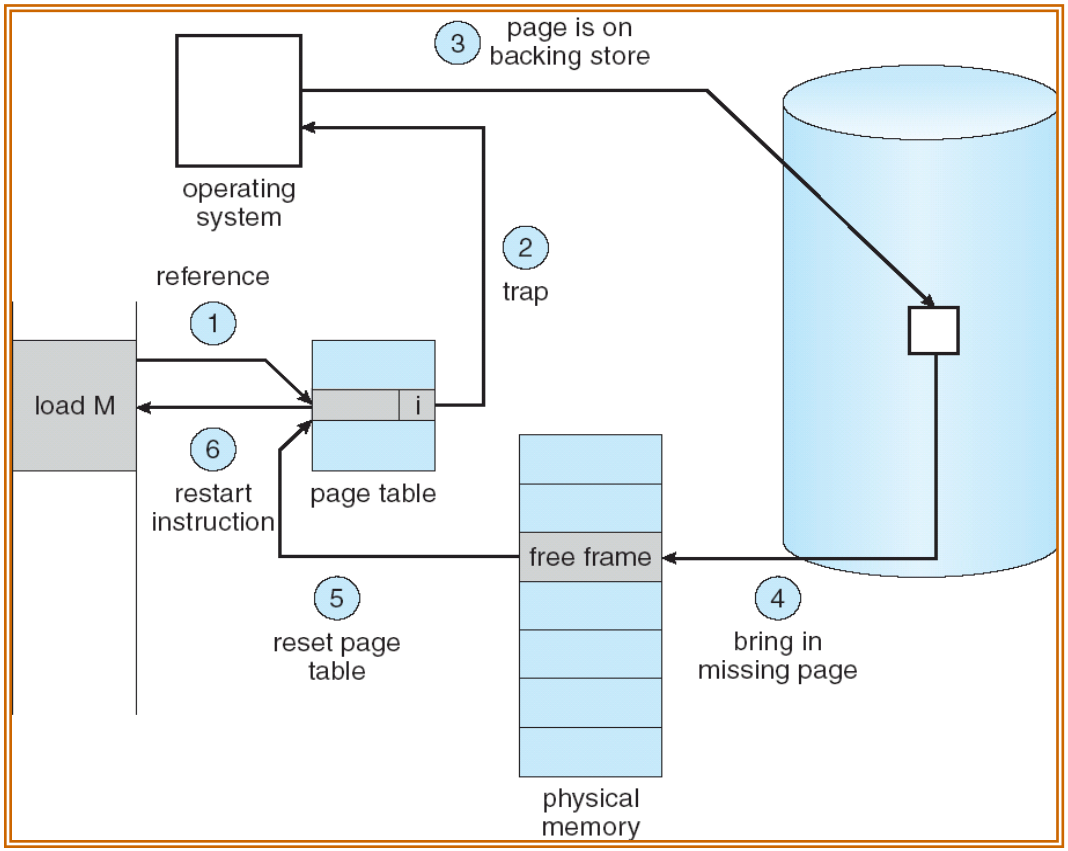
Procedure for handling page fault

- If there is a reference to a page marked invalid , first reference to that page will trap to operating system:

page fault

1. Operating system checks whether the reference was a valid or invalid memory access.
2. If the reference was invalid, we terminate the process. If it was valid, we now page it in.
3. Get empty frame
4. Swap page into the newly allocated frame
5. Reset tables
6. Set validation bit = **v**
7. Restart the instruction that caused the page fault

Steps in handling page fault



- In the extreme case, we can start executing a process with *no pages in* memory.
- When the operating system sets the instruction pointer to the first instruction of the process, which is on a non-memory-resident page, the process immediately faults for the page.
- After this page is brought into memory, the process continues to execute, faulting as necessary until every page that it needs is in memory.
- At that point, it can execute with no more faults.
- This scheme is **pure demand paging.**
- Never bring a page into memory until it is required.

Performance of Demand Paging

- Demand paging can significantly affect the performance of a computer system.
- Let *p be the probability of a page fault (0 ≤ p ≤ 1).*
- We would expect *p to* be close to zero—that is, we would expect to have only a few page faults.
- The effective access time is then
- effective access time = (1 - *p) X ma + p X page fault time.*

(page fault overhead, swap page out, swap page in, restart overhead)

Page Replacement

- While a user process is executing, a page fault occurs.
- The operating system determines where the desired page is residing on the disk but then finds that there are *no free frames* on the free-frame list; all memory is in use.

