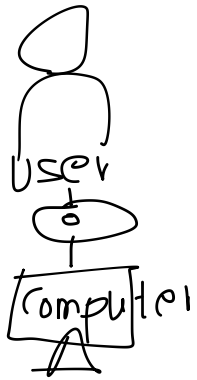
Operating Systems
Module-1

What is an Operating System?



- A program that acts as an intermediary between a user of a computer and the computer hardware.
- 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.

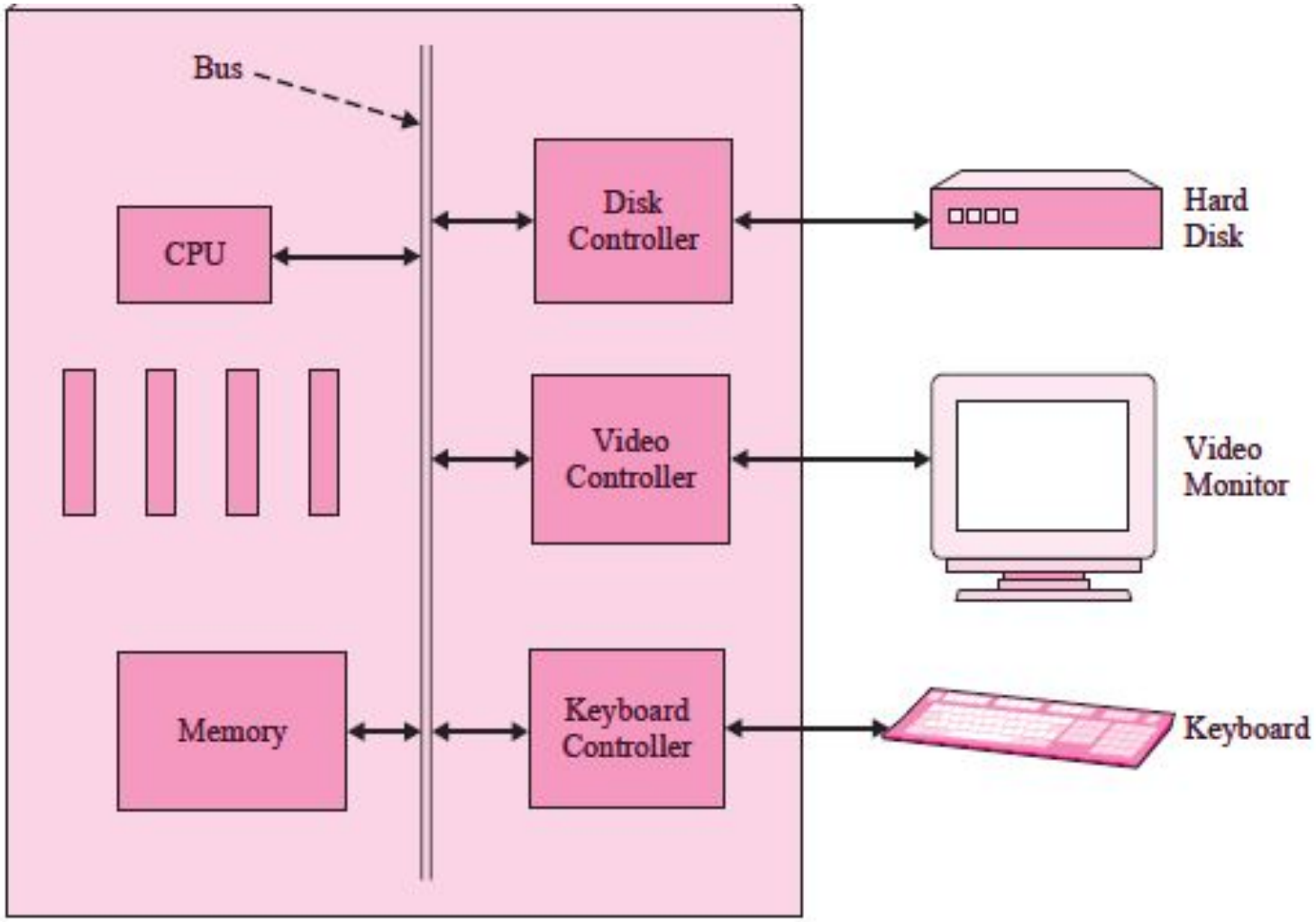
Definition

- A computer program that acts as an interface between the user and computer hardware and controls and manages the overall resources of the computer system.

A few questions…..

- Have you written any programs? yes
  - Is it platform-dependent or platform-independent?
- Where do you execute your programs? compiler
- Did you use any compilers? Yes!
- Can I copy and execute my compiled code from my desktop to laptop? Yes!
- Does the mother board contain only one component? No!
  - If not, who is going to manage all the components?
- What is abstraction and interface?

Hardware: A very simplistic view of a small personal computer



- Hardware resources of a computer:
  - CPUs (processors)
  - Main memory and caches,
  - Secondary storage,
  - I/O devices at the lowest level,
  - Network access

  Who is going to manage these resources?
  - A User or Software?
    - Software – Operating Systems
- Why to manage these resources?

- Operating System
  - The OS is a **collection of one or more software modules** that **manages and controls the resources of a computer** or other computing or electronic device, and **gives users and programs an interface to utilize these resources**. The managed resources include memory, processor, files, input or output devices, and so on.
  - Responsibilities/ Services
    - Memory management
    - Device management
    - Processor management
    - Security
    - Error detection
    - Job accounting
    - File management

Operating System Services/ Functionality

- 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)
  - **Program execution -** The system must be able to load a program into memory and to run that program, end execution, either normally or abnormally (indicating error)
  - **I/O operations -** A running program may require I/O, which may involve a file or an I/O device.
  - **File-system manipulation -** The file system is of particular interest. Obviously, programs need to read and write files and directories, create and delete them, search them, list file Information, permission management.

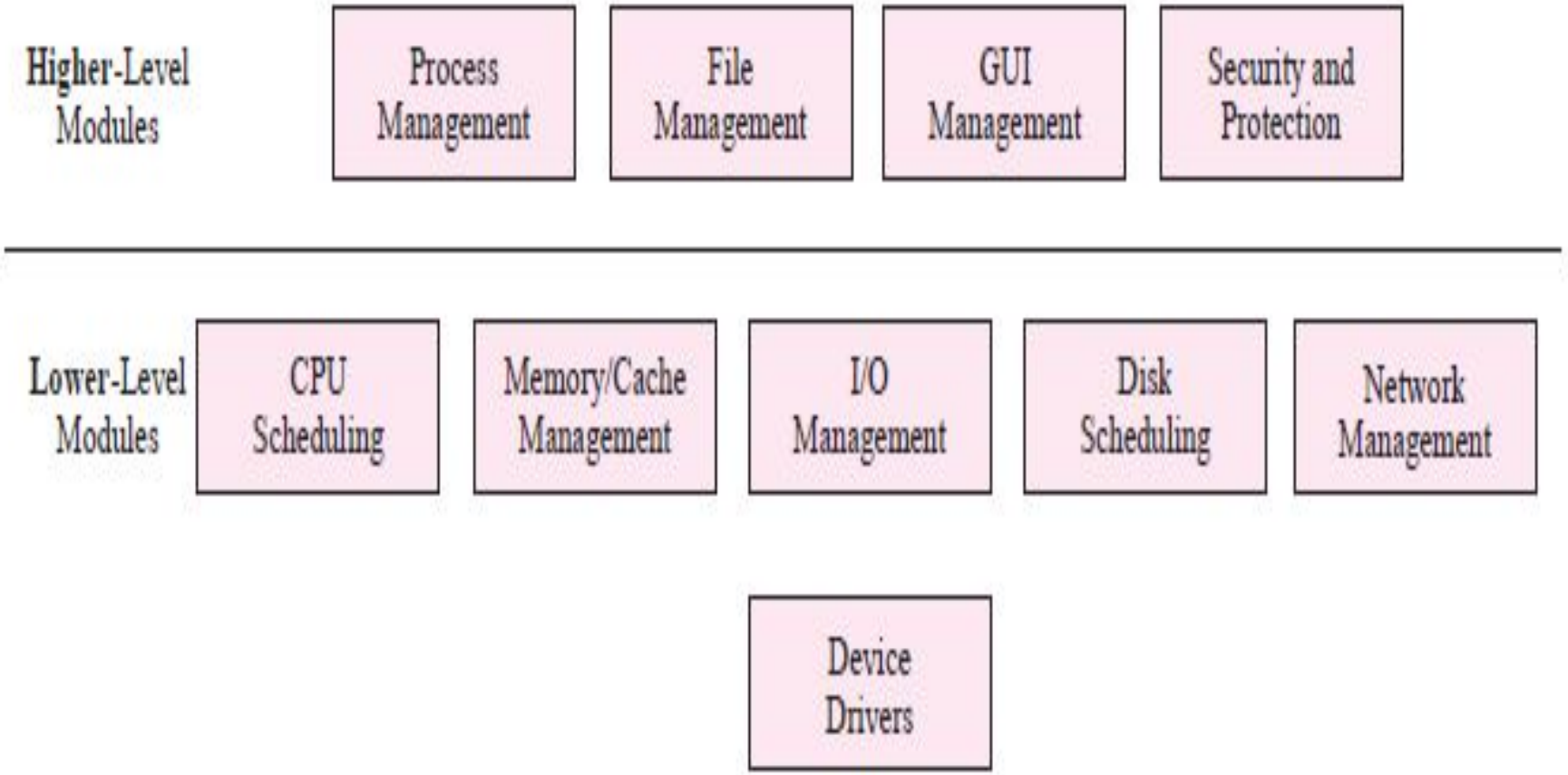
Operating System Services (Cont.)

