Virtual memory - Operating system

Chapter · May 2015

DOI: 10.13140/RG.2.1.5040.9126

CITATIONS
0

READS
7,387

1 author:

Qasim Mohammed Hussein
Tikrit University
73 PUBLICATIONS **38** CITATIONS

SEE PROFILE

Virtual memory

Assistance Prof. Dr.
Qasim Mohammed Hussein

Introduction

- The process being execution must be loaded in main memory.
- This requires placing the entire logical address space of program in physical memory.
- Since the physical memory space is limited, so the size of program limits to the size of physical memory.
- Therefore, we need to use techniques that enable us to execute the program without loading all the logical address space.

Virtual memory

- Virtual memory is a technique that allows the execution of processes which are not completely available in memory.
- The main advantage of this scheme is that programs can be larger than physical memory.
- Virtual memory is the separation of user logical memory from physical

Virtual memory

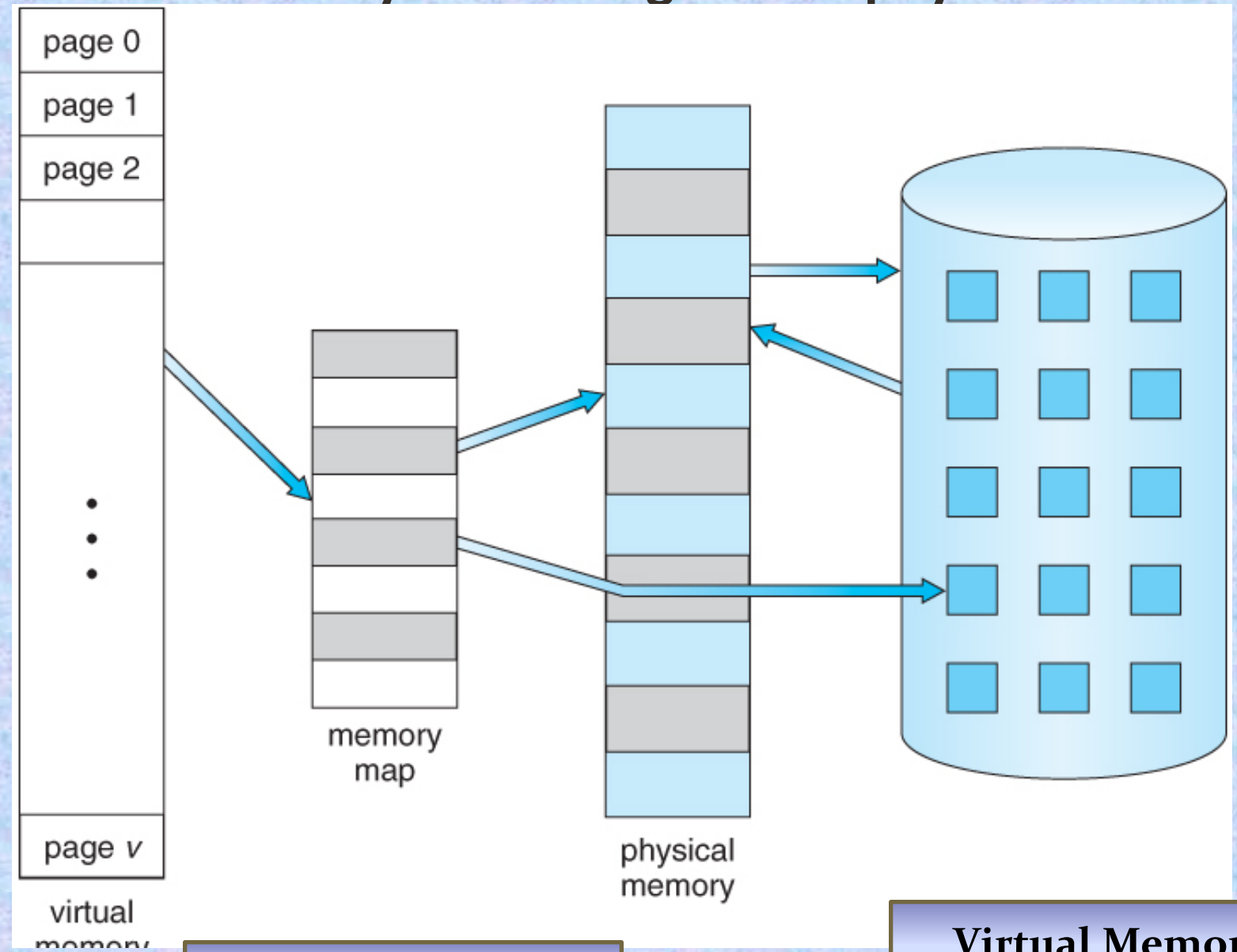
- This separation allows an extremely large virtual memory to be provided for programmers when only a smaller physical memory is available.
- Following are the situations, when entire program is not required to be loaded fully in main memory. For example;

