Operating Systems

Prof. Yongkun Li
中科大-计算机学院 特任教授
http:staff.ustc.edu.cn/~ykli

Chapter 1
Overview of an Operating System

Objectives

- Overview of OS
  - Overview of Computer System: Organization & Architecture
  - What is an OS
  - OS Operation: Interrupt-driven via system call
- Major OS Components
  - Process Management
  - Memory Management
  - Storage Management
- Kernel Data Structures
- Misc: Computing Environments & Open-Sourced OS

What is an Operating System?

- According to your experience…
  - Networking;
  - Storage;
  - Multimedia;
  - Gaming;
  - What else?

None of the above were about the OS!

Before we talk about OS…

Overview of Computer System

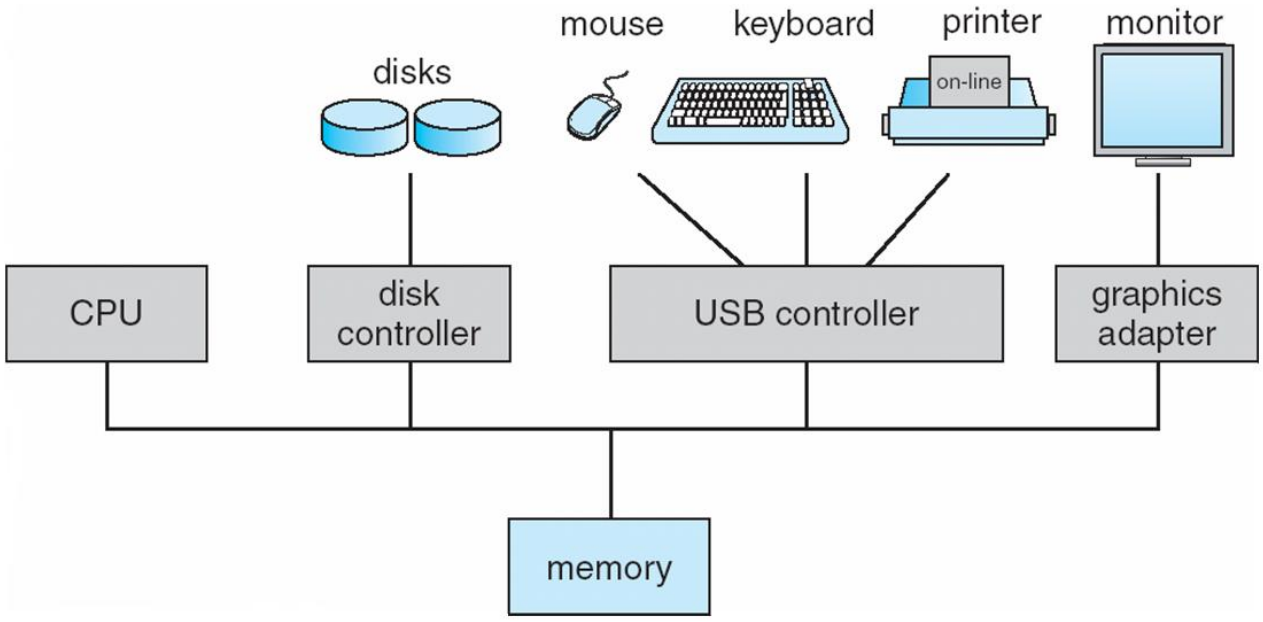
-System Organization

-Storage Structure

-System Architecture

Computer System Organization

- Computer-system organization
  - 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 Organization

- 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
  - Device controller informs CPU that it has finished its operation by causing an interrupt

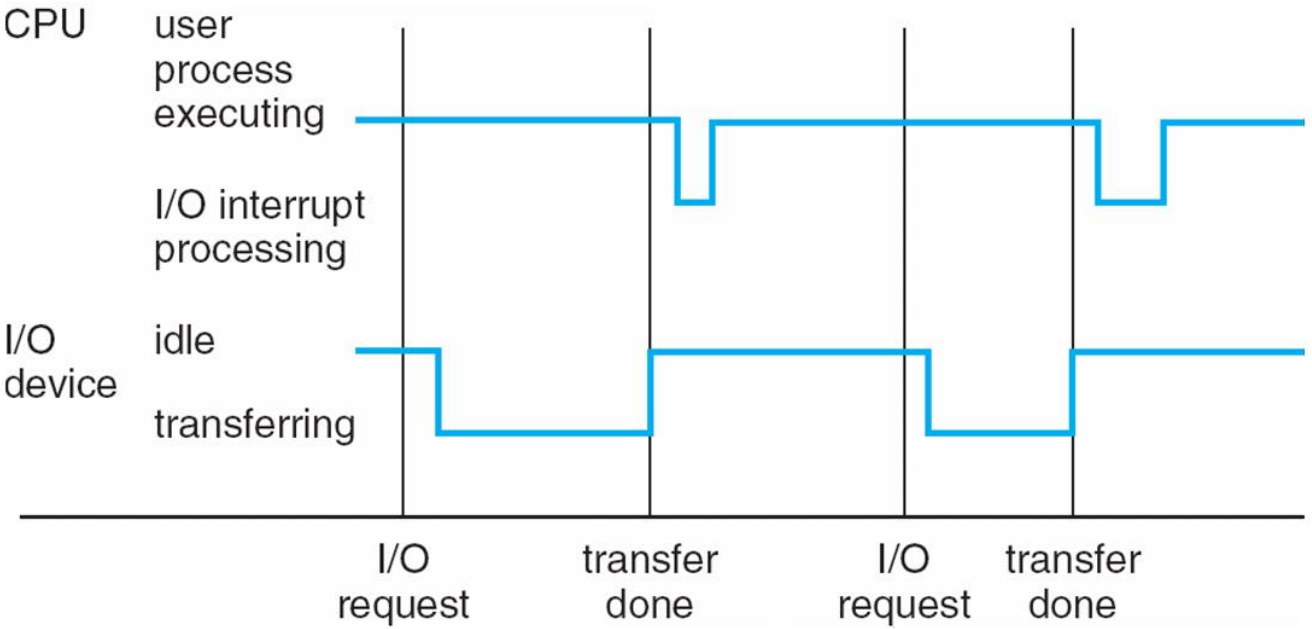
Computer Startup

- **bootstrap program** is loaded at power-up or reboot
  - Typically stored in ROM or EPROM, generally known as **firmware**
  - Initializes all aspects of system
  - Loads operating system kernel into memory and starts execution
- System processes or system daemons
  - Run the entire time the kernel is running
  - On UNIX, the first system process is “init”
- After fully booted, waits for events to occur
  - Signaled by **interrupt**

Interrupt Handling

- Interrupt can be triggered by hardware or software
  - Hardware sends signal to CPU
  - Software executes a special operation: **system call**
- Interrupt procedure
  - CPU stops what is doing
  - Execute the service routine for the interrupt
  - CPU resumes
- Operating system is **interrupt driven**

Interrupt Timeline



Common Functions of Interrupts

- Each computer design has its own interrupt mechanism
- Interrupt transfers control to the interrupt service routine
  - A table of pointers to interrupt routines, the **interrupt vector**, can be used to provide necessary speed
  - The table of pointers is stored in low memory
- Interrupt architecture must save the address of the interrupted instruction
  - Modern architectures store the return address on system stack

Overview of Computer System

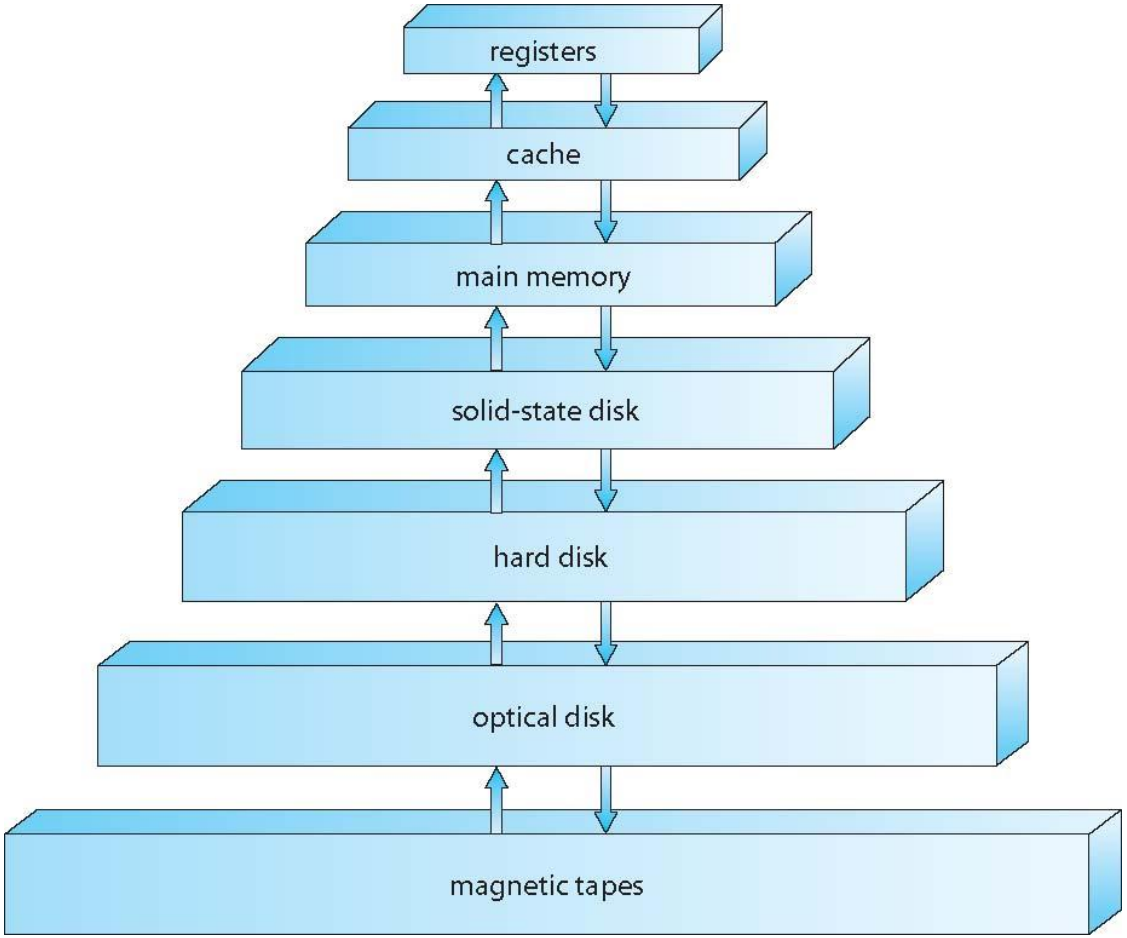
-System Organization

-Storage Structure

-System Architecture

Storage Structure

- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility



Storage Structure

- Main memory
  - CPU can load instructions only from memory (only large storage media that the CPU can access directly)
  - **Random access,** typically **small size** and **volatile**
  - All forms of memory provide an array of bytes
    - Each byte has its own address
    - Interaction: load & store (memory <-> register)
- Instruction-execution cycle
  - Fetch an instruction from memory and store in register
  - Decode instruction (fetch operands if necessary)
  - Store result back to memory

Storage Structure

- Secondary storage – extension of main memory that provides large **nonvolatile** storage capacity
  - Hard 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
  - **Solid-state disks** – faster than hard disks, nonvolatile
    - Various technologies
    - Becoming more popular

Caching

- **Caching** – copying information into faster storage system; main memory can be viewed as a cache for secondary storage
- 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
- Important principle, performed at many levels in a computer (in hardware, operating system, software)

