Module 4

Memory and Storage Management

Syllabus

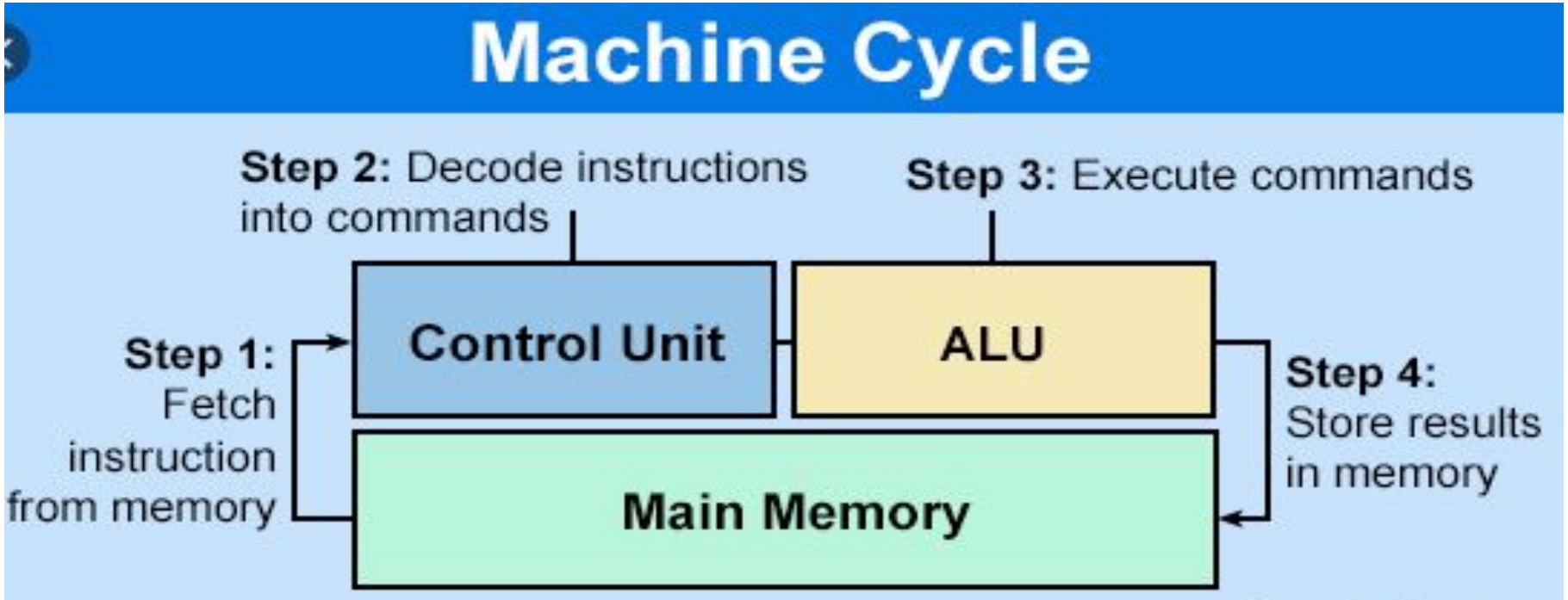
Memory and Storage Management:

- Logical and physical Address Space, Swapping, Contiguous Memory Allocation, Paging, Page table structure, Segmentation with Paging.
- Virtual Memory Management: Demand paging, Process creation, Copy-on-write, Page Replacement Algorithms, Demand segmentation.
- File-System Interface: File concept, Access Methods, Directory structure, File-system Mounting, File sharing

Memory Management

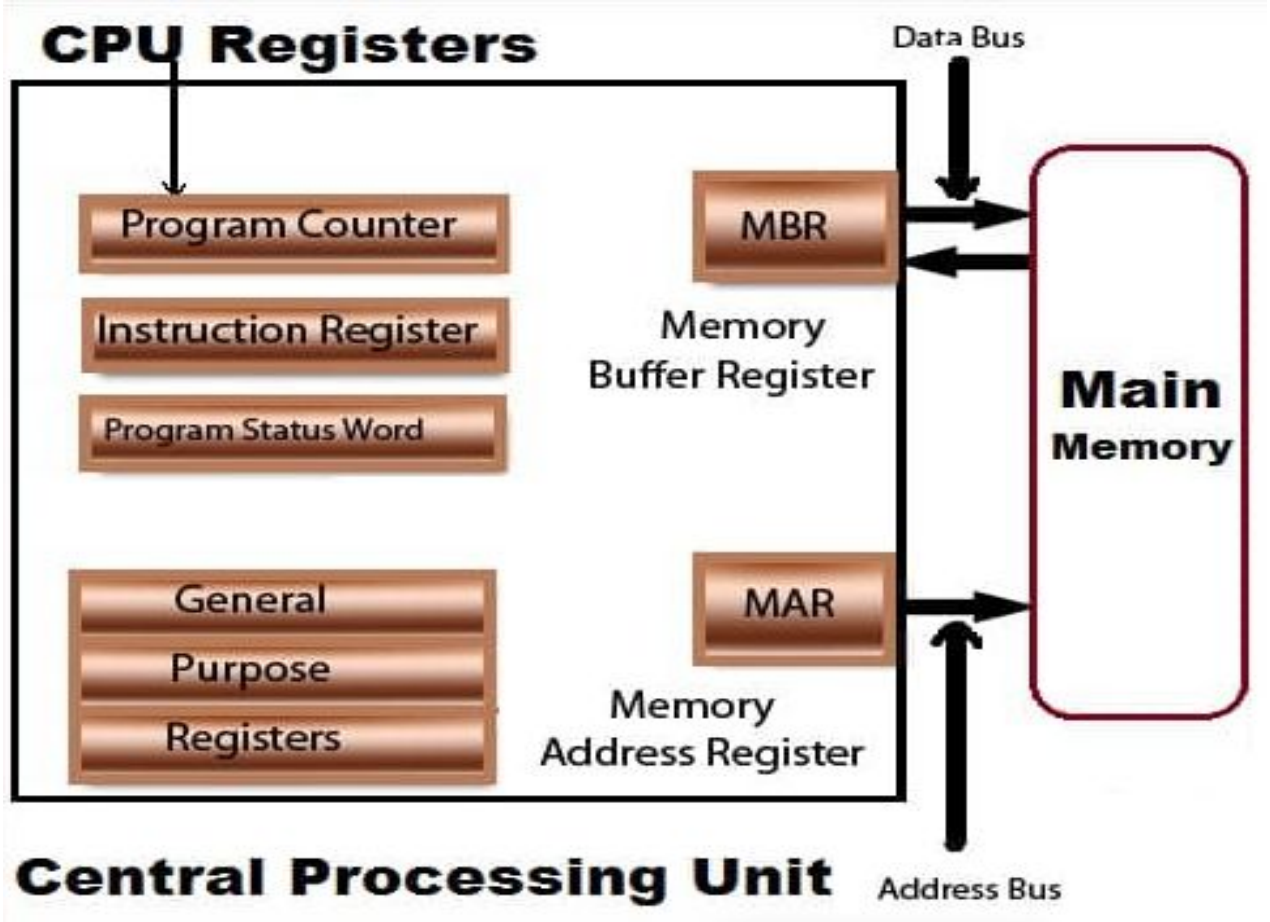
Background :

- Registers that reside in the CPU are easier to access and accessible within one CPU cycle.
- Performing simple instructions and performing basic operations on the contents of the register can be done using the CPUs.
- Memory access to other drivers consumes more than one CPU cycles via the memory bus.
- In certain cases, where the CPU takes many clock cycles to complete, the processor is stalled or made to pause, since the data required is yet to be retrieved.
- This can be avoided by adding **cache** between the main memory and CPU.


Machine Cycle
Step 2: Decode instructions into commands
Step 3: Execute commands
Control Unit
ALU
Step 1: Fetch instruction from memory
Step 4: Store results in memory
Main Memory

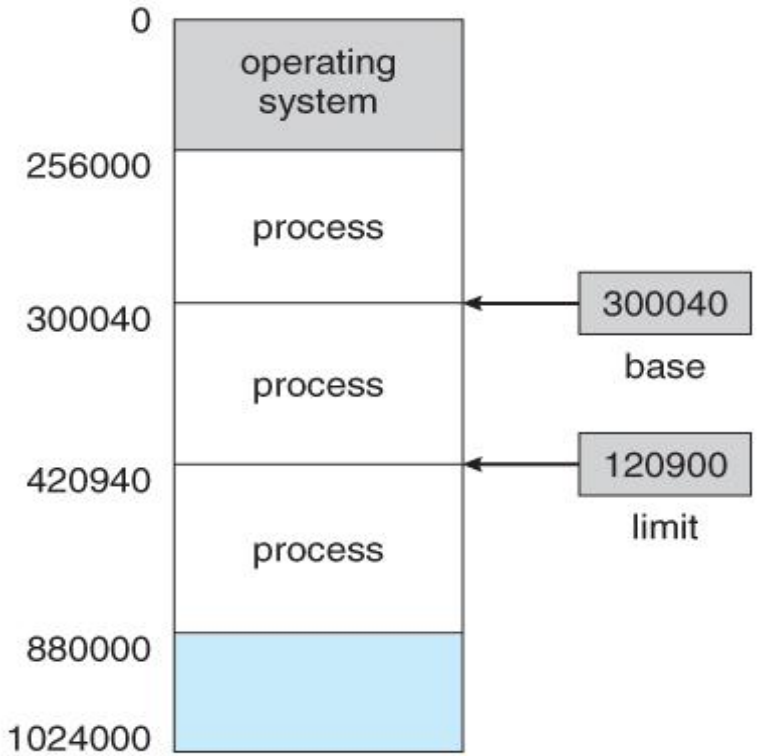
```
[root@linux-server root]#
[root@linux-server root]# objdump -D a.out | grep -A20 main.:
08048460 <main>:
 8048460:       55                      push   %ebp
 8048461:       89 e5                   mov    %esp,%ebp
 8048463:       83 ec 08                sub    $0x8,%esp
 8048466:       90                      nop
 8048467:       c7 45 fc 00 00 00 00    movl   $0x0,0xfffffffc(%ebp)
 804846e:       89 f6                   mov    %esi,%esi
 8048470:       83 7d fc 09             cmpl   $0x9,0xfffffffc(%ebp)
 8048474:       7e 02                   jle    8048478 <main+0x18>
 8048476:       eb 18                   jmp    8048490 <main+0x30>
 8048478:       83 ec 0c                sub    $0xc,%esp
 804847b:       68 08 85 04 08          push   $0x8048508
 8048480:       e8 93 fe ff ff          call   8048318 <_init+0x38>
 8048485:       83 c4 10                add    $0x10,%esp
 8048488:       8d 45 fc                lea    0xfffffffc(%ebp),%eax
 804848b:       ff 00                   incl   (%eax)
 804848d:       eb e1                   jmp    8048470 <main+0x10>
 804848f:       90                      nop
 8048490:       b8 00 00 00 00          mov    $0x0,%eax
 8048495:       c9                      leave
 8048496:       c3                      ret
[root@linux-server root]# _
```

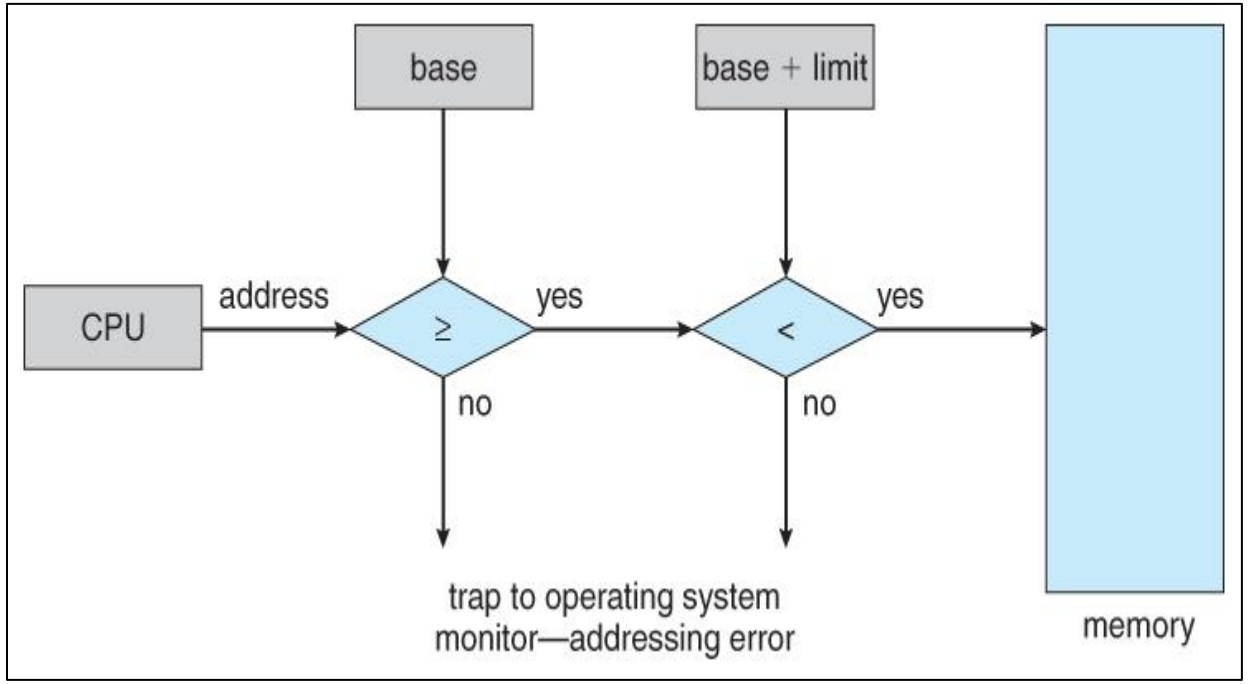
Memory Management



Logical Address Space

- Logical address spaces are defined with the help of base and limit registers which are stored by the operating system.
- Both the ends of the base and limit registers are also included in the address space.
- To protect hardware addresses, the limits are checked each time an attempt to access the memory is made.





Hardware address protection with base and limit registers

A base and a limit register define a logical address space

Mapping of Logical and Physical Address

- After a process is chosen from the input queue, the data and instructions pertaining to the process are loaded into memory and space becomes available once the process terminates.
- An **address binding** is a method of mapping the **symbolic addresses** i.e. the source program, to the **re-locatable addr**esses.
- The **loader** or editor turns the **re-locatable addresses to absolute addresses.**

Mapping of Logical and Physical Address

Symbolic address-> Re-locatable address -> absolute address

Compile time:

- If the memory location of the process is known during compile time, the **absolute code** can be generated and the compiler code starts from the location and extends.
- Recompilation is necessary if the location of the code changes.

