

Operating-System Functions and Structures

Shengzhong Liu

Department of Computer Science and Engineering

Shanghai Jiao Tong University

Resources

□ Course Webpage:

□ Canvas: <https://oc.sjtu.edu.cn/courses/75481>

- ▶ Make sure you can access the course webpage.

□ WeChat Group:

□ **CS2310/CS2303-2025Spring**



□ Textbooks:

□ Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne, “**Operating System Concepts**”, 10th Edition, John Wiley & Sons, Inc.

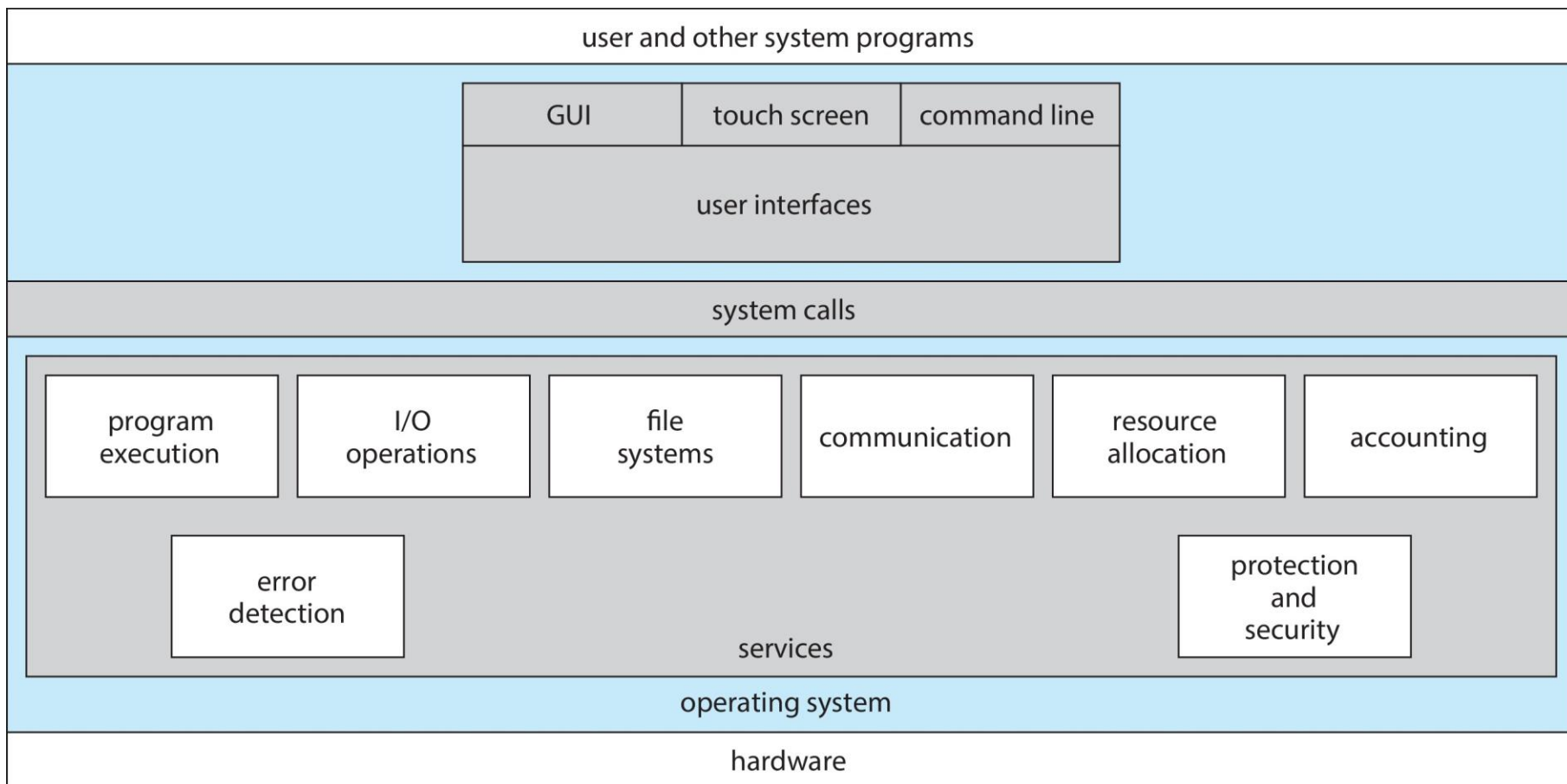
- ▶ 吴帆、刘功申、吴晨涛, 《**操作系统原理与实现**》, 人民邮电出版社, 2024.06.



Outline

- Operating System Functions
 - User Interfaces
 - System Calls
 - System Services
- Operating System Structures
 - Extension: Virtual Machine

Operating System Functions Overview

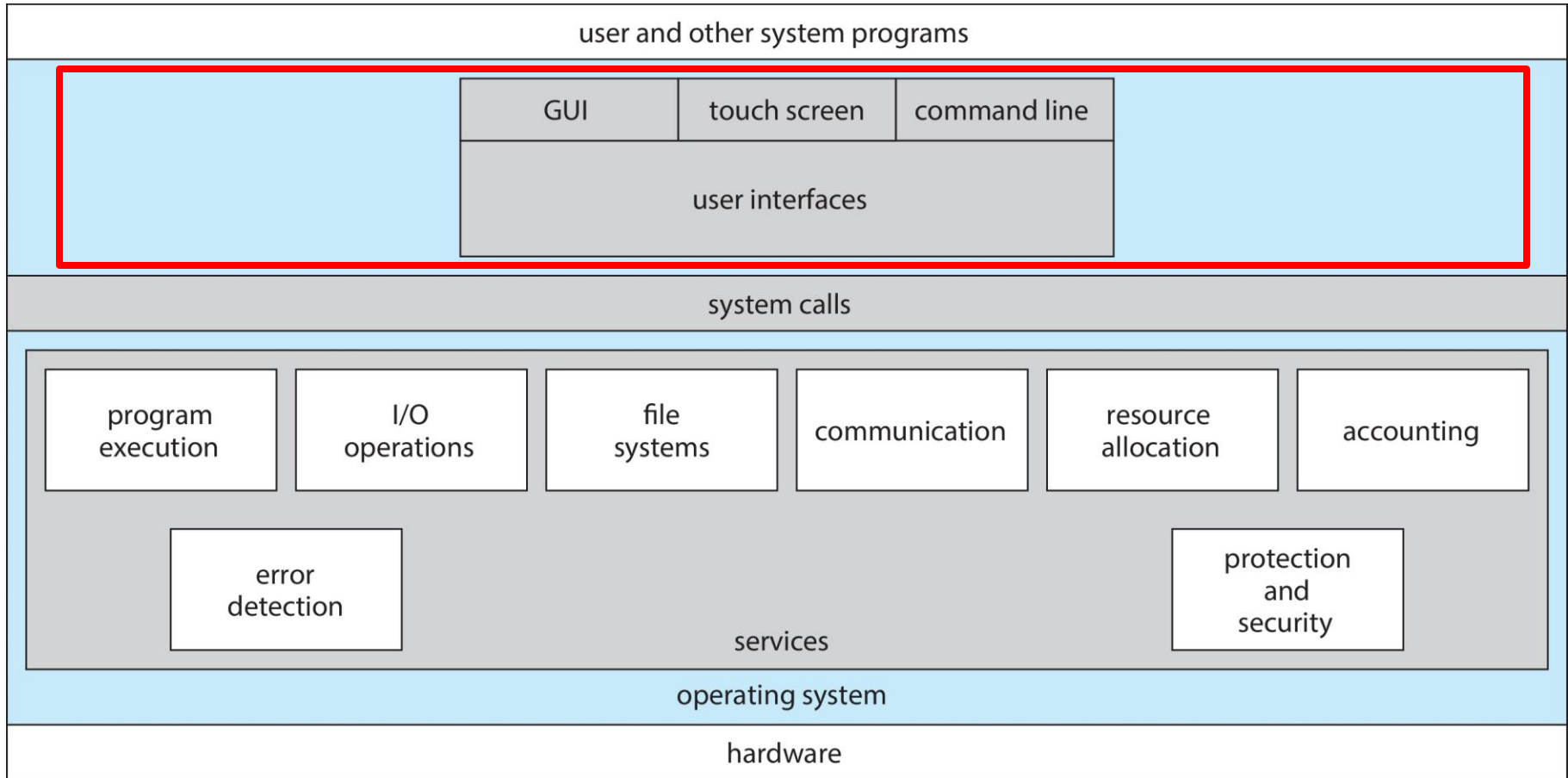


Operating systems provide an environment for
program executions and **services to programs/users**

Operating System Functions:

1. User Interface

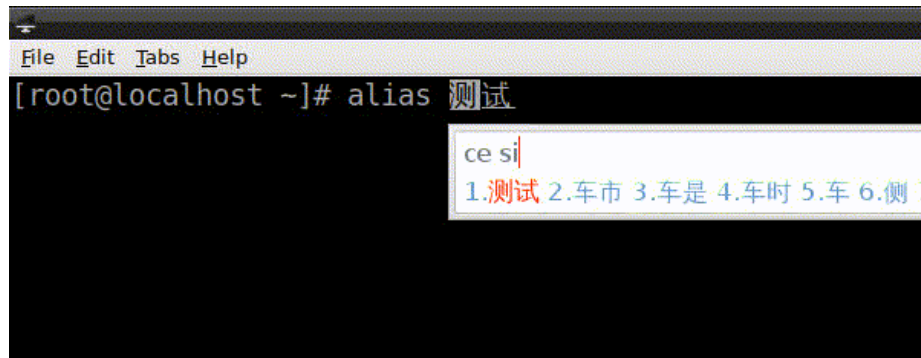
OS Functions: 1. User Interface



Shell: A computer program that exposes an OS's services to a human user or other programs. OS shells use either a **command-line interface (CLI)** or a **graphical user interface (GUI)**

User Interface - Command Line Interpreter

- Command-line interface (CLI):
 - A shell that uses alphanumeric characters (字母与数字) to provide instructions and data to the OS interactively.
 - It fetches a command from the user and executes it
 - OS can have multiple shell programs with different commands and syntax. Linux CLI shell types **sh**, **bash**, **zsh**, **csch**, **ksh**, ...
- Shell commands can be implemented in two ways:
 - **Built-in commands**: The interpreter contains the code to execute the command.
 - **System program commands**: The command is a program name.
 - ▶ Adding such commands doesn't require changing the shell.