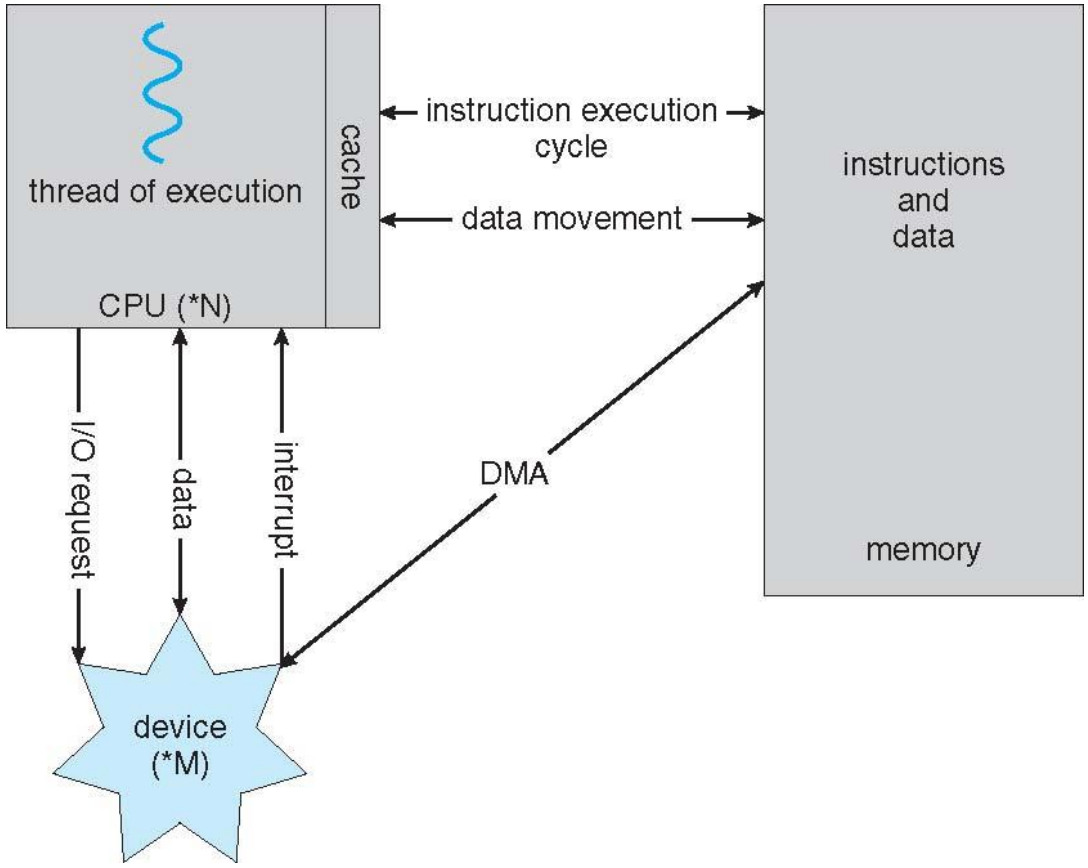
I/O Structure

- Storage is only one of many types of I/O devices
- Device controller
  - More than one device may be attached
  - Local buffer storage & a set of registers
- Device driver: for each device controller to manage I/O, provides uniform interface between controller and kernel
- Interrupt-driven I/O
  - Device driver loads registers within the controller
  - Controller examines the registers to decide what action to take
  - Device controller starts data transfer to its local buffer
  - Informs driver via an interrupt and returns control to OS

Direct Memory Access Structure

- Used for high-speed I/O devices able to transmit information at close to memory speeds
- Device controller transfers blocks of data from buffer storage directly to main memory without CPU intervention
- Only one interrupt is generated per block, rather than the one interrupt per byte

How a Modern Computer Works



Overview of Computer System

-System Organization

-Storage Structure

-System Architecture

Computer-System Architecture

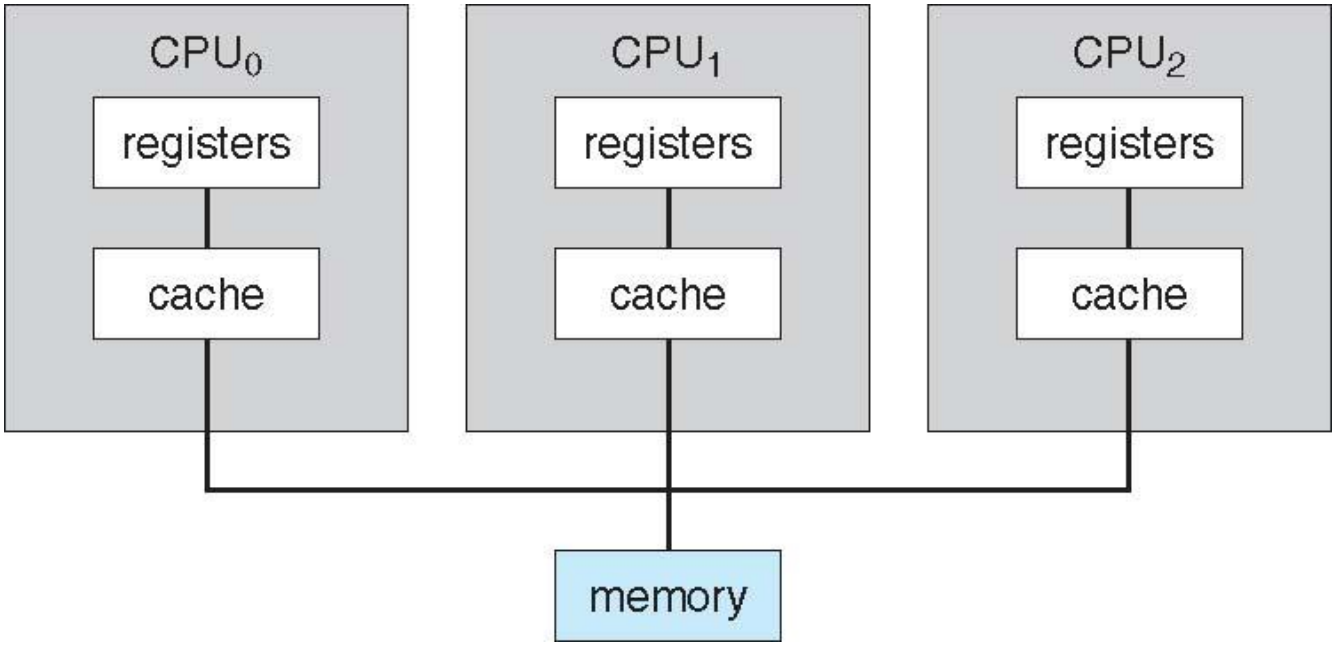
- Most systems use a single general-purpose processor
  - One main CPU capable of executing general-purpose instruction set
- May have special-purpose processors as well
  - Device-specific processors: disk, keyboard, etc…
  - Run a limited instruction set
  - Do not run user processes
  - Managed by OS or built into the hardware

Computer-System Architecture

- **Multiprocessors** systems grow in use and importance
  - Also known as **parallel systems, multicore systems**
- Advantages include:
  - **Increased throughput**
  - **Economy of scale:** share peripherals, mass storage and power supply
  - **Increased reliability** – graceful degradation or fault tolerance
- Two types
  - **Asymmetric Multiprocessing** – each processor is assigned a specie task: boss-worker relationship
  - **Symmetric Multiprocessing (SMP)** – each processor performs all tasks: all processors are peers

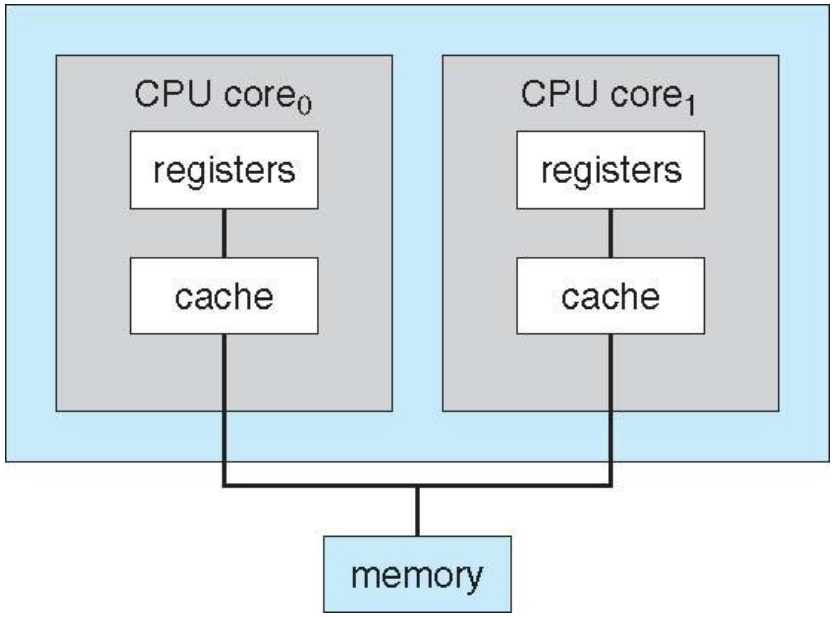
Symmetric Multiprocessing Architecture

- Symmetric Multiprocessing (SMP)
  - Result from hardware or software
  - Adds CPUs to increase computing power
  - Causes non-uniform memory access (NUMA)



Multicore

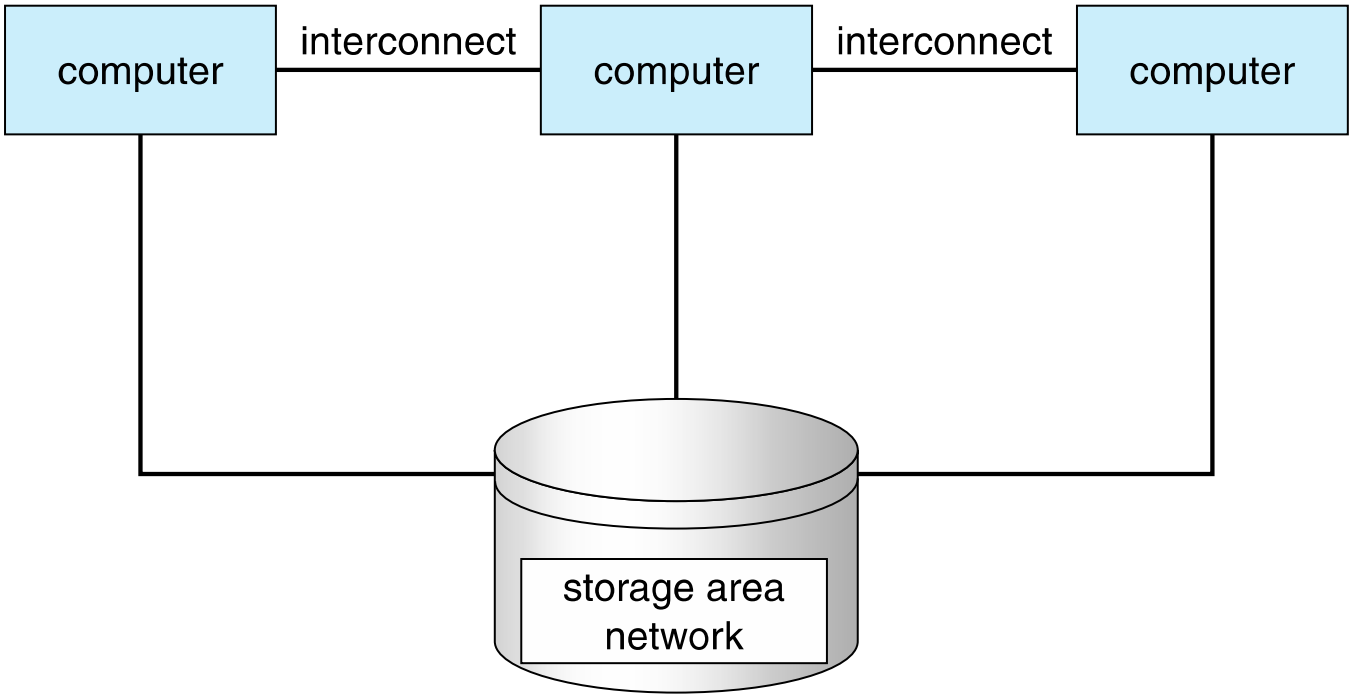
- **Multicore:** include multiple cores on a single chip
- More efficient
  - On-chip communication is faster than between-chip communication
  - Less power
- Dual-core design



Clustered Systems

- Like multiprocessor systems, but multiple systems working together
  - Usually sharing storage via a **storage-area network (SAN)**
  - Provides a **high-availability** service which survives failures
    - **Asymmetric clustering** has one machine in hot-standby mode
    - **Symmetric clustering** has multiple nodes running applications, monitoring each other
  - Some clusters are for **high-performance computing (HPC)**
    - Applications must be written to use **parallelization**
  - Some have **distributed lock manager** (**DLM**) to avoid conflicting operations

