UNIT-III

Memory Management: Basic concept, Logical and Physical address map, Memory allocation: Contiguous Memory allocation – Fixed and variable partition–Internal and External fragmentation and Compaction; Paging: Principle of operation – Page allocation – Hardware support for paging, protection and sharing, Disadvantages of paging.

Virtual Memory: Basics of Virtual Memory – Hardware and control structures – Locality of reference, Page fault , Working Set , Dirty page/Dirty bit – Demand paging, Page Replacement algorithms: Optimal, First in First Out (FIFO), Second Chance (SC), Not recently used (NRU) and Least Recently used (LRU).

Logical And Physical Addresses

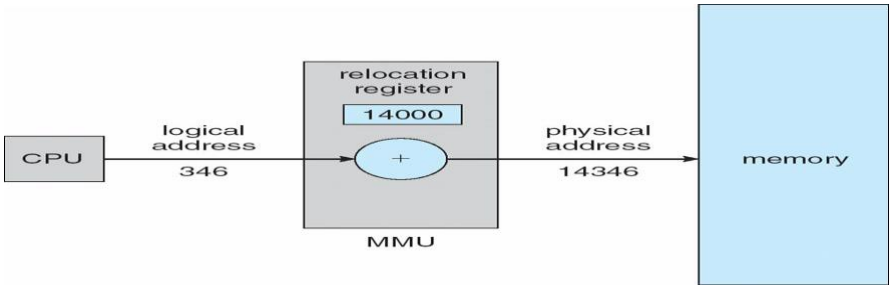
An address generated by the CPU is commonly refereed as **Logical Address**, whereas the address seen by the memory unit that is one loaded into the memory address register of the memory is commonly refereed as the **Physical Address**. The compile time and load time address binding generates the identical **logical and physical addresses**. However, the execution time addresses binding scheme results in differing **logical and physical addresses**.

The set of all **logical addresses** generated by a program is known as **Logical Address Space**, where as the set of all **physical addresses** corresponding to these logical addresses is **Physical Address Space**. Now, the run time mapping from virtual address to **physical address** is done by a hardware device known as **Memory Management Unit**. Here in the case of mapping the base register is known as **relocation register**. The value in the relocation register is added to the address generated by a user process at the time it is sent to memory .Let's understand this situation with the help of example: If the base register contains the value 1000,then an attempt by the user to address location 0 is dynamically relocated to location 1000,an access to location 346 is mapped to location 1346.

Memory-Management Unit (MMU)

Hardware device that maps virtual to physical address

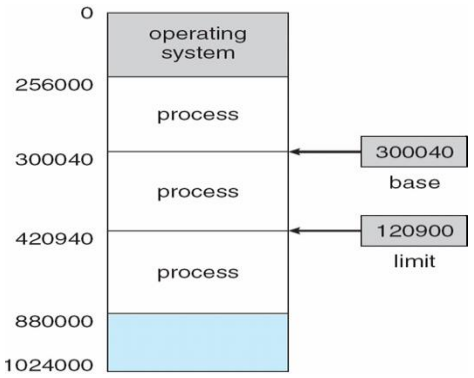
-
- In MMU scheme, the value in the relocation register is added to every address generated by a user process at the time it is sent to memory
- The user program deals with *logical* addresses; it never sees the *real* physical addresses



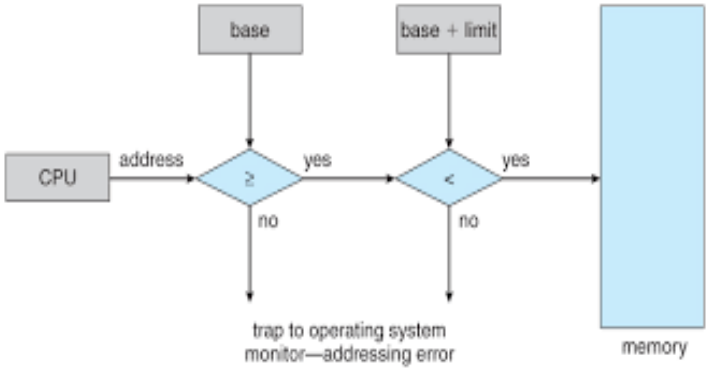
The user program never sees the **real physical address** space, it always deals with the **Logical addresses**. As we have two different type of addresses **Logical address** in the range (0 to max) and **Physical addresses** in the range(R to R+max) where R is the value of relocation register. The user generates only **logical addresses** and thinks that the process runs in location to 0 to max. As it is clear from the above text that user program supplies only logical addresses, these **logical addresses** must be mapped to **physical address** before they are used.

Base and Limit Registers

A pair of **base** and **limit** registers define the logical address space



HARDWARE PROTECTION WITH BASE AND LIMIT



Binding of Instructions and Data to Memory

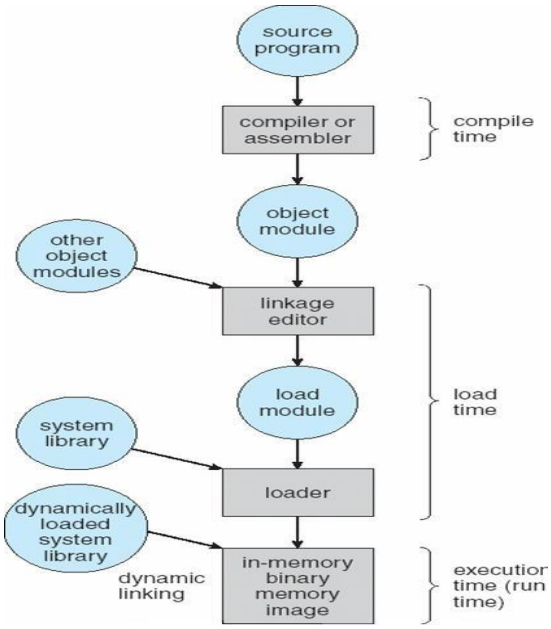
Address binding of instructions and data to memory addresses can happen at three different stages

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

Load time: Must generate **relocatable code** if memory location is not known at compile time

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

Multistep Processing of a User Program



Dynamic Loading

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

Dynamic Linking

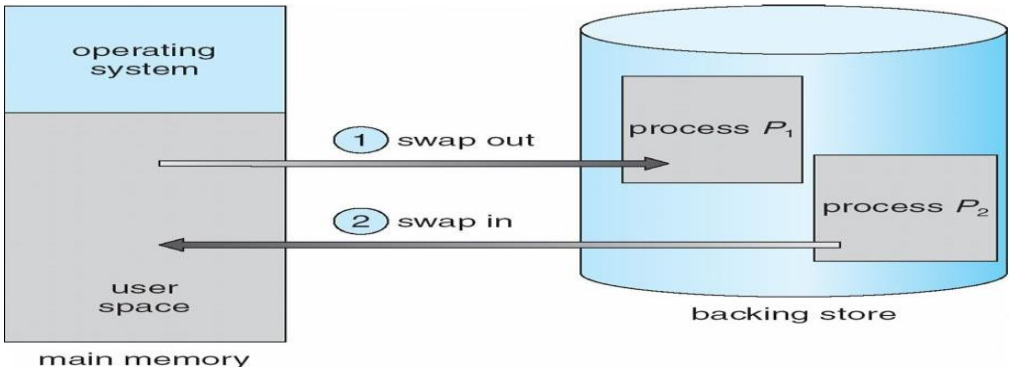
- Linking postponed until execution time
- Small piece of code, *stub*, used to locate the appropriate memory-resident library
- routine Stub replaces itself with the address of the routine, and executes the routine
- Operating system needed to check if routine is in processes' memory address Dynamic
- linking is particularly useful for libraries
- System also known as **shared libraries**

