

# **Advanced Operating System(CSc-538)**

**Dr. Binod Kumar Adhikari  
Asst. Prof., CDCSIT TU**

# Objectives of the course

- Introduce the underlying principles of an operating system, virtual memory and resource management concepts.
- Explore current state-of-art research in operating system
- Exposure of distributed operating system, real- operating system and multimedia time systems

# Text Books and Materials

## Text Books:

1. Silberschatz, A. and P. B. Galvin, *Operating System Concepts*, Sixth Edition, John-Wiley. (not Java edition)
2. Tanenbaum, A. S., *Modern Operating Systems*, Second Edition, PHI.

## References :

3. Research and Technical Papers
4. Stallings, W., *Operating Systems*, Fourth Edition, Pearson.
5. Deitel H. M., *Operating Systems*, Second Edition, Addison Wesley.
6. Tanenbaum, A. S. and Woodhull, A. S., *Operating Systems Design and Implementation*, Second Edition, PHI.

# Tentative Course Plan

- **Process Management and Synchronization** 12hrs
  - Process Model and communication, Process Scheduling, Deadlock
  - Research Paper Presentation( By Group of students).
- **Memory Management** 13hrs
  - (Virtual Memory Management: Paging and Segmentation, Storage/free space Management)
  - Research Paper Presentation(By Group of Students)
- **Protection and Security** 8hrs
  - System Protection and Security Principles and Algorithms
  - Research Paper Presentation(By Group of Students)
- **Special Purpose Systems** 12hrs
  - Distributed System, Real time System, Multimedia System
  - Research Paper Presentations

# Evaluations

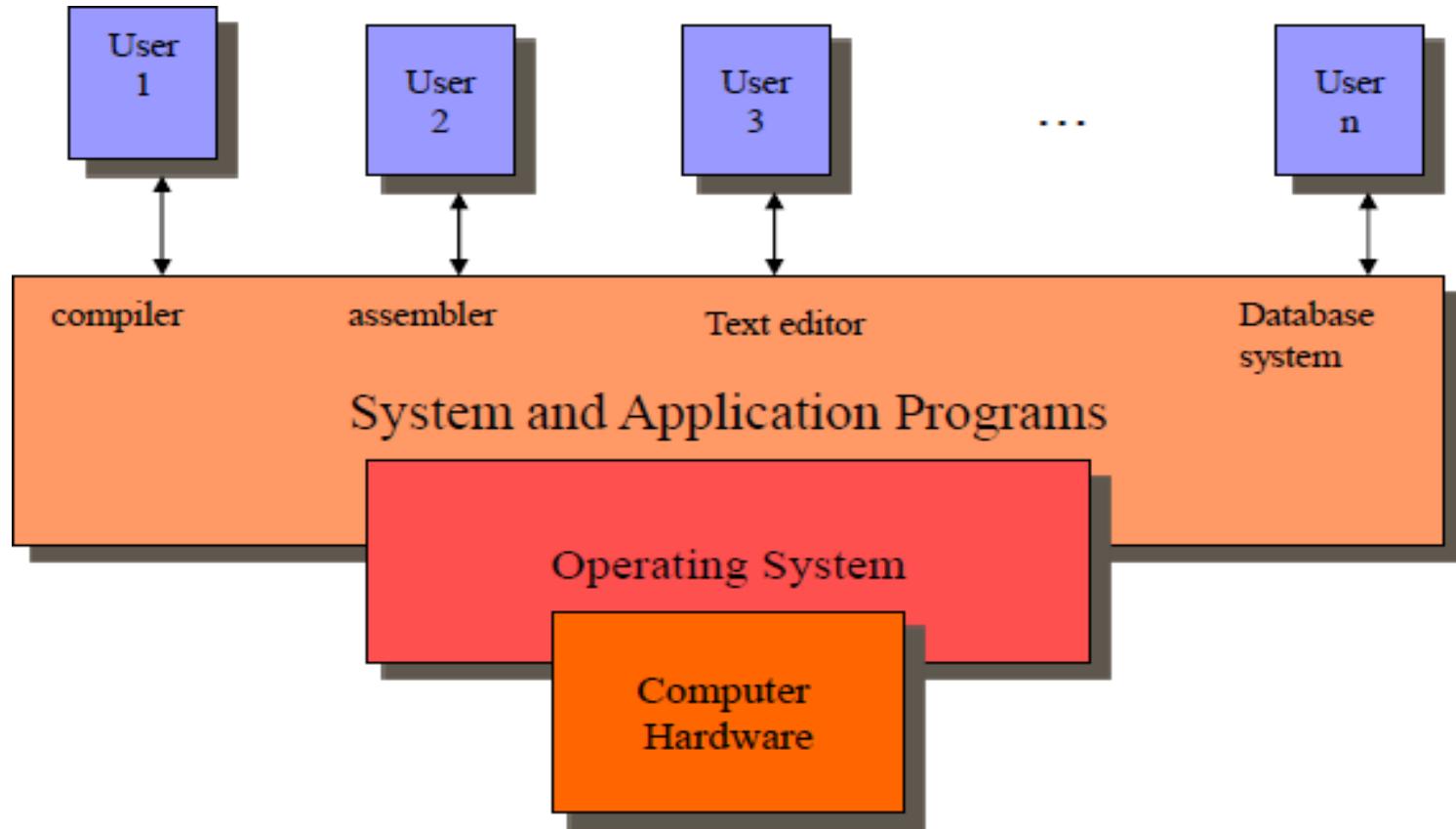
- Internal Assessment: 30 marks
  - First Term Exam
  - Pre-Board Exam
  - Research Paper Presentations
  - Attendance (80% compulsory)
- Semester Exam: 45 marks
  - End Semester Exam conducted by IOST, TU

# **Unit 1.1**

## **Introduction to Operating System(OS)**

**Reading: Chapter 1 of Textbook**

# Where does OS sits?



# Unit1.1 What is Operating System(OS)?

- An OS is a program that acts an intermediary between the user of a computer and computer hardware.
- OS is a system software which provides an environment to help the user to execute the programs.
  - Extend machine
- OS simplifies and manages the complexity of running application programs efficiently.
- OS is a resource manager which allocates and manages various resource like CPU, Main memory, I/O device, Hard disk etc.- **resource manager**

# Function of OS

- OS perform the following important functions
  - **CPU Management:** It means assigning **processor** to different **task** which has to be performed by computer.
  - **Memory management:** It means allocation of main memory to different task to execute.
  - **Input/output management:** it means co-ordination between different input and output devices while one or more program are running.
  - **File system management:** OS is responsible for maintenance of file system.

# Operating Systems Views

- **As a Resource allocator**
  - to allocate resources (software and hardware) of the computer system and manage them efficiently.
- **As a extended machine or controller**
  - Controls execution of user programs and operation of I/O devices.

# OS as an Extended Machine

- OS function is to present the user with the equivalent of an extended machine or virtual machine that is easier to program than the underlying hardware.
- OS creates higher-level abstraction for programmer.
- What are the right abstractions? How to achieve this?

# OS as an Extended Machine

- **Example: (Floppy disk I/O operation)**
  - disks contains a collection of named files
  - each file must be open for READ/WRITE
  - after READ/WRITE complete close that file
  - no any detail to deal
- OS shields the programmer from the disk hardware and presents a simple file oriented interface.

# OS as a Resource Manager

- OS primary function is to manage all component of a complex system.
- *What happens if three programs try to print their output on the same printer at the same time?*
- *What happens if two network users try to update a shared document at same time?*
- OS should take care of these.

# OS as resource Manager(cont...)

- OS as a resource manager does the following:
  - Virtualizes resource so multiple users/ applications can share the resources
  - Protect applications from one another
  - Provide efficient and fair access to resources.
- *To do these task, OS designer should include the appropriate policy and mechanism in OS design*

# Unit 1.2 OS Evolution( History)

- Present OS haven't been developed overnight.
- Like any other system, OS also have evolved over a period of time.
- Evolution starts from the very primitive OS to the present most complex and versatile OS

# OS Evolution( History)

## Computer Generation, Component and OS Types

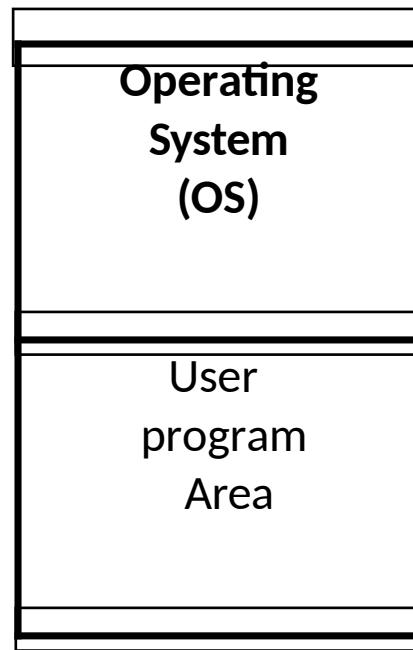
1 <sup>st</sup> (1945-55)	Vacuum Tubes	User Driven
2 <sup>nd</sup> (1955-65)	Transistor	Batch
3 <sup>rd</sup> (1965-80)	IC	Multiprogramming
4 <sup>th</sup> (1980-present)	PC	Client Server/Distributed

# Batch System

- In batch system, operator hired to run computer, the user prepared a job –which consisted of the program, the data, and some control information about the nature of the jobs – and submitted it to the computer operator.
- Output appears after some minutes and user collects the outputs from the operator.
- Why the name batch system?
  - Batch: Group of jobs submitted to machine together
  - Operator collects jobs; orders efficiently; runs one at a time
- Here the OS is very simple and its only task is to transfer control from one job to another.

# Batch System

- The OS is resident on the memory



# Batch System

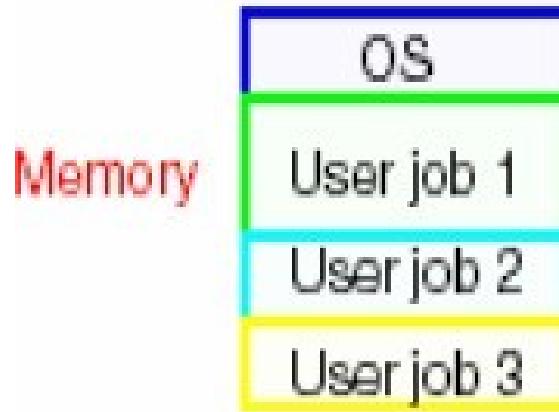
- Advantages:
  - Average setup costs over many jobs
  - Operator more skilled at loading tapes
  - Keep machine busy while programmer thinks
- Problem:
  - User must wait for results until batch collected and submitted
  - (*If bug receive, memory and register dump; submit job again!*)

# Multiprogramming operating system (MOS)

- A job pool on the disk consist of a number of jobs that are ready to be executed.
- Subset of these jobs reside in memory during execution.
- OS picks **one job ‘x’** and assign it to CPU to executes in memory.
- When this job x needs an I/O operation to complete, the CPU is **idle**.
- Now OS switches the CPU to another job (y) to utilize it until the previous job (x) resume (ready) to execute.
- As long as there are jobs in memory waiting for CPU, the CPU is never idle.
- Choosing one out of several jobs in memory for execution is called **CPU scheduling** (CPU execute jobs)

# Multiprogramming (MOS)

What is the main function of Multiprogramming OS?



## New OS functionality

- Job scheduling policies
- Memory management and protection (virtual memory)

# Timesharing OS

- Problem with MOS:
  - increasing number of users who want to interact with programs while they are running.
  - humans' time is expensive don't want to wait.
- Solution
  - Timesharing (a variant of Multiprogramming)
- *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.*

# Time sharing OS (MOS)

- It is the logical extension of Multiprogramming
- OS provides each user one time slice(slot or share) and each process execute for one time slot in one round.
- The CPU switches from one job to another at the end of time slice.
- The switching is so fast that the user gets the illusion that the CPU is executing only one user's job

# Timesharing OS(cont.....)

- Goal:
  - Improve user's response time (interaction)
- Advantage:
  - **Users easily submit jobs and get immediate feedback**
- New OS functionality
  - More complex job scheduling, memory management
  - Concurrency control and synchronization

# Personal Computers (PC-OS)

- The main mode of use of PC is by a single user( as name personal computer).
- The System of PC consist of two parts:
  - BIOS(basic input output system) which is stored in a ROM.
  - The other part called DOS is stored in a floppy disk or hard disk.
- BIOS does what is known is power on self test. Having done this it reads small portion of OS from the disk known as boot portion. Then DOS do all other task such as file management, disk management, CPU management.

# Personal Computers (PC-OS)

- Dedicated machine per user
- CPU utilization is not a prime concern
  - *Hence, some of the design decision made for advanced system may not be appropriate for these smaller system. But other design decisions, such as those for security, are appropriate because PCs can now be connected to other computer and users through the network and the web.*
- There are the different types of OS for PC such as Windows, MacOS, UNIX, Linux.

# Multiprocessor OS

- Most system to date are single-processor system, which will be having only one CPU. However there is a need for multiprocessor system.
- A System having more than one processor and all these processor will share the computer bus, the clock and sometime memory, is **Multiprocessor OS**.

# Distributed OS

- It is the logical extension of multiprocessor system
- In such system, all computation of jobs are distributed among several process.
- Here processor don't share memory and clock. They communicate with one another via communication lines.

# Distributed OS

- *Benefits:*
  - *Resource sharing*
  - *Computation speedup*
  - *Reliability*
  - *Communication*

# Process Management

- A process is a program in execution.
- For example
  - A batch job is a process
  - A time-shared user program is a process
- A system task (e.g. spooling output to printer) is a process.
- Remember a program itself is not a process rather it is a passive entity.

# Process Management

- A process needs certain resources, including CPU time, memory, files, and I/O devices, to accomplish its task.
- These resources are either given to the process when it is created or when it is running.
- When the process completes, the OS reclaims all the resources.
- Process manager (OS) handle all these responsibility

# Process Management

The operating system is responsible for the following activities in connection with process management.

- Process creation and deletion.
- Process suspension and resumption.
- Provision of mechanisms for:
  - Process synchronization
  - Process communication

# Memory Management

*Memory is a large array of words or bytes, each with its own address.*

- It is a repository of quickly accessible data shared by the CPU and I/O devices.
- Main memory is a volatile storage device. It loses its contents in the case of system failure.

# Memory Management

*The operating system is responsible for the following activities in connections with memory management:*

- Keep track of which parts of memory are currently being used and by whom.
- Decide which processes to load when memory space becomes available.
- Allocate and deallocate memory space as needed.

# File Management

*A file is a collection of related information defined by its creator. Commonly, files represent programs (both source and object forms) and data.*

- Most visible component of OS. Computers can store information on several different types of physical media (e.g. magnetic tape, magnetic disk, CD etc).
- A file is a logical storage unit, which abstract away the physical properties of its storage device.

# File Management

*The operating system is responsible for the following activities in connections with file management:*

- File creation and deletion.
- Directory creation and deletion.
- Support of primitives for manipulating files and directories.
- Mapping files onto secondary storage.
- File backup on stable (nonvolatile) storage media.

# I/O Management

*All computers have physical devices for acquiring input and producing output.*

- The I/O system consists of:
  - A memory management system for buffering and caching system
  - A general device-driver interface
  - Drivers for specific hardware devices
  - Disk management

# Secondary Storage Management

- The computer system must provide *secondary storage* to back up main memory since main memory is volatile.
- Most modern computer systems use disks as the principle on-line storage medium, for both programs and data.

# Secondary Storage Management

*The operating system is responsible for the following activities in connection with disk management:*

- Free space management
- Storage allocation
- Disk scheduling

# System calls

- System calls provide the interface between a process and the operating system.
  - These calls are generally available as assembly language instructions.
  - Some systems also allow to make system calls from a high level language, such as C.
- Example:
  - `count = read(fd, buffer, nbytes)`

**Next: Process Management  
Actual Course Begins Here**

# **Unit 1.1 Process Management**

Chapter 2 of Text Book

# Objectives

---

- **After studying this unit, you should be able to**
  - Explain process state
  - Describe the process control block(PCB)
  - Discuss the process scheduling
  - Describe threads in Operating system

# Process Concept

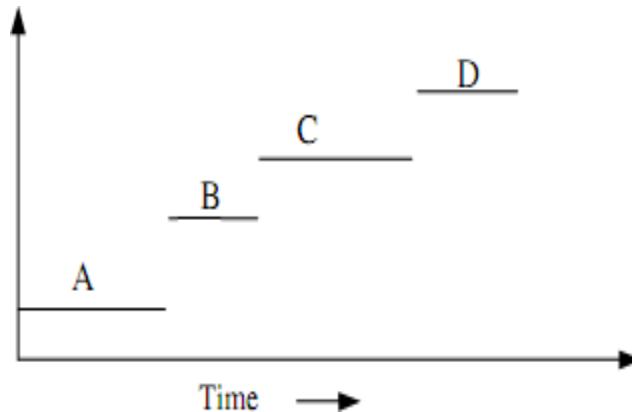
- An operating system executes a variety of programs
  - batch systems - jobs
  - time-shared systems - user programs or tasks
  - job and program used interchangeably
- Process - a program in execution
  - process execution proceeds in a sequential fashion
- A process is an activity of some kind, it has program, input, output and state.

# Process Model

- Uni-programming
- Multiprogramming
- Multiprocessing

# Uni-programming Model

**Only one process at a time**

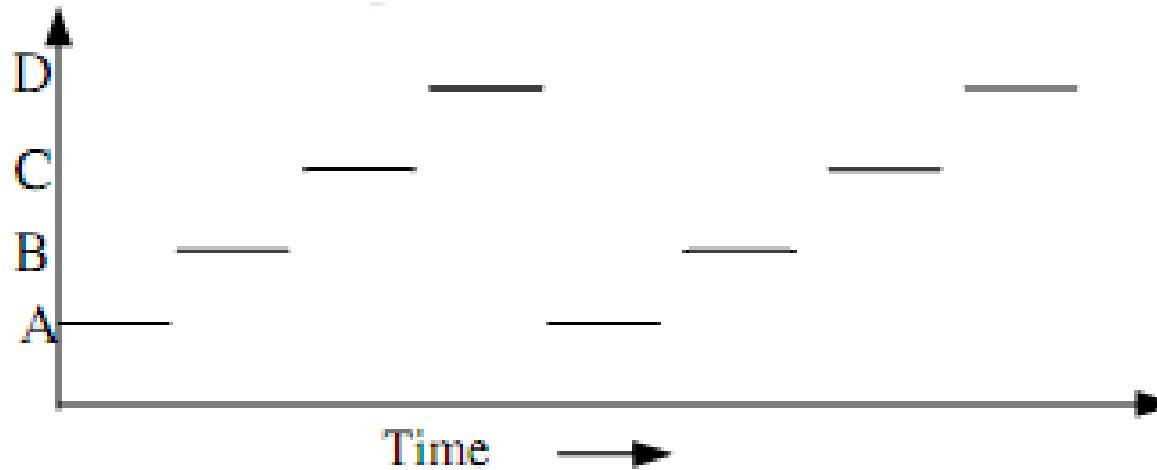


- Examples: Older systems
- Advantages: Easier for OS designer
- Disadvantages: Not convenient for user and poor performance

# Multiprogramming Mode

multiple process at a time

- OS requirements for multiprogramming:
- Policy: to determine which process is to schedule.
- Mechanism: to switch between the processes.

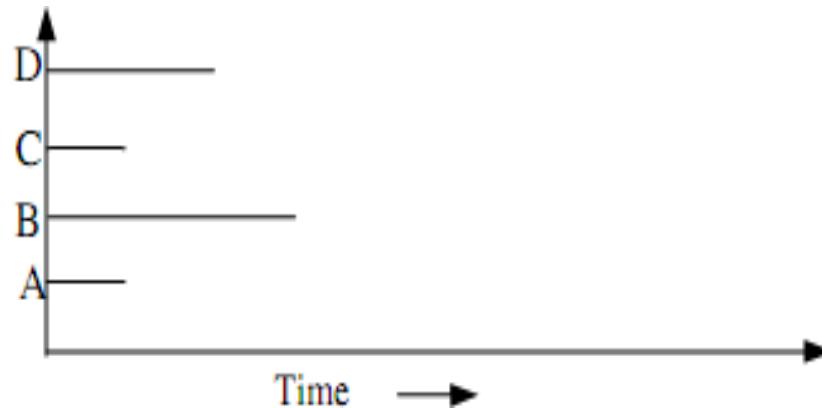


- Examples: Unix, WindowsNT
- Advantages: Better system performance and user convenience.
- Disadvantages: Complexity in OS

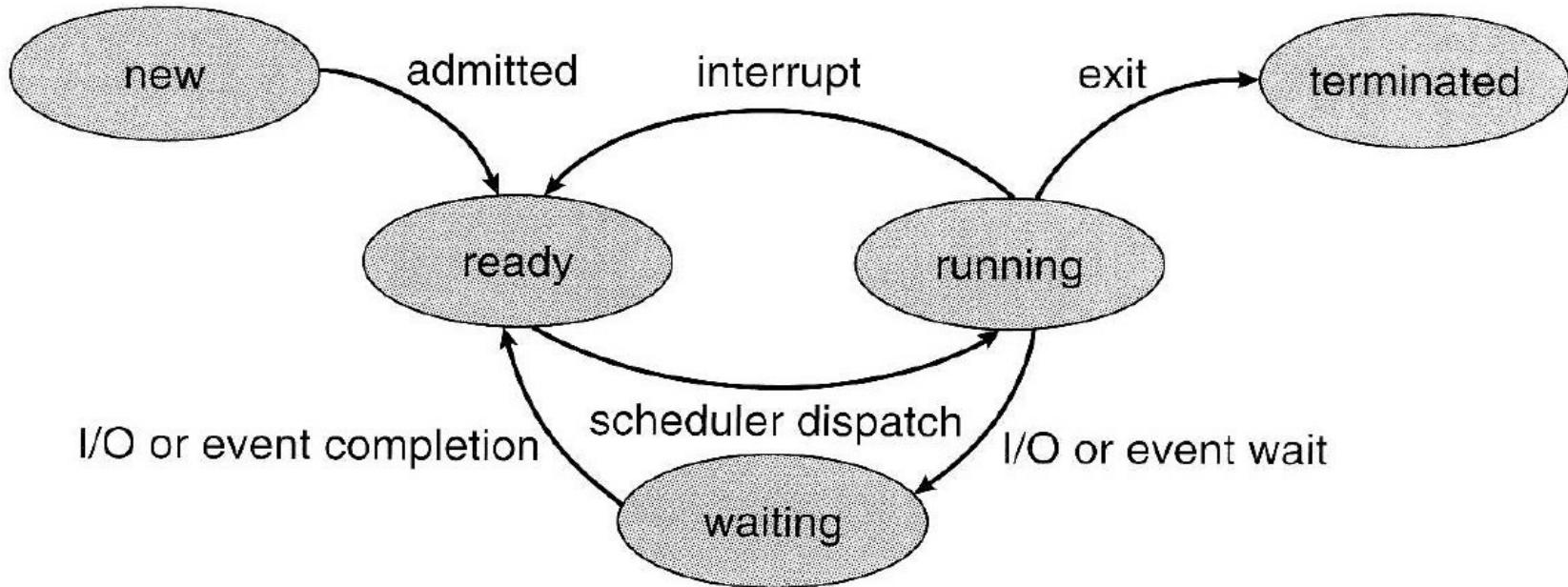
# Multiprocessing Model

System with multiple processor

- Distributed OS and Network OS



# Process States diagram



**Figure 3.2** Diagram of process state.

# Process States

---

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

# Process Control Block

- Process must be saved when the process is switched from one state to another so that it can be restarted later as it had never been stopped.
- The PCB is the data structure containing certain important information about the process -also called process table or processor descriptor.
  - Process state: running, ready, blocked.
  - Program counter: Address of next instruction for the process.
  - Registers: Stack pointer, accumulator, PSW etc.
  - Scheduling information: Process priority, pointer to scheduling queue etc.
  - Memory-allocation: value of base and limit register, page table, segment table etc.
  - Accounting information: time limit, process numbers etc.
  - Status information: list of I/O devices, list of open files etc.

# PCB

- The PCB simply serves as repository of any information that may vary from process to process.

Process state
Process number
Program counter
Registers
Memory Limits
List of open files
.....

# Operation on process

- The processes in the system can execute *concurrently*, and they must be ~~detected~~ dynamically. OS provide mechanism for process the termination.  
– Process Creation.  
– Process Termination

# Process Creation

- There are four principal events that cause the process to be created:
  - » System initialization.
  - » Execution of a process creation system call.
  - » User request to create a new process.
  - » Initiation of a batch job.
- A process may create several new processes during the course of execution. The creating process is called a ***parent process***, whereas the new processes are called children of that process.

# Process Termination

- Process are terminated on the following conditions
  - Normal exit.
  - Error exit.
  - Fatal error.
  - Killed by another process.
- Example:
  - In Unix the normal exit is done by calling a exit system call. The process returns data (output) to its parent process via the wait system call.
  - kill system call is used to kill other process.

# Process Scheduling

- Main objective of multiprogramming is to **maximize the CPU utilization**.
- At any time, a process is in any one of the new, ready, running or waiting state.
- The OS module known as scheduler, schedules processes from ready queue for execution by CPU
- Scheduler select one of the many process from ready queue on certain criteria.
- (Will discuss it in detail in next chapter)

# Context Switch

- CPU switching from one process to another requires **saving the state** of the current process and loading the latest state of next process.
- This is known as **context switch**.
- The time required to context switch is an overhead since it does not produce any useful work in aspect of process execution time.

# Thread

- It is a single sequence stream which allows a program to split itself into two or more simultaneously running task.
- It is the basic unit of CPU utilization and consists of program counter, a register set and a stack space.
- Sometime called light weight process.

# Thread and Process

- Threads are not independent as process.
- Thread shares with other thread: code section, data section, Other resource such as open files.
- Similarities:
  - Like process, thread share CPU and only one thread is running at a time.
  - Like process, threads within processes execute sequentially
  - Like process if one thread is blocked, another can run.

# Thread and process

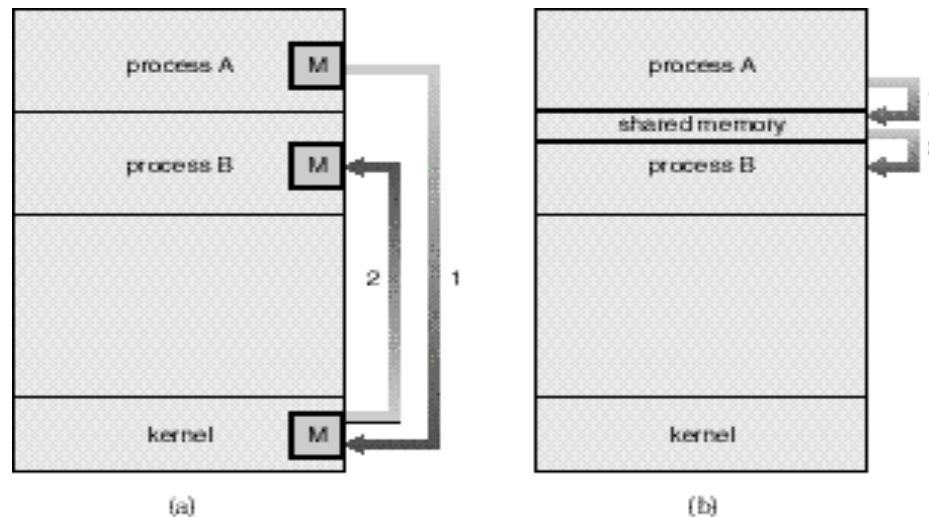
- Dissimilarities
  - Unlike process, threads are not independent of each other.
  - Unlike process, all threads can access every address space.
  - Unlike process, threads are designed to assist one other.

# IPC(Inter Process Communication)

- How one process can pass the information to the another?
- How to make sure two or more processes do not get into each other's way when engaging in critical activities?
- How to maintain the proper sequence when dependencies are presents?
- Thus, Inter Process Communication (IPC) and synchronization are needed.
- IPC provide the mechanism to allow the processes to communicate and to synchronize their actions.

# IPC models

- There are two fundamental models of inter-process communication:
  - Shared Memory:
  - Message Passing.



# Massage Passing

- Communication takes place by means of messages exchanged between the cooperating processes.
- Message passing is useful for exchanging *smaller amounts* of data, because no conflicts need be avoided. Message passing is also easier to implement than shared memory for inter-computer communication.

# Shared Memory

- A region of memory that is shared by cooperating processes is established.
- Processes can then exchange information by reading and writing data to the shared region.
- Shared memory allows maximum speed and convenience of communication, as it can be done at memory speeds when within a computer.
- Shared memory is faster than message passing, as message-passing systems are typically implemented using system calls and thus require the more time-consuming task of kernel intervention.

# Massage passing Vs Sharing Memory

- In contrast, in shared-memory systems, system calls are required only to establish shared-memory regions. Once shared memory is established, all accesses are treated as routine memory accesses, and no assistance from the kernel is required.

# Process Synchronization

- Synchronization is often necessary when processes communicate.
- Processes are executed with unpredictable speeds. One can view synchronization as a set of constraints on the ordering of events.
- The programmer employs a synchronization mechanism to delay execution of a process in order to satisfy such constraints.
- As an illustrative example, consider the batch operating system again. A shared buffer is used for communication between **the reader process and the writer process**.
- These processes must be synchronized so that, for example, the reader process never attempts to read data from the input if the buffer is empty.

# Race Condition

---

**Situation where two or more processes are reading or writing some shared data and the final result depends on who run precisely, are called race conditions**

# Race Condition?

- **Possibilities of race:**
  - one process reading and another process writing same data.
  - two or more process writing same data.
- **Solution:** prohibiting more than one process from reading /writing the same data at the same time- **Mutual Exclusion**.

# Solution to Race condition

- Make an operating system not to perform several tasks in parallel. This is called **serialization**. Two strategies to serializing processes in a multitasking environment:
  - The Scheduler can be disabled
  - A Protocol can be introduced
- **Scheduler** can be disabled for a short period of time, to prevent control being given to another process during a critical action like modifying shared data. This will be inefficient on multiprocessor machines, since all other processors have to be halted every time one wishes to execute a critical section.
- A **protocol** can be introduced which all programs sharing data must obey. The protocol ensures that processes have to **queue** up to gain access to shared data.

# Then what is Mutual Exclusion really?

---

- Some way of making sure that if one process is using a shared variables or files, the other process will be excluded from doing the same thing.

# Critical-Section Problem

- Code executed by the process can be grouped into sections, some of which require access to shared resources, and other that do not.
- The section of the code that require access to shared resources are called **critical section**.
- *Facts: The part of the time, process is busy doing internal computations and other things that do not lead to the race condition. Non critical section*
- ***Helpful to avoid race condition.***

# Critical-Section Problem

General structure of process Pi (other process Pj)

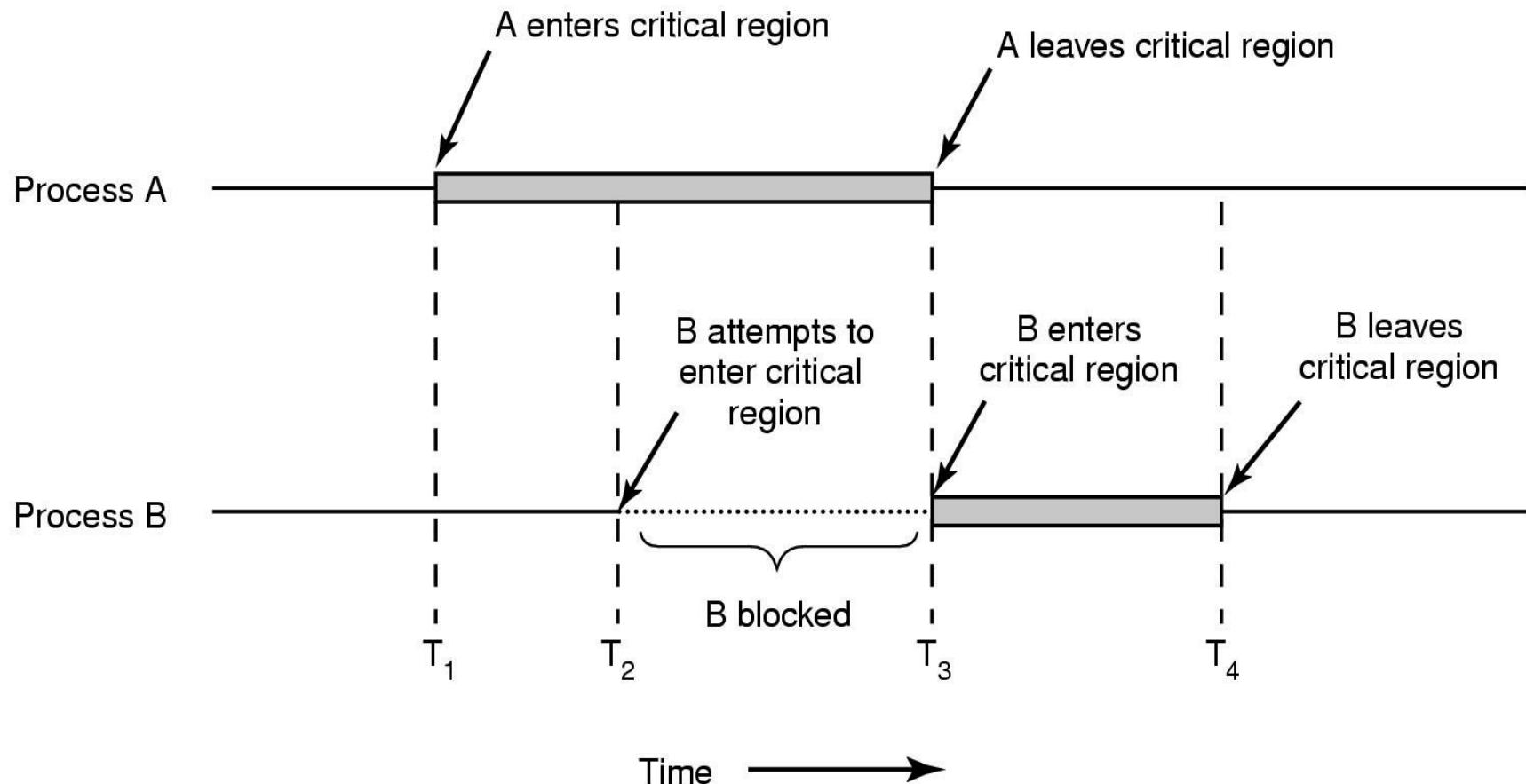
```
do{
    entry_section
        critical_section(CR)
    exit_section
        reminder_section
} while(true)
```

- When a process is accessing a shared modifiable data, the process is said to be in critical section.
- All other processes (those access the same data) are excluded from their own critical region.
- All other processes may continue executing outside their CR.
- When a process leaves its critical section, then another processes waiting to enter its own CR should be allowed to proceed.

# Critical-Section Problem

- The critical section problem is to design a **protocol** that the processes can use to co-operate.
- A good solution to critical section problem must satisfy the following three requirements.
  - No two processes may be simultaneously inside their CRs (**mutual exclusion**).
  - No process running outside its CR may block other process. (**Progress**)
  - No process should have to wait forever to enter its CR. (**Bounded Waiting**)

# Critical Section diagram (with mutual exclusion)



# Solution 1(Busy waiting) Interrupt Disabling

- *Each process disable all interrupts just after entering its CR and re-enable them just before leaving it.*
- No clock interrupt, no other interrupt, no CPU switching to other process until the process turn on (enable) the interrupt.
- The general structure of process may be as
- Do

```
{  
    DisableInterrupt()  
        // perform CR task  
    EnableInterrupt()  
        //perform  
        Remiander section  
        task  
}while(1);
```

# Facts about Disabling Interrupts

- **Advantages:**
  - Mutual exclusion can be achieved by implementing OS primitives to disable and enable interrupt.
- **Problems:**
  - allow the power of interrupt handling to the user.
  - The chances of never turned on **is a disaster.**
  - it only works in **single processor environment.**

# Solution 2(Busy waiting) Lock Variables

- A single, shared (lock) variable, initially 0.
- When a process wants to enter its CR, it first **test the lock**. If the **lock** is 0, the process set it to 1 and enters the CR. If the **lock** is already 1, the process just waits until it becomes 0.
- The general structure of process is
- Do{

```
    While(lock!=0); //busy waiting
        Lock=1;
            //perform the CR task
        Lock=0
            //perform Non CR
            task
    }while(true)
```

# Facts

- **Advantages:** seems no problems.
- Problems:
  - problem like spooler directory;
  - suppose that one process reads the lock and sees that it is 0, before it can set lock to 1, another process scheduled, enter the CR, set lock to 1 and can have two process at CR (violates mutual exclusion)

# Strict Alternation

- Processes share a common integer variable *turn*.

*If turn == i*

*Then process Pi is allowed to execute in its CR,*

*if turn == j*

*then process Pj is allowed to execute.*

# Strict Alternation

```
While (true){ while(turn!  
=i); /*loop*/  
    critical_section();  
turn = j;  
    noncritical_sectio  
n  
}
```

Structure of Process Pi

```
While (true){ while(turn!  
=j); /*loop*/  
    critical_section();  
turn = i;  
    noncritical_sectio  
n  
}
```

Structure of Process Pj

# Facts

*Ensures that only one process at a time can be in its CR.*

- Problems: strict alternation of processes in the execution of the CR.
  - What happens if process i just finished CR and again need to enter CR and the process j is still busy at non-CR work? (violate condition 2)

# Peterson's solution

```
#define FALSE 0
#define TRUE 1
#define N      2           /* number of processes */

int turn;                  /* whose turn is it? */
int interested[N];         /* all values initially 0 (FALSE) */

void enter_region(int process); /* process is 0 or 1 */
{
    int other;               /* number of the other process */

    other = 1 - process;    /* the opposite of process */
    interested[process] = TRUE; /* show that you are interested */
    turn = process;          /* set flag */
    while (turn == process && interested[other] == TRUE) /* null statement */ ;
}

void leave_region(int process) /* process: who is leaving */
{
    interested[process] = FALSE; /* indicate departure from critical region */
}
```

# Facts about Petersons

- Before entering its CR, each process call `enter_region` with its own process number, 0 or 1 as parameter.
  - Call will cause to wait, if need be, until it is safe to enter.
  - When leaving CR, the process calls `leave_region` to indicate that it is done and to allow other process to enter CR.
- Advantages: Preserves all conditions.
- Problems: difficult to program for n-processes system and less efficient.

# Hardware Assistance(TSL)

- Checking for mutual exclusion is also possible through Hardware.
- Specific instruction called Test and Set Lock(TSL) is used for this purpose.
- The special properties of these instruction is that they cant be interrupted i.e. they are atomic instruction and carried out with out interruption.

# TSL instruction

- Test and Set Lock (TSL) instruction reads the contents of the memory word lock (shared variable) into the register and then stores nonzero value at the memory address lock.
- This automatically as in a un-interruptible time unit.
- The CPU executing TSL locks the memory bus to prohibit other CPUs from accessing memory until it is done.
- When lock is 0, any process may set it to 1 using the TSL instruction.
- When it is done the process lock back to 0.

# Alternate to Busy waiting

## Busy waiting:

When a process want to enter its CR, it checks to see if the entry is allowed, if it is not, the process just sits in a tight loop waiting until it is.

## Waste of CPU time for NOTHING!

- Possibility of Sleep and Wakeup pair instead of waiting.
- Sleep causes to caller to block, until another process wakes it up.

# Sleep and week up Consumer Producer problem

---

- Two process share a common, fixed sized buffer.
- Suppose one process, producer, is generating information that the second process, consumer, is using.
  - Their speed may be mismatched, if producer insert item rapidly the buffer will full and go to the sleep until consumer consumes some item, while consumer consumes rapidly, the buffer will empty and go to sleep until producer put something in the buffer.

# Conti.

```
#define N = 100 /* number of slots in the buffer*/
int count = 0; /* number of item in the buffer*/
```

```
void producer(void)
```

```
{  
    int item;  
    while(TRUE)  
    { /* repeat  
        forever*/
```

```
        it  
        e  
        m  
        =  
        p  
        r  
        o  
        d  
        u  
        c
```

```
}
```

# Cont.....

```
void consumer(void)
{
    int item;
    while(TRUE)
    { /* repeat
        forever */
        if(coun
            t == 0)
            sleep
            (); /*

            if
            buffe
            r is
            emp
            ty go
            to
            sleep
        */
    }
}
```

# Producer Consumer with weak up and sleep signals

- **Problem:**
  - leads to race as in Lock variable.
  - **What happen if**
    - When the buffer is empty, the consumer just reads count and quantum is expired, the producer inserts an item in the buffer, increments count and wake up consumer.
    - The consumer not yet asleep, so the wakeup signal is lost, the consumer has the count value 0 from the last read so go to the sleep.
    - Producer keeps on producing and fill the buffer and go to sleep,  
**both will sleep forever.**

**Think: If we were able to save the wakeup signal that was lost.....(semaphores)**

# Semaphores

- E. W. Dijkstra (1965) suggested using an integer variable to count the number of wakeups, called a semaphore.
- It could have the value 0, indicating no wakeups were saved, or some positive value if one or more wakeups were pending.
- **Operations: Down and Up** (originally he proposed P and V in Dutch and sometimes known as wait and signal)
  - Down: checks if the value greater than 0
  - yes- decrement the value (i.e. Uses one stored wakeup) and continues.
  - No- process is put to sleep without completing down.
- **Checking value, changing it, and possibly going to sleep, is all done as single action.**
  - Up: increments the value; if one or more processes were sleeping, unable to complete earlier down operation, one of them is chosen and is allowed to complete its down.

# Semaphore Operations

```
void down(S)
{
    if(S> 0)
        S--;
    else
        sleep(); //wait until s>0
}
void up (S)
{
    if(one or more processes are sleeping on S)
        one of these process is proceed;
    else
        S++;
}
```

# Semaphore: Discussions

- To avoid busy waiting, a semaphore may have an associated waiting queue of process.
- If process does a down operation on a semaphore which is zero, the process is added to the semaphore's queue.
- When another process increments the semaphore by doing up operation and there are tasks in the queue, one is taken off and resumed.

# Consumer producer using Semaphore

```
#define N 100 /*number of slots in buffer*/
typedef int semaphore; /*defining semaphore*/
semaphore mutex = 1; /* control access to the CR */
semaphore empty = N /*counts empty buffer slots*/
semaphore full = 0; /*counts full buffer slots*/

void producer(void)
{
    int item;
    while(TRUE){ /*repeat forever */
        item = produce_item(); /*generate something */
        down(empty); /*decrement empty count*/
        down(mutex); /*enter CR */
        insert_item(); /* put new item in buffer*/
        up(mutex); /* leave CR*/
        up(full); /*increment co_u_n_t_o_f_f_u_l_l
                     _s_l_o_t_s */
    }
}
```

# Consumer producer using Semaphore

```
void consumer(void)
{
    int item;
    while(TRUE){ /*repeat forever*/
        down(full); /*decrement full count */
        down(mutex); /*enter CR*/
        item = remove_item(); /*take item
from buffer*/
        up(mutex); /*leave CR*/
        up(empty); /*increment count of empty slots*/
        consume_item(); /*print item*/
    }
}
```

# Semaphore's attraction

- They have the following attractive properties
  - Machine independent
  - Simple
  - Powerful: provide exclusion and waiting
  - Work with many process
  - Can ask for many resources simultaneously with multiple down operations

# Message Passing

- With the trend of distributed operating system, many OS are used to communicate through Internet, intranet, remote data processing etc.
- Inter process communication based on two primitives operations:
  - send
  - and receive.
- The general syntax for these operations looks like
  - send(destination, &message);
  - receive(source, &message);
- The send and receive calls are normally implemented as operating system calls accessible from many programming language environments.

# Producer-Consumer with Message Passing

```
#define N 100 /*number of slots in the buffer*/\n\n    void producer(void)\n    { int item;\n        message m; /*message buffer*/\n        while (TRUE){\n            item = produce_item(); /*generate something */\n            receive(consumer, &m); /*wait for an empty to arrive*/\n            build_message(&m, item); /*construct a message to send*/\n            send(consumer, &m); }\n    }\n\n    void consumer(void)\n    { int item;\n        message m;\n        for(i = 0;\n            i<N; i++)\n            send(producer,\n                &m);\n        /*send N\n         *empties*/\n        while(TRUE){\n            receive(consumer, &m); /*get message containing item*/\n            item = extract_item(m);\n            process(item);\n        }\n    }
```

# Message passing

- No shared memory.
- Messages sent but not yet received are buffered automatically by OS, it can save N messages.
- The total number of messages in the system remains constant, so they can be stored in given amount of memory known in advance.
- Implementing Message Passing can be with:
  - Direct addressing: provide ID of destination.
  - Indirect addressing: Send to a mailbox.
- **Mail box:** It is a message queue that can share by multiple senders and receivers, senders sends the message to the mailbox while the receiver pick ups the message from the mailbox.

# Monitors

- Higher level synchronization primitive.
- A monitor is a programming language construct that guarantees appropriate access to the CR.
- It is a collection of procedures, variables , and data structures that are all grouped together in a special kinds of module or package.
- Processes that wish to access the shared data, do through the execution of monitor functions.
- Only one process can be active in a monitor at any instant.

# Monitors: discussion

- A procedure defined within a monitor can access only those variables declared locally within the monitors.
- A **monitor** is an object or module intended to be used safely by more than one process.
- The defining characteristic of a monitor is that its methods are executed with mutual exclusion.
- That is, at each point in time, at most one process may be executing any of its methods.
- This mutual exclusion greatly simplifies reasoning about the implementation of monitors compared to reasoning about parallel code that updates a data structure.