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

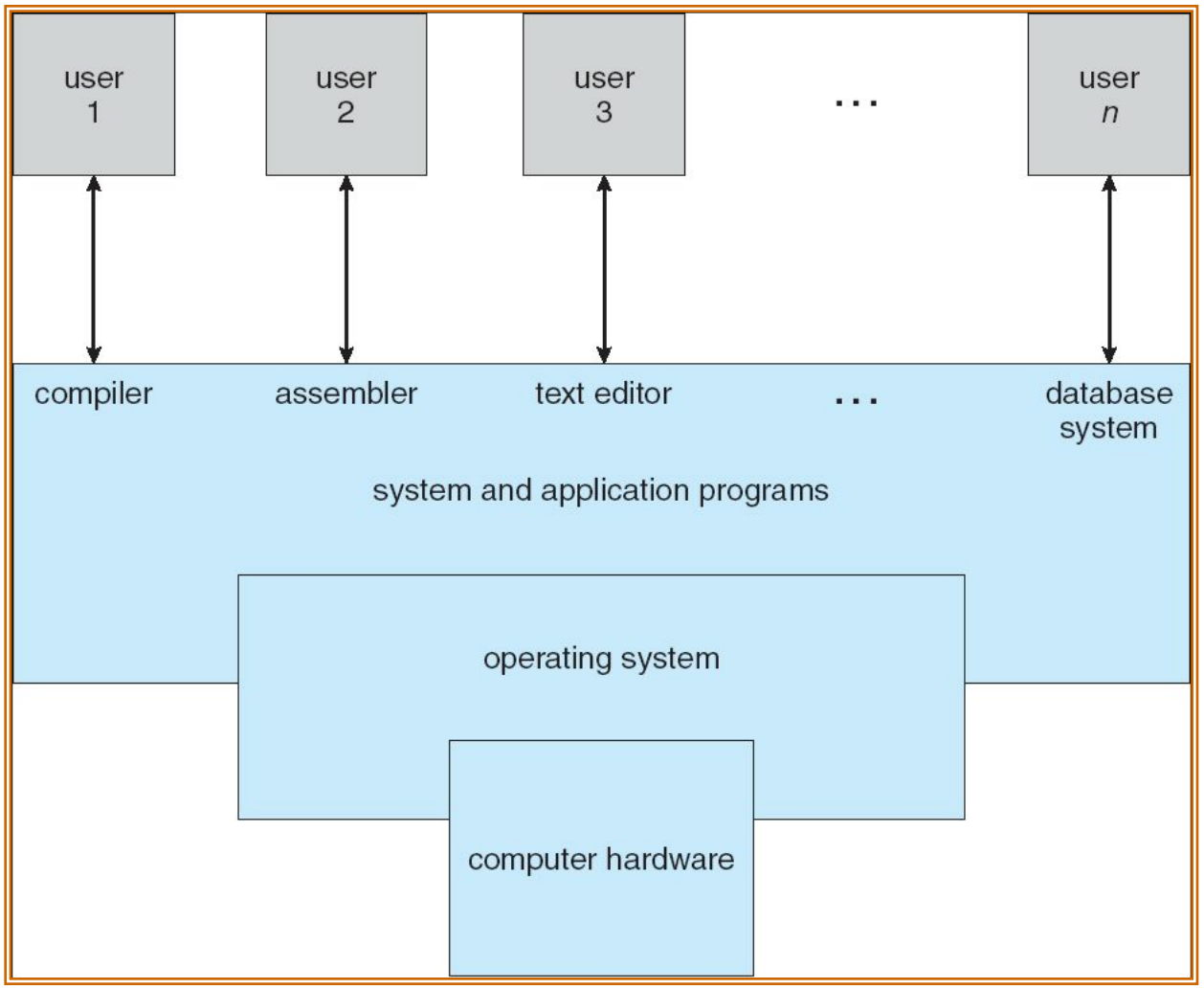
Operating System Services

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

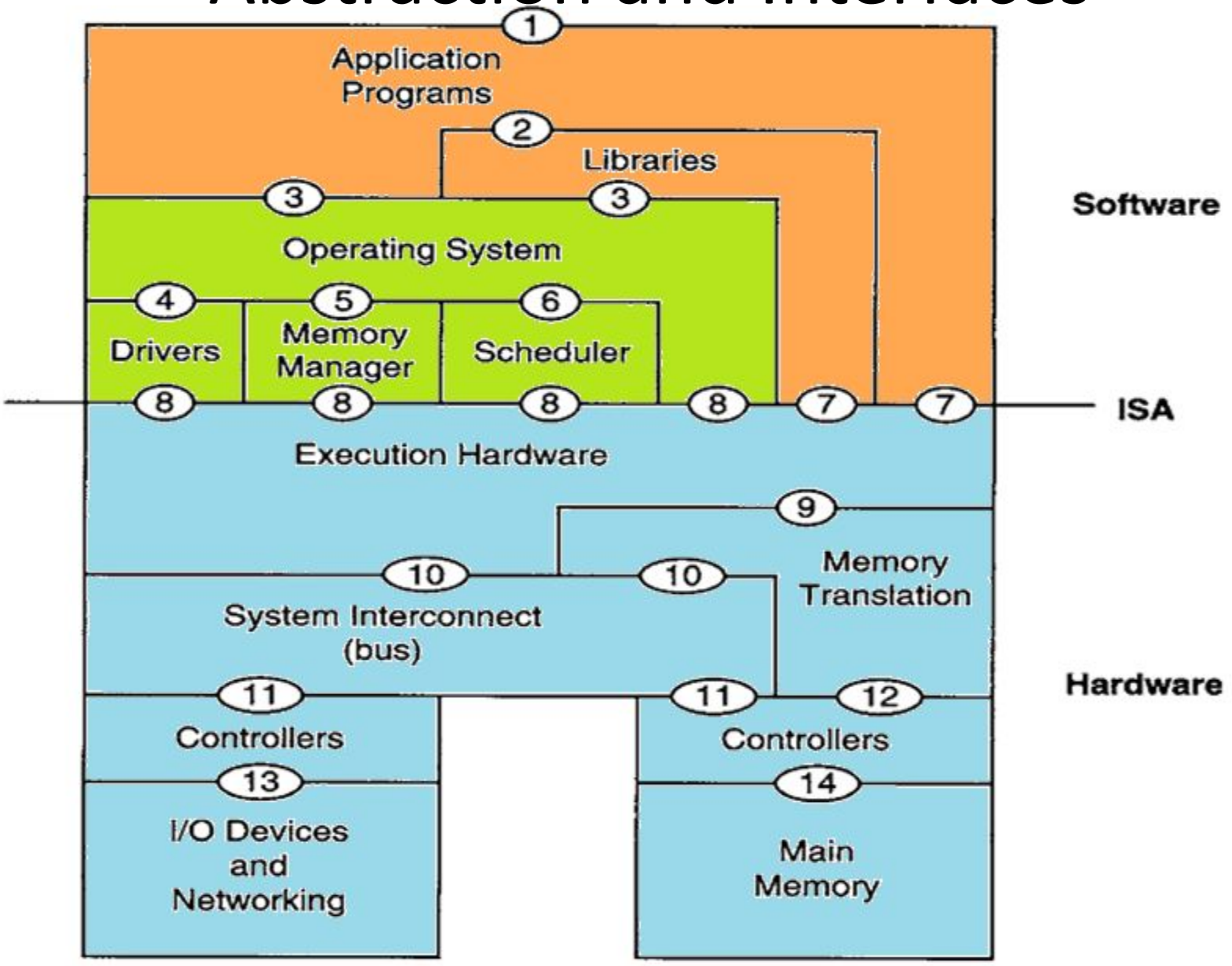
Major OS modules



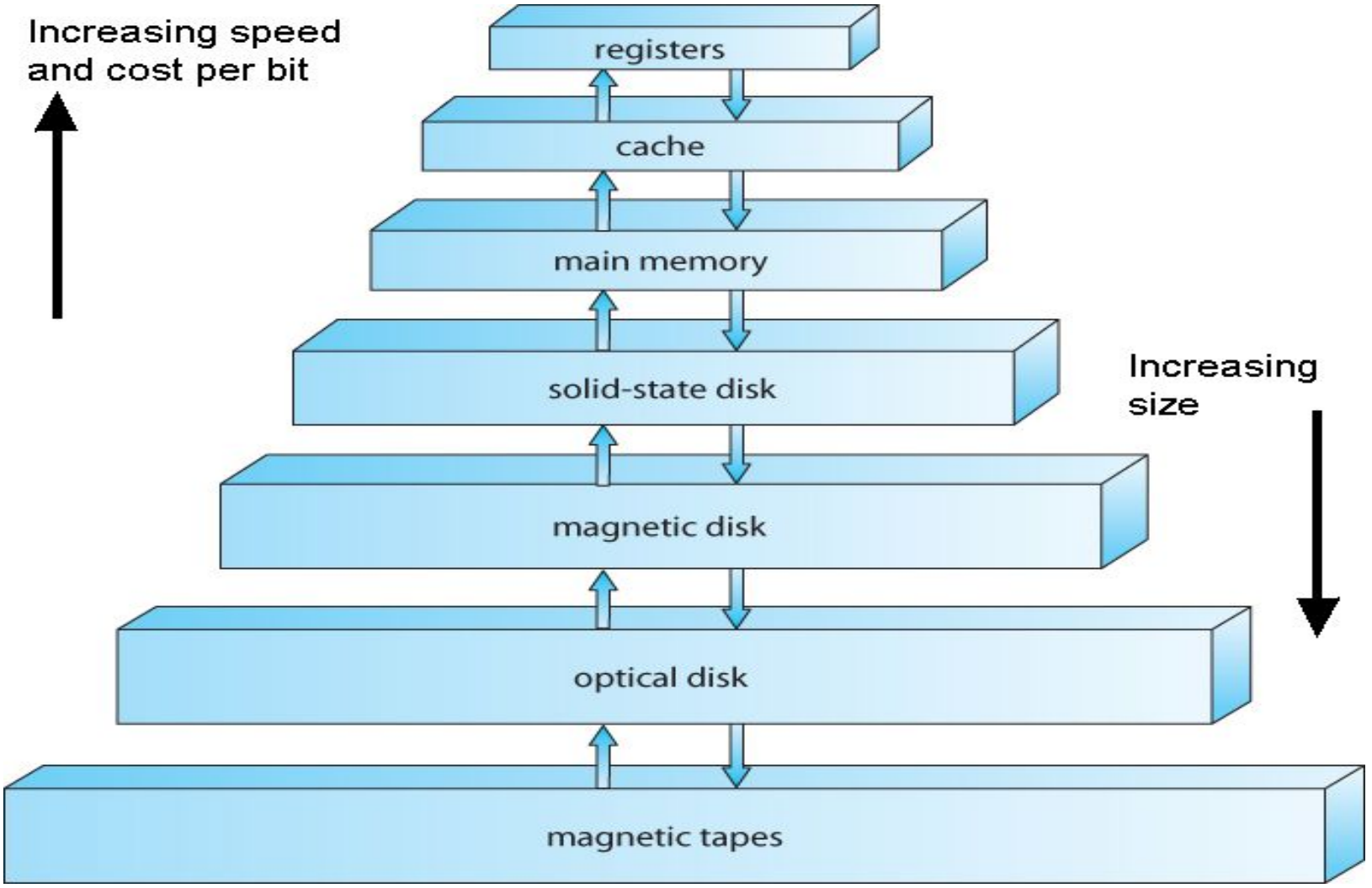
Four Components of a Computer System



Computer System Architecture: Abstraction and Interfaces



Storage Structure



Memory – Array of words

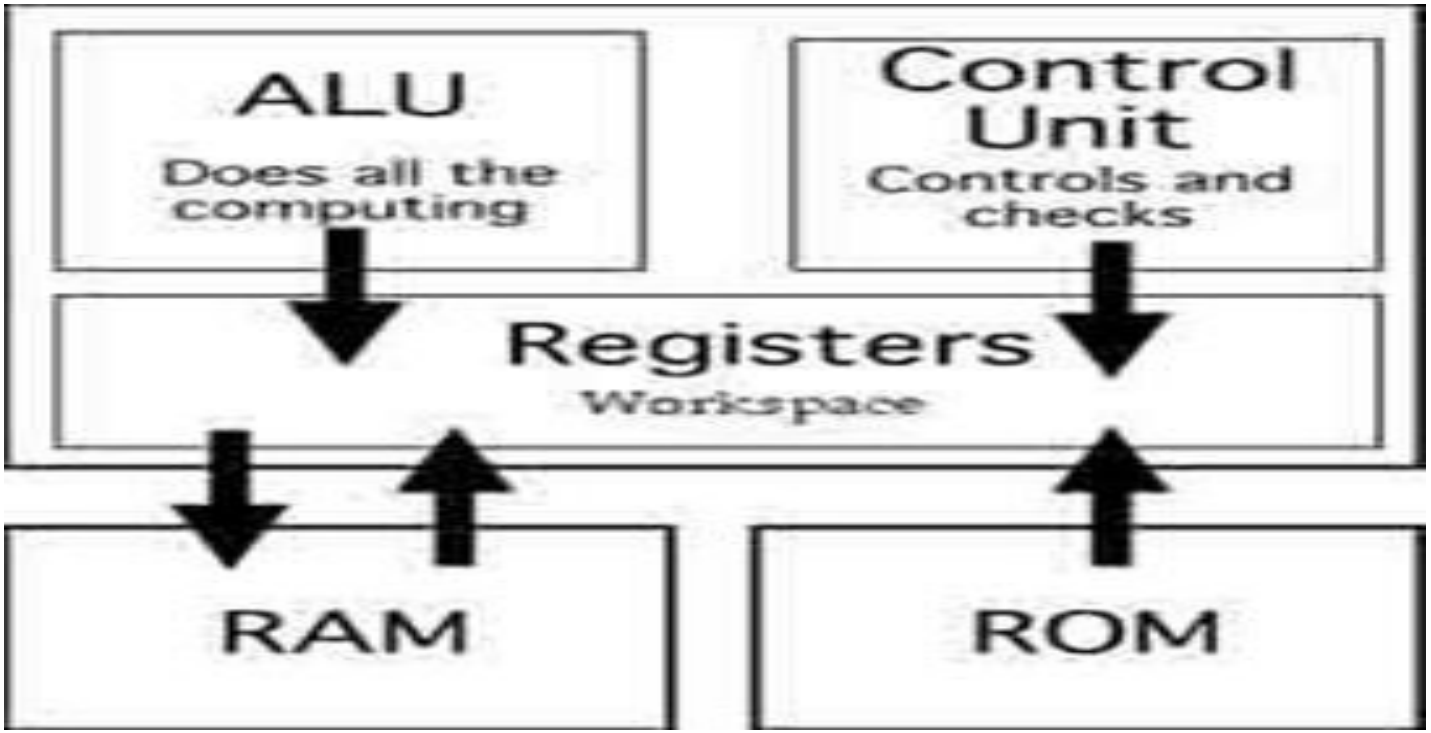
- Word - Largest size of data that can be transferred in and out of the memory.
- Interaction occurs through LOAD and STORE

Main Memory – rewritable memory (RAM)

- Implemented in a semiconductor technology called DRAM
- Volatile, very fast and expensive

Registers:

Very expensive and very fast.



Storage contd..

Secondary memory

- Electronic disk
- Magnetic disk
- Optical disk
- Magnetic tapes

Cache Memory

- Random access memory (RAM) that a computer microprocessor can access more quickly than it can access regular RAM.
- Most frequently used instructions are stored in the cache.

OS Abstraction

- The operating system provides a layer of abstraction between the user and the bare machine. Users and applications do not see the hardware directly, but view it through the operating system.

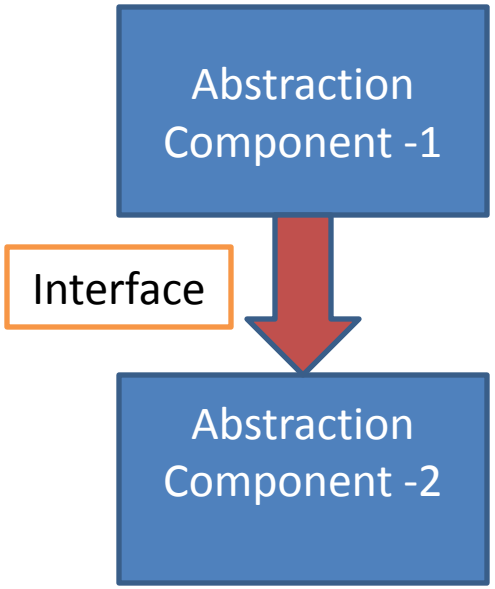