A terminal window with a menu bar (File, Edit, Tabs, Help) and a prompt [root@localhost ~]#. The user enters the command 'alias 测试'. A dropdown menu appears with the text 'ce si' and a list of suggestions: '1.测试 2.车市 3.车是 4.车时 5.车 6.侧 7'.

```
File Edit Tabs Help
[root@localhost ~]# alias 测试
ce si
1.测试 2.车市 3.车是 4.车时 5.车 6.侧 7
```

→ rm file.txt

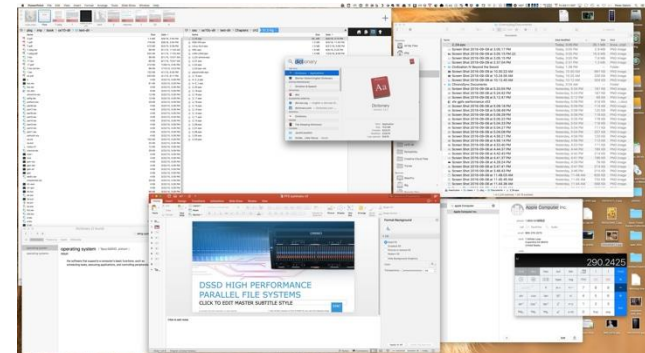
User Interface - GUI

- User-friendly **desktop metaphor** (比喻) 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
- Many systems now include both CLI and GUI interfaces
 - Windows: GUI with CLI “command” shell
 - Mac OS X: an “Aqua” GUI interface with UNIX kernel underneath and shells available
 - UNIX and Linux: have CLI with optional GUI interfaces (CDE, KDE, GNOME)



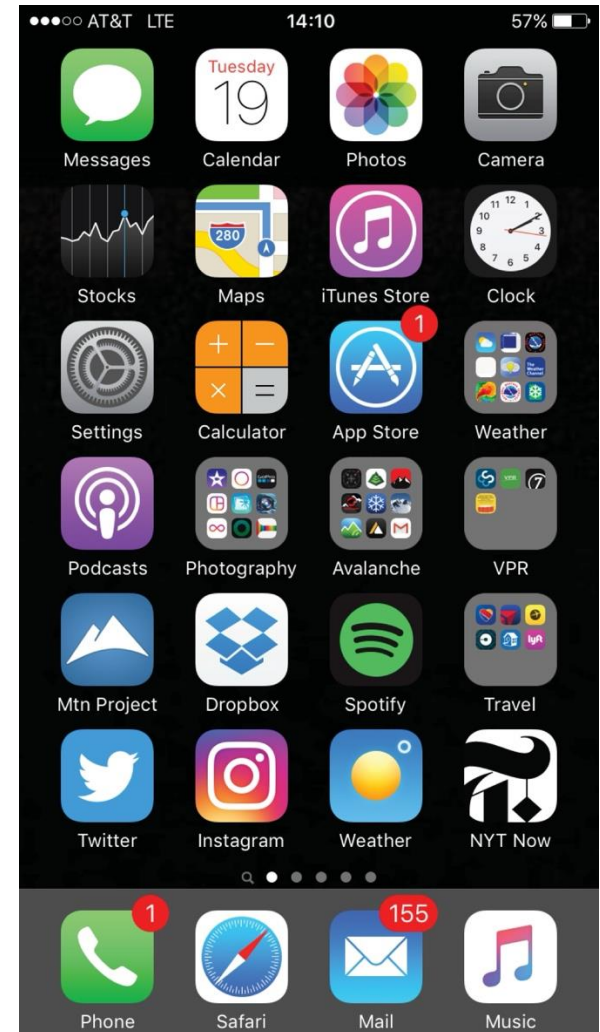
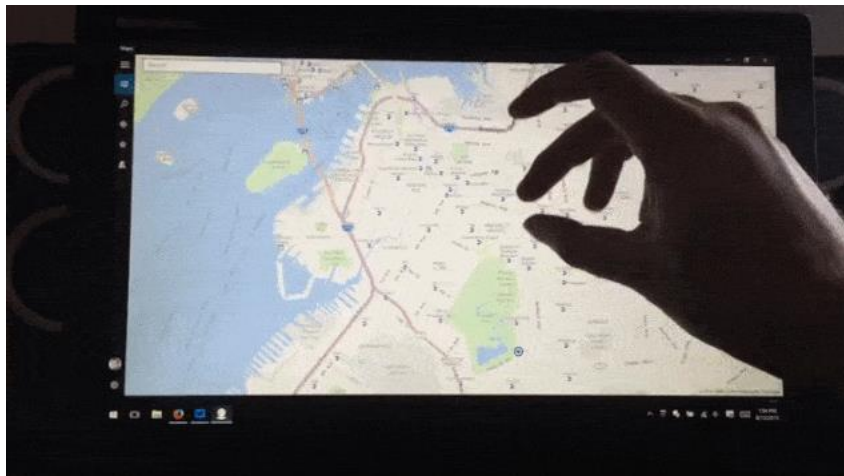
Desktop Metaphor

“Open a computer file with GUI is like physically opening a file on your desktop”



Touchscreen Interfaces

- ❑ Touchscreen devices require new interfaces
 - ❑ Mouse not possible or not desired
 - ❑ Actions and selection based on gestures
 - ❑ Virtual keyboard for text entry
- ❑ Voice commands



Future: AR Interface – Apple Vision Pro

Video Source: <https://www.bilibili.com/video/BV1pK421C7A9>

Future: Conversational Interface

这株白参 bilibili



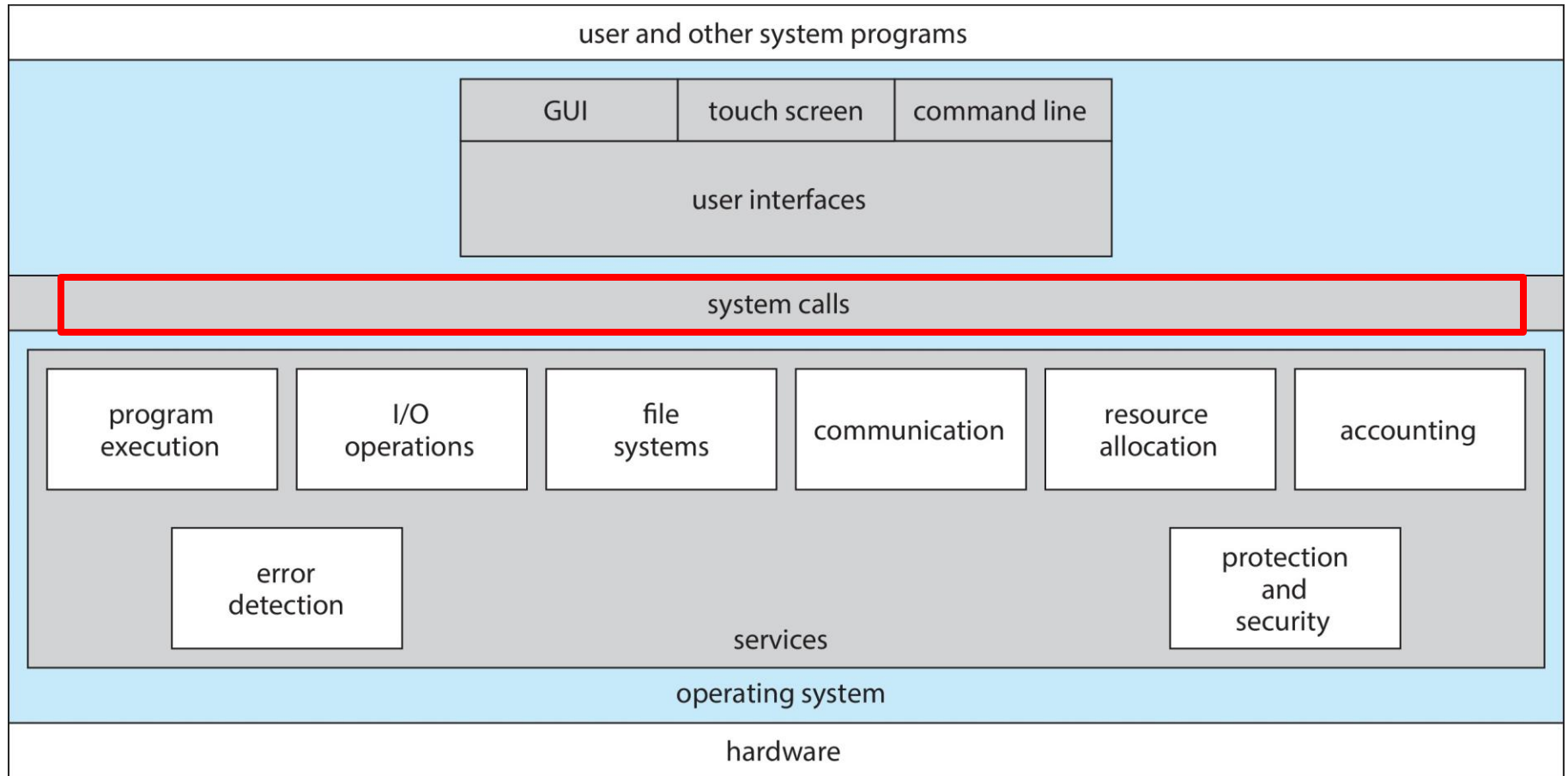
这株白参

Video Source: <https://www.bilibili.com/video/BV1xV411C7k7>

Operating System Functions :

