Virtual Memory

by

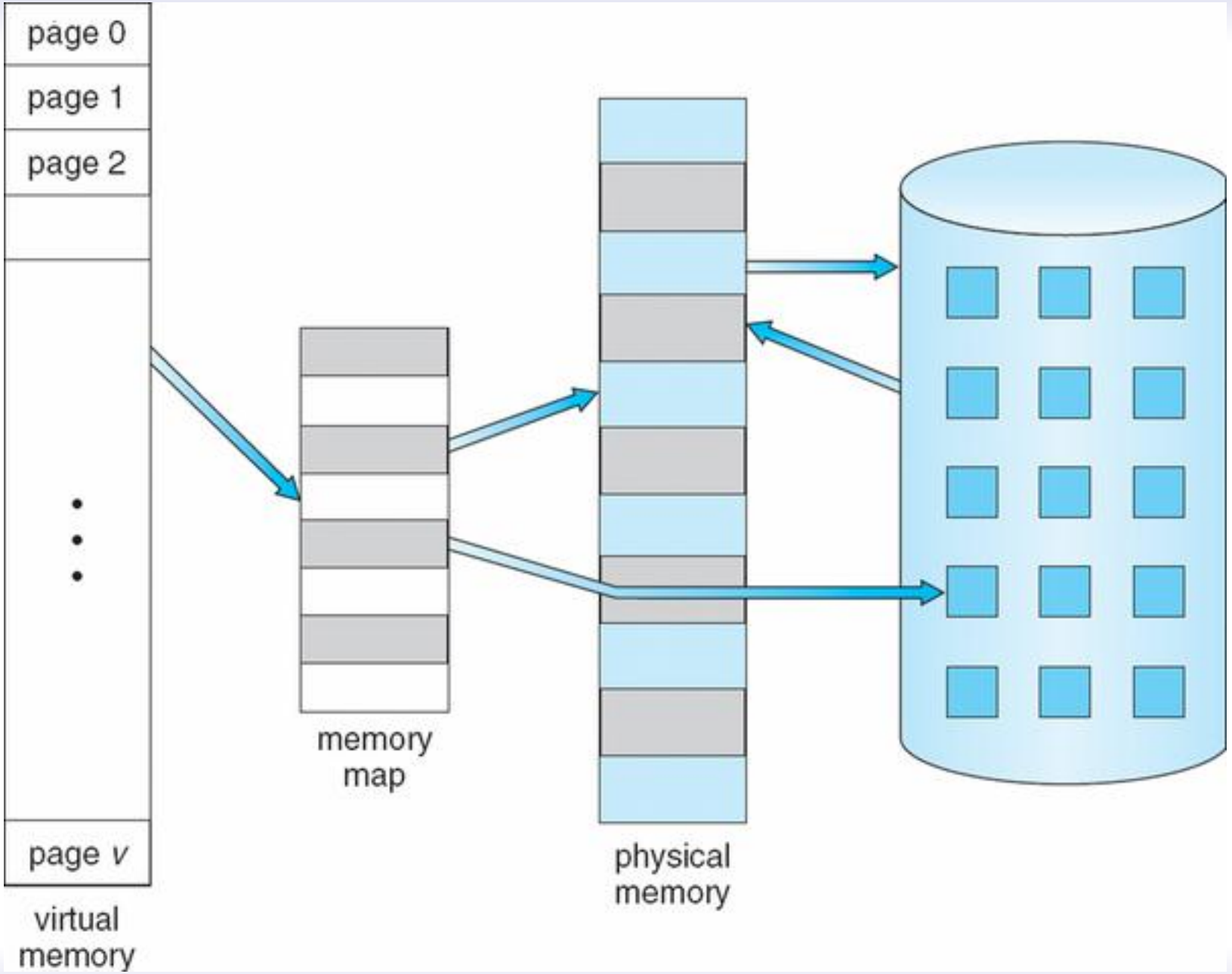
Asst. Lec. Amal A. Maryoosh

Virtual Memory

- **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:
  1. virtual addresses: those referenced by processes.
  2. physical or real addresses: those available in main memory.

Virtual Memory

- A key to implementing virtual memory systems is how to map virtual addresses to physical addresses.
- Virtual memory systems contain special-purpose hardware called the **memory management unit (MMU)** that quickly maps virtual addresses to real addresses.
- **Dynamic address translation (DAT)** mechanisms convert virtual addresses to physical addresses during execution.

