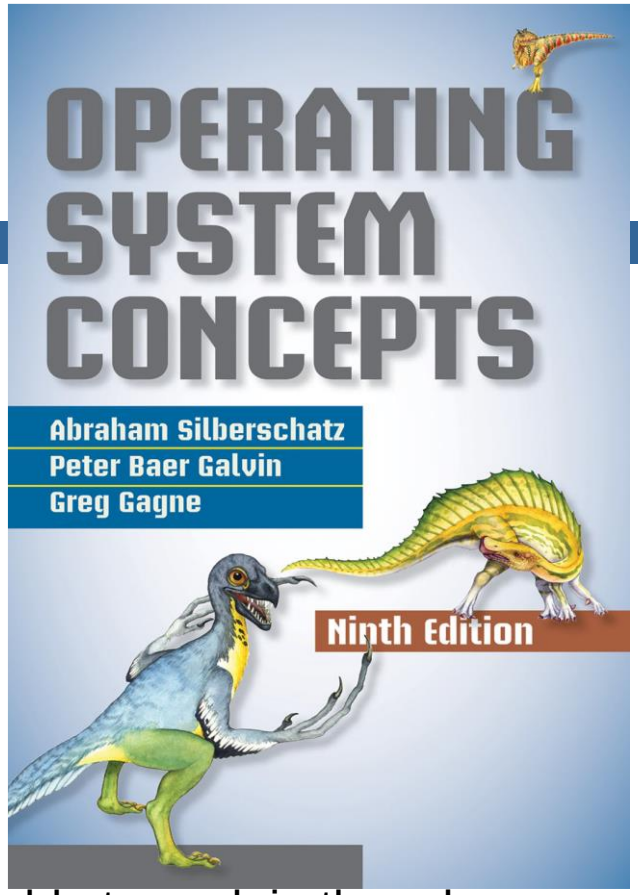
Module 1
Chapter 1 and 2
http://www.os-book.com

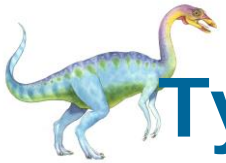


CO: Students will be able to explain the relevance, structure and functions of OS in computing devices.

Silberschatz, Galvin and Gagne ©2013

What is an Operating System?

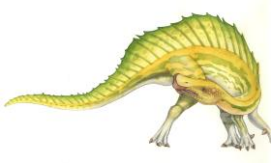
- A program that acts as an intermediary between a user of a computer and the computer hardware
- An operating system is a software which performs all the basic tasks like
  - file management
  - memory management
  - process management
  - handling input and output
  - controlling peripheral devices such as disk drives and printers.
- Some popular Operating Systems include Linux, Windows, OS X, VMS, OS/400, AIX, z/OS, etc.



Types of modern operating systems

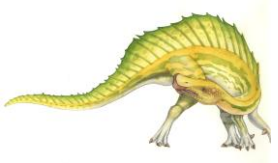
- Mainframe operating systems: MVS
- Server operating systems: FreeBSD, Solaris
- Multiprocessor operating systems: Cellular IRIX
- Personal computer operating systems: Windows, Unix
- Real-time operating systems: VxWorks
- Embedded operating systems
- Smart card operating systems

⇒ Some operating systems can fit into more than one category



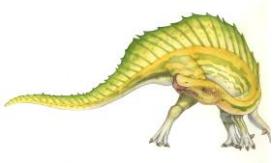
1.2 What Is an Operating System?

- Separates applications from the hardware they access
  - Software layer
  - Manages software and hardware to produce desired results
- Operating systems primarily are resource managers
  - Hardware
    - Processors
    - Memory
    - Input/output devices
    - Communication devices
  - Software applications



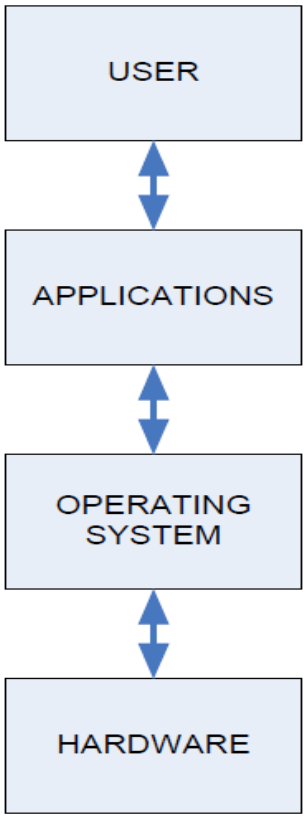
What is an Operating System?

- An operating system is a program that acts as an interface between the user and the computer hardware and controls the execution of all kinds of programs.
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner



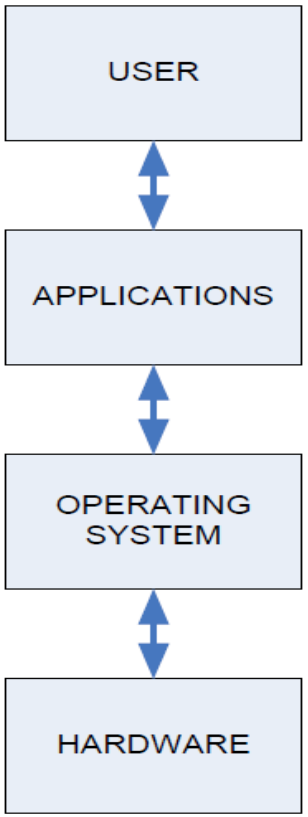
The Structure of Computer Systems

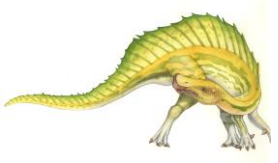
- Accessing computer resources is divided into *layers*.
- Each layer is isolated and only interacts directly with the layer below or above it.
- If we install a new hardware device
  - ✓ No need to change anything about the user/applications.
  - ✓ However, you do need to make changes to the operating system.
  - ✓ You need to install the device drivers that the os will use to control the new device.



The Structure of Computer Systems

- If we install a new software application
  - ✓No need to make any changes to your hardware.
  - ✓But we need to make sure the application is supported by the operating system
  - ✓user will need to learn how to use the new application.
- If we change the operating system
  - ✓Need to make sure that both applications and hardware will compatible with the new os

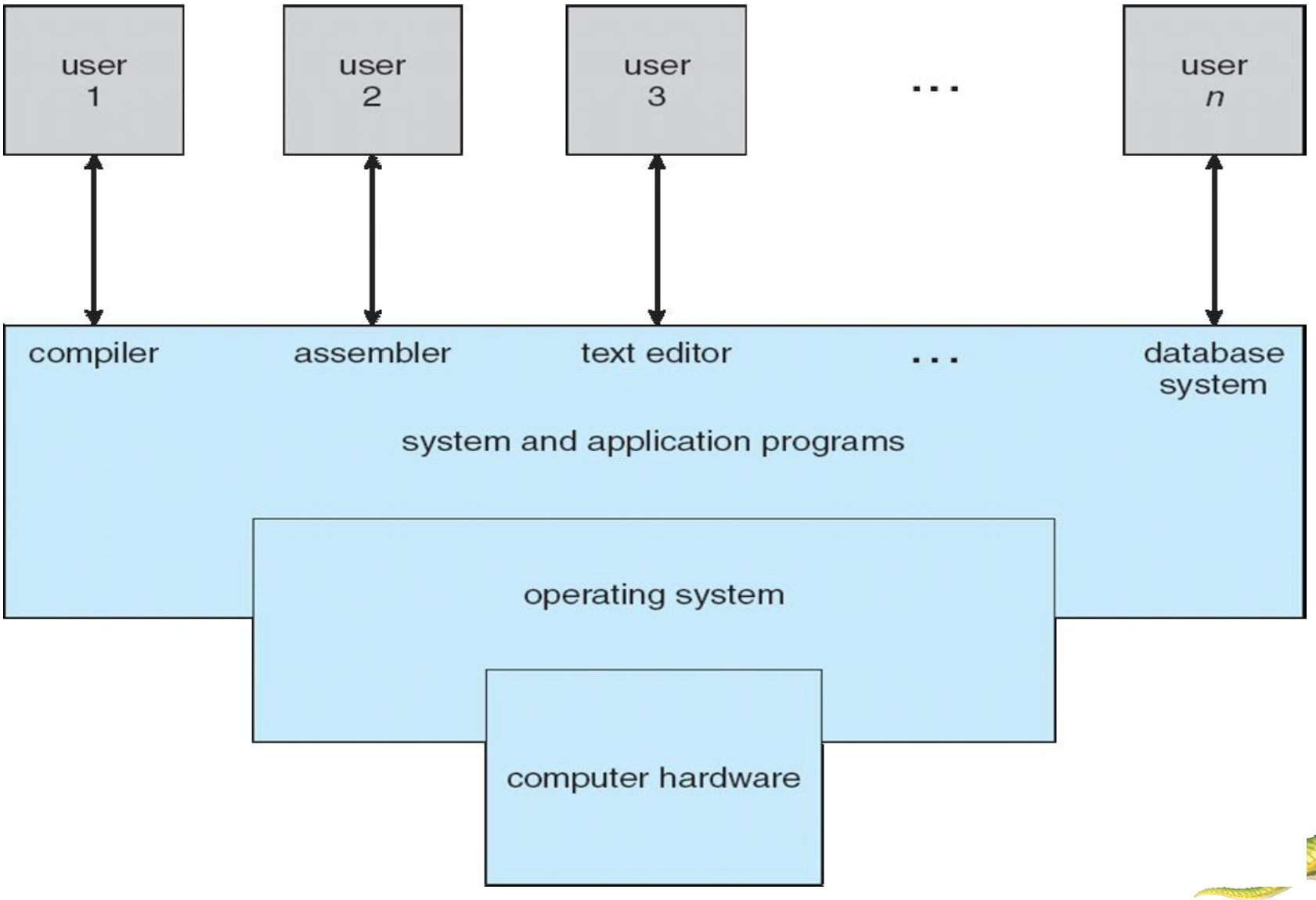


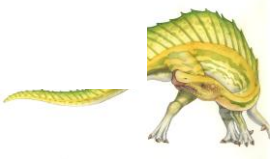


Computer System Structure

- Computer system can be divided into four components:
  - Hardware – provides basic computing resources
    - CPU, memory, I/O devices
  - Operating system
    - Controls and coordinates use of hardware among various applications and users
  - Application programs – define the ways in which the system resources are used to solve the computing problems of the users
    - Word processors, compilers, web browsers, database systems, video games
  - Users
    - People, machines, other computers

Four Components of a Computer System





What Operating Systems Do

- Depends on the point of view

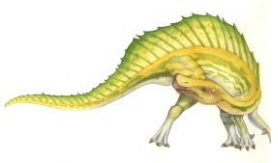
User view

- Users want convenience, **ease of use** and **good performance**
  - Don't care about **resource utilization**
- But shared computer such as **mainframe** or **minicomputer** must keep all users happy
- Users of dedicate systems such as **workstations** have dedicated resources but frequently use shared resources from **servers**
- Handheld computers are resource poor, optimized for usability and battery life
- Some computers have little or no user interface, such as embedded computers in devices and automobiles

Operating System Definition

System view

- OS is a **resource allocator**
  - Manages all resources
  - Decides between conflicting requests for efficient and fair resource use
- OS is a **control program**
  - Controls execution of programs to prevent errors and improper use of the computer



Operating System Definition (Cont.)

- No universally accepted definition
- "Everything a vendor ships when you order an operating system" is a good approximation
  - But varies wildly
- "The one program running at all times on the computer" is the **kernel**.
- Everything else is either
  - a system program (ships with the operating system) , or
  - an application program.

Important functions of an OS

- Memory Management
- Processor Management
- Device Management
- File Management
- Security
- Control over system performance
- Job accounting
- Error detecting aids
- Coordination between other software and users

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
  - Optimizing CPU utilization and computer response to users
- 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

Processor Management

- In multiprogramming environment, the OS decides which process gets the processor when and for how much time. This function is called process scheduling.
- An Operating System does the following activities for processor management −
  - Keeps tracks of processor and status of process. The program responsible for this task is known as traffic controller.
  - Allocates the processor (CPU) to a process.
  - De-allocates processor when a process is no longer required.

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

Process Management Activities

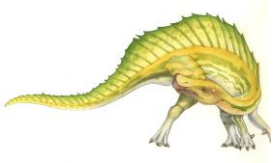
- Multi-threaded process has one program counter per thread
- Typically system has many processes, some user, some operating system running concurrently on one or more CPUs
  - Concurrency by multiplexing the CPUs among the processes / threads
- OS is responsible for the following activities in connection with process management:
  - Creating and deleting both user and system processes
  - Suspending and resuming processes
  - Providing mechanisms for process synchronization
  - Providing mechanisms for process communication
  - Providing mechanisms for deadlock handling

Device Management

An Operating System manages device communication via their respective drivers.

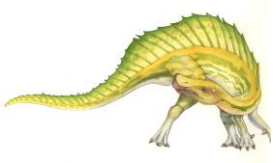
It does the following activities for device management −

- Keeps tracks of all devices.
  - Program responsible for this task is known as the I/O controller.
- Decides which process gets the device when and for how much time.
- Allocates the device in the efficient way.
- De-allocates devices.



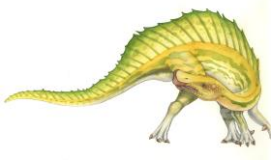
Storage Management

- OS provides uniform, logical view of information storage
  - Abstracts physical properties to logical storage unit - **file**
  - Each medium is controlled by device (i.e., disk drive, tape drive)
  - Varying properties include
    - access speed
    - Capacity
    - data-transfer rate
    - access method (sequential or random)



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

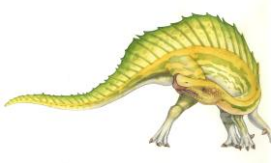


Mass-Storage Management

- Usually disks used to store data that does not fit in main memory or data that must be kept for a “long” period of time
- Proper management is of central importance
- Entire speed of computer operation hinges on disk subsystem and its algorithms
- OS activities
  - Free-space management
  - Storage allocation
  - Disk scheduling
- Some storage need not be fast
  - Tertiary storage includes optical storage, magnetic tape
  - Still must be managed – by OS or applications
  - Varies between WORM (write-once, read-many-times) and RW (read-write)

I/O Subsystem

- One purpose of OS is to hide peculiarities of hardware devices from the user
- I/O subsystem responsible for
  - Memory management of I/O including
    - buffering (storing data temporarily while it is being transferred)
    - caching (storing parts of data in faster storage for performance)
    - spooling (the overlapping of output of one job with input of other jobs)
  - General device-driver interface
  - Drivers for specific hardware devices

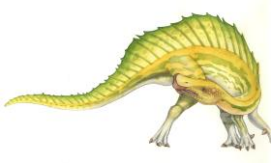


Protection and Security

- **Protection** – any mechanism for controlling access of processes or users to resources defined by the OS
- **Security** – defense of the system against internal and external attacks
  - Huge range, including denial-of-service, worms, viruses, identity theft, theft of service
- Systems generally first distinguish among users, to determine who can do what
  - User identities (**user IDs**, security IDs) include name and associated number, one per user
  - User ID then associated with all files, processes of that user to determine access control
  - Group identifier (**group ID**) allows set of users to be defined and controls managed, then also associated with each process, file
  - **Privilege escalation** allows user to change to effective ID with more rights

Operating System Services

- Operating systems provide an environment for execution of programs and services to programs and users
- 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)



Operating System Services

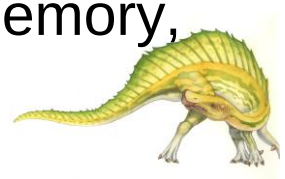
- **I/O Operation**: A running program may require I/O. This I/O may involve a file or a I/O device for specific device. Some special function can be desired. Therefore the operating system must provide a means to do I/O.
- **File System Manipulation**: Programs need to read and write files and directories, create and delete them, search them, list file Information, permission management. Therefore the operating system must maintain each and every files correctly.
- **Accounting** - To keep track of which users use how much and what kinds of computer resources

