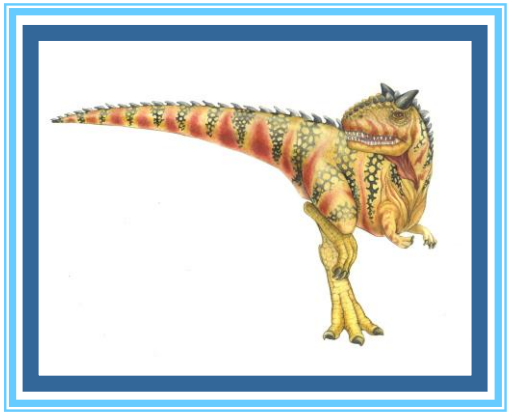
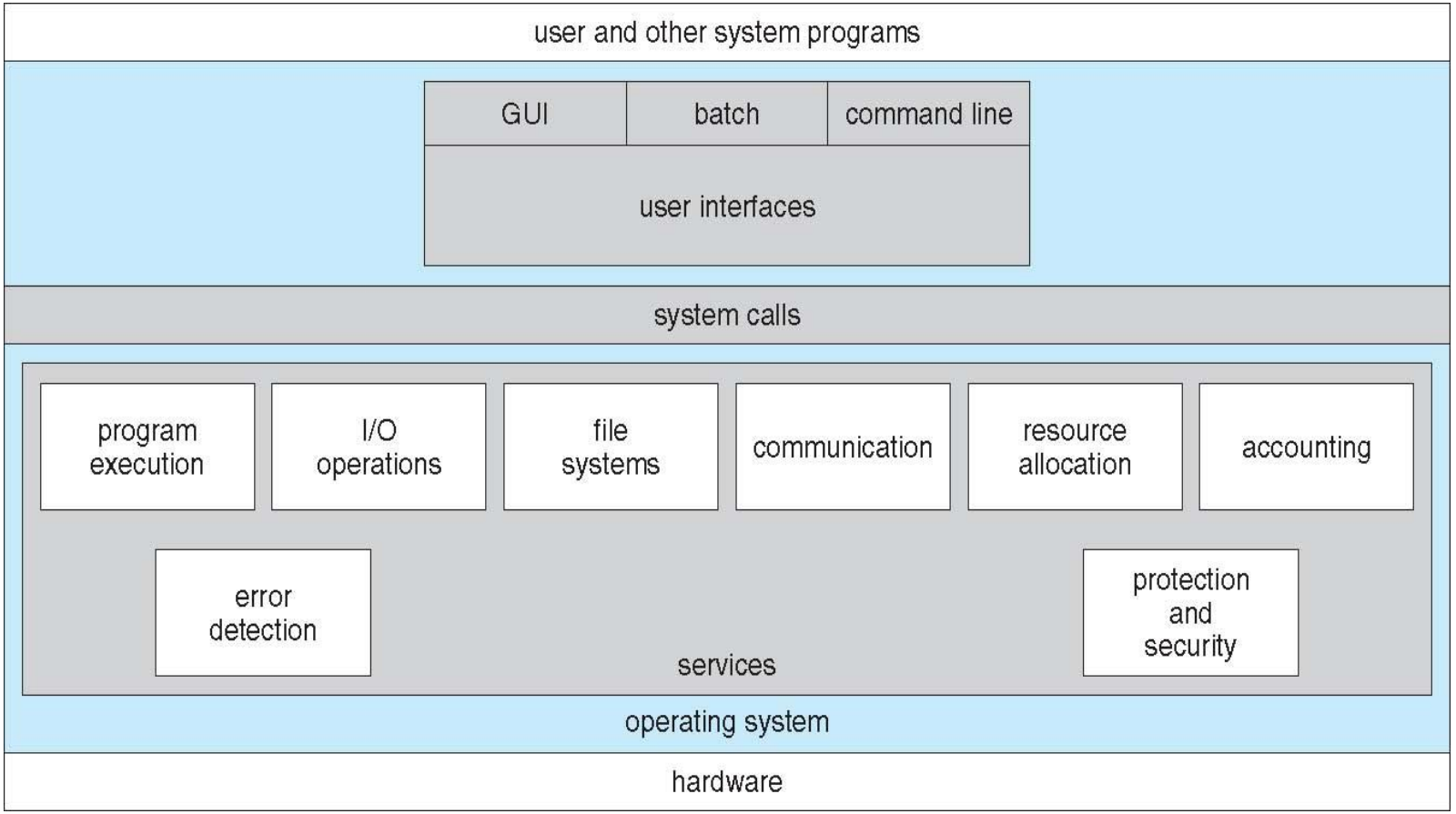
Chapter 2: System Structures

