**Memory Management**

Main Memory refers to a physical memory that is the internal memory to the computer. The word main is used to distinguish it from external mass storage devices such as disk drives. Main memory is also known as RAM. The computer is able to change only data that is in main memory. Therefore, every program we execute and every file we access must be copied from a storage device into main memory.

All the programs are loaded in the main memory for execution. Sometimes complete program is loaded into the memory, but sometimes a certain part or routine of the program is loaded into the main memory only when it is called by the program, this mechanism is called **Dynamic Loading**, this enhance the performance.

Also, at times one program is dependent on some other program. In such a case, rather than loading all the dependent programs, CPU links the dependent programs to the main executing program when its required. This mechanism is known as **Dynamic Linking**.

**Swapping**

A process needs to be in memory for execution. But sometimes there is not enough main memory to hold all the currently active processes in a timesharing system. So, excess process are kept on disk and brought in to run dynamically. **Swapping** is the process of bringing in each process in main memory, running it for a while and then putting it back to the disk.

[**CLICK HERE FOR DETAILS**](#_Swapping)

**Contiguous Memory Allocation**

In contiguous memory allocation each process is contained in a single contiguous block of memory. Memory is divided into several fixed size partitions. Each partition contains exactly one process. When a partition is free, a process is selected from the input queue and loaded into it. The free blocks of memory are known as *holes*. The set of holes is searched to determine which hole is best to allocate.

[**CLICK HERE for details**](#_Contiguous_Memory_Vs)

**Memory Protection**

Memory protection is a phenomenon by which we control memory access rights on a computer. The main aim of it is to prevent a process from accessing memory that has not been allocated to it. Hence prevents a bug within a process from affecting other processes, or the operating system itself, and instead results in a segmentation fault or storage violation exception being sent to the disturbing process, generally killing of process.

**Memory Allocation**

Memory allocation is a process by which computer programs are assigned memory or space. It is of three types:

* **First Fit**

The first hole that is big enough is allocated to program.

* **Best Fit**

The smallest hole that is big enough is allocated to program.

* **Worst Fit**

The largest hole that is big enough is allocated to program.

**Fragmentation**

Fragmentation occurs in a dynamic memory allocation system when most of the free blocks are too small to satisfy any request. It is generally termed as inability to use the available memory.

In such situation processes are loaded and removed from the memory. As a result of this, free holes exists to satisfy a request but is non contiguous i.e. the memory is fragmented into large no. of small holes. This phenomenon is known as **External Fragmentation.**

Also, at times the physical memory is broken into fixed size blocks and memory is allocated in unit of block sizes. The memory allocated to a space may be slightly larger than the requested memory. The difference between allocated and required memory is known as **Internal fragmentation** i.e. the memory that is internal to a partition but is of no use.

**Paging**

A solution to fragmentation problem is Paging. Paging is a memory management mechanism that allows the physical address space of a process to be non-contagious. Here physical memory is divided into blocks of equal size called **Pages**. The pages belonging to a certain process are loaded into available memory frames.

[**CLICK HERE FOR DETAILS**](#_Paging)

**Page Table**

A Page Table is the data structure used by a virtual memory system in a computer operating system to store the mapping between *virtual address* and *physical addresses.*

Virtual address is also known as Logical address and is generated by the CPU. While Physical address is the address that actually exists on memory.

**Segmentation**

Segmentation is another memory management scheme that supports the user-view of memory. Segmentation allows breaking of the virtual address space of a single process into segments that may be placed in non-contiguous areas of physical memory.

**[CLICK HERE FOR DETAILS](#_Segmentation)**

**Segmentation with Paging**

Both paging and segmentation have their advantages and disadvantages, it is better to combine these two schemes to improve on each. The combined scheme is known as 'Page the Elements'. Each segment in this scheme is divided into pages and each segment is maintained in a page table. So the logical address is divided into following 3 parts :

* Segment numbers(S)
* Page number (P)
* The displacement or offset number (D)

**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 :

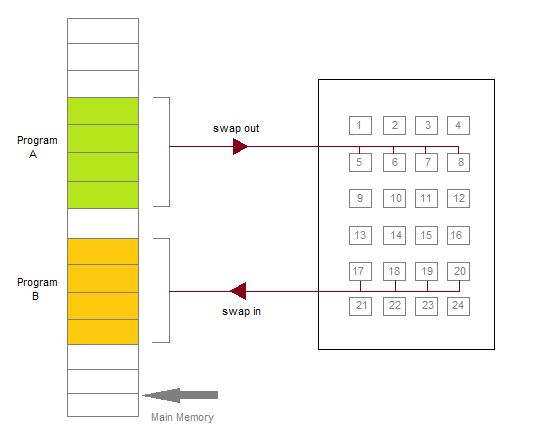
* Error handling code is not needed unless that specific error occurs, some of which are quite rare.
* Arrays are often over-sized for worst-case scenarios, and only a small fraction of the arrays are actually used in practice.
* Certain features of certain programs are rarely used.

