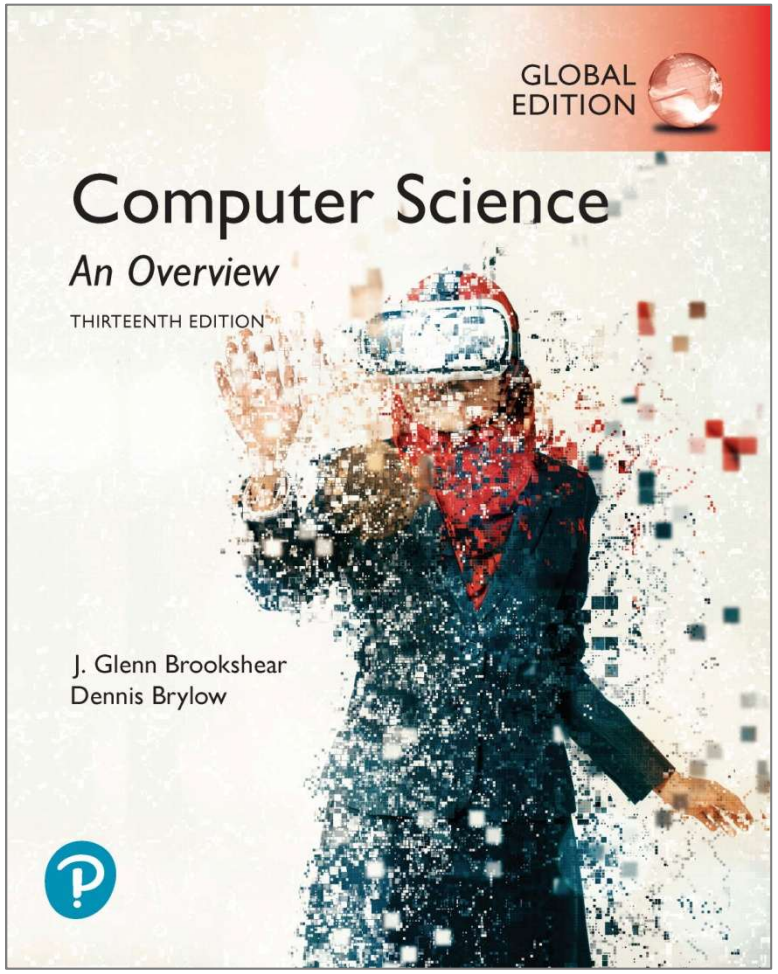
Computer Science An Overview

13th Edition, Global Edition



Chapter 3

Operating Systems

Pearson

Examples of Operating Systems

- Windows
- UNIX
- Mac OS
- Solaris (Sun/Oracle machines)
- Linux

Smartphone Operating Systems

- Apple iOS
- Windows Phone
- BlackBerry OS
- Nokia Symbian OS
- Google Android

Functions of Operating Systems

- Oversee operation of computer
- Store and retrieve files
- Provide the user interface to request execution of programs
- Coordinate the execution of programs

Main Components of an Operating System

- Kernel
  - The core of the operating system
  - Manages communication between hardware and software
  - Controls CPU, memory, and device access
- Process Management
  - Creates, schedules, and terminates processes
  - Allocates CPU time (scheduling)
  - Supports multitasking
- Memory Management
  - Allocates and deallocates main memory (RAM)
  - Manages virtual memory
  - Ensures memory protection

Main Components of an Operating System

- File System
  - Organizes and manages data on storage devices
  - Handles file creation, deletion, reading, and writing
- I/O and Device Management
  - Controls input/output devices
  - Uses device drivers
  - Manages buffering and caching
- Security and Protection
  - User authentication and authorization
  - Protects system resources from unauthorized access
- User Interface
  - Allows users to interact with the system

3.1 History of Operating Systems

- Each program is called a “job”
- Early computers required significant setup time
- Each “job” required its own setup
- Operating Systems began as systems for simplifying setup and transitions between jobs

