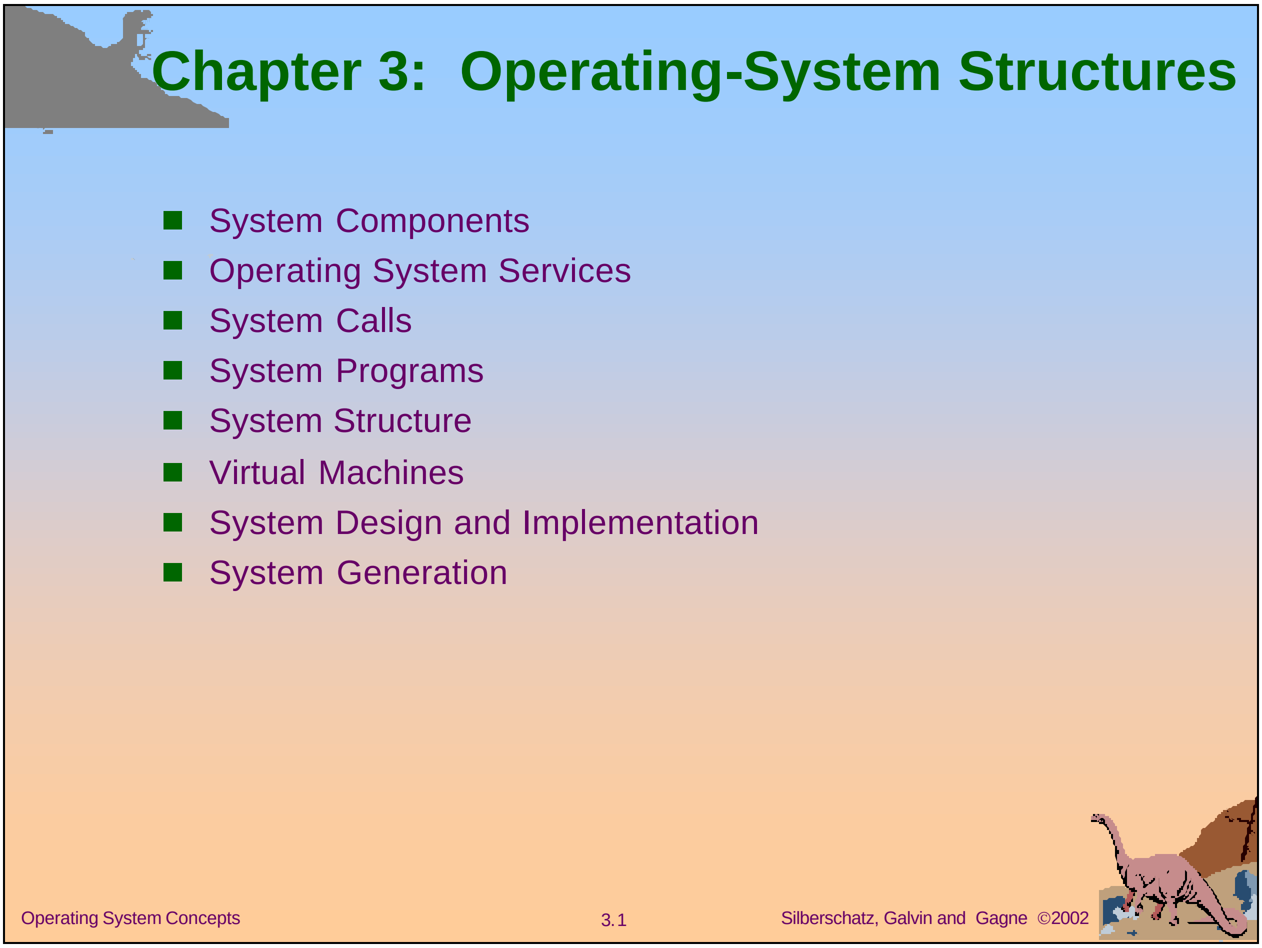
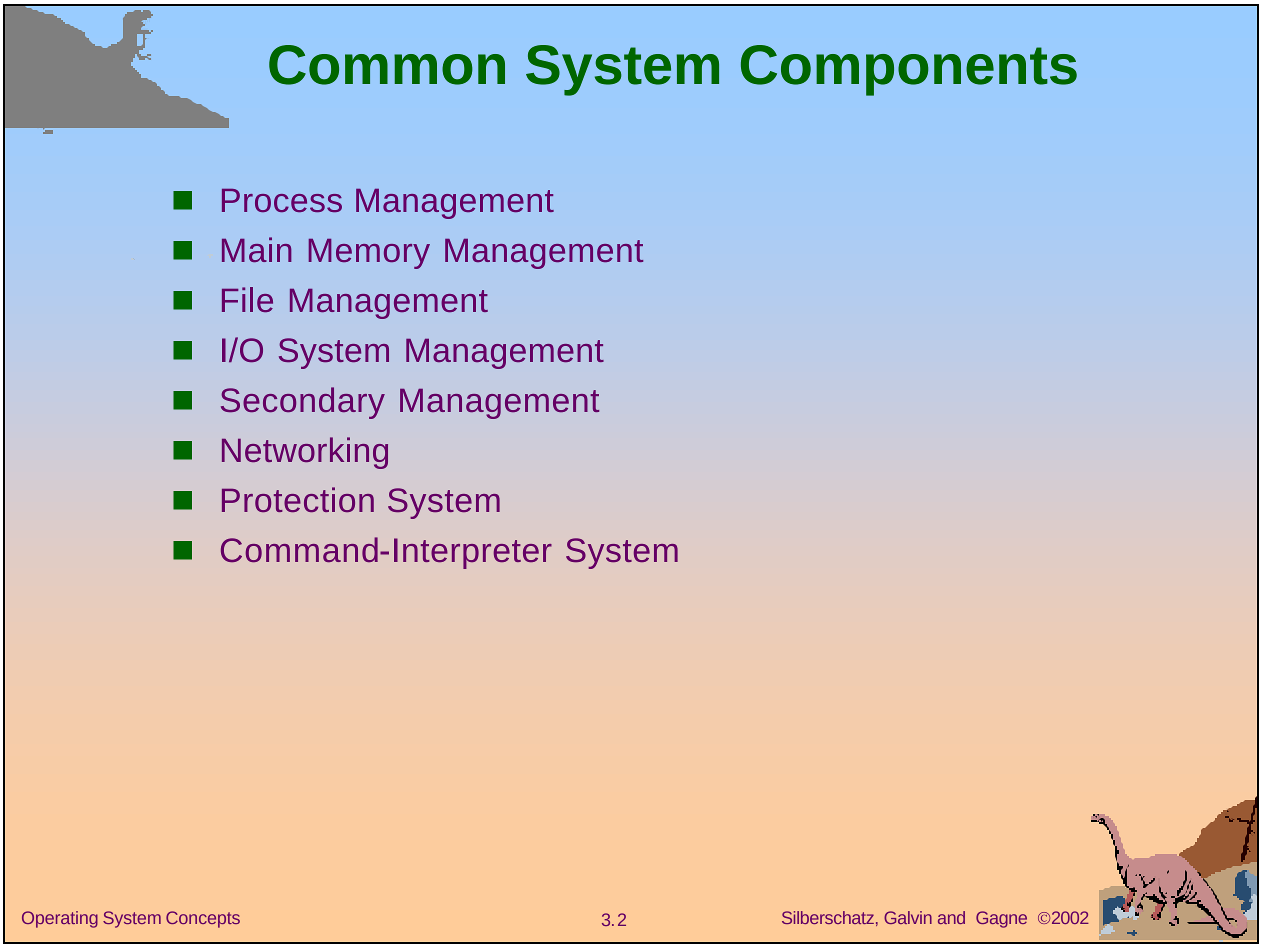


Chapter 3: Operating-System Structures

- System Components
- Operating System Services
- System Calls
- System Programs
- System Structure
- Virtual Machines
- System Design and Implementation
- System Generation

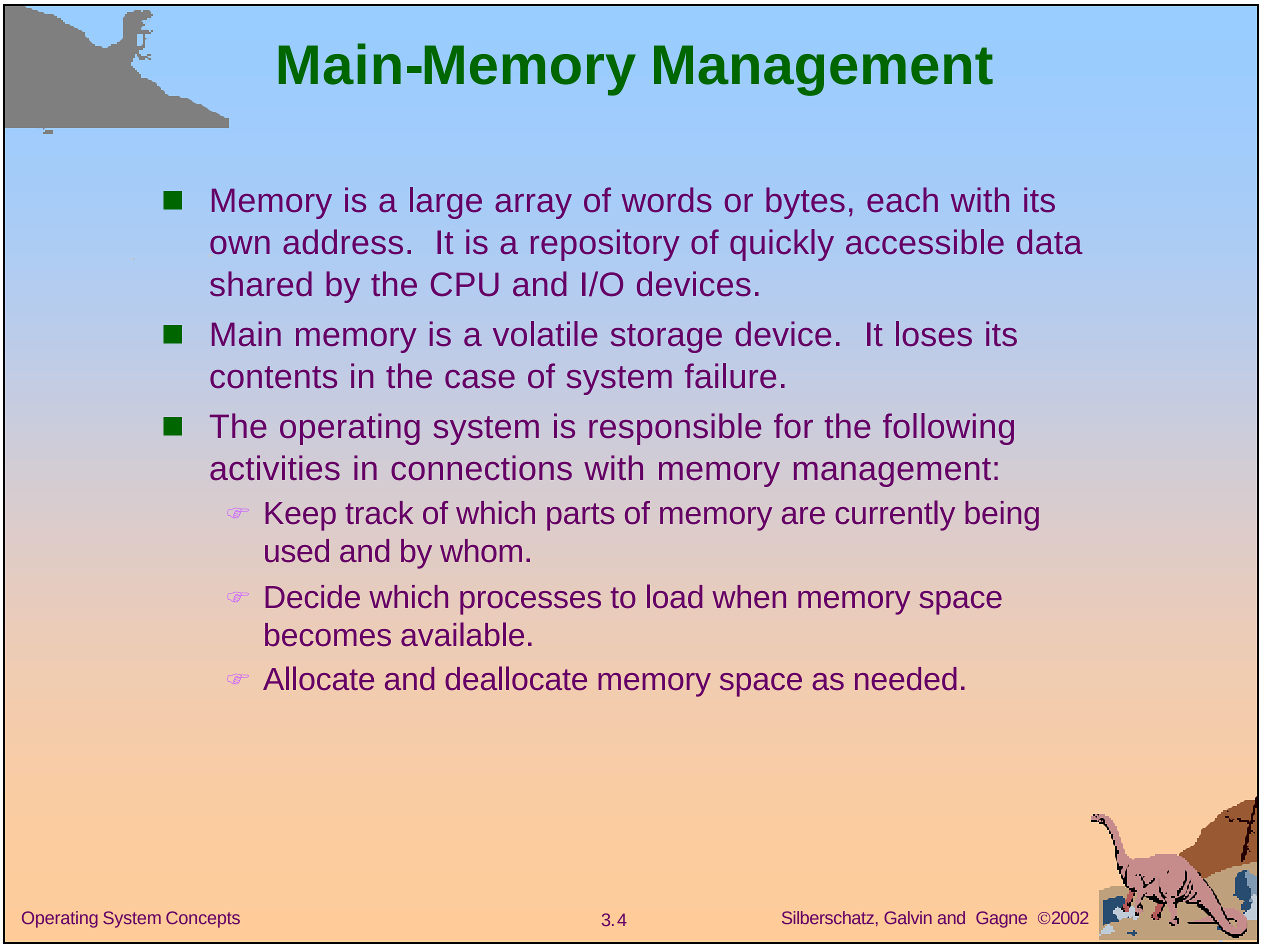


Common System Components

- Process Management
- Main Memory Management
- File Management
- I/O System Management
- Secondary Management
- Networking
- Protection System
- Command-Interpreter System

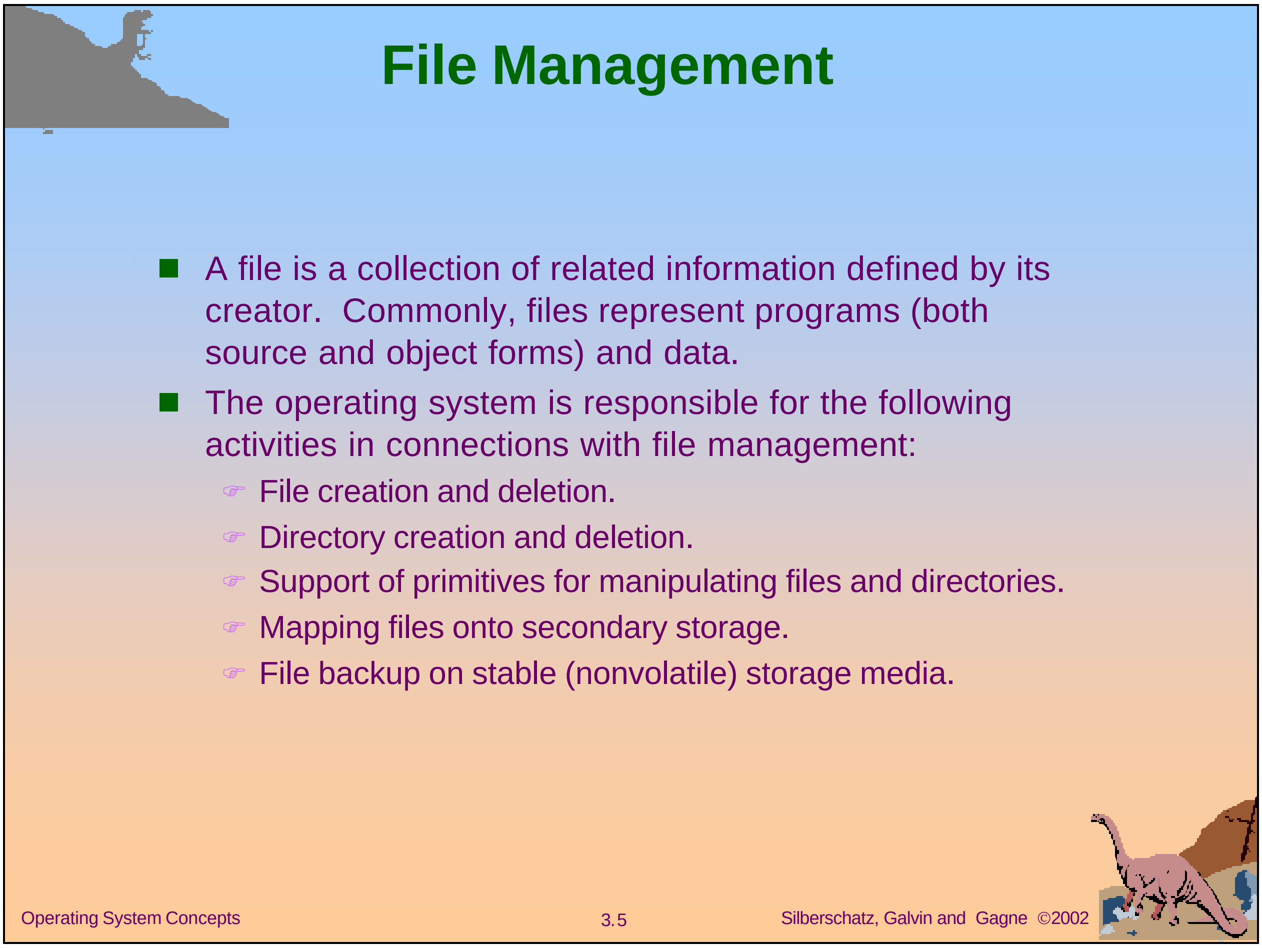
Process Management

- A *process* is a program in execution. A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- The operating system is responsible for the following activities in connection with process management.
  - Process creation and deletion.
  - process suspension and resumption.
  - Provision of mechanisms for:
    - process synchronization
    - process communication
    - Deadlock handling