- 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.
- The ability to execute a program that is only partially in memory would counter many benefits.

Virtual memory

- 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, (increase in CPU utilization and throughput.

virtual memory that is larger than physical memory



Virtual memory also allows the sharing of files and memory by multiple processes, with several benefits:

- System libraries can be shared by mapping them into the virtual address space of more than one process.
- Processes can also share virtual memory by mapping the same block of memory to more than one process.
- Process pages can be shared , eliminating the need to copy all of the pages of the original (parent) process.

- Virtual memory is implemented by

1) demand paging.
2) Or segmentation system.

- The basic idea behind ***demand paging*** , pages are only loaded when they are demanded during program execution. we use a ***lazy swapper***, called ***pager.*** .
- A lazy swapper never swaps a page into memory unless that page will be needed

Demand Paging

- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again.
- Instead of swapping in a whole process, the pager brings only those necessary pages into memory. Thus, it avoids reading into memory pages that will not be used in anyway, decreasing the swap time and the amount of physical memory needed.

Using the pager will yield to:

- Less swap time
- Less I/O needed
- Less amount of physical memory needed
- More users
- Faster response time

Advantages and disadvantages of demand paging

Advantages

- Large virtual memory.
- More efficient use of memory.
- Unconstrained multiprogramming. There is no limit on degree of multiprogramming.