# The Syntax of Monitors

Monitor monitor\_name

{

shared variable declarations;

procedure p1(){ ..... }

procedure p2(){ .... }

.....

procedure pn(){ ...}

{initialization code;}

}

# Homework 3

- Discuss intercrosses communication in details
- What is critical section and critical section problem?
- What are semaphores? Explain.
- What are monitors? Explains
- Show the Peterson's algorithm preserve mutual exclusion, indefinite postponement and progress (deadlock)

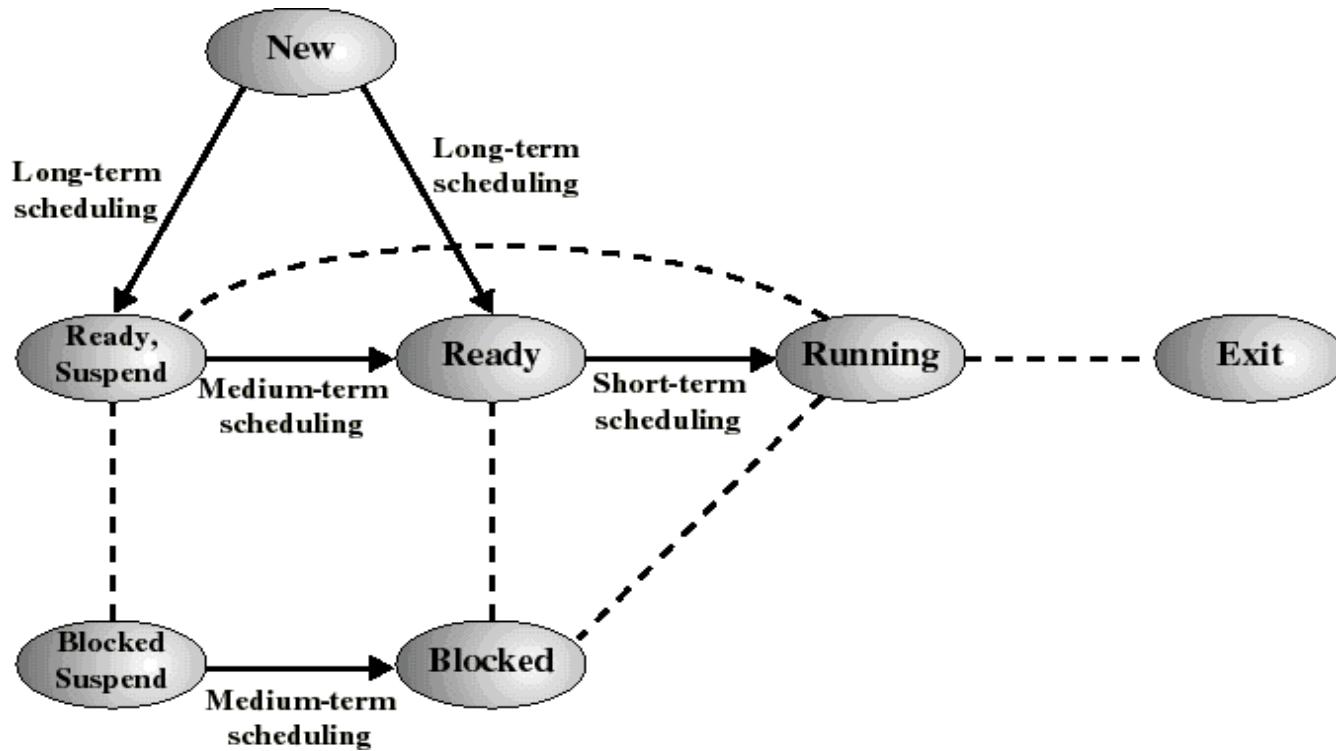
# **Process Scheduling**

Unit 1.2-Chapter 2

# CPU Scheduling

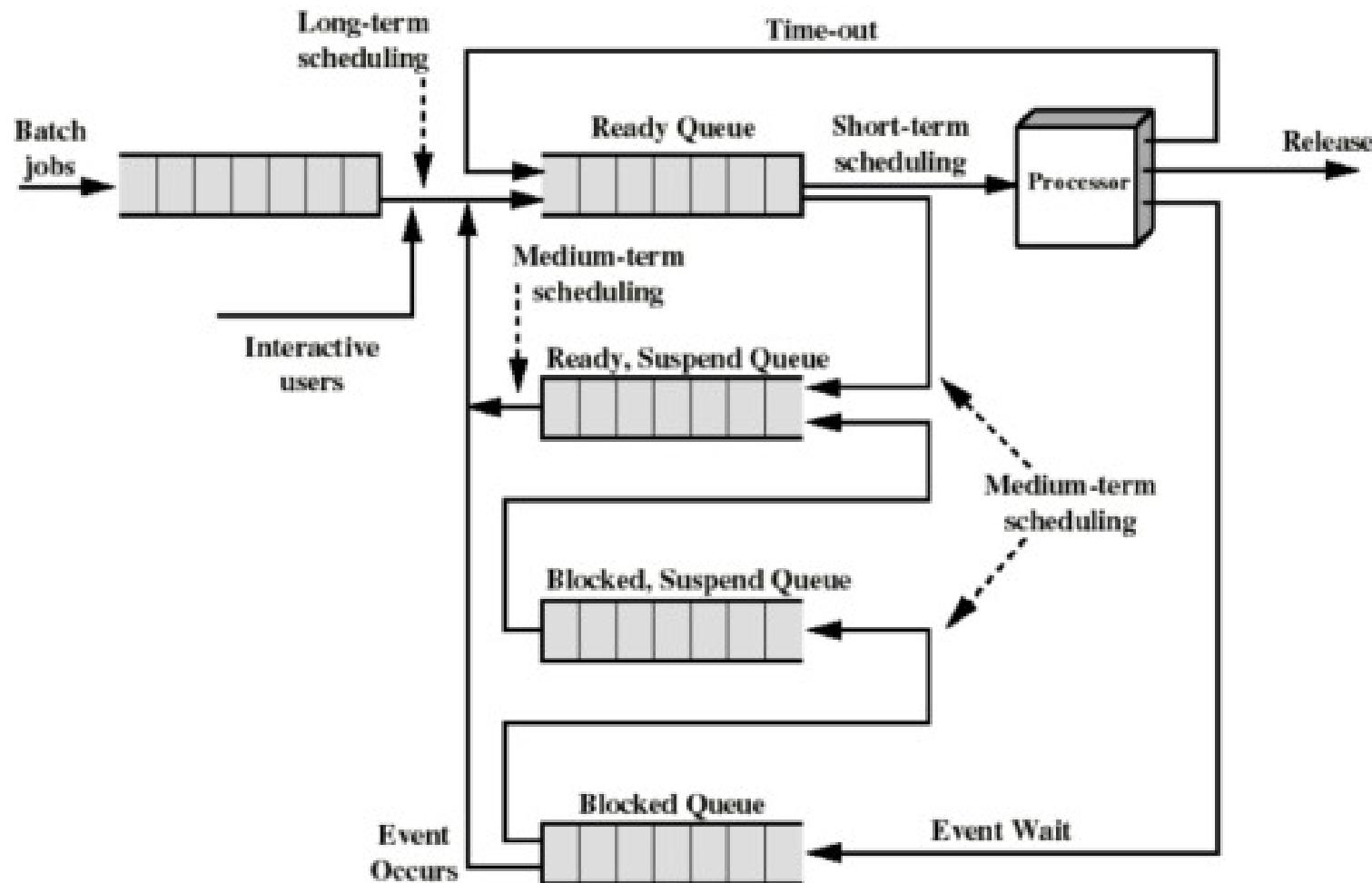
- Scheduling the processor among all ready processes
- The goal is to achieve:
  - High processor utilization
  - High throughput
    - number of processes completed per unit time
  - Low response time
    - time elapsed from the submission of a request until the first response is produced

# Classification of Scheduler



- **Long-term**: which process to admit?
- **Medium-term**: which process to swap in or out?
- **Short-term**: which ready process to execute next?

# Queuing Diagram for Scheduling



# Long-Term Scheduler

---

- Determines which programs are admitted to the system for processing
- Controls the degree of multiprogramming
- Attempts to keep a balanced mix of processor-bound and I/O-bound processes
  - CPU usage
  - System performance

# Medium-Term Scheduler

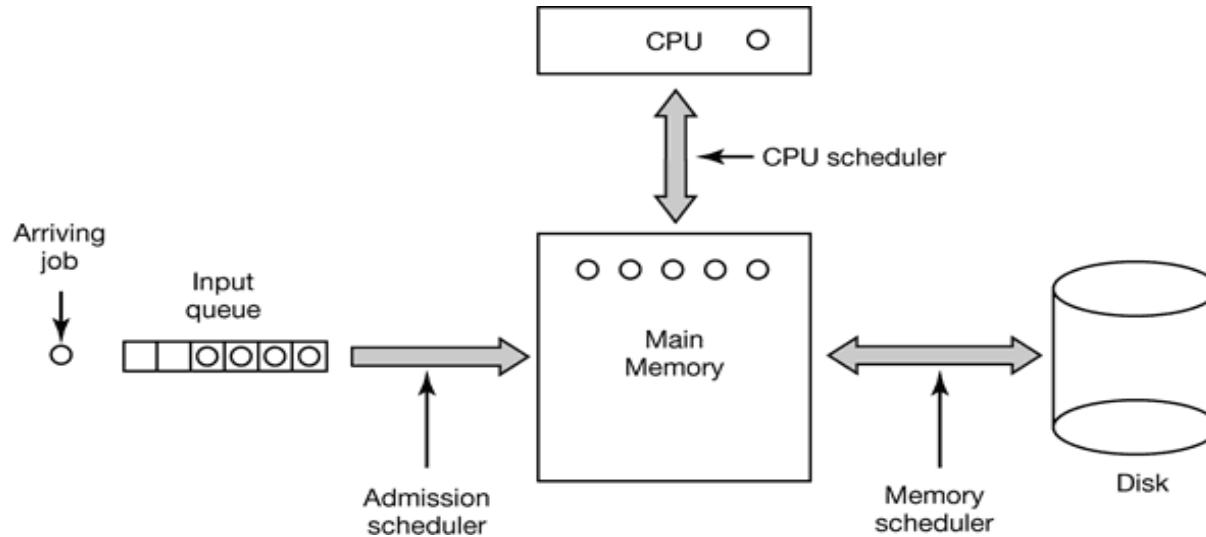
- Makes swapping decisions based on the current degree of multiprogramming
  - Controls which remains resident in memory and which jobs must be swapped out to reduce degree of multiprogramming

# Short-Term Scheduler

- Selects from among ready processes in memory which one is to execute next
  - The selected process is allocated the CPU
- It is invoked on events that may lead to choose another process for execution:
  - Clock interrupts
  - I/O interrupts
  - Operating system calls and traps
  - Signals

# Three level scheduling

- From a certain perspective, batch systems allow scheduling at three different levels, as illustrated in Fig
- As jobs arrive at the system, they are initially placed in an input queue stored on the disk.



# Three level Scheduling

- The **admission scheduler (long term scheduler)** decides which jobs to admit to the system. The others are kept in the input queue until they are selected.
- A typical algorithm for admission control might be to look for a mix of compute-bound jobs and I/O-bound jobs. Alternatively, short jobs could be admitted quickly whereas longer jobs would have to wait.
- The admission scheduler is free to hold some jobs in the input queue and admit jobs that arrive later if it so chooses.

# Three level scheduling(Cont)

- Once a job has been admitted to the system, a process can be created for it and it can contend for the CPU.
- However, it might well happen that the number of processes is so large that there is not enough room for all of them in memory. In that case, some of the processes have to be swapped out to disk.
- The second level of scheduling is deciding which processes should be kept in memory and which ones kept on disk.
- We will call this scheduler the **memory scheduler (short term scheduler)**, since it determines which processes are kept in memory and which on the disk.

# Three level of scheduling(cont.)

- The third level of scheduling is actually picking one of the ready processes in main memory to run next.
- Often this is called the **CPU scheduler** and is the one people usually mean when they talk about the “scheduler.”
- Any suitable algorithm can be used here, either preemptive or non-preemptive.

# Classification of Scheduling(CPU) Algorithms

- The **selection function** determines which ready process is selected next for execution
- The **decision mode** specifies the instants in time the selection function is exercised
  - **Nonpreemptive**
    - Once a process is in the running state, it will continue until it terminates or blocks for an I/O
  - **Preemptive**
    - Currently running process may be interrupted and moved to the Ready state by the OS
    - Prevents one process from monopolizing the processor

# Scheduling Criteria

- **User-oriented criteria**
  - **Response Time**: Elapsed time between the submission of a request and the receipt of a response
  - **Turnaround Time**: Elapsed time between the submission of a process to its completion
- **System-oriented criteria**
  - Processor utilization
  - Throughput: number of process completed per unit time
  - fairness

# Scheduling Algorithms

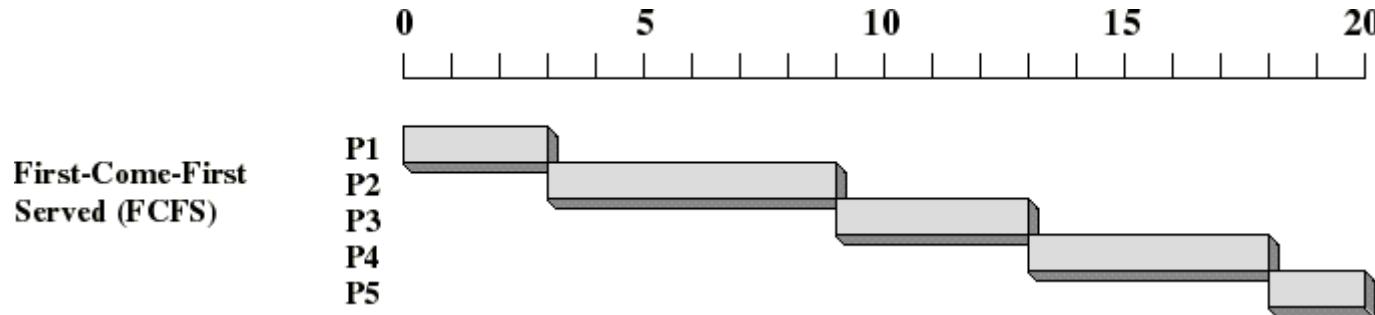
- First-Come, First-Served(FCFS)
- Shortest-Job-First(SJF) Scheduling
  - Also referred to as Shortest Process Next(SPN)
- Priority Scheduling
- **Round-Robin Scheduling**
- Multilevel Queue Scheduling

# Process Mix Example

Process	Arrival Time	Service Time
1	0	3
2	2	6
3	4	4
4	6	5
5	8	2

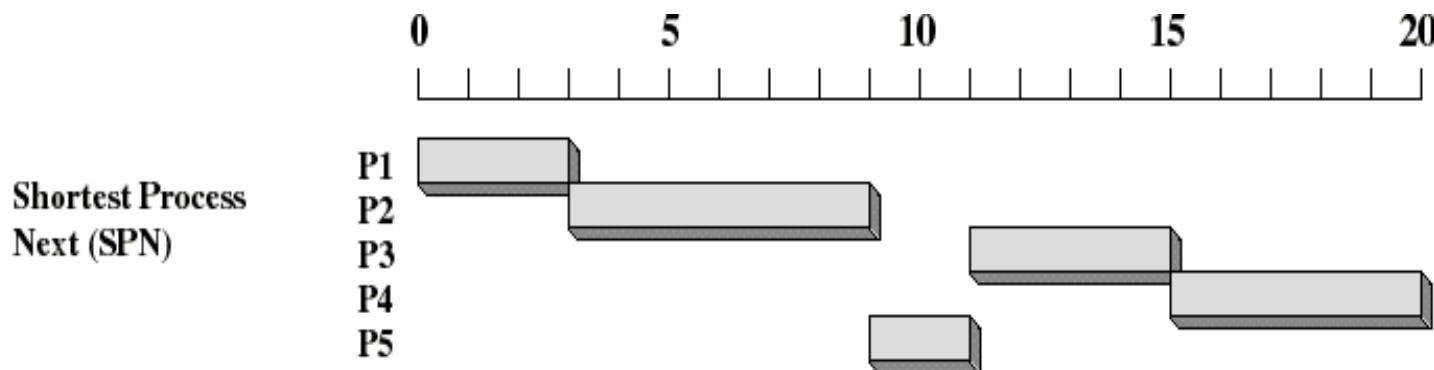
**Service time = total processor time needed in one (CPU-I/O) cycle**  
**Jobs with long service time are CPU-bound jobs and are referred to as “long jobs”**

# First Come First Served (FCFS)



- Selection function: the process that has been waiting the longest in the ready queue (hence, FCFS)
- Decision mode: **non-preemptive**
  - a process runs until it blocks for an I/O

# Shortest Job First (Shortest Process Next)

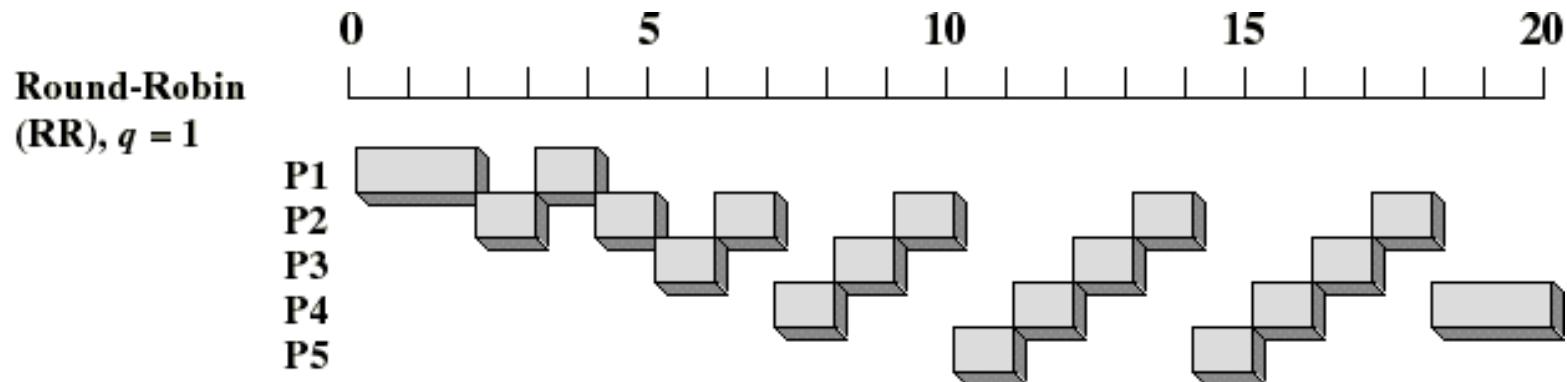


- Selection function: the process with the shortest expected CPU burst time
  - I/O-bound processes will be selected first
- Decision mode: **non-preemptive**
- The required processing time, i.e., the CPU burst time, must be estimated for each process

# Priorities

- Implemented by having multiple ready queues to represent each level of priority
- Schedule the process of a higher priority over one of lower priority
- Lower-priority may suffer starvation
- To alleviate starvation allow dynamic priorities
  - The priority of a process changes based on its age or execution history

# Round-Robin



- Selection function: same as FCFS
  - Decision mode: preemptive
- ◆ a process is allowed to run until the time slice period (quantum, typically from 10 to 100 ms) has expired
- ◆ a clock interrupt occurs and the running process is put on the ready queue

# RR Time Quantum

- Quantum must be substantially larger than the time required to handle the clock interrupt and dispatching
- Quantum should be larger then the typical interaction
  - but not much larger, to avoid penalizing I/O bound processes

# Round Robin: critique

- Still favors CPU-bound processes
  - An I/O bound process uses the CPU for a time less than the time quantum before it is blocked waiting for an I/O
  - A CPU-bound process runs for all its time slice and is put back into the ready queue
    - May unfairly get in front of blocked processes

# Round Robin Variants

- Weighted RR
- RR with variable quanta



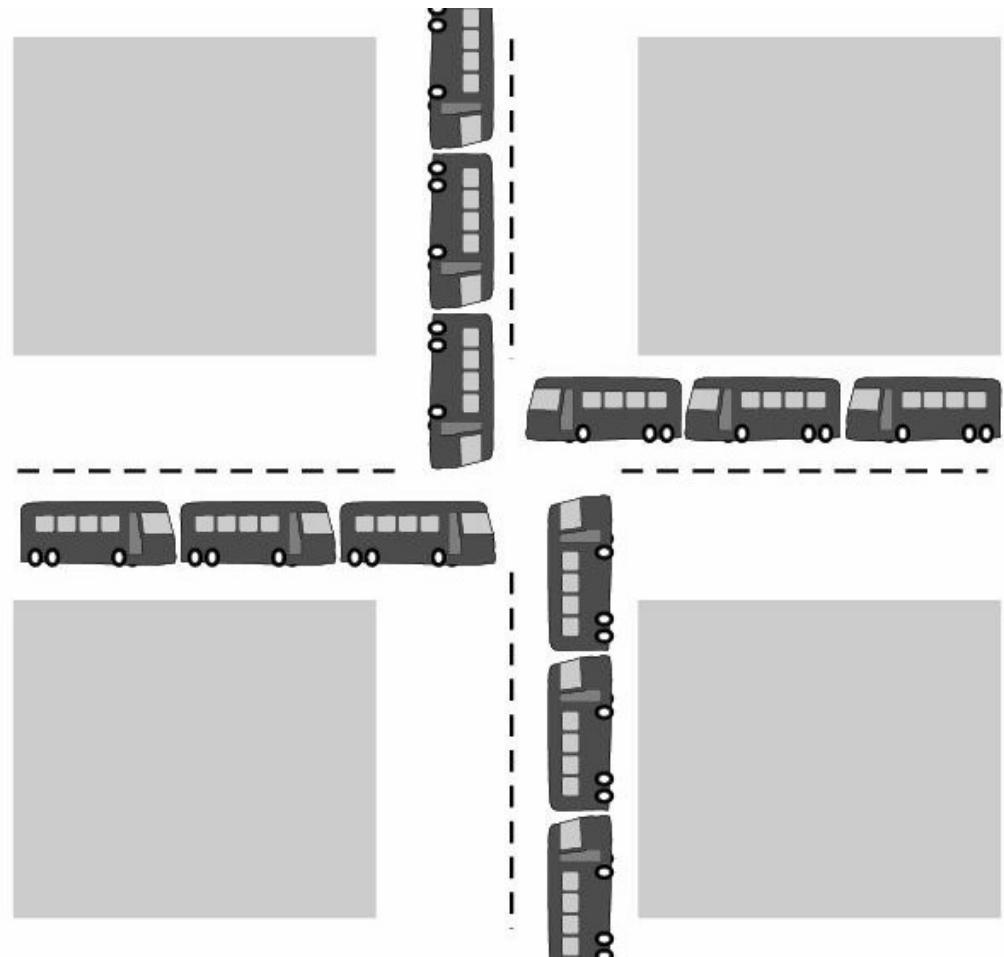
# Thank you very much

# Deadlock

A process in a multiprogramming system is said to be in dead lock if it is waiting for a particular event that will never occur.

# Deadlock Example

- All automobiles trying to cross.
- Traffic Completely stopped.
- Not possible without backing some.

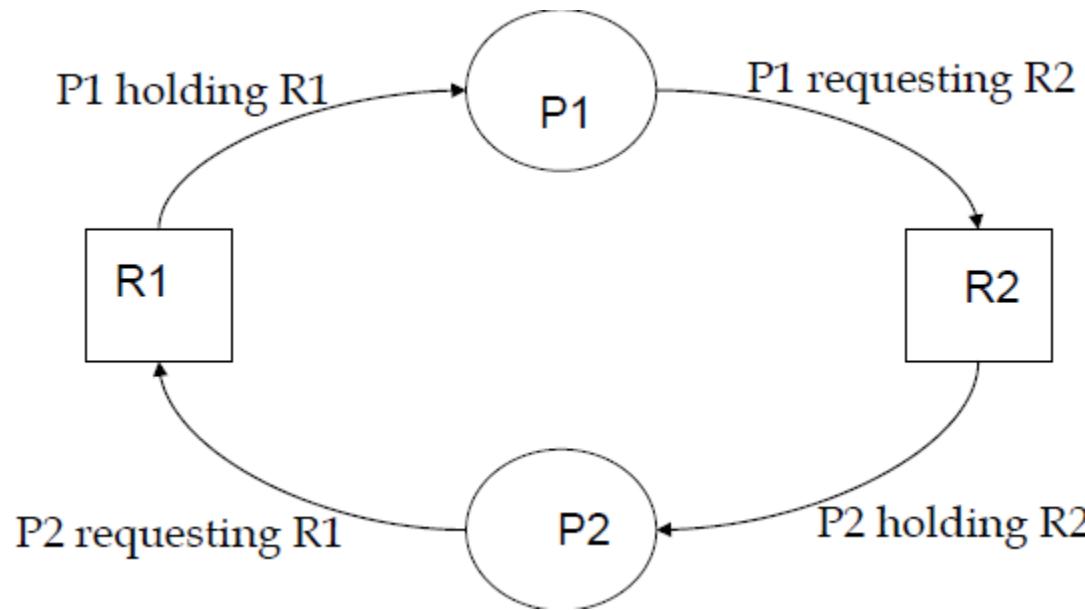


# Resource Deadlock

*A process request a resource before using it, and release after using it.*

- Request the resource.
  - Use the resource.
  - Release the resource.
- 
- If the resource is not available when it is requested, the requesting process is forced to wait.
  - Most deadlocks in OS developed because of the normal contention for dedicated resources.

# Resource Deadlock



**Process P1 holds resource R1 and needs resource R2 to continue;  
Process P2 holds resource R2 and needs resource R1 to continue  
-deadlock.**

***Key: Circular wait -deadlock***

# Condition for Deadlock

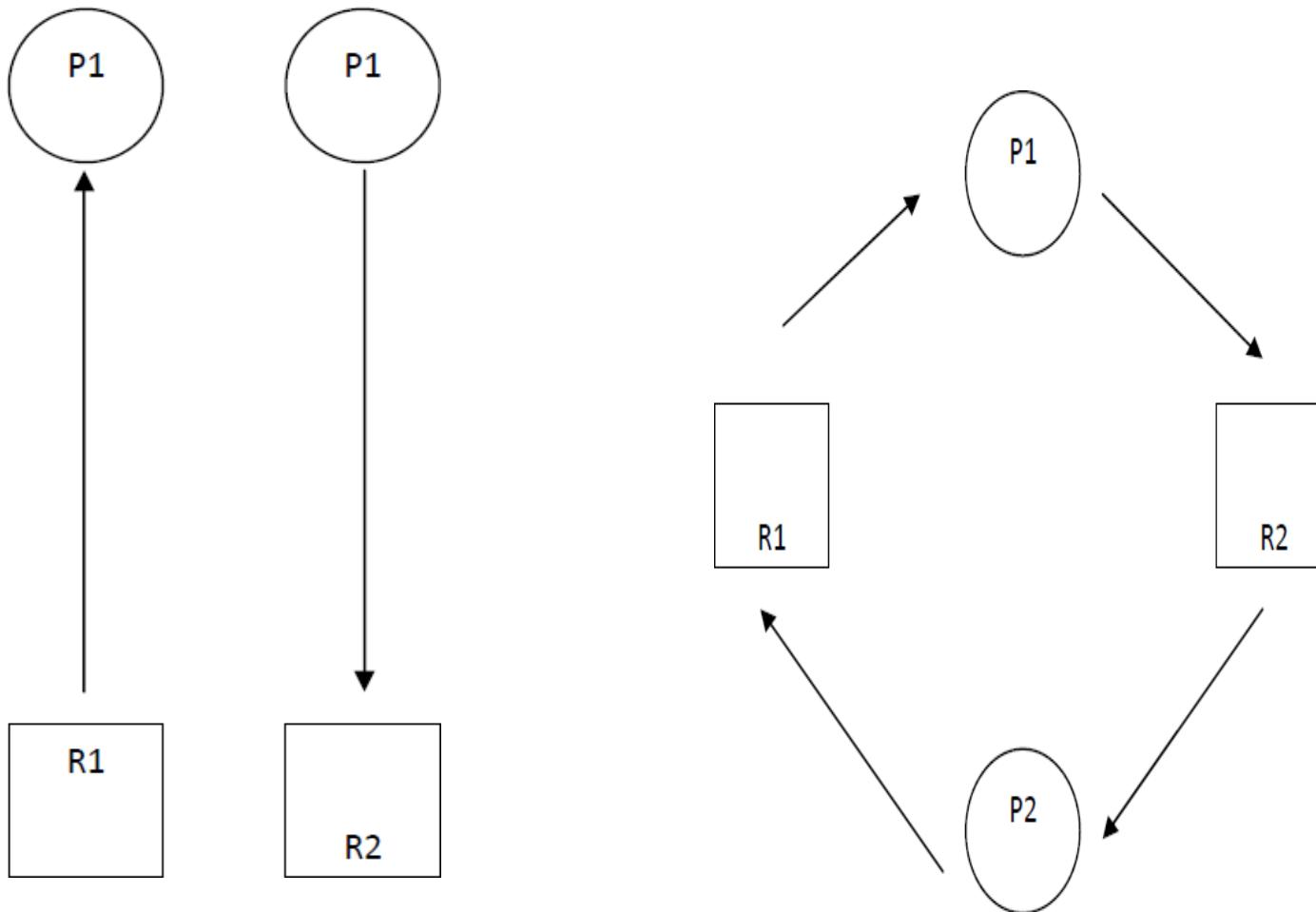
- **Mutual Exclusion:** Process claims exclusive control of resources they require.
- **Hold and Wait:** Processes hold resources already allocated to them while waiting for additional resources.
- **No preemption:** Resources previously granted can not be forcibly (or compulsorily) taken away from the process.
- **Circular wait:** Each process holds one or more resources that are requested by next process in the chain.

A deadlock situation can arise if all four conditions hold **simultaneously** in the system.

# Resource Allocation Graph (RAG)

- Deadlock can be described more precisely in terms of a directed graph called a system resource-allocation graph.
- It Consist of Set of vertices and set of edges.
- The set of vertices  $V$  is partitioned into two types of nodes:
  - $P=\{p_1, p_2, \dots, p_n\}$ , the set of process in the system.
  - $R=\{R_1, R_2, R_3, \dots, R_n\}$ , the set of resource types in the system.

# Example



# Method for Handling Deadlock

- We can use a protocol to *prevent or avoid deadlocks, ensuring that the system never enter a deadlock state.*
- We can allow the system to enter a deadlock state, *detect it, and recover.*
- We can ignore the problem altogether, and pretended that deadlock never occur in the system.

# Deadlock Prevention

*Idea: If any one of the four necessary conditions is denied, a deadlock can not occur.*

By implementing policy that makes one of the conditions impossible, the system will be assured to be deadlock free.

# First : Denying Mutual Exclusion

*Sharable resources do not require mutually exclusive access such as read only shared file.*

**But!!! Some resources are strictly non-sharable, mutually exclusive control required.**

*So We can not prevent deadlock by denying the mutual exclusion*

# Second: Denying Hold and Wait:

- Policy:
  - Resources grant on *all or none basis*;
  - If all resources needed for processing are available then granted and allowed to process.
  - If complete set of resources is not available, the process must wait set available.
  - While waiting, the process should not hold any resources.
- Problem:
  - Low resource utilization.
  - Starvation is possible. A process that needs several popular resources may have to wait indefinitely, because at least one of the resources that it needs is always allocated to some other process.

# Third: Denying No-preemption

- policy
  - When a process holding resources is denied a request for additional resources, that process must release its held resources and if necessary, request them again together with additional resources.
- Problem:
  - When process releases resources the process may loose all its works to that point .

# Denying Circular Wait:

- Policy
  - All resources are uniquely numbered, and processes must request resources in linear ascending order.
  - The only ascending order prevents the circular wait .
- Problem:
  - Difficult to maintain the resource order; dynamic update in addition of new resources.

# Deadlock Avoidance

Deadlock-prevention algorithms ensure that at least one of the necessary conditions for deadlock cannot occur and hence that deadlocks cannot hold. Possible side effects of preventing deadlocks **are low device utilization and reduced system throughput.**

## Alternative

- For example, in a system with one tape drive and one printer, the system might need to know that process P will request first the tape drive and then the printer before releasing both resources, whereas process Q will request first the printer and then the tape drive.
- Assign resource that leads to the system into safe state.

# Deadlock Avoidance

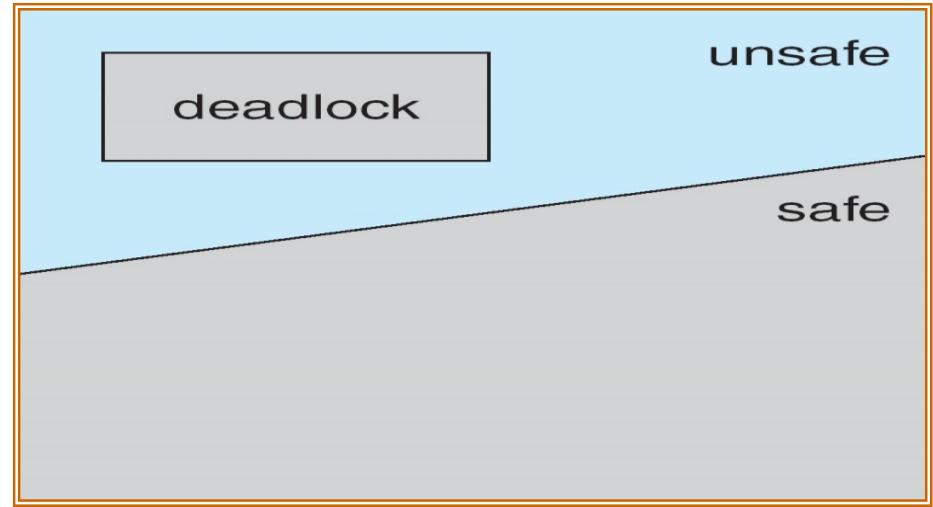
Each request requires that in making this decision the system consider:

- the resources currently available,
- the resources currently allocated to each process,
- the future requests and releases of each process.

Initially the system is in safe state. Whenever a process request resource that is currently available, the system must decide whether the resource can be allocated immediately or whether the process must wait.

# Safe state and Deadlock State

- If a system is in a safe state  $\Rightarrow$  No deadlocks.
- If a system is in unsafe state  $\Rightarrow$  Possibility of deadlock.
- Avoidance  $\Rightarrow$  ensure that a system will never reach an unsafe state.



# Example

- A state is said to be safe if it is not deadlocked and there is a some scheduling order in which every process can run to completion.
- To illustrate, we consider a system with 12 magnetic tape drives and three processes:

Process	Max	Has
P1	10	5
P2	4	2
P3	9	2

- Thus, there are 3 free tape drives.

# Continued

- At time  $t_0$ , the system is in a safe state. The sequence  $\langle p_2, p_1, p_3 \rangle$  satisfies the safety condition.
- A system can go from a safe state to an unsafe state. Suppose that, at time  $t_1$ , process  $p_3$  requests and is allocated one more tape drive. The system is no longer in a safe state.
- Our mistake was in granting the request from process  $p_3$  for one more tape drive.

Processes	Max.	Has
P1	10	5
P2	4	2
P3	9	2+1

# Deadlock Avoidance

## Banker's Algorithm

- Models on the way of banking system to ensure that the bank never allocates its available cash such that it can no longer satisfy the needs of all its customers.
- When a process request the set of resources, the system determine whether the allocation of these resources will left the system in safe state, if it will allocate the resources, otherwise process must wait until some other process release enough resources.
- Data structures: Available, Max, Allocation & Need.  
 $\text{need} = \text{max} - \text{allocation}$ .

# Banker's Algorithm

1. If a process is requesting more resource than its need, then raise an error
2. If request is valid, the number of resources requested is compared to the number of available resources.
3. If not enough resources are available, the request is denied and the process must wait.
4. If sufficient resources are available, the modified version of the current state **is created** that reflect the effect of granting request. The algorithm then checks if the modified version of the system is safe.
5. The resource is granted if and only if the modified version of the system state is safe.

It ensure that the system is never in unsafe state. so it avoid the deadlock

# Banker's Algorithm

allo. max

A	0	6
B	0	5
C	0	4
D	0	7

Available = 10

allo. max

A	1	6
B	1	5
C	2	4
D	4	7

Available = 2

allo. max

A	1	6
B	2	5
C	2	4
D	4	7

Available = 1

*Which one is unsafe?*

# Deadlock Avoidance Problem with Banker

- Algorithms requires fixed number of resources, some processes dynamically changes the number of resources.
- Algorithms requires the number of resources in advanced, it is very difficult to predict the resources in advanced.
- Algorithms predict all process returns within finite time, but the system does not guarantee it.

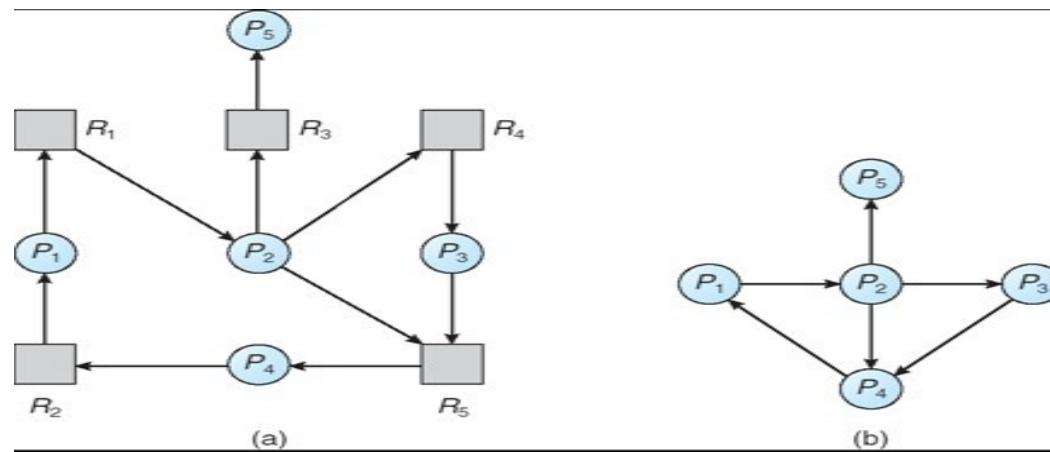
# Deadlock Detection and Recovery

Instead of trying to prevent or avoid deadlock,  
system allow to deadlock to happen,  
and recover from deadlock  
when it does occur.

Hence, the mechanism for deadlock detection  
and recovery from deadlock required.

# Deadlock Detection

- If all resource have only a single instance, then we can define a deadlock detection algorithm that use a variant of resource allocation graph, called wait for graph.
- Example



# Example

- Consider the following scenario:
- A system with 7 processes (A -G), and 6 resources (R -W) are in following state.
  1. Process A holds R and S.
  2. Process B holds nothing but wants T.
  3. Process C holds nothing but wants S.
  4. Process D holds U and wants S and T.
  5. Process E holds T and wants V.
  6. Process F holds W and wants S.
  7. Process G holds V and wants U.
  - *Is there any deadlock situation?*

# Deadlock Detection

- A deadlock exists in the system if and only if the wait for graph contains cycle.
- To detect deadlock the system needs to maintain the wait for graph and periodically invoke an algorithm that searches for a cycle in the graph.
- An algorithm to detect a cycle in graph requires an order of  $O(n^2)$

# Recovery from Deadlock

What do next if the deadlock detection algorithm succeed? -recover from deadlock.

- **By Resource Preemption**
  - Preempt some resources temporarily from a processes and give these resources to other processes until the deadlock cycle is broken.
- **Problem:**
  - Depends on the resources.
  - Need extra manipulation to suspend the process.
  - Difficult and sometime impossible.

# Recovery from Deadlock

- **By Process Termination**
  - Eliminating deadlock by killing one or more process in cycle or process not in cycle.
- Problem:
  - If the process was in the midst of updating file, terminating it will leave file in incorrect state.
- *Key idea: we should terminate those processes the termination of which incur the minimum cost.*

# Ostrich Algorithm

- Fact: there is no good way of dealing with deadlock.

Ignore the problem altogether

- For most operating systems, deadlock is a rare occurrence; So the problem of deadlock is ignored, like an ostrich sticking its head in the sand and hoping the problem will go away.

# Exercise

1. Is it possible to have a deadlock involving only one process?  
Explain
2. Consider the following snapshot of a system (given below)  
Answer the following questions using Banker's algorithm:
  - a) What is the content of the matrix **Need**?
  - b) Is the system in a safe state?
  - c) If a request from process P1 arrives for (0,4,2,0), can the request be granted immediately ?

Process	<u>Allocation</u>				<u>MAX</u>				<u>Available</u>			
	A	B	C	D	A	B	C	D	A	B	C	D
P0	0	0	1	2	0	0	1	2	1	5	2	0
P1	1	0	0	0	1	7	5	0				
P2	1	3	5	4	2	3	5	6				
P3	0	6	3	2	0	6	5	2				
P4	0	0	1	4	0	6	5	6				

# Exercise

3. A system has four processes and five allocable resources. The current allocation and maximum needs are given below:
- What is the smallest value of  $x$  **for which this is a safe state** based on the Banker's algorithm?

Process	Allocation					MAX					Available					
	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E	
P0	1	0	2	1	1	1	1	2	1	1	0	0	0	x	1	1
P1	2	0	1	1	0	2	2	2	1	0						
P2	1	1	0	1	0	2	1	3	1	0						
P3	1	1	1	1	0	1	1	2	2	1						

# Solutions

1. No.

This follows directly from the hold and wait condition.

Hold and wait: a process holding at least one resource is waiting to acquire additional resources held by other processes

Need				Process	Allocation				MAX				Available			
A	B	C	D		A	B	C	D	A	B	C	D	A	B	C	D
0	0	0	0	P0	0	0	1	2	0	0	1	2	1	1	0	0
0	7	5	0	P1	1	4	2	0	1	7	5	0				
1	0	0	2	P2	1	3	5	4	2	3	5	6				
0	0	2	0	P3	0	6	3	2	0	6	5	2				
0	6	4	2	P4	0	0	1	4	0	6	5	6				

2.

- Need = Max – Allocation. The matrix is shown bellow on the left-hand side;
- Yes, a sequence  $\langle P_3, P_4, P_2, P_1, P_0 \rangle$  is safe;
- Yes, there is a sequence  $\langle P_0, P_2, P_1, P_4, P_3 \rangle$  which is safe.

The snapshot of the system after resources allocation is given bellow on the right-hand side.

# Solution for 3

A: The matrix Need as shown on the right.

If  $x$  is 0, then we have an unsafe state immediately

If  $x$  is 1, then P3 can run, after it releases resources, P0 can run, after P2 and then P1, i.e. sequence  $\langle P3, P0, P2, P1 \rangle$  is safe, thus the system is in the safe state

Process	Need				
	A	B	C	D	E
P0	0	1	0	0	0
P1	0	2	1	0	0
P2	1	0	3	0	0
P3	0	0	1	1	1

Therefore the minimal value of  $x$  is 1

# **Paper Review on Deadlock**

By: Dr. Binod Kr. Adhikari

# Memory Management

The entire program and data of a process must be in main memory for the process to execute.

How to keep the track of processes currently being executed?

Which processes to load when memory space is available?

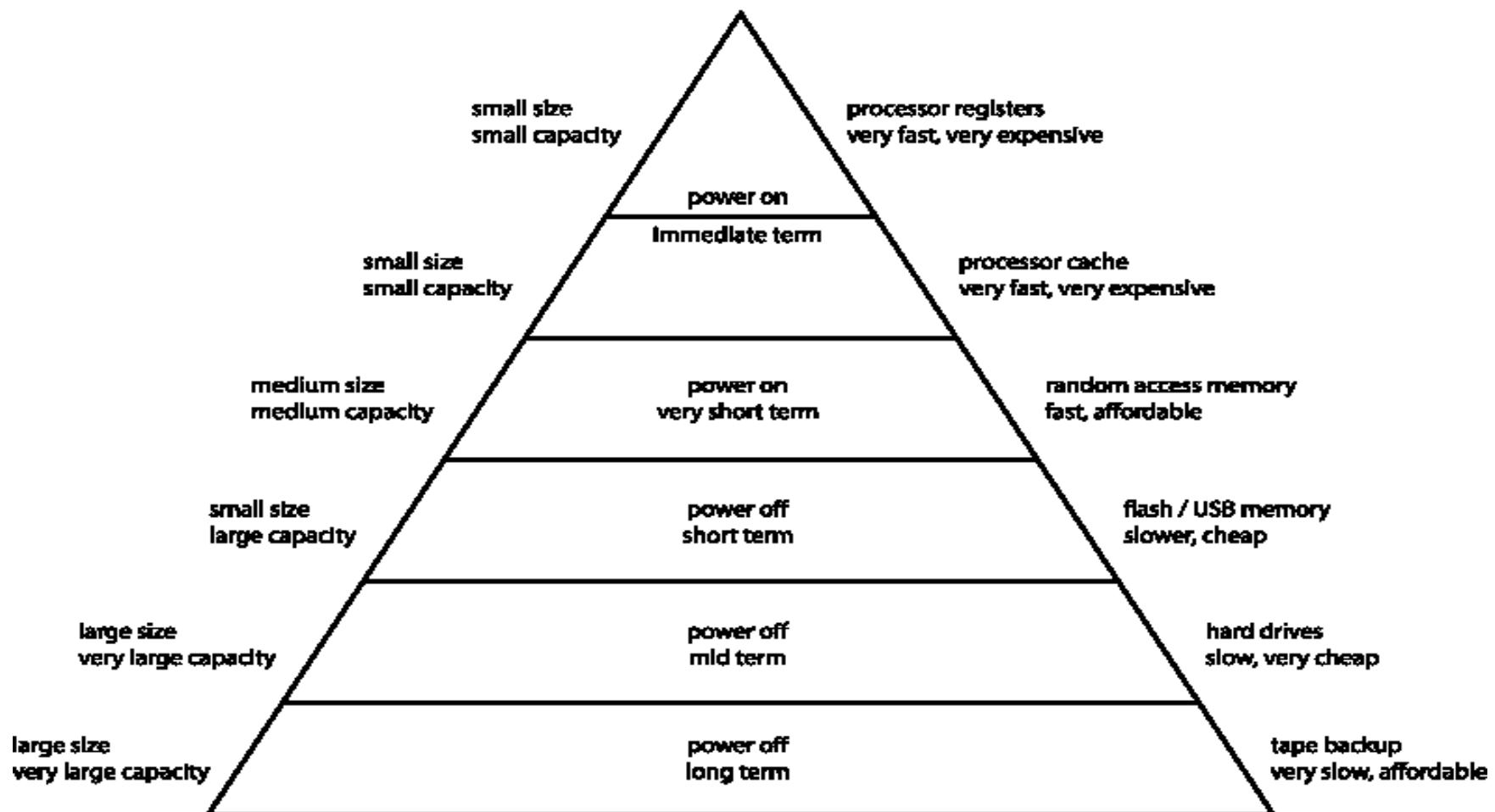
How to load the processes that are larger than main memory?

