VIRTUAL MEMORY

Computer System Architecture

By

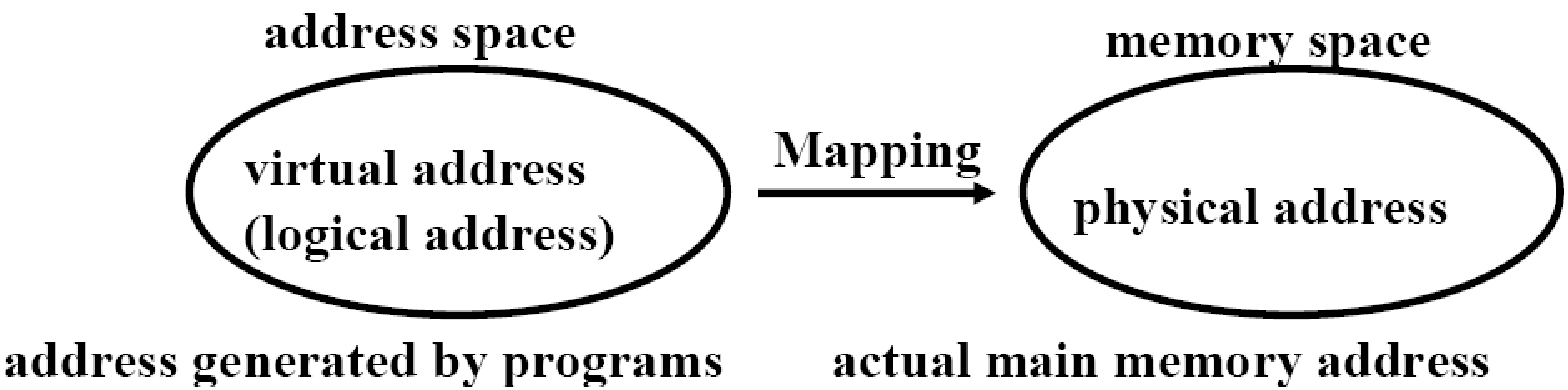
M. Morris Mano

Prof.S.Meenatchi, SITE, VIT

VIRTUAL MEMORY

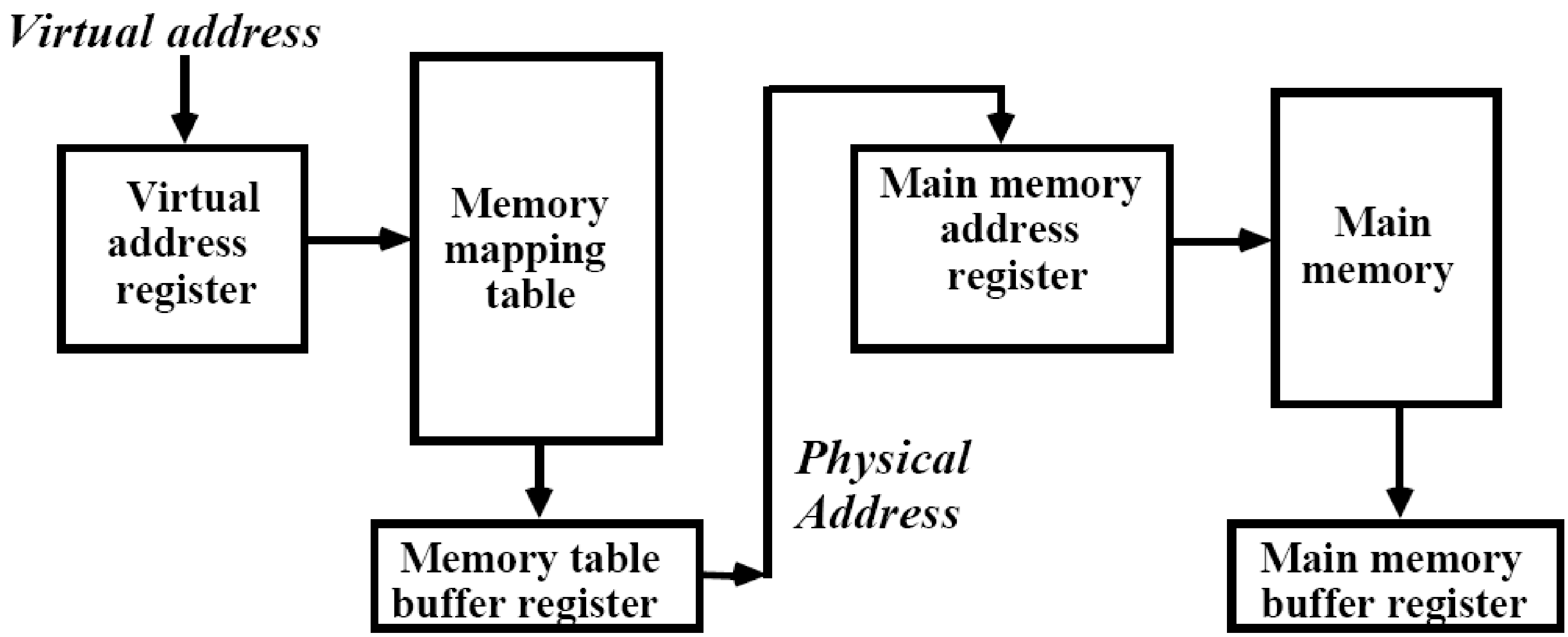
- Give the programmer the illusion that the system has a very large memory, even though the computer actually has a relatively small main memory

