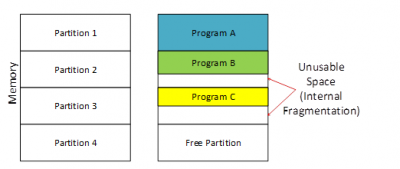
## Fragmentation

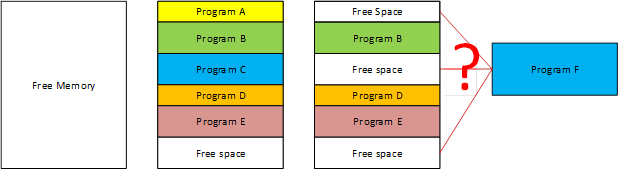
### Internal Fragmentation



Consider the figure above where a fixed sized memory allocation mechanism is being followed. Initially, the memory is empty and the allocator has divided the memory into fixed size [partitions](https://www.differencebetween.com/difference-between-internal-and-vs-external-fragmentation/www.differencebetween.com/difference-between-partition-and-vs-volume/). Then later three programs named A, B, C have been loaded to the first three partitions while the 4th partition is still free. Program A matches the size of the partition, so there is no wastage in that partition, but Program B and Program C are smaller than the partition size. Therefore, in partition 2 and partition 3 there is remaining free space. However, this free space is unusable as the memory allocator only assigns full partitions to programs but no t parts of it. This wastage of free space is called internal fragmentation.

In the above example, it is equal sized fixed partitions but this can even happen in a situation where partitions of various fixed sizes are available. Usually the memory or hardest space is divided into blocks that are usually the size of powers of 2 such as 2, 4, 8, 16 bytes. So a program or a file of 3 bytes will be assigned to a 4 byte block but one byte of that block will become unusable causing internal fragmentation.

### External Fragmentation

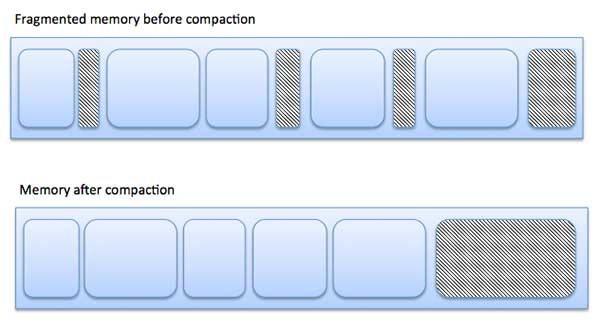


Consider the figure above where memory allocation is done dynamically. In dynamic memory allocation, the allocator allocates only the exact needed size for that program. First memory is completely free. Then the Programs A, B, C, D and E of different sizes are loaded one after the other and they are placed in memory contiguously in that order. Then later, Program A and Program C closes and they are unloaded from memory. Now there are three free space areas in the memory, but they are not adjacent. Now a large program called Program F is going to be loaded but neither of the free space block is not enough for Program F. The addition of all the free spaces is definitely enough for Program F, but due to the lack of adjacency that space is unusable for Program F. This is called External Fragmentation.

### What is the difference between Internal and External Fragmentation?

* Internal Fragmentation occurs when a fixed size memory allocation technique is used. External fragmentation occurs when a dynamic memory allocation technique is used.
* Internal fragmentation occurs when a fixed size partition is assigned to a program/file with less size than the partition making the rest of the space in that partition unusable. External fragmentation is due to the lack of enough adjacent space after loading and unloading of programs or files for some time because then all free space is distributed here and there.
* External fragmentation can be mined by compaction where the assigned blocks are moved to one side, so that contiguous space is gained. However, this operation takes time and also certain critical assigned areas for example system services cannot be moved safely. We can observe this compaction step done on hard disks when running the disk defragmenter in Windows.
* External fragmentation can be prevented by mechanisms such as segmentation and paging. Here a logical contiguous virtual memory space is given while in reality the files/programs are splitted into parts and placed here and there.
* Internal fragmentation can be maimed by having partitions of several sizes and assigning a program based on the best fit. However, still internal fragmentation is not fully eliminated.
* External fragmentation can be reduced by compaction or shuffle memory contents to place all free memory together in one large block. To make compaction feasible, relocation should be dynamic.
* The internal fragmentation can be reduced by effectively assigning the smallest partition but large enough for the process.

