Chapter 3: Operating- System Structures

- An OS provides the environment within which programs are executed.
- There are several vantage points to view OSs.
  - Functional view (what it does)
  - Components view (Designers view)
  - Services view (Users/ Programmers view)
  - Structure view (How it is implemented)

Functional view

- Program execution and handling
  - Starting programs, managing their execution and communicating their results.
- I/O operations
  - Mechanisms for initiating and managing I/O
- File-system Management
  - Creating, maintaining and manipulating files
- Communications
  - Between processes of the same user
    - ✔ Such as sending result of input request to a user program
  - Between different users
- Exception detection and handling
  - Protection related issues
  - Safety in the case of power failures via backups.
  - Detecting undesirable state such as printers out of paper.

Functional view (...)

- Resource allocation
  - Includes processor and I/O scheduling, memory management
- Accounting
  - To track users usage of resources for billing and statistical reasons
- Protection
  - Maintaining integrity of user's data
  - Integrity checks to keep out unauthorized users
  - Maintaining logs of incorrect attempts

Components view

- Processes
- Storage
- I/O
  - Devices
  - files
- Protection
  - Users

Component view: Processes

- A process is
  - Dynamic entity created by the execution of the program.
  - Typically program + run time control information.
    - ✔ Control information includes memory maps, program counter value etc.
    - ✔ May also have a context information.
  - OS attempts treat all processes uniformly.
- Processes may play different roles
  - User processes
  - OS (system) processes
- A single process can spawn other processes
  - A computation requires typically many processes
    - ✔ Shell, one or more user processes, one or more system processes

Component view: Process management

- Operations:
  - Creation and termination
  - Suspension and resumption
    - Due to interrupts, context switches
  - Synchronizing processes
    - Making sure that a process that is waiting on an I/O waits till it is completed and does not wait forever, i.e., wakes-up soon after an I/O process terminates.
  - Communication
    - Between two processes enabling them to cooperate.
  - Deadlock detection and avoidance.

Component view: Storage management

- Managing main memory
  - Allocating main memory to active processes
    - ✔ Maintaining a map of allocated vs. free memory
  - De- allocating currently used memory to make a room for other processes.
- Managing secondary storage
  - Managing the free sectors/ tracks on the disk
  - Allocating this storage to programs
  - Scheduling access requests to the disk

Component view: I/O management

- Devices
  - Device drivers
  - Accepting an I/O request and invoking appropriate device driver
- Files
  - Non-volatile representation of users/system programs and data.
  - File systems
    - ✔ Support logical organization of data that the user might want to see
    - ✔ Map data onto the physical storage devices and orchestrate their access and update.
  - Operations
    - ✔ Creation, manipulation and deletion of files and directories
    - ✔ Moving files from primary to secondary storage while maintaining structure.
    - ✔ Interaction with the memory manager
    - ✔ Backup and protection

Component view: Protection

- Controlling the access of programs, processes, or users to the resources defined by the computer system.
  - Specification and enforcement

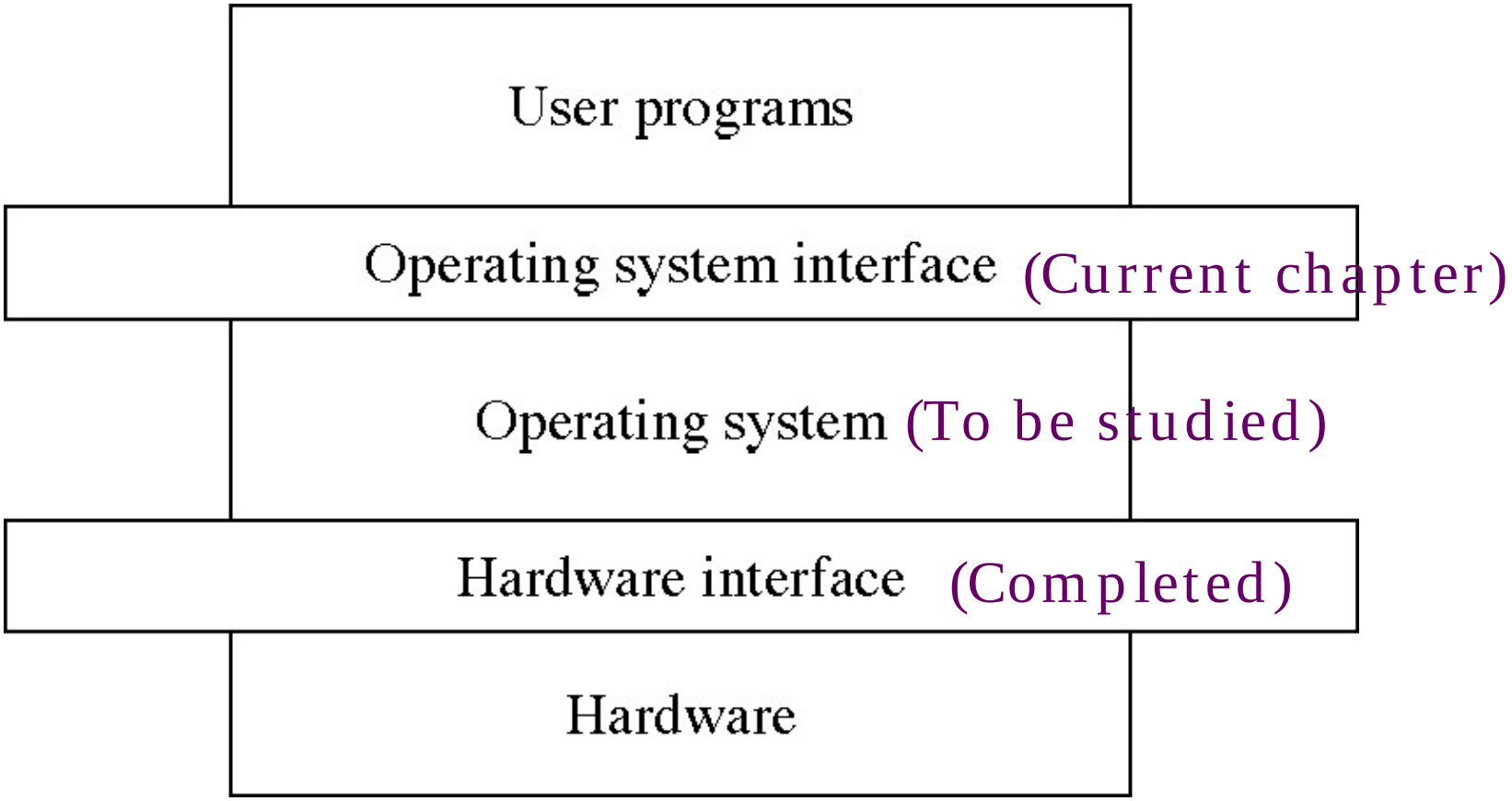
Services view

- Command interpreter
- Two levels of specification
  - System calls
    - The interfaces in which the processes invoke specific OS functions
  - System programs
    - May use capabilities not available through system calls
  - The interfaces at these levels can be standardized.
    - e.g network protocols, etc.

Services view: Command interpreters

- Interface between the user and the OS
- Can be a part of kernel or a separate process (shell)
- Striking differences between OSs
  - Ranges from GUIs to cryptic control card interpreters.
- Typical commands
  - Process creation and destruction
  - I/O handling and file system manipulation
  - Communication: interact with remote devices
  - Protection management: changing file/directory access control.