- This abstraction can be used to hide certain hardware details from users and applications. Thus, changes in the hardware are not seen by the user (even though the OS must accommodate them).

- This is particularly advantageous for vendors that want offer a consistent OS interface across an entire line of hardware platforms.
-

- Another way that abstraction can be used is to make related devices appear the same from the user point of view. For example, hard disks, floppy disks, CD-ROMs, and even tape are all very different media, but in many operating systems they appear the same to the user.

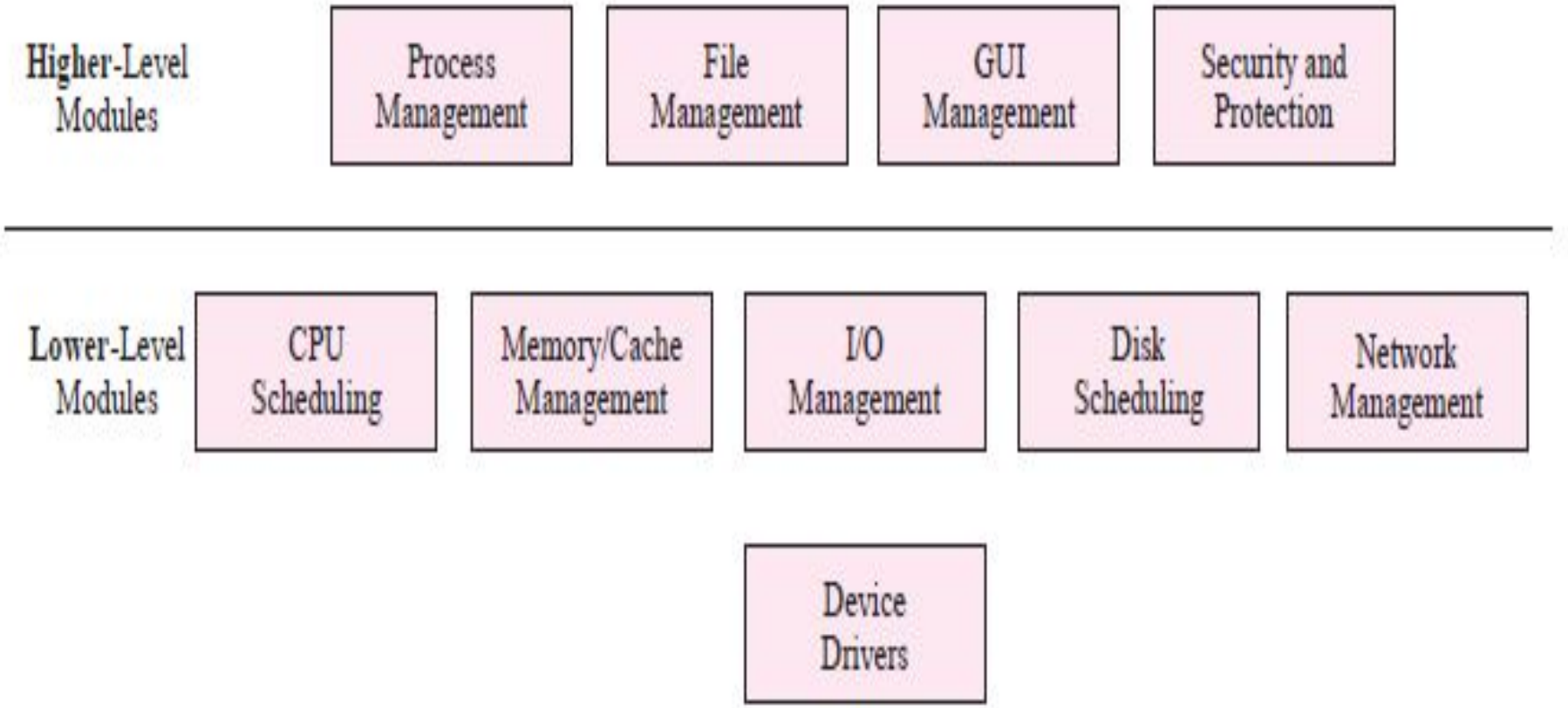
- Unix, and increasingly Windows NT, take this abstraction even further. From a user and application programmer standpoint, Unix is Unix regardless of the CPU make and model.

- Abstraction
  - Hide the implementation details of a component
  - Freedom to modify the implementation without affecting other underlying modules
  - Ex: int add(int a, int b) {
    return a + b; // Implementation details
    }
- Interface
  - To hide the details of lower level components
  - To enable the communication / data passing between the components without knowing details of their implementation
  - Ex: printf("%d, %d", add(5,10), add(10,10))



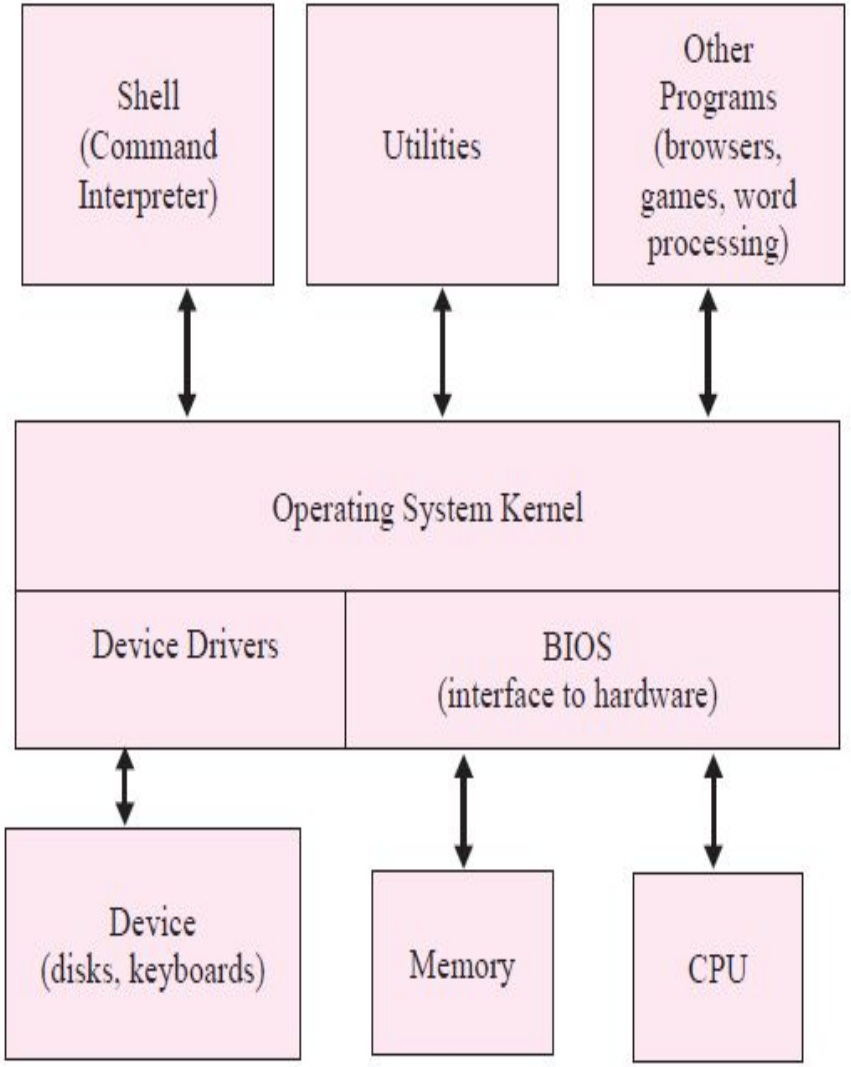
Take away:
If you introduce an interface,
you want to hide the implementation details below it