The following diagram shows how fragmentation can cause waste of memory and a compaction technique can be used to create more free memory out of fragmented memory −



## Memory Allocation techniques

Memory allocation is the process of assigning blocks of memory on request. Typically, the allocator receives memory from the operating system in a small number of large blocks that it must divide to satisfy the requests for smaller blocks. It must also make any returned blocks available for reuse. There are many common ways to perform this, with different strengths and weaknesses. A few are described briefly below.

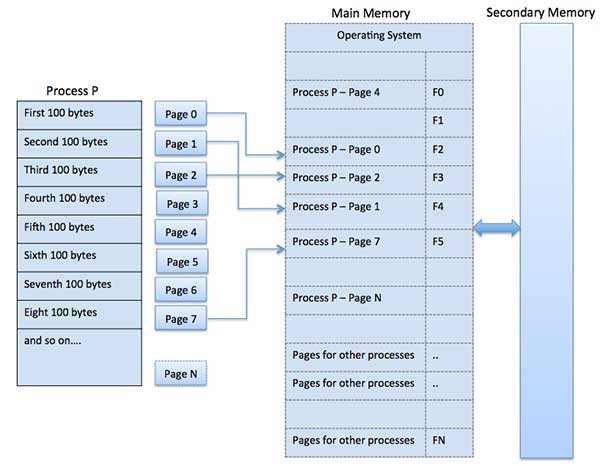
* First fit
* Buddy system
* Sub allocators

## 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.

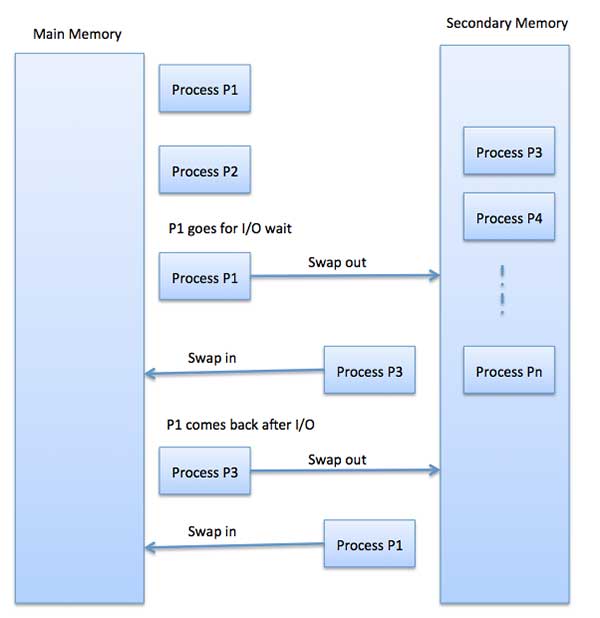
Similarly, main memory is divided into small fixed-sized blocks of (physical) memory called **frames** and the size of a frame is kept the same as that of a page to have optimum utilization of the main memory and to avoid external fragmentation.



## Swapping

Swapping is a mechanism 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 is 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

Now considering in and out time, it will take complete 4000 milliseconds plus other overhead where the process competes to regain main memory.

## File Allocation Methods

The allocation methods define how the files are stored in the disk blocks. There are three main disk space or file allocation methods.

* Contiguous Allocation
* Linked Allocation
* Indexed Allocation

The main idea behind these methods is to provide:

* Efficient disk space utilization.
* Fast access to the file blocks.

