, open-source released in 1994
  - Different distributions
    - FreeBSD, NetBSD, OpenBSD, DragonflyBSD
- Solaris
  - Commercial UNIX-based OS
  - Partially open-source OS
    - Since some part of the code is still owned by AT&T

End of Lecture 1