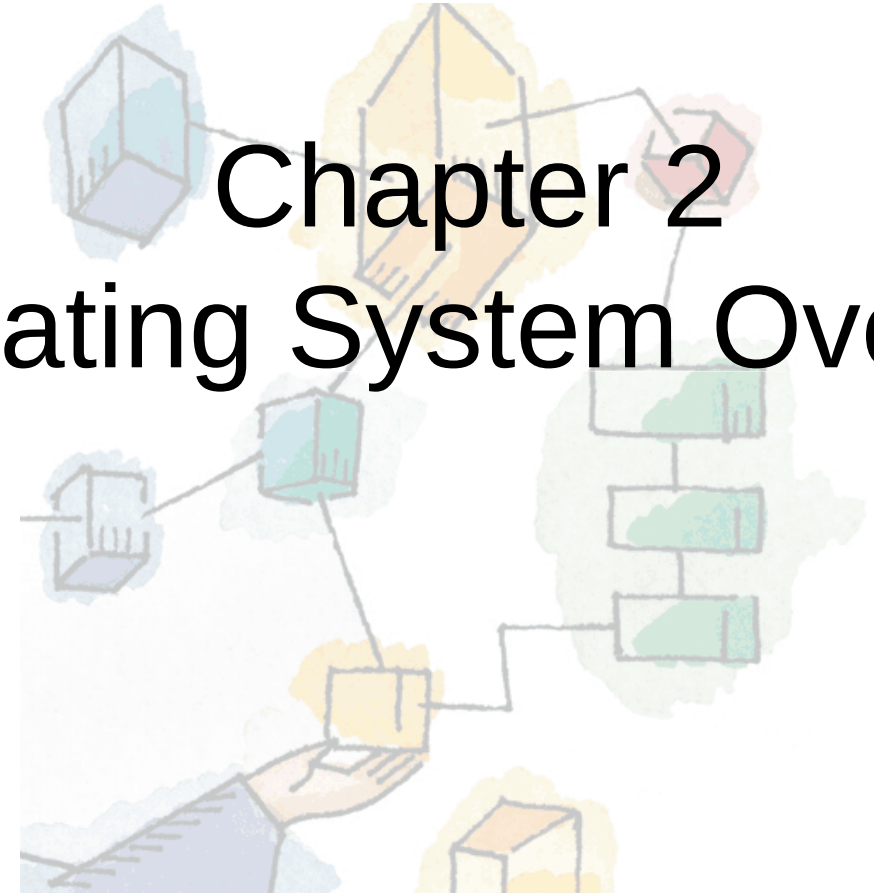
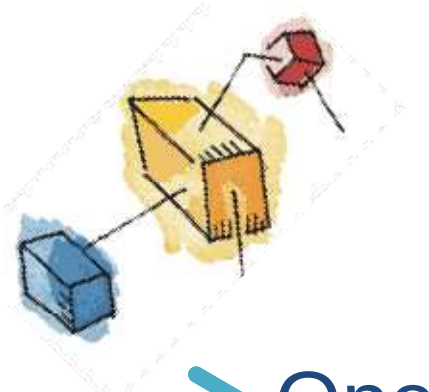
Operating Systems:
Internals and Design Principles, 6/E
William Stallings



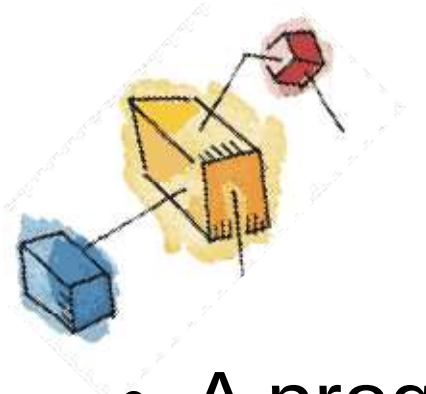
Chapter 2
Operating System Overview

Dave Bremer
Otago Polytechnic, N.Z.



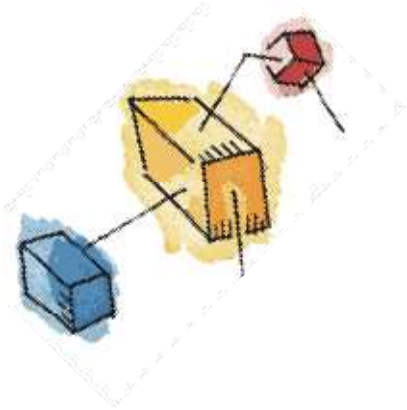
Roadmap

- Operating System Objectives/Functions
  - The Evolution of Operating Systems
  - Major Achievements
  - Developments Leading to Modern Operating Systems
  - Microsoft Windows Overview
  - UNIX Systems
  - Linux



Operating System

- A program that controls the execution of application programs
- An interface between applications and hardware
- Main objectives of an OS:
  - Convenience
  - Efficiency
  - Ability to evolve



Layers and Views

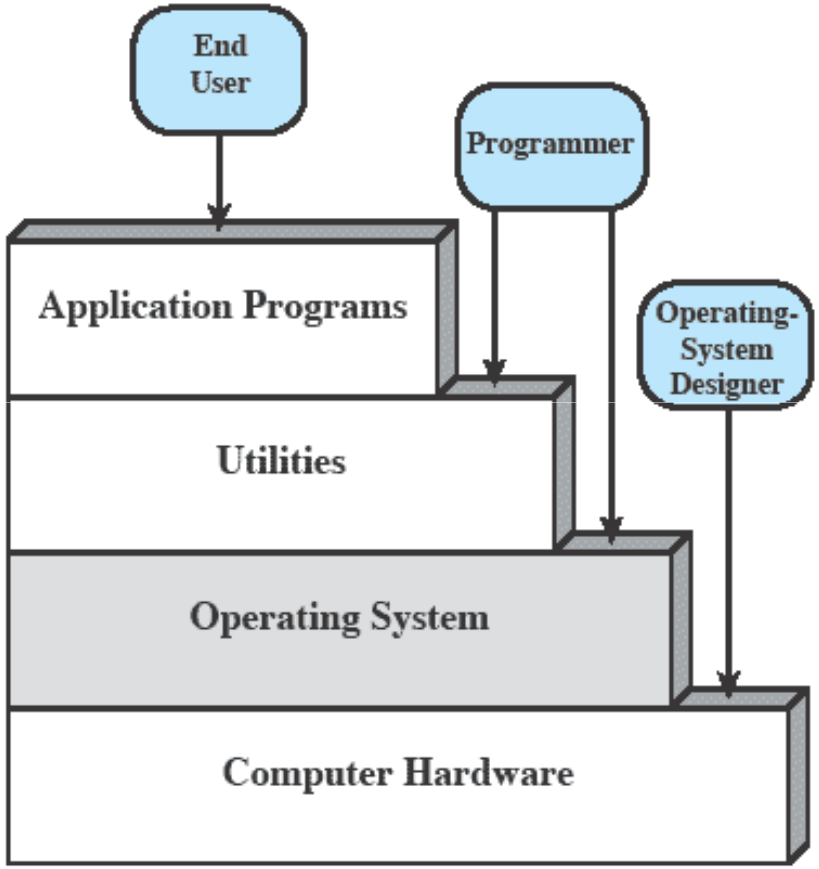
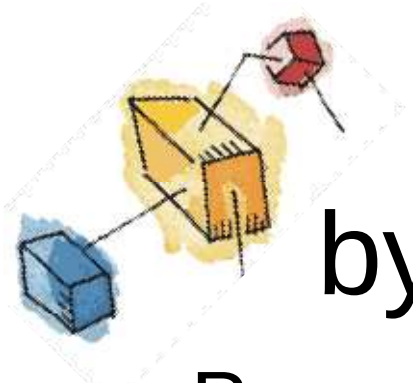
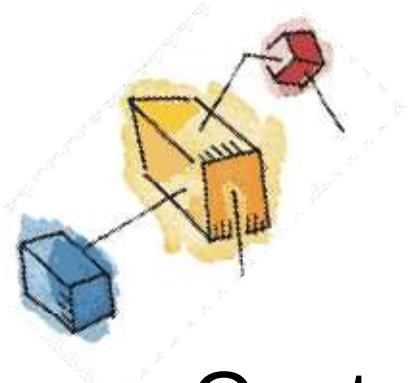


Figure 2.1 Layers and Views of a Computer System



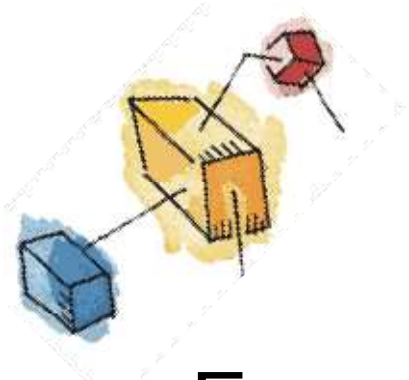
Services Provided by the Operating System

- Program development
  - Editors and debuggers.
- Program execution
  - OS handles scheduling of numerous tasks required to execute a program
- Access I/O devices
  - Each device will have unique interface
  - OS presents standard interface to users



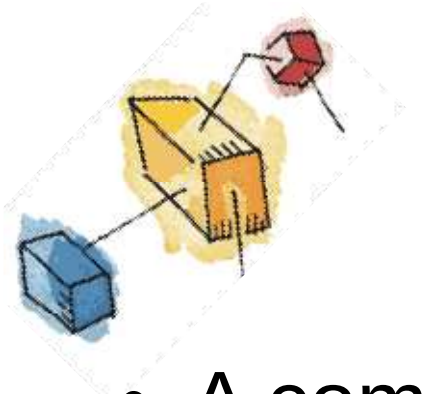
Services cont…

- Controlled access to files
  - Accessing different media but presenting a common interface to users
  - Provides protection in multi-access systems
- System access
  - Controls access to the system and its resources



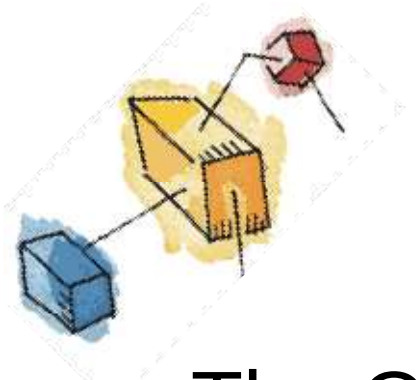
Services cont…

- Error detection and response
  - Internal and external hardware errors
  - Software errors
  - Operating system cannot grant request of application
- Accounting
  - Collect usage statistics
  - Monitor performance



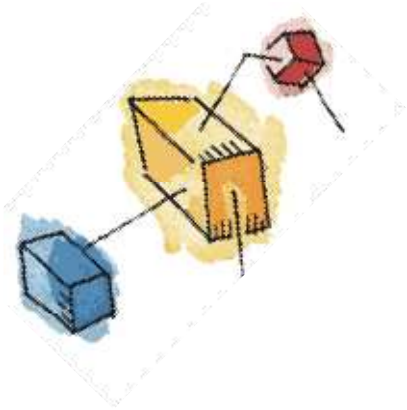
The Role of an OS

- A computer is a set of resources for the movement, storage, and processing of data.
- The OS is responsible for managing these resources.