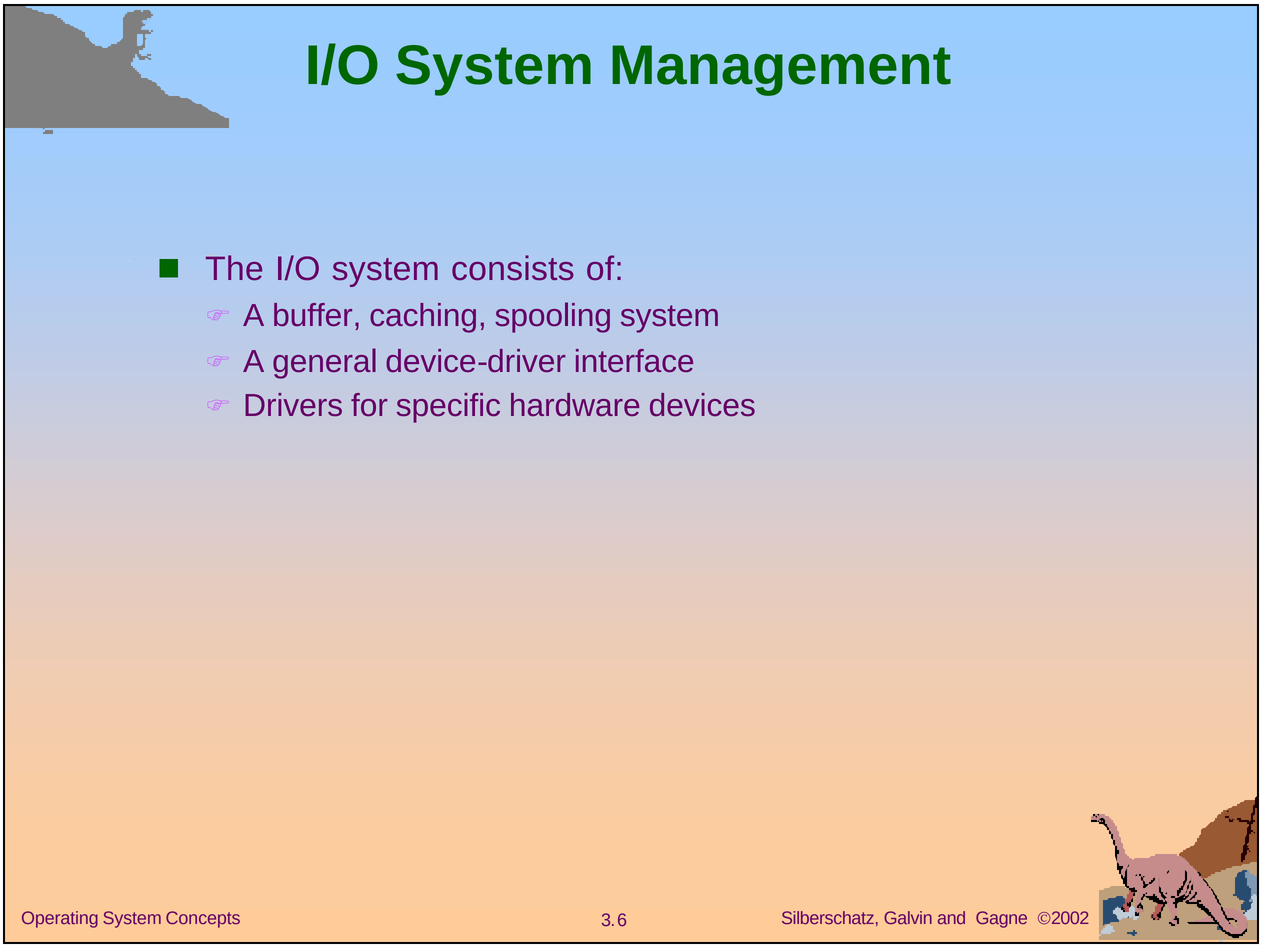
Main-Memory Management

- Memory is a large array of words or bytes, each with its own address. It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure.
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and deallocate memory space as needed.



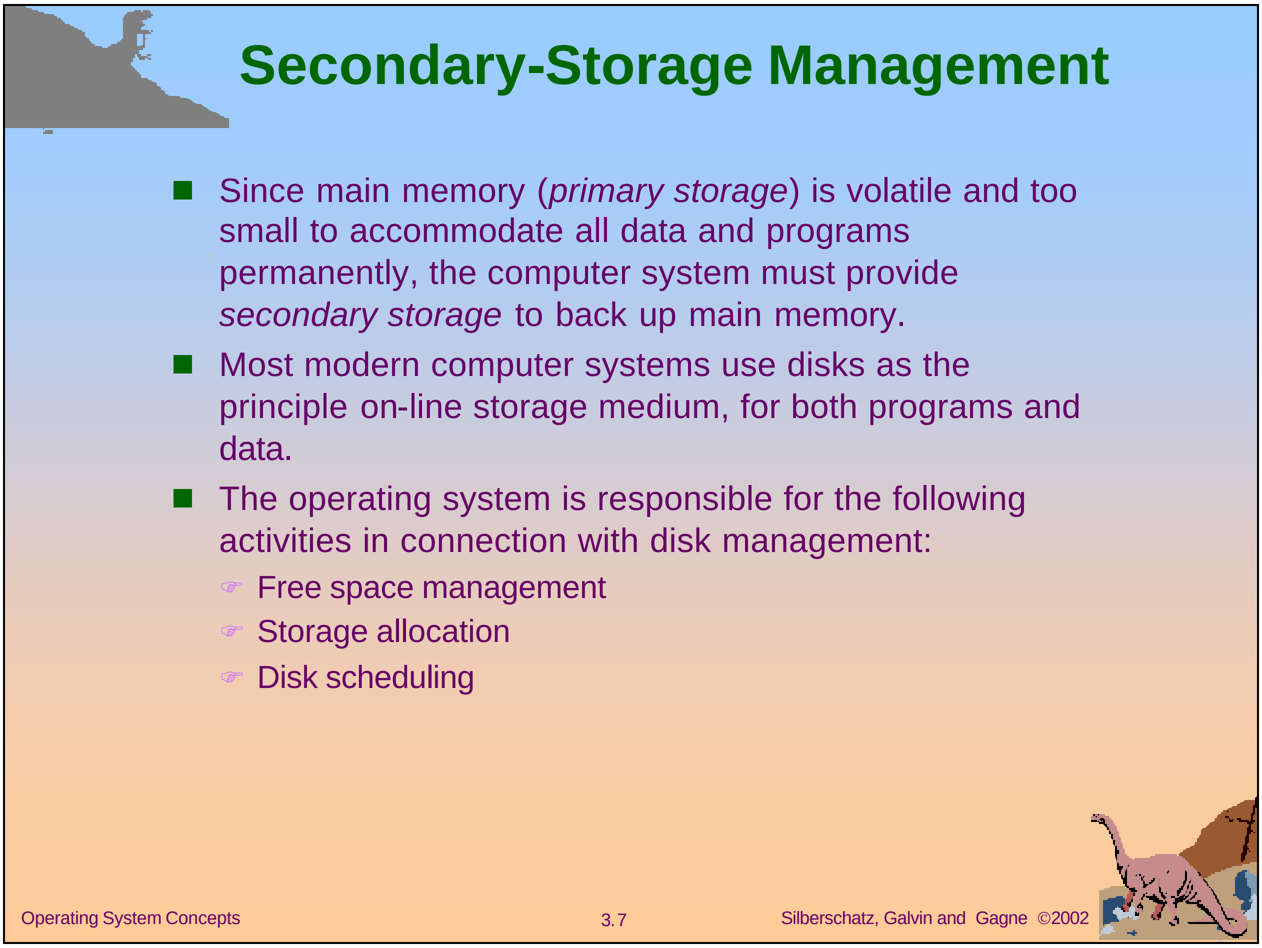
File Management

- A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.
- The operating system is responsible for the following activities in connections with file management:
  - File creation and deletion.
  - Directory creation and deletion.
  - Support of primitives for manipulating files and directories.
  - Mapping files onto secondary storage.
  - File backup on stable (nonvolatile) storage media.



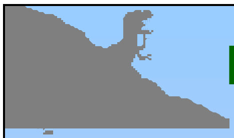
I/O System Management

- The I/O system consists of:
  - A buffer, caching, spooling system
  - A general device-driver interface
  - Drivers for specific hardware devices



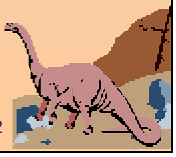
Secondary-Storage Management

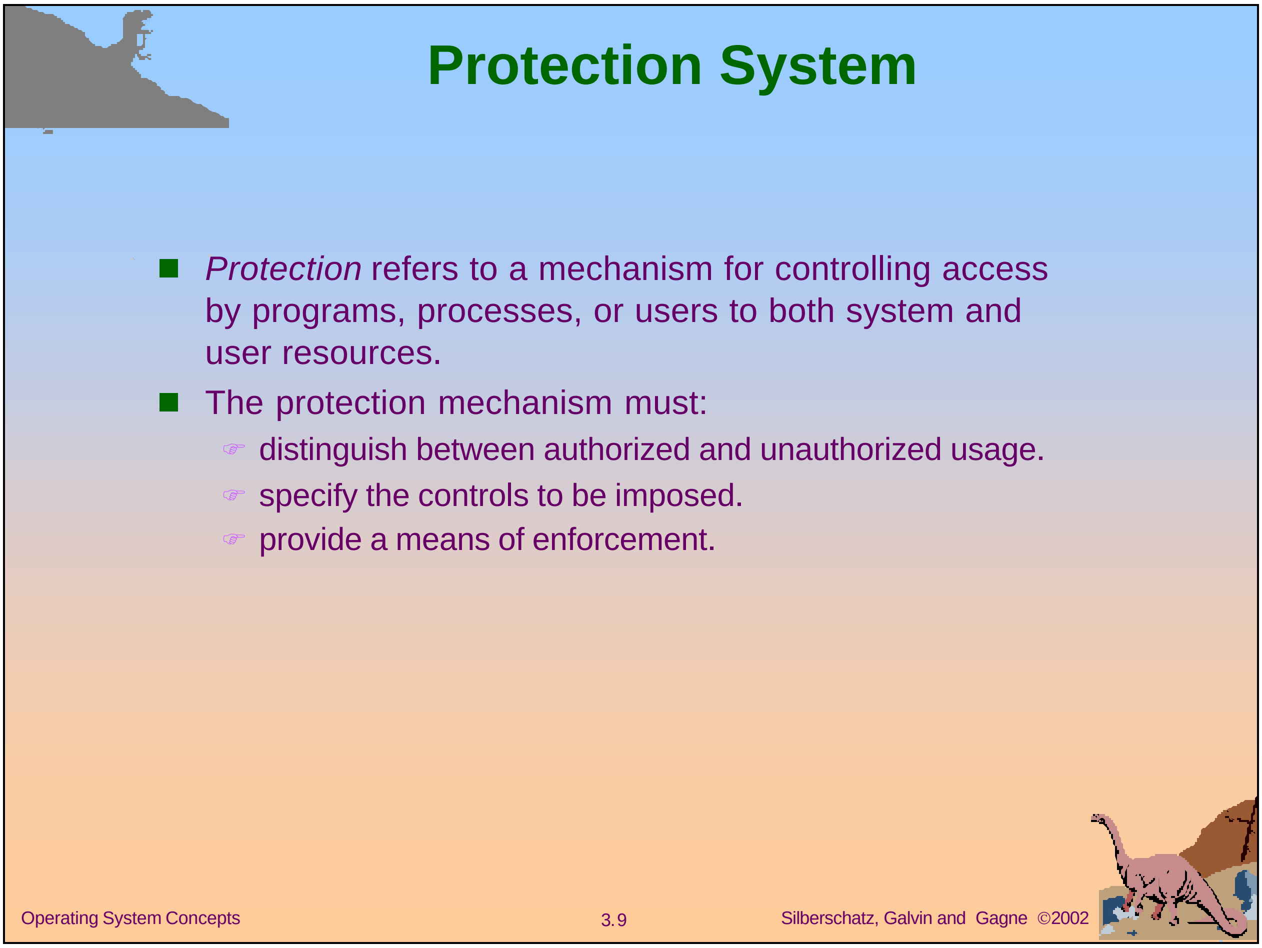
- Since main memory (*primary storage*) is volatile and too small to accommodate all data and programs permanently, the computer system must provide *secondary storage* to back up main memory.
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.
- The operating system is responsible for the following activities in connection with disk management:
  - Free space management
  - Storage allocation
  - Disk scheduling



Networking (Distributed Systems)

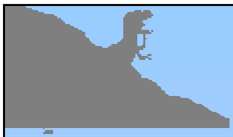
- A *distributed* system is a collection processors that do not share memory or a clock. Each processor has its own local memory.
- The processors in the system are connected through a communication network.
- Communication takes place using a *protocol (e.g. tcp/ip,ftp,nfs,http,rsh)*
- A distributed system provides user access to various system resources.
- Access to a shared resource allows:
  - ☞ Computation speed-up
  - ☞ Increased data availability
  - ☞ Enhanced reliability





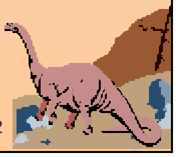
Protection System

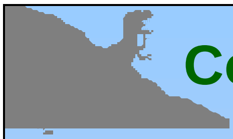
- *Protection* refers to a mechanism for controlling access by programs, processes, or users to both system and user resources.
- The protection mechanism must:
  - ☞ distinguish between authorized and unauthorized usage.
  - ☞ specify the controls to be imposed.
  - ☞ provide a means of enforcement.