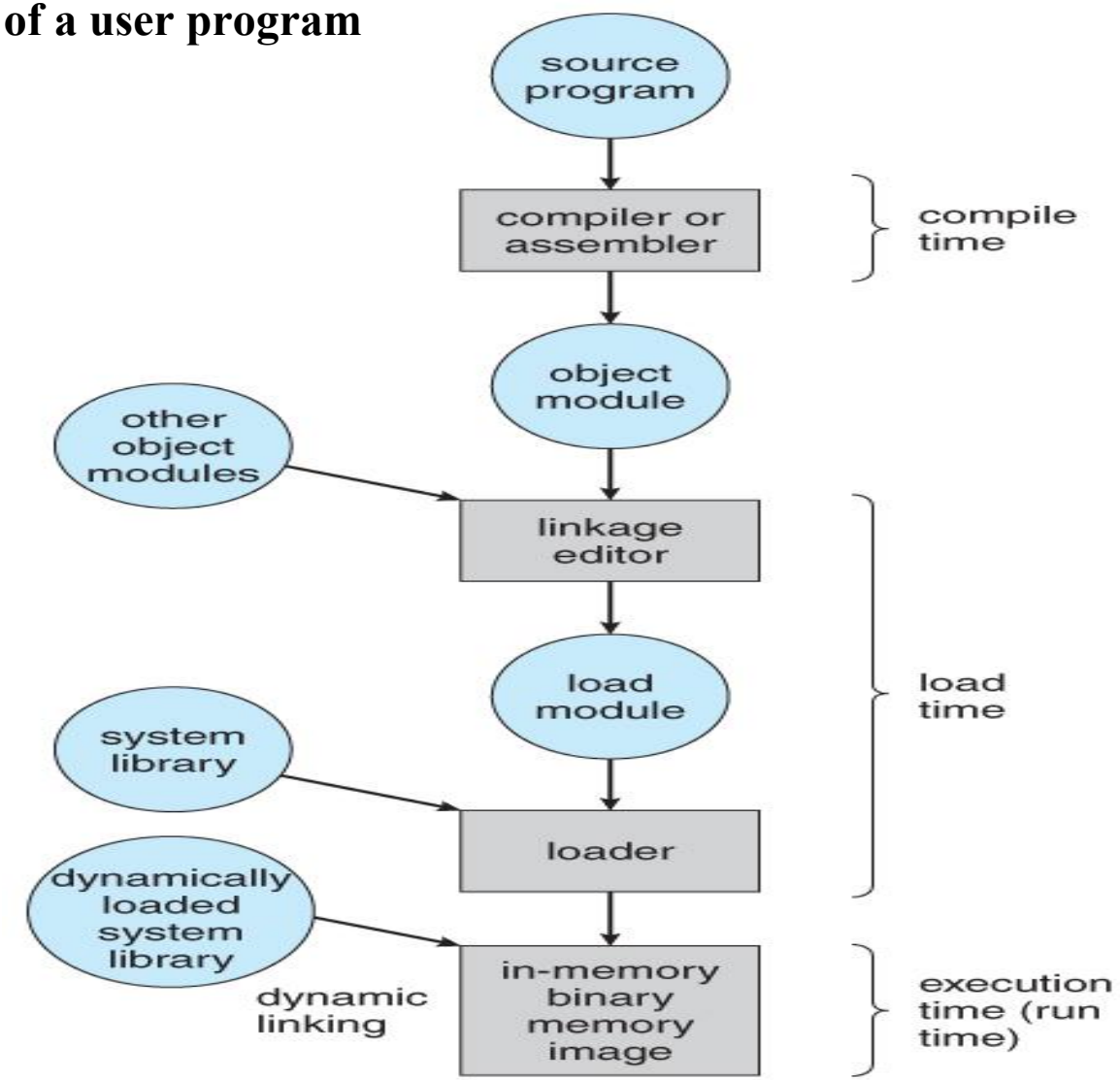
Load time:

- If the memory location of the process is not known, the compiler creates **re-locatable code** which is bound to absolute address only during load time.
- Reloading the user code will make it executable, even in the case of change of address.

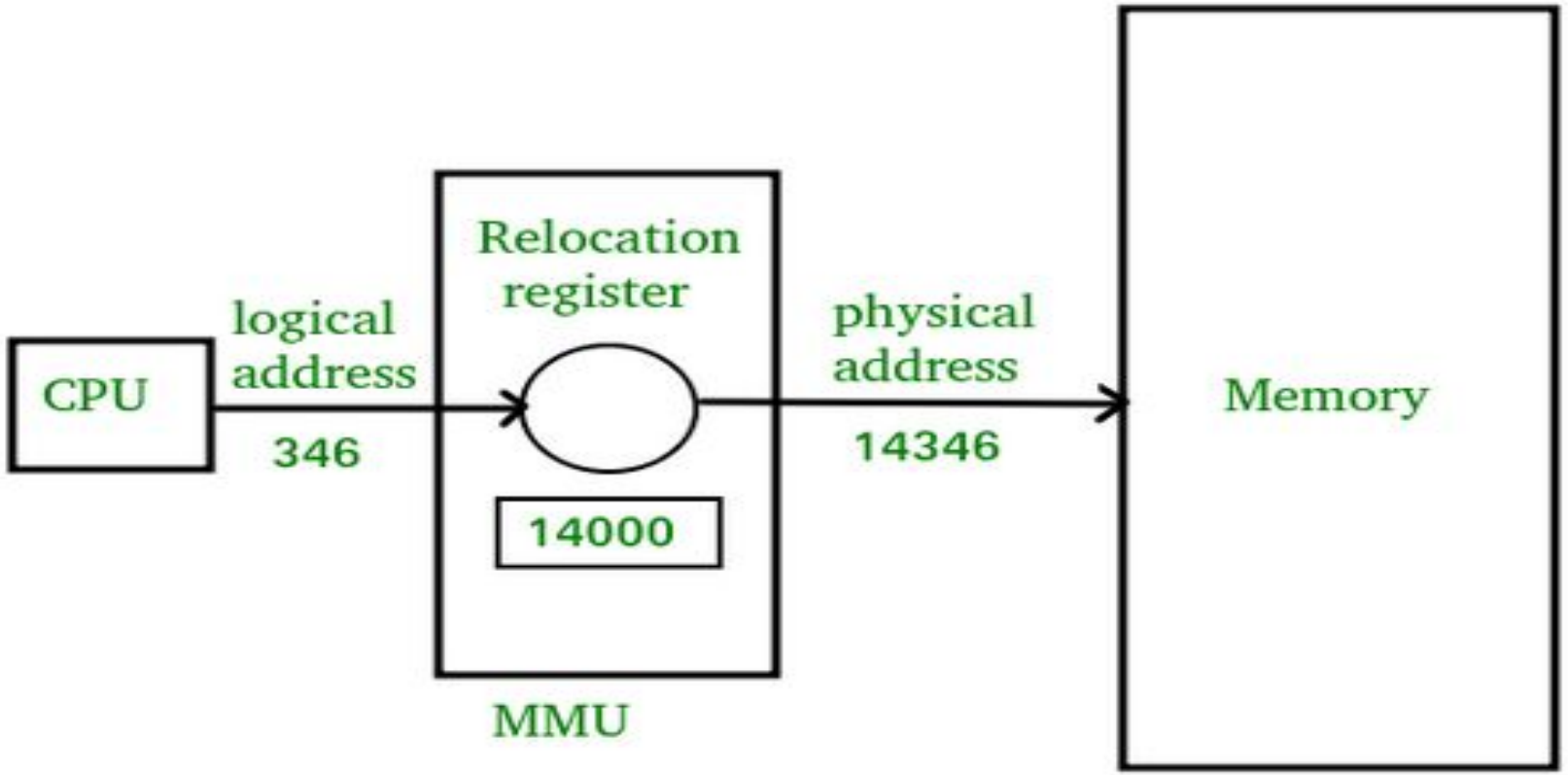
Mapping of Logical and Physical Address

- **Execution time:** In case, the process is shifted from one memory segment to another, then the binding is delayed until the execution time and done with the help of special hardware.

Multistep processing of a user program



The logical and physical address space



Dynamic relocation using a relocation register

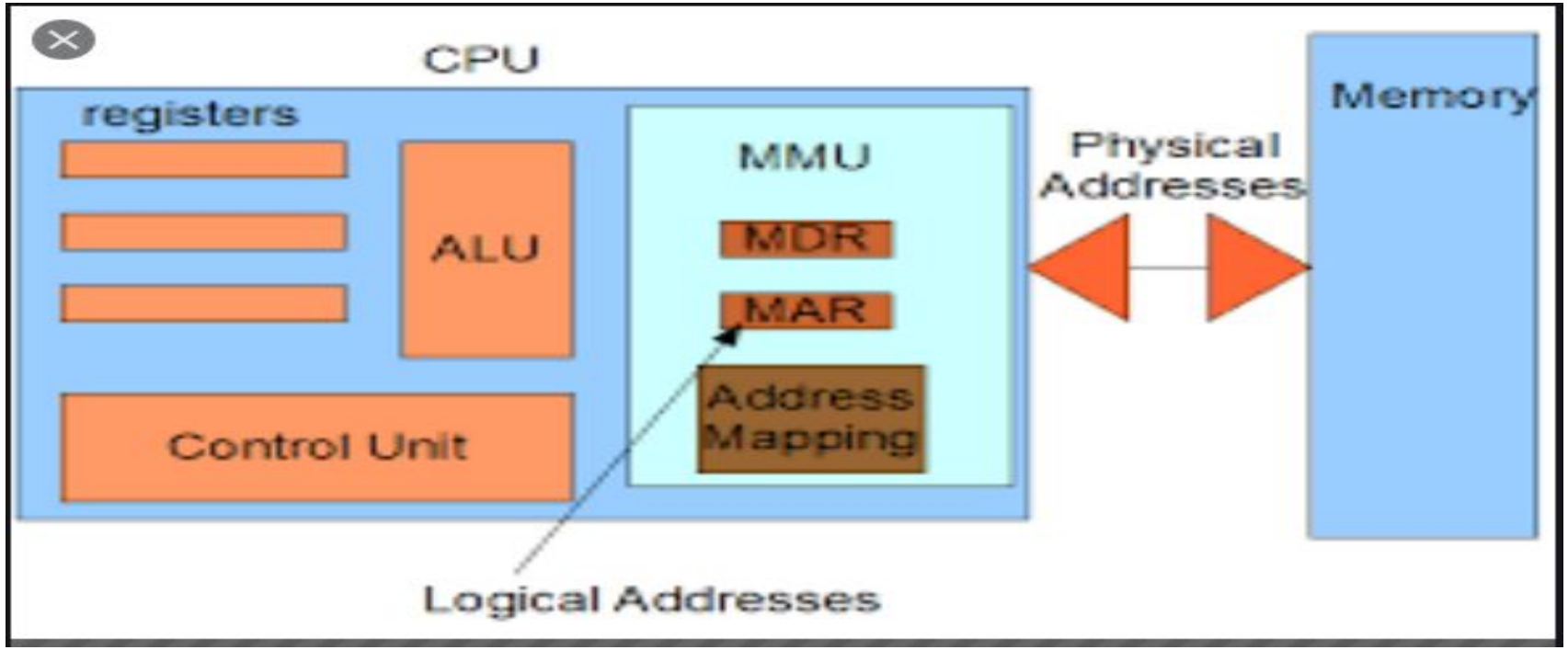
The logical and physical address space

- Logical address or the virtual address is the address generated by the **CPU** and physical address space is the address seen by the memory unit and loaded into the memory address register.
- Logical addresses generated by the program form the logical address space whereas the physical address space is formed with the set of physical addresses.

Memory Management Unit (MMU):

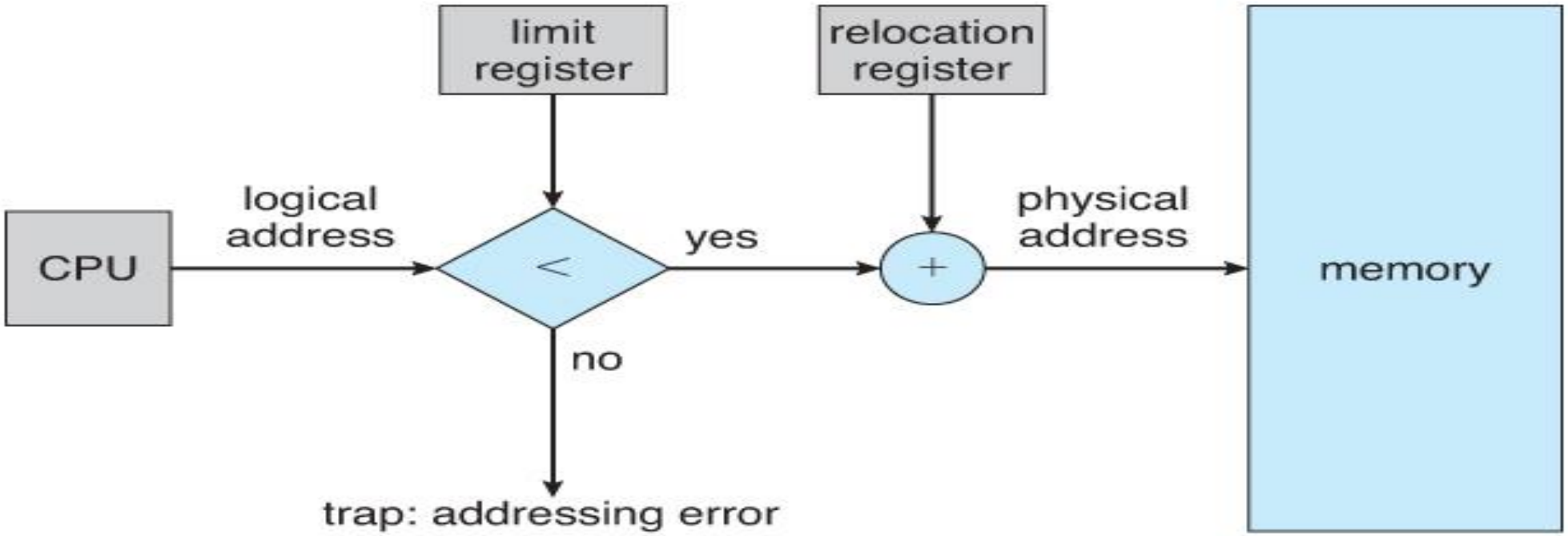
This maps the virtual addresses to the physical addresses with the help of hardware device during run time.

The logical and physical address space



Relocation of Address

- Relocation register R has the base address and all the addresses generated in the user process is added to the base address.
- Say if relocation register holds 7500 and logical address generated is 10, then the physical address would be 7500+10=7510
- Range of logical address: 0 to max
- Range of physical address: R+0 to R+max



Hardware support for relocation and limit registers

Dynamic Loading

- Dynamic loading is made use of for better utilization of memory space, and the routine is not loaded till the routine is called and all the routines are stored in re-locatable load format.

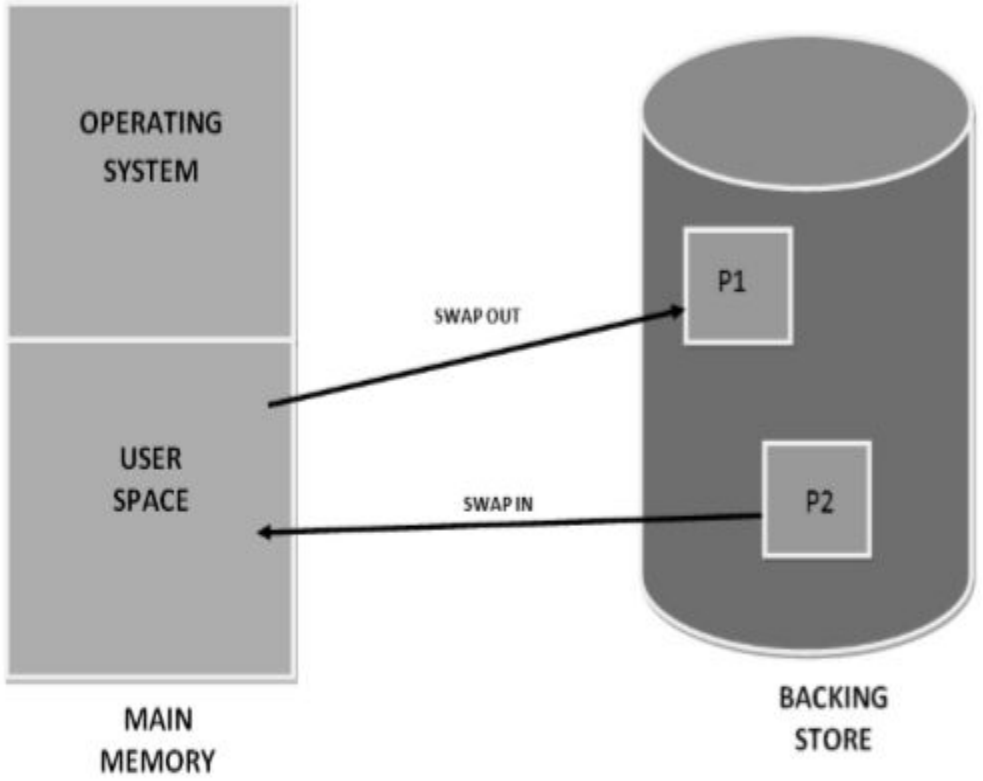
Advantages of Dynamic Loading

- No need to load the routines that are unused .
- No special OS support is required .
- Useful when huge codes are needed to handle the error routines .

