Ch8

Virtual Memory

By

Lecturer: Ameen A.Noor

Virtual Memory: Basic Concepts

- Virtual memory is a technique that allows the execution of processes that may not be completely in memory.
- One major advantage of this scheme is that programs can be larger than physical memory.
- There are two types of addresses in virtual memory systems: those referenced by processes (virtual addresses) and those available in main memory (physical or real addresses).

Virtual memory systems contain special-purpose hardware called the memory management unit (MMU) that quickly maps virtual addresses to physical addresses.

A key to implementing virtual memory systems is how to map virtual addresses to physical addresses as process execute. Dynamic address translation (DAT) mechanisms convert virtual addresses to physical addresses during execution.

Paging

- Paging is a memory management scheme that eliminates the need for contiguous allocation of physical memory.
- This scheme permits the physical address space of a process to be non – contiguous.
- The physical memory is broken into fixed-sized blocks called frames.
- Logical memory is also broken into blocks of the same size called pages.

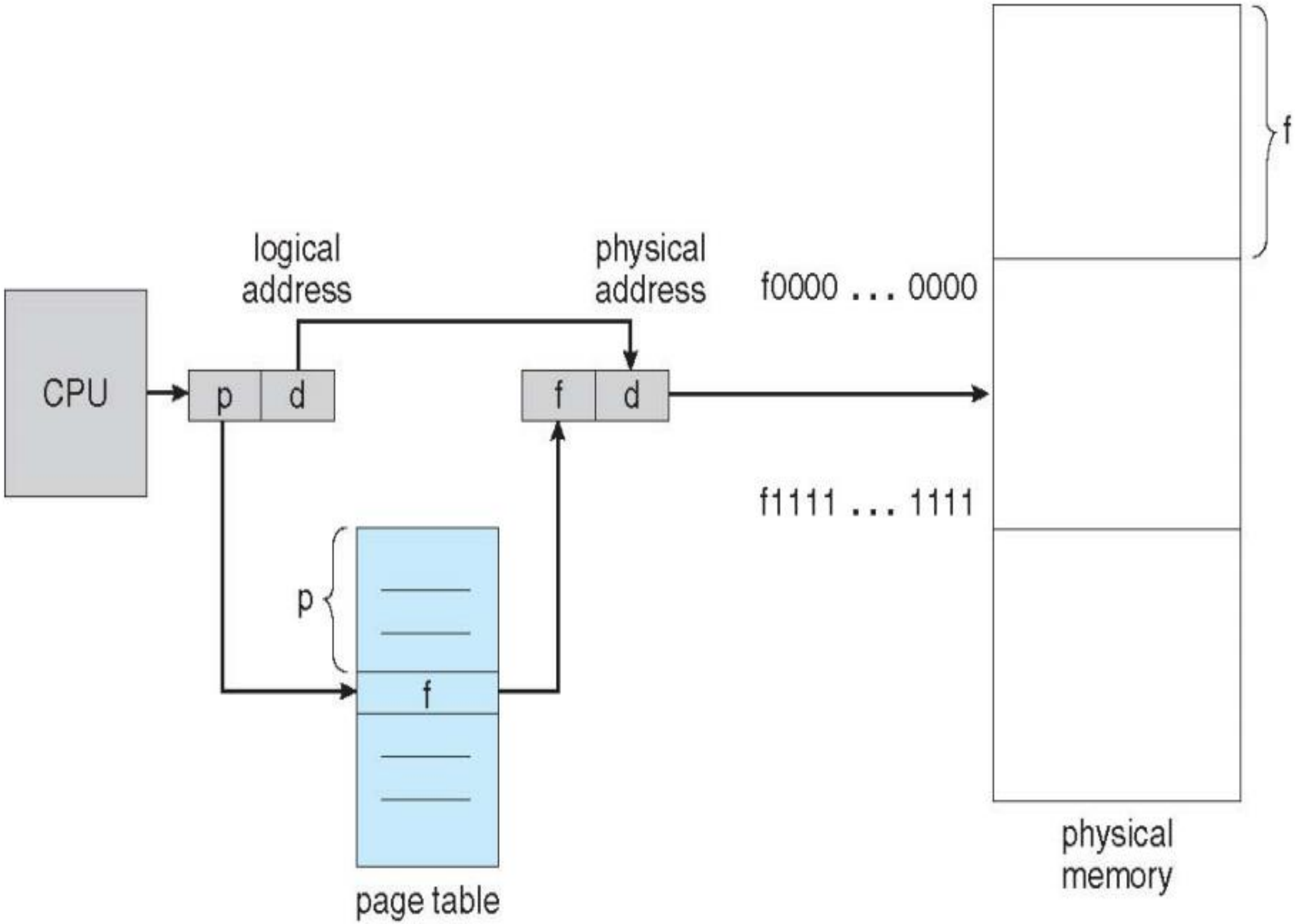
When a process is to be executed, its pages are loaded into any available memory frames from the backing store. The backing store is divided into fixed-sized blocks that are of the same size as the memory frames.

The hardware support for paging is illustrated in Figure (1). Every address generated by the CPU is divided into two parts:

Page number (p): Number of bits required to represent the pages in Logical Address Space.

Page offset (d): Number of bits required to represent particular word in a page or page size of Logical Address Space

The page number is used as an index into a page table. The page table contains the base address of each page in physical memory. This base address is combined with the page offset to define the physical memory address that is sent to the memory unit.

