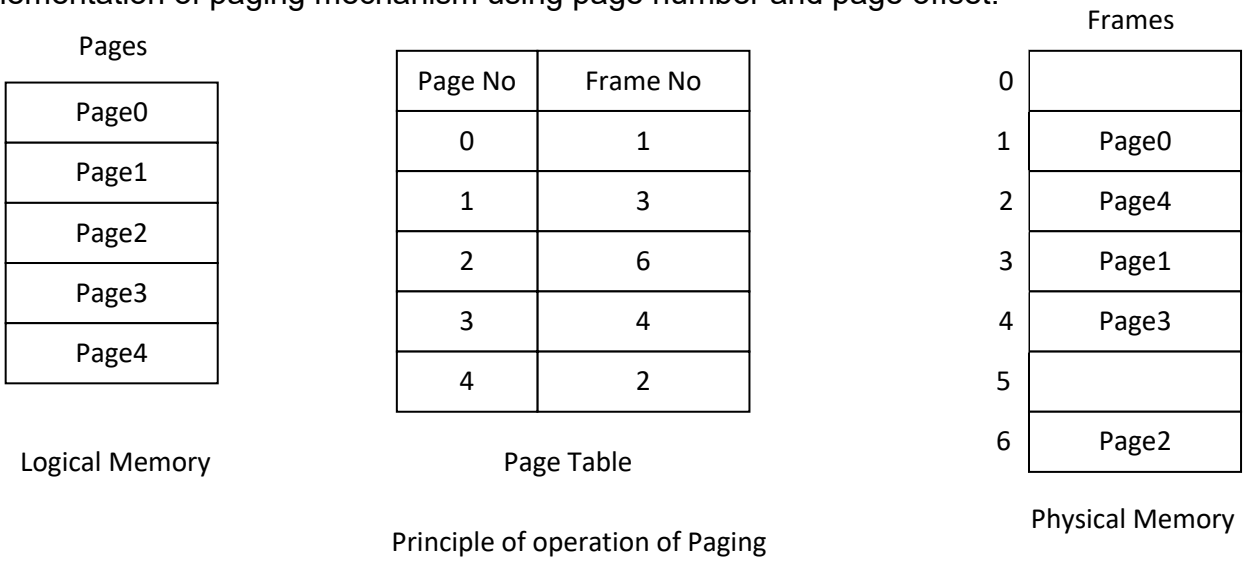
Memory Management – (cont…3)

Paging

Paging scheme solves the problem faced in variable sized partitions like external fragmentation. In a paged system, logical memory is divided into a number of fixed sized chunks called **Pages**. The physical memory is also pre-divided into same fixed sized blocks called **Frames**. The page size and the frame size are always in powers of 2, and vary between 512 bytes and 8192 bytes per page. The reason behind this is implementation of paging mechanism using page number and page offset.



Principle of operation of Paging

Each process page is allocated to some memory frame. These pages can be loaded into contiguous frames in memory or into noncontiguous frames.

Each time when a process of size n is to be loaded, it is important to know the best location from the list of available/free holes. This dynamic storage allocation is necessary to increase efficiency and throughput of system. Most commonly used strategies are:

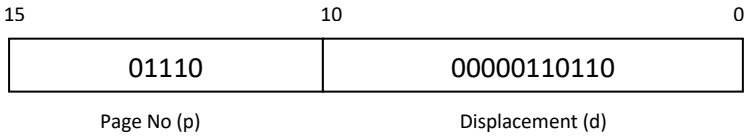
1. Best-fit Policy: Allocating the hole in which the process fits most tightly.
2. First-fit Policy: Allocating the first available hole, according to memory order, which is big enough to accommodate the new process.
3. Worst-fit Policy: Allocating the largest hole that will leave maximum amount of unused space.

