# Operating System

## Lecture 4: System Calls

Manoj Kumar Jain

M.L. Sukhadia University Udaipur

# Outline

- System Calls
- System Programs
- System Structure
- Virtual Machines
- System Design and Implementation
- System Generation

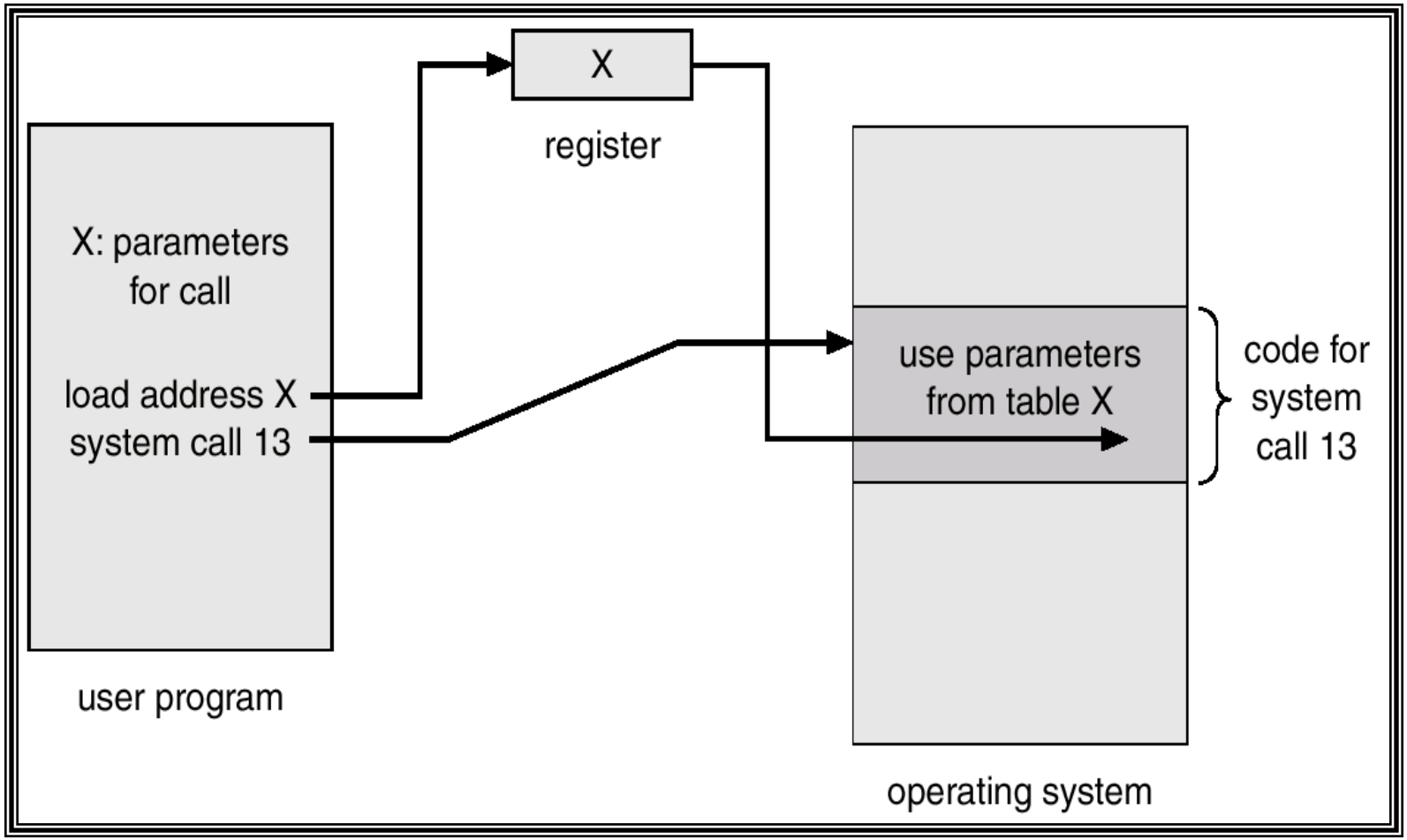
# System Calls

- System calls provide the interface between a running program and the operating system.
  - Generally available as assembly-language instructions.
  - Languages defined to replace assembly language for systems programming allow system calls to be made directly (e.g., C, C++)

# System Calls (Cont.)

- Three general methods are used to pass parameters between a running program and the operating system.
  - Pass parameters in *registers*.
  - Store the parameters in a table in memory, and the table address is passed as a parameter in a register.
  - *Push* (store) the parameters onto the *stack* by the program, and *pop* off the stack by operating system.

# Passing of Parameters As A Table



# Types of System Calls

- Process control
- File management
- Device management
- Information maintenance
- Communications

# Process Control

- end, abort
- Load, execute
- create process, terminate process
- get process attributes, set process attributes
- wait for time
- wait event, signal event
- allocate and free memory

# File Management

- create file, delete file
- open, close
- read, write, reposition
- get file attributes, set file attributes