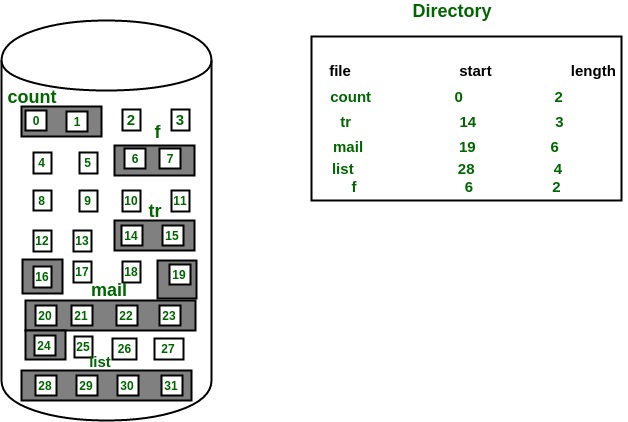
All the three methods have their own advantages and disadvantages as discussed below:

### Contiguous Allocation

In this scheme, each file occupies a contiguous set of blocks on the disk. For example, if a file requires n blocks and is given a block b as the starting location, then the blocks assigned to the file will be: *b, b+1, b+2,……b+n-1.* This means that given the starting block address and the length of the file (in terms of blocks required), we can determine the blocks occupied by the file.  
The directory entry for a file with contiguous allocation contains

* Address of starting block
* Length of the allocated portion.

The *file ‘mail’* in the following figure starts from the block 19 with length = 6 blocks. Therefore, it occupies *19, 20, 21, 22, 23, 24* blocks.

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Contiguous-Allocation.jpg)

#### Advantages:

* Both the Sequential and Direct Accesses are supported by this. For direct access, the address of the kth block of the file which starts at block b can easily be obtained as (b+k).
* This is extremely fast since the number of seeks are minimal because of contiguous allocation of file blocks.

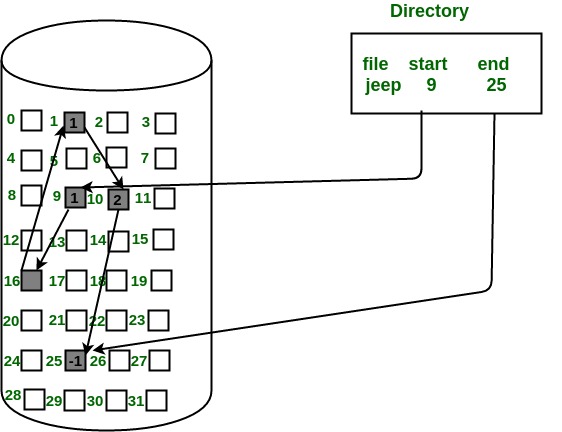
#### Disadvantages:

* This method suffers from both internal and external fragmentation. This makes it inefficient in terms of memory utilization.
* Increasing file size is difficult because it depends on the availability of contiguous memory at a particular instance.

### Linked List Allocation

In this scheme, each file is a linked list of disk blocks, which **need not be** contiguous. The disk blocks can be scattered anywhere on the disk. The directory entry contains a pointer to the starting and the ending file block. Each block contains a pointer to the next block occupied by the file.

The file ‘jeep’ in following image shows how the blocks are randomly distributed. The last block (25) contains -1 indicating a null pointer and does not point to any other block.

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/linkedListAllocation.jpg)

#### Advantages:

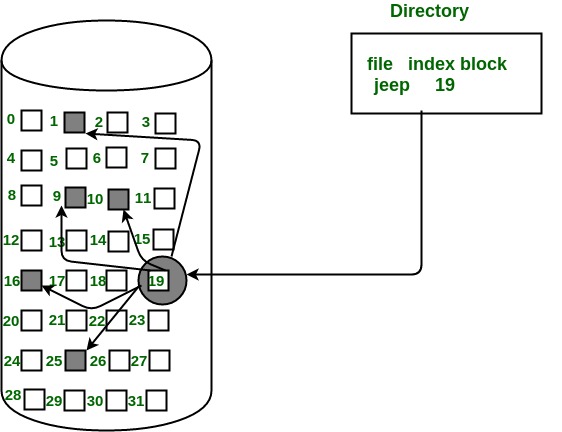
* This is very flexible in terms of file size. File size can be increased easily since the system does not have to look for a contiguous chunk of memory.
* This method does not suffer from external fragmentation. This makes it relatively better in terms of memory utilization.

#### Disadvantages:

* Because the file blocks are distributed randomly on the disk, a large number of seeks are needed to access every block individually. This makes linked allocation slower.
* It does not support random or direct access. We cannot directly access the blocks of a file. A block k of a file can be accessed by traversing k blocks sequentially (sequential access) from the starting block of the file via block pointers.
* Pointers required in the linked allocation incur some extra overhead.