Swapping

- The execution of the process is possible only if it is in the memory. Swapping provides the shifting of the process to a disk space which is then shifted to the memory to resume the process execution as shown in figure(next slide)
- For example, in a round-robin algorithm, once the allocated time slice allotted for the process lapses and the time slice allotted is large enough, it lets the other process to perform reasonable amounts of computing.

- Swapping gives preference to higher priority processes which require memory and replaces the lower priority processes.
- This variant of swapping which provides priority-based scheduling is named roll out, roll in.



Swapping

- Swapping makes use of a store for backing up.
- This store should be a fast device
- Large device which can store direct access mapped to memory images .

Constraints of Swapping:

- The process to be swapped should be idle.
- I/O Processes accessing buffers cannot be swapped.

Disadvantages of using the standard swapping are:

- High swapping time
- Less execution time

Contiguous Memory Allocation

- Contiguous memory allocation is a method deployed to partition the available memory space for both the user and the Operating system processes, operating in the system. To accommodate the several user processes, the single contiguous memory location is assigned to a process.
- Memory allocation involves division of memory into **partitions of fixed size** such that each partition has a fixed size and **each process takes up a partition**.
- Free partitions are selected and are loaded with the process present in the input queue.
- This partition is freed after the process ends. In this method, the available space is consolidated as a **hole** which is checked for available space when a process is to be loaded.
- If the hole is big enough, then the process is loaded. Else the process is kept on hold till space is freed. Holes which are adjacent to each other are merged to form a larger hole

Contiguous Memory Allocation

The strategies used for hole allocation may be:

1. **First fit**: The first hole that is bigger enough to fit the process is allocated .
2. **Best fit**: The hole which is just bigger enough to fit the process is allocated .
3. **Worst fit: The bigger hole** in the list of available holes is allocated .

Contiguous Memory Allocation

Example

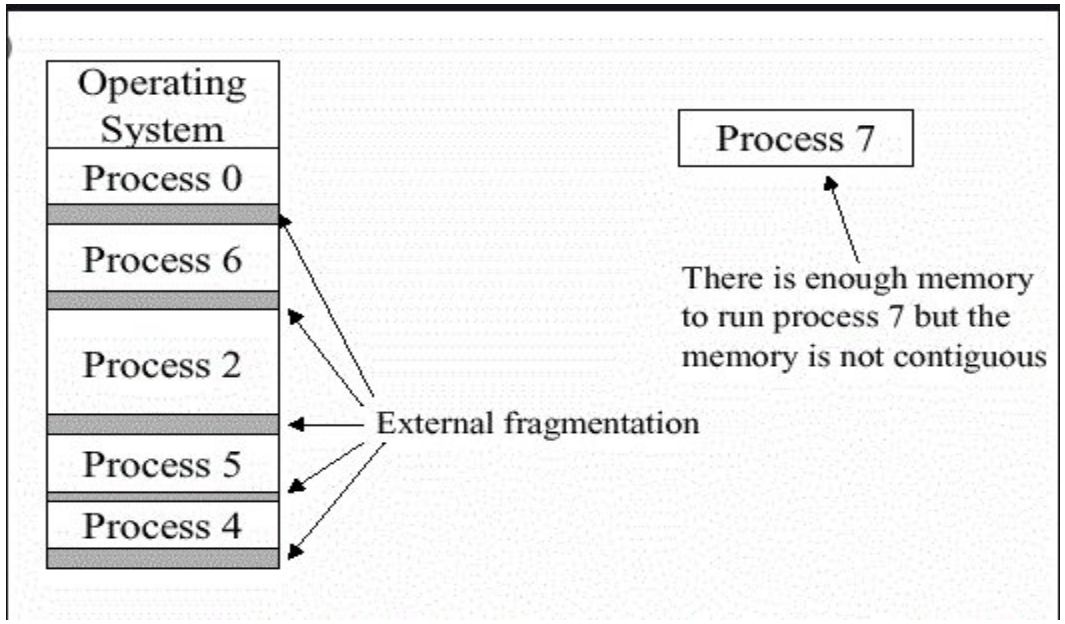
Consider a swapping system in which memory consists of the following whole sizes in memory order: 10K, 4k, 20k, 18k, 7k, 9k, 12k, and 15k. Which hole is taken for successive segment request of i)12k, ii)10k, iii)9k for first fit? Now repeat the question for best fit and worst fit.

First Fit		
12k	→	20k
10k	→	10k
9k	→	18k

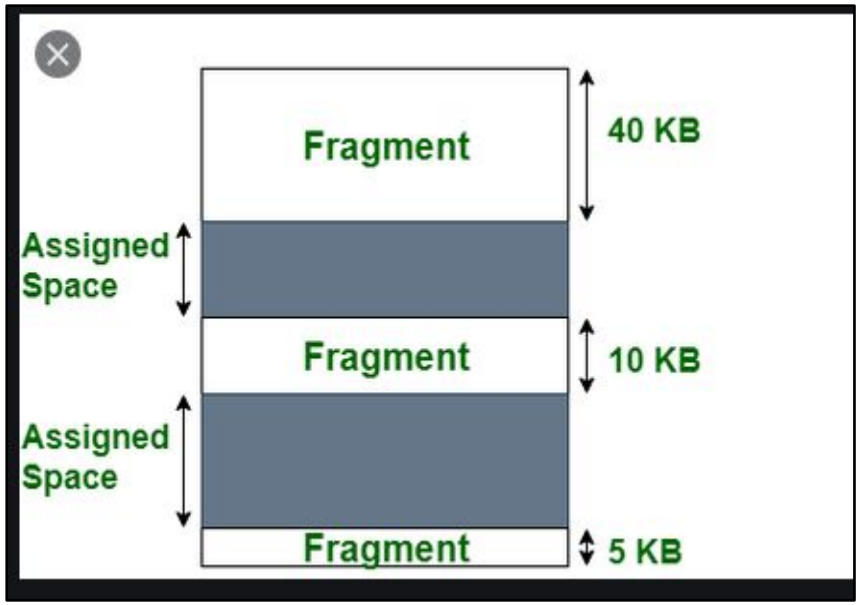
Best Fit		
12k	→	12k
10k	→	10k
9k	→	9k

Worst Fit		
12k	→	20k
10k	→	18k
9k	→	15k

Problems in contiguous memory allocation



External Fragmentation



Internal Fragmentation

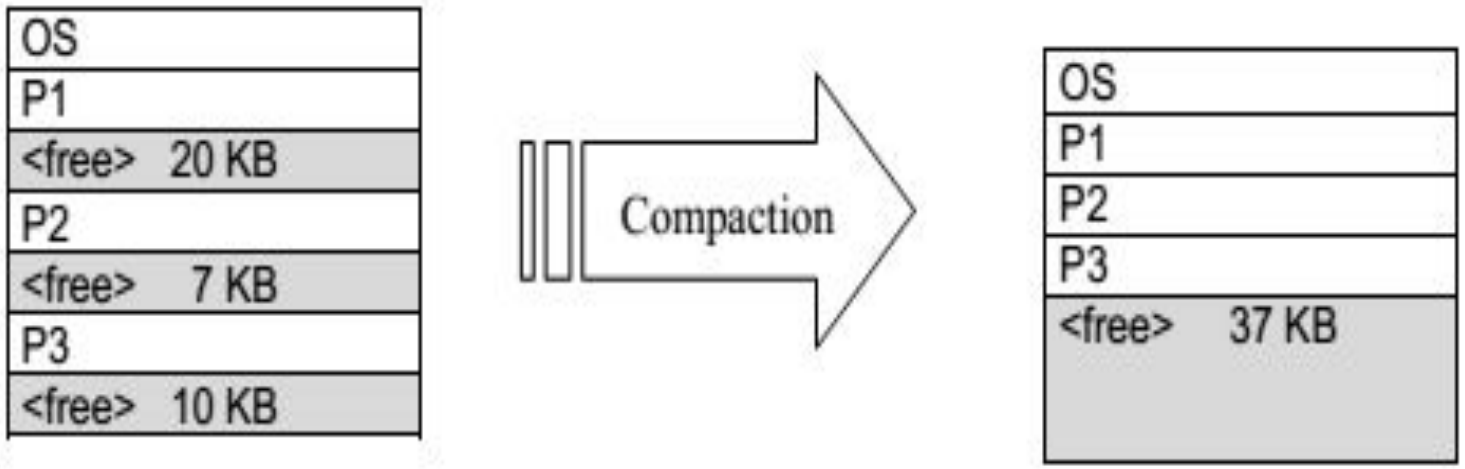
Problems in contiguous memory allocation

- **Internal fragmentation** happens when the memory is split into mounted-sized blocks. Whenever a method is requested for the memory, the mounted-sized block is allotted to the method. In the case where the memory allotted to the method is somewhat larger than the memory requested, then the difference between allotted and requested memory is called internal fragmentation.
- **External fragmentation** happens when there's a sufficient quantity of area within the memory to satisfy the memory request of a method. However, the process's memory request cannot be fulfilled because the memory offered is in a non-contiguous manner.

Problems in contiguous memory allocation

- **External fragmentation** occurs wherever best fit or first fit strategies are employed, since the loading and removing the processes from the hole creates segments of broken memory.
- This is avoided by including a block to accommodate these bits of fragmented memory between every two processes.
- Based on the system best fit or first fit algorithm is employed keeping in mind the fragmentation the strategy can induce.

Handling Fragmentation- Compaction



Handling Fragmentation

Ways to handle fragmentation:

1. **Compaction** or shuffling of memory spaces creates one large available block of data combining the fragments of empty memory space. Compaction cannot be adopted everywhere.
2. Making the address space of the processes **non-contiguous**.

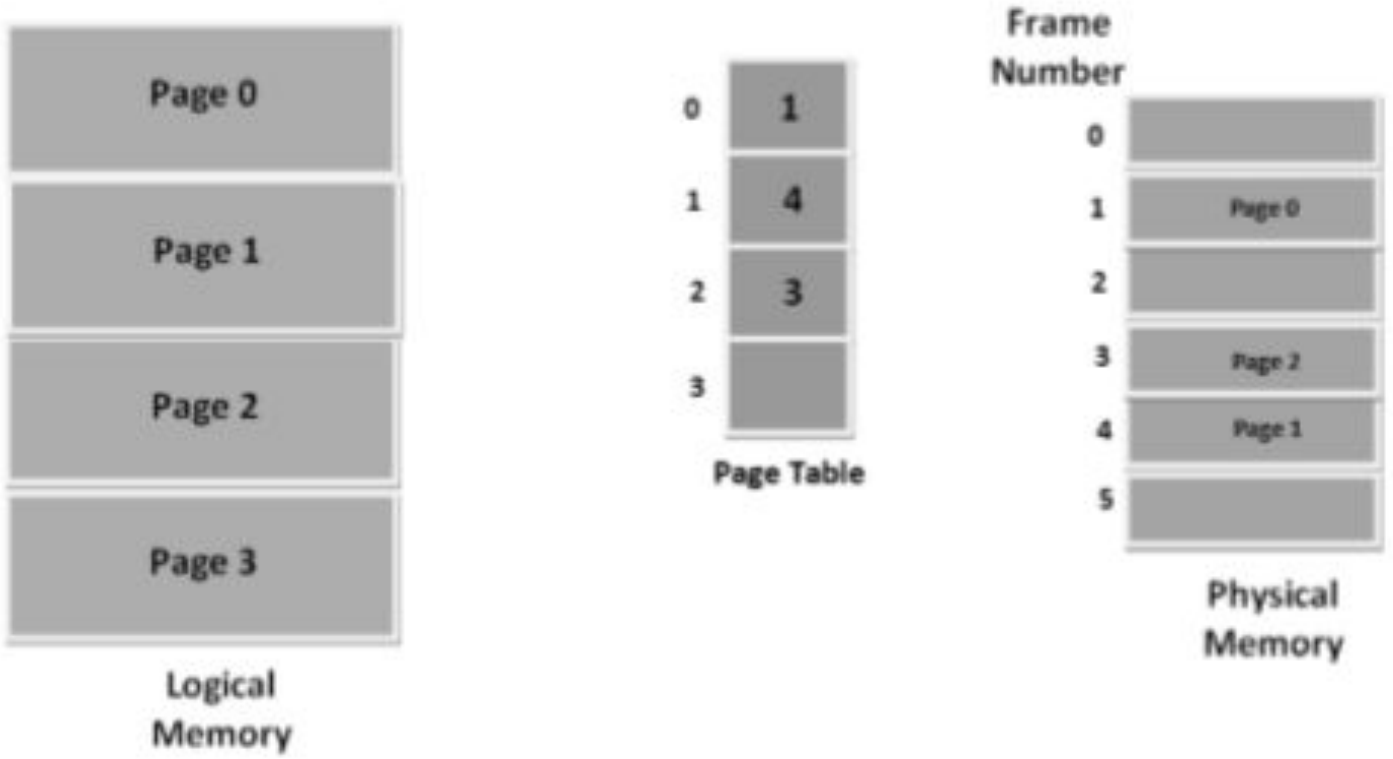
Paging and Structure of Paging Table

- Paging is a method of letting memory allotted to processes to be **non-sequential**, as a measure to **avoid fragmentation**.
- However, the storage unit that is used to backup needs to have enough space to perform paging.

Paging breaks the memory into:

- Logical: Into similar sized **pages**
- Physical: Into similar sized **frames**

Paging Model of Logical and Physical Memory



Paging

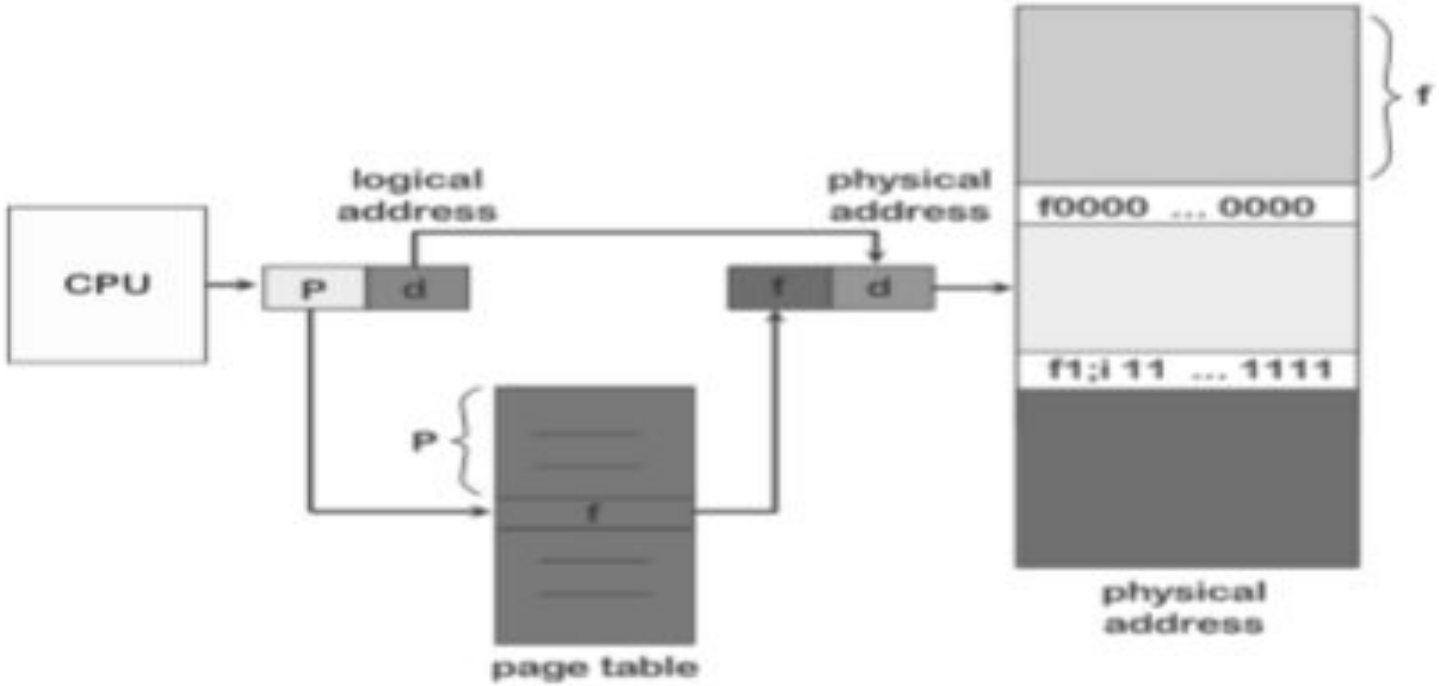
Address generated by CPU is divided into

- **Page number(p):** Number of bits required to represent the pages in Logical Address Space or Page number
- **Page offset(d):** Number of bits required to represent particular word in a page .