Command-Interpreter System

- Many commands are given to the operating system by control statements which deal with:
  - process creation and management
  - I/O handling
  - secondary-storage management
  - main-memory management
  - file-system access
  - protection
  - Networking





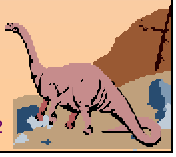
Command-Interpreter System (Cont.)

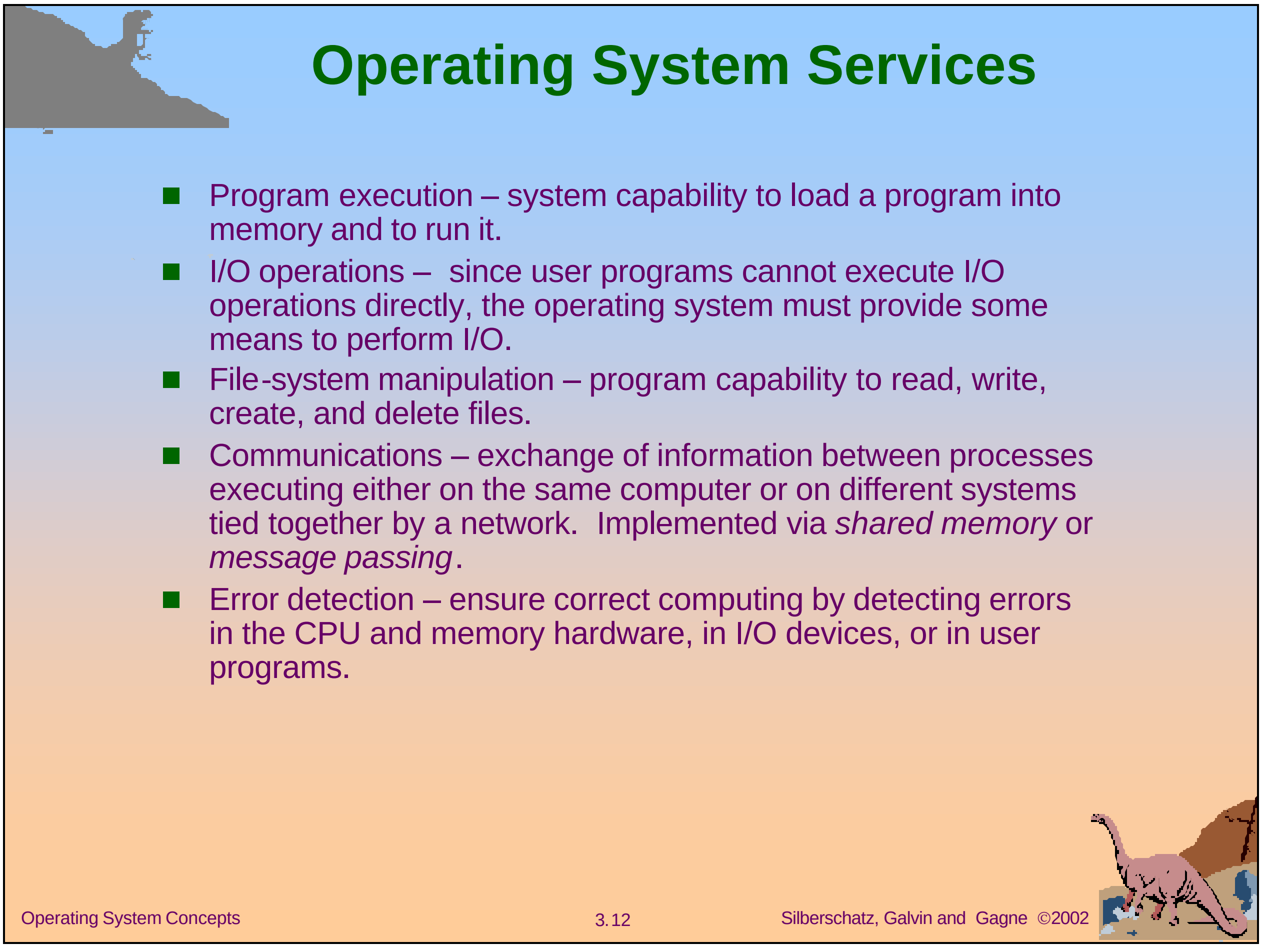
- The program that reads and interprets control statements is called variously:

  - command-line interpreter
  - shell (in UNIX)

  Its function is to get and execute the next command statement.

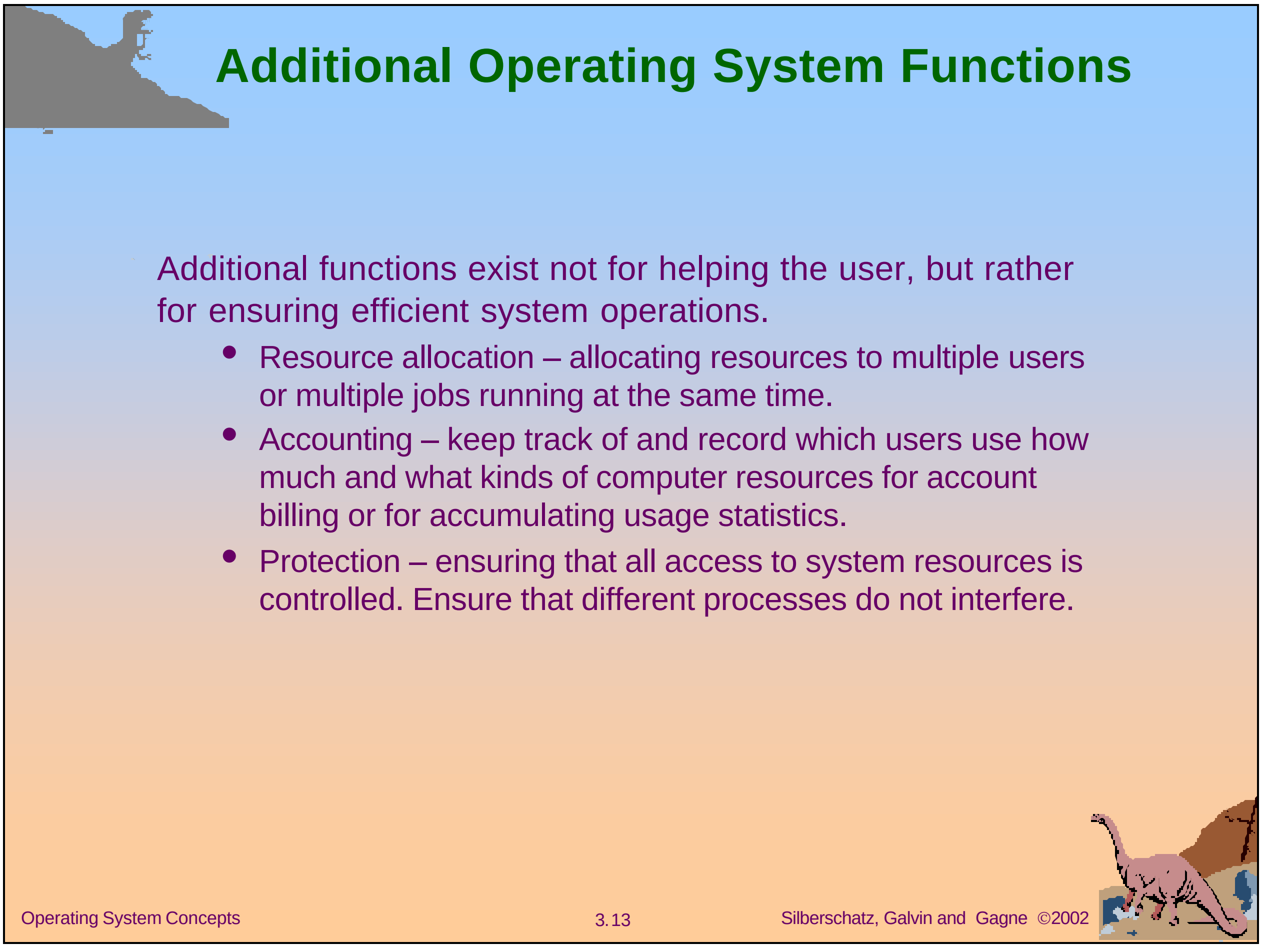
- GUI (Graphic User Interface) – mouse based window and menu driven (Windows, X windows)





Operating System Services

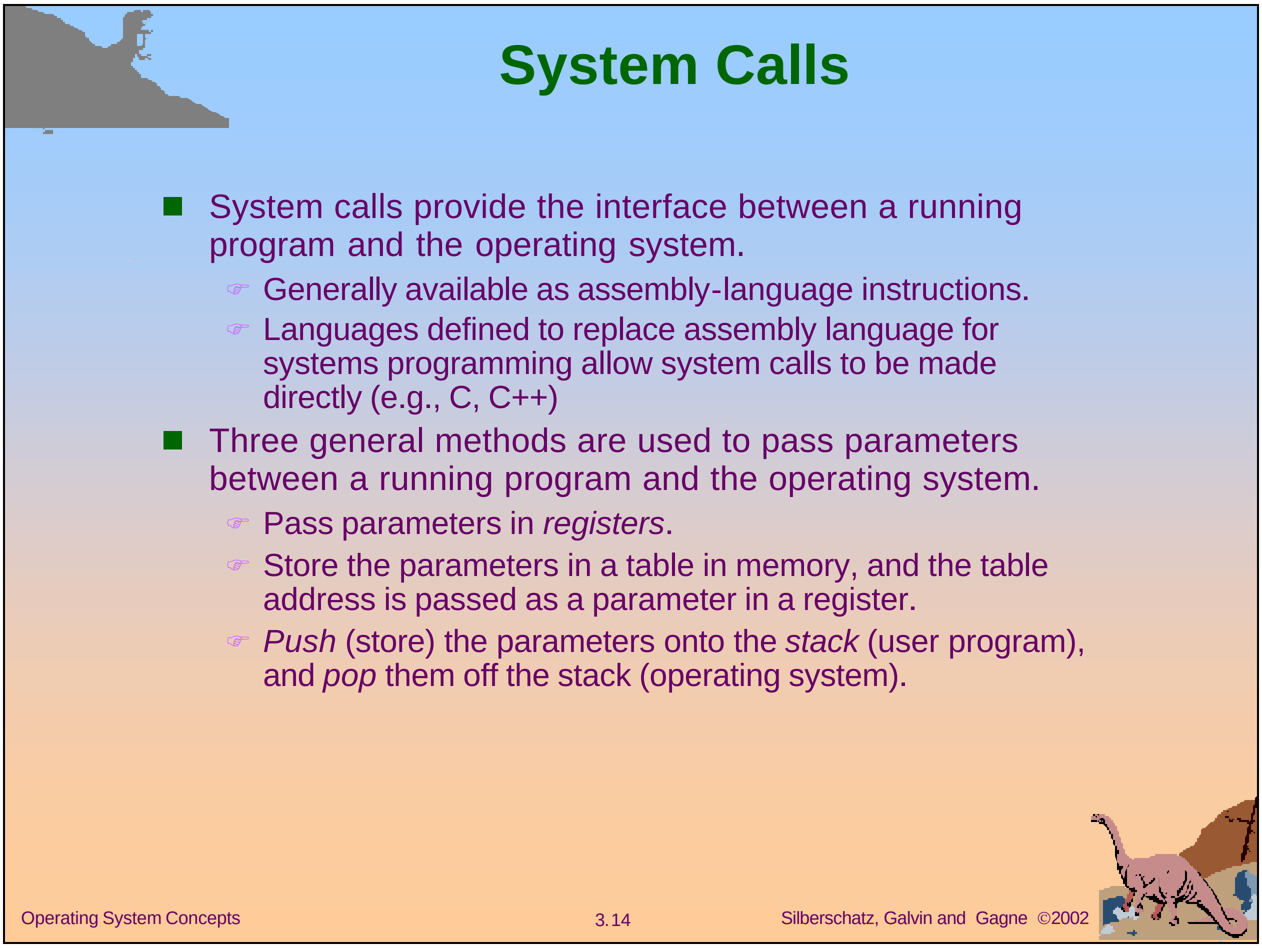
- Program execution – system capability to load a program into memory and to run it.
- I/O operations – since user programs cannot execute I/O operations directly, the operating system must provide some means to perform I/O.
- File-system manipulation – program capability to read, write, create, and delete files.
- Communications – exchange of information between processes executing either on the same computer or on different systems tied together by a network. Implemented via *shared memory* or *message passing*.
- Error detection – ensure correct computing by detecting errors in the CPU and memory hardware, in I/O devices, or in user programs.