Major OS modules



Each of the responsibilities is implemented as separate Modules (**abstraction**) which are combined together through appropriate **interfaces** to formulate an Operating System software

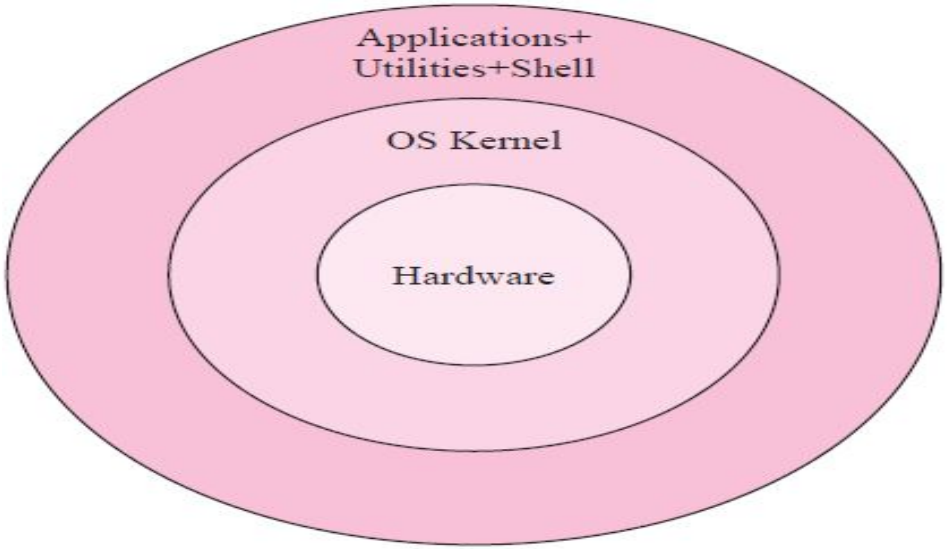
OS in relationship to hardware



Shell: programs that are not part of the OS core (or kernel), but work closely with the kernel to provide ease of use or access to system information. A **shell** or **command interpreter** is an example of a utility

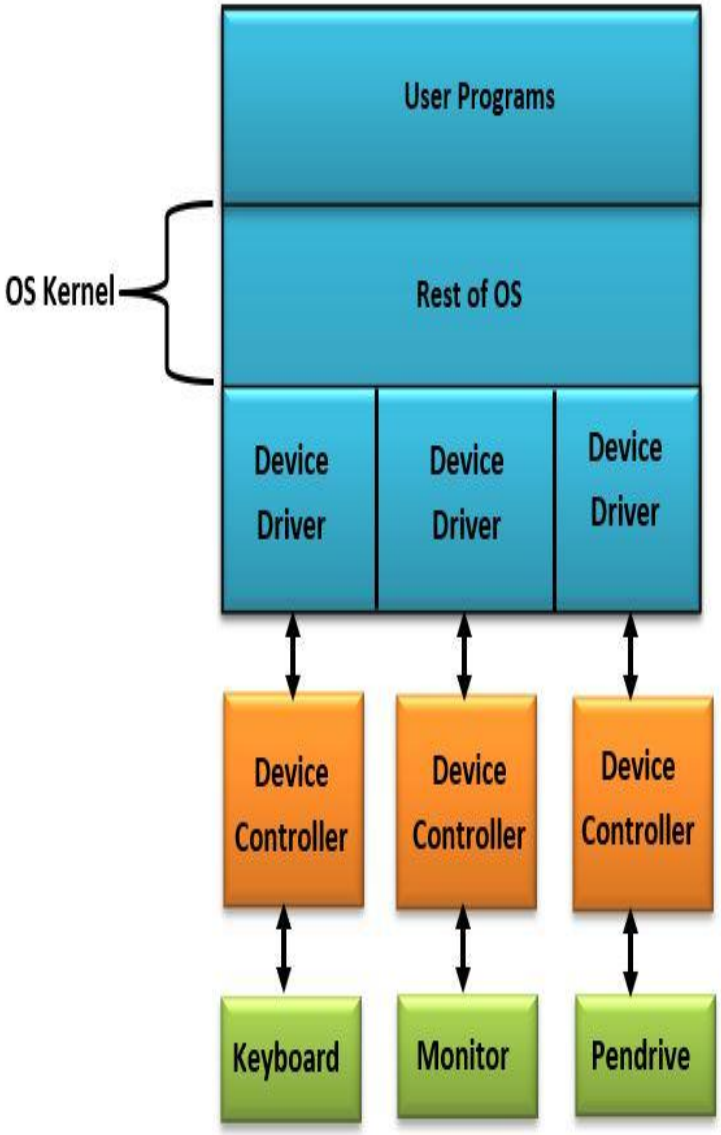
Service: Services are functions that the OS kernel provides to users, mostly through APIs via OS calls. Ex: file manipulation services (create, read, copy), memory allocation services (get, free)

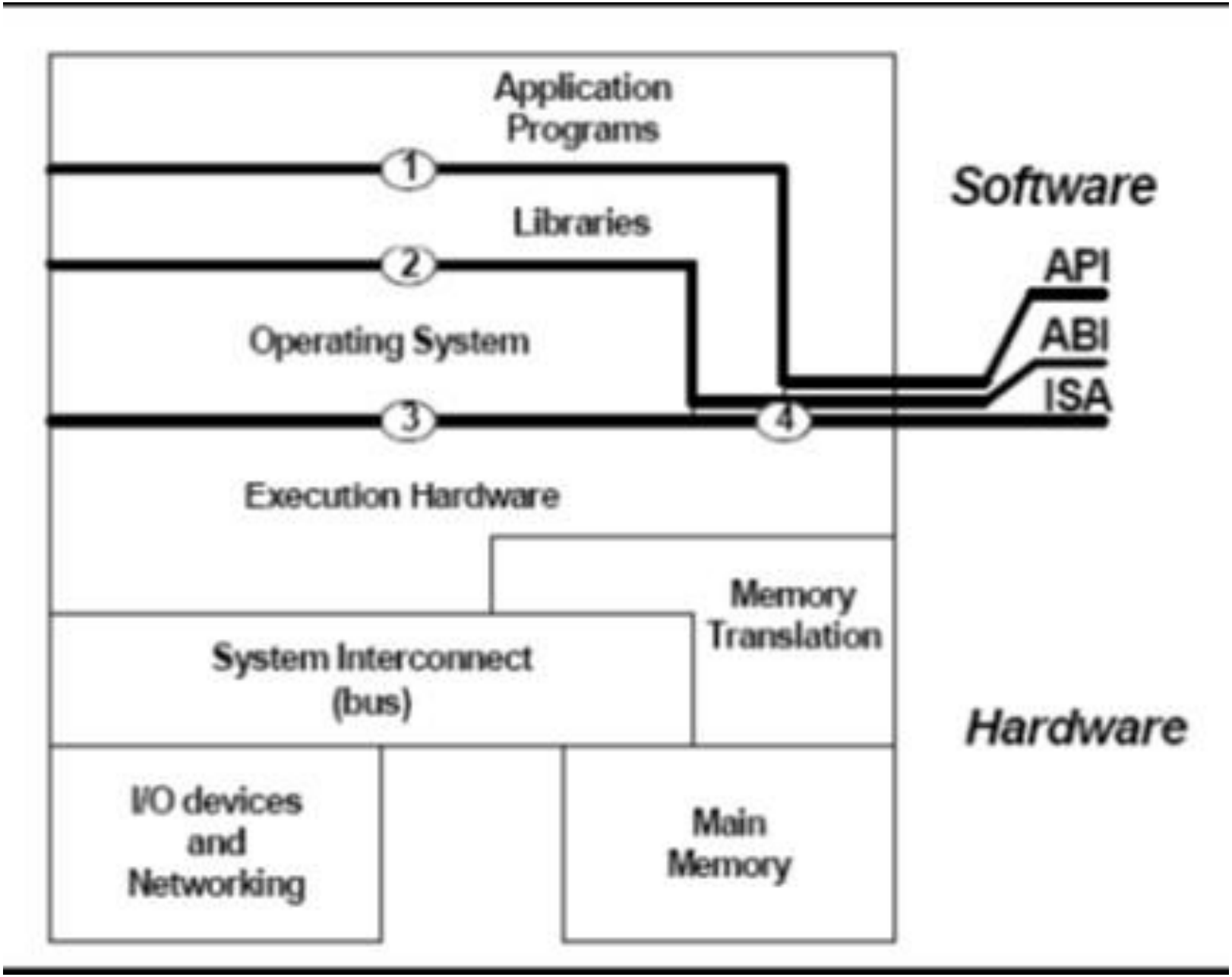
Kernel: refers to that part of the OS that implements basic functionality and is always present in memory



Layered or Levels approach

- Role of device, device controller and device driver
  - Device (Hardware)
    - A device is a piece of hardware connected t the main computer system hardware
      - Hard disks, DVDs, and video monitors
  - Device Controller (Hardware)
    - Usually, every device has a special electroni hardware interface, called a **device controll** which helps connect a device or a group of similar devices to a computer system throug **bus.**
      - hard disk controllers and video monito controllers
      - converts a serial bit stream to block of bytes and perform error correction as required
      - Record and save the data
  - Device driver (Software)
    - A device driver is a software routine that is p of the OS, and is used to communicate with and control a device through its device controller.
      - works as a translator between the hardware device and the application or the operating system that uses it




Application Programs
1
Libraries
2
Operating System
3
4
Execution Hardware
Memory Translation
System Interconnect (bus)
I/O devices and Networking
Main Memory
Software
API
ABI
ISA
Hardware

What is the role of the OS?

Role #1: abstract resources

What is a resource?

- Anything valuable (e.g., CPU, memory, disk)

Advantages of standard library

- Allow applications to reuse common facilities
- Make different devices look the same
- Provide higher-level abstractions

What is the role of the OS?