Physical Address is divided into

- **Frame number(f):** Number of bits required to represent the frame of Physical Address Space or Frame number.
- **Frame offset(d):** Number of bits required to represent particular word in a frame or frame size of Physical Address Space or word number of a frame or frame offset.

Paging Hardware



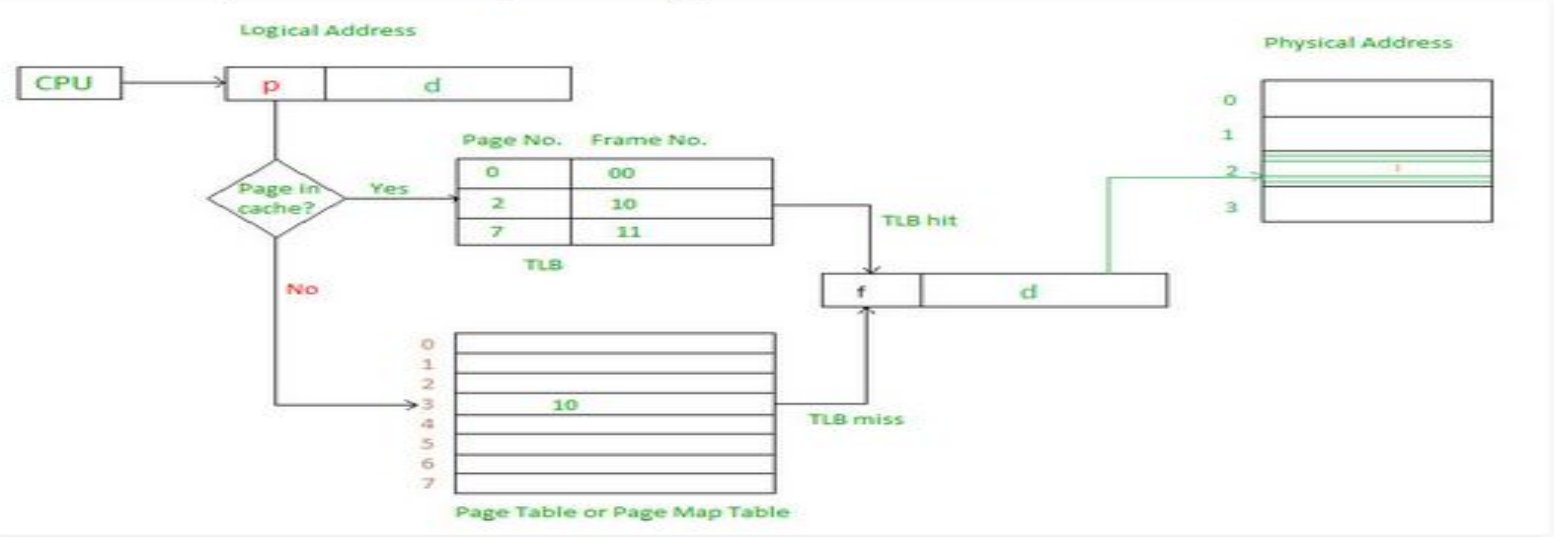
Paging

- ***Translation Lookaside Buffer* (*TLB*)** is nothing but a special cache used to keep track of recently used transactions. *TLB* contains *page table* entries that have been most recently used.
- If a *page table* entry is not found in the *TLB* (*TLB* miss), the *page* number is used as index while processing *page table*.
- Translation lookaside buffer (TLB) reduces the mapping time by presenting the associative memory with an item that consists of a value and key.
- The item is checked simultaneously with all the keys in the TLB and once it matches the value is returned.
- TLB is high-speed associative memory which holds only a few of the page table entries, also expensive to implement.
- TLB may store **ASID** (address space identifiers).

Paging

The hardware implementation of page table can be done by using dedicated registers. But the usage of register for the page table is satisfactory only if page table is small. If page table contain large number of entries then we can use TLB(translation Look-aside buffer), a special, small, fast look up hardware cache.

- The TLB is associative, high speed memory.
- Each entry in TLB consists of two parts: a tag and a value.
- When this memory is used, then an item is compared with all tags simultaneously.If the item is found, then corresponding value is returned.



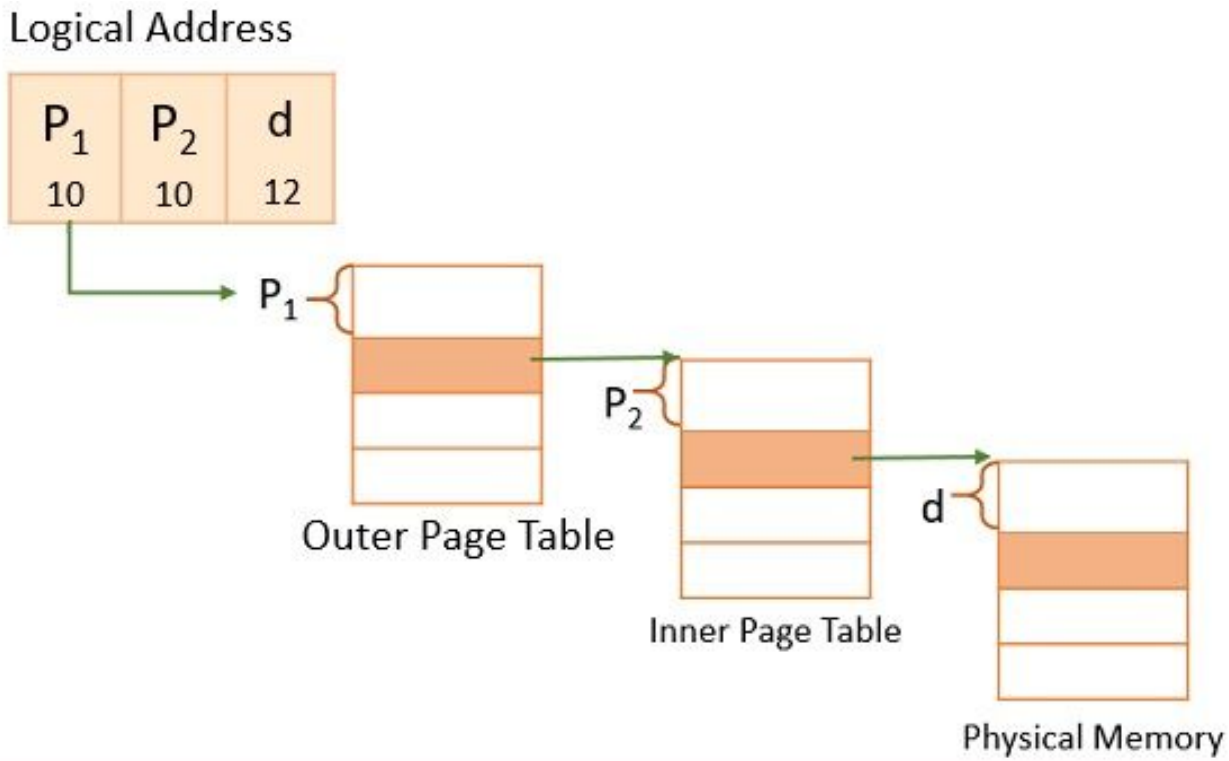
Structure of Page Table

- PLTR (Page table length register) can be used to include the size of the page table.
- The techniques which can be adopted in fabricating the page table are:

- Hierarchical Paging
- Hashed Paged Tables
- Inverted Page Tables

Hierarchical Paging

- With logical space using 2^{32} to 2^{64} bits for its address, each page entry becomes large. Page table can be disintegrated into smaller pieces to avoid a single large page table.
- As shown in the figure, a logical address is made up of **page number-20 bits, page offset-12 bytes which on splitting become page number-10 bits, page offset-10 bits .**This works for 32 bit logical address.
- When 64-bit logical addresses come into the play, three-layer or four-level paging table is made use of.

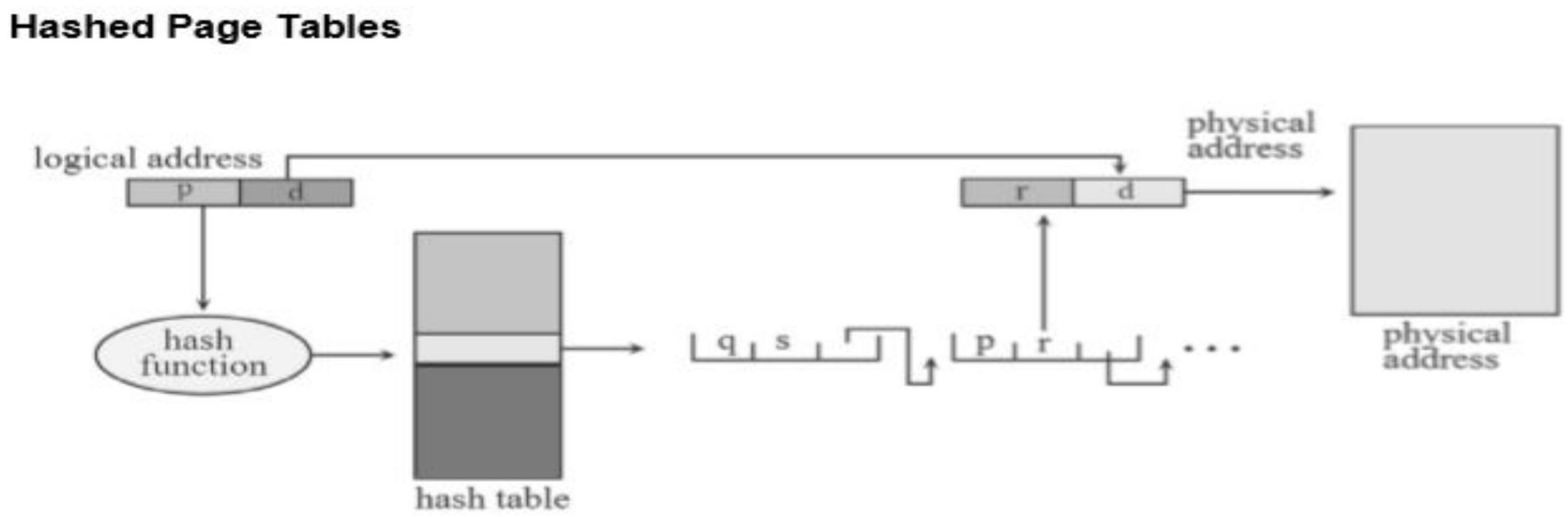


Structure of Two –Level Hierarchical Page Table

Hashed Page Tables

- When address space is denoted with the help of more than 32 bits, a **virtual page number called hash value** is used, as shown in Figure (next slide).
- A linked list of elements pointing to the same location is saved as an entry in the hash table and the hash table contains many such linked lists.
- Each element in the hash table is made up of the following:
  - Virtual page number
  - Value of the mapped page frame
  - Pointer to the next element in the list

Hashed Page Tables

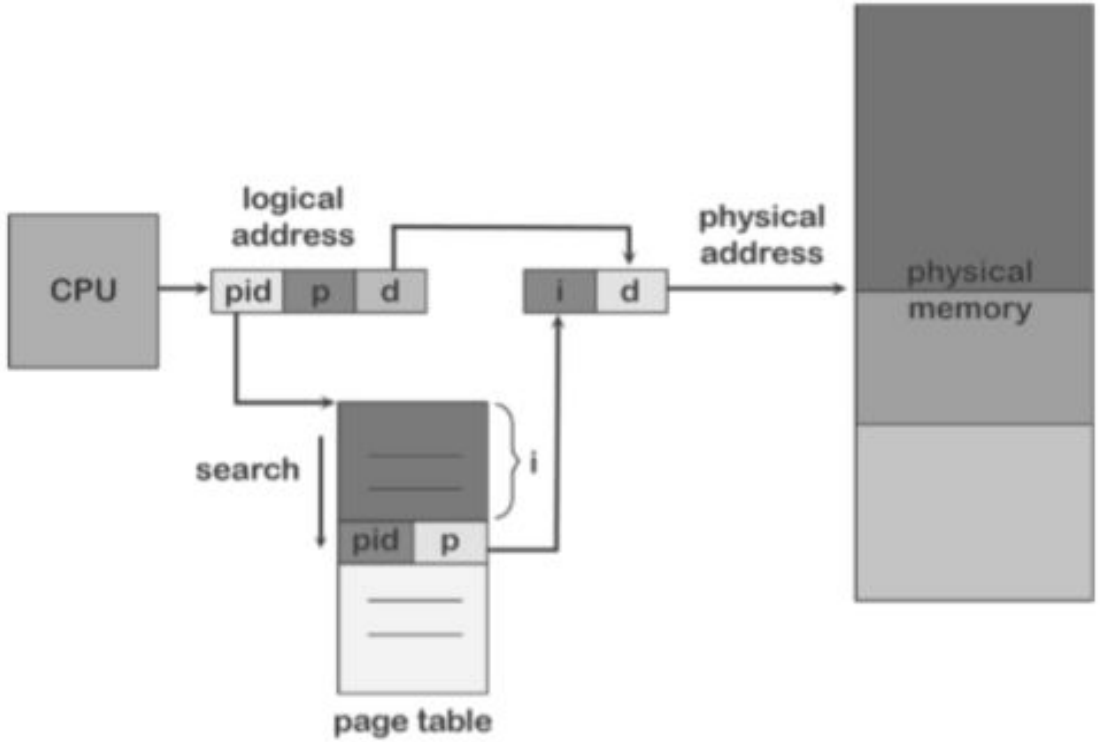


Inverted Page Tables

- Each process is associated with the page table which contains information about the page the process is making use of.
- Page table has a slot for each entry's **virtual address** which is mapped to a physical address.
- Page tables contain millions of entries and is memory consuming.
- To solve the above overhead, inverted page tables which **store real page of memory** and the **process associated with it**, is used.
- **Address space identifier** is required by every entry of the table, in certain cases of inverted table to map the process to the pertaining physical address.
- In short, the inverted page table maps **virtual page to its physical page frame.**
- Format: **< process id, page number>**

Inverted Page Tables

- As shown in the figure, at every memory occurrence the inverted page table is searched with ordered pair **<process id, page number>** of the virtual address against the entries in the inverted page table and on finding the match, the physical address is generated.
- Inverted page table decreases the memory usage but, time required for searching the virtual address against all the entries in the table is high.