Additional Operating System Functions

Additional functions exist not for helping the user, but rather for ensuring efficient system operations.

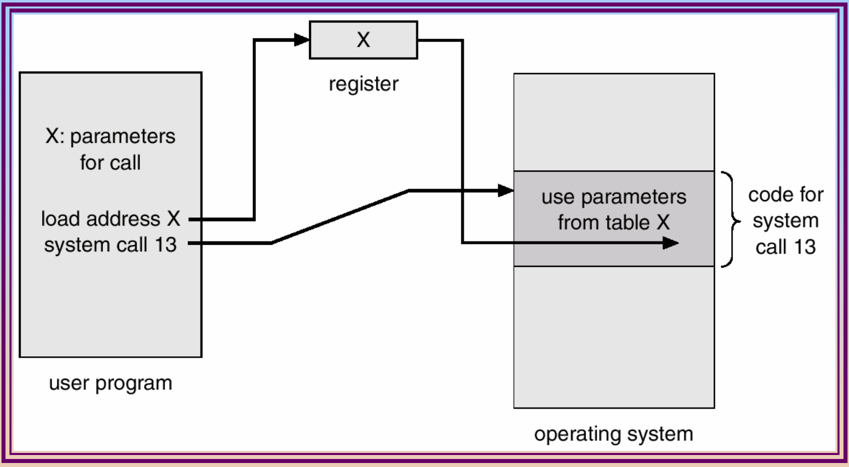
- Resource allocation – allocating resources to multiple users or multiple jobs running at the same time.
- Accounting – keep track of and record which users use how much and what kinds of computer resources for account billing or for accumulating usage statistics.
- Protection – ensuring that all access to system resources is controlled. Ensure that different processes do not interfere.

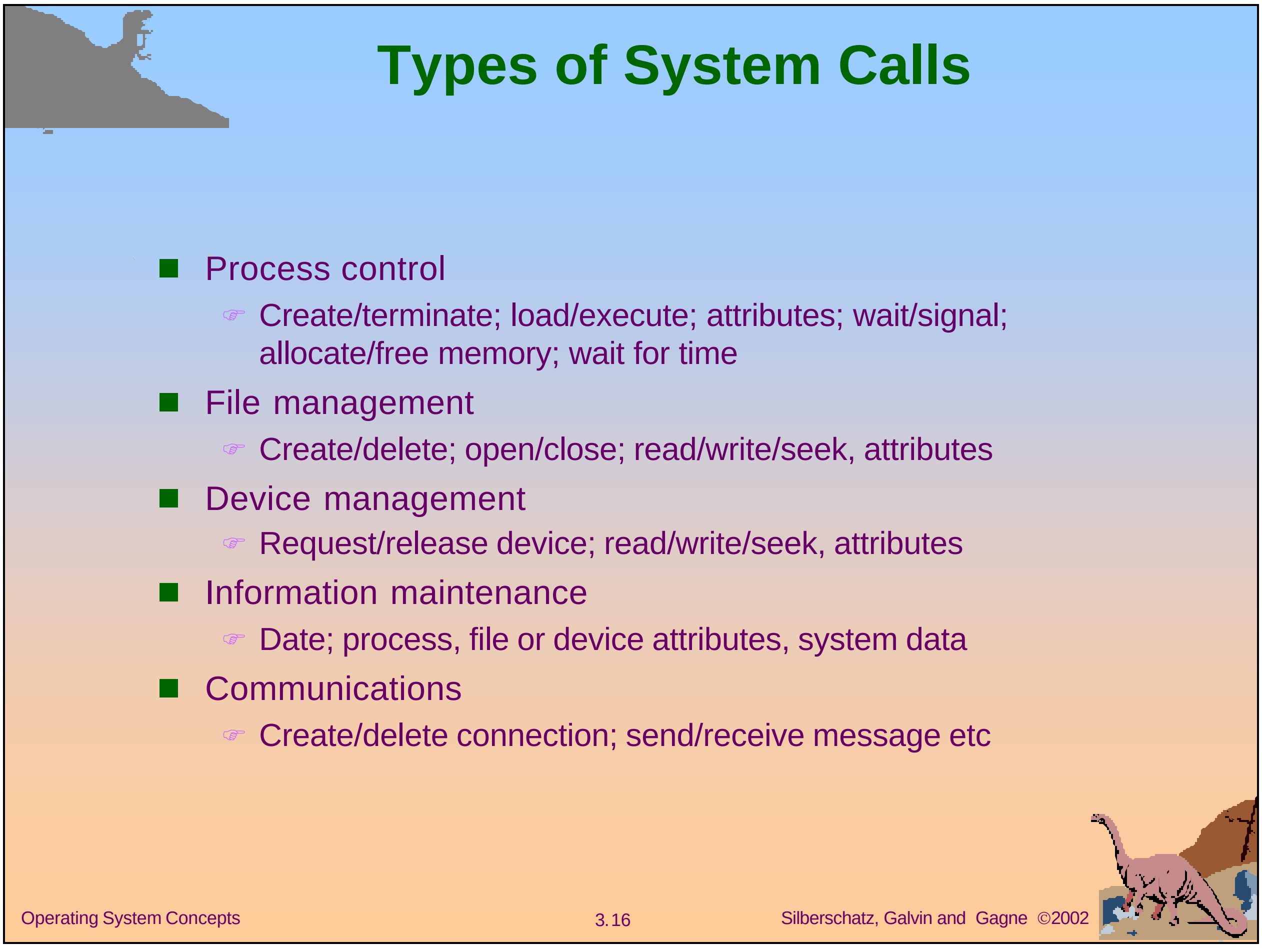


System Calls

- 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++)
- 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* (user program), and *pop* them off the stack (operating system).

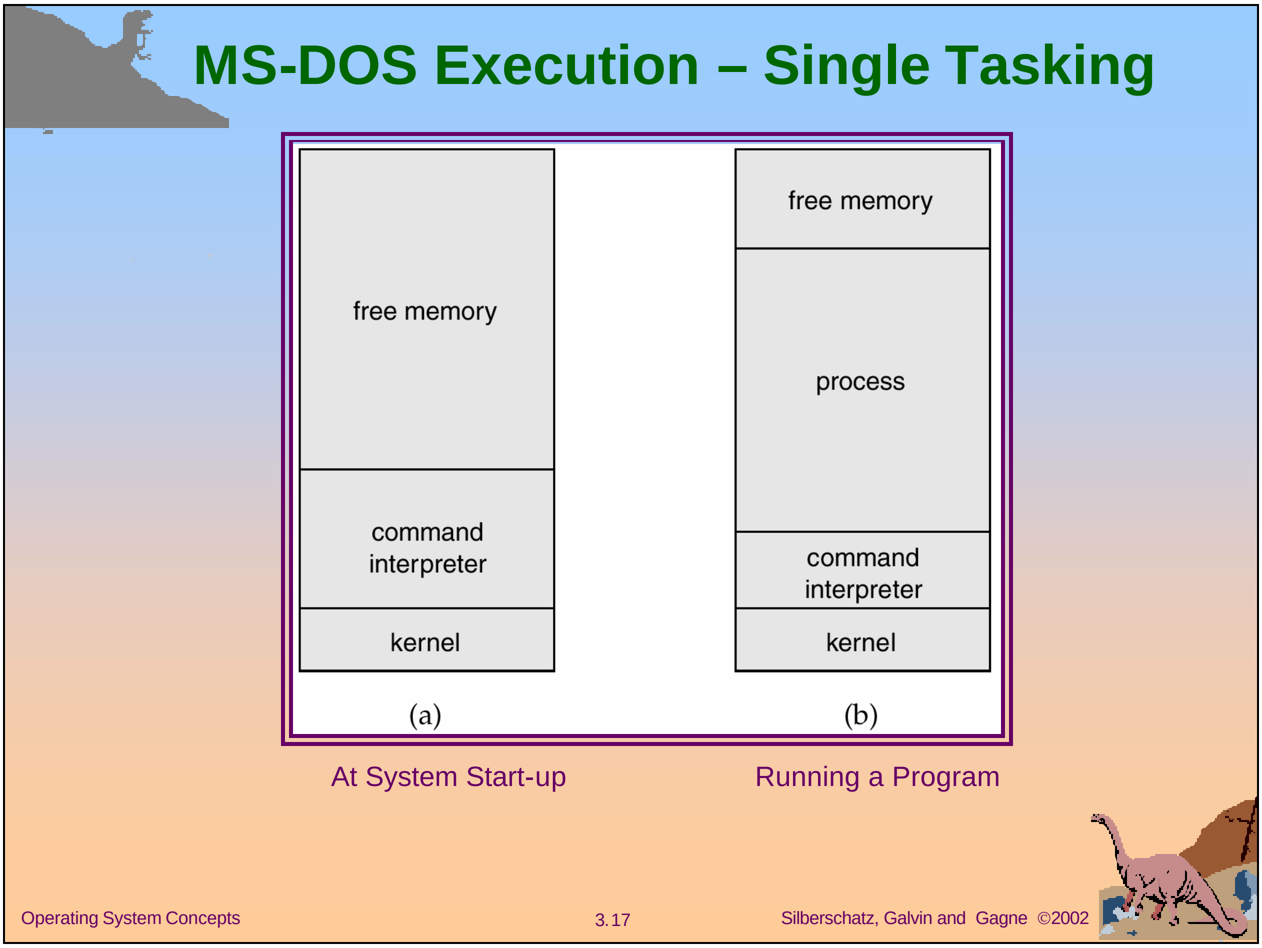
Passing of Parameters As A Table





Types of System Calls

- Process control
  - ☞ Create/terminate; load/execute; attributes; wait/signal; allocate/free memory; wait for time
- File management
  - ☞ Create/delete; open/close; read/write/seek, attributes
- Device management
  - ☞ Request/release device; read/write/seek, attributes
- Information maintenance
  - ☞ Date; process, file or device attributes, system data
- Communications
  - ☞ Create/delete connection; send/receive message etc



MS-DOS Execution – Single Tasking

free memory

command interpreter

kernel

(a)

free memory

process

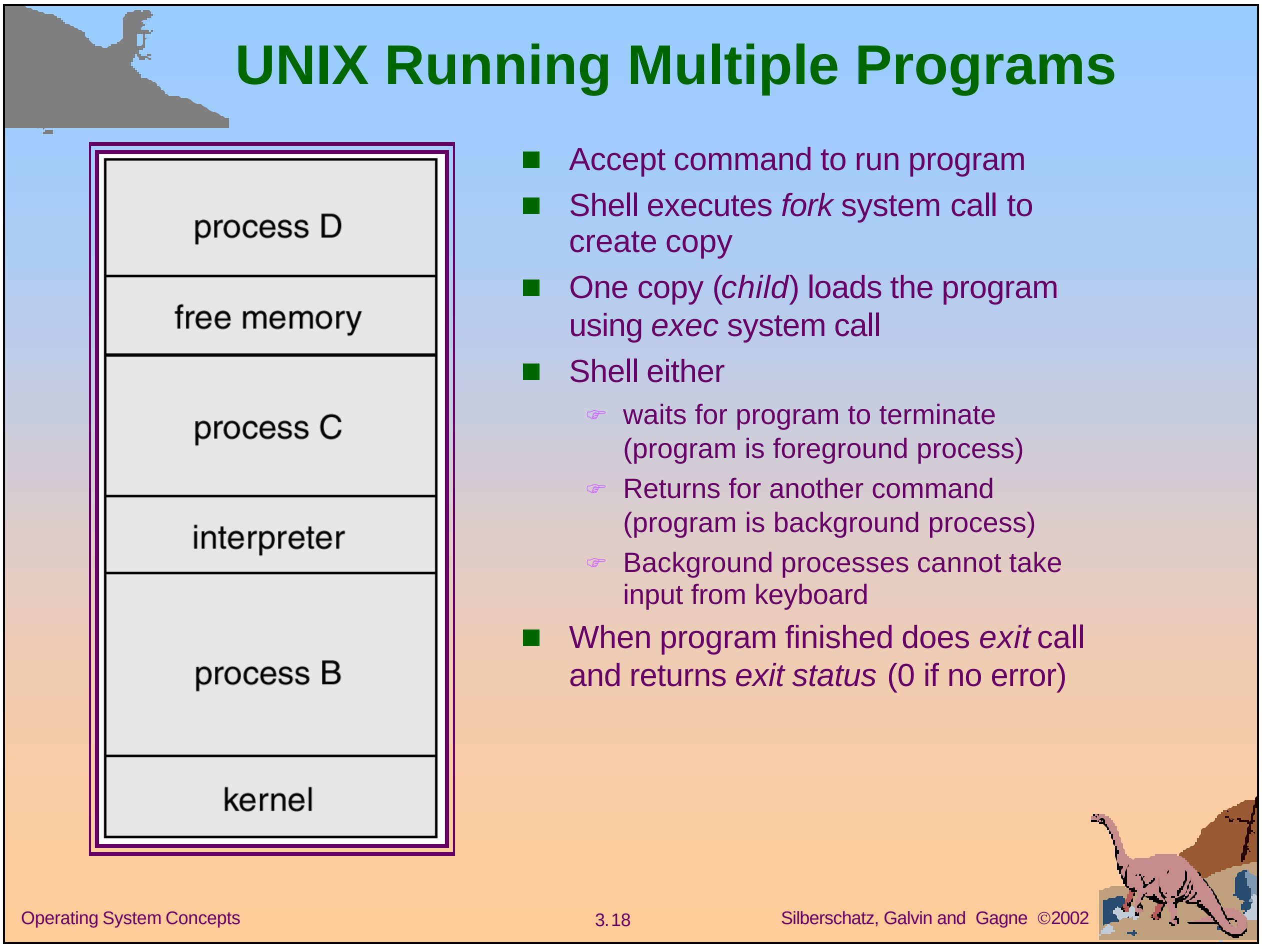
command interpreter

kernel

(b)

