CS/COE 1550
Intro to Operating Systems

Spring 2018

Sherif Khattab

ksm73@pitt.edu

6307 Sennott Square

(Some slides are from **Silberschatz, Galvin and Gagne ©2013)**

Course Goal

- Demystify a good portion of the magic about how computers work

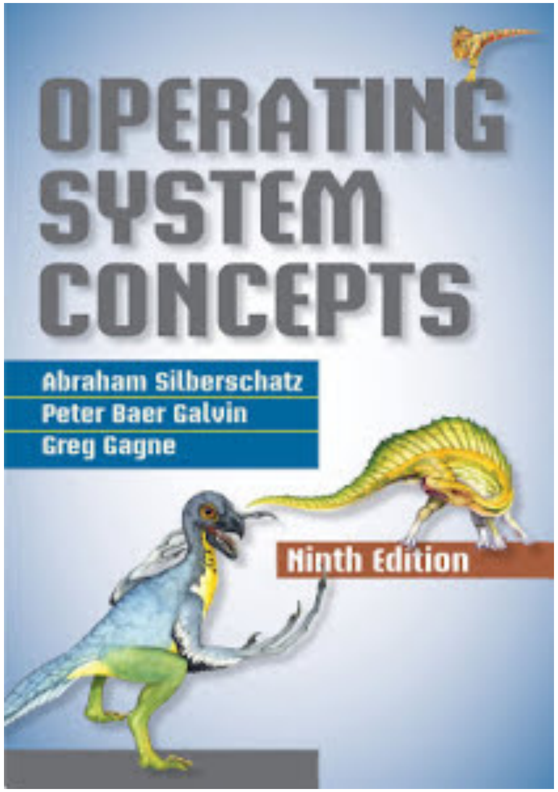
Contact Info

- **Course website:** http://www.cs.pitt.edu/~skhattab/cs1550/
- **Instructor:** Sherif Khattab ksm73@pitt.edu
  - OH: MW 9-1pm (other times available by appointment)
    - Please book at http://khattab.youcanbook.me
  - Office: 6307 Sennott Square
- **Teaching Assistants**
  - Nannan Wen naw66@pitt.edu

Grades

- **Grading**
  - 45% exams: 30% on higher grade and 15% on lower
  - 40% on 4 projects
    - System programming using C
    - Modifying the kernel of the MIT's xv6 operating system
  - 15% recitation: 2 quizzes (10%) and participation (5%)
- Projects posted and submitted on **Courseweb**

Textbook

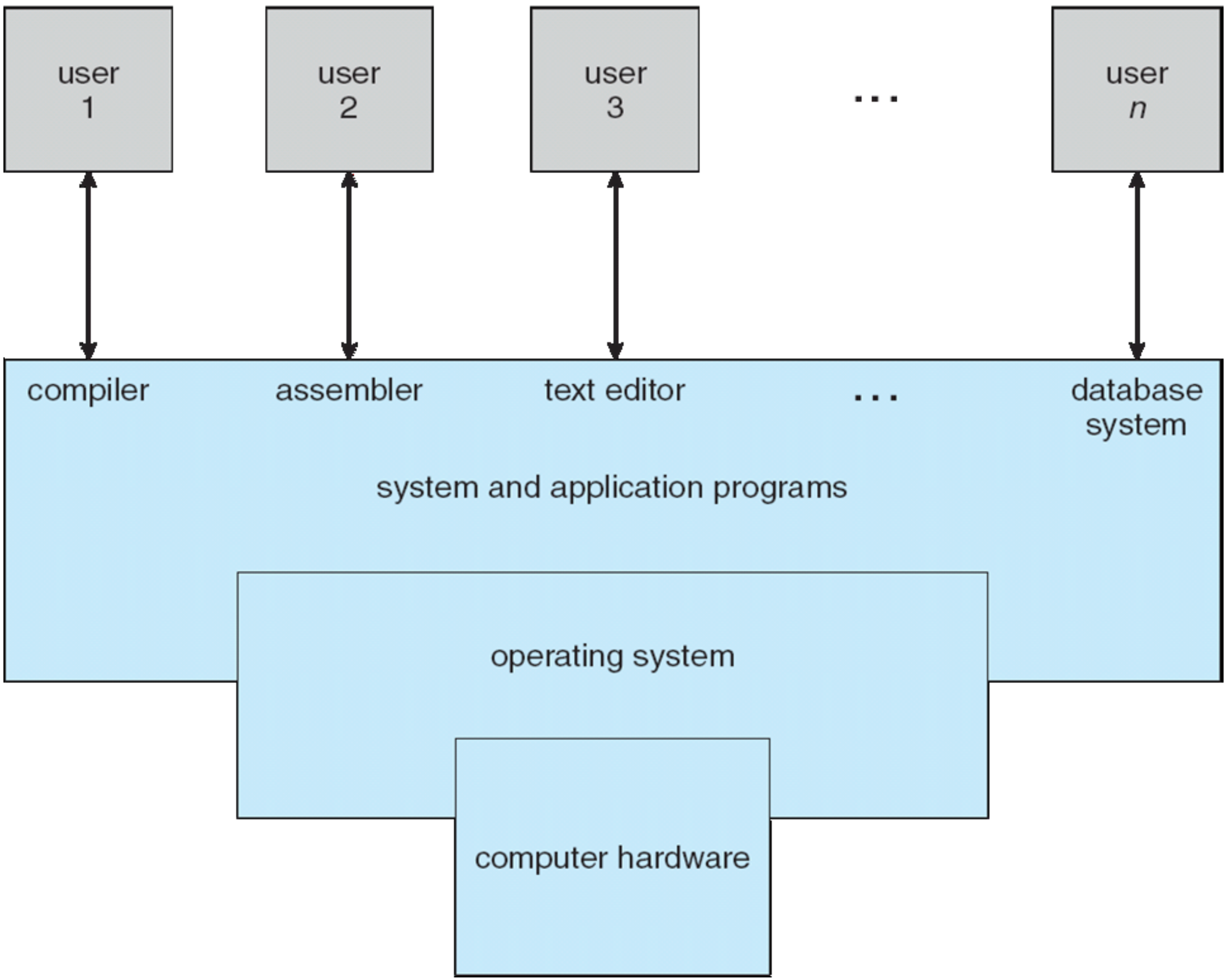


Operating System Concepts (9th Edition)

Silberschatz, Galvin, and Gagne

What is an Operating System?

A program that acts as an intermediary between a user of a computer and the computer hardware



What does an OS do?

- Manages (controls and arbitrates) resources
  - Processors, Memory, Input/output devices, Communication devices, Storage, Software applications
  - Conflicting goals:
    - Performance *vs.* utilization
- Provides abstractions to application programs
  - Ease of use
  - Virtualization
- "The one program running at all times on the computer" is the OS kernel.
  - Everything else is either
    - a system program (ships with the operating system) , or
    - an application program.