Segmentation

- Separation of the actual layout of physical memory from the user's aspect of memory is mandatory to protect the memory space.
- Segmentation is a memory management technique which supports user's aspect of memory with the logical address being viewed as a collection segments.
- Segments are identified by segment number and an offset which is the size of the segment.
- **<segment number, offset>**
- Segments can be of different sizes.

Segmentation

Separate segments are created by the compiler for:

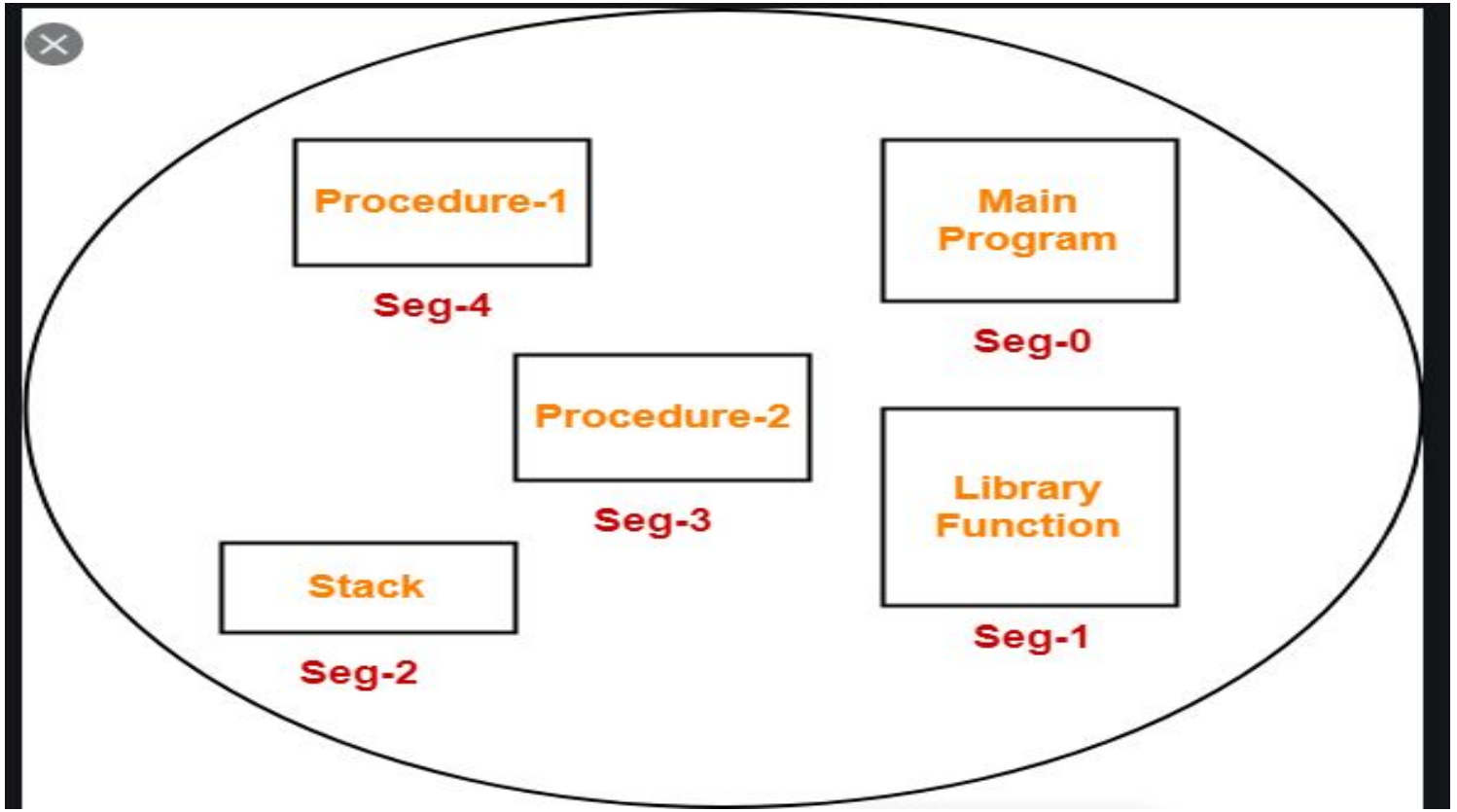
- C library
- Stacks that are used by each thread
- Heap from which the memory allocation is done
- Code
- Global variables

Segment number has:

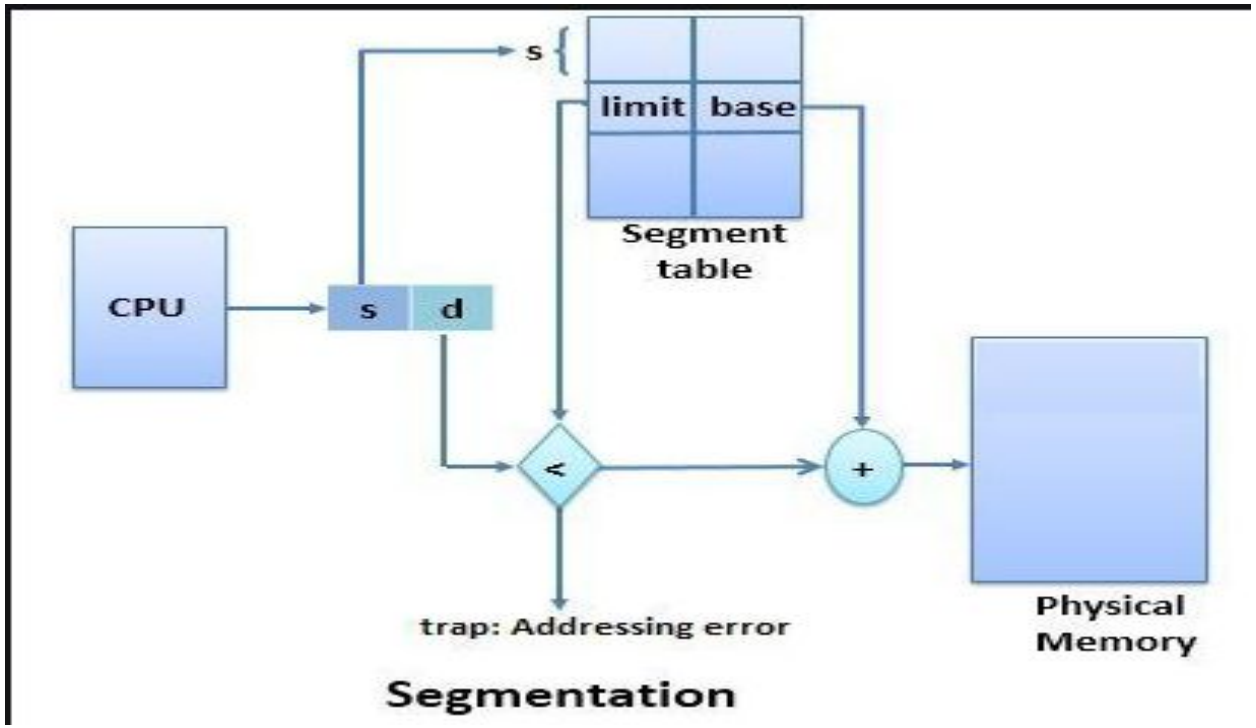
- The base address which is the physical address of the location where the segment is saved
- The limit address which provides the segment length.

For example, if 33rd byte of segment 1 needs to be accessed whose base address value is 4400, then 4400+33= 4433 is given the access.

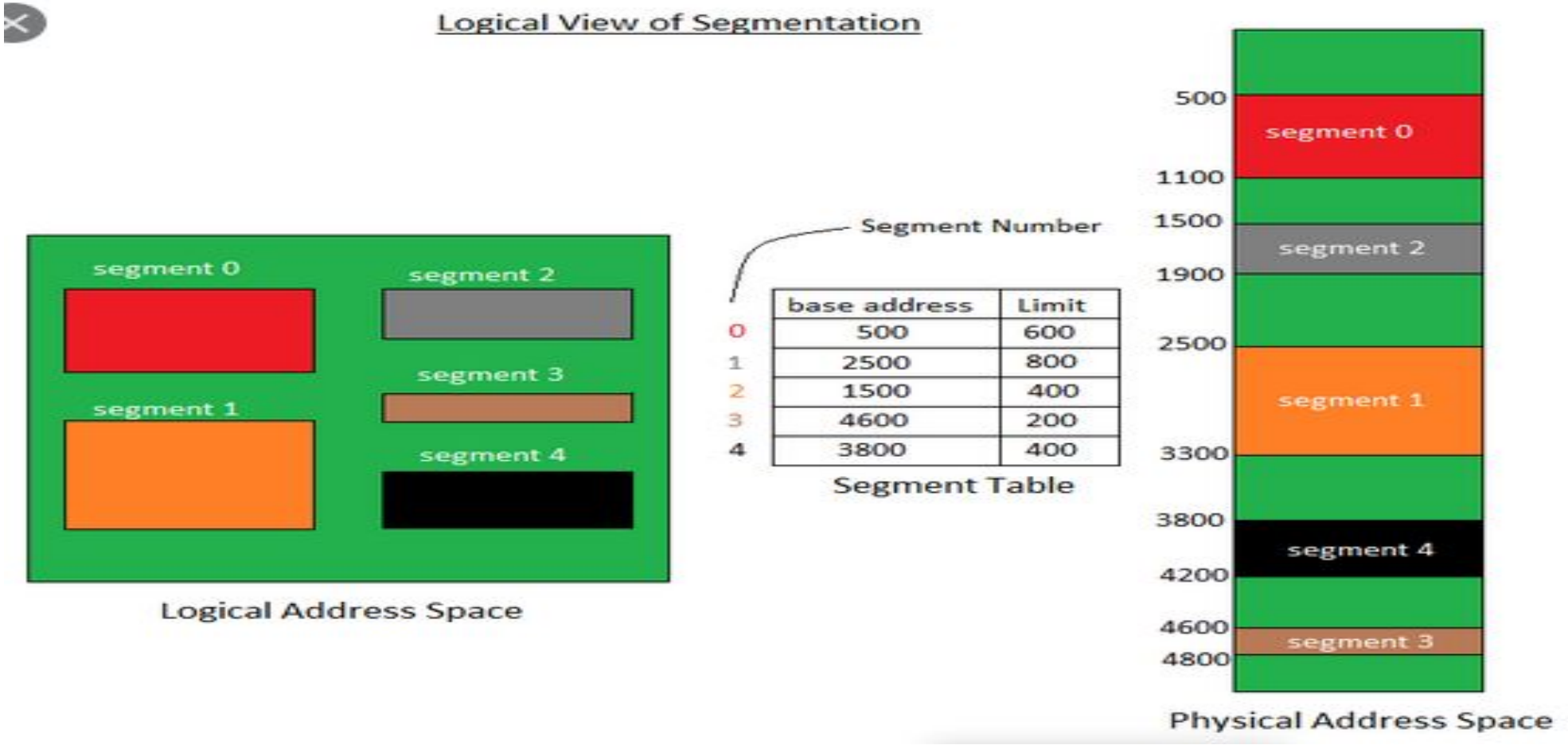
Segmentation



Segmentation



Segmentation



Page Protection

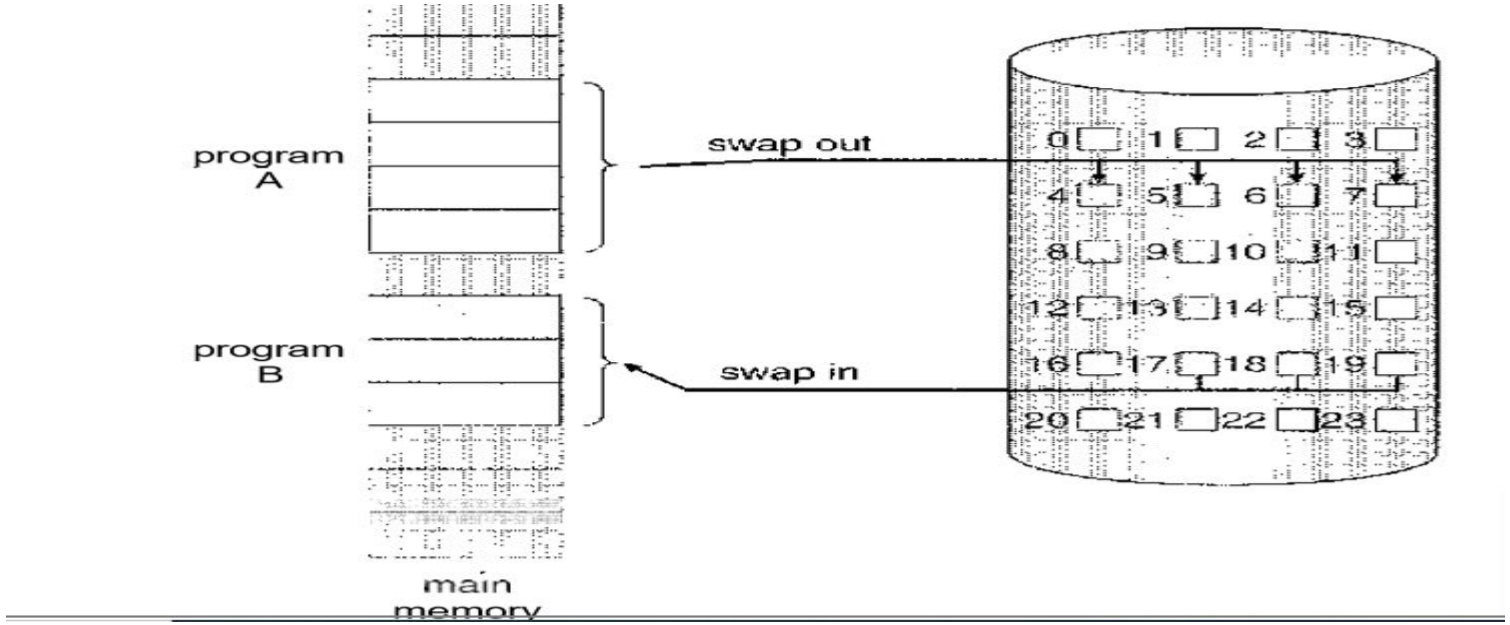
- Page protection is enabled by checking if the page is read-only or has read-write access using a bit assigned to the page for this purpose.
- The bit can be expanded to include execute only access as well.
- **Valid-invalid bit** is added to each page
- **Valid**: If the associated logical address stores the page
- **Invalid**: If the associated logical address does not store the page

Virtual Memory Management

- Virtual memory is a technique that allows the execution of processes that are not completely in memory.
- One major advantage of this scheme is that programs can be larger than physical memory.
- Further, virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory.
- This technique frees programmers from the concerns of memory-storage limitations.
- Virtual memory also allows processes to share files easily and to implement shared memory.

Benefits of Virtual Memory

- A program would no longer be constrained by the amount of physical memory that is available.
- Users would be able to write programs for an extremely large virtual address space, simplifying the programming task.



Benefits of Virtual Memory

- Because each user program could take less physical memory, more programs could be run at the same time, with a corresponding increase in CPU utilization and throughput but with no increase in response time or turnaround time.
- Less I/O would be needed to load or swap each user program into memory, so each user program would run faster.