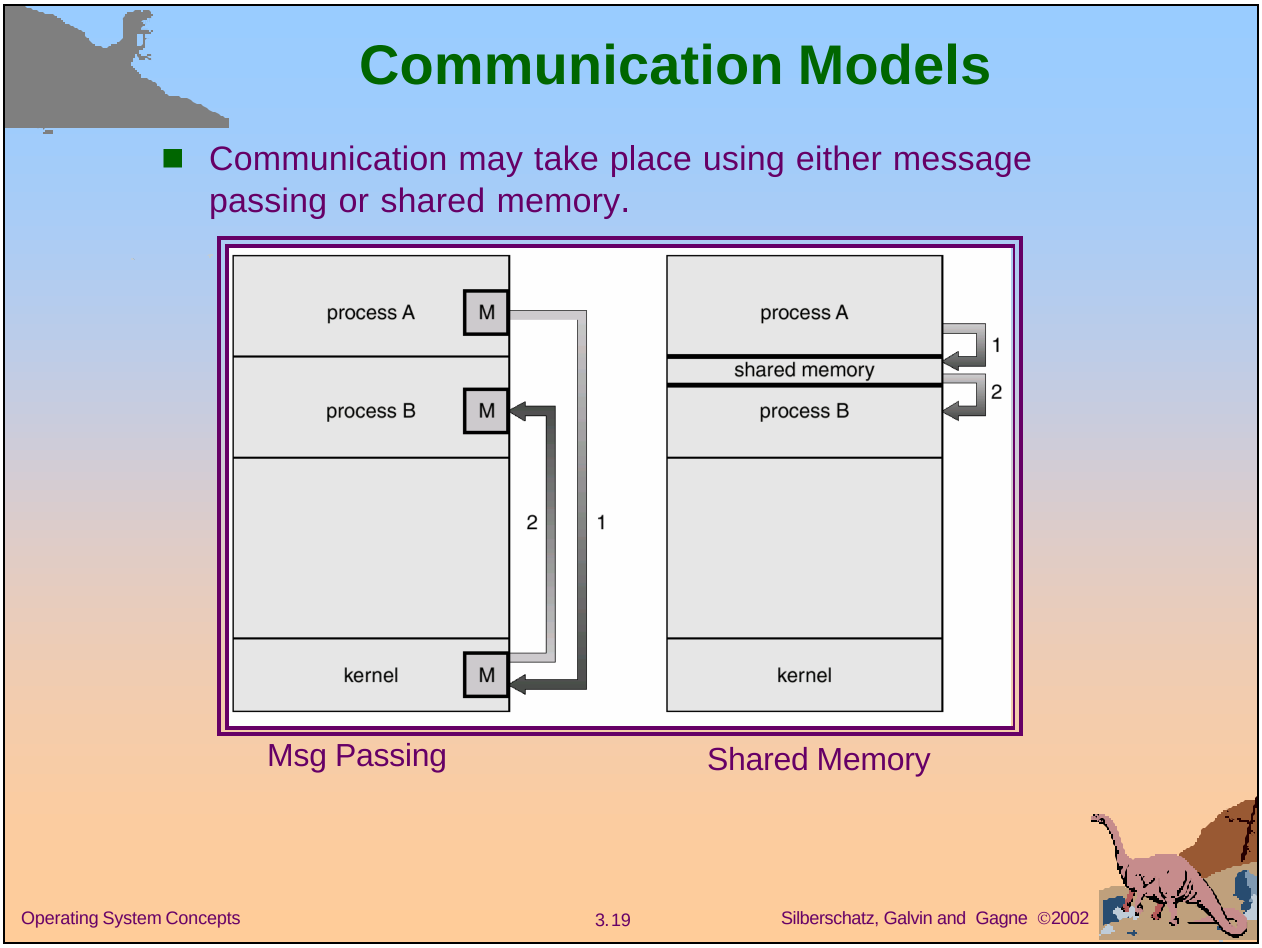
At System Start-up

Running a Program



UNIX Running Multiple Programs

- Accept command to run program
- Shell executes *fork* system call to create copy
- One copy (*child*) loads the program using *exec* system call
- Shell either
  - waits for program to terminate (program is foreground process)
  - Returns for another command (program is background process)
  - Background processes cannot take input from keyboard
- When program finished does *exit* call and returns *exit status* (0 if no error)

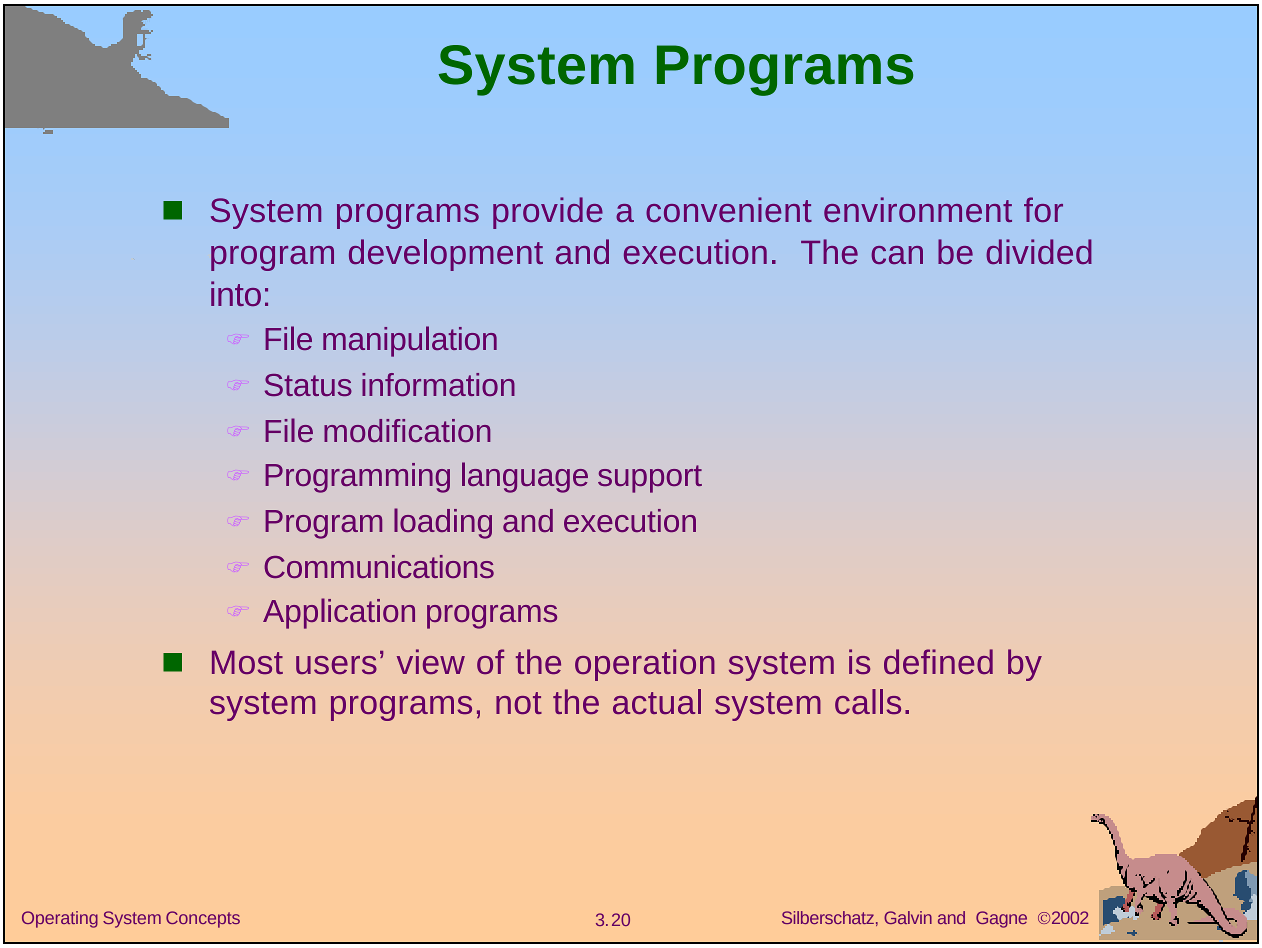


Communication Models

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

Msg Passing

Shared Memory

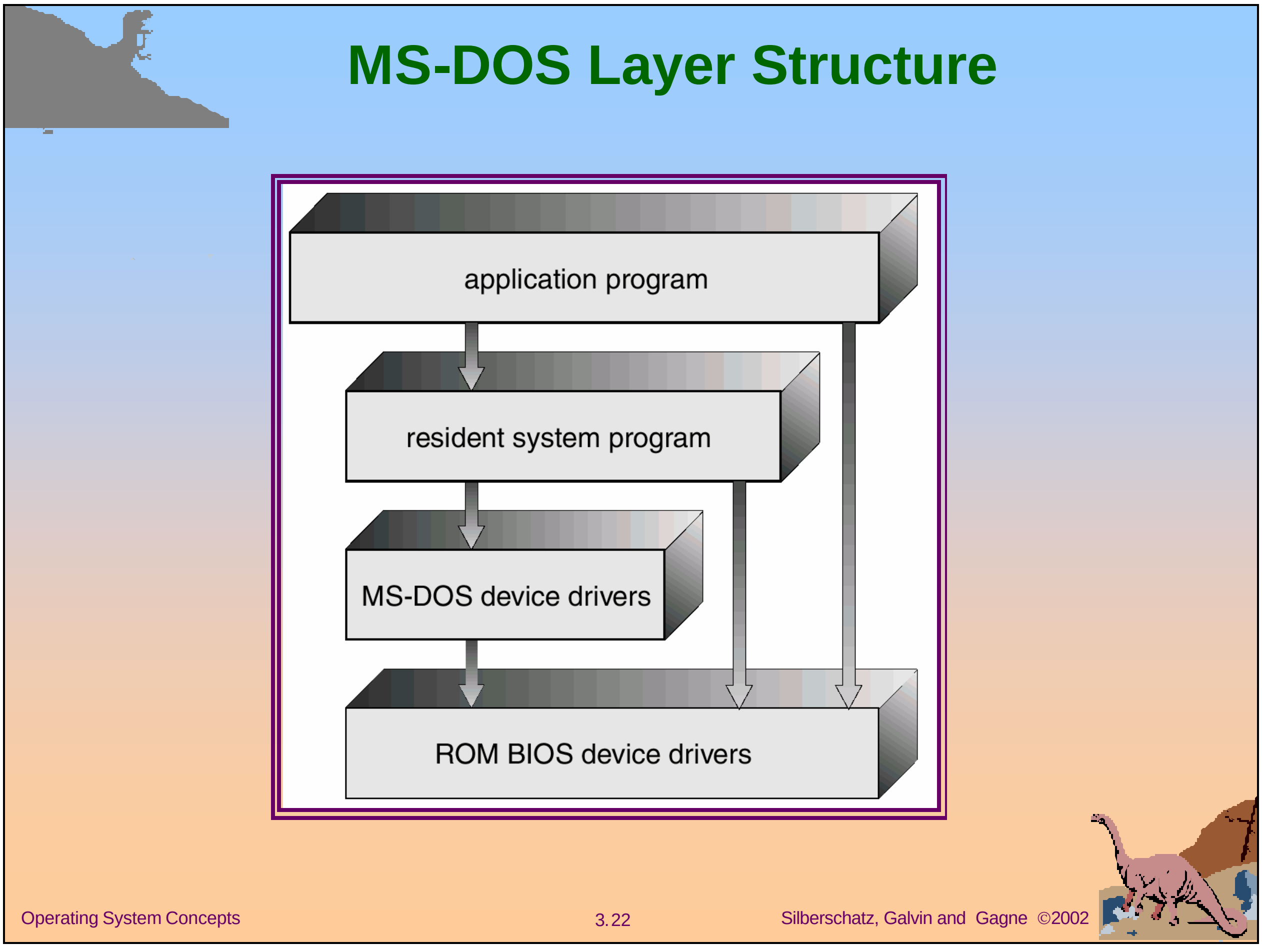


System Programs

- 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
- 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



