Course Code: AITB04

OPERATING SYSTEMS

Regulation: IARE-R18

B . TECH: IV SEM

Prepared by:

Dr. D.Kishore Babu, Associate. professor
Dr.K Suvarchala, Associate Professor, CSE
Dr. Ch Santaiah, Associate Professor, CSE
Mrs.Y.Deepthi, Assistant Professor, CSE
Mr. S.Laxman Kumar, Assistant Professor, CSE
Mrs. B Pravallika, Assistant Professor, CSE,
Mrs. T Navya, Assistant Professor, CSE

Course Objectives

The course should enable the students to:	
CO 1	Understand the fundamental principles of the operating system, its services and functionalities..
CO 2	Illustrate the concepts of processes, inter-process communication, synchronization and scheduling.
CO 3	Understand different types of memory management viz. virtual memory, paging and segmentation
CO 4	Identify the reasons for deadlock and understand the techniques for deadlock detection, prevention and recovery..
CO 5	Understand the need of protection and security mechanisms in computer systems

Course Learning Outcomes

The course will enable the students to:	
CLO 1	Describe the structure of operating system and basic architectural components involved in operating system design.
CLO 2	Describe how the computing resources are managed by the operating system.
CLO 3	Understand the objectives and functions of modern operating systems.
CLO 4	Analyze and design the applications to run in parallel either using process or thread models of different operating system
CLO 5	Understand and analyze implementation of virtual memory
CLO 6	Understand the various resource management techniques for timesharing and distributed systems.

Course Learning Outcomes cont.

The course will enable the students to:	
CLO 7	Describe the mutual exclusion, deadlock detection in operating system
CLO 8	Describe the common algorithms used for both pre-emptive and non-pre-emptive scheduling of tasks in operating systems, such a priority and performance comparison
CLO 9	Understand the difference between a process and a thread
CLO 10	Explain the state diagram that describes the states and state transitions during the whole lifetime of a process; likewise, interpret such a state transition diagram
CLO 11	Identify the mapping between virtual memory address into a physical address
CLO 12	Explain how a shared memory area can be implemented using virtual memory addresses in different processes

Course Learning Outcomes cont.

The course will enable the students to:	
CLO 13	Identify the need of memory management in operating systems and understand the limits of fixed memory allocation schemes
CLO 14	Understand the fragmentation in dynamic memory allocation, and identify dynamic allocation approaches
CLO 15	Understand how program memory addresses relate to physical memory addresses, memory management in base-limit machines, and swapping
CLO 16	Understand the mechanisms adopted for file distribution in applications
CLO 17	Describe different Mass storage structure and I/O systems
CLO 18	Understand issues related to file system interface and implementation, disk management

Course Learning Outcomes cont.

The course will enable the students to:	
CLO 19	Identify the mechanisms adopted for file sharing in distributed applications
CLO 20	Understand the concepts of Storage Management, disk management and disk scheduling

MODULE –I:
INTRODUCTION

Contents

- Operating System Objectives and functions
- Computer System Architecture
- Operating System Structure
- Operating System Operations
- Evolution of Operating System
- System Calls
- Protection And Security
- Operating System Design and Implementation
- Virtual Machines

Operating System Objectives and functions

FEATURES

- An operating system is a program that acts as an interface between the software and the computer hardware.
- It is an integrated set of specialized programs used to manage overall and resources operations of the computer.
- It is a specialized software that controls and monitors the execution of all other programs that reside in the computer, including application programs and other system software

Objectives of Operating System

The objectives of the operating system are

- To make the computer system convenient to use in an efficient manner.
- To hide the details of the hardware resources from the users.
- To provide users a convenient interface to use the computer system.
- To act as an intermediary between the hardware and its users, making it easier for the users to access and use other resources.
- To manage the resources of a computer system.
- To keep track of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- To provide efficient and fair sharing of resources among users and programs.

Functions of Operating System

An operating system includes all the programs of a computer system that control and monitor the operations of the system. Operating systems typically consist of a kernel that manages the hardware of the computer, as well as basic system programs that are used to boot the operating system and configure it. We are going to discuss main functions of operating system

1. Booting

Booting is a process of starting the computer operating system starts the computer to work. It checks the computer and makes it ready to work.

2. Memory Management

It is also an important function of operating system. The memory cannot be managed without operating system. Different programs and data execute in memory at one time. if there is no operating system, the programs may mix with each other. The system will not work properly.

3. Loading and Execution

A program is loaded in the memory before it can be executed. Operating system provides the facility to load programs in memory easily and then execute it.

4. Data Security

Data is an important part of computer system. The operating system protects the data stored on the computer from illegal use, modification or deletion.

5.Disk Management

Operating system manages the disk space. It manages the stored files and folders in a proper way.

6.Process Management

CPU can perform one task at one time. if there are many tasks, operating system decides which task should get the CPU.

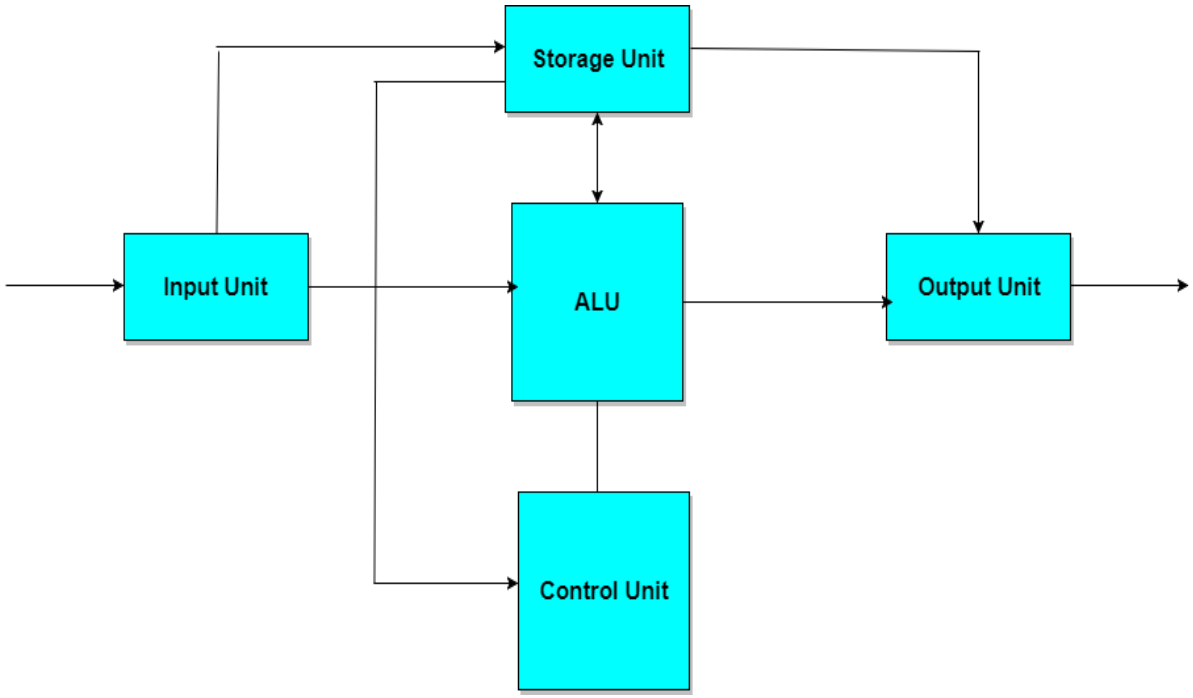
7.Device Controlling

operating system also controls all devices attached to computer. The hardware devices are controlled with the help of small software called device drivers.

8.Printing Controlling

Operating system also controls printing function. It a user issues two print commands at a time, it does not mix data of these files and prints them separately

Computer System structure

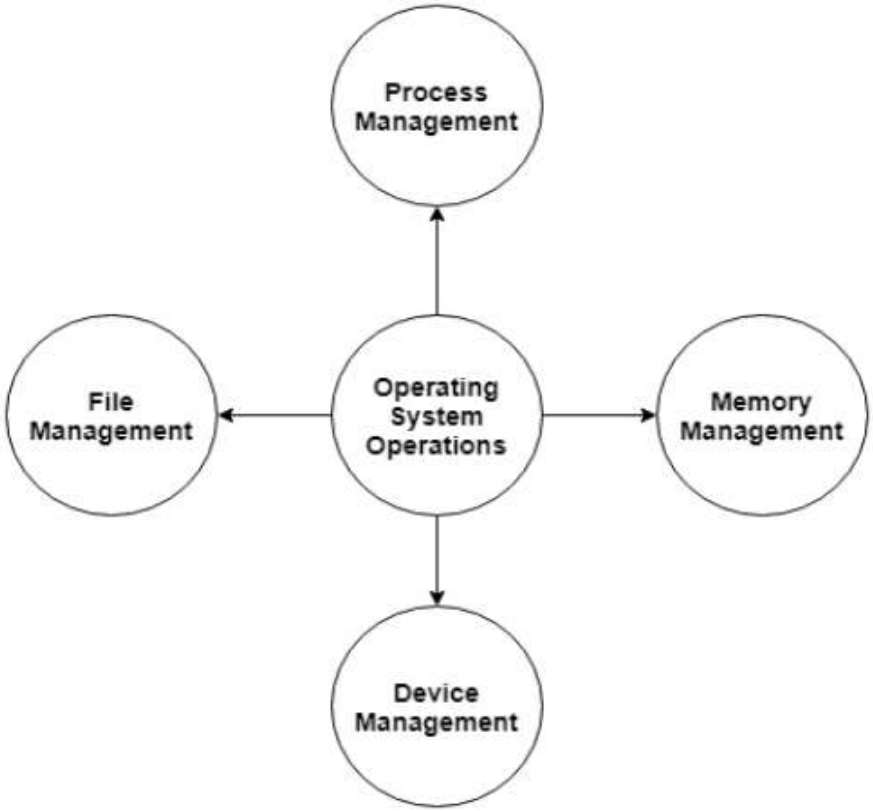


Operating System operations

- An operating system is a construct that allows the user application programs to interact with the system hardware.

- Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.
- The major operations of the operating system are process management, memory management, device management and file management. These are given in detail as follows:

Operating System operations



Process Management

Process Management

- The operating system is responsible for managing the processes i.e assigning the processor to a process at a time. This is known as process scheduling.

- The different algorithms used for process scheduling are FCFS (first come first served), SJF (shortest job first), priority scheduling, round robin scheduling etc.
- There are many scheduling queues that are used to handle processes in process management. When the processes enter the system, they are put into the job queue.

- The processes that are ready to execute in the main memory are kept in the ready queue. The processes that are waiting for the I/O device are kept in the device queue.

Memory Management

- Memory management plays an important part in operating system. It deals with memory and the moving of processes from disk to primary memory for execution and back again.
- The activities performed by the operating system for memory management are:
- The operating system assigns memory to the processes as required. This can be done using best fit, first fit and worst fit algorithms.
- All the memory is tracked by the operating system i.e. it nodes what memory parts are in use by the processes and which are empty.
- The operating system deallocated memory from processes as required. This may happen when a process has been terminated or if it no longer needs the memory.

Contd..

Device Management

There are many I/O devices handled by the operating system such as mouse, keyboard, disk drive etc. There are different device drivers that can be connected to the operating system to handle a specific device.

The device controller is an interface between the device and the device driver. The user applications can access all the I/O devices using the device drivers, which are device specific codes

- **File Management**

Files are used to provide a uniform view of data storage by the operating system.

All the files are mapped onto physical devices that are usually non volatile so data is safe in the case of system failure.

Contd..

The files can be accessed by the system in two ways i.e. sequential access and direct access:

- **Sequential Access** The information in a file is processed in order using sequential access.
- The files records are accessed on after another. Most of the file systems such as editors, compilers etc. use sequential access.
- **Direct Access** In direct access or relative access, the files can be accessed in random for read and write operations.
- The direct access model is based on the disk model of a file, since it allows random accesses.

Types of Operating Systems

Following are some of the most widely used types of Operating system.

- Simple Batch System
- Multiprogramming Batch System
- Multiprocessor System
- Desktop System
- Distributed Operating System
- Clustered System
- Realtime Operating System
- Handheld System

Simple Batch Systems

- In this type of system, there is no direct interaction between user and the computer.
- The user has to submit a job (written on cards or tape) to a computer operator.
- Then computer operator places a batch of several jobs on an input device.
- Jobs are batched together by type of languages and requirement.
- Then a special program, the monitor, manages the execution of each program in the batch.
- The monitor is always in the main memory and available for execution.

Multiprocessor Systems

A Multiprocessor system consists of several processors that share a common physical memory.

Multiprocessor system provides higher computing power and speed. In multiprocessor system all processors operate under single operating system.

Multiplicity of the processors and how they do act together are transparent to the others.

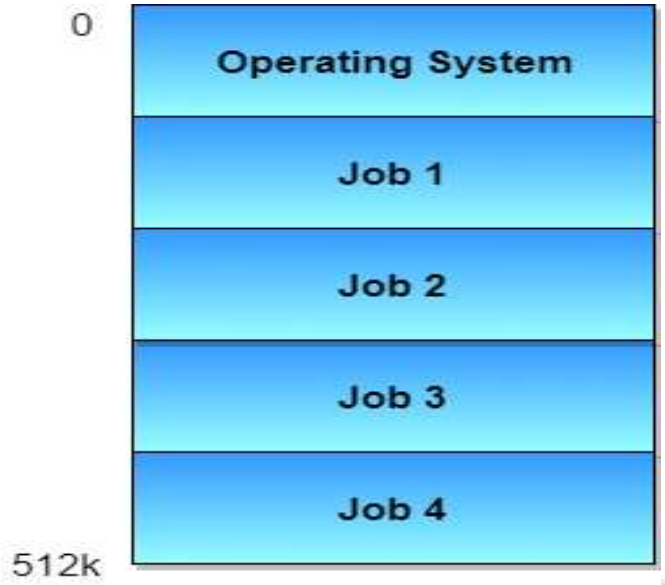
Advantages of Multiprocessor Systems

- Enhanced performance
- Execution of several tasks by different processors concurrently, increases the system's throughput without speeding up the execution of a single task.

- **Time Sharing Systems**

It very similar to Multiprogramming batch systems. In fact time sharing systems are an extension of multiprogramming systems.

In Time sharing systems the prime focus is on **minimizing the response time**, while in multiprogramming the prime focus is to maximize the CPU usage.



- Personal computer

An operating system is responsible for several tasks. These tasks fall into the following broad categories:

Processor management -- breaks down the processor's work into manageable chunks and prioritizes them before sending them to the CPU.
Memory management -- coordinates the flow of data in and out of RAM, and determines when to use virtual memory on the hard disk to supplement an insufficient amount of RAM.
Device management -- provides a software-based interface between the computer's internal components and each device connected to the computer.
Storage management -- directs where data should be stored permanently on hard drives, solid state drives, USB drives and other forms of storage. For example, storage management tasks assist when creating, reading, editing, moving, copying and deleting documents.

Personal computer

- **Application interface** -- provides data exchange between software programs and the PC. An application must be programmed to work with the application interface for the operating system you're using.

- Applications are often designed for specific versions of an OS, too. You'll see this in the application's requirements with phrases like "Windows Vista or later," or "only works on 64-bit operating systems.“

- **User interface (UI)** - provides a way for you to interact with the computer.

- Parallel and distributed systems

What is a parallel computer?

A collection of processing elements that communicate and coorperate to solve large problems fast.

What is a distributed system?

A collection of independent computers that appear to its users as a single coherent system.

A parallel computer is implicitly a distributed system

Real time systems

Real time systems

- It is defined as an operating system known to give maximum time for each of the critical operations that it performs, like OS calls and interrupt handling.

- The Real-Time Operating system which guarantees the maximum time for critical operations and complete them on time are referred to as Hard Real-Time Operating Systems.

- While the real-time operating systems that can only guarantee a maximum of the time, i.e.

- The critical task will get priority over other tasks, but no assurity of completeing it in a defined time. These systems are referred to as Soft Real-Time Operating Systems.

Clustered Systems

Clustered Systems

- Like parallel systems, clustered systems gather together multiple CPUs to accomplish computational work.
- Clustered systems differ from parallel systems, however, in that they are composed of two or more individual systems coupled together.
- The definition of the term clustered is **not concrete;** the general accepted definition is that clustered computers share storage and are closely linked via LAN networking.
- Clustering is usually performed to provide **high availability**.
- A layer of cluster software runs on the cluster nodes. Each node can monitor one or more of the others.
- If the monitored machine fails, the monitoring machine can take ownership of its storage, and restart the application(s) that were running on the failed machine. The failed machine can remain down, but the users and clients of the application would only see a brief interruption of service.

Clustered Systems

- **Asymmetric Clustering -** In this, one machine is in hot standby mode while the other is running the applications.
- The hot standby host (machine) does nothing but monitor the active server. If that server fails, the hot standby host becomes the active server.
- **Symmetric Clustering -** In this, two or more hosts are running applications, and they are monitoring each other.
- This mode is obviously more efficient, as it uses all of the available hardware.
- **Parallel Clustering -** Parallel clusters allow multiple hosts to access the same data on the shared storage.
- Because most operating systems lack support for this simultaneous data access by multiple hosts, parallel clusters are usually accomplished by special versions of software and special releases of applications.

System calls

- In computing, a **system call** is the programmatic way in which a computer program requests a service from the kernel of the operating system it is executed on. A system call is a way for programs to **interact with the operating system**. A computer program makes a system call when it makes a request to the operating system's kernel. System call **provides** the services of the operating system to the user programs via Application Program Interface(API). It provides an interface between a process and operating system to allow user-level processes to request services of the operating system. System calls are the only entry points into the kernel system. All programs needing resources must use system calls.
- **Services Provided by System Calls :**

1. Process creation and management
2. Main memory management
3. File Access, Directory and File system management
4. Device handling(I/O)
5. Protection

Types of System Calls

There are mainly five types of system calls. These are explained in detail as follows:

Process Control

These system calls deal with processes such as process creation, process termination etc.

File Management

These system calls are responsible for file manipulation such as creating a file reading file, writing into a file etc.

Device Management

These system calls are responsible for device manipulation such as reading from device buffers, writing into device buffers etc.

Information Maintenance

These system calls handle information and its transfer between the operating system and the user program.

Communication

These system calls are useful for interprocess communication. They also deal with creating and deleting a communication connection.

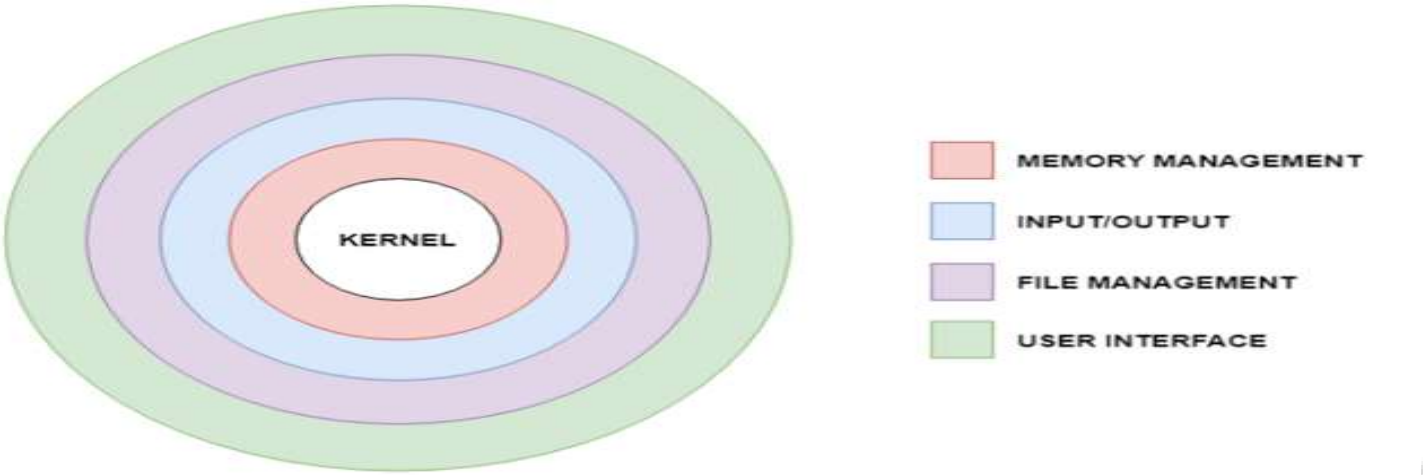
Operating System - Security

- Security refers to providing a protection system to computer system resources such as CPU, memory, disk, software programs and most importantly data/information stored in the computer system
- Authentication, One Time passwords, Program Threats, System Threats
- Computer Security Classifications
- Authentication
- Authentication refers to identifying each user of the system and associating the executing programs with those users.
- **Username / Password** – User need to enter a registered username and password with Operating system to login into the system.
- **User card/key** – User need to punch card in card slot, or enter key generated by key generator in option provided by operating system to login into the system.
- **User attribute - fingerprint/ eye retina pattern/ signature** – User need to pass his/her attribute via designated input device used by operating system to login into the system

Operating System Design and Implementation

An operating system is a construct that allows the user application programs to interact with the system hardware. Operating system by itself does not provide any function but it provides an atmosphere in which different applications and programs can do useful work.

There are many problems that can occur while designing and implementing an operating system. These are covered in operating system design and implementation.



Layered Operating System Design

Operating System Design and Implementation

Operating System Design Goals

- It is quite complicated to define all the goals and specifications of the operating system while designing it.The design changes depending on the type of the operating system i.e if it is batch system, time shared system, single user system, multi user system, distributed system etc.
- There are basically two types of goals while designing an operating system. These are:

User Goals

- The operating system should be convenient, easy to use, reliable, safe and fast according to the users. However, these specifications are not very useful as there is no set method to achieve these goals.

System Goals

- The operating system should be easy to design, implement and maintain. These are specifications required by those who create, maintain and operate the operating system. But there is not specific method to achieve these goals as well.

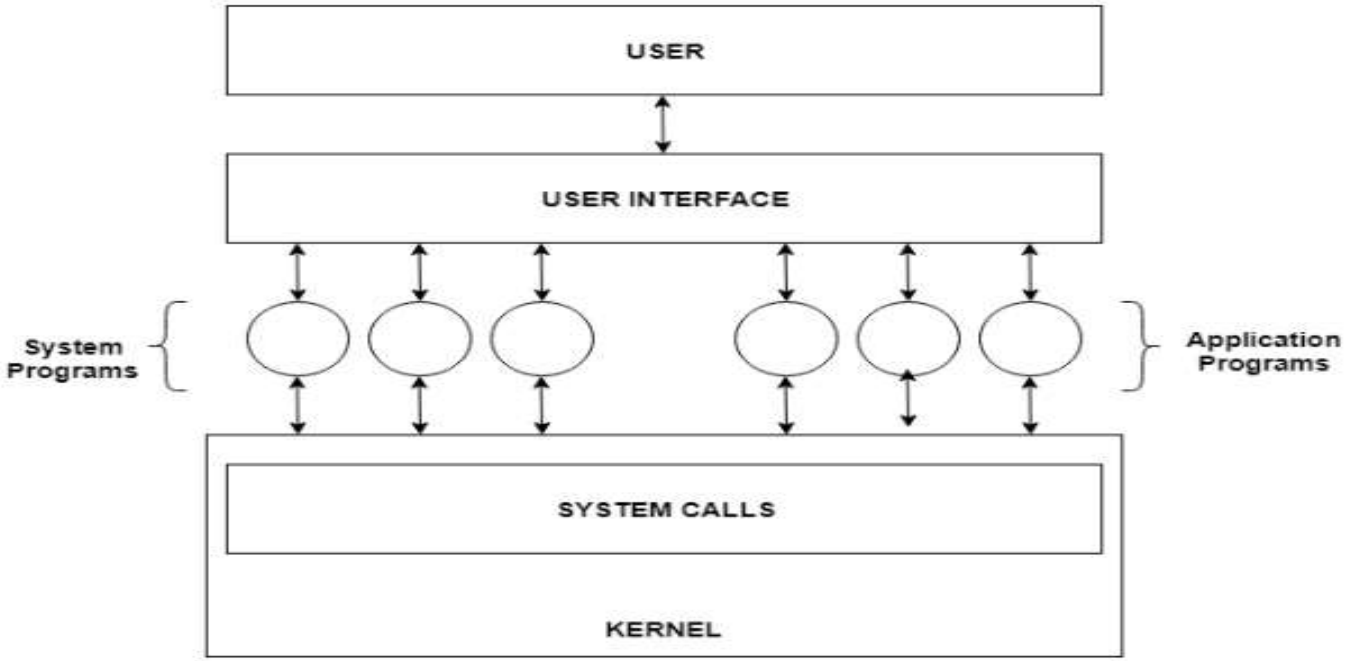
System programs

System programs provide an environment where programs can be developed and executed. In the simplest sense, system programs also provide a bridge between the user interface and system calls. In reality, they are much more complex. For example, a compiler is a complex system program.

System Programs Purpose

- The system program serves as a part of the operating system. It traditionally lies between the user interface and the system calls. The user view of the system is actually defined by system programs and not system calls because that is what they interact with and system programs are closer to the user interface.
- An image that describes system programs in the operating system hierarchy is as follows:
- In the above image, system programs as well as application programs form a bridge between the user interface and the system calls. So, from the user view the operating system observed is actually the system programs and not the system calls.

System programs



System programs

Types of System Programs

System programs can be divided into seven parts. These are given as follows:

Status Information

The status information system programs provide required data on the current or past status of the system. This may include the system date, system time, available memory in system, disk space, logged in users etc.

Communications

These system programs are needed for system communications such as web browsers. Web browsers allow systems to communicate and access information from the network as required.

File Manipulation

These system programs are used to manipulate system files. This can be done using various commands like create, delete, copy, rename, print etc. These commands can create files, delete files, copy the contents of one file into another, rename files, print them etc.

System programs

Program Loading and Execution

The system programs that deal with program loading and execution make sure that programs can be loaded into memory and executed correctly.

Loaders and Linkers are a prime example of this type of system programs.

File Modification

System programs that are used for file modification basically change the data in the file or modify it in some other way.

Text editors are a big example of file modification system programs.

Application Programs

Application programs can perform a wide range of services as per the needs of the users.

These include programs for database systems, word processors, plotting tools, spreadsheets, games, scientific applications etc.

Programming Language Support

These system programs provide additional support features for different programming languages. Some examples of these are compilers, debuggers etc.

Protection and security

- Protection and security requires that computer resources such as CPU, software, memory etc. are protected.

- This extends to the operating system as well as the data in the system.

- This can be done by ensuring integrity, confidentiality and availability in the operating system.
- The system must be protect against unauthorized access, viruses, worms etc.

Threats to Protection and Security

A threat is a program that is malicious in nature and leads to harmful effects for the system. Some of the common threats that occur in a system are:

Protection and security

Virus

Viruses are generally small snippets of code embedded in a system. They are very dangerous and can corrupt files, destroy data, crash systems etc. They can also spread further by replicating themselves as required.

Trojan Horse

A trojan horse can secretly access the login details of a system. Then a malicious user can use these to enter the system as a harmless being and wreak havoc

Trap Door

A trap door is a security breach that may be present in a system without the knowledge of the users.

Worm

A worm can destroy a system by using its resources to extreme levels.

Denial of Service

These type of attacks do not allow the legitimate users to access a system.

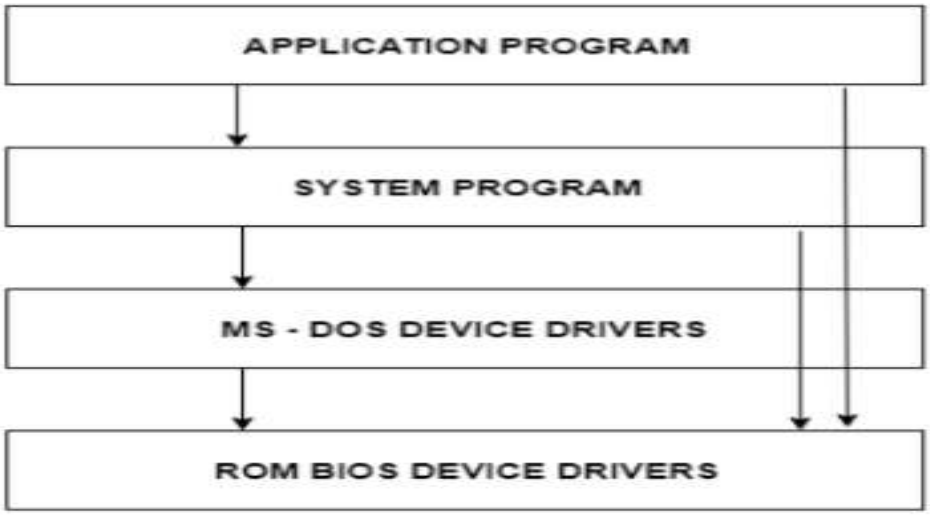
Operating system structure

- An **operating system** is a construct that allows the user application programs to interact with the **system** hardware. Since the **operating system** is such a complex **structure**, it should be created with utmost care so it can be used and modified easily.

Simple Structure

- There are many operating systems that have a rather simple structure. These started as small systems and rapidly expanded much further than their scope.
- A common example of this is MS-DOS. It was designed simply for a niche amount for people. There was no indication that it would become so popular.
- An image to illustrate the structure of MS-DOS is as follows:

Contd..



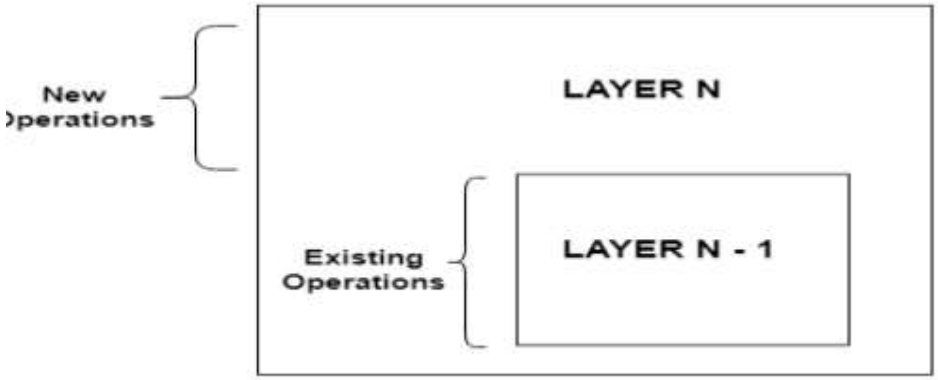
MS-DOS STRUCTURE

It is better that operating systems have a modular structure, unlike MS-DOS. That would lead to greater control over the computer system and its various applications. The modular structure would also allow the programmers to hide information as required and implement internal routines as they see fit without changing the outer specifications.

Contd..

Layered Structure

- One way to achieve modularity in the operating system is the layered approach. In this, the bottom layer is the hardware and the topmost layer is the user interface.
- An image demonstrating the layered approach is as follows:



Layered Structure of Operating System

Contd..

- As seen from the image, each upper layer is built on the bottom layer. All the layers hide some structures, operations etc from their upper layers.
- One problem with the layered structure is that each layer needs to be carefully defined. This is necessary because the upper layers can only use the functionalities of the layers below them.
- **Virtual Machine** abstracts the hardware of our personal computer such as CPU, disk drives, memory, NIC (Network Interface Card) etc, into many different execution environments as per our requirements, hence giving us a feel that each execution environment is a single computer. For example, VirtualBox.
- When we run different processes on an operating system, it creates an illusion that each process is running on a different processor having its own virtual memory, with the help of CPU scheduling and virtual-memory techniques. There are additional features of a process that cannot be provided by the hardware alone like system calls and a file system..

Contd..

- The main drawback with the virtual-machine approach involves disk systems. Let us suppose that the physical machine has only three disk drives but wants to support seven virtual machines.
- Obviously, it cannot allocate a disk drive to each virtual machine, because virtual-machine software itself will need substantial disk space to provide virtual memory and spooling. The solution is to provide virtual disks.
- Users are thus given their own virtual machines. After which they can run any of the operating systems or software packages that are available on the underlying machine.
- The virtual-machine software is concerned with multi-programming multiple virtual machines onto a physical machine, but it does not need to consider any user-support software.
- This arrangement can provide a useful way to divide the problem of designing a multi-user interactive system, into two smaller pieces.

Contd..

Advantages:

- There are no protection problems because each virtual machine is completely isolated from all other virtual machines.
- Virtual machine can provide an instruction set architecture that differs from real computers.
- Easy maintenance, availability and convenient recovery.

Disadvantages:

- When multiple virtual machines are simultaneously running on a host computer, one virtual machine can be affected by other running virtual machines, depending on the workload.
- Virtual machines are not as efficient as a real one when accessing the hardware.

MODULE –II:
PROCESS AND CPU SCHEDULING, PROCESS COORDINATION

Contents

- Process concepts
- Process scheduling: Scheduling queues, schedulers, context switch, preemptive scheduling, dispatcher, scheduling criteria, scheduling algorithms, multiple processor scheduling;
- Real time scheduling;
- Thread scheduling;
- Case studies Linux windows; Process synchronization, the critical section problem;
- Peterson‘s solution
- Synchronization hardware
- Semaphores and classic problems of synchronization, monitors.

Process Concept

An operating system executes a variety of programs:

- Batch system – jobs
- Time-shared systems – user programs or tasks

- Textbook uses the terms *job* and *process* almost interchangeably.
- Process – a program in execution; process execution must progress in sequential fashion.
- A process includes:
  - program counter
  - stack
  - data section

Process State

As a process executes, it changes *state*

- **new**: The process is being created.
- **running**: Instructions are being executed.
- **waiting**: The process is waiting for some event to occur.
- **ready**: The process is waiting to be assigned to a process.
- **terminated**: The process has finished execution.

