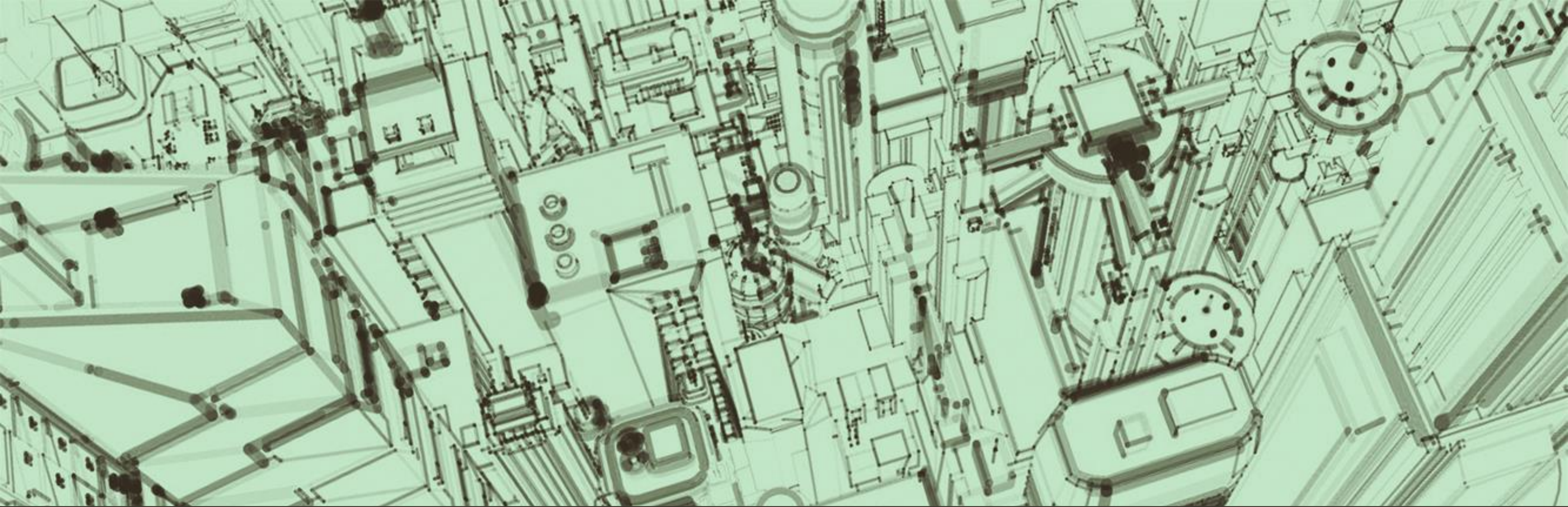


Chapter 2: **Operating-System Structures**

Users, processes and other systems.

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

Recap Question of Chapter 1.

Which of these are valid examples of privilege instructions?

a. Turn off all Interrupts
b. ~~Reading the status of Processor~~
c. Shut down the system
d. ~~Reading the System Time~~
e. Clear the Memory or Remove a process from the Memory
f. ~~Load and store instructions~~
g. Sending commands to I/O devices

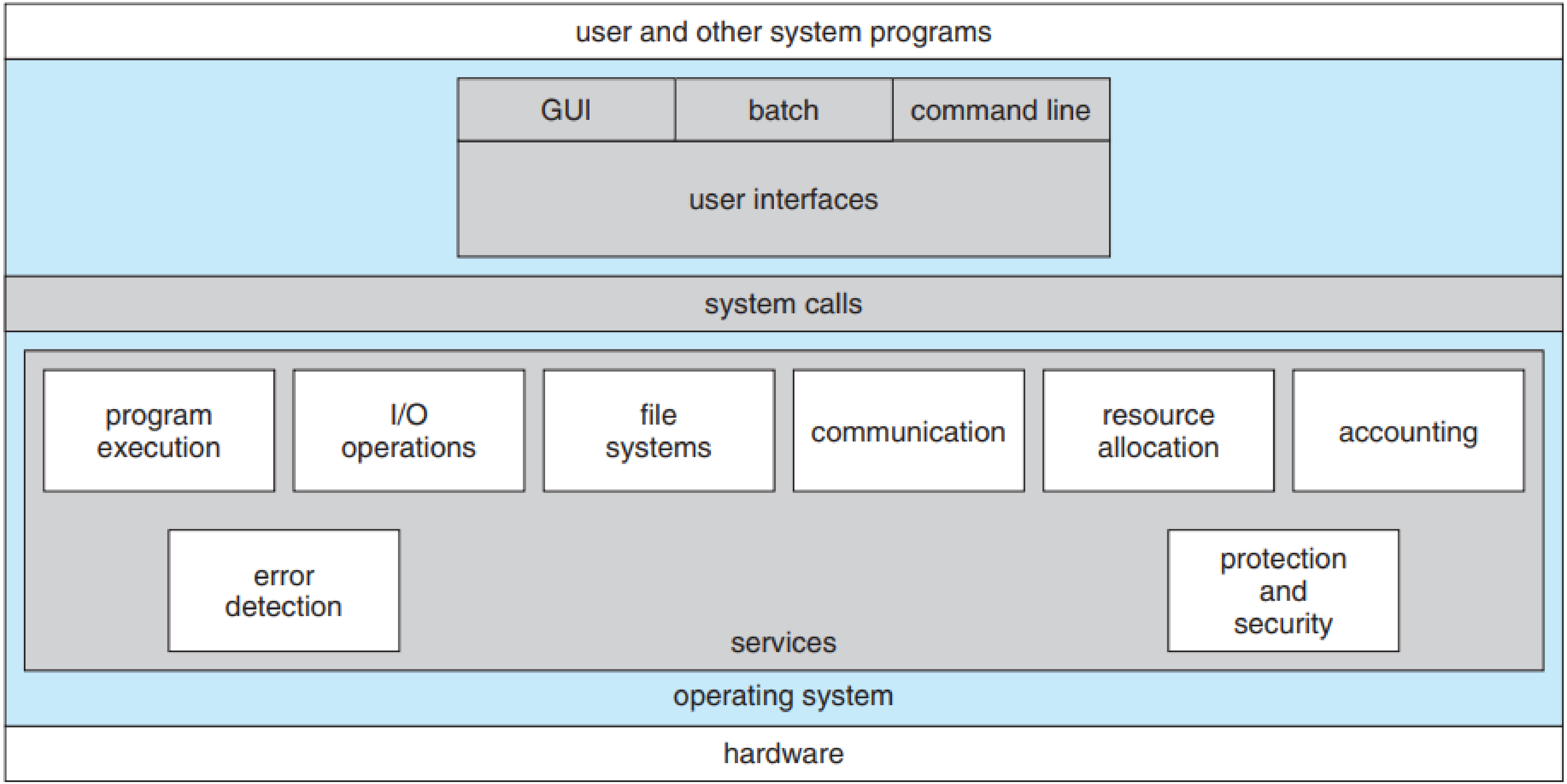


Figure 2.1 A view of operating system services.

Operating System Services (1/0)

- OS provides an **environment** for **execution of programs and services** to programs and users.
- Provides two sets of services:

Helpful to users

Ensuring efficient operation of system

Operating System Services (1/1)

- ***First Set*** **of** **OS-services** provides functions that are **helpful to the user:**
  - **User Interface (UI):**
    - Varies between CLI, GUI, <u>Batch</u>
  - **Program execution:** 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.