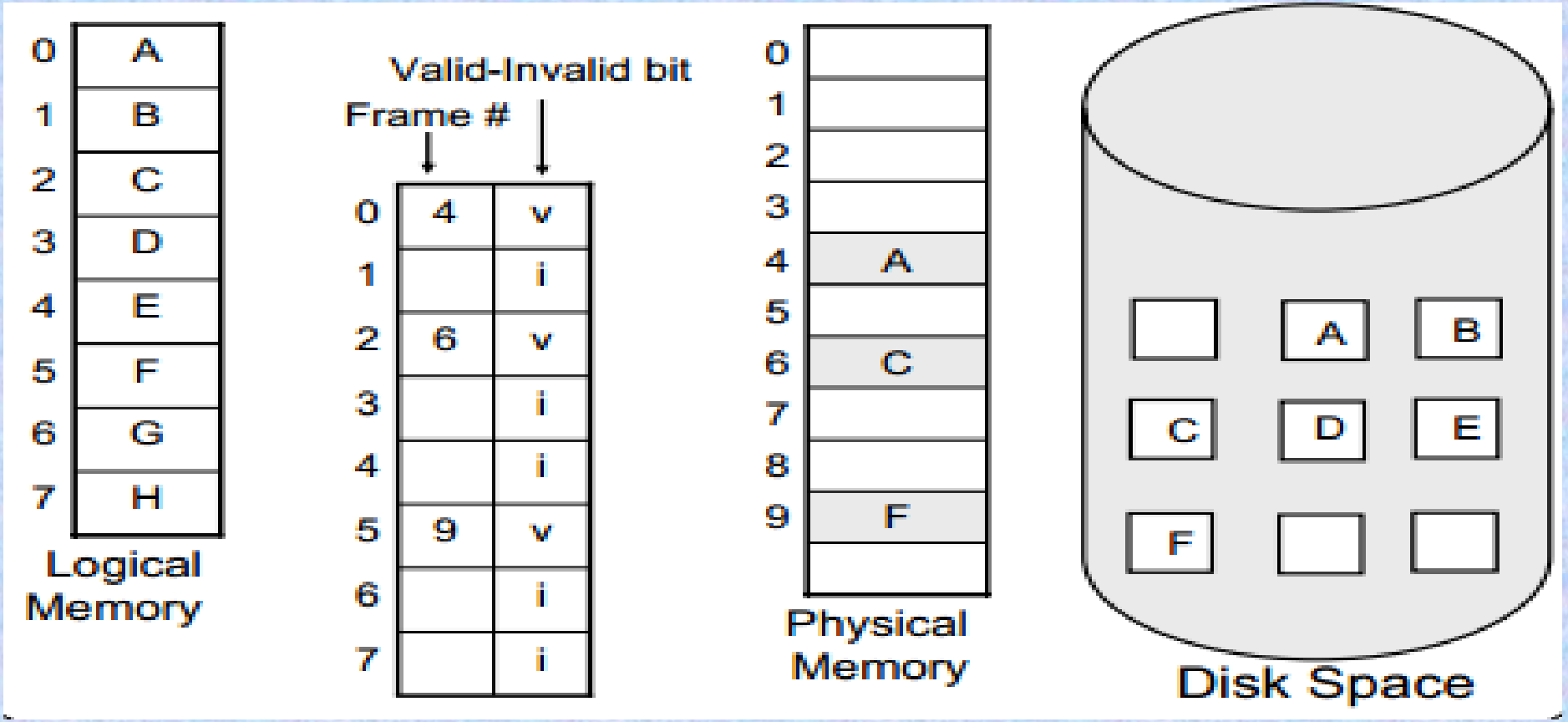
Disadvantages

- Following are the disadvantages of Demand Paging
- Number of tables and amount of processor overhead for handling page interrupts are greater than in the case of the simple paged management techniques.
- Due to the lack of an explicit constraints on a jobs address space size.

Valid and invalid page

- To distinguish between the pages that are in main memory and the pages that are on the disk, the valid-invalid bit is used.
- This bit is set to "valid" when the page in memory, while it set to "invalid" when the page either not valid or is the page is valid but is on the disk . If the process tries to access a page which was not into main memory the a **page fault** will occur.

Page table when some pages are not in main memory



Page Fault

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

page fault

1. Operating system looks at another table to decide:
   - Invalid reference ⇒ abort
   - Just not in memory
2. Get empty frame
3. Swap page into frame
4. Reset tables
5. Set validation bit = **v**
6. Restart the instruction that caused the page fault

