

Chapter 2: Operating-System Structures

Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

Objectives

- To describe the **services** an operating system provides to users, processes, and other systems
- To discuss the various ways of **structuring** an operating system
- To explain how operating systems are installed and customized and how they boot

Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

Operating System Services

These are from
the user's
perspective

- One set of operating-system services provides functions that are helpful to the user:
 - **User interface** - Almost all operating systems have a user interface (UI)
 - ▶ Varies between Command-Line (CLI), Graphics User Interface (GUI), Batch
 - **Program execution** - The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
 - **I/O operations** - A running program may require I/O, which may involve a file or an I/O device.
 - **File-system manipulation** - The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

Operating System Services (Cont.)

These are from
the user's
perspective

- One set of operating-system services provides functions that are helpful to the user (Cont):
 - **Communications** – Processes may exchange information, on the same computer or between computers over a network
 - ▶ Communications may be via shared memory or through message passing (packets moved by the OS)
 - **Error detection** – OS needs to be constantly aware of possible errors
 - ▶ May occur in the CPU and memory hardware, in I/O devices, in user program
 - ▶ For each type of error, OS should take the appropriate action to ensure correct and consistent computing
 - ▶ Debugging facilities can greatly enhance the user's and programmer's abilities to efficiently use the system

Operating System Services (Cont.)

These are from the system perspective

- Another set of OS functions exists for ensuring the efficient operation of the system itself via resource sharing
 - **Resource allocation** - When multiple users or multiple jobs running concurrently, resources must be allocated to each of them
 - ▶ Many types of resources - Some (such as CPU cycles, memory, and file storage) may have special allocation code, others (such as I/O devices) may have general request and release code.
 - **Accounting** - To keep track of which users use how much and what kinds of computer resources
 - **Protection and security** - The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other
 - ▶ **Protection** involves ensuring that all access to system resources is controlled
 - ▶ **Security** of the system from outsiders requires user authentication, extends to defending external I/O devices from invalid access attempts
 - ▶ If a system is to be protected and secure, precautions must be instituted throughout it. A chain is only as strong as its weakest link.

Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

User Operating System Interface - CLI

CLI allows direct command entry

- ▶ Sometimes implemented in **kernel**, sometimes by **systems program**
- ▶ Sometimes multiple flavors implemented – **shells**
- ▶ Primarily fetches a command from user and executes it
 - Sometimes commands built-in (**DOS**), sometimes just names of programs (**Unix**)
 - » If the latter, adding new features doesn't require shell modification

User Operating System Interface - GUI

- User-friendly **desktop** metaphor interface
 - Usually mouse, keyboard, and monitor
 - **Icons** represent files, programs, actions, etc
 - Various mouse buttons over objects in the interface cause various actions (provide information, options, execute function, open directory (known as a **folder**))
 - Invented at Xerox PARC
- Many systems now include both CLI and GUI interfaces
 - Microsoft Windows is GUI with CLI “command” shell
 - Apple Mac OS X as “Aqua” GUI interface with UNIX kernel underneath and shells available
 - Solaris is CLI with optional GUI interfaces (Java Desktop, KDE)

Chapter 2: Operating-System Structures

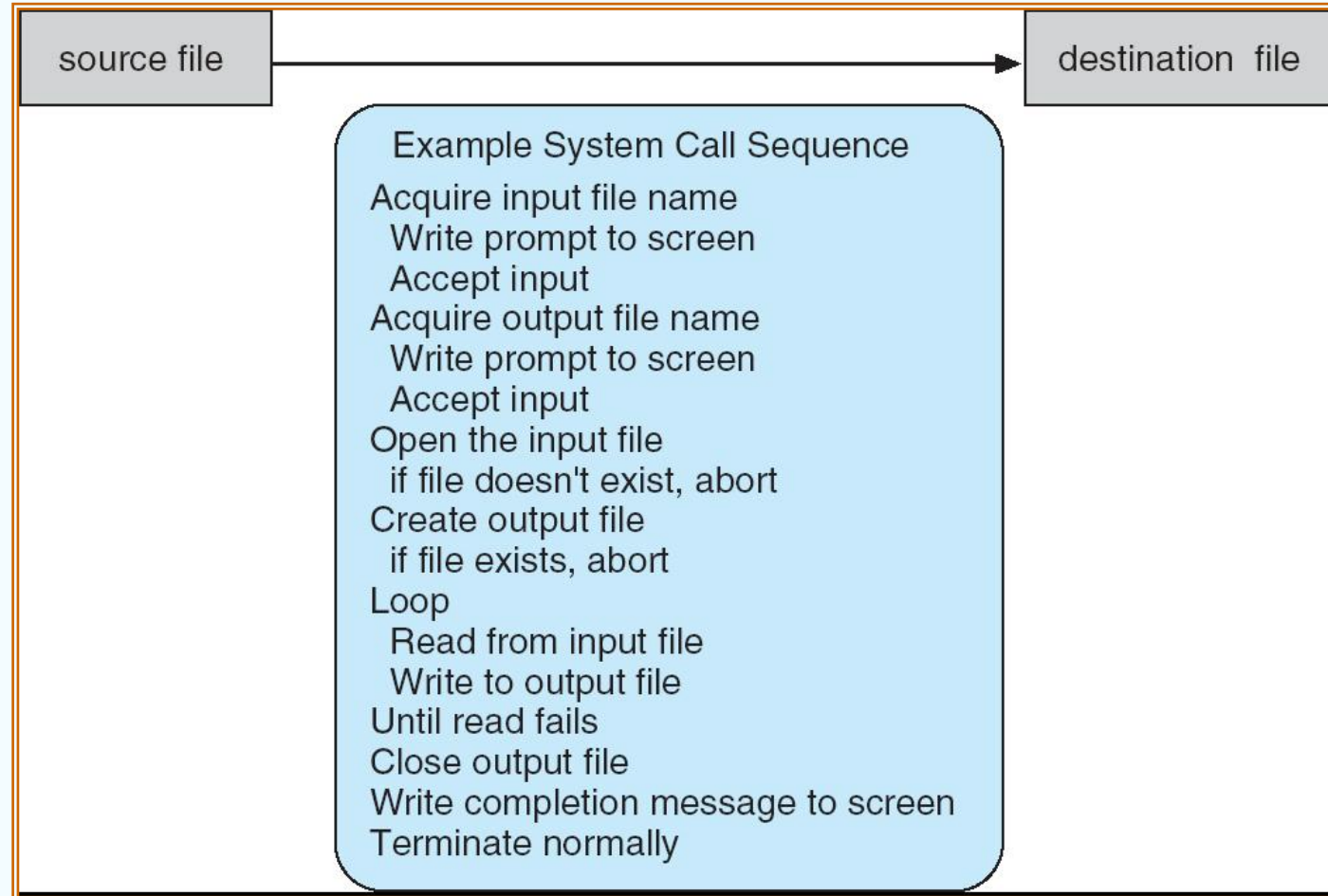
- Operating System Services
- User Operating System Interface
- **System Calls**
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