Swapping

A process can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution **Backing store** – fast disk large enough to accommodate copies of all memory images for all users; must provide direct access to these memory images **Roll out, roll in** – swapping variant used for priority-based scheduling algorithms; lower-priority process is swapped out so higher-priority process can be loaded and executed Major part of swap time is transfer time; total transfer time is directly proportional to the amount of memory swapped and Modified versions of swapping are found on many systems (i.e., UNIX, Linux, and Windows)

System maintains a **ready queue** of ready-to-run processes which have memory images on disk

Schematic View of Swapping



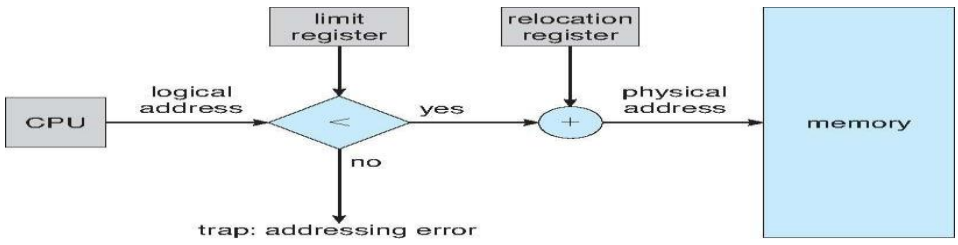
Contiguous Allocation

- Main memory usually into two partitions:
- Resident operating system, usually held in low memory with interrupt vector
- User processes then held in high memorynRelocation registers used to protect user processes from each other, and from changing operating-system code and data

Base register contains value of smallest physical address

- Limit register contains range of logical addresses – each logical address must be less than the limit register
- MMU maps logical address *dynamically*

Hardware Support for Relocation and Limit Registers



- Multiple-partition allocation
- Hole – block of available memory; holes of various size are scattered throughout memory
- When a process arrives, it is allocated memory from a hole large enough to accommodate it

Contiguous memory allocation is one of the efficient ways of allocating main memory to the processes. The memory is divided into two partitions. One for the Operating System and another for the user processes. Operating System is placed in low or high memory depending on the interrupt vector placed. In contiguous memory allocation each process is contained in a single contiguous section of memory.

Memory protection

Memory protection is required to protect Operating System from the user processes and user processes from one another. A relocation register contains the value of the smallest physical address for example say 100040. The limit register contains the range of logical address for example say 74600. Each logical address must be less than limit register. If a logical address is greater than the limit register, then there is an addressing error and it is trapped. The limit register hence offers memory protection.

The MMU, that is, Memory Management Unit maps the logical address dynamically, that is at run time, by adding the logical address to the value in relocation register. This added value is the physical memory address which is sent to the memory.

The CPU scheduler selects a process for execution and a dispatcher loads the limit and relocation registers with correct values. The advantage of relocation register is that it provides an efficient way to allow the Operating System size to change dynamically.

Memory allocation

There are two methods namely, multiple partition method and a general fixed partition method. In multiple partition method, the memory is divided into several fixed size partitions. One process occupies each partition. This scheme is rarely used nowadays. Degree of multiprogramming depends on the number of partitions. Degree of multiprogramming is the number of programs that are in the main memory. The CPU is never left idle in multiprogramming. This was used by IBM OS/360 called MFT. MFT stands for Multiprogramming with a Fixed number of Tasks.

Generalization of fixed partition scheme is used in MVT. MVT stands for Multiprogramming with a Variable number of Tasks. The Operating System keeps track of which parts of memory are available and which is occupied. This is done with the help of a table that is maintained by the Operating System. Initially the whole of the available memory is treated as

one large block of memory called a **hole**. The programs that enter a system are maintained in an input queue. From the hole, blocks of main memory are allocated to the programs in the input queue. If the hole is large, then it is split into two, and one half is allocated to the arriving process and the other half is returned. As and when memory is allocated, a set of holes in scattered. If holes are adjacent, they can be merged.

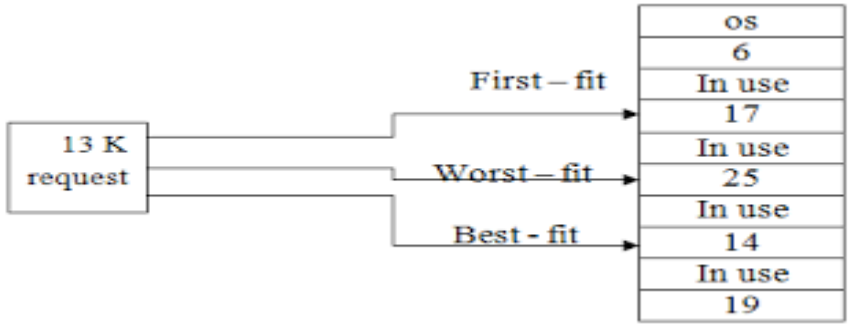
Now there comes a general dynamic storage allocation problem. The following are the solutions to the dynamic storage allocation problem.

- First fit: The first hole that is large enough is allocated. Searching for the holes starts from the beginning of the set of holes or from where the previous first fit search ended.

- Best fit: The smallest hole that is big enough to accommodate the incoming process is allocated. If the available holes are ordered, then the searching can be reduced.

- Worst fit: The largest of the available holes is allocated.

Example:



First and best fits decrease time and storage utilization. First fit is generally faster.

Fragmentation

The disadvantage of contiguous memory allocation is **fragmentation**. There are two types of fragmentation, namely, internal fragmentation and External fragmentation.

Internal fragmentation

When memory is free internally, that is inside a process but it cannot be used, we call that fragment as internal fragment. For example say a hole of size 18464 bytes is available. Let the size of the process be 18462. If the hole is allocated to this process, then two bytes are left which is not used. These two bytes which cannot be used forms the internal fragmentation. The worst part of it is that the overhead to maintain these two bytes is more than two bytes.

External fragmentation

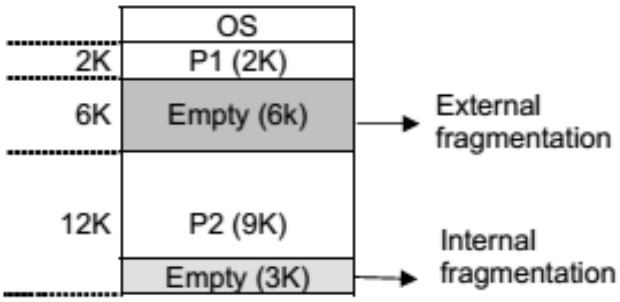
All the three dynamic storage allocation methods discussed above suffer external fragmentation. When the total memory space that is got by adding the scattered holes is sufficient to satisfy a request but it is not available contiguously, then this type of

fragmentation is called external fragmentation.

The solution to this kind of external fragmentation is compaction. **Compaction** is a method by which all free memory that are scattered are placed together in one large memory block. It is to be noted that compaction cannot be done if relocation is done at compile time or assembly time. It is possible only if dynamic relocation is done, that is relocation at execution time.

One more solution to external fragmentation is to have the logical address space and physical address space to be non contiguous. Paging and Segmentation are popular non contiguous allocation methods.

