

# Computer Organization & Operating Systems (COOS)



<b>Course Code</b>	<b>EEE2001B</b>			
<b>Course Category</b>	<b>Professional Core</b>			
<b>Course Title</b>	<b>Computer Organization and Operating Systems</b>			
<b>Weekly Teaching Hrs. and Credits</b>	<b>L</b>	<b>T</b>	<b>Laboratory</b>	<b>Credits</b>
	<b>2</b>	<b>-</b>	<b>-</b>	<b>2 + 0 + 0</b>

**Pre-requisites:** Fundamentals of Computers, Data Structures, Computer Organization

**Course Objectives:**

1. To discuss a basic structure of computers.
2. To understand the I/O ports and memory system of the computers.
3. To acknowledge the concepts of operating systems and process management.
4. To explain the concepts of Memory Management and I/O management

**Course Outcomes:** After completion of this course students will be able to

1. Understand the basics of computer organization.
2. Learn about I/O ports and memory system of the computer.
3. Comprehend key mechanisms of the Operating System functions.
4. Assess memory management issues.



# Text and Reference Books

- **Text Books:**

- Carl Hamacher, Zvonko Vranesic, Safwat Zaky: Computer Organization, 5th Edition, Tata McGraw Hill, 2002.
- Carl Hamacher, Zvonko Vranesic, Safwat Zaky, Naraig Manjikian : Computer Organization and Embedded Systems, 6<sup>th</sup> Edition, Tata McGraw Hill, 2012.

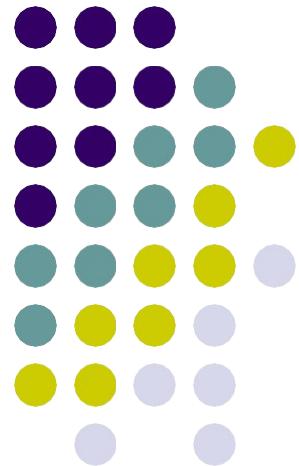
- **Reference Books:**

- William Stallings: Computer Organization & Architecture, 9<sup>th</sup> Edition, Pearson, 2015.

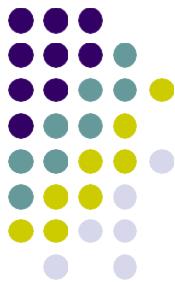
# **Unit-1 (Part 1)**

# **Basic Structure of Computers**

---



# Syllabus Points



- Basic operational concepts
- Bus structures
- Performance – processor clock, basic performance equation, clock rate
- Performance measurement
- Memory location and addresses
- Memory operations
- Instructions and instruction sequencing
- Addressing modes
- Assembly language
- Basic input and output operations
- Stacks and queues
- Subroutines
- Additional instructions
- Encoding of machine instructions



# The Computer Revolution

- Progress in computer technology
  - Underpinned by Moore's Law
- Makes novel applications feasible
  - Computers in automobiles
  - Cell phones
  - Human genome project
  - World Wide Web
  - Search Engines
- Computers are universal



# Classes of Computers

- Desktop/laptop computers
  - General purpose, variety of software
  - Subject to cost/performance tradeoff
- Workstations
  - More computing power used in engg. applications, graphics etc.
- Enterprise System/ Mainframes
  - Used for business data processing
- Server computers (Low End Range)
  - Network based
  - High capacity, performance, reliability
  - Range from small servers to building sized
- Supercomputer (High End Range)
  - Large scale numerical calculation such as weather forecasting, aircraft design
- Embedded computers
  - Hidden as components of systems
  - Stringent power/performance/cost constraints

# Functional Units



# Functional Units

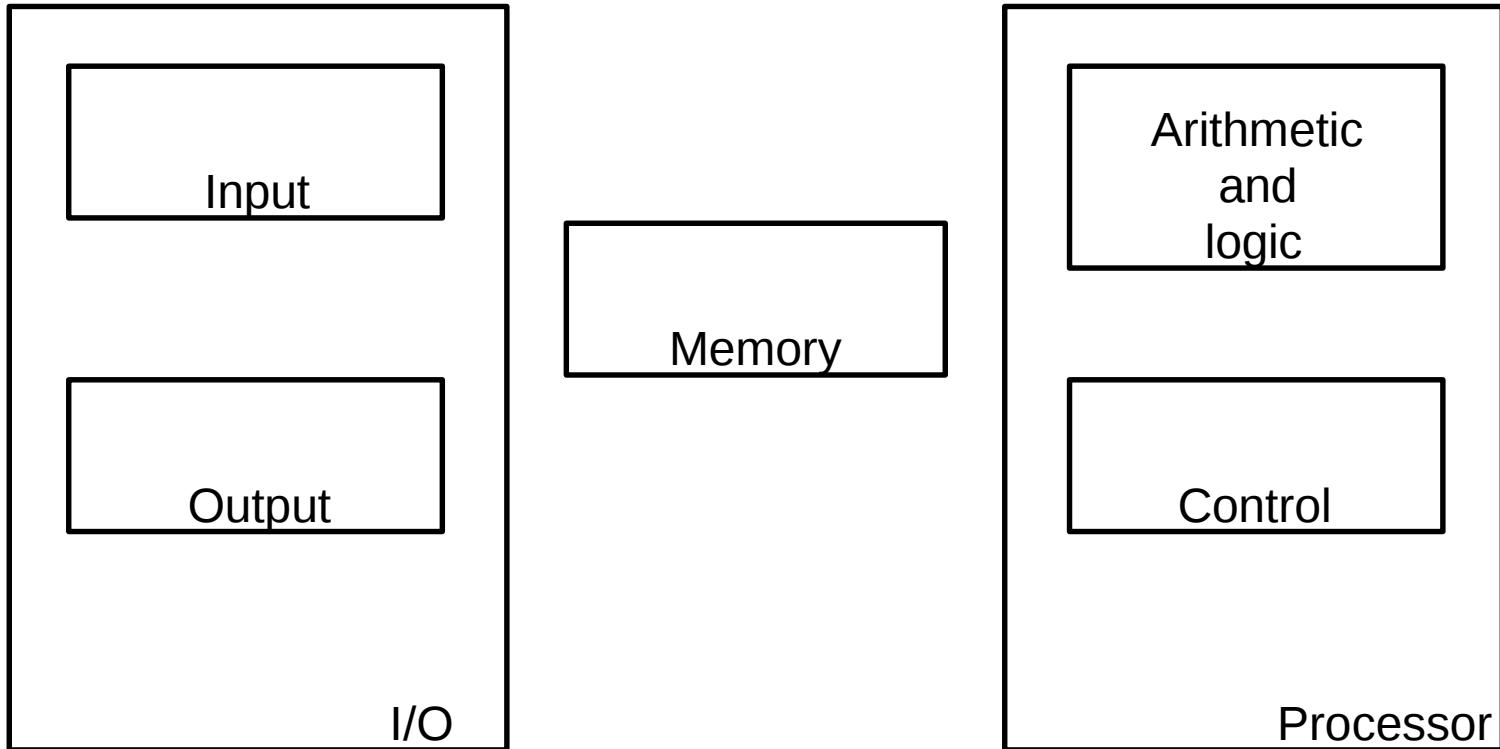
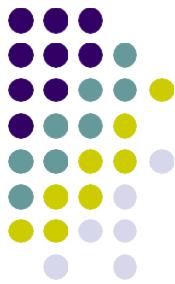


Figure 1.1. Basic functional units of a computer.

# Information Handled by a Computer



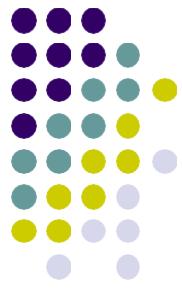
- Instructions/machine instructions
  - Govern the transfer of information within a computer as well as between the computer and its I/O devices
  - Specify the arithmetic and logic operations performed
  - Program
- Data
  - Used as operands by the instructions
  - Source program
- Encoded in binary code – 0 and 1



# Memory Unit

- Store programs and data
- Two classes of storage
  - Primary storage
    - ❖ Fast
    - ❖ Programs must be stored in memory while they are being executed
    - ❖ Large number of semiconductor storage cells
    - ❖ Processed in words
    - ❖ Address
    - ❖ RAM and memory access time
    - ❖ Memory hierarchy – cache, main memory
  - Secondary storage – larger and cheaper

# Data Representation



Bit – 1 bit 0 or 1

4 bit = nibble

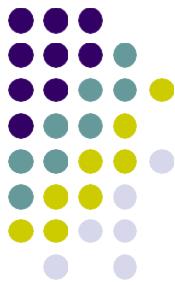
1Byte – 8 bits

2 byte – 16 bits ☐ 1 word

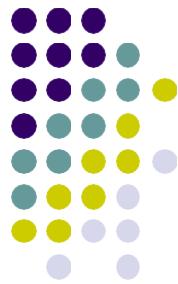
32 bits ☐ 2 words ☐ double word

64 bit ☐ 4 words ☐ quad word

# Arithmetic and Logic Unit (ALU)



- Most computer operations are executed in ALU of the processor.
  - – Load the operands into memory
  - – bring them to the processor
  - – perform operation in ALU
  - – store the result back to memory or retain in the processor.
- Registers
- Fast control of ALU



# Control Unit

- All computer operations are controlled by the control unit.
- The timing signals that govern the I/O transfers are also generated by the control unit.
- Control unit is usually distributed throughout the machine instead of standing alone.
- Operations of a computer:
  - Accept information in the form of programs and data through an input unit and store it in the memory
  - Fetch the information stored in the memory, under program control, into an ALU, where the information is processed
  - Output the processed information through an output unit
  - Control all activities inside the machine through a control unit



# The operations of a computer

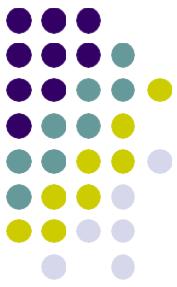
- The computer accepts information in the form of programs and data through an input unit and stores it in the memory.
- Information stored in the memory is fetched under program control into an arithmetic and logic unit, where it is processed.
- Processed information leaves the computer through an output unit.
- All activities in the computer are directed by the control unit.

# Basic Operational Concepts



# Review

- Activity in a computer is governed by instructions.
- To perform a task, an appropriate program consisting of a list of instructions is stored in the memory.
- Individual instructions are brought from the memory into the processor, which executes the specified operations.
- Data to be used as operands are also stored in the memory.



## Instruction:

[Label] : Opcode operand1, operand2

READ A

MOV R0, A

READ B

MOV R1, B

ADD R0, R1

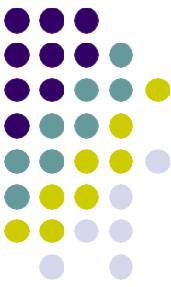
MOVE C, R0

PRINT C

STOP

START 100

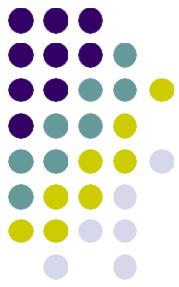
COOS - Unit 1 Basic Structure  
of Computers



# A Typical Instruction

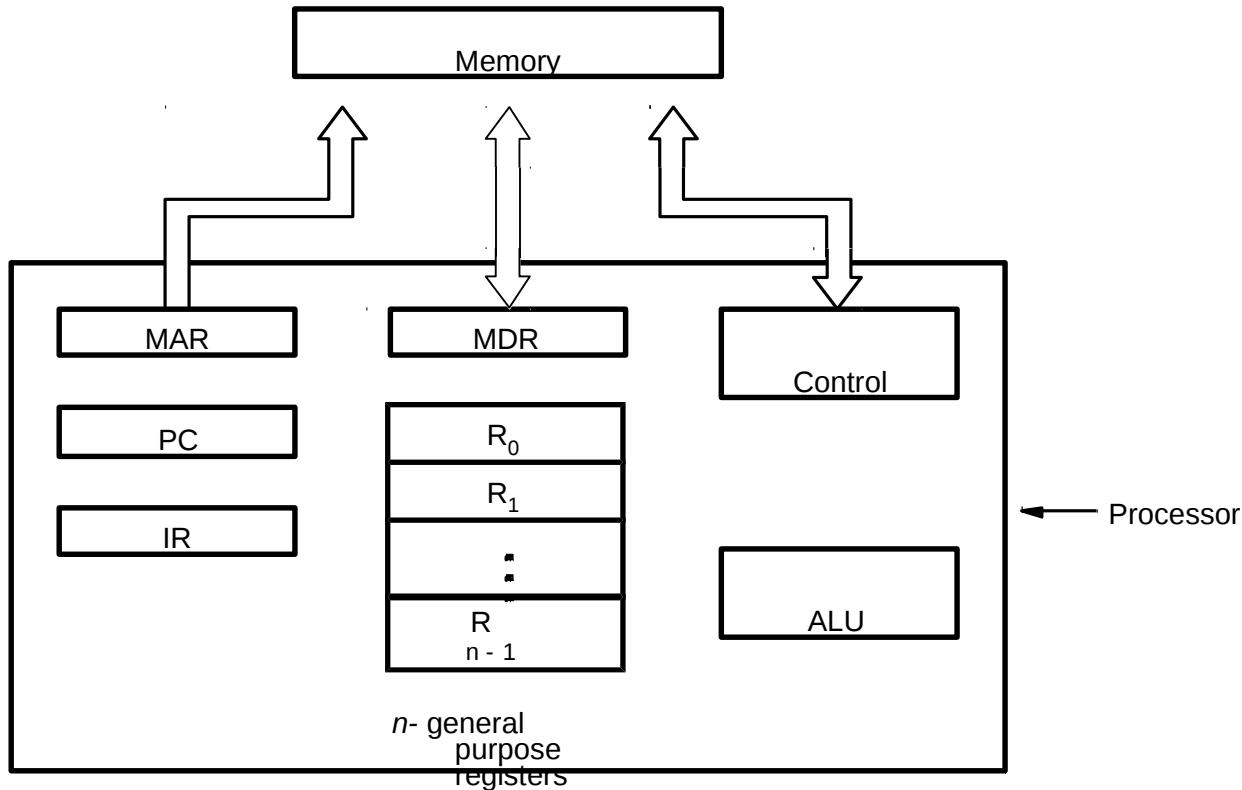
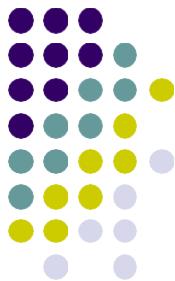
- Add LOCA, R0
- Add the operand at memory location LOCA to the operand in a register R0 in the processor.
- Place the sum into register R0.
- The original contents of LOCA are preserved.
- The original contents of R0 is overwritten.
- Instruction is fetched from the memory into the processor – the operand at LOCA is fetched and added to the contents of R0 – the resulting sum is stored in register R0.

# Separate Memory Access and ALU Operation



- Load LOC A, R1
- Add R1, R0
- Whose contents will be overwritten?

# Connection Between the Processor and the Memory



Connections between the processor and the memory.



# Registers

- Instruction register (IR)
- Program counter (PC)
- General-purpose register ( $R_0 - R_{n-1}$ )
- Memory address register (MAR)
- Memory data register (MDR)



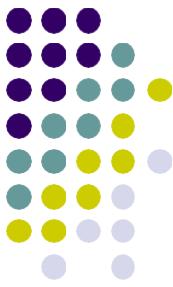
# Typical Operating Steps

- Programs reside in the memory through input devices
- PC is set to point to the first instruction
- The contents of PC are transferred to MAR
- A Read signal is sent to the memory
- The first instruction is read out and loaded into MDR
- The contents of MDR are transferred to IR
- Decode and execute the instruction

# Typical Operating Steps (Cont')



- Get operands for ALU
  - General-purpose register
  - Memory (address to MAR – Read – MDR to ALU)
- Perform operation in ALU
- Store the result back
  - To general-purpose register
  - To memory (address to MAR, result to MDR – Write)
- During the execution, PC is incremented to the next instruction



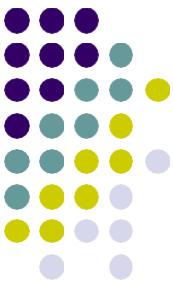
# Interrupt

- Normal execution of programs may be preempted if some device requires urgent servicing.
- The normal execution of the current program must be interrupted – the device raises an *interrupt* signal.
- Interrupt-service routine
- Current system information backup and restore (PC, general-purpose registers, control information, specific information)



# Bus Structures

- There are many ways to connect different parts inside a computer together.
- A group of lines that serves as a connecting path for several devices is called a *bus*.
- Address/data/control



# Bus Structure

- Single-bus

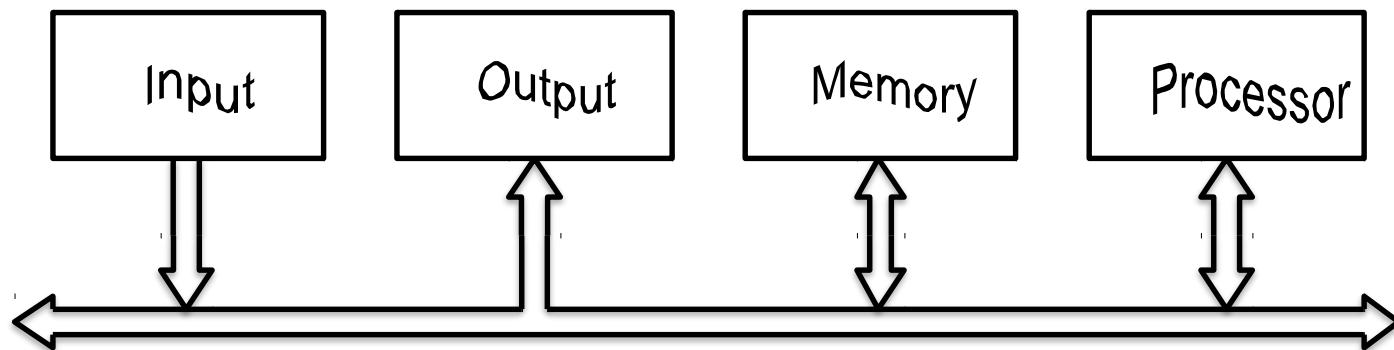


Figure 1.3. Single-bus structure.

- Multiple Buses

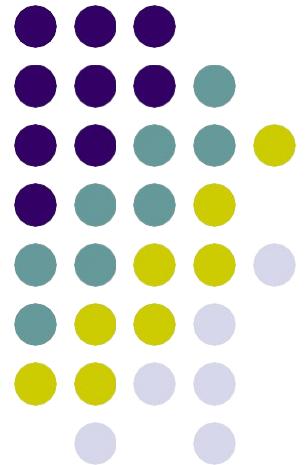


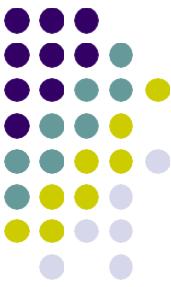
# Speed Issue

- Different devices have different transfer/operate speed.
- If the speed of bus is bounded by the slowest device connected to it, the efficiency will be very low.
- How to solve this?
- A common approach – use buffers.  
e.g.- Printing the characters



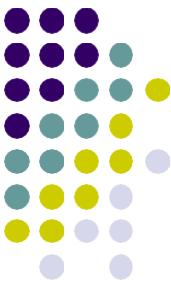
# Performance





# Performance

- The most important measure of a computer is how quickly it can execute programs.
- Three factors affect performance:
  - Hardware design
  - Instruction set
  - Compiler



# Performance

- Processor time to execute a program depends on the hardware involved in the execution of individual machine instructions.

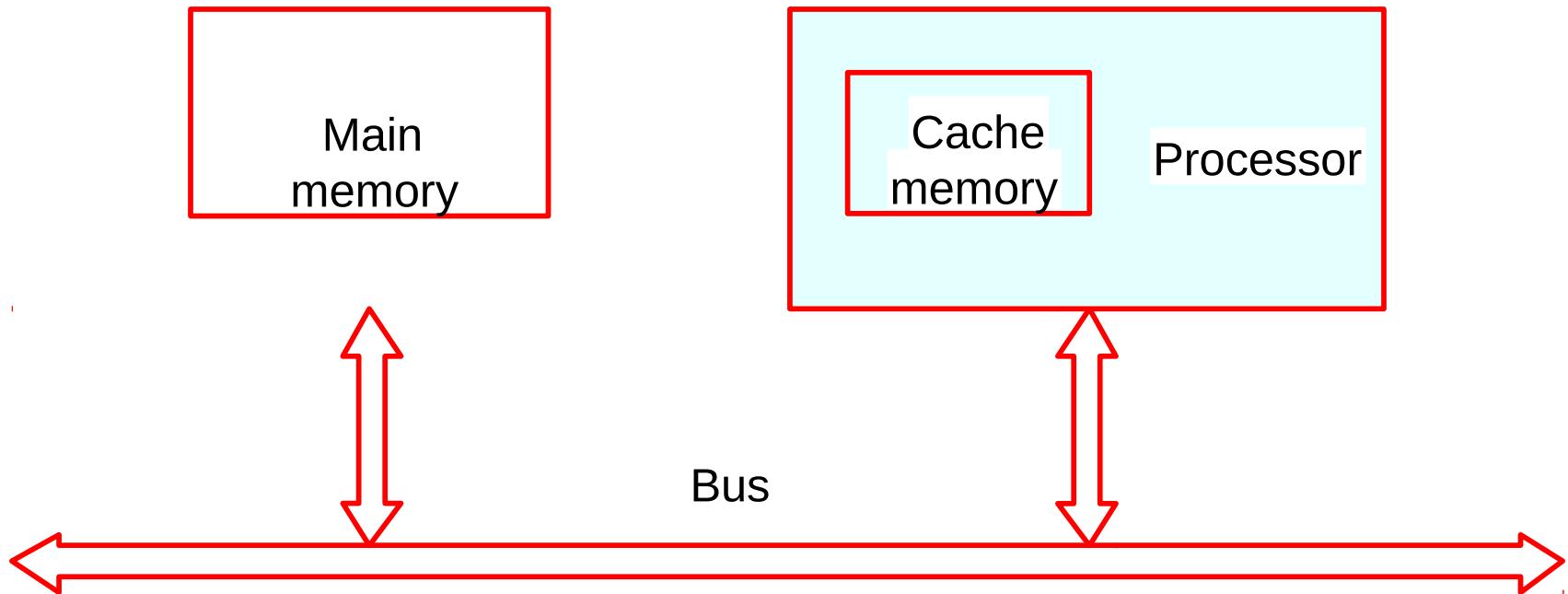
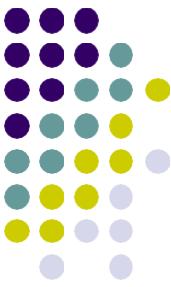


Figure 1.5. The processor cache.



# Performance

- The processor and a relatively small ~~memory~~ can be fabricated integrated circuit on a single chip.
  - Speed
  - Cost
  - Memory management



# Processor Clock

- Clock, clock cycle (P), and clock rate ( $R=1/P$ )
- The execution of each instruction is divided into several steps (Basic Steps), each of which completes in one clock cycle.
- Hertz – cycles per second



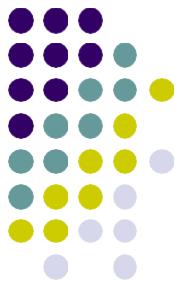
# Basic Performance Equation

- T – processor time required to execute a program that has been prepared in high-level language
- N – number of actual machine language instructions needed to complete the execution (note: loop)
- S – average number of basic steps needed to execute one machine instruction. Each step completes in one clock cycle
- R – clock rate
- Note: these are not independent to each other

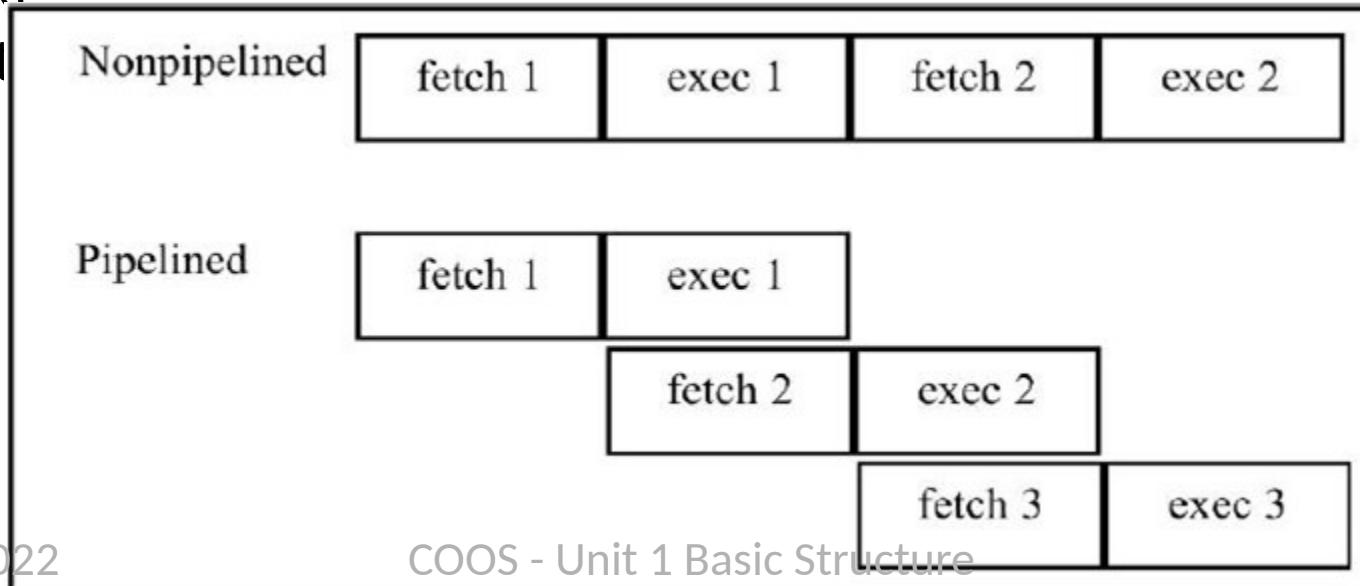
$$T = \frac{N \times S}{R}$$

- How to improve T?
- 12/15/2022 COOS - Unit 1 Basic Structure of Computers
- Reduce N and S, Increase R, but these affect

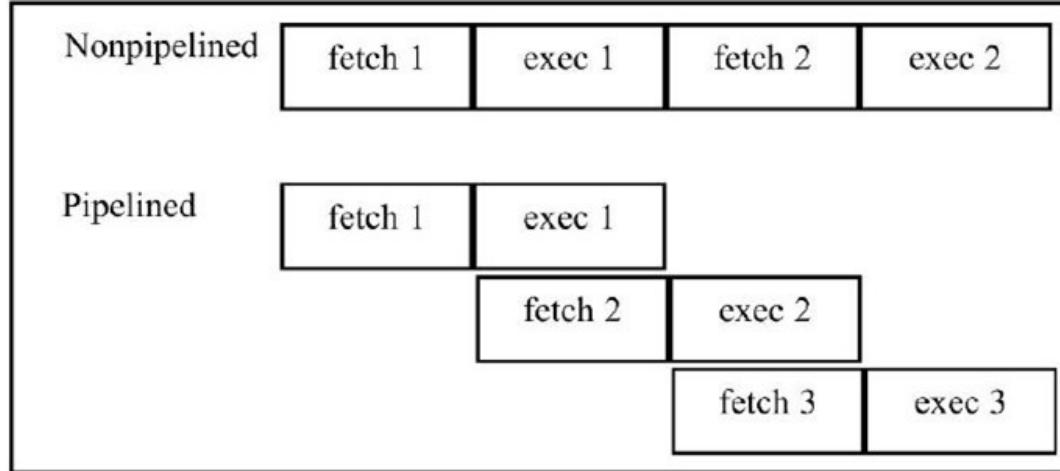
# Pipeline and Superscalar Operation



- Instructions are not necessarily executed one after another.
- The value of S doesn't have to be the number of clock cycles to execute one instruction.
- Pipelining – overlapping the execution of successive instructions.
- Add R1, R2, R3 at the same time processor reads next inst



1 clk cycle per phase



2 = Fetch and execute  
3 instruction

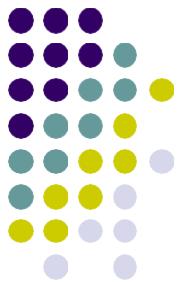
NP = No. of instruction \* no. of phases =  $3*2 =$   
6 clk cycle

Pipelines = 1 Instruction \* 2 +(2) remaining  
instruction\*1

$$= 1 * 2 + 2$$

$$= 4$$

# Pipeline and Superscalar Operation

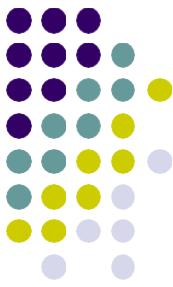


- Superscalar operation – multiple instruction pipelines are implemented in the processor.
- Goal – reduce  $S$  (could become  $<1!$ )



# Clock Rate

- Increase clock rate
  - Improve the integrated-circuit (IC) technology to make the circuits faster
  - Reduce the amount of processing done in one basic step (however, this may increase the number of basic steps needed)
- Increases in R that are entirely caused by improvements in IC technology affect all aspects of the processor's operation equally except the time to access the main memory.



# Compiler

- A compiler translates a high-level language program into a sequence of machine instructions.
- To reduce N, we need a suitable machine instruction set and a compiler that makes good use of it.
- Goal – reduce  $N \times S$
- A compiler may not be designed for a specific processor; however, a high-quality compiler is usually designed for, and with, a specific processor.



# Performance Measurement

- T is difficult to compute.
- Measure computer performance using benchmark programs.
- System Performance Evaluation Corporation (SPEC) selects and publishes representative application programs for different application domains, together with test results for many commercially available computers.
- Compile and run (no simulation)
- Reference computer

---

*SPEC rating* =

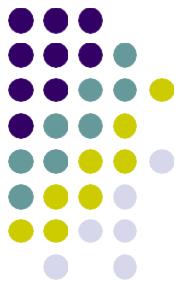
Running time on the

reference computer  $\frac{1}{\text{Running time on}}$

*SPEC rating* =  $(\prod_{i=1}^n \frac{\text{SPEC}_{i,r}}{\text{SPEC}_{i,t}})^{\frac{1}{n}}$  for computer under test

- n is the number of program in the suite

# Multiprocessors and Multicomputers



- Multiprocessor computer
  - Execute a number of different application tasks in parallel
  - Execute subtasks of a single large task in parallel
  - All processors have access to all of the memory – shared-memory multiprocessor
  - Cost – processors, memory units, complex interconnection networks
- Multicomputers
  - Each computer only have access to its own memory
  - Exchange message via a communication network – message-passing multicomputers

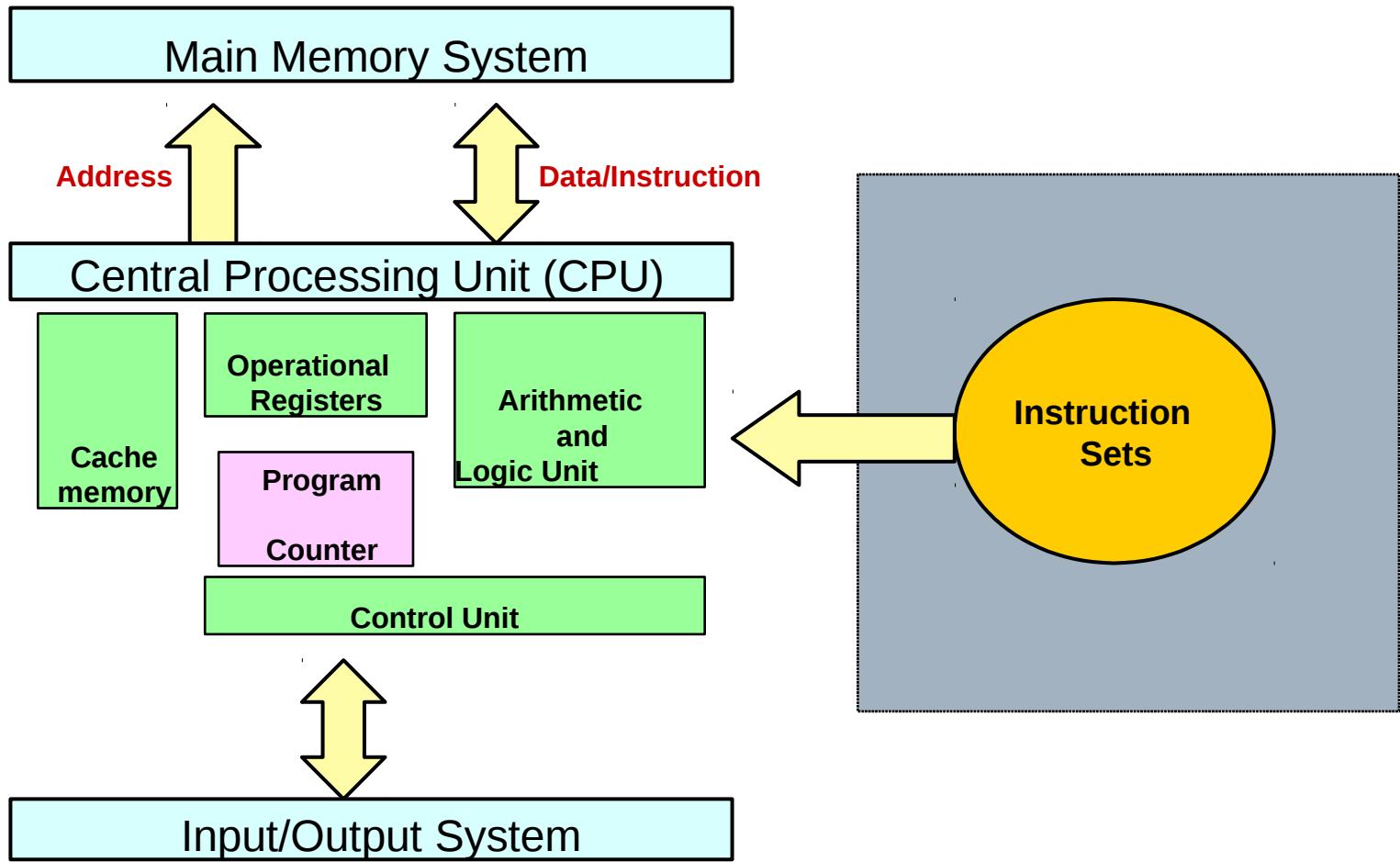
# Unit 1 Part -II

## Machine Instructions & Programs

# Outline

- Numbers, Arithmetic Operations, and Characters
- Memory Locations and Addresses
- Memory Operation
- Instructions and Instruction Sequencing
- Addressing Modes
- Assembly Language
- Basic Input / Output Operations
- Stacks and Queues
- Subroutines
- Linked List
- Encoding of Machine Instructions

# Computer System



# Number Representation

- Consider an  $n$ -bit vector  $B = b_{n-1}Kb_1b_0$ , where  $b_i = 0$  or  $1$  for  $0 \leq i \leq n - 1$
- The vector B can represent unsigned integer values V in
  - ◆  $V(B) = b_{n-1}b_{n-2}\dots b_1b_0$ , where  $V = b_{n-1} \times 2^{n-1} + b_{n-2} \times 2^{n-2} + \dots + b_1 \times 2^1 + b_0 \times 2^0$
- We need to represent positive and negative numbers for most applications
- Three systems are used for representing such numbers
  - ◆ Sign-and-magnitude
  - ◆ 1's-complement
  - ◆ 2's-complement

- 0000 0
- 0001 1
- 0010 2
- (+12)
- (-5)
- Sign + Magnitude
  - + = 0
  - - = 1
- 0 0101
- 1's complement □
- 10001101
- 01110010
- 2's complement
- 10001101 (Given no.)
- 01110010 (1's complement)
- 1 (add 1)
- -----
- 0111001 1

# Number Systems

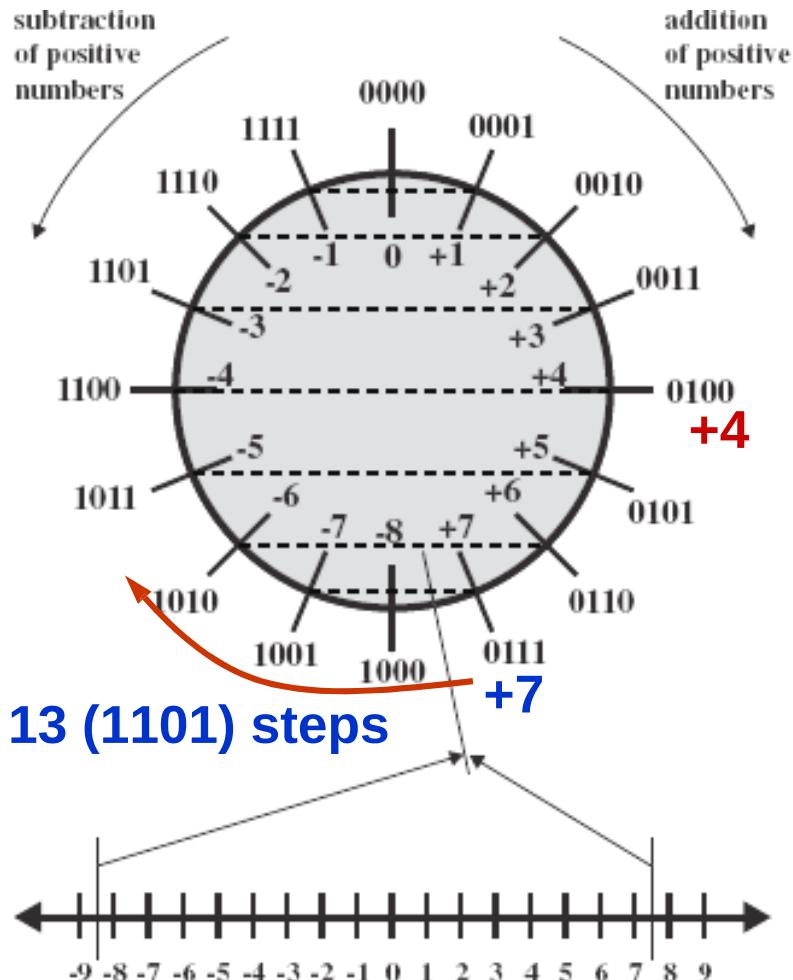
- In *sign-and-magnitude system*
  - ◆ Negative values are represented by changing the most significant bit from 0 to 1
- In *1's-complement system*
  - ◆ Negative values are obtained by complementing each bit of the corresponding positive number
  - ◆ The operation of forming the 1's-complement of a given number is equivalent to subtracting that number from  $2^n - 1$
- In *2's-complement system*
  - ◆ The operation of forming the 2's-complement of a given number is done by subtracting that number from  $2^n$
- The 2's-complement of a number is obtained by adding 1 to 1's-complement of that number

# An Example of Number Representations

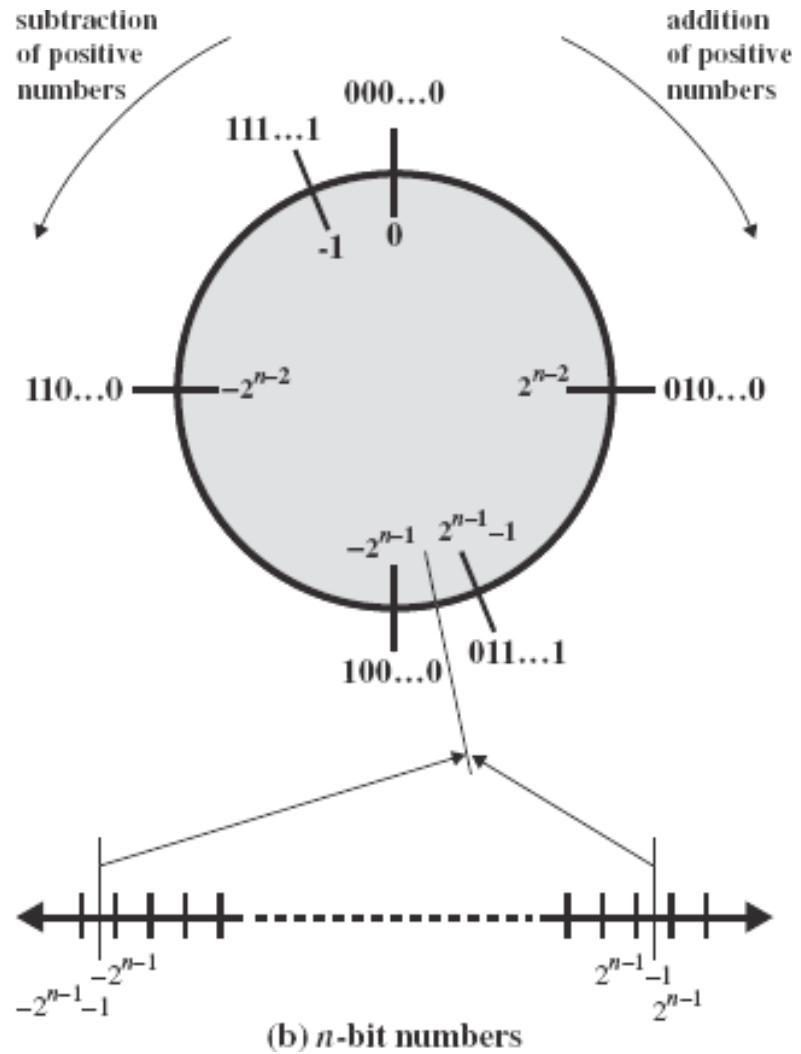
$b_3 b_2 b_1 b_0$	sign and magnitude	1's-complement	2's-complement
0 1 1 1	+7	+7	+7
0 1 1 0	+6	+6	+6
0 1 0 1	+5	+5	+5
0 1 0 0	+4	+4	+4
0 0 1 1	+3	+3	+3
0 0 1 0	+2	+2	+2
0 0 0 1	+1	+1	+1
0 0 0 0	+0	+0	+0
1 0 0 0	-0	-7	-8
1 0 0 1	-1	-6	-7
1 0 1 0	-2	-5	-6
1 0 1 1	-3	-4	-5
1 1 0 0	-4	-3	-4
1 1 0 1	-5	-2	-3
1 1 1 0	-6	-1	-2
1 1 1 1	-7	-0	-1

# 2's-Complement System

$$+7 + (-3)$$



(a) 4-bit numbers



(b)  $n$ -bit numbers

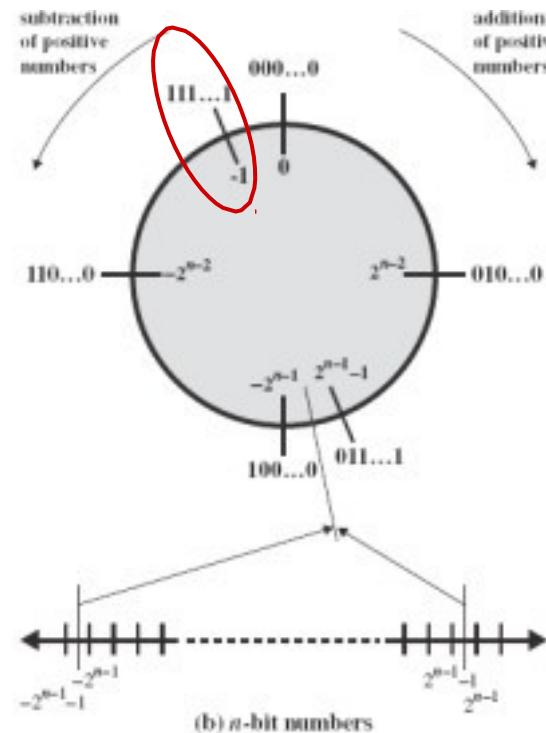
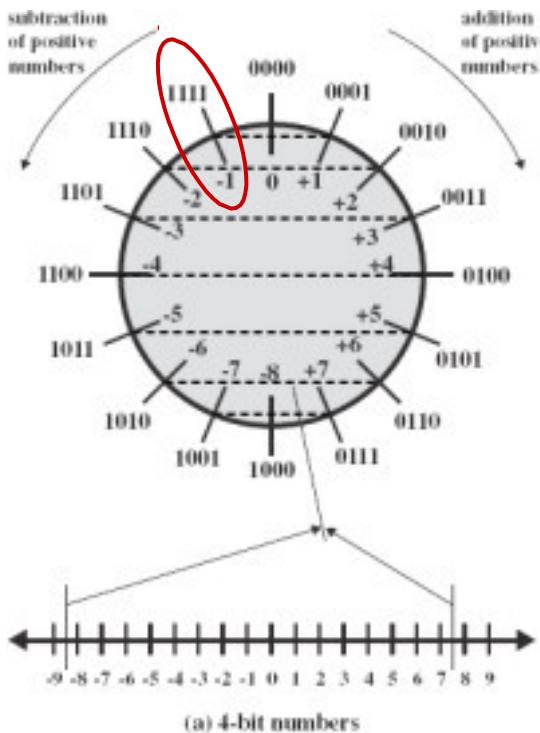
# Addition of Numbers in 2's Complement

$\begin{array}{r} 1001 = -7 \\ +0101 = 5 \\ \hline 1110 = -2 \end{array}$ <p>(a) <math>(-7) + (+5)</math></p>	$\begin{array}{r} 1100 = -4 \\ +0100 = 4 \\ \hline 10000 = 0 \end{array}$ <p>(b) <math>(-4) + (+4)</math></p>
$\begin{array}{r} 0011 = 3 \\ +0100 = 4 \\ \hline 0111 = 7 \end{array}$ <p>(c) <math>(+3) + (+4)</math></p>	$\begin{array}{r} 1100 = -4 \\ +1111 = -1 \\ \hline 11011 = -5 \end{array}$ <p>(d) <math>(-4) + (-1)</math></p>
$\begin{array}{r} 0101 = 5 \\ +0100 = 4 \\ \hline 1001 = \text{Overflow} \end{array}$ <p>(e) <math>(+5) + (+4)</math></p>	$\begin{array}{r} 1001 = -7 \\ +1010 = -6 \\ \hline 10011 = \text{Overflow} \end{array}$ <p>(f) <math>(-7) + (-6)</math></p>

# Sign Extension of 2's Complement

## ➤ Sign extension

- ◆ To represent a signed number in 2's complement form using a larger number of bits, repeat the sign bit as many times as needed to the left



# Memory Locations

- A memory consists of cells, each of which can store a bit of binary information (0 or 1)
- Because a single bit represents a very small amount of information
  - ◆ Bits are seldom handled individually
- The memory usually is organized so that a group of  $n$  bits can be stored or retrieved in a single, basic operation
  - ◆ Each group of  $n$  bits is referred to as a *word* of information, and  $n$  is called the *word length*
  - ◆ A unit of 8 bits is called a byte
- Modern computers have word lengths that typically range from 16 to 64 bits

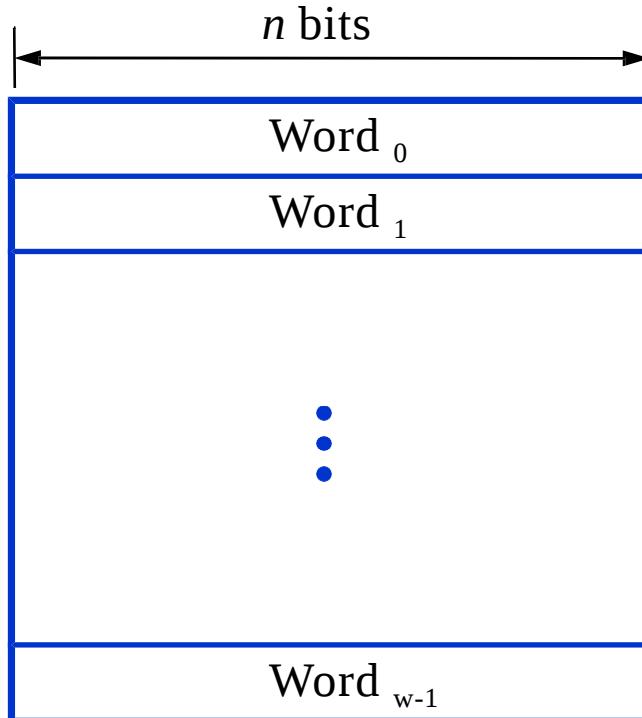
# Memory Addresses

- Accessing the memory to store or retrieve a single item of information, either a word or a byte, requires distinct names or *addresses* for each item location
- It is customary to use numbers from 0 to  $2^k - 1$  as the address space of successive locations in the memory
  - ◆ K denotes address
  - ◆  $2^k - 1$  denotes address space of memory locations
- For example, a 24-bit address generates an address space of  $2^{24}$ (16,777,216) locations
- Terminology
  - ◆  $2^{10}$ : 1K (kilo)
  - ◆  $2^{20}$ : 1M (mega)
  - ◆  $2^{30}$ : 1G (giga)
  - ◆  $2^{40}$ : 1T (tera)

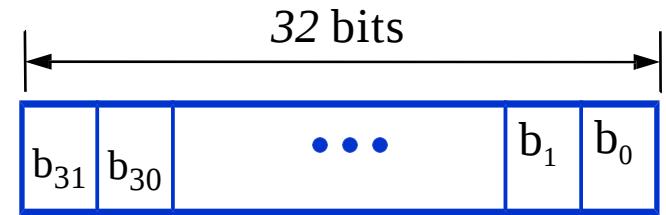
- SY
  - A 2102201 -75
  - B 2202201 -75
  - C 2302201-75
- 
- A ☐ 8 bit ☐ 0000 1010

# Memory Words

Memory words



A signed integer



Four characters



ASCII  
character

# Big-Endian & Little-Endian Assignments

- Byte addresses can be assigned across words in two ways
  - ◆ Big-endian and little-endian

Word address      Byte address

0	0	1	2	3
4	4	5	6	7
⋮				
$2^{k-4}$	$2^{k-4}$	$2^{k-3}$	$2^{k-2}$	$2^{k-1}$

**Big-endian assignment**

Word address      Byte address

0	3	2	1	0
4	7	6	5	4
⋮				
$2^{k-4}$	$2^{k-1}$	$2^{k-2}$	$2^{k-3}$	$2^{k-4}$

**Little-endian assignment**

# Memory Operation

- Random access memories must have two basic operations
  - ◆ **Write:** writes a data into the specified location
  - ◆ **Read:** reads the data stored in the specified location
- In machine language program, the two basic operations usually are called
  - ◆ **Store:** write operation
  - ◆ **Load:** read operation
- The Load operation transfers a copy of the contents of a specific memory location to the processor. The memory contents remain unchanged
- The Store operation transfers an item of information from the processor to a specific memory location, destroying the former contents of that location

# Instructions

- A computer must have instructions capable of performing four types of operations
  - ◆ Data transfers between the memory and the processor registers
  - ◆ Arithmetic and logic operations on data
  - ◆ Program sequencing and control
  - ◆ I / O transfers
- **Register transfer notation**
  - ◆ The contents of a location are denoted by placing square brackets around the name of the location
  - ◆ For example,  $R1 \leftarrow [LOC]$  means that the contents of ~~many~~ location LOC are transferred into processor register R1
  - ◆ As another example,  $R3 \leftarrow [R1] + [R2]$  means that adds the ~~contents~~ of registers R1 and R2, and then places their sum into register R3

$$C = A + B$$

- Read A
- Read B
- Add A, B
- Move B, C
- Print C

OPCODE	Operand list		
ADD	A Source operand	B Destinatio n operand	

# Assembly Language Notation

## ➤ Types of instructions

- ◆ Zero-address instruction
- ◆ One-address instruction
- ◆ Two-address instruction
- ◆ Three-address instruction

## ➤ Zero-address instruction

- ◆ For example, store operands in a structure called a **pushdown stack**

## ➤ One-address instruction

- ◆ Instruction form: **Operation Destination**
- ◆ For example, **Add A**: add the contents of memory location A to the contents of the accumulator register and place the sum back into the accumulator
- ◆ As another example, **Load A**: copies the contents of memory location A into the accumulator

# Assembly Language Notation

## ➤ Two-address instruction

- ◆ Instruction form: **Operation Source, Destination**
- ◆ For example, **Add A, B**: performs the operation  $B \leftarrow [A] + [B]$ . When the sum is calculated, the result is sent to the memory and stored in location B
- ◆ As another example, **Move B, C** : performs the operation  $C \leftarrow [B]$ , leaving the contents of location B unchanged

## ➤ Three-address instruction

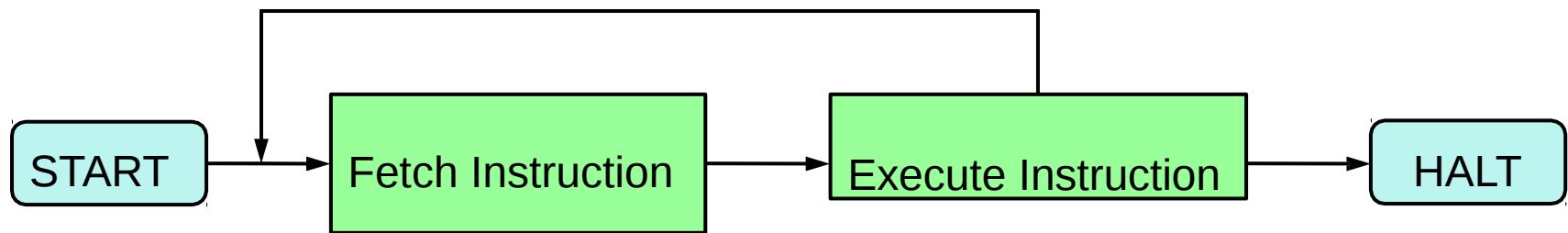
- ◆ Instruction form: **Operation Source1, Source2, Destination**
- ◆ For example, **Add A, B, C**: adds A and B, and the result is sent to the memory and stored in location C
- ◆ If k bits are needed to specify the memory address of each operand, the encoded form of the above instruction must contain 3k bits for addressing purposes in addition to the bits needed to denote the Add operation

# Instruction Execution

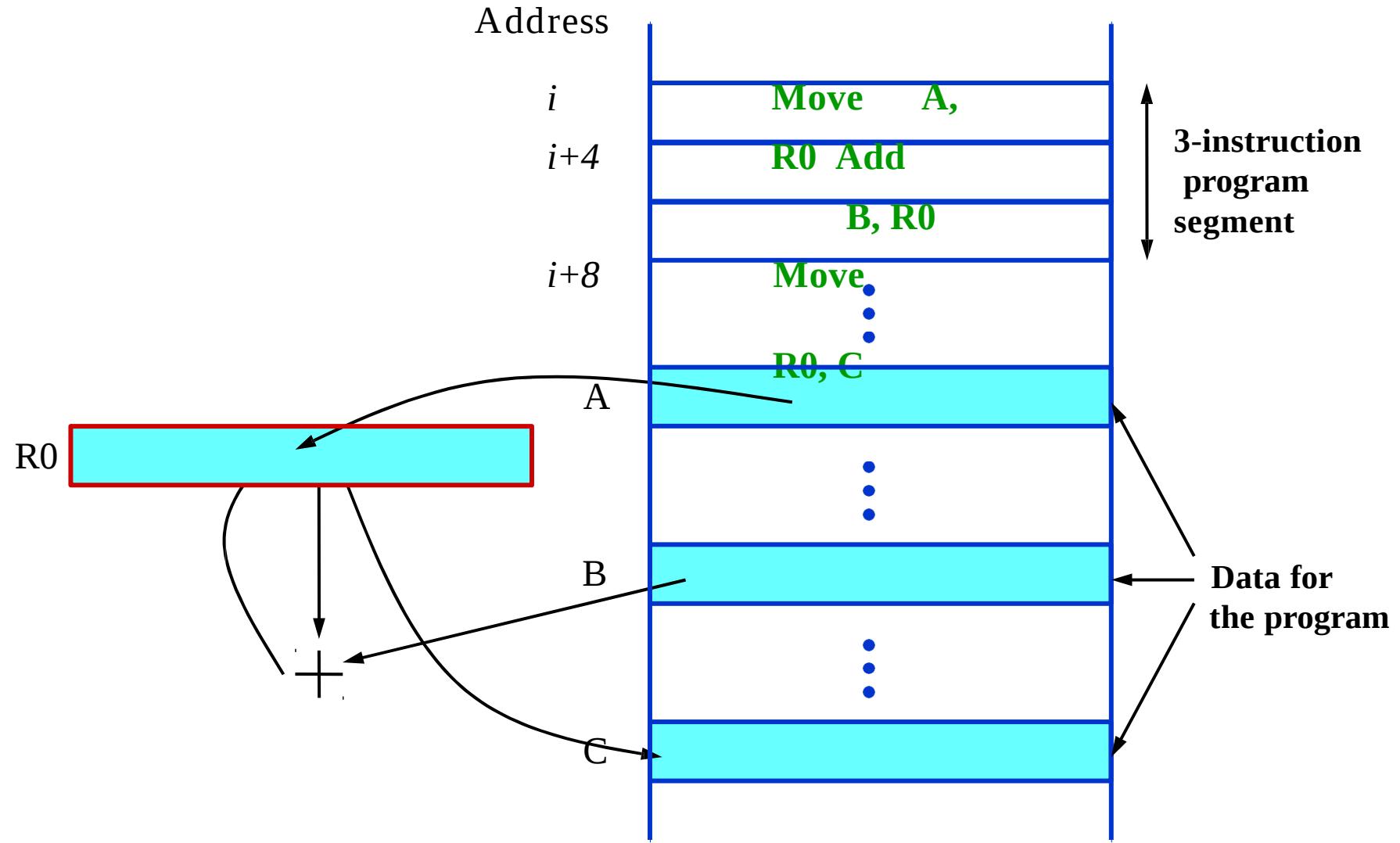
## ➤ How a program is executed

- ◆ The processor contains a register called the program counter (PC), which holds the address of the instruction to be executed next. To begin executing a program, the address of its first instruction must be placed into the PC, then the processor control circuits use the information in the PC to fetch and execute instruction, one at a time, in the order of increasing address

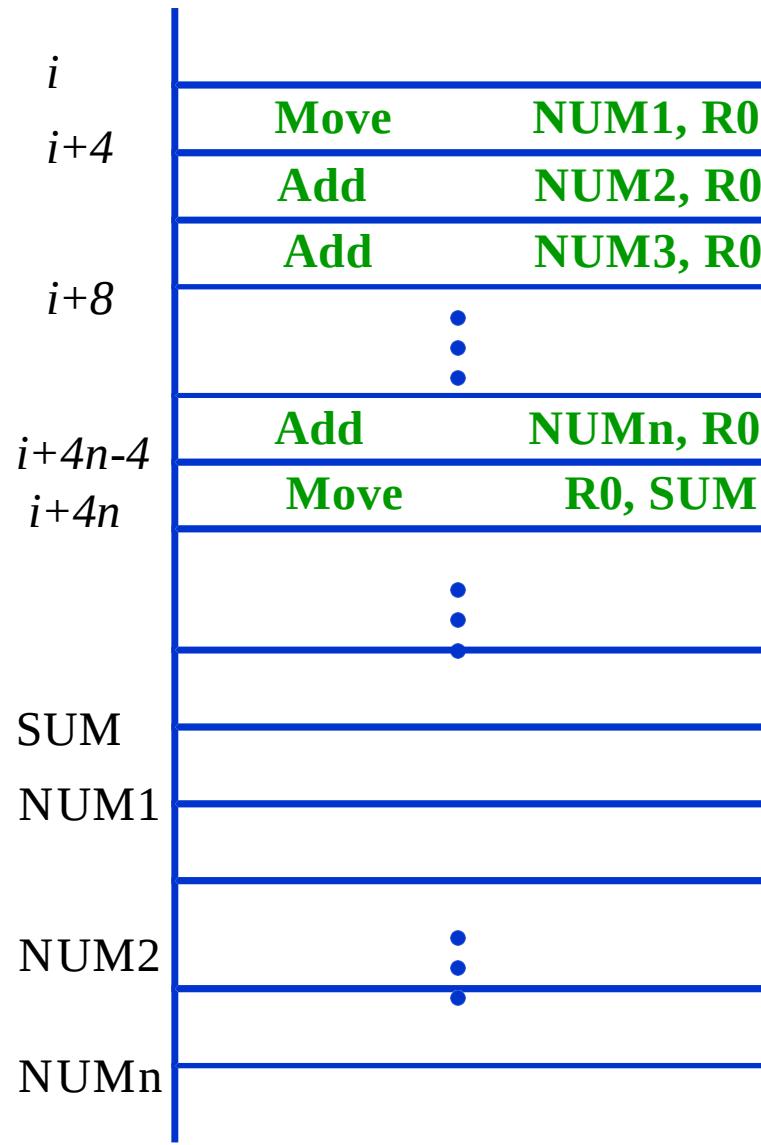
## ➤ Basic instruction cycle



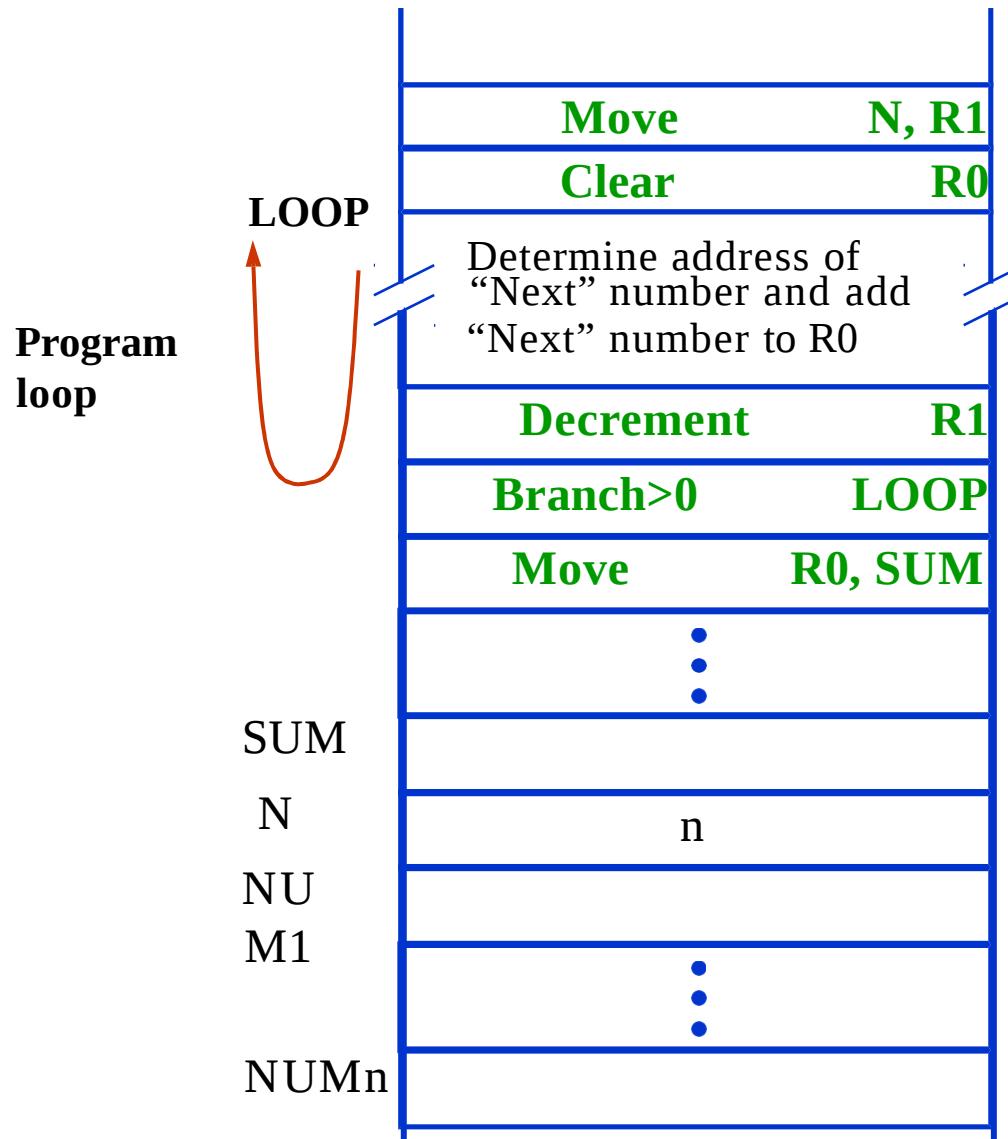
# A Program for $C = [A] + [B]$



# Straight-Line Sequencing



# Branching



# Condition Codes

- The processor keeps track of information about the results of various operations for use by subsequent conditional branch instructions. This is accomplished by recoding required information in individual bits, often called *condition code flags*
- **Four commonly used flags are**
  - ◆ N (negative): set to 1 if the result is negative; otherwise, cleared to 0
  - ◆ Z (zero): set to 1 if the result is 0; otherwise, cleared to 0
  - ◆ V (overflow): set to 1 if arithmetic overflow occurs; otherwise, cleared to 0
  - ◆ C (carry): set to 1 if a carry-out results from the operation; otherwise, cleared to 0
- N and Z flags caused by an arithmetic or a logic operation, V and C flags caused by an arithmetic operation

# Addressing Modes

- Programmers use data structures to represent the data used in computations. These include lists, linked lists, array, queues, and so on
- A high-level language enables the programmer to use constants, local and global variables, pointers, and arrays
- When translating a high-level language program into assembly language, the compiler must be able to implement these constructs using the facilities in the instruction set of the computer
- The different ways in which the location of an operand is specified in an instruction are referred to as *addressing modes*

# Generic Addressing Modes

Name	Assembler syntax	Addressing function
Immediate	#Value	Operand=Value
Register	Ri	EA=Ri
Absolute (Direct)	LOC	EA=LOC
Indirect	(Ri)	EA=[Ri]
	(LOC)	EA=[LOC]
Index	X(Ri)	EA=[Ri]+X
Base with index	(Ri, Rj)	EA=[Ri]+[Rj]
Base with index and offset	X(Ri, Rj)	EA=[Ri]+[Rj]+X
Relative	X(PC)	EA=[PC]+X
Autoincrement	(Ri)+	EA=[Ri]; Increment Ri
Autodecrement	-(Ri)	Decrement Ri; EA=[Ri]

EA: effective address

Value: a signed number

# Register, Absolute and Immediate Modes

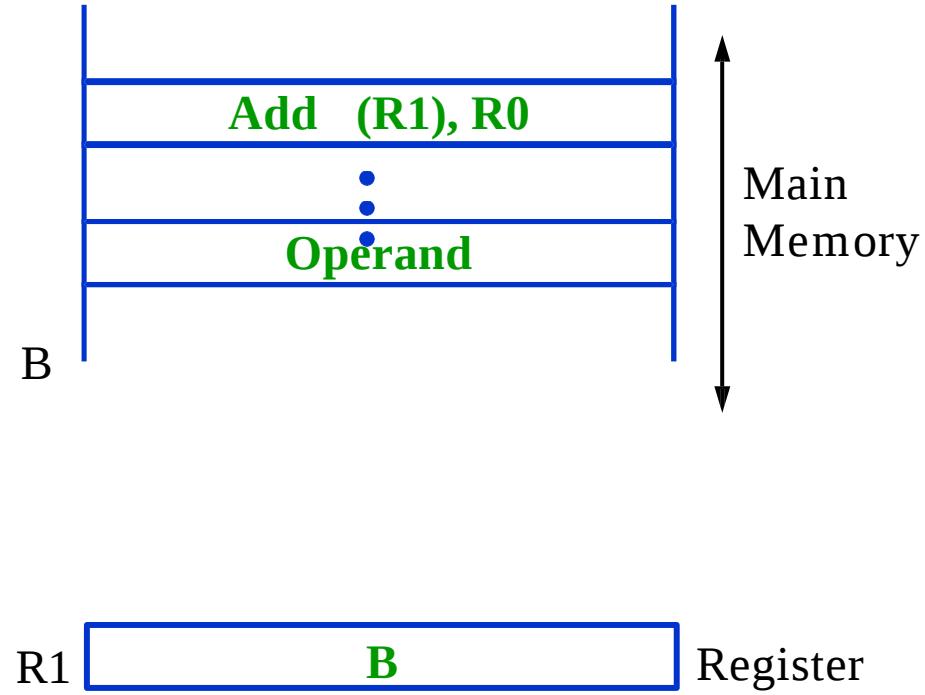
- **Register mode:** the operand is the contents of a processor register; the name (address) of the register is given in the instruction
  - ◆ For example, **Add                  Ri, Rj** (adds the contents of Ri and Rj and the result is stored in Rj)
- **Absolute mode:** the operand is in a memory location; the address of this location is given explicitly in the instruction. (In some assembly languages, this mode is called Direct)
  - ◆ For example, **Move LOC, R2** (moves the content of the memory with address LOC to the register R2)
  - ◆ The Absolute mode can represent global variables in a program.  
For example, a declaration such as Integer A, B;
- **Immediate mode:** the operand is given explicitly in the instruction

# Indirection and Pointers

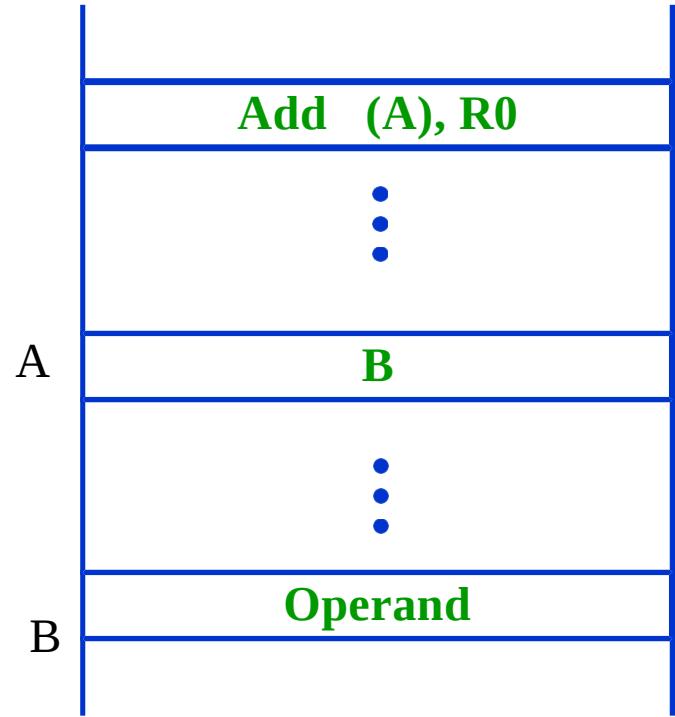
- **Indirect mode:** the effective address of the operand is the contents of a register or memory location whose address appears in the instruction
- Indirection is denoted by placing the name of the register or the memory address given in the instruction in parentheses
- The register or memory location that contains the address of an operand is called a pointer