- **Address Space (Logical) and Memory Space (Physical)**



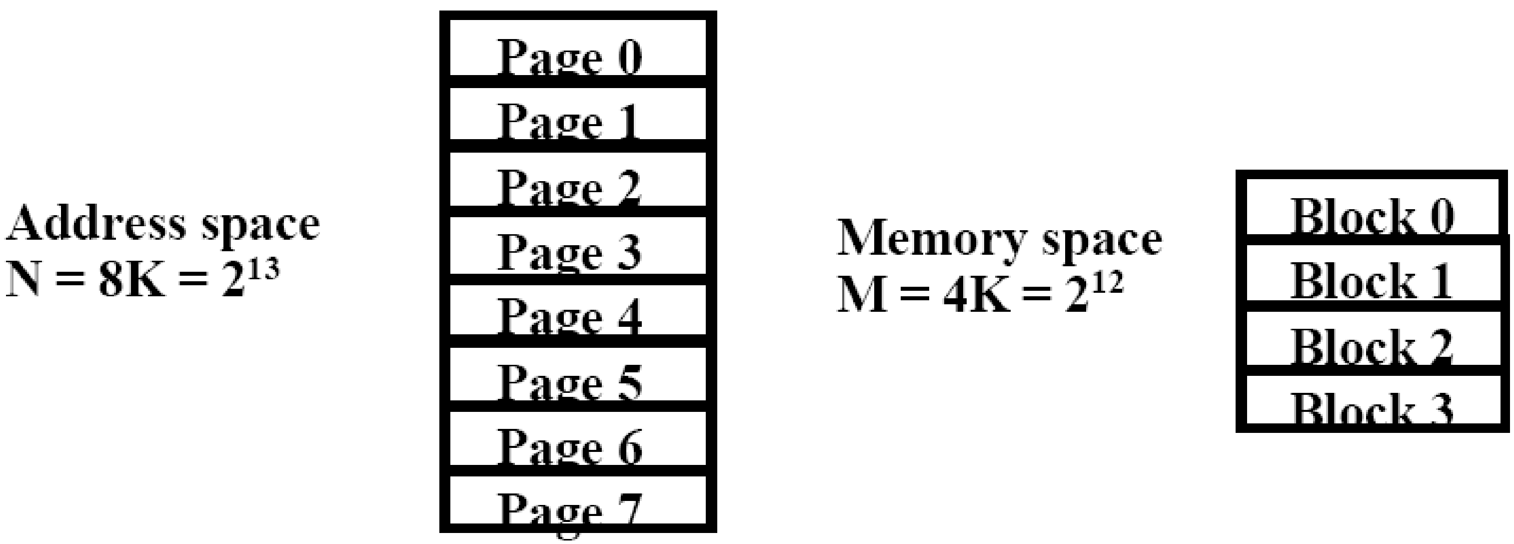
Address Mapping

- Memory *Mapping Table* for *Virtual Address -> Physical Address*