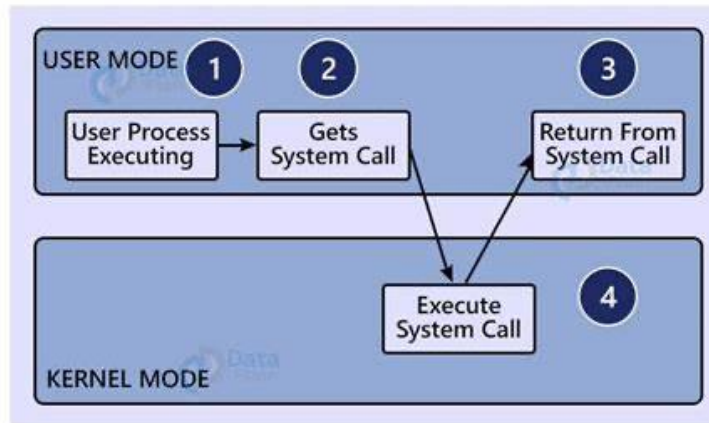
2. System Calls

OS Functions: 2. System Calls



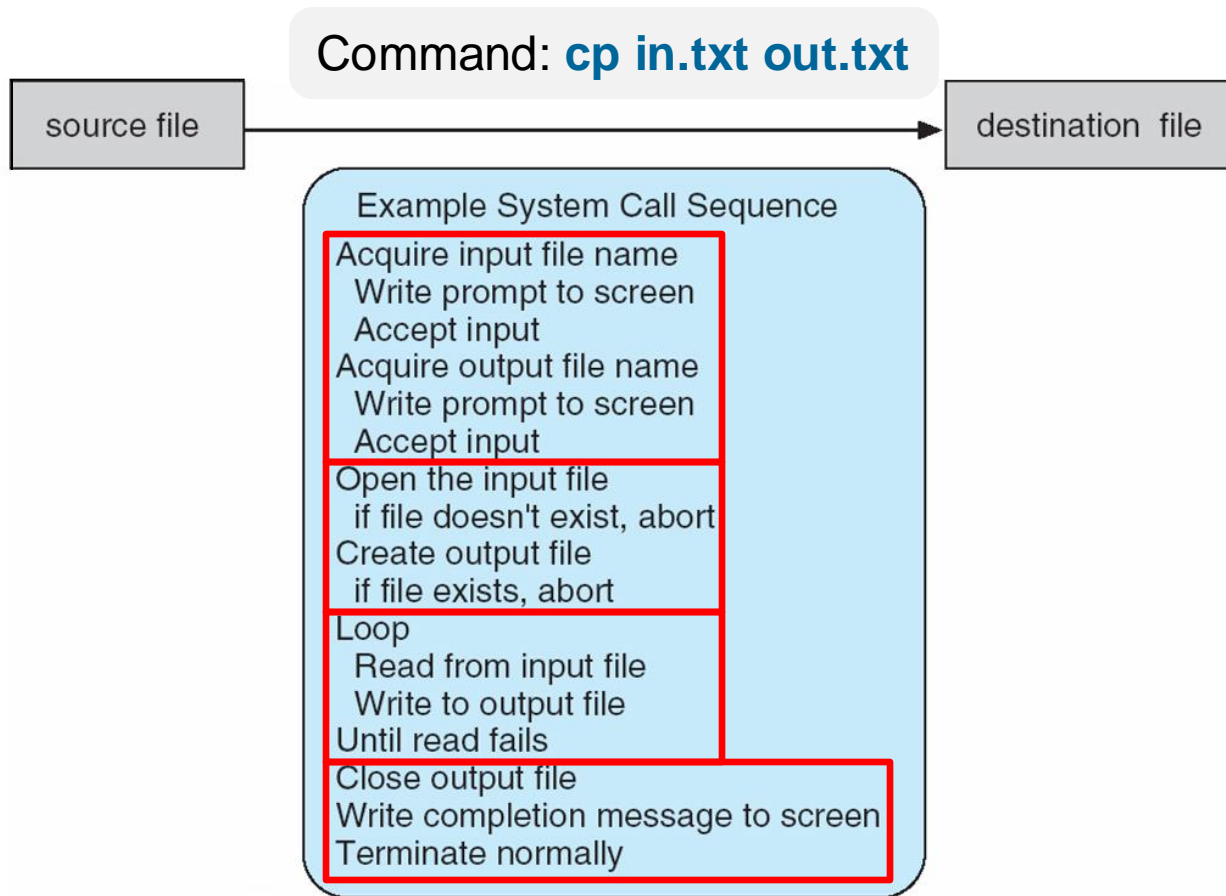
System Calls

- ❑ System calls are programming interface to the services provided by the OS
- ❑ Typically written in a high-level language (C or C++)



Example of System Calls

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



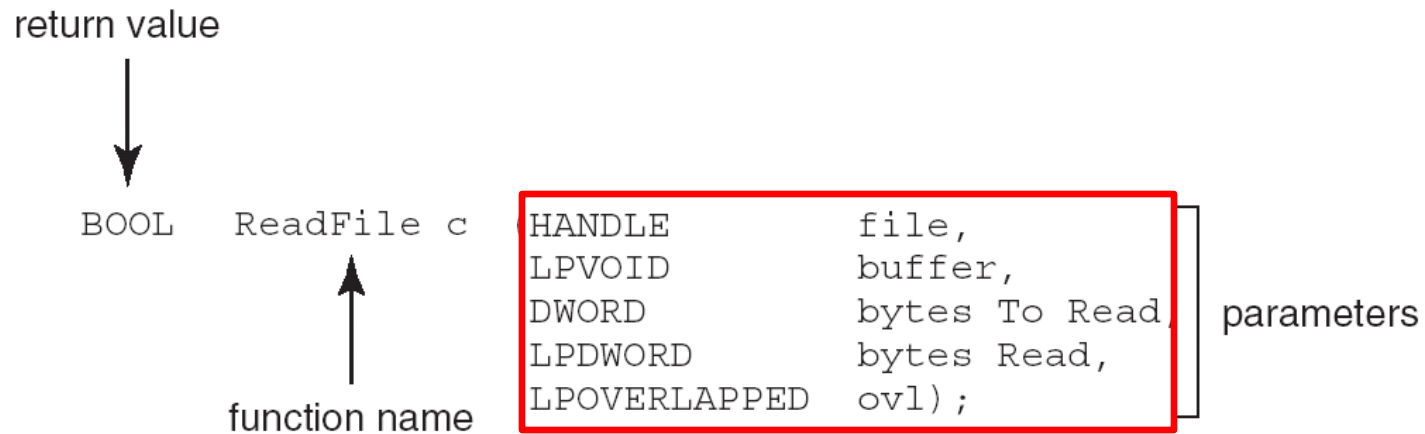
Application Program Interface (API)

- ❑ System calls are mostly accessed by programs via a high-level **Application Program Interface (API)** rather than direct system call use.
 - ❑ More abstract to programmers with details hidden.
- ❑ Three most common APIs:
 - ❑ Win32 API for Windows
 - ❑ POSIX API for POSIX-based systems (UNIX, Linux, and macOS X)
 - ❑ Java API for the Java virtual machine (JVM)
- ❑ Why use APIs rather than system calls?