Batch Interface

```
Command started at: Dec 9, 2004 4:01:54 PM

Batch environment set to:


  for target = newtarget
    user     = undefined (default is abc )
    password = ******
    product  = undefined (default is System Management Hub)
    version  = undefined (default is last version)

  for target = newtarget2
    user     = undefined (default is abc )
    password = ******
    product  = undefined (default is System Management Hub)
    version  = undefined (default is last version)

  default settings:
    user     = undefined (default is abc )
    password = *******
    product  = undefined (default is System Management Hub)
    version  = undefined (default is last version)


  target              = newtarget2
  milsrv              = localhost
  xmlstyle            = off

  ARGBATCH_USER       = undefined
  ARGBATCH_PASSWORD   = undefined

  ARGBATCH_LANGUAGE   = undefined
  ARGBATCH_LINE_SIZE  = undefined

Command ended at:   Dec 9, 2004 4:01:54 PM
```

Operating System Services (1/2)

- **File-system manipulation:**
  - Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.
- **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)

Operating System Services (1/3)

- **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 (2/1)

- **Second Set of OS-services** 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. Types of resources: CPU cycles, main memory, file storage, I/O devices.
  - **Accounting:**
    - To keep track of which users use how much and what kinds of computer resources

Operating System Services (2/1)

- **Protection and security:**
  - Owners of information stored in a multiuser or networked computer system 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

User Operating System Interface: CLI

- CLI or command interpreter 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, sometimes just names of programs

Bourne Shell

Default

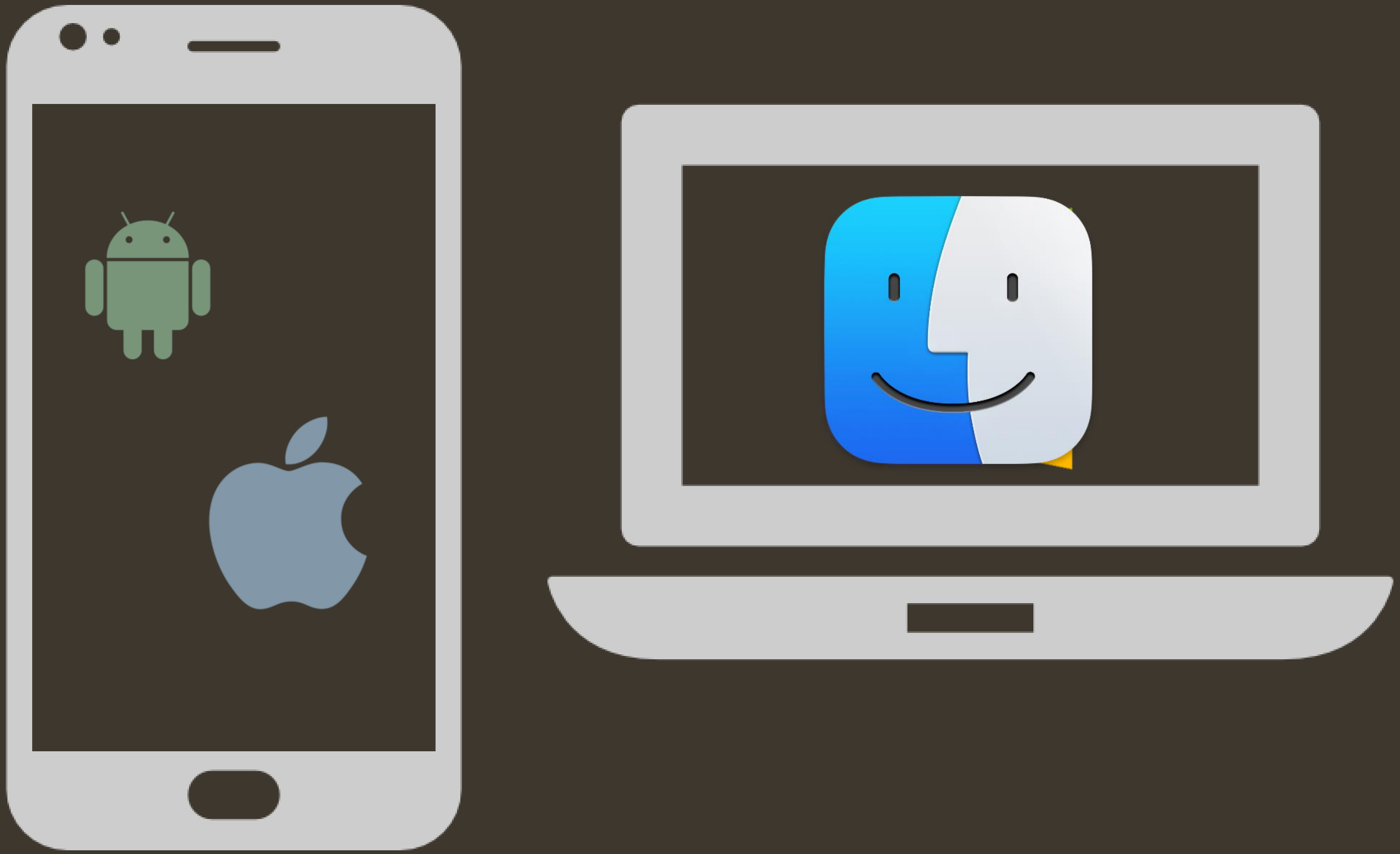
New Info Close Execute Bookmarks

Default Default

```
PBG-Mac-Pro:~ pbg$ w
15:24  up 56 mins, 2 users, load averages: 1.51 1.53 1.65
USER     TTY      FROM              LOGIN@  IDLE WHAT
pbg      console  -                14:34      50 -
pbg      s000     -                15:05       - w
PBG-Mac-Pro:~ pbg$ iostat 5
          disk0           disk1           disk10        cpu     load average
    KB/t tps  MB/s    KB/t tps  MB/s    KB/t tps  MB/s  us sy id   1m   5m   15m
   33.75 343 11.30   64.31  14  0.88   39.67   0  0.02  11  5 84  1.51 1.53 1.65
    5.27 320  1.65    0.00   0  0.00    0.00   0  0.00   4  2 94  1.39 1.51 1.65
    4.28 329  1.37    0.00   0  0.00    0.00   0  0.00   5  3 92  1.44 1.51 1.65
^C
PBG-Mac-Pro:~ pbg$ ls
Applications                    Music                           WebEx
Applications (Parallels)        Pando Packages                  config.log
Desktop                         Pictures                        getsmartdata.txt
Documents                       Public                          imp
Downloads                       Sites                           log
Dropbox                         Thumbs.db                       panda-dist
Library                         Virtual Machines                prob.txt
Movies                          Volumes                         scripts
PBG-Mac-Pro:~ pbg$ pwd
/Users/pbg
PBG-Mac-Pro:~ pbg$ ping 192.168.1.1
PING 192.168.1.1 (192.168.1.1): 56 data bytes
64 bytes from 192.168.1.1: icmp_seq=0 ttl=64 time=2.257 ms
64 bytes from 192.168.1.1: icmp_seq=1 ttl=64 time=1.262 ms
^C
--- 192.168.1.1 ping statistics ---
2 packets transmitted, 2 packets received, 0.0% packet loss
round-trip min/avg/max/stddev = 1.262/1.760/2.257/0.498 ms
PBG-Mac-Pro:~ pbg$ 
```

Operating Systems, GUIs and Devices

To meet specific needs of the device it runs on.



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
  - Invented at Xerox PARC, 1973