Operating System Services

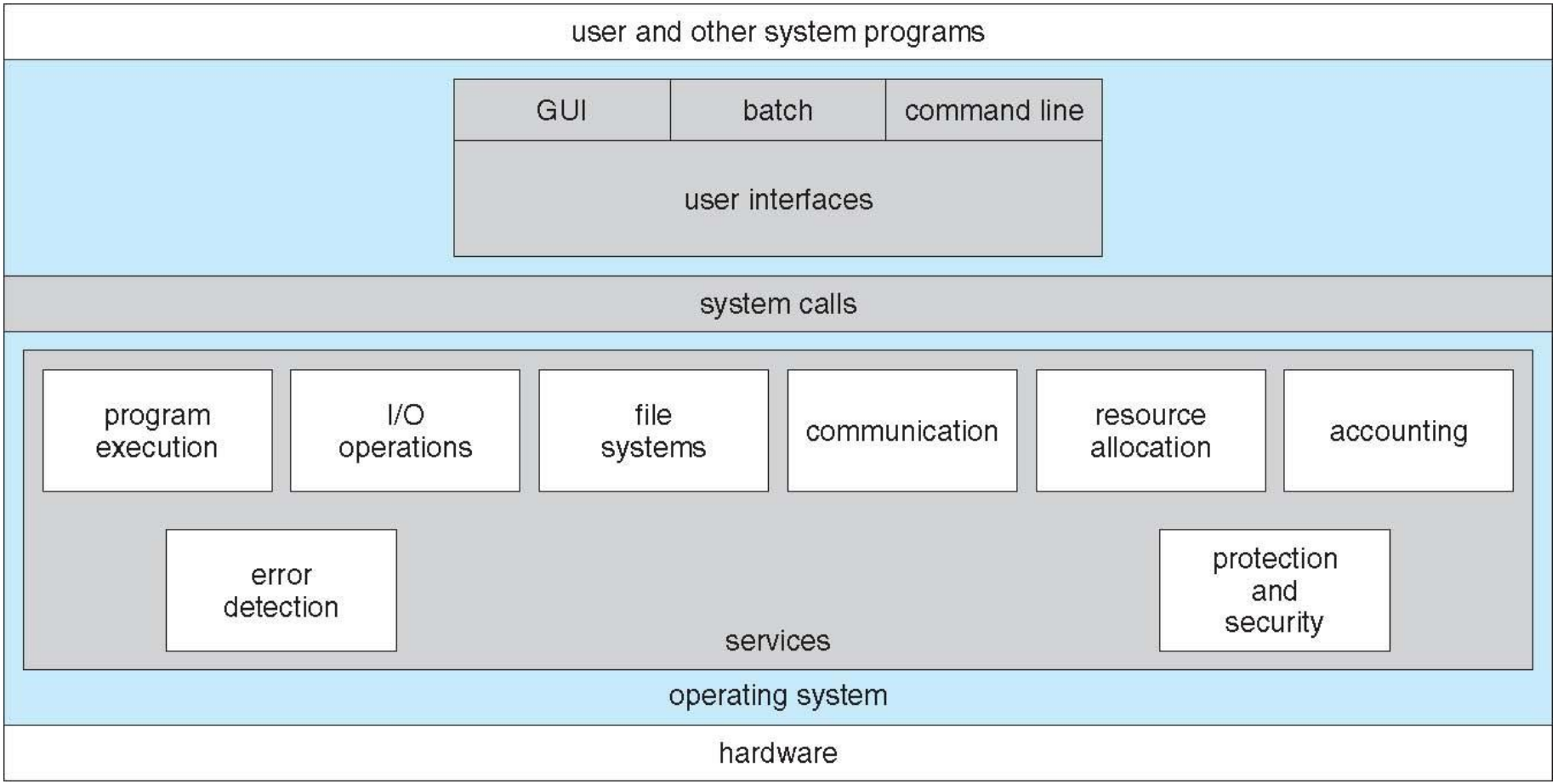
- **Communication**: Processes may exchange information, on the same computer or between computers over a network
  - Implemented via shared memory or by the technique of message passing in which packets of information are moved between the processes by the operating system.
- **Error detection**: OS 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

- **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
- 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 - CPU cycles, main memory, file storage, I/O devices.

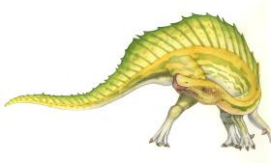


A View of Operating System Services



Operating-System Operations

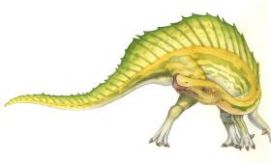
- **Interrupt driven** (hardware and software)
  - Hardware interrupt by one of the devices
  - Software interrupt (**exception** or **trap):**
    - Software error (e.g., division by zero)
    - Request for operating system service
    - Other process problems include infinite loop, processes modifying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



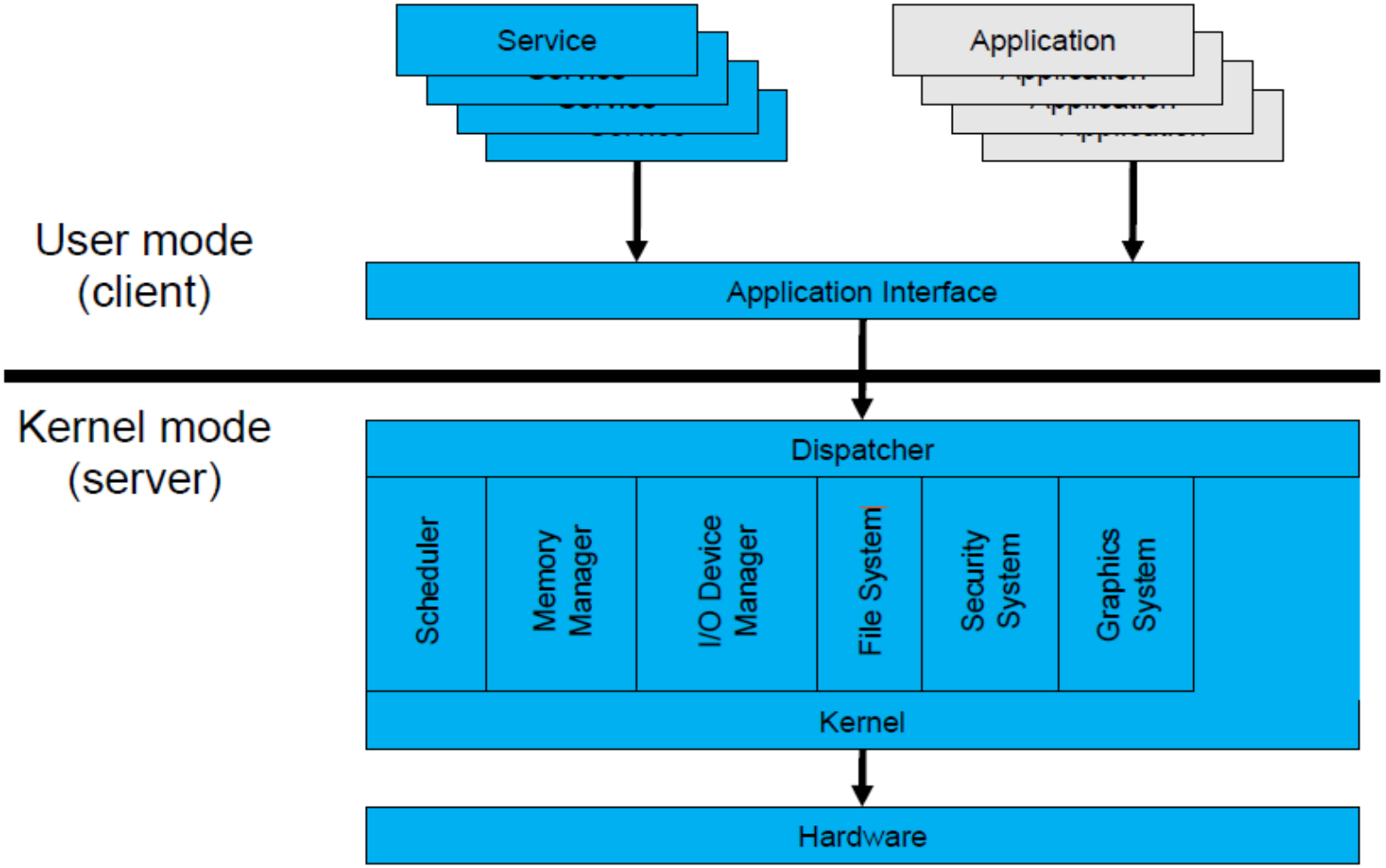
Operating System Mode

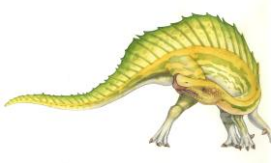
- *User Mode* is concerned with the actual interface between the user and the system.
  - It controls things like running applications and accessing files.

- *Kernel Mode* is concerned with everything running in the background.
  - It controls things like accessing system resources, controlling hardware functions and processing program instructions.
- System calls are used to change mode from User to Kernel.



Operating System Mode





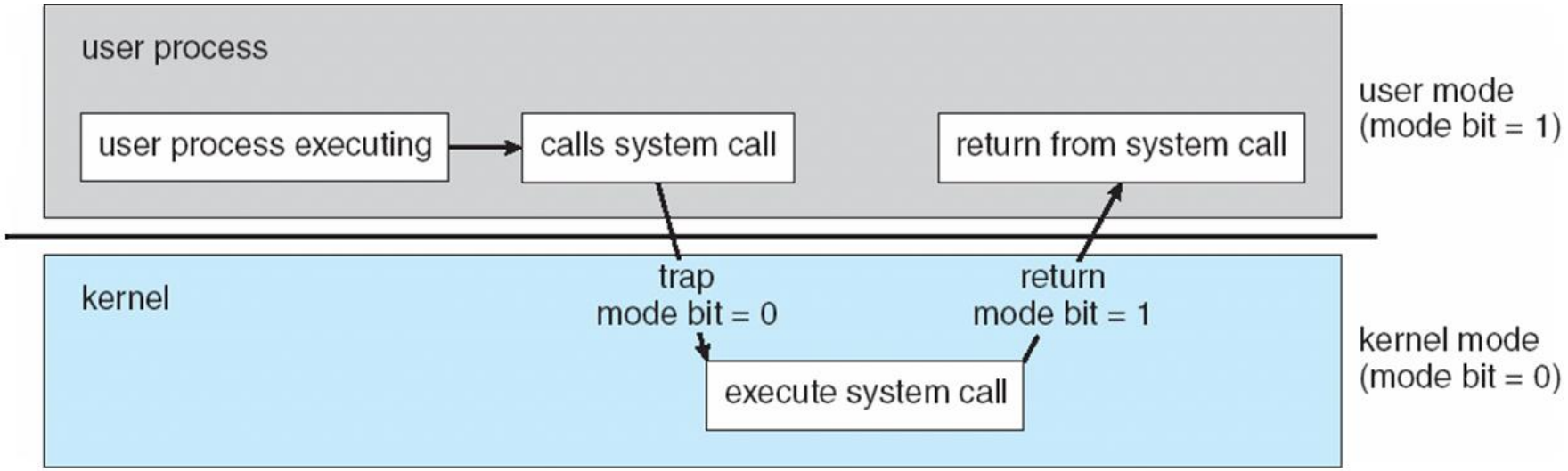
Operating-System Operations (cont.)

- **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
  - i.e. **virtual machine manager** (**VMM**) mode for guest **VMs**

Transition from User to Kernel Mode

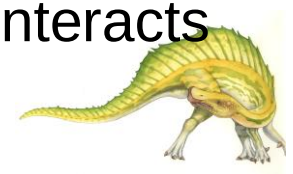
- Timer to prevent infinite loop / process hogging resources
  - Timer is set to interrupt the computer after some time period
  - Keep a counter that is decremented by the physical clock.
  - Operating system set the counter (privileged instruction)
  - When counter zero generate an interrupt
  - Set up before scheduling process to regain control or terminate program that exceeds allotted time

Transition from User to Kernel Mode



Kernel

- Kernel is a software code that reside in central core of OS. It has complete control over system.
- When operation system boots, kernel is first part of OS to load in main memory.
- Kernel remains in main memory for entire duration of computer session. The kernel code is usually loaded in to protected area of memory.
- Kernel performs it's task like executing processes and handling interrupts in kernel space.
- User performs it's task in user area of memory.
- This memory separation is made in order to prevent user data and kernel data from interfering with each other.
- Kernel does not interact directly with user, but it interacts using SHELL and other programs and hardware.



Kernel cont…

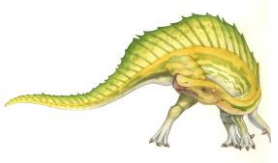
- Kernel includes:-

  1. Scheduler: It allocates the Kernel's processing time to various processes.

  2. Supervisor: It grants permission to use computer system resources to each process.

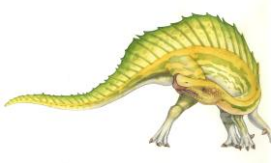
  3. Interrupt handler : It handles all requests from the various hardware devices which compete for kernel services.

  4. Memory manager : allocates space in memory for all users of kernel service.



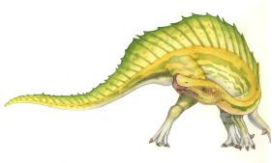
System Call

- Kernel provides services for process management, file management, I/O management, memory management.
- System calls are used to provide these type of services.
- **System call** is the programmatic way in which a computer program/user application requests a service from the kernel of the operating system on which it is executed.
- Application program is just a user-process. Due to security reasons , user applications are not given access to privileged resources(the ones controlled by OS).



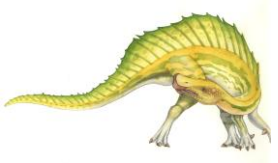
System Call

- When they need to **do any I/O** or have **some more memory** or **spawn a process** or wait for **signal/interrupt,** it requests operating system to facilitate all these. This **request is made through System Call.**
- System calls are also called **software-interrupts.**



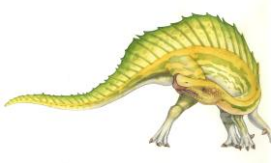
System Calls

- 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
  - Win32 API for Windows
  - POSIX API for POSIX-based systems (including virtually all versions of UNIX, Linux, and Mac OS X)
  - Java API for the Java virtual machine (JVM)



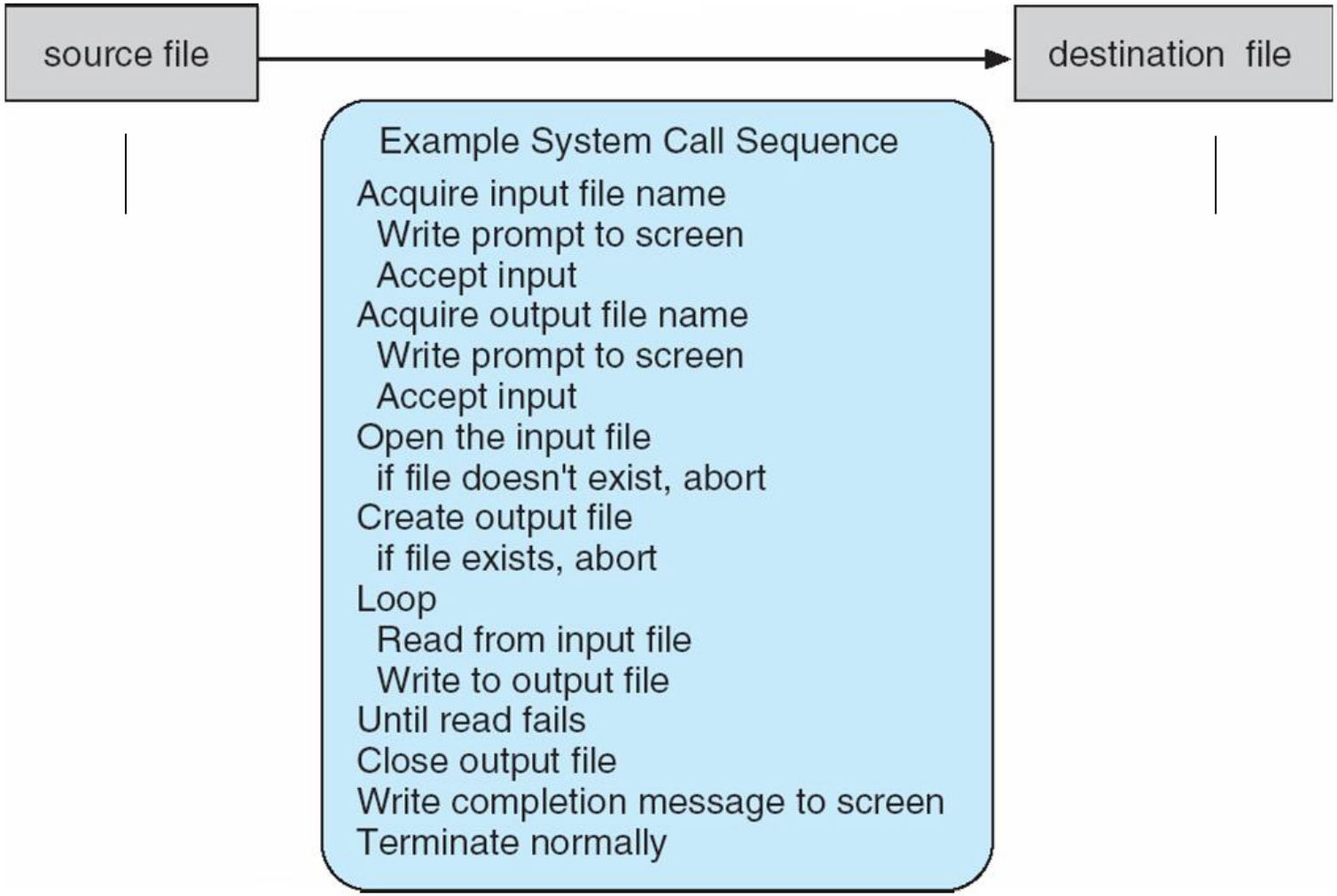
Why do we need System calls in OS

- Reading and writing files demand system calls
- System calls are required if the file system want to create or delete a file
- Used for creation and management of new process
- Network connections need system calls for sending and receiving packets
- Access to hardware devices like scanner. Printer, need a system call