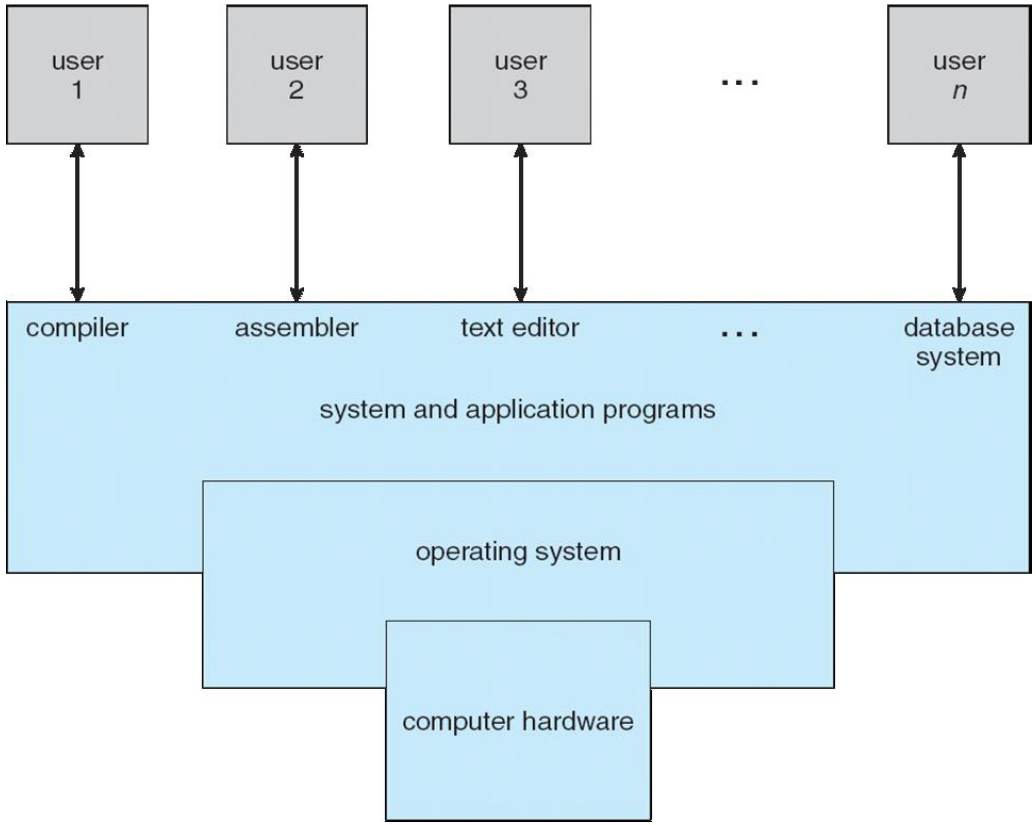
Clustered Systems



What is an Operating System?

Where is the OS?

- Let's start understanding an OS from this question:
  Where is it?

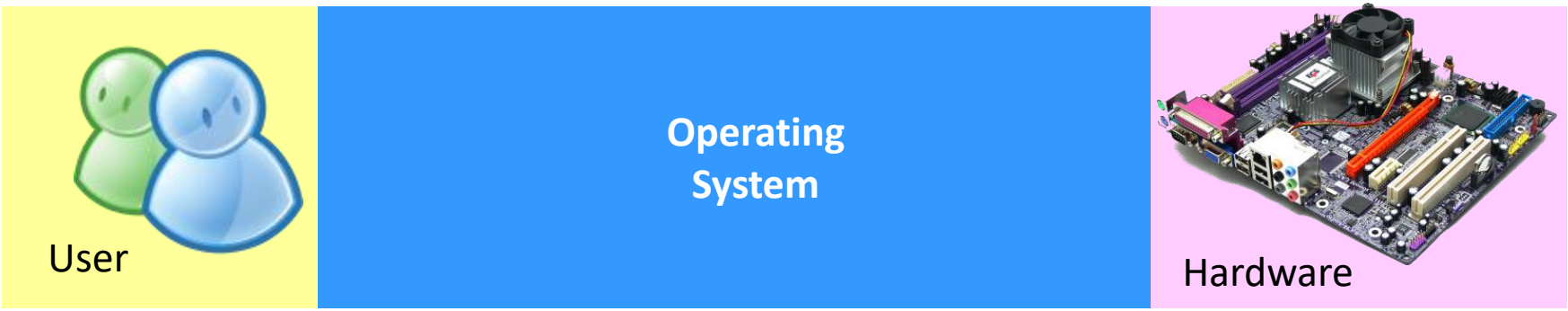


Where is the OS?

- Four components of a computer system
  - Users: People, machines, other computers
  - Hardware – provides basic computing resources (CPU, memory, I/O devices)
  - App. programs – define the ways in which the sys. resources are used to solve the computing problems
    - Word processors, compilers, web browsers, database systems, video games, etc.
  - Operating system
    - Controls and coordinates use of hardware among various applications and users

What is an Operating System?

- It stands **between** the hardware and the user.
  - A program that acts as an intermediary between a user of a computer and the computer hardware

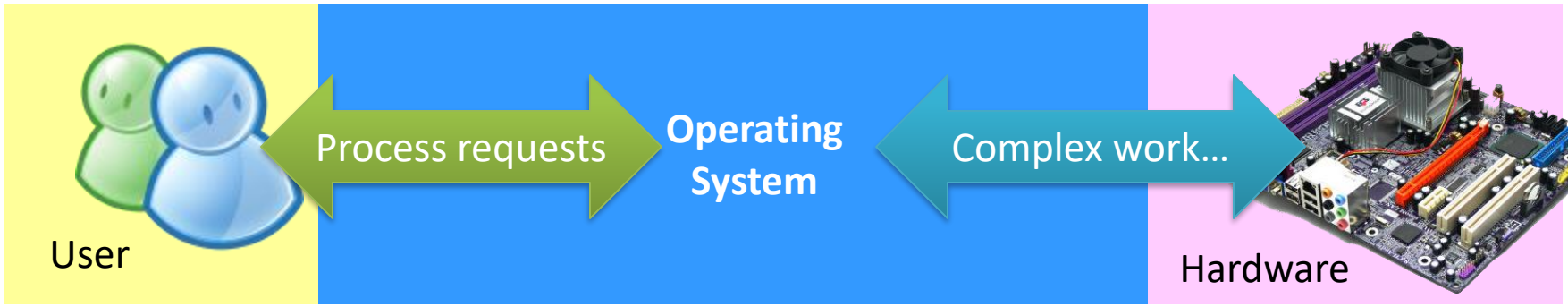


- Operating system goals:
  - Execute user programs & make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner
  - Design tradeoff between convenient and efficiency

What is an Operating System?

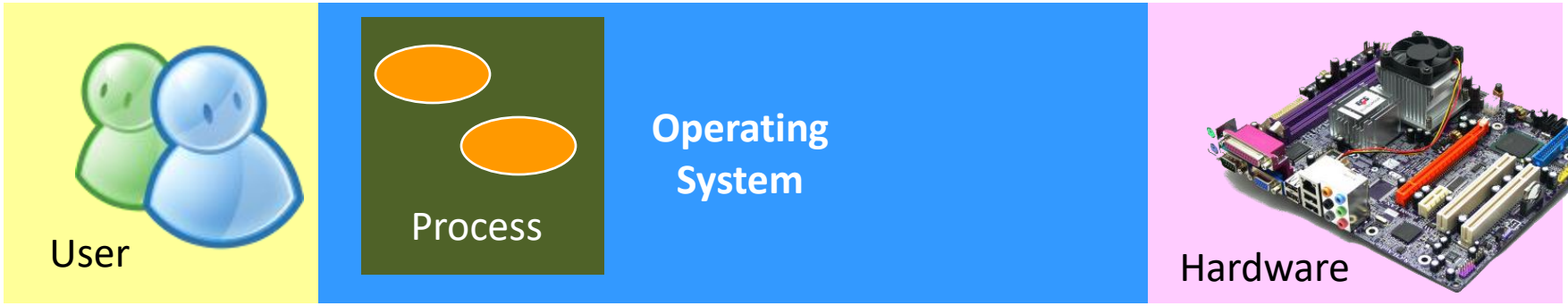
- How good is this design?
  - The user does not have to program the hardware directly.
    - It hides *all the troublesome operations* of the hardware.

Example. The OS, on one hand, hides the physical system memory away from you. On the other hand, it tells you that there is system memory available when you run your applications.



What is an Operating System?

- **Processes** as the starting point!
  - Whatever programs you run, you create processes.
    - i.e., you need processes to open files, utilize system memory, listen to music, etc.
  - So, process lifecycle, process management, and other related issues are essential topics of this course.

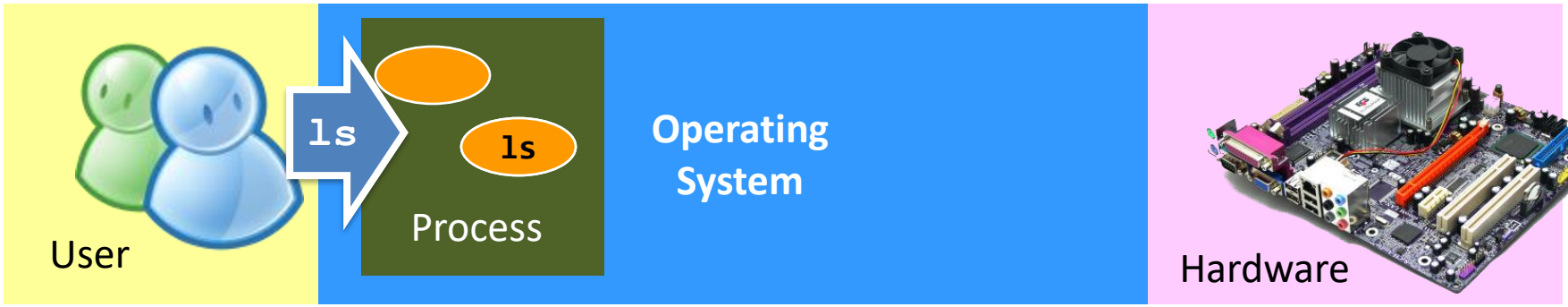


What is an Operating System?

- Example (step 1)

```
$ ls
```

Most commands you type in the **shell** are the same as starting a new process.



What is an Operating System?

- Example (step 2)

```
$ ls
```

The operating system contains the codes that are needed to work with the file system.

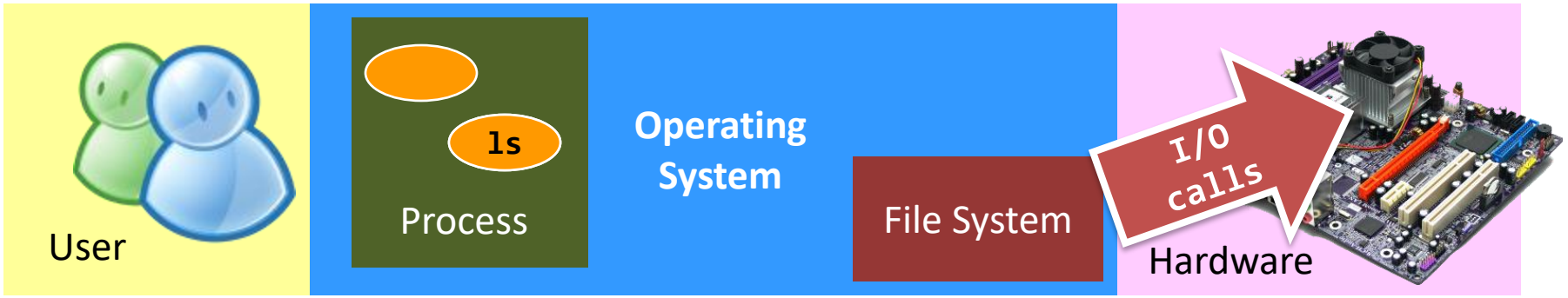
The codes are called the **kernel**.

What is an Operating System?

- Example (step 3)

```
$ ls
```

The file system module inside the operating system knows how to work with devices, using device drivers.