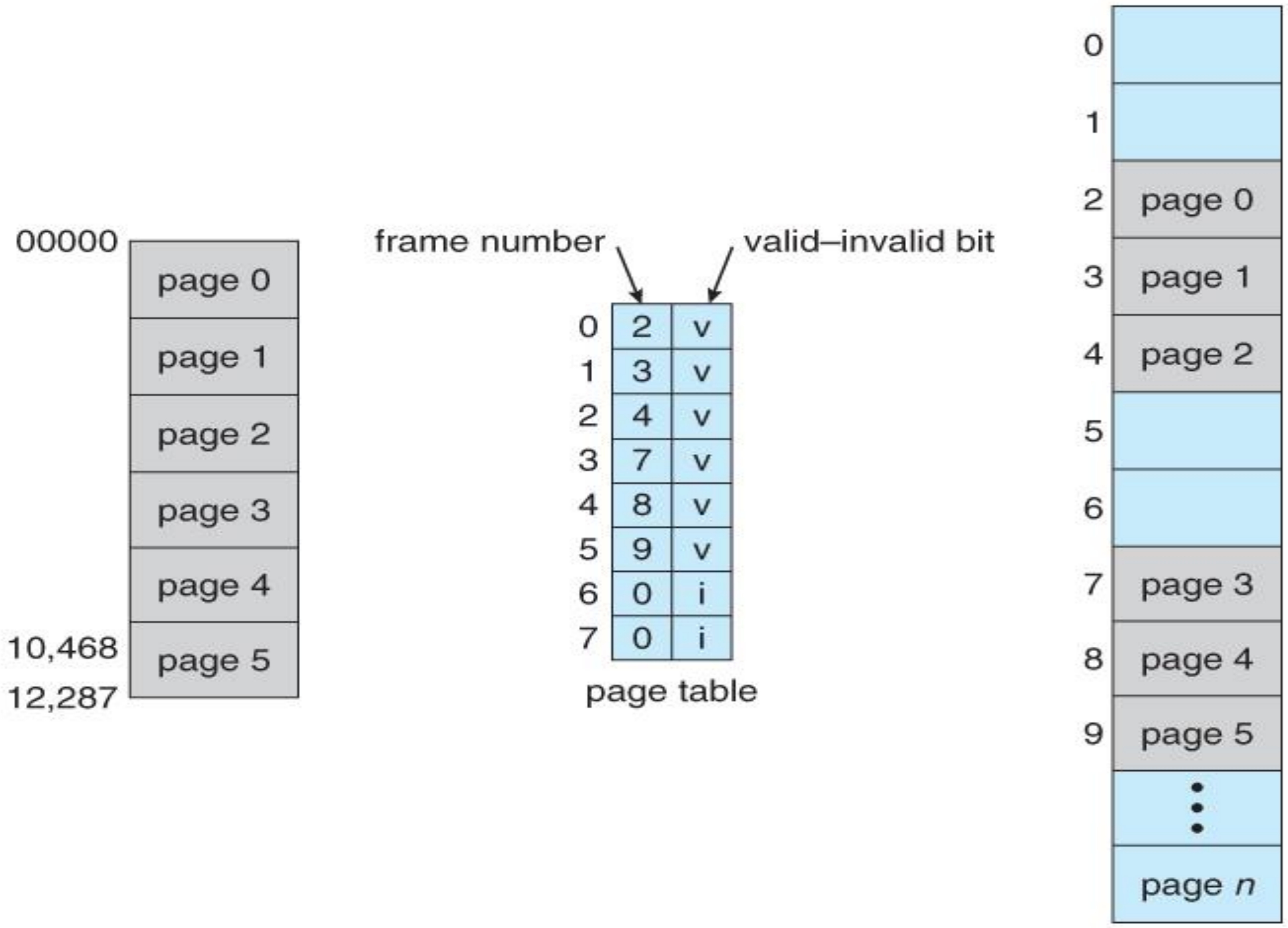

CPU
logical address
p
d
physical address
f
d
f0000 … 0000
f1111 … 1111
f
p
f
page table
physical memory

Protection

- Memory protection in a paged environment is accomplished by protection bits that are associated with each frame.
- these bits are kept in the page table. One bit can define a page to be read-write or read-only.
- Every reference to memory goes through the page table to find the correct frame number. At the same time that the physical address is being computed, the protection bits can be checked to verify that no writes are being made to a read-only page.
- An attempt to write to a read-only page causes a hardware trap to the operating system

a valid-invalid bit

- When this bit is set to "valid," this value indicates that the associated page is in the process' logical-address space, and is thus a legal (or valid) page. If the bit is set to "invalid", this value indicates that the page is not in the process' logical-address space. Illegal addresses are trapped by using the valid-invalid bit. The operating system sets this bit for each page to allow or disallow accesses to that page.



00000
page 0
page 1
page 2
page 3
page 4
10,468
page 5
12,287
frame number
valid–invalid bit
0 2 v
1 3 v
2 4 v
3 7 v
4 8 v
5 9 v
6 0 i
7 0 i
page table
0
1
2 page 0
3 page 1
4 page 2
5
6
7 page 3
8 page 4
9 page 5
page n