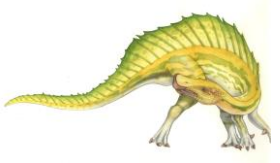


Chapter 2: System Structures

- Operating System Services
  - User Operating System Interface
  - System Calls
- Operating System Design and Implementation

A View of Operating System Services



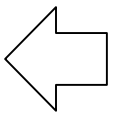


User Operating System Interface - CLI

- CLI or **command interpreter** allows direct command entry
  - Sometimes implemented in kernel, sometimes by systems program
  - Sometimes multiple CLIs are implemented – **shells**
  - Primarily fetches a command from user and executes it
    - commands built-in, or just names of programs
      - for latter, adding new features doesn't require shell modification

```
PBG-Mac-Pro:~ pbg$ ls
Applications                    Music                   WebEx
Applications (Parallels)        Pando Packages          config.log
Dropbox                         Thumbs.db               panda-dist
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
PBG-Mac-Pro:~ pbg$
```

Bourne Shell Command Interpreter



User Operating System Interface - GUI

- **Graphic User Interface (GUI)**
  - 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 is "Aqua" GUI interface with UNIX kernel underneath and shells available
  - Unix and Linux have CLI with optional GUI interfaces (CDE, KDE, GNOME)

User Operating System Interface - Touchscreen Interfaces

- **Touchscreen Interface**
  - Touchscreen devices require new interfaces because mouse not possible or not desired
  - Actions and selection based on gestures
  - Virtual keyboard for text entry

Chapter 2: System Structures

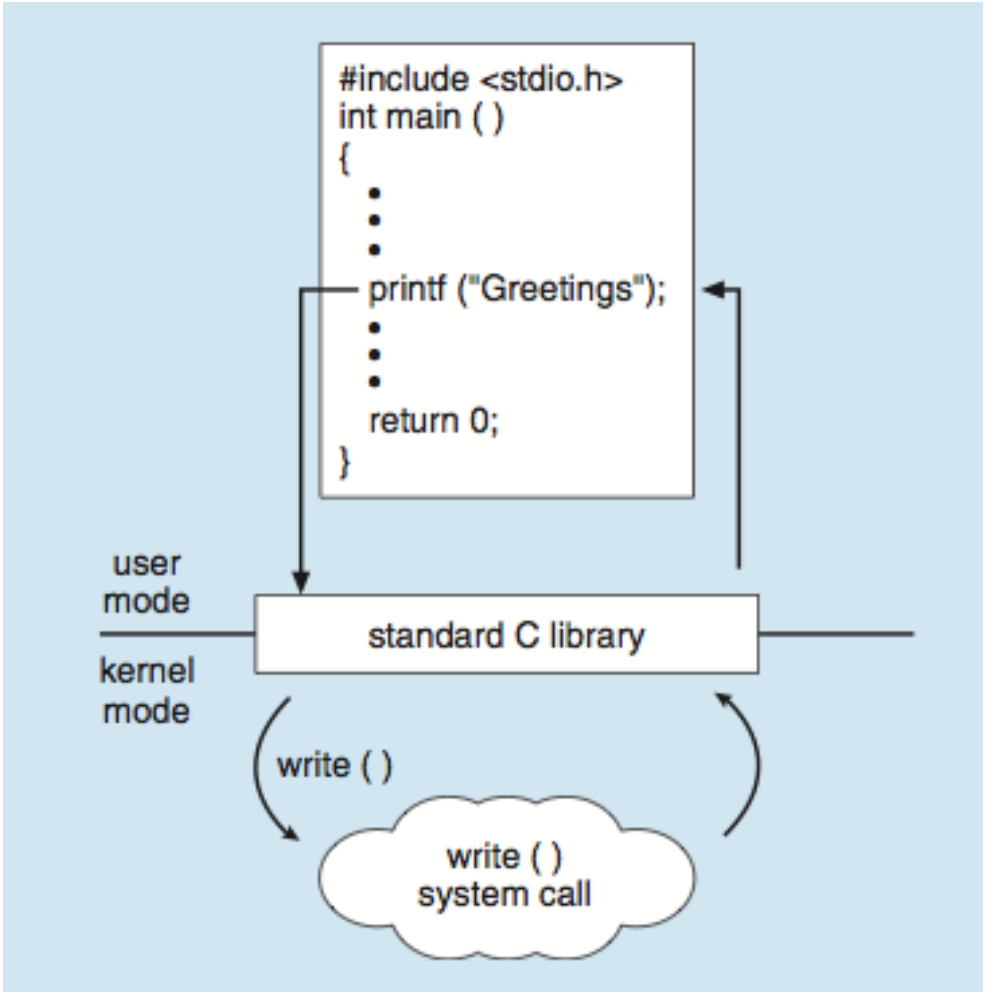
- Operating System Services
  - User Operating System Interface
  - System Calls
- Operating System Design and Implementation