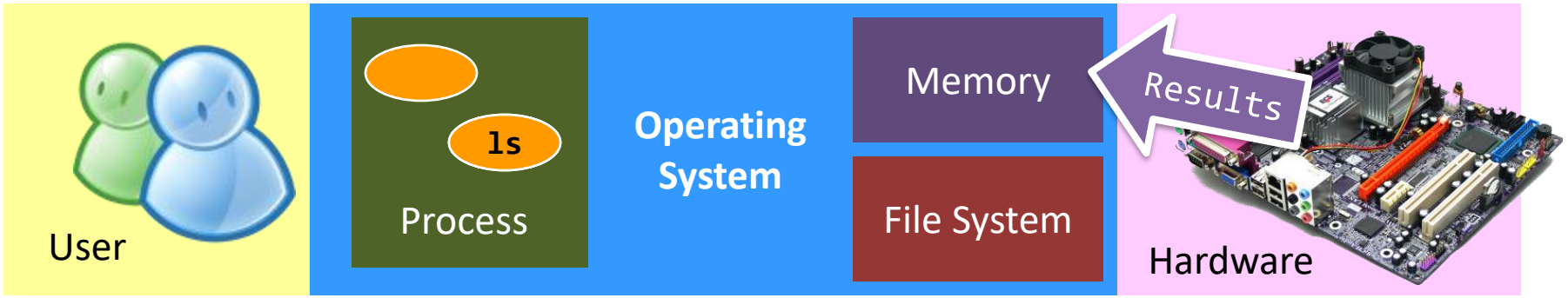


What is an Operating System?

- Example (step 4)

```
$ ls
```

Of course, the operating system will allocate memory for the results.



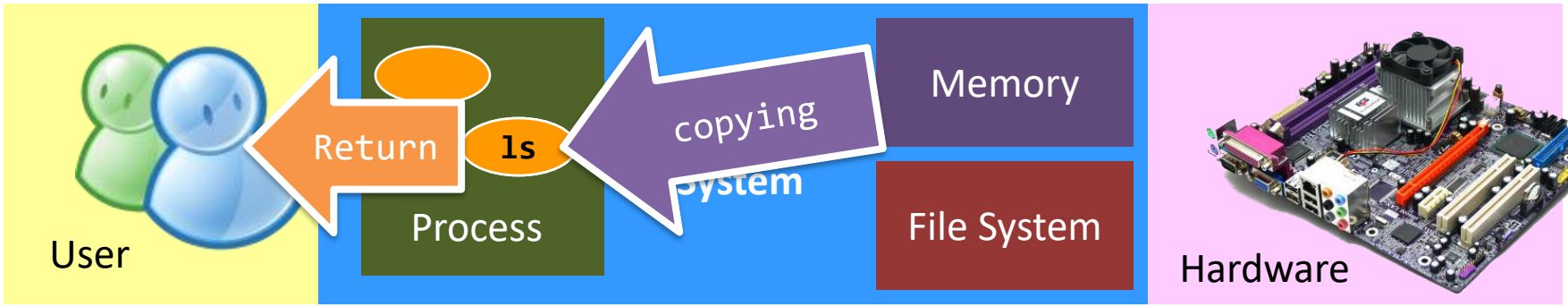
What is an Operating System?

- Example (final step)

```
$ ls
.  ..  index.html
$ _
```

The memory management sub-system will copy the result to the memory of the process.

At last, the result returns.



What Operating Systems Do

- System View
  - OS is a **control program**
    - Controls execution of programs to prevent errors and improper use of the computer
  - OS is a **resource allocator**
    - Manages all resources
    - Decides between conflicting requests for efficient and fair resource use

What Operating Systems Do

- Depends on the point of view
- User View
  - PC users want convenience, **ease of use** and **good performance**, don't care about resource utilization
  - But shared computer such as mainframe or minicomputer must keep all users happy: maximize **resource utilization**
  - Users of dedicate systems such as workstations have dedicated resources but frequently use shared resources from servers: **tradeoff**
  - Mobile computers are resource poor, optimized for **usability and battery life**
  - Some computers have little or no user interface, such as embedded computers in devices and automobiles

Operating System Definition

- No universally accepted definition of what is part of the operating system
  - Operating systems grew increasingly sophisticated
  - Microsoft case
- Current Mobile OS
  - Once again the number of features constituting the OS is increasing
  - Core kernel + Middleware
    - Databases, multimedia, graphics, etc…

Operating System Definition

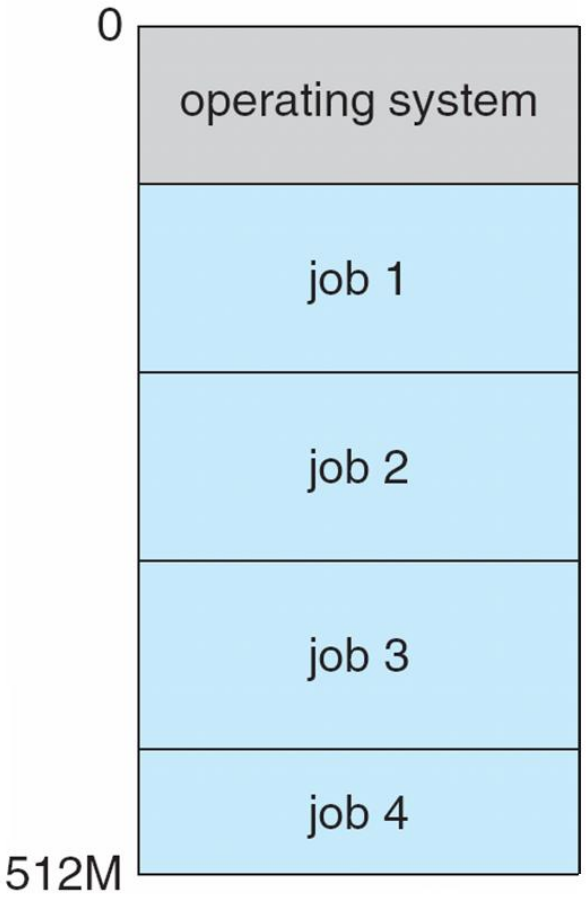
- No universally accepted definition
- Simple viewpoint
  - "Everything a vendor ships when you order an operating system" is a good approximation
  - But varies wildly
- Common definition
  - "The one program running at all times on the computer" is the **kernel**.
- Everything else is either
  - a system program (ships with the operating system) , or
  - an application program.

Operating System Operations

Multiprogramming

- Operating system provides the environments within which programs are executed
  - Single user cannot keep CPU and I/O devices busy at all times
- **Multiprogramming** needed for efficiency: most important aspect of OS
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - All jobs are initially kept on disk in the job pool, 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

Memory Layout for Multi-programmed System



Multitasking

- **Time sharing** (**multitasking**) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating **interactive** computing
  - **Response time** should be < 1 second
- Allow many users to share the computer
  - Each user has at least one program executing in memory ⇨**process**
- Issues
  - If several jobs ready to run at the same time ⇨ **CPU scheduling**
  - If processes don't fit in memory, **swapping** moves them in and out to run
  - **Virtual memory** allows execution of processes not completely in memory

Interrupt Driven Mechanism

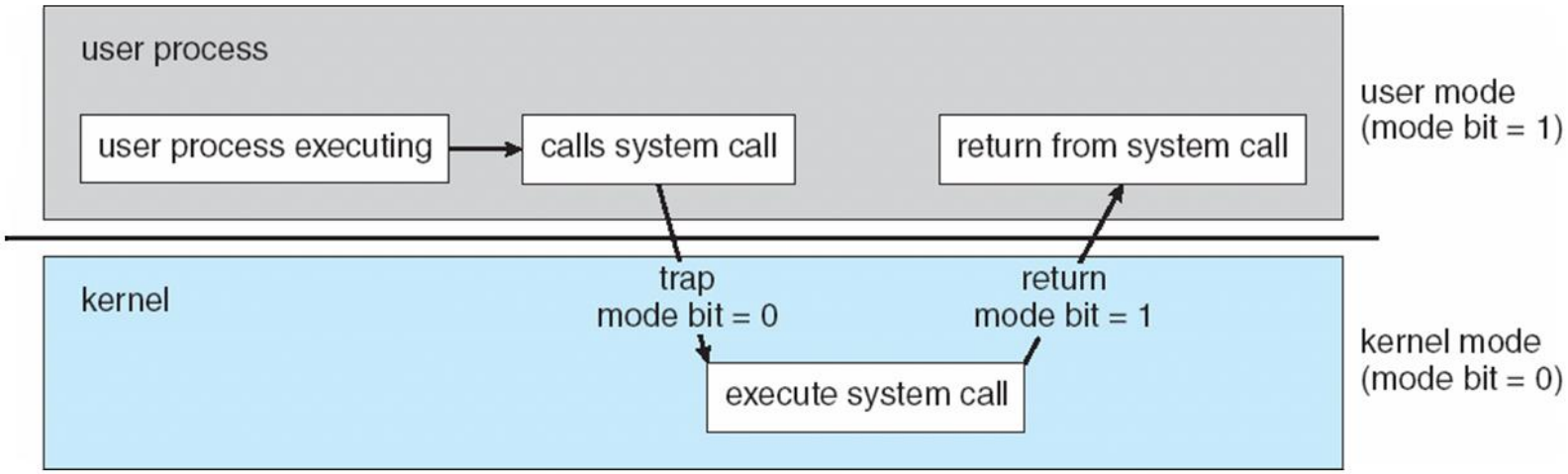
- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap):**
    - Software error (e.g., division by zero)
    - Request for operating system service
    - Other process problems include infinite loop, processes modifying each other or the operating system
  - An interrupt service routine is provided to deal with the interrupt

Dual-mode Operation

- **Dual-mode** operation allows OS to protect itself and other system components
  - **User mode** and **kernel mode**
  - **Mode bit** provided by hardware
    - Provides ability to distinguish when system is running user code or kernel code
    - Some instructions designated as **privileged**, only executable in kernel mode
    - System call changes mode to kernel, return from call resets it to user

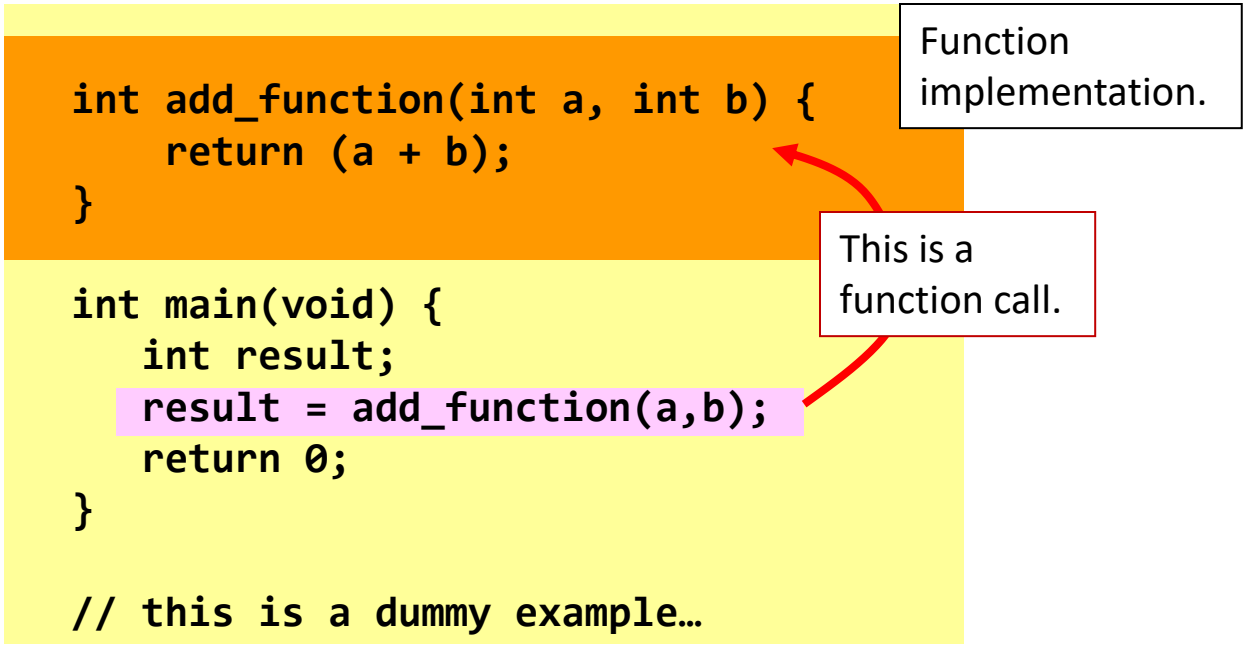
Transition from User to Kernel Mode

- At system boot time, the hardware starts in kernel mode
- OS is loaded and starts user application in user mode
- Interrupt occurs, the hardware switches from user mode to kernel mode
- Whenever the OS gains control, it is in kernel mode