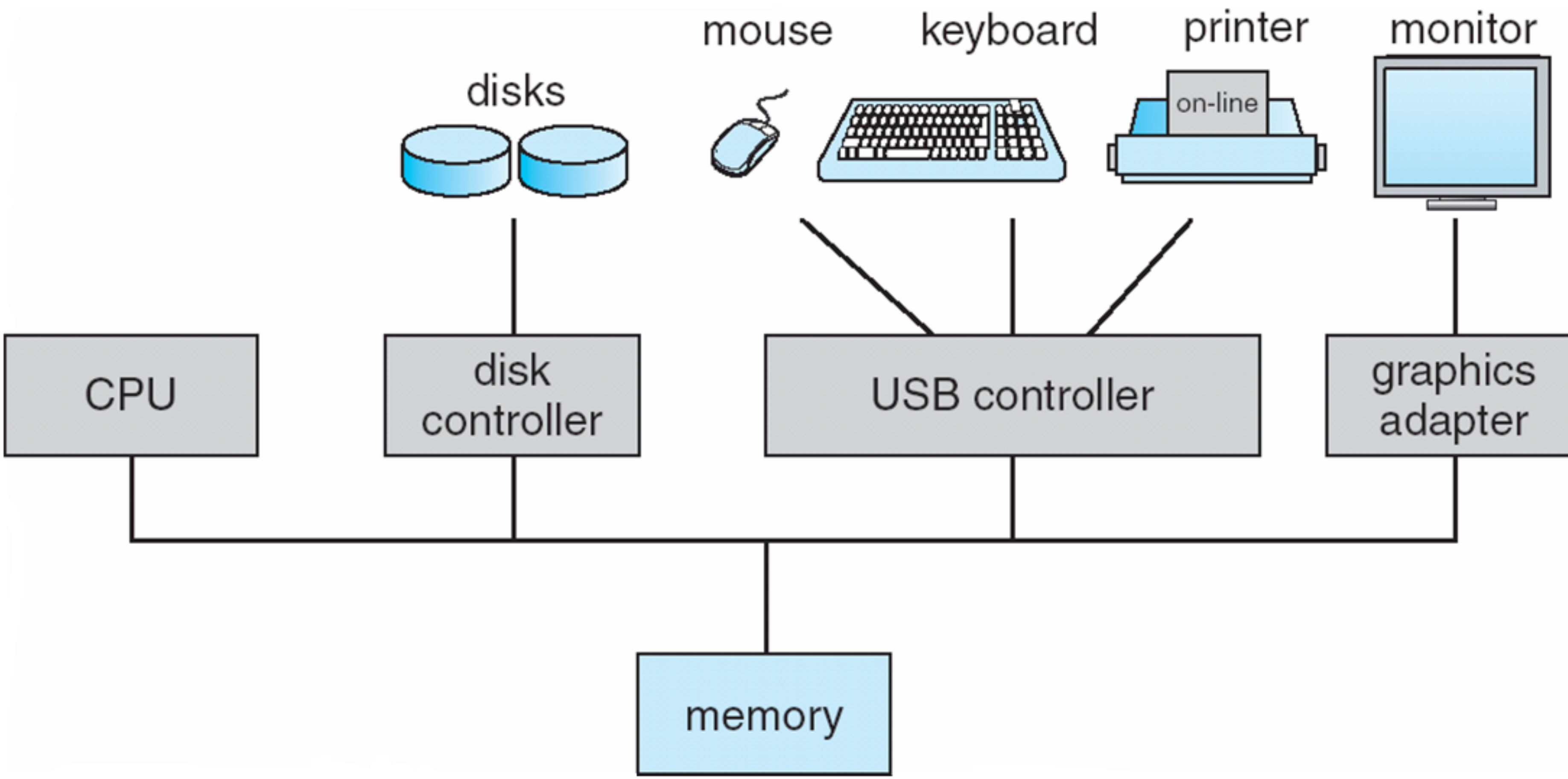
Who guards the guards?

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 and starts execution

How complex is the OS's job?

Let's look at one of the resources managed by the OS:
I/O devices



I/O Devices

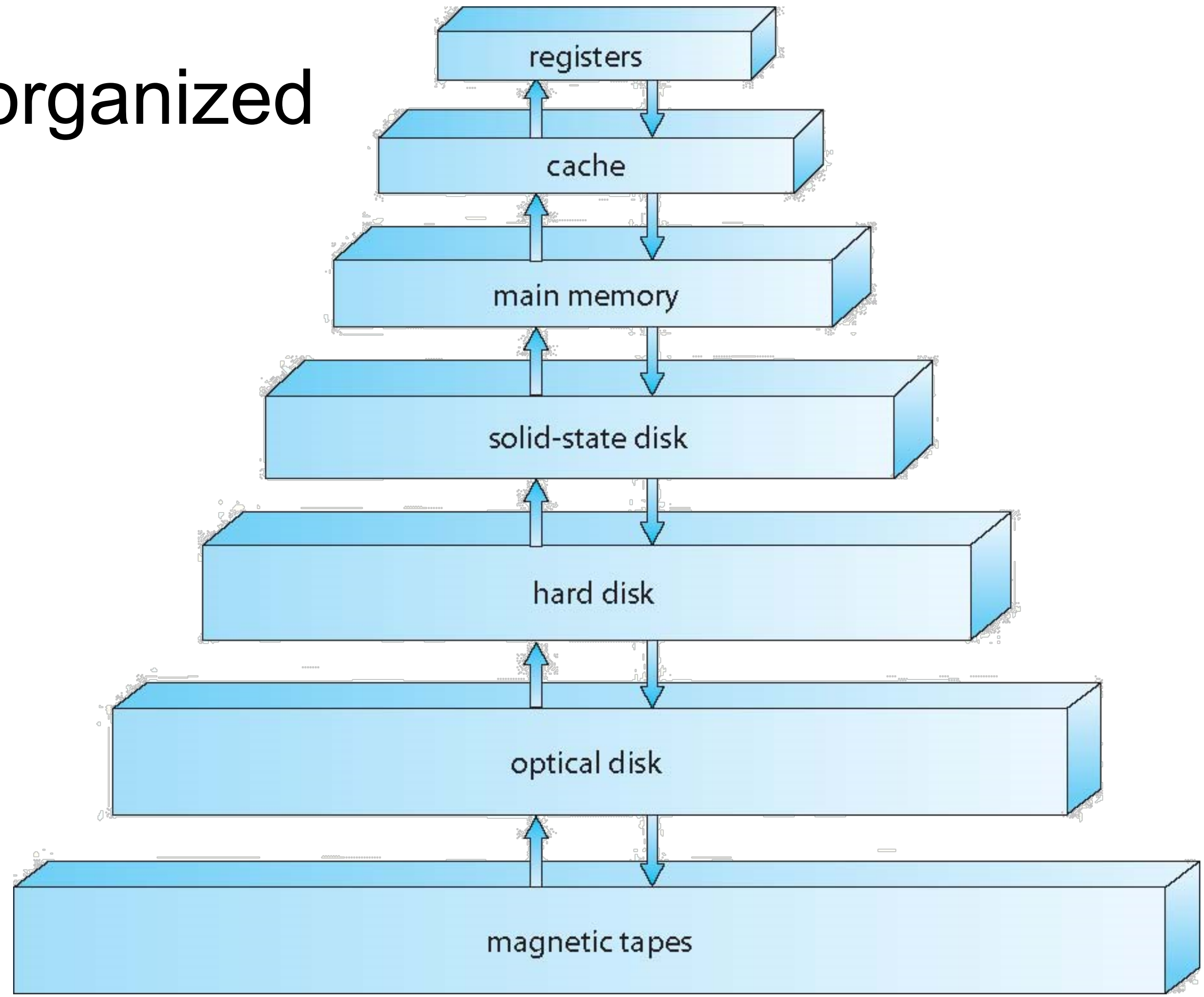
- I/O devices and the CPU can execute concurrently
- Device controller
  - in charge of a particular device type
  - has a local buffer
- CPU moves data from/to main memory to/from local buffers
- I/O is from the device to local buffer of controller
- Device controller informs CPU that it has finished its operation by causing an interrupt
- **Device Driver** for each device controller
  - Provides uniform interface between controller and kernel