Role #2: Resource coordinator (I.e., manager)

Advantages of resource coordinator

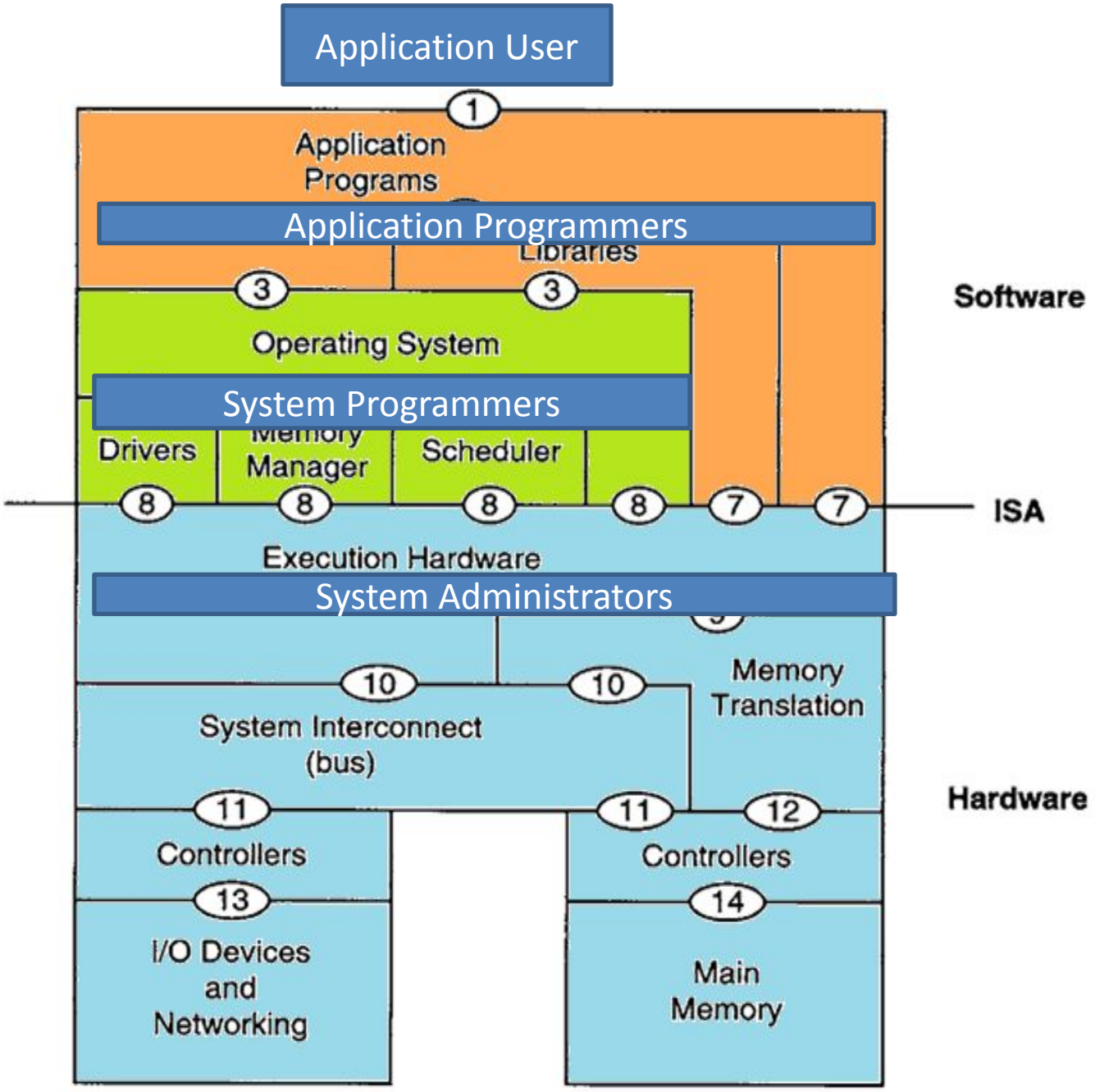
- Virtualize resources so multiple users or applications can share
- Protect applications from one another
- Provide efficient and fair access to resources

- Desirable qualities of an OS
  - Certain OS parts those handle the user and application program interaction should never be exposed to the users
    - Abstraction and Interface
  - OSs should provide new features and be easier to work with.
    - Updated periodically
  - OSs must provide support for old features
    - Backward compatibility
  - OSs must be able to connect, adapt and handle new devices that are not yet available and not even have been thought of when the OS was created
    - Extendable

- User and System view of an OS
  - User View: How users or programs utilize the OS?
    - Type of users
      - Application Users / End Users
      - Application Programmers
  - System View: How the OS software actually does the required action?
    - how it gets keystrokes, separates out special ones like shift, and makes them available to the user or program.
    - Type of Users
      - System programmers
      - System Administrators

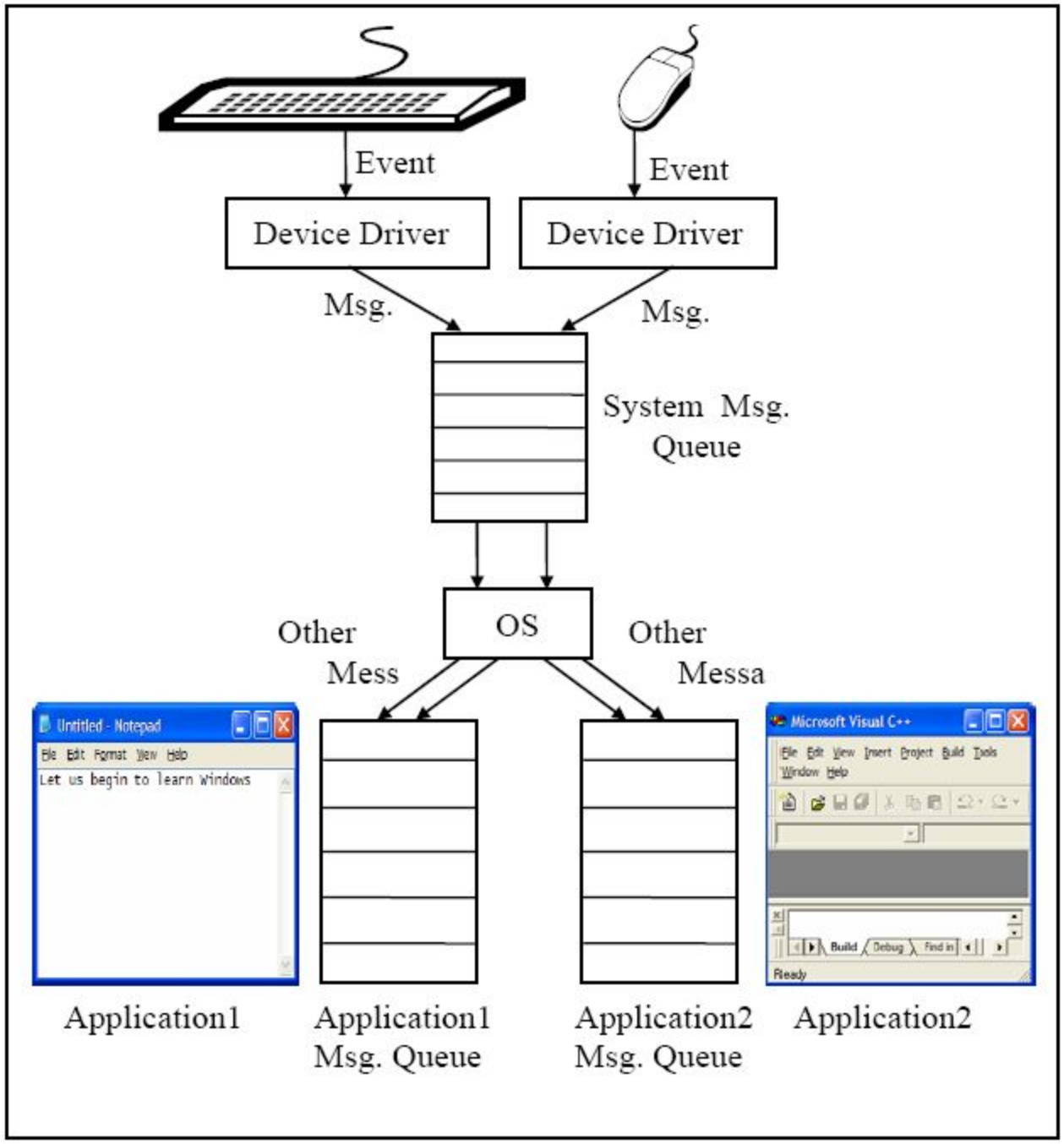
- User View – Types of Users
  - Application User / End User
    - people who use (or run) application or system programs
      - expect a quick, reliable response (to keystrokes or mouse movement
  - Application Programmers
    - people who write application programs
      - system calls or an API (application program interface)
- System View –Types of Users
  - Systems Programmers
    - these are the people who write software—either programs or components—that is closely tied to the OS.
      - have a detailed understanding of the internal functioning of the OS
      - A utility that shows the status of the computer's network connection or an installable driver for a piece of hardware are examples of systems programs.
  - System Administrators
    - people who manage computer facilities, responsible for installing and upgrading the OS, as well as other systems programs and utilities