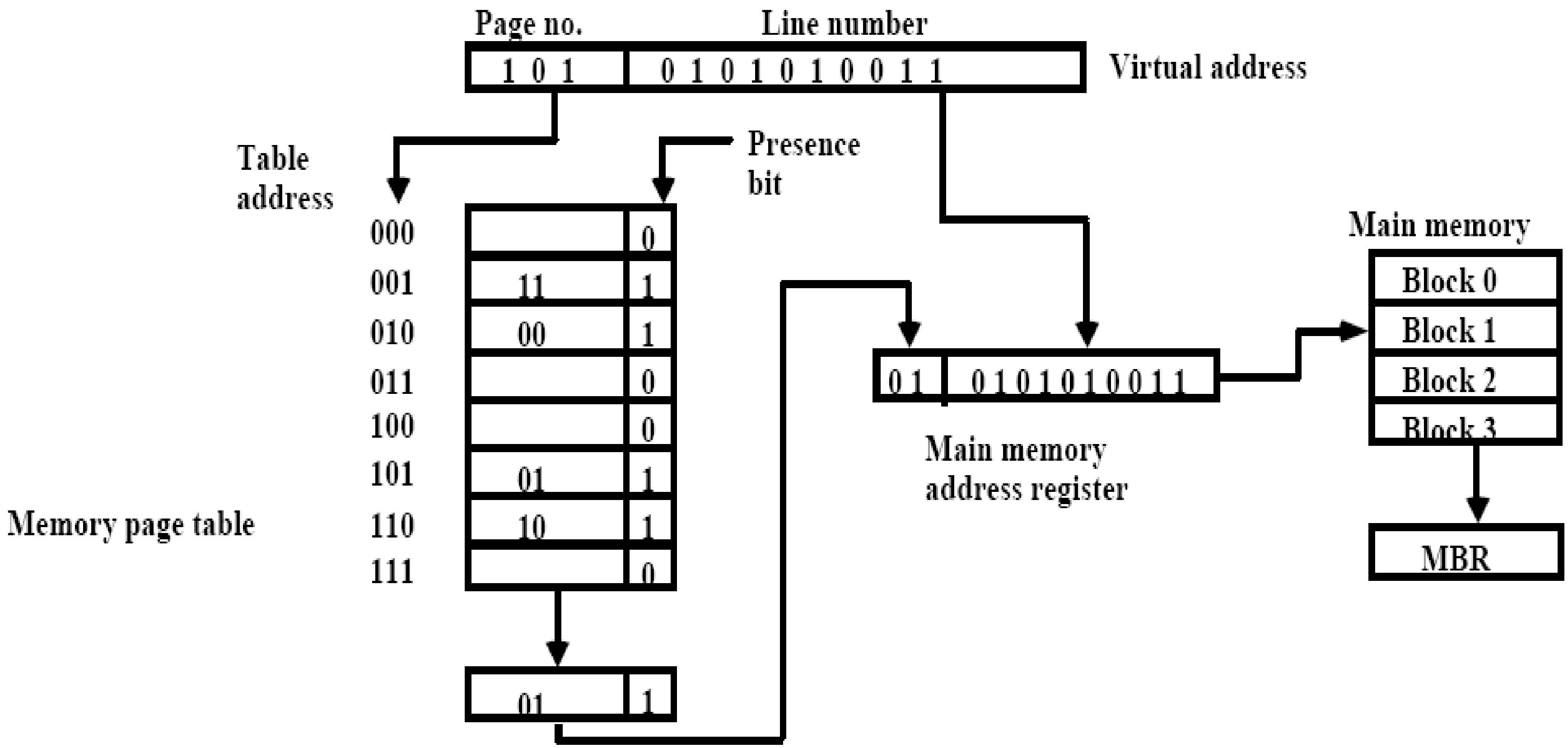


ADDRESS MAPPING

- Address Space and Memory Space are each divided into fixed size group of words called *blocks* or *pages*
- 1K words group



