Operating System

Introduction to Operating System

What is an operating system?

- Operating system (OS) is a program or set of programs, which acts as an interface between a user of the computer & the computer hardware and acts as resource manager.

- The main purpose of an OS is to provide an environment in which we can execute programs effectively.

The main goals of the OS are

(i) To make the computer system convenient to use

(ii)To make the use of computer hardware in efficient way.

- Operating System is a system software which may be viewed as a collection of software consisting of procedures for operating the computer & providing an environment for execution of programs.

- It’s an interface between user & computer.

- OS makes everything in the computer to work together smoothly & efficiently.

OS ROLES

1. Interface 2. Resource Manager

- Interface
  - Interface between User and Computer
  - The language of user and computer are different.
  - OS plays the role of mediator/translator and makes communication between User and Computer possible.

OS ROLES

- Resource Manager

  - There needs a coordination between different parts of a computer to accomplish a particular task.

  - This is made possible by OS. It monitors and controls every part of the computer.

Operating System Terminologies

User:

A user is anybody that desires work to be done by a computer system.

Job:

A **Job** is the collection of activities needed to do the work required.

Job Steps:

Job Steps are units of work that must be done sequentially.

Once the OS accepts an user's job, it may create several processes.

Process:

- A process(or task) is a computation that may be done concurrently with other computations.

-The following fig depicts the relationship between user,job,process and address spaces

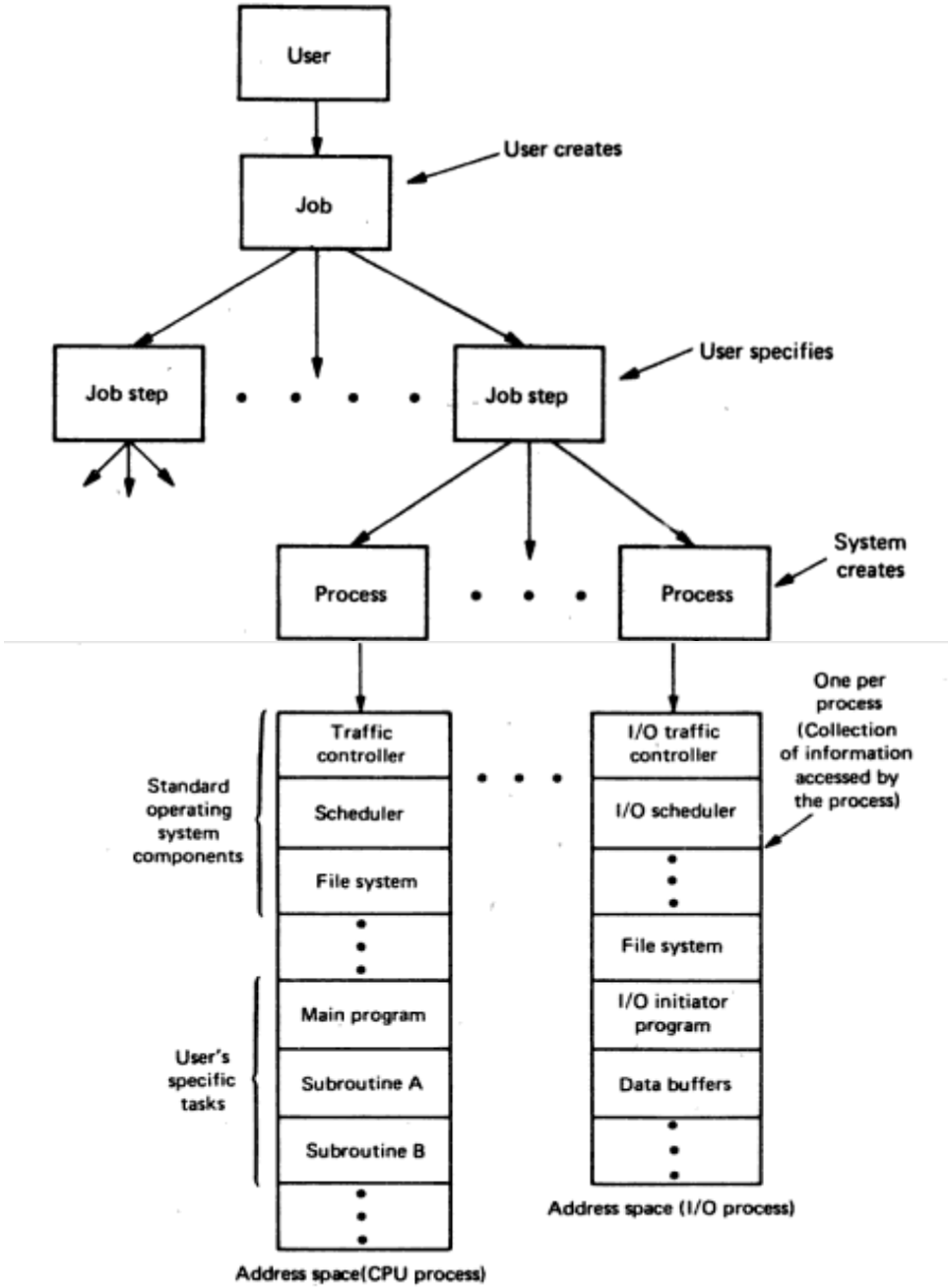


Figure User, job, job step, process, and address space

Multiprogramming:

- A Multiprogramming system may have several processes in '**state of execution**' at the same time.

- A process is in a state of execution if the computation has been started but has not been completed or terminated(error completion).

- Multiprogramming aims to reduce the idle time of processor during execution of a process.

-Due to mismatch in the speed of processor and other devices of the computer, many a times processor remains idle in between execution of a process.

- Processor continues executing another running process which is currently ready.

Various views of OS

1. Viewing OS as User/Computer Interface
2. Viewing OS as Resource Manager
3. Historical view
4. Functional view

Historical development of OS
(Evolution of OS)

1. Job-by-Job processing
2. Early Batch processing
3. Executive systems
4. Multiprogramming OS
5. Comprehensive Information Management and Timesharing
6. Virtual storage and virtual machines
7. Consolidation and Refinement

1. Job-by-Job processing

   Each programmer operated the computer personally-loading decks, pushing buttons, examining storage locations, etc.

2. Early Batch processing

   To reduce set up time(time to load program and data, ie.job), monitor programs are developed that allows users to batch their jobs together which constituted an automatic job sequencing.

3. Executive systems

   When computers became more complex, especially with regard to I/O device management, executive systems were developed that permanently resided in memory and provided Input/Output Control Services(IOCS) for user jobs.

4. Multiprogramming OS

   Multiprogramming was used as a technique to enhance the throughput efficiently. Multiprogramming is most useful when we have more percentage of I/O oriented jobs.

5. Comprehensive Information Management and Timesharing

   The concept of direct user interaction via timesharing techniques was incorporated into several systems. Sharing processor time among the currently running processors.

6. Virtual storage

   Using part of secondary memory as primary memory to store instruction and data of programmes in state of execution and bringing in and out of main memory as part or whole program.

7. Consolidation and Refinement

   Combination of more than techniques like paging, time sharing, virtual storage, etc came into existence.

Functional view of OS

Most users wish to use the computer only to solve particular problems.

Such users want only that the system provide them with a number of packages to assist them in defining and solving their problems.

Examples:

Assembly language translators, Compilers, Subroutines, Linkers and loaders, Utility programs, Application packages, debugging facilities.

- The primary view is that the OS is a collection of programs designed to manage the system's resources, namely,

  memory

  processors

  peripheral devices &

  File/information.

- It is the function of OS to see that they are used efficiently & to resolve conflicts arising from competition among the various users.

An Operating System as Resource Manager

- The OS is a manager of system resources.

- A computer system has many resources.

- Since there can be many conflicting requests for the resources especially in multiprogramming, the OS must decide which process to be allocated resources, when and how long to operate the computer system fairly & efficiently.

-

The Operating system as a resource manager, each manager do the following.

1. Keep track of the STATUS of resources
2. Enforce policy to determine who gets what , when and how much
3. Allocate the Resource
4. Deallocate /Reclaim the Resource

The major functions of each category of OS are.

1.Memory Management Functions

-The OS is responsible for the following memory management functions:

1. Keep track of which segment of memory is in use & by whom and which parts are free.

2. Deciding policy on which processes are to be loaded into memory when space becomes available. In multiprogramming environment it decides which process gets the available memory, when it gets it, where does it get it, & how .

3. Allocate the memory

4. Reclaim the resource when the process no longer needs it or process has been terminated.

2. Processor/Process Management Functions

1. Keeps track of processor & status of processes. The program that does this has been called the traffic controller.

2. Decide policy on which process gets the processor, when and how much. This is done by process scheduler.

. Allocate the processor to a process. This is done by dispatcher.

5. Deallocate the processor when process relinquishes the processor usage, terminates.

3. I/O Device Management Functions

-The OS is responsible for the following I/O Device Management Functions:

1.Keep track of the I/O devices, I/O channels, etc. This module is typically called I/O traffic controller.

2. Decide what is an efficient way to allocate the I/O resource. If it is to be shared, then decide who gets it, how much of it is to be allocated, & for how long. This is called I/O scheduling

3. Allocate the I/O device & initiate the I/O operation.

4. Deallocate the resource

4.Information/File Management Functions

1. Keeps track of the information, its location, its usage, status, etc. The module called a file system provides these facilities.

2.Decides who gets hold of information, enforce protection mechanism, & provides for information access mechanism, etc.

3.Allocate the information to a requesting process, e.g., open a file.

4.De-allocate the resource, e.g., close a file

GENERAL DESIGN CONSIDERATIONS

A design procedure for software is based on the following steps.

- Make a clear statement of the problem
- List the relevant databases
- Specify the format of those database
- Devise an algorithm
- Look for modularity
- Repeat steps (1) through (5) on each module

In case of OS

First Iteration:

Steps 1 through 5 – produce a statement of the facilities desired to the system.

In step 5- OS is functionally divided into four managers.

Second Iteration:

would pinpoint the key functions within each manager (keep track, policy, allocate and deallocate)

Third iteration:

would require the development of algorithms to perform these tasks.

Fourth iteration:

would involve recognizing basic table maintenance functions, databases and queue handling.

Implementation tools:

Few tools that greatly improve the effectiveness of a system's designer and implementer are

* high-level languages
* timesharing
* structured programming

Use of high-level language may increase programmer's clarity and productivity considerably.

Use of timesharing systems with powerful information management capabilities provides both rapid and easier debugging as well as simplified control and maintenance of the source code module.

The term software factory has been used by several groups to describe such a facility.

Structured programming is an approach to simplify the conceptual design of large systems and ease the implementation and debugging of the system.

I/O Programming ,Interrupt Programming ,Machine Structure

(I) I/O Programming:

- There are 3 basic components of early computer system:
  - (i) The CPU
  - (ii) Main storage Unit(memory)
  - (iii) I/O devices

These components can be interconnected

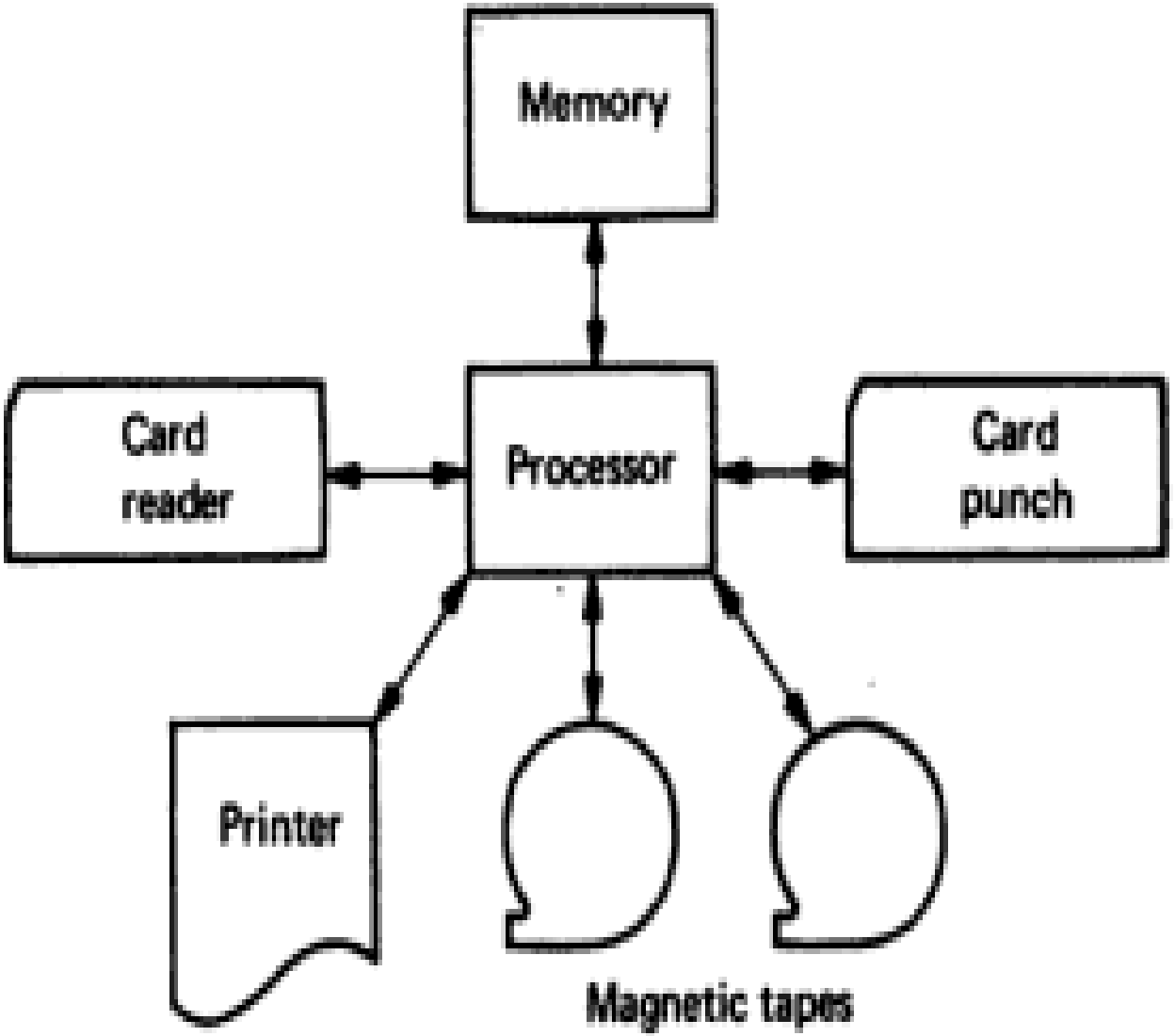


Figure Early computer system

- The early systems also contains I/O instructions such as

  READ A CARD

  PUNCH A CARD

  PRINT A LINE

- to operate I/O devices one at a time.

- An ADD instruction may take one ms of CPU time and a READ CARD instruction 500 ms of CPU time .

- The disparity in speeds between the I/O devices and the CPU motivated the development of I/O Processors(also called I/O Channels)

I/O Channels:

- It provide a path for the data to flow between I/O devices and the main memory.

I/O Processors:

- They are specialized processing units intended to operate the I/O devices.

- They are much less expensive than a CPU.

- The following fig. shows a computer system with a separate I/O channel for each device

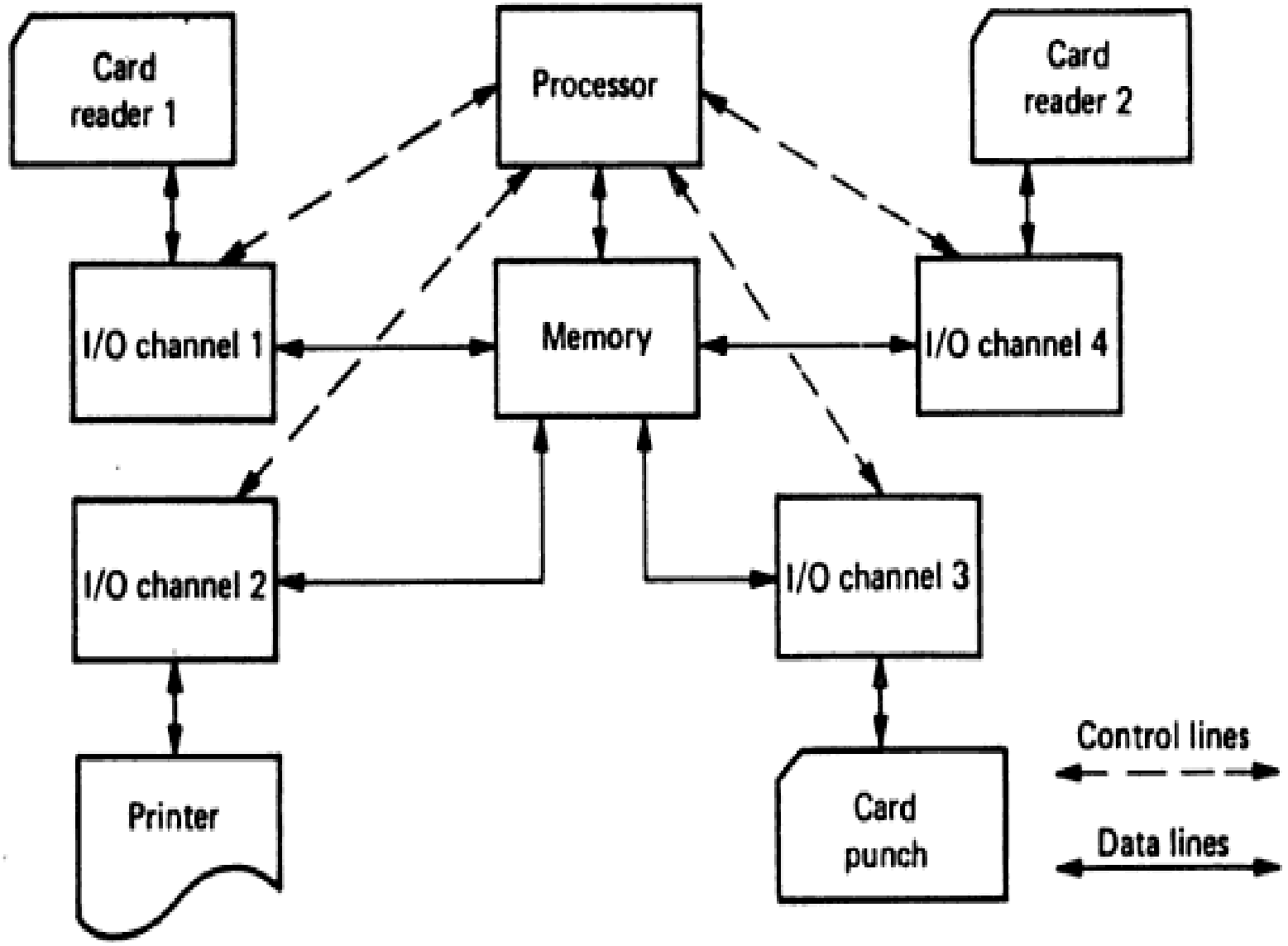


Figure Computer system with I/O channels

Types of I/O Channel:

- I/O channels come in all shapes and sizes, ranging from very simple processors to highly complex CPUs.

- It is costly to use one channel per device so it leads to development of selector and multiplexor channels.

- multiple devices may be connected to each channel (up to 256 devices per channel).

(i) Selector channel

- A selector channel can service only one device at a time-i.e., one device is selected for service.

- These channels are normally used for very high-speed I/O devices, such as magnetic tapes , disks and drums.

- Each I/O request is completed quickly .

(ii) Multiplexor Channel:

- It simultaneously service many devices(up to 256).

- It is used for slow I/O devices, such as card readers, card punches and printers.

(iii) Block Multiplexor Channel:

- It allows multiple channel programs for high-speed devices .

- It performs one channel instruction for one device and then, automatically , switches to perform an instruction for another device, and so on.

- A typical 370 configuration is given below

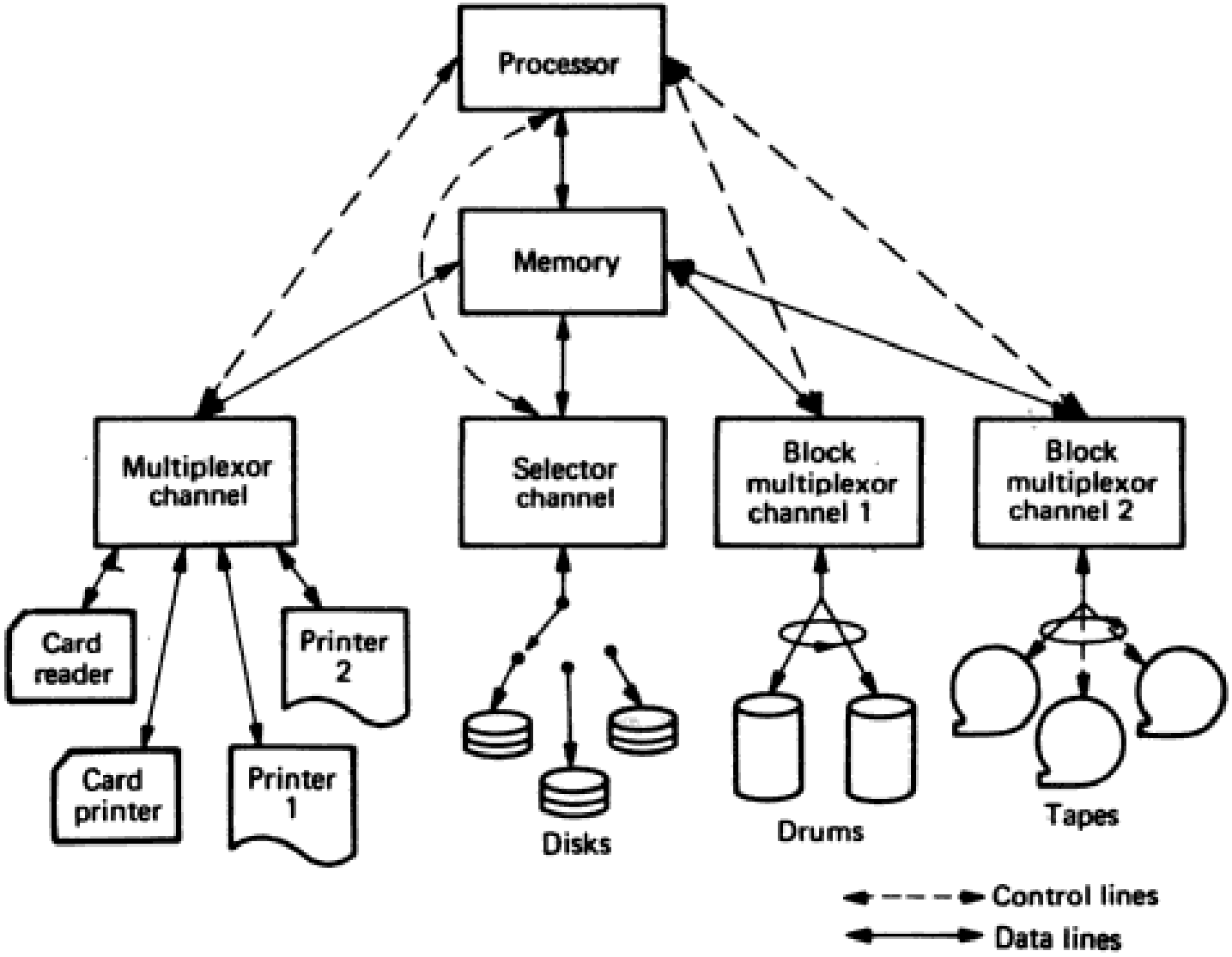


Figure 370-type system configuration

(ii) I/O Programming Concepts:

- The CPU communicates with the I/O processor to perform certain actions such as START I/O and HALT I/O.

- The I/O processor communicate with the CPU by means of interrupts.

- I/O processor interprets its own set of instructions.

- I/O channel instructions are called I/O commands.

- The program contains set of I/O commands are known as I/O programs.

(iii) I/O Processor Structure:

* Memory * Register * Data * Instructions

- Memory:
  - The basic memory unit is a byte and size is up to 2^{24} bytes.
  - For addressing , the I/O channels uses a 24-bit absolute address.
- Register:

  - The I/O channel has no explicit registers, but does have an instruction address register and a data counter .

  - some I/O devices have internal registers like CPU has.
- Data:

  - Primarily, only character data can be handled.

  - Some I/O devices may include some types of code conversions on the data

    Eg: code conversions from EBCDIC to BCD

  - I/O Processor also uses some data pertaining to the state of an I/O device and may read and act on this type of data

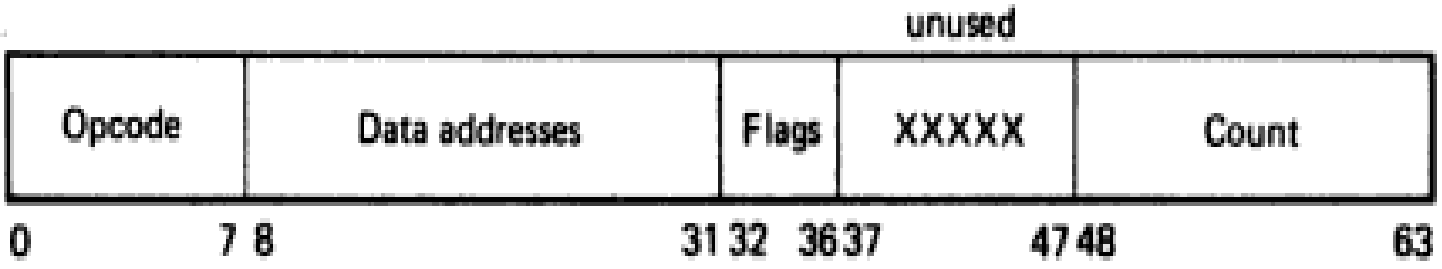
.

- Instructions:

  - There are three basic groupings of I/O commands.

  1. Data Transfers: read; read backwards;write;sense
  2. Device control: Control(page eject, tape rewind , etc)
  3. Branching: transfers of control within the channel program

  - The Channel fetches the channel commands [Channel Command Words(CCW)] from memory and decodes them according to the following format.



Opcode(bits 0-7):

It indicates the command to be performed

- it actually consists of two parts

2 to 4 -→ operation bits

4 to 6--→ modifier bits (varies for each type of device)

Data Address(bits 8-31):

- it specifies the beginning location of the data field referenced by the command.

Count field:

- it specifies the byte length of the data field.

The data address and count are used primarily for the data transfer-type commands.

The flag bits further specialize the command. The principal flags are:

1 The *command chain* flag (bit 33) denotes that the next sequential CCW is to be executed on normal completion of the current command. (*Note:* under default conditions, i.e., all flags = 0, the channel stops at completion of current command.)

2 The *data chain* flag (bit 32) denotes that the storage area designated by the next CCW is to be used with the current command, once the current data area count is exhausted.

3 The *suppress length indication* flag (bit 34) suppresses the indication to the program of an incorrect length (see Figure 2-12).

4 The *skip* flag (bit 35) specifies suppression of transfer of information to storage.

5 The *programmed controlled interruption* flag (bit 36) causes the channel to generate an interruption condition when this CCW takes control of the channel.

(iv) Special Features of I/O programming:

The channel has an internal register that acts as the **instruction address register**.

Also three specific words of memory are used for status information.

1. Channel Address Word (CAW):

contains the address of the first instruction to be executed which is referred only during execution of START I/O.

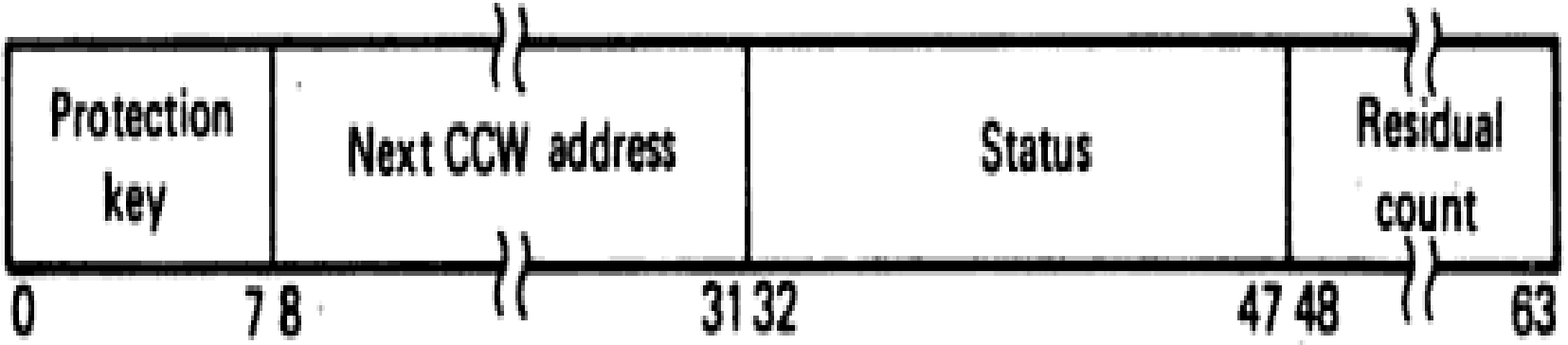
2. Channel Status Word(CSW):

- it is a double word containing coded information indicating the status of the channel

3. Channel Command Word(CCW):

- fetches the channel commands from the memory and decides according to the format of the CSW.

The format of CSW



Key–protection key being used by channel

Address–address of next channel command

Status–e.g., building on fire, I/O completed, I/O error occurred, etc.

Count–how many bytes of the last CCW were not processed? (usually zero unless the channel abnormally terminated an I/O operation)

(v) Communication between the CPU and the Channel:

- The CPU and the Channel are usually in a Master/Slave relationship.

- This means that the CPU tells the channel when to start and commands it to stop or change what it is doing.

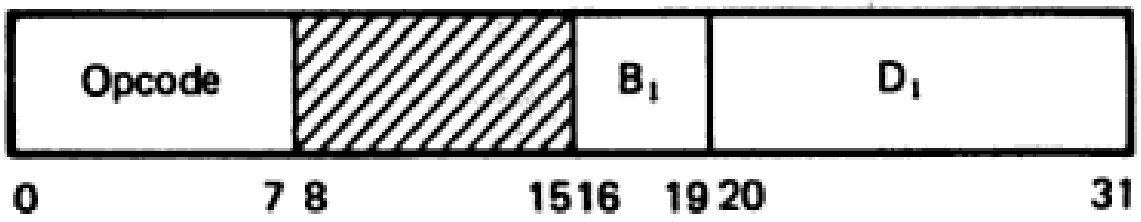
- The channel cannot usually start operations unless instructed by the CPU.

-There are 2 types of communications between the CPU and the Channel.

i) CPU to Channel : I/O instructions initiated by the CPU.

ii) Channel to CPU: interrupt initiated by the channel.

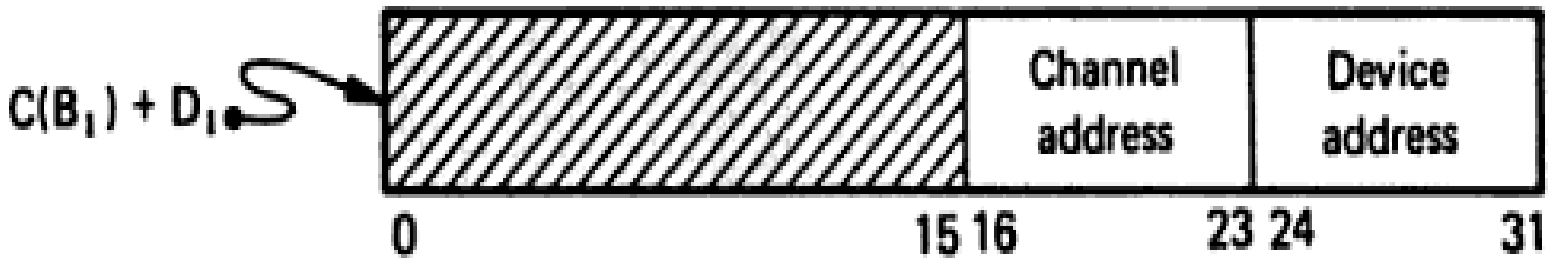
- All CPU I/O instructions has the following format



B1 – The Channel numbers

D1 – The Device numbers

- Bits 16-23 of the sum contain the channel address,
- while bits 24-31 contain the device on the channel



- we are mainly concerned with 3 CPU I/O instructions. They are

* START I/O
* TEST I/O
* HALT I/O

* START I/O(SIO):

- Two items are needed to start I/O:

  (i) The Channel and Device number

  (ii) The beginning address of the channel program

eg. : SIO X00E-→specifies channel number 0 and device number OE

Location 72-75 in memory contains CAW, which specifies the start of the channel program.

* TEST I/O (TIO):

The CPU indicates the state of the addressed channel and device by setting the Condition Code(busy or not).

* HALT I/O(HIO):

Execution of the current I/O operation at the addressed I/O device and channel is abruptly terminated.

- After executing an SIO or a TIO the CPU gets a condition code of either:

    8 -- ok(not busy)

    2 -- busy

    1 – not operational

    4 – indicates that lot more to tell us in the CSW which was just stored at location 64.

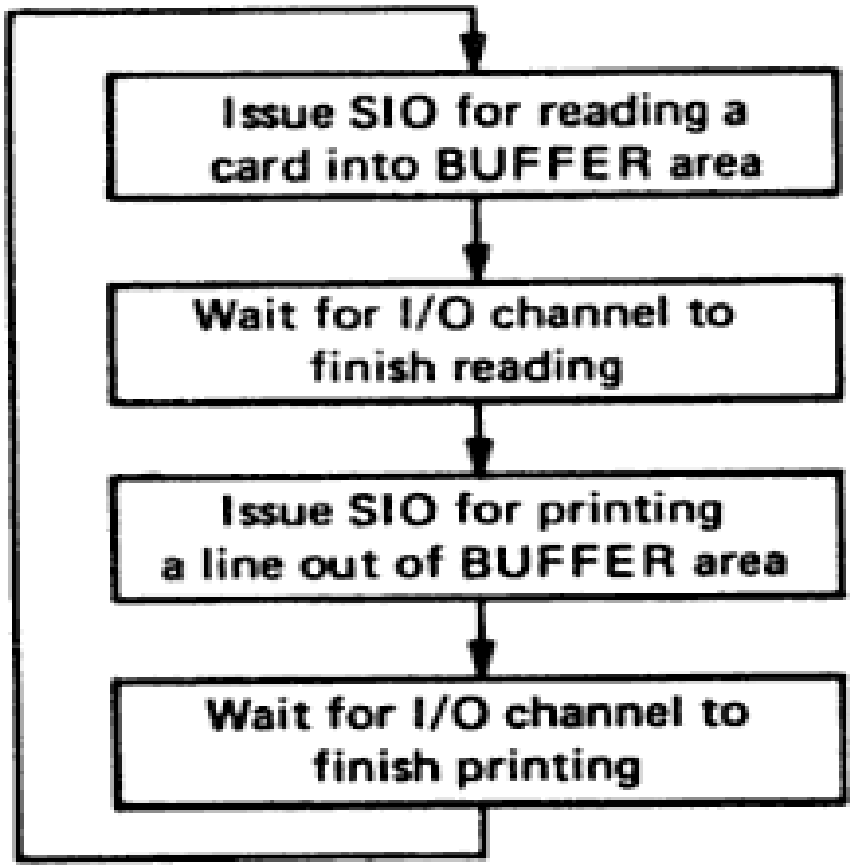
- The channel status word(CSW) provides the detailed status of an I/O device or the conditions under which an I/O operation has been terminated.

- An SIO causes I/O to start only if the channel returns a Condition code of 8. If any other CC is returned, the channel has rejected the I/O request.

- Although the I/O interrupt mechanism has some powerful capabilities, it is not needed to perform simple I/O processing. Instead by using TIO repeatedly, CPU can come to know the end of I/O operation.
- But in some I/O operation like printing a line in PRINTER we find it inefficient.
- It takes 60 ms to print, and there could be 30,000 TIO instructions executed, because, it takes only one micro second to execute an TIO(and every time we get CC=2(busy).
- In this situation, interrupt mechanism is more appropriate.

(vi) I/O Example using Single Buffering:

- We wish to read and print a series of cards.
- One simple strategy is illustrated in the following fig:



Single-buffering program for reading a card and printing

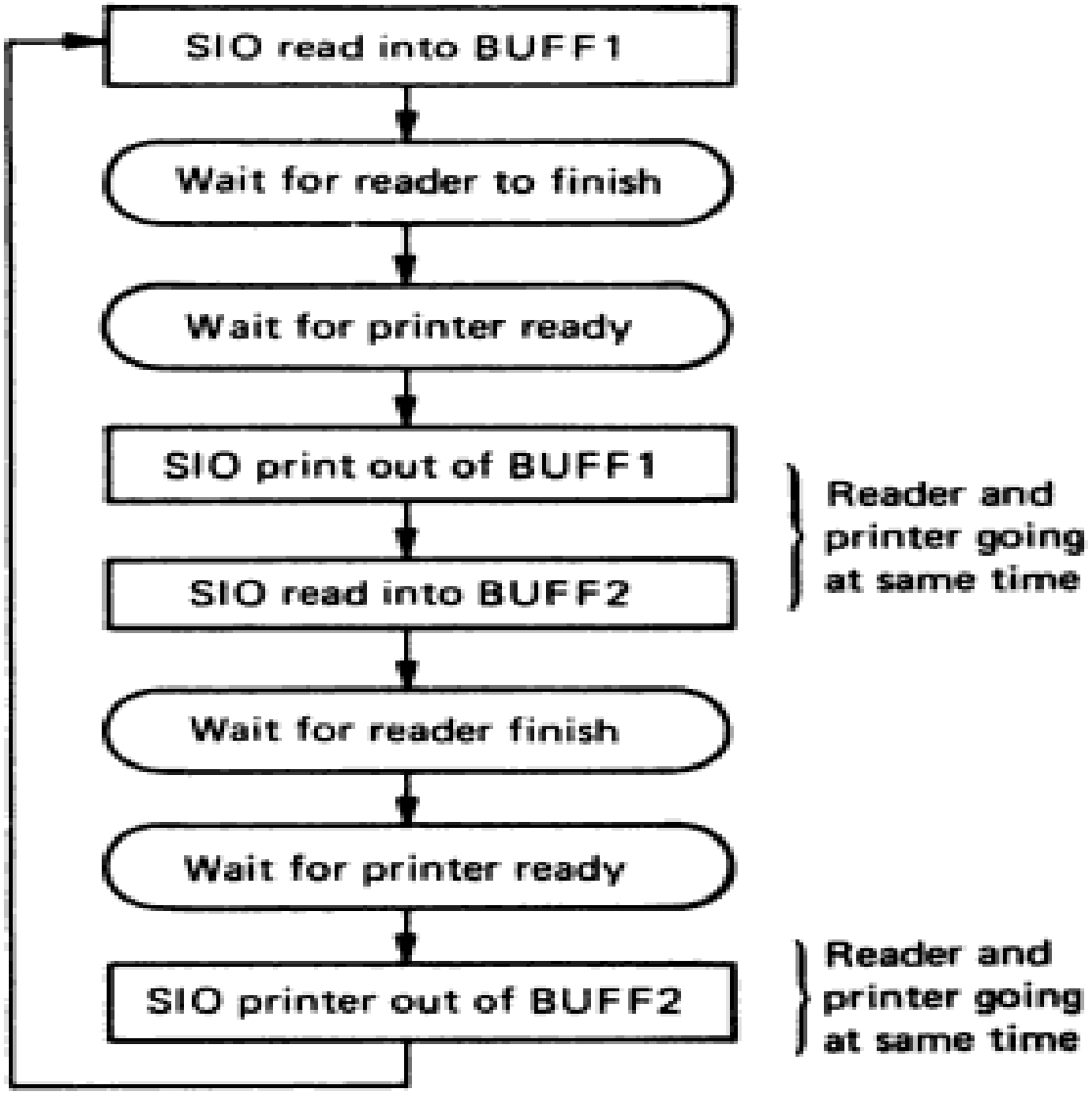
- This sample program does not really take advantage of the I/O channels, because
- whenever the CPU is running, the channels are idle
and whenever a channel is running , the CPU is idle(issuing TIOs).
- Only advantage is that possibility of I/O errors is very minimum.

(vii) I/O Example using double buffering:

- To increase the number of cards processed for minutes, double buffering technique is used.

- With double buffering , we first read a card into buffer area 1; then , while printing out this area, we read the next card into buffer area 2;when we are finished printing buffer area1, we then can start reading into buffer area 1 and repeat this process

GENERAL STRATEGY OF CPU PROGRAM USING DOUBLE BUFFERING



- Double buffering for reading and printing cards

We first read a card into buffer area 1;

Then, while printing out this area, we read the next card into buffer area 2;

When we are finished printing buffer area 1, we then can start reading into buffer area 1;

And repeat this process.

(viii) Multiple card Buffering:

- This buffering is useful when we read more than one cards.

- This technique supports maximum 60 cards for reading and printing at a time.

Unit - II

- Memory Management
- Single contiguous memory management
- Paged Memory Management
- Demand Paged Memory Management
- Segmented Memory Management
- Segmented and Demand Paged Memory Management
- Other Memory Managements

Memory Management:

- It is one of the module of an Operating system.

- It is concerned with the management of primary memory

- The processor directly access the instruction and data from main memory.

Four Functions of Memory Management:

1. Keep track of which segment of memory is in use & by whom.

2. Deciding which processes are to be loaded into memory when space becomes available.

In multiprogramming environment it decides which process gets the available memory, when it gets it, where does it get it, & how .

3. Allocate the memory

4. De-allocate the memory when the process terminated.

Memory Management Techniques:

1. Single Contiguous Memory Management
2. Partitioned Memory Management
3. Relocation Memory Management
4. Paged Memory Management
5. Demand-paged Memory Management
6. Segmented Memory Management
7. Segmented and Demand-Paged Memory Management
8. Other Memory Management
   - Swapping
   - Overlays

Contiguous Storage Allocation:

- Each program had to occupy a single contiguous block of location.

Non-contiguous storage Allocation:

- A program is divided into several blocks or segments

- These blocks (or) segments may be placed throughout main storage.

- not necessarily adjacent to each other.

1. Single Contiguous Allocation:

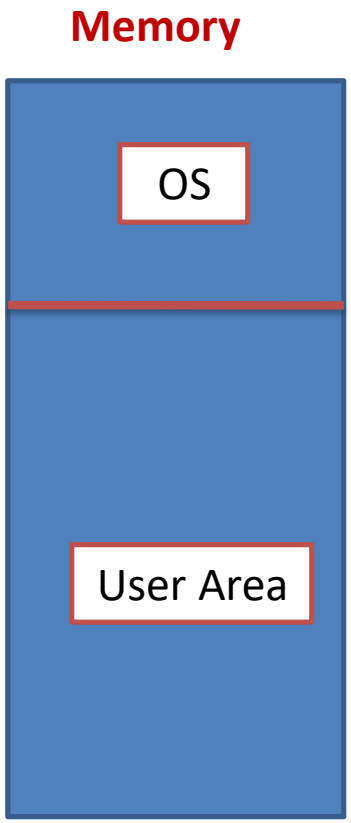
- It is a simple memory management scheme

- The main memory is usually divided into two partitions
    * A portion of memory is permanently allocated to the os
    * Another one for the user process.

- It requires no special h/w

- There is no multiprogramming

- The Job uses only a porting of the allocated memory.



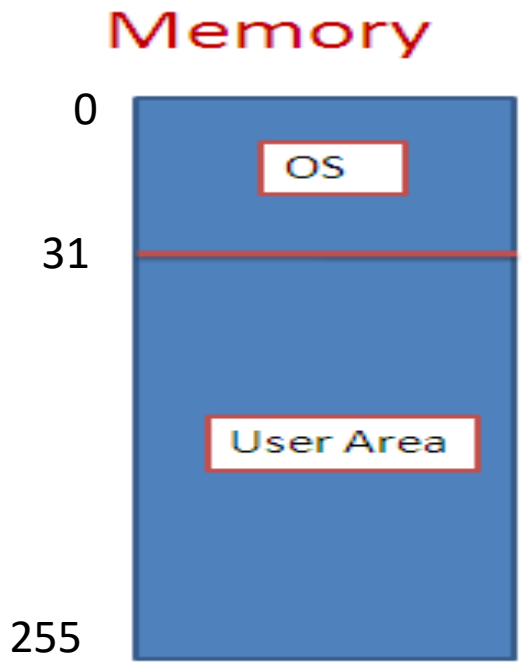
Example:

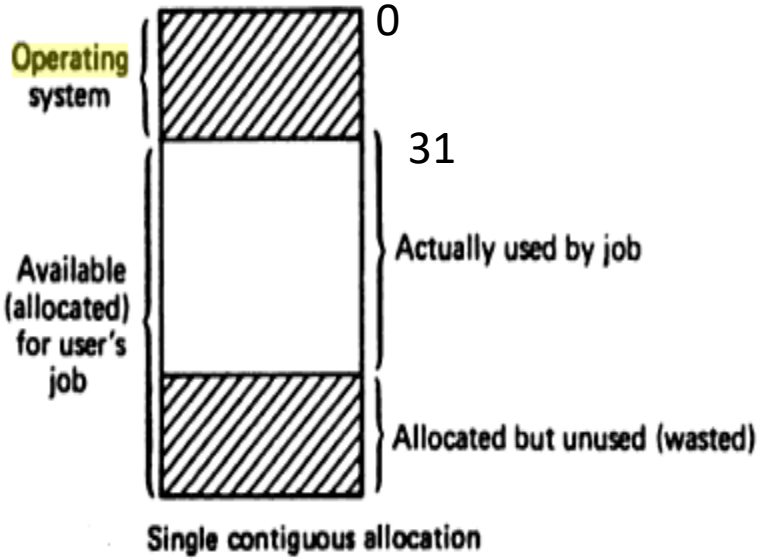
Memory Size = 256 k Bytes

OS require = 32 k bytes

User Area = 224 k bytes

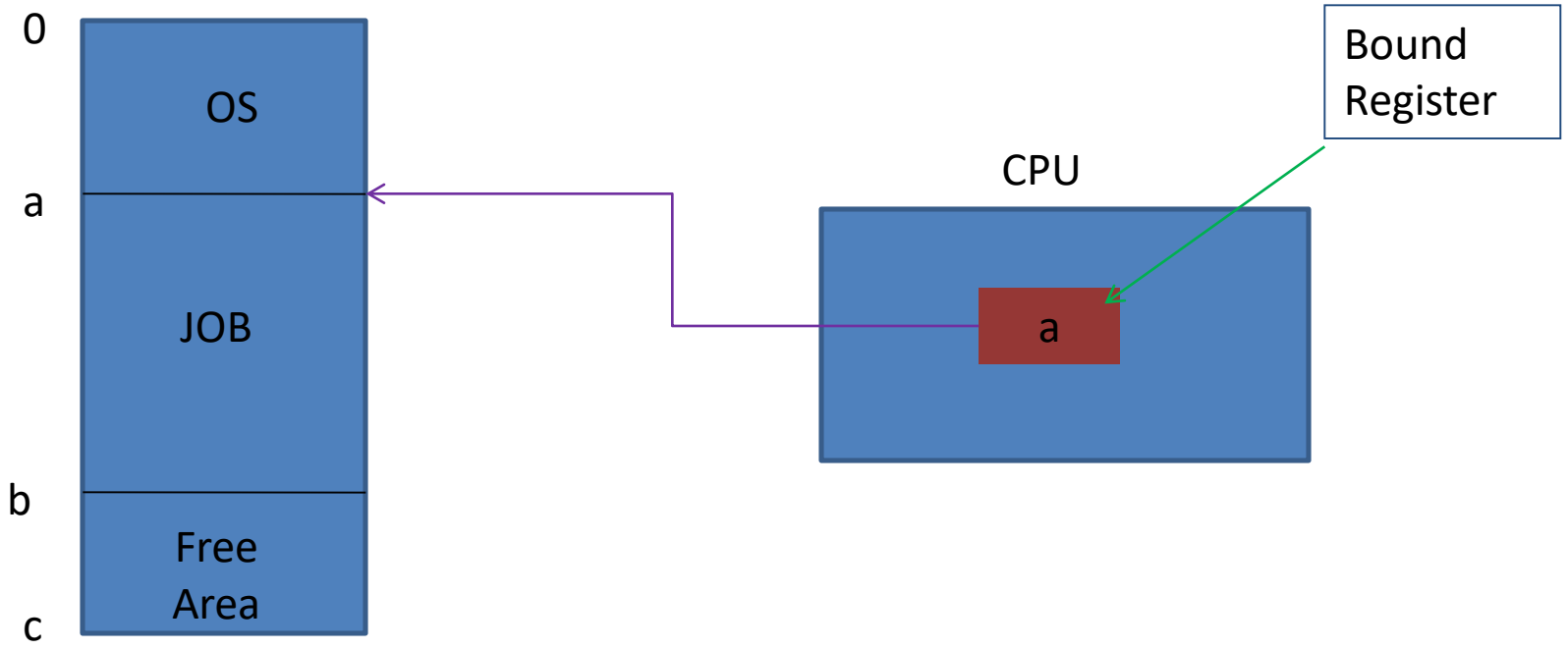
(1 kb=1024 bytes)





Boundary Register:

- It contain highest address used by os.
- During the execution of a program , the memory protection hardware compute every address used in the program with the content of register.