Interrupts

- Interrupt transfers control to the interrupt service routine (ISR)
- ISRs are segments of code determine what action should be taken for each type of interrupt
- The **interrupt vector** contains the addresses of all the service routines
- A **trap** or **exception** is a software-generated interrupt caused either by an error or a user request

Storage Devices

- Let's look at a particular type of I/O devices: storage devices
- Storage systems organized in hierarchy
  - Speed
  - Cost
  - Volatility



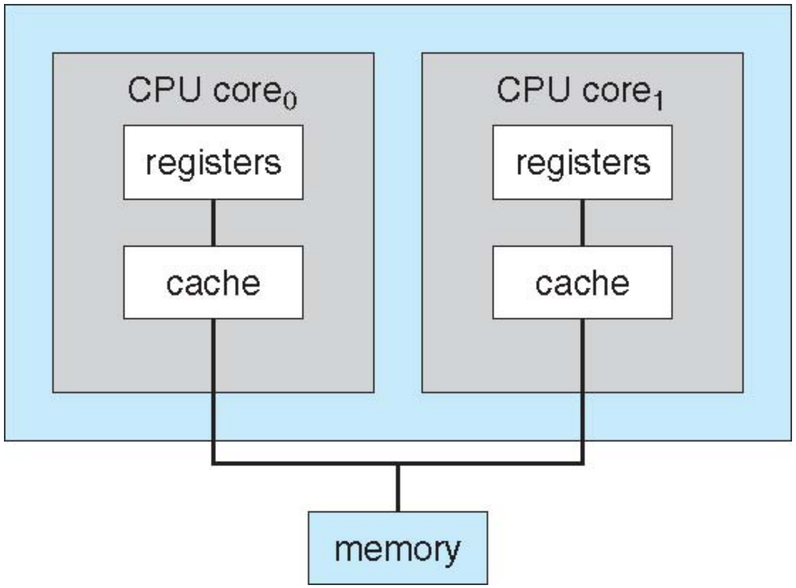
Caching

- copying some information into faster storage system temporarily
  - main memory can be viewed as a cache for secondary storage
- performed at many levels (in hardware, operating system, software)
- Cache checked first to determine if information is there
- Cache management is an important design problem
  - Cache size and replacement policy