System Calls

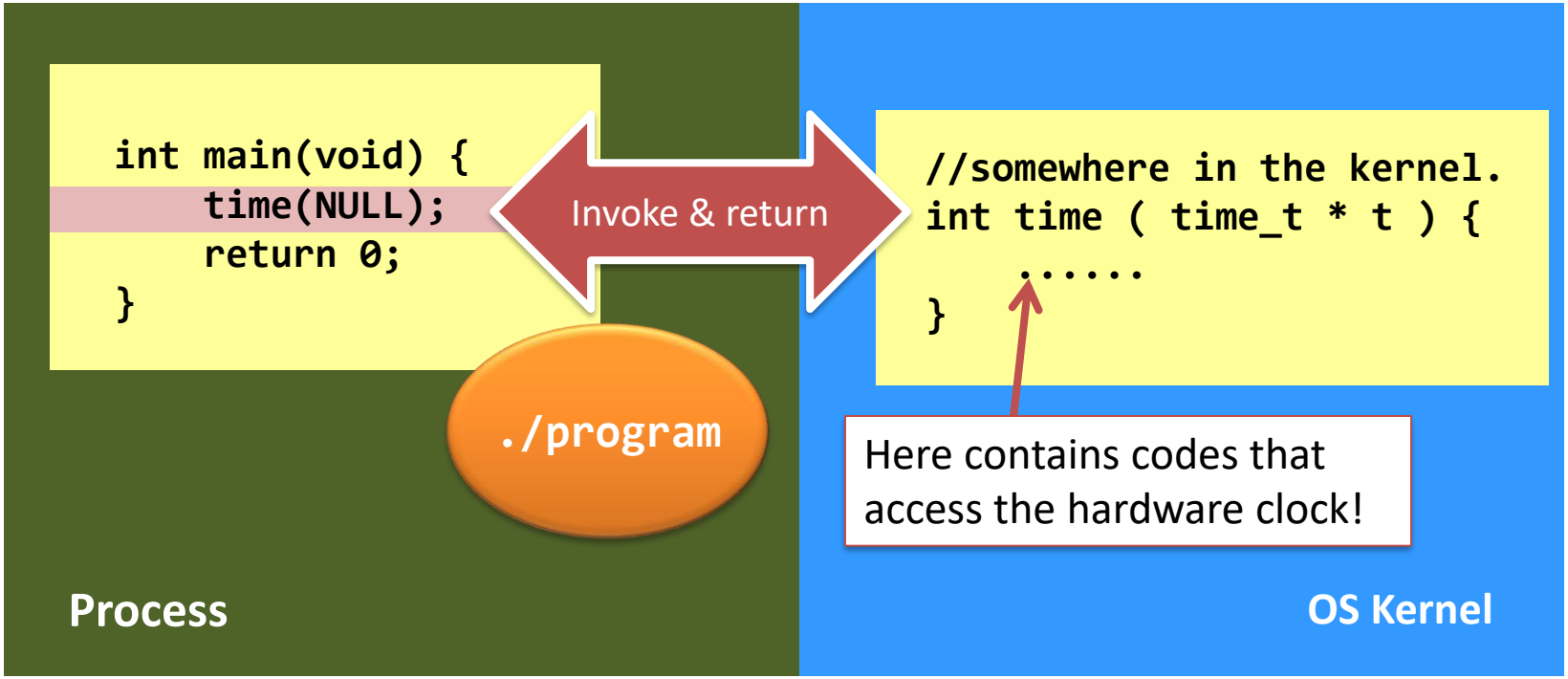
- Informally, a system call is similar to a function call, but…
  - The function implementation is inside the OS.
  - We name it the **OS kernel**.



System Calls

- System calls are the programming interface between processes and the OS kernel
  - System calls provide the means for a user program to ask the operating system to perform tasks
- A system call usually takes the form of a trap to a specific location in the interrupt vector, treated by the hardware as a software interrupt
- The system call service routine is a part of the OS

Interacting with the OS



System calls

- The system calls are usually
  - **primitive**,
  - **important**, and
  - **fundamental**.
  - e.g., the **`time()`** system call.

- Roughly speaking, we can categorize system calls as follows:

Process	File System	Memory
Security	Device	

System calls VS Library function calls

- If a call is not system calls, then they are **library calls** (or function calls)!

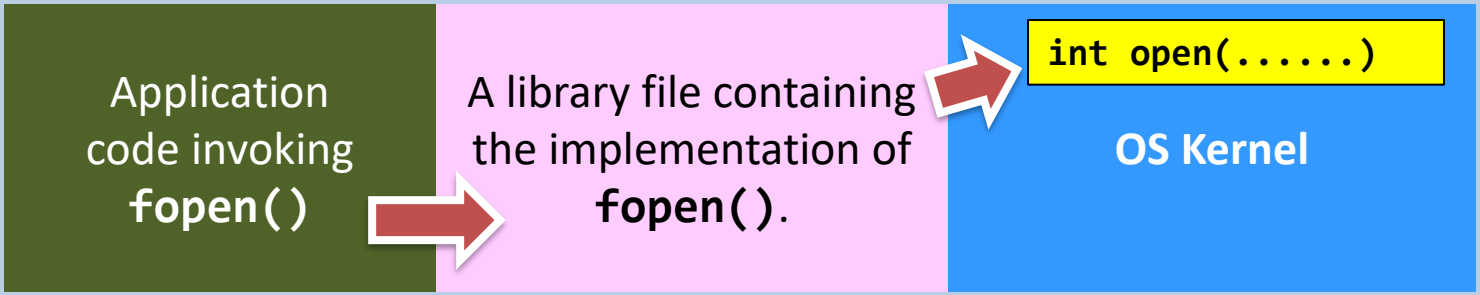
- Take **`fopen()`** as an example.
  - **`fopen()`** invokes the system call **`open()`**.
  - So, why people invented **`fopen()`**?
  - Because **`open()`** is too primitive and is not programmer-friendly!

Library call	`fopen("hello.txt", "w");`
System call	`open("hello.txt", O_WRONLY | O_CREAT | O_TRUNC, 0666);`

System calls VS Library function calls

- Library functions are usually compiled and packed inside an object called the **library file**.
  - In windows: DLL – dynamically linked library.
  - In Linux: SO – shared objects.

Big picture



OS Standards

- Who defines the system calls? Functionalities? Arguments? Return values?

- There are standards!

Standards	Full Name	Example OS
POSIX	Portable Operating System Interface	Linux
BSD	Berkeley Software Distribution	Mac OS Darwin
SVR4	System V (five) Release 4	Solaris Unix

Introduction to Operating System Components

Process

Process OR Program?

- A process is not a program!

Let's consider the following two commands

Command A	`ls -R /`	Recursively print the directory entries, starting from the directory '/'
Command B	`ls -R /home`	Recursively print the directory entries, starting from the directory '/home'

Similarity	Difference
Both use the program file "**`/bin/ls`**".	The program arguments are different.
---	The processes' internal status are different, such as running time.

Program != Process

- A process is an execution instance of a program.
  - More than one process can execute the same program code
  - Later, you'll find that a process is not bounded to execute just one program!

- A process is active.
  - A process has its local states concerning the execution. E.g.,
    - which line of codes it is running;
    - which CPU core (if there are many) it is running on.
  - The local states change over time.

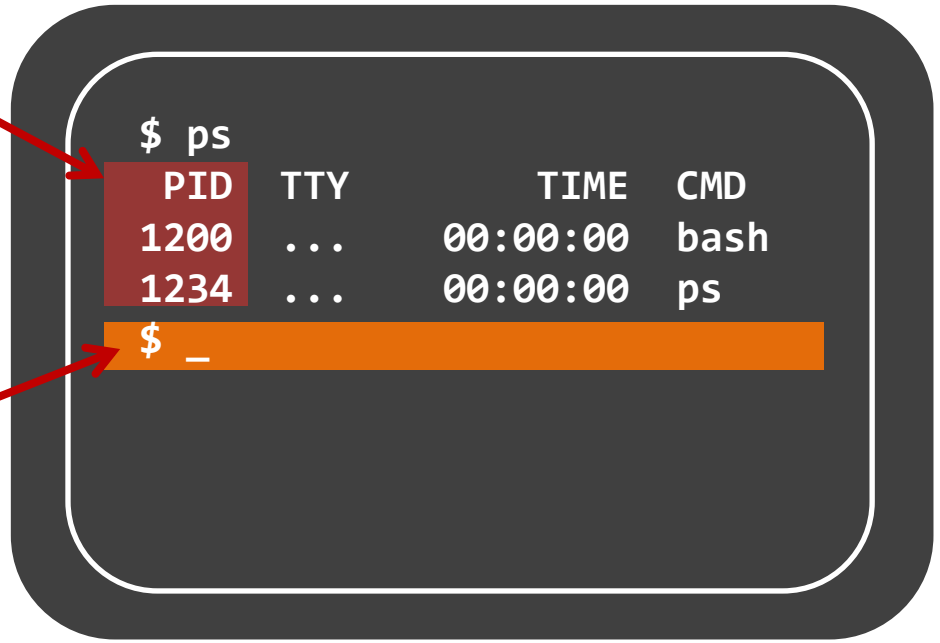
- Commands about processes (and hopefully you've tried them before) – e.g., **`ps`** & **`top`**.

Process-Related Tools

- The tool "**ps**" can report a vast amount of information about every process in the system
  - Try "**ps -ef**".

This column shows the unique identification number of a process, called **Process ID**, or PID for short.

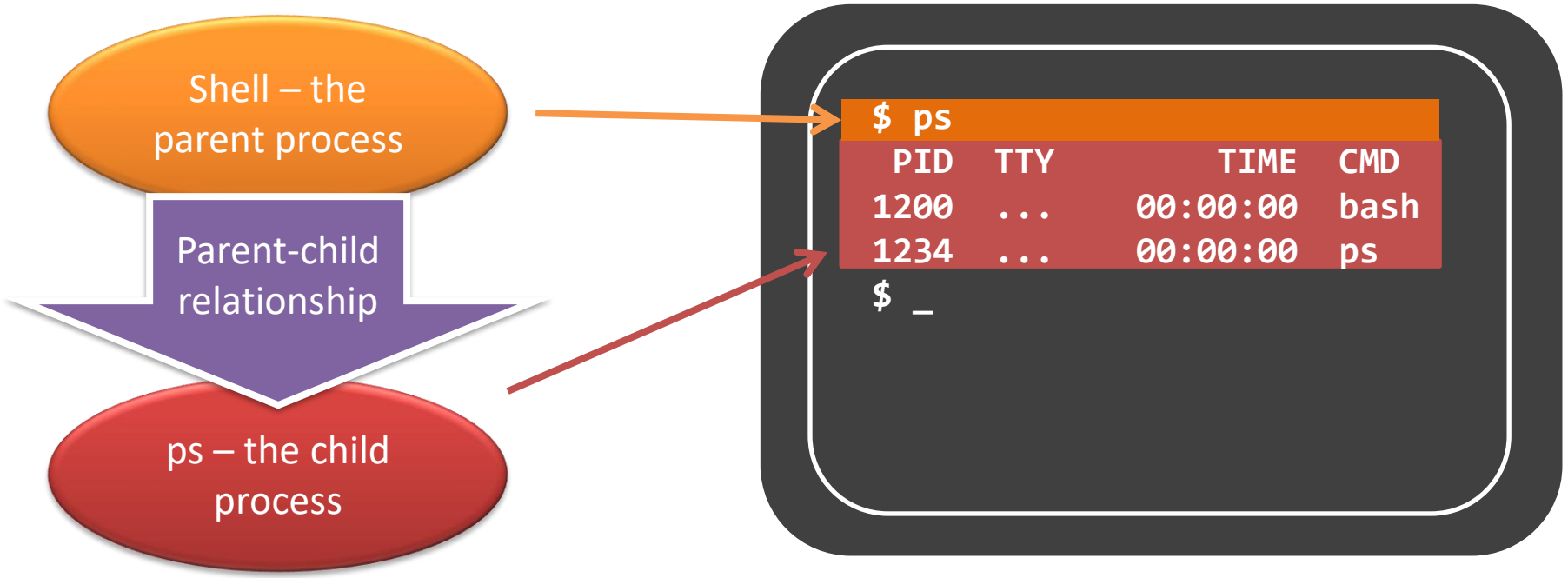
Hint: you can treat **ps** as the short-form of "**process status**"



By the way, this is called **shell**.

