VIRTUAL MEMORY

Virtual memory

- Virtual memory permits the user to construct programs as though a large memory space were available.
- It gives the programmer an illusion that they have a very large memory.
- It provides a mechanism for translating program generated addresses into correct main memory locations by means of mapping table.

Virtual Memory

- Virtual address – address used by a programmer
- Address space - Set of virtual addresses
- Physical address – address in main memory
- Memory space – set of physical addresses
- Memory mapping - process of translating virtual addresses into physical addresses

Paged Virtual Memory

- **Block** - Organization of memory space.
- **Page** – blocks of contiguous virtual memory addresses - Organization of address space .

- **Page table-** a data structure used for translating virtual addresses (seen by application) to physical addresses (used by hardware) to process instructions.

- The hardware that handles this specific translation is called as **Memory Management Unit** (MMU).
- Pages are moved from auxiliary memory to main memory in records equal to size of a page.

Page Table & Address Translation

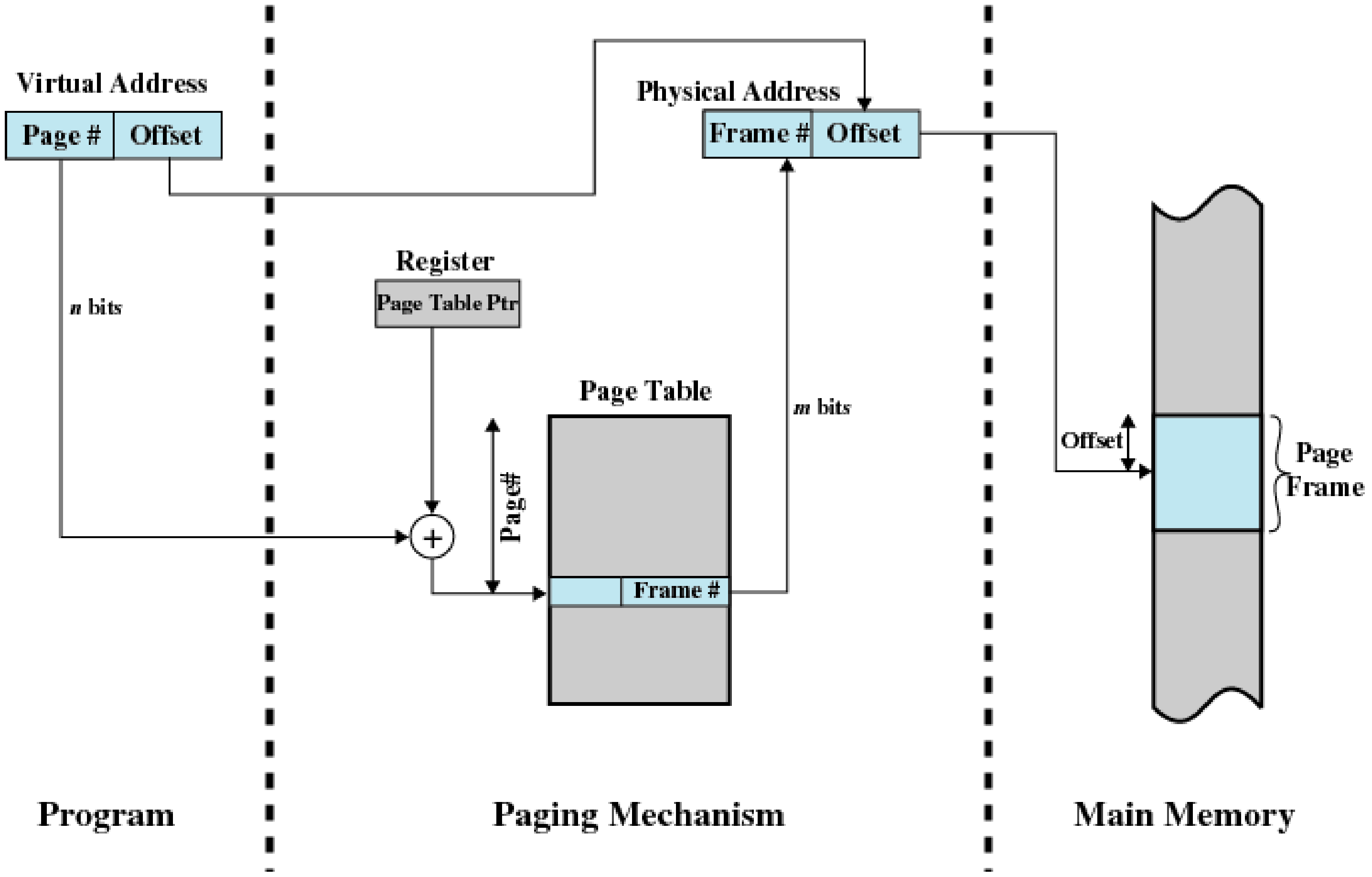
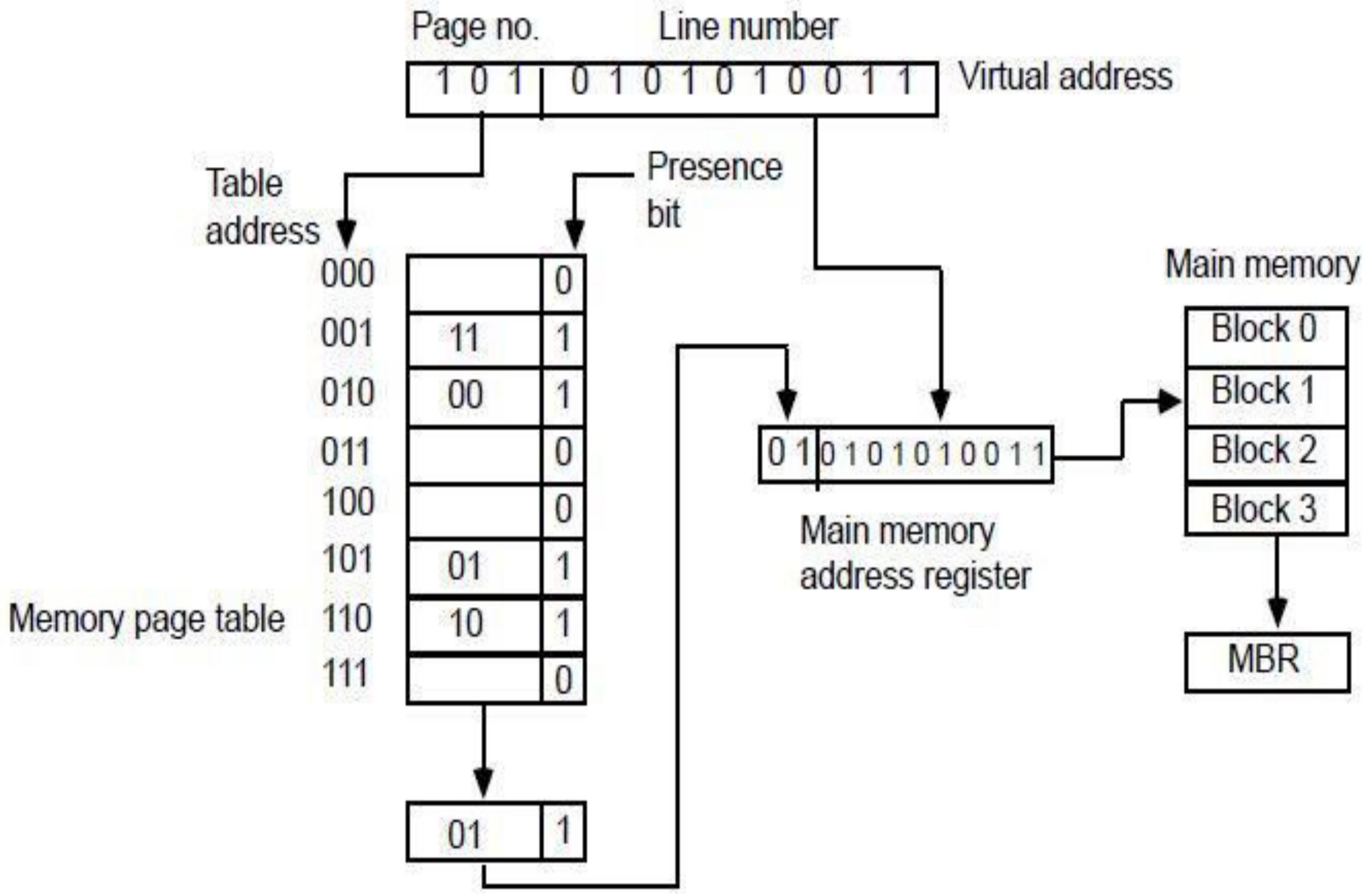