System Calls

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

Example of System Calls

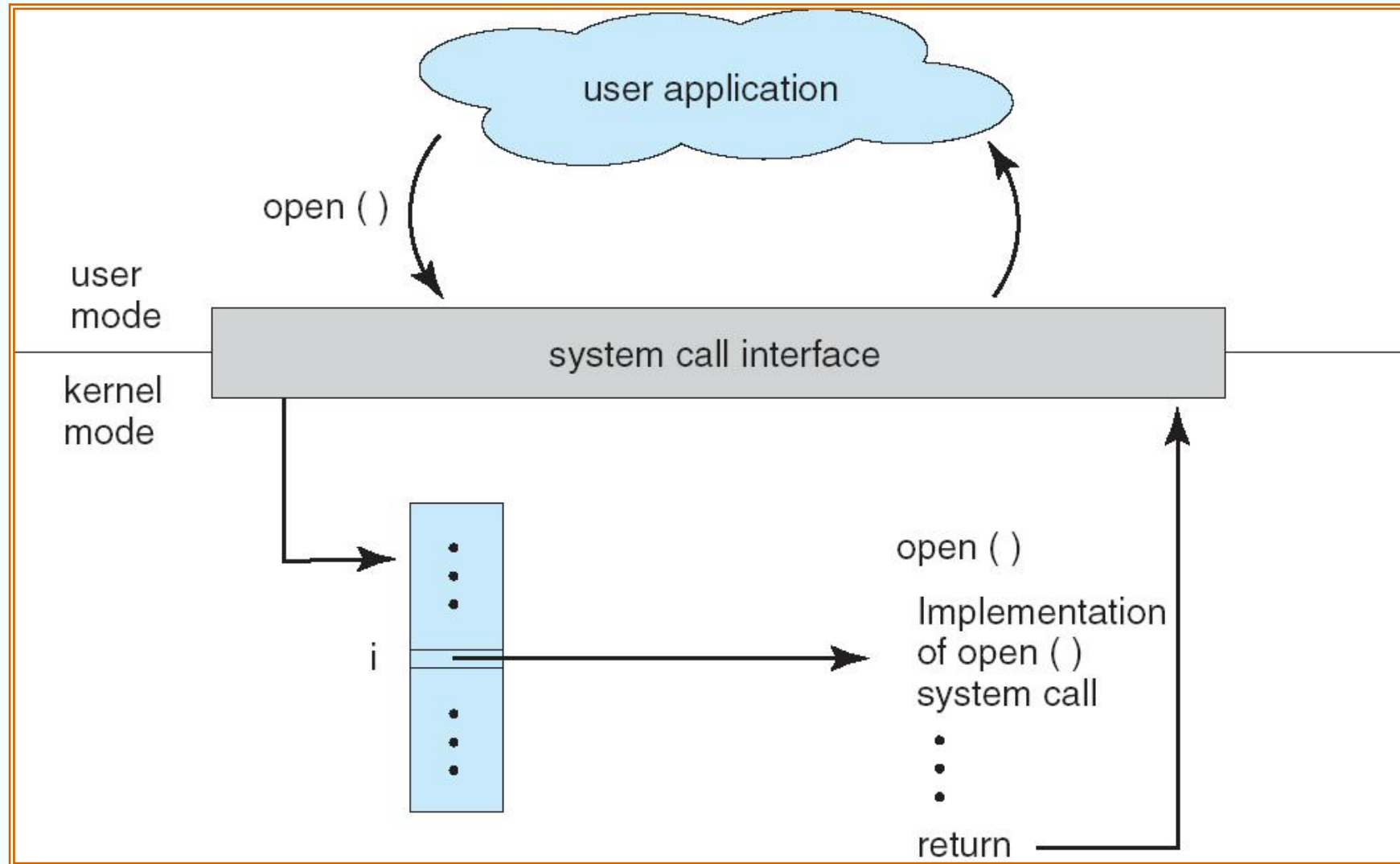
- System call sequence to copy the contents of one file to another file



System Call Implementation

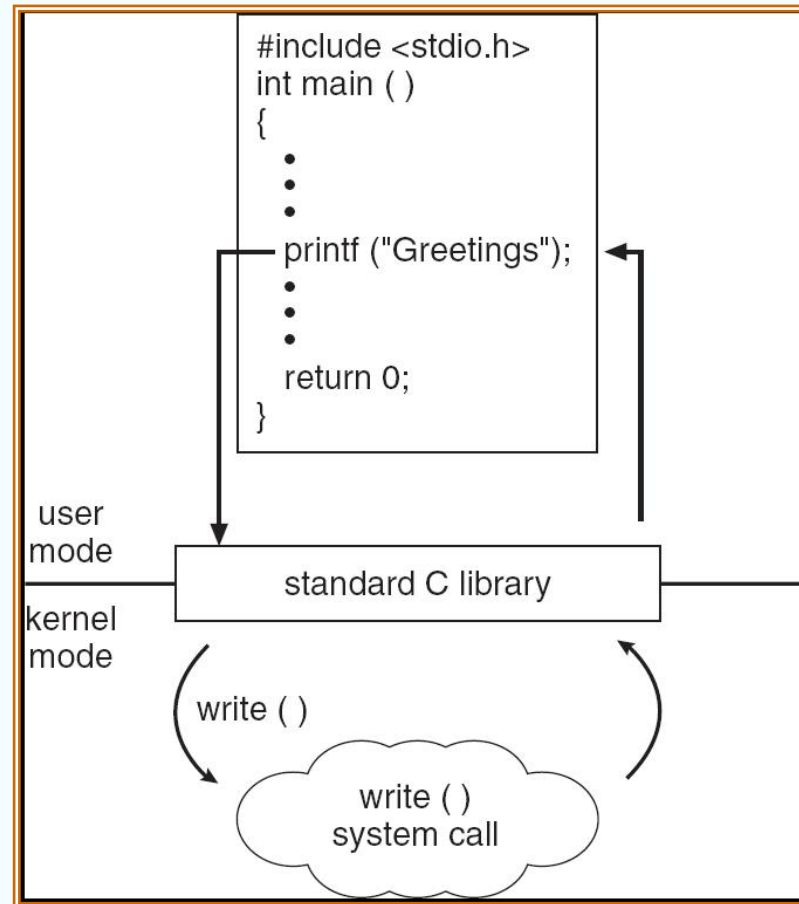
- Typically, a **number** associated with each system call
 - System-call interface maintains a **table** indexed according to these numbers
- The system call interface invokes intended system call in OS kernel and returns status of the system call and any return values
- The caller needs to know nothing about how the system call is implemented
 - Just needs to obey API and understand what OS will do as a result call
 - Most **details** of OS interface hidden from programmer by API
 - ▶ Managed by run-time support library (set of functions built into libraries included with compiler)

API – System Call – OS Relationship



Standard C Library Example

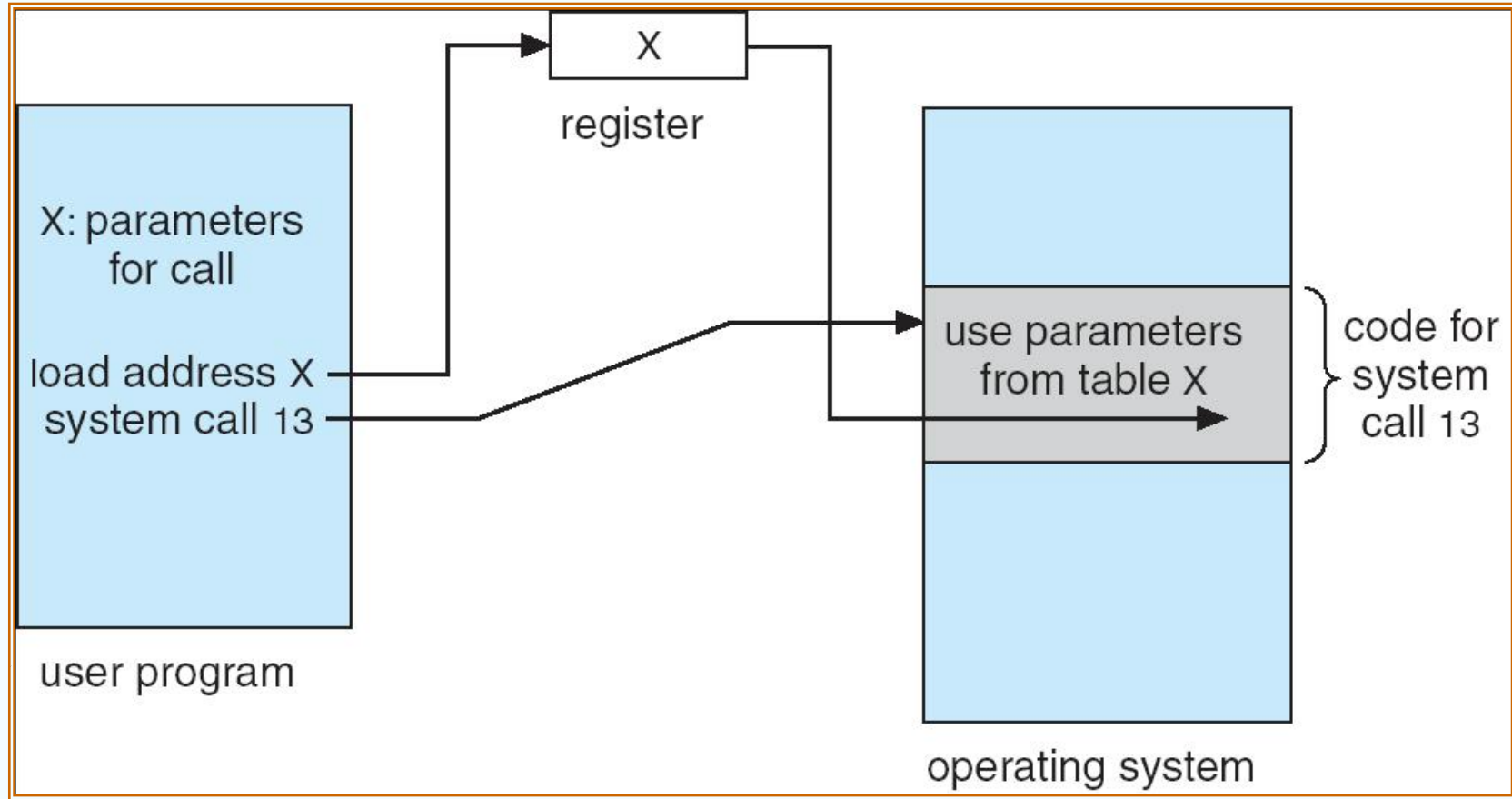
- C program invoking printf() library call, which calls write() system call