### Indexed Allocation

In this scheme, a special block known as the **Index block** contains the pointers to all the blocks occupied by a file. Each file has its own index block. The ith entry in the index block contains the disk address of the ith file block. The directory entry contains the address of the index block as shown in the image:

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/indexedAllocation.jpg)

#### Advantages:

* This supports direct access to the blocks occupied by the file and therefore provides fast access to the file blocks.
* It overcomes the problem of external fragmentation.

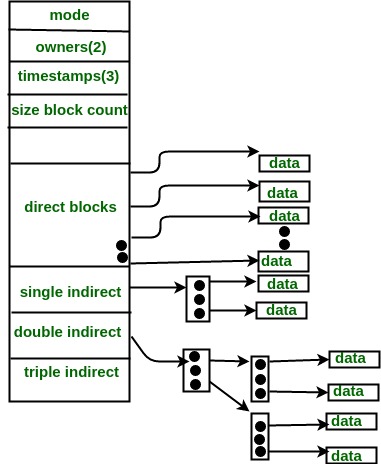
#### Disadvantages:

* The pointer overhead for indexed allocation is greater than linked allocation.
* For very small files, say files that expand only 2-3 blocks, the indexed allocation would keep one entire block (index block) for the pointers, which is inefficient in terms of memory utilization. However, in linked allocation we lose the space of only 1 pointer per block.

For files that are very large, single index block may not be able to hold all the pointers.  
Following mechanisms can be used to resolve this:

1. **Linked scheme:** This scheme links two or more index blocks together for holding the pointers. Every index block would then contain a pointer or the address to the next index block.
2. **Multilevel index:** In this policy, a first level index block is used to point to the second level index blocks which in turn points to the disk blocks occupied by the file. This can be extended to 3 or more levels depending on the maximum file size.
3. **Combined Scheme:** In this scheme, a special block called the **Inode (information Node)** contains all the information about the file such as the name, size, authority, etc. and the remaining space of Inode is used to store the Disk Block addresses, which contain the actual file *as shown in the image below.* The first few of these pointers in Inode point to the **direct blocks** i.e. the pointers contain the addresses of the disk blocks that contain data of the file.

The next few pointers point to indirect blocks. Indirect blocks may be single indirect, double indirect or triple indirect. **Single Indirect block** is the disk block that does not contain the file data but the disk address of the blocks that contain the file data. Similarly, **double indirect blocks** do not contain the file data but the disk address of the blocks that contain the address of the blocks containing the file data.

[](https://cdncontribute.geeksforgeeks.org/wp-content/uploads/Combined-Scheme.jpg)

## Memory management – mapping virtual address to physical addresses

Memory consists of large array of words or arrays, each of which has address associated with it. Now the work of CPU is to fetch instructions from the memory based program counter. Now further these instructions may cause loading or storing to specific memory address.

Address binding is the process of mapping from one address space to another address space. Logical address is address generated by CPU during execution whereas Physical Address refers to location in memory unit (the one that is loaded into memory).Note that user deals with only logical address (Virtual address). The logical address undergoes translation by the MMU or address translation unit in particular. The output of this process is the appropriate physical address or the location of code/data in RAM. An address binding can be done in three different ways:

1. **Compile Time –** If you know that during compile time where process will reside in memory then absolute address is generated i.e. physical address is embedded to the executable of the program during compilation. Loading the executable as a process in memory is very fast. However, if another process preoccupies the generated address space, then the program crashes and it becomes necessary to recompile the program to change the address space.
2. **Load time –** If it is not known at the compile time where process will reside then relocatable address will be generated. Loader translates the relocatable address to absolute address. The base address of the process in main memory is added to all logical addresses by the loader to generate absolute address. In this, if the base address of the process changes then we need to reload the process again.
3. **Execution time-** The instructions are in memory and are being processed by the CPU. Additional memory may be allocated and/or deallocated at this time. This is used if process can be moved from one memory to another during execution (dynamic linking-Linking that is done during load or run time). e.g. – Compaction.