	System Calls	APIs
Ease of Use	Complex, requires low-level details	High-level, user-friendly
Portability	OS-specific (Linux, Windows differ)	Cross-platform compatibility
Security	Exposes kernel directly, risky	Provides controlled access

Example of Standard API

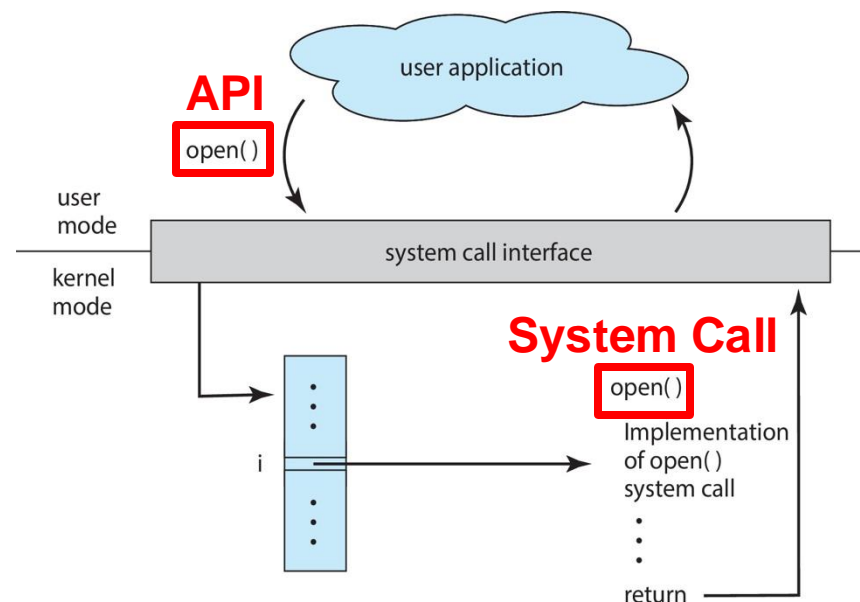
- Consider the **ReadFile()** function in the Win32 API
 - A function for reading from a file



- A description of the parameters passed to **ReadFile()**
 - **HANDLE** file—the file to be read
 - **LPVOID** buffer—a buffer where the data will be read into and written from
 - **DWORD** bytesToRead—the number of bytes to be read into the buffer
 - **LPDWORD** bytesRead—the number of bytes read during the last read
 - **LPOVERLAPPED** ovl—indicates if overlapped I/O is being used

System Call Implementation

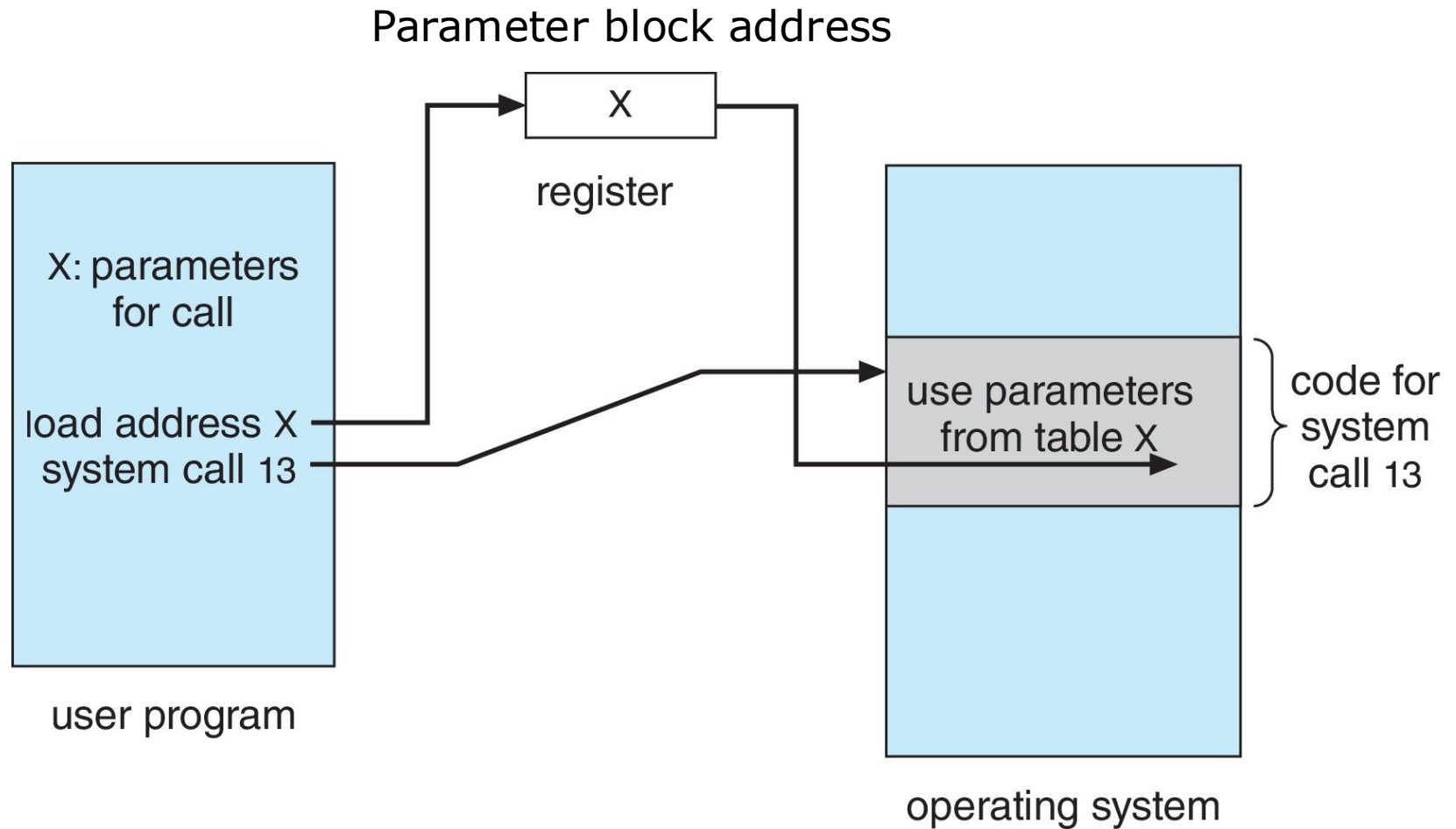
- Typically, an ID number is associated with each system call
 - **System-call interface** maintains a table indexed according to these numbers
- The system call interface:
 - Invokes the intended system call in OS kernel
 - Returns status of the system call and any return values
- The caller does not know how the system call is implemented
 - Just needs to obey API and understand what OS will do as a result of the call



System Call Parameter Passing

- Three general methods used to pass parameters to the OS
 - **Option 1:** Pass the parameters in registers
 - ▶ In some cases, there could be more parameters than registers
 - **Option 2:** Store the parameters in a block or table in memory, then pass the block address as a parameter in a register
 - ▶ This approach taken by Linux and Solaris
 - **Option 3:** Place or push the parameters onto the **stack (栈)** by the program, which can be popped off the stack by the operating system
 - ▶ Stack: Last-in first-out data structure
- The last two methods do not limit the number or length of parameters being passed

Example: Parameter Passing via Table



Types of System Calls (1)

□ Type 1: Process control

□ Control the current process:

- ▶ Finish current process: end, abort
- ▶ Run a different algorithm within current process: load, execute
- ▶ Wait for time, wait event, signal event
- ▶ Dump memory if error

□ Control a different process:

- ▶ create process, terminate process
- ▶ get process attributes, set process attributes

□ Allocate memory and release memory

□ **Debugger**: determining bugs, single-step execution

□ **Locks** for managing access to shared data between processes

Types of System Calls (2)

□ Type 2: File management

- create file, delete file
- open, close file
- read, write, re-position
- get and set file attributes

□ Type 3: Device management

- request device, release device
- read, write, re-position
- get device attributes, set device attributes
- logically attach or detach devices

Types of System Calls (3)

□ Type 4: Information maintenance

- get time or date, set time or date
- get system data, set system data
- get and set process, file, or device attributes

□ Type 5: 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 (4)

□ Type 6: Protection

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

Examples of Windows and UNIX System Calls

EXAMPLES OF WINDOWS AND UNIX SYSTEM CALLS

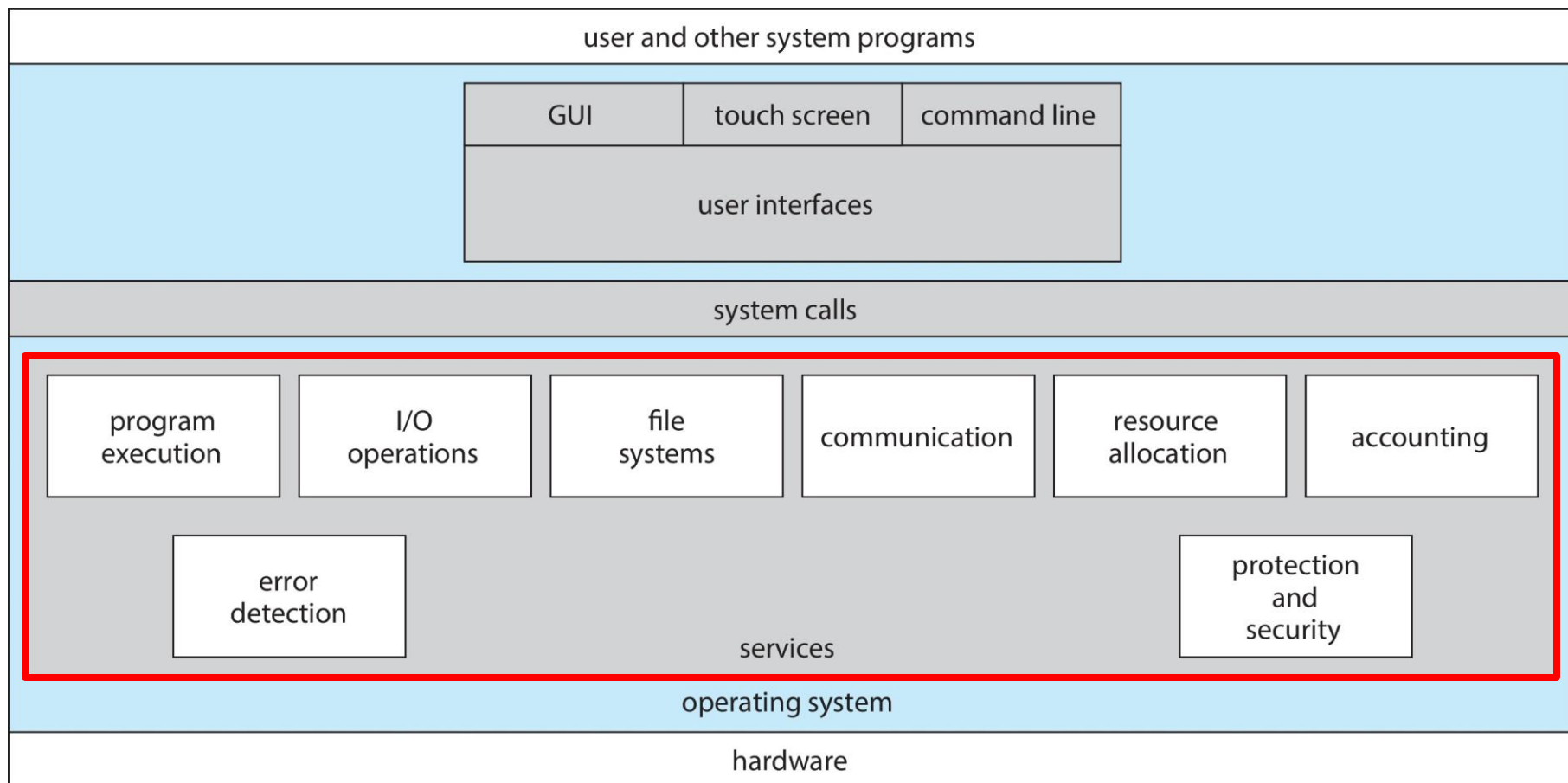
The following illustrates various equivalent system calls for Windows and UNIX operating systems.