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
 - Exact type and amount of information vary according to OS and call
- 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



Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

Types of System Calls

- Process control
- File management
- Device management
- Information maintenance (e.g. time, date)
- Communications
- Protection

	Windows	Unix
Process control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File management	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device management	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communications	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shm_open() mmap()
Protection	SetFileSecurity() InitializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()

Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- **System Programs**
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

System Programs

- System programs provide a convenient environment for program development and execution. They can be divided into:
 - File manipulation
 - Status information
 - File modification
 - Programming language support
 - Program loading and execution
 - Communications
 - Application programs
- Most users' view of the operating system is defined by system programs, not the actual system calls

System Programs

- Provide a **convenient environment** for program development and execution
 - Some of them are simply user interfaces to system calls; others are considerably more complex
- **File management** - Create, delete, copy, rename, print, dump, list, and generally manipulate files and directories
- **Status information**
 - Some ask the system for info - date, time, amount of available memory, disk space, number of users
 - Others provide detailed performance, logging, and debugging information
 - Typically, these programs format and print the output to the terminal or other output devices
 - Some systems implement a registry - used to store and retrieve configuration information

System Programs (cont'd)

- **File modification**
 - Text editors to create and modify files
 - Special commands to search contents of files or perform transformations of the text
- **Programming-language support** - Compilers, assemblers, debuggers and interpreters sometimes provided
- **Program loading and execution**- Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Communications** - Provide the mechanism for creating virtual connections among processes, users, and computer systems
 - Allow users to send messages to one another's screens, browse web pages, send electronic-mail messages, log in remotely, transfer files from one machine to another

Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

Operating System Design and Implementation

- Design and Implementation of OS not “solvable”, but some approaches have proven successful
- Internal structure of different Operating Systems can vary widely
- Start by defining goals and specifications
- Affected by choice of hardware, type of system
- *User* goals and *System* goals
 - User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast
 - System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient



Creative
process

Operating System Design and Implementation (Cont.)

- Important principle to separate (2.6.2)

Policy: What will be done? 策略（确定具体做什么事）

Mechanism: How to do it? 机制（定义做事方式）

Example: timer for CPU protection is a mechanism

- 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: Separation of Mechanism and Policy

- An example of mechanism/policy separation is the use of **card-keys** to gain access to locked doors.
- *The mechanisms* (magnetic card readers, remote controlled locks, connections to a security server)
- *The entrance policy* (which people should be allowed to enter which doors, at which times).
- These decisions are made by a centralized security server, which (in turn) probably makes its decisions by consulting a database of room access rules.

Another Example

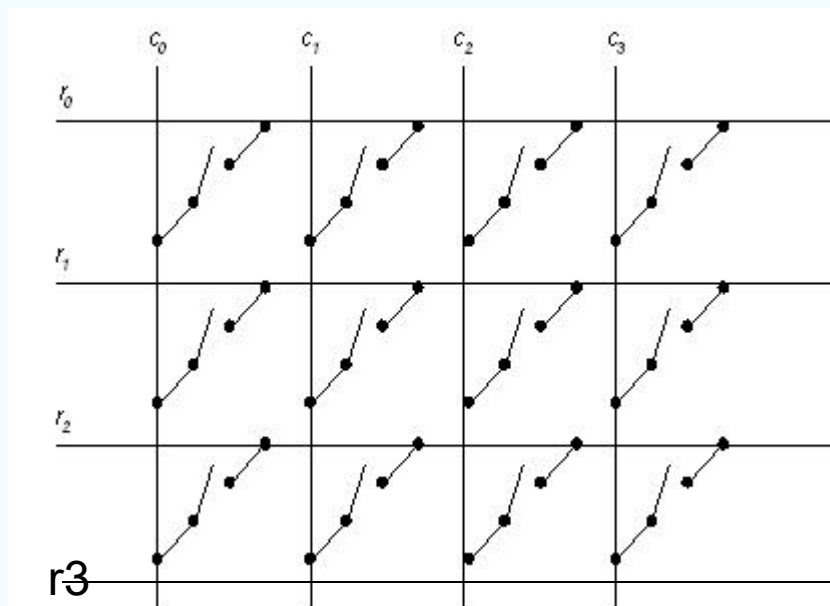
- The matrix data determines a policy, and the switching network is a mechanism

■ 1,0,0,0

■ 0,0,1,0

■ 0,1,0,0

■ 0,0,0,1



A 4x4 switching
matrix

Additional readings: The early principles of X

A very good illustration of the separation: X-Window System

- **Do not add new functionality unless an implementor cannot complete a real application without it.**
- **It is as important to decide what a system is not as to decide what it is. Do not serve all the world's needs; rather, make the system extensible so that additional needs can be met in an upwardly compatible fashion.**
- **The only thing worse than generalizing from one example is generalizing from no examples at all.**
- **If a problem is not completely understood, it is probably best to provide no solution at all.**
- **If you can get 90 percent of the desired effect for 10 percent of the work, use the simpler solution.**
- **Isolate complexity as much as possible.**
- **Provide mechanism rather than policy. In particular, place user interface policy in the clients' hands.**

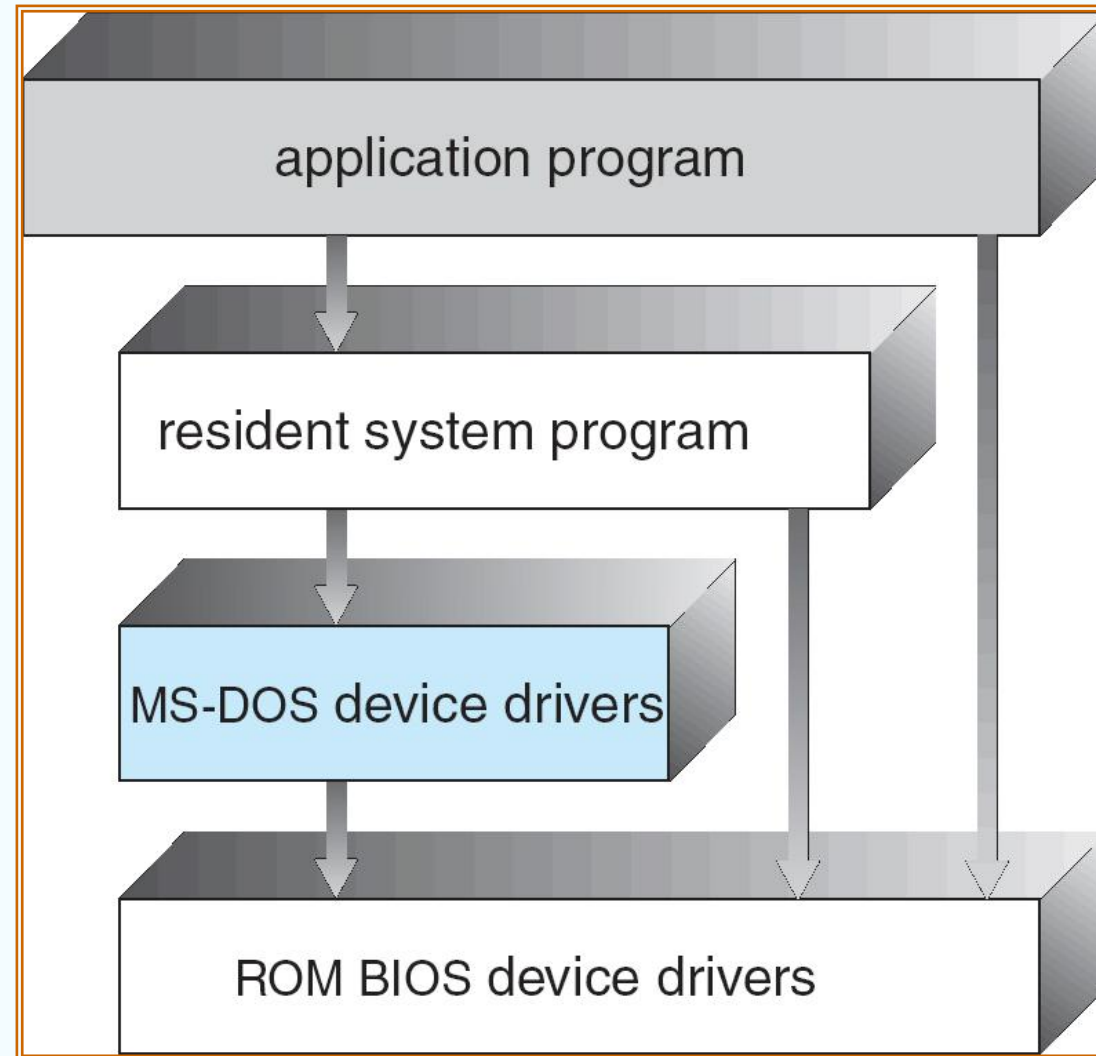
Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