Diagram of Process State

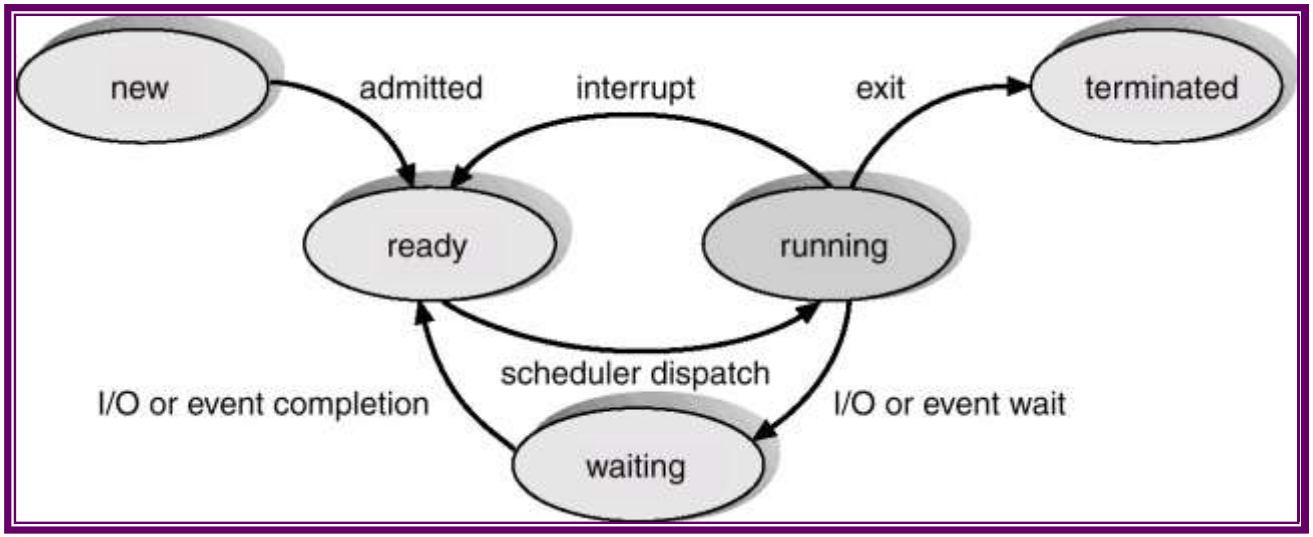


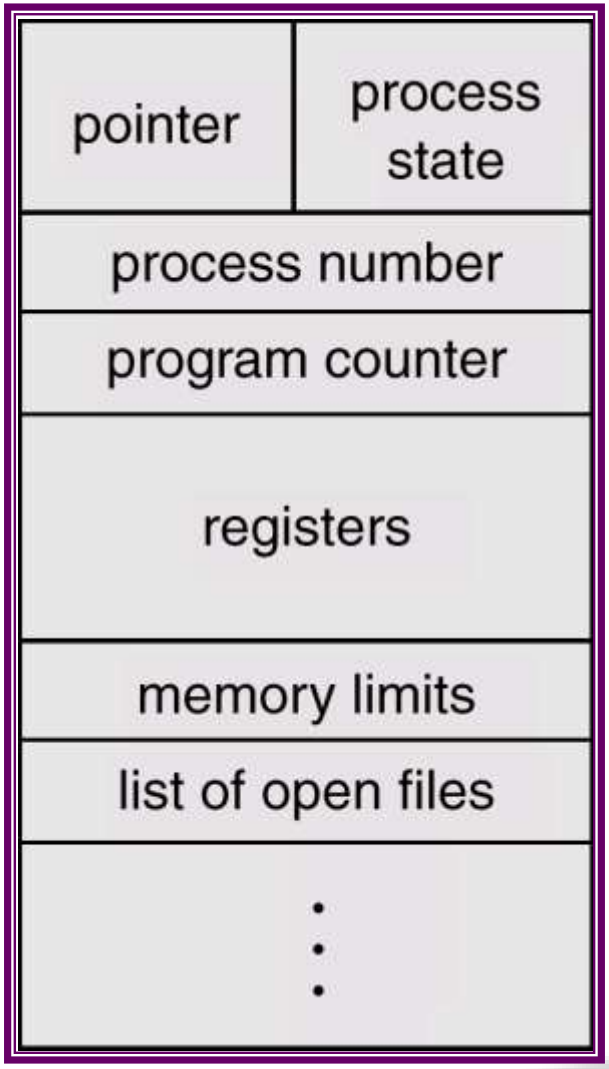
Fig.Diagram of Process State

Process Control Block (PCB)

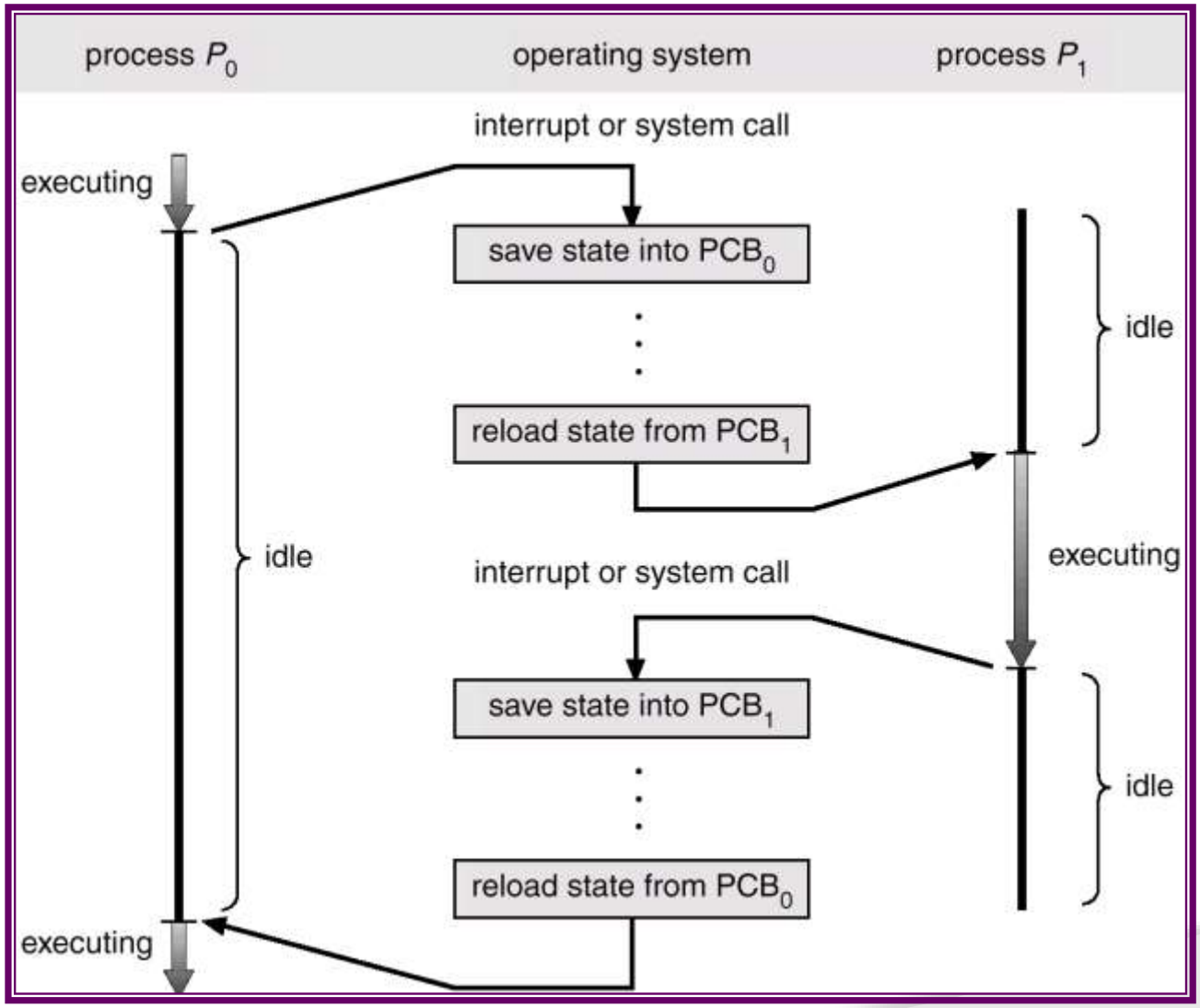
Information associated with each process.

- Process state
- Program counter
- CPU registers
- CPU scheduling information
- Memory-management information
- Accounting information
- I/O status information

Process Control Block (PCB)



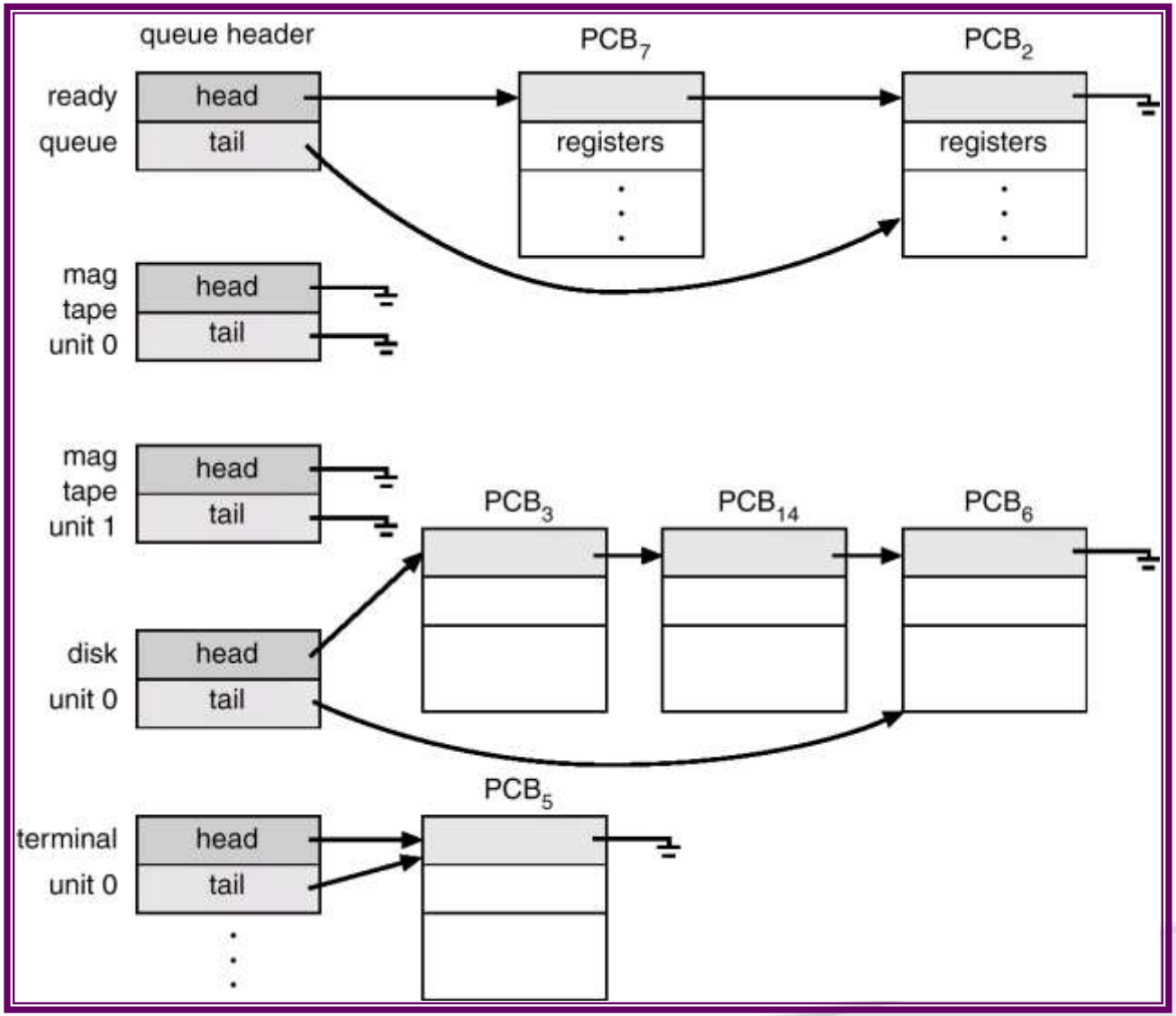
CPU Switch From Process to Process



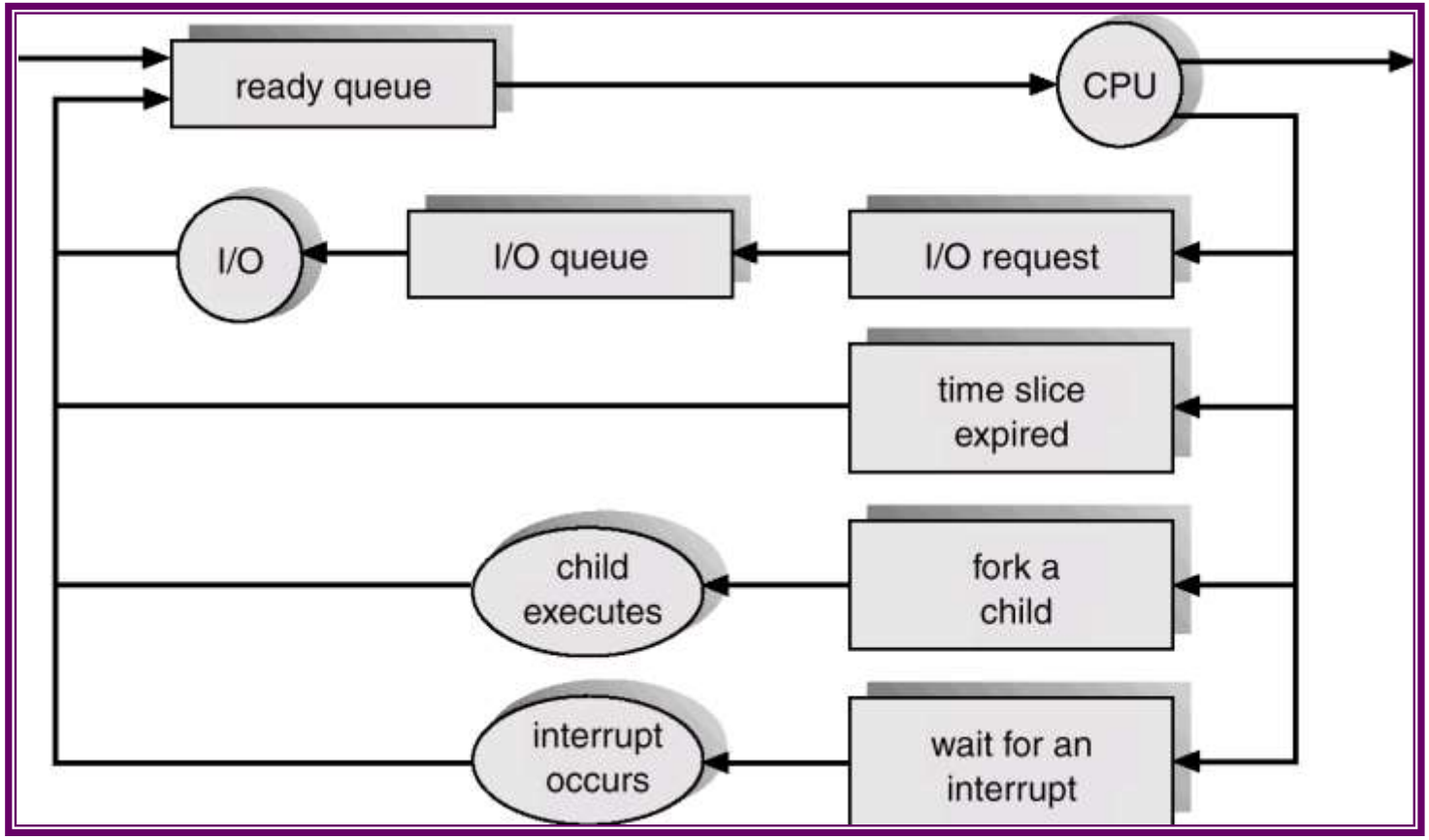
Process Scheduling Queues

- Job queue – set of all processes in the system.
- Ready queue – set of all processes residing in main memory, ready and waiting to execute.
- Device queues – set of processes waiting for an I/O device.
- Process migration between the various queues.

Ready Queue And Various I/O Device Queues



Representation of Process Scheduling

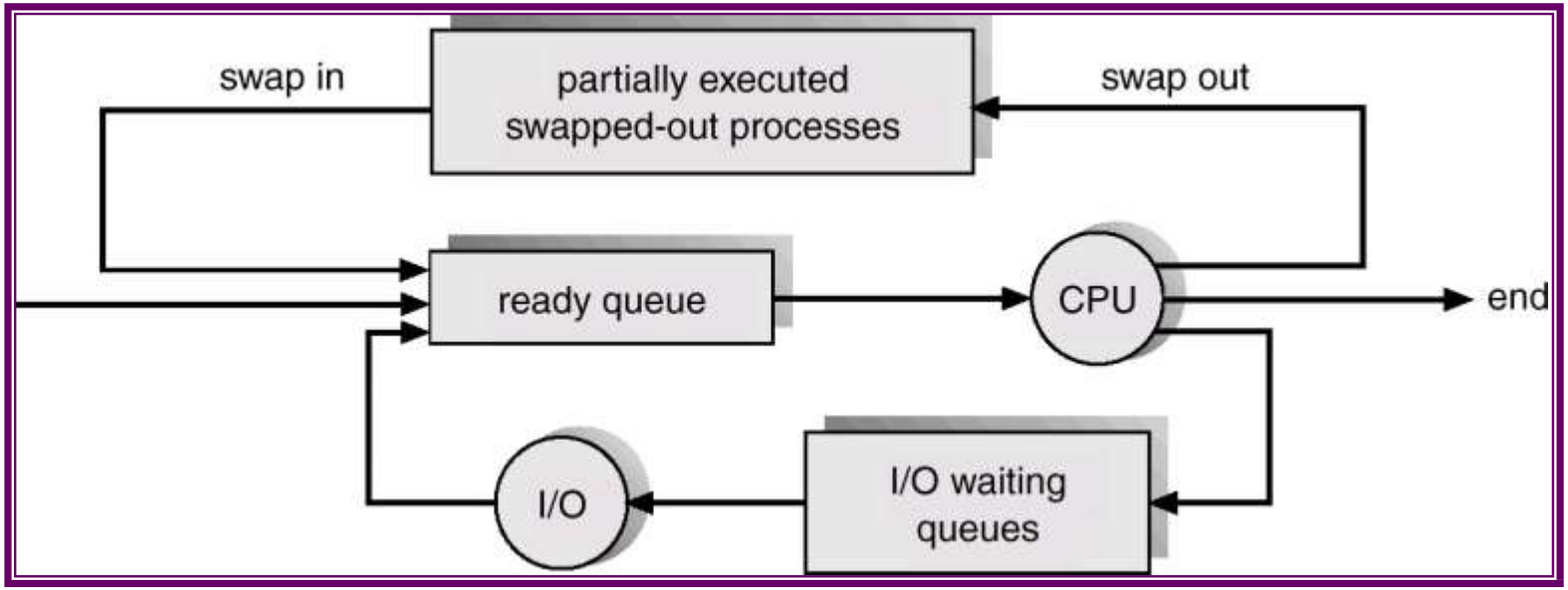


Schedulers

- Long-term scheduler (or job scheduler) – selects which processes should be brought into the ready queue.

- Short-term scheduler (or CPU scheduler) – selects which process should be executed next and allocates CPU.

Addition of Medium Term Scheduling



Schedulers (Cont.)

- Short-term scheduler is invoked very frequently (milliseconds) ⇒ (must be fast).
- Long-term scheduler is invoked very infrequently (seconds, minutes) ⇒ (may be slow).
- The long-term scheduler controls the *degree of multiprogramming.*
- Processes can be described as either:
  - I/O-*bound process* – spends more time doing I/O than computations, many short CPU bursts.
  - *CPU-bound process* – spends more time doing computations; few very long CPU bursts.

Context Switch

- When CPU switches to another process, the system must save the state of the old process and load the saved state for the new process.
- Context-switch time is overhead; the system does no useful work while switching.
- Time dependent on hardware support.

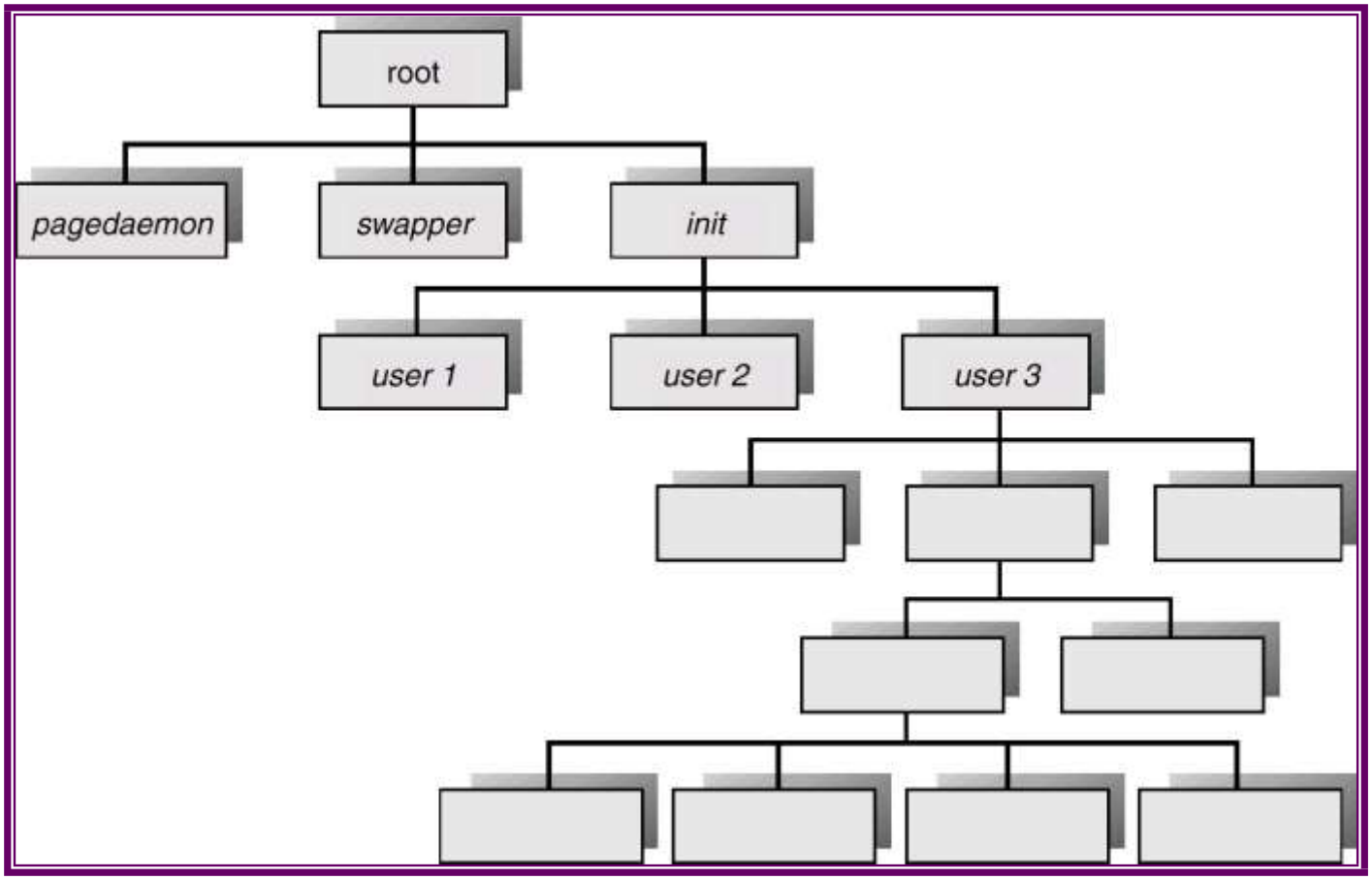
Process Creation

- Parent process create children processes, which, in turn create other processes, forming a tree of processes.
- Resource sharing
  - Parent and children share all resources.
  - Children share subset of parent's resources.
  - Parent and child share no resources.
- Execution
  - Parent and children execute concurrently.
  - Parent waits until children terminate.

Process Creation (Cont.)

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

Processes Tree on a UNIX System



Process Termination

- Process executes last statement and asks the operating system to decide it (**exit**).
  - Output data from child to parent (via **wait**).
  - Process' resources are deallocated by operating system.
- Parent may terminate execution of children processes (**abort**).
  - Child has exceeded allocated resources.
  - Task assigned to child is no longer required.
  - Parent is exiting.
    - Operating system does not allow child to continue if its parent terminates.
    - Cascading termination.

Cooperating Processes

- *Independent* process cannot affect or be affected by the execution of another process.
- *Cooperating* process can affect or be affected by the execution of another process
- Advantages of process cooperation
  - Information sharing
  - Computation speed-up
  - Modularity
  - Convenience

Producer-Consumer Problem

- Paradigm for cooperating processes, *producer* process produces information that is consumed by a *consumer* process.

- *unbounded-buffer* places no practical limit on the size of the buffer.

- *bounded-buffer* assumes that there is a fixed buffer size.

Bounded-Buffer – Shared-Memory Solution

- Shared data

```
#define BUFFER_SIZE 10

Typedef struct {

   . . .

} item;

item buffer[BUFFER_SIZE];

int in = 0;

int out = 0;
```

- Solution is correct, but can only use BUFFER_SIZE-1 elements

Interprocess Communication (IPC)

- Mechanism for processes to communicate and to synchronize their actions.
- Message system – processes communicate with each other without resorting to shared variables.
- IPC facility provides two operations:
  - **send**(*message*) – message size fixed or variable
  - **receive**(*message*)
- If *P* and *Q* wish to communicate, they need to:
  - establish a *communication link* between them
  - exchange messages via send/receive
- Implementation of communication link: physical (e.g., shared memory, hardware bus), logical (e.g., logical properties)

Direct Communication

- Processes must name each other explicitly:
  - **send** (*P, message*) – send a message to process P
  - **receive**(*Q, message*) – receive a message from process Q
- Properties of communication link
  - Links are established automatically.
  - A link is associated with exactly one pair of communicating processes.
  - Between each pair there exists exactly one link.
  - The link may be unidirectional, but is usually bi-directional.

Indirect Communication

- Messages are directed and received from mailboxes (also referred to as ports).
  - Each mailbox has a unique id.
  - Processes can communicate only if they share a mailbox.
- Properties of communication link
  - Link established only if processes share a common mailbox
  - A link may be associated with many processes.
  - Each pair of processes may share several communication links.
  - Link may be unidirectional or bi-directional.

Indirect Communication

- Operations
  - create a new mailbox
  - send and receive messages through mailbox
  - destroy a mailbox
- Primitives are defined as:

send(*A, message*) – send a message to mailbox A

receive(*A, message*) – receive a message from mailbox A

Synchronization

- Message passing may be either blocking or non-blocking.
- **Blocking** is considered **synchronous**
- **Non-blocking** is considered **asynchronous**
- **send** and **receive** primitives may be either blocking or non-blocking**.**

CPU Scheduler

- Selects from among the processes in memory that are ready to execute, and allocates the CPU to one of them.
- CPU scheduling decisions may take place when a process:
  1. Switches from running to waiting state.
  2. Switches from running to ready state.
  3. Switches from waiting to ready.
  4. Terminates.
- Scheduling under 1 and 4 is *nonpreemptive*.
- All other scheduling is *preemptive.*

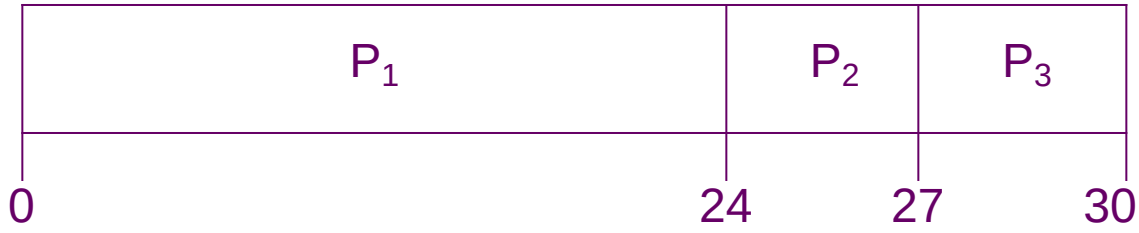
Scheduling Criteria

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

First-Come, First-Served (FCFS) Scheduling

Process	Burst Time
P_1	24
P_2	3
P_3	3

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



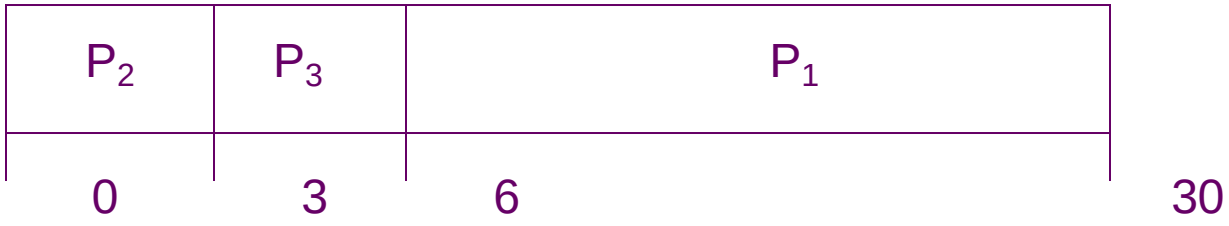
- Waiting time for P_1 = 0; P_2 = 24; P_3 = 27
- Average waiting time: (0 + 24 + 27)/3 = 17

FCFS Scheduling (Cont.)

Suppose that the processes arrive in the order

P_2 , P_3 , P_1 .

- The Gantt chart for the schedule is:



- Waiting time for P_1 = 6; P_2 = 0, P_3 = 3
- Average waiting time: (6 + 0 + 3)/3 = 3
- Much better than previous case.
- *Convoy effect* short process behind long process

Objectives of Operating System

The objectives of the operating system are

- To make the computer system convenient to use in an efficient manner.
- To hide the details of the hardware resources from the users.
- To provide users a convenient interface to use the computer system.
- To act as an intermediary between the hardware and its users, making it easier for the users to access and use other resources.
- To manage the resources of a computer system.
- To keep track of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- To provide efficient and fair sharing of resources among users and programs.

Shortest-Job-First (SJR) Scheduling

- Associate with each process the length of its next CPU burst. Use these lengths to schedule the process with the shortest time.
- Two schemes:

 nonpreemptive – once CPU given to the process it cannot be preempted until completes its CPU burst.

 preemptive – if a new process arrives with CPU burst length less than remaining time of current executing process, preempt. This scheme is know as the Shortest-Remaining-Time-First (SRTF).

- SJF is optimal – gives minimum average waiting time for a given set of processes.

Objectives of Operating System

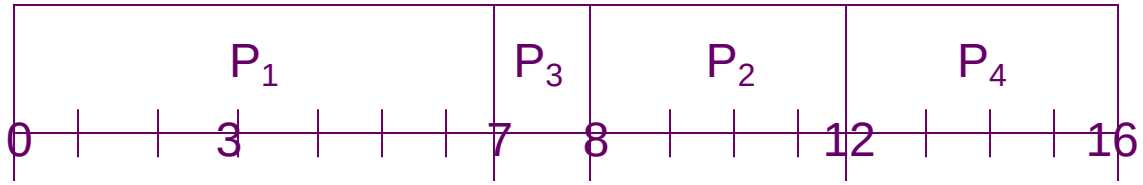
The objectives of the operating system are

- To make the computer system convenient to use in an efficient manner.
- To hide the details of the hardware resources from the users.
- To provide users a convenient interface to use the computer system.
- To act as an intermediary between the hardware and its users, making it easier for the users to access and use other resources.
- To manage the resources of a computer system.
- To keep track of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- To provide efficient and fair sharing of resources among users and programs.

Example of Non-Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (non-preemptive)



- Average waiting time = (0 + 6 + 3 + 7)/4 - 4

Objectives of Operating System

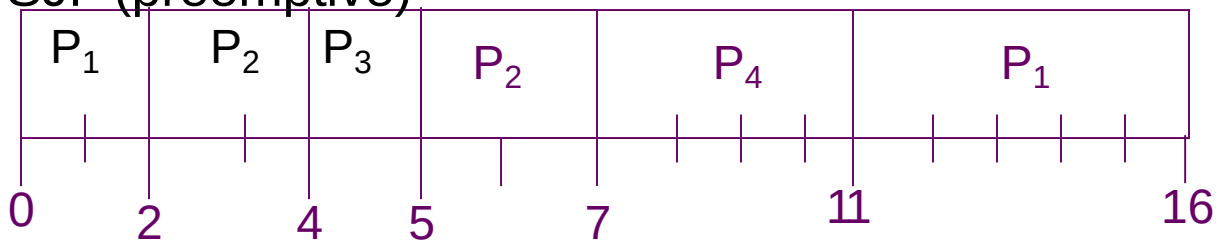
The objectives of the operating system are

- To make the computer system convenient to use in an efficient manner.
- To hide the details of the hardware resources from the users.
- To provide users a convenient interface to use the computer system.
- To act as an intermediary between the hardware and its users, making it easier for the users to access and use other resources.
- To manage the resources of a computer system.
- To keep track of who is using which resource, granting resource requests, and mediating conflicting requests from different programs and users.
- To provide efficient and fair sharing of resources among users and programs.

Example of Preemptive SJF

Process	Arrival Time	Burst Time
P_1	0.0	7
P_2	2.0	4
P_3	4.0	1
P_4	5.0	4

- SJF (preemptive)



- Average waiting time = (9 + 1 + 0 +2)/4 - 3

Priority Scheduling

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

Round Robin (RR)

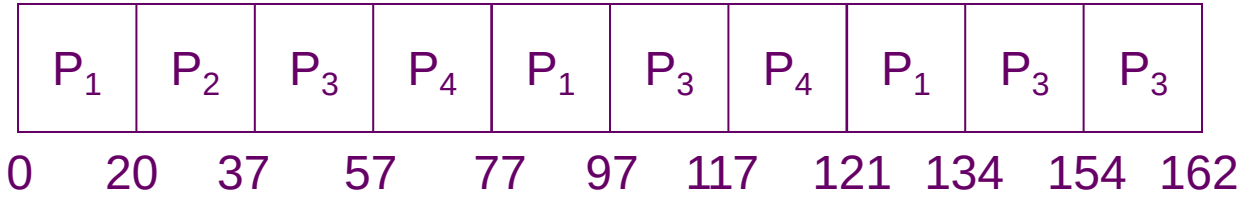
- Each process gets a small unit of CPU time (*time quantum*), usually 10-100 milliseconds. After this time has elapsed, the process is preempted and added to the end of the ready queue.
- If there are *n* processes in the ready queue and the time quantum is *q*, then each process gets 1/*n* of the CPU time in chunks of at most *q* time units at once.
- No process waits more than (*n*-1)*q* time units.
- Performance

 q large FIFO

 q small *q* must be large with respect to context switch, otherwise overhead is too high.

Example of RR with Time Quantum = 20

Process	Burst Time
P_1	53
P_2 P_3 P_4	17
The Gantt chart is:	68
	24



- Typically, higher average turnaround than SJF, but better *response*.

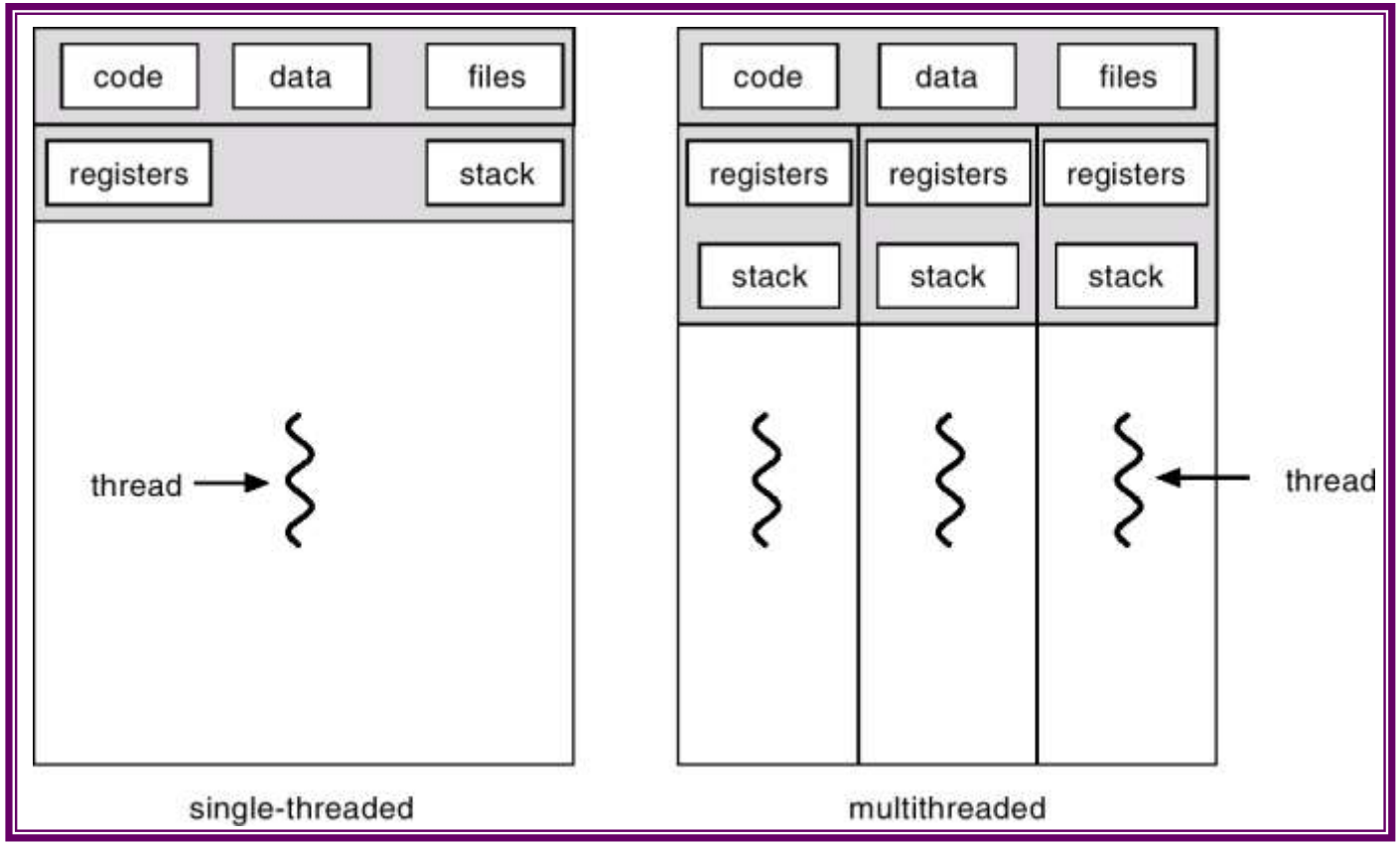
Multiple-Processor Scheduling

- CPU scheduling more complex when multiple CPUs are available.
- *Homogeneous processors* within a multiprocessor.
- *Load sharing*
- *Asymmetric multiprocessing* – only one processor accesses the system data structures, alleviating the need for data sharing.

Real-Time Scheduling

- *Hard real-time* systems – required to complete a critical task within a guaranteed amount of time.
- *Soft real-time* computing – requires that critical processes receive priority over less fortunate ones.