Benefits of Virtual Memory

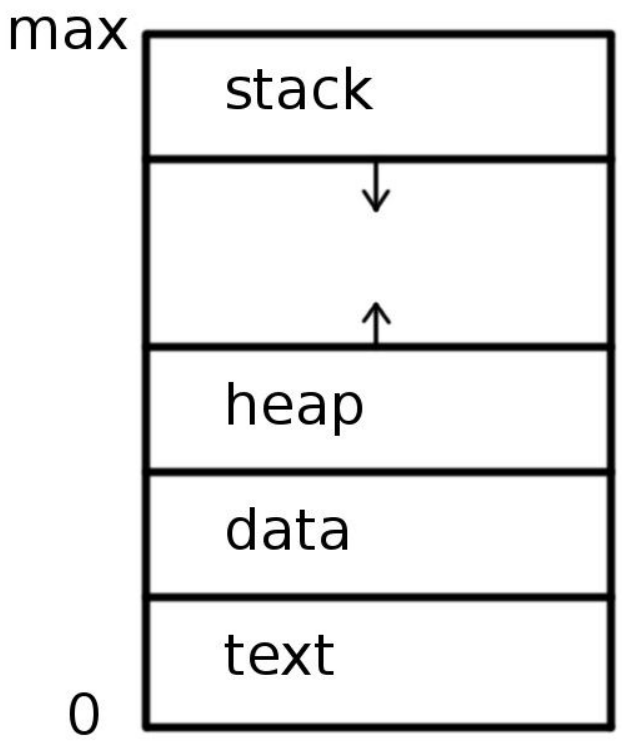
- Virtual memory involves the separation of logical memory as perceived by users from physical memory.
- This separation, allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available .
- Virtual memory makes the task of programming much easier, because the programmer no longer needs to worry about the amount of physical memory available; they can concentrate instead on the problem to be programmed.

Physical Address vs Virtual Address

- The virtual address space of a process refers to the logical (or virtual) view of how a process is stored in memory.
- Typically, this view is that a process begins at a certain logical address—say, address 0—and exists in contiguous memory.
- In fact physical memory may be organized in page frames and that the physical page frames assigned to a process may not be contiguous.
- It is up to the memory management unit (MMU) to map logical pages to physical page frames in memory.

Virtual Address Space

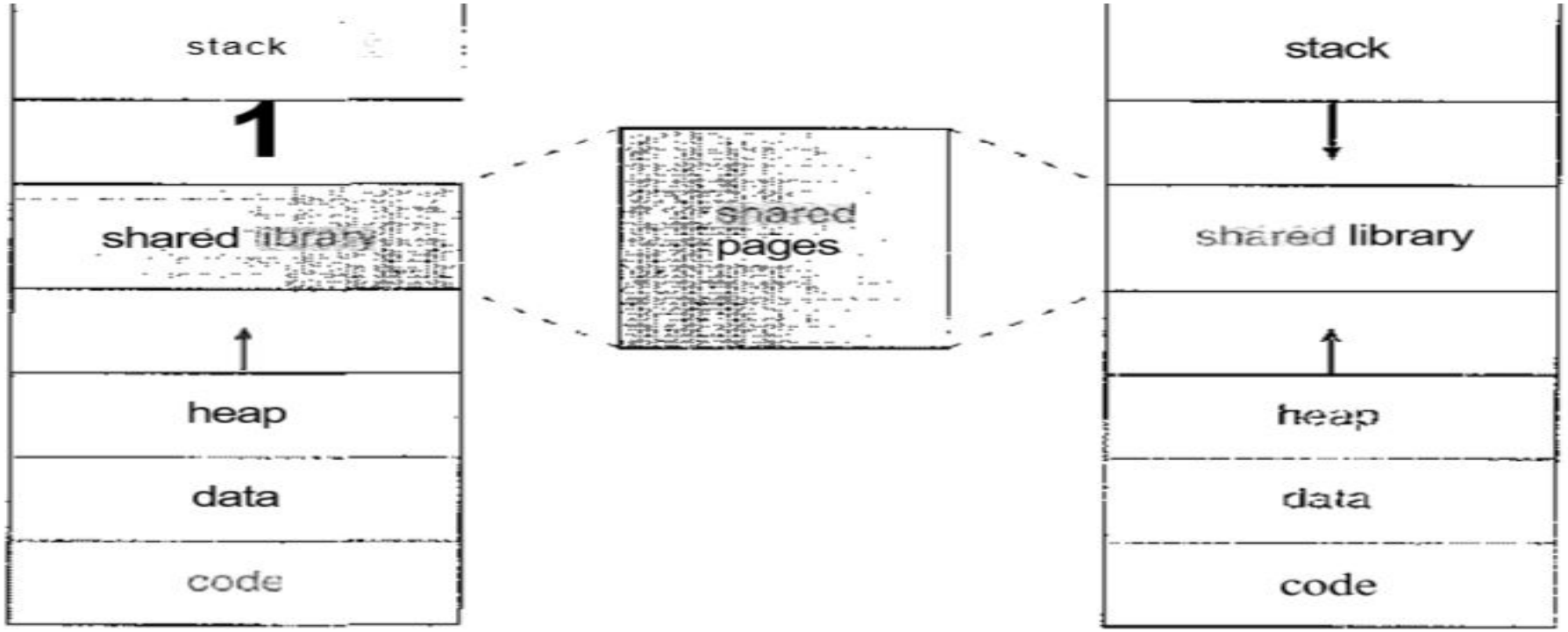
- We allow for the heap to grow upward in memory as it is used for dynamic memory allocation.
- Similarly, we allow for the stack to grow downward in memory through successive function calls.
- The large blank space (or hole) between the heap and the stack is part of the virtual address space



Virtual Address Space

- But will require actual physical pages only if the heap or stack grows. Virtual address spaces that include holes are known as **sparse address spaces.**
- Using a sparse address space is beneficial because the holes can be filled as the stack or heap segments grow or if we wish to dynamically link libraries (or possibly other shared objects) during program execution

Shared library using virtual memory



Swapping in Virtual Memory

- When a process is to be swapped in, the **pager** guesses which pages will be used before the process is swapped out again.
- Instead of swapping in a whole process, the pager brings only those necessary pages into memory.
- Thus, it avoids reading into memory pages that will not be used anyway, decreasing the swap time and the amount of physical memory needed.

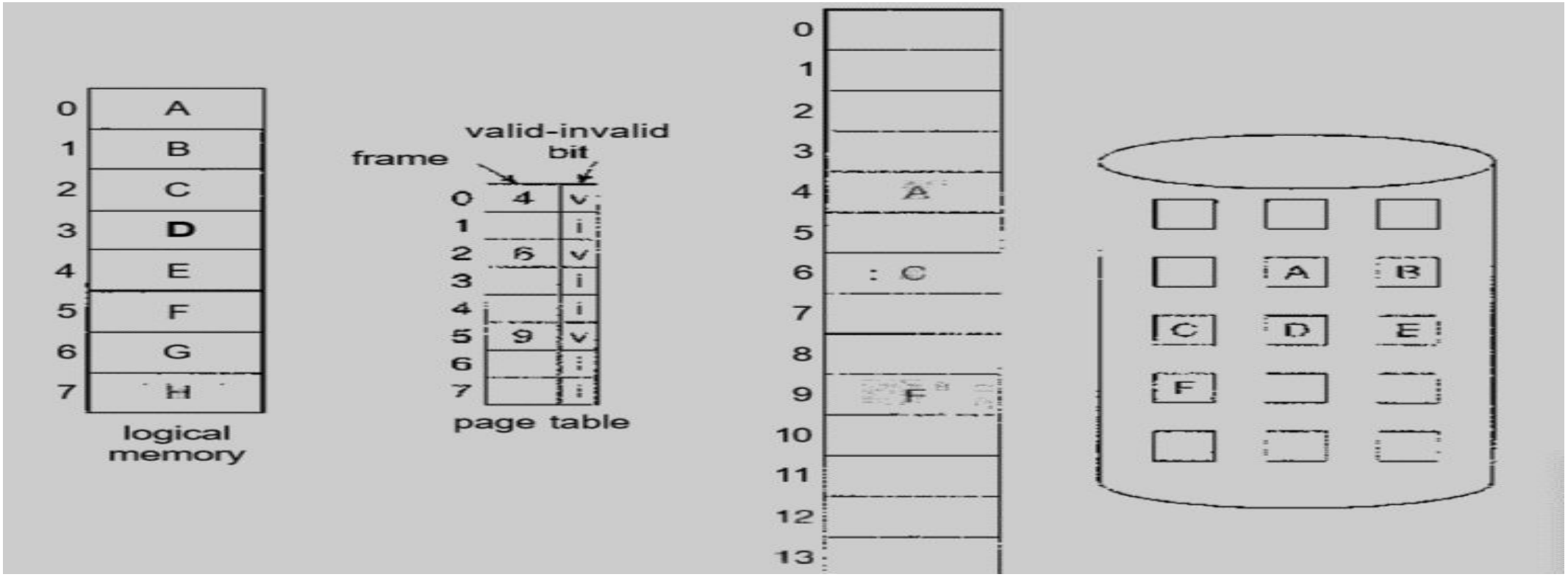
Demand Paging

- Demand paging is implemented using a **lazy swapper** mechanism which swaps a page into the memory on demand.
- A demand-paging system is similar to a paging system with swapping where processes reside in secondary memory (usually a disk).
- When we want to execute a process, we swap it into memory. Rather than swapping the entire process into memory, however, we use a lazy swapper.
- A **lazy swappe**r never swaps a page into memory unless that page will be needed.
- Since we are now **viewing a process as a sequence of pages**, rather than as one large contiguous address space, use of the term swapper is technically incorrect.
- A swapper manipulates entire processes, whereas a pager is concerned with the individual pages of a process.
- We thus use **pager**, rather than swapper, in connection with demand paging.

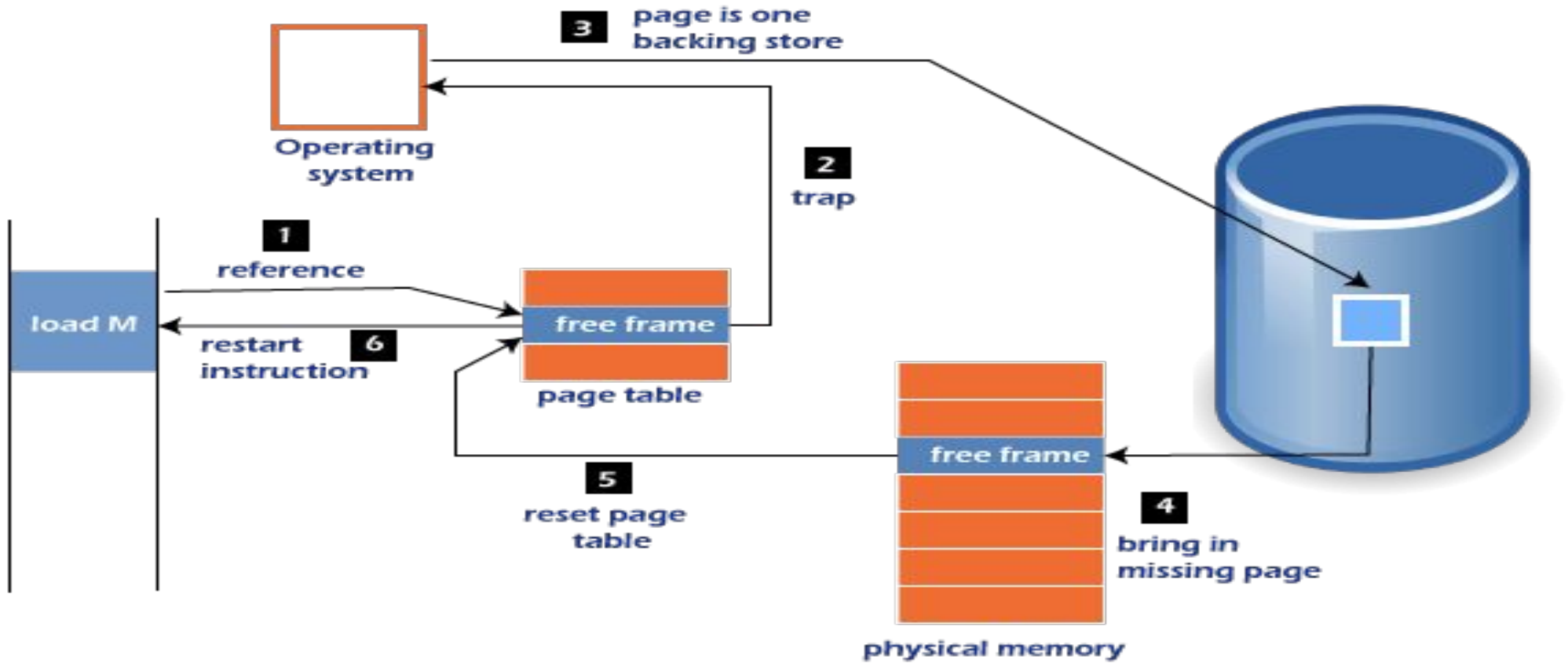
Page Fault

- If the process tries to access a page that was not brought into memory, access to a page marked invalid causes a **page-fault** trap.
- The paging hardware, in translating the address through the page table, will notice that the **invalid bit is set**, causing a trap to the operating system.
- This trap is the result of the operating system's failure to bring the desired page into memory.

Page table when some pages are not in main memory



Handling Page Faults



Steps in handling Page Faults

1.We check an internal table (usually kept with the process control block) for this process to determine whether the reference was a valid or an invalid memory access.

2.If the reference was invalid, we terminate the process. If it was valid, but we have not yet brought in that page, we now page it in.

3. We find a free frame (by taking one from the free-frame list, for example).

4. We schedule a disk operation to read the desired page into the newly allocated frame.

5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.

6. We restart the instruction that was interrupted by the trap. The process can now access the page as though it had always been in memory.

Hardware Support for demand paging

Page table: This table has the ability to mark an entry invalid through a valid-invalid bit or special value of protection bits.

Secondary memory:

- This memory holds those pages that are not present in main memory.
- The secondary memory is usually a high-speed disk. It is known as the swap device, and the section of disk used for this purpose is known as swap space.

Page Replacement Algorithm

The below-given steps are to be followed for the page replacement:

- Page to be loaded onto the disk is chosen, and its disk location is found
- Search for a free frame .
- If available, the free frame is used .When there is no free frame available, page replacement algorithm frees frames from the victim frame and updates frame table and page table .
- The page that is required is read into the freed frame, and the page table and frame table are updated.
- The user process is restarted.

Page Replacement Algorithm

- This algorithm performs two-page transfers in case a free frame does not exist (i.e. the victim page transfer done to free frame and the desired page transfer read into the freed frame).
- It can be reduced by using dirty or modify bit which signals if the page is:
- Modified, if the bit is set to one
- Not modified from the time it was read into the memory and is available if the bit is set to zero

FIFO Page Replacement Algorithm

- The easiest page replacement algorithm is FIFO which replaces the first page loaded in the memory.
- All the pages are linked to the time that they were brought into the memory.
- FIFO creates a queue and replaces the page present at the beginning of the queue.

FIFO Page Replacement Algorithm

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in FIFO = 6

Bélády's *anomaly* is the name given to the phenomenon where increasing the number of *page* frames results in an increase in the number of *page* faults

Least Recently Used (LRU) Page Replacement

Replace the page which is being least recently used.

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	6	7	7
Frame 2		7	7	7	7	7	7	2	2	2
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Miss	Hit

Number of Page Faults in LRU = 6

Optimal Page Replacement

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

Request	4	7	6	1	7	6	1	2	7	2
Frame 3			6	6	6	6	6	2	2	2
Frame 2		7	7	7	7	7	7	7	7	7
Frame 1	4	4	4	1	1	1	1	1	1	1
Miss/Hit	Miss	Miss	Miss	Miss	Hit	Hit	Hit	Miss	Hit	Hit

Number of Page Faults in Optimal Page Replacement Algorithm = 5

Allocation of Frames

For a process to execute the frames that are to be allotted to each process must be more; else, it slows the process execution by raising the page fault rate.