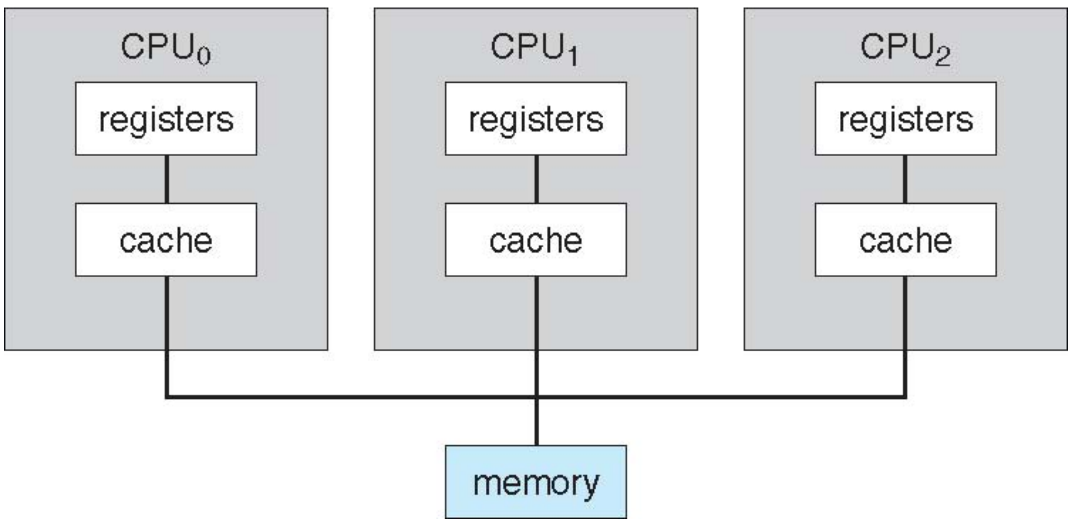
OS and Computer System Architecture

- single general-purpose processor
  - special-purpose processors as well

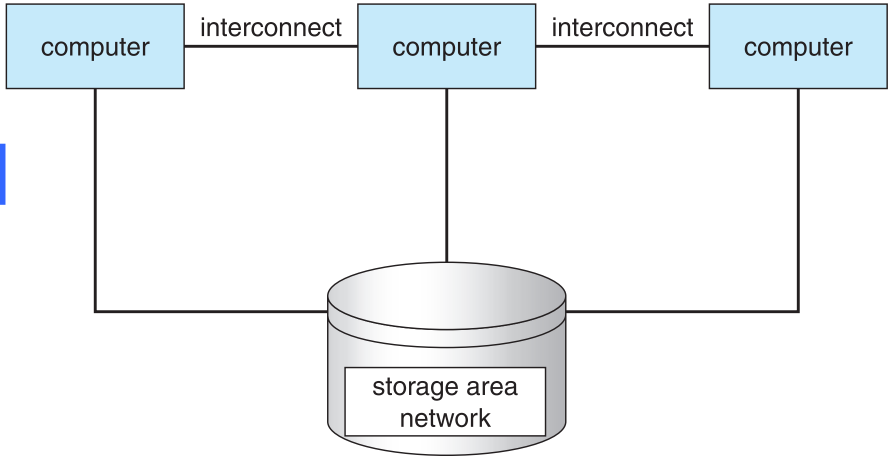
Multi-core



Multiprocessors



Clustered



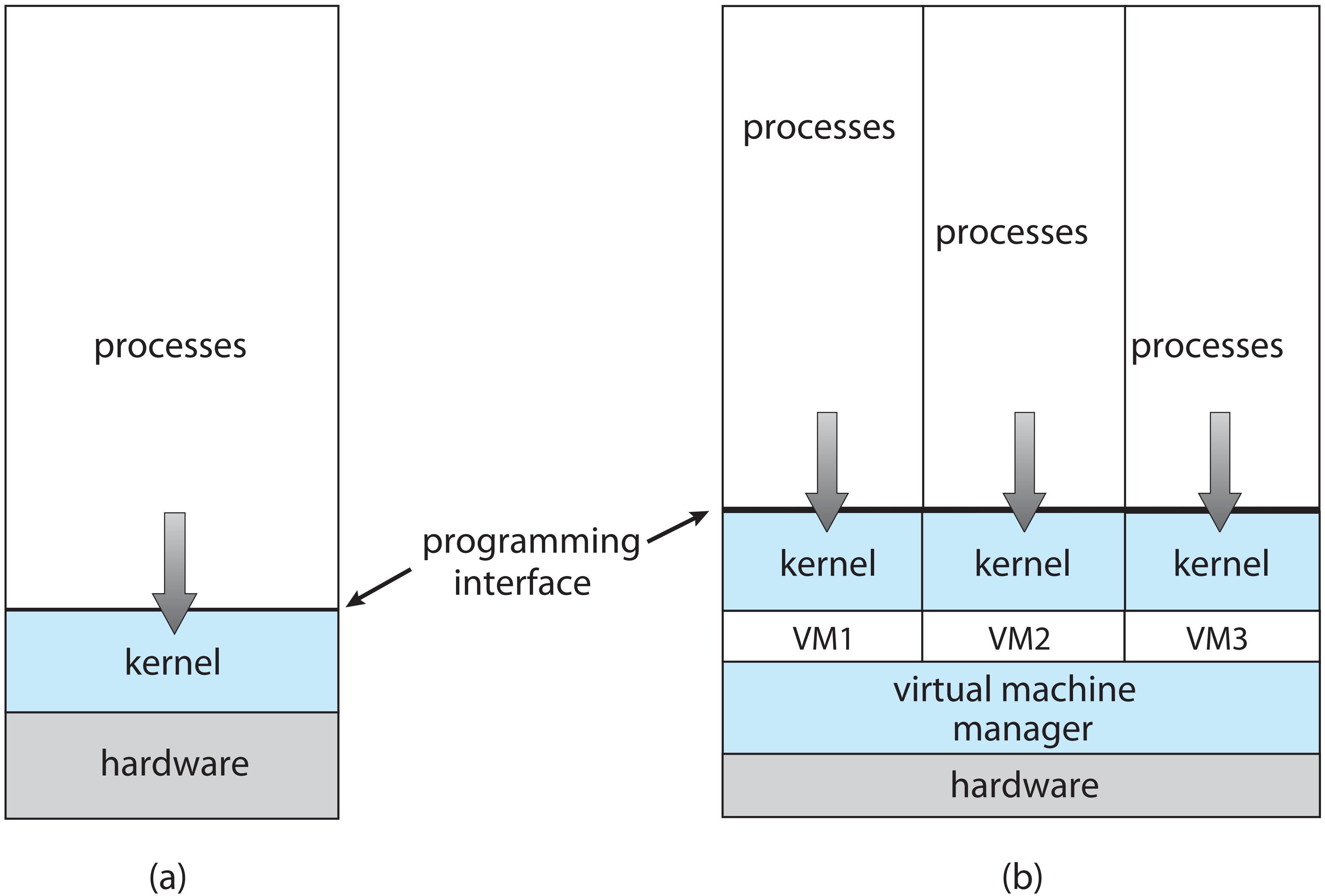
OS and Computing Environments

- Traditional
  - Stand-alone general purpose machines
  - Massively and ubiquitously networked
- Mobile
  - smartphones, tablets, etc.
  - more OS features (GPS, sensors)

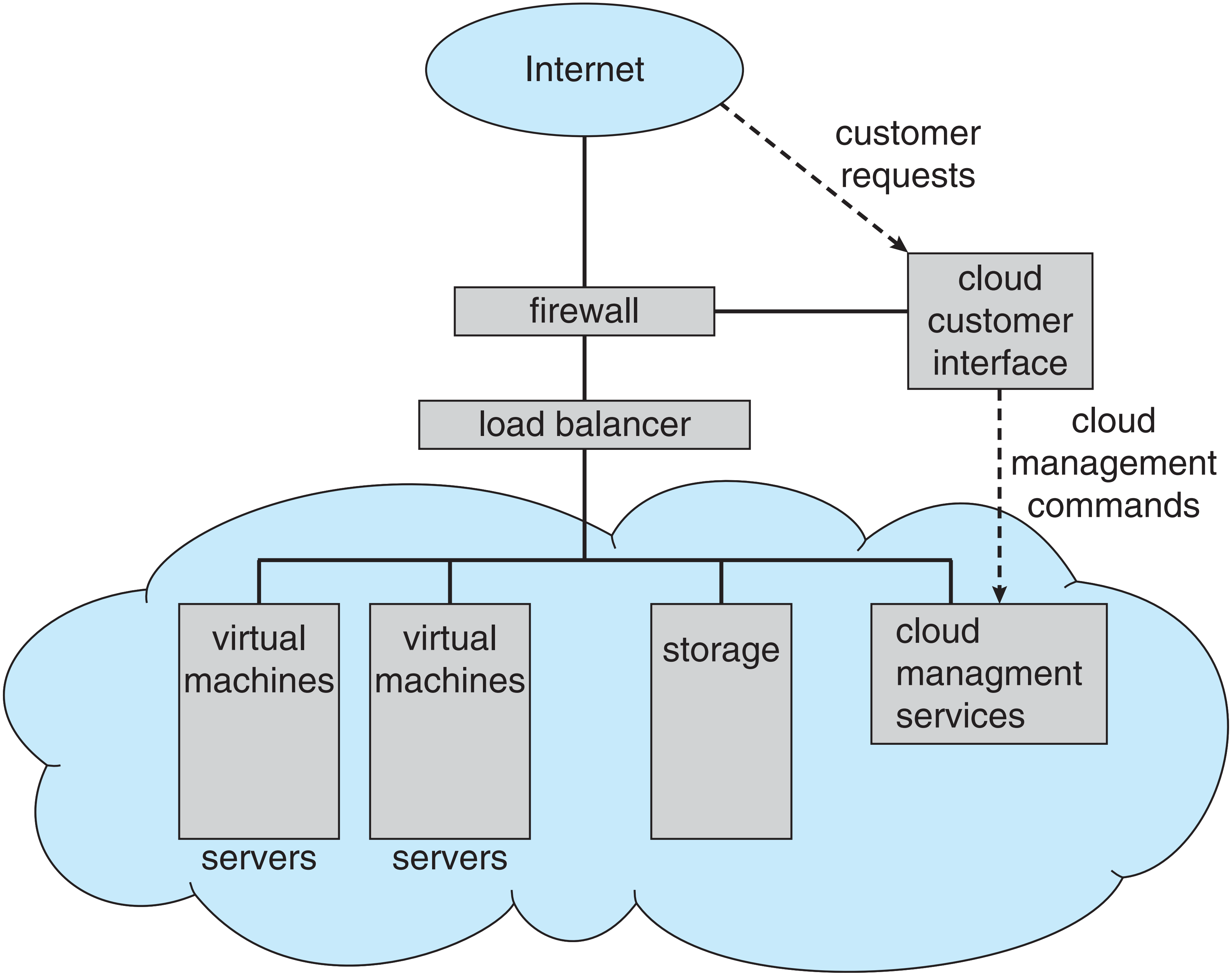
Computing Environments – Distributed Computing

- Distributed computing
  - separate, possibly heterogeneous, systems networked
  - Network Operating System
    - Illusion of a single system
  - Client-Server
    - Compute-server system
    - File-server system
  - Peer-to-Peer
    - does not distinguish clients and servers
    - central lookup service on network or service *discovery protocol*
    - e.g., Skype

Computing Environments - Virtualization



Computing Environments – Cloud Computing



- special purpose, limited-purpose, **real-time OS**
- Real-time OS has well-defined fixed time constraints
  - Processing ***must*** be done within constraint
  - Correct operation only if constraints met

Operating-System Operations

Interrupt driven (hardware and software)

- Hardware interrupt by one of the devices
- Software interrupt (**exception** or **trap):**
  - Request for operating system service
  - Software error (e.g., division by zero)
  - Other process problems include infinite loop, processes trying to modify each other or the operating system