Example of System Calls

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



EXAMPLE OF STANDARD API

As an example of a standard API, consider the `read()` function that is available in UNIX and Linux systems. The API for this function is obtained from the `man` page by invoking the command

```
man read
```

on the command line. A description of this API appears below:

```
#include <unistd.h>

ssize_t     read(int fd, void *buf, size_t count)
```

return value — function name — parameters

A program that uses the `read()` function must include the `unistd.h` header file, as this file defines the `ssize_t` and `size_t` data types (among other things). The parameters passed to `read()` are as follows:

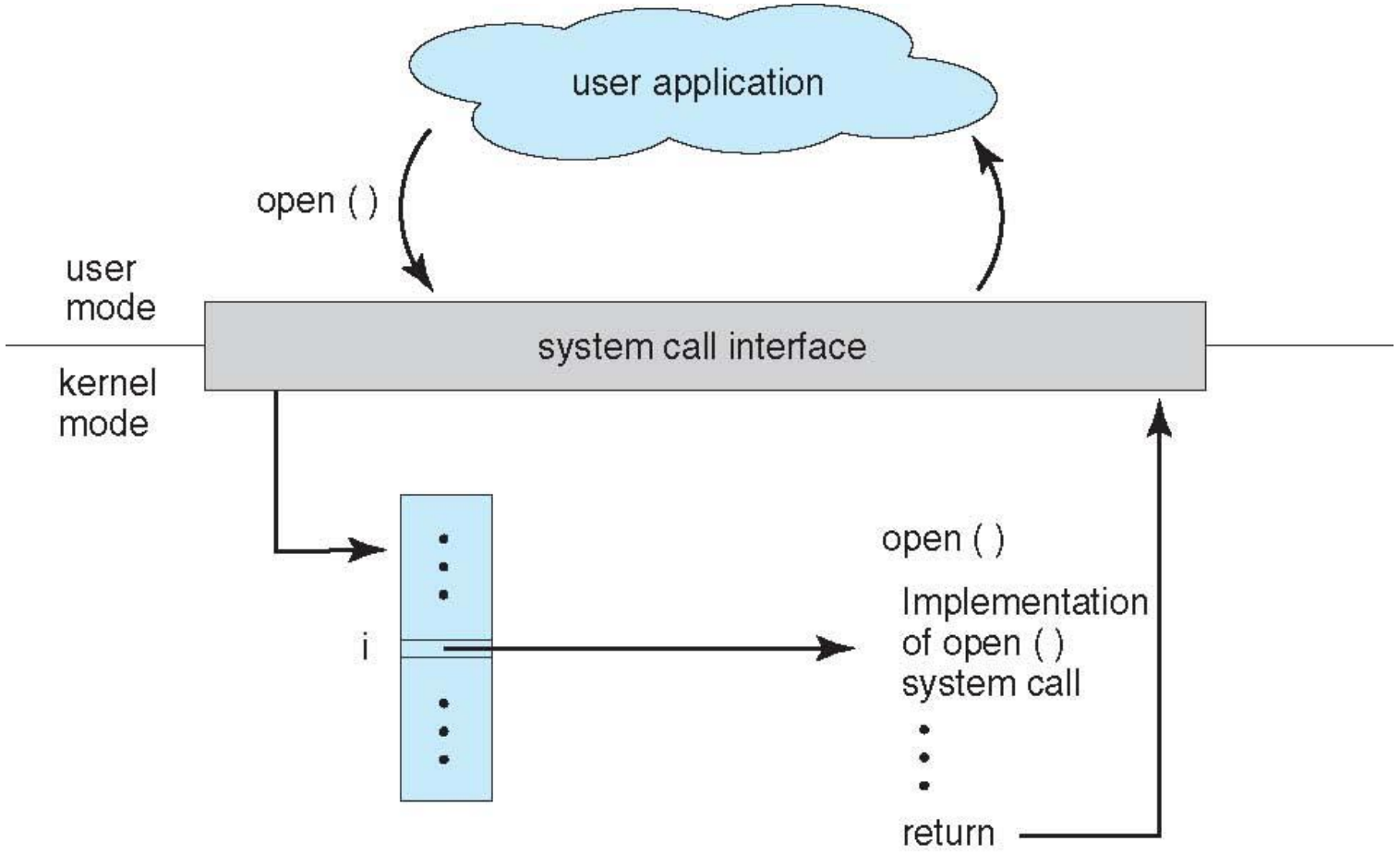
- `int fd`—the file descriptor to be read
- `void *buf`—a buffer where the data will be read into
- `size_t count`—the maximum number of bytes to be read into the buffer

On a successful read, the number of bytes read is returned. A return value of 0 indicates end of file. If an error occurs, `read()` returns −1.

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
  - 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)

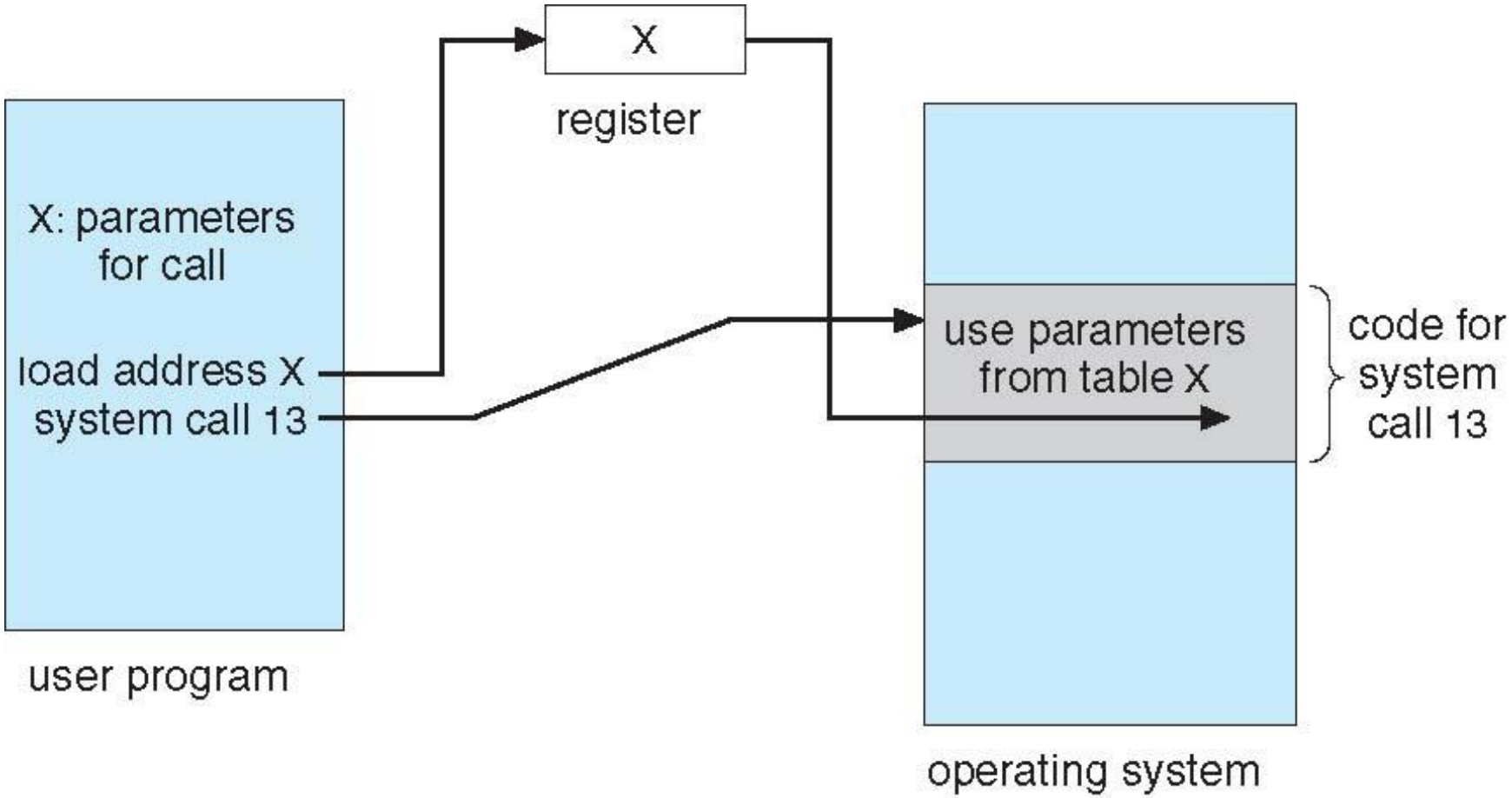
API – System Call – OS Relationship



System Call Parameter Passing

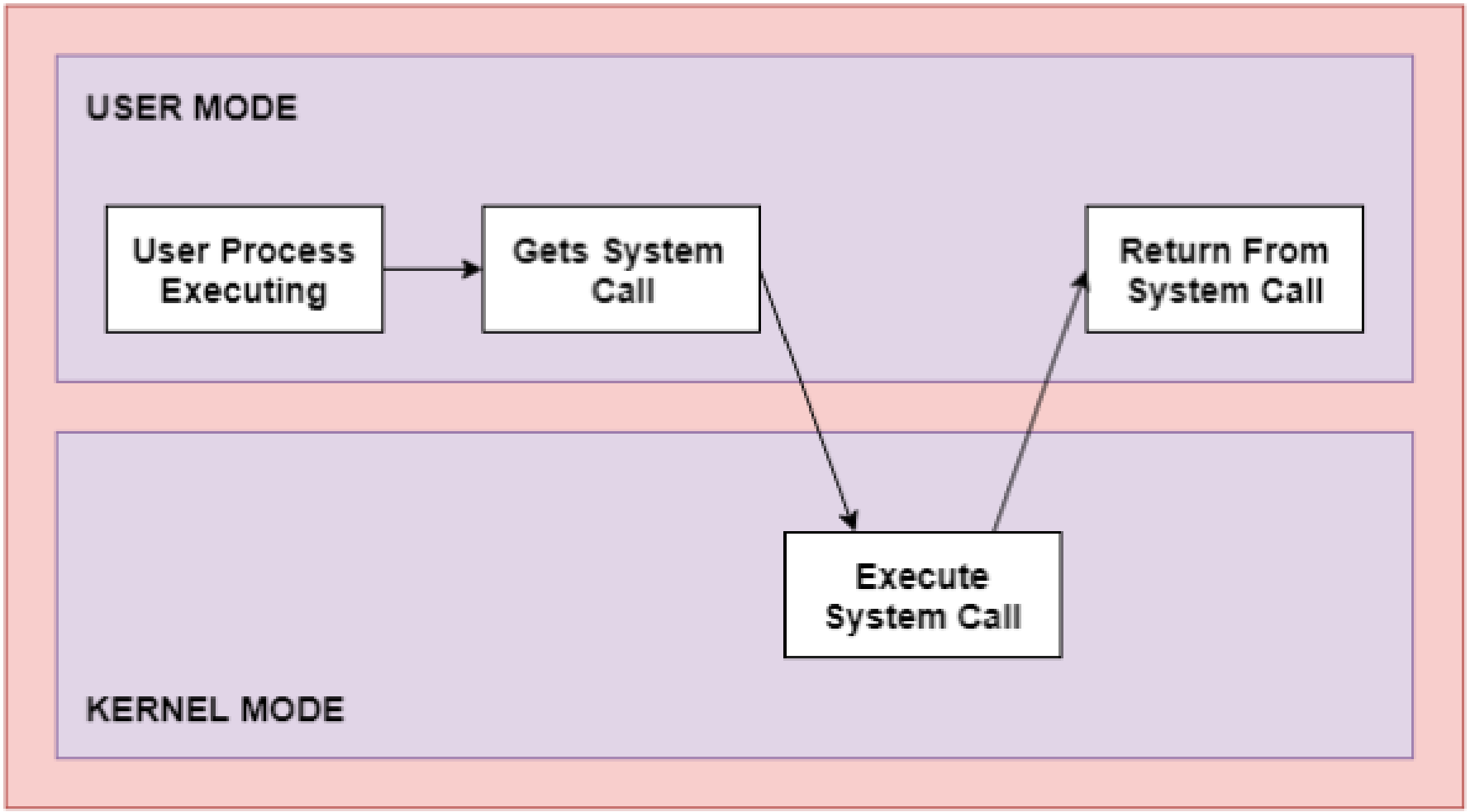
- Often, more information is required than simply identity of desired system call
  - Exact type and amount of information vary according to OS and call
- Three general methods used to pass parameters to the OS
  - Simplest: pass the parameters in registers
    - In some cases, may be more parameters 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



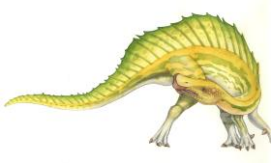
System Call

- **System call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on.
- System call is a way for programs to **interact with OS**.
- A computer program makes a system call when it makes a request to the operating system's kernel.
- System call **provides** the services of the operating system to the user programs via Application Program Interface(API).
- It provides an interface between a process and OS
- System calls are the only entry points into the kernel system


USER MODE
User Process Executing
Gets System Call
Return From System Call
Execute System Call
KERNEL MODE

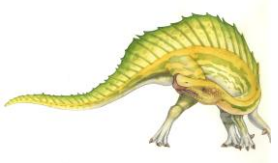
Services Provided by System Calls

- Process creation and management
- Main memory management
- File Access, Directory and File system management
- Device handling(I/O)
- Protection
- Networking



Types of System call

- Process control
- File manipulation
- Device manipulation
- Information maintenance
- Communications
- Protection



Types of System Calls

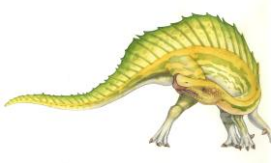
- Process control
  - create process, terminate process
  - end, abort
  - load, execute
  - get process attributes, set process attributes
  - wait for time
  - wait event, signal event
  - allocate and free memory
  - Dump memory if error
  - **Debugger** for determining **bugs, single step** execution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

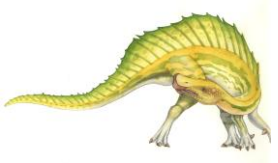
Types of System Calls

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



Types of System Calls (Cont.)

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

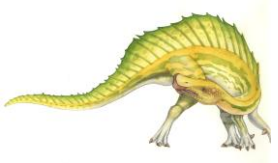


Types of System Calls (Cont.)