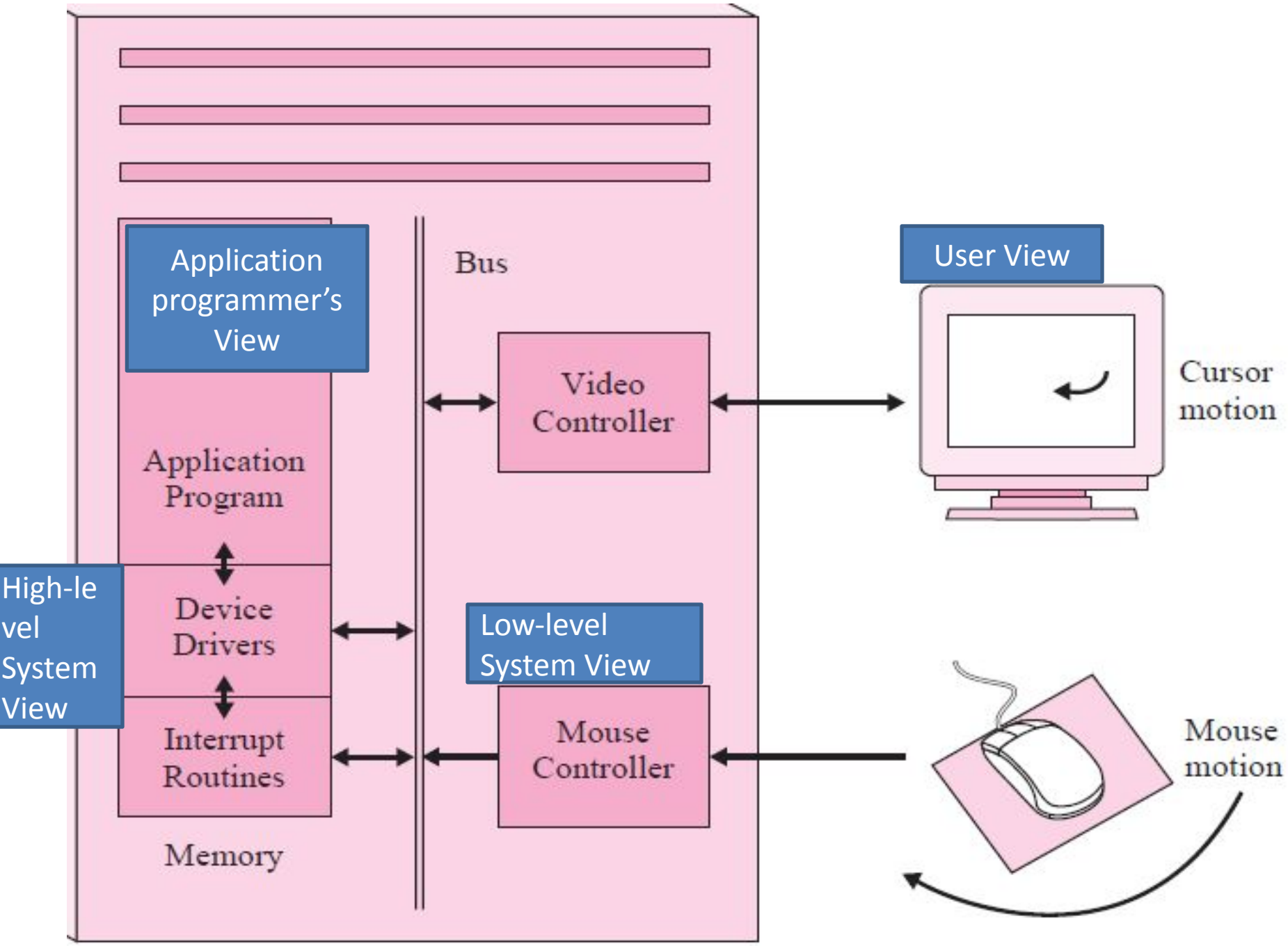
End Users	Easy to use and learn Adapts to user's style of doing things Lively response to input Provides lots of visual cues Free of unpleasant surprises (e.g., deleting a file without warning) Uniform ways to do the same thing (e.g., moving an icon or scrolling down a window—in different places) Alternative ways to do one thing (e.g., some users like to use the mouse, others like to use the keyboard)
Application Programmers	Easy to access low-level OS calls by programs (e.g., reading keystrokes, drawing to the screen, getting mouse position) Provide a consistent programmer view of the system Easy to use higher-level OS facilities and services (e.g., creating new windows, or reading from and writing to the network) Portability to other platforms
Systems Programmers	Easy to create correct programs Easy to debug incorrect programs Easy to maintain programs Easy to expand existing programs
System Managers and Administrators	Easy addition or removal of devices such as disks, scanners, multimedia accessories, and network connections Provide OS security services to protect the users, system, and data files Easy to upgrade to new OS versions Easy to create and manage user accounts Average response is good and predictable System is affordable


Application User
1
Application Programs
Application Programmers
Libraries
3
3
Operating System
System Programmers
Drivers
Memory Manager
Scheduler
8
8
8
8
7
7
Software
ISA
Execution Hardware
System Administrators
10
10
System Interconnect (bus)
Memory Translation
11
11
12
Controllers
Controllers
13
14
I/O Devices and Networking
Main Memory
Hardware

- Example : M**oving a mouse (and mouse cursor)**
  - When the pointing device is moved,
    - it generates a hardware event called an **interrupt** which the OS handles.
    - OS notes the movements of the mouse in terms of some hardware-specific units:
      - **System view:** Which application gets this mouse movement if there are multiple open windows?
        - » The mouse movements may need to be queued up if there are multiple movements before the application retrieves them.
        - » movements may even be lost if the OS is busy doing other things
      - **Low-level System view:** actual software **reading the mouse movements as number of pulses** which is part of the OS, and is called a **mouse device driver**
      - **High-level system view:** device driver reads the low-level mouse movement information and another part of the OS interprets it so that it can be converted into a higher-level system view,
        - » How is the mouse movements info presented to the application programmer?
      - **Application programmers' view:** How do I get the mouse movement information in order to use it and display it in my application?
      - **User view:** user's view is that the cursor will smoothly move on the screen and that as the mouse moves greater distances faster, the screen movement will appear faster too.

System View


Event
Event
Device Driver
Device Driver
Msg.
Msg.
System Msg.
Queue
OS
Other
Mess
Other
Messa
Untitled - Notepad
File Edit Format View Help
Let us begin to learn Windows
Microsoft Visual C++
File Edit View Insert Project Build Tools
Window Help
Build
Debug
Find in
Ready
Application1
Application1
Msg. Queue
Application2
Msg. Queue
Application2


Application programmer's View
Bus
User View
Video Controller
Cursor motion
Application Program
Device Drivers
Interrupt Routines
High-level System View
Low-level System View
Mouse Controller
Mouse motion
Memory

- Example : Files
  - **End user's view:** File names
    - Can file names contain spaces? How long can they be? Are upper and lowercase letters allowed? Are they treated as different or the same characters? How about non-English characters or punctuation?
  - **Application programmer's view:** the file system is a frequently used, critical part of the system
    - commands for creating a new file, using an existing file, reading or appending data to a file, and other file operations.
  - **System view:** file system is so large it is usually divided into subparts:
    - file naming and name manipulation (directory services),
    - file services such as locating and mapping a file name to its data (file allocation and storage),
    - trying to keep parts of open files in main memory to speed up access to its data (file buffering and caching), and
    - the actual management of the storage devices (disk scheduling).

History of the OS

Two distinct phases of history

- Phase 1: Computers are expensive
  - Goal: Use computer's time efficiently
  - Maximize throughput (I.e., jobs per second)
  - Maximize utilization (I.e., percentage busy)
- Phase 2: Computers are inexpensive
  - Goal: Use people's time efficiently
  - Minimize response time

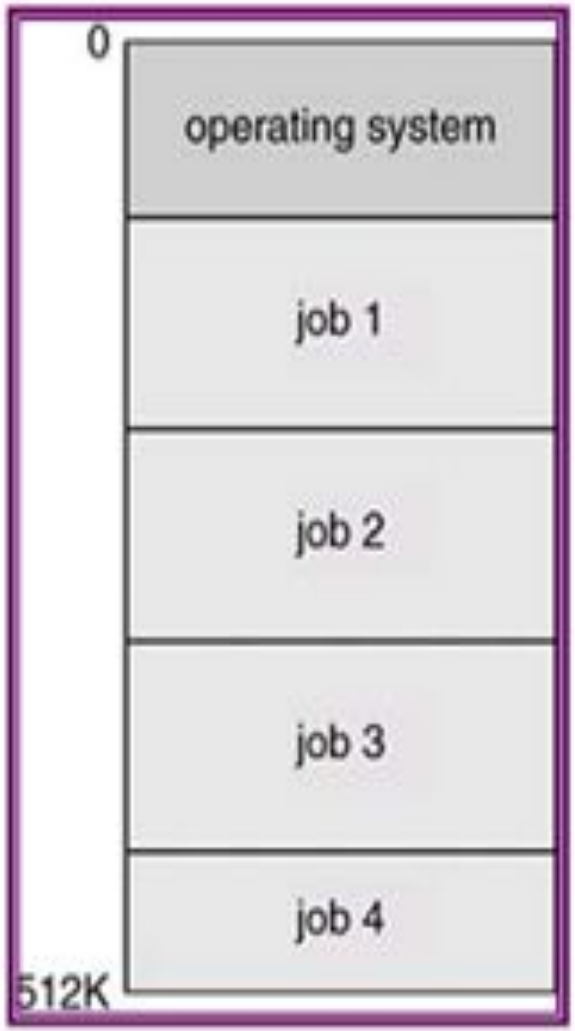
- **Single-tasking OS**
  - OS runs a single process at a time
    - Example: CP/M and MS-DOS
    - Handling I/O, starting and terminating programs
    - Fairly simple Memory management
    - no need for CPU scheduling
  - Supports batch job
    - have no live user so rapid response is not a requirement
    - less context switching is needed and more time is spent on productive computing

Single-user single-tasking Batch processing system

- User prepare a job and submit it to a computer operator
- User get output some time later
- No interaction between the user and the computer system
- Operator batches together jobs with similar needs to speedup processing
- Task of OS: automatically transfers control from one job to another.
- OS always resident in memory
- Disadvantages of one job at a time:
  - CPU idle during I/O
  - I/O devices idle when CPU busy

- **Multitasking or Multiprogramming** OS
  - OS will control multiple processes running concurrently
    - CPU scheduling component is used to choose which of the **ready to run** processes to run next
    - The CPU can switch to run another process while I/O is performed.
    - **Context Switching:** Changing from one running process to run another process is known as **context switching**
      - entire CPU state must be saved on process-1's **Process Control Block (PCB)**, execute the process-2 for a stipulated period of time, save the CPU state of process-2 in PCB of 2, switch back to process-1 and resume from stored CPU state by loading PCB-1
    - Supports both interactive and batch jobs
      - Interactive jobs are processes that handle a user interacting directly with the computer through mouse, keyboard, video monitor display, and other interactive I/O devices whereas batch jobs do not support user interaction with computers.
  - Advantages
    - Improve processor utilization by keeping the CPU busy while I/O is performed
      - When a process waiting for I/O, the CPU will execute other process

- Multiprogramming needed for efficiency
  - Single program cannot keep CPU and I/O devices busy at all times
  - Multiprogramming organizes jobs (code and data) so CPU always has one to execute
  - A subset of total jobs in system is kept in memory
  - One job selected and run via job scheduling
  - When it has to wait (for I/O for example), OS switches to another job



- Multi-programming systems demand
  - Job scheduling
    - To decide which jobs in a job pool should be brought into memory
  - Memory management
    - Allocate memory to many jobs
  - CPU scheduling
    - Choose the jobs in memory that are in ready to run state
  - Allocation of devices
    - If more than one job is competing for resources
  - Jobs that are running concurrently should not affect the other.

Timesharing Systems

- (multitasking) is logical extension in which CPU switches jobs so frequently that users can interact with each job while it is running, creating interactive computing
  - Response time should be < 1 second
  - Each user has at least one program executing in memory i.e. process
  - If several jobs ready to run at the same time need CPU scheduling to decide the job to be executed
  - If processes don’t fit in memory, swapping moves them in and out to run
  - Virtual memory allows execution of processes not completely in memory

Multiprocessor system

- Also known as parallel systems or tightly coupled systems
- More than one processor in close communication, sharing computer bus, clock, memory, and usually peripheral devices
  - Communication usually takes place through the shared memory.
- Advantages
  - Increased throughput: speed-up ratio with N processors < N
  - Economy of scale: cheaper than multiple single-processor systems
  - Increased reliability: graceful degradation, fault tolerant

Multiprocessor system

- Symmetric multiprocessing (SMP)
  - Each processor runs an identical copy of the operating system.
  - All processors are peers: any processor can work on any task
  - OS can distribute load evenly over the processors.
  - Most modern operating systems support SMP
- Asymmetric multiprocessing
  - Master-slave relationship: a master processor controls the system, assigns works to other processors
  - Each processor is assigned a specific task
    - Don't have the flexibility to assign processes to the least-loaded CPU
  - More common in extremely large systems

- Clustered System
  - Clustered systems are typically constructed by combining multiple computers into a single system to perform a computational task distributed across the cluster.
  - Multiprocessor systems on the other hand could be a single physical entity comprising of multiple CPUs.
  - A clustered system is less tightly coupled than a multiprocessor system.
  - Clustered systems communicate using messages, while processors in a multiprocessor system could communicate using shared memory.
  - In order for two machines to provide a highly available service, the state on the two machines should be replicated and should be consistently updated.
  - When one of the machines fail, the other could then take-over the functionality of the failed machine.

