Memory Hierarchy

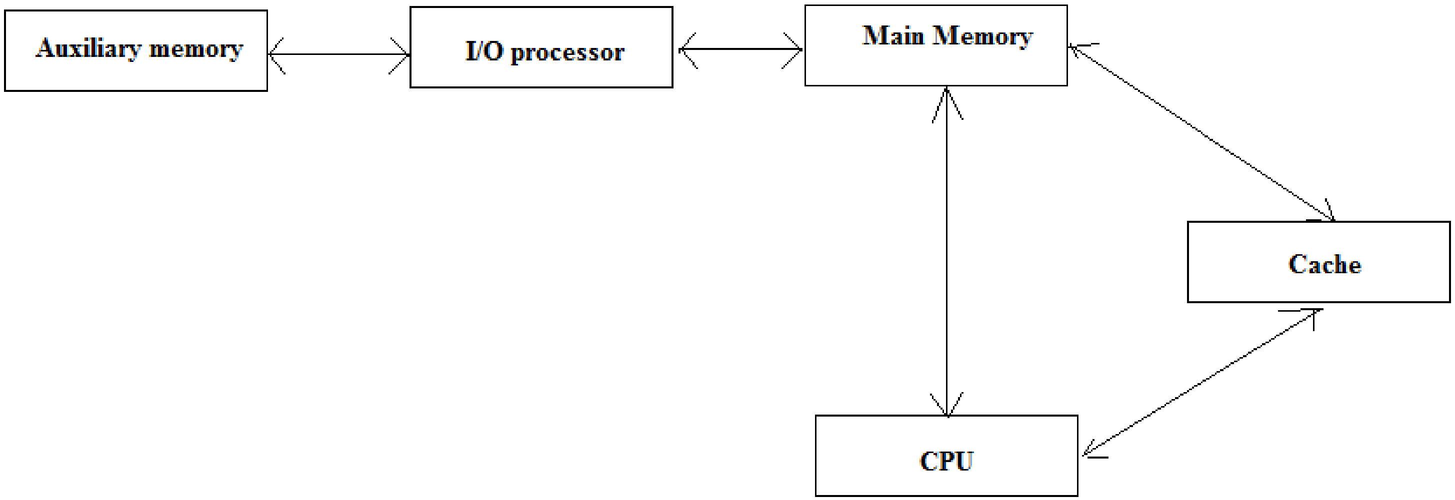
- The memory unit is an essential component in any digital computer since it is needed for storing programs and data
- Not all accumulated information is needed by the CPU at the same time
- Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by CPU

Memory Hierarchy-II

- The memory unit that directly communicate with CPU is called the main memory .
- Devices that provide backup storage are called auxiliary memory .
- The memory hierarchy system consists of all storage devices employed in a computer system from the slow by high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory.

Memory Hierarchy-III

- The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor
- A special very-high-speed memory called **cache** is used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate



Memory Hierarchy-IV

- CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory
- The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations
- The typical access time ratio between cache and main memory is about 1to7
- Auxiliary memory access time is usually 1000 times that of main memory

Main Memory

- Most of the main memory in a general purpose computer is made up of RAM integrated circuits chips, but a portion of the memory may be constructed with ROM chips

- RAM– Random Access memory
    - In tegated RAM are available in two possible operating modes, S*tatic and Dynamic*
- ROM– Read Only memory

Random-Access Memory (RAM)

- Static RAM (SRAM)
  - Each cell stores bit with a six-transistor circuit.
  - Retains value indefinitely, as long as it is kept powered.
  - Relatively insensitive to disturbances such as electrical noise.
  - Faster and more expensive than DRAM.

- Dynamic RAM (DRAM)
  - Each cell stores bit with a capacitor and transistor.
  - Value must be refreshed every 10-100 ms.
  - Sensitive to disturbances.
  - Slower and cheaper than SRAM.

SRAM vs DRAM Summary

	Tran. per bit	Access time	Persist?	Sensitive?	Cost	Applications
SRAM	6	1X	Yes	No	100x	cache memories
DRAM	1	10X	No	Yes	1X	Main memories, frame buffers

ROM

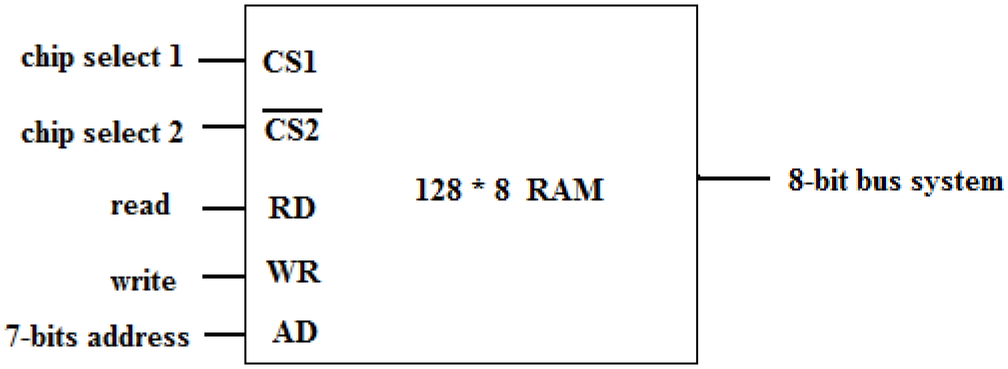
- ROM is used for storing programs that are **PERMENTLY** resident in the computer and for tables of constants that do not change in value once the production of the computer is completed
- The ROM portion of main memory is needed for storing an initial program called *bootstrap loader,* witch is to start the computer software operating when power is turned off

Main Memory

- A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip when needed

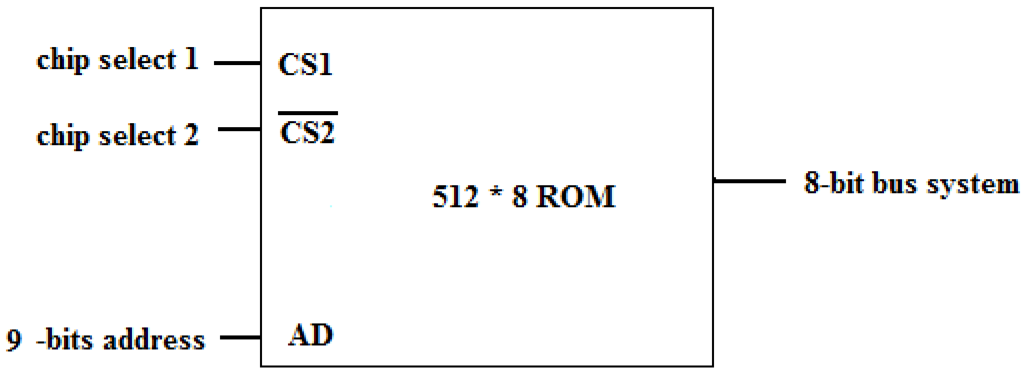
- The Block diagram of a RAM chip is shown next slide, the capacity of the memory is 128 words of 8 bits (one byte) per word

RAM



CS1	$\overline{CS2}$	RD	WD	Memory Function	State of data bus
0	0	*	*	Inhibit	High-impedance
0	1	*	*	Inhibit	High-impedance
1	0	0	0	Inhibit	High-impedance
1	0	0	1	Write	Input data to RAM
1	0	1	*	Read	Output data from RAM
1	1	*	*	Inhibit	High-impedance

ROM


chip select 1
CS1
chip select 2
$\overline{CS2}$
512 * 8 ROM
8-bit bus system
9 -bits address
AD

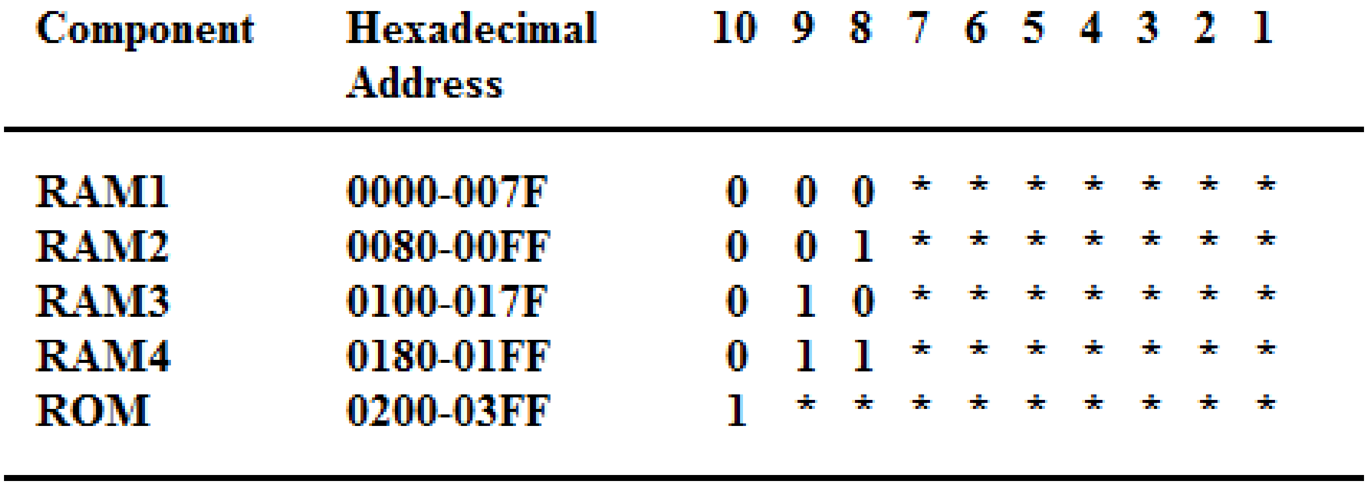
Memory Address Map

- Memory Address Map is a pictorial representation of assigned address space for each chip in the system

- To demonstrate an example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM

- The RAM have 128 byte and need seven address lines, where the ROM have 512 bytes and need 9 address lines

Memory Address Map



Component	Hexadecimal Address	10	9	8	7	6	5	4	3	2	1
RAM1	0000-007F	0	0	0	*	*	*	*	*	*	*
RAM2	0080-00FF	0	0	1	*	*	*	*	*	*	*
RAM3	0100-017F	0	1	0	*	*	*	*	*	*	*
RAM4	0180-01FF	0	1	1	*	*	*	*	*	*	*
ROM	0200-03FF	1	*	*	*	*	*	*	*	*	*