Organization of in a memory Mapping Table paged system



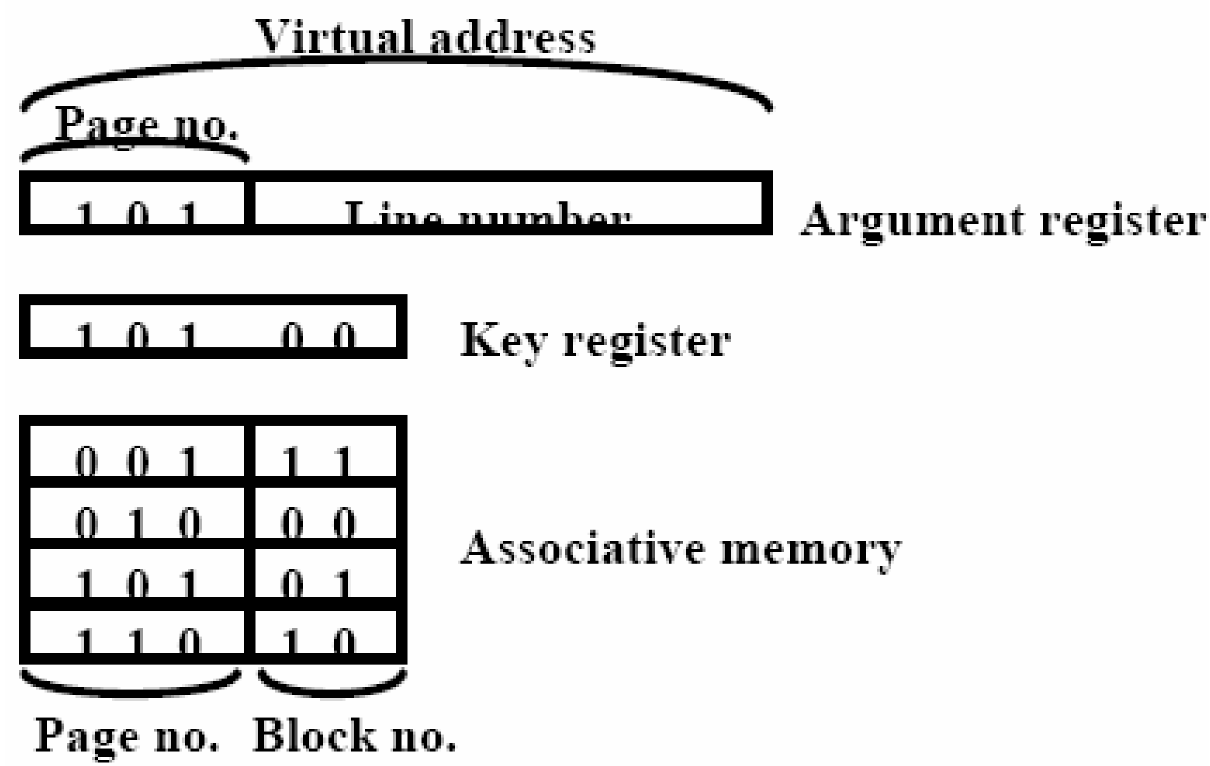
ASSOCIATIVE MEMORY PAGE TABLE

- 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

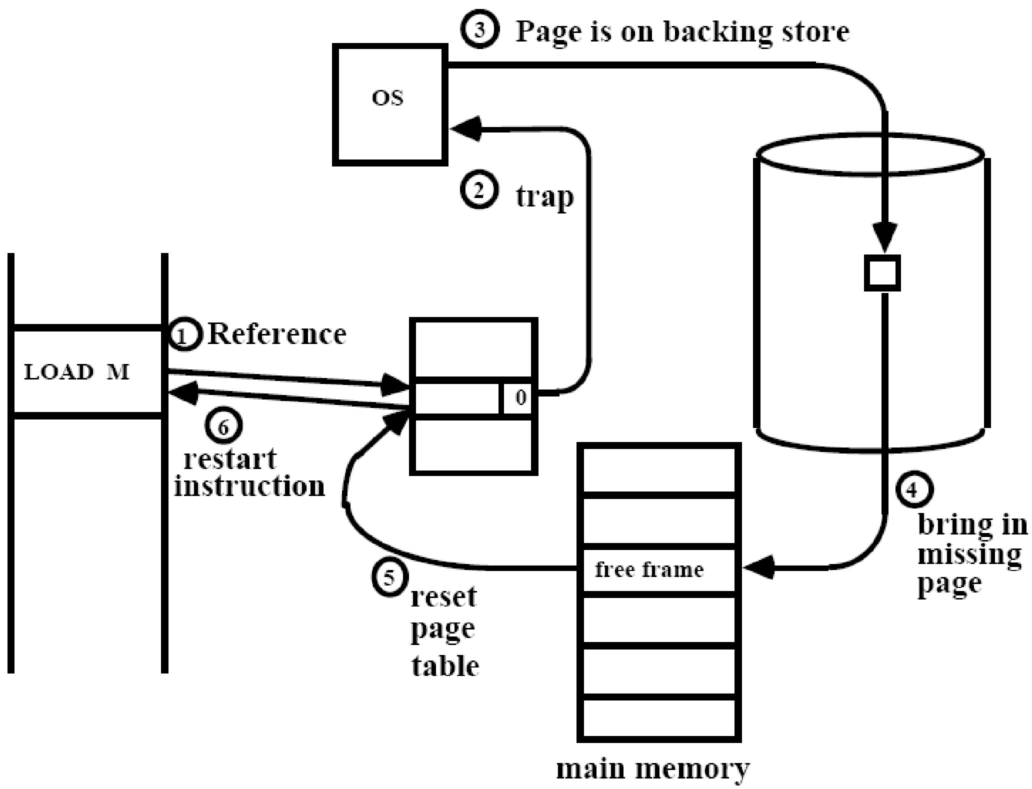
- Page number cannot be found in the Page Table



PAGE FAULT

1. Trap to the OS
2. Save the user registers and program state
3. Determine that the interrupt was a page fault
4. Check that the page reference was legal and determine the location of the page on the backing store(disk)
5. Issue a read from the backing store to a free frame
   a. Wait in a queue for this device until serviced
   b. Wait for the device seek and/or latency time
   c. Begin the transfer of the page to a free frame
6. While waiting, the CPU may be allocated to some other process
7. Interrupt from the backing store (I/O completed)
8. Save the registers and program state for the other user
9. Determine that the interrupt was from the backing store
10. Correct the page tables (the desired page is now in memory)
11. Wait for the CPU to be allocated to this process again
12. Restore the user registers, program state, and new page table, then resume the interrupted instruction.

Processor architecture should provide the ability to restart any instruction after a page fault.