Hardware support for paging

Every logical page in paging scheme is divided into two parts:

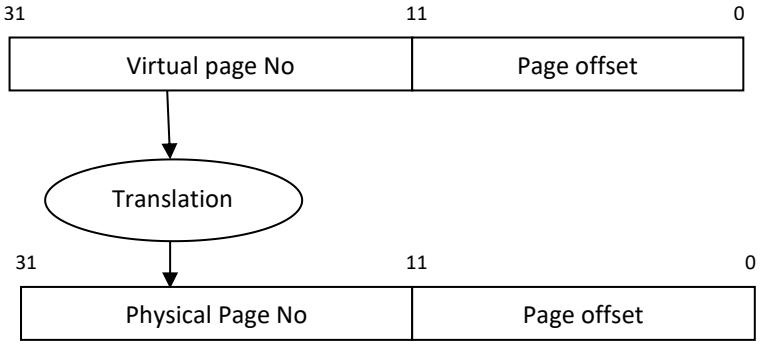
1. A page number (p) in logical address space and
2. The displacement (d), also called offset, the displacement of the given page p, from the start of it.

This is known as the Address Translation Scheme. Let us take an example of a 16-bit address, which is divided into two parts with page number (p) taking 5 bits and Displacement (d) taking 11 bits, as shown in the figure below.



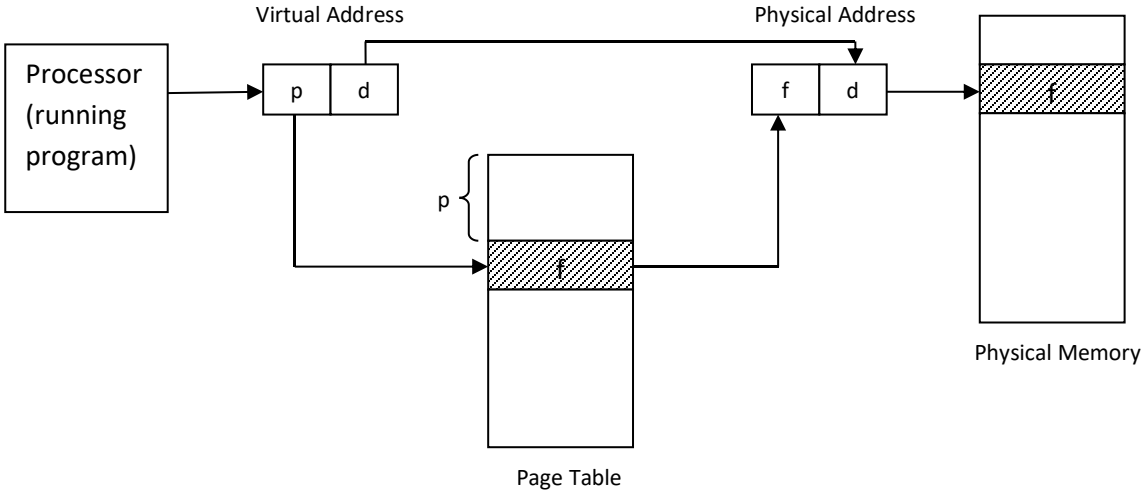
Here the page no. (n) takes 5 bits, so range of values is 0 – 31 (i.e. 2^5-1). Similarly, offset value uses 11-bits, so range is 0 to 2047 (i.e. 2^{11}-1). Summarizing this we can say that this paging scheme uses 32 pages, each with 2048 locations.

The table which holds virtual address to physical address translations, is called the page table. As displacement is constant, so only translation of virtual page number of physical page number is required.



Address Translation scheme (32 bit address)