Simple Structure

- MS-DOS – written to provide the most functionality in the least space
 - Not divided into modules
 - Although MS-DOS has some structure, its interfaces and levels of functionality are not well separated

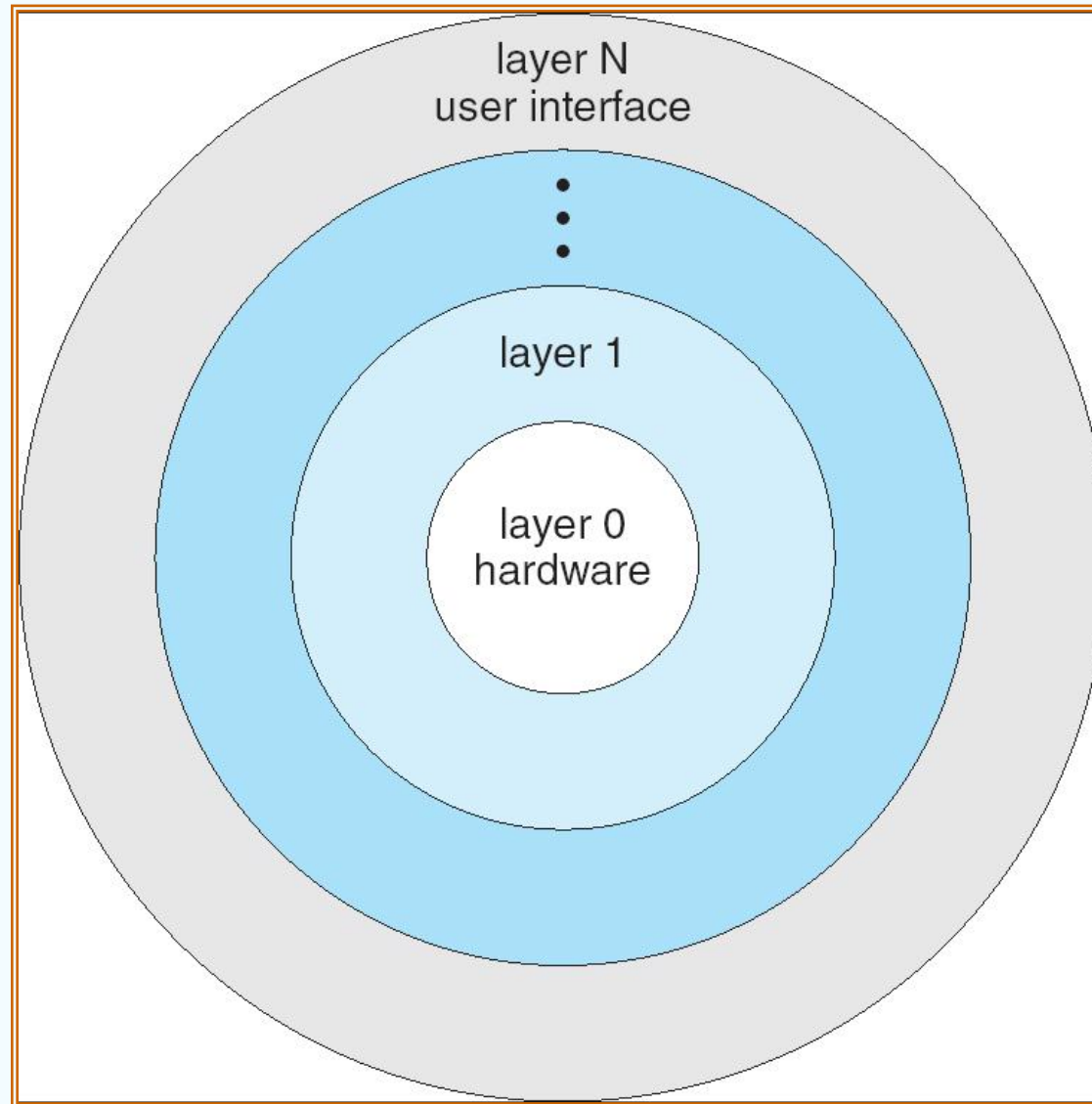
MS-DOS Layer Structure



Layered Approach

- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers

Layered Operating System

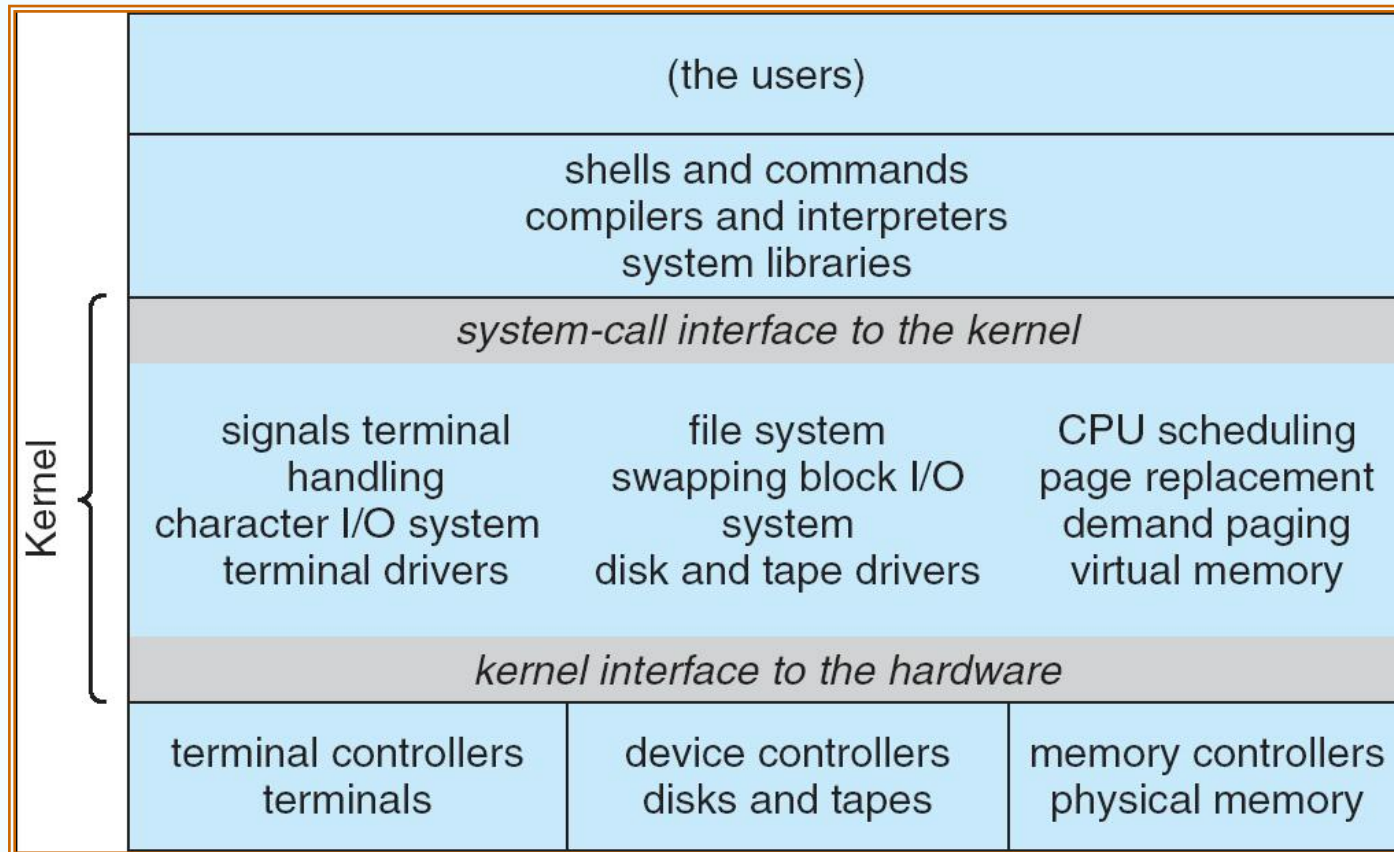


UNIX

- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring. The UNIX OS consists of two separable parts
 - System programs
 - The kernel
 - ▶ Consists of everything below the system-call interface and above the physical hardware
 - ▶ Provides the file system, CPU scheduling, memory management, and other operating-system functions; a large number of functions for one level

Monolithic structure!