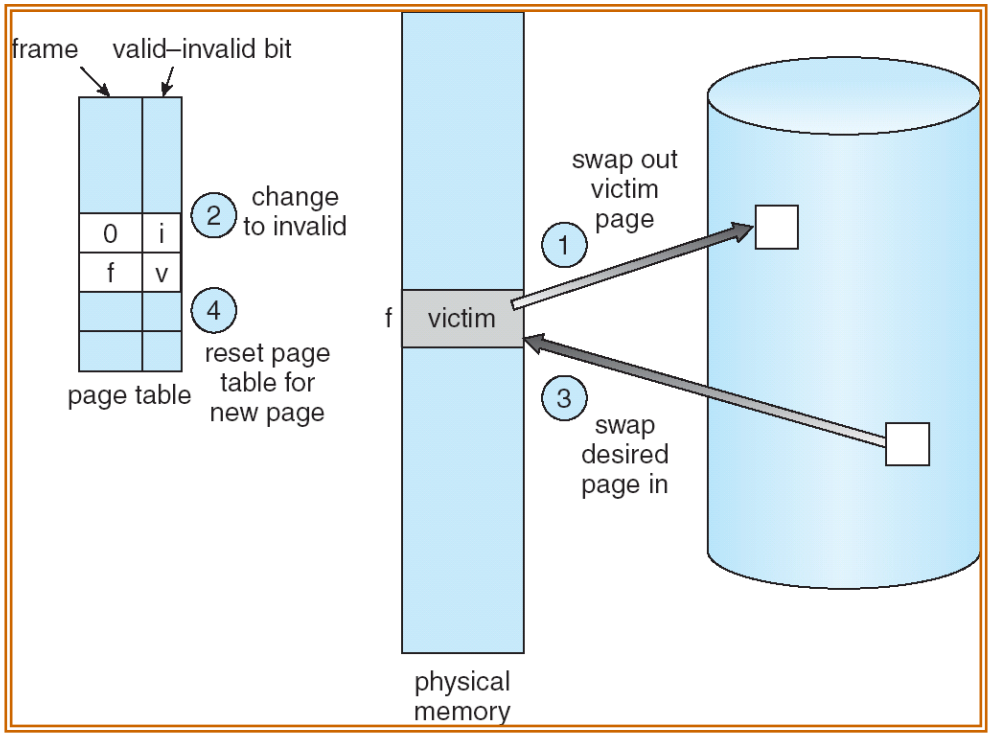
- The operating system could terminate the user process.
- Demand paging is the operating system's attempt to improve the computer system's utilization and throughput.
- The operating system could instead swap out a process, freeing all its frames and reducing the level of multiprogramming.

Basic page Replacement

- If no frame is free, we find one that is not currently being used and free it.
- We can free a frame by writing its contents to swap space and changing the page table to indicate that the page is no longer in memory.
- We can now use the freed frame to hold the page for which the process faulted.

- Find the location of the desired page 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 a victim frame.
  - Write the victim frame to the disk; change the page and frame tables accordingly.
- Read the desired page into the newly freed frame; change the page and frame tables.
- Restart the user process.

Page Replacement



- If no frames are free, *two page transfers (one out and one in) are* required.
- When a modify bit is used, each page or frame has a modify bit associated with it in the hardware.
- The modify bit for a page is set by the hardware whenever any word or byte in the page is written into, indicating that the page has been modified.