- 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

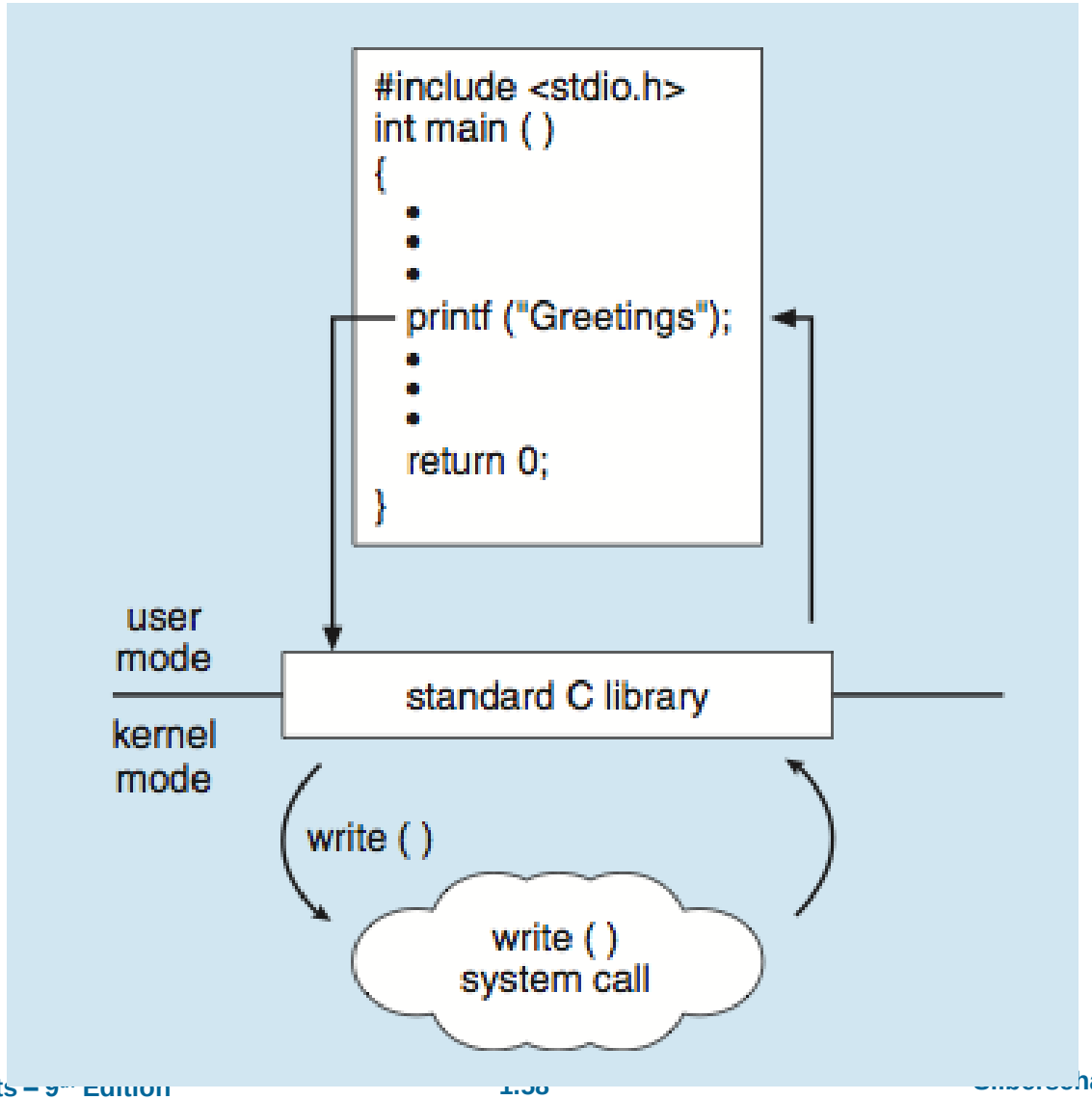


Examples of Windows and Unix System Calls

	Windows	Unix
Process Control	CreateProcess() 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()

Standard C Library Example

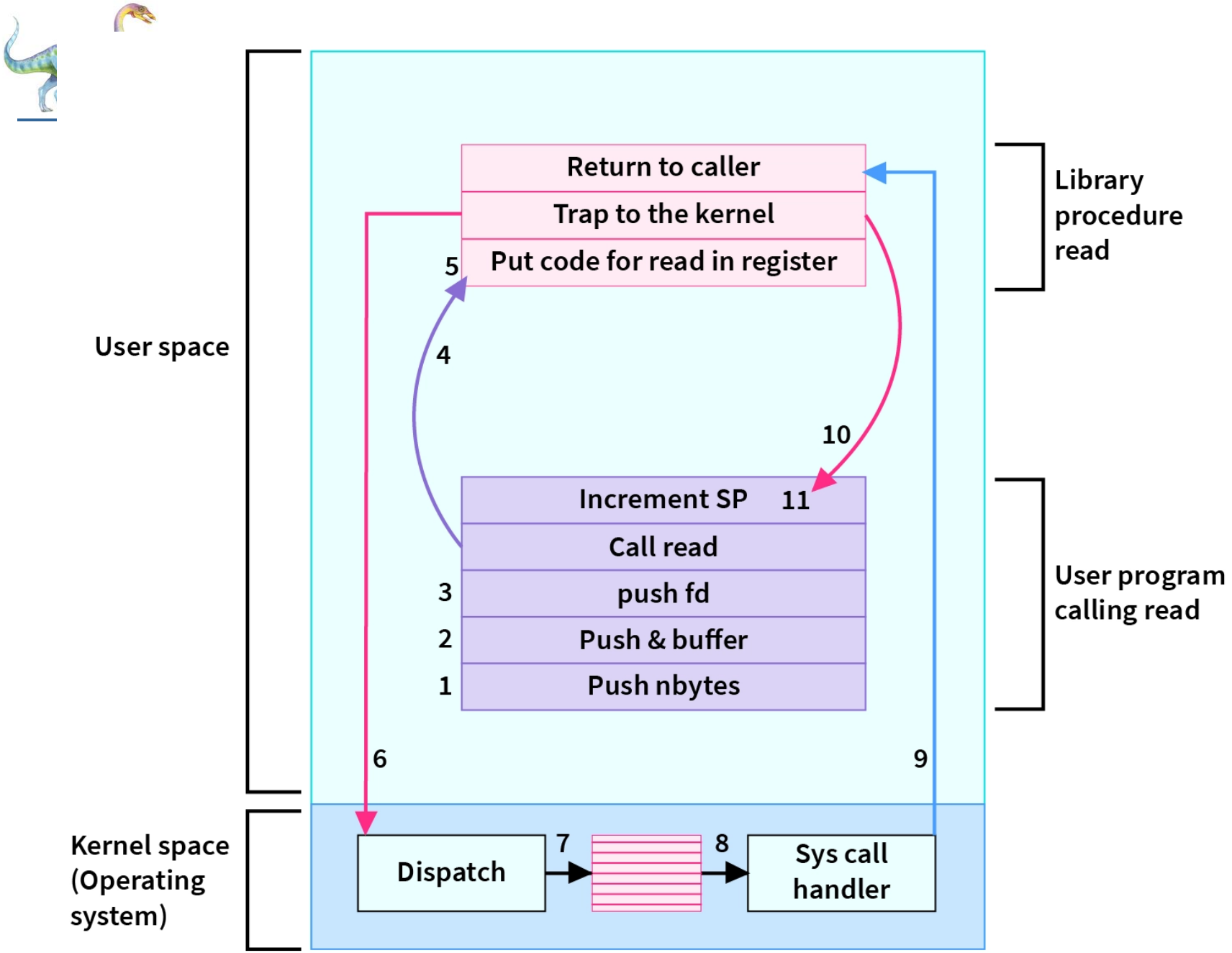
- C program invoking printf() library call, which calls write() system call



System call examples

- open()
- The open() system call is used to provide access to a file in a file system.
- This system call allocates resources to the file and provides a handle that the process uses to refer to the file.
- A file can be opened by multiple processes at the same time or be restricted to one process.
- It all depends on the file organisation and file system

- **read()**
- The read() system call is used to access data from a file that is stored in the file system.
- The file to read can be identified by its file descriptor and it should be opened using open() before it can be read.
- In general, the read() system calls takes three arguments i.e. the file descriptor, buffer which stores read data and number of bytes to be read from the file


User space
Kernel space (Operating system)
Return to caller
Trap to the kernel
Put code for read in register
5
4
10
Increment SP
11
Call read
3
push fd
2
Push & buffer
1
Push nbytes
6
9
Dispatch
7
8
Sys call handler
Library procedure read
User program calling read

- Initially, the process pushes the parameter onto the stack.
- After that, the read routine from the library is called, which puts the system call code (unique code for each system call is assigned) into the register.
- Trap instruction is used which causes the transfer of control from user mode to kernel mode.
- Kernel reads the system call code present in the register and finds out that the system call is read, so the system call handler executes the read system call.
- After the execution of the read system call, the transfer is given back to the user mode.
- The library routine in user mode returns the output to the process which called it.
- On receiving the output, the process continues to execute the next instruction.

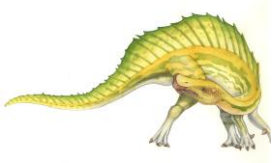
- **write()**
- The write() system calls writes the data from a user buffer into a device such as a file.
- This system call is one of the ways to output data from a program.
- In general, the write system calls takes three arguments i.e. file descriptor, pointer to the buffer where data is stored and number of bytes to write from the buffer.

- **close()**
- The close() system call is used to terminate access to a file system.
- Using this system call means that the file is no longer required by the program and so the buffers are flushed, the file metadata is updated and the file resources are de-allocated.

- **wait()**
- In some systems, a process needs to wait for another process to complete its execution.
- This type of situation occurs when a parent process creates a child process, and the execution of the parent process remains suspended until its child process executes.
- The suspension of the parent process automatically occurs with a wait() system call.
- When the child process ends execution, the control moves back to the parent process

- **fork()**
- Processes use this system call to create processes that are a copy of themselves.
- With the help of this system Call parent process creates a child process
- **kill():**
- The kill() system call is used by OS to send a termination signal to a process that urges the process to exit.

- **exec()**
- This system call runs when an executable file in the context of an already running process that replaces the older executable file.
- However, the original process identifier remains as a new process is not built, but stack, data, head etc. are replaced by the new process.



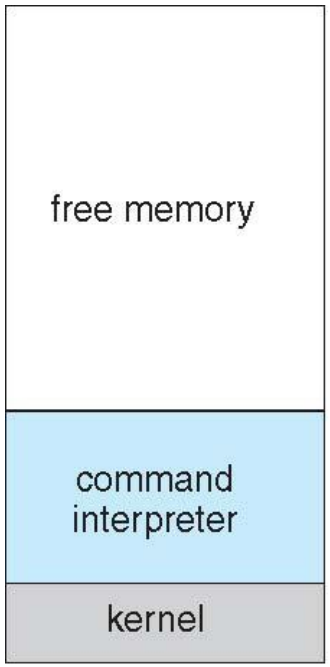
- **exit():**
- The exit() system call is used to terminate program execution.
- Specially in the multi-threaded environment, this call defines that the thread execution is complete.
- The OS reclaims resources that were used by the process after the use of exit() system call.

Rules for passing parameters in System call

- The floating-point parameters can’t be passed as a parameter in the system call.
- Only a limited number of arguments can be passed in the system call.
- If there are more arguments, then they should be stored in the block of memory and the address of that memory block is stored in the register.
- Parameters can be pushed and popped from the stack only by the operating system.

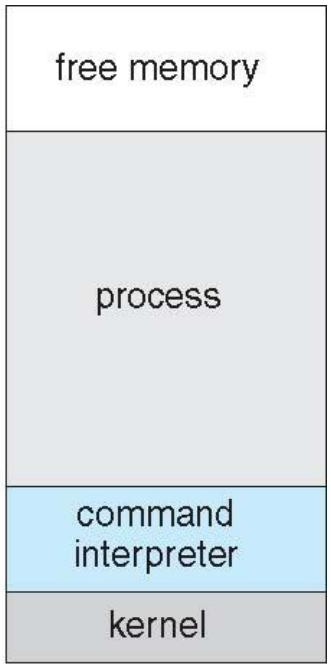
Example: MS-DOS

- Single-tasking
- Shell invoked when system booted
- Simple method to run program
  - No process created
- Single memory space
- Loads program into memory, overwriting all but the kernel
- Program exit -> shell reloaded



(a)

At system startup

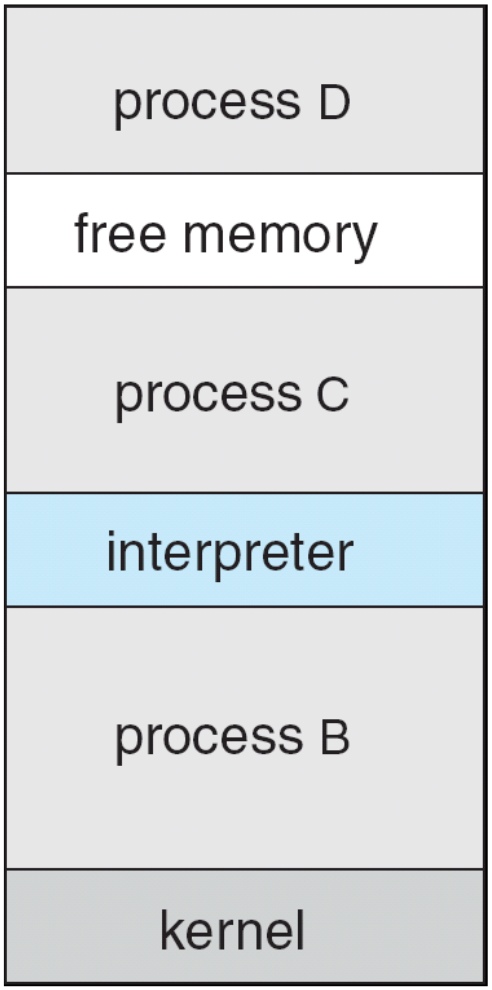


(b)

running a program

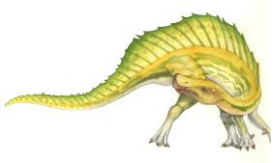
Example: FreeBSD

- Unix variant
- Multitasking
- User login -> invoke user's choice of shell
- Shell executes fork() system call to create process
  - Executes exec() to load program into process
  - Shell waits for process to terminate or continues with user commands
- Process exits with:
  - code = 0 – no error
  - code > 0 – error code



Operating System Architectures

- Today's operating systems tend to be complex
  - Provide many services
  - Support variety of hardware and software
  - Operating system architectures help manage this complexity
    - Organize operating system components
    - Specify privilege with which each component executes

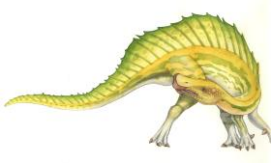


Operating System Structure

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

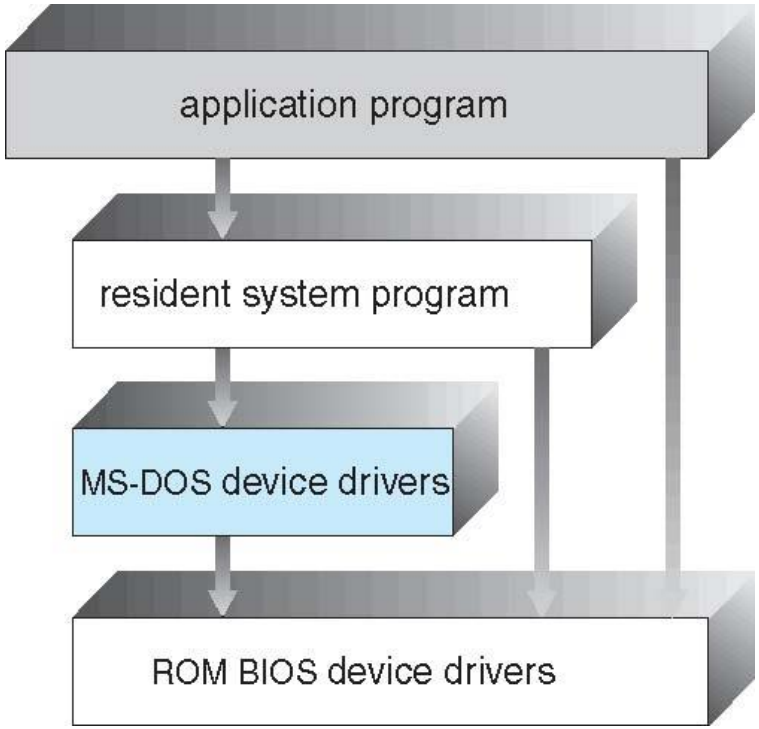
Simple Structure

- Such operating systems do not have well defined structure and are small, simple and limited systems.
- The interfaces and levels of functionality are not well separated.
- MS-DOS is an example.
- No dual mode of operation
- In MS-DOS application programs are able to access the basic I/O routines.
- These types of operating system cause the entire system to crash if one of the user programs fails



