

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



**Chapter 3**  
**Process Description and Control**

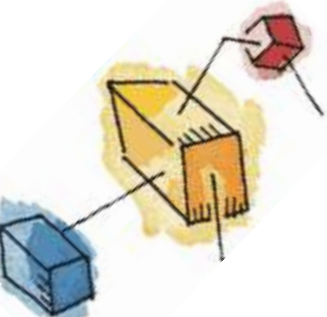


# Roadmap

→ How are processes represented and controlled by the OS.

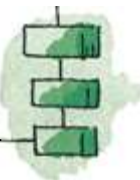
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.





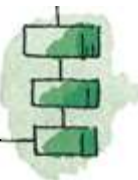
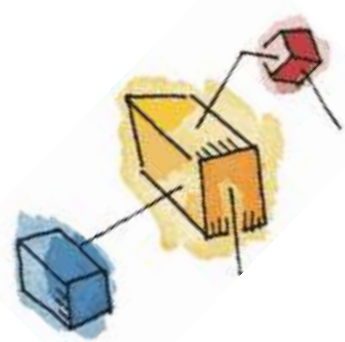
# Requirements of an Operating System

- *Fundamental Task: Process Management*
- The Operating System must
  - Interleave the execution of multiple processes
  - Allocate resources to processes, and protect the resources of each process from other processes,
  - Enable processes to share and exchange information,
  - Enable synchronization among processes.



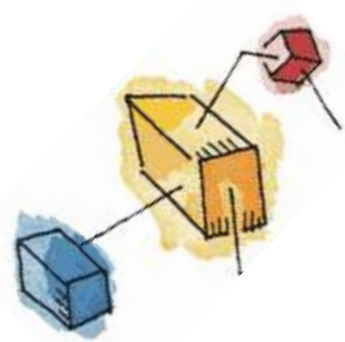
# Concepts

- From earlier chapters we saw:
  - Computer platforms consists of a collection of hardware resources
  - Computer applications are developed to perform some task
  - It is inefficient for applications to be written directly for a given hardware platform



# Concepts cont...

- OS provides an interface for applications to use
- OS provides a representation of resources that can be requested and accessed by application

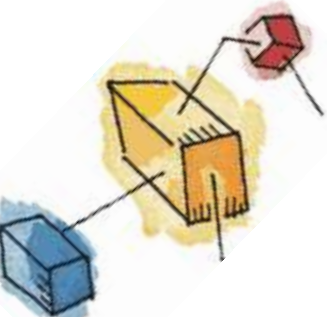




# The OS Manages Execution of Applications

- Resources are made available to multiple applications
- The processor is switched among multiple application
- The processor and I/O devices can be used efficiently





# What is a “*process*”?

- *A program in execution*
- An instance of a program running on a computer
- The entity that can be assigned to and executed on a processor
- A unit of activity characterized by the execution of a sequence of instructions, a current state, and an associated set of system instructions



# Process Elements

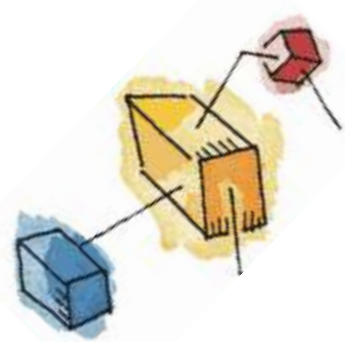
- A process is comprised of:
  - Program code (possibly shared)
  - A set of data
  - A number of attributes describing the state of the process





# Process Elements

- While the process is running it has a number of elements including
  - Identifier
  - State
  - Priority
  - Program counter
  - Memory pointers
  - Context data
  - I/O status information
  - Accounting information



# Process Control Block

- Contains the process elements
- Created and manage by the operating system
- Allows support for multiple processes

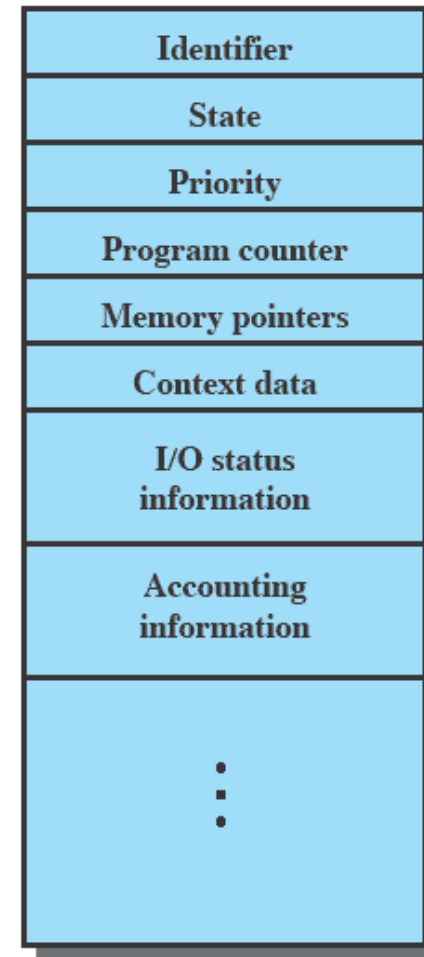


Figure 3.1 Simplified Process Control Block

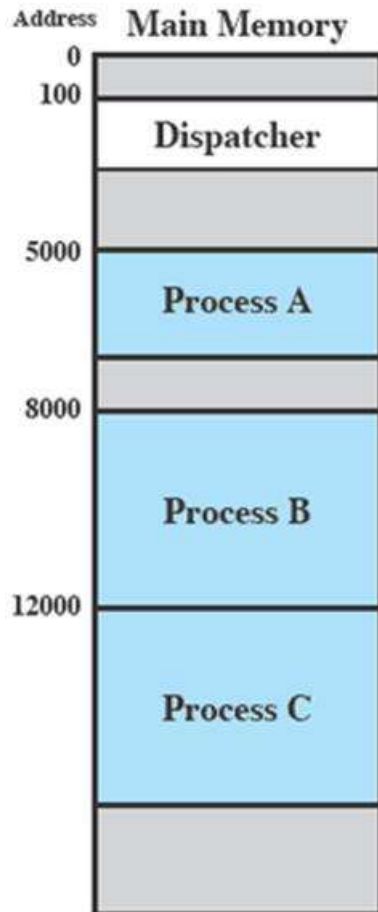


# Trace of the Process

- The behavior of an individual process is shown by listing the sequence of instructions that are executed
- This list is called a ***Trace***
- ***Dispatcher*** is a small program which switches the processor from one process to another



# Process Execution



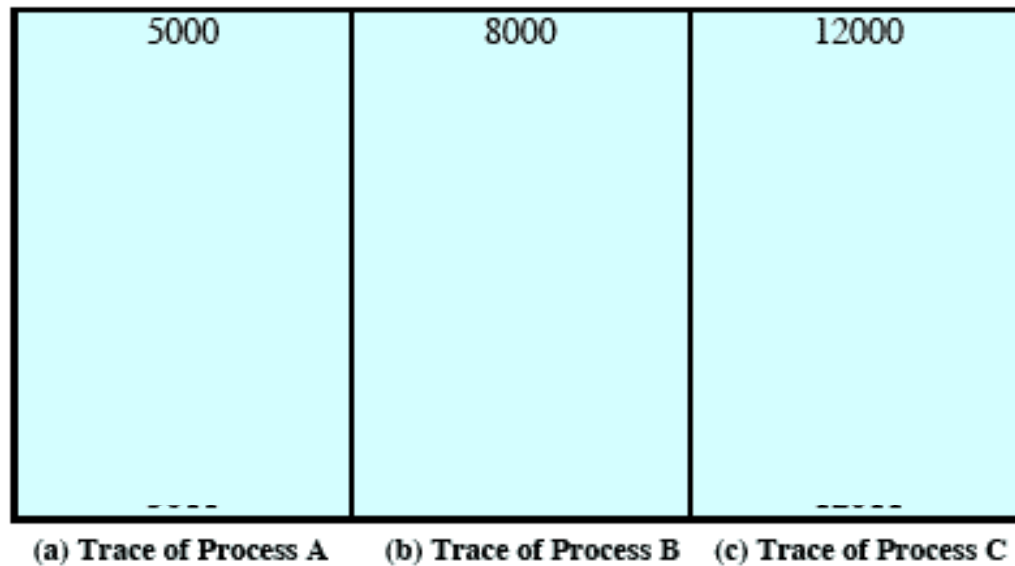
- Consider three processes being executed
- All are in memory (plus the dispatcher)
- Lets ignore virtual memory for this.





# Trace from the *processes* point of view:

- Each process runs to completion



5000 = Starting address of program of Process A  
8000 = Starting address of program of Process B  
12000 = Starting address of program of Process C

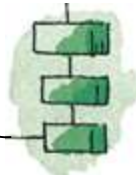
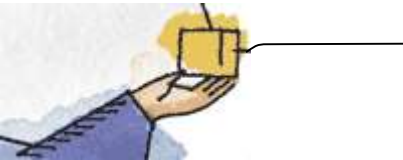
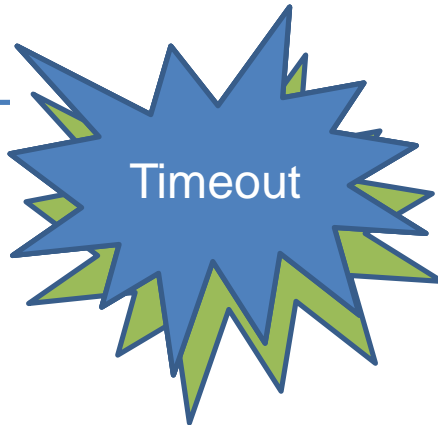
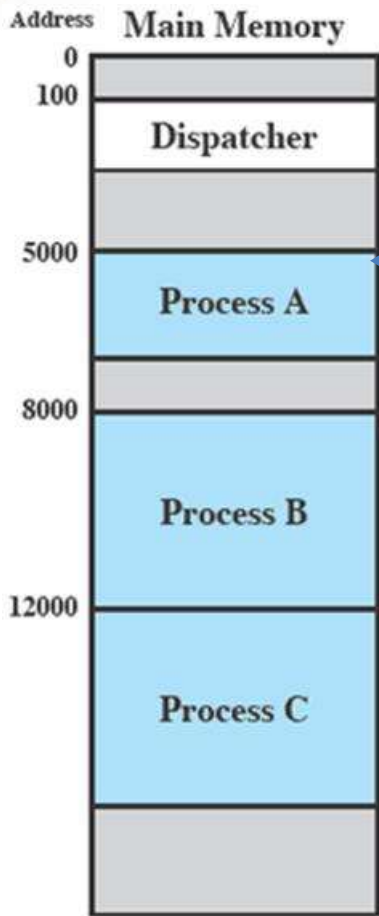


Figure 3.3 Traces of Processes of Figure 3.2

# Trace from Processors point of view



1	5000
2	5001
3	5002
4	5003
5	5004
6	5005
Timeout	
7	100
8	101
9	102
10	103
11	104
12	105
IO Request	
13	8000
14	8001
15	8002
16	8003
17	100
18	101
19	102
20	103
21	104
22	105
23	12000
24	12001
25	12002
26	12003

27	12004
28	12005
Timeout	
29	100
30	101
31	102
32	103
33	104
34	105
35	5006
36	5007
37	5008
38	5009
39	5010
40	5011
Timeout	
41	100
42	101
43	102
44	103
45	104
46	105
47	12006
48	12007
49	12008
50	12009
51	12010
52	12011
Timeout	

100 = Starting address of dispatcher program

Shaded areas indicate execution of dispatcher process;  
 first and third columns count instruction cycles;  
 second and fourth columns show address of instruction being executed

Figure 3.4 Combined Trace of Processes of Figure 3.2



# Roadmap

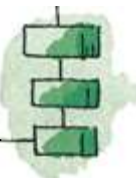
- How are processes represented and controlled by the OS.

→ **Process states** which characterize the behaviour of processes.

- **Data structures** used to manage processes.

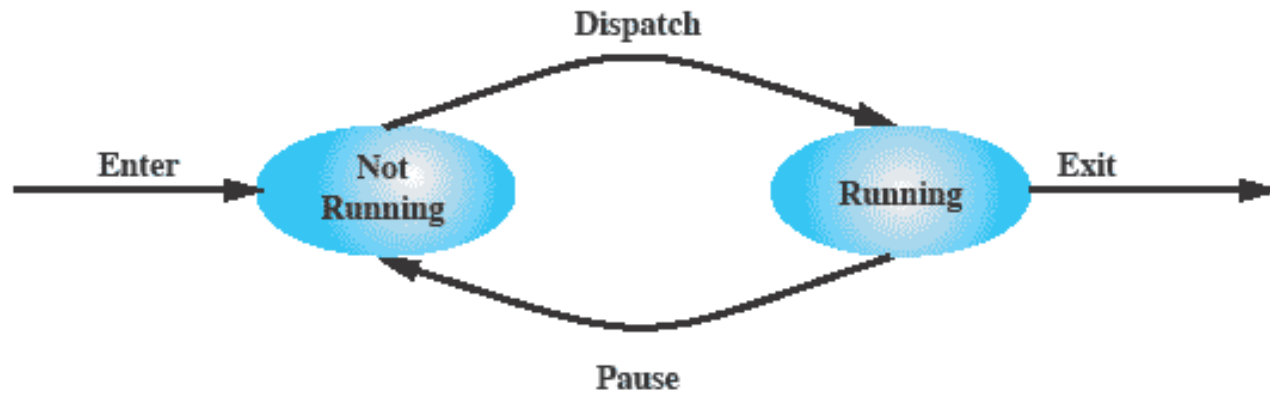
- Ways in which the OS uses these data structures to control process execution.

- Discuss process management in UNIX SVR4.



# Two-State Process Model

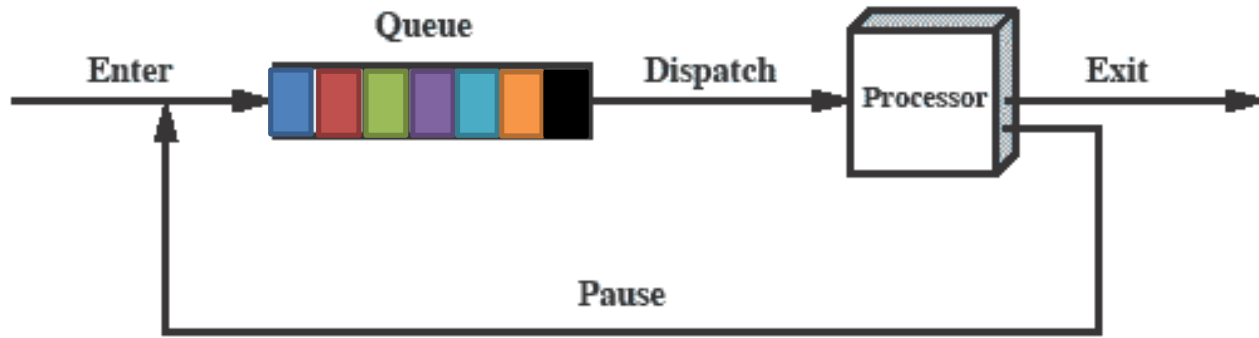
- Process may be in one of two states
  - Running
  - Not-running



(a) State transition diagram



# Queuing Diagram



(b) Queuing diagram

Etc ... processes moved by the dispatcher of the OS to the CPU then back to the queue until the task is completed





# Process Birth and Death

Creation	Termination
New batch job	Normal Completion
Interactive Login	Memory unavailable
Created by OS to provide a service	Protection error
Spawned by existing process	Operator or OS Intervention

See tables 3.1 and 3.2 for more





# Process Creation

- The OS builds a data structure to manage the process
- Traditionally, the OS created all processes
  - But it can be useful to let a running process create another
- This action is called ***process spawning***
  - ***Parent Process*** is the original, creating, process
  - ***Child Process*** is the new process





# Process Termination

- There must be some way that a process can indicate completion.
- This indication may be:
  - A HALT instruction generating an interrupt alert to the OS.
  - A user action (e.g. log off, quitting an application)
  - A fault or error
  - Parent process terminating



# Five-State Process Model

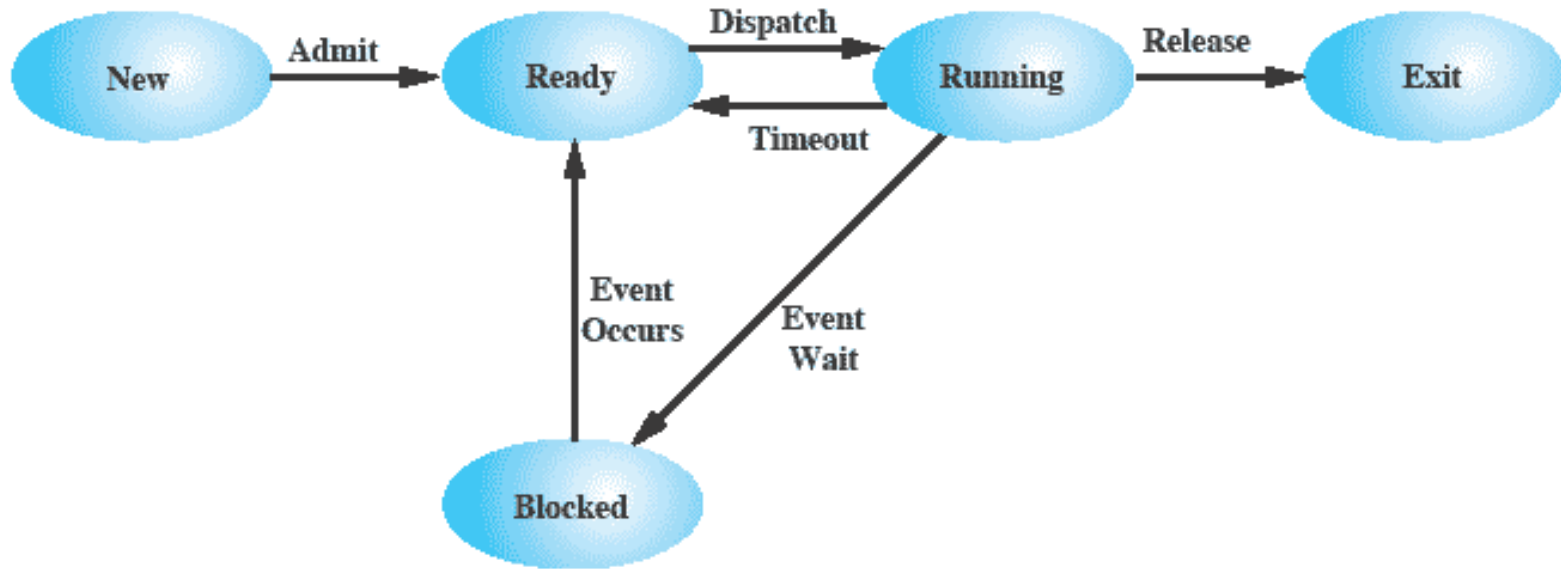
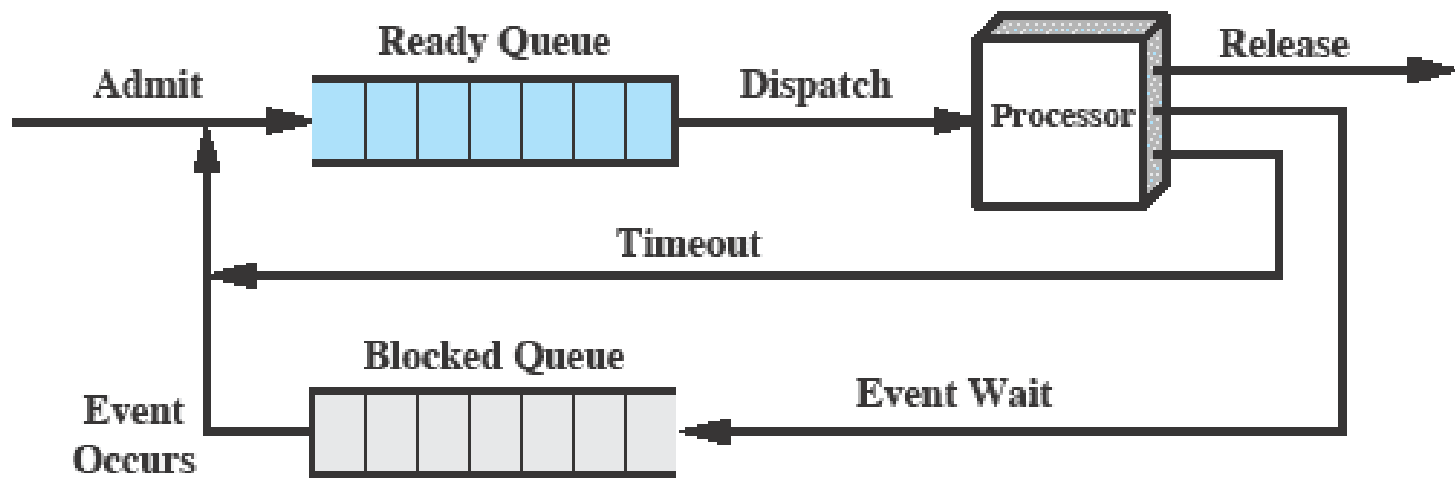


Figure 3.6 Five-State Process Model

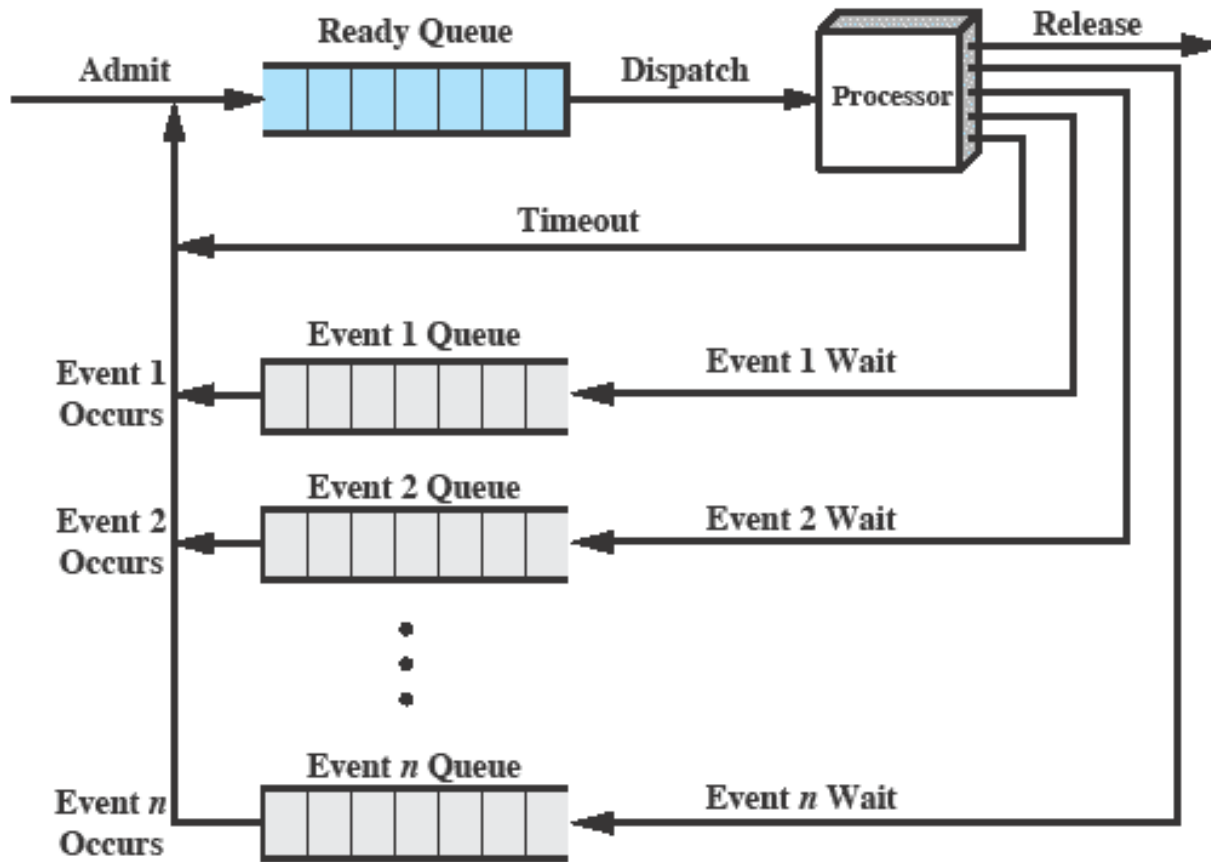
# Using Two Queues



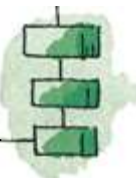
(a) Single blocked queue



# Multiple Blocked Queues



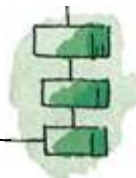
(b) Multiple blocked queues





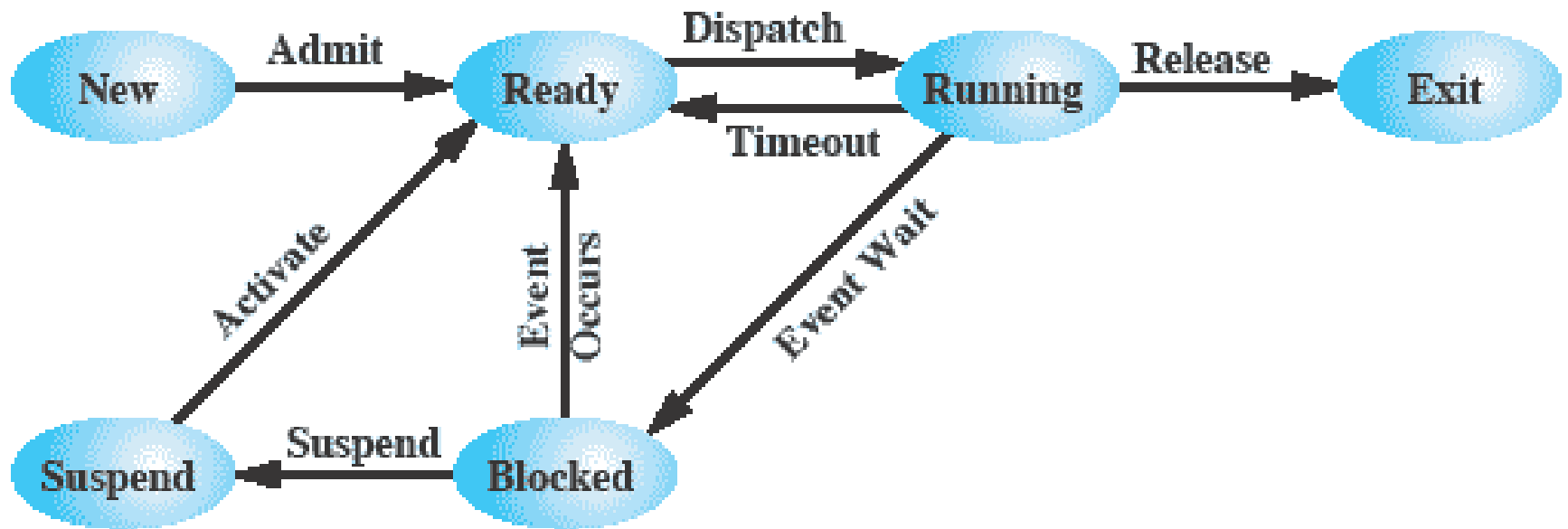
# Suspended Processes

- Processor is faster than I/O so all processes could be waiting for I/O
  - Swap these processes to disk to free up more memory and use processor on more processes
- Blocked state becomes ***suspend*** state when swapped to disk
- Two new states
  - Blocked/Suspend
  - Ready/Suspend