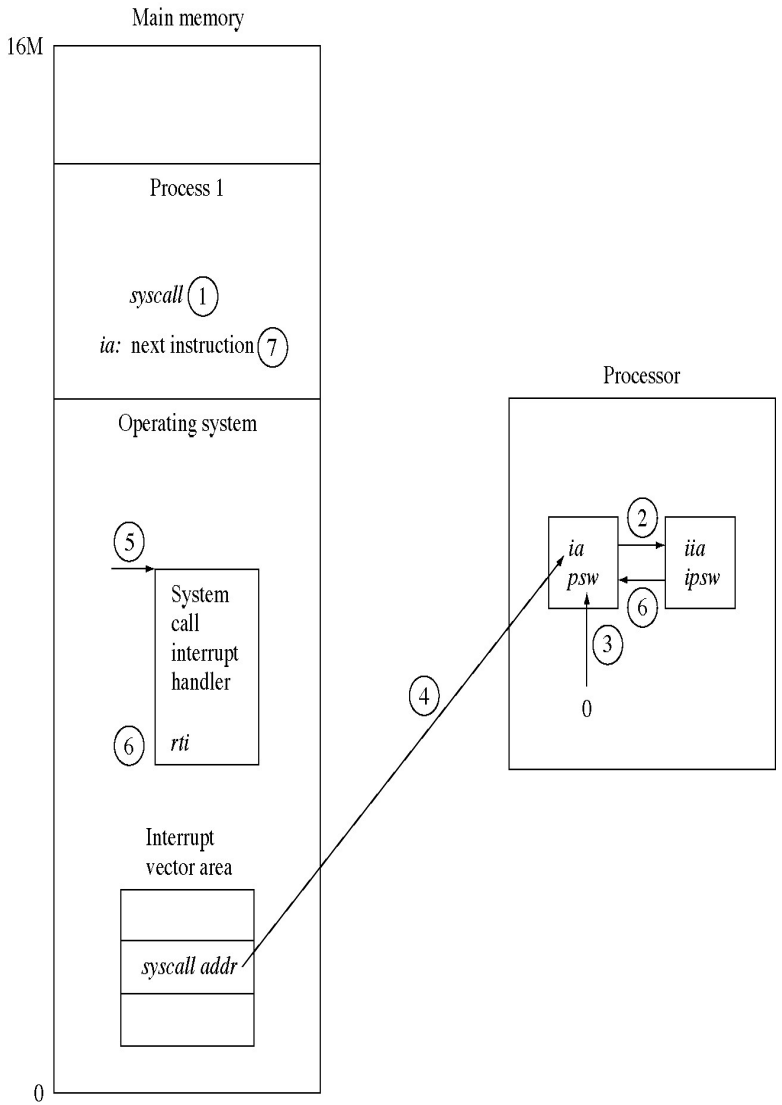
The OS Level Structure



System calls

- A special machine instruction
  - that causes an interrupt
  - various names: syscall, trap, svc
- Usually not generated by HLLs
  - but in assembly language functions
- The system calls are the instruction set of the OS virtual processor.

System call flow of control

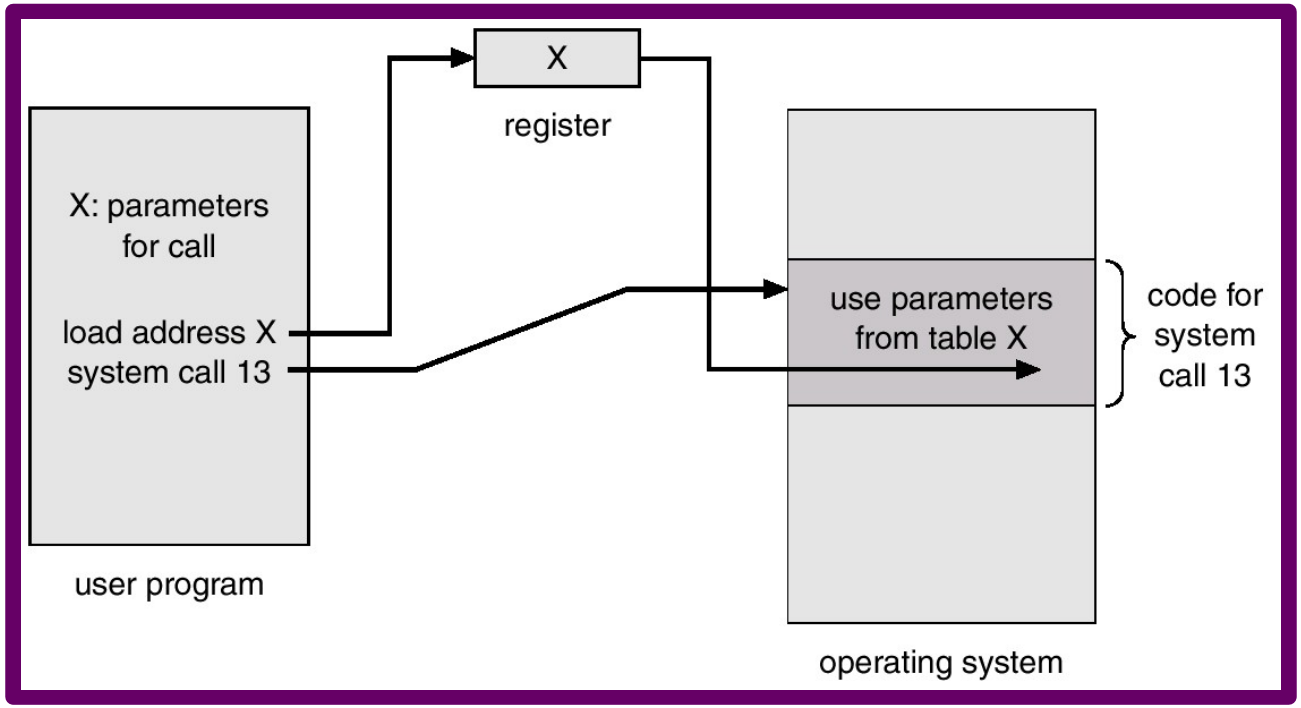


1) Program Executes System call
2) The hardware saves current ia (instruction address register) and psw in the iia (interrupt instruction address register) and ipsw (interrupt program status word) registers
3) The hardware converts puts 0 in PSW and interrupts are disabled.
4) The hardware loads the ia register with system call interrupt vector
5) Instruction execution continues at the beginning of interrupt handler.
6) The The system call handler completes and executes a return from interrupt instruction. This restores the ia and psw from iia and ipsw.
7) The process that executed the system call instruction continues at the instruction after the system call.

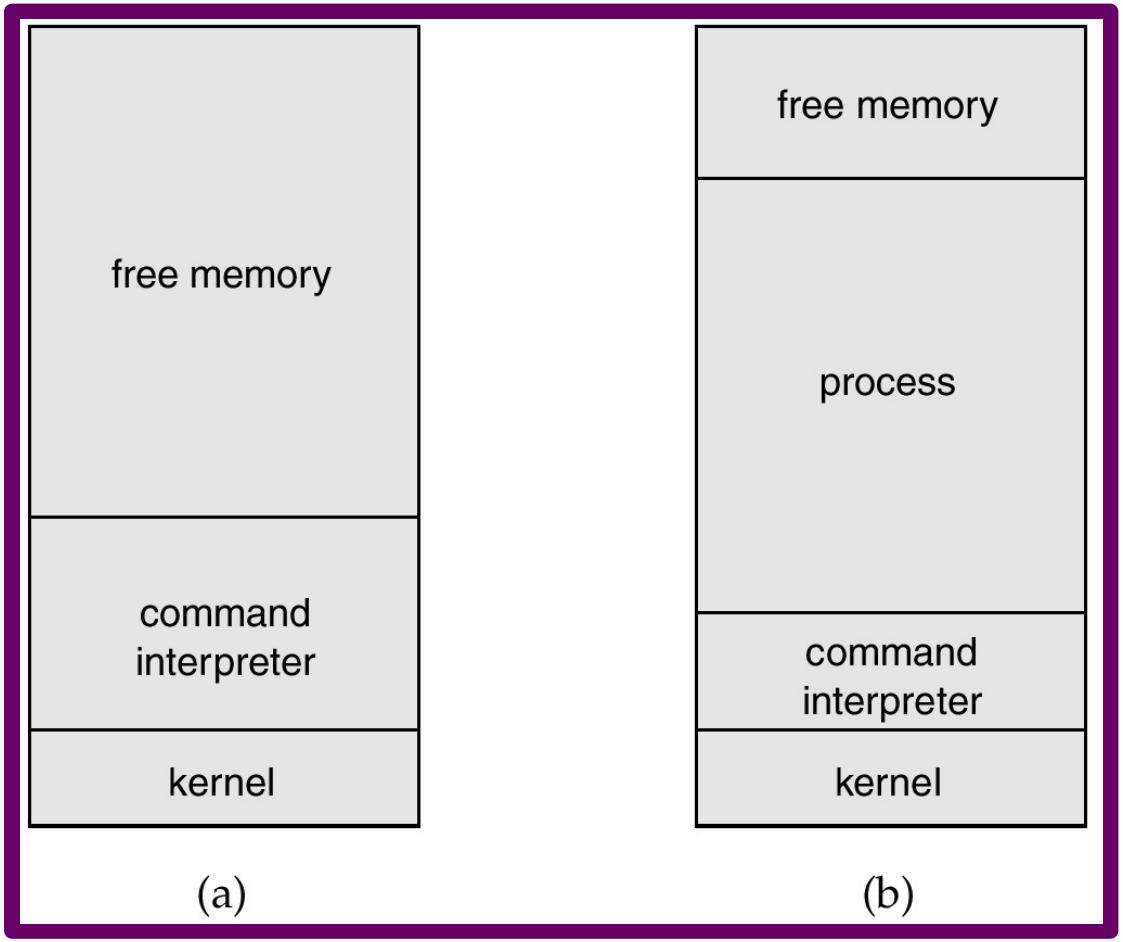
Services view: System Calls

- Interface between a process and OS
  - Arguments typically passed through registers, a memory block or on stack.
- Process job control
  - load, execute, end, abort, create/ terminate
  - get and set process attributes (priorities and allowable execution times)
  - Wait for time/ event, signal event, allocate memory
- File manipulation
  - create/ delete, open/ close, read/ write, reposition
  - get/ set file attributes (protection parameters, locations in directories,...)
- Device manipulation
  - Same set of calls as above (devices are treated as files)
- Information maintenance
  - Get/ set system data (time, memory/ CPU usage), process and device attributes
- Communications
  - Create/ delete links, send and receive messages
  - Transfer status information
  - Modes can be message passing or shared memory