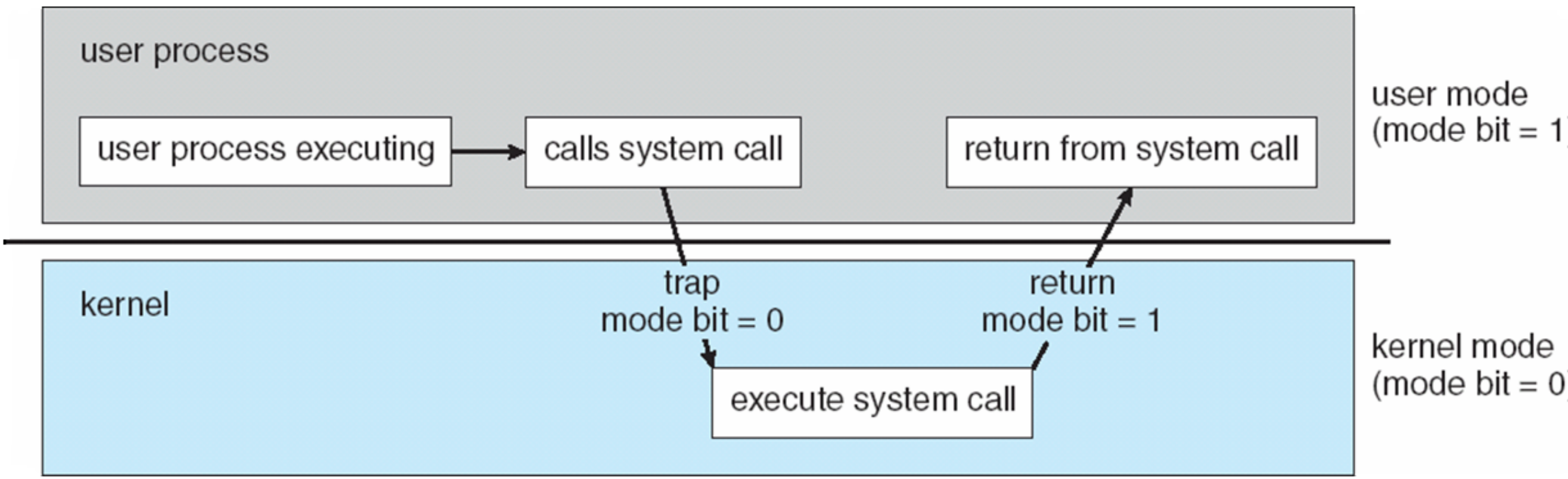
Operating-System Operations (cont.)

- **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
- 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 the computer after some time period
- Operating system set the counter (privileged instruction)
- When counter zero generate an interrupt



Process Management

- A process is a program in execution. 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
  - CPU, memory, I/O, files
  - Initialization data
- Process termination requires reclaim of any reusable resources
- Single-threaded process has one **program counter** specifying location of next instruction to execute
  - Process executes instructions sequentially, one at a time, until completion

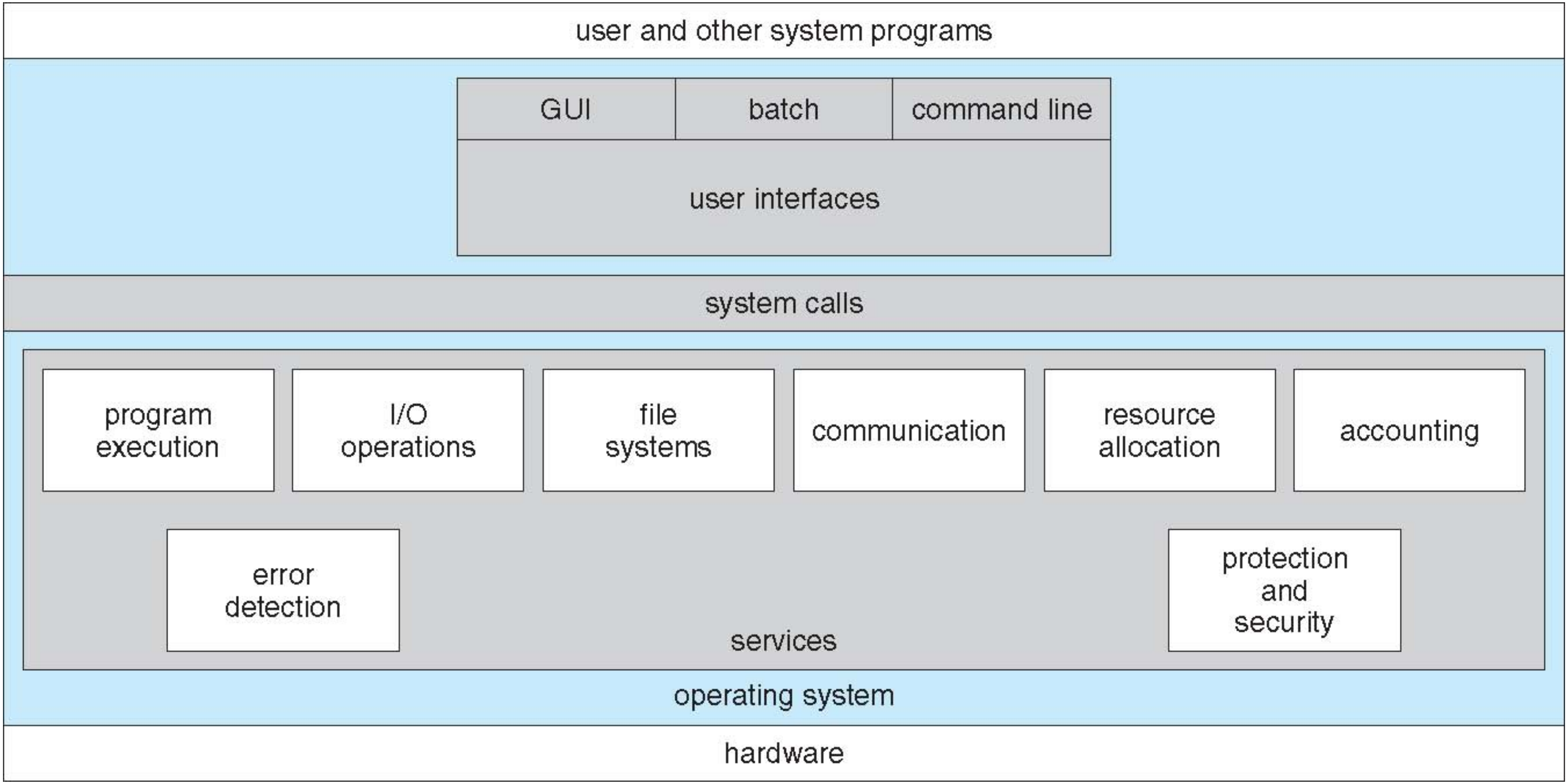
Memory Management

- 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 and when
- 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

Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
- File-System management
  - Files usually organized into directories
  - Access control on most systems 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