3.1 History of Operating Systems

- Batch processing (job queue)
- Interactive processing (real time)
- Time-sharing (one machine, many users)
- Multitasking (one user, many tasks)
- Multiprocessor machines (load balancing)
- Embedded Systems (specific devices)

Figure 3.1 Batch processing

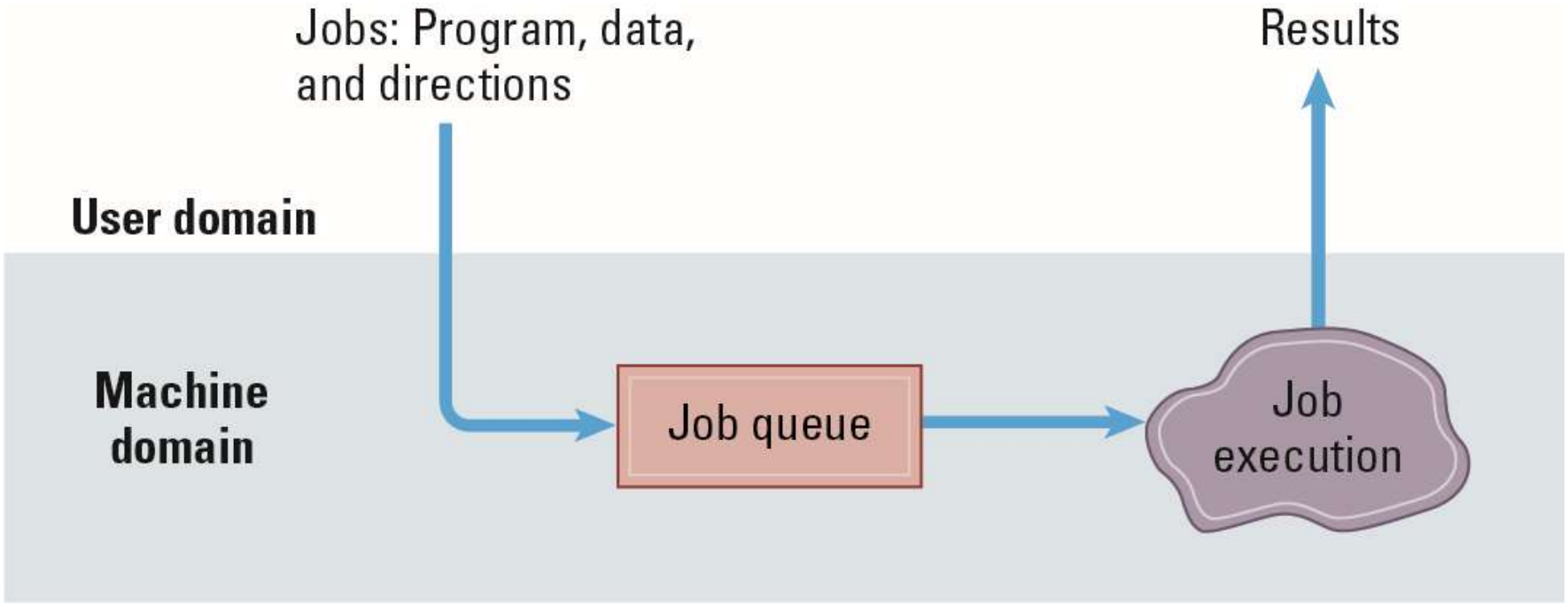


Figure 3.2 Interactive processing

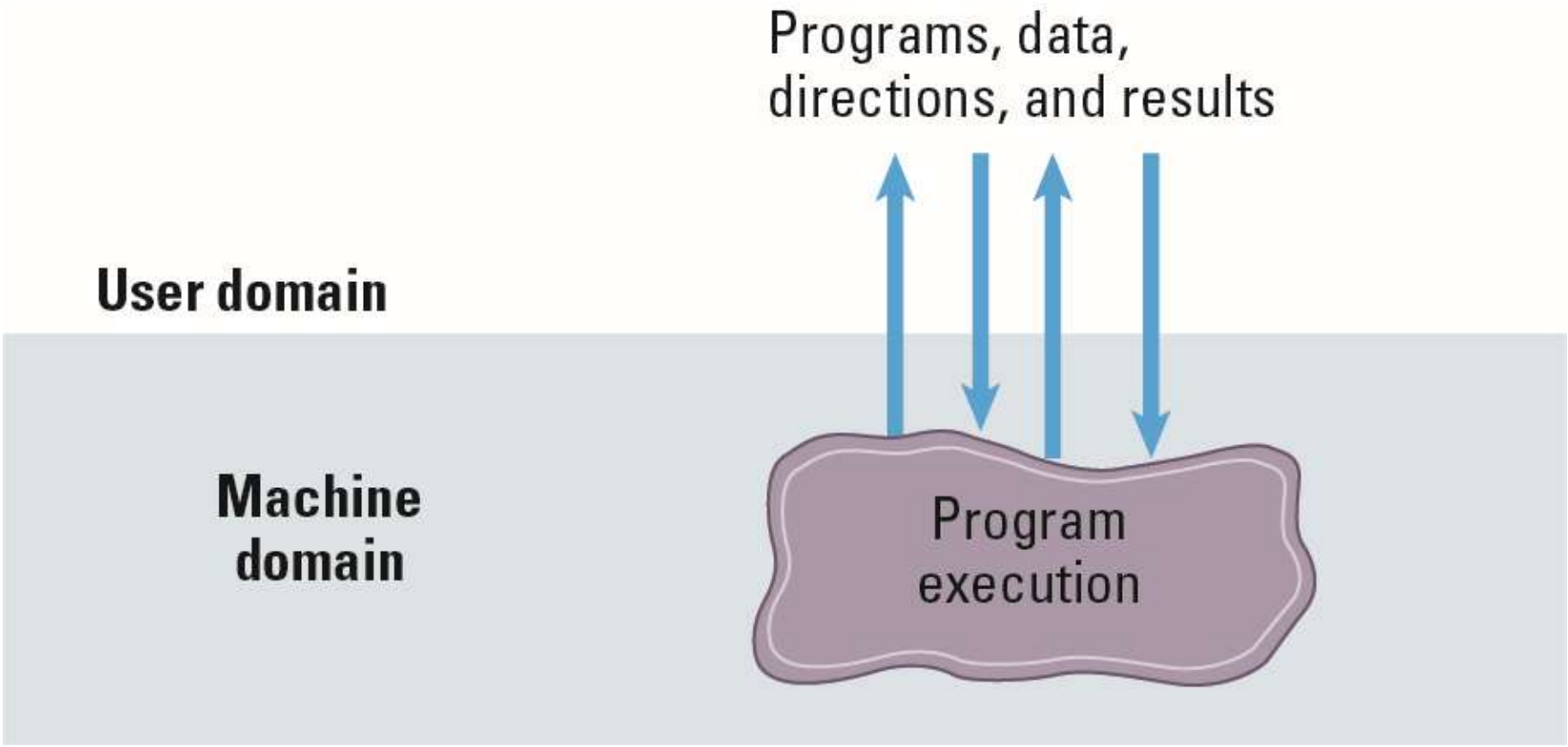
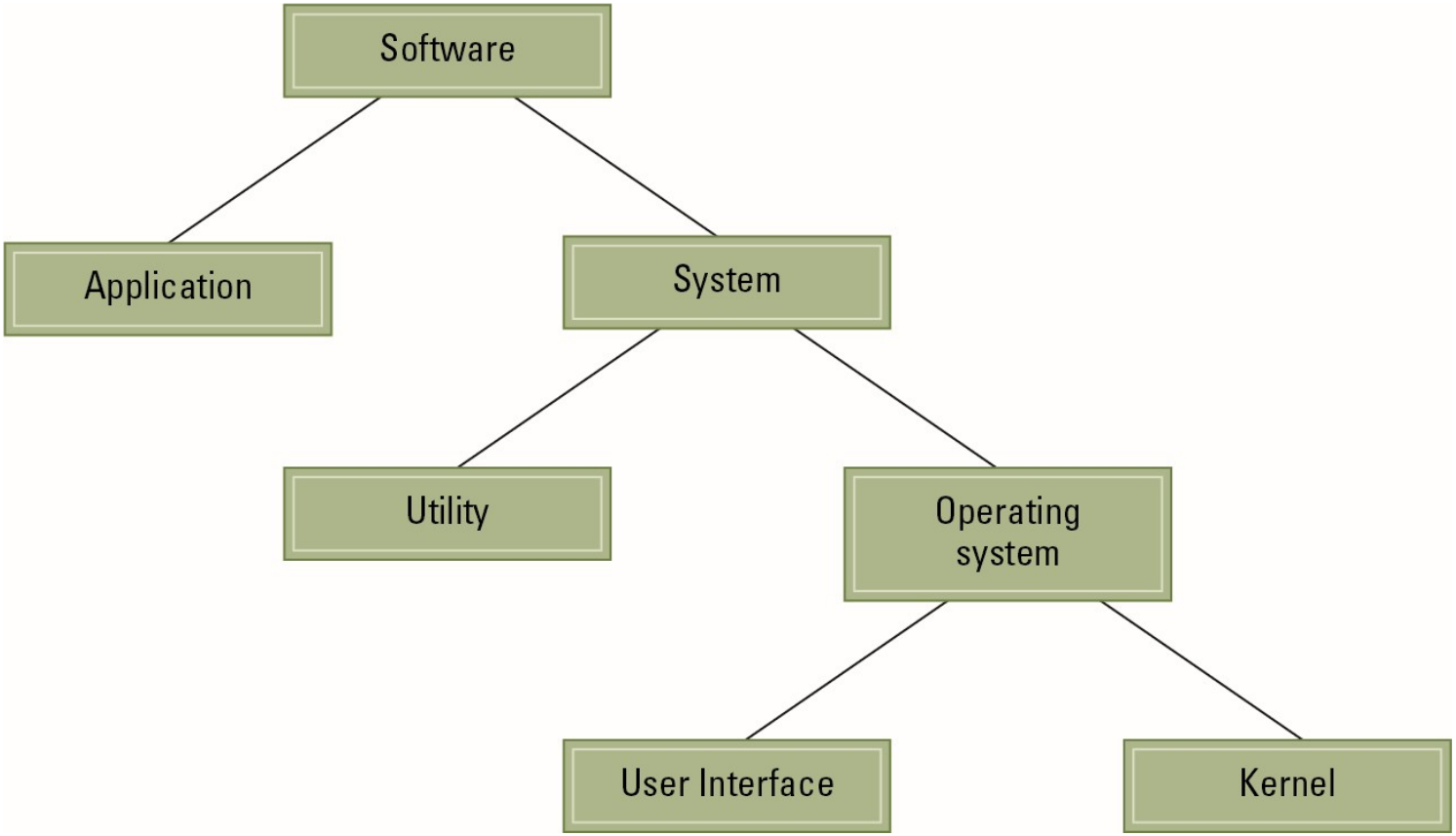