Operating System as Software

- The OS functions in the same way as an ordinary computer software
  - It is a program that is executed by the CPU
- Operating system relinquishes control of the processor



OS as Resource Manager

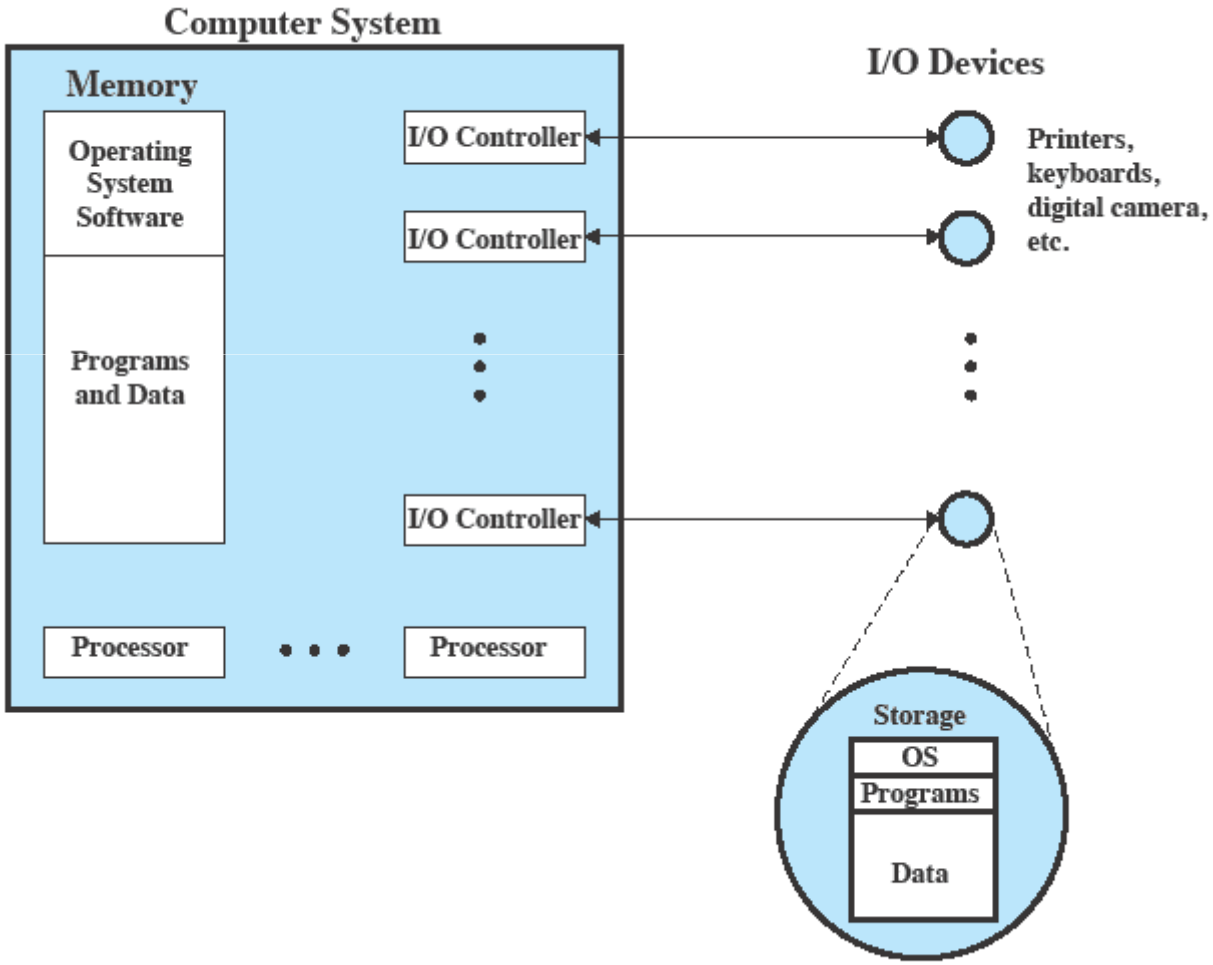
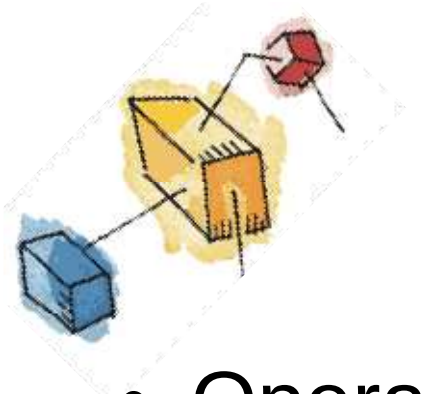
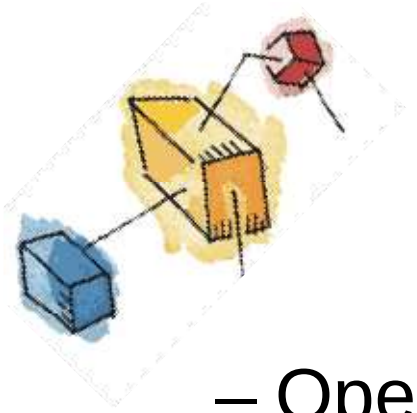


Figure 2.2 The Operating System as Resource Manager



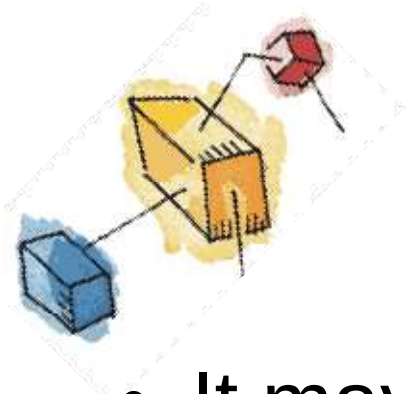
Evolution of Operating Systems

- Operating systems will evolve over time
  - Hardware upgrades plus new types of hardware
  - New services
  - Fixes



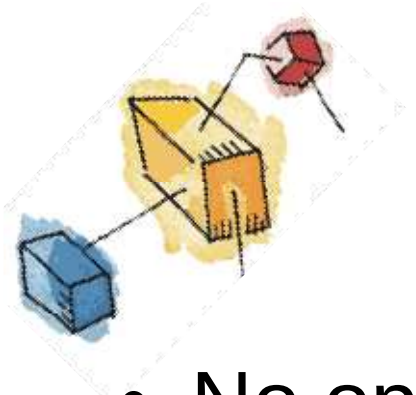
Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems
- Linux



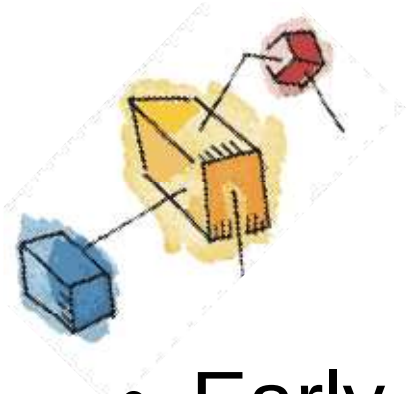
Evolution of Operating Systems

- It may be easier to understand the key requirements of an OS by considering the evolution of Operating Systems
- Stages include
  - Serial Processing
  - Simple Batch Systems
  - Multiprogrammed batch systems
  - Time Sharing Systems



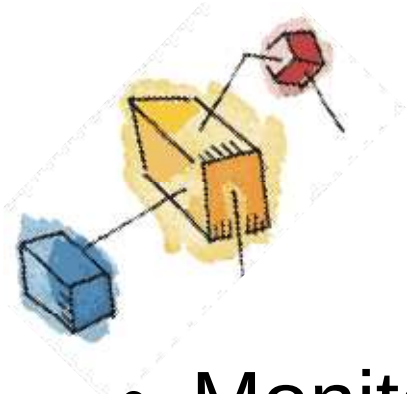
Serial Processing

- No operating system
- Machines run from a console with display lights, toggle switches, input device, and printer
- Problems include:
  - Scheduling
  - Setup time



Simple batch system

- Early computers were extremely expensive
  - Important to maximize processor utilization
- Monitor
  - Software that controls the sequence of events
  - Batch jobs together
  - Program returns control to monitor when finished



Monitor's perspective

- Monitor controls the sequence of events
- *Resident Monitor* is software always in memory
- Monitor reads in job and gives control
- Job returns control to monitor

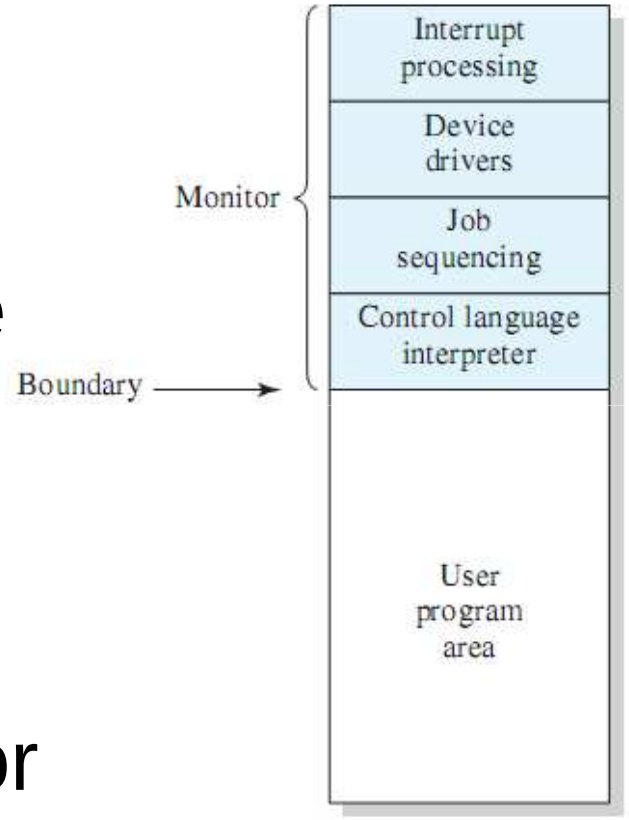
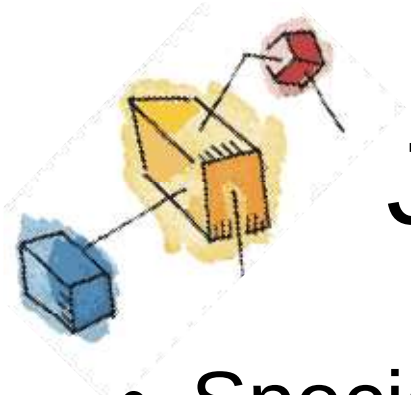
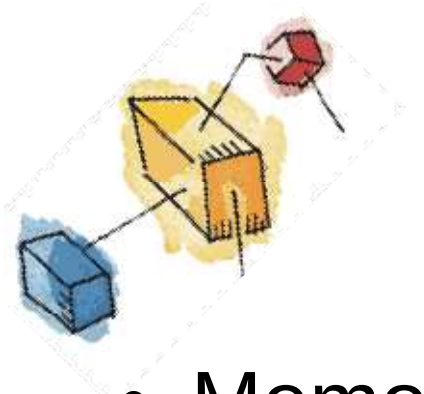


Figure 2.3 **Memory Layout for a Resident Monitor**



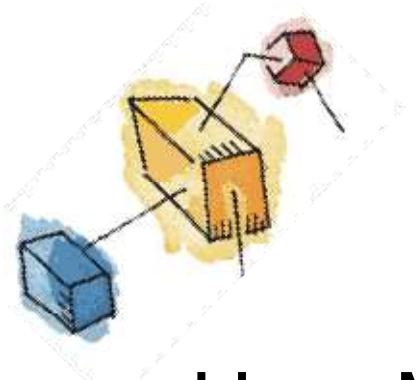
Job Control Language

- Special type of programming language to control jobs
- Provides instruction to the monitor
  - What compiler to use
  - What data to use



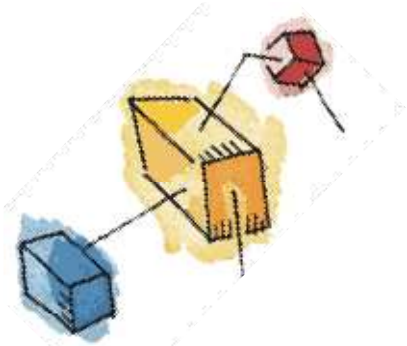
Desirable Hardware Features

- Memory protection for monitor
  - Jobs cannot overwrite or alter
- Timer
  - Prevent a job from monopolizing system
- Privileged instructions
  - Only executed by the monitor
- Interrupts



Modes of Operation

- User Mode
  - User program executes in user mode
  - Certain areas of memory protected from user access
  - Certain instructions may not be executed
- Kernel Mode
  - Monitor executes in kernel mode
  - Privileged instructions may be executed, all memory accessible.