Simple Structure -- MS-DOS

- 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



- There is no hardware protection and I/O protection
- No system call concept
- Direct interaction causes errors or malfunctions

Advantages of Simple structure:

- It delivers better application performance because of the few interfaces between the application program and the hardware.
- Easy for kernel developers to develop such an operating system.
- More functionality in less space

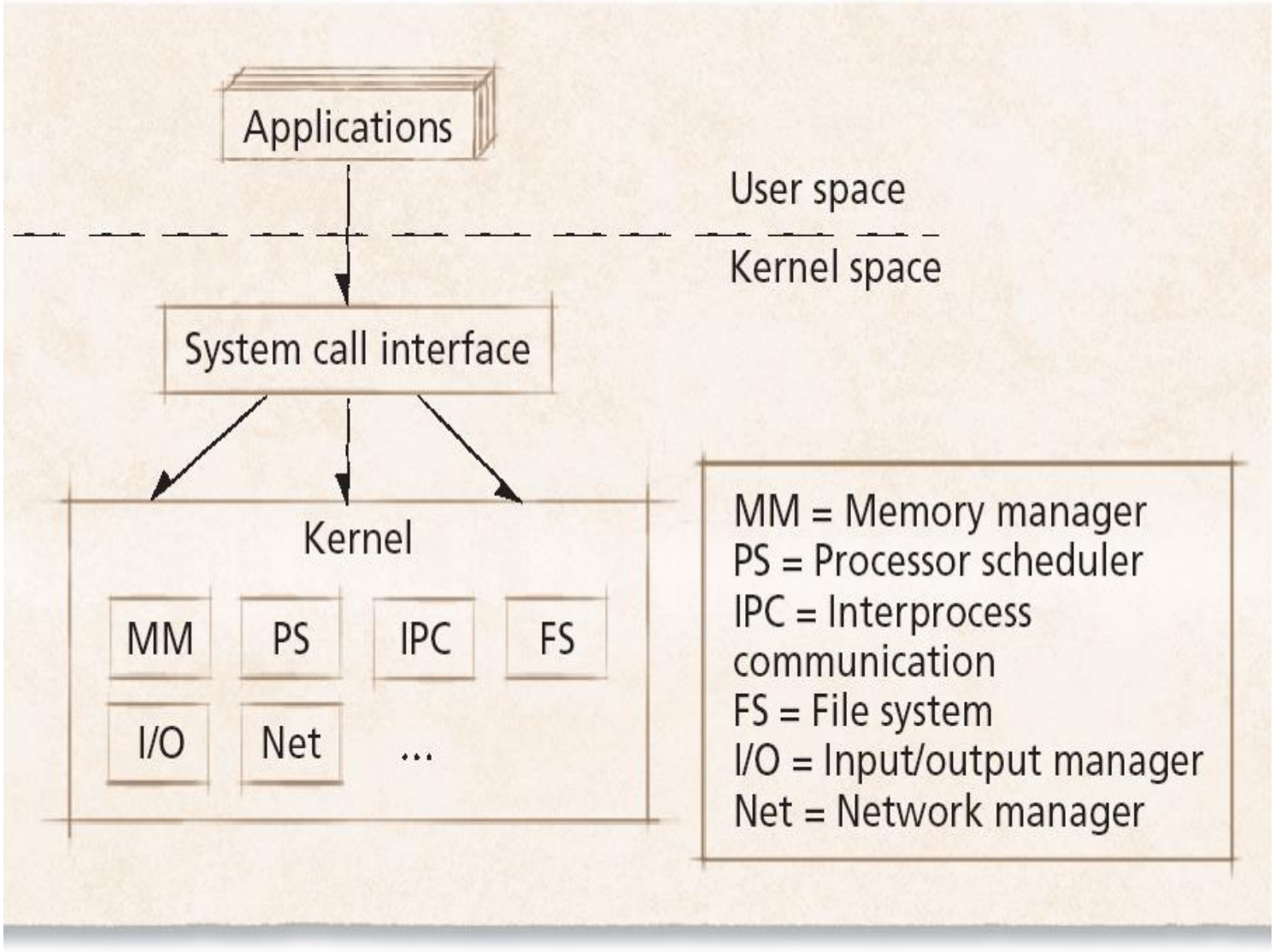
Disadvantages of Simple structure:

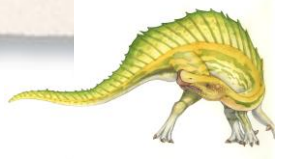
- The structure is very complicated as no clear boundaries exists between modules.
- It does not enforce data hiding in the operating system.
- Direct access to hardware made it vulnerable to malicious program
- Whole system is affected if application program crashes.

Monolithic Architecture

- Monolithic operating system
  - Every component contained in kernel
    - Any component can directly communicate with any other
  - Tend to be highly efficient
  - Disadvantage is difficulty determining source of subtle errors

Monolithic Architecture





Non Simple Structure -- UNIX

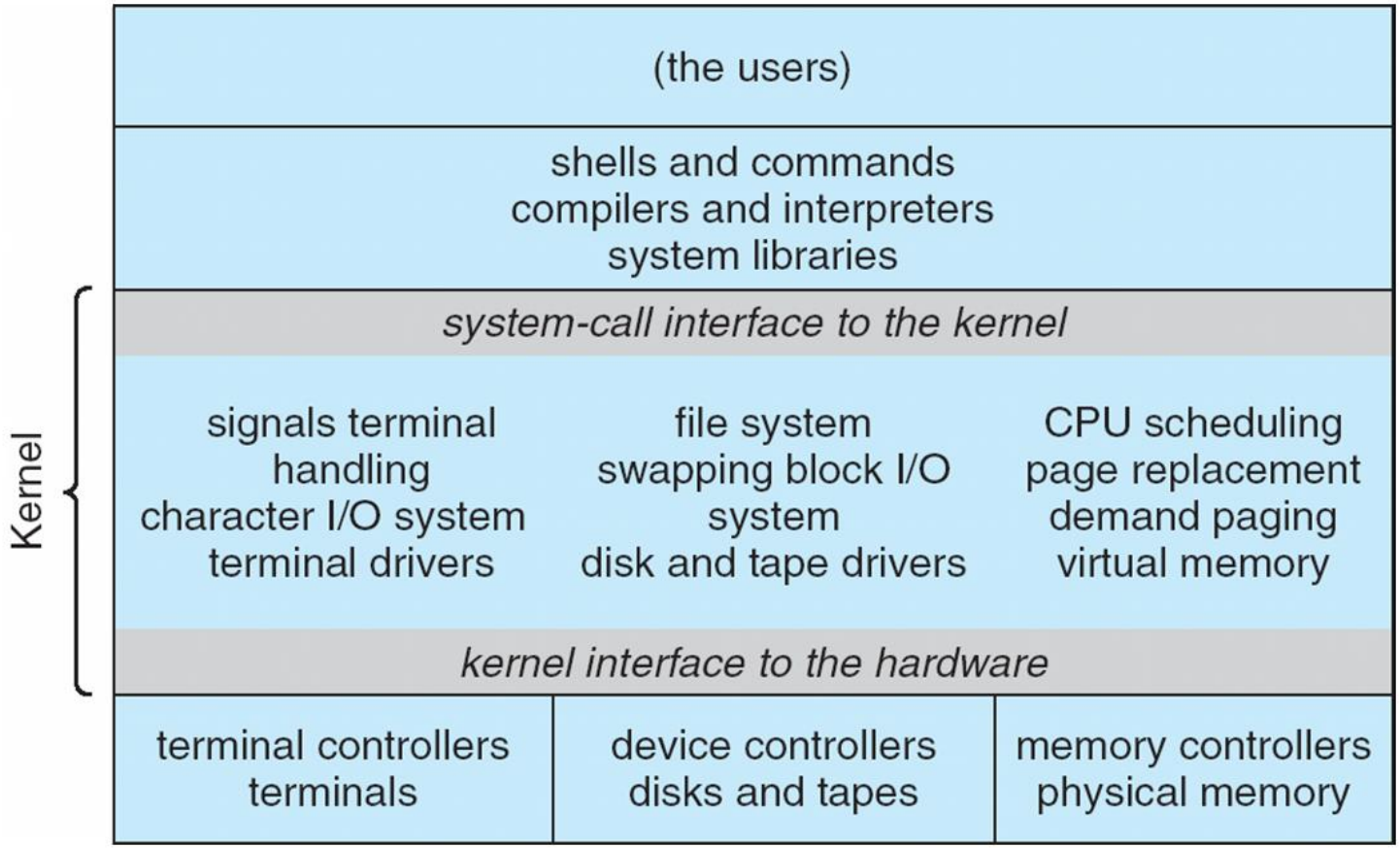
UNIX – limited by hardware functionality, the original UNIX operating system had limited structuring.

The UNIX OS consists of two separable parts

- Systems programs
- The kernel
  - 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

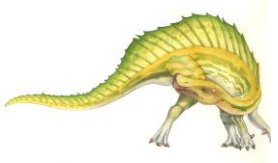
Traditional UNIX System Structure

Beyond simple but not fully layered



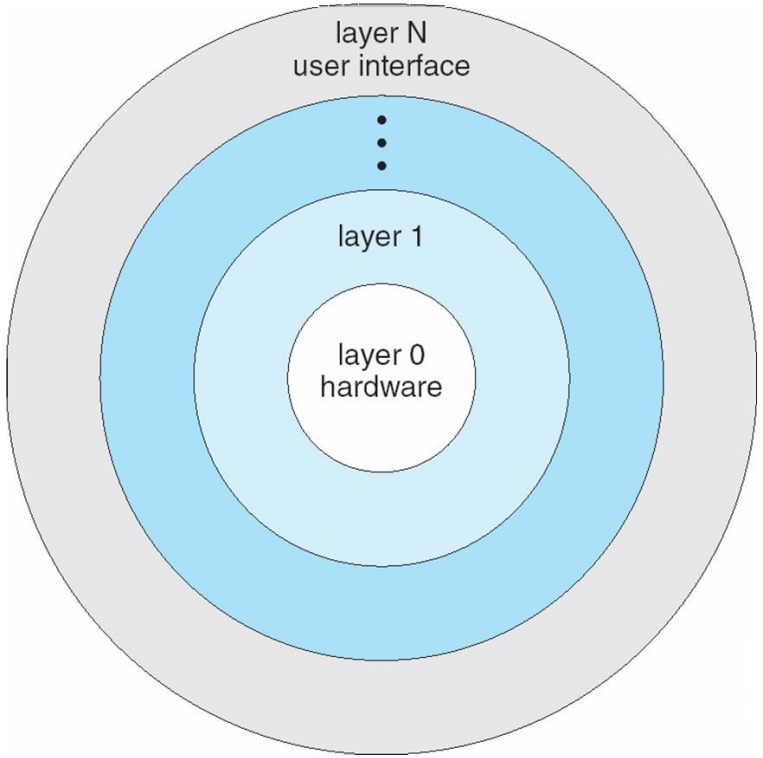
Layered Architecture

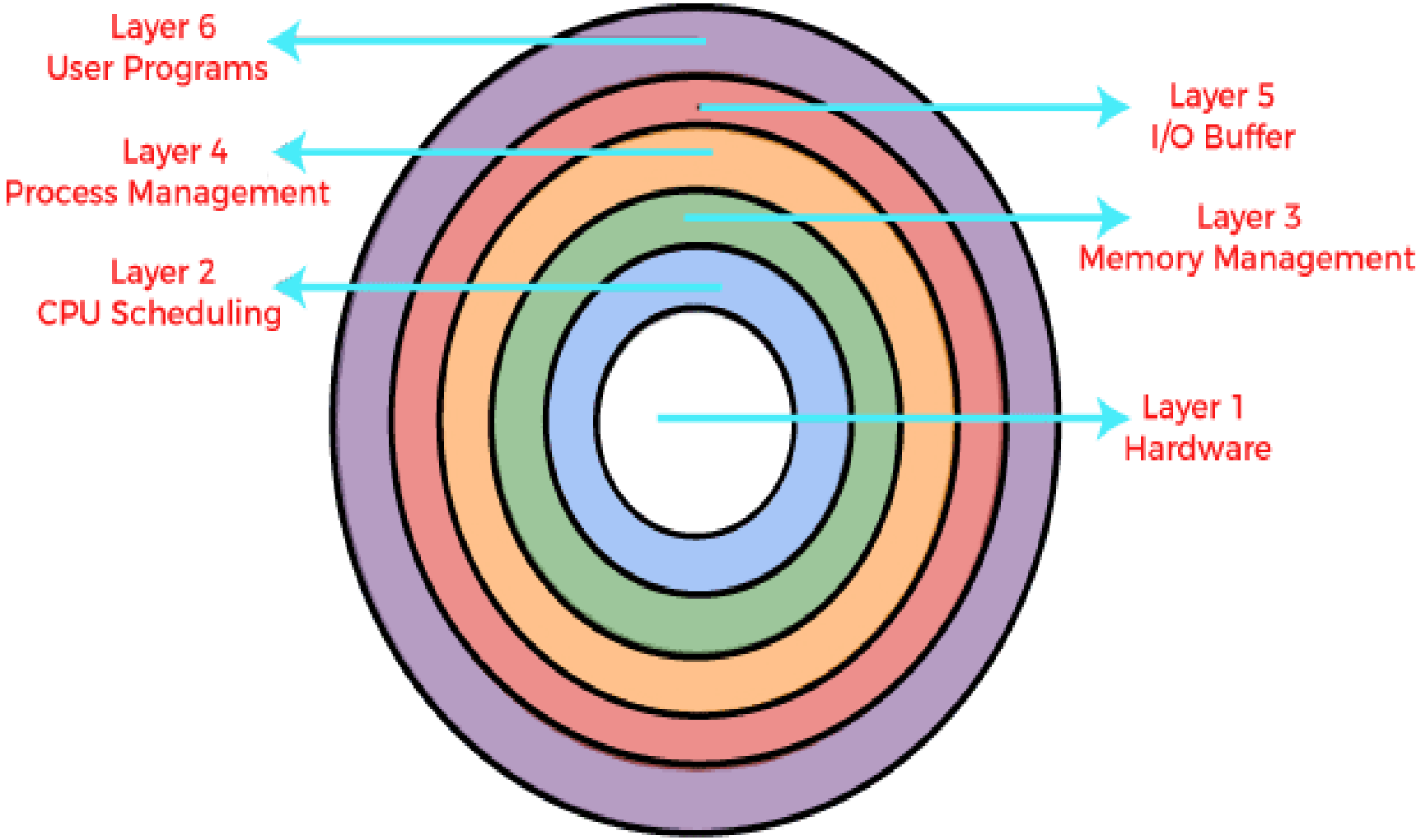
- Layered approach to operating systems
  - Tries to improve on monolithic kernel designs
    - Groups components that perform similar functions into layers
  - Each layer communicates only with layers immediately above and below it
  - Processes' requests might pass through many layers before completion
  - System throughput can be less than monolithic kernels
    - Additional methods must be invoked to pass data and control



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

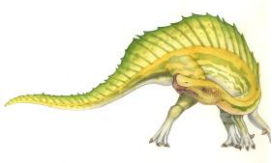



Layer 6
User Programs
Layer 5
I/O Buffer
Layer 4
Process Management
Layer 3
Memory Management
Layer 2
CPU Scheduling
Layer 1
Hardware

- Layered approach introduces the modularization of the system.
  - Design and implementation of system are simplified where system is broken into layers.
- A layer does not need to know how these operations are implemented, it needs to know only what these operations do.
- Each layer hides existence of certain data structures, operations and hardware from higher level layer
- Each layer has a specific function. Each layer has its own Data structures and operations.
- Highest level has high functionality, i.e., the services for highest level are provided by the lowest level.

Advantages of Layered structure

- **Modularity:** This design promotes modularity as each layer performs only the tasks it is scheduled to perform.
- **Easy debugging:** As the layers are discrete so it is very easy to debug. Suppose an error occurs in the CPU scheduling layer. The developer can only search that particular layer to debug, unlike the Monolithic system where all the services are present.
- **Easy update:** A modification made in a particular layer will not affect the other layers.



Advantages of Layered structure

- **No direct access to hardware:** The hardware layer is the innermost layer present in the design. So a user can use the services of hardware but cannot directly modify or access it, unlike the Simple system in which the user had direct access to the hardware.
- **Abstraction:** Every layer is concerned with its functions. So the functions and implementations of the other layers are abstract to it.