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)

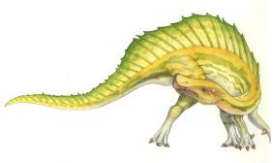
Standard C Library Example

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

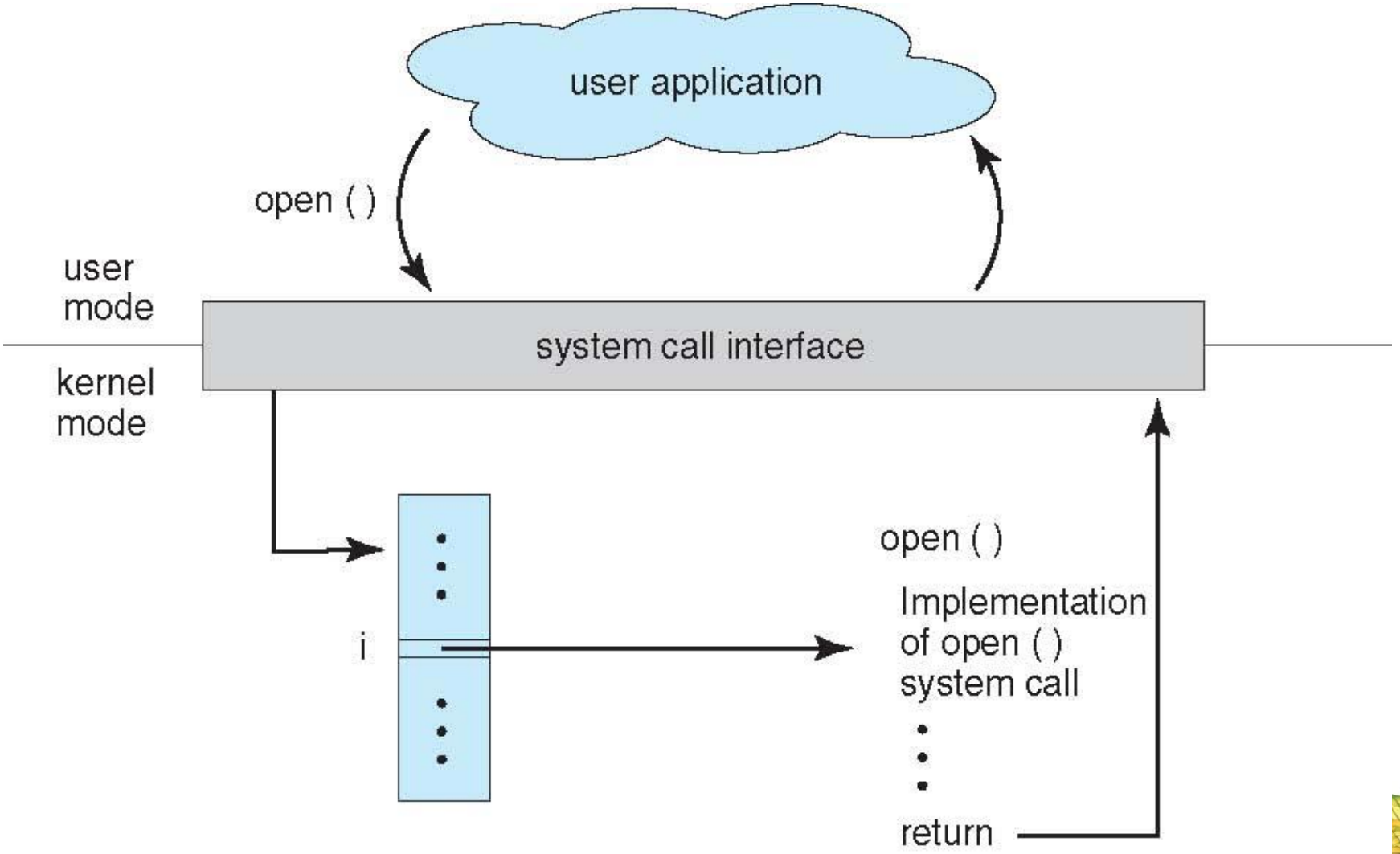


System Call Implementation

- 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 need 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

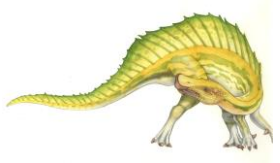


API – System Call – OS Relationship



System Call Parameter Passing

- Often, more information is required than simply identity of desired system call
  - required 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 **pushed**, onto the **stack** by the program and **popped** off the stack by the operating system
    - Some OS prefer the block or stack method because they do not limit the number or length of parameters being passed



Types of System Calls

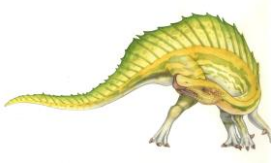
- **File management**
  - create file, delete file
  - open, close file
  - read, write, reposition
  - get and set file attributes
- **Device management**
  - request device, release device
  - read, write, reposition
  - get device attributes, set device attributes
  - logically attach or detach devices
- **Information maintenance**
  - get time or date, set time or date
  - get system data, set system data