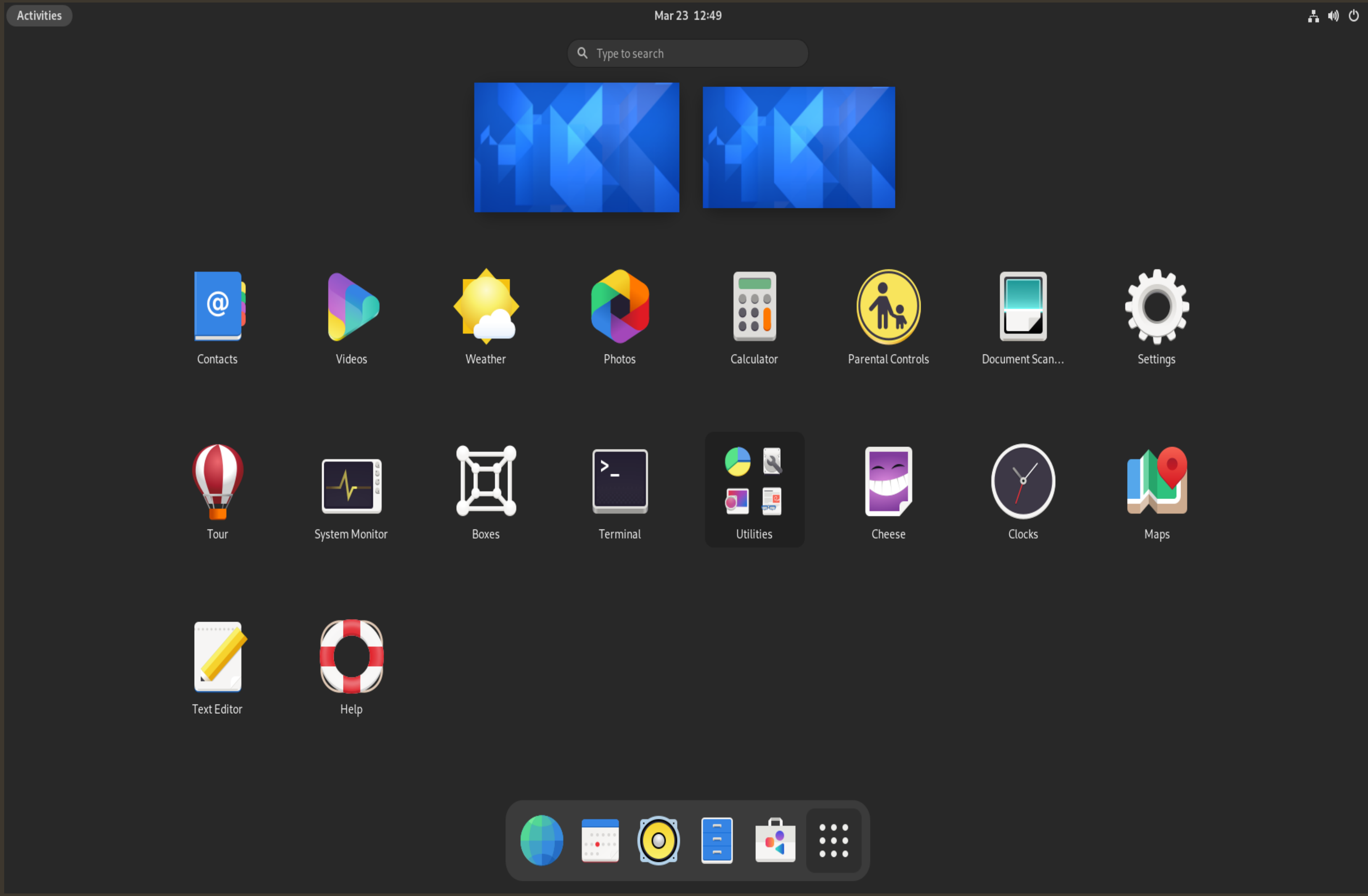
User Operating System Interface: GUI

Many systems now include both CLI and GUI interfaces

Microsoft Windows is GUI with CLI “**command” shell**

Apple Mac OS X is “Aqua” GUI interface with UNIX kernel underneath and shells available

Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)



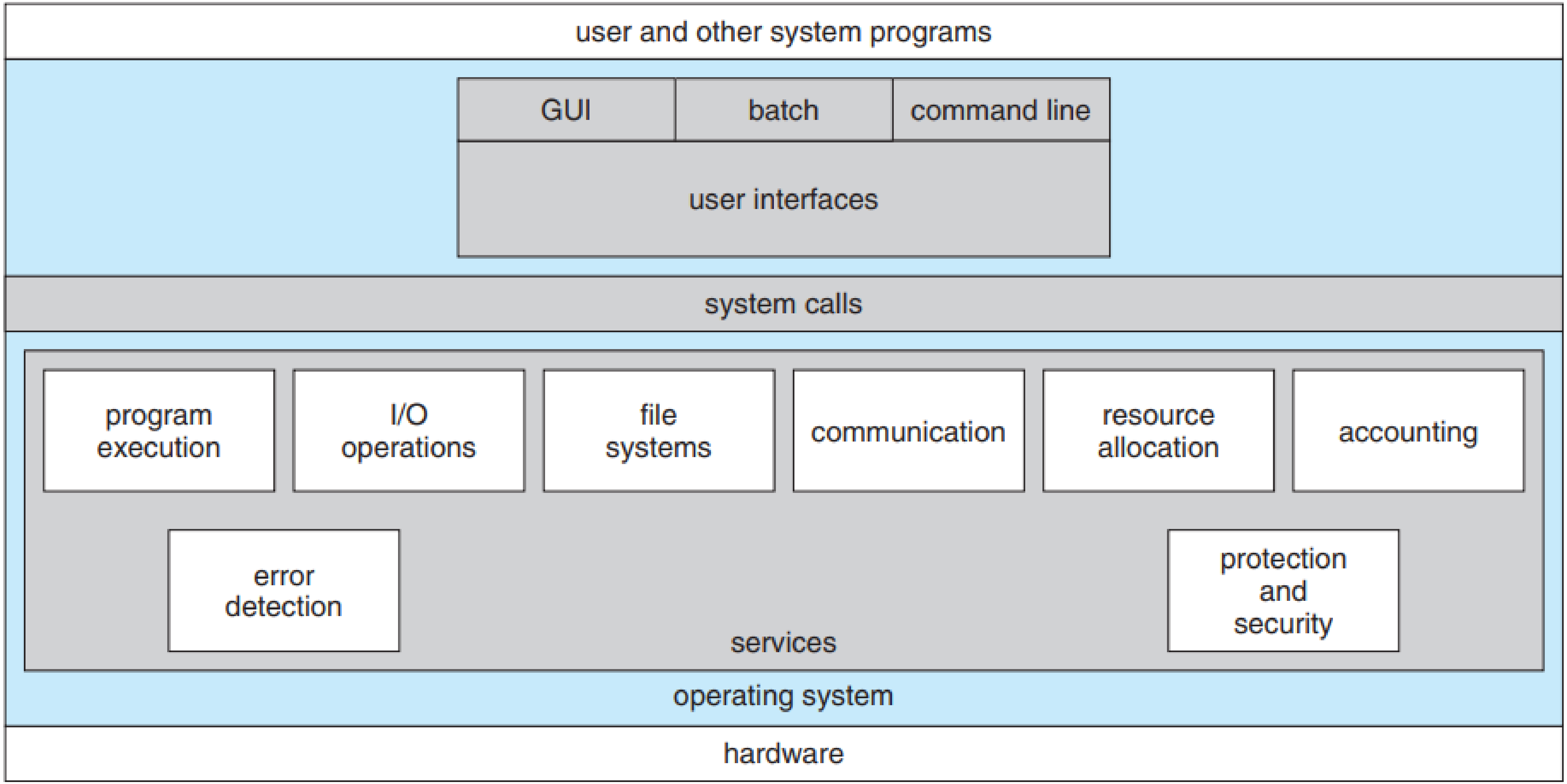


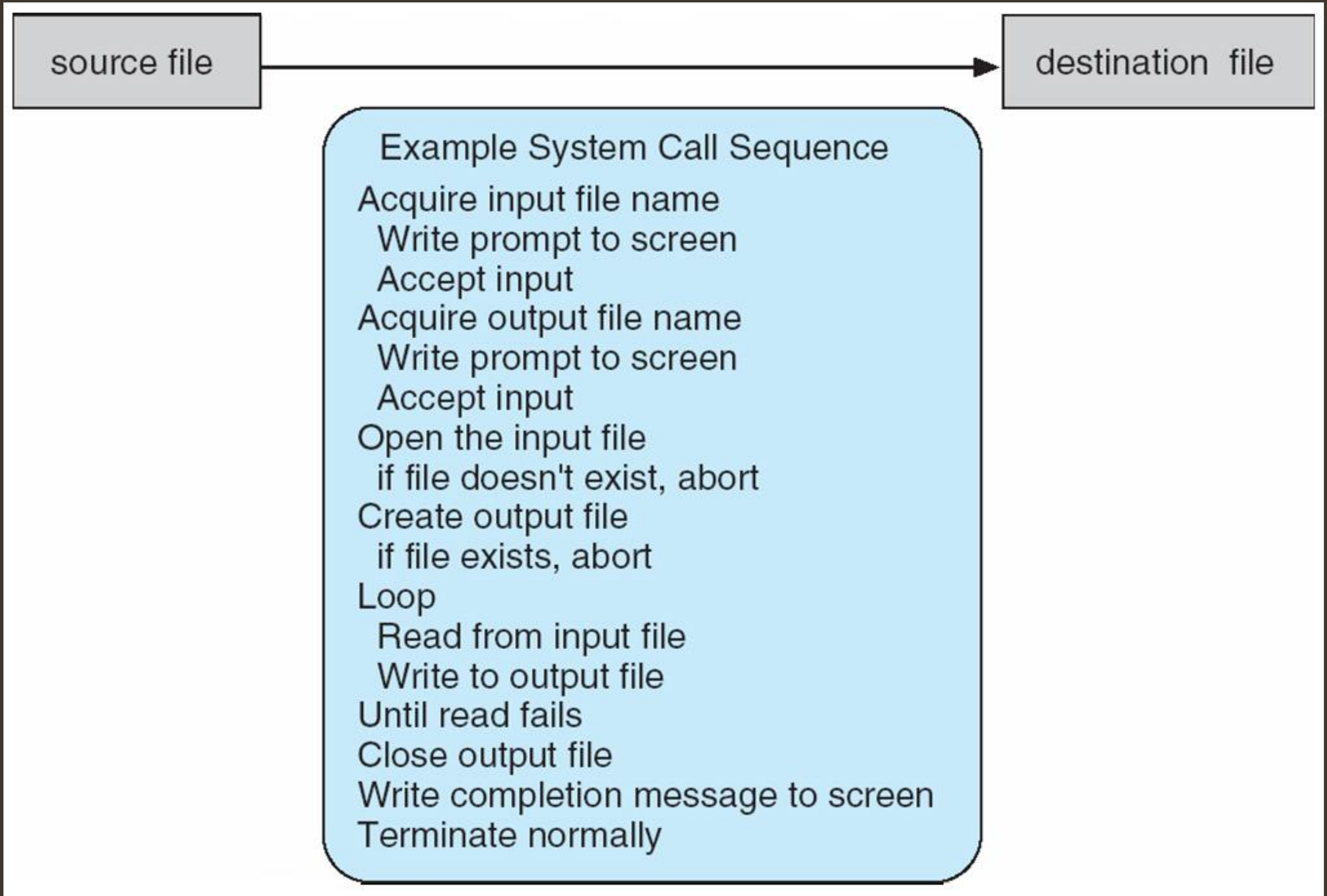
Figure 2.1 A view of operating system services.