How do processes share the main memory?

**OS component that is responsible for handling these issues is a memory manager.**

# Memory Management

## Computer Memory Hierarchy



# Memory Manager

- The part of the operating system that manages the memory hierarchy is called the **memory manager**.
- Its job are:
  - to keep track of which parts of memory are in use and which parts are not in use,
  - to allocate memory to processes when they need it and de-allocate it when they are done,
  - to manage **swapping** between main memory and disk when main memory is too small to hold all the processes.

# Memory Management System

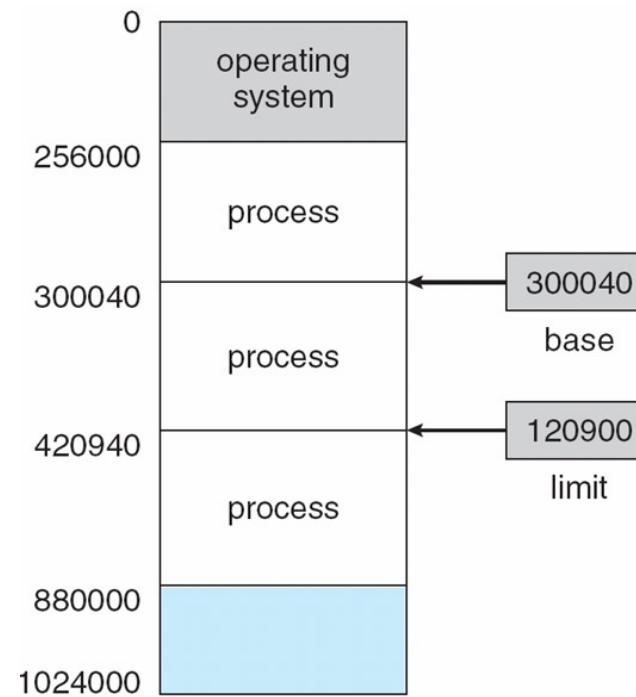
- Memory management systems can be divided into two classes:
  - Those that move processes back and forth between main memory and disk during execution (**swapping and paging**),
  - and those without swapping and paging.
- The latter are simpler, so we will study them first.

# Memory Addresses

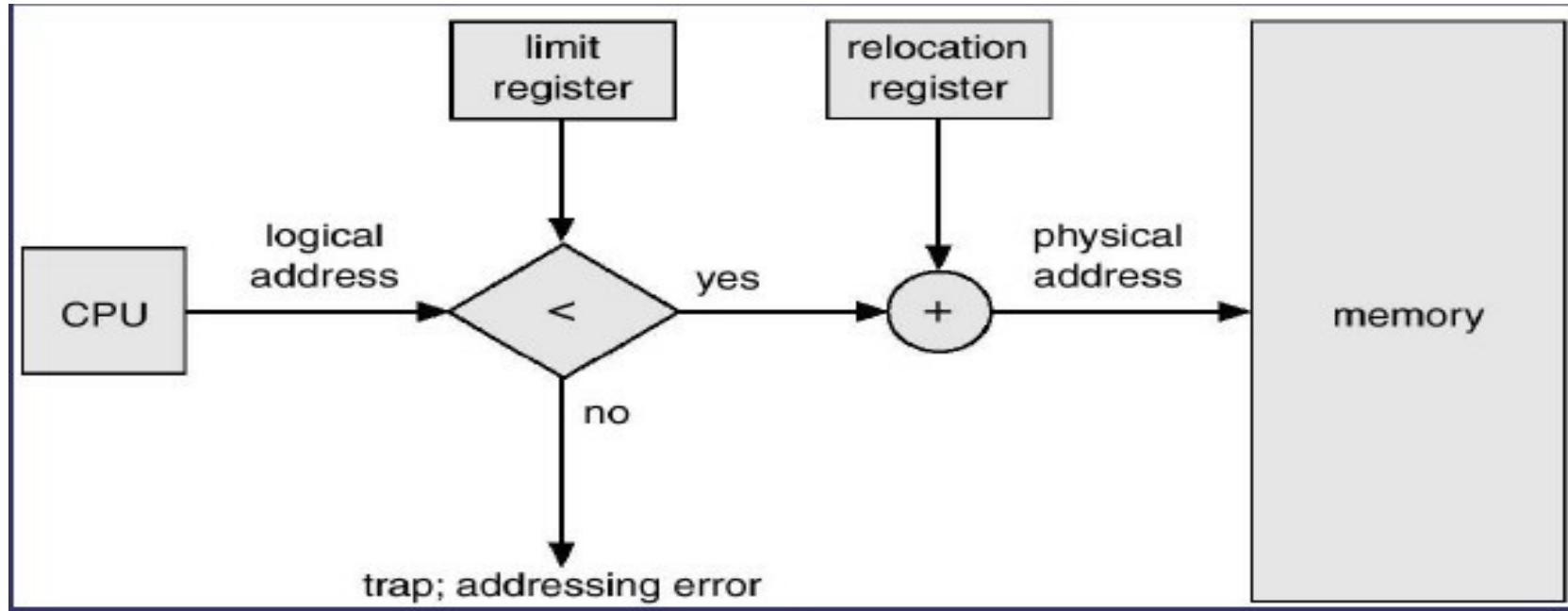
- Logical Address: The address generated by CPU. Also known as virtual address.
- Physical Address: Actual address loaded into the memory address register.
- Mapping from logical address to physical address is **relocation**.
- Two registers:
  - Base and Limit are used in mapping.
- This mechanism also used in memory protection .
  - memory outside the range are protected.

# Base and Limit Registers

- A pair of **base** and **limit registers** define the logical address space
- CPU must check every memory access generated in user mode to be sure it is between base and limit for that user



# Address Translation



- Base register (relocation): Holds smallest legal physical memory address.
- Limit register: contain the size of range.
- Example:
  - logical address = 300
  - if Base = 1500
    - physical address =  $1500 + 300 = 1800$ .

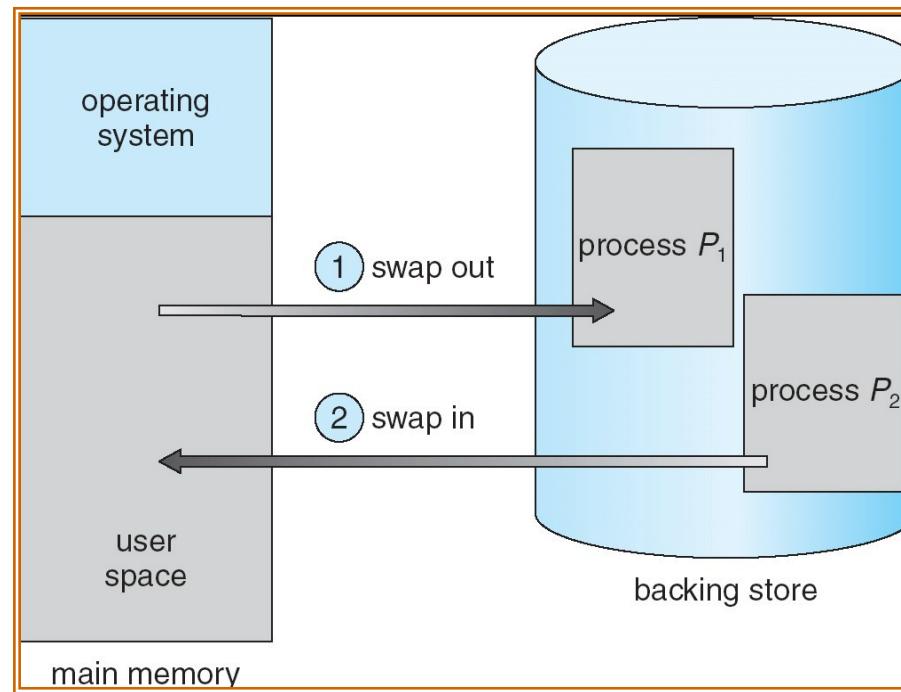
# Virtual Memory

- The basic idea behind the virtual memory is that the combined size of the program, data, and stack may exceed the amount of physical memory available for it.
- The OS keeps those part of the program currently in use in main memory, and the rest on the disk.

# Review: Swapping

- A process can be swapped temporarily out of memory to a backing store and then brought back into memory for continued execution.
  - Backing Store - fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images.
  - Roll out, roll in - swapping variant used for priority based scheduling algorithms; lower priority process is swapped out, so higher priority process can be loaded and executed.
  - 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 and Microsoft Windows.

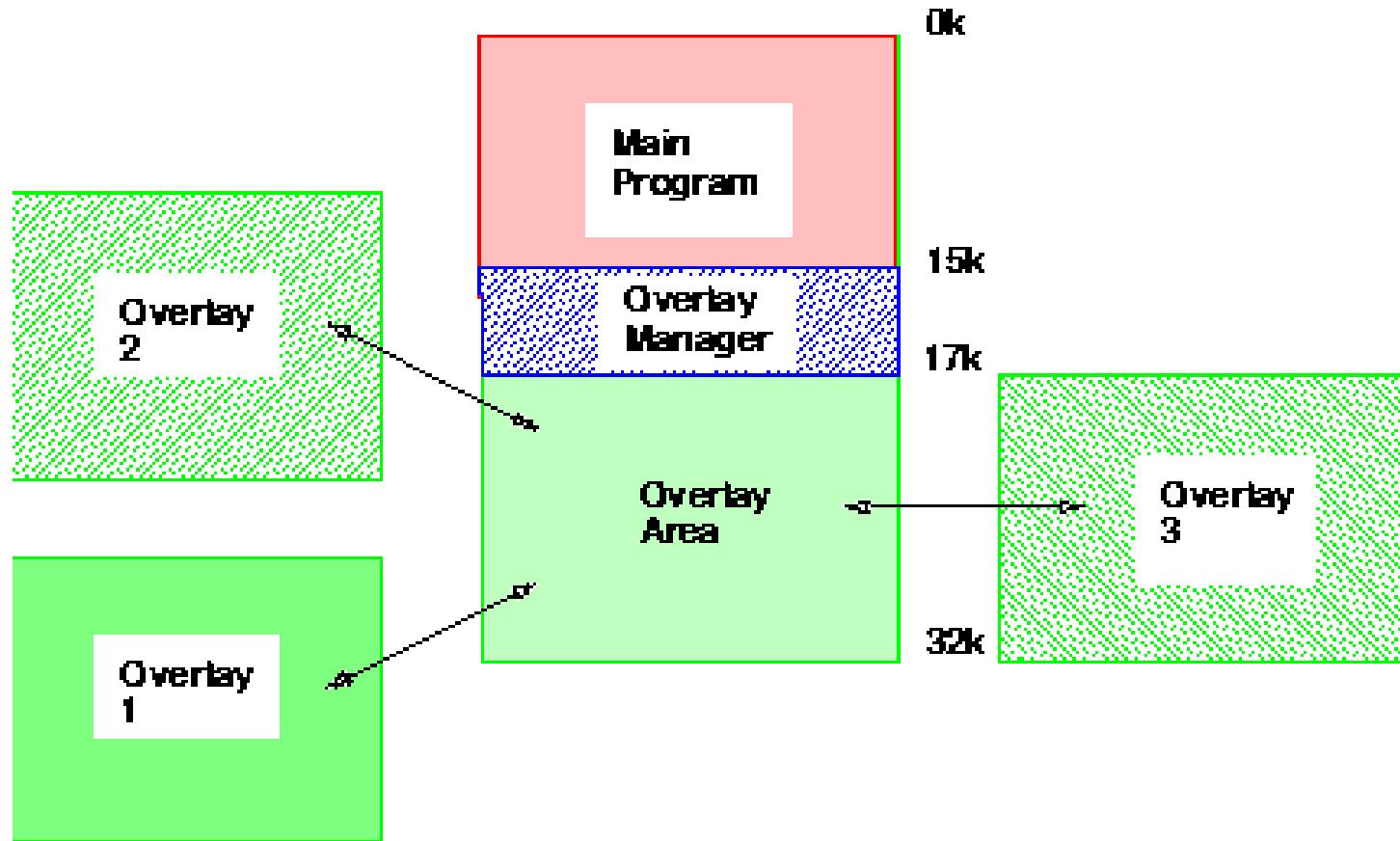
# Schematic view of swapping



# Review: Overlays

- Keep in memory only those instructions and data that are needed at any given time.
- Needed when process is larger than amount of memory allocated to it.
- Implemented by user, no special support from operating system; programming design of overlay structure is complex.

# Overlaying

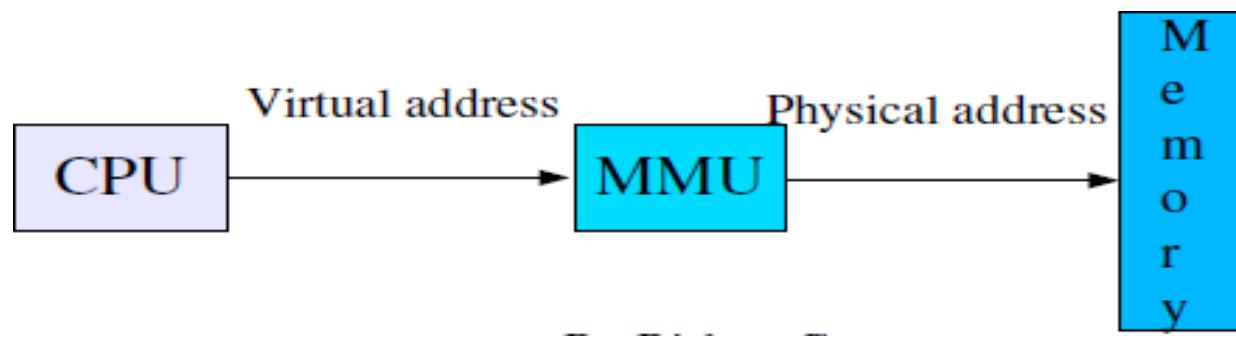


# Virtual Memory

- Virtual memory can be implemented by two most commonly used methods :
  - Paging
  - Segmentation
  - or mix of both.

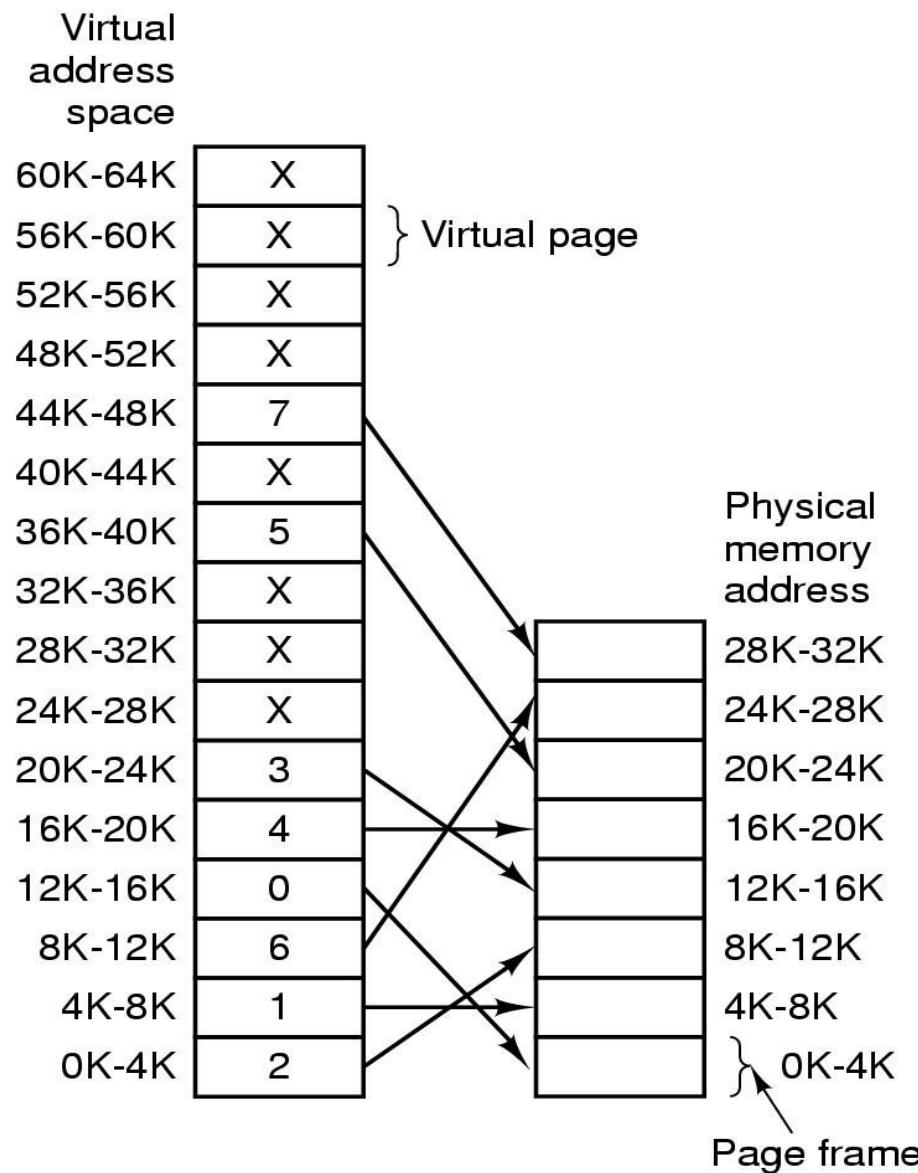
# Background MMU

- The run time mapping from virtual address to physical address is done by hardware devices called memory-management-unit (MMU).



# Paging

- Logical address space of a process can be non-contiguous;
  - process is allocated physical memory wherever the latter is available.
  - Divide physical memory into fixed size blocks called **frames**
    - size is power of 2, 512 bytes - 8K
  - Divide logical memory into same size blocks called **pages**.
    - Keep track of all free frames.
    - To run a program of size n pages, find n free frames and load program.
  - Set up a page table to translate logical to physical addresses.
  - Note:: Internal Fragmentation possible!!



This example shows 64KB program can run in 32KB physical Memory. The complete copy is stored in the disk and the pieces can be brought into memory as needed.

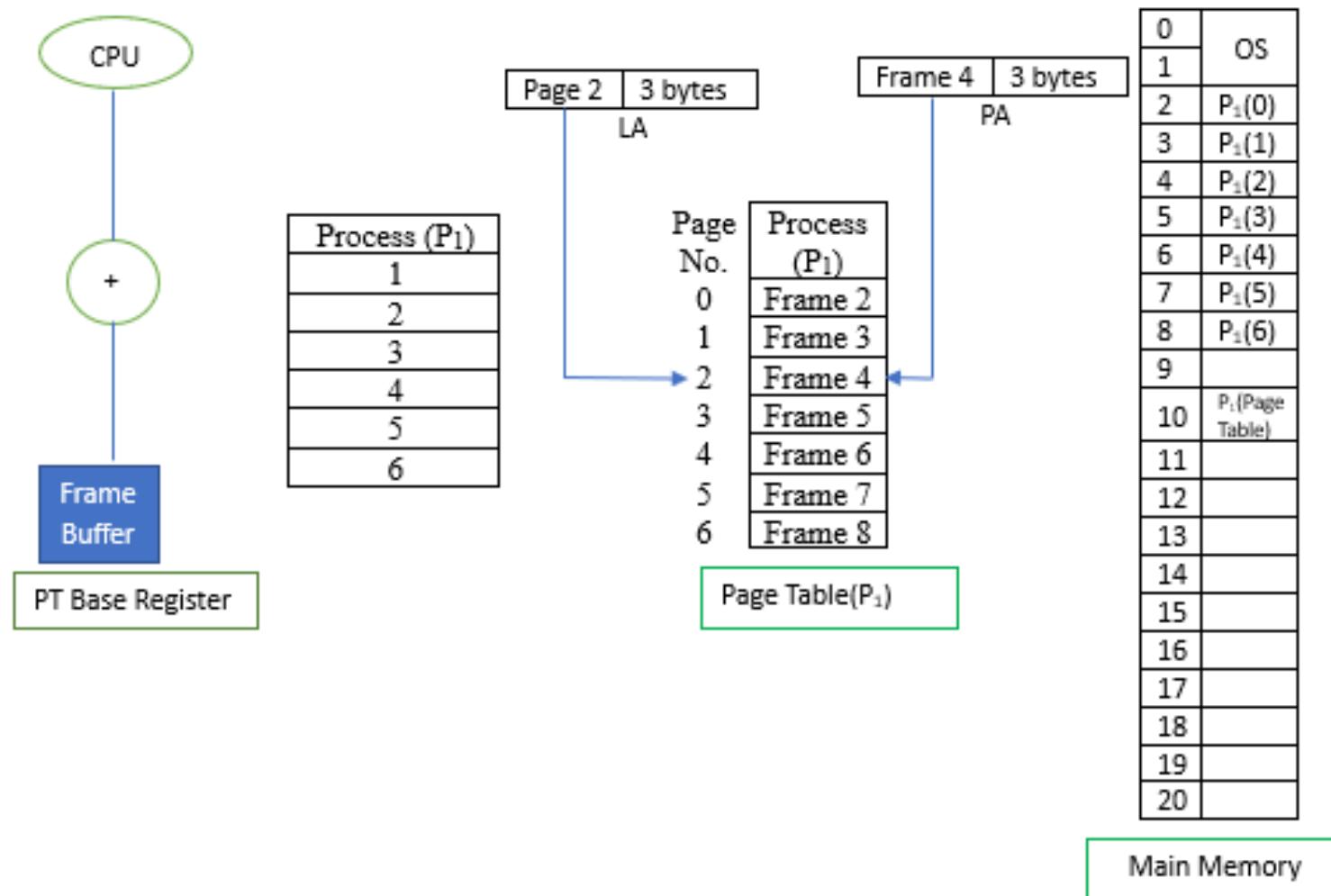
# Paging

- With 64KB of virtual address space and 32KB of physical memory and 4KB page size, we get 16 virtual pages and 8 frames.
- What happen in following instruction?
- MOV REG 0, 2
  - This virtual address, 0, is sent to the MMU. The MMU sees that this virtual address falls in page 0 (0-4095), which is mapping to frame 2 (8192 -12287). Thus the address 0 is transformed to 8192 and output address is 8192.

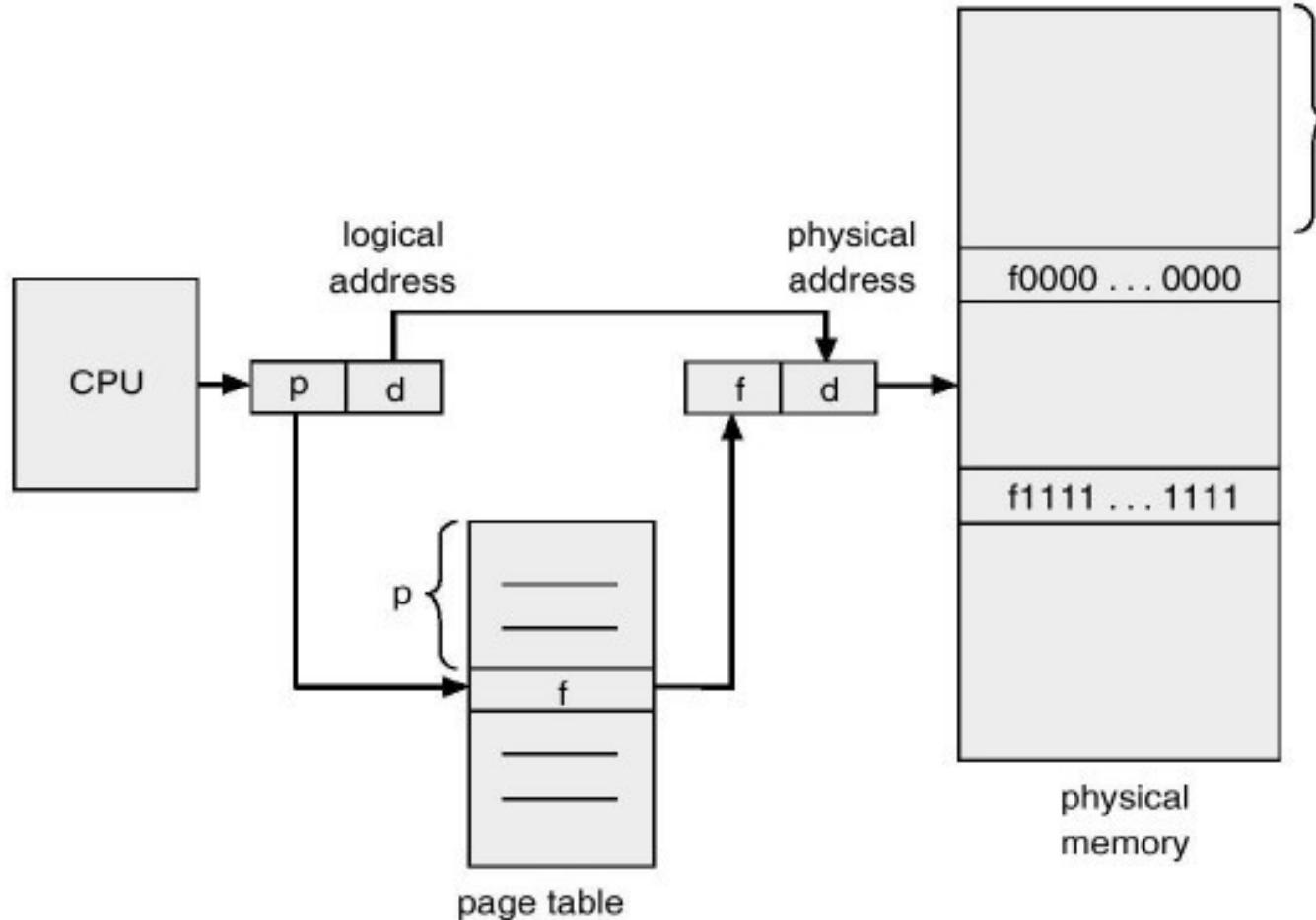
# Paging

- Address generated by CPU is divided into:
  - Page number (p) . used as an index into a page table which contains base address of each page in physical memory.
  - Page offset (d) . combined with base address to define the physical memory address that is sent to the memory unit

# Paging



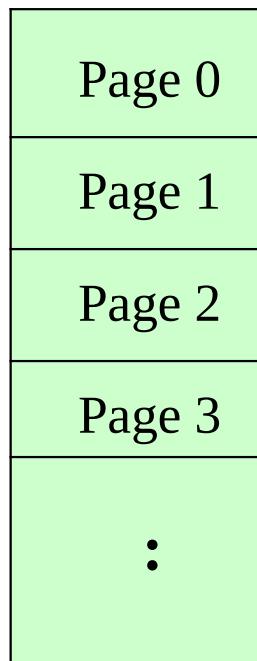
# Paging



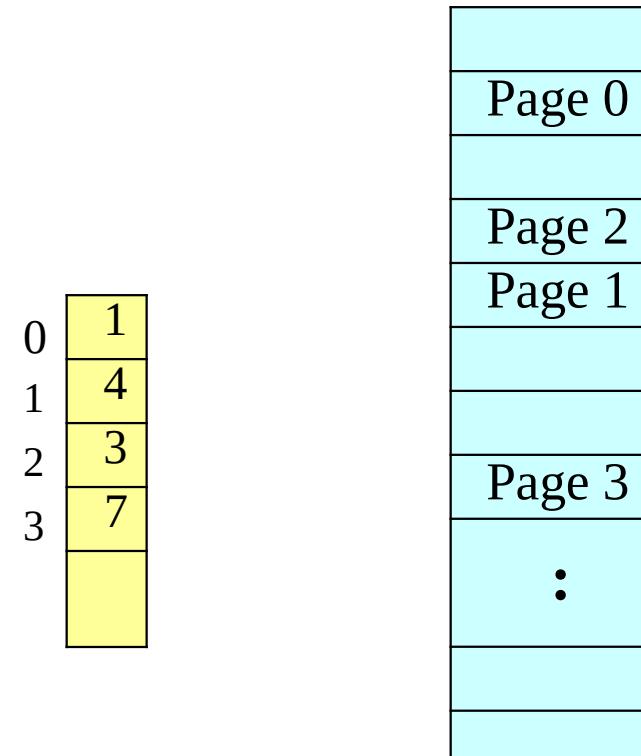
**Figure: Paging Hardware**

# Example of Paging

Logical memory

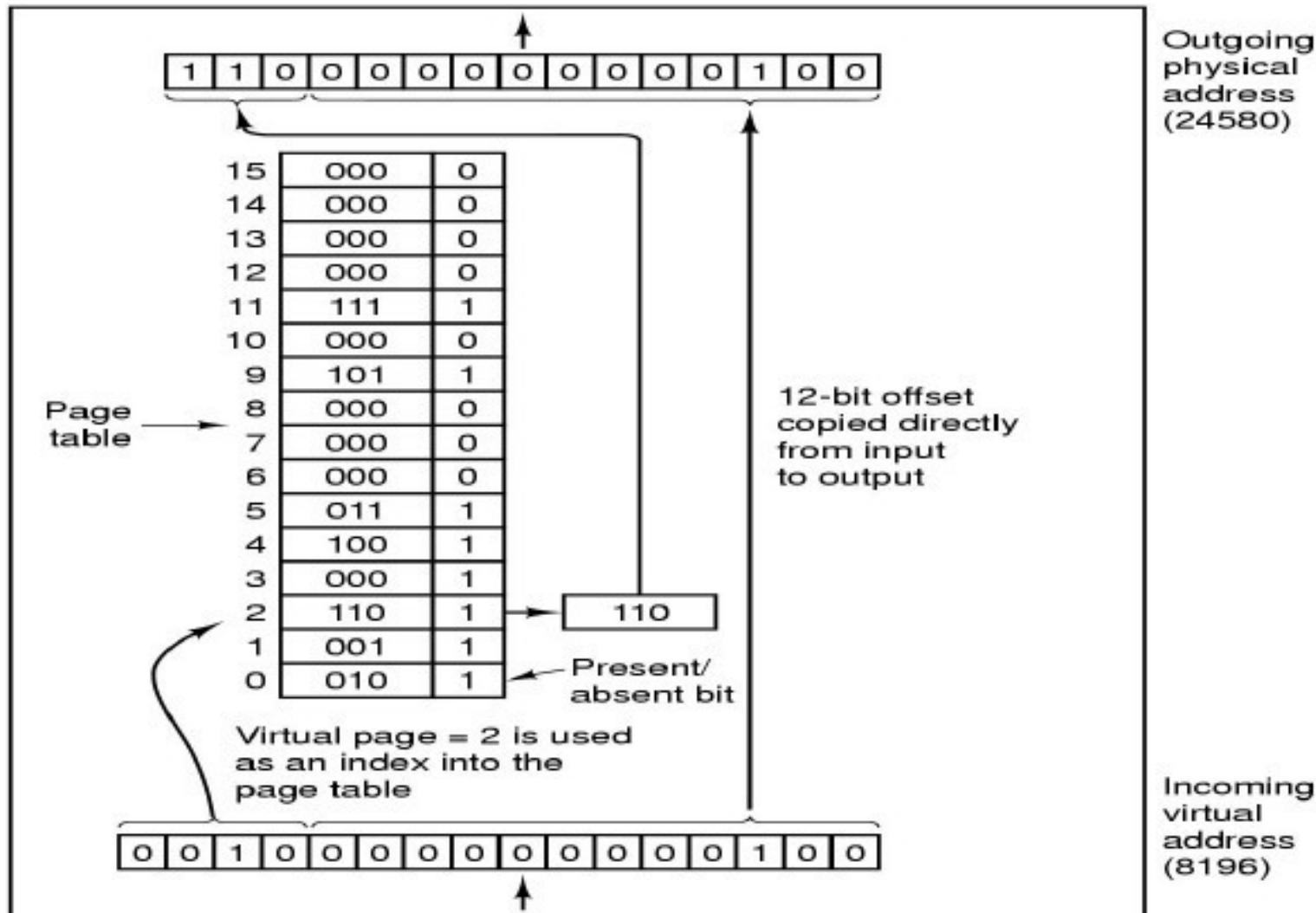


Physical memory



**Figure: Paging Model of Logical and Physical Memory**

# Example



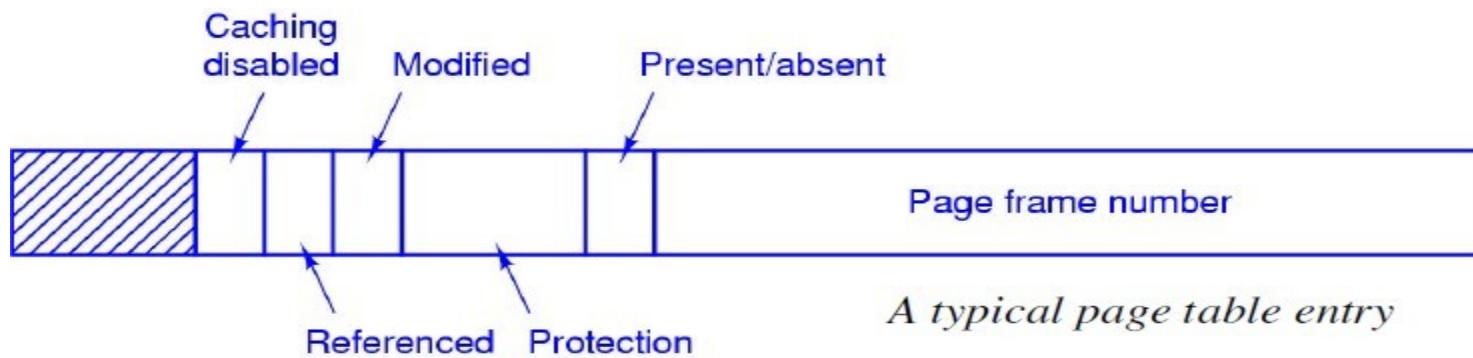
# Page Tables

For each process, page table stores the number of frame, allocated for each page.

- The purpose of the page table is to map virtual pages into pages frames. This function of page table can be represented in mathematical notation as:  
$$\text{page\_frame} = \text{page\_table}(\text{page\_number})$$
- The virtual page number is used as an index into the page table to find the corresponding page frame.

# Page Table Structure

- The exact layout of page table entry is highly machine dependent, but more common structure for 32-bit system is as:



- Frame number: The goal is to locate this value

# Page Table Structure

- **Present/absent bit:** If present/absent bit is present, the virtual addresses is mapped to the corresponding physical address. If present/absent is absent the trap is occur called page fault.
- **Protection bit:** Tells what kinds of access are permitted read, write or read only.
- **Modified bit (dirty bit):** Identifies the changed status of the page since last access; if it is modified then it must be rewritten back to the disk.
- **Referenced bit:** set whenever a page is referenced; used in page replacement.
- **Caching disabled:** used for that system where the mapping into device register rather than memory.

# Page Tables Issues

- Size of page table.
  - Most modern computers support a large virtual-address space ( $2^{32}$  to  $2^{64}$ ). If the page size is 4KB, a 32-bit address space has 1 million pages. With 1 million pages, the page table must have 1 million entries.
  - think about 64-bit address space???
- Efficiency of mapping.
  - If a particular instruction is being mapped, the table lookup time should be very small than its total mapping time to avoid becoming CPU idle.
- What would be the performance, if such a large table have to load at every mapping.

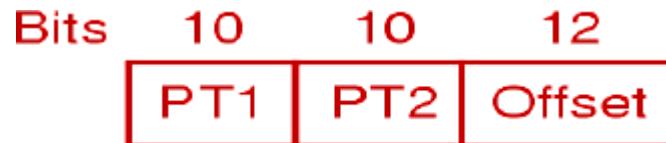
How to handle these issues?

# Multilevel Page Tables

- To get around the problem of having to store huge page tables in memory all the time, many computers use the multilevel page table in which the page table itself is also paged.
- Pentium II -2 level, 32-bit Motorola -4 level, 32-bit SPARC-3 level etc.

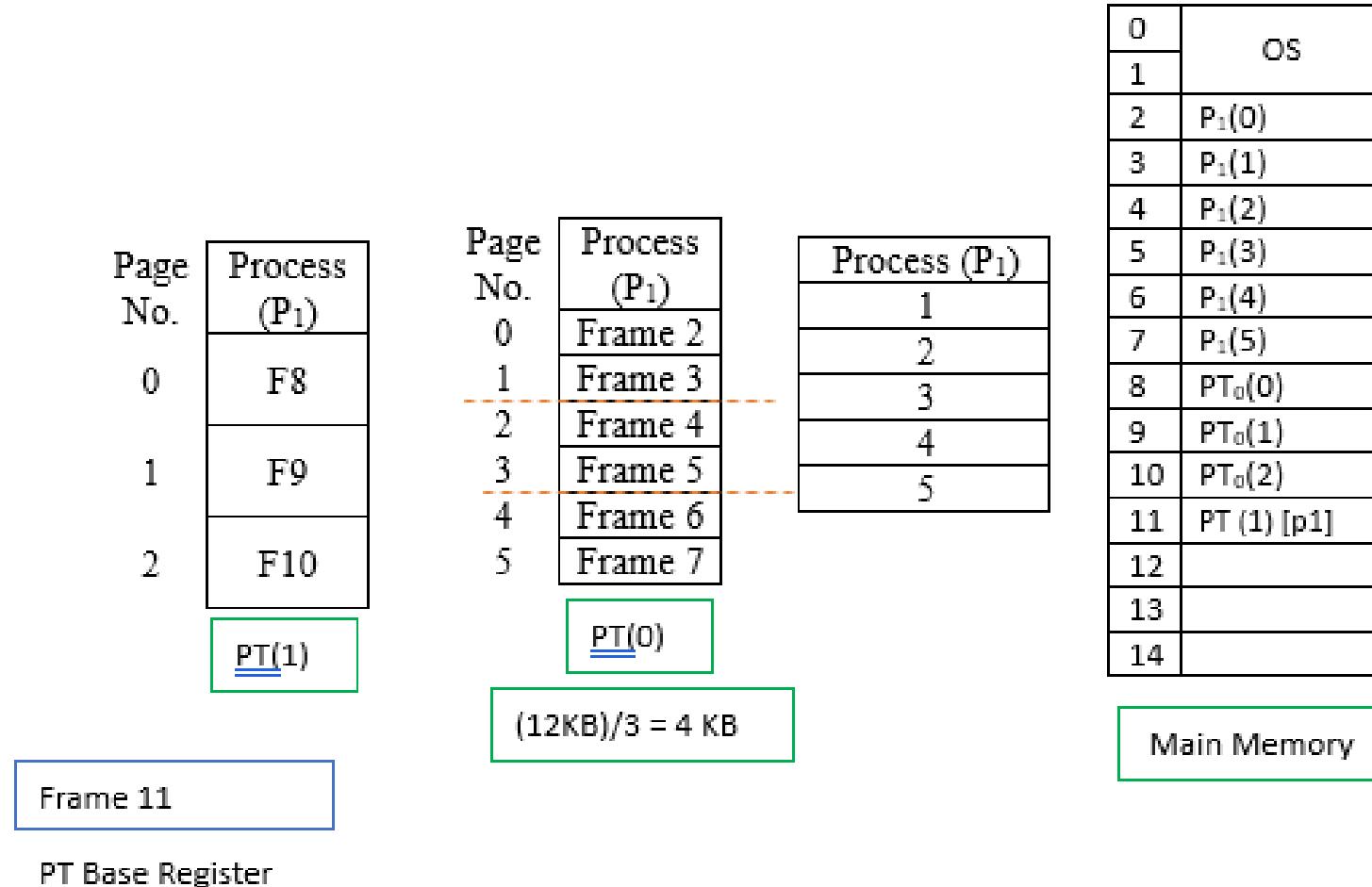
# Multilevel Page Tables

- Example: Two-Level Page Tables
- A 32-bit virtual address space with a page size of 4 KB, the virtual address space is partitioned into a 10-bit PT1 field, a 10-bit PT2 field, and a 12-bit offset field.

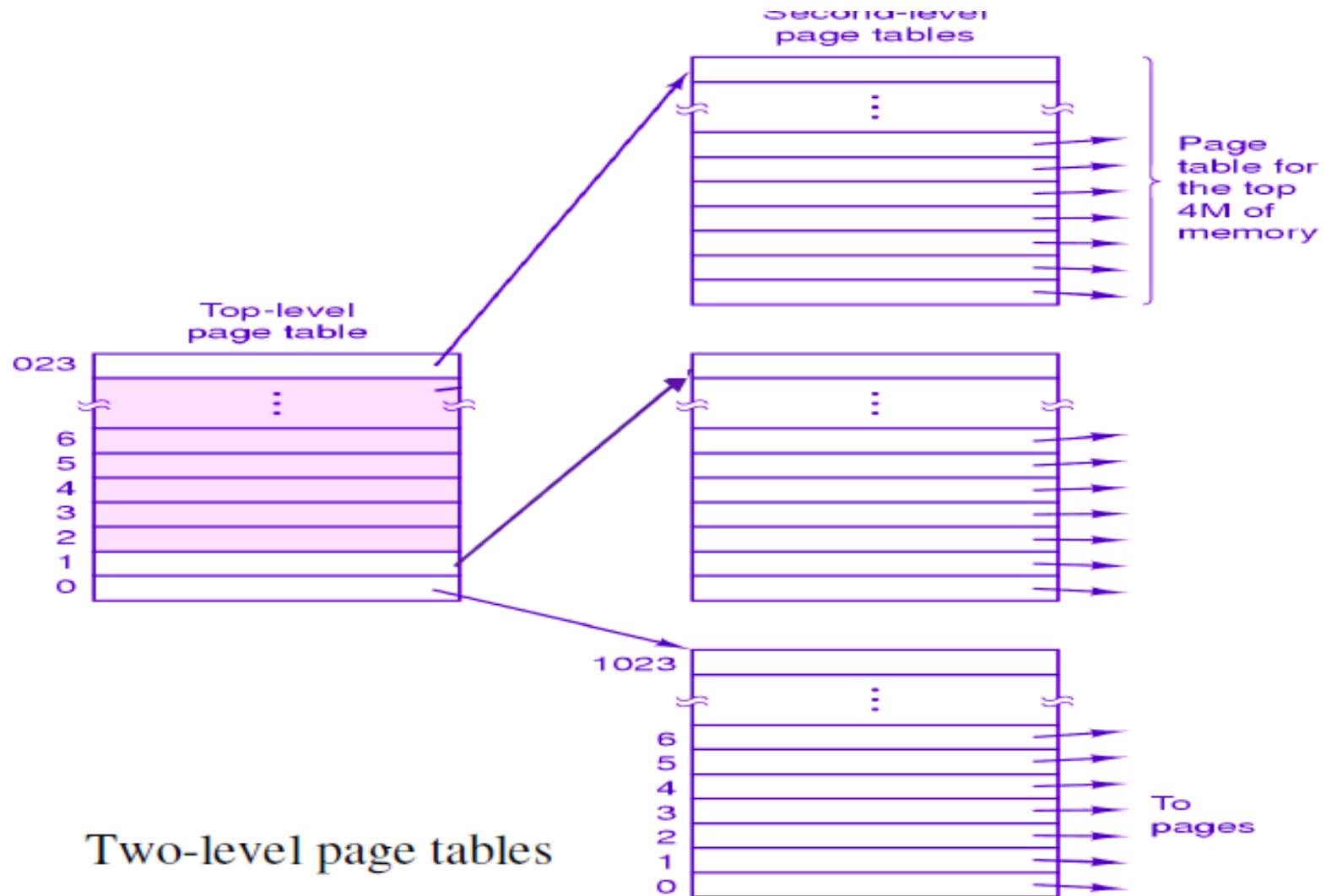


- The top level have 1024 entries, corresponding to PT1. At mapping, it first extracts the PT1 and uses this value as an index into the top level page table. Each of these entries have again 1024 entries, the resulting address of top-level yields the address or page frame number of second-level page table.

# Multilevel Page tables



# Multilevel Page tables



# Translation Look aside Buffers (TLBs)

How to speed up address translation?

Locality: Most processes use large number of reference to small number of pages.

- The small, fast, lookup hardware cache, called TLB is used to overcome this problem, by **mapping logical address to physical address without page tables**.
- The TLB is associative, high-speed, memory; each entry **consists information about virtual page-number and physical page frame number**.
- Typical sizes: only 64 to 1024 entries.
- Key of improvement: Parallel search for all entries.

# TLB

Tag (or Key)

Page Number	Process Number
-------------	----------------

Tag	Frame No.