# Device Management

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

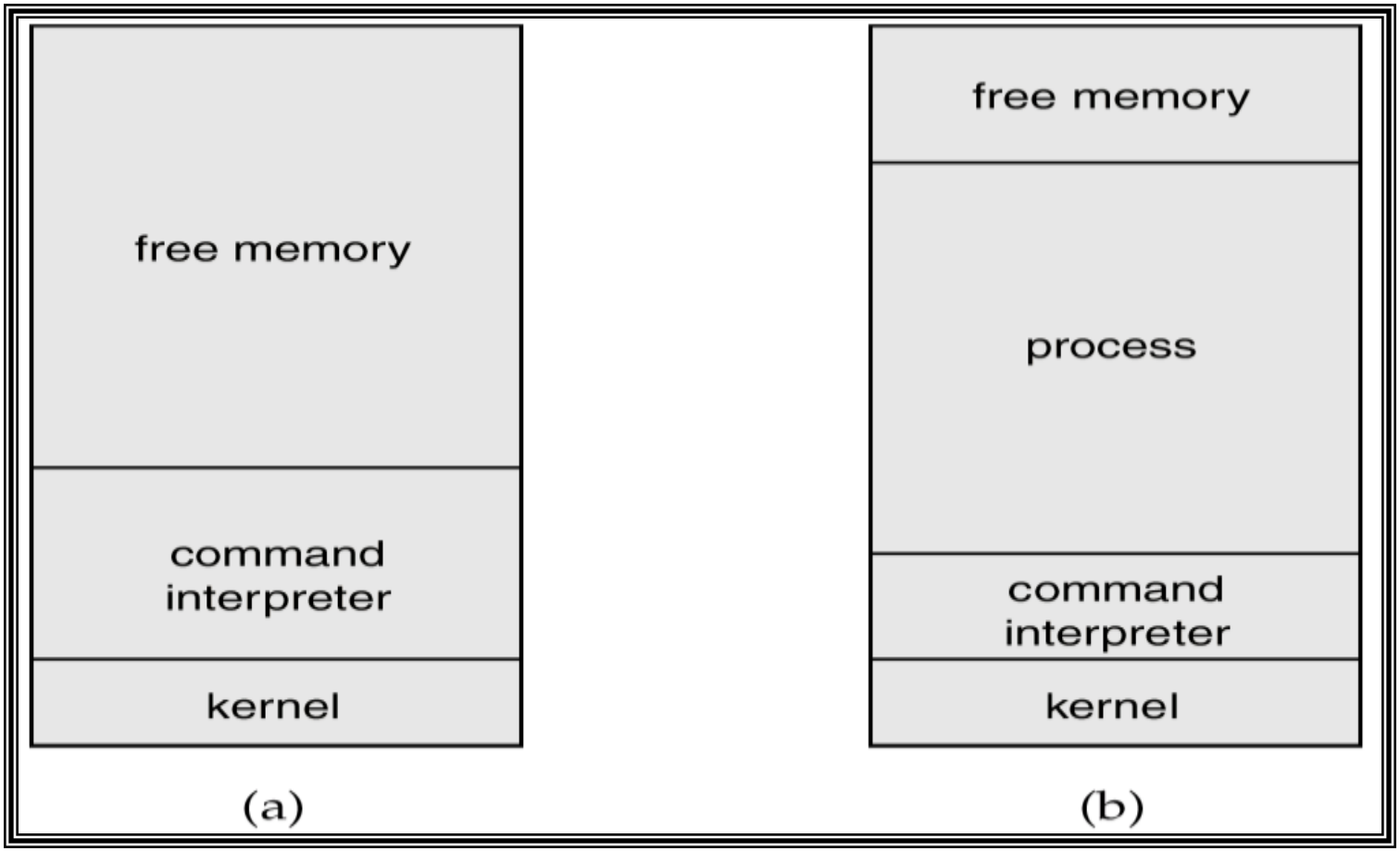
# Information Maintenance

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

# Communications

- create, delete communication connection
- send, receive messages
- transfer status information
- attach or detach remote devices