# Two Types of Indirect Addressing

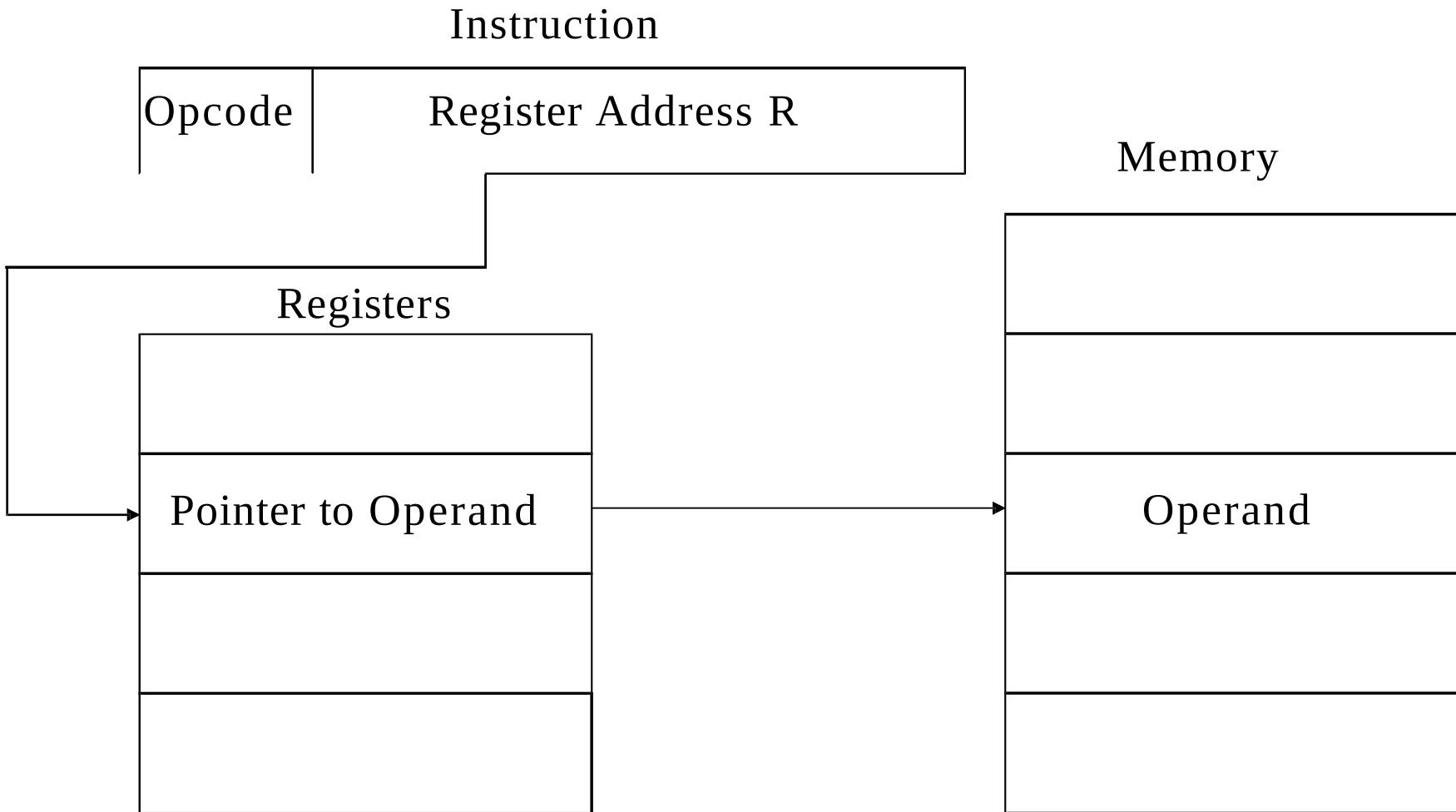
Through a general-purpose register



Through a memory location



# Register Indirect Addressing Diagram



# Using Indirect Addressing in a Program

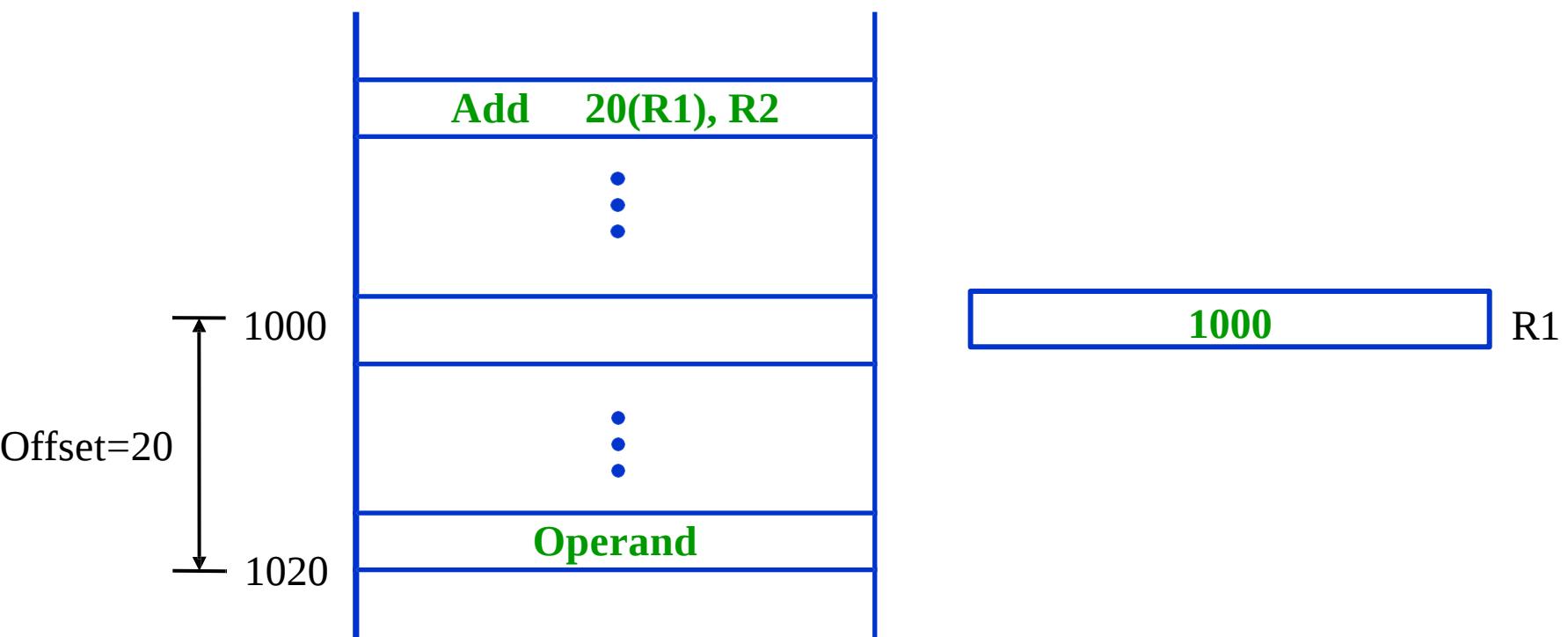
Address	Contents	
	Move	N, R1
	Move	#NUM1, R2
	Move	R0
LOOP	Clear	
	Add	(R2), R0
	Add	#4, R2
	Decrement	R1
	Branch>0	LOOP
	Move	R0, SUM

# **Indexing and Arrays**

- **Index mode:** the effective address of the operand is generated by adding a constant value to the contents of a register
  - ◆ The register used may be either a special register provided for this purpose, or, more commonly, it may be any one of a set of general-purpose registers in the processor. It is referred to as an index register
  - ◆ The index mode is useful in dealing with lists and arrays
  - ◆ We denote the Index mode symbolically as  $X(R_i)$ , where  $X$  denotes the constant value contained in the instruction and  $R_i$  is the name of the register involved. The effective address of the operand is given by  $EA = X + (R_i)$ . The contents of the index register are not changed in the process of generating the effective address

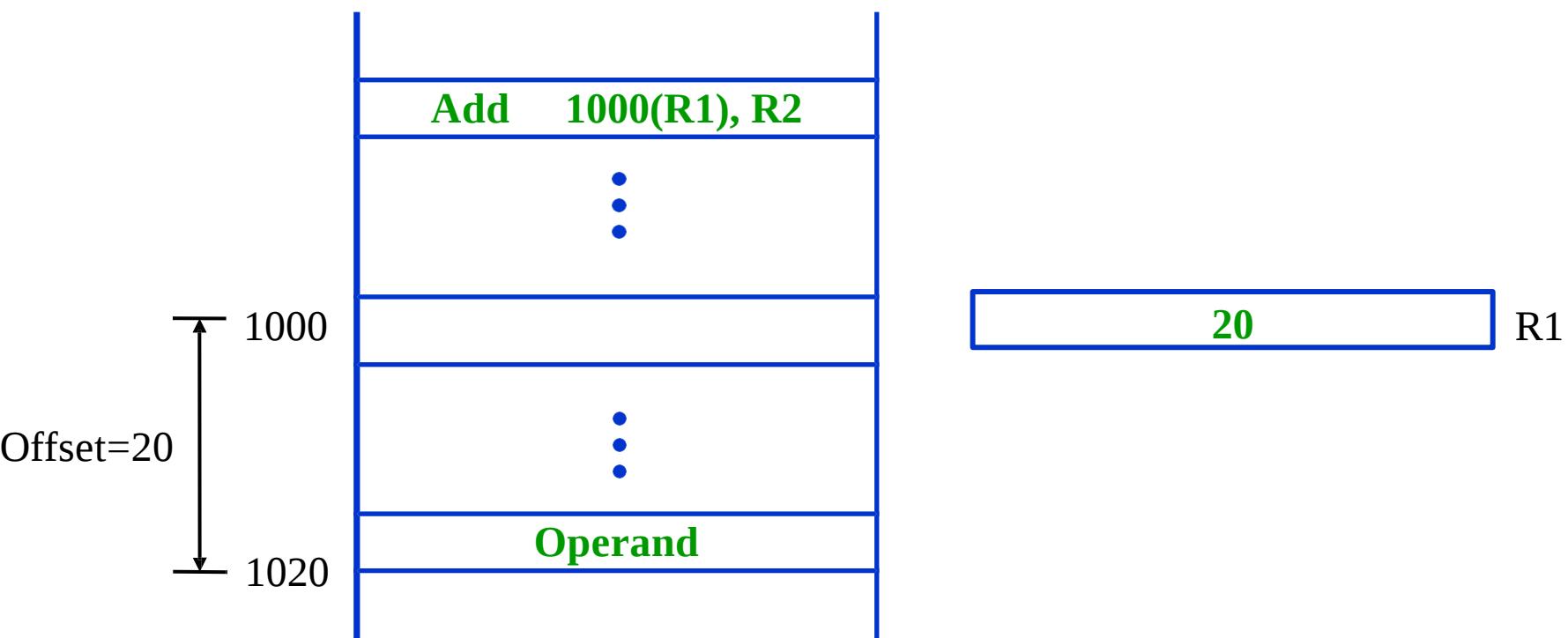
# Indexed Addressing

Offset is given as a constant

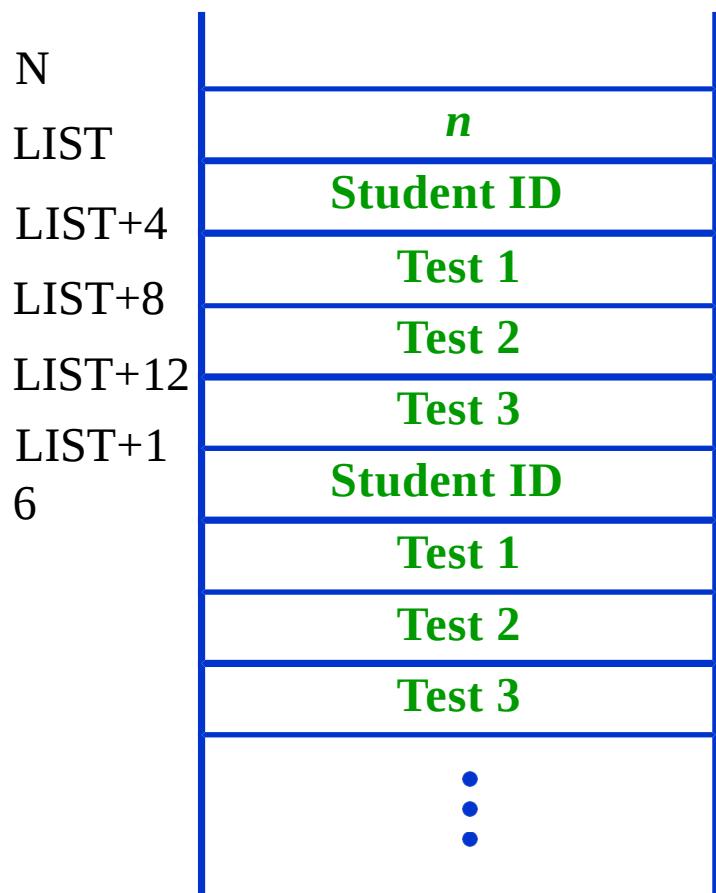


# Indexed Addressing

Offset is in the index register



# An Example for Indexed Addressing



Move	#LIST, R0
Clear	R1
Clear	R2
Clear	R3
Move	N, R4
→LOOP Add	4(R0), R1
Add	8(R0), R2
Add	12(R0), R3
Add	#16, R0
Decrement	R4
Branch>0	LOOP
Move	R1, SUM1
Move	R2, SUM2
Move	R3, SUM3

# Variations of Indexed Addressing Mode

- A second register may be used to contain the offset X, in which case we can write the Index mode as  $(R_i, R_j)$ 
  - ◆ The effective address is the sum of the contents of registers  $R_i$  and  $R_j$
  - ◆ The second register is usually called the base register
  - ◆ This mode implements a two-dimensional array
- Another version of the Index mode uses two registers plus a constant, which can be denoted as  $X(R_i, R_j)$ 
  - ◆ The effective address is the sum of the constant  $X$  and the contents of registers  $R_i$  and  $R_j$
  - ◆ This mode implements a three-dimensional array

# Additional Modes

- Autoincrement mode: the effective address of the operand is the contents of a register specified in the instruction. After accessing the operand, the contents of this register are automatically incremented to point to the next item in a list
  - ◆ The Autoincrement mode is denoted as  $(R_i) +$
- Autodecrement mode: the contents of a register specified in the instruction are first automatically decremented and are then used as the effective address of the operand
  - ◆ The Autodecrement mode is denoted as  $-(R_i)$

# An Example of Autoincrement Addressing

Move	N, R1
	#NUM1, R2
Move	R0
LOOP	(R2)+, R0
Clear	R1
Add	LOOP
Decrement	R0, SUM
Branch>0	
Move	

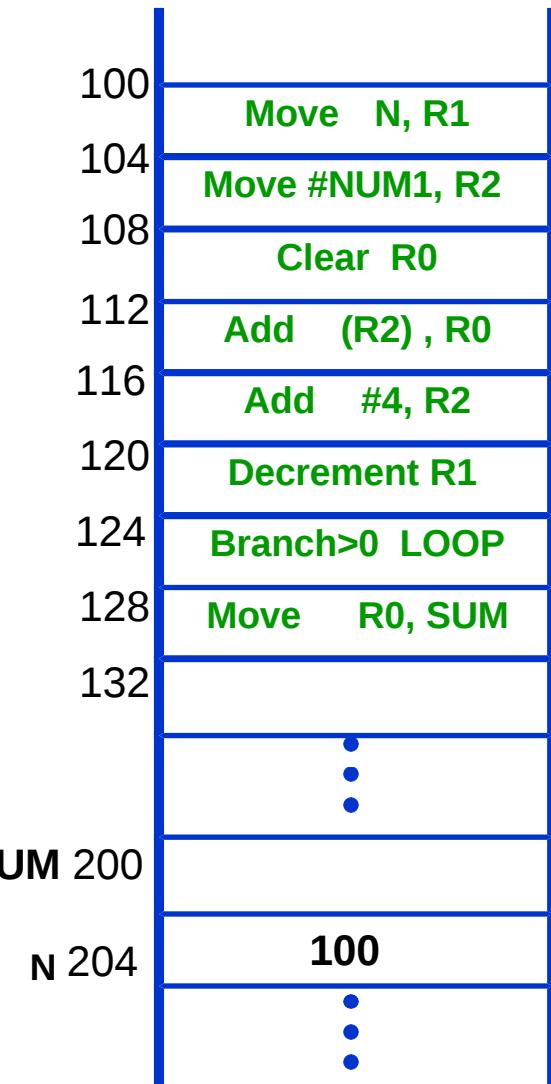
# Assembly Language

- A complete set of symbolic names and rules for their use constitute a programming language, generally referred to as an assembly language
- Programs written in an assembly language can be automatically translated into a sequence of machine instructions by a program called an *assembler*
- When the assembler program is executed, it reads the user program, analyzes it, and then generates the desired machine language program
- The user program in its original alphanumeric text format is called a *source program*, and the assembled machine language program is called an *object program*

# Assembler Directives

- In addition to providing a mechanism for representing instructions in a program, the assembly language allows the programmer to specify other information needed to translate the source program into the object program
- Suppose that the name SUM is used to represent the value 200. The fact may be conveyed to the assembler program through a statement such as
  - ◆ **SUM EQU 200**
- This statement does not denote an instruction that will be executed when the object program is run; it will not even appear in the object program
  - ◆ Such statements, called *assembler directives* (or *commands*)

# Assembler



Memory arrangement

Assembler directives	SUM	EQU	200
		ORIGIN	204
	N	DATAWORD	100
	NUM1	RESERVE	400
Statements that generate machine instructions	START	MOVE	N, R1
		MOVE	#NUM1,
	R2	CLR	R0
	LOOP	ADD	(R2), R0
		ADD	#4, R2
		DEC	R1
		BGTZ	LOOP
		MOVE	R0, SUM
Assembler directives		RETURN	
		END	START

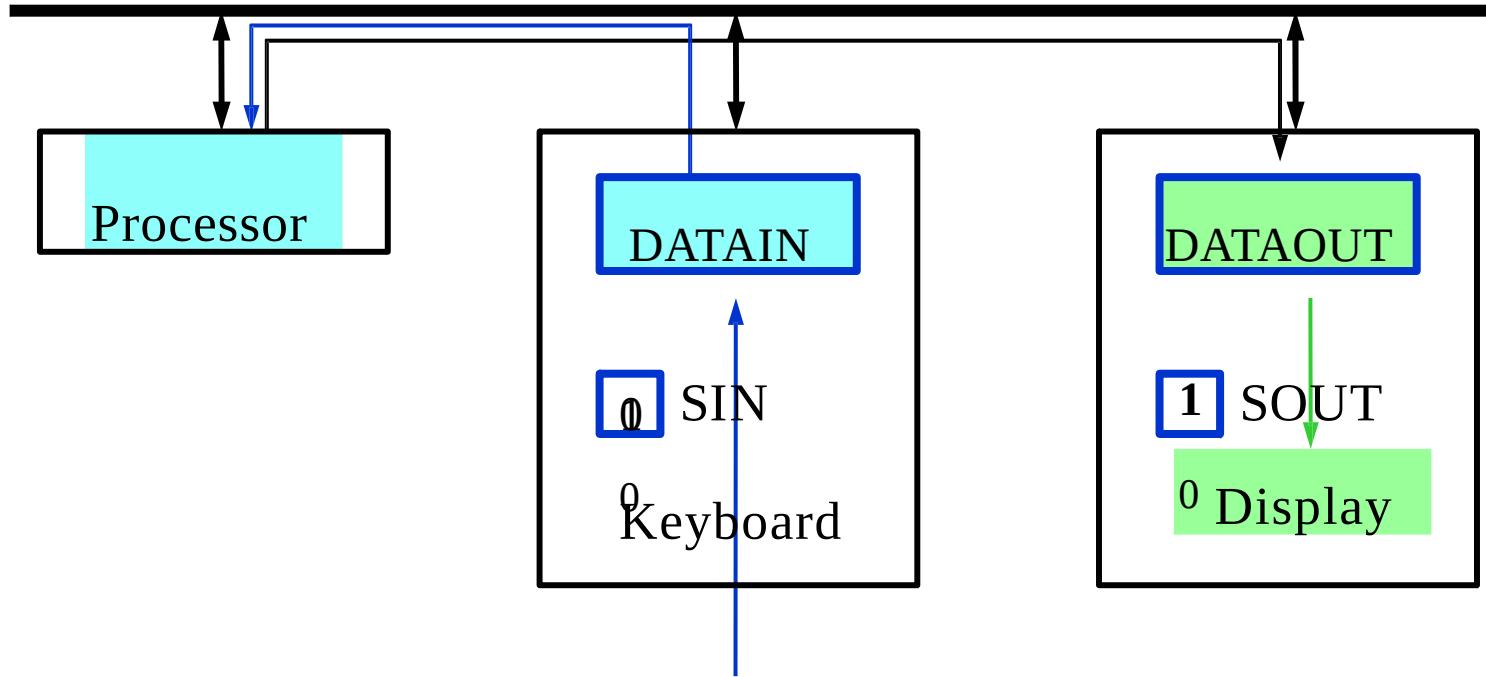
Assembly language representation

# Number Notation

- When dealing with numerical values, most assemblers allow numerical values to be specified in different ways
- For example, consider the number 93, which is represented by the 8-bit binary number 01011101. If the value is to be used as immediate operand,
  - ◆ It can be given as a decimal number , as in the instruction ADD #93, R1
  - ◆ It can be given as a binary number, as in the instruction ADD # %01011101,R1 (a binary number is identified by a prefix symbol such as percent sign)
  - ◆ It also can be given as a hexadecimal number, as in the instruction ADD #\$5D, R1 (a hexadecimal number is identified by a prefix symbol such as dollar sign)

# Basic Input/Output Operations

- Bus connection for processor, keyboard, and display



DATAIN, DATAOUT: buffer registers  
SIN, SOUT: status control flags

# Wait Loop

- In order to perform I / O transfers, we need machine instructions that can check the state of the status flags and transfer data between the processor and I / O device
- Wait loop for Read operation
  - ◆ **READWAIT** Branch to READWAIT if SIN=0  
Input from DATAIN to R1
- Wait loop for Write operation
  - ◆ **WRITEWAIT** Branch to WRITEWAIT if SOUT=0 Output from R1 to DATAOUT
- We assume that the initial state of SIN is 0 and the initial state of SOUT is 1

# Memory-Mapped I/O

- Many computers use an arrangement called memory-mapped I / O in which some memory address values are used to refer to peripheral device buffer registers, such as DATAIN and DATAOUT
- Thus no special instructions are needed to access the contents of these registers; data can be transferred between these registers and the processor using instructions that we have discussed, such as Move, Load, or Store
- Also, the status flags SIN and SOUT can be handled by including them in device status registers, one for each of the two devices

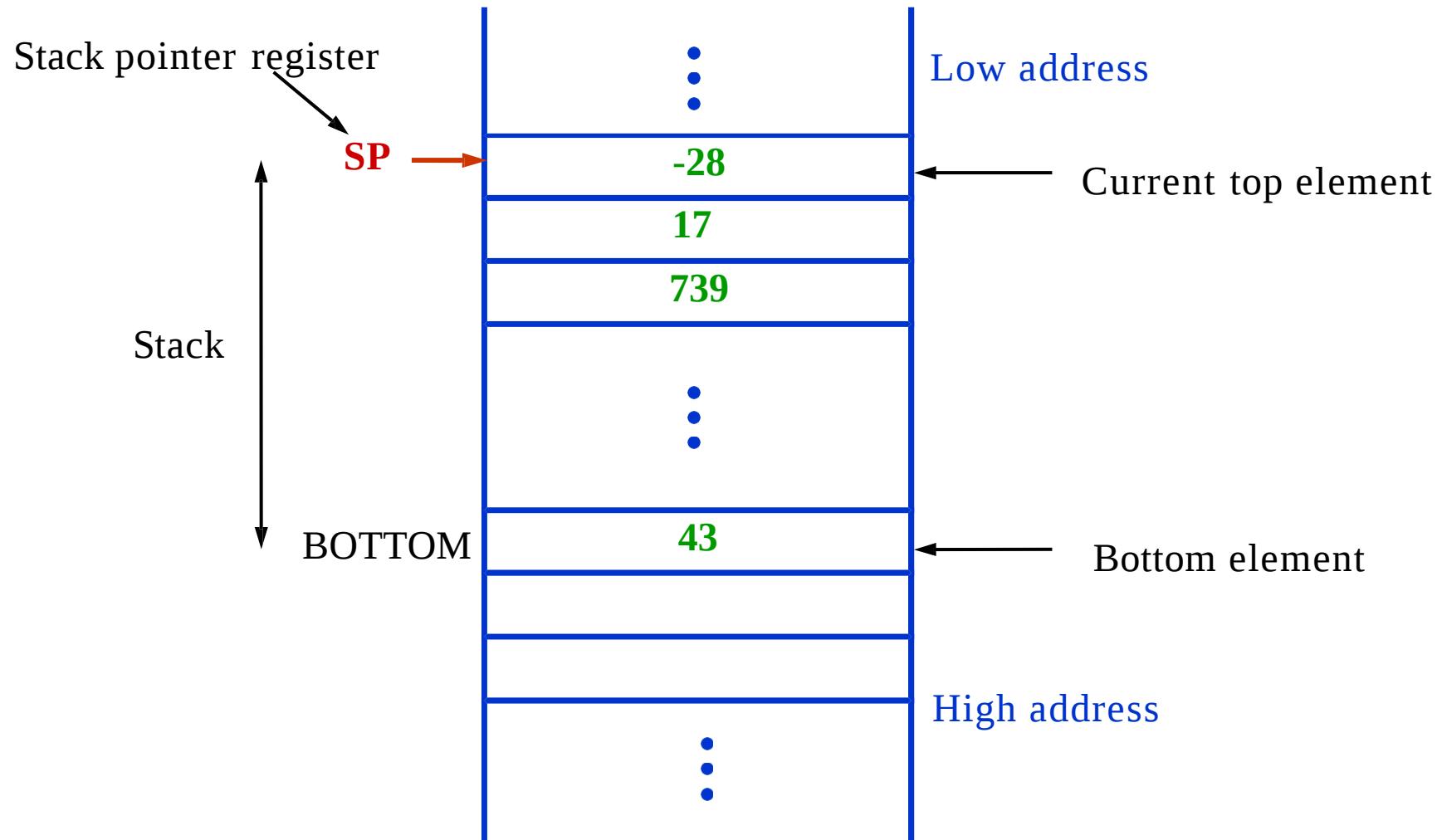
# Read and Write Programs

- Assume that bit b<sub>3</sub> in registers INSTATUS and OUTSTATUS corresponds to SIN and SOUT, respectively
- Read Loop
  - ◆ READWAIT Testbit #3, INSTATUS  
Branch=0 READWAIT  
                  DATAIN, R1
- Write Loop    MoveByte
  - ◆ WRITEWAIT Testbit #3, OUTSTATUS  
Branch=0 WRITEWAIT  
MoveByte R1, DATAOUT

# Stacks and Queues

- A stack is a list of data elements, usually words or bytes, with the accessing restriction that elements can be added or removed at one end of the list only
  - ◆ It is also called a last-in-first-out (LIFO) stack
  - ◆ A stack has two basic operations: push and pop
  - ◆ The terms push and pop are used to describe placing a new item on the stack and removing the top item from the stack, respectively.
- Another useful data structure that is similar to the stack is called a queue
  - ◆ Data are stored in and retrieved from a queue on a first-in-first-out (FIFO) basis
  - ◆ Two pointers are needed to keep track of the two ends of the queue

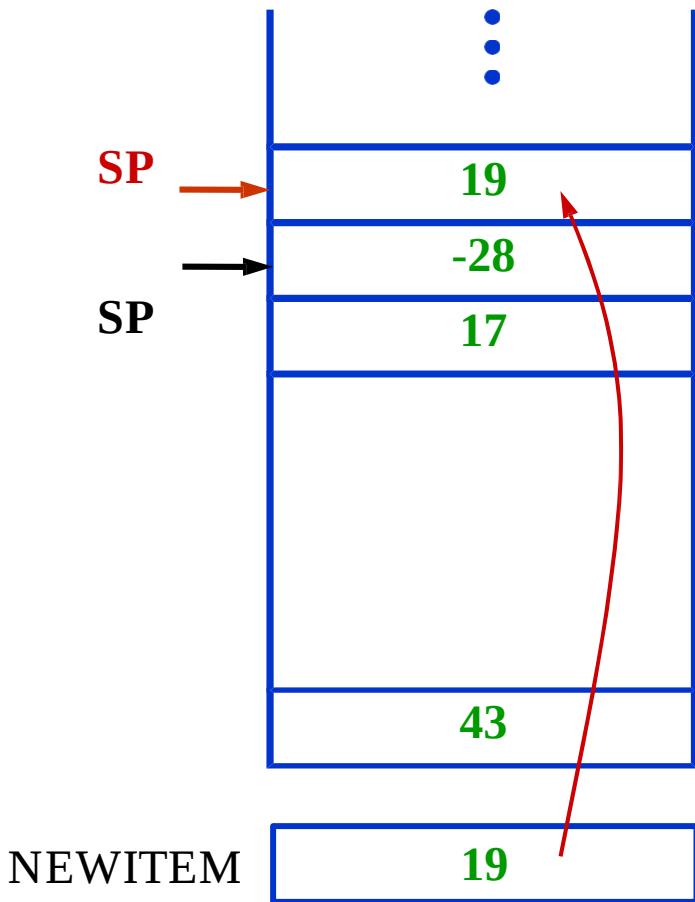
# A Stack of Words in the Memory



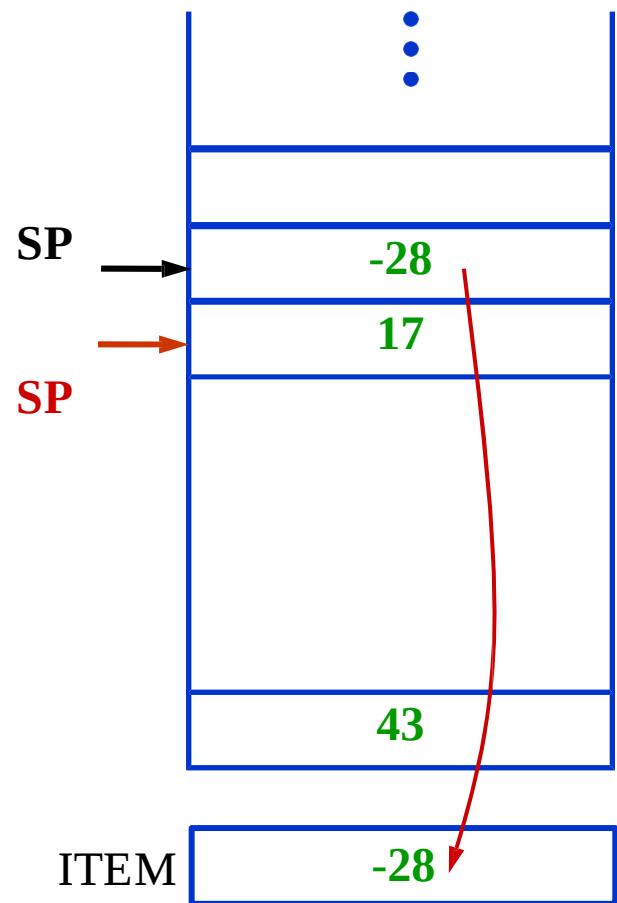
# Push and Pop Operations

- Assume that a byte-addressable memory with 32-bit words
- The push operation can be implemented as
  - Subtract #4, SP
  - Move NEWITEM, (SP)
- The pop operation can be implemented as
  - Move (SP), ITEM Add
  - #4, SP
- If the processor has the Autoincrement and Autodecrement addressing modes, then the push operation can be implemented by the single instruction
  - Move NEWITEM, -(SP)
- And the pop operation can be implemented as
  - Move (SP)+, ITEM

# Examples



Push operation



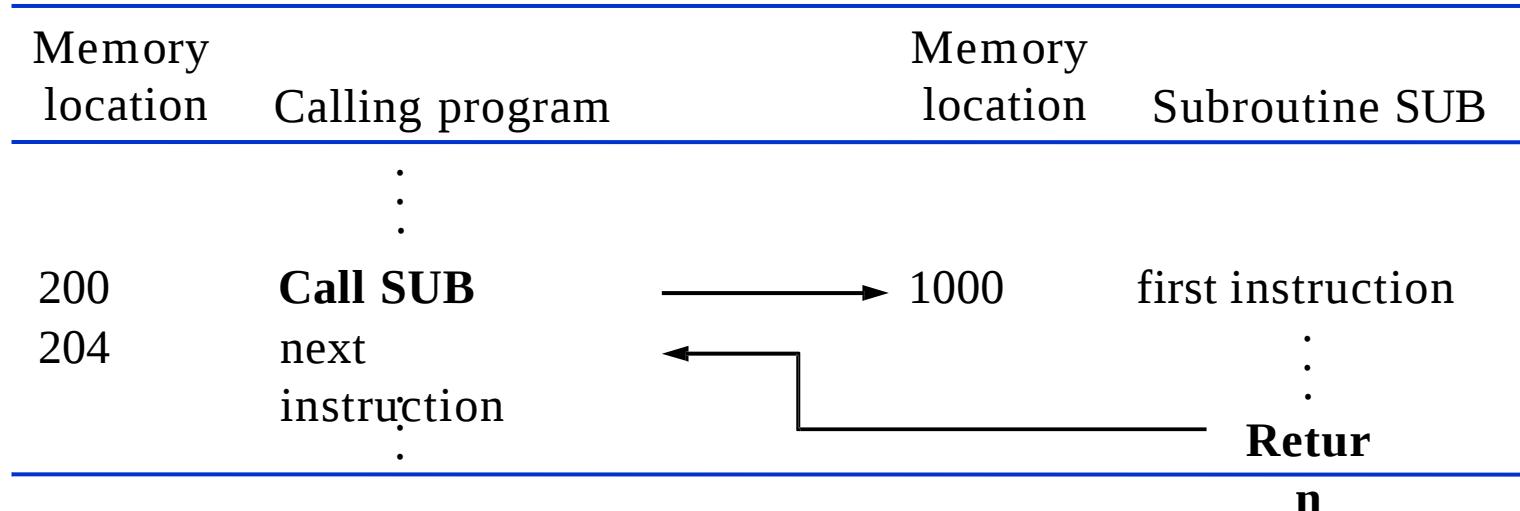
Pop operation

# Checking for Empty and Full Errors

- When a stack is used in a program, it is usually allocated a fixed amount of space in the memory
  - ◆ We must avoid pushing an item onto the stack when the stack has reached its maximum size, i.e., the stack is full
  - ◆ On the other hand, we must avoid popping an item off the stack when the stack has reached its minimum size, i.e., the stack is empty
- Routine for a safe pop or a safe push operation
  - ◆ Compare src, dst
  - ◆ Perform [dst]-[src]
  - ◆ Sets the condition code flags according to the result

# Subroutines

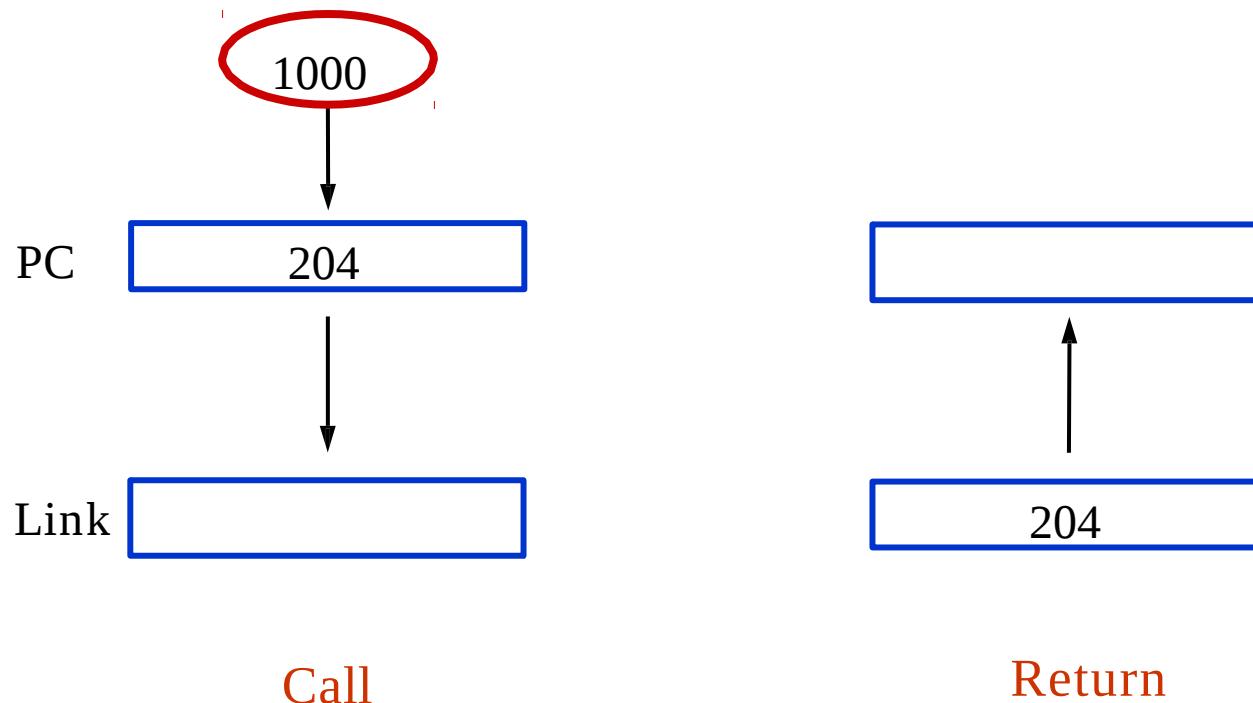
- In a given program, it is often necessary to perform a particular subtask many times on different data values. Such a subtask is called a *subroutine*.



- The location where the calling program resumes execution is the location pointed by the updated PC while the Call instruction is being executed. Hence the contents of the PC must be saved by the Call instruction to enable correct return to the calling program

# Subroutine Linkage

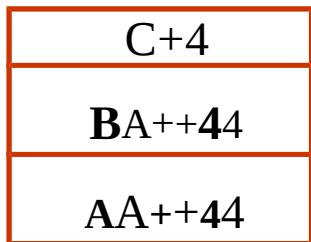
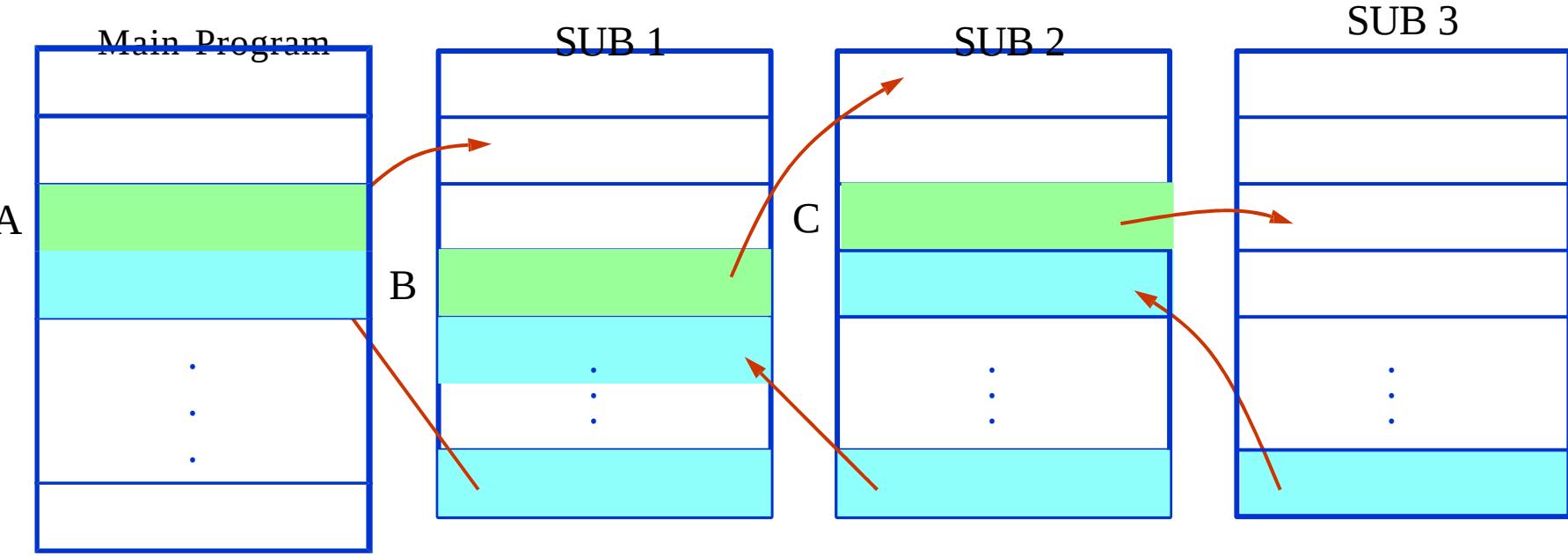
- The way in which a computer makes it possible to call and return from subroutines is referred to as its subroutine linkage method
- Subroutine linkage using a link register



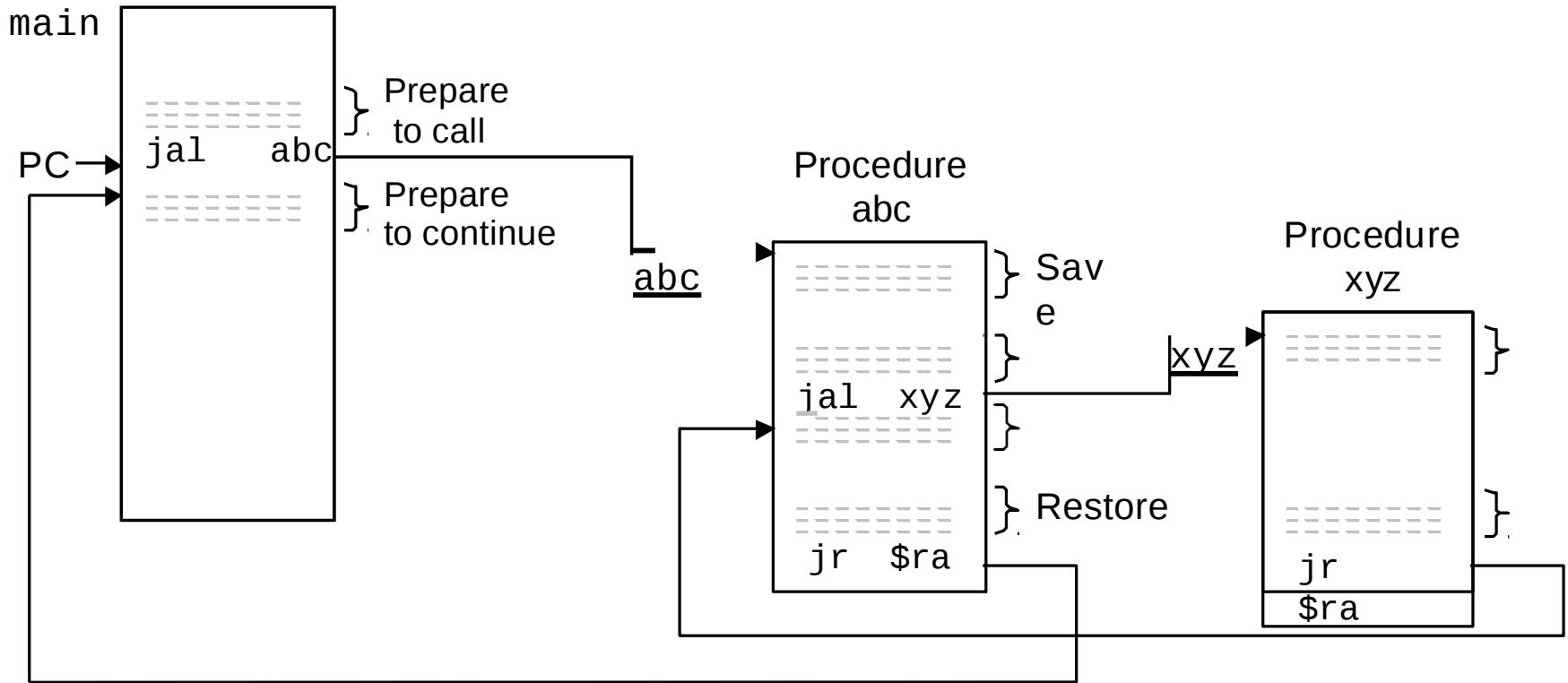
# Subroutine Nesting

- A common programming practice, called subroutine nesting, is to have one subroutine call another
- Subroutine nesting call be carried out to any depth. Eventually, the last subroutine called completes its computations and returns to the subroutine that called it
- The return address needed for this first returns is the last one generated in the nested call sequence. That is, return addresses are generated and used in a last-in-first-out order
- Many processors do this by using a stack pointer and the stack pointer points to a stack called the processor stack

# Example of Subroutine Nesting



# Example of Subroutine Nesting



[Source: B. Parhami, UCSB]

# Parameter Passing

- When calling a subroutine, a program must provide to the subroutine the parameters, that is, the operands or their addresses, to be used in the computation. Later, the subroutine returns other parameters, in this case, the result of computation
- The exchange of information between a calling program and a subroutine is referred to as parameter passing
- Parameter passing approaches
  - ◆ The parameters may be placed in registers or in memory locations, where they can be accessed by the subroutine
  - ◆ The parameters may be placed on the processor stack used for saving the return address

# Passing Parameters with Registers

---

				passed by value	passing by reference
<b>Calling program</b>					
Move	N, R1			R1 serves as a counter	
	#NUM1, R2			R2 points to the list	
Move	LISTADD			Call subroutine	
Call	R0, SUM			Save result	
M ove					
:					
<b>Subroutine</b>					
LISTADD	Clear	R0		Initialize sum to 0	
LOOP	Add	(R2)+, R0		Add entry from list	
	Decrement	R1			
	Branch>0	LOOP			
	Return			Return to calling program	

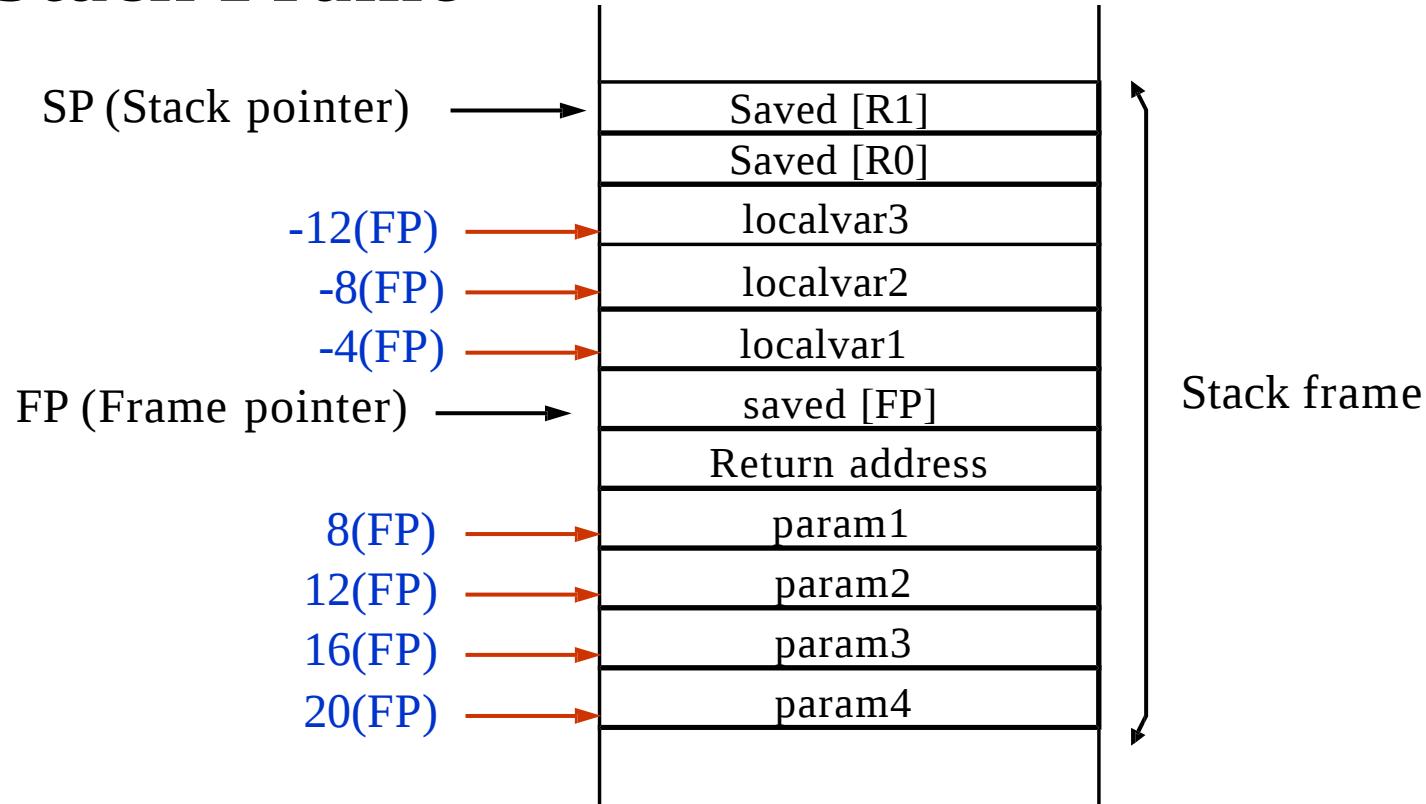
---

# Passing Parameters with Stack

Assume top of stack is at level 1 below.

Move	#NUM1, -(SP)	Push parameters onto stack
Move	N, -(SP)	
Call	LISTADD	Call subroutine (top of stack at level 2)
Move	4(SP), SUM	Save result
Add	#8, SP	
	.	Restore top of stack (top of stack at level 1)
	:	
	:	
<b>LISTADD</b>	MoveMultiple R0-R2, -(SP)	Save registers (top of stack at level 3)
	Move 16(SP), R1	Initialize counter to N.
	20(SP), R2	Initialize pointer to the list
	Move R0	Initialize sum to 0
	(R2)+, R0	Add entry from list
<b>LOOP</b>	Clear R1	
	Add LOOP	
	Decrement R0,	Put result on the stack
	Branch>0 20(SP)	Restore registers
	Move (SP)+,	Return to calling
	MoveMultiple R0-R2	program
	Return	

# Stack Frame



# Shift Instructions

- Logical shifts

## Logic shift left

LShiftL  
#2, R0



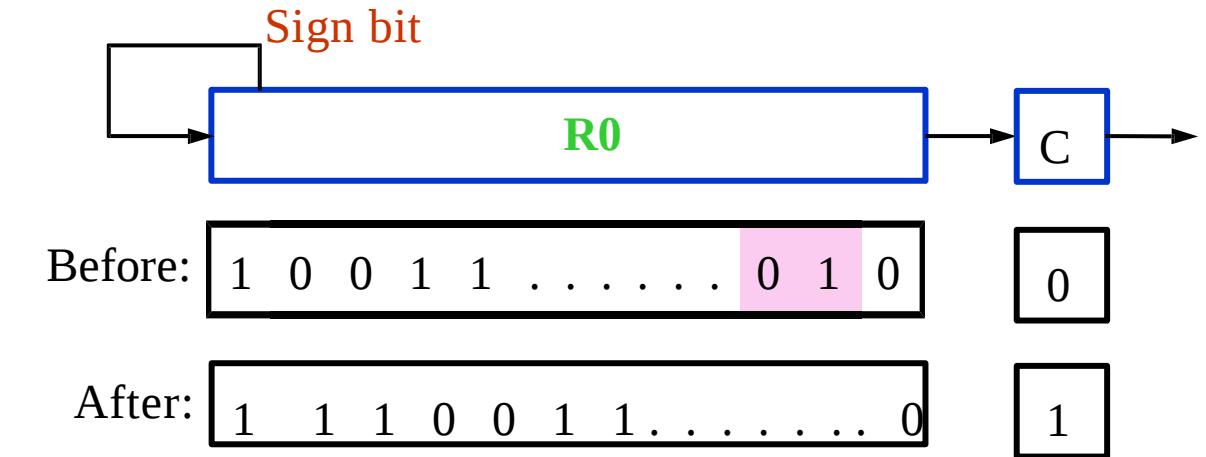
Before: 0 1 1 1 0 . . . . . 0 1 1

After: 1 1 0 . . . . . 0 1 1 0 0

- Arithmetic shifts

## shift right

AShiftR  
#2,  
R0



Before: 1 0 0 1 1 . . . . . 0 1 0 0

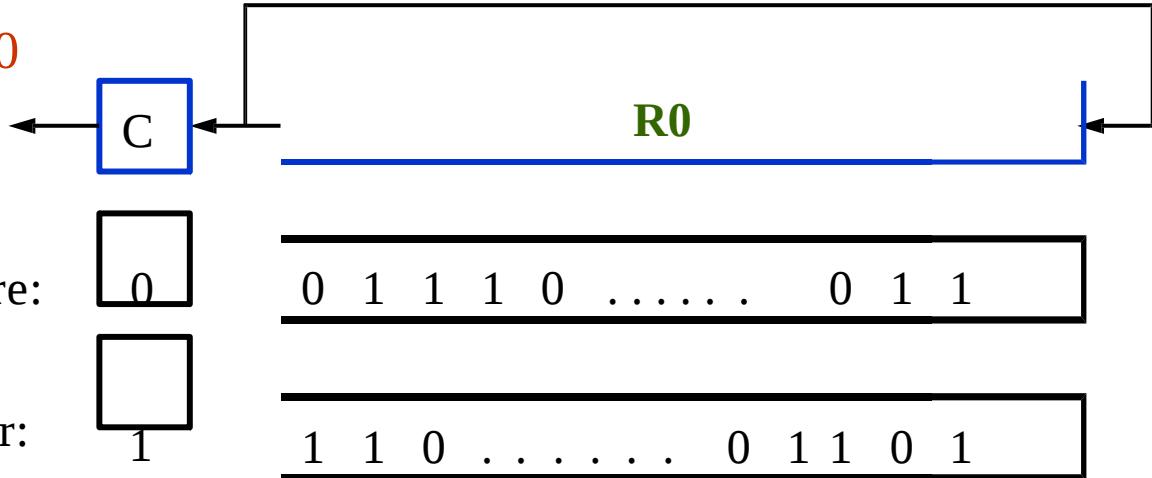
After: 1 1 1 0 0 1 1 . . . . . 0 1

# Rotate Instructions

- Rotate left without carry

RotateL

#2, R0

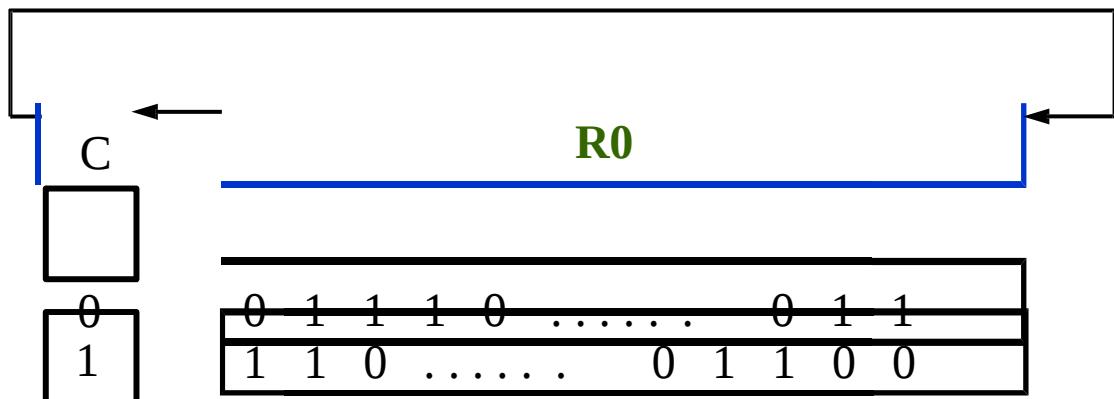


- Rotate left with carry

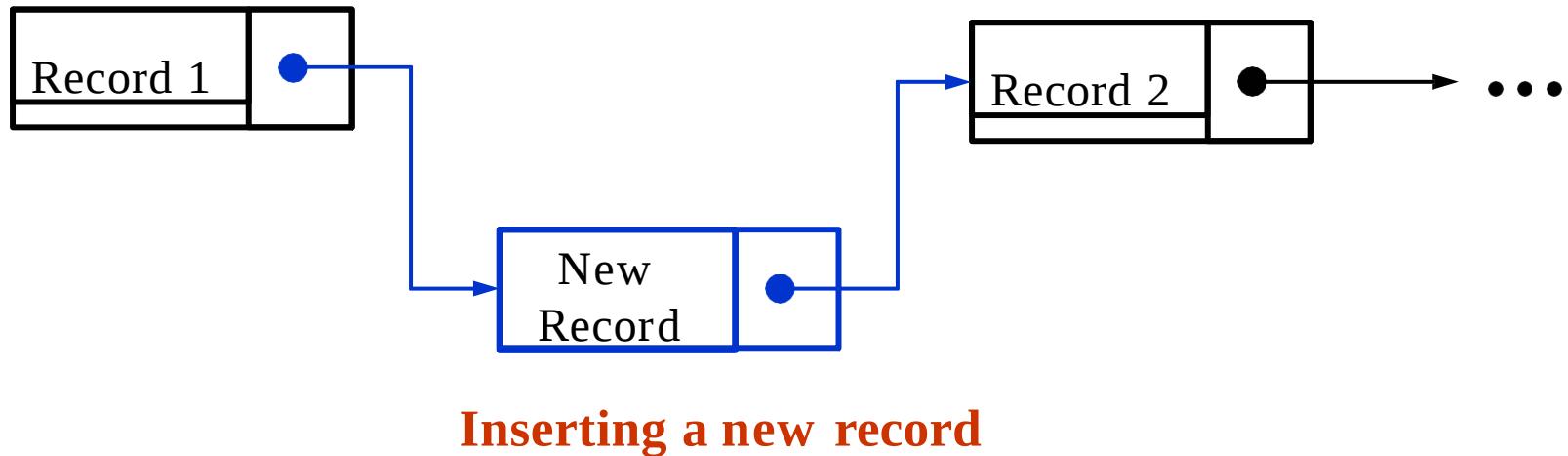
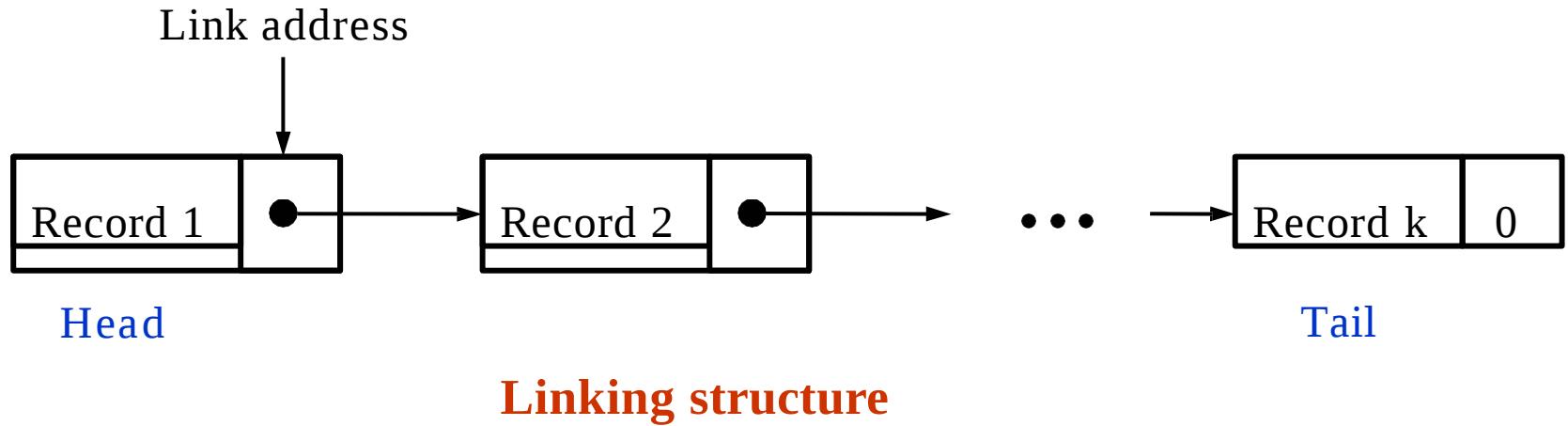
RotateLC

#2, R0

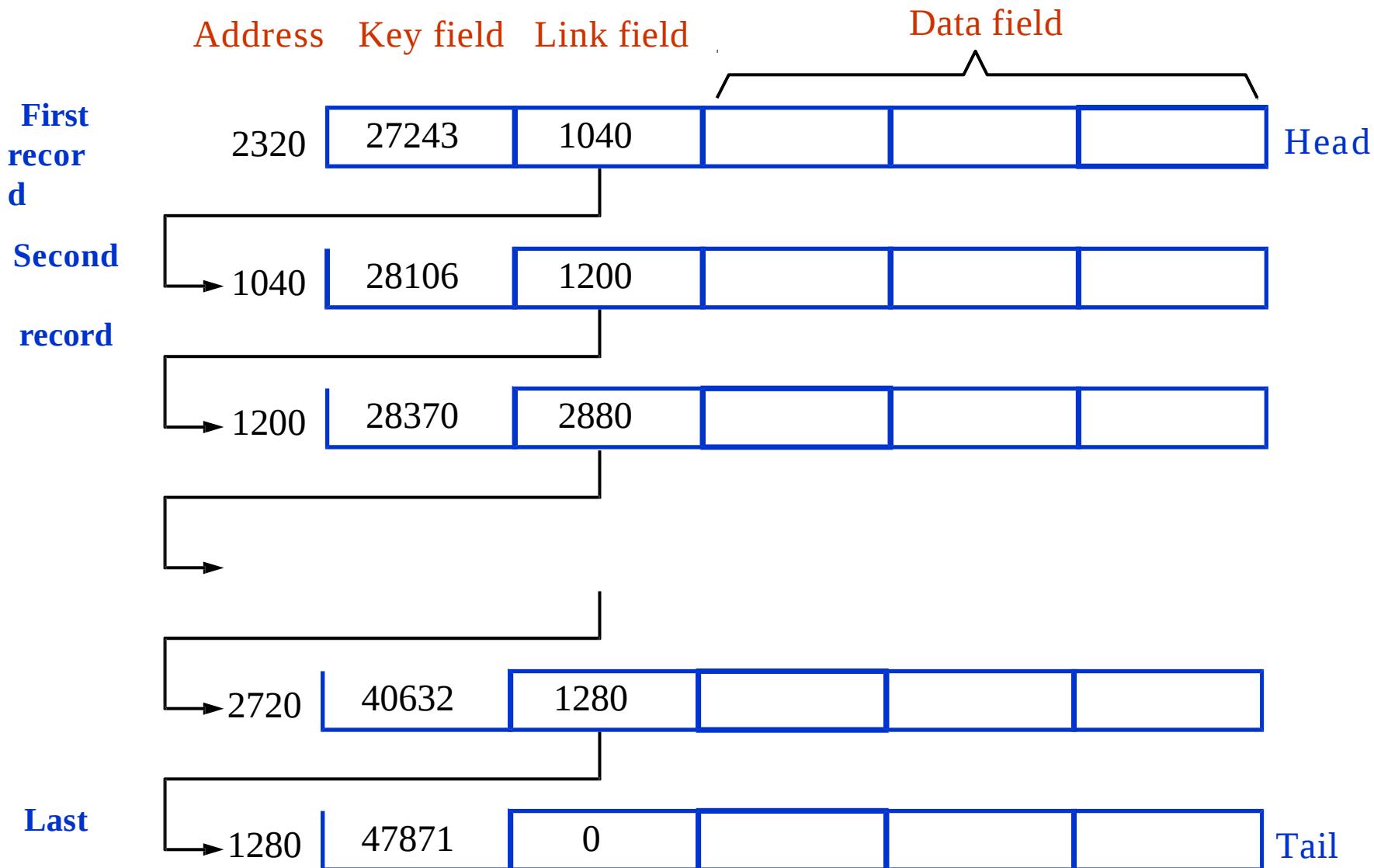
Before:  
After:



# Linked List



# A List of Student Test Scores



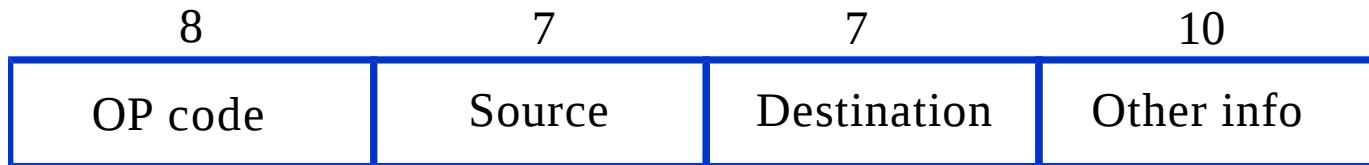
# Encoding of Machine Instructions

- To be executed in a processor, an instruction must be encoded in a compact binary pattern. Such encoded instructions are properly referred to as machine instructions. The instructions that use symbolic names and acronyms are called assembly language instructions, which are converted into the machine instructions using assembler program
- For a given instruction, the type of operation that is to be performed and the type of operands used may be specified using an encoded binary pattern referred to as the OP code
- In addition to the OP code, the instruction has to specify the source and destination registers, and addressing mode, etc,

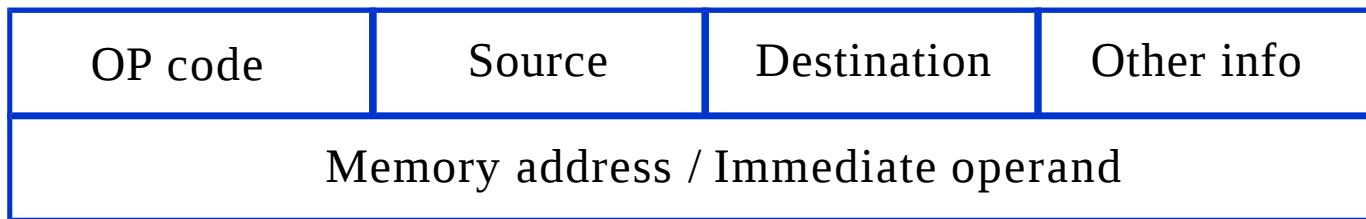
# Examples

- Assume that 8 bits are allocated for OP code, and 4 bits are needed to identify each register, and 6 bits are needed to specify an addressing mode
- The instruction Move 24(R0), R5
  - ◆ Require 16 bits to denote the OP code and the two registers
  - ◆ Require 6 bits to choose the addressing mode
  - ◆ Only 10 bits are left to give the index value
- The instruction LshiftR #2, R0
  - ◆ Require 18 bits to specify the OP code, the addressing modes, and the register
  - ◆ This limits the size of the immediate operand to what is expressible in 14 bits
- In the two examples, the instructions can be encoded in a 32-bit word.

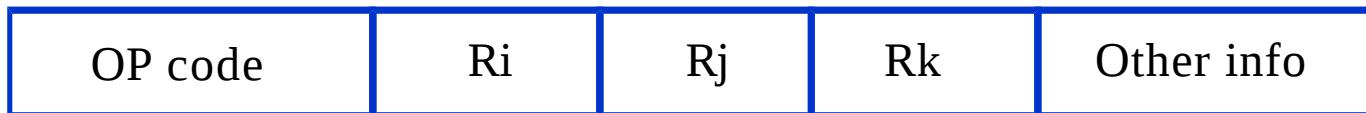
# Encoding Instructions into 32-bit Words



One-word instruction



Two-word instruction



Three-operand instruction

# Encoding Instructions into 32-bit Words

- But, what happens if we want to specify a memory operand using the Absolute addressing mode?
- The instruction Move R2, LOC
  - ◆ Require 18 bits to denote the OP code, the addressing modes, and the register
  - ◆ The leaves 14 bits to express the address that corresponds to LOC, which is clearly insufficient
- If we want to be able to give a complete 32-bit address in the instruction, an instruction must have two words
- If we want to handle this type of instructions: Move LOC1, LOC2
  - ◆ An instruction must have three words

# CISC & RISC

- Using multiple words, we can implement quite complex instructions, closely resembling operations in high-level programming language
- The term *complex instruction set computer* (CISC) has been used to refer to processors that use instruction sets of this type
- The restriction that an instruction must occupy only one word has led to a style of computers that have become known as *reduced instruction set computer* (RISC)

Topic: Cache Memory

# Characteristics of Memory

## “Location wrt Processor”

- Inside CPU – temporary memory or registers
- Inside processor – L1 cache
- Motherboard – main memory and L2 cache
- Main memory – DRAM and L3 cache
- External – peripherals such as disk, tape, and networked memory devices

# Characteristics of Memory

## “Capacity – Word Size”

- The natural data size for a processor.
- A 32-bit processor has a 32-bit word.
- Typically based on processor's data bus width (i.e., the width of an integer or an instruction)
- Varying widths can be obtained by putting memory chips in parallel with same address lines

# Characteristics of Memory

## “Capacity - Addressable Units”

- Varies based on the system's ability to allow addressing at byte level etc.
- Typically smallest location which can be uniquely addressed
- At mother board level, this is the word
- It is a cluster on disks
- Addressable units (N) equals  $2^{\text{number of bits in the address bus}}$

# Characteristics of Memory

## “Unit of transfer”

- The number of bits read out of or written into memory at a time.
- Internal – Usually governed by data bus width, i.e., a word
- External – Usually a block which is much larger than a word

# Characteristics of Memory

## “Access method”

- Based on the hardware implementation of the storage device
- Four types
  - Sequential
  - Direct
  - Random
  - Associative

# Sequential Access Method

- Start at the beginning and read through in order
- Access time depends on location of data and previous location
- Example: tape

# Direct Access Method

- Individual blocks have unique address
- Access is by jumping to vicinity then performing a sequential search
- Access time depends on location of data within "block" and previous location
- Example: hard disk

# Random Access Method

- Individual addresses identify locations exactly
- Access time is consistent across all locations and is independent previous access
- Example: RAM

# Associative Access Method

- Addressing information must be stored with data in a general data location
- A specific data element is located by comparing desired address with address portion of stored elements
- Access time is independent of location or previous access
- Example: cache

# Performance - Access Time

- Time between "requesting" data and getting it
- RAM
  - Time between putting address on bus and getting data.
  - It's predictable.
- Other types, Sequential, Direct, Associative
  - Time it takes to position the read-write mechanism at the desired location.
  - Not predictable.