### Memory Management Unit (MMU)

The run time mapping between Virtual address and Physical Address is done by hardware device known as MMU. In memory management, Operating System will handle the processes and moves the processes between disk and memory for execution. It keeps the track of available and used memory.

**Instruction-execution cycle Follows steps:**

1. First instruction is fetched from memory e.g. ADD A,B
2. Then these instructions are decoded i.e., Addition of A and B
3. Further loading or storing at some particular memory location takes place.

**Basic Hardware**

As main memory and registers are built into processor and CPU can access these only. Therefore, every instructions should be written in direct access storage devices.

1. If CPU access instruction from register then it can be done in one CPU clock cycle as registers are built into CPU.
2. If instruction resides in main memory then it will be accessed via memory bus that will take lot of time. So remedy to this add fast memory in between CPU and main memory i.e. adding cache for transaction.
3. Now we should insure that process resides in legal address.
4. Legal address consists of base register (holds smallest physical address) and limit register (size of range).

For example:

Base register = 300040

limit register = 120900

then legal address = (300040+120900)= 420940(inclusive).

legal address = base register+ limit register

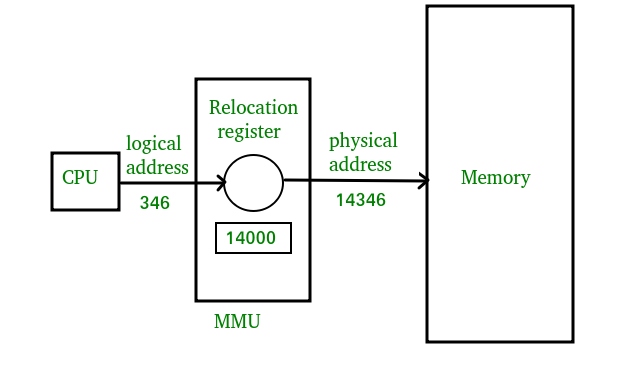
**How processes are mapped from disk to memory?**

1. Usually process resides in disk in form of binary executable file.
2. So to execute process it should reside in main memory.
3. Process is moved from disk to memory based on memory management in use.
4. The processes waits in disk in form of ready queue to acquire memory.

**Procedure of mapping of disk and memory 0**

Normal procedure is that process is selected from input queue and loaded in memory. As process executes it accesses data and instructions from memory and as soon as it completes it will release memory and now memory will be available for other processes.

#### MMU scheme –



* CPU will generate logical address for eg: 346
* MMU will generate relocation register(base register) for eg:14000
* In Memory, physical address is l.

## Disk Scheduling Algorithms

### First Come First Serve

FCFS is the simplest of all the Disk Scheduling Algorithms. In FCFS, the requests are addressed in the order they arrive in the disk queue.

#### Advantages:

* Every request gets a fair chance
* No indefinite postponement

#### Disadvantages:

* Does not try to optimize seek time
* May not provide the best possible service

### Shortest Seek Time First

In SSTF (Shortest Seek Time First), requests having shortest seek time are executed first. So, the seek time of every request is calculated in advance in queue and then they are scheduled according to their calculated seek time. As a result, the request near the disk arm will get executed first. SSTF is certainly an improvement over FCFS as it decreases the average response time and increases the throughput of system.

#### Advantages:

* Average Response Time decreases
* Throughput increases

#### Disadvantages:

* Overhead to calculate seek time in advance
* Can cause Starvation for a request if it has higher seek time as compared to incoming requests
* High variance of response time as SSTF favours only some requests

### SCAN

In SCAN algorithm the disk arm moves into a particular direction and services the requests coming in its path and after reaching the end of disk, it reverses its direction and again services the request arriving in its path. Therefore, this algorithm works like an elevator and hence known as **elevator algorithm.** As a result, the requests at the midrange are serviced more and those arriving behind the disk arm will have to wait.

#### Advantages:

* High throughput
* Low variance of response time
* Average response time

#### Disadvantages:

* Long waiting time for requests for locations just visited by disk arm

### CSCAN

In SCAN algorithm, the disk arm again scans the path that has been scanned, after reversing its direction. Therefore, it may be possible that too many requests are waiting at the other end or there may be zero or few requests pending at the scanned area. These situations are avoided in *CSAN* algorithm in which the disk arm instead of reversing its direction goes to the other end of the disk and starts servicing the requests from there. Therefore, the disk arm moves in a circular fashion and this algorithm is also similar to SCAN algorithm and hence it is known as C-SCAN (Circular SCAN).