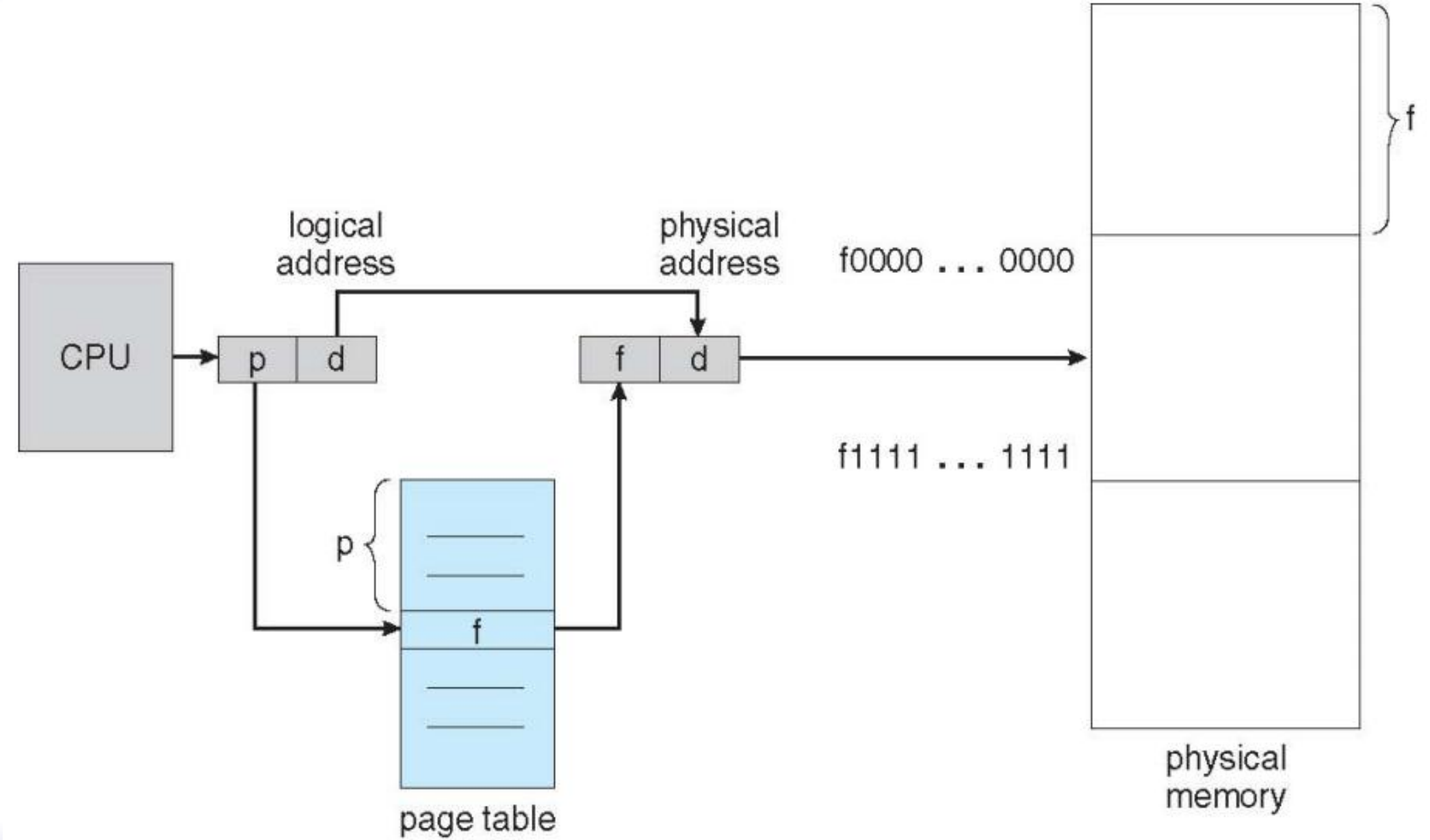

page 0
page 1
page 2
page v
virtual memory
memory map
physical memory

Paging

- **Paging** is a memory management scheme that eliminates the need for contiguous allocation of physical memory.
- The physical memory is broken into fixed-sized blocks called **frames**.
- Logical memory is also broken into blocks of the same size called **pages**.
- 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.