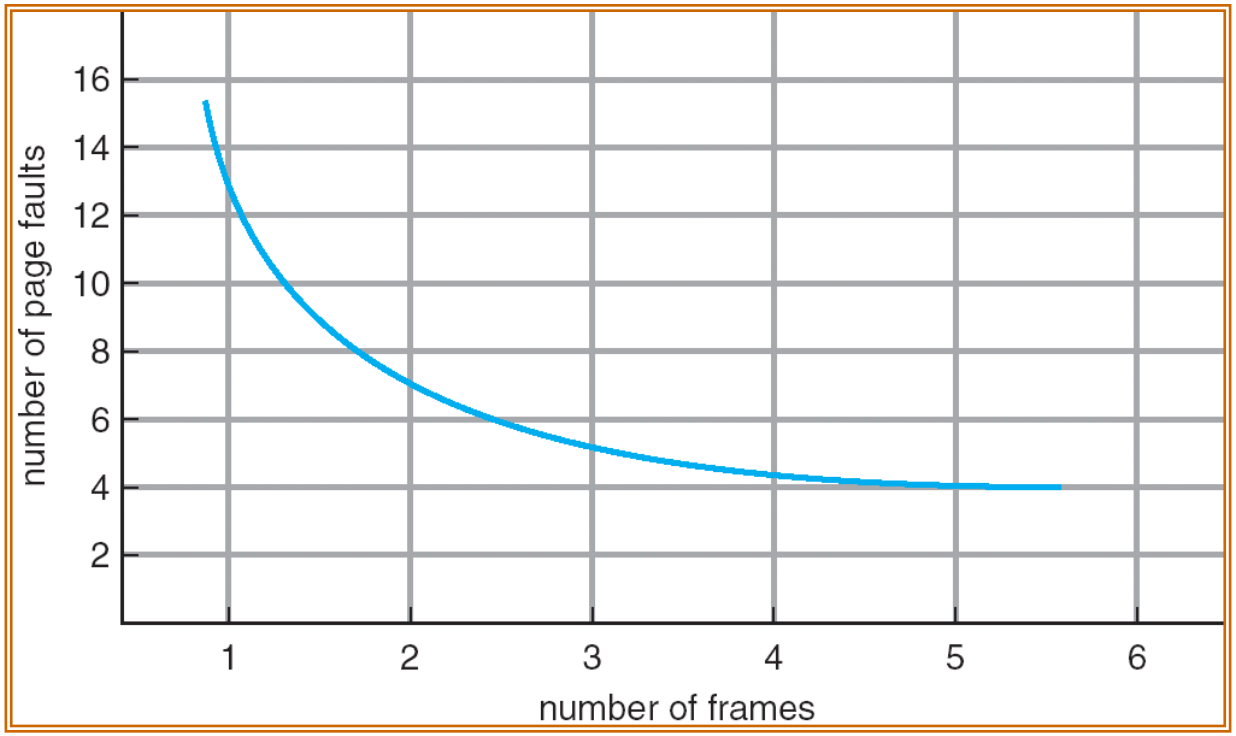
- When we select a page for replacement, we examine its modify bit.
- If the bit is set, we know that the page has been modified since it was read in from the disk.
- In this case, we must write that page to the disk.
- If the modify bit is not set the page has *not* been modified since it was read into memory

- Two major problems to implement demand paging: **frame-allocation algorithm and a page-replacement algorithm.**
- If we have multiple processes in memory, we must decide how many frames to allocate to each process.
- Further, when page replacement is required, we must select the frames that are to be replaced.
- We evaluate an algorithm by running it on a particular string of memory references and computing the number of page faults.
- The string of memory references is called a **reference string.**

- We can generate reference strings artificially (by using a random-number generator, for example), or we can trace a given system and record the address of each memory reference.
- The latter choice produces a large number of data.
- To reduce the number of data, we use two facts.
- First, for a given page size, we need to consider only the page number, rather than the entire address.

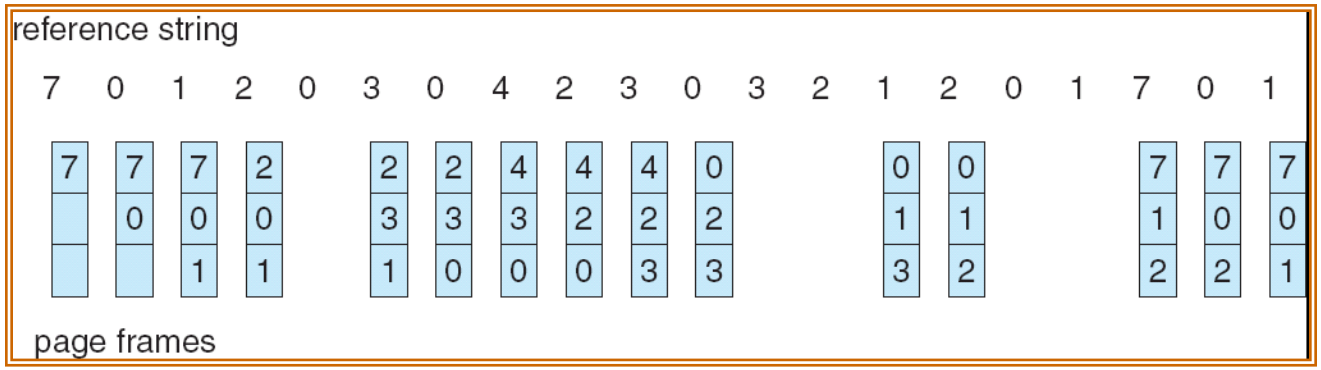
- Second, if we have a reference to a page p, then any immediately following references to page p will never cause a page fault.
- Page p will be in memory after the first reference, so the immediately following references will not fault.
- To determine the number of page faults for a particular reference string and page-replacement algorithm, we also need to know the number of page frames available.
- As the number of frames available increases, the number of page faults decreases.