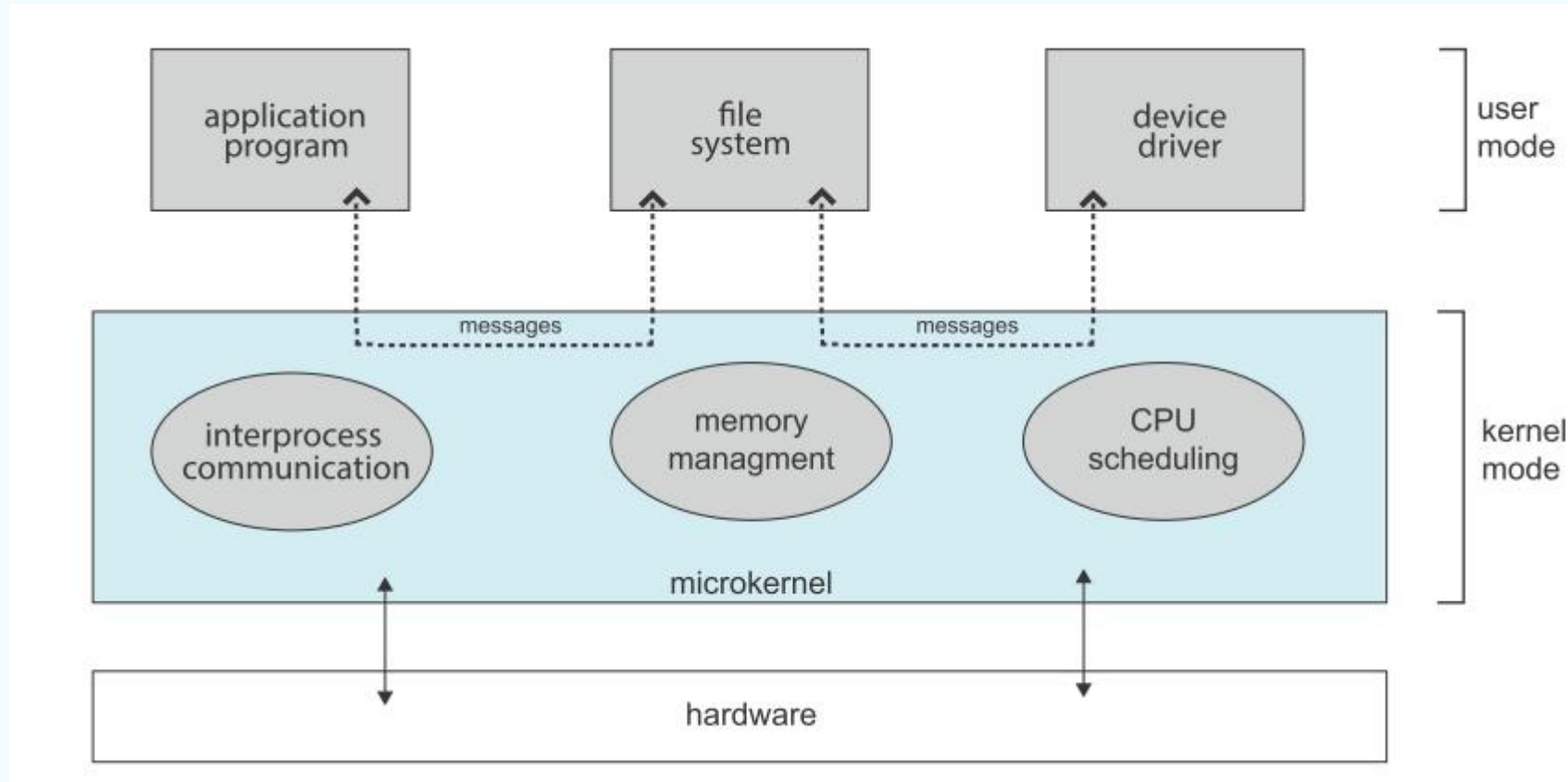
UNIX System Structure



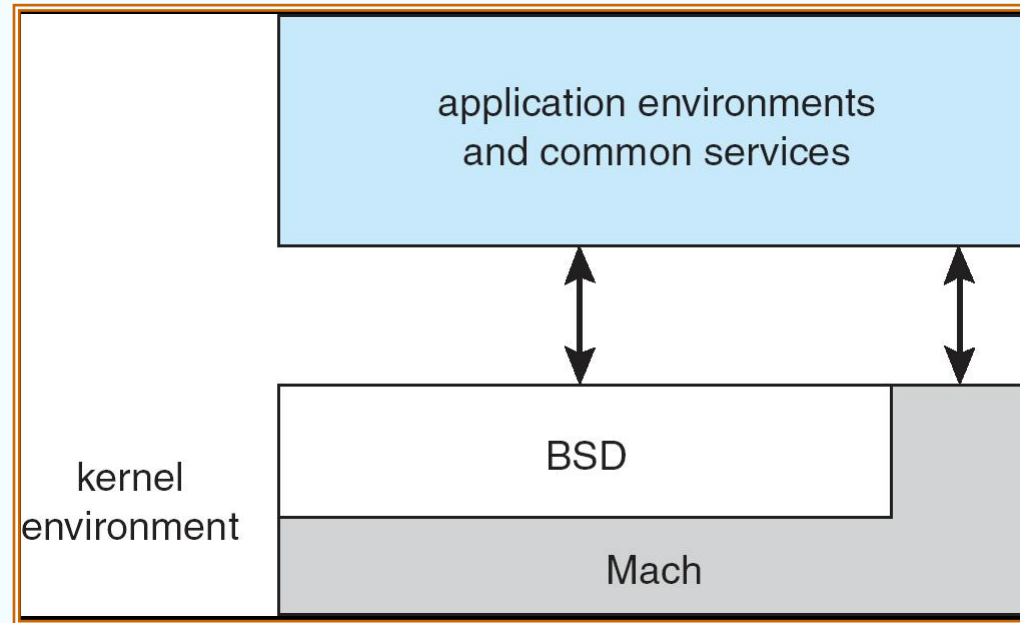
Microkernel System Structure

- Moves as much from the kernel into “*user*” space
- Communication takes place between *user modules* using *message passing*
- Benefits:
 - Easier to *extend* a microkernel
 - Easier to *port* the operating system to new architectures
 - More *reliable* (less code is running in kernel mode)
 - More *secure*
- Detriments:
 - *Performance overhead* of user space to kernel space communication