System Calls

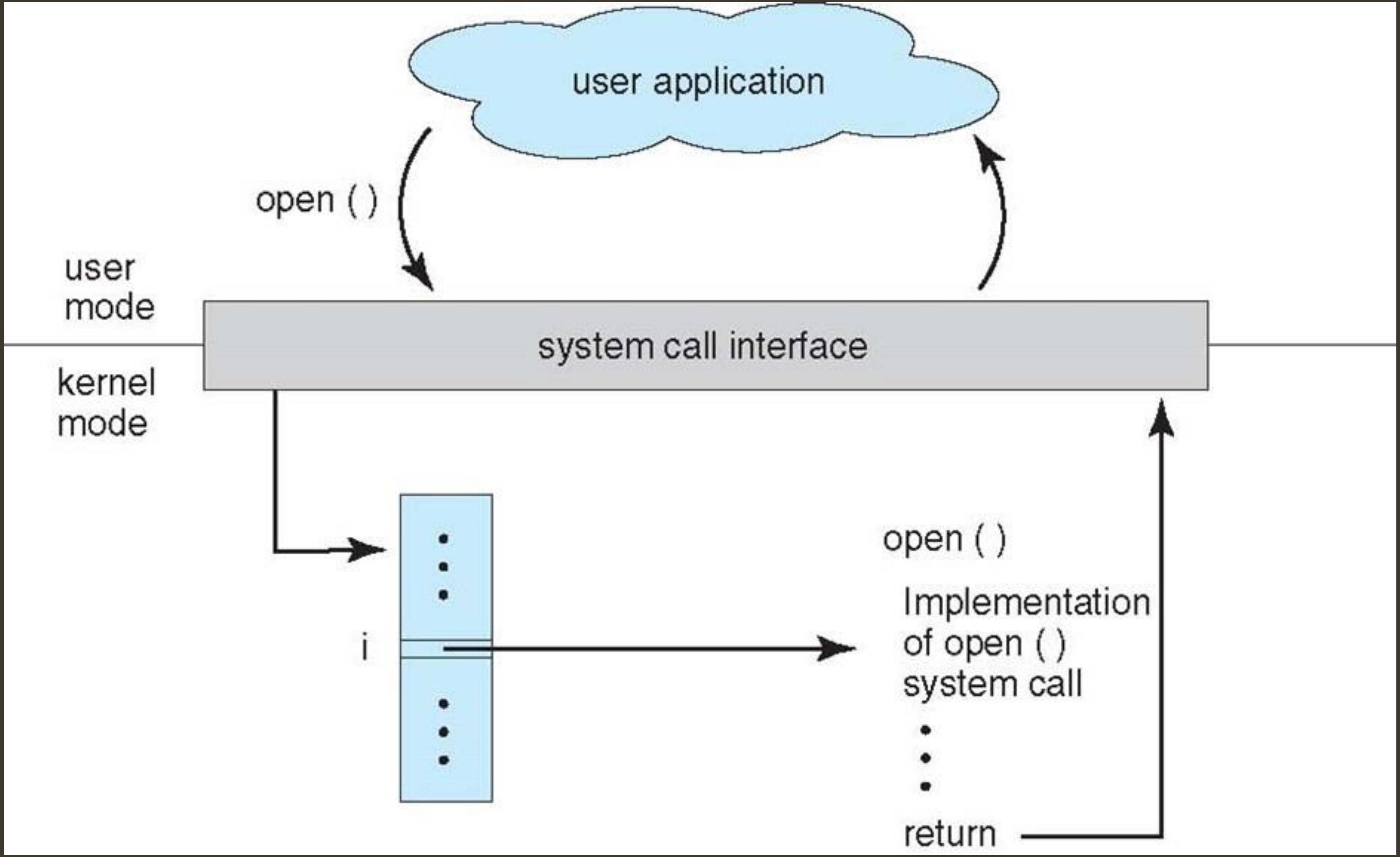
- **Programming interface** to the **services** provided by the OS
- Typically written in a high-level language (C or C++)
- Mostly accessed by programs via a high-level **Application Programming Interface (API)** rather than direct system call use
- Three most common APIs are **Win32 API** for Windows, **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



API – System Call - OS Relationship



System Call Parameter Passing

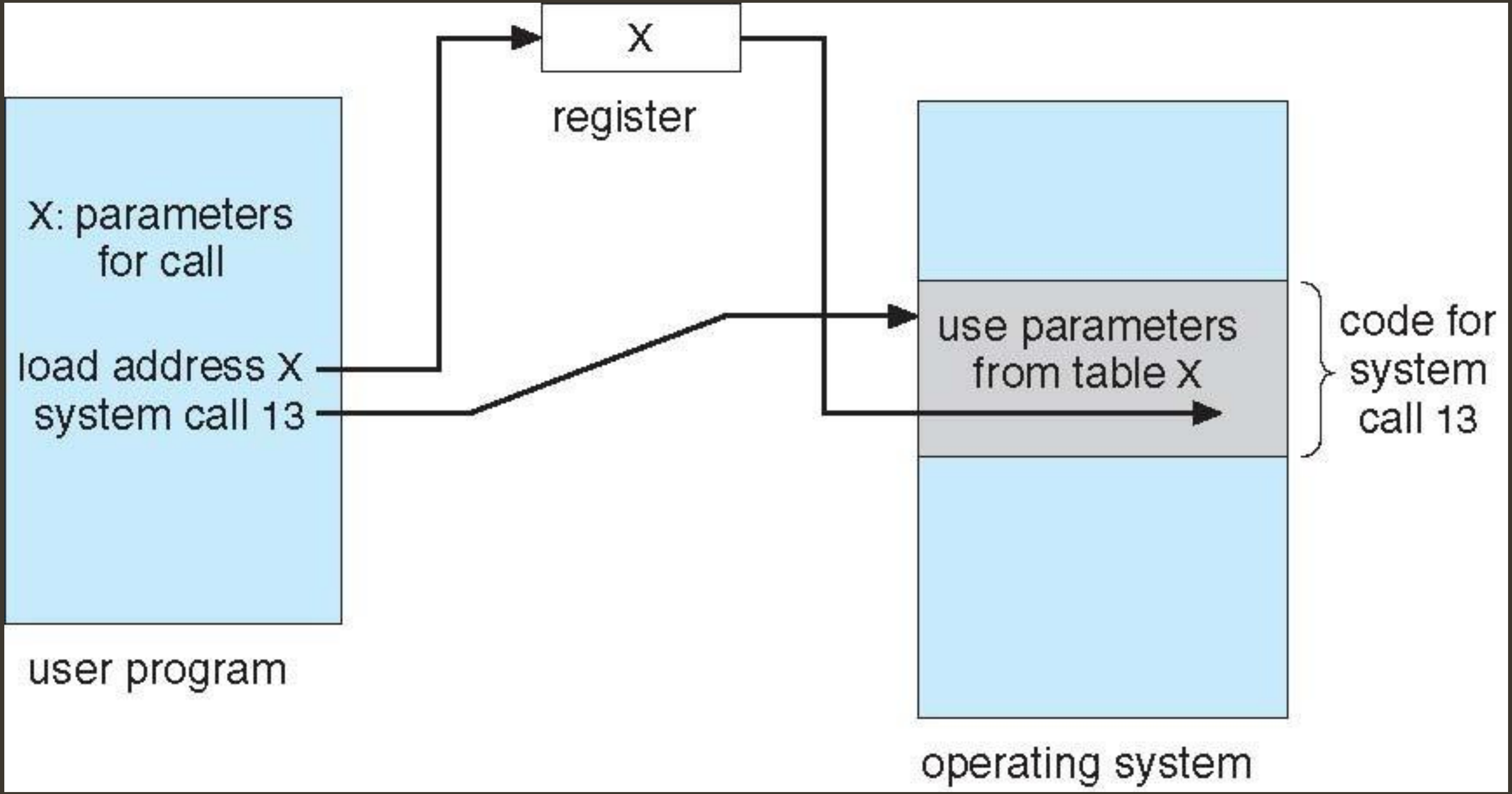
- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- **Three general methods** used to pass parameters to the OS:
  1. Simplest: Pass the parameters in registers

System Call Parameter Passing

2. 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
3. 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
- File Management
- Device Management
- Information Maintenance
- Communications
- Protection

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

Types of System Calls.