PAGE REPLACEMENT

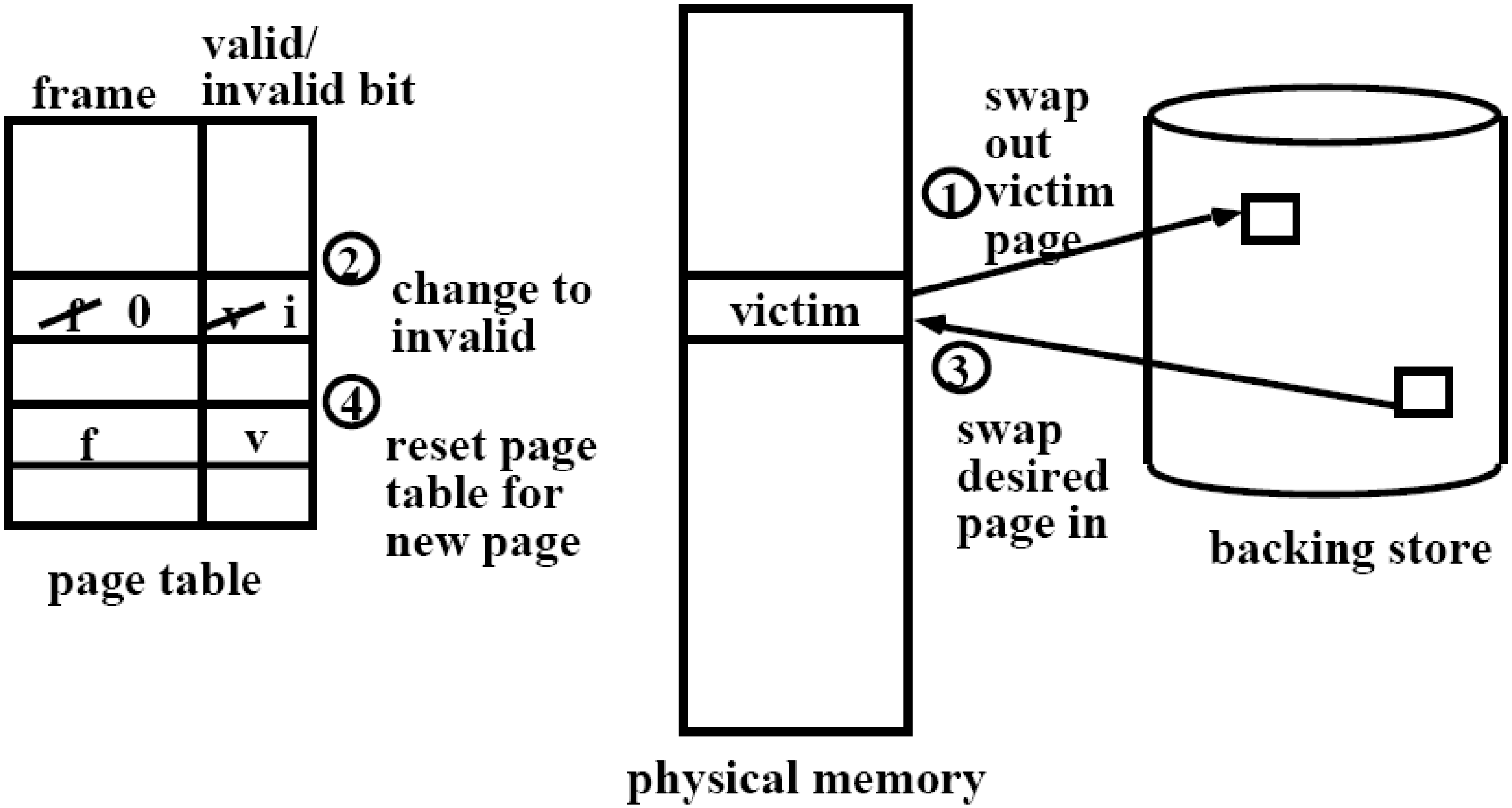
- Decision on which page to displace to make room for an incoming page when no free frame is available
- **Modified page fault service routine**