Example for internal and external fragmentation



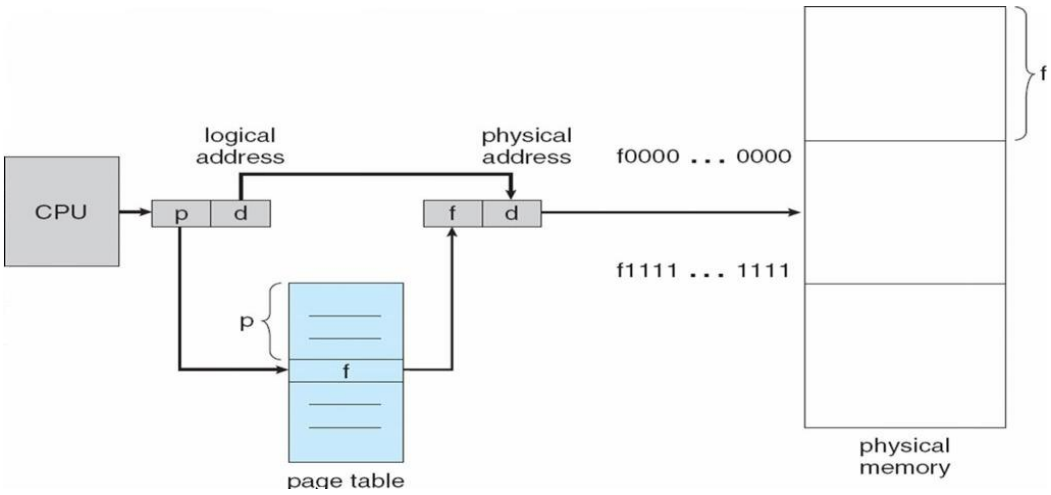
Paging

A computer can address more memory than the amount physically installed on the system. This extra memory is actually called virtual memory and it is a section of a hard that's set up to emulate the computer's RAM. Paging technique plays an important role in implementing virtual memory.

Paging is a memory management technique in which process address space is broken into blocks of the same size called **pages** (size is power of 2, between 512 bytes and 8192 bytes). The size of the process is measured in the number of pages.

Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.

Paging Hardware



Address Translation

Page address is called **logical address** and represented by **page number** and the **offset**.

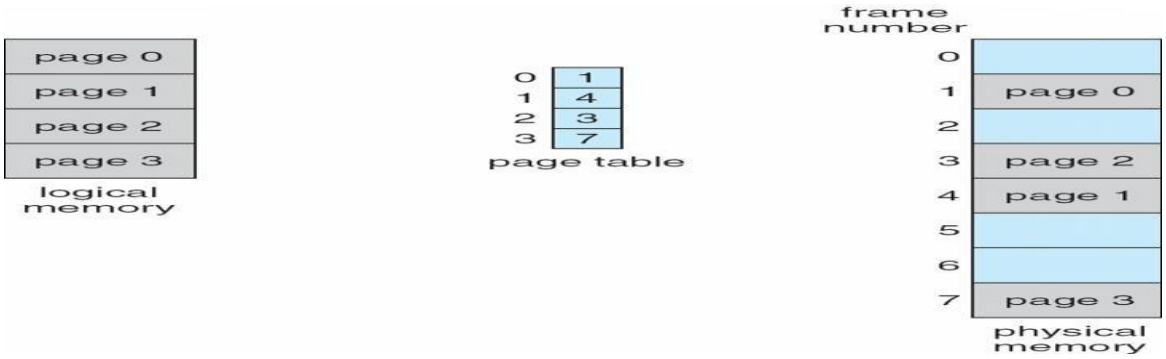
```
Logical Address = Page number + page offset
```

Frame address is called **physical address** and represented by a **frame number** and the **offset**.

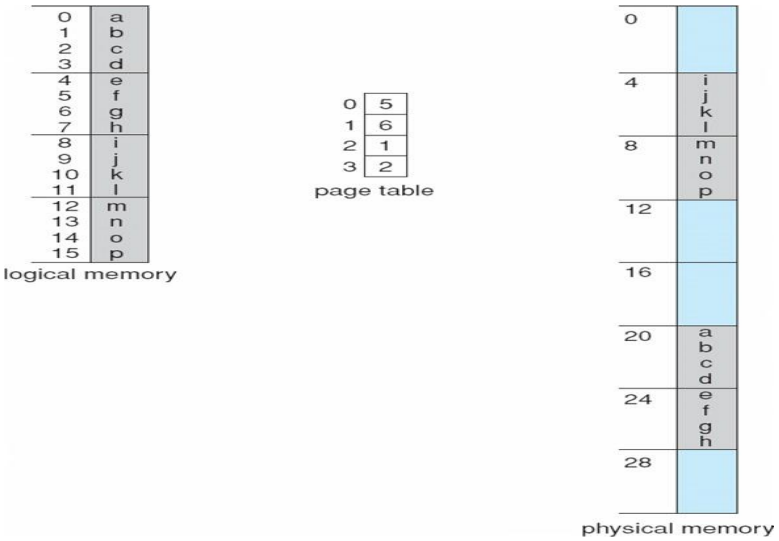
```
Physical Address = Frame number + page offset
```

A data structure called **page map table** is used to keep track of the relation between a page of a process to a frame in physical memory.

Paging Model of Logical and Physical Memory

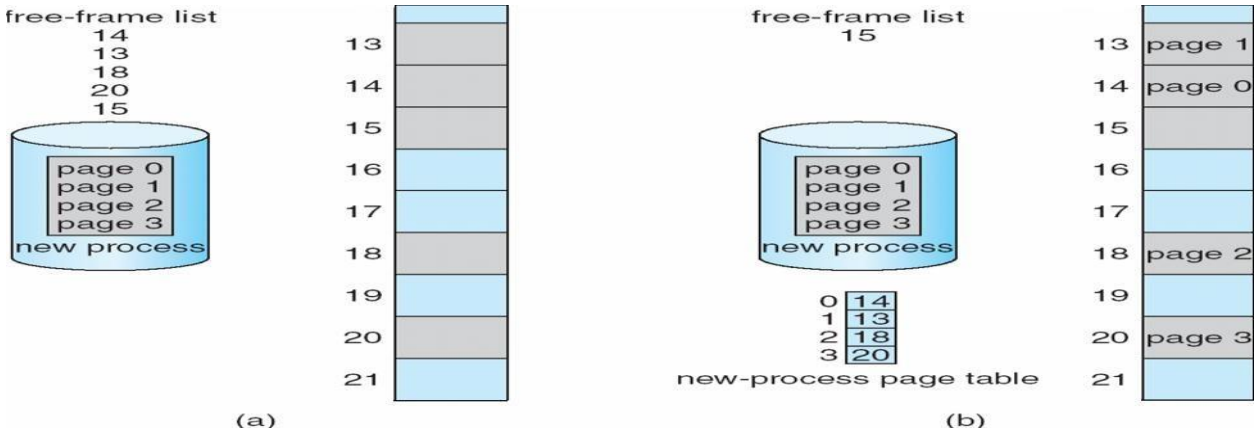


Paging Example



32-byte memory and 4-byte pages

Free Frames



When the system allocates a frame to any page, it translates this logical address into a physical address and create entry into the page table to be used throughout execution of the program.

When a process is to be executed, its corresponding pages are loaded into any available memory frames. Suppose you have a program of 8Kb but your memory can accommodate only 5Kb at a given point in time, then the paging concept will come into picture. When a

computer runs out of RAM, the operating system (OS) will move idle or unwanted pages of memory to secondary memory to free up RAM for other processes and brings them back when needed by the program.

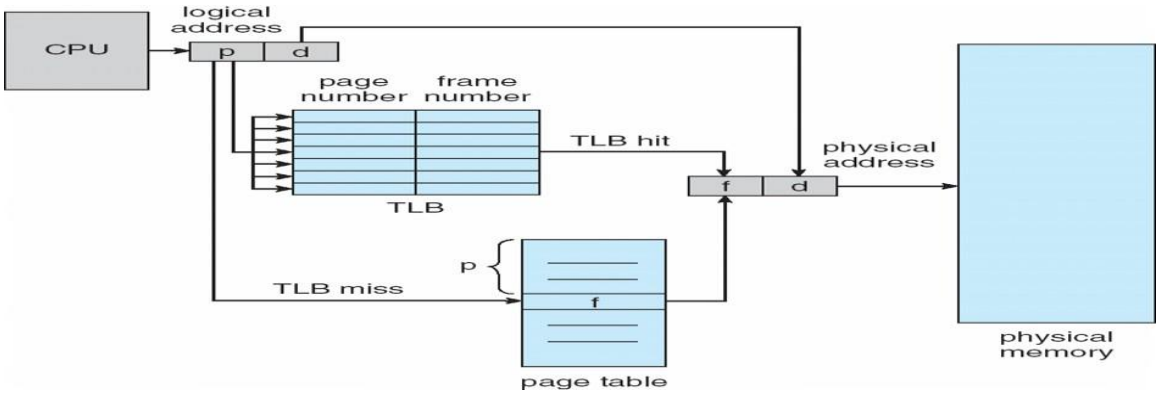
This process continues during the whole execution of the program where the OS keeps removing idle pages from the main memory and write them onto the secondary memory and bring them back when required by the program.