  - get and set process, file, or device attributes
- **Protection**
  - Control access to resources
  - Get and set permissions
  - Allow and deny user access

Types of System Calls (Cont.)

- **Communications**
  - create, delete communication connection
  - Send, receive messages
    - **message passing model**
      - Information is exchanged by OS.
      - Useful when smaller numbers of data need to be exchanged.
      - Easier to implement than shared memory.
    - **shared-memory model**
      - exchange info. by reading and writing data in shared areas.
      - Fast and convenient.
      - The protection and synchronization problems in the shared area.
  - transfer status information
  - attach and detach remote devices

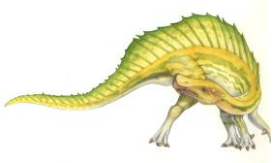
Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() ExitProcess() WaitForSingleObject()	fork() exit() wait()
File Manipulation	CreateFile() ReadFile() WriteFile() CloseHandle()	open() read() write() close()
Device Manipulation	SetConsoleMode() ReadConsole() WriteConsole()	ioctl() read() write()
Information Maintenance	GetCurrentProcessID() SetTimer() Sleep()	getpid() alarm() sleep()
Communication	CreatePipe() CreateFileMapping() MapViewOfFile()	pipe() shmget() mmap()
Protection	SetFileSecurity() InitlializeSecurityDescriptor() SetSecurityDescriptorGroup()	chmod() umask() chown()



