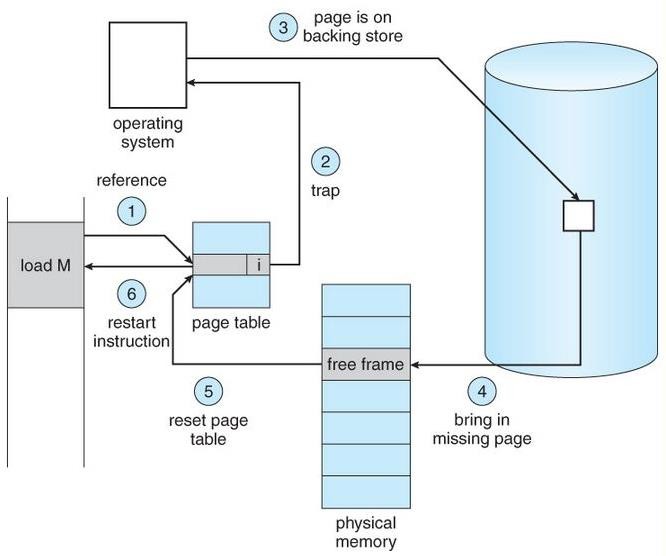
PAGE FAULT HANDLING IN OPERATING SYSTEM

A page fault occurs when a program attempts to access data or code that is in its address space, but is not currently located in the system RAM. So when page fault occurs then following sequence of events happens :

**Steps for handling page fault**

* The basic idea behind paging is that when a process is swapped in, the pager only loads into memory those pages that it expects the process to need ( right away. )
* Pages that are not loaded into memory are marked as invalid in the page table, using the invalid bit. ( The rest of the page table entry may either be blank or contain information about where to find the swapped-out page on the hard drive. )
* If the process only ever accesses pages that are loaded in memory ( **memory resident** pages ), then the process runs exactly as if all the pages were loaded in to memory.
* On the other hand, if a page is needed that was not originally loaded up, then a **page fault trap** is generated, which must be handled in a series of **steps**:

1. The memory address requested is first checked, to make sure it was a valid memory request.
2. If the reference was invalid, the process is terminated. Otherwise, the page must be paged in.
3. A free frame is located, possibly from a free-frame list.
4. A disk operation is scheduled to bring in the necessary page from disk. ( 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 indicate that this is now a valid page reference.
6. The instruction that caused the page fault must now be restarted from the beginning, ( as soon as this process gets another turn on the CPU. )

* In an extreme case, NO pages are swapped in for a process until they are requested by page faults. This is known as **pure demand paging.**
* In theory each instruction could generate multiple page faults. In practice this is very rare, due to **locality of reference**, covered in section 9.6.1.
* The hardware necessary to support virtual memory is the same as for paging and swapping: A page table and secondary memory. ( **Swap space,** whose allocation is discussed in chapter

12. )

* A crucial part of the process is that the instruction must be restarted from scratch once the desired page has been made available in memory. For most simple instructions this is not a major difficulty. However there are some architectures that allow a single instruction to modify a fairly large block of data, ( which may span a page boundary ), and if some of the data gets modified before the page fault occurs, this could cause problems. One solution is to access both ends of the block before executing the instruction, guaranteeing that the necessary pages get paged in before the instruction begins.

IN SHORT-

* 1. Check the location of the referenced page in the PMT
  2. If a page fault occured, call on the operating system to fix it
  3. Using the frame replacement algorithm, find the frame location
  4. Read the data from disk to memory
  5. Update the page map table for the process
  6. The instruction that caused the page fault is restarted when the process resumes execution.

Page Replacement Algoritms

Page replacement is a process of swapping out an existing page from the frame of a main memory and replacing it with the required page.

A good page replacement algorithm is one that minimizes the number of page faults.

* + When there is a page fault, the referenced page must be loaded.
  + If there is no available frame in memory, then one page is selected for replacement
  + If the selected page has been modified, it must be copied back to disk (swapped out)
  + A page replacement algorithm is said to satisfy the inclusion property or is called a **stack algorithm** if the set of pages in a k-frame memory is always a subset of the pages in a (k + 1) frame memory.