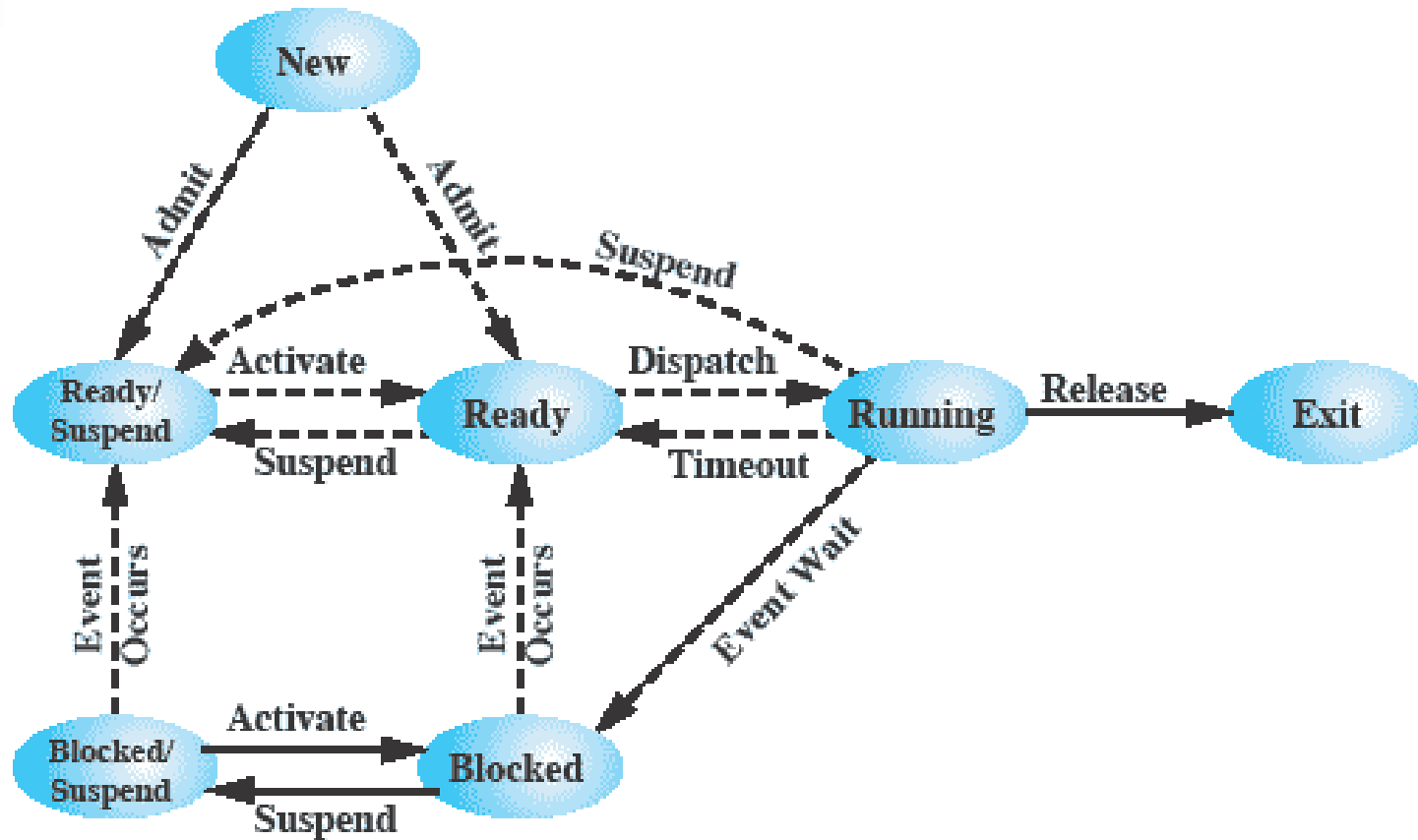
# One Suspend State



(a) With One Suspend State



# Two Suspend States



(b) With Two Suspend States



# Reason for Process Suspension

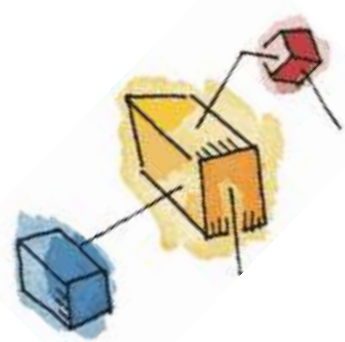
Reason	Comment
Swapping	The OS needs to release sufficient main memory to bring in a process that is ready to execute.
Other OS Reason	OS suspects process of causing a problem.
Interactive User Request	e.g. debugging or in connection with the use of a resource.
Timing	A process may be executed periodically (e.g., an accounting or system monitoring process) and may be suspended while waiting for the next time.
Parent Process Request	A parent process may wish to suspend execution of a descendent to examine or modify the suspended process, or to coordinate the activity of various descendants.

**Table 3.3 Reasons for Process Suspension**



# Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
  - Ways in which the OS uses these data structures to control process execution.
  - Discuss process management in UNIX SVR4.



# Processes and Resources

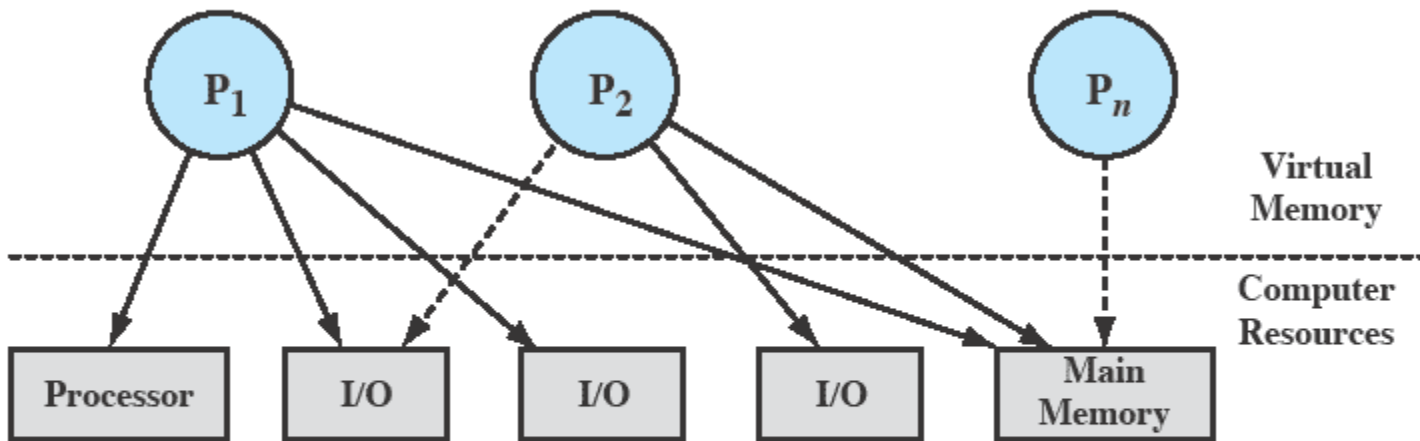


Figure 3.10 Processes and Resources (resource allocation at one snapshot in time)





# Operating System Control Structures

- For the OS is to manage processes and resources, it must have information about the current status of each process and resource.
- Tables are constructed for each entity the operating system manages



# OS Control Tables

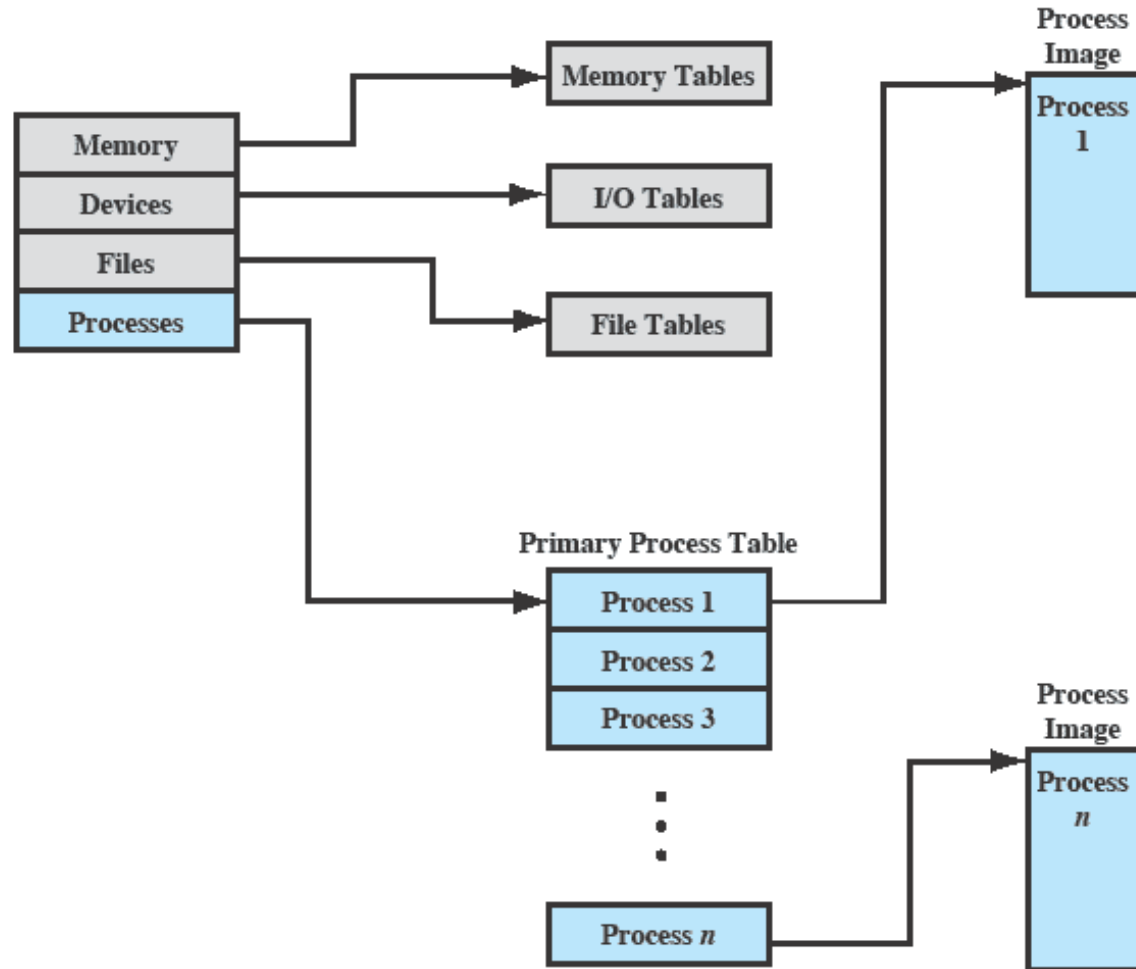
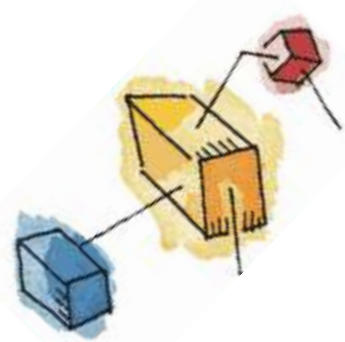


Figure 3.11 General Structure of Operating System Control Tables

# Memory Tables

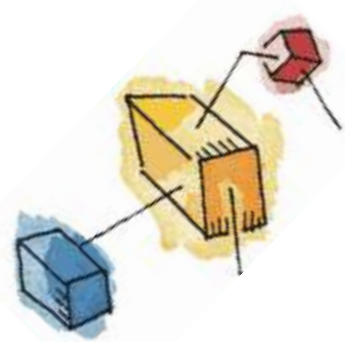
- Memory tables are used to keep track of both main and secondary memory.
- Must include this information:
  - Allocation of main memory to processes
  - Allocation of secondary memory to processes
  - Protection attributes for access to shared memory regions
  - Information needed to manage virtual memory





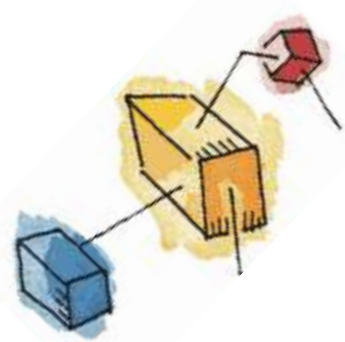
# I/O Tables

- Used by the OS to manage the I/O devices and channels of the computer.
- The OS needs to know
  - Whether the I/O device is available or assigned
  - The status of I/O operation
  - The location in main memory being used as the source or destination of the I/O transfer



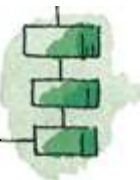
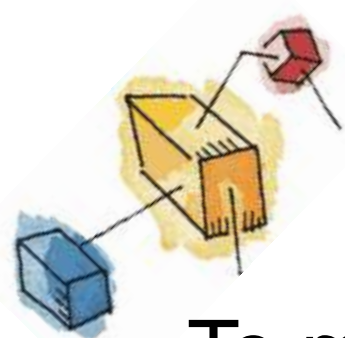
# File Tables

- These tables provide information about:
  - Existence of files
  - Location on secondary memory
  - Current Status
  - other attributes.
- Sometimes this information is maintained by a file management system



# Process Tables

- To manage processes the OS needs to know details of the processes
  - Current state
  - Process ID
  - Location in memory
  - etc
- Process control block
  - ***Process image*** is the collection of program. Data, stack, and attributes

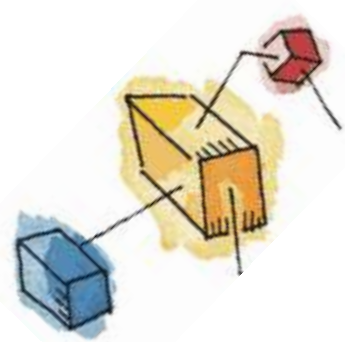




# Process Attributes

- We can group the process control block information into three general categories:
  - Process identification
  - Processor state information
  - Process control information





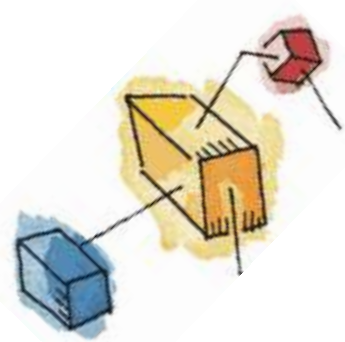
# Process Identification

- Each process is assigned a unique numeric identifier.
- Many of the other tables controlled by the OS may use process identifiers to cross-reference process tables

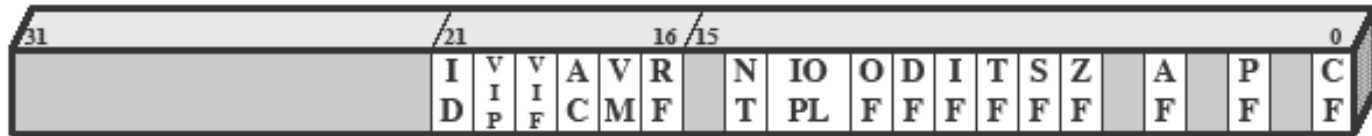


# Processor State Information

- This consists of the contents of processor registers.
  - User-visible registers
  - Control and status registers
  - Stack pointers
- Program status word (PSW)
  - contains status information
  - Example: the EFLAGS register on Pentium processors



# Pentium II EFLAGS Register



ID = Identification flag

VIP = Virtual interrupt pending

VIF = Virtual interrupt flag

AC = Alignment check

VM = Virtual 8086 mode

RF = Resume flag

NT = Nested task flag

IOPL = I/O privilege level

OF = Overflow flag

DF = Direction flag

IF = Interrupt enable flag

TF = Trap flag

SF = Sign flag

ZF = Zero flag

AF = Auxiliary carry flag

PF = Parity flag

CF = Carry flag

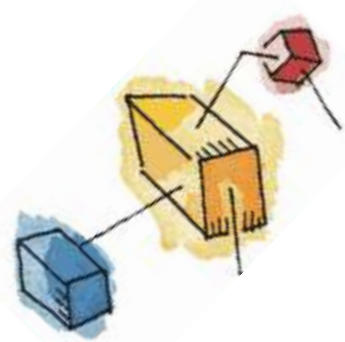
Also see Table 3.6

Figure 3.12 Pentium II EFLAGS Register



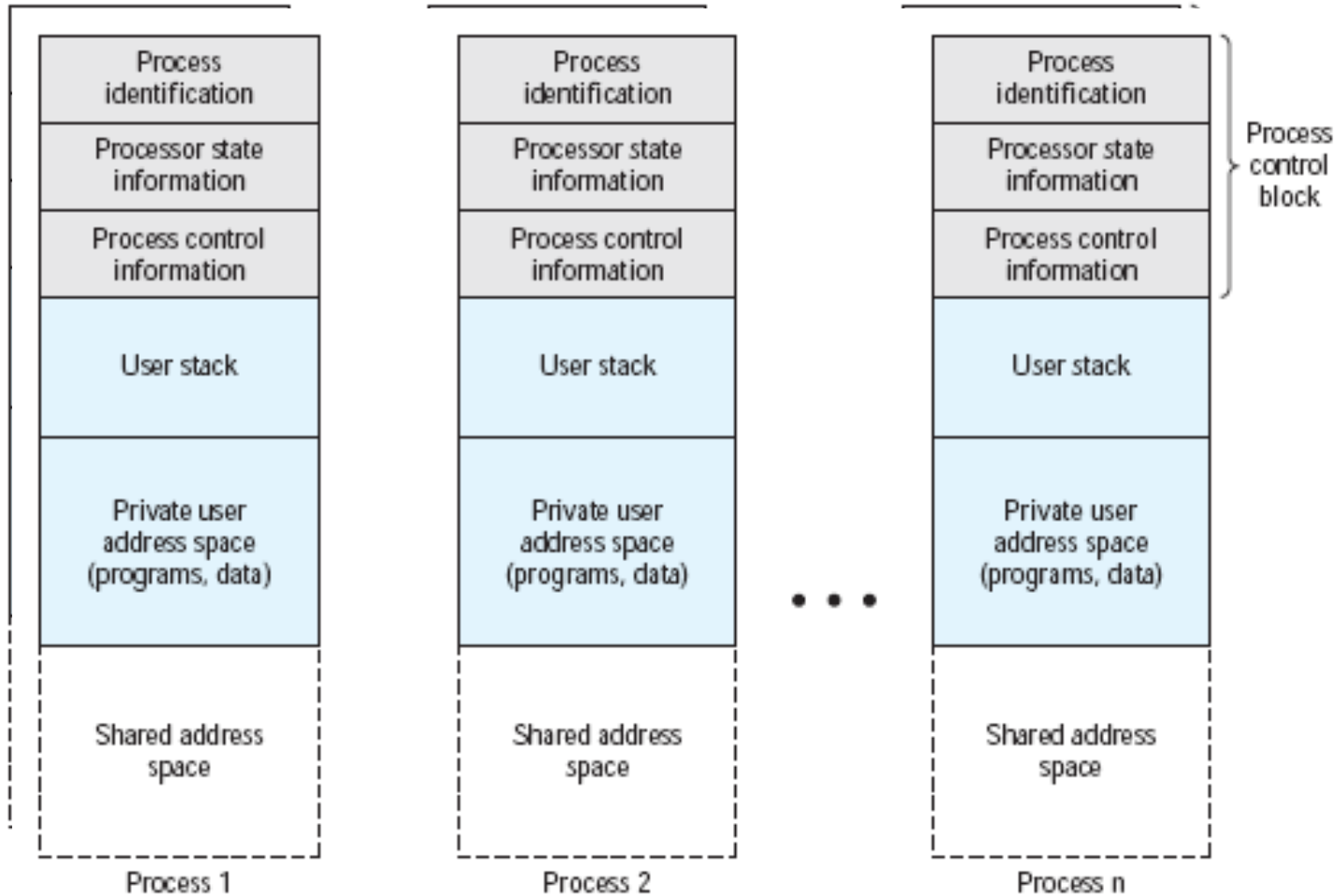
# Process Control Information

- This is the additional information needed by the OS to control and coordinate the various active processes.
  - See table 3.5 for scope of information





# Structure of Process Images in Virtual Memory



**F** Figure 3.13 User Processes in Virtual Memory



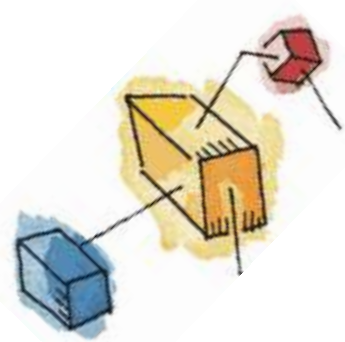
# Role of the Process Control Block

- The most important data structure in an OS
  - It defines the state of the OS
- Process Control Block requires protection
  - A faulty routine could cause damage to the block destroying the OS's ability to manage the process
  - Any design change to the block could affect many modules of the OS



# Roadmap

- How are processes represented and controlled by the OS.
- **Process states** which characterize the behaviour of processes.
- **Data structures** used to manage processes.
- Ways in which the OS uses these data structures to control process execution.
- Discuss process management in UNIX SVR4.





# Modes of Execution

- Most processors support at least two modes of execution
- User mode
  - Less-privileged mode
  - User programs typically execute in this mode
- System mode
  - More-privileged mode
  - Kernel of the operating system

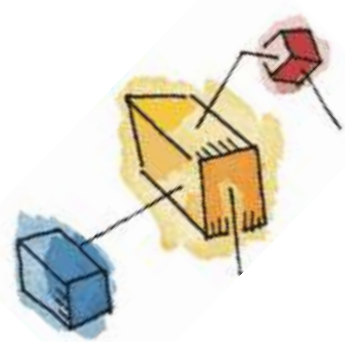




# Process Creation

- Once the OS decides to create a new process it:
  - Assigns a unique process identifier
  - Allocates space for the process
  - Initializes process control block
  - Sets up appropriate linkages
  - Creates or expand other data structures

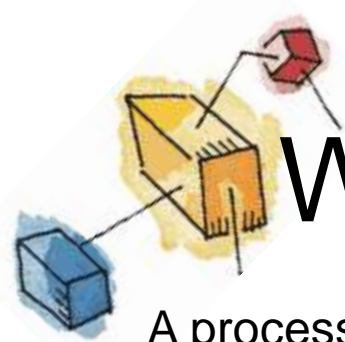




# Switching Processes

- Several design issues are raised regarding process switching
  - What events trigger a process switch?
  - We must distinguish between mode switching and process switching.
  - What must the OS do to the various data structures under its control to achieve a process switch?



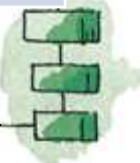


# When to switch processes

A process switch may occur any time that the OS has gained control from the currently running process. Possible events giving OS control are:

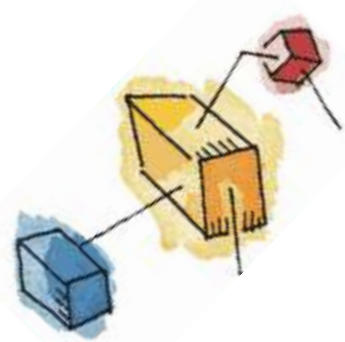
Mechanism	Cause	Use
Interrupt	External to the execution of the current instruction	Reaction to an asynchronous external event
Trap	Associated with the execution of the current instruction	Handling of an error or an exception condition
Supervisor call	Explicit request	Call to an operating system function

Table 3.8 Mechanisms for Interrupting the Execution of a Process

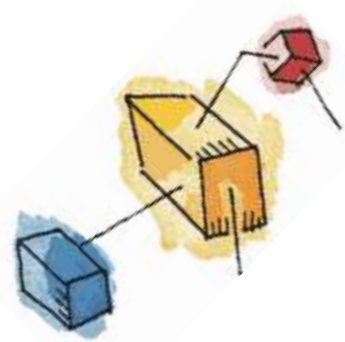


# Change of Process State ...

- The steps in a process switch are:
  1. Save context of processor including program counter and other registers
  2. Update the process control block of the process that is currently in the Running state
  3. Move process control block to appropriate queue – ready; blocked; ready/suspend



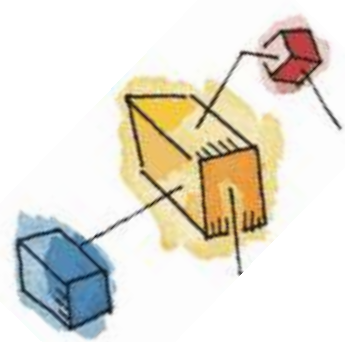




# Change of Process State cont...

4. Select another process for execution
5. Update the process control block of the process selected
6. Update memory-management data structures
7. Restore context of the selected process



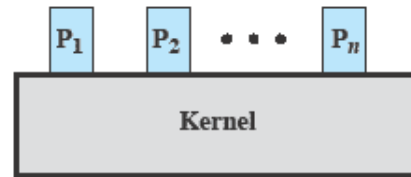


# Is the OS a Process?

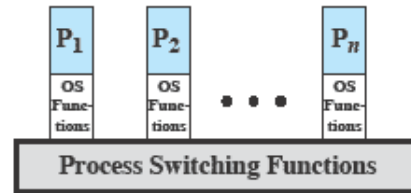
- If the OS is just a collection of programs and if it is executed by the processor just like any other program, is the OS a process?
- If so, how is it controlled?
  - Who (what) controls it?