KINDS OF OS & PROPERTIES

Properties of the following types of operating systems:

- a. Batch
- b. Interactive
- c. Time sharing
- d. Real time
- e. Network
- f. Parallel
- g. Distributed
- h. Clustered
- i. Handheld

KINDS OF OS & PROPERTIES

- a. **Batch**. Jobs with similar needs are batched together and run through the computer as a group by an operator or automatic job sequencer. Performance is increased by attempting to keep CPU and I/O devices busy at all times through buffering, off-line operation, spooling, and multiprogramming. Batch is good for executing large jobs that need little interaction; it can be submitted and picked up later.

- b. **Interactive**. This system is composed of many short transactions where the results of the next transaction may be unpredictable. Response time needs to be short (seconds) since the user submits and waits for the result.

- c. **Time sharing**. This systems uses CPU scheduling and multiprogramming to provide economical interactive use of a system. The CPU switches rapidly from one user to another. Instead of having a job defined by spooled card images, each program reads its next control card from the terminal, and output is normally printed immediately to the screen.

KINDS OF OS & PROPERTIES

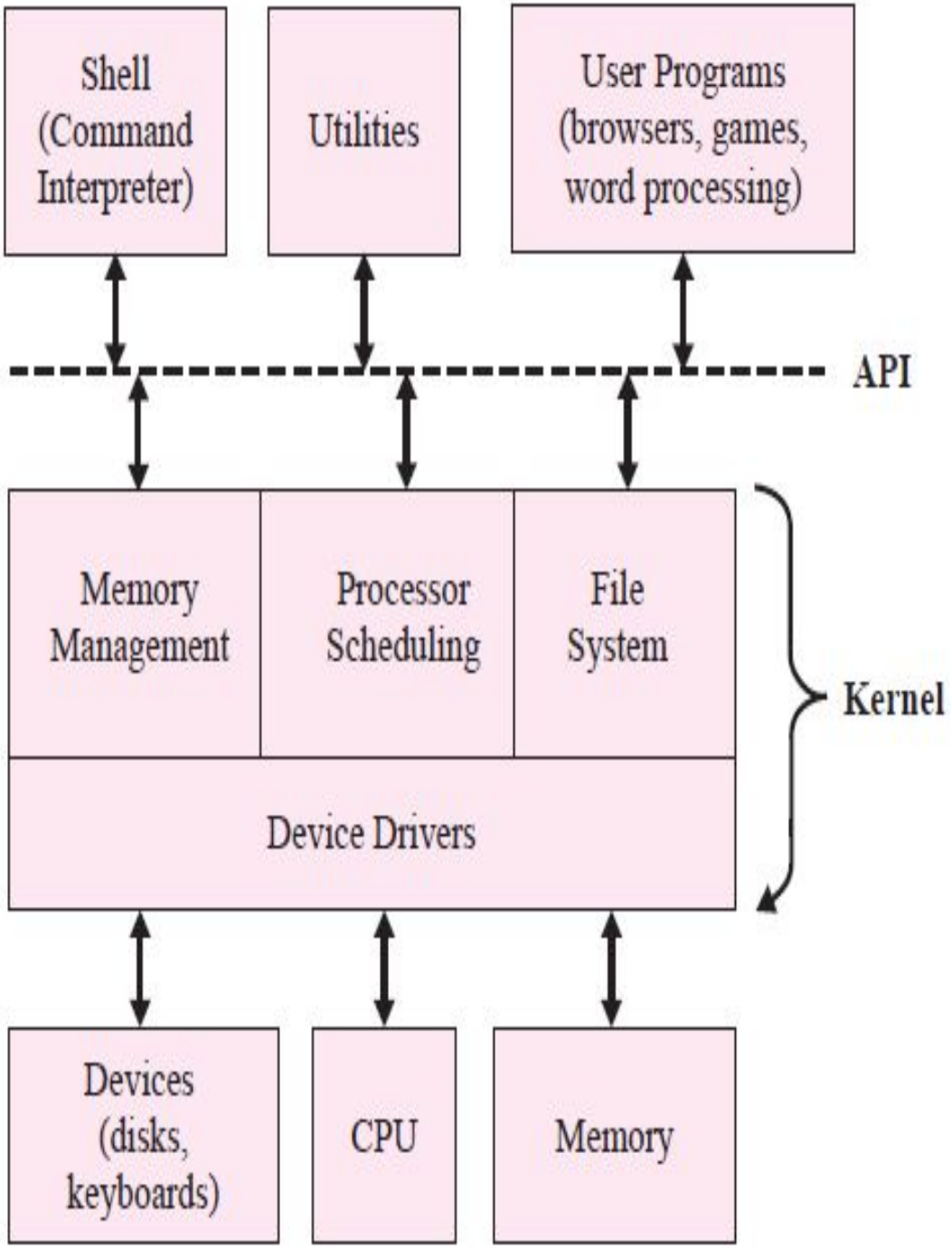
- d. **Real time**. Often used in a dedicated application, this system reads information from sensors and must respond within a fixed amount of time to ensure correct performance.
- e. **Network**. Provides operating system features across a network such as file sharing.
- f. **SMP**. Used in systems where there are multiple CPU's each running the same copy of the operating system. Communication takes place across the system bus.
- g. **Distributed**. This system distributes computation among several physical processors. The processors do not share memory or a clock. Instead, each processor has its own local memory. They communicate with each other through various communication lines, such as a high-speed bus or local area network.
- h. **Clustered**. A clustered system combines multiple computers into a single system to perform computational task distributed across the cluster.
- i. **Handheld**. A small computer system that performs simple tasks such as calendars, email, and web browsing. Handheld systems differ from traditional desktop systems with smaller memory and display screens and slower processors.

ARCHITECTURAL APPROACHES TO BUILDING AN OS

1. Monolithic single-kernel OS approach
2. Layered OS approach
3. Modular approach
4. Microkernel OS approach

- **Monolithic single-kernel OS approach**
  - written as a single program (no modules at all)
    - **kernel** or **monolithic kernel**
  - **Disadvantages**
    - As monolithic kernel size grew if more and more functionalities are added, percentage of main memory occupied by the OS is too large.
    - Lead to had more bugs,
    - Difficult to maintain,
    - Difficult to add features or to fix bugs
  - Solution
    - develop OSs based on a more modular, layered design

- **Layered OS approach**
  - modular OS approach that was developed was a **layered architecture.**
  - OS is divided into modules that were limited to a specific function such as processor scheduling or memory management.
  - The modules were grouped into layers of increasing abstraction
    - each layer provides a more abstract view of the system and relies on the services of the layers below it.
    - The layered approach would hide the peculiarities and details of handling hardware devices, and provide a common abstract view to the rest of the OS.
    - Thus, when new devices entered the marketplace, new device drivers could be added to the kernel without drastically affecting the other OS modules, which provide memory management, processor scheduling, and the file system interface.
  - **Disadvantage/Critic:**
    - OS design should return to a minimum amount of code in the kernel
  - **Solution**
    - Microkernel approach



Different Variations

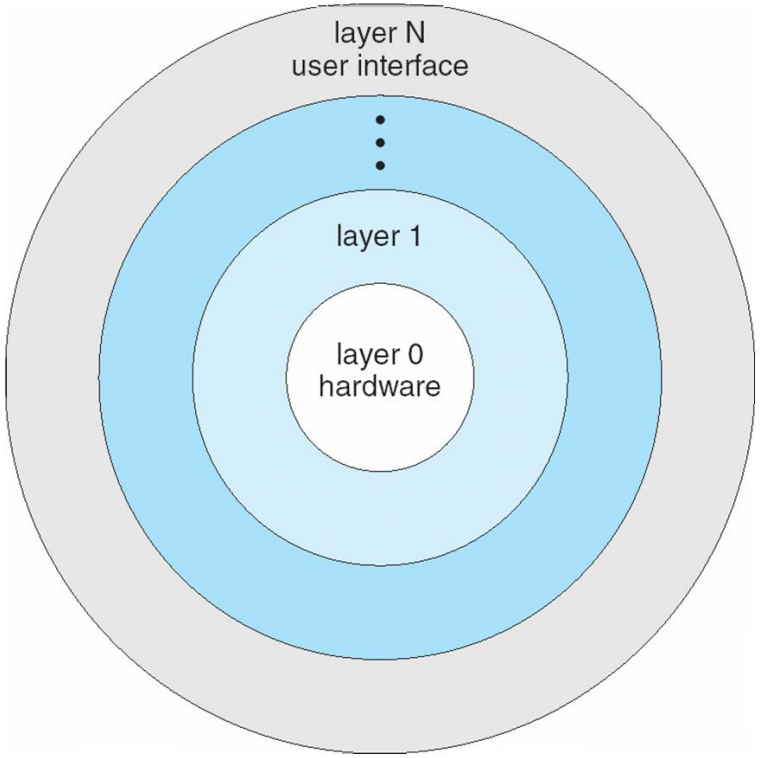
1. Allow modules at **layer *n*** to call only the modules in the next lower **layer *n-1***

2. allow modules **at layer *n*** to call modules at any of the lower l**ayers (*n-1, n-2,* and so on**).