Figure 8.3 Address Translation in a Paging System

Page table



Page no.
Line number
1 0 1 | 0 1 0 1 0 1 0 0 1 1
Virtual address
Table address
Presence bit
000
001
010
011
100
101
110
111
11
00
01
10
0
1
1
0
0
1
1
0
Memory page table
01 | 1
0 1 | 0 1 0 1 0 1 0 0 1 1
Main memory address register
Main memory
Block 0
Block 1
Block 2
Block 3
MBR

Page table

- Assume that
- Number of Blocks in memory = m
- Number of Pages in Virtual Address Space = n
- Page Table
  - Straight forward design -> n entry table in memory
  - Inefficient storage space utilization => n-m entries of the table is empty
- More efficient method is m-entry Page Table
- Page Table made of an Associative Memory - m words; (Page Number: Block Number)
- Page Fault - Page number cannot be found in the Page Table

Translation Look-aside Buffer

- Each virtual memory reference can cause two physical memory accesses
  - One to fetch the page table
  - One to fetch the data
- To overcome this problem a high-speed cache is set up for page table entries
  - Called a Translation Look-aside Buffer (TLB)
  - TLB Contains page table entries that have been most recently used

Translation Look-aside Buffer

- Given a virtual address, processor examines the TLB
- If page table entry is present (TLB hit), the frame number is retrieved and the real address is formed
- If page table entry is not found in the TLB (TLB miss), the page number is used to index the process page table

Translation Look-aside Buffer

- First checks if page is already in main memory
  - If not in main memory a page fault is issued
- The TLB is updated to include the new page entry