-if the user tries to enter the os, the instruction is intercepted and job terminates with an appropriate error message.

(i) Memory Management Functions for Single Contiguous Allocation:

1. keep track of memory
2. determining the policy
3. allocating of memory
4. deallocation of memory

(ii) Hardware Support:

- no special h/w is required for single contiguous allocation .

(iii) Protection through Bound Register:

- we need to protect the os code and data from changes made by user processes.

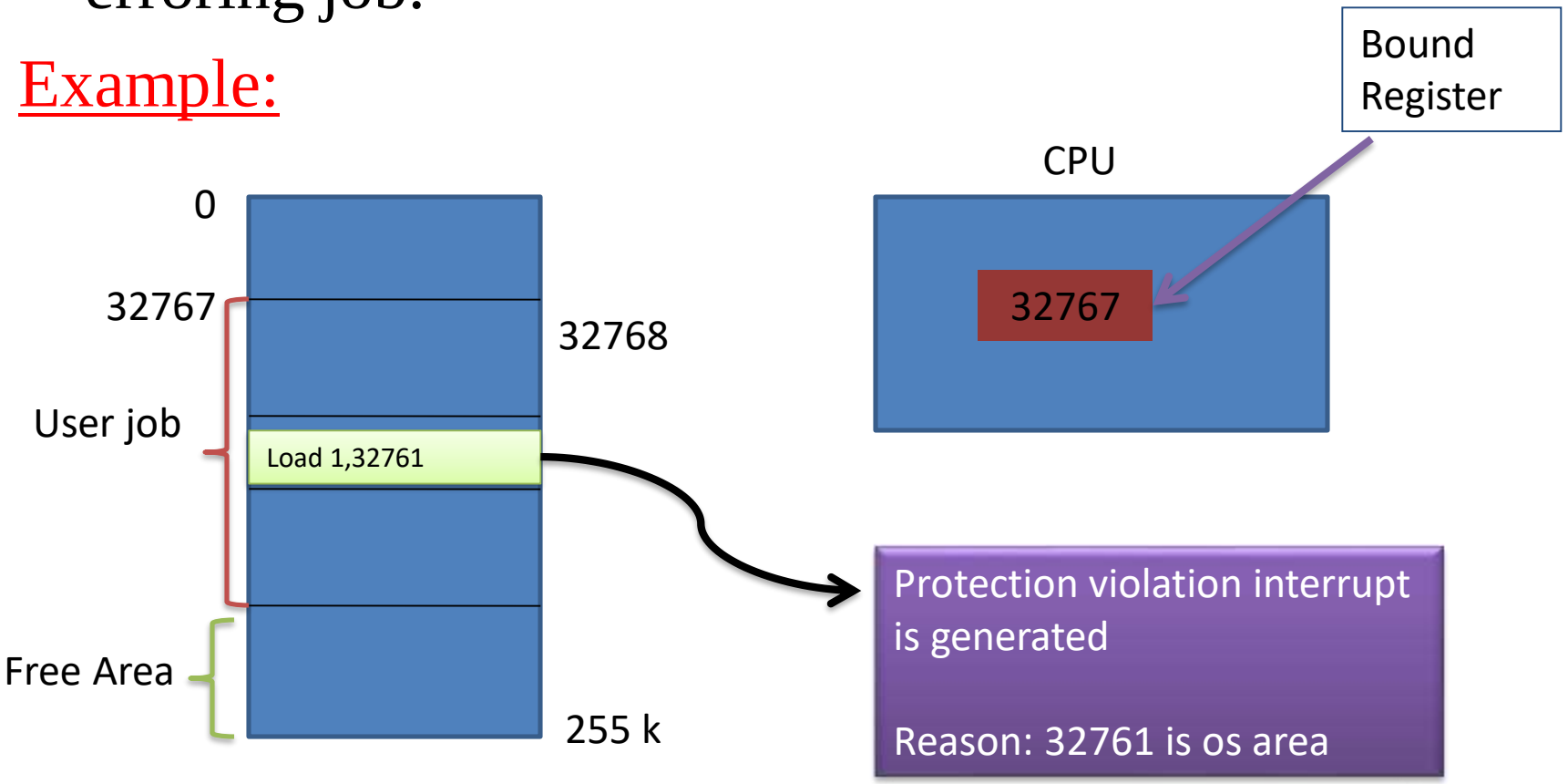
- It can be done by using single boundary register built into the CPU.

Memory production violation interrupt

-This interrupts is generated if an address lies outside the allocated area.

- On sensing this interrupts the os can terminate the erroring job.

Example:



(iii) A flow chart of a single contiguous Allocation:

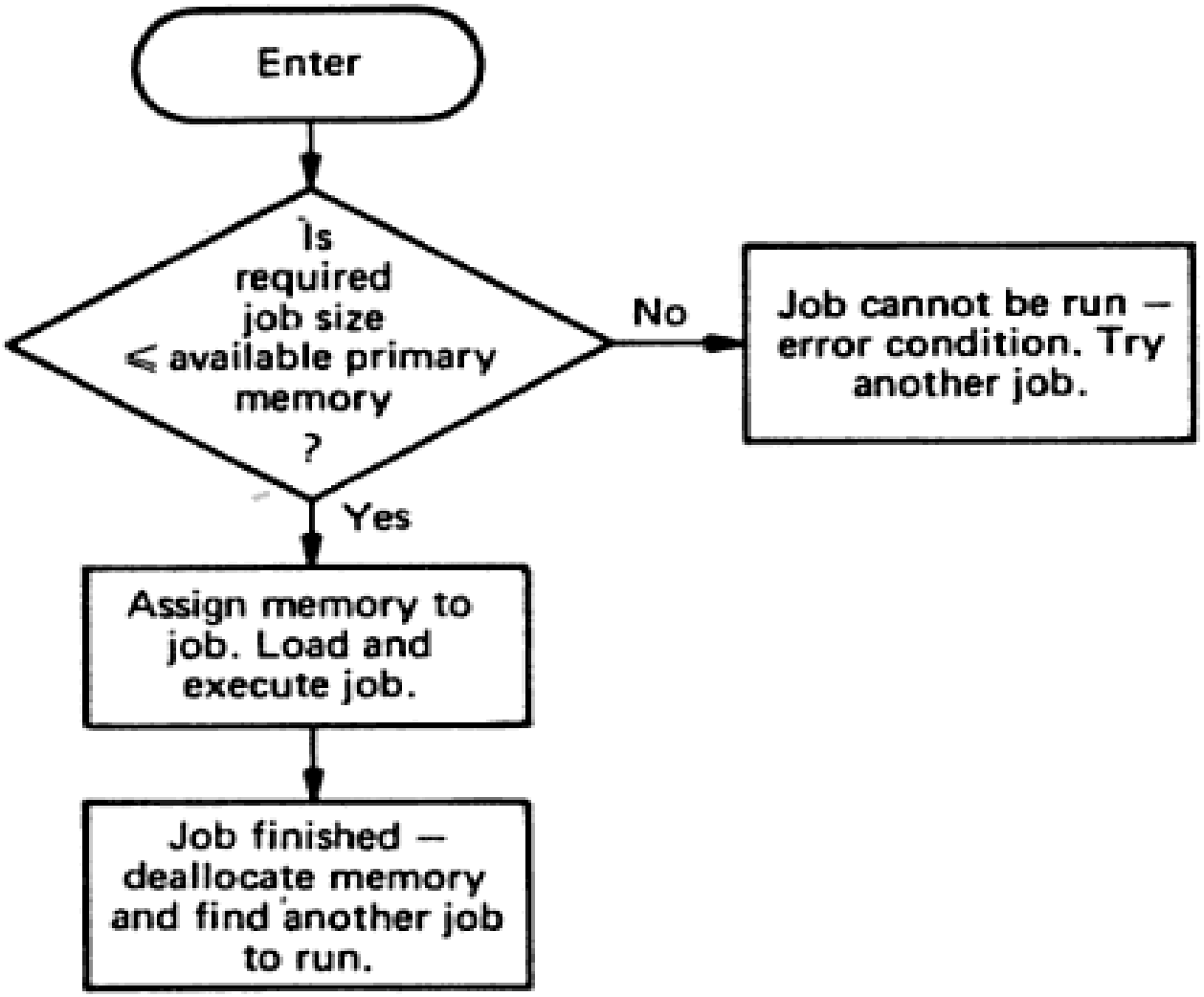


Figure Single contiguous allocation

(iv) Advantages

- Easy to understand
- Very Simple
- OS require little amount of memory

(v) Disadvantages

- Poor utilization of memory
- Poor utilization of CPU [waiting for I/O]
- Job size is limited

Memory Management Functions for Partitioned Allocation

2.2 Partitioned Allocation

-Main memory is divided into separate memory register or memory partitions.

- Each partition holds a separate job

-It supporting “ Multiprogramming”

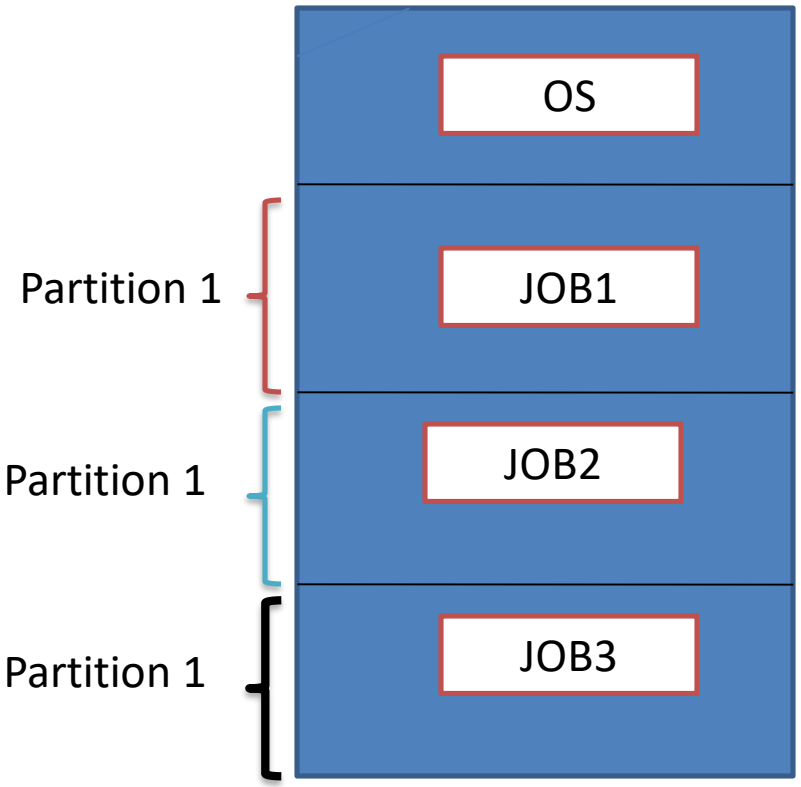
Memory Management Functions for Partitioned Allocation

1. Keeping track of the status of each partition
   [Eg: “in use” or “not in use”, size]

2. Determining who gets memory

3. Allocation an available partition of sufficient size is assigned.

4. De-allocation when the job terminates the partition is indicated “not in use”.


OS
JOB1
JOB2
JOB3
Partition 1
Partition 1
Partition 1

Types of partition

1. Static or Fixed partition specification
2. Dynamic or variable partition specification

1. Static or Fixed partition specification

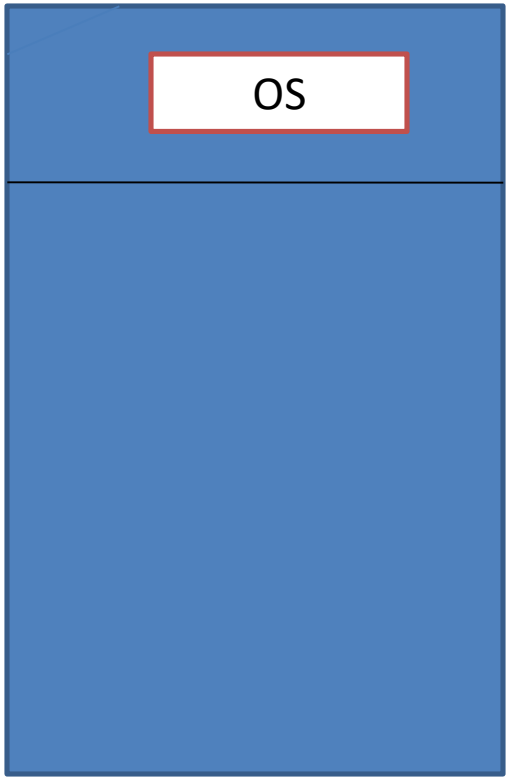
- The memory is divided into partitions prior to the processing of any jobs.
- It is similar to multiprogramming with fixed number of tasks
- The degree of multiprogramming based on number of partition.

(a) Fixed partitioning with equal size
(b) Fixed partitioning with variable size

(a) Fixed partitioning with equal size

- Memory is divided into fixed number of partitions with equal size.

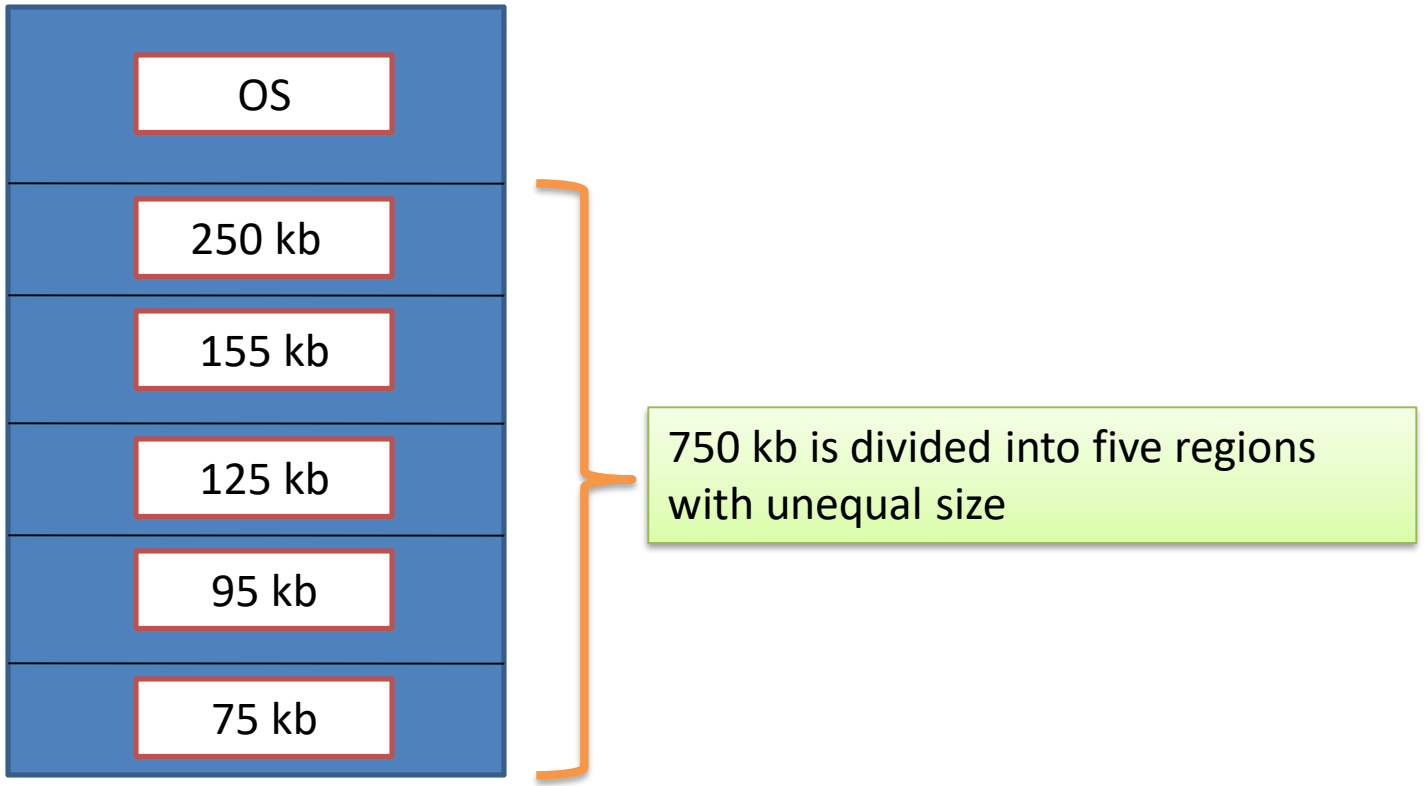
- In this case, any process whose size is less than or equal to partitioned size can be loaded into memory.



750 kb is divided into five regions with equal size

(b) Fixed Partitioning with variable size:

- the memory is divided into Fixed number of partitions with unequal size.



(iii) Advantages

- Multiprogramming
- Efficient CPU utilization

(iv) Disadvantages

- If there is a job whose memory requirement is more than that of any partition, then the job cannot be run

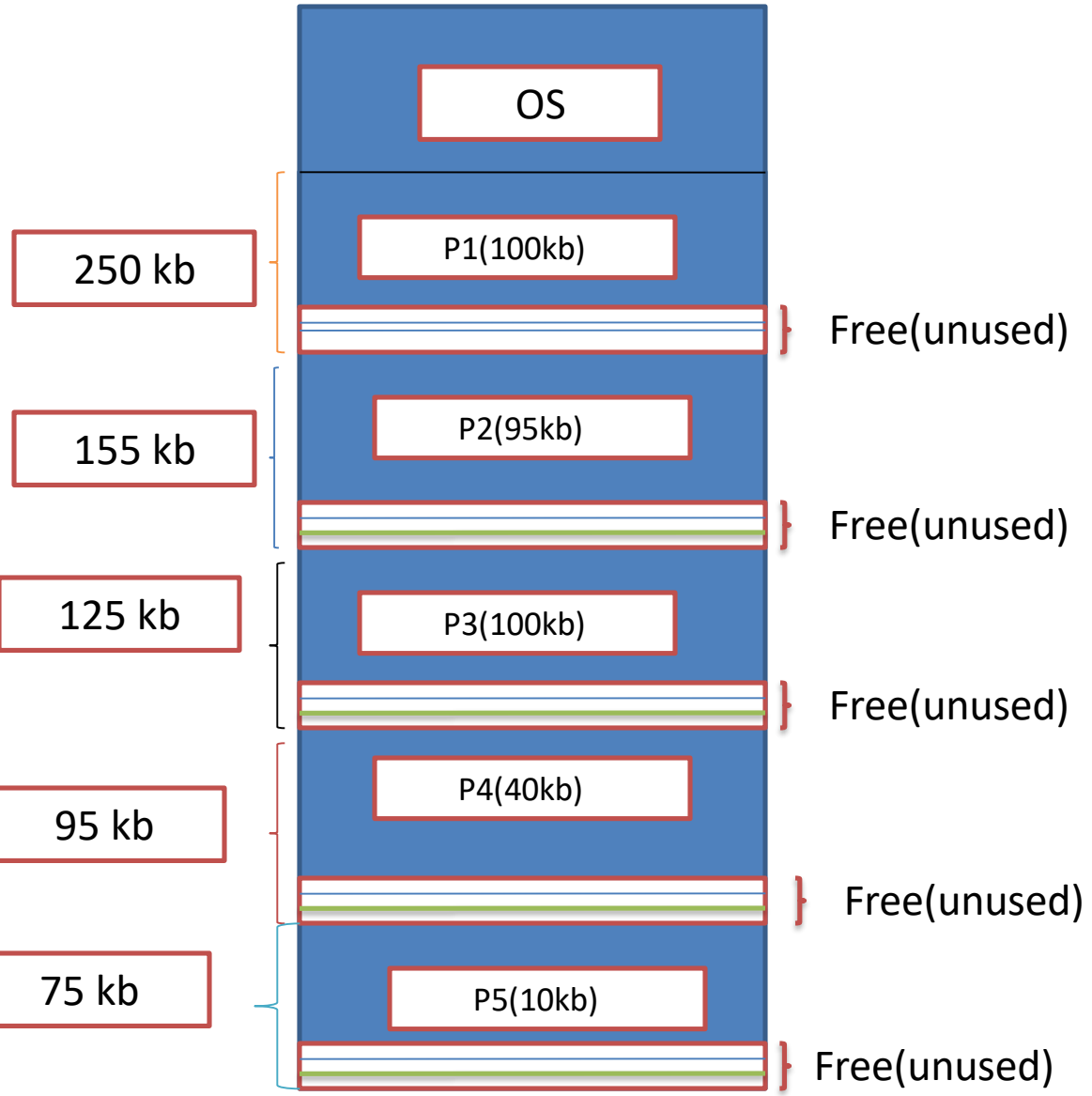
  ie. Job size<= partition size

- Poor memory utilization

JOB QUEUE

Process	Memory
P1	100 kb
P2	95 kb
P3	100 kb
P4	40 KB
P5	10 KB

This Technique is appropriate when the sizes and frequency of the jobs are well known



2. Dynamic Partition Specification:
(or) Variable partition Specification:

- it is also called MVT[Multiprogramming with a variable number of task]

- The partitions are created during job processing.

- The partitions are of variable length and number.

- The operating system keeps a table indicating which parts of memory are available and which parts are occupied.

Allocated Partition Status table

Size	Location	Status

UnAllocated Area Status table

Size	Location	Status

size --→ indicate the size of region or partition

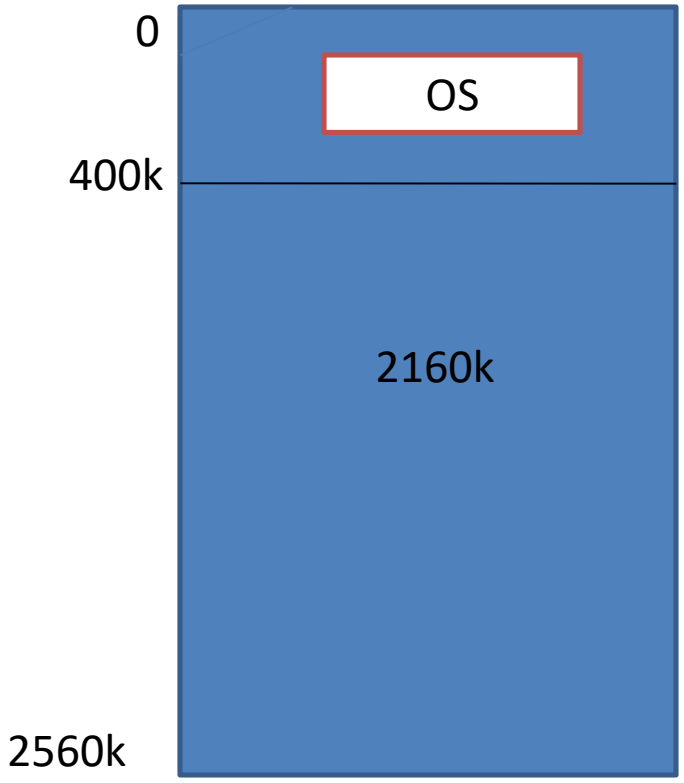
location -→ indicate the starting location of partition

status --→ indicate whether corresponding entry is current in use or not in use

- initially , all memory is available for user processes, and is considered as one large block of available memory.

- When a process arrives and needs memory, we search for hole large enough for this process.

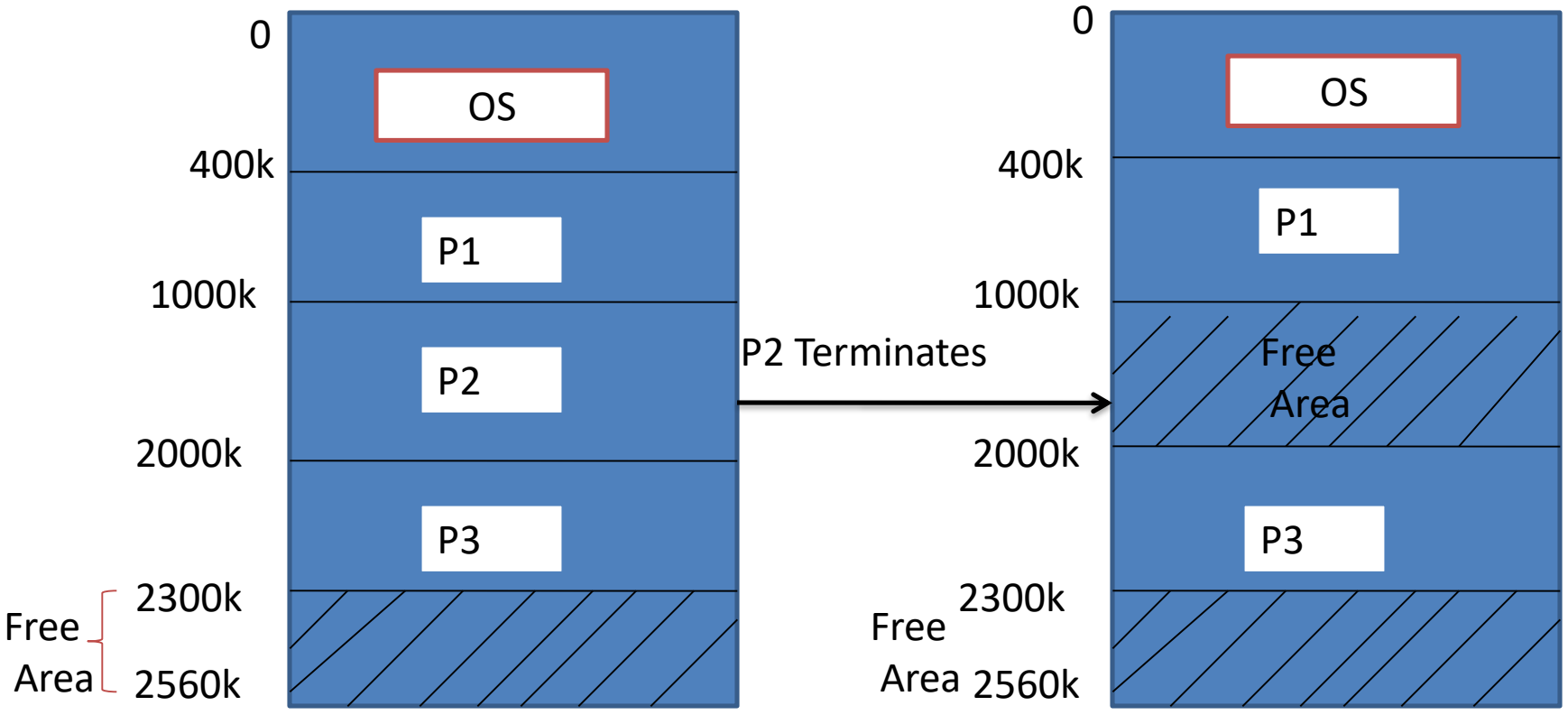
- If we fine one, we allocate only as much memory as is needs.



Available Memory = 2560k
OS = 400 k
User Process = 2160k

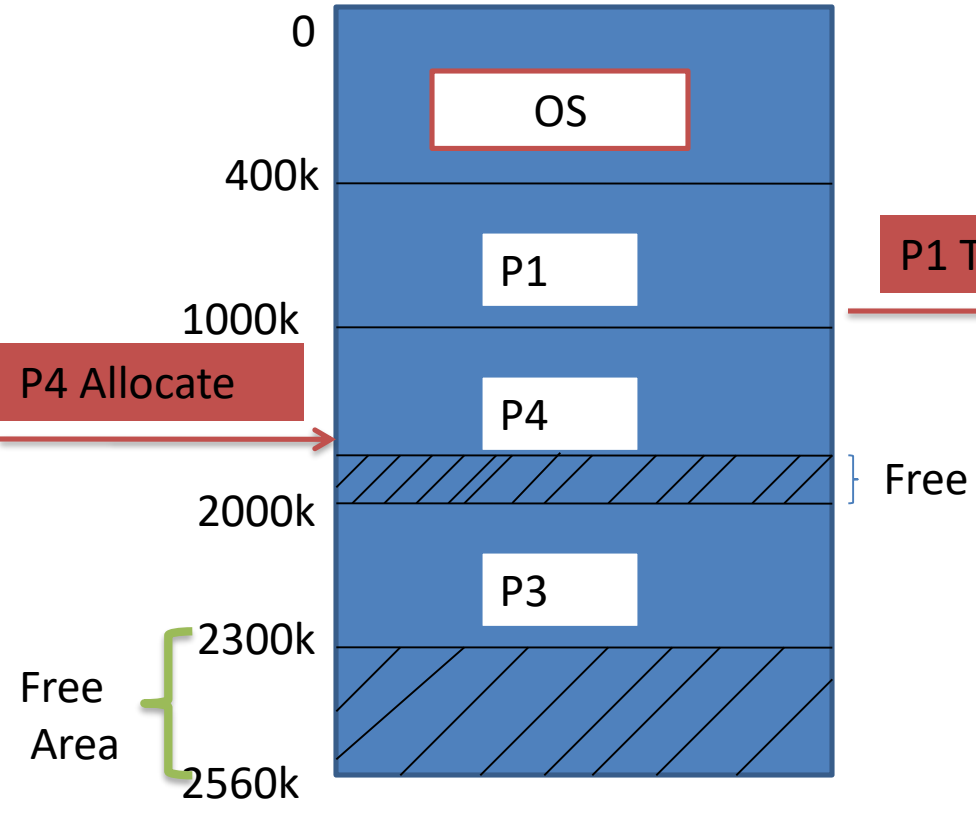
Job Queue

Process	Memory
P1	600k
P2	1000k
P3	300k
P4	700k
P5	500k

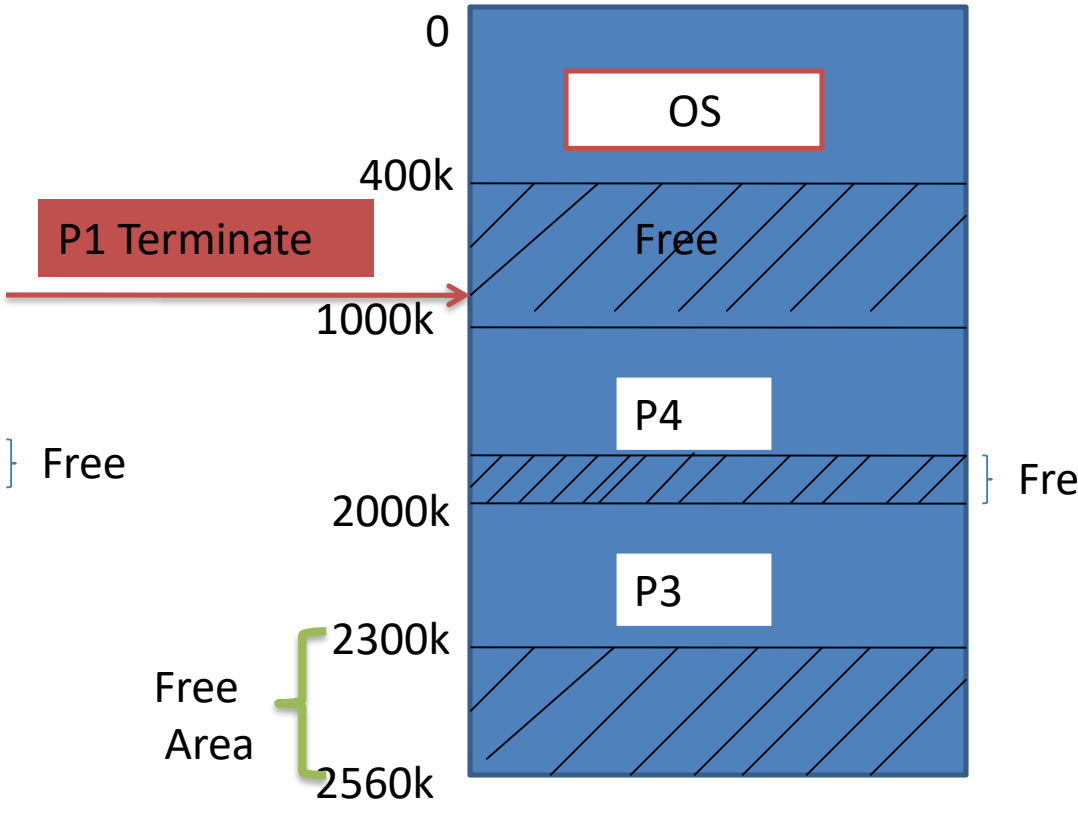


Allocate memory to p1,p2 and p3

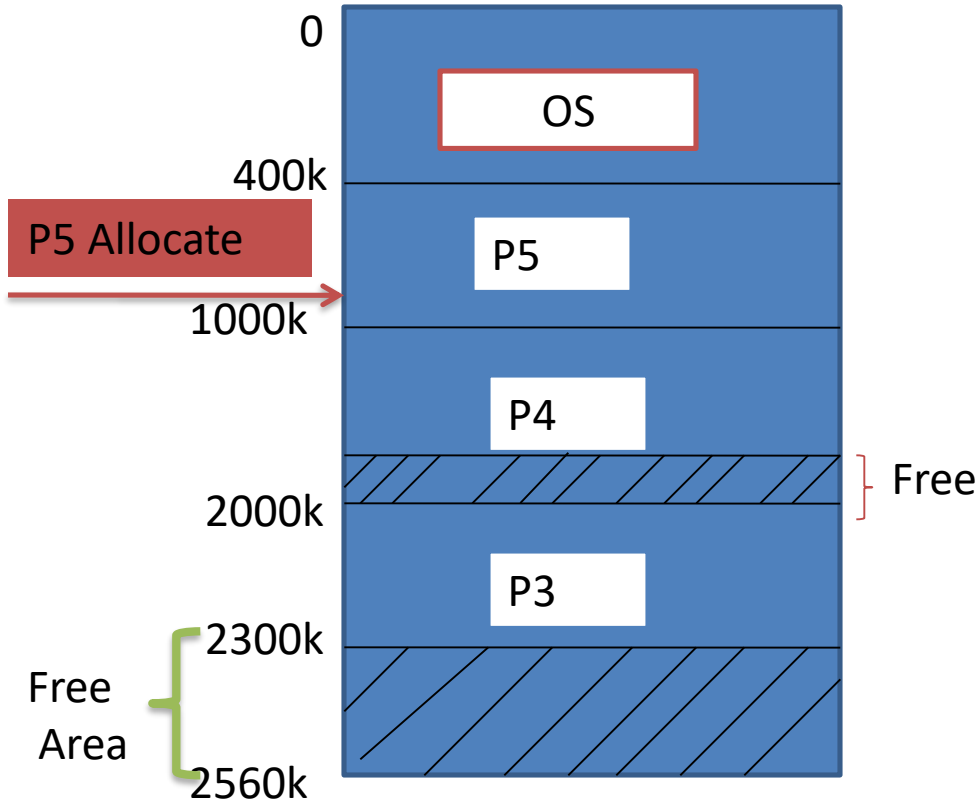
Process p2 terminates and releasing its memory



Allocate Process P4



Process P1 terminates and releasing its memory



Process P5 is scheduled

Allocation Scheme:

- As Processes enter into system, they are put into the queue.

- The os determining the amount of available memory and memory requirements of each process.

- Determining which process are allocated to memory

- After that , the process is loaded into memory.

- When the process terminates then it release its memory

Placement Algorithms:

1 . First Fit Partition Algorithm

2 . Best Fit Partition Algorithm

1. First Fit Partition Algorithm:

- Allocate first hole that big enough.

- Searching can start at the beginning of set of holes.

- We can stop searching as soon as we find a free hole that is large enough.

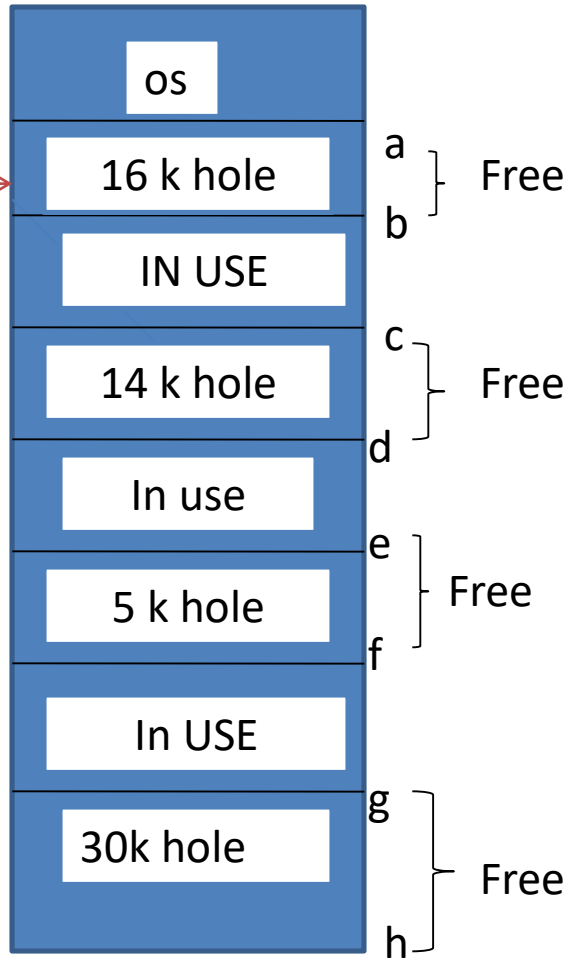
Free Storage List

Start Address	Length
a	16k
c	14k
e	5k
g	30k

Queue

Job request
For 15k

Main Memory



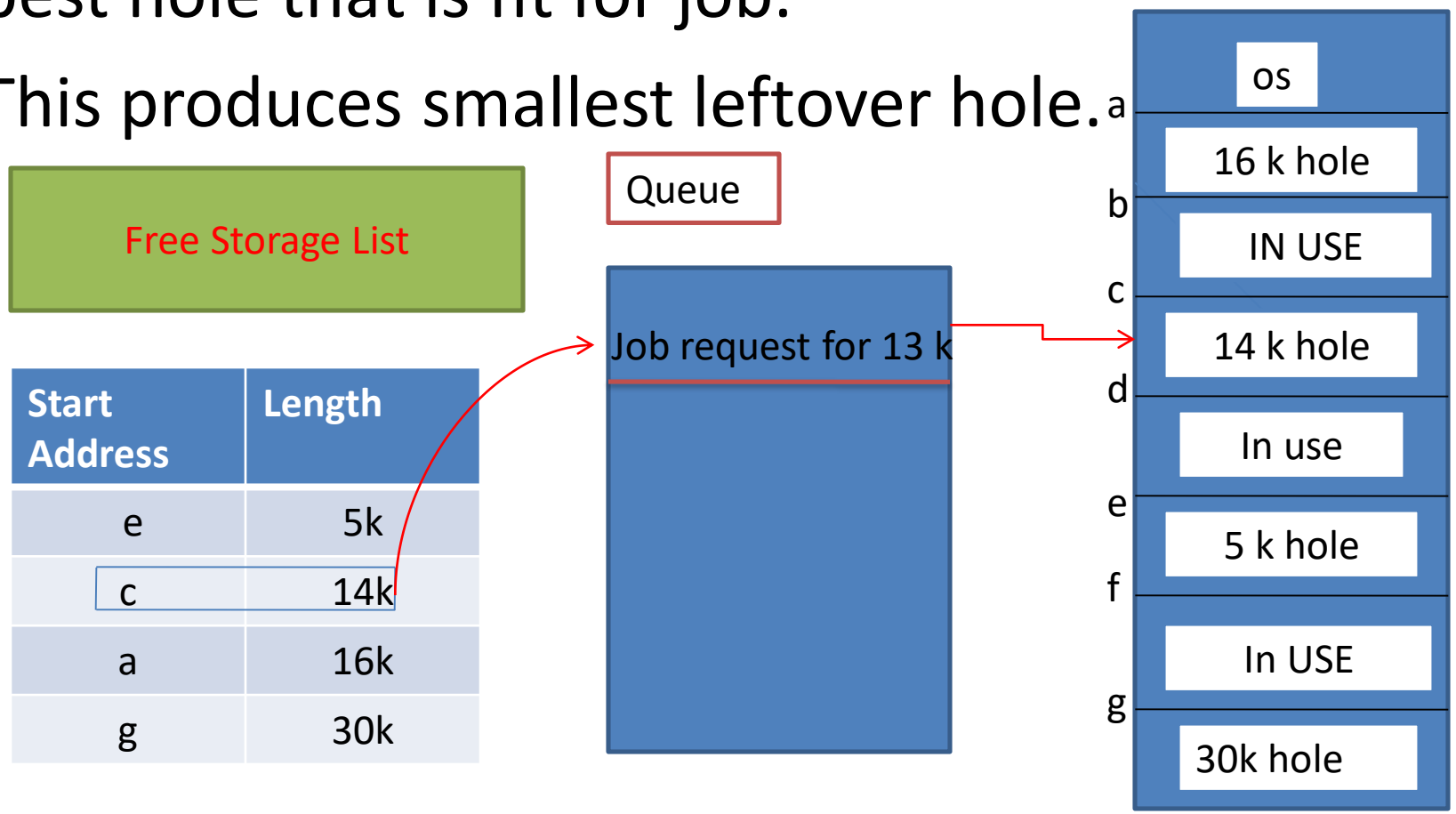
Disadvantages:

-Memory Spaces is wasted highly

- Poor utilization of memory

2. Best Fit Partition Algorithm

- Allocate the smallest hole that is big enough
- We must search the entire list, and select the best hole that is fit for job.
- This produces smallest leftover hole.



Flow chart:

Problems in Partitioned Allocation:

- The main problem in the partitioned allocation is “memory fragmentation”

Memory Fragmentation:

- It implies that the existence of unusable memory areas in a computer system.

Types of Fragmentations:

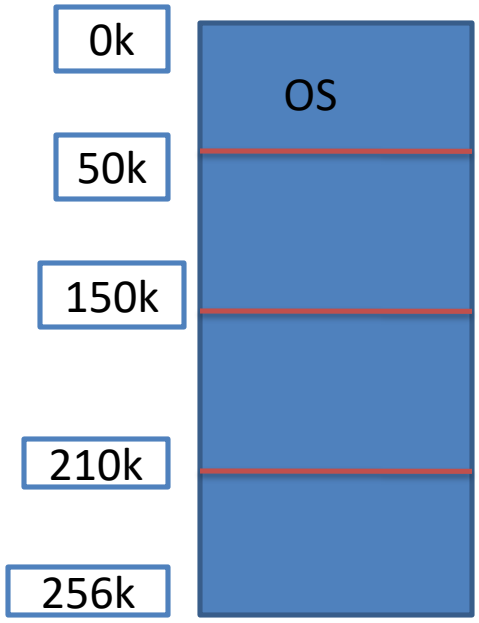
(i) Internal Fragmentation

(ii) External Fragmentation

(i) Internal Fragmentation:

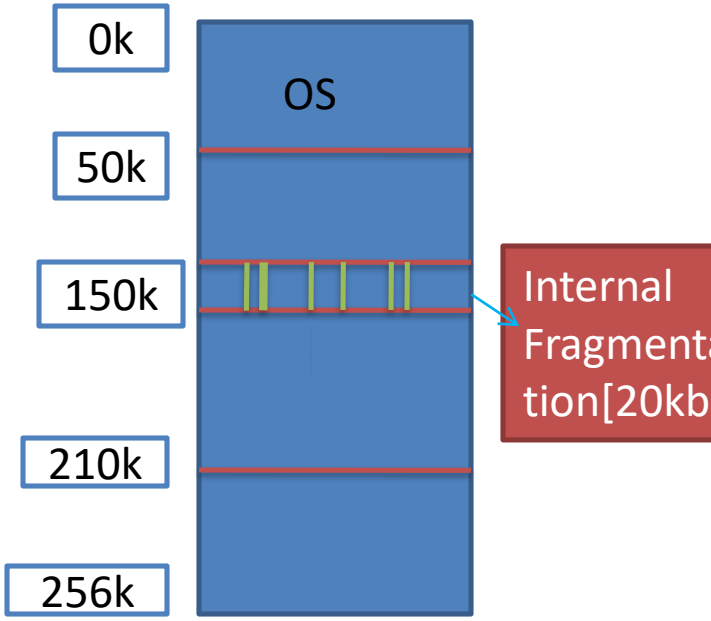
- The allocated memory may be slightly larger than the requested memory
- The difference between these two numbers is “internal fragmentation”.
- The memory that is internal to a partition, but is not being used.