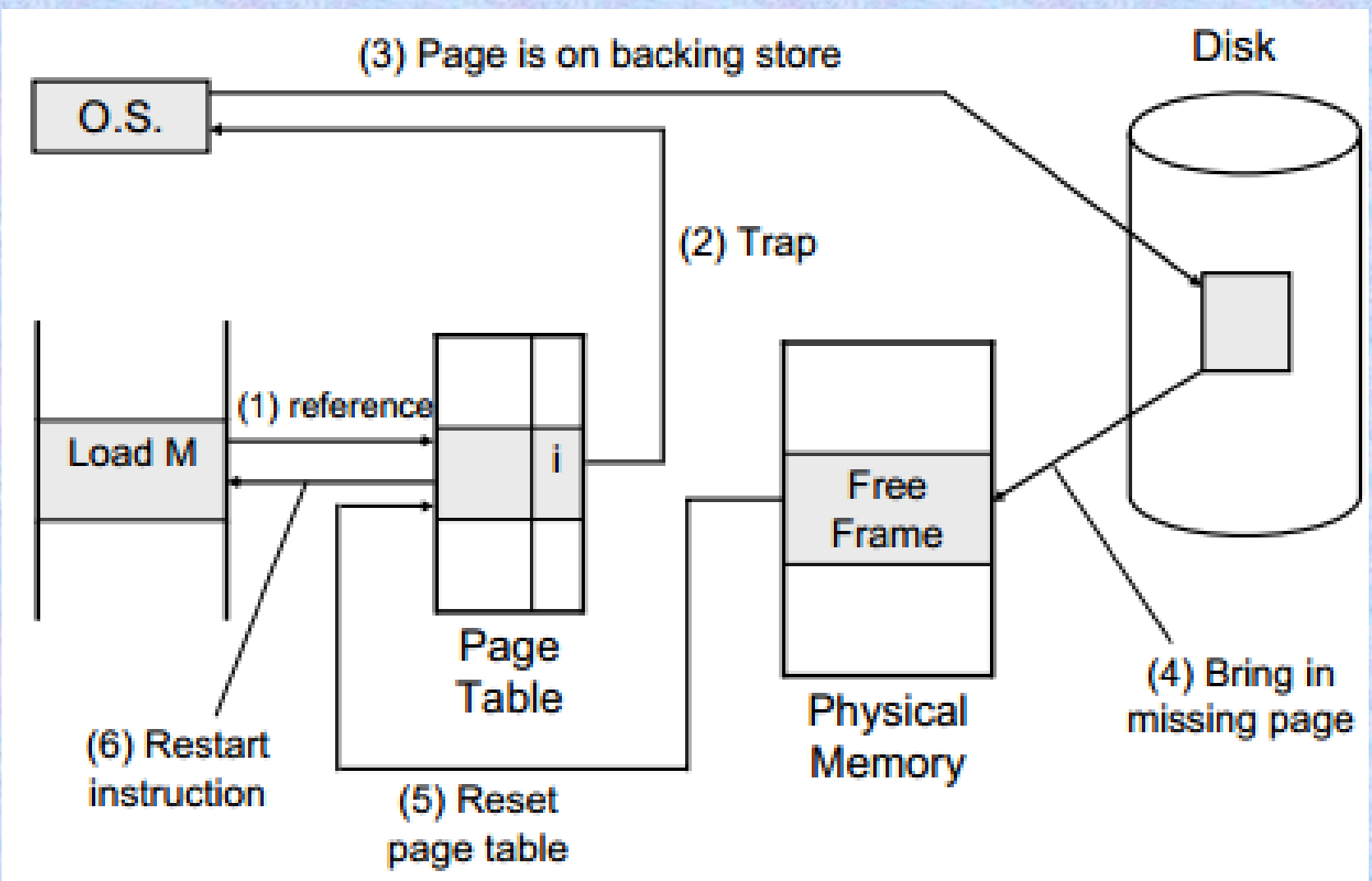
The procedure for handling this page fault

1. We check an internal table for this process, to determine whether the reference was a valid or invalid memory access.
2. If the reference was invalid, we terminate process. If it was valid, but we have not yet brought in that page, we now page in the latter.
3. We find a free frame.

The procedure for handling this page fault

4. We schedule a disk operation to read the desired page into the newly allocated frame.
5. When the disk read is complete, we modify the internal table kept with the process and the page table to indicate that the page is now in memory.
6. We restart the instruction that was interrupted by the illegal address trap.

The procedure for handling this page fault



Performance of Demand Paging

- Page Fault Rate $0 \le p \le 1.0$
  - if $p = 0$ no page faults
  - if $p = 1$, every reference is a fault

  Effective Access Time (EAT)

$$EAT = (1 - p) \times \text{memory access} + p\ (\text{page fault overhead} + \text{swap page out} + \text{swap page in} + \text{restart overhead})$$

Demand Paging Example

- Memory access time = 200 nanoseconds
- Average page-fault service time = 8 milliseconds
  EAT = (1 – p) x 200 + p (8 milliseconds)

  = (1 – p x 200 + p x 8,000,000

  = 200 + p x 7,999,800
- If one access out of 1,000 causes a page fault, then

  EAT = 8.2 microseconds.

  This is a slowdown by a factor of 40!!

Page Replacement Algorithm

- Page replacement algorithms are the techniques using which 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.

Page replacement

When a page fault occurs, the page fault service handling must be modified to include page replacement.