Graph of Page Faults Versus The Number of Frames



FIFO Page Replacement

- 7, 0,1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2,1, 2, 0, 1, 7, 0,1
- The simplest page-replacement algorithm is a first-in, first-out (FIFO) algorithm.
- A FIFO replacement algorithm associates with each page the time when that page was brought into memory.
- When a page must be replaced, the oldest page is chosen.
- We can create a FIFO queue to hold all pages in memory.
- We replace the page at the head of the queue. When a page is brought into memory, we insert it at the tail of the queue.

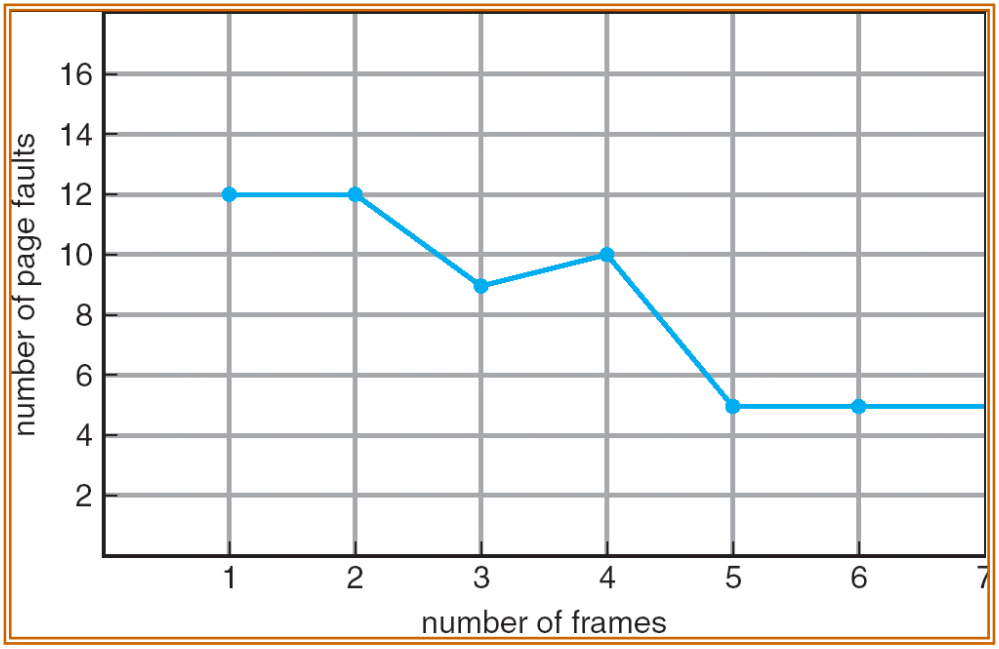

reference string
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
page frames

R	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1																				
F2																				
F3																				
H/M																				

Replace the oldest page.

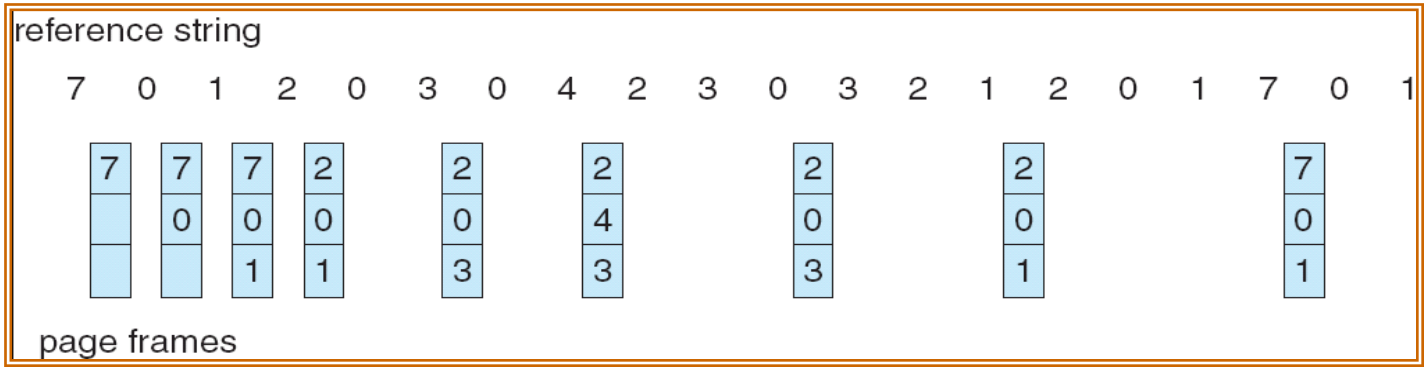
P	3	2	1	3	4	1	6	2	4	3	4	2	1	4	5	2	1	3	4
F1																			
F2																			
F3																			