Passing of Parameters As A Table



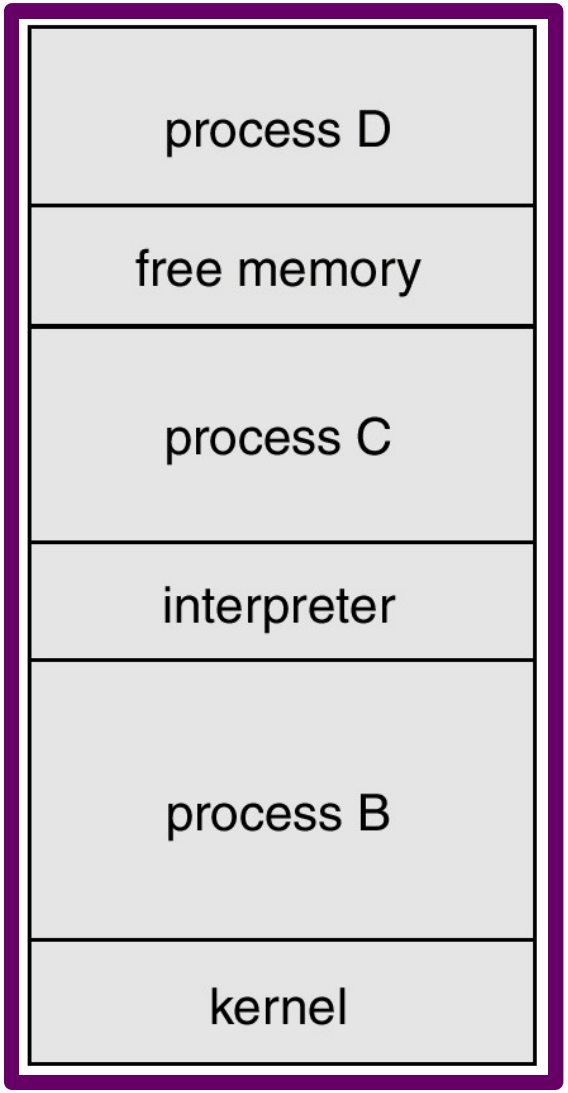
MS- DOS Execution



At System Start- up

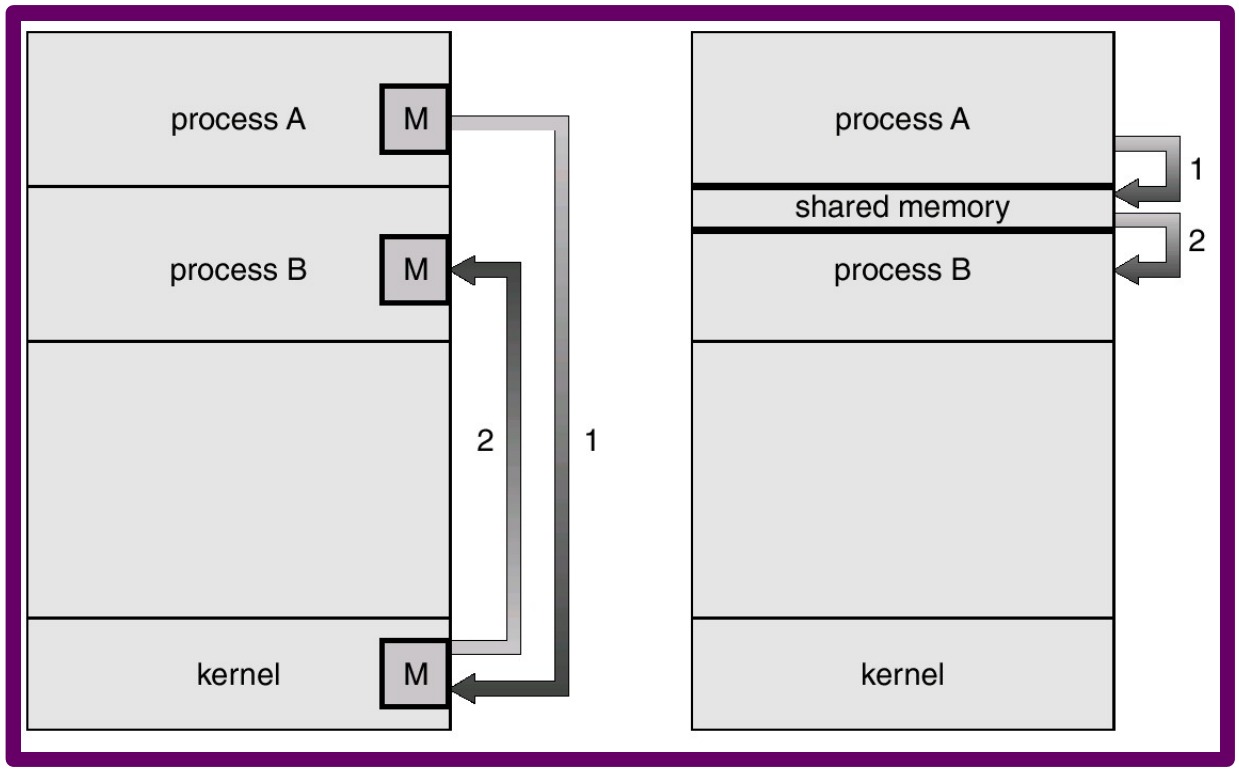
Running a Program

UNIX Running Multiple Programs



Communication Models

- Communication may take place using either message passing or shared memory.



Msg Passing

Shared Memory

Services view: System Programs

- Each system call is usually supported by a system program.
  - File/ directory manipulation: create, delete, copy name, print, dump, list
  - Status information: Programs that ask system information and display to users.
  - File modification: Text editor programs
  - Command interpreter
  - Programming language support
    - ✔ Compilers, assemblers, debuggers
  - Loaders and linkers
    - ✔ For enabling the execution of programs that start new processes and connect different ones.
  - Application programs
    - ✔ Text processing software, graphics support, spread sheets, games