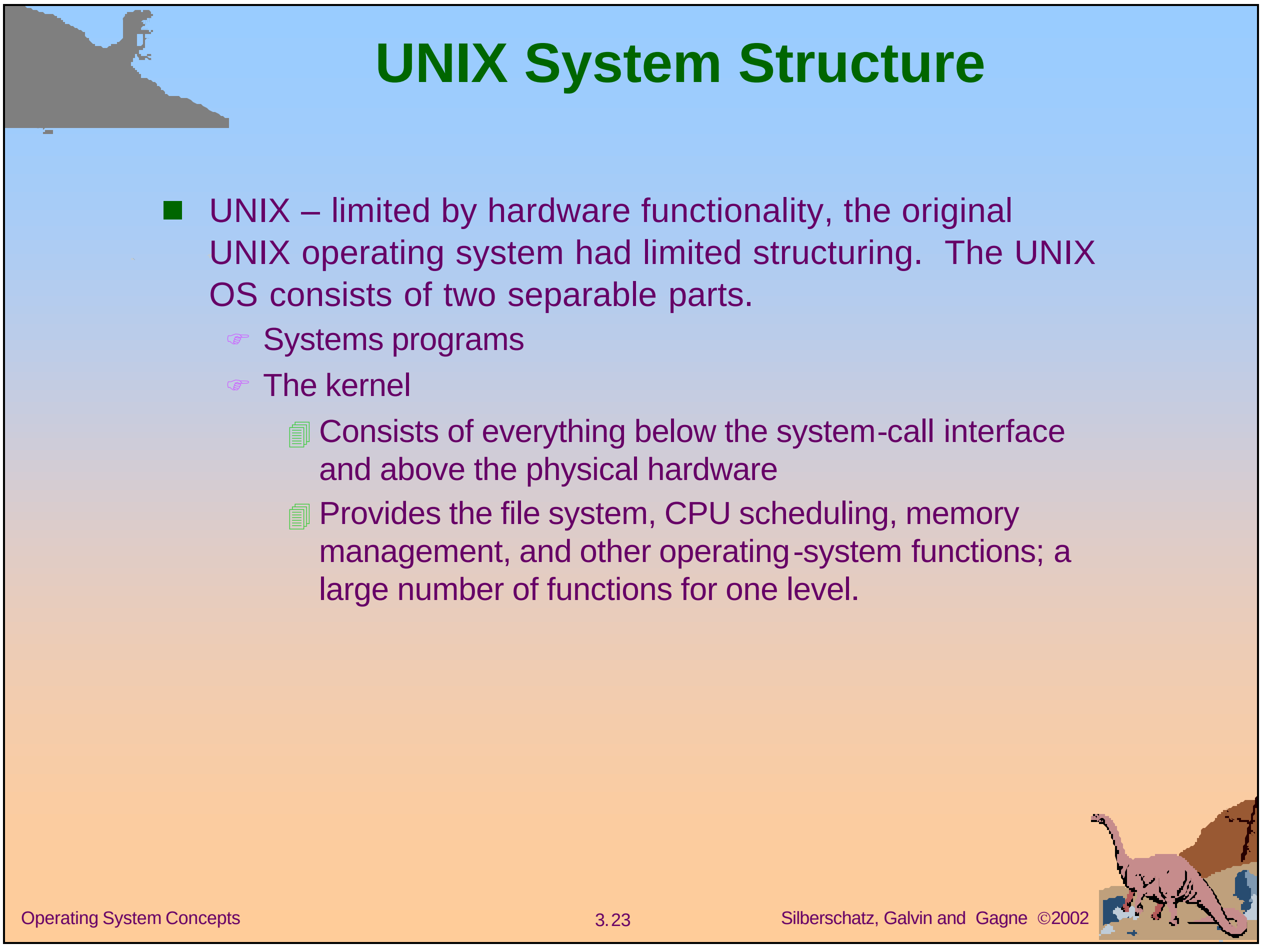
MS-DOS Layer Structure

application program

resident system program

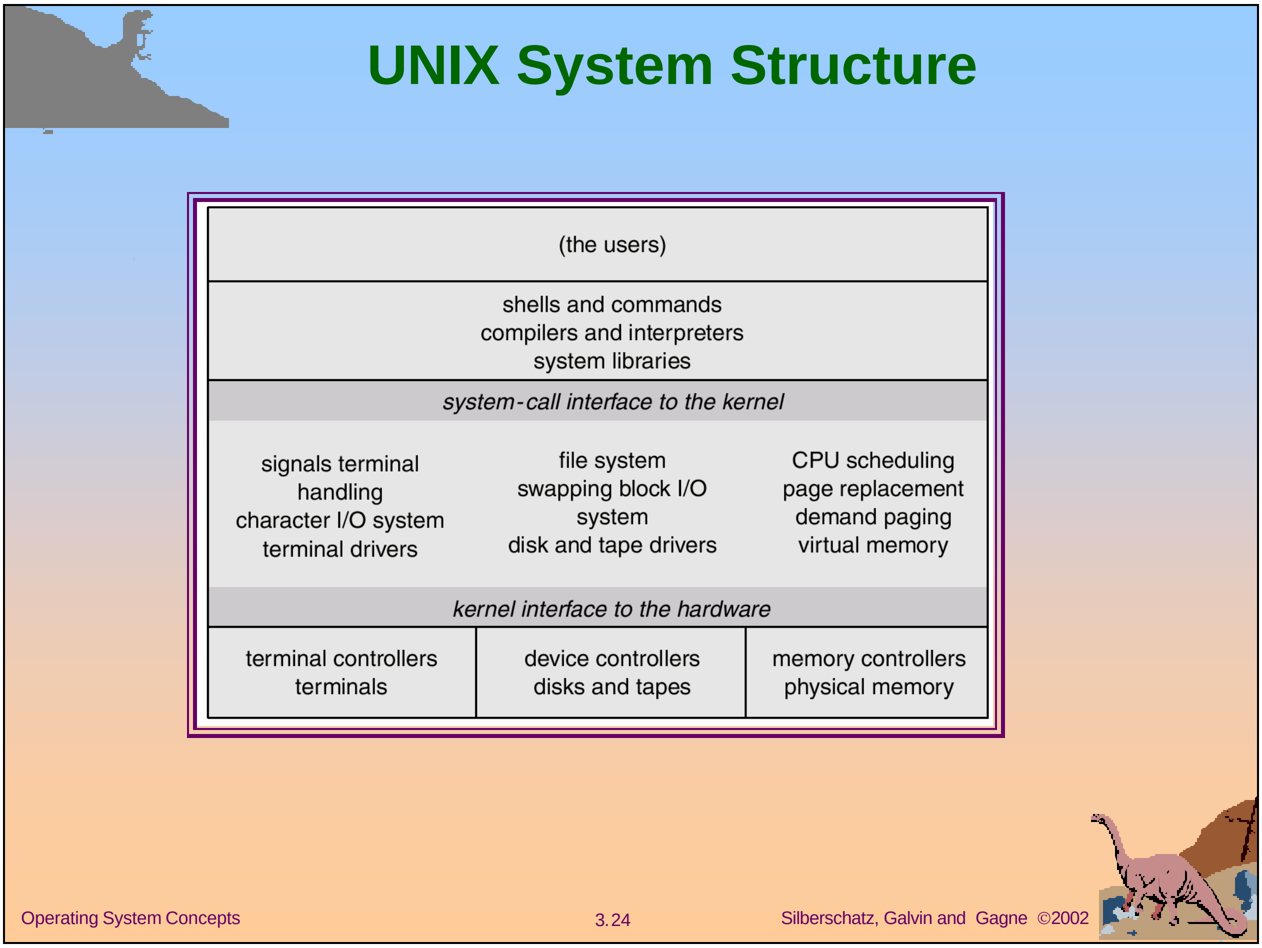
MS-DOS device drivers

ROM BIOS device drivers



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

(the users)

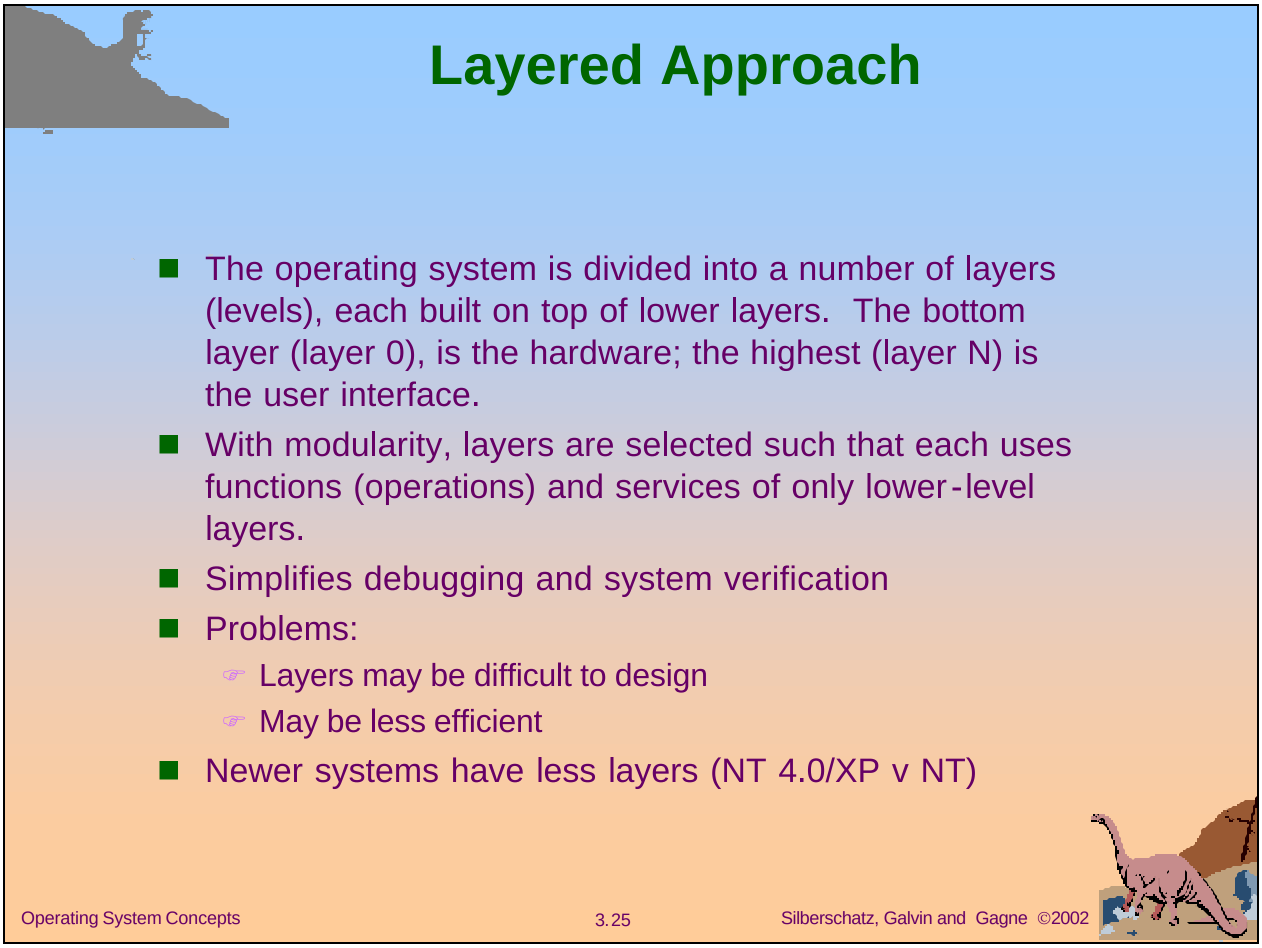
shells and commands
compilers and interpreters
system libraries

system-call interface to the kernel

signals terminal handling character I/O system terminal drivers	file system swapping block I/O system disk and tape drivers	CPU scheduling page replacement demand paging virtual memory

kernel interface to the hardware

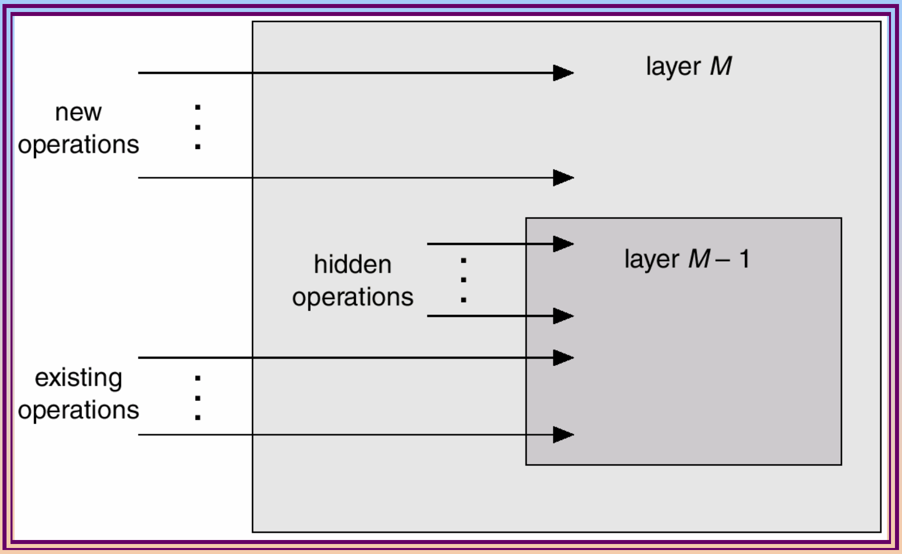
terminal controllers terminals	device controllers disks and tapes	memory controllers physical memory