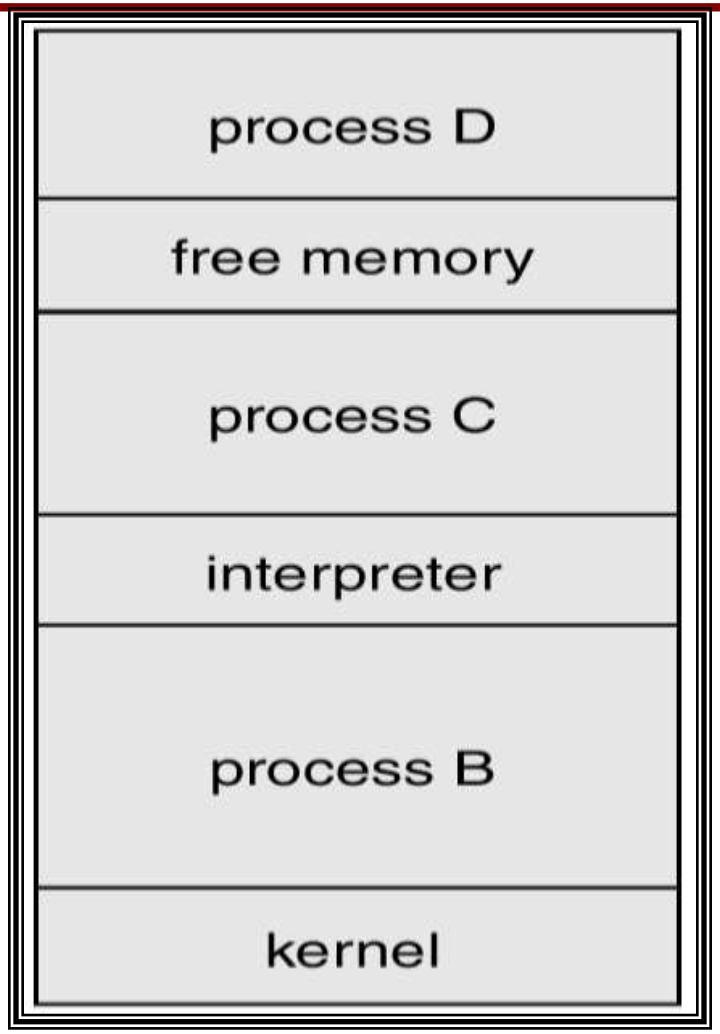
# MS-DOS Execution




At System Start-up

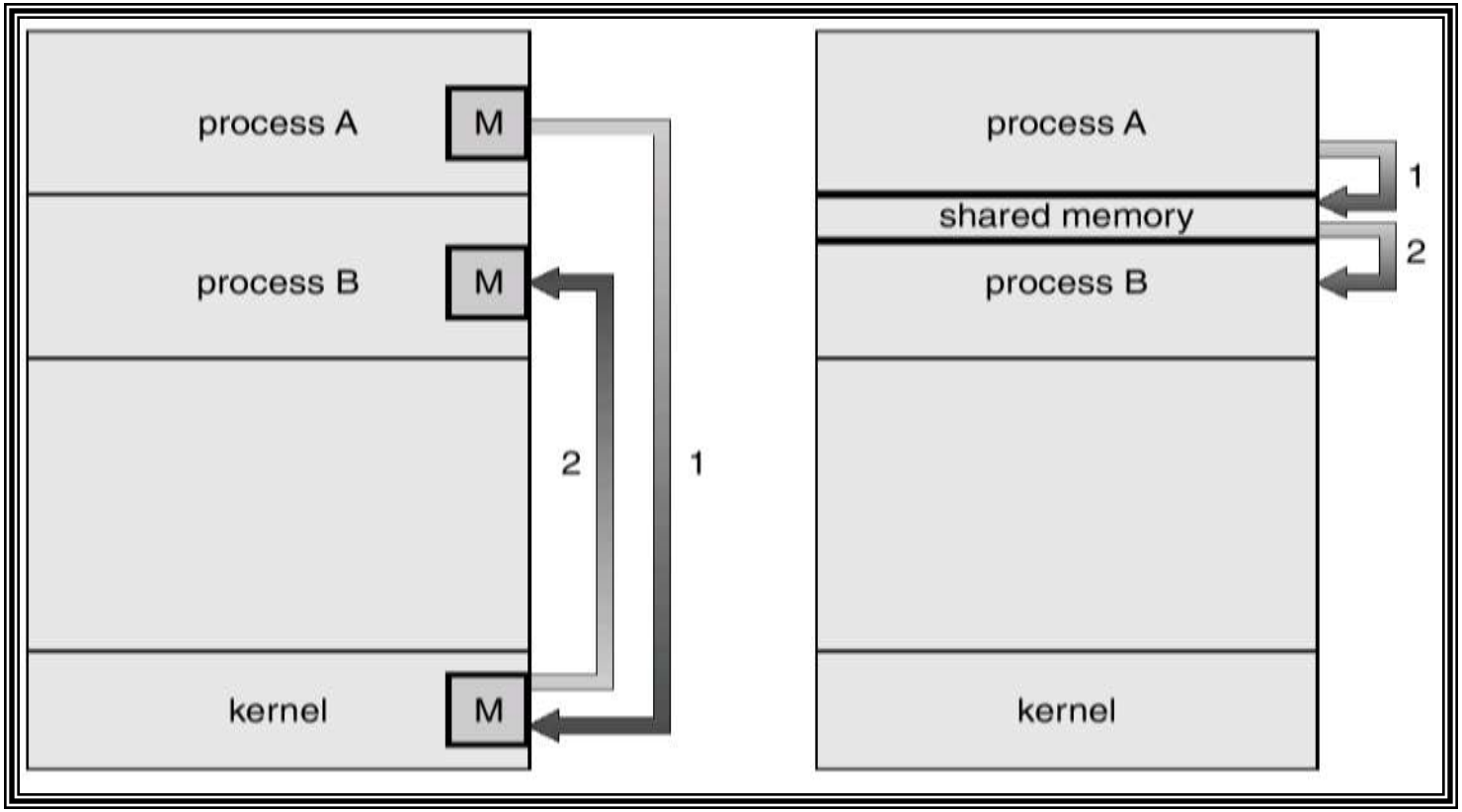
Running a Program

# UNIX Running Multiple Programs



# Communication Models

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



# System Programs

- System programs provide a convenient environment for program development and execution. The can be divided into:
  - File manipulation
  - Status information
  - File modification
  - Programming language support
  - Program loading and execution
  - Communications
  - Application programs