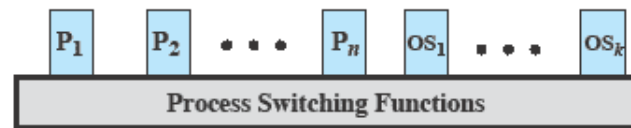
# Execution of the Operating System



(a) Separate kernel

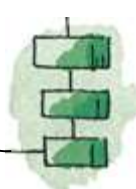
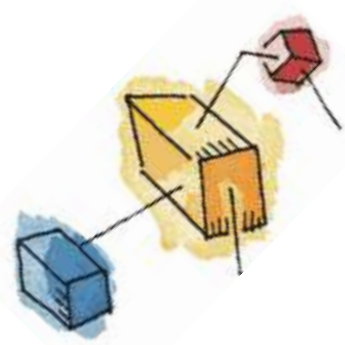


(b) OS functions execute within user processes



(c) OS functions execute as separate processes

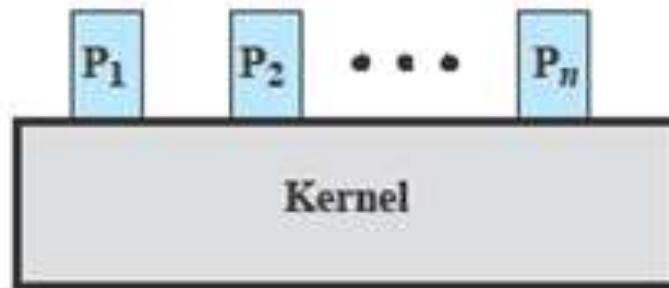
Figure 3.15 Relationship Between Operating System and User Processes





# Non-process Kernel

- Execute kernel outside of any process
- The concept of process is considered to apply only to user programs
  - Operating system code is executed as a separate entity that operates in privileged mode

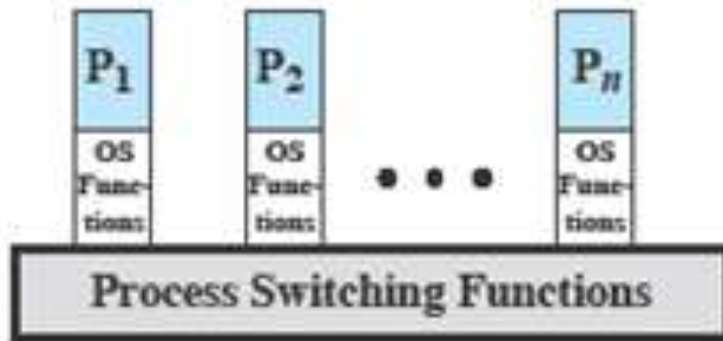


(a) Separate kernel



# Execution *Within* User Processes

- Execution Within User Processes
  - Operating system software within context of a user process
  - No need for Process Switch to run OS routine



(b) OS functions execute within user processes

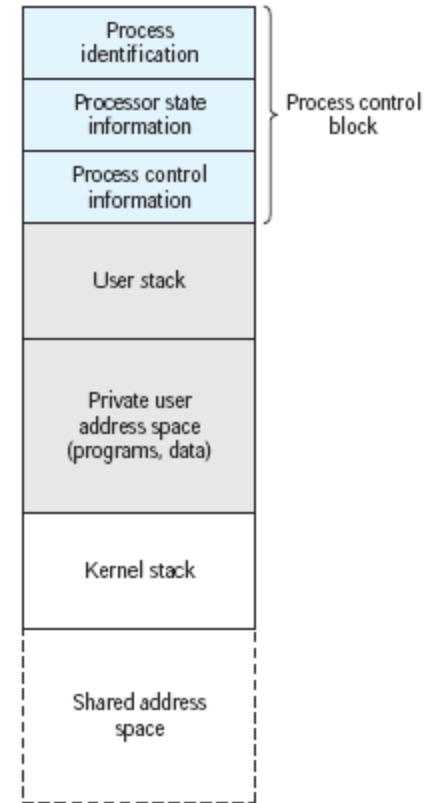
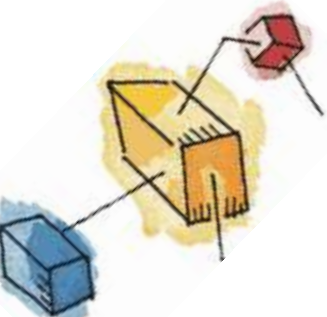
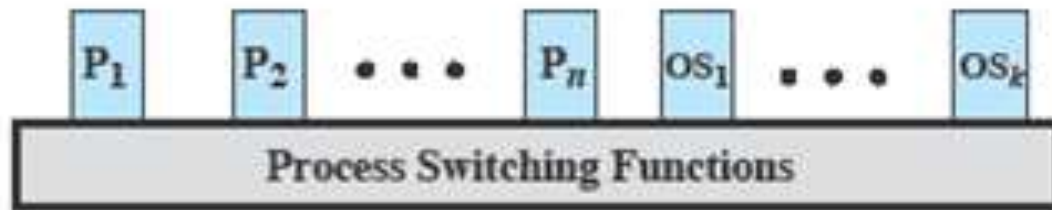


Figure 3.16 Process Image: Operating System Executes within User Space

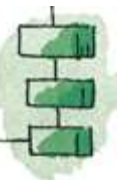


# Process-based Operating System

- Process-based operating system
  - Implement the OS as a collection of system process



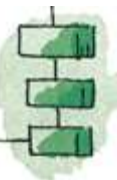
(c) OS functions execute as separate processes

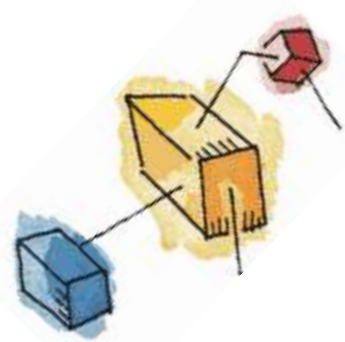




# Security Issues

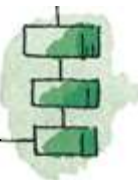
- An OS associates a set of privileges with each process.
  - Highest level being administrator, supervisor, or root, access.
- A key security issue in the design of any OS is to prevent anything (user or process) from gaining unauthorized privileges on the system
  - Especially - from gaining root access.





# System access threats

- Intruders
  - Masquerader (outsider)
  - Misfeasor (insider)
  - Clandestine user (outside or insider)
- Malicious software (malware)

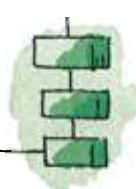






# Countermeasures: Intrusion Detection

- Intrusion detection systems are typically designed to detect human intruder and malicious software behaviour.
- May be host or network based
- Intrusion detection systems (IDS) typically comprise
  - Sensors
  - Analyzers
  - User Interface





# Countermeasures: Authentication

- Two Stages:
  - Identification
  - Verification
- Four Factors:
  - Something the individual **knows**
  - Something the individual **possesses**
  - Something the individual **is** (static biometrics)
  - Something the individual **does** (dynamic biometrics)





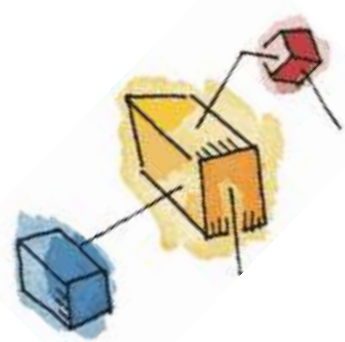
# Countermeasures: Access Control

- A policy governing access to resources
- A security administrator maintains an authorization database
  - The access control function consults this to determine whether to grant access.
- An auditing function monitors and keeps a record of user accesses to system resources.



# Countermeasures: Firewalls

- Traditionally, a firewall is a dedicated computer that:
  - interfaces with computers outside a network
  - has special security precautions built into it to protect sensitive files on computers within the network.

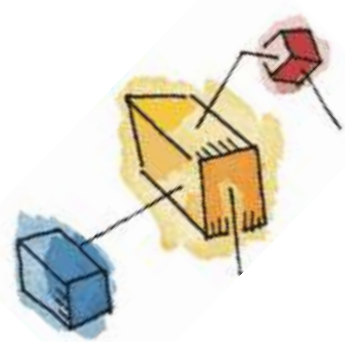




# Roadmap

- How are processes represented and controlled by the OS.
  - **Process states** which characterize the behaviour of processes.
  - **Data structures** used to manage processes.
  - Ways in which the OS uses these data structures to control process execution.
- ➔ Discuss process management in UNIX SVR4.

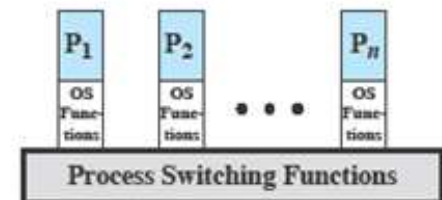




# Unix SVR4

## System V Release 4

- Uses the model of fig3.15b where most of the OS executes in the user process
- System Processes - Kernel mode only
- User Processes
  - User mode to execute user programs and utilities
  - Kernel mode to execute instructions that belong to the kernel.



(b) OS functions execute within user processes



# UNIX Process State Transition Diagram

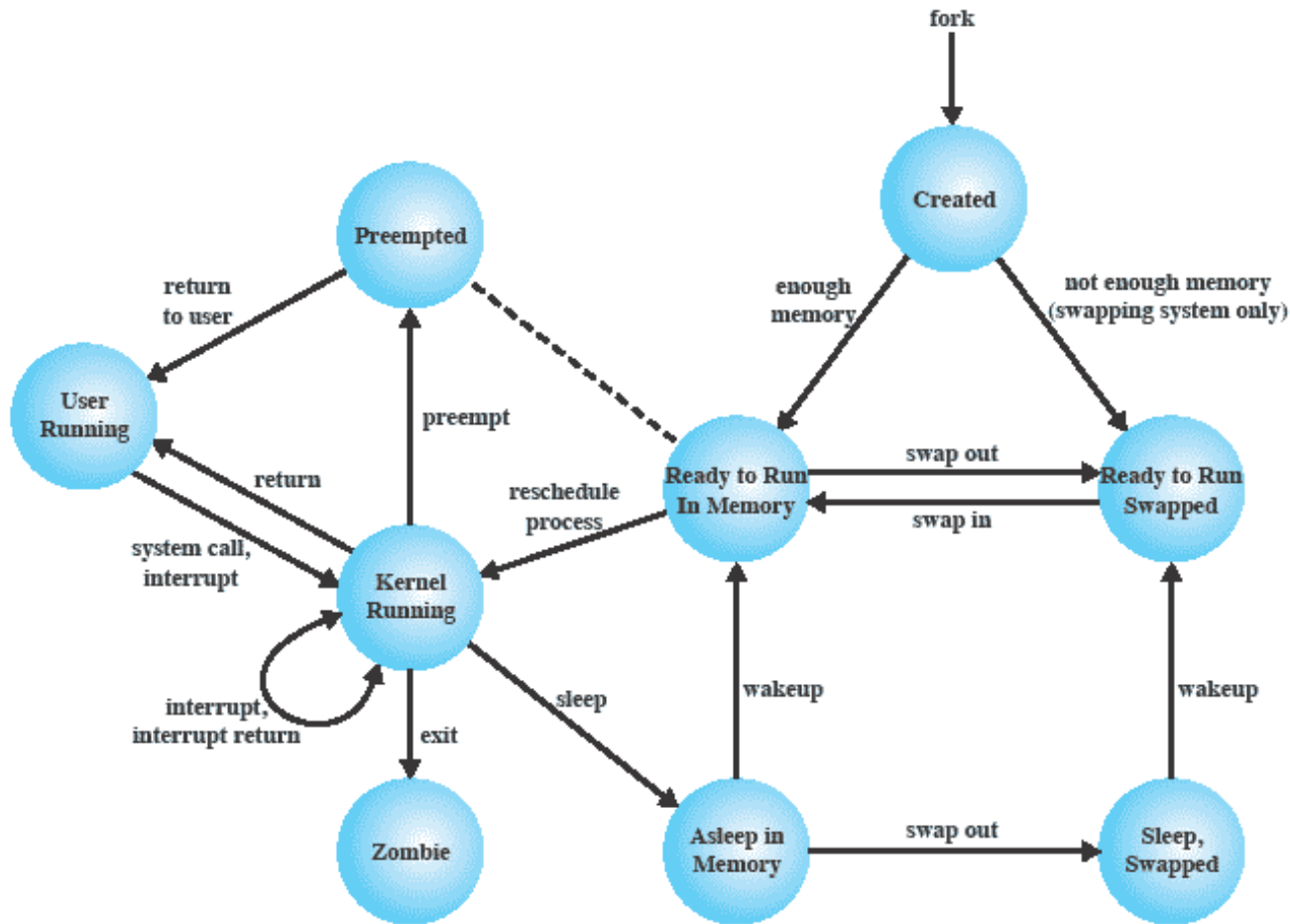
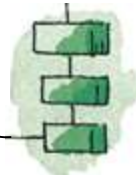



Figure 3.17 UNIX Process State Transition Diagram



# UNIX Process States

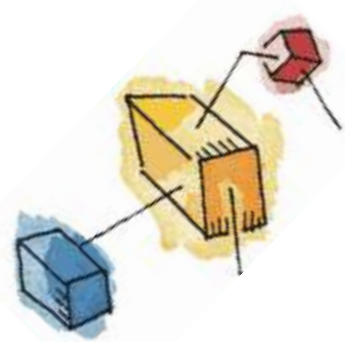
<b>User Running</b>	Executing in user mode.
<b>Kernel Running</b>	Executing in kernel mode.
<b>Ready to Run, in Memory</b>	Ready to run as soon as the kernel schedules it.
<b>Asleep in Memory</b>	Unable to execute until an event occurs; process is in main memory (a blocked state).
<b>Ready to Run, Swapped</b>	Process is ready to run, but the swapper must swap the process into main memory before the kernel can schedule it to execute.
<b>Sleeping, Swapped</b>	The process is awaiting an event and has been swapped to secondary storage (a blocked state).
<b>Preempted</b>	Process is returning from kernel to user mode, but the kernel preempts it and does a process switch to schedule another process.
<b>Created</b>	Process is newly created and not yet ready to run.
<b>Zombie</b>	Process no longer exists, but it leaves a record for its parent process to collect.





# A Unix Process

- A process in UNIX is a set of data structures that provide the OS with all of the information necessary to manage and dispatch processes.
- See Table 3.10 which organizes the elements into three parts:
  - user-level context,
  - register context, and
  - system-level context.



# Process Creation

- Process creation is by means of the kernel system call, `fork( )`.
- This causes the OS, in Kernel Mode, to:
  1. Allocate a slot in the process table for the new process.
  2. Assign a unique process ID to the child process.
  3. Copy of process image of the parent, with the exception of any shared memory.



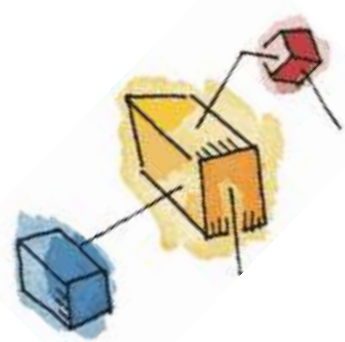
# Process Creation cont...

4. Increment the counters for any files owned by the parent, to reflect that an additional process now also owns those files.
5. Assign the child process to the Ready to Run state.
6. Returns the ID number of the child to the parent process, and a 0 value to the child process.

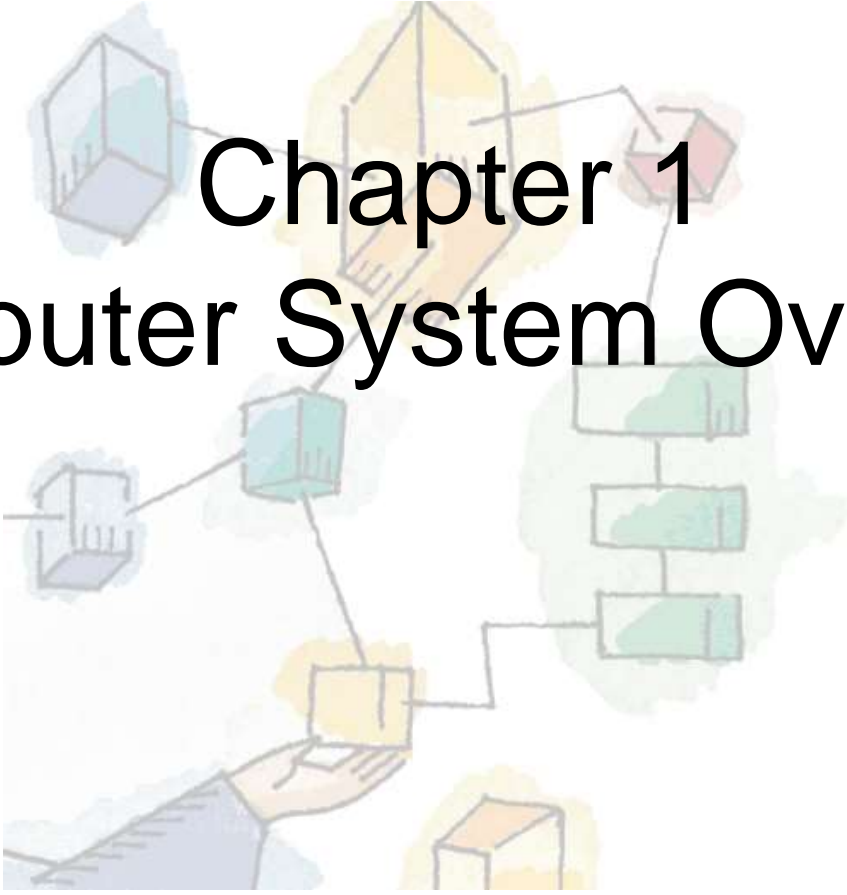


# After Creation

- After creating the process the Kernel can do one of the following, as part of the dispatcher routine:
  - Stay in the parent process.
  - Transfer control to the child process
  - Transfer control to another process.



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



# Chapter 1

## Computer System Overview



# Roadmap



## Basic Elements

- Processor Registers
- Instruction Execution
- Interrupts
- The Memory Hierarchy
- Cache Memory
- I/O Communication Techniques





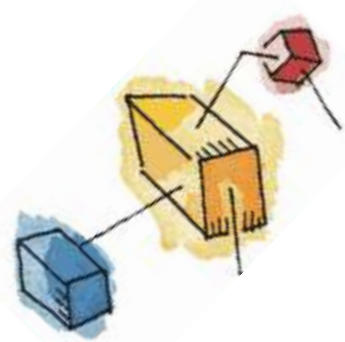
# Operating System

- Exploits the hardware resources of one or more processors
- Provides a set of services to system users
- Manages secondary memory and I/O devices



# A Computer's Basic Elements

- Processor
- Main Memory
- I/O Modules
- System Bus





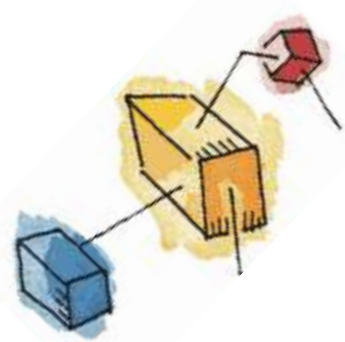
# Processor

- Controls operation, performs data processing
- Two internal registers
  - Memory address register (MAR)
  - Memory buffer register (MBR)
- I/O address register
- I/O buffer register



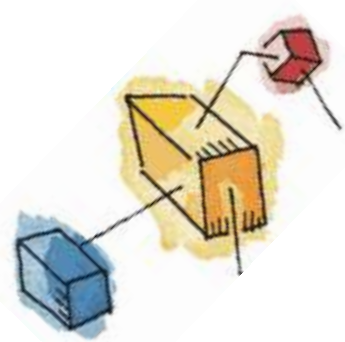
# Main Memory

- Volatile
  - Data is typically lost when power is removed
- Referred to as real memory or primary memory
- Consists of a set of locations defined by sequentially numbers addresses
  - Containing either data or instructions



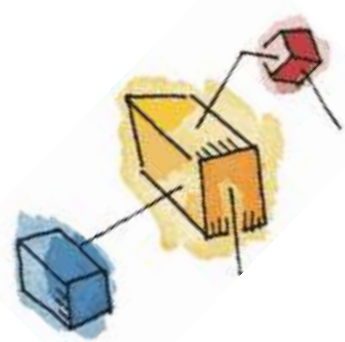
# I/O Modules

- Moves data between the computer and the external environment such as:
  - Storage (e.g. hard drive)
  - Communications equipment
  - Terminals
- Specified by an I/O Address Register
  - (I/OAR)



# System Bus

- Communication among processors, main memory, and I/O modules



# Top-Level View

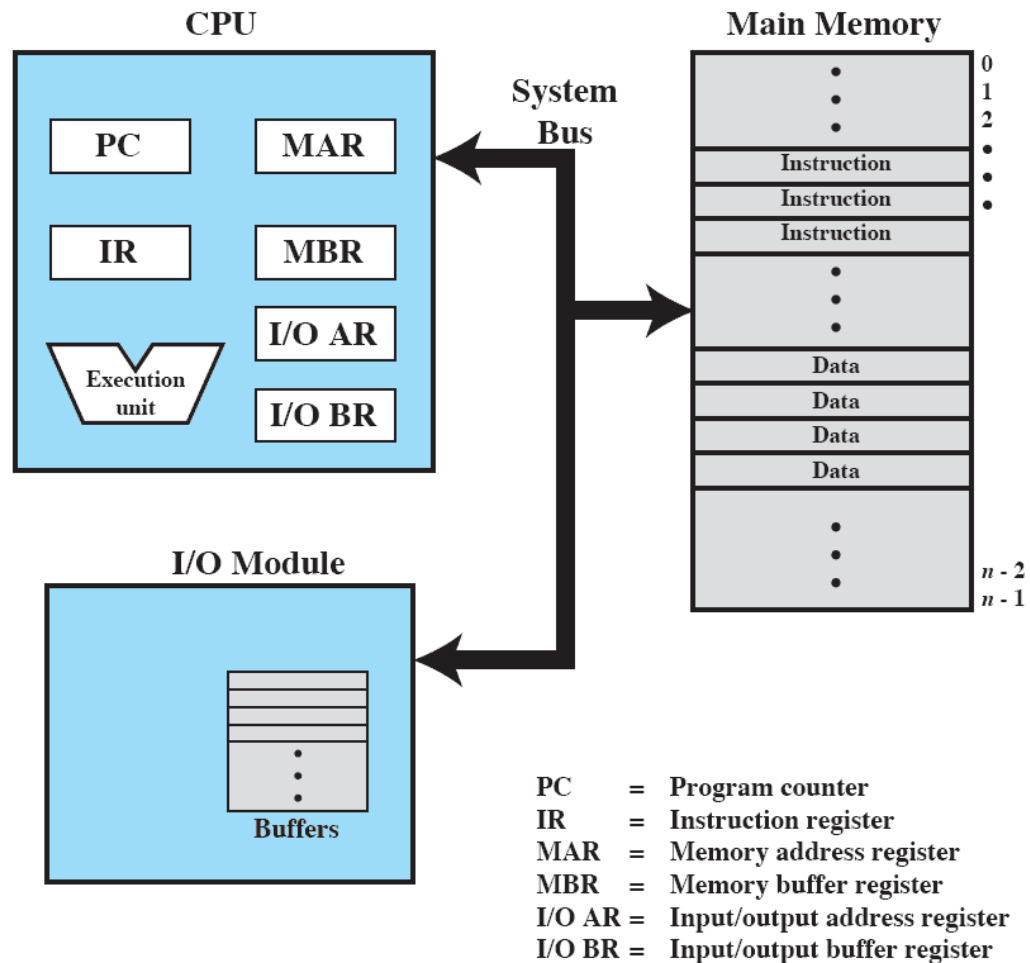


Figure 1.1 Computer Components: Top-Level View

# Roadmap



- Basic Elements

→ Processor Registers

- Instruction Execution

- Interrupts

- The Memory Hierarchy

- Cache Memory

- I/O Communication Techniques





# Processor Registers

- Faster and smaller than main memory
- User-visible registers
  - Enable programmer to minimize main memory references by optimizing register use
- Control and status registers
  - Used by processor to control operating of the processor
  - Used by privileged OS routines to control the execution of programs





# User-Visible Registers

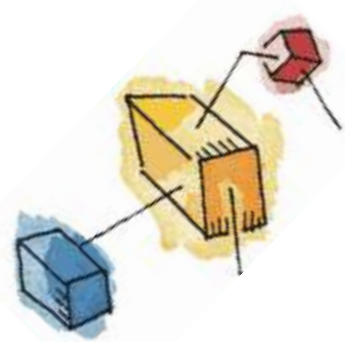
- May be referenced by machine language
  - Available to all programs – application programs and system programs
- Types of registers typically available are:
  - data,
  - address,
  - condition code registers.





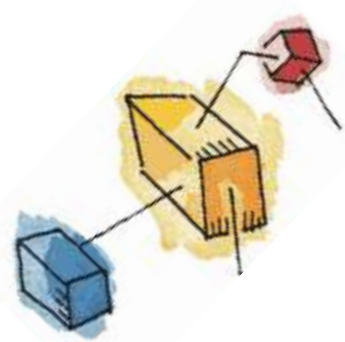
# Data and Address Registers

- Data
  - Often general purpose
  - But some restrictions may apply
- Address
  - Index Register
  - Segment pointer
  - Stack pointer



# Control and Status Registers

- Program counter (PC)
  - Contains the address of an instruction to be fetched
- Instruction register (IR)
  - Contains the instruction most recently fetched
- Program status word (PSW)
  - Contains status information





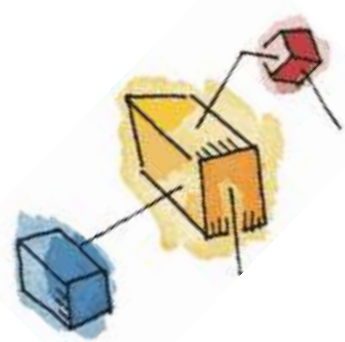
# Condition codes

- Usually part of the control register
  - Also called *flags*
- Bits set by processor hardware as a result of operations
  - Read only, intended for feedback regarding the results of instruction execution.