Segmentation

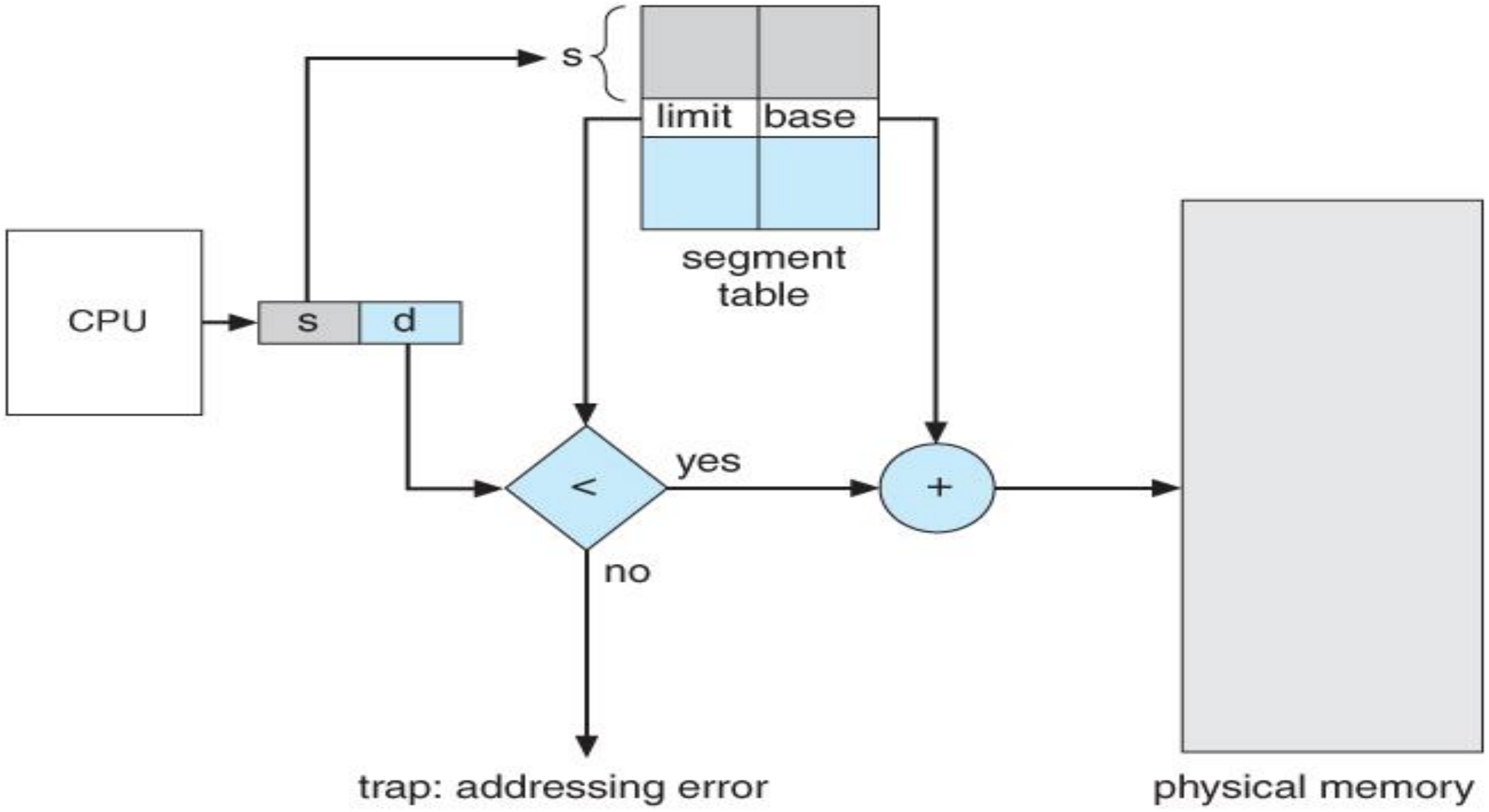
- A Memory Management technique in which memory is divided into variable sized chunks which can be allocated to processes.
- Each chunk is called a **Segment**. Each segment consists of contiguous locations.
- The segments need not be the same size nor must be placed adjacent to one another in main memory.

A table stores the information about all such segments and is called segment table.

It maps two dimensional Logical address into one dimensional Physical address. Each entry of the segment table has a segment base and a segment limit.

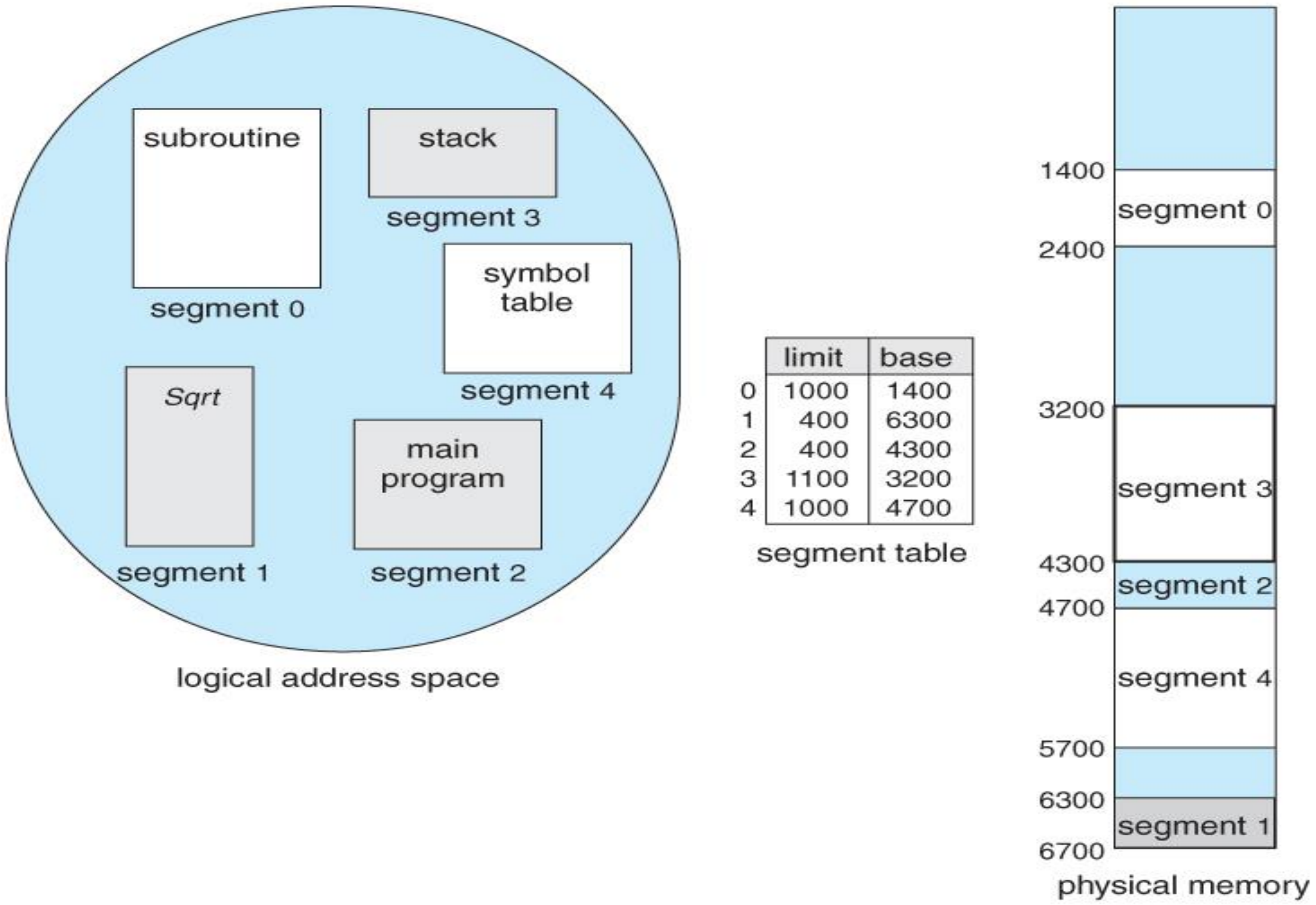
The segment base contains the starting physical address where the segment resides in memory, whereas the segment limit specifies the length of the segment.