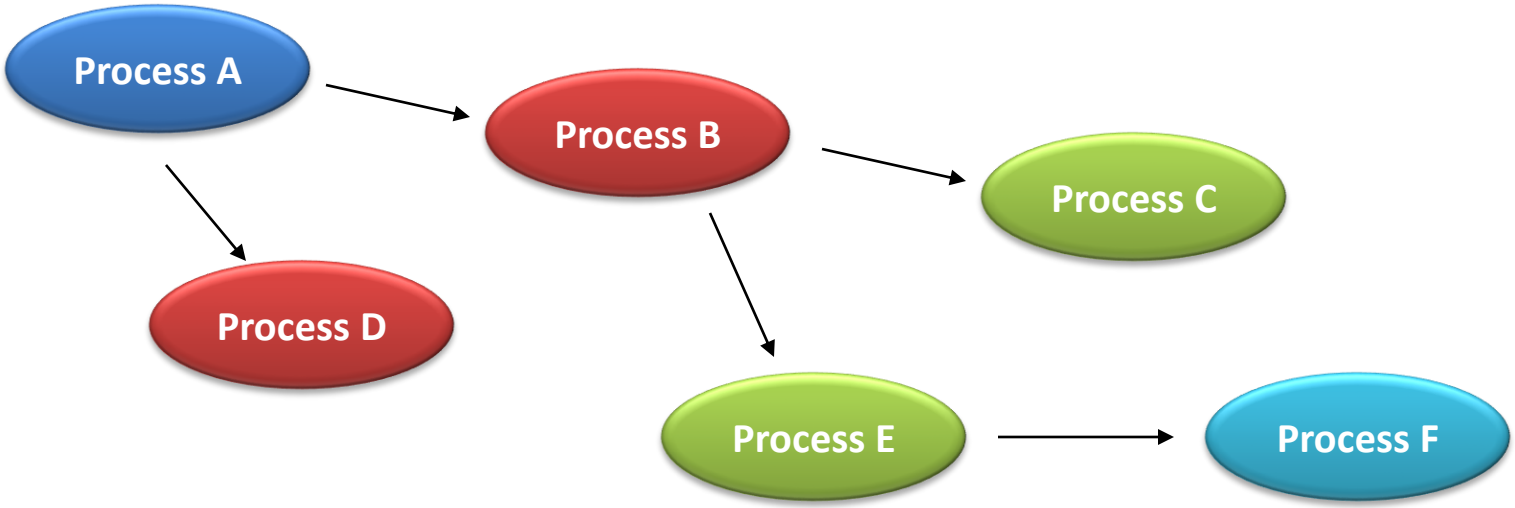
Shell – a process launching pad

- So, what is going on inside that shell?
  - The shell creates a new process, and is called a **child process** of the shell.
- The child process then executes the command "**ps**".



Process Hierarchy

- Process relationship:
  - A parent process will have its child process.
  - Also, a child process will have its child processes.
  - This form a **tree hierarchy**.



E.g., “Process E” is the shell and “Process F” is “**ps**”.

Process Summary

- A process is an execution instance of a program. It is a unit of work within the system.
  - Program is a ***passive entity***, process is an ***active entity***.
- Process needs resources to accomplish its task, process termination requires reclaim of any reusable resources
  - CPU, memory, I/O, files, Initialization data
- Single-threaded process has one **program counter** specifying location of next instruction to execute, multi-threaded process has one program counter per thread
  - Process executes instructions sequentially, one at a time, until completion
- Typically, system has many processes, some user, some operating system running concurrently

Process Management Activities

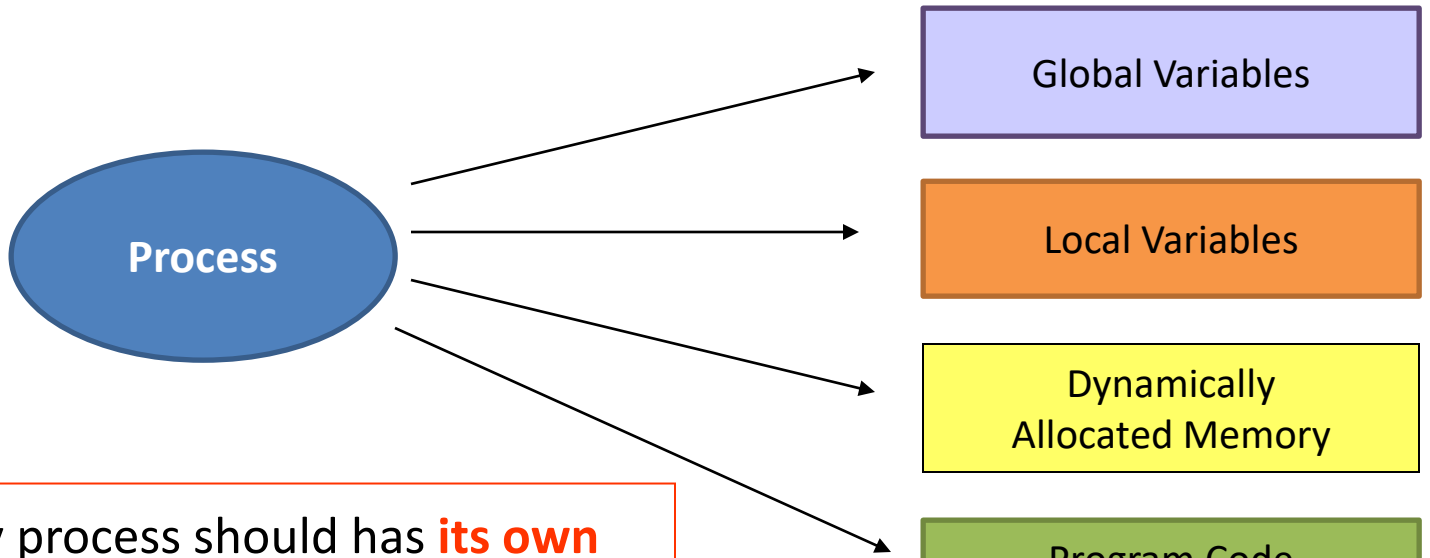
- The operating system is responsible for the following activities in connection with process management:
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling

Introduction to Operating System Components

Memory

Process' Memory

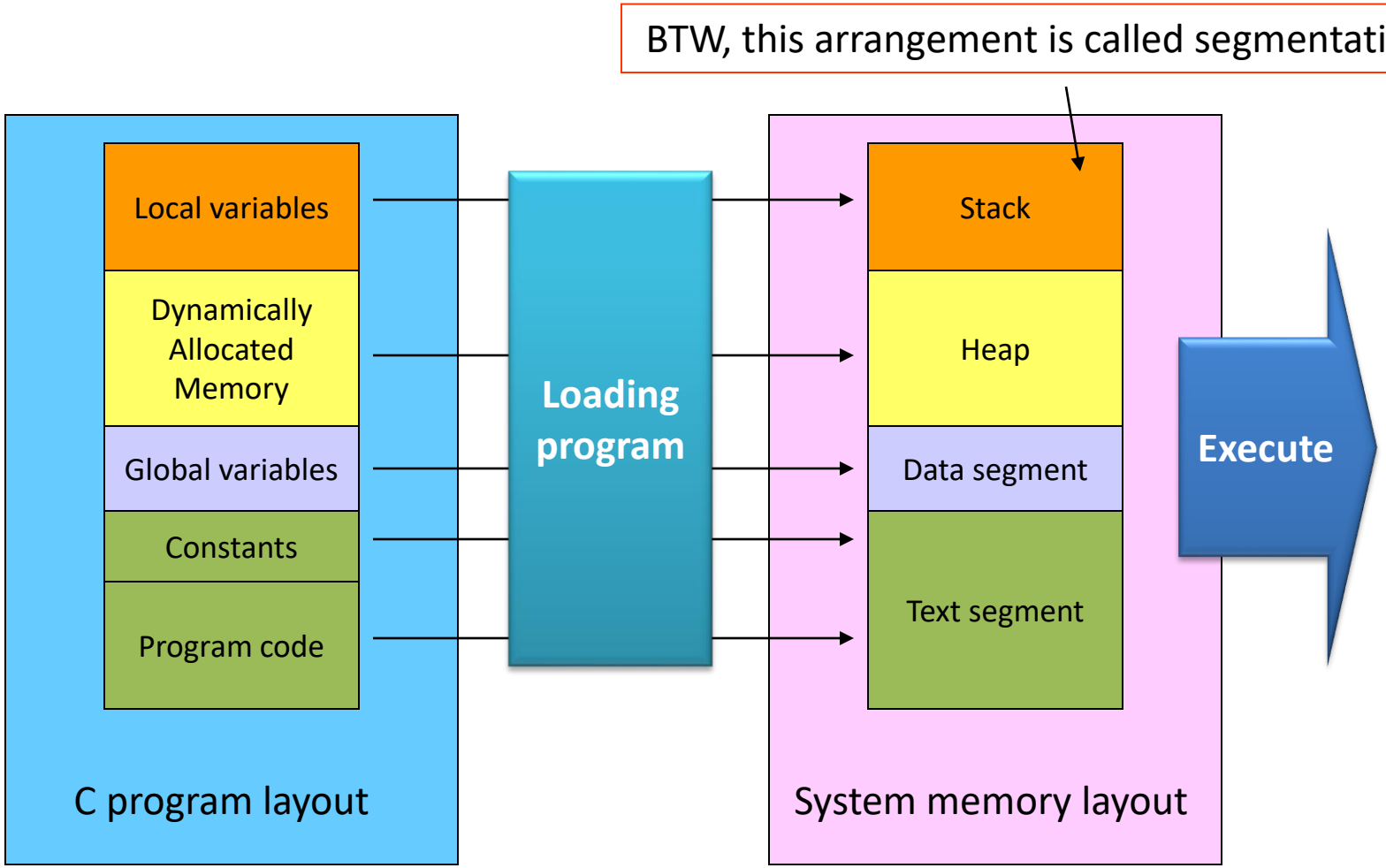
- What are the things that a process has to store?



Every process should has **its own set** of global variables, local variables, and allocated memory.

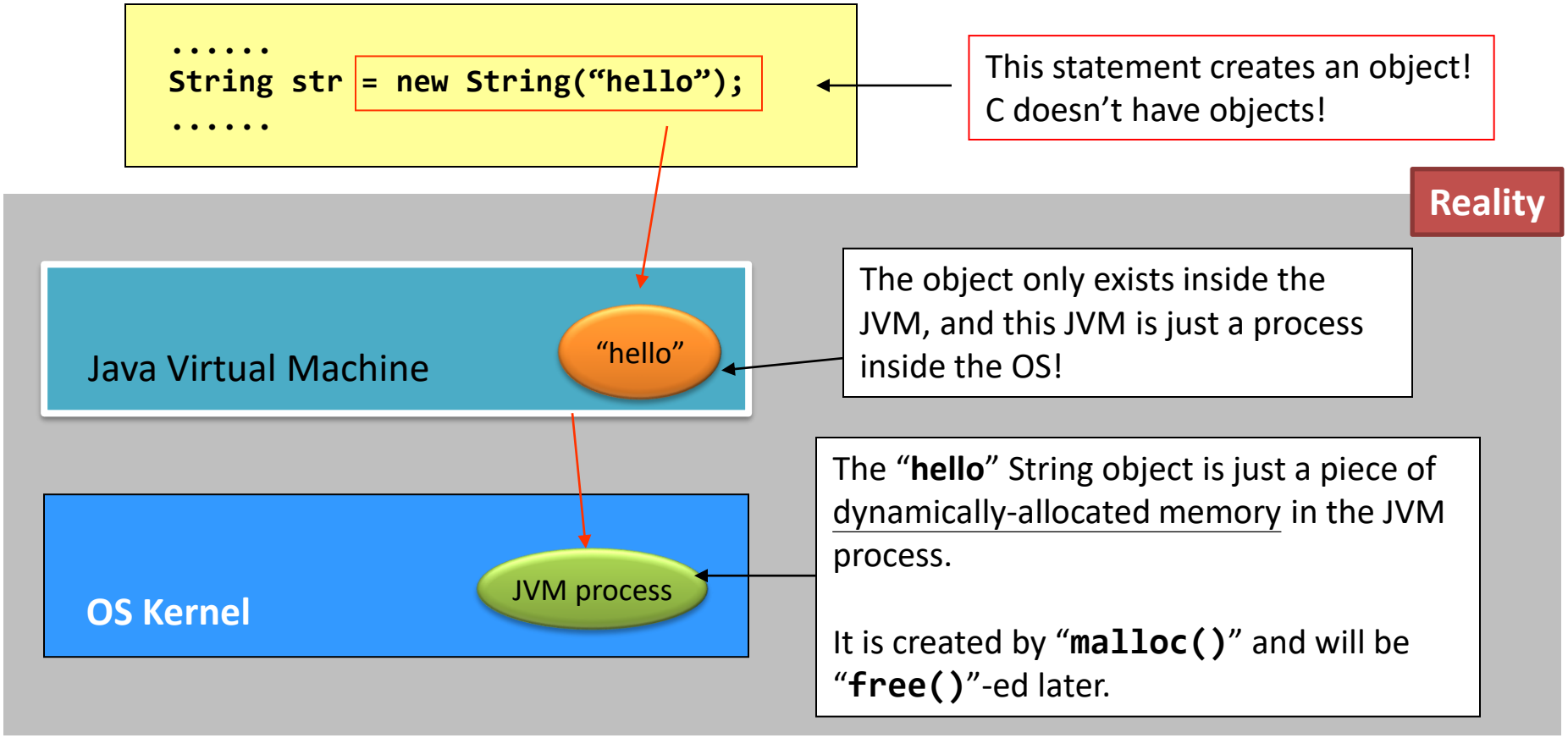
Process' Memory

- OMG…C is too low-level…



Process' Memory

- "*Hey, you're wrong! Some languages, e.g., Java, do not have the above layout…*", you asked.