# Roadmap

- Basic Elements
- Processor Registers
- **Instruction Execution**
- Interrupts
- The Memory Hierarchy
- Cache Memory
- I/O Communication Techniques





# Instruction Execution

- A program consists of a set of instructions stored in memory
- Two steps
  - Processor reads (fetches) instructions from memory
  - Processor executes each instruction



# Basic Instruction Cycle

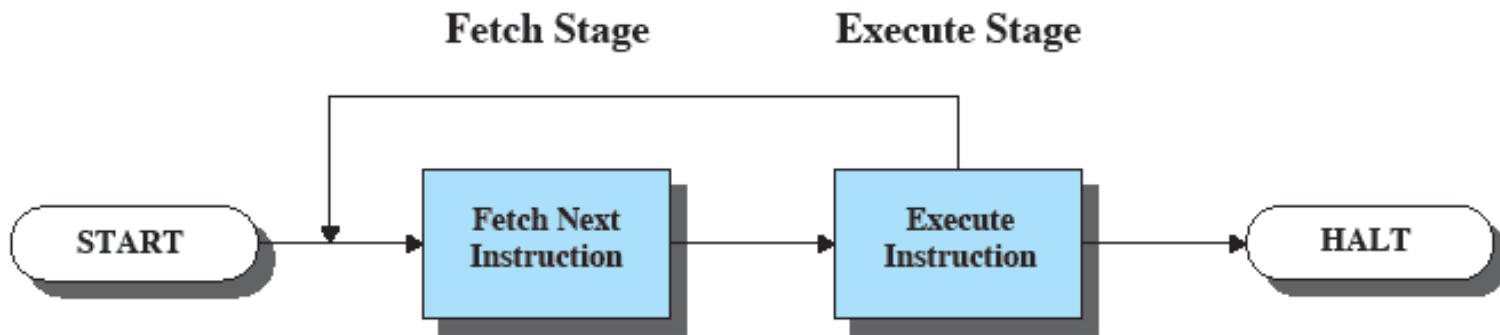
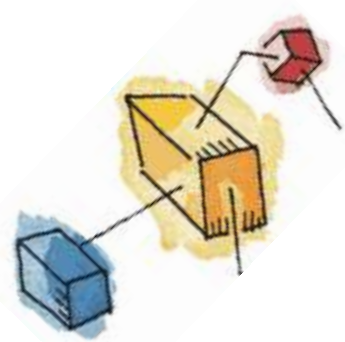


Figure 1.2 Basic Instruction Cycle



# Instruction Fetch and Execute

- The processor fetches the instruction from memory
- Program counter (PC) holds address of the instruction to be fetched next
  - PC is incremented after each fetch





# Instruction Register

- Fetched instruction loaded into instruction register
- Categories
  - Processor-memory,
  - processor-I/O,
  - Data processing,
  - Control

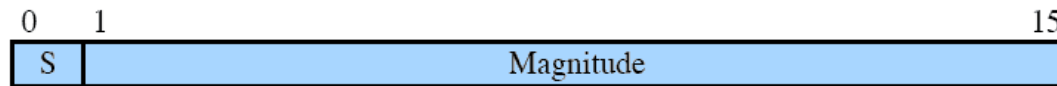




# Characteristics of a Hypothetical Machine



(a) Instruction format



(b) Integer format

Program counter (PC) = Address of instruction  
Instruction register (IR) = Instruction being executed  
Accumulator (AC) = Temporary storage

(c) Internal CPU registers

0001 = Load AC from memory  
0010 = Store AC to memory  
0101 = Add to AC from memory

(d) Partial list of opcodes

Figure 1.3 Characteristics of a Hypothetical Machine

# Example of Program Execution

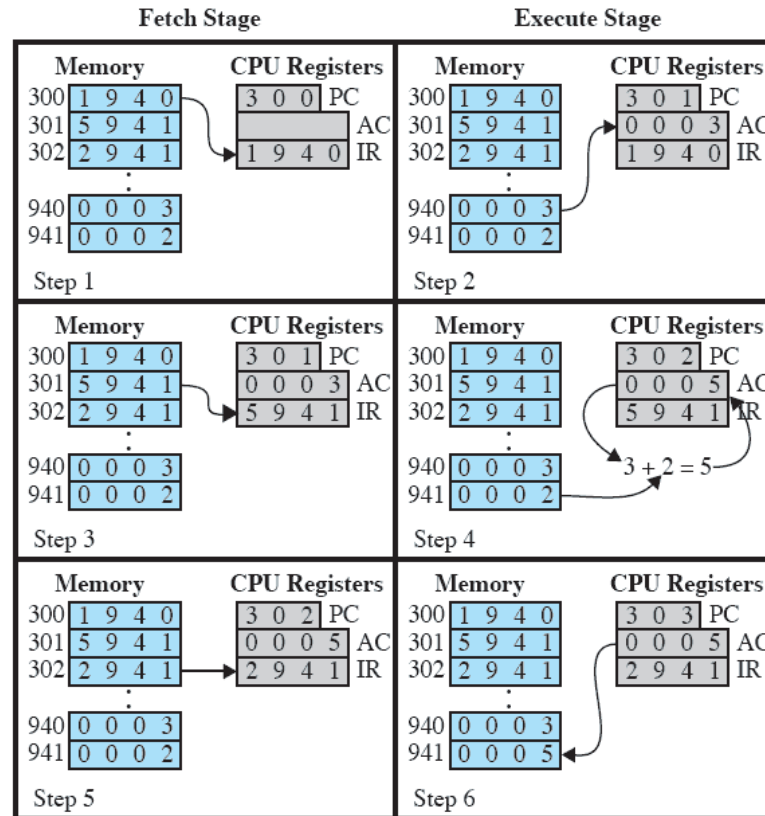


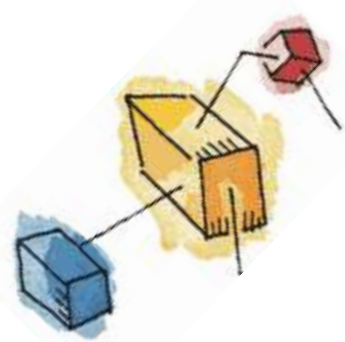
Figure 1.4 Example of Program Execution  
(contents of memory and registers in hexadecimal)

# Roadmap

- Basic Elements
- Processor Registers
- Instruction Execution

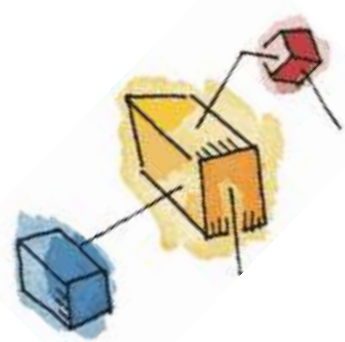
## → Interrupts

- The Memory Hierarchy
- Cache Memory
- I/O Communication Techniques



# Interrupts

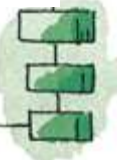
- Interrupt the normal sequencing of the processor
- Provided to improve processor utilization
  - Most I/O devices are slower than the processor
  - Processor must pause to wait for device



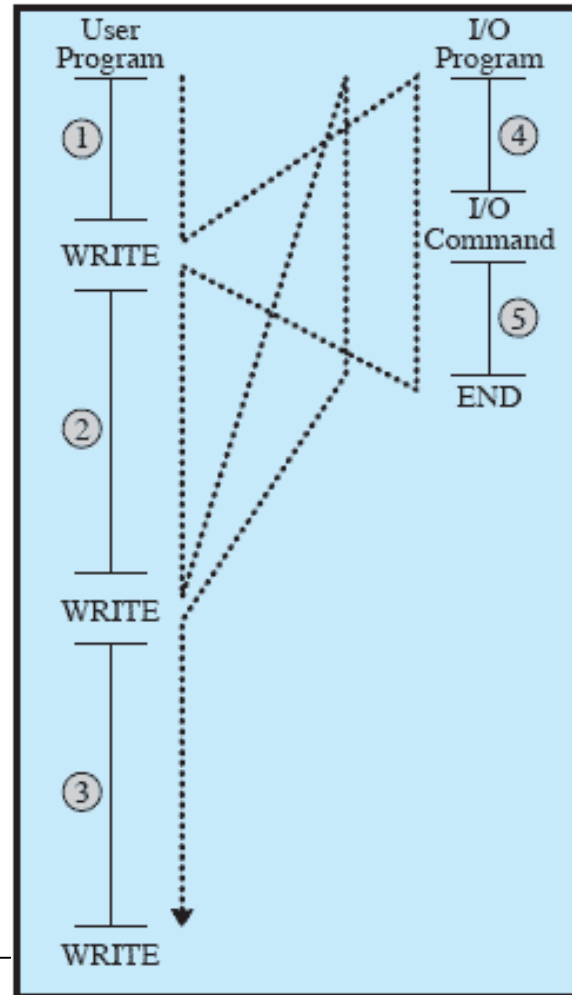
# Common Classes of Interrupts

**Table 1.1** Classes of Interrupts

<b>Program</b>	Generated by some condition that occurs as a result of an instruction execution, such as arithmetic overflow, division by zero, attempt to execute an illegal machine instruction, and reference outside a user's allowed memory space.
<b>Timer</b>	Generated by a timer within the processor. This allows the operating system to perform certain functions on a regular basis.
<b>I/O</b>	Generated by an I/O controller, to signal normal completion of an operation or to signal a variety of error conditions.
<b>Hardware failure</b>	Generated by a failure, such as power failure or memory parity error.

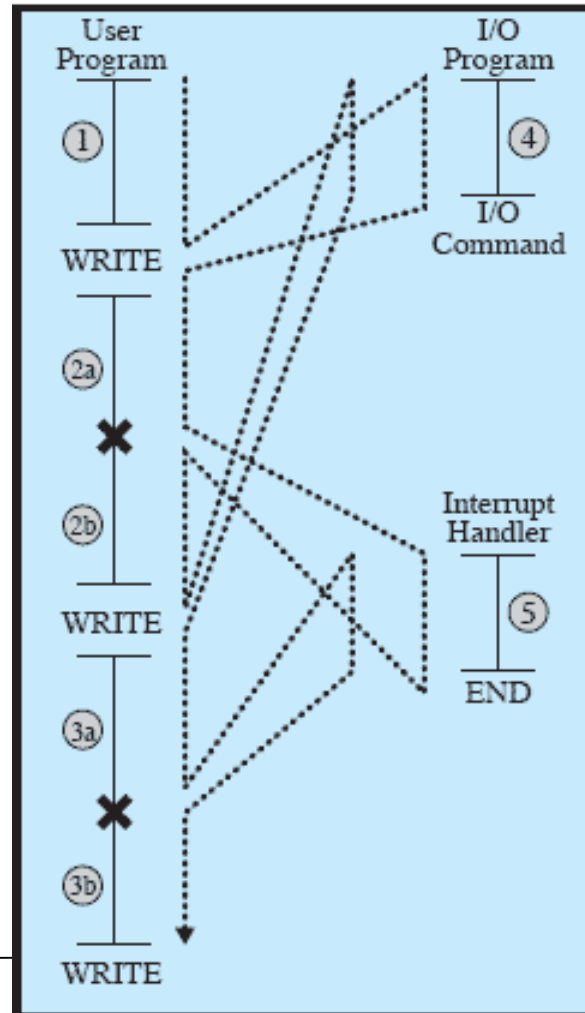


# Flow of Control without Interrupts



(a) No interrupts

# Interrupts and the Instruction Cycle



(b) Interrupts; short I/O wait

# Transfer of Control via Interrupts

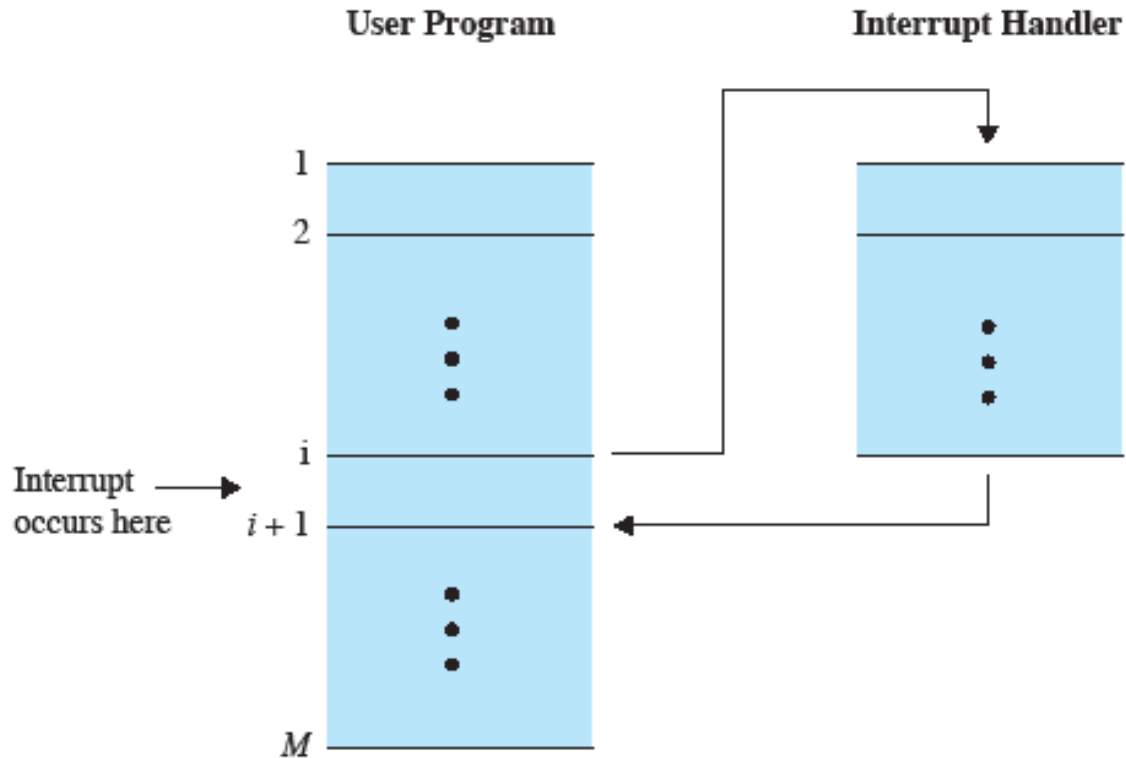
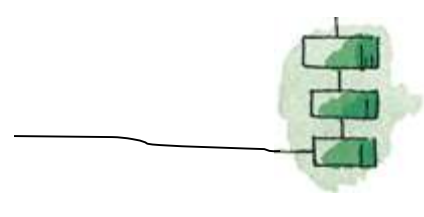
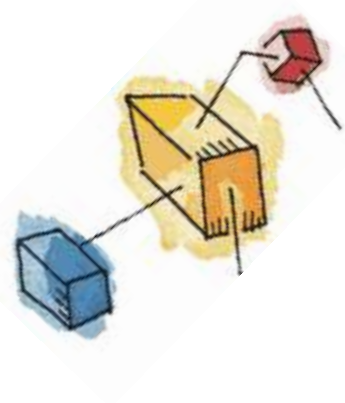


Figure 1.6 Transfer of Control via Interrupts





# Instruction Cycle with Interrupts

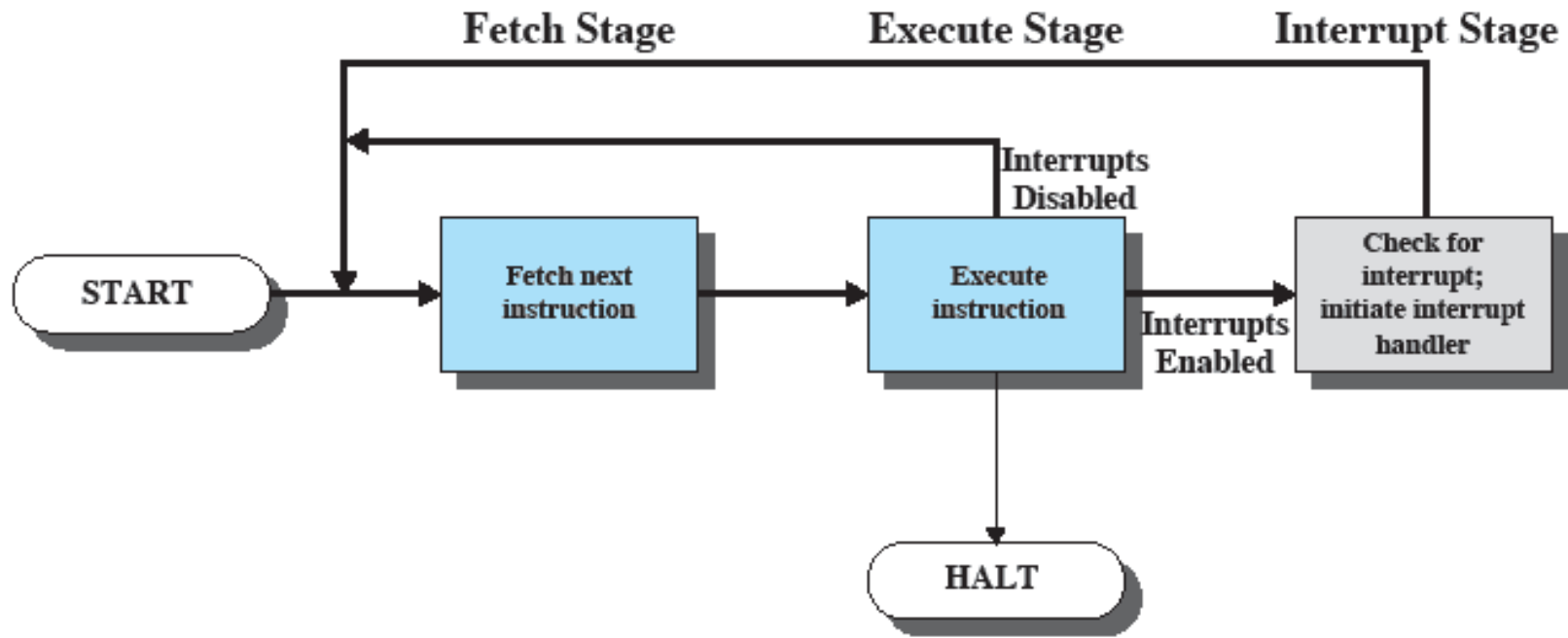
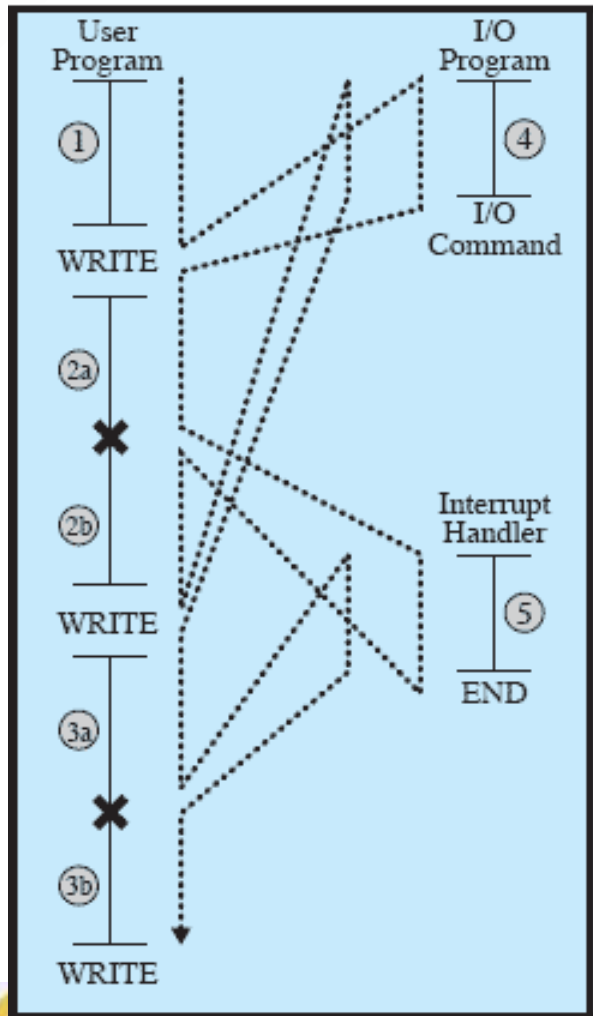
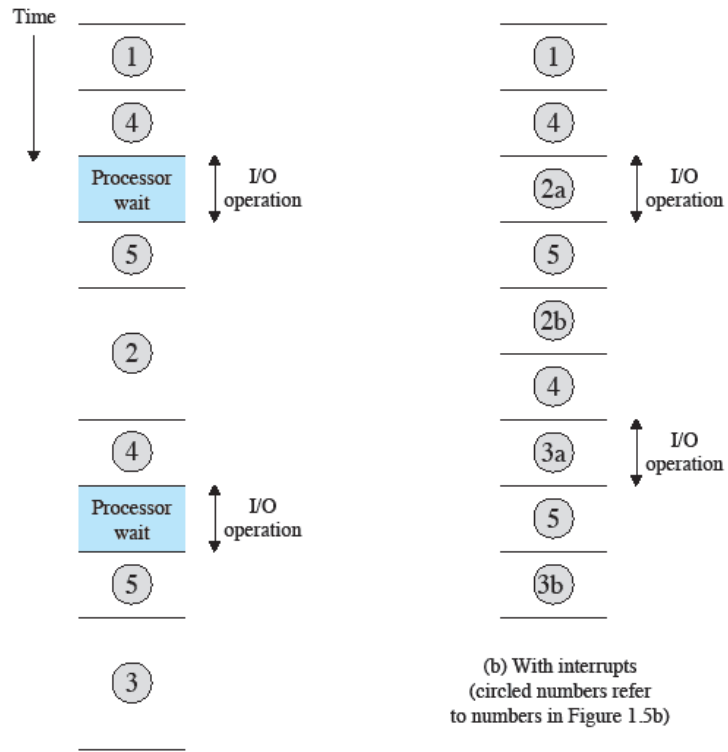


Figure 1.7 Instruction Cycle with Interrupts

# Short I/O Wait



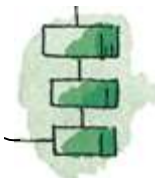
(b) Interrupts; short I/O wait



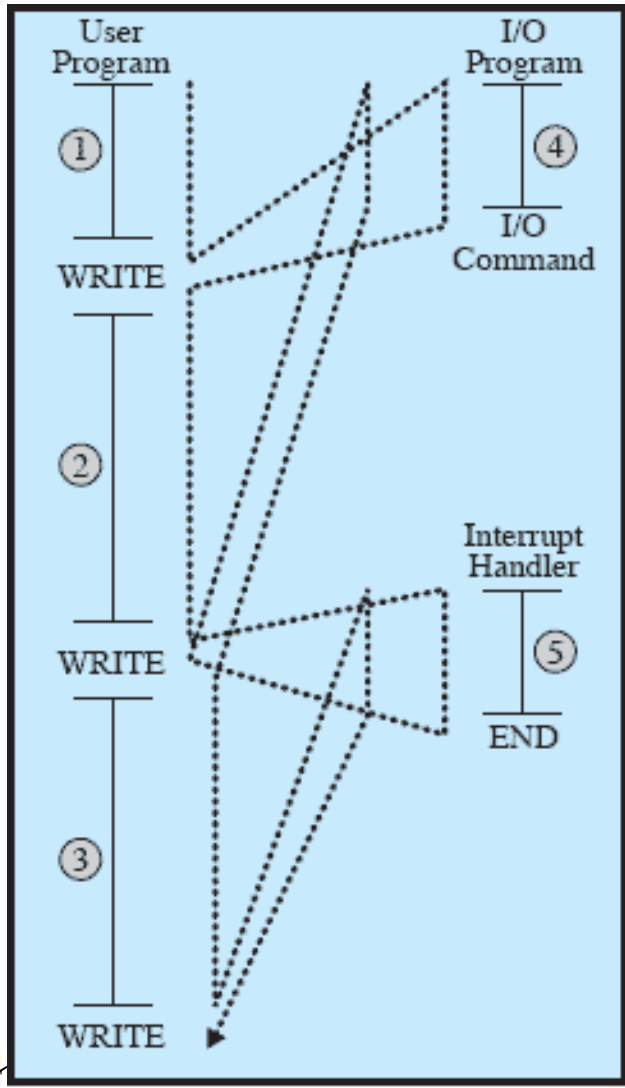
(a) Without interrupts  
(circled numbers refer to numbers in Figure 1.5a)

(b) With interrupts  
(circled numbers refer to numbers in Figure 1.5b)

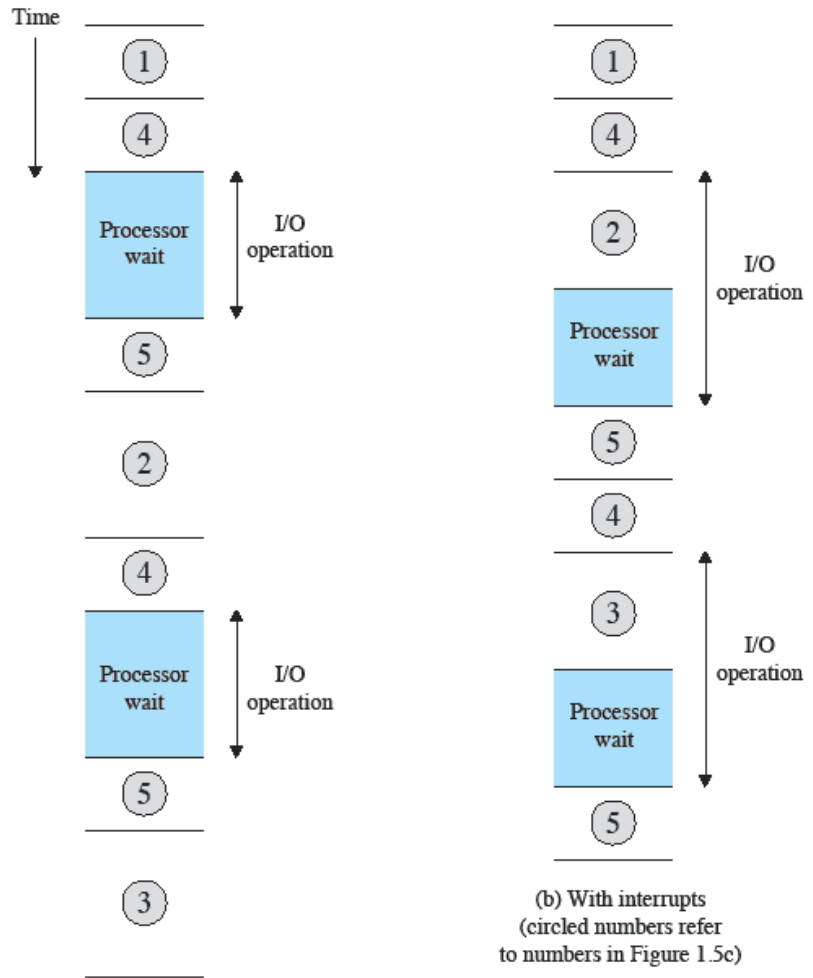
Figure 1.8 Program Timing: Short I/O Wait



# Long I/O wait



(c) Interrupts; long I/O wait



(a) Without interrupts (circled numbers refer to numbers in Figure 1.5a)

(b) With interrupts (circled numbers refer to numbers in Figure 1.5c)

Figure 1.9 Program Timing: Long I/O Wait

# Simple Interrupt Processing

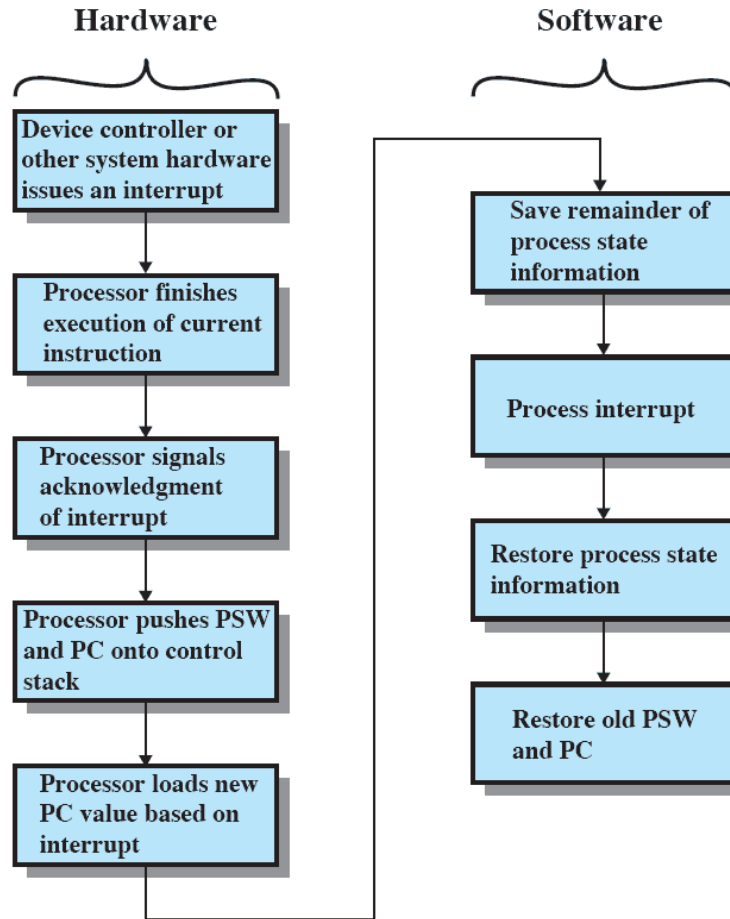
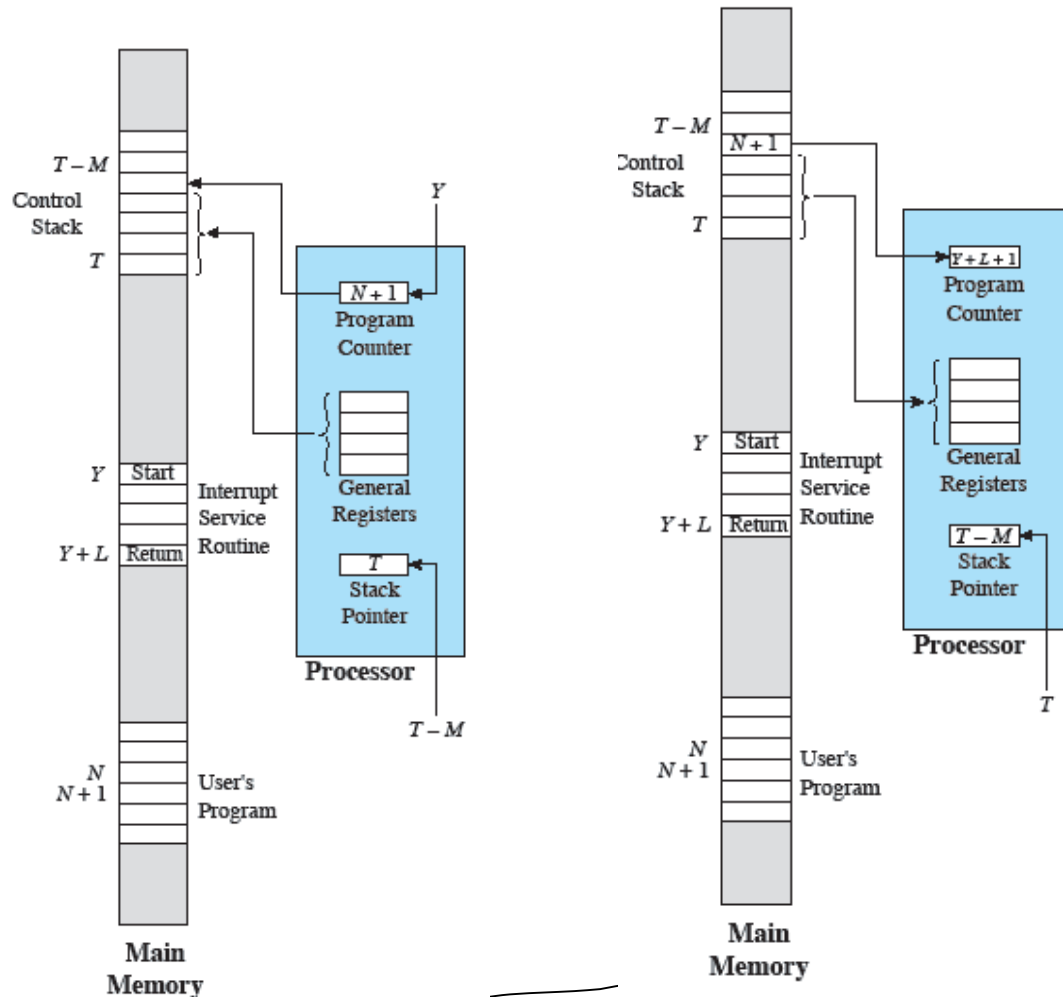