	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()

Operating System Functions :

3. System Services

OS Functions: 3. System Services



System Services

- **System services** provide a convenient environment for program development and execution.
 - In computer hierarchy, **system services are higher than system calls**.
 - System services use system calls to interact with the OS kernel
- System services interact with both **system/application programs** and **kernel**

Component	Interacts With	Used Interface	Role
Application Programs	System Programs, System Services	✓ Uses APIs	Requests OS services (e.g., file access, networking) through APIs
System Programs	System Services, APIs	✓ Uses APIs	Provides system utilities for OS management (e.g., shells, file managers)
APIs	System Services, System Calls	✓ Uses System Calls	Provides a structured way for programs to request system resources
System Services	System Calls, Kernel	✓ Uses APIs and System Calls	Background services that manage files, networking, processes
System Calls	OS Kernel	✓ Direct Kernel Interface	Low-level interface to access hardware and system resources
Kernel	Hardware	✗ No APIs/System Calls	Manages system resources and controls hardware

System Programs vs. Application Programs

	System Programs	Application Programs
Definition	Programs that manages some aspect of the operating system or operating environment.	Programs that perform a particular function directly for the users.
Install	Comes installed with the OS and cannot be uninstalled	User-installable
Start and stop	Typically started by the system, and not as the result of a user interaction	Explicitly started by the user, and stops when the user exits the program
User Interface	May or may not	Yes
Examples	Login program, shell, window manager	Email, web browsers, gaming software, word processors

Why Applications are Operating System Specific

- **Observation**: Apps compiled on one system usually not executable on other operating systems
 - e.g., you can not install .exe applications on a Mac OS computer.
- **Reason**: Each operating system provides its own unique system calls
 - Own file formats, etc.
- **Solution**: Apps can be multi-operating system in 3 ways
 - **Option 1**: Written in interpreted language like Python, Ruby, and interpreter available on multiple operating systems
 - **Option 2**: App written in language that includes a VM containing the running app (like Java)
 - **Option 3**: Use standard language (like C), compile separately on each operating system to run on each

System Services (1)

□ 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 (注册表)** - used to store and retrieve configuration information

System Services (2)

□ 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

System Services (3)

□ Background Services

- Launch at boot time
 - ▶ Some for system startup, then terminate
 - ▶ Some from system boot to shutdown
- Provide utilities like disk checking, process scheduling, error logging, printing
- Run in user context not kernel context
- Known as **services**, **subsystems**, **daemons**

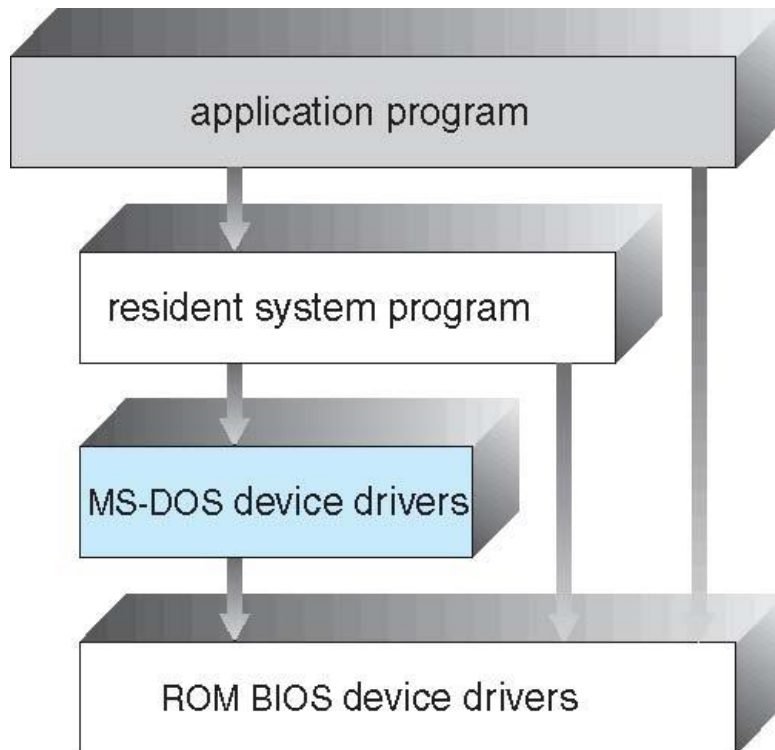
Operating System Structures

Operating System Structures

- General-purpose OS is very large program
- Various ways for OS structures:
 - Simple structure – MS-DOS
 - Monolithic structure – UNIX
 - Layered structure – an abstraction
 - Microkernel structure – Mach (微内核)
 - Hybrid structure – Windows, macOS, Android

Simple Structure – MS-DOS

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



Floppy Disk (软盘)

Monolithic (单体) Structure – Original UNIX

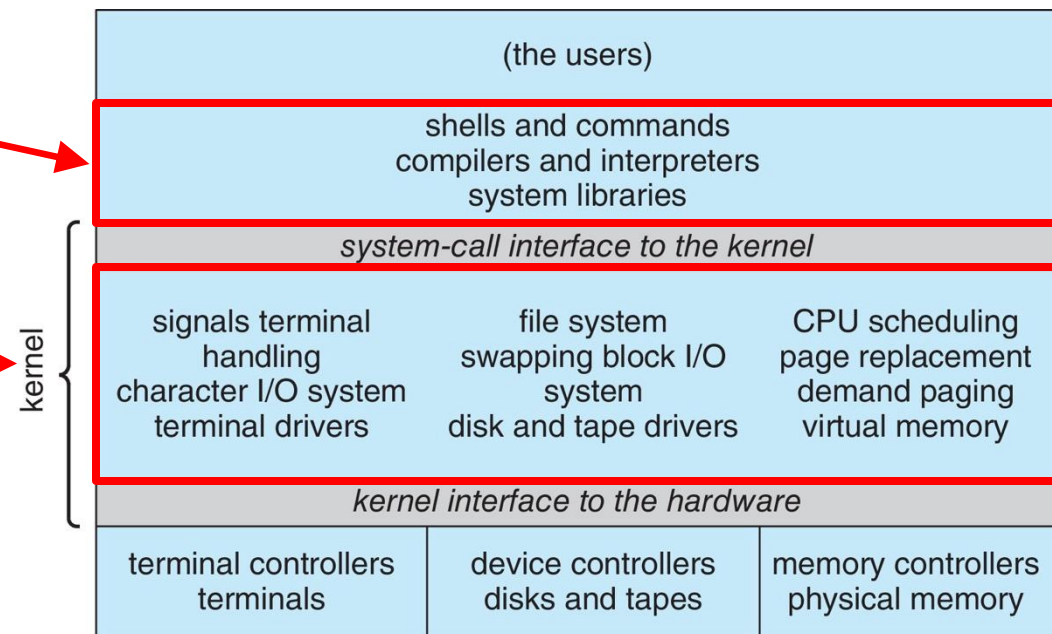
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts

- **Systems programs:**

- ▶ Primarily run in user space.
- ▶ Shell programs, text editors, command line utilities

- **The kernel**

- ▶ Everything below the system-call interface and above the physical hardware
- ▶ Many functions for one level
- ▶ Provides the file system, CPU scheduling, memory management, and other operating-system functions;



Monolithic: 整体结构

Monolithic Structure – Original UNIX

□ Benefits:

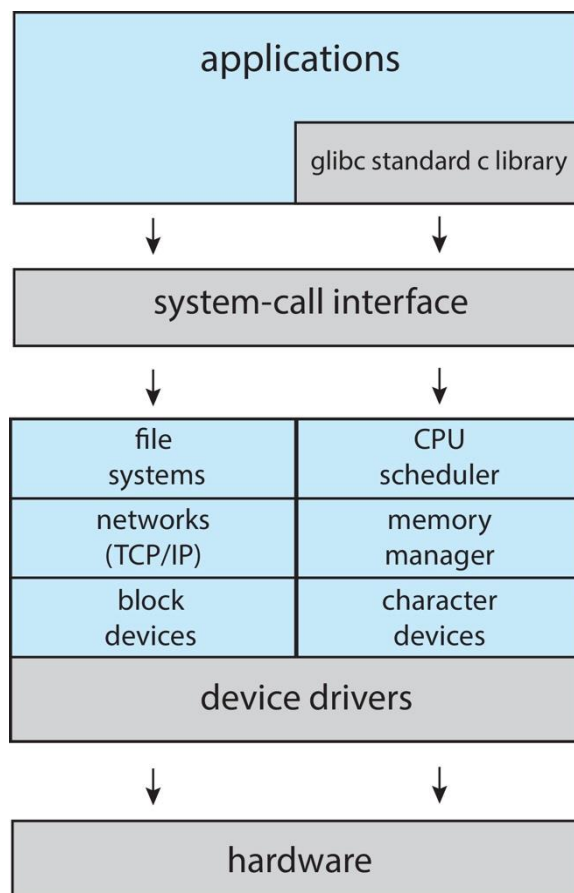
- **Efficiency**: Better performance since they have direct access to hardware resources without additional layers of abstraction.
- **Simplicity**: Simple design, easier to develop and maintain.