Single and Multithreaded Processes



Benefits

- Responsiveness
- Resource Sharing
- Economy
- Utilization of MP Architectures

User Threads

- Thread management done by user-level threads library
- Examples
  - POSIX *Pthreads*
  - Mach *C-threads*
  - Solaris *threads*

Kernel Threads

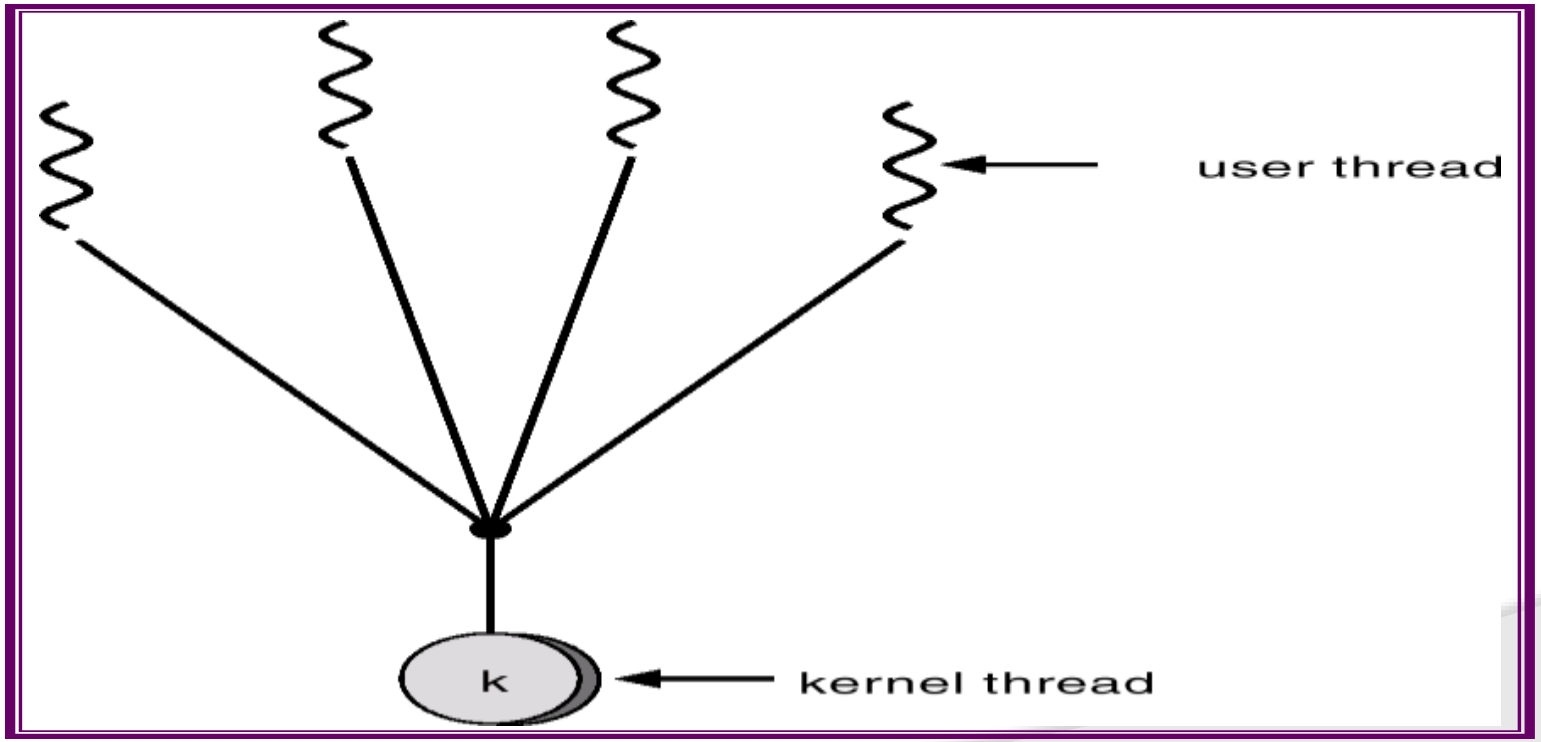
- Supported by the Kernel
- Examples
  - Windows 95/98/NT/2000
  - Solaris
  - Tru64 UNIX
  - BeOS
  - Linux

Multithreading Models

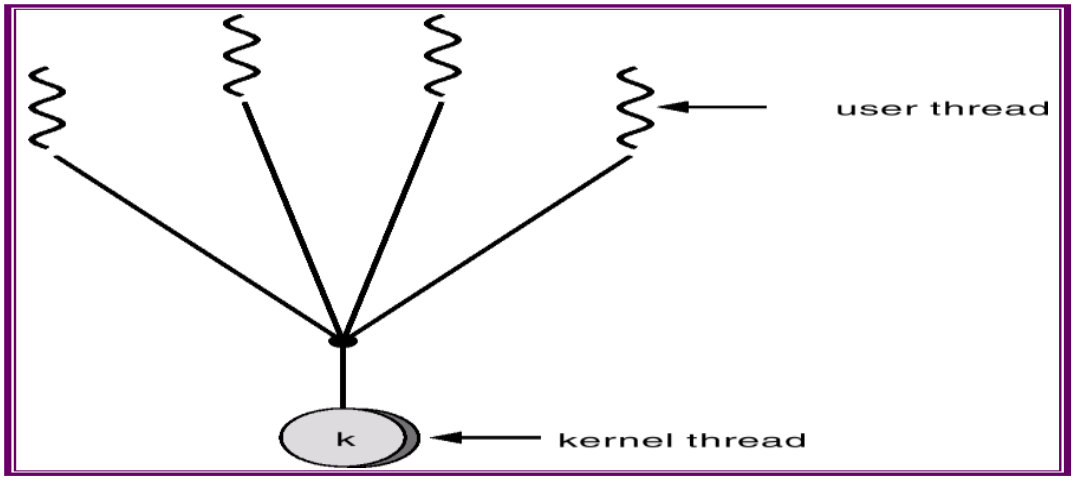
- Many-to-One
- One-to-One
- Many-to-Many

Many-to-One

- Many user-level threads mapped to single kernel thread.
- Used on systems that do not support kernel threads.

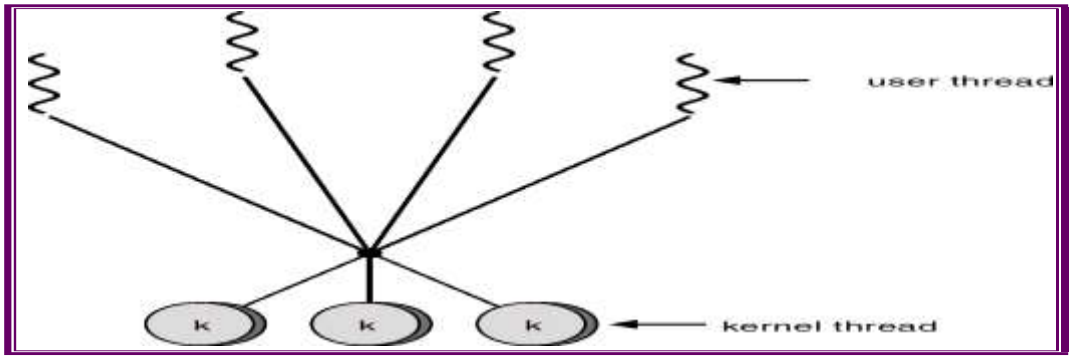


Many-to-One Model



- Allows many user level threads to be mapped to many kernel threads.
- Allows the operating system to create a sufficient number of kernel threads.
- Solaris 2
- Windows NT/2000 with the *ThreadFiber* package

Many-to-Many Model

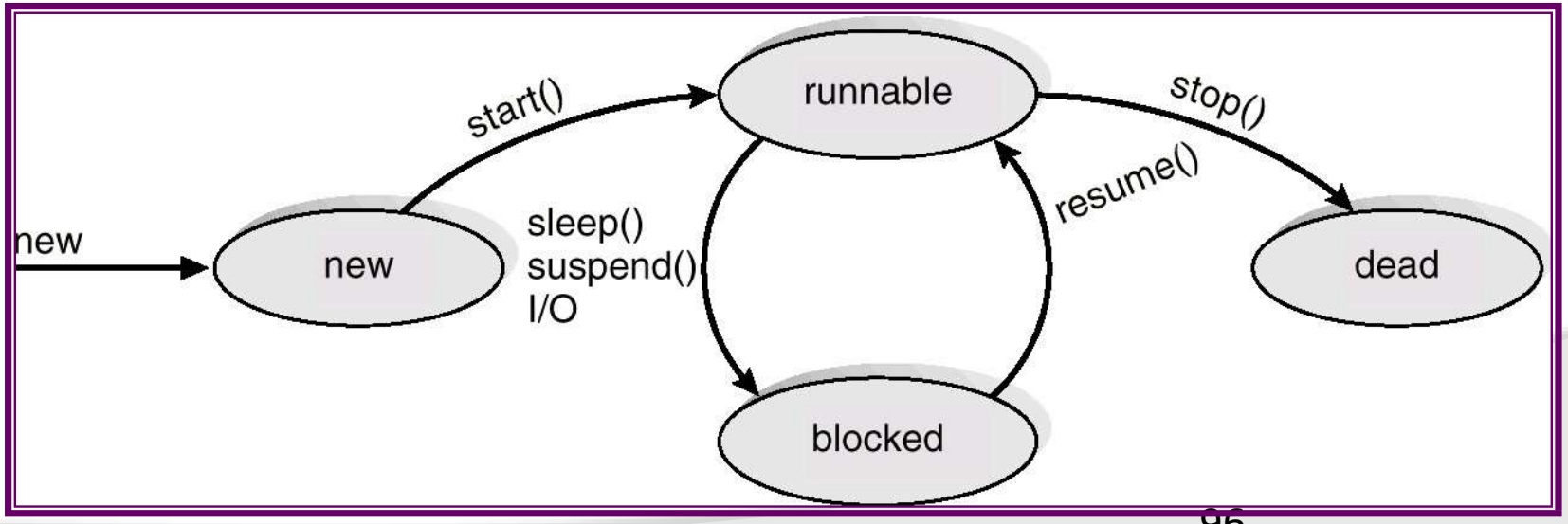


Linux Threads

- Linux refers to them as *tasks* rather than *threads*.
- Thread creation is done through clone() system call.
- Clone() allows a child task to share the address space of the parent task (process)

Java Threads

- Java threads may be created by:
  - ☞ Extending Thread class
  - ☞ Implementing the Runnable interface
- Java threads are managed by the JVM.



The Critical-Section Problem

- *n* processes all competing to use some shared data
- Each process has a code segment, called *critical section*, in which the shared data is accessed.
- Problem – ensure that when one process is executing in its critical section, no other process is allowed to execute in its critical section.

Solution to Critical-Section Problem

1. **Mutual Exclusion**. If process P_i is executing in its critical section, then no other processes can be executing in their critical sections.
2. **Progress**. If no process is executing in its critical section and there exist some processes that wish to enter their critical section, then the selection of the processes that will enter the critical section next cannot be postponed indefinitely.
3. **Bounded Waiting**. A bound must exist on the number of times that other processes are allowed to enter their critical sections after a process has made a request to enter its critical section and before that request is granted.
   - Assume that each process executes at a nonzero speed
   - No assumption concerning relative speed of the *n* processes.

Semaphores

- Synchronization tool that does not require busy waiting.
- Semaphore *S* – integer variable
- can only be accessed via two indivisible (atomic) operations

wait (*S*):

while *S*? 0 do *no-op*; *S*--;

signal (*S*):

S++;

MODULE-3
Memory Management

Memory Management

- Background
- Swapping
- Contiguous Allocation
- Paging
- Segmentation
- Segmentation with Paging
- Demand Paging
- Process Creation
- Page Replacement
- Allocation of Frames
- Thrashing
- Operating System Examples

Binding of Instructions and Data to Memory

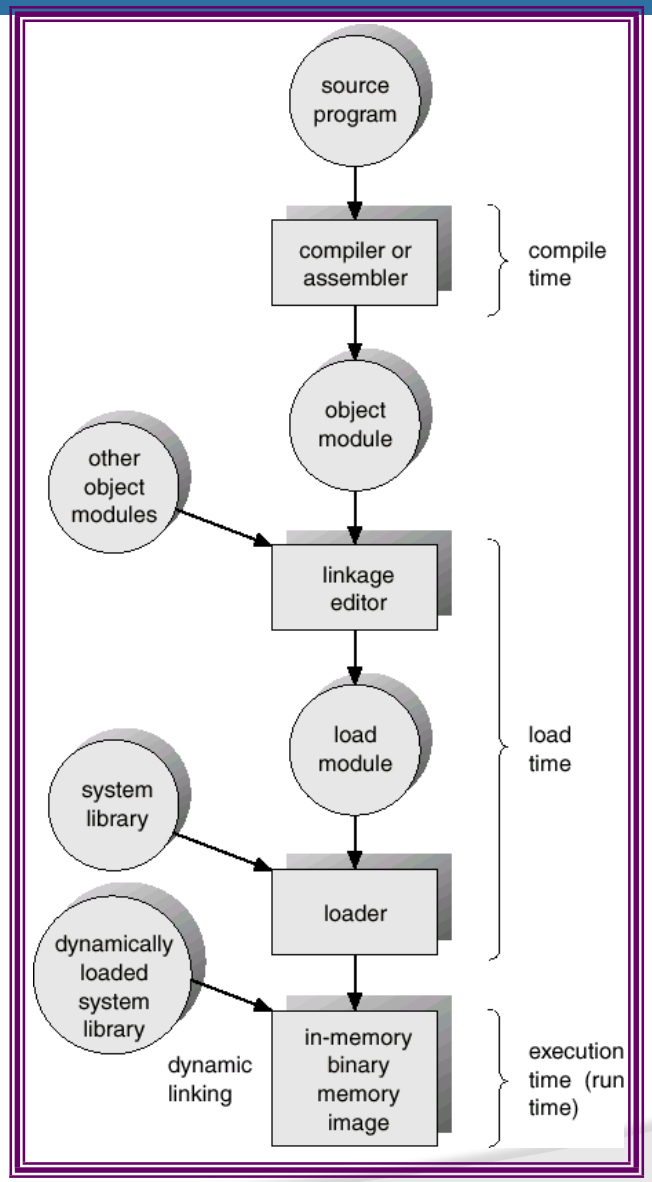
Address binding of instructions and data to memory addresses can happen at three different stages.

- **Compile time**: If memory location known a priori, absolute code can be generated; must recompile code if starting location changes.

- **Load time**: Must generate *reloadable* code if memory location is not known at compile time.

- **Execution time**:Binding delayed until run time if the process can be moved during its execution from one memory segment to another. Need hardware support for address maps (e.g., *base* and *limit registers*).

Multistep Processing of a User Program



Logical vs. Physical Address Space

- The concept of a logical *address space* that is bound to a separate *physical address space* is central to proper memory management.

- *Logical address* – generated by the CPU;
  also referred to as *virtual address*.

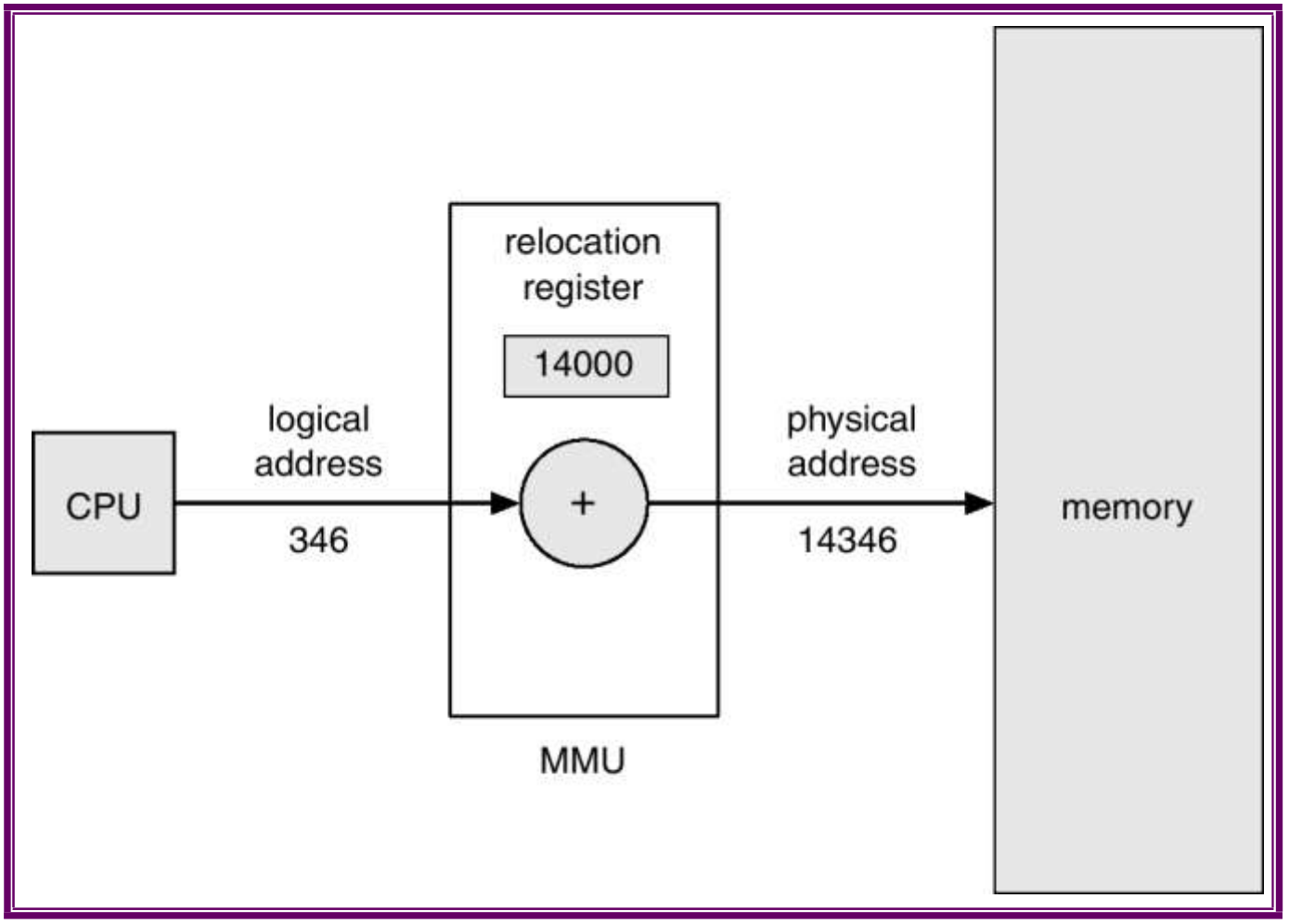
- *Physical address* – address seen by the memory unit.

- Logical and physical addresses are the same in compile-time and load-time address-binding schemes; logical (virtual) and physical addresses differ in execution-time address-binding scheme.

Memory-Management Unit (MMU)

- Hardware device that maps virtual to physical address.
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory.
- The user program deals with *logical* addresses; it never sees the *real* physical addresses.

Dynamic relocation using a relocation register



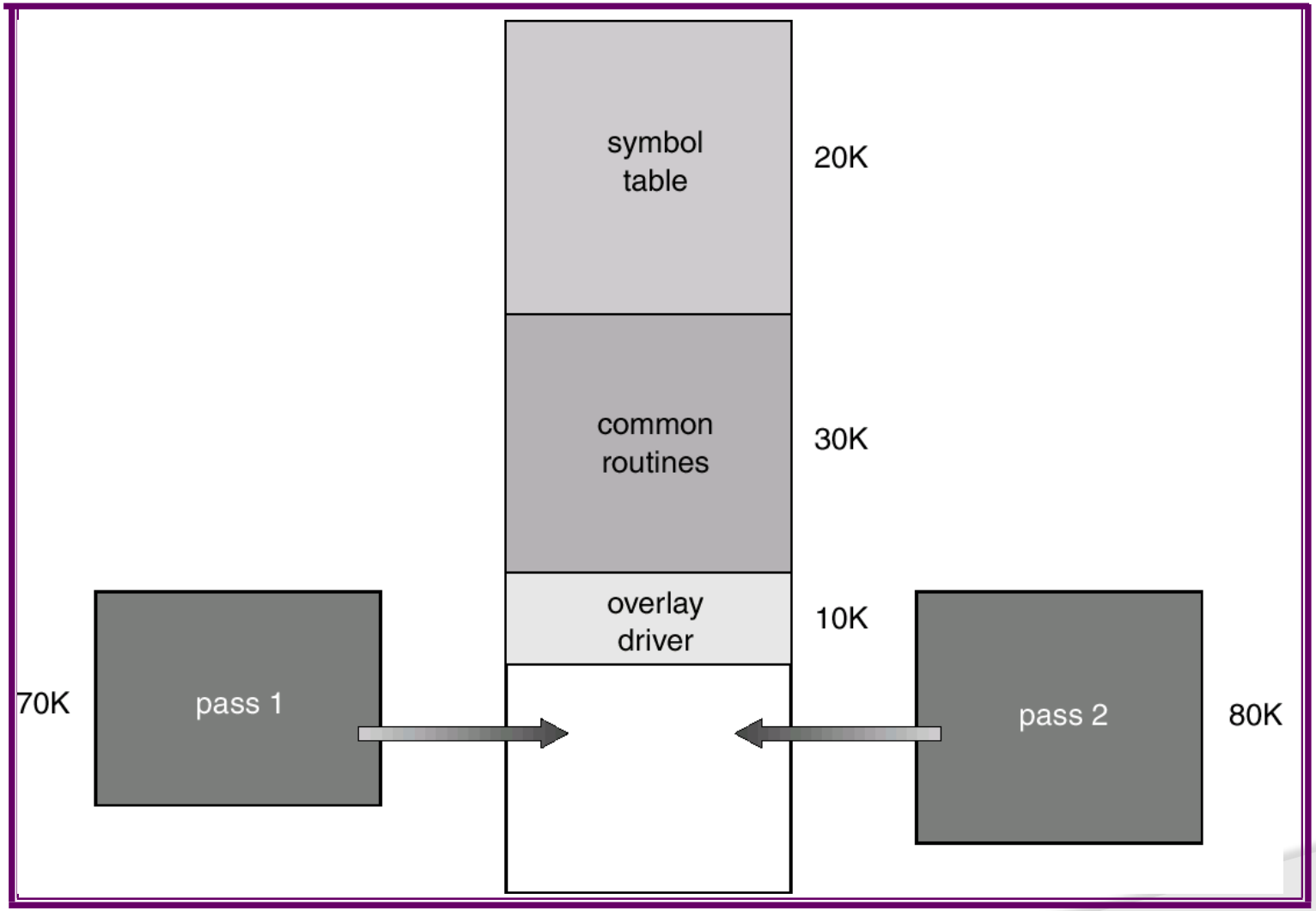
Dynamic Loading

- Routine is not loaded until it is called
- Better memory-space utilization; unused routine is never loaded.
- Useful when large amounts of code are needed to handle infrequently occurring cases.
- No special support from the operating system is required implemented through program design.

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 needed from operating system, programming design of overlay structure is complex

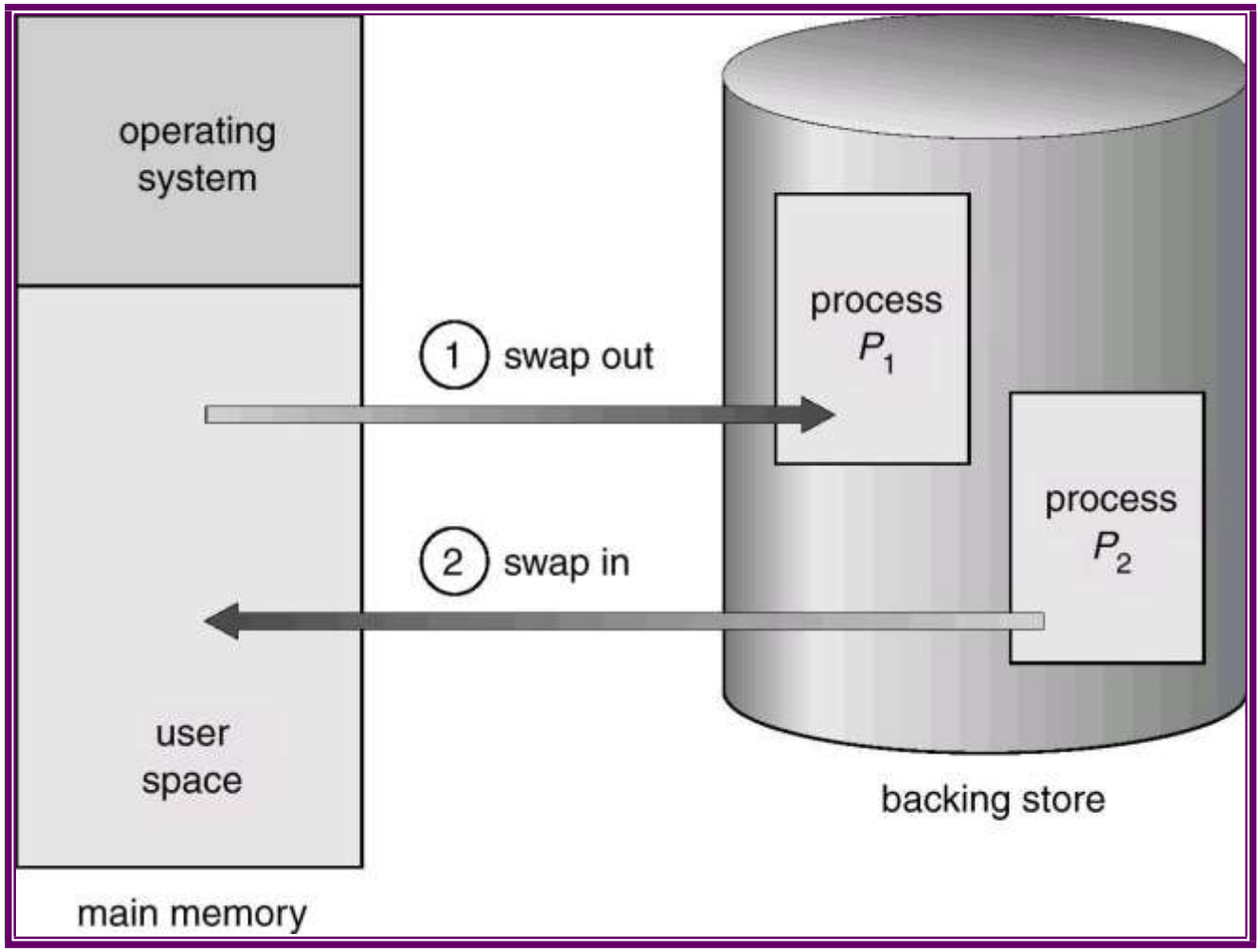
Overlays for a Two-Pass Assembler



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 accessto 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 andexecuted.
- Major part of swap time is transfer time; total transfer time is directly proportional to the *amount* of memory swapped.

Schematic View of Swapping

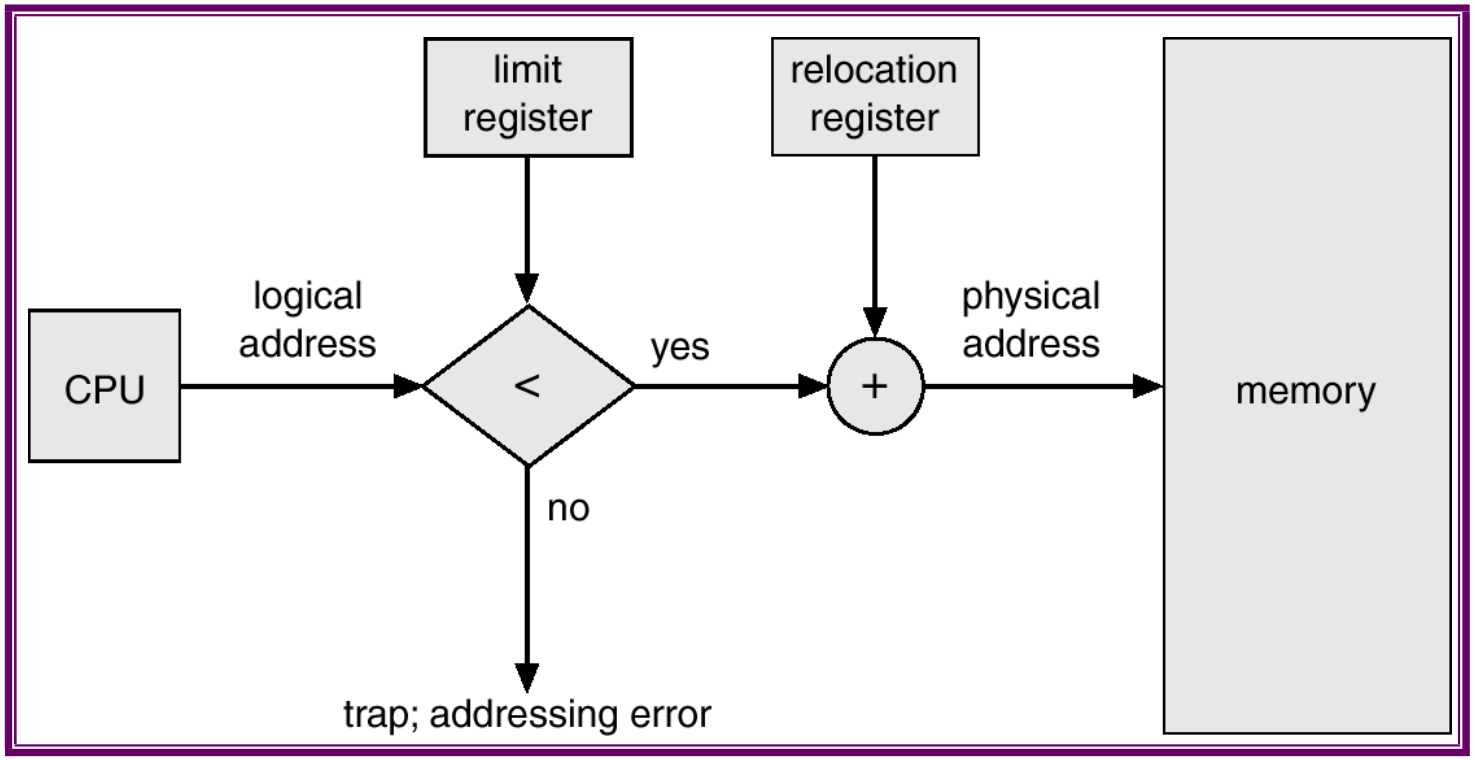


Contiguous Allocation

- Main memory usually into two partitions:
  - Resident operating system, usually held in low memory with interrupt vector.
  - User processes then held in high memory.

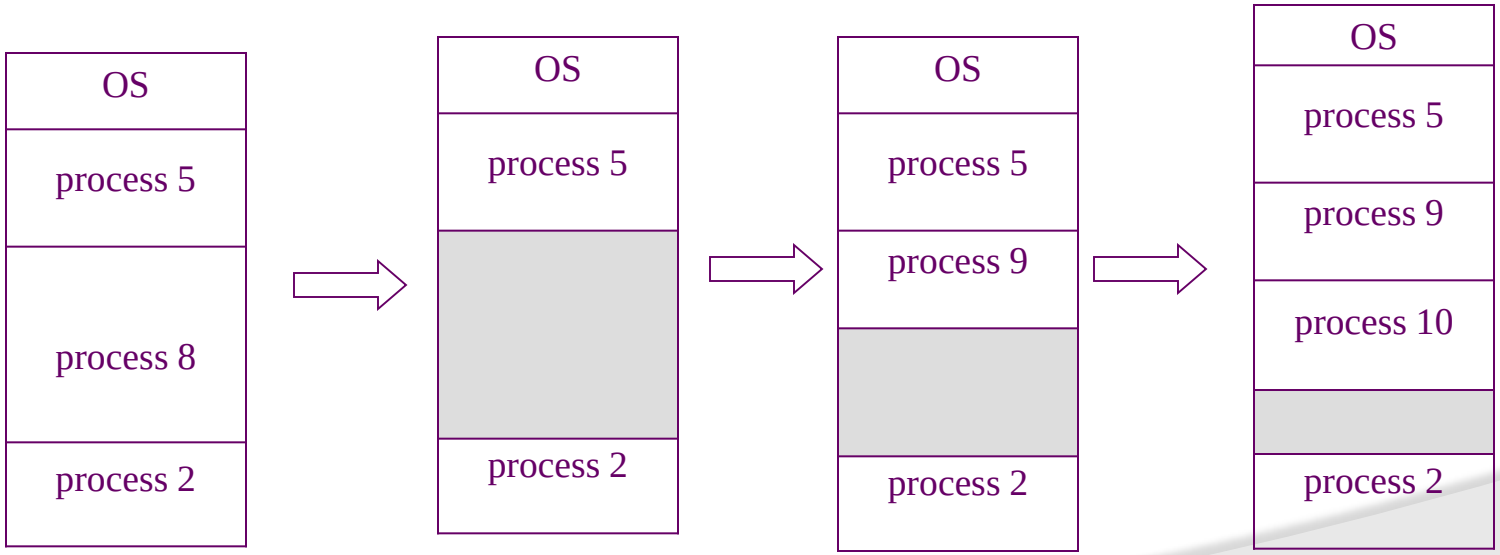
- Single-partition allocation
  - Relocation-register scheme used to protect user processes from each other, and from changing operating-system code and data.
  - Relocation register contains value of smallest physical address; limit register contains range of logical addresses each logical address must be less than the limit register.

Hardware Support for Relocation and Limit Registers



Contiguous Allocation (Cont.)

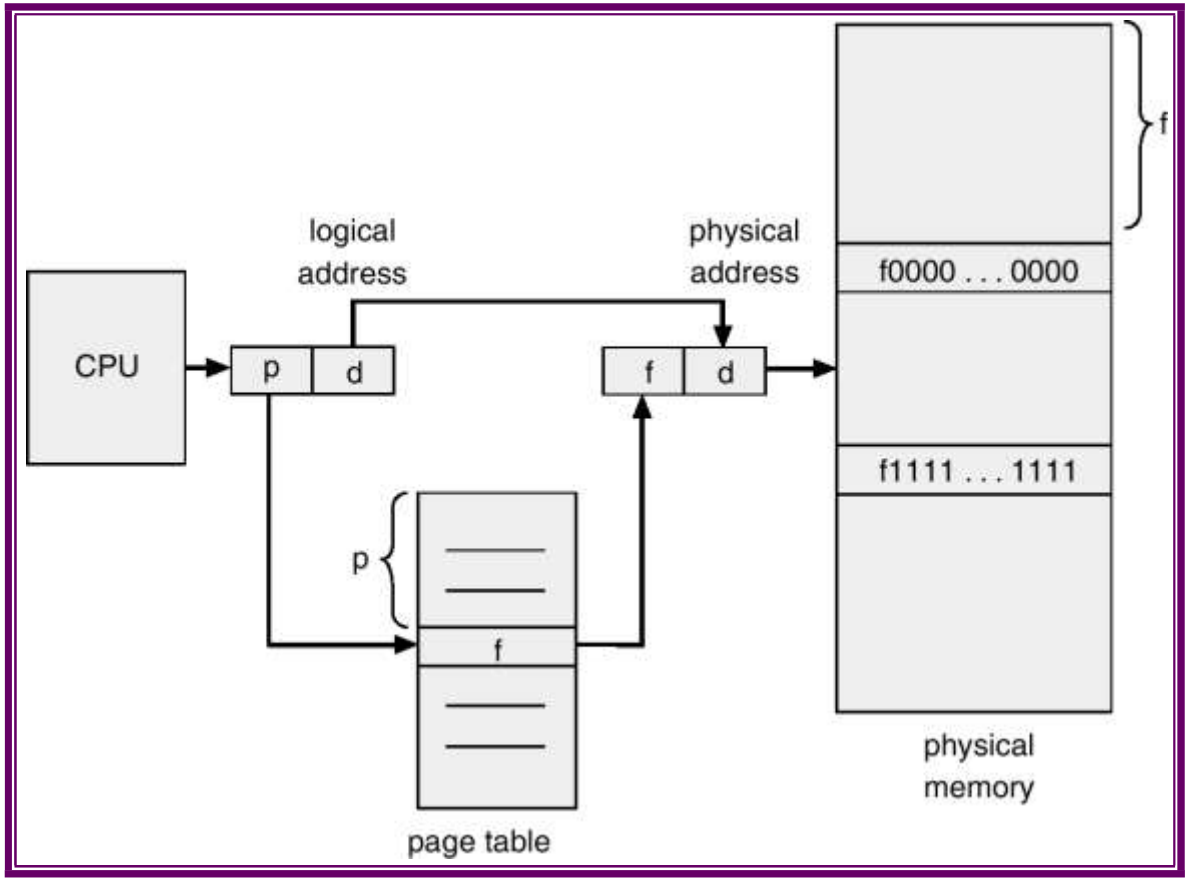
- Multiple-partition allocation
- *Hole* – block of available memory; holes of various size are scattered throughout memory.
- When a process arrives, it is allocated memory from a hole large enough to accommodate it.
- Operating system maintains information about:
  a) allocated partitions b) free partitions (hole)



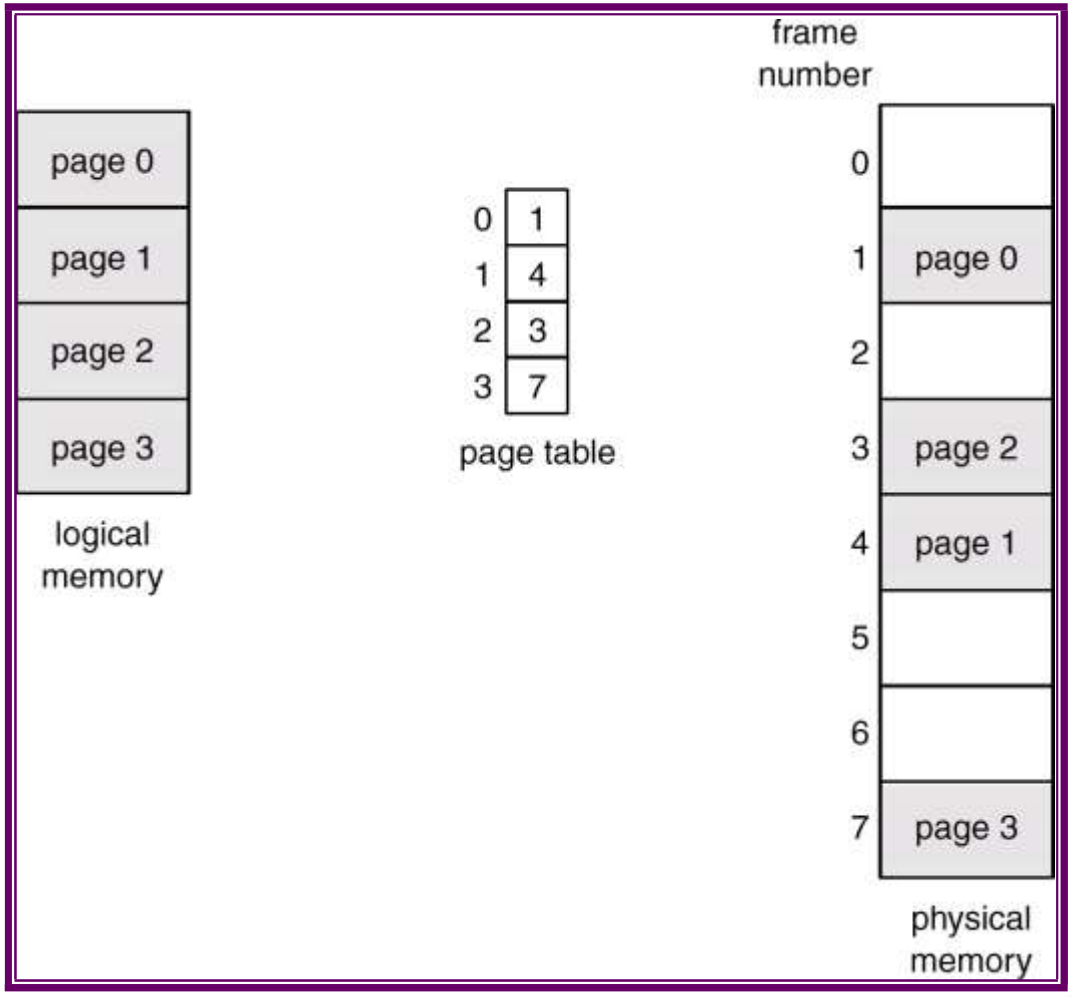
Address Translation Scheme

- 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.