Ex:



job requires

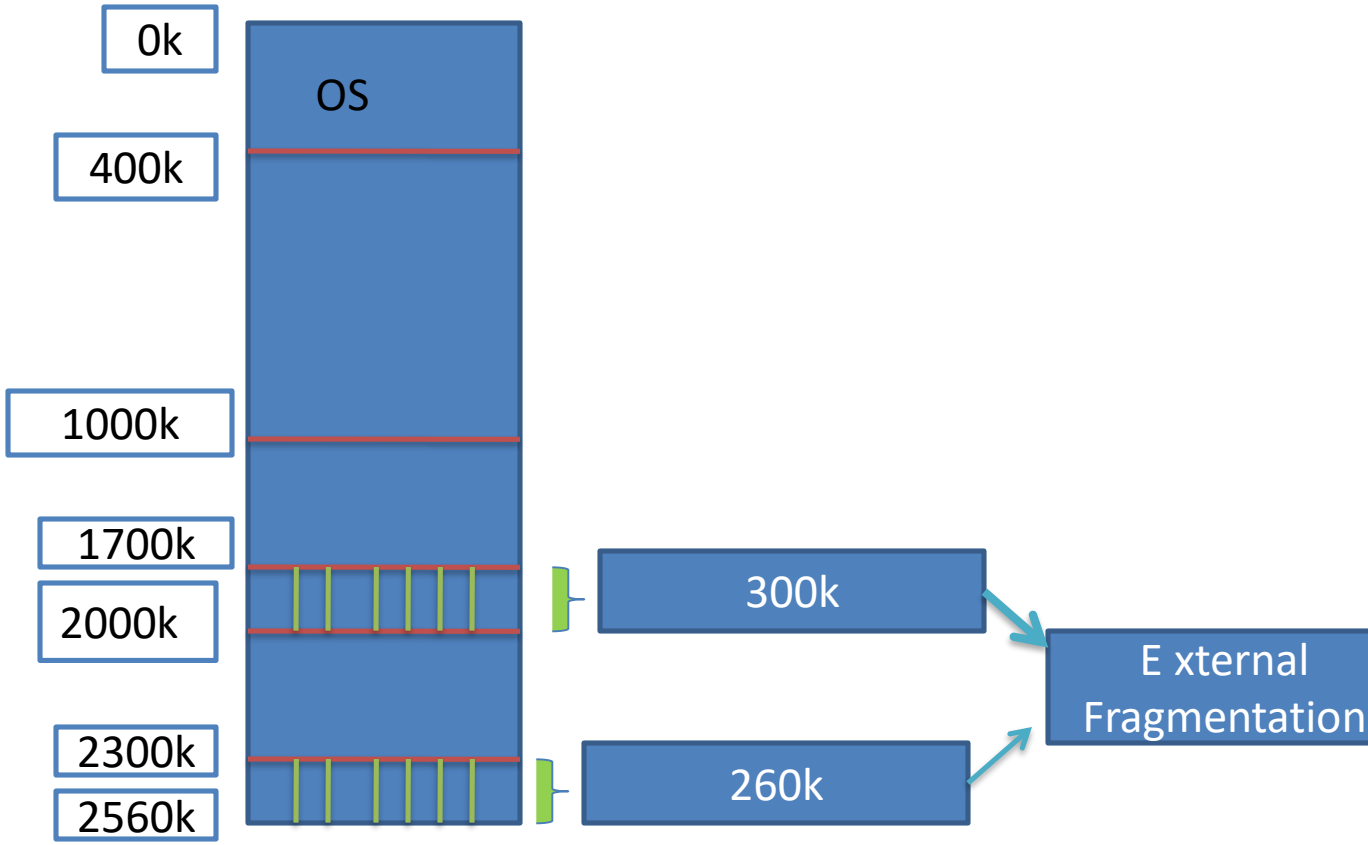
80k



- 20k bytes of the memory allocated to job A is currently unused
- it cannot be allocated to any other job
- this situation is called internal fragmentation.

(ii) External Fragmentation:

- External Fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into larger number of holes.



Process	Memory
P5	5ook

This space is large enough to run process P5 but the free space is not enough

Advantages and Disadvantages of Partitioned Allocation:

Advantages:

1. allows multiprogramming
2. efficient utilization of CPU
3. does not require any special costly hardware
4. algorithm is simple and easy to implement

Disadvantages:

1. Fragmentation
2. It requires more memory than a single contiguous allocation system.
3. a jobs partition size is limited to the size of physical memory.

2.3 Relocatable Partitioned Memory Management:

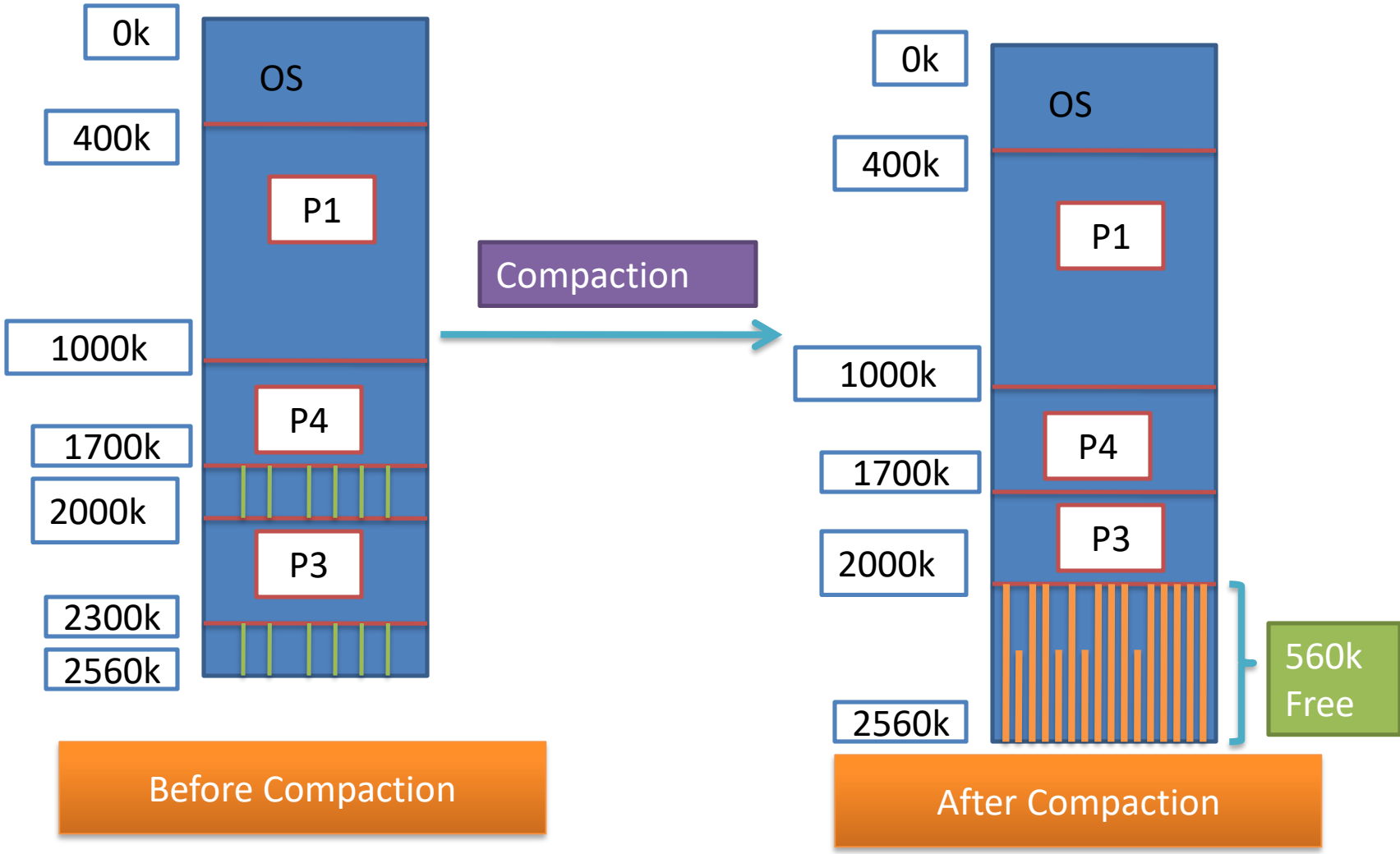
Memory Compaction:

-an one solution to the fragmentation problem is to periodically combine all free areas into one contiguous area.

-this can be done by moving the contents of all allocated partition.

- this process is called “compaction”

Example:


0k
OS
400k
P1
1000k
P4
1700k
2000k
P3
2300k
2560k
Compaction
0k
OS
400k
P1
1000k
P4
1700k
P3
2000k
560k
Free
2560k
Before Compaction
After Compaction

- moving a job's partition doesn't guarantee that the job will still run correctly as its new location.

-This is because there are many location-sensitive items, such as:

(1) base registers
(2) memory referencing instructions
(3) parameter lists
(4) data structures

- the above sensitive items uses address pointers

- to operate correctly, all location, sensitive items must be suitably modified

- for example, in the above figure, P3 is moved from location 2000k to 1700k

- all addresses within the P3 partition must be decreased by 300k

- This process of adjusting location sensitive addresses is called "Relocation"

Problems in Relocation:

- Relocation of job partition can be quite difficult

Reason:

- It is very difficult for an OS to identify the address pointer that must be altered.

- For example, if the number 36400 appeared in P3 partition it is very difficult to indentify whether it is an address pointer to be relocated or variable value.

Solution to Relocation Problem:

Solution 1:

Reload all jobs to be relocated and restart them from the beginning[But it is insufficient because of the computation may have to be repeated.

Solution 2:

Mapped Memory

Two approaches to the Relocation Problem:

1. Data Typing Concepts
2. Dynamic Relocation concept

1. Data Typing concept:

- Physically records the type of value stored in every memory location

Example:

add 2 bits to every word to designate the value type

Eg: 00= integer
01= floating point number
10 = character
11 = Address pointer

- These bits are automatically set by h/w

A=B not only sets A to the value B but also copies the type of information

- In this case , we have to easily identify the address pointer and then manipulated during relocation

Disadvantages:

1. It requires extra bits on every word
2. compaction may be slow.

2 . Dynamic Relocation concept:

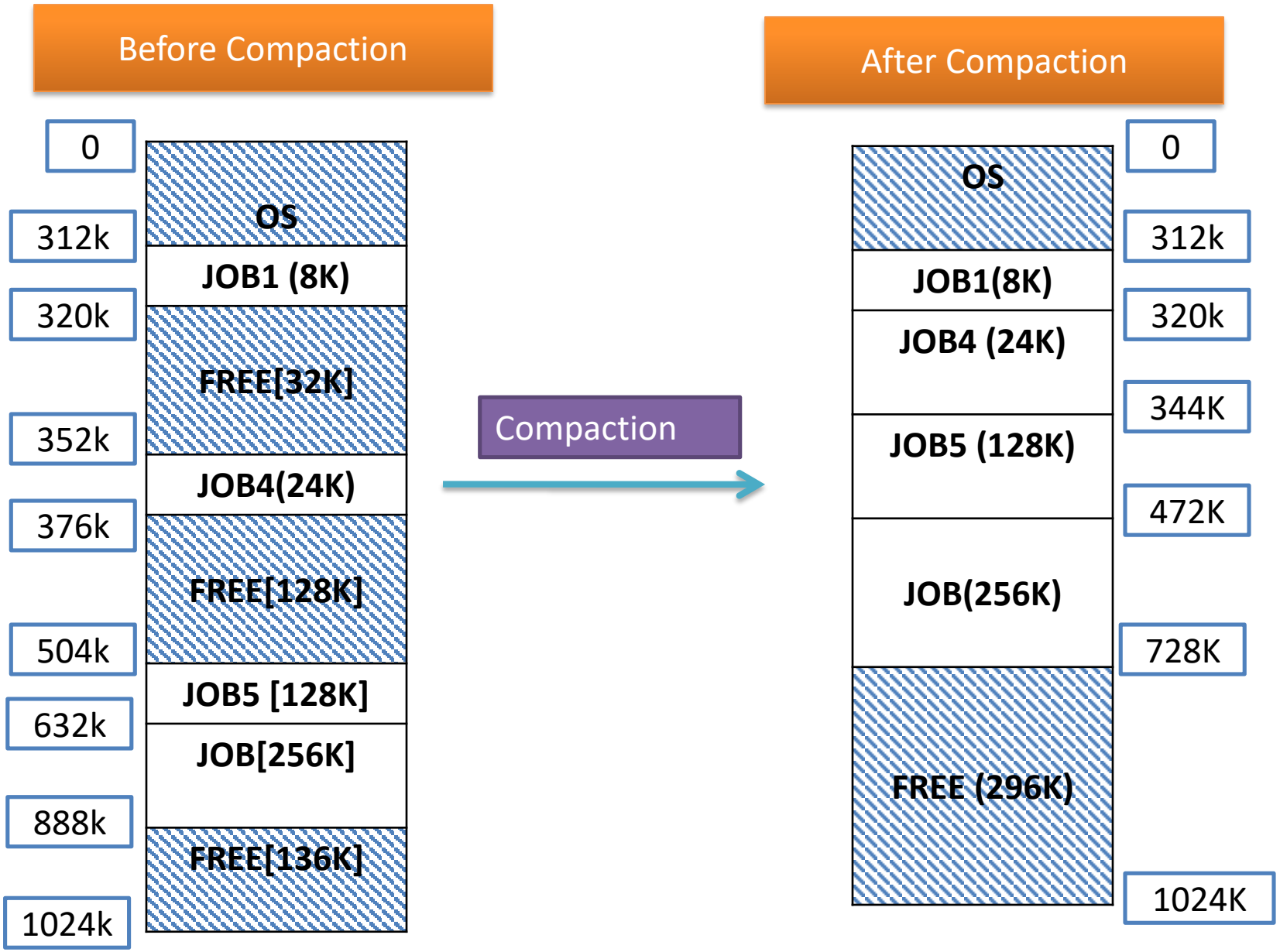
- This method uses two special Privileged Registers such as,
    * Base Relocation Register
    * Bounds Register
- It can be accessed only by the operating system.

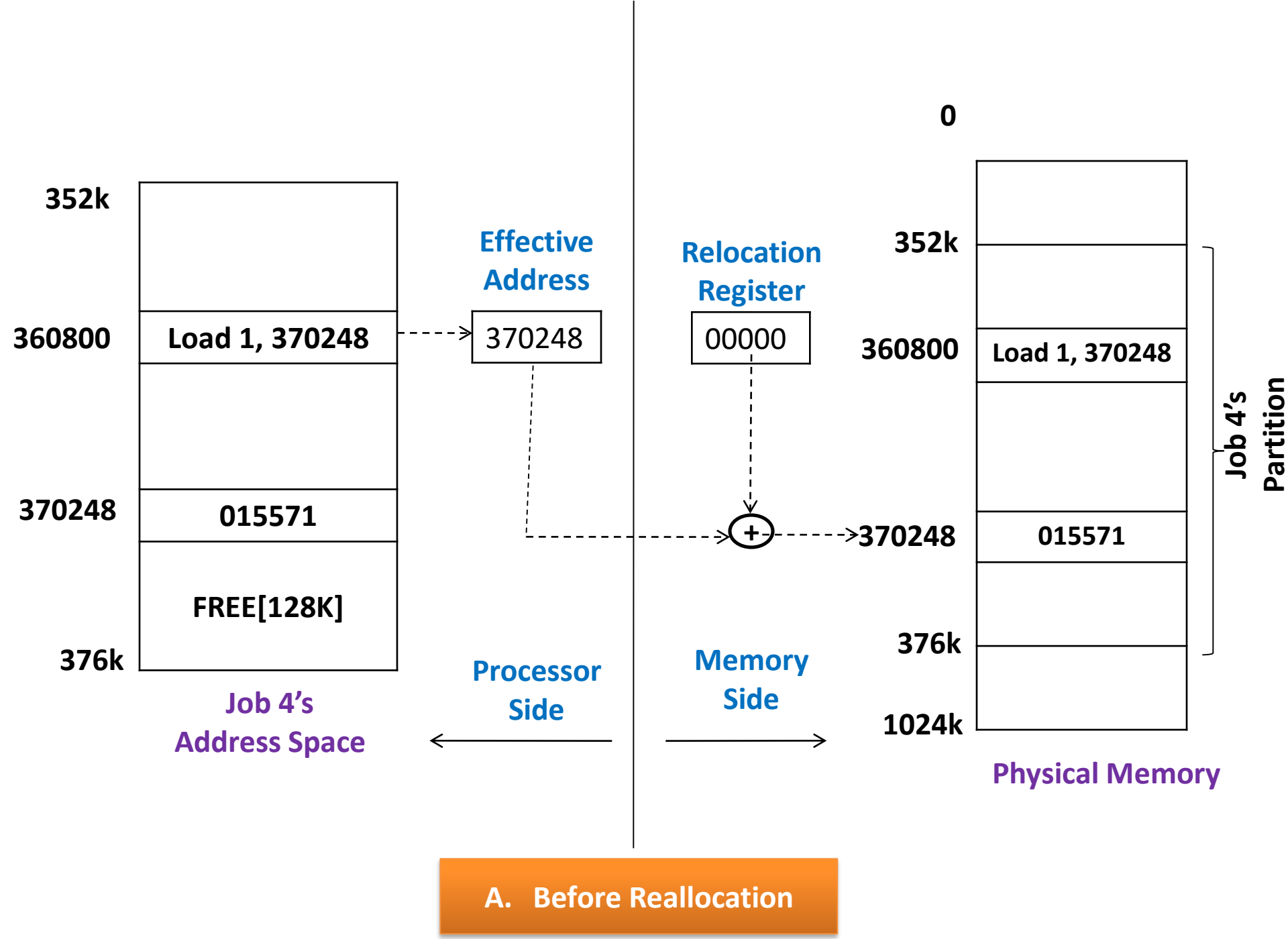
- on every memory reference, the content of the base relocation register is automatically added to the effective address

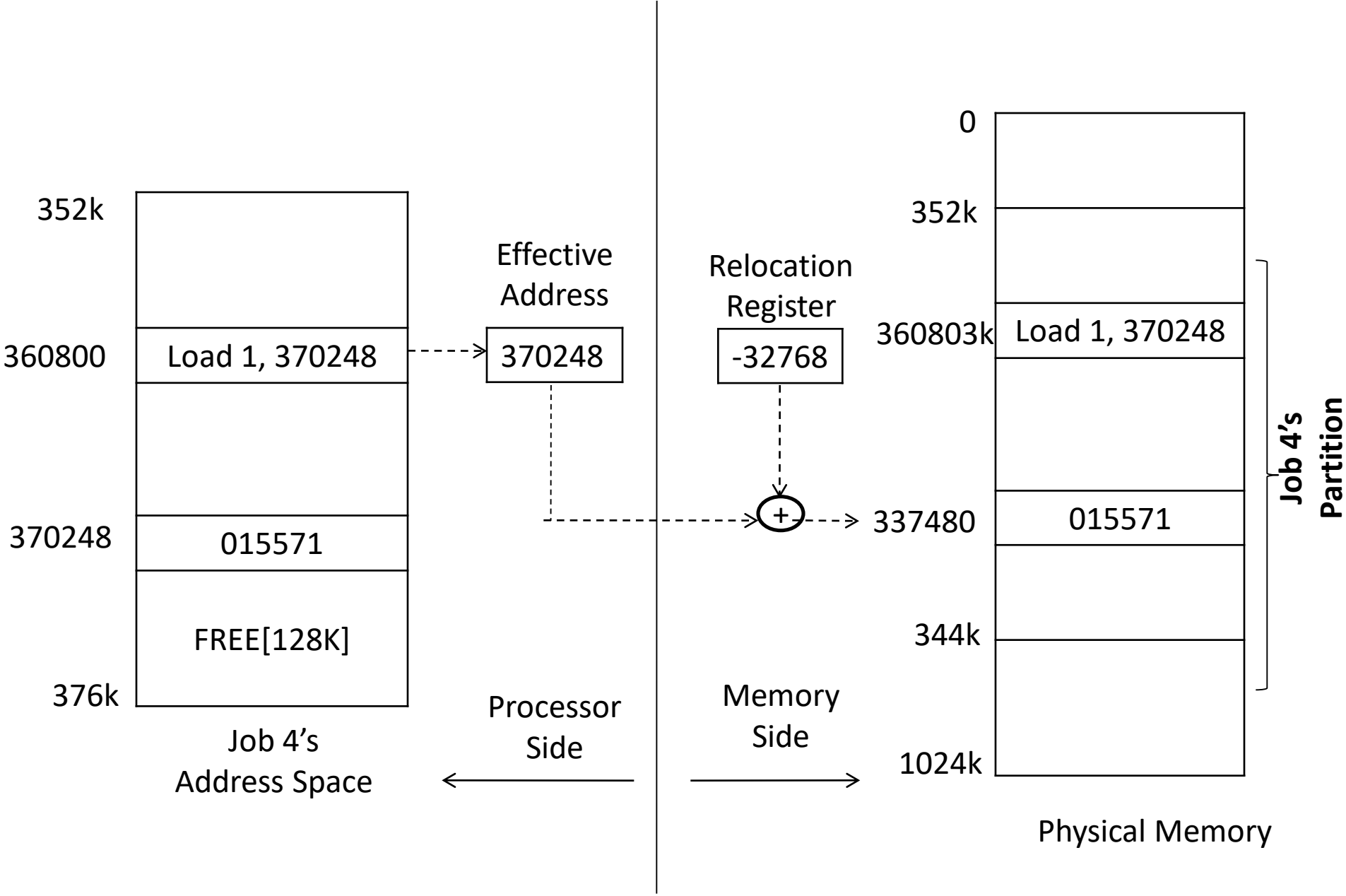
- The effective address is the final reference address computed by the processor.

Effective Address:

- It is the final reference address computed by the processor




352k
360800
Load 1, 370248
370248
015571
FREE[128K]
376k
Job 4's
Address Space
Effective
Address
370248
Relocation
Register
00000
+
Processor
Side
Memory
Side
0
352k
360800
Load 1, 370248
370248
015571
376k
1024k
Job 4's
Partition
Physical Memory
A. Before Reallocation



B. After Reallocation

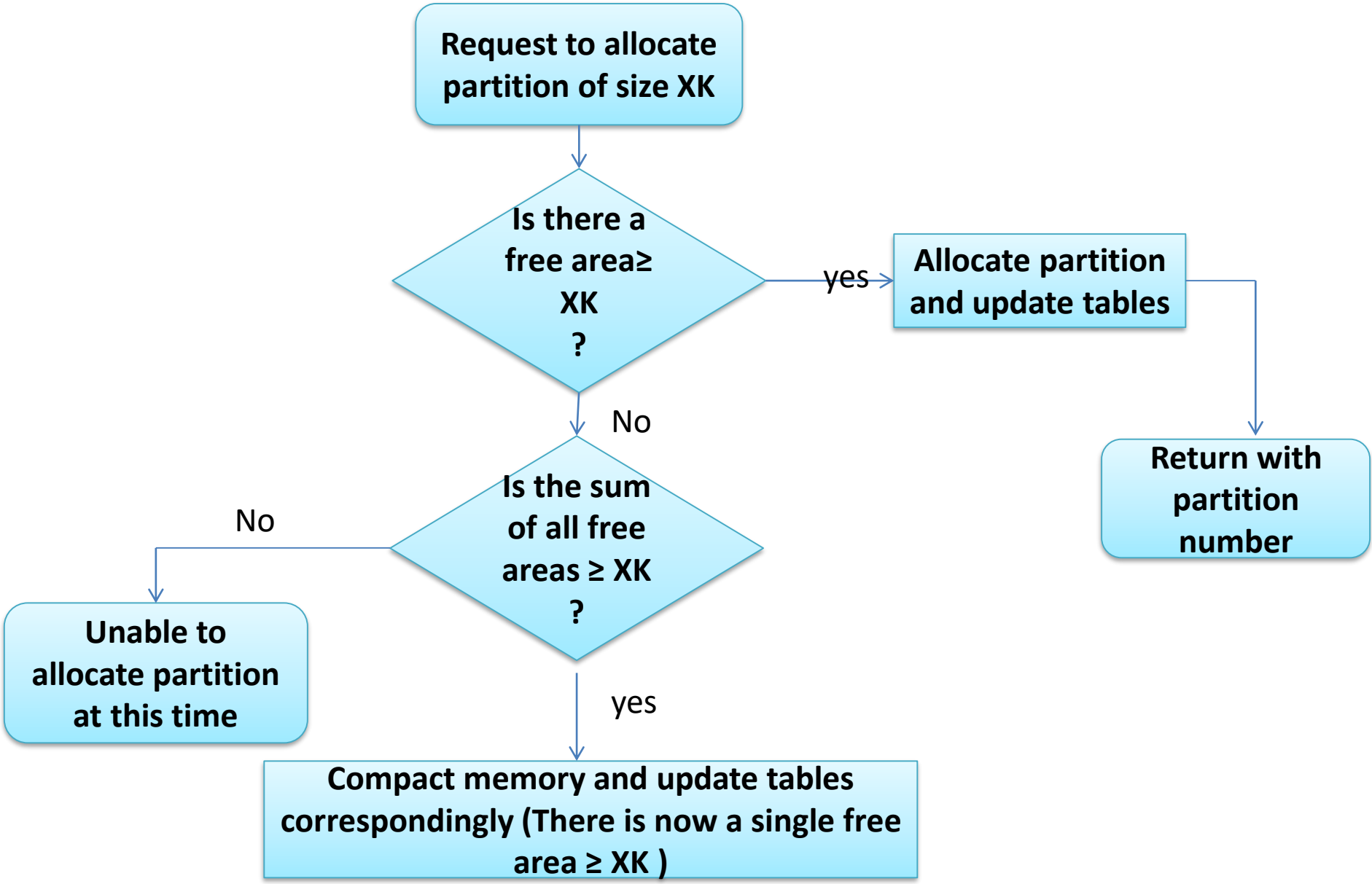
The instruction LOAD 1, 370248 at 360800 will load the value 015571 into register 1.

After relocation

- The LOAD instruction is at location 328032
- The effective address for the data to be loaded is still 370248 even though the data value has been moved to 337480.
- To produce correct result, the operating system must set the relocation register to -32768 (320K-352K=-32K)
- When the instruction LOAD 1, 370248 is encountered.
- -32768 is automatically added to the effective address, 370248, to determine the actual physical memory location to be accessed (370248-32768=337480).
- This relocation adjustment is done automatically as each instruction is executed, it is called “dynamic relocation”.

Software Algorithm:

Flowchart for relocation partition allocation



Advantages

- It eliminates fragmentation
- This allows multiprogramming
- Increased memory and processor utilization

Disadvantages

- Relocation hardware increase the cost of the computer and may slow down the speed
- Compaction time may be substantial
- Some memory will still be unused
- Memory may contain information that is never used. Job's partition size is limited to the size of physical memory

2.4 Paged Memory Management

- Each Job's address space is divided into equal pieces, called pages.

- Likewise, physical memory is divided into pieces of the same size called block.

- The hardware to perform the mapping from address space to physical memory there must be a separate register for each page. These register are often called Page Maps or Page Map Table (PMTs)

- Since each page can be separately located, there is no need for a job's partition to be completely contiguous in memory, only locations in a single page must be contiguous

Process	Memory
Job 1	2000 B
Job 2	3000 B
Job 3	1000 B
Job 4	2000 B

- Job2 has an address space of 3000 bytes is divided into 3 pages
- The Page Map Table associated with Job2 indicates the location of its pages

PMT(Page Map Table)

Page Number	Block Number
0	2
1	4
2	7

In this case:

- ✓ Page 0 is in block 2
- ✓ Page 1 is in block 4
- ✓ Page 2 is in block 7

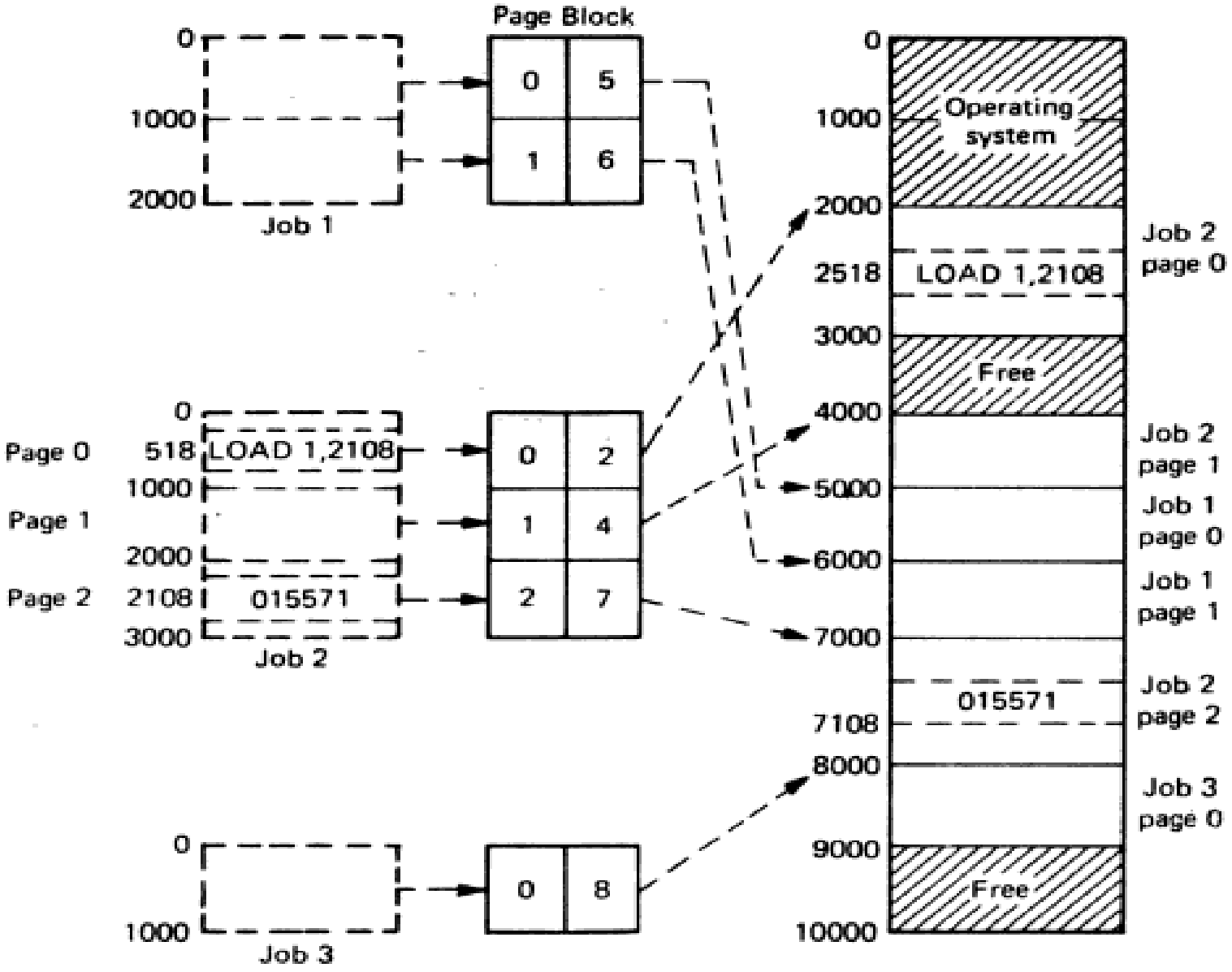


Figure 3-15 Page mapped memory

•The LOAD 1, 2108 instruction at location 0518 (page0, byte 518) in Job 2's address space is actually stored at physical location 2518 (block 2, byte 518).

•Likewise, the data 015571 logically located at address space -2108 is stored at physical memory location 7108.

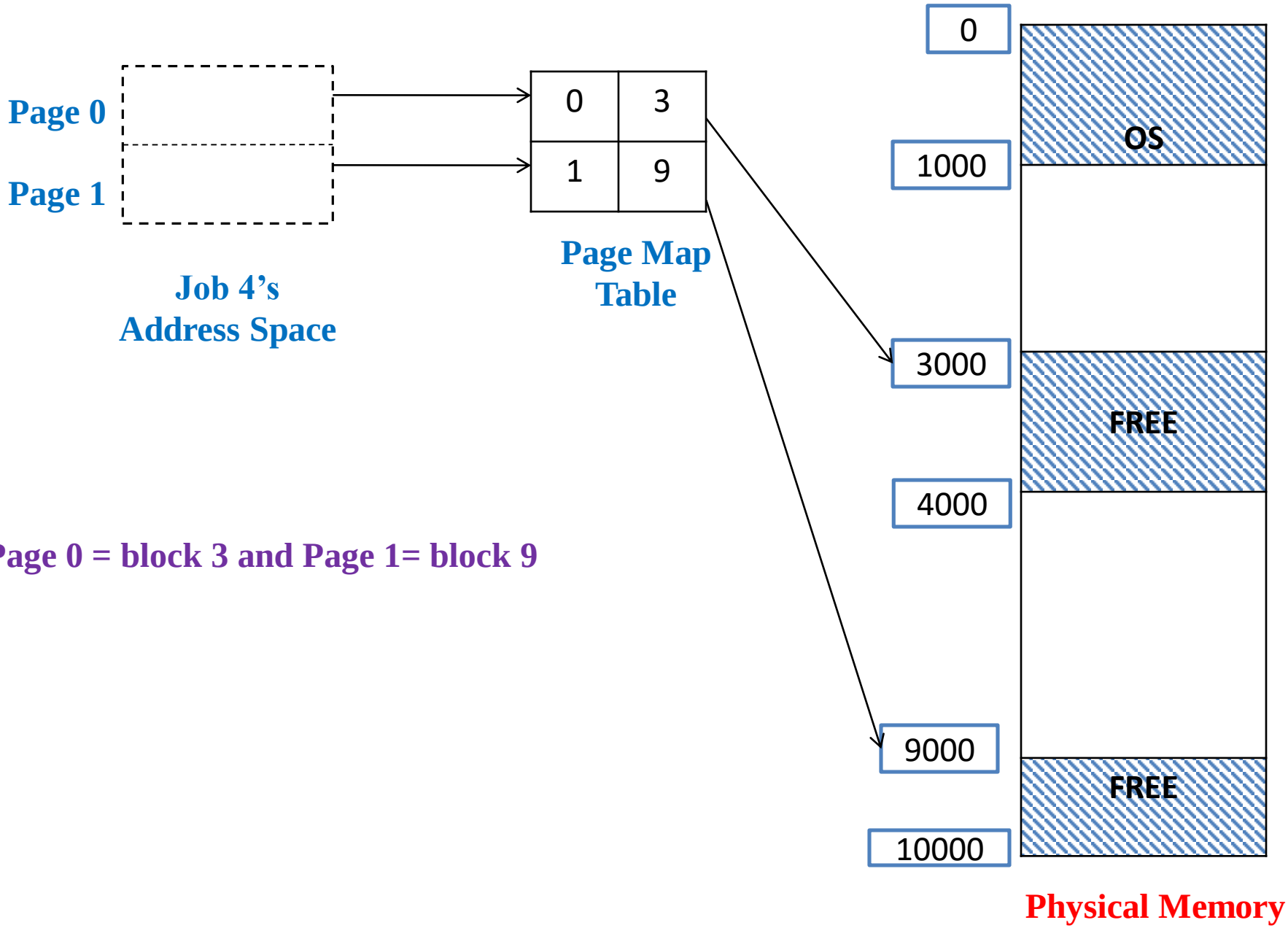
Advantages:

The paged memory management approach solves the fragmentation problem without physically moving partitions.

Example:

Remaining there are 2000 bytes of available memory, but they are not contiguous.

We can assign Job4’s into two pages and assign these pages to the available block. Such as



Page 0 = block 3 and Page 1= block 9

Four functions of Paged Memory Management:

- Keeping track of status – accomplished through two sets of tables:
    a. Page Map Tables – One entry for each page
    a. Memory Block Tables – One entry for each memory block (eg: allocated or available)
- Determining who gets memory- this is largely decided by the Job scheduler
- Allocation all pages of the job must be loaded into assigned blocks.
- Deal location-when the job is done, blocks must be returned to free

Hardware Support:

A hardware mechanism is needed to perform the mapping from each instructions effective address to the appropriate physical memory location

1. High – Speed Page Map Registers:

This scheme can be made attractive in several ways:

- First, if the Job's address space is limited such as l00 k bytes, no more than 25 registers will needed for each Job's address space.
- Second since only one job is running at a time only one set of hardware page mapping register will be needed.

Disadvantage:

- Continuous resetting when multiprogramming.
- Cost is very high

2. Page Map Table

•Page Map Tables used to eliminate continuous resetting.

•To simplify address mapping the page is usually chosen to be a power of two (eg:1024(1k) bytes, 2048(2kb) or 4096(4kb)

•The DAT(Dynamic Address Transaction) mechanism automatically separates the effective address into two parts

- Bits 8 through 19 become the 12 bit page number
- Bits 20 through 31 become the 12 bit byte number.

•Use PMT the page number is replaced by the block number to produce the resultant physical memory address to be used.

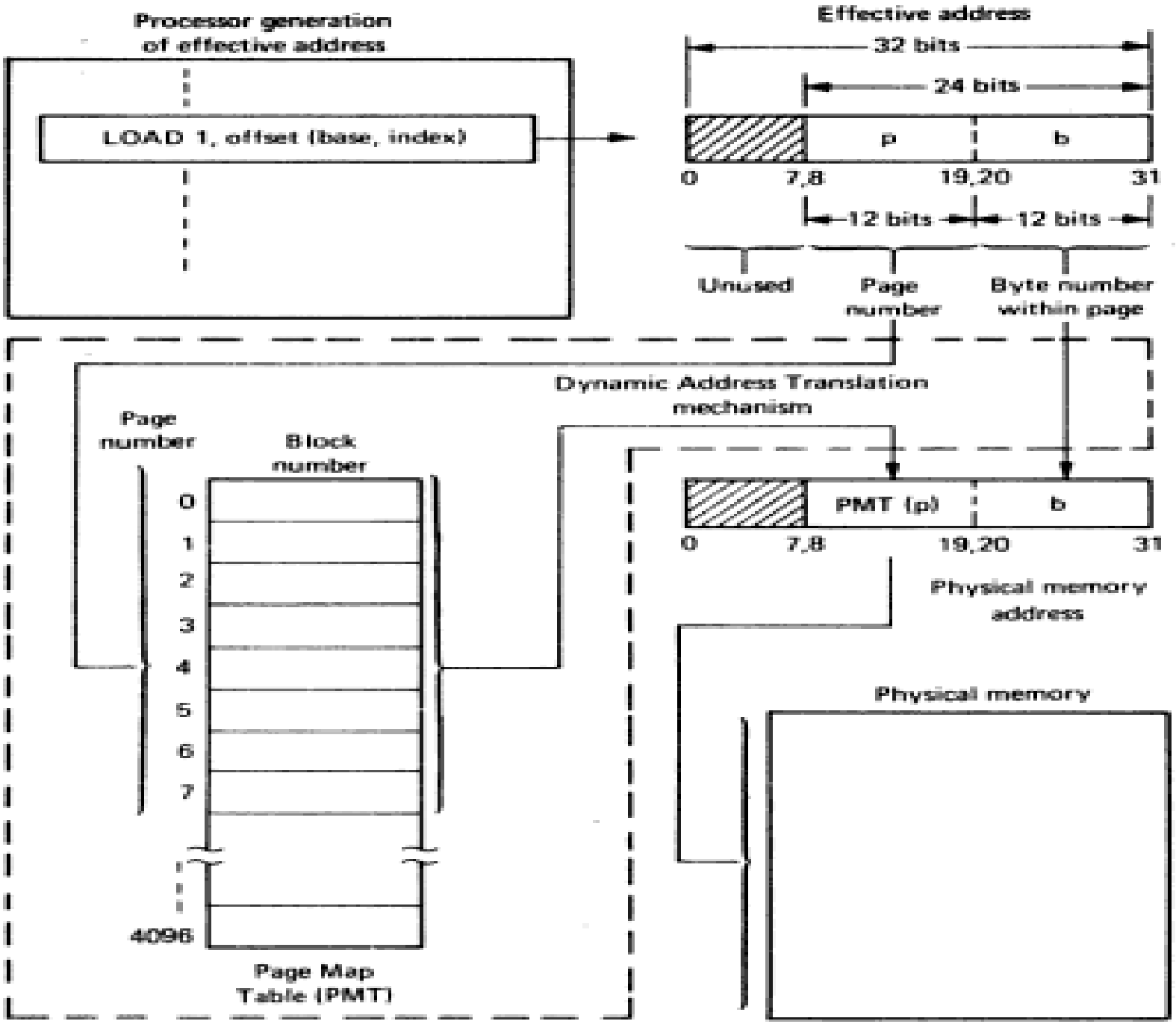


Figure 3-16 Determination of physical memory address from effective address

•When the processor is switched to new job, only the PMTAR (Page Map Table Address Register) has to be changed to indicate the location of the new Job PMT.

•The computer would run at about half its normal speed.

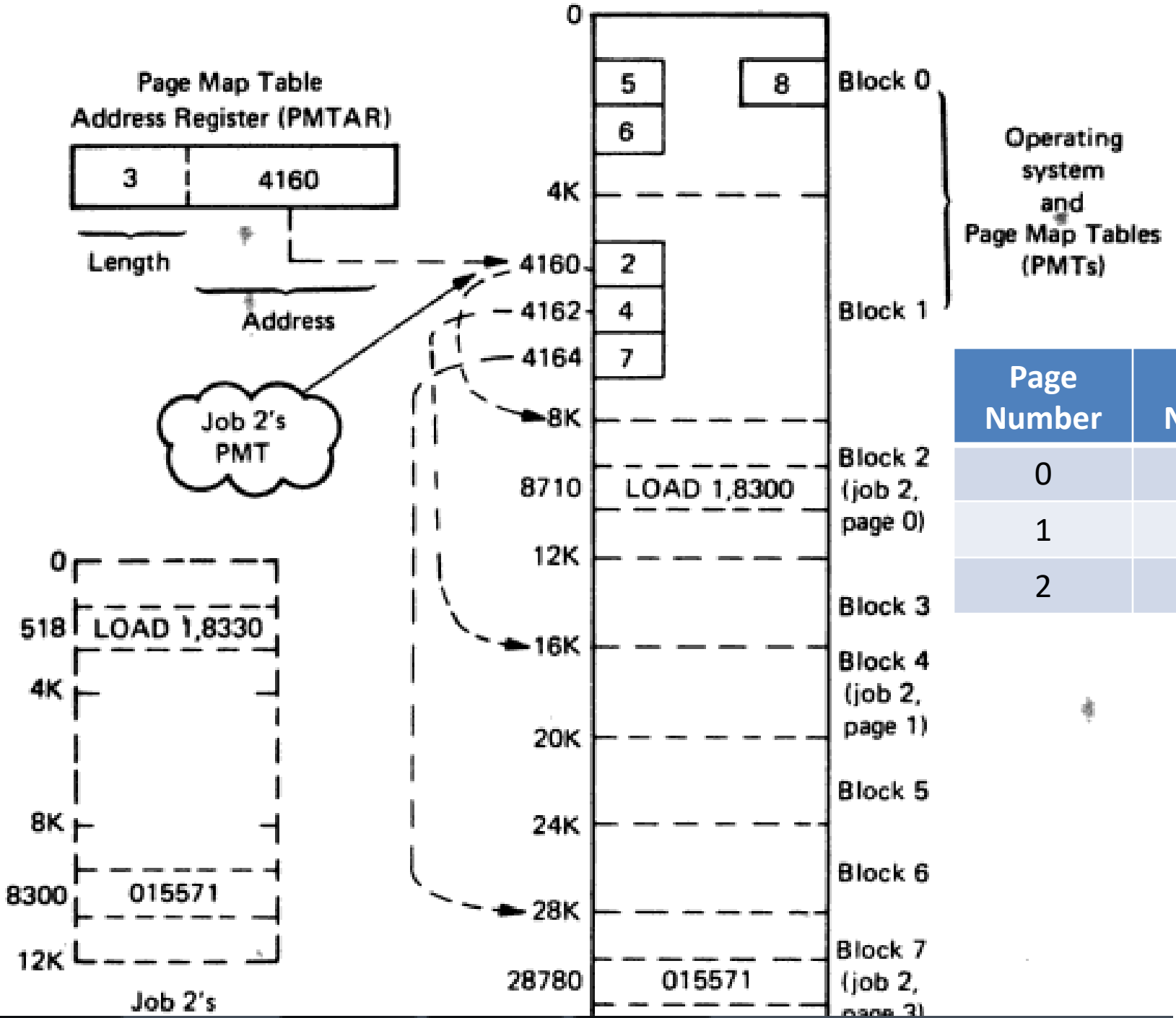
3. Hybrid Page Map Table:

- A hybrid scheme, combining aspects of the high-speed mapping registers and the PMTs is often used to overcome this speed problem.

- A small number of special high-speed registers are used to hold portions of the PMTs.

- Whenever possible , these registers are used to dispose with accessing the PMT in memory.

-These special registers are often called an associative memory or a table look-aside buffer.