Device Management

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

Types of System Calls.

Information Maintenance

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

Types of System Calls.

Commun-ications

- 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.

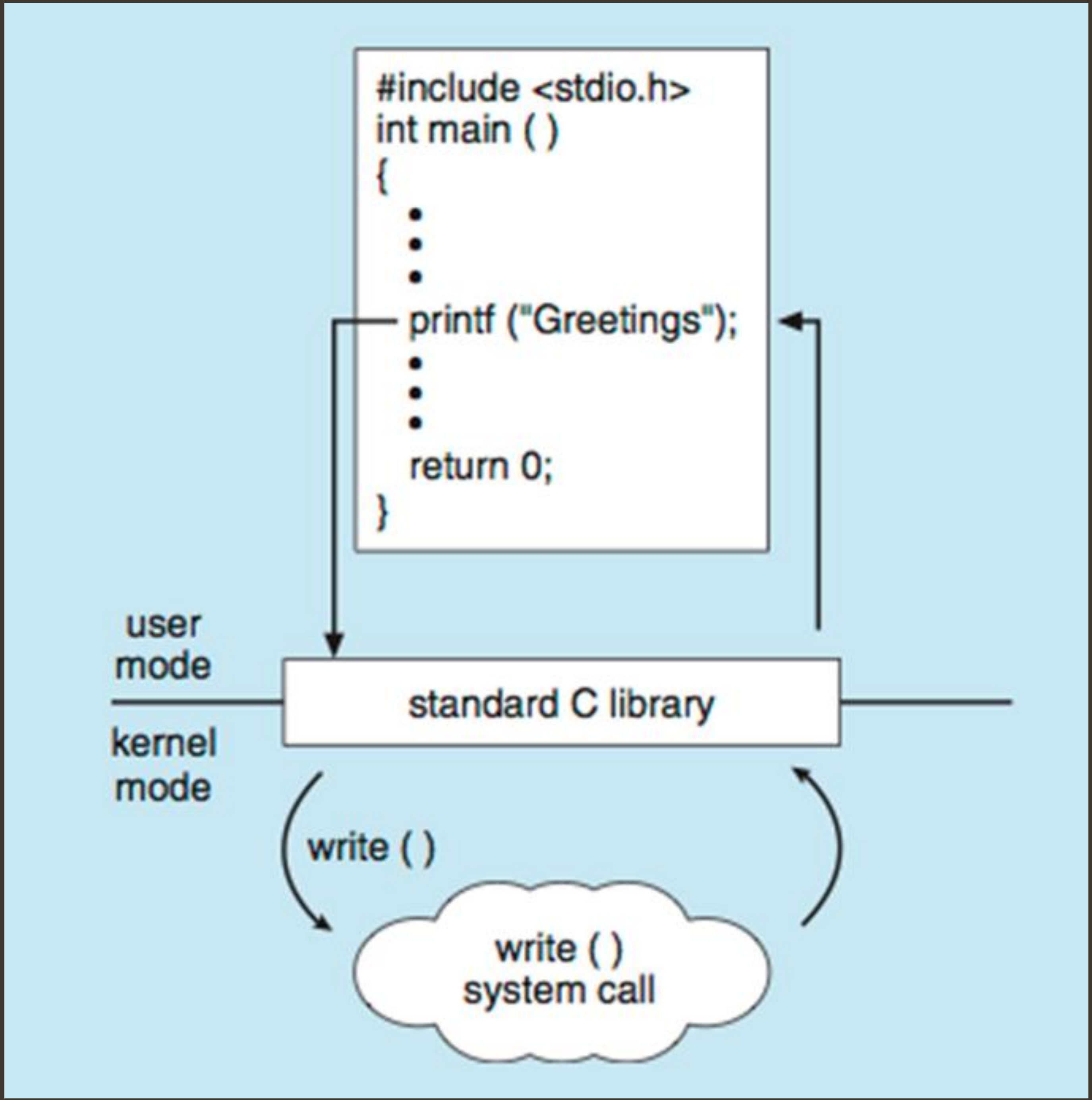
Protection

- 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()
Device Manipulation	SetConsoleMode()	ioctl()
	ReadConsole()	read()
	WriteConsole()	write()
Information Maintenance	GetCurrentProcessID()	getpid()
	SetTimer()	alarm()
	Sleep()	sleep()
Communication	CreatePipe()	pipe()
	CreateFileMapping()	shmget()
	MapViewOfFile()	mmap()
Protection	SetFileSecurity()	chmod()
	InitlializeSecurityDescriptor()	umask()
	SetSecurityDescriptorGroup()	chown()

Standard C Library Example



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

System Programs (1/1)

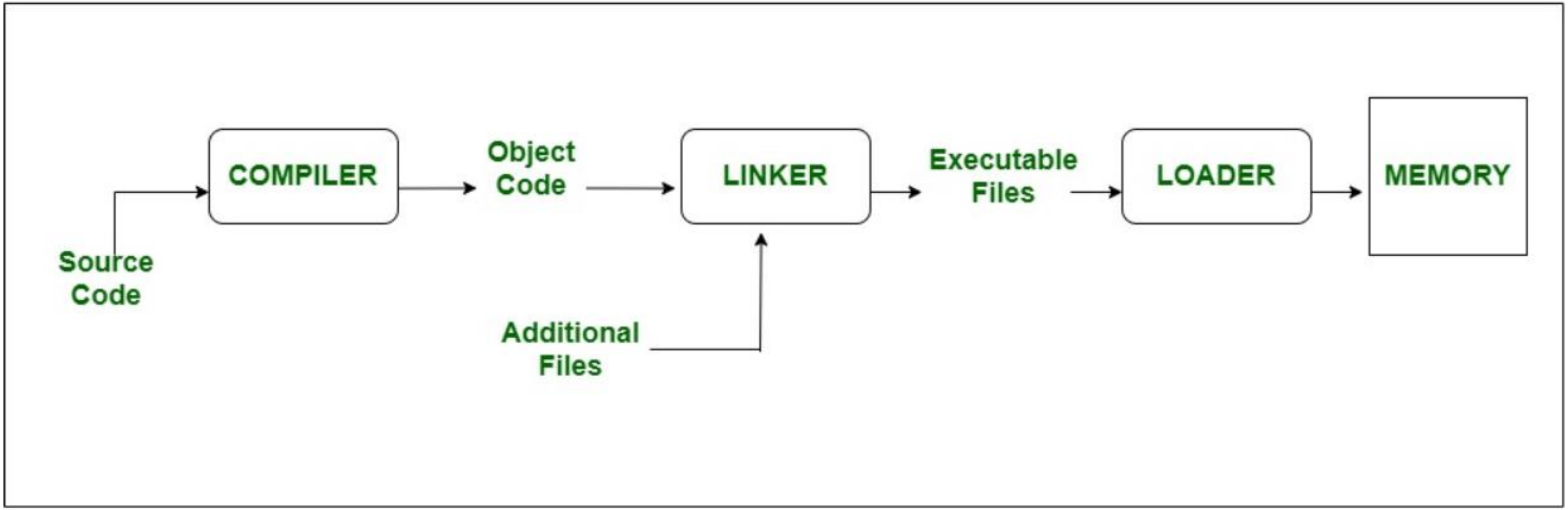
- Provide a **convenient environment for program development and execution.** Can be divided into:
  - File manipulation
  - Status information sometimes stored in a File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Background services
  - Application programs
- Most users’ view of the operation system is defined by system programs, not the actual system calls

System Programs (1/2)

- **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

System Programs (1/3)

- **Program loading and execution:**
  - Absolute loaders, relocatable loaders, linkage editors, and overlay-loaders, debugging systems for higher-level and machine language
- **Programming-language support:**
  - 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


COMPILER
Object Code
LINKER
Executable Files
LOADER
MEMORY
Source Code
Additional Files

AFTER THE BREAK.