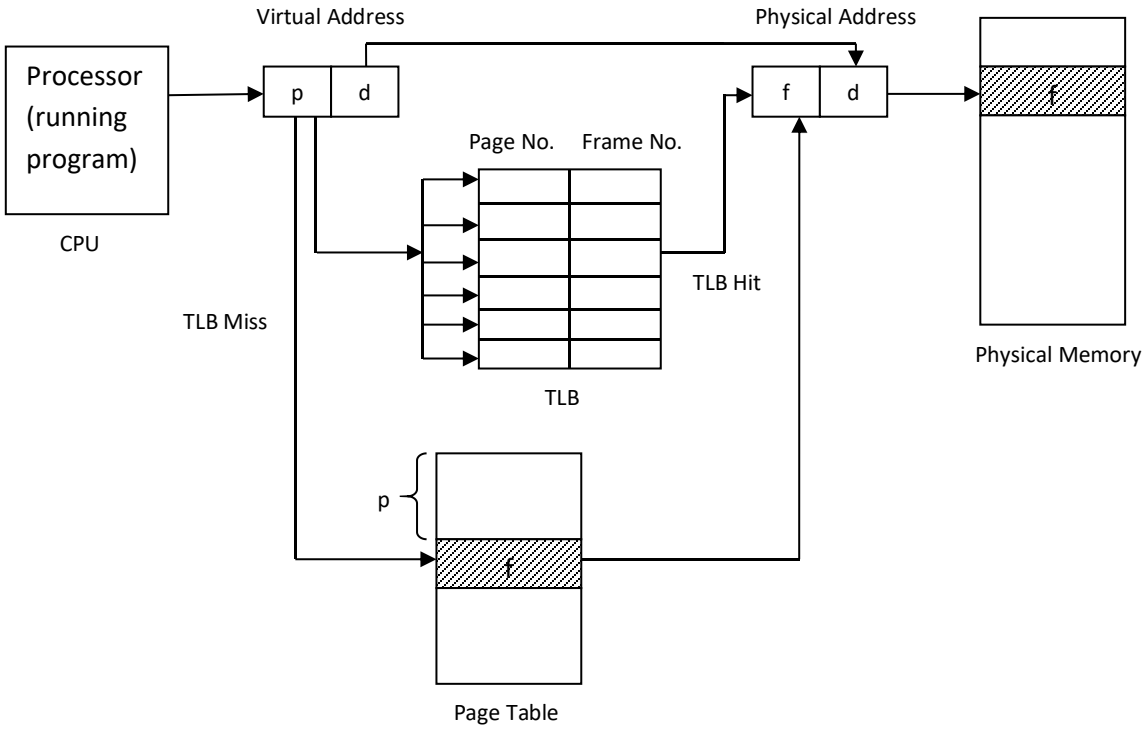
The page number is used as an index into a page table and the latter contains base address of each corresponding physical memory page number (Frame number). This reduces dynamic relocation efforts. The paging hardware support is shown below.



Paging Address Translation, using Direct Mapping

This is the scenario of paging address translation, using direct mapping. Here the page table is used directly to locate the address of physical memory page. But main disadvantage of this direct mapping is its speed of translation. As because the page table resides in the main memory and its size may increase considerably, which increases instruction execution time.

We can overcome this by using additional hardware support of registers and buffers. This is the Paging Address Translation with Associative Mapping. This scheme is based on use of dedicated registers with high speed and efficiency. These small, fast-lookup cache help to place a part of the entire page table into a content-addresses associative storage and hence speed-up the lookup problem with a cache. These are known as Associative Registers or Translation Look-aside Buffers (TLBs).



Paging Address Translation, using Associative and Direct Mapping

Each register consists of two entries.

(i) Key, which is matched with logical page p. and

(ii) Value which returns page frame number corresponding to p.

It is similar to direct mapping scheme but here as TLBs contain only few page table entries, so search is fast, although it is quite expensive due to register support. So both direct and associative mapping schemes can be combined to get more benefits. Here, page number is matched with all associative registers simultaneously. The percentage of the number of times the page is found in TLB is call **hit ratio**. If it is

not found, it is searched in page table and added into TLB. If TLB is already full, then page replacement policies can be used.

Protection and Sharing

Paging hardware also contains some protection mechanism. In page table corresponding to each frame a protection bit is associated. This bit can tell if page is read-only or read-write.

Sharing code and data takes place if two page table entries in different processes point to same physical page, i.e. the processes share the same memory.

Advantages of Paging

1. Virtual address space may be greater than main memory size. i.e. programs with large logical address space, compared with physical address space can be executed.
2. Avoid external fragmentation and hence storage compaction.
3. Full utilization of available main storage.