Figure 3.3 Software classification



Operating System Components

- **User Interface:** Communicates with users
  - Text based (Shell)
  - Graphical user interface (GUI)
- **Kernel:** Performs basic required functions
  - File manager
  - Device drivers
  - Memory manager
  - Scheduler and dispatcher

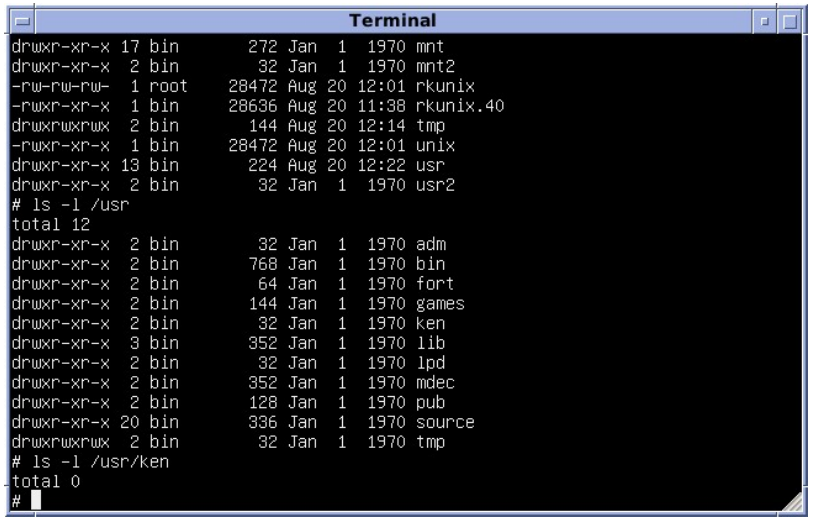
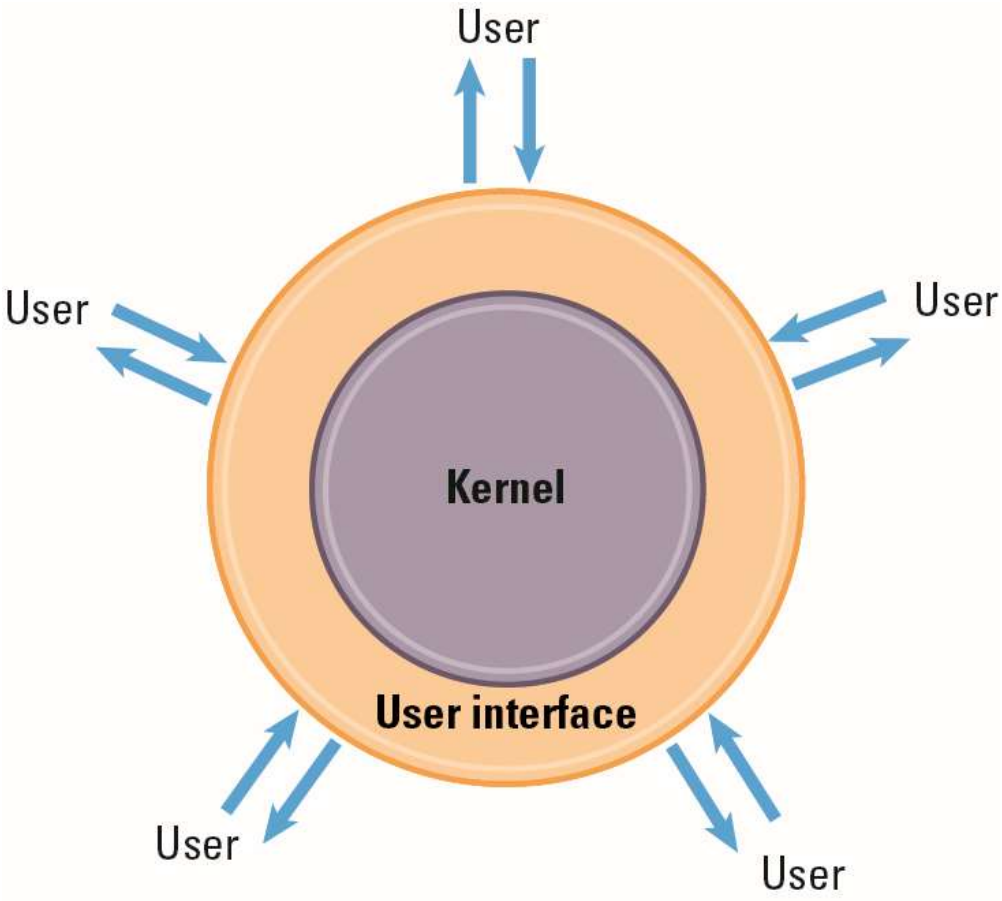