RECAP TIME – Chapter 2 (Part 1)

1. How many categories are the services and functions provided by an operating system divided into?
   a) Six
   b) Four
   c) Two
   d) Actually no categories

2. Name them.

Helpful to users

Ensuring efficient operation of system

RECAP TIME – Chapter 2 (Part 1)

1. Number of methods for passing parameters to the operating system.
    a) Six
    b) Four
    c) Three
    d) Two

2. Identify them.

Registers

Blocks or tables

Stack

System Programs (1/2)

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

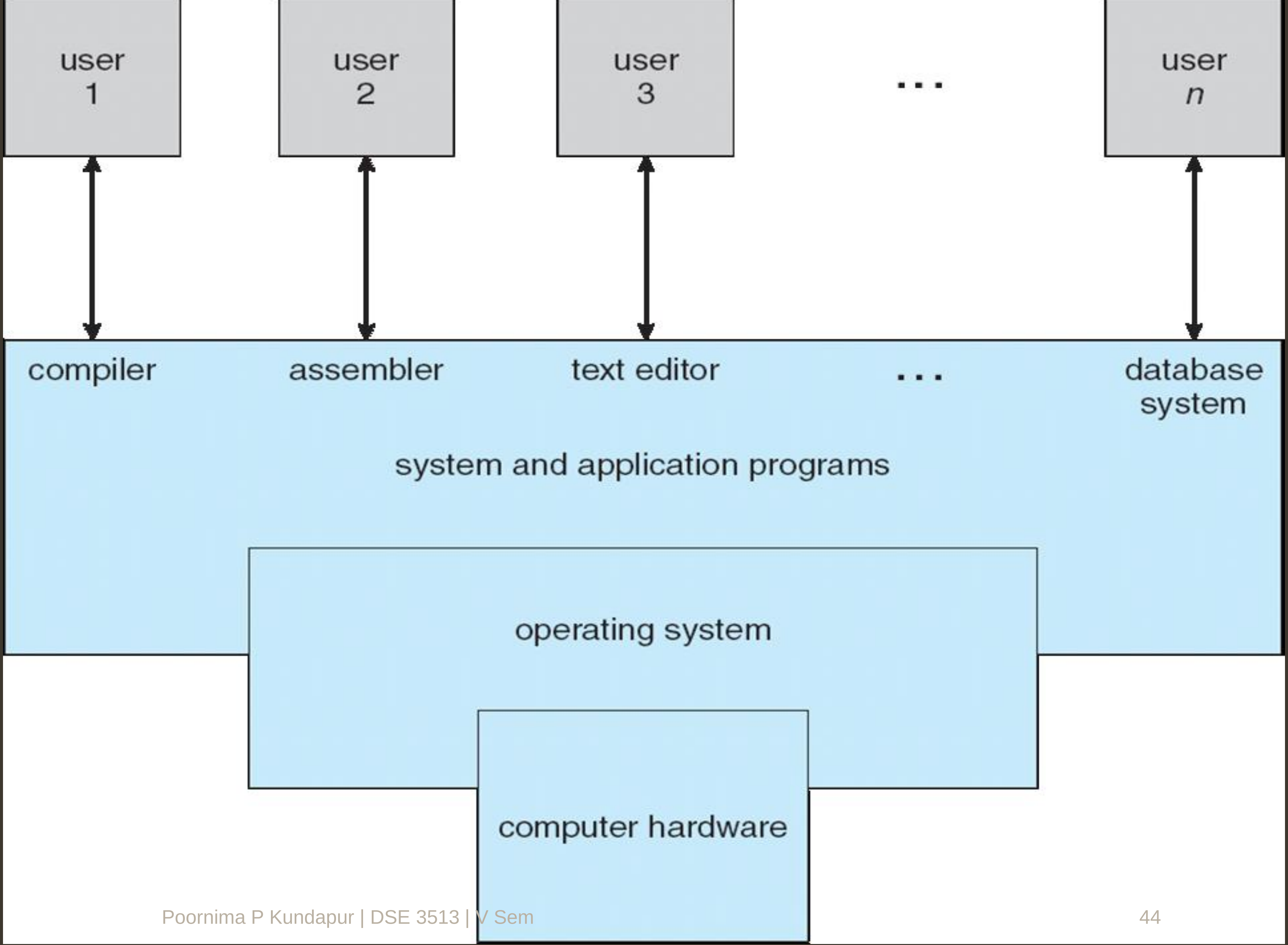
System Programs (1/2)

- **Application programs:**
  - Don’t pertain to system
  - Run by users
  - Not typically considered part of OS
  - Launched by command line, mouse click, finger poke

Operating System Structure

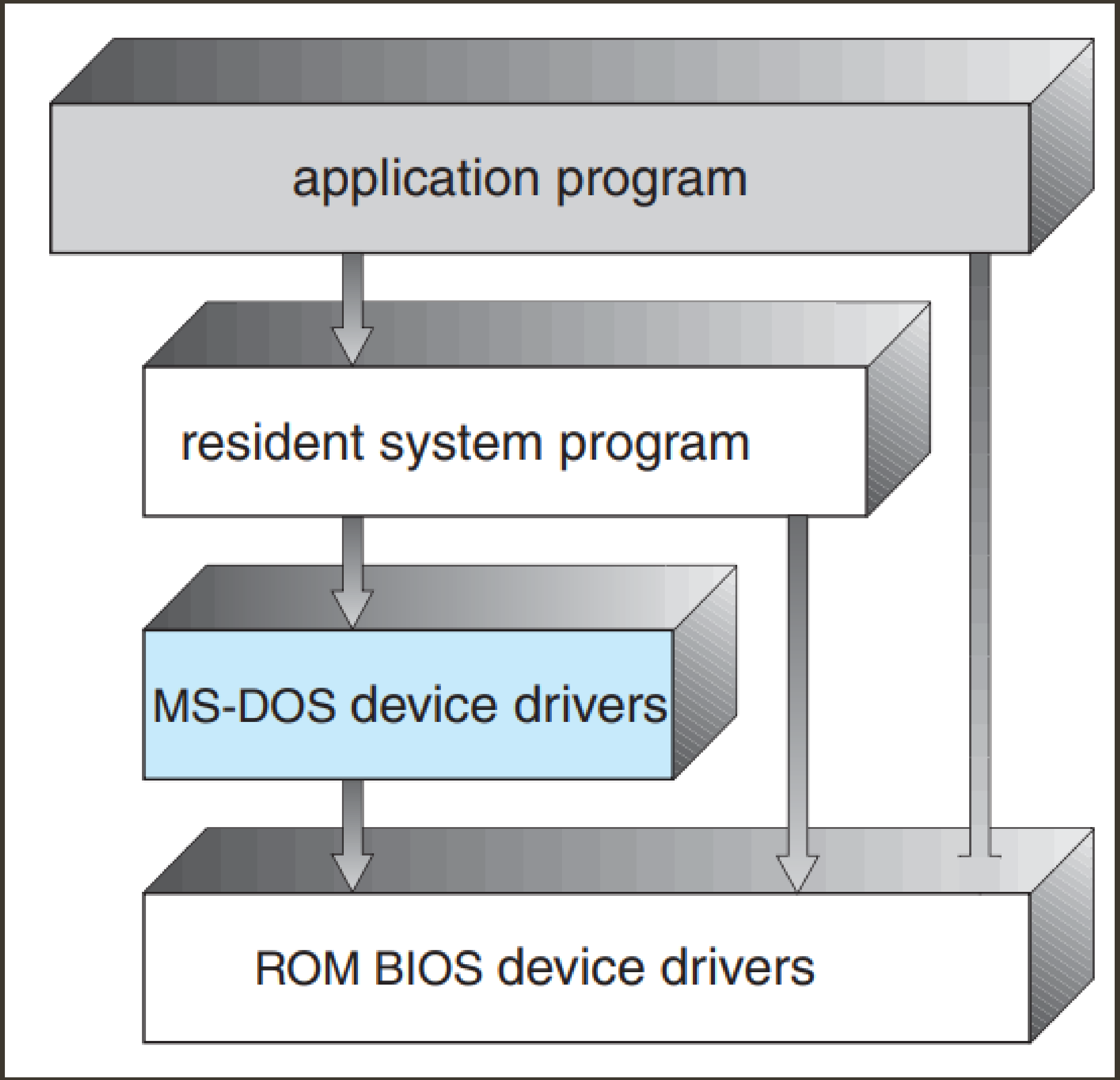
- General-purpose OS is very **large program**
- Various ways to structure ones
- Simple structure – MS-DOS
- More complex -- UNIX
- Layered – an abstraction
- Microkernel -Mach