Figure 1.10 Simple Interrupt Processing

# Changes in Memory and Registers for an Interrupt



(a) Interrupt occurs after instruction at location  $N$

(b) Return from interrupt

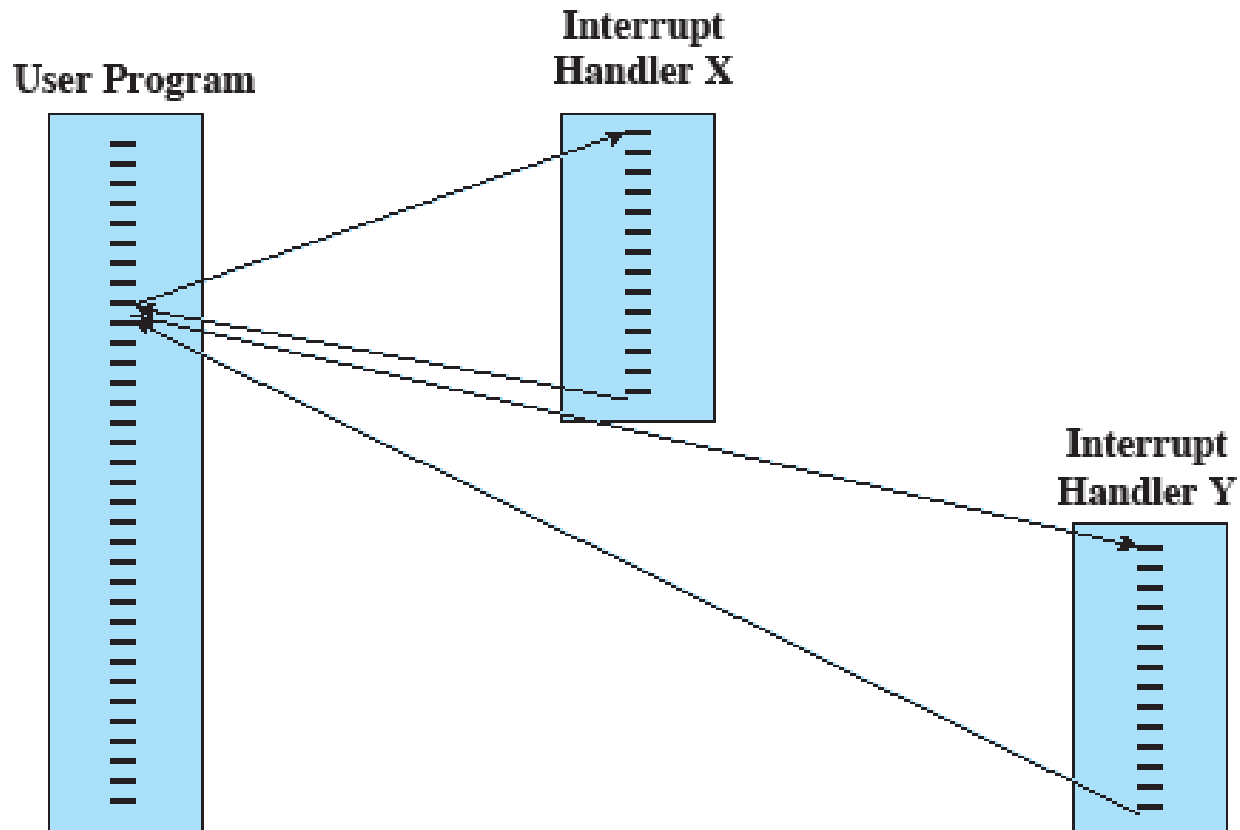


# Multiple Interrupts

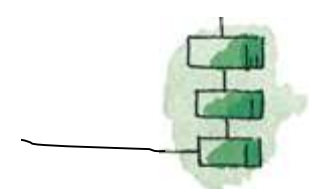
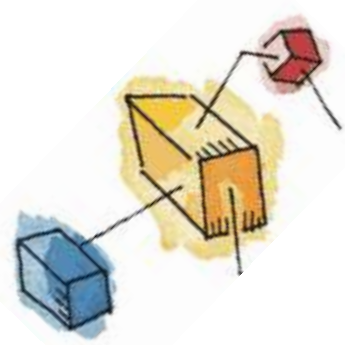
- Suppose an interrupt occurs while another interrupt is being processed.
  - E.g. printing data being received via communications line.
- Two approaches:
  - Disable interrupts during interrupt processing
  - Use a priority scheme.



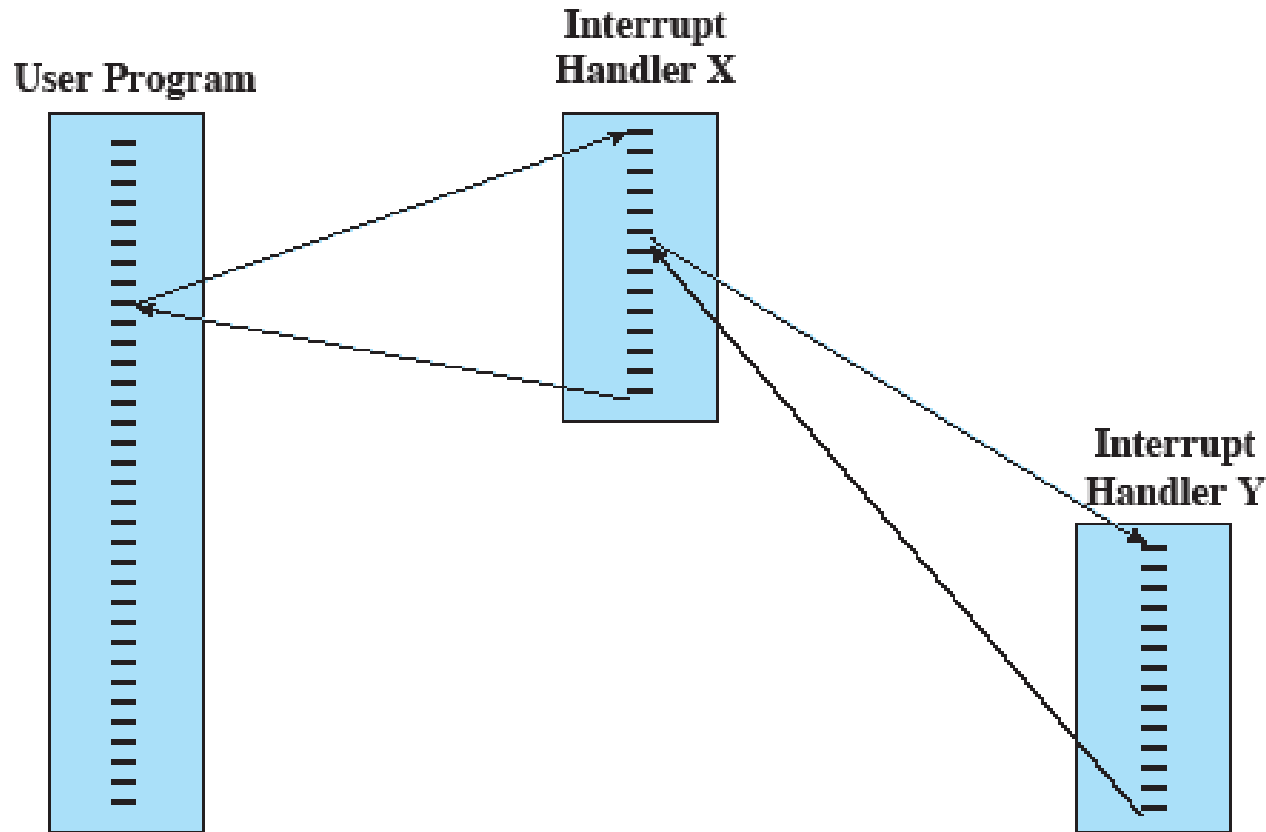
# Sequential Interrupt Processing



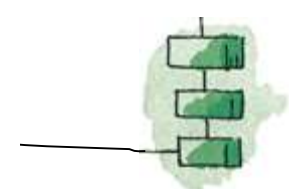
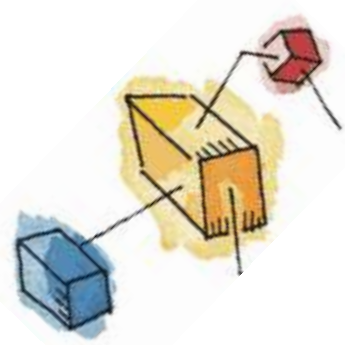
(a) Sequential interrupt processing



# Nested Interrupt Processing

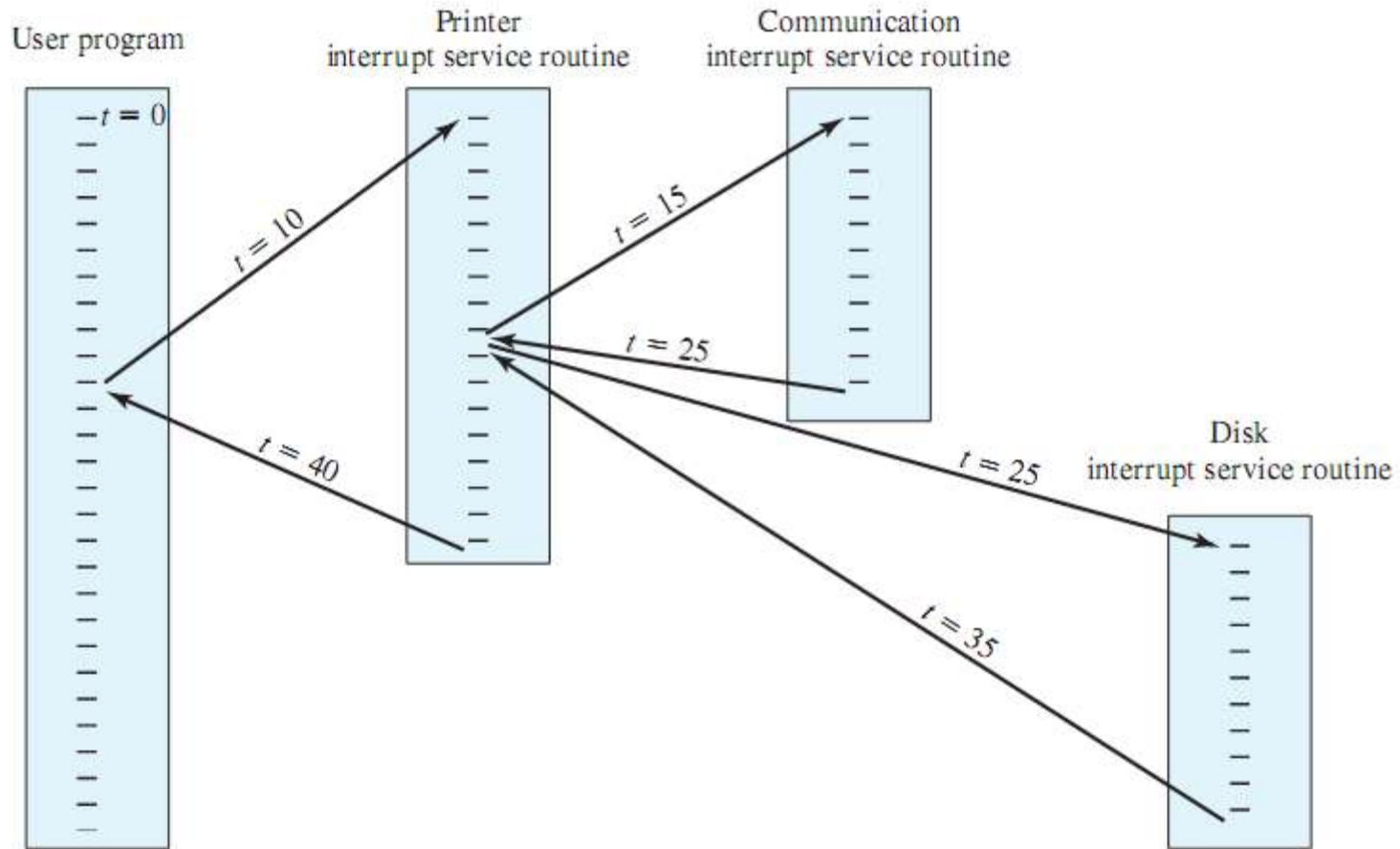


(b) Nested interrupt processing





# Example of Nested Interrupts



**Figure 1.13** Example Time Sequence of Multiple Interrupts



# Multiprogramming

- Processor has more than one program to execute
- The sequence the programs are executed depend on their relative priority and whether they are waiting for I/O
- After an interrupt handler completes, control may not return to the program that was executing at the time of the interrupt

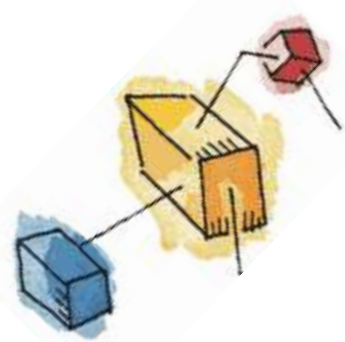


# Roadmap

- Basic Elements
- Processor Registers
- Instruction Execution
- Interrupts

## → The Memory Hierarchy

- Cache Memory
- I/O Communication Techniques





# Memory Hierarchy

- Major constraints in memory
  - Amount
  - Speed
  - Expense
- Faster access time, greater cost per bit
- Greater capacity, smaller cost per bit
- Greater capacity, slower access speed





# The Memory Hierarchy

- Going down the hierarchy
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
  - Decreasing frequency of access to the memory by the processor

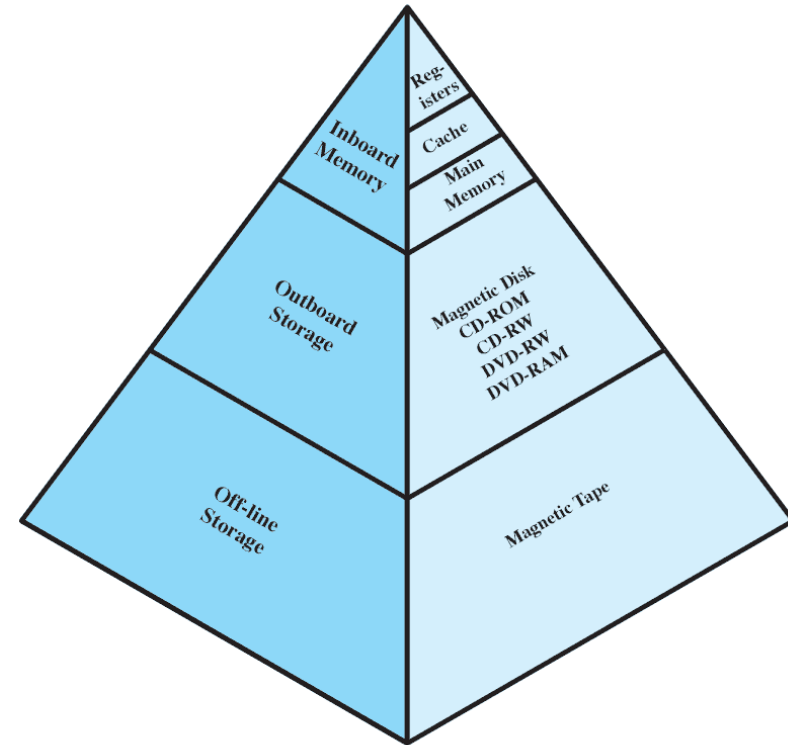
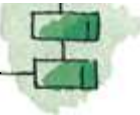
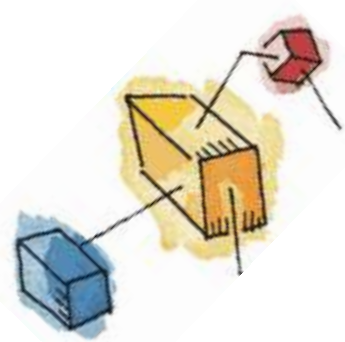


Figure 1.14 The Memory Hierarchy



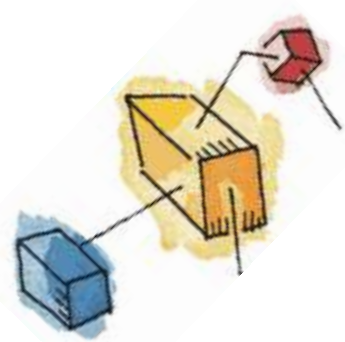
# Secondary Memory

- Auxiliary memory
- External
- Nonvolatile
- Used to store program and data files



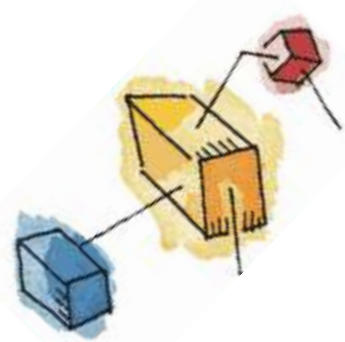
# Roadmap

- Basic Elements
- Processor Registers
- Instruction Execution
- Interrupts
- The Memory Hierarchy
- Cache Memory
- I/O Communication Techniques



# Cache Memory

- Invisible to the OS
  - Interacts with other memory management hardware
- Processor must access memory at least once per instruction cycle
  - Processor speed faster than memory access speed
- Exploit the principle of locality with a small fast memory





# Principal of Locality

- More details later but in short ...
- Data which is required soon is often close to the current data
  - If data is referenced, then it's neighbour might be needed soon.



# Cache and Main Memory

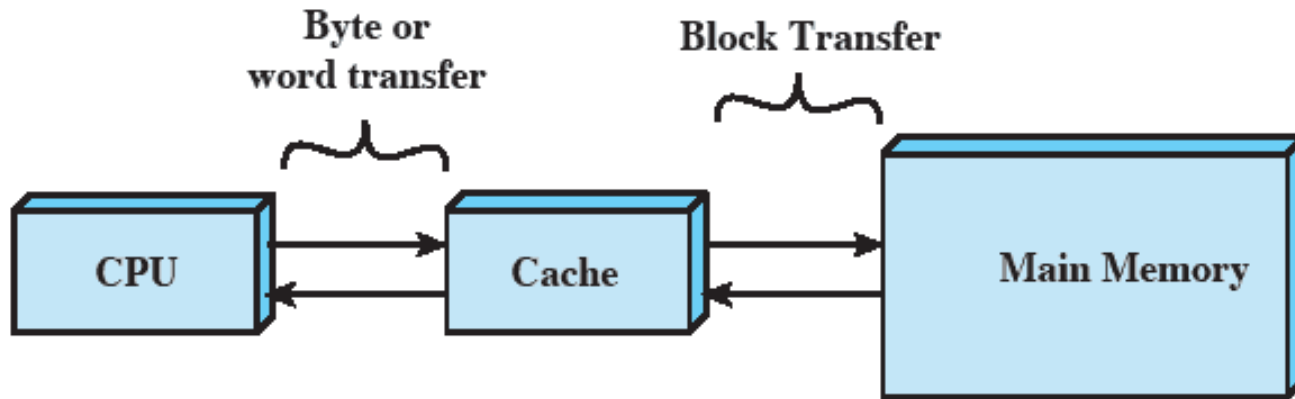
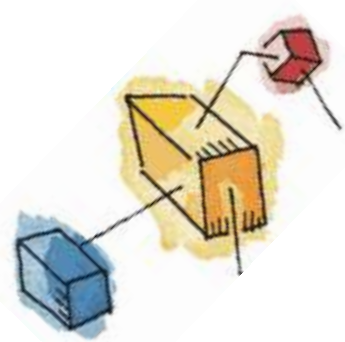


Figure 1.16 Cache and Main Memory



# Cache Principles

- Contains copy of a portion of main memory
- Processor first checks cache
  - If not found, block of memory read into cache
- Because of locality of reference, likely future memory references are in that block



# Cache/Main-Memory Structure

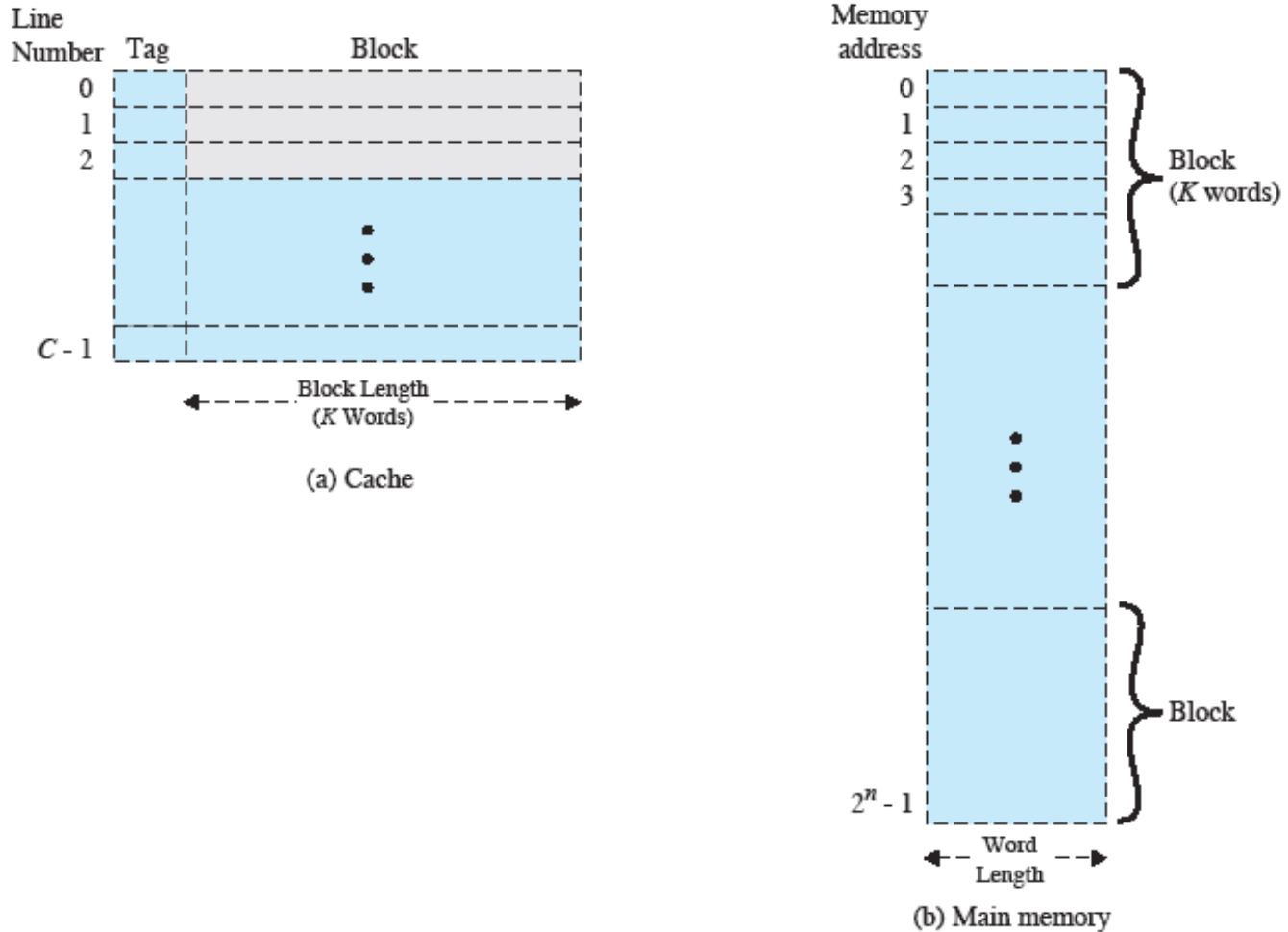


Figure 1.17 Cache/Main-Memory Structure

# Cache Read Operation

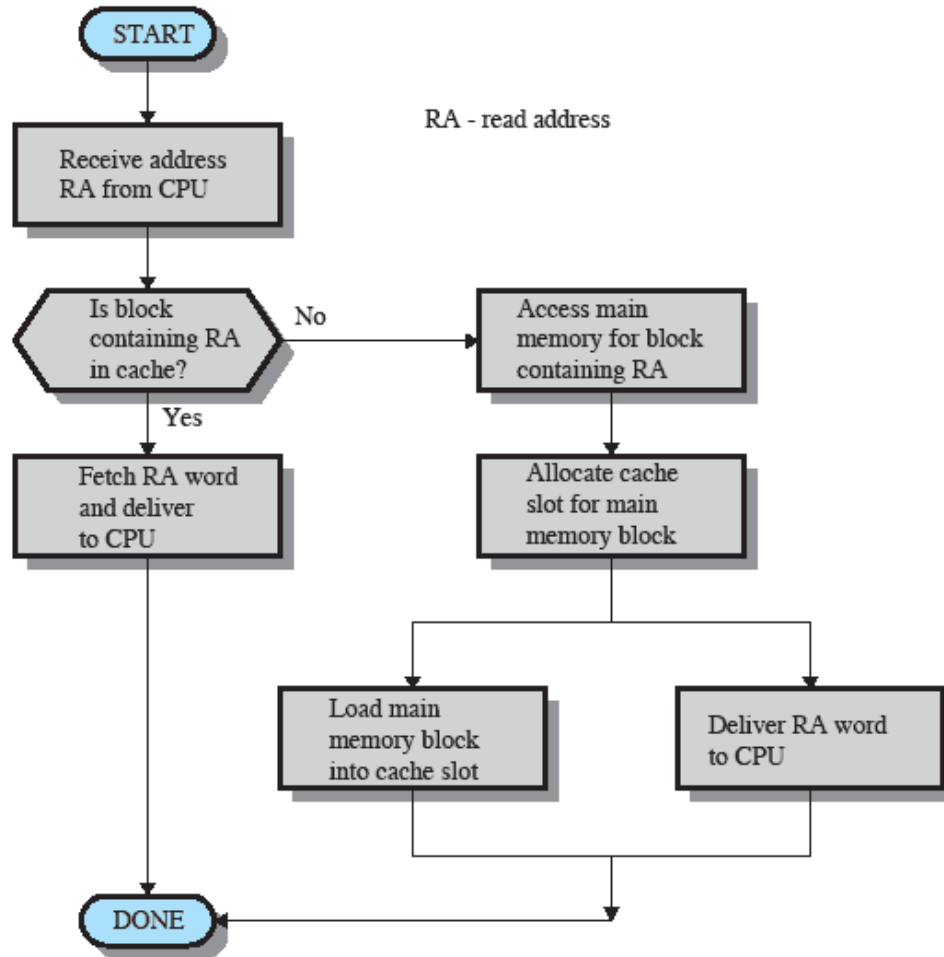
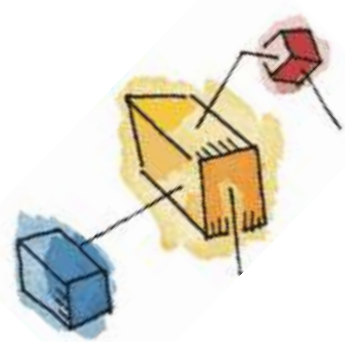


Figure 1.18 Cache Read Operation



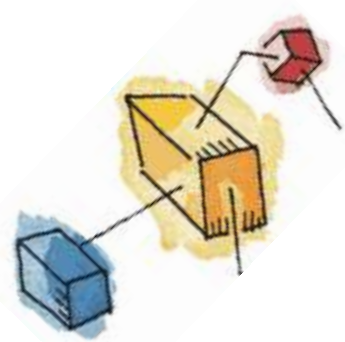
# Cache Design Issues

- Main categories are:
  - Cache size
  - Block size
  - Mapping function
  - Replacement algorithm
  - Write policy



# Size issues

- Cache size
  - Small caches have significant impact on performance
- Block size
  - The unit of data exchanged between cache and main memory
  - Larger block size means more hits
  - But too large reduces chance of reuse.