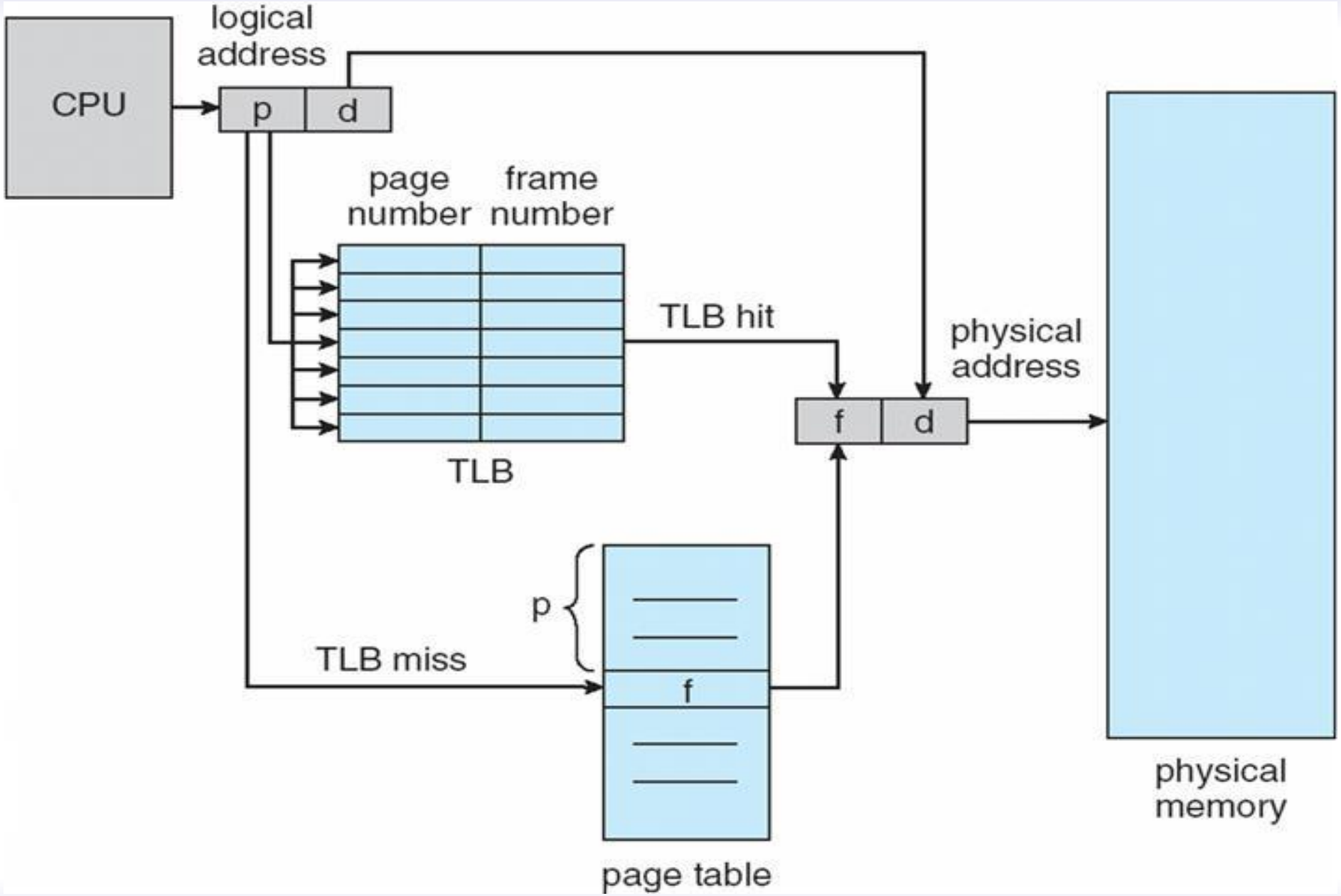
Paging

- 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.
- When we use a paging scheme, we have no external fragmentation: Any free frame can be allocated to a process that needs it. However, we may have some internal fragmentation.


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

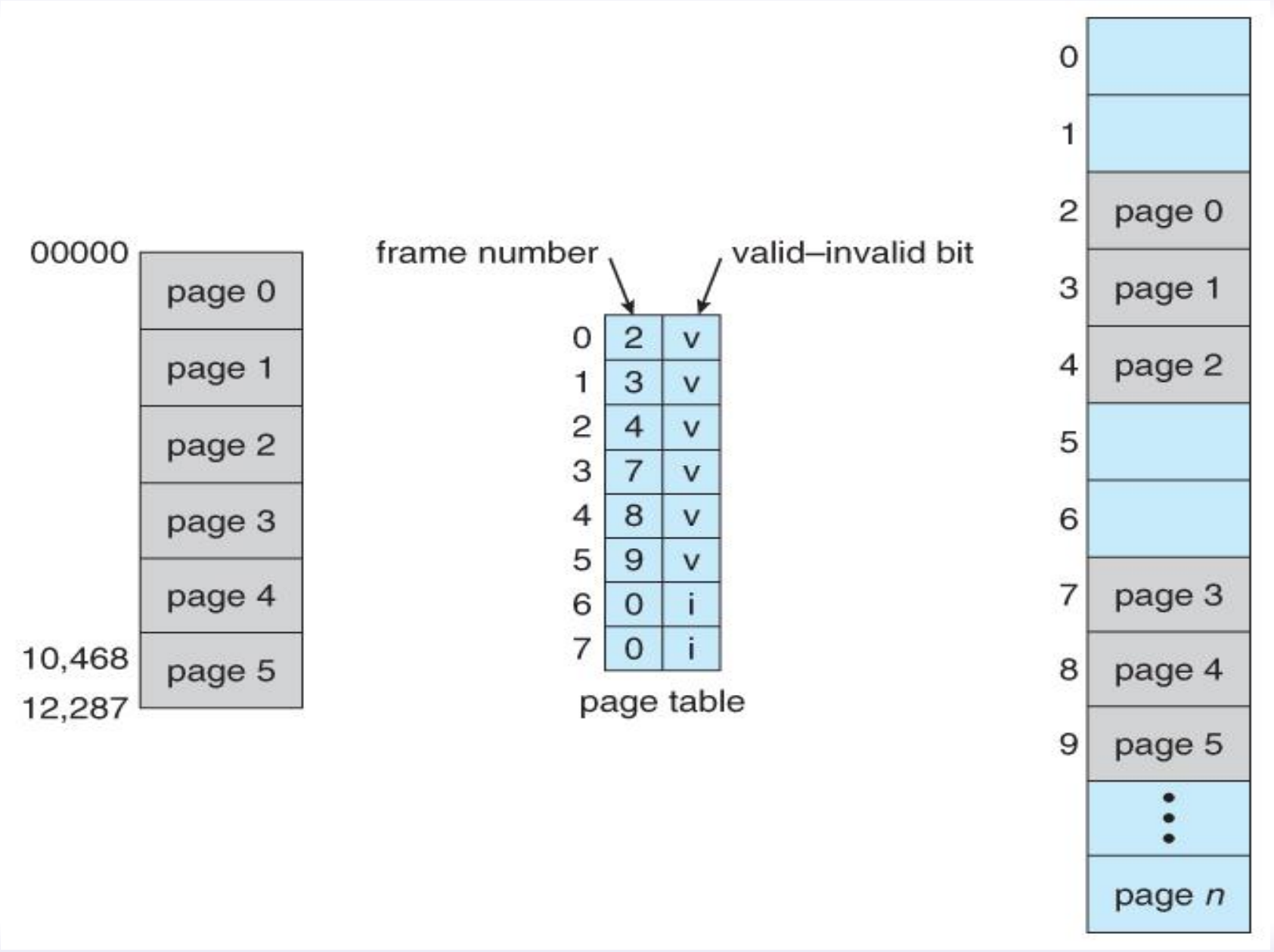
Page Table

- The hardware implementation of page table can be done by using dedicated registers.
- the usage of register for the page table is suitable only if page table is small. If page table contain large number of entries then we can use **TLB (translation Look-aside buffer).**
- **TLB** is a special, small, fast look up hardware cache.


CPU
logical address
p
d
page number
frame number
TLB hit
physical address
f
d
TLB
p
TLB miss
f
page table
physical memory

Protection

- Memory protection in a paged environment is accomplished by **protection bits** that are associated with each frame.
- One bit is generally attached to each entry in the page table: 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.