Four Components of a Computer System



Simple Structure (MS DOS)

- MS-DOS was 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**

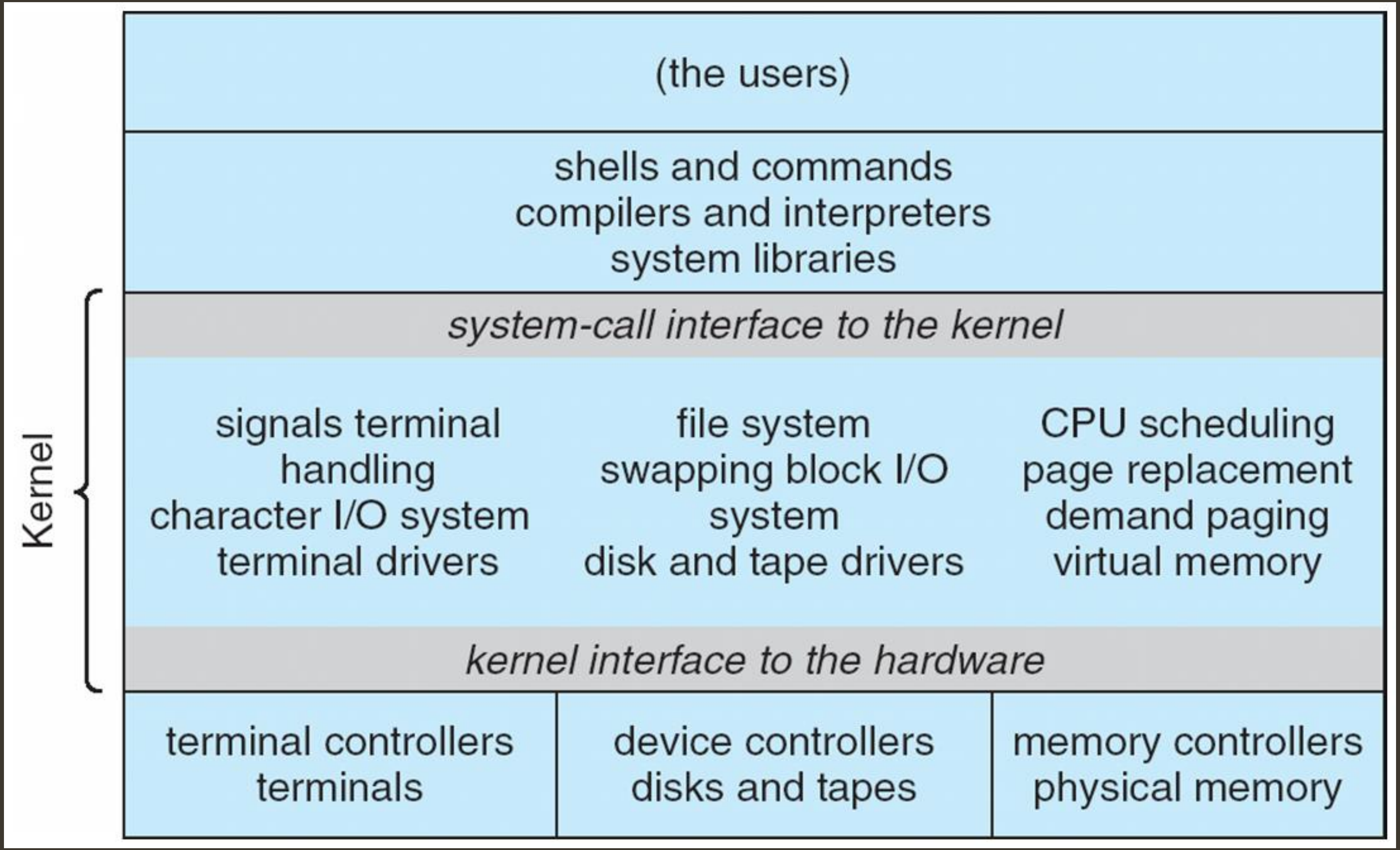


Non-Simple Structure (UNIX)

- Limited by hardware functionality, the original UNIX operating system had limited structuring with **two separable parts:**
  - **Systems 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

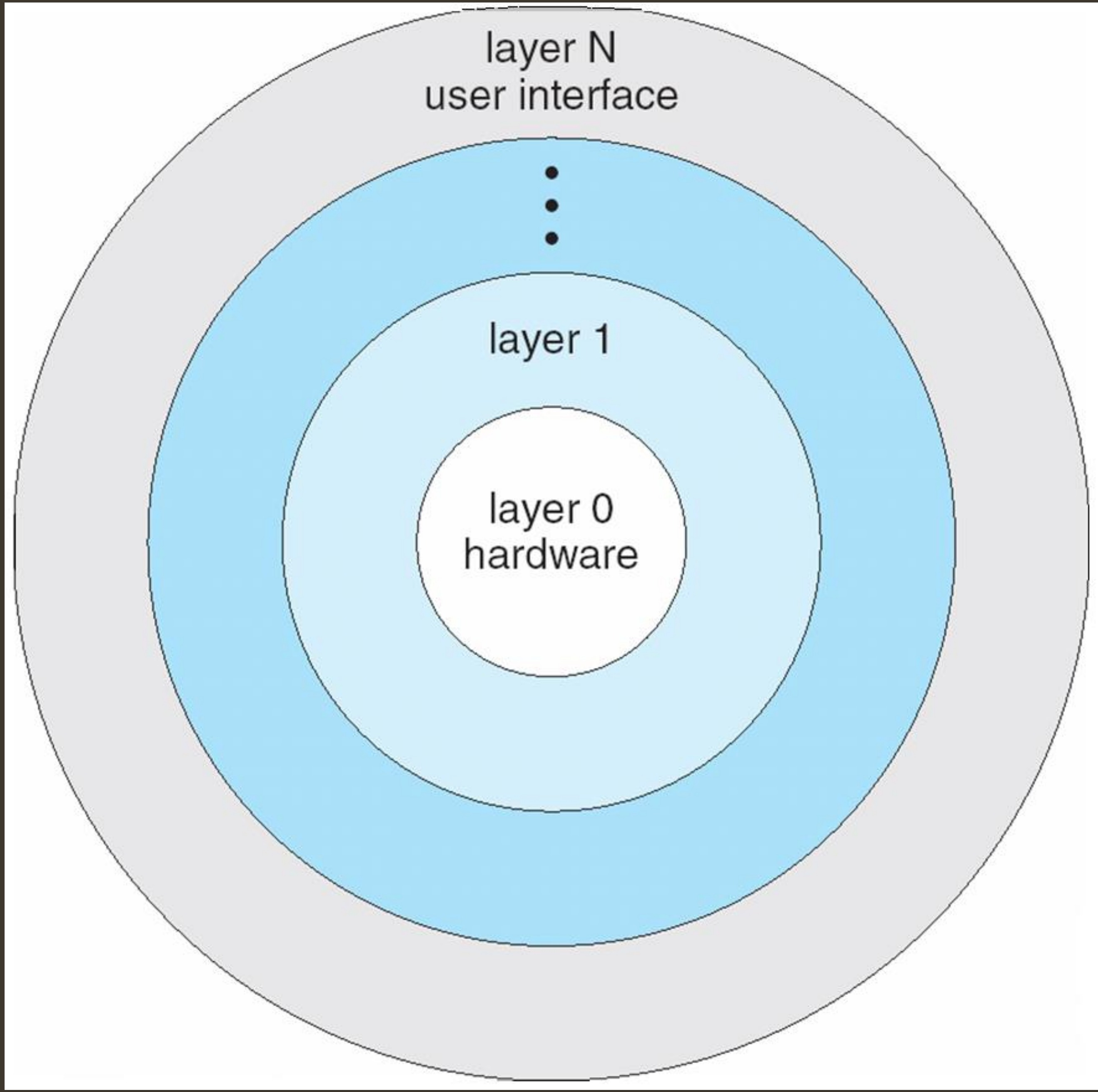
Traditional UNIX System Structure

Beyond simple but not fully layered



Layered Approach.