# Performance – Memory Cycle time

- Primarily a RAM phenomenon
- Adds "recovery" time to cycle allowing for transients to dissipate so that next access is reliable.
- Cycle time is access + recovery

# Performance - Transfer Rate

- Rate at which data can be moved
- RAM – Predictable; equals  $1/(\text{cycle time})$
- Non-RAM – Not predictable; equals

$$T_N = T_A + (N/R)$$

where

- $T_N$  = Average time to read or write N bits
- $T_A$  = Average access time
- N = Number of bits
- R = Transfer rate in bits per second

# Physical Types

- Semiconductor – RAM
- Magnetic – Disk & Tape
- Optical – CD & DVD
- Others
  - Bubble (old) – memory that made a "bubble" of charge in an opposite direction to that of the thin magnetic material that on which it was mounted
  - Hologram (new) – much like the hologram on your credit card, laser beams are used to store computer-generated data in three dimensions. (10 times faster with 12 times the density)

# Physical Characteristics

- Decay
  - Power loss
  - Degradation over time
- Volatility – RAM vs. Flash
- Erasable – RAM vs. ROM
- Power consumption – More specific to laptops, PDAs, and embedded systems

# Organization

- Physical arrangement of bits into words
- Not always obvious
- Non-sequential arrangements may be due to speed or reliability benefits, e.g. interleaved

# Memory Hierarchy

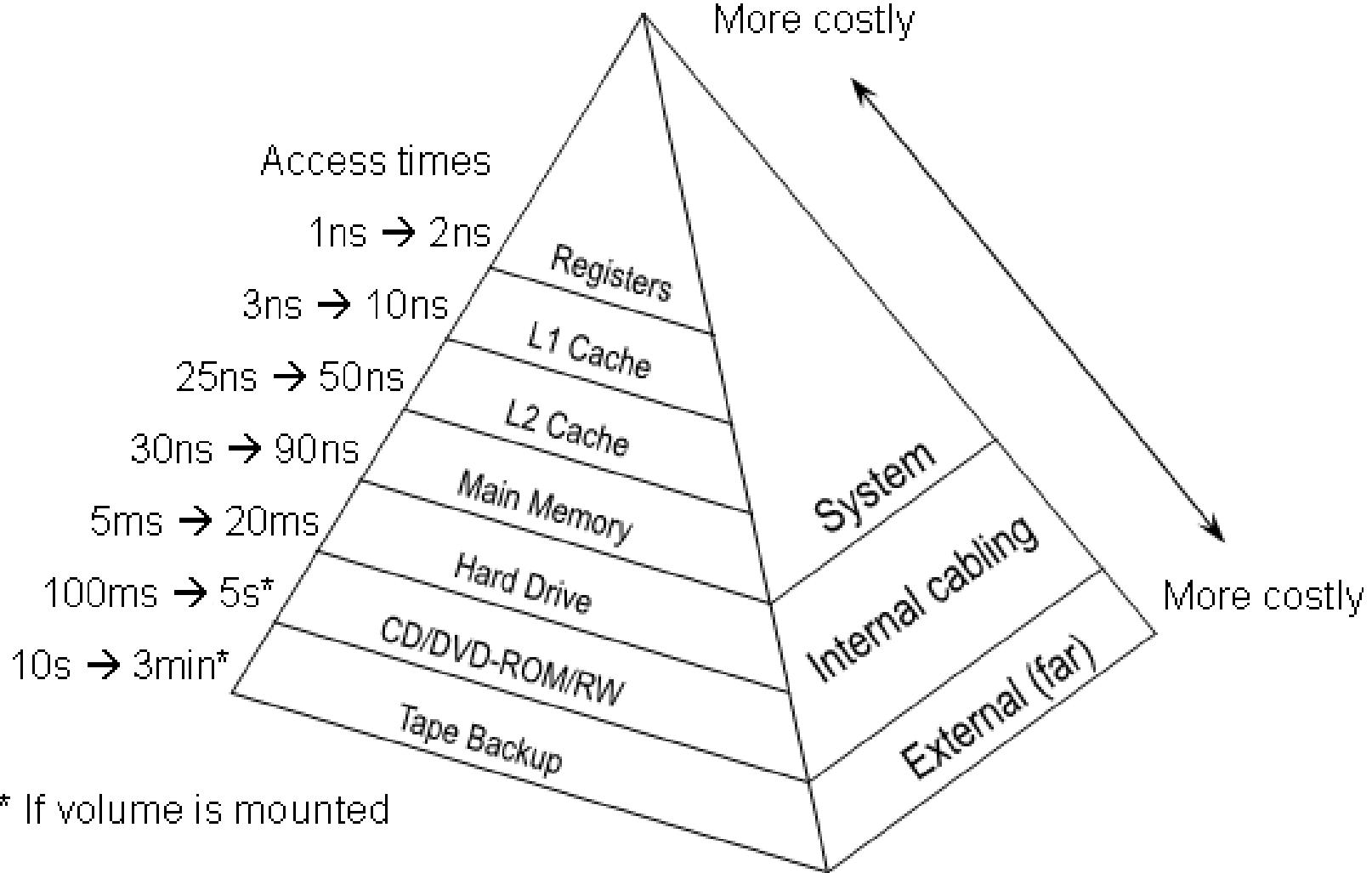
- Trade-offs among three key characteristics
  - Amount – Software will ALWAYS fill available memory
  - Speed – Memory should be able to keep up with the processor
  - Cost – Whatever the market will bear
- Balance these three characteristics with a memory hierarchy
- Analogy –  
Refrigerator & cupboard (fast access – lowest variety)  
freezer & pantry (slower access – better variety)  
grocery store (slowest access – greatest variety)

# Memory Hierarch (continued)

Implementation – Going down the hierarchy has the following results:

- Decreasing cost per bit (cheaper)
- Increasing capacity (larger)
- Increasing access time (slower)
- **KEY** – Decreasing frequency of access of the memory by the processor

# Memory Hierarchy (continued)



\* If volume is mounted

Source: Null, Linda and Lobur, Julia (2003). *Computer Organization and Architecture* (p. 236). Sudbury, MA: Jones and Bartlett Publishers.

# Mechanics of Technology

- The basic mechanics of creating memory directly affect the first three characteristics of the hierarchy:
  - Decreasing cost per bit
  - Increasing capacity
  - Increasing access time
- The fourth characteristic is met because of a principle known as **locality of reference**

# Locality of Reference

Due to the nature of programming, instructions and data tend to cluster together (loops, subroutines, and data structures)

- Over a long period of time, clusters will change
- Over a short period, clusters will tend to be the same

# Breaking Memory into Levels

- Assume a hypothetical system has two levels of memory
  - Level 2 should contain all instructions and data
  - Level 1 doesn't have room for everything, so when a new cluster is required, the cluster it replaces must be sent back to the level 2
- These principles can be applied to much more than just two levels
- If performance is based on amount of memory rather than speed, lower levels can be used to simulate larger sizes for higher levels, e.g., virtual memory

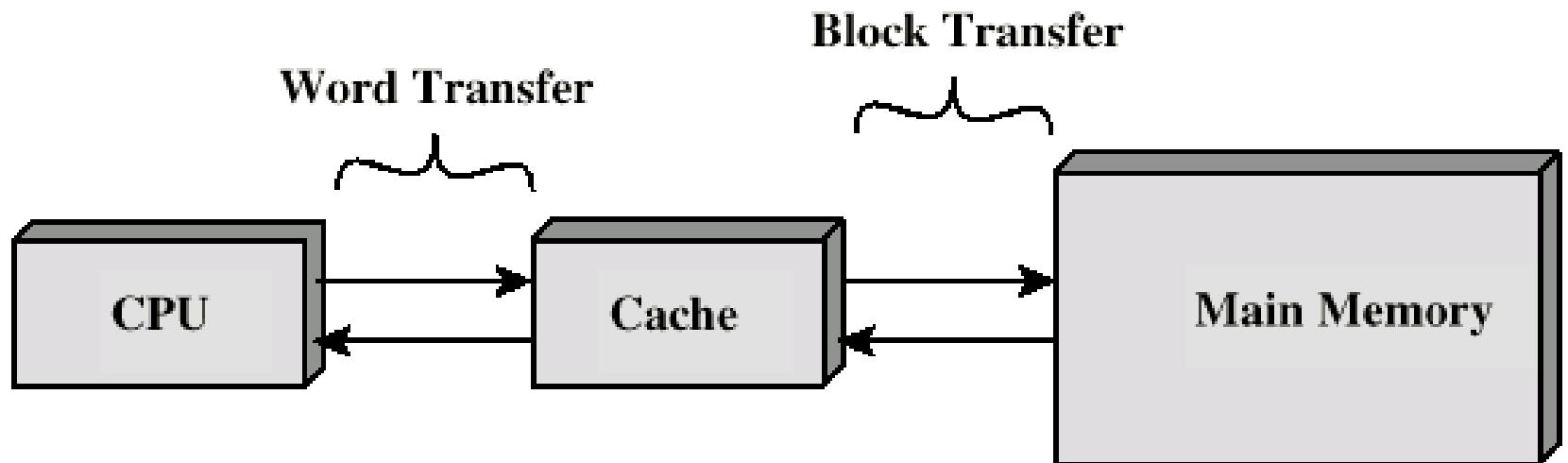
# Hierarchy List

- Registers – volatile
- L1 Cache – volatile
- L2 Cache – volatile
- CDRAM (main memory) cache – volatile
- Main memory – volatile
- Disk cache – volatile
- Disk – non-volatile
- Optical – non-volatile
- Tape – non-volatile

# Cache

- What is it? A cache is a small amount of fast memory
- What makes small fast?
  - Simpler decoding logic
  - More expensive SRAM technology
  - Close proximity to processor – Cache sits between normal main memory and CPU or it may be located on CPU chip or module

# Cache (continued)



# Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, one of two things happens:
  - read required block from main memory to cache then deliver from cache to CPU (cache physically between CPU and bus)
  - read required block from main memory to cache and simultaneously deliver to CPU (CPU and cache both receive data from the same data bus buffer)

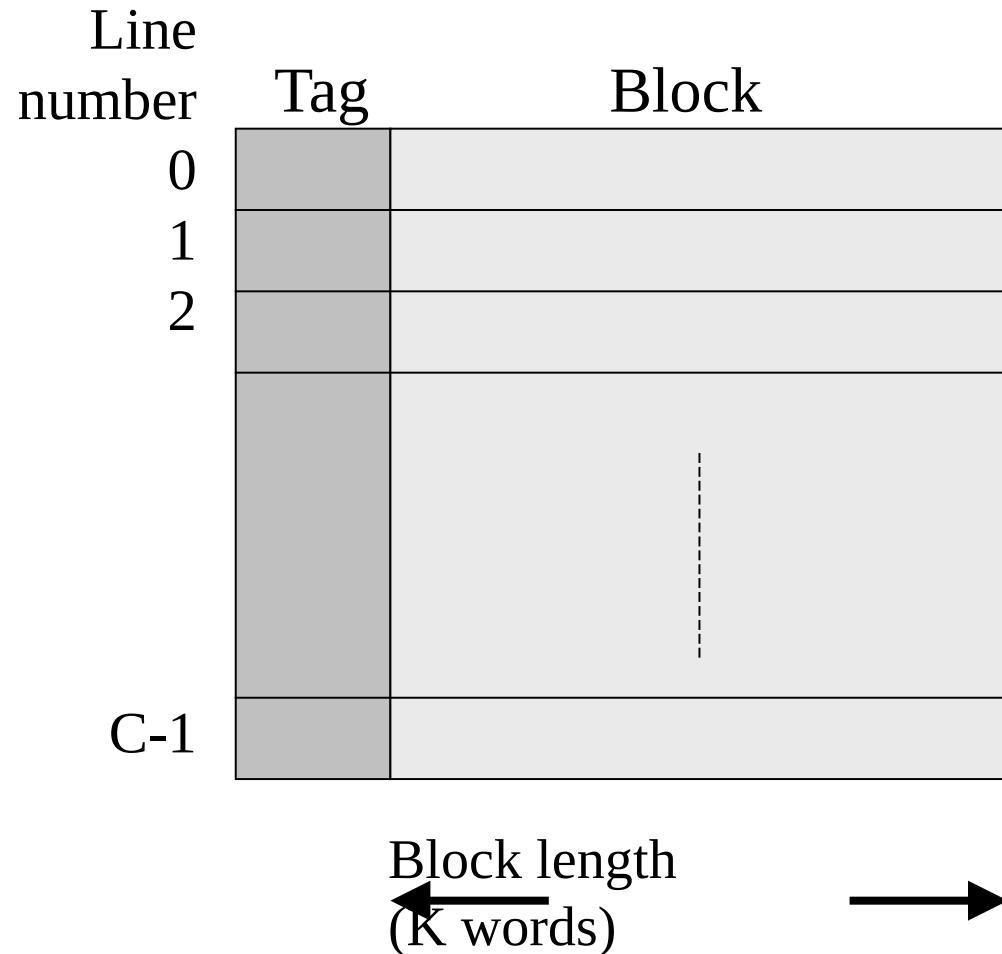
# Going Deeper with Principle of Locality

- Cache "misses" are unavoidable, i.e., every piece of data and code thing must be loaded at least once
- What does a processor do during a miss? It waits for the data to be loaded.
- Power consumption varies linearly with clock speed and the square of the voltage.
- Adjusting clock speed and voltage of processor has the potential to produce cubic (cubed root) power reductions (<http://www.visc.vt.edu/~mhsiao/papers/pacs00ch.pdf>)
- Identify places in in-class exercise where this might happen.

# Cache Structure

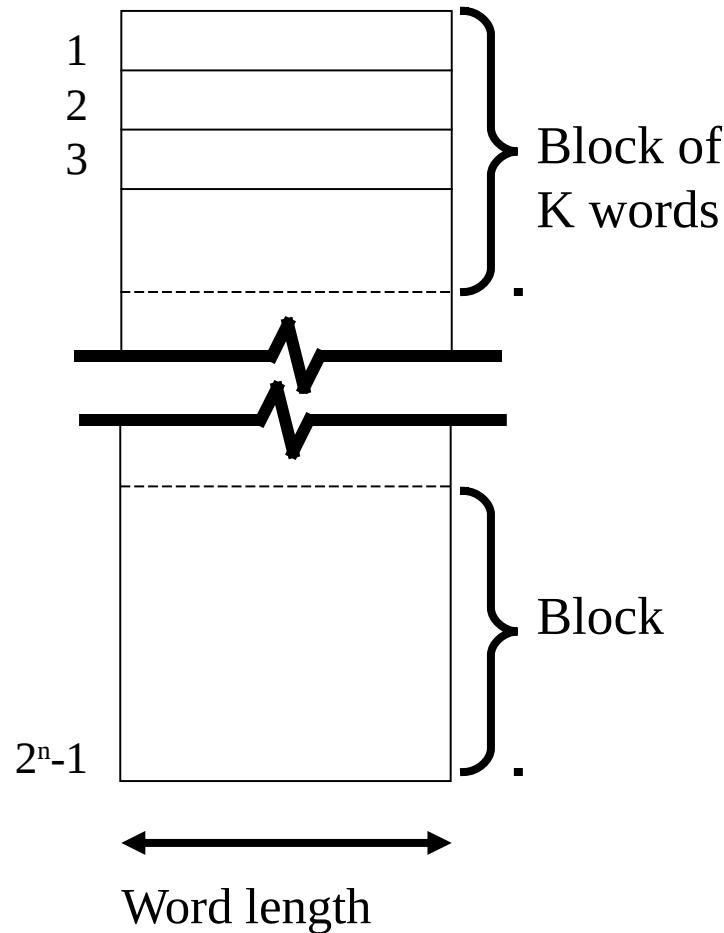
- Cache includes tags to identify the address of the block of main memory contained in a line of the cache
- Each word in main memory has a unique n-bit address
- There are  $M=2^n/K$  block of K words in main memory
- Cache contains C lines of K words each plus a tag uniquely identifying the block of K words

# Cache Structure (continued)



# Memory Divided into Blocks

Memory  
Address



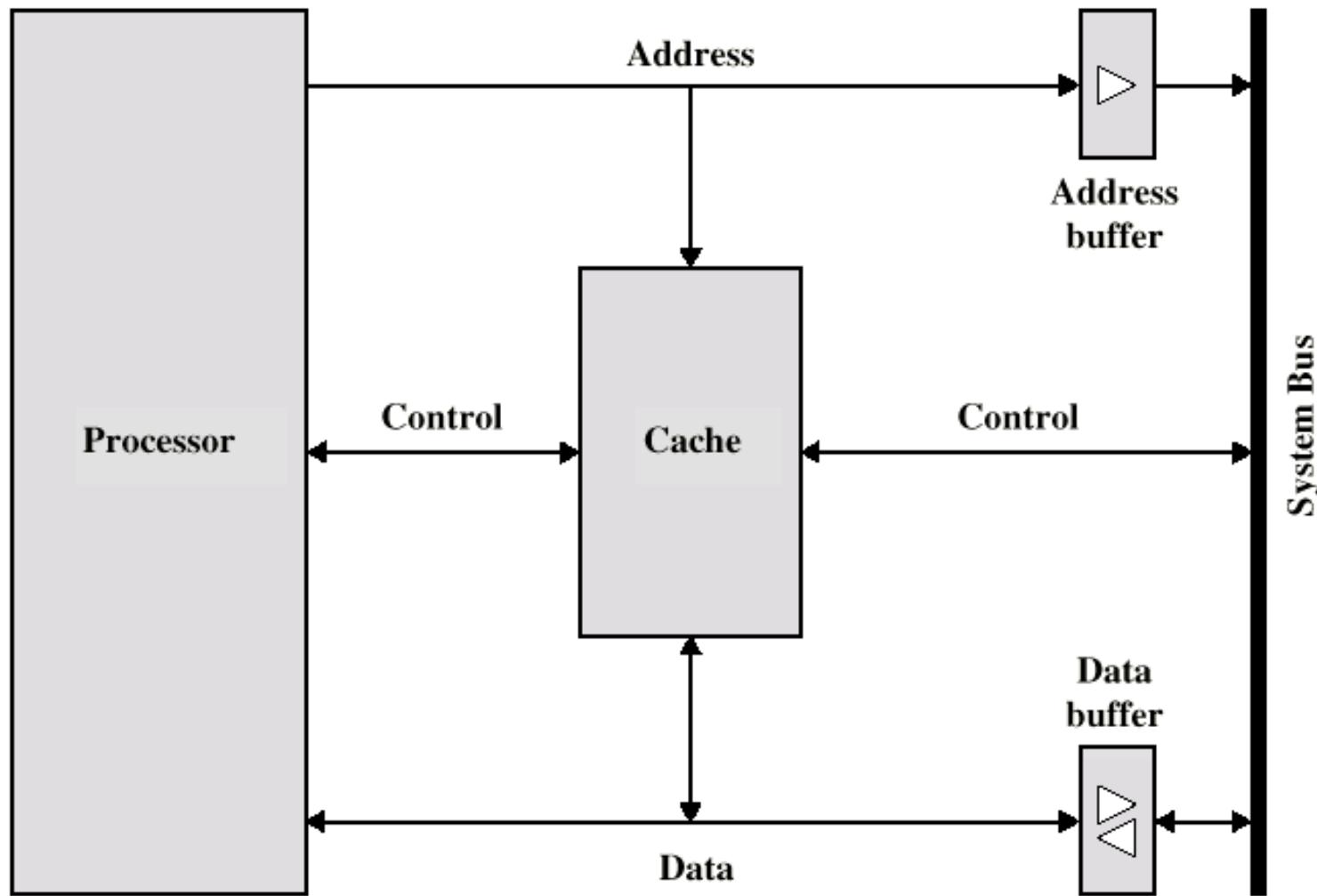
# Cache Design

- Size
- Mapping Function
- Replacement Algorithm
- Write Policy
- Block Size
- Number of Caches

# Cache size

- Cost – More cache is expensive
- Speed
  - More cache is faster (up to a point)
  - Larger decoding circuits slow up a cache
  - Algorithm is needed for mapping main memory addresses to lines in the cache. This takes more time than just a direct RAM

# Typical Cache Organization



# Mapping Functions

- A mapping function is the method used to locate a memory address within a cache
- It is used when copying a block from main memory to the cache and it is used again when trying to retrieve data from the cache
- There are three kinds of mapping functions
  - Direct
  - Associative
  - Set Associative

# Cache Example

These notes use an example of a cache to illustrate each of the mapping functions. The characteristics of the cache used are:

- Size: 64 kByte
- Block size: 4 bytes – i.e. the cache has 16k ( $2^{14}$ ) lines of 4 bytes
- Address bus: 24-bit- i.e., 16M bytes main memory divided into 4M 4 byte blocks

# Direct Mapping Traits

- Each block of main memory maps to only one cache line – i.e. if a block is in cache, it will always be found in the same place
- Line number is calculated using the following function

$$i = j \text{ modulo } m$$

where

i = cache line number

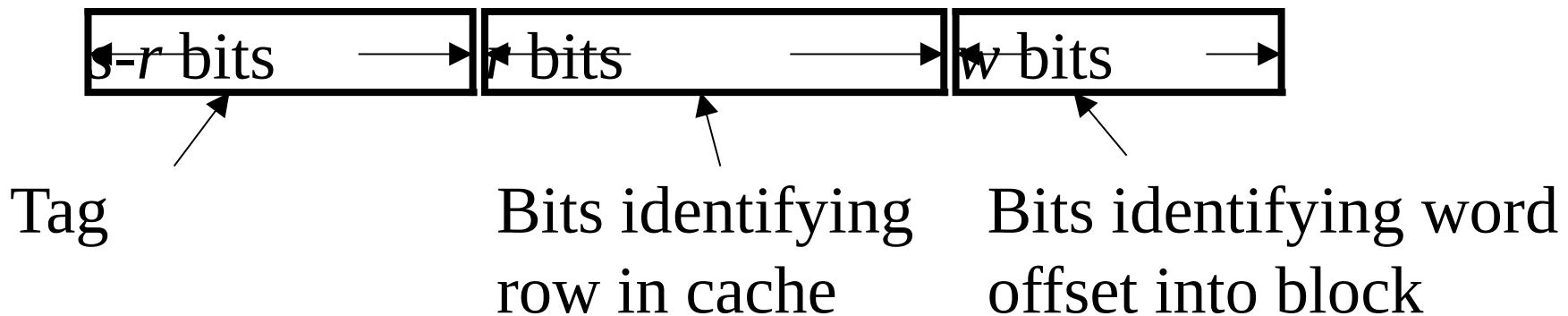
j = main memory block number

m = number of lines in the cache

# Direct Mapping Address Structure

Each main memory address can be divided into three fields

- Least Significant  $w$  bits identify unique word within a block
- Remaining bits ( $s$ ) specify which block in memory. These are divided into two fields
  - Least significant  $r$  bits of these  $s$  bits identifies which line in the cache
  - Most significant  $s-r$  bits uniquely identifies the block within a line of the cache



# Direct Mapping Address Structure (continued)

- Why are the r-bits used to identify which line in cache?
- More likely to have unique r bits than s-r bits based on principle of **locality of reference**

# Direct Mapping Address Structure Example

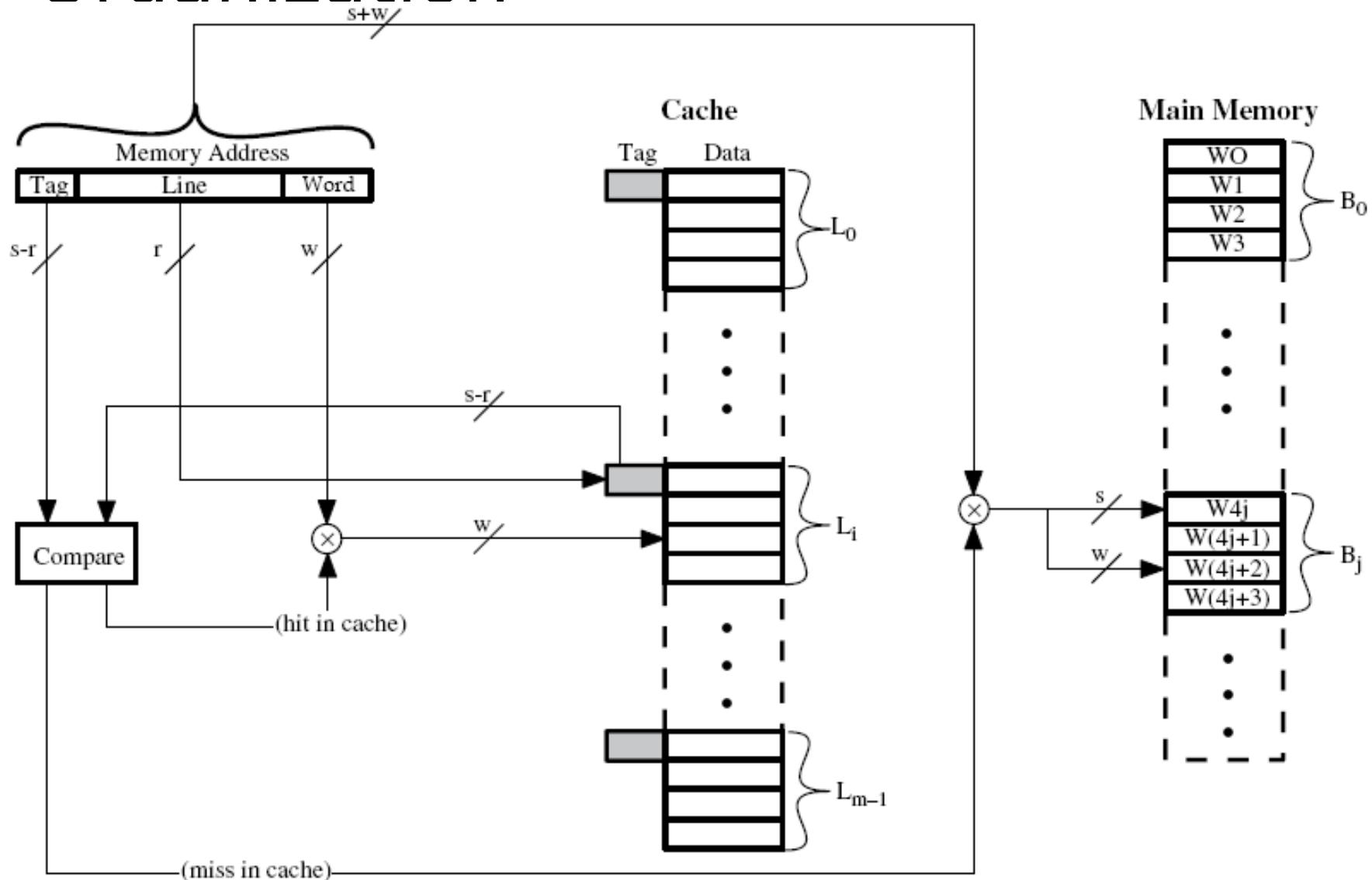
Tag s-r	Line or slot r	Word w
8	14	2

- 24 bit address
- 2 bit word identifier (4 byte block)
- 22 bit block identifier
- 8 bit tag (=22-14)
- 14 bit slot or line
- No two blocks in the same line have the same tag
- Check contents of cache by finding line and comparing tag

# Direct Mapping Cache Line Table

Cache line	Main Memory blocks held
0	0, m, 2m, 3m... $2^s - m$
1	1, m+1, 2m+1... $2^s - m + 1$
$m - 1$	$m - 1, 2m - 1, 3m - 1 \dots 2^s - 1$

# Direct Mapping Cache Organization



# Direct Mapping Examples

What cache line number will the following addresses be stored to, and what will the minimum address and the maximum address of each block they are in be if we have a cache with 4K lines of 16 words to a block in a 256 Meg memory space (28-bit address)?

	Tag s-r	Line or slot r	Word w
a.) 9ABCDEF <sub>16</sub>	12	12	4
b.) 1234567 <sub>16</sub>			

# More Direct Mapping Examples

Assume that a portion of the tags in the cache in our example looks like the table below. Which of the following addresses are contained in the cache?

- a.)  $438EE8_{16}$
- b.)  $F18EFF_{16}$
- c.)  $6B8EF3_{16}$
- d.)  $AD8EF3_{16}$

Tag (binary)	Line number (binary)	Addresses wi/ block			
		00	01	10	11
0101 0011	1000 1110 1110 10				
1110 1101	1000 1110 1110 11				
1010 1101	1000 1110 1111 00				
0110 1011	1000 1110 1111 01				
1011 0101	1000 1110 1111 10				
1111 0001	1000 1110 1111 11				

# Direct Mapping Summary

- Address length =  $(s + w)$  bits
- Number of addressable units =  $2^{s+w}$  words or bytes
- Block size = line width =  $2^w$  words or bytes
- Number of blocks in main memory =  $2^{s+w}/2^w = 2^s$
- Number of lines in cache =  $m = 2^r$
- Size of tag =  $(s - r)$  bits

# Direct Mapping pros & cons

- Simple
- Inexpensive
- Fixed location for given block –  
If a program accesses 2 blocks that map to the same line repeatedly, cache misses are very high  
(thrashing)

# Associative Mapping Traits

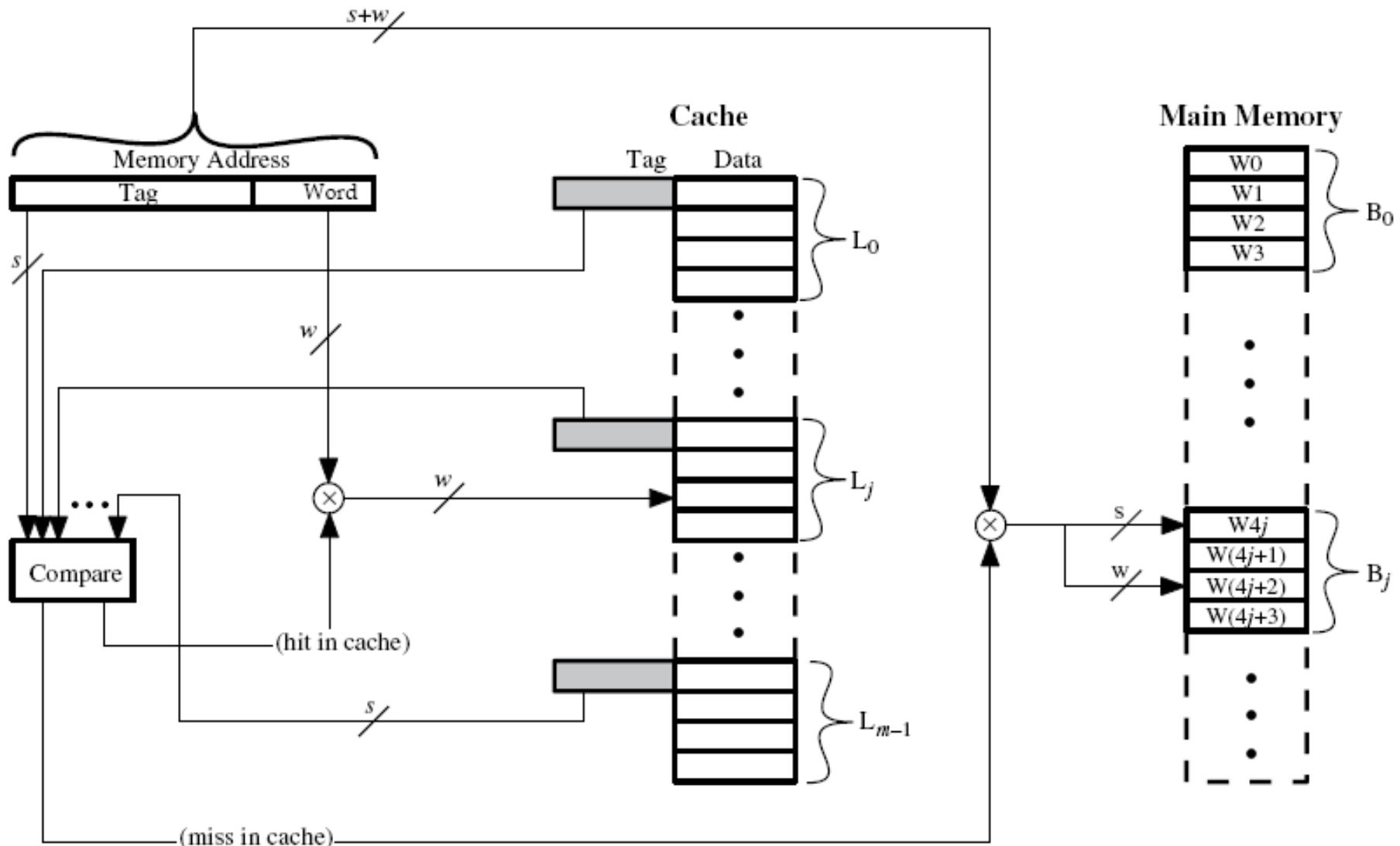
- A main memory block can load into any line of cache
- Memory address is interpreted as:
  - Least significant  $w$  bits = word position within block
  - Most significant  $s$  bits = tag used to identify which block is stored in a particular line of cache
- Every line's tag must be examined for a match
- Cache searching gets expensive and slower

# Associative Mapping Address Structure Example

Tag – s bits (22 in example)	Word – w bits (2 in ex.)
---------------------------------	-----------------------------

- 22 bit tag stored with each 32 bit block of data
- Compare tag field with tag entry in cache to check for hit
- Least significant 2 bits of address identify which of the four 8 bit words is required from 32 bit data block

# Fully Associative Cache Organization



# Fully Associative Mapping Example

Assume that a portion of the tags in the cache in our example looks like the table below. Which of the following addresses are contained in the cache?

- a.)  $438EE8_{16}$
- b.)  $F18EFF_{16}$
- c.)  $6B8EF3_{16}$
- d.)  $AD8EF3_{16}$

Tag (binary)	Addresses wi/ block			
	00	01	10	11
0101 0011 1000 1110 1110 10				
1110 1101 1100 1001 1011 01				
1010 1101 1000 1110 1111 00				
0110 1011 1000 1110 1111 11				
1011 0101 0101 1001 0010 00				
1111 0001 1000 1110 1111 11				

# Associative Mapping Summary

- Address length =  $(s + w)$  bits
- Number of addressable units =  $2^{s+w}$  words or bytes
- Block size = line size =  $2^w$  words or bytes
- Number of blocks in main memory =  $2^{s+w}/2^w = 2^s$
- Number of lines in cache = undetermined
- Size of tag =  $s$  bits

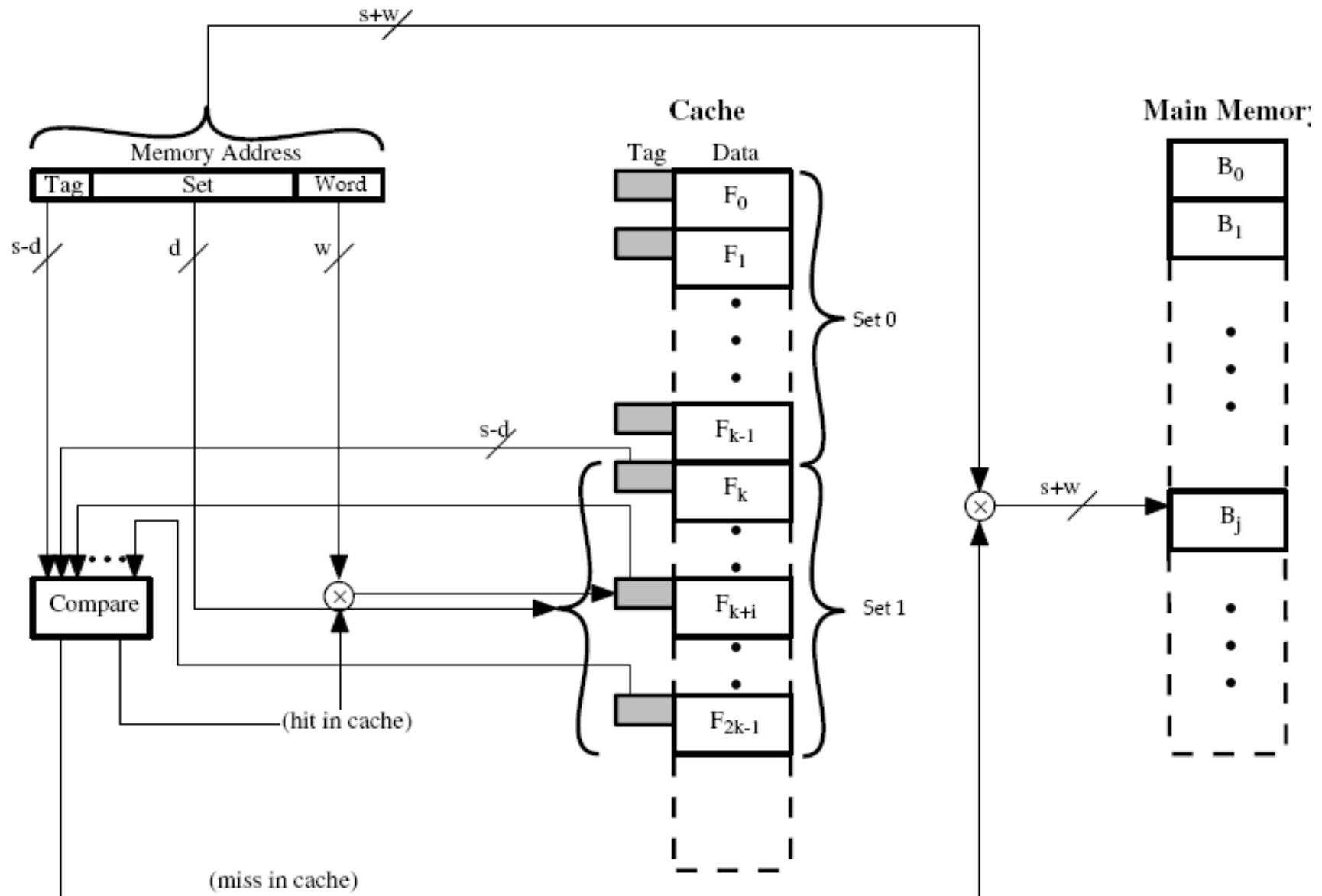
# Set Associative Mapping Traits

- Address length is  $s + w$  bits
- Cache is divided into a number of sets,  $v = 2^d$
- $k$  blocks/lines can be contained within each set
- $k$  lines in a cache is called a  $k$ -way set associative mapping
- Number of lines in a cache =  $v \bullet k = k \bullet 2^d$
- Size of tag =  $(s-d)$  bits

# Set Associative Mapping Traits (continued)

- Hybrid of Direct and Associative
  - $k = 1$ , this is basically direct mapping
  - $v = 1$ , this is associative mapping
- Each set contains a number of lines, basically the number of lines divided by the number of sets
- A given block maps to any line within its specified set - e.g. Block B can be in any line of set i.
- 2 lines per set is the most common organization.
  - Called 2 way associative mapping
  - A given block can be in one of 2 lines in only one specific set
  - Significant improvement over direct mapping

# K-Way Set Associative Cache Organization



# How does this affect our example?

- Let's go to two-way set associative mapping
- Divides the 16K lines into 8K sets
- This requires a 13 bit set number
- With 2 word bits, this leaves 9 bits for the tag
- Blocks beginning with the addresses  $000000_{16}$ ,  $008000_{16}$ ,  $010000_{16}$ ,  $018000_{16}$ ,  $020000_{16}$ ,  $028000_{16}$ , etc. map to the same set, Set 0.
- Blocks beginning with the addresses  $00000416$ ,  $008004_{16}$ ,  $010004_{16}$ ,  $018004_{16}$ ,  $020004_{16}$ ,  $028004_{16}$ , etc. map to the same set, Set 1.

# Set Associative Mapping Address Structure

Tag 9 bits	Set 13 bits	Word 2 bits
---------------	----------------	----------------

- Note that there is one more bit in the tag than for this same example using direct mapping.
- Therefore, it is 2-way set associative
- Use set field to determine cache set to look in
- Compare tag field to see if we have a hit

# Set Associative Mapping Example

For each of the following addresses, answer the following questions based on a 2-way set associative cache with 4K lines, each line containing 16 words, with the main memory of size 256 Meg memory space (28-bit address):

- What cache set number will the block be stored to?
- What will their tag be?
- What will the minimum address and the maximum address of each block they are in be?

1.  $9ABCDEF_{16}$

Tag s-r	Set s	Word w
13	11	4

2.  $1234567_{16}$

# Set Associative Mapping Summary

- Address length =  $(s + w)$  bits
- Number of addressable units =  $2^{s+w}$  words or bytes
- Block size = line size =  $2^w$  words or bytes
- Number of blocks in main memory =  $2^{s+w}/2^w = 2^s$
- Number of lines in set =  $k$
- Number of sets =  $v = 2^d$
- Number of lines in cache =  $kv = k * 2^d$
- Size of tag =  $(s - d)$  bits

# Replacement Algorithms

- There must be a method for selecting which line in the cache is going to be replaced when there's no room for a new line
- Hardware implemented algorithm (speed)
- Direct mapping
  - There is no need for a replacement algorithm with direct mapping
  - Each block only maps to one line
  - Replace that line

# Associative & Set Associative Replacement Algorithms

- Least Recently used (LRU)
  - Replace the block that hasn't been touched in the longest period of time
  - Two way set associative simply uses a USE bit. When one block is referenced, its USE bit is set while its partner in the set is cleared
- First in first out (FIFO) – replace block that has been in cache longest

# Associative & Set Associative Replacement Algorithms (continued)

- Least frequently used (LFU) – replace block which has had fewest hits
- Random – only slightly lower performance than use-based algorithms LRU, FIFO, and LFU

# Writing to Cache

- Must not overwrite a cache block unless main memory is up to date
- Two main problems:
  - If cache is written to, main memory is invalid or if main memory is written to, cache is invalid – Can occur if I/O can address main memory directly
  - Multiple CPUs may have individual caches; once one cache is written to, all caches are invalid

# Write through

- All writes go to main memory as well as cache
- Multiple CPUs can monitor main memory traffic to keep local (to CPU) cache up to date
- Lots of traffic
- Slows down writes

# Write back

- Updates initially made in cache only
- Update bit for cache slot is set when update occurs
- If block is to be replaced, write to main memory only if update bit is set
- Other caches get out of sync
- I/O must access main memory through cache
- Research shows that 15% of memory references are writes

# Multiple Processors/Multiple Caches

- Even if a write through policy is used, other processors may have invalid data in their caches
- In other words, if a processor updates its cache and updates main memory, a second processor may have been using the same data in its own cache which is now invalid.

# Solutions to Prevent Problems with Multiprocessor/cache systems

- **Bus watching with write through** – each cache watches the bus to see if data they contain is being written to the main memory by another processor. All processors must be using the write through policy
- **Hardware transparency** – a "big brother" watches all caches, and upon seeing an update to any processor's cache, it updates main memory AND all of the caches
- **Noncacheable memory** – Any shared memory (identified with a chip select) may not be cached.

# Line Size

- There is a relationship between line size (i.e., the number of words in a line in the cache) and hit ratios
- As the line size (block size) goes up, the hit ratio could go up due to more words available to the principle of **locality of reference**
- As block size increases, however, the number of blocks goes down, and the hit ratio will begin to go back down after a while
- Lastly, as the block size increases, the chances of a hit to a word farther from the initially referenced word goes down

# Multi-Level Caches

- Increases in transistor densities have allowed for caches to be placed inside processor chip
- Internal caches have very short wires (within the chip itself) and are therefore quite fast, even faster than any zero wait-state memory accesses outside of the chip
- This means that a super fast internal cache (level 1) can be inside of the chip while an external cache (level 2) can provide access faster than to main memory

# Unified versus Split Caches

- Split into two caches – one for instructions, one for data
- Disadvantages
  - Questionable as unified cache balances data and instructions merely with hit rate.
  - Hardware is simpler with unified cache
- Advantage
  - What a split cache is really doing is providing one cache for the instruction decoder and one for the execution unit.
  - This supports pipelined architectures.

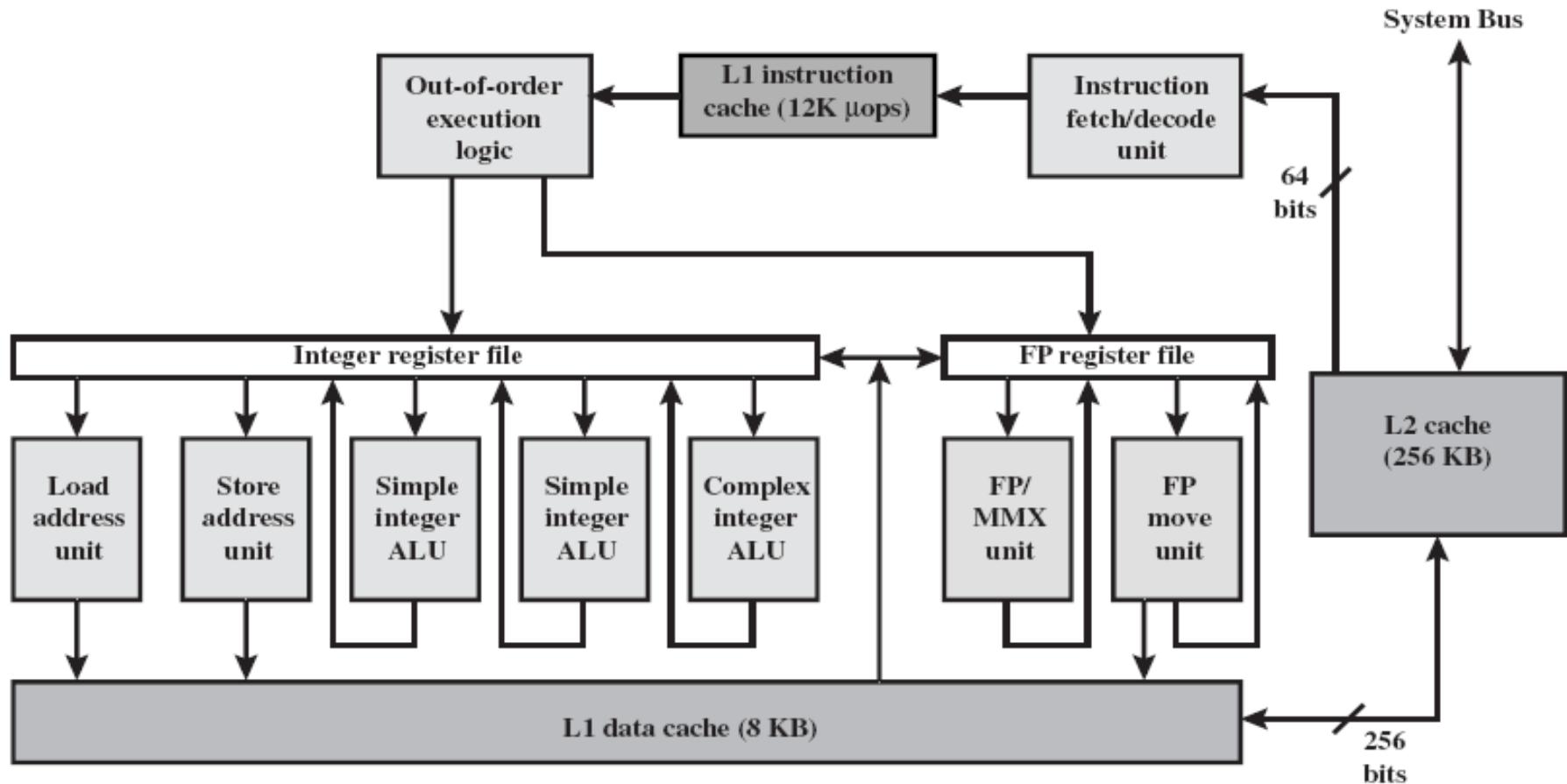
# Intel x86 caches

- 80386 – no on chip cache
- 80486 – 8k using 16 byte lines and four-way set associative organization (main memory had 32 address lines – 4 Gig)
- Pentium (all versions)
  - Two on chip L1 caches
  - Data & instructions

# Pentium 4 L1 and L2 Caches

- L1 cache
  - 8k bytes
  - 64 byte lines
  - Four way set associative
- L2 cache
  - Feeding both L1 caches
  - 256k
  - 128 byte lines
  - 8 way set associative

# Pentium 4 (Figure 4.13)



# Pentium 4 Operation – Core Processor

- Fetch/Decode Unit
  - Fetches instructions from L2 cache
  - Decode into micro-ops
  - Store micro-ops in L1 cache
- Out of order execution logic
  - Schedules micro-ops
  - Based on data dependence and resources
  - May speculatively execute

# Pentium 4 Operation – Core Processor (continued)

- Execution units
  - Execute micro-ops
  - Data from L1 cache
  - Results in registers
- Memory subsystem – L2 cache and systems bus

# Pentium 4 Design Reasoning

- Decodes instructions into RISC like micro-ops before L1 cache
- Micro-ops fixed length – Superscalar pipelining and scheduling
- Pentium instructions long & complex
- Performance improved by separating decoding from scheduling & pipelining – (More later - ch14)

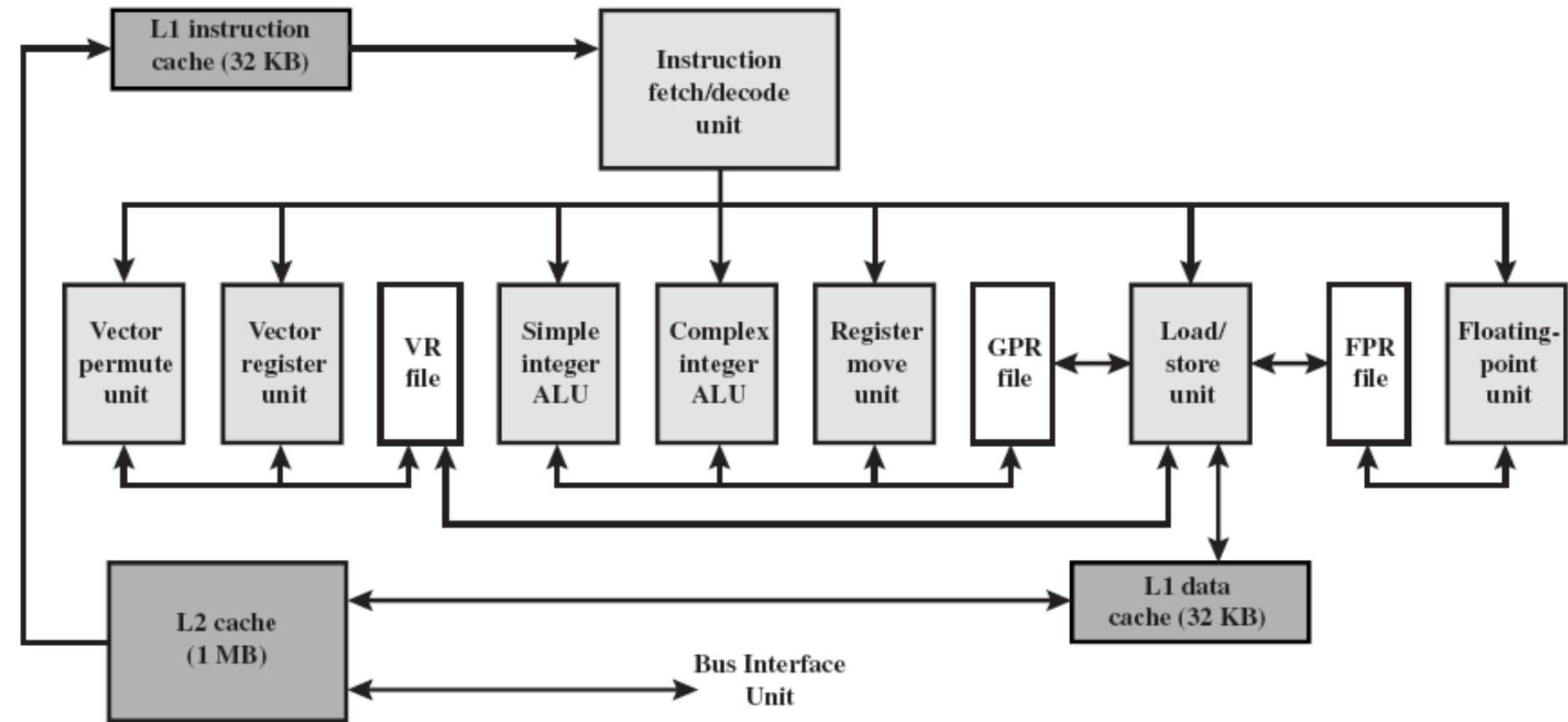
# Pentium 4 Design Reasoning (continued)

- Data cache is write back – Can be configured to write through
- L1 cache controlled by 2 bits in register
  - CD = cache disable
  - NW = not write through
  - 2 instructions to invalidate (flush) cache and write back then invalidate

# Power PC Cache Organization

- 601 – single 32kb 8 way set associative
- 603 – 16kb (2 x 8kb) two way set associative
- 604 – 32kb
- 610 – 64kb
- G3 & G4
  - 64kb L1 cache – 8 way set associative
  - 256k, 512k or 1M L2 cache – two way set associative

# PowerPC G4 (Figure 4.14)



# Comparison of Cache Sizes (Table 4.3)

Table 4.3 Cache Sizes of Some Processors

Processor	Type	Year of Introduction	L1 cache <sup>a</sup>	L2 cache	L3 cache
IBM 360/85	Mainframe	1968	16 to 32 KB	—	—
PDP-11/70	Minicomputer	1975	1 KB	—	—
VAX 11/780	Minicomputer	1978	16 KB	—	—
IBM 3033	Mainframe	1978	64 KB	—	—
IBM 3090	Mainframe	1985	128 to 256 KB	—	—
Intel 80486	PC	1989	8 KB	—	—
Pentium	PC	1993	8 KB/8 KB	256 to 512 KB	—
PowerPC 601	PC	1993	32 KB	—	—
PowerPC 620	PC	1996	32 KB/32 KB	—	—
PowerPC G4	PC/server	1999	32 KB/32 KB	256 KB to 1 MB	2 MB
IBM S/390 G4	Mainframe	1997	32 KB	256 KB	2 MB
IBM S/390 G6	Mainframe	1999	256 KB	8 MB	—
Pentium 4	PC/server	2000	8 KB/8 KB	256 KB	—
IBM SP	High-end server/ supercomputer	2000	64 KB/32 KB	8 MB	—
CRAY MTA <sup>b</sup>	Supercomputer	2000	8 KB	2 MB	—
Itanium	PC/server	2001	16 KB/16 KB	96 KB	4 MB
SGI Origin 2001	High-end server	2001	32 KB/32 KB	4 MB	—

<sup>a</sup> Two values separated by a slash refer to instruction and data caches

<sup>b</sup> Both caches are instruction only; no data caches

# **Unit 2**

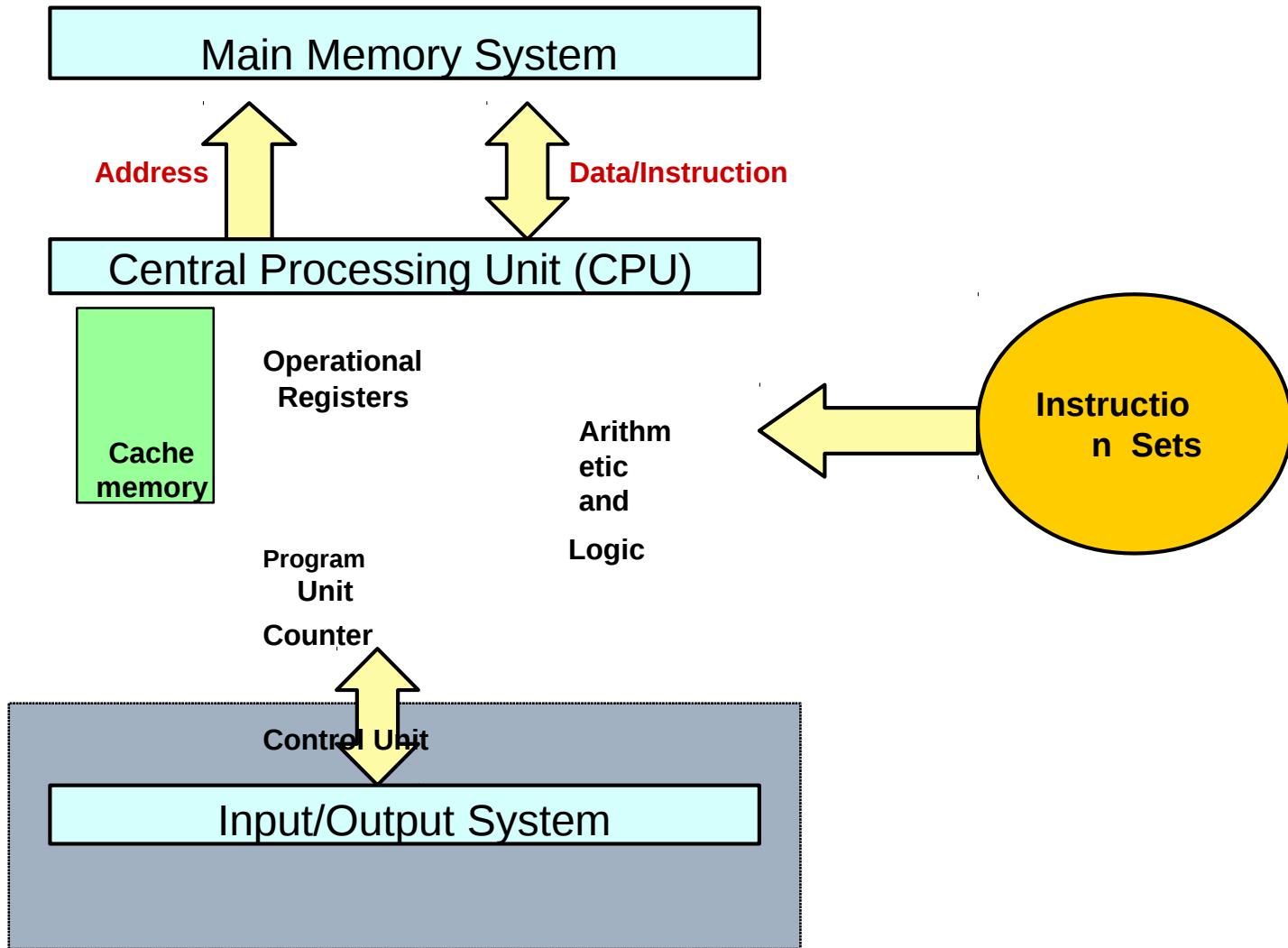
# **Input/Output Ports &**

# **Memory Systems**

# Outline

- Accessing I / O Devices
- Interrupts
- Direct Memory Access
- Buses
- Interface Circuits
- Standard I / O  
Interfaces

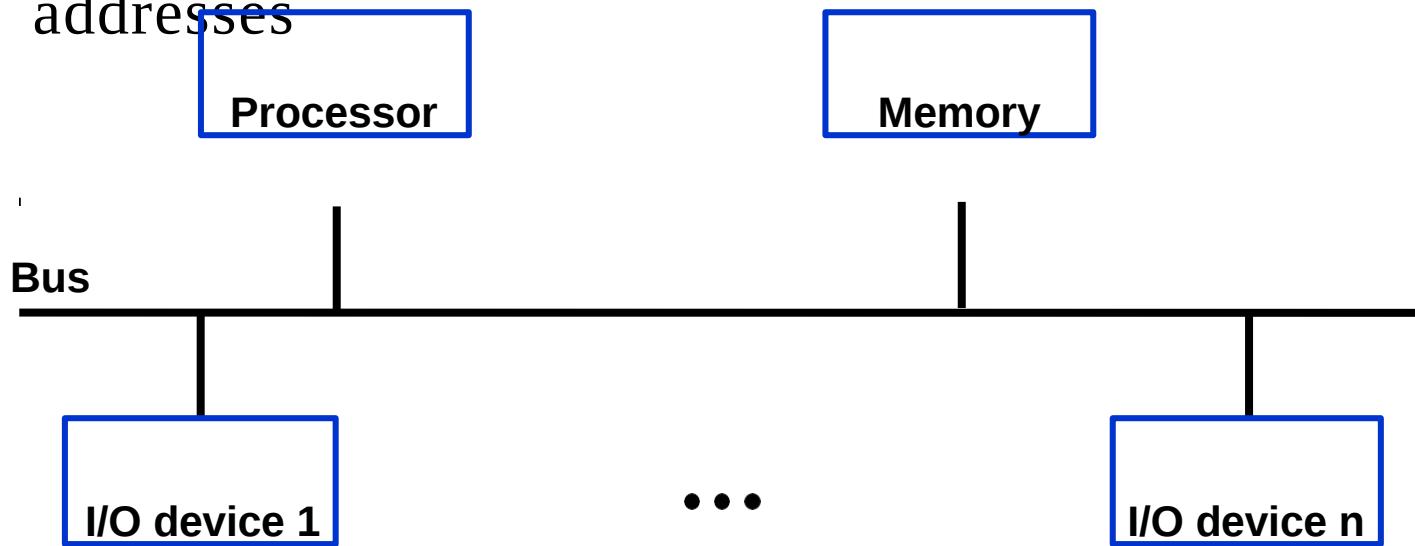
# Content Coverage



# Accessing I/O Devices

## ➤ Single-bus structure

- ◆ The bus enables all the devices connected to it to exchange information
- ◆ Typically, the bus consists of three sets of lines used to carry address, data, and control signals
- ◆ Each I / O device is assigned a unique set of addresses



# I/O Mapping

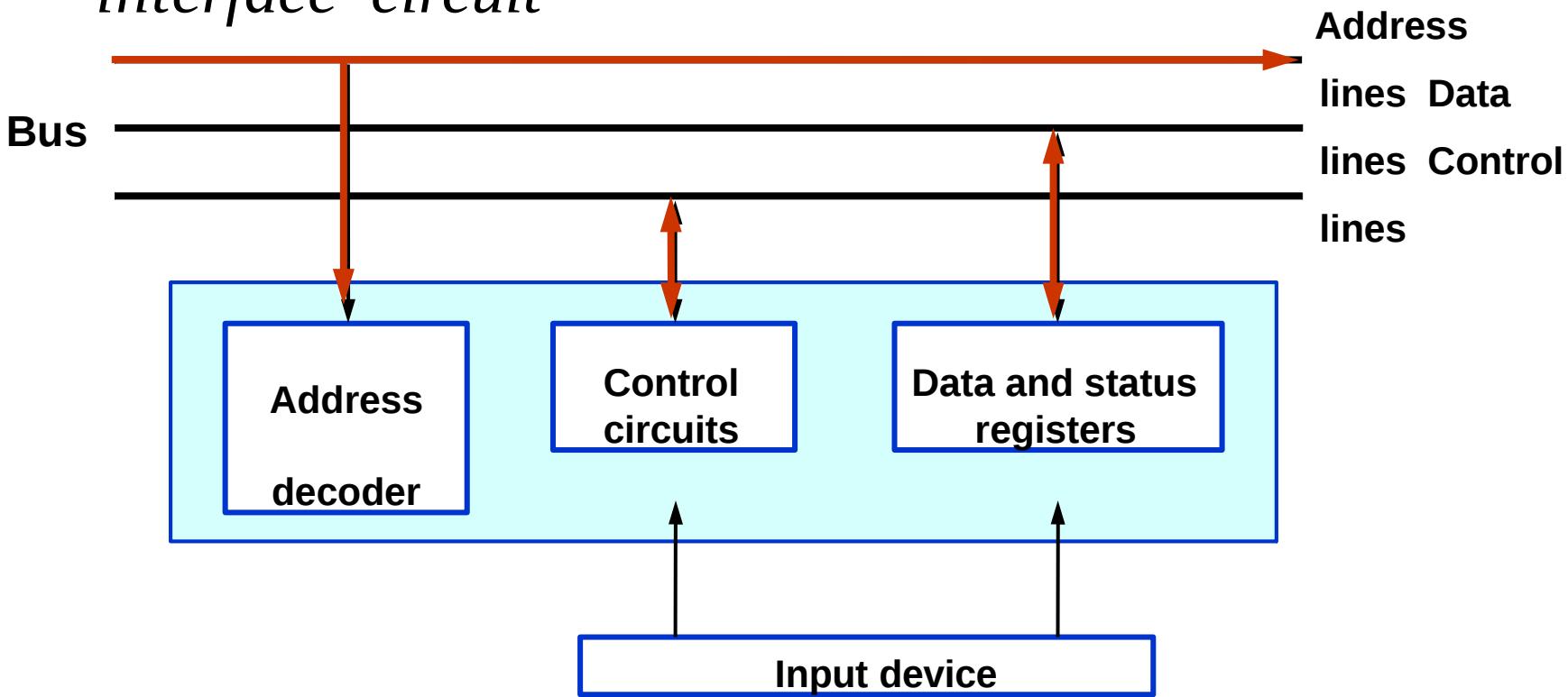
- Memory mapped I / O
  - ◆ Devices and memory share an address space
  - ◆ I / O looks just like memory read / write
  - ◆ No special commands for I / O
    - Large selection of memory access commands available
- Isolated I / O
  - ◆ Separate address spaces
  - ◆ Need I / O or memory select lines
  - ◆ Special commands for I / O
    - Limited set

# Memory-Mapped I/O

- When I / O devices and the memory share the same address space, the arrangement is called memory-mapped I / O
- With memory-mapped I / O, any machine instruction that can access memory can be used to transfer data to or from an I / O device
- Most computer systems use memory-mapped I / O.
- Some processors have special IN and OUT instructions to perform I / O transfers
  - ◆ When building a computer system based on these processors, the designer has the option of connecting I / O devices to use the special I / O address space or simply incorporating them as part of the memory address space

# I/O Interface for an Input Device

- The address decoder, the data and status registers, and the control circuitry required to coordinate I / O transfers constitute the device's *interface circuit*



# I/O Techniques

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

# Program-Controlled I/O

➤ Consider a simple example of I / O operations involving a keyboard and a display device in a computer system. The four registers shown below are used in the data transfer operations

- ◆ The two flags KIRQ and DIRQ in STATUS register

are  
~~DATAIN~~

used in conjunction with interrupts

~~DATAOUT~~

~~STATUS~~

DIRQ    KIRQ    SOUT    SIN

~~S~~

DEN    KEN

~~CONTROL~~

7        6        5        4        3        2        1        0

# An Example

- A program that reads one line from the keyboard, stores it in memory buffer, and echoes it back to the display
- 

	Move	#LINE, R0	Initialize memory pointer
WAITK	TestBit	#0,STATUS	Test SIN
	Branch=0	WAITK	Wait for character to be entered
	Move	DATAIN,R1	Read character
WAITD	TestBit	#1,STATUS	Test SOUT
	Branch=0	WAITD	Wait for display to become ready
	Move	R1,DATAOUT	Send character to display
	Move	R1,(R0)+	Store character and advance pointer
	Compare	#\$0D,R1	Check if Carriage Return
	Branch $\neq$ 0	WAITK	If not, get another character
	Move	#\$0A,DATAOUT	Otherwise, send Line Feed
	Call	PROCESS	Call a subroutine to process the input line

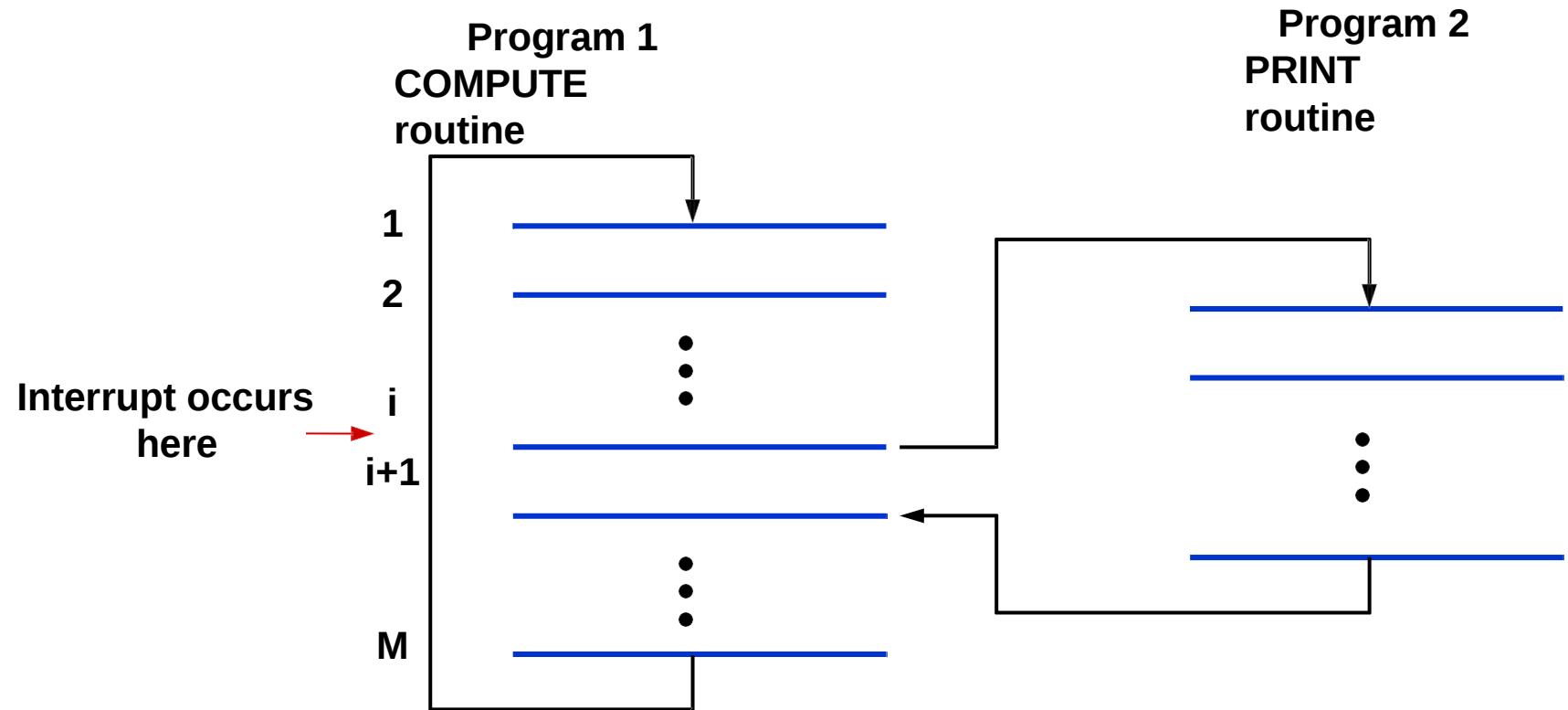
# Program-Controlled I/O

- The example described above illustrates program-controlled I / O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor *polls* the devices
- There are two other commonly used mechanisms for implementing I / O operations: *interrupts* and *direct memory access*
  - ◆ Interrupts: synchronization is achieved by having the I / O device send a special signal over the bus whenever it is ready for a data transfer operation
  - ◆ Direct memory access: it involves having the device interface transfer data directly to or from the memory

# Interrupts

- To avoid the processor being not performing any useful computation, a hardware signal called an *interrupt* to the processor can do it. At least one of the bus control lines, called an *interrupt-request* line, is usually dedicated for this purpose
- An *interrupt-service routine* usually is needed and is executed when an interrupt request is issued
- On the other hand, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal. An *interrupt-acknowledge* signal serves

# Example



# Interrupt-Service Routine & Subroutine

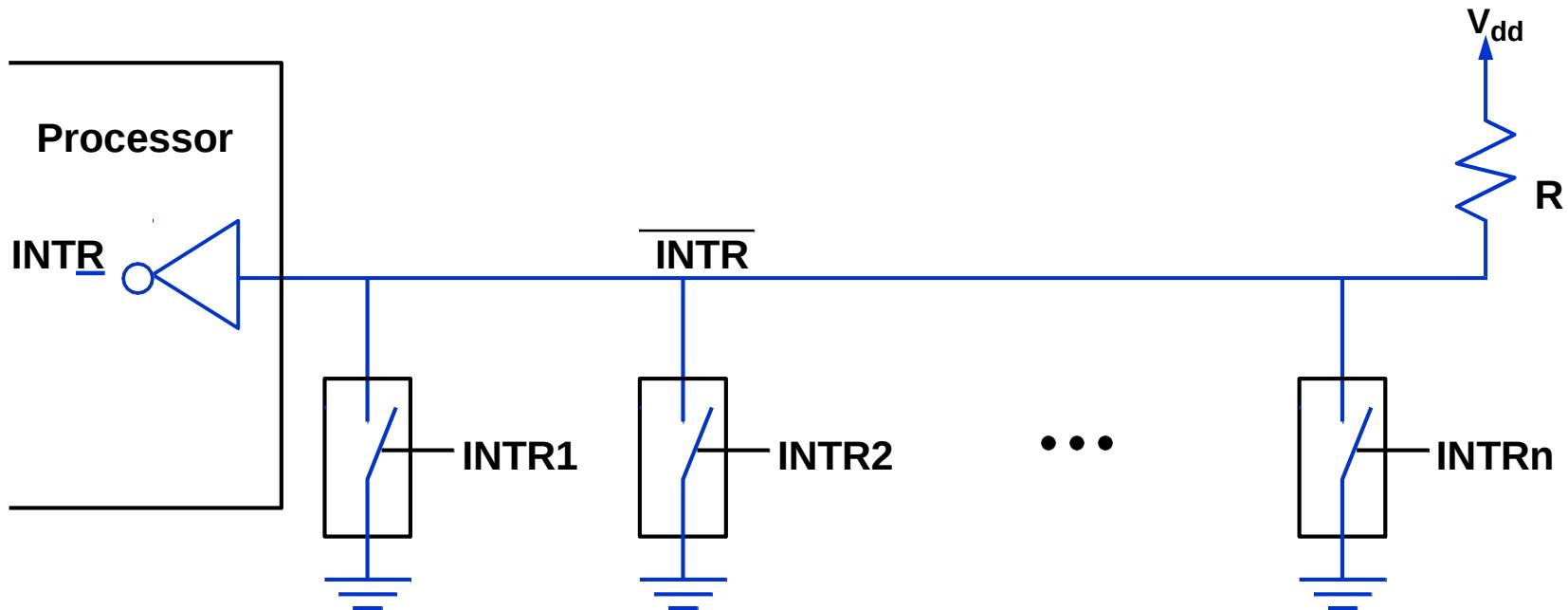
- Treatment of an interrupt-service routine is very similar to that of a subroutine
- An important departure from the similarity should be noted
  - ◆ A subroutine performs a function required by the program from which it is called.
  - ◆ The interrupt-service routine may not have anything in common with the program being executed at the time the interrupt request is received. In fact, the two programs often belong to different users
- Before executing the interrupt-service routine, any information that may be altered during the execution of that routine must be saved. This information must be restored before the interrupted program is resumed