Implementation of Page Table

- Page table is kept in main memory
- **Page-table base register (PTBR)** points to the page table
- **Page-table length register (PRLR)** indicates size of the page table
- In this scheme every data/instruction access requires two memory accesses. One for the page table and one for the data/instruction.

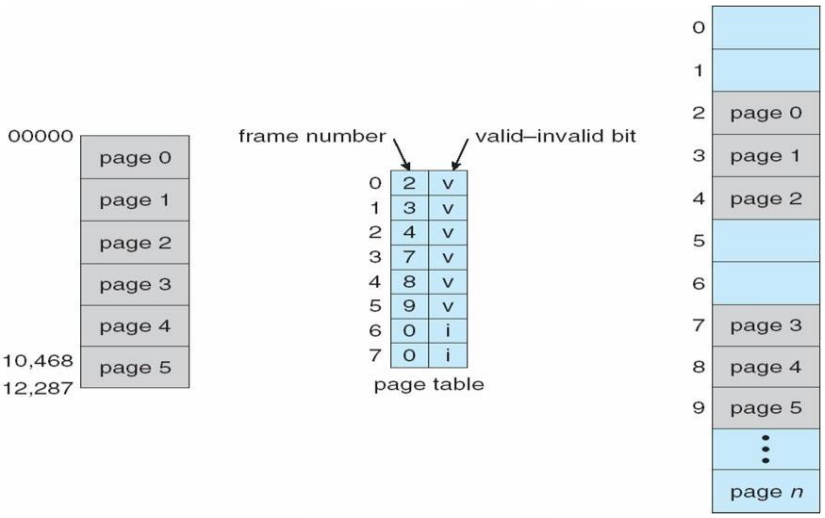
The two memory access problem can be solved by the use of a special fast-lookup hardware cache called **associative memory** or **translation look-aside buffers (TLBs)**

Paging Hardware With TLB



Memory Protection

- Memory protection implemented by associating protection bit with each frame
- **Valid-invalid** bit attached to each entry in the page table:
- "valid" indicates that the associated page is in the process' logical address space, and is thus a legal
- page "invalid" indicates that the page is not in the process' logical address space
- Valid (v) or Invalid (i) Bit In A Page Table



Shared Pages

Shared code

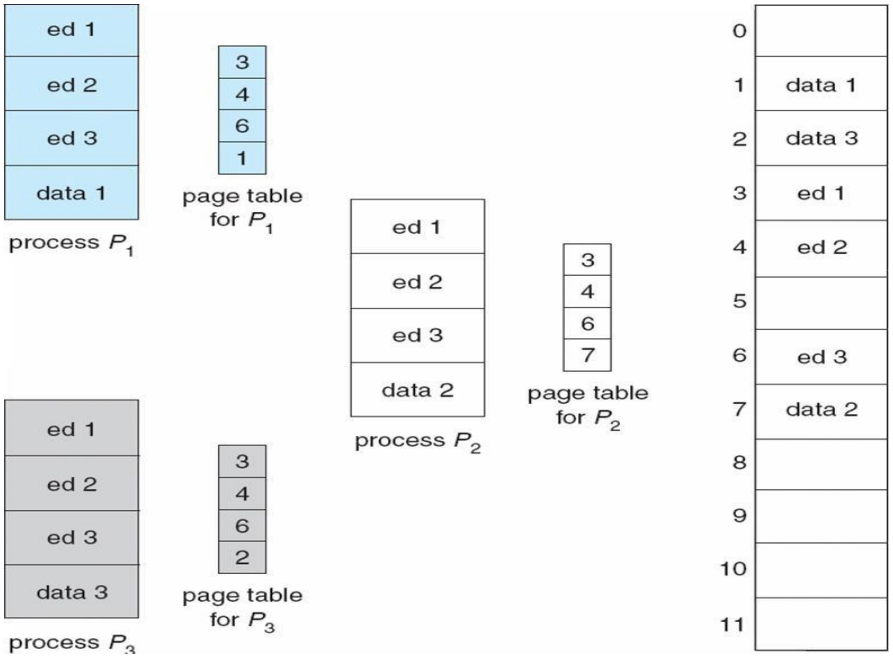
- One copy of read-only (reentrant) code shared among processes (i.e., text editors, compilers, window systems).
- Shared code must appear in same location in the logical address space of all processes

Private code and data

Each process keeps a separate copy of the code and data

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

Shared Pages Example



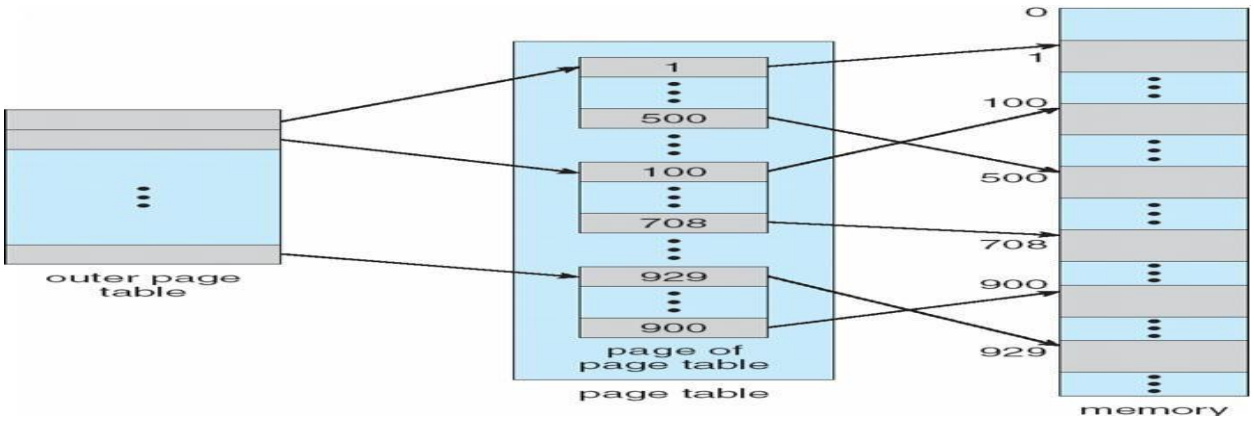
Structure of the Page Table

- Hierarchical Paging
- Hashed Page Tables
- Inverted Page Tables

Hierarchical Page Tables

Break up the logical address space into multiple page tables A simple technique is a two-level page table