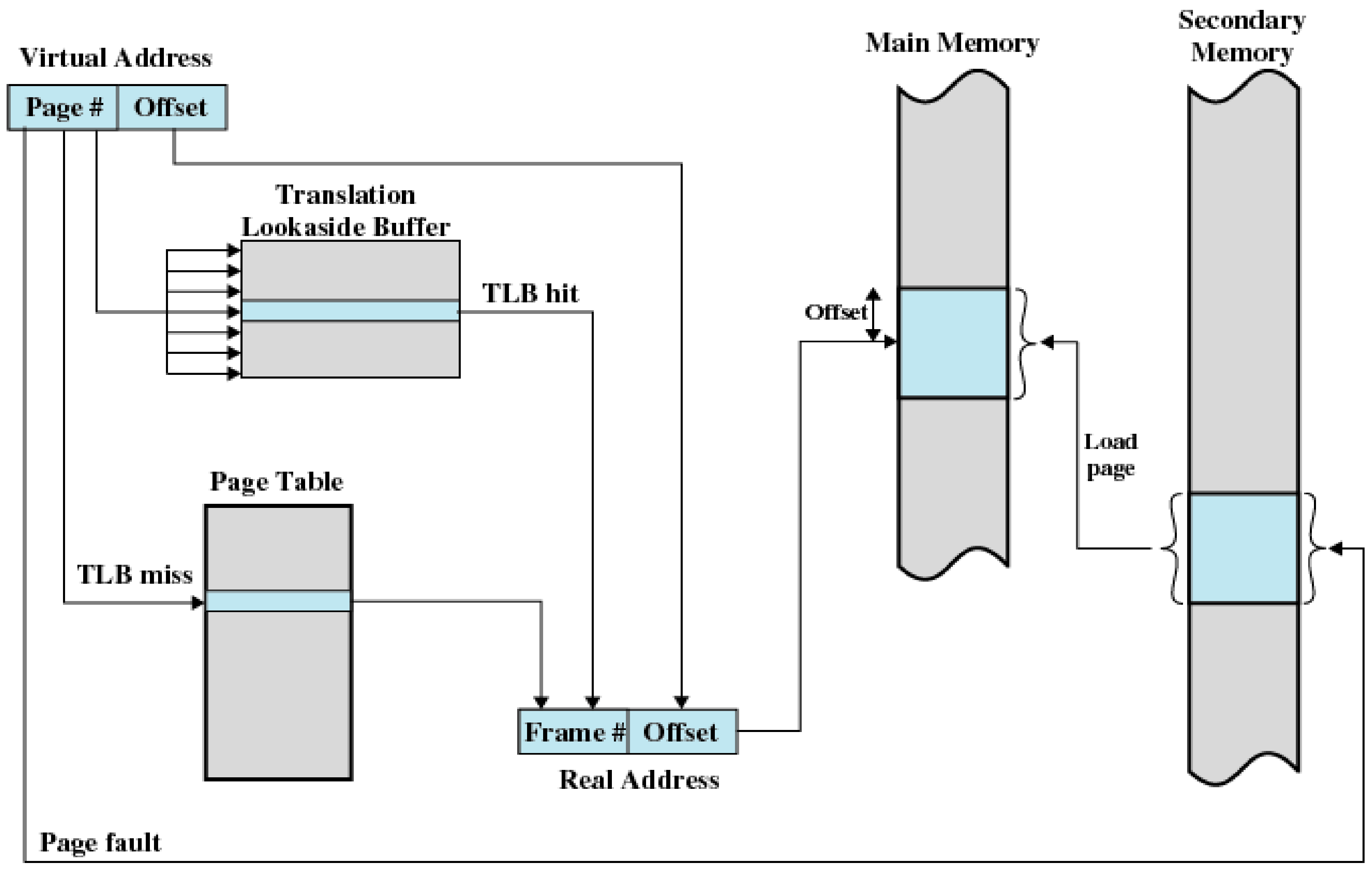


Figure 8.7 Use of a Translation Lookaside Buffer

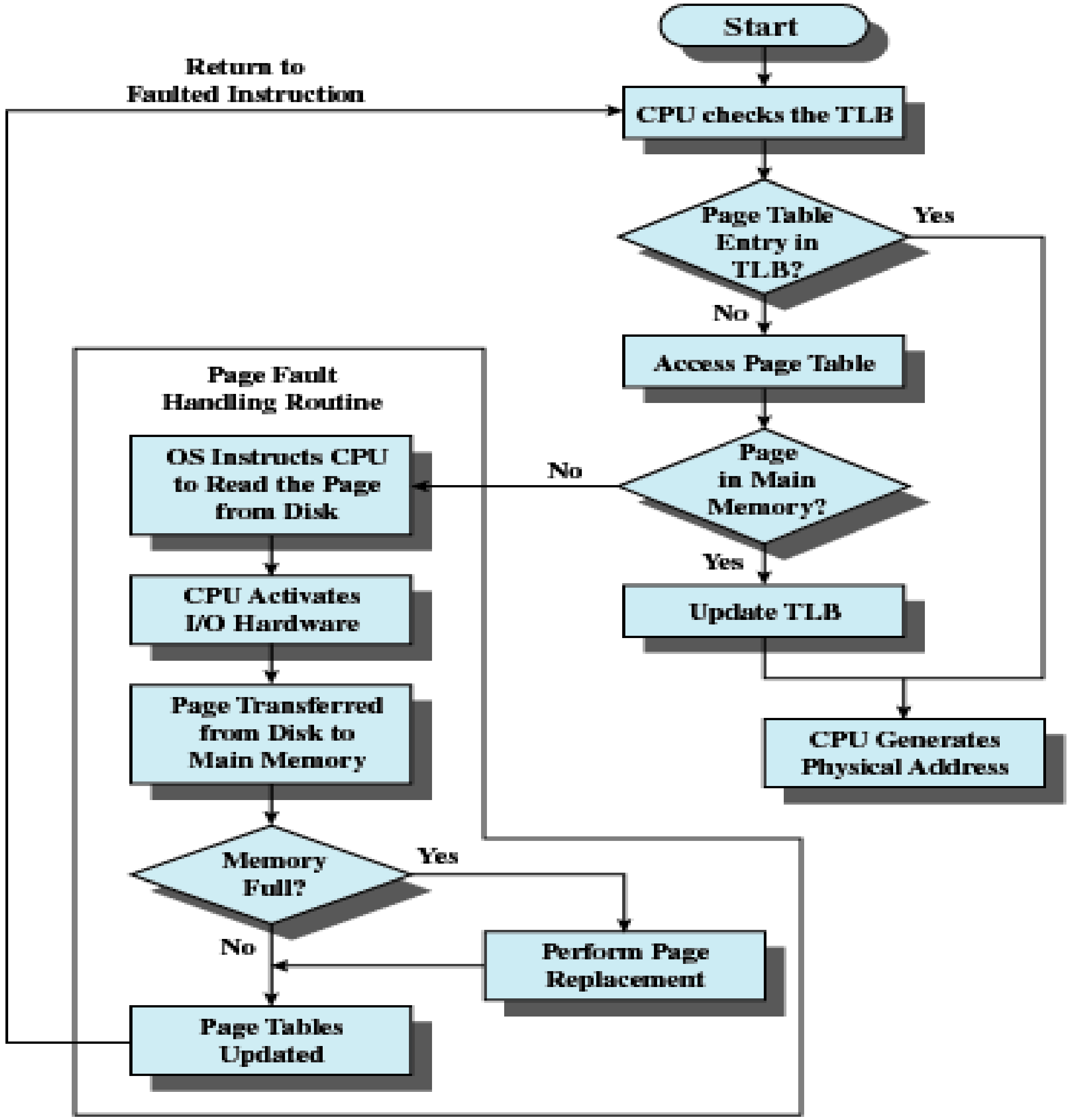

Start
CPU checks the TLB
Return to Faulted Instruction
Page Table Entry in TLB?
Yes
No
Access Page Table
Page in Main Memory?
No
Yes
Update TLB
CPU Generates Physical Address
Page Fault Handling Routine
OS Instructs CPU to Read the Page from Disk