PT (1)

Page No.	Process (P <sub>1</sub> )
0	Frame 2
1	Frame 3
2	Frame 4
3	Frame 5
4	Frame 6
5	Frame 7

Page Table

1
2
3
4
5

Process (P<sub>1</sub>)

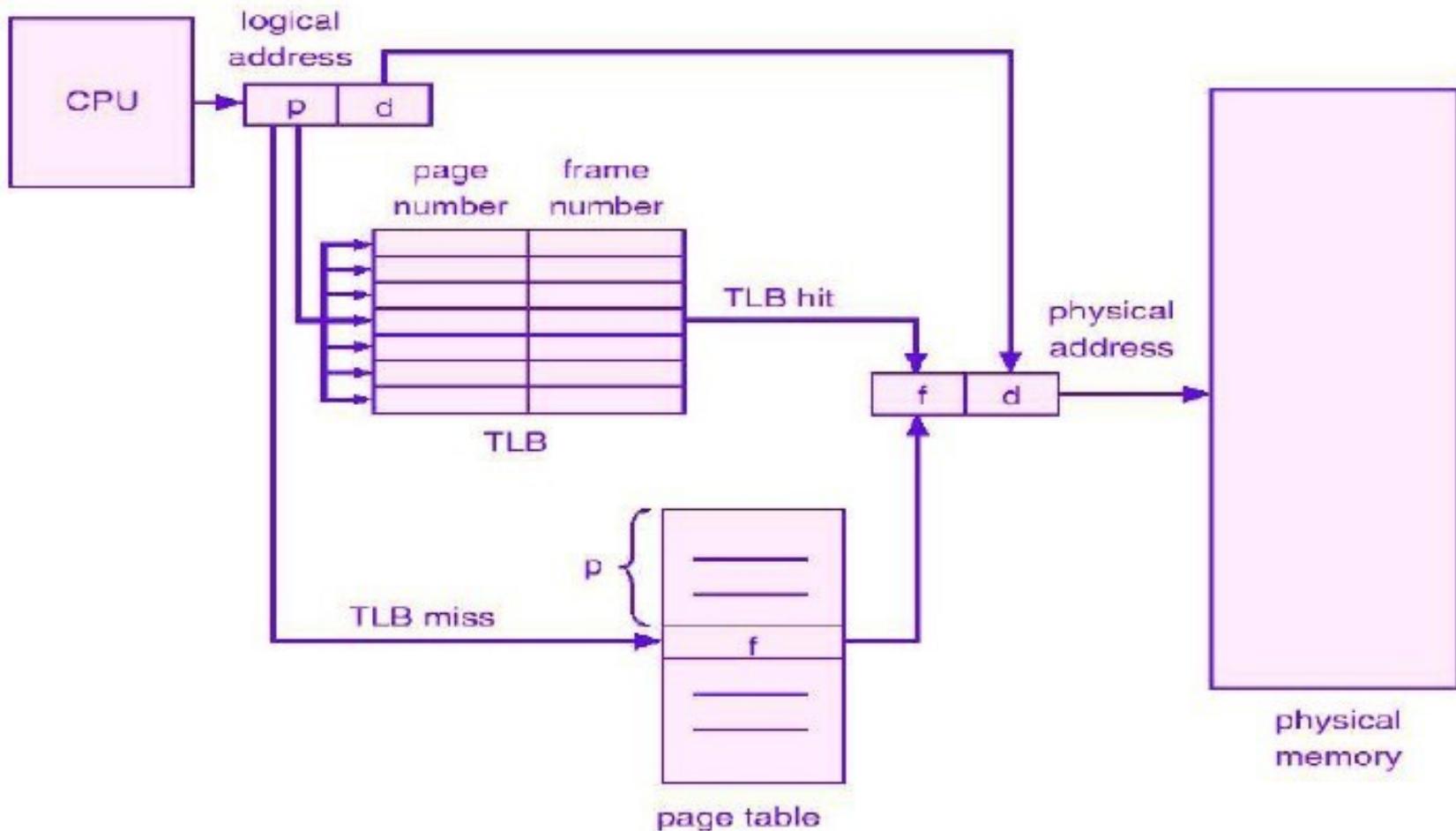
0	OS
1	
2	P <sub>1</sub> (0)
3	P <sub>1</sub> (1)
4	P <sub>1</sub> (2)
5	P <sub>1</sub> (3)
6	P <sub>1</sub> (4)
7	P <sub>1</sub> (5)
8	PT(0)
9	
10	
11	
12	
13	
14	

Main Memory

Frame 8

PT Base Register

# TLB



# TLB

- When logical address is generated by CPU, its page number is presented to the TLB; if page number is found (TLB hit), its frame number is immediately available, the whole task would be very fast because it compare all TLB entries simultaneously.
- If the page number is not in TLB (TLB miss), a memory reference to the page table must be made. It then replace one entry of TLB with the page table entry just referenced. Thus in next time, for that page, TLB hit will found.

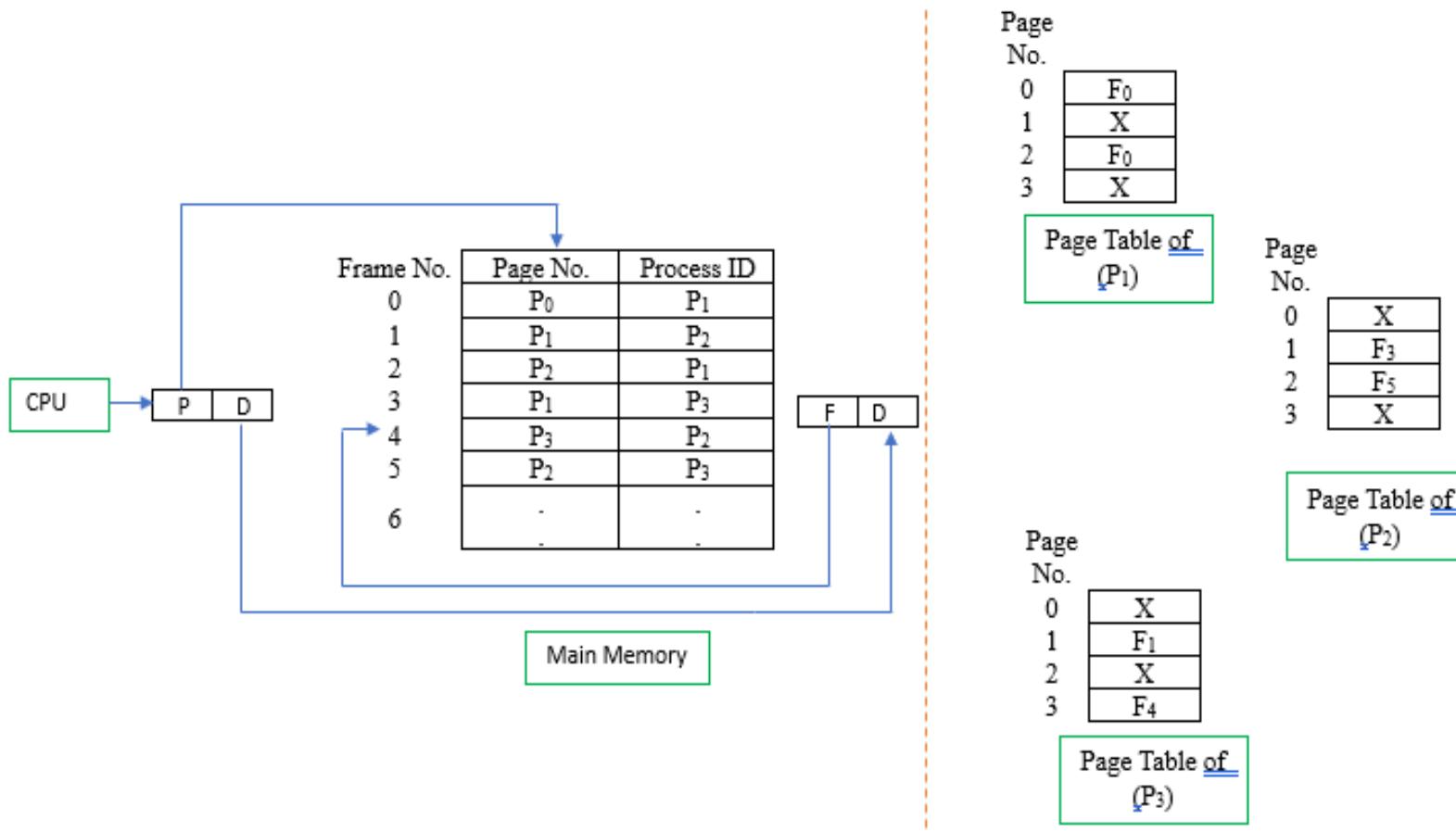
# Advantages/Disadvantages of Paging

- Advantages:
  - Fast to allocate and free:
    - Alloc: keep free list of free pages, grab first page in the list.
    - Free: Add pages to free list.
  - Easy to swap-out memory to disk.
    - Frame size matches disk page size.
    - Swap-out only necessary pages.
    - Easy to swap-in back from disk.
- Disadvantages:
  - Additional memory reference.
  - Page table are kept in memory.
  - Internal fragmentation: process size does not match allocation size.

# Inverted Page Table

- Rather than each process having a page table and keeping track of all possible logical pages, track all physical pages
- One entry for each real page 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
- 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
  - TLB can accelerate access
- But how to implement shared memory?
  - One mapping of a virtual address to the shared physical address

# Inverted Paging



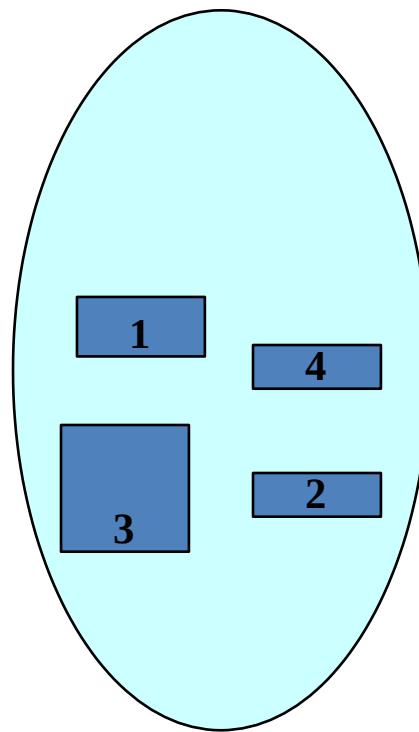
# Exercises

- Why are page sizes always a power of 2?
- On a simple paging system with  $2^{24}$  bytes of physical memory, 256 pages of logical address space, and a page size of  $2^{10}$  bytes, how many bits are in a logical address?
- Describe, how TLB increase performance in paging.

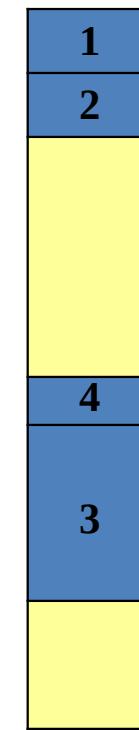
# Segmentation

- 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
  - local variables, global variables, common block
  - stack, symbol table, arrays
- Protect each entity independently
- **Allow each segment to grow independently**
- Share each segment independently

# Logical view of segmentation



*User Space*

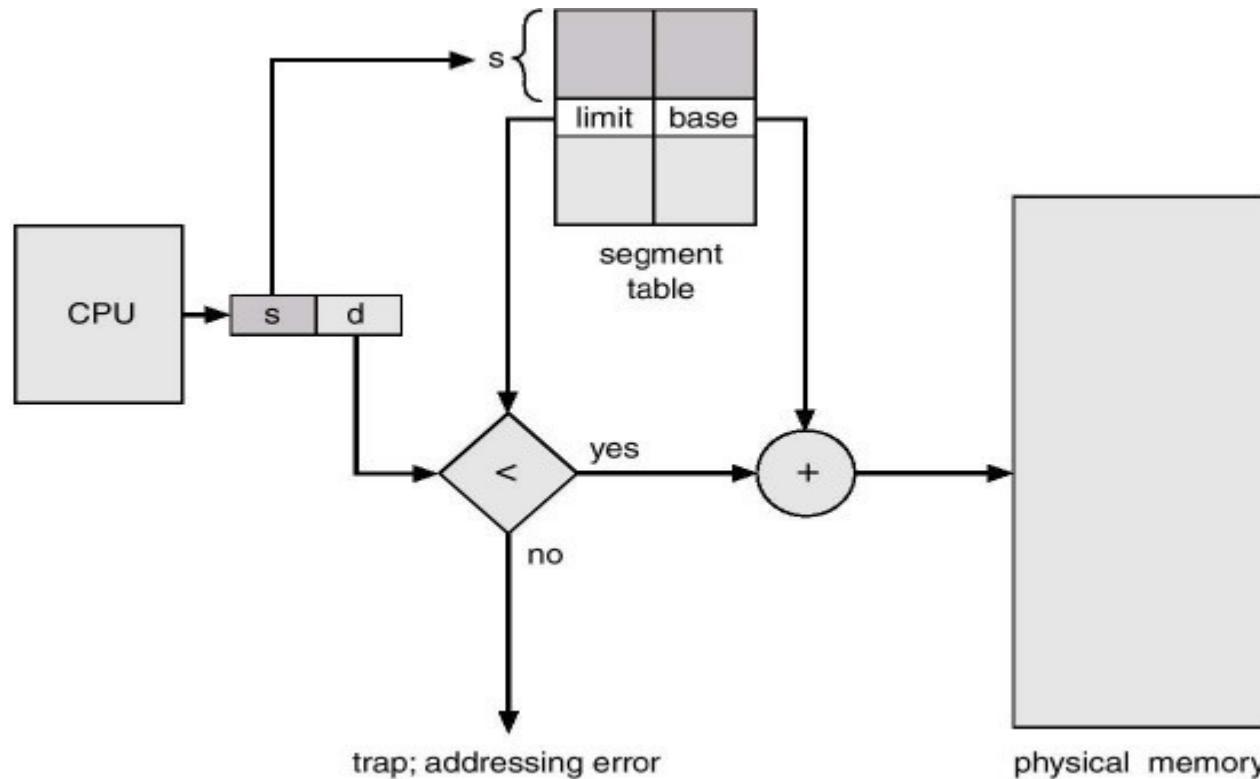


*Physical Memory*

# Segmentation Architecture

- Logical address consists of a two tuple  
    <segment-number, offset>
- Segment Table
  - Maps two-dimensional user-defined addresses into one-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. We call it **Base register**
  - *Segment-table length register* (STLR) indicates the number of segments used by a program; segment number is legal if  $s < \text{STLR}$ .
  - **Limit register**

# Address Translation in segmentation

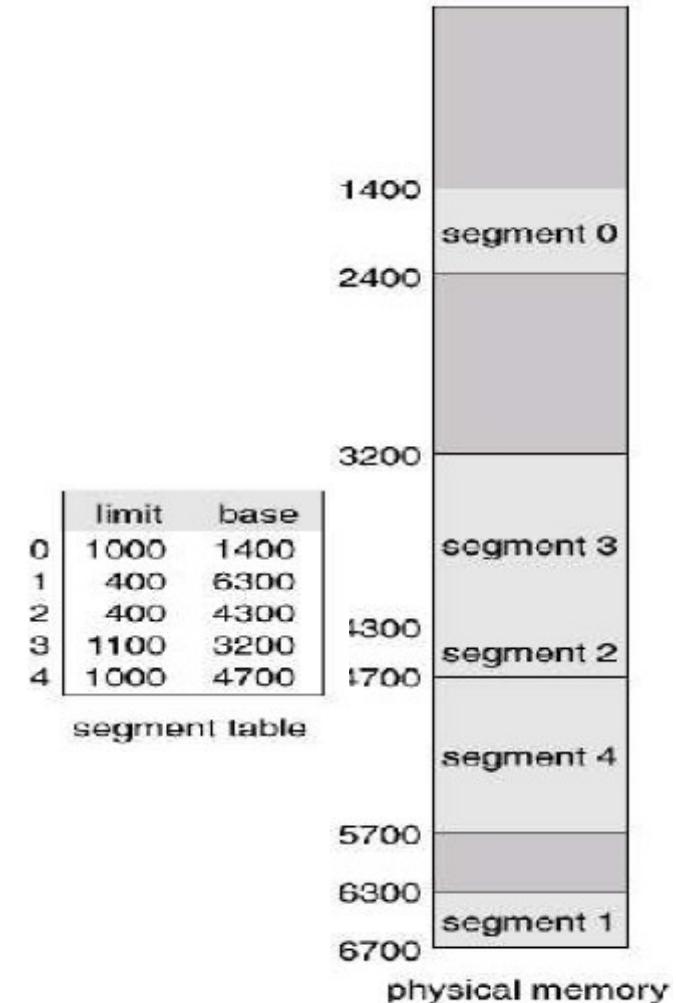


# Segmentation Architecture (cont.)

- Relocation is dynamic - by segment table
- Sharing
  - Code sharing occurs at the segment level.
  - Shared segments must have same segment number.
- Allocation - dynamic storage allocation problem
  - use best fit/first fit, may cause external fragmentation.
- Protection
  - protection bits associated with segments
    - read/write/execute privileges
    - array in a separate segment - hardware can check for illegal array indexes.

# Segmentation Architecture (cont.)

- The segment number used as index into the segment table. The offset d of the logical address must be between 0 and the segment limit. If not , trap occur, if it is legal it is added to the segment base to produce the address in the physical memory.
- The segmentation scheme causes External fragmentation, this can be handle by **memory compaction technique**.



# Example - Address translation in segmentation

Compute the Physical address for

- a) 099
- b) 278
- c) 1265

Ans:

a) 099, here segment no is 0 and offset is 99.  
since  $99 < 124$  and segment 0 begins at  
location 330, the physical address is  
 $330 + 99 = 429$

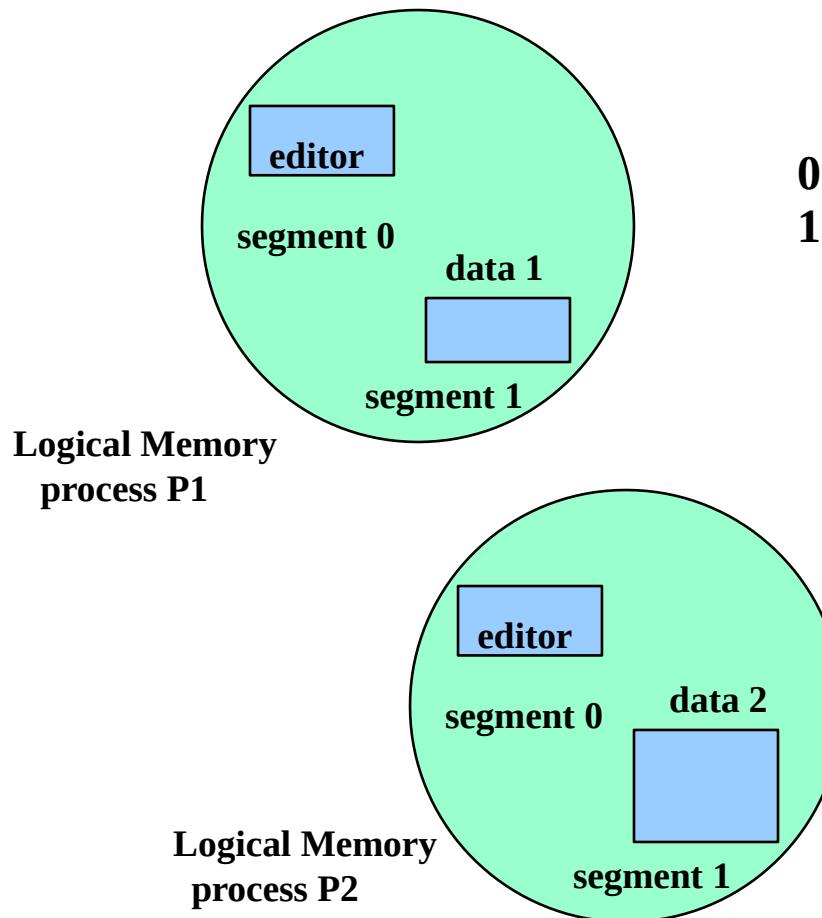
b) 278, here segment no is 2 and offset is  $78 < 99$ .  
Segment 2 begins at 111, the physical address  
is  $111 + 78 = 189$

c) 1265, here segment no is 1 and offset is  
 $265 > 211$ . this address result in segment fault.

Consider the segment table:

Segment	Base	Size	
0			124
1	330		
2	111		99
3	498		302

# Shared segments

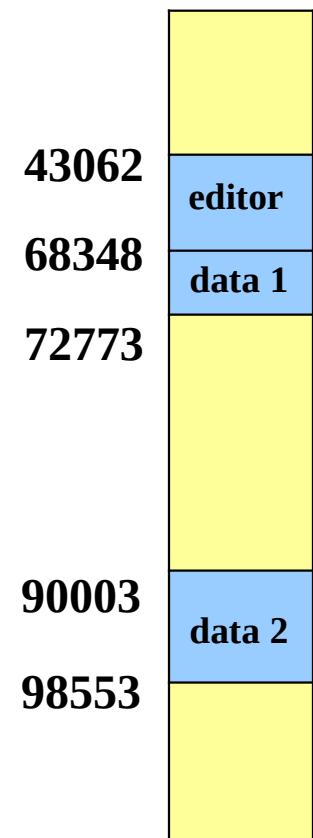


	Limit	Base
0	25286	43602
1	4425	68348

Segment  
Table  
process P1

	Limit	Base
0	25286	43602
1	8850	90003

Segment  
Table  
process P2



# Paging Vs Segmentation

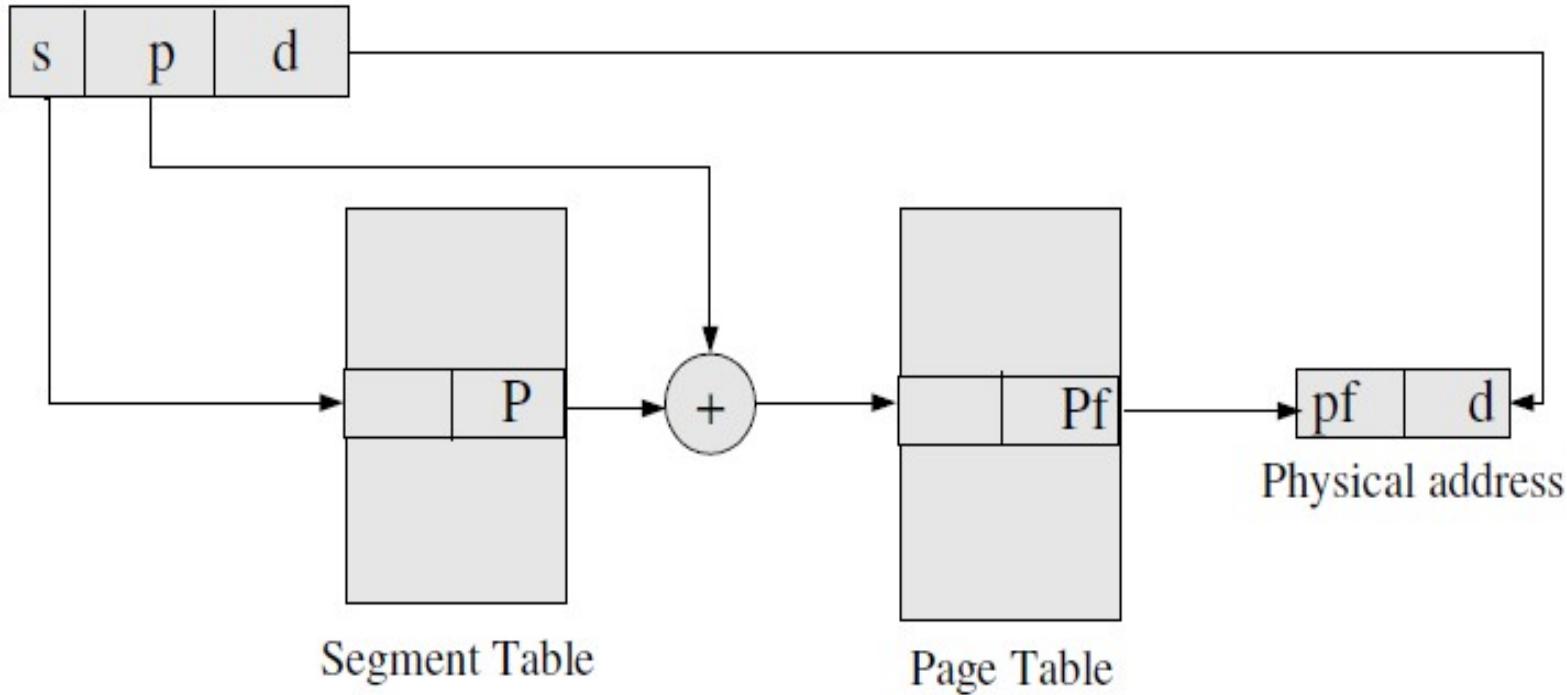
Consideration	Paging	Segmentation
Need the programmer be aware that this technique is being used?	No	Yes
How many linear address spaces are there?	1	Many
Can the total address space exceed the size of physical memory?	Yes	Yes
Can procedures and data be distinguished and separately protected?	No	Yes
Can tables whose size fluctuates be accommodated easily?	No	Yes
Is sharing of procedures between users facilitated?	No	Yes
Why was this technique invented?	To get a large linear address space without having to buy more physical memory	To allow programs and data to be broken up into logically independent address spaces and to aid sharing and protection

# Segmentation with Paging

- What happen when segment are larger than main memory?
- Segmentation can be combined with paging to provide the efficiency of paging with the protection and sharing capabilities of segmentation.
- As with simple segmentation, the logical address specifies the segment number and the offset within the segment.
- When paging is added, the segment offset is further divided into a page number and page offset.
- The segment table entry contains the address of the segment's page table.

# Segmentation with Paging

Logical address



# Questions?

- A smaller page size leads to smaller page tables –
  - » False -> need more entries because we have more pages
- A smaller page size leads to more TLB misses
  - True -> less likely that page will encompass address we are after.
- A smaller page size leads to fewer page faults
  - True

# Continue?

- **What is swapping? Can swapping permit an application requiring 16M memory to run on a machine with 8M of RAM?**
- Swapping is the act of running each whole process in main memory for a time then placing back onto disk and vice versa. It is used when the system does not have enough main memory to hold all the currently active processes.
- Assuming that an application is one program and hence a single process, swapping will not permit an application(process) requiring 16M of memory to run on a machine with 8M RAM (main memory) as the whole process is too large to fit into main memory.

# Continue?

- **Enumerate some pros and cons for increasing the page size.**
- pros:
  - reduces page table size
  - increases TLB coverage
  - increases swapping I/O throughput, as small disk
- cons:
  - increases page fault latency, as swap response time is slower due to more data.
  - increases internal fragmentation of pages, as there is more 'wasted page' in the working set

# Exercise

- Why are segmentation and paging sometimes combined into one scheme?
- Consider the following segment table:

Segment	base	size
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

- What are the physical address for the following logical address?
  - a) 0430 b) 110 c) 2500 d) 3400 e) 4112
- Distinguish the paging and segmentation.

# Paging/Segmentation Policies

- Fetch Strategies
  - When should a page or segment be brought into primary memory from secondary (disk) storage?
    - Demand Fetch
    - Anticipatory Fetch
- Placement Strategies
  - When a page or segment is brought into memory, where is it to be put?
    - Paging - trivial
    - Segmentation - significant problem
- Replacement Strategies
  - Which page/segment should be replaced if there is not enough room for a required page/segment?

# Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less Memory needed
  - Faster response
  - More users
- The first reference to a page will trap to OS with a page fault.
- OS looks at another table to decide
  - Invalid reference - abort
  - Just not in memory.

# Valid-Invalid Bit

- With each page table entry a valid-invalid bit is associated (1 in-memory, 0 not in memory).
- Initially, valid-invalid bit is set to 0 on all entries.
  - During address translation, if valid-invalid bit in page table entry is 0 --- ***page fault*** occurs.
  - Example of a page-table snapshot

Frame # Valid-invalid bit

Frame #	Valid-invalid bit
	1
	1
	1
	1
	0
:	
	0
	0
	0

**Page  
Table**

# Handling a Page Fault: Process

- Page is needed - reference to page
  - Step 1: Page fault occurs - trap to OS (process suspends).
  - Step 2: Check if the virtual memory address is valid. Kill job if invalid reference. If valid reference, and page not in memory, continue.
  - Step 3: Bring into memory - Find a free page frame, map address to disk block and fetch disk block into page frame. When disk read has completed, add virtual memory mapping to indicate that page is in memory.
  - Step 4: Restart instruction interrupted by illegal address trap. The process will continue as if page had always been in memory.

# What happens if there is no free frame?

- Page replacement - find some page in memory that is not really in use and swap it.
  - Need page replacement algorithm
  - Performance Issue - need an algorithm which will result in minimum number of page faults.
- Same page may be brought into memory many times?????.

# Page Replacement

- Prevent over-allocation of memory by modifying page fault service routine to include page replacement.
- Use modify(dirty) bit to reduce overhead of page transfers - only modified pages are written to disk.
- Page replacement
  - large virtual memory can be provided on a smaller physical memory.

# Page Replacement Strategies (algorithms)

- The Principle of Optimality
  - Replace the page that will not be used again the farthest time into the future.
- Random Page Replacement
  - Choose a page randomly
- FIFO - First in First Out
  - Replace the page that has been in memory the longest.
- LRU - Least Recently Used
  - Replace the page that has not been used for the longest time.
- LFU - Least Frequently Used
  - Replace the page that is used least often.
- NUR - Not Used Recently
  - An approximation to LRU
- Working Set
  - Keep in memory those pages that the process is actively using

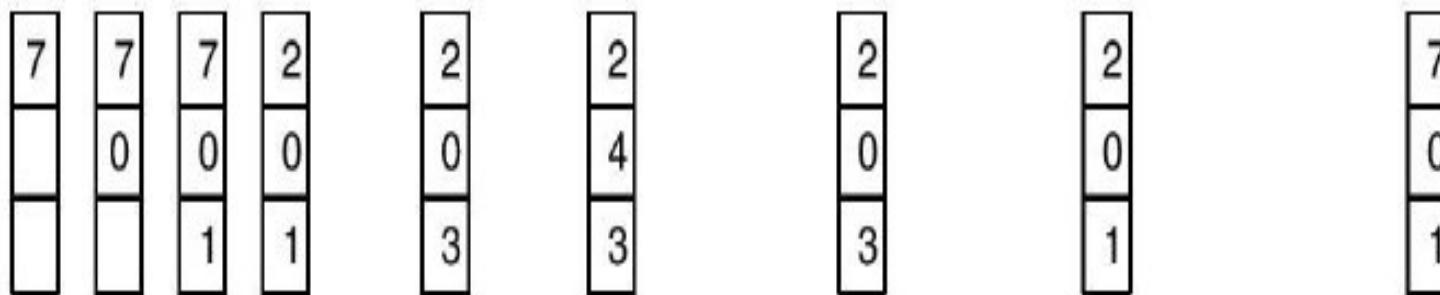
# Optimal Page Replacement (OPR)

Replace the page that will not be used for the longest period of time.

- Each page can be labeled with number of instructions that will be executed before that page is first referenced.
- Ex: For three page frames and 8 pages system the optimal page replacement is as:

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



# No of page fault?

**For three page frames and page references in the order**

- 0      1    2    3    2    1    0    3    2    3

- Note: Not Implementable “Future reference can not be determined exactly”. It depend upon the process behavior.
- Used to compare the efficiency of other page replacement algorithms

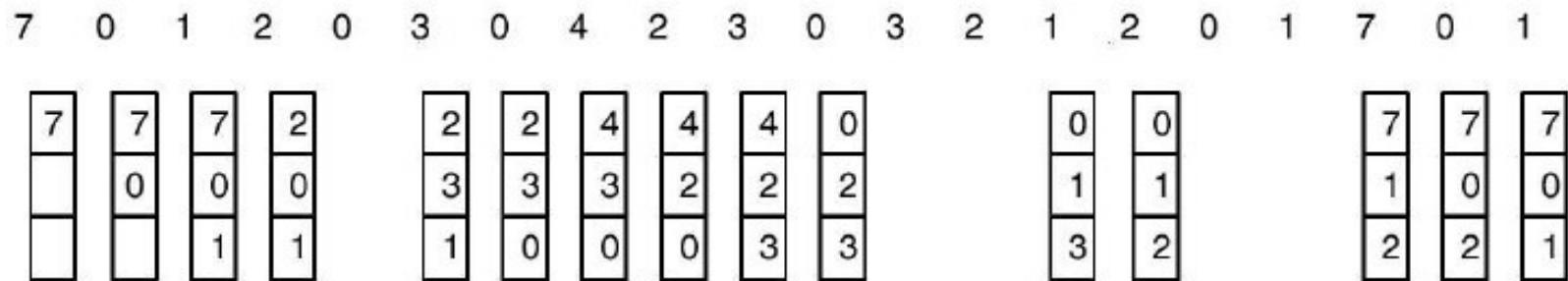
# OPR: Discussion

- Advantages:
  - An optimal page-replacement algorithm; it guarantees the lowest possible page fault rate.
- Problems:
  - Unrealizable, at the time of the page fault, the OS has no way of knowing when each of the pages will be referenced next.
- This is not used in practical system.

# First-In-First-Out (FIFO)

- This associates with each page the time when that page was brought into the memory. The page with highest time is chosen to replace.
- This can also be implemented by using queue of all pages in memory.

reference string



# How many Page Fault?

**for four page frames and page references in the order**

- 0      1    2    3    2    1    0    3    2    3

# FIFO

- Advantages:
  - Easy to understand and program.
  - Distributes fair chance to all.
- Problems:
  - FIFO is likely to replace heavily (or constantly) used
  - pages and they are still needed for further processing.

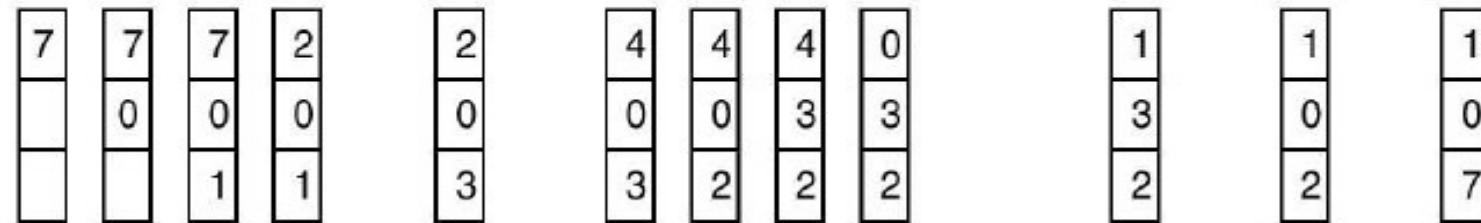
# Least Recently Used (LRU)

Recent past is a good indicator of the near future.

- When a page fault occurs, throw out the page that has been unused for longest time.
- It maintains a linked list of all pages in memory with the most recently used page at the front and least recently used page at the rear. The list must be updated on every memory reference.

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1



page frames

# Calculate no of Page fault

- For 3 frame and the page reference in order of

- 0      1    2    3    2    1    0    3    2    3

# Implementation of LRU algorithm

- Counter Implementation
  - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter.
  - When a page needs to be changes, look at the counters to determine which page to change (page with smallest time value).
- Stack Implementation
  - Keeps a stack of page numbers in a doubly linked form
  - Page referenced
    - move it to the top
    - requires 6 pointers to be changed
  - No search required for replacement

# Least Frequently Used (LFU)

**One approximation to LRU, software implementation.**

- LFU requires that the page with the smallest count be replaced.
- The reason for this selection is that an actively used page should have a large reference count.
- It requires a software counter associated with each page. When page fault occur the page with lowest counter is chosen for replacement.
- **Problem:** likely to replace highly active pages. This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.

# Not Recently Used (NRU)

- Pages not recently used are not likely to be used in near future and they must be replaced with incoming pages.
- To keep useful statistics about which pages are being used and which pages are not, most computers have two status bits associated with each page . referenced and modified.
- These bits must be updated on every memory reference.
- When a page fault occurs, the OS inspects all the pages and divides them into four categories based on the current value of their referenced and modified bits.

# NRU(Cont)

- Class 0: not referenced, not modified(00). Class 1: not referenced, modified(01).
- Class 2: referenced, not modified(10). Class 3: referenced, modified(11).
- Pages in the lowest numbered class should be replaced first, and those in the highest numbered groups should be replaced last.
- Pages within the same class are randomly selected.

# Working Set (WS) Page Replacement

What to do when a process just swapped out and another process has to load?

**The set of pages that a process is currently using is called its working set.**

If the entire working set is in memory, the process will run without causing many faults until it moves into another execution.

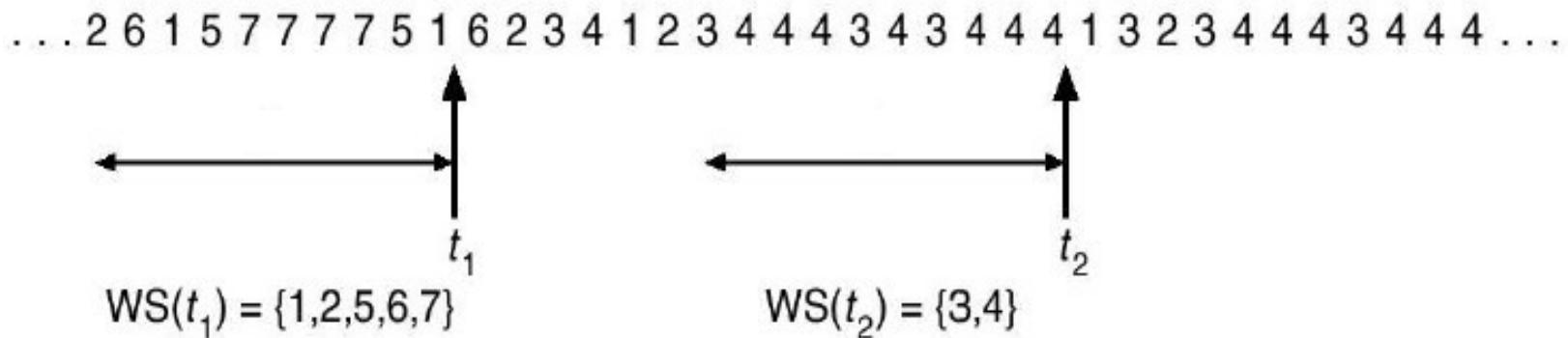
Otherwise, excessive page fault might occur called thrashing.

Many paging systems try to keep track of each process' working set and make sure that it is in memory before the process runs--working set model or pre-paging.

# Working Set (WS) Page Replacement

- The working set of pages of process,  $ws(t, k)$  at time  $t$ , is the set of pages referenced by the processes in time interval  $t-k$  to  $t$ .
- The variable  $k$  is called working-set-window, the size of  $k$  is central issue in this model.
- For example: working set with  $k=10$

page reference table

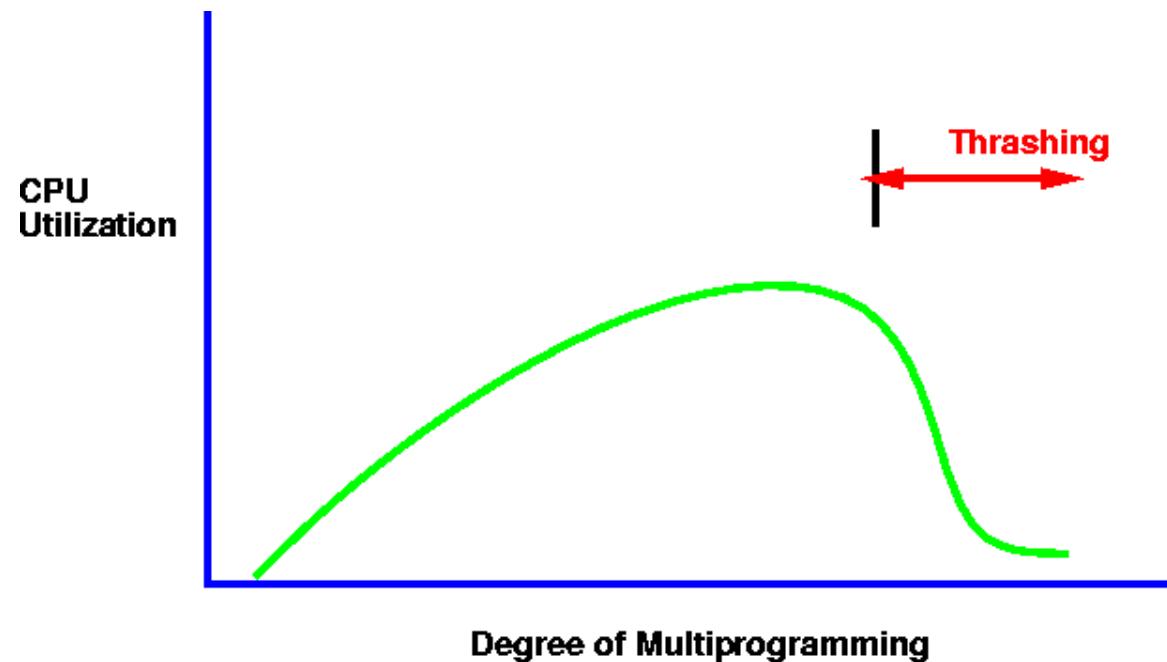


# Thrashing

- If a process does not have enough pages, the page-fault rate is very high. This leads to:
  - low CPU utilization.
  - OS thinks that it needs to increase the degree of multiprogramming
  - Another process is added to the system.
  - System throughput plunges or falls down...
- *Thrashing*
  - A process is busy swapping pages in and out.
  - In other words, **a process is spending more time in paging than in executing.**

# Thrashing

- Why does thrashing occur?



# Belady's Anomaly

- Intuitively, it might seem that the more page frames the memory has, the fewer page faults a program will get.
- Surprisingly enough, this is not always the case. Belady et al. (1969) discovered a counterexample, in which FIFO caused more page faults with four page frames than with three.
- This strange situation has become known as **Belady's anomaly**.

# Example from Belady's paper

- The pages are referenced in the order
- 0 1 2 3 0 1 4 0 1 2 3 4
- Now Calculate the Page faults
  - Case with 3 frame
  - Case with 4 frame
- Now you will find the more page fault in case of 4 frame: Belady's Anomaly

Thank  
You

# Disk Management

**Disk is an I/O devices that is common to every computer.**

Disk Structure

Disk Scheduling RAID

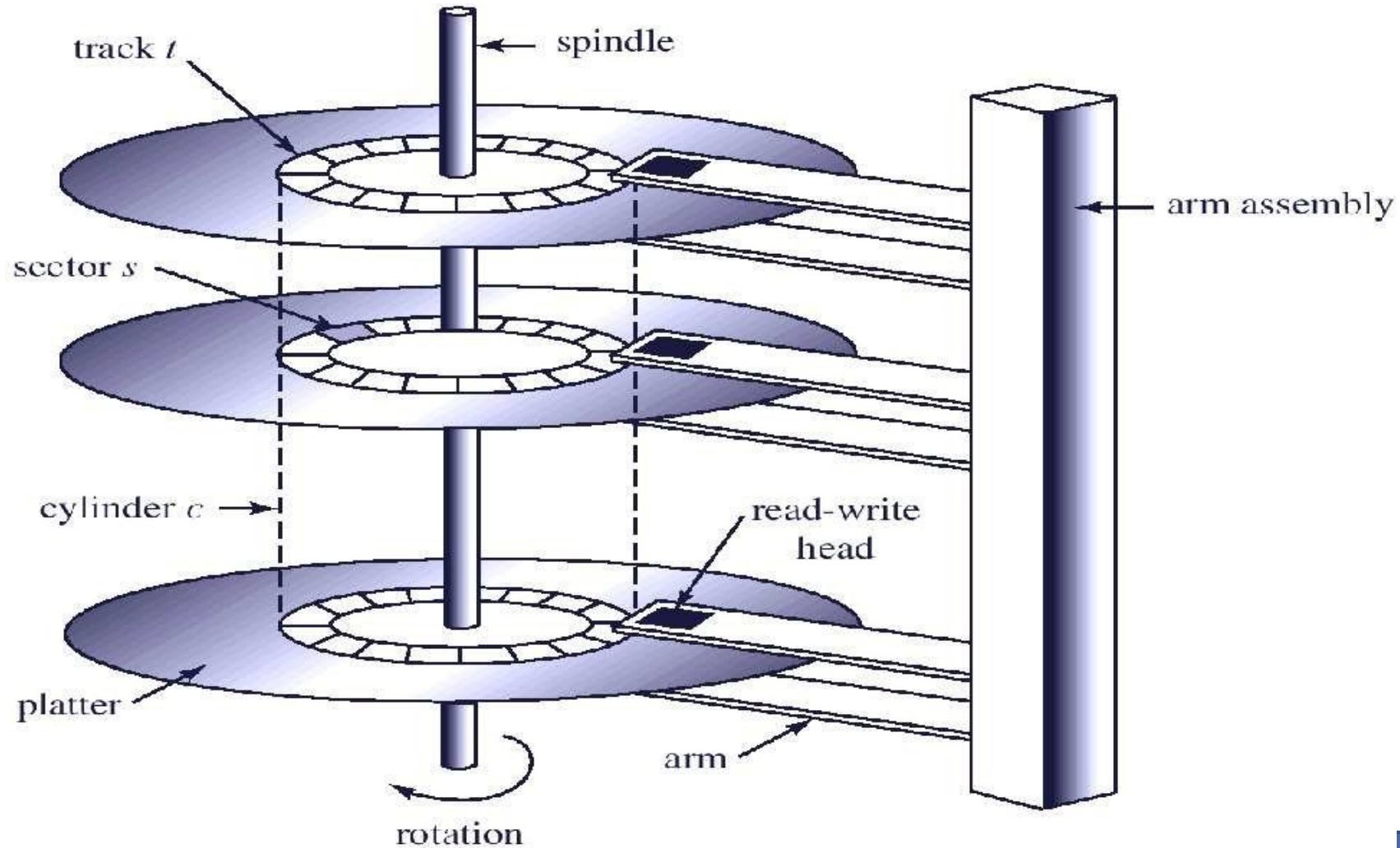
Disk Formatting & Error Handling

RAM Disks

# Disk Structure

- Disks comes in many sizes and information may be stored optically or magnetically; however, all disks share a number of important features.
- For Example: floppy disks, hard disks, CD-ROMs and DVDs.
- Disk surface is divided into number of logical block called sectors and tracks.
- The term cylinder refers to all the tracks at particular head position in hard disk.

# Hard Disk Structure



# Disk Operations

- **Latency Time:** The time taken to rotate from its current position to a position adjacent to the read write head.
- **Seek:** The processes of moving the arm assembly to new cylinder.
- *To access a particular record, first the arm assembly must be moved to the appropriate cylinder, and then rotate the disk until it is immediately under the read-write head.*
- The time taken to access the whole record is called **transmission time**.