A View of Operating System Services

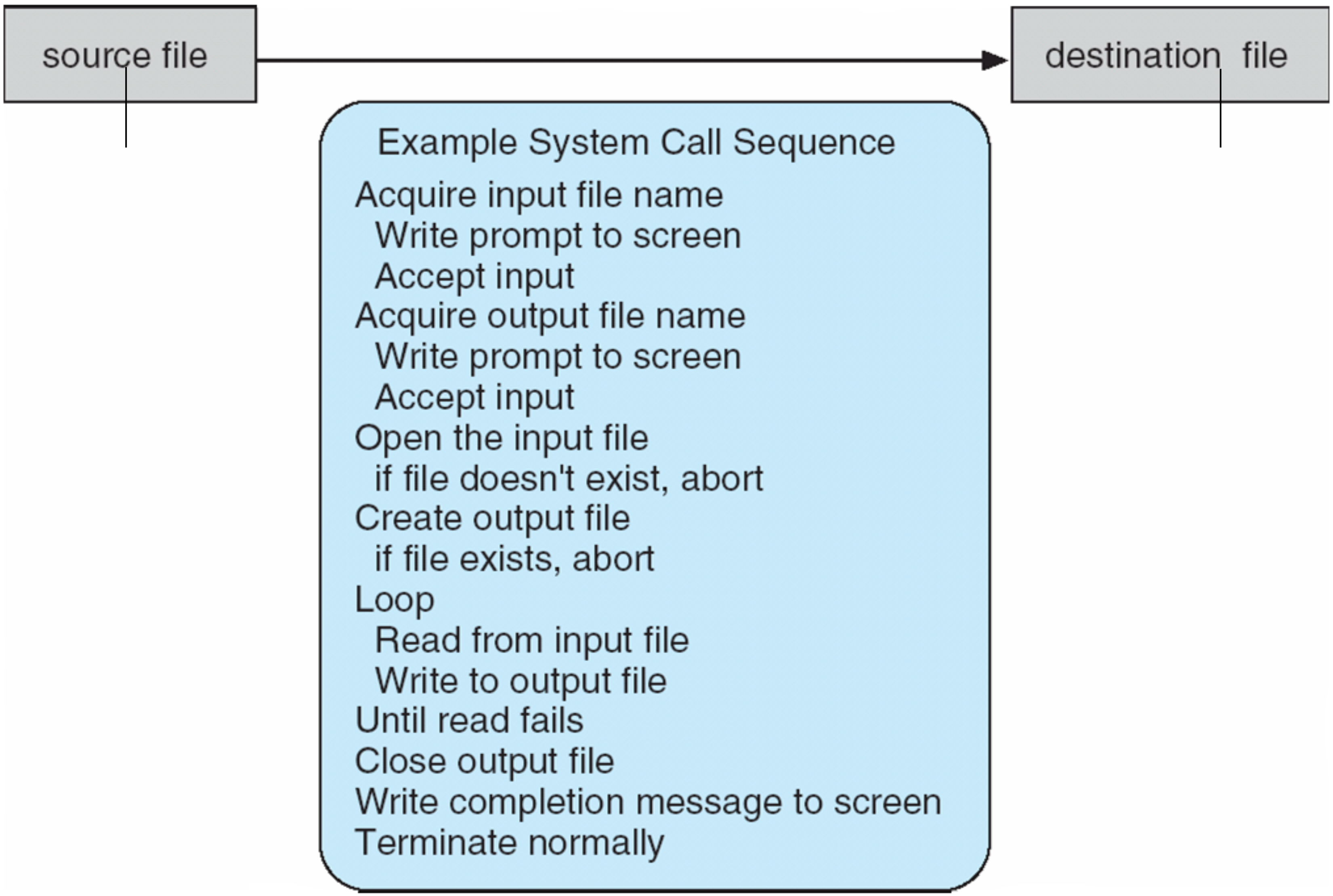


System Calls

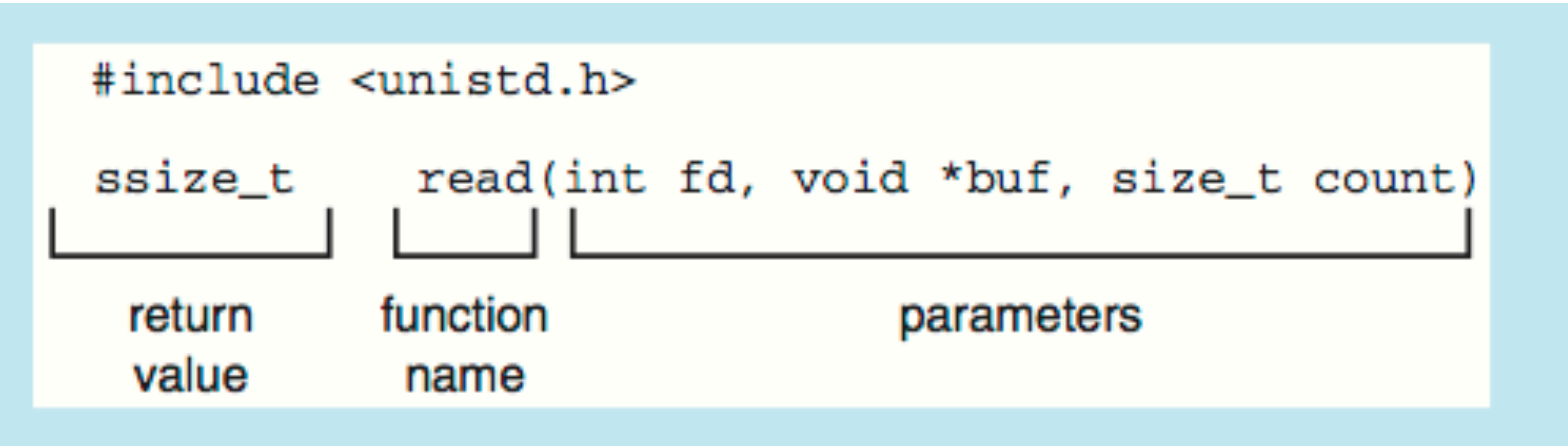
- Programming interface to the services provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface** (**API**) rather than direct system call use
- Three most common APIs are Win32 API for Windows, POSIX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