Every address generated by the CPU is divided into two parts:
Segment number (s): Number of bits required to represent the segment.
Segment offset (d): Number of bits required to represent the size of th



(Segmentation hardware)

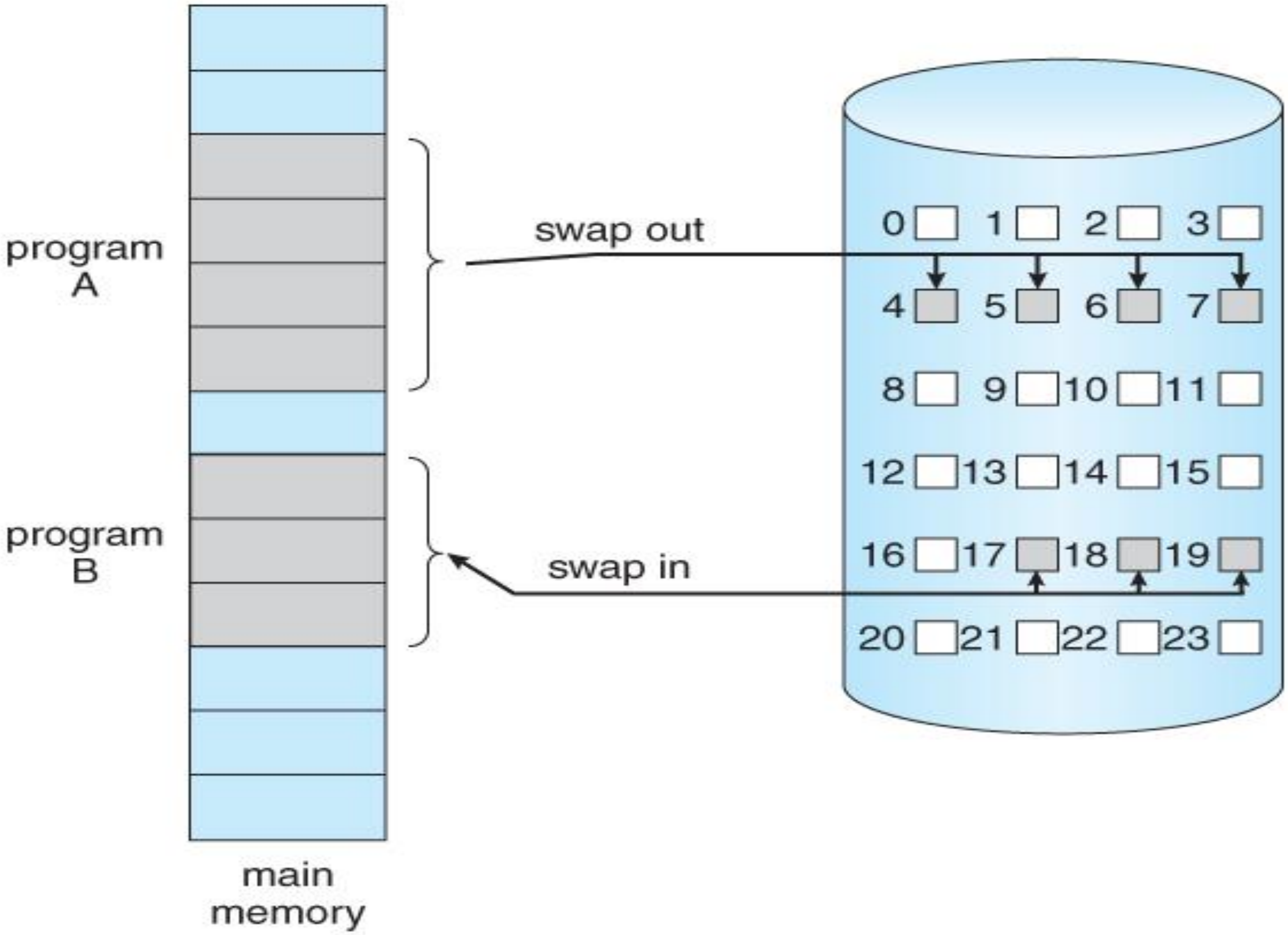
As an example, consider the situation shown in Figure (5). We have five segments numbered from 0 through 4. The segments are stored in physical memory as shown. The segment table has a separate entry for each segment, giving the beginning address of the segment in physical memory (or base) and the length of that segment (or limit). For example, segment 2 is 400 bytes long and begins at location 4300.