- Belady's **anomaly: For some page-replacement** algorithms, the page-fault rate may increase as the number of allocated frames increases.



Optimal Page Replacement

- Replace the page that will not be used for the longest period of time.



- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

Optimal

R	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1																				
F2																				
F3																				
H/ M																				

Replace the page that will not be used for the longest period time.

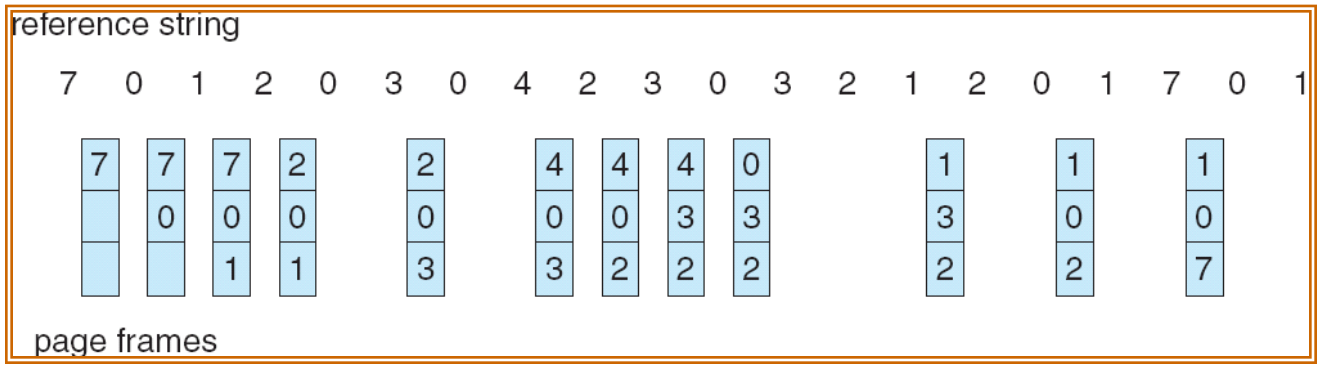
- Unfortunately, the optimal page-replacement algorithm is difficult to implement, because it requires future knowledge of the reference string.
- The optimal algorithm is used mainly for comparison studies.

P	3	2	1	3	4	1	6	2	4	3	4	2	1	4	5	2	1	3	4

LRU Page Replacement

- The key distinction between the FIFO and OPT algorithms is that the FIFO algorithm uses the time when a page was brought into memory, whereas the OPT algorithm uses the time when a page is to be *used.*
- Replace the page that *has not been used for the longest period of time .*
- *This approach is* the **least-recently-used (LRU) algorithm.**

- LRU replacement associates with each page the time of that page's last use.
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time.
- We can think of this strategy as the optimal page-replacement algorithm looking backward in time, rather than forward.


reference string
7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1
7 7 7 2 2 4 4 4 0 1 1 1
0 0 0 0 0 0 3 3 3 0 0
1 1 3 2 2 2 2 2 7
page frames

LRU

R	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
F1																				
F2																				
F3																				
H/M																				

Replace the page that has not been used for the longest time.

- The LRU policy is often used as a page-replacement algorithm and is considered to be good.
- An LRU page-replacement algorithm may require substantial hardware assistance.
- The problem is to determine an order for the frames defined by the time of last use.
- Two implementations are feasible:
  - Counters
  - Stacks

P	3	2	1	3	4	1	6	2	4	3	4	2	1	4	5	2	1	3	4
F1																			
F2																			
F3																			

- We replace the page with the smallest time value.
- This scheme requires a search of the page table to find the LRU page and a write to memory for each memory access.