1. Find the location of the desired page on the backing store
2. Find a free frame

   - If there is a free frame, use it

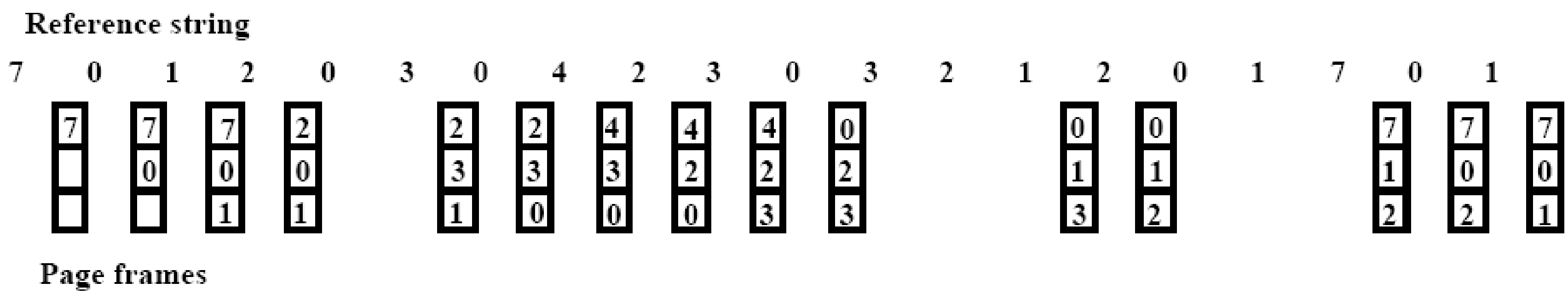
   - Otherwise, use a page-replacement algorithm to select a *victim* frame

   - Write the victim page to the backing store
3. Read the desired page into the (newly) free frame
4. Restart the user process


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

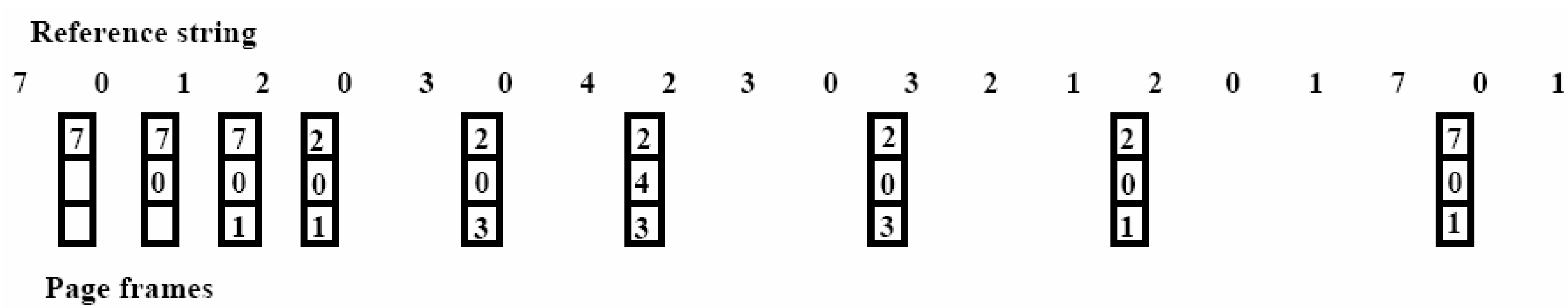
PAGE REPLACEMENT ALGORITHMS

- **<u>FIFO</u>**
  - FIFO algorithm selects the page that has been in memory the longest time
  - Using a queue - every time a page is loaded, its identification is inserted in the queue
  - Easy to implement
  - May result in a frequent page fault