# Disk Scheduling

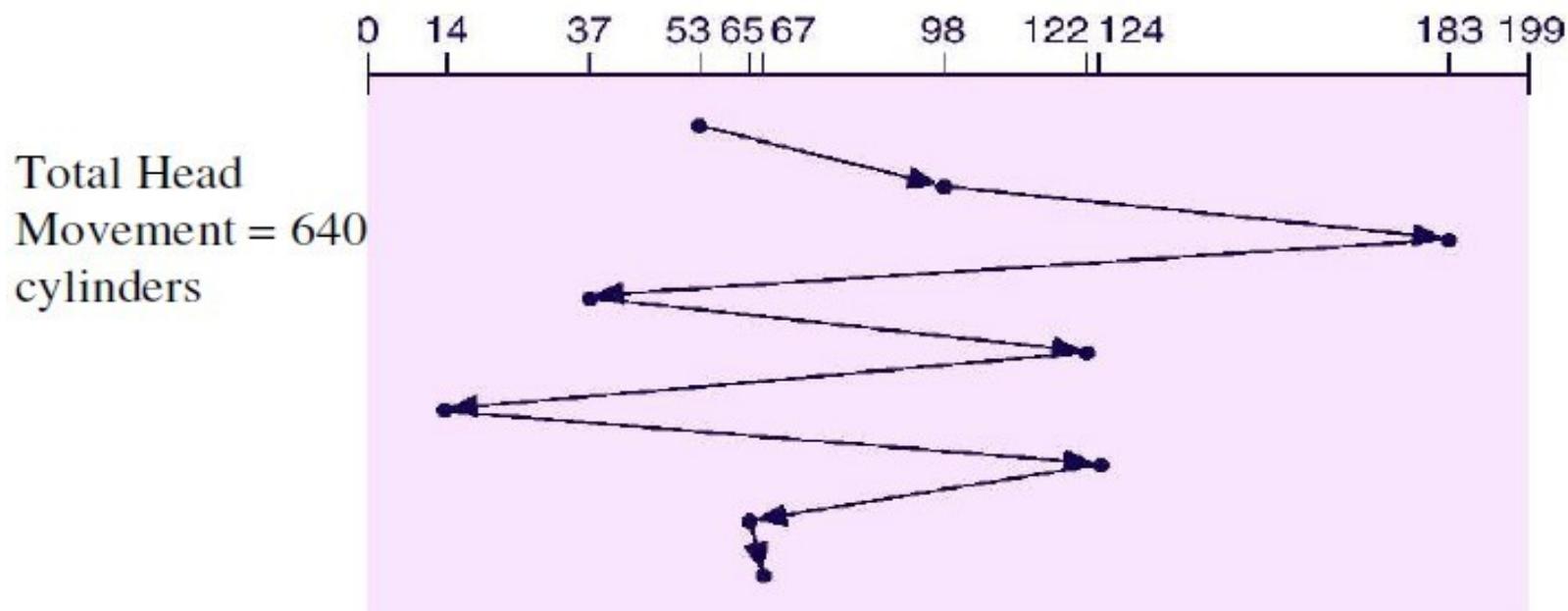
- OS is responsible to use the hardware efficiently for the disk drive this means **fast seek, latency and transmission time.**
- For most disks, the seek time dominates the other two times, so reducing the mean seek time can improve system performance substantially.

# First-Come First-Served (FCFS)

*The first request to arrive is the first one serviced.*

Example:

queue = 98, 183, 37, 122, 14, 124, 65, 67  
head starts at 53



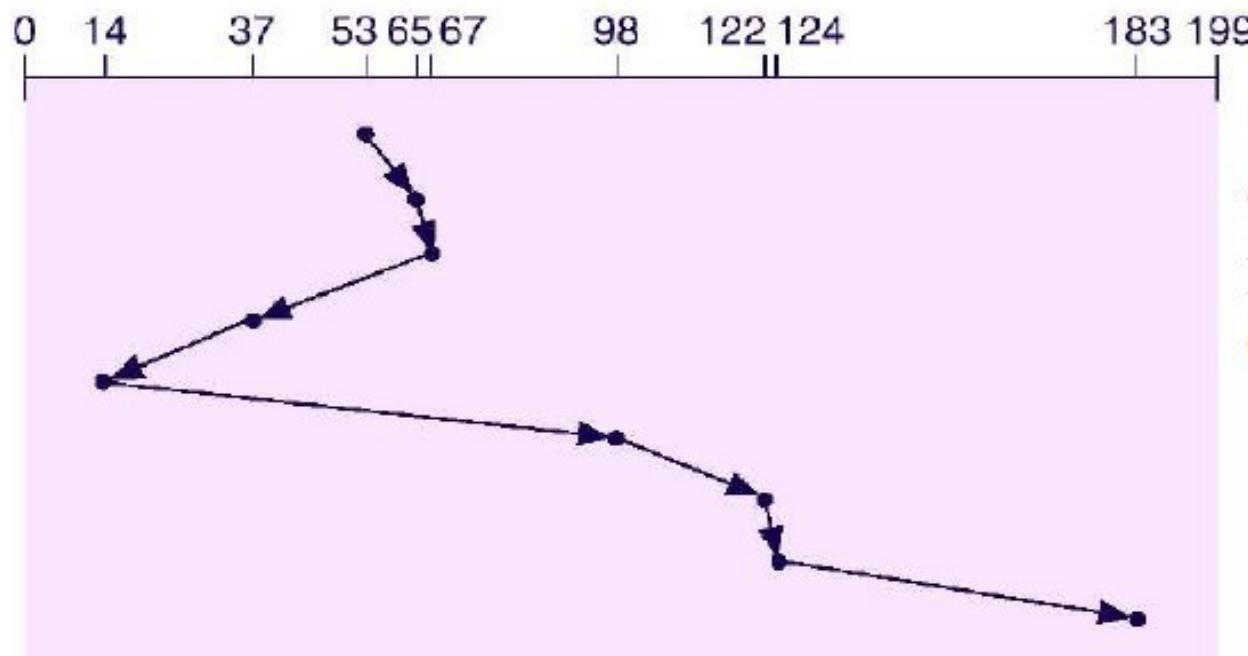
**Advantages:** Simple and Fair.

**Problems:** Does not provide fastest service.

# Shortest-Seek-Time-First (SSTF)

*Selects the request with the minimum seek time from the current head position.*

queue = 98, 183, 37, 122, 14, 124, 65, 67  
head starts at 53



# Shortest-Seek-Time-First (SSTF)

- Advantages:
  - Gives a substantial improvement in performance.
- Problems:
  - SSTF scheduling is a form of SJF scheduling;
  - may cause starvation of some requests.
  - Not optimal.
- Used in batch system where throughput is the major consideration but unacceptable in interactive system.

# SCAN

The disk arm starts at one end of the disk, and moves toward the other end, servicing requests until it gets to the other end of the disk, where the head movement is reversed and servicing continues.

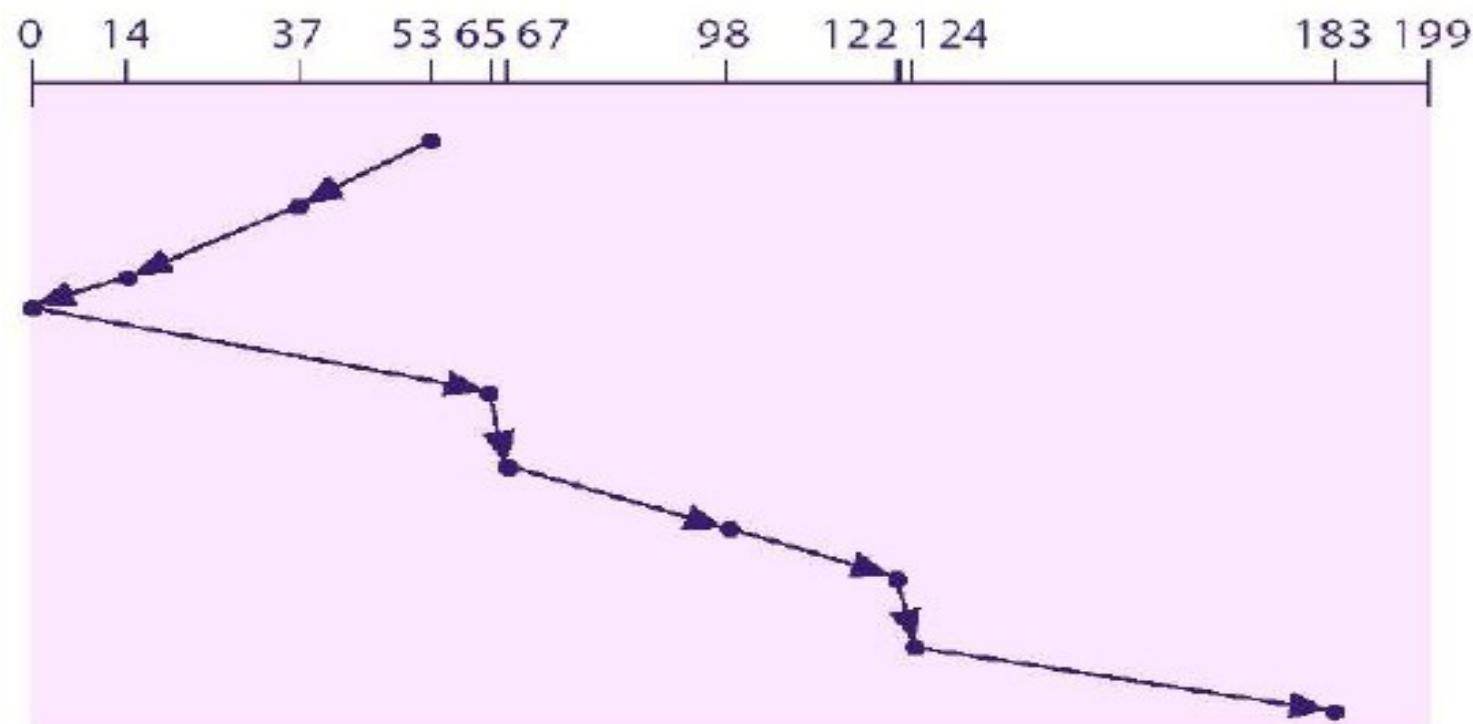
**Sometimes called the elevator algorithm.**

- Advantages:
  - Decreases variances in seek and
  - improve response time.
- Problem:
  - Starvation is possible if there are repeated request in current track.

# SCAN

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



Total Head Movement = 208 cylinders

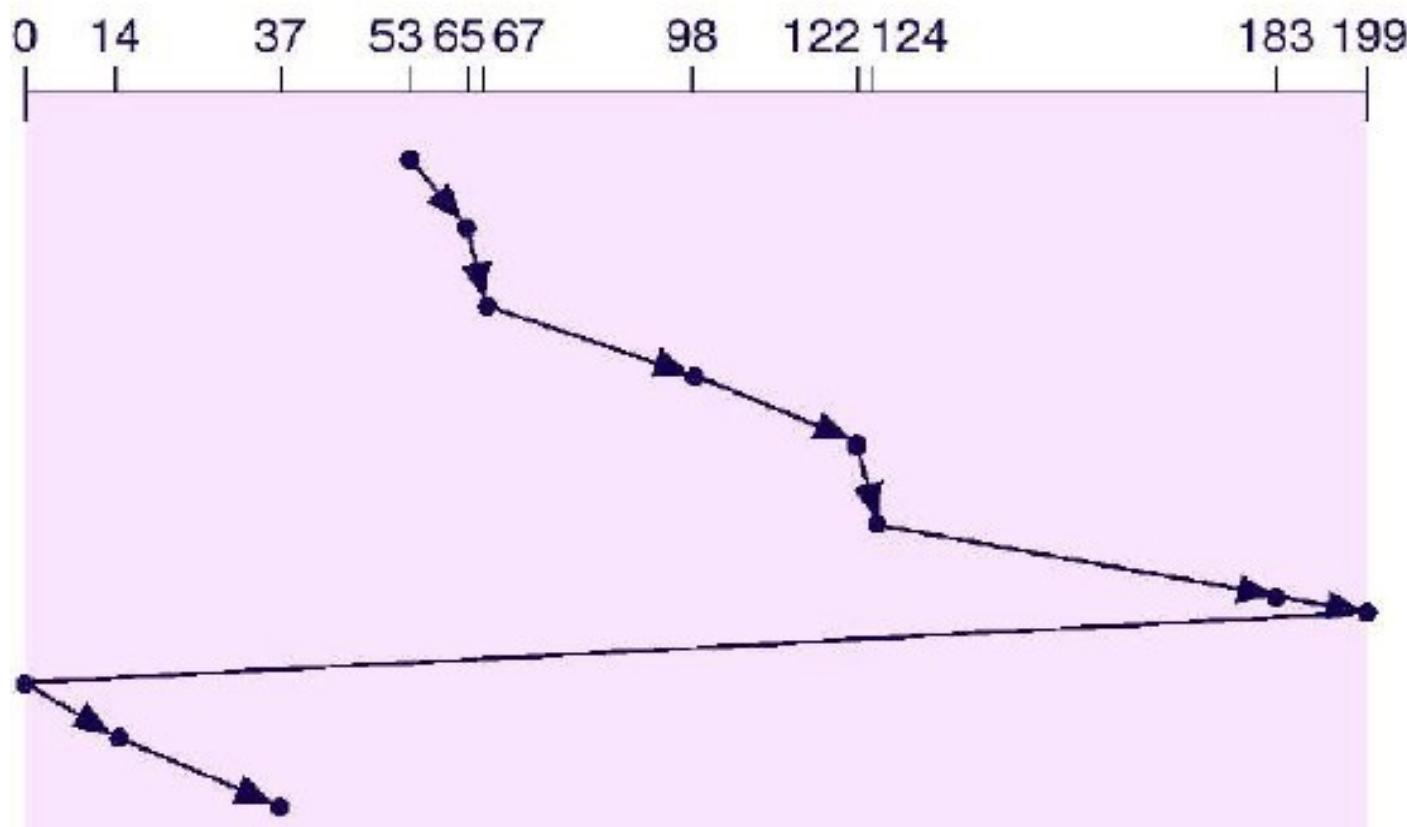
# C-SCAN

- When a uniform distribution of request for cylinders, only the few request are in extreme cylinders, since these cylinders have recently been serviced.
  - Why not go to the next extreme?
- **Circular** SCAN is a variant of SCAN designed to provide a more uniform wait time.
- The head moves from one end of the disk to the other. Servicing requests as it goes. When it reaches the other end, however, it immediately returns to the beginning of the disk, without servicing any requests on the return trip.
- Treats the cylinders as a circular list that wraps around from the last cylinder to the first

# C-SCAN

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53

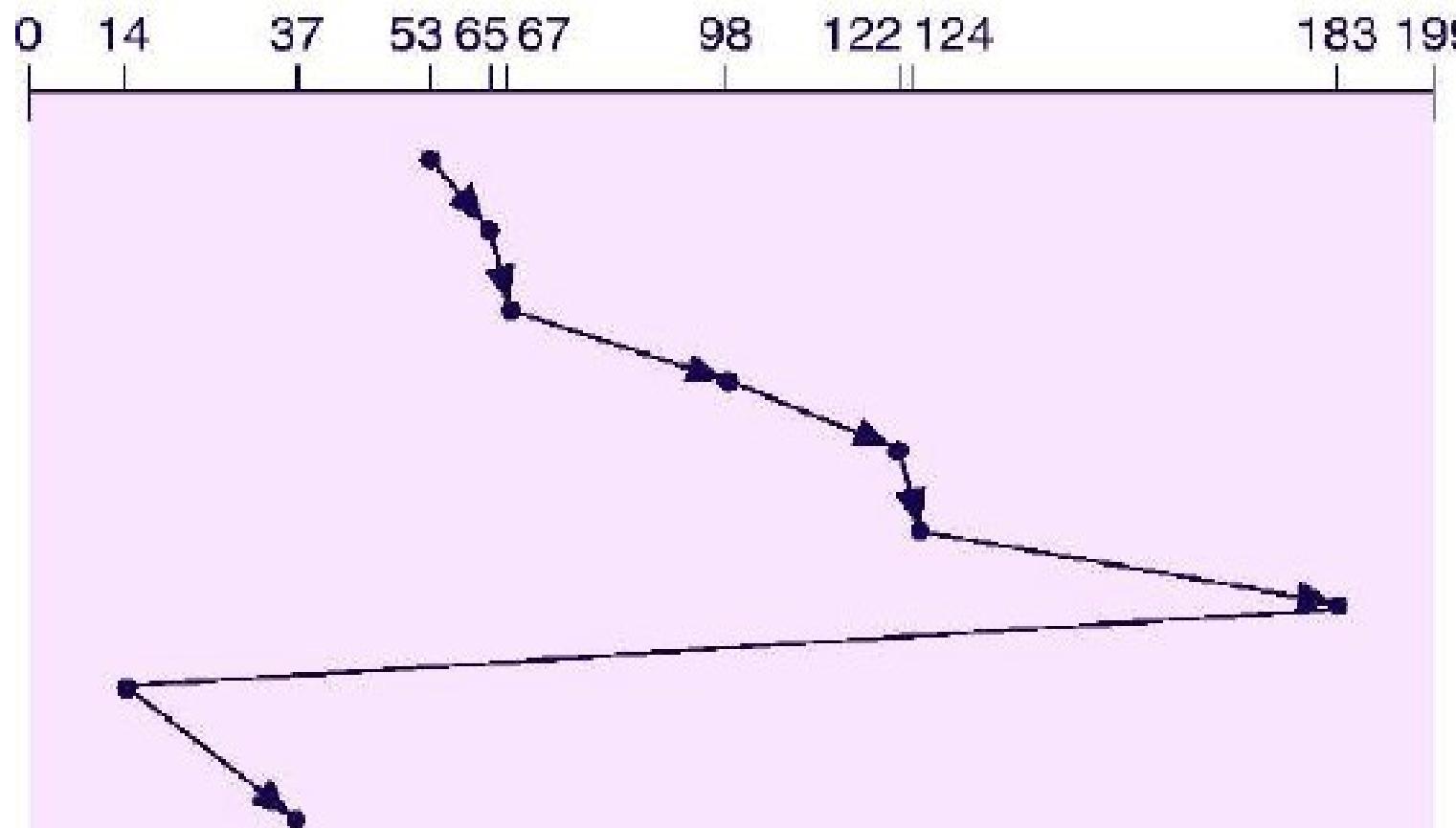


# C-LOOK

- Version of C-SCAN
- Arm only goes as far as the last request in each direction, then reverses direction immediately, without first going all the way to the end of the disk.

queue = 98, 183, 37, 122, 14, 124, 65, 67

head starts at 53



# SLED Vs RAID

- SLED
  - Single Large Expensive Disk
- RAID
  - Redundant Array of Inexpensive (Independent) Disks.

# RAID

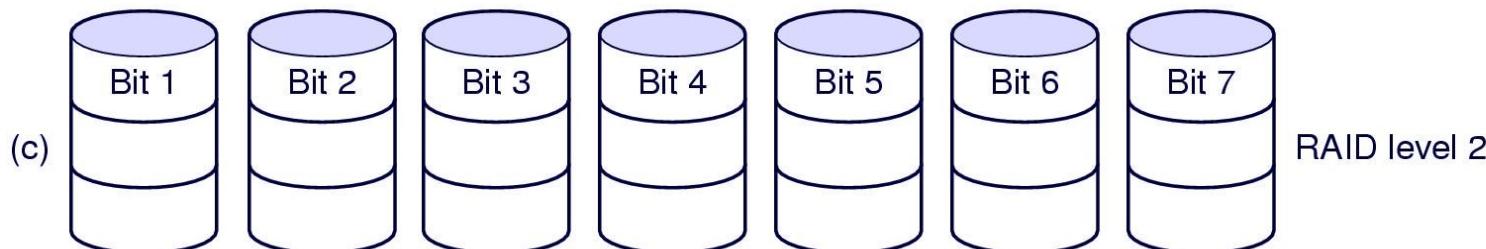
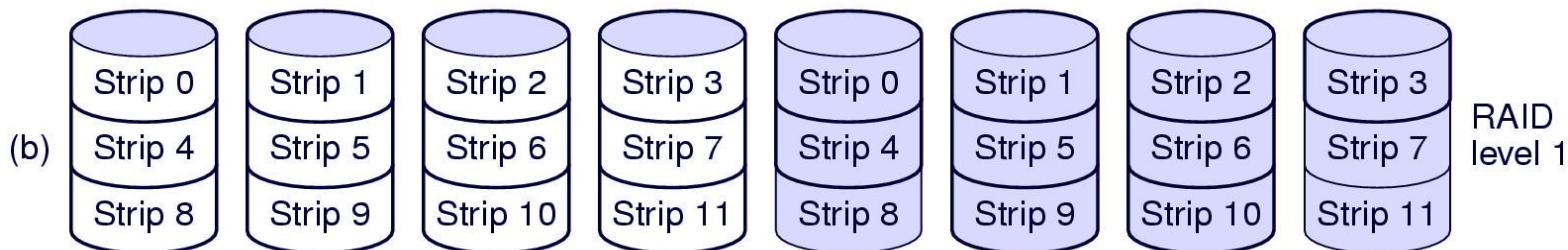
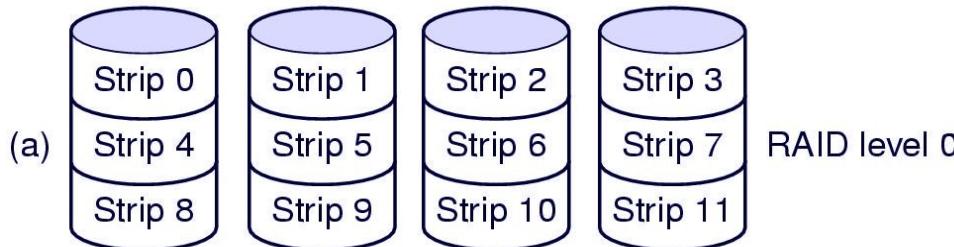
Redundant Array of Inexpensive (Independent) Disks.

CPU performance has been increasing exponentially over the past decade, roughly doubling every 18 months. Not so with disk performance. In the 1970s, average seek times on minicomputer disks were 50 to 100 msec

- **Issues:** Disk performance, Amount of storage required & Reliability
- A technique of organizing multiple disks to address above issues is RAID.
- RAID allows more than one disk to be used for a given operation, and allows continued operation and even automatic recovery in the face of disk failure.
- **Implemented in hardware or in OS.**

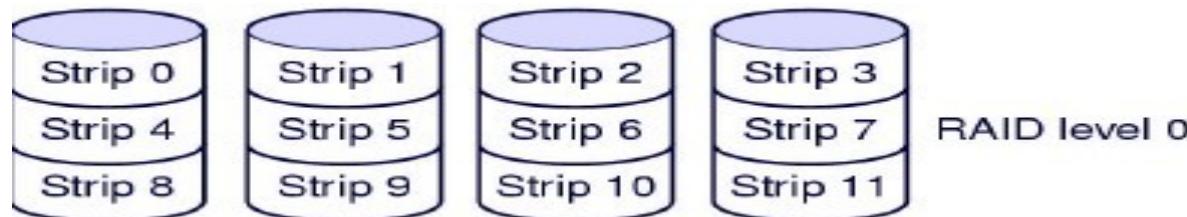
# RAID Levels

There are six types of organization of RAID called RAID levels



# RAID Level 0

- RAID level 0 creates one large virtual disk from a number of smaller disks.
- Storage is grouped into logical units called strips (It is a collection of sectors) with the size of a strip being some multiple of sector size.
- The virtual storage is sequence of strips interleaved among the disks in the array.
- Distributing data over multiple drives like this is called **striping**.

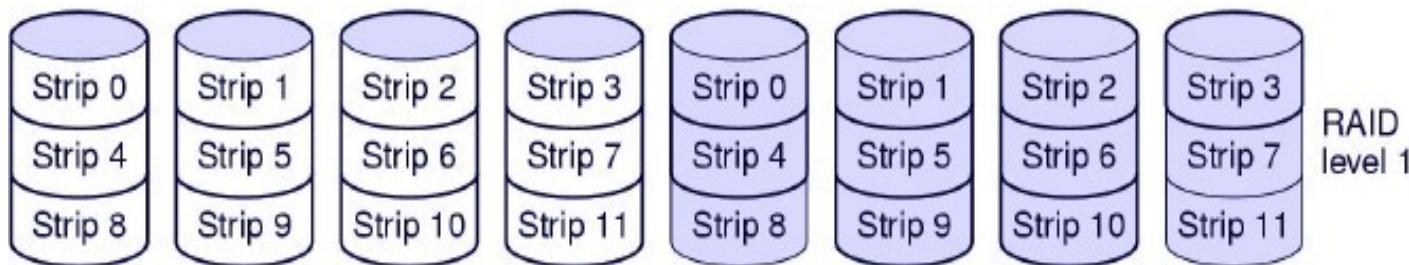


# Discussion : RAID level 0

- If the software issues a command to read a data block consisting of four consecutive strips starting at a strip boundary, the RAID controller will break this command up into four separate commands, one for each of the four disks, and have them operate in parallel. Thus we have **parallel I/O** without the software knowing about it.
- RAID level 0 works worst with operating systems that habitually ask for data one sector at a time. The results will be correct, but there is no parallelism and hence no performance gain.
- Another disadvantage of this organization is that the reliability is potentially worse than having a SLED.
- Because no redundancy is present in this design, it is **not really a true RAID**.

# RAID level 1

- Stores duplicate copy of each strip, with each copy on a different disk.



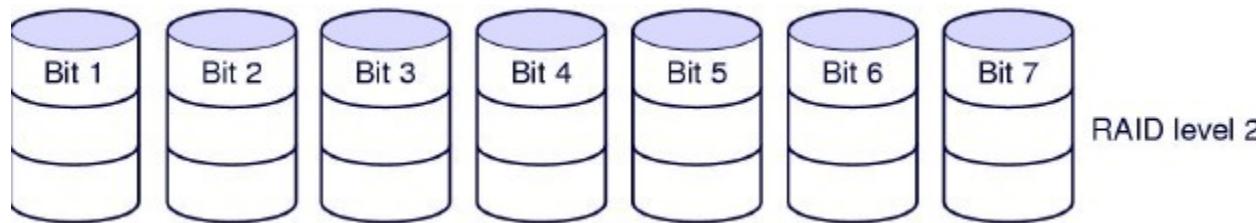
- Advantages:** Excellent reliability; if drive crashes, the copy is used. Read performance can be achieved.
- Disadvantages:** Write Performance is no better than in single drive.

# Discussion : RAID level 1

- write performance is no better than for a single drive, but read performance can be up to twice as good.
- Fault tolerance is excellent: if a drive crashes, the copy is simply used instead.
- Recovery consists of simply installing a new drive and copying the entire backup drive to it.
- Double space !!!

# RAID Level 2

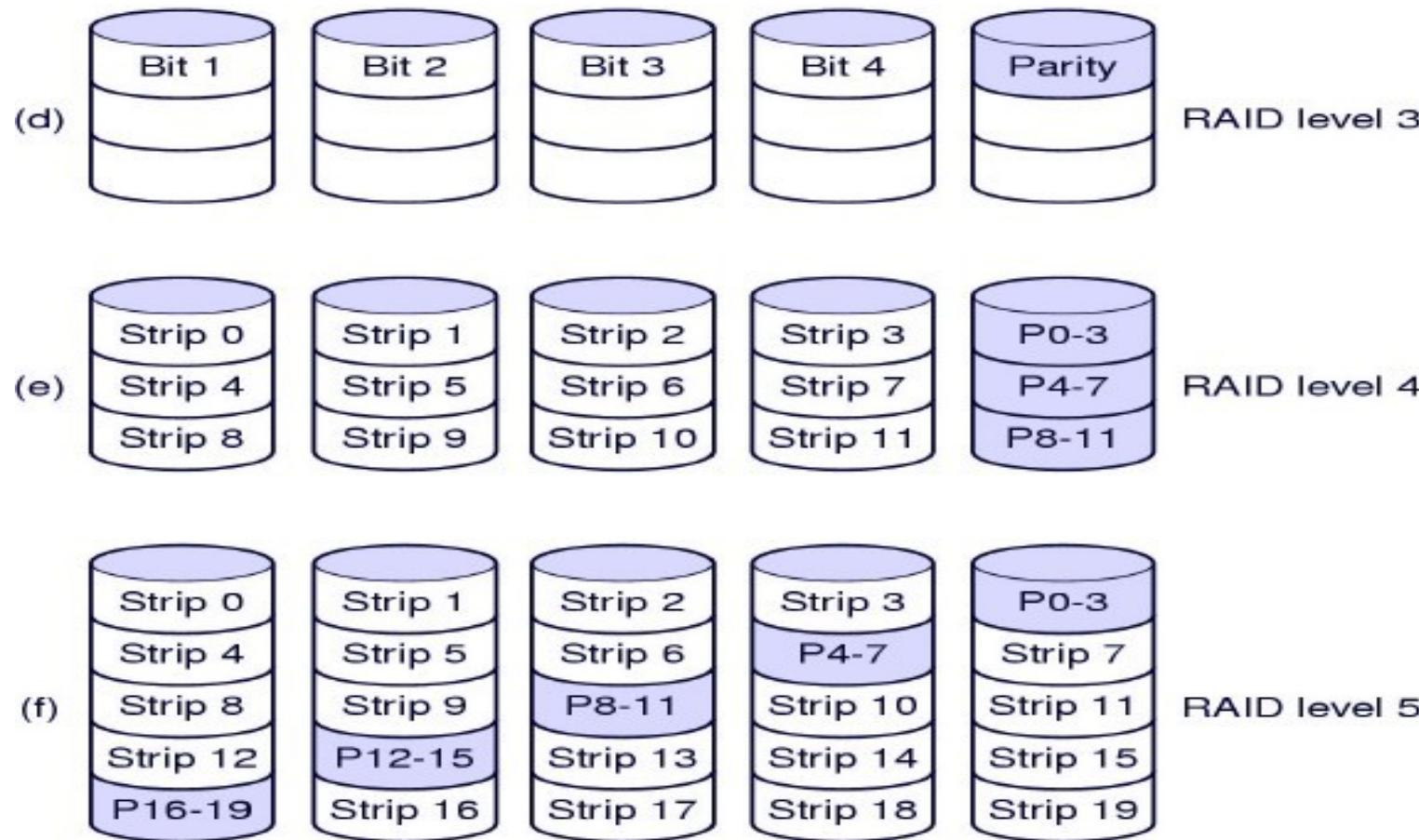
- An **error-correcting code** is used for corresponding bits on each data disks.
- Error-correcting scheme store two or more extra bits, and can reconstruct the data if a single bit get damaged
- For Example, the first bit of each byte is stored in disk 1, second bit in disk 2, and until eighth bit in disk 8, and error correcting bits are stored in further disks. If one of the disk fail, the remaining bits of the byte and associated error- correction bits can be read from other disks and be used to reconstruct the damage data.



# RAID level 2

- Advantages: Total parallelism at bit level.
- Disadvantages: Requires substantial number of drives. Out of 7 drives, 3 are for error correcting code
- Hamming code of 7 bit= 4 bit original +3 bit error correcting code

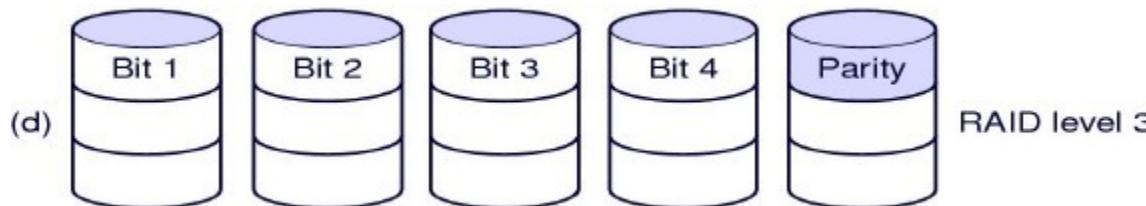
# RAID levels



# RAID level 3

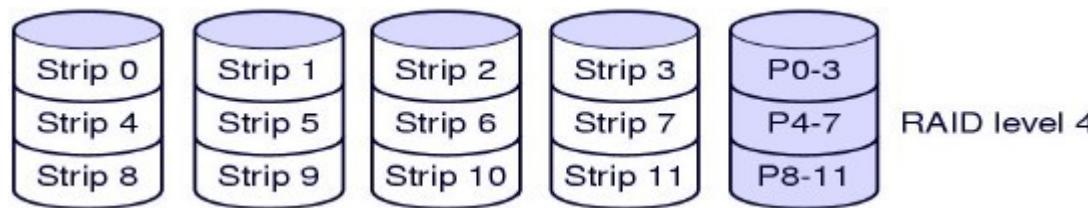
## Simplified version of Level 2.

- A *single parity bit is used instead of error-correcting code*, hence required just one extra disk.
- If any disk in the array fails, its data can be determined from the data on the remaining disks.
- It is as good as Level 2 but is less expensive in the number of extra disks.



# RAID Level 4

- It uses block-level striping, as in Level 0, and in addition keeps a parity block on separate disk for corresponding blocks from other disks.
- If one of the disks fails, the parity block can be used with the corresponding blocks from other disks to restore the blocks of the fail disks.
- The transfer rate for large read as well as large write is high since reads and writes in parallel but small read and write can not be in parallel.

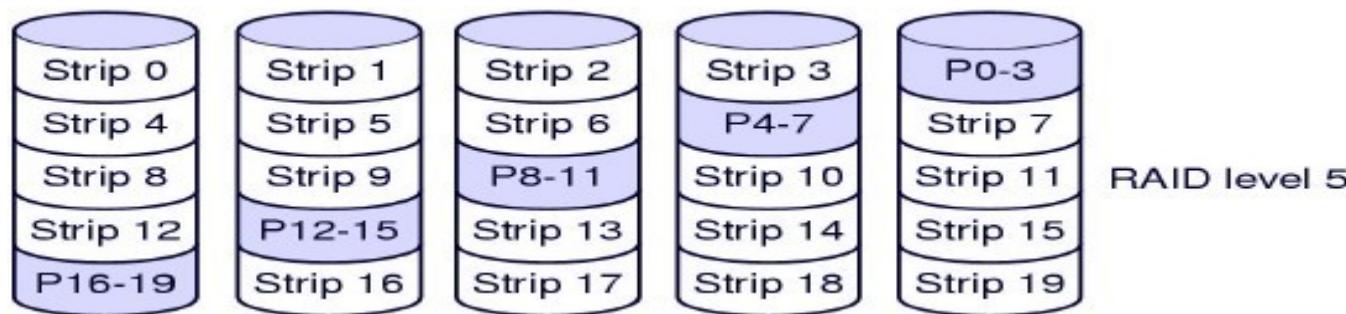


# RAID level 4

- This design protects against the loss of a drive but performs poorly for small updates. If one sector is changed, it is necessary to read all the drives in order to recalculate the parity, which must then be rewritten.
- As a consequence of the heavy load on the parity drive, it may become a bottleneck.

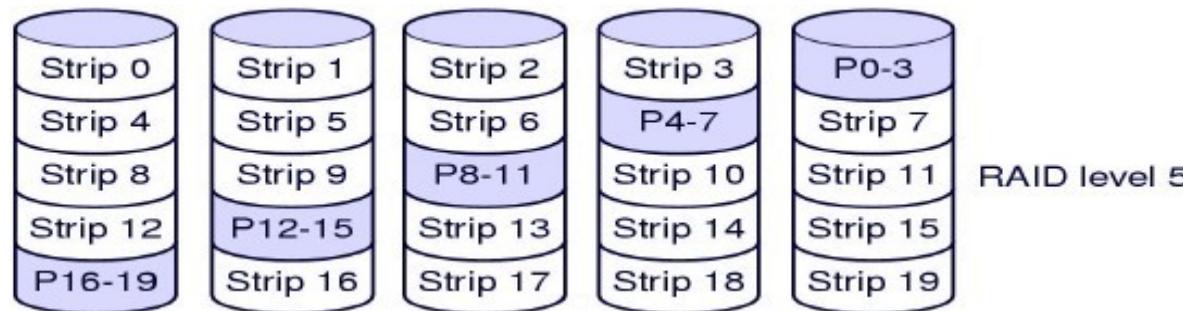
# RAID level 5

- The bottleneck of parity is eliminated in RAID level 5 by distributing the parity bits uniformly over all the drives, round robin fashion,



# RAID level 5

- Similar to level 4 but parity information is distributed in all disks.
- For each block one of the disk stores parity and other stores data.
- For example, with an array of five disks, the parity for nth blocks is stored in disks  $(n \bmod 5) + 1$ ; the nth block of the other four disks stores actual data for that block.

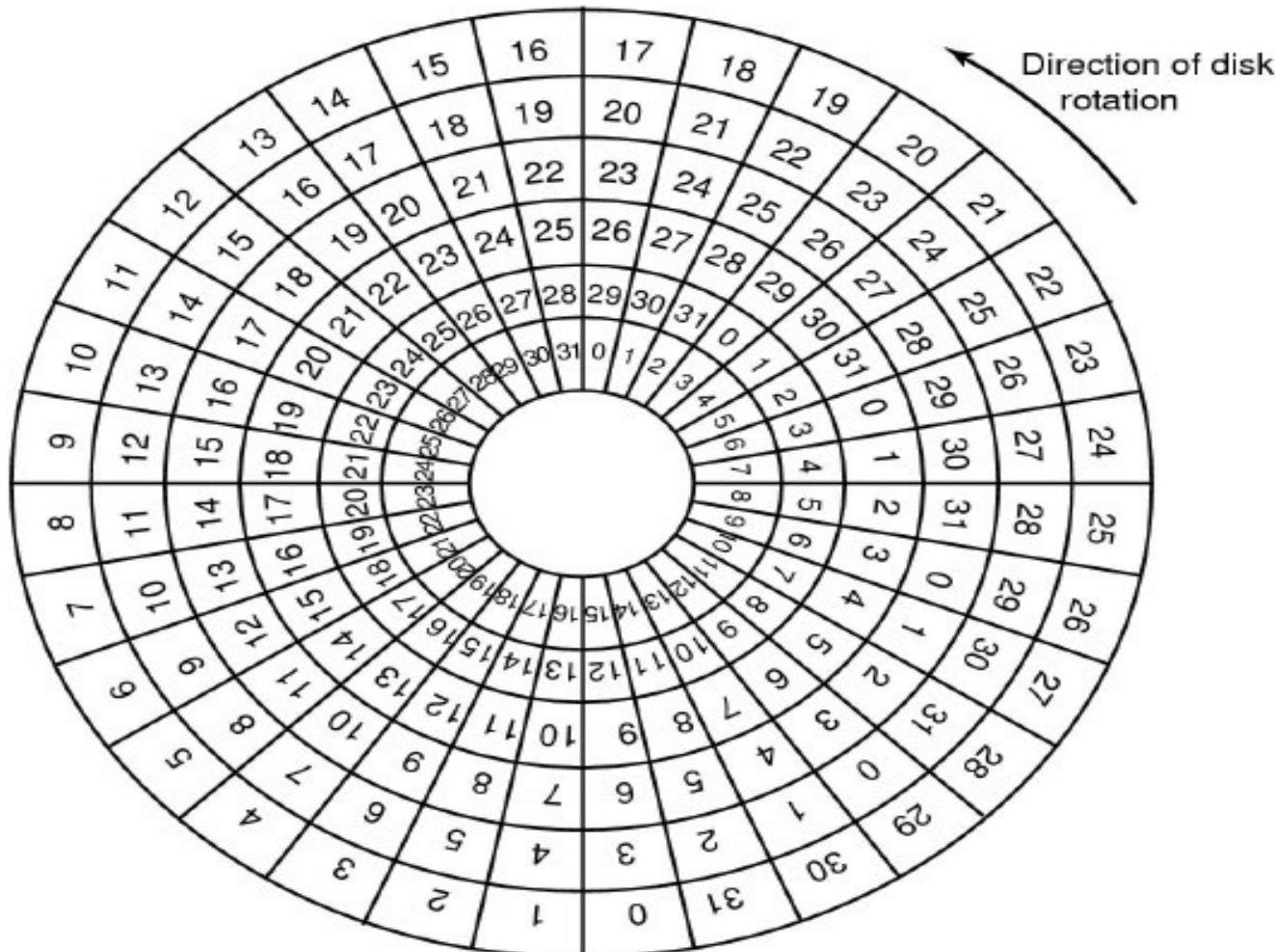


# Disk Formatting

- Before a disk can store data, it must be divided into sectors that the disk controller can read and write, called **low-level formatting**.
- The **sector** typically consists of preamble, data and Error correcting code(ECC).
- The preamble contains the cylinder and sector number and the ECC contains redundant information that can be used to recover from read error.
- The size depends upon the manufacturer, depending on reliability.



# Disk Formatting

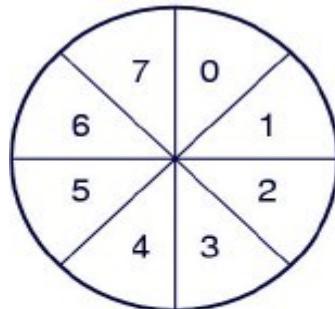


# Disk Formatting

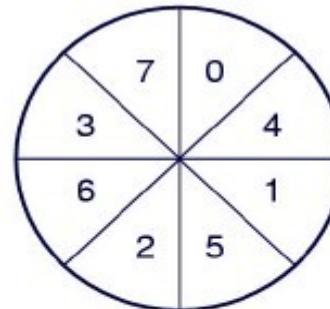
- If disk I/O operations are limited to transferring a single sector at a time, it reads the first sector from the disk and doing the ECC calculation, and transfers to main memory, during this time the next sector will fly by the head.
- When transferring completes, the controller will have to wait almost an entire rotation for the second sector to come around again.

# Disk Formatting

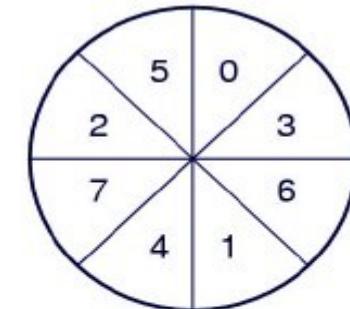
- This problem can be eliminated by numbering the sectors in an interleaved fashion when formatting the disk.
- According to the copying rate, interleaving may be of single or double.



(a)



(b)



(c)

a) No interleaving

b) Single interleaving    c) Double interleaving

# RAM Disk

- RAM Disk is virtual block created from main device memory.
- Commands to read or write disks blocks are implemented by RAM disk driver.
- It completely eliminates seek and rotational delays suffered in disk devices because main memory is direct access.
- RAM disks are particularly useful for storing files that are frequently accessed or temporary.
- RAM disks are especially used in high performance applications.
- Some OS define the RAM disks at boot time, other dynamically.

# RAM Disk: Discussion

- **Disadvantages: cost and volatility.**
- To implement RAM disk, OS must reserve a section of main memory for RAM disk use. So additional space should be purchased otherwise memory will be insufficient for use by process.
- It loses its content when power is off. If RAM disk is used to store file system, that file system must be remade each time a system is boot.
- The volatility is solved by providing battery backups.

# Exercise

- Suppose that a disk drive has 5,000 cylinders, numbered 0 to 4999. The drive is currently serving a request at cylinder 143, and the previous request was at cylinder 125. The queue of the pending requests, in FIFO order, is
  - – 86, 1470, 913, 1774, 948, 1502, 1022, 1750, 130.
- Starting from the current head position, what is the total distance (in cylinders) that the disk arm moves to satisfy all the pending requests for each of the following disk-scheduling algorithms?
- a) FCFS b) SSTF c) SCAN d) C-SCAN e) C-LOOK

# Exercise

- A disk has 8 sectors per track and spins at 600 rpm. It takes the controller 10ms from the end of one I/O operation before it can issue a subsequent one. How long does it take to read all 8 sectors using the following interleaving system?
  - a) No interleaving b) Single interleaving c) Double interleaving

# Thank You

# Protection and Security in Operating System

System Protection: protection principles and domain, access matrix and its implementation, access controls and rights, capability-based system, language-based protection.

# System Protection

- Protection refers to strategy for controlling the access of program, processes, and users to resources.
- It ensure that resources are used in consistent ways.
- Further more, it ensure that each resource is accessed correctly and only by those process that are allowed to do so.

# System Protection

- If a computer program is run by an unauthorized user, then he/she may cause severe damage to computer or data stored in it.
- So a computer system must be protected against unauthorized access, malicious access to system memory, viruses, worms etc.

# Example of Protection

- **Privileged instructions** – the process must be executing in kernel mode in order to execute without causing an exception.
- **Memory protection** – the kernel address space is protected from user level instructions. Similarly, one process' address space is protected from access by another.
- **File system** – one user's files are protected from access by another user.

# Goals of Protection

- to prevent malicious misuse of the system by users or programs.
- To ensure that each shared resource is used only in accordance with system *policies*.
- To ensure that errant programs cause the minimal amount of damage possible.

System protection only provide the *mechanisms* for enforcing policies and ensuring reliable systems. It is up to administrators and users to implement those mechanisms effectively.