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

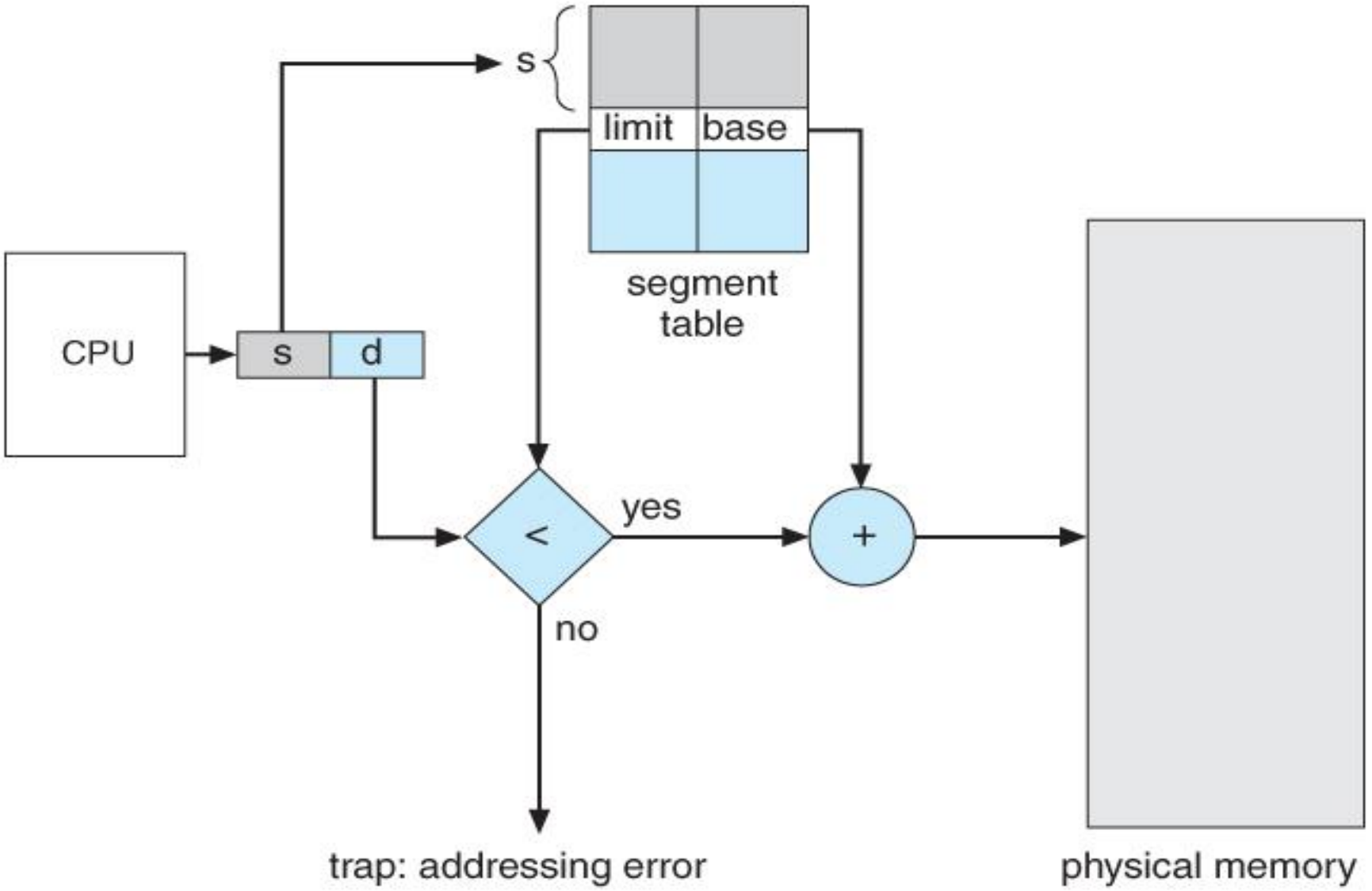
Segmentation

- 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**.