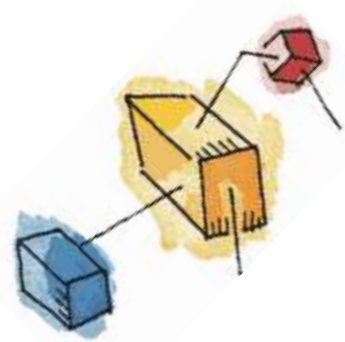


# Mapping function

- Determines which cache location the block will occupy
- Two constraints:
  - When one block read in, another may need replaced
  - Complexity of mapping function increases circuitry costs for searching.







# Replacement Algorithm

- Chooses which block to replace when a new block is to be loaded into the cache.
- Ideally replacing a block that isn't likely to be needed again
  - Impossible to guarantee
- Effective strategy is to replace a block that has been used less than others
  - Least Recently Used (LRU)



# Write policy

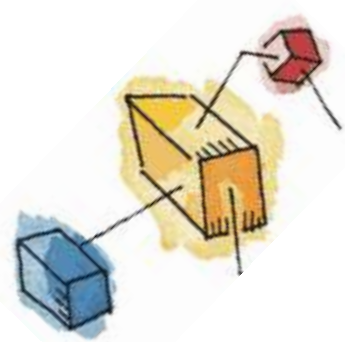
- Dictates when the memory write operation takes place
- Can occur every time the block is updated
- Can occur when the block is replaced
  - Minimize write operations
  - Leave main memory in an obsolete state



# Roadmap

- Basic Elements
- Processor Registers
- Instruction Execution
- Interrupts
- The Memory Hierarchy
- Cache Memory

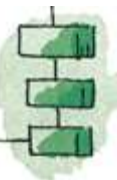
→ I/O Communication Techniques





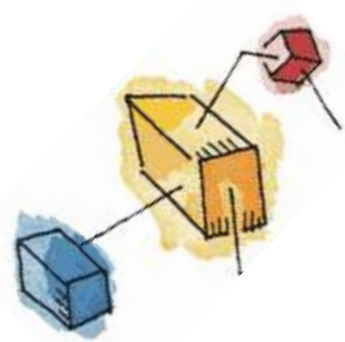
# I/O Techniques

- When the processor encounters an instruction relating to I/O,
  - it executes that instruction by issuing a command to the appropriate I/O module.
- Three techniques are possible for I/O operations:
  - Programmed I/O
  - Interrupt-driven I/O
  - Direct memory access (DMA)



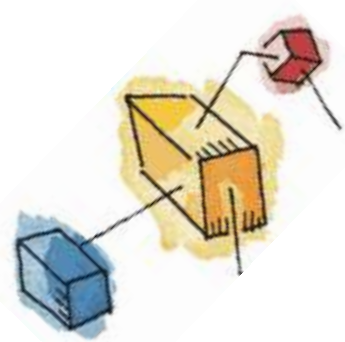
# Programmed I/O

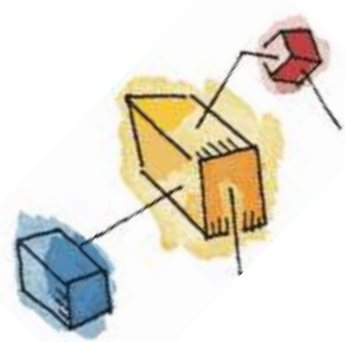
- The I/O module performs the requested action
  - then sets the appropriate bits in the I/O status register
  - but takes no further action to alert the processor.
- As there are no interrupts, the processor must determine when the instruction is complete



# Programmed I/O Instruction Set

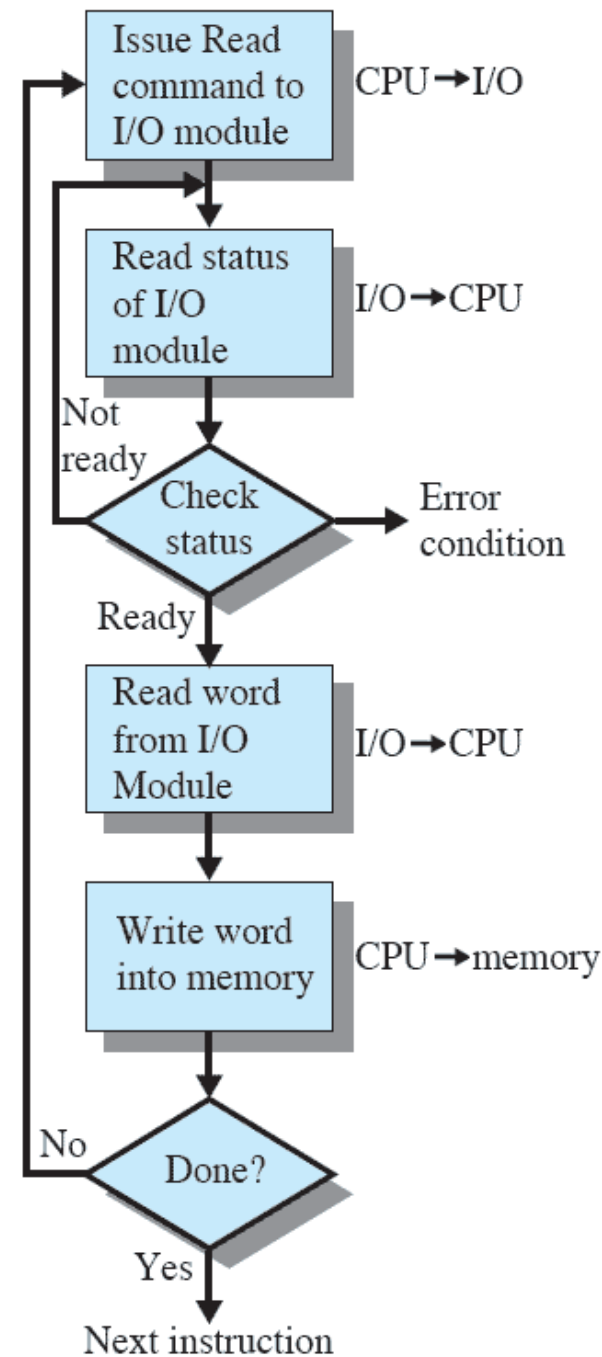
- Control
  - Used to activate and instruct device
- Status
  - Tests status conditions
- Transfer
  - Read/write between process register and device





# Programmed I/O Example

- Data read in a word at a time
  - Processor remains in status-checking look while reading



(a) Programmed I/O



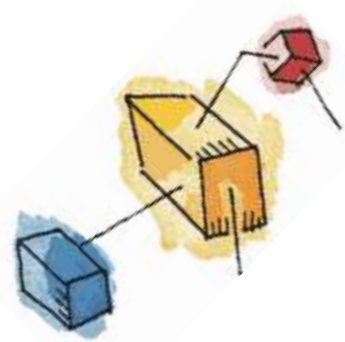


# Interrupt-Driven I/O

- Processor issues an I/O command to a module
  - and then goes on to do some other useful work.
- The I/O module will then interrupt the processor to request service when it is ready to exchange data with the processor.

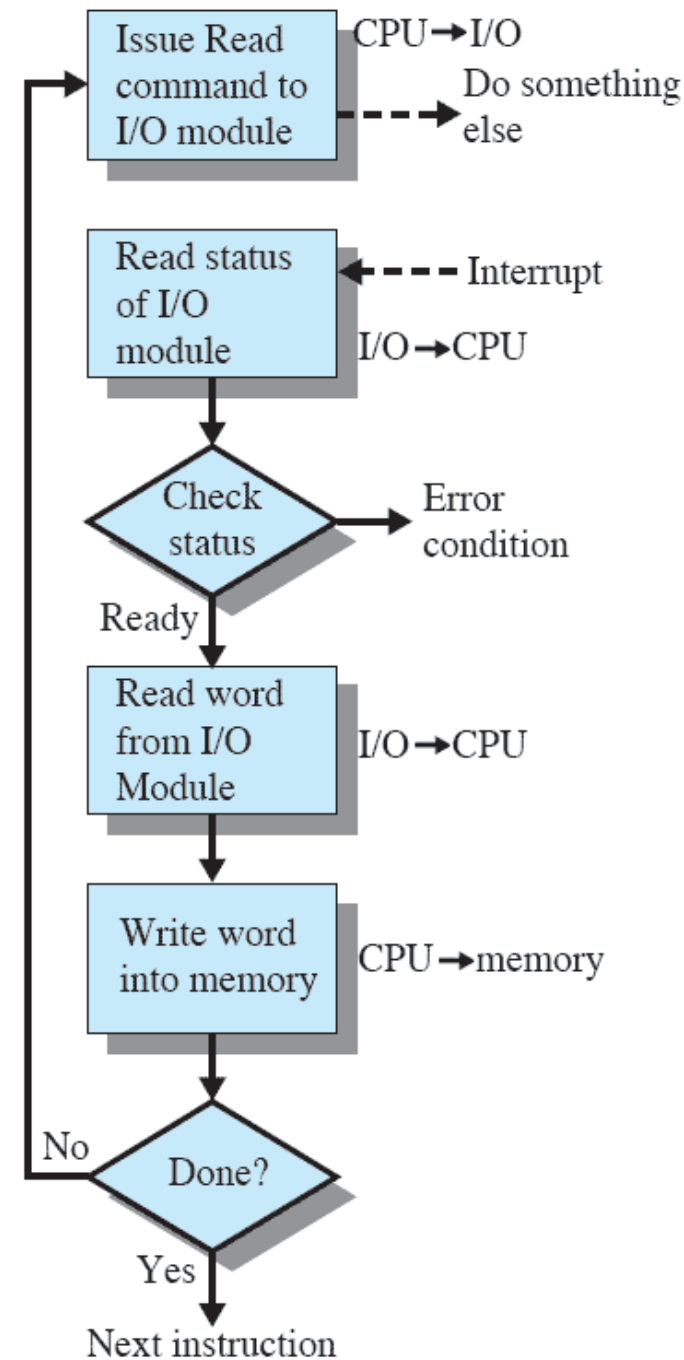






# Interrupt-Driven I/O

- Eliminates needless waiting
  - But everything passes through processor.



(b) Interrupt-driven I/O





# Direct Memory Access

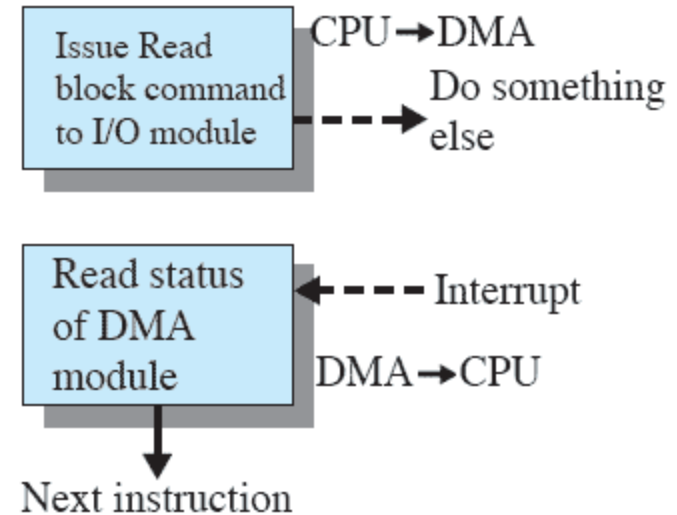
- Performed by a separate module on the system
- When needing to read/write processor issues a command to DMA module with:
  - Whether a read or write is requested
  - The address of the I/O device involved
  - The starting location in memory to read/write
  - The number of words to be read/written





# Direct Memory Access

- I/O operation delegated to DMA module
- Processor only involved when beginning and ending transfer.
- Much more efficient.



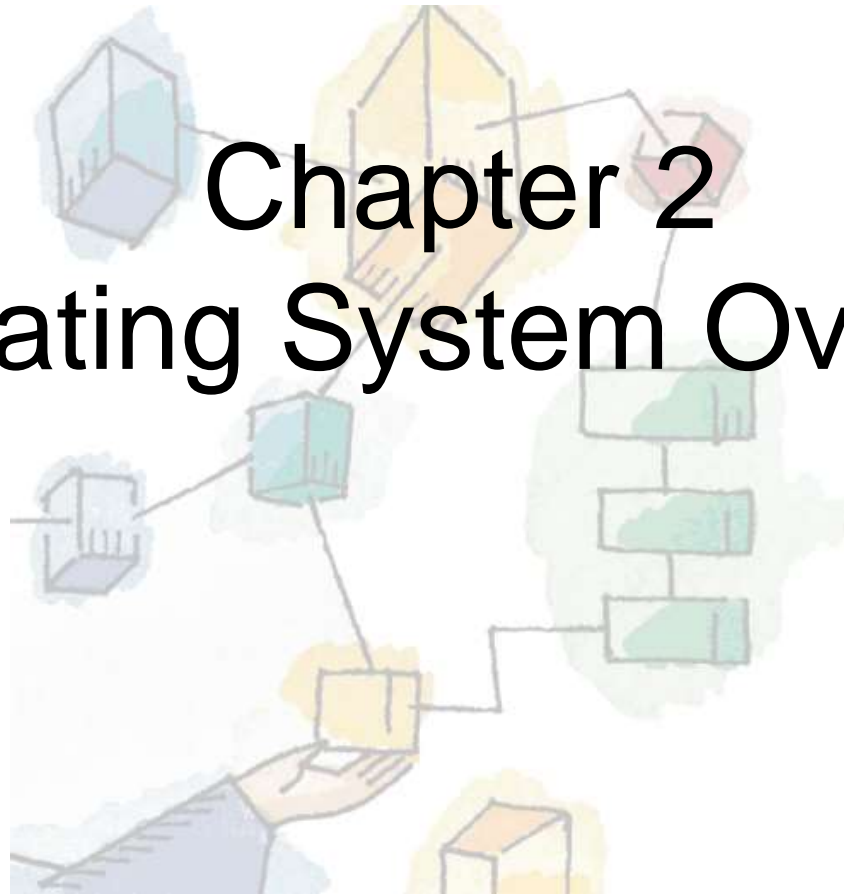
(c) Direct memory access



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

# Chapter 2

# Operating System Overview





# 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

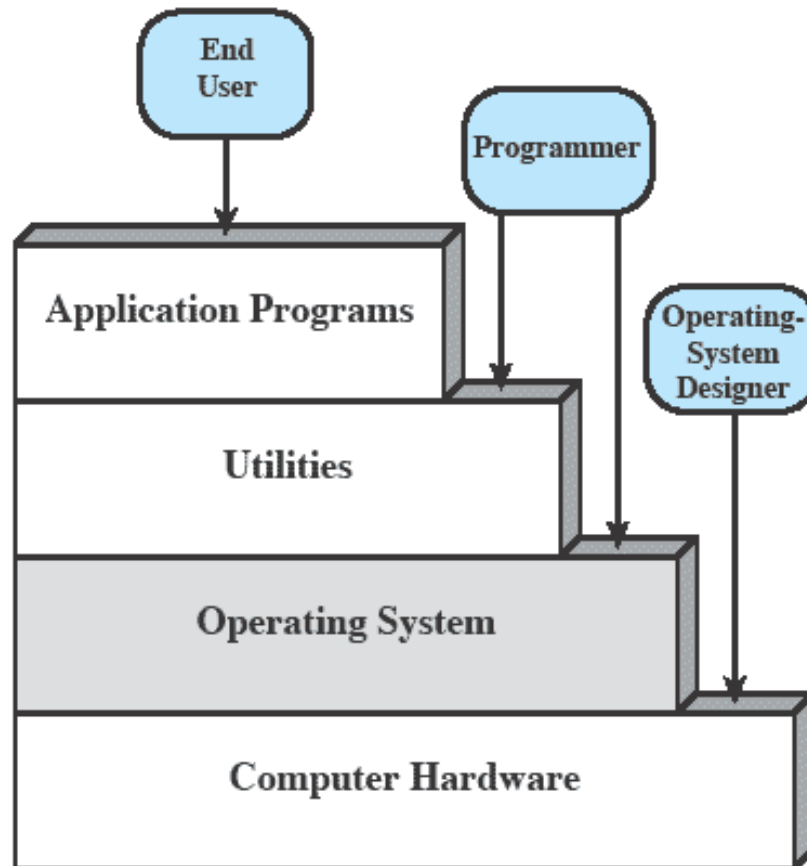


Figure 2.1 Layers and Views of a Computer System



# 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





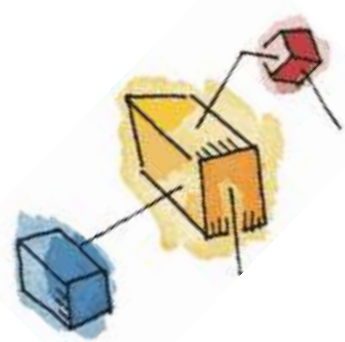
# The Role of an OS

- 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

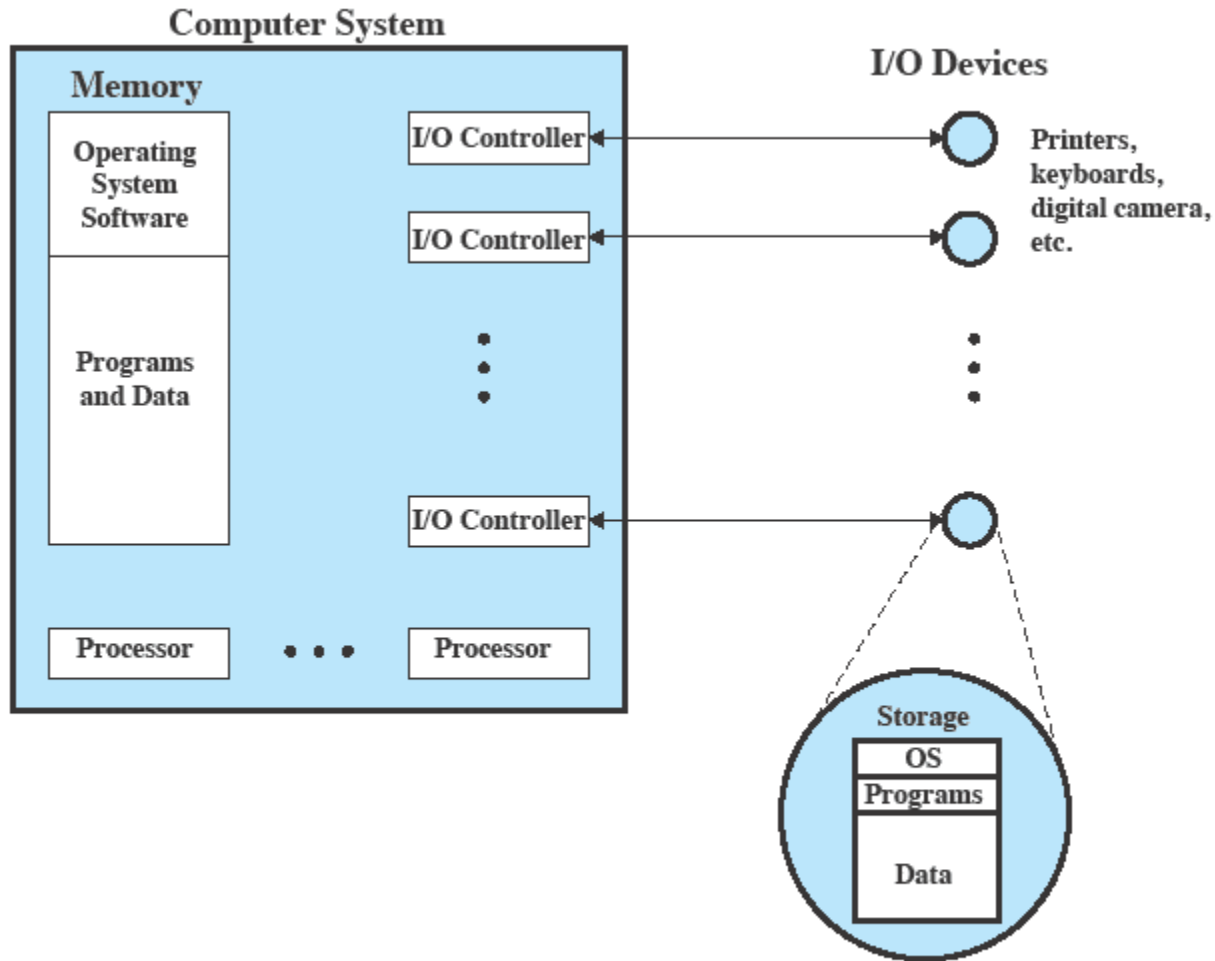
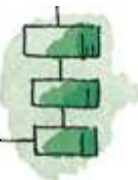
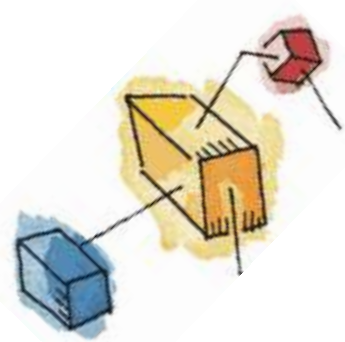


Figure 2.2 The Operating System as Resource Manager

# 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

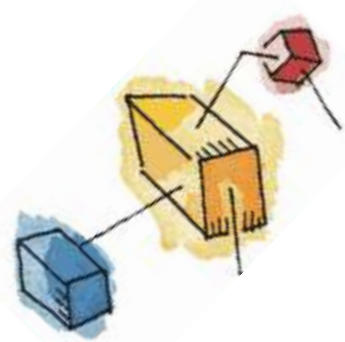
- UNIX Systems

- Linux



# Evolution of Operating Systems

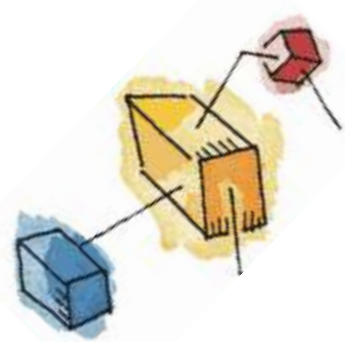
- It may be easier to understand the key requirements of an OS by considering the evolution of Operating Systems
- Stages include
  - Serial Processing
  - Simple Batch Systems
  - Multiprogrammed batch systems
  - Time Sharing Systems





# Serial Processing

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

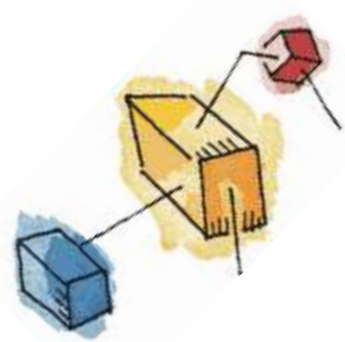




# Simple batch system

- 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

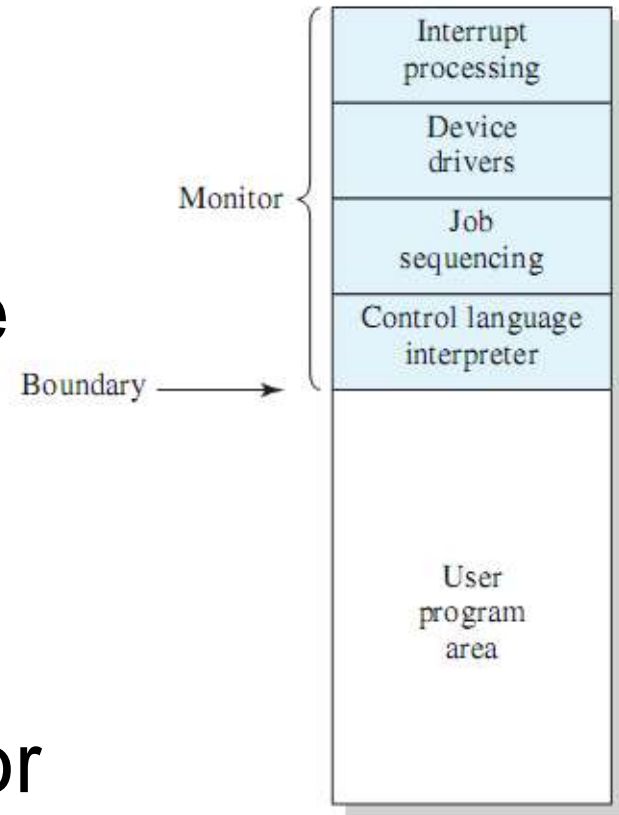
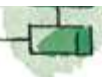


Figure 2.3 Memory Layout for a Resident Monitor





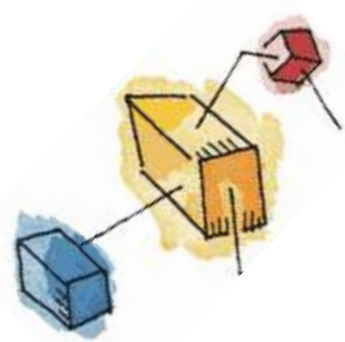
# Job Control Language

- 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

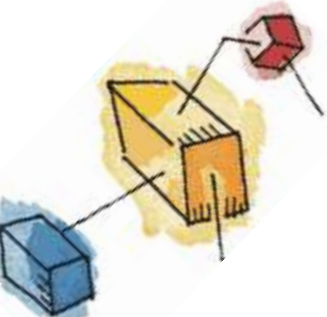




# Modes of Operation

- 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 $\mu$ s
Execute 100 instructions	1 $\mu$ s
Write one record to file	<u>15 <math>\mu</math>s</u>
TOTAL	31 $\mu$ s