# Principles of Protection

- ***Principle of least privilege :***
  - dictates that programs, users, and systems be given just enough privileges to perform their tasks.
  - Ensure a little damage even if system fails
- Example:
  - if a program needs special privileges to perform a task, it is better to make it a **Set Group ID (SGID)** program with group ownership of "network" or "backup" or some other pseudo group, rather than **Set User ID(SUID)** with root ownership

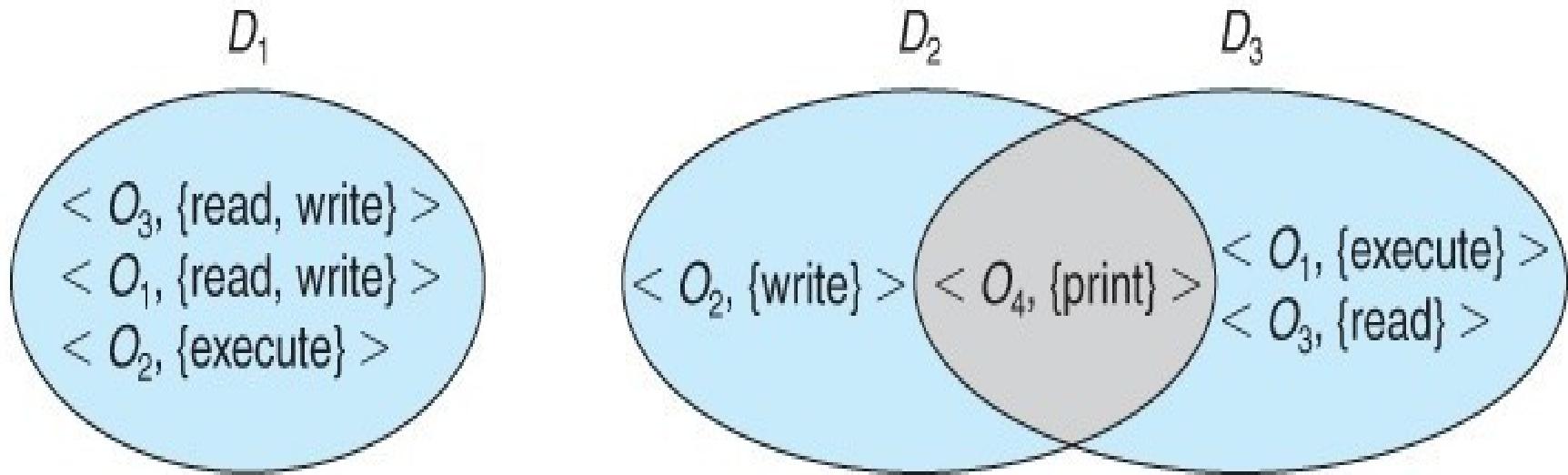
# Domain of Protection

- Computer System consists of objects and process
- Objects consists of both hardware objects (CPU, Memory)and software objects(files and programs)
- Each domain defines a set of objects and the types of operations that may be invoked on each object.

# Domain of Protection

- An **access right** is the ability to execute an operation on an object.
- A domain is defined as a set of
  - < object, { access right set } > pairs
  - Example if domain D has the access right <**file F**,  
**{read, write}**> then a process executing on domain D can both read and write file F. However, it can't perform other operation on it.

# Domain of Protection



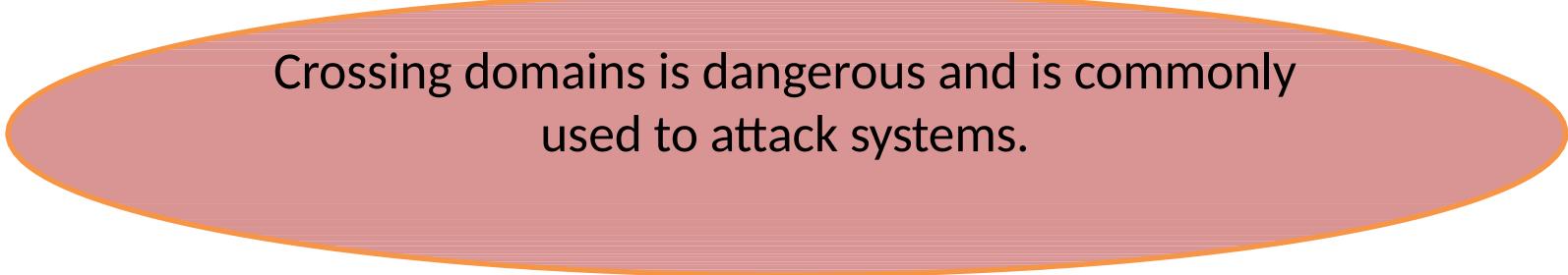
Domain can overlap: Permission in overlap are available to both domains.

# Protection Domain

- A domain can be realized in various way:
  - Each user may have a domain. In this case the set of object that can be accessed depends upon the **identity of user**.
  - Each process may have a domain. In this case the set of object that can be accessed depends upon the identity of process.

# Crossing Domain

- We want users to have controlled access to resources they don't have direct access to.
- e.g. a database, particular hardware, networks
- So we give the user access to a program that does have access to the restricted resource.



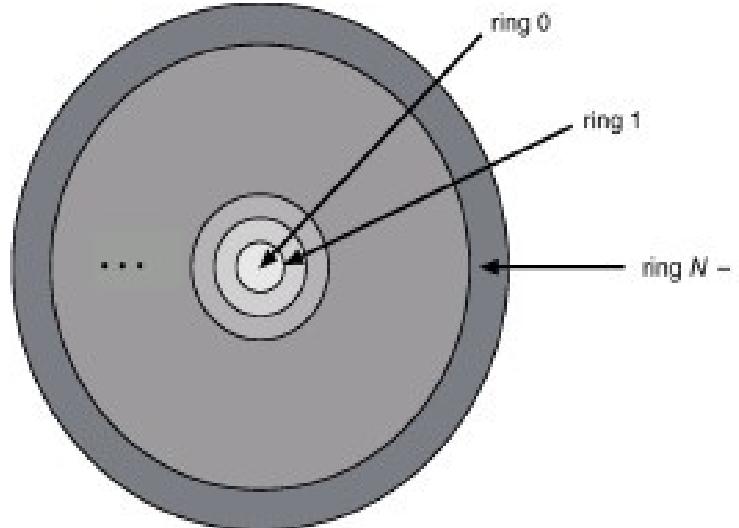
Crossing domains is dangerous and is commonly used to attack systems.

# An Example: UNIX

- UNIX associates domains with users.
- Here users ID is used to identify the domains

# Multics ring structure: Domain Protection

- Let  $D_i$  and  $D_j$  be any two domain rings.
- If  $j < i \Rightarrow D_i \subsetneq D_j$  Then a process executing in  $D_j$  has more privileges than one executing in  $D_i$ .



# Access Matrix

- A *model of system protection called access matrix,*
  - Where columns represent different system resources and rows represent different protection domains.
  - Entries within the matrix indicate what access that domain has to that resource.

# Example

object domain	$F_1$	$F_2$	$F_3$	printer
$D_1$	read		read	
$D_2$				print
$D_3$		read	execute	
$D_4$	read write		read write	

# Domain Switching

object domain	$F_1$	$F_2$	$F_3$	laser printer	$D_1$	$D_2$	$D_3$	$D_4$
$D_1$	read		read			switch		
$D_2$				print		switch	switch	
$D_3$		read	execute					
$D_4$	read write		read write		switch			

# Access Matrix : Owner Right

- The **owner** right adds the privilege of adding new rights or removing existing ones:

object domain	$F_1$	$F_2$	$F_3$
$D_1$	owner execute		write
$D_2$		read* owner	read* owner write
$D_3$	execute		

(a)

object domain	$F_1$	$F_2$	$F_3$
$D_1$	owner execute		write
$D_2$		owner read* write*	read* owner write
$D_3$		write	write

(b)

# Implementation of Access Matrix

- It Contains a set of ordered triples  
**<domain,object,rights-set>**
- Whenever an operation M is executed on an object Oj within domain Di the global table is searched for a triple **<Di,Oj,R<sub>k</sub>>** where M ∈ R<sub>k</sub>
- If this triple is found, the operation is allowed to continue or else an exception is raised.

# Implementation of Access Matrix

- **Global Table**
  - The simplest approach is one big global table with < domain, object, rights > entries.
  - Unfortunately, this table is very large ( even if sparse ) and so cannot be kept in memory ( without invoking virtual memory techniques. )
  - There is also no good way to specify groupings - If everyone has access to some resource, then it still needs a separate entry for every domain.

# Implementation of Access Matrix

- Access Lists for Objects
  - Each column implemented as an access list for one object
  - Resulting per-object list consists of ordered pairs  $< \text{domain}, \text{rights-set} >$  defining all domains with non-empty set of access rights for the object
  - Easily extended to contain default set -> If  $M \in$  default set, also allow access.

# Access List (cont...)

- Each column = Access-control list for one object .
- Defines who can perform what operation

Domain 1 = Read, Write

Domain 2 = Read

Domain 3 = Read

# Implementation of Access Matrix

- **Capability Lists for Domains**
  - Instead of object-based, list is domain based
  - **Capability list** for domain is list of objects together with operations allowed on them
  - Object represented by its *name or address*, called a **capability**
  - Execute operation M on object O<sub>j</sub>, process requests operation and specifies capability as parameter
    - Possession of capability means access is allowed
  - Capability list associated with domain but never directly accessible by domain
    - Rather, protected object, maintained by OS and accessed indirectly
    - Like a “secure pointer”
    - Idea can be extended up to applications

# Capability List(cont..)

- Each Row = Capability List (like a key)  
For each domain, what operations allowed on what objects
  - Object F1 – Read
  - Object F4 – Read, Write, Execute
  - Object F5 – Read, Write, Delete, Copy

# Implementation of Access Matrix

## A Lock-Key Mechanism

- Each resource has a list of **unique bit patterns, termed locks.**
- Each domain has its own list of unique bit patterns, termed **keys**.
- Access is granted if one of the domain's keys fits one of the resource's locks.
- Again, a process is not allowed to modify its own keys.

# Comparison of Implementations

- Many trade-offs to consider
  - Global table is simple, but can be large
  - Access lists correspond to needs of users
    - Determining set of access rights for domain non-localized difficult
    - Every access to an object must be checked
      - Many objects and access rights -> slow
  - Capability lists useful for localizing information for a given process
    - But revocation capabilities can be inefficient
  - Lock-key effective and flexible, keys can be passed freely from domain to domain, easy revocation

# Access Matrix Implementation

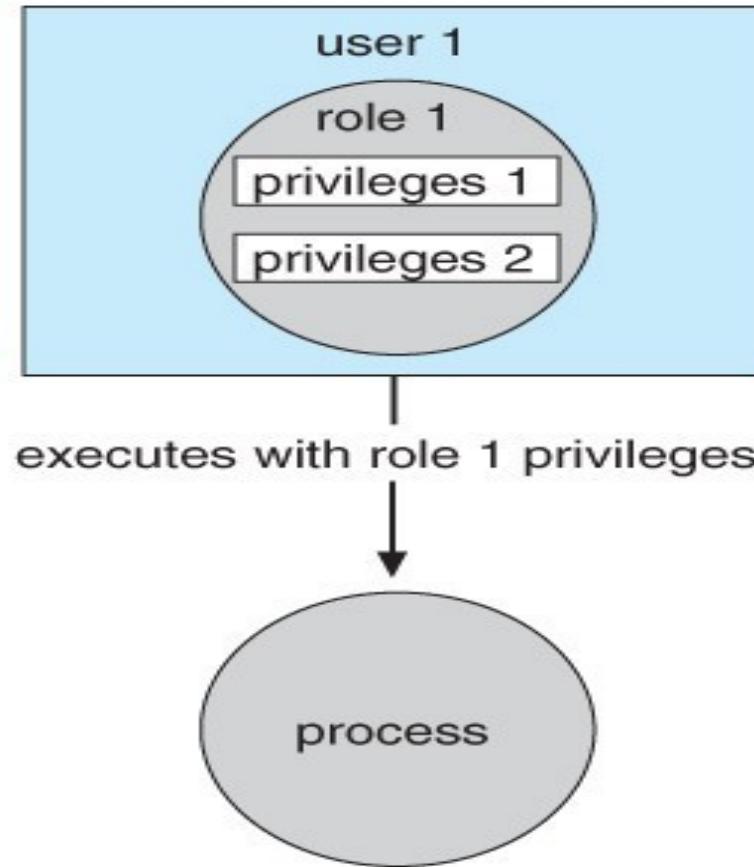
- Each method has advantage and disadvantages
- Many commercial system uses combination of these system

# Access Controls

**Role-Based Access Control, RBAC** assigns privileges to users, programs, or roles as appropriate, where "privileges" refer to the right to call certain system calls, or to use certain parameters with those calls.

- RBAC supports the principle of least privilege, and reduces the susceptibility to abuse as opposed to SUID or SGID programs

# Role-Based Access Control



Role based access control in solaris 10

# Thank You

# Security in Operating System

System Security: Program threats, system and network threats, Cryptography as a Security Tool, User Authentication, Firewalling, Computer security classification, Guarded Models for intrusion detection  
(Paper Based Study)

# System Protection (Preview)

- Protecting files and other resources accidental misuse by ~~sb@p@g@t@n~~ system.
- This is for normal purposes

# System Security

- Deals with protecting systems from deliberate attacks, either internal or external individuals.
- Individuals intentionally attempts to steal information, damage information.

# Types of Violations

- Basically there are following breach of security:
  - *Breach of Confidentiality*
  - *Breach of Integrity*
  - *Breach of Availability*

*This is a commonly known as CIA triad in information security Theory*

# Breach of Confidentiality

- Theft of private or confidential information,
- For example: credit-card numbers, trade secrets, patents, secret formulas, manufacturing procedures, medical information, financial information, etc.

# Breach of Integrity

- Unauthorized ***modification*** of data, which may have serious indirect consequences.
- For example a popular game or other program's source code could be modified to open up security holes on users systems before being released to the public.

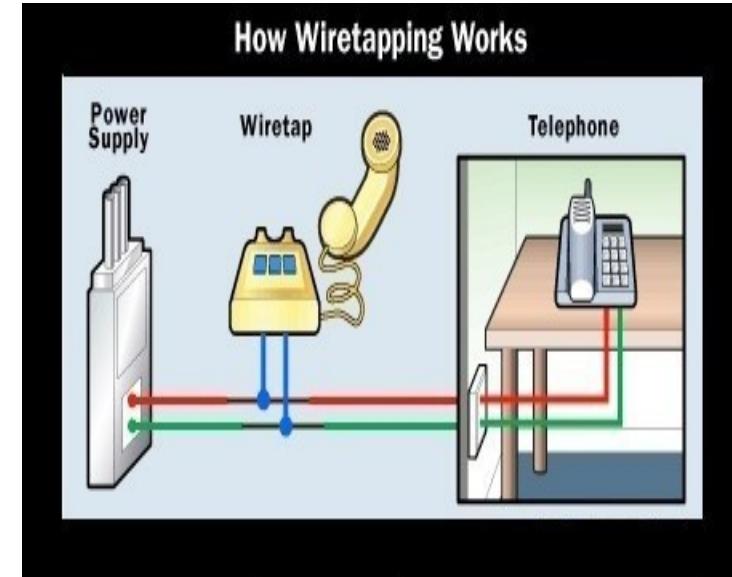
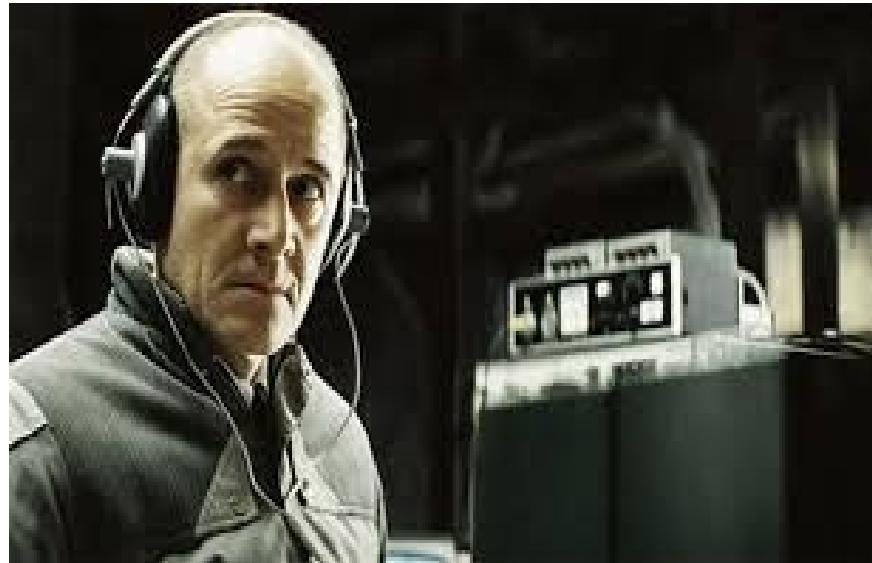
# Breach of Availability

- Unauthorized **destruction** of data, often just for the "fun" of causing havoc and for bragging rites. Vandalism of web sites is a common form of this violation.

# Types of attack

- In general, there are following possible attack in computer system.
  - Active attack Vs Passive Attack
  - Inside Attack Vs Outside Attack
  - Passive Attack
    - Eavesdropping
    - Wiretapping/port scan etc
  - Active Attack
    - Denial of Service
    - Man in the middle attack

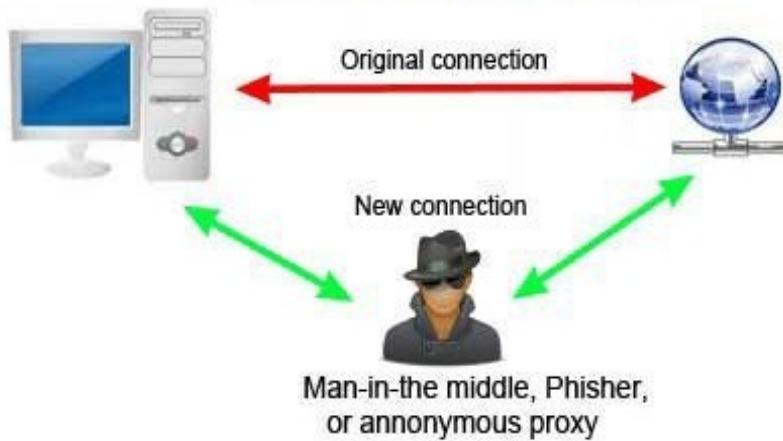
# Some Pictures



Eavesdropping and wiretapping

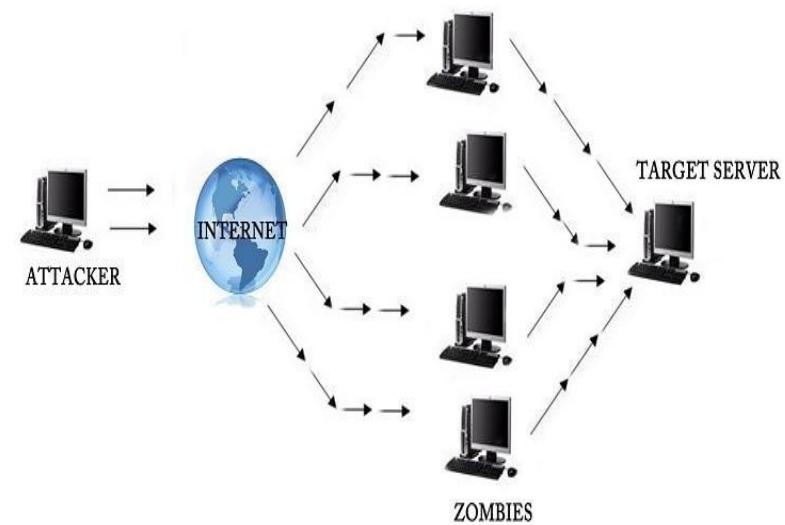
# Some pictures

Man-in-the-middle attack



ComputerHope.com

**DENIAL OF SERVICE ATTACK**



Picture Source: Google Image

# Level of Protections

- There are four levels at which a system must be protected:
  - **Physical**
  - **Human**
    - social engineering,
    - **Phishing**
    - **Dumpster Diving**
    - **Password Cracking**
  - **Operating System**
  - **Network**

# Threat

---

- Program Threat
- System threat

# Program Threat

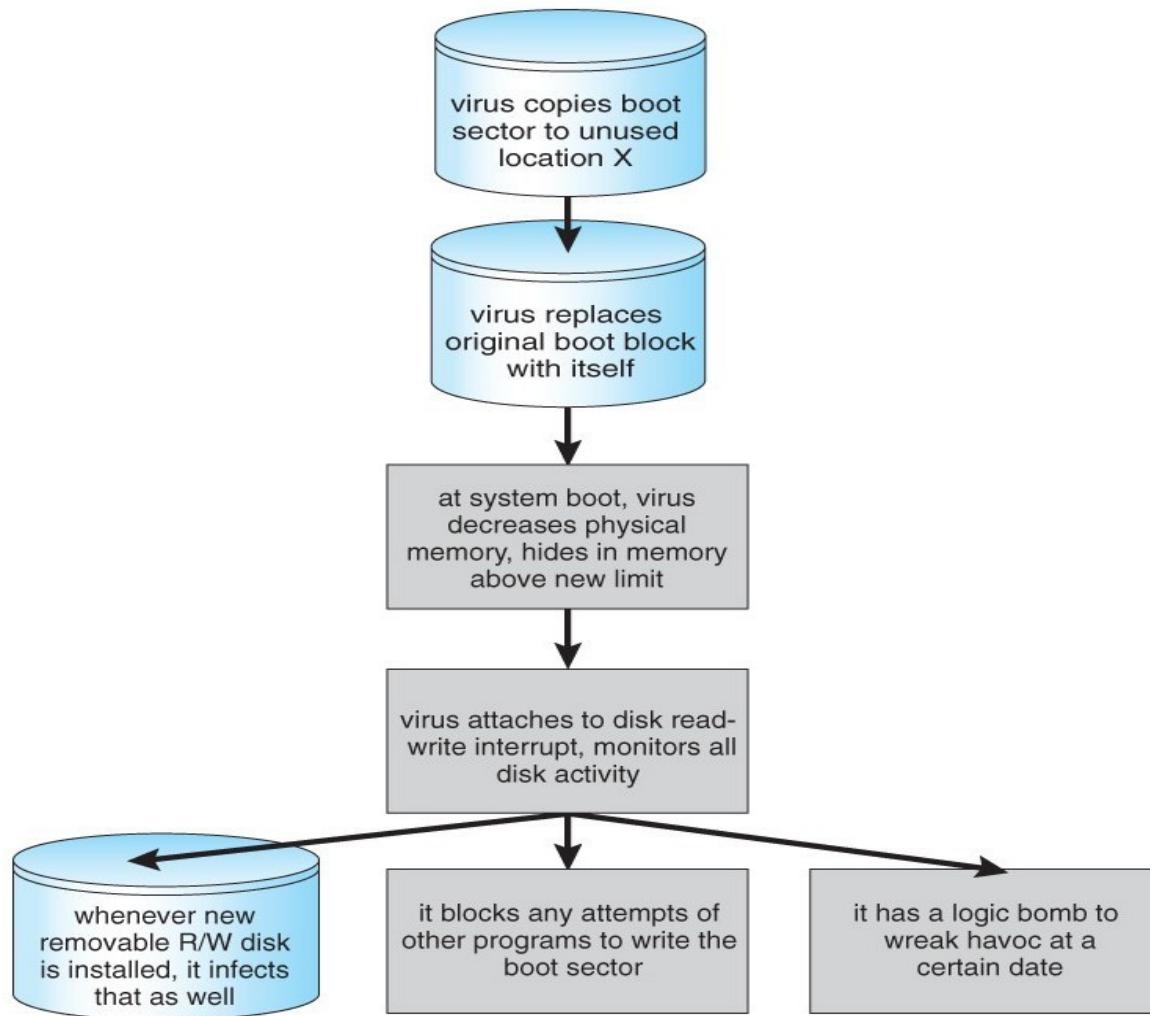
- A *Trojan Horse* is a program that secretly performs some maliciousness in addition to its visible actions.
- A classic Trojan Horse is a **login emulator**,
  - which records a user's account name and password, issues a "password incorrect" message, and then logs off the system
  - The user then tries again ( with a proper login prompt ), logs in successfully, and doesn't realize that their information has been stolen

# Threat: Trap Door

- A *Trap Door* is when a designer or a programmer ( or hacker ) deliberately inserts a security hole that they can use later to access the system
- A clever trap door could be inserted into a compiler, so that any programs compiled with that compiler would contain a security hole

# Other threats

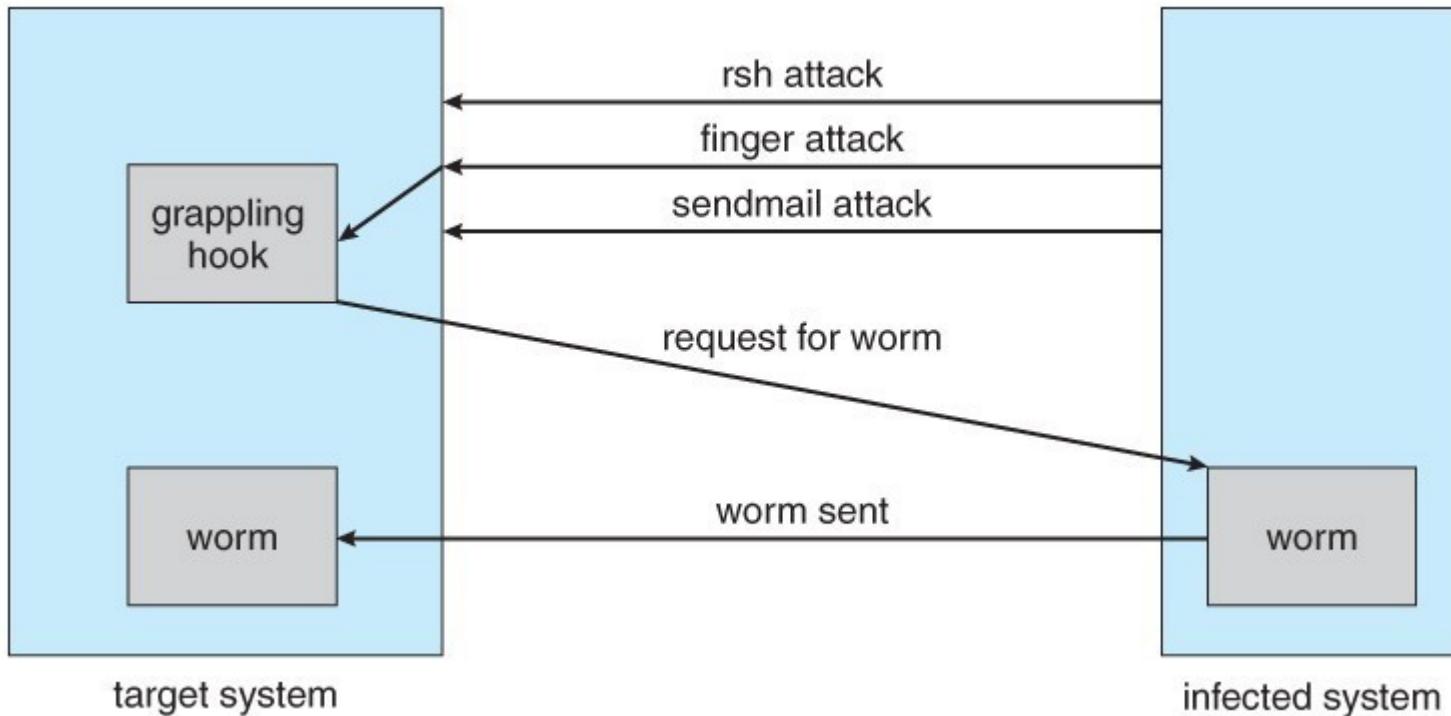
- **Logic Bomb**
- **Stack and Buffer Overflow**
- **Viruses:** designed to replicate itself



# System and Network Threats

- A **worm** is a process that uses the fork / spawn process to make copies of itself in order to wreak havoc on a system.
- Worms consume system resources, often blocking out other, legitimate processes.
- Worms that propagate over networks can be especially problematic, as they can tie up vast amounts of network resources and bring down large-scale systems.

# The Morris Internet worm



# Port Scanning

- **Port Scanning** is technically not an attack, but rather a search for vulnerabilities to attack.
- Try to connect to every node and attempt to contact.
- Once the port is determined, next step is to determine what daemon is listening, and is there any flaw in the version of daemon.
- **nmap** ( <http://www.insecure.org/nmap> ) and  
**nessus** ( <http://www.nessus.org> )

# Denial of Service

- *Denial of Service ( DOS )* attacks do not attempt to actually access or damage systems, but merely to clog them up so badly that they cannot be used for any useful work. Tight loops that repeatedly request system services are an obvious form of this attack.
- DDOS: distributed DOS

# Cryptography as Security tools

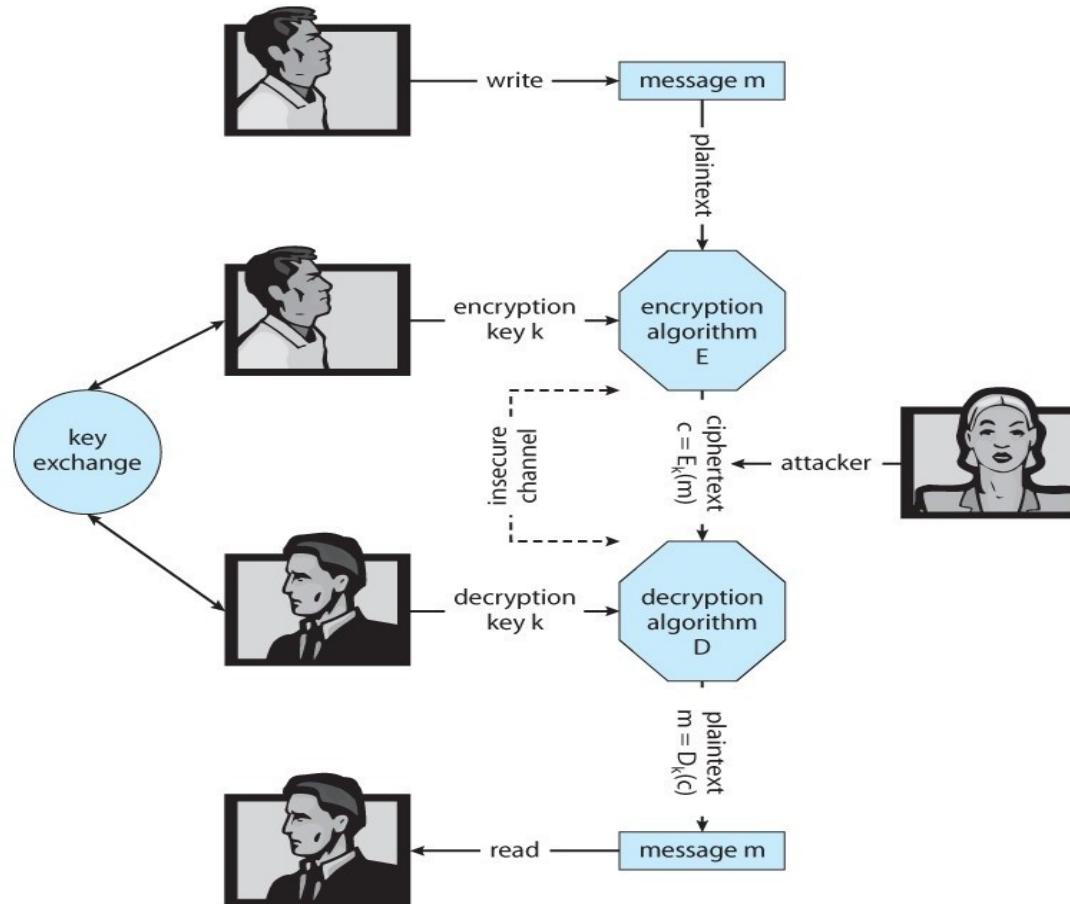


Fig: Encryption and Decryption

# Implementation of Cryptography

- Network communications are implemented in multiple layers - Physical, Data Link, Network, Transport, and Application being the most common breakdown.
- At the network layer the most common standard is **IPSec**, a secure form of the IP layer, which is used to set up **Virtual Private Networks, VPNs**. At the transport layer the most common implementation is SSL.

# Research Paper Review

[dl.acm.org/citation.cfm?id=1255345](http://dl.acm.org/citation.cfm?id=1255345)

 **DIGITAL LIBRARY**

**Guarded models for intrusion detection**

Full Text:  PDF  [Get this Article](#)

Author: [Hassen Saïdi](#) [SRI International](#)

Published in:

- Proceeding  
[PLAS '07 Proceedings of the 2007 workshop on Programming languages and analysis for security](#)  
Pages 85-94

San Diego, California, USA — June 14 - 14, 2007  
ACM New York, NY, USA ©2007  
[table of contents](#) ISBN: 978-1-59593-711-7 doi:>[10.1145/1255329.1255345](https://doi.org/10.1145/1255329.1255345)

 2007 Article

 [Bibliometrics](#)

- Citation Count: 4
- Downloads (cumulative): 583
- Downloads (12 Months): 12
- Downloads (6 Weeks): 3

 [Feedback](#)

 [Recent authors with related interests](#) ▾ [Concepts in this article](#) ▾

powered by  
**IBM Watson™**

# Guarded Models For Intrusion Detection

Hassen Saïdi

Computer Science Laboratory  
SRI International  
[saidi@csl.sri.com](mailto:saidi@csl.sri.com)

## Abstract

Host-based intrusion detection systems that monitor an application execution and report any deviation from its statically built model have seen tremendous progress in recent years. However, the weakness of these systems is that they often rely on overly abstracted models that reflect only the control flow structure of programs, and therefore are subject to so-called "mimicry attacks". Authors of these models have argued that capturing more of the data flow characteristics of a program is necessary to prevent a large class of attacks, in particular, non-control-data attacks. In this paper, we present the guarded model, a novel model that addresses the various deficiencies of the state-of-the-art intrusion detection systems. Our model is a generalization of previous models that offers no false alarms, a very low monitoring overhead, and is automatically generated. Our model detects mimicry attacks by combining control flow and data flow analysis, but can also tackle the ever increasingly threatening non-control-data flow attacks. Our model is the first model built automatically by combining control flow and data flow analysis using state-of-the-art tools for automatic generation and propagation of invariants. Our model not only prevents intrusions, but allows in some cases the detection of application logic bugs. Such bugs are beyond the reach of current intrusion detection systems.

**Categories and Subject Descriptors** D [4]: 6 Security and Protection

**General Terms** Security, Verification

**Keywords** Static analysis, dynamic analysis, invariant generation, intrusion detection

program. Detection of attacks rely on the fact that code injection often results in the execution of a system call that is not invoked by the application at all, or a sequence of system calls that are executed in an order not allowed by the application code. Finally, false alarms are eliminated because the model is an overapproximation of the behavior of the application. Therefore, while the model might be porous and not detect attacks, it will never raise a false alarm. Starting with Wagner and Dean's paper, various models capturing the sequences of legitimate system calls have been proposed [32, 18, 13, 16, 8].

Current state-of-the-art host-based intrusion detection systems approaches struggle with several issues. The most important one is the precision of the model. The more precise the model is, the less an attack is possible. Current models are limited when it comes to handling complex control structures such as loops, recursive function calls, and circuits in function calls. These limitations produce today's models in forms of very crude overapproximations of the actual behavior of the application and are all subject to so-called *mimicry attacks* [33] that produce a sequence of system calls that conforms to the model but is not a sequence in the original code. The imprecision of state-of-the-art models comes from the weakness of the static analysis methods employed to construct them. These weaknesses can be summarized by the absence of any significant data flow analysis. Mimicry attacks exploit the nondeterminism that arises from the imprecision of the models. A new wave of attacks called, *non-control-data attacks* [10] exploit the nondeterminism in the application code itself and therefore allow the subversion of the intended data flow. We observe that the attacks mentioned above are all preventable by making sure that the application models used for intrusion detection capture as precisely as possible

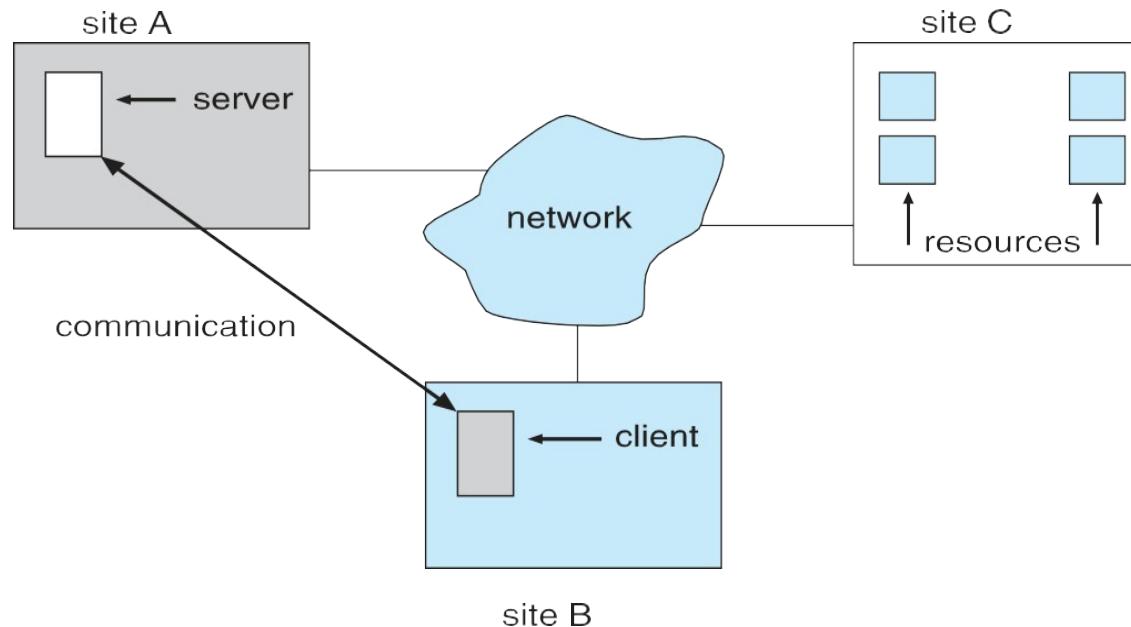
# Thank You

# Distributed Special Propose System

**Distributed Operating Systems:** Types of Network based Operating Systems, Network structure and topology, Communication structure and protocols, Robustness, Design Issues, **Distributed File Systems:** Naming and Transparency, Remote File Access, Stateful versus Stateless Service, File Replication.

# Introduction

- A distributed system is a collection of loosely coupled processors interconnected by a communications network and Processors variously called ***nodes, computers, machines, hosts.***



# Reasons for Distributed Systems

- Reasons for distributed systems
  - **Resource sharing**
    - Sharing and printing files at remote sites
    - Using remote specialized hardware devices
  - **Computation speedup – load sharing or job migration**
  - Reliability – detect and recover from site failure, function transfer, reintegrate failed site
  - Communication – **message** passing
    - All higher-level functions of a standalone system can be expanded to encompass a distributed system
  - Computers can be downsized, more flexibility, better user interfaces and easier maintenance by moving from large system to multiple smaller systems performing **distributed computing**

# Types of Distributed Operating Systems

- Network Operating Systems
- Distributed Operating Systems

# Network-Operating Systems

- Users are aware of multiplicity of machines
- Access to resources of various machines is done explicitly by:
  - Remote logging into the appropriate remote machine (telnet, ssh)
  - **Remote Desktop (Microsoft Windows)**
  - Transferring data from remote machines to local machines, via the File Transfer Protocol (FTP) mechanism
- Users must change paradigms - establish a **session**, give network-based commands
  - More difficult for users

# Distributed-Operating Systems

- Users are not aware of multiplicity of machines
  - Access to remote resources similar to access to local resources
- **Data Migration** – transfer data by transferring entire file, or transferring only those portions of the file necessary for the immediate task
- **Computation Migration** – transfer the computation, rather than the data, across the system
  - Via remote procedure calls (RPCs)
  - or via messaging system

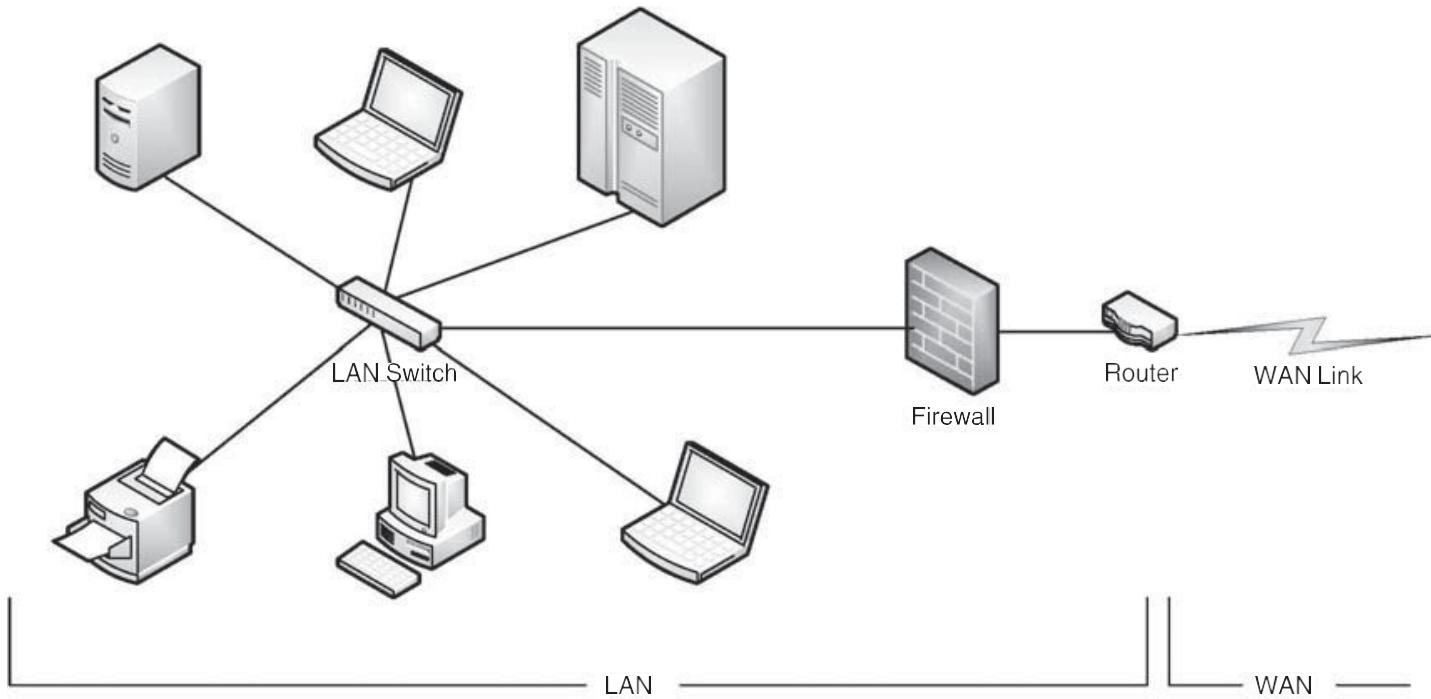
# Distributed-Operating Systems

- **Process Migration** – execute an entire process, or parts of it, at different sites
  - **Load balancing** – distribute processes across network to even the workload
  - **Computation speedup** – subprocesses can run concurrently on different sites
  - **Hardware preference** – process execution may require specialized processor
  - **Software preference** – required software may be available at only a particular site
  - **Data access** – run process remotely, rather than transfer all data locally
- Consider the World Wide Web

# Network Structure

- **Local-Area Network (LAN)** – designed to cover small geographical area
  - Multiple topologies like star or ring
  - Speeds from 1Mb per second (Appletalk, bluetooth) to 40 Gbps for fastest Ethernet over twisted pair copper or optical fibre
  - Consists of multiple computers (mainframes through mobile devices), peripherals (printers, storage arrays), routers (specialized network communication processors) providing access to other networks
  - Ethernet most common way to construct LANs
    - Multiaccess bus-based
    - Defined by standard IEEE 802.3
  - Wireless spectrum (**WiFi**) increasingly used for networking
    - I.e. IEEE 802.11g standard implemented at 54 Mbps

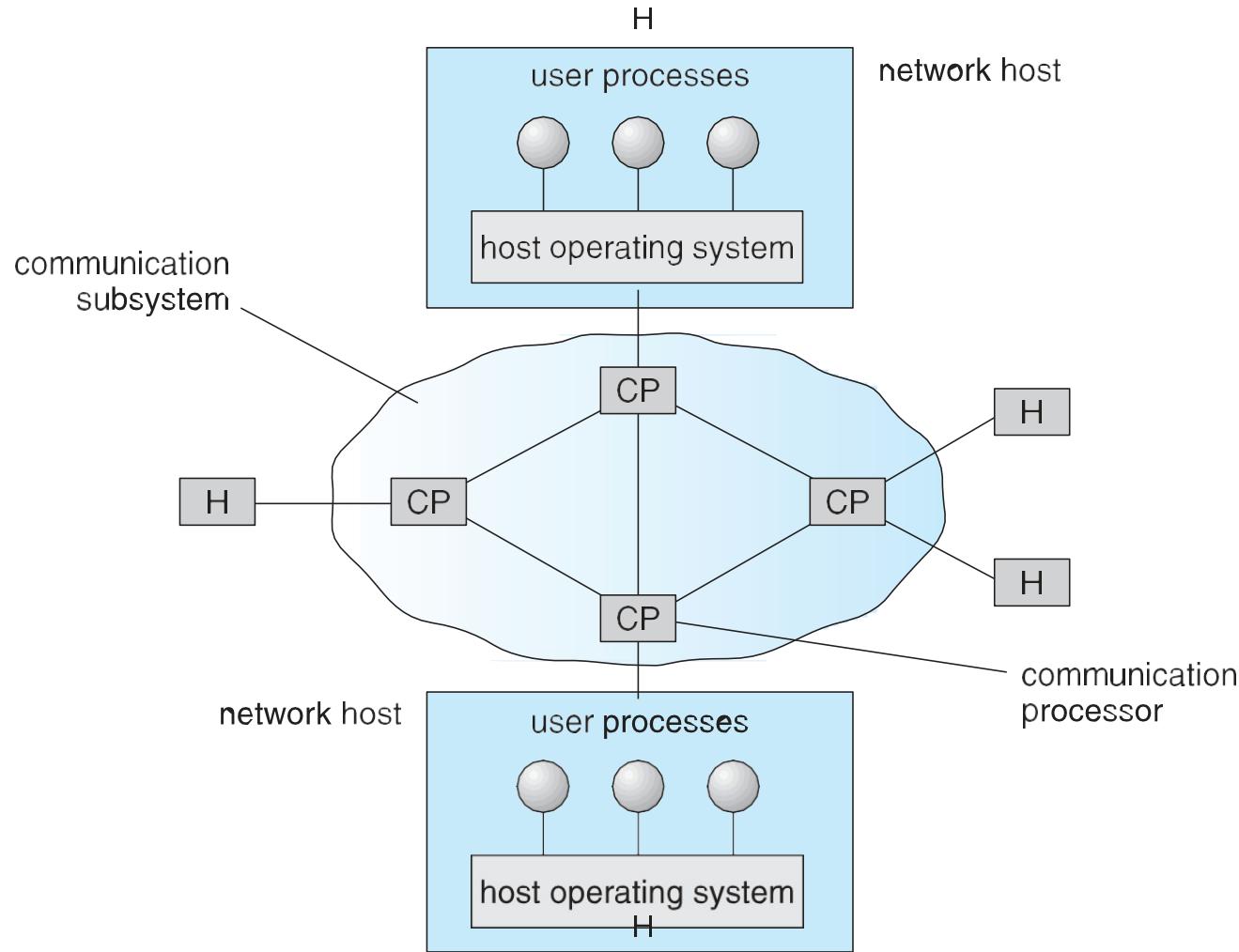
# Local-area Network



# Network Types

- **Wide-Area Network (WAN)** – links geographically separated sites
  - Point-to-point connections over long-haul lines (often leased from a phone company)
    - Implemented via **connection processors** known as **routers**
  - Internet WAN enables hosts world wide to communicate
    - Hosts differ in all dimensions but WAN allows communications
  - Speeds
    - T1 link is 1.544 Megabits per second
    - T3 is  $28 \times T1s = 45$  Mbps
    - OC-12 is 622 Mbps
  - WANs and LANs interconnect, similar to cell phone network:
    - Cell phones use radio waves to cell towers
    - Towers connect to other towers and hubs

# Communication Processors in a Wide-Area Network



# Communication Structure

The design of a communication network must address four basic issues:

- **Naming and name resolution** - How do two processes locate each other to communicate?
- **Routing strategies** - How are messages sent through the network?
- **Connection strategies** - How do two processes send a sequence of messages?
- **Contention** - The network is a shared resource, so how do we resolve conflicting demands for its use?