Segmentation

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

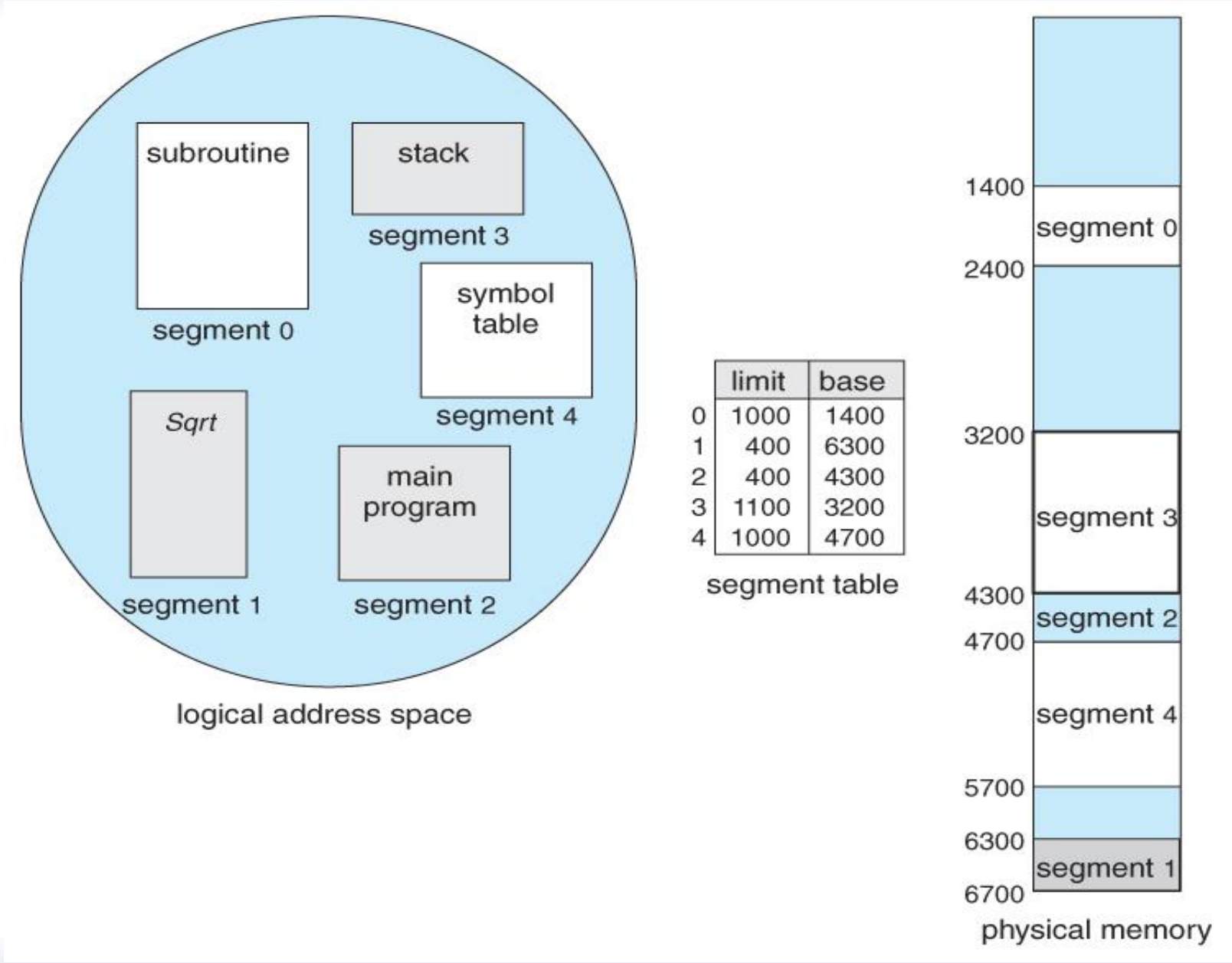
- Every address generated by the CPU is divided into two parts:
  - **Segment number (s):** Number of bits required to represent the segment.
  - **Segment offset (d):** Number of bits required to represent the size of the segment.


s
limit
base
segment table
CPU
s
d
<
yes
+
no
trap: addressing error
physical memory

Segmentation

Example:

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.


subroutine
stack
segment 3
symbol table
segment 0
segment 4
Sqrt
main program
segment 1
segment 2
logical address space
limit
base
0 1000 1400
1 400 6300
2 400 4300
3 1100 3200
4 1000 4700
segment table
1400
segment 0
2400
3200
segment 3
4300
segment 2
4700
segment 4
5700
6300
segment 1
6700
physical memory