# Interrupt Latency

- The information that needs to be saved and restored typically includes the condition code flags and the contents of any registers used by both the interrupted program and the interrupt-service routine
- Saving registers also increases the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine. The delay is called *interrupt latency*
- Typically, the processor saves only the contents of the program counter and the processor status register. Any additional information that needs to be saved must be saved by program instruction at the beginning of the interrupt-service routine and restored at the end of the routine

# Interrupt Hardware

- An equivalent circuit for an open-drain bus used to implement a common interrupt-request line



$$\text{INTR} = \text{INTR1} + \text{INTR2} + \dots + \text{INTRn}$$

# Handling Multiple Devices

- Handling multiple devices gives rise to a number of questions:
  - ◆ How can the processor recognize the device requesting an interrupt?
  - ◆ Given that different devices are likely to require different interrupt-service routines, how can the processor obtain the starting address of the appropriate routine in each case?
  - ◆ Should a device be allowed to interrupt the processor while another interrupt is being serviced?
  - ◆ How should two or more simultaneous interrupt requests be handled?
- The information needed to determine whether a device is requesting an interrupt is available in its status register

# Identify the Interrupting Device

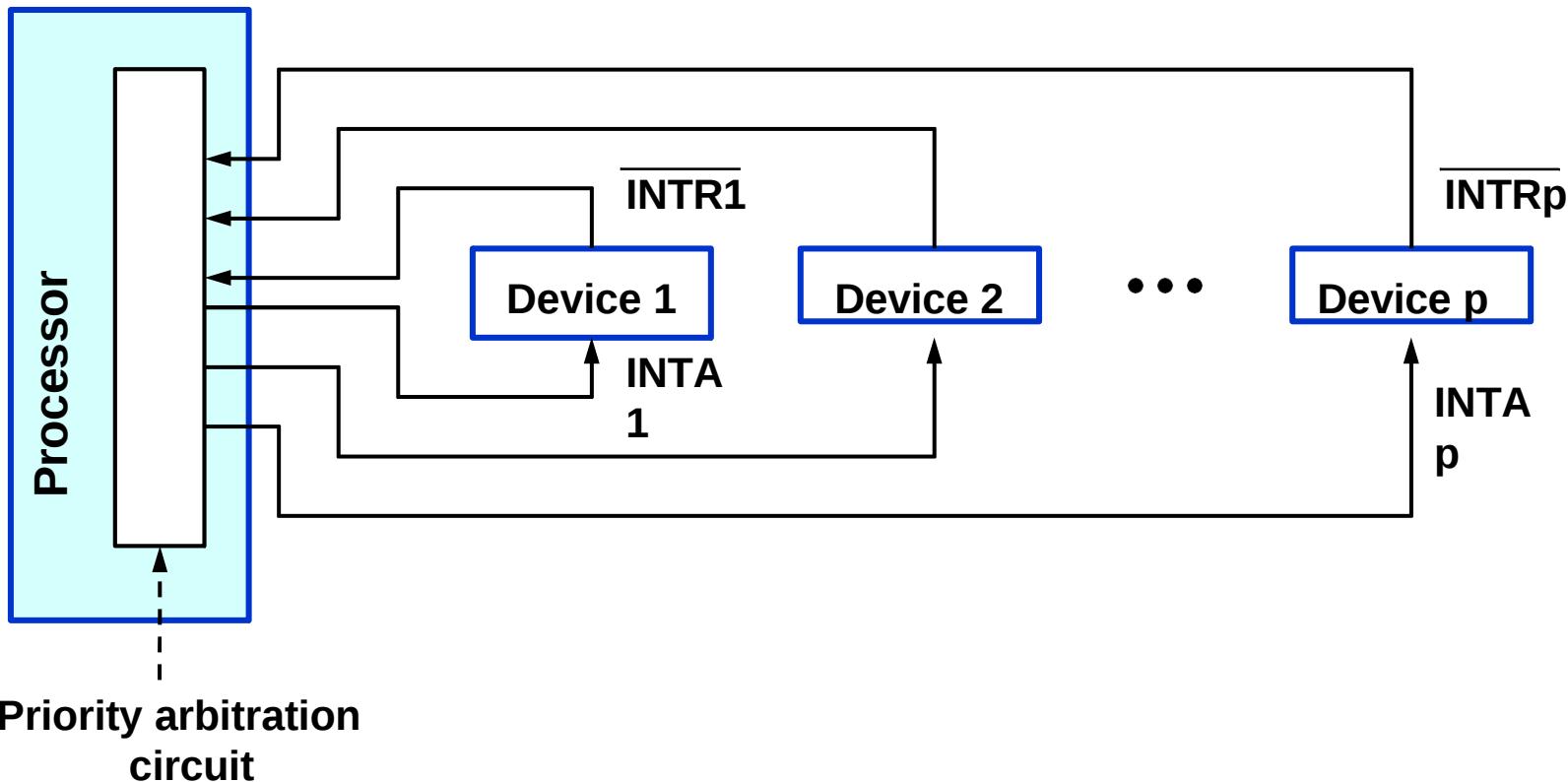
- The simplest way to identify the interrupting device is to have the interrupt-service routine poll all the I / O devices connected to the bus
  - ◆ The polling scheme is easy to implement. Its main disadvantage is the time spent interrogating all the devices
- A device requesting an interrupt may identify itself directly to the processor. Then, the processor can immediately start executing the corresponding interrupt-service routine. This is called vectored interrupts
- An interrupt request from a high-priority device should be accepted while the processor is servicing another request from a lower-priority device

# Interrupt Priority

- The processor's priority is usually encoded in a few bits of the processor status word. It can be changed by program instructions that write into the program status register (PS). These are privileged instructions, which can be executed only while the processor is running in the supervisor mode
- The processor is in the supervisor mode only when executing operating system routines. It switches to the user mode before beginning to execute application program
- An attempt to execute a privileged instruction while in the user mode leads to a special type of interrupt called a privilege exception

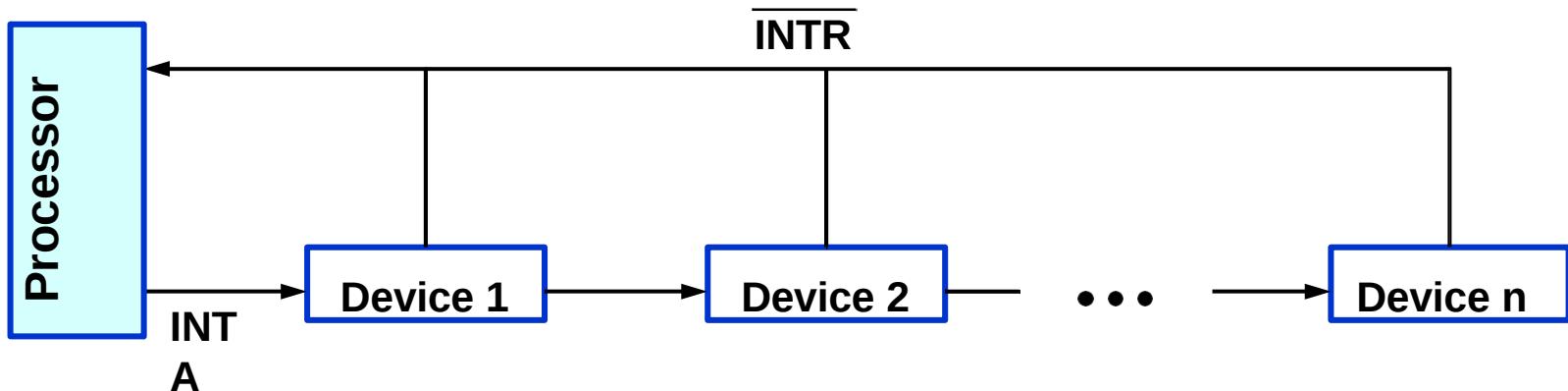
# Implementation of Interrupt Priority

- An example of the implementation of a multiple-priority scheme



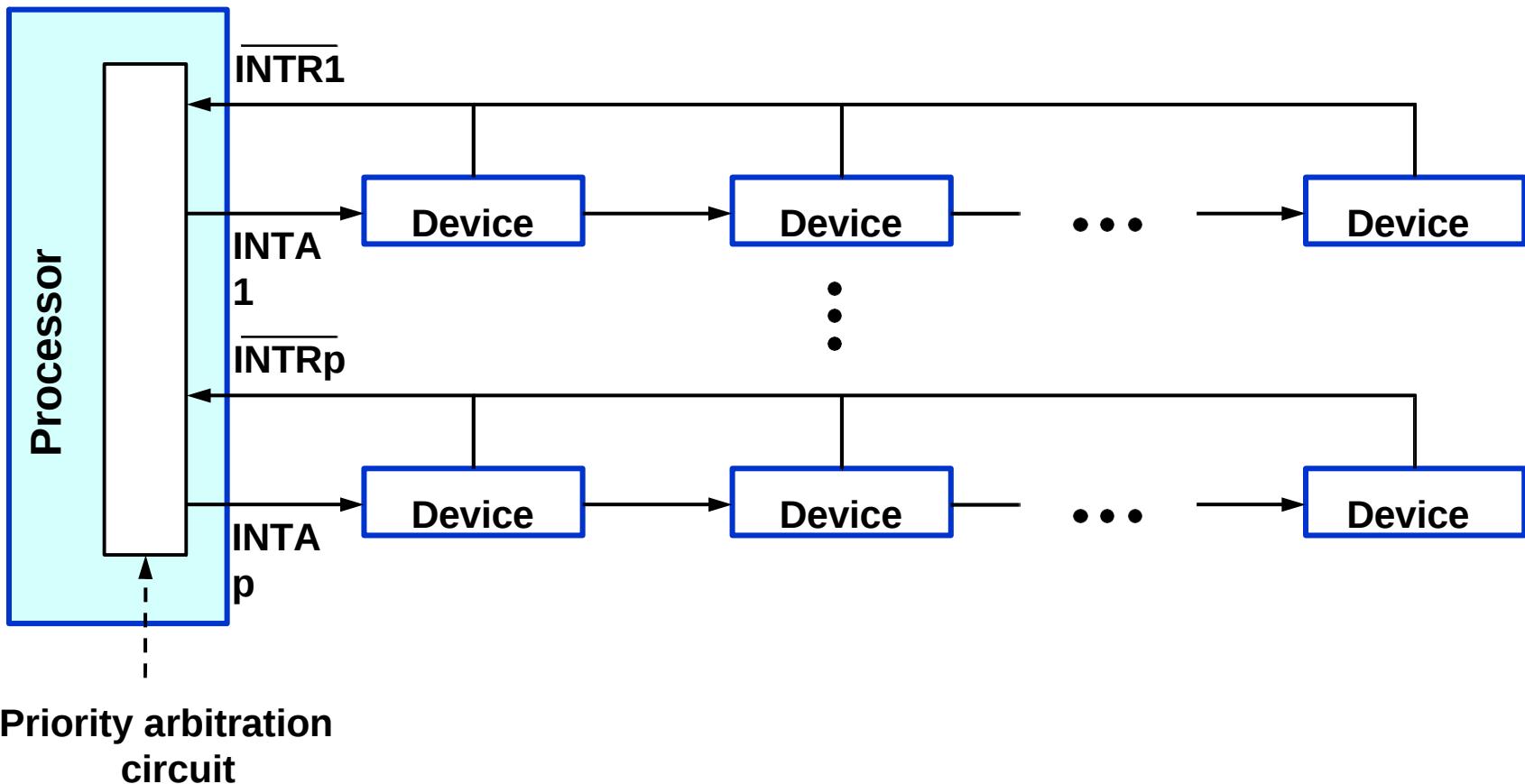
# Simultaneous Requests

- Consider the problem of simultaneous arrivals of interrupt requests from two or more devices. The processor must have some means of deciding which request to service first
- Interrupt priority scheme with daisy chain



# Priority Group

- Combination of the interrupt priority scheme with daisy chain and with individual interrupt-request and interrupt-acknowledge lines

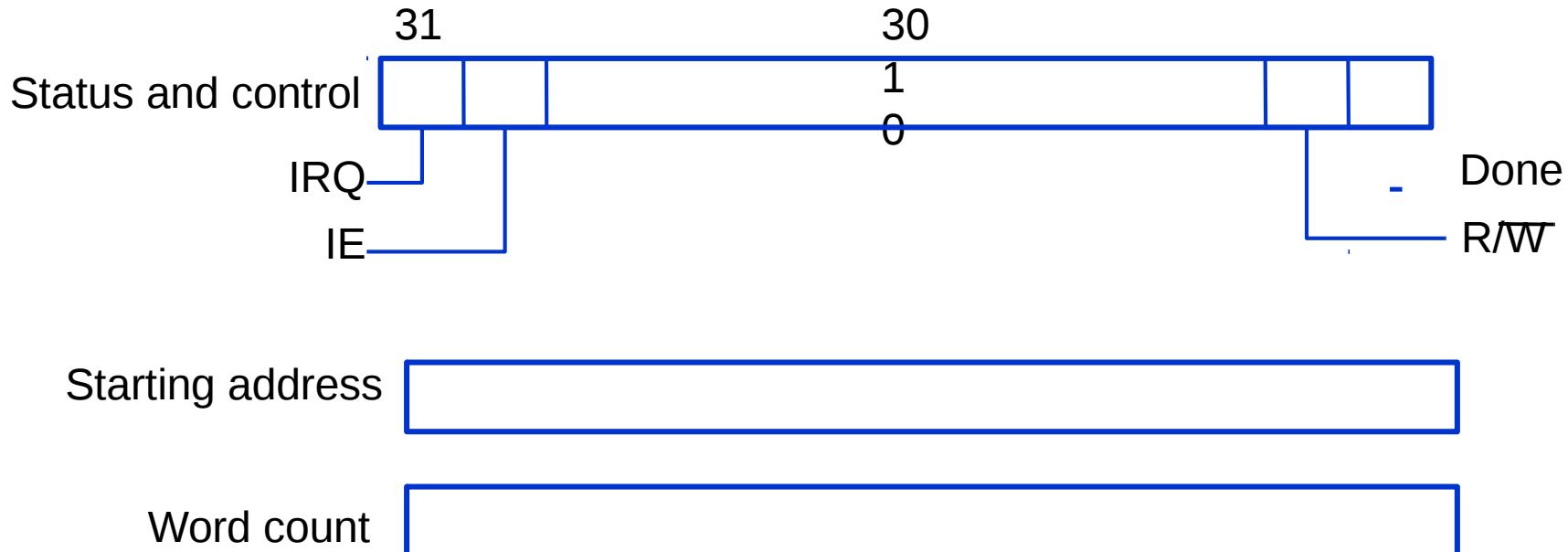


# Direct Memory Access

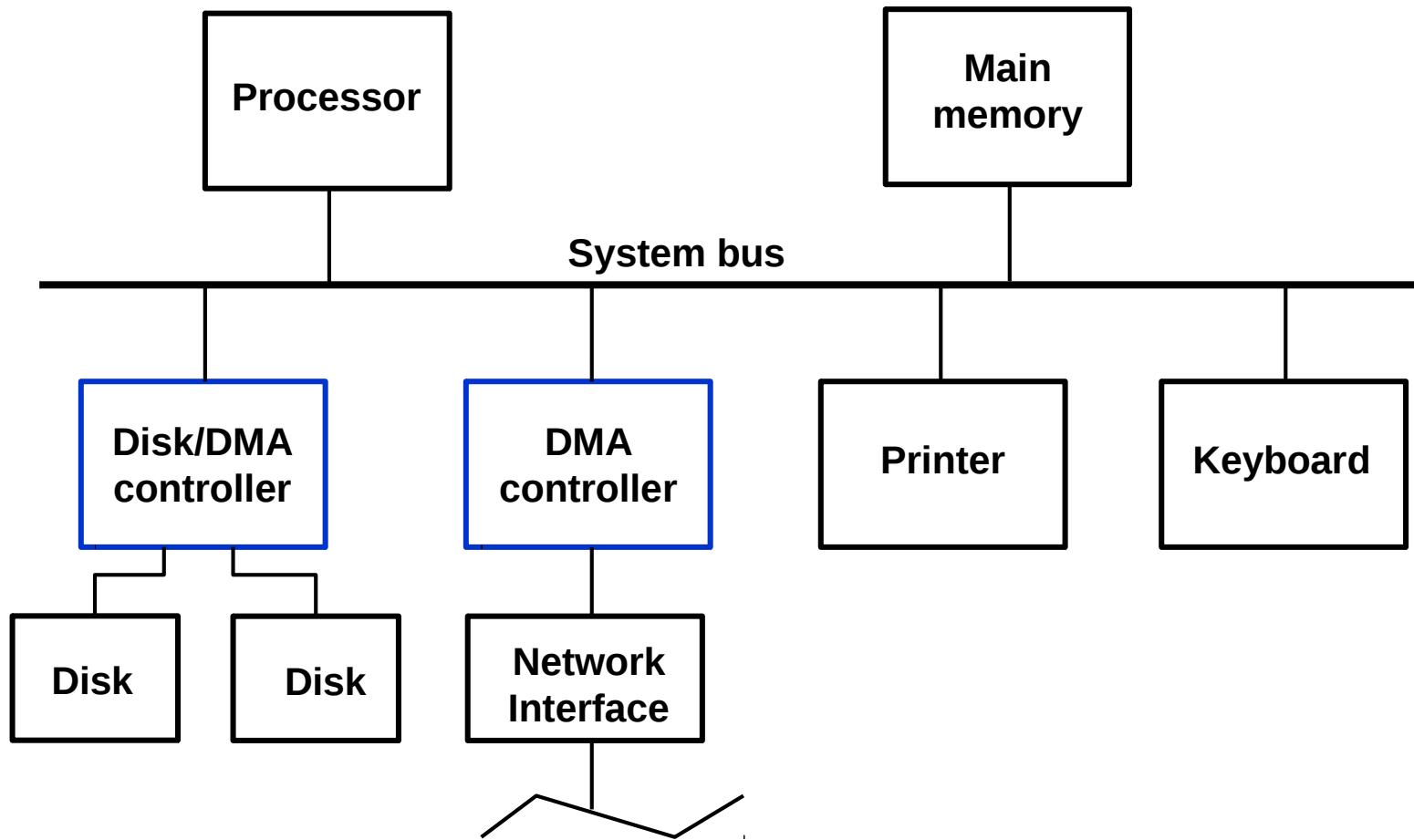
- To transfer large blocks of data at high speed, a special control unit may be provided between an external device and the main memory, without continuous intervention by the processor. This approach is called *direct memory access* (DMA)
- DMA transfers are performed by a control circuit that is part of the I / O device interface. We refer to this circuit as a DMA controller.
- Since it has to transfer blocks of data, the DMA controller must increment the memory address for successive words and keep track of the number of transfers

# DMA Controller

- Although a DMA controller can transfer data without intervention by the processor, its operation must be under the control of a program executed by the processor
- An example



# DMA Controller in a Computer System



# Memory Access Priority

- Memory accesses by the processor and the DMA controllers are interwoven. Request by DMA devices for using the bus are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, etc.
- Since the processor originates most memory access cycles, the DMA controller can be said to “steal” memory cycles from the processor. Hence, this interweaving technique is usually called *cycle stealing*
- The DMA controller may transfer a block of data without interruption. This is called *block/burst mode*

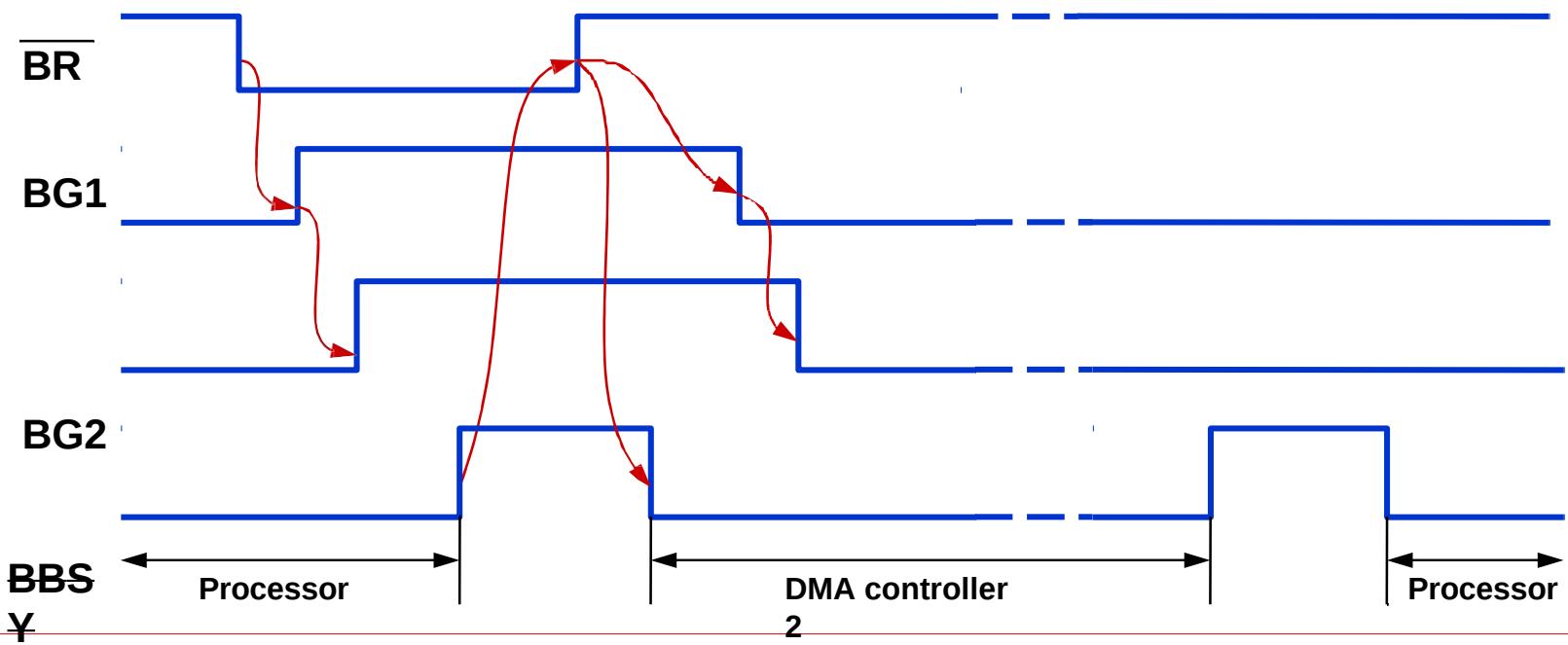
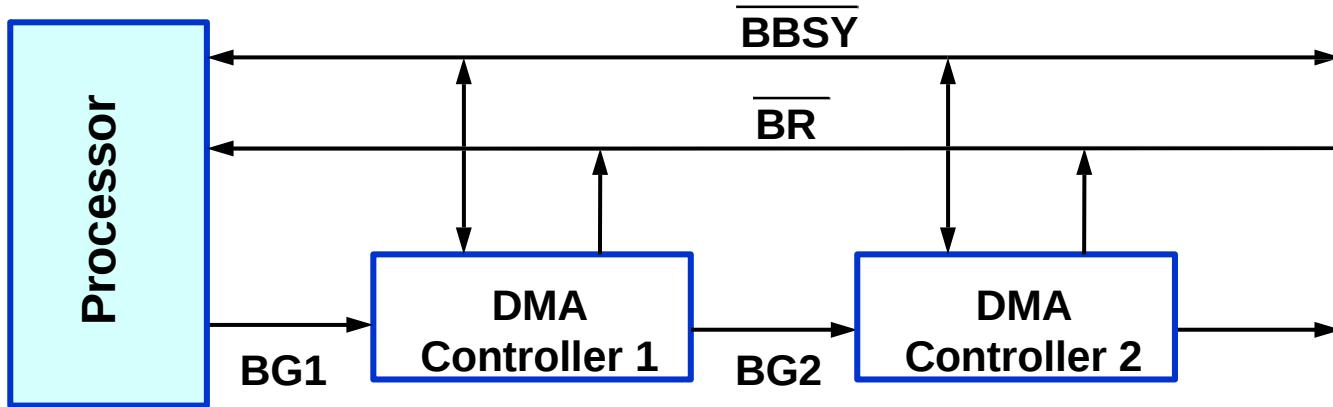
# Bus Arbitration

- A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve this problem, an arbitration procedure on bus is needed
- The device that is allowed to initiate data transfer on the bus at any given time is called the bus master. When the current master relinquishes control of the bus, another device can acquire this status
- Bus arbitration is the process by which the next device to become the bus master take into account the needs of various devices by establishing a priority system for gaining access to the bus

# Bus Arbitration

- There are two approaches to bus arbitration
  - ◆ Centralized and distributed
- In centralized arbitration, a single bus arbiter performs the required arbitration
- In distributed arbitration, all devices participate in the selection of the next bus master

# Centralized Arbitration



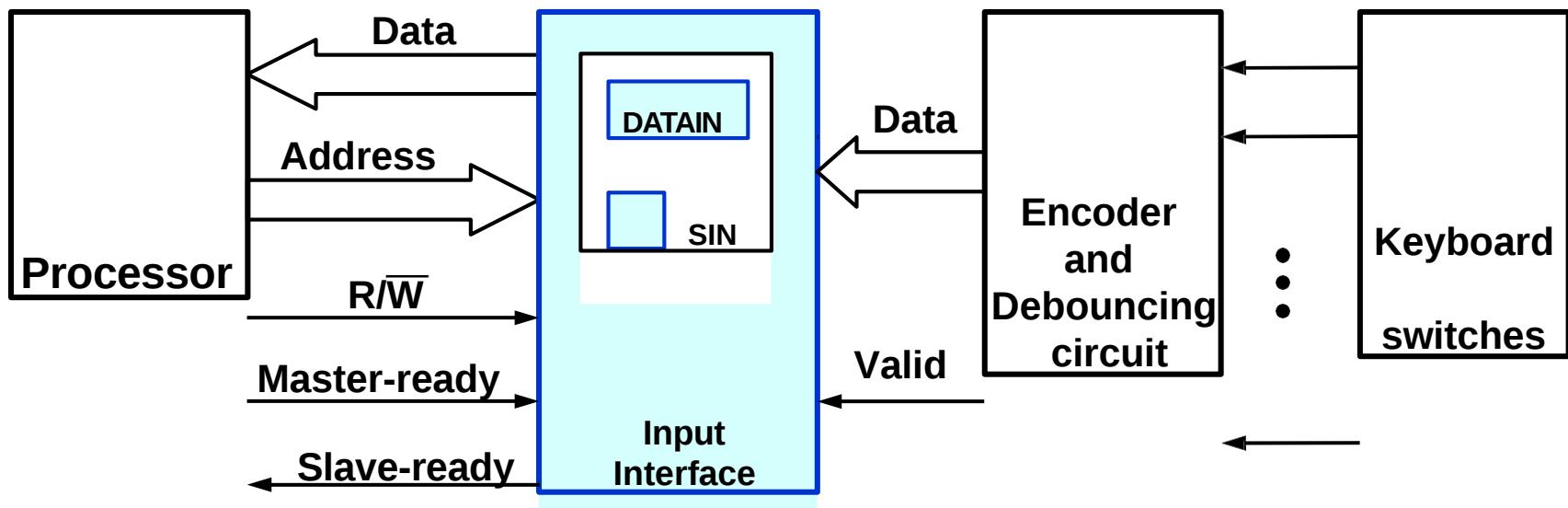
# Buses

- A bus protocol is the set of rules that govern the behavior of various devices connected to the bus as to when to place information on the bus, assert control signals, and so on
- In a synchronous bus, all devices derive timing information from a common clock line. Equal spaced pulses on this line define equal time intervals
- In the simplest form of a synchronous bus, each of these intervals constitutes a bus cycle during which one data transfer can take place

# Interface Circuits

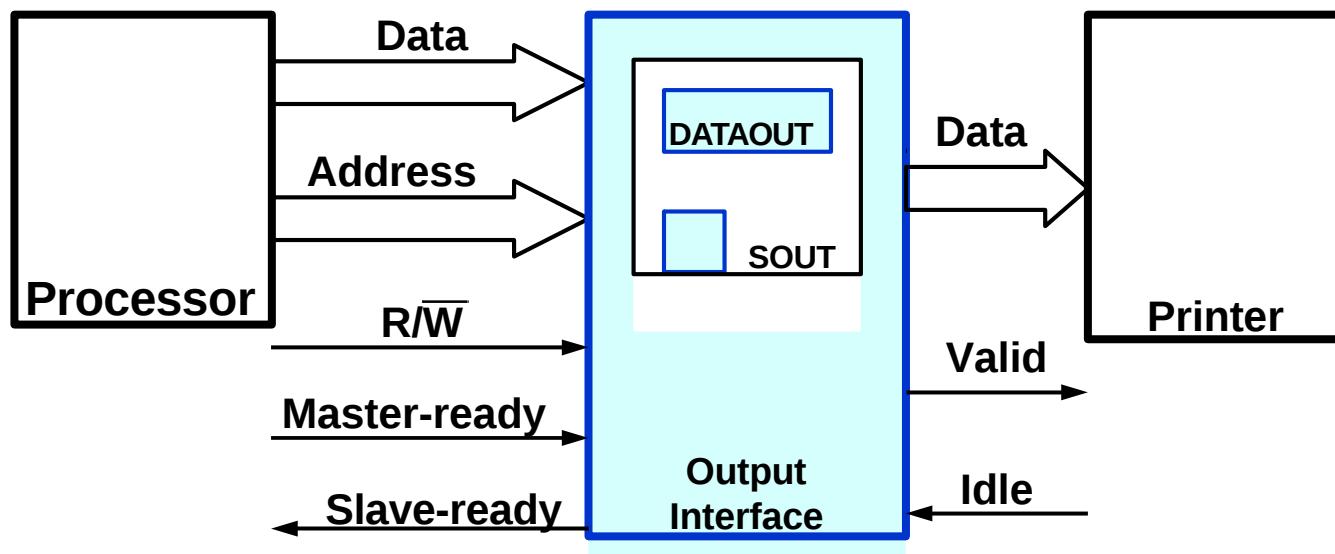
## ➤ Keyboard to processor connection

- ◆ When a key is pressed, the Valid signal changes from 0 to 1, causing the ASCII code to be loaded into DATAIN and SIN to be set to 1
- ◆ The status flag SIN is cleared to 0 when the processor reads the contents of the DATAIN register



# Printer to Processor Connection

- The interface contains a data register, DATAOUT, and a status flag, SOUT
  - ◆ The SOUT flag is set to 1 when the printer is ready to accept another character, and it is cleared to 0 when a new character is loaded into DATAOUT by the processor
  - ◆ When the printer is ready to accept a character, it asserts its idle signal



# Serial Port

- A serial port is used to connect the processor to I / O devices that require transmission of data one bit at a time
- The key feature of an interface circuit for a serial port is that it is capable of communicating in a bit-serial fashion on the device side and in a bit-parallel fashion on the bus side
- The transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability

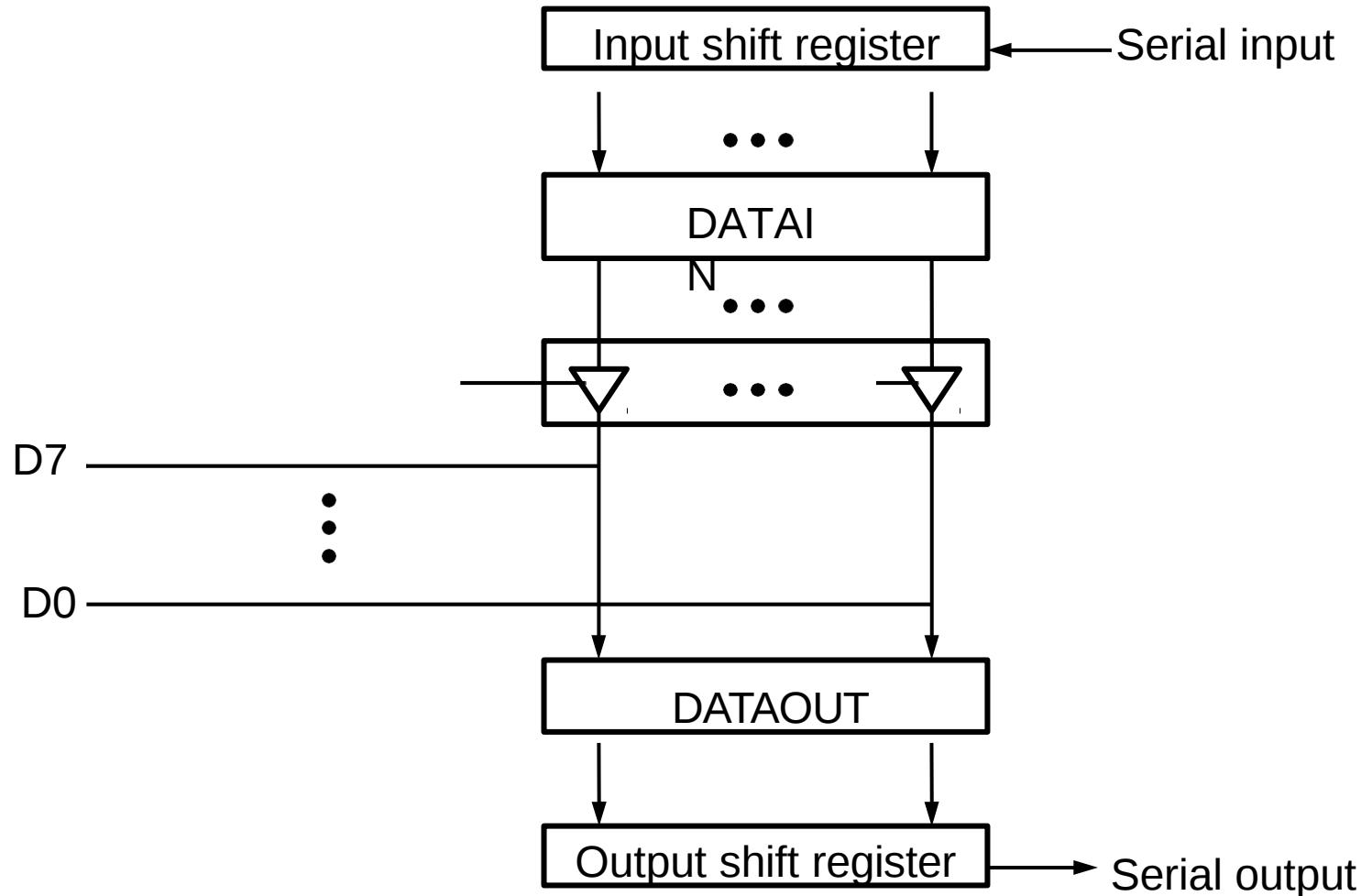
# Shift Register

8 bit register == 8 bit data

4 modes

1. Serial In Serial Out (SISO)
2. Serial In Parallel Out (SIPO)
3. Parallel In Serial Out (PISO)
4. Parallel In Parallel Out (PIPO)

# A Serial Interface

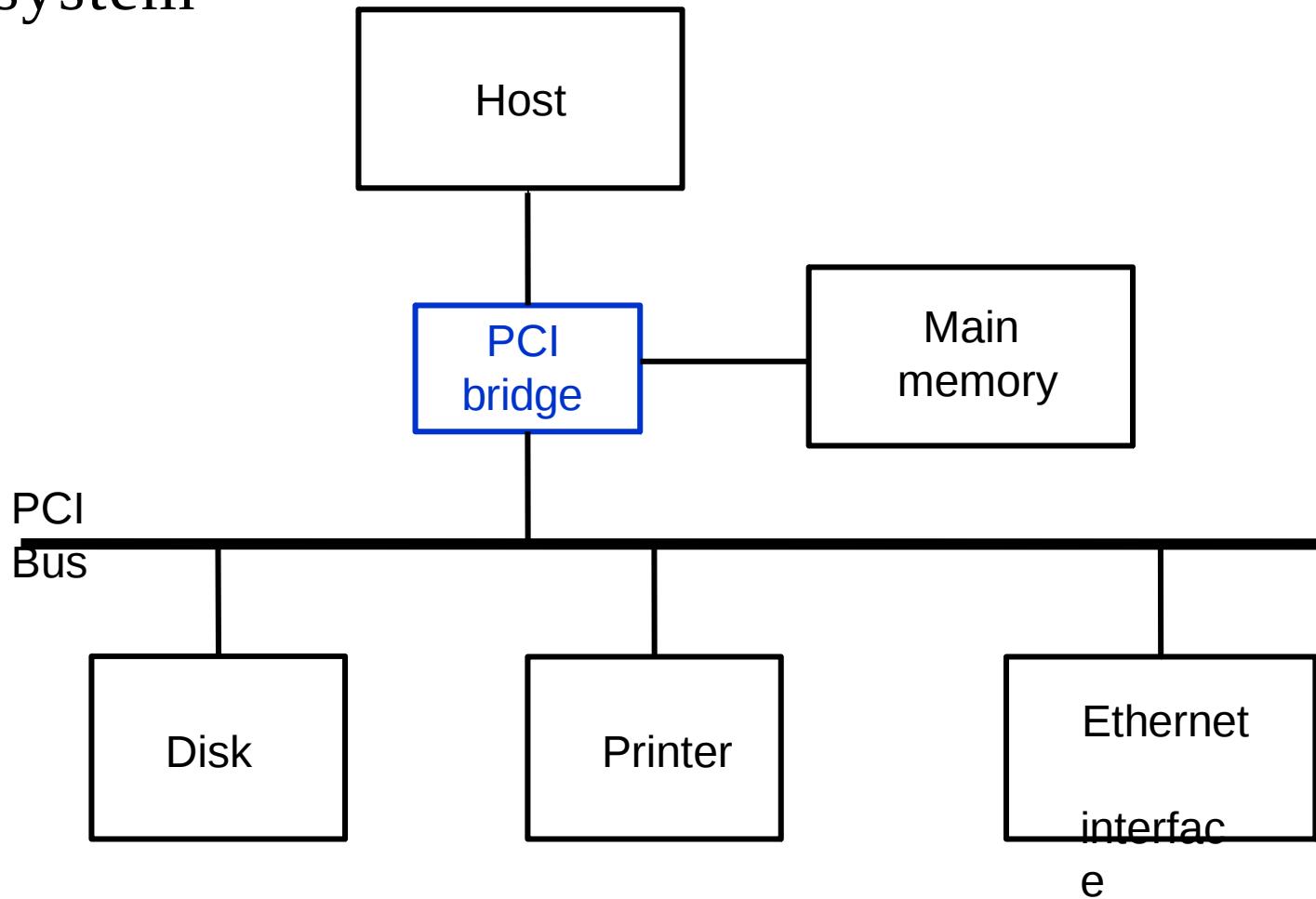


# Standard I/O Interfaces

- The processor bus is the bus defined by the signals on the processor chip itself. Devices that require a very high speed connection to the processor, such as the main memory, may be connected directly to this bus
- The motherboard usually provides another bus that can support more devices.
- The two buses are interconnected by a circuit, which we called a bridge, that translates the signals and protocols of one bus into those of the other
- It is impossible to define a uniform standards for the processor bus. The structure of this bus is closely tied to the architecture of the processor
- The expansion bus is not subject to these limitations, and therefore it can use a standardized signaling structure

# Peripheral Component Interconnect Bus

- Use of a PCI bus in a computer system



# PCI Bus

- The bus support three independent address spaces: memory, I / O, and configuration.
- The I / O address space is intended for use with processors, such Pentium, that have a separate I / O address space.
- However, the system designer may choose to use memory-mapped I / O even when a separate I / O address space is available
- The configuration space is intended to give the PCI its plug-and-play capability.
  - ◆ A 4-bit command that accompanies the address identifies which of the three spaces is being used in a given data transfer operation

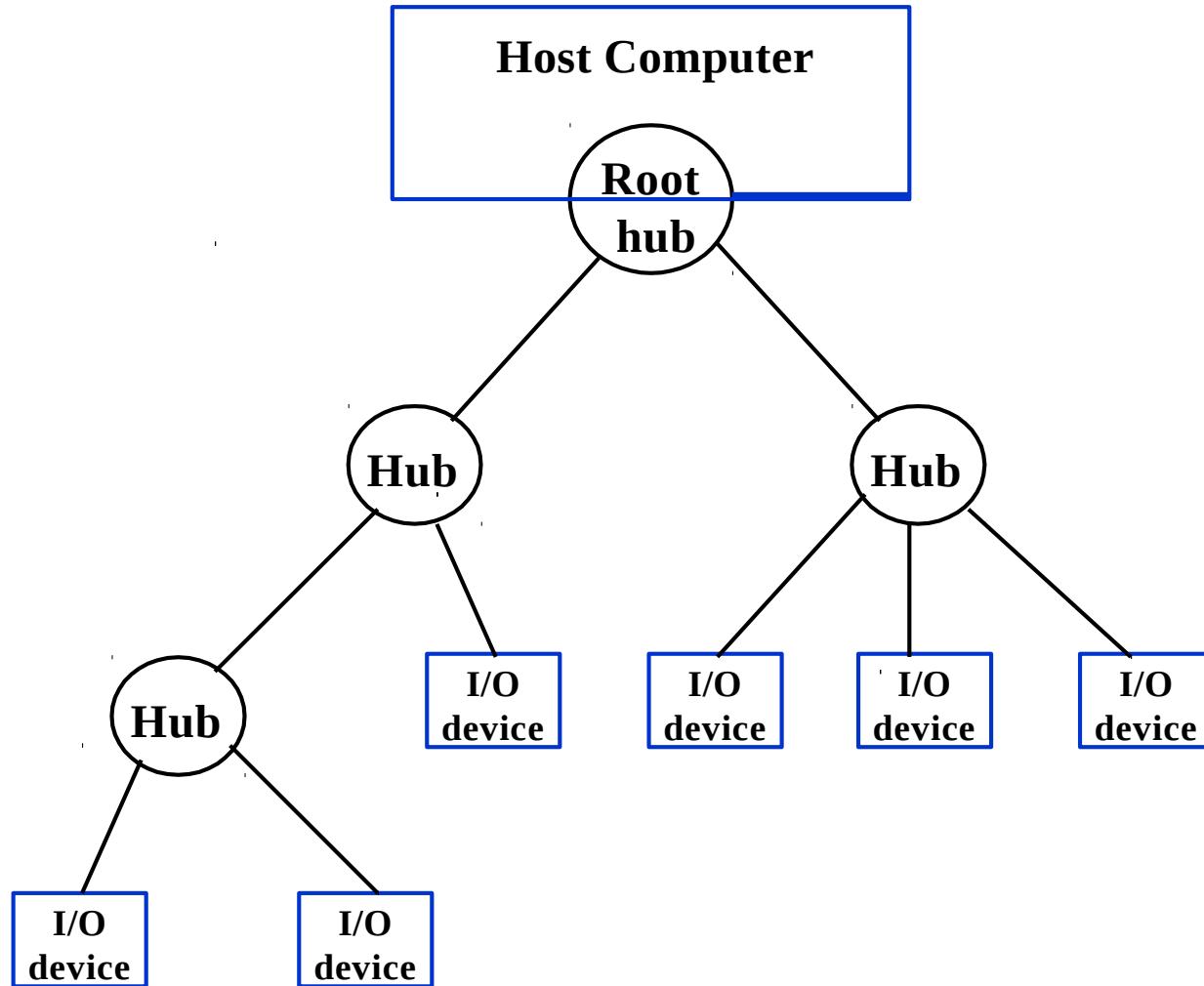
# Universal Serial Bus (USB)

- The USB has been designed to meet several key objectives
  - ◆ Provide a simple, low-cost, and easy to use interconnection system that overcomes the difficulties due to the limited number of I / O ports available on a computer
  - ◆ Accommodate a wide range of data transfer characteristics for I / O devices, including telephone and Internet connections
  - ◆ Enhance user convenience through a “plug-and-play” mode of operation

# USB Structure

- A serial transmission format has been chosen for the USB because a serial bus satisfies the low-cost and flexibility requirements
- Clock and data information are encoded together and transmitted as a single signal
  - ◆ Hence, there are no limitations on clock frequency or distance arising from data skew
- To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure
  - ◆ Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I / O device
  - ◆ At the root of the tree, a root hub connects the entire tree to the host computer

# USB Tree Structure



# USB Tree Structure

- The tree structure enables many devices to be connected while using only simple point-to-point serial links
- Each hub has a number of ports where devices may be connected, including other hubs
- In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports
  - ◆ As a result, a message sent by the host computer is broadcast to all I / O devices, but only the addressed device will respond to that message
- A message sent from an I / O device is sent only upstream towards the root of the tree and is not seen by other devices
  - ◆ Hence, USB enables the host to communicate with the I / O devices, but it does not enable these devices to communicate with each other

# USB Protocols

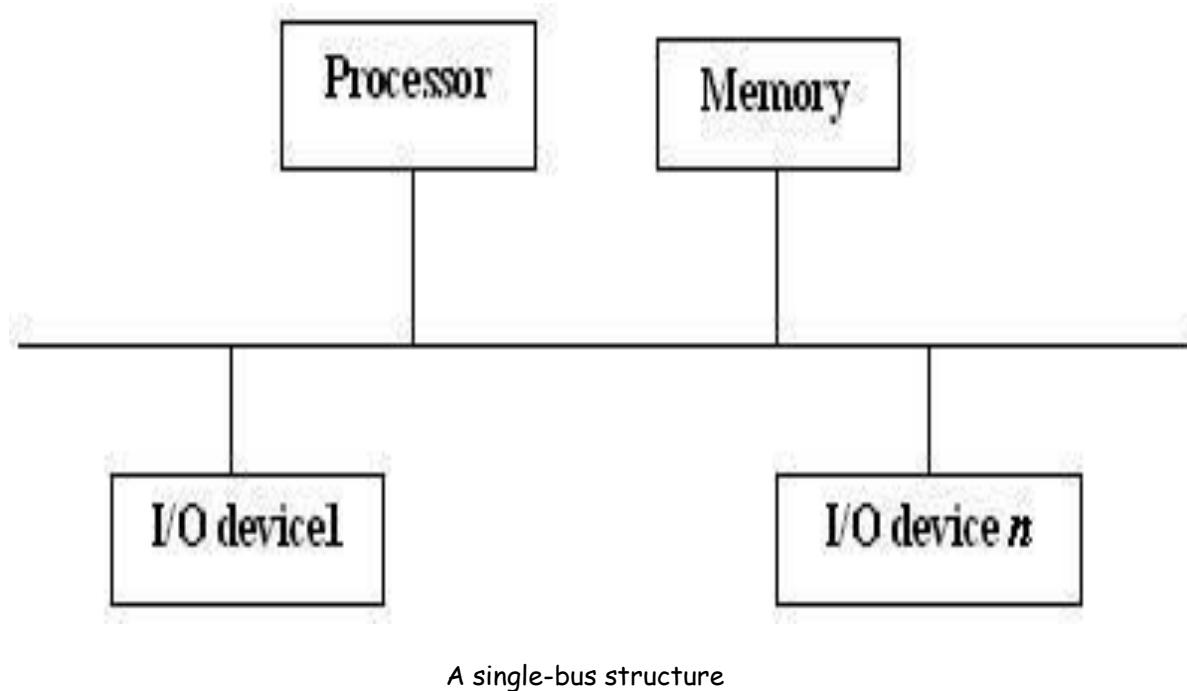
- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information
- The information transferred on the USB can be divided into two broad categories: control and data
  - ◆ Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error
  - ◆ Data packets carry information that is delivered to a device. For example, input and output data are transferred inside data packets

# INPUT/OUTPUT ORGANIZATION

## Introduction

A general purpose computer should have the ability to exchange information with a wide range of devices in varying environments. Computers can communicate with other computers over the Internet and access information around the globe. They are an integral part of home appliances, manufacturing equipment, transportation systems, banking and point-of-sale terminals. In this chapter, we study the various ways in which I/O operations are performed.

## 4.1 Accessing I/O Devices



A simple arrangement to connect I/O devices to a computer is to use a single bus arrangement, as shown in above figure. Each I/O device is assigned a unique set of address. When the processor places a particular address on the address lines, the device that recognizes this address responds to the commands issued on the control lines. The processor requests either a read or a write operation which is transferred over the data lines. When I/O devices and the memory share the same address space, the arrangement is called memory-mapped I/O.

Consider, for instance, with memory-mapped I/O, if DATAIN is the address of the input buffer of the keyboard

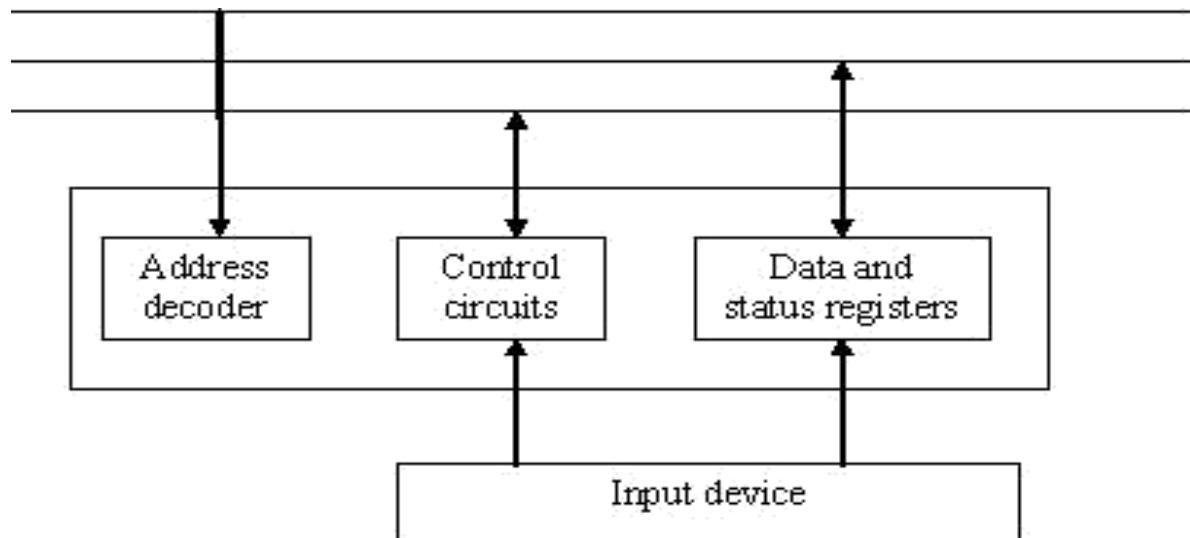
Move DATAIN, R0

And DATAOUT is the address of the output buffer of the display/printer

Move R0, DATAOUT

This sends the contents of register R0 to location DATAOUT, which may be the output data buffer of a display unit or a printer.

Most computer systems use memory-mapped I/O. Some processors have special I/O instructions to perform I/O transfers. The hardware required to connect an I/O device to the bus is shown below:



I/O interface for an input device

The address decoder enables the device to recognize its address when this address appears on the address lines. The data register holds the data. The status register contains information. The address decoder, data and status registers and controls required to coordinate I/O transfers constitutes interface circuit

For eg: Keyboard, an instruction that reads a character from the keyboard should be executed only when a character is available in the input buffer of the keyboard interface. The processor repeatedly checks a status flag to achieve the synchronization between processor and I/O device, which is called as program-controlled I/O.

Two commonly used mechanisms for implementing I/O operations are:

- Interrupts and
- Direct memory access

**Interrupts:** synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation.

**Direct memory access:** For high speed I/O devices. The device interface transfer data directly to or from the memory without informing the processor.

## 4.2 Interrupts

There are many situations where other tasks can be performed while waiting for an I/O device to become ready. A hardware signal called an Interrupt will alert the processor when an I/O device becomes ready. Interrupt-request line is usually dedicated for this purpose.

For example, consider, COMPUTE and PRINT routines. The routine executed in response to an interrupt request is called interrupt-service routine. Transfer of control through the use of interrupts happens. The processor must inform the device that its request has been recognized by sending interrupt-acknowledge signal. One must therefore know the difference between Interrupt Vs Subroutine. Interrupt latency is concerned with saving information in registers will increase the delay between the time an interrupt request is received and the start of execution of the interrupt-service routine.

### Interrupt hardware

Most computers have several I/O devices that can request an interrupt. A single interrupt request line may be used to serve n devices.

#### Enabling and Disabling Interrupts

All computers fundamentally should be able to enable and disable interruptions as desired. Again reconsider the COMPUTE and PRINT example. When a device activates the interrupt-request signal, it keeps this signal activated until it learns that the processor has accepted its request. When interrupts are enabled, the following is a typical scenario:

- The device raises an interrupt request.
- The processor interrupts the program currently being executed.
- Interrupts are disabled by changing the control bits in the processor status register (PS).
- The device is informed that its request has been recognized and deactivates the interrupt request signal.
- The action requested by the interrupt is performed by the interrupt-service routine.
- Interrupts are enabled and execution of the interrupted program is resumed.

### Handling multiple devices

While handling multiple devices, the issues concerned are:

- How can the processor recognize the device requesting an interrupt?
- How can the processor obtain the starting address of the appropriate routine?
- Should a device be allowed to interrupt the processor while another interrupt is being serviced?
- How should two or more simultaneous interrupt requests be handled?

### **Vectored interrupts**

A device requesting an interrupt may identify itself (by sending a special code) directly to the processor, so that the processor considers it immediately.

### **Interrupt nesting**

The processor should continue to execute the interrupt-service routine till completion, before it accepts an interrupt request from a second device. Privilege exception means they execute privileged instructions. Individual interrupt-request and acknowledge lines can also be implemented. Implementation of interrupt priority using individual interrupt-request and acknowledge lines has been shown in figure 4.7.

### **Simultaneous requests**

The processor must have some mechanisms to decide which request to service when simultaneous requests arrive. Here, daisy chain and arrangement of priority groups as the interrupt priority schemes are discussed. Priority based simultaneous requests are considered in many organizations.

### **Controlling device requests**

At the device end, an interrupt enable bit determines whether it is allowed to generate an interrupt request. At the processor end, it determines whether a given interrupt request will be accepted.

### **Exceptions**

The term exception is used to refer to any event that causes an interruption. Hence, I/O interrupts are one example of an exception.

- Recovery from errors - These are techniques to ensure that all hardware components are operating properly.
- Debugging - find errors in a program, trace and breakpoints (only at specific points selected by the user).
- Privilege exception - execute privileged instructions to protect OS of a computer.

### **Use of interrupts in Operating Systems**

Operating system is system software which is also termed as resource manager, as it manages all variety of computer peripheral devices efficiently. Different issues addressed by the operating systems are: Assign priorities among jobs, Security and protection features, incorporate interrupt-service routines for all devices and Multitasking, time slice, process, program state, context switch and others.

#### **4.4 Direct Memory Access**

As we have seen earlier, the two commonly used mechanisms for implementing I/O operations are:

- Interrupts and
- Direct memory access

**Interrupts:** synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation

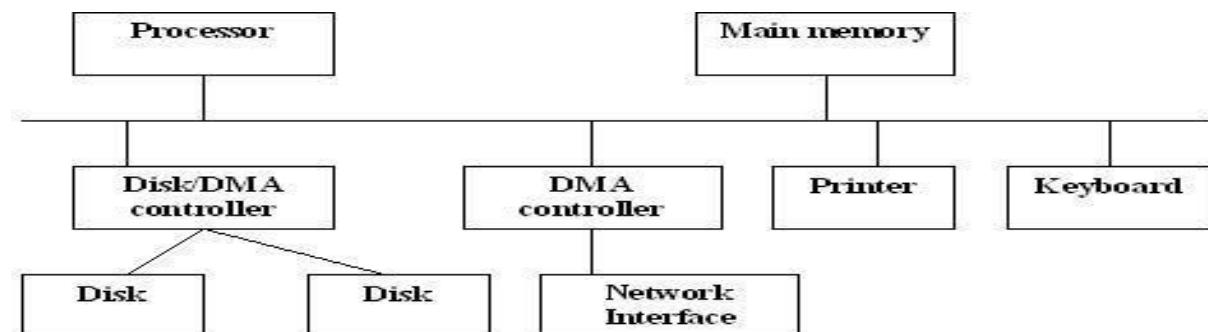
**Direct memory access:**

Basically for high speed I/O devices, the device interface transfer data directly to or from the memory without informing the processor. When interrupts are used, additional overhead involved with saving and restoring the program counter and other state information. To transfer large blocks of data at high speed, an alternative approach is used. A special control unit will allow transfer of a block of data directly between an external device and the main memory, without continuous intervention by the processor.

DMA controller is a control circuit that performs DMA transfers, is a part of the I/O device interface. It performs functions that normally be carried out by the processor. DMA controller must increment the memory address and keep track of the number of transfers. The operations of DMA controller must be under the control of a program executed by the processor. To initiate the transfer of block of words, the processor sends the starting address, the number of words in the block and the direction of the transfer. On receiving this information, DMA controller transfers the entire block and informs the processor by raising an interrupt signal. While a DMA transfer is taking place, the processor can be used to execute another program. After the DMA transfer is completed, the processor can return to the program that requested the transfer.

Three registers in a DMA interface are:

- Starting address
- Word count
- Status and control flag



Use of DMA controllers in a computer system

A conflict may arise if both the processor and a DMA controller or two DMA controllers try to use the bus at the same time to access the main memory. To resolve this, an arbitration procedure is implemented on the bus to coordinate the activities of all devices requesting memory transfers.

### **Bus Arbitration**

The device that is allowed to initiate data transfers on the bus at any given time is called the bus master. Arbitration is the process by which the next device to become the bus master is selected and bus mastership is transferred to it. The two approaches are centralized and distributed arbitrations.

In centralized, a single bus arbiter performs the required arbitration whereas in distributed, all device participate in the selection of the next bus master. The bus arbiter may be the processor or a separate unit connected to the bus. The processor is normally the bus master unless it grants bus mastership to one of the DMA controllers. A simple arrangement for bus arbitration using daisy chain and a distributed arbitration scheme are discussed in figure 4.20 and 4.22 respectively.

In Centralized arbitration, A simple arrangement for bus arbitration using a daisy chain shows the arbitration solution. A rotating priority scheme may be used to give all devices an equal chance of being serviced (BR1 to BR4). In Distributed arbitration, all devices waiting to use the bus have equal responsibility in carrying out the arbitration process, without using a central arbiter. The drivers are of the open-collector type. Hence, if the input to one driver is equal to 1 and the input to another driver connected to the same bus line is equal to 0 the bus will be in the low-voltage state. This uses ARB0 to ARB3.

## **4.5 Buses**

The Primary function of the bus is to provide a communication path for the transfer of data. It must also look in to,

- When to place information on the bus?
- When to have control signals?

Some bus protocols are set. These involve data, address and control lines. A variety of schemes have been devised for the timing of data transfers over a bus. They are:

### **Synchronous and Asynchronous schemes**

Bus master is an initiator. Usually, processor acts as master. But under DMA setup, any other device can be master. The device addressed by the master is slave or target.

### **Synchronous bus**

All devices derive timing information from a common clock line. Equally spaced pulses on this line define equal time intervals. Each of these intervals constitutes a bus cycle during which one data transfer can take place. Timing of an input/output transfer on a synchronous bus is shown in figure 4.23.

### Asynchronous bus

This is a scheme based on the use of a handshake between the master and the slave for controlling data transfers on the bus. The common clock is replaced by two timing control lines, master-ready and slave-ready. The first is asserted by the master to indicate that it is ready for a transaction and the second is a response from the slave. The master places the address and command information on the bus. It indicates to all devices that it has done so by activating the master-ready line. This causes all devices on the bus to decode the address. The selected slave performs the required operation and informs the processor it has done so by activating the slave-ready line. A typical handshake control of data transfer during an input and an output operations are shown in figure 4.26 and 4.27 respectively. The master waits for slave-ready to become asserted before it removes its signals from the bus. The handshake signals are fully interlocked. A change of state in one signal is followed by a change in the other signal. Hence this scheme is known as a full handshake.

### 4.6 Interface Circuits

An I/O interface consists of the circuitry required to connect an I/O device to a computer bus. On one side of the interface, we have bus signals. On the other side, we have a data path with its associated controls to transfer data between the interface and the I/O device - port. We have two types:

- Serial port and
- Parallel port

A parallel port transfers data in the form of a number of bits (8 or 16) simultaneously to or from the device. A serial port transmits and receives data one bit at a time. Communication with the bus is the same for both formats. The conversion from the parallel to the serial format, and vice versa, takes place inside the interface circuit. In parallel port, the connection between the device and the computer uses a multiple-pin connector and a cable with as many wires. This arrangement is suitable for devices that are physically close to the computer. In serial port, it is much more convenient and cost-effective where longer cables are needed.

Typically, the functions of an I/O interface are:

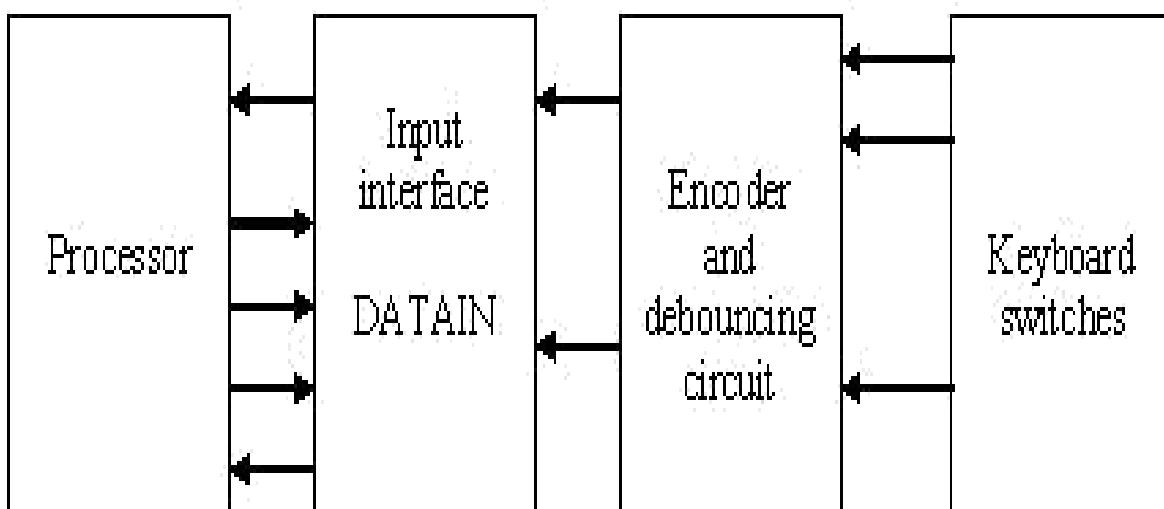
- Provides a storage buffer for at least one word of data
- Contains status flags that can be accessed by the processor to determine whether the buffer is full or empty
- Contains address-decoding circuitry to determine when it is being addressed by the processor
- Generates the appropriate timing signals required by the bus control scheme
- Performs any format conversion that may be necessary to transfer data between the bus and the I/O device, such as parallel-serial conversion in the case of a serial port.

### Parallel Port

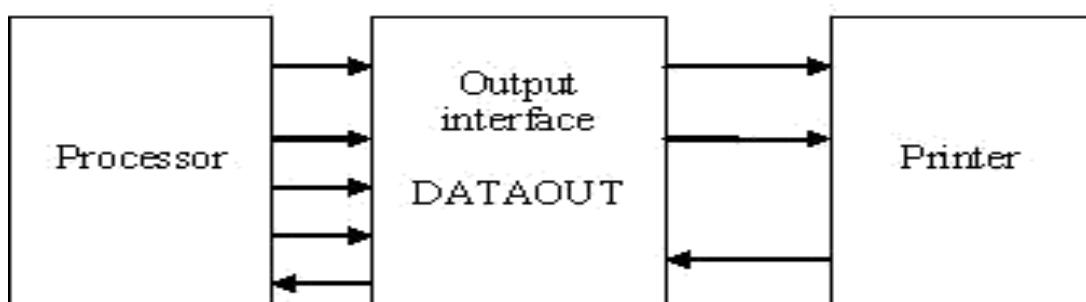
The hardware components needed for connecting a keyboard to a processor Consider the circuit of input interface which encompasses (as shown in below figure):

- Status flag, SIN
- R/~W
- Master-ready
- Address decoder

A detailed figure showing the input interface circuit is presented in figure 4.29. Now, consider the circuit for the status flag (figure 4.30). An edge-triggered D flip-flop is used along with read-data and master-ready signals



Keyboard to processor connection



Printer to processor connection

The hardware components needed for connecting a printer to a processor are:  
the circuit of output interface, and

- Slave-ready
- R/~W
- Master-ready
- Address decoder
- Handshake control

The input and output interfaces can be combined into a single interface. The general purpose parallel interface circuit that can be configured in a variety of ways. For increased flexibility, the circuit makes it possible for some lines to serve as inputs and some lines to serve as outputs, under program control.

### **Serial Port**

A serial interface circuit involves - Chip and register select, Status and control, Output shift register, DATAOUT, DATAIN, Input shift register and Serial input/output - as shown in figure 4.37.

### **4.7 Standard I/O interfaces**

Consider a computer system using different interface standards. Let us look in to Processor bus and Peripheral Component Interconnect (PCI) bus. These two buses are interconnected by a circuit called bridge. It is a bridge between processor bus and PCI bus. An example of a computer system using different interface standards is shown in figure 4.38. The three major standard I/O interfaces discussed here are:

- PCI (Peripheral Component Interconnect)
- SCSI (Small Computer System Interface)
- USB (Universal Serial Bus)

### **Peripheral Component Interconnect (PCI) Bus**

The topics discussed under PCI are: Data Transfer, Use of a PCI bus in a computer system, A read operation on the PCI bus, Device configuration and Other electrical characteristics. Use of a PCI bus in a computer system is shown in figure 4.39 as a representation.

Host, main memory and PCI bridge are connected to disk, printer and Ethernet interface through PCI bus. At any given time, one device is the bus master. It has the right to initiate data transfers by issuing read and write commands. A master is called an initiator in PCI terminology. This is either processor or DMA controller. The addressed device that responds to read and write commands is called a target. A complete transfer operation on the bus, involving an address and a burst of data, is called a transaction. Device configuration is also discussed.

- The PCI bus is a good example of a system bus that grew out of the need for standardization.
- It supports the functions found on a processor bus bit in a standardized format that is independent of any particular processor.
- Devices connected to the PCI bus appear to the processor as if they were connected directly to the processor bus. They are assigned addresses in the memory address space of the processor.
- The PCI follows a sequence of bus standards that were used primarily in IBM PCs. Early PCs used the 8-bit XT bus, whose signals closely mimicked those of Intel's 80x86 processors. Later, the 16-bit bus used on the PC AT computers became known as the **ISA bus**. Its extended 32-bit version is known as the **EISA bus**.
- Other buses developed in the eighties with similar capabilities are the Microchannel used in IBM PCs and the NuBus used in Macintosh computers.
- The PCI was developed as a low-cost bus that is truly processor independent. Its design anticipated a rapidly growing demand for bus bandwidth to support high-speed disks and graphic and video devices, as well as the specialized needs of multiprocessor systems. As a result, the PCI is still popular as an industry standard almost a decade after it was first introduced in 1992.
- **An important feature that the PCI pioneered is a plug-and-play capability** for connecting I/O devices. To connect a new device, the user simply connects the device interface board to the bus. The software takes care of the rest.

### Data Transfer

- In today's computers, most memory transfers involve a burst of data rather than just one word. The reason is that modern processors include a cache memory. Data are transferred between the cache and the main memory in burst of several words each.
- The words involved in such a transfer are stored at successive memory locations. When the processor (actually the cache controller) specifies an address and requests a read operation from the main memory, the memory responds by sending a sequence of data words starting at that address. Similarly, during a write operation, the processor sends a memory address followed by a sequence of data words, to be written in successive memory locations starting at the address.
- The PCI is designed primarily to support this mode of operation. A read or write operation involving a single word is simply treated as a burst of length one.
- The bus supports three independent address spaces: **memory, I/O, and configuration**. The first two are self-explanatory. The I/O address space is intended for use with processors, such as Pentium, that have a separate I/O address space. However, as noted, the system designer may choose to use memory-mapped I/O even when a separate I/O address space is available.
- In fact, this is the approach recommended by the PCI its plug-and-play capability. A 4-bit command that accompanies the address identifies which of the three spaces is being used in a given data transfer operation.
- The signaling convention on the PCI bus is similar to the one used, we assumed that the master maintains the address information on the bus until data transfer is completed. But, this is not necessary. The address is needed only long enough for the slave to be selected. The slave can store the address in its internal buffer. Thus, the address is needed on the bus for one clock cycle only, freeing the address lines to be used for

sending data in subsequent clock cycles. The result is a significant cost reduction because the number of wires on a bus is an important cost factor. This approach is used in the PCI bus.

- At any given time, one device is the bus master. It has the right to initiate data transfers by issuing read and write commands. A **master** is called an initiator in PCI terminology. This is either a processor or a DMA controller.
- The addressed device that responds to read and write commands is called a target.

### Device Configuration

- When an I/O device is connected to a computer, several actions are needed to configure both the device and the software that communicates with it.
- The PCI simplifies this process by incorporating in each I/O device interface a small configuration ROM memory that stores information about that device. The configuration ROMs of all devices is accessible in the configuration address space.
- The PCI initialization software reads these ROMs whenever the system is powered up or reset. In each case, it determines whether the device is a printer, a keyboard, an Ethernet interface, or a disk controller. It can further learn about various device options and characteristics. Devices are assigned addresses during the initialization process. This means that during the bus configuration operation, devices cannot be accessed based on their address, as they have not yet been assigned one. Hence, the configuration address space uses a different mechanism. Each device has an input signal called Initialization Device Select, IDSEL#.
- The PCI bus has gained great popularity in the PC world. It is also used in many other computers, such as SUNs, to benefit from the wide range of I/O devices for which a PCI interface is available.
- In the case of some processors, such as the Compaq Alpha, the PCI-processor bridge circuit is built on the processor chip itself, further simplifying system design and packaging.

### SCSI Bus

It is a standard bus defined by the American National Standards Institute (ANSI). A controller connected to a SCSI bus is an initiator or a target. The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:

- The SCSI controller contends for control of the bus (initiator).
- When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
- The target starts an output operation. The initiator sends a command specifying the required read operation.
- The target sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.