Figure 3.4 The user interface acts as an intermediary between users and the operating system's kernel



File Manager

- **Directory** (or **Folder**): A user-created bundle of files and other directories (subdirectories)
- **Directory Path:** A sequence of directories within directories

animals/prehistoric/dinosaurs

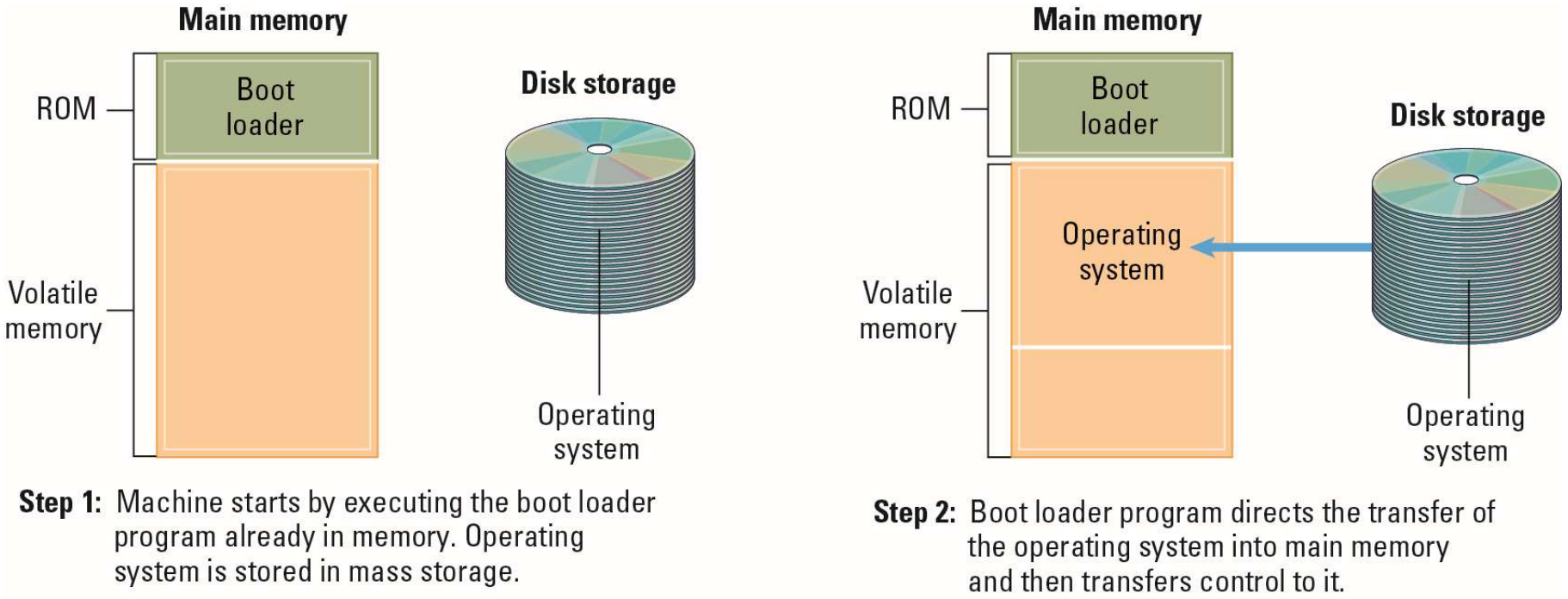
Memory Manager

- Allocates space in main memory
- May create the illusion that the machine has more memory than it actually does (**virtual memory**) by playing a “shell game” in which blocks of data (**pages**) are shifted back and forth between main memory and mass storage

Getting it Started (Bootstrapping)

- **Boot loader:** Program in ROM
  - Run by the CPU when power is turned on
  - Transfers operating system from mass storage to main memory
  - Executes jump to operating system
- Firmware routines can be used by the boot loader to perform I/O activities before the operating system becomes functional. For example, they are used to communicate with the computer user before the boot process actually begins and to report errors during booting. Widely used firmware systems include the BIOS (Basic Input/Output System)

Figure 3.5 The booting process



The instructions in the boot loader direct the CPU to transfer the operating system from a predetermined location into the volatile area of main memory

3.3 Coordinating the Machine's Activities

An operating system coordinates the execution of application software, utility software, and units within the operating system itself.