Chapter 2: System Structures

- Operating System Services
  - User Operating System Interface
  - System Calls
- Operating System Design and Implementation



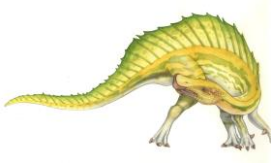
Operating System Design and Implementation

- 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

- Specifying and designing OS is highly creative task of **software engineering**

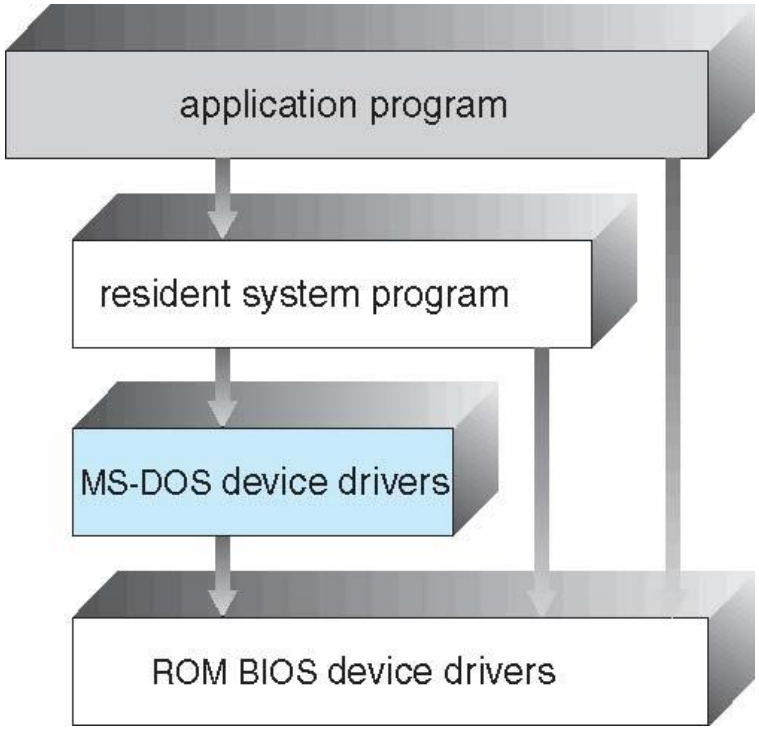


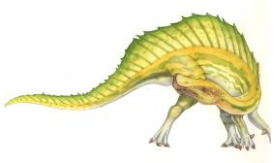
Implementation

- Much variation
  - Early OSes in assembly language
  - Now C, C++
- Actually usually a mix of languages
  - Lowest levels in assembly
  - Main body in C
  - Systems programs in C, C++, scripting languages like PERL, Python, shell scripts
- More high-level language easier to **port** to other hardware
  - But slower
- General-purpose OS is very large program - various ways to structure one.