	limit	base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

Demand Paging

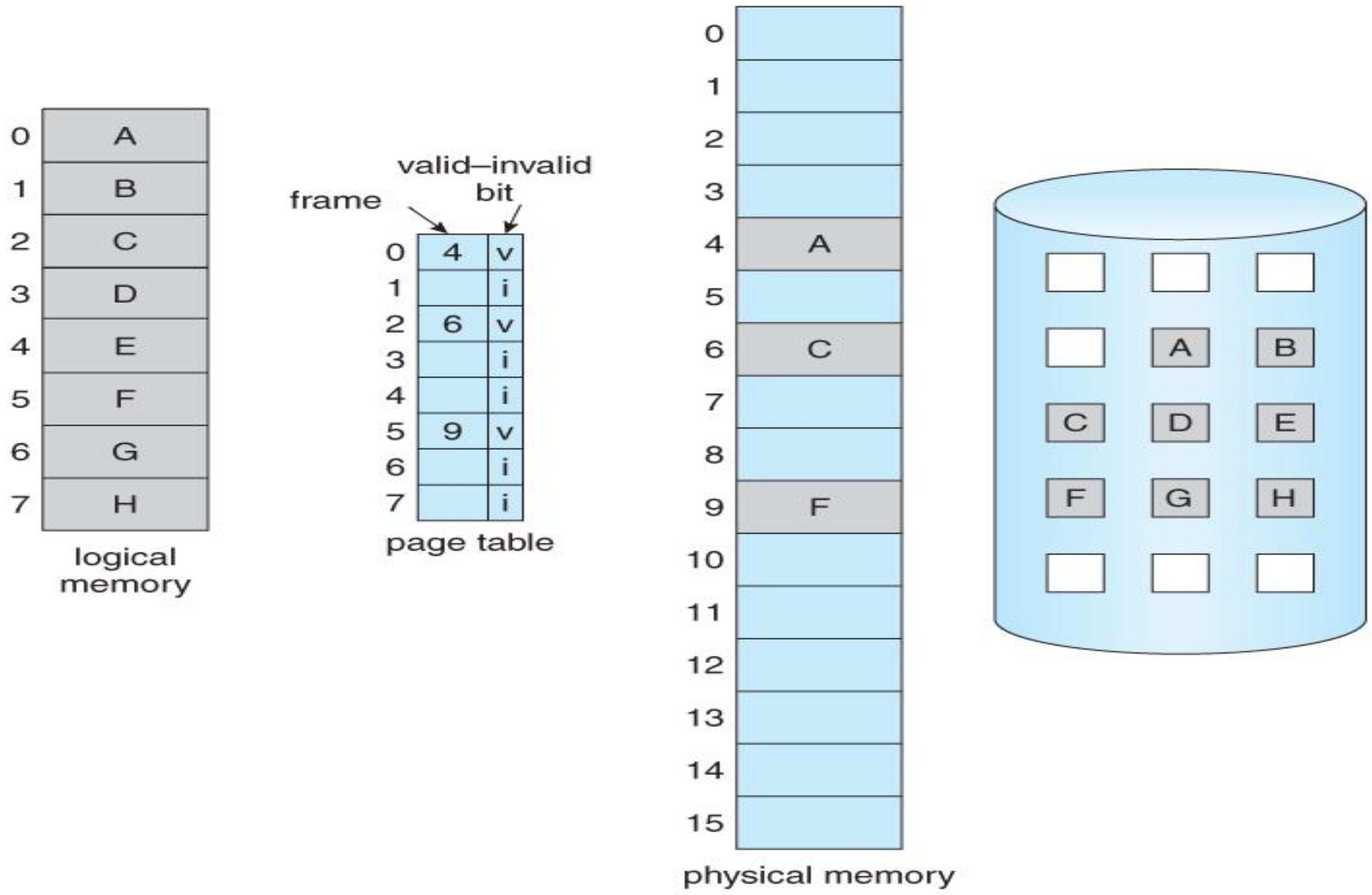
- A demand-paging system is similar to a paging system with swapping .Processes reside on secondary memory (which is usually a disk). When we want to execute a process, we swap it into memory.
- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped 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 anyway, decreasing the swap time and the amount of physical memory needed.


program A
program B
swap out
swap in
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
main memory

With this scheme, we need some form of hardware support to distinguish between those pages that are in memory and those pages that are on the disk. The valid invalid bit scheme can be used for this purpose.
When this bit is set to "valid," this value indicates that the associated page is both legal and in memory. If the bit is set to "invalid," this value indicates that the page either is not valid or is valid but is currently on the disk.
The page-table entry for a page that is brought into memory is set as usual, but the page-table entry for a page that is not currently in memory is simply marked invalid, or contains the address of the page on disk.