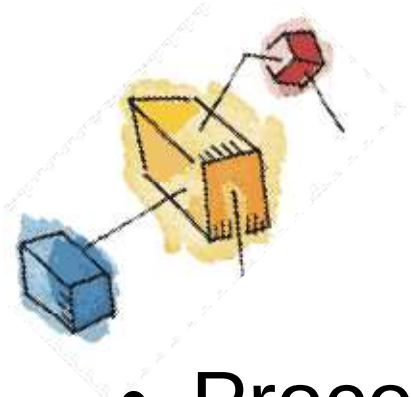
Multiprogrammed Batch Systems

- CPU is often idle
  - Even with automatic job sequencing.
  - I/O devices are slow compared to processor

Read one record from file	15 μs
Execute 100 instructions	1 μs
Write one record to file	15 μs
TOTAL	31 μs

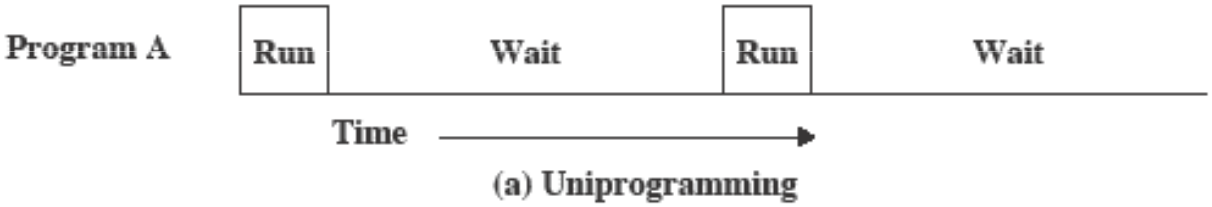
Percent CPU Utilization $= \frac{1}{31} = 0.032 = 3.2\%$

Figure 2.4 System Utilization Example



Uniprogramming

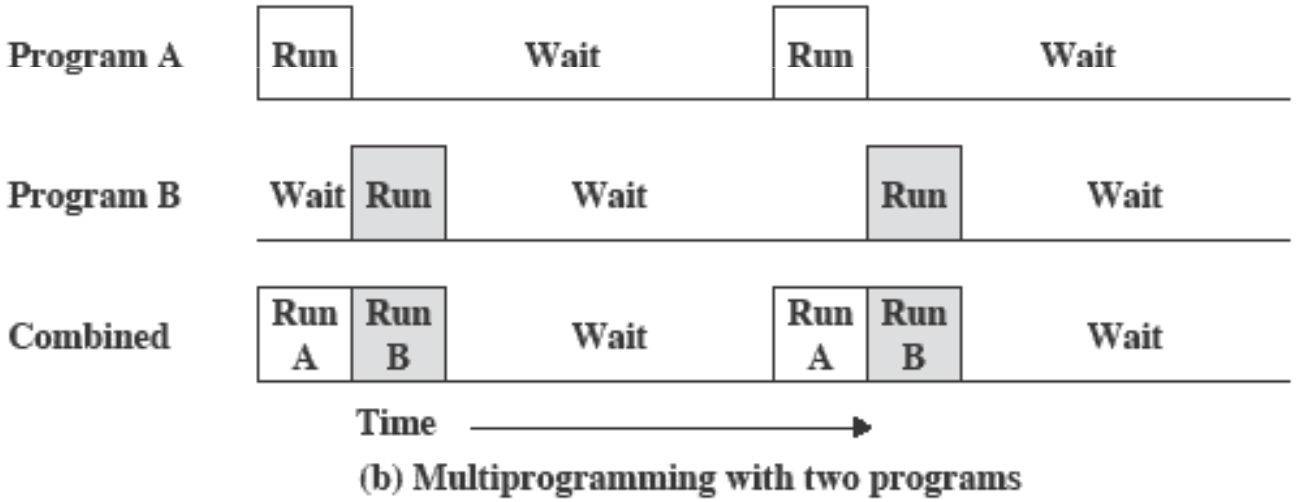
- Processor must wait for I/O instruction to complete before preceding



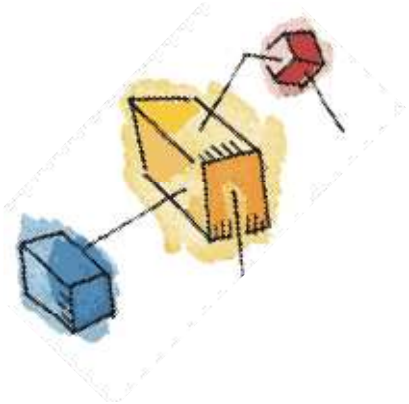
(a) Uniprogramming

Multiprogramming

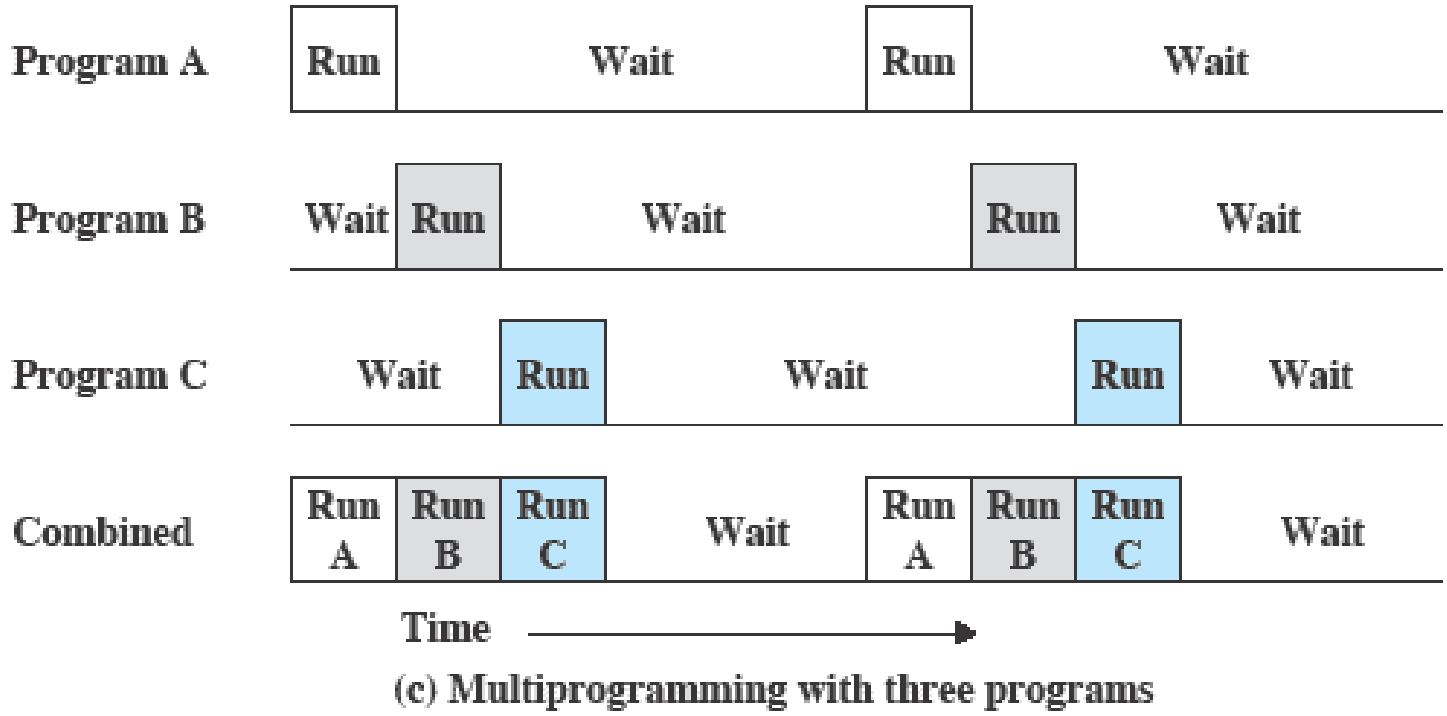
- When one job needs to wait for I/O, the processor can switch to the other job



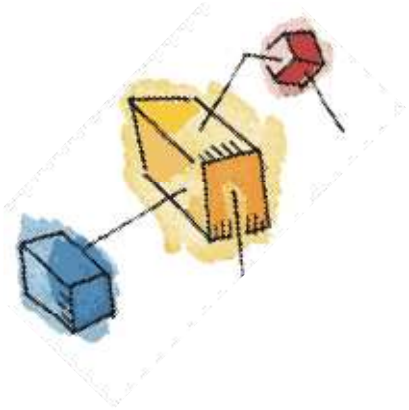
(b) Multiprogramming with two programs



Multiprogramming



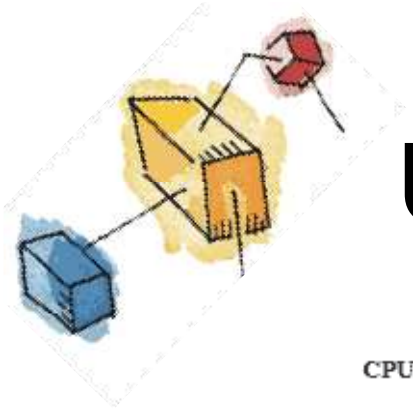
(c) Multiprogramming with three programs



Example

Table 2.1 Sample Program Execution Attributes

	JOB1	JOB2	JOB3
Type of job	Heavy compute	Heavy I/O	Heavy I/O
Duration	5 min	15 min	10 min
Memory required	50 M	100 M	75 M
Need disk?	No	No	Yes
Need terminal?	No	Yes	No
Need printer?	No	No	Yes



Utilization Histograms

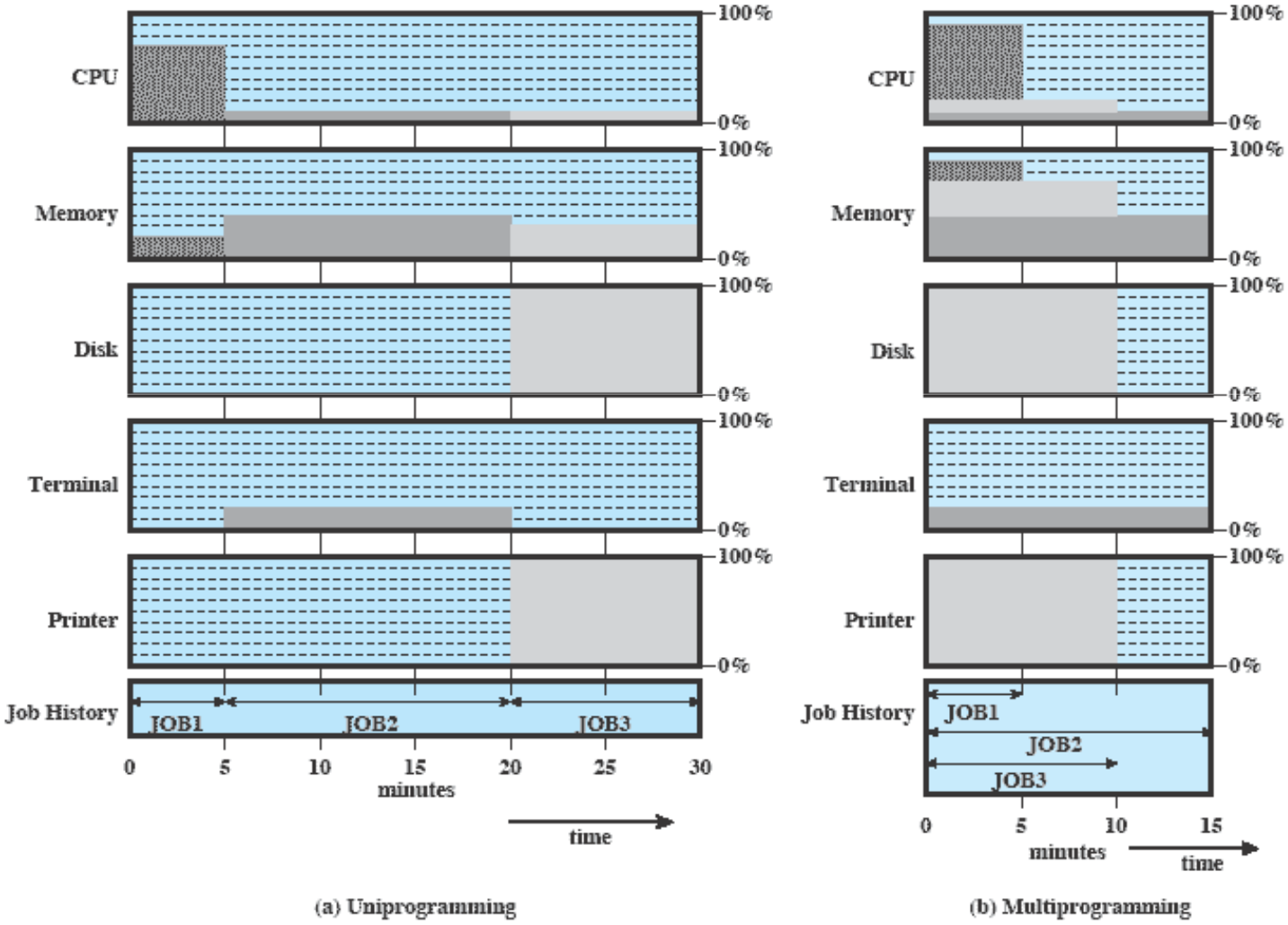
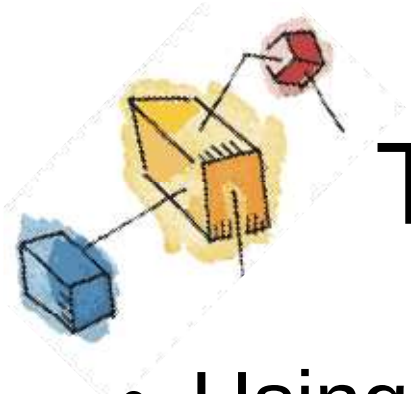
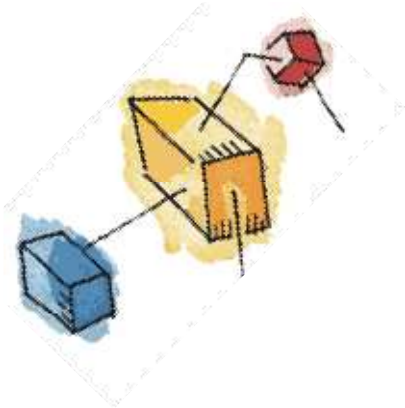