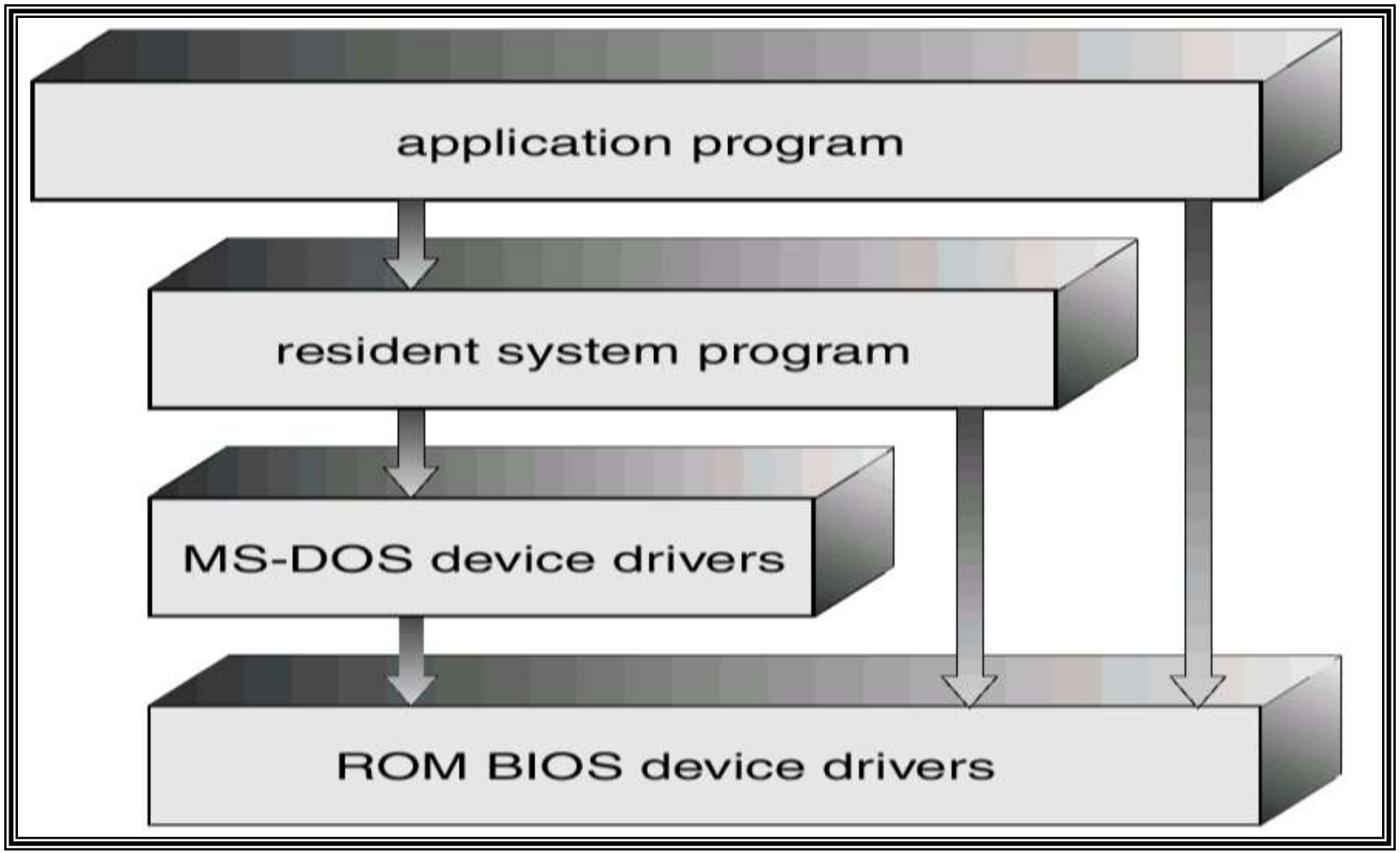
# System Programs (Cont.)

Most users' view of the operation system is defined by system programs, not the actual system calls.