**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**

The basic idea behind demand paging is that when a process is swapped in, its pages are not swapped in all at once. Rather they are swapped in only when the process needs them(On demand). This is termed as lazy swapper, although a pager is a more accurate term.



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.
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. Its is not a big issue for small programs, but for larger programs it affects performance drastically.

**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,*

**Basic Page Replacement**

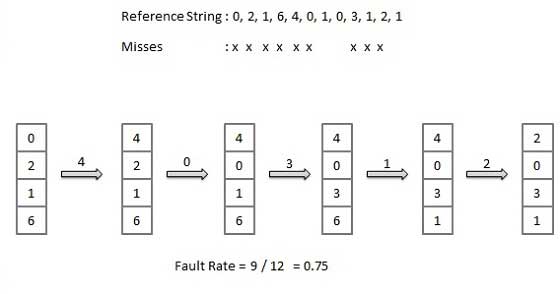
* Find the location of the page requested by ongoing process on the disk.
* Find a free frame. If there is a free frame, use it. If there is no free frame, use a page-replacement algorithm to select any existing frame to be replaced, such frame is known as victim frame.
* Write the victim frame to disk. Change all related page tables to indicate that this page is no longer in memory.
* Move the required page and store it in the frame. Adjust all related page and frame tables to indicate the change.
* Restart the process that was waiting for this page.

**FIFO Page Replacement**

* A very simple way of Page replacement is FIFO (First in First Out)
* As new pages are requested and are swapped in, they are added to tail of a queue and the page which is at the head becomes the victim.
* Its not an effective way of page replacement but can be used for small systems.

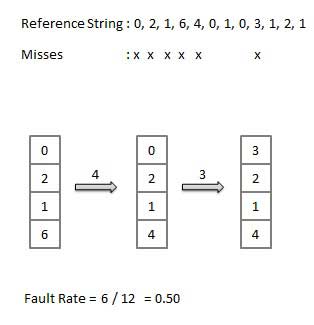
**First In First Out (FIFO) algorithm**

* 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.



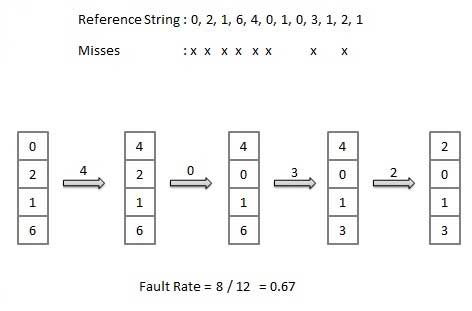
**Optimal Page algorithm**

* 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.



**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.



# Contiguous Memory Vs Non-contiguous Memory Allocation

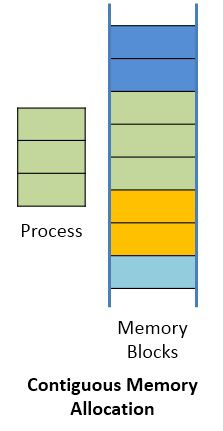
Memory is a large array of bytes, where each byte has its own address. The memory allocation can be classified into two methods contiguous memory allocation and non-contiguous memory allocation. The major difference between Contiguous and Noncontiguous memory allocation is that the **contiguous memory allocation** assigns the consecutive blocks of memory to a process requesting for memory whereas, the **noncontiguous memory allocation** assigns the separate memory blocks at the different location in memory space in a nonconsecutive manner to a process requesting for memory. We will discuss some more differences between contiguous and non-contiguous memory allocation with the help of comparison chart shown below.