Frame allocation decides how the frames are allotted to the processes from:

- The minimum frames per process as defined by the system's architecture
- The maximum frames based on the available physical memory

Types of frame allocation

- Proportional allocation: Allocating frames based on the process size
- Equal allocation: Dividing the available frames equally among the requesting processes

Global vs. Local allocation:

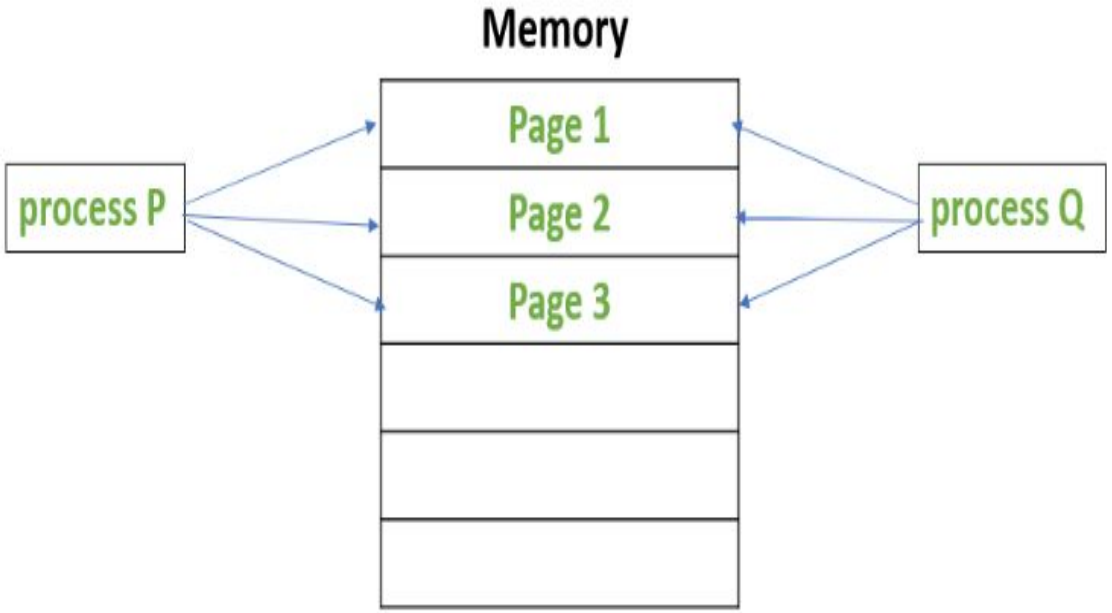
Global replacement: When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from the set of all frames, even if that frame is currently allocated to some other process; that is, one process can take a frame from another.

Local replacement: When a process needs a page which is not in the memory, it can bring in the new page and allocate it a frame from its own set of allocated frames only.

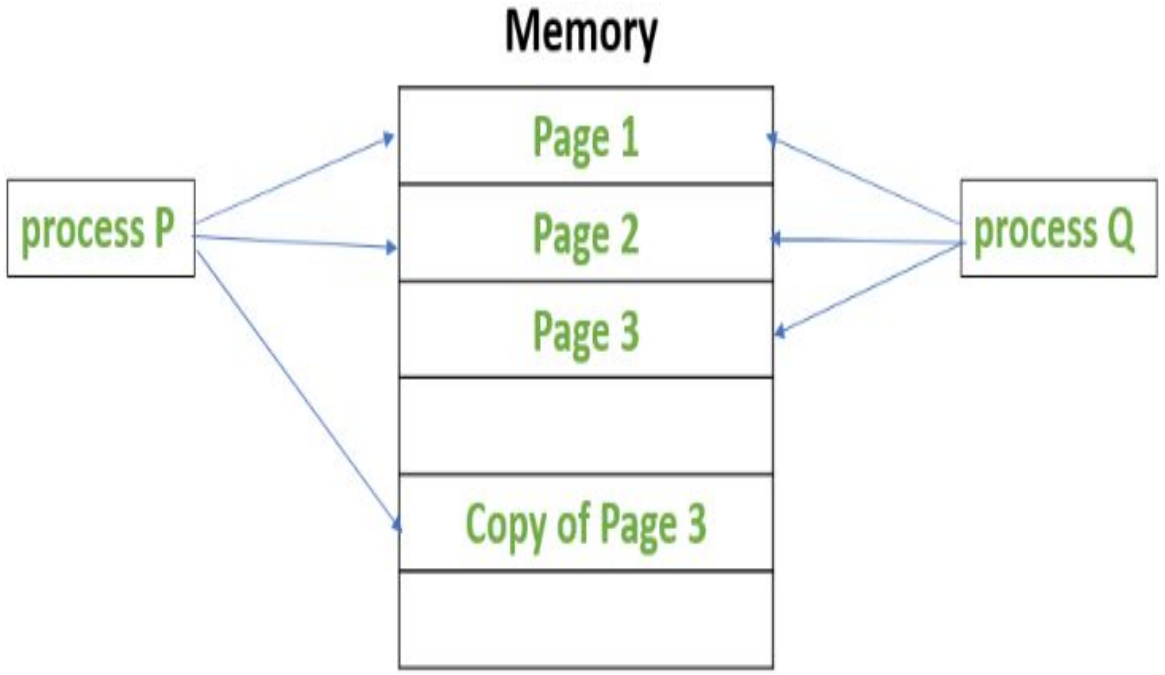
Copy-on-Write

- Copy on Write or simply COW is a resource management technique. One of its main use is in the implementation of the fork system call in which it shares the virtual memory(pages) of the OS.
- The idea behind a copy-on-write is that when a parent process creates a child process then both of these processes initially will share the same pages in memory and these shared pages will be marked as copy-on-write which means that if any of these processes will try to modify the shared pages then only a copy of these pages will be created and the modifications will be done on the copy of pages by that process and thus not affecting the other process.
- Suppose, there is a process P that creates a new process Q and then process P modifies page 3.

Copy-on-Write



Before process P modifies Page 3

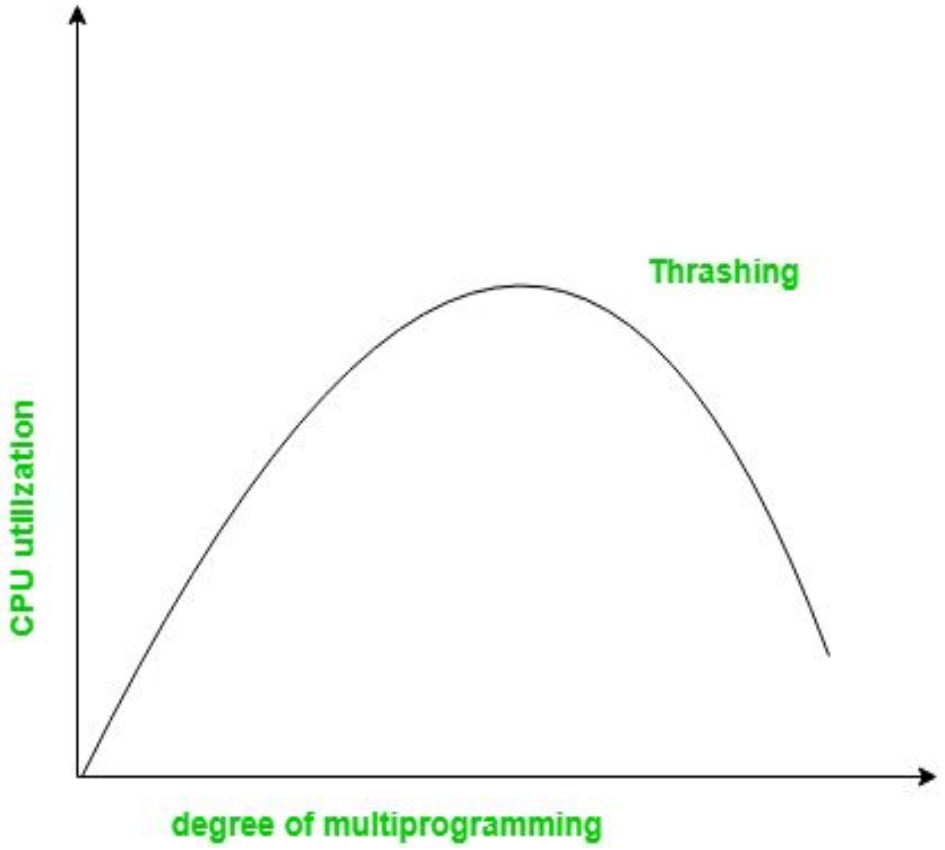


After process P modifies Page 3

Thrashing

- Low priority processes have the problem of not getting enough frames, since the frames allocated is less than the minimum number of frames required by the system architecture.
- Since the process does not have enough frames, it keeps paging again and again, since **the page which is replaced is immediately needed again and again.**
- The process which undergoes the above problem is said to be thrashing.

- The basic concept involved is that if a process is allocated too few frames, then there will be too many and too frequent page faults.
- As a result, no useful work would be done by the CPU and the CPU utilization would fall drastically.



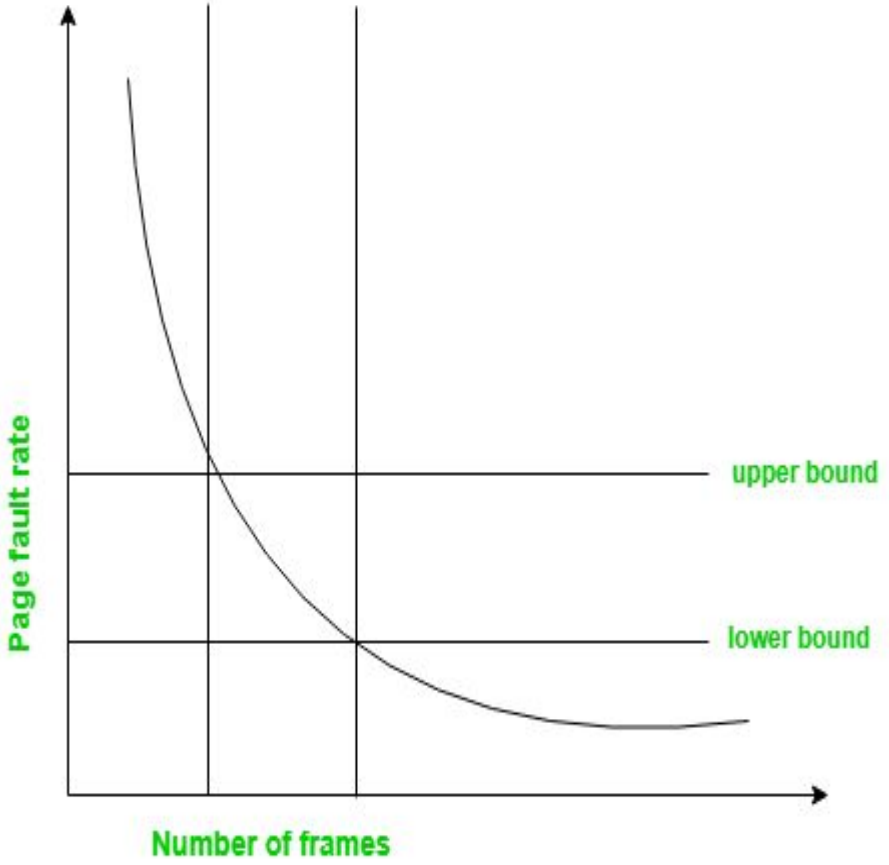
Causes for Thrashing

Thrashing is a result of the improper allocation of frames. Thrashing is caused as follows:

- When CPU utilization is low, the degree of multiprogramming is increased, and more processes need frames and take the frames from other processes via global replacement.
- This increases the paging requests and causes the processes to wait, which in turn decreases the CPU utilization.
- Again, the CPU scheduler tries to increase CPU utilization by starting execution of more processes

Page Fault Frequency:

- PPF or the **page fault frequency** is monitored as a measure to prevent thrashing and increase CPU utilization. The PPF has a permissible range which is constantly monitored.
- If page fault rate is too low, indicates that the process has too many frames, the frames that are allotted to each process are more than what is required
- If page fault rate is too high, it indicates that the process has too few frames allocated to it.
- If the page fault rate falls below the lower limit, frames can be removed from the process. Similarly, if the page fault rate exceeds the upper limit, more frames can be allocated to the process.



Other factors considered for paging

Prepaging:

- Prepaging involves keeping a set of pages in the working set and remembering the working set even if the process is to be suspended. Prepaging is employed if the cost of prepaging is lesser than the cost of page faults caused otherwise.

Page size

- ***Small page size*** increases the number of pages and entries in the page table, decrease total I/O and total allocated memory and makes it easier for the locality concept reducing internal fragmentation as the memory is maximum utilised.
- ***Large page size:*** All the contents of the page are allocated or transferred even if they are not needed, and provides for smaller page table.

Other factors considered for paging

- **Program structure**: The user with proper knowledge about demand paging can design programs which can reduce paging.
- **I/O interlock**: Few pages need to be locked during demand paging especially I/O operations done involving the virtual memory.

This problem can be solved by:

- Not to execute I/O operations in user memory
- Locking the pages using a bit and locked pages cannot be replaced. However, since lock bit cannot be turned off and the frame remains blocked.

Filesystem Interface

- The file is a named collection of related information that is recorded on secondary storage.
- The file system is a collection of files and a directory structure that organizes the file information.
- Users consider files as the smallest logical units of the operating system.
- The files may range from source code or text or sequence of lines or bytes or numeric data

Filesystem Interface

File is said to be:

- Text file, if it is a sequence of lines or pages
- Source file, if it contains subroutines and functions along with code
- Object file, if it contains sequences understood by system linker
- Executable file, if it can be used by the loader to execute
- Multimedia file, if it includes audio or video
- Print preview file, which is in the format for printing as ASCII code
- Library files, containing routines which can be loaded
- Batch file, if it contains commands to the command interpreter

File attributes

Attribute	Its correspondence
Name	Human readable form of name
Identifies	Unique number which the system uses to identify the file
Type	Reflects the type of file
Location	Pointer which locates the file
Size	Size in bytes/words/blocks
Protection	Access information (read/write/execution access)
Time-date and user identification	Time stamp with data about modification

File Operations

- Creating a file: Space is allotted and file entry is made in the directory.
- Writing a file: Write pointer is used to write contents in the file .
- Reading a file: Name of the file and read pointer are used .
- Repositioning within a file: Current file position pointer is shifted to another location within the file (file seek).
- Deleting a file: Releasing of file space, deleting the file entry in the directory .
- Truncating a file: Deleting the contents of the file while retaining its attributes .