- The acronym SCSI stands for Small Computer System Interface. It refers to a standard bus defined by the American National Standards Institute (ANSI) under the designation X3.131 .
- In the original specifications of the standard, devices such as disks are connected to a computer via a 50-wire cable, which can be up to 25 meters in length and can transfer data at rates up to **5 megabytes/s**.
- The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years. SCSI-2 and SCSI-3 have been defined, and each has several options.
- A SCSI bus may have **eight data lines**, in which case it is called a narrow bus and transfers data one byte at a time.
- Alternatively, a wide SCSI bus has 16 data lines and transfers data 16 bits at a time.
- There are also several options for the electrical signaling scheme used.
- **Devices connected to the SCSI bus are not part of the address space of the processor** in the same way as devices connected to the processor bus.
- The SCSI bus is connected to the processor bus through a **SCSI controller**. This controller uses **DMA** to transfer data packets from the main memory to the device, or vice versa.
- A packet may contain a block of data, commands from the processor to the device, or status information about the device.
- Communication with a disk drive differs substantially from communication with the main memory.
- A controller connected to a SCSI bus is one of two types - an initiator or a target.
- An **initiator** has the ability to select a particular target and to send commands specifying the operations to be performed.
- Clearly, the controller on the processor side, such as the SCSI controller, must be able to operate as an initiator.
- The disk controller operates as a **target**. It carries out the commands it receives from the initiator.
- The initiator establishes a logical connection with the intended target. Once this connection has been established, it can be suspended and restored as needed to transfer commands and bursts of data.
- While a particular connection is suspended, other device can use the bus to transfer information.
- **This ability to overlap data transfer requests is one of the key features of the SCSI bus that leads to its high performance.**
- Data transfers on the SCSI bus are always controlled by the **target controller**. To send a command to a target, an initiator requests control of the bus and, after winning arbitration, selects the controller it wants to communicate with and hands control of the bus over to it.

Then the controller starts a data transfer operation to receive a command from the initiator.

The processor sends a command to the SCSI controller, which causes the following sequence of event to take place:

- The SCSI controller, acting as an initiator, contends for control of the bus.
- When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
- The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.

- The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
  - The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.
  - The target transfers the contents of the data buffer to the initiator and then suspends the connection again. Data are transferred either 8 or 16 bits in parallel, depending on the width of the bus.
  - The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of this transfers, the logical connection between the two controllers is terminated.
  - As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
  - The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.
  - This scenario shows that the messages exchanged over the SCSI bus are at a higher level than those exchanged over the processor bus. In this context, a "higher level" means that the messages refer to operations that may require several steps to complete, depending on the device. Neither the processor nor the SCSI controller need be aware of the details of operation of the particular device involved in a data transfer. In the preceding example, the processor need not be involved in the disk seek operation.
- The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation.
  - The target transfers the contents of the data buffer to the initiator and then suspends the connection again.
  - The target controller sends a command to the disk drive to perform another seek operation.
  - As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
  - The SCSI controller sends an interrupt to the processor to inform it that the requested operation has been completed.

The bus signals, arbitration, selection, information transfer and reselection are the topics discussed in addition to the above.

#### Universal Serial Bus (USB)

The USB has been designed to meet several key objectives such as:

- Provide a simple, low-cost and easy to use interconnection system that overcomes the difficulties due to the limited number of I/O ports available on a computer
- Accommodate a wide range of data transfer characteristics for I/O devices, including telephone and Internet connections
- Enhance user convenience through a "plug-and-play" mode of operation

### Port Limitation

Here to add new ports, a user must open the computer box to gain access to the internal expansion bus and install a new interface card. The user may also need to know how to configure the device and the software. And also it is to make it possible to add many devices to a computer system at any time, without opening the computer box.

### Device Characteristics

The kinds of devices that may be connected to a computer cover a wide range of functionality - speed, volume and timing constraints. A variety of simple devices attached to a computer generate data in different asynchronous mode. A signal must be sampled quickly enough to track its highest-frequency components.

### Plug-and-play

Whenever a device is introduced, do not turn the computer off/restart to connect/disconnect a device. The system should detect the existence of this new device automatically, identify the appropriate device-driver software and any other facilities needed to service that device, and establish the appropriate addresses and logical connections to enable them to communicate.

### USB architecture

To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure. Each node has a device called a hub. Root hub, functions, split bus operations - high speed (HS) and Full/Low speed (F/LS).

### 4.8 Concluding remarks

The three basic approaches of I/O transfers are discussed. The simplest technique is programmed I/O, in which the processor performs all the necessary control functions under direct control of program instructions. The second approach is based on the use of interrupts. The third I/O scheme involves DMA, the DMA controller transfers data between an I/O device and the main memory without continuous processor intervention. Access to memory is shared between the DMAQ controller and the processor.

Three popular interconnection standards - PCI, SCSI, USB are discussed. They represent different approaches that meet the needs of various devices and reflect the increasing importance of plug-and-play features that increase user convenience.

**References:**

1. Carl Hamacher, Zvonko Vranesic, Safwat Zaky, *Computer Organization*, fifth edition, Mc-graw Hill higher education.
2. Computer Architecture and Organization, John P. Hayes, 3rd Edition, McGraw Hill.
3. Computer Organization and Architecture - William Stallings Sixth Edition, Pearson/PHI

# )Introduction to Operating System (OS

# :Course Content

- What is an OS.
- What are its key functions.
- The evaluation of OS.
- What are the popular types of OS.
- Basics of UNIX and Windows.
- Advantages of open source OS like Linux.
- Networks OS.

- User
- Application
- Operating System
- Hardware

# ?What is an Operating System

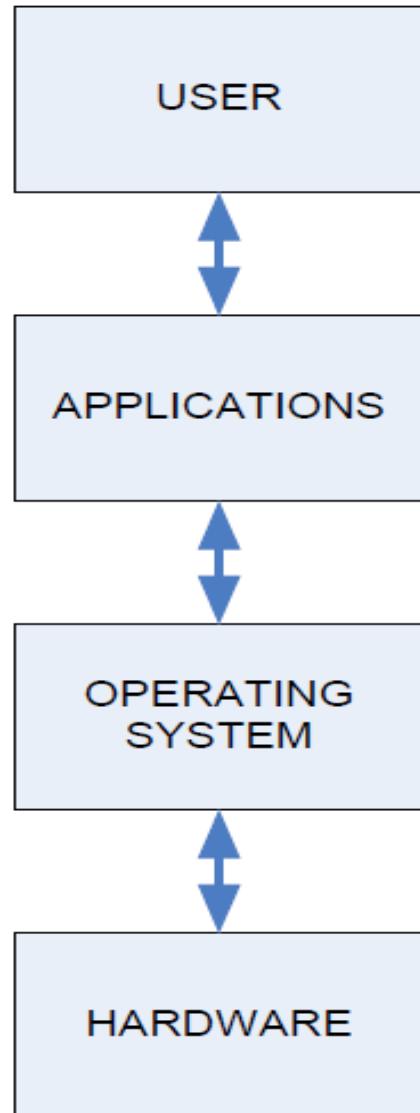
- Computer System = Hardware + Software
- Software = Application Software + System Software(OS)
- An Operating System is a system Software that acts as an intermediary/interface between a **user** of a computer and the **computer hardware**.
- Operating system goals:
  - Execute user programs and make solving user problems easier
  - Make the computer system convenient to use
  - Use the computer hardware in an efficient manner

# Features of OS

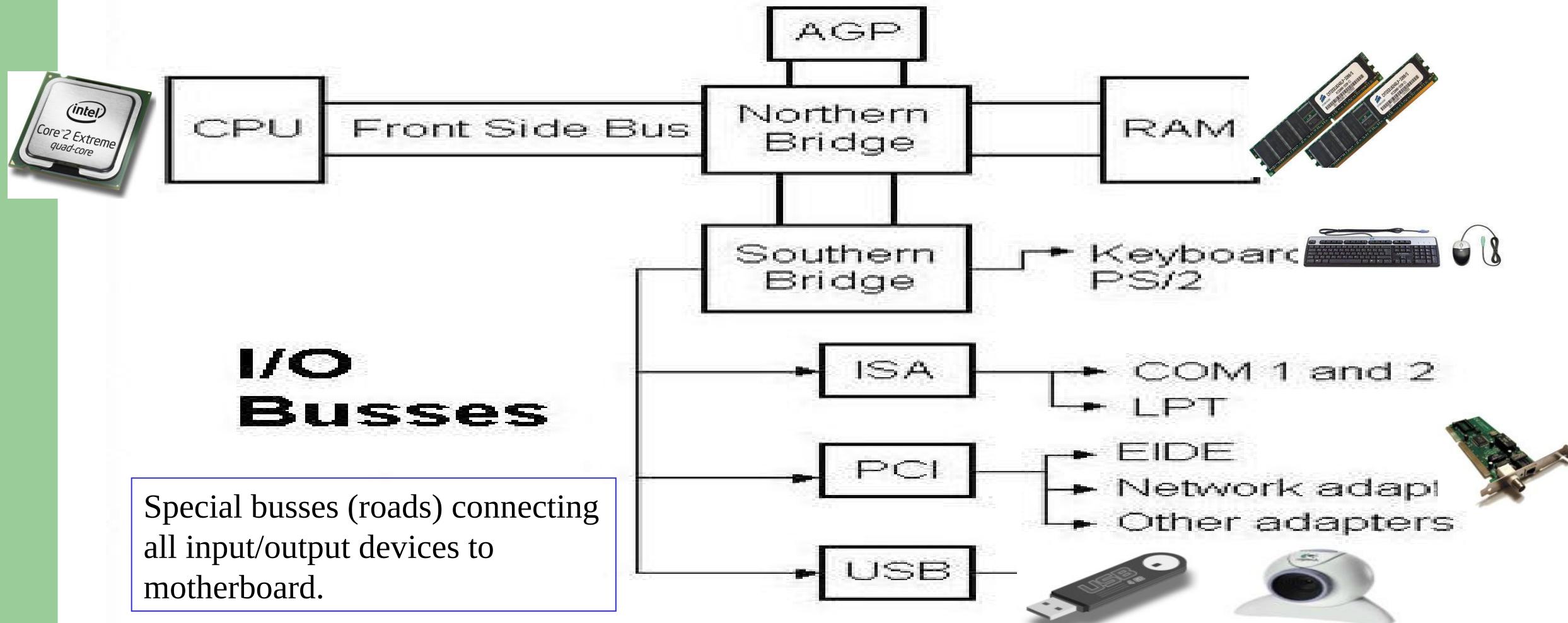
- Convenience
- Efficiency
- Ability to Evolve
- Throughput
- Resource management
- Process management
- Storage management
- Memory management
- Security / Privacy management

# The Structure of Computer Systems

- Accessing computer resources is divided into *layers*.
- Each layer is isolated and only interacts directly with the layer below or above it.
- **If we install a new hardware device**
  - ✓ No need to change anything about the user/applications.
  - ✓ However, you do need to make changes to the operating system.
  - ✓ You need to install the device drivers that the operating system will use to control the new device.
- **If we install a new software application**
  - ✓ No need to make any changes to your hardware.
  - ✓ But we need to make sure the application is supported by the operating system
  - ✓ user will need to learn how to use the new application.
- **If we change the operating system**
  - ✓ Need to make sure that both applications and hardware will compatible with the new operating system.



# Computer Architecture



# CPU – Central Processing Unit

- This is the brain of your computer.
- It performs all of the calculations.
- In order to do its job, the CPU needs commands to perform, and data to work with.
- The instructions and data travel to and from the CPU on the system bus.
- The operating system provides rules for how that information gets back and forth, and how it will be used by the CPU.

# RAM – Random Access Memory

- This is like a desk, or a workspace, where your computer temporarily stores all of the information (data) and instructions (software or program code) that it is currently using.
- Each RAM chip contains millions of address spaces.
- Each address space is the same size, and has its own unique identifying number (address).
- The operating system provides the rules for using these memory spaces, and controls storage and retrieval of information from RAM.
- Device drivers for RAM chips are included with the operating system.

***Problem: If RAM needs an operating system to work, and an operating system needs RAM in order to work, how does your computer activate its RAM to load the operating system?***

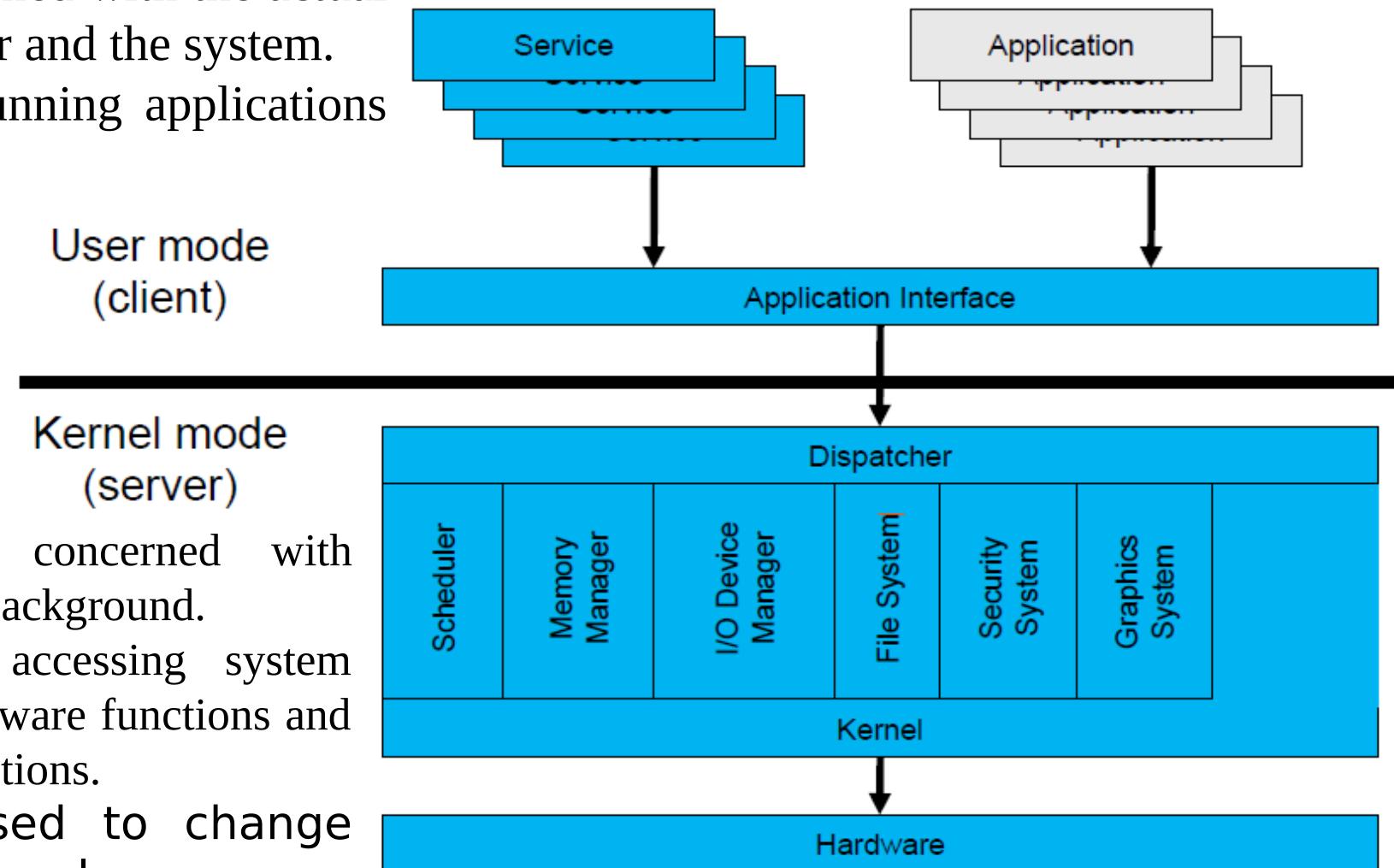
# Operating System Mode

- ❖ The *User Mode* is concerned with the actual interface between the user and the system.
- ❖ It controls things like running applications and accessing files.

User mode  
(client)

Kernel mode  
(server)

- ❖ The *Kernel Mode* is concerned with everything running in the background.
- ❖ It controls things like accessing system resources, controlling hardware functions and processing program instructions.
- ❖ **System calls** are used to change mode from User to Kernel.



# Kernel

- Kernel is a software code that reside in central core of OS. It has complete control over system.
- When operation system boots, kernel is first part of OS to load in main memory.
- Kernel remains in main memory for entire duration of computer session. The kernel code is usually loaded in to protected area of memory.
- Kernel performs it's task like executing processes and handling interrupts in kernel space.
- User performs it's task in user area of memory.
- This memory separation is made in order to prevent user data and kernel data from interfering with each other.
- Kernel does not interact directly with user, but it interacts using SHELL and other programs and hardware.

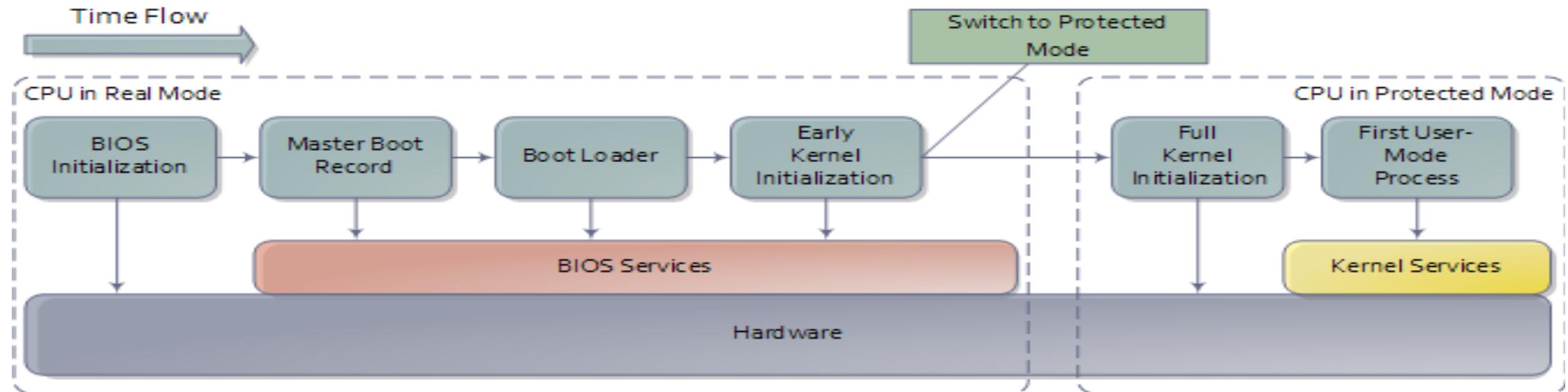
## ...Kernel cont

- Kernel includes:-
  1. **Scheduler**: It allocates the Kernel's processing time to various processes.
  2. **Supervisor**: It grants permission to use computer system resources to each process.
  3. **Interrupt handler** : It handles all requests from the various hardware devices which compete for kernel services.
  4. **Memory manager** : allocates space in memory for all users of kernel service.
- kernel provides services for process management, file management, I/O management, memory management.
- System calls are used to provide these type of services.

# System Call

- **System call** is the programmatic way in which a computer program/user application requests a service from the kernel of the operating system on which it is executed.
- Application program is just a user-process. Due to security reasons , user applications are not given access to privileged resources(the ones controlled by OS).
- When they need to **do any I/O** or have **some more memory** or **spawn a process** or wait for **signal/interrupt**, it requests operating system to facilitate all these. This **request is made through System Call**.
- System calls are also called **software-interrupts**.

# Starting an Operating System(Booting)

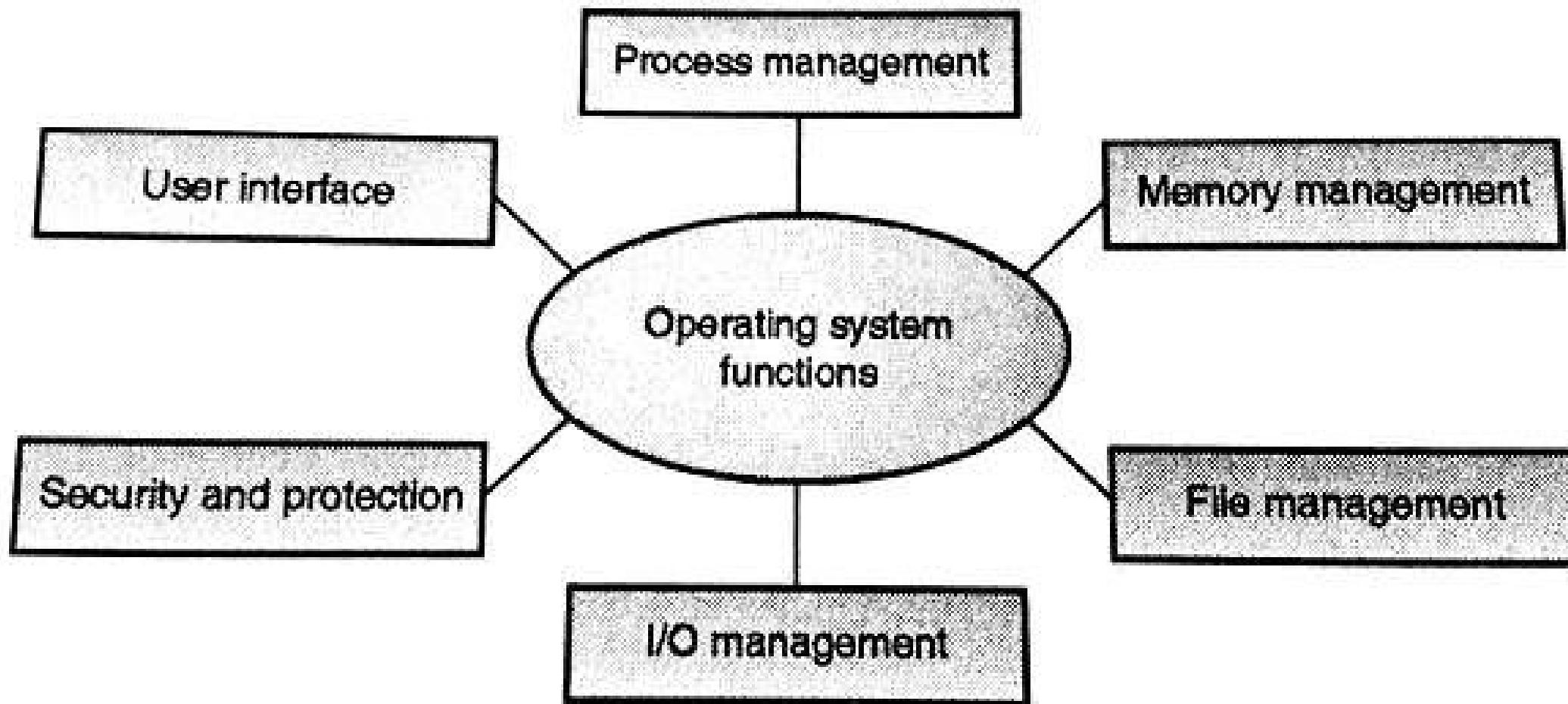


- ✓ Power On Switch sends electricity to the motherboard on a wire called the **Voltage Good** line.
- ✓ If the power supply is good, then the **BIOS (Basic Input/Output System)** chip takes over.
- ✓ In Real Mode, CPU is only capable of using approximately 1 MB of memory built into the motherboard.
- ✓ The BIOS will do a **Power-On Self Test** (POST) to make sure that all hardware are working.
- ✓ BIOS will then look for a small sector at the very beginning of your primary hard disk called **MBR**.
- ✓ The MBR contains a list, or map, of all of the **partitions** on your computer's hard disk (or disks).
- ✓ After the MBR is found the **Bootstrap Loader** follows basic instructions for starting up the rest of the computer, including the operating system.
- ✓ In Early Kernel Initialization stage, a smaller core of the Kernel is activated.
- ✓ This core includes the **device drivers** needed to use computer's **RAM chips**.

# BIOS

- BIOS firmware was stored in a ROM/EPROM (Erasable Programmable Read-Only Memory) chip known as **firmware** on the PC motherboard.
- BIOS can be accessed during the initial phases of the boot procedure by pressing del, F2 or F10.
- Finally, the firmware code cycles through all storage devices and looks for a **boot-loader**. (usually located in first sector of a disk which is 512 bytes)
- If the boot-loader is found, then the firmware hands over control of the computer to it.

# Functions of Operating System



# Process Management. 1

- **A process is a program in execution.**
- A process needs certain resources, including CPU time, memory, files, and I/O devices to accomplish its task.
- Simultaneous execution leads to multiple processes. Hence creation, execution and termination of a process are the most basic functionality of an OS
- If processes are **dependent**, than they may try to share same resources. thus task of **process synchronization** comes to the picture.
- If processes are **independent**, than a due care needs to be taken to avoid their overlapping in memory area.
- Based on priority, it is important to allow more important processes to execute first than others.

# Memory management. 2

- Memory is a large array of words or bytes, each with its own address.
- It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a **volatile** storage device. When the computer made turn off everything stored in RAM will be erased automatically.
- In addition to the physical RAM installed in your computer, most modern operating systems allow your computer to use a *virtual memory system*. *Virtual memory allows your computer to use part of a permanent storage device (such as a hard disk) as extra memory.*
- The operating system is responsible for the following activities in connections with memory management:
  - Keep track of which parts of memory are currently being used and by whom.
  - Decide which processes to load when memory space becomes available.
  - Allocate and de-allocate memory space as needed.

# File Management. 3

- A file is a collection of related information defined by its creator.
- *File systems provide the conventions for the encoding, storage and management of data on a storage device such as a hard disk.*
  - FAT12 (floppy disks)
  - FAT16 (DOS and older versions of Windows)
  - FAT32 (older versions of Windows)
  - NTFS (newer versions of Windows)
  - EXT3 (Unix/Linux)
  - HFS+ (Mac OS X)
- The operating system is responsible for the following activities in connections with file management:
  - ◆ File creation and deletion.
  - ◆ Directory creation and deletion.
  - ◆ Support of primitives for manipulating files and directories.
  - ◆ Mapping files onto secondary storage.
  - ◆ File backup on stable (nonvolatile) storage media.

# Device Management or I/O Management. 4

- *Device controllers* are components on the motherboard (or on expansion cards) that act as an interface between the CPU and the actual device.
- *Device drivers*, which are the operating system software components that interact with the devices controllers.
- A special device (inside CPU) called the **Interrupt Controller** handles the task of receiving interrupt requests and prioritizes them to be forwarded to the processor.
- **Deadlocks** can occur when two (or more) processes have control of different I/O resources that are needed by the other processes, and they are unwilling to give up control of the device.
- It performs the following activities for device management.
  - Keeps tracks of all devices connected to system.
  - Designates a program responsible for every device known as Input/output controller.
  - Decides which process gets access to a certain device and for how long.
  - Allocates devices in an effective and efficient way.
  - Deallocates devices when they are no longer required.

# Security & Protection. 5

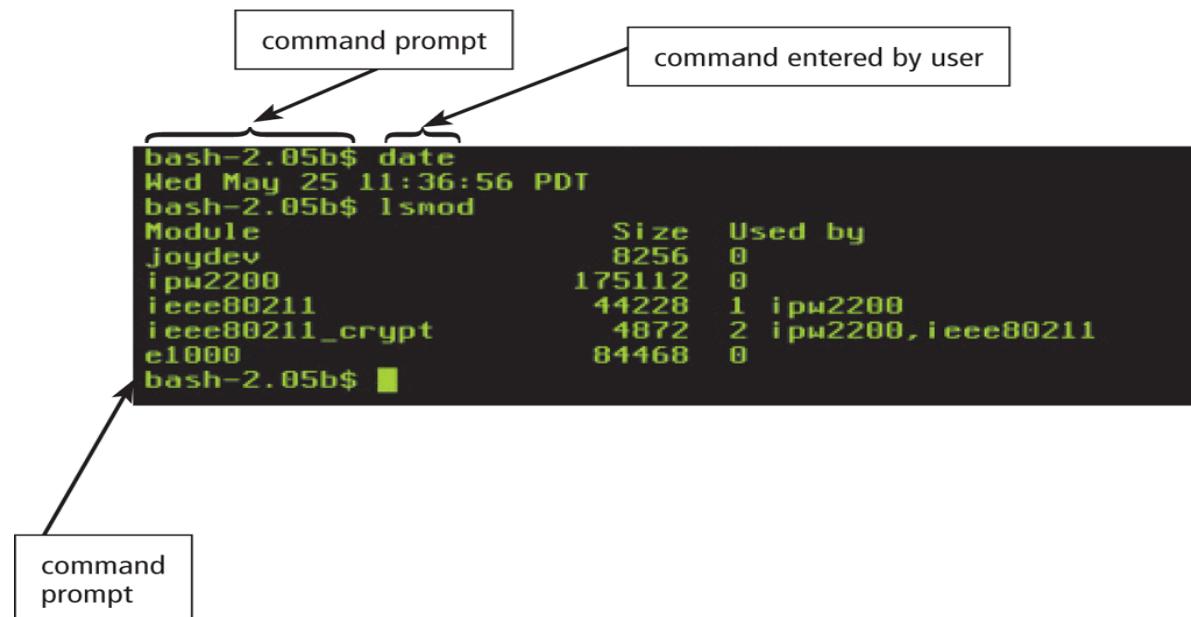
- The operating system uses password protection to protect user data and similar other techniques.
- It also prevents unauthorized access to programs and user data by assigning access right permission to files and directories.
- The owners of information stored in a multiuser or networked computer system may want to control use of that information, concurrent processes should not interfere with each other.

# User Interface Mechanism. 6

- A **user interface (UI)** controls how you enter data and instructions and how information is displayed on the screen
- There are two types of user interfaces
  1. Command Line Interface
  2. Graphical user Interface

# Command-line interface. 1

- In a command-line interface, a user types commands represented by short keywords or abbreviations or presses special keys on the keyboard to enter data and instructions



# Graphical User Interface. 2

- With a graphical user interface (GUI), you interact with menus and visual images



# History of Operating System

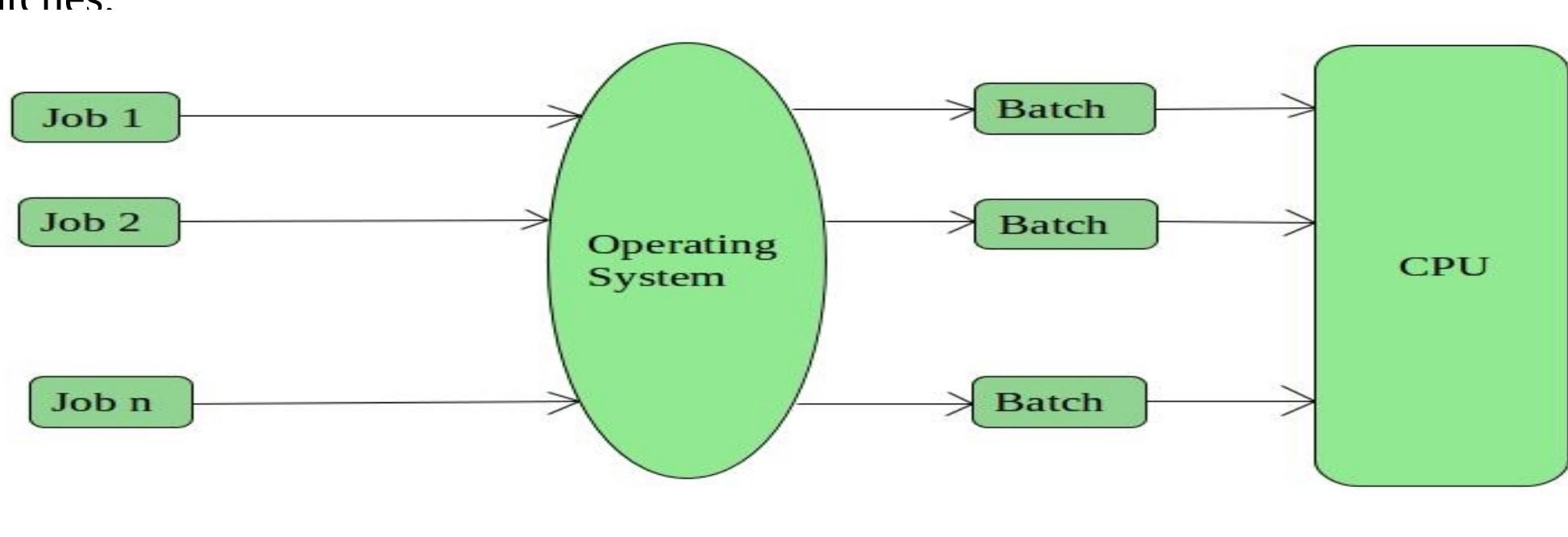
- ❖ **The First Generation (1940's to early 1950's)**
  - No Operating System
  - All programming was done in absolute machine language, often by wiring up plug-boards to control the machine's basic functions.
- ❖ **The Second Generation (1955-1965)**
  - First operating system was introduced in the early 1950's. It was called GMOS
  - Created by General Motors for IBM's machine the 701.
  - Single-stream batch processing systems
- ❖ **The Third Generation (1965-1980)**
  - Introduction of multiprogramming
  - Development of Minicomputer
- ❖ **The Fourth Generation (1980-Present Day)**
  - Development of PCs
  - Birth of Windows/MaC OS

# Types of Operating Systems

1. Batch Operating System
2. Multiprogramming Operating System
3. Time-Sharing OS
4. Multiprocessing OS
5. Distributed OS
6. Network OS
7. Real Time OS
8. Embedded OS

# Batch Operating System. 1

- The users of this type of operating system does not interact with the computer directly.
- Each user prepares his job on an off-line device like punch cards and submits it to the computer operator
- There is an operator which takes similar jobs having the same requirement and group them into batches.



# ...Batch Operating System cont. 1

## **Advantages of Batch Operating System:**

- Processors of the batch systems know how long the job would be when it is in queue
- Multiple users can share the batch systems
- The idle time for the batch system is very less
- It is easy to manage large work repeatedly in batch systems

## **Disadvantages of Batch Operating System:**

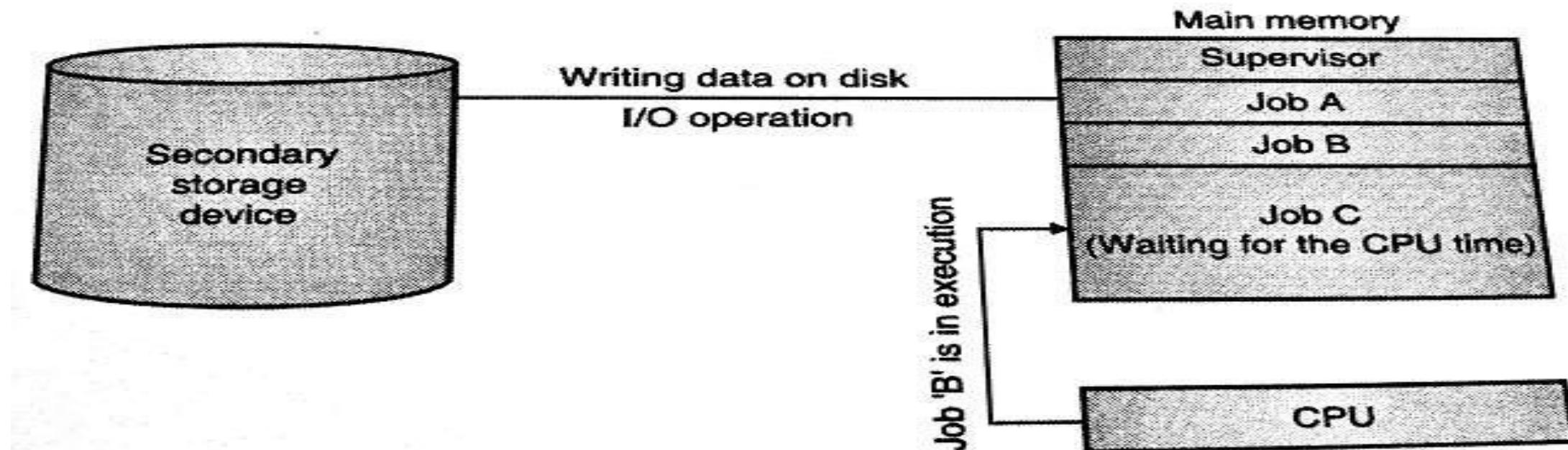
- The computer operators should be well known with batch systems
- Batch systems are hard to debug
- It is sometimes costly
- The other jobs will have to wait for an unknown time if any job fails

## **Examples of Batch based Operating System:**

IBM's MVS

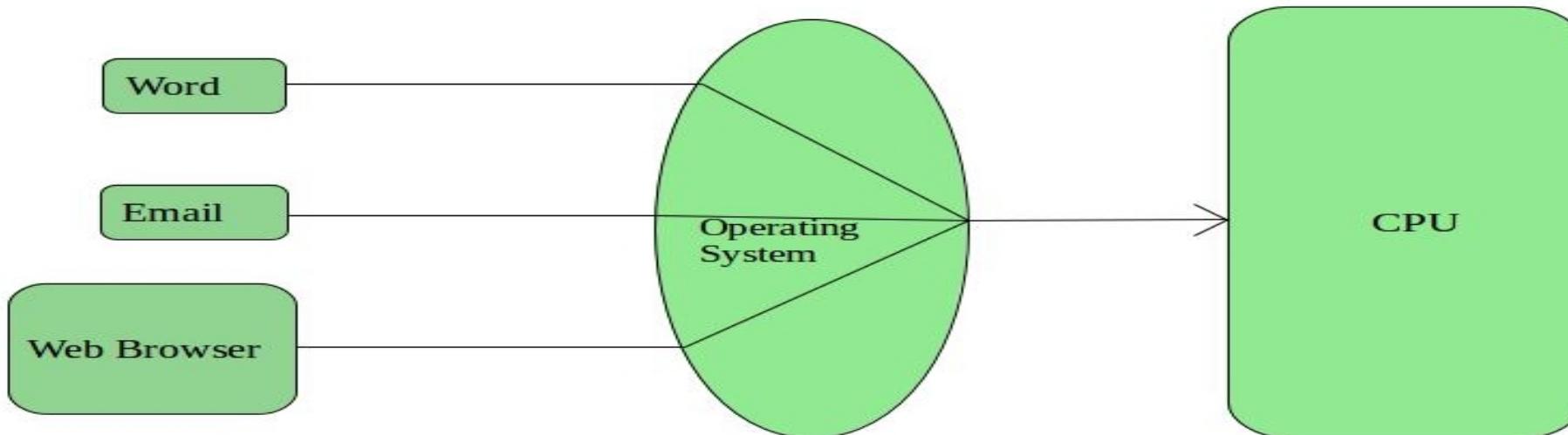
## :Multiprogramming Operating System. 2

- This type of OS is used to execute more than one jobs simultaneously by a single processor.
- It increases CPU utilization by organizing jobs so that the CPU always has one job to execute.
- Multiprogramming operating systems use the mechanism of job scheduling and CPU scheduling.



# Time-Sharing Operating Systems. 3

- Each task is given some time to execute so that all the tasks work smoothly.
- These systems are also known as **Multi-tasking Systems**.
- The task can be from a single user or different users also.
- The time that each task gets to execute is called quantum.
- After this time interval is over OS switches over to the next task.



## ..Time-Sharing Operating Systems cont. 3

- **Advantages of Time-Sharing OS:**
  - Each task gets an equal opportunity
  - Fewer chances of duplication of software
  - CPU idle time can be reduced
- **Disadvantages of Time-Sharing OS:**
  - Reliability problem
  - One must have to take care of the security and integrity of user programs and data
  - Data communication problem
- **Examples of Time-Sharing Oss**

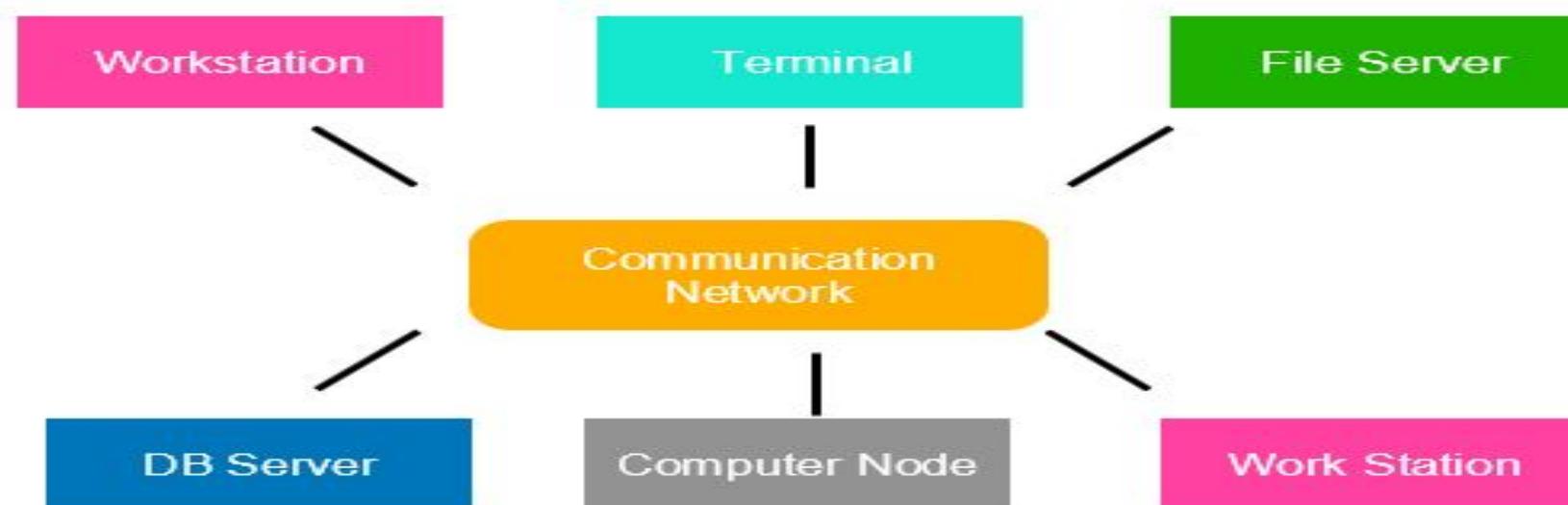
Multics, Unix, etc.

# Multiprocessor operating systems. 4

- *Multiprocessor operating systems are also known as parallel OS or tightly coupled OS.*
- *Such operating systems have more than one processor in close communication that sharing the computer bus, the clock and sometimes memory and peripheral devices.*
- *It executes multiple jobs at the same time and makes the processing faster.*
- *It supports large physical address space and larger virtual address space.*
- *If one processor fails then other processor should retrieve the interrupted process state so execution of process can continue.*
- *Inter-processes communication mechanism is provided and implemented in hardware.*

# Distributed Operating System. 5

- Various autonomous interconnected computers communicate with each other using a shared communication network.
- Independent systems possess their own memory unit and CPU.
- These are referred to as **loosely coupled systems**.
- Examples:- Locus, DYSEAC



# Network Operating System. 6

- These systems run on a server and provide the capability to manage data, users, groups, security, applications, and other networking functions.
- These types of operating systems allow shared access of files, printers, security, applications, and other networking functions over a small private network.
- The “other” computers are called client computers, and each computer that connects to a network server must be running client software designed to request a specific service.
- popularly known as **tightly coupled systems**.

# Network Operating System. 6

## **Advantages of Network Operating System:**

- Highly stable centralized servers
- Security concerns are handled through servers
- New technologies and hardware up-gradation are easily integrated into the system
- Server access is possible remotely from different locations and types of systems

## **Disadvantages of Network Operating System:**

- Servers are costly
- User has to depend on a central location for most operations
- Maintenance and updates are required regularly

## **Examples of Network Operating System are:**

Microsoft Windows Server 2003/2008/2012, UNIX, Linux, Mac OS X, Novell NetWare, and BSD, etc.

# Real-Time Operating System. 7

- These types of OSs serve real-time systems.
- The time interval required to process and respond to inputs is very small.
- This time interval is called **response time**.
- **Real-time systems** are used when there are time requirements that are very strict like
  - missile systems,
  - air traffic control systems,
  - robots, etc.

# Embaded Operating System. 8

- An embedded operating system is one that is built into the circuitry of an electronic device.
- Embedded operating systems are now found in automobiles, bar-code scanners, cell phones, medical equipment, and personal digital assistants.
- The most popular embedded operating systems for consumer products, such as PDAs, include the following:
  - Windows XP Embedded
  - Windows CE .NET:- it supports wireless communications, multimedia and Web browsing. It also allows for the use of smaller versions of Microsoft Word, Excel, and Outlook.
  - Palm OS:- It is the standard operating system for Palm-brand PDAs as well as other proprietary handheld devices.
  - Symbian:- OS found in “smart” cell phones from Nokia and Sony Ericsson

# Popular types of OS

- Desktop Class
  - ❖ Windows
  - ❖ OS X
  - ❖ Unix/Linux
  - ❖ Chrome OS
- Server Class
  - ❖ Windows Server
  - ❖ Mac OS X Server
  - ❖ Unix/Linux
- Mobile Class
  - ❖ Android
  - ❖ iOS
  - ❖ Windows Phone

# **:-Desktop Class Operating Systems**

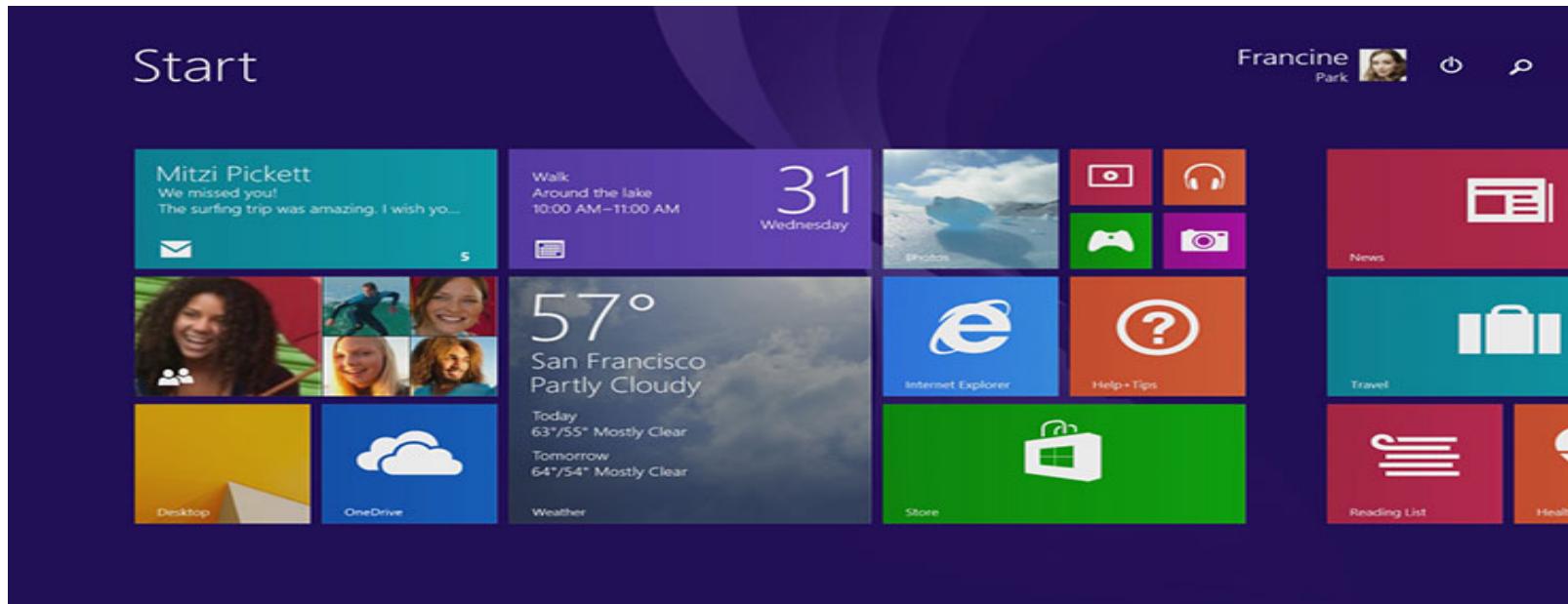
- **Platform:** the hardware required to run a particular operating system
  - Intel platform (IBM-compatible)
    - Windows
    - DOS
    - UNIX
    - Linux
  - Macintosh platform
    - Mac OS
  - iPad and iPhone platform
    - iOS

# Ms-DOS

- Single User Single Tasking OS.
- It had no built-in support for networking, and users had to manually install drivers any time they added a new hardware component to their PC.
- DOS supports only 16-bit programs.
- Command line user interface.
- So, why is DOS still in use? Two reasons are its size and simplicity. It does not require much memory or storage space for the system, and it does not require a powerful computer.



- The graphical Microsoft operating system designed for Intel-platform desktop and notebook computers.
- Best known, greatest selection of applications available.
- Current editions include Windows 7, 8, 8.1 and 10.

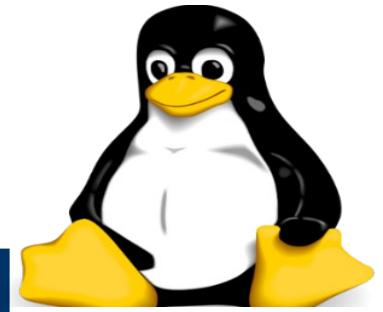


# Mac OS

- User-friendly, runs on Mac hardware. Many applications available.
- Current editions include: Sierra, High Sierra, Mojave, Catalina & Big Sur—Version XI(Released in Nov 2020)

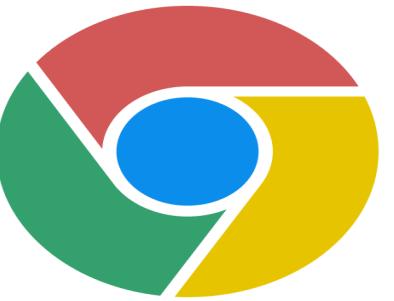


# Linux



- **Linux:** An open-source, cross-platform operating system that runs on desktops, notebooks, tablets, and smartphones.
  - The name *Linux* is a combination *Linus* (the first name of the first developer) and *UNIX* (*another operating system*).
- Users are free to modify the code, improve it, and redistribute it,
- Developers are not allowed to charge money for the Linux kernel itself (the main part of the operating system), but they can charge money for **distributions (distros** for short).

# Google Chrome OS



- **Chrome OS.** Is a popular thin client operating system.
- **Thin client** A computer with minimal hardware, designed for a specific task.  
For example, a thin web client is designed for using the Internet.

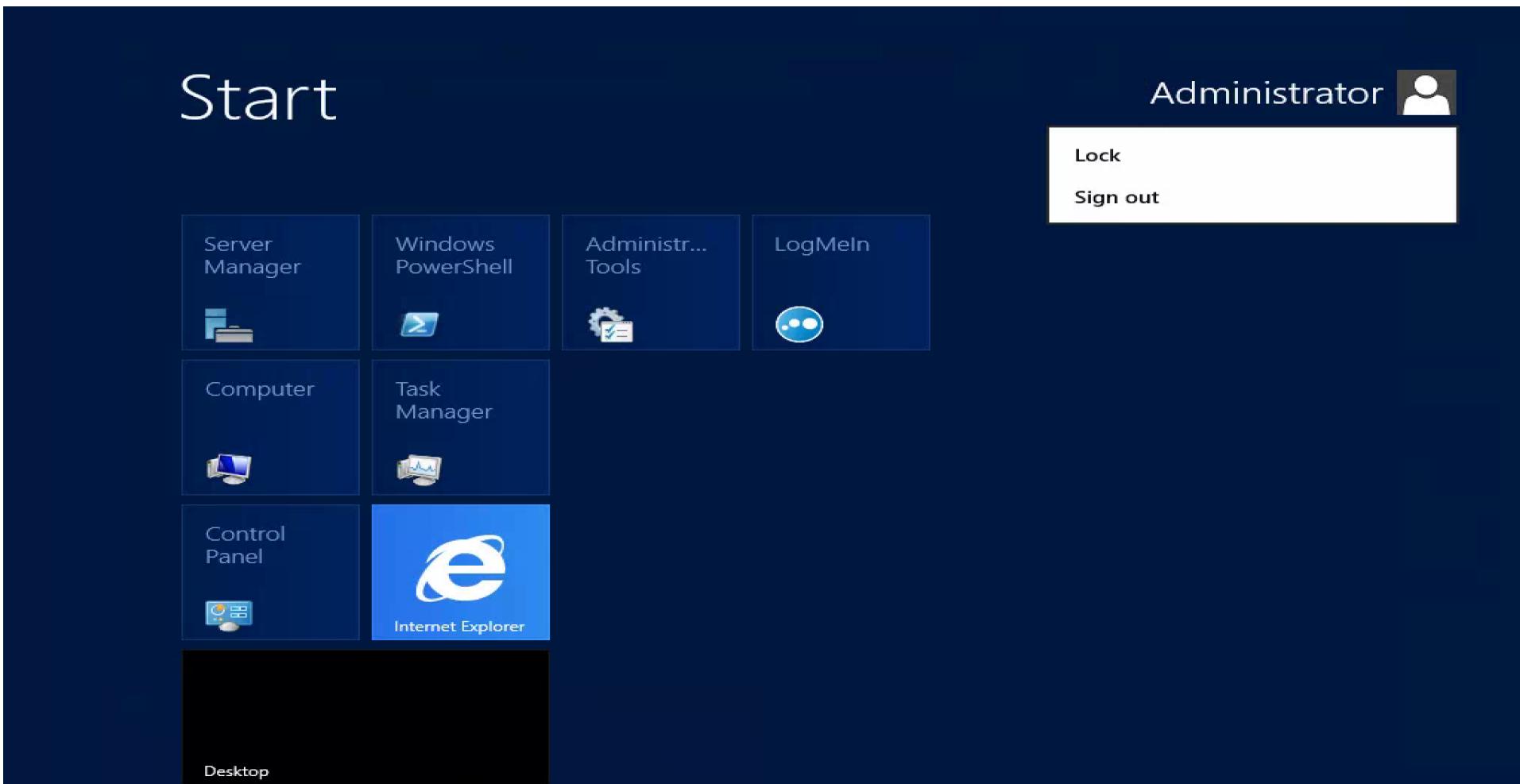


chromebbook

# Server Operating Systems

- Windows Server
  - Familiar GUI interface for those experienced with Windows
- UNIX
  - Very mature server capabilities, time-tested, large user community, stable
- Linux
  - Free, customizable, many free services and utilities available

# Windows Server



# UNIX

```
mars@marsmain /usr/portage/app-shells/bash $ sudo /etc/init.d/bluetooth status
Password:
* status: started
mars@marsmain /usr/portage/app-shells/bash $ ping -q -c1 en.wikipedia.org
PING rr.esams.wikimedia.org (91.198.174.2) 56(84) bytes of data.

--- rr.esams.wikimedia.org ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 2ms
rtt min/avg/max/mdev = 49.820/49.820/49.820/0.000 ms
mars@marsmain /usr/portage/app-shells/bash $ grep -i /dev/sda /etc/fstab | cut --fields=-3
/dev/sda1          /boot
/dev/sda2          none
/dev/sda3          /
mars@marsmain /usr/portage/app-shells/bash $ date
Sat Aug  8 02:42:24 MSD 2009
mars@marsmain /usr/portage/app-shells/bash $ lsmod
Module           Size  Used by
rndis_wlan        23424  0
rndis_host         8696  1 rndis_wlan
cdc_ether          5672  1 rndis_host
usbnet            18688  3 rndis_wlan,rndis_host,cdc_ether
parport_pc        38424  0
fglrx            2388128  20
parport           39648  1 parport_pc
iTCO_wdt          12272  0
i2c_i801           9380  0
mars@marsmain /usr/portage/app-shells/bash $
```

# Tablet and Phone Operating Systems

- **System-on-chip (SoC):** An operating system that comes preinstalled on a chip on a portable device such as a smartphone.
- Popular SoC operating systems:
  - iOS: for iPad, iPhone
  - Android: for a variety of tablets and phones
- Downloadable applications (apps) from an App store, for example:
  - Apple App Store
  - Google Play Store



# iOS on the iPhone and iPad

- The Apple-created operating system for Apple tablets and phones.
- The current stable version, iOS 14, was released to the public on September 16, 2020.



# Android



- Android, a popular OS for smartphones and tablets, is based on Linux Kernel.
  - Developed by Google
- Current versions include:
  - Android 8 Oreo
  - Android 9 Pie
  - Android 10
  - Android 11 (released on Sep, 2020)



# Advantage of Linux Operating System

## 1. Open Source

As it is open-source, its source code is easily available.

Anyone having programming knowledge can customize the operating system.

One can contribute, modify, distribute, and enhance the code for any purpose.

## 2. Security

The Linux security feature is the main reason that it is the most favourable option for developers.

It is not completely safe, but it is less vulnerable than others.

Each application needs to authorize by the admin user.

Linux systems do not require any antivirus program.

## 3. Free

Certainly, the biggest advantage of the Linux system is that it is free to use.

We can easily download it, and there is no need to buy the license for it.

It is distributed under GPL (General Public License).

Comparatively, we have to pay a huge amount for the license of the other OS

# Advantage of Linux Operating System

## 4. Lightweight

The requirements for running Linux are much less than other operating system  
In Linux, the memory footprint and disk space are also lower.

Generally, most of the Linux distributions required as little as 128MB of RAM  
around the same amount for disk space.

## 5. Stability

Linux is more stable than other operating systems.

Linux does not require to reboot the system to maintain performance levels.  
It rarely hangs up or slow down. It has big up-times.

# Advantage of Linux Operating System

## 6. Performance

Linux system provides high performance over different networks.

It is capable of handling a large number of users simultaneously.

## 7. Flexibility

Linux operating system is very flexible.

It can be used for desktop applications, embedded systems, and server applications too.

It also provides various restriction options for specific computers.

We can install only necessary components for a system.

## 8. Software Updates

In Linux, the software updates are in user control.

We can select the required updates.

There a large number of system updates are available.

These updates are much faster than other operating systems.

So, the system updates can be installed easily without facing any issue.

# Advantage of Linux Operating System

## 9. Distributions/ Distros

There are many Linux distributions available in the market.

It provides various options and flavors of Linux to the users.

We can choose any distros according to our needs.

Some popular distros are **Ubuntu**, **Fedora**, **Debian**, **Linux Mint**, **Arch Linux**,

For the beginners, Ubuntu and Linux Mint would be useful.

Debian and Fedora would be good choices for proficient programmers.

## 10. Live CD/USB

Almost all Linux distributions have a **Live CD/USB** option.

It allows us to try or run the Linux operating system without installing it.

## 11. Graphical User Interface

Linux is a command-line based OS but it provides an interactive user interface like Windows.

# Advantage of Linux Operating System

## 12. Suitable for programmers

It supports almost all of the most used programming languages such as [C/C++](#), Java, Python, Ruby, and more.

Further, it offers a vast range of useful applications for development.

The programmers prefer the Linux terminal over the Windows command line.

The package manager on Linux system helps programmers to understand how things are done.

Bash scripting is also a functional feature for the programmers.

It also provides support for SSH, which helps in managing the servers quickly.

## 13. Community Support

Linux provides large community support.

We can find support from various sources.

There are many forums available on the web to assist users.

Further, developers from the various open source communities are ready to help us.

# Advantage of Linux Operating System

## 14. Privacy

Linux always takes care of user privacy as it never takes much private data from the user. Comparatively, other operating systems ask for the user's private data.

## 15. Networking

Linux facilitates with powerful support for networking. The client-server systems can be easily set to a Linux system. It provides various command-line tools such as ssh, ip, mail, telnet, and more for connectivity with the other systems and servers. Tasks such as network backup are much faster than others.

## 16. Compatibility

Linux is compatible with a large number of file formats as it supports almost all file formats.

## 17. Installation

Linux installation process takes less time than other operating systems such as Windows. Further, its installation process is much easy as it requires less user input. It does not require much more system configuration even it can be easily installed on old machines having less configuration.

# Advantage of Linux Operating System

## 18. Multiple Desktop Support

Linux system provides multiple desktop environment support for its enhanced use. The desktop environment option can be selected during installation. We can select any desktop environment such as **GNOME (GNU Network Object Model Environment)** or **KDE (K Desktop Environment)** as both have their specific environment.

## 19. Multitasking

It is a multitasking operating system as it can run multiple tasks simultaneously without affecting the system speed.

## 20. Heavily Documented for beginners

There are many command-line options that provide documentation on commands, libraries, standards such as manual pages and info pages. Also, there are plenty of documents available on the internet in different formats, such as Linux tutorials, Linux documentation project, Serverfault, and more. To help the beginners, several communities are available such as **Ask Ubuntu**, **Reddit**, and **StackOverflow**.

thank you!



# **Unit 3**

# **CPU Scheduling**

# CPU Scheduling

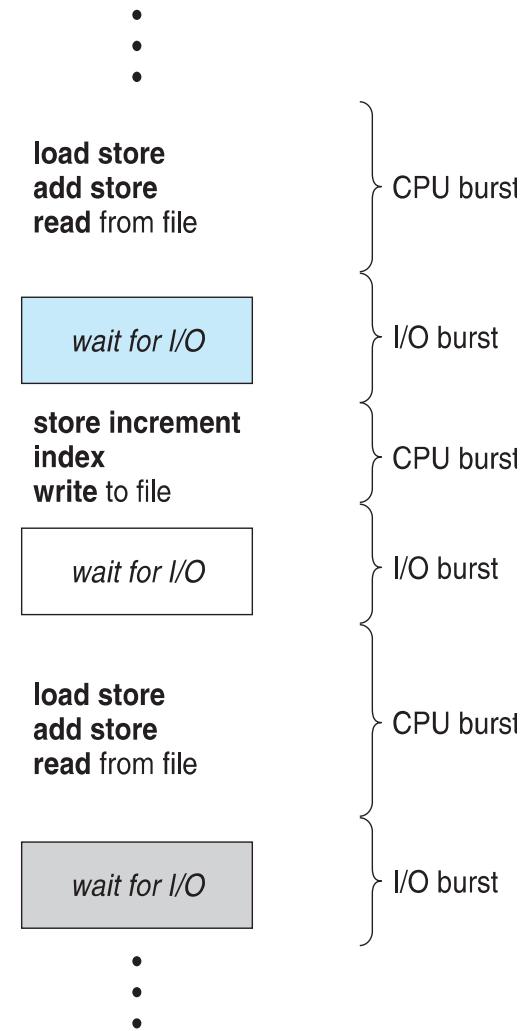
- Basic Concepts
- Scheduling Criteria
- Scheduling Algorithms

# Objectives

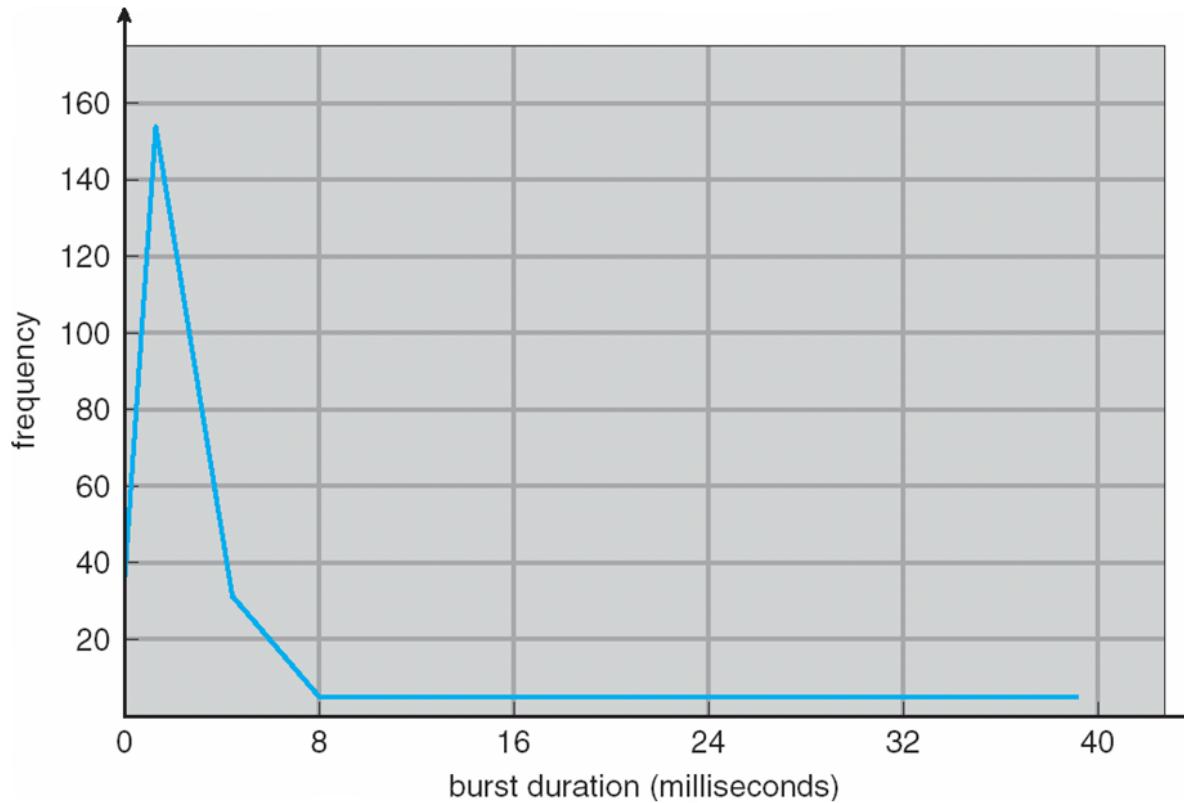
- To introduce CPU scheduling, which is the basis for multiprogrammed operating systems
- To describe various CPU-scheduling algorithms
- To discuss evaluation criteria for selecting a CPU-scheduling algorithm for a particular system
- To examine the scheduling algorithms of several operating systems

# Basic Concepts

- Maximum CPU utilization obtained with multiprogramming
- CPU-I/O Burst Cycle – Process execution consists of a **cycle** of CPU execution and I/O wait
- **CPU burst** followed by **I/O burst**
- CPU burst distribution is of main concern



# Histogram of CPU-burst Times



# CPU Scheduler

- Short-term scheduler selects from among the processes in ready queue, and allocates the CPU to one of them
  - Queue may be ordered in various ways
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state
  2. Switches from running to ready state
  3. Switches from waiting to ready
  4. Terminates
- Scheduling under 1 and 4 is nonpreemptive
- All other scheduling is preemptive
  - Consider access to shared data
  - Consider preemption while in kernel mode
  - Consider interrupts occurring during crucial OS activities

# Dispatcher

- Dispatcher module gives control of the CPU to the process selected by the short-term scheduler; this involves:
  - switching context
  - switching to user mode
  - jumping to the proper location in the user program to restart that program
- **Dispatch latency** – time it takes for the dispatcher to stop one process and start another running

# Scheduling Criteria

- **CPU utilization** – keep the CPU as busy as possible
- **Throughput** – # of processes that complete their execution per time unit
- **Turnaround time** – amount of time to execute a particular process
- **Waiting time** – amount of time a process has been waiting in the ready queue
- **Response time** – amount of time it takes from when a request was submitted until the first response is produced, not output (for time-sharing environment)

# **Scheduling Algorithm Optimization Criteria**

- Max CPU utilization
- Max throughput
- Min turnaround time
- Min waiting time
- Min response time

# First- Come, First-Served (FCFS) Scheduling

<u>Process</u>	<u>Burst Time</u>
$P_1$	24
$P_2$	3
$P_3$	3

- Suppose that the processes arrive in the order:  $P_1, P_2, P_3$   
The Gantt Chart for the schedule is:



- Waiting time for  $P_1 = 0$ ;  $P_2 = 24$ ;  $P_3 = 27$
- Average waiting time:  $(0 + 24 + 27)/3 = 17$

# FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order:

$$P_2, P_3, P_1$$

■ The Gantt chart for the schedule is:



- Waiting time for  $P_1 = 6$ ;  $P_2 = 0$ ;  $P_3 = 3$
- Average waiting time:  $(6 + 0 + 3)/3 = 3$
- Much better than previous case
- **Convoy effect** - short process behind long process
  - Consider one CPU-bound and many I/O-bound processes

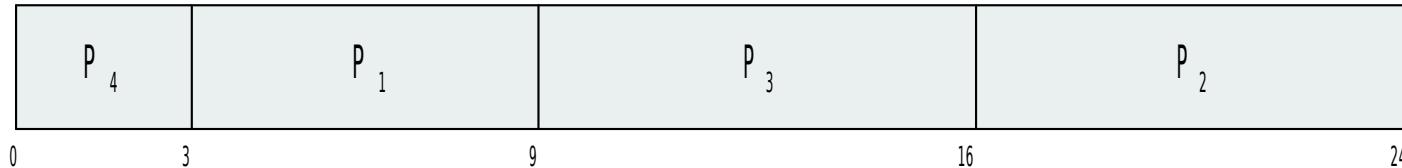
# **Shortest-Job-First (SJF) Scheduling**

- Associate with each process the length of its next CPU burst
  - Use these lengths to schedule the process with the shortest time
- SJF is optimal – gives minimum average waiting time for a given set of processes
  - The difficulty is knowing the length of the next CPU request
  - Could ask the user

# Example of SJF

<u>Process</u>	<u>Burst Time</u>
$P_1$	6
$P_2$	8
$P_3$	7
$P_4$	3

- SJF scheduling chart

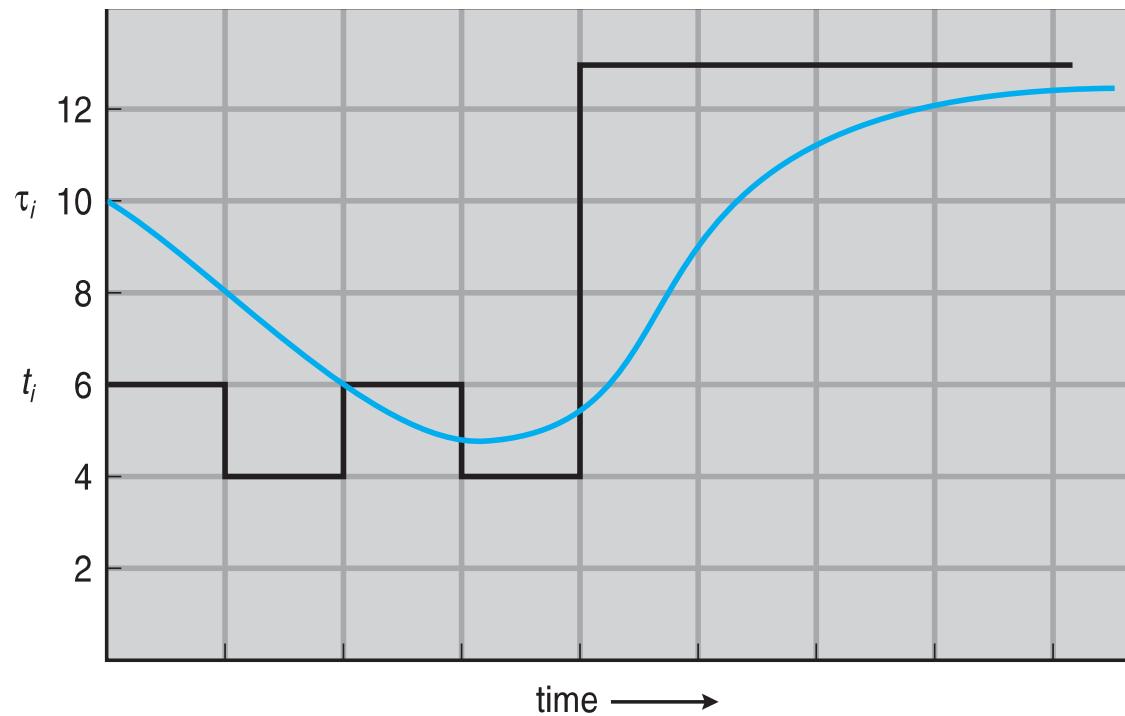


- Average waiting time =  $(3 + 16 + 9 + 0) / 4 = 7$

# Determining Length of Next CPU Burst

- Can only estimate the length – should be similar to the previous one
  - Then pick process with shortest predicted next CPU burst
- Can be done by using the length of previous CPU bursts, using exponential averaging
  1.  $t_n$  =actual length of  $n^{th}$  CPU burst
  2.  $\tau_{n+1}$  =predicted value for the next CPU burst
  3.  $0 \leq \alpha \leq 1$
  4. Define :  $\tau_{n+1} = \alpha t_n + (1 - \alpha)\tau_n$ .
- Commonly,  $\alpha$  set to  $\frac{1}{2}$
- Preemptive version called **shortest-remaining-time-first**

# Prediction of the Length of the Next CPU Burst



CPU burst ( $t_i$ )

6    4    6    4    13    13    13

...