Static Replacement Algorithms

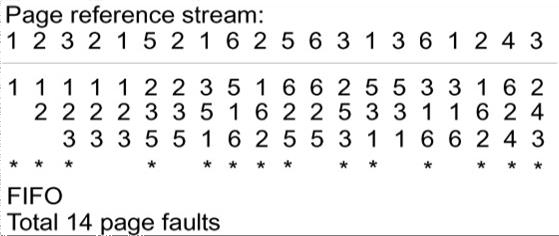
The static paging algorithms implement the replacement policy when the frame allocation to a process is fixed.

**PAGE REPLACEMENT ALGORITHM:**

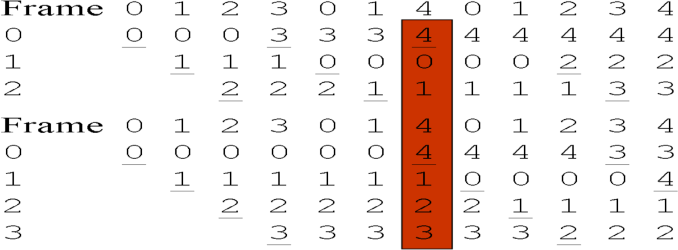
1. FIRST-IN –FIRST-OUT PAGE REPLACEMENT ALG.
2. OPTIMAL PAGE REPLACEMENT ALG.
3. LEAST RECENTLY USED ALG.
4. LRU APPROXIMATION ALG.
5. RANDOM PAGE REPLACEMENT

# First-In-First-Out (FIFO) Replacement

On a page fault, the frame that has been in memory the longest is replaced.

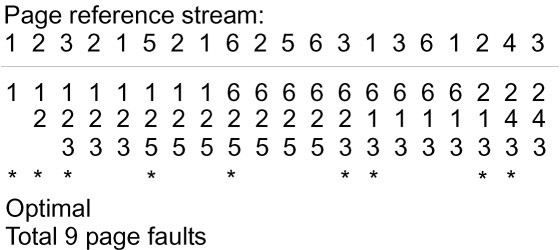


FIFO is not a stack algorithm. In certain cases, the number of page faults can actually increase when more frames are allocated to the process. In the example below, there are 9 page faults for 3 frames and 10 page faults for 4 frames.



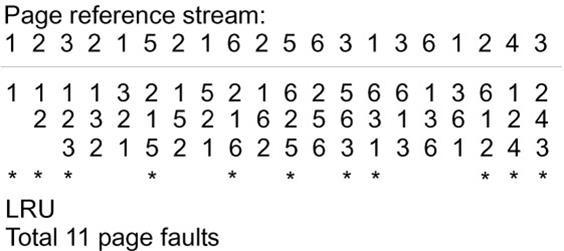
# Optimal Replacement

The Belady’s optimal algorithm cheats. It looks forward in time to see which frame to replace on a page fault. Thus it is not a real replacement algorithm. It gives us a frame of reference for a given static frame access sequence.



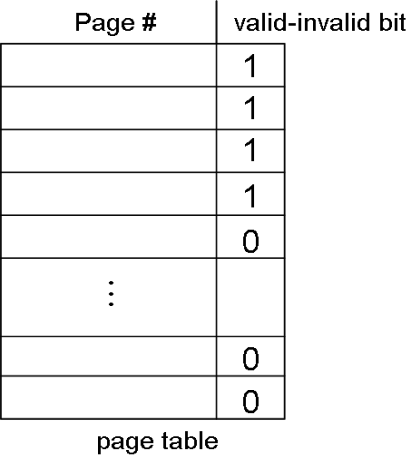
# Least Recently Used (LRU) Replacement

On a page fault, the frame that was least recently used in replaced.



# LRU Approximation

Pages with a current copy on disk are first choice for pages to be removed when more memory is needed. To facilitate [Page Replacement Algoritms,](http://faculty.salina.k-state.edu/tim/ossg/Memory/virt_mem/page_replace.html#page-replace) a table of valid or invalid bits (also called *dirty bits*) is maintained.



* With each page table entry a valid-invalid bit is associated:
* 1 (in-memory)
* 0 (not-in-memory)
* Initially valid-invalid but is set to 0 on all entries.
* In idle times, dirty frame are copied to disk.
* Additional **Reference Bit**: whether the frame was recently referenced.

The reference bits are refreshed on a periodic basis

**RANDOM PAGE REPLACEMENT ALGORITHM**

This algorithm randomly replaces any pages.