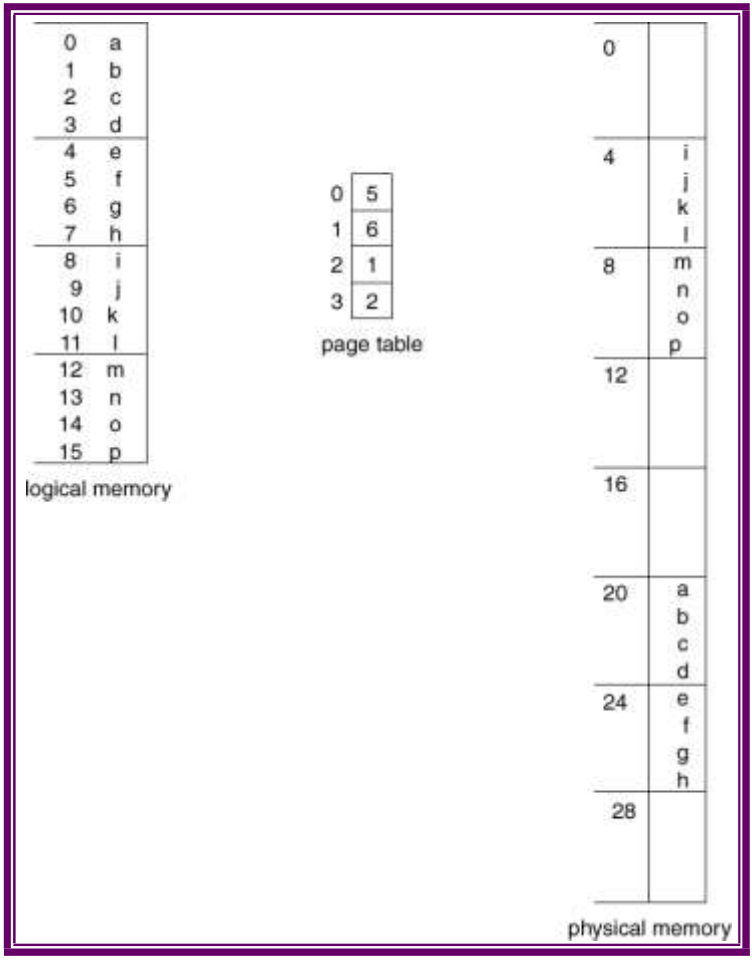
Address Translation Architecture



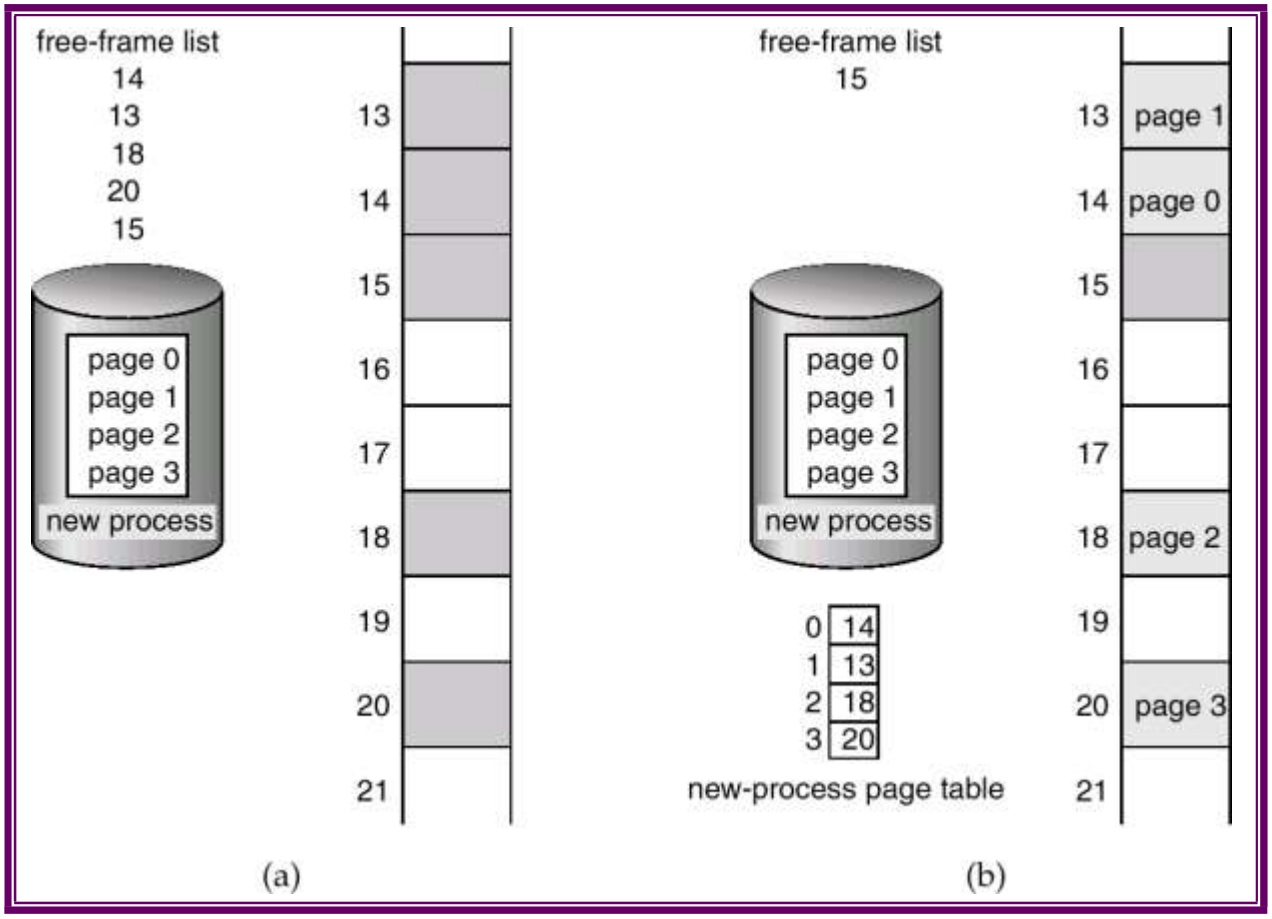
Paging Example



Paging Example



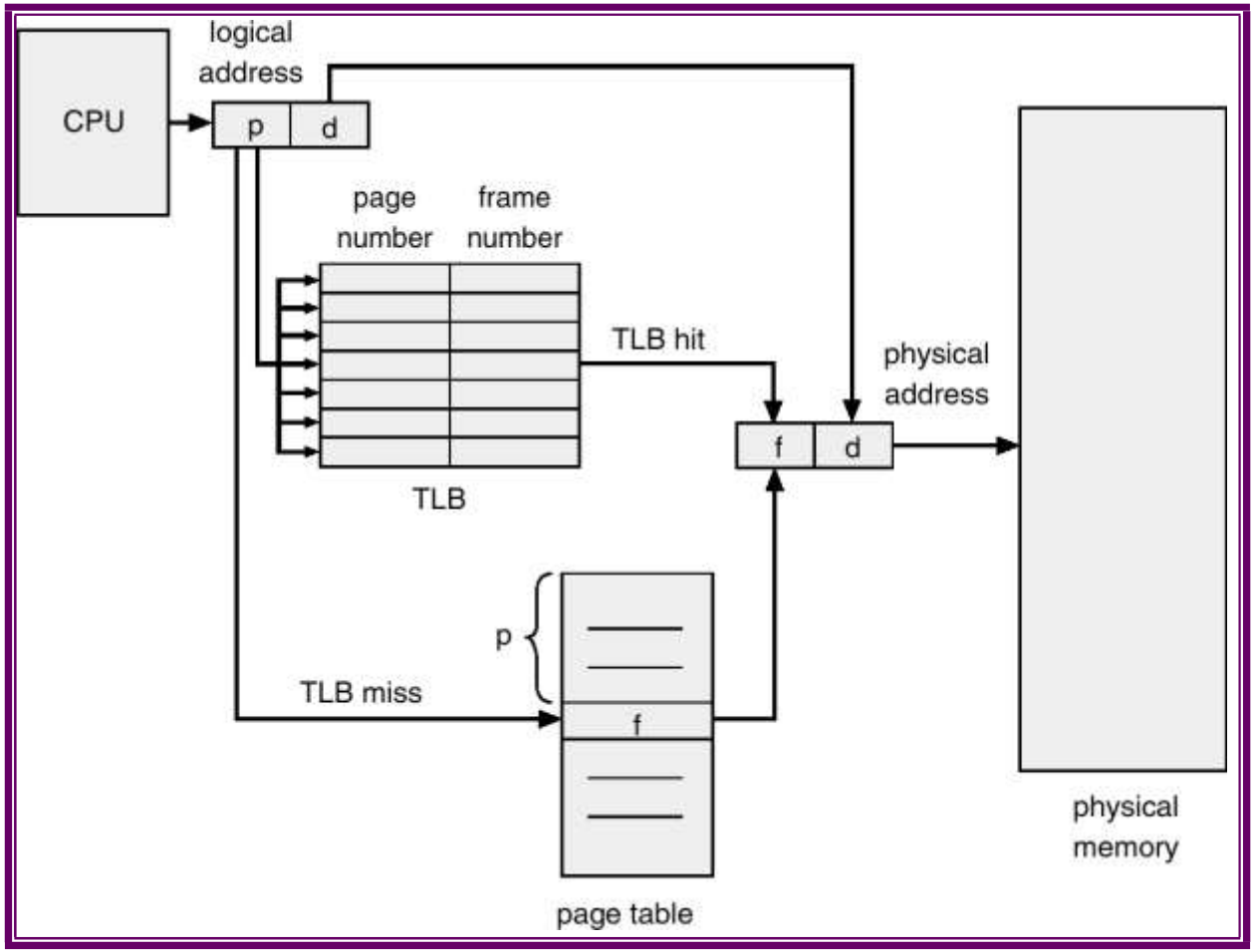
Free Frames



Before allocation

After allocation

Paging Hardware With TLB



Effective Access Time

- Associative Lookup = □ time unit
- Assume memory cycle time is 1 microsecond
- Hit ratio – percentage of times that a page number is found in the associative registers; ration related to number of associative registers.
- Hit ratio = □
- Effective Access Time (EAT)

Memory Protection

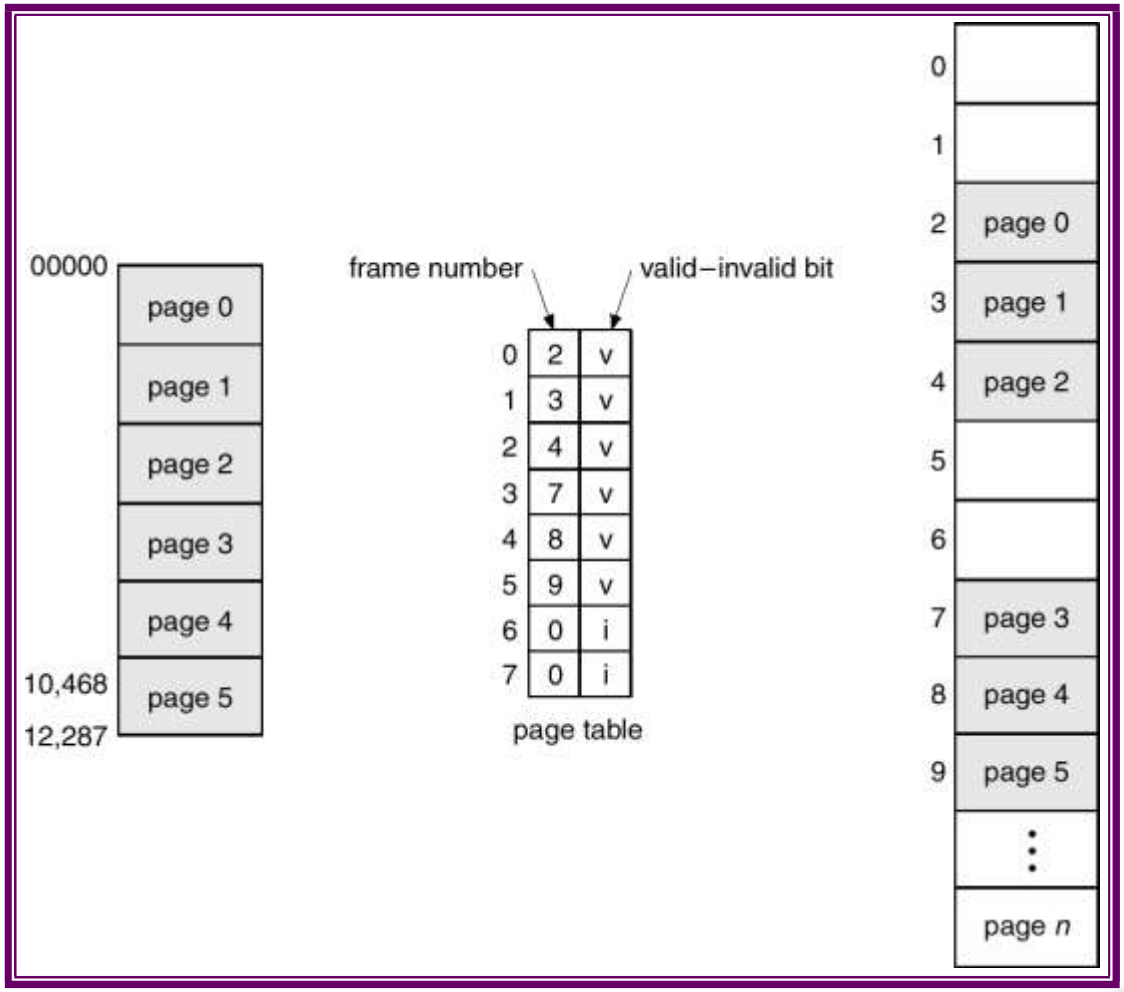
- Memory protection implemented by associating protection bit with each frame.

- *Valid-invalid* bit attached to each entry in the page table:

 “valid” indicates that the associated page is in the process’ logical address space, and is thus a legal page.

 “invalid” indicates that the page is not in the process’ logical address space.

Valid (v) or Invalid (i) Bit In A Page Table



Page Table Structure

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Two-Level Paging Example

- A logical address (on 32-bit machine with 4K page size) is divided into:

a page number consisting of 20bits.

☞ a page offset consisting of 12 bits.

- Since the page table is paged, the page number is further divided into:

 a 10-bit page number.

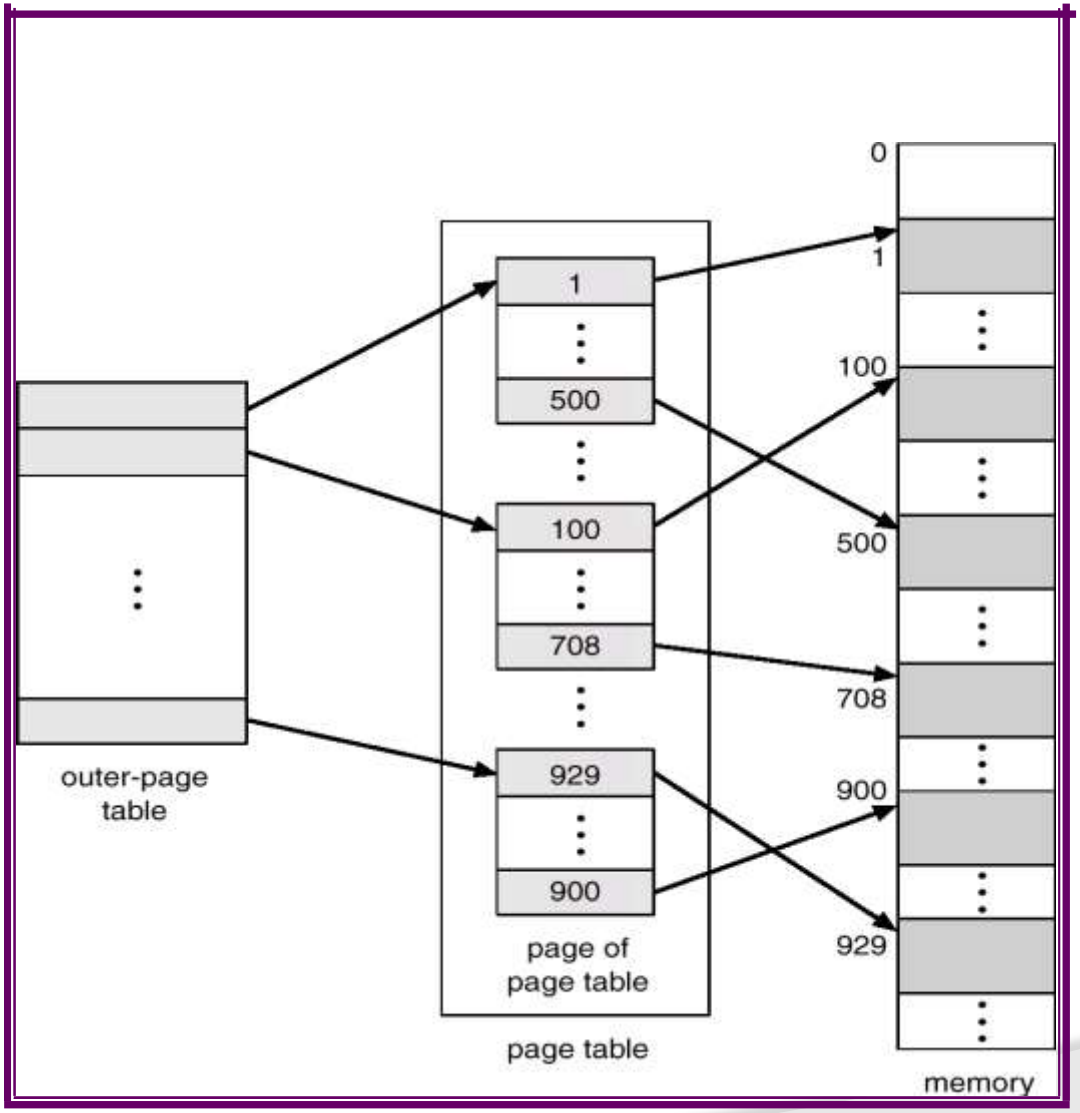
☞ a 10-bit page offset.

- Thus, a logical address is as follows:

page number		page offset
P_i	P_2	d
10	10	12

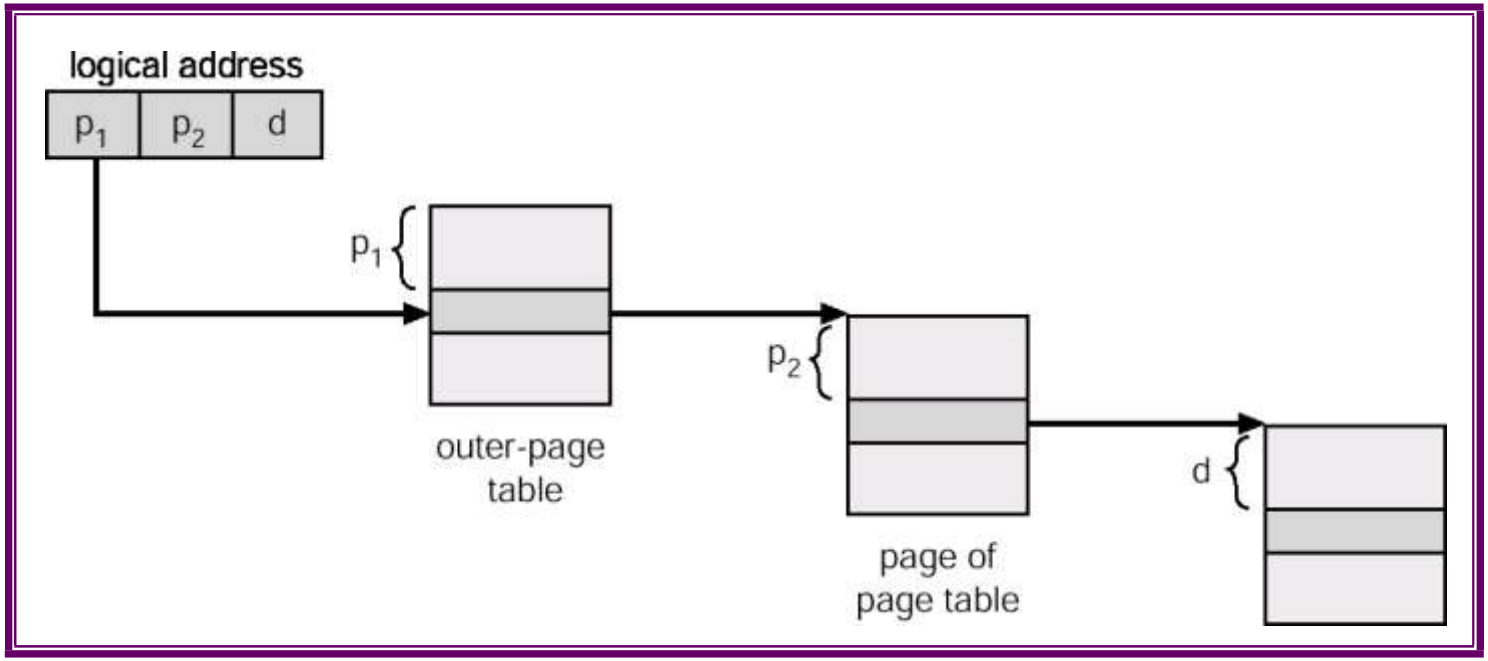
- where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table.

Two-Level Page-Table Scheme



Address-Translation Scheme

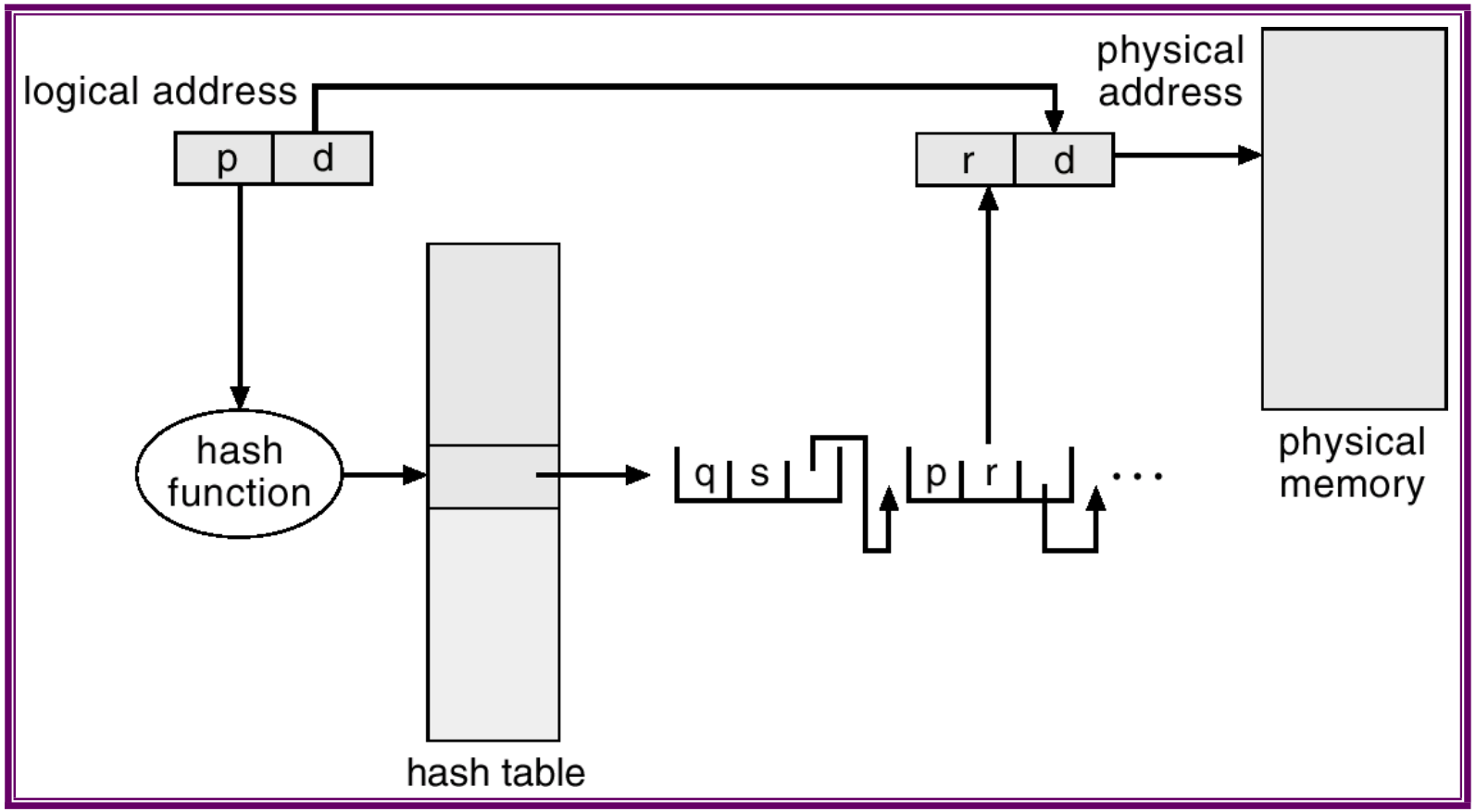
- Address-translation scheme for a two-level 32-bit paging architecture



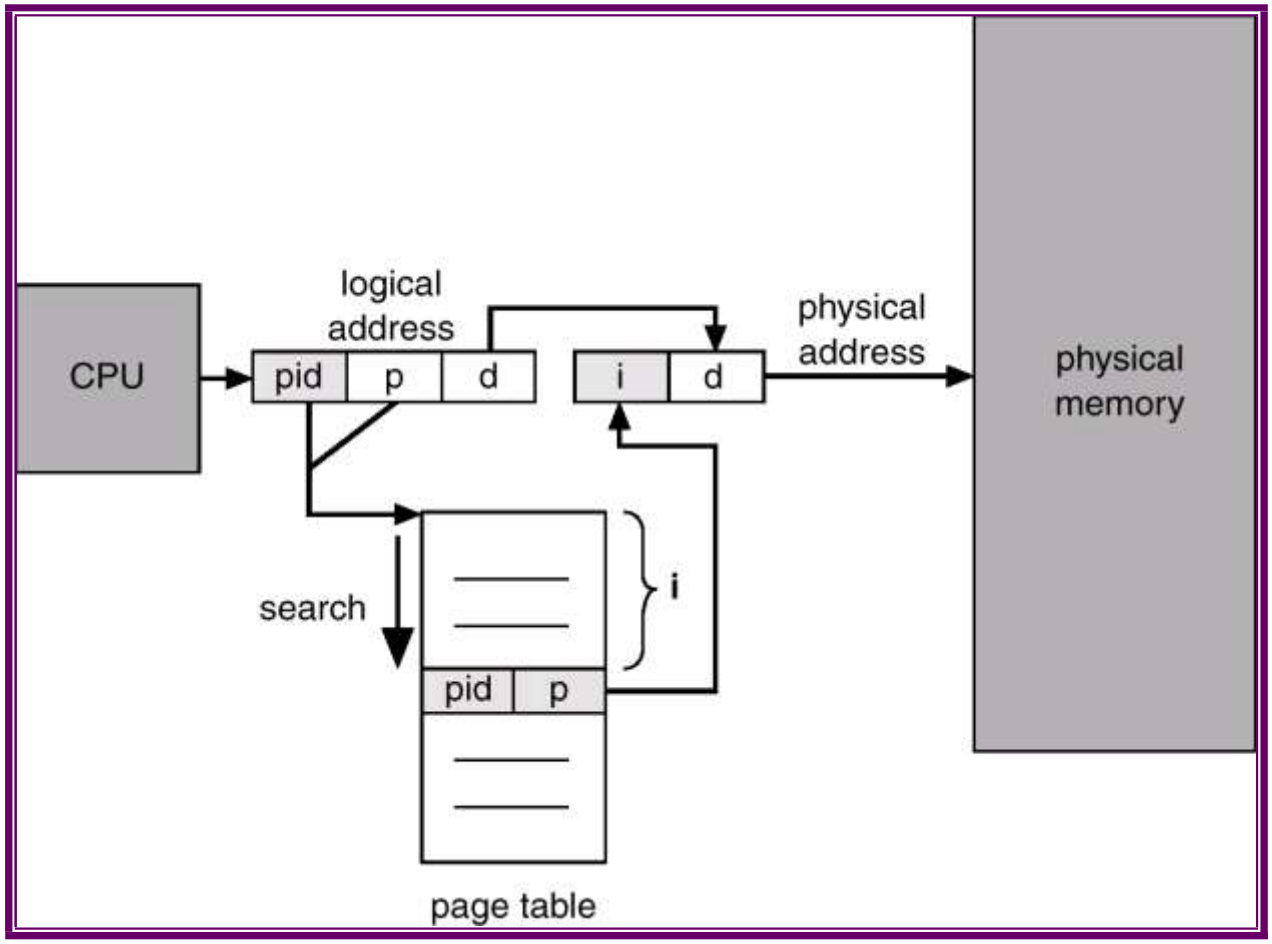
Hashed Page Tables

- Common in address spaces > 32 bits.
- The virtual page number is hashed into a page table. This page table contains a chain of elements hashing to the same location.
- Virtual page numbers are compared in this chain searching for a match.
- If a match is found, the corresponding physical frame is extracted.

Hashed Page Table



Inverted Page Table Architecture



Shared Pages

- Shared code

 One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).

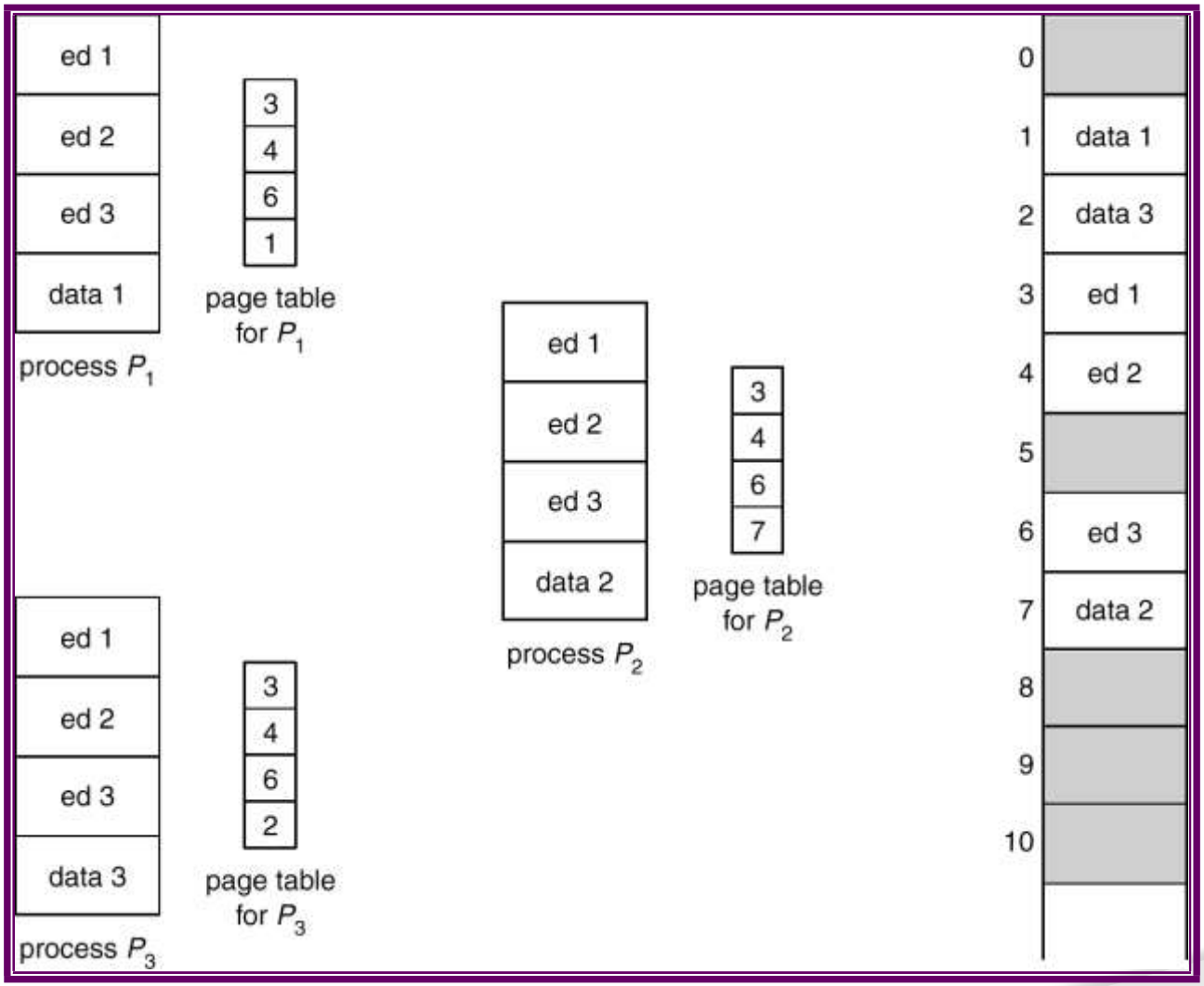
 Shared code must appear in same location in the logical address space of all processes.

- Private code and data

 Each process keeps a separate copy of the code and data.

 The pages for the private code and data can appear anywhere in the logical address space.

Shared Pages Example



Segmentation

- Memory-management scheme that supports user view of memory.