"guess" ( $\tau_i$ )

10    8    6    6    5    9    11    12    ...

# Examples of Exponential Averaging

- $\alpha = 0$ 
  - $\tau_{n+1} = \tau_n$
  - Recent history does not count
- $\alpha = 1$ 
  - $\tau_{n+1} = \alpha t_n$
  - Only the actual last CPU burst counts
- If we expand the formula, we get:

$$\begin{aligned}\tau_{n+1} &= \alpha t_n + (1 - \alpha)\alpha t_{n-1} + \dots \\ &\quad + (1 - \alpha)^j \alpha t_{n-j} + \dots \\ &\quad + (1 - \alpha)^{n+1} \tau_0\end{aligned}$$

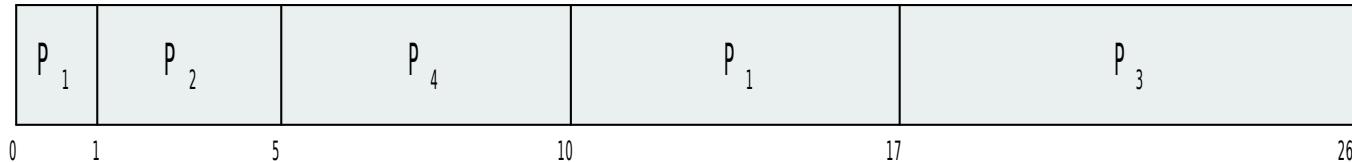
- Since both  $\alpha$  and  $(1 - \alpha)$  are less than or equal to 1, each successive term has less weight than its predecessor

# Example of Shortest-remaining-time-first

- Now we add the concepts of varying arrival times and preemption to the analysis

<u>Process</u>	<u>Arrival Time</u>	<u>Burst Time</u>
$P_1$	0	8
$P_2$	1	4
$P_3$	2	9
$P_4$	3	5

- Preemptive SJF Gantt Chart



- Average waiting time =  $[(10-1)+(1-1)+(17-2)+5-3]/4 = 26/4 = 6.5$  msec

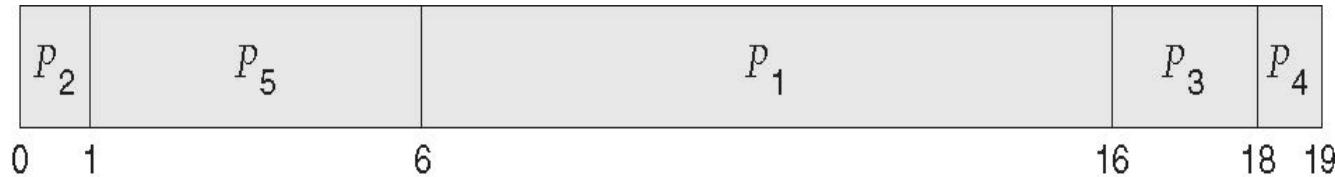
# Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer  $\equiv$  highest priority)
  - Preemptive
  - Nonpreemptive
- SJF is priority scheduling where priority is the inverse of predicted next CPU burst time
- Problem  $\equiv$  **Starvation** – low priority processes may never execute
- Solution  $\equiv$  **Aging** – as time progresses increase the priority of the process

# Example of Priority Scheduling

<u>Process</u>	<u>Burst Time</u>	<u>Priority</u>
$P_1$	10	3
$P_2$	1	1
$P_3$	2	4
$P_4$	1	5
$P_5$	5	2

- Priority scheduling Gantt Chart



- Average waiting time = 8.2 msec

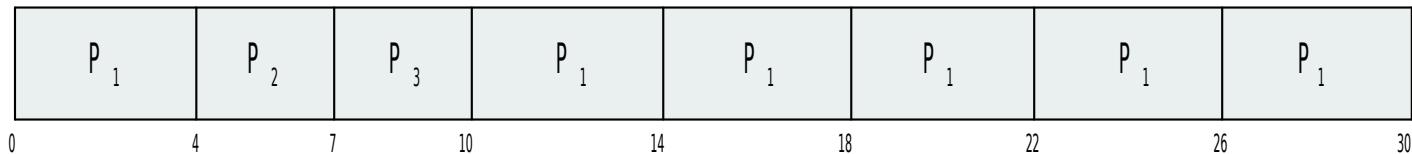
# Round Robin (RR)

- Each process gets a small unit of CPU time (**time quantum  $q$** ), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are  $n$  processes in the ready queue and the time quantum is  $q$ , then each process gets  $1/n$  of the CPU time in chunks of at most  $q$  time units at once. No process waits more than  $(n-1)q$  time units.
- Timer interrupts every quantum to schedule next process
- Performance
  - $q$  large  $\Rightarrow$  FIFO
  - $q$  small  $\Rightarrow q$  must be large with respect to context switch, otherwise overhead is too high

# Example of RR with Time Quantum = 4

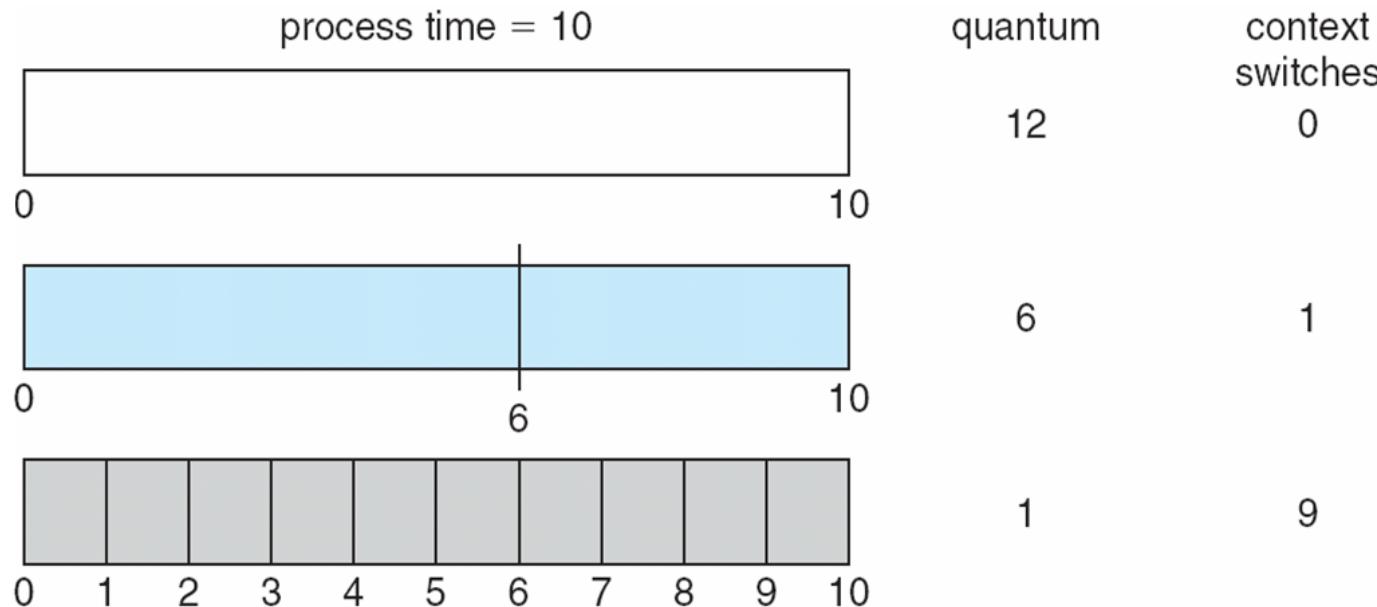
<u>Process</u>	<u>Burst Time</u>
$P_1$	24
$P_2$	3
$P_3$	3

- The Gantt chart is:

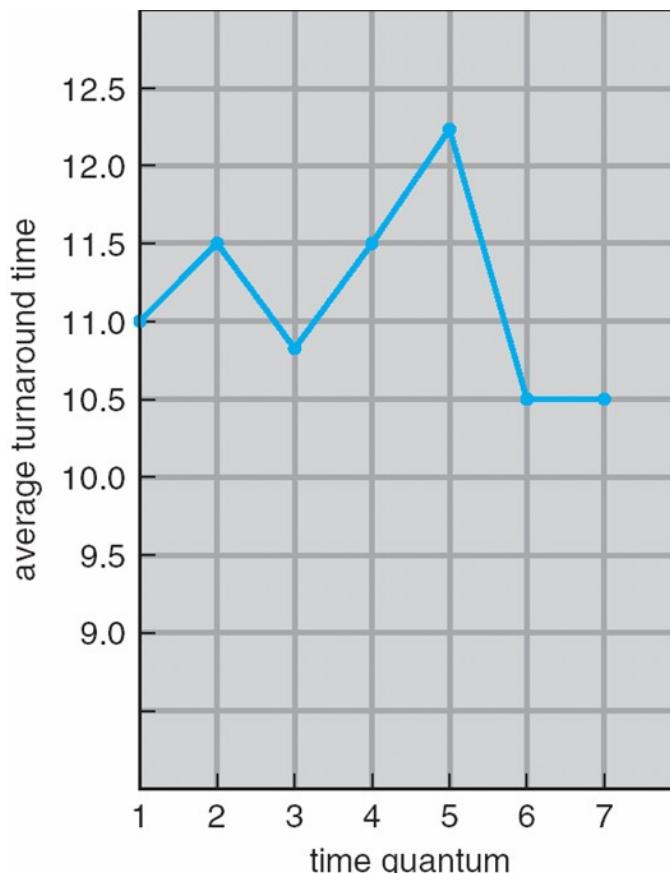


- Typically, higher average turnaround than SJF, but better **response**
- $q$  should be large compared to context switch time
- $q$  usually 10ms to 100ms, context switch < 10 usec

# Time Quantum and Context Switch Time



# Turnaround Time Varies With The Time Quantum



process	time
$P_1$	6
$P_2$	3
$P_3$	1
$P_4$	7

80% of CPU bursts should  
be shorter than  $q$

# **Unit 3**

# **Processes**

# Processes

- Process Concept
- Process Scheduling
- Operations on Processes
- Interprocess Communication
- Examples of IPC Systems
- Communication in Client-Server Systems

# Objectives

- To introduce the notion of a process -- a program in execution, which forms the basis of all computation
- To describe the various features of processes, including scheduling, creation and termination, and communication
- To explore interprocess communication using shared memory and message passing
- To describe communication in client-server systems

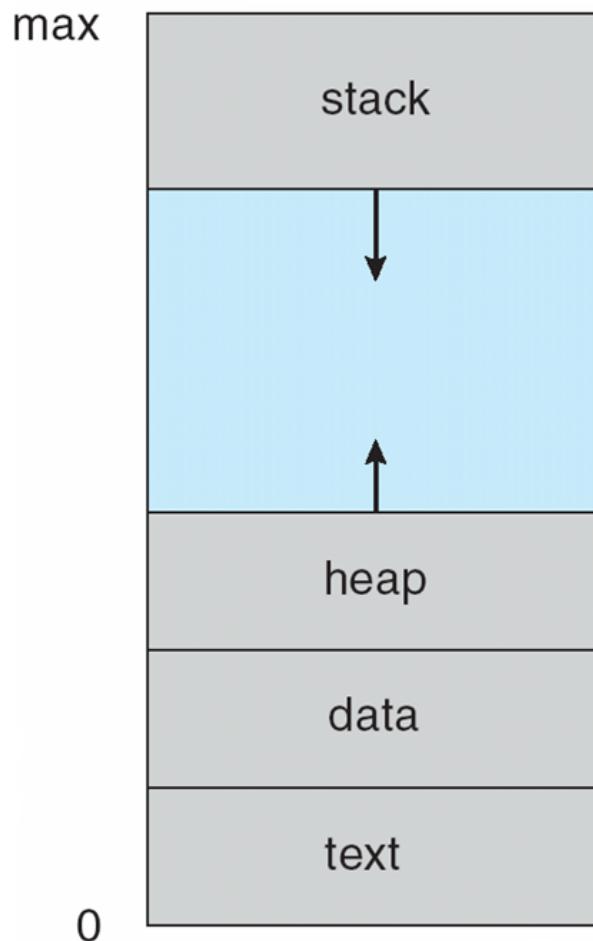
# Process Concept

- An operating system executes a variety of programs:
  - Batch system – **jobs**
  - Time-shared systems – **user programs** or **tasks**
- Textbook uses the terms **job** and **process** almost interchangeably
- **Process** – a program in execution; process execution must progress in sequential fashion
- Multiple parts
  - The program code, also called **text section**
  - Current activity including **program counter**, processor registers
  - **Stack** containing temporary data
    - Function parameters, return addresses, local variables
  - **Data section** containing global variables
  - **Heap** containing memory dynamically allocated during run time

# Process Concept (Cont.)

- Program is *passive* entity stored on disk (**executable file**), process is *active*
  - Program becomes process when executable file loaded into memory
- Execution of program started via GUI mouse clicks, command line entry of its name, etc
- One program can be several processes
  - Consider multiple users executing the same program

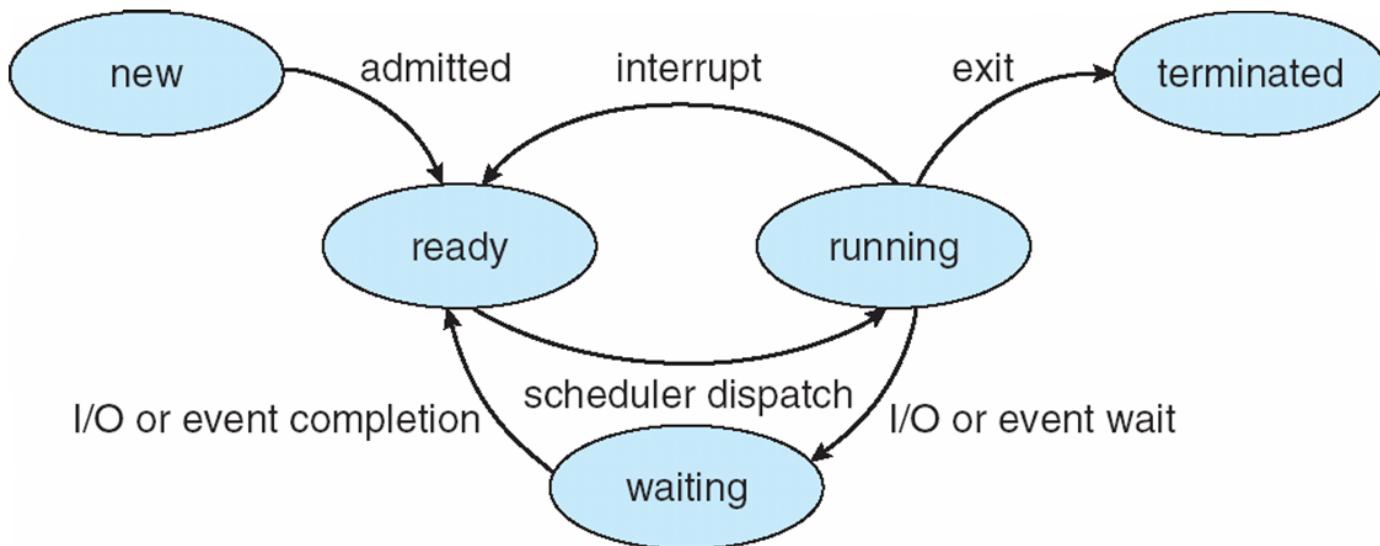
# Process in Memory



# Process State

- As a process executes, it changes **state**
  - **new**: The process is being created
  - **running**: Instructions are being executed
  - **waiting**: The process is waiting for some event to occur
  - **ready**: The process is waiting to be assigned to a processor
  - **terminated**: The process has finished execution

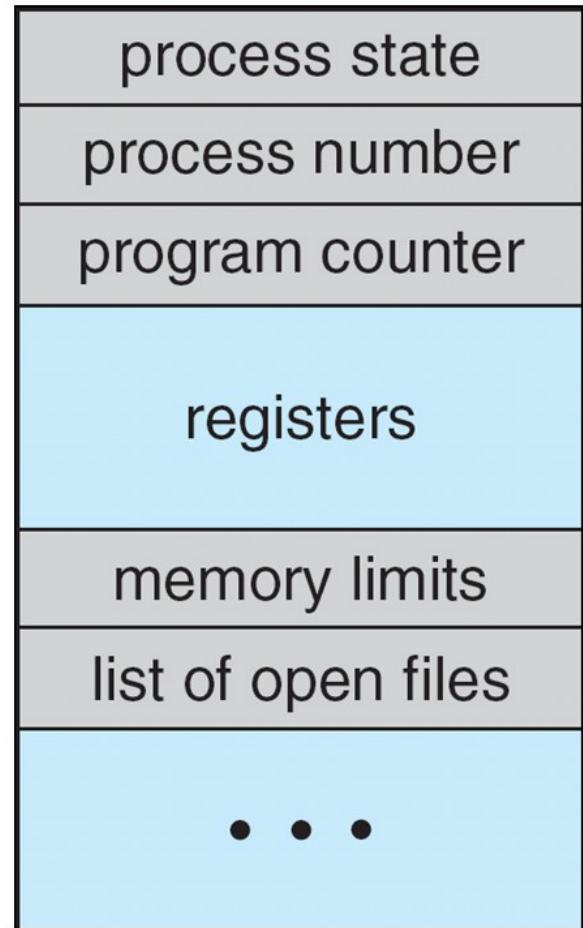
# Diagram of Process State



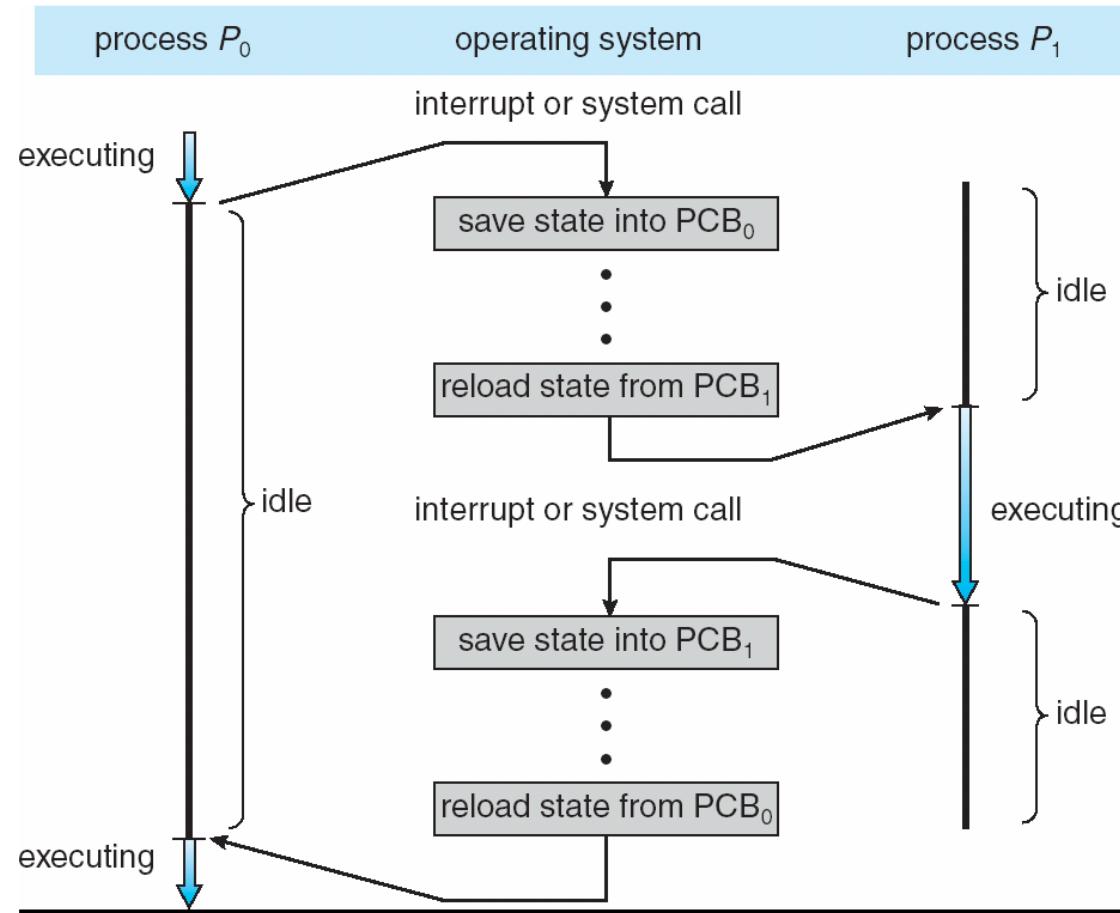
# Process Control Block (PCB)

Information associated with each process  
(also called **task control block**)

- Process state – running, waiting, etc
- Program counter – location of instruction to next execute
- CPU registers – contents of all process-centric registers
- CPU scheduling information- priorities, scheduling queue pointers
- Memory-management information – memory allocated to the process
- Accounting information – CPU used, clock time elapsed since start, time limits
- I/O status information – I/O devices allocated to process, list of open files



# CPU Switch From Process to Process



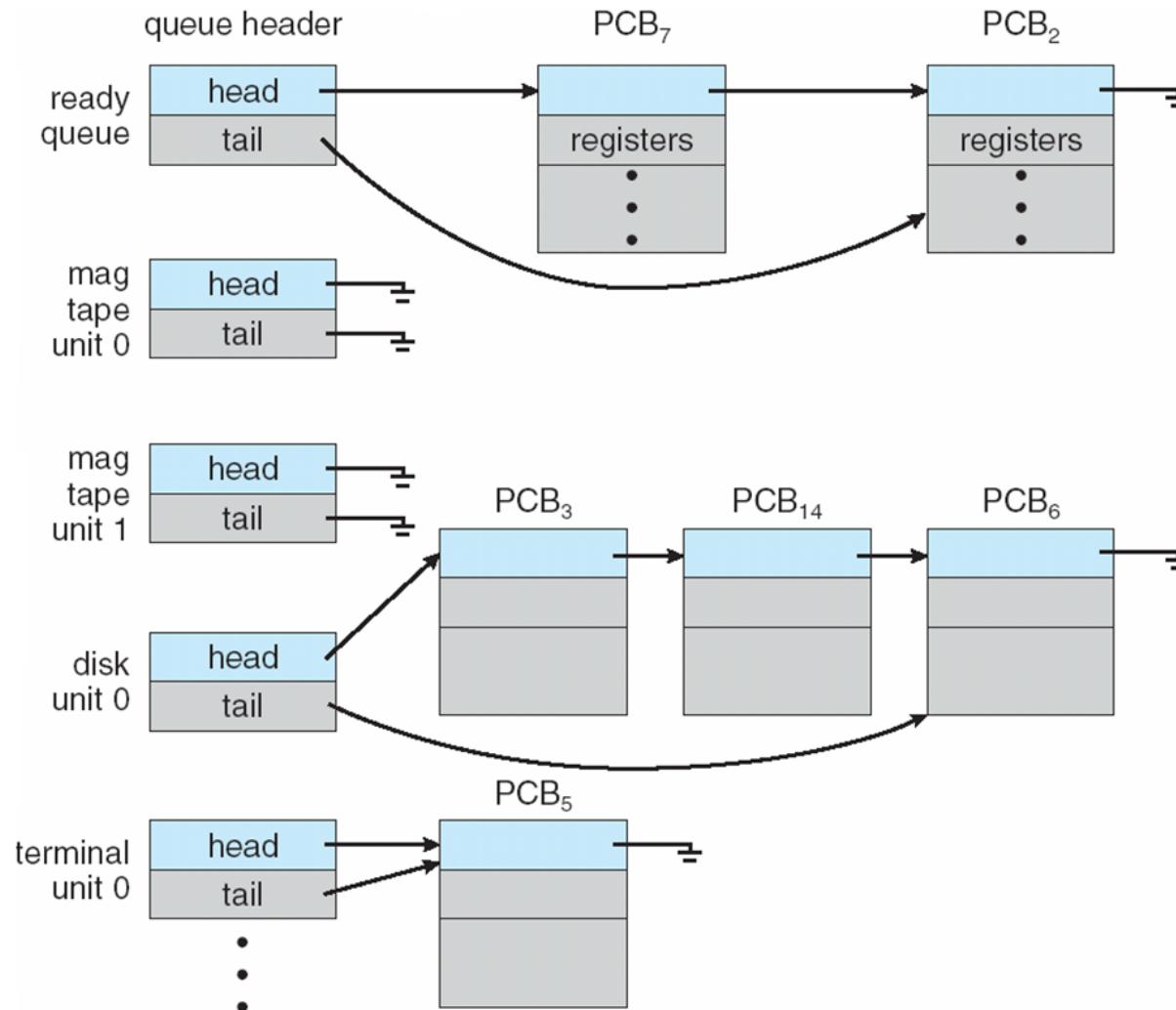
# Threads

- So far, process has a single thread of execution
- Consider having multiple program counters per process
  - Multiple locations can execute at once
    - Multiple threads of control -> **threads**
- Must then have storage for thread details, multiple program counters in PCB
- See next chapter

# Process Scheduling

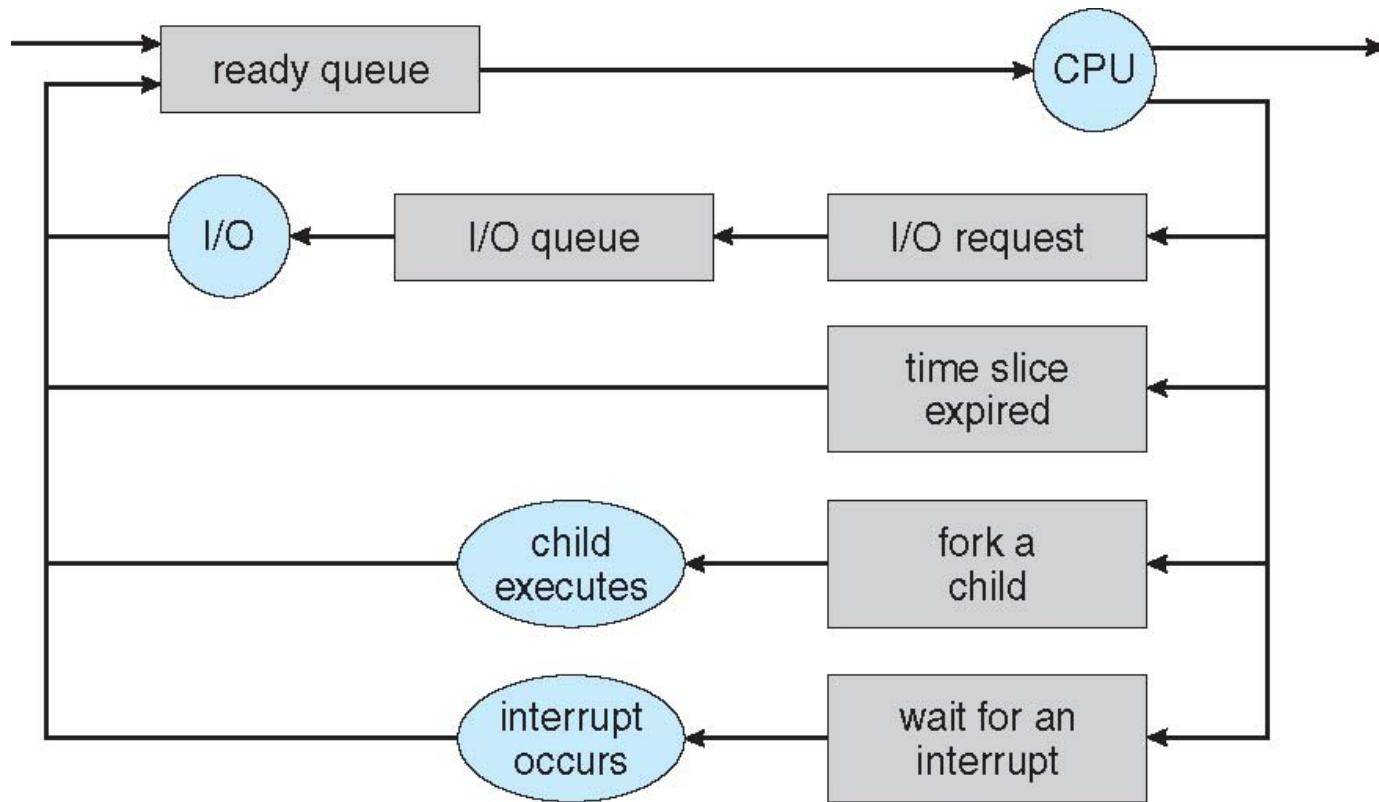
- Maximize CPU use, quickly switch processes onto CPU for time sharing
- **Process scheduler** selects among available processes for next execution on CPU
- Maintains **scheduling queues** of processes
  - **Job queue** – set of all processes in the system
  - **Ready queue** – set of all processes residing in main memory, ready and waiting to execute
  - **Device queues** – set of processes waiting for an I/O device
  - Processes migrate among the various queues

# Ready Queue And Various I/O Device Queues



# Representation of Process Scheduling

- **Queueing diagram** represents queues, resources, flows

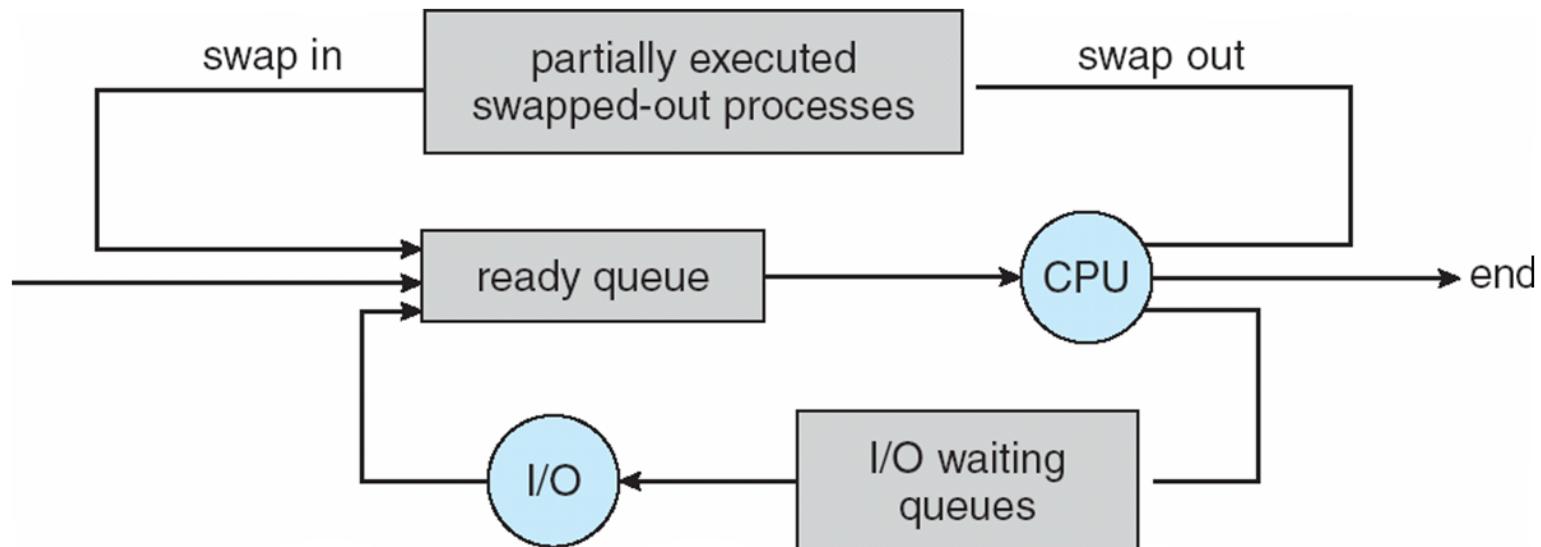


# Schedulers

- **Short-term scheduler** (or **CPU scheduler**) – selects which process should be executed next and allocates CPU
  - Sometimes the only scheduler in a system
  - Short-term scheduler is invoked frequently (milliseconds) ⇒ (must be fast)
- **Long-term scheduler** (or **job scheduler**) – selects which processes should be brought into the ready queue
  - Long-term scheduler is invoked infrequently (seconds, minutes) ⇒ (may be slow)
  - The long-term scheduler controls the **degree of multiprogramming**
- Processes can be described as either:
  - **I/O-bound process** – spends more time doing I/O than computations, many short CPU bursts
  - **CPU-bound process** – spends more time doing computations; few very long CPU bursts
- Long-term scheduler strives for good **process mix**

# Addition of Medium Term Scheduling

- **Medium-term scheduler** can be added if degree of multiple programming needs to decrease
  - Remove process from memory, store on disk, bring back in from disk to continue execution: **swapping**



# Multitasking in Mobile Systems

- Some mobile systems (e.g., early version of iOS) allow only one process to run, others suspended
- Due to screen real estate, user interface limits iOS provides for a
  - Single **foreground** process- controlled via user interface
  - Multiple **background** processes- in memory, running, but not on the display, and with limits
  - Limits include single, short task, receiving notification of events, specific long-running tasks like audio playback
- Android runs foreground and background, with fewer limits
  - Background process uses a **service** to perform tasks
  - Service can keep running even if background process is suspended
  - Service has no user interface, small memory use

# Context Switch

- When CPU switches to another process, the system must **save the state** of the old process and load the **saved state** for the new process via a **context switch**
- **Context** of a process represented in the PCB
- Context-switch time is overhead; the system does no useful work while switching
  - The more complex the OS and the PCB → the longer the context switch
- Time dependent on hardware support
  - Some hardware provides multiple sets of registers per CPU → multiple contexts loaded at once

# Operations on Processes

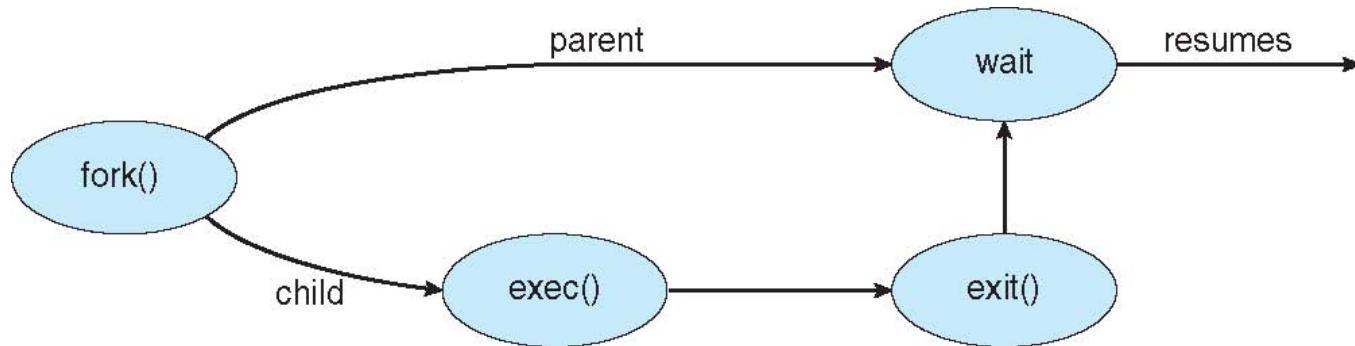
- System must provide mechanisms for:
  - process creation,
  - process termination,
  - and so on as detailed next

# Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes
- Generally, process identified and managed via a process identifier (pid)
- Resource sharing options
  - Parent and children share all resources
  - Children share subset of parent's resources
  - Parent and child share no resources
- Execution options
  - Parent and children execute concurrently
  - Parent waits until children terminate

# Process Creation (Cont.)

- Address space
  - Child duplicate of parent
  - Child has a program loaded into it
- UNIX examples
  - **fork()** system call creates new process
  - **exec()** system call used after a **fork()** to replace the process' memory space with a new program



# Process Termination

- Process executes last statement and then asks the operating system to delete it using the **exit( )** system call.
  - Returns status data from child to parent (via **wait( )**)
  - Process' resources are deallocated by operating system
- Parent may terminate the execution of children processes using the **abort( )** system call. Some reasons for doing so:
  - Child has exceeded allocated resources
  - Task assigned to child is no longer required
  - The parent is exiting and the operating systems does not allow a child to continue if its parent terminates

# Process Termination

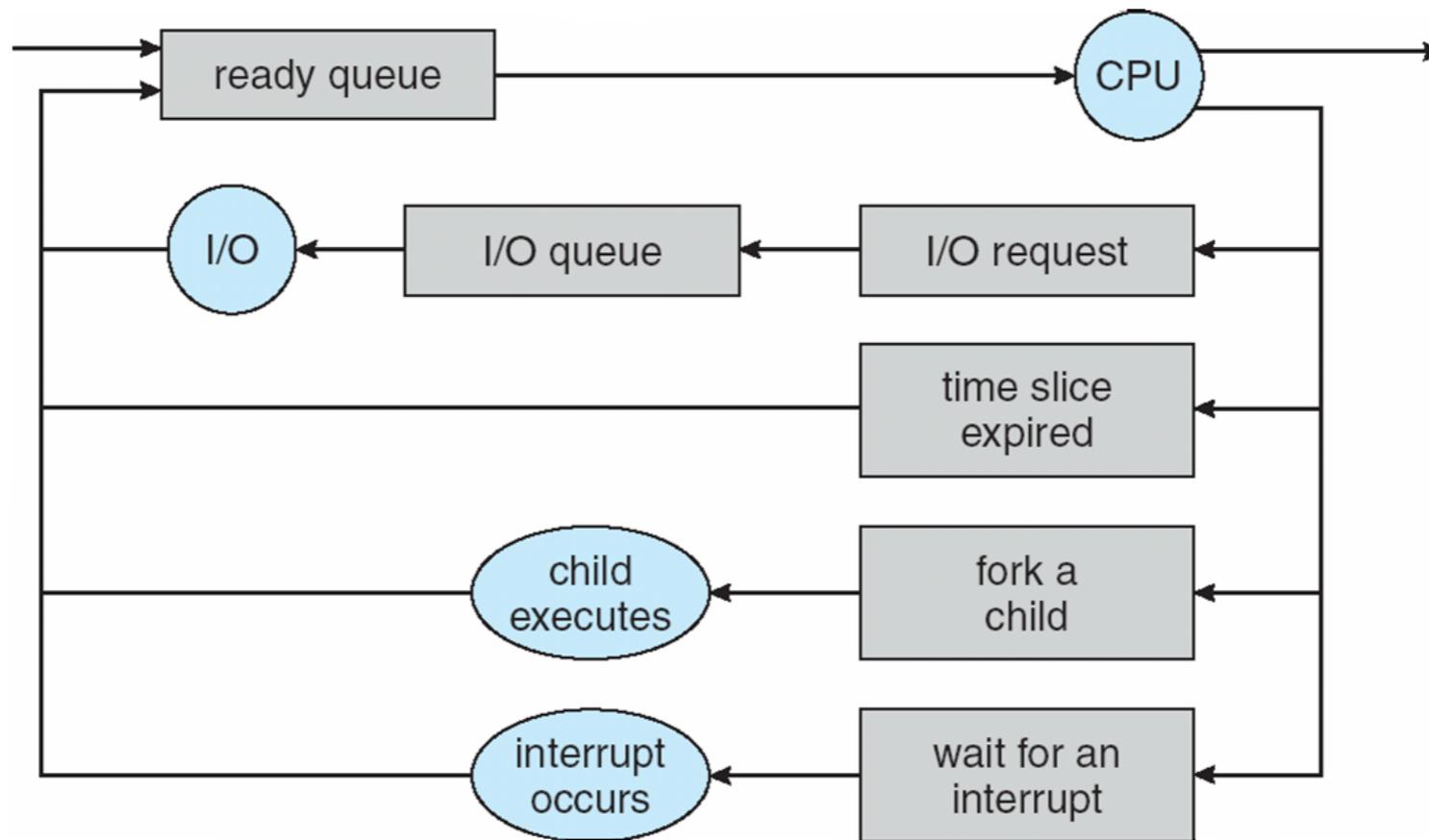
- Some operating systems do not allow child to exists if its parent has terminated. If a process terminates, then all its children must also be terminated.
  - **cascading termination.** All children, grandchildren, etc. are terminated.
  - The termination is initiated by the operating system.
- The parent process may wait for termination of a child process by using the **wait( )** system call . The call returns status information and the pid of the terminated process

```
pid = wait(&status);
```

- If no parent waiting (did not invoke **wait( )**) process is a **zombie**
- If parent terminated without invoking **wait** , process is an **orphan**

# **Operating Systems**

# CPU Scheduling

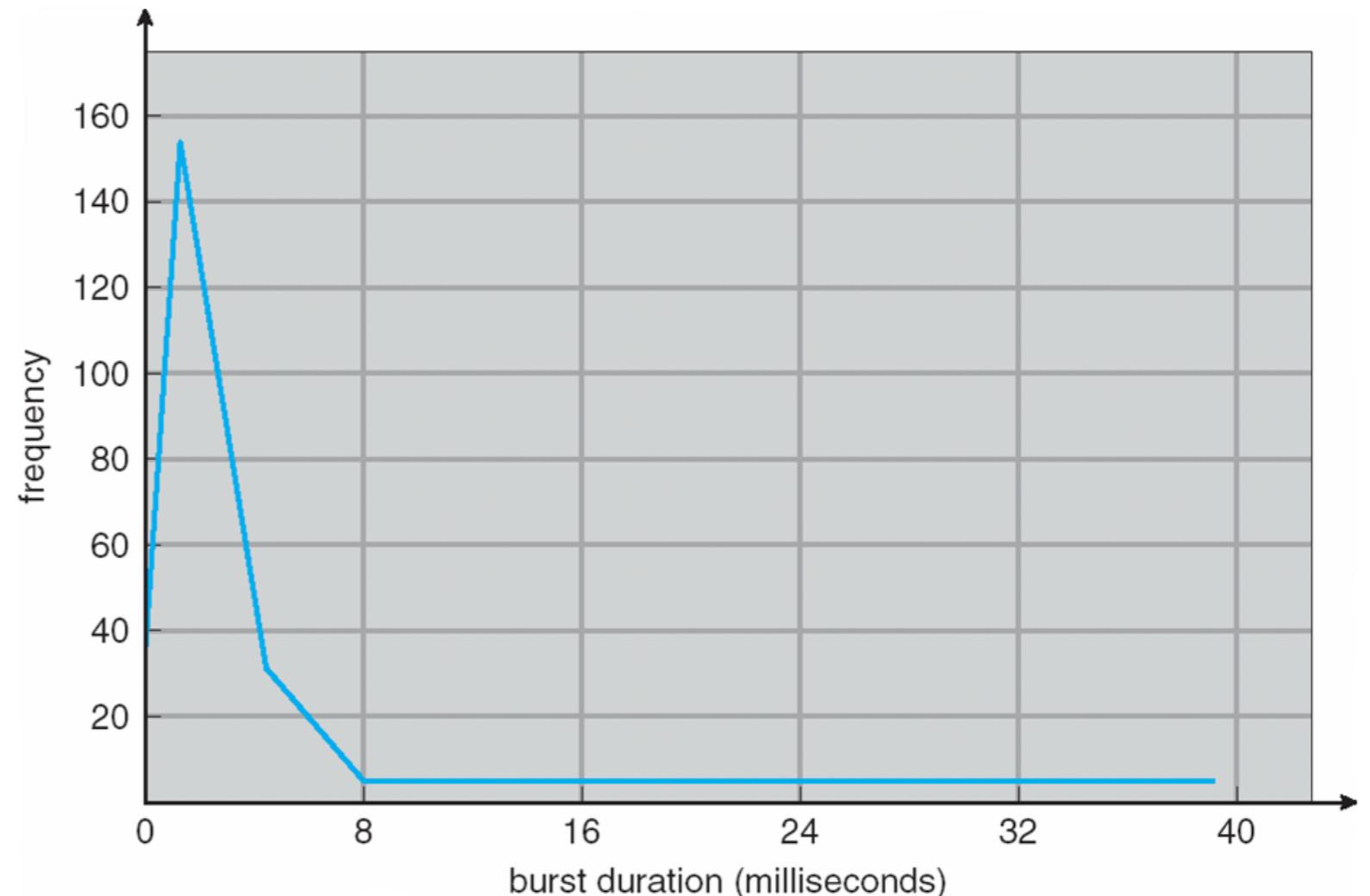
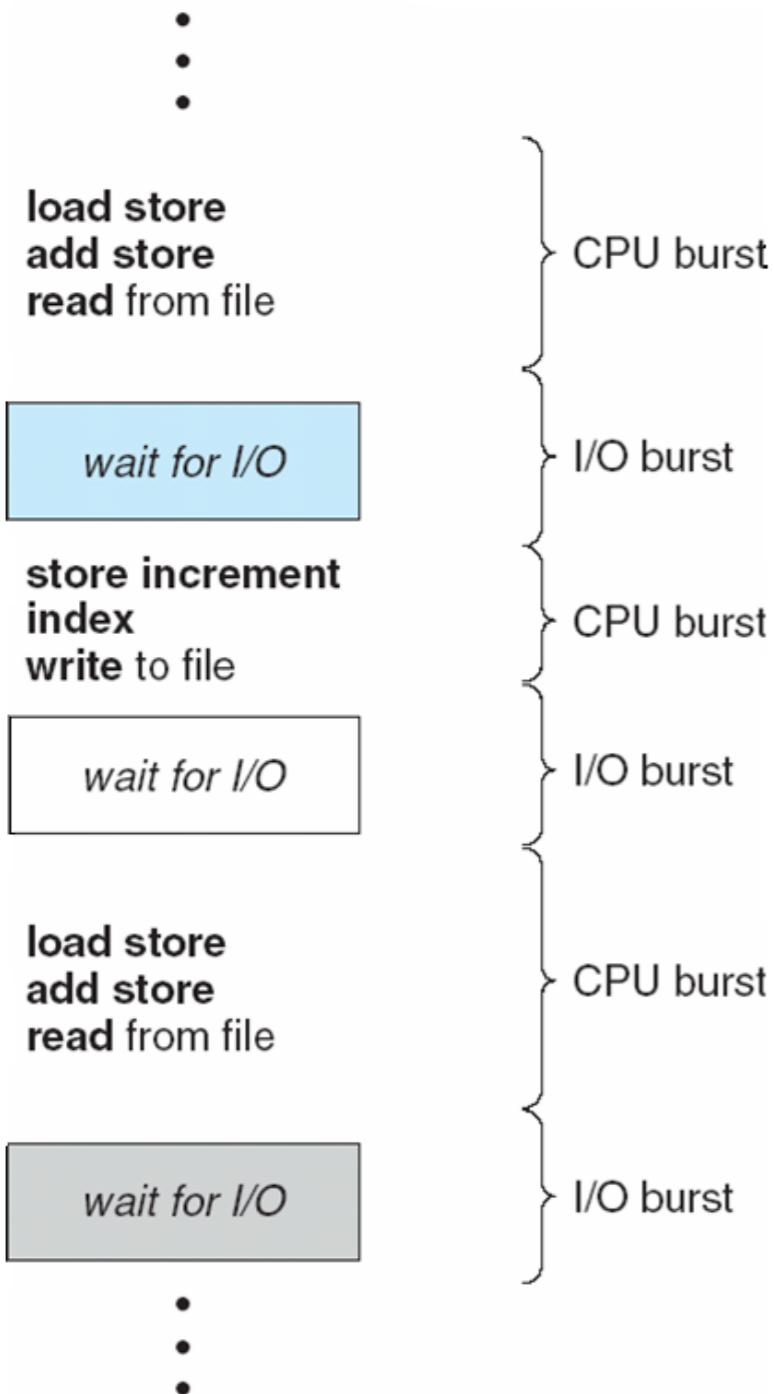


- How is the OS to decide which of several tasks to take off a queue?
- Scheduling: deciding which threads are given access to resources from moment to moment.

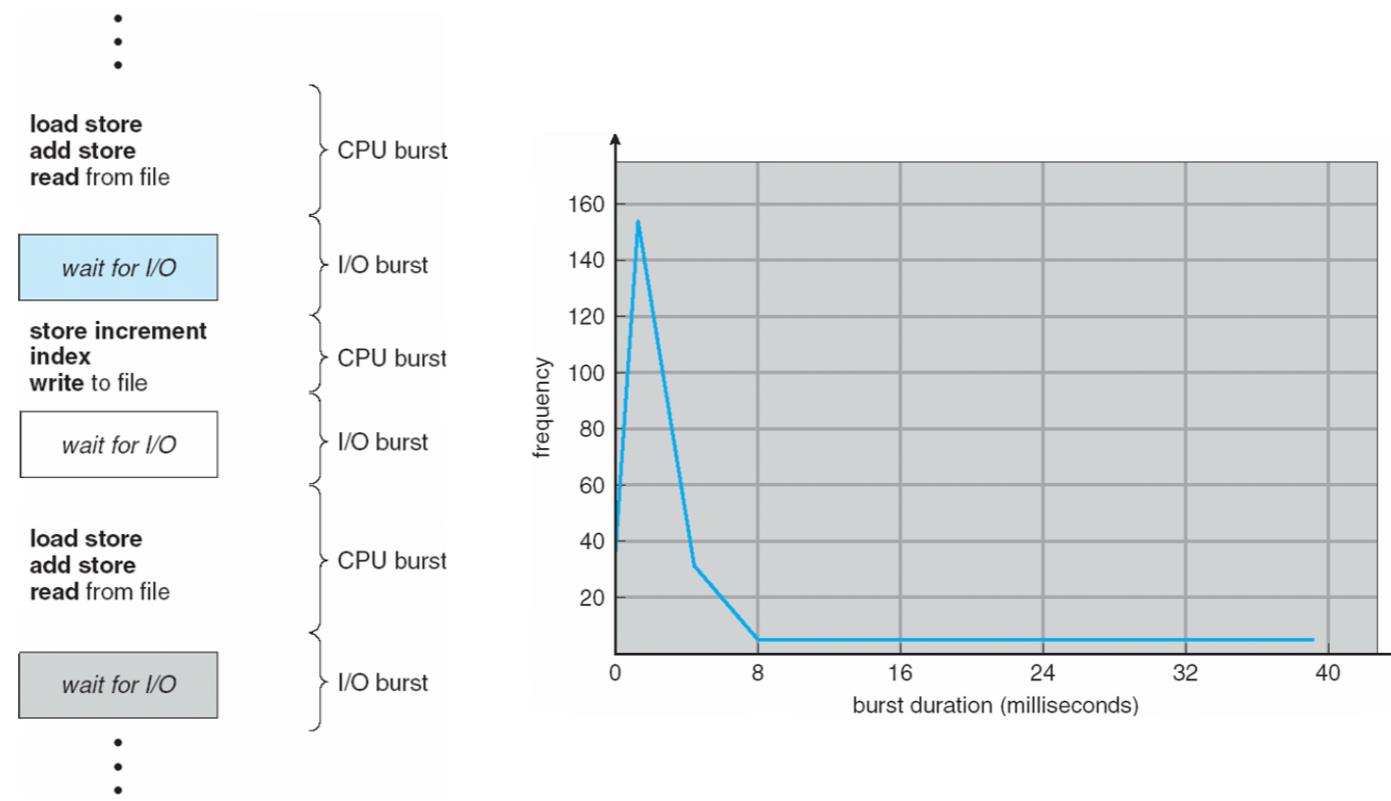
# Assumptions about Scheduling

- CPU scheduling big area of research in early ‘70s
- Many implicit assumptions for CPU scheduling:
  - One program per user
  - One thread per program
  - Programs are independent
- These are unrealistic but simplify the problem
- Does “fair” mean fairness among users or programs?
  - If I run one compilation job and you run five, do you get five times as much CPU?
    - Often times, yes!
- Goal: dole out CPU time to optimize some desired parameters of the system.
  - What parameters?

# Assumption: CPU Bursts



# Assumption: CPU Bursts



- Execution model: programs alternate between bursts of CPU and I/O
  - Program typically uses the CPU for some period of time, then does I/O, then uses CPU again
  - Each scheduling decision is about which job to give to the CPU for use by its next CPU burst
  - With timeslicing, thread may be forced to give up CPU before finishing current CPU burst.

# **What is Important in a Scheduling Algorithm?**



# What is Important in a Scheduling Algorithm?

- Minimize Response Time
  - Elapsed time to do an operation (job)
  - Response time is what the user sees
    - Time to echo keystroke in editor
    - Time to compile a program
    - Real-time Tasks: Must meet deadlines imposed by World
- Maximize Throughput
  - Jobs per second
  - Throughput related to response time, but not identical
    - Minimizing response time will lead to more context switching than if you maximized only throughput
  - Minimize overhead (context switch time) as well as efficient use of resources (CPU, disk, memory, etc.)
- Fairness
  - Share CPU among users in some equitable way
  - Not just minimizing average response time

# Scheduling Algorithms: First-Come, First-Served (FCFS)

- “Run until Done:” FIFO algorithm
- In the beginning, this meant one program runs non-preemptively until it is finished (including any blocking for I/O operations)
- Now, FCFS means that a process keeps the CPU until one or more threads block
- Example: Three processes arrive in order P1, P2, P3.
  - P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3
- Draw the Gantt Chart and compute Average Waiting Time and Average Completion Time.

# Scheduling Algorithms: First-Come, First-Served (FCFS)

- Example: Three processes arrive in order P1, P2, P3.
  - P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3
- Waiting Time
  - P1: 0
  - P2: 24
  - P3: 27
- Completion Time:
  - P1: 24
  - P2: 27
  - P3: 30
- Average Waiting Time:  $(0+24+27)/3 = 17$
- Average Completion Time:  $(24+27+30)/3 = 27$



# Scheduling Algorithms: First-Come, First-Served (FCFS)

- What if their order had been P2, P3, P1?
  - P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3

# Scheduling Algorithms: First-Come, First-Served (FCFS)

- What if their order had been P2, P3, P1?

- P1 burst time: 24
  - P2 burst time: 3
  - P3 burst time: 3



- Waiting Time

- P1: 0
  - P2: 3
  - P3: 6

- Completion Time:

- P1: 3
  - P2: 6
  - P3: 30

- Average Waiting Time:  $(0+3+6)/3 = 3$  (compared to 17)

- Average Completion Time:  $(3+6+30)/3 = 13$  (compared to 27)

# Scheduling Algorithms: First-Come, First-Served (FCFS)

- Average Waiting Time:  $(0+3+6)/3 = 3$  (compared to 17)
- Average Completion Time:  $(3+6+30)/3 = 13$  (compared to 27)
- FIFO Pros and Cons:
  - Simple (+)
  - Short jobs get stuck behind long ones (-)
    - If all you're buying is milk, doesn't it always seem like you are stuck behind a cart full of many items
  - Performance is highly dependent on the order in which jobs arrive (-)

# How Can We Improve on This?



# Round Robin (RR) Scheduling

- FCFS Scheme: Potentially bad for short jobs!
  - Depends on submit order
  - If you are first in line at the supermarket with milk, you don't care who is behind you; on the other hand...
- Round Robin Scheme
  - Each process gets a small unit of CPU time (time quantum)
    - Usually 10-100 ms
  - After quantum expires, the process is preempted and added to the end of the ready queue
  - Suppose N processes in ready queue and time quantum is Q ms:
    - Each process gets  $1/N$  of the CPU time
    - In chunks of at most Q ms
    - What is the maximum wait time for each process?



# Round Robin (RR) Scheduling

- FCFS Scheme: Potentially bad for short jobs!
  - Depends on submit order
  - If you are first in line at the supermarket with milk, you don't care who is behind you; on the other hand...
- Round Robin Scheme
  - Each process gets a small unit of CPU time (time quantum)
    - Usually 10-100 ms
  - After quantum expires, the process is preempted and added to the end of the ready queue
  - Suppose N processes in ready queue and time quantum is Q ms:
    - Each process gets  $1/N$  of the CPU time
    - In chunks of at most Q ms
    - What is the maximum wait time for each process?
      - No process waits more than  $(n-1)q$  time units

# Round Robin (RR) Scheduling

- Round Robin Scheme
  - Each process gets a small unit of CPU time (time quantum)
    - Usually 10-100 ms
  - After quantum expires, the process is preempted and added to the end of the ready queue
  - Suppose N processes in ready queue and time quantum is Q ms:
    - Each process gets  $1/N$  of the CPU time
    - In chunks of at most Q ms
    - What is the maximum wait time for each process?
      - No process waits more than  $(n-1)q$  time units
- Performance Depends on Size of Q
  - Small Q => interleaved
  - Large Q is like...
  - Q must be large with respect to context switch time, otherwise overhead is too high (spending most of your time context switching!)

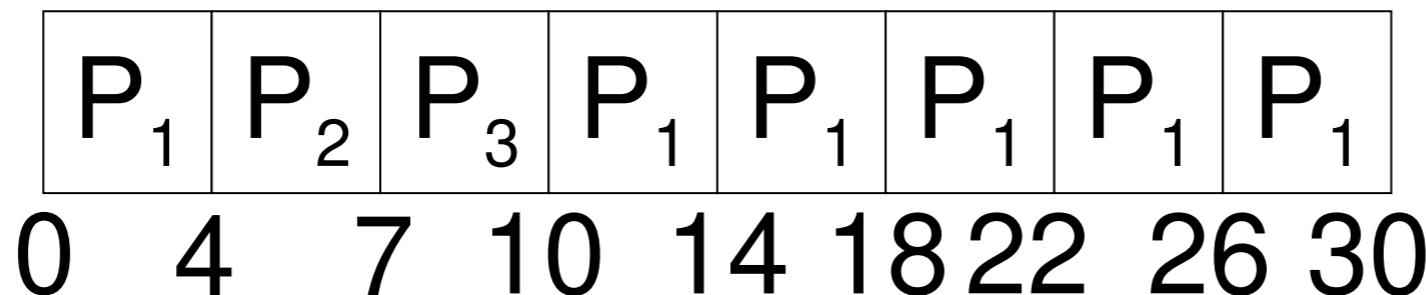
# Round Robin (RR) Scheduling

- Round Robin Scheme
  - Each process gets a small unit of CPU time (time quantum)
    - Usually 10-100 ms
  - After quantum expires, the process is preempted and added to the end of the ready queue
  - Suppose N processes in ready queue and time quantum is Q ms:
    - Each process gets  $1/N$  of the CPU time
    - In chunks of at most Q ms
    - What is the maximum wait time for each process?
      - No process waits more than  $(n-1)q$  time units
- Performance Depends on Size of Q
  - Small Q => interleaved
  - Large Q is like FCFS
  - Q must be large with respect to context switch time, otherwise overhead is too high (spending most of your time context switching!)

# Example of RR with Time Quantum = 4

<u>Process</u>	<u>Burst Time</u>
$P_1$	24
$P_2$	3
$P_3$	3

- The Gantt chart is:



# Example of RR with Time Quantum = 4

Process Burst Time

$P_1$

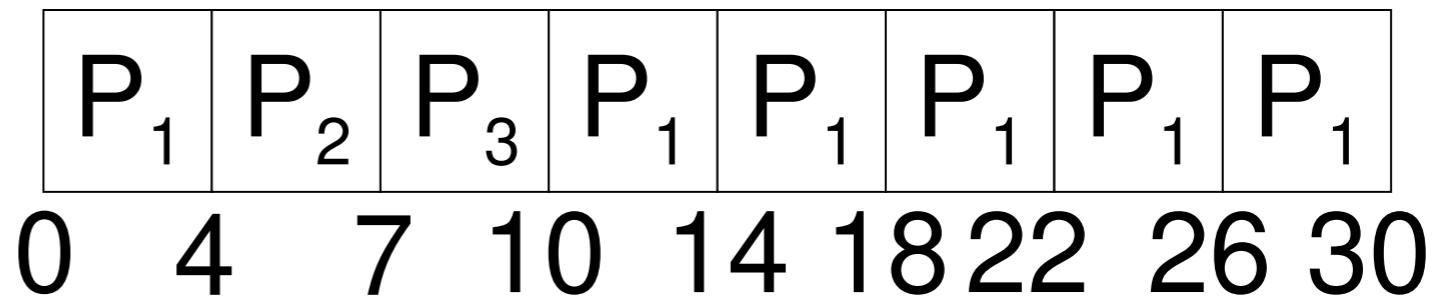
24

$P_2$

3

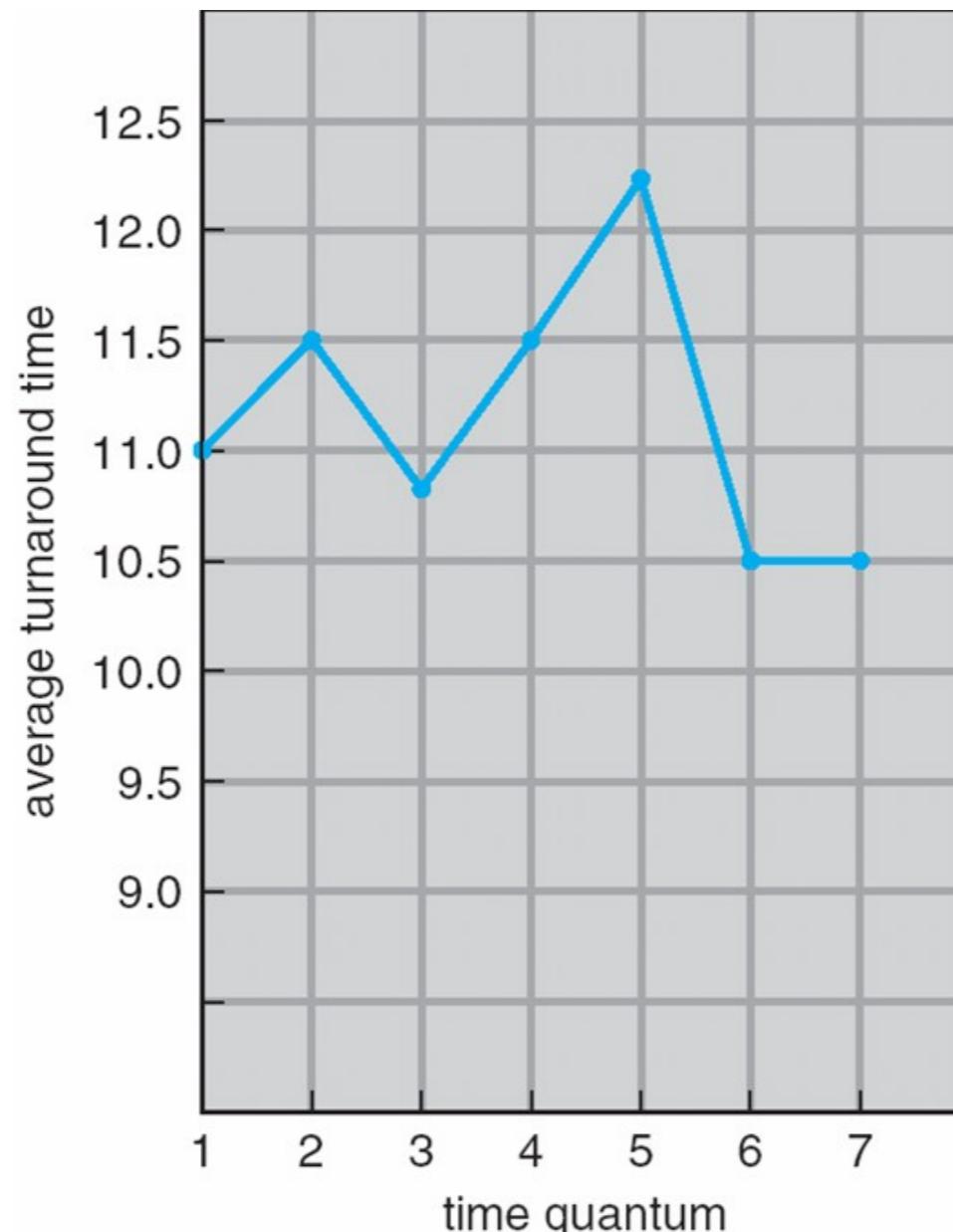
$P_3$

3



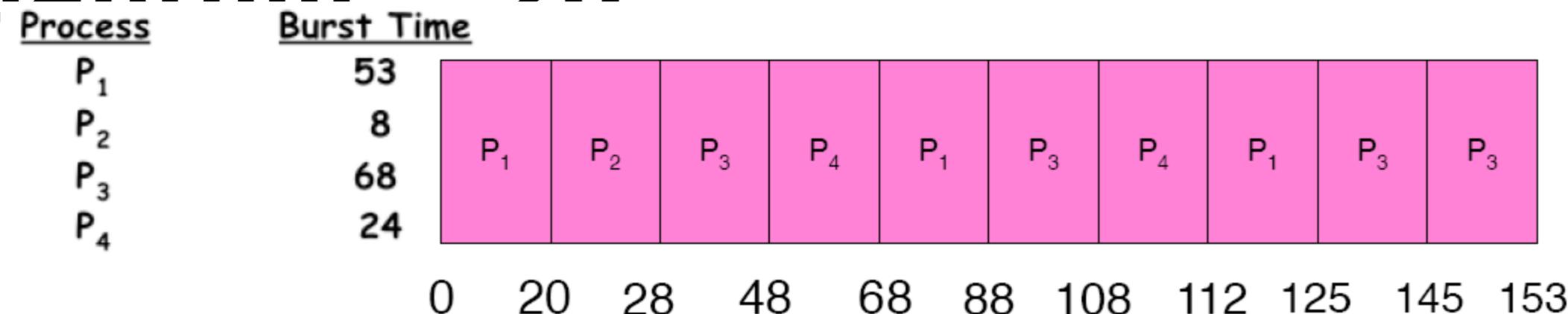
- Waiting Time:
  - $P_1: (10-4) = 6$
  - $P_2: (4-0) = 4$
  - $P_3: (7-0) = 7$
- Completion Time:
  - $P_1: 30$
  - $P_2: 7$
  - $P_3: 10$
- Average Waiting Time:  $(6 + 4 + 7)/3 = 5.67$
- Average Completion Time:  $(30+7+10)/3=15.67$

# Turnaround Time Varies With The Time Quantum



process	time
$P_1$	6
$P_2$	3
$P_3$	1
$P_4$	7

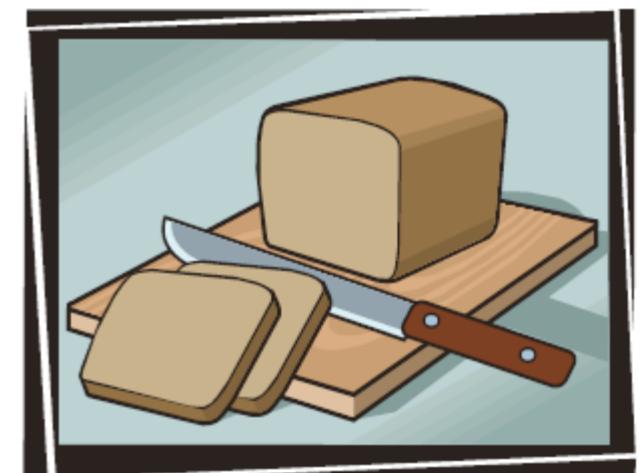
# Example of RR with Time Quantum – 20



- Waiting Time: A process can finish before the time quantum expires, and release the CPU.
  - P<sub>1</sub>:  $(68-20)+(112-88) = 72$
  - P<sub>2</sub>:  $(20-0) = 20$
  - P<sub>3</sub>:  $(28-0)+(88-48)+(125-108) = 85$
  - P<sub>4</sub>:  $(48-0)+(108-68) = 88$
- Completion Time:
  - P<sub>1</sub>: 125
  - P<sub>2</sub>: 28
  - P<sub>3</sub>: 153
  - P<sub>4</sub>: 112
- Average Waiting Time:  $(72+20+85+88)/4 = 66.25$
- Average Completion Time:  $(125+28+153+112)/4 = 104.5$

# RR Summary

- Pros and Cons:
  - Better for short jobs (+)
  - Fair (+)
  - Context-switching time adds up for long jobs (-)
    - The previous examples assumed no additional time was needed for context switching – in reality, this would add to wait and completion time without actually progressing a process towards completion.
    - Remember: the OS consumes resources, too!
- If the chosen quantum is
  - too large, response time suffers
  - infinite, performance is the same as FIFO
  - too small, throughput suffers and percentage overhead grows
- Actual choices of timeslice:
  - UNIX: initially 1 second:
    - Worked when only 1-2 users
    - If there were 3 compilations going on, it took 3 seconds to echo each keystroke!
  - In practice, need to balance short-job performance and long-job throughput:
    - Typical timeslice **10ms-100ms**
    - Typical context-switch overhead **0.1ms – 1ms (about 1%)**

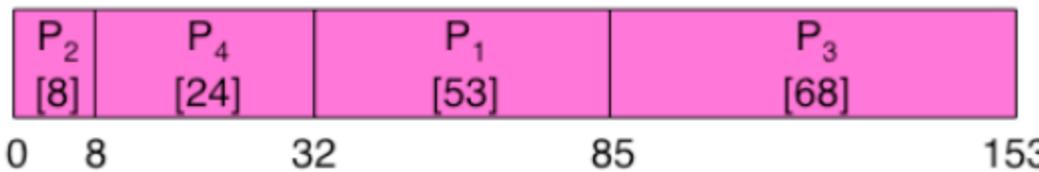


# Comparing FCFS and RR

- Assuming zero-cost context switching time, is RR always better than FCFS?
- Assume 10 jobs, all start at the same time, and each require 100 seconds of CPU time
- RR scheduler quantum of 1 second
- Completion Times (CT)
  - Both FCFS and RR finish at the same time
  - But average response time is much worse under RR!
    - Bad when all jobs are same length
- Also: cache state must be shared between all jobs with RR but can be devoted to each job with FIFO
  - Total time for RR longer even for zero-cost context switch!