Architecture of A Typical Microkernel

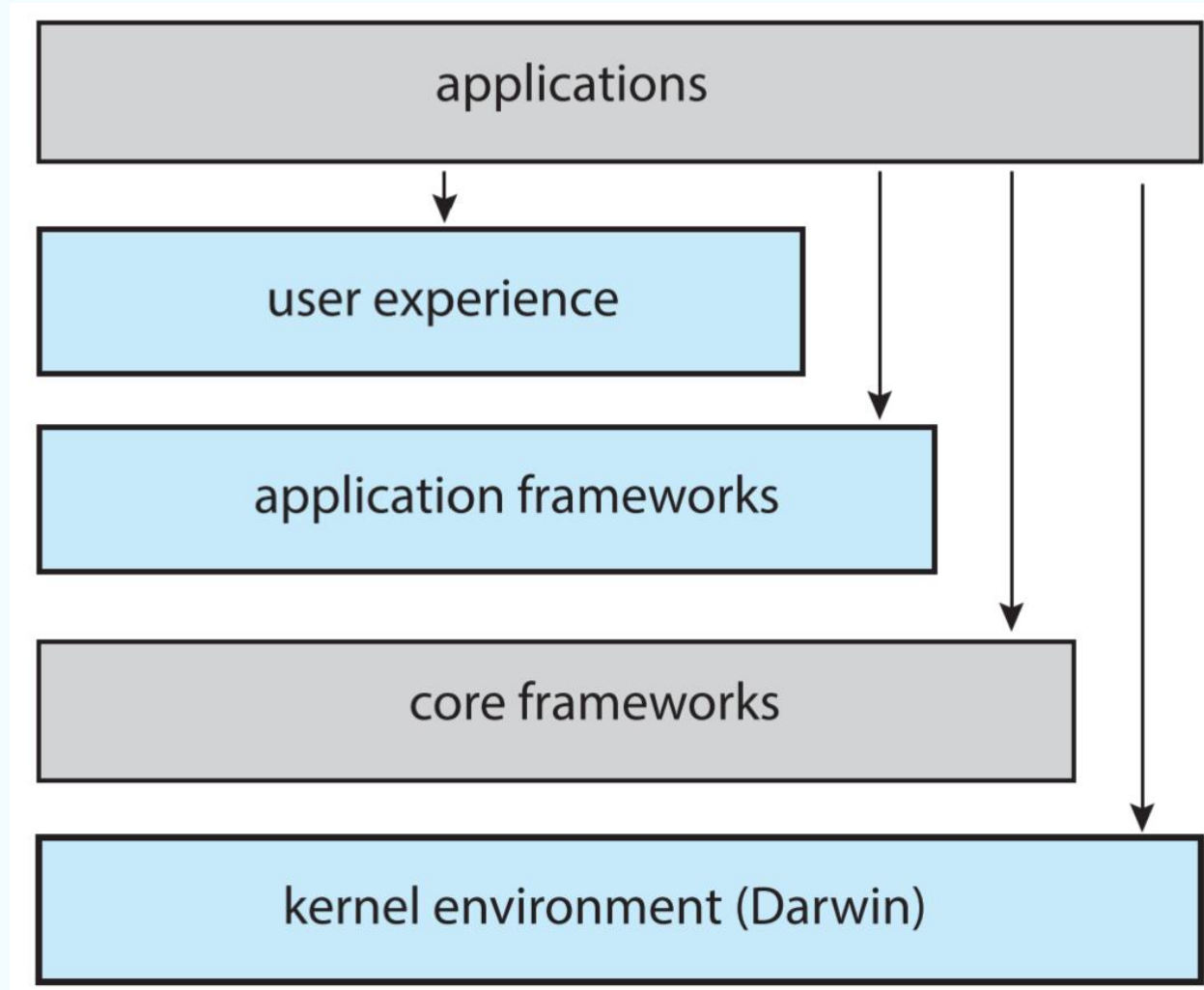


Mac OS X Structure



A hybrid structure!

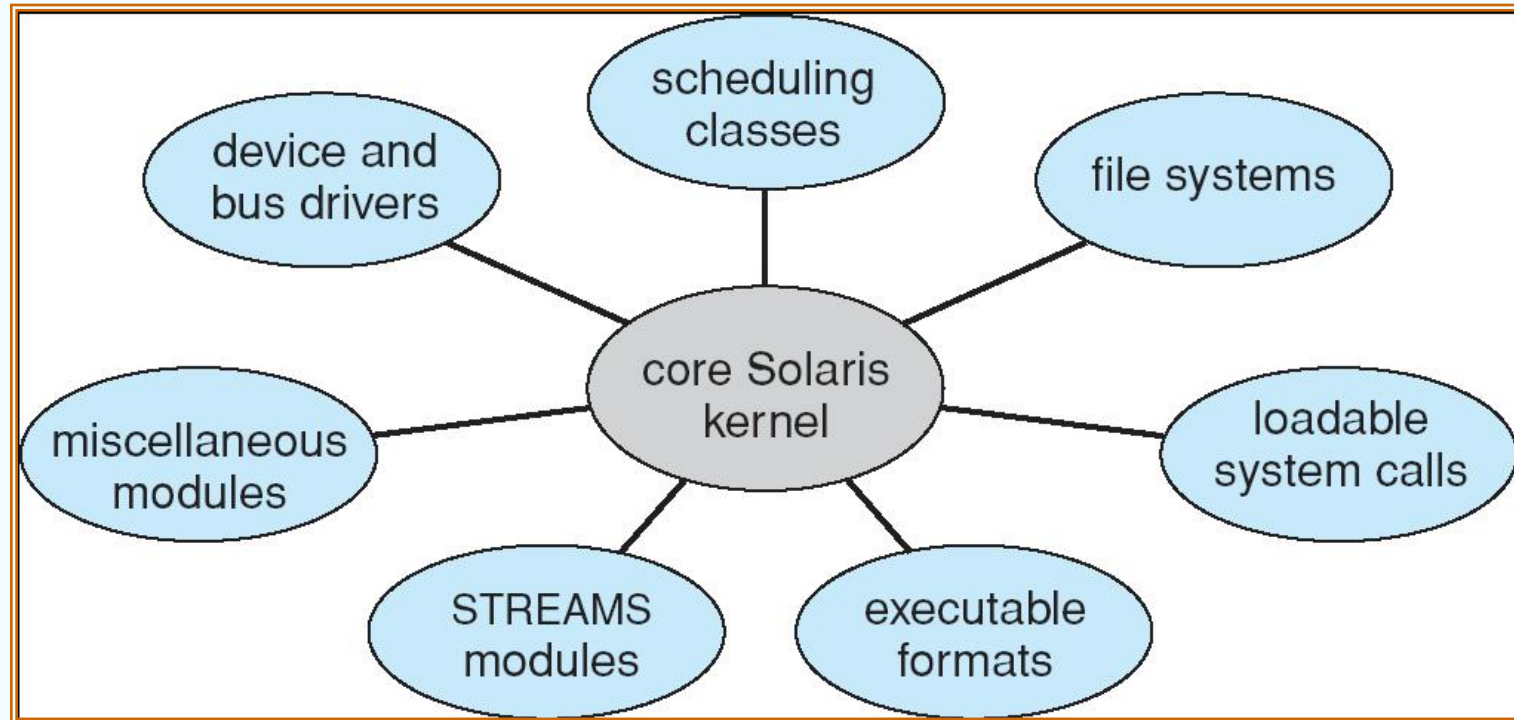
MacOS Structure



Modules

- Most modern operating systems implement **kernel modules**
 - Uses object-oriented approach
 - Each core component is separate
 - Each talks to the others over known interfaces
 - ▶ Modules call each other instead of message passing
 - Each is **loadable** as needed within the kernel
- Overall, similar to layers but more flexible

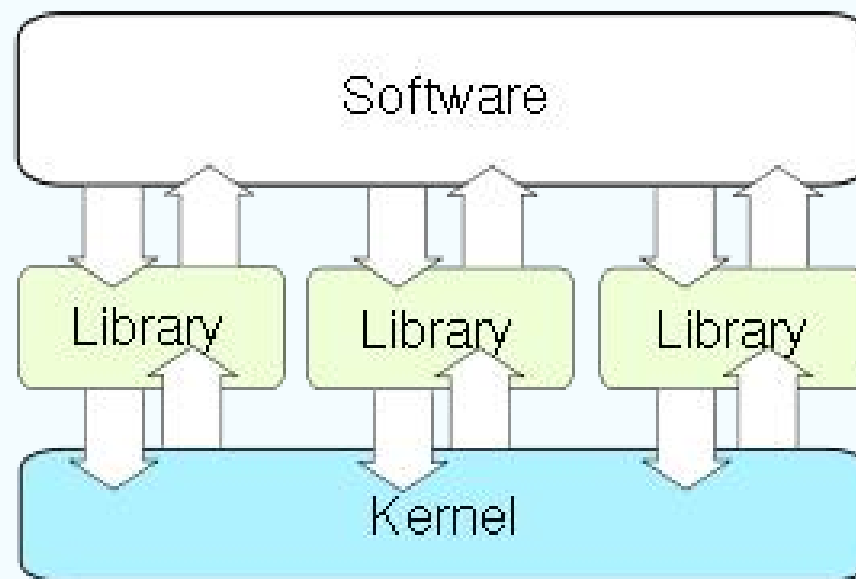
Solaris Modular Approach



Modular kernel

Other Structures

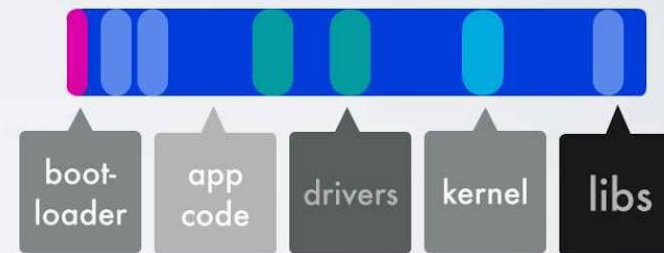
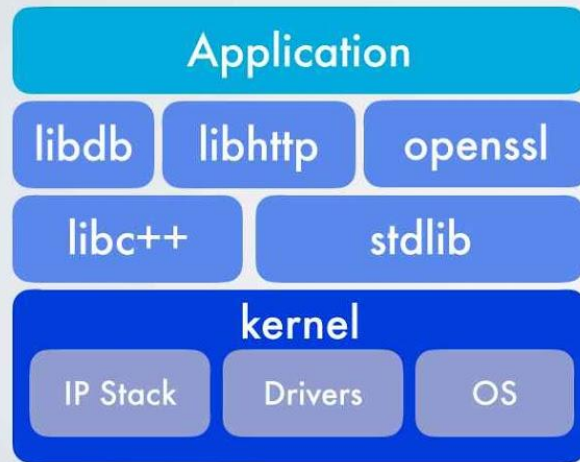
- Exokernel (1994), 高度简化kernel, 只负责资源分配, 提供了低级的硬件操作, 必须通过定制library供应用使用
- 高性能, 但定制化library难度大, 兼容性差



Other Structures

- Unikernel: statically linked with the OS code needed.
- Good for cloud service, APP boots in tens of milliseconds.