Page Number	Block Number
0	2
1	4
2	7

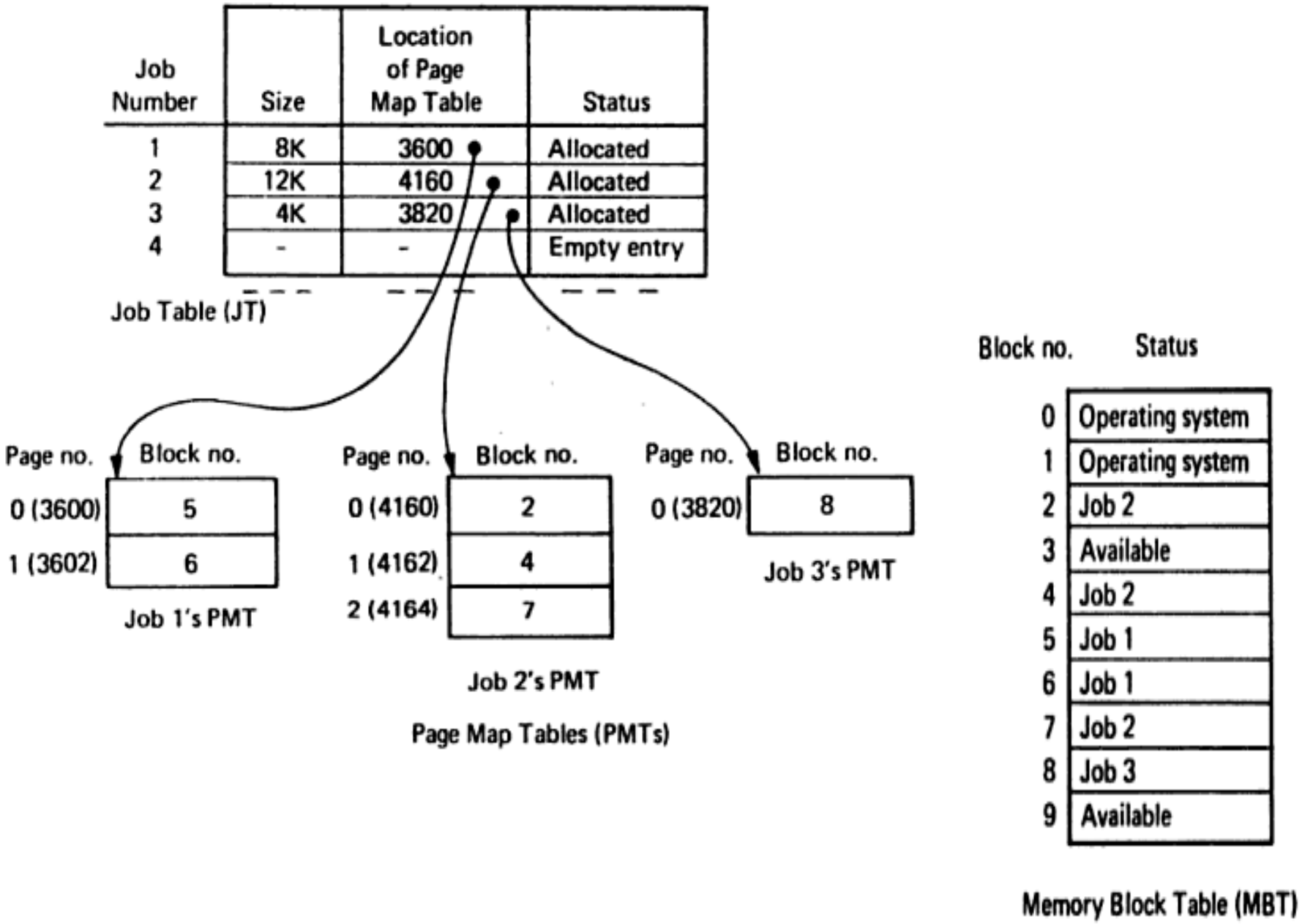


Figure 3-18 Tables used by paged memory allocation software

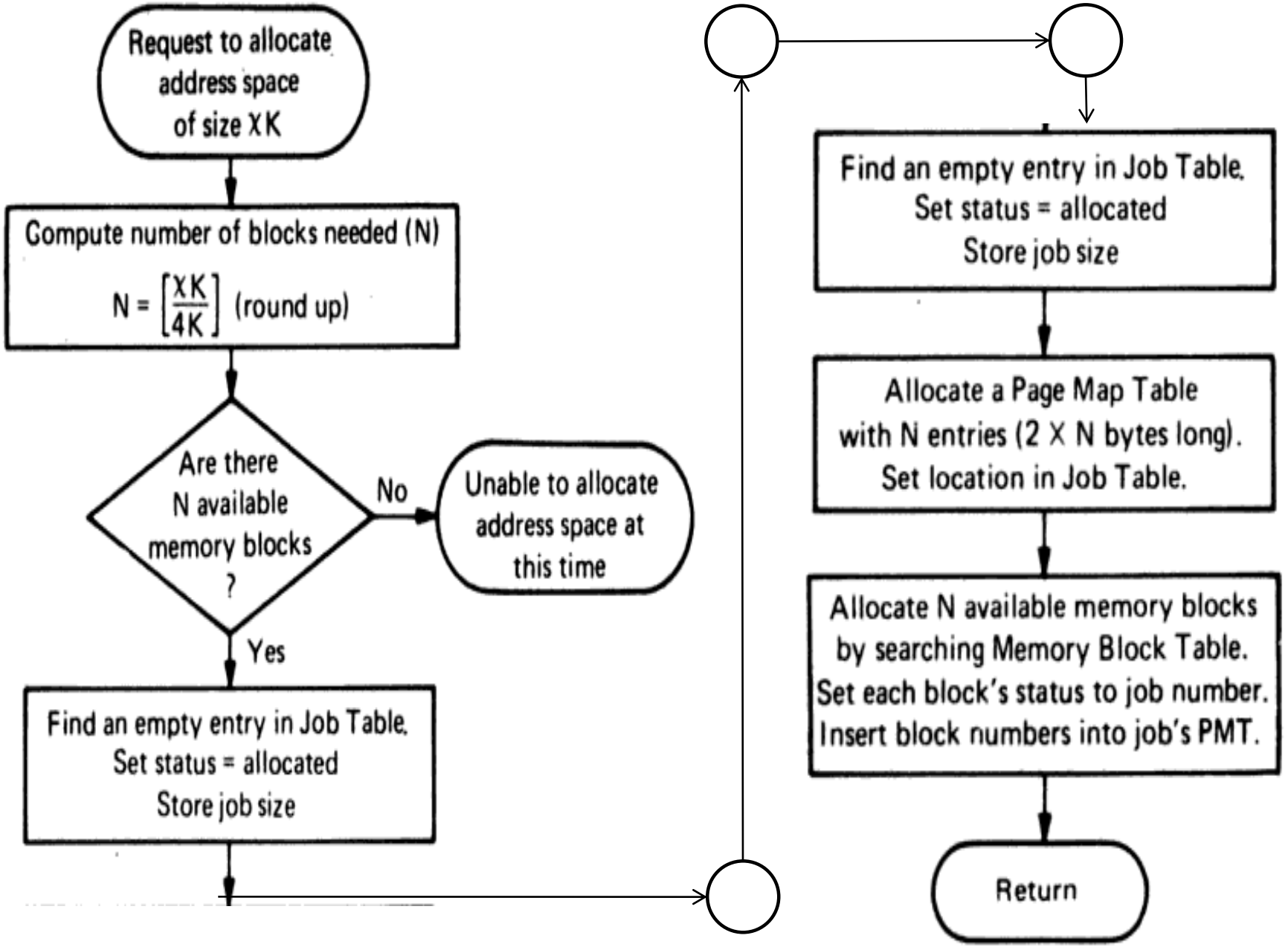


Figure 3-19 Paged memory allocation algorithm

Advantages:

(1) It eliminates Fragmentation

(2) It allows multiprogramming

(3) The relocation partition scheme is also eliminated

Disadvantages:

(1) Mapping h/w increase the cost

(2) using various tables PMT,MBT, etc…

(3) Internal fragmentation occur. Half a page is wasted for each job.

(4) some memory will still be unused.

5. Demand Paged Memory Management:

- In all the previous schemes a job could not be run until there was sufficient available memory.

- These problems could be solved by using extremely large main memory.

- The OS to produce the illusion of an extremely large memory. It is called Virtual Memory

There are Two Virtual Memory Techniques:

1. Demand Paged Memory Management
2. Segmented Memory Management

Example:

- consider the following figure, three jobs are loaded in memory
- fourth job requires 4000 bytes (i.e.) it needs four block, but only two blocks are available
- here , we can load only two pages of job4

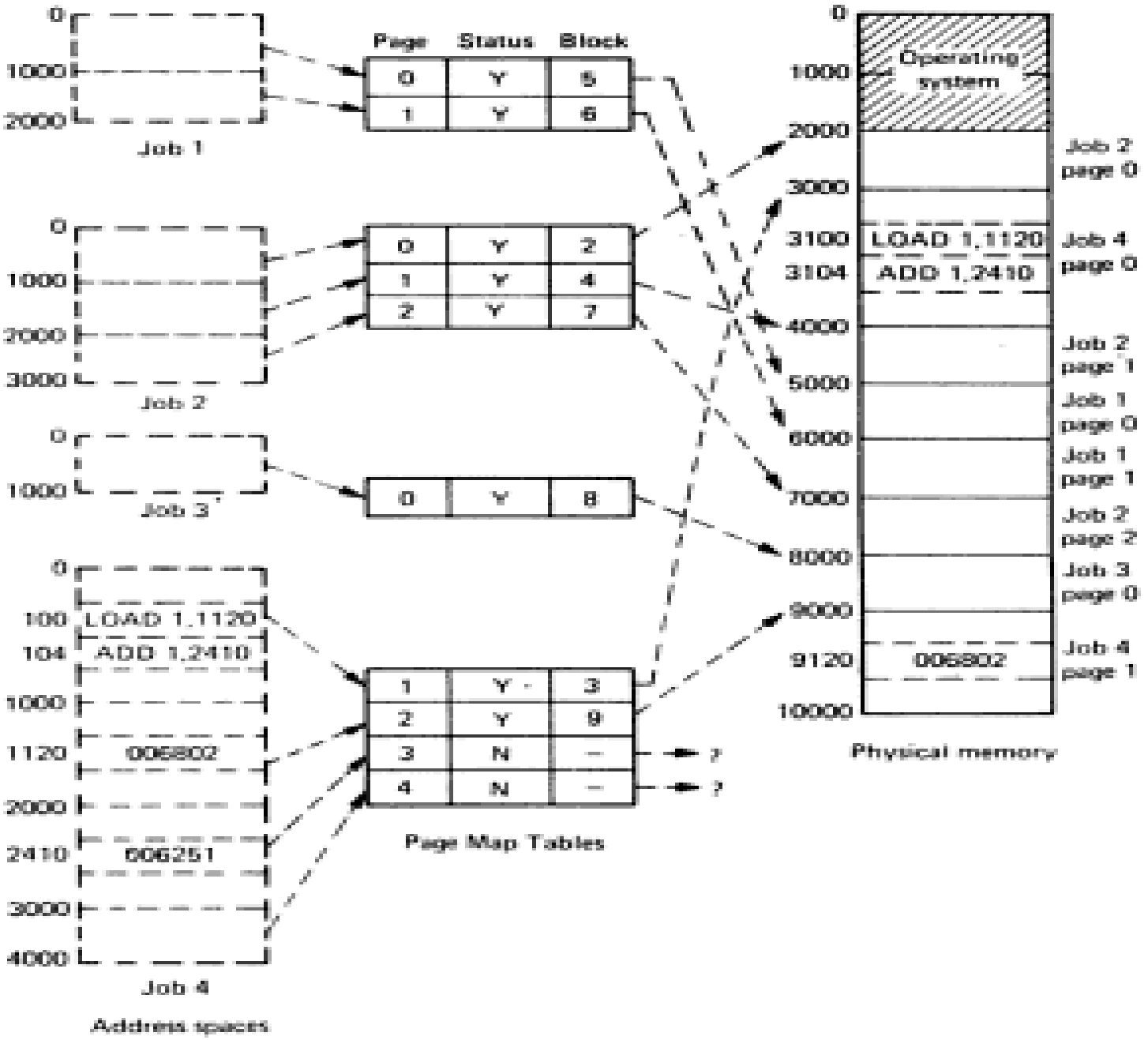


Figure 3-21 Demand-paged-mapped memory

Two Key questions remain to be answered:

(1) What do we do if a job references an area of space not in physical memory.

(2) How do we decide which pages to keep in memory?

Note in above Fig:

- The ADD 1, 2410 instruction at location 104 in Job 4 references a page that is not in memory

- To handle this case, the PMT h/w to include a status

Y= yes , reference is ok

N= no, reference is impossible

- if the address mapping h/w encounters a page table entry with status=N, it generates a Page Interrupt.

- The OS must process this interrupt by loading the required page and adjusting the page table entries correspondingly.

- We say this page was loaded on demand

- This scheme is called "Demand Paged Memory Management".

- when a job is initially scheduled for execution, only its first page is actually loaded. All other pages needed by the job are subsequently loaded on demand.

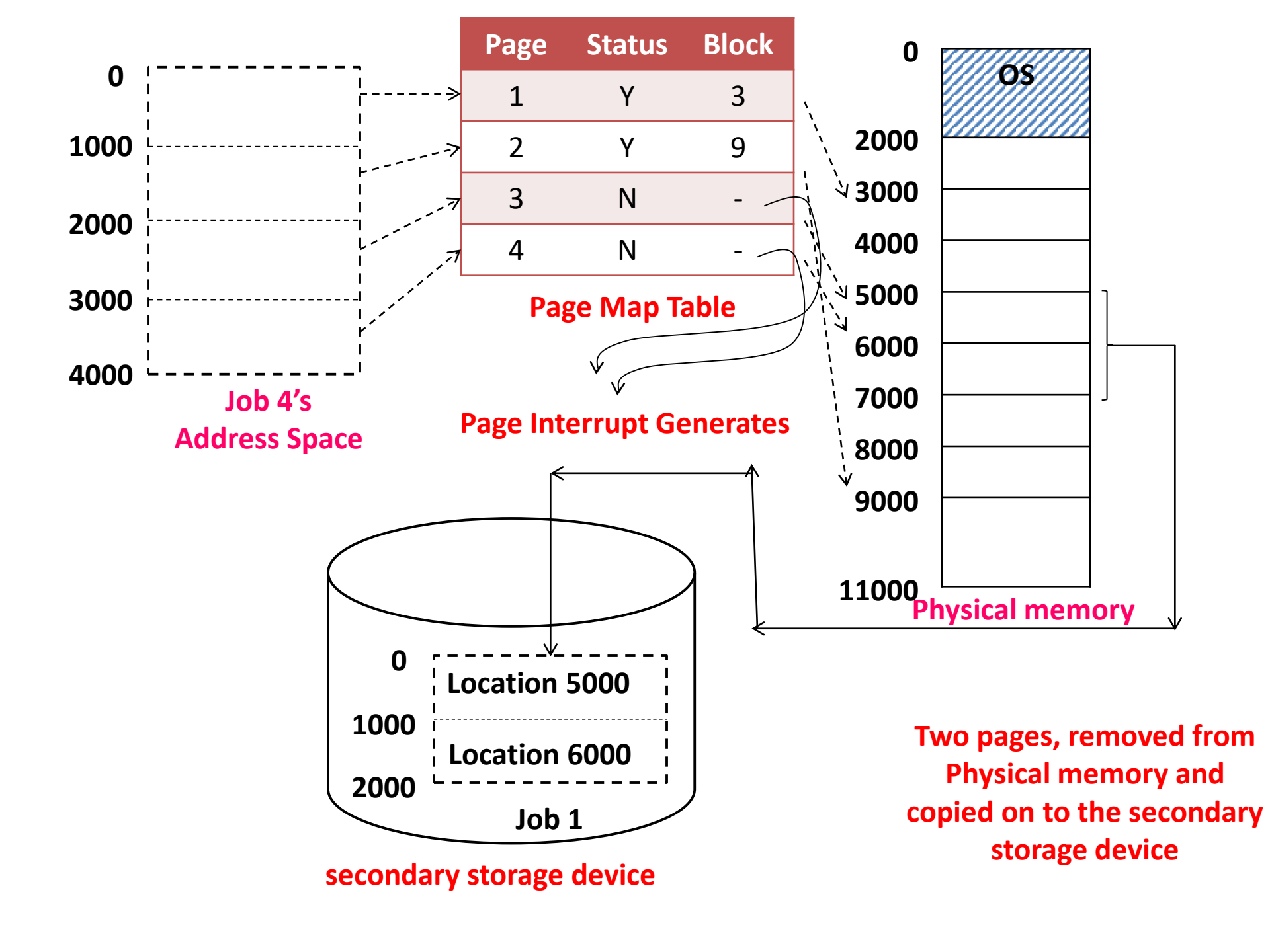
- This guarantees that an unnecessary page is not loaded.

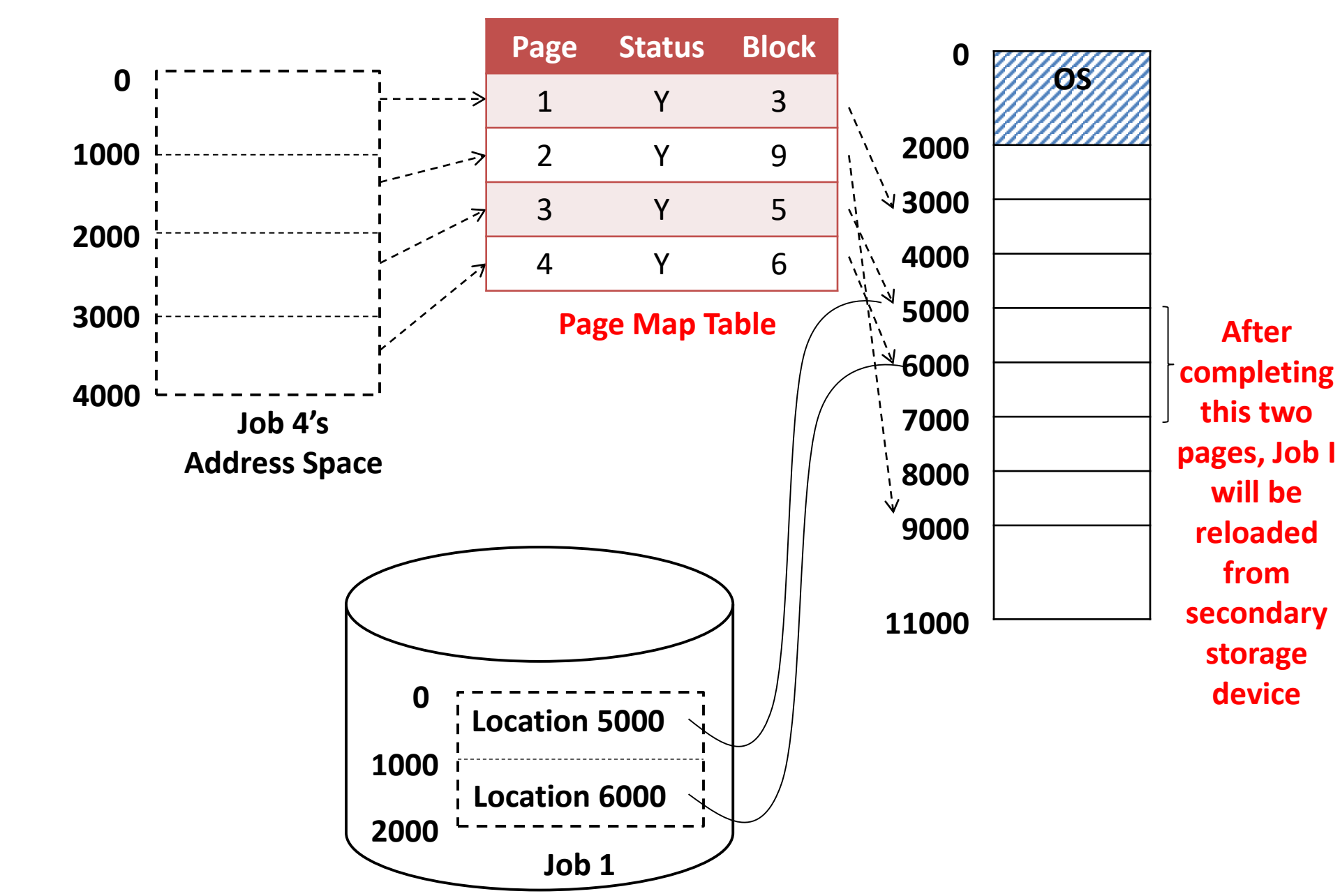
Thrashing:

- only memory has becomes filled with pages it is possible to load another page only by first removing one of the pages presently in memory.

- The replaced page is copied back onto the secondary storage device before the new page is loaded. i.e. the two pages swap places between main memory and secondary storage.

-Moving pages back and forth between main memory and second memory has been called "Thrashing".


Page
Status
Block
1
Y
3
2
Y
9
3
N
-
4
N
-
Page Map Table
0
1000
2000
3000
4000
Job 4's
Address Space
Page Interrupt Generates
0
OS
2000
3000
4000
5000
6000
7000
8000
9000
11000
Physical memory
0
Location 5000
1000
Location 6000
2000
Job 1
secondary storage device
Two pages, removed from Physical memory and copied on to the secondary storage device



The OS software is making these page replacement decisions.

Four function of Demand-Paged Memory Management:

1. Keeping track of status – this is accomplished through three sets of tables:
    - Page Map Tables
    - Memory Block Tables
    - File Map Tables

2. The policy of who gets memory and when, how long.

3. Allocation of Block

4. Deallocation of a Block, when a job terminates

Hardware Support

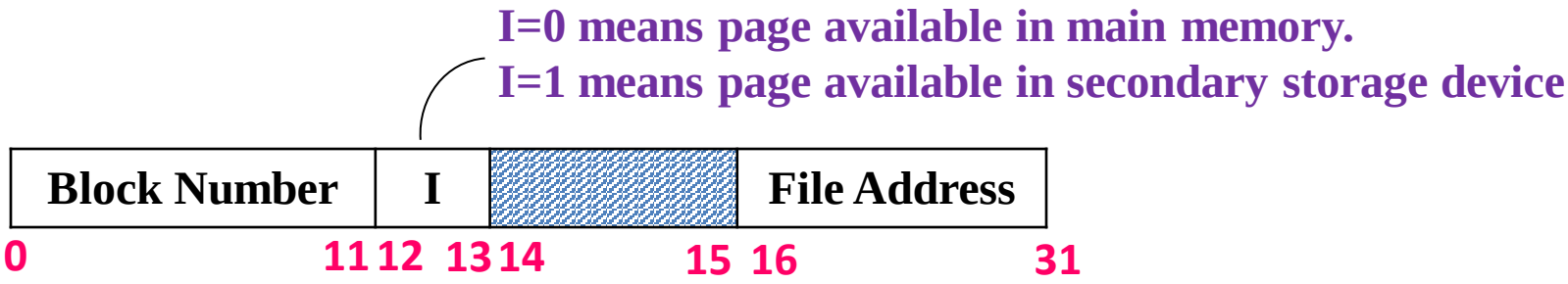
The address mapping hardware Via Page Map Table needed for demand paging.

1. A status bit in the PMT to indicate whether the page is in main memory or secondary storage.
   - Status =Y→ Physical memory
   - Status =N→ Secondary memory
2. Interrupt action to transfer control to the OS. If the job attempts to access a page not in main memory.
3. Record of individual page usage to assist the os in determining which page to remove, if necessary.

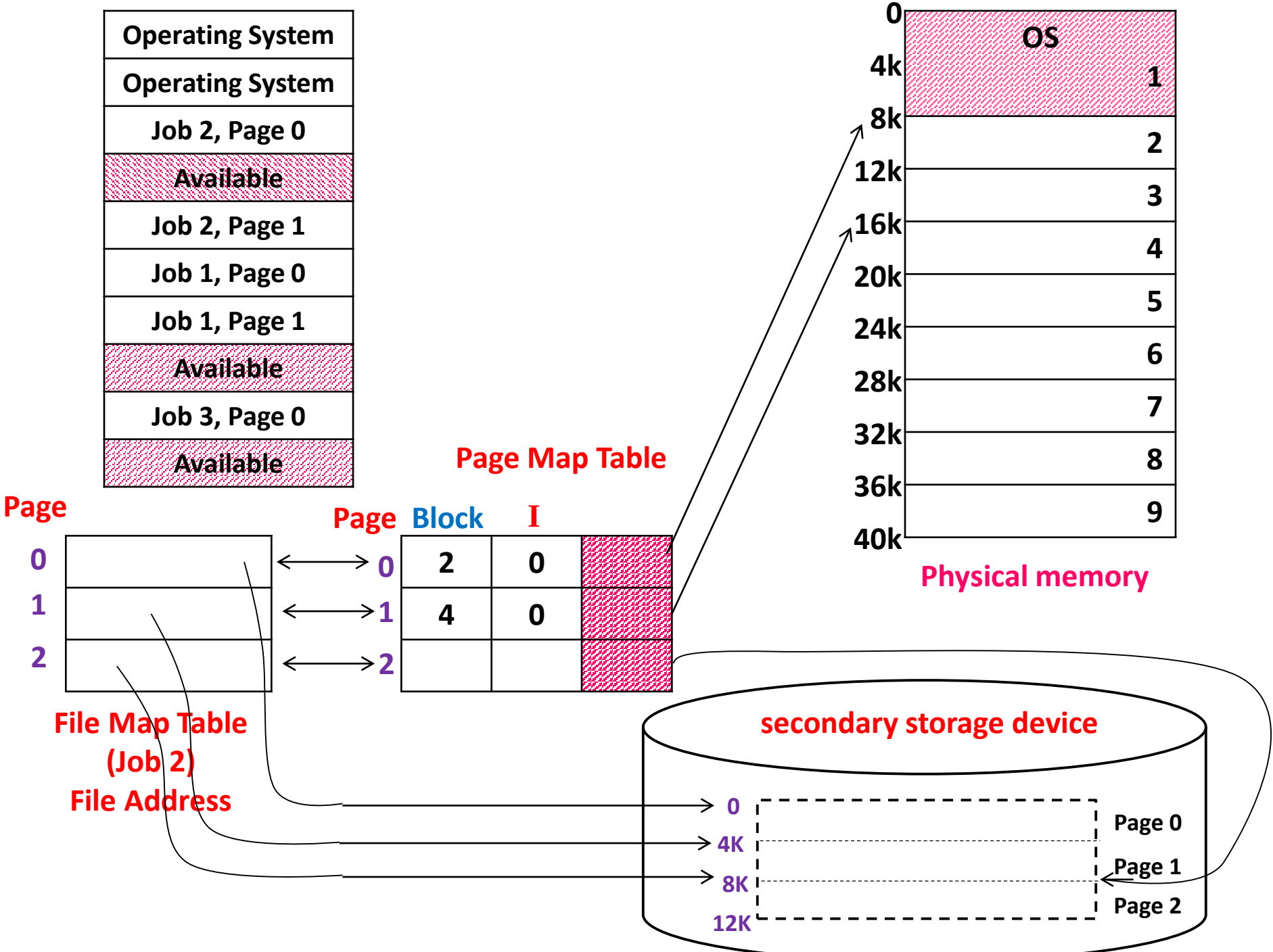
Software Algorithm

1. File Map Requirement:

- Demand-Paged memory management must interact with information management to access and store copies of the Job's address spaces on secondary storage.
- Each page has a unique secondary storage address called its file address. This address logically is part of each PMT entry



- The file information is usually stored in a separate table called the File Map Table.
- The relationship of the File Map Table to the Page Map Table and the Memory Block Table is illustrated in the following diagram.
- In this diagram Job 2 has only two of its three pages in main memory
- A copy of all pages is available on a secondary storage

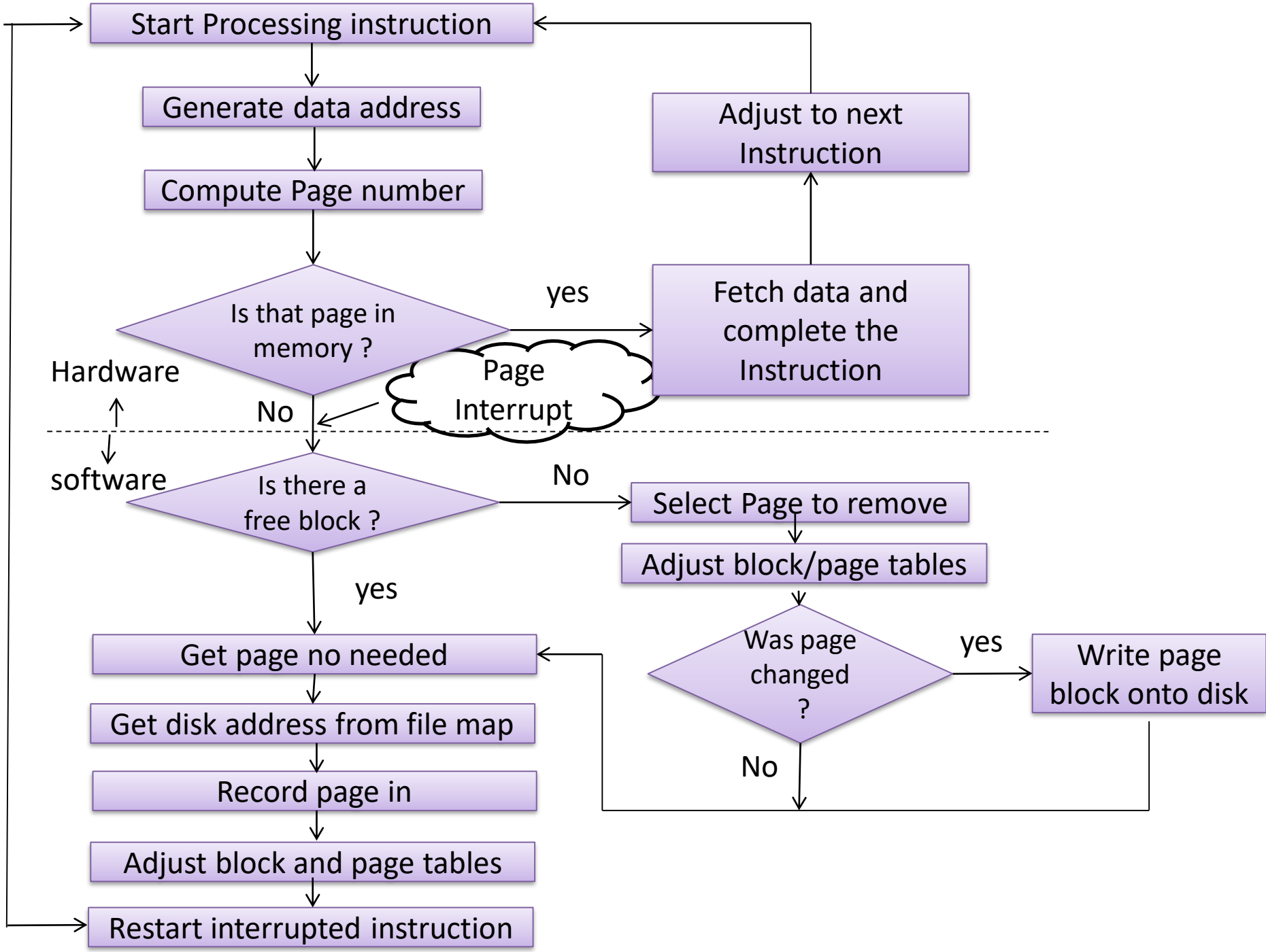

Operating System
Operating System
Job 2, Page 0
Available
Job 2, Page 1
Job 1, Page 0
Job 1, Page 1
Available
Job 3, Page 0
Available
0
4k
8k
12k
16k
20k
24k
28k
32k
36k
40k
OS
1
2
3
4
5
6
7
8
9
Physical memory
Page Map Table
Page
Page
Block
I
0
1
2
0
1
2
2
0
4
0
File Map Table
(Job 2)
File Address
secondary storage device
0
4K
8K
12K
Page 0
Page 1
Page 2

2. Overview of Page Interrupt Processing

- In previous memory management schemes, allocation of memory was completely static. That is memory was allocated or rearranged only when a job started or terminated.
- In demand paging, it is necessary to allocate and deallocate memory during the job's execution. The memory management routines are involved by means of the page interrupt.

3. Interaction between hardware and software

- In demand paging there is very close interaction between the hardware and software, which is illustrated in the following flowchart.
- The first part of the flowchart is implemented as part of the address mapping hardware
- The second part is implemented as an interrupt handler routine within the operating system.


Start Processing instruction
Generate data address
Compute Page number
Is that page in memory ?
yes
Fetch data and complete the Instruction
Adjust to next Instruction
Hardware
Page Interrupt
No
software
Is there a free block ?
No
Select Page to remove
Adjust block/page tables
yes
Was page changed ?
yes
Write page block onto disk
No
Get page no needed
Get disk address from file map
Record page in
Adjust block and page tables
Restart interrupted instruction

Page Removal Algorithm

There are 3 schemes for replacement algorithm

1. FIFO(First In First Out)
    - The oldest Job will be removed
2. LRU(Least Recently Used)
    - Removes the page that has been in memory for the largest time.
3. Tuple – Coupling
    - The smallest pages will be removed

Advantages

1. Fragmentation is eliminated
2. Compaction is not needed
3. Large virtual memory
4. More efficient use of memory
5. Unconstrained multiprogramming

Disadvantages

1. The number of tables and amount of processor overhead for handling page interrupt
2. Under extreme thrashing situations over 99% of processor time may be consumed by overhead activities and less than 1% by user jobs

3.Processor Management

-It is concerned with the **management of the physical processor**

-The assignment of **processors to processes**

3 Important topics in this chapter:

i) **Job scheduling** (Macro scheduling):

-It is used to choosing **which jobs will run**.

-It is concerned with the **management of jobs**.

ii) **Process scheduling** (Micro scheduler):

-It is used to **assigning processors to the processes** associated with scheduled jobs.

-It is concerned with the **management of processes**

iii) Traffic Controller:

-Keep track of status of all processes

-Providing the mechanism for changing process – states

-co-ordinate inter process synchronization & communication

3.1 State Model

Totally we have 6 states, there are:

i) Submit state

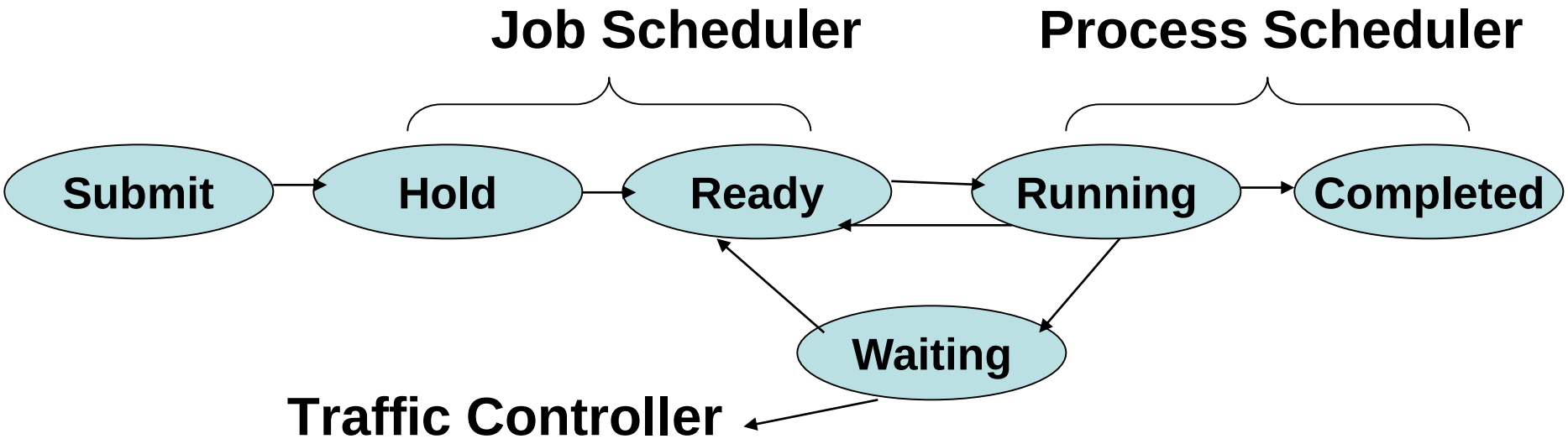
ii) Hold state

iii) Ready state

iv) Running state

v) Blocked state/ Waiting state

vi) Completed state



i) Submit state:

- Program or instruction code submitted into the computer

ii) Hold State:

- Job scheduler could examine whether all processors available to process some processes

iii) Ready state:

- Job Scheduler set up a processor to each processes

iv) Running state:

-Process was finally assigned a processor to execute the program or instruction code

v) Blocked state / Waiting state:

- process scheduler gave particular time quantum to each processor

- if it is finished, process initiated another I/O process

- The processor had to wait until I/O was completed. It is namely called as blocked state

- Traffic Controller is used to change from the blocked state to ready state.

vi) Completed state:

- Process Scheduler assigned a processor to a process and if it finished, it will come to completed state.

Here we have 3 important topics:

i) Job scheduler
ii) process scheduler
iii) job and process synchronization

i) Job scheduler:

1) Keep track of status of all jobs(whether it is hold, ready, running or blocked state)

2) choose the policy by which job will " enter the system"

3) Allocate the necessary resources

4) Reallocate these resources when the job is done.

ii) Process scheduling:

1) Keep track of status of the process
2) Deciding which process gets a processor and for how long
3) Allocate of a processor to a process
4) Reallocate these processor to a process, when it terminates

iii) Job & Process synchronization:

Intermixed:

-One process requests a printer while another process is printing, if the printer were to be also assigned to the second process, it will produce intermixed between the two processes

Deadlock:

-There are two processes, each of which is waiting for resources that other will not give up.

The P & V operator and semaphores:

-They are one set of mechanisms for coordinating the assignment of processors to processes

3.2 JOB SCHEDULING

-It is uses scheduling time, priority, memory needs, device needs, processor needs, and synchronization

Functions:

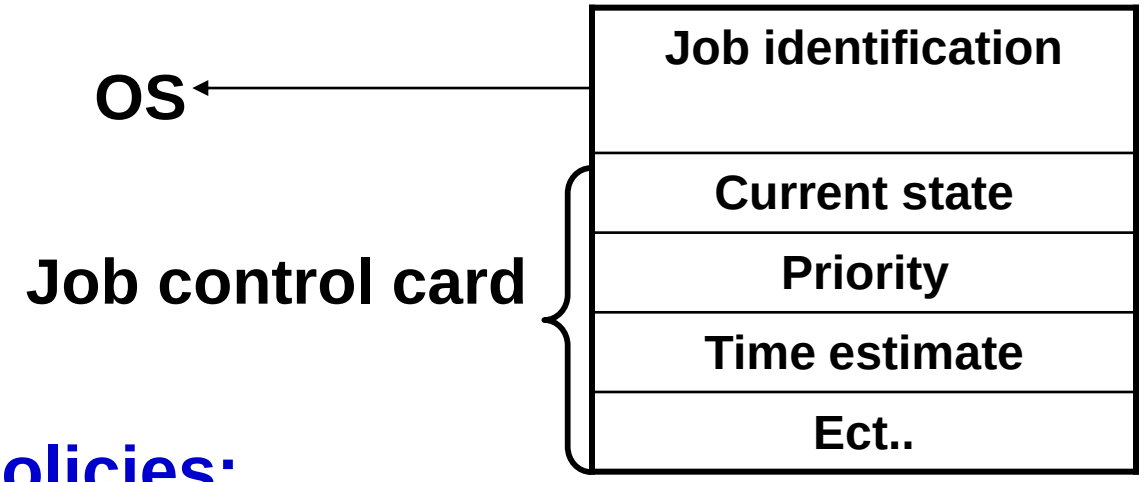
1) Keep track of the jobs
2) invoke policies for deciding which jobs get resources
3) Allocate the resources
4) De-allocate the resources

Job Control Block:

-Each job have its own Job Control Block

-Priority and time estimation is obtained from the Job control card submitted with the job

-Current state is set by the operating system



Policies:

-The job scheduler must choose from among the “hold” jobs.

Typical considerations one must deal with in determining a job scheduling policy are:

1) Availability of special limited resources

2) Cost-higher rates for faster service

3) System commitments – processor time and memory - the more you want, the longer you wait

4) System balancing – mixing I/O-intensive and CPU-intensive

5) Guaranteed service - setting a specific waiting time limit(1hr) or general wait limit (within 24 hr) (at least don't lose the job)

6) Completing the job by a specific time

a) Job scheduling in Non-multiprogramming Environment:

-Once a processor has been assigned a process, it does not release the processor until it is finished.

i) Job scheduling using FIFO:

Sample Table:

Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00	10.00	12.00	2.00
2	1.00 hrs				
3	0.25 min				

a) Job scheduling in Non-multiprogramming Environment:

-Once a processor has been assigned a process, it does not release the processor until it is finished.

i) Job scheduling using FIFO:

Sample Table:

Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00	10.00	12.00	2.00
2	1.00 hrs	10.10	12.00	13.00	2.90
3	0.25 min	10.25	13.00	13.25	3.00
					7.90

-If we use a **FIFO** algorithm, the jobs will be run as depicted in FIFO manner

-**Average Turn around time**

$T = \left(\sum_{i}^{n} Ti \right) X \; 1/n$

where Ti=Fi-Ai

Fi – Finish time

Ai – Arrival time

$T = 7.90/3 =$ **2.63 hrs**

ii) Job scheduling using Shortest Job First:

Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00	10.00	12.00	
2	1.00 hrs	10.10	12.25	13.25	
3	0.25 min	10.25	12.00	12.25	

ii) Job scheduling using Shortest Job First:

Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00	10.00	12.00	2.00
2	1.00 hrs	10.10	12.25	13.25	3.15
3	0.25 min	10.25	12.00	12.25	2.00
					7.15 hrs

-Average Turn around time T=7.15 / 3 = 2.38 Hrs

Job1

Job3

Job2

- When Job1 arrives, it is run
- While Job1 running Job2 and Job3 arrive
- We choose to run Job3 next because it has a shorter run time than Job2
- Algorithm did reduce the average turn around time

- Also we can reduce average turn arround time, that is to run Job3 first, next to run Job1 and then Job2

Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00			
2	1.00 hrs	10.10			
3	0.25 min	10.25			

Job3
Job2
Job1

Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00			
2	1.00 hrs	10.10			
3	0.25 min	10.25	10.25	10.50	2.00

Job3
Job1
Job2

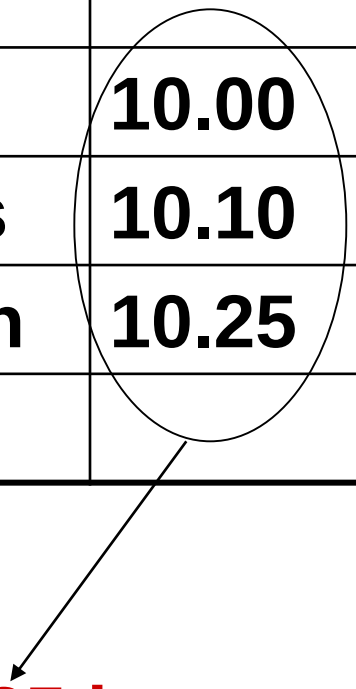
Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00			
2	1.00 hrs	10.10	10.50	11.50	3.15
3	0.25 min	10.25	10.25	10.50	2.00

Job3
Job1
Job2

Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00	11.50	13.50	2.00
2	1.00 hrs	10.10	10.50	11.50	3.15
3	0.25 min	10.25	10.25	10.50	2.00
					7.15 hrs

Job3
Job1
Job2

Job No	Processing Time	Arrival Time (Ai)	Start time	Finish Time (Fi)	Turn arround Time (Ti)=Fi-Ai
1	2.00 hr	10.00	11.50	13.50	3.50
2	1.00 hrs	10.10	10.50	11.50	1.40
3	0.25 min	10.25	10.25	10.50	0.25
					5.15 hrs



CPU idle =0.25 hrs

-Average Turn around time T=5.15 / 3 =1.72 hrs

-We did reduce average turn around time, but we wasted 0.25 hrs of CPU time.

iii) Measure of scheduling performance:

i) Average Turn around time T= Ti / n

ii) Weighted turn around time W = T /R

T – Turn around Time

R – Actual run Time

b) Job scheduling in Multiprogrammed Environment:

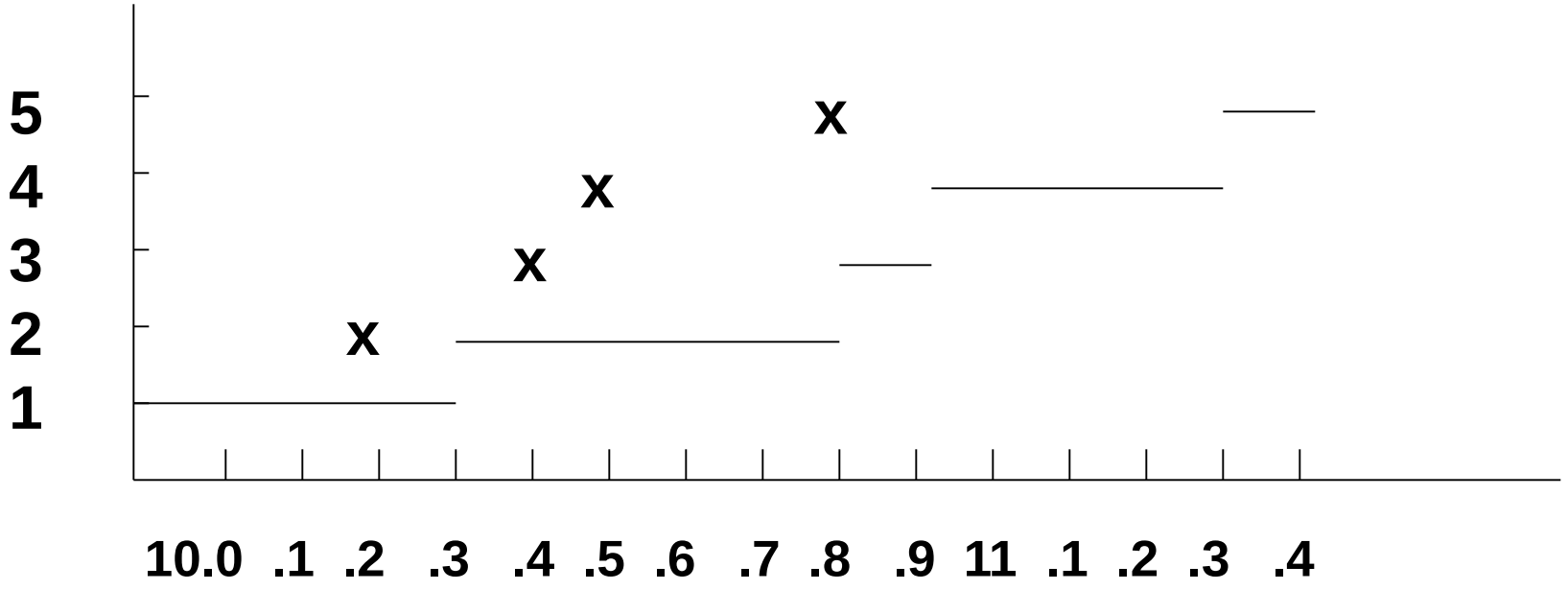
-Round robin algorithm is used to assigned a processor for some small time quantum

-That is, if n jobs are running simultaneously, they each get an equal share of run time.

i) Job scheduling with multiprogramming But no I/O Overlap:

Job No	Arrival Time	Run Time (R)
1	10.0	0.3 hr
2	10.2	0.5 hr
3	10.4	0.1 hr
4	10.5	0.4 hr
5	10.8	0.1 hr

Graphical Representation:



FIFO – no multiprogramming:

Job no	Arrival Time (Ai)	Start Time	Finish Time (Fi)	Turn around time Ti=Fi-Ai	Weighted Turn around time W= T / R T—Turn Around Time R –Actual time
1	10.0	10.0	10.3	0.3	(0.3 /0.3)
2	10.2	10.3	10.8	0.6	(0.6 / 0.5)
3	10.4	10.8	10.9	0.5	
4	10.5	10.9	11.3	0.8	
5	10.8	11.3	11.4	0.6	

FIFO – no multiprogramming:

Job no	Arrival Time (Ai)	Start Time	Finish Time (Fi)	Turn around time Ti=Fi-Ai	Weighted Turn around time W= T / R
1	10.0	10.0	10.3	0.3	0.3 / 0.3
2	10.2	10.3	10.8	0.6	0.6 / 0.5
3	10.4	10.8	10.9	0.5	
4	10.5	10.9	11.3	0.8	
5	10.8	11.3	11.4	0.6	

FIFO – no multiprogramming:

Job no	Arrival Time (Ai)	Start Time	Finish Time (Fi)	Turn around time Ti=Fi-Ai	Weighted Turn around time W= T / R
1	10.0	10.0	10.3	0.3	1.00
2	10.2	10.3	10.8	0.6	1.20
3	10.4	10.8	10.9	0.5	5.00
4	10.5	10.9	11.3	0.8	2.00
5	10.8	11.3	11.4	0.6	6.00

Total = 2.8 hrs 15.20 hrs

-Average turn around time T =2.8 / 5 = 0.56

-Weighted average turn around time W = 15.20 / 5 =3.04

FIFO with multiprograming:

CPU Headway:

-It is the amount of CPU time spent on a job.