- OS **divided into a number of layers (levels)**, each built on top of lower layers
  - **Bottom layer (layer 0)** is *hardware*
  - **Highest (layer N)** is *user interface*
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers



Microkernel System Structure

- Moves as much from the kernel into user space (Example mach)
  - Mac OS X kernel (Darwin) partly based on Mach
- ***Communication*** takes place between user modules using **message passing**

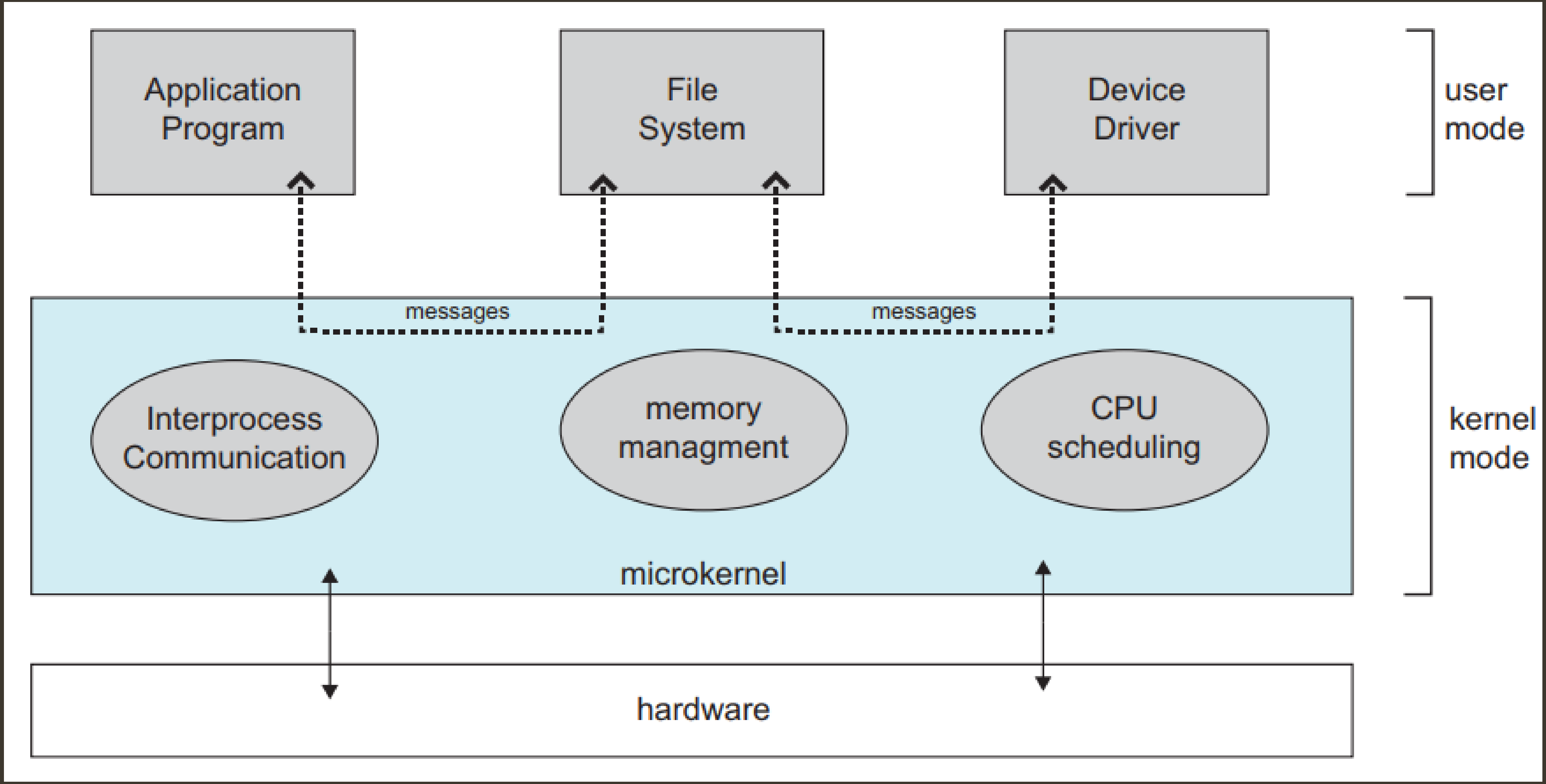
Benefits

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

Detriments:

Performance overhead of user space to kernel space communication

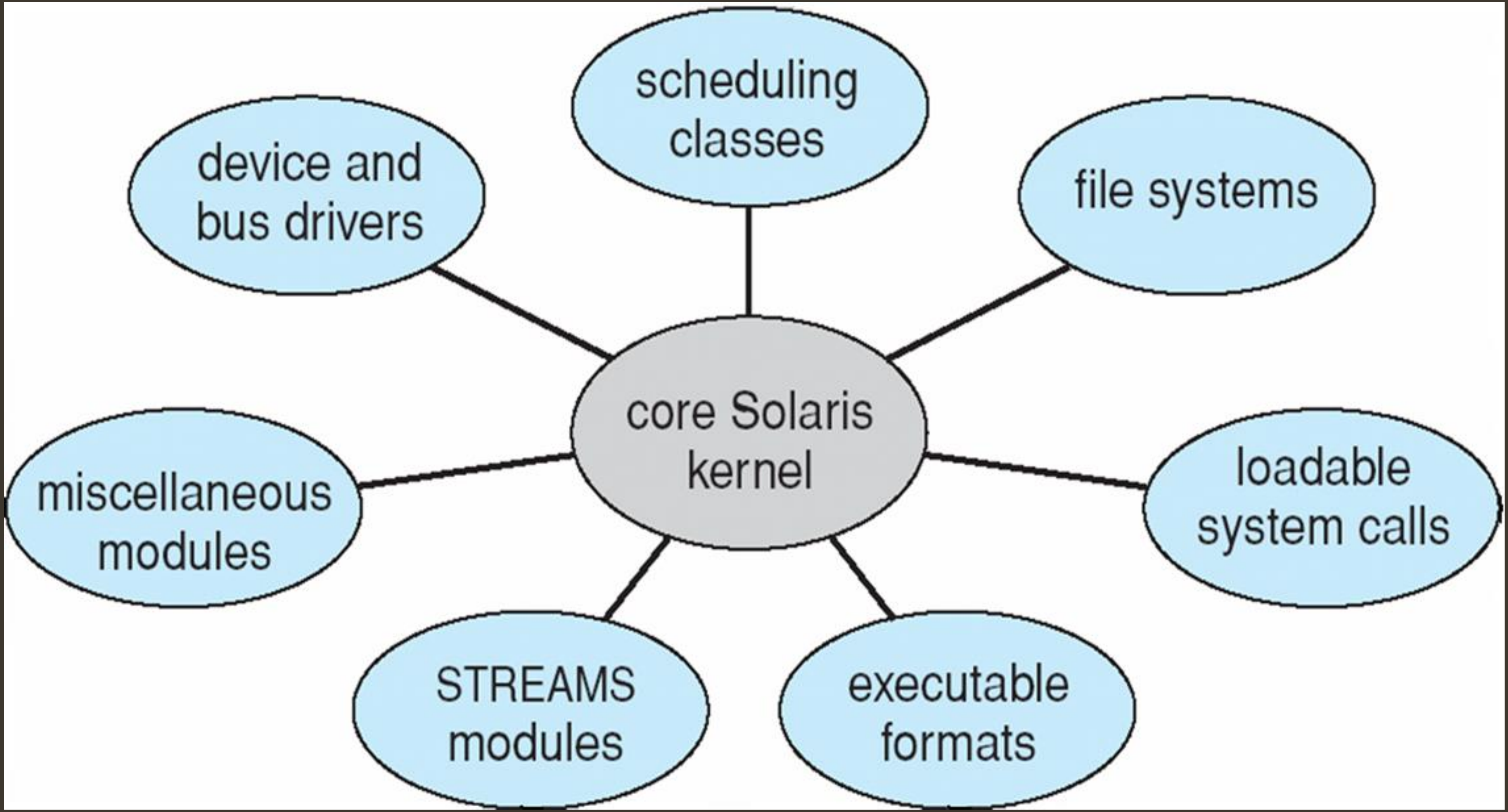
Microkernel System Structure



Modules.

- Many modern operating systems implement loadable kernel modules
  - Uses object-oriented approach
  - Each core component is separate
  - Each talks to the others over known interfaces
  - Each is loadable as needed within the kernel
- Overall, similar to layers but with more flexible
- Linux, Solaris, etc

Solaris Module Structure



Hybrid Systems

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

System Boot

- When power initialized on system, execution starts at a fixed memory location
    - **Firmware ROM** used to hold initial boot code
- OS must be made available to hardware so hardware can start it
    - Small piece of code – bootstrap loader, stored in ROM or EEPROM locates the kernel, loads it into memory, and starts it
    - Sometimes two-step process where boot block at fixed location loaded by ROM code, which loads bootstrap loader from disk
- Common bootstrap loader, GRUB, allows selection of kernel from multiple disks, versions, kernel options
- Kernel loads and system is then running

End of Chapter 2