Two-Level Page-Table Scheme



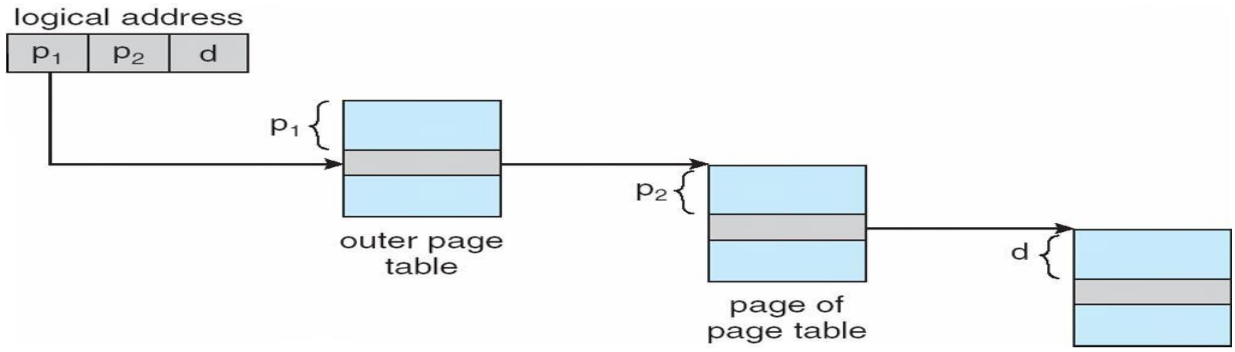
Two-Level Paging Example

- A logical address (on 32-bit machine with 1K page size) is divided
- into: a page number consisting of 22 bits
- a page offset consisting of 10 bits
- Since the page table is paged, the page number is further divided into:
- a 12-bit page number a 10-bit page offset
- Thus, a logical address is as follows:
-

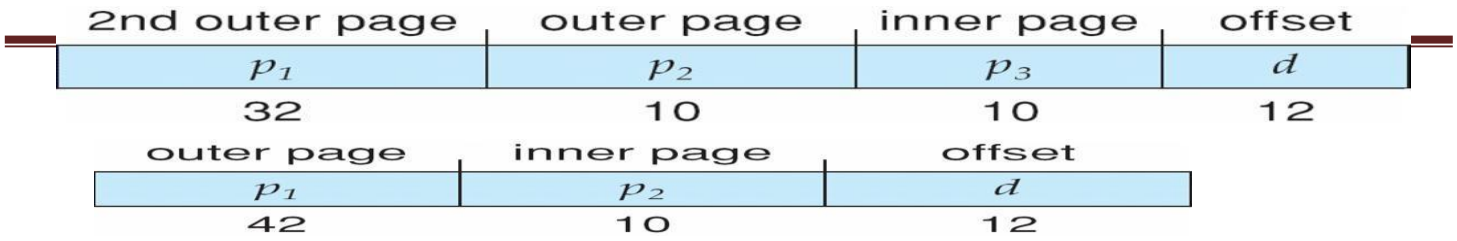
where p_i is an index into the outer page table, and p_2 is the displacement within the page of the outer page table

Page number		page offset
p_{12}	p_{10}	d_{10}

Address-Translation Scheme



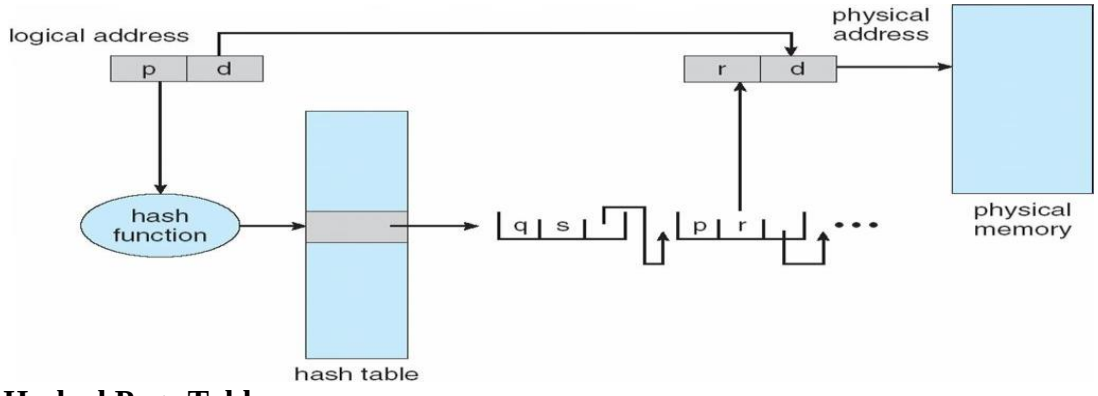
Three-level Paging Scheme



Hashed Page Tables

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

If a match is found, the corresponding physical frame is extracted



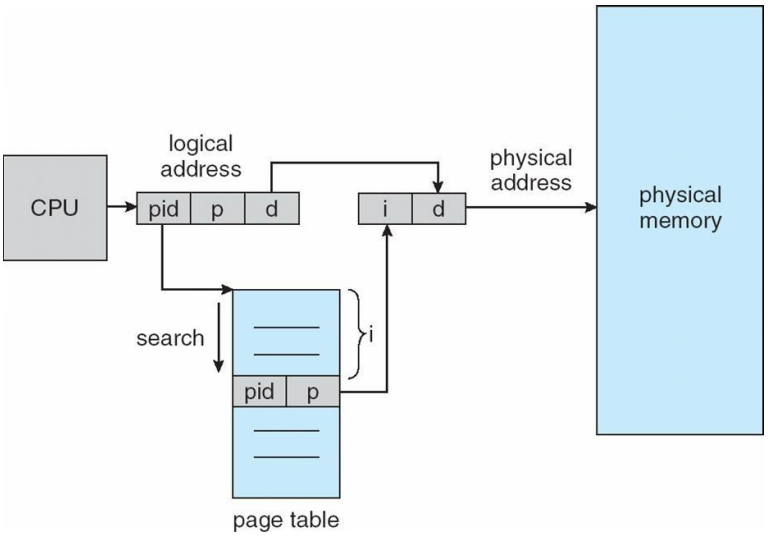
Hashed Page Table

Inverted Page Table

One entry for each real page of memory

- Entry consists of the virtual address of the page stored in that real memory location, with information about the process that owns that page
- Decreases memory needed to store each page table, but increases time needed to search the table when a page reference occurs
- Use hash table to limit the search to one — or at most a few — page-table entries

Inverted Page Table Architecture



Advantages and Disadvantages of Paging

Here is a list of advantages and disadvantages of paging −

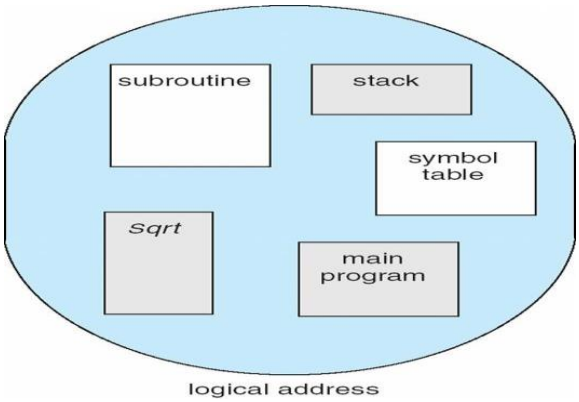
- Paging reduces external fragmentation, but still suffers from internal fragmentation.
- Paging is simple to implement and assumed as an efficient memory management technique.
- Due to equal size of the pages and frames, swapping becomes very easy.
- Page table requires extra memory space, so may not be good for a system having small RAM.