Job #	FCFS CT	RR CT
1	100	991
2	200	992
...	...	...
9	900	999
10	1000	1000

# Comparing FCFS and RR



	Quantum	P <sub>1</sub>	P <sub>2</sub>	P <sub>3</sub>	P <sub>4</sub>	Average
Wait Time	Best FCFS	32	0	85	8	$31\frac{1}{4}$
	Q = 1	84	22	85	57	62
	Q = 5	82	20	85	58	$61\frac{1}{4}$
	Q = 8	80	8	85	56	$57\frac{1}{4}$
	Q = 10	82	10	85	68	$61\frac{1}{4}$
	Q = 20	72	20	85	88	$66\frac{1}{4}$
	Worst FCFS	68	145	0	121	$83\frac{1}{2}$
Completion Time	Best FCFS	85	8	153	32	$69\frac{1}{2}$
	Q = 1	137	30	153	81	$100\frac{1}{2}$
	Q = 5	135	28	153	82	$99\frac{1}{2}$
	Q = 8	133	16	153	80	$95\frac{1}{2}$
	Q = 10	135	18	153	92	$99\frac{1}{2}$
	Q = 20	125	28	153	112	$104\frac{1}{2}$
	Worst FCFS	121	153	68	145	$121\frac{3}{4}$

# Scheduling

- The performance we get is somewhat dependent on what “kind” of jobs we are running (short jobs, long jobs, etc.)
- If we could “see the future,” we could mirror best FCFS
- Shortest Job First (SJF) a.k.a. Shortest Time to Completion First (STCF):
  - Run whatever job has the least amount of computation to do
- Shortest Remaining Time First (SRTF) a.k.a. Shortest Remaining Time to Completion First (SRTCF):
  - Preemptive version of SJF: if a job arrives and has a shorter time to completion than the remaining time on the current job, immediately preempt CPU
- These can be applied either to a whole program or the current CPU burst of each program
  - Idea: get short jobs out of the system
  - Big effect on short jobs, only small effect on long ones
  - Result: better average response time

# Scheduling

- But, this is hard to estimate
- We could get feedback from the program or the user, but they have incentive to lie!
- SJF/SRTF are the best you can do at minimizing average response time
  - Provably optimal (SJF among non-preemptive, SRTF among preemptive)
  - Since SRTF is always at least as good as SJF, focus on SRTF
- Comparison of SRTF with FCFS and RR
  - What if all jobs are the same length?
  - What if all jobs have varying length?



# Scheduling

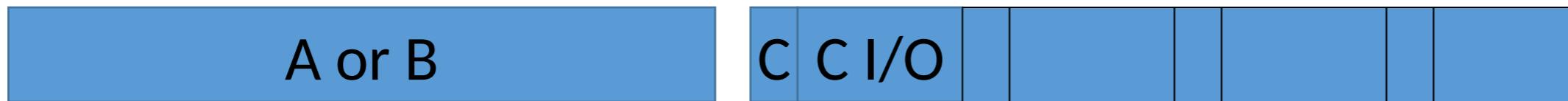
- But, this is hard to estimate
- We could get feedback from the program or the user, but they have incentive to lie!
- SJF/SRTF are the best you can do at minimizing average response time
  - Provably optimal (SJF among non-preemptive, SRTF among preemptive)
  - Since SRTF is always at least as good as SJF, focus on SRTF
- Comparison of SRTF with FCFS and RR
  - What if all jobs are the same length?
    - SRTF becomes the same as FCFS (i.e. FCFS is the best we can do)
  - What if all jobs have varying length?
    - SRTF (and RR): short jobs are not stuck behind long ones

# Example: SRTF

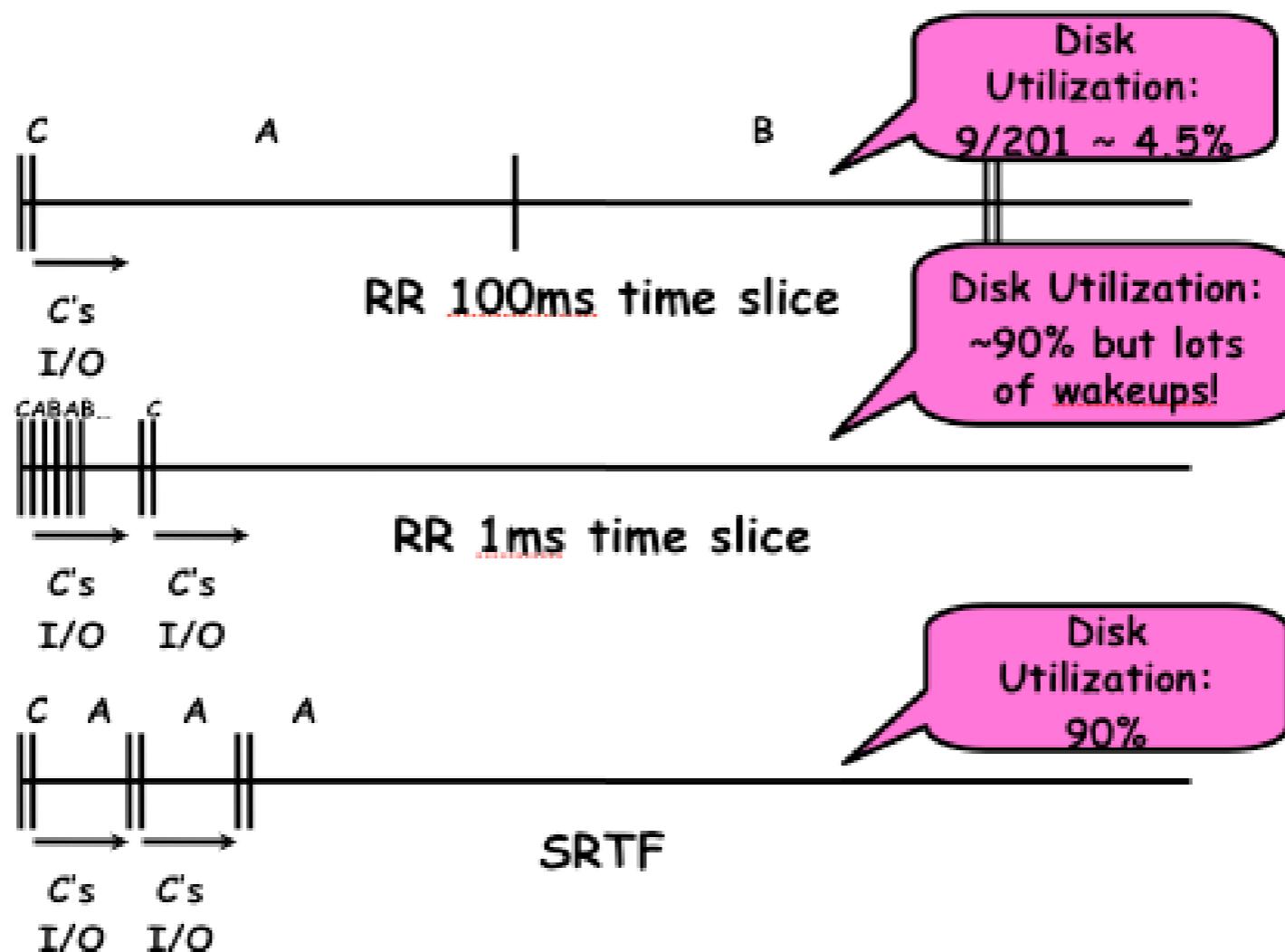


- A,B: both CPU bound, run for a week
- C: I/O bound, loop 1ms CPU, 9ms disk I/O
- If only one at a time, C uses 90% of the disk, A or B could use 100% of the CPU
- With FIFO: Once A and B get in, the CPU is held for two weeks
- What about RR or SRTF?
  - Easier to see with a timeline

# Example: SRTF



- A,B: both CPU bound, run for a week
- C: I/O bound, loop 1ms CPU, 9ms disk I/O



# Last Word on SRTF

- Starvation
  - SRTF can lead to starvation if many small jobs!
  - Large jobs never get to run
- Somehow need to predict future
  - How can we do this?
  - Some systems ask the user
    - When you submit a job, you have to say how long it will take
    - To stop cheating, system kills job if it takes too long
  - But even non-malicious users have trouble predicting runtime of their jobs
- Bottom line, can't really tell how long job will take
  - However, can use SRTF as a yardstick for measuring other policies, since it is optimal
- SRTF Pros and Cons
  - Optimal (average response time) (+)
  - Hard to predict future (-)
  - Unfair, even though we minimized average response time! (-)

# Priority Scheduling

- A priority number (integer) is associated with each process
- The CPU is allocated to the process with the highest priority (smallest integer  $\equiv$  highest priority)
  - Preemptive (if a higher priority process enters, it receives the CPU immediately)
  - Nonpreemptive (higher priority processes must wait until the current process finishes; then, the highest priority ready process is selected)
- SJF is a priority scheduling where priority is the predicted next CPU burst time
- Problem  $\equiv$  Starvation – low priority processes may never execute
- Solution  $\equiv$  Aging – as time progresses increase the priority of the process

# Scheduling Details

- Result approximates SRTF
  - CPU bound jobs drop rapidly to lower queues
  - Short-running I/O bound jobs stay near the top
- Scheduling must be done between the queues
  - Fixed priority scheduling: serve all from the highest priority, then the next priority, etc.
  - Time slice: each queue gets a certain amount of CPU time (e.g., 70% to the highest, 20% next, 10% lowest)
- Countermeasure: user action that can foil intent of the OS designer
  - For multilevel feedback, put in a bunch of meaningless I/O to keep job's priority high
  - But if everyone does this, it won't work!
  - Consider an Othello program, playing against a competitor. Key was to compute at a higher priority than the competitors.
    - Put in printf's, run much faster!

# Scheduling Details

- It is apparent that scheduling is facilitated by having a “good mix” of I/O bound and CPU bound programs, so that there are long and short CPU bursts to prioritize around.
- There is typically a long-term and a short-term scheduler in the OS.
- We have been discussing the design of the short-term scheduler.
- The long-term scheduler decides what processes should be put into the ready queue in the first place for the short-term scheduler, so that the short-term scheduler can make fast decisions on a good mix of a subset of ready processes.
- The rest are held in memory or disk
  - Why else is this helpful?



# Scheduling Details

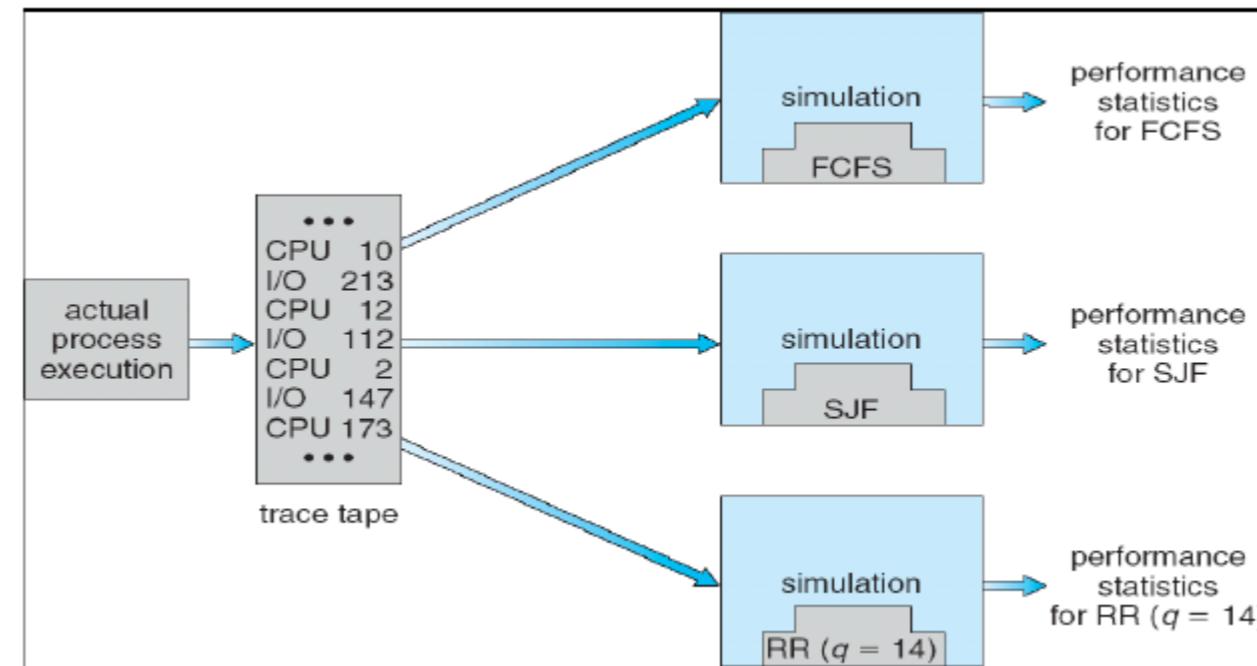
- It is apparent that scheduling is facilitated by having a “good mix” of I/O bound and CPU bound programs, so that there are long and short CPU bursts to prioritize around.
- There is typically a long-term and a short-term scheduler in the OS.
- We have been discussing the design of the short-term scheduler.
- The long-term scheduler decides what processes should be put into the ready queue in the first place for the short-term scheduler, so that the short-term scheduler can make fast decisions on a good mix of a subset of ready processes.
- The rest are held in memory or disk
  - This also provides more free memory for the subset of ready processes given to the short-term scheduler.

# Fairness

- What about fairness?
  - Strict fixed-policy scheduling between queues is unfair (run highest, then next, etc.)
    - Long running jobs may never get the CPU
    - In Multics, admins shut down the machine and found a 10-year-old job
  - Must give long-running jobs a fraction of the CPU even when there are shorter jobs to run
    - Tradeoff: fairness gained by hurting average response time!
- How to implement fairness?
  - Could give each queue some fraction of the CPU
    - i.e., for one long-running job and 100 short-running ones?
    - Like express lanes in a supermarket – sometimes express lanes get so long, one gets better service by going into one of the regular lines
  - Could increase priority of jobs that don't get service (as seen in the multilevel feedback example)
    - This was done in UNIX
    - Ad hoc – with what rate should priorities be increased?
    - As system gets overloaded, no job gets CPU time, so everyone increases in priority
      - Interactive processes suffer

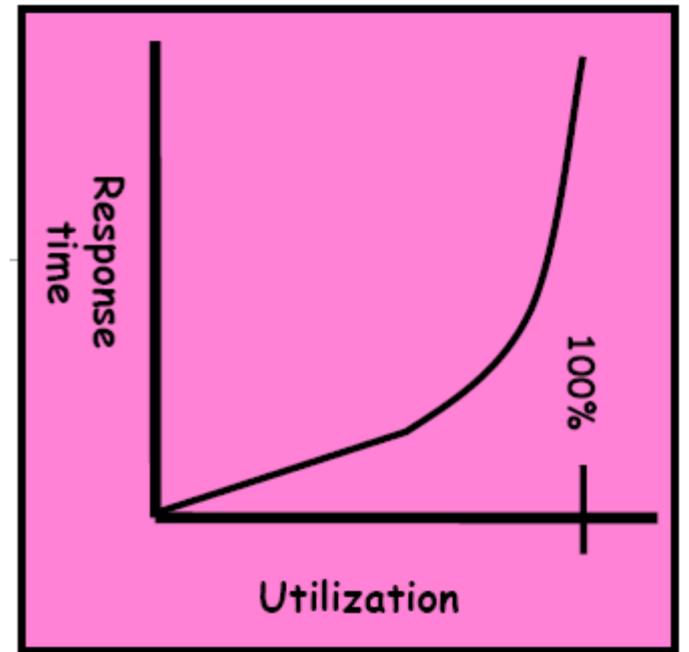
# Scheduling Algorithm Evaluation

- Deterministic Modeling
  - Takes a predetermined workload and compute the performance of each algorithm for that workload
- Queuing Models
  - Mathematical Approach for handling stochastic workloads
- Implementation / Simulation
  - Build system which allows actual algorithms to be run against actual data. Most flexible / general.



# Conclusion

- Scheduling: selecting a waiting process from the ready queue and allocating the CPU to it
- When do the details of the scheduling policy and fairness really matter?
  - When there aren't enough resources to go around
- When should you simply buy a faster computer?
  - Or network link, expanded highway, etc.
  - One approach: buy it when it will pay for itself in improved response time
    - Assuming you're paying for worse response in reduced productivity, customer angst, etc.
    - Might think that you should buy a faster X when X is utilized 100%, but usually, response time goes to infinite as utilization goes to 100%
  - Most scheduling algorithms work fine in the “linear” portion of the load curve, and fail otherwise
  - Argues for buying a faster X when utilization is at the “knee” of the curve



- FCFS scheduling, FIFO Run Until Done:
  - Simple, but short jobs get stuck behind long ones
- RR scheduling:
  - Give each thread a small amount of CPU time when it executes, and cycle between all ready threads
  - Better for short jobs, but poor when jobs are the same length
- SJF/SRTF:
  - Run whatever job has the least amount of computation to do / least amount of remaining computation to do
  - Optimal (average response time), but unfair; hard to predict the future
- Priority Scheduling:
  - Preemptive or Non-preemptive
  - Priority Inversion

# **Unit 3**

# **Threads**

# Threads

- Overview
- Multicore Programming
- Multithreading Models
- Thread Libraries
- Implicit Threading
- Threading Issues
- Operating System Examples

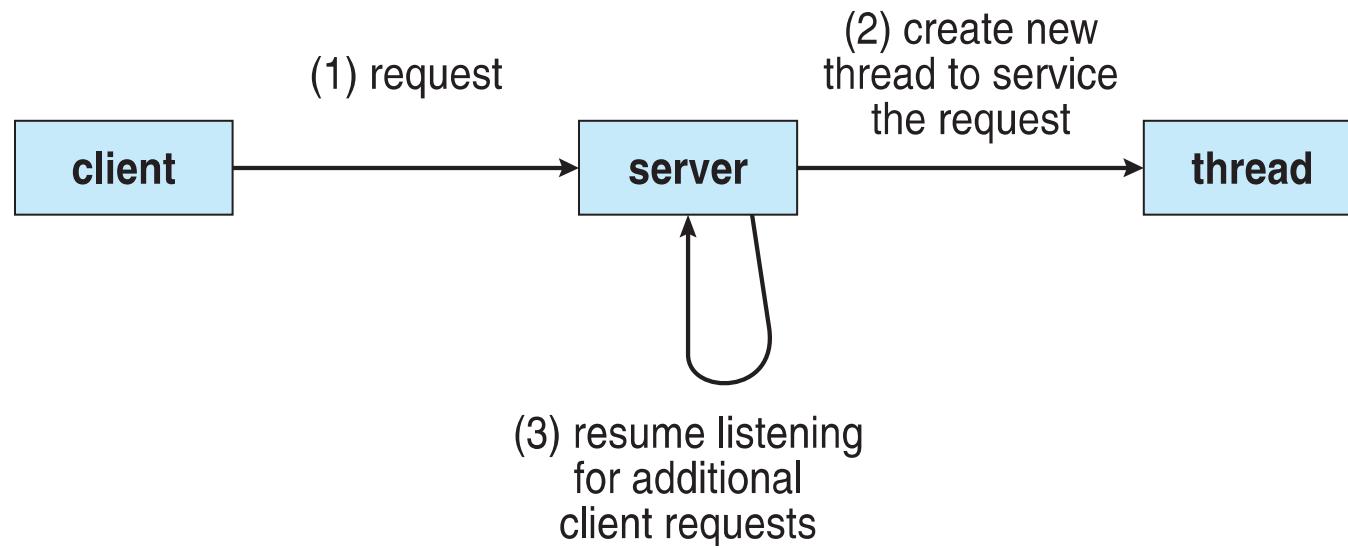
# Objectives

- To introduce the notion of a thread—a fundamental unit of CPU utilization that forms the basis of multithreaded computer systems
- To discuss the APIs for the Pthreads, Windows, and Java thread libraries
- To explore several strategies that provide implicit threading
- To examine issues related to multithreaded programming
- To cover operating system support for threads in Windows and Linux

# Motivation

- Most modern applications are multithreaded
- Threads run within application
- Multiple tasks within the application can be implemented by separate threads
  - Update display
  - Fetch data
  - Spell checking
  - Answer a network request
- Process creation is heavy-weight while thread creation is light-weight
- Can simplify code, increase efficiency
- Kernels are generally multithreaded

# Multithreaded Server Architecture



# Benefits

- **Responsiveness** – may allow continued execution if part of process is blocked, especially important for user interfaces
- **Resource Sharing** – threads share resources of process, easier than shared memory or message passing
- **Economy** – cheaper than process creation, thread switching lower overhead than context switching
- **Scalability** – process can take advantage of multiprocessor architectures

# Multicore Programming

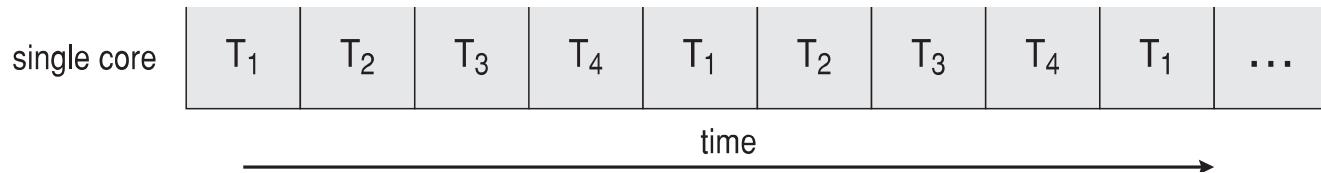
- **Multicore** or **multiprocessor** systems putting pressure on programmers, challenges include:
  - **Dividing activities**
  - **Balance**
  - **Data splitting**
  - **Data dependency**
  - **Testing and debugging**
- **Parallelism** implies a system can perform more than one task simultaneously
- **Concurrency** supports more than one task making progress
  - Single processor / core, scheduler providing concurrency

# Multicore Programming (Cont.)

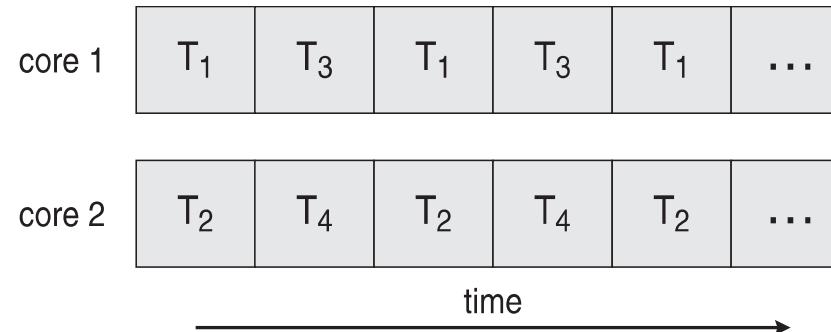
- Types of parallelism
  - **Data parallelism** – distributes subsets of the same data across multiple cores, same operation on each
  - **Task parallelism** – distributing threads across cores, each thread performing unique operation
- As # of threads grows, so does architectural support for threading
  - CPUs have cores as well as ***hardware threads***
  - Consider Oracle SPARC T4 with 8 cores, and 8 hardware threads per core

# Concurrency vs. Parallelism

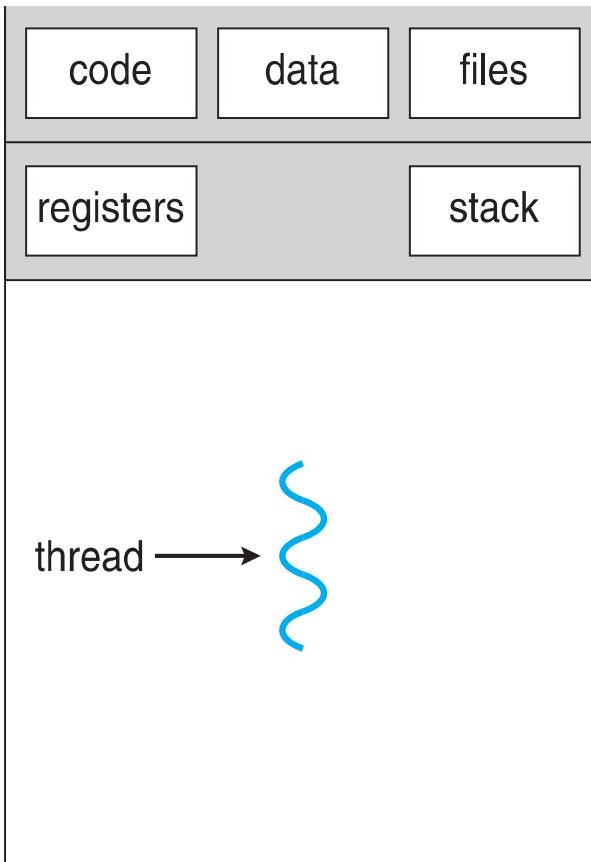
- Concurrent execution on single-core system:



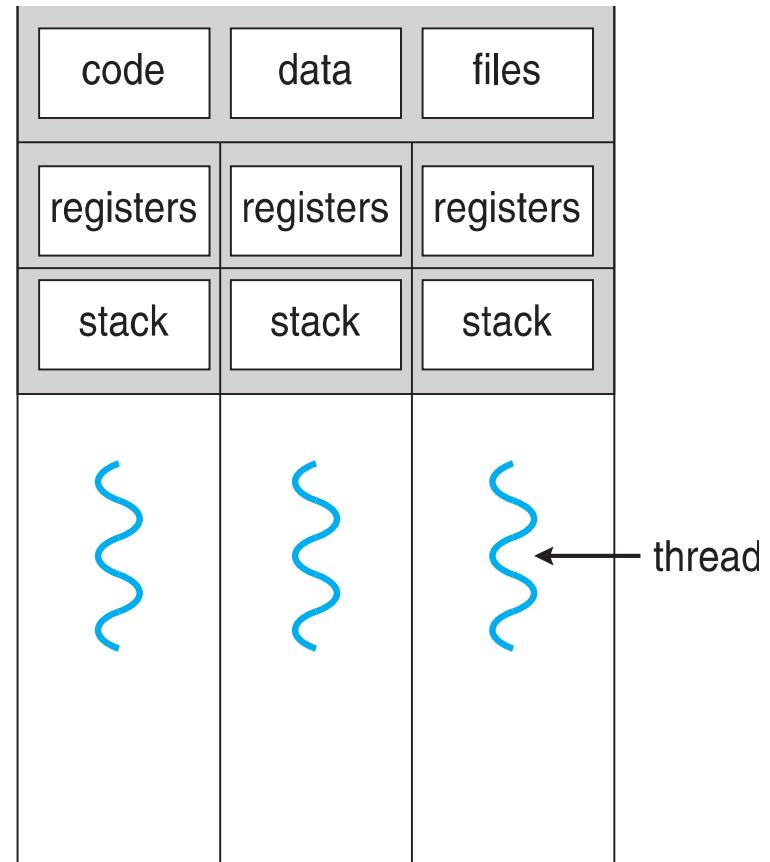
- Parallelism on a multi-core system:



# Single and Multithreaded Processes



single-threaded process



multithreaded process

# Amdahl's Law

- Identifies performance gains from adding additional cores to an application that has both serial and parallel components
- $S$  is serial portion
- $N$  processing cores

$$speedup \leq \frac{1}{S + \frac{(1-S)}{N}}$$

- That is, if application is 75% parallel / 25% serial, moving from 1 to 2 cores results in speedup of 1.6 times
- As  $N$  approaches infinity, speedup approaches  $1 / S$

**Serial portion of an application has disproportionate effect on performance gained by adding additional cores**

- But does the law take into account contemporary multicore systems?

# User Threads and Kernel Threads

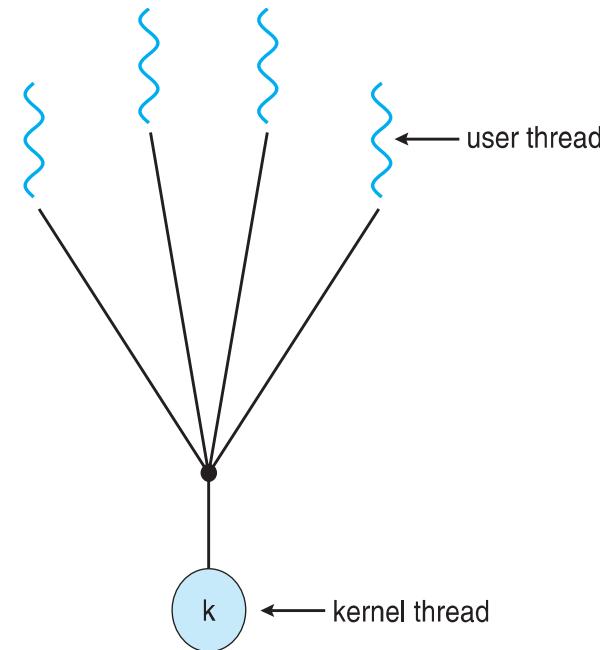
- **User threads** - management done by user-level threads library
- Three primary thread libraries:
  - POSIX [Pthreads](#)
  - Windows threads
  - Java threads
- **Kernel threads** - Supported by the Kernel
- Examples – virtually all general purpose operating systems, including:
  - Windows
  - Solaris
  - Linux
  - Tru64 UNIX
  - Mac OS X

# Multithreading Models

- Many-to-One
- One-to-One
- Many-to-Many

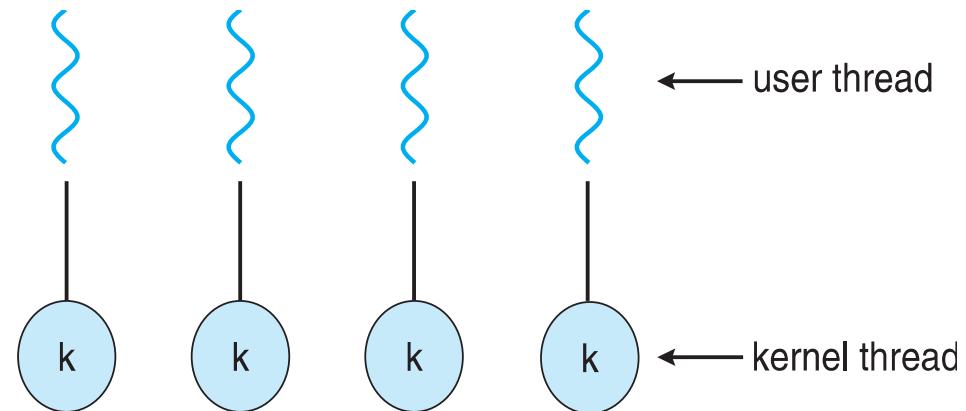
# Many-to-One

- Many user-level threads mapped to single kernel thread
- One thread blocking causes all to block
- Multiple threads may not run in parallel on multicore system because only one may be in kernel at a time
- Few systems currently use this model
- Examples:
  - Solaris Green Threads
  - GNU Portable Threads



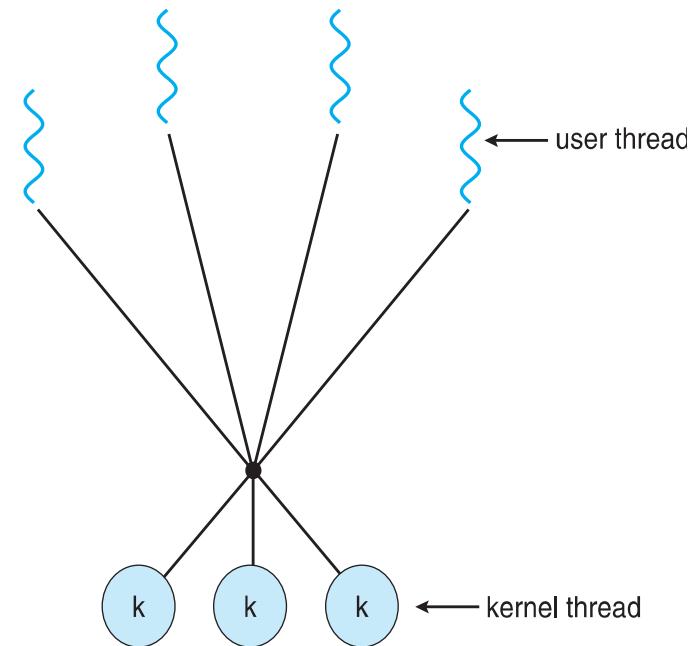
# One-to-One

- Each user-level thread maps to kernel thread
- Creating a user-level thread creates a kernel thread
- More concurrency than many-to-one
- Number of threads per process sometimes restricted due to overhead
- Examples
  - Windows
  - Linux
  - Solaris 9 and later



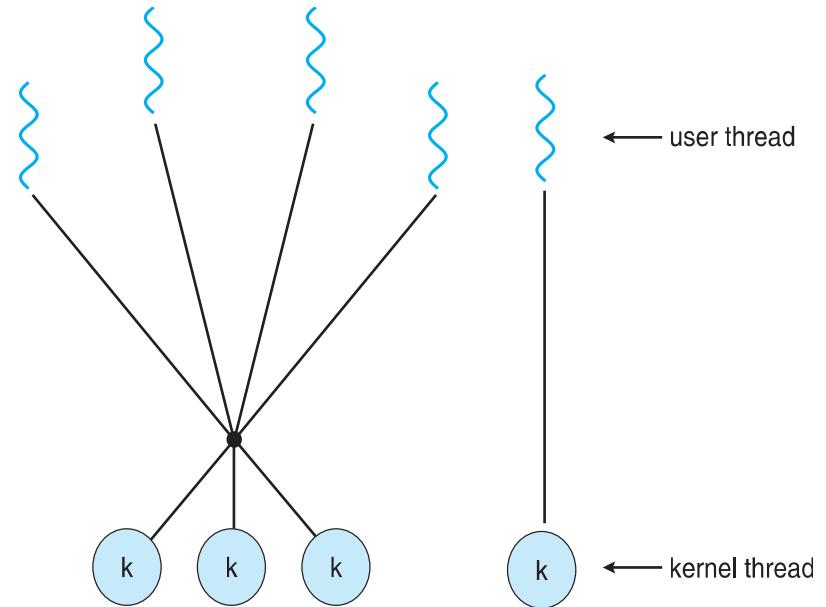
# Many-to-Many Model

- Allows many user level threads to be mapped to many kernel threads
- Allows the operating system to create a sufficient number of kernel threads
- Solaris prior to version 9
- Windows with the *ThreadFiber* package



# Two-level Model

- Similar to M:M, except that it allows a user thread to be **bound** to kernel thread
- Examples
  - IRIX
  - HP-UX
  - Tru64 UNIX
  - Solaris 8 and earlier



# Thread Libraries

- **Thread library** provides programmer with API for creating and managing threads
- Two primary ways of implementing
  - Library entirely in user space
  - Kernel-level library supported by the OS

# Pthreads

- May be provided either as user-level or kernel-level
- A POSIX standard (IEEE 1003.1c) API for thread creation and synchronization
- *Specification*, not *implementation*
- API specifies behavior of the thread library, implementation is up to development of the library
- Common in UNIX operating systems (Solaris, Linux, Mac OS X)

# Implicit Threading

- Growing in popularity as numbers of threads increase, program correctness more difficult with explicit threads
- Creation and management of threads done by compilers and run-time libraries rather than programmers
- Three methods explored
  - Thread Pools
  - OpenMP
  - Grand Central Dispatch
- Other methods include Microsoft Threading Building Blocks (TBB), `java.util.concurrent` package

# Thread Pools

- Create a number of threads in a pool where they await work
- Advantages:
  - Usually slightly faster to service a request with an existing thread than create a new thread
  - Allows the number of threads in the application(s) to be bound to the size of the pool
  - Separating task to be performed from mechanics of creating task allows different strategies for running task
    - i.e. Tasks could be scheduled to run periodically
- Windows API supports thread pools:

```
DWORD WINAPI PoolFunction(VOID Param) {  
    /*  
     * this function runs as a separate thread.  
    */  
}
```

# Threading Issues

- Semantics of **fork()** and **exec()** system calls
- Signal handling
  - Synchronous and asynchronous
- Thread cancellation of target thread
  - Asynchronous or deferred
- Thread-local storage
- Scheduler Activations

# **Operating System**

# Syllabus

Evolution of Operating Systems: Types of operating systems - Different views of the operating systems – Principles of Design and Implementation. The process concept – system programmer's view of processes – operating system's views of processes – operating system services for process management. Process scheduling – Schedulers – Scheduling Algorithms.

Structural overview, Concept of process and Process synchronization, Process Management and Scheduling, Hardware requirements: protection, context switching, privileged mode; Threads and their Management;

Memory Management paging, virtual memory management, Contiguous allocation – static, dynamic partitioned memory allocation – segmentation. Non-contiguous allocation – paging – Hardware support – Virtual Memory, Dynamic Resource Allocation.

File Systems: A Simple file system – General model of a file system – Symbolic file system – Access control verification – Logical file system – Physical file system – allocation strategy module – Device strategy module, I/O initiators, Device handlers – Disk scheduling, Design ofIO systems, File Management.

**TEXT BOOK:**

1. Operating System Concepts – Abraham Silberschatz, Peter Baer Galvin, Greg Gagne, 8th edition, Wiley-India, 2009.
2. Mordern Operating Systems – Andrew S. Tanenbaum, 3rd Edition, PHI
3. Operating Systems: A Spiral Approach – Elmasri, Carrick, Levine, TMH Edition

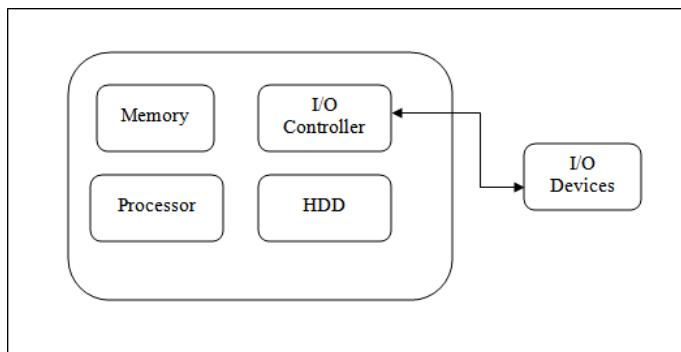
**REFERENCE BOOK:**

1. Operating Systems – Flynn, McHoes, Cengage Learning
2. Operating Systems – Pabitra Pal Choudhury, PHI
3. Operating Systems – William Stallings, Prentice Hall
4. Operating Systems – H.M. Deitel, P. J. Deitel, D. R. Choffnes, 3rd Edition, Pearson

## **Operating System:**

- An operating system is a program which manages all the computer hardware.
- It provides the base for application program and acts as an intermediary between a user and the computer hardware.
- The operating system has two objectives such as:
  - Firstly, an operating system controls the computer's hardware.
  - The second objective is to provide an interactive interface to the user and interpret commands so that it can communicate with the hardware.
- The operating system is very important part of almost every computer system.

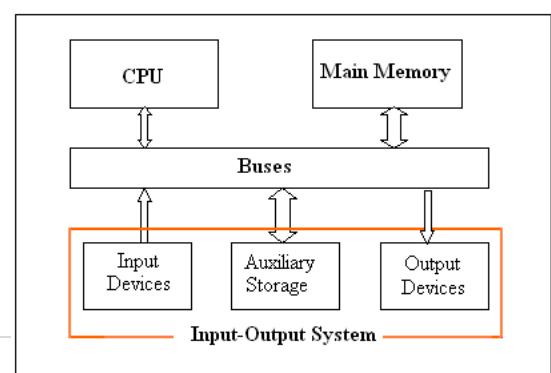
## **Managing Hardware**



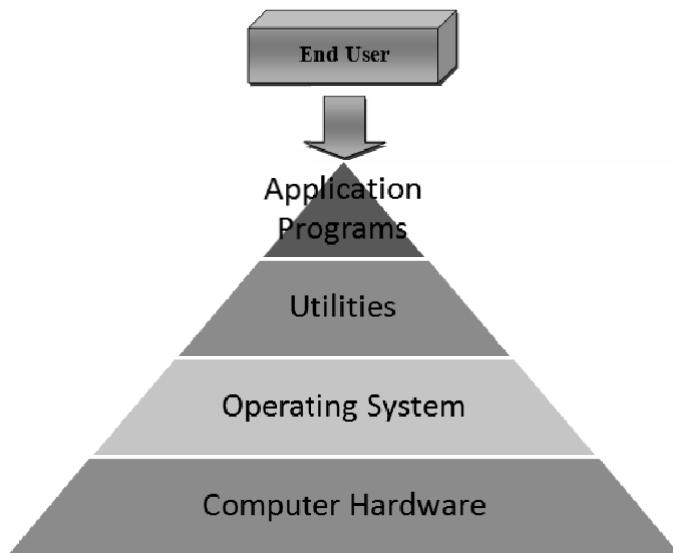
- The prime objective of operating system is to manage & control the various hardware resources of a computer system.
- These hardware resources include processor, memory, and disk space and so on.
- The output result was display in monitor. In addition to communicating with the hardware the operating system provides an error handling procedure and display an error notification.
- If a device not functioning properly, the operating system cannot communicate with the device.

## **Providing an Interface**

- The operating system organizes application so that users can easily access, use and store them.



- It provides a stable and consistent way for applications to deal with the hardware without the user having known details of the hardware.
- If the program is not functioning properly, the operating system again takes control, stops the application and displays the appropriate error message.
- Computer system components are divided into 5 parts
  - Computer hardware
  - operating system
  - utilities
  - Application programs
  - End user



- The operating system controls and coordinate a user of hardware and various application programs for various users.
- It is a program that directly interacts with the hardware.
- The operating system is the first encoded with the Computer and it remains on the memory all time thereafter.

### **System goals**

- The purpose of an operating system is to be provided an environment in which an user can execute programs.
- Its primary goals are to make the computer system convenience for the user.
- Its secondary goals are to use the computer hardware in efficient manner.

## **View of operating system**

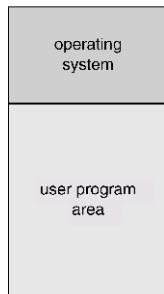
- **User view:** The user view of the computer varies by the interface being used. The examples are -windows XP, vista, windows 7 etc. Most computer user sit in the in front of personal computer (pc) in this case the operating system is designed mostly for easy use with some attention paid to resource utilization. Some user sit at a terminal connected to a mainframe/minicomputer. In this case other users are accessing the same computer through the other terminals. There user are share resources and may exchange the information. The operating system in this case is designed to maximize resources utilization to assume that all available CPU time, memory and I/O are used efficiently and no individual user takes more than his/her fair and share. The other users sit at workstations connected to network of other workstations and servers. These users have dedicated resources but they share resources such as networking and servers like file, compute and print server. Here the operating system is designed to compromise between individual usability and resource utilization.
- **System view:** From the computer point of view the operating system is the program which is most intermediate with the hardware. An operating system has resources as hardware and software which may be required to solve a problem like CPU time, memory space, file storage space and I/O devices and so on. That's why the operating system acts as manager of these resources. Another view of the operating system is it is a control program. A control program manages the execution of user programs to present the errors in proper use of the computer. It is especially concerned of the user the operation and controls the I/O devices.

## **Types of Operating System**

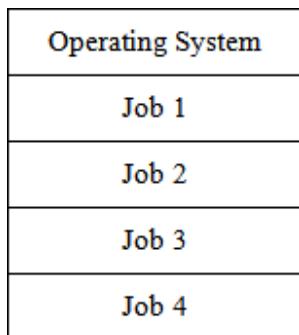
1. **Mainframe System:** It is the system where the first computer used to handle many commercial scientific applications. The growth of mainframe systems traced from simple batch system where the computer runs one and only one application to time shared systems which allowed for user interaction with the computer system
  - a. **Batch /Early System:** Early computers were physically large machine. The common input devices were card readers, tape drivers. The common output devices were line printers, tape drivers and card punches. In these systems the user did not interact directlywith the computer system. Instead the user preparing a job which consists of programming data and some control information and then submitted it to the computer

operator after some time the output is appeared. The output in these early computer was fairly simple is main task was to transfer control automatically from one job to next. The operating system always resides in the memory. To speed up processing operators batched the jobs with similar needs and ran them together as a group. The disadvantages of batch system are that in this execution environment the CPU is often idle because the speed up of I/O devices is much slower than the CPU.

**Memory Layout for a Simple Batch System**



- b. **Multiprogrammed System:** Multiprogramming concept increases CPU utilization by organization jobs so that the CPU always has one job to execute the idea behind multiprogramming concept. The operating system keeps several jobs in memory simultaneously as shown in below figure.



This set of job is subset of the jobs kept in the job pool. The operating system picks and beginning to execute one of the jobs in the memory. In this environment the operating system simply switches and executes another job. When a job needs to wait the CPU is simply switched to another job and so on. The multiprogramming operating system is sophisticated because the operating system makes decisions for the user. This is known as scheduling. If several jobs are ready to run at the same time the system choose one among

them. This is known as CPU scheduling. The disadvantages of the multiprogrammed system are

- It does not provide user interaction with the computer system during the program execution.
- The introduction of disk technology solved these problems rather than reading the cards from card reader into disk. This form of processing is known as spooling.

SPOOL stands for simultaneous peripheral operations online. It uses the disk as a huge buffer for reading from input devices and for storing output data until the output devices accept them. It is also used for processing data at remote sites. The remote processing is done at its own speed with no CPU intervention. Spooling overlaps the input, output one job with computation of other jobs. Spooling has a beneficial effect on the performance of the systems by keeping both CPU and I/O devices working at much higher time.

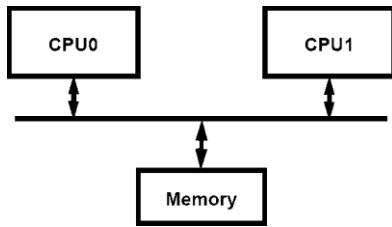
c. **Time Sharing System:** The time sharing system is also known as multi user systems. The CPU executes multiple jobs by switching among them but the switches occurs so frequently that the user can interact with each program while it is running. An interactive computer system provides direct communication between a user and system. The user gives instruction to the operating systems or to a program directly using keyboard or mouse and wait for immediate results. So the response time will be short. The time sharing system allows many users to share the computer simultaneously. Since each action in this system is short, only a little CPU time is needed for each user. The system switches rapidly from one user to the next so each user feels as if the entire computer system is dedicated to his use, even though it is being shared by many users. The disadvantages of time sharing system are:

- It is more complex than multiprogrammed operating system
- The system must have memory management & protection, since several jobs are kept in memory at the same time.
- Time sharing system must also provide a file system, so disk management is required.
- It provides mechanism for concurrent execution which requires complex CPU scheduling schemes.

2. **Personal Computer System/Desktop System:** Personal computer appeared in 1970's. They are microcomputers that are smaller & less expensive than mainframe systems. Instead of maximizing CPU & peripheral utilization, the systems opt for maximizing user convenience & responsiveness. At first file protection was not necessary on a personal machine. But when other computers 2<sup>nd</sup> other users can access the files on a pc file protection becomes necessary. The lack of protection made it easy for malicious programs to destroy data on such systems. These programs may be self-replicating & they spread via worm or virus mechanisms. They can disrupt entire companies or even world wide networks. E.g: windows 98, windows 2000, Linux.
3. **Microprocessor Systems/ Parallel Systems/ Tightly coupled Systems:** These Systems have more than one processor in close communications which share the computer bus, clock, memory & peripheral devices. Ex: UNIX, LINUX. Multiprocessor Systems have 3 main advantages.
  - a. **Increased throughput:** No. of processes computed per unit time. By increasing the no. of processors more work can be done in less time. The speed up ratio with N processors is not N, but it is less than N. Because a certain amount of overhead is incurred in keeping all the parts working correctly.
  - b. **Increased Reliability:** If functions can be properly distributed among several processors, then the failure of one processor will not halt the system, but slow it down. This ability to continue to operate in spite of failure makes the system fault tolerant.
  - c. **Economic scale:** Multiprocessor systems can save money as they can share peripherals, storage & power supplies.

The various types of multiprocessing systems are:

- **Symmetric Multiprocessing (SMP):** Each processor runs an identical copy of the operating system & these copies communicate with one another as required. Ex: Encore's version of UNIX for multi max computer. Virtually, all modern operating system including Windows NT, Solaris, Digital UNIX, OS/2 & LINUX now provide support for SMP.



- **Asymmetric Multiprocessing (Master – Slave Processors):** Each processor is designed for a specific task. A master processor controls the system & schedules & allocates the work to the slave processors. Ex- Sun's Operating system SUNOS version 4 provides asymmetric multiprocessing.
4. **Distributed System/Loosely Coupled Systems:** In contrast to tightly coupled systems, the processors do not share memory or a clock. Instead, each processor has its own local memory. The processors communicate with each other by various communication lines such as high speed buses or telephone lines. Distributed systems depend on networking for their functionalities. By being able to communicate distributed systems are able to share computational tasks and provide a rich set of features to the users. Networks vary by the protocols used, the distances between the nodes and transport media. TCP/IP is the most common network protocol. The processor is a distributed system varies in size and function. It may microprocessors, work stations, minicomputer, and large general purpose computers. Network types are based on the distance between the nodes such as LAN (within a room, floor or building) and WAN (between buildings, cities or countries). The advantages of distributed system are resource sharing, computation speed up, reliability, communication.
5. **Real time Systems:** Real time system is used when there are rigid time requirements on the operation of a processor or flow of data. Sensors bring data to the computers. The computer analyzes data and adjusts controls to modify the sensors inputs. System that controls scientific experiments, medical imaging systems and some display systems are real time systems. The disadvantages of real time system are:
- a. A real time system is considered to function correctly only if it returns the correct result within the time constraints.
  - b. Secondary storage is limited or missing instead data is usually stored in short term memory or ROM.
  - c. Advanced OS features are absent.
- Real time system is of two types such as:

- **Hard real time systems:** It guarantees that the critical task has been completed on time. The sudden task is takes place at a sudden instant of time.
- **Soft real time systems:** It is a less restrictive type of real time system where a critical task gets priority over other tasks and retains that priority until it completes. These have more limited utility than hard real time systems. Missing an occasional deadline is acceptable e.g. QNX, VX works. Digital audio or multimedia is included in this category.

It is a special purpose OS in which there are rigid time requirements on the operation of a processor. A real time OS has well defined fixed time constraints. Processing must be done within the time constraint or the system will fail. A real time system is said to function correctly only if it returns the correct result within the time constraint. These systems are characterized by having time as a key parameter.

### **Basic Functions of Operation System:**

The various functions of operating system are as follows:

#### **1. Process Management:**

- A program does nothing unless their instructions are executed by a CPU. A process is a program in execution. A time shared user program such as a compiler is a process. A word processing program being run by an individual user on a pc is a process.
- A system task such as sending output to a printer is also a process. A process needs certain resources including CPU time, memory files & I/O devices to accomplish its task.
- These resources are either given to the process when it is created or allocated to it while it is running. The OS is responsible for the following activities of process management.
- Creating & deleting both user & system processes.
- Suspending & resuming processes.
- Providing mechanism for process synchronization.
- Providing mechanism for process communication.
- Providing mechanism for deadlock handling.

#### **2. Main Memory Management:**

The main memory is central to the operation of a modern computer system. Main memory is a large array of words or bytes ranging in size from hundreds of thousand to billions. Main memory stores the quickly accessible data shared by the CPU & I/O device. The central processor reads instruction from main memory during instruction fetch cycle & it both reads

& writes data from main memory during the data fetch cycle. The main memory is generally the only large storage device that the CPU is able to address & access directly. For example, for the CPU to process data from disk. Those data must first be transferred to main memory by CPU generated E/O calls. Instruction must be in memory for the CPU to execute them. The OS is responsible for the following activities in connection with memory management.

- Keeping track of which parts of memory are currently being used & by whom.
- Deciding which processes are to be loaded into memory when memory space becomes available.
- Allocating & deallocating memory space as needed.

### **3. File Management:**

File management is one of the most important components of an OS computer can store information on several different types of physical media magnetic tape, magnetic disk & optical disk are the most common media. Each medium is controlled by a device such as disk drive or tape drive those has unique characteristics. These characteristics include access speed, capacity, data transfer rate & access method (sequential or random). For convenient use of computer system the OS provides a uniform logical view of information storage. The OS abstracts from the physical properties of its storage devices to define a logical storage unit the file. A file is collection of related information defined by its creator. The OS is responsible for the following activities of file management.

- Creating & deleting files.
- Creating & deleting directories.
- Supporting primitives for manipulating files & directories.
- Mapping files into secondary storage.
- Backing up files on non-volatile media.

### **4. I/O System Management:**

One of the purposes of an OS is to hide the peculiarities of specific hardware devices from the user. For example, in UNIX the peculiarities of I/O devices are hidden from the bulk of the OS itself by the I/O subsystem. The I/O subsystem consists of:

- A memory management component that includes buffering, catching & spooling.
- A general device- driver interfaces drivers for specific hardware devices. Only the device driver knows the peculiarities of the specific device to which it is assigned.

## **5. Secondary Storage Management:**

The main purpose of computer system is to execute programs. These programs with the data they access must be in main memory during execution. As the main memory is too small to accommodate all data & programs & because the data that it holds are lost when power is lost. The computer system must provide secondary storage to back-up main memory. Most modern computer systems use disks as the storage medium to store data & program. The operating system is responsible for the following activities of disk management.

- Free space management.
- Storage allocation.
- Disk scheduling

Because secondary storage is used frequently it must be used efficiently.

## **Networking:**

A distributed system is a collection of processors that don't share memory peripheral devices or a clock. Each processor has its own local memory & clock and the processors communicate with one another through various communication lines such as high speed buses or networks. The processors in the system are connected through communication networks which are configured in a number of different ways. The communication network design must consider message routing & connection strategies are the problems of connection & security.

## **Protection or security:**

If a computer system has multi users & allows the concurrent execution of multiple processes then the various processes must be protected from one another's activities. For that purpose, mechanisms ensure that files, memory segments, CPU & other resources can be operated on by only those processes that have gained proper authorization from the OS.

## **Command interpretation:**

One of the most important functions of the OS is command interpretation where it acts as the interface between the user & the OS.

## **System Calls:**

System calls provide the interface between a process & the OS. These are usually available in the form of assembly language instruction. Some systems allow system calls to be made directly from a high level language program like C, BCPL and PERL etc. System calls occur in different ways depending on the computer in use. System calls can be roughly grouped into 5 major categories.

## 1. Process Control:

- **End, abort:** A running program needs to be able to have its execution either normally (end) or abnormally (abort).
- **Load, execute:** A process or job executing one program may want to load and execute another program.
- **Create Process, terminate process:** There is a system call specifying for the purpose of creating a new process or job (create process or submit job). We may want to terminate a job or process that we created (terminates process, if we find that it is incorrect or no longer needed).
- **Get process attributes, set process attributes:** If we create a new job or process we should be able to control its execution. This control requires the ability to determine & reset the attributes of a job or processes (get process attributes, set process attributes).
- **Wait time:** After creating new jobs or processes, we may need to wait for them to finish their execution (wait time).
- **Wait event, signal event:** We may wait for a specific event to occur (wait event). The jobs or processes then signal when that event has occurred (signal event).

## 2. File Manipulation:

- **Create file, delete file:** We first need to be able to create & delete files. Both the system calls require the name of the file & some of its attributes.
- **Open file, close file:** Once the file is created, we need to open it & use it. We close the file when we are no longer using it.
- **Read, write, reposition file:** After opening, we may also read, write or reposition the file (rewind or skip to the end of the file).
- **Get file attributes, set file attributes:** For either files or directories, we need to be able to determine the values of various attributes & reset them if necessary. Two system calls get file attribute & set file attributes are required for their purpose.

## 3. Device Management:

- **Request device, release device:** If there are multiple users of the system, we first request the device. After we finished with the device, we must release it.
- **Read, write, reposition:** Once the device has been requested & allocated to us, we can read, write & reposition the device.

#### **4. Information maintenance:**

- **Get time or date, set time or date:** Most systems have a system call to return the current date & time or set the current date & time.
- **Get system data, set system data:** Other system calls may return information about the system like number of current users, version number of OS, amount of free memory etc.
- **Get process attributes, set process attributes:** The OS keeps information about all its processes & there are system calls to access this information.

#### **5. Communication:** There are two modes of communication such as:

- **Message passing model:** Information is exchanged through an inter process communication facility provided by operating system. Each computer in a network has a name by which it is known. Similarly, each process has a process name which is translated to an equivalent identifier by which the OS can refer to it. The get host id and get processed systems calls to do this translation. These identifiers are then passed to the general purpose open & close calls provided by the file system or to specific open connection system call. The recipient process must give its permission for communication to take place with an accept connection call. The source of the communication known as client & receiver known as server exchange messages by read message & write message system calls. The closeconnection call terminates the connection.
- **Shared memory model:** processes use map memory system calls to access regions of memory owned by other processes. They exchange information by reading & writing data in the shared areas. The processes ensure that they are not writing to the same location simultaneously.

### **SYSTEM PROGRAMS:**

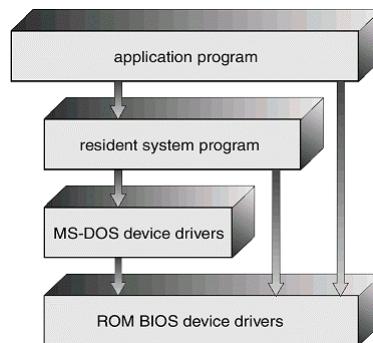
System programs provide a convenient environment for program development & execution. They are divided into the following categories.

- **File manipulation:** These programs create, delete, copy, rename, print & manipulate files and directories.
- **Status information:** Some programs ask the system for date, time & amount of available memory or disk space, no. of users or similar status information.
- **File modification:** Several text editors are available to create and modify the contents of file stored on disk.

- **Programming language support:** compilers, assemblers & interpreters are provided to the user with the OS.
- **Programming loading and execution:** Once a program is assembled or compiled, it must be loaded into memory to be executed.
- **Communications:** These programs provide the mechanism for creating virtual connections among processes users 2<sup>nd</sup> different computer systems.
- **Application programs:** Most OS are supplied with programs that are useful to solve common problems or perform common operations. Ex: web browsers, word processors & text formatters etc.

### System structure:

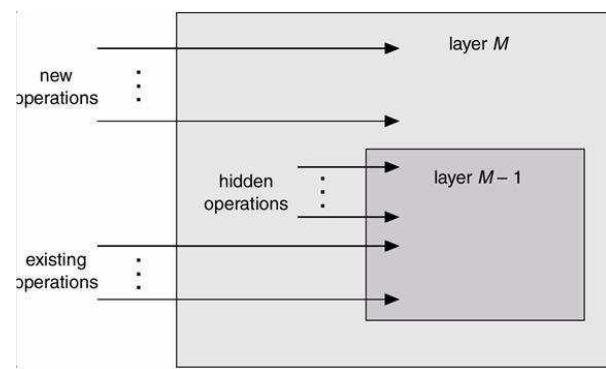
1. **Simple structure:** There are several commercial system that don't have a well-defined structure such operating systems begins as small, simple & limited systems and then grow beyond their original scope. MS-DOS is an example of such system. It was not divided into modules carefully. Another example of limited structuring is the UNIX operating system.



(MS DOS Structure)

2. **Layered approach:** In the layered approach, the OS is broken into a number of layers (levels) each built on top of lower layers. The bottom layer (layer 0) is the hardware & top most layer (layer N) is the user interface. The main advantage of the layered approach is modularity.

- The layers are selected such that each users functions (or operations) & services of only lower layer.



- This approach simplifies debugging & system verification, i.e. the first layer can be debugged without concerning the rest of the system. Once the first layer is debugged, its correct functioning is assumed while the 2<sup>nd</sup> layer is debugged & so on.
- If an error is found during the debugging of a particular layer, the error must be on that layer because the layers below it are already debugged. Thus the design & implementation of the system are simplified when the system is broken down into layers.
- Each layer is implemented using only operations provided by lower layers. A layer doesn't need to know how these operations are implemented; it only needs to know what these operations do.
- The layer approach was first used in the operating system. It was defined in six layers.

<b>Layers</b>	<b>Functions</b>
5	User Program
4	I/O Management
3	Operator Process Communication
2	Memory Management
1	CPU Scheduling
0	Hardware

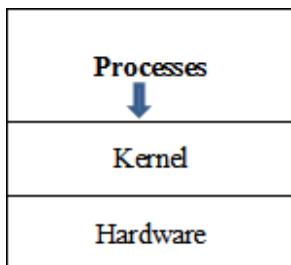
The main disadvantage of the layered approach is:

- The main difficulty with this approach involves the careful definition of the layers, because a layer can use only those layers below it. For example, the device driver for the disk space used by virtual memory algorithm must be at a level lower than that of the memory management routines, because memory management requires the ability to use the disk space.
- It is less efficient than a non layered system (Each layer adds overhead to the system call & the net result is a system call that take longer time than on a non layered system).

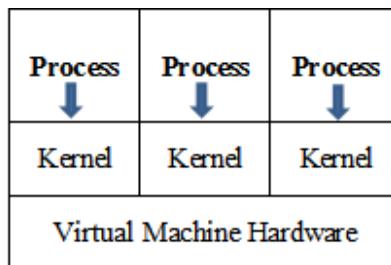
### **Virtual Machines:**

By using CPU scheduling & virtual memory techniques an operating system can create the illusion of multiple processes, each executing on its own processors & own virtual memory. Each processor is provided a virtual copy of the underlying computer. The resources of the computer are shared to

create the virtual machines. CPU scheduling can be used to create the appearance that users have their own processor.



(Non virtual Machine)



(Virtual Machine)

**Implementation:** Although the virtual machine concept is useful, it is difficult to implement since much effort is required to provide an exact duplicate of the underlying machine. The CPU is being multiprogrammed among several virtual machines, which slows down the virtual machines in various ways.

**Difficulty:** A major difficulty with this approach is regarding the disk system. The solution is to provide virtual disks, which are identical in all respects except size. These are known as mini disks in IBM's VM OS. The sum of sizes of all mini disks should be less than the actual amount of physical disk space available.

## I/O Structure

A general purpose computer system consists of a CPU and multiple device controller which is connected through a common bus. Each device controller is in charge of a specific type of device. A device controller maintains some buffer storage and a set of special purpose register. The device controller is responsible for moving the data between peripheral devices and buffer storage.

**I/O Interrupt:** To start an I/O operation the CPU loads the appropriate register within the device controller. In turn the device controller examines the content of the register to determine the actions which will be taken. For example, suppose the device controller finds the read request then, the controller will start the transfer of data from the device to the buffer. Once the transfer of data is complete the device controller informs the CPU that the operation has been finished. Once the I/O is started, two actions are possible such as

- In the simplest case the I/O is started then at I/O completion control is return to the user process. This is known as synchronous I/O.

- The other possibility is asynchronous I/O in which the control is returned to the user program without waiting for the I/O completion. The I/O then continues with other operations.

When an interrupt occurs first determine which I/O device is responsible for interrupting. After searching the I/O device table the signal goes to the each I/O request. If there are additional requests waiting in the queue for one device the operating system starts processing the next request. Finally control is returned from the I/O interrupt.

**DMA controller:** DMA is used for high speed I/O devices. In DMA access the device controller transfers an entire block of data to or from its own buffer storage to memory. In this access the interrupt is generated per block rather than one interrupt per byte. The operating system finds a buffer from the pool of buffers for the transfer. Then a portion of the operating system called a device driver sets the DMA controller registers to use appropriate source and destination addresses and transfer length. The DMA controller is then instructed to start the I/O operation. While the DMA controller is performing the data transfer, the CPU is free to perform other tasks. Since the memory generally can transfer only one word at a time, the DMA controller steals memory cycles from the CPU. This cycle stealing can slow down the CPU execution while a DMA transfer is in progress. The DMA controller interrupts the CPU when the transfer has been completed.

## Storage Structure

The storage structure of a computer system consists of two types of memory such as

- Main memory
- Secondary memory

Basically the programs & data are resided in main memory during the execution. The programs and data are not stored permanently due to following two reasons.

- Main memory is too small to store all needed programs and data permanently.
- Main memory is a volatile storage device which loses its contents when power is turned off.

**Main Memory:** The main memory and the registers are the only storage area that the CPU can access the data directly without any help of other device. The machine instruction which take memory address as arguments do not take disk address. Therefore in execution any instructions and any data must be resided in any one of direct access storage device. If the data are not in memory they must be moved before the CPU can operate on them. There are two types of main memory such as:

- **RAM (Random Access Memory):** The RAM is implemented in a semiconductor technology called D-RAM (Dynamic RAM) which forms an array of memory words/cells. Each & every word should have its own address/locator. Instruction is performed through a sequence of load and store instruction to specific memory address. Each I/O controller includes register to hold commands of the data being transferred. To allow more convenient access to I/O device many computer architecture provide memory mapped I/O. In the case of memory mapped I/O ranges of memory address are mapped to the device register. Read and write to this memory address because the data to be transferred to and from the device register.

**Secondary Storage:** The most common secondary storage devices are magnetic disk and magnetic tape which provide permanent storage of programs and data.

**Magnetic Disk:** It provides the bulk of secondary storage for modern computer systems. Each disk platter has flat circular shape like a CD. The diameter of a platter range starts from 1.8 to 5.25 inches. The two surfaces of a platter are covered with a magnetic material which records the information/data given by the user. The read, write head are attached to a disk arm, which moves all the heads as a unit. The surface of a platter is logically divided into circular tracks which are subdivided into sectors. The set of tracks which are at one arm position forms a cylinder. There are may be thousands of cylinders in a disk drive & each track contains 100 of sectors. The storage capacity of a common disk drive is measured in GB. When the disk is in use a drive motor spins it at high speed. Most drives rotated 62 to 200 time/sec. The disk speed has two parts such as transfer rate & positioning time. The transfer rate is the rate at which data flow between the drive & the computer. The positioning time otherwise called as random access time. It consists of two parts such as seek time & rotational latency. The seek time is the time taken to move the disk arc to the desired cylinder. The rotational latency is the time taken to rotate the disk head.

**Magnetic Tape:** It was used as early secondary storage medium. It is also permanent and can hold large quantity of data. Its access time is slower, comparison to main memory devices. Magnetic tapes are sequential in nature. That's why random access to magnetic tape is thousand times slower than the random access to magnetic disk. The magnetic tapes are used mainly for backup the data. The magnetic tape must be kept in a non-dusty environment and temperature controlled area. But the main advantage of the secondary storage device is that it can hold 2 to 3 times more data than a large disk drive. There are 4 types of magnetic tapes such as:

- ½ Inch

- ¼ Inch
- 4 mm
- 8 mm

## Operating System Services

An operating system provides an environment for the execution of the program. It provides some services to the programs. The various services provided by an operating system are as follows:

- **Program Execution:** The system must be able to load a program into memory and to run that program. The program must be able to terminate this execution either normally or abnormally.
- **I/O Operation:** A running program may require I/O. This I/O may involve a file or a I/O device for specific device. Some special function can be desired. Therefore the operating system must provide a means to do I/O.
- **File System Manipulation:** The programs need to create and delete files by name and read and write files. Therefore the operating system must maintain each and every files correctly.
- **Communication:** The communication is implemented via shared memory or by the technique of message passing in which packets of information are moved between the processes by the operating system.
- **Error detection:** The operating system should take the appropriate actions for the occurrences of any type like arithmetic overflow, access to the illegal memory location and too large user CPU time.
- **Resource Allocation:** When multiple users are logged on to the system the resources must be allocated to each of them. For current distribution of the resource among the various processes the operating system uses the CPU scheduling run times which determine which process will be allocated with the resource.
- **Accounting:** The operating system keep track of which users use how many and which kind of computer resources.
- **Protection:** The operating system is responsible for both hardware as well as software protection. The operating system protects the information stored in a multiuser computer system.

## Process Management:

**Process:** A process or task is an instance of a program in execution. The execution of a process must programs in a sequential manner. At any time at most one instruction is executed. The process includes the current activity as represented by the value of the program counter and the content of the processor's registers. Also it includes the process stack which contains temporary data (such as method parameters return address and local variables) & a data section which contains global variables.

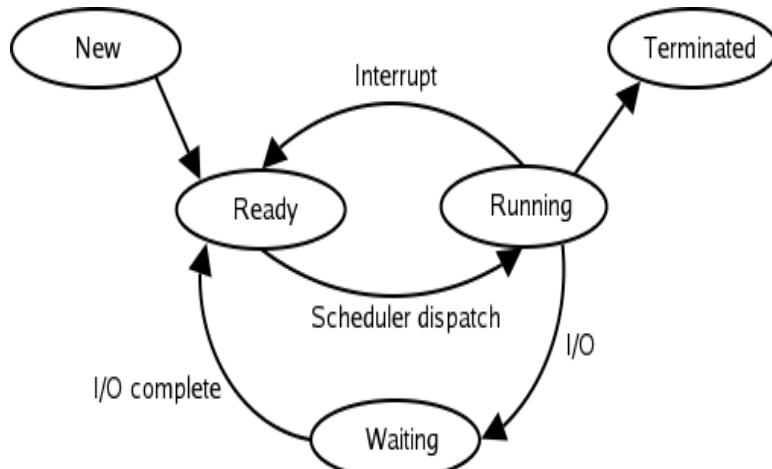
### Difference between process & program:

A program by itself is not a process. A program in execution is known as a process. A program is a passive entity, such as the contents of a file stored on disk whereas process is an active entity with a program counter specifying the next instruction to execute and a set of associated resources may be shared among several processes with some scheduling algorithm being used to determine when the system stops work on one process and services a different one.

**Process state:** As a process executes, it changes state. The state of a process is defined by the correct activity of that process. Each process may be in one of the following states.

- **New:** The process is being created.
- **Ready:** The process is waiting to be assigned to a processor.
- **Running:** Instructions are being executed.
- **Waiting:** The process is waiting for some event to occur.
- **Terminated:** The process has finished execution.

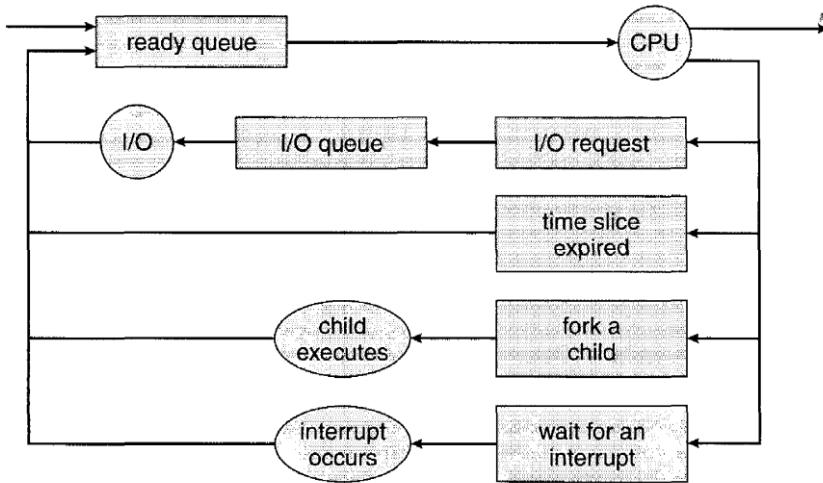
Many processes may be in ready and waiting state at the same time. But only one process can be running on any processor at any instant.



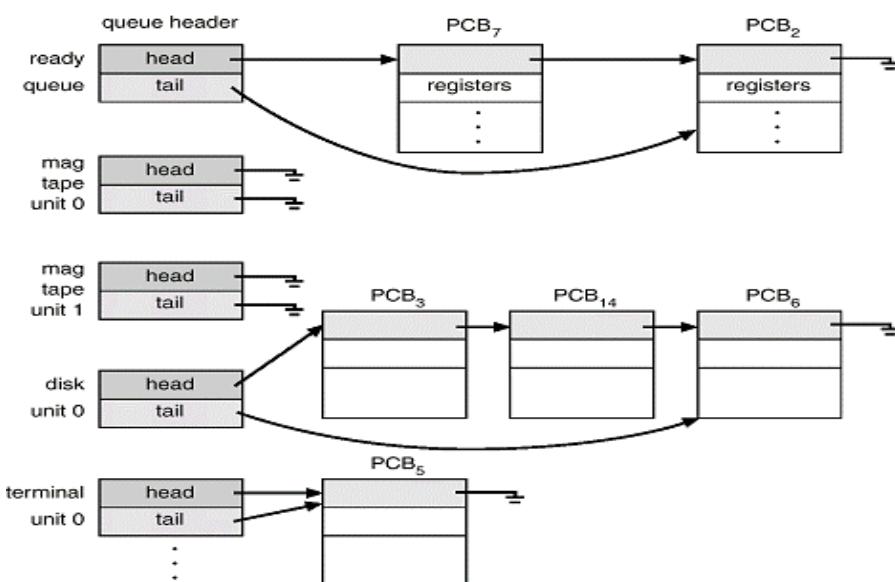
### Process scheduling:

Scheduling is a fundamental function of OS. When a computer is multiprogrammed, it has multiple processes completing for the CPU at the same time. If only one CPU is available, then a choice has to be made regarding which process to execute next. This decision making process is known as scheduling and the part of the OS that makes this choice is called a scheduler. The algorithm it uses in making this choice is called scheduling algorithm.