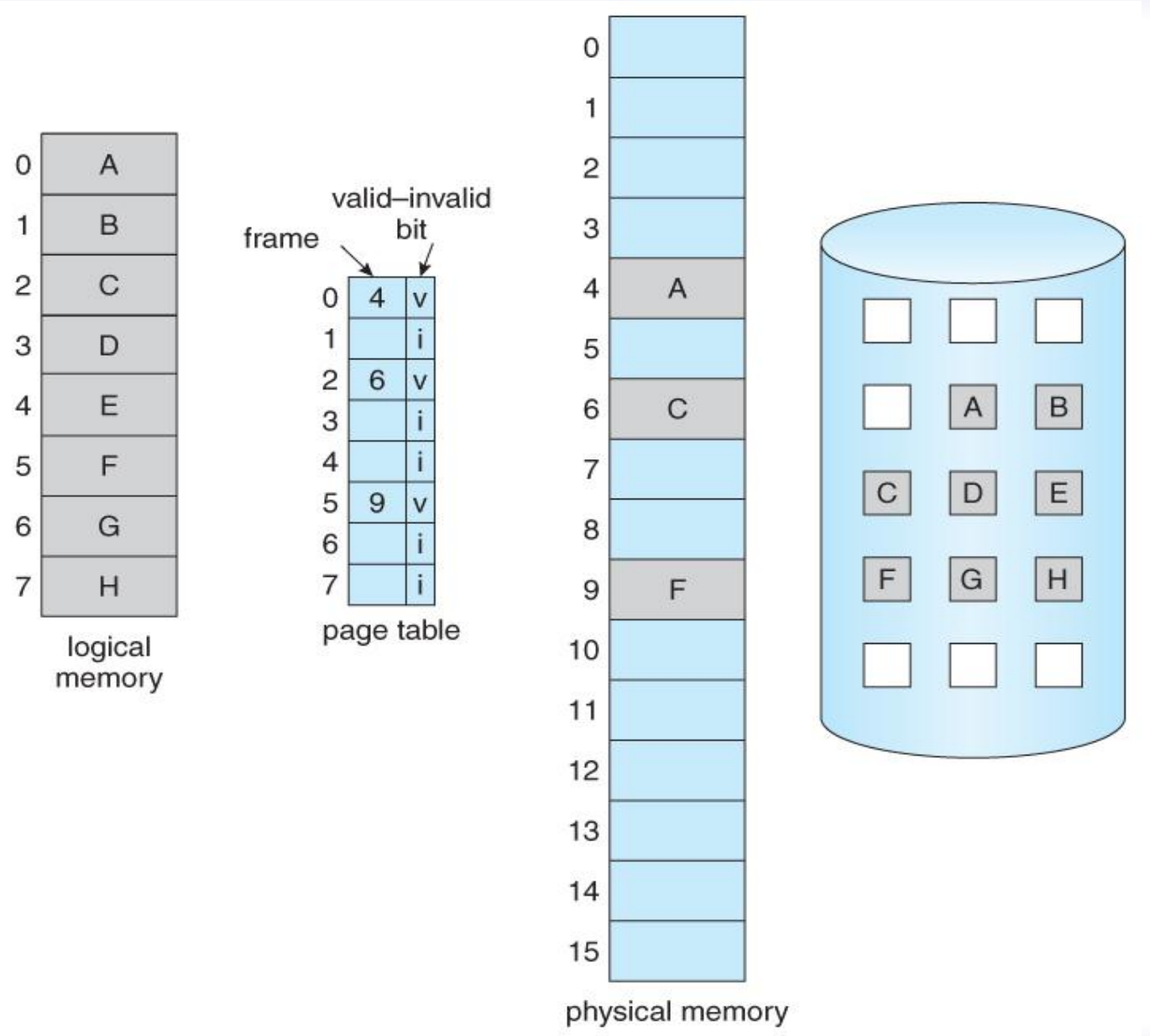
Segmentation

Example:

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.

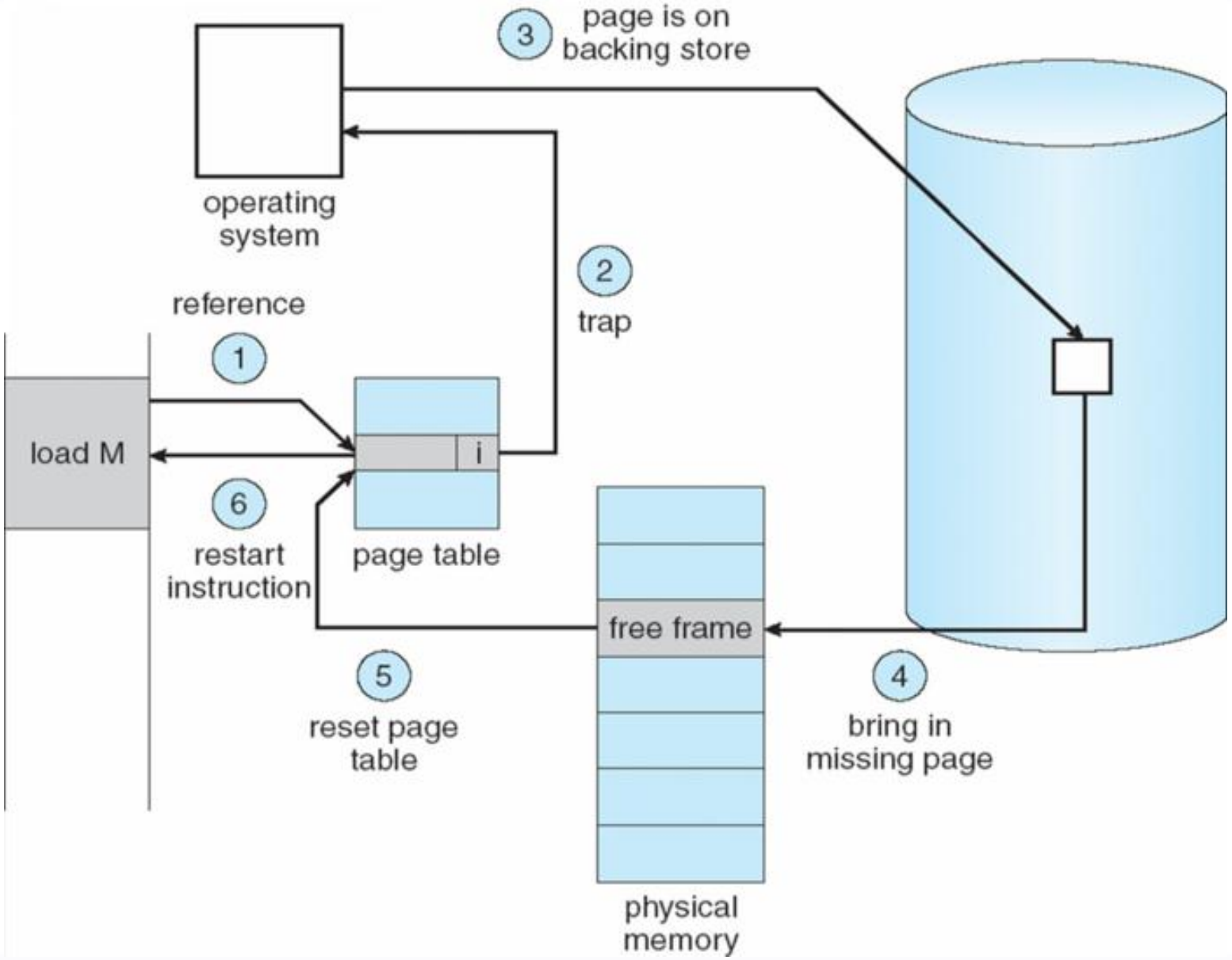
Demand Paging

- 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 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 (that is, not in the logical address space of the process), or is valid but is currently on the disk.