Example of System Calls

System call sequence to copy the contents of one file to another file



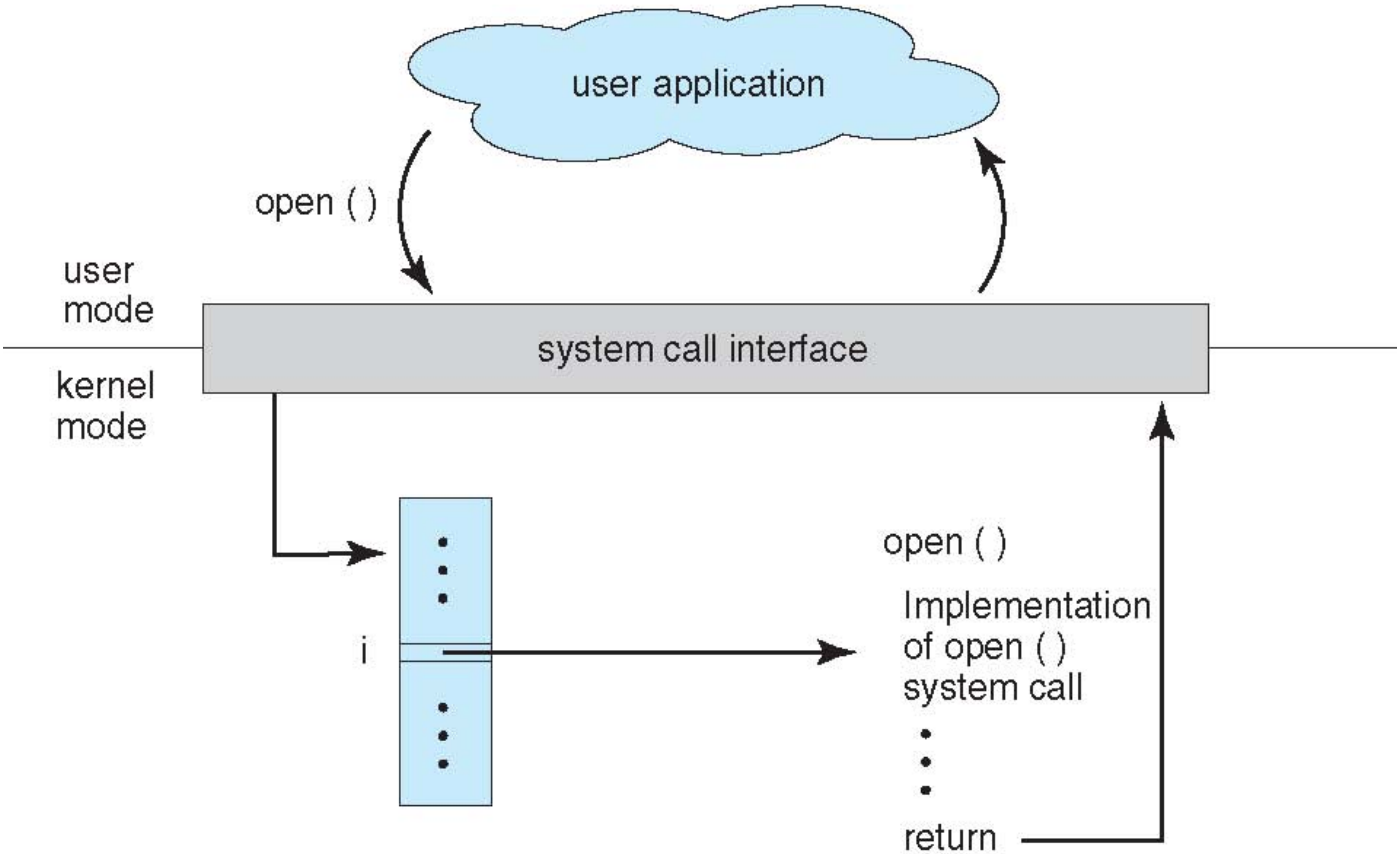
Example of Standard API



System Call Implementation

- Typically, a number associated with each system call
  - **System-call interface** maintains a table indexed according to these numbers
- The system call interface invokes the intended system call in OS kernel and returns status of the system call and any return values
- The caller need know nothing about how the system call is implemented

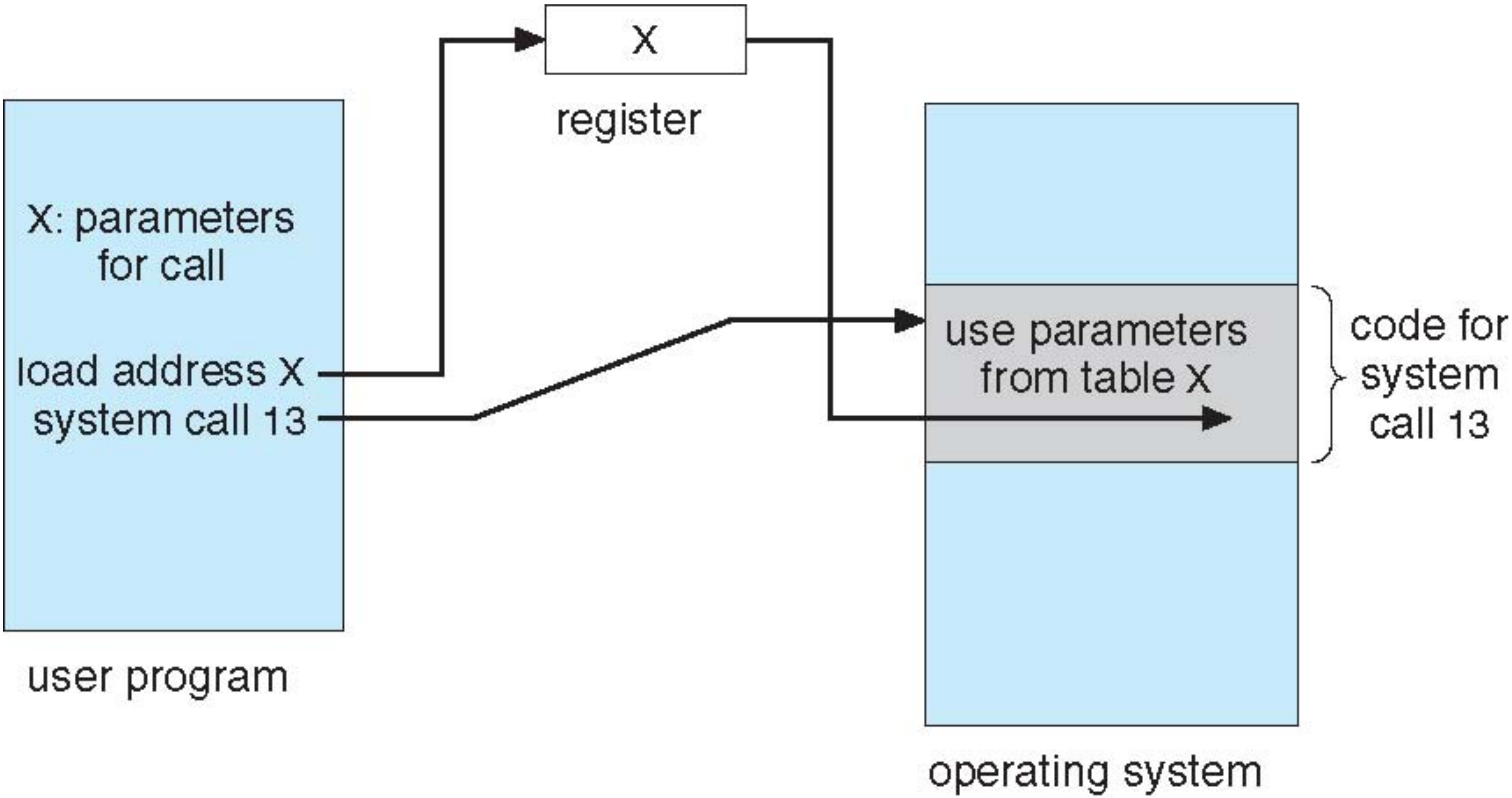
API – System Call – OS Relationship



System Call Parameter Passing

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

Parameter Passing via Table



Types of System Calls

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

Types of System Calls

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

Types of System Calls (Cont.)

- Communications
  - create, delete communication connection
  - send, receive messages if **message passing model** to **host name** or **process name**
    - From **client** to **server**
  - **Shared-memory model** create and gain access to memory regions
  - transfer status information
  - attach and detach remote devices

Types of System Calls (Cont.)