Sidetrack: Pros and Cons in using C

- Cons:
  - Some people argued that C is a **bad beginner's programming language**. Now, you can understand why…

Because C requires a programmer to take care of the process-level memory management. Every programmer needs to know about the low-level memory layout in order for him/her to understand what segmentation fault means!
Every aspect on memory management can be manipulated using C. Learning **`malloc()`** exposes you to the heap manipulation. This makes a high-level programming language becoming low-level. Plus, **this exposes you to unpredictable dangers!**

* Disclaimer: choosing which programming language is really a personal choice.

Sidetrack: Pros and Cons in using C

- Pros:
  - Some people argued that C is an **efficient programming language**. Now, you can understand why…

Because C allows a programmer to **manipulate the process-level memory management "directly"**.
That's why many user libraries are implemented using C because of efficiency consideration. E.g., the Java Virtual Machine is implemented using C!
Most importantly, **C is the only language to interact with the OS directly**! In other words, the system call interface is written in C.

* Disclaimer: choosing which programming language is really a personal choice.

Memory Hierarchy

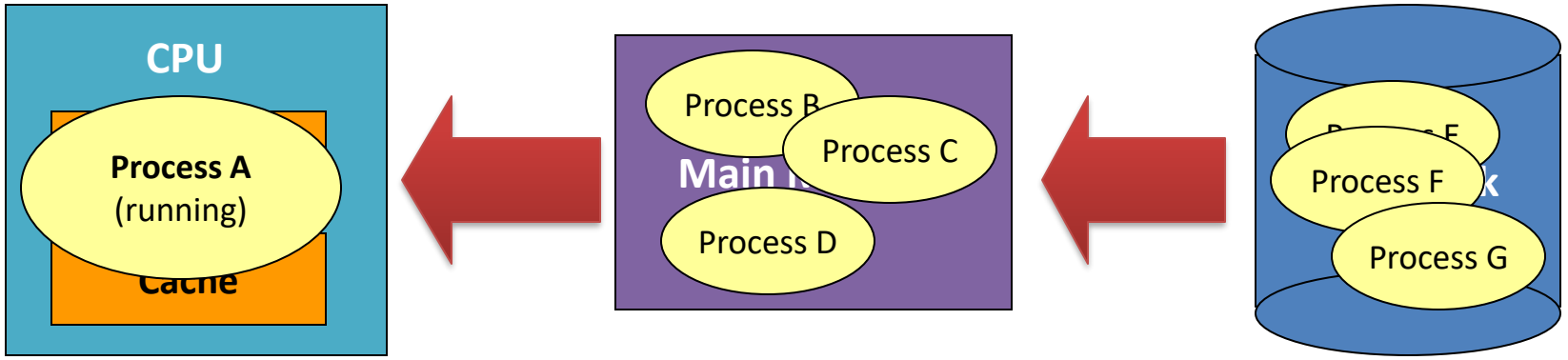
- In case that someone doesn't know about the hierarchy below…
  - A program is fetched from hard disk to main memory.
  - When executed, instructions in the program are fetched from the main memory to CPU.

Memory Hierarchy

- However, did you ever need to program those three things when you want to run the program “ls”?
  - Never! Then, who have the jobs done?
  - Of course, **OS**!

Memory Hierarchy

- Typically, there are more than 100 processes running "*at the same time*".
  - There is only a finite number of CPU cores, depending on how much money you spent.
  - Then, only a finite number of processes can be executed "*really at the same time*".
  - So, other (non-running) processes are stored at different devices controlled by the OS before they get a chance to run.



Memory Management Summary

- To execute a program
  - All (or part) of the instructions must be in memory
  - All (or part) of the data that is needed by the program must be in memory.
- Memory management determines what is in memory
  - Optimizing CPU utilization and computer response to users
- Memory 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

Introduction to Operating System Components

Storage Management

What is a File System?

- A file system, FS, means the way that a storage device is used.

- Have you heard of…
  - FAT16, FAT32, NTFS, Ext3, Ext4, BtrFS?
  - They are all file systems.
  - It is about how a storage device is utilized.

What is a File System?

- A file system must record the following things:
  - directories;
  - files;
  - allocated space;
  - free space.

- Think about the consequences if any one of the above is missing…

Two faces of a file system

- The **storage design** of the file system.
  - A file spends most of its time on the disk.
  - So, a file system is about how they are stored.
  - Apart from files, many others things are stored in the disk.

- The **operations** of the file system.
  - A file can be manipulated by processes.
  - So, a file system is also about the operations which manipulate the content stored.

FS VS OS

- **A FS is independent of an OS!**
  - If an OS supports a FS, then the OS can do whatever operations over that storage device.
  - Else, the OS doesn't know how to read or update the device's content.

Windows XP supports	Linux supports
NTFS, FAT32, FAT16, ISO9660, Juliet, CIFS	NTFS, FAT32, FAT16, ISO9660, Juliet, CIFS, Ext2, Ext3, etc…

Linux supports far more FS-es than any versions of Windows

File Operations?

- Pop quiz!
  - Guess, what are the fundamental file (not dir) operations?

Open	Read	Write	Close	Rename	Delete

- Well…creating is not...
  - It is just a special case of opening a file.
- Sorry…copying is not…
  - Do you know how it is implemented through the above operations?
- Sorry…moving is the same as renaming…
  - Except that a file is moving from one disk to another.

Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Various devices (i.e., disk drive, tape drive)
    - Varying properties include access speed, capacity, data-transfer rate, access method (sequential or random)
- File-System management
  - Files usually organized into directories
  - Access control to determine who can access what
  - OS activities include
    - Creating and deleting files and directories
    - Primitives to manipulate files and directories
    - Mapping files onto secondary storage
    - Backup files onto stable (non-volatile) storage media

Mass-Storage Management

- Usually disks 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
- 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

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	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

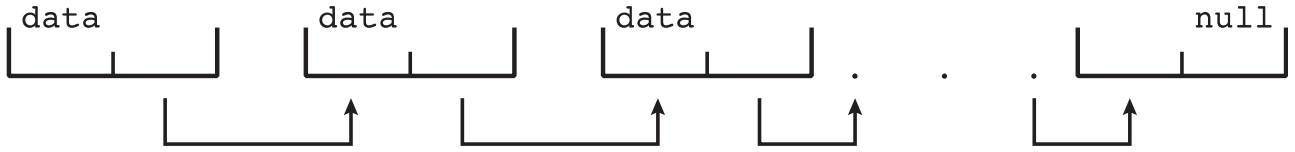
Kernel Data Structures

Kernel Data Structures

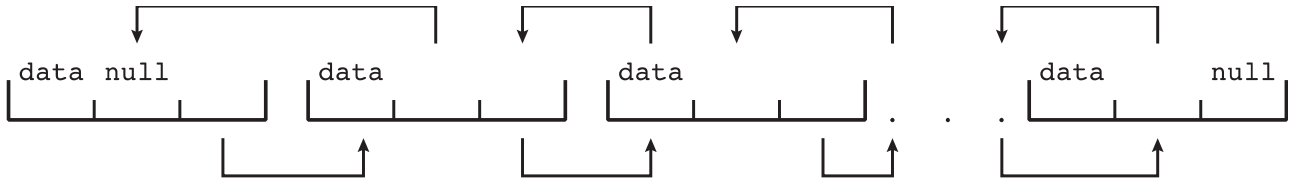
Lists, Trees, Hash Map and Bitmaps

Kernel Data Structures

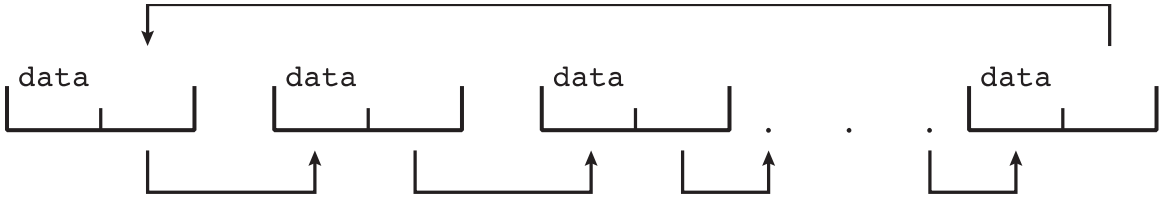
- Many similar to standard programming data structures
- **Lists**
  - Singly linked list



  - Doubly linked list



  - Circularly linked list

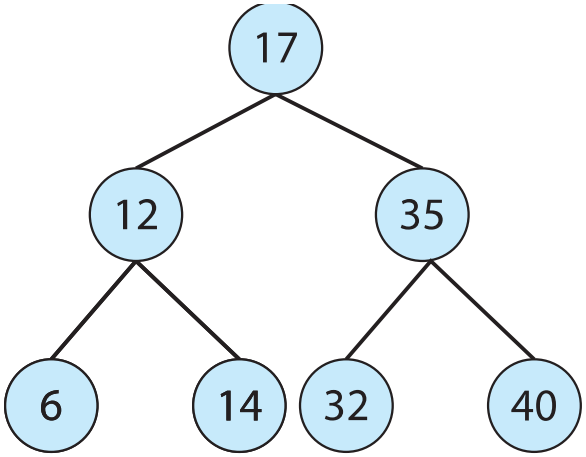


Kernel Data Structures

- **Stack**
  - Last in first out (LIFO)
  - Widely used when invoking function calls
- **Queue**
  - First in first out (FIFO)
  - Widely used in job scheduling

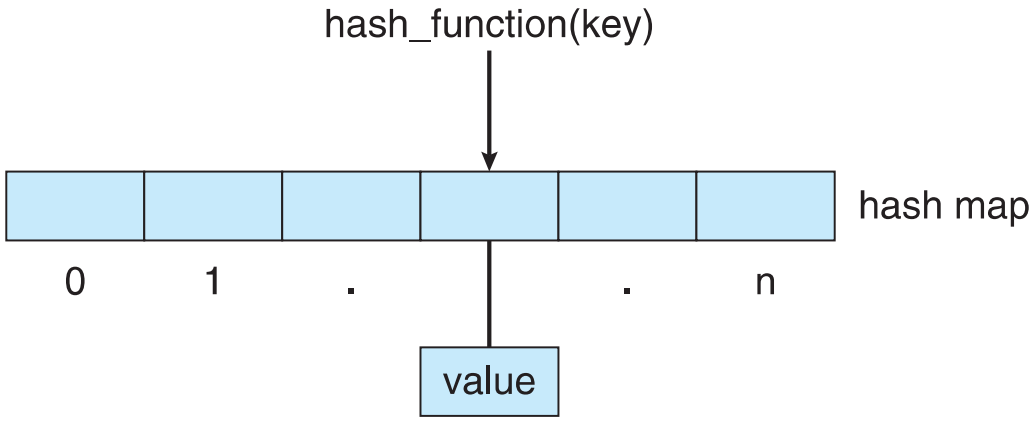
Kernel Data Structures

- **Trees**
  - **Binary tree**
  - **Binary search tree**: left <= right
    - Worse-case search performance is *O(n)*
  - **Balanced binary search tree**
    - Worse-case search performance is *O(lg n)*