Figure 2.6 Utilization Histograms



Time Sharing Systems

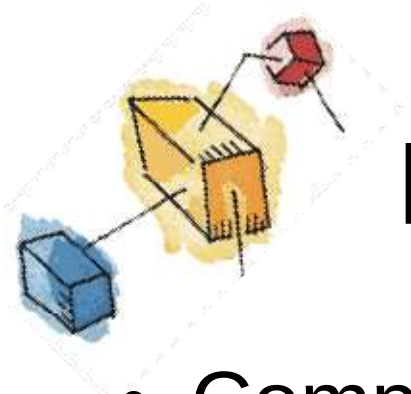
- Using multiprogramming to handle multiple interactive jobs
- Processor’s time is shared among multiple users
- Multiple users simultaneously access the system through terminals



Batch Multiprogramming vs. Time Sharing

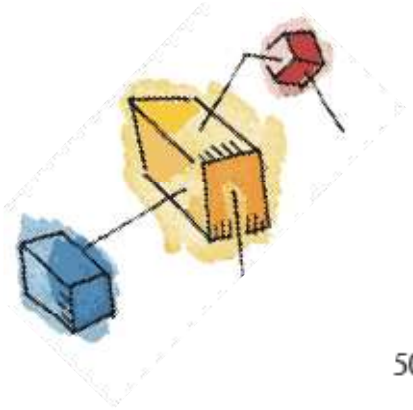
Table 2.3 Batch Multiprogramming versus Time Sharing

	Batch Multiprogramming	Time Sharing
Principal objective	Maximize processor use	Minimize response time
Source of directives to operating system	Job control language commands provided with the job	Commands entered at the terminal



Early Example: CTSS

- Compatible Time-Sharing System (CTSS)
  - Developed at MIT as project MAC
- Time Slicing:
  - When control was passed to a user
  - User program and data loaded
  - Clock generates interrupts about every 0.2 sec
  - At each interrupt OS gained control and could assign processor to another user



CTSS Operation

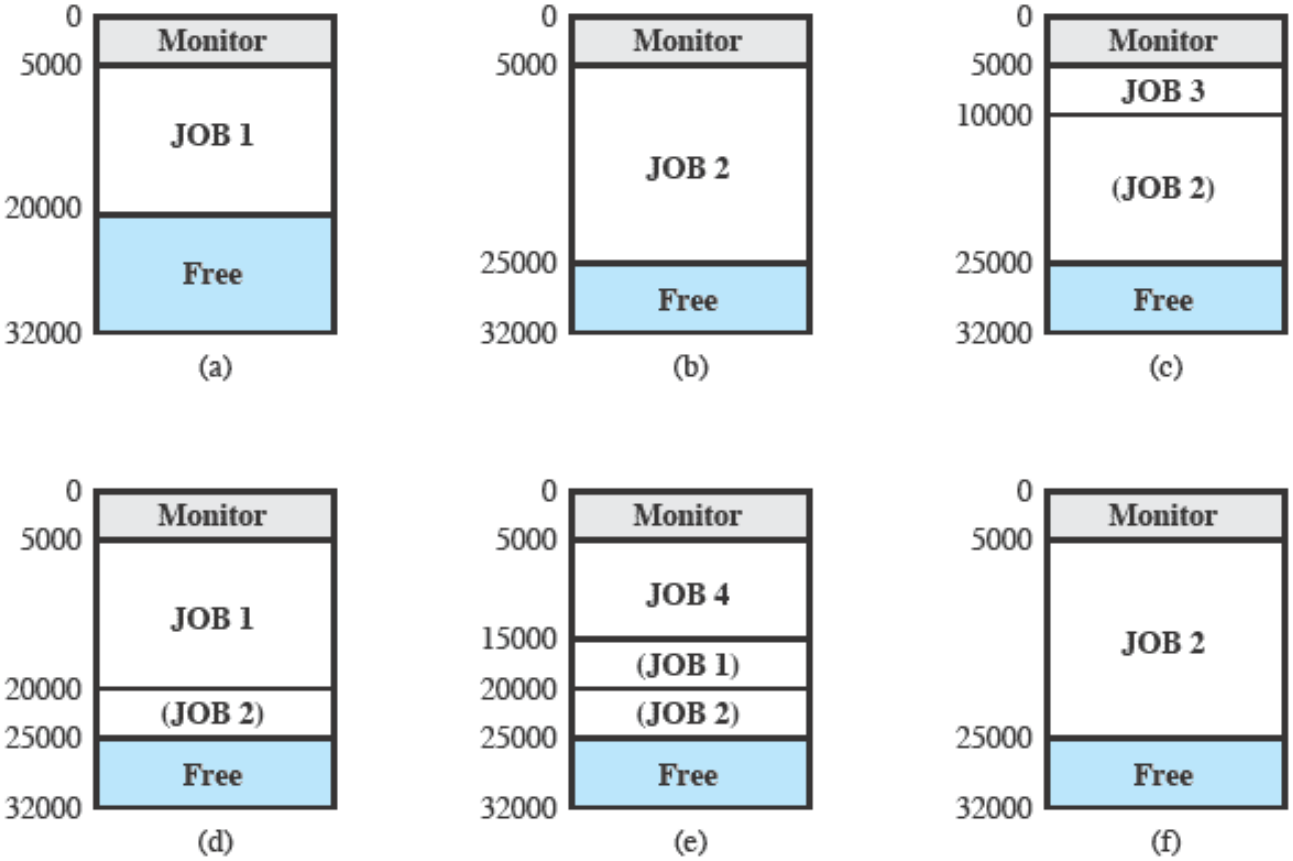
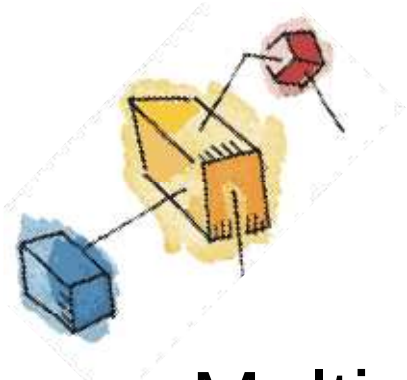
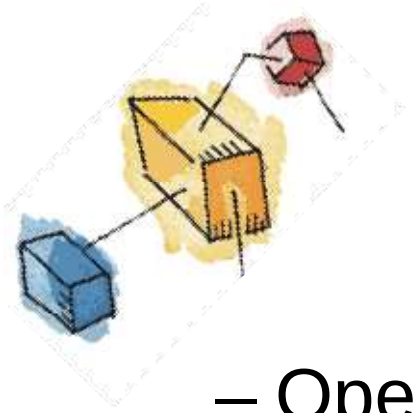


Figure 2.7 CTSS Operation



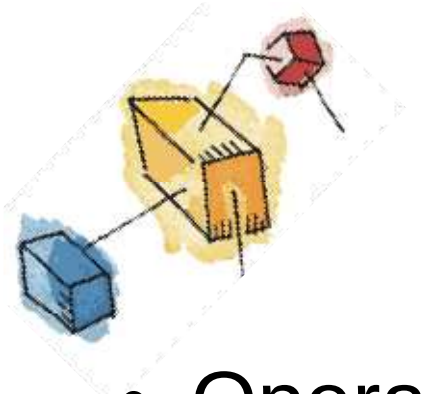
Problems and Issues

- Multiple jobs in memory must be protected from each other's data
- File system must be protected so that only authorised users can access
- Contention for resources must be handled
  - Printers, storage etc



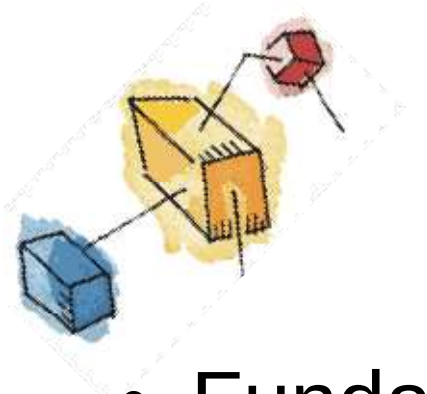
Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems
- Linux



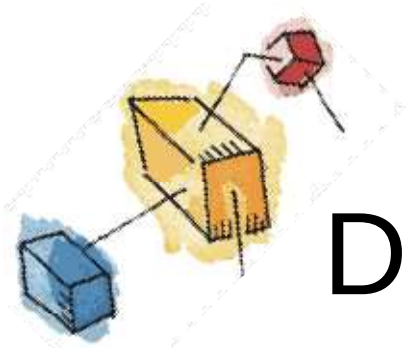
Major Advances

- Operating Systems are among the most complex pieces of software ever developed
- Major advances include:
  - Processes
  - Memory management
  - Information protection and security
  - Scheduling and resource management
  - System



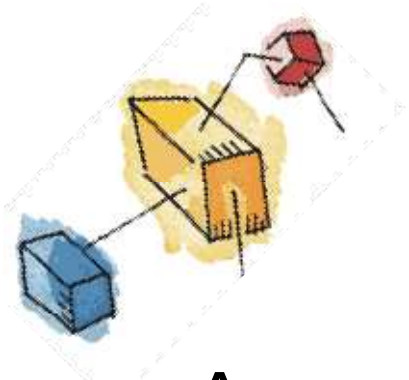
Process

- Fundamental to the structure of OS's
- A *process* is:
  - A program in execution
  - An instance of a running program
  - The entity that can be assigned to and executed on a processor
  - A single sequential thread of execution, a current state, and an associated set of system resources.



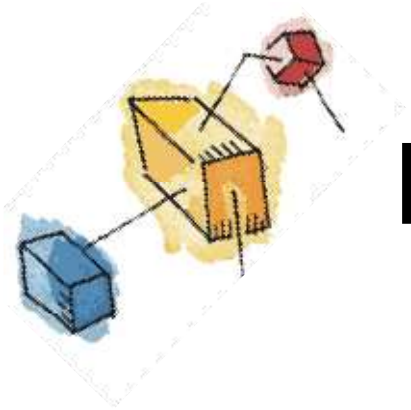
Causes of Errors when Designing System Software

- Error in designing an OS are often subtle and difficult to diagnose
- Errors typically include:
  - Improper synchronization
  - Failed mutual exclusion
  - Non-determinate program operation
  - Deadlocks



Components of a Process

- A process consists of
  - An executable program
  - Associated data needed by the program
  - Execution context of the program (or “process state”)
- The execution context contains all information the operating system needs to manage the process



Process Management

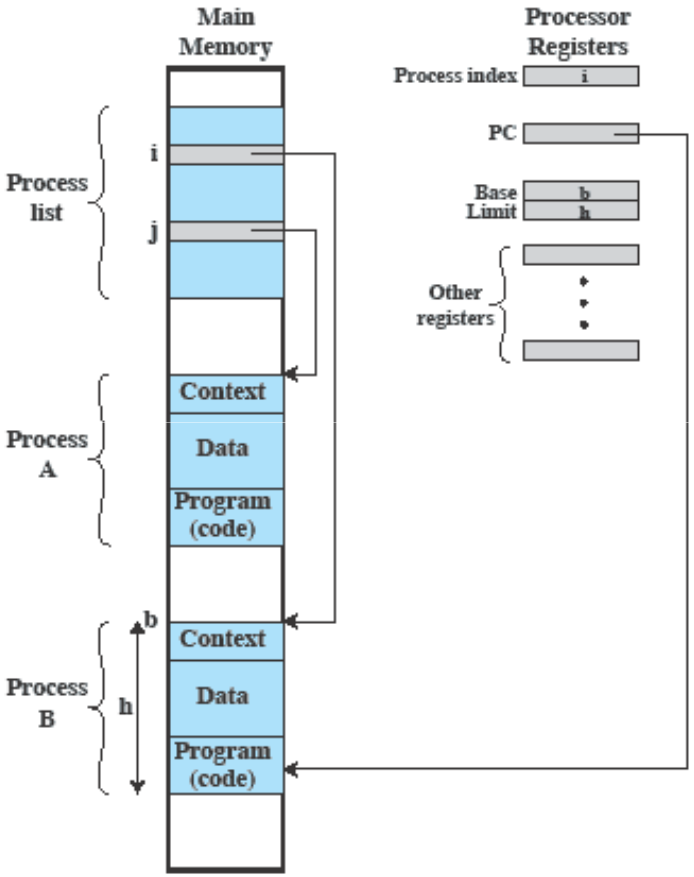
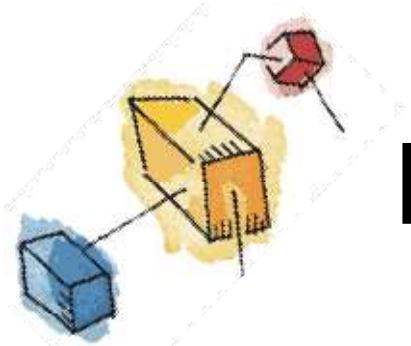
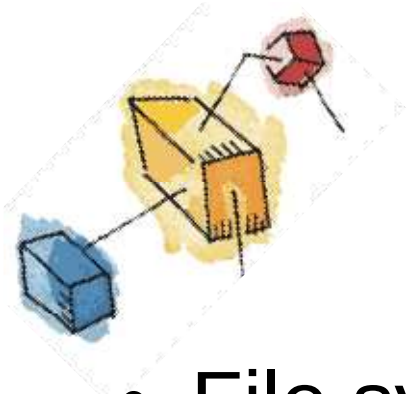