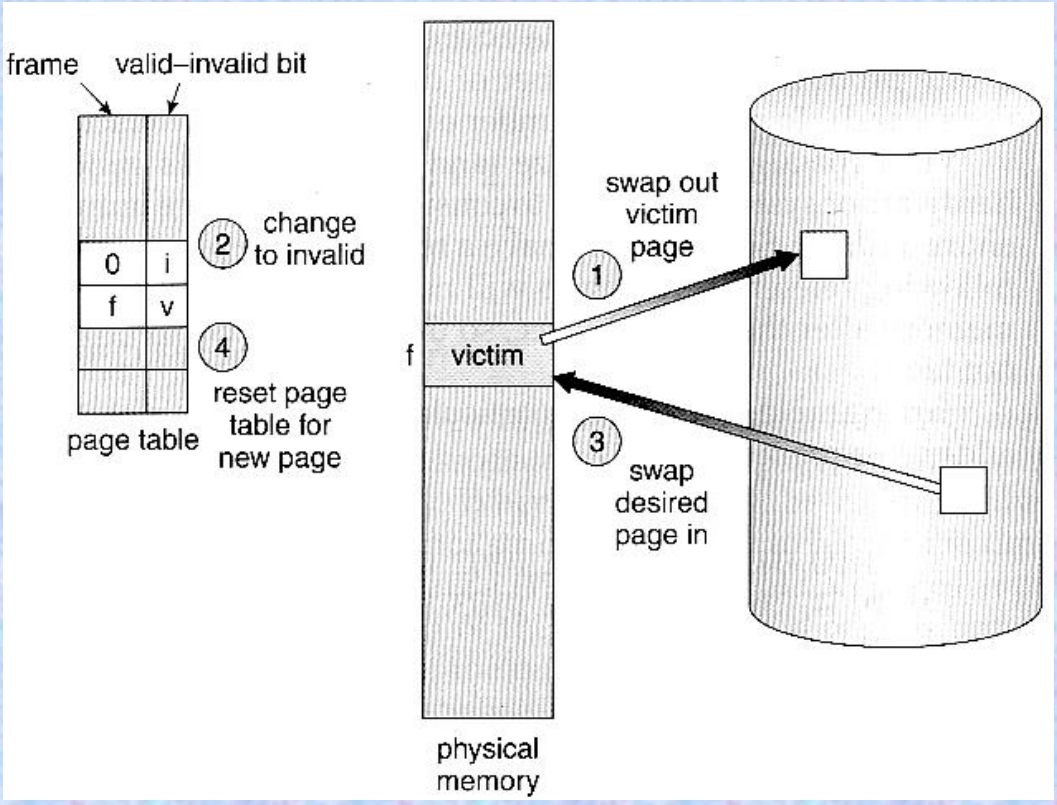
1. Find the location of the desired page on the disk.
2. Find a free frame:
   a) If there is a free frame, use it
   b) If there are no free frames, use a page-Replacement algorithm to select a **victim frame.**
   c) Write the victim frame to the disk, change the pages table.

Page replacement

3. Read the desired page and store it in the free frame. Adjust the page table.
4. Restart the user process.

To implement demand paging, we must develop:

1. Frame-allocation algorithm, to decide how many frames to allocate to each process.
2. Page-replacement algorithm, to select the frames that are to be replaced.



Page replacement algorithm

- There are many different page replacement algorithms. Every page replacement algorithm is operated by the following three operations:

1) Search: To search required pages from main memory.