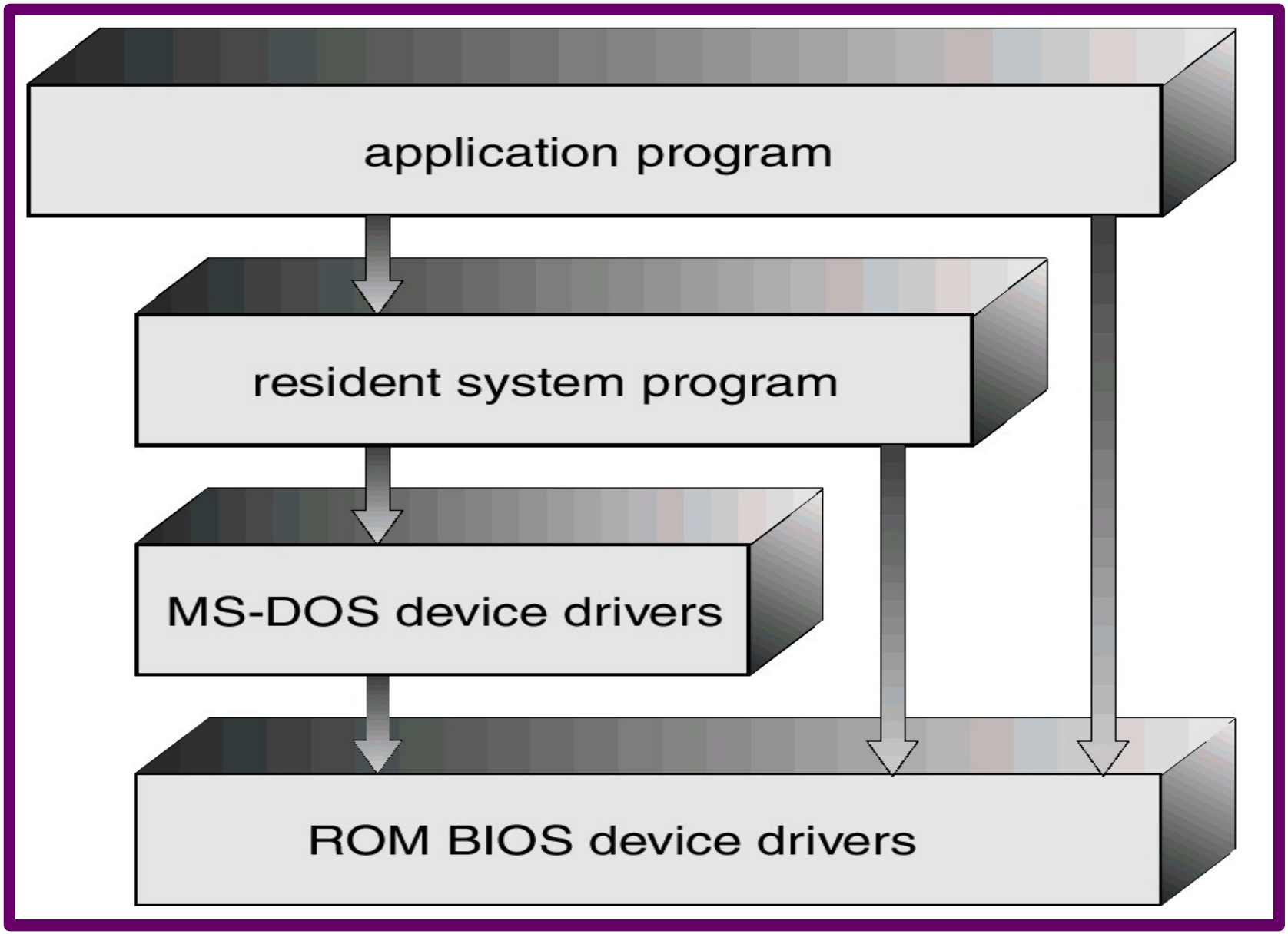
Structure view

- OS is a complex program and should be engineered carefully
- Divide into components with carefully defined inputs, outputs and functions.
- How to structure OS functionality ?
  - Layering
  - Virtual machines
- Designing and implementation of OS.

MS- DOS System Structure

- 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
  - The designers of MSDOS had no idea that it would become popular.
  - Limited hardware and space.
  - Application programs are able to access basic I/ O routines to write correctly to the display and disk writes.
  - MSDOS is vulnerable to errant (or malicious) programs

MS- DOS Layer Structure



OS design hierarchy: hypothetical model

Level	Name	Objects	Example operations
13	Shell	User programming environment	Statements in shell language
12	User programs	User processes	Quit, Kill, suspend
11	Directories (maintains association between external and internal identifiers of system resources and objects)	Directories	Create, destroy, search, list, access rights
10	Devices (access to external devices)	Printers, displays, and key boards	Create, destroy, open, close
9	File system (long storage of named files)	files	Create, destroy, open, close
8	Communications	pipes	Create, destroy, open, close, read, write
7	Virtual memory (creating logical address space for programs)	Segments, pages	Read, write, fetch
6	Local secondary storage (position of read/write heads)	Blocks of data, device channels	Read, write, allocate, free
5	Primitive processes	Primitive process, semaphores, synchronization premitives	Suspend, resume, wait, signal
4	Interrupts	Interrupts handling programs	Invoke, mask, unmask, retry
3	Procedures	Procedures, call stack	Mark stack, call, return
2	Instruction set	Evaluation, stack, micro-program, interpreter	Load, store, add, subtract, branch
1	Electronic circuits	Registers, gates, busses	Clear, transfer, activate, complement

Pipe and shell

- A pipe is defined with its output from one process and its input into another process.
- Shell provides simply as a collection of services
- The shell accepts user commands or job control statements, interprets these and creates and controls processes.

Structure view: Structure of OS designs

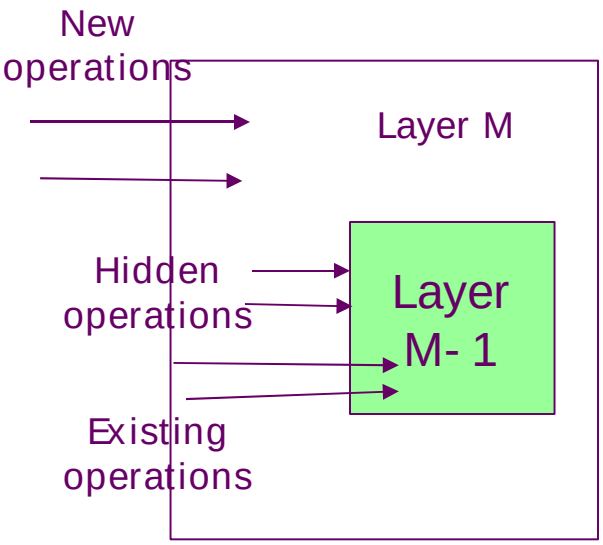
- Traditionally adhoc and unstructured approaches.
- Structured designs
  - Layered approach is followed to realize modularity
  - Encapsulate the modules separately
  - Have a hierarchy of layers where each layer calls procedures from the layers below.

- The THE OS (Dijkstra)
  - Layer 0 : Hardware
  - Layer 1: CPU scheduler
  - Layer 2: Memory manager
  - Layer 3: Operator- console device driver
  - Layer 4: I/O buffering
  - Layer 5: user program layer

- The VENUS OS (Liskov)
  - Layer 0 : Hardware
  - Layer 1: Instruction interpreter
  - Layer 2: CPU scheduler
  - Layer 3: I/O channel
  - Layer 4: Virtual memory
  - Layer 5: device drivers and schedulers
  - Layer 6: user program layer

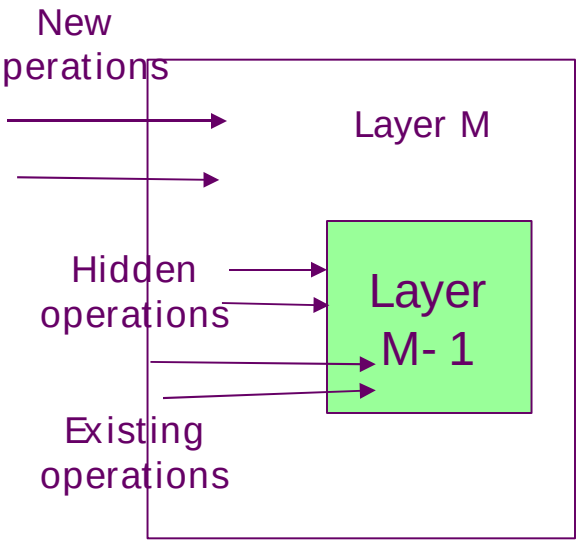
Structure view: layered system design

- A general philosophy that builds on the above approach
  - Decompose functionality into layers such that
    - ✔ Hardware is level 0, and layer t accesses functionality at layer (t- 1) or less
    - ✔ Access via appropriately defined system calls.
- Advantages
  - Modular design: well defined interfaces between layers
  - Prototyping/ development
    - ✔ Association between function and layer eases overall OS design.
    - ✔ OS development and debugging is layer by layer.
    - ✔ Simplifies debugging and system verification