Kernel Data Structures

- **Hash function**
  - Takes data as input, performs numeric operation on the data, and returns a numeric value
  - Retrieve data: O(1)
  - Hash collision
- **Hash function** can create a **hash map**



Kernel Data Structures

- **Bitmap** – string of *n* binary digits representing the status of *n* items
- Pros:
  - Space efficiency
- Example: used to indicate the availability of disk blocks
- Linux data structures defined in ***include*** files `<linux/list.h>`, `<linux/kfifo.h>`, `<linux/rbtree.h>`

MISC

Protection and Security, Computing Environments and Open-sourced OS

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, 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

Computing Environments - Traditional

- Stand-alone general purpose machines
- Blurred as most systems interconnect with others (i.e., the Internet)
  - **Portals** provide web access to internal systems
  - **Network computers** (**thin clients**) are like Web terminals
  - Mobile computers interconnect via **wireless networks**
- Networking becoming ubiquitous – even home systems use **firewalls** to protect home computers from Internet attacks

Computing Environments - Mobile

- Handheld smartphones, tablets, etc
- What is the functional difference between them and a “traditional” laptop?
  - Extra feature – more OS features (GPS, gyroscope)
  - Allows new types of apps like ***augmented reality***
  - Use IEEE 802.11 wireless, or cellular data networks for connectivity
- Leaders are **Apple iOS** and **Google Android**

Computing Environments – Distributed

- Distributed computing
  - Collection of separate, possibly heterogeneous, systems networked together
  - **Network** is a communication path, **TCP/IP** most common
    - **Local Area Network** (**LAN**)
    - **Wide Area Network** (**WAN**)
    - **Metropolitan Area Network** (**MAN**)
    - **Personal Area Network** (**PAN**)
  - **Network Operating System** provides features between systems across network
    - Communication scheme allows systems to exchange messages
    - Illusion of a single system

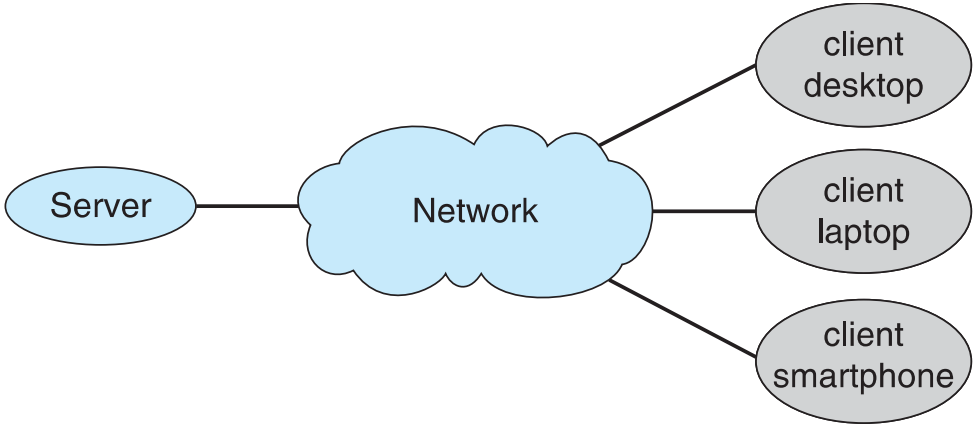
Computing Environments – Client-Server

Client-Server Computing

Dumb terminals supplanted by smart PCs

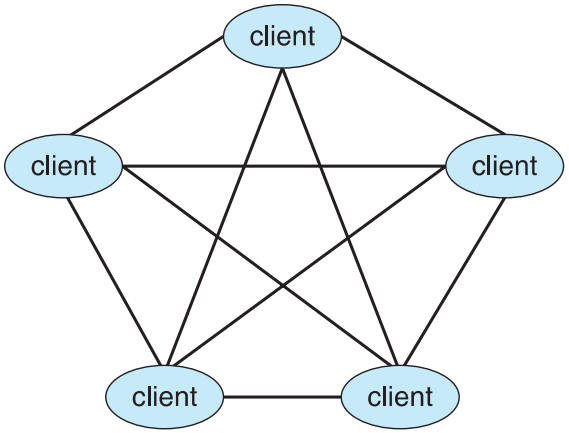
Many systems act as **servers**, responding to requests generated by **clients**

- **Compute-server system** provides an interface to client to request services (i.e., database)
- **File-server system** provides interface for clients to store and retrieve files



Computing Environments - Peer-to-Peer

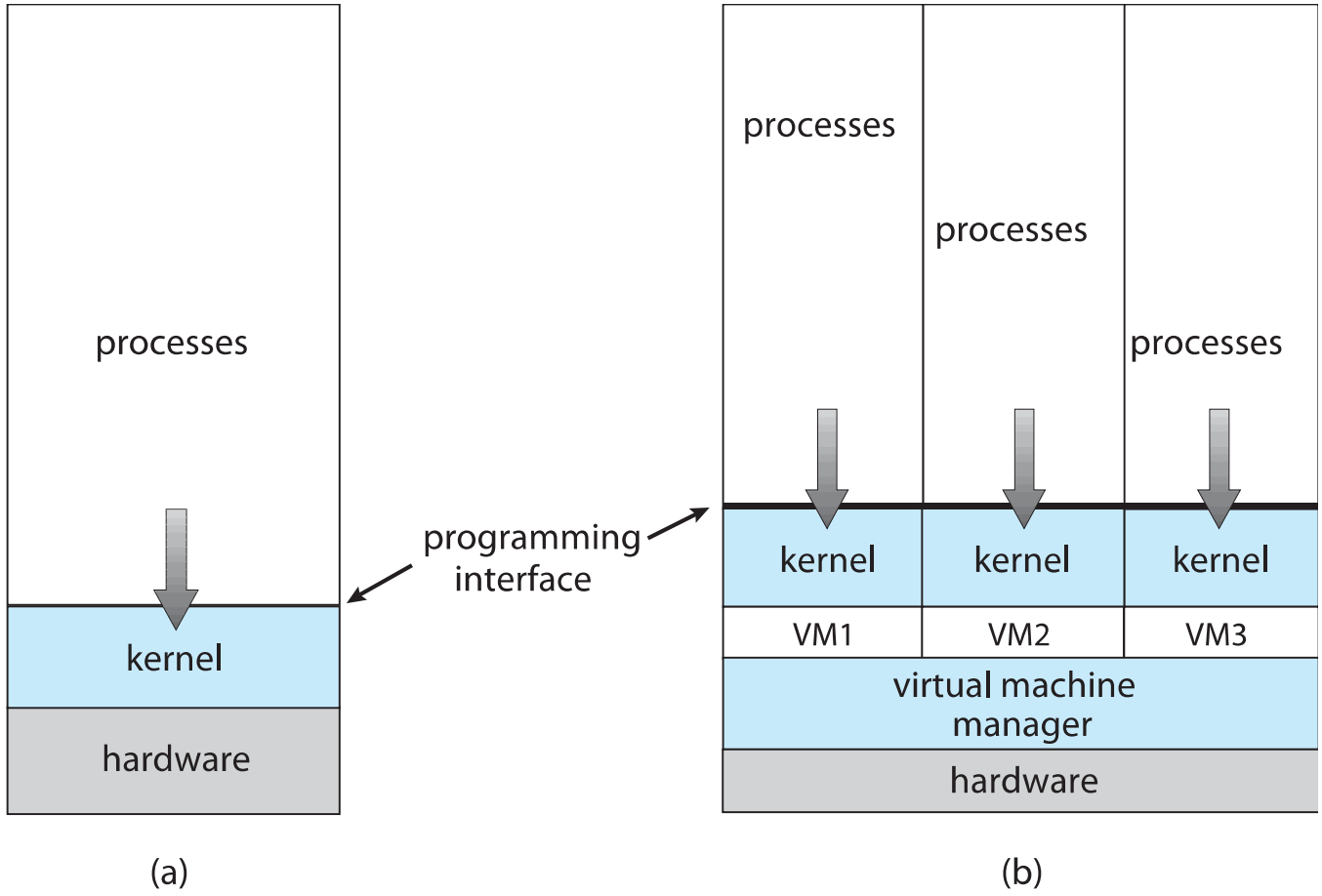
- Another model of distributed system, does not distinguish clients and servers
  - Instead all nodes are considered peers
  - May each act as client, server or both
  - Node must join P2P network
    - Registers its service with central lookup service on network, or
    - Broadcast request for service and respond to requests for service via ***discovery protocol***
  - Examples include BitTorrent



Computing Environments - Virtualization

- Allows OSes to run applications within other OSes
- **Emulation** used when source CPU type is different from target type (i.e. PowerPC to Intel x86)
  - Generally slowest method
  - Every machine-level instruction must be translated
- **Virtualization** – OS natively compiled for CPU, running **guest** OSes also natively compiled
  - Running multiple VMs allows many users to run tasks on a system designed for a single user
  - **VMM** (Virtual Machine Manager) provides virtualization services

Computing Environments - Virtualization

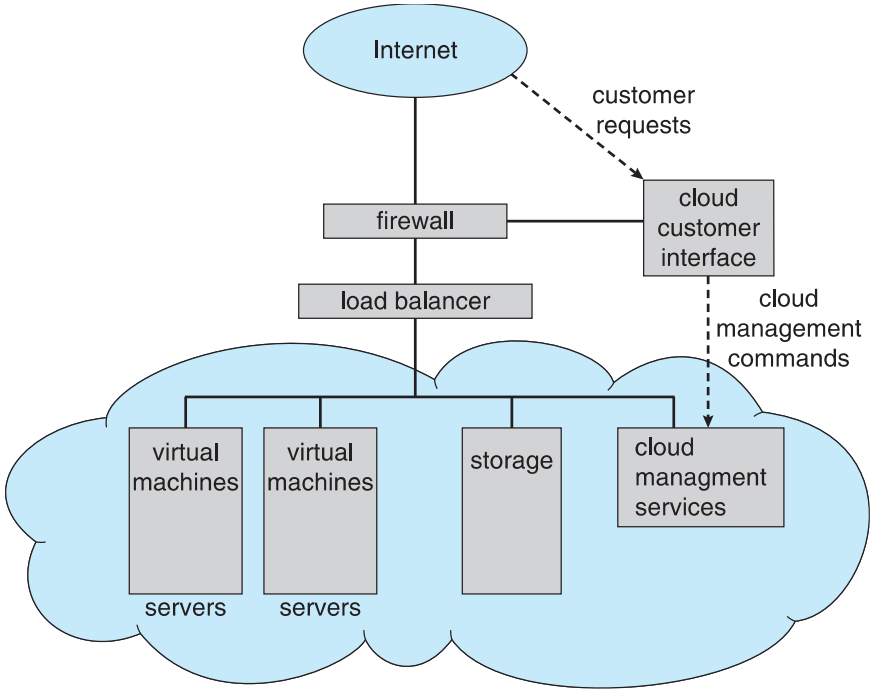


Computing Environments – Cloud Computing

- Delivers computing, storage, even apps as a service across a network
- Logical extension of virtualization because it uses virtualization as the base for it functionality.
  - Amazon **EC2** has thousands of servers, millions of virtual machines, petabytes of storage available across the Internet, pay based on usage
- Many types
  - **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)

Computing Environments – Cloud Computing

- Cloud computing environments composed of traditional OSes, plus VMMs, plus cloud management tools
  - Internet connectivity requires security like firewalls
  - Load balancers spread traffic across multiple applications



Computing Environments – Real-Time Embedded Systems

- Real-time embedded systems: most prevalent form of computers
  - Car engines, robots, DVDs, etc.
- Real-time OS has well-defined fixed time constraints
  - Processing ***must*** be done within constraint
  - Correct operation only if constraints met
- Many other special computing environments as well
  - Some have OSes, some perform tasks without an OS

Open-Source Operating Systems

- Operating systems made available in source-code format rather than just binary **closed-source**
- Started by **Free Software Foundation (FSF)**, which has "copyleft" **GNU Public License (GPL)**
- Examples include **GNU/Linux** and **BSD UNIX** (including core of **Mac OS X**)
- Can use VMM like VMware Player (Free on Windows), Virtualbox (open source and free on many platforms - http://www.virtualbox.com)
  - Use to run guest operating systems for exploration

Summary

- OS Overview
  - OS Concept
  - Multiprogramming & Multitasking
  - Dual Mode & System Call
- OS Components
  - Process Management
  - Memory Management
  - Storage Management
- Computer System Organization & Architecture
  - Interrupt

End of Chapter 1