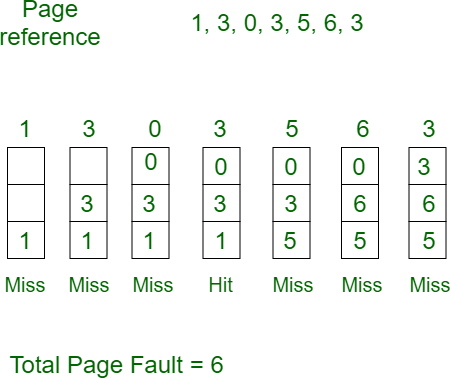
This algorithm may behave like any other algorithm like FIFO, LRU, OPTIMAL etc.

**Page Replacement Algorithms :**

* **First In First Out (FIFO) –**

This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.

**Example-1**Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames.Find number of page faults.



Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**

when 3 comes, it is already in memory so —> **0 Page Faults.**

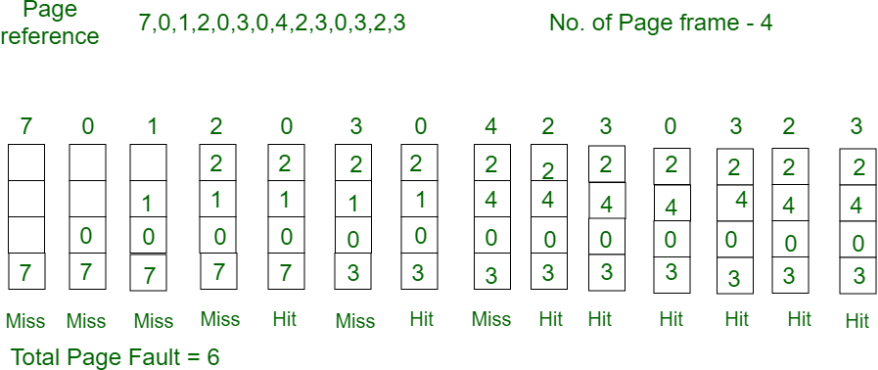
Then 5 comes, it is not available in memory so it replaces the oldest page slot i.e 1. —>**1 Page Fault.** 6 comes, it is also not available in memory so it replaces the oldest page slot i.e 3 —>**1 Page Fault.** Finally when 3 come it is not avilable so it replaces 0 **1 page fault**

[**Belady’s anomaly**](https://www.geeksforgeeks.org/operating-system-beladys-anomaly/) **–** Belady’s anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

* **Optimal Page replacement –**

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

**Example-2:**Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.



Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**

0 is already there so —> **0 Page fault.**

when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>**1 Page fault.**

0 is already there so —> **0 Page fault.**. 4 will takes place of 1 —> **1 Page Fault.**

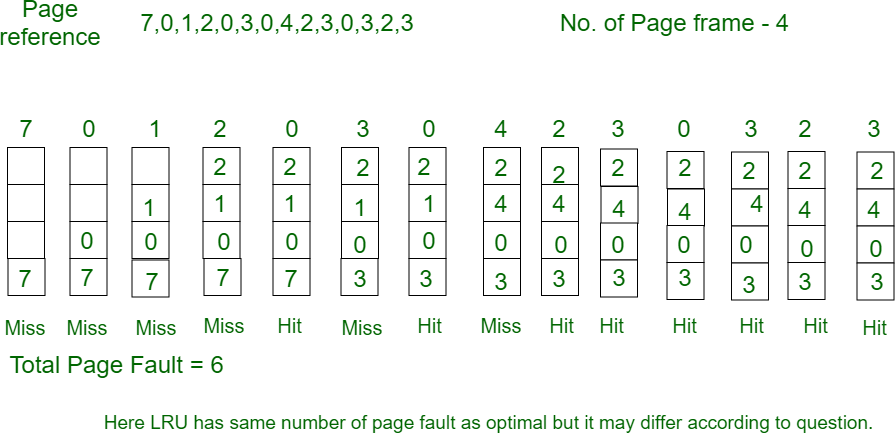
Now for the further page reference string —> **0 Page fault** because they are already available in the memory.

Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

* **Least Recently Used –**

In this algorithm page will be replaced which is least recently used.

* **Example-3**Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2 with 4 page frames.Find number of page faults.



Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**

0 is already their so —> **0 Page fault.**

when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**

0 is already in memory so —> **0 Page fault**. 4 will takes place of 1 —> **1 Page Fault**

Now for the further page reference string —> **0 Page fault** because they are already available in the memory.