Traditional vs unikernel application



Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- **Virtual Machines**
- Operating System Generation
- System Boot

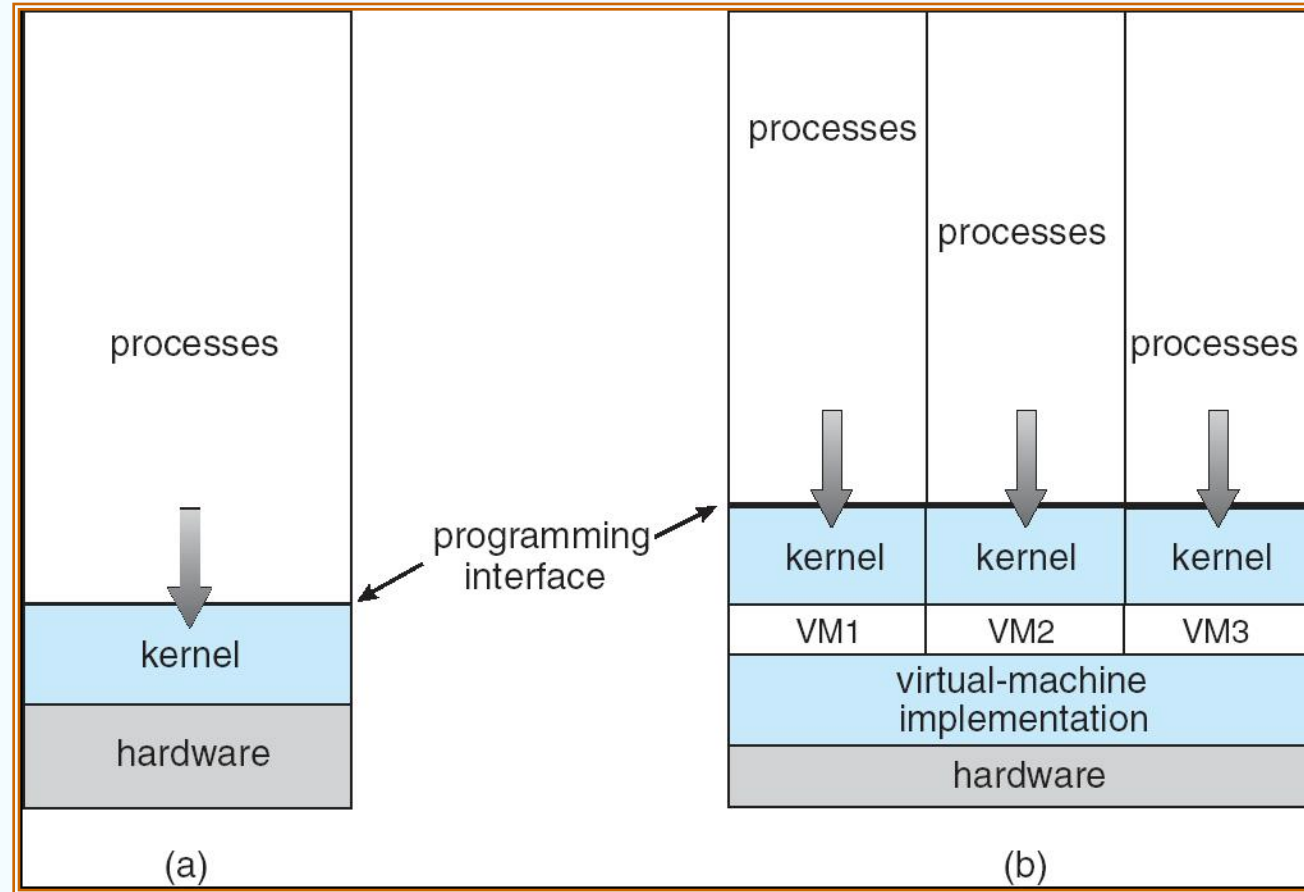
Virtual Machines

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all **hardware**
- A virtual machine provides an interface *identical* to the underlying bare hardware
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory

Virtual Machines (Cont.)

- The resources of the physical computer are shared to create the virtual machines
 - CPU scheduling can create the appearance that users have their own processor
 - Spooling and a file system can provide virtual card readers and virtual line printers
 - A normal user time-sharing terminal serves as the virtual machine operator's console

Virtual Machines (Cont.)