Figure 2.8 Typical Process Implementation



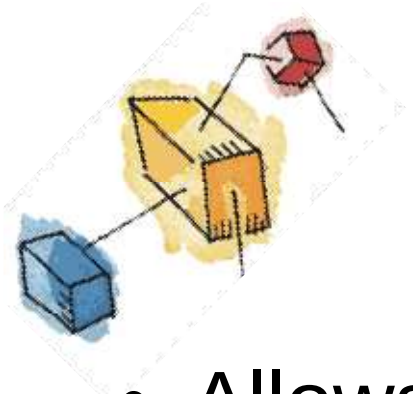
Memory Management

- The OS has 5 principal storage management responsibilities
  - Process isolation
  - Automatic allocation and management
  - Support of modular programming
  - Protection and access control
  - Long-term storage



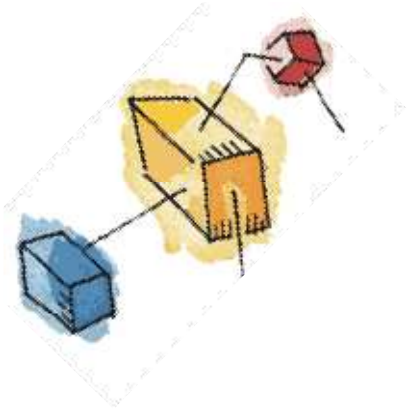
Virtual Memory

- File system implements long-term store
- Virtual memory allows programs to address memory from a logical point of view
  - Without regard to the limits of physical memory

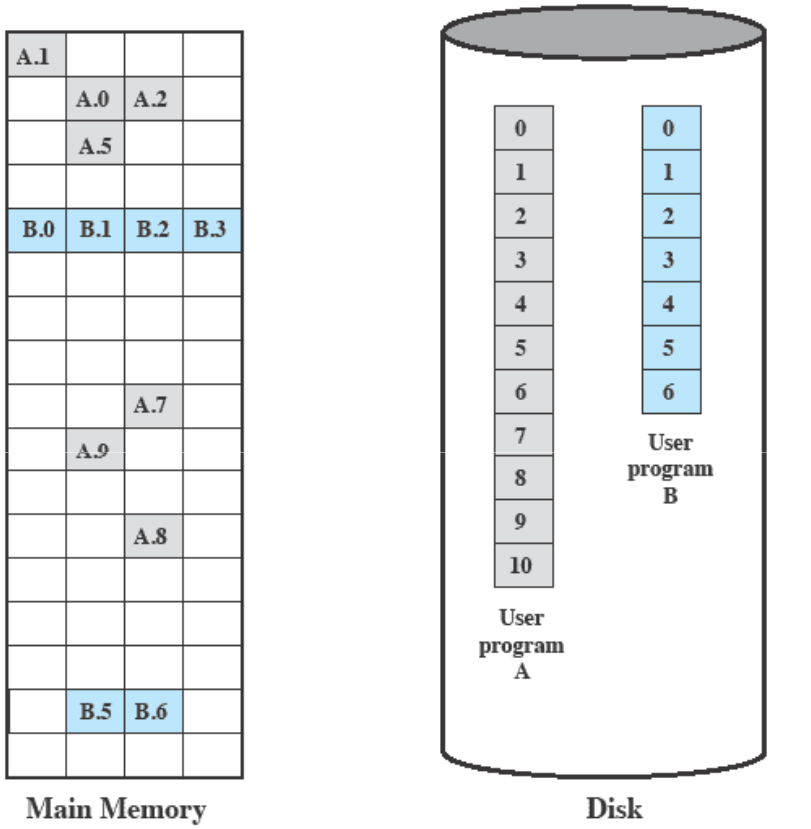


Paging

- Allows process to be comprised of a number of fixed-size blocks, called pages
- Virtual address is a page number and an offset within the page
- Each page may be located any where in main memory



Virtual Memory



Main memory consists of a number of fixed-length frames, each equal to the size of a page. For a program to execute, some or all of its pages must be in main memory.

Secondary memory (disk) can hold many fixed-length pages. A user program consists of some number of pages. Pages for all programs plus the operating system are on disk, as are files.

Figure 2.9 Virtual Memory Concepts

Virtual Memory Addressing

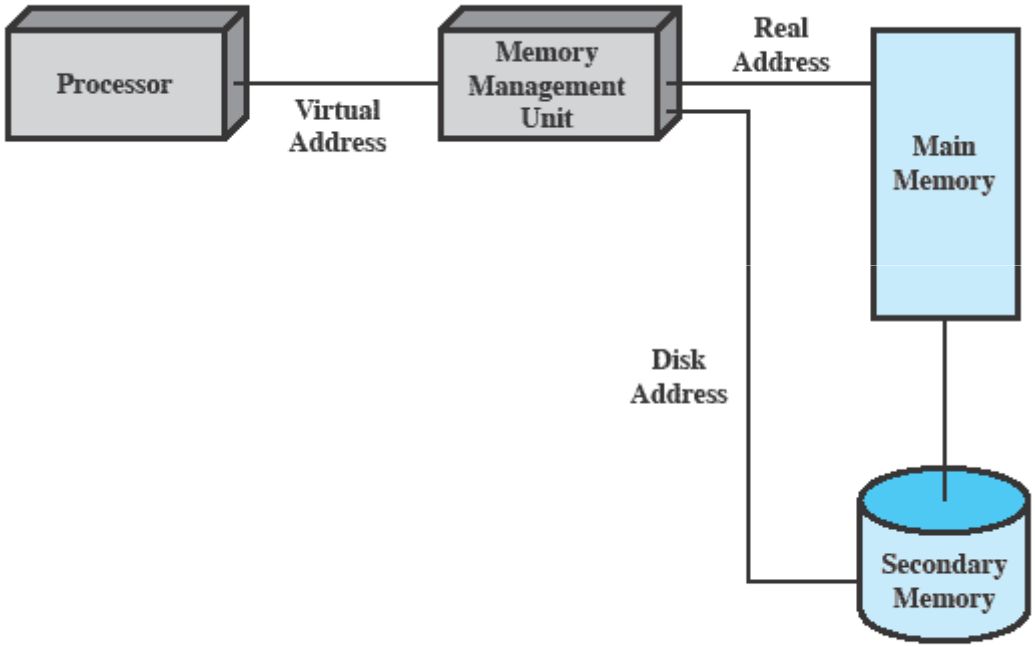
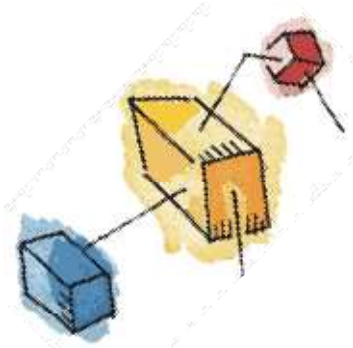
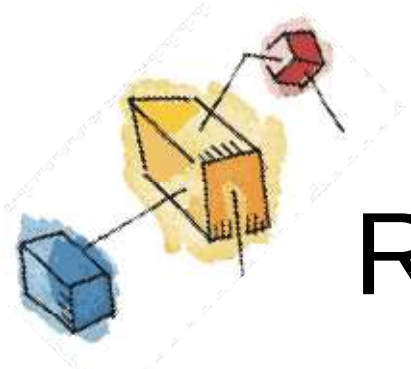


Figure 2.10 Virtual Memory Addressing



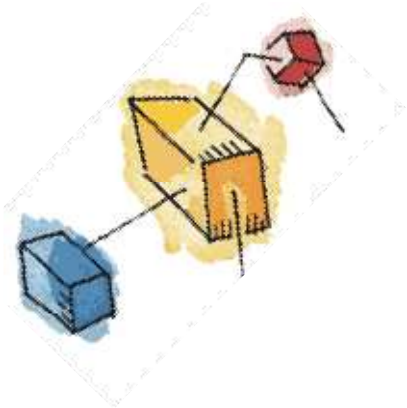
Information Protection and Security

- The problem involves controlling access to computer systems and the information stored in them.
- Main issues are:
  - Availability
  - Confidentiality
  - Data integrity
  - Authenticity



Scheduling and Resource Management

- Key responsibility of an OS is managing resources
- Resource allocation policies must consider:
  - Fairness
  - Differential responsiveness
  - Efficiency



Key Elements of an Operating System

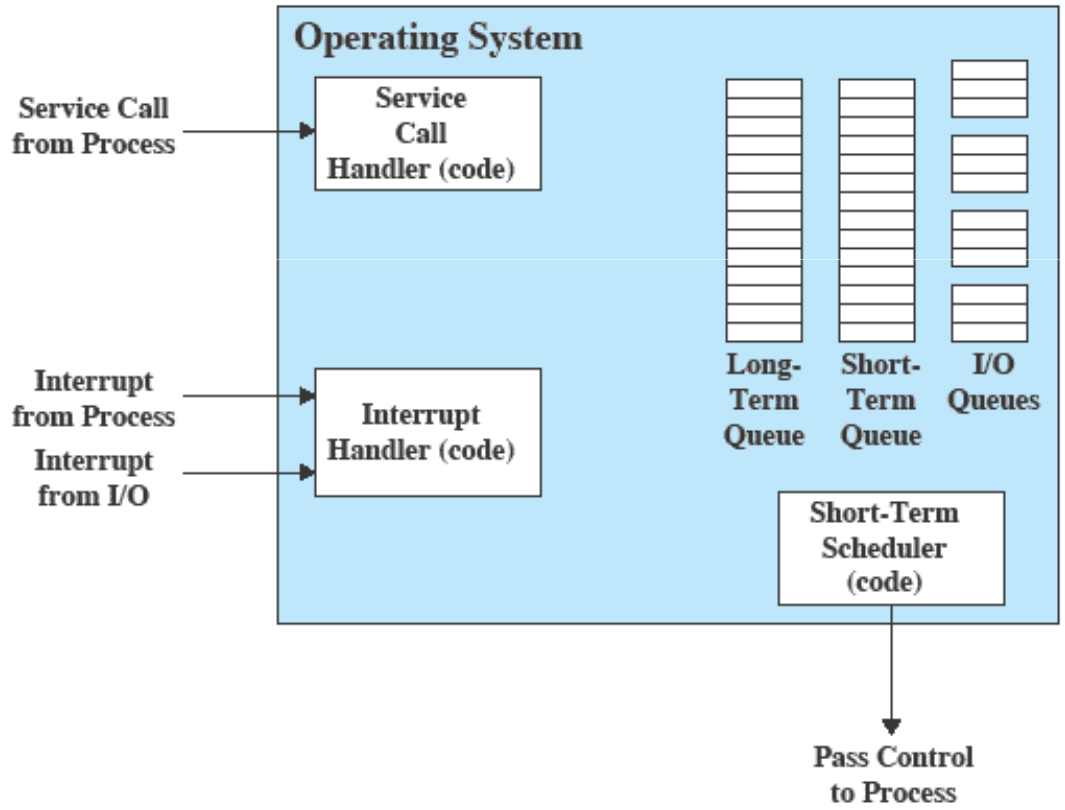
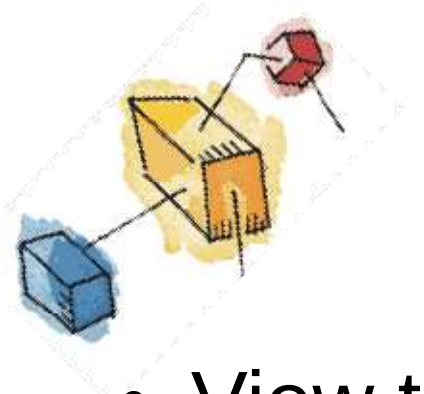
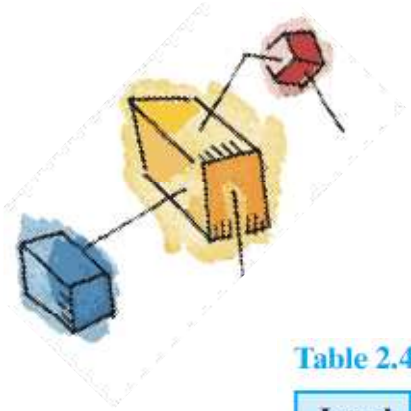