| BASIS THE COMPARISON | CONTIGUOUS MEMORY ALLOCATION | NONCONTIGUOUS MEMORY ALLOCATION |
| --- | --- | --- |
| Basic | Allocates consecutive blocks of memory to a process. | Allocates separate blocks of memory to a process. |
| Overheads | Contiguous memory allocation does not have the overhead of address translation while execution of a process. | Noncontiguous memory allocation has overhead of address translation while execution of a process. |
| Execution rate | A process executes faster in contiguous memory allocation | A process executes quite slower comparatively in noncontiguous memory allocation. |
| Solution | The memory space must be divided into the fixed-sized partition and each partition is allocated to a single process only. | Divide the process into several blocks and place them in different parts of the memory according to the availability of memory space available. |
| Table | A table is maintained by operating system which maintains the list of available and occupied partition in the memory space | A table has to be maintained for each process that carries the base addresses of each block which has been acquired by a process in memory. |

**Definition of Contiguous Memory Allocation**

The operating system and the user’s processes both must be accommodated in the main memory. Hence the main memory is **divided into two** partitions: at one partition the operating system resides and at other the user processes reside. In usual conditions, the several user processes must reside in the memory at the same time, and therefore, it is important to consider the allocation of memory to the processes.

The Contiguous memory allocation is one of the methods of memory allocation. In contiguous memory allocation, when a process requests for the memory, a **single contiguous section of memory blocks** is assigned to the process according to its requirement.

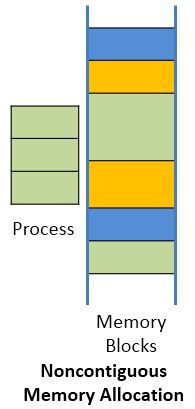


The contiguous memory allocation can be achieved by dividing the memory into the fixed-sized**partition** and allocate each partition to a single process only. But this will cause the degree of multiprogramming, bounding to the number of fixed partition done in the memory. The contiguous memory allocation also leads to the **internal fragmentation**. Like, if a fixed sized memory block allocated to a process is slightly larger than its requirement then the left over memory space in the block is called internal fragmentation. When the process residing in the partition terminates the partition becomes available for another process.

In the variable partitioning scheme, the operating system maintains a **table** which indicates, which partition of the memory is free and which occupied by the processes. The contiguous memory allocation fastens the execution of a process by reducing the overheads of address translation.

**Definition Non-Contiguous Memory Allocation**

The Non-contiguous memory allocation allows a process to**acquire the several memory blocks at the different location in the memory** according to its requirement. The noncontiguous memory allocation also **reduces** the **memory wastage** caused due to internal and external fragmentation. As it utilizes the memory holes, created during internal and external fragmentation.



**Paging and segmentation** are the two ways which allow a process’s physical address space to be non-contiguous. In non-contiguous memory allocation, the process is divided into **blocks** (pages or segments) which are placed into the different area of memory space according to the availability of the memory.

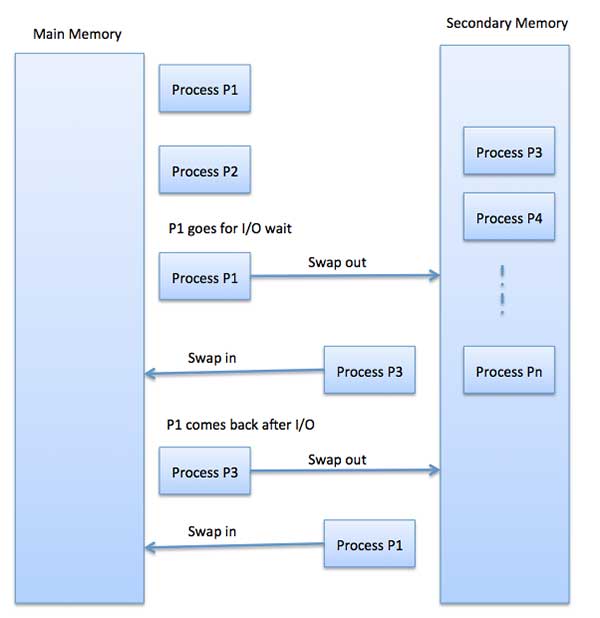
The noncontiguous memory allocation has an advantage of reducing memory wastage but, but it **increases** the **overheads** of address translation. As the parts of the process are placed in a different location in memory, it **slows the execution** of the memory because time is consumed in address translation.

Here, the operating system needs to maintain the **table** for each **process** which contains the base address of the each block which is acquired by the process in memory space.

## Swapping

Swapping is mechanisms in which a process can be swapped temporarily out of main memory (or move) to secondary storage (disk) and make that memory available to other processes. At some later time, the system swaps back the process from the secondary storage to main memory.