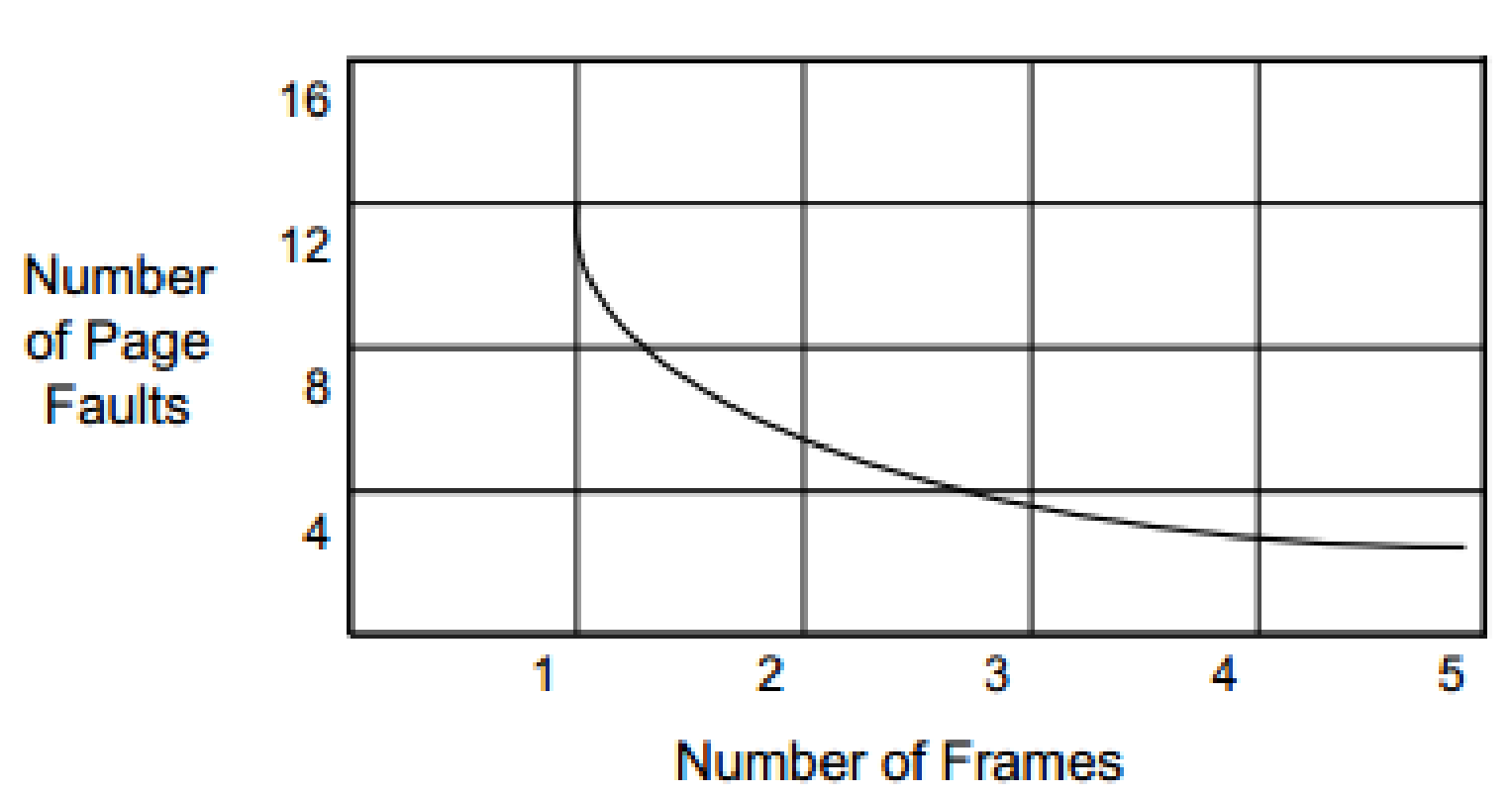
2) Delete: To delete the evict page from main memory.

3) Insert: To insert the page into main memory.

Fewer faults means better performance.

- We select a page replacement algorithm with the lowest page fault rate.
- The string of memory references is called **a reference string.**
- For example the reference string is 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5.

When the number of frames available increases, the number of page faults will decrease as shown in figure. A bad replacement choice increases the page-fault rate and slows process execution.



1) FIFO Algorithm

A FIFO algorithm associates with the time of each page brought into memory. The oldest page is chosen when a page must be replaced,.

Or we can use a FIFO queue, remove pages at queue head and insert at the queue tail.

Example: Consider the following page reference using three frames that are initially empty. Find the page faults using FIFO algorithm?