Memory Address Map

- The hexadecimal address assigns a range of hexadecimal equivalent address for each chip
- Line 8 and 9 represent four distinct binary combination to specify which RAM we chose
- When line 10 is 0, CPU selects a RAM. And when it’s 1, it selects the ROM

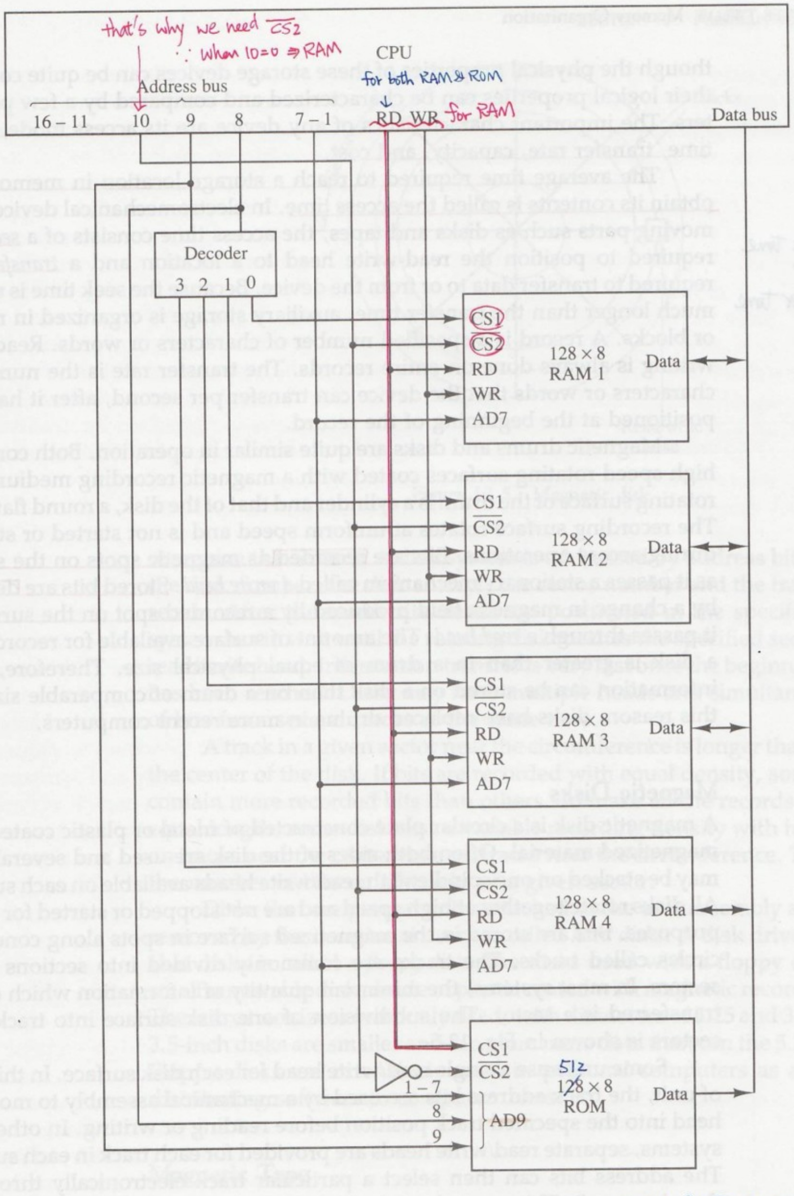


Figure 12-4 Memory connection to the CPU. RAM & ROM connection

Memory Interleaving concept (p324)

Auxiliary Memory

- The average time required to reach a storage location in memory and obtain its contents is called the **access** time

- The access time = seek time + transfer time
    - Seek time: required to position the read-write head to a location
    - Transfer time: required to transfer data to or from the device

Cache memory

- If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced,
- Thus reducing the total execution time of the program
- Such a fast small memory is referred to as cache memory
- The cache is the fastest component in the memory hierarchy and approaches the speed of CPU component

Cache memory

- When CPU needs to access memory, the cache is examined
- If the word is found in the cache, it is read from the fast memory
- If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word

Cache memory

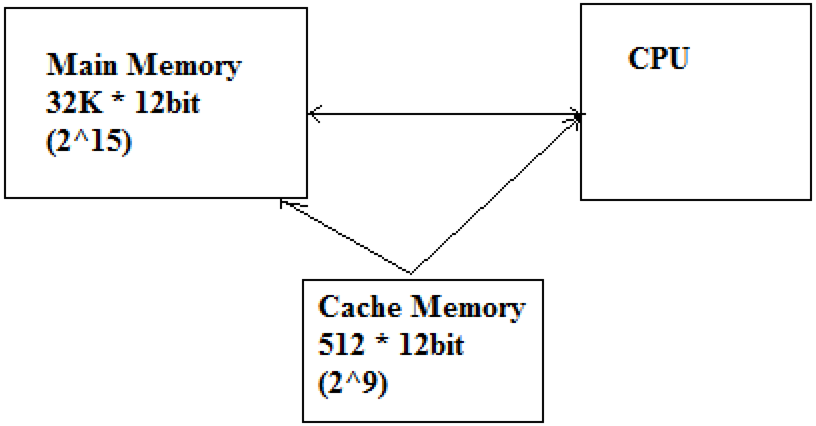
- The performance of cache memory is frequently measured in terms of a quantity called **hit ratio**
- When the CPU refers to memory and finds the word in cache, it is said to produce a **hit**
- Otherwise, it is a **miss**
- **Hit ratio = hit / (hit+miss)**

Cache memory

- The basic characteristic of cache memory is its fast access time,
- Therefore, very little or no time must be wasted when searching the words in the cache
- The transformation of data from main memory to cache memory is referred to as a **mapping** process, there are three types of mapping:
  - Associative mapping
  - Direct mapping
  - Set-associative mapping

Cache memory

- To help understand the mapping procedure, we have the following example:

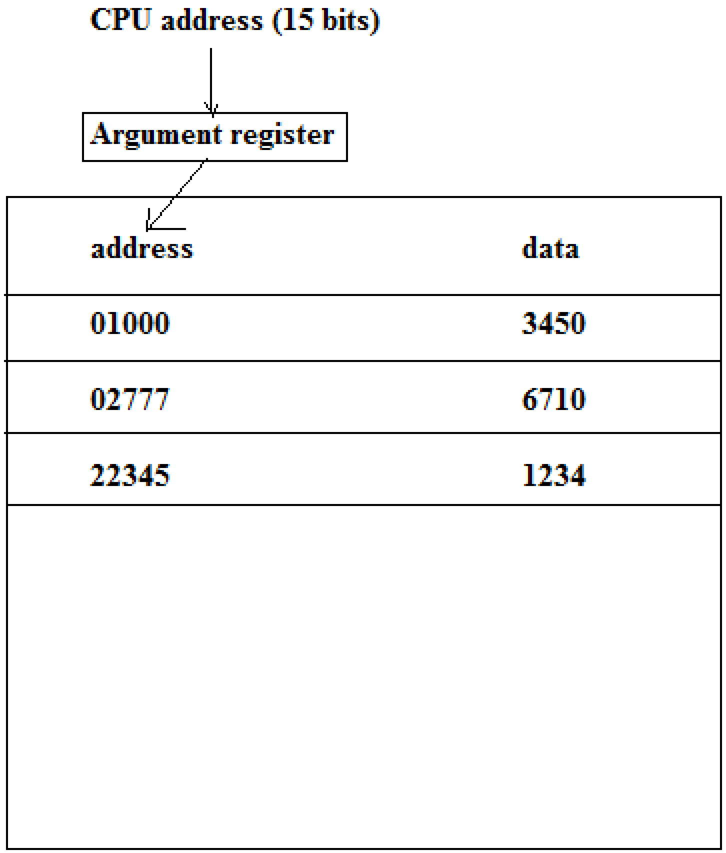


Associative mapping

- The fastest and most flexible cache organization uses an associative memory
- The associative memory stores both the address and data of the memory word
- This permits any location in cache to store ant word from main memory

- The address value of 15 bits is shown as a five-digit **octal** number and its corresponding 12-bit word is shown as a four-digit octal number

Associative mapping



CPU address (15 bits)

Argument register

address	data
01000	3450
02777	6710
22345	1234

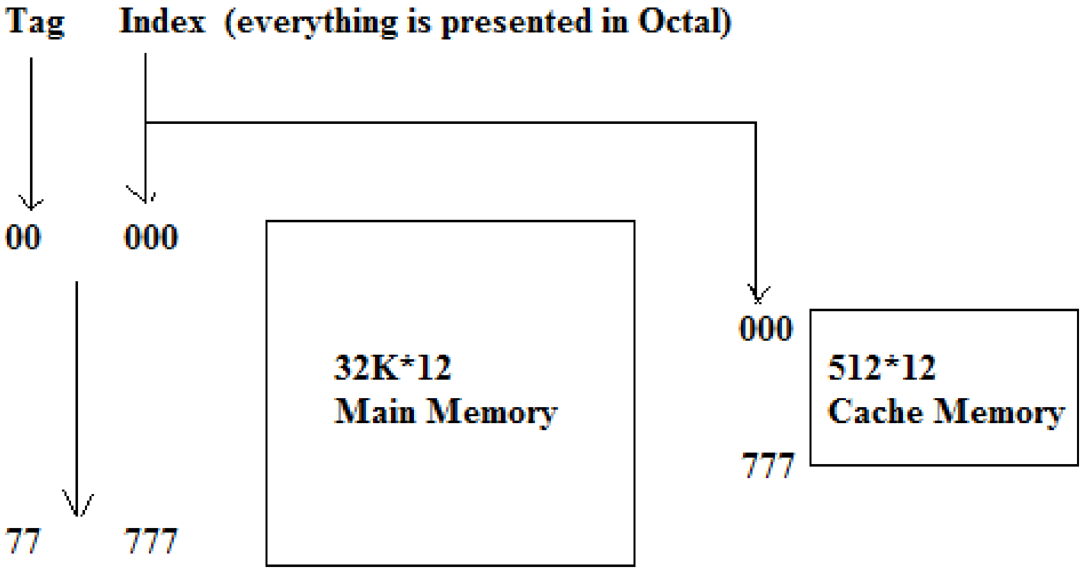
Associative mapping

- A CPU address of 15 bits is places in the argument register and the associative memory us searched for a matching address
- If the address is found, the corresponding 12-bits data is read and sent to the CPU
- If not, the main memory is accessed for the word
- If the cache is full, an address-data pair must be displaced to make room for a pair that is needed and not presently in the cache