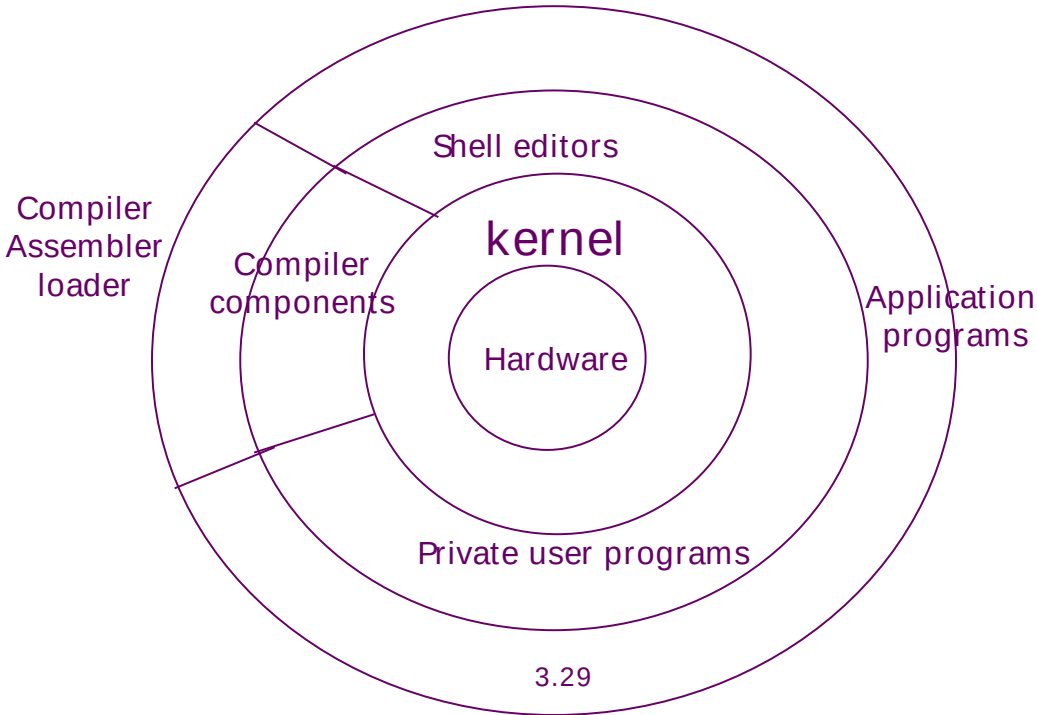
Structure view: layered system design...

- Disadvantages
  - Substantial design time since layers have to be designed carefully
  - Crossing multiple layers (of system calls) leads to inefficiency
  - An I/O request by the user has to pass each layer (I/O layer, CPU scheduling layer, hardware layer)
  - Each layer adds overhead to the system
  - OSs are not naturally hierarchical
  - Two layers (micro kernel and one more) approaches are successful.
  - Example: WINDOWS NT
    - ✔ First release had a layered architecture
    - ✔ But performance was low over WINDOWS98
    - ✔ Performance was improved by integrating some of the layers in WINDOWS4.0

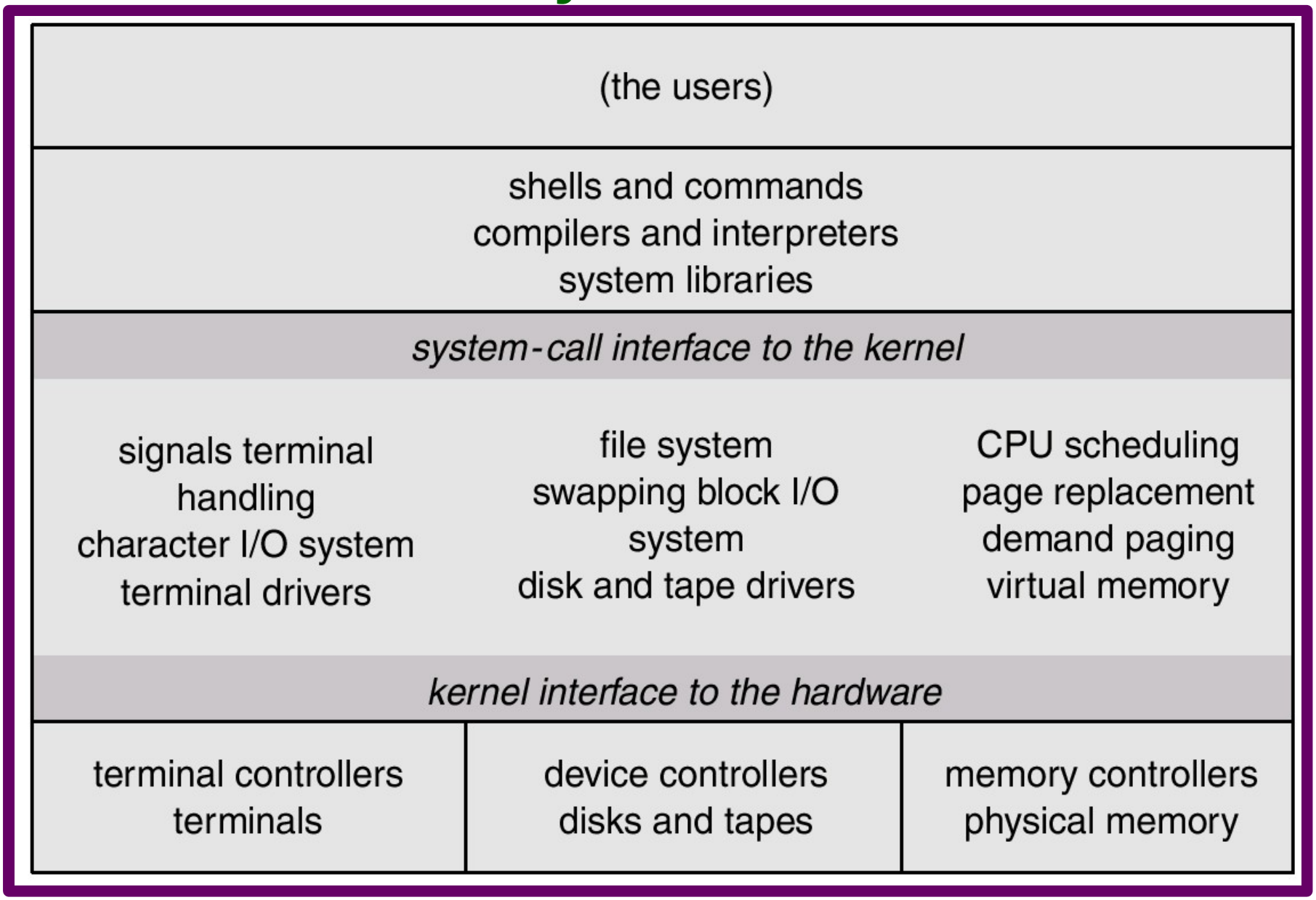


UNIX System Structure

- 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.



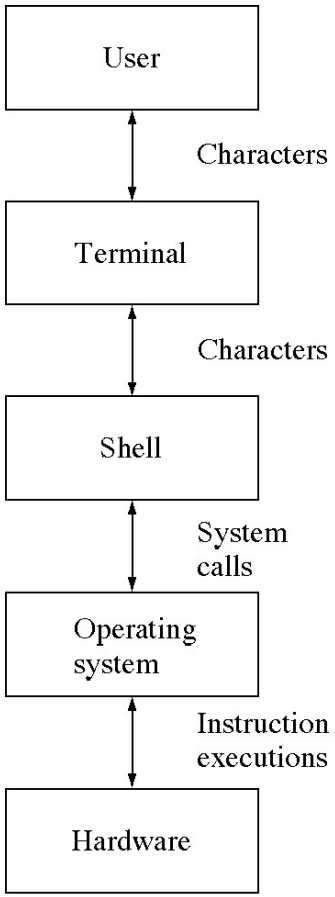
UNIX System Structure



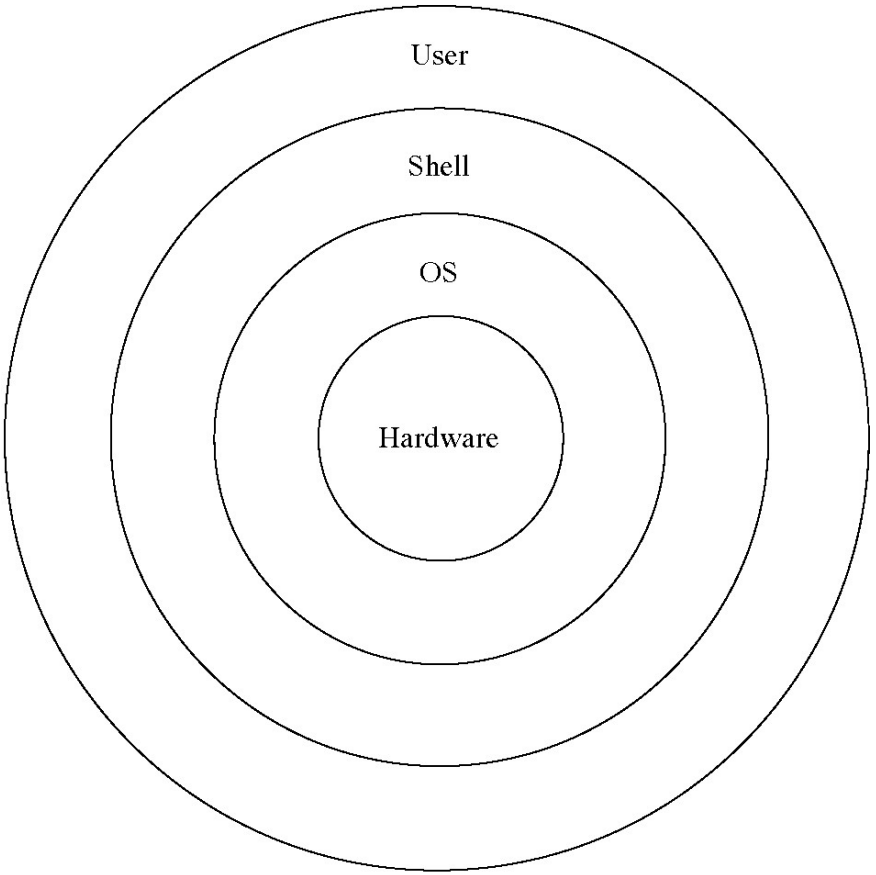
Shell: an OS interface

- Interactive access to the OS system calls
  - copy fromFile toFile
- Contains a simple programming language
- Popularized by UNIX
  - Before UNIX: JCL, OS CLs (command languages)
  - Bourne shell, C shell (csh), Korn shell (ksh), Bourne-again shell (bash), etc.