The Page reference sequence:

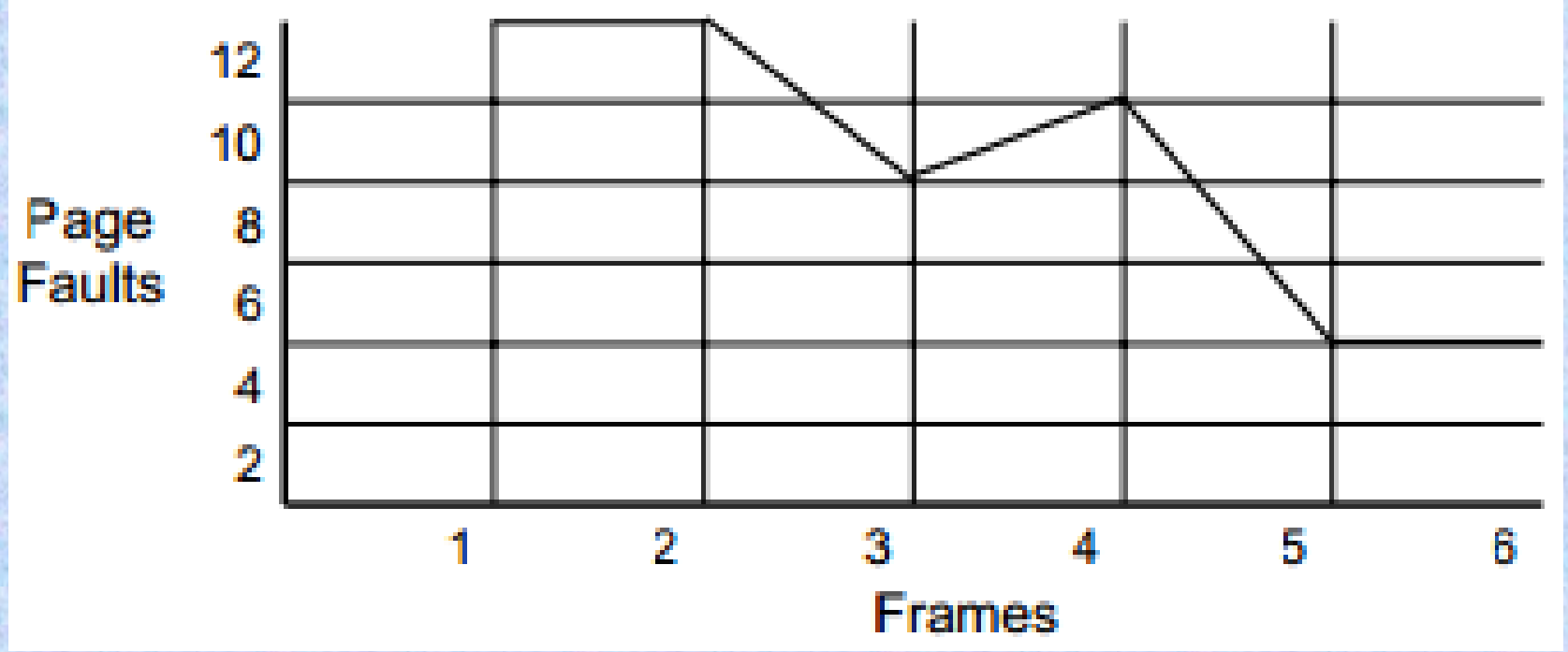
7,0,1, 2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

FIFO Algorithm

7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
7	7	7	0	0	1	2	3	0	4	2	2	2	3	0	0	0	1	2	7
	0	0	1	1	2	3	0	4	2	3	3	3	0	1	1	1	2	7	0
		1	2	2	3	0	4	2	3	0	0	0	1	2	2	2	7	0	2
*	*	*	*		*	*	*	*	*	*			*	*			*	*	*

- The page fault = 15.
- FIFO algorithm is easy to understand and to programs.
- A bad replacement choice increases the page-fault rate and slows process execution.

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



Least-recently-used (LRU) algorithm

In LRU, replace the page that *has not been used* for the longest period of time.

The implementation of LRU replacement requires substantial hardware assistance to determine an order for the frames defined by the time of last use.

Example: Consider the following page reference using three frames that are initially empty. Find the page faults using LRU algorithm, where the page reference sequence: 7,0,1, 2, 0,3, 0,4, 2,3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1?

Answer

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	2	2	4	4	4	0	0	0	1	1	1	1	1	1	1
	0	0	0	0	0	0	0	0	3	3	3	3	3	3	0	0	0	0	0
		1	1	1	3	3	3	2	2	2	2	2	2	2	2	2	7	7	7

Thanks for
Your attention