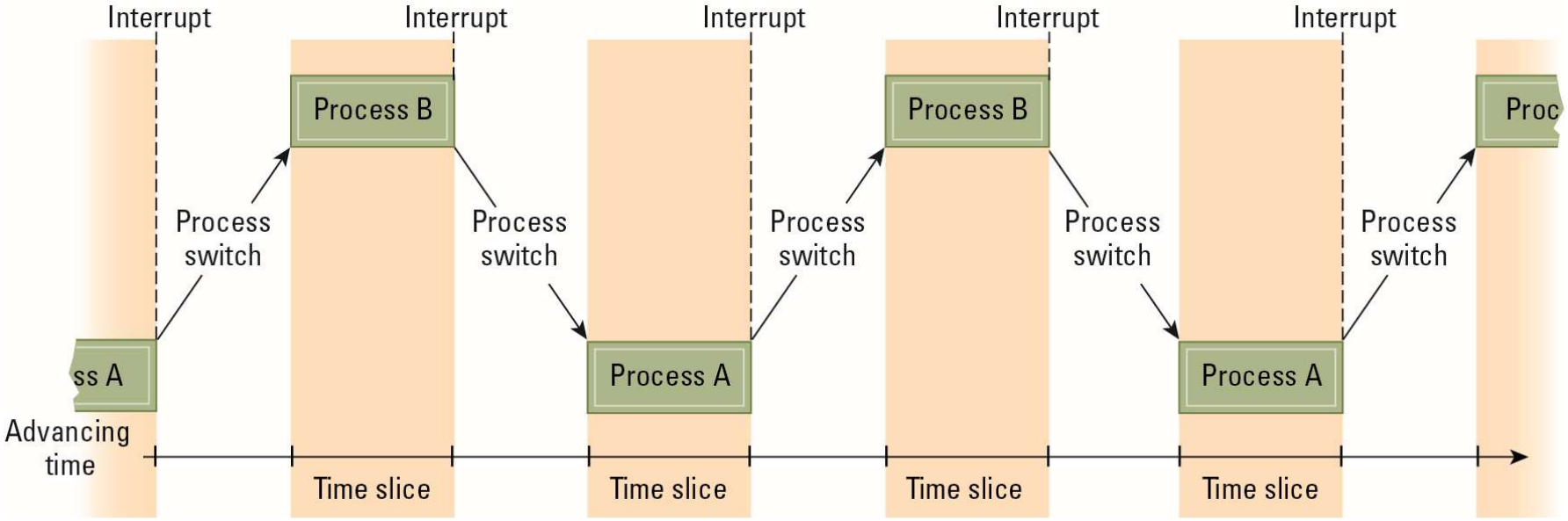
The Concept of a Process

- **Process:** The activity of executing a program
- **Process State:** Current status of the activity
  - Program counter
  - General purpose registers
  - Related portion of main memory

Process Administration

- To keep track of all the processes, the scheduler maintains a block of information in main memory called the **process table.**
- **Scheduler:** Adds new processes to the process table and removes completed processes from the process table
- **Dispatcher:** Controls the allocation of time slices (typically measured in milliseconds or microseconds) to the processes in the process table
  - The end of a time slice is signaled by an interrupt.
  - The procedure of changing from one process to another is called a process switch

Figure 3.6 Multiprogramming between process A and process B



3.4 Handling Competition Among Processes

- Consider a time-sharing/multitasking operating system controlling the activities of a computer with a single printer. If a process needs to print its results, it must request that the operating system give it access to the printer's device driver
- At this point, the operating system must decide whether the printer is already being used by another process.
- If it is not, the operating system should grant the request and allow the process to continue
- After all, if two processes were given simultaneous access to the computer's printer, the results would be worthless to both.