-If 2 jobs are being multipogrammed, each job's CPU headway will be equal to half of the clock time elapsed.

-Average turn around time is .56 without multiprogramming

- And .6 with multiprogramming
- Note that average weighted turn around time improve.

Time	Event	No.of Jos (NJ)	CPU per job (1/NJ)	Elapsed time (ET=CT-PT)	Headway per job (HW=ET/NJ)	Job	Time left (TL =TL-HW)
10.0	Jo1 arrives					1	.3
10.2	Job 2 arrives	1	(1/1)	.2 (10.0-10.2)	.2 (.2/1)	1 2	.1(.3 - .2) .5

Time	Event	No.of Jos (NJ)	CPU per job (1/NJ)	Elapsed time (ET=CT-PT)	Headway per job (HW=ET/NJ)	Job	Time left (TL =TL-HW)
10.4	Jo3 arrives Job 1 Terminates	2	(1/2)	.2 (10.4 -10.2)	.1 (0.2/2)	1 2 3	- .4 (.5 - .1) .1
10.5	Job 4 arrives	2	(1/2)	.1 (10.5 -10.4)	0.05 (0.1 /2)	2 3 4	.35 (.4 - 0.05) .05 (.1– 0.05) .4
10.65	Job 3 terminates	3	(1/3)	.15 (10.65 -10.5)	0.05 (0.15 /3)	2 3 4	.3 (.35 - 0.05) - (0.05– 0.05) .35(.4-0.05)
10.8	Job 5 arrives	2	(1/2)	.15 (10.8 -10.65)	0.075 (0.15/2)	2 4 5	.225(.3-075) .275(.35-0.075) .1

Time	Event	No.of Jos (NJ)	CPU per job (1/NJ)	Elapsed time (ET=CT-PT)	Headway per job (HW=ET/NJ)	Job	Time left (TL =TL-HW)
11.1	Jo5 Terminates	3	(1/3)	.3 (11.1 -10.8)	.1 (0.3/3)	2 4 5	.125(.225-.1) .175(.275-.1) - (.1 - .1)
11.35	Job 2 terminates	2	(1/2)	.25 (11.35 -11.1)	.125 (0.25 /2)	2 4	- (.125- .125) .05 (.175– .125)
11.4	Job 4 terminates	1	(1/1)	.05 (11.4 - 1.135)	0.05 (0.05 /1)	4	-(0.05 – 0.05)

No.of Jos (NJ)	Run time (R)	Start time (Si)	Finished time (Fi)	Tun around time T=(Fi-Si)	Weighted Turn around time W= T/R
1	0.3	10.0	10.4		
2	0.5	10.2	11.35		
3	0.1	10.4	10.65		
4	0.4	10.5	11.4		
5	0.1	10.8	11.1		

No.of Jos (NJ)	Run time (R)	Start time (Si)	Finished time (Fi)	Tun around time T=(Fi-Si)	Weighted Turn around time W= T/R
1	0.3	10.0	10.4	0.4	1.33 (0.4 / 0.3)
2	0.5	10.2	11.35	1.15	2.3 (1.15 / 0.5)
3	0.1	10.4	10.65	0.25	2.5 (0.25 / 0.1)
4	0.4	10.5	11.4	0.9	2.25 (0.9 / 0.4)
5	0.1	10.8	11.1	0.3	3.0 (0.3 / 0.1)

No.of Jos (NJ)	Run time (R)	Start time (Si)	Finished time (Fi)	Tun around time T=(Fi-Si)	Weighted Turn around time W= T/R
1	0.3	10.0	10.4	0.4	1.33 (0.4 / 0.3)
2	0.5	10.2	11.35	1.15	2.3 (1.15 / 0.5)
3	0.1	10.4	10.65	0.25	2.5 (0.25 / 0.1)
4	0.4	10.5	11.4	0.9	2.25 (0.9 / 0.4)
5	0.1	10.8	11.1	0.3	3.0 (0.3 / 0.1)
				3.00	**11.38**

Average Turn around time T = Ti/n = 3.00 / 5 = 0.6

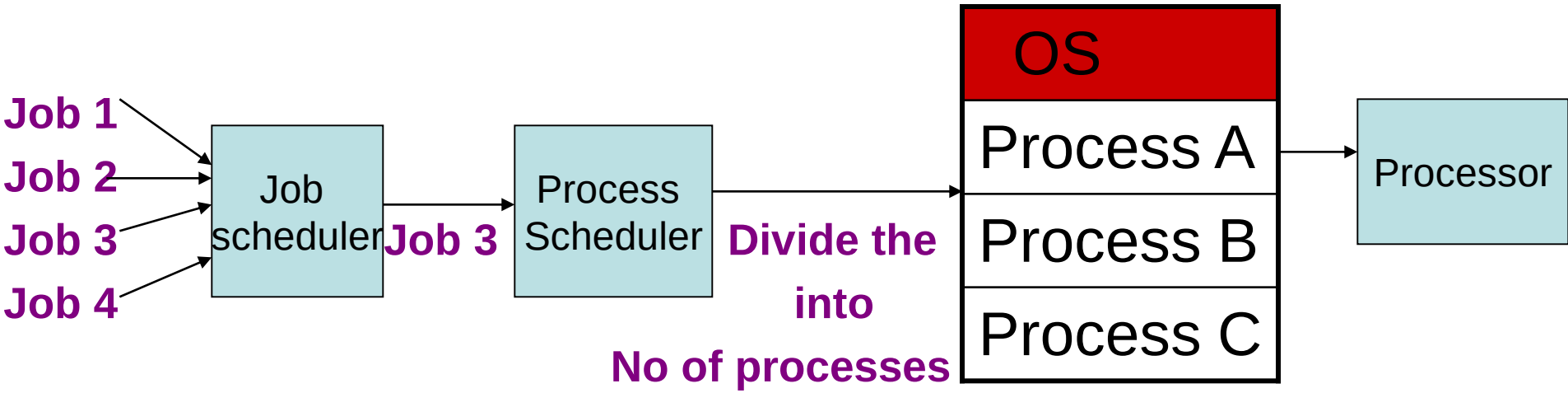
Average weighted turn around time W = 11.38 / 5 = 2.276

3.4 Process scheduling

-Once the job scheduler has selected job to run, it creates one or more process for this job

-The process scheduler does the assignment of processor to processes.

-It is also called as the dispatcher or low level scheduler.



Functions of Process Scheduling:

1) Keep track of the status of the process (ready, run or wait state)

2) Deciding which process gets a processor and for how long

3) Allocate processors to processes

4) De allocate processors from processes

Process Control Block:

-The traffic controller keeps track of the status of the processes by maintaining a database associated with each process in the system called PCB

-It contains entries like process identification, current state, priority, etc.

-All PCS's in the same state are linked together & forming a ready list, wait list, etc.

Process identification
Current state
Priority
Etc

-The wait list or blocked list may further be subdivided & providing one list that contains the reason why the process was blocked.

-The traffic controller is called whenever the status of a resource is changed.

-whenever a resource becomes free, one of the blocked list associated with that device can be placed in ready list.

Policies:

Assignment of processor to process that depends on the following events:

- The process is complete

-The process becomes blocked

- A higher priority process needs the processor

- A time quantum has elapsed

- An error occurs

The decision of choosing a process to run:

-It is made by scanning the PCB's ready list & applying some policies

The Ready list can be organized in Two ways:

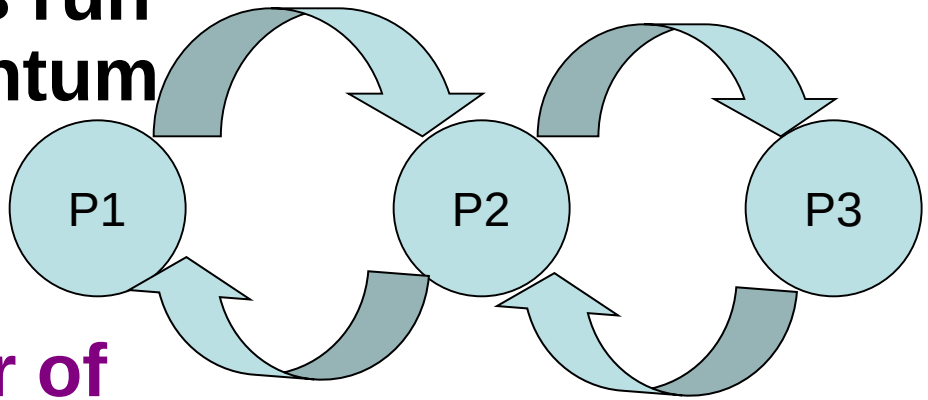
i) The top process on the list is run

ii) Scan the entire list & pick one process to run

The Various Process Scheduling Policies are: (RIM-PLSPM)

1) Round robin:

Each process in turn is run for a specific time quantum



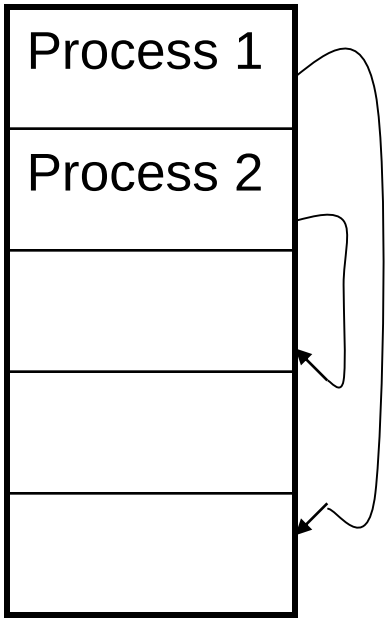
2) Inverse of the remainder of quantum:

-If a process has used its entire time quantum last time,

it goes to the end of the list

-if it has used only half

(because of I/O), it goes to the middle of the list.



(0.5)

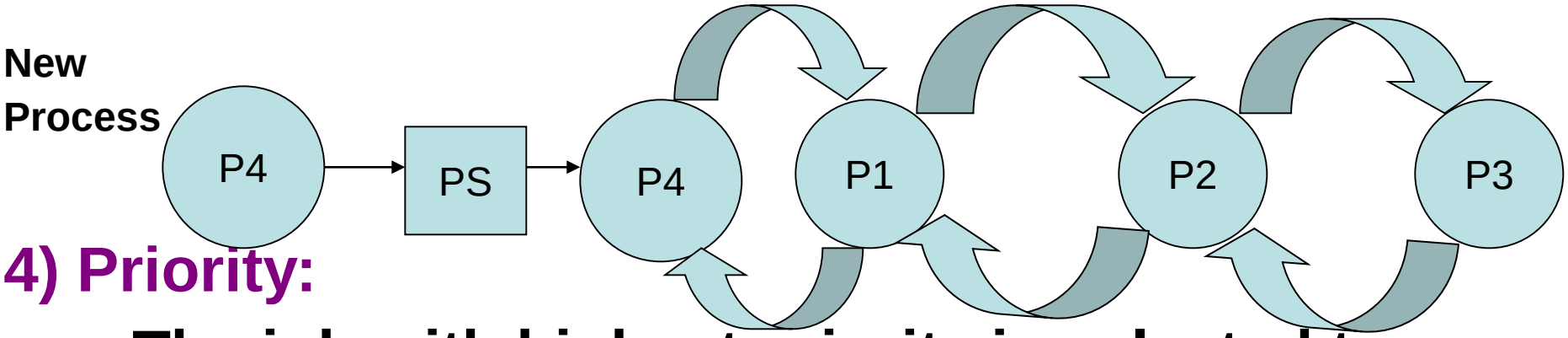
(0.4-0.2)

I/O interrupt

The Various Process Scheduling Policies are:

3) Multiple level – feedback variant on round robin:

-When a new process is entered, it is run for as many time quantum as all the other jobs in the system. Then regular round robin proceeds.



4) Priority:

-The job with highest priority is selected to run

5) Limited round robin:

- Jobs are run round robin for a fixed number of time. Then they are run only when there are no other jobs in the system.

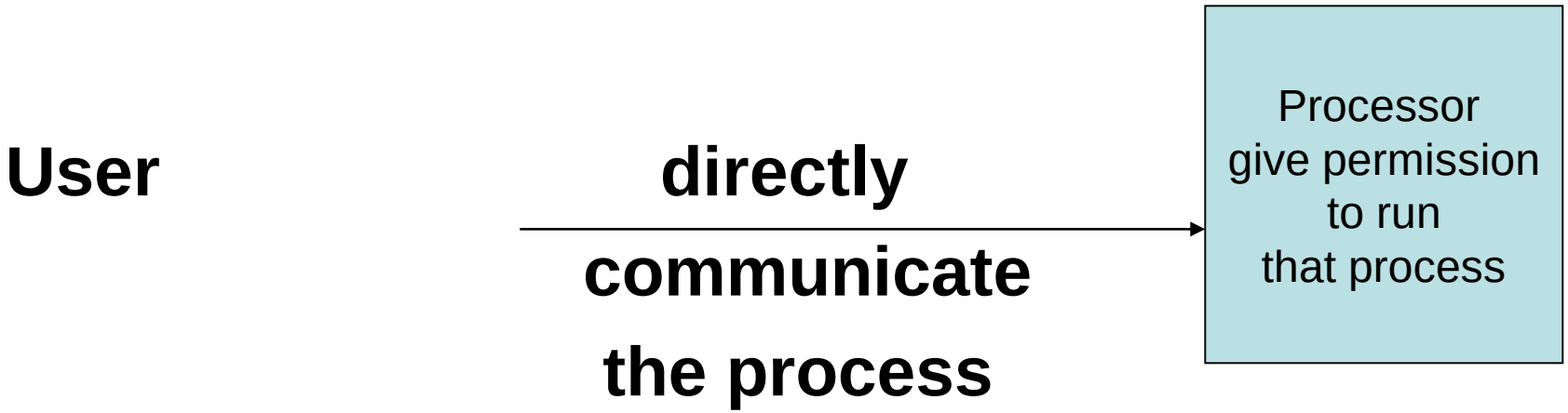
The Various Process Scheduling Policies are:

6) System balance:

-In order to keep the I/O devices busy, processes that do a lot of I/O operations are given preference.

7) Preferred treatment to interactive jobs:

-If an user is directly communicating with his process, that process is given the processor immediately after user input to provide rapid service



The Various Process Scheduling Policies are:

8) Merits of the job:

-In some cases, the system itself will assign priorities. It may assign high priority to short jobs.

Example:

MS Word – I/O bound higher, CPU Bound lower

Search any file – I/O bound lower, CPU Bound higher

3.5 Multiprocessor system

- A multiprocessor system has two or more processors, each of which have equal power.

- There are various way to connect and operate a multiprocessor system. Such as:

1) Separate system
2) Co-ordinate job scheduling
3) Master / Slave scheduling
4) Homogeneous Scheduling

1) Separate system:

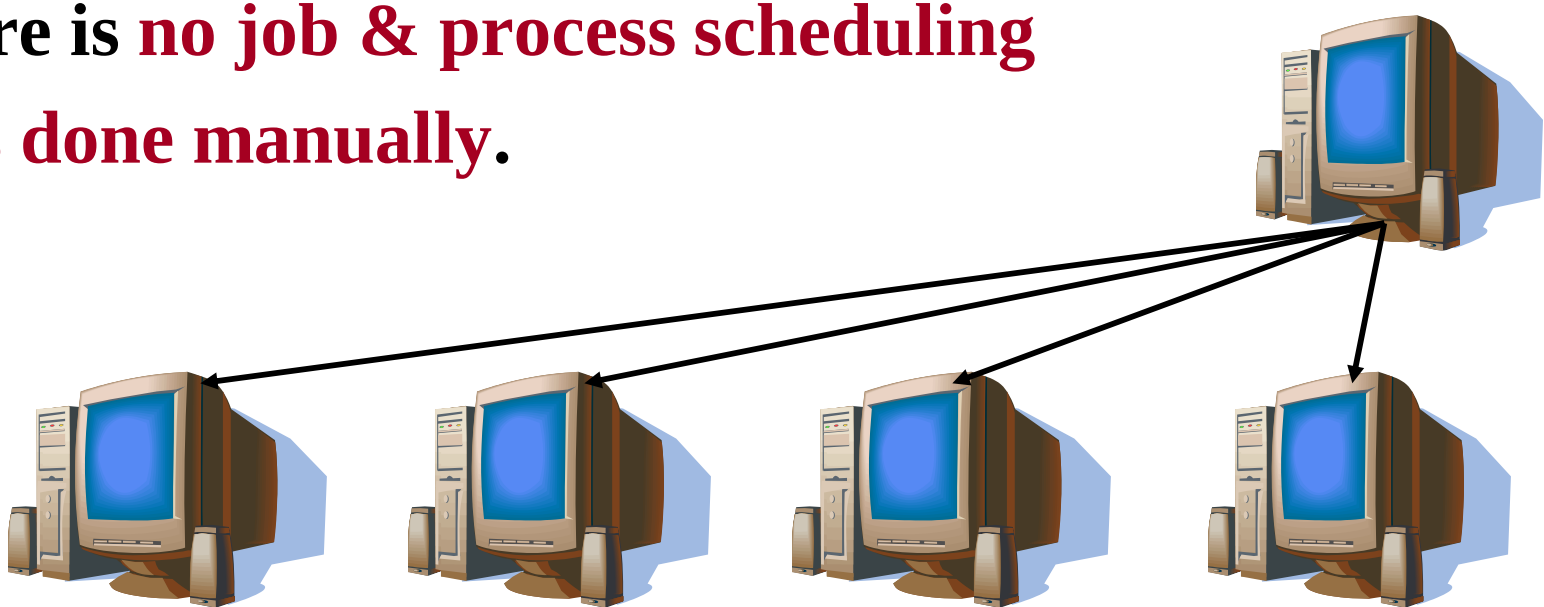
-Here, the systems can be logically subdivided into two or more separate systems, each with one processor, some main memory and peripheral devices.

Advantages:

-If one processor is being repaired, all other resources can be pooled into one large system rather than keeping them idle.

-There is no job & process scheduling

- It is done manually.



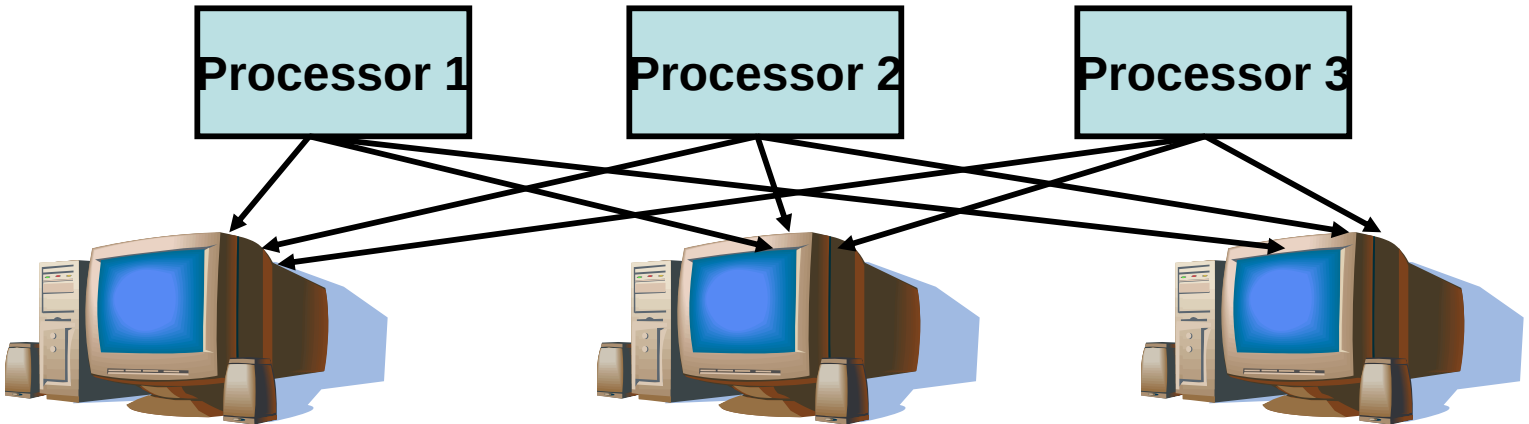
2) Co-ordinate Job scheduling:

- It is also called as **Loosely coupled multiprocessing.**

- Each processor is **associated with a separate system**

- When a job arrives, it may be assigned to any system.

- The assignment of a job to a system may be based on the **requirements, policies.**



3) Master / Slave scheduling:

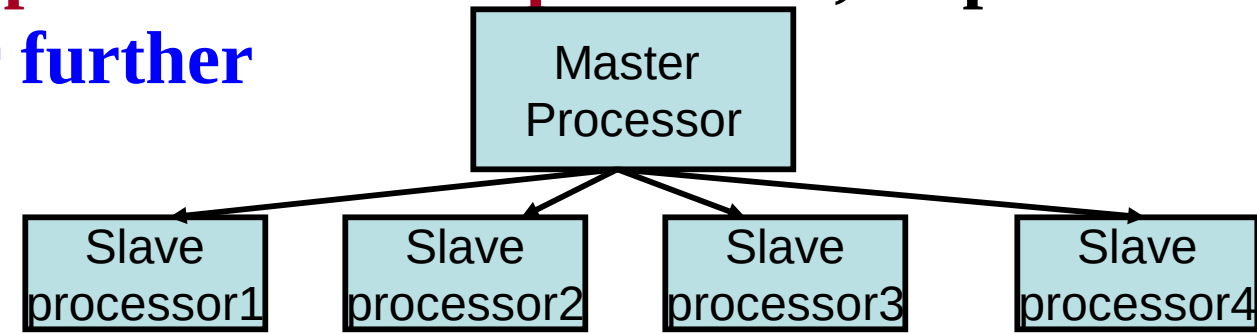
-One processor maintain the status of all processes in the system and schedules the work of all slave processors

- It is namely called as tightly coupled multiprocessing.

-The master processor selects a process to be run, finds a processor and issues a start processor instruction.

-The slave processor starts execution at the indicated memory location.

-When the slave encounters an I/O request, it generates an interrupt to the master processor, stop working & waits for further orders.



4) Homogeneous processor scheduling:

- The master / slave approach has several disadvantages:

- Under heavy scheduling loads, the master processor may become overload and cause a bottleneck.

- In this homogeneous approach all processors are treated equally both master & slaves.

- A list of processes & their status will maintained with the help of Process state list.

- Whenever a process is stopped due to I/O wait or time limit, the processor goes to the Process state list and finds another process to run.

- Each processor uses the same scheduling algorithm to select the next process to run.

Process State List

Process 1
Process 2
Process 3
Process 4
Process 5

3.6 Process synchronization

✓**The sharing resources requires coordination and cooperation to ensure correct operation.**

✓**In some case the coordination is forced upon, because of the scarcity of the resources.**

Two synchronization problems are:

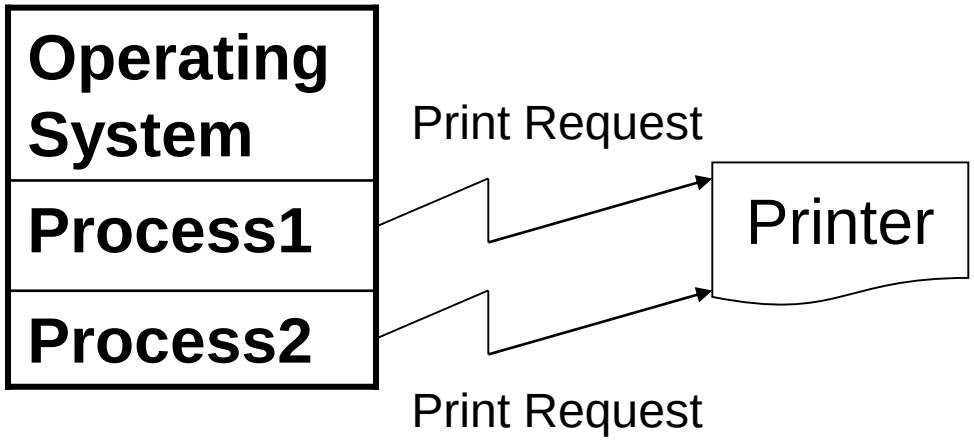
1. **Race condition**
2. **Deadly Embrace**

3.6.1.Race Condition:

❖ **Race condition occurs when the scheduling occurs.**

❖**Scheduling of two processes is so critical THAT DIFFERENT ORDER GIVES DIFFERENT RESULTS.**

The following figure explains race condition:



✓ **Two process are being run in a multiprogrammed environment.**

✓ **Depending on the scheduling the print outs of process1 may precede or process2.**

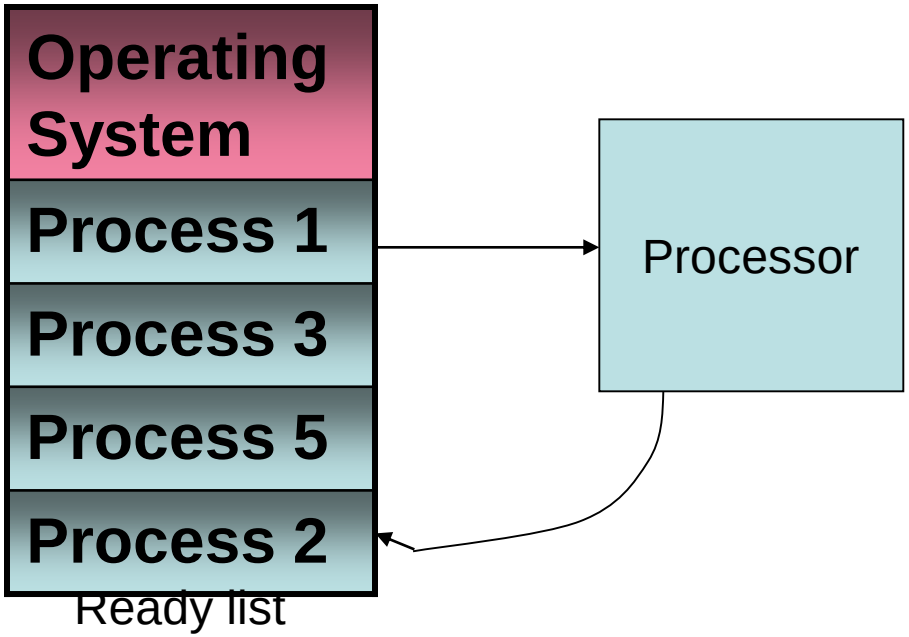
✓**If a process request a resource that is in free, that process automatically allocated to resource.**

✓**If a process request a resource that is already in use, that process automatically becomes blocked.**

- ✓ **When that resource becomes free, It can assigned to the process.**
- ✓ **This Request and release facility is handled by traffic controller of OS.**

This algorithm works well for a single processor, but trouble for multiple processors.

i) Single Processor:



Selecting a process from the ready list and it in the running state to run the process

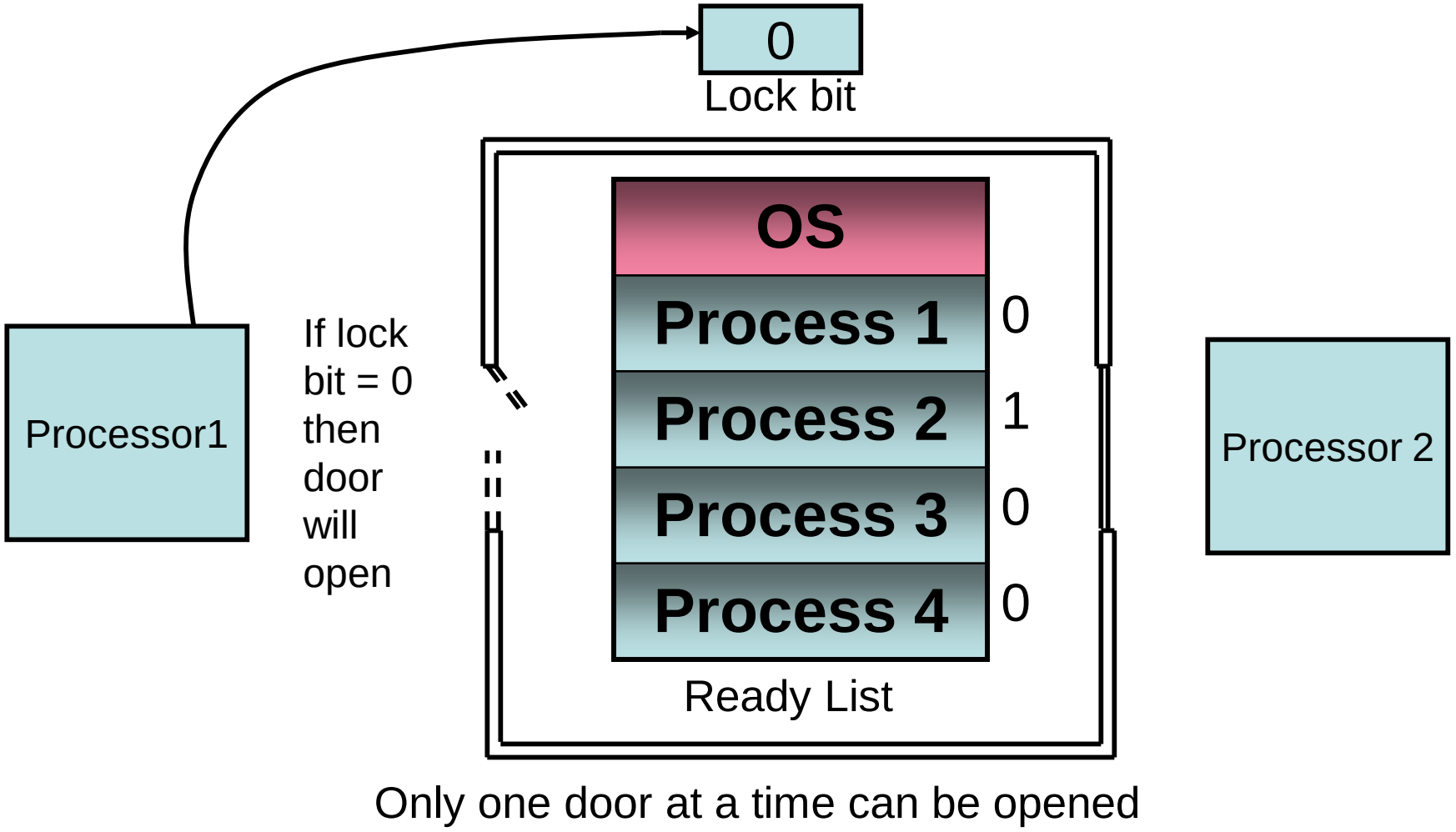
ii) MultiProcessor:

simultaneously two processor will choose the same process from ready list means, it is possible to upset the ready list.

In order to resolve this problem:

- **We use Lock bit mechanism.**
- **Before accessing ready list database, a processor checks a specific " Lock Bit ".**
- **If it is not set (lock bit =0) then database not in use.**
- **If it is set (lock bit =1) then database in use.**
- **When it has completed its function, the processor resets the bit as '0' .**

- **If another processor requires access to the ready list database in the mean time, it finds the lock bit set and has to wait until lock is removed.**
- **The second processor is temporarily in idle state.**



Multiprogramming

Several programs are run at the same time on a uniprocessor.

Multiprocessing

Multiprocessing is a type of processing in which two or more processors work together to process more than one program simultaneously

3.6.2 Synchronization mechanism:

Various synchronization mechanism are available to provide interprocess co-ordination and communication.

There are:

1. **Test and set instruction**
2. **Wait and signal mechanisms**
3. **P and V Operations on counting semaphores**
4. **Message communication**

1. Test and set instruction:

- ✓ **In most of the synchronization techniques, a physical entity called lock byte or semaphore must be used to represent the resource.**
- ✓ **A semaphore is an integer variable which can take 0 or 1 value.**
- ✓ **For each shared device, there should be separate lock byte.**
- ✓ **If lock byte = 0, then the resource is available.**
- ✓ **If lock byte = 1, then the resource is already in use.**

Before operation on a shared resource, a process must perform the following actions:

i) Examine the value of the lock byte (either 0 or 1)

ii) Set lock byte to 1 (if it is 0)

iii) If it was 1, go back to step (i)

After a process completes its use of the resource, it sets the lock byte to zero.

if we call the lock byte as X:

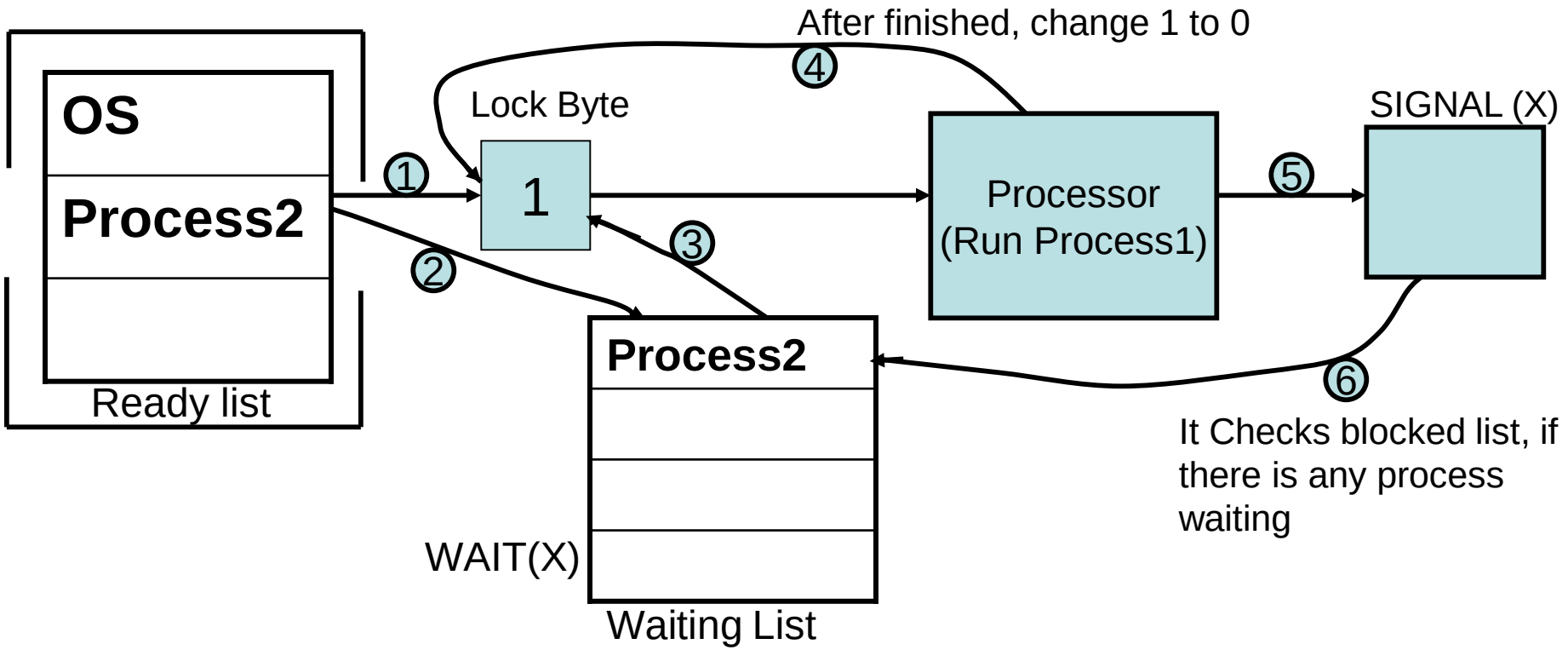
- the action **prior to the use of shared resource** is called **LOCK(X)** or **REQUEST(X)**

- the action **after use** is **UNLOCK(X)** or **RELEASE(X)**

2. Wait and signal Mechanism:

- ✓ In the previous method, the process does not stop, if the requested resource is not available.
- ✓ It continuously loops, testing the lock byte & waiting for it to change to zero.

Modified wait & lock Mechanism is:



LOCK (X):

1. Examine the value of the lock byte
2. Set it to 1
3. If it was 1 already, call WAIT (X)

UNLOCK (X):

1. Set lock byte to 0
2. Call SIGNAL (X)

- WAIT and SIGNAL are primitives of the traffic controller.
- WAIT (X) – sets the PCB of the process to blocked state and links it to the lock byte X.
- If there are any process waiting for X, one of them is selected and its PCB is set to ready state.

3. P and V operations on counting semaphores:

- A semaphore is a protected variable
- Whose values can be accessed by P and V initialization operation

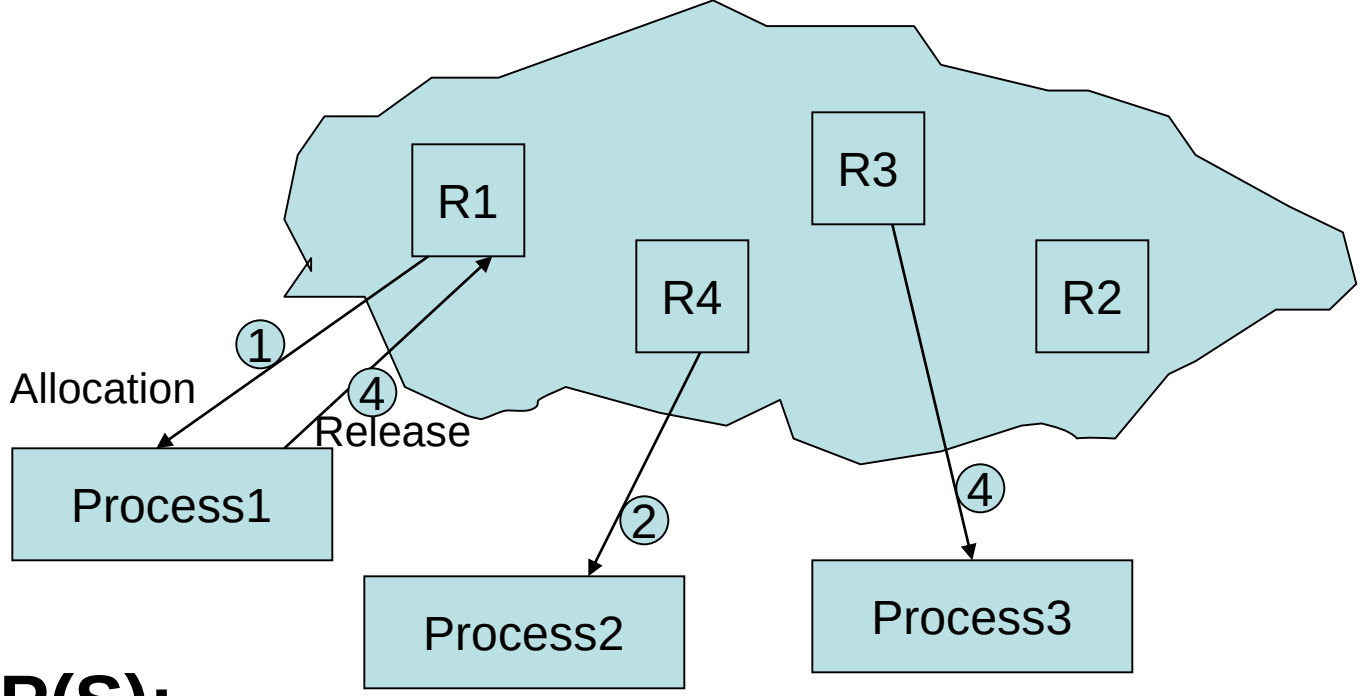
Semaphore initialize mechanism is:

- **P(S):**

  1. decrement the value of S (ie S = S - 1)
  2. if S < 0 , WAIT(S)

- **V(S):**

  1. increment the value of S (ie S = S + 1)
  2. if S < = 0, SIGNAL(S)



S = 4

P(S):

If R1 allocate to process1, then S = S – 1, S = 4 -1 = 3

V(S):

If process1 release R1, then S = S + 1, S = 3 + 1 = 4

- ✓ **Counting semaphores are useful when a resource is to be allocated from a pool.**

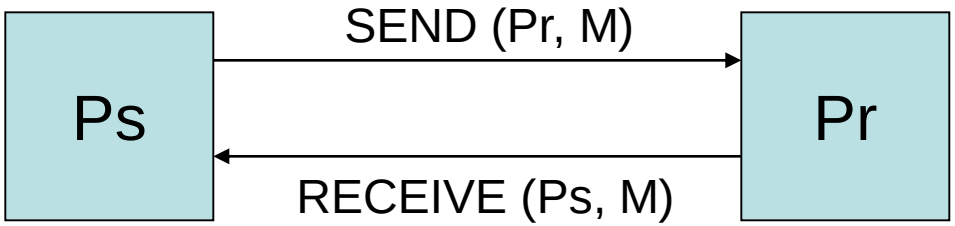
- ✓ **Semaphore is initialized to the number of resources in the pool.**
- ✓ **Each P operation decrement the semaphore by 1**
- ✓ **Each V operation increment the semaphore by 1**
- ✓ **If a P operation is attempted when the semaphore has been decremented by zero, then there are no resources available and it has to wait until it is returned to the pool by a V operation.**

4. Message communication:

- ➢ **The above three synchronization mechanisms provide communication between processes indirectly.**

Process to process Communication by means of the primitives:

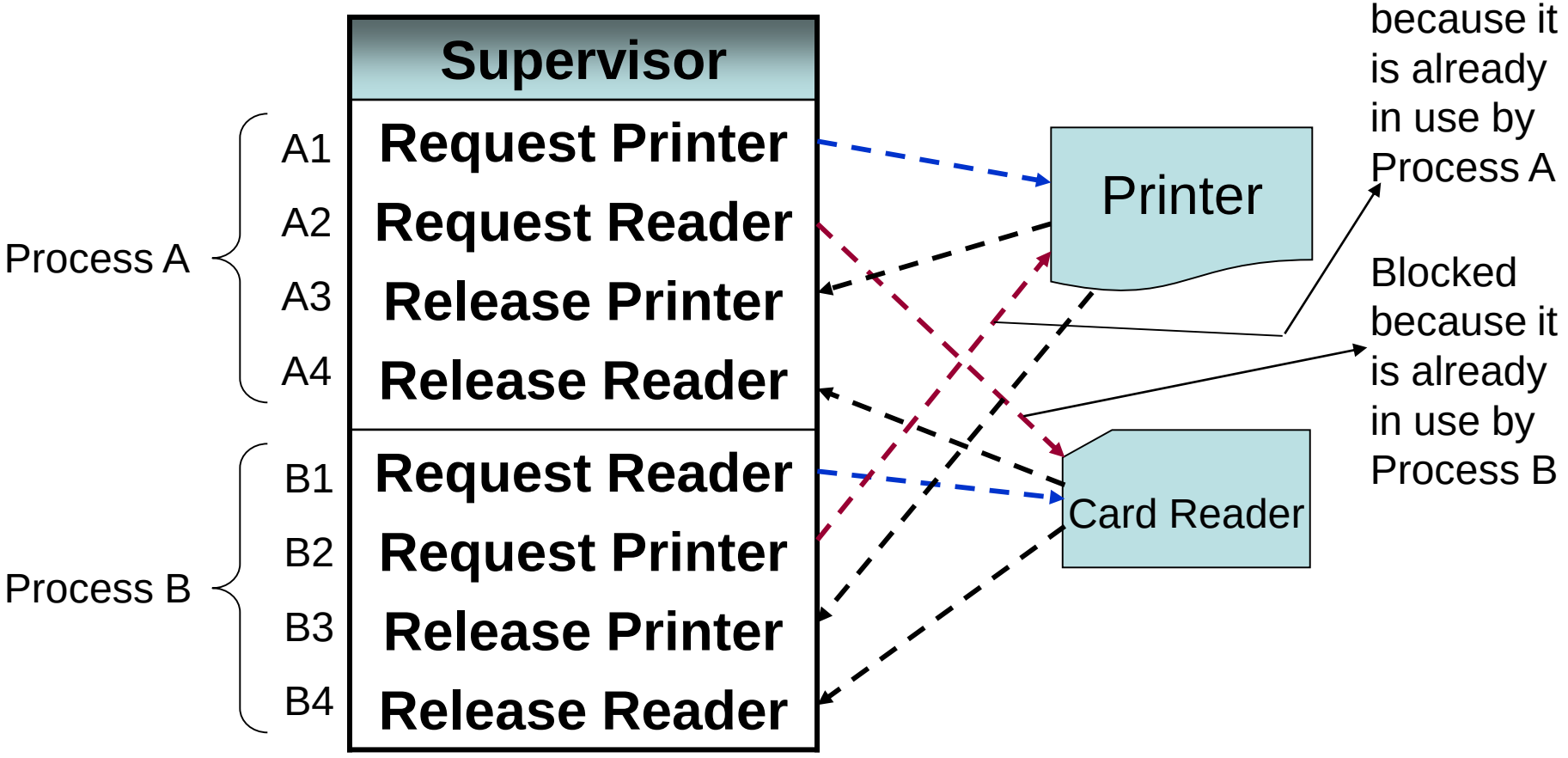
- SEND (Pr, M) and RECEIVE (Ps, M) can also be done.
- Pr and Ps are the names of the processes and M is the K-byte character string (Message)



- SEND (Pr, M) saves the message M for the receiver processor.
- RECEIVE (Ps, M) returns a message M to the sender process.
- If there are no message for the RECEIVE request, it becomes blocked until a message is sent to it.

3.6.3. Deadly Embrace (Deadlock)

- **A Deadly Embrace is a situation in which two processes are unknowingly waiting for resources held by each other.**



- **Process A & B are sharing use of the printer and card reader by means of the request and release operations.**
- **The Following sequence are performed correctly:**

  1. A1 A2 A3 A4 B1 B2 B3 B4

  2. B1 B2 B3 B4 A1 A2 A3 A4

Let us Consider a sequence:

- ✓ **A1 requests printer for process A and B1 Request reader for process B.**
- ✓ **If A2 requests reader for process A that time process A must be blocked because the reader is already in use by process B.**
- ✓ **When B2 request for printer for process B that time Process B must also be blocked because the printer is in by Process A**

- Here each process is waiting for the other to release a needed resource.
- This is a truly deadly embrace.