Disadvantages of Layered structure

- **Complex and careful implementation:** As a layer can access the services of the layers below it, so the arrangement of the layers must be done carefully.
  - For example, the backing storage layer uses the services of the memory management layer. So it must be kept above the memory management layer. Thus with great modularity comes complex implementation.
- **Functionality:** It is not always possible to divide the functionalities. Many times, they are interrelated and can't be separated.

Disadvantages of Layered structure

- **Slower in execution:** If a layer wants to interact with another layer, it requests to travel through all the layers present between the two interacting layers. Thus it increases response time, unlike the Monolithic system, which is faster than this. Thus an increase in the number of layers may lead to a very inefficient design.
- **Communication:** No communication between non-adjacent layers.

Microkernel

- This structure designs the operating system by removing all non-essential components from the kernel and implementing them as system and user programs.
- This result in a smaller kernel called the micro-kernel.
- Advantages of this structure are that all new services need to be added to user space and does not require the kernel to be modified.
- Thus it is more secure and reliable as if a service fails then rest of the operating system remains untouched.

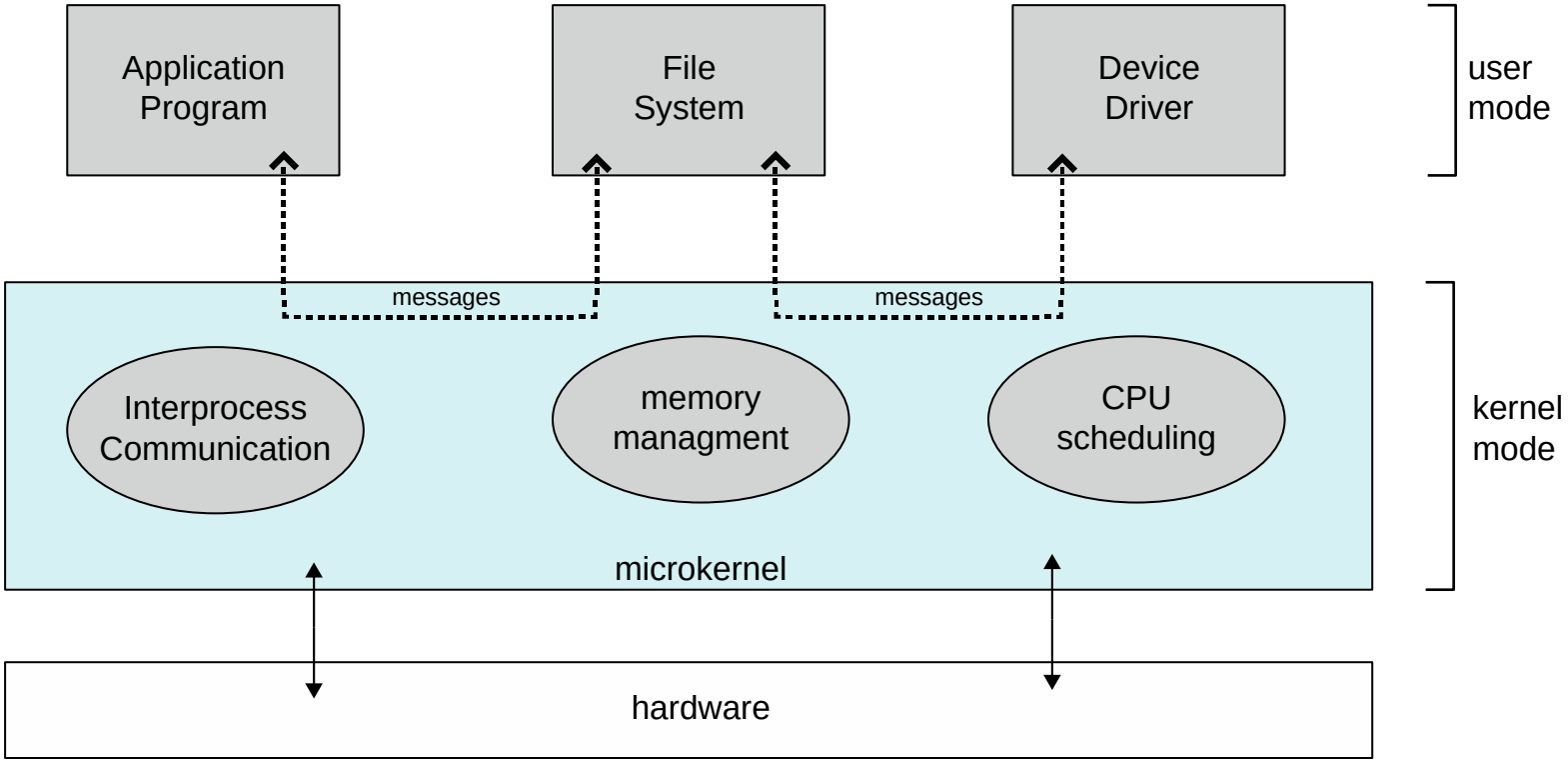
Microkernel System Structure

- 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

Microkernel System Structure

Detriments:

Performance overhead of user space to kernel space communication



Advantages of Micro-kernel structure:

- It makes the operating system portable to various platforms.
- As microkernels are small so these can be tested effectively.

Disadvantages of Micro-kernel structure:

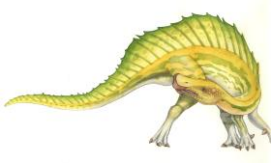
- Increased level of inter module communication degrades system performance.

Microkernel Architecture

- Microkernel operating system architecture
  - Provides only small number of services
    - Attempt to keep kernel small and scalable
  - High degree of modularity
    - Extensible, portable and scalable
  - Increased level of intermodule communication
    - Can degrade system performance

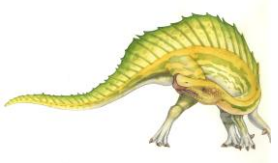
Modular approach

- It is considered as the best approach for an OS.
- It involves designing of a modular kernel.
- The kernel has only set of core components and other services are added as dynamically loadable modules to the kernel either during run time or boot time.
- It resembles layered structure due to the fact that each kernel has defined and protected interfaces but it is more flexible than the layered structure as a module can call any other module.

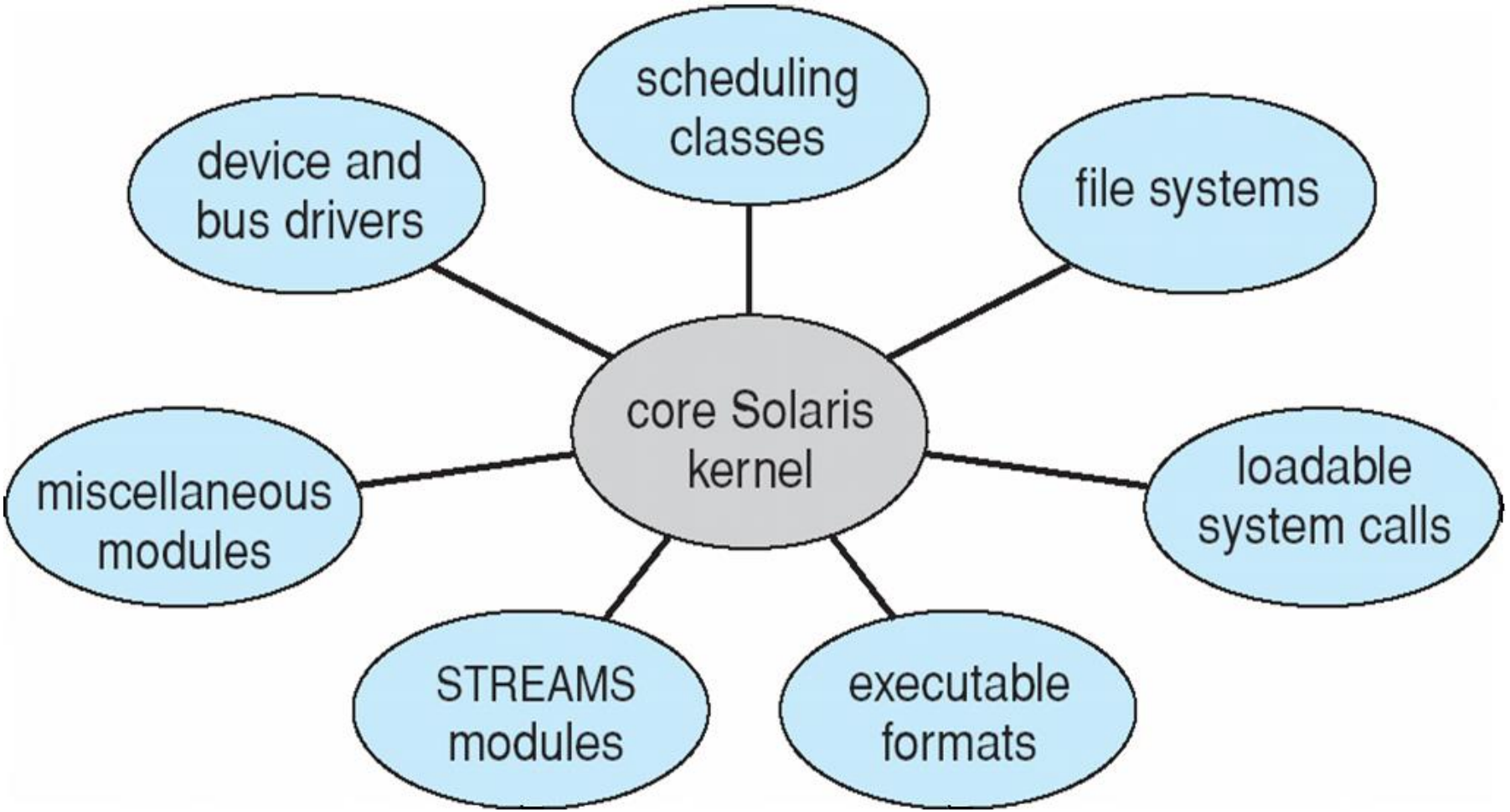


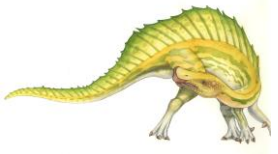
Modular structure or approach

- 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



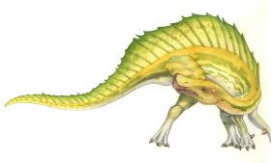
Solaris Modular Approach





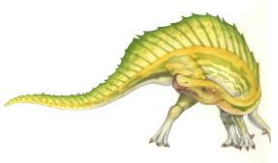
Modular approach

- No need to recompile kernel every time a new service is added.
- Modules need only to be linked dynamically
- More efficient – no need to pass messages/recurrently call lower layers
- Object oriented Programming (OOP) technique is used to create a modular kernel



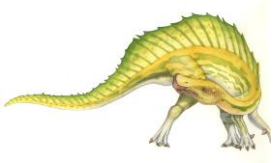
Hybrid structure

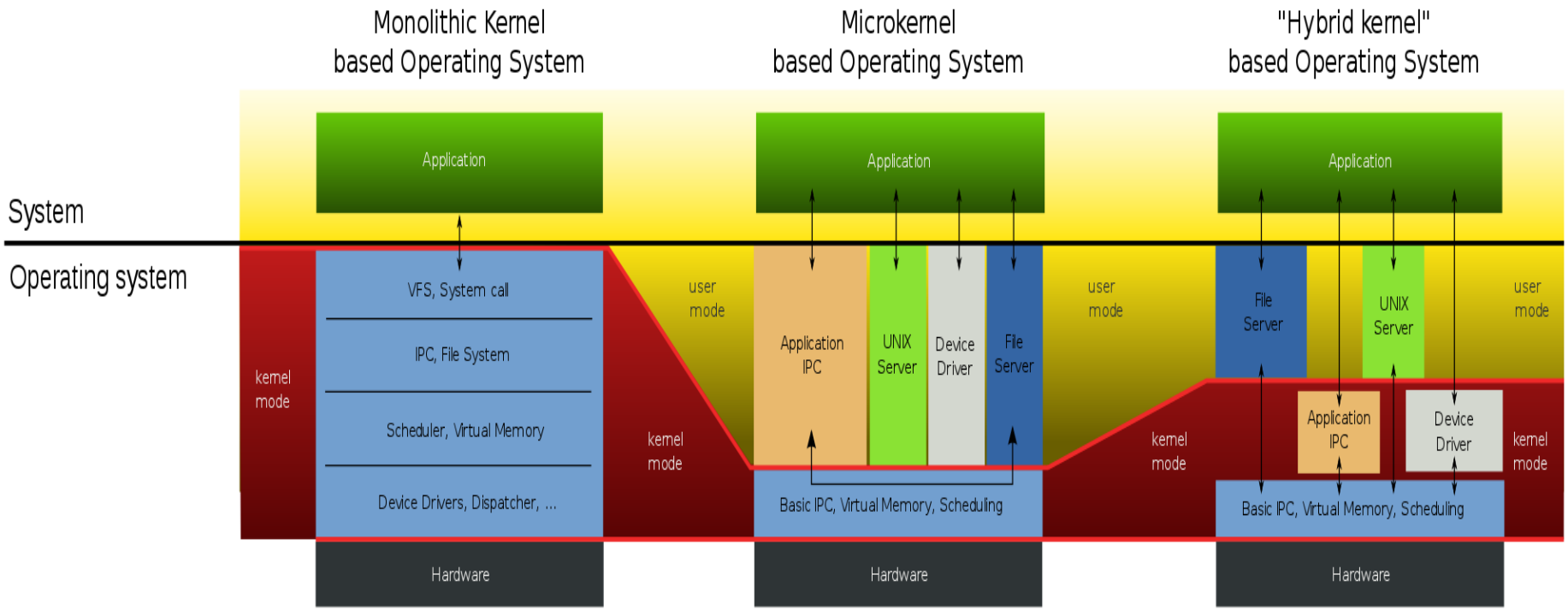
- Combination of different structures
- Mixture of monolith and modular to achieve better efficiency.
- Can use layers, microkernel and loadable modules to optimize performance

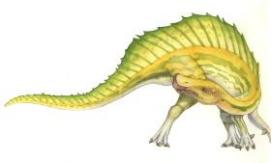


Hybrid Systems

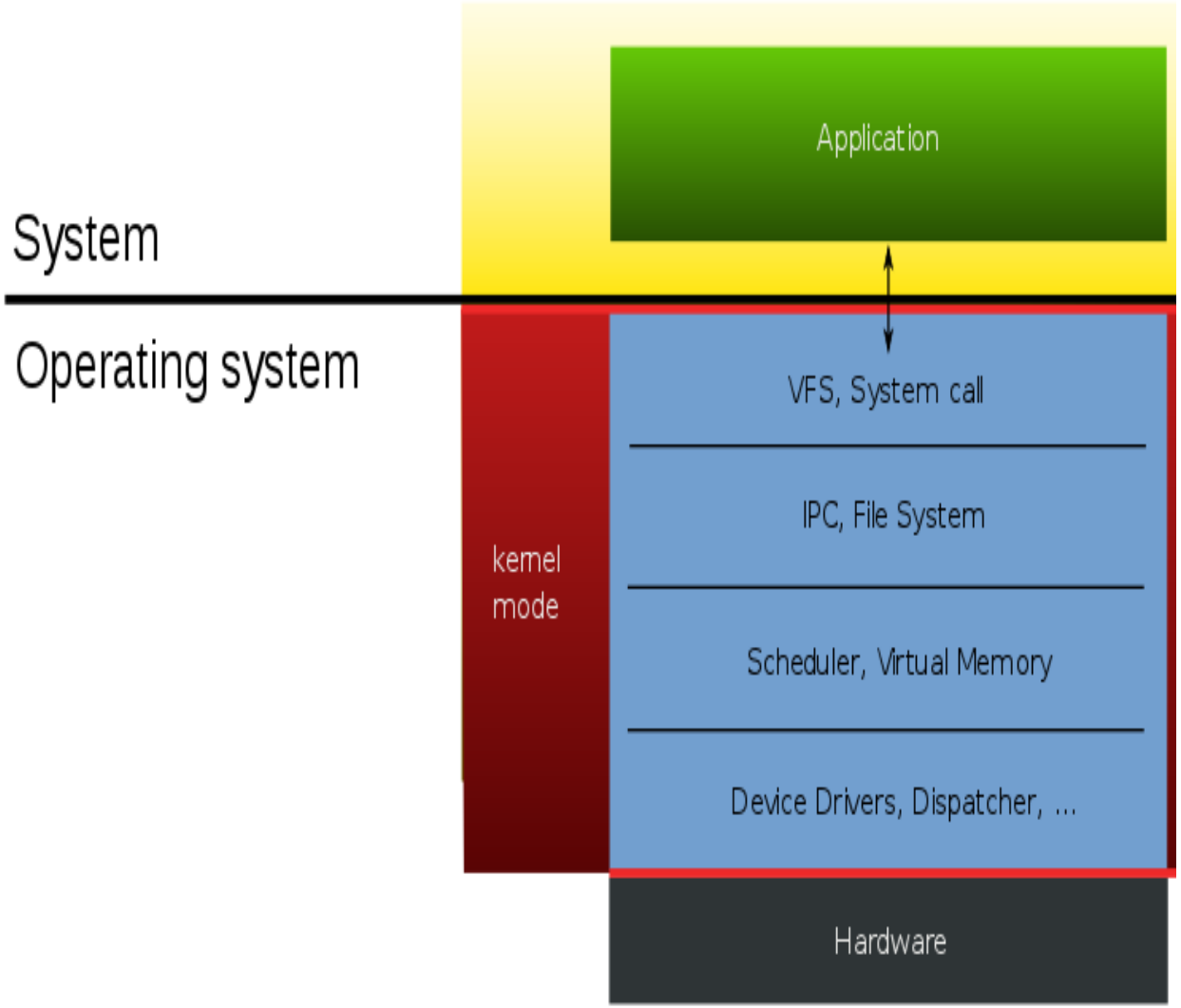
- Most modern operating systems are actually not one pure model: Combination of different structures
- Hybrid combines multiple approaches to address performance, security, usability needs
  - Linux and Solaris kernels in kernel address space, so monolithic, plus modular for dynamic loading of functionality
  - Windows mostly monolithic, plus microkernel for different subsystem ***personalities***