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

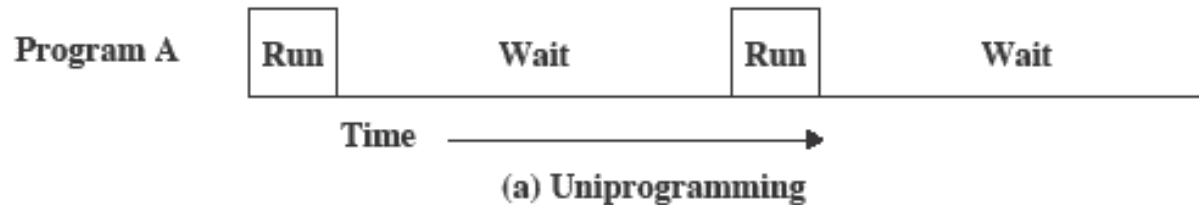


**Figure 2.4 System Utilization Example**



# Uniprogramming

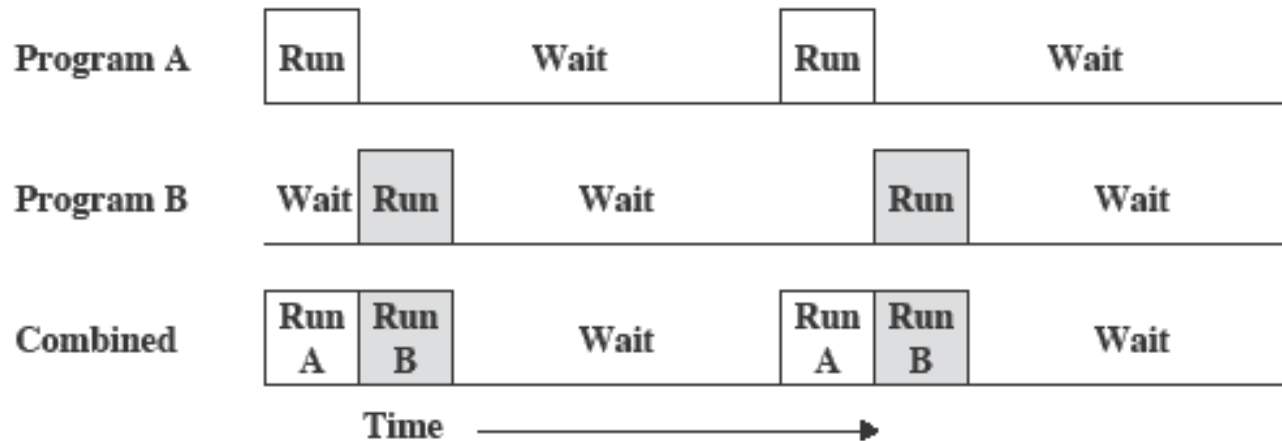
- Processor must wait for I/O instruction to complete before proceeding



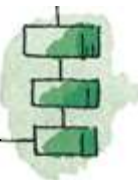


# Multiprogramming

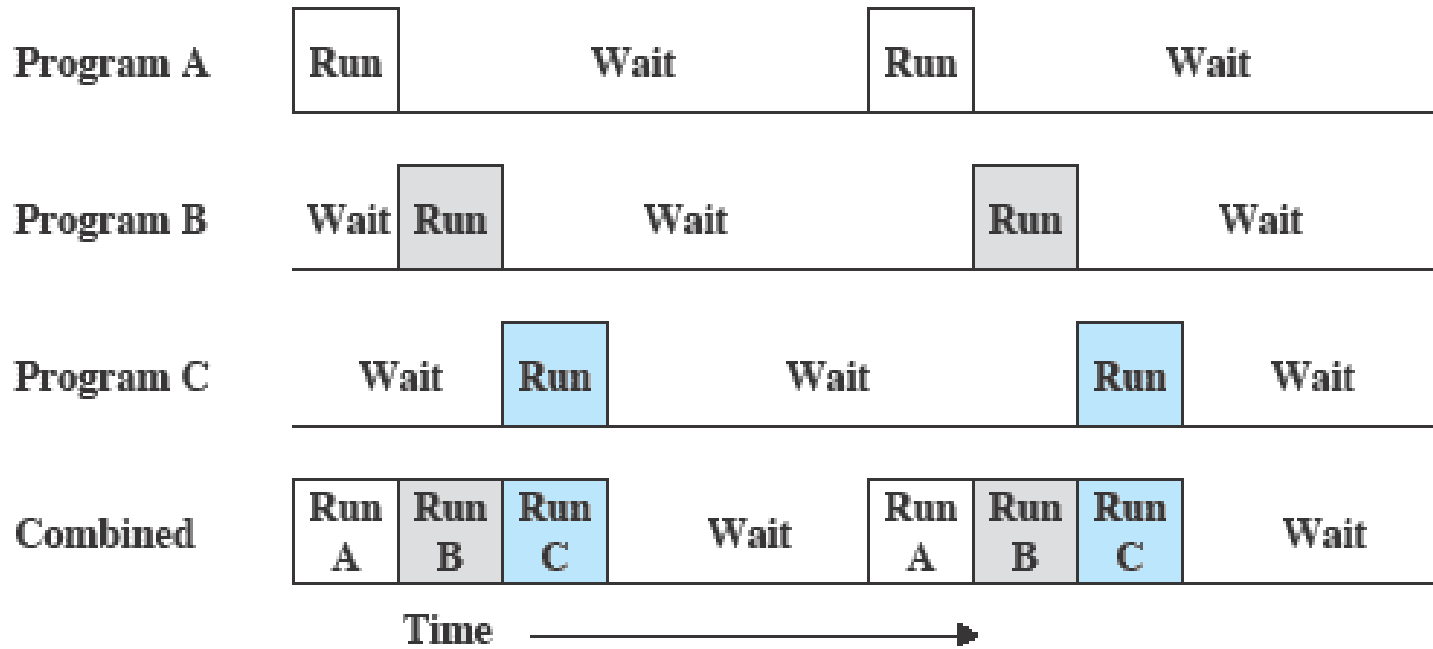
- When one job needs to wait for I/O, the processor can switch to the other job



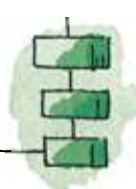
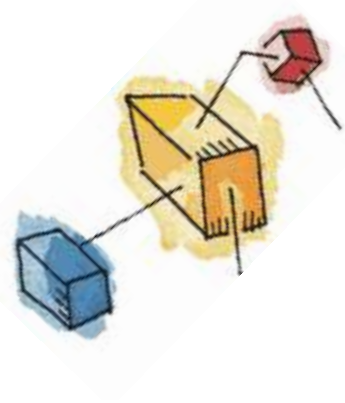
(b) Multiprogramming with two programs



# Multiprogramming



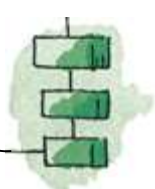
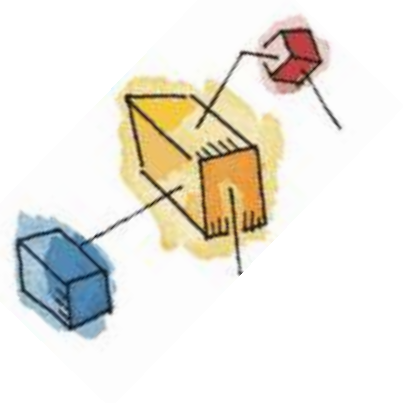
(c) Multiprogramming with three programs



# Example

**Table 2.1 Sample Program Execution Attributes**

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



# Utilization Histograms

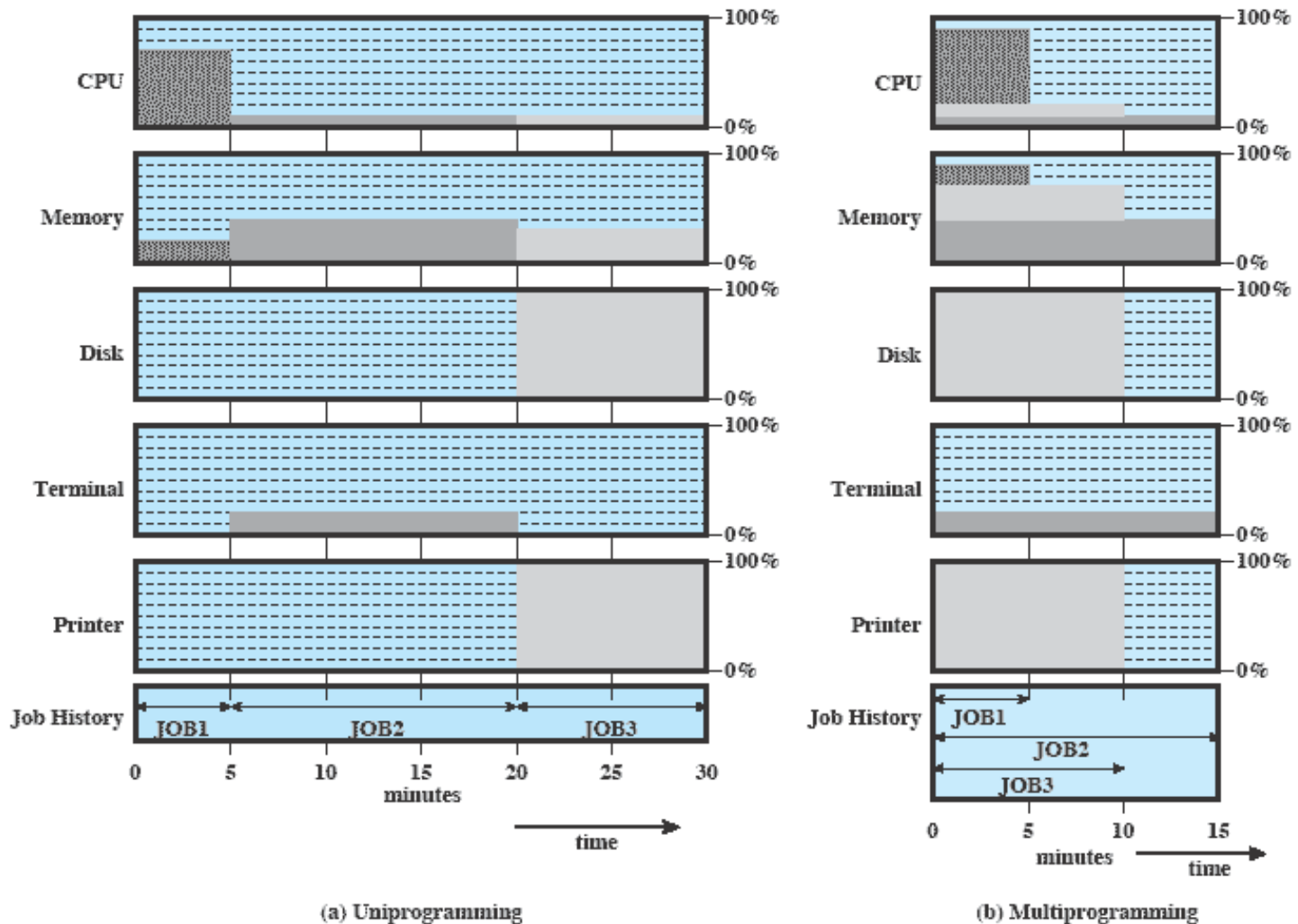
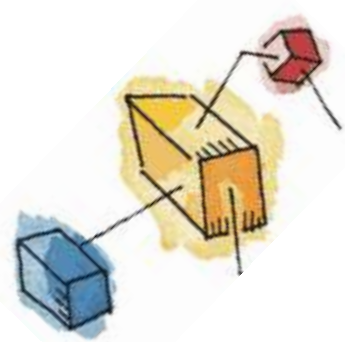


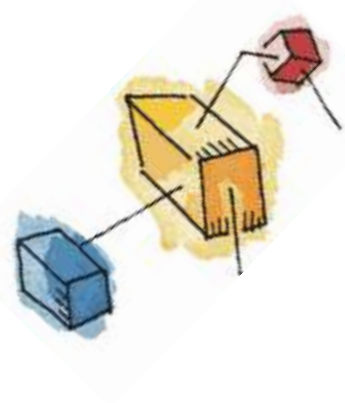
Figure 2.6 Utilization Histograms



# 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

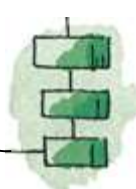


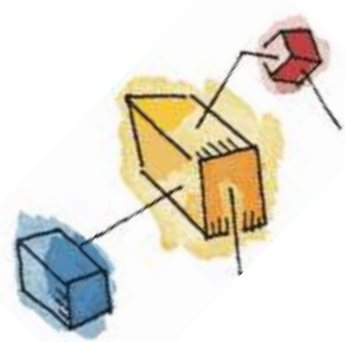


# Batch Multiprogramming vs. Time Sharing

**Table 2.3 Batch Multiprogramming versus Time Sharing**

	<b>Batch Multiprogramming</b>	<b>Time Sharing</b>
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

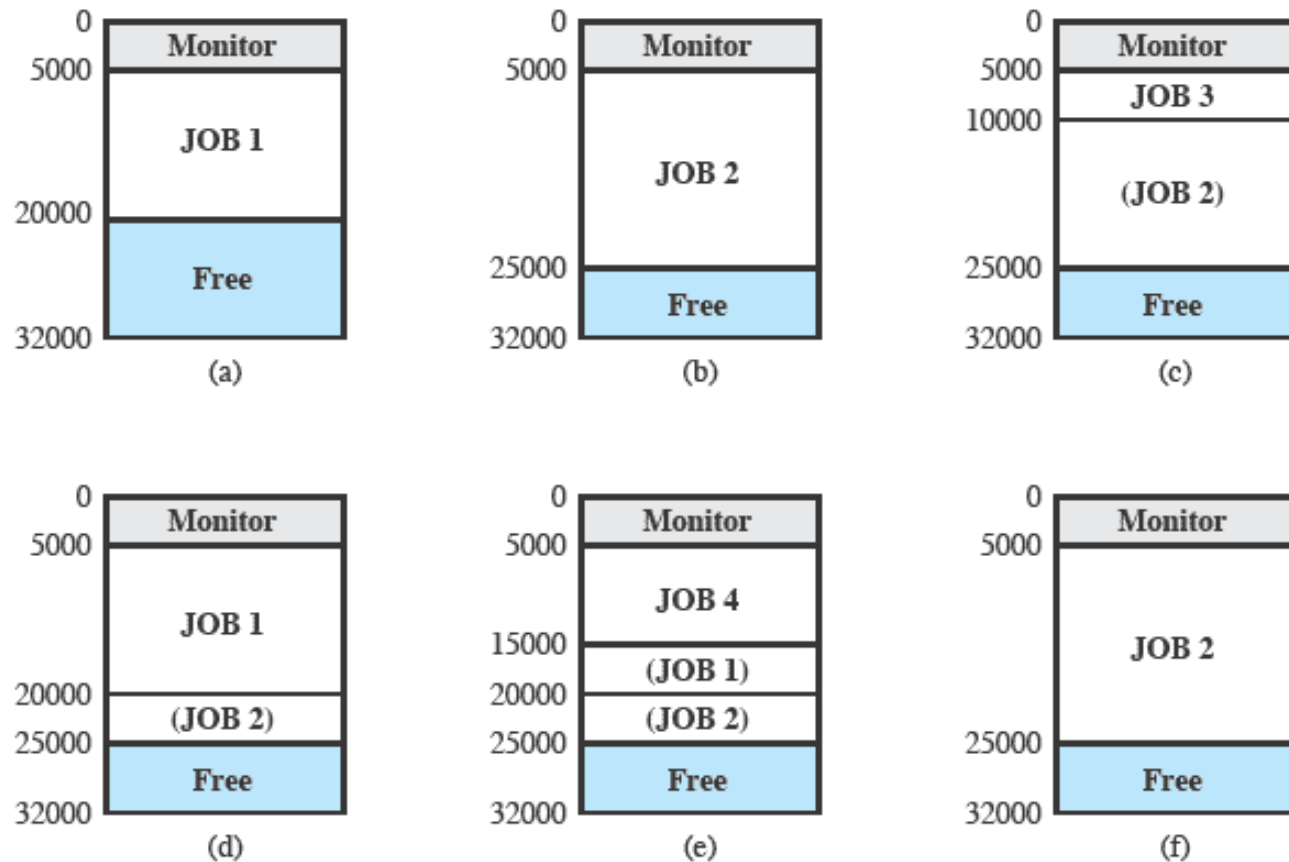
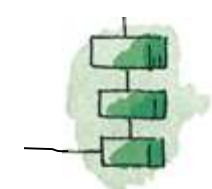
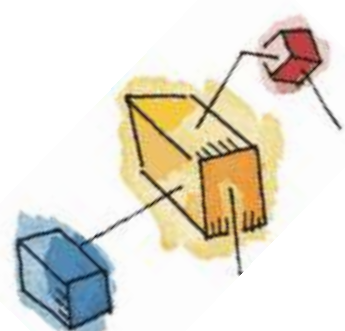


Figure 2.7 CTSS Operation







# 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

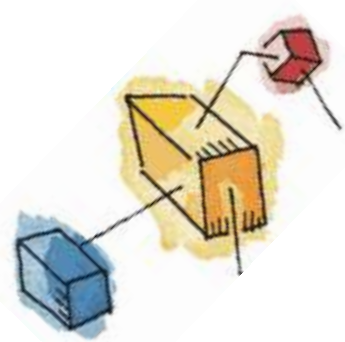
## → Major Achievements

- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems
- Linux



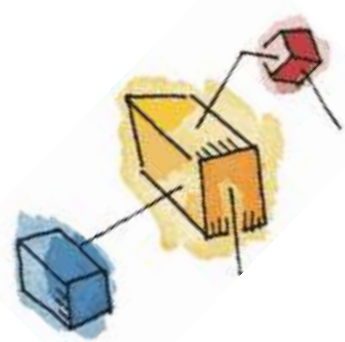
# Major Advances

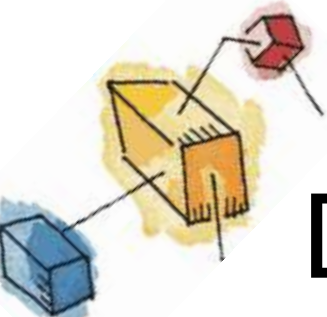
- Operating Systems are among the most complex pieces of software ever developed
- Major advances include:
  - Processes
  - Memory management
  - Information protection and security
  - Scheduling and resource management
  - System



# 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.





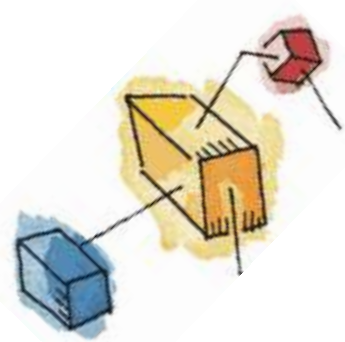
# Causes of Errors when Designing System Software

- 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

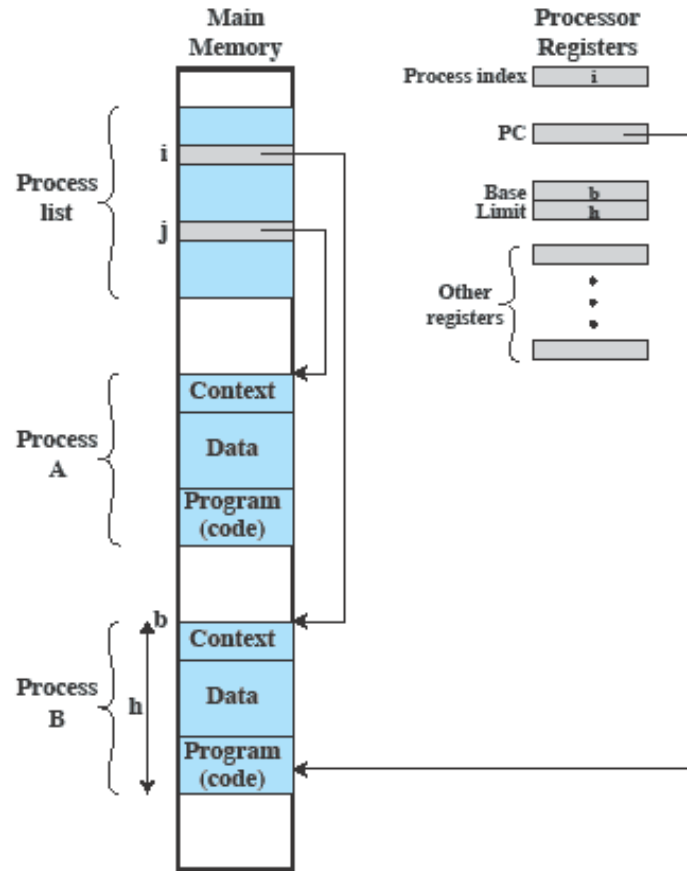
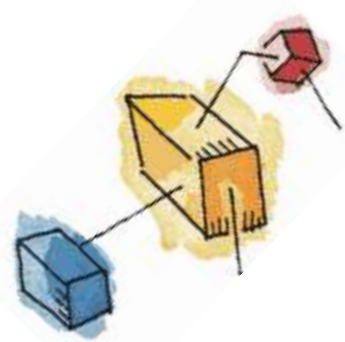


Figure 2.8 Typical Process Implementation



# 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



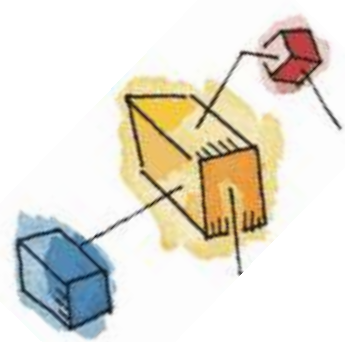




# Virtual Memory

- 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

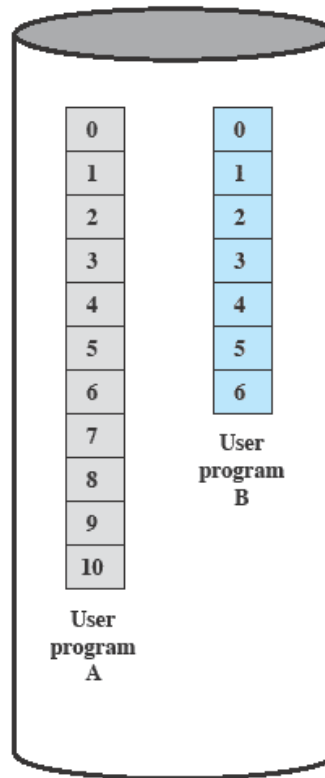


# Virtual Memory

A.1			
	A.0	A.2	
	A.5		
B.0	B.1	B.2	B.3
		A.7	
	A.9		
		A.8	
	B.5	B.6	

Main 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.



Disk

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

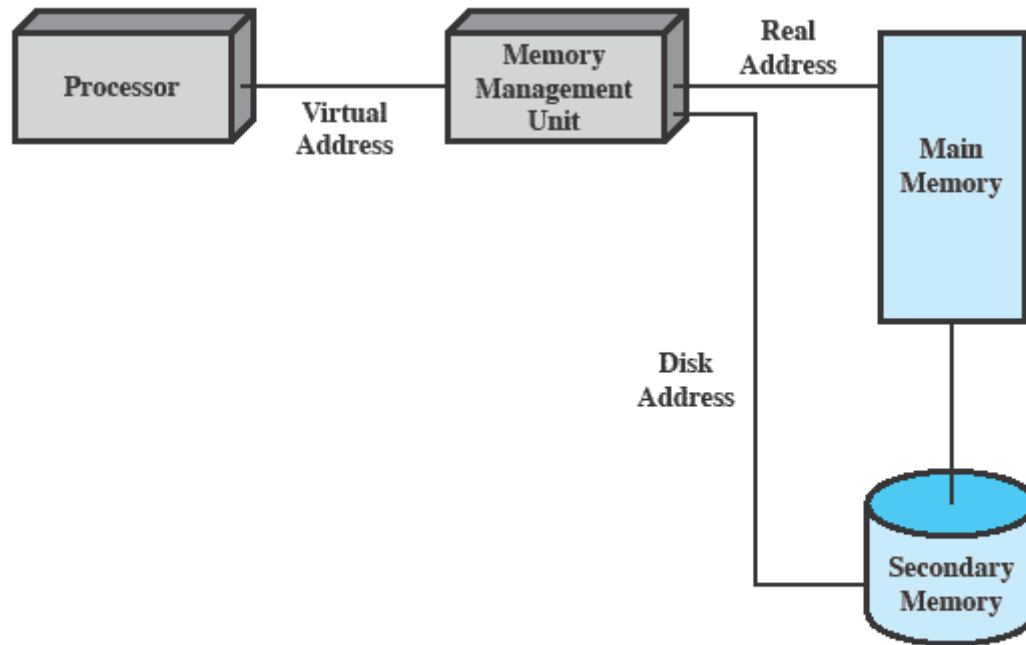
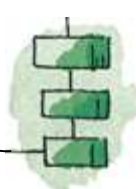
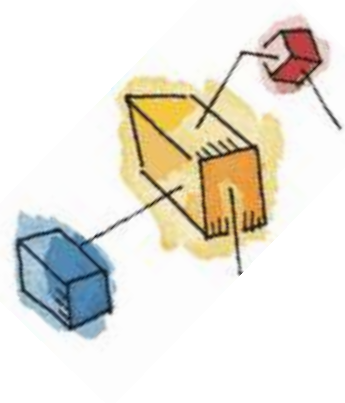
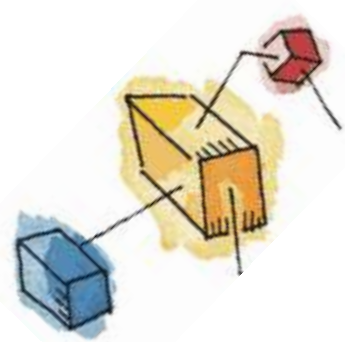