0 A
1 B
2 C
3 D
4 E
5 F
6 G
7 H
logical memory
frame
valid–invalid bit
0 4 v
1 i
2 6 v
3 i
4 i
5 9 v
6 i
7 i
page table
0
1
2
3
4 A
5
6 C
7
8
9 F
10
11
12
13
14
15
physical memory
A B
C D E
F G H

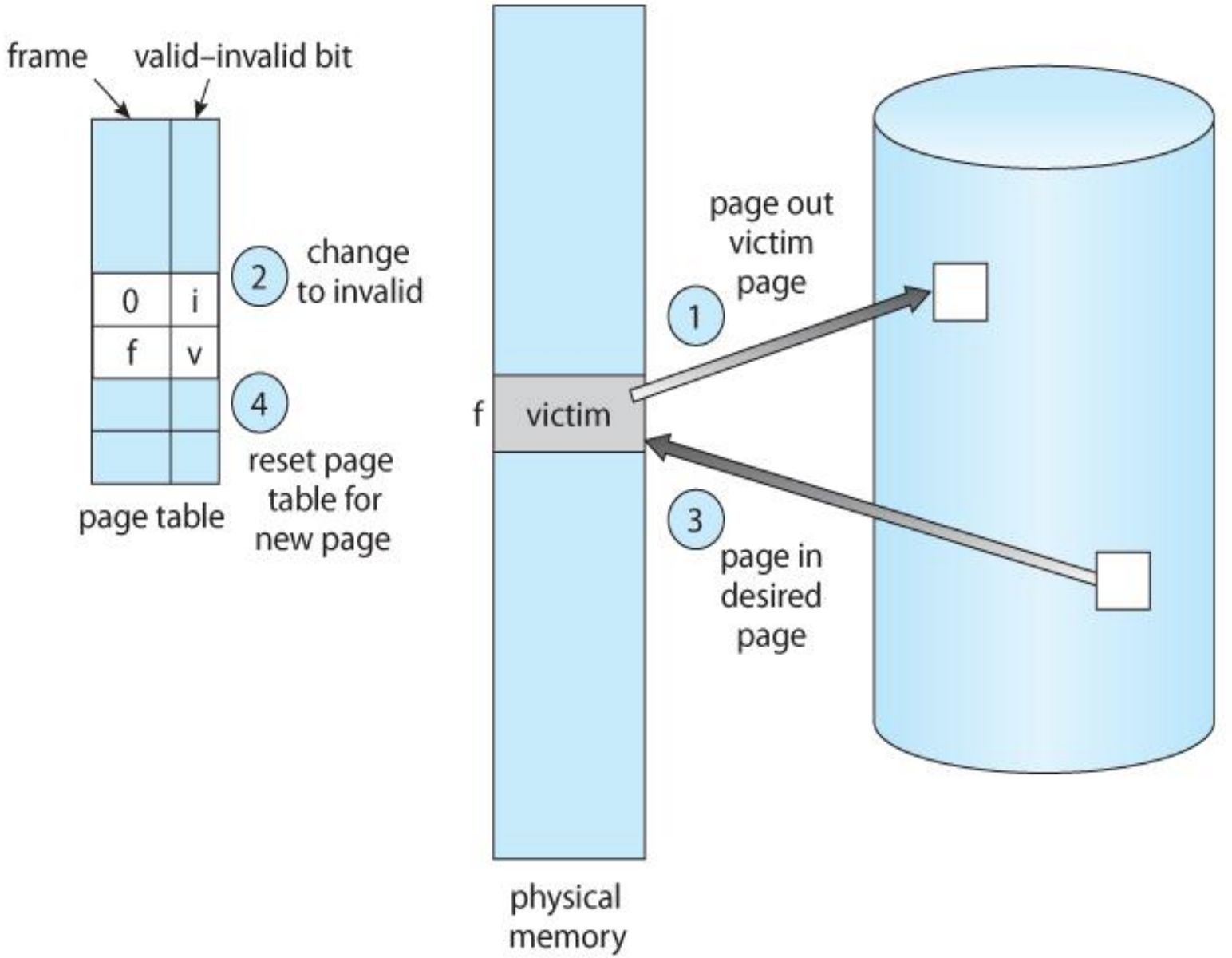
Page Fault

- 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**.
- invalid bit causing a **trap** to the operating system.


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

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.

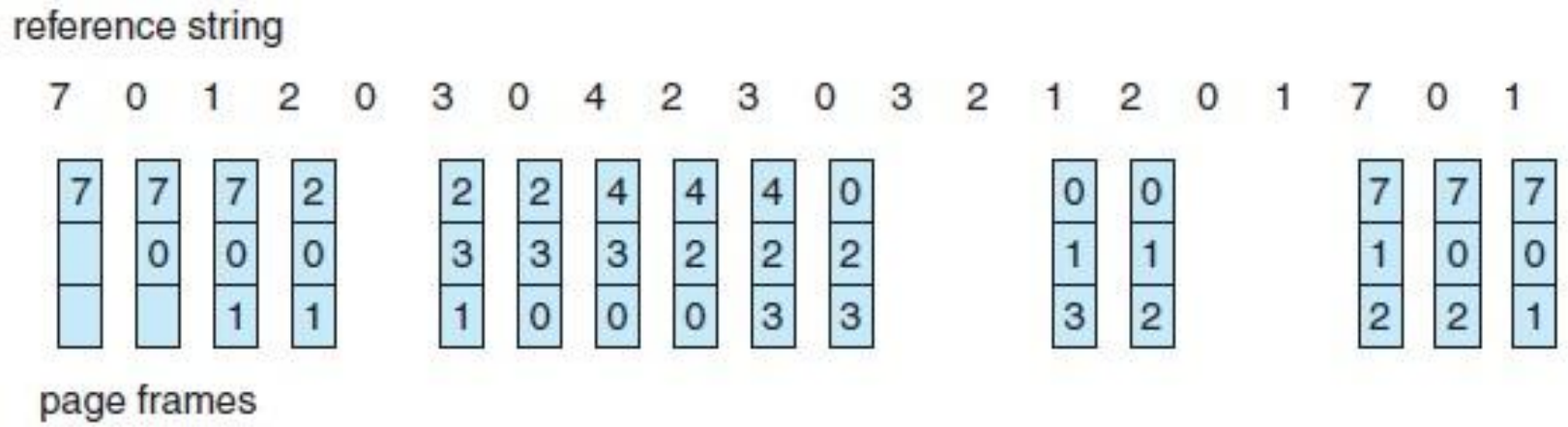

frame
valid–invalid bit
0
i
f
v
page table
2
change
to invalid
4
reset page
table for
new page
f
victim
1
page out
victim
page
3
page in
desired
page
physical
memory

Page Replacement Algorithms

- 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**.