CPU Activates I/O Hardware
Page Transferred from Disk to Main Memory
Memory Full?
Yes
No
Perform Page Replacement
Page Tables Updated

Figure 8.8 Operation of Paging and Translation Lookaside Buffer (TLB) [FURH87]

Need for a fully- associative virtual memory

Fully associative: A virtual memory page can be placed anywhere in the physical memory.

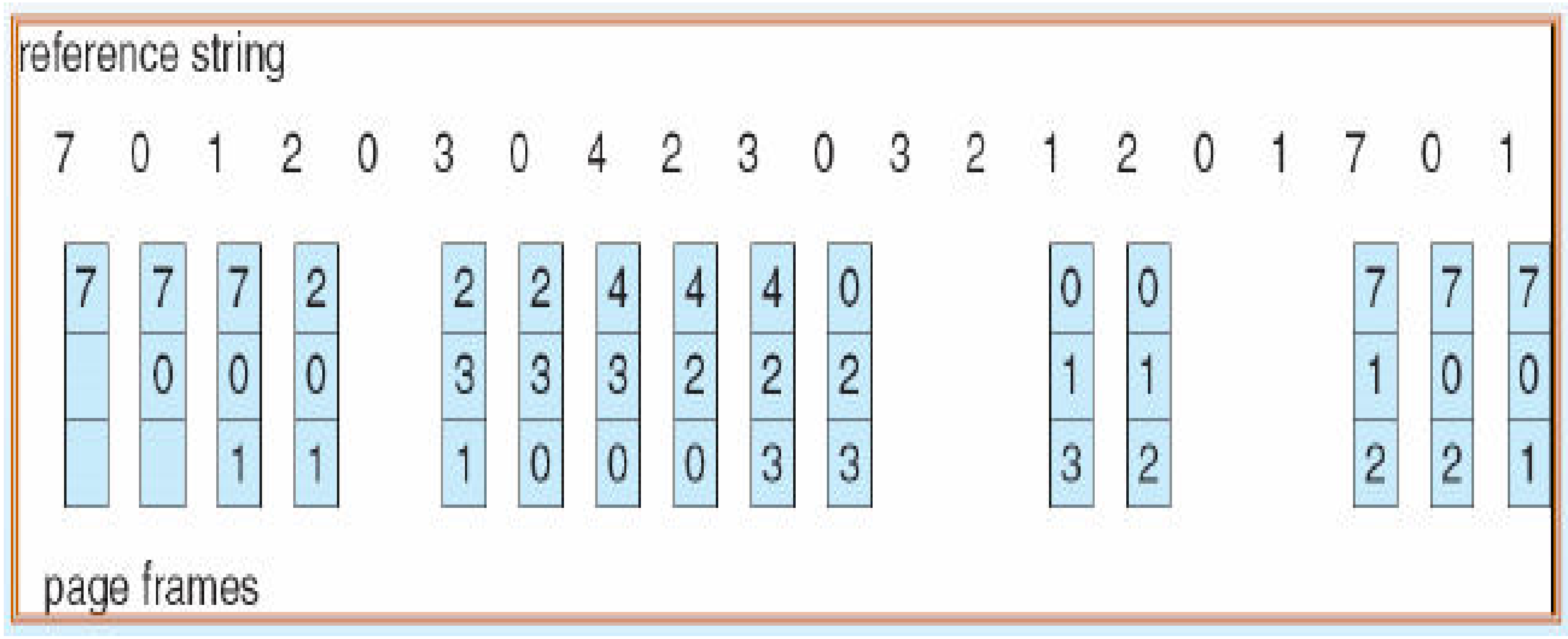
- The design of the virtual memory focuses in reducing the miss rate
- an important source for misses are the conflicts in direct mapped and set associative caches (refer caches).
- a fully associative cache (TLB) eliminates these conflicts but it is very expensive.
- Hence a fully associative mapping is required through which any virtual page can be placed anywhere in the physical memory.