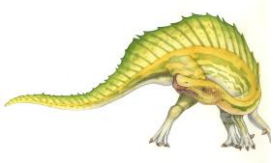



Monolithic Kernel based Operating System
Microkernel based Operating System
"Hybrid kernel" based Operating System
System
Operating system
Application
VFS, System call
IPC, File System
Scheduler, Virtual Memory
Device Drivers, Dispatcher, ...
Hardware
kernel mode
user mode
Application IPC
UNIX Server
Device Driver
File Server
Basic IPC, Virtual Memory, Scheduling

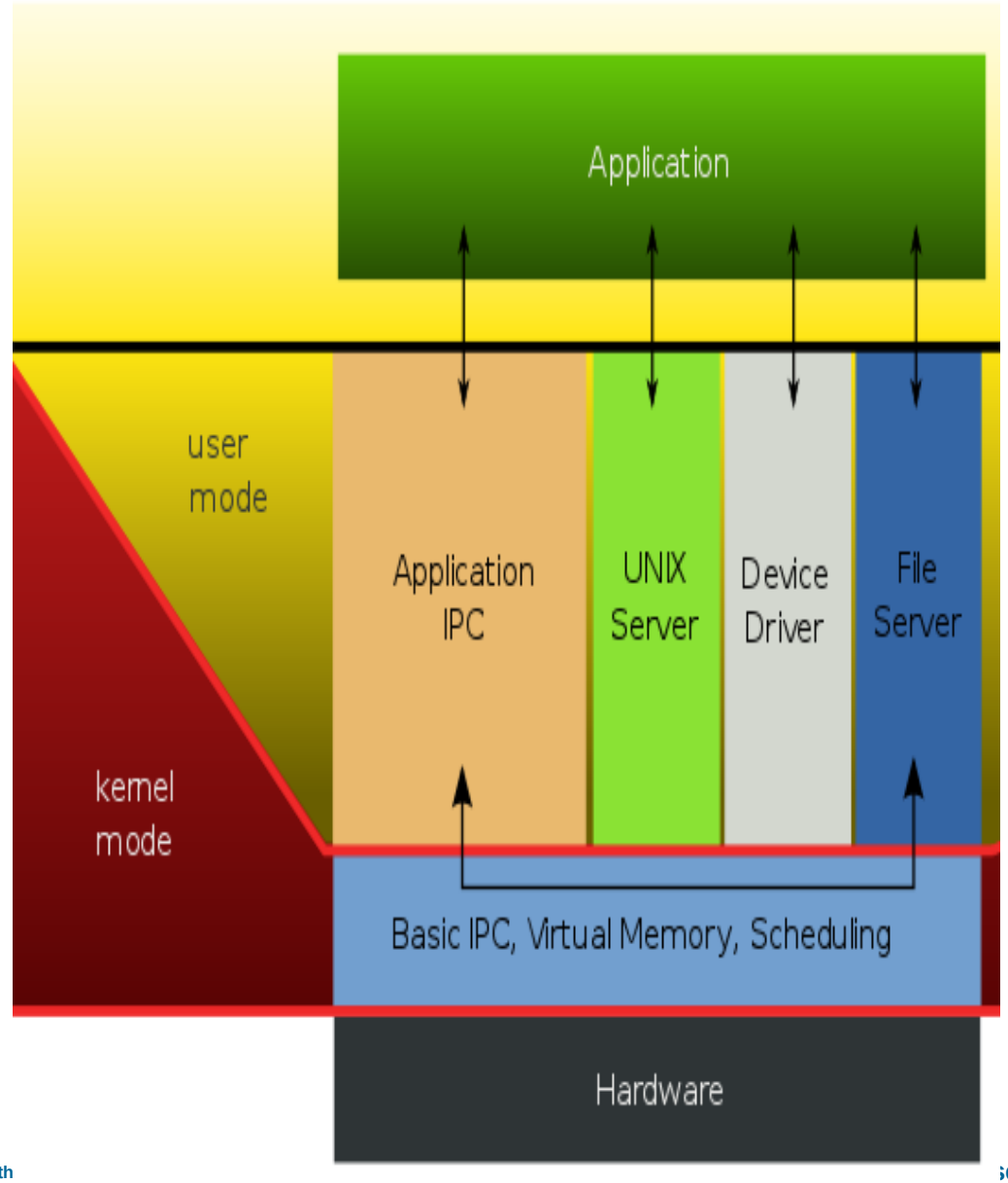


Monolithic Kernel based Operating System

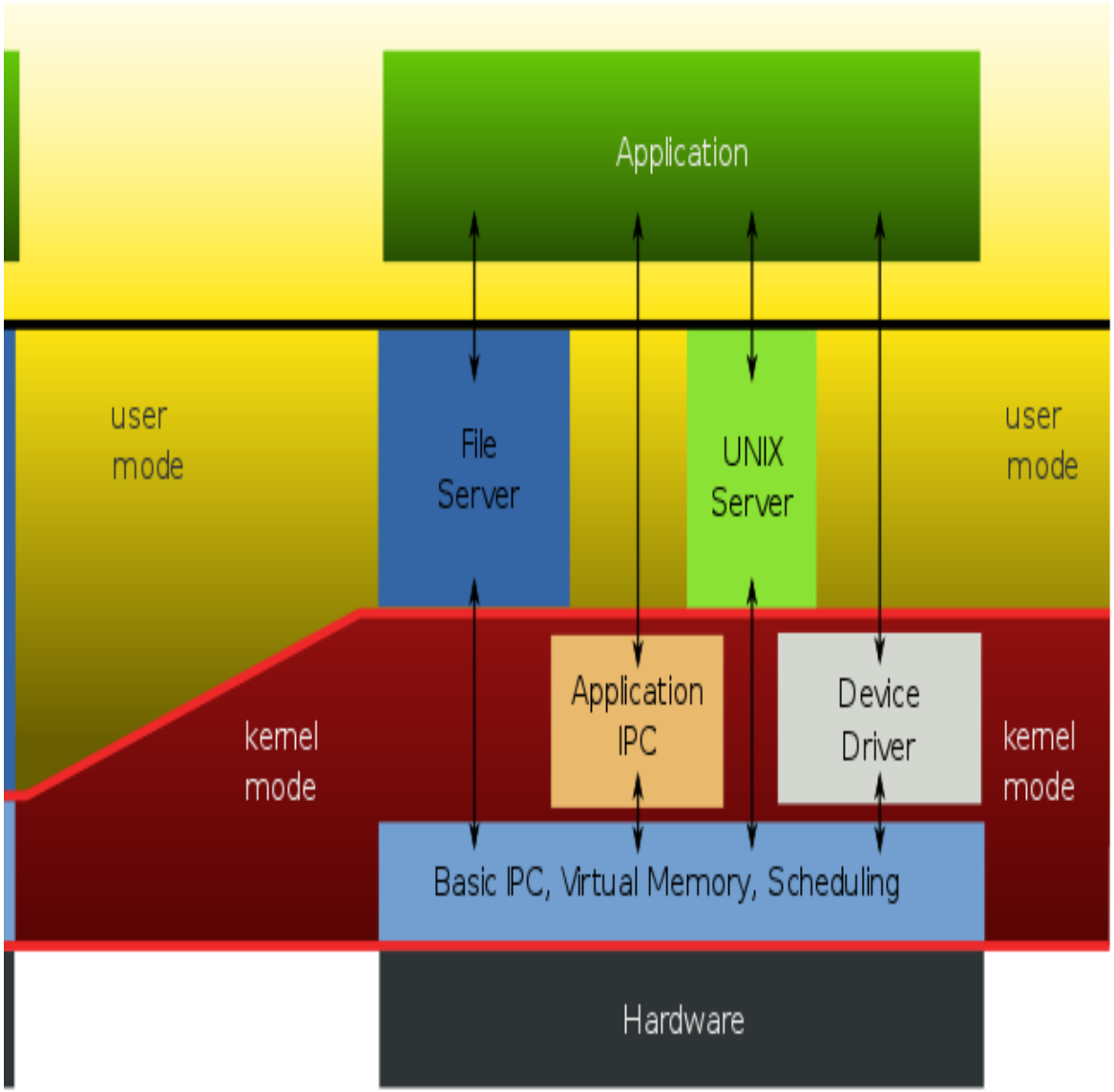




Microkernel based Operating System

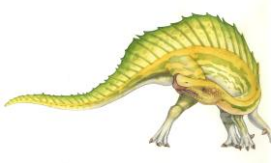


"Hybrid kernel" based Operating System

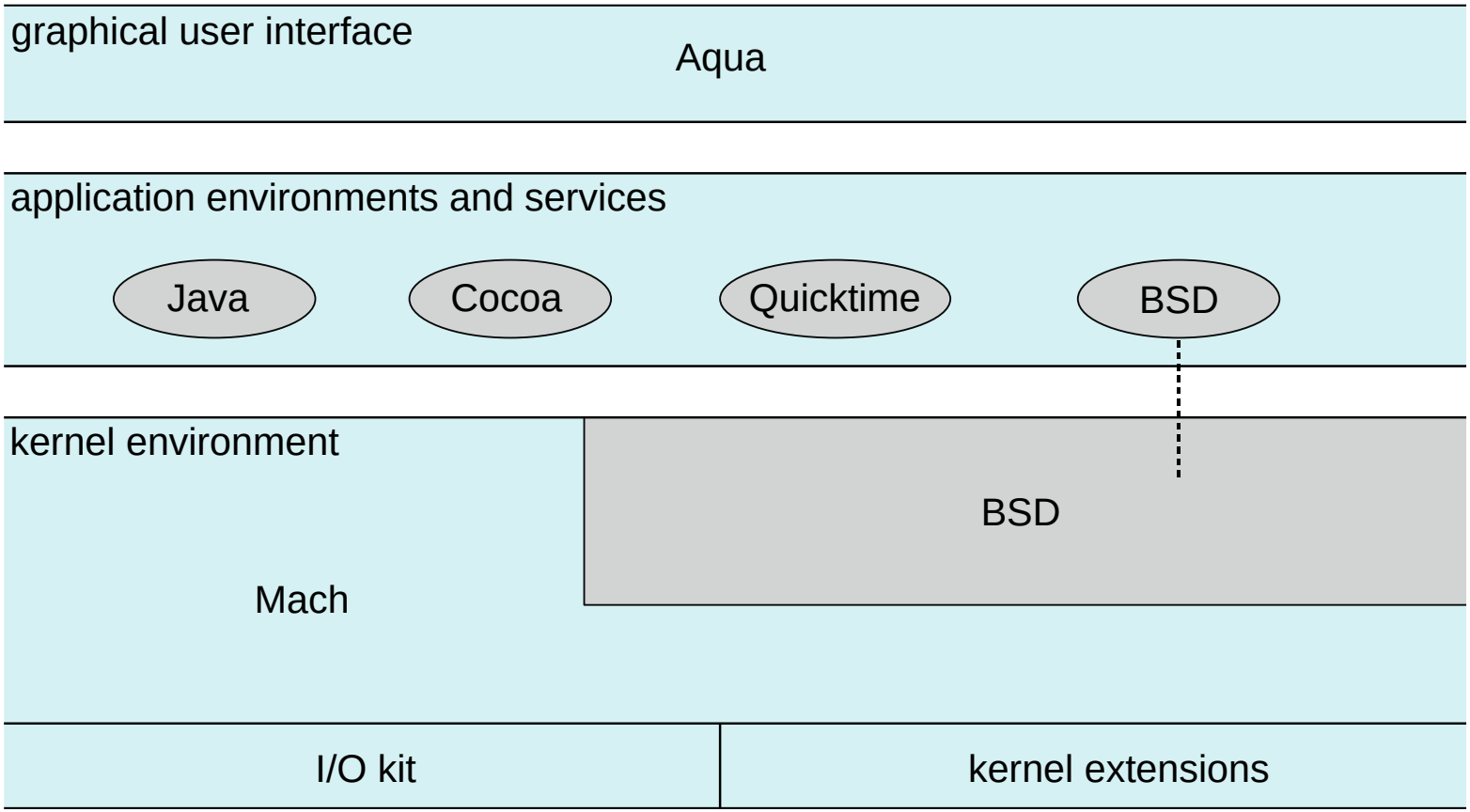


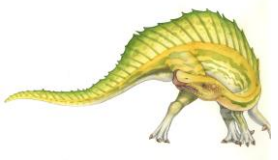
Mac OS X

- Apple Mac OS X hybrid, layered, **Aqua** UI plus **Cocoa** programming environment
  - Below is kernel consisting of
    - Mach microkernel and BSD Unix parts,
    - I/O kit and dynamically loadable modules (called **kernel extensions**)



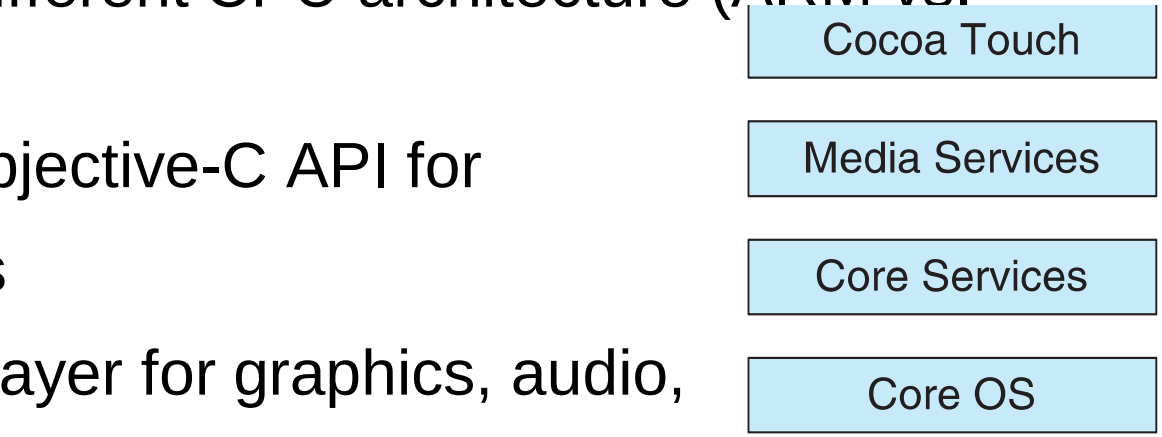
Mac OS X Structure





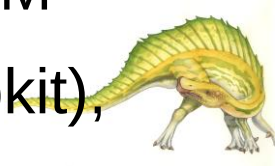
iOS

- Apple mobile OS for ***iPhone***, ***iPad***
  - Structured on Mac OS X, added functionality
  - Does not run OS X applications natively
    - Also runs on different CPU architecture (ARM vs. Intel)
  - **Cocoa Touch** Objective-C API for developing apps
  - **Media services** layer for graphics, audio, video
  - **Core services** provides cloud computing, databases
  - Core operating system, based on Mac OS X kernel