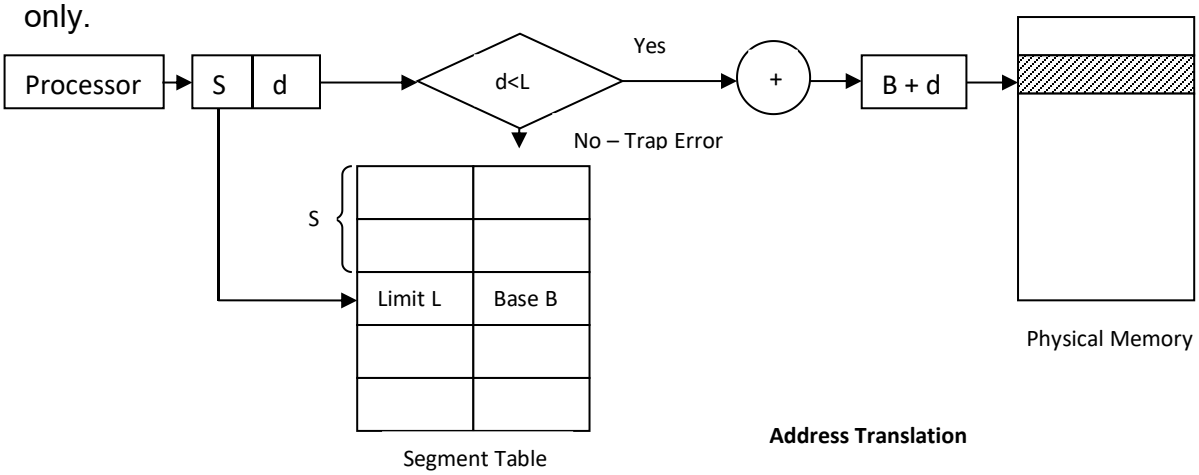
Disadvantages of Paging

1. Internal fragmentation problem, i.e. wastage within allocated page when process is smaller than page boundary exists.
2. Extra resource consumption and overheads for paging hardware is involved.

Segmentation

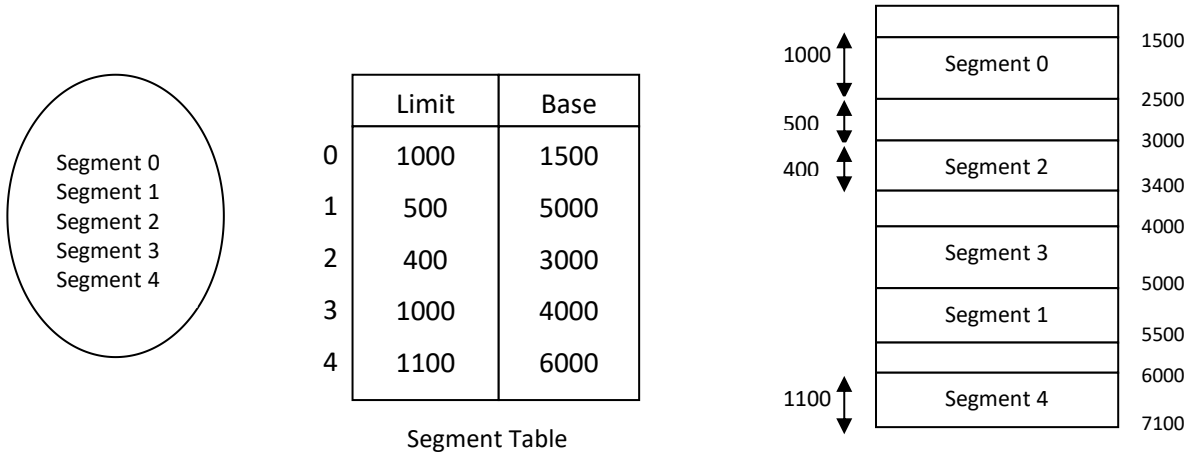
Segmentation presents an alternative scheme for memory management. This scheme divides the logical address space into variable length chunks, called **segments**, with no proper ordering among them. Each segment has a segment number and a length. Thus the logical addresses are expressed as a pair of segment number and offset within segment.

It allows a program to be broken down into logical parts according to the user view of the memory, which is then mapped into physical memory. Thus logical addresses are two-dimensional but physical addresses are still one-dimensional array of bytes only.



Address Translation

This mapping between the logical and physical memory is done by the segment table, which contains segment base and length of segment. The offset d must range between 0 and segment limit/length, otherwise it will generate address error. This situation is shown in the figure below.



Segmentation - Principle of operation

This method also allows read-only segments to be shared, so that two processes can use shared code for better memory efficiency. With each segment-table entry, protection bit specifying segment as read-only or execute only can be used. Hence illegal attempts to write into a read-only segment can be prevented.

Segmentation may suffer from external fragmentation, i.e. when blocks of free memory are not enough to accommodate a segment. Storage compaction and coalescing can minimize this drawback.