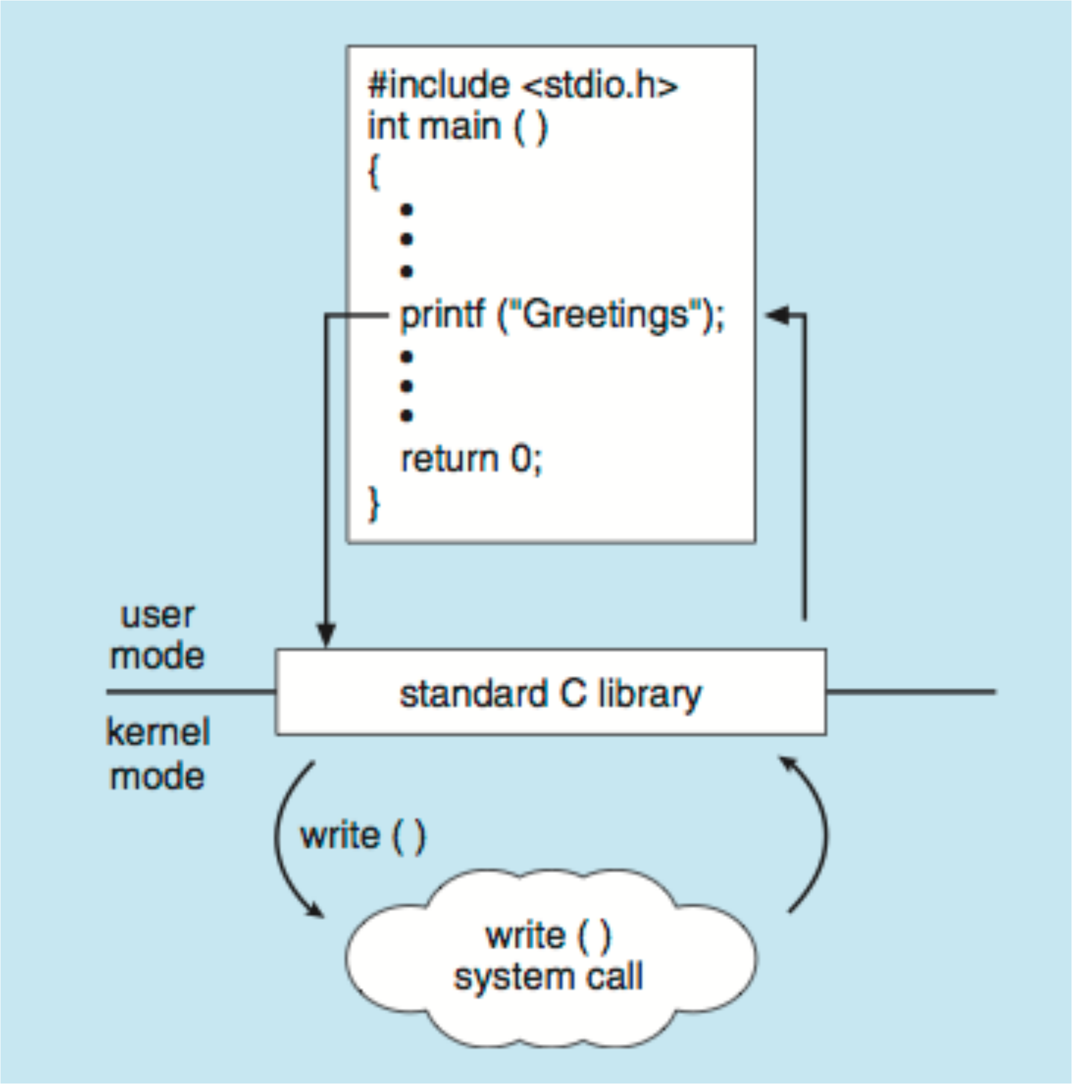
- Protection
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess()	fork()
	ExitProcess()	exit()
	WaitForSingleObject()	wait()
File Manipulation	CreateFile()	open()
	ReadFile()	read()
	WriteFile()	write()
	CloseHandle()	close()

Standard C Library Example

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



Operating System Design and Implementation

- Important principle to separate
- **Policy**: ***What*** will be done?
  Mechanism: ***How*** to do it?
- Mechanisms determine how to do something, policies decide what will be done
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later (example – timer)
- Specifying and designing an OS is highly creative task of **software engineering**

Operating System Structure

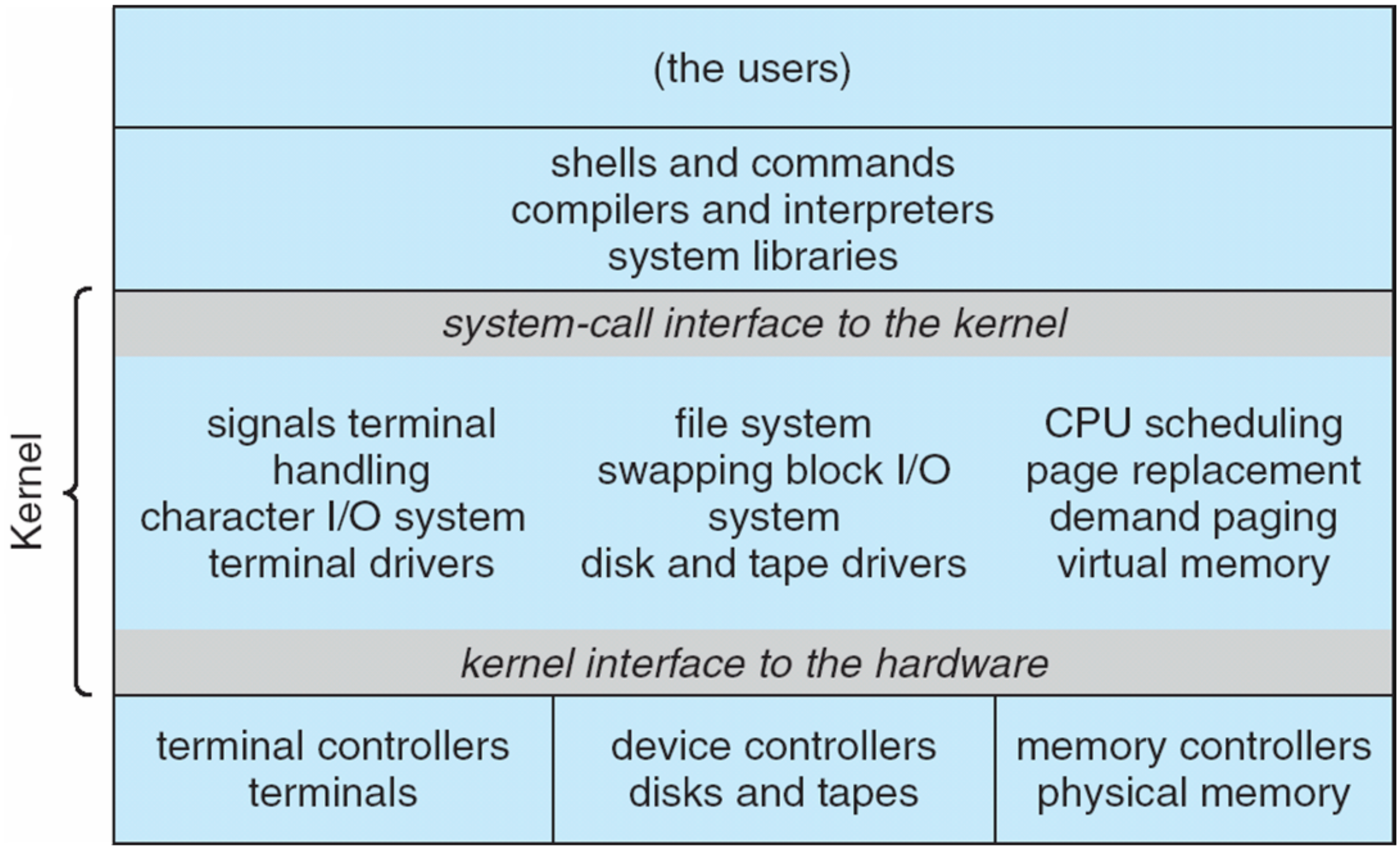
General-purpose OS is very large program

Various ways to structure ones

- Simple structure – MS-DOS
- More complex -- UNIX
- Layered – an abstrcation
- Microkernel -Mach

Traditional UNIX System Structure

Beyond simple but not fully layered



Microkernel System Structure