The following techniques for handling Deadly Embrace:

1. Preallocate all shared resources
2. Constrain allocation
   - Controlled allocation
   - Standard allocation pattern
3. Detect and recover

1. Preallocate all shared resources:

- Here the user declare all the devices and other resources he will use when submitting his job.
- The scheduler does not schedule any job until all the necessary resources are available.

Disadvantages:

1. A user may not know before execution time all the devices his job will use.
2. It is necessary to wait until all resources are available.
3. It is wasteful for the system to commit a device to a job when there is a small likelihood that the job will use that device.
4. A particular job may not need all devices for the entire duration of the job.

2. Constrained Allocation:

i) Controlled allocation:

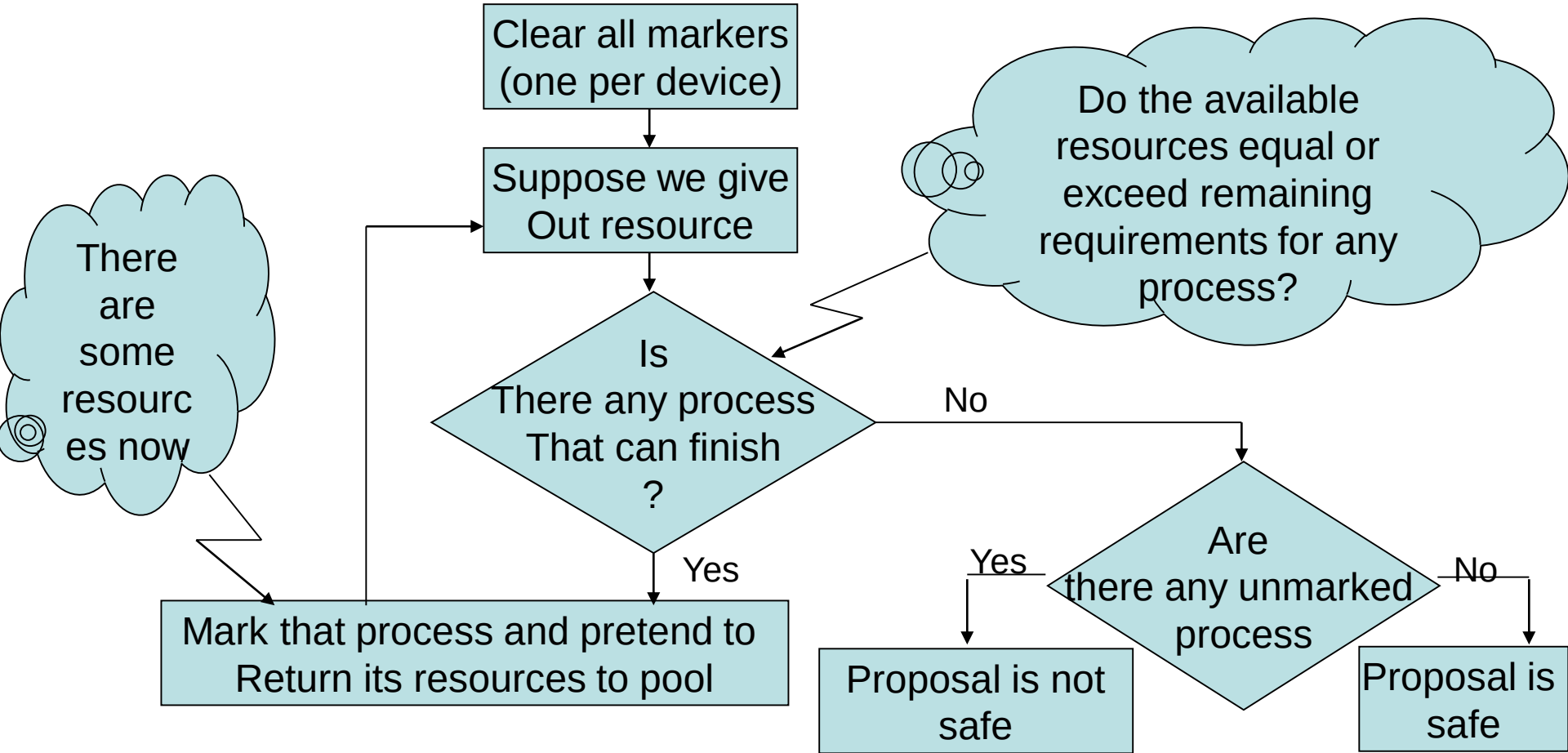
Thus with this method, the job must:

1. Declare in advance the maximum resources needed.

2. Before a resource is assigned – there is a possible Deadly Embrace.

Disadvantage:

1. **We must still know maximum needs in advance.**
2. **The algorithm is too conservative**
3. **A job may not actually use its maximum needs.**



ii) Standard Allocation pattern:

- ✓ In this scheme all resources are **assigned a unique number.**

Eg:

Reader = 1, printer = 2, Scanner = 3

Tape = 4, Disk = 5

- ✓ All allocation request must be in **ascending** order.

For Example:

- ✓ Reader(1), Scanner(3), tape(4) is a legal request sequence while **reader(1), tape(4), Scanner(3)** is not.
- ✓ This scheme **guarantees that** there **can't be any Deadly Embrace.**

Disadvantages:

1. **The standard sequence may not correspond to actual resource requirement ordering of the process.**

Eg:

- **If the tape is needed immediately and the printer may be needed later, The printer must be requested prior to the tape request.**

  (Printer = 2, Tape = 4)

3. Detect and recover:

- **It will allow deadly Embrace to occur as long as it can**

  i) detect it and

  ii) recover from the problem.

Detecting Deadly Embrace:

1. Arbitrarily assign each resource and process a unique number.
2. Allow processes to apply software locks when seizing resources.
3. Set up tables for keeping track of resources and process.

Resource Assignment Table (RATBL)

No.Of Resources	Assigned to Process no
1	
2	
3	
.	
.	

Process Wait Table (PWTBL)

Process	Resource being waited for
1	
2	
3	
.	
.	

4. **Make Appropriate RATBL and PNTBL entries as resources are seized and released.**

5. **When a locked resource is requested, use the deadly embrace detection algorithm.**

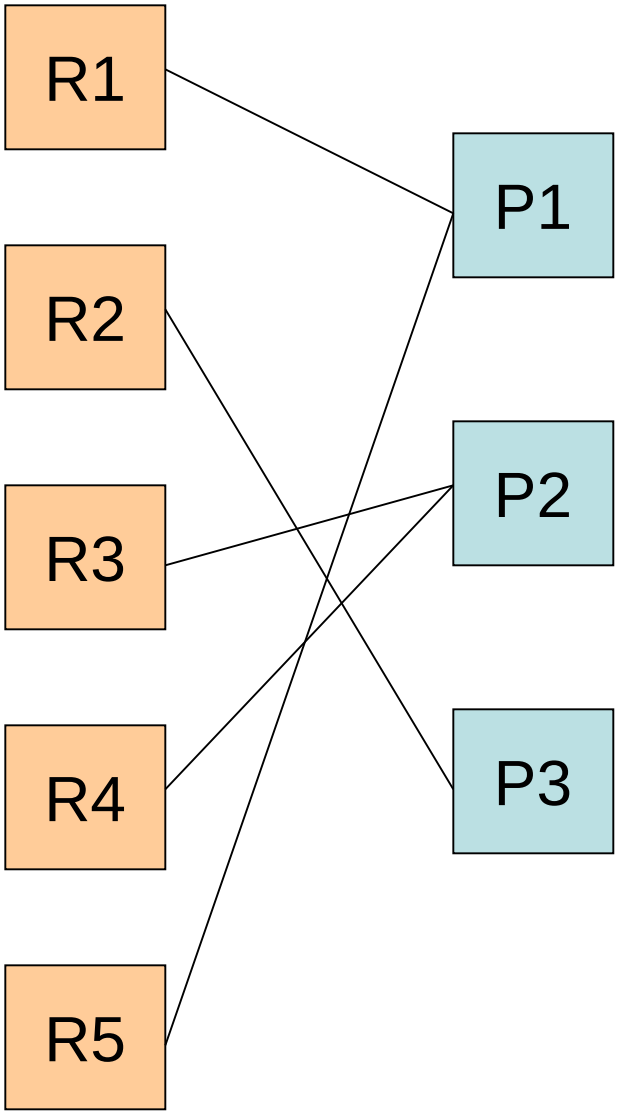
Consider the following sequence:

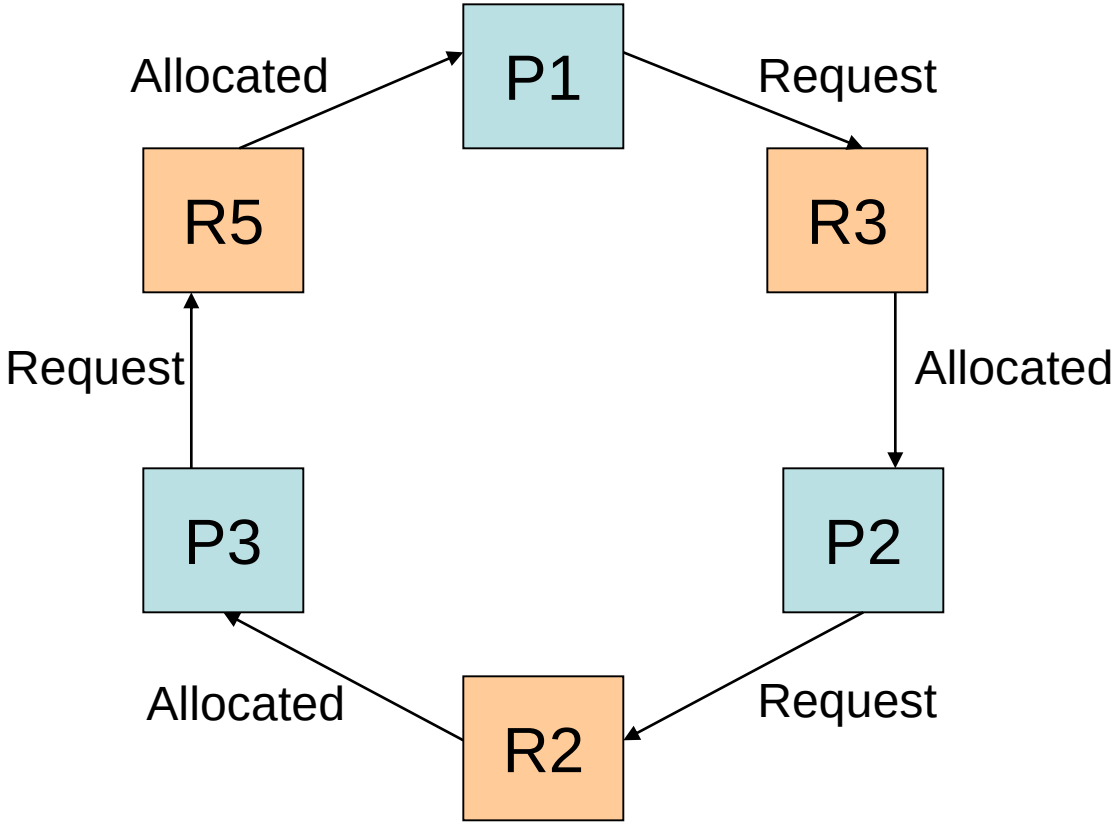
(RATBL)

Resource	Process
1	1
2	3
3	2
4	2
5	1

(PWTBL)

Process	Resource
1	3
2	2
3	5


R1
R2
R3
R4
R5
P1
P2
P3


P1
Allocated
Request
R5
R3
Request
Allocated
P3
P2
Allocated
Request
R2

- When deadly embrace is detected, it must be removed
- One of the jobs must release one or more resources to undo the deadly embrace
- Backtracking can be difficult

There are some techniques used:

1. Decompute algorithm (Undo computation)
2. Make a snapshot (or checkpoint) which is used to restore conditions

- Detection is always possible, but recovery is not always possible
- Even when recovery is possible, it is usually difficult

UNIT – IV
DEVICE MANAGEMENT

This chapter focuses on the management of I/O devices:

- **Such as printers, card readers, tapes, disks and drums**
- **And supporting devices such as control units or control channels**

Functions:

1. Keeping track of the status of all devices, which requires special mechanisms such as Unit Control Block associated with each device.

2. Deciding on policy to determining who gets a device, for how long and when.

There are 3 technique for implementing the policies of device management:

i) **Dedicated** - **A technique whereby a device is assigned to a single process**

ii) **Shared** - **A technique whereby a device is shared by many processes**

iii) **Virtual** - **A technique whereby one physical device is simulated on another physical device**

3. Allocated – physically assigning a device to process

4. De-allocation policy & techniques.

4.1 Techniques for device Management:

Three major Techniques are used for managing and allocating devices:

i) Dedicated Devices

ii) Shared Devices

iii) Virtual Devices

i) Dedicated Devices:

- A dedicated device is allocated to a job for the job's entire duration
- It is difficult one

Ex:

- To share a card reader, printer or tape
- Unfortunately dedicated assignment may be inefficient, if the job does not fully and continually utilize the device

ii) Shared Devices:

- Some devices such as disks, drums and most other Direct Access Storage Devices (DASD) may be shared by several process
- But the management of a shared device can become quite complicated.

For Ex:

- If two processes simultaneously request a Read from Disk A, Some mechanism must be employed to determine which request should be handled first.
- This may be done partially by software or entirely by hardware.

Policy for establishing which process request is to be satisfied first might be based on:

a) A priority list

b) The objective of achieving improved system output

For Ex:

- **By choosing whichever request is nearest to the current position of the read head of the disk.**

iii) Virtual Devices:

- **A disk may be easily shared by several users, we have converted a dedicated device to a shared device.**

Ex:

- **Changing one card reader into many "Virtual" card readers.**

This technique is equally applicable to a large number of peripheral devices, such as teletypes, printers and most dedicated slow input / output devices.

Ex:

Fax software can act as a virtual printer. When print is selected the document is sent to a fax / modem, which then sends information to another fax machine instead of a printer printing the file.

4.2 Device Characteristics:

Peripheral devices can be generally categorized into two major groups:

i) Input or Output Devices ii) Storage Devices

i) Input or Output Devices:

- An input device is one by which the computer "sense" or "feeds" the outside world.
- They are devices to read punched cards, punched papers, tape or message typed on typewriter like terminals.
- An output devices is one by which the computer "affects" or "controls" the outside world.
- It is a device to punch holes in cards or paper tape, print letters and number on paper, or control the typing of typewriter like terminals.

ii) Storage Devices:

A storage device is a mechanism by which the computer may

- store information (a procedure called writing)

- in such a way that this information may be retrieved at a later time (reading).

These storage devices can be differentiated based on the variation of access time (Tij) where;

Tij = time to access item j from current position item i

The storage devices are classified into the following categories:

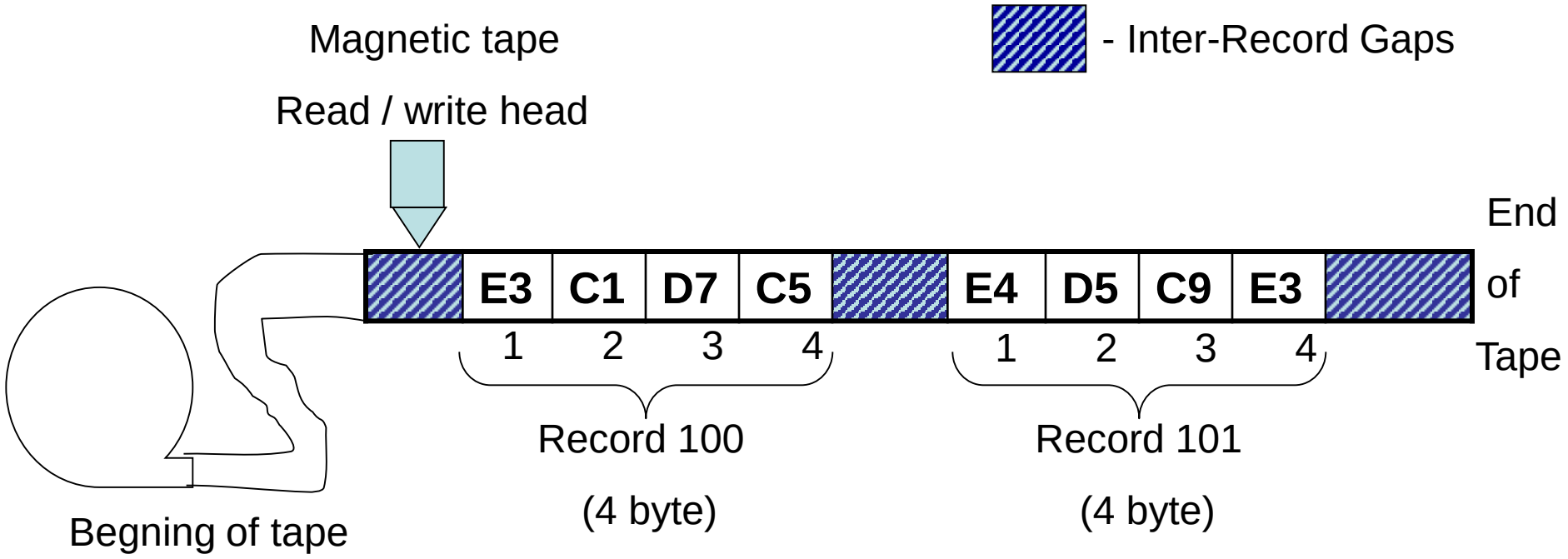
1) Serial Access devices
2) Completely Direct Access Devices
3) Direct Access Storage Devices

1) Serial Access Devices: (8 Mark)

- A serial access storage device can be characterized as **sequential positioning** and **accessing of information**.
- Access to an arbitrary stored item requires a "**Linear Search**".

Ex:

Magnetic Tape Unit (MTU) – serial Access Device



- Information is usually stored as groups of bytes called records.
- In this figure all the records are four bytes long
- Each record can be identified by its physical positioning on the tape.
- The current record counter shows the record under the read / write head of the MTU.
- If the tape is positioned at its beginning, to read record number 101, it is necessary to skip over all 100 records to reach record 101.

Some of the I/O Commands used with the MTU are:

i) Read the next record (or) write the next record

ii) Read the last Record

iii) Rewind to the beginning of the tape at high speed

iv) Skip forward (or backword) one record without actually reading or writing data

v) Write tape mark (record type can be written)

Tape marks are somewhat analogous to putting **bookmarks** in a book to help find your place.

Typical Characteristics of a MTU are:

Density	- 1600 bytes per inch
Speed	- 200 inches per second
Length	- 2400 feet long
Maximum Access	- 2 minutes
Random Access (Average)	- 1 minute
Serial Access (at one record read the next)	≈ 4 ms

- The inter record gap (IRG) shown in the diagram is necessary due to the physical limitations of starting and Stopping the tape.
- The IRG is usually from ¼ to ¾ of an inch (equal to 400 to 1200 bytes).
- To minimized gap waste, blocking is Used.

Blocking:

- It is placing multiple logical records into one physical record.
- Blocks are typically 800 to 8000 bytes long

- When the file system receives a request to read a logical record,
- Then entire physical record is read into a buffer area in the main memory and the request is satisfied by extracting the logical record from the buffer area.

Three Advantages to Blocking:

1. Fewer i/o operations are needed. Since each i/o operation reads & writes multiple logical records at a time.
2. Less wasted space. Since record length is larger than the IRG length

3. Smaller tape space is covered when reading many records (less distance to travel). Since there is less wasted space.

Three Major Disadvantages:

1. Software overhead and routines are required to do the blocking, detecting and record keeping.

2. Buffer space is wasted (when we want only 80 bytes of data, we have to read 8000 bytes)

3. There is more likelihood of tape errors. Since long records are being read.

2) Completely Direct Access Devices: (8 Mark)

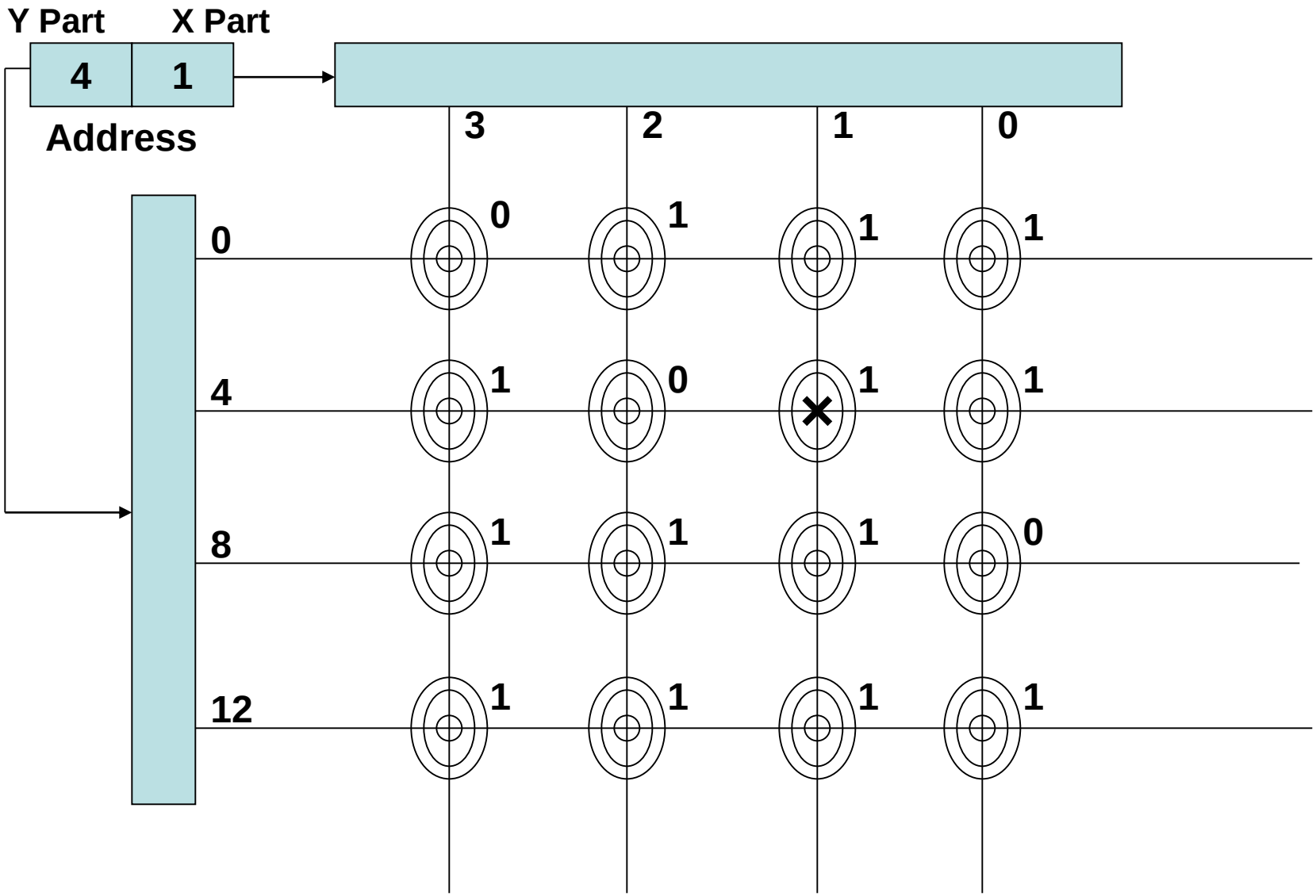
- A completely direct access device is one in which the access time Tij is a constant

Ex:

- Magnetic core memory, semiconductor memory, read-only wired memory and diode matrix memories
- In the given diagram, 16 circles represented magnetic ferrite cores.
- All cores are connected by wires.
- When the hardware receives a read request for some address the address is decomposed into X and Y portions
- The hardware then selects the appropriate core.

- This is done by passing current through the two selection wires.
- Only the core at the specified (x, y) co-ordinate receives a "double dose" of current
- Then double dose of current is enough to "flip" the magnetic field from one to zero.
- When it changes magnetic state, then a third sensing wire is passed through all the cores.
- If the appropriate core was one and it was switched to zero, a current would be included in this wire.
- If it was already zero, there would be no included current.
- Note that: The process of reading a core destroys its information, thus an additional cycle is necessary to write the information back.

Magnetic Core Memory



3. Direct Access Storage Devices (DASD): (15 Mark)

In this section we have two Example, there are:

a) Fixed – Head Drums and Disks

b) Moving – Head Drums and Disks.

3.a) Fixed – Head Drums and Disks: (8 Mark)

- It is similar to magnetic drum.
- A magnetic drum can be viewed as several adjacent strips of magnetic tape wrapped around a drum.
- So that the ends of each tape strip join.
- Each tape strip called a track.
- Each track has a separate read/write head.

- The drum continuously revolves at high speed, so that the records repeatedly pass under the read/write heads.
- Each individual record is identified by a track number and then a record number

Ex:

Record (3, 1) is ‘E4D5C9E3’

- Typical magnetic drums spin very fast and have several hundred read/write heads.
- Random access to read / write can be accomplished in 5 or 10 milliseconds

5-2.2.3.1 *Fixed-Head Drums and Disks*

Figure 5-4 depicts a DASD storage device similar to a magnetic drum.

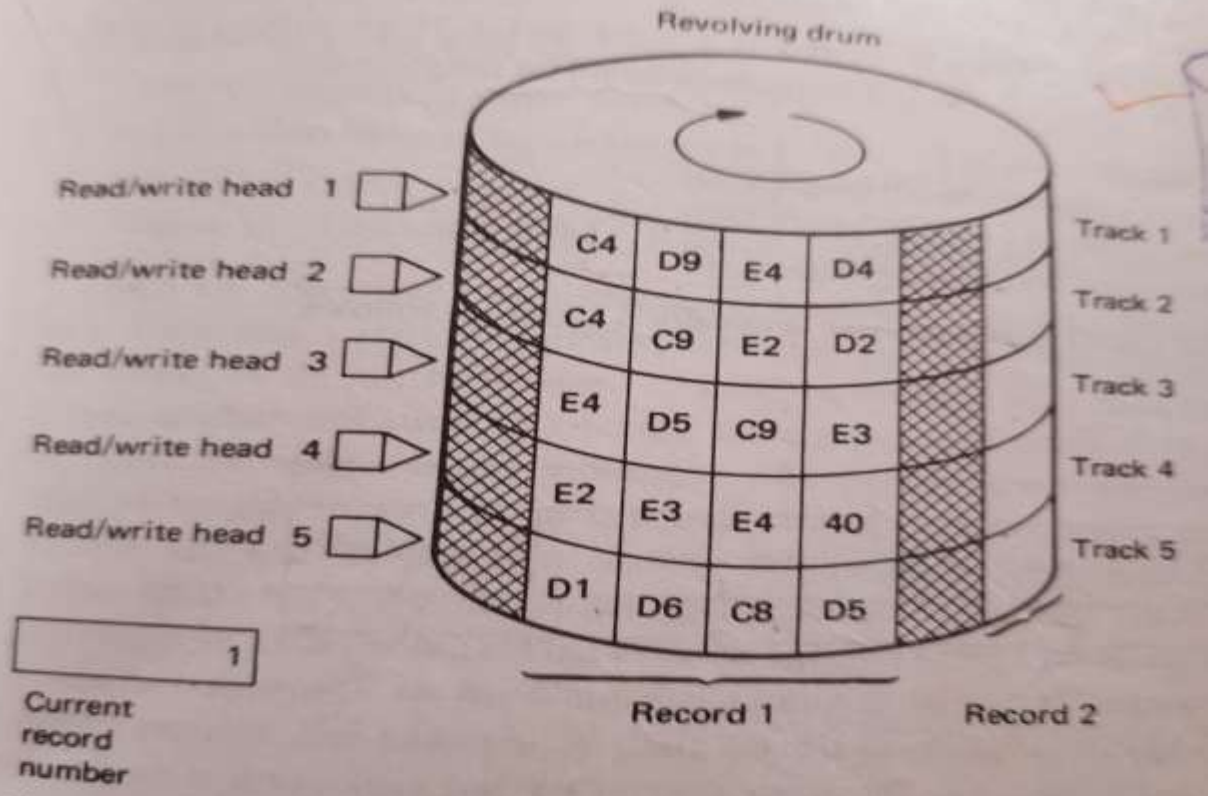


Figure 5-4 Fixed-head drum DASD

Necessary to specify the DASD address in the i/o channel command. The i/o command would be:

i) DASD record (3, 1) into

ii) memory address (28768)

iii) length (80)

In most cases this i/o command is broken into two parts:

1) DASD positioning

2) Data transfer operation

Thus the above i/o command would become:

POSITION to DASD record (3, 1)

READ into memory address (28765) length (80)

Typical Characteristics of a Drum are:

Rotation speed	- 10 ms
Maximum access time	- 10 ms
Average random access time	- 5 ms
Serial Access (depending on the length of record)	< 1 ms
Capacity	= 8 million bytes (256 heads, 32,000 bytes per track)

- The drum is **always rotating**.
- Hence the **inter record gap (IRG) is not need**.

One use of IRG on a tape:

- IRG allows time for CPU for processing the last record before it reads the next record.
- When reading records sequentially on a drum. Since there is no IRG after processing the last record.
- The CPU may have to wait for the drum to rotate all the way around to get the next record.
- This can make a significant difference in time for sequential processing.

Drum Record	Data Record
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J

- To avoid the above mentioned problem of timing, "Inter Leaving" technique is used.

Ex:

- If there are 10 records (A, B, C, ….. , J) to be stored on a drum that has 10 records per track.

The records can be stored as follows:

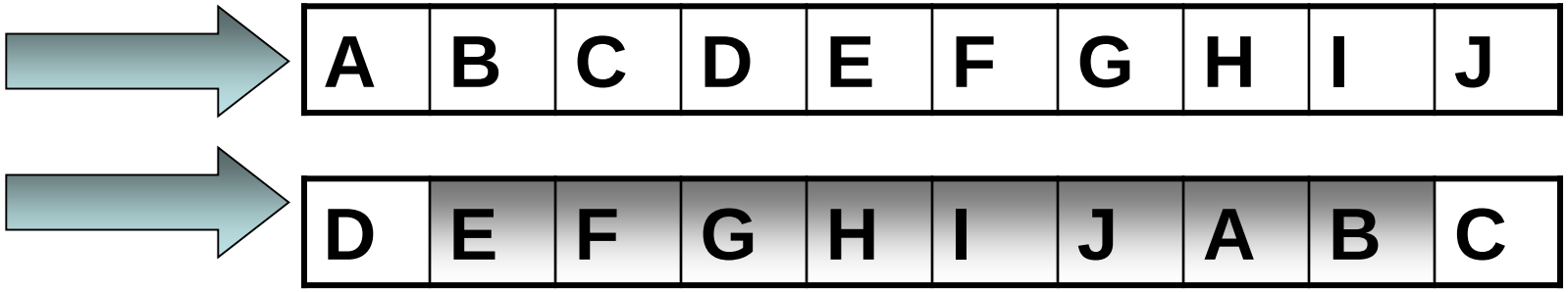
Drum Record	Data Record
1	A
2	B
3	C
4	D
5	E
6	F
7	G
8	H
9	I
10	J

- **Let us assume that these records are frequently read sequentially (ie, Read record from A, B, C, … , J)**
- **Assume that the job takes 1ms to read and 2ms to process the record.**

- When the CPU is processing record ‘A’ after reading it, the drum would have rotated to the beginning of drum it reaches record 4.
- In order then to read record ‘B’, the drum must rotate around to record 2 again, which will take 8ms.

= 10 X [1ms (read record) + 2ms (process record) + 8ms (access next record)]

= 10 X 11ms = 110ms

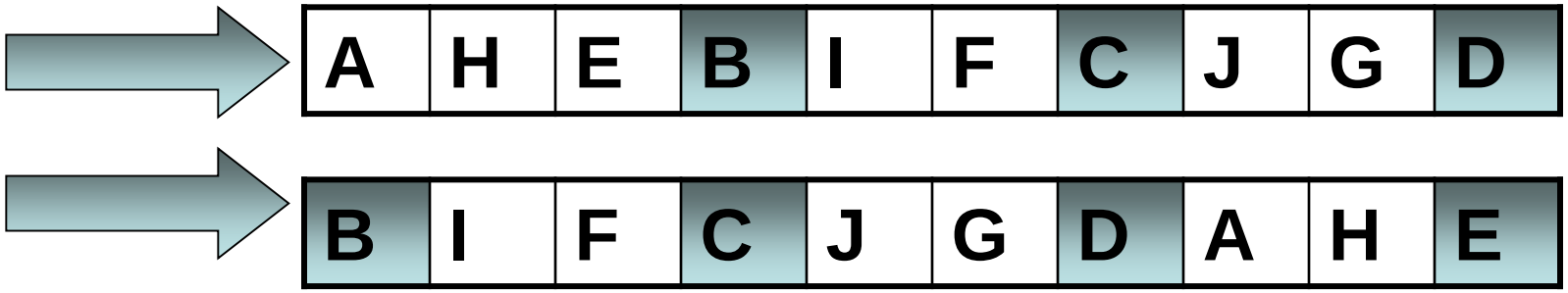


Alternatively, the record could have been stored as follows:

Drum Record	Data Record
1	A
2	H
3	E
4	B
5	I
6	F
7	C
8	J
9	G
10	D

- **This time, by the time CPU finishes reading and processing record A, the drum would again have rotated to the beginning of drum record 4.**

- The data record B is stored at drum record 4 which is wanted for processing next.



= 10 X [1ms (read record) + 2ms (process record)]

= 10 X 3 = 30ms

- In comparing these two schemes for storing data record, it is found that the second scheme is four times faster than the first scheme.
- This difference becomes quite significant when there are 1,000 or 10,000 record to be processed.

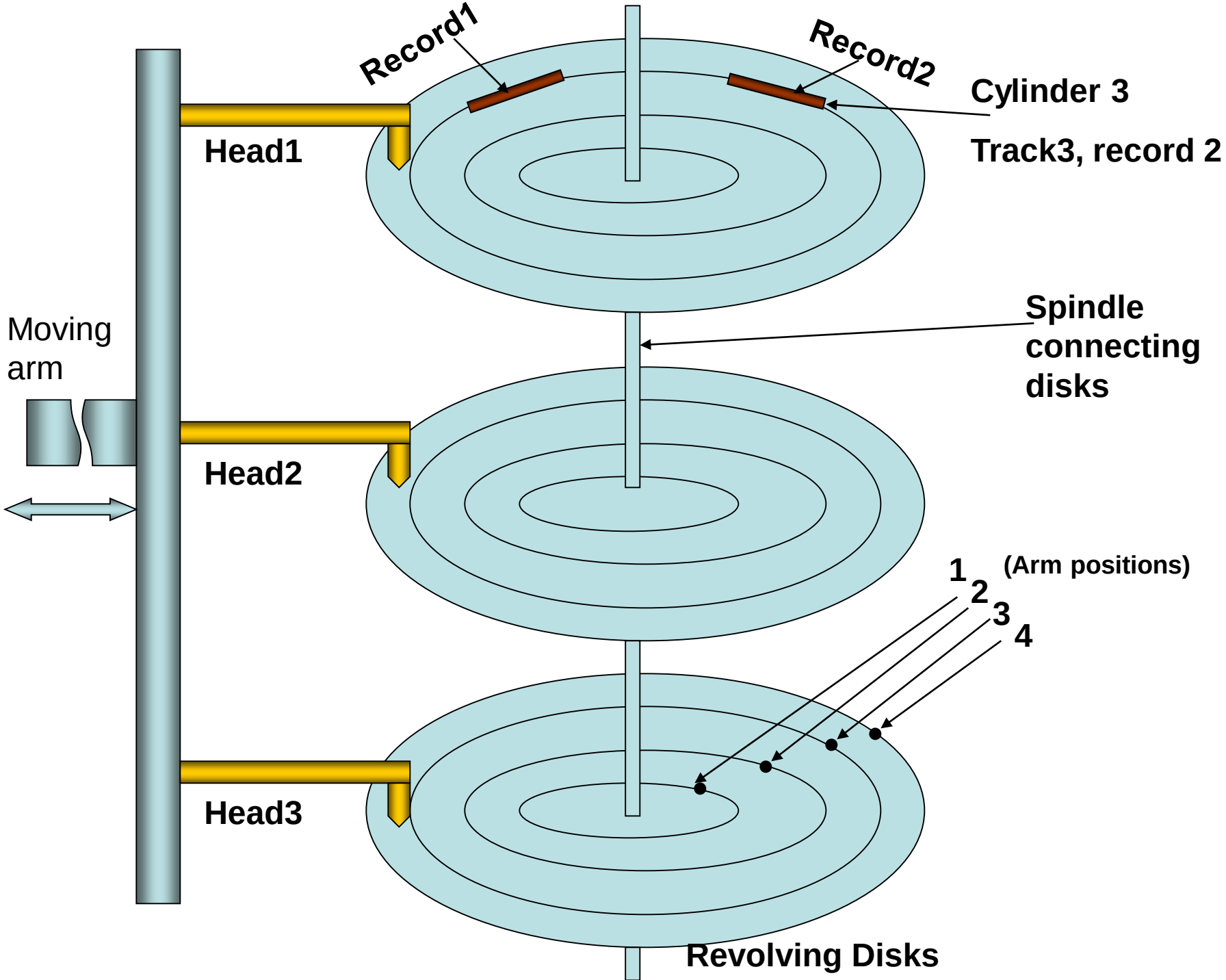
- **Blocking is also helpful in improving the performance of sequential processing**
- **Instead of IRG's, If the drum uses blocking to minimize the amount of wasted space.**

3.b) Moving – Head Drums and disks: (8 Mark)

- **The magnetic disk contains one or more flat disks**
- **Each with a series of concentric circles, one per read/write head.**
- **The heads are physically moved from track to track, such units are called moving arm or moving head DASD's.**
- **Each position is called cylinder.**

Identification of a particular record stored on the moving head DASD shown in the given Diagram:

- It is necessary to specify the arm position, track number and record number.
- Thus to access a record, then to wait for the correct record to rotate under the read/write head.
- A separate I/O command, called seek is used to position the arm on a cylinder.
- Once the arm is positioned, the particular track (head) and the particular record on that track can be read.


Record1
Record2
Cylinder 3
Track3, record 2
Head1
Moving arm
Spindle connecting disks
Head2
1
2
3
4
(Arm positions)
Head3
Revolving Disks

- **Typical moving head disk DASDs have 10 to 50 tracks and 200 to 400 arm positions with characteristics such as:**
- **Maximum access time = 75ms**
- **maximum arm movement = 55ms**
- **maximum rotation = 20ms**
- **Average (random) access = 30ms**
- **Serial access < 1ms**
- **Capacity = 100 million bytes (400 cylinders, 20 heads, 16,000 bytes per track)**

4.5 I/O traffic Controller, I/O Scheduler, I/O Device Handlers: (15 Marks)

The functions of device management can be conveniently divided into 3 parts:

1. I/O traffic controller **2. I/O scheduler**

3. I/O Device Handler

i) I/O traffic controller:

Keep track of the status of all devices, control units and channels.

The traffic controller accepts to answer at least three key questions:

1. Is there a path available to service an i/o request?
2. Is more than one path available?
3. If no path is currently available, when will one be free?

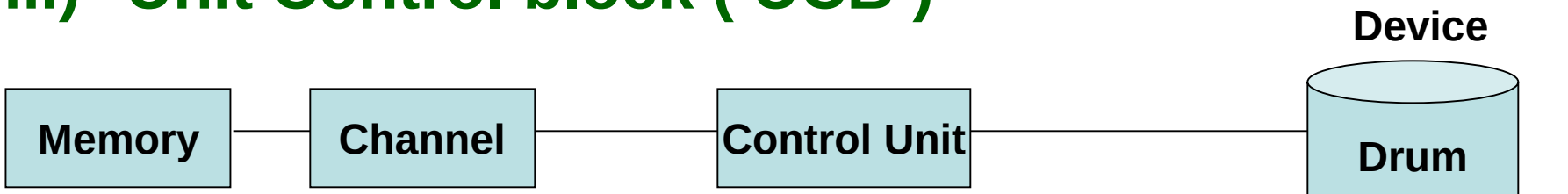
- In order to maintain these questions, the traffic controller maintains a database. It maintains 3 control blocks:

It maintains 3 control blocks:

i) Channel control block (CCB)

ii) Control unit control block (CUCB)

iii) Unit Control block (UCB)



Channel Control Block (CCB)

Channel identification
Status identification
List of control units connected to this channel
List of processes waiting for this channel

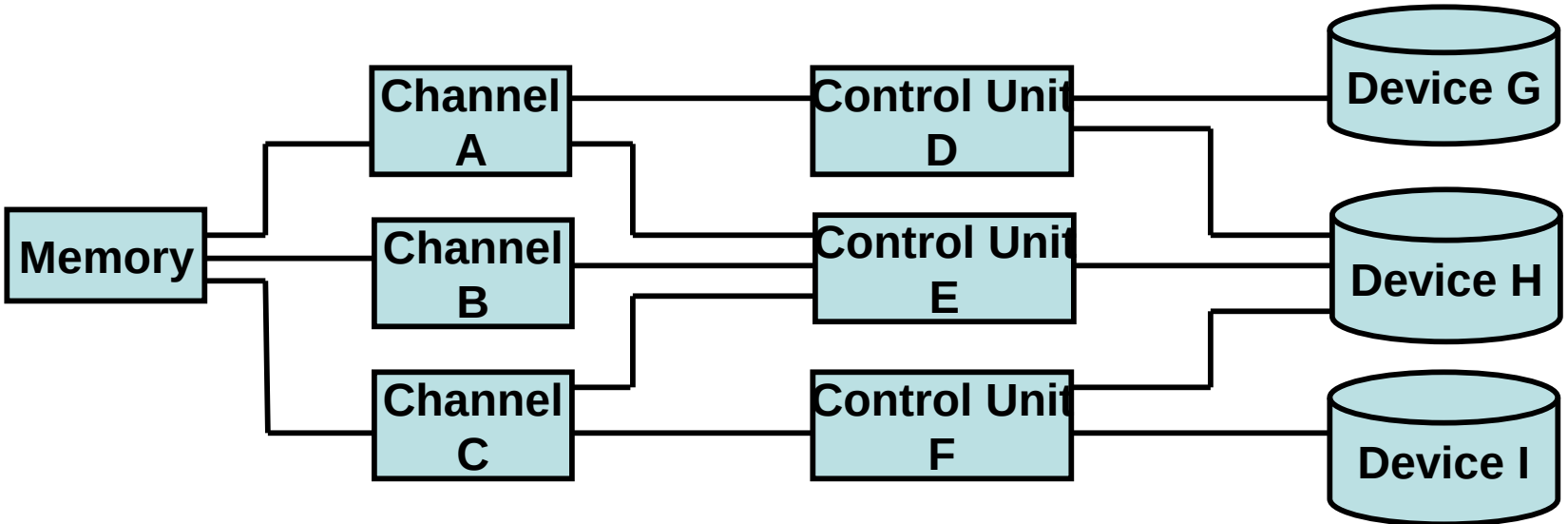
Control Unit Control Block (CUCB)

Control unit identification
Status of Control unit
List of channels connected to this control unit
List of processes waiting for this control unit

Unit Control Block (UCB)

Device unit identification
Status of Device
List of control nits connected to this Device
List of processes waiting for this Device

- The first question can be answered by starting at the desired UCB and working back through the connected units and channels trying to find a combination that is available.
- The second question may be important, especially when I/O configuration is symmetric, since choosing one path may block out other I/O request.
- For third question, when an I/O completion interrupt occurs, one or more components(ie device, control unit, channel) become available again.



ii) I/O Scheduler:

- If there are more I/O requests pending than available paths, it is necessary to choose which I/O requests to satisfy first.
- I/O requests are not normally time sliced that is, not usually interrupted until it has been completed.
- Most channel programs are quite short and terminate within 50 – 100ms.
- Many different policies may be incorporated into the I/O scheduler like higher priority request is assigned.

iii) I/O Device Handler: (8 Marks)

- The I/O device handler provide detailed scheduling algorithms that are dependent upon the peculiarities of the device type.
- There are different device handler algorithm for each type of I/O device.

a) Rotational Ordering b) Alternate Address c) Seek Address

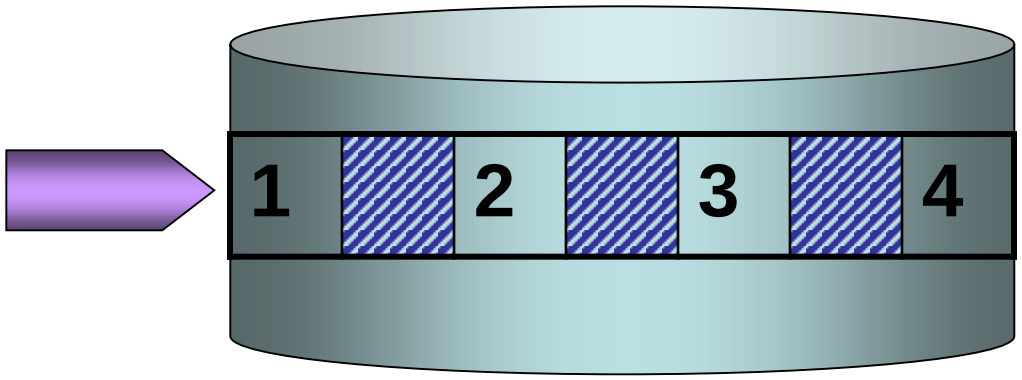
a) Rotational Ordering:

- **Under heavy I/O loads, several I/O requests may be wait for the same device.**

Ex:

Consider a drum like device, it has only four records. There are four I/O requests:

1. **Read record 4**
2. **Read record 3**
3. **Read record 2**
4. **Read record 1**



Speed=20ms per rotation

- **There are several ways that these I/O requests can be ordered to accomplish the same result.**

Ordering A:

Reading request order: 4, 3, 2, 1

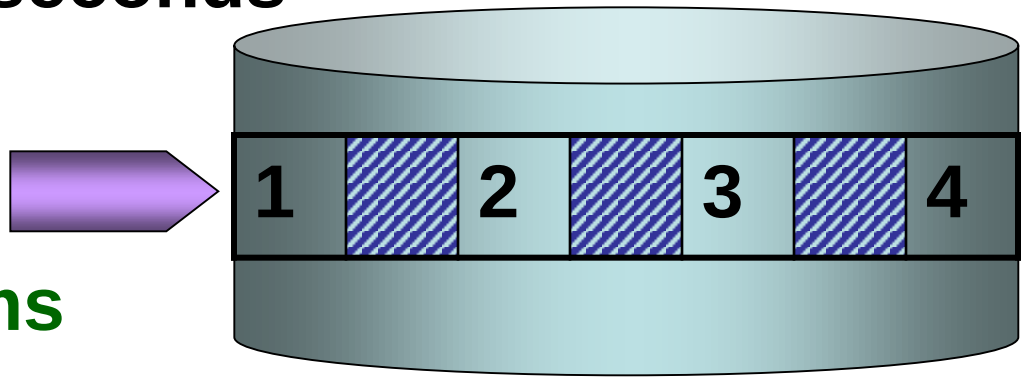
$T_{?4} + T_{43} + T_{32} + T_{21} \approx 3$ revolutions

($\frac{3}{4} + 3 \times \frac{3}{4} = \frac{3}{4} + 9/4 = 12/4 = 3$ rotations)

1 rotation = 20 milliseconds

½ rotation = 10 ms

¼ rotation = 5 ms

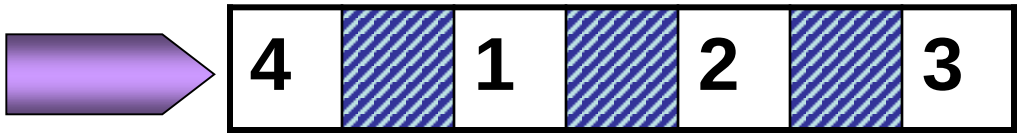


3 rotation X 20ms = 60ms

(or)

(4 X ¾) = 4 X 15ms(one record reading) = 60ms

Approximately it equals to 60ms, to read all four records. Since we do not know the current position.