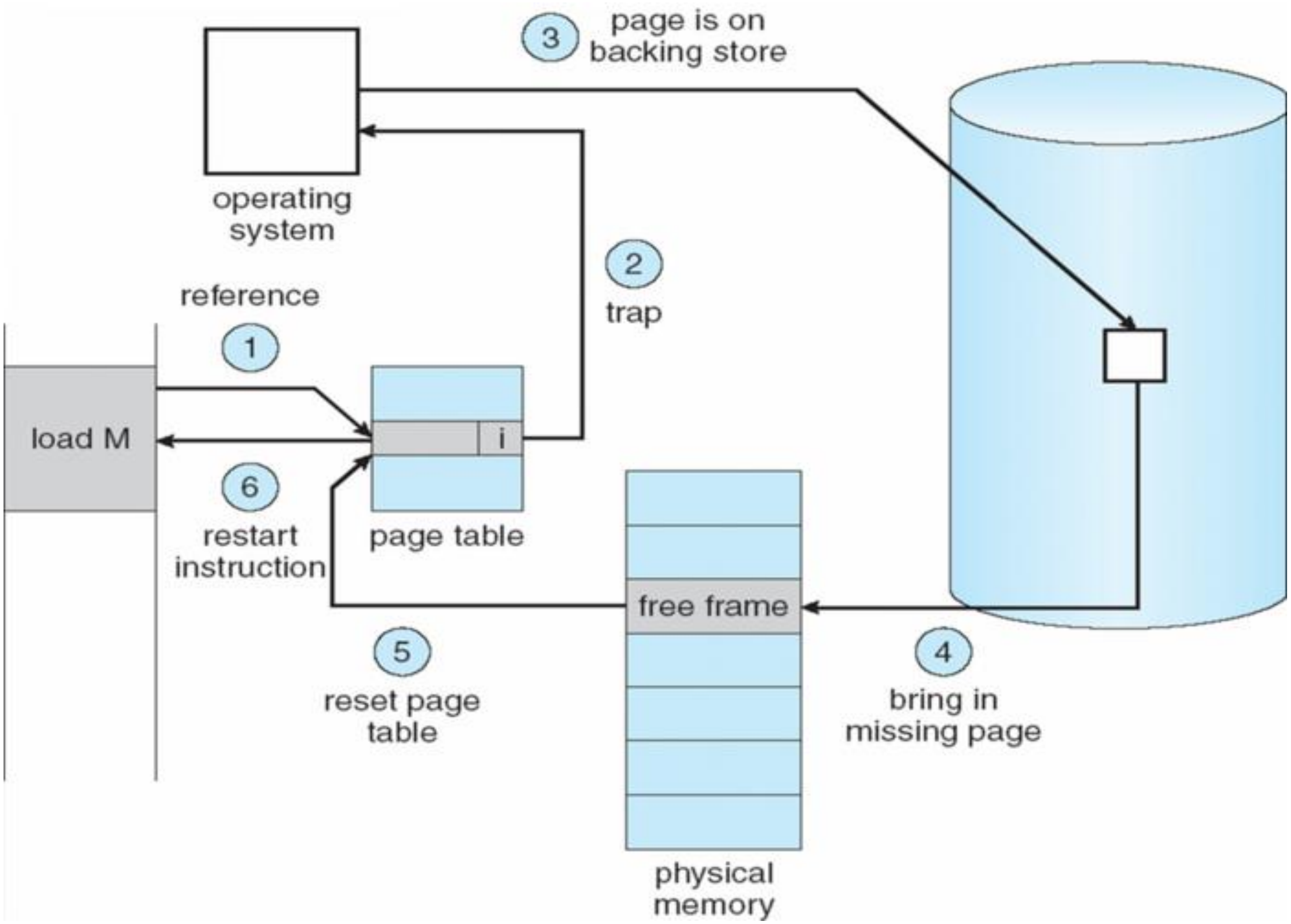
if p = 0 no page faults

if p = 1, every reference is a fault



Page Fault

- When the process tries to access a page that was not brought into memory (access to a page marked invalid) causes a page-fault trap.
- The paging hardware, in translating the address through the page table, will notice that the invalid bit is set, causing a trap to the operating system. This trap is the result of the operating system's failure to bring the desired page into memory,


3 page is on backing store
operating system
2 trap
reference
1
load M
i
page table
6 restart instruction
free frame
5 reset page table
4 bring in missing page
physical memory

The procedure for handling this page fault

1- Check the page table (usually kept with the process control block) for this process, to determine whether the reference was a valid or invalid memory access.

2- If the reference was invalid, a page fault exception is raised.

3- The operating system must locate logical page in secondary memory.

4- Schedule a disk operation to read the desired page and swap into memory into a free frame.

5- When the disk read is complete, we modify the page table to indicate that the page is now in memory (set valid bit).

6- Restart the instruction that was interrupted by the illegal address trap.

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 free frame to hold the page for which the process faulted.