Page replacement Algorithms

- Which page need to be replaced when page fault occurs?
- Page Replacement algorithms
  - FIFO (First-In First-Out)
  - Optimal page replacement
  - LRU (Least Recently Used)
  - LFU(Least Frequently Used)

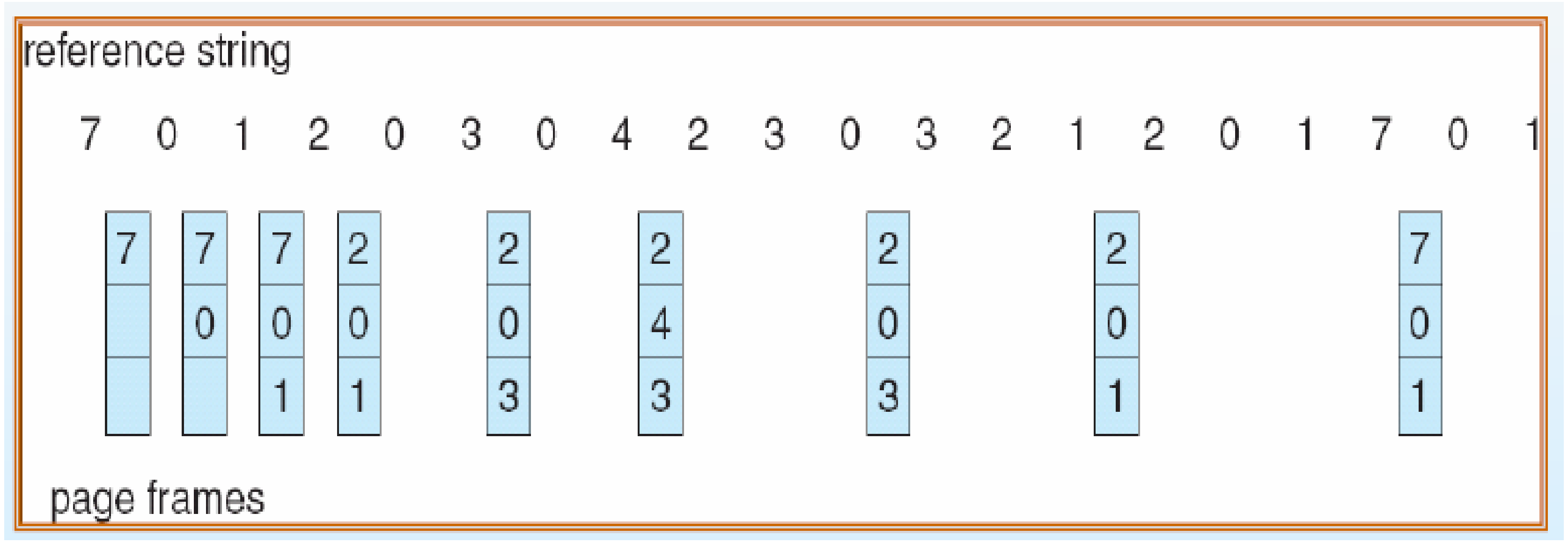
FIFO

- When a page must be replaced, the oldest page is chosen



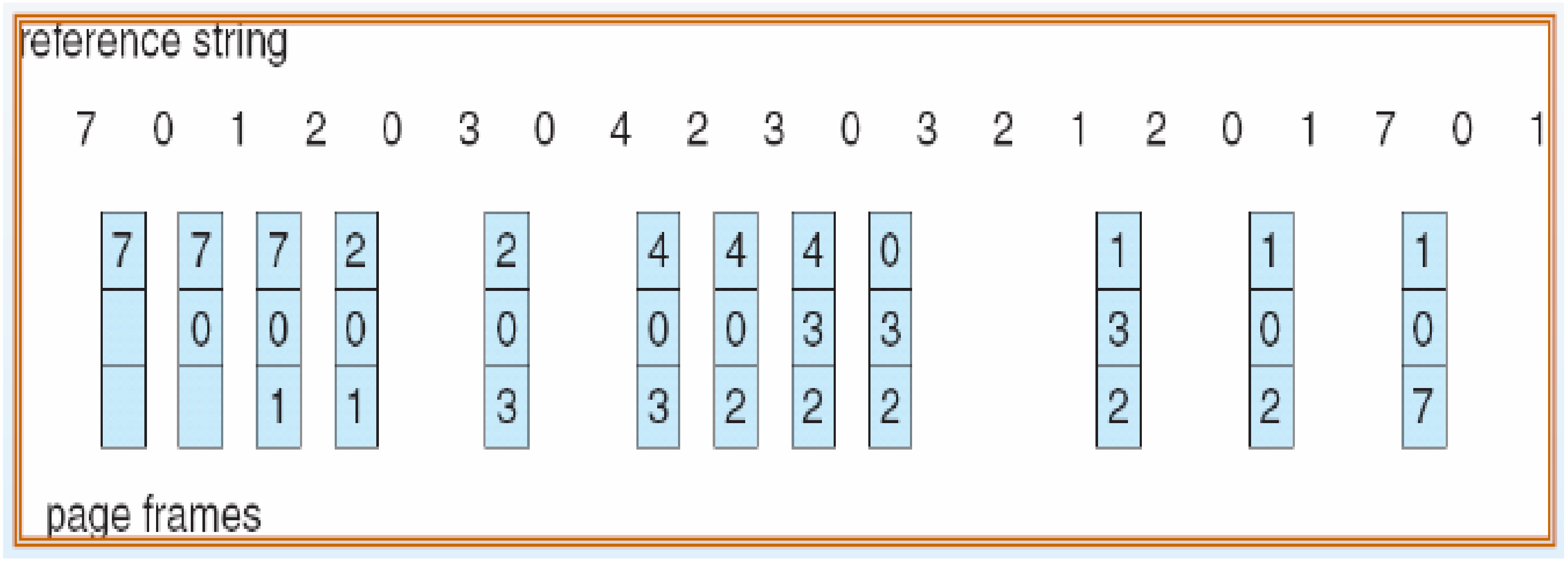
Optimal Page-Replacement Algorithm

Replace the page that won't be needed for the longest time in the future



Least-recently-used (LRU) algorithm

Replace the page that hasn't been referenced for the longest time



Problem 1

A virtual memory system has an address space of 8K words, a memory space of 4K words and page and block sizes of 1K words. The following page reference changes occur during a given time interval.

Determine the four pages that are resident in main memory after each page reference change if the replacement algorithm is (a) FIFO (b) LRU (c) Optimal and (d) LFU page replacement algorithm. Compare the performance of the algorithms using

the page hit ratio.

4 3 2 0 1 2 0 4 3 5 2 0 1 2 3

Problem 2

The following sequence of virtual page numbers is encountered in the course of execution on a computer with virtual memory:

3 4 2 6 4 7 1 3 2 6 3 5 1 2 3

Determine the four pages that are resident in main memory after each page reference. Calculate the hit ratio considering LRU as the replacement policy adopted. Identify the type of miss when occurred and total number of misses in each case.