- A program is a collection of segments. A segment is a logical unit such as:

main program, procedure,

function,

method,

object,

local variables,

global variables,

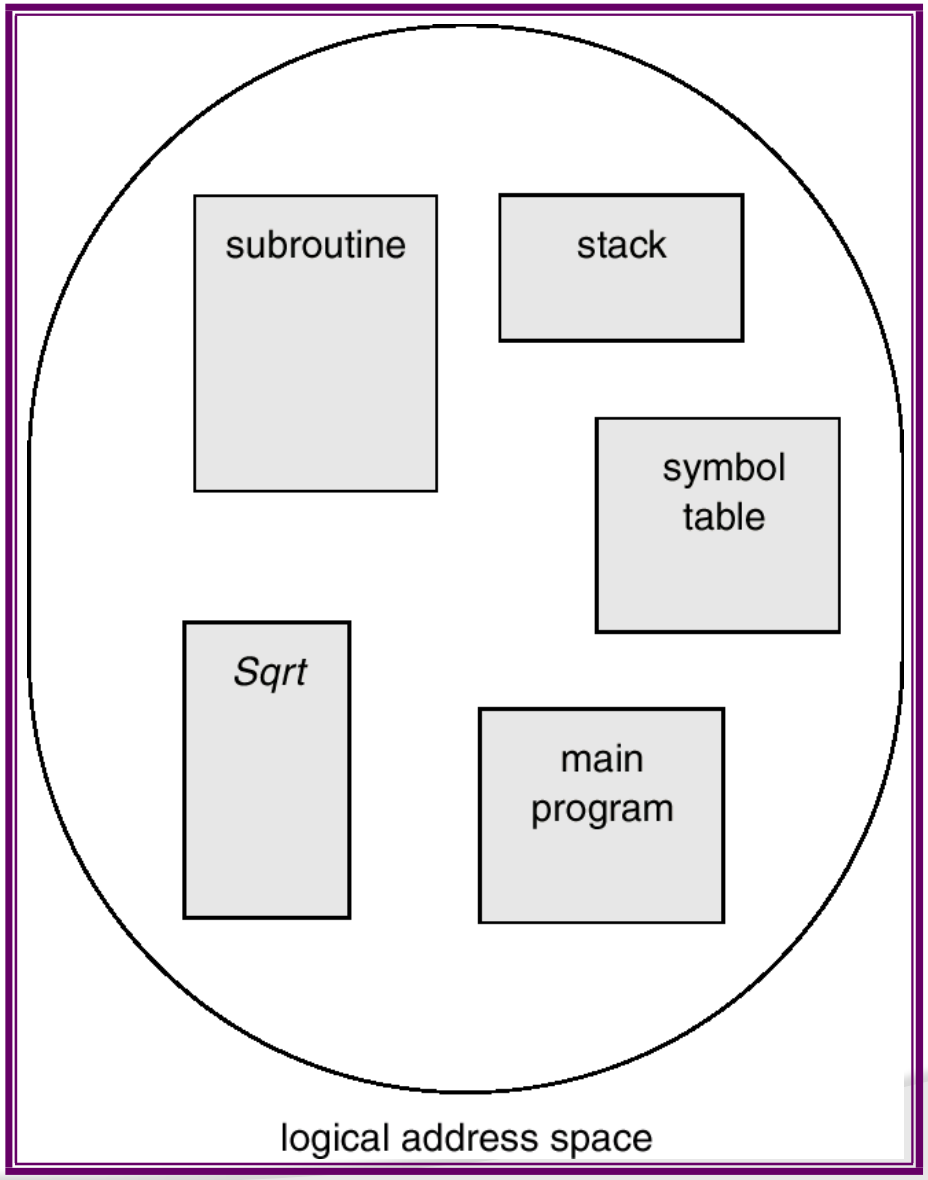
common block,

stack,

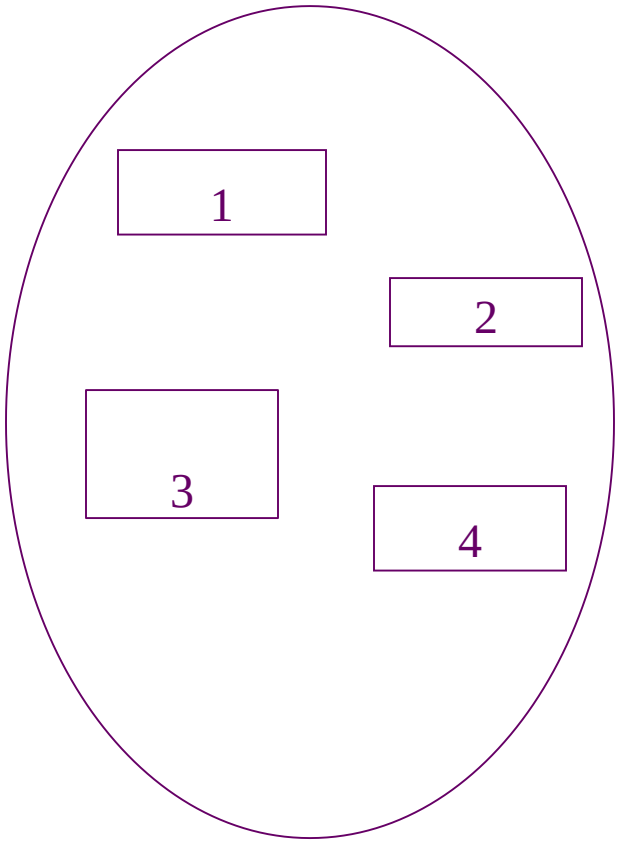
symbol table,

arrays

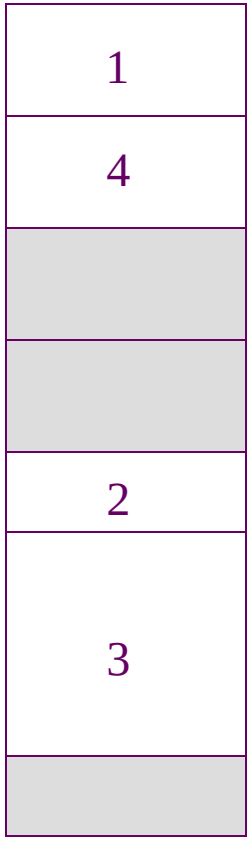
User's View of a Program



Logical View of Segmentation



user space



physical memory space

Segmentation Architecture

- Logical address consists of a two tuple:

<segment-number, offset>,

- *Segment table* – maps two-dimensional physical addresses; each table entry has:
  - base – contains the starting physical address where the segments reside in memory.
  - *limit* – specifies the length of the segment.

- *Segment-table base register (STBR)* points to the segment table's location in memory.

- *Segment-table length register (STLR)* indicates number of STLR.

Segmentation Architecture (Cont.)

- Protection. With each entry in segment table associate:

 validation bit = 0 illegalsegment

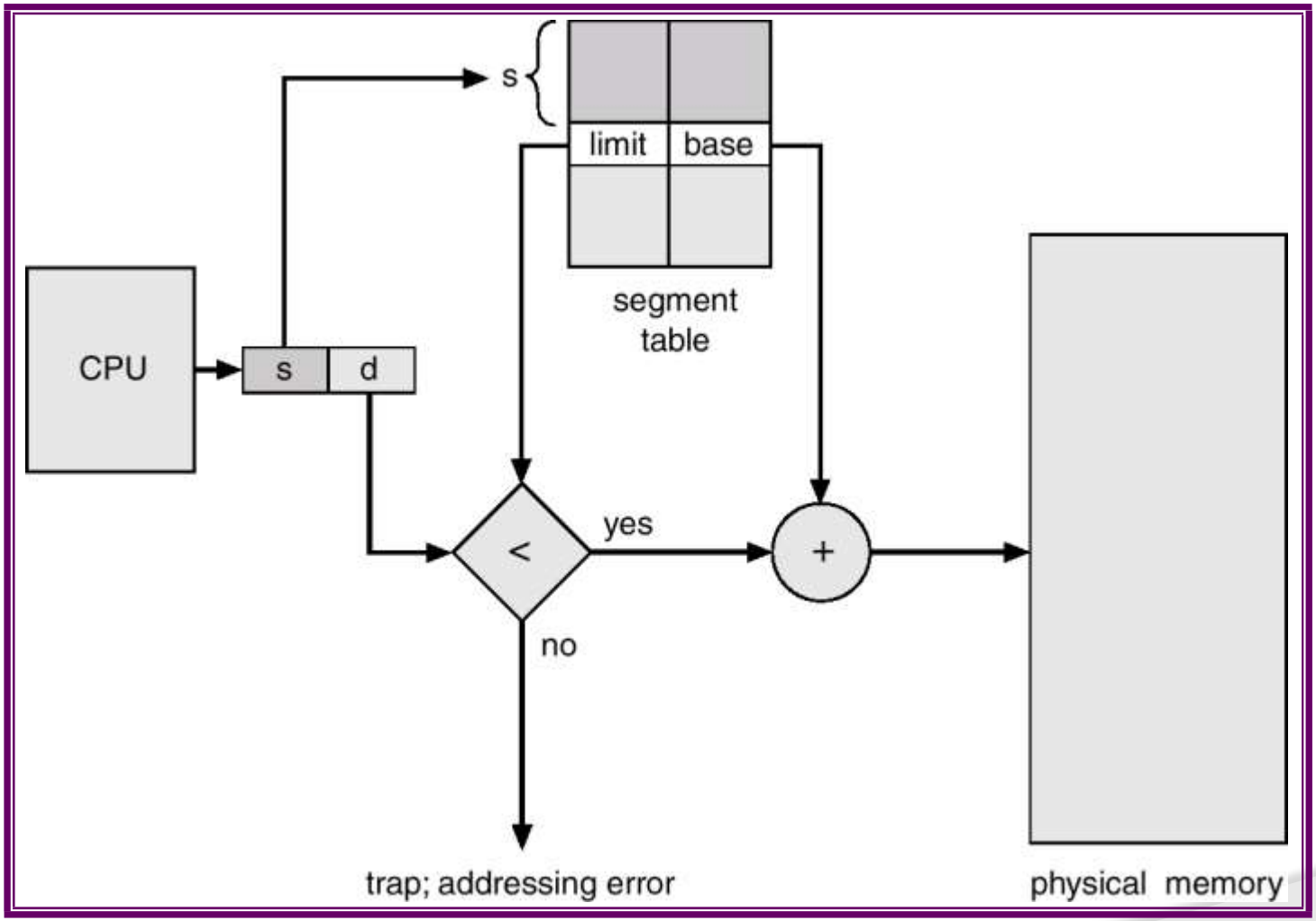
 read/write/executeprivileges

- Protection bits associated with segments; code sharing occurs at segment level.

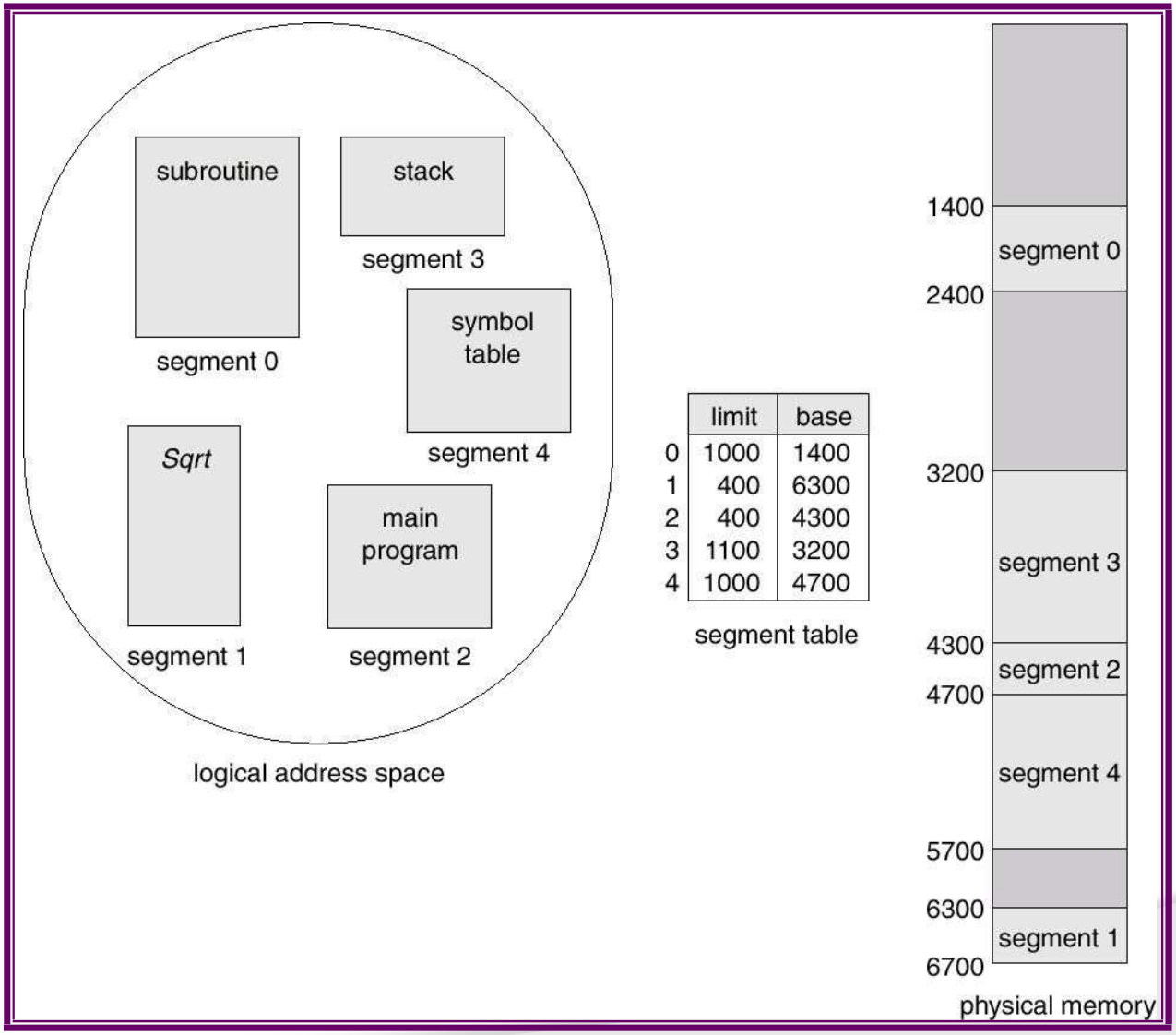
- Since segments vary in length, memory allocation is a dynamic storage-allocation problem.

- A segmentation example is shown in the following diagram

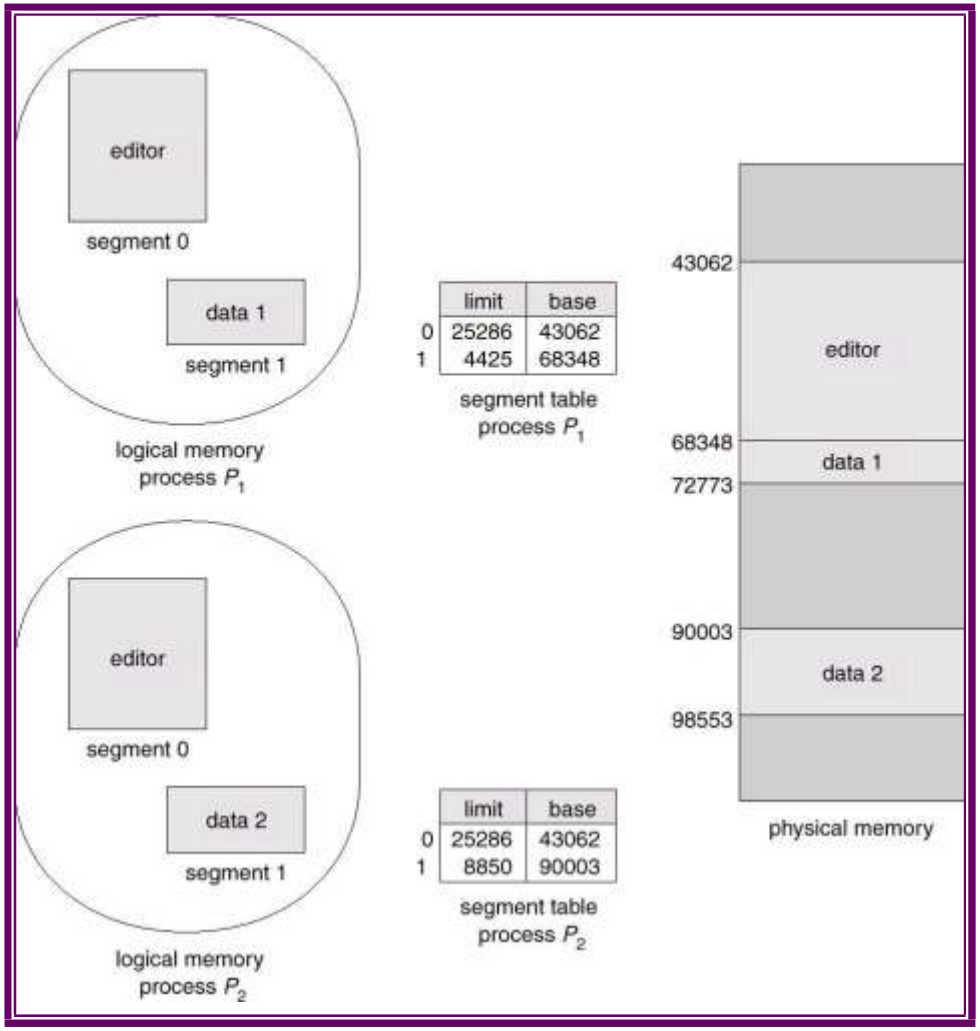
Segmentation Hardware



Example of Segmentation



Sharing of Segments

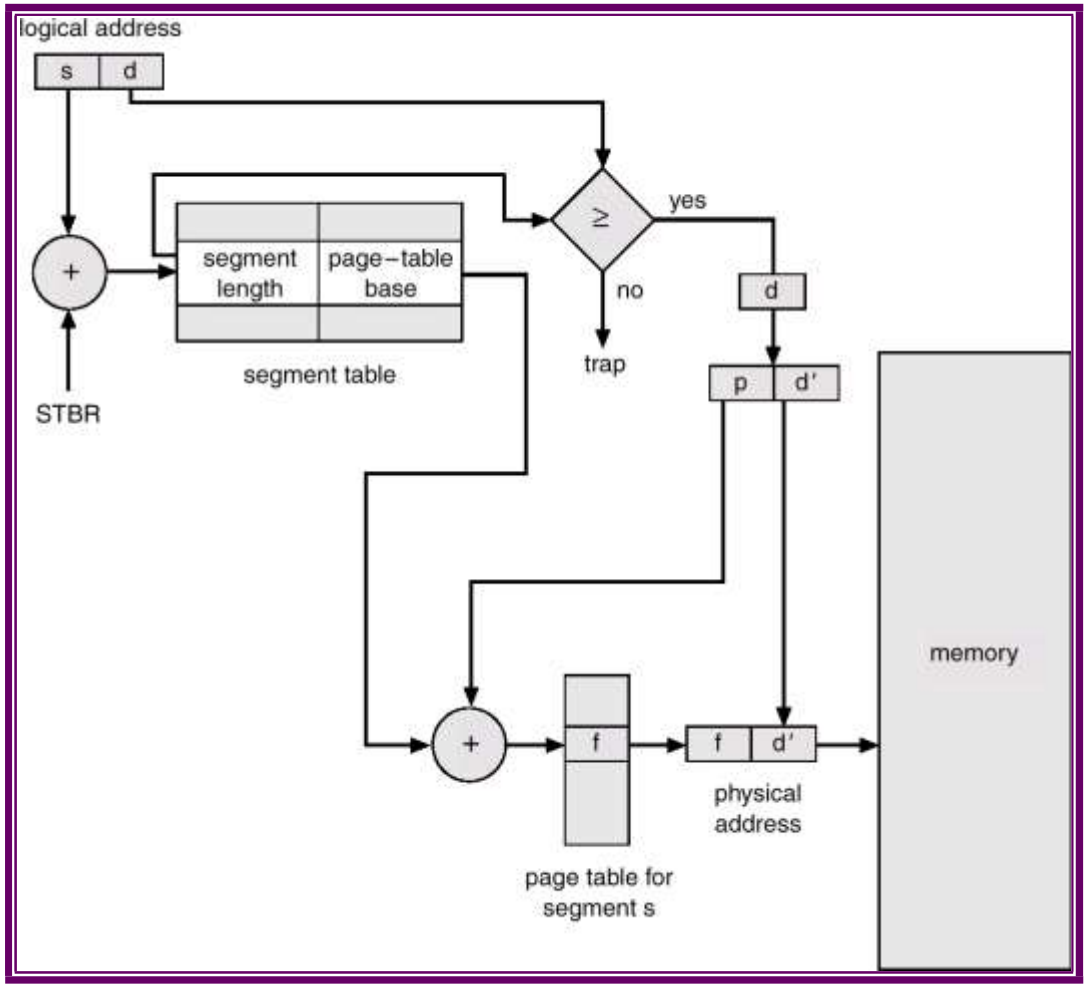


Segmentation with Paging – MULTICS

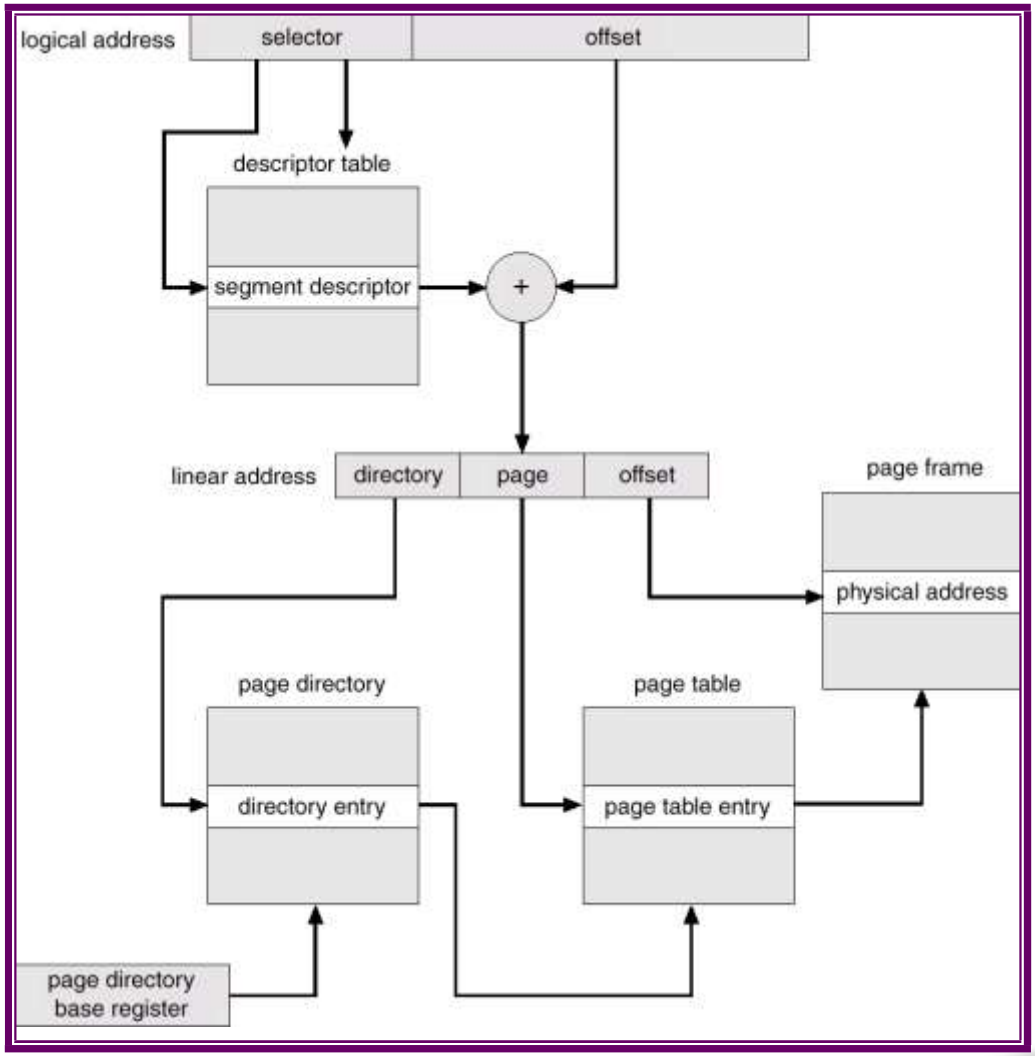
- The MULTICS system solved problems of external fragmentation and lengthy search times by paging the segments.

- Solution differs from pure segmentation in that the segment-table entry contains not the base address of the segment, but rather the base address of a *page table* for this segment.

MULTICS Address Translation Scheme



Intel 30386 Address Translation

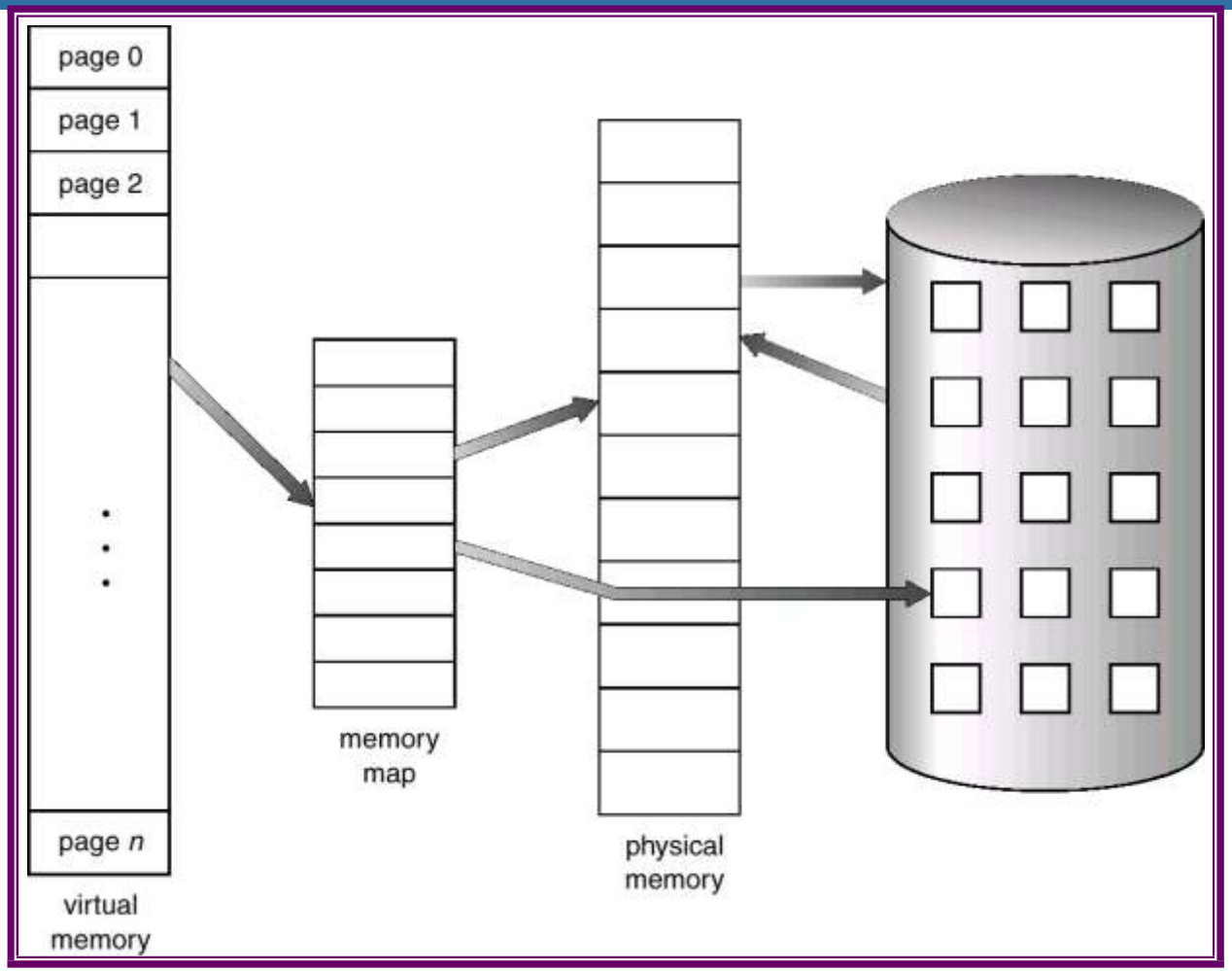


Background

- **Virtual memory** – separation of user logical memory from physical memory.
  - Only part of the program needs to be in memory for execution.
  - Logical address space can therefore be much larger than physical address space.
  - Allows address spaces to be shared by several processes.
  - Allows for more efficient process creation.

- Virtual memory can be implemented via:
  - Demand paging
  - Demand segmentation

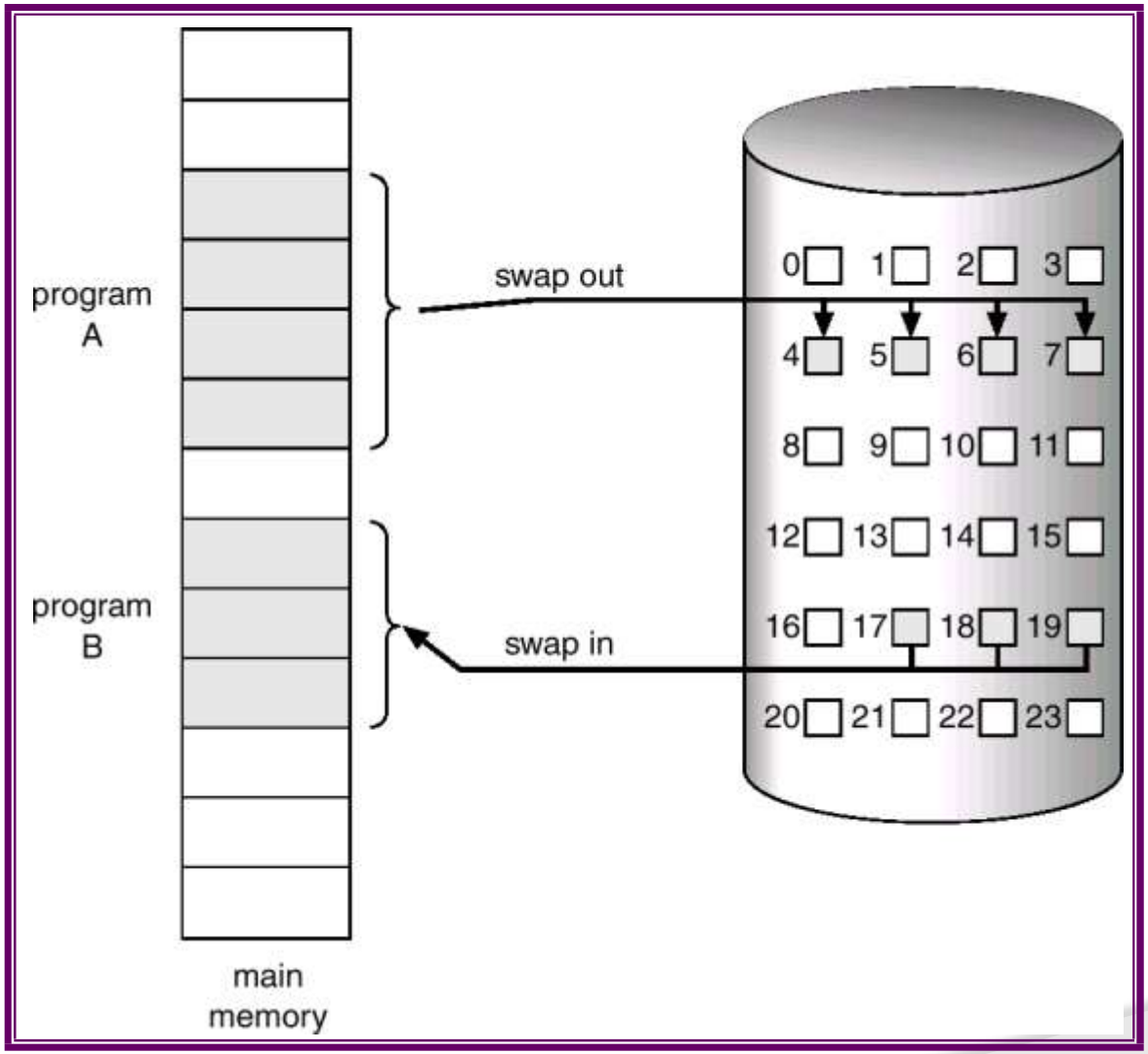
Virtual Memory That is Larger Than Physical Memory



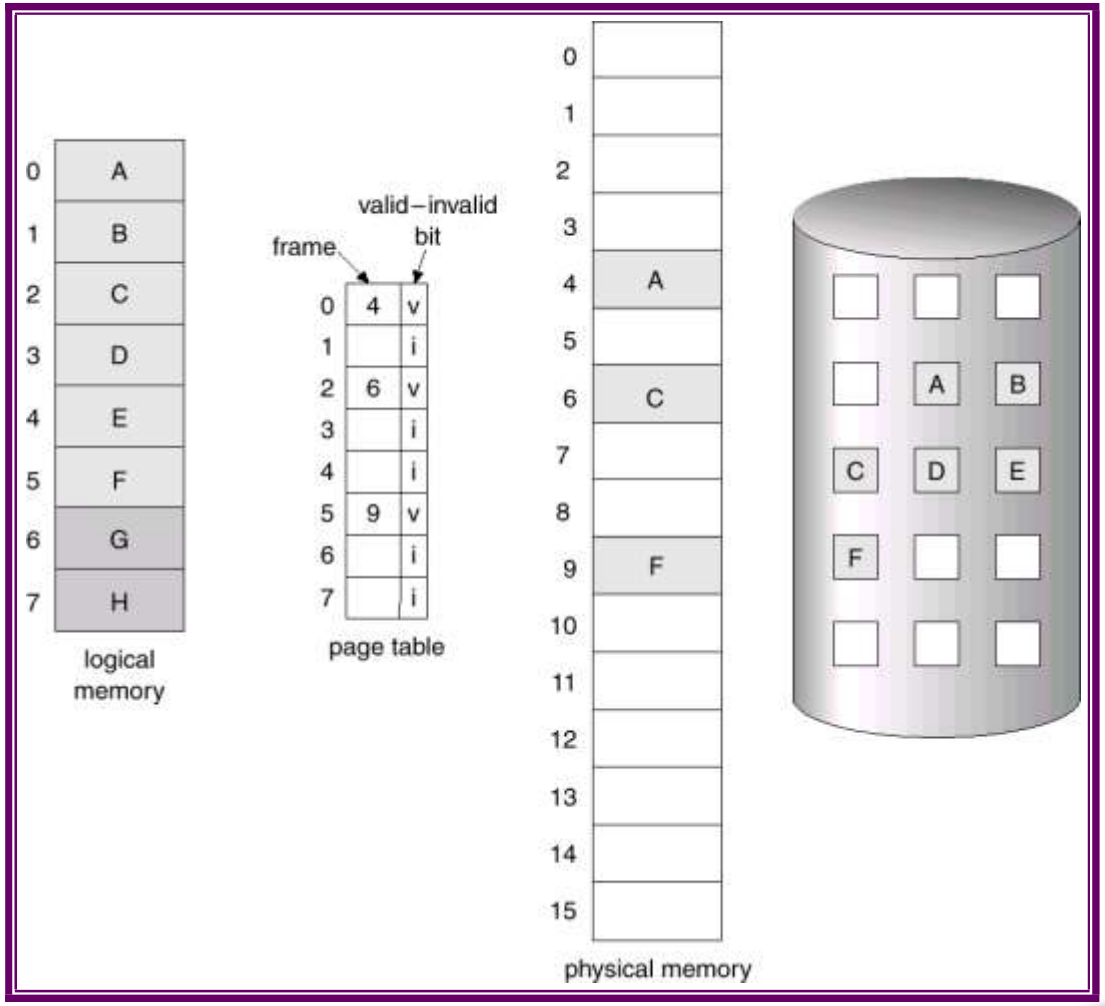
Demand Paging

- Bring a page into memory only when it is needed.
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users
- Page is needed reference to it
  - invalid reference abort
  - not-in-memory bring to memory

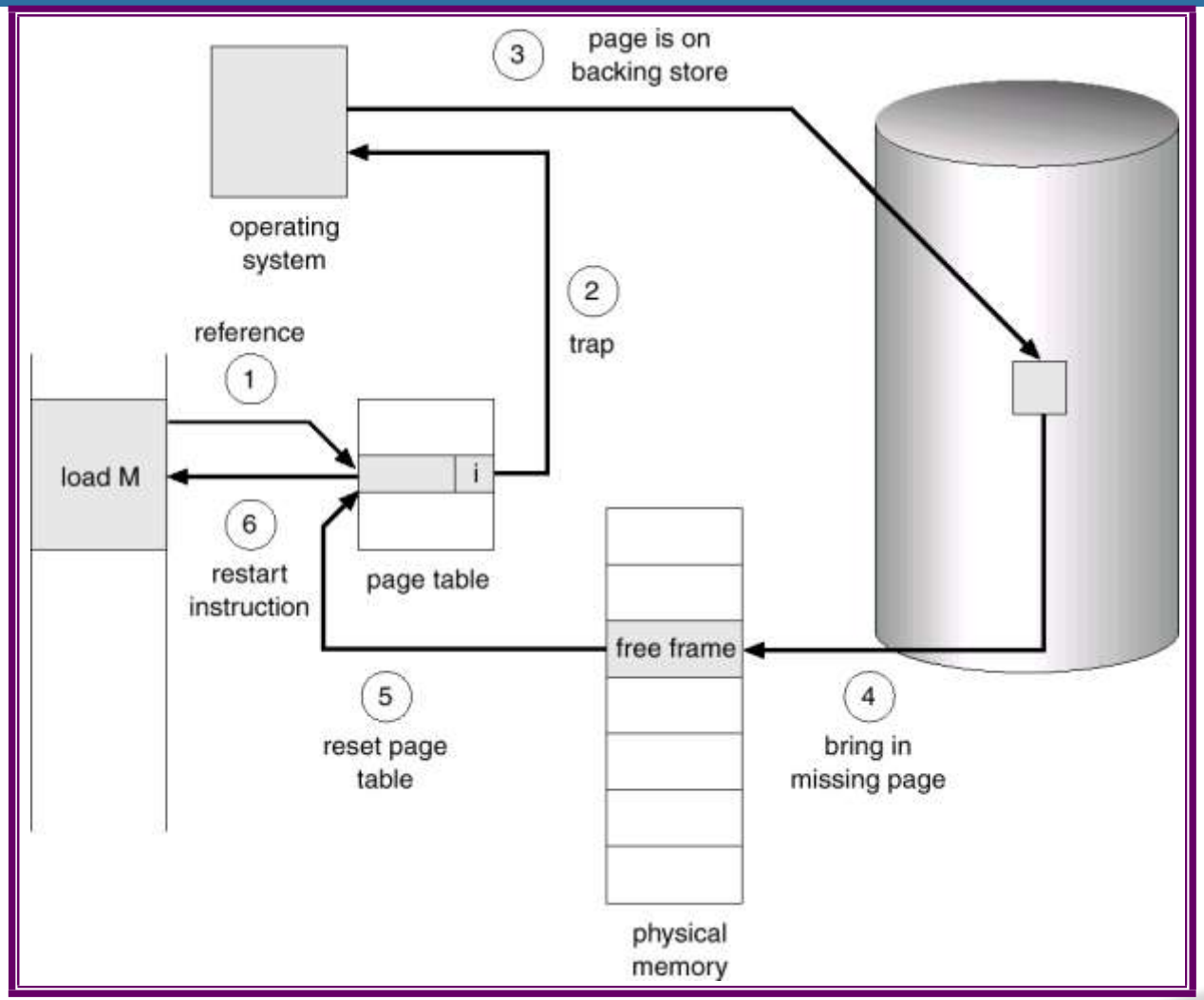
Transfer of a Paged Memory to Contiguous Disk Space



Page Table When Some Pages Are Not in Main Memory



Steps in Handling a Page Fault



What happens if there is no free frame?

- Page replacement – find some page in memory, but not really in use, swap it out.
  - algorithm
  - performance – want an algorithm which will result in minimum number of page faults.

- Same page may be brought into memory several times.