Operating System Structure – Simple Structure

- I.e. 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

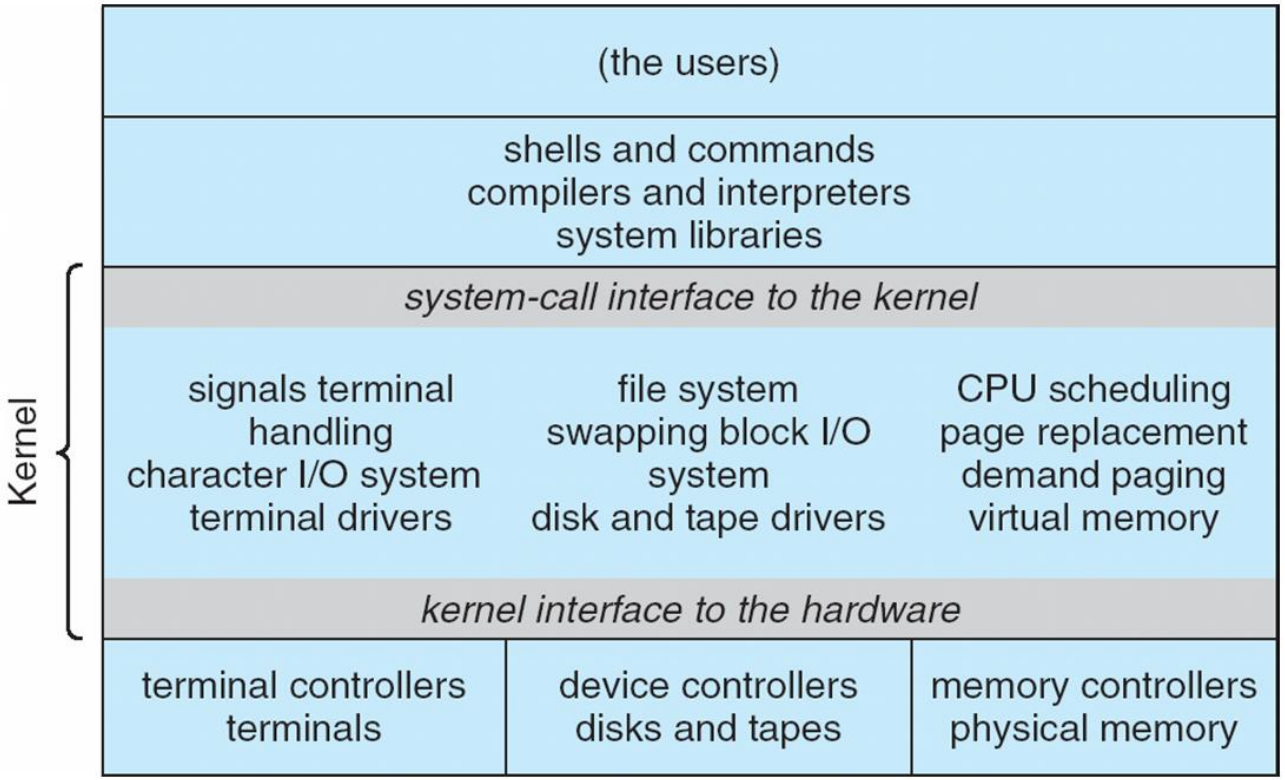




UNIX

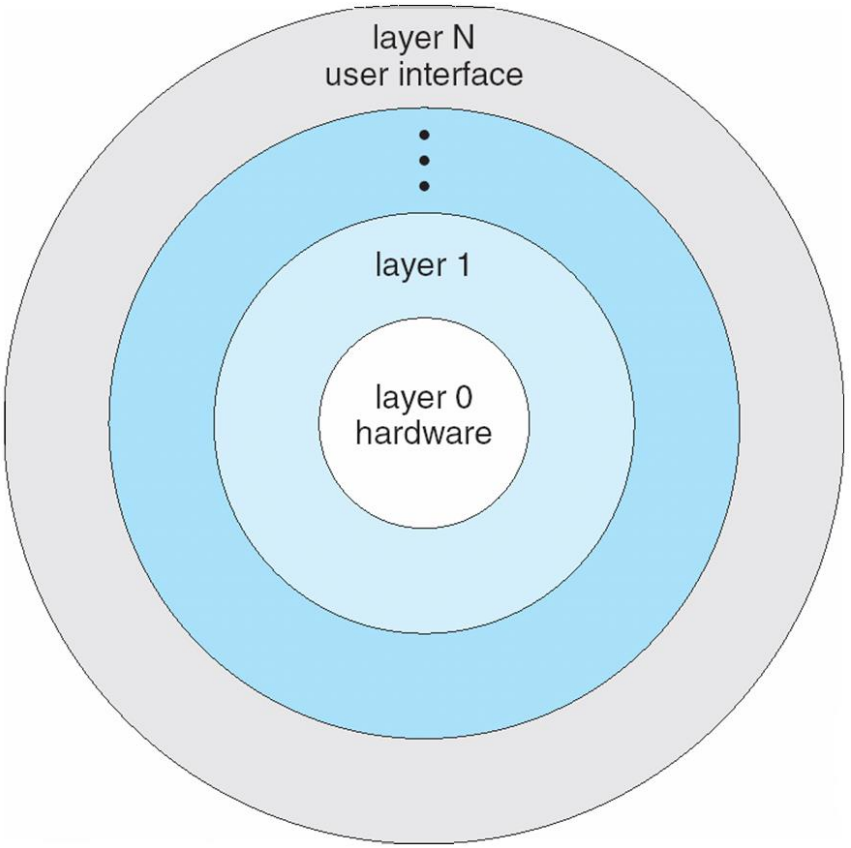
- UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.
- The UNIX OS consists of two separable parts
  - Systems programs
  - The kernel