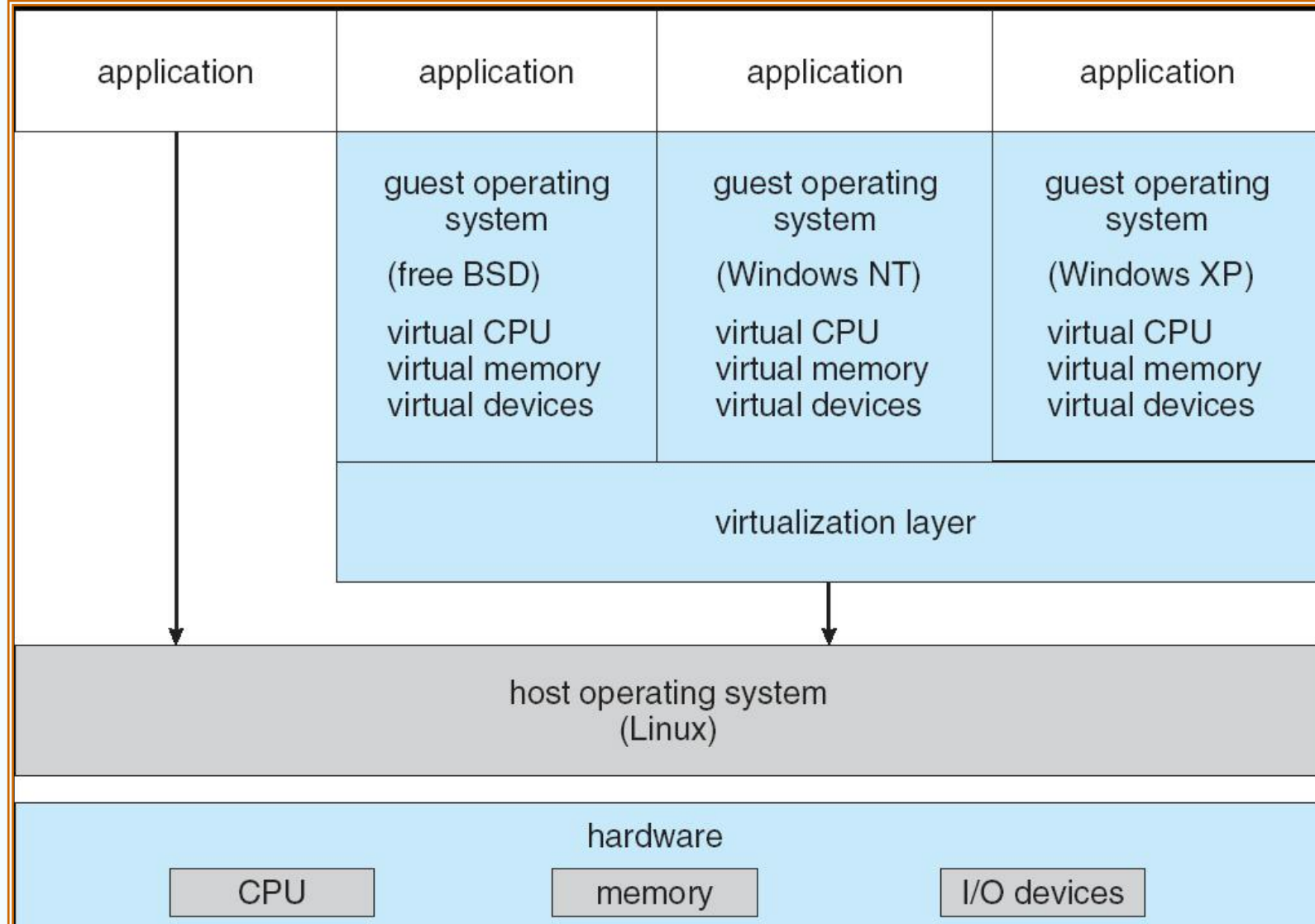
(a) Nonvirtual machine

(b) virtual machine

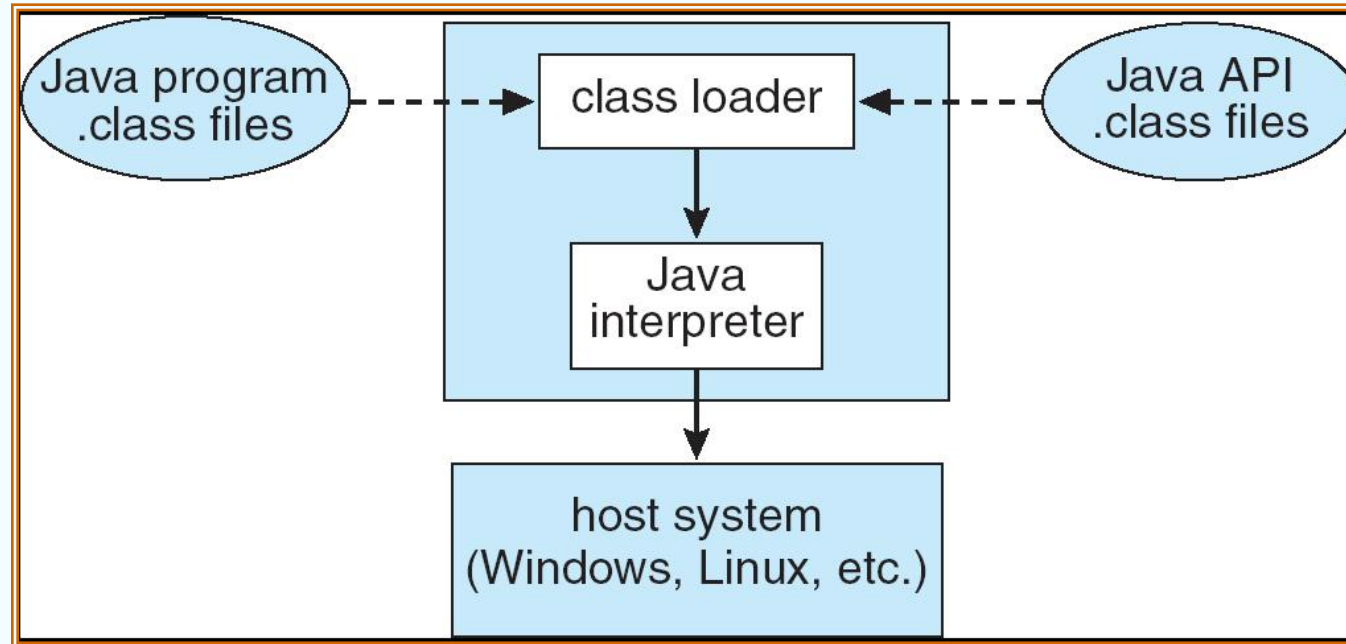
Virtual Machines (Cont.)

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. **System development** is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.
- The virtual machine concept is difficult to implement due to the effort required to provide an **exact** duplicate to the underlying machine. (for example, virtual user mode and kernel mode)

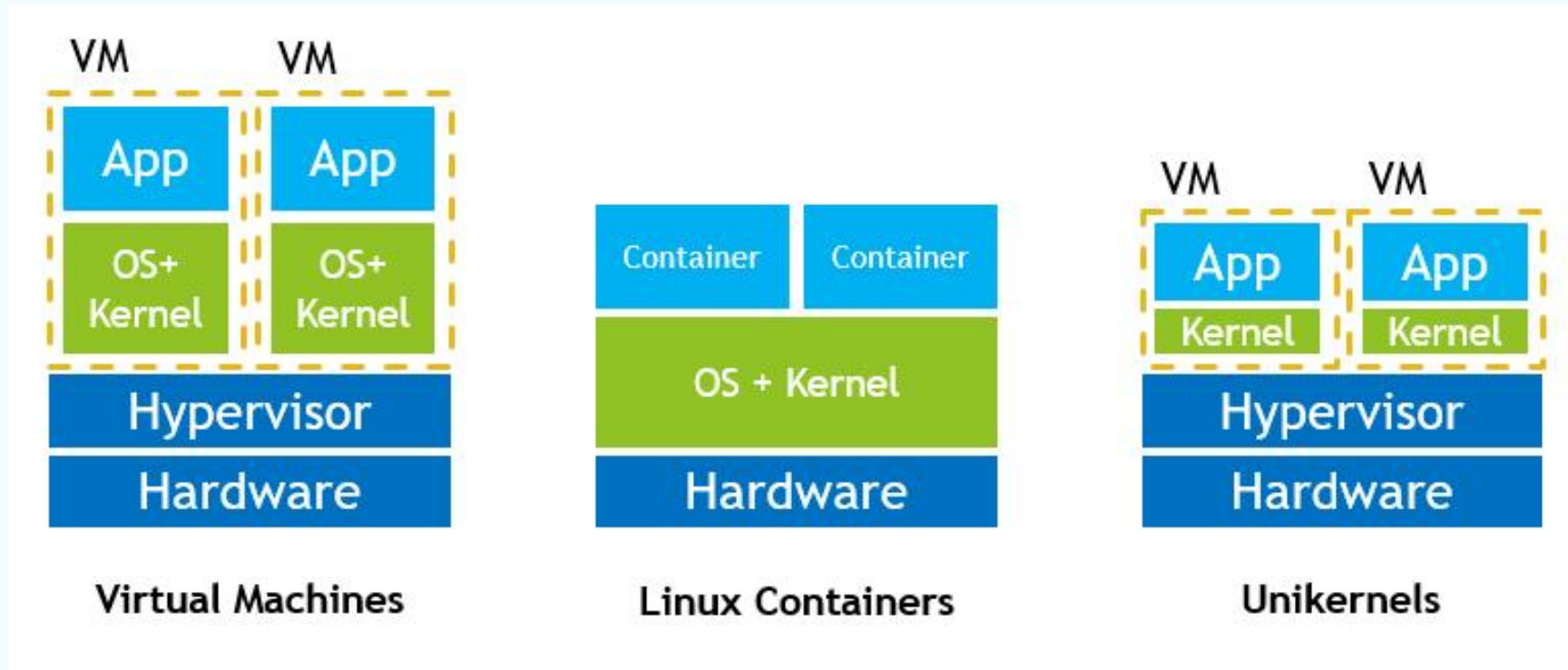
VMware Architecture



The Java Virtual Machine



Different Techniques



Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- System Boot

Operating System Generation

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site
- SYSGEN program obtains information concerning the specific configuration of the hardware system
- *Booting* – starting a computer by loading the kernel
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution

Chapter 2: Operating-System Structures

- Operating System Services
- User Operating System Interface
- System Calls
- Types of System Calls
- System Programs
- Operating System Design and Implementation
- Operating System Structure
- Virtual Machines
- Operating System Generation
- **System Boot**

System Boot

- Operating system must be made available to hardware so hardware can start it
 - Small piece of code – **bootstrap program (a.k.a. bootstrap loader)**, locates the kernel, loads it into memory, and starts it
 - Sometimes two-step process where **boot block** at fixed location loads bootstrap loader
 - When power initialized on system, execution starts at a fixed memory location
 - ▶ Firmware used to hold initial boot code

End of Chapter 2

Lab1

- Risc-V kernel startup process

Part of the startup process

- Xv6 book, page 27, section 2.6:
- “The function `start` performs some configuration that is only allowed in machine mode, and then switches to supervisor mode. To enter supervisor mode, RISC-V provides the instruction `mret`. This instruction is most often used to return from a previous call from supervisor mode to machine mode. `start` isn’t returning from such a call, and instead sets things up as if there had been one: it sets the previous privilege mode to supervisor in the register `mstatus`, it sets the return address to `main` by writing `main`’s address into the register `mepc`, disables virtual address translation in supervisor mode by writing 0 into the page-table register `satp`, and delegates all interrupts and exceptions to supervisor mode.
- Before jumping into supervisor mode, `start` performs one more task: it programs the clock chip to generate timer interrupts. With this housekeeping out of the way, `start` “returns” to supervisor mode by calling `mret`. This causes the program counter to change to `main`.”