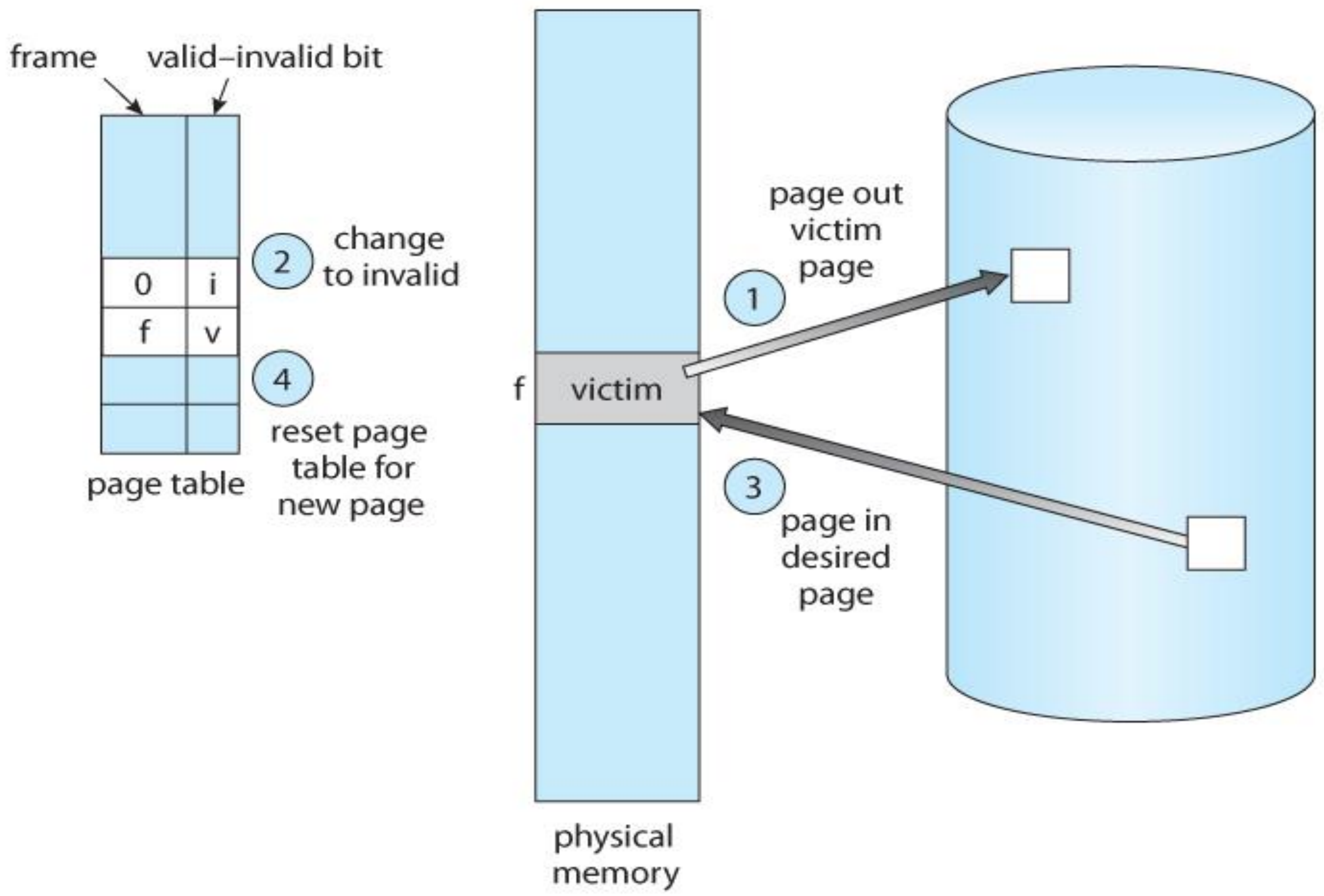
Basic Page Replacement

1. Find the location of the desired page on disk

2. 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

3. Bring the desired page into the (newly) free frame; update the page and frame tables

4. Restart the process


frame
valid–invalid bit
0
i
f
v
2
change to invalid
4
reset page table for new page
page table
f
victim
physical memory
1
page out victim page
3
page in desired page

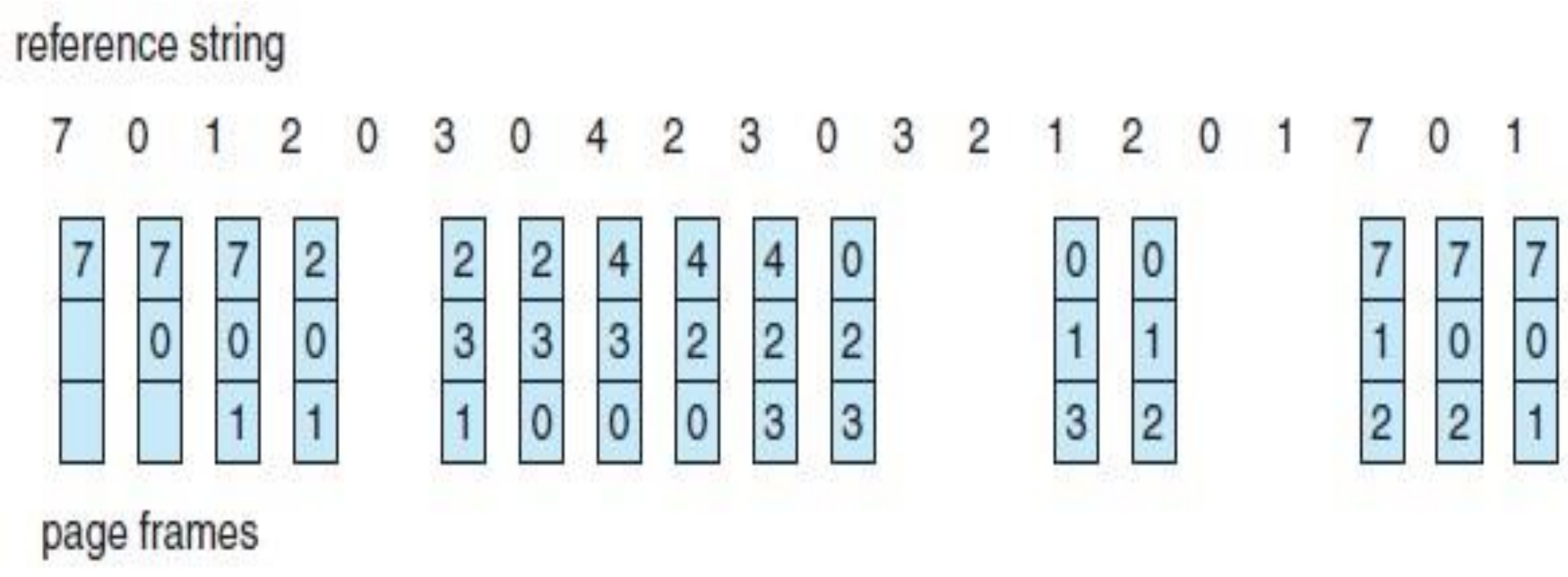
Page Replacement Algorithms

In an operating systems that use paging for memory management, page replacement algorithm are needed to decide which page needed to be replaced when new page comes in. Whenever a new page is referred and not present in memory, page fault occurs and Operating System replaces one of the existing pages with newly needed page. Different page replacement algorithms suggest different ways to decide which page to replace. The target for all algorithms is to reduce number of page faults.

1. FIFO Page Replacement

- 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.
- Notice that it is not strictly necessary to record the time when a page is brought in. 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.

Example: Suppose three pages can be in memory at a time per process.Process references pages: 7,0,1,2, 0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1 , what is the number of page fault?