Beyond simple but not fully layered



Operating System 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.

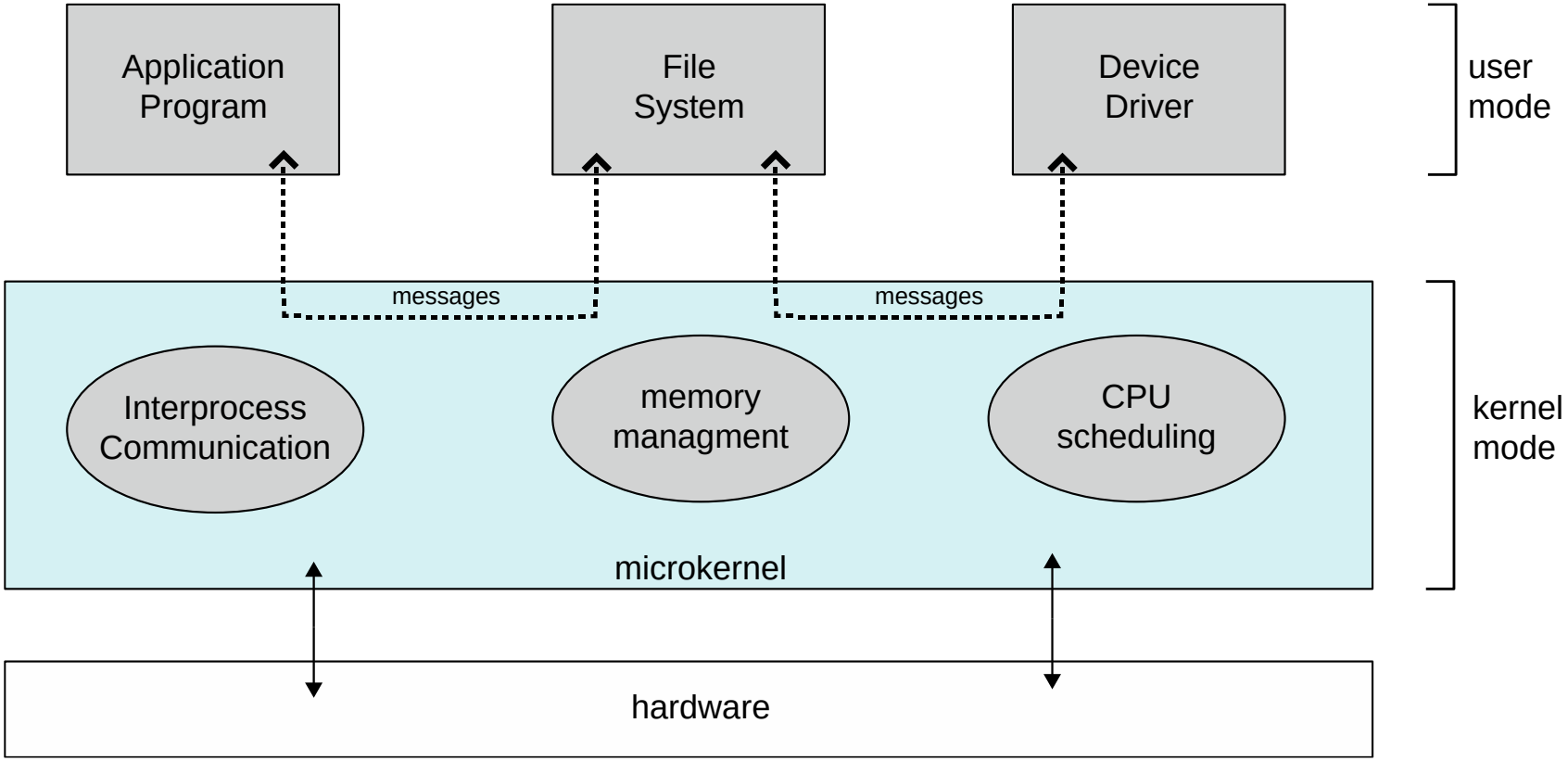


- With modularity, layers are selected such that each uses functions and services of only lower-level layers
- Advantage: easy to debug
- Not easy to appropriately define layers and the layer approach is less efficient sometimes

Operating System Structure – Microkernel

- Moves as much from the kernel into user space
- **Mach:** example of **microkernel**
  - 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 is running in kernel mode)
  - More secure
- Detriments:
  - Performance overhead of user to kernel space communication