Direct Mapping

- Associative memory is expensive compared to RAM
- In general case, there are 2^k words in cache memory and 2^n words in main memory (in our case, k=9, n=15)
- The n bit memory address is divided into two fields: k-bits for the index and n-k bits for the tag field

Direct Mapping



Tag Index (everything is presented in Octal)

Direct Mapping

Memory Address	Memory Data
00000	1220
00777	2340
01000	3450
01111	2222
01777	4560
02000	5670
02777	6710

Index Address	Tag	Data
000	00	1220
111	01	2222
777	02	6710

Set-Associative Mapping

- The disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time
- Set-Associative Mapping is an improvement over the direct-mapping in that each word of cache can store two or more word of memory under the same index address

Set-Associative Mapping

Memory Address	Memory Data
00000	1220
00777	2340
01000	3450
01111	2222
01777	4560
02000	5670
02777	6710

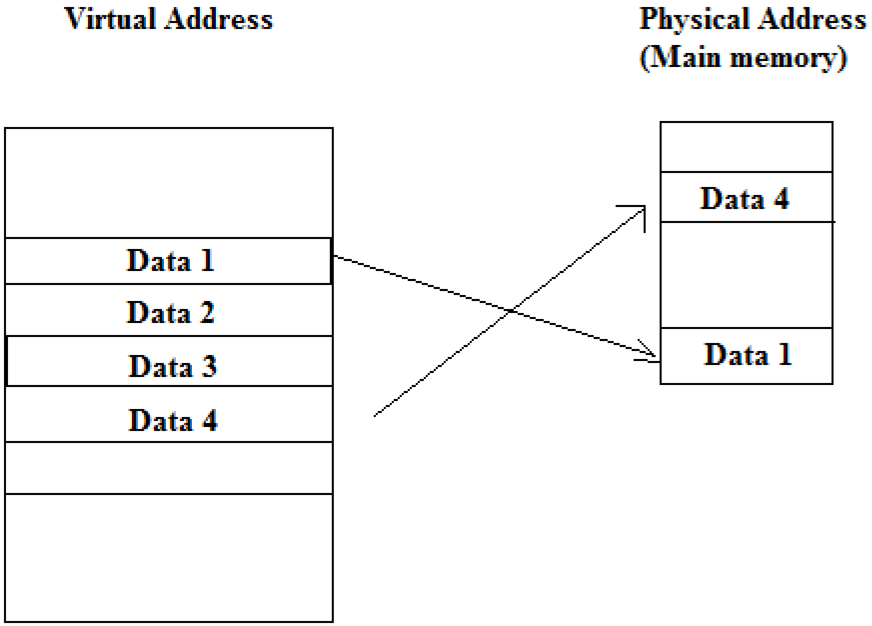
Index Address	Tag	Data	Tag	Data
000	01	3450	02	5670
111	01	2222		
777	02	6710	00	2340

Set-Associative Mapping

- In the slide, each index address refers to two data words and their associated tags
- Each tag requires six bits and each data word has 12 bits, so the word length is 2*(6+12) = 36 bits

Virtual Memory

- The address used by a programmer will be called a virtual address or logical address.
- An address in main memory is called a physical address



Virtual Memory

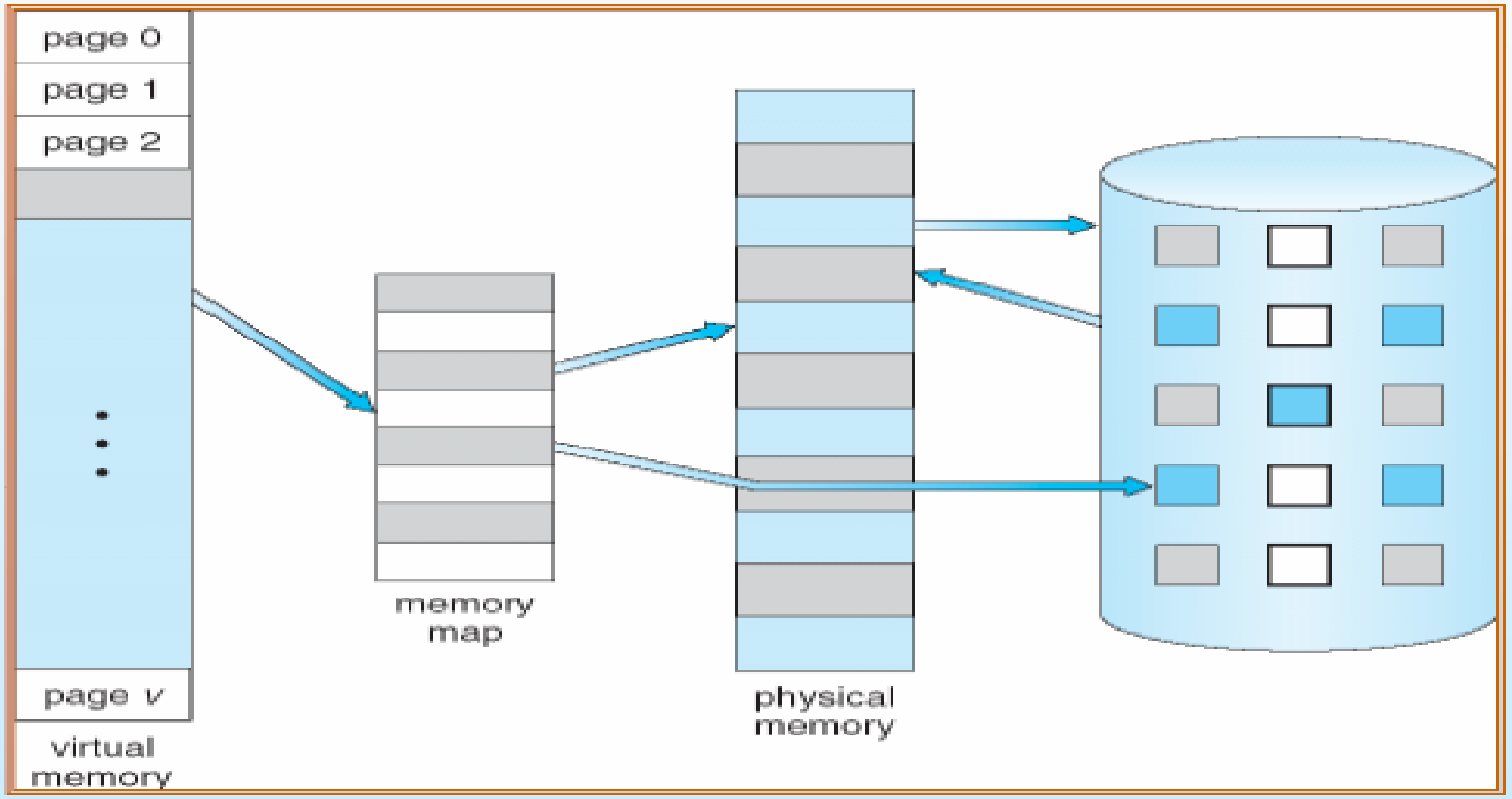
- The term **page** refers to groups of address space of the same size

- For example: if auxiliary memory contains 1024K and main memory contains 32K and page size equals to 1K, then auxiliary memory has 1024 pages and main memory has 32 pages

Virtual Memory

- Only part of the program needs to be in memory for execution
- Logical address space can therefore be much larger than physical address space
- Allows for more efficient process creation