Two views of a shell



(a) Shell as a level



(b) Shell as a covering

Structure view: Virtual machines

- System programs above kernel can use either system calls or hardware instructions.
- System programs treat the hardware and the system calls as though they both are at the same level.
- In general application programs may view everything under them in the hierarchy as a part of machine itself, which leads to the concept of virtual machines..
- The virtual machine approach provides an interface that is identical to the underlying bare hardware.
- Each process is provided with a (virtual) copy of the underlying computer.
- The resources of the physical computer are shared to create the virtual machines.
  - CPU scheduling and spooling are used.

Structure view: Virtual machines

- Logical conclusion of layered approach.
  - The OS offers an abstract machine to execute on
    - ✔ Hides the specifics and details of hardware
  - Users are given their own VM, they can run any of the OS or software packages that are available on the underlying machine.
  - Provides a complete copy of underlying machine to the user
    - ✔ Pioneered under the name virtual machine (VM) by IBM
    - ✔ Became a household name with JAVA
- Mechanism of creating this illusion
  - CPU scheduling
    - ✔ Sharing the CPU in a transparent way
  - Memory management
    - ✔ Having large virtual memory space to address
  - Additional features such as file systems and so on are offered through file management subsystems
    - ✔ Problem is with partitioning the disk
      - – VM introduced minidisks such partitioning of tracks on the physical disk.

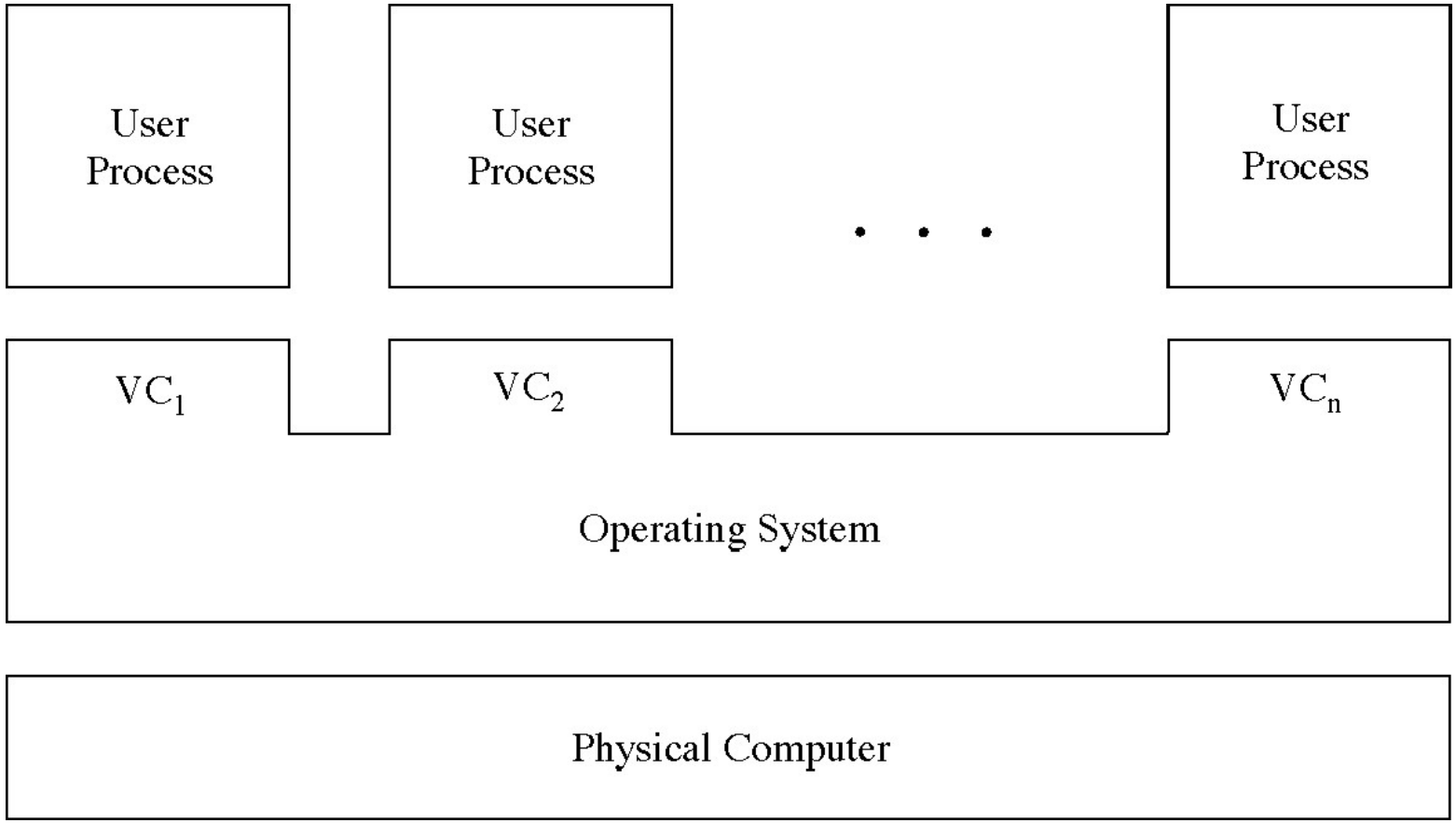
Types of multiplexing

- Time multiplexing
  - time-sharing
  - scheduling a serially-reusable resource among several users
- Space multiplexing
  - space-sharing
  - dividing a multiple-use resource up among several users

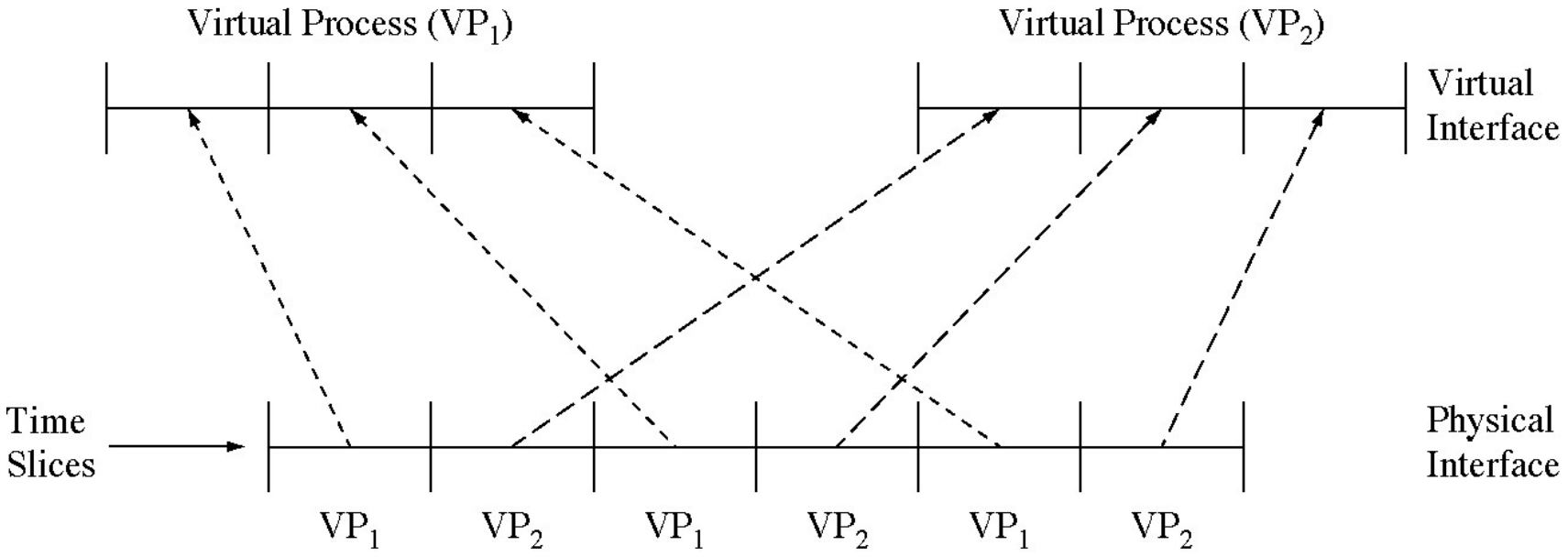
Virtual computers

- Processor virtualized to processes
  - mainly time- multiplexing
- Memory virtualized to address spaces
  - space and time multiplexing
- Disks virtualized to files
  - space- multiplexing
  - transforming