□ Limitations:

- **Less Modular**: Challenging to add or remove features without affecting the entire system.
- **Limited Stability and Security**: All components operate within the same address space, a bug or error in one component can potentially crash the entire system, or worse, compromise system security.

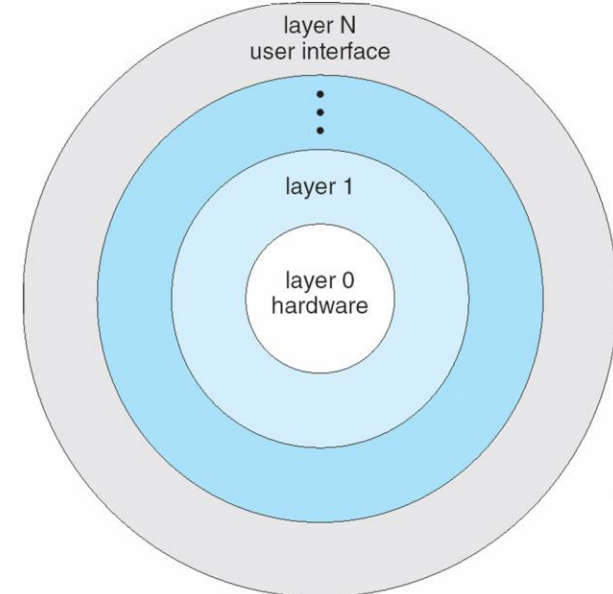
Extension: Linux System Structure

Linux = Monolithic plus **modular design**



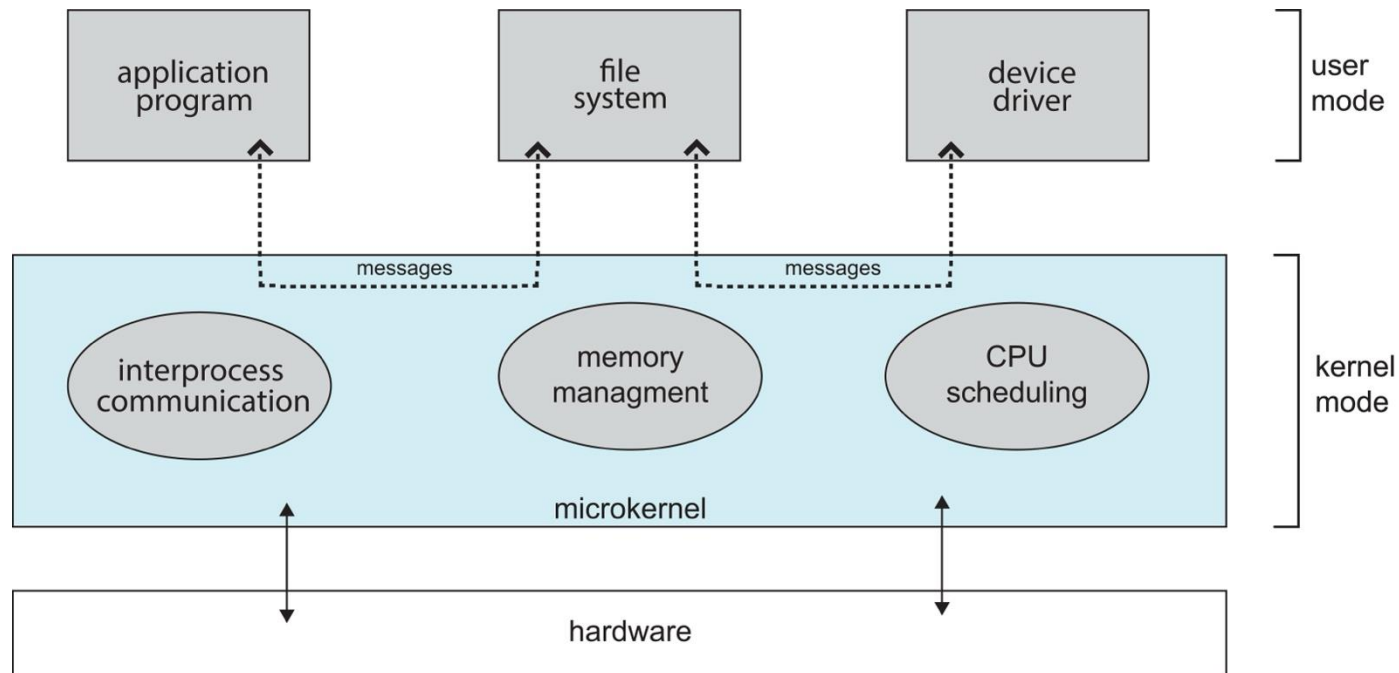
Layered Approach

- ❑ OS is divided into several 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.
 - ❑ Layers are selected such that each uses functions (operations) and services of only lower-level layers
- ❑ Benefits:
 - ❑ With the simplicity of construction and debugging, changes in one component affect itself only.
- ❑ Limitations:
 - ❑ Hard to separate the functionality of each layer.
 - ❑ The performance is poor.
- ❑ Fact:
 - ❑ Very few OSes use a pure layered structure.



Microkernel System Structure - Mach

- Moves as much from the kernel into “*user*” space
 - Lightweight kernel
 - Communication takes place between user modules using message passing
- Mach is an example of microkernel
 - macOS X kernel (Darwin) partly based on Mach



Microkernel System Structure - Mach

□ Benefits:

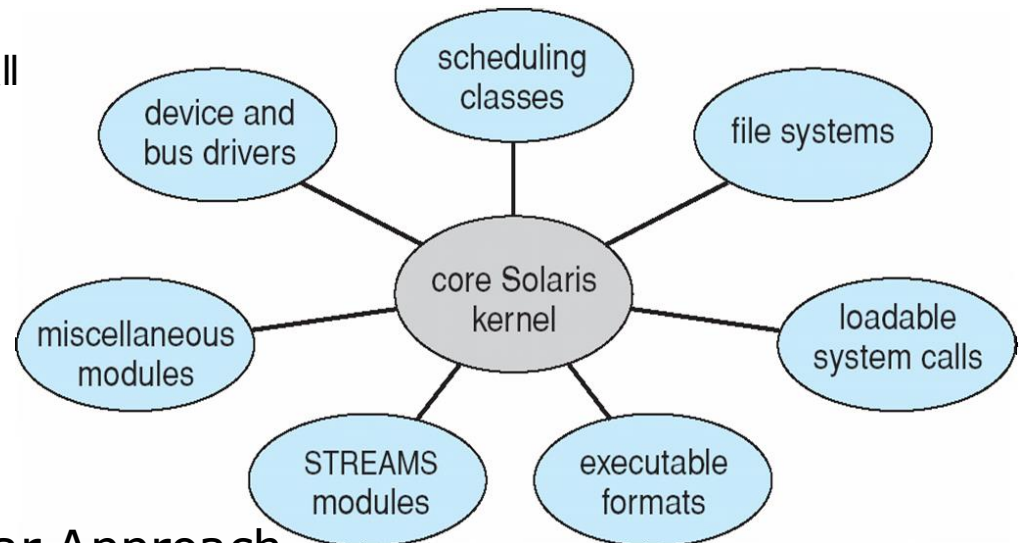
- Easier to extend
- Easier to port the operating system to new architectures
- More reliable (less code is running in kernel mode)
- More secure

□ Limitations:

- Slower performance due to the overhead of user space to kernel space communication

Modules

- ❑ Most modern operating systems implement kernel modules
 - ❑ Used in Linux, macOS, Solaris, and Windows.
 - ❑ Each core component is separated
 - ❑ Each talks to the others over known interfaces
 - ❑ Each is loadable as needed within the kernel
- ❑ Overall, similar to layers but with more flexibility, like microkernel but more efficient
 - ❑ **Flexibility:** Any module can call any other module.
 - ❑ **Efficiency:** No need to use message passing for communication