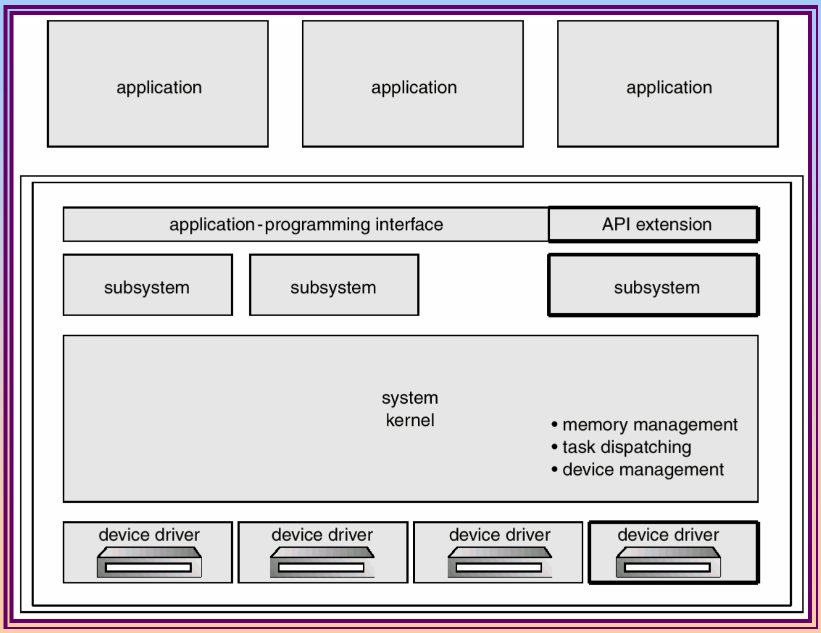
Layered Approach

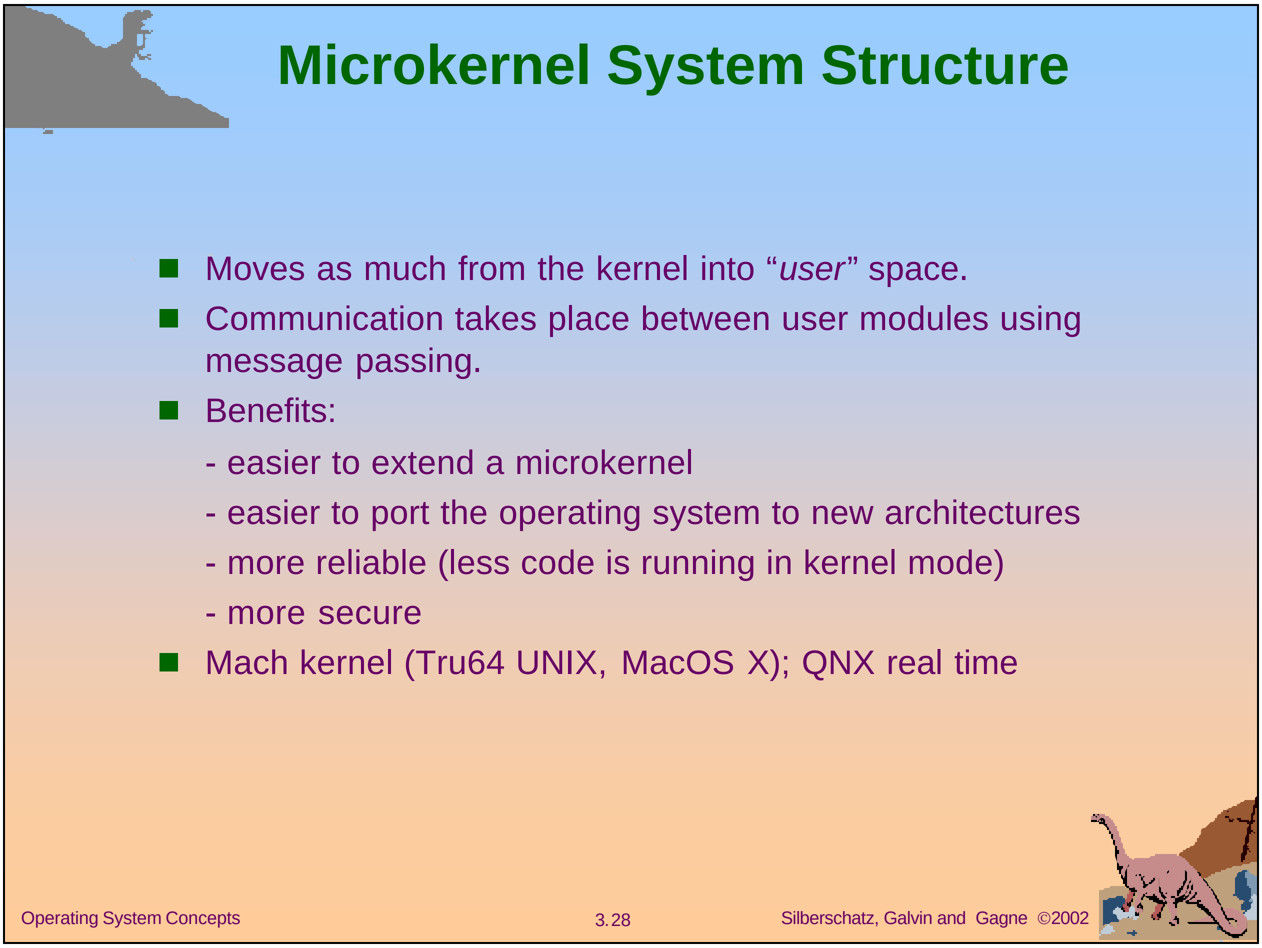
- The operating system is divided into a number of layers (levels), each built on top of lower layers. The bottom layer (layer 0), is the hardware; the highest (layer N) is the user interface.
- With modularity, layers are selected such that each uses functions (operations) and services of only lower-level layers.
- Simplifies debugging and system verification
- Problems:
  - Layers may be difficult to design
  - May be less efficient
- Newer systems have less layers (NT 4.0/XP v NT)

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:

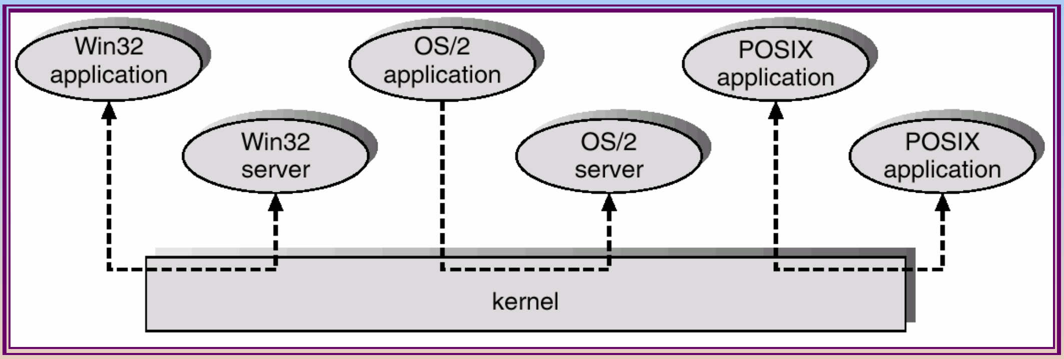
  - easier to extend a microkernel

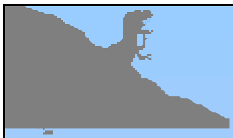
  - easier to port the operating system to new architectures

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

  - more secure
- Mach kernel (Tru64 UNIX, MacOS X); QNX real time

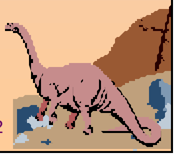
Windows NT Client-Server Structure

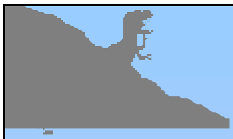




Virtual Machines

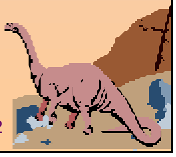
- A *virtual machine* takes the layered approach to its logical conclusion. It treats hardware and the operating system kernel as though they were all hardware.
- A virtual machine provides an interface *identical* to the underlying bare hardware.
- The operating system creates the illusion of multiple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



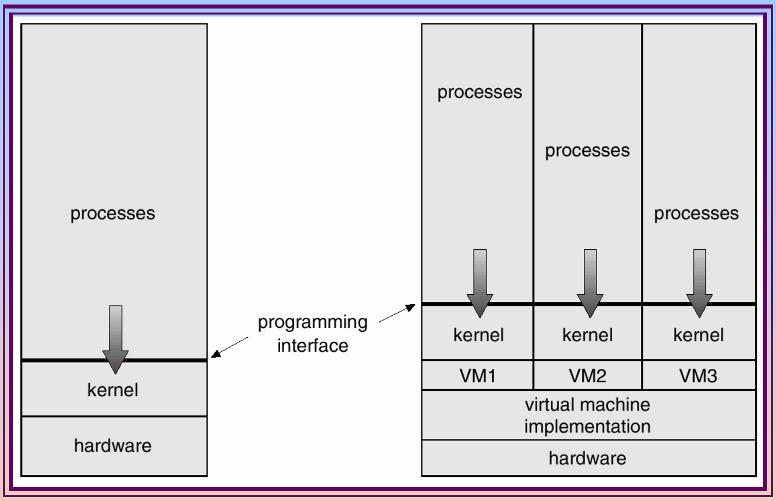


Virtual Machines (Cont.)

- 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.
- Issue with disks – IBM VM creates *minidisks* by allocating tracks from disk
- Users can run any OS available on underlying machine (VM/CMS)