3.4 Handling Competition Among Processes

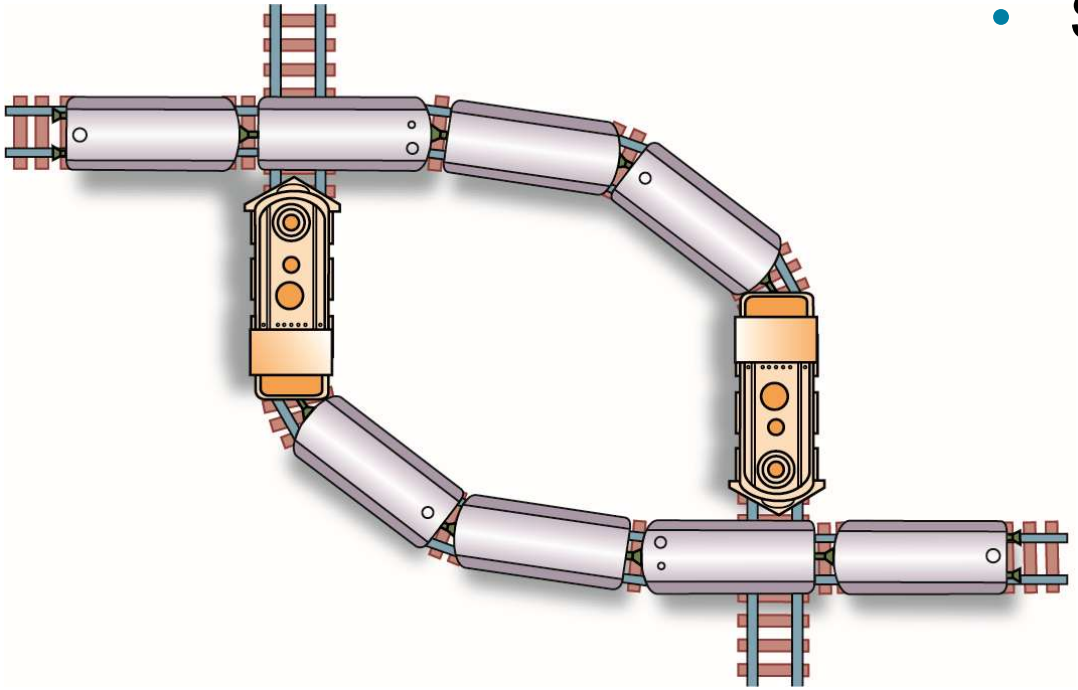
- To control access to the printer, the operating system must keep track of whether the printer has been allocated.
- One approach to this task would be to use a flag, which in this context refers to a bit in memory whose states are often referred to as set and clear
- However, this simple flag system has a problem. The task of testing and possibly setting the flag may require several machine instructions.
- Solution:
  - **interrupt disable and interrupt enable**
  - **test-and-set instruction**

3.4 Handling Competition Among Processes

- **Semaphore:** CPU retrieves the value of a flag, note the value received, and then set the flag—all within a single machine instruction. A properly implemented flag, as just described, is called a semaphore
- **Critical Region:** A group of instructions that should be executed by only one process at a time
- **Mutual exclusion:** Requirement that only one process at a time be allowed to execute a Critical Region
- In summary, a common way of obtaining mutual exclusion to a critical region is to guard the critical region with a semaphore.

Deadlock

- Processes block each other from continuing because each is waiting for a resource that is allocated to another



- Solutions:
  - Require processes to request **all resources at once**
  - Terminate one or more processes
  - Manual recovery …