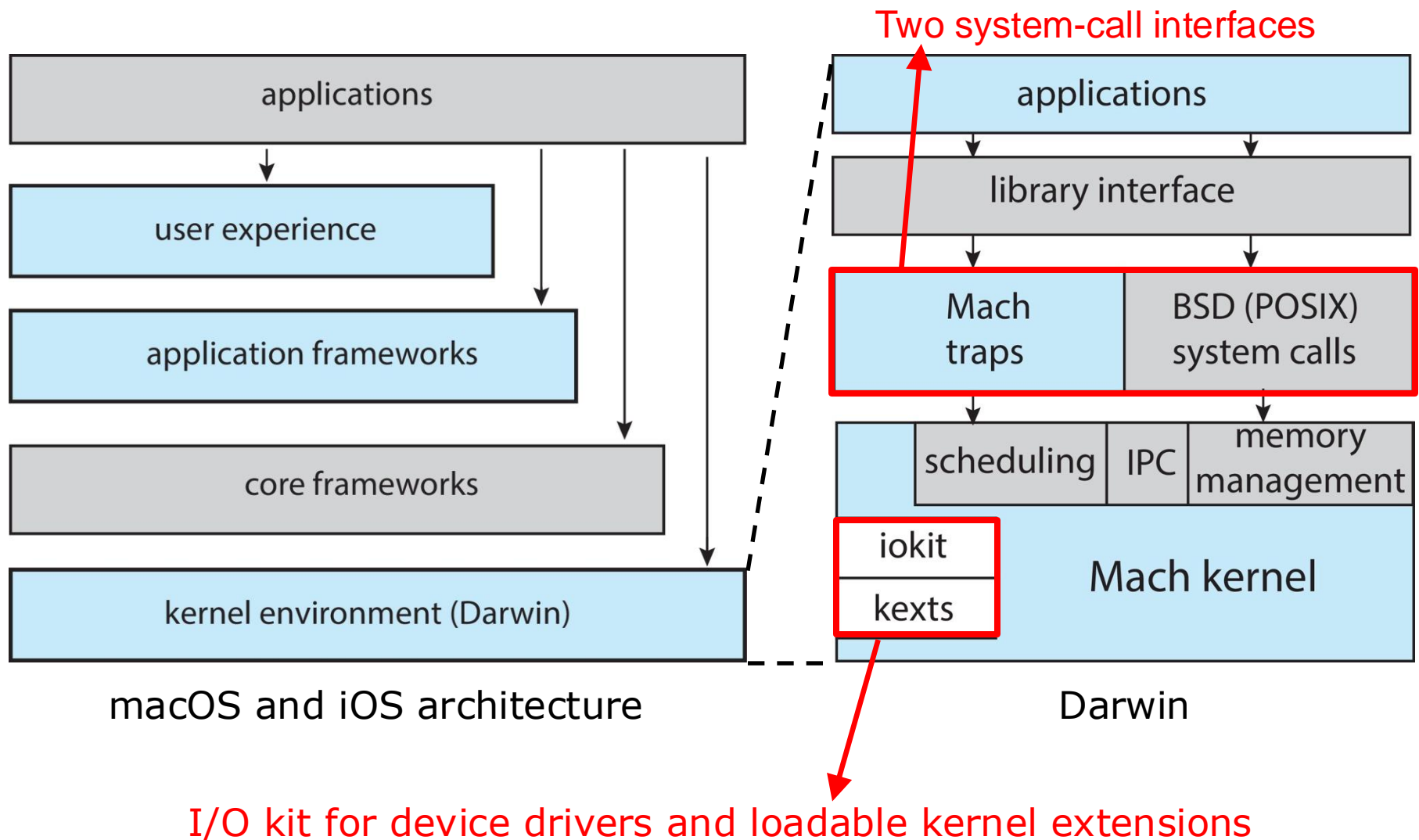


Solaris Modular Approach

Hybrid Systems

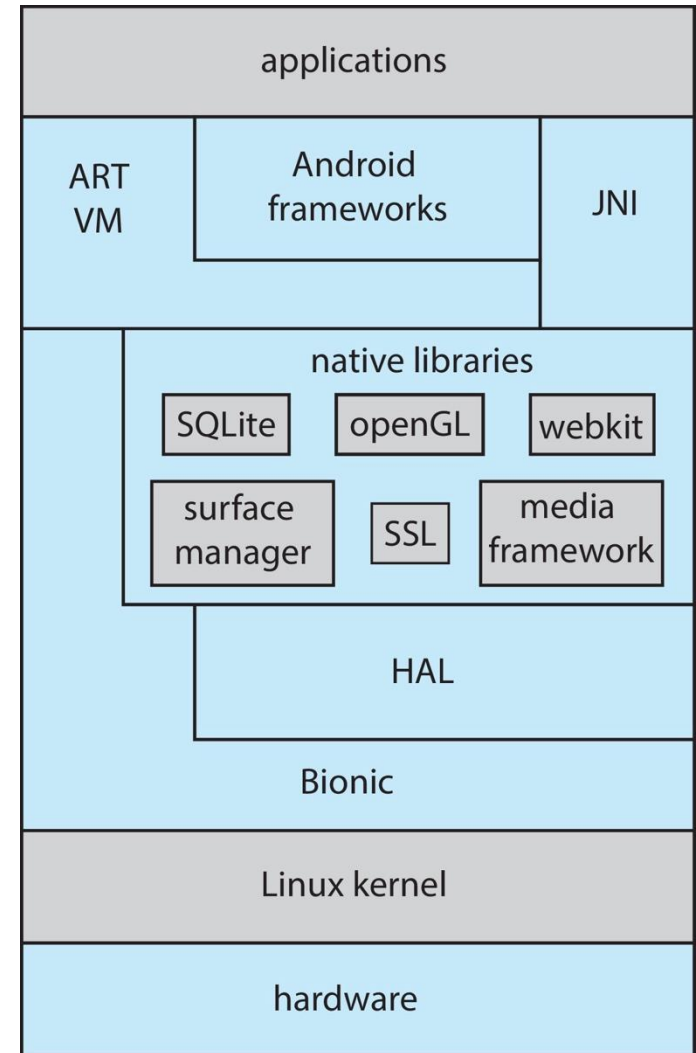
- Most modern operating systems are not one pure model
 - Hybrid system combines multiple approaches to balance performance, security, usability needs
- Examples:
 - **Linux and Solaris:** kernels are in kernel address space, so monolithic, plus modular for dynamic loading of functionality
 - **Windows:** mostly monolithic, plus microkernel for different subsystem personalities
 - **Apple macOS:** hybrid, layered, Aqua UI plus Cocoa programming environment

Example 1: macOS and iOS Structure



Example 2: Android

- ❑ Developed by Open Handset Alliance led by Google
 - ❑ Open Source
- ❑ Native libraries:
 - ❑ Frameworks for web browsers (webkit)
 - ❑ Database support (SQLite)
 - ❑ Network support (SSL)
- ❑ Based on Linux kernel but modified
 - ❑ Provides process, memory, device-driver management
 - ❑ Adds power management



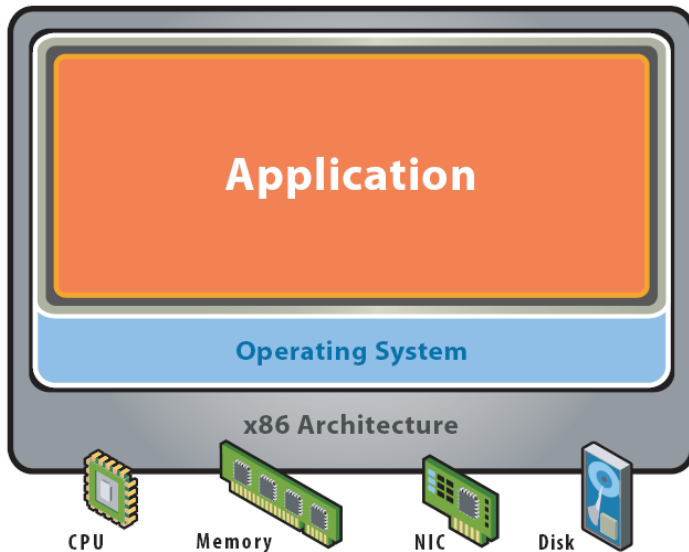
OS Structure Extension: Virtual Machines

Virtual Machines

- Goals of virtualization (虚拟化) :
 - Centralize admin tasks
 - Improve scalability and resource utilization: More users
- Solution:
 - A **virtual machine** takes the layered approach to its logical conclusion.
 - ▶ It treats the OS kernel as part of the hardware.
 - A virtual machine provides an interface identical to the underlying bare hardware.
- Effect:
 - The **host OS** creates the illusion that a process has its own processor and (virtual) memory.
 - Each **guest OS** is provided with a (virtual) copy of the underlying computer.