# 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

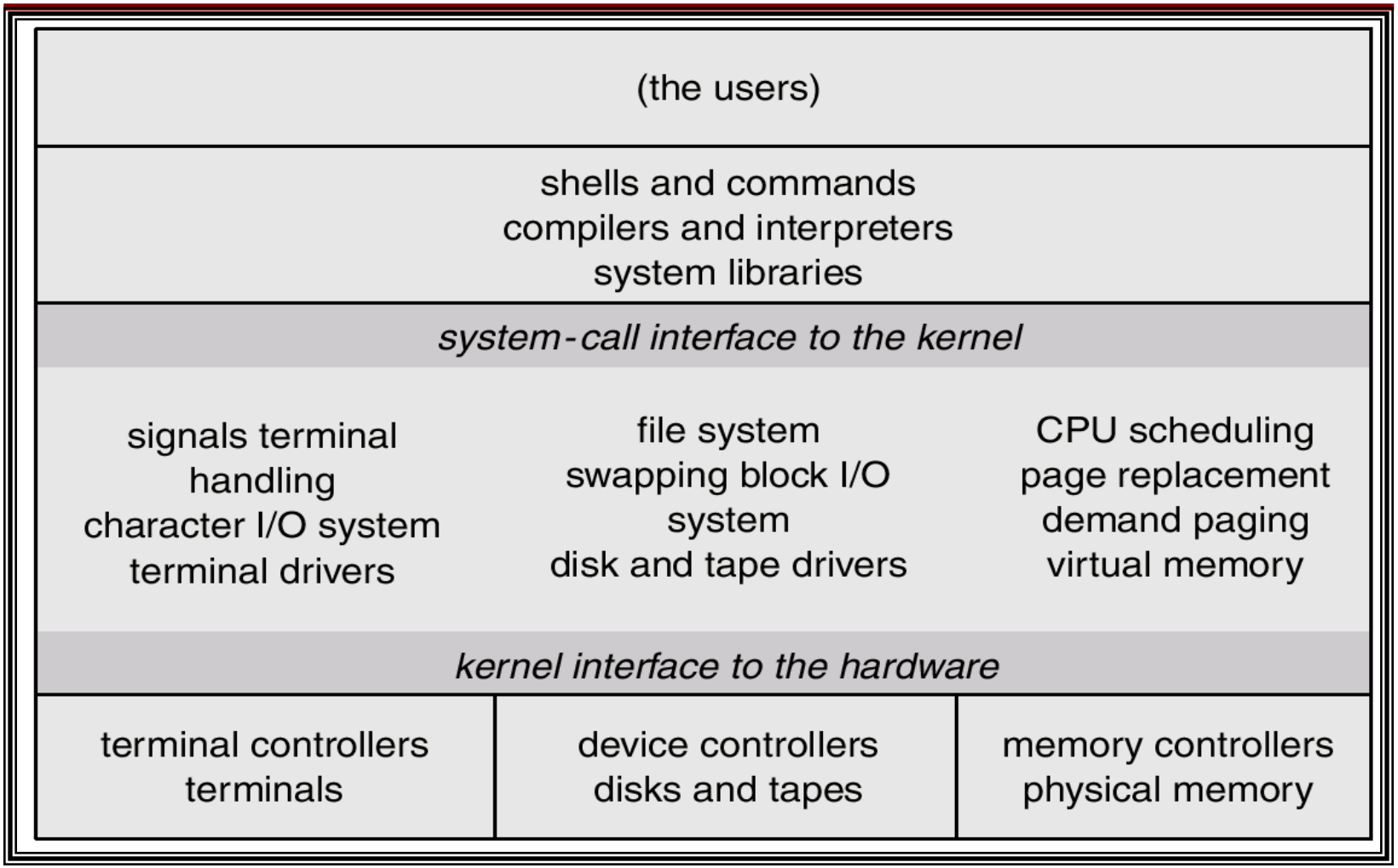
# MS-DOS Layer Structure



# 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.

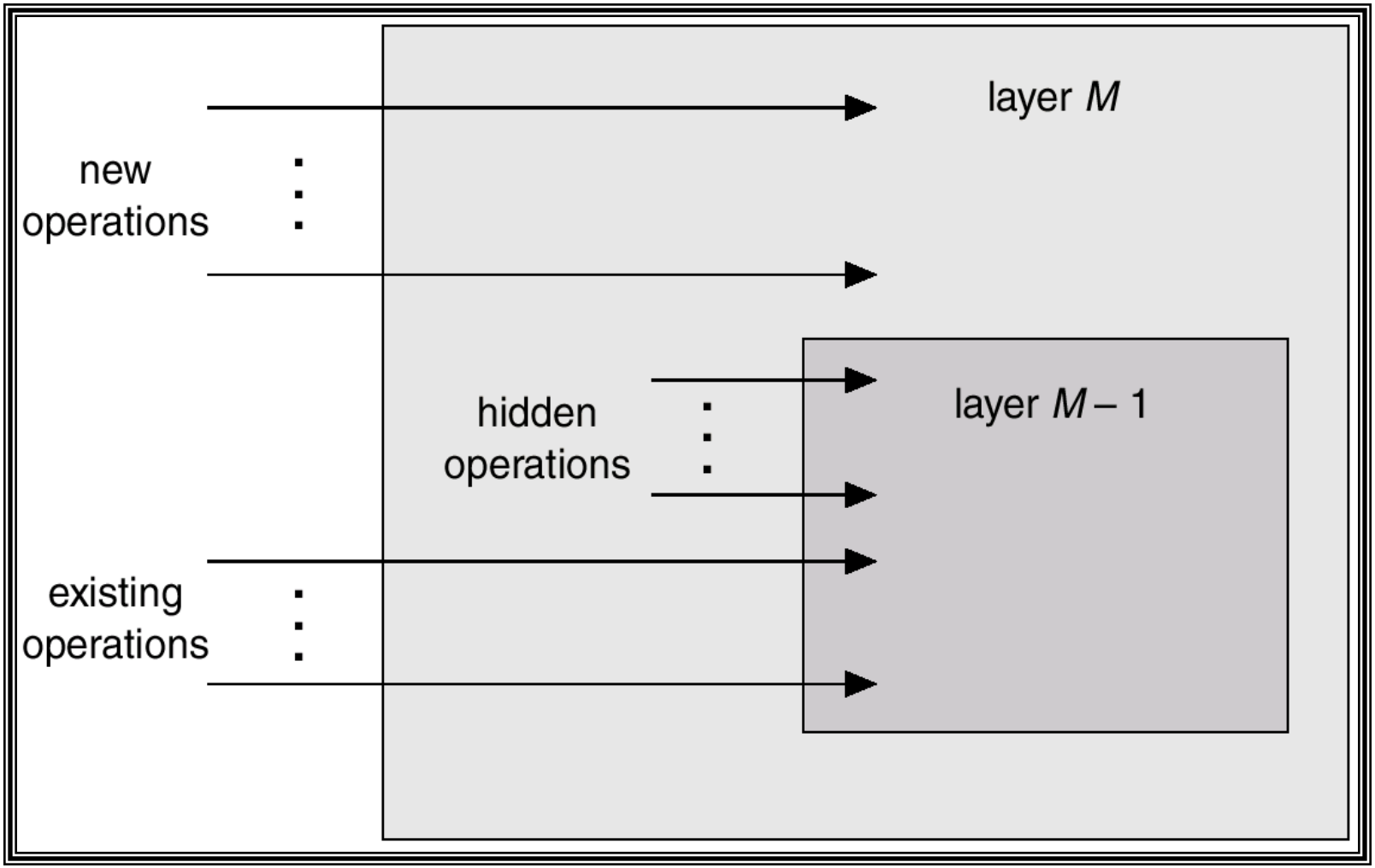
# UNIX System Structure



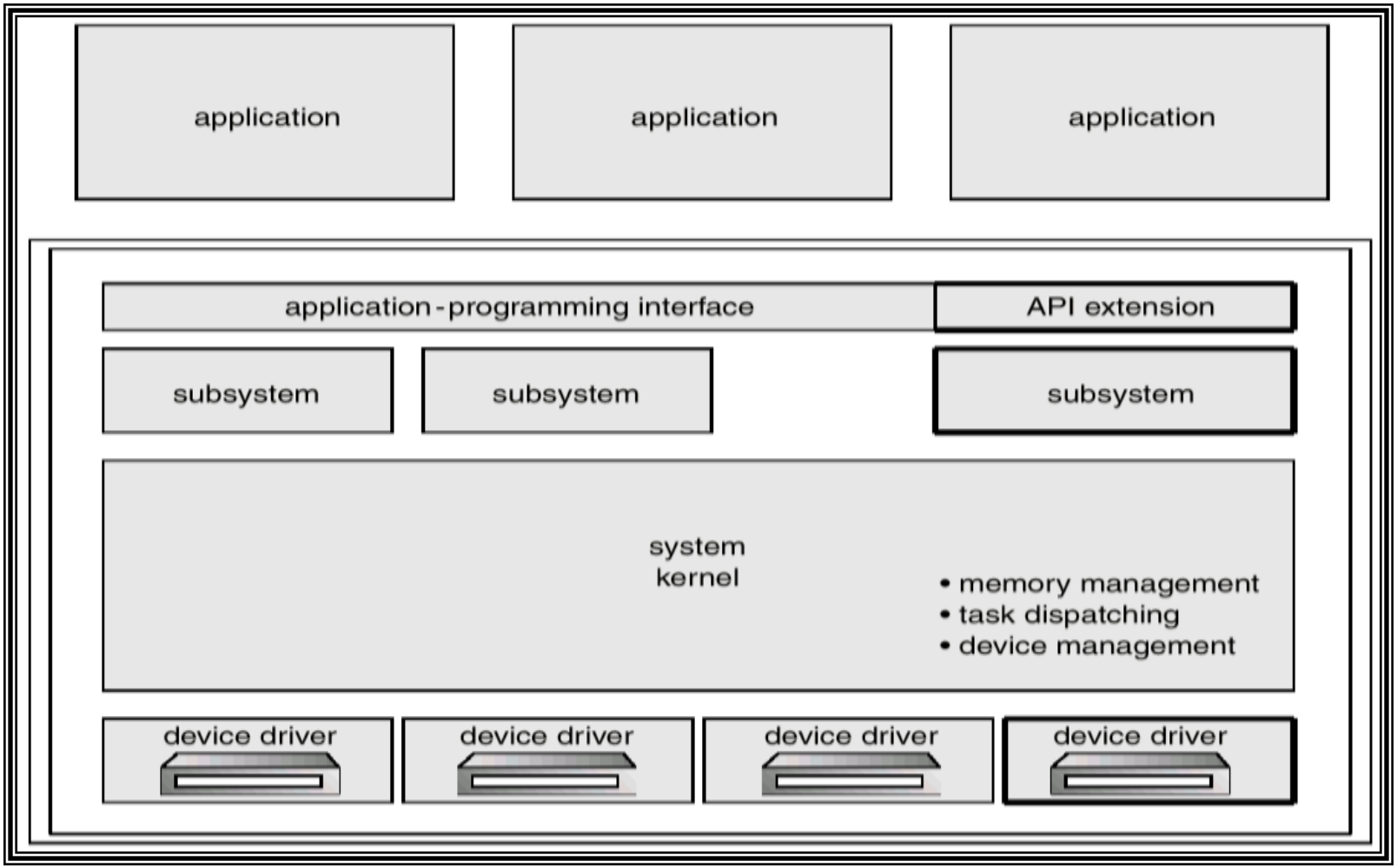
# Layered Approach

- 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.

# An Operating System Layer



# OS/2 Layer Structure



# Microkernel System Structure

- Moves as much from the kernel into "*user*" space.
- Communication takes place between user modules using message passing.
- Benefits:

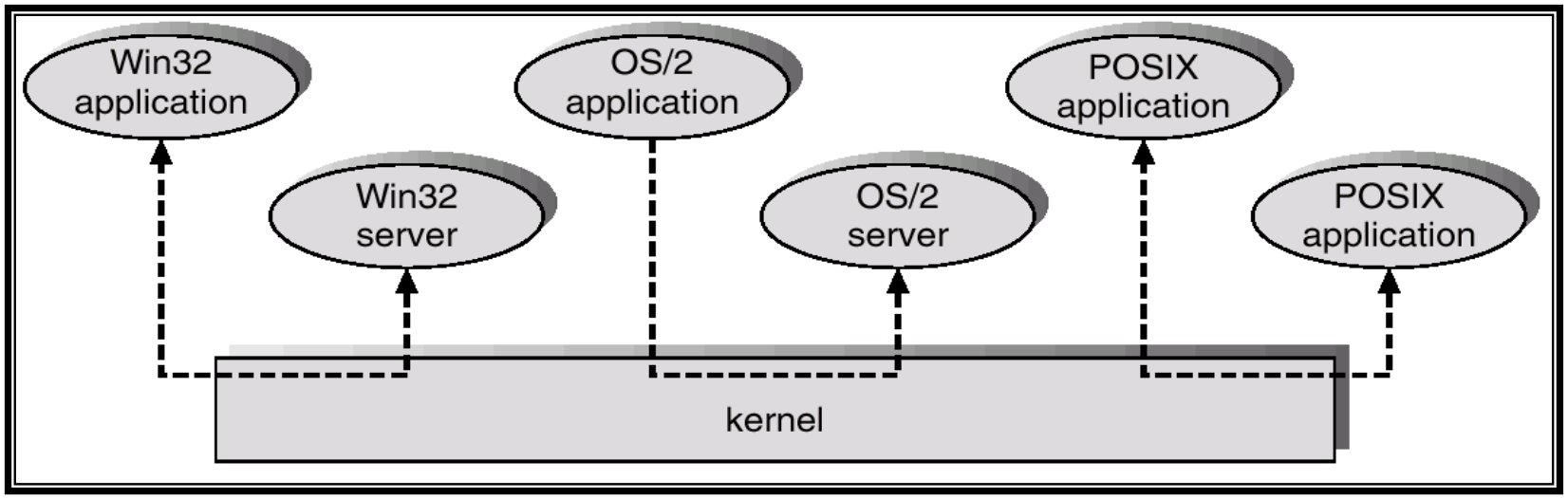
  - easier to extend a microkernel

  - easier to port the operating system to new architectures

  - more reliable (less code is running in kernel mode)

  - more secure

# Windows NT Client-Server Structure




It uses a hybrid structure.

Part of windows NT architecture uses layering.

Designed to run various applications, including Win32, OS/2 and POSIX.

It provides a server that runs in user space for each application type.

The kernel coordinates the message passing between client applications and application servers.

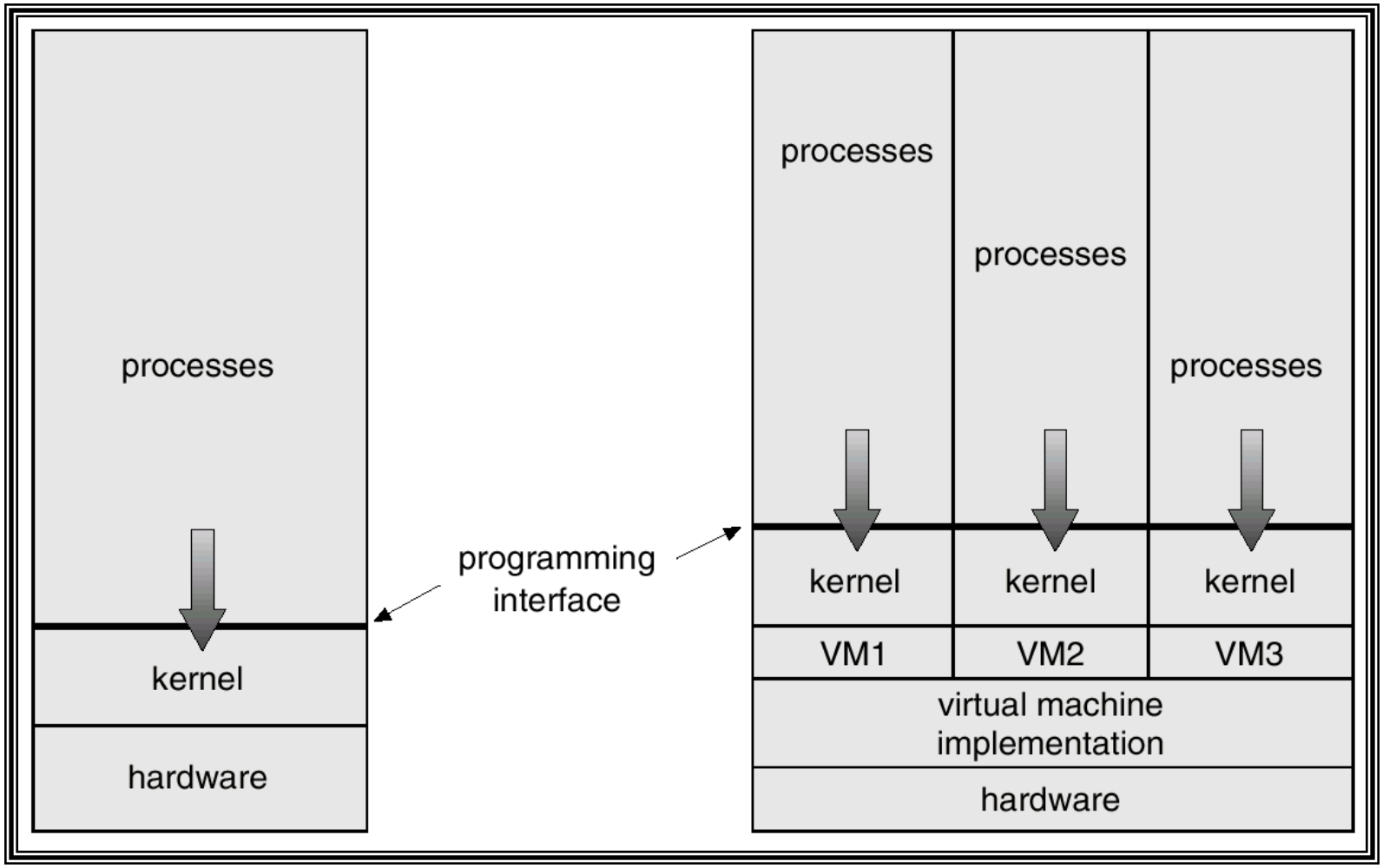
# Virtual Machines

- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- The operating system creates the illusion of multiple processes, each executing on its own processor with its own (virtual) memory.