Virtual Memory

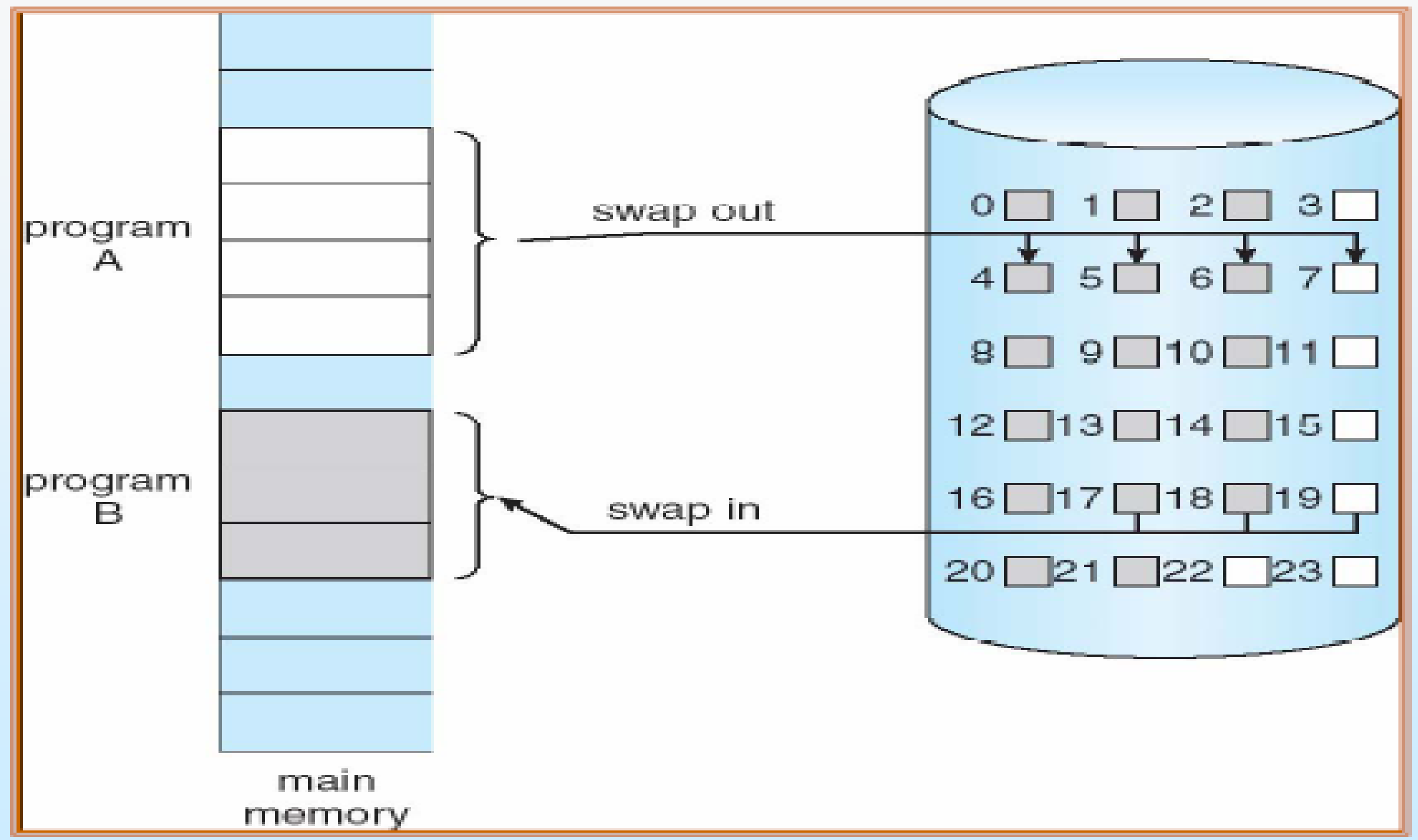


Demand Paging

- In stead of loading whole program into memory, **demand paging** is an alternative strategy to initially load pages only as they are needed

- **Lazy Swapper:** Pages are only loaded when they are demanded during program execution

Transfer of a page memory to continuous disk space



Demand paging basic concepts

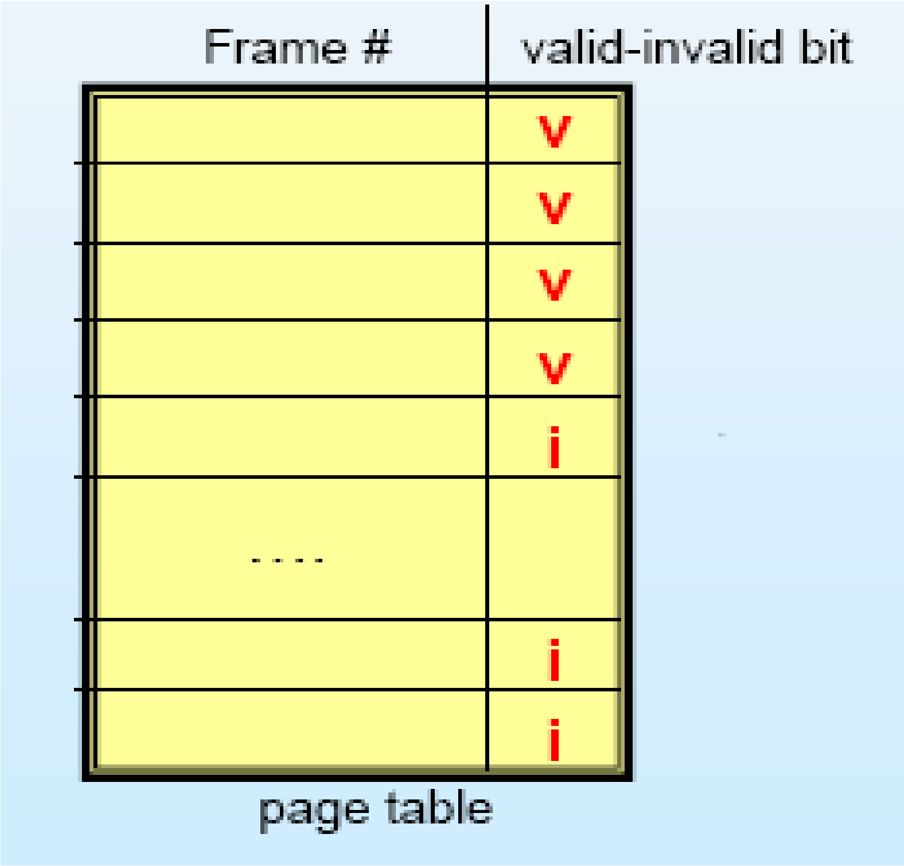
- When a process is to be swapped in, the pager guesses which pages will be used before the process is swapped out again.
- Instead of swapping in a whole process, the pager brings only those necessary pages into memory

Valid-Invalid Bit

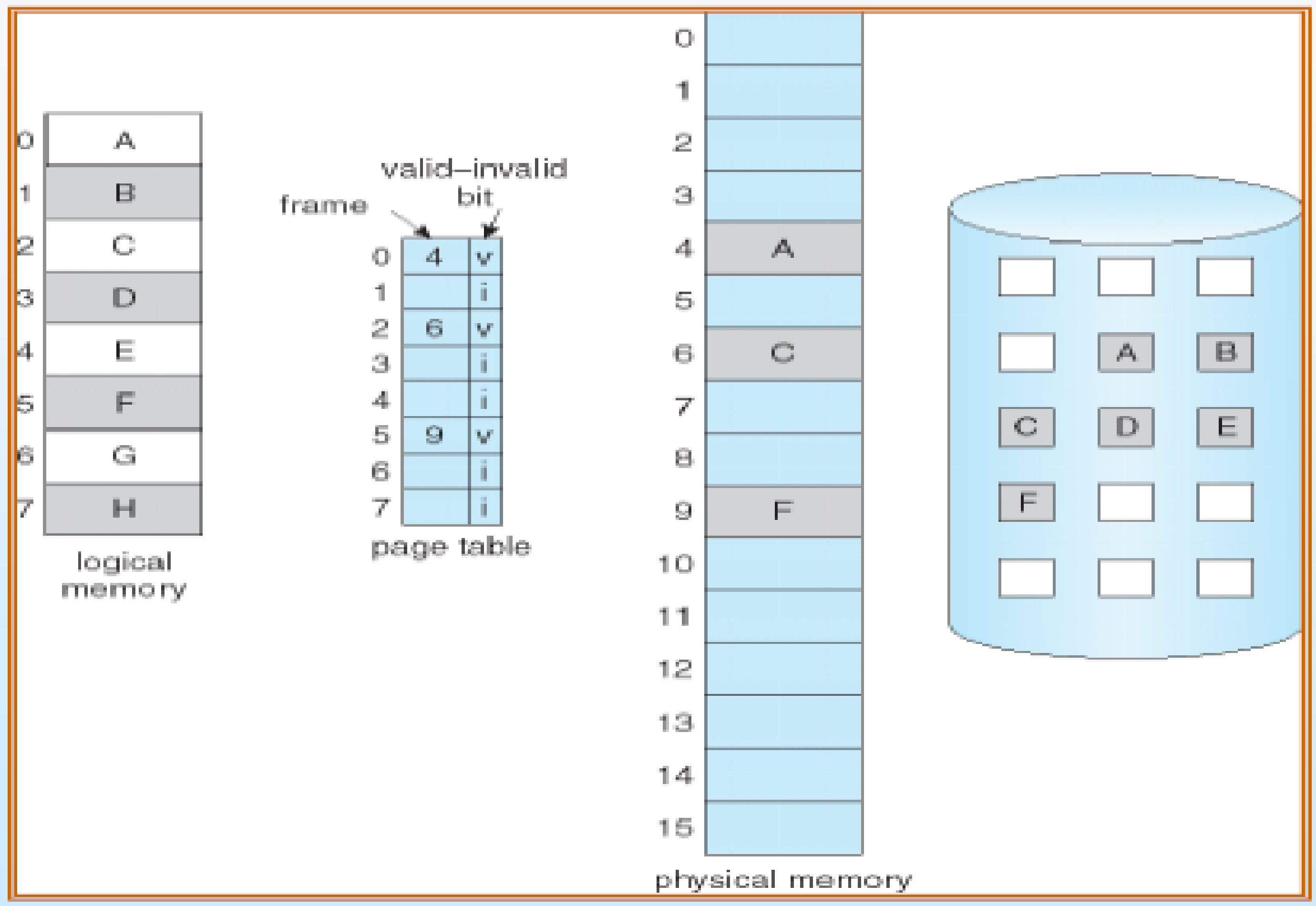
- With each page table entry a valid–invalid bit is associated (**v**=> in-memory , **i** =>not-in-memory)
- Initially valid–invalid bit is set to **i** on all entries

- During address translation, if valid–invalid bit in page table entry is **i** => page fault