Segmentation

- Memory-management scheme that supports user view of memory A program is a collection of segments

- A segment is a logical unit such as:
- main program
- Procedure
- function method
- object

- local variables, global variables
- common block
- stack
- symbol table
- arrays



User's View of a Program

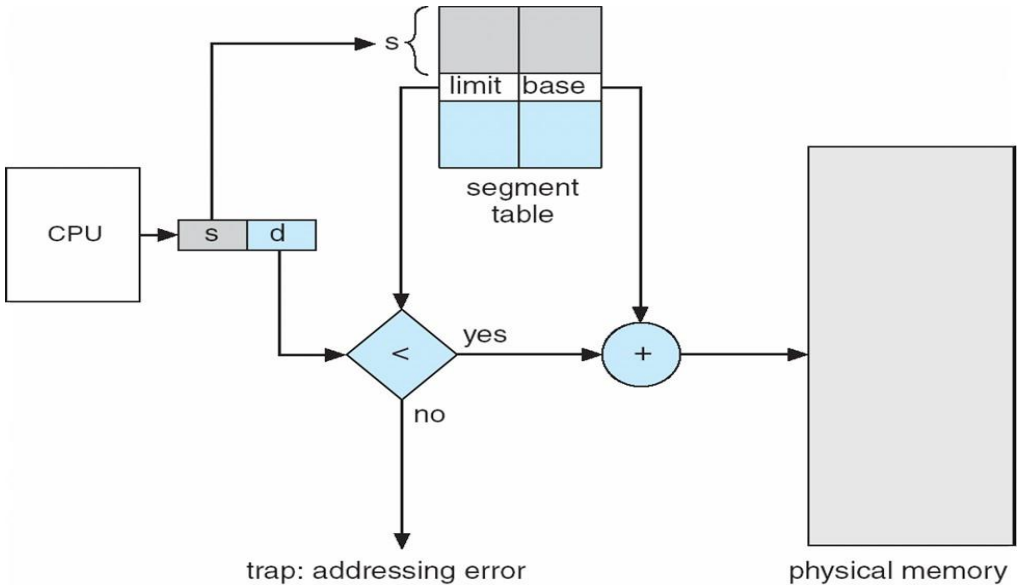
Segmentation Architecture

- Logical address consists of a two tuple:

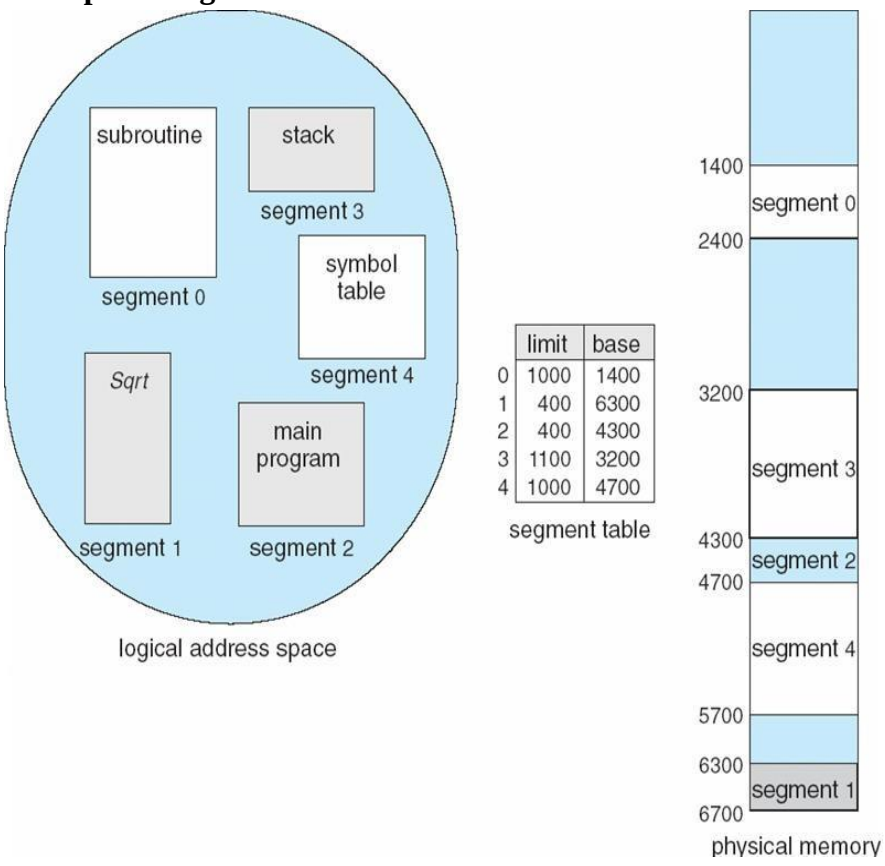
o <segment-number, offset>,

- **Segment table** – maps two-dimensional physical addresses; each table entry has:space
- **base** – contains the starting physical address where the segments reside in memory
- **limit** – specifies the length of the segment
- **Segment-table base register (STBR)** points to the segment table's location in memory
- **Segment-table length register (STLR)** indicates number of segments used by a program; segment number ***s*** is legal if ***s*** < **STLR**
- Protection
- With each entry in segment table associate:
- validation bit = 0 Þ illegal segment
- read/write/execute privileges
- Protection bits associated with segments; code sharing occurs at segment level
- Since segments vary in length, memory allocation is a dynamic storage-allocation
- problem A segmentation example is shown in the following diagram

Segmentation Hardware



Example of Segmentation



Segmentation with paging

Instead of an actual memory location the segment information includes the address of a page table for the segment. When a program references a memory location the offset is translated to a memory address using the page table. A segment can be extended simply by allocating another memory page and adding it to the segment's page table.
An implementation of virtual memory on a system using segmentation with paging usually only moves individual pages back and forth between main memory and secondary storage, similar to a paged non-segmented system. Pages of the segment can be located anywhere in main memory and need not be contiguous. This usually results in a reduced amount of input/output between primary and secondary storage and reduced memory fragmentation.

Virtual Memory

Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.
In real scenarios, most processes never need all their pages at once, for following reasons :

- Error handling code is not needed unless that specific error occurs, some of which are quite rare.
- Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
- Certain features of certain programs are rarely used.

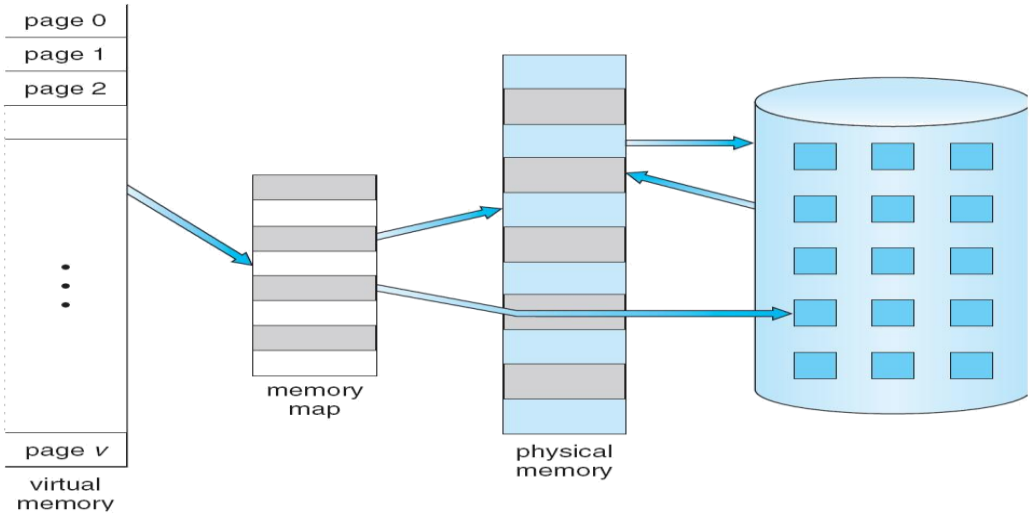


Fig. Diagram showing virtual memory that is larger than physical memory.
Virtual memory is commonly implemented by demand paging. It can also be implemented in a segmentation system. Demand segmentation can also be used to provide virtual memory.

Benefits of having Virtual Memory :

1. Large programs can be written, as virtual space available is huge compared to physical memory.

2. Less I/O required, leads to faster and easy swapping of processes.
3. More physical memory available, as programs are stored on virtual memory, so they occupy very less space on actual physical memory.