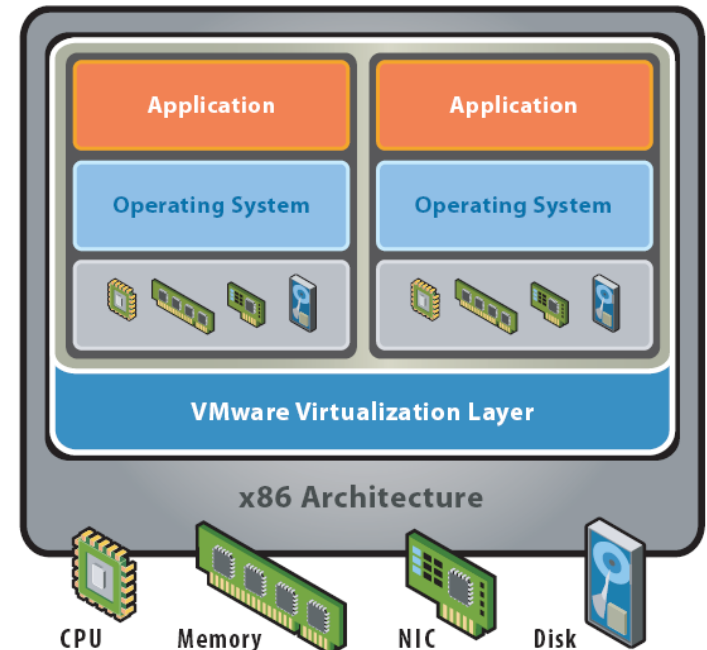
Benefits of Virtualization

Before Virtualization



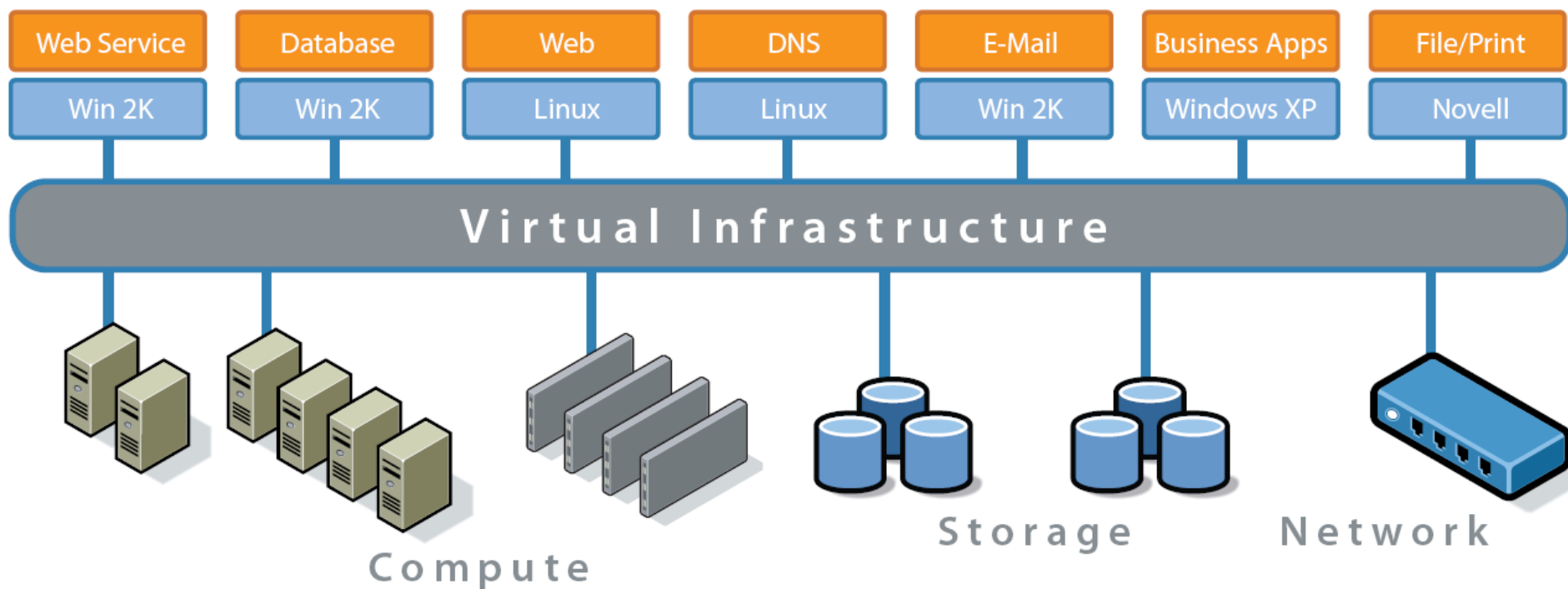
- Single OS image per machine
- Software and hardware tightly coupled
- Underutilized resources
- Inflexible and costly infrastructure

After Virtualization



- Multiple OSs on a single machine
- Hardware-independence of operating system and applications
- Better utilization of resources
- Encapsulating OS and application into virtual machines

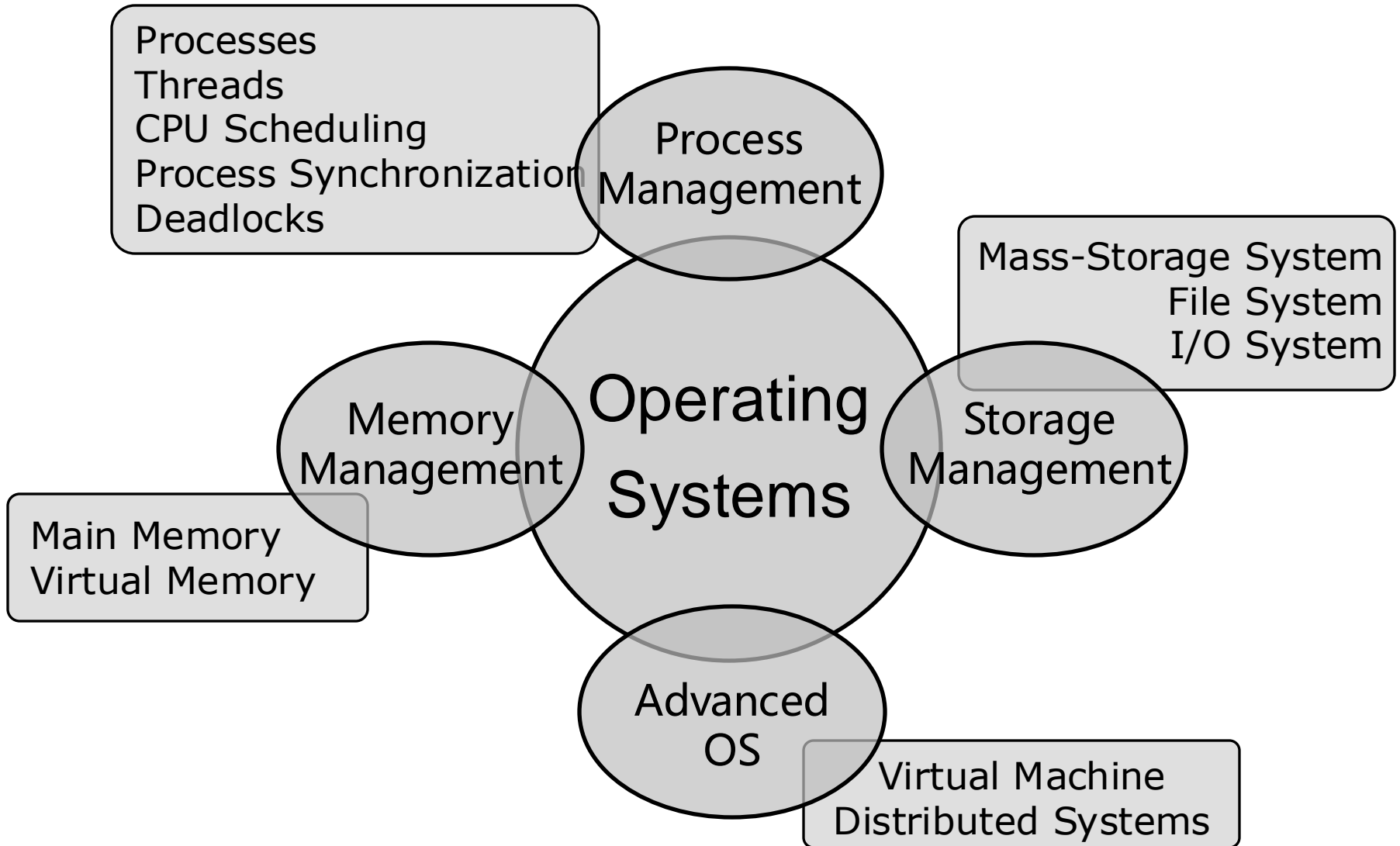
Example: Virtual Infrastructure for Data Center



Summary

- ❑ OS creates an environment for the execution of programs by providing services to users and programs
- ❑ OS functions:
 - ❑ User interface:
 - ▶ Command line interfaces (CLI)
 - ▶ Graphical user interfaces (GUI)
 - Touch-screen interfaces
 - ❑ System calls provide an interface to the available services by an OS
 - ▶ APIs are used by user programs for accessing system-call services
 - ❑ OS includes a collection of system services as utilities
- ❑ Introduced the benefits and limitations of different OS structures
- ❑ Virtualization is created to support multiple OSes on a single machine

Operating System Topics



Course Syllabus

Part	Topic	Week	Date	Note
Process Management	Processes	2	Feb 26	
	Threads		Feb 28	
	CPU Scheduling	3	Mar 05	HW1 Out
	Process Synchronization		Mar 07	
	Deadlock	4	Mar 12	HW1 Due, HW2 Out
Memory Management	Main Memory		Mar 14	
	Virtual Memory	5	Mar 19	HW2 Due, HW3 Out
Storage Management	Mass-Storage System		Mar 21	Survey Out
	File System	6	Mar 26	HW3 Due
			Mar 28	HW4 Out
	I/O System	7	Apr 02	
Advanced Topics	Virtual Machine		Apr 04 (TBD)	HW4 Due
	Distributed Systems & Review	8	Apr 09	
	Guest Lecture		Apr 11	Survey Due on Apr 20

Homework

- Reading
 - Chapter 2: Operating-System Structures