# Naming and Name Resolution

- Name systems in the network
- Address messages with the process-id
- Identify processes on remote systems by **<host-name, identifier>** pair
- **Domain name system (DNS)** – specifies the naming structure of the hosts, as well as name to address **resolution** (Internet)

# Routing Strategies

- **Fixed routing** - A path from A to B is specified in advance; path changes only if a hardware failure disables it
  - Since the shortest path is usually chosen, communication costs are minimized
  - Fixed routing cannot adapt to load changes
  - Ensures that messages will be delivered in the order in which they were sent
- **Virtual routing**- A path from A to B is fixed for the duration of one session. Different sessions involving messages from A to B may have different paths
  - Partial remedy to adapting to load changes
  - Ensures that messages will be delivered in the order in which they were sent

# Routing Strategies

- **Dynamic routing** - The path used to send a message from site A to site B is chosen only when a message is sent
  - Usually a site sends a message to another site on the least used at that particular time
  - Adapts to load changes by avoiding routing messages on heavily used path
  - Messages may arrive out of order
    - This problem can be remedied by appending a sequence number to each message
  - Most complex to set up
- Tradeoffs mean all methods are used
  - UNIX provides ability to mix fixed and dynamic
  - Hosts may have fixed routes and **gateways** connecting networks together may have dynamic routes

# Routing Strategies

- Router is communications processor responsible for routing messages
  - Must have at least 2 network connections
  - May be special purpose or just function running on host.
  - Checks its tables to determine where destination host is, where to send messages.
- Static routing – table only changed manually
- Dynamic routing – table changed via **routing protocol**

# Routing Strategies

- More recently, routing is managed by **intelligent software** more intelligently than routing protocols
  - **OpenFlow** is device-independent, allowing developers to introduce network efficiencies by decoupling data-routing decisions from underlying network devices
- Messages vary in length – simplified design breaks them into **packets** (or **frames**, or **datagrams**)
- **Connectionless message** is just one packet
  - Otherwise need a connection to get a multi- packet message from source to destination

# Connection Strategies

- **Circuit switching** – A permanent physical link is established for the duration of the communication (i.e., telephone system)
- **Message switching** - A temporary link is established for the duration of one message transfer (i.e., post-office mailing system)
- **Packet switching** - Messages of *variable* length are divided into fixed-length packets which are sent to the destination
  - Each packet may take a different path through the network
  - The packets must be reassembled into messages as they arrive
- Circuit switching requires *setup time*, but incurs less overhead for shipping each message, and may waste network bandwidth
- Message and packet switching require less setup time, but incur more overhead per message

# Communication Protocol

The communication network is partitioned into the following multiple layers:

- **Layer 1: Physical layer** – handles the mechanical and electrical details of the physical transmission of a bit stream
- **Layer 2: Data-link layer** – handles the *frames*, or fixed-length parts of packets, including any error detection and recovery that occurred in the physical layer
- **Layer 3: Network layer** – provides connections and routes packets in the communication network, including handling the address of outgoing packets, decoding the address of incoming packets, and maintaining routing information for proper response to changing load levels

# Communication Protocol

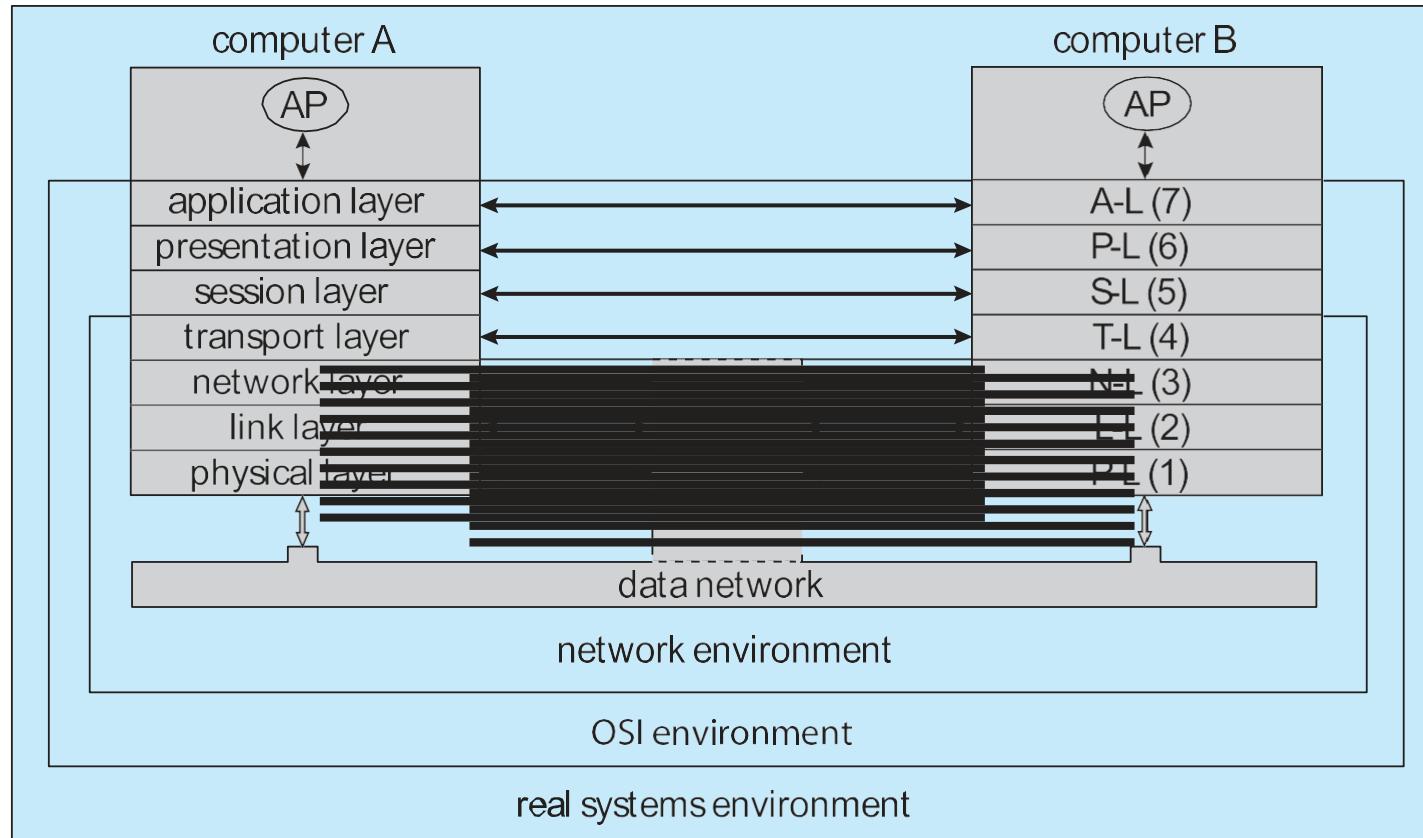
**Layer 4: Transport layer** – responsible for low-level network access and for message transfer between clients, including partitioning messages into packets, maintaining packet order, controlling flow, and generating physical addresses

**Layer 5: Session layer** – implements sessions, or process-to-process communications protocols

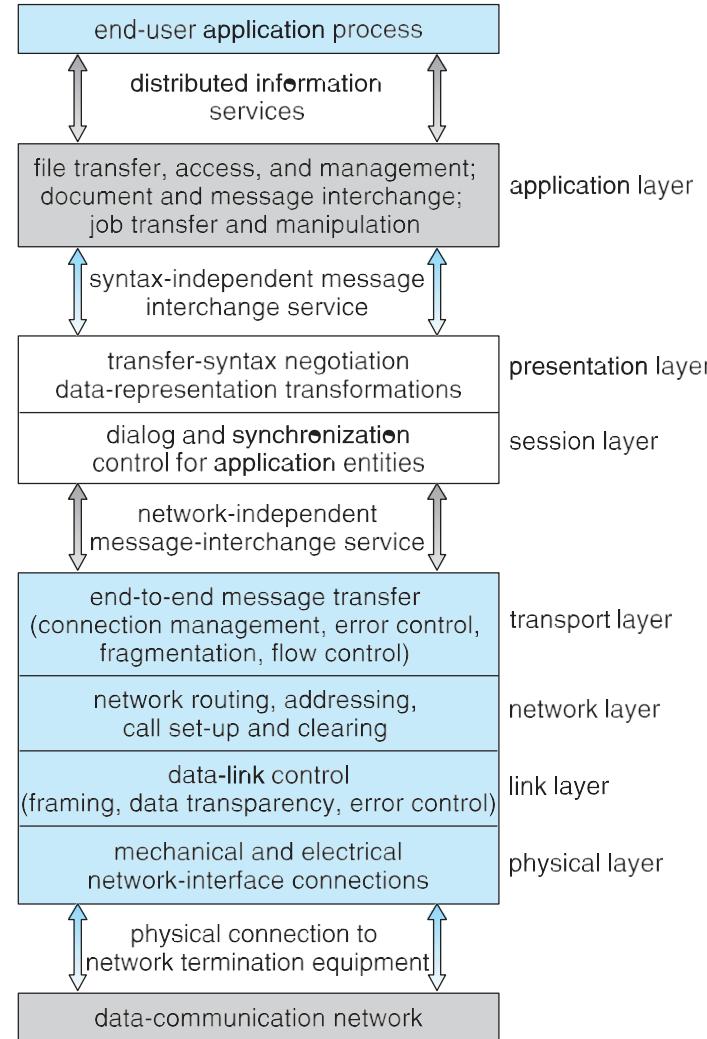
**Layer 6: Presentation layer** – resolves the differences in formats among the various sites in the network, including character conversions, and half duplex/full duplex (echoing)

**Layer 7: Application layer** – interacts directly with the users, deals with file transfer, remote-login protocols and electronic mail, as well as schemas for distributed databases

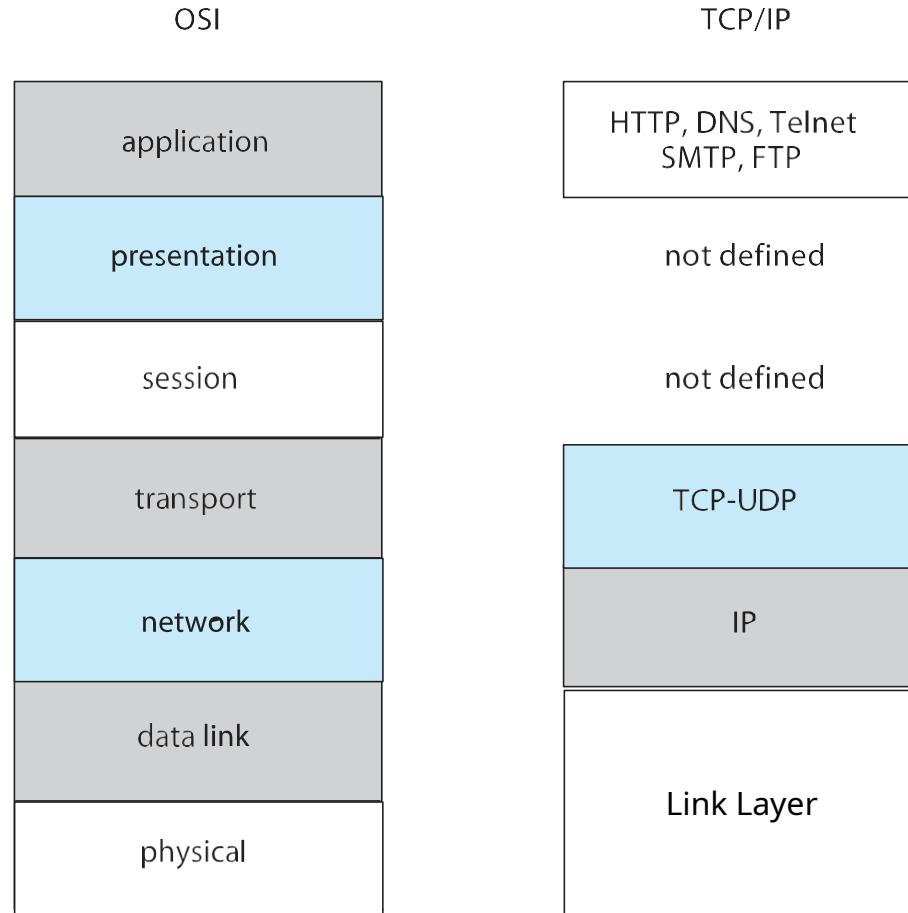
# Communication Via ISO Network Model



# The ISO Protocol Layer



# The TCP/IP Protocol Layers



# Example: TCP/IP

- The transmission of a network packet between hosts on an Ethernet network
- Every host has a unique IP address and a corresponding Ethernet **Media Access Control (MAC)** address
- Communication requires both addresses
- **Domain Name Service (DNS)** can be used to acquire IP addresses
- **Address Resolution Protocol (ARP)** is used to map MAC addresses to IP addresses
  - **Broadcast** to all other systems on the Ethernet network
- If the hosts are on the same network, ARP can be used
  - If the hosts are on different networks, the sending host will send the packet to a router which routes the packet to the destination network

# Robustness

- Failure detection
- Reconfiguration

# Failure Detection

- Detecting hardware failure is difficult
- To detect a link failure, a **heartbeat** protocol can be used
- Assume Site A and Site B have established a link
  - At fixed intervals, each site will exchange an *I-am-up* message indicating that they are up and running
- If Site A does not receive a message within the fixed interval, it assumes either (a) the other site is not up or (b) the message was lost
- Site A can now send an *Are-you-up?* message to Site B
- If Site A does not receive a reply, it can repeat the message or try an alternate route to Site B

# Failure Detection (Cont.)

- If Site A does not ultimately receive a reply from Site B, it concludes some type of failure has occurred
- Types of failures:
  - Site B is down
  - The direct link between A and B is down
  - The alternate link from A to B is down
  - The message has been lost
- However, Site A cannot determine exactly **why** the failure has occurred

# Reconfiguration

- When Site A determines a failure has occurred, it must reconfigure the system:
  1. If the link from A to B has failed, this must be broadcast to every site in the system
  2. If a site has failed, every other site must also be notified indicating that the services offered by the failed site are no longer available
- When the link or the site becomes available again, this information must again be broadcast to all other sites

# Design Issues

- **Transparency** – the distributed system should appear as a conventional, centralized system to the user
- **Fault tolerance** – the distributed system should continue to function in the face of failure
- **Scalability** – as demands increase, the system should easily accept the addition of new resources to accommodate the increased demand
  - Consider **Hadoop** open source programming framework for processing large datasets in distributed environments (based on **Google search indexing**)
- **Clusters** – a collection of semi-autonomous machines that acts as a single system

# Distributed File System

- **Distributed file system (DFS)** – a distributed implementation of the classical time-sharing model of a file system, where multiple users share files and storage resources
- A DFS manages set of dispersed storage devices
- Overall storage space managed by a DFS is composed of different, remotely located, smaller storage spaces
- There is usually a correspondence between constituent storage spaces and sets of files
- Challenges include:
  - Naming and Transparency
  - Remote File Access

# DFS

## Structure

- **Service** – software entity running on one or more machines and providing a particular type of function to a priori unknown clients
- **Server** – service software running on a single machine
- **Client** – process that can invoke a service using a set of operations that forms its client interface
- A client interface for a file service is formed by a set of primitive file operations (create, delete, read, write)
- Client interface of a DFS should be transparent, i.e., not distinguish between local and remote files
- Sometimes lower level **intermachine** interface need for cross-machine interaction

# Naming and Transparency

- **Naming** – mapping between logical and physical objects
- **Multilevel mapping** – abstraction of a file that hides the details of how and where on the disk the file is actually stored
- A **transparent** DFS hides the location where in the network the file is stored
- For a file being **replicated** in several sites, the mapping returns a set of the locations of this file's replicas; both the existence of multiple copies and their location are hidden

# Naming Structures

- **Location transparency** – file name does not reveal the file 's physical storage location
- **Location independence** – file name does not need to be changed when the file 's physical storage location changes

# Naming Schemes — Three Main Approaches

- Files named by combination of their **host name and local name**; guarantees a unique system-wide name
- **Attach remote directories to local directories**, giving the appearance of a coherent directory tree; only previously mounted remote directories can be accessed transparently
- Total integration of the component file systems
  - A **single global name structure** spans all the files in the system
  - If a server is unavailable, some arbitrary set of directories on different machines also becomes unavailable
- In practice most DFSs use static, location-transparent mapping for user-level names
  - Some support file migration
  - Hadoop supports file migration but without following POSIX standards

# Remote File Access

- **Remote-service mechanism** is one transfer approach
- Reduce network traffic by retaining recently accessed disk blocks in a cache, so that repeated accesses to the same information can be handled **locally**
  - If needed data not already cached, a copy of data is brought from the server to the user
  - Accesses are performed on the cached copy
  - Files identified with one master copy residing at the server machine, but copies of (parts of) the file are scattered in different caches
  - **Cache-consistency problem** – keeping the cached copies consistent with the master file
    - Could be called **network virtual memory**

# Thank You

# Real time System: CPU scheduling

Real-Time Systems: Real time system characteristics and kernel features, Implementing real-time operating system, real-time CPU scheduling.

# Overview

- System Characteristics
- Features of Real-Time Systems
- Implementing Real-Time Operating Systems
- Real-Time CPU Scheduling

# Introduction

- A real-time system requires that results be produced within a specified deadline period
- An embedded system is a computing device that is part of a larger system (I.e. automobile, airliner)
- A safety-critical system is a real-time system with catastrophic results in case of failure

# Hard Vs Soft RTS

- A hard real-time system guarantees that real-time tasks be completed within their required deadlines.
- A soft real-time system provides priority of real-time tasks over non real time tasks"
- Periodic and aperiodic tasks."
- Starting deadline/ completion deadline."

# RTS Characteristic

- Single purpose
- Small size
- Inexpensively mass-produced
- Specific timing requirements

# RTS Characteristic

- Many real-time systems are designed using **system-on-a-chip (SOC)** strategy.
- SOC allows the CPU, memory, memory-management unit, and attached peripheral ports (i.e., USB) to be contained in a single integrated circuit.

# Real Time Kernels

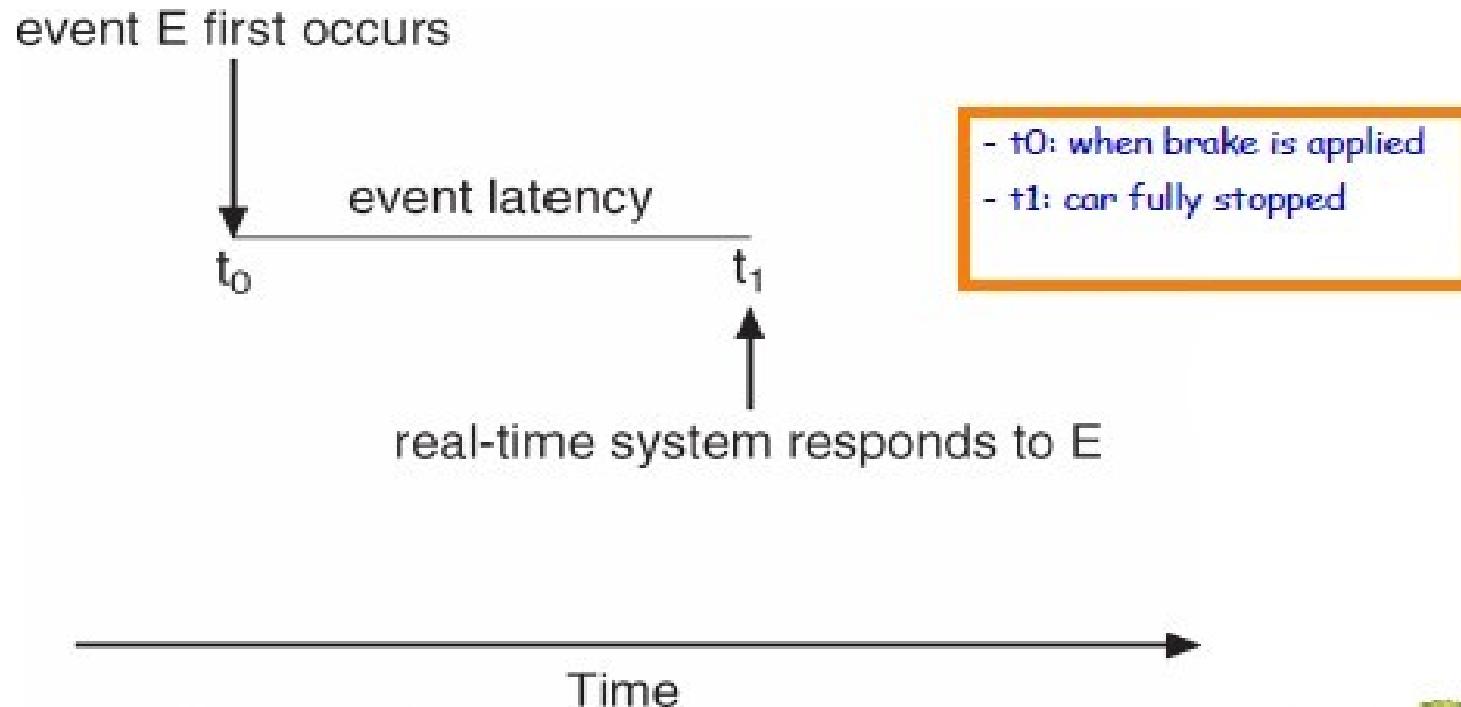
- Most real-time systems do not provide the features found in a standard desktop system
- Because:
  - Real-time systems are typically single-purpose
  - Real-time systems often do not require interfacing with a user

# Implementing Real-Time Systems

- In general, real-time operating systems must provide:
  - Preemptive, priority-based scheduling.
  - Preemptive kernels.
  - Latency must be minimized

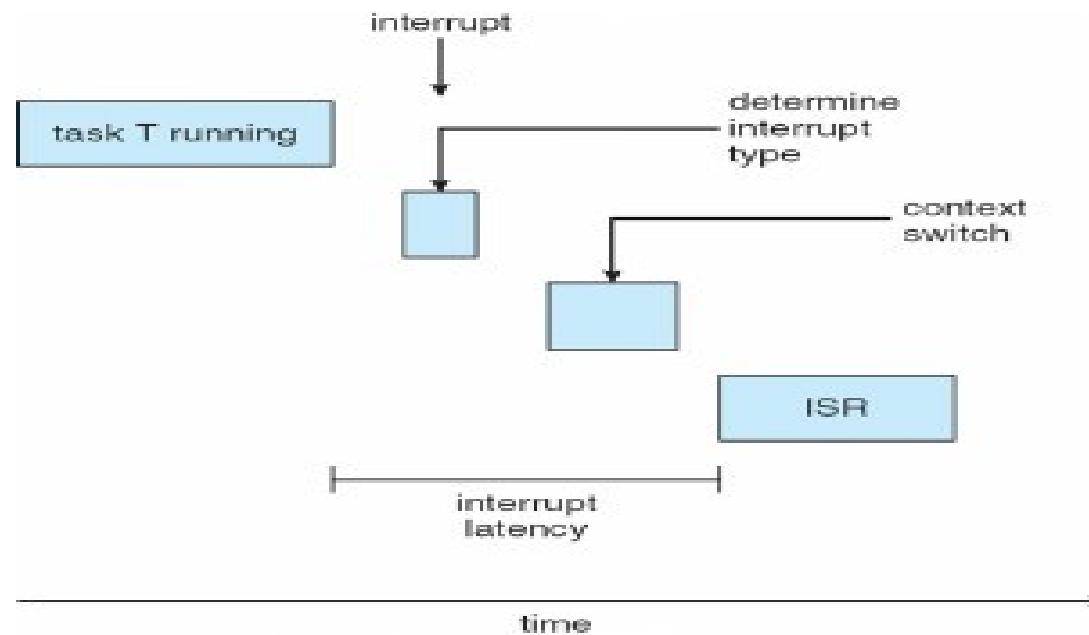
# Minimizing Latency

- **Event latency** is the amount of time from when an event occurs to when it is serviced



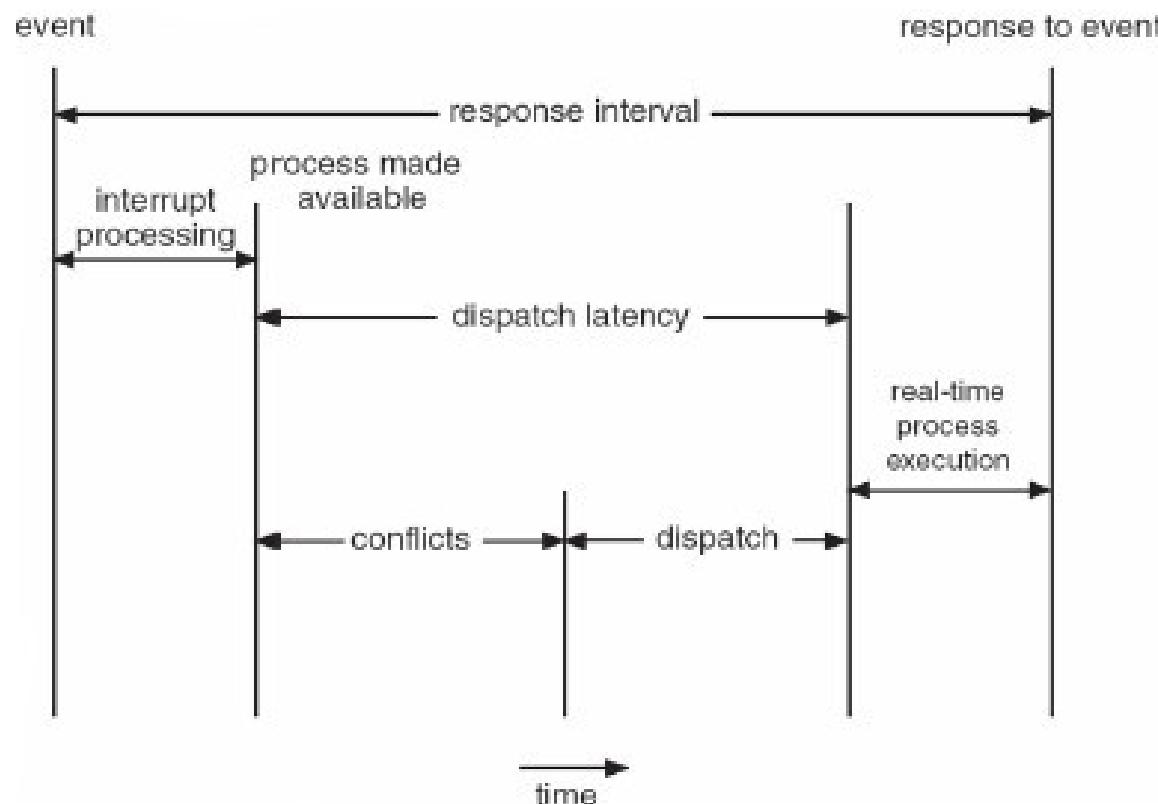
# Interrupt Latency

- Interrupt latency is the period of time from when an interrupt arrives at the CPU to when it is serviced



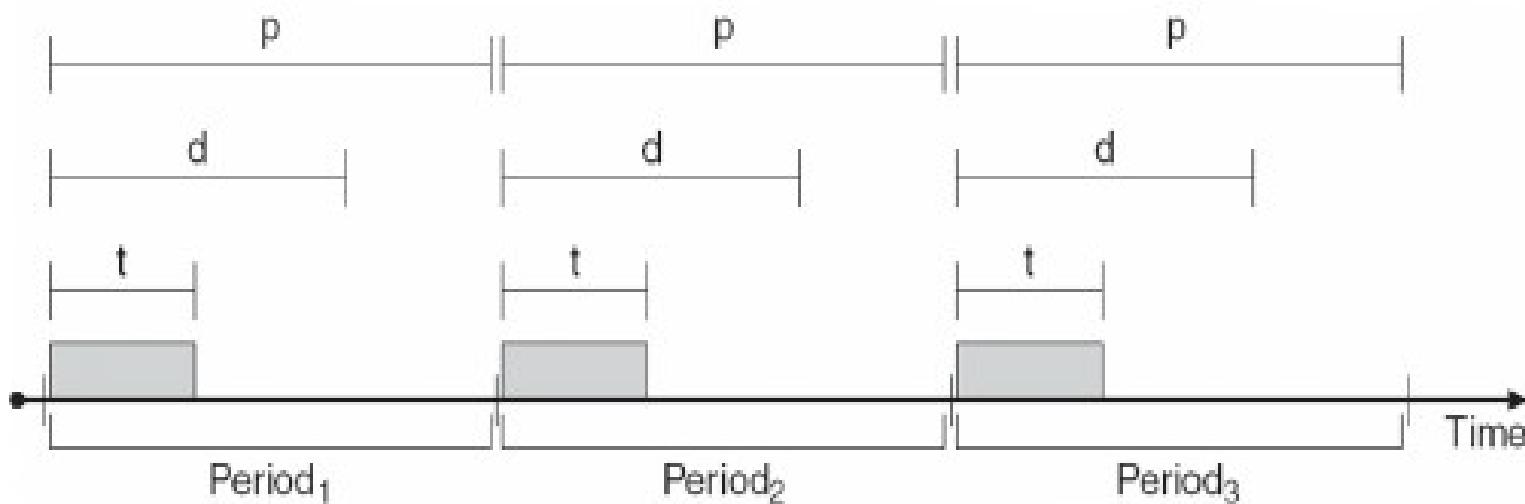
# Dispatch Latency

- Dispatch latency is the amount of time required for the scheduler to stop one process and start another"



# RTS scheduling

- How to schedule the Tasks such that given timing constraints are satisfied?



d may be the same as p.

# Task models

- Non periodic/Aperiodic (three parameters)
  - A: arriving time
  - C: computing time
  - D: deadline (relative deadline)
- Timing constraints: deadline for each task,
  - Relative to arriving time or absolute deadline

# Scheduling Problem

- Given a set of tasks (ready queue)
  - Check if the set is schedulable
  - If yes, construct a schedule to meet all deadlines
  - If no, construct an optimal schedule
    - e.g. minimizing response times

# Tasks with the same arrival time

- Assume a list of tasks
  - $(A, C_1, D_1)(A, C_2, D_2) \dots (A, C_n, D_n)$  that arrive at the same time i.e. A
- How to find a feasible schedule?
  - (there may be many feasible schedules)

# Earliest Due Date first (EDD) [Jackson 1955]

- EDD: order tasks with non-decreasing deadlines.
  - Simple form of EDF (earliest deadline first)
  - Example: (1,10)(2,3)(3,5)
  - Schedule: (2,3)(3,5)(1,10)
- FACT: EDD is optimal
  - If EDF can't find a feasible schedule for a task set, then no other algorithm can, i.e. The task set is non schedulable.

# Tasks with different arrival times

- Assume a list of tasks
  - $S = (A_1, C_1, D_1)(A_2, C_2, D_2) \dots (A_n, C_n, D_n)$
- Preemptive EDF [Horn 1974]:
  - Whenever new tasks arrive, sort the ready queue according to earliest deadlines first at the moment
  - Run the first task of the queue if it is non empty

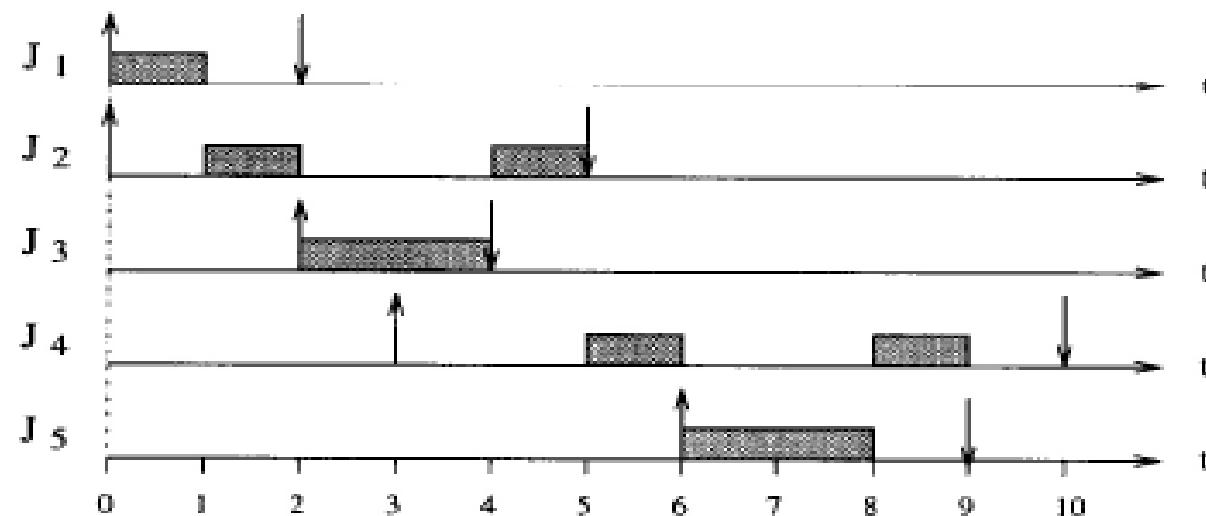
# Earliest Deadline First Scheduling

- Priorities are assigned according to deadlines:
  - the earlier the deadline, the higher the priority;
  - the later the deadline, the lower the priority

# EDF: Example

► **Example:**

	$J_1$	$J_2$	$J_3$	$J_4$	$J_5$
$a_i$	0	0	2	3	6
$C_i$	1	2	2	2	2
$d_i$	2	5	4	10	9



# Least slack scheduling

- similar to shortest remaining time scheduling with the concept of a deadline thrown in.
- pick the process that we can least afford to delay.
- Least slack is computed as *the time to the deadline minus the amount of computation*.
- For example, suppose that our remaining computation time,  $C$ , is 5 msec. and the deadline,  $D$ , is 20 msec from now. The slack time is  $D - C$ , or 15 msec.
- The scheduler will compare this slack time with the slack times of other processes in the system and run the one with the lowest slack time.

# Least Slack Time scheduling

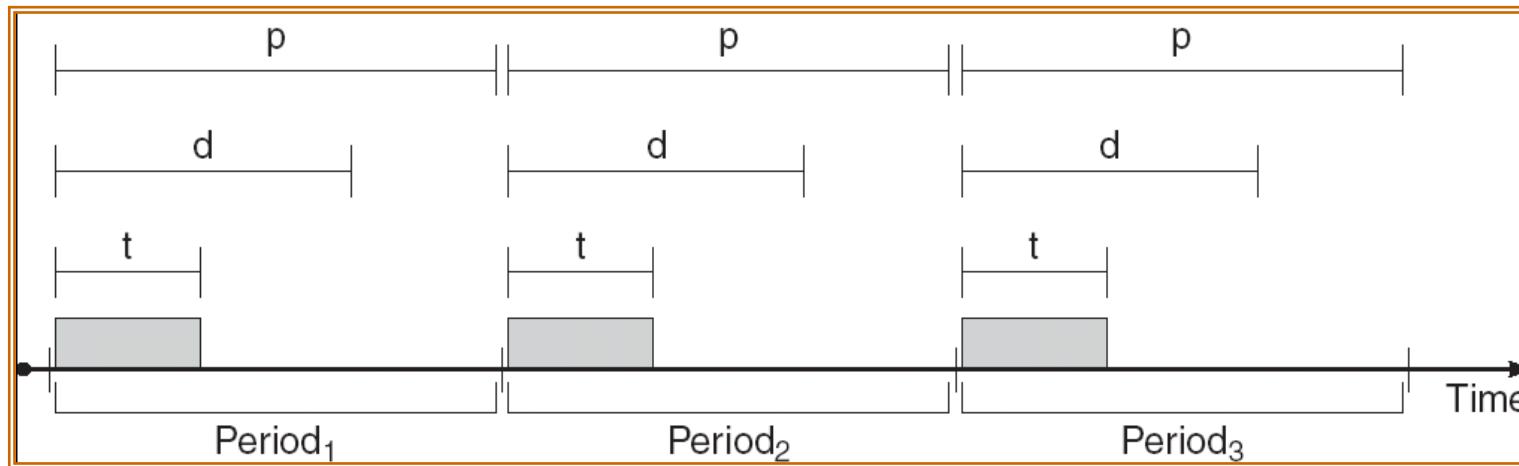
- With earliest deadline, we will always work on the process with the nearest deadline, delaying all the other processes.
- With least slack scheduling, we get a balanced result in that we attempt to keep the differences from deadlines balanced among processes.

# Processor Scheduling for Real-Time *Rate Monotonic Scheduling* (RMS)

- Assume  $m$  periodic processes
  - Process  $i$  requires  $t_i$  msec of processing time every  $p_i$  msec.
  - Equal processing every interval — like clockwork!

# Example

- Periodic process  $i$  requires the CPU at specified intervals (periods)
- $p_i$  is the duration of the period
- $t_i$  is the processing time
- $d_i$  is the deadline by when the process must be serviced
  - Often same as end of period



# Processor Scheduling for Real-Time

## Rate Monotonic Scheduling (RMS)

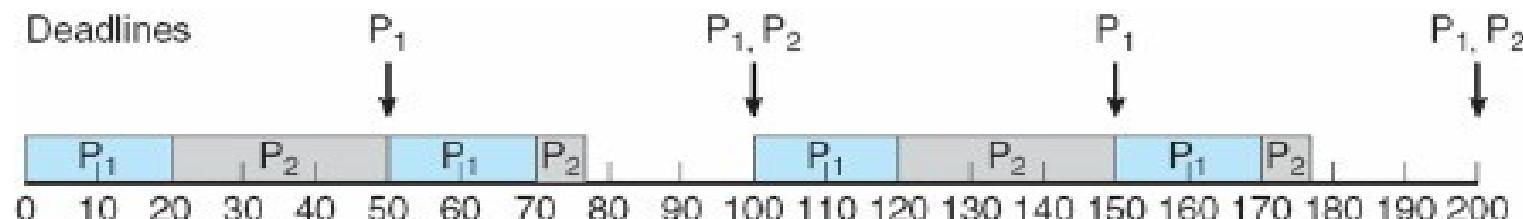
- Assume  $m$  periodic processes
  - Process  $i$  requires  $t_i$  msec of processing time every  $p_i$  msec.
  - Equal processing every interval — like clockwork!
- Assume  $\frac{t_i}{p_i} \leq 1$
- Assign priority of process  $i$  to be  $\frac{1}{p_i}$ 
  - Statically assigned
- Let priority of non-real-time processes be 0

# Rate Monotonic Scheduling (continued)

- Scheduler simply runs highest priority process that is ready
  - May pre-empt other real-time processes
  - Real-time processes become ready in time for each frame or sound interval
  - Non-real-time processes run only when no real-time process needs CPU

# Rate Monotonic Scheduling

- priority is assigned based on the inverse of its period (its rate)"
  - Shorter periods = higher priority;"
  - Longer periods = lower priority"
- P1 is assigned a higher priority than P2.



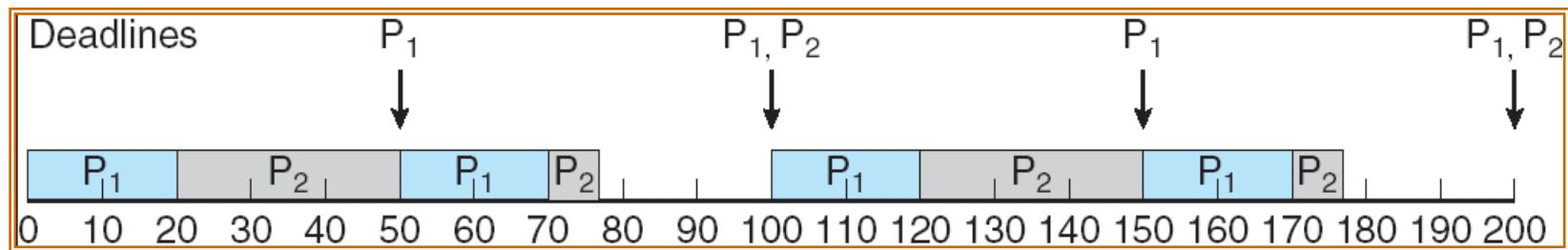
Periodic tasks with completion deadline

Rate: 1/period, the higher the rate, the higher the priority (the smaller P is)

Schedulability

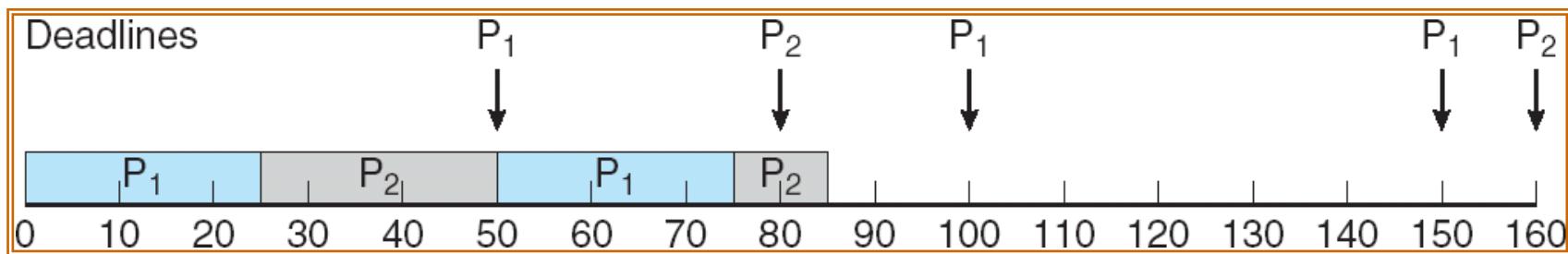
# Example

- $p_1 = 50 \text{ msec}$ ;  $t_1 = 20 \text{ msec}$
- $p_2 = 100 \text{ msec}$ ;  $t_2 = 35 \text{ msec}$
- Priority( $p_1$ ) > Priority( $p_2$ )
- Total compute load is 75 msec per every 100 msec.
- Both tasks complete within every period
  - 25 msec per 100 msec to spare



# Example 2

- $p_1 = 50 \text{ msec}$ ;  $t_1 = 25 \text{ msec}$
- $p_2 = 80 \text{ msec}$ ;  $t_2 = 35 \text{ msec}$
- Priority( $p_1$ ) > Priority( $p_2$ )
- Total compute load is  $(25/50)+(35/80) = 0.94 = \sim 94\%$  of CPU.
- Cannot complete both tasks within some periods
  - Even though there is still CPU capacity to spare!



# Processor Scheduling for Real-Time Earliest Deadline First (EDF)

- When each process  $i$  become ready, it announces deadline  $D_i$  for its next task.
- Scheduler always assigns processor to process with earliest deadline.
  - May pre-empt other real-time processes

# Earliest Deadline First Scheduling

## (continued)

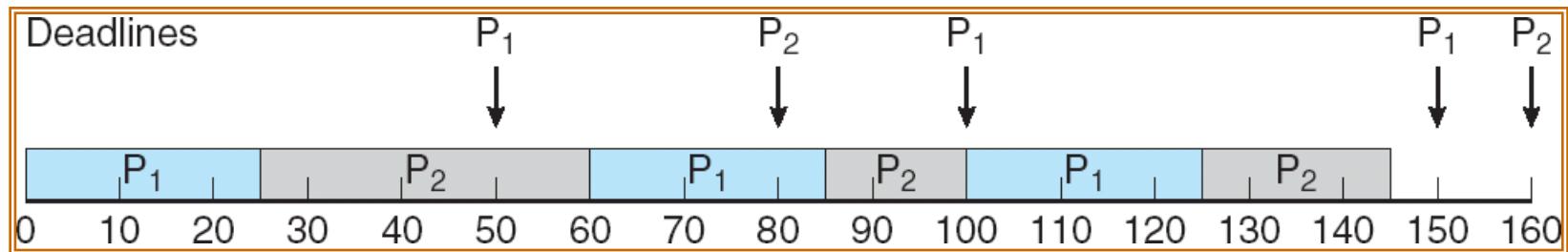
- No assumption of periodicity
- No assumption of uniform processing times
- **Theorem:** If *any* scheduling policy can satisfy QoS requirement for a sequence of real time tasks, then EDF can also satisfy it.
  - *Proof:* If  $i$  scheduled before  $i+1$ , but  $D_{i+1} < D_i$ , then  $i$  and  $i+1$  can be interchanged without affecting QoS guarantee to either one.

# Earliest Deadline First Scheduling

- EDF is more complex scheduling algorithm
  - Priorities are dynamically calculated
  - Processes must know deadlines for tasks
- EDF can make higher use of processor than RMS
  - Up to 100%
- There is a large body of knowledge and theorems about EDF analysis

# Example 2 (again)

- Priorities are assigned according to deadlines:
  - the earlier the deadline, the higher the priority;
  - the later the deadline, the lower the priority.