Figure 2.11 Key Elements of an Operating System for Multiprogramming



System Structure

- View the system as a series of levels
- Each level performs a related subset of functions
- Each level relies on the next lower level to perform more primitive functions
- This decomposes a problem into a number of more manageable subproblems

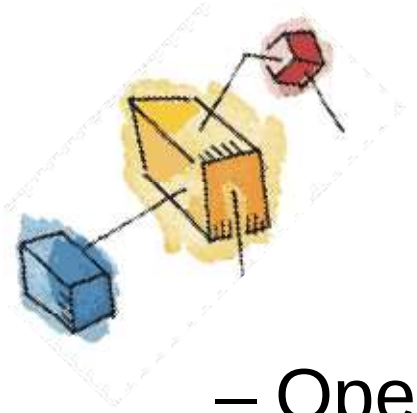


OS Design Hierarchy

Table 2.4 Operating System Design Hierarchy

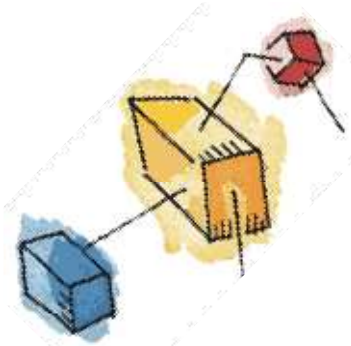
Level	Name	Objects	Example Operations
13	Shell	User programming environment	Statements in shell language
12	User processes	User processes	Quit, kill, suspend, resume
11	Directories	Directories	Create, destroy, attach, detach, search, list
10	Devices	External devices, such as printers, displays, and keyboards	Open, close, read, write
9	File system	Files	Create, destroy, open, close, read, write
8	Communications	Pipes	Create, destroy, open, close, read, write
7	Virtual memory	Segments, pages	Read, write, fetch
6	Local secondary store	Blocks of data, device channels	Read, write, allocate, free
5	Primitive processes	Primitive processes, semaphores, ready list	Suspend, resume, wait, signal
4	Interrupts	Interrupt-handling programs	Invoke, mask, unmask, retry
3	Procedures	Procedures, call stack, display	Mark stack, call, return
2	Instruction set	Evaluation stack, microprogram interpreter, scalar and array data	Load, store, add, subtract, branch
1	Electronic circuits	Registers, gates, buses, etc.	Clear, transfer, activate, complement

Gray shaded area represents hardware.



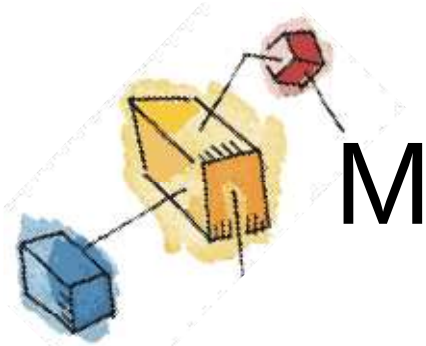
Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems
- Linux



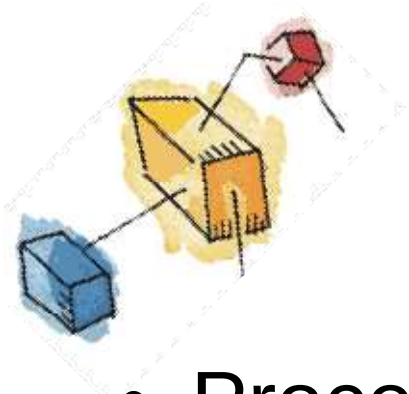
Different Architectural Approaches

- Various approaches have been tried, categories include:
  - Microkernel architecture
  - Multithreading
  - Symmetric multiprocessing
  - Distributed operating systems
  - Object-oriented design



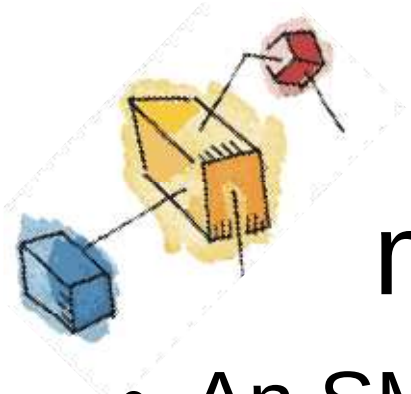
Microkernel Architecture

- Most early OS are a monolithic kernel
  - Most OS functionality resides in the kernel.
- A microkernel assigns only a few essential functions to the kernel
  - Address spaces
  - Interprocess communication (IPC)
  - Basic scheduling



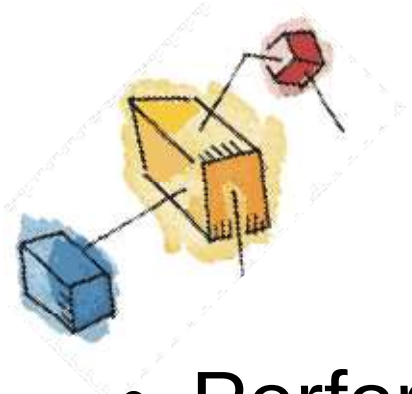
Multithreading

- Process is divided into threads that can run concurrently
- Thread
  - Dispatchable unit of work
  - executes sequentially and is interruptible
- Process is a collection of one or more threads



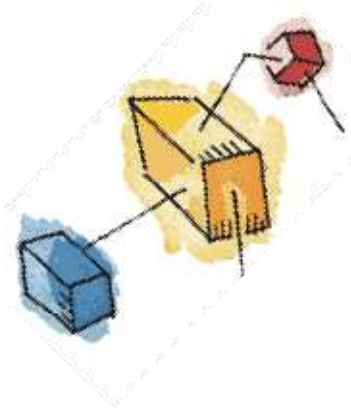
Symmetric multiprocessing (SMP)

- An SMP system has
  - multiple processors
  - These processors share same main memory and I/O facilities
  - All processors can perform the same functions
- The OS of an SMP schedules processes or threads across all of the processors.



SMP Advantages

- Performance
  - Allowing parallel processing
- Availability
  - Failure of a single process does not halt the system
- Incremental Growth
  - Additional processors can be added.
- Scaling



Multiprogramming and Multiprocessing

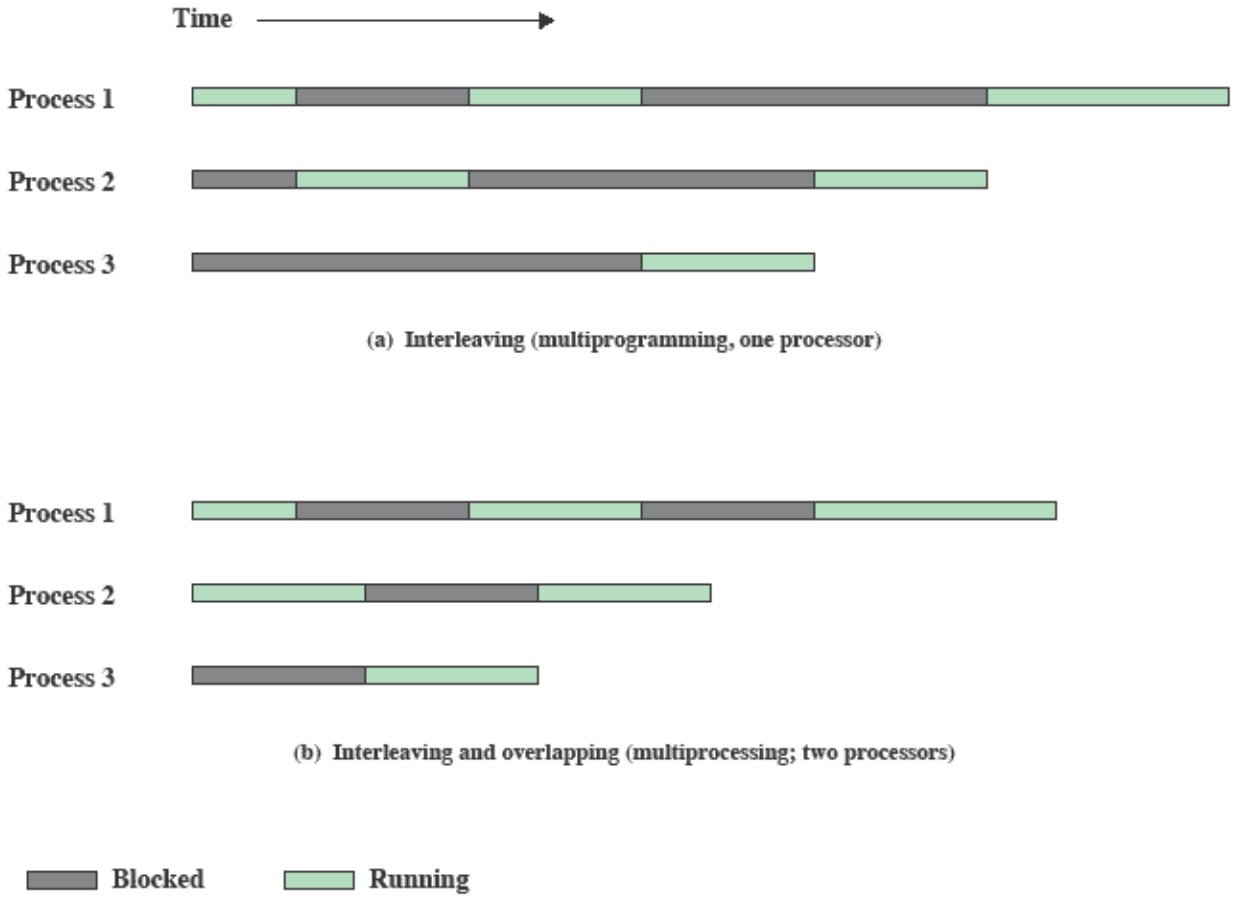
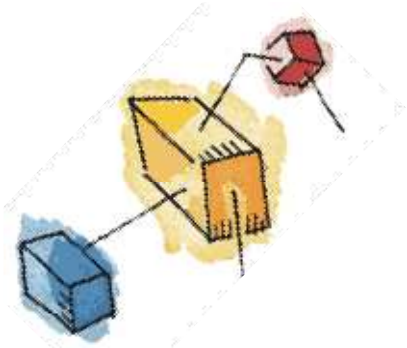
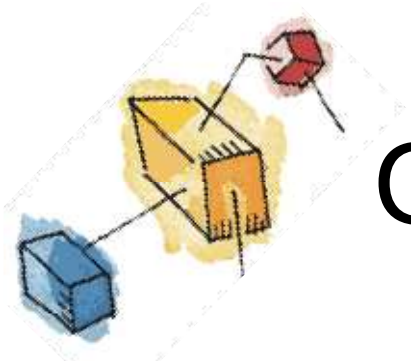


Figure 2.12 Multiprogramming and Multiprocessing



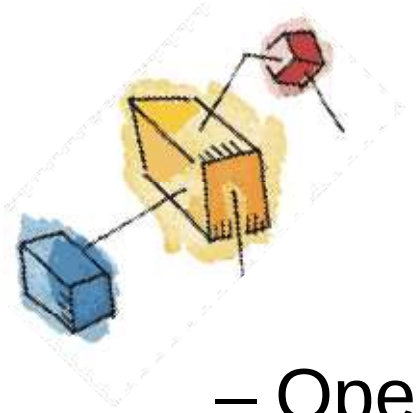
Distributed Operating Systems

- Provides the illusion of
  - a single main memory space and
  - single secondary memory space
- Early stage of development



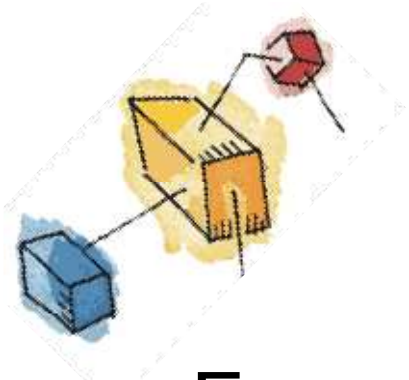
Object-oriented design

- Used for adding modular extensions to a small kernel
- Enables programmers to customize an operating system without disrupting system integrity



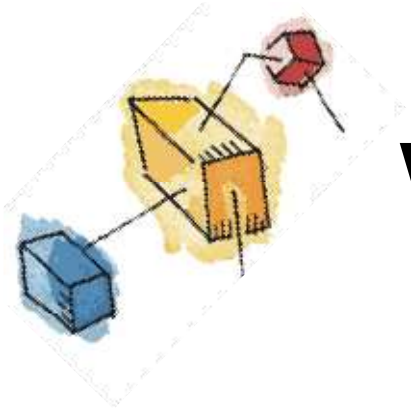
Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems
- Linux