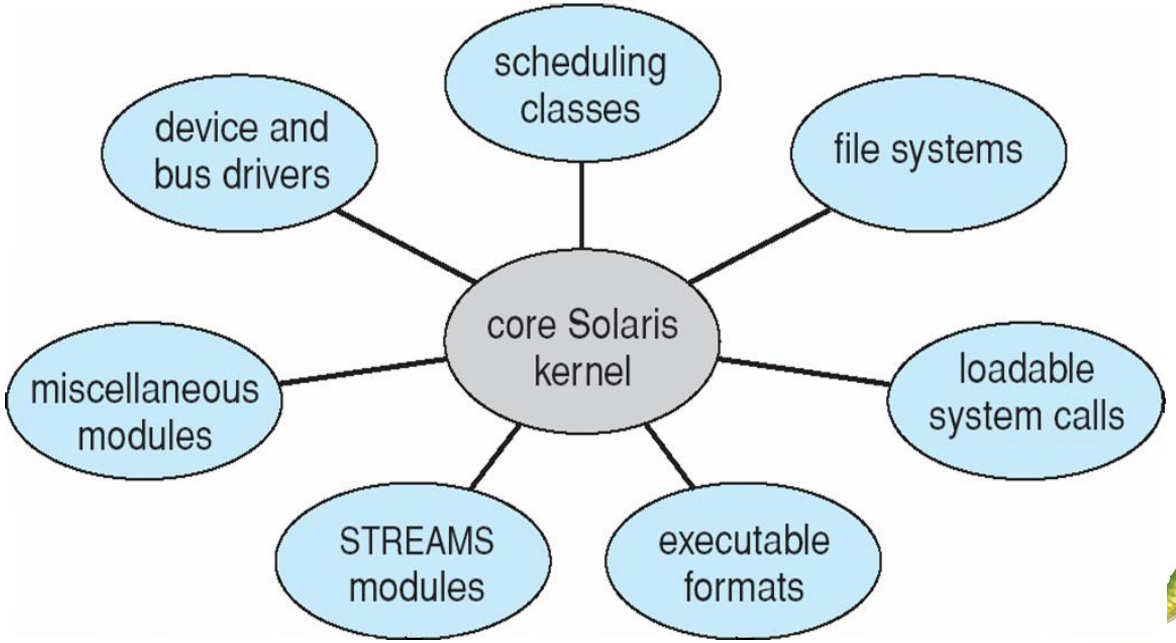
Operating System Structure – Microkernel



Operating System Structure – Modules

- Most 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

Solaris loadable modules



Operating System Structure – Hybrid Systems

- Most modern operating systems 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 adding of new functionality
  - Windows mostly monolithic, plus microkernel and providing separate subsystems that run as user-ode process
- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of Mach microkernel and BSD Unix parts, plus I/O kit and dynamically loadable modules (called **kernel extensions**)

Hybrid System Example – Mac OS X