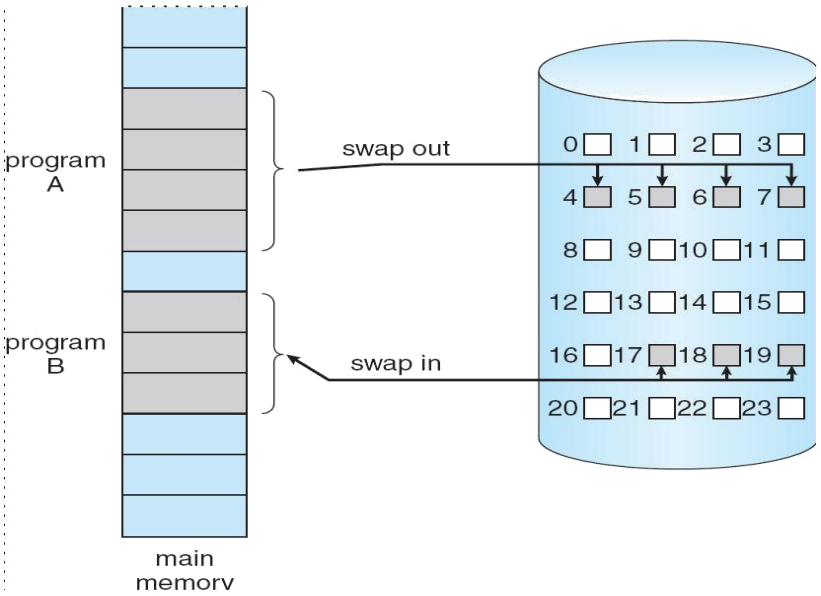
Demand Paging

A demand paging is similar to a paging system with swapping(Fig 5.2). When we want to execute a process, we swap it into memory. Rather than swapping the entire process into 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 in anyway, decreasing the swap time and the amount of physical memory needed.

Hardware support is required to distinguish between those pages that are in memory and those pages that are on the disk using the valid-invalid bit scheme. Where valid and invalid pages can be checked checking the bit and marking a page will have no effect if the process never attempts to access the pages. While the process executes and accesses pages that are memory resident, execution proceeds normally.

Fig. Transfer of a paged memory to continuous disk space



Access to a page marked invalid causes a page-fault trap. This trap is the result of the operating system's failure to bring the desired page into memory.

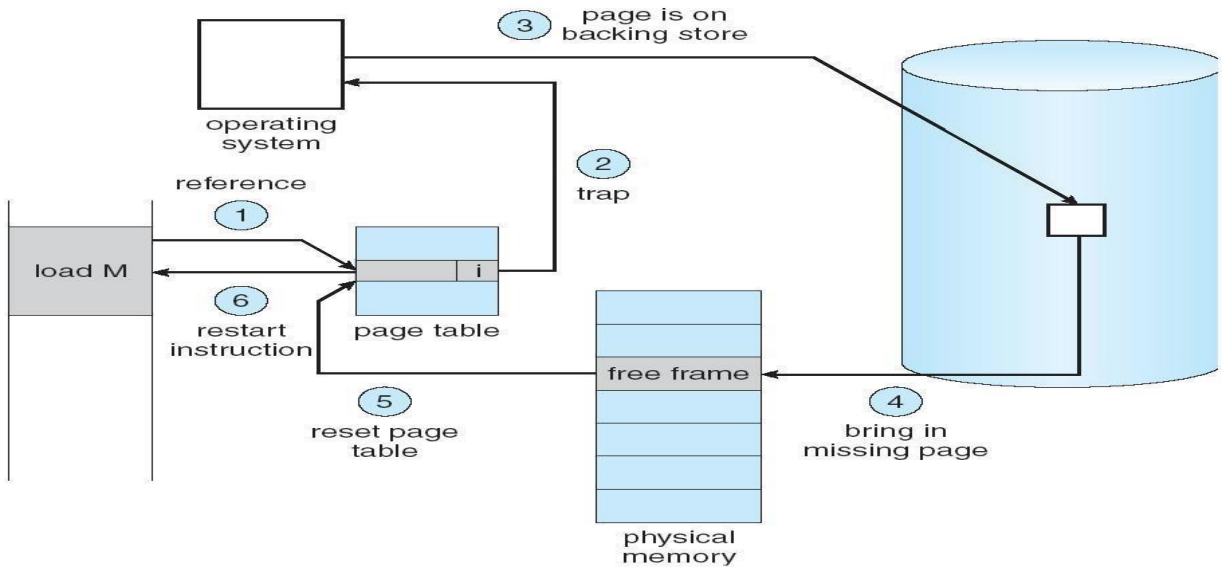
Initially only those pages are loaded which will be required the process immediately.
The pages that are not moved into the memory are marked as invalid in the page table. For

an invalid entry the rest of the table is empty. In case of pages that are loaded in the memory, they are marked as valid along with the information about where to find the swapped out page.

When the process requires any of the page that is not loaded into the memory, a page fault trap is triggered and following steps are followed,

1. The memory address which is requested by the process is first checked, to verify the request made by the process.
2. If its found to be invalid, the process is terminated.
3. In case the request by the process is valid, a free frame is located, possibly from a free-frame list, where the required page will be moved.
4. A new operation is scheduled to move the necessary page from disk to the specified memory location. (This will usually block the process on an I/O wait, allowing some other process to use the CPU in the meantime.)
5. When the I/O operation is complete, the process's page table is updated with the new frame number, and the invalid bit is changed to valid.

Fig. Steps in handling a page fault



6. The instruction that caused the page fault must now be restarted from the beginning.

There are cases when no pages are loaded into the memory initially, pages are only loaded when demanded by the process by generating page faults. This is called **Pure Demand Paging**.

The only major issue with Demand Paging is, after a new page is loaded, the process starts execution from the beginning. It is not a big issue for small programs, but for larger programs it affects performance drastically.

What is dirty bit?

When a bit is modified by the CPU and not written back to the storage, it is called as a dirty bit. This bit is present in the memory cache or the virtual storage space.

Advantages of Demand Paging:

1. Large virtual memory.
2. More efficient use of memory.
3. Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

Disadvantages of Demand Paging:

1. Number of tables and amount of processor over head for handling page interrupts are greater than in the case of the simple paged management techniques.
2. due to the lack of an explicit constraints on a jobs address space size.

Page Replacement

As studied in Demand Paging, only certain pages of a process are loaded initially into the memory. This allows us to get more number of processes into the memory at the same time. but what happens when a process requests for more pages and no free memory is available to bring them in. Following steps can be taken to deal with this problem :

1. Put the process in the wait queue, until any other process finishes its execution thereby freeing frames.
2. Or, remove some other process completely from the memory to free frames.
3. Or, find some pages that are not being used right now, move them to the disk to get free frames. This technique is called **Page replacement** and is most commonly used. We have some great algorithms to carry on page replacement efficiently.

Page Replacement Algorithm

Page replacement algorithms are the techniques using which an Operating System decides which memory pages to swap out, write to disk when a page of memory needs to be allocated. Paging happens whenever a page fault occurs and a free page cannot be used for allocation purpose accounting to reason that pages are not available or the number of free pages is lower than required pages.

When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

A page replacement algorithm looks at the limited information about accessing the pages provided by hardware, and tries to select which pages should be replaced to minimize the total number of page misses, while balancing it with the costs of primary storage and processor time of the algorithm itself. There are many different page replacement algorithms. We evaluate an algorithm by running it on a particular string of memory reference and computing the number of page faults,

Reference String

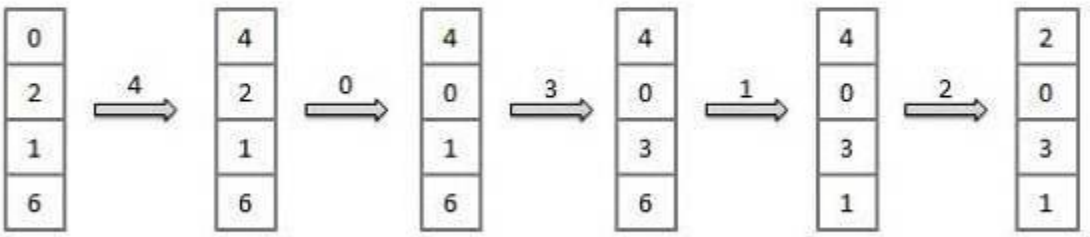
The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference.