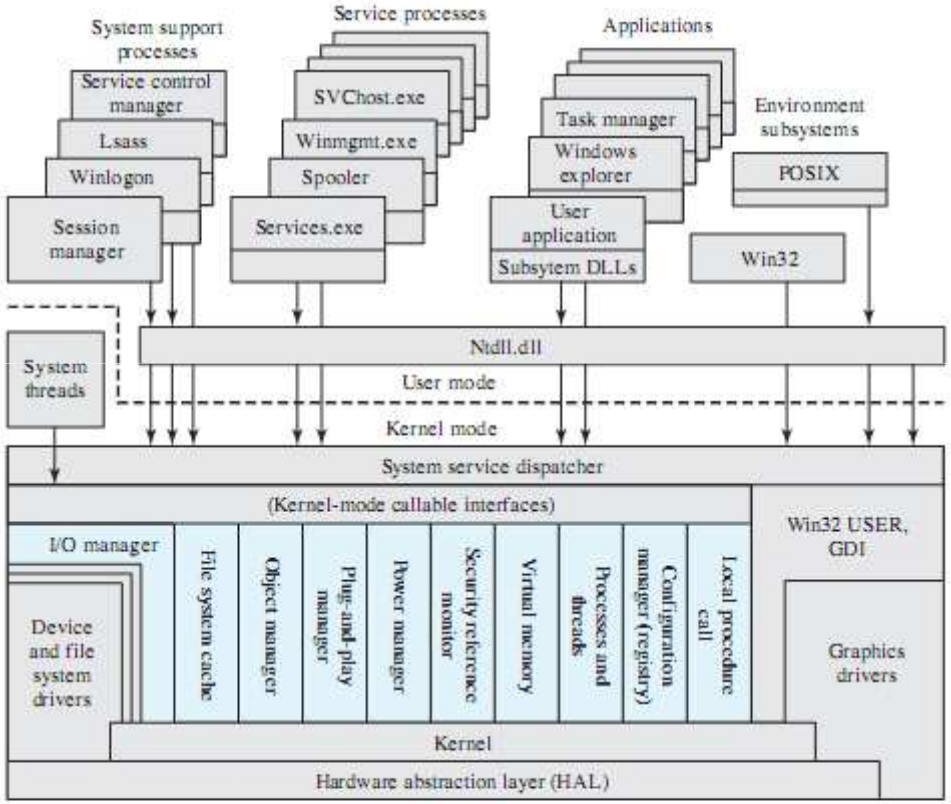


Single-User Multitasking

- From Windows 2000 on Windows development developed to exploit modern 32-bit and 64-bit microprocessors
- Designed for single users who run multiple programs
- Main drivers are:
  - Increased memory and speed of microprocessors
  - Support for virtual memory



Windows Architecture



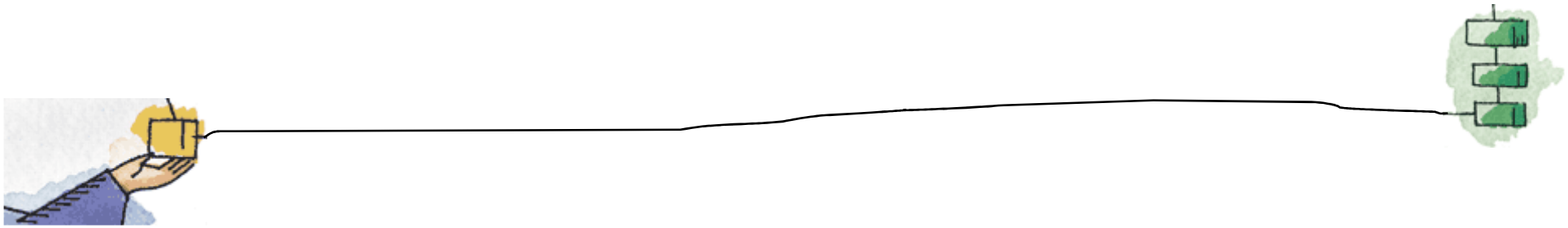
Lsass = local security authentication server
POSIX = portable operating system interface
GDI = graphics device interface
DLL = dynamic link libraries

Colored area indicates Executive

Figure 2.13 **Windows and Windows Vista Architecture [RUSS05]**

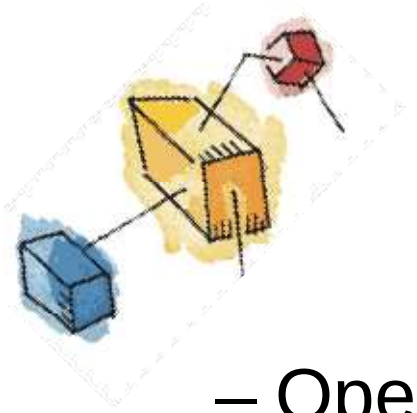
Client/Server Model

- Windows OS, protected subsystem, and applications all use a client/server model
  - Common in distributed systems, but can be used internal to a single system
- Processes communicate via RPC



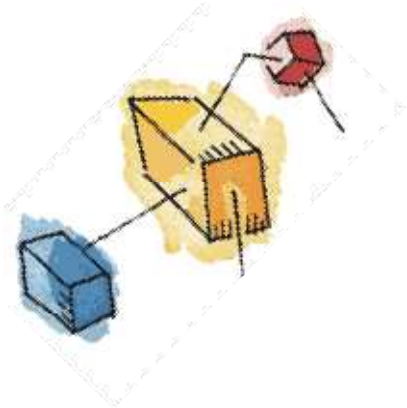
Windows Objects

- Windows draws heavily on the concepts of object-oriented design.
- Key Object Oriented concepts used by Windows are:
  - Encapsulation
  - Object class and instance



Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems
- Linux



Description of UNIX

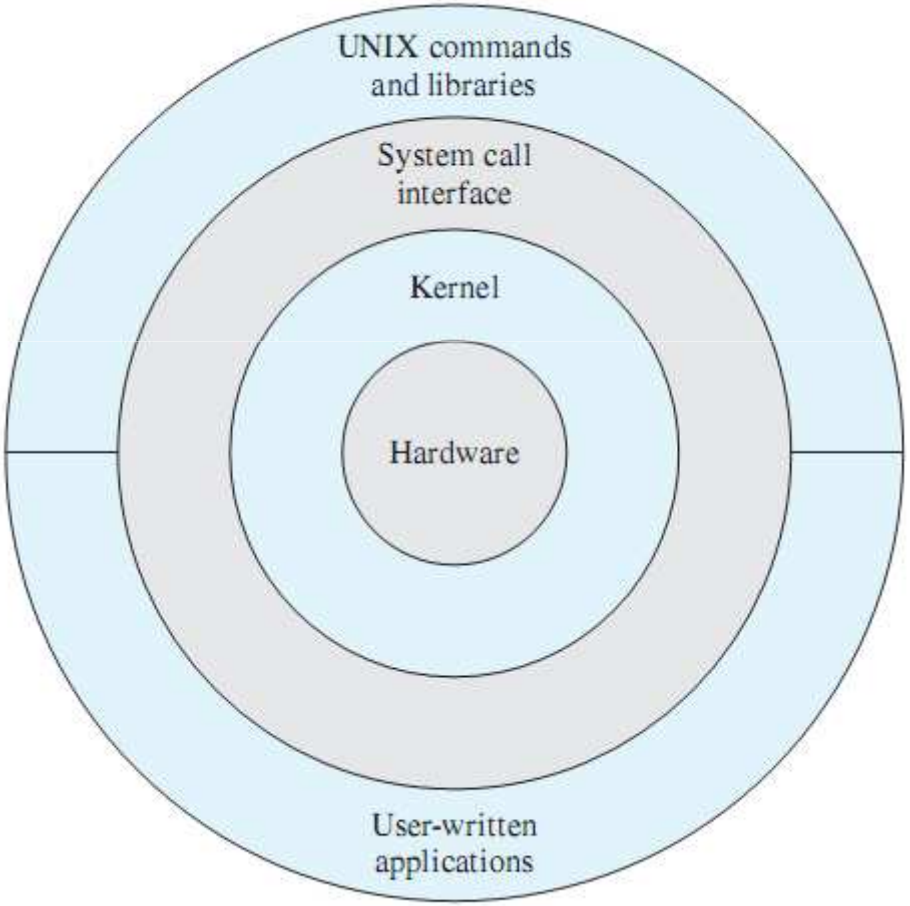
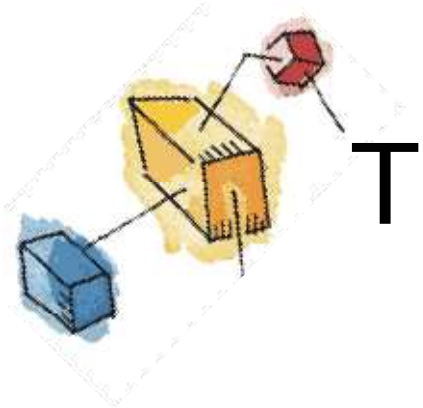


Figure 2.14 General UNIX Architecture



Traditional UNIX Kernel

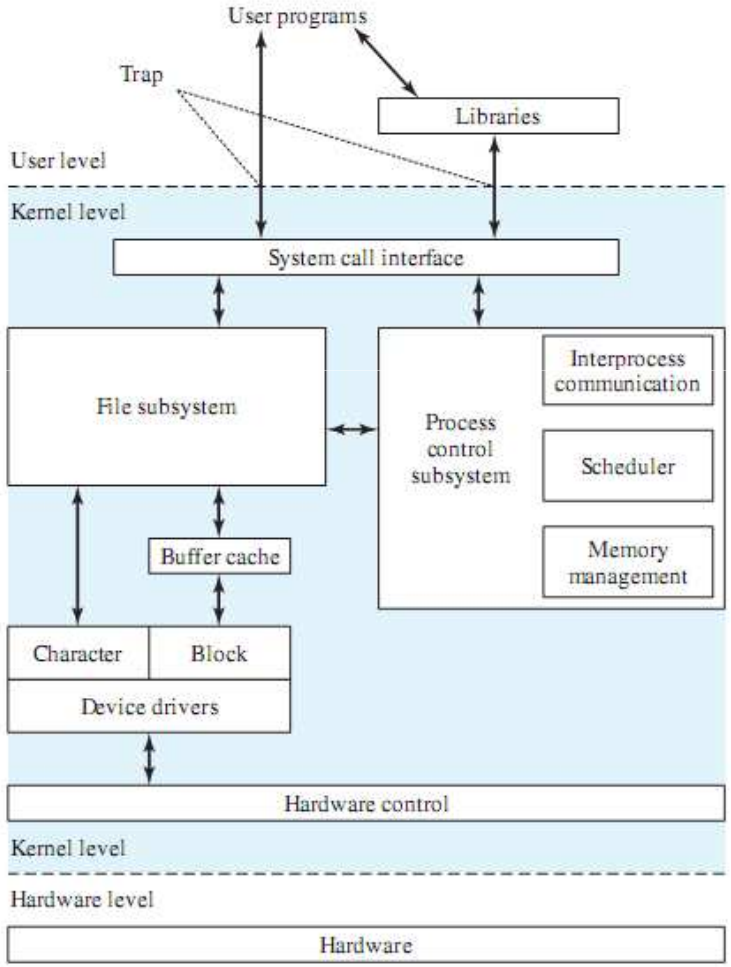
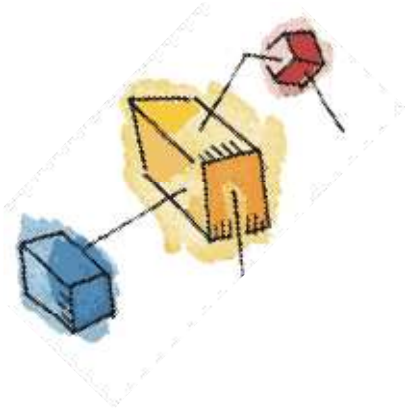


Figure 2.15 **Traditional UNIX Kernel**



System V Release 4 (SVR4)