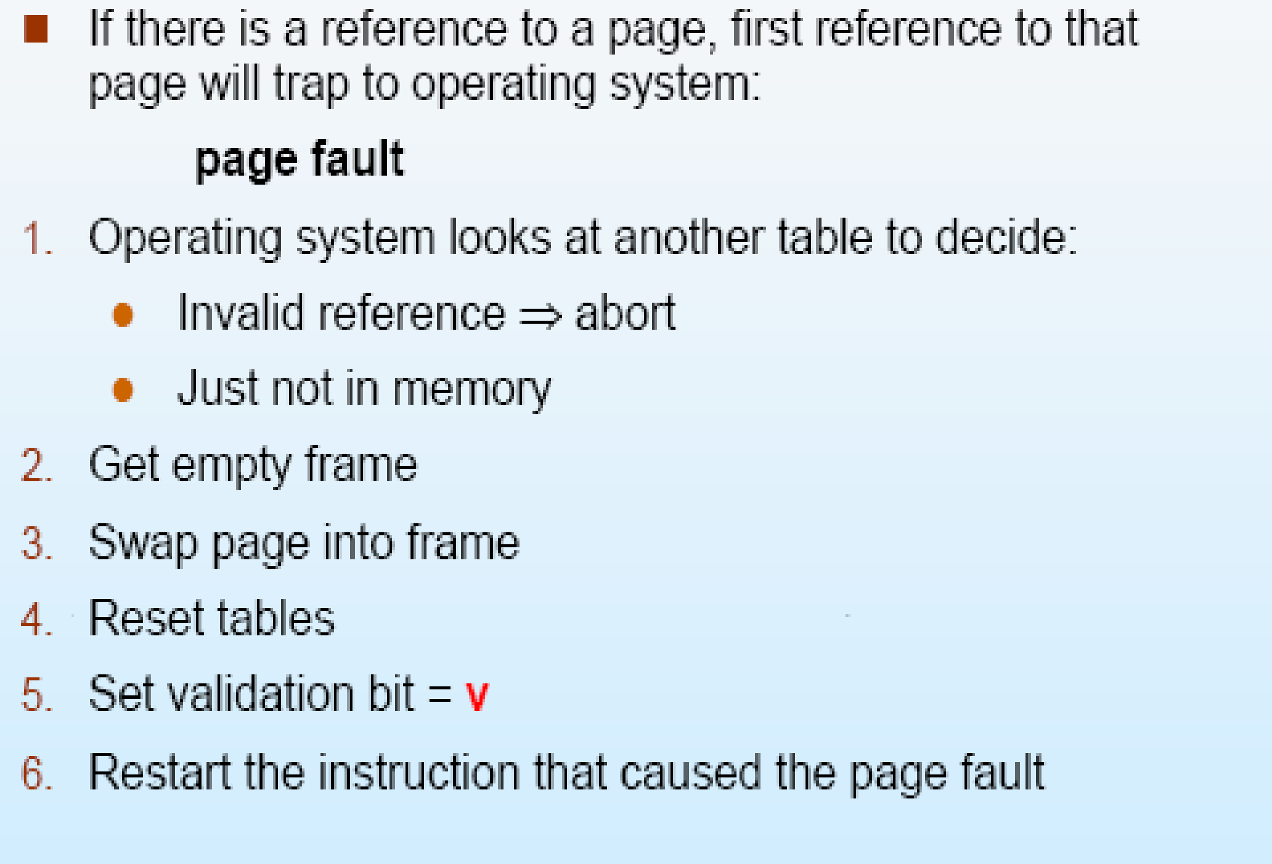
Valid-Invalid Bit Example



Valid-Invalid Bit Example



Page Fault

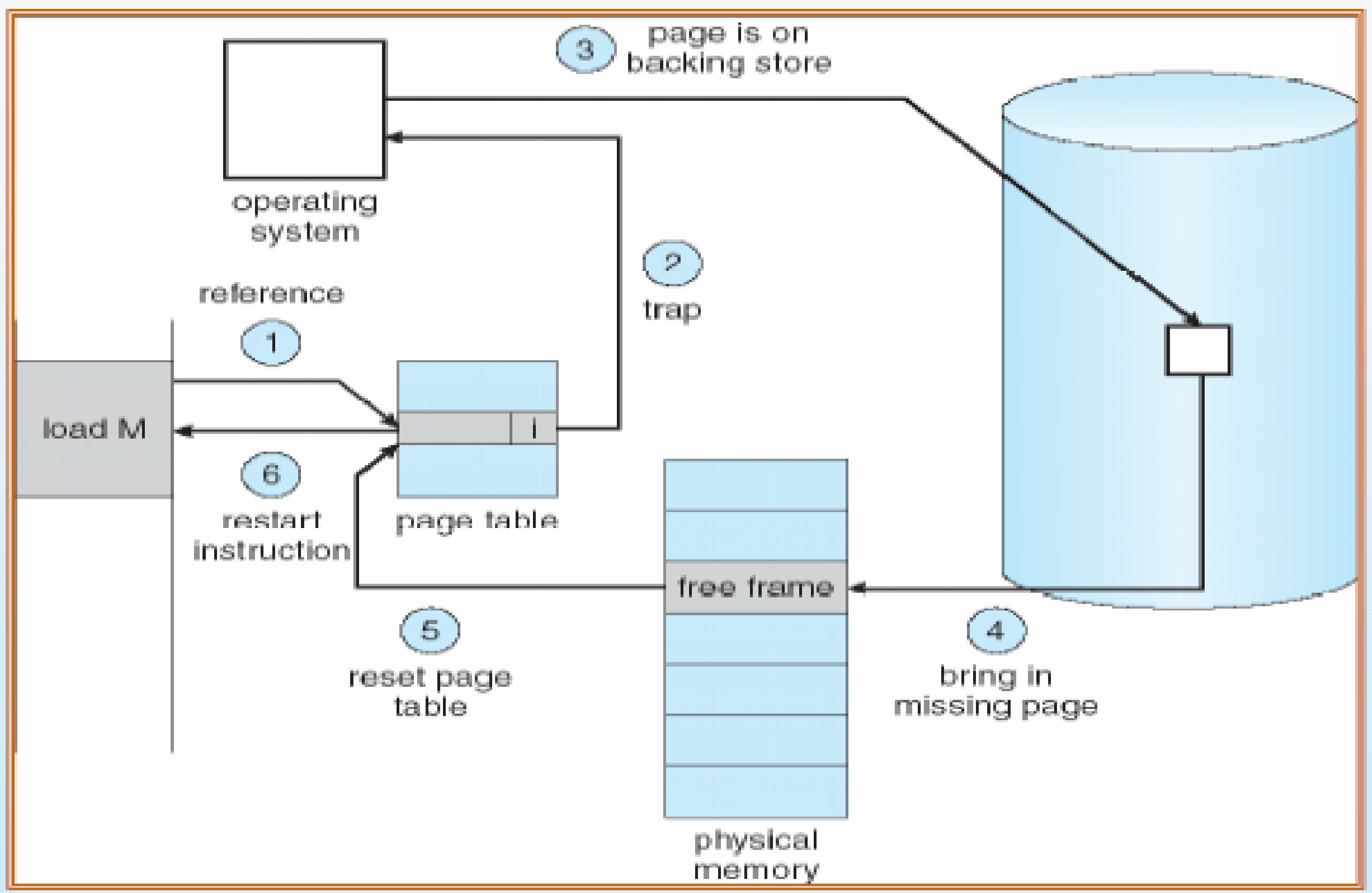


- If there is a reference to a page, first reference to that page will trap to operating system:

  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

Page Fault


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

Performance of Demand Paging

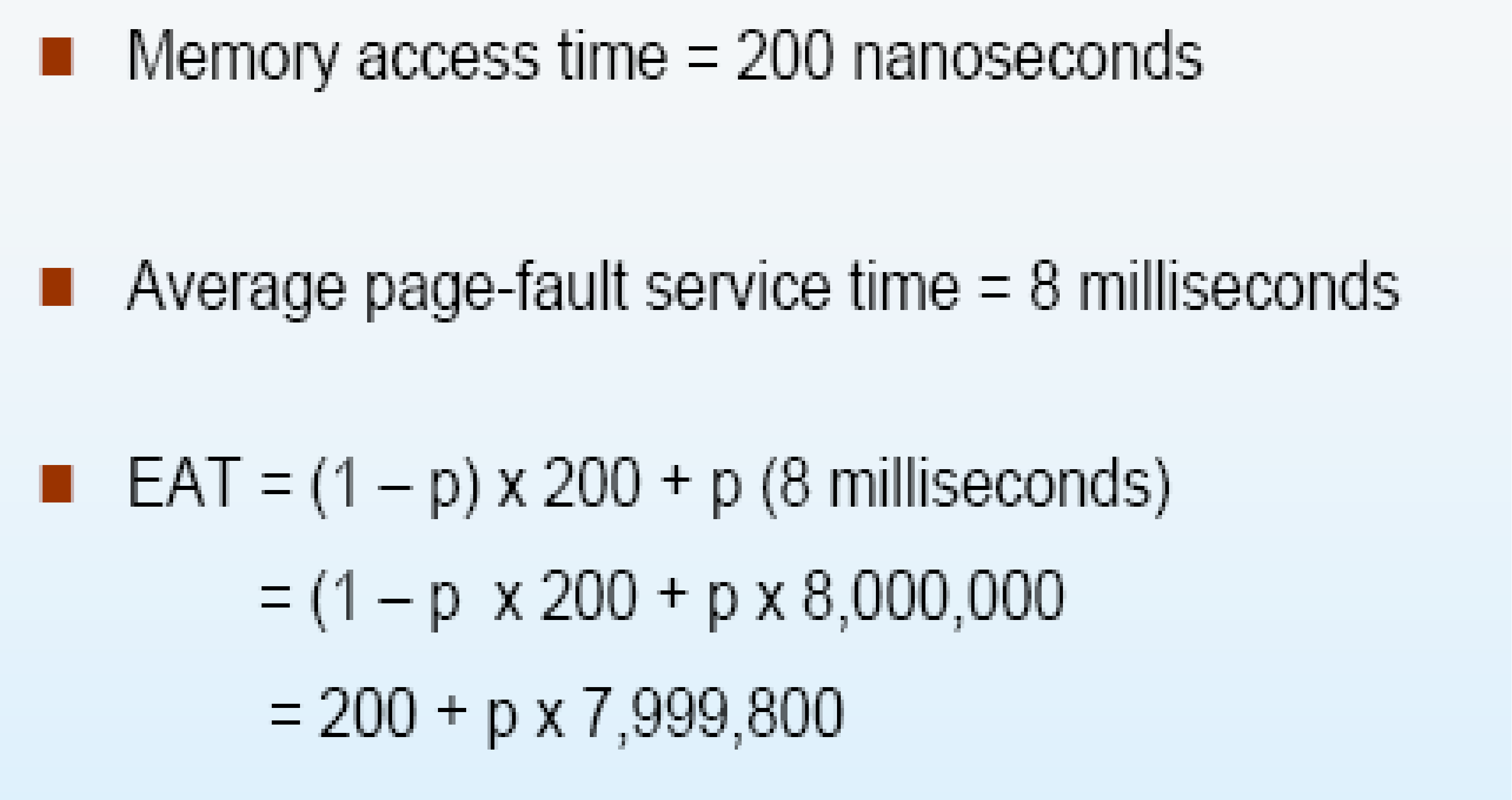
Page Fault Rate $0 \le p \le 1.0$

- if $p = 0$ no page faults
- if $p = 1$, every reference is a fault

- Effective Access Time (EAT)=

(1-p)*ma + p*page fault time

Performance of Demand Paging



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

  $= (1 – p \times 200 + p \times 8{,}000{,}000$

  $= 200 + p \times 7{,}999{,}800$

Performance of Demand Paging

- If we want performance degradation to be less than 10%, we need

$220 > 200+7{,}999{,}800*p$

$20>7{,}999{,}800*p$

$P<0.0000025$

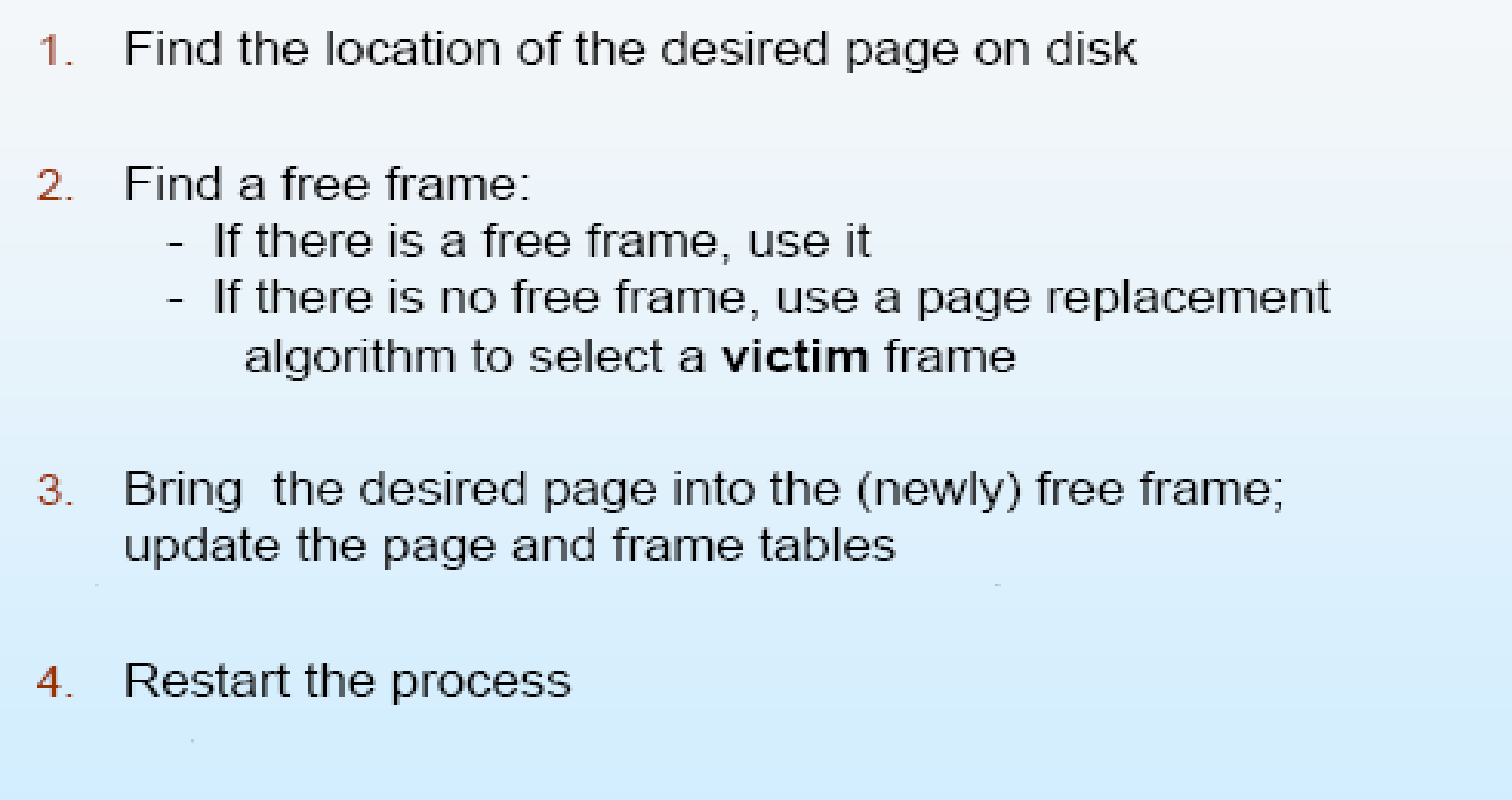
It is important to keep the page-fault rate low in a demand-paging system

Page Replacement

- What if there is no free frame?

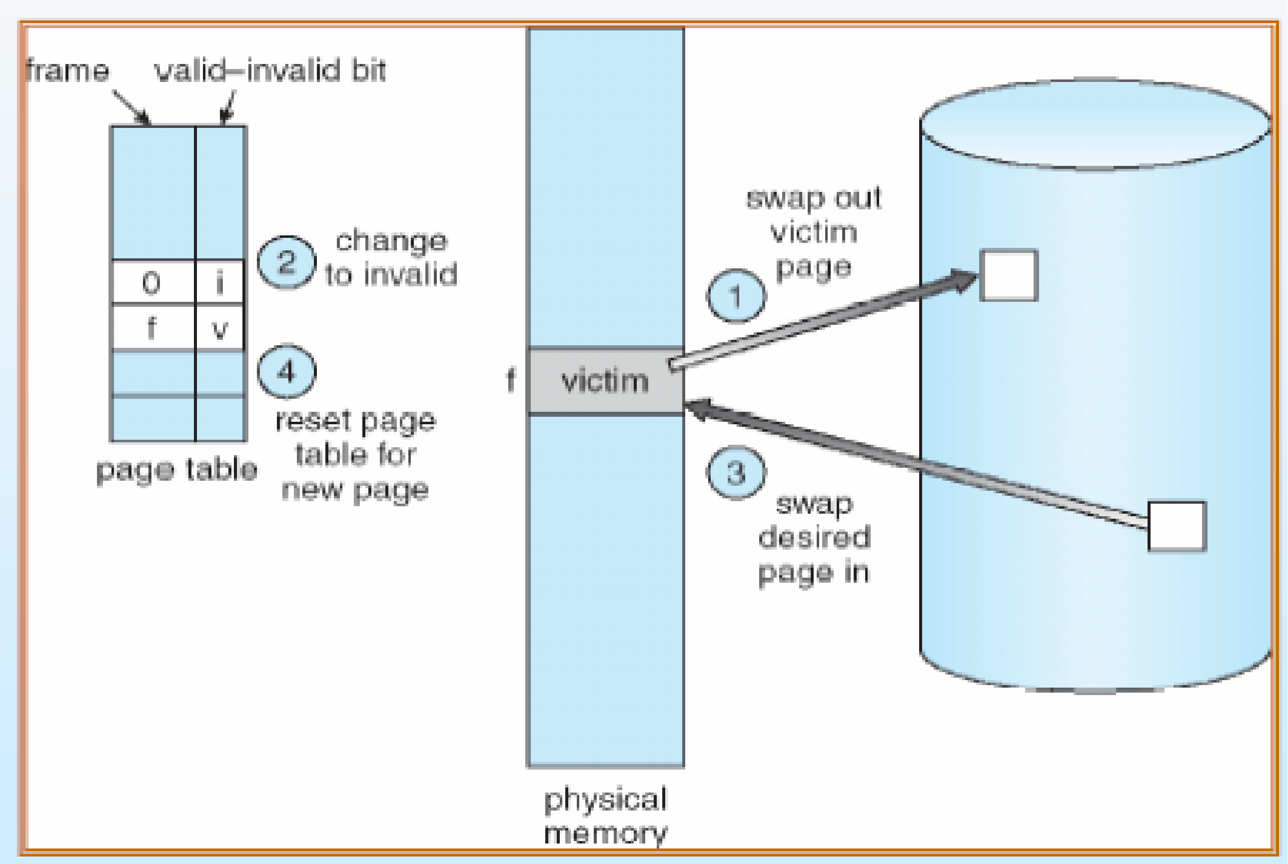
- Page replacement –find some page in memory, but not really in use, swap it out
- In this case, same page may be brought into memory several times

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

Page Replacement



Page Replacement Algorithms

- Goal:

  Want lowest page-fault rate

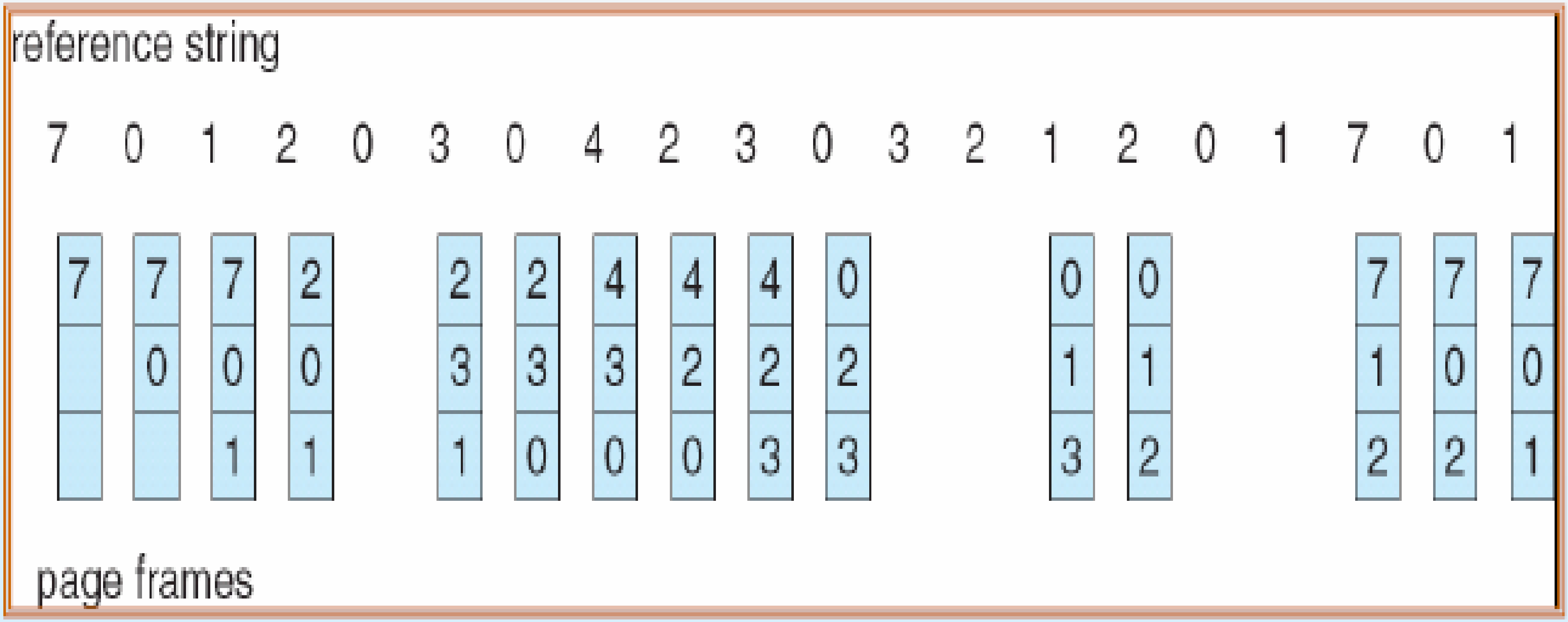
  Evaluate algorithm by running it on a particular string of memory references (reference string) and computing the number of page faults on that string