# Thank You

# Multimedia Systems

Multimedia Systems: Multimedia system overview, multimedia kernels, compression, CPU scheduling, disk scheduling.

# Outline

- What is multi-media?
- Requirements and challenges for audio and video in computer systems
  - CD devices and formats
- Systems for multimedia
- Compression and bandwidth
- If time:-
  - Processor and disk scheduling
  - Network streaming and management

# **what do we mean by “multimedia”**

---

- Audio and video within a computer system
  - CD's & DVD's
  - Computer hard drive
  - Interactive
- Live broadcast & web casts
  - Radio stations, Network TV, Webcams, Skype, ...
- Video on demand
  - Pause, fast forward, reverse, etc.
- Interactive activities involving audio, video, images
  - Meetings and presentations with 2-way audio
  - Teleconferencing
- Handheld devices
  - iPod, MP-3 players; personal video; mobile phones, ...

# Multimedia Data and Delivery

- Stored in file system like ordinary data.
- Must be accessed with specific timing requirements.
- E.g., video *must* be displayed at 24-30 frames per second.
  - System must guarantee timely delivery
- *Continuous-media data*
  - Data with specific rate requirements

# Multimedia Characteristics

- Multimedia files can be *very* large.
- Continuous media data may require *very* high data rates.
- Multimedia applications are usually sensitive to timing delays during playback.
- Note: human ear is more sensitive to *jitter* in audio than eye is to *jitter* in video!

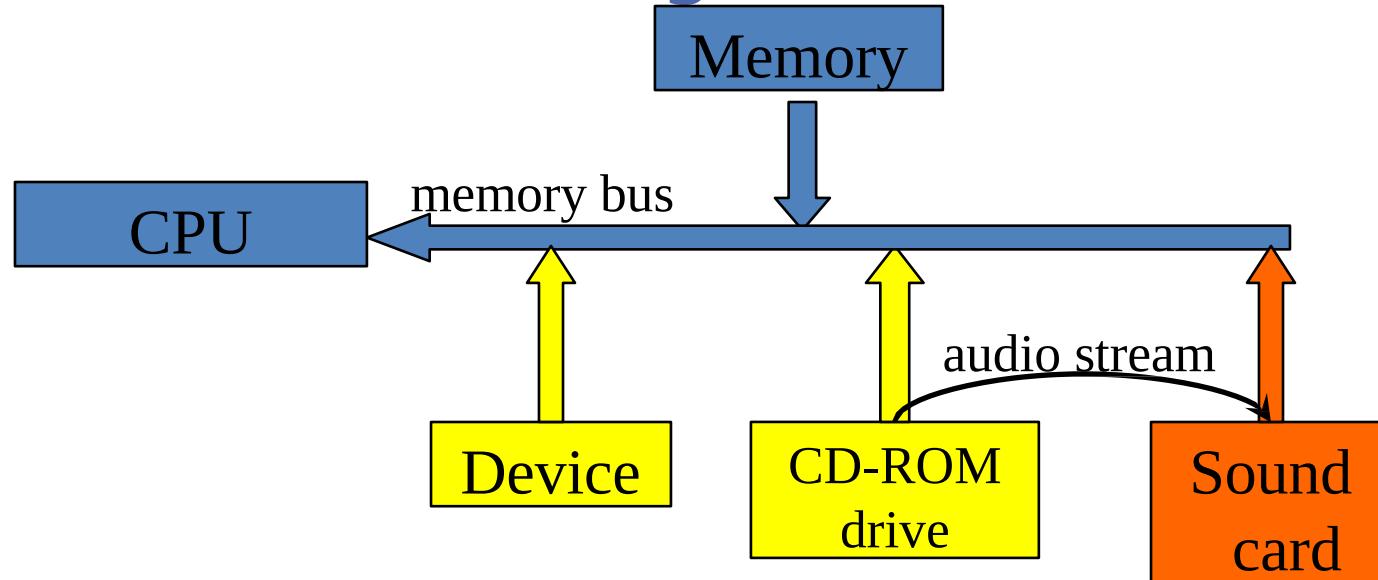
# Requirements

- “Smooth” audio and video
  - Deterioration in quality jerky playback
- Multiple concurrent streams
  - Video & multimedia servers
  - TiVo, etc.
- Wide range of network bandwidths
- Audio/video while PC is doing something else

# Some System Architectures

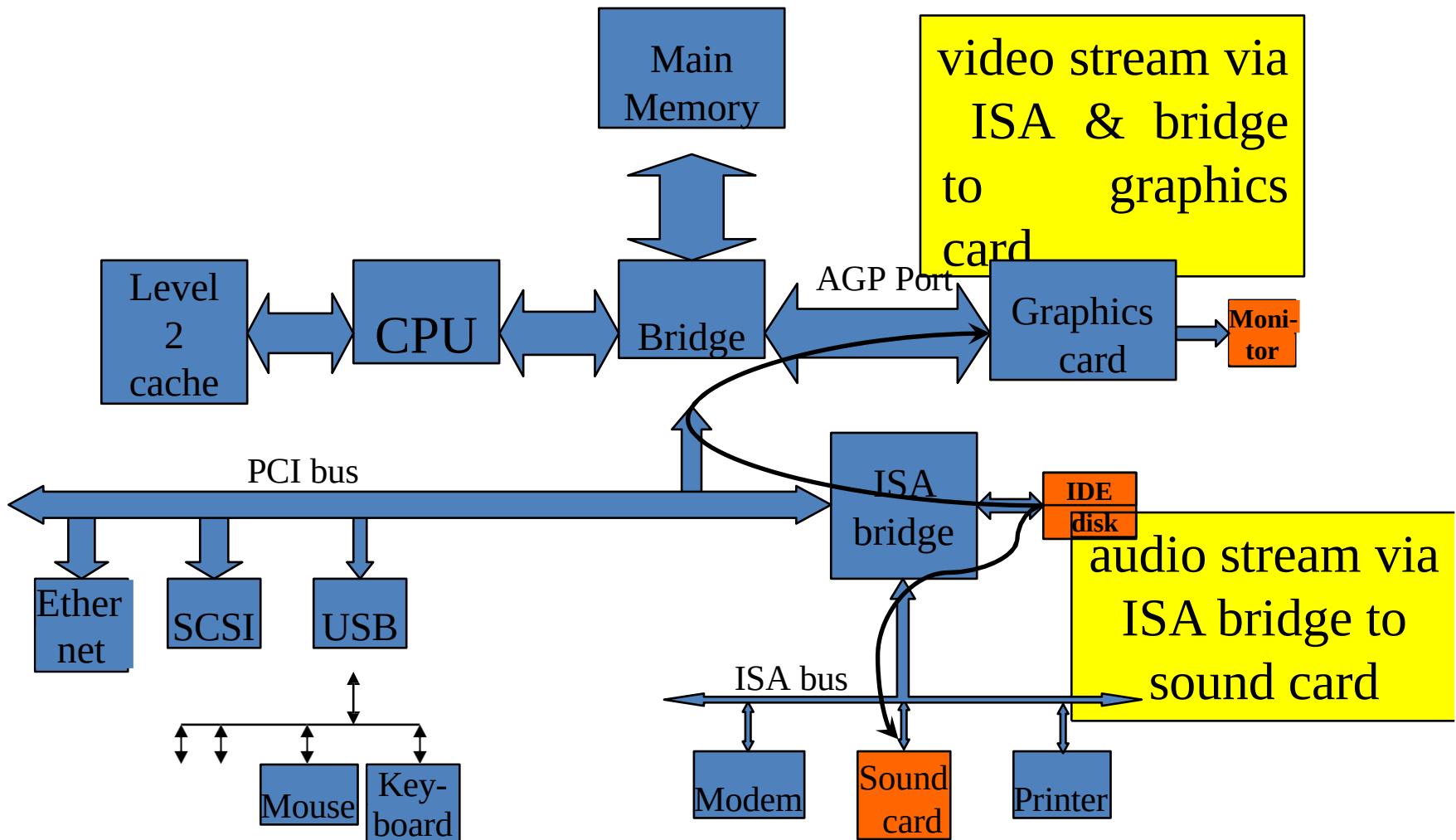
- Simple:
  - Data paths for audio/video that are separate from computational data paths
- Modern
  - Fast system bus, CPU, devices
- Video server
  - Disk farm and multiple streams

# System Organization (a decade ago)



- Separate data path for audio stream
  - Or headphone jack and volume control on CD drive itself
- Main system bus and CPU were too busy/slow to handle real-time audio

# System Organization (typical Pentium PC today)



# DVD devices

- Similar in concept, different in details
- More data, higher bandwidth
  - *Track spacing:*    0.834μ       vs.       1.6μ
  - *Bit spacing:*       0.4μ       vs.       0.74μ
  - *Capacity:*              4.7 Gbytes    vs.       650 Mbytes

# Compression

An essential part of audio and video representation as files

# Why Compression? – CD-quality audio

- 22,050 Hz 44,100 samples/sec
  - 16 bits per sample
- Two channels 176,400 bytes/sec  
1.4 mbytes/sec
  - Okay for a modern PC
  - Not okay for 56 kb/sec modem (speed) or iPod (space)!
- MP-3 0.14 mbytes/sec (10:1)
  - *Same* audio quality!
  - Compression ratio varies with type of music

# Why Compression? – Video

- “Standard” TV frame = 640 480 pixels @ 25-30 frames/sec (fps)  
9,216,000 pixels/sec = 27,648,000 bytes/sec
- Hollywood “standard” movie 133 minutes  
approx. 210 gigabytes (standard resolution)!
- HDTV = 1280 720 pixels @ 30 fps  
82,944,000 bytes/sec (and rising!)
- DVD holds ~ 4.7 gigabytes  
average of ~ 620 kilobytes/sec!
- “Standard” movie of 133 minutes requires serious compression just to *fit* onto DVD

# Video Compression Requirements

- Compression ratio > 50:1
  - i.e., 210 gigabytes:4.7 gigabytes
- Visually indistinguishable from original
  - Even when paused
- Fast, cheap decoder
  - Slow encoder is okay
- VCR controls
  - Pause, fast forward, reverse

# Video Compression Standards

- MPEG (*Motion Picture Experts Group*)
  - Based on JPEG (*Joint Photographic Experts Group*)
  - Multi-layer
    - Layer 1 = system and timing information
    - Layer 2 = video stream
    - Layer 3 = audio and text streams
- Three standards
  - MPEG-1 – 352 240 frames; < 1.5 mb/sec ( < VHS quality)
    - Layer 3 = MP3 Audio standard
    - Typical uses:- video clips on internet; video for handhelds
  - MPEG-2 – standard TV & HDTV; 1.5-15 mb/sec
    - DVD encoding
  - MPEG-4 – combined audio, video, interactive graphics
    - 2D & 3D animations

# JPEG compression (single frame)

## 1. Convert RGB into YIQ

- $Y$  = *luminance* (i.e., brightness) ~ black-white TV
- $I, Q$  = *chrominance* (similar to *saturation* and *hue*)

Reason: Human eye is more sensitive to luminance than to color (rods vs. cones)

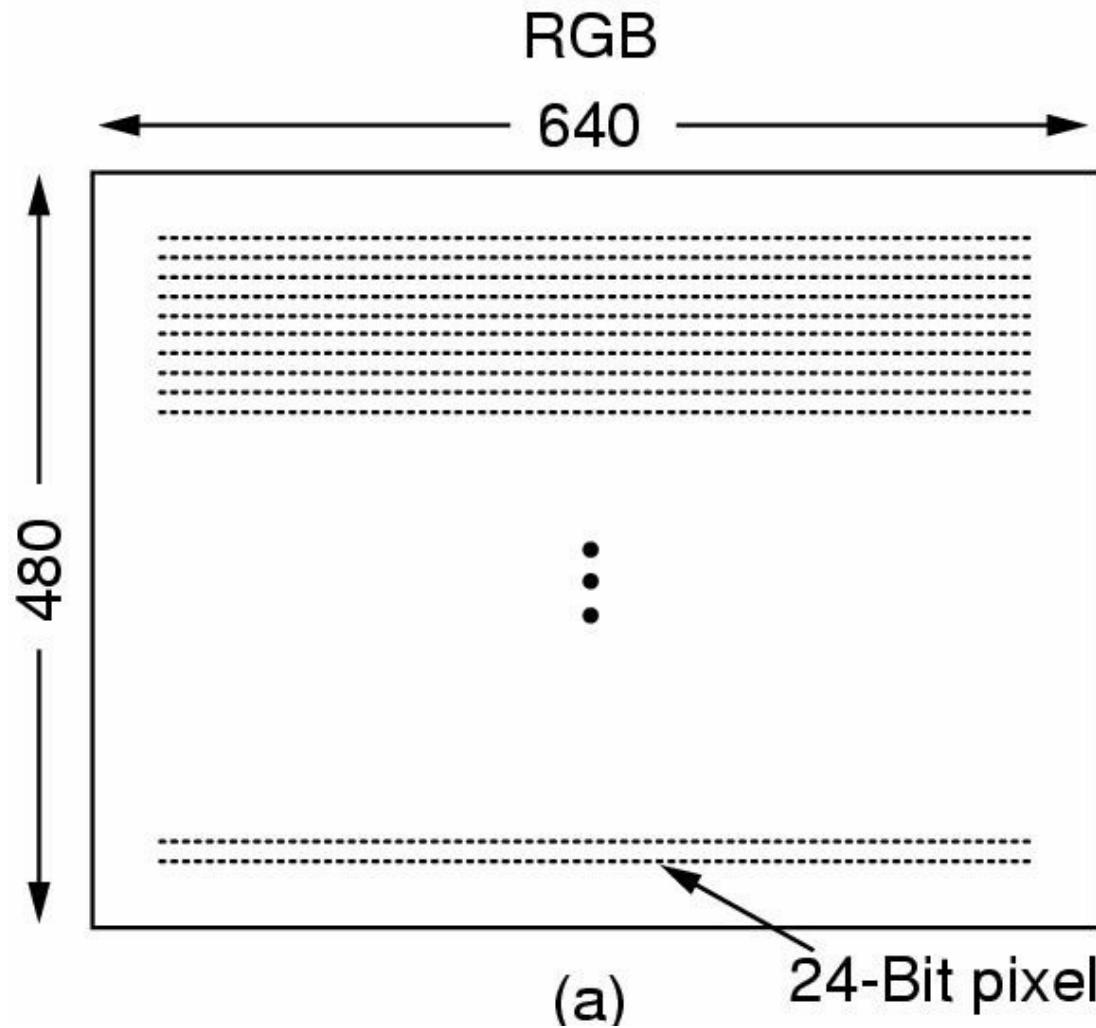
## 2. Down-sample $I, Q$ channels

- i.e., average over 2x2 pixels to reduce resolution
- lossy compression, but barely noticeable to eye

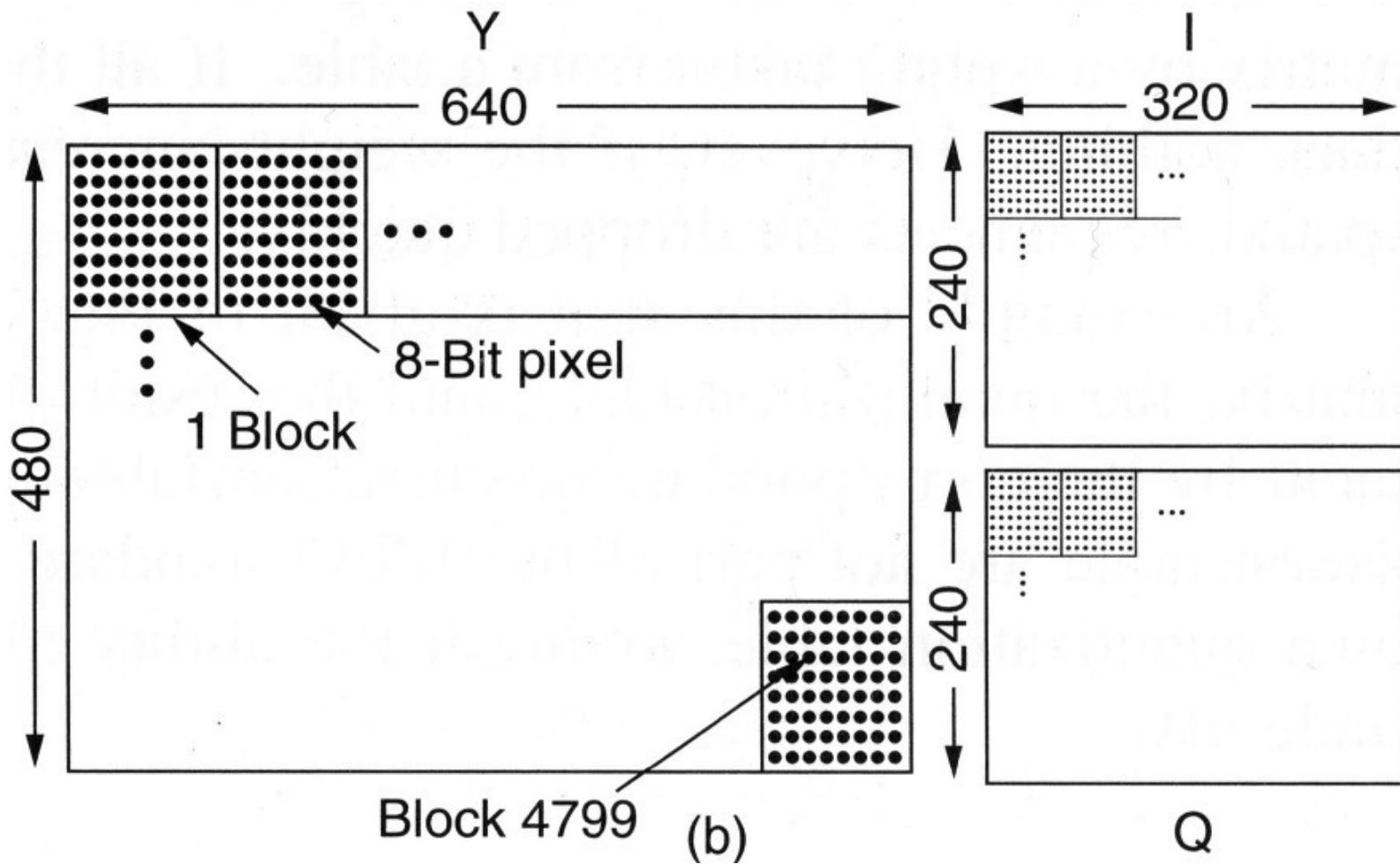
## 3. Partition each channel into 8x8 blocks

- 4800  $Y$  blocks, 1200 each  $I$  &  $Q$  blocks (for 640x480)

# JPEG (continued)



# JPEG (continued)



# JPEG (continued)

4. Calculate *Discrete Cosine Transform* (DCT) of each 8 8 block
5. Divide 8 8 block of DCT values by 8 8 *quantization table*
  - Effectively throwing away higher frequencies
6. Linearize 8 8 block, run-length encode, and apply a Huffman code to reduce to a small fraction of original size (in bytes)

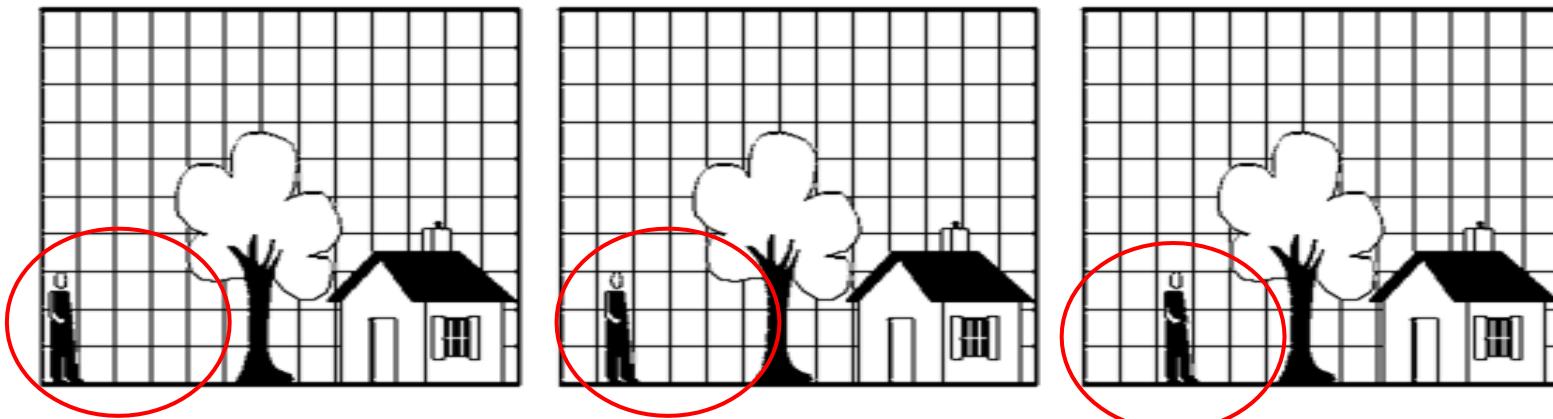
# JPEG (concluded)

7. Store or transmit 8 8 quantization table followed by list of compressed blocks
  - Achieves 20:1 compression with good visual characteristics
    - Higher compression ratios possible with visible degradation
  - JPEG algorithm executed backwards to recover image
    - Visually indistinguishable from original @ 20:1
  - JPEG algorithm is symmetric
    - Same speed forwards and backwards

# MPEG

- JPEG-like encoding of each frame
- Takes advantage of *temporal locality*
  - I.e., each frame usually shares similarities with previous frame
    - encode and transmit only differences
- Sometimes an object moves relative to background
  - find object in previous frame, calculate difference, apply *motion vector*

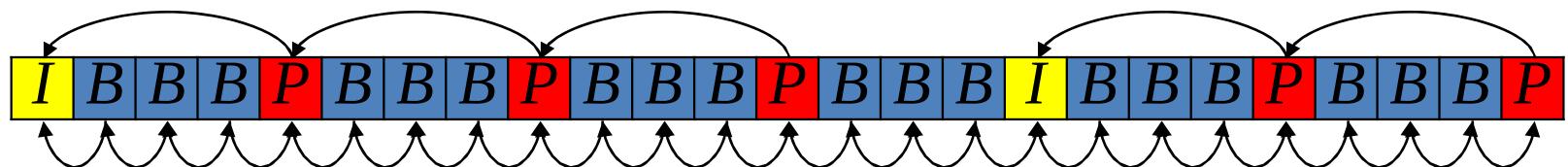
# Temporal Locality (example)



Consecutive Video Frames

# MPEG organization

- Three types of frames
  - *I-frame: Intracoded or Independent.*
    - Full JPEG-encoded frame
    - Occurs at intervals of a second or so
    - Also at start of every scene
  - *P-frame: Predictive frame*
    - Difference from previous frame
  - *B-frame: Bidirectional frame*
    - Like *p-frame* but difference from both *previous* and *next* frame



# MPEG Characteristics

- Non-uniform data rate!
- Compression ratios of 50:1 - 80:1 are readily obtainable
- Asymmetric algorithm
  - Fast decode (like JPEG)
  - Encode requires image search and analysis to get high quality differences
- Cheap decoding chips available for
  - graphics cards
  - DVD players, etc.

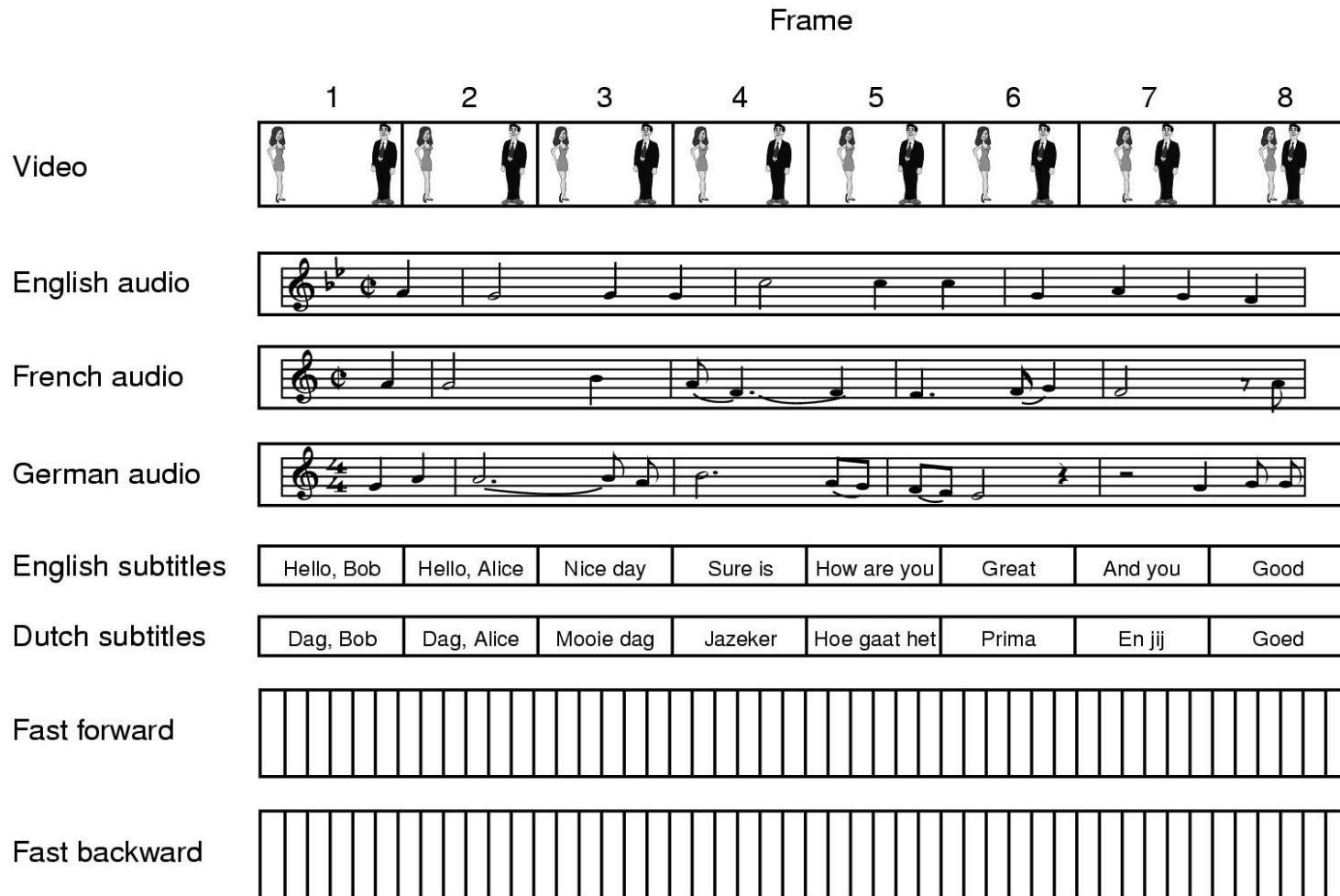
# MPEG Problem – Fast Forward/Reverse

- Cannot simply skip frames
  - Next desired frame might be  $B$  or  $P$  derived from a skipped frame
- Options:
  - Separate fast forward and fast reverse files
    - MPEG encoding of every  $n$ th frame
    - Often used in video-on-demand server
  - Display only  $I$  and  $P$  frames
    - If  $B$  frame is needed, derive from nearest  $I$  or  $P$

# “Movie” File Organization

- One MPEG-2 video stream
  - Possible fast forward, fast reverse sub-streams
- Multiple audio streams
  - Multiple languages
- Multiple text streams
  - Subtitles in multiple languages
- All interleaved
  - Possibly in multiple files!

# Frame



# Operating System Challenge

- How to get the contents of a movie file from disk or DVD drive to video screen and speakers.
  - Fixed frame rate (25 or 30 fps)
  - Steady audio rate
  - Bounded *jitter*
- Such requirements are known as *Quality-of-Service* (QoS) guarantees.
- Classical problem in *real-time scheduling*
  - Obscure niche become mainstream!

# QoS Guarantees

- Building a system to guarantee QoS effects the following:-
  - CPU processing
  - Scheduling and interrupt handling
  - File systems
  - Network protocols

# Requirement of Multimedia Operating Systems

- There are three levels of QoS
  - *Best-effort service*: the system makes a best effort with no attempt to guarantee
  - *Soft QoS*: allows different traffic streams to be prioritized, but no QoS guarantees are made
  - *Hard QoS*: system ensures QoS requirements are always met
    - Prioritization
    - Admission control
    - Bounded latency on interrupt handling, processing, etc.

# Parameters Defining QoS

- *Throughput*: the total amount of work completed during a specific time interval
- *Delay*: the elapsed time from when request is first submitted to desired result
- *Jitter*: delays that occur during playback of a stream.
- *Reliability*: how errors are handled during transmission and processing of continuous media

# Further QoS Issues

- QoS may be *negotiated* between the client and server.
- Operating systems may use an *admission control* algorithm
  - Admits a request for service only if the server has sufficient resources to satisfy the request

# Network Management

---

## (continued)

- Broadcasting
  - Like cable TV
  - Fixed portion of network bandwidth dedicated to set of broadcast streams; inflexible
- Multicasting
  - Typical webcast
  - Multicast tree set up through internet to reach customers
  - Customers can come and go
- Unicast
  - Dedicated connection between server and client
  - Streaming version of familiar transport protocols

# Streaming – Delivery of Multimedia Data over Network

- Two types of streaming:-
  - *Progressive download*: client begins playback of multimedia file during delivery.
    - File is ultimately stored on client computer
    - (Hopefully) download speed > playback speed
  - *Real-time streaming*: multimedia file is delivered to, but *not* stored on, client computer
    - Played back at same speed as delivery
    - Limited amount of buffering to remove jitter

# Two types of Real-time Streaming

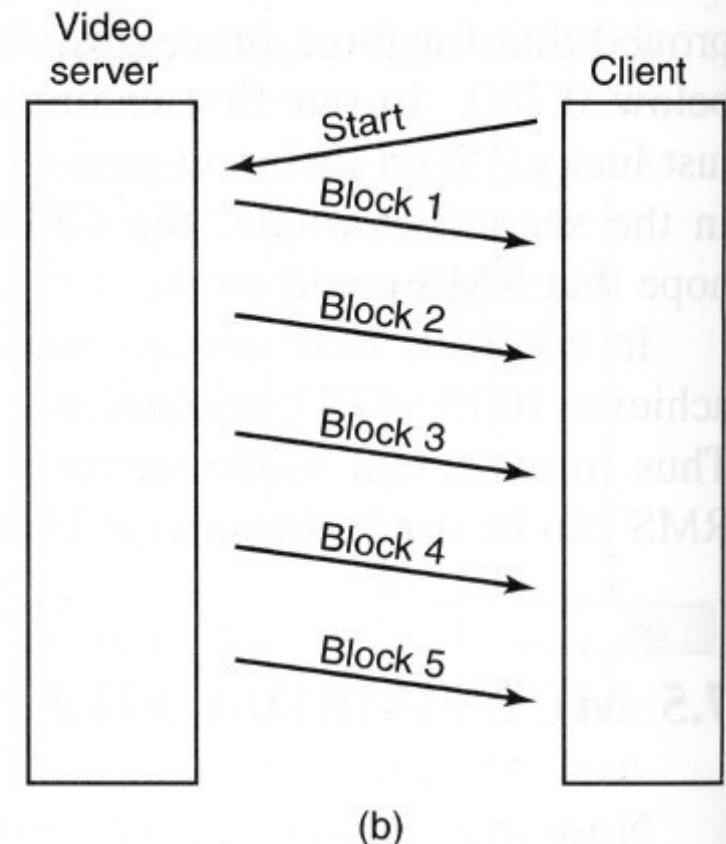
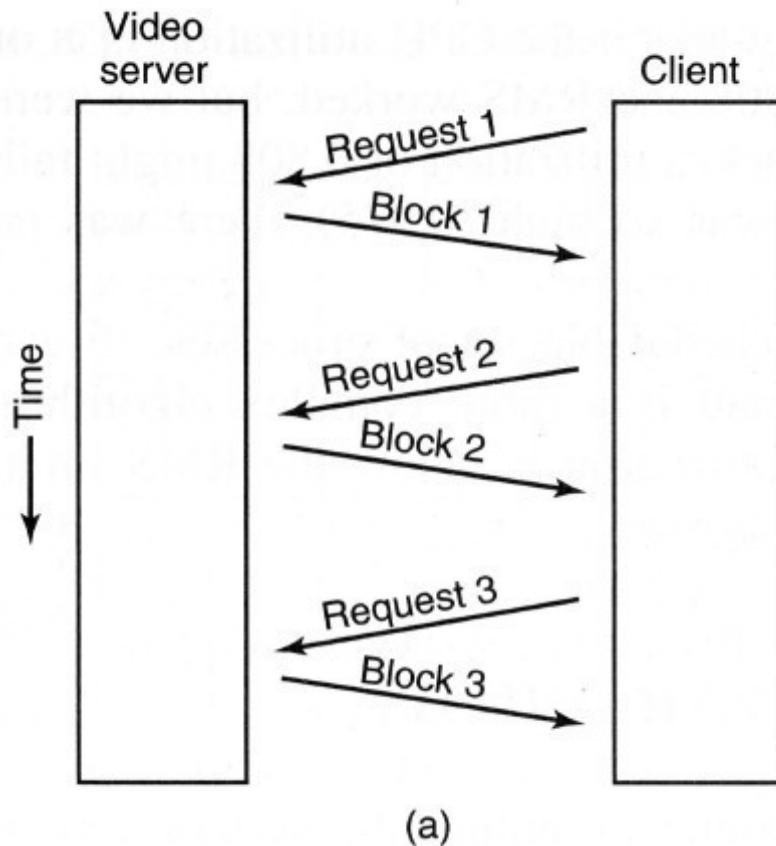
---

- *Live streaming*: to deliver a live event while it is occurring.
  - Broadcast or multicast
- *On-demand streaming*: to deliver archived media streams
  - Movies, lectures, old TV shows, etc.
  - Events *not* delivered when they occur.
  - Playback with *pause, fast forward, reverse, etc*
  - Unicast (usually)

# Network Streaming

- Traditional HTTP
  - Stateless
  - Server responds to each request independently
- Real-Time Streaming Protocol (RTSP)
  - Client initiates a “push” request for stream
  - Server provides media stream at frame rate

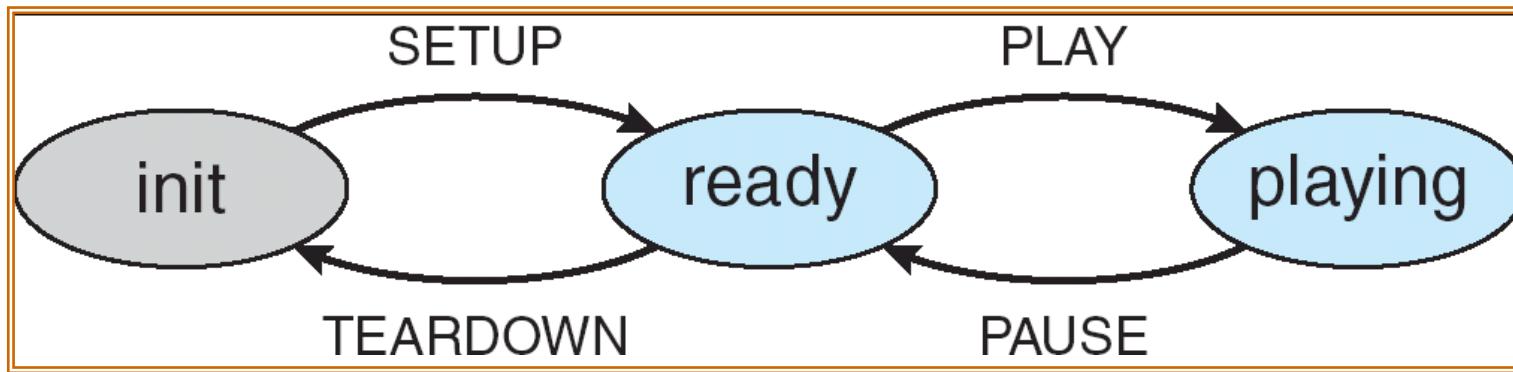
# Pull (HTTP) vs. Push (RTSP) server



# RTSP States

- *SETUP*: server allocates resources for client session.
- *PLAY*: server delivers stream to a client session.
- *PAUSE*: server suspends delivery of a stream.
- *TEARDOWN*: server releases resources and breaks down connection.

# RTP state machine



# Bandwidth Negotiation

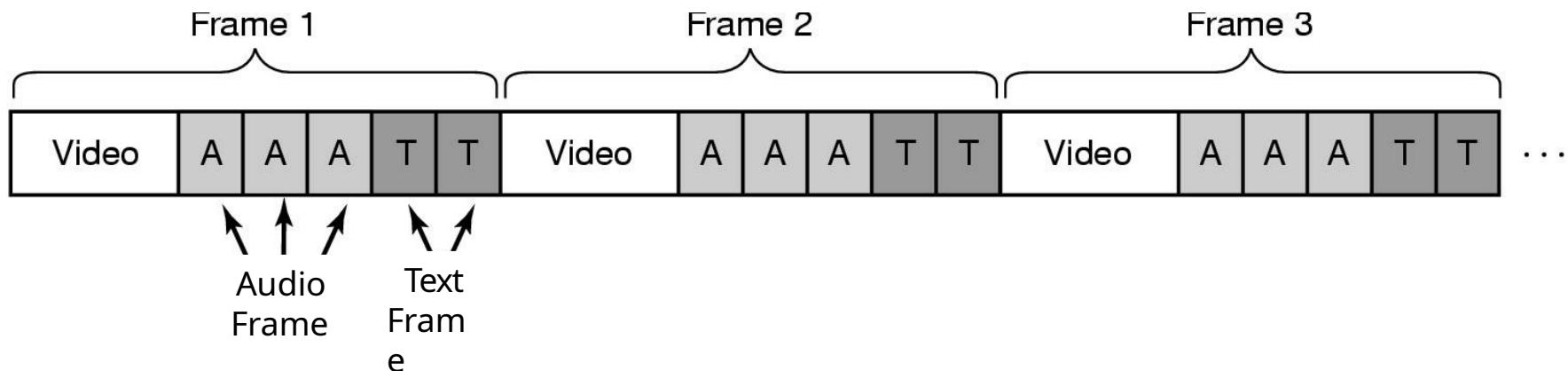
- Client application provides feedback to server to adjust bandwidth
- E.g.,
  - Windows Media Player
  - RealPlayer
  - Quicktime

# Video Server

- Multiple CPUs
- Disk farm
  - 1000s of disks
- Multiple high-bandwidth network links
  - Cable TV
  - Video on demand
  - Internet

# Multimedia File & Disk Management

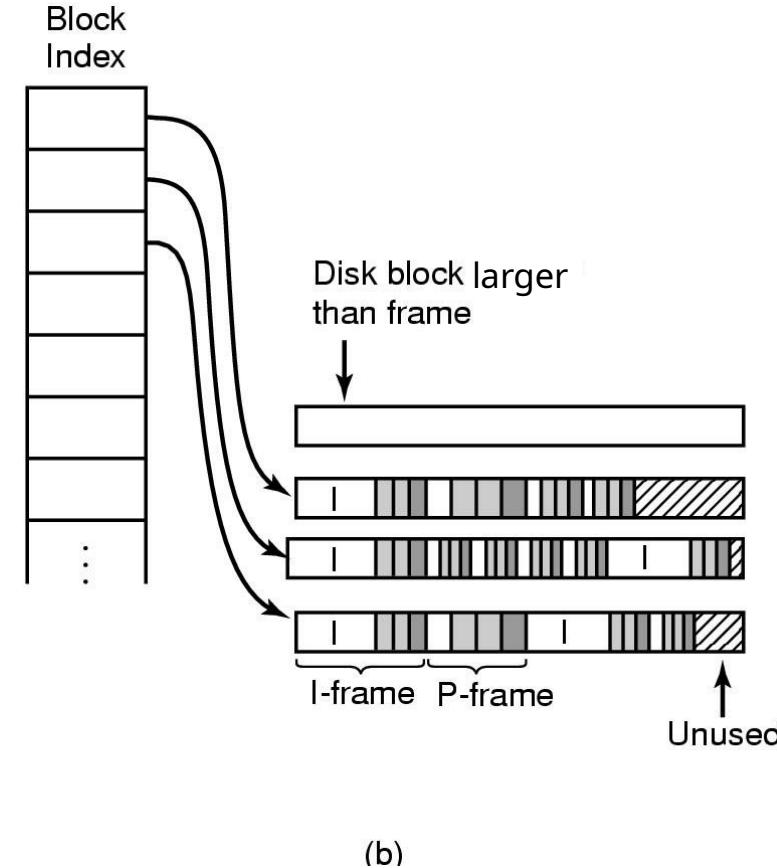
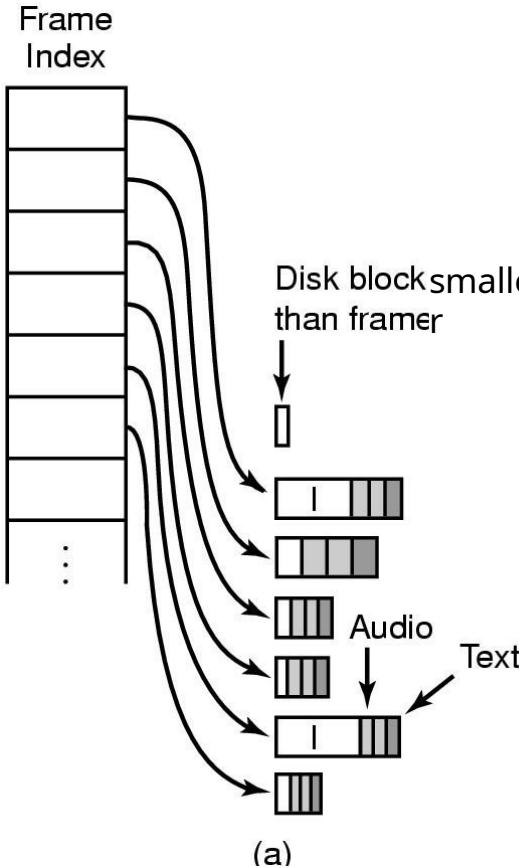
- Single movie or multimedia file on disk
  - Interleave audio, video, etc.
    - So temporally equivalent blocks are near each other
  - Attempt contiguous allocation
    - Avoid seeks within a frame



# File organization - Frame vs. Block

- Frame organization
  - Small disk blocks (4-16 Kbytes)
  - Frame index entries point to starting block for each frame
  - Frames vary in size (MPEG)
  - Advantage: very little storage fragmentation
  - Disadvantage: large frame table in RAM
- Block organization
  - Large disk block (256 Kbytes or more)
  - Block index entries point to first *I-frame* of a sequence
  - Multiple frames per block
  - Advantage: much smaller block table in RAM
  - Disadvantage: large storage fragmentation on disk

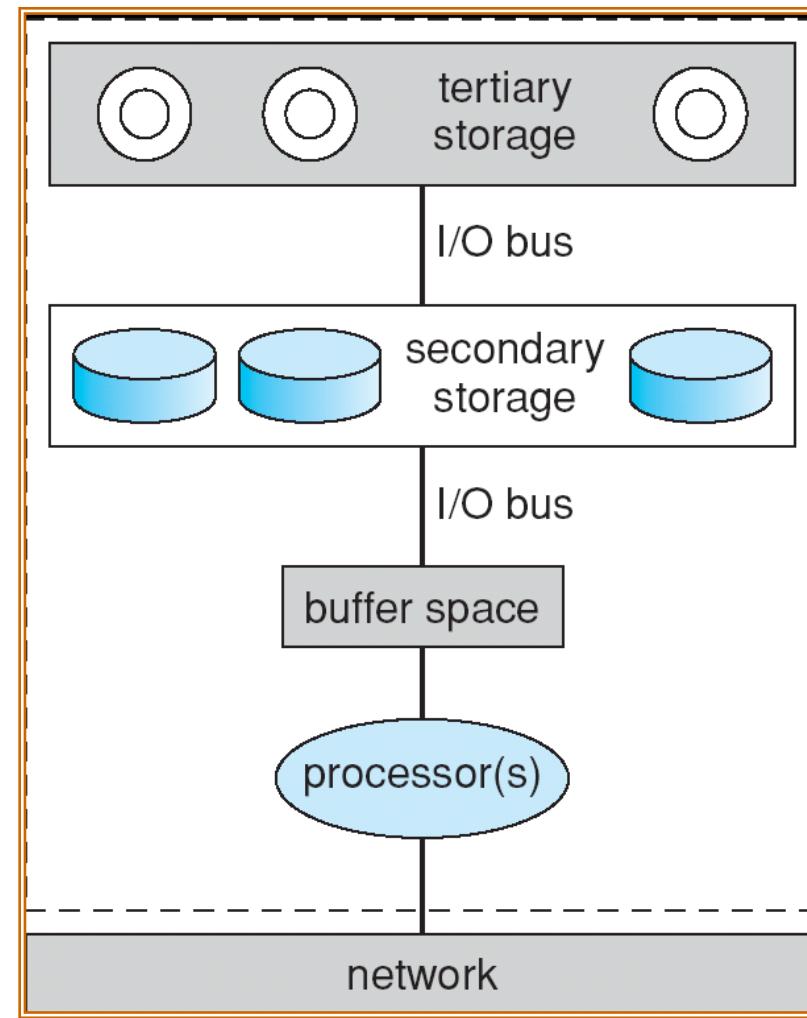
# Frame vs. Block organization



# File Placement on Server

- Random
- Striped
- “Organ pipe” allocation
  - Most popular video in center of disk
  - Next most popular on either side of it, etc.
  - Least popular at edges of disk
  - Minimizes seek distance

# Resources on a file server



# Conclusion

- Multimedia computing is challenging
- Possible with modern computers
  - Compression is essential, especially for video
- Real-time computing techniques move into mainstream
  - Processor and disk scheduling
- There is much more to this subject than fits into one class

# Thank You