Demand Paging Example

- Memory access time = 1 microsecond
- 50% of the time the page that is being replaced has been modified and therefore needs to be swapped out.
- Swap Page Time = 10 m sec = 10,000 m sec

$$EAT = (1 - p) \times 1 + p\ (15000)$$

$$1 + 15000P \qquad \text{(in m sec)}$$

Process Creation

- Virtual memory allows other benefits during process creation:
  - Copy-on-Write
  - Memory-Mapped Files

Copy-on-Write

- Copy-on-Write (COW) allows both parent and child processes to initially *share* the same pages in memory.

If either process modifies a shared page, only then is the page copied.

- COW allows more efficient process creation as only modified pages are copied.

- Free pages are allocated from a *pool* of zeroed-out pages.

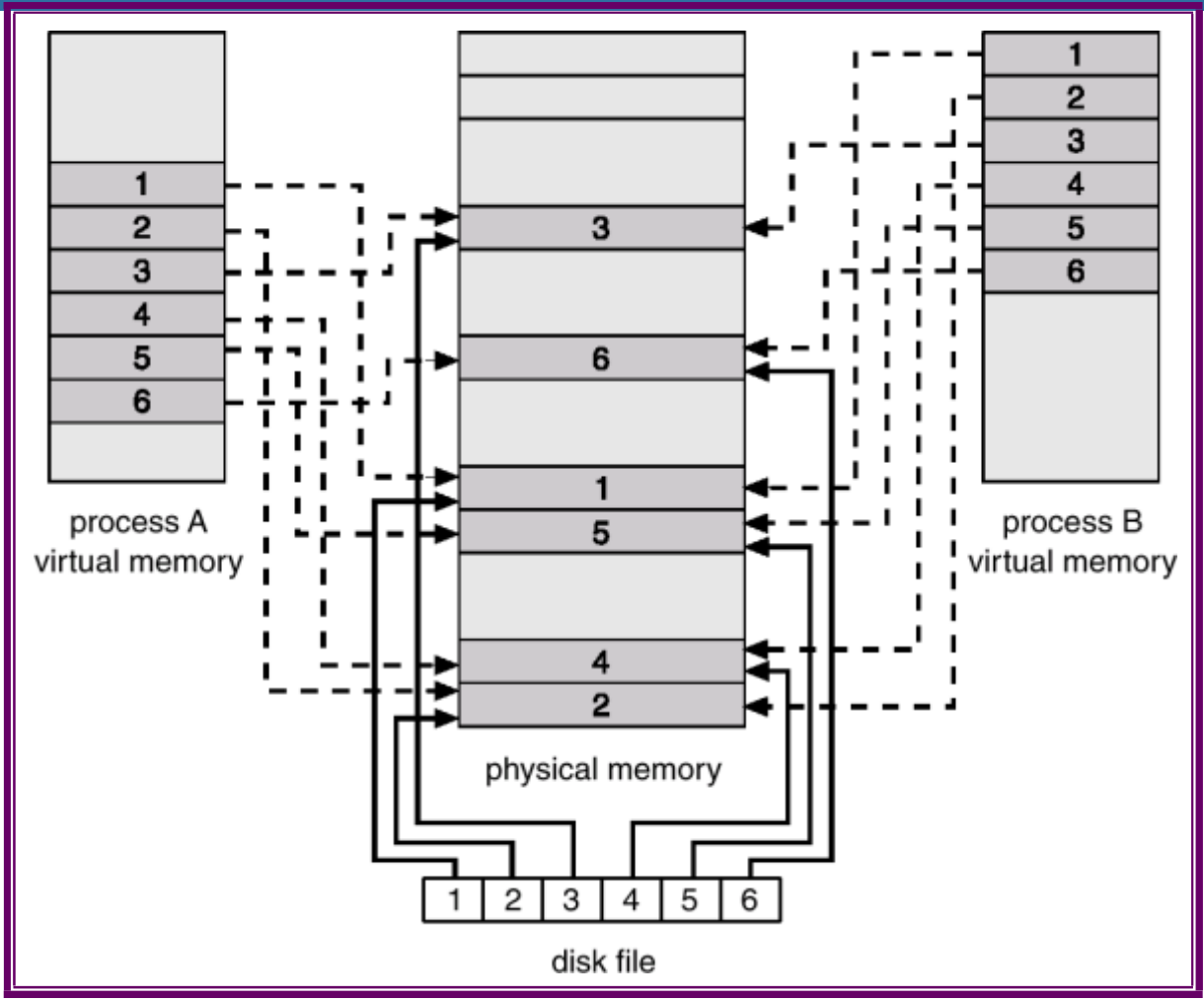
Memory-Mapped Files

- Memory-mapped file I/O allows file I/O to be treated as routine memory access by *mapping* a disk block to a page in memory.

- A file is initially read using demand paging. A page-sized portion of the file is read from the file system into a physical page. Subsequent reads/writes to/from the file are treated as ordinary memory accesses.

- Simplifies file access by treating file I/O through memory rather than **read() write()** system calls.

Memory Mapped Files



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 completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

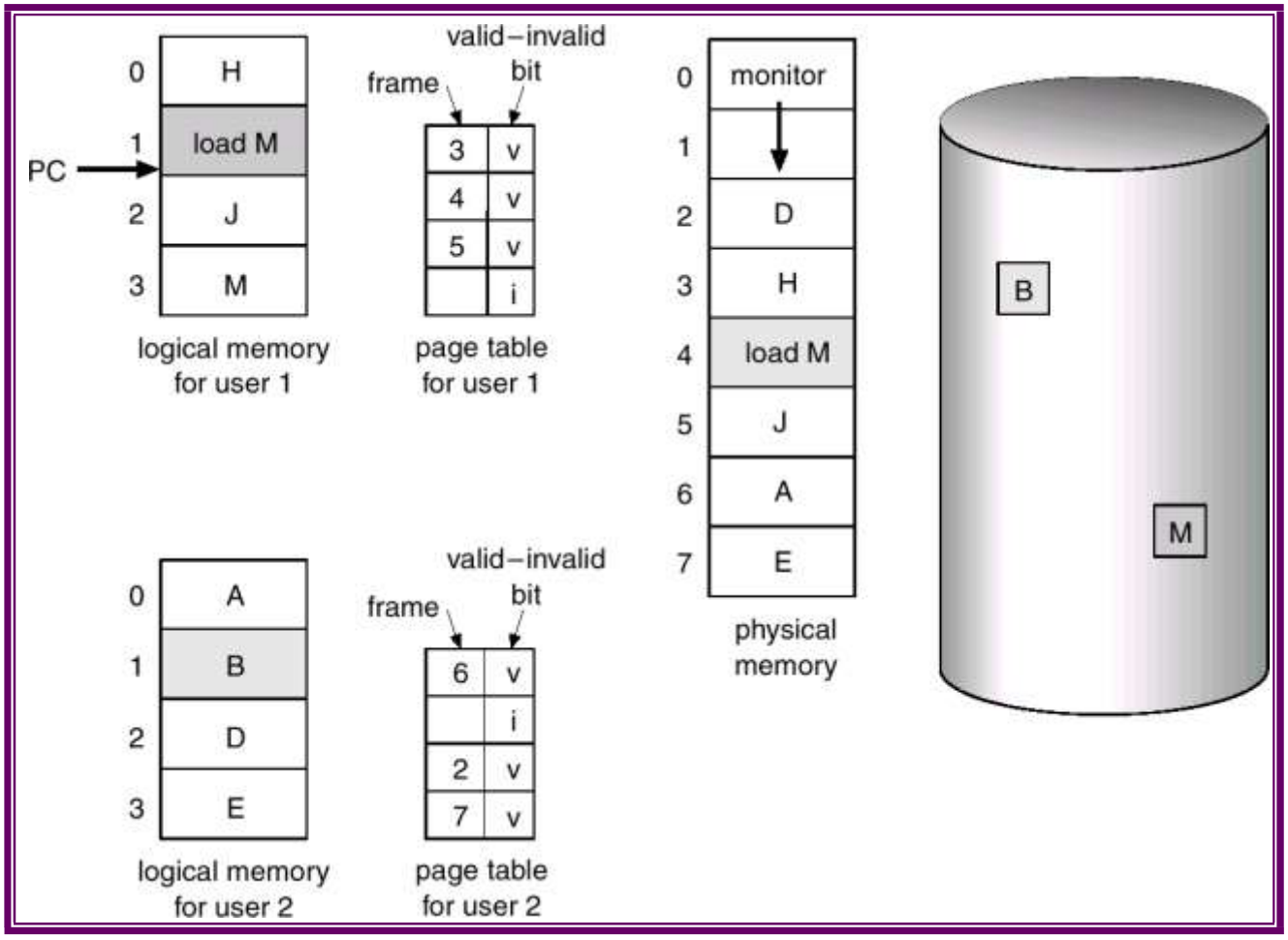
Page Replacement

- Prevent over-allocation of memory by modifying page-fault service routine to include page replacement.

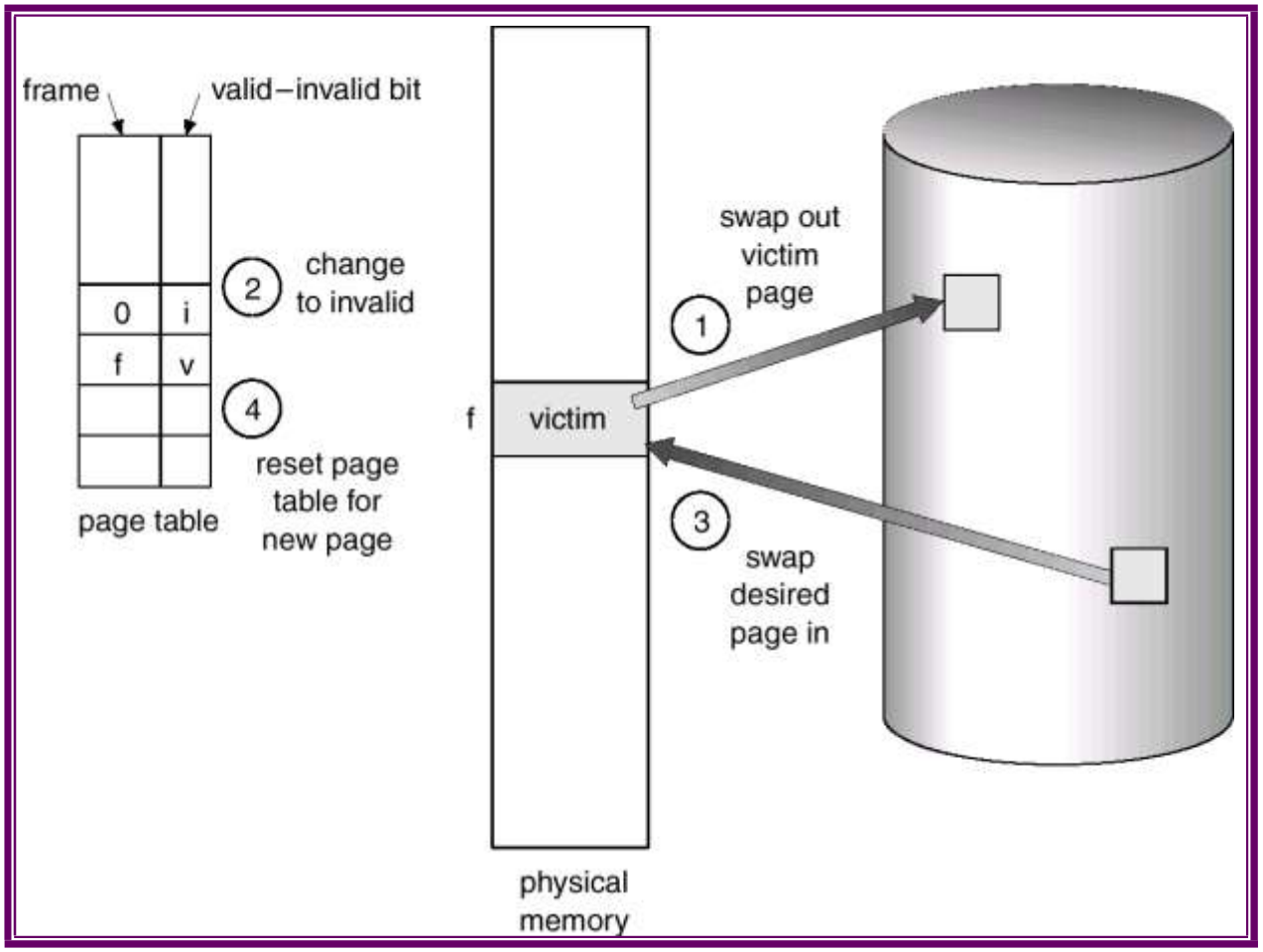
- Use *modify* (*dirty*) *bit* to reduce overhead of page transfers – only modified pages are written to disk.

- Page replacement completes separation between logical memory and physical memory – large virtual memory can be provided on a smaller physical memory.

Need For Page Replacement



Page Replacement

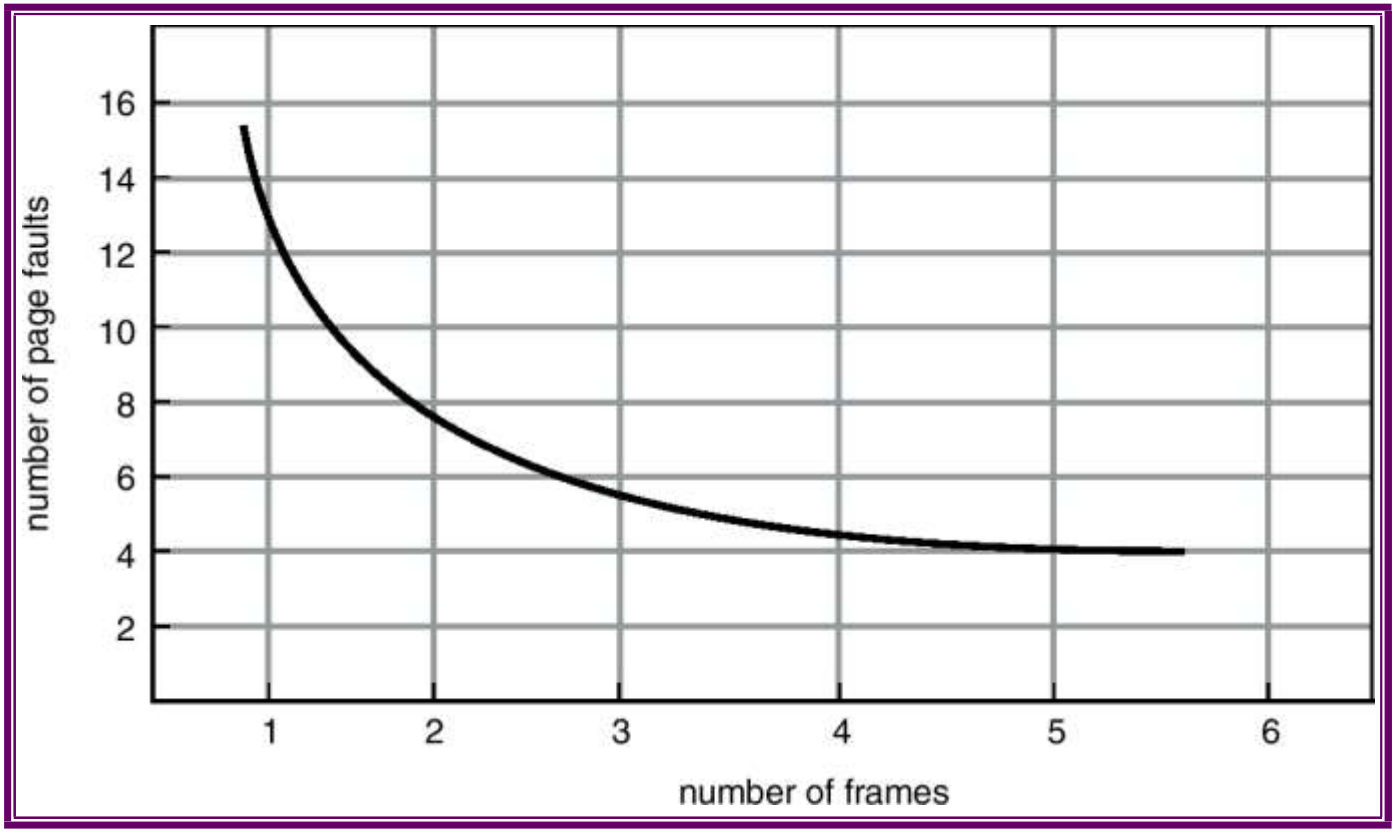


Page Replacement Algorithms

- Want lowest page-fault rate.
- Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string.
- In all our examples, the reference string is

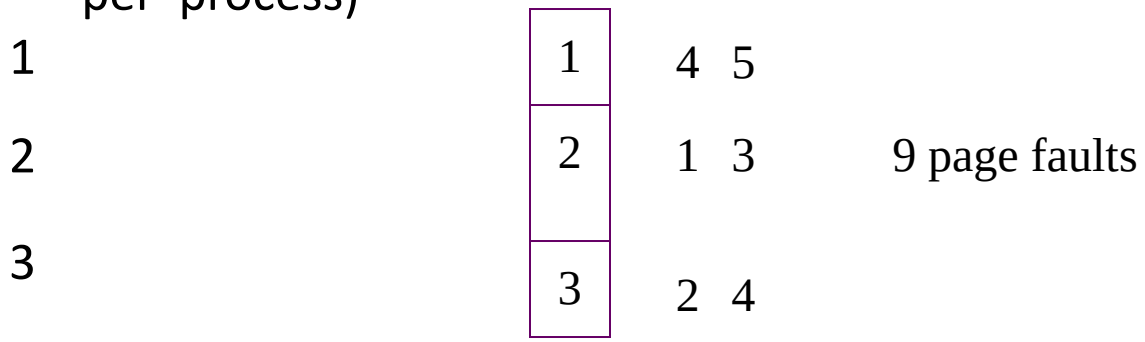
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

Graph of Page Faults Versus The Number of Frames



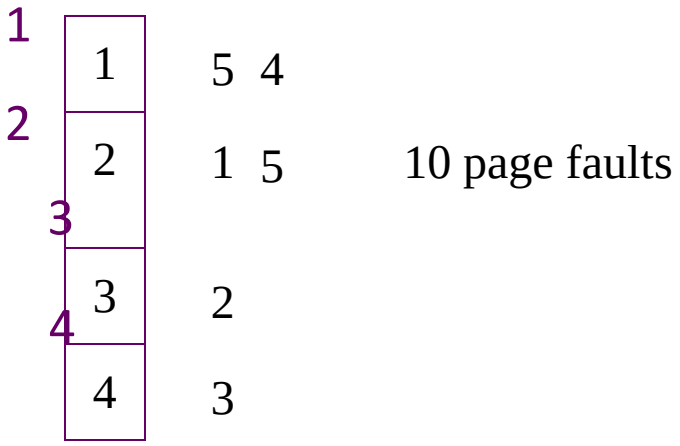
First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



9 page faults

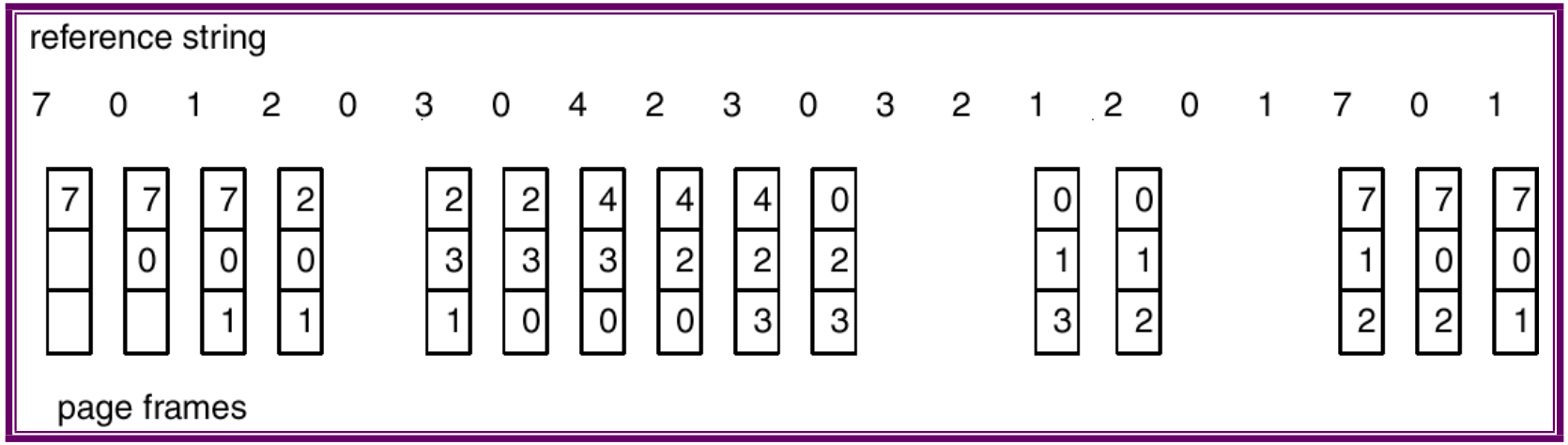
- 4 frames



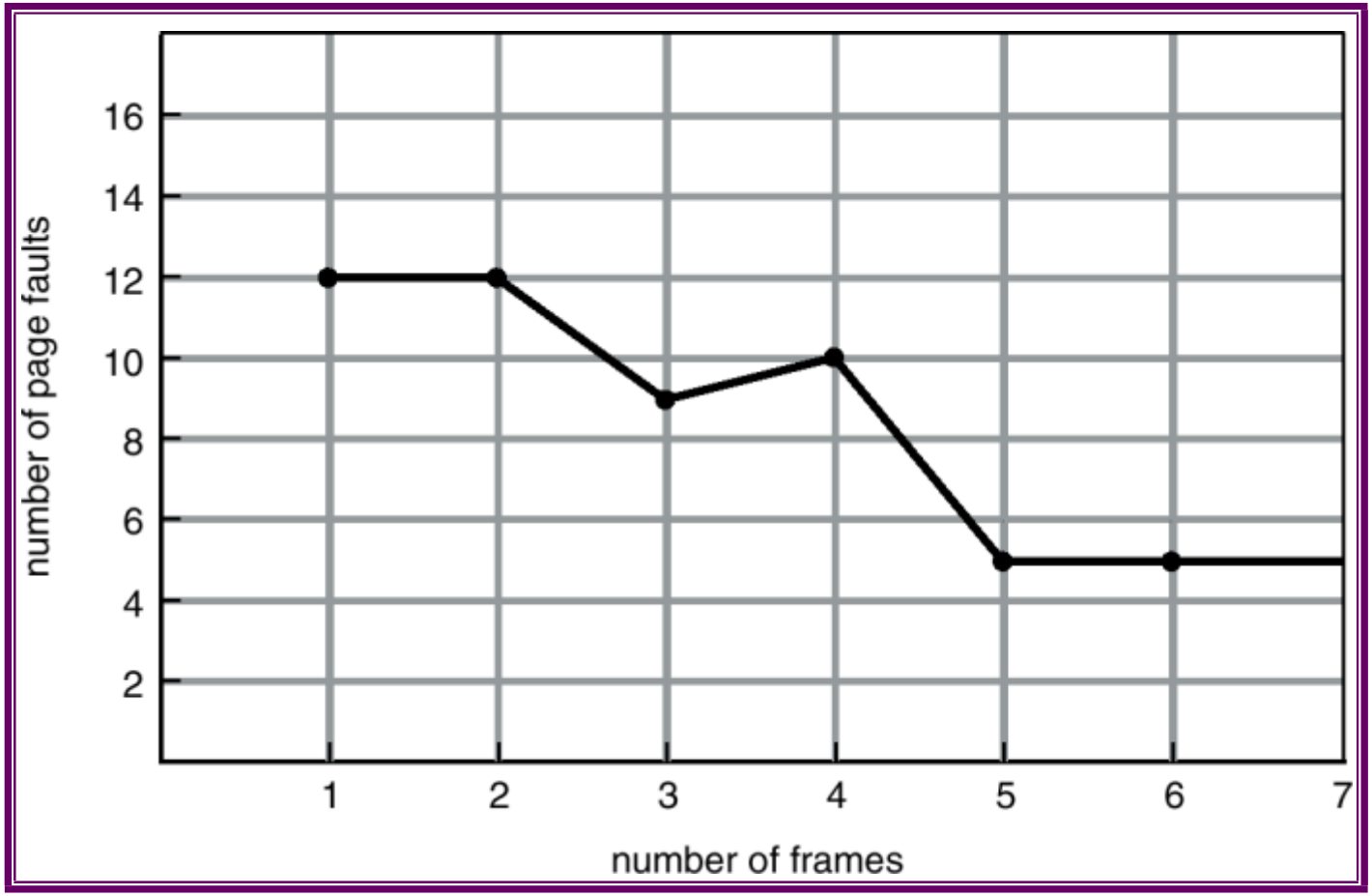
10 page faults

- FIFO Replacement – Belady's Anomaly
  - more frames less page faults

FIFO Page Replacement



FIFO Illustrating Belady's Anamoly



Optimal Algorithm

- Replace page that will not be used for longest period of time.
- 4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4,5

4

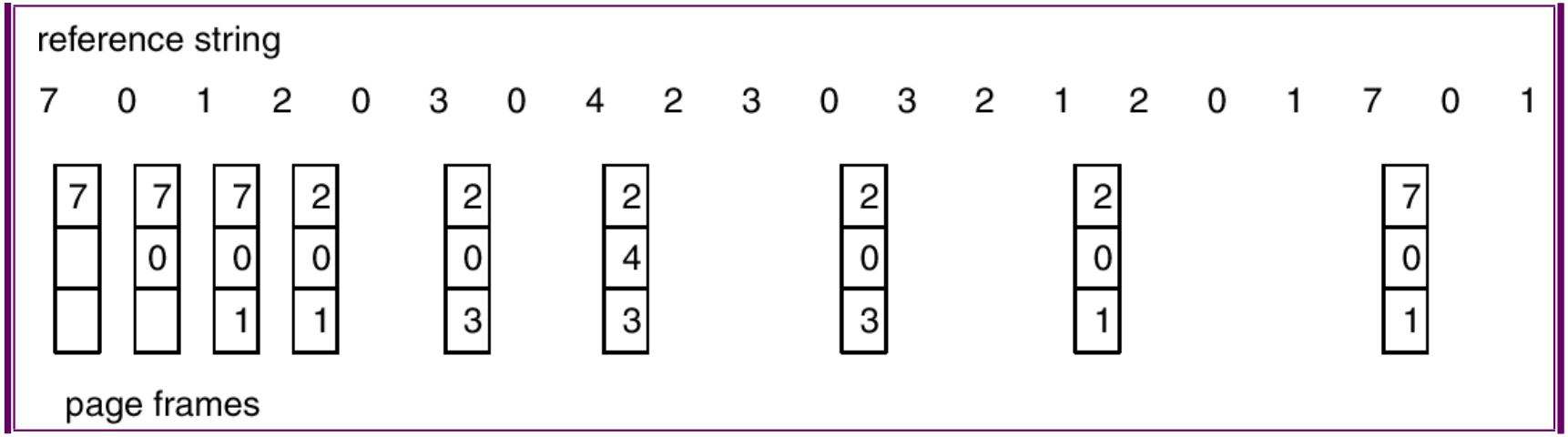
6 page faults

1
2
3
4

5

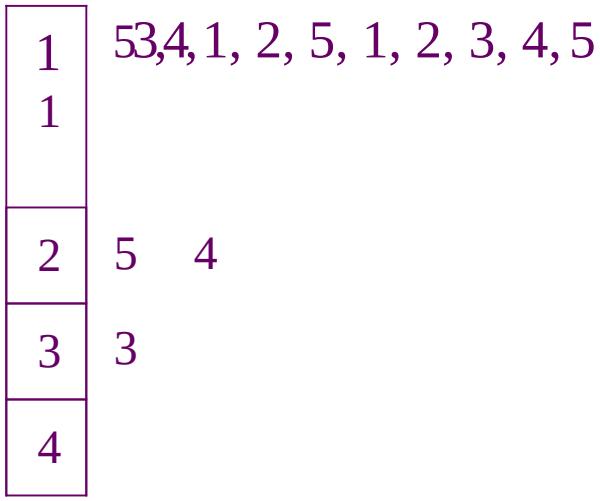
- How do you know this?
- Used for measuring how well your algorithm performs.

Optimal Page Replacement



Least Recently Used (LRU) Algorithm

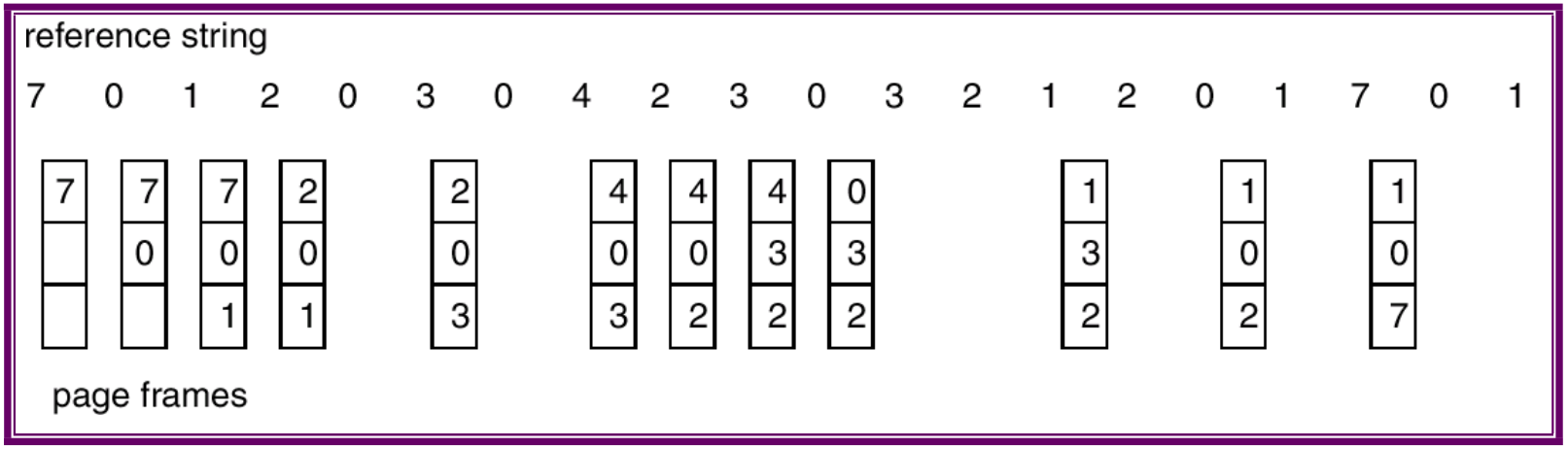
- Reference string:



  - Counter implementation

- Every page entry has a counter; every time page is referenced through this entry, copy the clock intothe counter.
- When a page needs to be changed, look at the counters to determine which are to change.

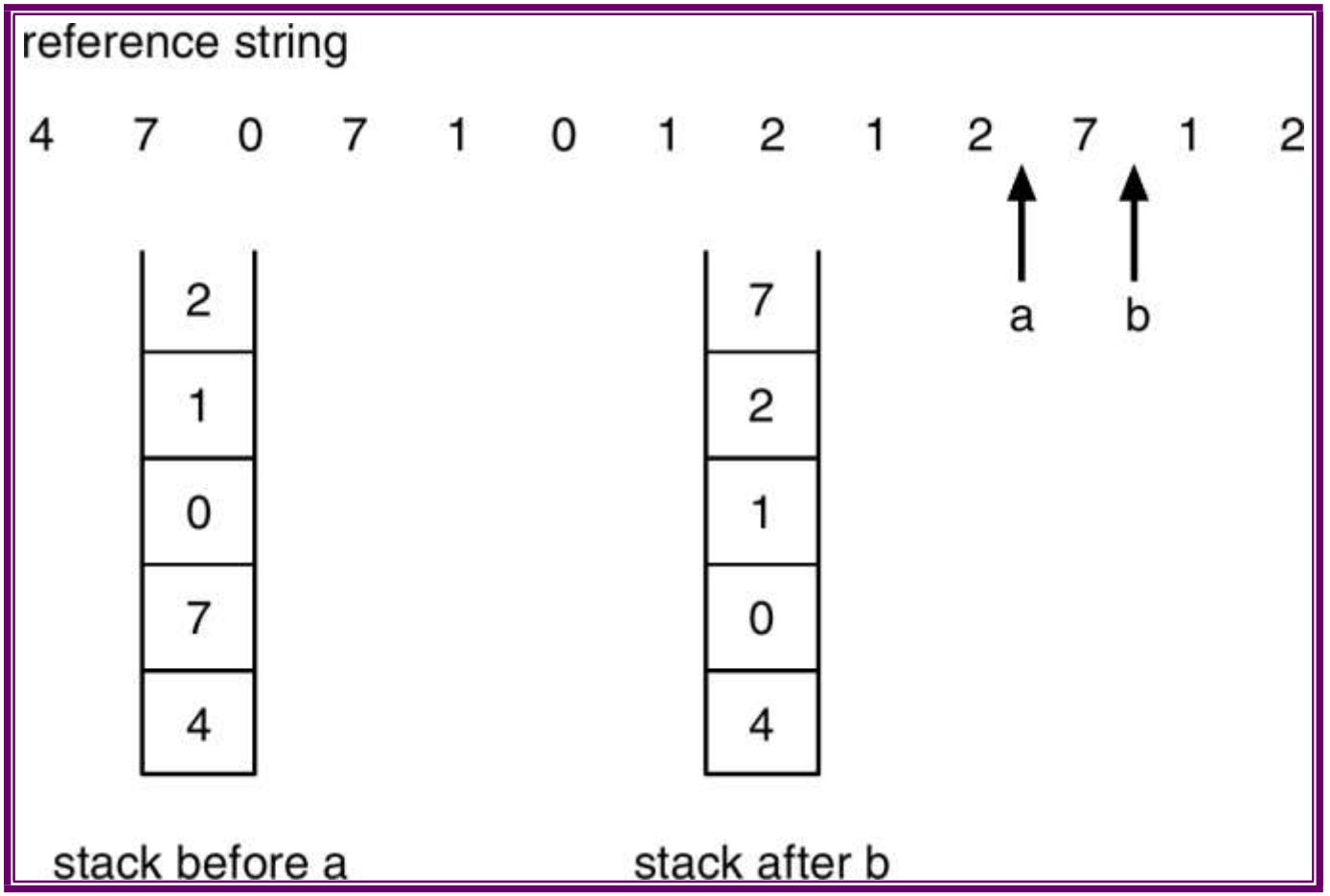
LRU Page Replacement



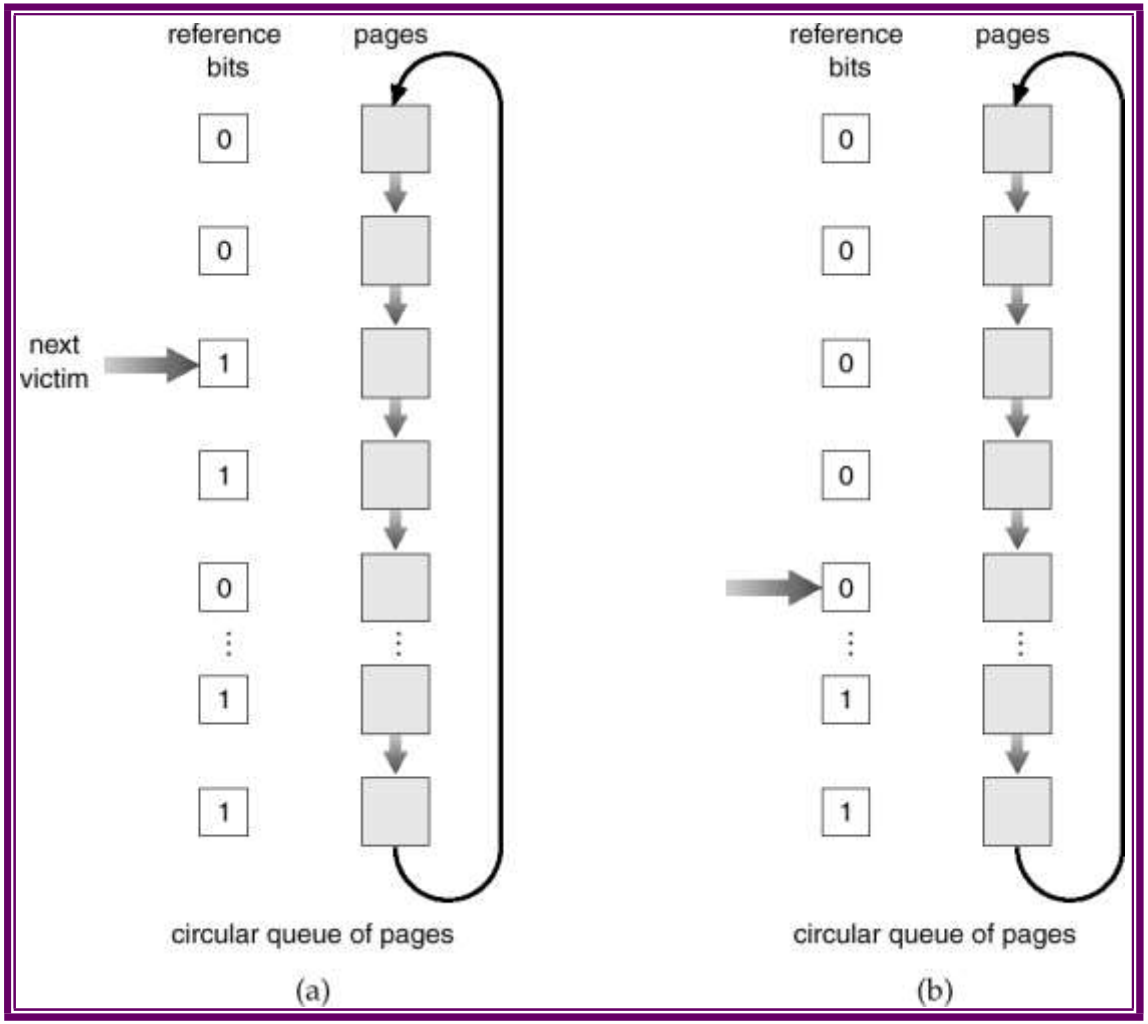
LRU Algorithm (Cont.)