# Virtual Machines (Cont.)

- The resources of the physical computer are shared to create the virtual machines.
  - CPU scheduling can create the appearance that users have their own processor.
  - Spooling and a file system can provide virtual card readers and virtual line printers.
  - A normal user time-sharing terminal serves as the virtual machine operator's console.

# System Models



# Adv./Disadv. of Virtual Machines

- The virtual-machine concept provides complete protection of system resources since each virtual machine is isolated from all other virtual machines. This isolation, however, permits no direct sharing of resources.
- A virtual-machine system is a perfect vehicle for operating-systems research and development. System development is done on the virtual machine, instead of on a physical machine and so does not disrupt normal system operation.

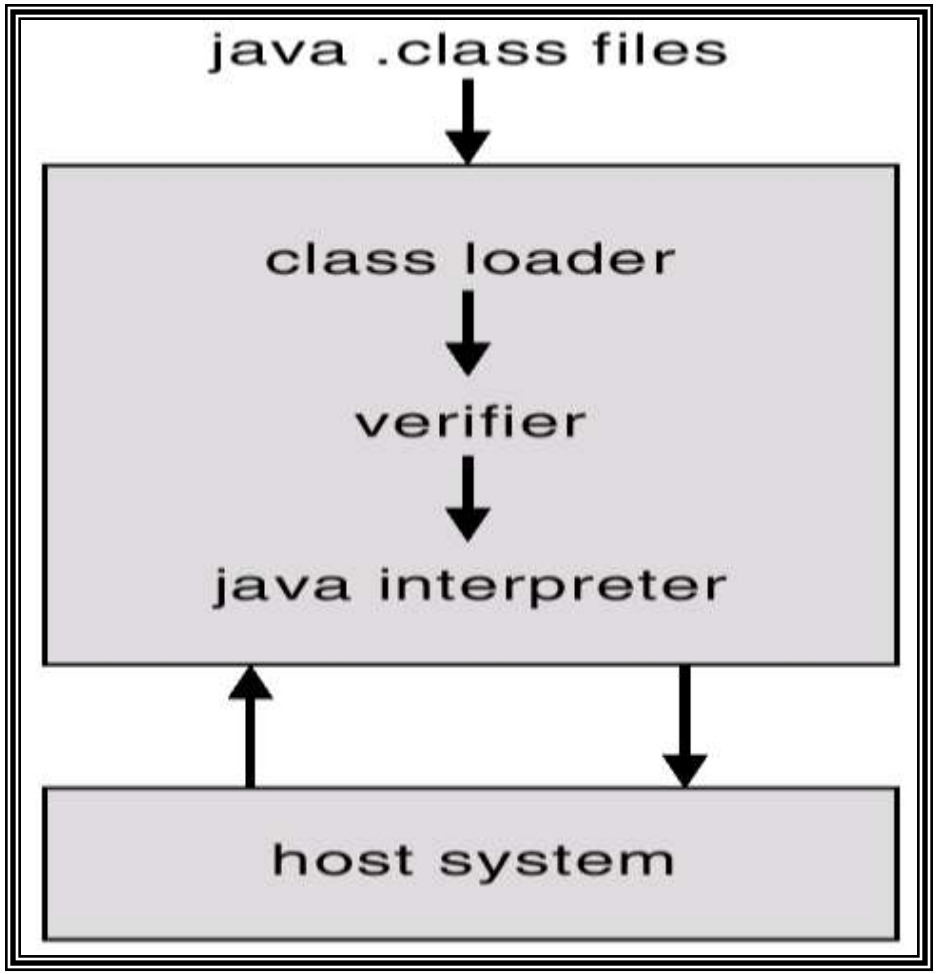
# Adv./Disadv. of Virtual Machines (Cont.)

- The virtual machine concept is difficult to implement due to the effort required to provide an *exact* duplicate to the underlying machine.

# Java Virtual Machine

- Compiled Java programs are platform-neutral bytecodes executed by a Java Virtual Machine (JVM).
- JVM consists of
  - class loader
  - class verifier
  - runtime interpreter
- Just-In-Time (JIT) compilers increase performance

# Java Virtual Machine



# System Design Goals

- User goals – operating system should be convenient to use, easy to learn, reliable, safe, and fast.
- System goals – operating system should be easy to design, implement, and maintain, as well as flexible, reliable, error-free, and efficient.

# Mechanisms and Policies

- Mechanisms determine how to do something, policies decide what will be done.
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later.

# System Implementation

- Traditionally written in assembly language, operating systems can now be written in higher-level languages.
- Code written in a high-level language:
  - can be written faster.
  - is more compact.
  - is easier to understand and debug.
- An operating system is far easier to *port* (move to some other hardware) if it is written in a high-level language.

# System Generation (SYSGEN)

- Operating systems are designed to run on any of a class of machines; the system must be configured for each specific computer site.
- SYSGEN program obtains information concerning the specific configuration of the hardware system.
- *Booting* – starting a computer by loading the kernel.
- *Bootstrap program* – code stored in ROM that is able to locate the kernel, load it into memory, and start its execution.

# *Thanks*