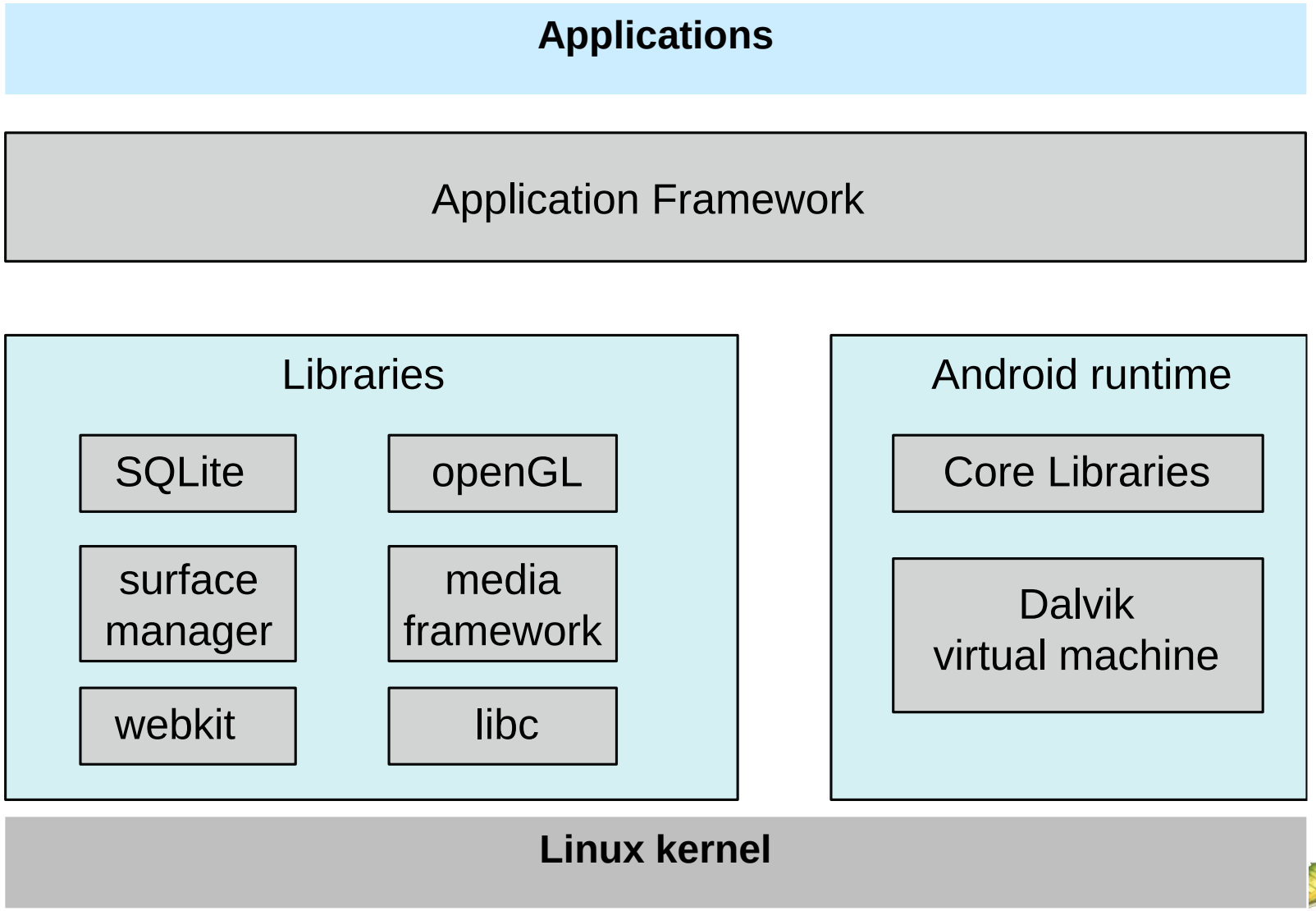


Android

- Developed by Open Handset Alliance (mostly Google)
  - Open Source
- Similar stack to IOS
- Based on Linux kernel but modified
  - Provides process, memory, device-driver management
  - Adds power management
- Runtime environment includes core set of libraries and Dalvik virtual machine
  - Apps developed in Java plus Android API
    - Java class files compiled to Java bytecode then translated to executable than runs in Dalvik VM
- Libraries include frameworks for web browser (webkit), database (SQLite), multimedia, smaller libc

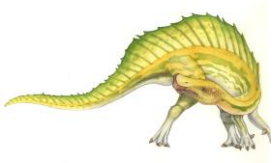


Android Architecture



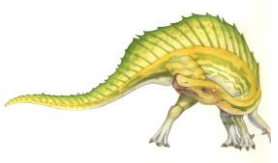
Booting an Operating System

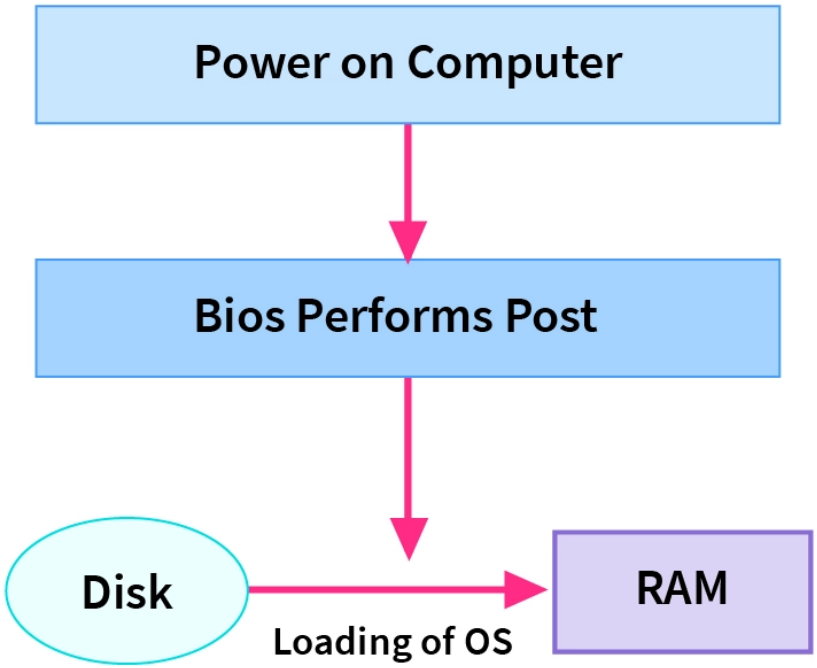
- Whenever we press the power button of our computer system, all the devices gets the power and they are initialized.
- Our main memory which is responsible to hold the instructions will be initially empty as RAM is volatile memory. So, there will be small set of instructions present in the non-volatile memory called ROM.
- These instructions will be passed to CPU and the execution of instructions takes place which will check all the hardware connected with the system.



Booting an Operating System

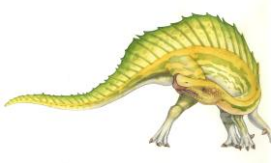
- If there are any problems with the hardware, we will get the alert by beep sounds or even on-screen messages.
- After the testing of hardware is completed, the booting process continues and load the operating system.





The instructions present in non-volatile memory are hardwired on motherboard so it can't be erased.

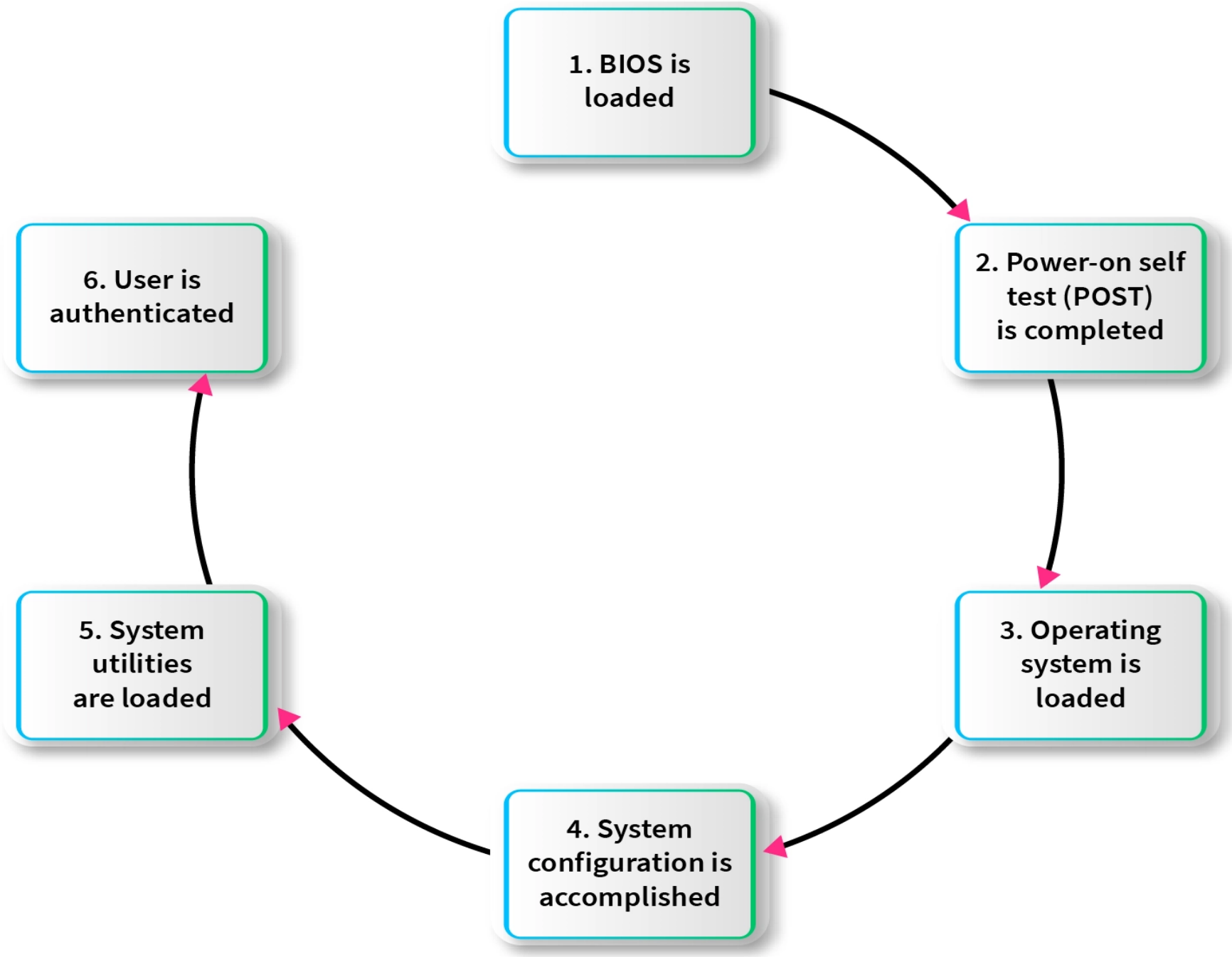
The small set of instructions present in the ROM is called BIOS which stands for Basic Input Output System.



- As BIOS is lightweight, it will just load the Boot Loader which can load the Complex set of libraries required for loading the Operating System.
- BIOS can’t directly load the heavy weighted set of instructions responsible for loading the Operating System.

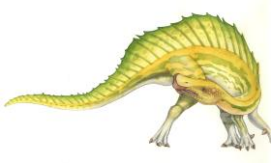
- **Loading of Operating System:**
  - After the successful completion of POST, the bootable sequence present in CMOS (Common Metal Oxide Semiconductor) is read by BIOS.
  - Based on the bootable sequence it will search for Master Boot Record (MBR) in bootable device like floppy disk, CD-ROM and hard disk.
  - If MBR is not found in any of them, the system will halt by displaying “No Boot Device Found”.
  - if MBR is found, the BIOS will load the special application program called Boot Loader, that will eventually load the Operating system.

- **System Configuration is Accomplished:** After the OS is loaded, device drivers are loaded into the memory so that our devices can function correctly.
- **System Utilities are Loaded:** System utilities like antivirus, volume control etc. are loaded into the memory in this step
- **User Authentication:** If any user authentication is being set, the system will ask the user to enter the credentials and on receiving the correct credentials the computer system will run GUI shell (in most cases) or CLI shell.


1. BIOS is loaded
2. Power-on self test (POST) is completed
3. Operating system is loaded
4. System configuration is accomplished
5. System utilities are loaded
6. User is authenticated

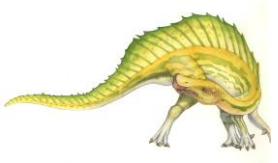
Type of Booting

- **Cold Booting/ Hard Booting:** Cold booting is the process when our computer system moves from shut down state to the start by pressing the power button. The system reads the BIOS from ROM and will eventually load the Operating System.
- **Warm Booting/ Soft Booting:** Warm booting is the process in which the computer gets restart due to reasons like setting the configuration for newly installed software or hardware. Warm booting is called as *rebooting*.



How does a System Boot?

- When we press the power button, all the components of system get power and they get initialized. Once the CPU is initialized, it needs instruction to be executed.
- The small set of instructions called BIOS are being loaded from ROM.
- After the successful completion of Power-On-Self-Test (POST) BIOS find the bootable sequence from CMOS.
- Based on the bootable sequence, it finds the first bootable device.

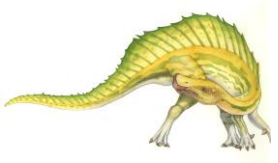


How does a System Boot?

- From the first bootable device, it loads instructions present from Master Boot Record which is present in logical Sector 0.
- This set of instructions contains the information about the Boot Loader which can load the Operating System.
- This boot loader information is Operating System specific, for example the boot loader for Linux is GRUB (GRand Unified Bootloader).
- The boot loader then load the operating system into the memory.
- Lastly all the important system files and drivers are loaded into memory and the control is being passed to operating system.

Dual Booting of Operating System

- When we have two different operating system on our computer it is called Dual Booting.
- But as we have multiple OS present, bootloader need to load the Operating System which user selects from the menu that is being displayed on the monitor.
- If no action is performed within few seconds, the default Operating System is loaded.



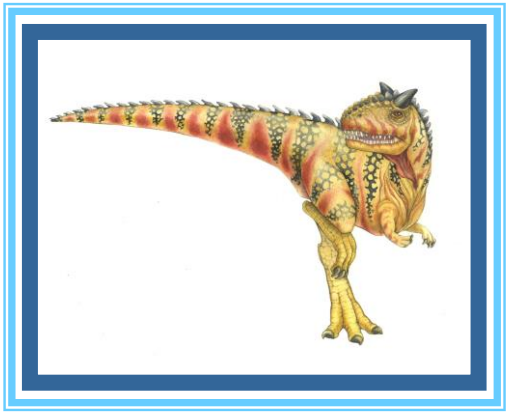
System Boot

- When power initialized on system, execution starts at a fixed memory location
  - Firmware ROM used to hold initial boot code
- Operating system 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**

How could a system be designed to allow a choice of operating systems from which to boot?

- During system boot-up, a special program (**boot manager**) will determine which operating system to boot into.
- Boot manager will first run during system startup.
- Typically boot managers must be stored at certain locations of the hard disk to be recognized during system startup.
- Boot managers are designed to boot into a default operating system if no choice is selected by the user.

RECAP



Silberschatz, Galvin and Gagne ©2013

- What are the four components of a computer system?
- Provide at least three resources the operating system allocates.
- Major activities of an operating system with regard to
  - process management
  - memory management
  - Device management
  - File management
- What is the term for a program that has been loaded and is executing?
- Provide an example of an open source operating system.

- What are the two separate modes of operation?
- What is the mode of the system at boot time?
- What is the mode of the system when the operating system gains control?
- What is the mode of the system when a user program is running?
- How does the distinction between kernel mode and user mode function as a rudimentary form of protection (security) system?

- What is the purpose of system programs?
- What is the purpose of system calls?
- List at least three different ways for structuring an operating system.
- What is the main advantage of the layered approach to system design?
- What technique do microkernels use to communicate between services?
- Provide an example of an operating system that uses the simple structure.
- What is the name of the small piece of code that locates the kernel and loads it into main memory?