3. allow **level *n* modules** to interact with other **level *n* modules**

Layered Approach

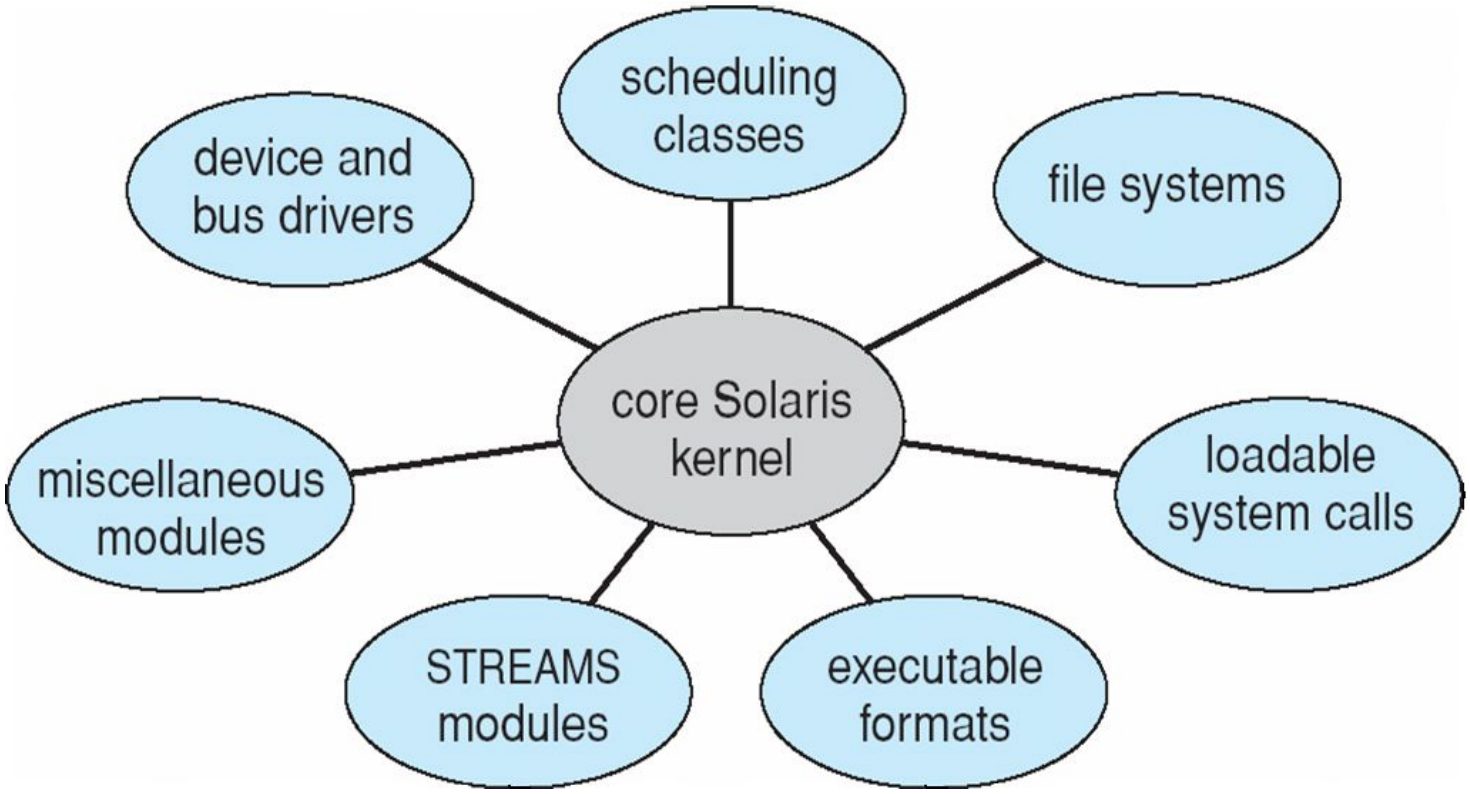
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level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



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 more flexible
  - Linux, Solaris, etc

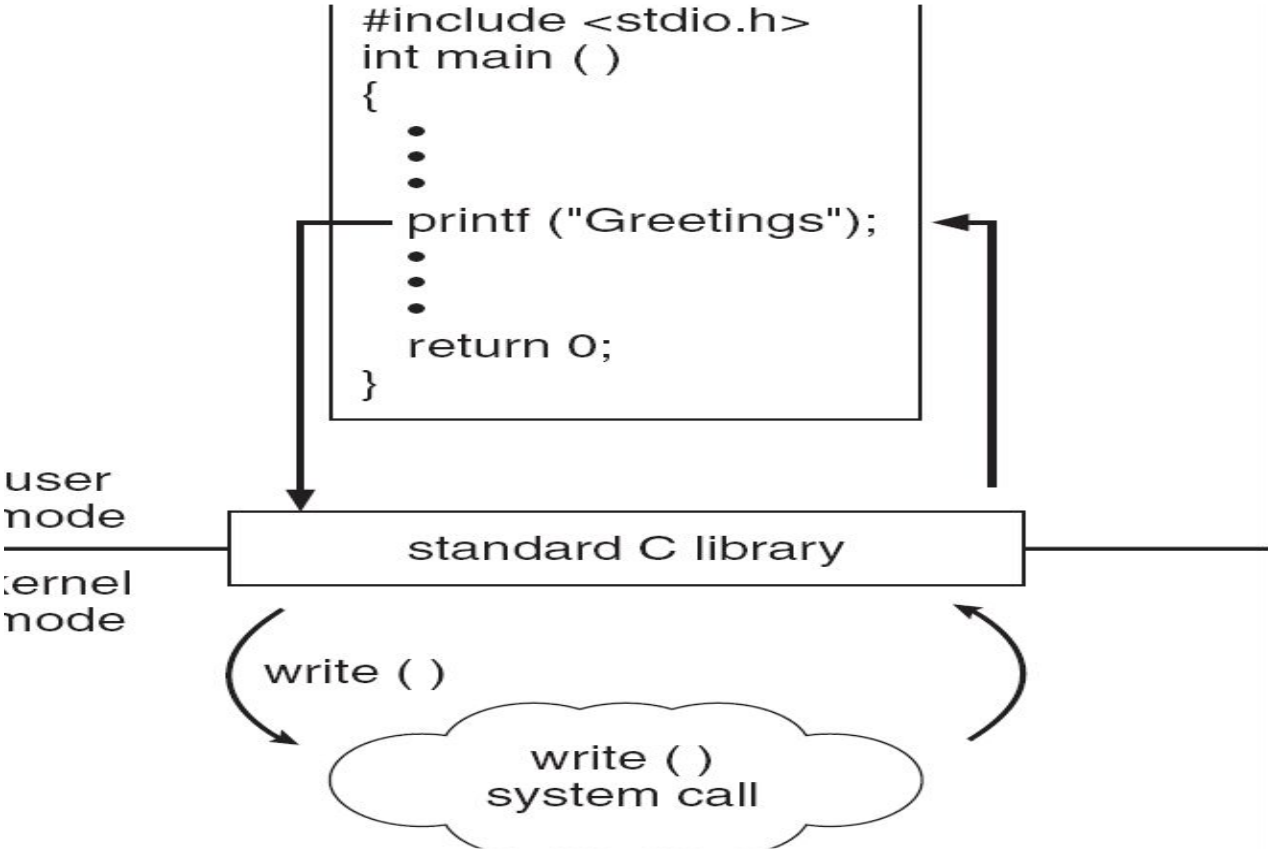
Solaris Modular Approach

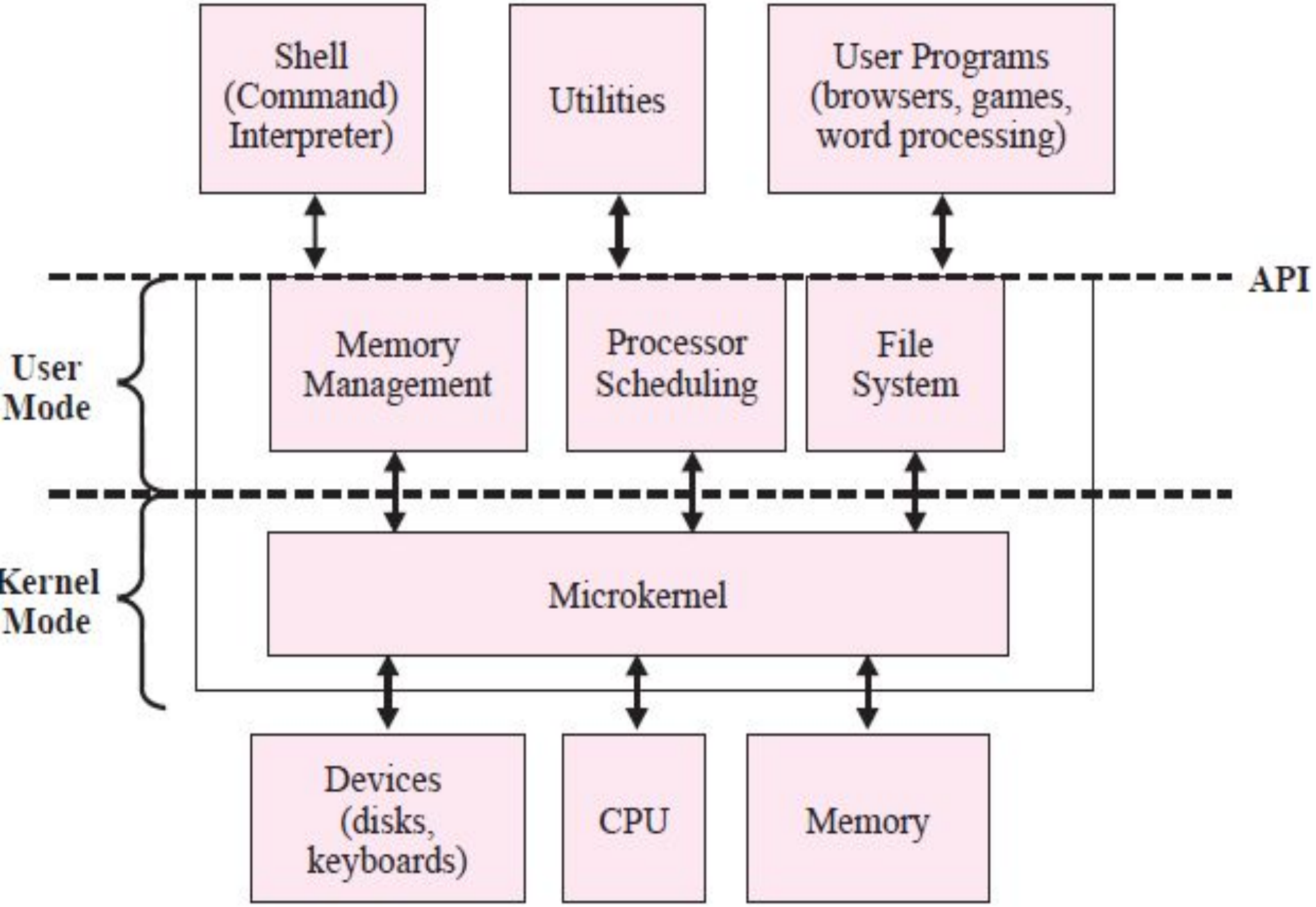


- **Microkernel OS approach**
  - **More robust** – As the amount of code that is running in supervisor mode is smaller
    - easier to inspect for flaws
    - easier to port a small microkernel to a new platform
  - Only basic functionality, usually **the interfaces to the various types of device drivers**, is included in the microkernel
    - **code that must run in supervisor mode** because it actually uses privileged resources such as protected instructions or accesses memory not in the kernel space
    - **Code running in protected mode** literally can do anything, so an error in this code **can do more damage** than code running in user mode
  - **Remainder of the OS functions** are still **part of the resident OS**, but they **run in user mode** rather than protected mode
  - **Make use of interrupts** to **make the necessary calls** from the **user mode portions** of the OS to the **supervisor mode portions**.
  - **Disadvantage/Critics:**
    - Makes a microkernel OS run more slowly due to context switching and not yet resolved

Standard C Library Example

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




Shell (Command) Interpreter)
Utilities
User Programs (browsers, games, word processing)
API
User Mode
Memory Management
Processor Scheduling
File System
Kernel Mode
Microkernel
Devices (disks, keyboards)
CPU
Memory

- OS Design Issues
  - **Minimalist**
    - only those things that really must go into the kernel (or microkernel) are included in the OS.
    - Other components may be added into **library routines** or as **"user" programs but not written by the user.**
    - **Claims:**
      - Load when it is really needed
      - Write and integrate new components
      - Easy to design
      - Elegant or cleaner
  - **Maximalist**
    - philosophy—to put most of the commonly used services in the OS
    - **Claims:**
      - Consistent look and feel