- Stack implementation – keep a stack of page numbers in a double link form:
  - Page referenced:
    - move it to the top
    - requires 6 pointers to be changed
  - No search for replacement

Use Of A Stack to Record The Most Recent Page References



Second-Chance (clock) Page-Replacement Algorithm



Counting Algorithms

- Keep a counter of the number of references that have been made to each page.
- LFU Algorithm: replaces page with smallest count.
- MFU Algorithm: based on the argument that the page with the smallest count was probably just brought in and has yet to be used.

Allocation of Frames

- Each process needs **minimum** number of pages.

- Example: IBM 370 – 6 pages to handle SS MOVE instruction:

  - instruction is 6 bytes, might span 2 pages.
  - 2 pages to handle **from**.
  - 2 pages to handle **to**.

- Two major allocation schemes.

  - fixed allocation
  - priority allocation

Fixed Allocation

- Equal allocation – e.g., if 100 frames and 5 processes, give each 20 pages.
- Proportional allocation – Allocate according to the size of Process.

s_i = size of process p_i S
$= \sum s_i$
m = total number of frames
a_i = allocation for $P_i = s_i / S \times m$

$$m = 64$$
$$s_i = 10$$
$$s_2 = 127$$

$$a_1 = \frac{10}{137} \times 64 \approx 5$$
$$a_2 = \frac{127}{137} \times 64 \approx 59$$

Priority Allocation

Use a proportional allocation scheme using priorities rather than size.

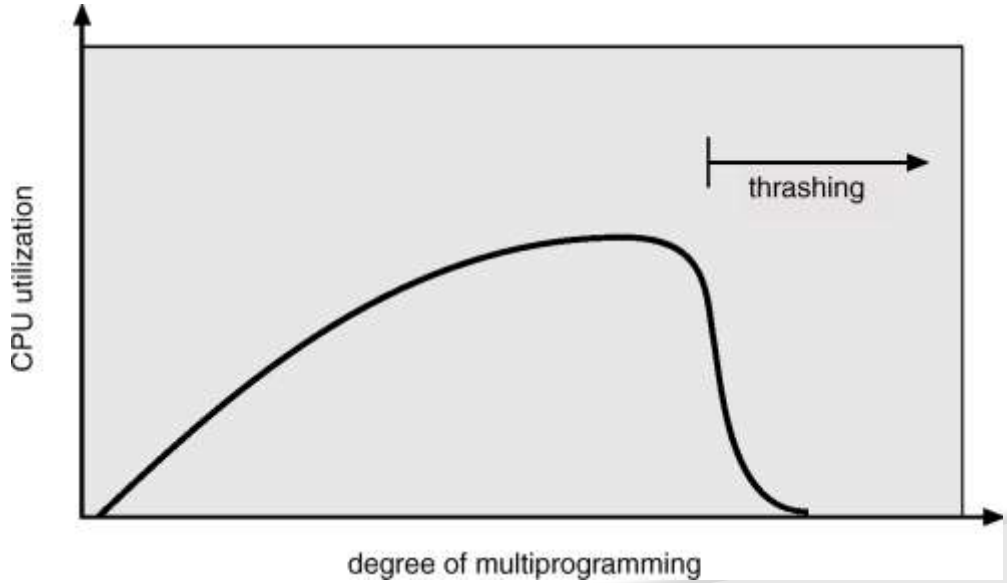
If process P_i generates a pagefault,

- select for replacement one of its frames.
- select for replacement a frame from a process with lower priority number.

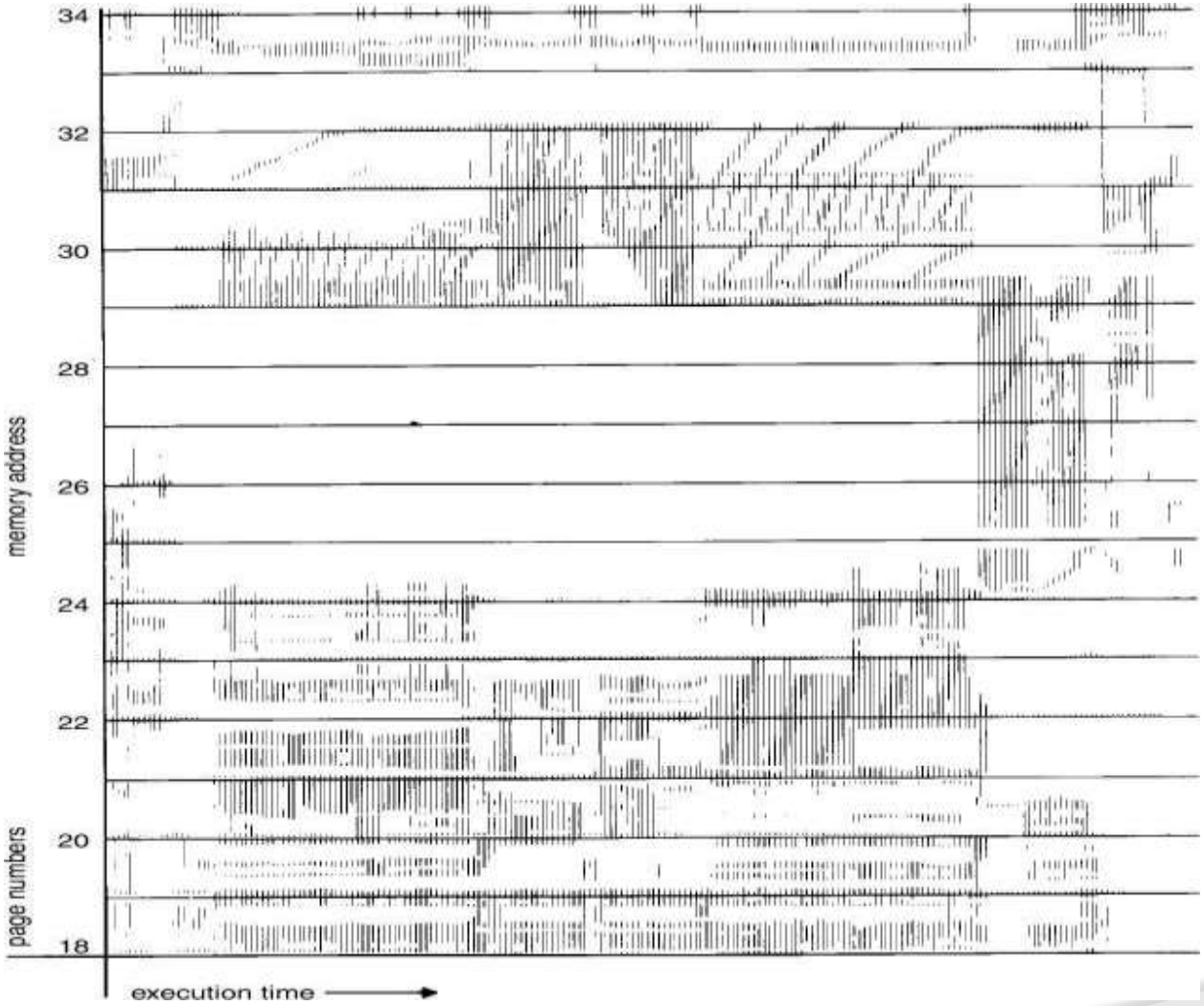
Global vs. Local Allocation

- **Global** replacement – process selects a replacement frame from the set of all frames; one process can take a frame from another.
- **Local** replacement – each process selects from only its own set of allocated frames**.**

Thrashing



Locality In A Memory-Reference Pattern



Working-Set Model

- working-set window ⇒ a fixed number of page references
  Example: 10,000 instruction

- WSS_i (working set of Process P_i) =
  total number of pages referenced in the most recent Δ (varies in time)
  - ❖ if Δ too small will not encompass entire locality.
  - ❖ if Δ too large will encompass several localities.
  - ❖ if $\Delta = \infty$ will encompass entire program.

- $D = \sum WSS_i$ = total demand frames

- if $D > m \Rightarrow$ Thrashing

- Policy if $D >$ m, then suspend one of the processes.

Keeping Track of the Working Set

- Approximate with interval timer + a reference bit

- Example: □ = 10,000

□ Timer interrupts after every 5000 time units.

□ Keep in memory 2 bits for each page.

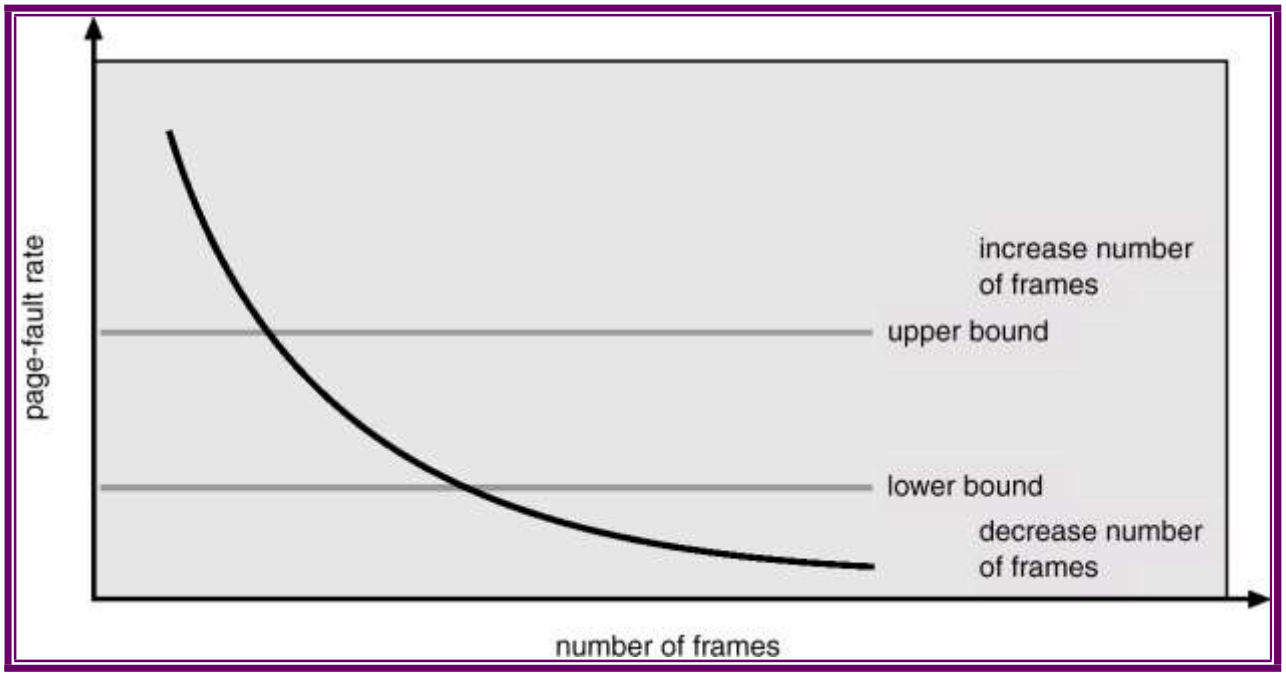
□ Whenever a timer interrupts copy and sets the values of all reference bits to 0.

□ If one of the bits in memory = 1 □ page in working set.

- Why is this not completely accurate?

- Improvement = 10 bits and interrupt every 1000 time units.

Page-Fault Frequency Scheme



- Establish “acceptable” page-fault rate.
  - If actual rate too low, process loses frame.
  - If actual rate too high, process gains frame.

Other Considerations

- Preparing
- Page size selection
  - fragmentation
  - table size
  - I/O overhead
  - locality
- **TLB Reach** - The amount of memory accessible from the TLB.
- TLB Reach = (TLB Size) X (Page Size)
- Ideally, the working set of each process is stored in the TLB. Otherwise there is a high degree of page faults.

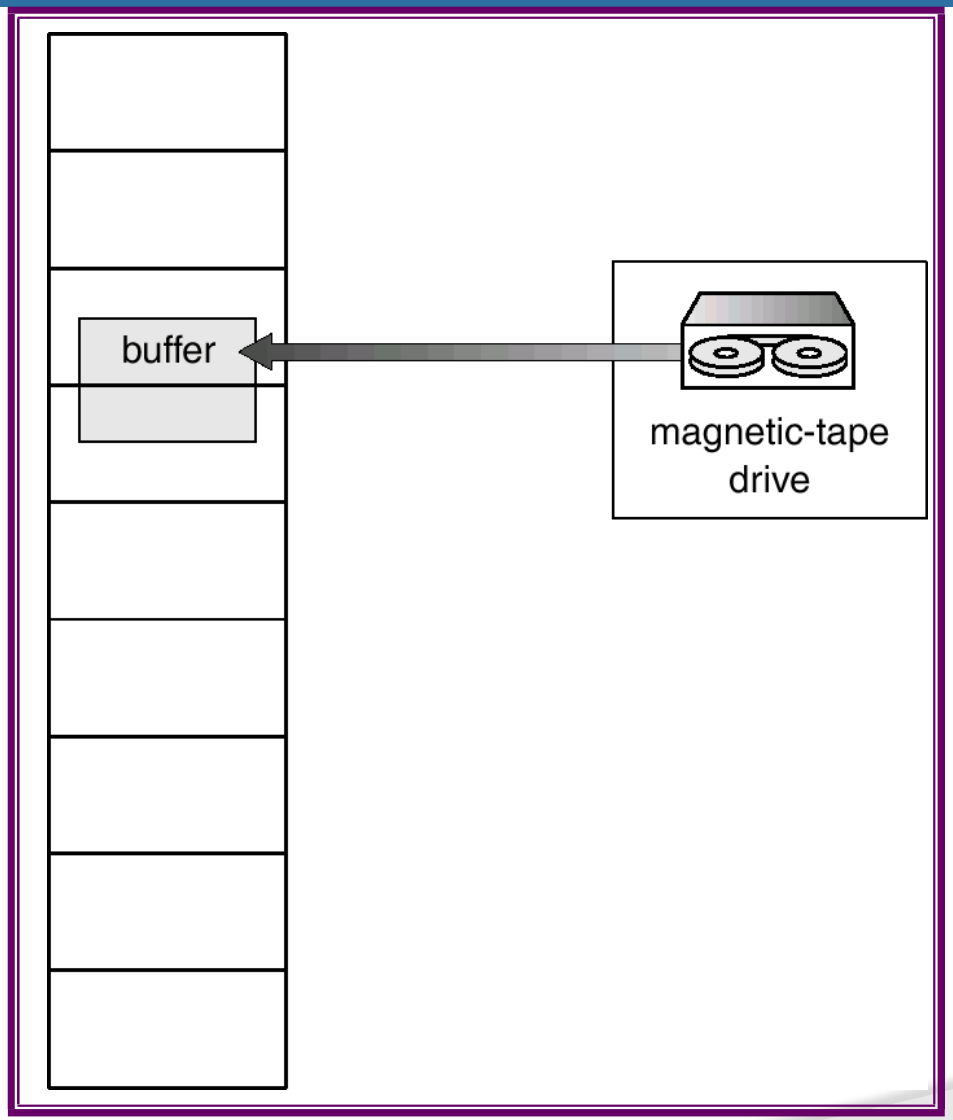
Increasing the Size of the TLB

- **Increase the Page Size**. This may lead to an increasein

fragmentation as not all applications require a large page size.

- **Provide Multiple Page Sizes**. This allows applications that require larger page sizes the opportunity to use them without an increase in fragmentation.

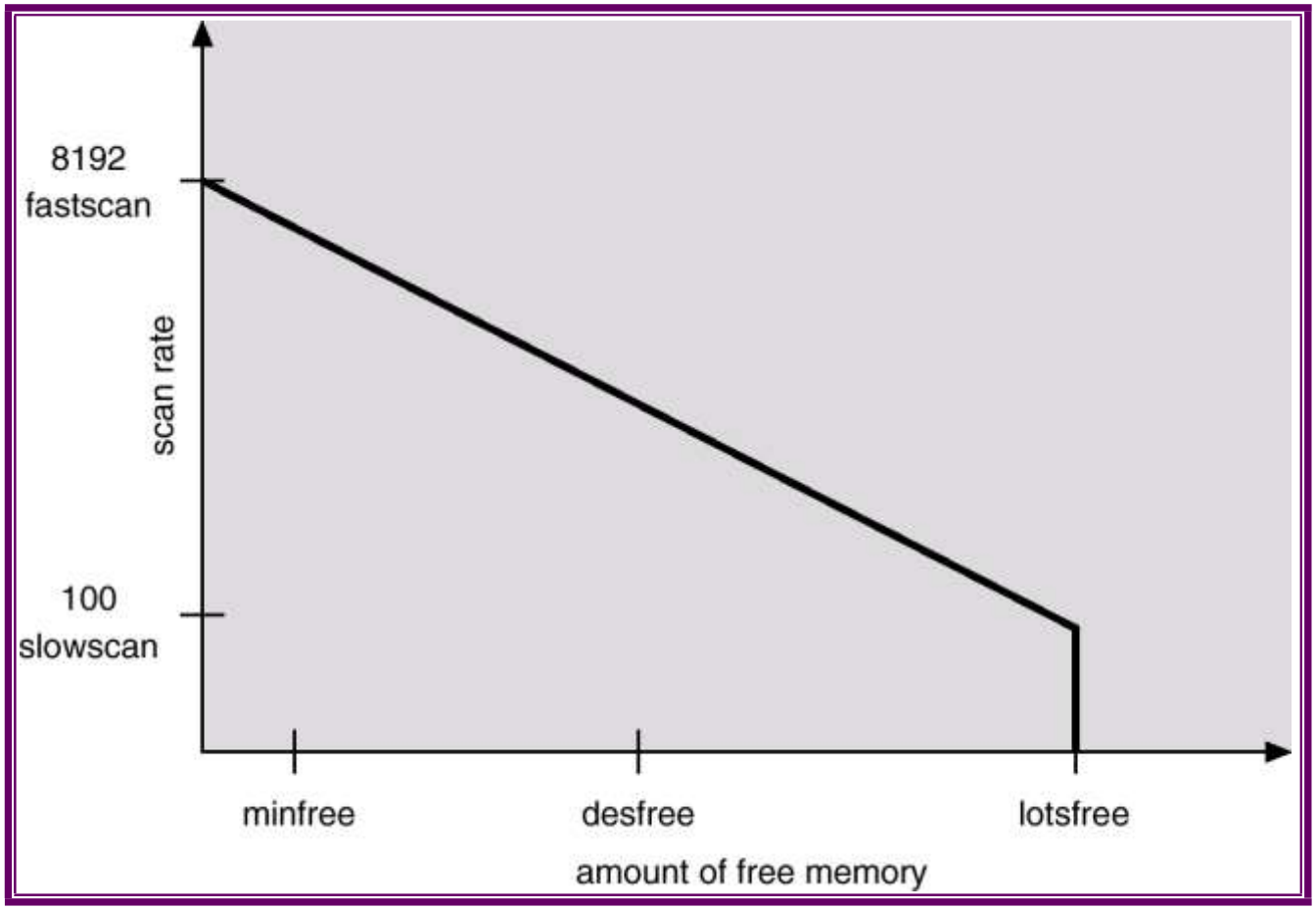
Reason Why Frames Used For I/O Must Be In Memory



Solaris 2

- Maintains a list of free pages to assign faulting processes.
- **Lots free** – threshold parameter to begin paging.
- Paging is performed by *page out* process.
- Page out scans pages using modified clock algorithm.
- **Scan rate** is the rate at which pages are scanned. This ranged from **slows can** to **fasts can**.
- Page out is called more frequently depending upon the amount of free memory available.

Solar Page Scanner



MODULE- IV :
FILE SYSTEM INTERFACE, MASS-STORAGE STRUCTURE

File System Layers

- Device drivers manage I/O devices at the I/O control layer
- Basic file system given command like “retrieve block 123” translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data

File-System Structure

- File structure
  - Logical storage unit
  - Collection of related information
- File system resides on secondary storage (disks)
  - Provided user interface to storage, mapping logical to physical
  - Provides efficient and convenient access to disk by allowing data to be stored, located retrieved easily
- Disk provides in-place rewrite and random access
  - I/O transfers performed in blocks of sectors (usually 512 bytes)
- File control block – storage structure consisting of information about a file
- Device driver controls the physical device

File System Layers

- Device drivers manage I/O devices at the I/O control layer
- Basic file system given command like “retrieve block 123” translates to device driver
- Also manages memory buffers and caches (allocation, freeing, replacement)
  - Buffers hold data in transit
  - Caches hold frequently used data

File-System Implementation

- We have system calls at the API level, but how do we implement their functions?
  - On-disk and in-memory structures
- Boot control block contains info needed by system to
  - boot OS from that volume
  - Needed if volume contains OS, usually first block of volume
- Volume control block (superblock, master file table)
  - contains volume details
  - Total of blocks, of free blocks, block size, free block pointers or array
- Directory structure organizes the files
  - Names and inode numbers, master file table

File-System Implementation (Cont.)

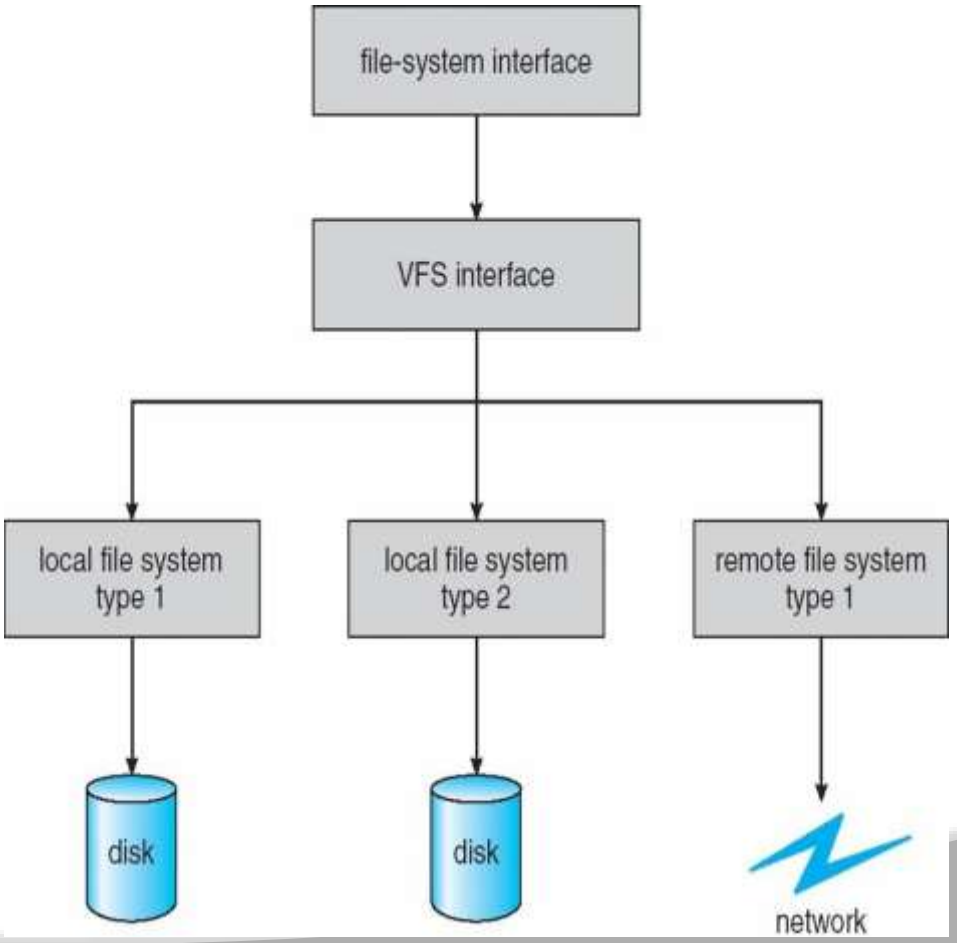
- Per-file File Control Block (FCB) contains many details about the file
  - inode number, permissions, size, dates
  - NFTS stores into in master file table using relational DB structures

Virtual File Systems

- Virtual File Systems (VFS) on Unix provide an object- oriented way of implementing file systems
- VFS allows the same system call interface (the API) to be used for different types of file systems
  - Separates file-system generic operations from implementation details
  - Implementation can be one of many file systems types, or
  - network file system
  - Implements vnodes which hold inodes or network file details
  - Then dispatches operation to appropriate file system implementation routines

Virtual File Systems (Cont.)

- The API is to the VFS interface, rather than any specific type of file system



Virtual File System Implementation

- For example, Linux has four object types:
  - inode, file, superblock, dentry
- VFS defines set of operations on the objects that must be implemented
  - Every object has a pointer to a function table
  - Function table has addresses of routines to implementthat
  - function on that object
  - For example:
  - int open(. . .)—Open a file
  - int close(. . .)—Close an already-openfile
  - ssize t read(. . .)—Read from afile
  - ssize t write(. . .)—Write to afile
  - int mmap(. . .)—Memory-map a file

Directory Implementation

- Linear list of file names with pointer to the data blocks
  - Simple to program
  - Time-consuming to execute
  - Linear search time
  - Could keep ordered alphabetically via linked list or use
  - B+ tree
- Hash Table – linear list with hash data structure
  - Decreases directory search time
  - Collisions – situations where two file names hash to the same location
  - Only good if entries are fixed size, or use chained-overflow method

Allocation Methods - Contiguous

- An allocation method refers to how disk blocks are allocated for files:
- Contiguous allocation – each file occupies set of contiguous blocks
  - Best performance in most cases
  - Simple – only starting location (block #) and length (number of blocks) are required
  - Problems include finding space for file, knowing file size, external fragmentation, need for compaction off-line (downtime) or on-line

Linked Allocation

- Each file is a linked list of disk blocks: blocks may be scattered anywhere on the disk

block =pointer

Indexed Allocation – Mapping

- Mapping from logical to physical in a file of unbounded length (block size of 512 words)
- Linked scheme – Link blocks of index table (no limit on size)

MODULE- V:

Deadlocks

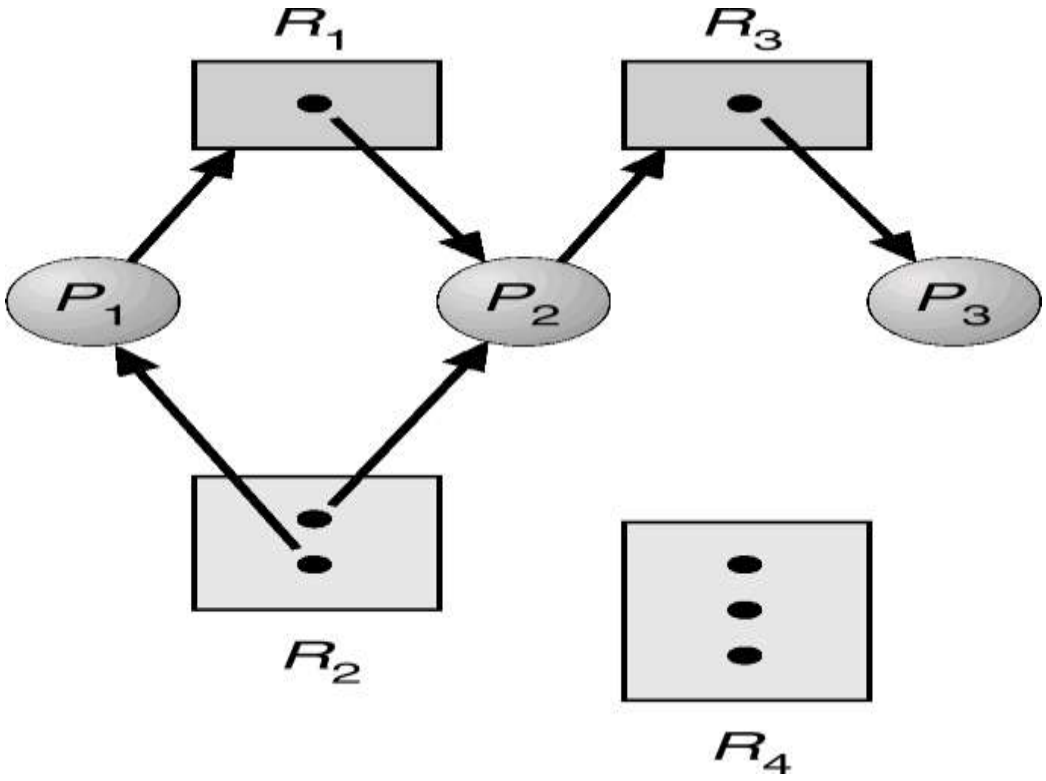
- System Model
- Deadlock Characterization
- Methods for Handling Deadlocks
- Deadlock Prevention
- Deadlock Avoidance
- Deadlock Detection
- Recovery from Deadlock
- Combined Approach to Deadlock Handling

Deadlock Characterization

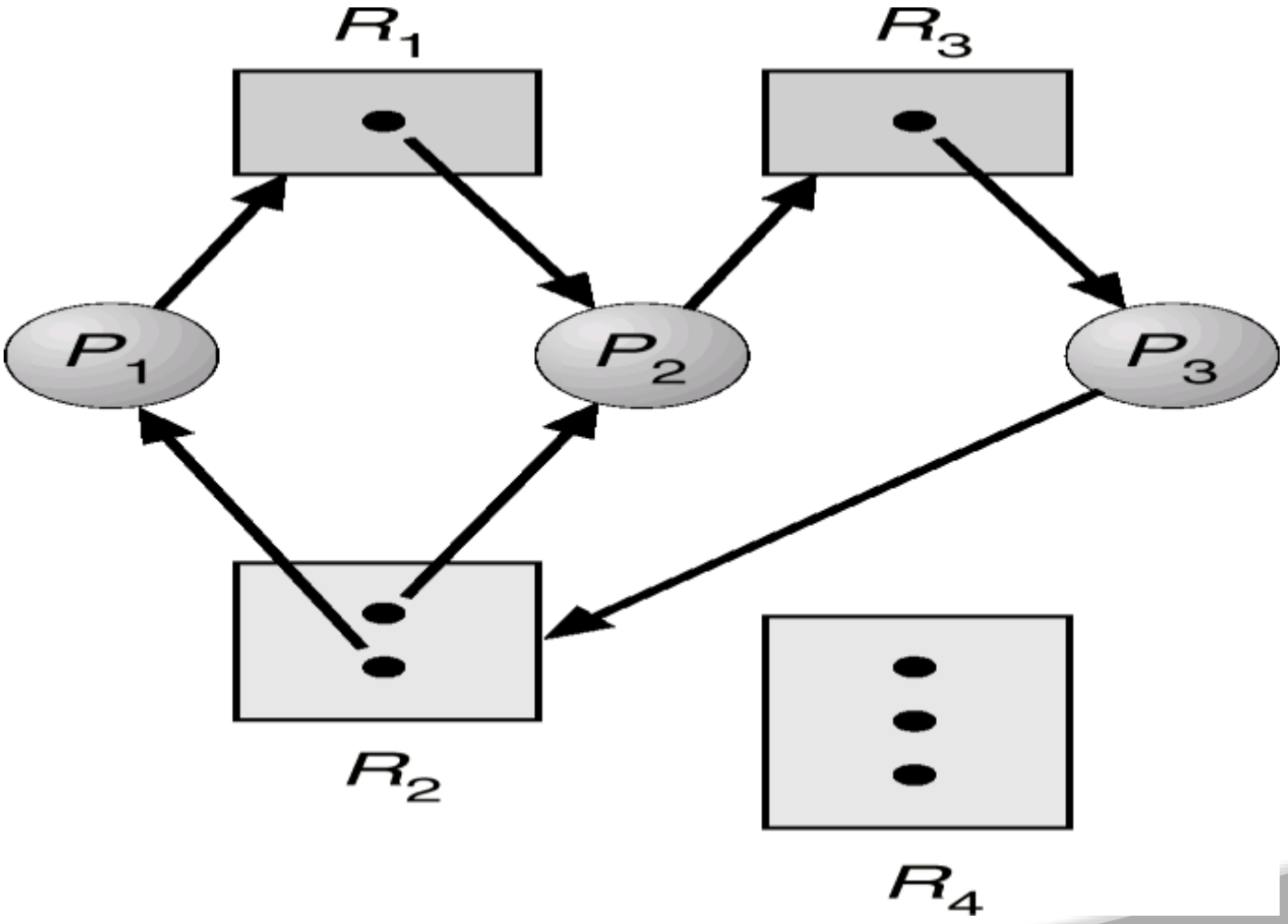
Deadlock can arise if four conditions hold simultaneously.

- Mutual exclusion:only one process at a time can use a resource.
- Hold and wait:a process holding at least one resource is waiting to acquire additional resources held by other processes.
- No preemption:a resource can be released only voluntarily by the process holding it, after that process has completed its task.
- Circular wait: there exists a set $\{P_0, P_1, \ldots, P_0\}$ of waiting processes such that P_0 is waiting for a resource that is held by P_1, P_1 is waiting for a resource that is held by $P_2, \ldots, P_{n-1}$ is waiting for a resource.

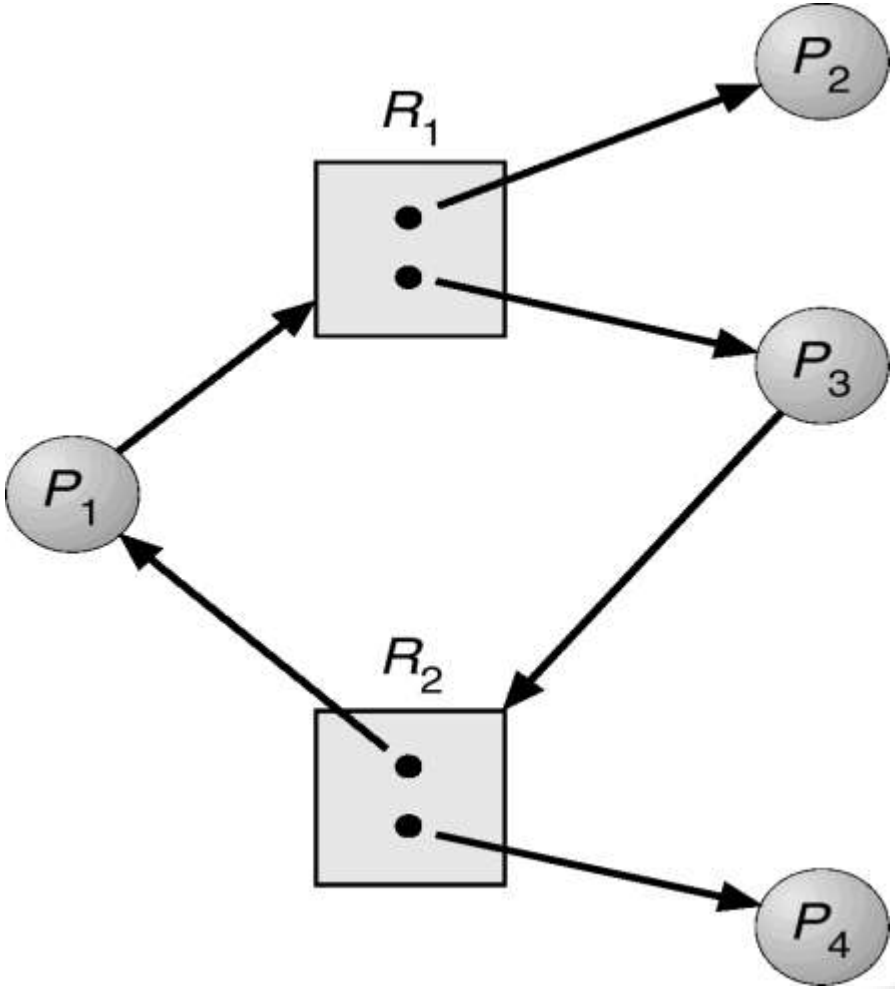
Resource-Allocation Graph



Resource Allocation Graph With A Deadlock



Resource Allocation Graph With A Cycle But No Deadlock



Methods for Handling Deadlocks

- Ensure that the system will *never* enter a deadlock state.
- Allow the system to enter a deadlock state and then recover.
- Ignore the problem and pretend that deadlocks never occur in the system; used by most operating systems, including UNIX.

Deadlock Prevention

Restrain the ways request can be made.

- Mutual Exclusion – not required for sharable resources; must hold for nonsharable resources.

- Hold and Wait – must guarantee that whenever a process requests a resource, it does not hold any other resources.

Require process to request and be allocated all its resources before it begins execution, or allow process to request resources only when the process has none. Low resource utilization,starvationpossible.

Deadlock Prevention (Cont.)

- No Preemption –
  - If a process that is holding some resources requests another resource that cannot be immediately allocated to it, then all resources currently being held arereleased.
  - Preempted resources are added to the list of resources for which the process is waiting.
  - Process will be restarted only when it can regain its oldresources, as well as the new ones that it is requesting.
  - Circular Wait – impose a total ordering of all resource types, and require that each process requests resources in an increasing order of enumeration.

Deadlock Avoidance

The number of available and allocate Requires that the system has some additional *a priori* information available.