File Table

- **Open file table** contains all the information about the currently open files. Information associated with open file is:
- **File pointer**: Usage of file pointer overrides the need to specify the offset in read() and write()
- **File open count**: File open counter monitors the number of opens and closes, and when the count reaches zero, the entry is removed from the open file table
- **Disk location of the file**: The disk location is needed to perform operations on the file .
- **Access rights**: The access mode specifies the type of operations allowed on file

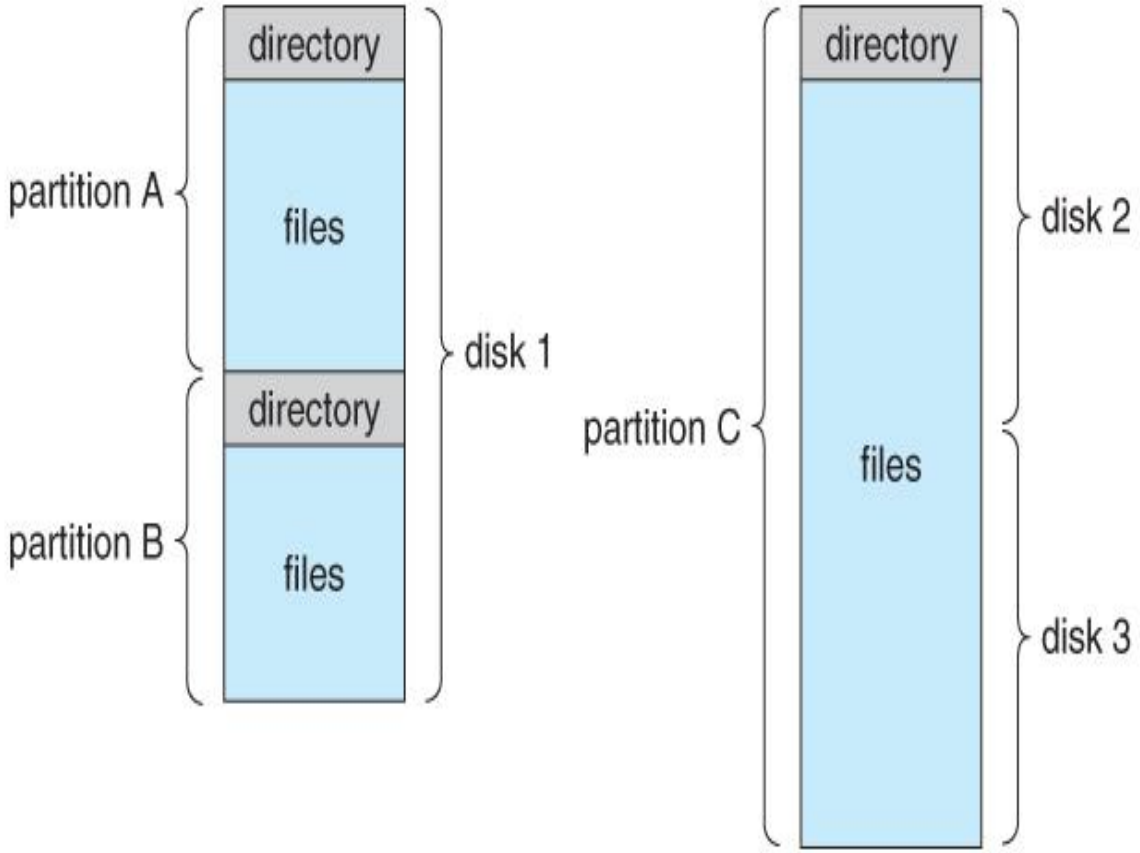
File Access Methods

Files need to be accessed in order to perform operations. Various access methods are as follows:

- **Sequential access:** This mode of access is the most frequent mode and simplest, accessing ***one record after the other sequentially*** i.e. when a read operation is performed; the file pointer automatically shifts to the next record.
- **Direct access:** This method of access is also called as **relative access** where records can be accessed randomly in no specific order. Index relative to the beginning of the file called relative block number is made use of to point to a particular record.
- **Other access methods:** Index for the file is created and records are reached using the index. For large files, primary index file stores entries of secondary index files, which in turn leads to the record.

Directory Structure

- The directory structure is used to store the file system. It can be divided into minidisks, slices or partitions.
- The combination of large partitions lead to volumes and the files in the volume is found out using the volume table of contents or device directory.



Directory Structure

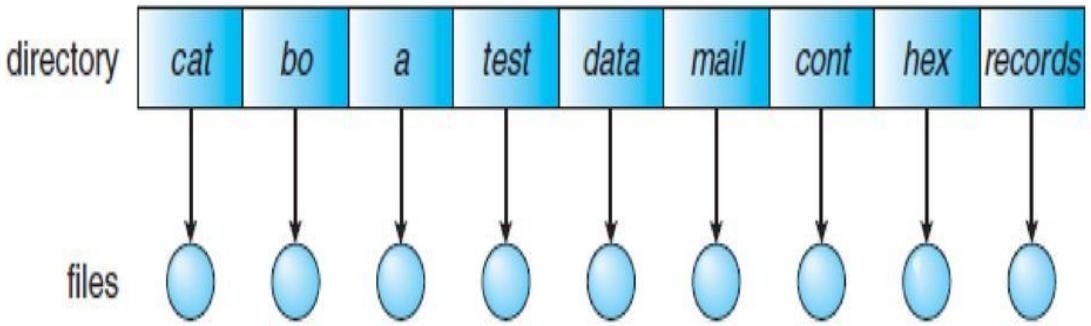
The operations that can be done in the directory are:

- Searching for a file: Searching the directory for a particular file.
- Creating a file: Creating new files and adding them to the directory
- Deleting a file: Deleting files from the directory
- List of Directory: Listing the files in the directory
- Renaming a file: Renaming the file in the directory even if the file is in use and also repositioning the file in the directory
- Traversing the file system: The contents and structure of the file system is saved at regular intervals making it easier for traversing or browsing through the contents of the directory

Types of Directories

- **Single level directory**: A simple directory structure where all the files are saved in one and only directory in the system (all files are contained in the same directory)
- This directory is organized at the same level and can be easily accessed. However, when there are multiple users, it is difficult to manage the directory as there are several files in the same directory.
- **Two level directory**: To overcome the problems faced in the single level directory, the two level directory assigns a separate directory to every user. The outer directory is named the MFD or master file directory, indexed by user name or account number, which points to the nested directory called User file directory (UFD) which is unique to each user.
- When a file is searched in the system, the MFD is searched for the user file directory. Once the user file directory is found, then the target file is searched.

Single Level Directory



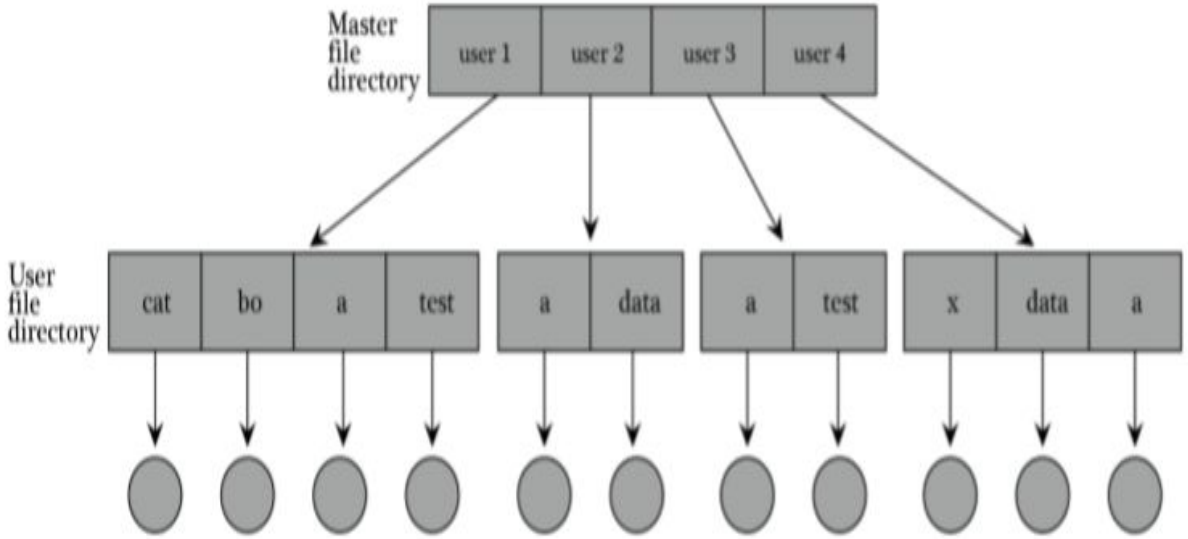


Figure 3.2.1: Two-Level Directory

Tree-Structured Directory

- **Tree-structured directory**: Directories are organized using a tree structure, which gives an access to each file using the unique path name.
- Each process contains a current directory which contains all the files that are accessed by the process. If the current directory does not contain the file, then a system call is made with the target directory name as a parameter.

The types of path names are:

- **Absolute path name**: This path name is complete in nature, starting from the root directory and ends at the targeted file name. For example, root/spell/exec/srt/first.txt
- **Relative path name:** This path name is fabricated from the current directory. For example, if the current directory is root/spell then exec/srt/first is the relative path name.

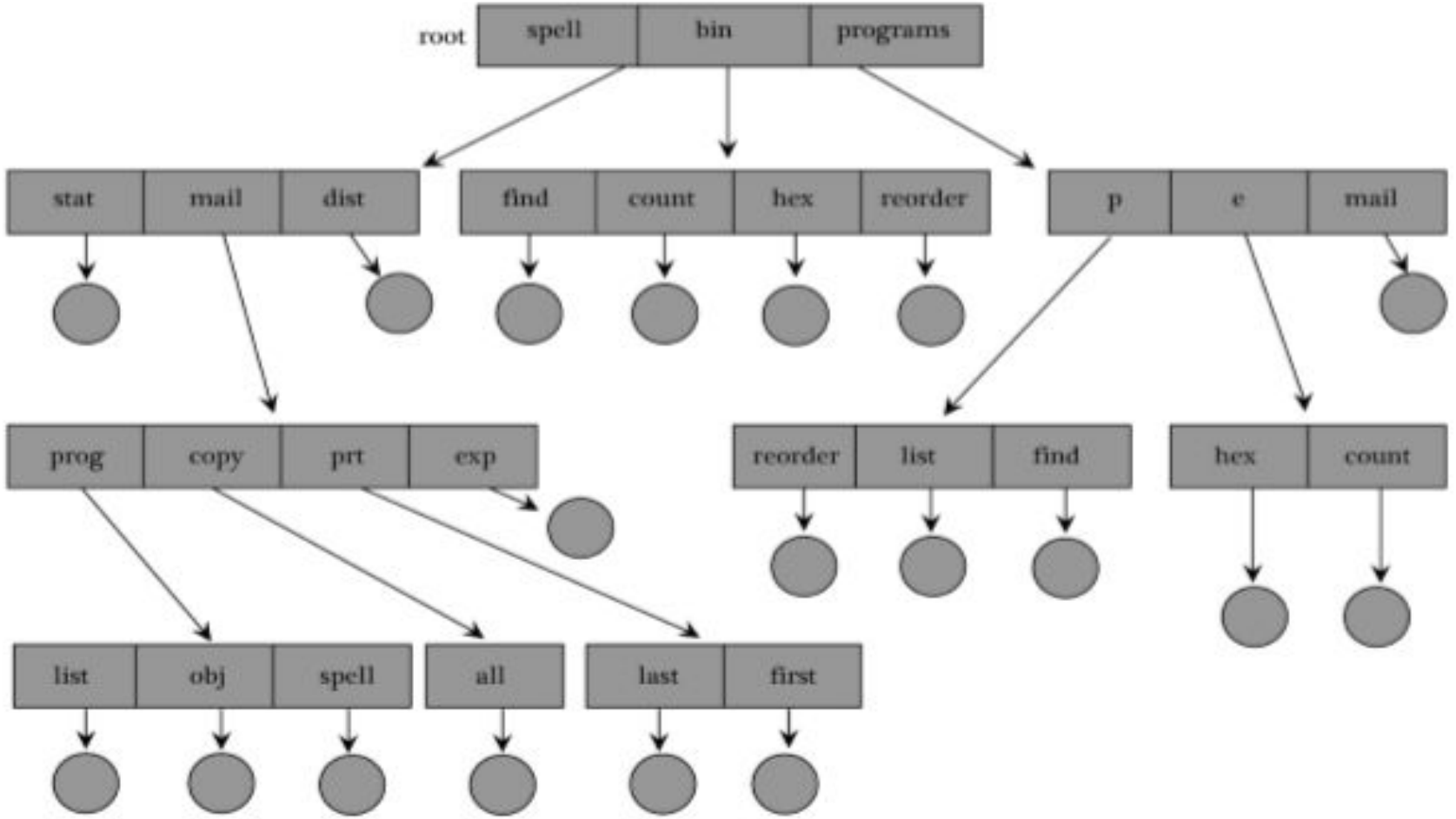


Figure 3.2.2: Tree-Structured Directories

Acyclic Graph Directory

- When two processes share a file space which is isolated from other processes, the common subdirectory is shared and an acyclic graph directory is used.
- The same subdirectory can be a part of two directories.
- Acyclic graph directory allows sharing of subdirectories and files among the processes

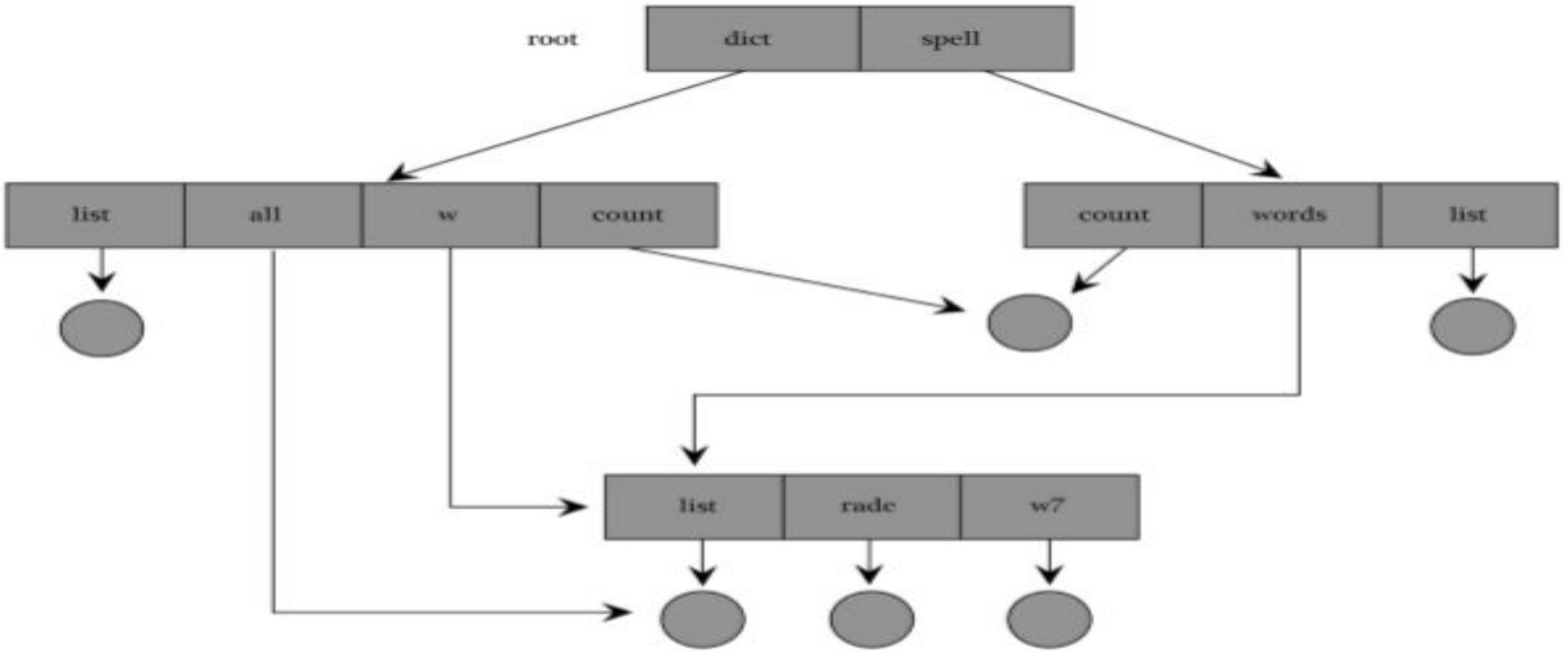


Figure 3.2.3: Acyclic-Graph Directories

File-System Mounting

- To make the filesystem available to the processes, they must be mounted on the system.
- The multiple volumes that constitute the directory are made available by mounting the multiple volumes and making the volumes present within the file system name space.

Procedure to mount:

- To mount a file, parameters like the name of the file and the mount point where the file system is to be attached are sent to the Operating system.
- The mount point is initially an empty directory.
- The device is checked for the validity of the file system and the format of the directory.

File Sharing

- When two or more users work towards the same goal, file sharing for the users become easy by reducing the efforts required.
- When multiple users share the files, protection mechanisms for the files are to be devised.
- The system can protect the files by allowing only read access to the files of other users or by giving a special grant access to the shared files.
- In UNIX system, file sharing is enabled by granting all the permissions on the files to the owner, while granting one set of operations on the file to the certain users in the file group and another set of operations on the file to all the other users.