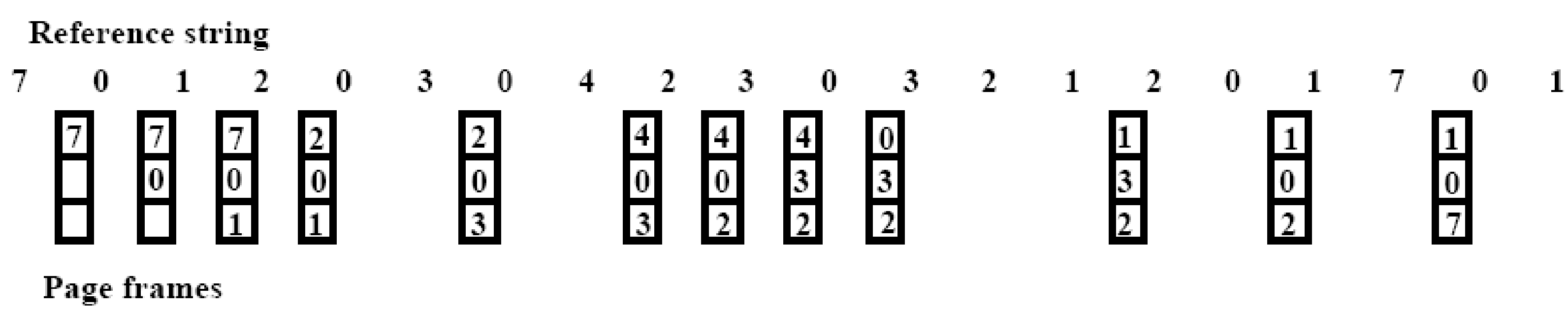
PAGE REPLACEMENT ALGORITHMS

- **<u>Optimal Replacement</u> (OPT)**
  - Lowest page fault rate of all algorithms
  - Replace that page which will not be used for the longest period of time



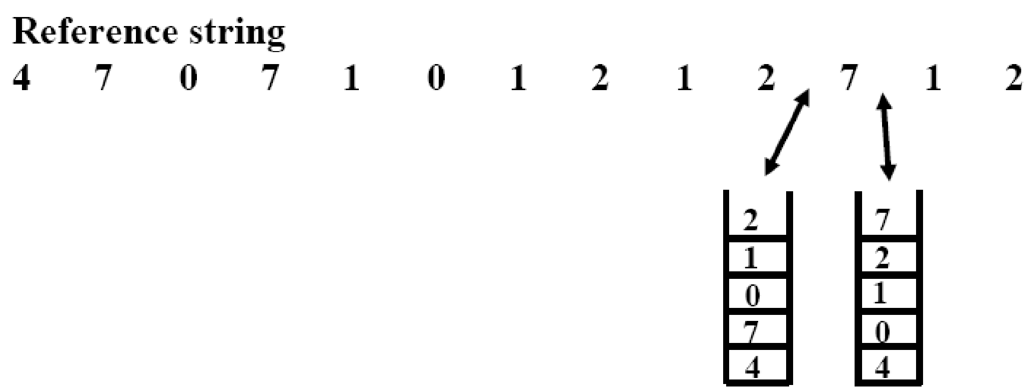
PAGE REPLACEMENT ALGORITHMS

- **<u>LRU</u>**
  - OPT is difficult to implement since it requires future knowledge
  - LRU uses the recent past as an approximation of near future.
  - Replace that page which has not been used for the longest period of time
  - LRU may require substantial hardware assistance
  - The problem is to determine an order for the frames defined by the time of last use



PAGE REPLACEMENT ALGORITHMS

- LRU Implementation Methods
- **Counters**
  - For each page table entry - time-of-use register
  - Incremented for every memory reference
  - Page with the smallest value in time-of-use register is replaced
- **Stack**
  - Stack of page numbers
  - Whenever a page is referenced its page number is removed from the stack and pushed on top
  - Least recently used page number is at the bottom



- **LRU Approximation**
  - Reference (or use) bit is used to approximate the LRU
  - Turned on when the corresponding page is
  - referenced after its initial loading
  - Additional reference bits may be used