Ordering B:

Reading request order: 1, 2, 3, 4

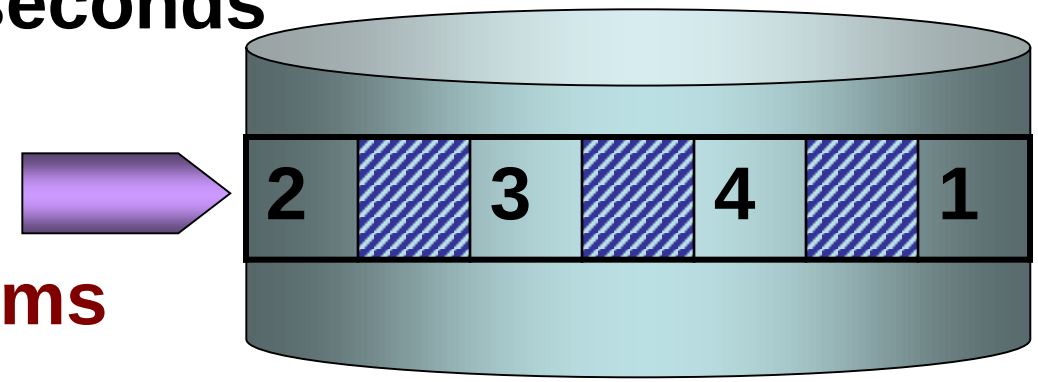
$T_{?1} + T_{12} + T_{23} + T_{34} \approx 1.5$ revolutions

($\frac{3}{4} + 3 \times \frac{1}{4} = \frac{3}{4} + \frac{3}{4} = 6/4 = 1.5$ rotations)

1 rotation = 20 milliseconds

½ rotation = 10 ms

¼ rotation = 5 ms



1.5 rotation X 20ms = 30ms

(or)

15ms + (3 X 5ms) = 30ms

Approximately it equals to 30ms, to read all four records.

The result of A and B are the same, there is a difference in speed.

Ordering C:

If we knew that "?" equaled record 3.

Current Read position is record 3.

Reading request order: 4, 1, 2, 3

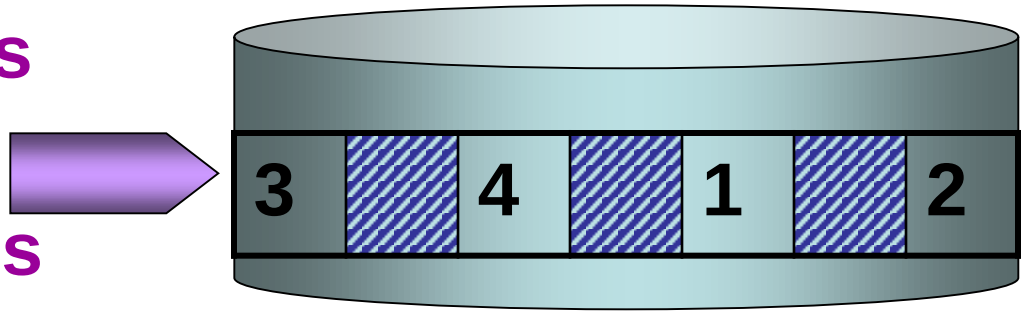
$T_{34} + T_{41} + T_{12} + T_{23} \approx$ 1 revolution

(1/4 + 1/4 + 1/4 + 1/4 = 4/4 = 1 rotation)

1 rotation X 20ms = 20ms

(or)

(4 X ¼) = 4 X 5ms = 20ms

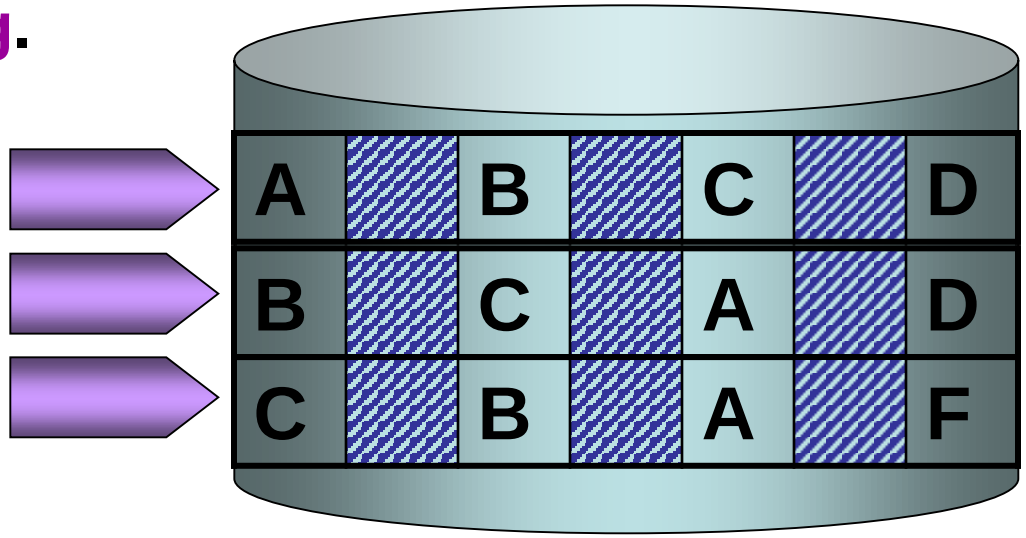


Approximately it equals to 20ms, to read all four records.

Device handler to know the current position for the rotating drum. This hardware facility is called Rotational position sensing.

b) Alternate Address:

- There are several alternate addresses for reading the same data record. This technique has also been called folding.



The Device has been folded N times

- If record A is stored on track1, record1, then it would take half revolution ie 10ms to access record.
- If a copy of record A is stored at both track1, record1 and track1, record3 then always accessing the "closest" copy, using rotational position sensing.
- Access time can be reduced to 5ms or 2.5ms or even 1.25ms by storing even more copies of the same.

c) Seek Ordering:

- Moving-head storage devices have the problem of seek position.

I/O request requires a three-part address:

cylinder number

track number

record number

If the read request;

cylinder1, track2, record1
cylinder40, track3, record3
cylinder5, track6, record5
cylinder1, track5, record7

➢Considerable time to spend move the seek arm back and forth.

A more effective ordering would be:

cylinder1, track2, record1
cylinder1, track5, record7
cylinder5, track6, record5
cylinder40, track3, record3

➢The distance between seeks have been minimized

4.7. VIRTUAL DEVICES

- Using Spooling a dedicated device like a card reader can be converted into many "virtual" card readers.
- This Technique is applicable to slow input/output devices like Card readers, punches and printers.

There are two serious problems that hamper effective device utilization in slow input/output devices. There are:

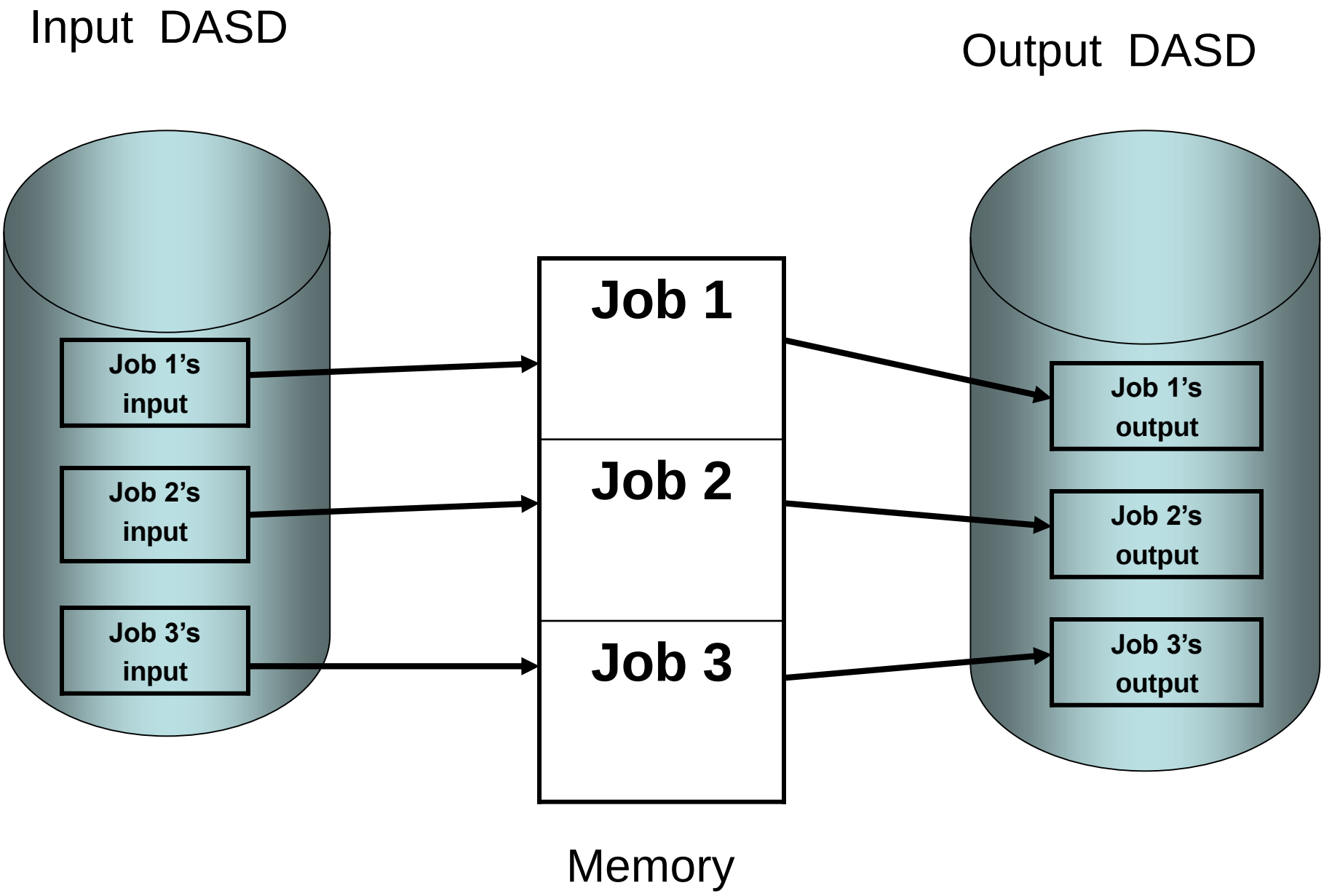
i. If a job attempts to generate requests faster than the device performance. The job must wait a significant amount of time.

ii. On the other hand, if a job generates requests at a much lower rate, the device is idle much of the time and is under utilized.

ii. These devices must be dedicated to a single job at a time. When these devices are allocated to jobs, only a fraction of the capacity is being made use, all other program generate irregular requests.

a) Historical solutions:

- These problem will disappear, if it is possible to use direct access storage devices for all input and output.
- As in the following diagram, a single DASD can be efficiently shared and simultaneously used for reading and / or writing data by many jobs
- DASDs provide very high performance rates and decrease a amount of wait time for jobs that require substantial amounts of input/output.


Input DASD
Output DASD
Job 1's input
Job 2's input
Job 3's input
Job 1
Job 2
Job 3
Job 1's output
Job 2's output
Job 3's output
Memory

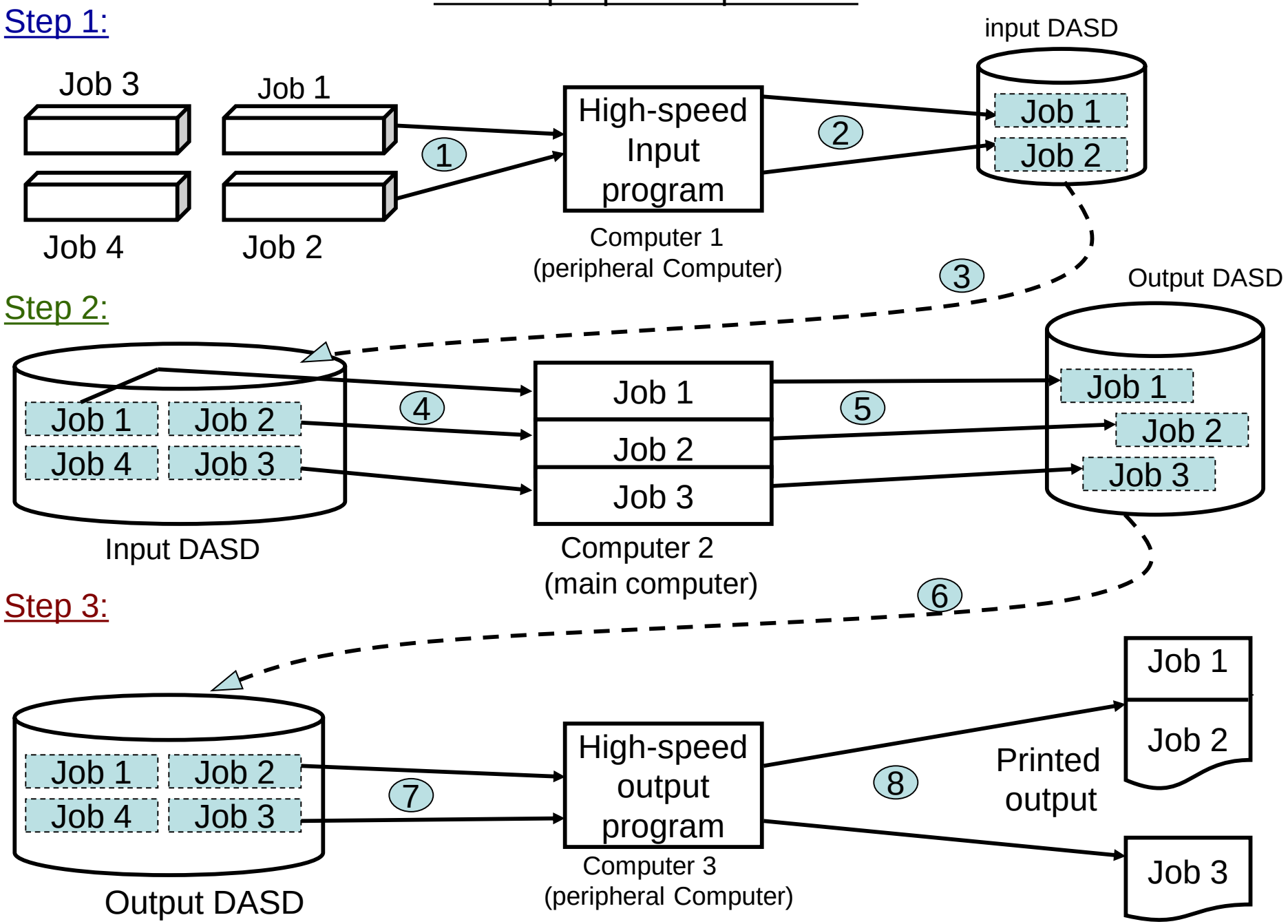
Disadvantages:

- The use of DASD's for all input and output is impractical because of the problems of putting the input data into the input DASD and getting out the data recorded on the output DASD.

i. Offline Peripheral Operations:

- A solution to the problem can be found in the three step process illustrated in the following diagram:

Off line peripheral operations



Step 1:

A Separate computer is used whose sole function is to read cards at maximum speed and record the corresponding information on a DASD. Two or more card readers can be used by computer 1, depending upon the amount of input.

Step 2:

The DASD containing the input recorded by computer 1 is moved over to the main processing computer (computer 2). Multiple jobs can be in execution, each reading its respective input from the input DASD and writing its output onto an output DASD.

Step 3:

The output DASD is moved to a third computer that reads the recorded output at high speed and prints the information in the printers.

- The work performed by the computer 1 and 3 was termed as offline peripheral processing and the computers were called peripheral computers.

- There are several observation that can be made at this time.
- The peripheral computers are required to perform only rather simple tasks, they can be quite simple, slow and inexpensive.
- If there is a relatively low peripheral load, one peripheral computer might handle it all-possibly switching from input processing to output processing every few hours.

- The offline peripheral processing techniques solved the problem presented earlier, but it also introduced several new problems in regards to

  1) Human Intervention
  2) Turn around
  3) Scheduling

- Since human operators were required to move the input peripheral computer to the main computer and to perform a similar task for processing, there were many opportunities for human errors and inefficiency.

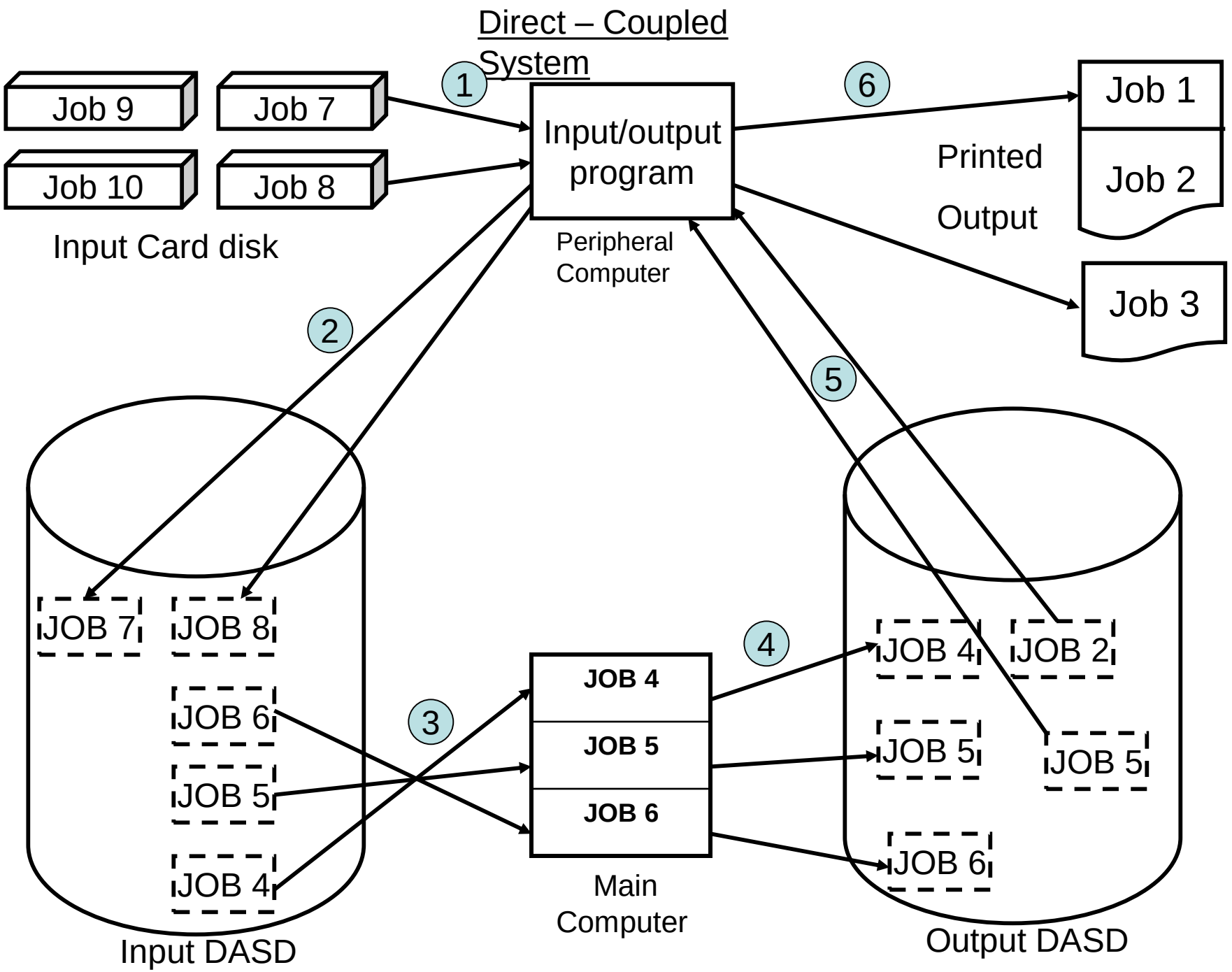
➢This batch processing approach made each job wait in line at each step and often increased its turnaround time. As a result of this batch processing it was difficult to provide the desired priority scheduling.

ii. Direct – Coupled Systems:

➢The following diagram shows a configuration in which the input and output DASDs are physically connected to both the peripheral computer and the main computer

➢ Thus eliminating the need for human handling which was the main drawback of the offline peripheral processing approach.

➢ This configuration is called a Direct-Coupled System (DCS)


Direct – Coupled System
Job 9
Job 7
Job 10
Job 8
Input Card disk
1
Input/output program
Peripheral Computer
6
Job 1
Job 2
Printed Output
Job 3
2
5
JOB 7
JOB 8
JOB 6
JOB 5
JOB 4
Input DASD
3
JOB 4
JOB 5
JOB 6
Main Computer
4
JOB 4
JOB 2
JOB 5
JOB 5
JOB 6
Output DASD

The direct-coupled system approach eliminates most of the problem of offline peripheral processing:

- No human intervention is required.

- There is no "batch processing" and turn around time delay.

- An actual direct-coupled system might use a single shared DASD for both input and output or several shared DASDs.

- It is important to coordinate carefully the use of a shared DASD; Mechanisms must be provide so that the main computer knows that a job has been placed on the input DASD and where it is located; similar request exist for output..

- Since both the peripheral computer and main computer wish to use the same DASD at the same time, there are frequent access conflicts that can reduce performance, there by making the DASD a critical bottleneck.

/

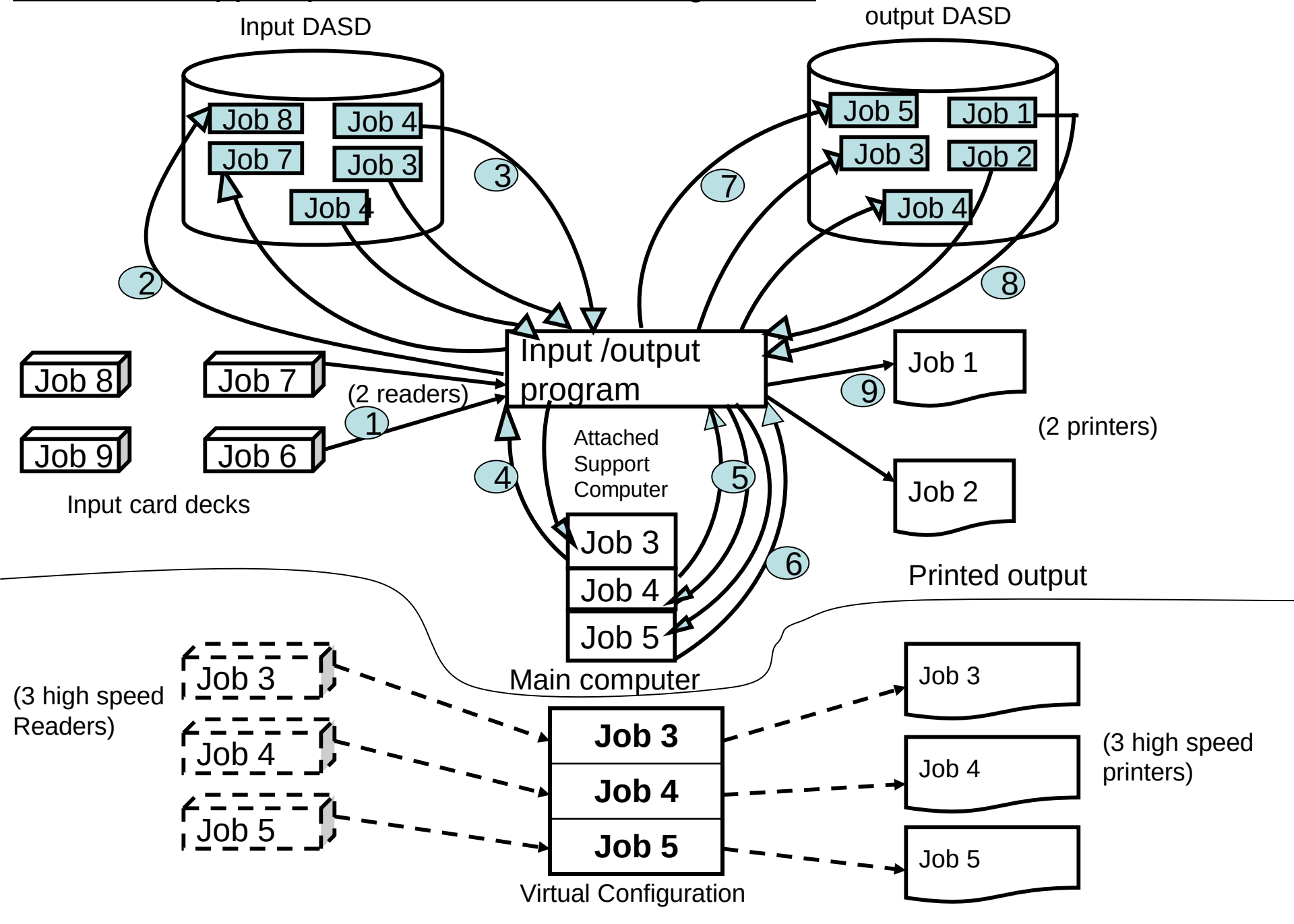
(iii) Attached Support Processor:

- Another variation of the direct coupled systems consists of directly connecting the peripheral and the main computers via a high speed connection as shown in the following figure.

- In this configuration, the peripheral computer is called an attached support processor (ASP).

- The support processor assumes all responsibility for control input/output peripherals as well as the input/output DASDs. It also performs all buffering and blocking. The attached processor produces the effect of virtual devices.

“Attached support processor “ Actual Configuration



➢ **Disadvantages of this ASP approach are:**

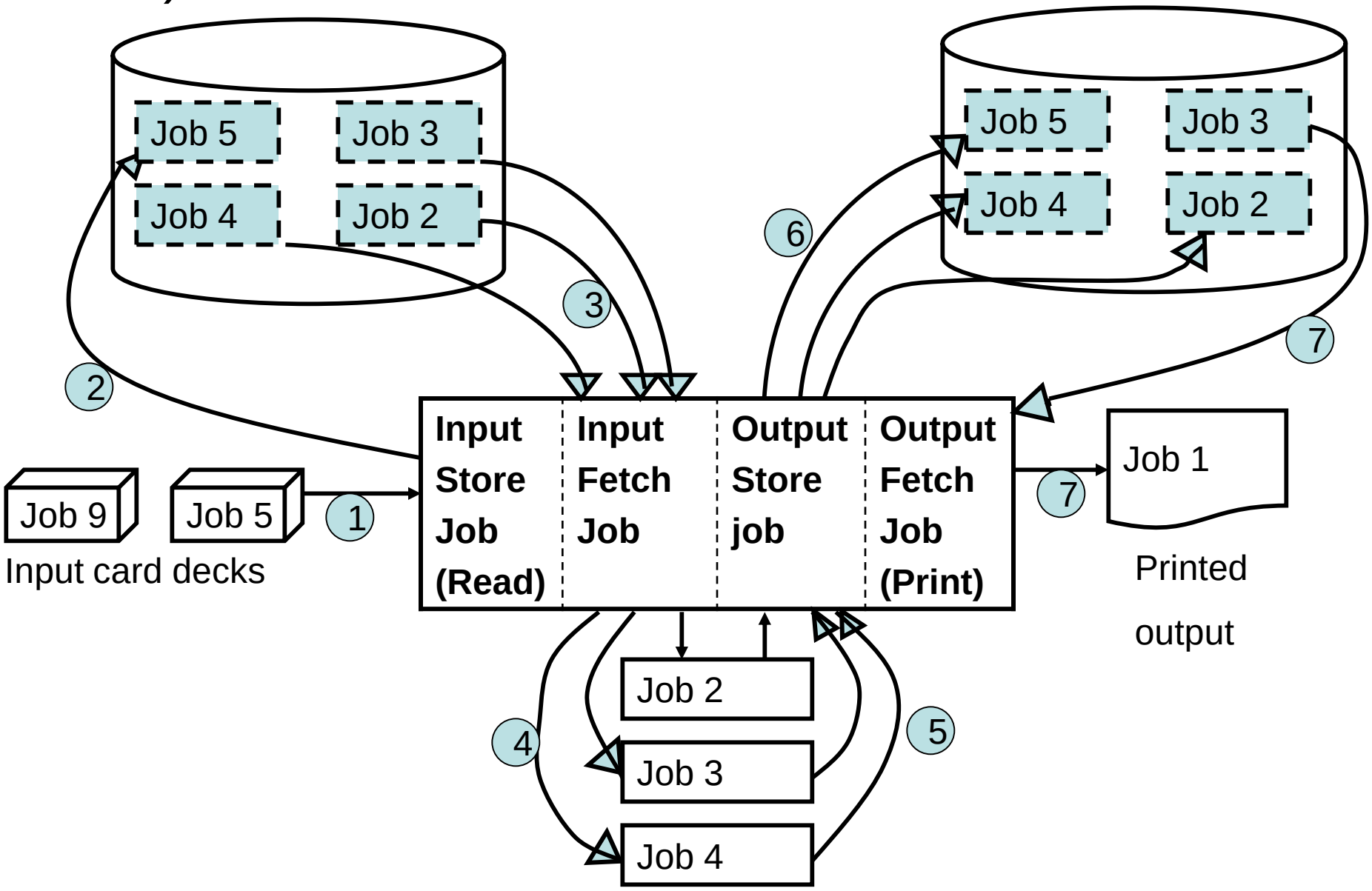
Two (or more) processors are required and certain processor are assigned specific tasks, it is possible at times one processor may be idle or under utilized while another is unable to handle the load.

(iv) Virtual System:

Considering the power processing facilities available in convectional i/o channels and multi programming capabilities, a separate computer for I/O is not necessary. A spooling system in which the main computer performs simultaneous peripheral operation On line (SPOOL) provides the better solution.

- Share the use of the main computer with normal jobs via multiprogramming.
- Since the jobs are actually system jobs rather than user jobs, they are given special names such as “phantoms” or “daemons”.
- The case of implementing a spool mechanism depends directly upon the specific facilities provided by the memory management, processor management, Information management and device management of the operating system.
- Spooling often improves the system so much.
- That is frequently provided as part of same simple operating systems that do not support multiprogramming.

SPOOLing system(Simultaneous Peripheral Operations OnLine)



(a)Design of a spooling System:

The spooling programs are assumed to be an integral part of an operating system and perform their own specialized information management. In such system, there are two special operating system functions provide.

They are:

(i) Read next input card - CALL READNEXT(BUFFER)

(ii) Print next output line –CALL PRINTNEXT(BUFFER)

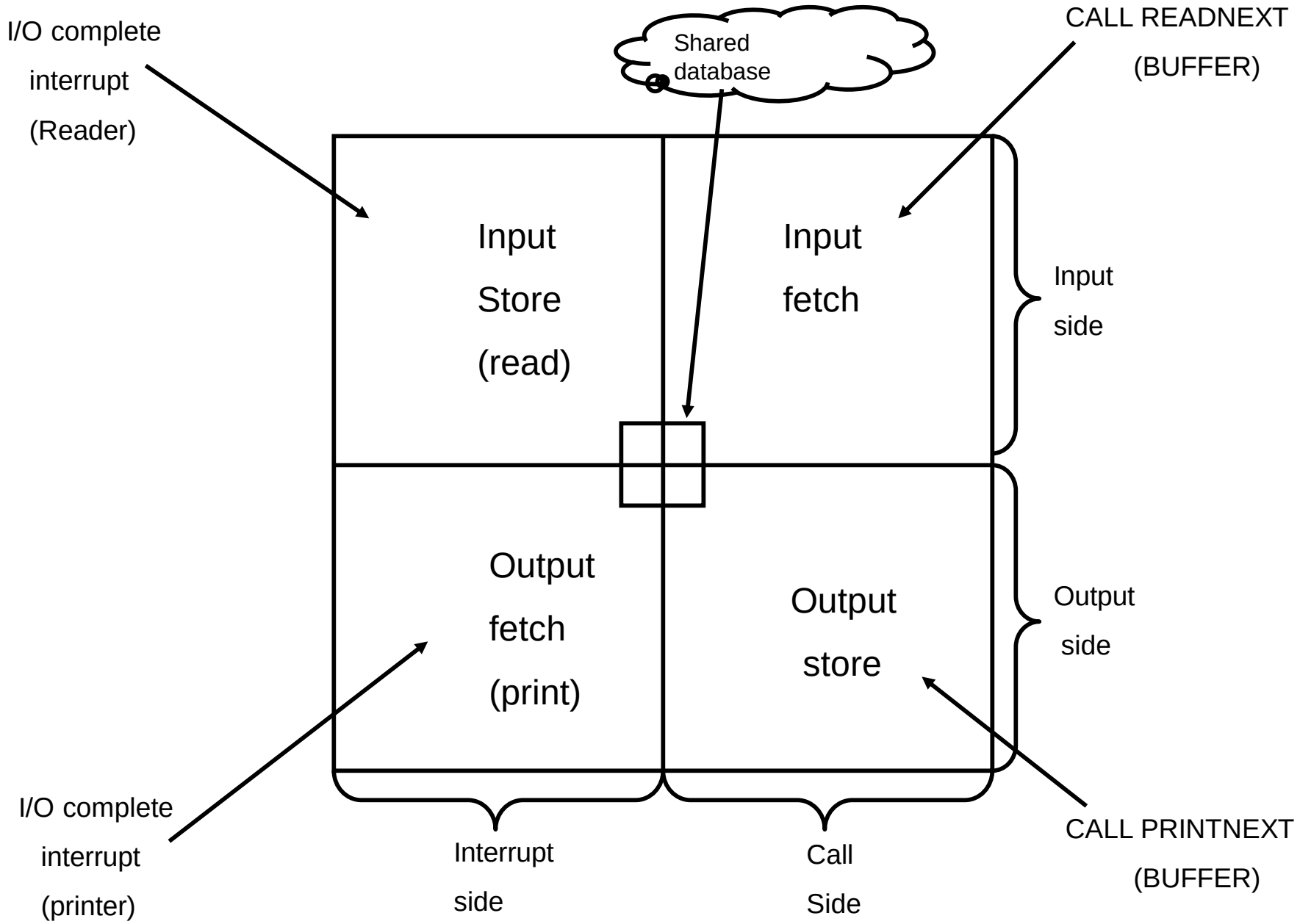
- Actually, the card is not physically read, nor the line printed at the time of these calls, since virtual device are being used.

- A general input and output spool system can be subdivided into four components as presented in the following diagram.

- These individual components can be grouped in several ways, on the basis of function (input or output) or by the method of gaining control (call or interrupt).

- The division of a program in to "call side" and an "interrupt side" is used in handling various devices and operating system functions.

General structure of the SPOOL system



i) Input SPOOL:

Two major operations here:

1. **To Read each input card and stored it on a DASD**
2. **To Provide access to the DASD copy of the next input card during execution.**

- **The reading of input card is done independently of the job, even before the job begins.**
- **The storing of input card physically read from the card reader is done as soon as the input is completed.**
- **This operation initiated in response to the I/O complete indication from card reader.**
- **This type of operation is called "interrupt driven".**

The following diagram show the Shared data base to coordinate the interrupt and the call side of the input SPOOL:

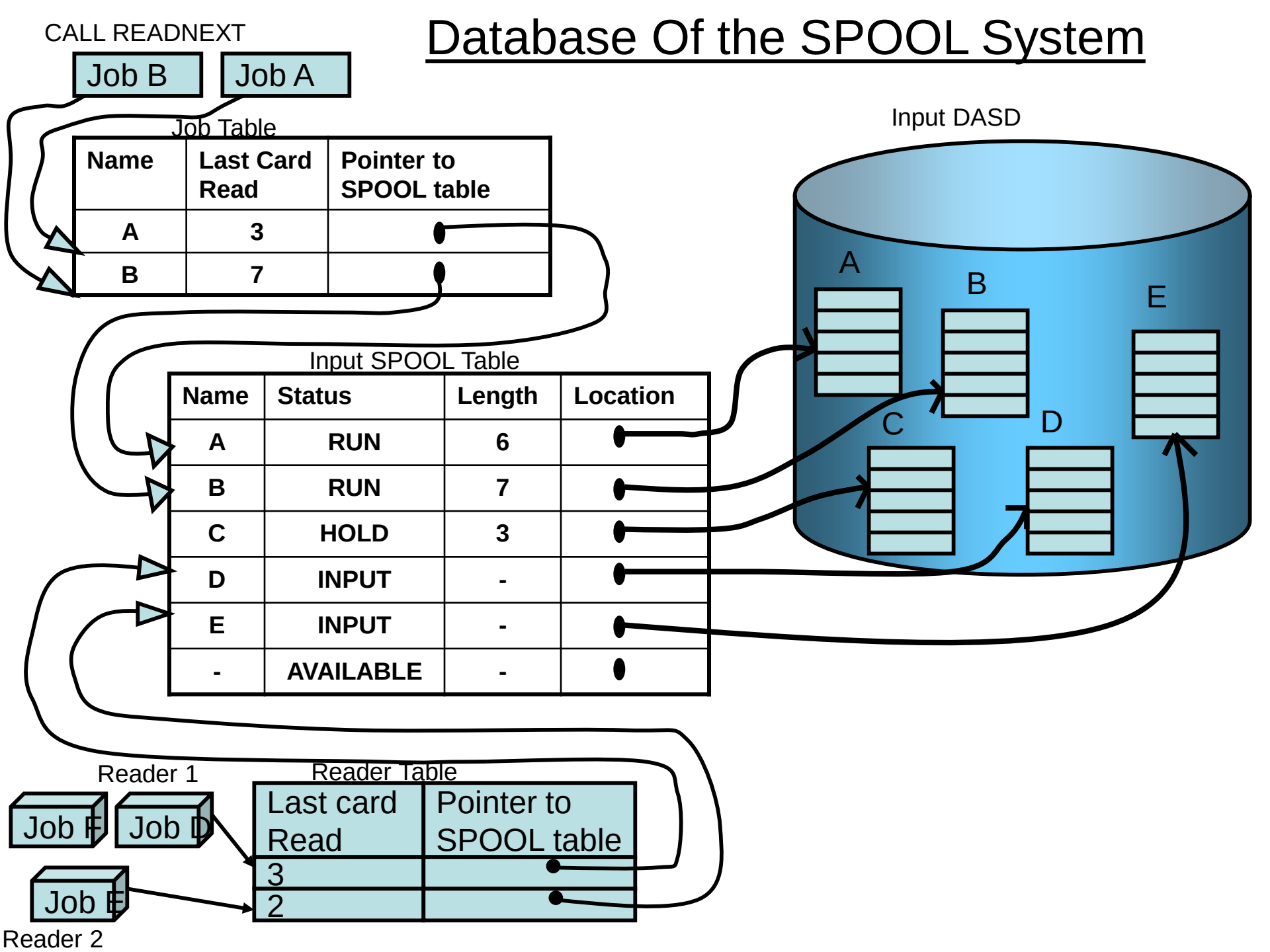
- **The DASD is divided into sections, which is holding an input deck**
- **The SPOOL table maintains the status of each input DASD storage area (SPOOL area).**
- **It may be hold INPUT, HOLD, RUN or AVAILABLE state.**

INPUT: The input card is still being read

HOLD: The input deck has been completed copied on the DASD but the Corresponding job has not been started yet.

RUN: The Corresponding job is currently running and reading the input data from the SOOL area

Database Of the SPOOL System



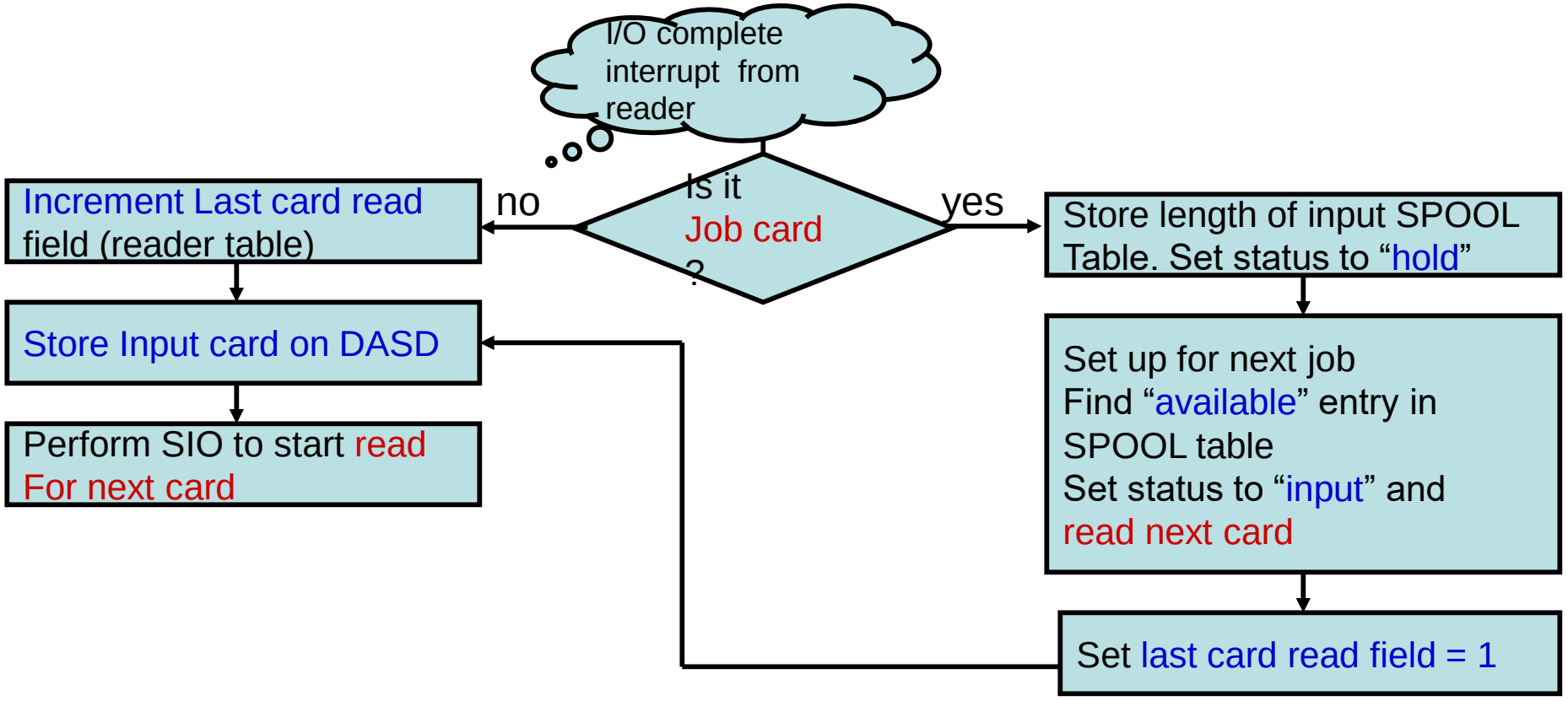
ii) Input SPOOL Algorithm:

There are two databases:

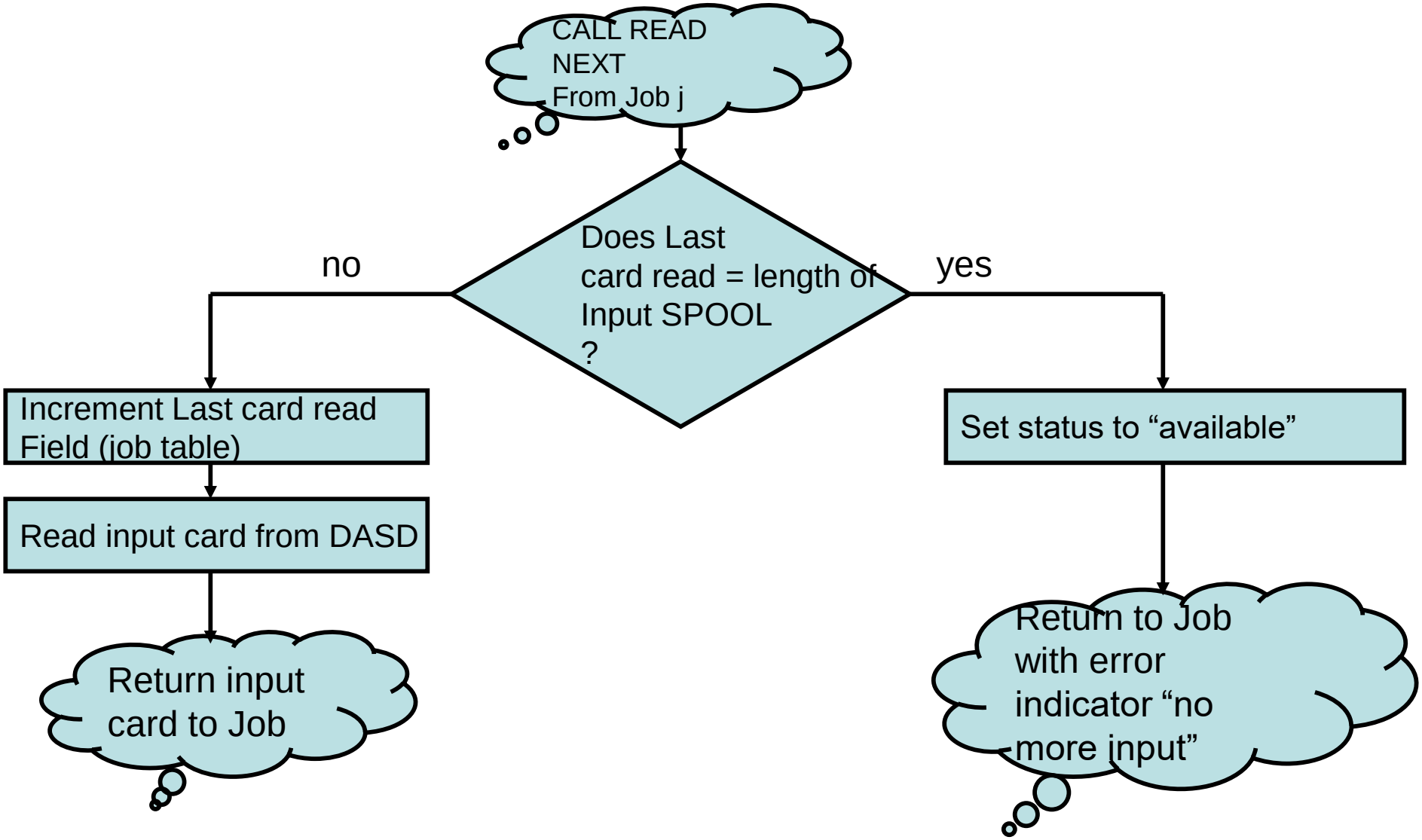
i) The Read Table – maintain status of each card reader

ii) The Job Table – maintain status of all jobs currently running

Interrupt side of input SPOOL program:



Call side of input SPOOL program:



UNIT - 5

Information Management

Information management is concerned with the storage and retrieval of information.