- In all our examples, the reference string is

  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

FIFO

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



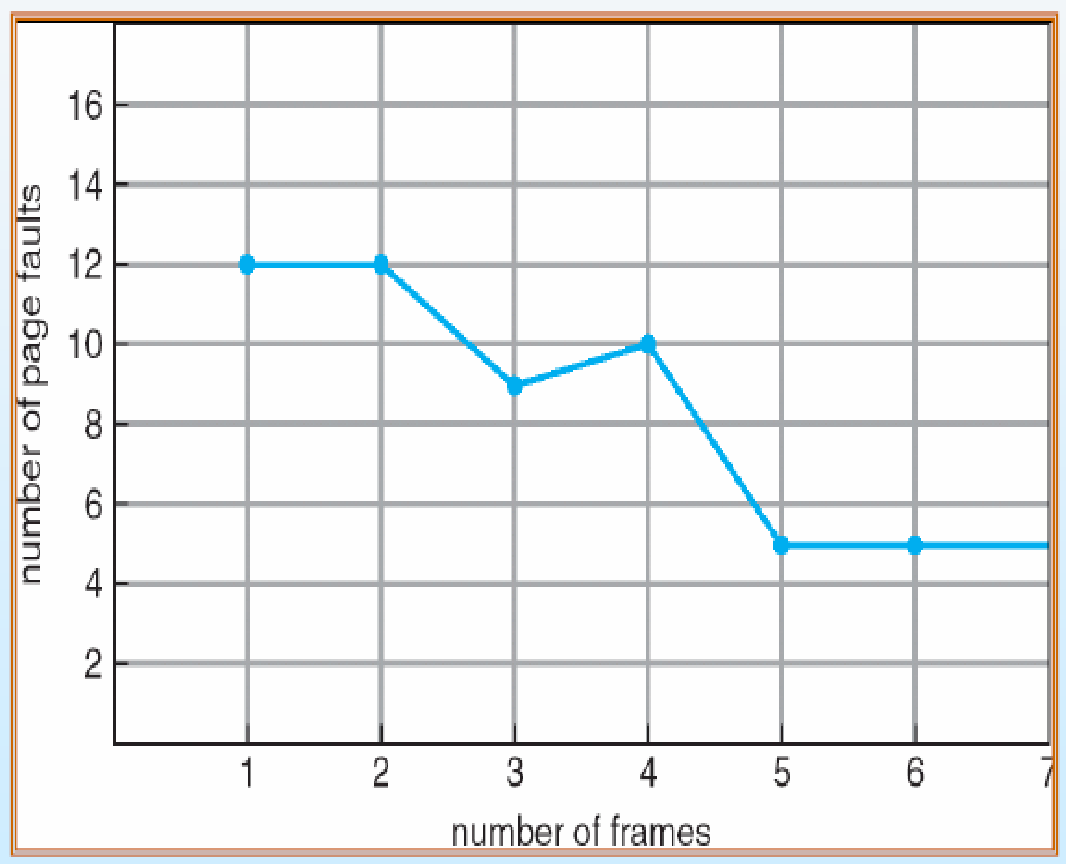
FIFO

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

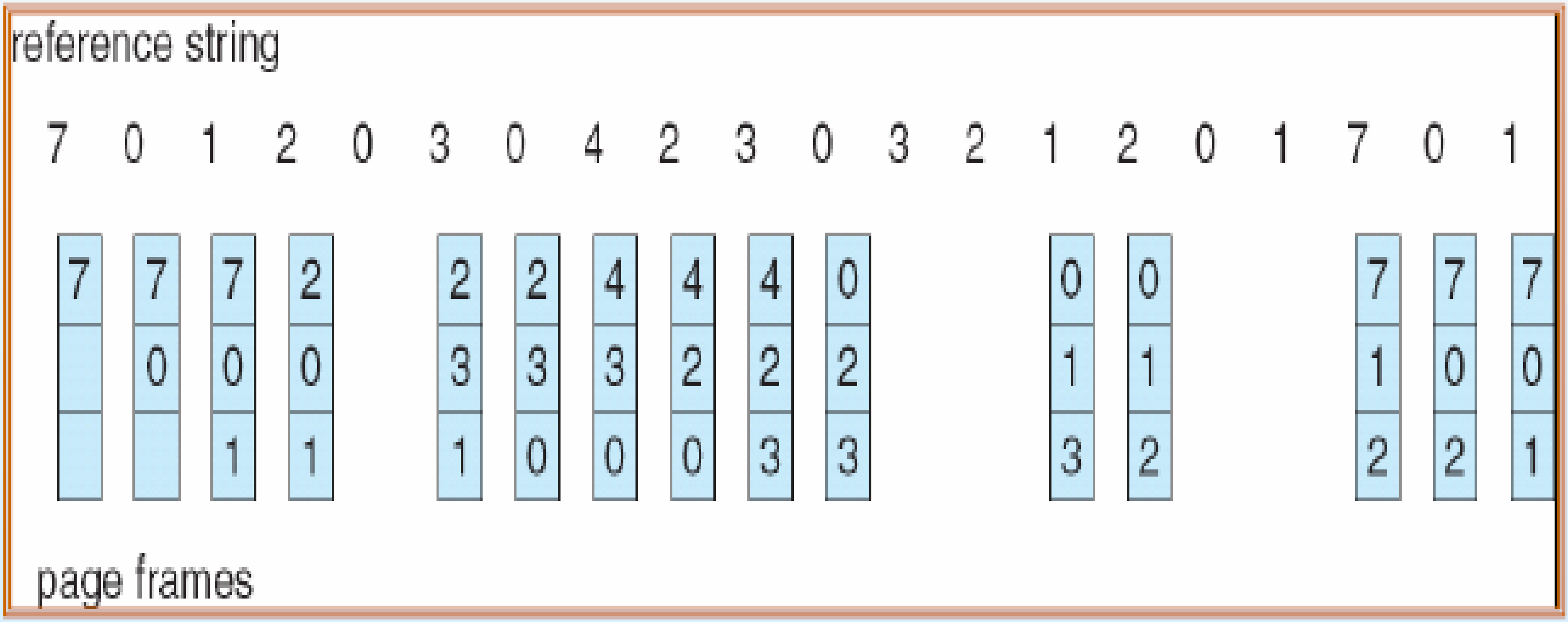
- In all our examples, the reference string is
  1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frame (9 page faults)
- 4 frame (10 page faults)

- Notice that the number of faults for 4 frames is greater than the umber of faults for 3 frames!! This unexpected result is known as **Belady's anomaly**

FIFO Illustrating Belady's Anomaly



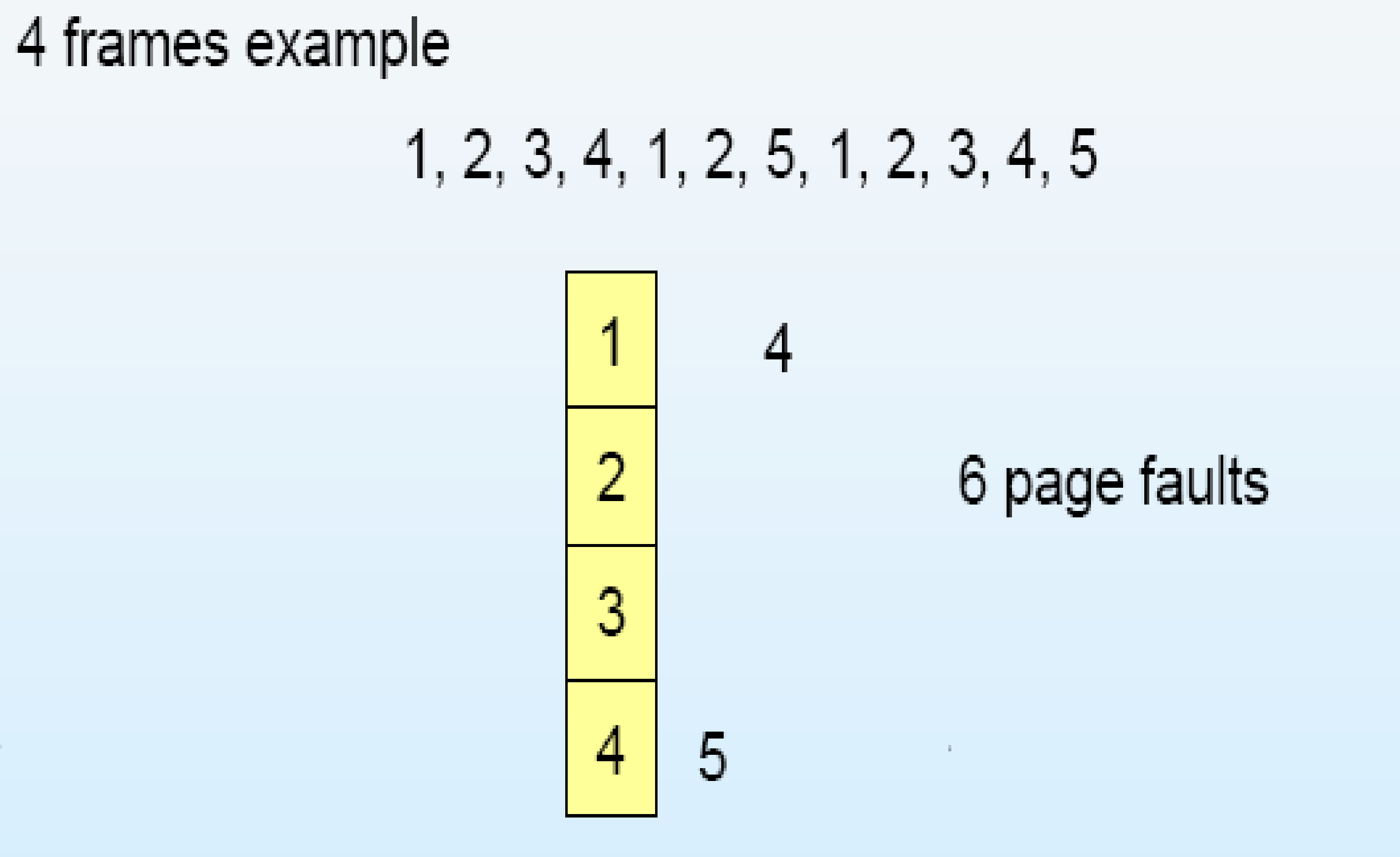
FIFO Algorithm



Optimal Page-Replacement Algorithm

- Replace page that will not be used for longest period of time
- This is a design to guarantee the lowest page-fault rate for a fixed number of frames