System Models



processes

kernel

hardware

programming interface

processes

processes

processes

kernel

kernel

kernel

VM1

VM2

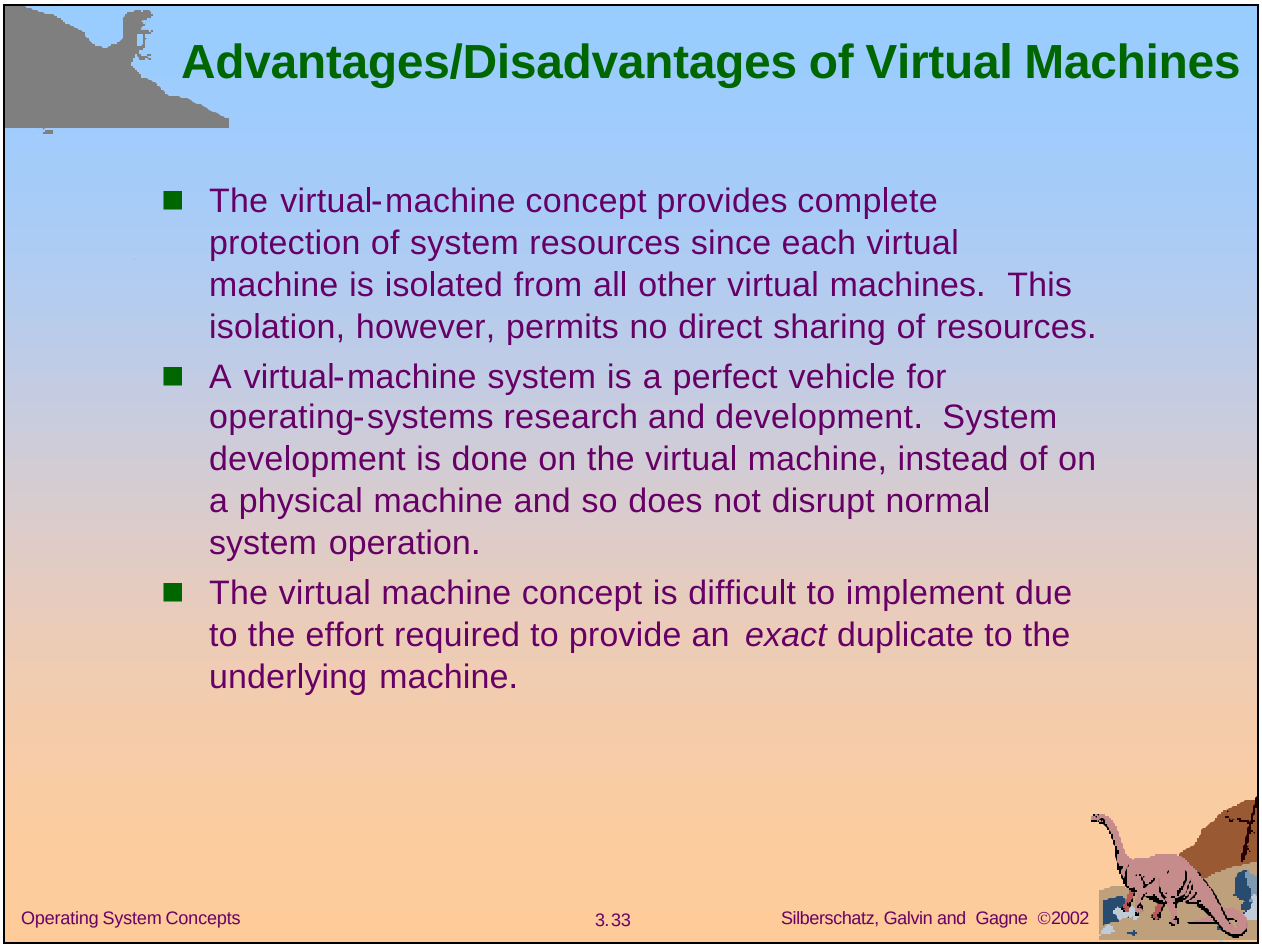
VM3

virtual machine implementation

hardware

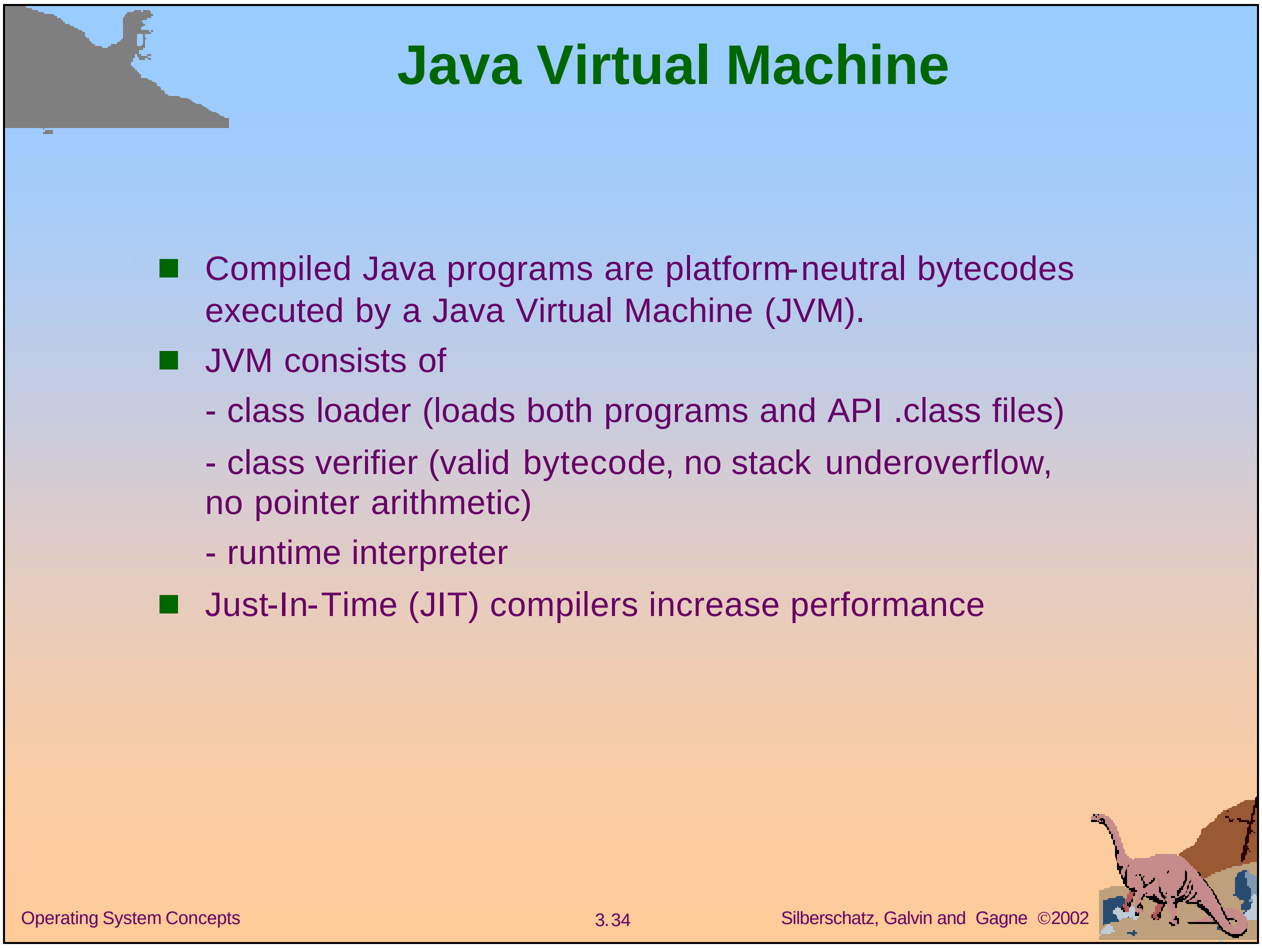
Non-virtual Machine

Virtual Machine



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



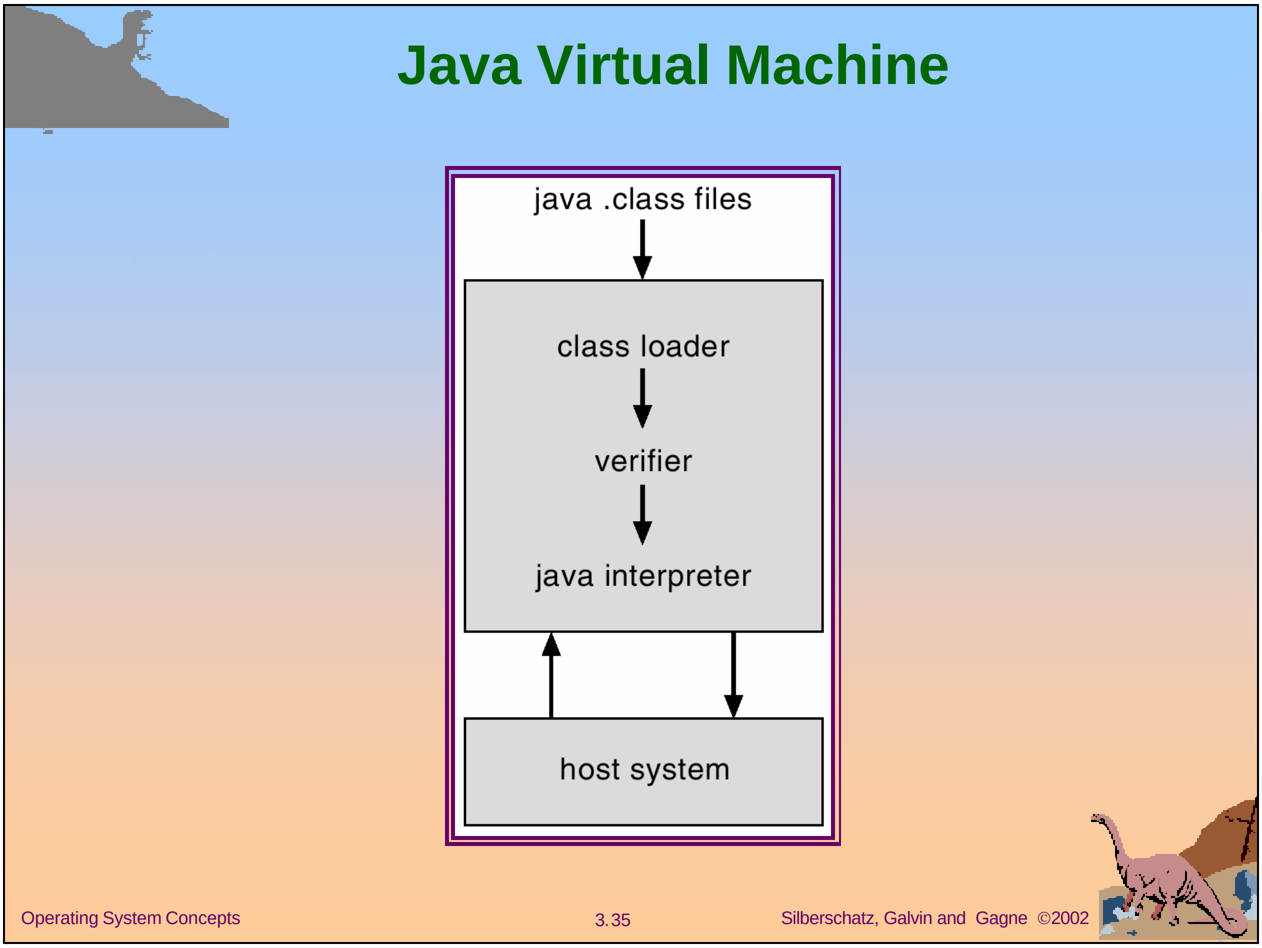
Java Virtual Machine

- Compiled Java programs are platform-neutral bytecodes executed by a Java Virtual Machine (JVM).
- JVM consists of

  - class loader (loads both programs and API .class files)

  - class verifier (valid bytecode, no stack underoverflow, no pointer arithmetic)

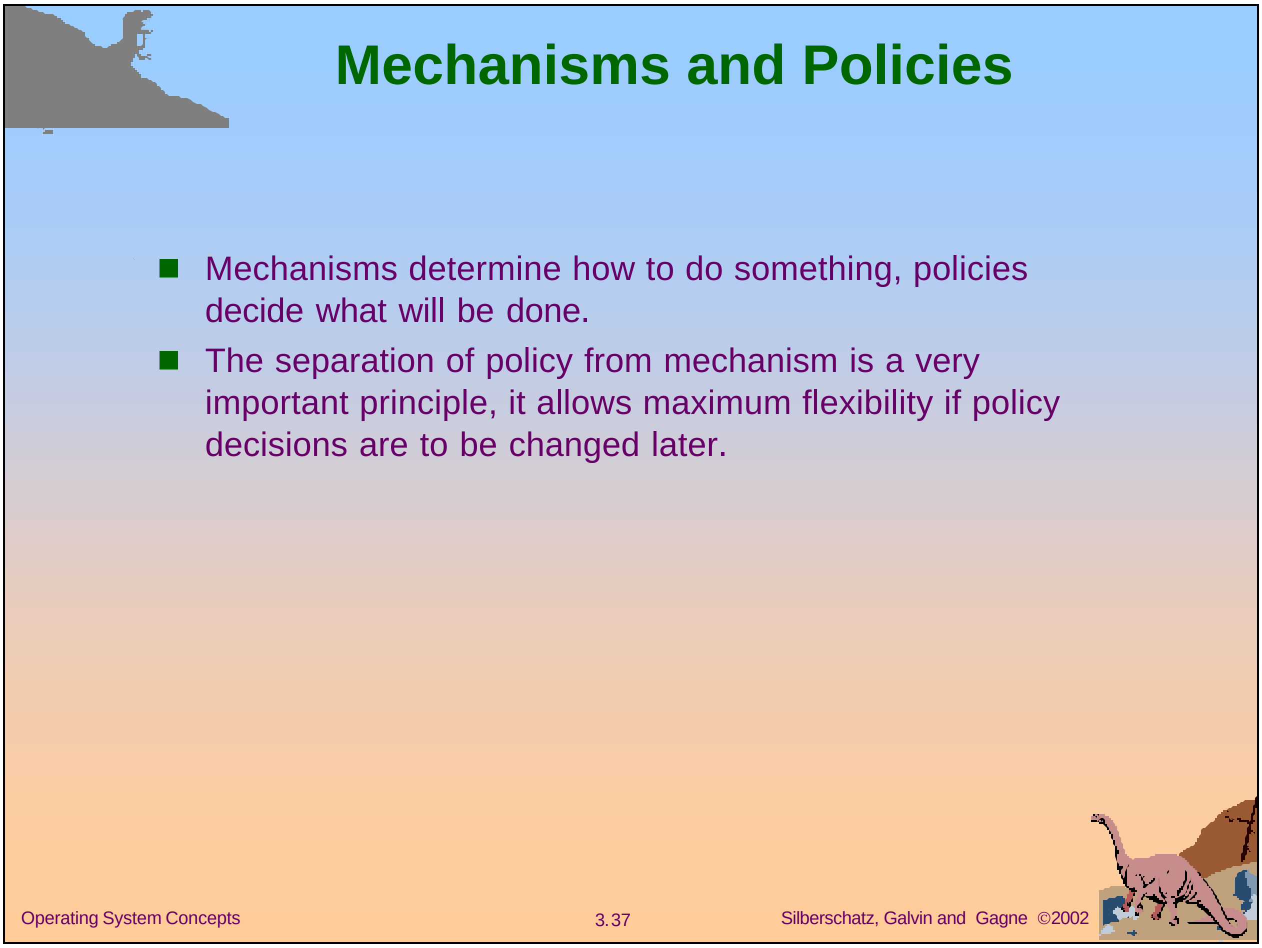
  - runtime interpreter
- Just-In-Time (JIT) compilers increase performance



Java Virtual Machine

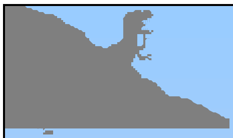
System Design Goals

- 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.



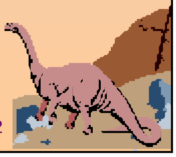
Mechanisms and Policies

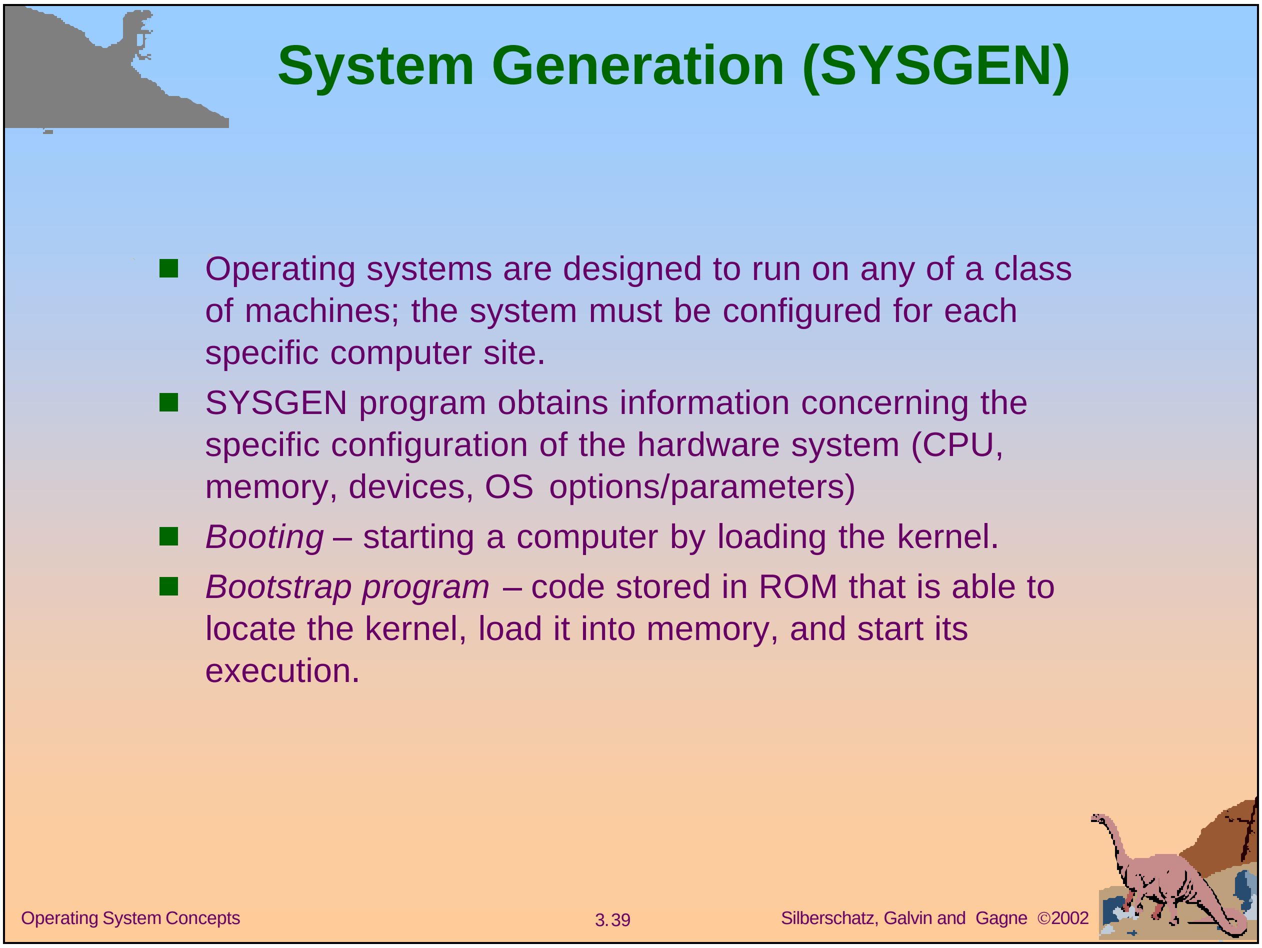
- Mechanisms determine how to do something, policies decide what will be done.
- The separation of policy from mechanism is a very important principle, it allows maximum flexibility if policy decisions are to be changed later.



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 (CPU, memory, devices, OS options/parameters)
- *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.