Though performance is usually affected by swapping process but it helps in running multiple and big processes in parallel and that's the reason **Swapping is also known as a technique for memory compaction**.



The total time taken by swapping process includes the time it takes to move the entire process to a secondary disk and then to copy the process back to memory, as well as the time the process takes to regain main memory.

Let us assume that the user process is of size 2048KB and on a standard hard disk where swapping will take place has a data transfer rate around 1 MB per second. The actual transfer of the 1000K process to or from memory will take

2048KB / 1024KB per second

= 2 seconds

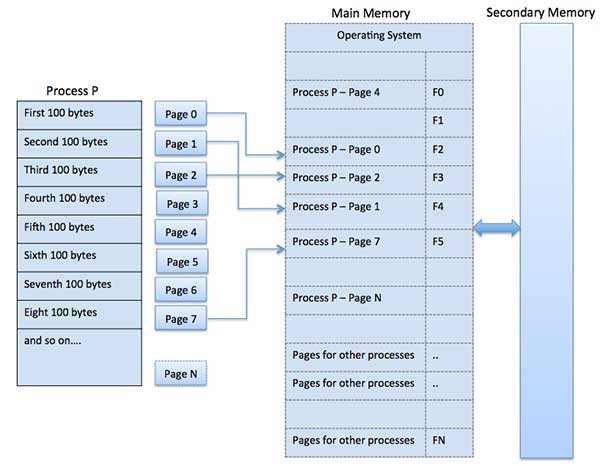
= 2000 milliseconds

### 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.



**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.

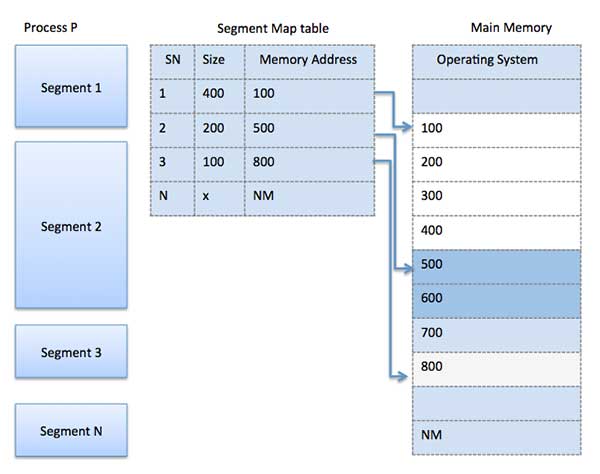
#### Segmentation

Segmentation is a memory management technique in which each job is divided into several segments of different sizes, one for each module that contains pieces that perform related functions. Each segment is actually a different logical address space of the program.

When a process is to be executed, its corresponding segmentation are loaded into non-contiguous memory though every segment is loaded into a contiguous block of available memory.

Segmentation memory management works very similar to paging but here segments are of variable-length where as in paging pages are of fixed size.

A program segment contains the program's main function, utility functions, data structures, and so on. The operating system maintains a **segment map table** for every process and a list of free memory blocks along with segment numbers, their size and corresponding memory locations in main memory. For each segment, the table stores the starting address of the segment and the length of the segment. A reference to a memory location includes a value that identifies a segment and an offset.



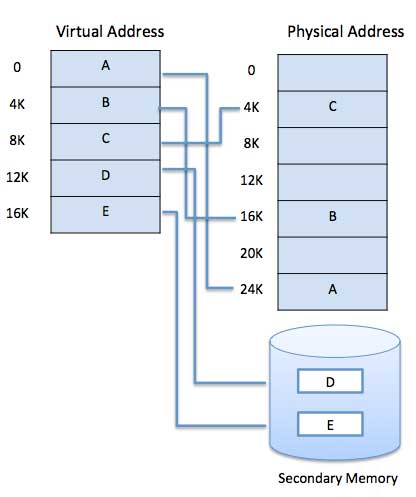
A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.

The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Following are the situations, when entire program is not required to be loaded fully in main memory.

* User written error handling routines are used only when an error occurred in the data or computation.
* Certain options and features of a program may be used rarely.
* Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
* The ability to execute a program that is only partially in memory would counter many benefits.
* Less number of I/O would be needed to load or swap each user program into memory.
* A program would no longer be constrained by the amount of physical memory that is available.
* Each user program could take less physical memory, more programs could be run the same time, with a corresponding increase in CPU utilization and throughput.

Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses. A basic example is given below −



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.