Figure 2.10 Virtual Memory Addressing





# 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

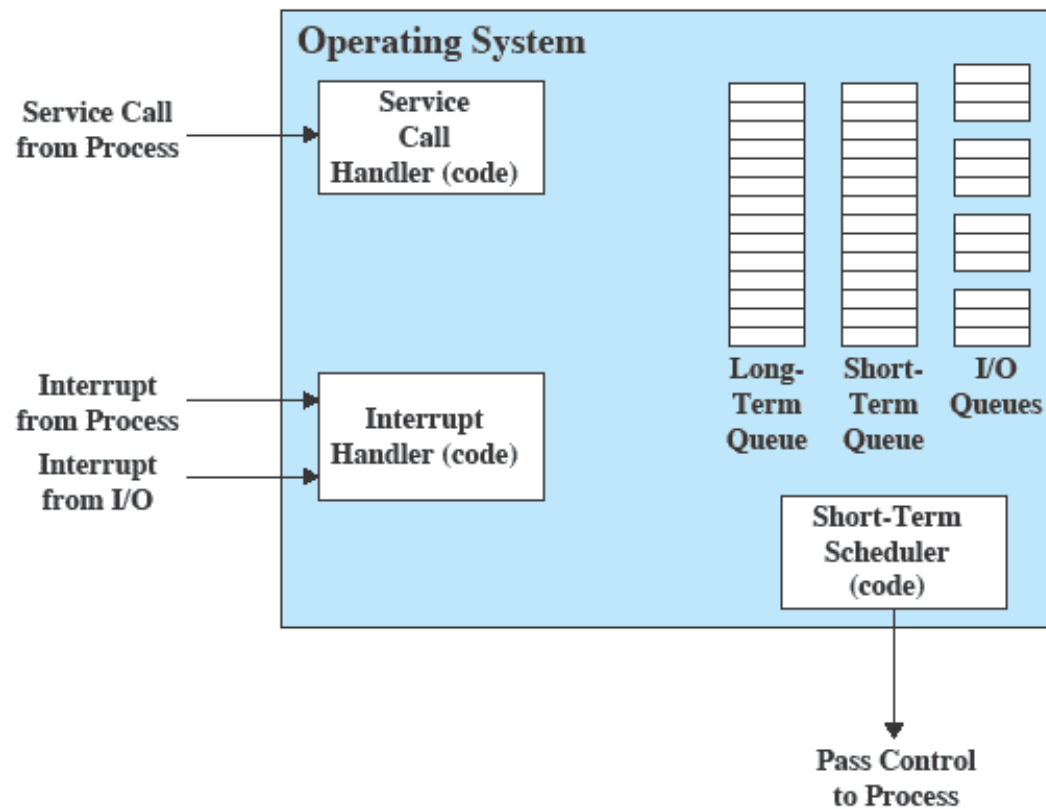
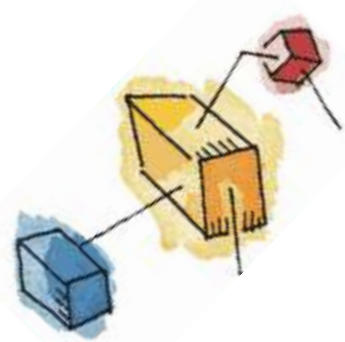


Figure 2.11 Key Elements of an Operating System for Multiprogramming

# 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



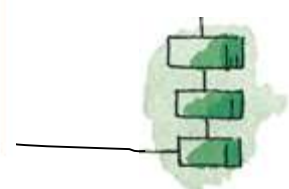
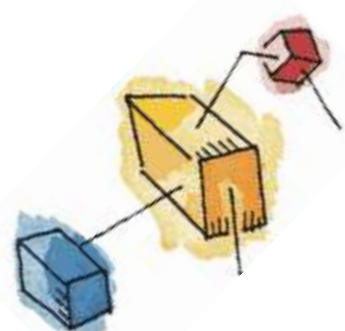


# OS Design Hierarchy

**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.

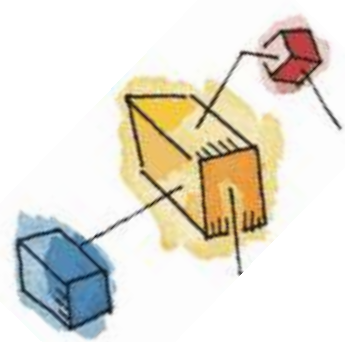


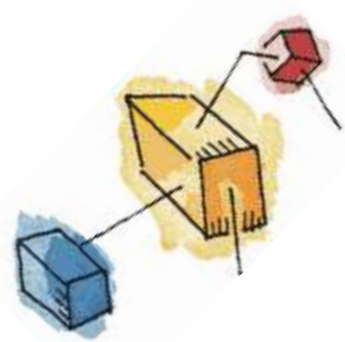
# Roadmap

- 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





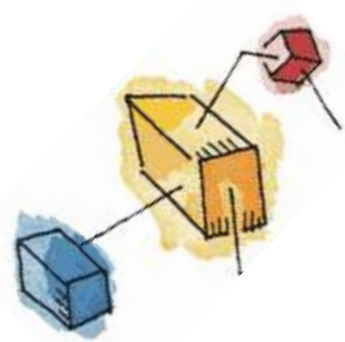
# Microkernel Architecture

- 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



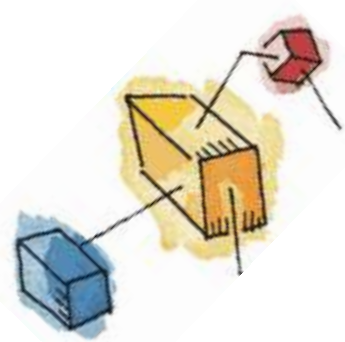
# Multithreading

- 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.





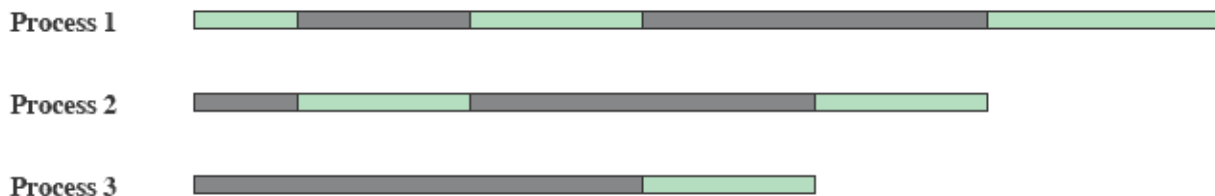
# SMP Advantages

- 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

Time →



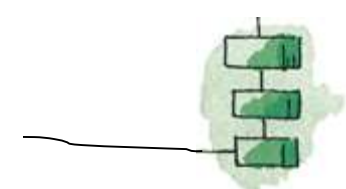
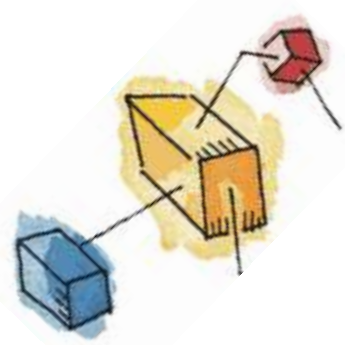
(a) Interleaving (multiprogramming, one processor)



(b) Interleaving and overlapping (multiprocessing; two processors)

■ Blocked    ■ Running

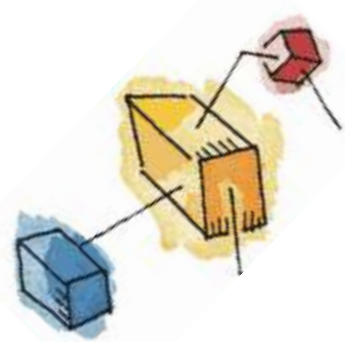
Figure 2.12 Multiprogramming and Multiprocessing

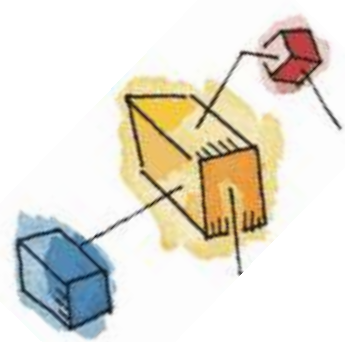




# Distributed Operating Systems

- 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



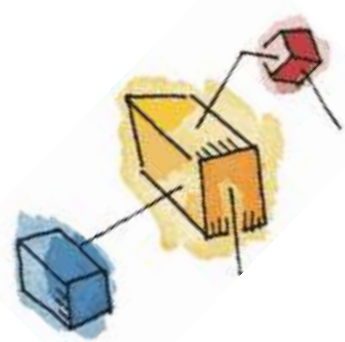
## Microsoft Windows Overview

- UNIX Systems
- Linux

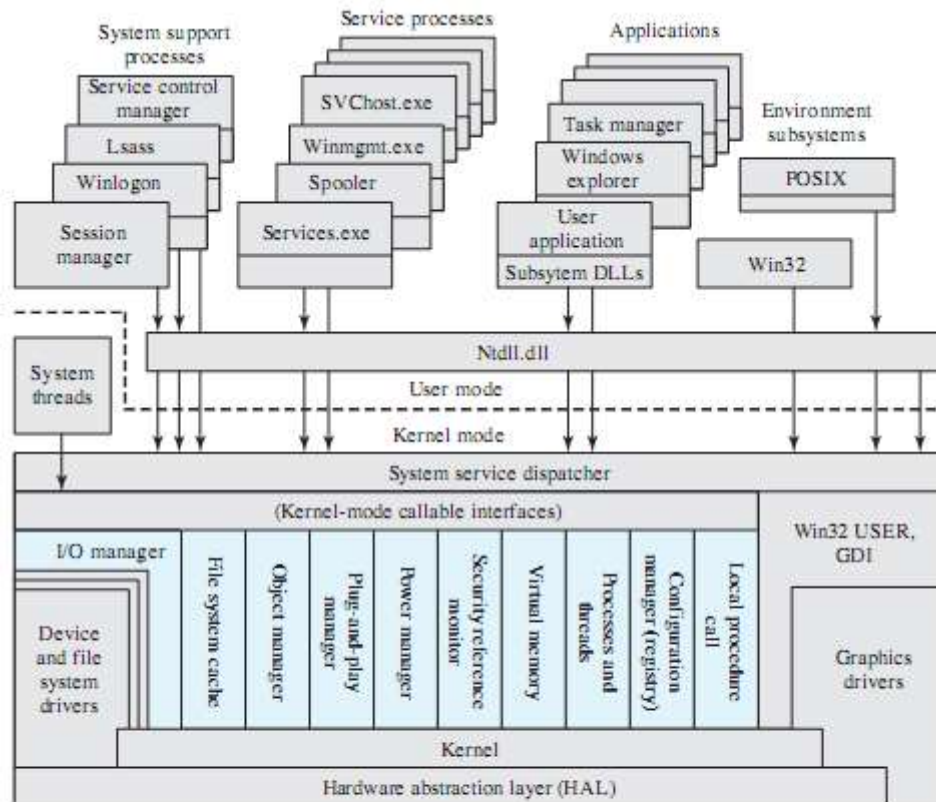


# Single-User Multitasking

- 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



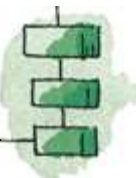
# Windows Architecture



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

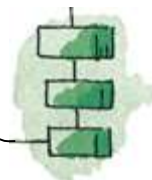
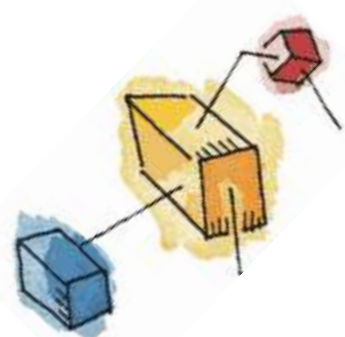


# Roadmap

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

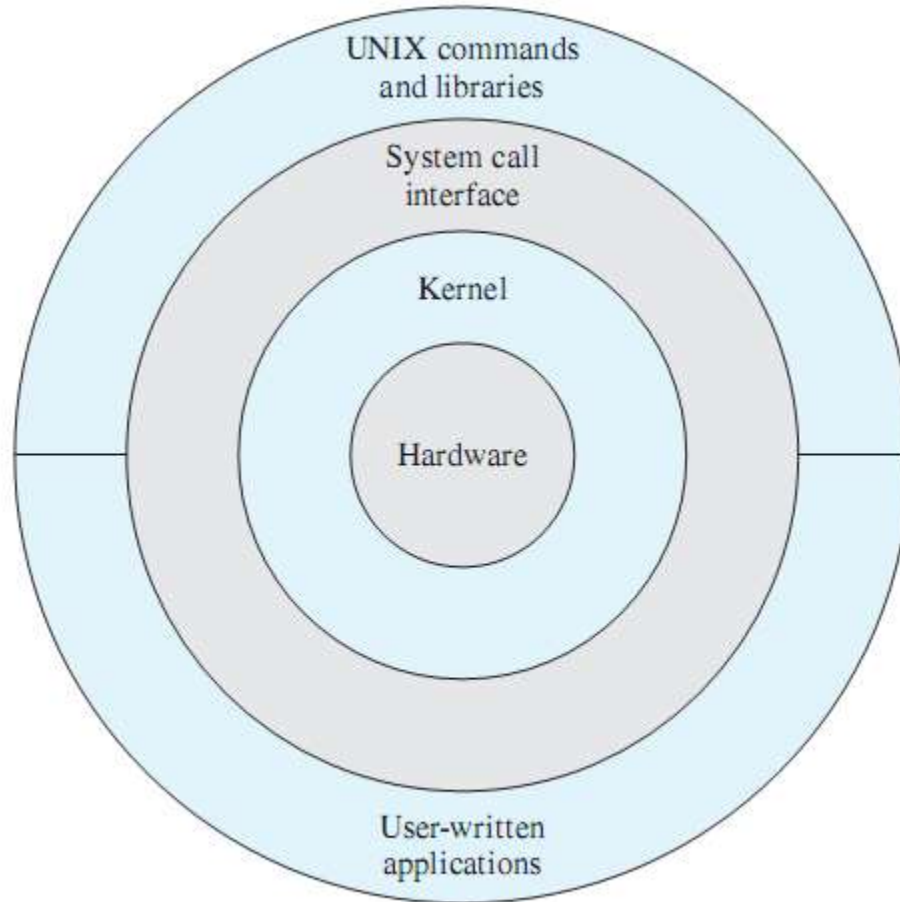
## → UNIX Systems

- Linux





# Description of UNIX



**Figure 2.14** General UNIX Architecture

# Traditional UNIX Kernel

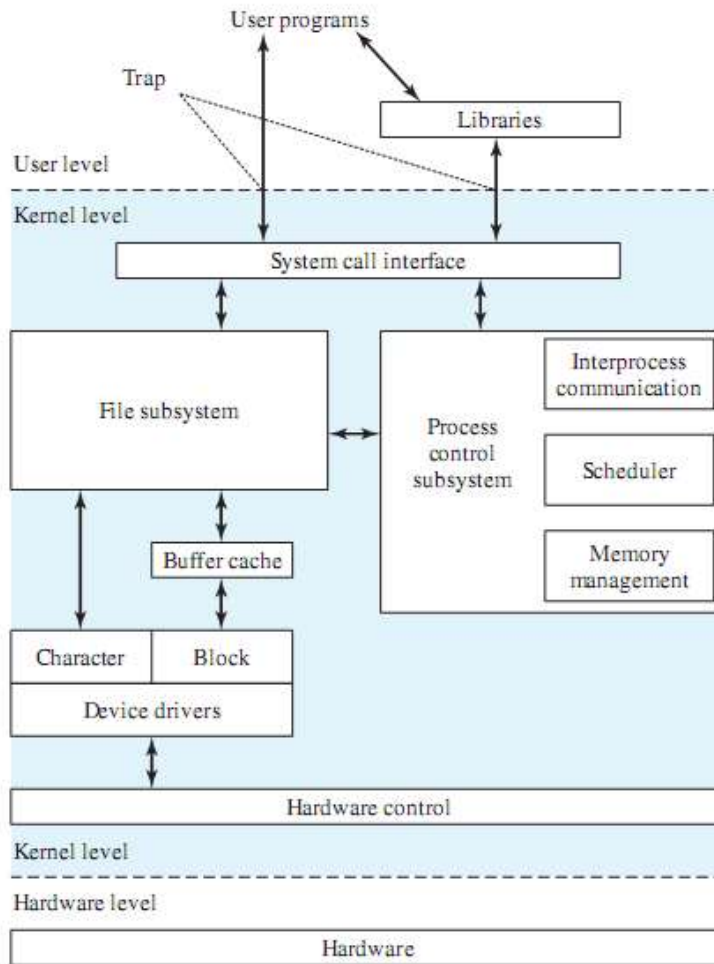


Figure 2.15 Traditional UNIX Kernel

# System V Release 4 (SVR4)

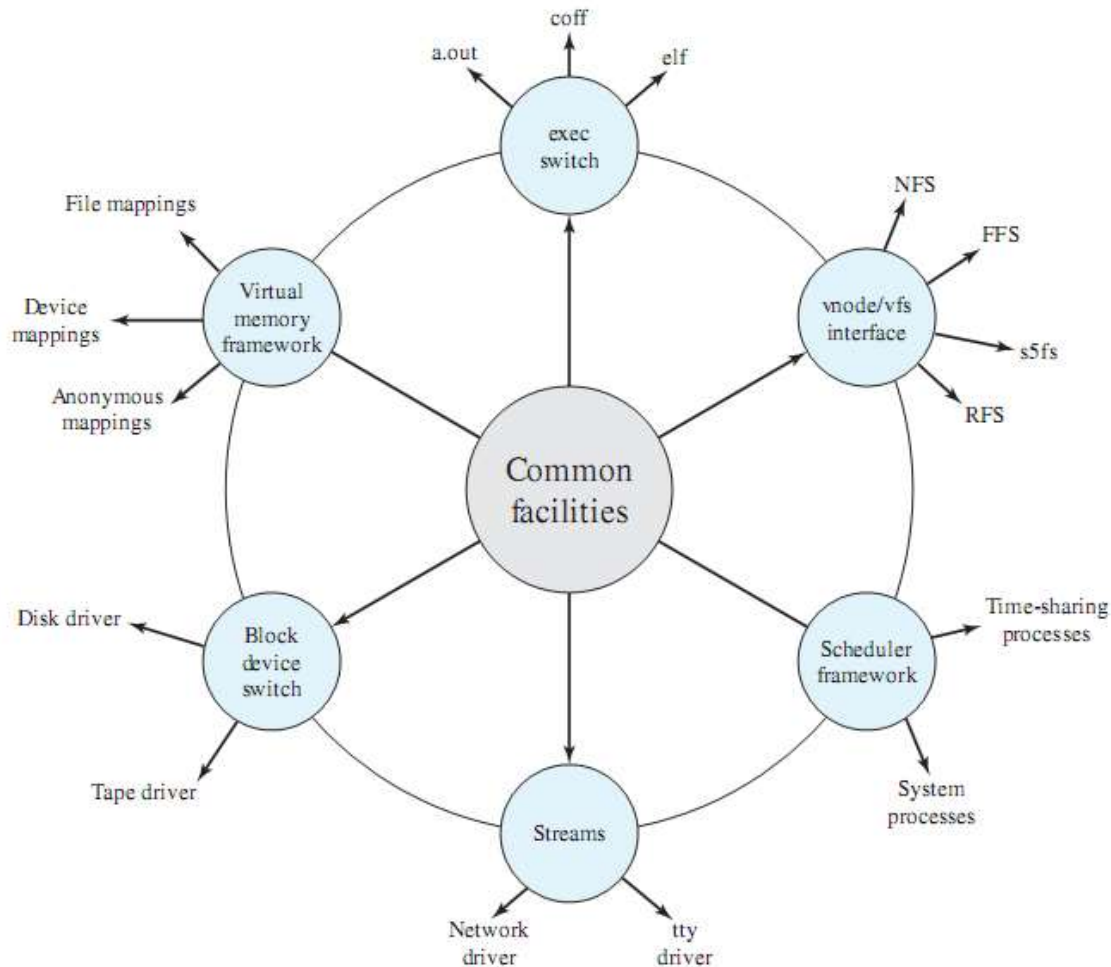


Figure 2.16 Modern UNIX Kernel

# Roadmap

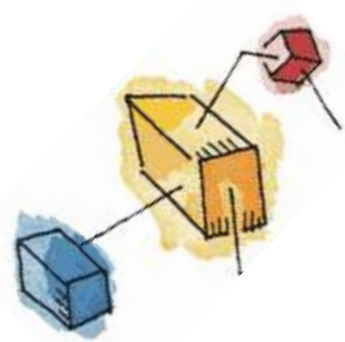
- Operating System Objectives/Functions
- The Evolution of Operating Systems
- Major Achievements
- Developments Leading to Modern Operating Systems
- Microsoft Windows Overview
- UNIX Systems

→ Linux



# Modular Monolithic Kernel

- Although monolithic, the kernel is structured 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

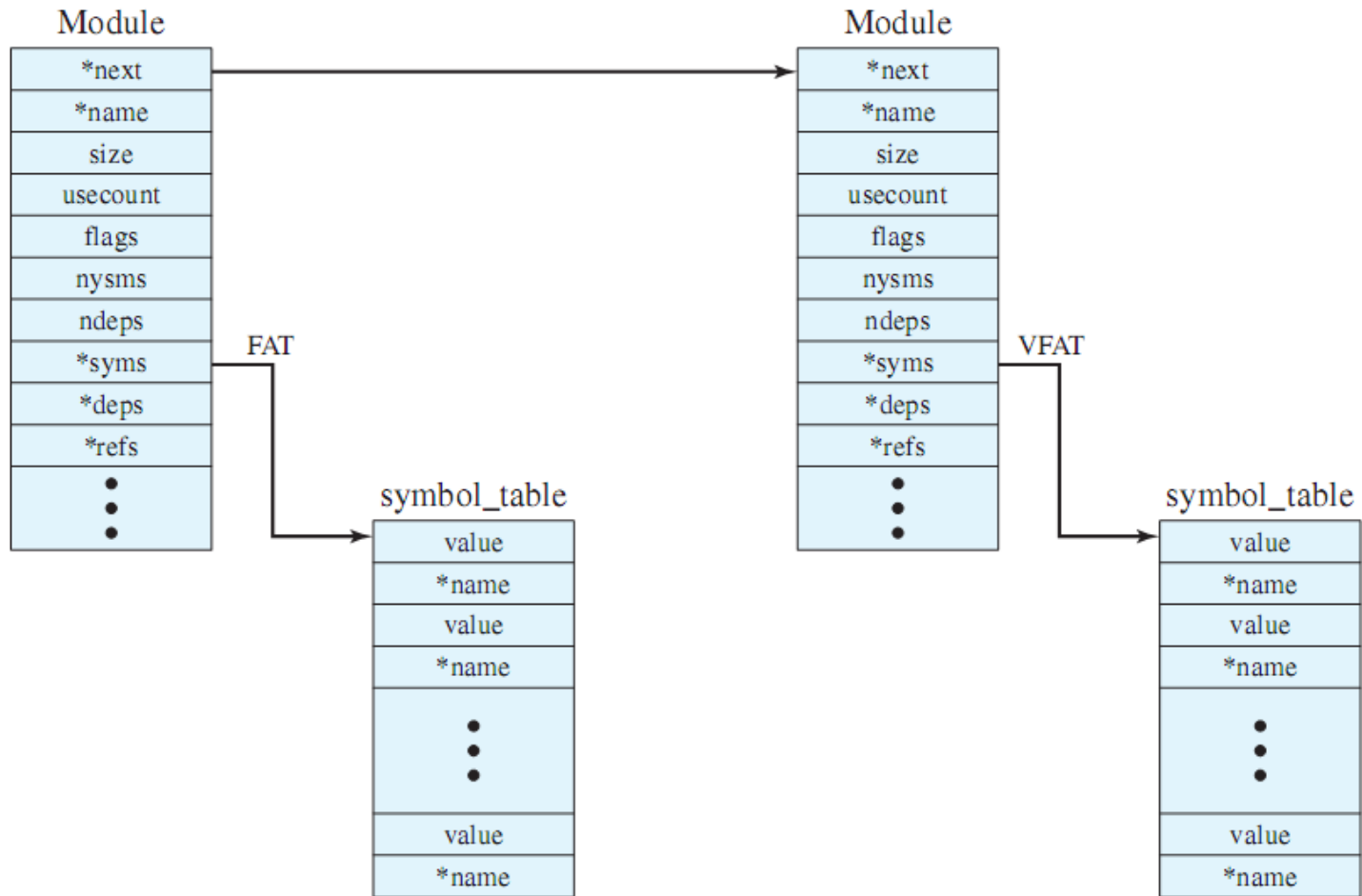


Figure 2.17 Example List of Linux Kernel Modules

# Linux Kernel Components

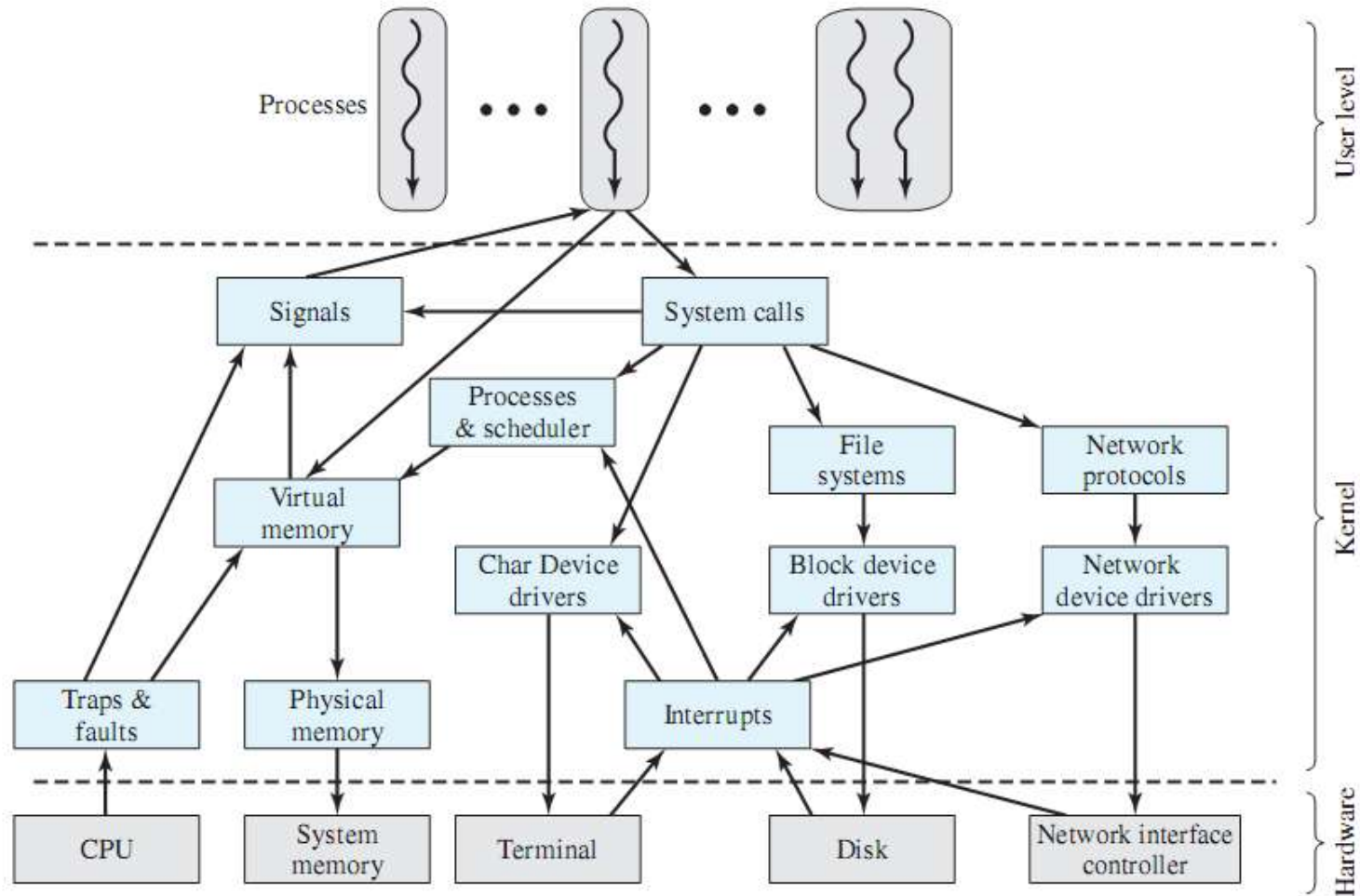


Figure 2.18 Linux Kernel Components