The latter choice produces a large number of data, where we note two things.

- For a given page size, we need to consider only the page number, not the entire address.
- If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
- For example, consider the following sequence of addresses − 123,215,600,1234,76,96
- If page size is 100, then the reference string is 1,2,6,12,0,0 First In First Out (FIFO) algorithm
- Oldest page in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages from the tail and add new pages at the head.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x x



Fault Rate = 9 / 12 = 0.75

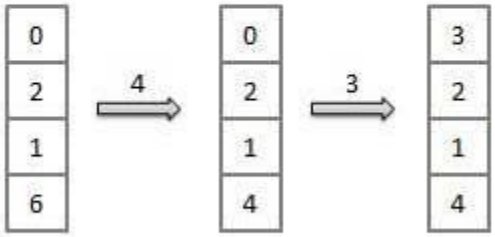
Optimal Page algorithm

- An optimal page-replacement algorithm has the lowest page-fault rate of all algorithms. An optimal page-replacement algorithm exists, and has been called OPT or MIN.

- Replace the page that will not be used for the longest period of time. Use the time when a page is to be used.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x



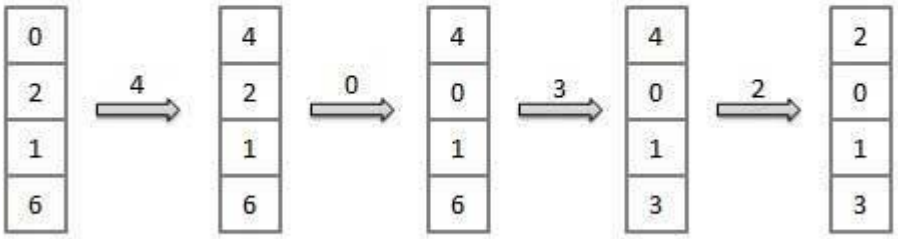
Fault Rate = 6 / 12 = 0.50

Least Recently Used (LRU) algorithm

- Page which has not been used for the longest time in main memory is the one which will be selected for replacement.
- Easy to implement, keep a list, replace pages by looking back into time.

Reference String : 0, 2, 1, 6, 4, 0, 1, 0, 3, 1, 2, 1

Misses : x x x x x x x x



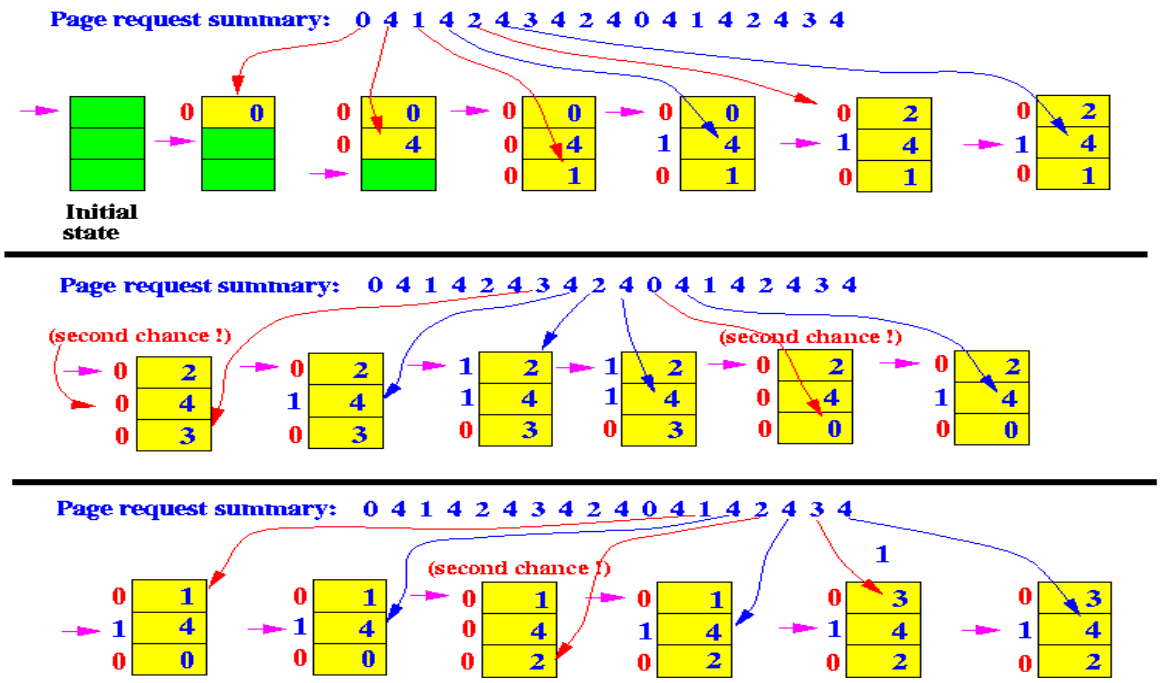
Fault Rate = 8 / 12 = 0.67

Second chance page replacement algorithm

- **Second Chance** replacement policy is called the **Clock** replacement policy...
- In the Second Chance page replacement policy, the candidate pages for removal are **consider** in a round robin matter, and a page that has been **accessed between consecutive considerations** will not be replaced.

The page replaced is the one that - considered in a round robin matter - has not been accessed since its last consideration.

- Implementation:
  - Add a "second chance" bit to each memory frame.
  - Each time a memory frame is referenced, set the "second chance" bit to ONE (1) - this will give the frame a second chance...
  - A new page read into a memory frame has the second chance bit set to ZERO (0)
  - When you need to find a page for removal, look in a round robin manner in the memory frames:
    - If the second chance bit is ONE, reset its second chance bit (to ZERO) and continue.
    - If the second chance bit is ZERO, replace the page in that memory frame.
- The following figure shows the behavior of the program in paging using the Second Chance page replacement policy:



  - We can see notably that the **bad** replacement decision made by FIFO is **not present** in Second chance!!!
  - There are a total of **9 page read operations** to satisfy the total of 18 page requests - just as good as the more computationally expensive LRU method !!!

NRU (Not Recently Used) Page Replacement Algorithm - This algorithm requires that each page have two additional status bits 'R' and 'M' called reference bit and change bit respectively. The reference bit(R) is automatically set to 1 whenever the page is referenced. The change bit (M) is set to 1 whenever the page is modified. These bits are stored in the PMT and are updated on every memory reference. When a page fault occurs, the memory manager inspects all the pages and divides them into 4 classes based on R and M bits.

- **Class 1: (0,0)** – neither recently used nor modified - the best page to replace.
- **Class 2: (0,1)** – not recently used but modified - the page will need to be written out before replacement.
- **Class 3: (1,0)** – recently used but clean - probably will be used again soon.
- **Class 4: (1,1)** – recently used and modified - probably will be used again, and write out will be needed before replacing it.

This algorithm removes a page at random from the lowest numbered non-empty class.

Thrashing

If the number of frames allocated to a low-priority process falls below the minimum number required by the computer architecture, we must suspend that process' execution. We should then page out its remaining pages, freeing all its allocated frames. This provision introduces a swap-in, swap-out level of intermediate CPU scheduling.

In fact, look at any process that does not have "enough" frames. Although it is technically possible to reduce the number of allocated frames to the minimum, there is some (larger) number of pages in active use. If the process does not have this number of frames, it will quickly page fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again. The process continues to fault, replacing pages for which it then faults and brings back in right away.

This high paging activity is called **thrashing**. A process is thrashing if it is spending more time paging than executing.