The basic functions of information management are

1. Keeping track of information in the system through various tables, the major one being the file directory-sometimes called Volume Table of Contents (VTOC). These tables contain the name, location and accessing rights of all information within the system.
2. Determining policy for determining where and how information is stored and who gets access to the information.
3. Allocation of information resource. The allocation module must find the desired information accessible to the process, and establish the appropriate access rights.
4. Deallocation of resources.

SIMPLE FILE SYSTEM

Let us consider the steps needed to process the following PL/I-like statement.

READ FILE (ETHEL) RECORD (4) INTO LOCATION (12000)

This is a request to read the fourth record of the file named ETHEL into a memory location 12000. Such a request invokes elements of the file system.

For the purpose of this example, let us assume that the file ETHEL is stored on a disk, as shown in Figure 5.2.

The file ETHEL is the shaded portion and consists of seven logical records. Each logical record is 500 bytes long.

The disk consists of 16 physical blocks of 1000 bytes each.

We can store two logical records of file ETHEL. into each physical block.

Instead of using the physical two-component (track number, record number) address, we will assign each physical block a unique number, as indicated in Figure below.

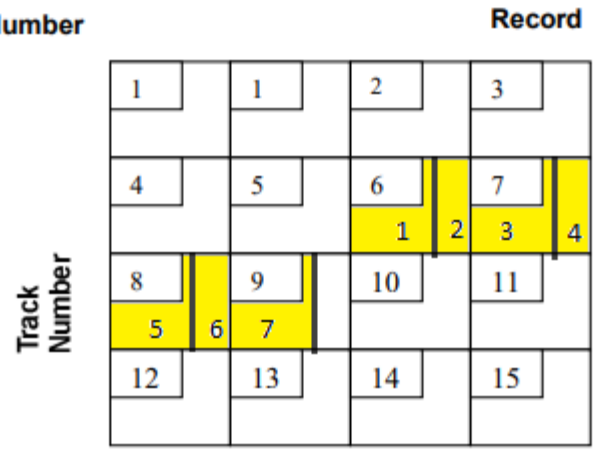


Figure 5.2 Physical File Storage

File Directory Database

Information on each existing file's status, for example, name and location, must be maintained in a table.

This table is often called a **file directory** or a **Volume Table of Contents**.

The name Volume Table of Contents comes from viewing the storage device medium as a "container of information" and thus a volume.

Figure 5-3 depicts a sample file directory. Each entry contains information necessary to find and access a file. The logical organization, i.e., programmer's view, is described in terms of the logical record length and the number of records.

The physical location of the file on the storage device can be determined by means of the starting block number information in conjunction with the logical record length and number of records. File ETHEL, for example, consists of seven SOD-byte records that require 3500 bytes of storage. Since the physical blocks are 1000 bytes long, four blocks are needed to hold files ETHEL. The first of these blocks, noted in the file directory, is 6. Thus, ETHEL occupies physical storage blocks 6, 7, 8, and 9, as previously depicted in Figure 5.2.

ETHEL is logically structured from the user's point of view as a sequential file, i.e., it consists of seven records numbered 1 through 7.

It is also physically stored as a sequential file, i.e.. its records are stored adjacent to one another in consecutive physical blocks.

However, the physical structure of ETHEL need not be sequential. The blocks

of ETHEL could be scattered all over secondary storage, and a mapping between the logical records and the physical blocks might then become quite complicated.

ENTRY NUMBER	NAME	LOGICAL RECORD SIZE	NUMBER OF LOGICAL RECORD	ADDRESS OF FIRST PHYSICAL BLOCK	PROTECTION AND ACCESS CONTROL
1	MARYLIN	80	10	2	Access by everyone
2	Free	1000	3	3	(Free)
3	ETHEL	500	7	6	Read only
4	JOHN	100	30	12	Read for MANDICK Read/Write for DONOVAN
5	Free	1000	2	10	(Free)
6	Free	1000	1	15	(Free)

Figure 5.3 Sample File Directories

There are six entries listed in the file directory of Figure 5.3.

Three of these (MARILYN, ETHEL and JOHN) corresponds to real user files.

The other three entries are needed merely to account for presently unused storage space.

Steps to Be Performed

The following figure presents the steps that must be performed to satisfy a request, such as:

READ FILE (ETHEL) RECORD (4) INTO LOCATION (12000)

STEP NUMBER	ACTION
Step 1	The file directory is searched to find the entry for ETHEL (entry number 3).
Step 2	Information about ETHEL is retrieved and extracted from the file directory entry, namely: logical record size, address of first physical block, and protection and access control information.
Step 3	Based on the protection information, a decision is made whether to allow the requesting process to access ETHEL.
Step 4	The Logical Record Address specified (record 4 of ETHEL) is transformed into a Logical Byte Address. Logical Byte Address = (record number −1) X (record size) which is (4−1) X (500) = 1500.
Step 5	The Logical Byte Address is then transformed into a Physical Byte Address. Logical Byte Address 1500 corresponds to byte 500 within the second 1000-byte physical block of ETHEL, which corresponds to physical block 7.
Step 6	Physical block 7 is read into a 1000-byte buffer area in main memory. This is accomplished by an I/O program similar to those described in Chapters 2 and 5.
Step 7	Record 4 is extracted from the buffer area and moved into the user's area at location 12000. That is, locations 500 through 999 of BUFFER are moved to locations 12000 through 12499 in main memory.

Fig. 5.4

GENERAL MODEL OF A FILE SYSTEM

In this section a general model of a file system is introduced. Most of key components of a file system correspond to generalizations of specific steps of Figure 5.4.

The components in this file system are organized as a structured hierarchy. Each level in the hierarchy depends only on levels below it. This concept improves the possibility of systematic debugging procedures. For example, once the lowest level is debugged, changes at higher levels have no effect. Debugging can proceed from lowest to highest levels.

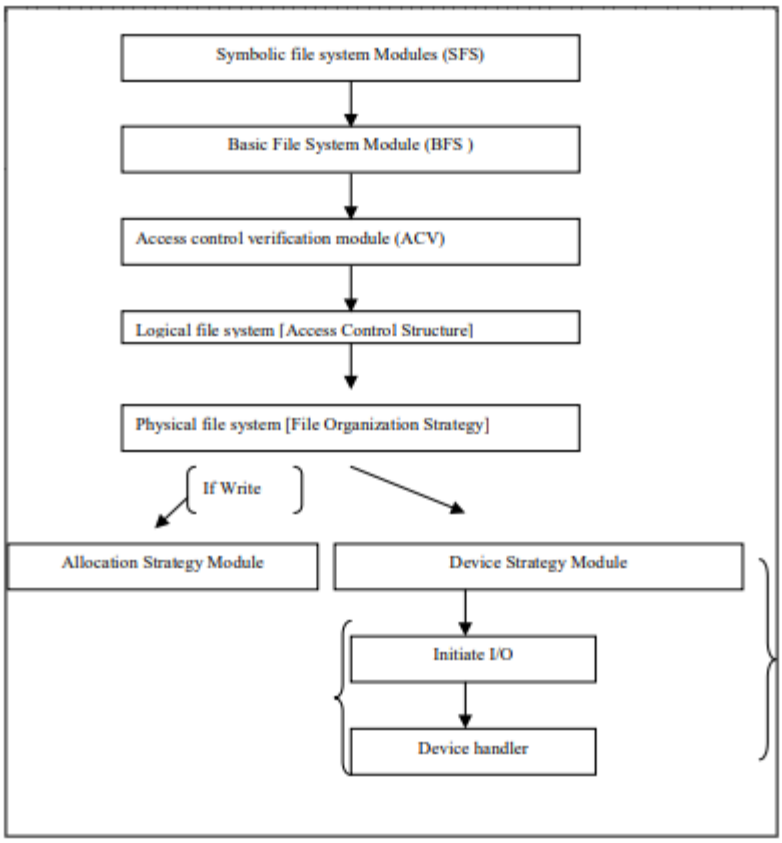


FIGURE 5.4 GENERAL MODEL OF A FILE SYSTEM

Although the particular details presented in this chapter (e.g., format of file directory, file maps) may vary significantly from system to system, the basic structure is common to most contemporary me systems. For example, depending on the structure of a specific file system, certain of the modules in Figure 5.5 may be merged together, further subdivided, or even eliminated-but the underlying structure should still be the same.

File Directory Maintenance

Before discussing the functions of the file system components, it is important to answer certain questions about the file directory, such as: How are the file directory entries created and filled in? Where is the file directory stored? Is it necessary to search the entire directory for every request?

In a basic system such as IBM's Basic Operating System/360 (OS/360), the programmer must keep track of available storage space and maintain the me directory by control cards similar to:

CREATE ETHEL, RECSIZE=500, NUMRECS=7, LOCATlON = 6 DELETE JOHN

The CREATE command adds a new entry to the file directory and the DELETE command removes an old one.

If the entire file directory is kept in main memory all the time, a significant amount of memory may be needed to store it. A more general approach is to treat the file directory as a file and place it on the storage volume. Furthermore, if the me directory is stored on the volume, the files may be easily transferred to another system by physically transferring the volume (tape reel, disk pack, etc.) containing these files as well as the corresponding me directory.

If the file directory is stored on the volume, then it may take a substantial amount of time to search it for each access to files. Although the directory may be quite large, possibly containing thousands of entries, usually only a few files are ever used at one time. Thus if we copy the entries for files that are currently in use into main memory, the subsequent search times can be substantially reduced. Many file systems have two special requests, OPEN, to copy a specific file directory entry into main memory, and CLOSE, to indicate the entry is no longer needed in main memory. Likewise, we talk about a file being open or closed depending upon the current location of its file directory entry.

Symbolic File System

The first processing step is called the Symbolic file System (SFS). A typical call would be :

CALL SFS (function, file name, record number, location), such as: CALL SFS (READ, "ETHEL", 4, 12000)

The Symbolic File System uses the file name to locate that file's unique file directory entry. This function corresponds to step 1 in Figure 5.4.

In our simple system we had assumed a one-to-one correspondence between me names and files. In more advanced file systems, the facility must exist to have several files with the same name and be able to call the same me by different names. To implement such facilities, we divide the file directory of Figure 5-3 into two separate directories, a Symbolic File Directory (SFD) and a Basic File Directory (BFD) as shown in Figure 5-5. The symbolic file system must search the symbolic file directory to determine a unique identifier (10) for the file requested (see Figure

5-5). This ID is used instead of the file's name to locate entries in the Basic File Directory. The Basic File System allows operations on files using the 10 such as :

CALL BFS (READ, 5, 4, 12000)

Where 5 is ETHEL's 10.

Since the SFDs are usually kept on the storage device, entries for files that are currently in use (called active or opened files) are copied into main memory. These copies can be used to eliminate repeated 1/0 access to the same entry on the storage device. We will call this table of active me entries the Active Name Table (ANT).

	NAME	UNIQUE IDENTIFIER (ID)
1	MARILYN	3
2	ETHEL	5
3	JOHN	6

16 bytes — 4 bytes

20 bytes per entry

(a) Sample Symbolic File Directory

ID	LOGICAL RECORD SIZE	NUMBER OF LOGICAL RECORDS	ADDRESS OF FIRST PHYSICAL BLOCK	PROTECTION AND ACCESS CONTROL
1	–	–	0	(Basic File Directory)
2	20	3	1	(Symbolic File Directory)
3	80	10	2	Access by everyone
4	1000	3	3	(Free)
5	500	7	6	Read only
6	100	30	12	Read for MADNICK Read/write for DONOVAN
7	1000	2	10	(Free)
8	1000	1	15	(Free)

(b) Sample Basic File Directory

Figure 6-6 Sample Symbolic and Basic File Directory

In summary the Symbolic File System is responsible for :

1. The Symbolic File Directory
2. The Active Name Table
3. Communication with the Basic File System

Basic File System

The second processing step is called the Basic File System (BFS). A typical call would be :

CALL BFS (function, file 10, record number, location), such as : CALL BFS (READ, 5, 4, 12000)

The Basic File System uses the file ID to locate that file's entry in the Basic File Directory and copies the entry into main memory. This function corresponds to step 2 in Figure 5.4.

```
DECLARE     1     SFD_ENTRY
                      2 NAME CHARACTER (16)
                      2 ID        FIXED BINARY (31)
DO     I = 1 TO 3
         CALL BFS (READ, 2,1,SFD_ENTRY)
         IF SFD_ENTRY.NAME = DESIRED_NAME
         THEN GOTO FOUND
          END;
FOUND: DESIRED_ID = SFD_ENTRY.ID;
```

By permanently assigning the Symbolic File Directory a specific 10, the SFS can use the facilities of the BFS to search the directory for the desired name and 10. For example, the PL/I program segment Verification (ACV), which processes calls such as

CALL ACV (READ, 2,4, 1200)

Where AFT entry 2 contains the information for me 10 5 (me ETHEL).

In summary, the Basic File System is responsible for:

1. The Basic File Directory
2. The Active File Table
3. Communication with the Access Control Verification module

For the simple file system of Section 5-2, it is quite easy to combine the SFS and BFS functions. In later sections we will introduce additional functionality where this logical separation is much more significant.

Access Control Verification

The third processing step is called Access Control Verification (ACV). A typical call would be

CALL ACV (function, AFT entry, record number, location) such as CALL ACV (READ, 2,4, 12000)

The Access Control Verification acts as check-point between the Basic File System and the Logical File System. It compares the desired function (e.g., READ, WRITE) with the allowed accesses indicated in the AFT entry. If the access is not allowed, an error condition exists and the file sys-tem request is aborted. If the access is allowed, control passes directly to the Logical File System. This function corresponds to step 3 in Figure 5.4.

Logical File System

The fourth processing step is called the Logical File System (LFS). A typical call would be

CALL LFS (function. AFT entry, record number, location) such as CALL LFS (READ, 2, 4, 12000)

The Logical File System converts the request for a logical record into a request for a logical byte string. That is, a file is treated as a sequential byte string without any explicit record format by the Physical File System. In this simple case of fixed length records, the necessary conversion can be accomplished by using the record length information from the AFT entry. That is, the

Logical Byte Address = (Record Number - I) X Record Length

and

Logical Byte Length = Record Length

This function corresponds to step 4 in Figure 5.4.

Note that by permanently assigning the BFD file a specific ID (such as ID I in Figure 5-5b) and a specific AFT entry number (such as AFT entry I), the Basic File System can call upon the Logical File System to read and write entries in the Basic

File Directory. (It is necessary to have a special procedure for fetching BFD entry 1 into AFT entry 1 when the system is initially started or restarted.)

In summary, the Logical File System is responsible for: CALL PFS (READ, 2, 1500, 500, 12000)

Note that by permanently assigning the BFD file a specific ID (such as ID 1 in Figure 5-5b) and a specific AFT entry number (such as AFT entry 1), the Basic File System can call upon the Logical File System to read and write entries in the Basic File Directory. (It is necessary to have a special procedure for fetching BFD entry 1 into AFT entry I when the system is initially started or restarted.)

In summary, the Logical File System is responsible for :

1. Conversion of record request to byte string request
2. Communication with the Physical File System

Examples of more complex logical record organizations, such as variable-length records, are presented in later sections.

Physical File System

The fifth processing step is called the Physical File System (PFS). A typical call would be

CALL PFS (function, AFT entry, byte address, byte length, location) such as

CALL PFS (READ, 2, 1500, 500, 12000)

The Physical File System uses the byte address plus the AFT entry to determine the physical block that contains the desired byte string. This block is read into an assigned buffer in main memory, using the facilities of the Device Strategy Module, and the byte string is extracted and moved to the user's buffer area. This function corresponds to steps 5 and 7 in Figure 5.4.

The mapping from Logical Byte Address to Physical Block Number, for the simple contiguous organization of Figure 5-2, can be accomplished by:

Physical Block Number = Logical Byte Address / Physical Block Size + address of first physical block

For the request for the byte string starting at Logical Byte Address 1500, the Physical Block Number is :

Physical Block Number = 1500 / 1000 + 6 = I + 6 = 7

which is the block containing record 4 of me ETHEL (see Figure 5-2).

The location of the byte string within the physical block can be determined by:

Block Offset = remainder [Logical Block Address / Physical Block Size]

such as :

Physical Block Offset = remainder [1500 / 1000] = 500.

Thus, the byte string starts at the offset 500 within the physical block, as expected from Figure 5.2.

In order to perform these calculations, the Physical File System must know the mapping functions and physical block size used for each storage device. If these were the same for all devices, the information could be built into the PFS routines. Usually there are variations depending upon the device type (e.g., a large disk may be handled differently from a small drum). Thus, this information is usually kept on the storage volume itself and read into the Physical Organization Table (POT) when the system is first started.

If a WRITE request were being handled and the corresponding physical block had not been assigned, the Physical File System would call upon the Allocation Strategy Module (ASM) for the address of a free block of secondary storage.

In summary, the Physical File System is responsible for:

1. Conversion of logical byte string request into Physical Block Number and Offset
2. The Physical Organization Table
3. Communication with the Allocation Strategy Module and the Device Strategy Module

Later sections will introduce many more physical system organizations and performance considerations that have an impact on the physical file system.

Allocation Strategy Module

The Allocation Strategy Module (ASM) is responsible for keeping track of unused blocks on each storage device. A typical call would be:

CALL ASM (POT entry, number of blocks, first block) such as:

CALL ASM (6, 4, FIRSTBLOCK)

where POT entry 6 corresponds to the storage device on which file ETHEL is

to reside and FIRSTBLOCK is a variable in which the address of the first of the four blocks requested is returned.

Figure 5-5 indicates how the location of groups of available blocks can be recorded in the Basic File Directory by treating them as special files. Other techniques for maintaining this information are presented in subsequent sections.

The Device Strategy Module (DSM) converts the Physical Block Number to the address format needed by the device (e.g., physical block 7 = track I, record 3, as shown in Figure 5-2). It sets up appropriate I/O commands for that device type. This function corresponds to step 6 in Figure 5.4. All previous modules have been device-independent. This one is device-dependent. Control then passes to the I/O scheduler.

I/O Scheduler and Device Handler

These modules correspond to the device management routines that were discussed in Chapter 3. They schedule and perform the reading of the physical block containing the requested information.

Calls and Returns between File System Modules

After the physical I/O is completed, control is returned to the Device Strategy Module. The DSM checks for correct completion and returns control to the PFS, which extracts the desired information from the buffer and places it into the desired location. A "success" code is then returned back through all other modules of the file system.

Why isn't an immediate return made to the uppermost level? There are two reasons: (I) An error may be detected at any level, and that level must return the appropriate error code (for example, the Logical File System may detect the attempted access of an address beyond the file); and (2) any level may generate several calls to lower levels (for example, the Symbolic File System may call the Basic File System several times to read entries from the Symbolic File Directory).

The Symbolic File System checks to see whether or not a file exists. The Access Control Verification checks' to see whether or not access is permitted. The Logical File System checks to see whether the requested logical address is within the file. The Device Strategy Module checks to see whether or not such a device exists.

Shortcomings of Simple File System Design

Let us summarize some of the assumptions, inefficiencies, and inadequacies of our simple file-system system :

1. **Symbolic File System**

   We assumed that each file had a single unique name.

2. **Access Control Verification**

   How are files shared? How can the same file have different protection rights for different uses?

3. **Logical File System**

   We assumed that each file consisted of fixed-length records, and that these records were viewed as sequential. How are accesses to variable-length records handled? Are there other file structuring methods? Are there more flexible methods for the user?

4. **Physical File System**

   Are there physical structuring other than sequential? Are these more efficient in space? In accesses? We assumed that for each request this module would activate an access to the disk. What is the request following READ ETHEL RECORD (4) were READ ETHEL RECORD (3)? Isn't record 3 in the system buffer area already?

5. **Allocation Strategy**

   How are files stored on secondary storage? Are there fragmentation problems when files are deleted? In other words, is secondary storage being used efficiently?

6. **Device Strategy Module**

   Can this module restructure or reorder requests to match more efficiently the characteristics of the device on which the information is stored?

In the sections that follow we will explore more general techniques for each of these levels, techniques that give the user more flexibility and may thus increase the system's efficiency.

Logical File System (LFS):

LFS is mapping the structure of the logical records onto linear byte-string of a PFS(Physical File System) file. It must covert a request for a record into a request for a byte string.

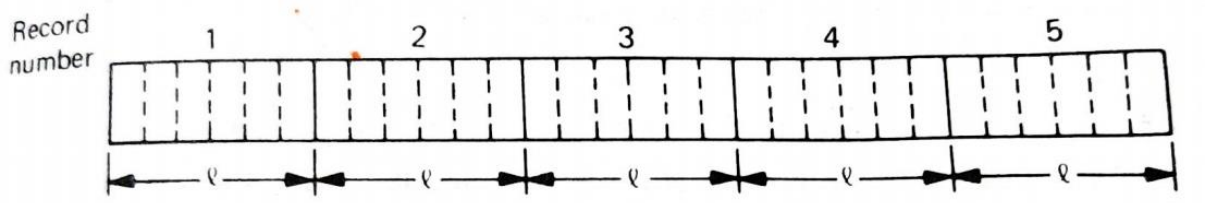
In the sample file system of section LFS provides facilities for direct access, based on record number to any of the records, all of which are of same length.

In conventional file systems and data management systems many additional record structures are supported. These are often called access methods, such as:

1. Sequentially Structured fixed-length records (sequential and direct access)
2. Sequentially Structured variable-length records (sequential and direct access)
3. Sequentially Structured keyed records
4. Multiple keyed records
5. Chained Structured Records
6. Relational or triple structured records

1. Sequentially Structured fixed-length records:

User views his files as a sequence of fixed length records(i.e, all records are the same length). This structure is useful for storing 'card-image', 'print-image' and 'SPOOL' files.



Sequential fixed-length records (all records l bytes length)

i. Sequential Access:

For sequential access, on each request the user wishes to have the "next" record the same as reading cards from a card reader. This request would look similar to

READ FILE(ETHEL) NEXT INTO LOCATION(BUFFER)

where the explicit record number is omitted.

In order to implement this, LFS must maintain Current Logical Byte Address (CLBA) in the AFT entry for the file. When the file is initially opened, the CLBA is set to 0. Subsequently, after processing each request, the CLBA is updated by

CLBA = CLBA + RL (Record Length)

Where RL is the Record Length.

ii. Direct Access:

For direct access the user explicitly species the record desired.

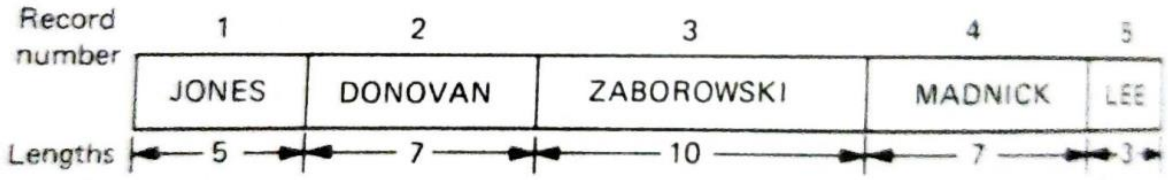
READ FILE (ETHEL) RECORD (4) INTO LOCATION (BUFFER)

CLBA computed by

CLBA = (RN-1) x RL where RN is designated record number.

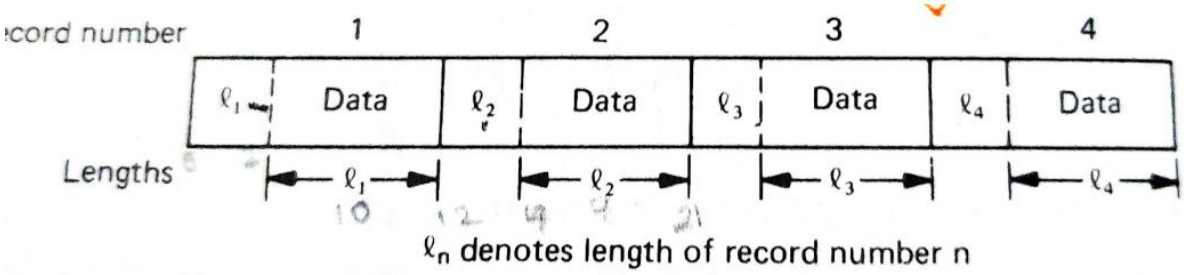
2. Sequentially structured variable-length records:

User views his file as a sequence of records of possibly differing lengths. This structure is useful for storing names, employment histories and printer output.



Sample of Sequentially structured variable-length records

There are various ways of storing these records. Note that it is necessary to be able to identify the boundaries of each record. For example, how do we know that there are two separate records, JONES and DONOVAN instead of one record JONESDONOVAN? The following format can be used to handle this situation. In this approach the length of each record is stored in the file.



Sequentially structured variable-length records

i. Sequential Access:

Its look like

READ FILE (NAMES) NEXT INTO LOCATION (BUFFER) LENGTH (N)

Where the variable N is set to the length of the record that was read. The length of the BUFFER area must be equal to or larger than the largest record in the file. Let us assume that the length field 1n is 2bytes long, so that records can be up to 64K bytes long. If the CLBA is always set to the beginning of the length field for the next record (e.g., it is set to 0 initially), the length N, can be extracted from the 2byte string at location CLBA. The record itself can then be read as the N-bytes string at location CLBA+2. Since we can determine the byte address and length for both steps, the PFS can be used to process these requests. The CLBA is updated in preparation for the next request by

CLBA = CLBA + 2 + N

ii. Direct Access:

A Direct access request would look like

READ FILE (NAMES) RECORD (3) INTO LOCATION (BUFFER) LENGTH (N)

The logical byte address for a direct access to a sequentially structured variable length record is found essentially by sequencing through records until you find the right record. That is, if record 3 is requested.

CLBA = l_1+2+l_2+2

Where $l_1$and l_2 are the length of record l and record 2, and can be found by accessing these records sequentially.

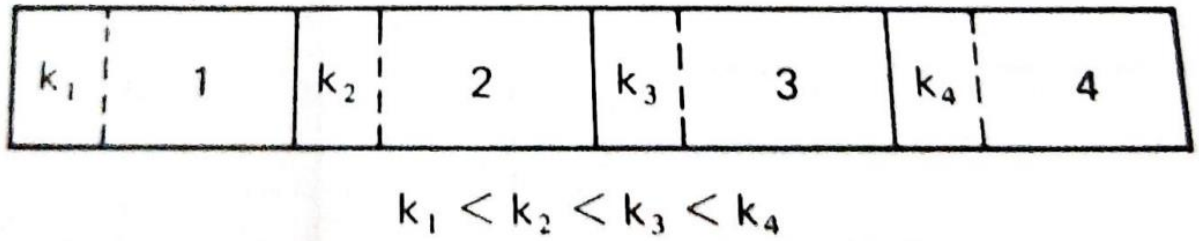
Direct access can be very inefficient, but these are several methods for improving the efficiency, such as the following:

- Keep track of the number and address of the last record requested. If the present request is a record number larger than the last one accessed, then go forward. If less, then start sequencing from the beginning of the file.
- Keep a table of record numbers and LBAs.
- Each time a record is accessed its LBA could be kept by the user's program for possible future use(e.g., OS/360-NOTE/POINT facility). This would make possible very fast subsequent accesses.

3. Sequentially structured keyed records:

A different type of access is based on content rather than record number or address. An example would be a request for the record containing Madnick's payroll information. Such single keyed records may be arranged where $k_1,\ldots k_n$ denote keys,

e.g., k_4 = MADNICK



Sequential single-keyed records

All records are kept in ascending order based on the keys, i.e, k_1, k_2, k_3, etc, for example, MADNICK would follow DONOVAN but precede ZIERING.

The Logical Byte Address for either sequential or direct access may be computed by searching all records for the one with the correct key. In order to avoid such massive search, a separate symbol table or index table file may be used to indicate correspondence between keys and the LBA of the record. In practical system the Index Table may be divided into levels.

In practical systems the Index Table may be divided into levels. The following figure depicts the structure used by IBM's Indexed Sequential Access Method (ISAM). The master index points to the secondary index. The secondary index contains the starting locations of subsets of the data items. Every n^{th} record in the data file is left empty for inserts. A request requires one access to the master index, one access to the secondary index, and a sequential search of the records in the subset.

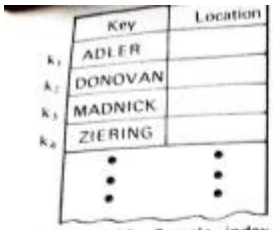
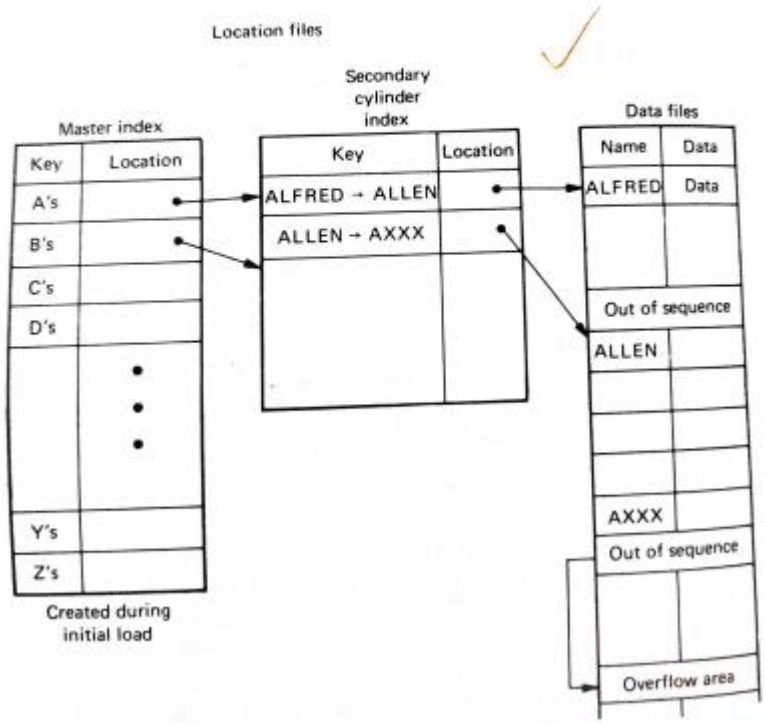


Figure 6-14 Sample index table for Figure 6-13



An ISAM structuring

Multiple keyed records:

The following four in a record –(ID = 1743, NAME-DONOVAN, SS#=030-34-2674, JOB=FASTEST PROFESSOR AT MIT) all identify the same person. Sometimes users wish to organize their data so that they can be referenced by any one of several keys. This is similar to library indexing of books by title, author, and subject. The major techniques for this are:

- Multiple Indices (note that the Index tables may exceed the size of the data!)
- Chaining records with the same keys together

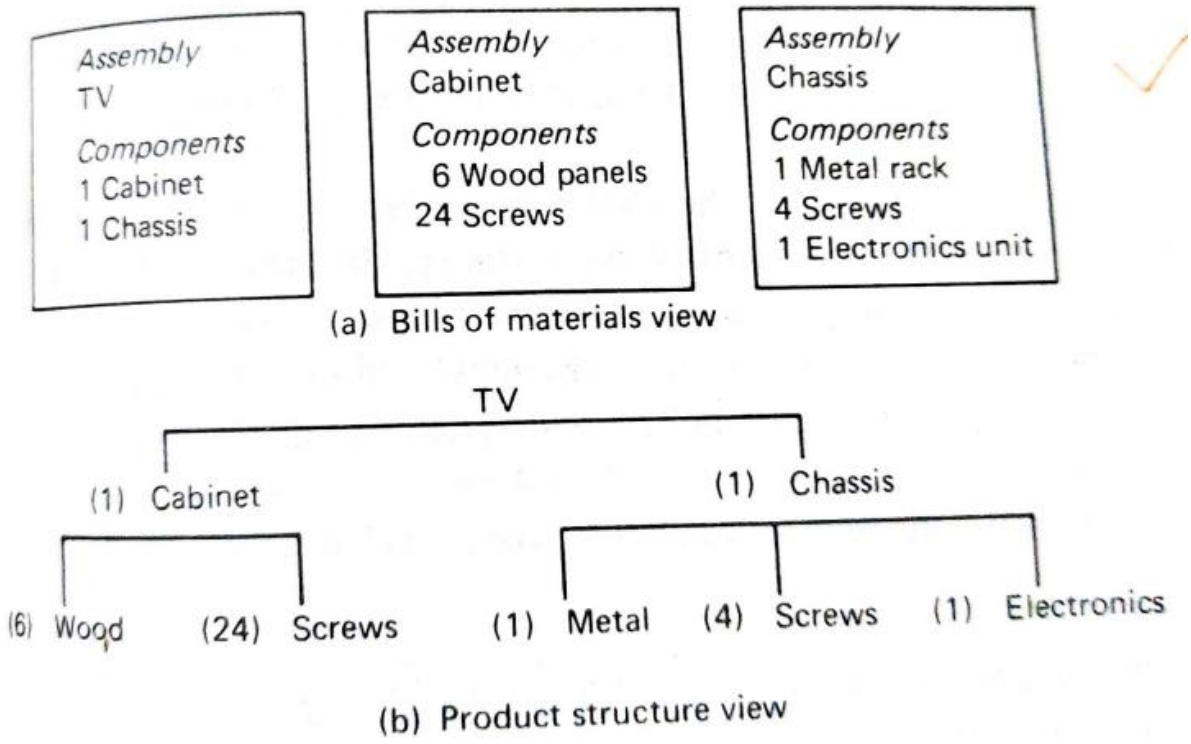
4. Chained Structured Records:

The keyed and multiple keyed record structures are not usually offered as part of a basic file system, but rather are built on top of one. They are data management systems, and provide facilities for complex record structures. A chained structured record organization provides a more complex data management facility.

Eg.

An example can be found in Figure a, which shows the "bill of materials ie. list of components) for each assembly used to produce a television set. Figure

b presents the same information in a "tree" or "chained" form.



Sample of Chained Structured Records

5. Relational or triple structured records:

Another database organization that may be built on top of a basic file system is a relational or triple structure (Codd, 1970). Logical records may bear a relation to other records, e.g., Frank is John's father. Consider the following table.

RECORD 1	RELATION	RECORD 2
FRANK	FATHER	JOHN
JOHN	BROTHER	PAUL
JOHN	FATHER	JAMES

Questions that a user may ask of such a system are:

- Who is JAMES FATHER (i.e., ?-FATHER-JAMES)
- Name all the FATHERS (i.e., ?FATHER-?)

Physical File System:

The primary function of the PFS is to perform the mapping of Logical Byte Addresses into Physical Block Addresses. The Logical File System calls the Physical File System by passing the logical byte address and length of the information requested.

The PFS may first call the Allocation Strategy Module (ASM)(if a write request) and then the Device Strategy Module (DSM) directly.

The PFS keeps track of the mapping from logical byte Address to the blocks of physical storage.

(physical block address = first physical block of file + Logical byte address / block size) simple mapping function was a consequence of the simple physical storage structure of the file.

Three additional considerations may be

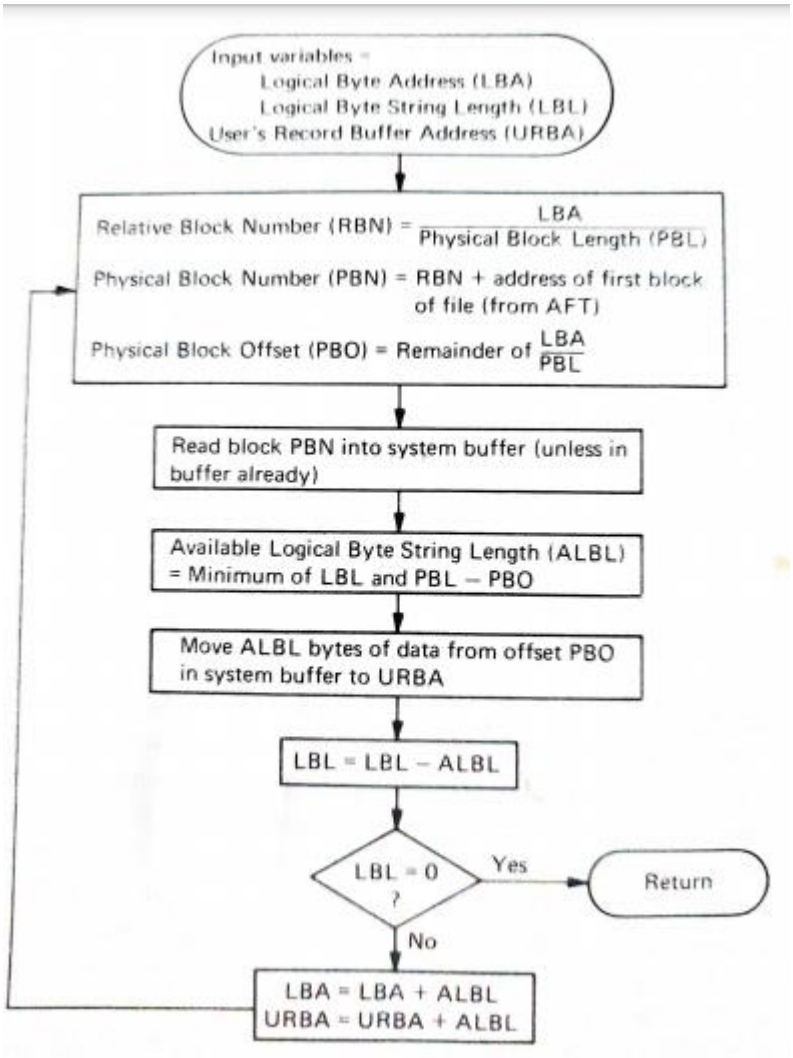
1. Minimizing I/O Operations.
2. Allowing Logical record size independent of Physical Block Size
3. Allowing noncontiguous allocation of file space.

Minimizing I/O Operations:

If all the records of file ETHEL were to be read sequentially, it would require seven I/O operations. The entire file occupies only four physical blocks, we might suspect that we could reduce the number of I/O operations of eliminating redundant operations. For each I/O read operation a physical block is copied into a buffer in main memory. Before reading a physical, check whether the block is already in buffer.

Allowing Logical record size independent of physical Block Size:

A physical record must hold some integral number of logical records, to process logical records of lengths that are not even factor of block length. Many storage devices allow a certain amount of flexibility in the format of the individual tracks.



Revised version of steps 5 and 7

Allowing noncontiguous allocation of file space:

Contiguous allocation of file space is not practically possible always. If the physical blocks of a file are not contiguous, a different algorithm is needed to map from logical byte address to physical block number. Two popular techniques are

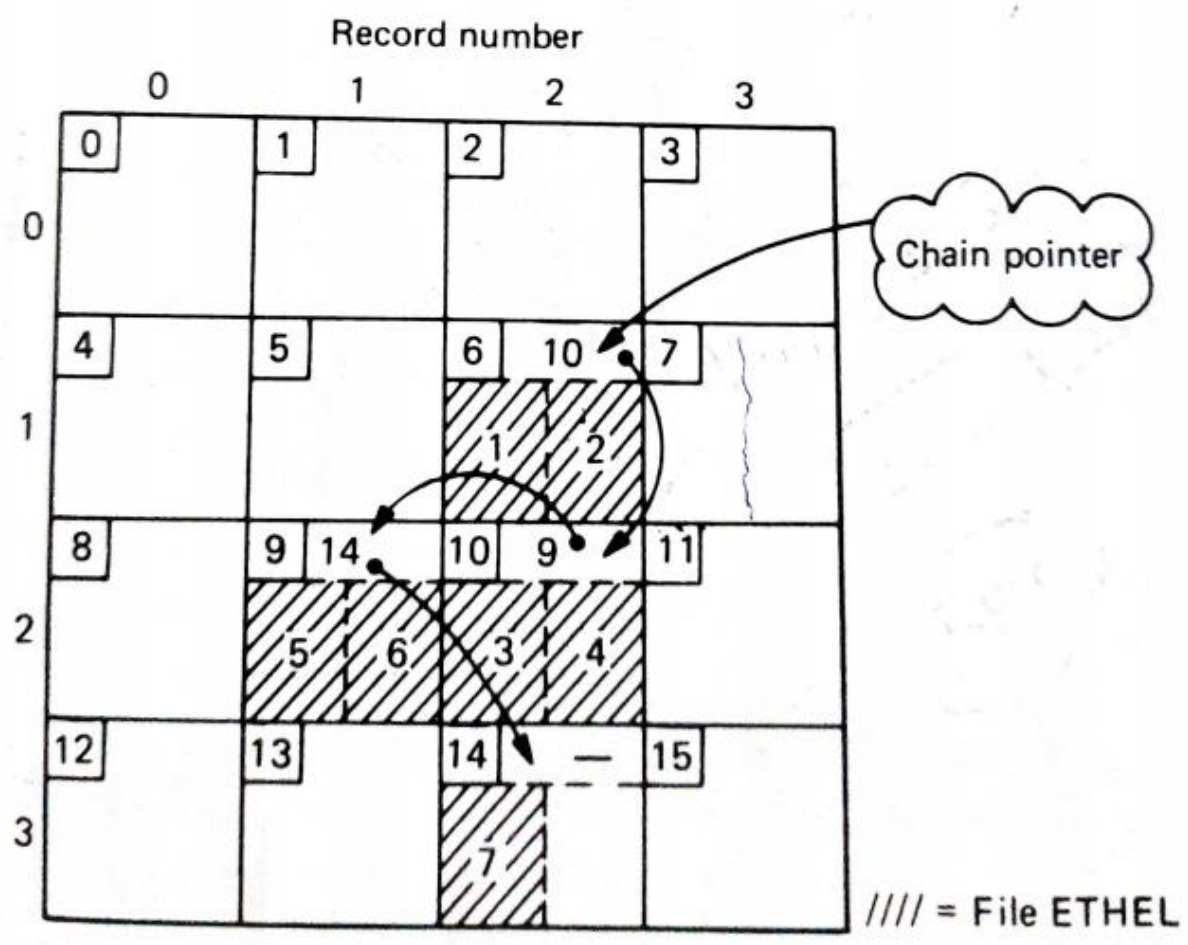
(1) Chained blocks

(2) File maps

Chained Blocks:

In a chained blocks mapping, each physical block contains the address of the next "logically contiguous" block. The Basic File Directory entries contain the address of the file physical blocks , but subsequent block addresses are found by means of the address pointer.

The address pointer may be stored within the physical block, the chained blocks approach is efficient for sequential access to the blocks.



Chained Blocks

File Maps:

Another approach to noncontiguous allocation is to use a File Map Table to map each Logical Block Address to its physical block address. This file map may be stored as part of the entry in the BFD or in a separate block.

Physical Block Address = File Map (Logical Byte Address / Block Size)

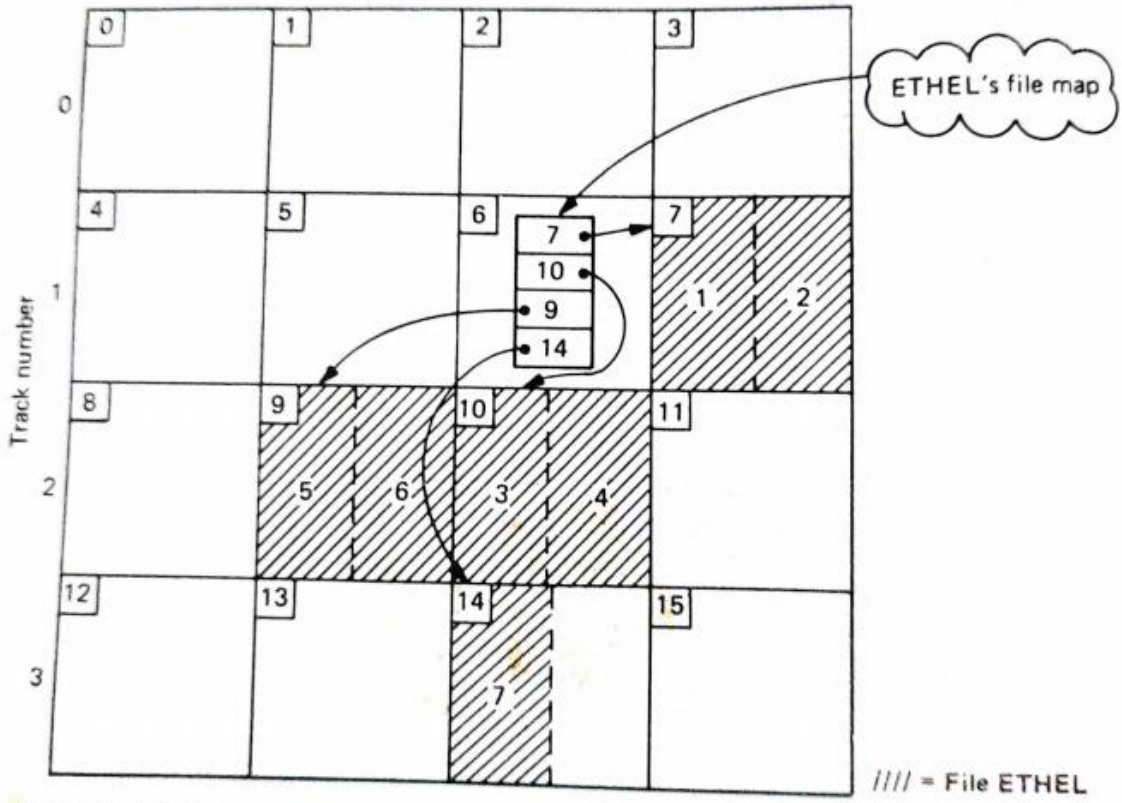


Figure 6-20 File map

File Map