- Simplest and most useful model requires that each process declare the *maximum number* of resources of each type that it may need.

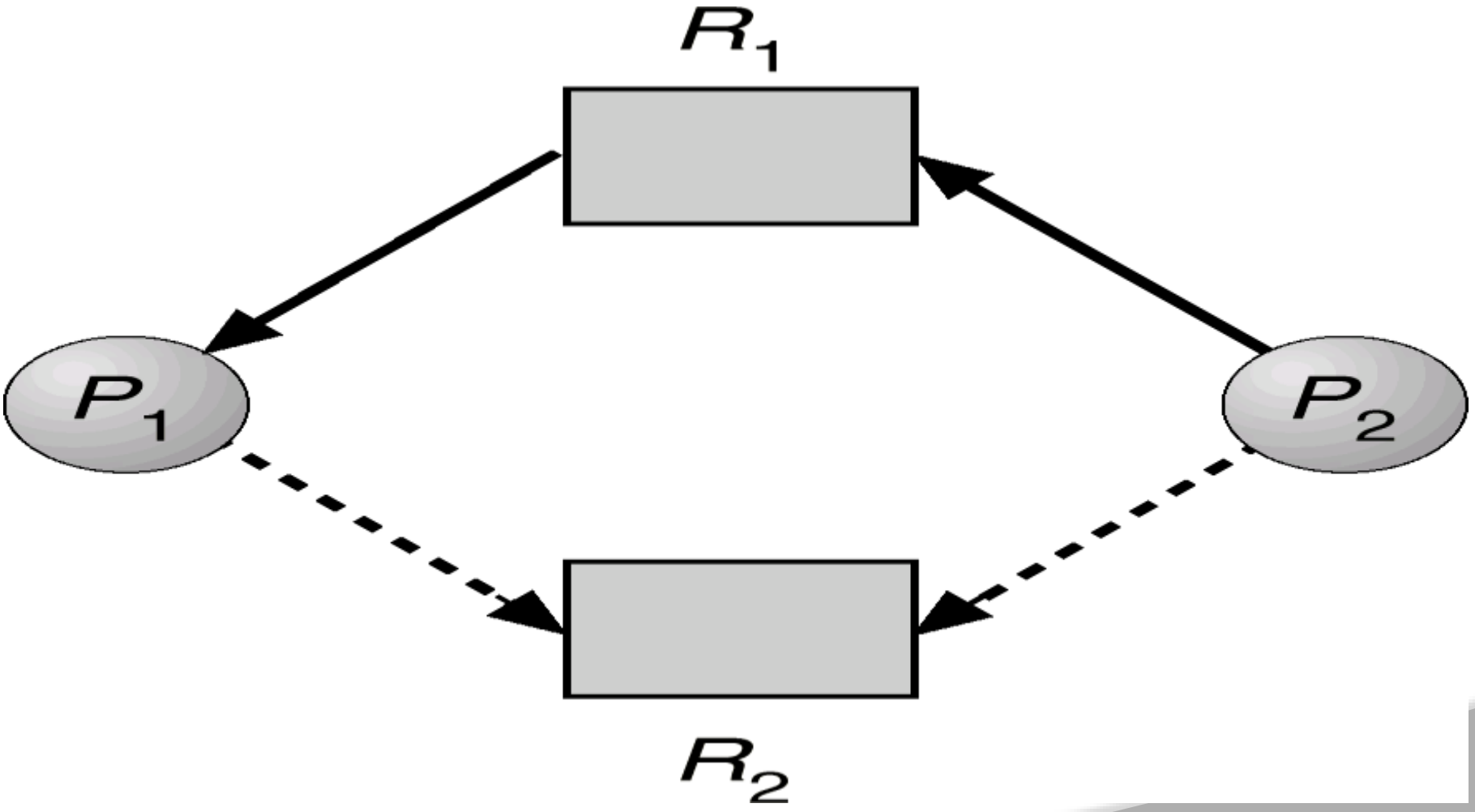
- The deadlock-avoidance algorithm dynamically examines the resource-allocation state to ensure that there can never be a circular-wait condition.

- Resource-allocation *state* is defined by the d resources, and the maximum demands of the processes.

Safe State

- When a process requests an available resource, systemmust decide if immediate allocation leaves the system in a *safe state*.
- System is in safe state if there exists a safe sequence of all processes.
- Sequence <P_1, P_2, …, P_n> is safe if for each P_i, the resources that
  - *Pi* can still request can be satisfied by currently available resources resources held by all the P_j, with $j<I$.
  - If P_i resource needs are not immediately available, then P_i can wait until all P_j havefinished.
  - When P_j is finished, P_i can obtain needed resources, execute, return allocated resources, and terminate.
  - When P_i terminates, P_{i+1} can obtain its needed resources, and so on.

Safe, Unsafe , Deadlock State



Data Structures for the Banker's Algorithm

Let n = number of processes, and m = number of resources types.

- *Available:* Vector of length m. If available [j] = k, there are

k instances of resource type R_j available.

- *Max: n x m* matrix. If *Max* [i,j] = k, then process P_i may request at most k instances of resource type R_j.
- *Allocation:* n x m matrix. If Allocation[i,j] = k then P_i is currently allocated k instances of R_j.
- *Need:* n x m matrix. If *Need*[i,j] = k, then P_i may need k

more instances of R_j to complete its task.

Need [i,j] = *Max*[i,j] – *Allocation* [i,j].

Safety Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively. Initialize:
   Work = *Available*
   Finish [*i*] = *false* for *i* - 1,3, …, *n*.
2. Find and *i* such that both:
   Finish [*i*] = *false*
   $Need_i$? *Work*

If no such *i* exists, go to step 4.

3. *Work* = *Work* + $Allocation_i$
   Finish[*i*] = *true* ,go to step 2.
4. If *Finish* [*i*] == true for all *i*, then the system is in a safe state.

Resource-Request Algorithm for Process P_i

Request = request vector for process P_i.If $Request_i[j] = k$ then process P_i wants k instances of resource type R_j.

1. If $Request_i$ $Need_i$ go to step 2.Otherwise, raise error condition, since process has exceeded its maximumclaim.
2. If $Request_i$? *Available*, go to step 3. Otherwise P_i must wait, since resources are not available.

Pretend to allocate requested resources to P_i by modifying the state as follows:

Available = Available = Request$_i$*; Allocation*$_i$ *= Allocation*$_i$ *+ Request*$_i$*;*

Need$_i$ *= Need*$_i$ *– Request*$_{i;;}$

- *If safe ? the resources are allocated to* P_i.
- *If unsafe ?* P_i *must wait, and the old resource-allocation state is restored*

Detection Algorithm

1. Let *Work* and *Finish* be vectors of length *m* and *n*, respectively Initialize:
   (a) *Work* = *Available*
   (b) For *i* = 1,2, …, *n*, if $Allocation_i$ ≠ 0, then
   Finish[i] = false;otherwise, *Finish*[i] = *true*.
2. Find an index *i* such that both:
   (a) *Finish*[*i*] == *false*
   (b) $Request_i$ ≤ *Work*

   If no such *i* exists, go to step 4.

Detection Algorithm (Cont.)

3. *Work* = *Work* + *Allocation*$_i$ *Finish*[*i*] = *true*
   go to step 2.

4. If *Finish*[*i*] == false, for some *i*, *n*, then the system is in deadlock state. Moreover, if *Finish*[*i*] == *false*, then P_i is deadlocked.

Algorithm requires an order of O(m x $n^{2)}$ operations to detect whether the system is in deadlocked state.

Detection-Algorithm Usage

- When, and how often, to invoke depends on:
  - How often a deadlock is likely to occur?
  - How many processes will need to be rolledback?
  - one for each disjointcycle

- If detection algorithm is invoked arbitrarily, there may be many cycles in the resource graph and so we would not be able to tell which of the many deadlocked processes “caused” the deadlock.

Recovery from Deadlock : Process Termination

- Abort all deadlocked processes.
- Abort one process at a time until the deadlock cycle is eliminated.
- In which order should we choose to abort?
  - Priority of the process.
  - How long process has computed, and how much longer to completion.
  - Resources the process has used.
  - Resources process needs to complete.
  - How many processes will need to be terminated.
  - Is process interactive or batch?

Recovery from Deadlock: Resource Preemption

- Selecting a victim – minimize cost.
- Rollback – return to some safe state, restart process for that state.
- Starvation – same process may always be picked as victim, include number of rollback in cost factor.

Combined Approach to Deadlock Handling

- Combine the three basic approaches
  - prevention
  - avoidance
  - detection
  - allowing the use of the optimal approach for each of resources in the system.

- Partition resources into hierarchically ordered classes.

- Use most appropriate technique for handling deadlocks within each class.

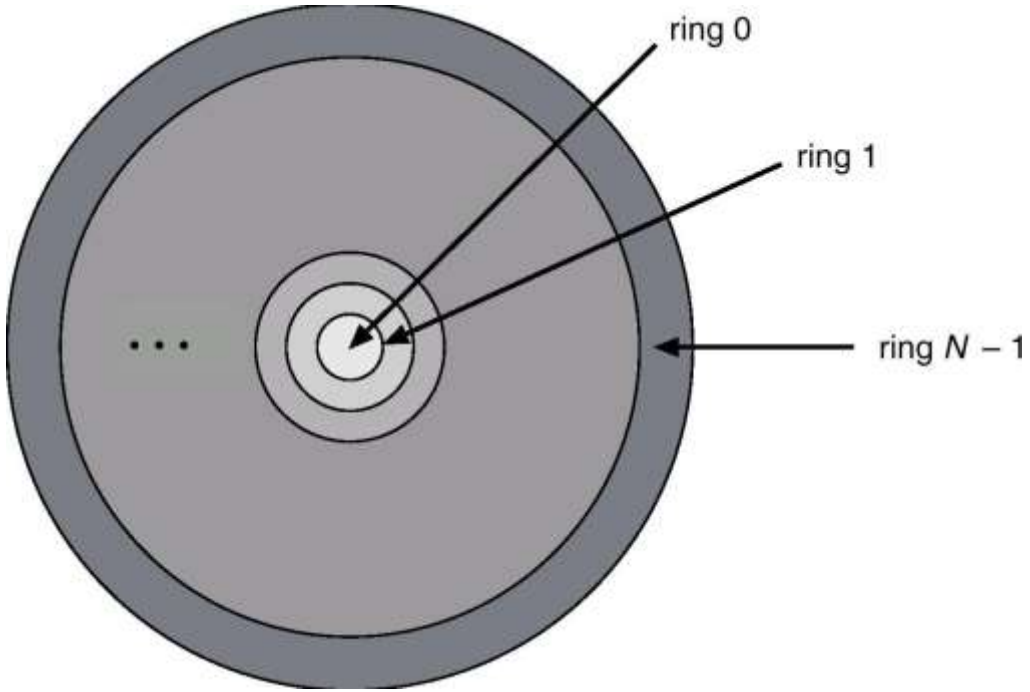
Domain Structure

- Access-right = <*object-name*, *rights-set*>
  - where *rights-set* is a subset of all valid operations that
  - can be performed on the object.

- Domain = set of access-rights

Domain Implementation (Multics)

- Let D_i and D_j be any two domainrings.



Access Matrix

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

Use of Access Matrix

- If a process in Domain D_i tries to do "op" on object O_j, then "op" must be in the access matrix.

- Can be expanded to dynamic protection.
  - Operations to add, delete accessrights.
  - Special accessrights:
    - *owner of* O_i
    - *copy op from* O_i *to* O_j
    - *control* – D_i *can modify* D_j *access rights*
    - *transfer – switch from domain* D_i *to* D_j

Access Matrix With *Owner* Rights

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		
D_2		owner read* write*	read* owner write*
D_3		write	write

(b)

Revocation of Access Rights

- *Access List* – Delete access rights from access list.
  - Simple
  - Immediate

- *Capability List* – Scheme required to locate capability in the system before capability can be revoked.
  - Reacquisition
  - Back-pointers
  - Indirection
  - Keys

Capability-Based Systems

- Hydra
  - Fixed set of access rights known to and interpreted by the system.
  - Interpretation of user-defined rights performed solely by user's program; system provides access protection for use of these rights.

- Cambridge CAP System
  - Data capability - provides standard read, write, execute of individual storage segments associated with object.
  - Software capability -interpretation left to the subsystem,
  - through its protected procedures.

Stack Inspection

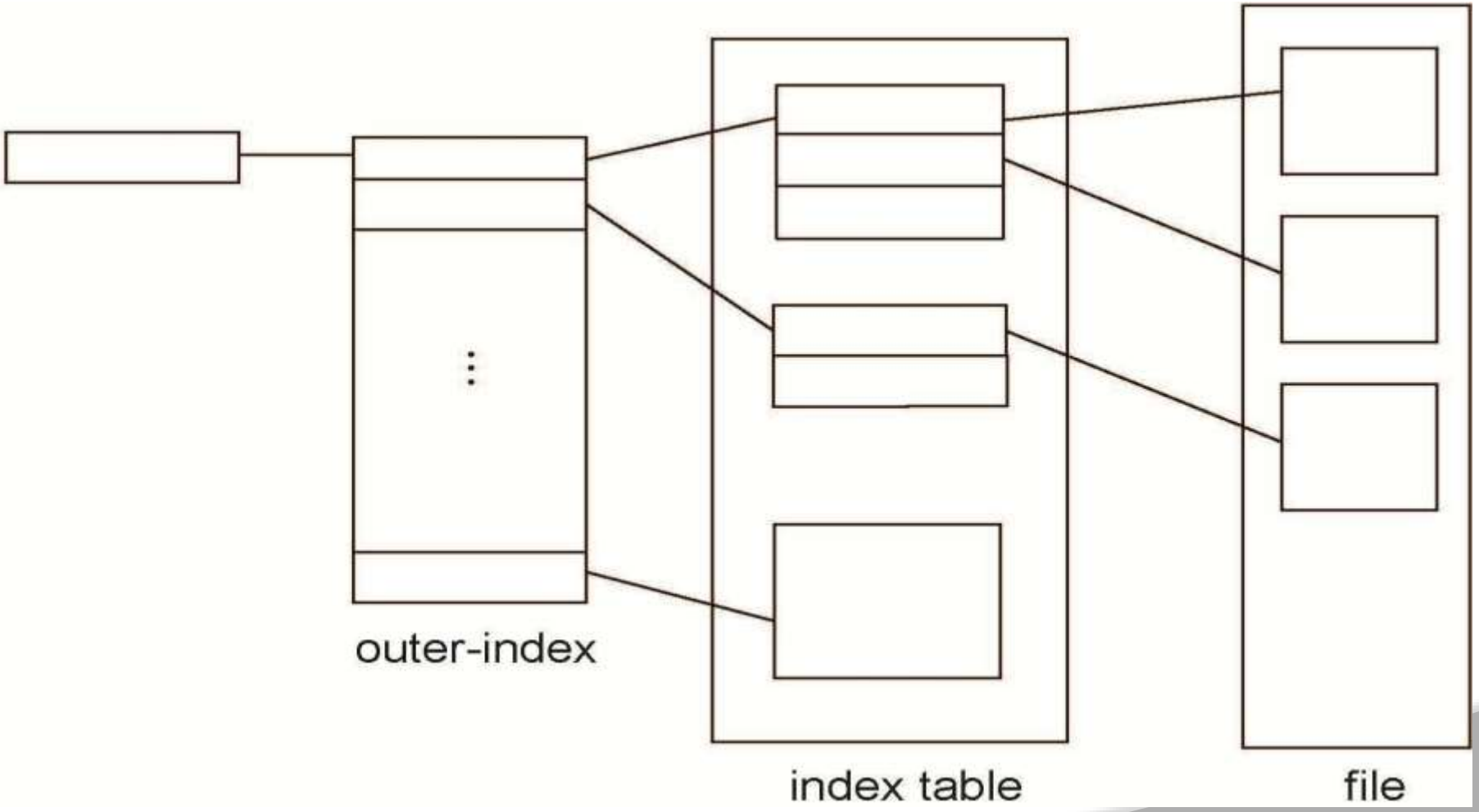
protection domain:	untrusted applet	URL loader	networking
socket permission:	none	*.lucent.com:80, connect	any

class:	gui:
	. . .
	get(url);
	open(addr);
	. . .
		get(URL u):
		. . .
		doPrivileged {
		open('proxy.lucent.com:80');
		}
		<request u from proxy>
		. . .
			open(Addr a):
			. . .
			checkPermission(a, connect);
			connect (a);
			. . .

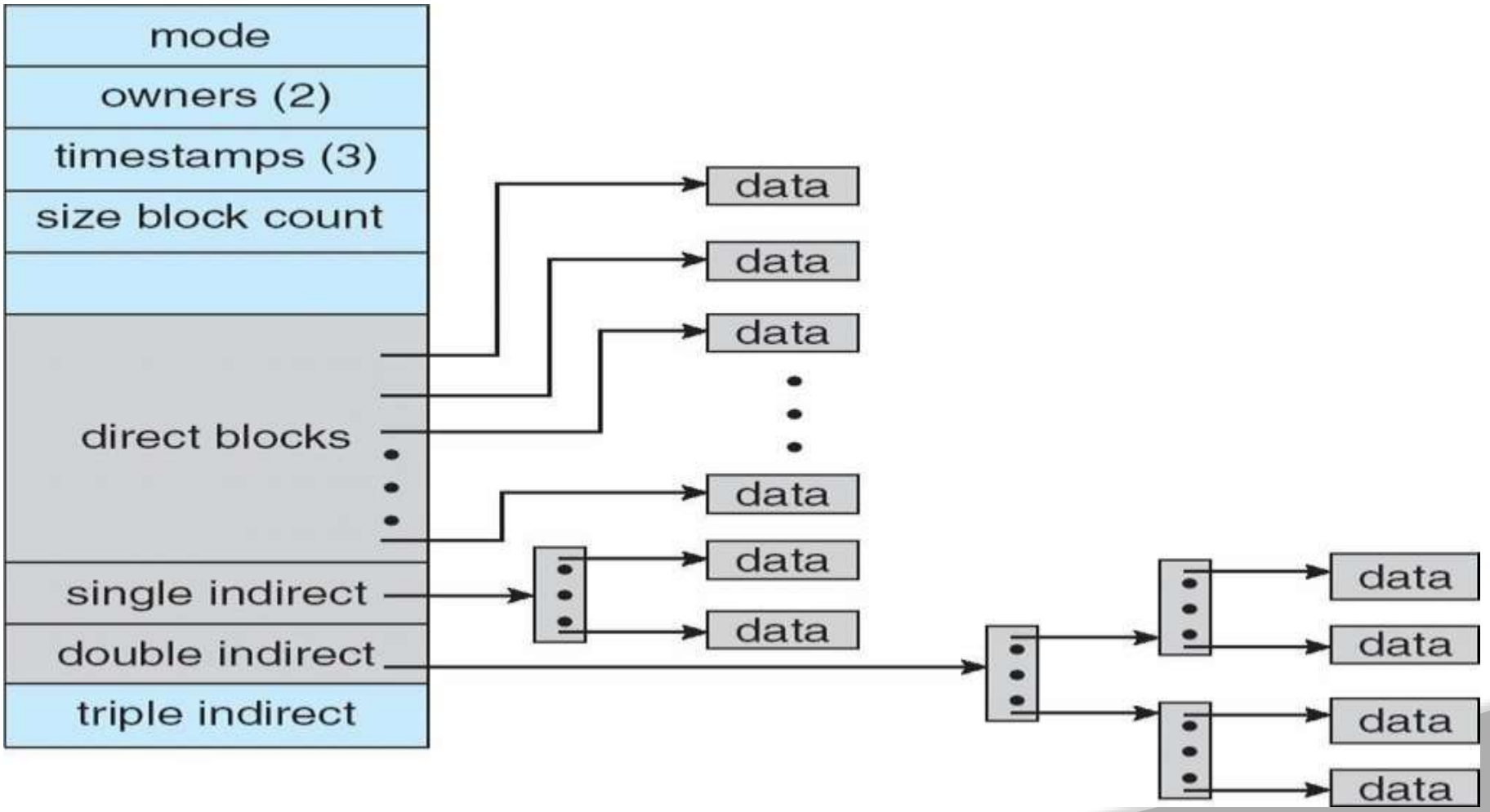
Language-Based Protection

- Specification of protection in a programming language allows the high-level description of policies for the allocation and use of resources.
- Language implementation can provide software for protection enforcement when automatic hardware- supported checking is unavailable.
- Interpret protection specifications to generate calls on whatever protection system is provided by the hardware and the operating system.

Indexed Allocation – Mapping (Cont.)



Combined Scheme: UNIX UFS



Free-Space Management

- File system maintains free-space list to track available blocks/clusters
- (Using term “block” for simplicity)
- Space Maps
- Used in ZFS
- Consider meta-data I/O on very large filesystems
- Full data structures like bit maps couldn’tfit in memory -> thousands of I/Os
- Divides device space into metaslab units and managesmetaslabs
- Given volume can contain hundreds ofmetaslabs
- Each metaslab has associated space map
- Uses counting algorithm
- But records to log file rather than filesystem
- Log of all block activity, in time order, in countingformat
- Metaslab activity -> load space map into memory in balanced-tree

Free-Space Management

- Space Maps
  - Used in ZFS
  - Consider meta-data I/O on very large filesystems
  - Full data structures like bit maps couldn’tfit in memory -> thousands of I/Os
  - Divides device space into metaslab units and managesmetaslabs
  - Given volume can contain hundreds ofmetaslabs
  - Each metaslab has associated space map
  - Uses counting algorithm
  - But records to log file rather than filesystem
  - Log of all block activity, in time order, in countingformat
  - Metaslab activity -> load space map into memory in balanced-tree
  - structure, indexed by offset
  - Replay log into that structure
  - Combine contiguous free blocks into single entry

Efficiency and Performance

- Efficiency dependent on:
  - Disk allocation and directory algorithms.
  - Types of data kept in file's directory entry.
  - Pre-allocation or as-needed allocation of metadata structures.
  - Fixed-size or varying-size data structures.

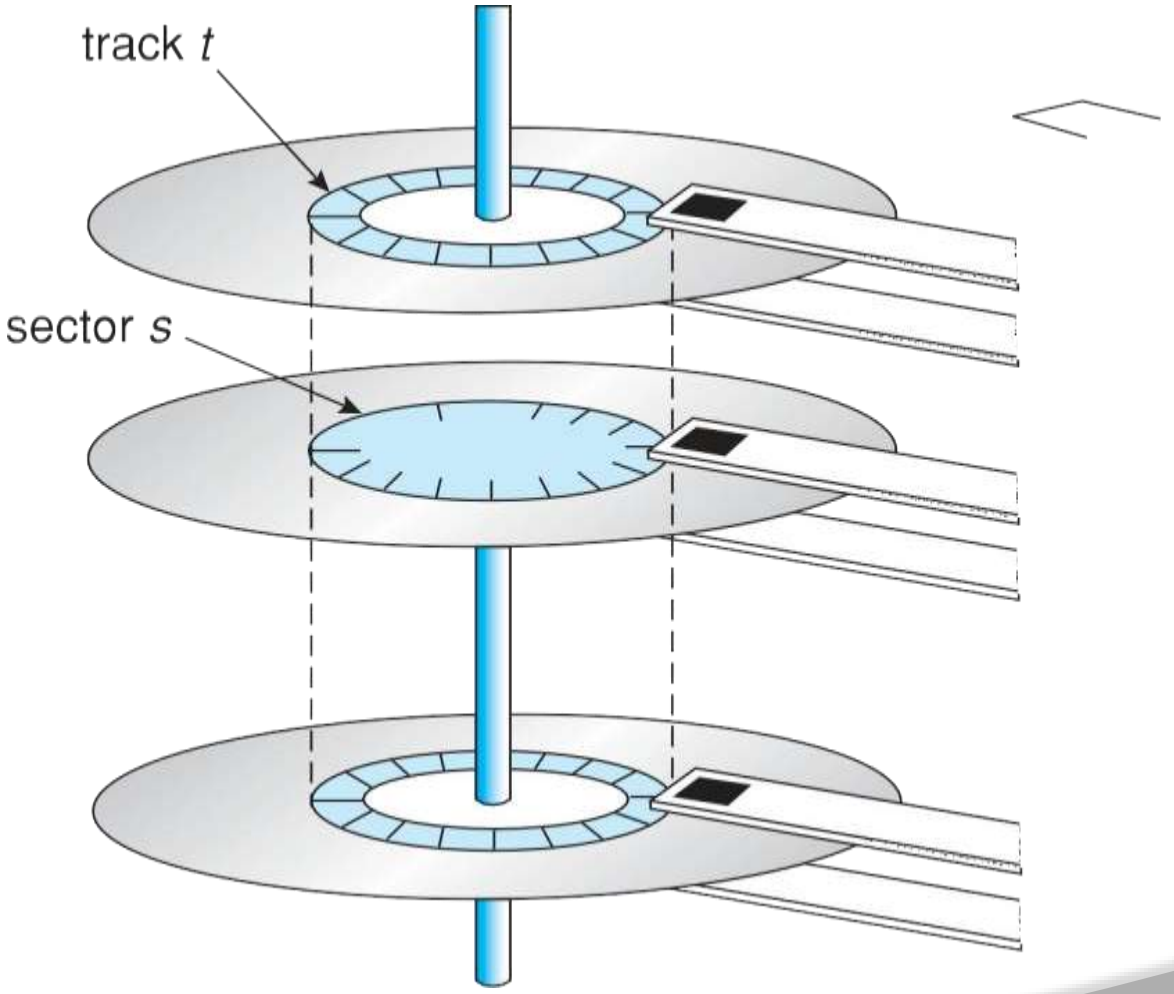
Efficiency and Performance

- Performance
- Keeping data and metadata close together
- Buffer cache – separate section of main memory for frequently used blocks
- Synchronous writes sometimes requested by apps orneeded by OS
- No buffering / caching – writes must hit diskbefore acknowledgement
- Asynchronous writes more common, buffer-able, faster
- Free-behind and read-ahead – techniques to optimize sequential access
- Reads frequently slower thanwrites

Overview of Mass Storage Structure

- Magnetic disks provide bulk of secondary storage of modern
- computers
- Drives rotate at 60 to 250 times persecond
- Transfer rate is rate at which data flow between drive and computer
- Positioning time (random-access time) is time to move disk arm to desired cylinder (seek time) and time for desired sector to rotate under the disk head (rotational latency)
- Head crash results from disk head making contact with the disk surface -- That's bad
- Disks can be removable
- into drive or storage array

Moving-head Disk Mechanism



Disk Structure

- Disk drives are addressed as large 1-dimensional arrays of logical blocks, where the logical block is the smallest unit of transfer
- Low-level formatting creates logical blocks on physicalmedia
- The 1-dimensional array of logical blocks is mapped into the sectors of the disk sequentially
- Sector 0 is the first sector of the first track on the outermost cylinder
- Mapping proceeds in order through that track, then the rest of the tracks in that cylinder, and then through the rest of the cylinders from outermost to innermost
- Logical to physical address should be easy
- Except for bad sectors

Disk Attachment

- Host-attached storage accessed through I/O ports talking to I/O busses
- SCSI itself is a bus, up to 16 devices on one cable, SCSI
  - initiator requests operation and SCSI targets perform tasks
  - Each target can have up to 8 logical units (disks attached to device controller)
- FC is high-speed serial architecture
  - Can be switched fabric with 24-bit address space – the basis of storage area networks (SANs) in which many hosts attach to many storage units
- I/O directed to bus ID, device ID, logical unit (LUN)

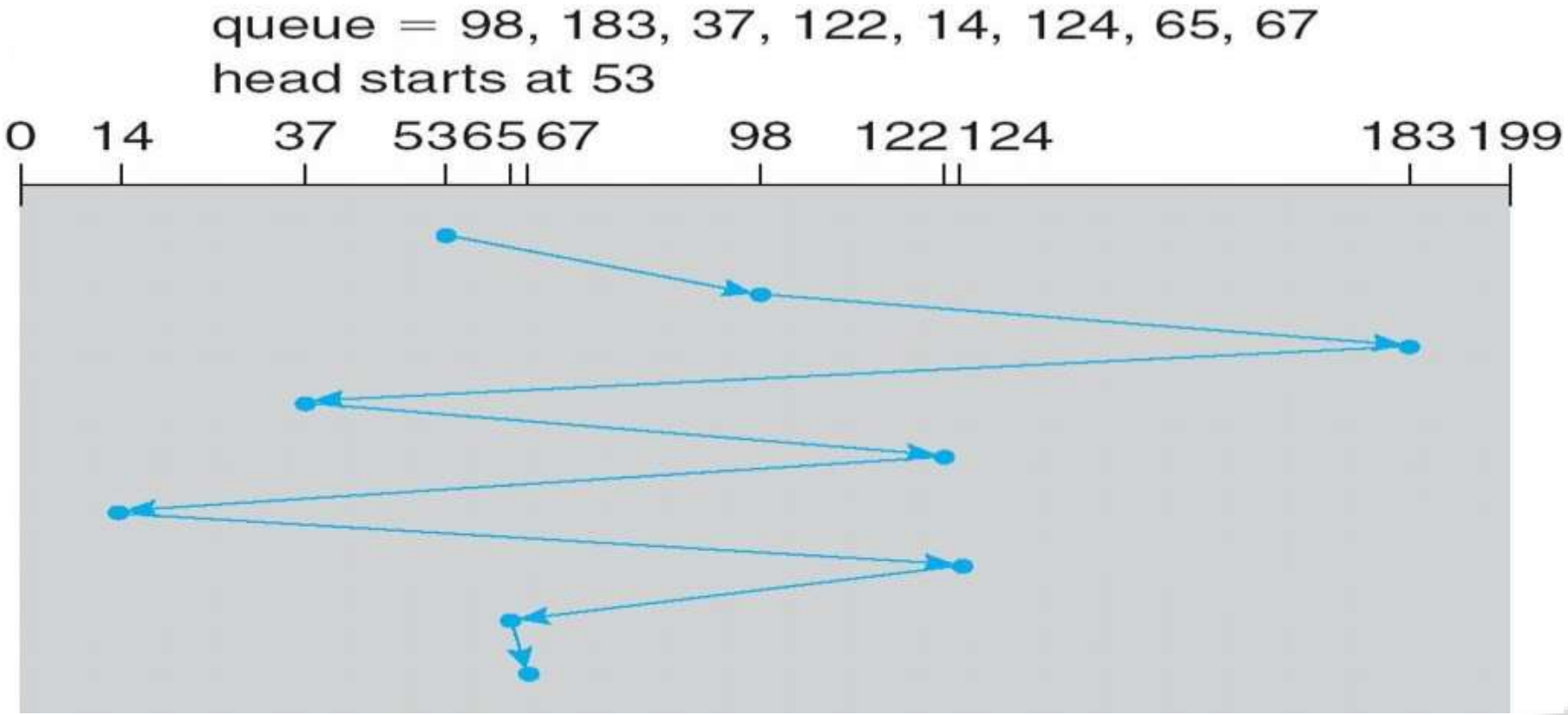
Disk Scheduling

- The operating system is responsible for using hardware efficiently — for the disk drives, this means having a fast access time and disk bandwidth
- Minimize seek time
- Seek time seek distance
- Disk bandwidth is the total number of bytes transferred, divided by the total time between the first request for service and the completion of the last transfer

Disk Scheduling

- There are many sources of disk I/O request
  - OS
  - System processes
  - Users processes
- I/O request includes input or output mode, disk address, memory address, number of sectors to transfer
- OS maintains queue of requests, per disk or device
- Idle disk can immediately work on I/O request, busy disk means work must queue
  - Optimization algorithms only make sense when a queue exists

FCFS


queue = 98, 183, 37, 122, 14, 124, 65, 67
head starts at 53
0
14
37
53
65
67
98
122
124
183
199

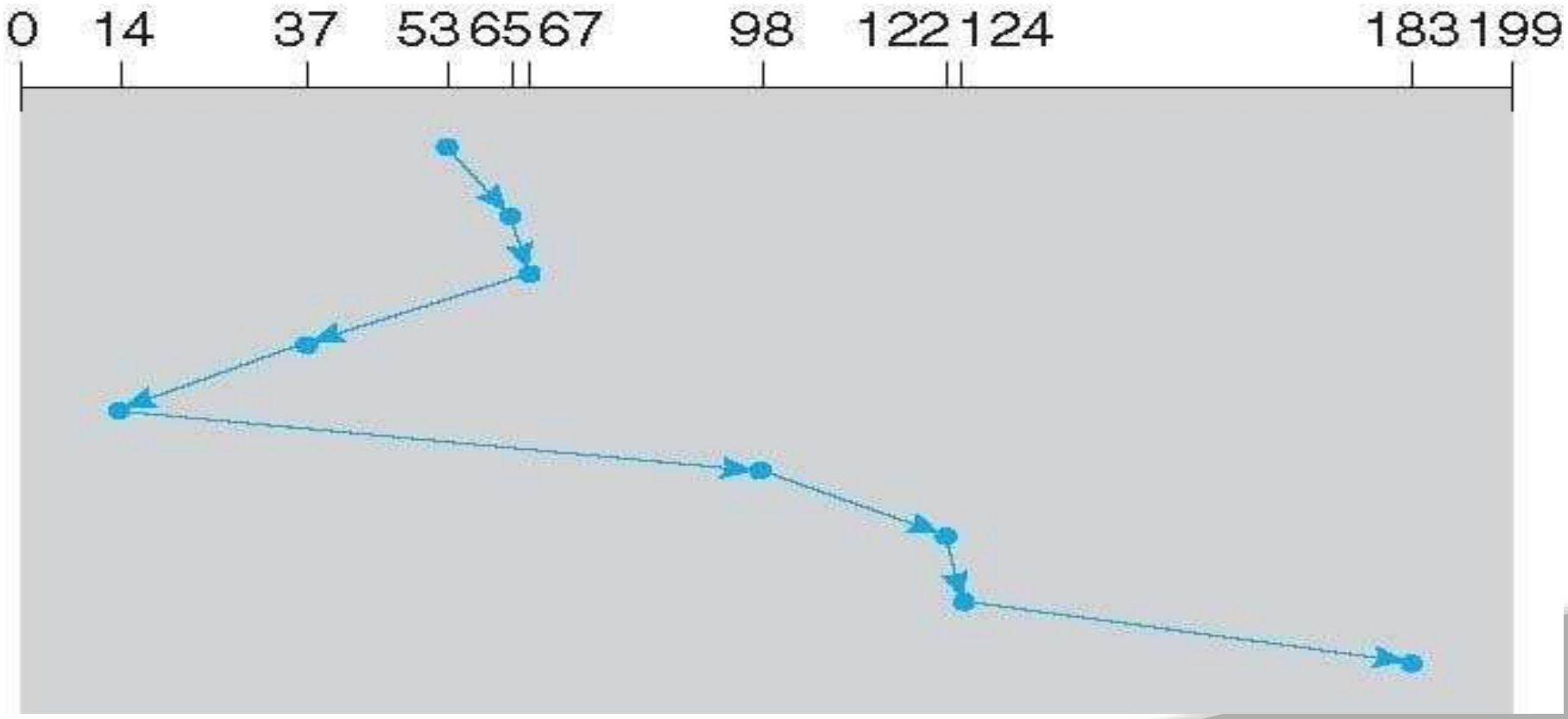
SSTF

- Shortest Seek Time First selects the request with the minimum seek time from the current head position
- SSTF scheduling is a form of SJF scheduling; may cause starvation of some requests
- Illustration shows total head movement of 236 cylinders

SSTF

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

head starts at 53



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.
- SCAN algorithm Sometimes called the elevator algorithm.
- Illustration shows total head movement of 208 cylinders.
- But note that if requests are uniformly dense, largest density at other end of disk and those wait the longest.

C-LOOK

- LOOK a version of SCAN, C-LOOK a 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
- Total number of cylinders?

Selecting a Disk-Scheduling Algorithm

- SSTF is common and has a natural appeal
- SCAN and C-SCAN perform better for systems that place a heavy load on the disk
  - Less starvation
- Performance depends on the number and types of requests
- Requests for disk service can be influenced by the file-allocation method And meta datalayout
- The disk-scheduling algorithm should be written as a separate module of the operating system, allowing it to be replaced with a different algorithm if necessary
- Either SSTF or LOOK is a reasonable choice for the default algorithm
- What about rotational latency?
  - Difficult for OS to calculate
- How does disk-based queueing effect OS queue ordering efforts?

Disk Management

- Low-level formatting, or physical formatting — Dividing a disk into sectors that the disk controller can read and write.
  - Each sector can hold header information, plus data, plus error correction code (ECC)
  - Usually 512 bytes of data but can be selectable
- To use a disk to hold files, the operating system still needs to record its own data structures on the disk.
  - Partition the disk into one or more groups of cylinders, each treated as a logical disk.
  - Logical formatting or “making a file system”
  - To increase efficiency most file systems group blocks into clusters
  - Disk I/O done in blocks
  - File I/O done in clusters

Swap-Space Management

- Swap-space — Virtual memory uses disk space as an extension of main memory
  - Less common now due to memory capacity increases
- Swap-space can be carved out of the normal file system, or, more commonly, it can be in a separate disk partition (raw)
- Swap-space management
  - 4.3BSD allocates swap space when process starts; holds text segment (the program) and data segment
  - Kernel uses swap maps to track swap-space use
  - Solaris 2 allocates swap space only when a dirty page is forced out of physical memory, not when the virtual memory page is first created
  - File data written to swap space until write to file system requested
  - Other dirty pages go to swap space due to no other home
  - Text segment pages thrown out and reread from the file system as needed