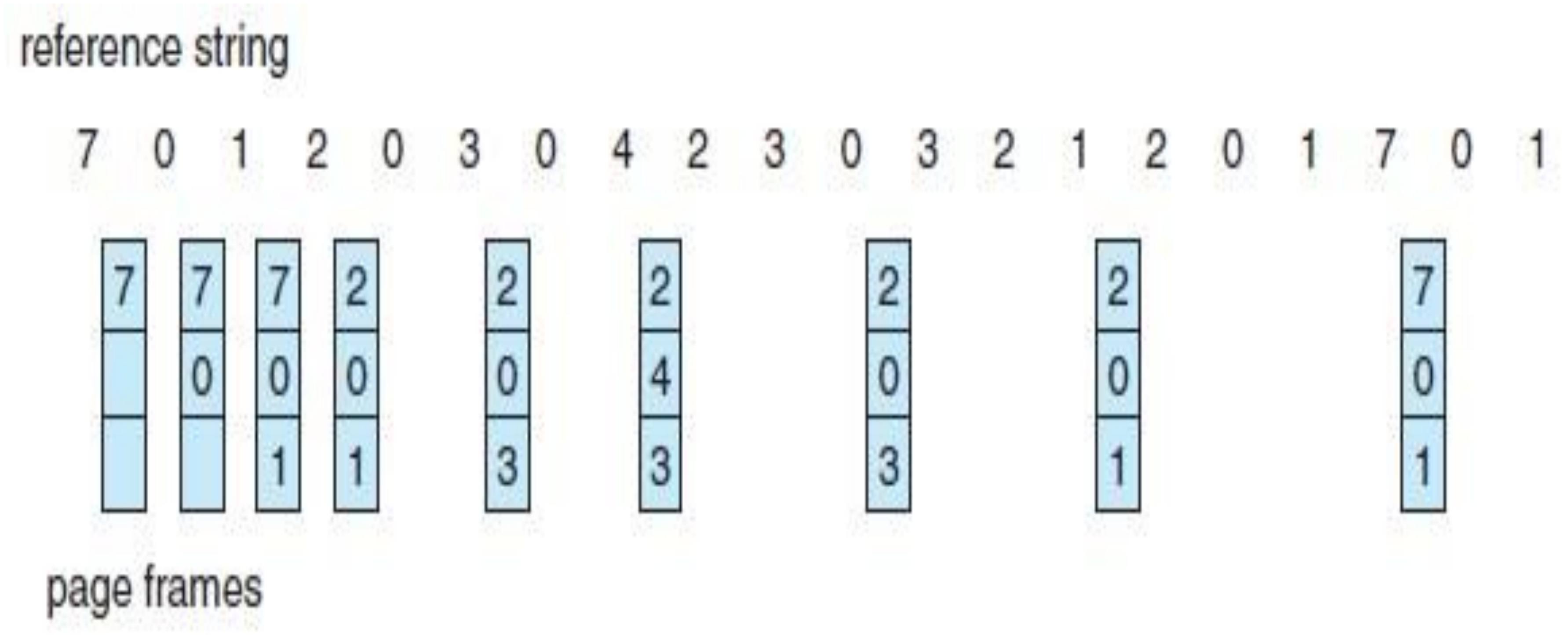
The number of page fault = 15

2. Optimal Page Replacement

- In this algorithm, pages are replaced which are not used for the longest duration of time in the future.
- Use of this page-replacement algorithm guarantees the lowest possible page fault rate for a fixed number of frames.

Example: Suppose we have the following process references pages:7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

What is the number of page fault if we use three page frames?

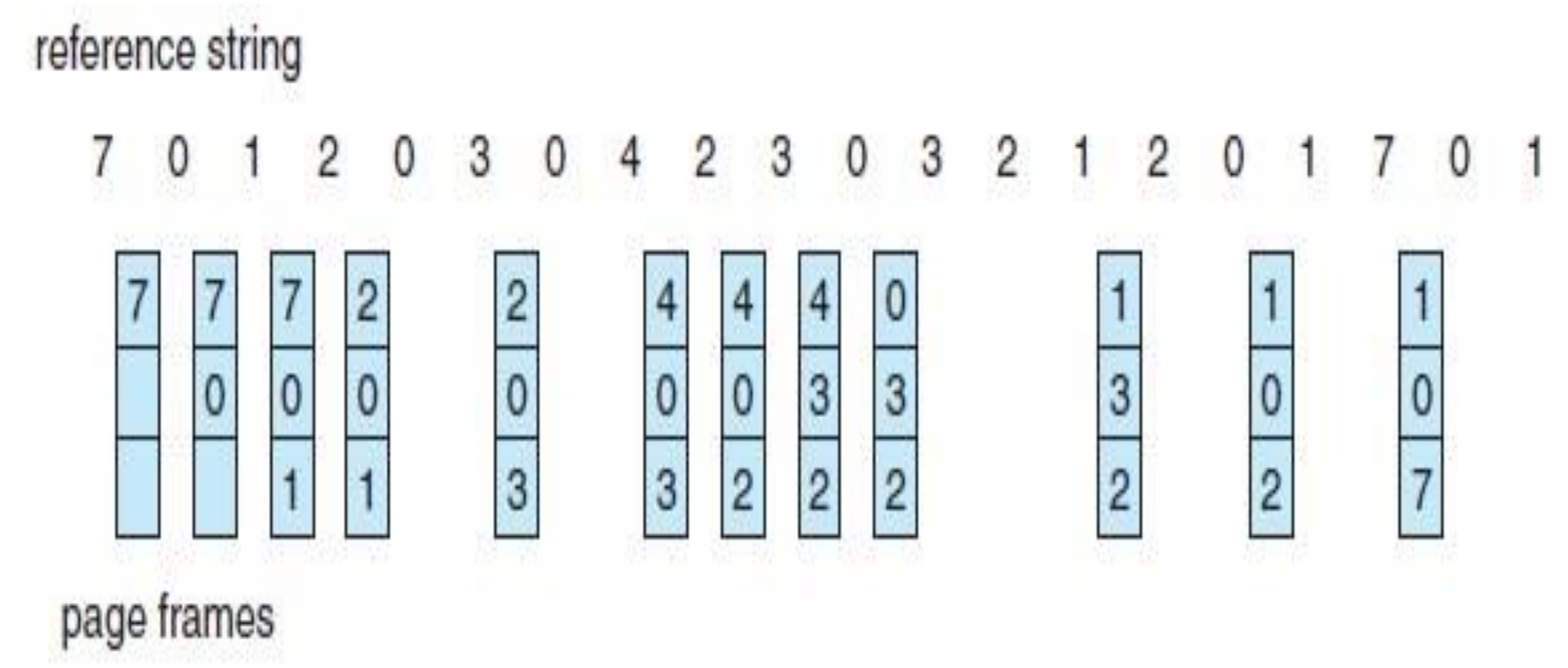


The number of page fault = 9

3. LRU Page Replacement

- In least recently used (LRU) replacement algorithm page will be replaced which is least recently used.
- When a page must be replaced, LRU chooses that page that has not been used for the longest period of time.

Example: Suppose three pages can be in memory at a time per process. Process references pages: 7,0,1,2,0,3,0,4,2,3,0,3, 2,1,2,0,1,7,0,1 , what is the number of page fault?



The number of page fault = 12