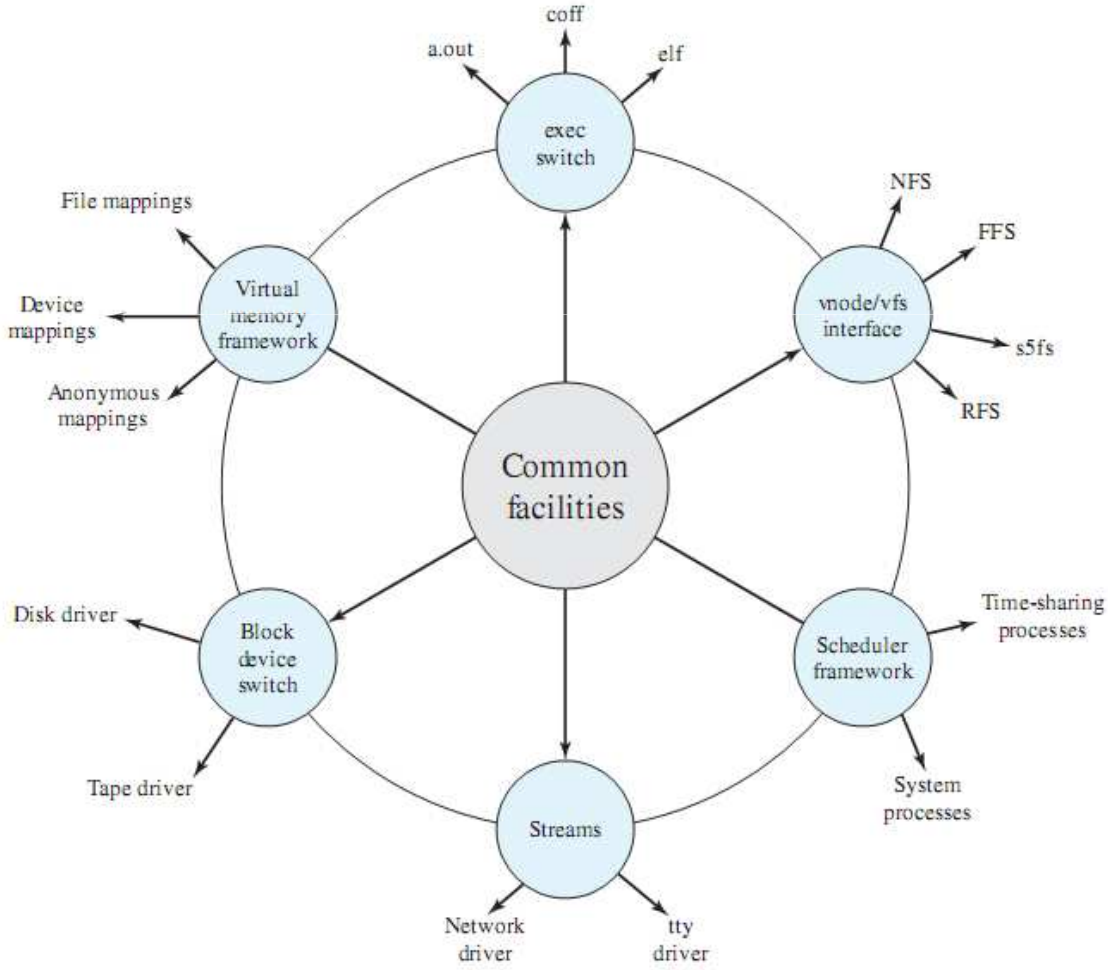
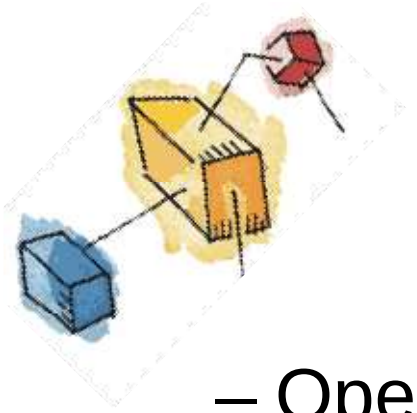
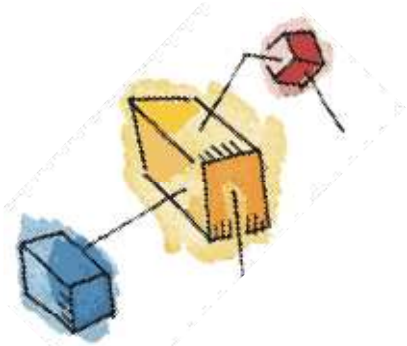


Figure 2.16 Modern UNIX Kernel



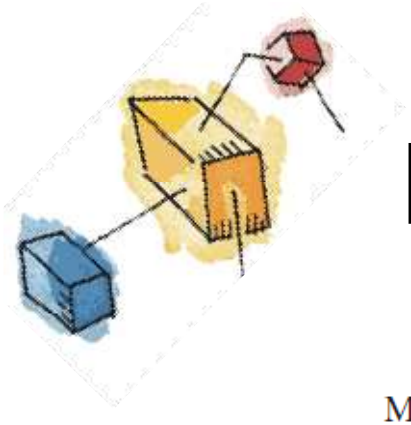
Roadmap

- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems
- Linux



Modular Monolithic Kernel

- Although monolithic, the kernel is structures as a collection of modules
  - Loadable modules
  - An object file which can be linked and unlinked at run time
- Characteristics:
  - Dynamic Linking
  - Stackable modules



Linux Kernel Modules

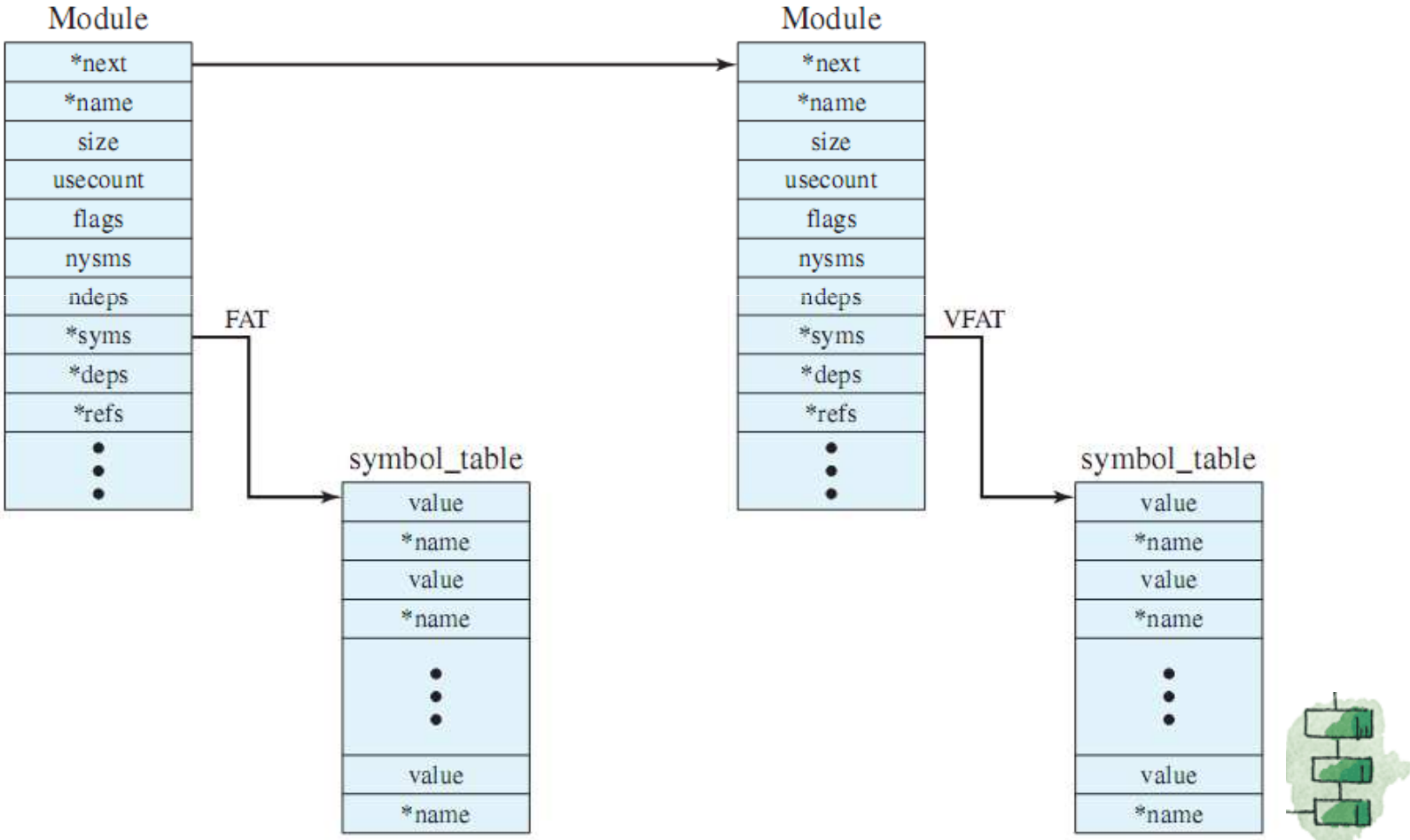
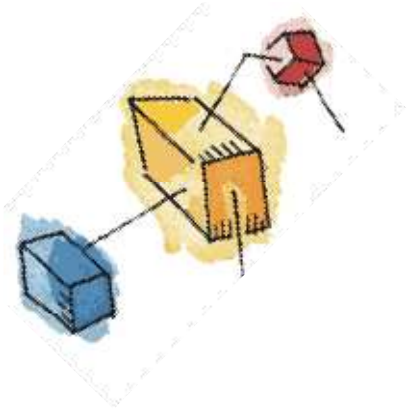


Figure 2.17 **Example List of Linux Kernel Modules**



Linux Kernel Components

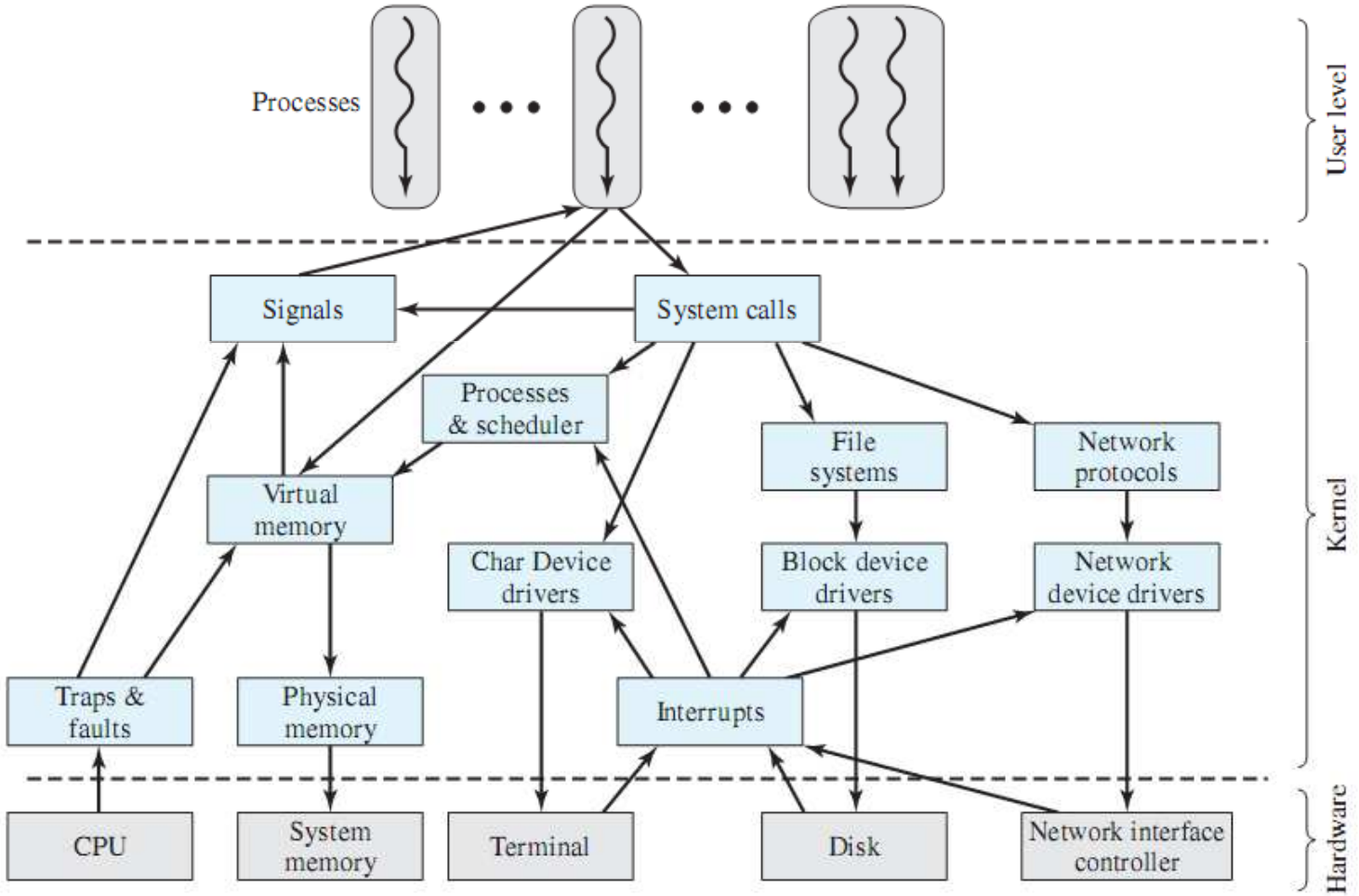


Figure 2.18 **Linux Kernel Components**