**Scheduling queues:** As processes enter the system, they are put into a job queue. This queue consists of all process in the system. The process that are residing in main memory and are ready & waiting to execute or kept on a list called ready queue.



This queue is generally stored as a linked list. A ready queue header contains pointers to the first & final PCB in the list. The PCB includes a pointer field that points to the next PCB in the ready queue. The lists of processes waiting for a particular I/O device are kept on a list called device queue. Each device has its own device queue. A new process is initially put in the ready queue. It waits in the ready queue until it is selected for execution & is given the CPU.



### SCHEDULERS:

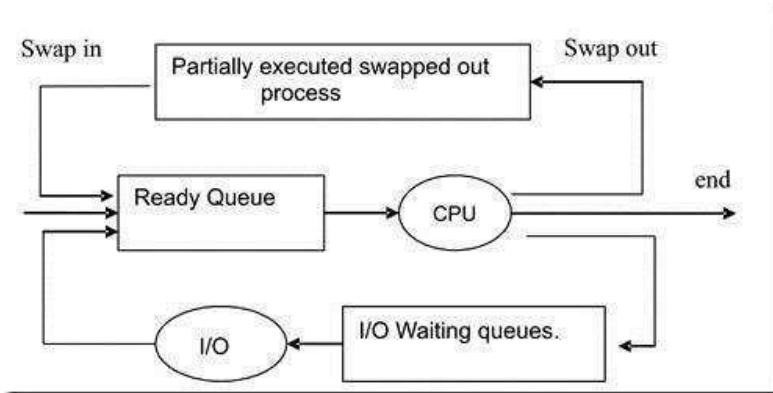
A process migrates between the various scheduling queues throughout its life-time purposes. The OS must select for scheduling processes from these queues in some fashion. This selection process is carried out by the appropriate scheduler. In a batch system, more processes are submitted and then executed immediately. So these processes are spooled to a mass storage device like disk, where they are kept for later execution.

### Types of schedulers:

There are 3 types of schedulers mainly used:

- Long term scheduler:** Long term scheduler selects process from the disk & loads them into memory for execution. It controls the degree of multi-programming i.e. no. of processes in memory. It executes less frequently than other schedulers. If the degree of multiprogramming is stable than the average rate of process creation is equal to the average departure rate of processes leaving the system. So, the long term scheduler is needed to be invoked only when a process leaves the system. Due to longer intervals between executions it can afford to take more time to decide which process should be selected for execution. Most processes in the CPU are either I/O bound or CPU bound. An I/O bound process (an interactive ‘C’ program) is one that spends most of its time in I/O operation than it spends in doing I/O operation. A CPU bound process is one that spends more of its time in doing computations than I/O operations (complex sorting program). It is important that the long term scheduler should select a good mix of I/O bound & CPU bound processes.

2. **Short - term scheduler:** The short term scheduler selects among the process that are ready to execute & allocates the CPU to one of them. The primary distinction between these two schedulers is the frequency of their execution. The short-term scheduler must select a new process for the CPU quite frequently. It must execute at least one in 100ms. Due to the short duration of time between executions, it must be very fast.
3. **Medium - term scheduler:** some operating systems introduce an additional intermediate level of scheduling known as medium - term scheduler. The main idea behind this scheduler is that sometimes it is advantageous to remove processes from memory & thus reduce the degree of multiprogramming. At some later time, the process can be reintroduced into memory & its execution can be continued from where it had left off. This is called as swapping. The process is swapped out & swapped in later by medium term scheduler. Swapping is necessary to improve the process miss or due to some change in memory requirements, the available memory limit is exceeded which requires some memory to be freed up.



### **Process control block:**

Each process is represented in the OS by a process control block. It is also by a process control block. It is also known as task control block.

<b>pointer</b>	<b>process state</b>
	<b>process number</b>
	<b>program counter</b>
<b>registers</b>	
	<b>memory limits</b>
	<b>list of open files</b>
⋮	

A process control block contains many pieces of information associated with a specific process. It includes the following information.

- **Process state:** The state may be new, ready, running, waiting or terminated state.
- **Program counter:** it indicates the address of the next instruction to be executed for this purpose.
- **CPU registers:** The registers vary in number & type depending on the computer architecture. It includes accumulators, index registers, stack pointer & general purpose registers, plus any condition- code information must be saved when an interrupt occurs to allow the process to be continued correctly after- ward.
- **CPU scheduling information:** This information includes process priority pointers to scheduling queues & any other scheduling parameters.
- **Memory management information:** This information may include such information as the value of the bar & limit registers, the page tables or the segment tables, depending upon the memory system used by the operating system.
- **Accounting information:** This information includes the amount of CPU and real time used, time limits, account number, job or process numbers and so on.
- **I/O Status Information:** This information includes the list of I/O devices allocated to this process, a list of open files and so on. The PCB simply serves as the repository for any information that may vary from process to process.

### **CPU Scheduling Algorithm:**

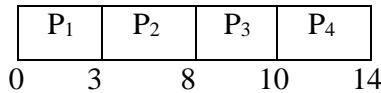
CPU Scheduling deals with the problem of deciding which of the processes in the ready queue is to be allocated first to the CPU. There are four types of CPU scheduling that exist.

1. **First Come, First Served Scheduling (FCFS) Algorithm:** This is the simplest CPU scheduling algorithm. In this scheme, the process which requests the CPU first, that is allocated to the CPU first. The implementation of the FCFS algorithm is easily managed with a FIFO queue. When a process enters the ready queue its PCB is linked onto the rear of the queue. The average waiting time under FCFS policy is quite long. Consider the following example:

<b>Process</b>	<b>CPU time</b>
P <sub>1</sub>	3
P <sub>2</sub>	5
P <sub>3</sub>	2
P <sub>4</sub>	4

Using FCFS algorithm find the average waiting time and average turnaround time if the order is P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub>.

**Solution:** If the process arrived in the order P<sub>1</sub>, P<sub>2</sub>, P<sub>3</sub>, P<sub>4</sub> then according to the FCFS the Gantt chart will be:



The waiting time for process P<sub>1</sub> = 0, P<sub>2</sub> = 3, P<sub>3</sub> = 8, P<sub>4</sub> = 10 then the turnaround time for process P<sub>1</sub> = 0 + 3 = 3, P<sub>2</sub> = 3 + 5 = 8, P<sub>3</sub> = 8 + 2 = 10, P<sub>4</sub> = 10 + 4 = 14.

Then average waiting time =  $(0 + 3 + 8 + 10)/4 = 21/4 = 5.25$

Average turnaround time =  $(3 + 8 + 10 + 14)/4 = 35/4 = 8.75$

The FCFS algorithm is non preemptive means once the CPU has been allocated to a process then the process keeps the CPU until it releases the CPU either by terminating or requesting I/O.

2. **Shortest Job First Scheduling (SJF) Algorithm:** This algorithm associates with each process if the CPU is available. This scheduling is also known as shortest next CPU burst, because the scheduling is done by examining the length of the next CPU burst of the process rather than its total length. Consider the following example:

<b>Process</b>	<b>CPU time</b>
P <sub>1</sub>	3
P <sub>2</sub>	5
P <sub>3</sub>	2
P <sub>4</sub>	4

**Solution:** According to the SJF the Gantt chart will be

P <sub>3</sub>	P <sub>1</sub>	P <sub>2</sub>	P <sub>4</sub>
0	2	5	9

14

The waiting time for process P<sub>1</sub> = 0, P<sub>2</sub> = 2, P<sub>3</sub> = 5, P<sub>4</sub> = 9 then the turnaround time for process P<sub>3</sub> = 0 + 2 = 2, P<sub>1</sub> = 2 + 3 = 5, P<sub>4</sub> = 5 + 4 = 9, P<sub>2</sub> = 9 + 5 = 14.

Then average waiting time =  $(0 + 2 + 5 + 9)/4 = 16/4 = 4$  Average

turnaround time =  $(2 + 5 + 9 + 14)/4 = 30/4 = 7.5$

The SJF algorithm may be either preemptive or non-preemptive algorithm. The preemptive SJF is also known as shortest remaining time first.

Consider the following example.

Process	Arrival Time	CPU time
P <sub>1</sub>	0	8
P <sub>2</sub>	1	4
P <sub>3</sub>	2	9
P <sub>4</sub>	3	5

In this case the Gantt chart will be

P <sub>1</sub>	P <sub>2</sub>	P <sub>4</sub>	P <sub>1</sub>	P <sub>3</sub>
0	1	5	10	17

26

The waiting time for process

$$P_1 = 10 - 1 = 9$$

$$P_2 = 1 - 1 = 0$$

$$P_3 = 17 - 2 = 15$$

$$P_4 = 5 - 3 = 2$$

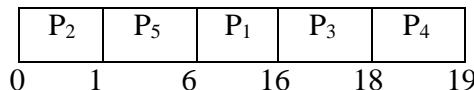
The average waiting time =  $(9 + 0 + 15 + 2)/4 = 26/4 = 6.5$

3. **Priority Scheduling Algorithm:** In this scheduling a priority is associated with each process and the CPU is allocated to the process with the highest priority. Equal priority processes are scheduled in FCFS manner. Consider the following example:

Process	Arrival Time	CPU time
P <sub>1</sub>	10	3
P <sub>2</sub>	1	1
P <sub>3</sub>	2	3

P <sub>4</sub>	1	4
P <sub>5</sub>	5	2

According to the priority scheduling the Gantt chart will be



The waiting time for process

$$P_1 = 6$$

$$P_2 = 0$$

$$P_3 = 16$$

$$P_4 = 18$$

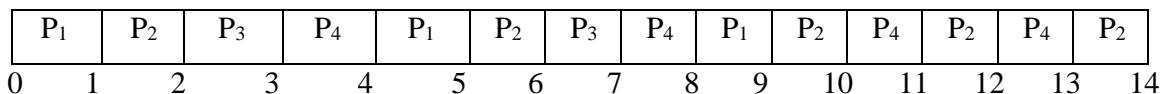
$$P_5 = 1$$

$$\text{The average waiting time} = (0 + 1 + 6 + 16 + 18)/5 = 41/5 = 8.2$$

4. **Round Robin Scheduling Algorithm:** This type of algorithm is designed only for the time sharing system. It is similar to FCFS scheduling with preemption condition to switch between processes. A small unit of time called quantum time or time slice is used to switch between the processes. The average waiting time under the round robin policy is quiet long. Consider the following example:

Process	CPU time
P <sub>1</sub>	3
P <sub>2</sub>	5
P <sub>3</sub>	2
P <sub>4</sub>	4

Time Slice = 1 millisecond.



The waiting time for process

$$P_1 = 0 + (4 - 1) + (8 - 5) = 0 + 3 + 3 = 6$$

$$P_2 = 1 + (5 - 2) + (9 - 6) + (11 - 10) + (12 - 11) + (13 - 12) = 1 + 3 + 3 + 1 + 1 + 1 = 10$$

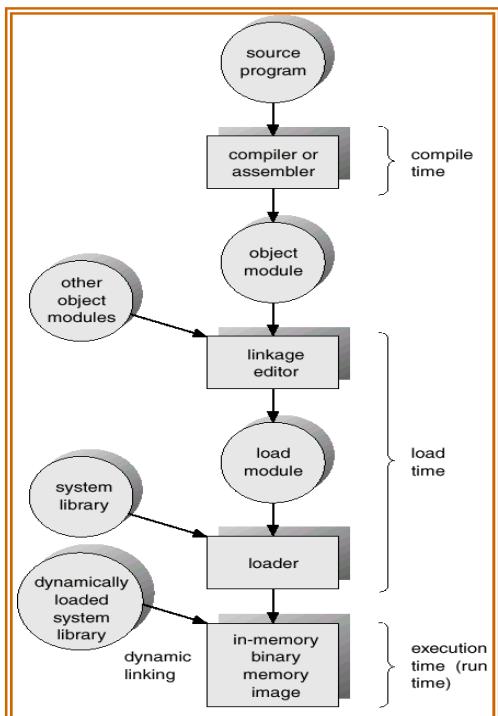
$$P_3 = 2 + (6 - 3) = 2 + 3 = 5$$

$$P_4 = 3 + (7 - 4) + (10 - 8) + (12 - 11) = 3 + 3 + 2 + 1 = 9$$

$$\text{The average waiting time} = (6 + 10 + 5 + 9)/4 = 7.5$$

## **Memory Management**

- Memory consists of a large array of words or bytes, each with its own address. The CPU fetches instructions from memory according to the value of the program counter. These instructions may cause additional loading from and storing to specific memory addresses.
  - Memory unit sees only a stream of memory addresses. It does not know how they are generated.
  - Program must be brought into memory and placed within a process for it to be run.
  - Input queue – collection of processes on the disk that are waiting to be brought into memory for execution.
  - User programs go through several steps before being run.
-



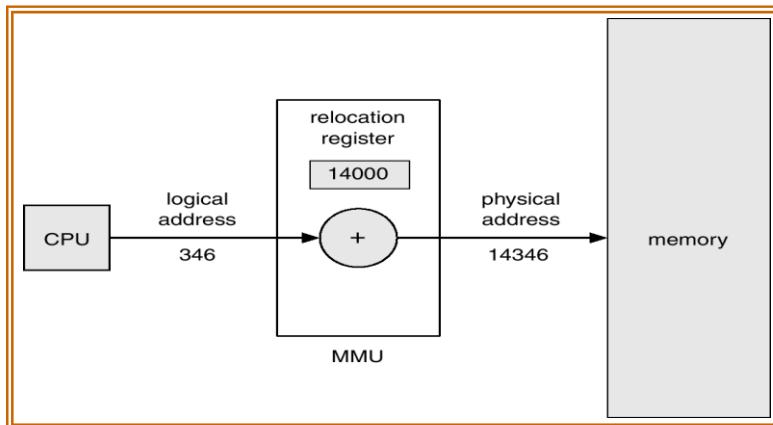
Address binding of instructions and data to memory addresses can happen at three different stages.

- **Compile time:** If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.  
Example: .COM-format programs in MS-DOS.
- **Load time:** Must generate relocatable code if memory location is not known at compile time.
- **Execution time:** Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., relocation registers).

## Logical Versus Physical Address Space

- The concept of a logical address space that is bound to a separate physical address space is central to proper memory management.
  - Logical address – address generated by the CPU; also referred to as virtual address.
  - Physical address – address seen by the memory unit.
- The set of all logical addresses generated by a program is a logical address space; the set of all physical addresses corresponding to these logical addresses are a physical address space.

- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.
- The run-time mapping from virtual to physical addresses is done by a hardware device called the memory management unit (MMU).



- This method requires hardware support slightly different from the hardware configuration. The base register is now called a relocation register. The value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program never sees the real physical addresses. The program can create a pointer to location 346, store it in memory, manipulate it and compare it to other addresses. The user program deals with logical addresses. The memory mapping hardware converts logical addresses into physical addresses. The final location of a referenced memory address is not determined until the reference is made.

## Dynamic Loading

- Routine is not loaded until it is called.
- All routines are kept on disk in a relocatable load format.
- The main program is loaded into memory and is executed. When a routine needs to call another routine, the calling routine first checks to see whether the other the desired routine into memory and to update the program's address tables to reflect this change. Then control is passed to the newly loaded routine.
- Better memory-space utilization; unused routine is never loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases.

- No special support from the operating system is required.
- Implemented through program design.

## **Dynamic Linking**

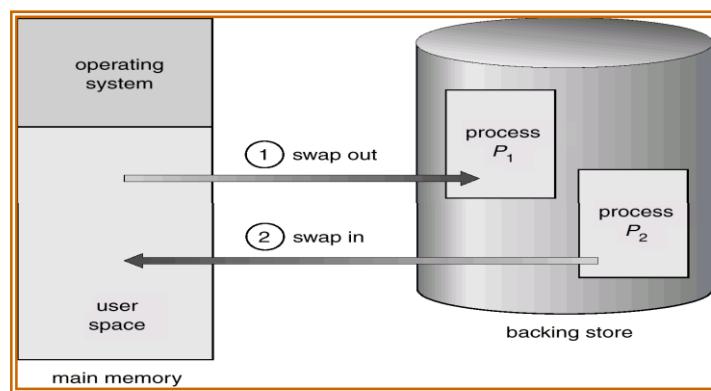
- Linking is postponed until execution time.
- Small piece of code, stub, is used to locate the appropriate memory-resident library routine, or to load the library if the routine is not already present.
- When this stub is executed, it checks to see whether the needed routine is already in memory. If not, the program loads the routine into memory.
- Stub replaces itself with the address of the routine, and executes the routine.
- Thus the next time that code segment is reached, the library routine is executed directly, incurring no cost for dynamic linking.
- Operating system is needed to check if routine is in processes' memory address.
- Dynamic linking is particularly useful for libraries.

## **Swapping**

- A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution. For example, assume a multiprogramming environment with a round robin CPU scheduling algorithm. When a quantum expires, the memory manager will start to swap out the process that just finished, and to swap in another process to the memory space that has been freed. In the meantime, the CPU scheduler will allocate a time slice to some other process in memory. When each process finished its quantum, it will be swapped with another process. Ideally, the memory manager can swap processes fast enough that some processes will be in memory, ready to execute, when the CPU scheduler wants to reschedule the CPU. The quantum must also be sufficiently large that reasonable amounts of computing are done between swaps.
- Roll out, roll in – swapping variant used for priority-based scheduling algorithms. If a higher priority process arrives and wants service, the memory manager can swap out the lower priority process so that it can load and execute lower priority process can be swapped back in and continued. This variant is sometimes called roll out, roll in. Normally a process that is swapped out will be swapped back into the same memory space that it occupied previously. This restriction is dictated by the fact that the process cannot be moved to different locations. If execution time

binding is being used, then a process can be swapped into a different memory space, because the physical addresses are computed during execution time.

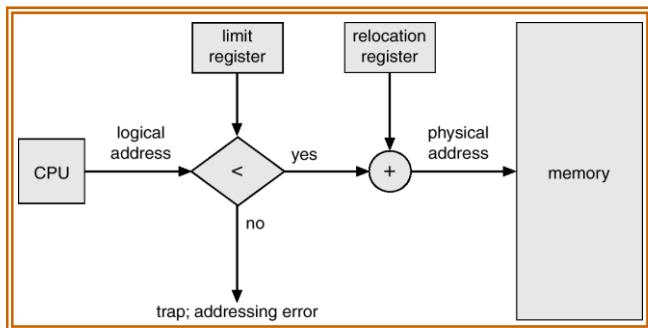
- Backing store – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images. It must be large enough to accommodate copies of all memory images for all users, and it must provide direct access to these memory images. The system maintains a ready queue consisting of all processes whose memory images are in memory. If not, and there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process. It then reloads registers as normal and transfers control to the selected process.
- Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped.
- Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows).



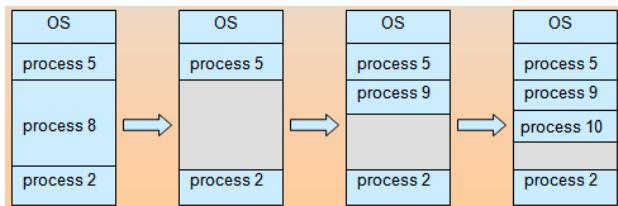
## Contiguous Memory Allocation

- Main memory is usually divided into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes, held in high memory.
- In contiguous memory allocation, each process is contained in a single contiguous section of memory.
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.

- Relocation register contains value of smallest physical address; limit register contains range of logical addresses – each logical address must be less than the limit register.



- Multiple-partition allocation
  - Hole – block of available memory; holes of various size are scattered throughout memory.
  - When a process arrives, it is allocated memory from a hole large enough to accommodate it.
  - Operating system maintains information about:
    - a) allocated partitions b) free partitions (hole)
    - A set of holes of various sizes, is scattered throughout memory at any given time. When a process arrives and needs memory, the system searches this set for a hole that is large enough for this process. If the hole is too large, it is split into two: one part is allocated to the arriving process; the other is returned to the set of holes. When a process terminates, it releases its block of memory, which is then placed back in the set of holes. If the new hole is adjacent to other holes, these adjacent holes are merged to form one larger hole.
    - This procedure is a particular instance of the general dynamic storage allocation problem, which is how to satisfy a request of size n from a list of free holes. There are many solutions to this problem. The set of holes is searched to determine which hole is best to allocate. The first-fit, best-fit and worst-fit strategies are the most common ones used to select a free hole from the set of available holes.



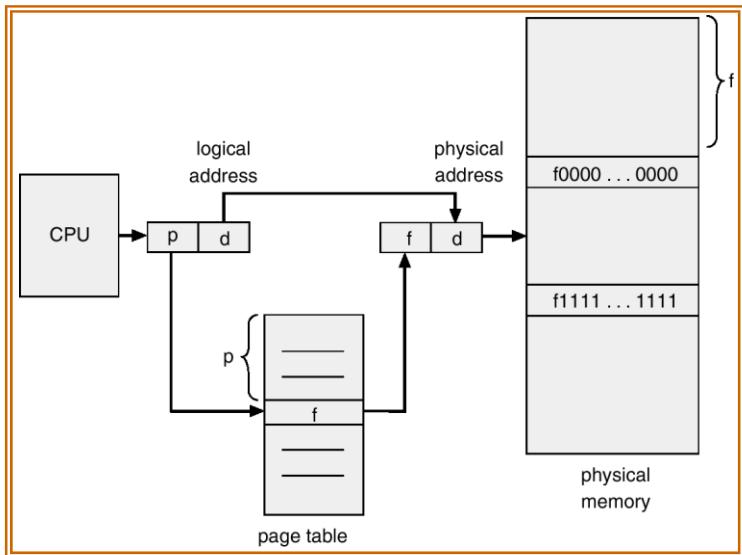
- **First-fit:** Allocate the first hole that is big enough.
- **Best-fit:** Allocate the smallest hole that is big enough; must search entire list, unless ordered by size.
- **Worst-fit:** Allocate the largest hole; must also search entire list.

## Fragmentation

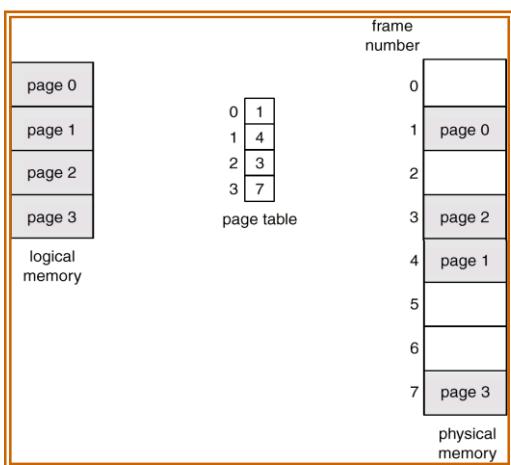
- **External Fragmentation** – total memory space exists to satisfy a request, but it is not contiguous.
- **Internal Fragmentation** – allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.
- Reduce external fragmentation by compaction
  - Shuffle memory contents to place all free memory together in one large block.
  - Compaction is possible only if relocation is dynamic, and is done at execution time.

## Paging

- Paging is a memory management scheme that permits the physical address space of a process to be non contiguous.
- Divide physical memory into fixed-sized blocks called **frames** (size is power of 2, for example 512 bytes).
- Divide logical memory into blocks of same size called **pages**. When a process is to be executed, its pages are loaded into any available memory frames from the backing store. The backing store is divided into fixed sized blocks that are of the same size as the memory frames.
- The hardware support for paging is illustrated in below figure.
- Every address generated by the CPU is divided into two parts: a page number ( $p$ ) and a page offset ( $d$ ). The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.



- The paging model of memory is shown in below figure. The page size is defined by the hardware. The size of a page is typically of a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture. The selection of a power of 2 as a page size makes the translation of a logical address into a page number and page offset particularly easy. If the size of logical address is  $2^m$ , and a page size is  $2^n$  addressing units, then the high order  $m-n$  bits of a logical address designate the page number, and the  $n$  low order bits designate the page offset.



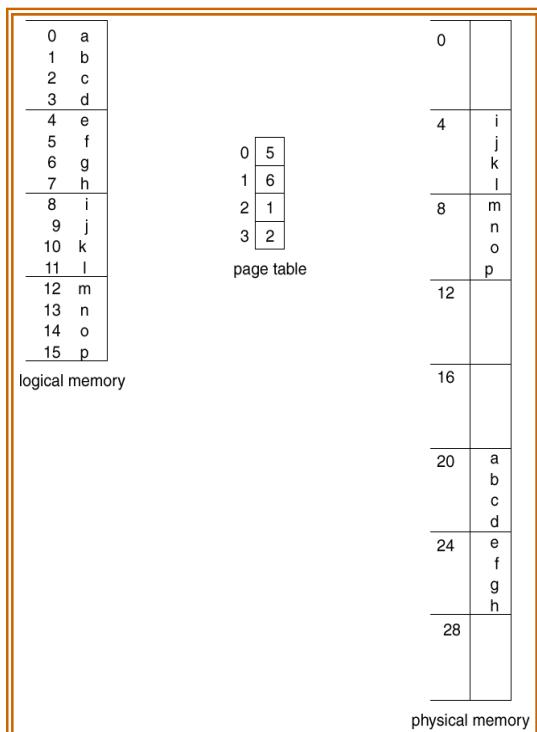
- Keep track of all free frames.
- To run a program of size  $n$  pages, need to find  $n$  free frames and load program.
- Set up a page table to translate logical to physical addresses.
- Internal fragmentation may occur.

Let us take an example. Suppose a program needs 32 KB memory for allocation. The whole program is divided into smaller units assuming 4 KB and is assigned some address. The address consists of two parts such as:

- A large number in higher order positions and
- Displacement or offset in the lower order bits.

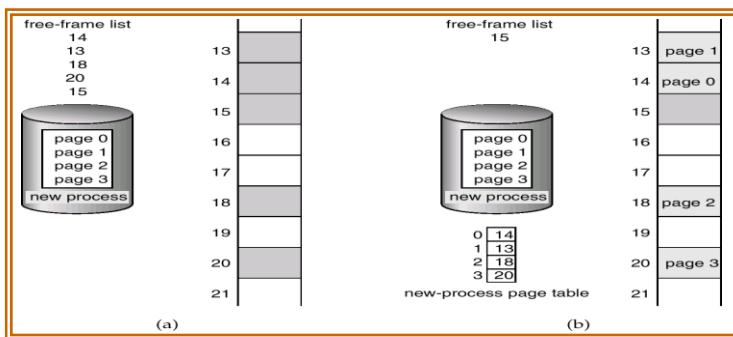
The numbers allocated to pages are typically in power of 2 to simplify extraction of page numbers and offsets. To access a piece of data at a given address, the system first extracts the page number and the offset. Then it translates the page number to physical page frame and access data at offset in physical page frame. At this moment, the translation of the address by the OS is done using a page table. Page table is a linear array indexed by virtual page number which provides the physical page frame that contains the particular page. It employs a lookup process that extracts the page number and the offset. The system in addition checks that the page number is within the address space of process and retrieves the page number in the page table. Physical address will calculated by using the formula.

Physical address = page size of logical memory X frame number + offset



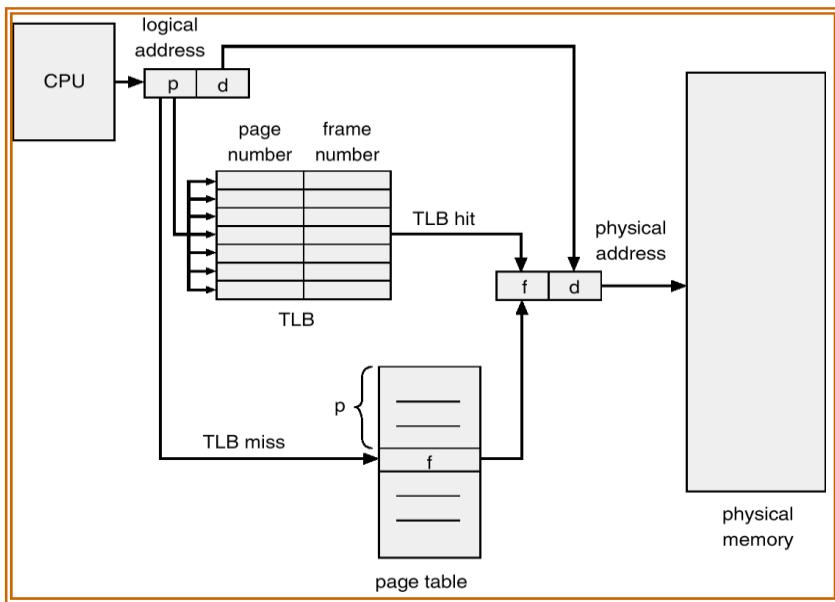
When a process arrives in the system to be executed, its size expressed in pages is examined. Each page of the process needs one frame. Thus if the process requires n pages, at least n frames must be

available in memory. If  $n$  frames are available, they are allocated to this arriving process. The first page of the process is loaded into one of the allocated frames, and the frame number is put in the page table for this process. The next page is loaded into another frame, and its frame number is put into the page table and so on as in below figure. An important aspect of paging is the clear separation between the user's view of memory and the actual physical memory. The user program views that memory as one single contiguous space, containing only this one program. In fact, the user program is scattered throughout physical memory, which also holds other programs. The difference between the user's view of memory and the actual physical memory is reconciled by the address-translation hardware. The logical addresses are translated into physical addresses. This mapping is hidden from the user and is controlled by the operating system.



### Implementation of Page Table

- Page table is kept in main memory.
- Page-tablebase register (PTBR) points to the page table.
- In this scheme every data/instruction-byte access requires two memory accesses. One for the page-table entry and one for the byte.
- The two memory access problem can be solved by the use of a special fast-lookup hardware cache called associative registers or associative memory or translation look-aside buffers(TLBs).
- Typically, the number of entries in a TLB is between 32 and 1024.



- The TLB contains only a few of the page table entries. When a logical address is generated by the CPU, its page number is presented to the TLB. If the page number is found, its frame number is immediately available and is used to access memory. The whole task may take less than 10 percent longer than it would if an unmapped memory reference were used.
- If the page number is not in the TLB (known as a TLB miss), a memory reference to the page table must be made. When the frame number is obtained, we can use it to access memory.

### Hit Ratio

- Hit Ratio: the percentage of times that a page number is found in the associative registers.
- For example, if it takes 20 nanoseconds to search the associative memory and 100 nanoseconds to access memory; for a 98-percent hit ratio, we have

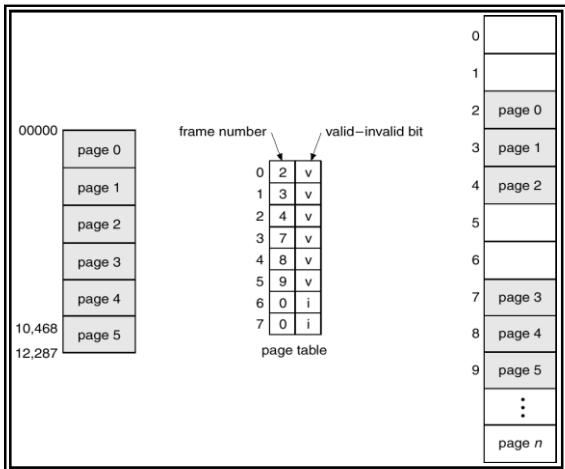
$$\begin{aligned}\text{Effective memory-access time} &= 0.98 \times 120 + 0.02 \times 220 \\ &= 122 \text{ nanoseconds.}\end{aligned}$$

- The Intel 80486 CPU has 32 associative registers, and claims a 98-percent hit ratio.

### Valid or invalid bit in a page table

- Memory protection implemented by associating protection bit with each frame.
- Valid-invalid bit attached to each entry in the page table:
  - “Valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page.
  - “Invalid” indicates that the page is not in the process’ logical address space.

- Pay attention to the following figure. The program extends to only address 10,468, any reference beyond that address is illegal. However, references to page 5 are classified as valid, so accesses to addresses up to 12,287 are valid. This reflects the internal fragmentation of paging.



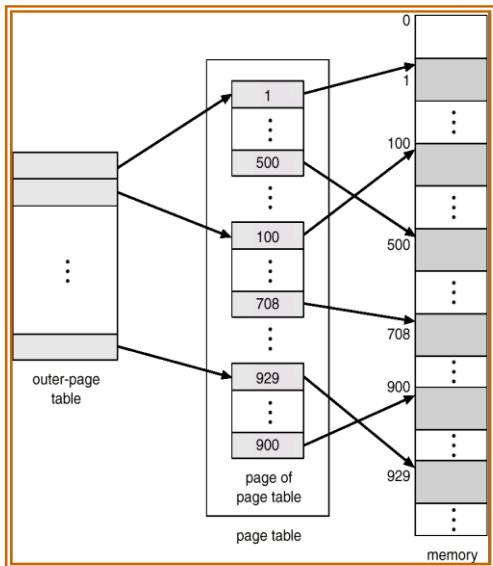
## Structure of the Page Table

### Hierarchical Paging:

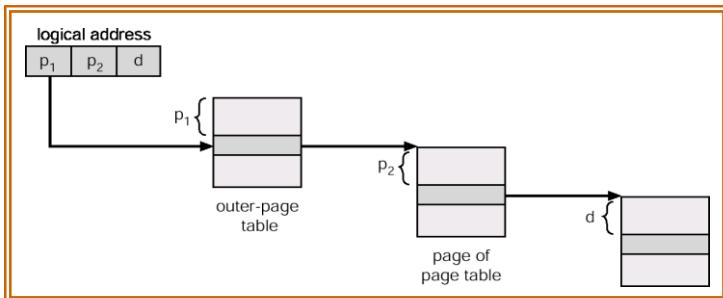
- A logical address (on 32-bit machine with 4K page size) is divided into:
  - A page number consisting of 20 bits.
  - A page offset consisting of 12 bits.
- Since the page table is paged, the page number is further divided into:
  - A 10-bit page number.
  - A 10-bit page offset.
- Thus, a logical address is as follows:

page number	page offset
$p_1$	$p_2$
10	12

Where  $p_1$  is an index into the outer page table, and  $p_2$  is the displacement within the page of the outer page table. The below figure shows a two level page table scheme.

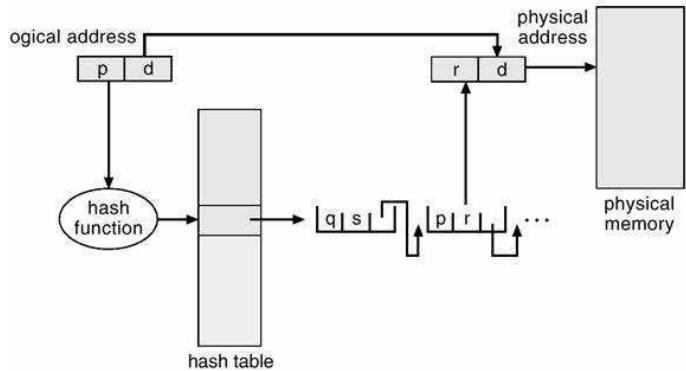


Address-translation scheme for a two-level 32-bit paging architecture is shown in below figure.



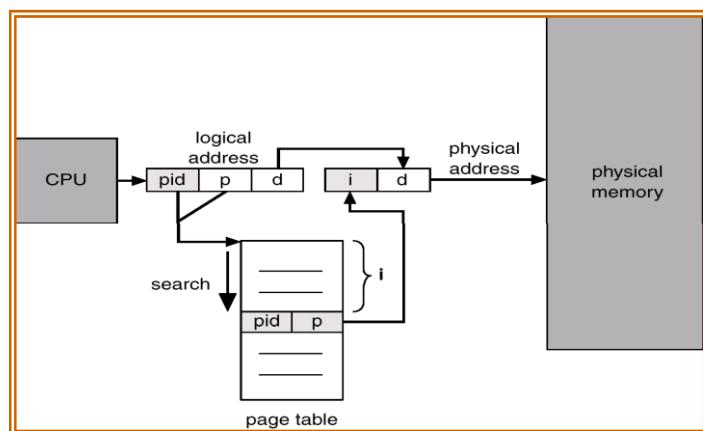
### Hashed Page Table:

A common approach for handling address spaces larger than 32 bits is to use a hashed page table, with the hash value being the virtual page number. Each entry in the hash table contains a linked list of elements that map to the same location. Each element consists of three fields: (a) the virtual page number, (b) the value of the mapped page frame, and (c) a pointer to the next element in the linked list. The algorithm works as follows: The virtual page number in the virtual address is hashed into the hash table. The virtual page number is compared to field (a) in the first element in the linked list. If there is a match, the corresponding page frame (field (b)) is used to form the desired physical address. If there is no match, subsequent entries in the linked list are searched for a matching virtual page number. The scheme is shown in below figure.



### Inverted Page Table:

- One entry for each real page (frame) of memory.
- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page.
- There is only one page table in the system. Not per process.
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs.
- Use hash table to limit the search to one — or at most a few — page-table entries.



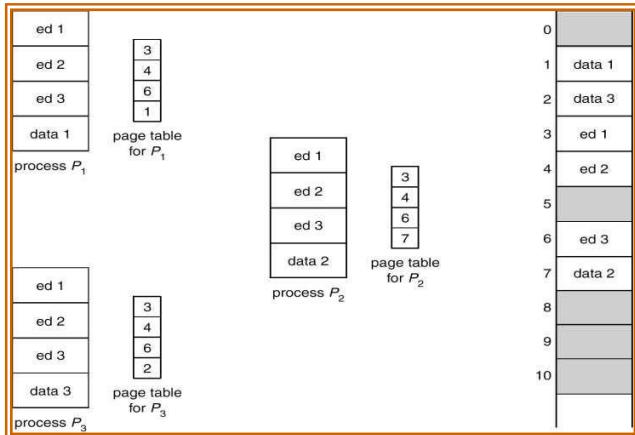
Each virtual address in the system consists of a triple  $\langle$ process-id, page-number, offset $\rangle$ . Each inverted page table entry is a pair  $\langle$ process-id, page-number $\rangle$  where the process-id assumes the role of the address space identifier. When a memory reference occurs, part of the virtual address, consisting of  $\langle$ process-id, page-number $\rangle$ , is presented to the memory subsystem. The inverted page table is then searched for a match. If a match is found say at entry i, then the physical address  $\langle$ i, offset $\rangle$  is generated. If no match is found, then an illegal address access has been attempted.

### Shared Page:

- Shared code

- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
  - Shared code must appear in same location in the logical address space of all processes.
- Private code and data
  - Each process keeps a separate copy of the code and data.
  - The pages for the private code and data can appear anywhere in the logical address space.

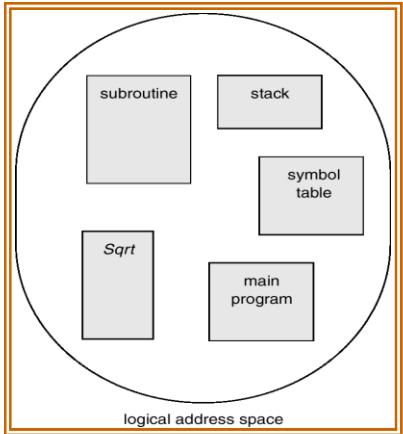
Reentrant code or pure code is non self-modifying code. If the code is reentrant, then it never changes during execution. Thus, two or more processes can execute the same code at the same time. Each process has its own copy of registers and data storage to hold the data for the process' execution. The data for two different processes will of course vary for each process.



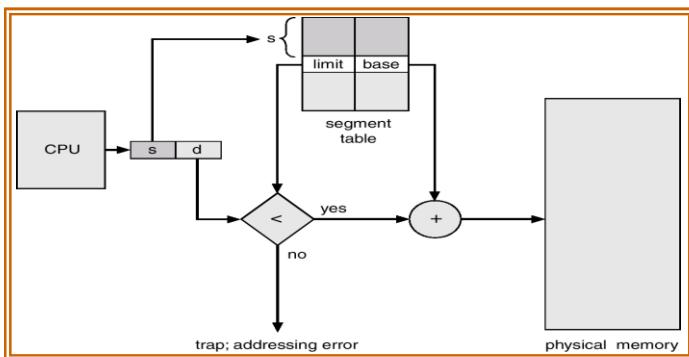
## Segmentation

- Memory-management scheme that supports user view of memory.
- A program is a collection of segments. A segment is a logical unit such as:
  - Main program,
  - Procedure,
  - Function,
  - Method,
  - Object,
  - Local variables, global variables,
  - Common block,
  - Stack,

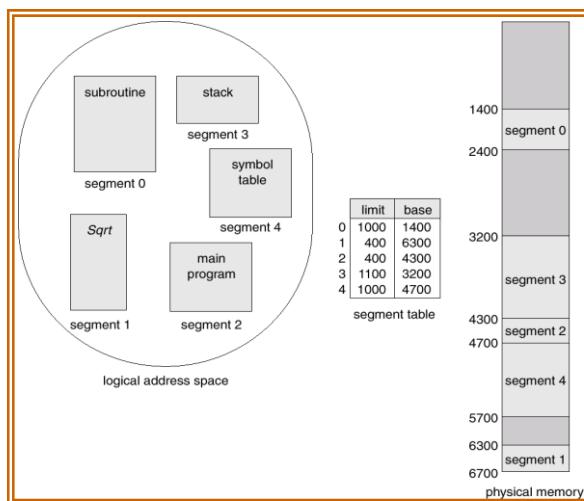
- Symbol table, arrays



- Segmentation is a memory management scheme that supports this user view of memory.
- A logical address space is a collection of segments. Each segment has a name and a length.
- The addresses specify both the segment name and the offset within the segment.
- The user therefore specifies each address by two quantities such as segment name and an offset. For simplicity of implementation, segments are numbered and are referred to by a segment number, rather than by a segment name.
- Logical address consists of a two tuples:
  - <segment-number, offset>
- Segment table – maps two-dimensional physical addresses; each table entry has:
  - Base – contains the starting physical address where the segments reside in memory.
  - Limit – specifies the length of the segment.
- Segment-table base register (STBR) points to the segment table's location in memory.
- Segment-table length register (STLR) indicates number of segments used by a program;
  - Segment number s is legal if  $s < \text{STLR}$ .



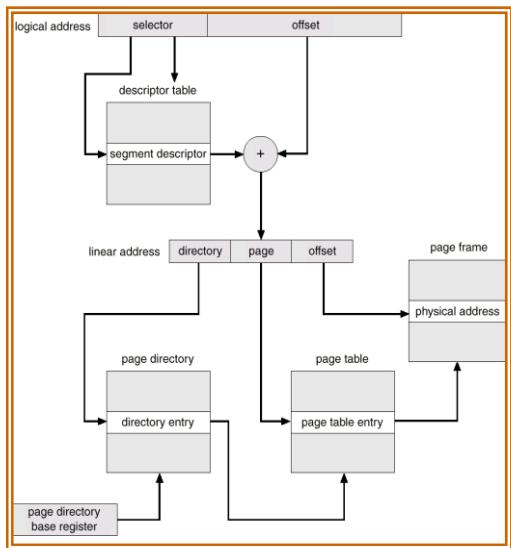
- When the user program is compiled by the compiler it constructs the segments.
- The loader takes all the segments and assigned the segment numbers.
- The mapping between the logical and physical address using the segmentation technique is shown in above figure.
- Each entry in the segment table as limit and base address.
- The base address contains the starting physical address of a segment where the limit address specifies the length of the segment.
- The logical address consists of 2 parts such as segment number and offset.
- The segment number is used as an index into the segment table. Consider the below example is given below.



## Segmentation with Paging

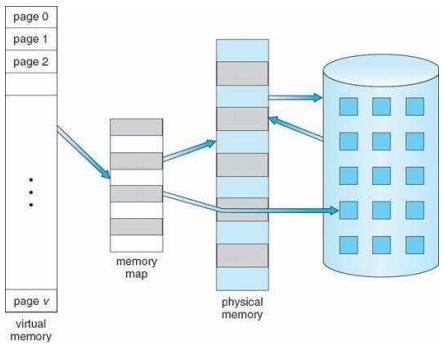
- Both paging and segmentation have advantages and disadvantages, that's why we can combine these two methods to improve this technique for memory allocation.
- These combinations are best illustrated by architecture of Intel-386.
- The IBM OS/2 is an operating system of the Intel-386 architecture. In this technique both segment table and page table is required.
- The program consists of various segments given by the segment table where the segment table contains different entries one for each segment.
- Then each segment is divided into a number of pages of equal size whose information is maintained in a separate page table.

- If a process has four segments that is 0 to 3 then there will be 4 page tables for that process, one for each segment.
- The size fixed in segmentation table (SMT) gives the total number of pages and therefore maximum page number in that segment with starting from 0.
- If the page table or page map table for a segment has entries for page 0 to 5.
- The address of the entry in the PMT for the desired page p in a given segment s can be obtained by  $B + P$  where B can be obtained from the entry in the segmentation table.
- Using the address ( $B + P$ ) as an index in page map table (page table), the page frame (f) can be obtained and physical address can be obtained by adding offset to page frame.



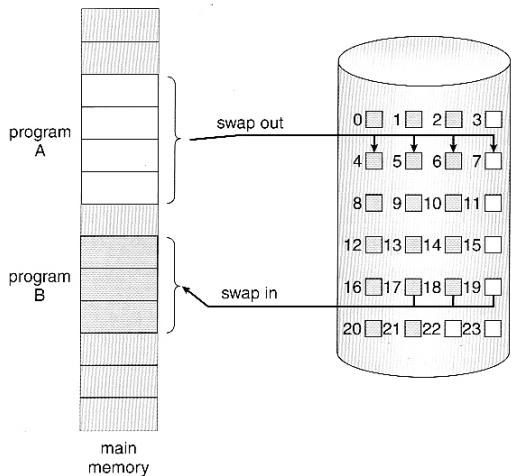
## Virtual Memory

- It is a technique which allows execution of process that may not be compiled within the primary memory.
- It separates the user logical memory from the physical memory. This separation allows an extremely large memory to be provided for program when only a small physical memory is available.
- Virtual memory makes the task of programming much easier because the programmer no longer needs to work about the amount of the physical memory is available or not.
- The virtual memory allows files and memory to be shared by different processes by page sharing.
- It is most commonly implemented by demand paging.

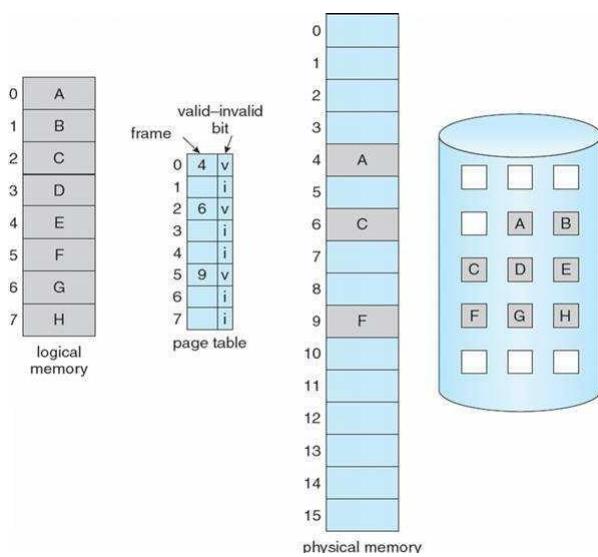


## Demand Paging

A demand paging system is similar to the paging system with swapping feature. When we want to execute a process we swap it into the memory. A swapper manipulates entire process where as a pager is concerned with the individual pages of a process. The demand paging concept is using pager rather than swapper. When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again. Instead of swapping in a whole process, the pager brings only those necessary pages into memory. The transfer of a paged memory to contiguous disk space is shown in below figure.



Thus it avoids reading into memory pages that will not be used any way decreasing the swap time and the amount of physical memory needed. In this technique we need some hardware support to distinguish between the pages that are in memory and those that are on the disk. A valid and invalid bit is used for this purpose. When this bit is set to valid it indicates that the associate page is in memory. If the bit is set to invalid it indicates that the page is either not valid or is valid but currently not in the disk.



Marking a page invalid will have no effect if the process never attempts to access that page. So while a process executes and access pages that are memory resident, execution proceeds normally. Access to a page marked invalid causes a page fault trap. It is the result of the OS's failure to bring the desired page into memory.

### Procedure to handle page fault

If a process refers to a page that is not in physical memory then

- We check an internal table (page table) for this process to determine whether the reference was valid or invalid.
- If the reference was invalid, we terminate the process, if it was valid but not yet brought in, we have to bring that from main memory.
- Now we find a free frame in memory.
- Then we read the desired page into the newly allocated frame.
- When the disk read is complete, we modify the internal table to indicate that the page is now in memory.
- We restart the instruction that was interrupted by the illegal address trap. Now the process can access the page as if it had always been in memory.

### Page Replacement

- Each process is allocated frames (memory) which hold the process's pages (data)
- Frames are filled with pages as needed – this is called demand paging

- Over-allocation of memory is prevented by modifying the page-fault service routine to replace pages
- The job of the page replacement algorithm is to decide which page gets victimized to make room for a new page
- Page replacement completes separation of logical and physical memory

## Page Replacement Algorithm

### Optimal algorithm

- Ideally we want to select an algorithm with the lowest page-fault rate
- Such an algorithm exists, and is called (unsurprisingly) the optimal algorithm:
- Procedure: replace the page that will not be used for the longest time (or at all) – i.e. replace the page with the greatest forward distance in the reference string
- Example using 4 frames:

Reference #	1	2	3	4	5	6	7	8	9	10	11	12
Page referenced	1	2	3	4	1	2	5	1	2	3	4	5
Frames	1	1	1	1	1	1	1	1	1	1	4	4
_ = faulting page				2	2	2	2	2	2	2	2	2
				3	3	3	3	3	3	3	3	3
				4	4	4	5	5	5	5	5	5

- Analysis: 12 page references, 6 page faults, 2 page replacements. Page faults per number of frames =  $6/4 = 1.5$
- Unfortunately, the optimal algorithm requires special hardware (crystal ball, magic mirror, etc.) not typically found on today's computers
- Optimal algorithm is still used as a metric for judging other page replacement algorithms

### FIFO algorithm

- Replaces pages based on their order of arrival: oldest page is replaced
- Example using 4 frames:

<b>Reference #</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>
<b>Page referenced</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>1</b>	<b>2</b>	<b>5</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>Frames</b>	<u>1</u>	1	1	1	1	1	<u>5</u>	5	5	5	4	4
_ = faulting page	<u>2</u>	2	2	2	2	2	<u>1</u>	1	1	1	1	<u>5</u>
	<u>3</u>	3	3	3	3	3	3	<u>2</u>	2	2	2	2
		<u>4</u>	4	4	4	4	4	4	<u>3</u>	3	3	3

- Analysis: 12 page references, 10 page faults, 6 page replacements. Page faults per number of frames =  $10/4 = 2.5$

### LFU algorithm (page-based)

- procedure: replace the page which has been referenced least often
- For each page in the reference string, we need to keep a reference count. All reference counts start at 0 and are incremented every time a page is referenced.
- example using 4 frames:

<b>Reference #</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>	<b>6</b>	<b>7</b>	<b>8</b>	<b>9</b>	<b>10</b>	<b>11</b>	<b>12</b>
<b>Page referenced</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>1</b>	<b>2</b>	<b>5</b>	<b>1</b>	<b>2</b>	<b>3</b>	<b>4</b>	<b>5</b>
<b>Frames</b>	<u>1</u>	1	<u>1</u>	<u>1</u>	<u>1</u>							
_ = faulting page	<u>2</u>	<u>2</u>	<u>2</u>									
n = reference count	<u>1</u>	1	<u>1</u>	<u>1</u>	<u>1</u>							
	<u>3</u>	3	<u>3</u>	<u>3</u>	<u>3</u>							
		<u>4</u>	<u>4</u>	<u>4</u>								

- At the 7th page in the reference string, we need to select a page to be victimized. Either 3 or 4 will do since they have the same reference count (1). Let's pick 3.
- Likewise at the 10th page reference; pages 4 and 5 have been referenced once each. Let's pick page 4 to victimize. Page 3 is brought in, and its reference count (which was 1 before we paged it out a while ago) is updated to 2.
- Analysis: 12 page references, 7 page faults, 3 page replacements. Page faults per number of frames =  $7/4 = 1.75$

### LFU algorithm (frame-based)

- Procedure: replace the page in the frame which has been referenced least often
- Need to keep a reference count for each frame which is initialized to 1 when the page is paged in, incremented every time the page in the frame is referenced, and reset every time the page in the frame is replaced
- Example using 4 frames:

Reference #	1	2	3	4	5	6	7	8	9	10	11	12
Page referenced	1	2	3	4	1	2	5	1	2	3	4	5
Frames	<sup>1</sup> <u>1</u>	<sup>1</sup> 1	<sup>1</sup> 1	<sup>1</sup> 1	<sup>2</sup> 1	<sup>2</sup> 1	<sup>2</sup> 1	<sup>3</sup> 1				
_ = faulting page		<sup>1</sup> 2	<sup>1</sup> 2	<sup>1</sup> 2	<sup>1</sup> 2	<sup>2</sup> 2	<sup>2</sup> 2	<sup>2</sup> 2	<sup>3</sup> 2	<sup>3</sup> 2	<sup>3</sup> 2	<sup>3</sup> 2
n = reference count			<sup>1</sup> 3	<sup>1</sup> 3	<sup>1</sup> 3	<sup>1</sup> 3	<sup>1</sup> 5	<sup>1</sup> 5	<sup>1</sup> 5	<sup>1</sup> 3	<sup>1</sup> 3	<sup>1</sup> 5
			<sup>1</sup> 4	<sup>2</sup> 4	<sup>2</sup> 4							

- At the 7th reference, we victimize the page in the frame which has been referenced least often -- in this case, pages 3 and 4 (contained within frames 3 and 4) are candidates, each with a reference count of 1. Let's pick the page in frame 3. Page 5 is paged in and frame 3's reference count is reset to 1.
- At the 10th reference, we again have a page fault. Pages 5 and 4 (contained within frames 3 and 4) are candidates, each with a count of 1. Let's pick page 4. Page 3 is paged into frame 3, and frame 3's reference count is reset to 1.
- Analysis: 12 page references, 7 page faults, 3 page replacements. Page faults per number of frames =  $7/4 = 1.75$

### LRU algorithm

- Replaces pages based on their most recent reference – replace the page with the greatest backward distance in the reference string
- Example using 4 frames:

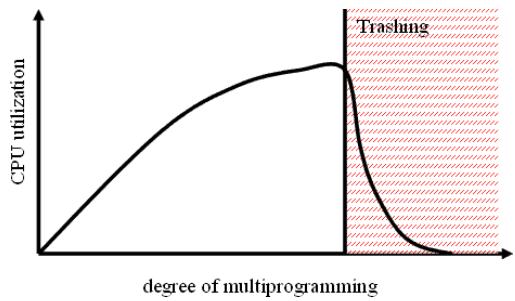
Reference #	1	2	3	4	5	6	7	8	9	10	11	12

<b>Page referenced</b>	1	2	3	4	1	2	5	1	2	3	4	5
<b>Frames</b>	1	1	1	1	1	1	1	1	1	1	1	5
_ = faulting page	2	2	2	2	2	2	2	2	2	2	2	2
		3	3	3	3	3	5	5	5	5	4	4
			4	4	4	4	4	4	4	3	3	3

- Analysis: 12 page references, 8 page faults, 4 page replacements. Page faults per number of frames =  $8/4 = 2$
- One possible implementation (not necessarily the best):
  - Every frame has a time field; every time a page is referenced, copy the current time into its frame's time field
  - When a page needs to be replaced, look at the time stamps to find the oldest

## Thrashing

- If a process does not have “enough” pages, the page-fault rate is very high
  - low CPU utilization
  - OS thinks it needs increased multiprogramming
  - adds another process to system
- Thrashing is when a process is busy swapping pages in and out
- Thrashing results in severe performance problems. Consider the following scenario, which is based on the actual behaviour of early paging systems. The operating system monitors CPU utilization. If CPU utilization is too low, we increase the degree of multiprogramming by introducing a new process to the system. A global page replacement algorithm is used; it replaces pages with no regard to the process to which they belong. Now suppose that a process enters a new phase in its execution and needs more frames.



## FILE SYSTEM

### File concept:

A file is a collection of related information that is stored on secondary storage. Information stored in files must be persistent i.e. not affected by power failures & system reboots. Files may be of free from such as text files or may be formatted rigidly. Files represent both programs as well as data.

Part of the OS dealing with the files is known as file system. The important file concepts include:

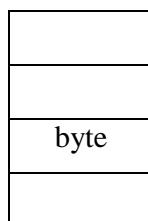
1. **File attributes:** A file has certain attributes which vary from one operating system to another.
  - **Name:** Every file has a name by which it is referred.
  - **Identifier:** It is unique number that identifies the file within the file system.
  - **Type:** This information is needed for those systems that support different types of files.
  - **Location:** It is a pointer to a device & to the location of the file on that device
  - **Size:** It is the current size of a file in bytes, words or blocks.
  - **Protection:** It is the access control information that determines who can read, write & execute a file.
  - **Time, date & user identification:** It gives information about time of creation or last modification & last use.
2. **File operations:** The operating system can provide system calls to create, read, write, reposition, delete and truncate files.
  - **Creating files:** Two steps are necessary to create a file. First, space must be found for the file in the file system. Secondly, an entry must be made in the directory for the new file.
  - **Reading a file:** Data & read from the file at the current position. The system must keep a read pointer to know the location in the file from where the next read is to take place. Once the read has been taken place, the read pointer is updated.

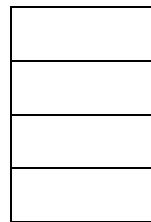
- **Writing a file:** Data are written to the file at the current position. The system must keep a write pointer to know the location in the file where the next write is to take place. The write pointer must be updated whenever a write occurs.
  - **Repositioning within a file (seek):** The directory is searched for the appropriate entry & the current file position is set to a given value. After repositioning data can be read from or written into that position.
  - **Deleting a file:** To delete a file, we search the directory for the required file. After deletion, the space is released so that it can be reused by other files.
  - **Truncating a file:** The user may erase the contents of a file but allows all attributes to remain unchanged except the file length which is reset to '0' & the space is released.
3. **File types:** The file name is split into 2 parts, Name & extension. Usually these two parts are separated by a period. The user & the OS can know the type of the file from the extension itself. Listed below are some file types along with their extension:

File Type	Extension
Executable File	exe, bin, com
Object File	obj, o (compiled)
Source Code file	C, C++, Java, pas
Batch File	bat, sh (commands to command the interpreter)
Text File	txt, doc (textual data documents)
	arc, zip, tar (related files grouped together into file compressed for storage)
Archive File	
Multimedia File	mpeg (Binary file containing audio or A/V information)

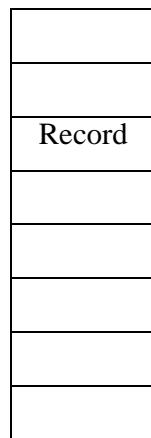
4. **File structure:** Files can be structured in several ways. Three common possible are:

- **Byte sequence:** The figure shows an unstructured sequence of bytes. The OS doesn't care about the content of file. It only sees the bytes. This structure provides maximum flexibility. Users can write anything into their files & name them according to their convenience. Both UNIX & windows use this approach.





- **Record sequence:** In this structure, a file is a sequence of fixed length records. Here the read operation returns one records & the write operation overwrites or append or record.



- **Tree:** In this organization, a file consists of a tree of records of varying lengths. Each record consists of a key field. The tree is stored on the key field to allow first searching for a particular key.

**Access methods:** Basically, access method is divided into 2 types:

- **Sequential access:** It is the simplest access method. Information in the file is processed in order i.e. one record after another. A process can read all the data in a file in order starting from beginning but can't skip & read arbitrarily from any location. Sequential files can be rewound. It is convenient when storage medium was magnetic tape rather than disk.
- **Direct access:** A file is made up of fixed length-logical records that allow programs to read & write records rapidly in no particular order. This method can be used when disk are used for storing files. This method is used in many applications e.g. database systems. If an airline customer wants to reserve a seat on a particular flight, the reservation program must be able to access the record for that flight directly without reading the records before it. In a direct access file, there is no restriction in the order of reading or writing. For example, we can read block 14, then read block 50 & then write block 7 etc. Direct access files are very useful for immediate access to large amount of information.

**Directory structure:** The file system of computers can be extensive. Some systems store thousands of file on disk. To manage all these data, we need to organize them. The organization is done in 2 steps. The file system is broken into partitions. Each partition contains information about file within it.

### **Operation on a directory:**

- **Search for a file:** We need to be able to search a directory for a particular file.
- **Create a file:** New files are created & added to the directory.
- **Delete a file:** When a file is no longer needed, we may remove it from the directory.
- **List a directory:** We should be able to list the files of the directory.
- **Rename a file:** The name of a file is changed when the contents of the file changes.
- **Traverse the file system:** It is useful to be able to access every directory & every file within a directory.

**Structure of a directory:** The most common schemes for defining the structure of the directory are:

1. **Single level directory:** It is the simplest directory structure. All files are present in the same directory. So it is easy to manage & understand.  
**Limitation:** A single level directory is difficult to manage when the no. of files increases or when there is more than one user. Since all files are in same directory, they must have unique names. So, there is confusion of file names between different users.
2. **Two level directories:** The solution to the name collision problem in single level directory is to create a separate directory for each user. In a two level directory structure, each user has its own user file directory. When a user logs in, then master file directory is searched. It is indexed by user name & each entry points to the UFD of that user.  
**Limitation:** It solves name collision problem. But it isolates one user from another. It is an advantage when users are completely independent. But it is a disadvantage when the users need to access each other's files & co-operate among themselves on a particular task.
3. **Tree structured directories:** It is the most common directory structure. A two level directory is a two level tree. So, the generalization is to extend the directory structure to a tree of arbitrary height. It allows users to create their own subdirectories & organize their files. Every file in the system has a unique path name. It is the path from the root through all the sub-directories to a specified file. A directory is simply another file but it is treated in a special way. One bit in each

directory entry defines the entry as a file (O) or as sub-directories. Each user has a current directory. It contains most of the files that are of current interest to the user. Path names can be of two types: An absolute path name begins from the root directory & follows the path down to the specified files. A relative path name defines the path from the current directory. E.g. If the current directory is root/spell/mail, then the relative path name is prt/first & the absolute path name is root/ spell/ mail/ prt/ first. Here users can access the files of other users also by specifying their path names.

4. **A cyclic graph directory:** It is a generalization of tree structured directory scheme. An a cyclic graph allows directories to have shared sub-directories & files. A shared directory or file is not the same as two copies of a file. Here a programmer can view the copy but the changes made in the file by one programmer are not reflected in the other's copy. But in a shared file, there is only one actual file. So many changes made by a person would be immediately visible to others. This scheme is useful in a situation where several people are working as a team. So, here all the files that are to be shared are put together in one directory. Shared files and sub-directories can be implemented in several ways. A common way used in UNIX systems is to create a new directory entry called link. It is a pointer to another file or sub-directory. The other approach is to duplicate all information in both sharing directories. A cyclic graph structure is more flexible than a tree structure but it is also more complex.

**Limitation:** Now a file may have multiple absolute path names. So, distinct file names may refer to the same file. Another problem occurs during deletion of a shared file. When a file is removed by any one user. It may leave dangling pointer to the non existing file. One serious problem in a cyclic graph structure is ensuring that there are no cycles. To avoid these problems, some systems do not allow shared directories or files. E.g. MS-DOS uses a tree structure rather than a cyclic to avoid the problems associated with deletion. One approach for deletion is to preserve the file until all references to it are deleted. To implement this approach, we must have some mechanism for determining the last reference to the file. For this we have to keep a list of reference to a file. But due to the large size of the no. of references. When the count is zero, the file can be deleted.

5. **General graph directory:** When links are added to an existing tree structured directory, the tree structure is destroyed, resulting in a simple graph structure. Linking is a technique that allows a file to appear in more than one directory. The advantage is the simplicity of algorithm to transverse the graph & determines when there are no more references to a file. But a similar

problem exists when we are trying to determine when a file can be deleted. Here also a value zero in the reference count means that there are no more references to the file or directory & the file can be deleted. But when cycle exists, the reference count may be non-zero even when there are no references to the directory or file. This occurs due to the possibility of self referencing (cycle) in the structure. So, here we have to use garbage collection scheme to determine when the last references to a file has been deleted & the space can be reallocated. It involves two steps:

- Transverse the entire file system & mark everything that can be accessed.
- Everything that isn't marked is added to the list of free space.

But this process is extremely time consuming. It is only necessary due to presence of cycles in the graph. So, a cyclic graph structure is easier to work than this.

## Protection

When information is kept in a computer system, a major concern is its protection from physical damage (reliability) as well as improper access.

**Types of access:** In case of systems that don't permit access to the files of other users. Protection is not needed. So, one extreme is to provide protection by prohibiting access. The other extreme is to provide free access with no protection. Both these approaches are too extreme for general use. So, we need controlled access. It is provided by limiting the types of file access. Access is permitted depending on several factors. One major factor is type of access requested. The different type of operations that can be controlled are:

- **Read**
- **Write**
- **Execute**
- **Append**
- **Delete**
- **List**

### Access lists and groups:

Various users may need different types of access to a file or directory. So, we can associate an access lists with each file and directory to implement identity dependent access. When a user access requests access to a particular file, the OS checks the access list associated with that file. If that user is granted the requested access, then the access is allowed. Otherwise, a protection violation occurs & the user is denied access to the file. But the main problem with access lists is their length. It is

very tedious to construct such a list. So, we use a condensed version of the access list by classifying the users into 3 categories:

- **Owners:** The user who created the file.
- **Group:** A set of users who are sharing the files.
- **Others:** All other users in the system.

Here only 3 fields are required to define protection. Each field is a collection of bits each of which either allows or prevents the access. E.g. The UNIX file system defines 3 fields of 3 bits each: rwx

- r( read access)
- w(write access)
- x(execute access)

Separate fields are kept for file owners, group & other users. So, a bit is needed to record protection information for each file.

## Allocation methods

There are 3 methods of allocating disk space widely used.

### 1. Contiguous allocation:

- a. It requires each file to occupy a set of contiguous blocks on the disk.
- b. Number of disk seeks required for accessing contiguously allocated file is minimum.
- c. The IBM VM/CMS OS uses contiguous allocation. Contiguous allocation of a file is defined by the disk address and length (in terms of block units).
- d. If the file is ‘n’ blocks long and starts at location ‘b’, then it occupies blocks b, b+1, b+2,----, -b+ n-1.
- e. The directory for each file indicates the address of the starting block and the length of the area allocated for each file.
- f. Contiguous allocation supports both sequential and direct access. For sequential access, the file system remembers the disk address of the last block referenced and reads the next block when necessary.
- g. For direct access to block i of a file that starts at block b we can immediately access block b + i.
- h. **Problems:** One difficulty with contiguous allocation is finding space for a new file. It also suffers from the problem of external fragmentation. As files are deleted and allocated, the free disk space is broken into small pieces. A major problem in contiguous allocation is how

much space is needed for a file. When a file is created, the total amount of space it will need must be found and allocated. Even if the total amount of space needed for a file is known in advance, pre-allocation is inefficient. Because a file that grows very slowly must be allocated enough space for its final size even though most of that space is left unused for a long period of time. Therefore, the file has a large amount of internal fragmentation.

## 2. Linked Allocation:

- a. Linked allocation solves all problems of contiguous allocation.
- b. In linked allocation, each file is a linked list of disk blocks, which are scattered throughout the disk.
- c. The directory contains a pointer to the first and last blocks of the file.
- d. Each block contains a pointer to the next block.
- e. These pointers are not accessible to the user. To create a new file, we simply create a new entry in the directory.
- f. For writing to the file, a free block is found by the free space management system and this new block is written to & linked to the end of the file.
- g. To read a file, we read blocks by following the pointers from block to block.
- h. There is no external fragmentation with linked allocation & any free block can be used to satisfy a request.
- i. Also there is no need to declare the size of a file when that file is created. A file can continue to grow as long as there are free blocks.
- j. **Limitations:** It can be used effectively only for sequential access files. To find the 'i' th block of the file, we must start at the beginning of that file and follow the pointers until we get the ith block. So it is inefficient to support direct access files. Due to the presence of pointers each file requires slightly more space than before. Another problem is reliability. Since the files are linked together by pointers scattered throughout the disk. What would happen if a pointer were lost or damaged.

## 3. Indexed Allocation:

- a. Indexed allocation solves the problem of linked allocation by bringing all the pointers together to one location known as the index block.
- b. Each file has its own index block which is an array of disk block addresses. The ith entry in the index block points to the ith block of the file.

- c. The directory contains the address of the index block. To read the  $i^{\text{th}}$  block, we use the pointer in the  $i^{\text{th}}$  index block entry and read the desired block.
- d. To write into the  $i^{\text{th}}$  block, a free block is obtained from the free space manager and its address is put in the  $i^{\text{th}}$  index block entry.
- e. Indexed allocation supports direct access without suffering external fragmentation.
- f. **Limitations:** The pointer overhead of index block is greater than the pointer overhead of linked allocation. So here more space is wasted than linked allocation. In indexed allocation, an entire index block must be allocated, even if most of the pointers are nil.

## Free Space Management

Since there is only a limited amount of disk space, it is necessary to reuse the space from the deleted files. To keep track of free disk space, the system maintains a free space list. It records all the disk blocks that are free i.e. not allocated to some file or dictionary. To create a file, we search the free space list for the required amount of space and allocate it to the new file. This space is then removed from the free space list. When a file is deleted, its disk space is added to the free space list.

### Implementation:

There are 4 ways to implement the free space list such as:

- **Bit Vector:** The free space list is implemented as a bit map or bit vector. Each block is represented as 1 bit. If the block is free, the bit is 1 and if it is allocated then the bit is 0. For example, consider a disk where blocks 2, 3, 4, 5, 8, 9, 10, 11, 12, 13, 17, 18, 25, 26 & 27 are free and rest of the blocks are allocated. The free space bit map would be 0011110011111100011000000111.....

The main advantage of this approach is that it is simple and efficient to find the first free block or n consecutive free blocks on the disk. But bit vectors are inefficient unless the entire vector is kept in main memory. It is possible for smaller disks but not for larger ones.

- **Linked List:** Another approach is to link together all the free disk blocks and keep a pointer to the first free block. The first free block contains a pointer to the next free block and so on. For example, we keep a pointer to block 2 as the free block. Block 2 contains a pointer to block which points to block 4 which then points to block 5 and so on. But this scheme is not efficient. To traverse the list, we must read each block which require a lot of I/O time.

- **Grouping:** In this approach, we store the address of n free blocks in the first free block. The first n-1 of these blocks is actually free. The last block contains the address of another n free blocks and so on. Here the addresses of a large number of free blocks can be found out quickly.
- **Counting:** Rather than keeping a list of n free disk block addresses, we can keep the address of the first free block and the number of free contiguous blocks. So here each entry in the free space list consists of a disk address and a count.