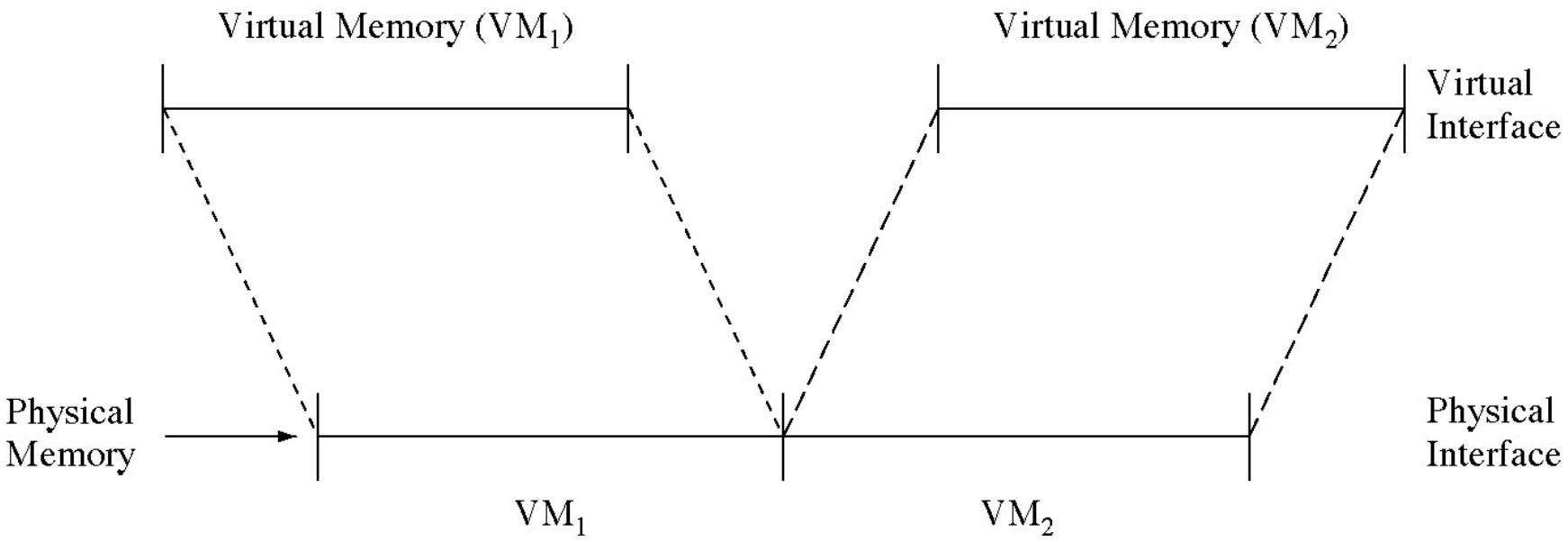
Multiple virtual computers



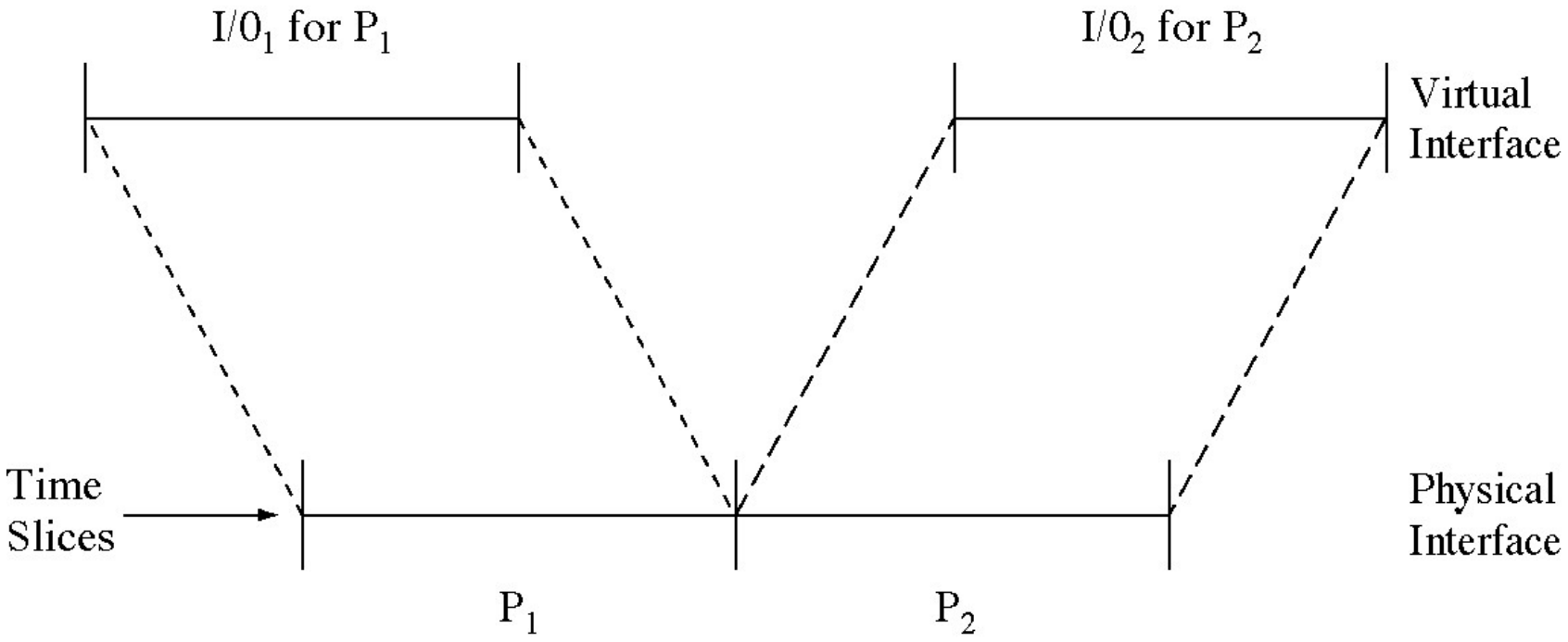
Time- multiplexing the processor



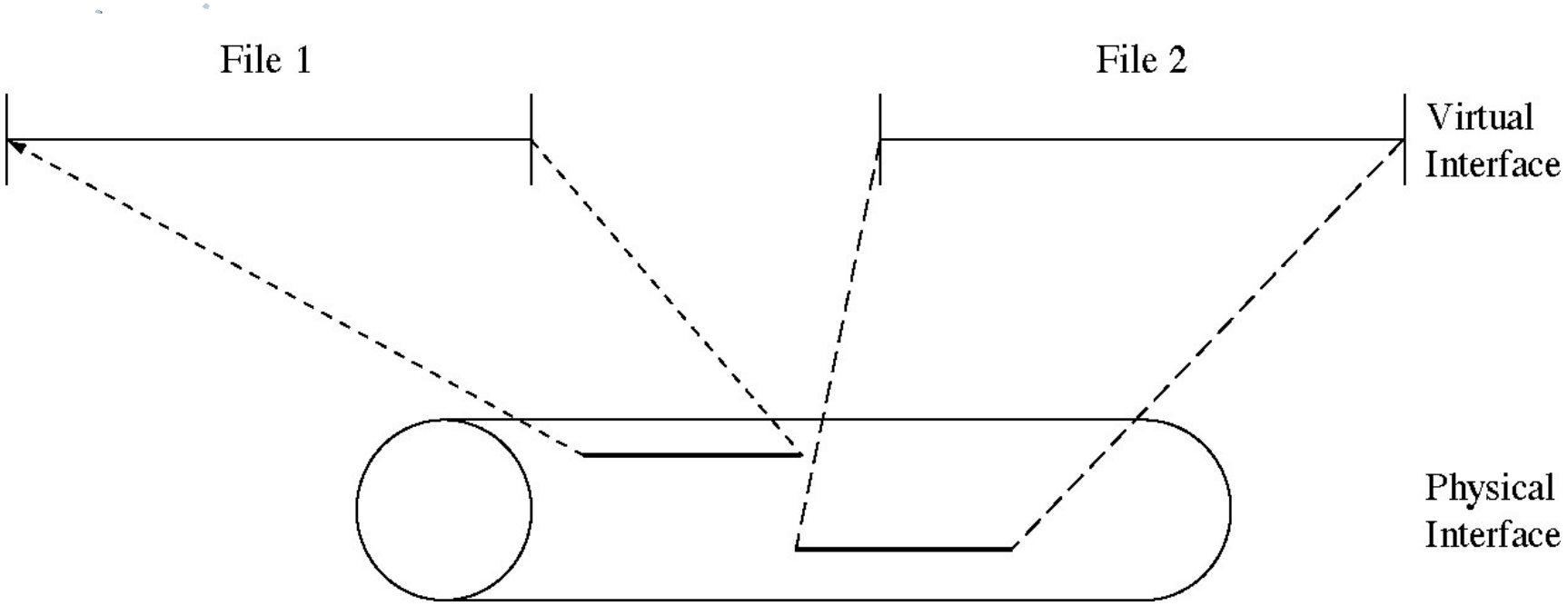
Space- multiplexing memory



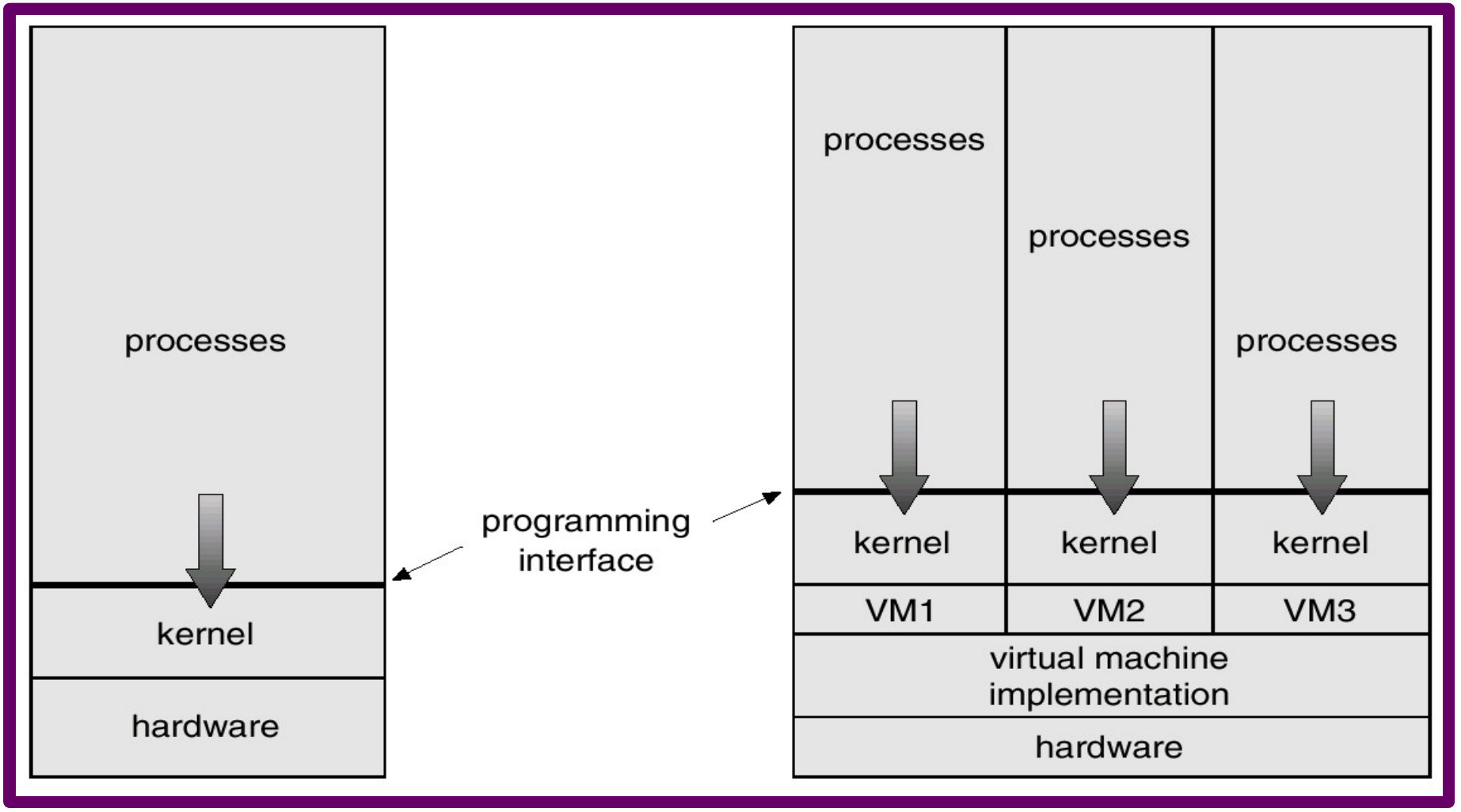
Time- multiplexing I/ O devices



Space- multiplexing the disk



System Models



Non- virtual Machine

Virtual Machine

Structure view: Virtual machines (cont..)

- Operation
  - User-level code executes as is
  - Supervisor code executes at user level
    - ✔ Privileged instructions are simulated.
      - – Generate trap to VM emulator
    - ✔ Hidden registers and I/O instructions are simulated.
  - Virtual machine software has two modes
    - ✔ User mode and monitor mode
  - The user program will execute in two modes
    - ✔ Virtual user mode
    - ✔ Virtual monitor mode

Structure view: Virtual machines (cont..)

- Advantages
  - A software created abstraction that is a replica of the underlying machine
  - Additional functionality can be provided
    - ✔ Can run different OSs on different VMs.
  - Each VM is completely isolated from others, so secure.
  - Protection of various system resources.
  - It can be used for OS research and development.
  - System programs are given their own virtual machine.
  - Thousands of programs are available for MSDOS on Intel CPU- based systems.
  - SUN micro- systems and DEC use faster machines
    - ✔ To test MSDOS programs on the faster machines, the best way is to provide a virtual Intel machine.
  - System compatibility problems can be solved
- Disadvantages
  - Performance degradation is inevitable

JAVA

- JAVA is designed by SUN Micro systems.
- JAVA compiler generates byte code output.
- These instructions run on Java virtual machine.
- JVM runs on many types of computer.
- JVM is implemented in web browsers.
- JVM implements a stack based instruction set.

System Design and Implementation :design goals

- First problem is defining the goals
  - Batch, time-shared, single-user, multi-user, distributed, real-time, or general purpose.