#### Advantages:

* Provides more uniform wait time compared to SCAN

### LOOK

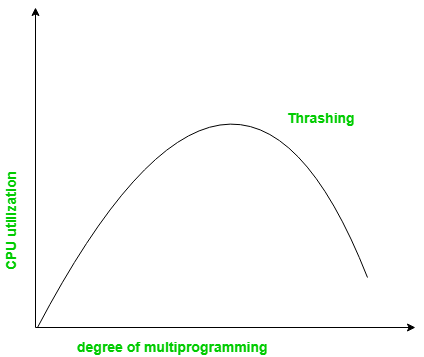
It is similar to the SCAN disk-scheduling algorithm except the difference that the disk arm in spite of going to the end of the disk goes only to the last request to be serviced in front of the head and then reverses its direction from there only. Thus, it prevents the extra delay, which occurred due to unnecessary traversal to the end of the disk.

#### CLOOK

As LOOK is similar to SCAN algorithm, in similar way, CLOOK is similar to CSCAN disk scheduling algorithm. In CLOOK, the disk arm in spite of going to the end goes only to the last request to be serviced in front of the head and then from there goes to the other end’s last request. Thus, it also prevents the extra delay, which occurred due to unnecessary traversal to the end of the disk.

## Thrashing

Thrashing is computer activity that makes little or no progress, usually because memory or other resources have become exhausted or too limited to perform needed operations. When this happens, a pattern typically develops in which a request is made of the operating system by a process or program, the operating system tries to find resources by taking them from some other process, which in turn makes new requests that can't be satisfied. In a virtual storage system, (an operating system that manages its logical storage or memory in units called pages), thrashing is a condition in which excessive paging operations are taking place. A system that is thrashing can be perceived as either a very slow system or one that has come to a halt.



## Page Replacement Algorithms

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,

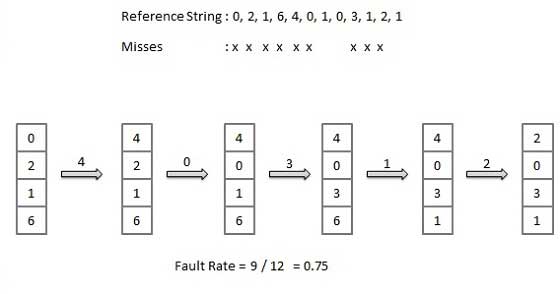
### Reference String

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

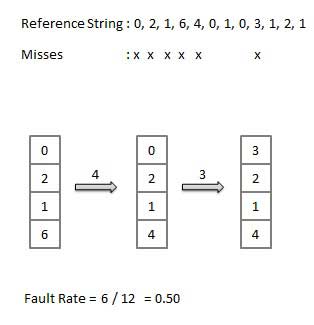
### First In First Out (FIFO) Algorithm

* Oldest page in main memory is the one that will be selected for replacement.
* Easy to implement, keep a list, replace pages from the tail and add new pages at the head.



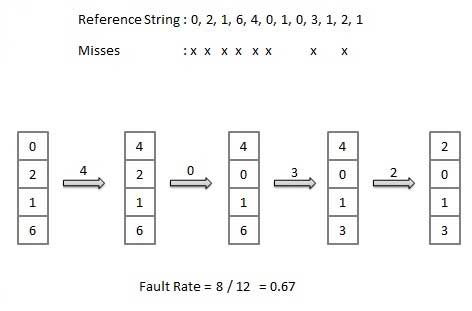
### Optimal Page algorithm

* 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. Use the time when a page is to be used.



### Least Recently Used (LRU) algorithm

* Page that 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.



### Least frequently Used (LFU) algorithm

* The page with the smallest count is the one, which will be selected for replacement.
* This algorithm suffers from the situation in which a page is used heavily during the initial phase of a process, but then is never used again.Most frequently Used (MFU) algorithm
* This algorithm is based on the argument that the page with the smallest count was probably just brought in and has yet to be used.