FIFO Page Replacement

- **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?



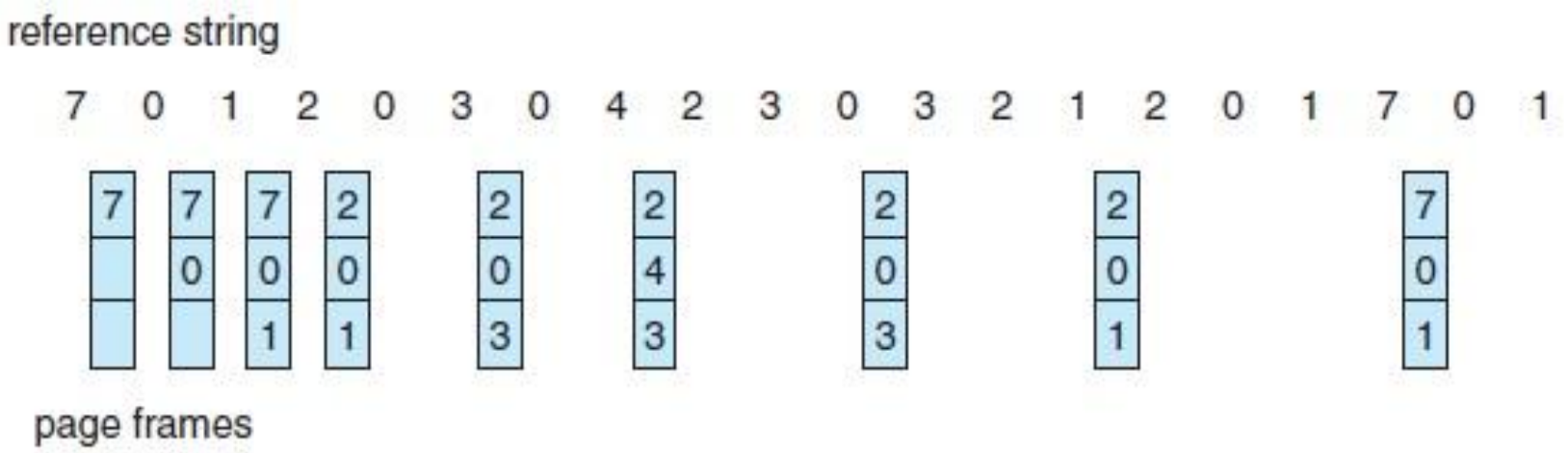
- The number of page fault = 15

Optimal Page Replacement

- **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

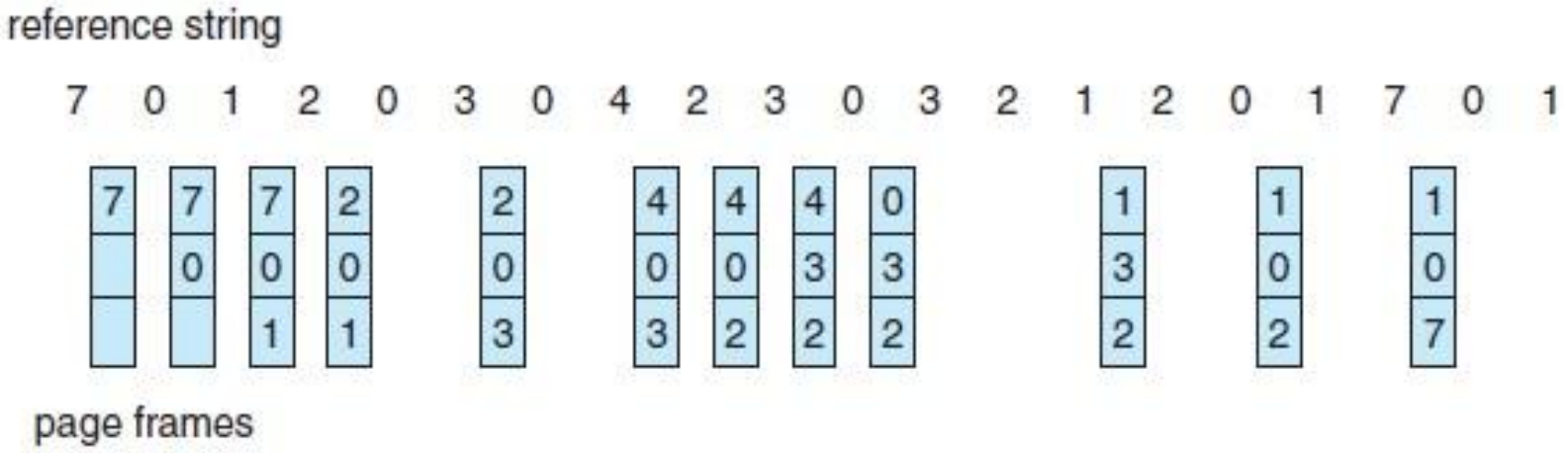
  What is the number of page fault if we use three page frames?



- The number of page fault = 9

LRU Page Replacement

- **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