- The goals can be divided into two parts: user goals and system goals
  - Users: Convenient to use, easy to learn, easy to use, reliable, safe, and fast.
  - System goals: easy to design, create, maintain, and operate. It should be flexible, reliable, and error-free.
- Software engineering principles can be applied for deciding goals.

Mechanisms and policies

- The specification and design of OS is highly creative task.
- One important principle is separation of policy and mechanism.
- Mechanisms describe how things are done
  - The function of CPU scheduling is an example
  - For CPU protection, timer will be used.
  - Does not include decisions as to which process gets priority
- Policies describe what will be done
  - Deciding whether processes with more I/O get priority over CPU bound processes is an example
  - This policy can be a input parameter and can change from scheduler to scheduler
  - For CPU protection, how long the timer is set is a policy decision.
- Micro-kernel OSs only implement mechanisms. Policies are provided at user level

Implementation

- Assembly language can be used.
  - High speed
- High level language
  - Easy to develop, easy to understand and debug, easy to port.
  - UNIX was written in C.
  - Dis adv: reduced speed and increased storage requirements.
  - We have sophisticated compilers.

System generation (SYSGEN)

- The OS program is normally distributed on disk or tape.
- A special program SYSGEN reads from a given file and asks the operator regarding specific configuration of the system.
  - Probes the hardware directly to determine what components are there.
- Regenerate/ Configure the OS for new machine parameters
  - CPU type
  - Machine parameters include
    - ✔ Memory size, I/ O device parameters, address maps etc.
    - ✔ Mechanisms describe how things are done
  - Devices
  - Options of OS
    - ✔ CPU scheduling algorithm, buffer size.
- Approaches
  - Recompile
  - Have precompiled parameterized libraries: link rather than compiling
  - Use table driven run- time selection