Optimal Page-Replacement Algorithm

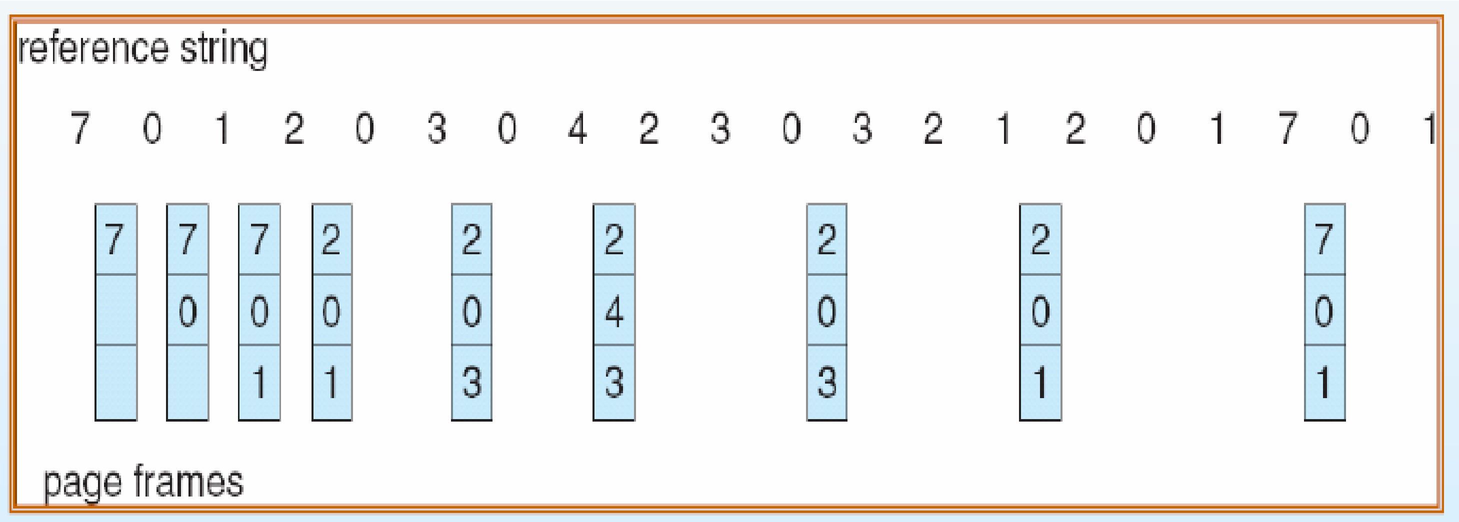


4 frames example

1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

6 page faults

Optimal Page-Replacement Algorithm



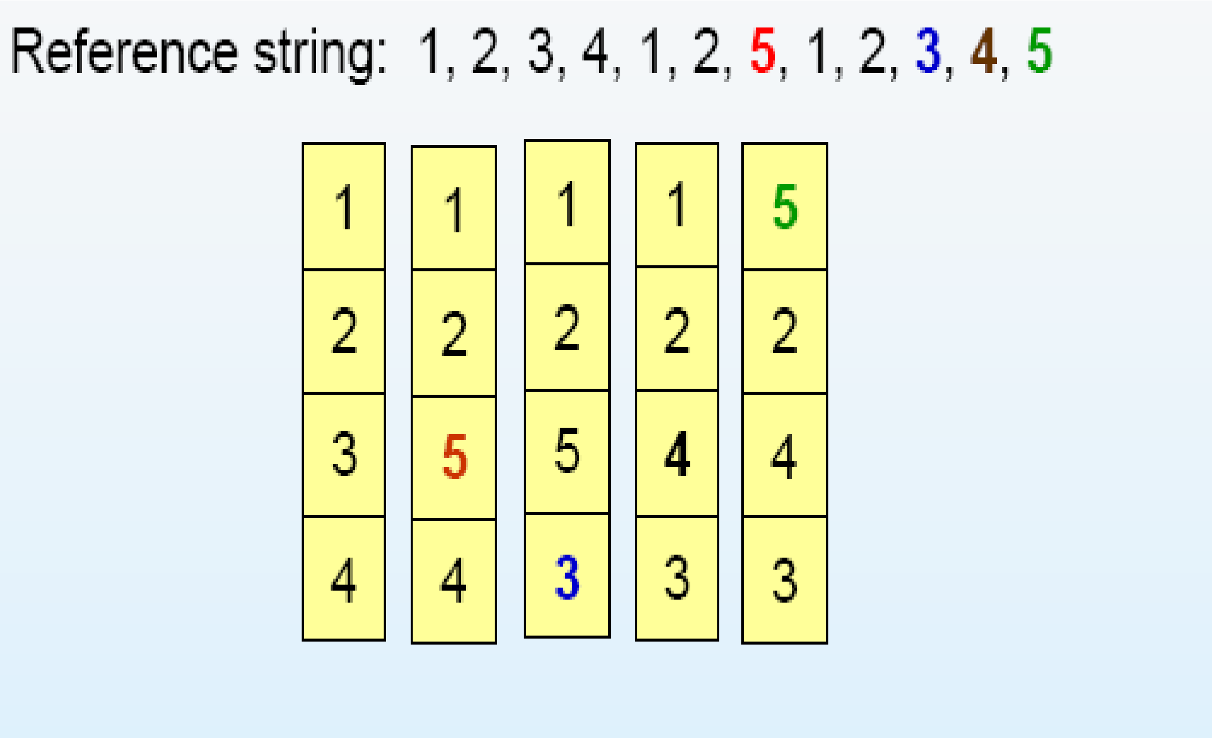
Optimal Page-Replacement Algorithm

- Unfortunately, the optimal page-replacement is difficult to implement, because it requires future knowledge of the reference string

Least-recently-used (LRU) algorithm

- LRU replacement associates with each page the time of that page’s last use
- When a page must be replaced, LRU chooses the page that has not been used for the longest period of time

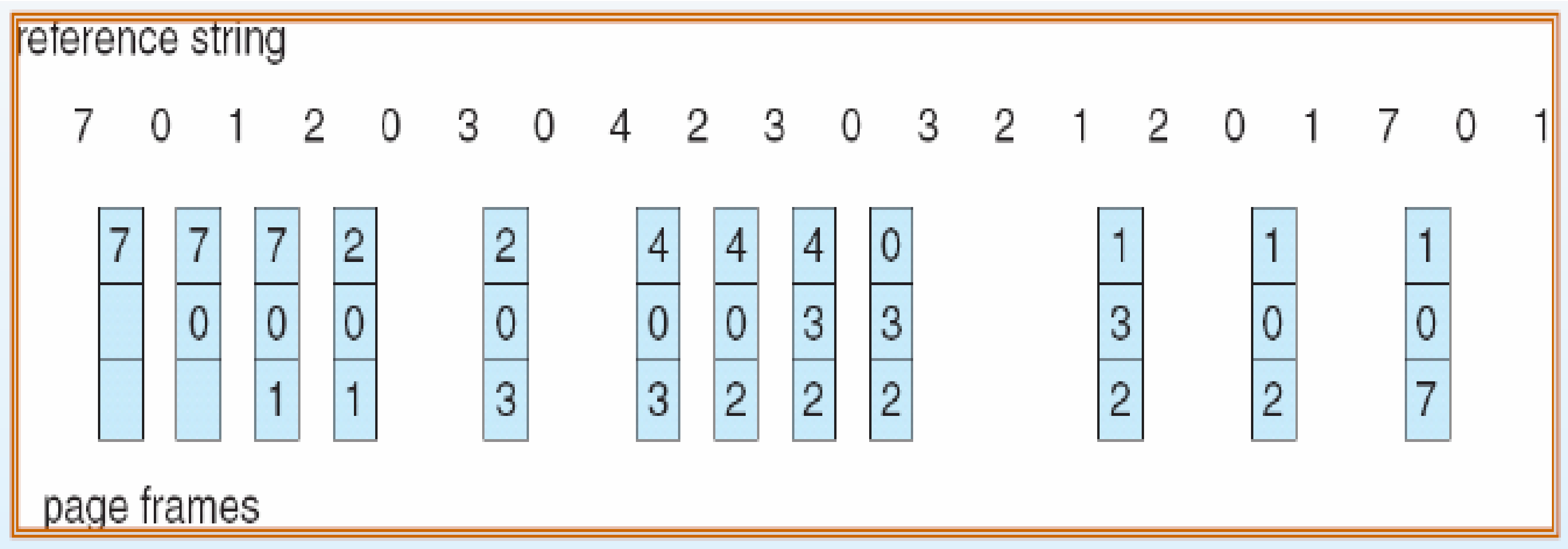
Least-recently-used (LRU) algorithm



Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

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

Least-recently-used (LRU) algorithm

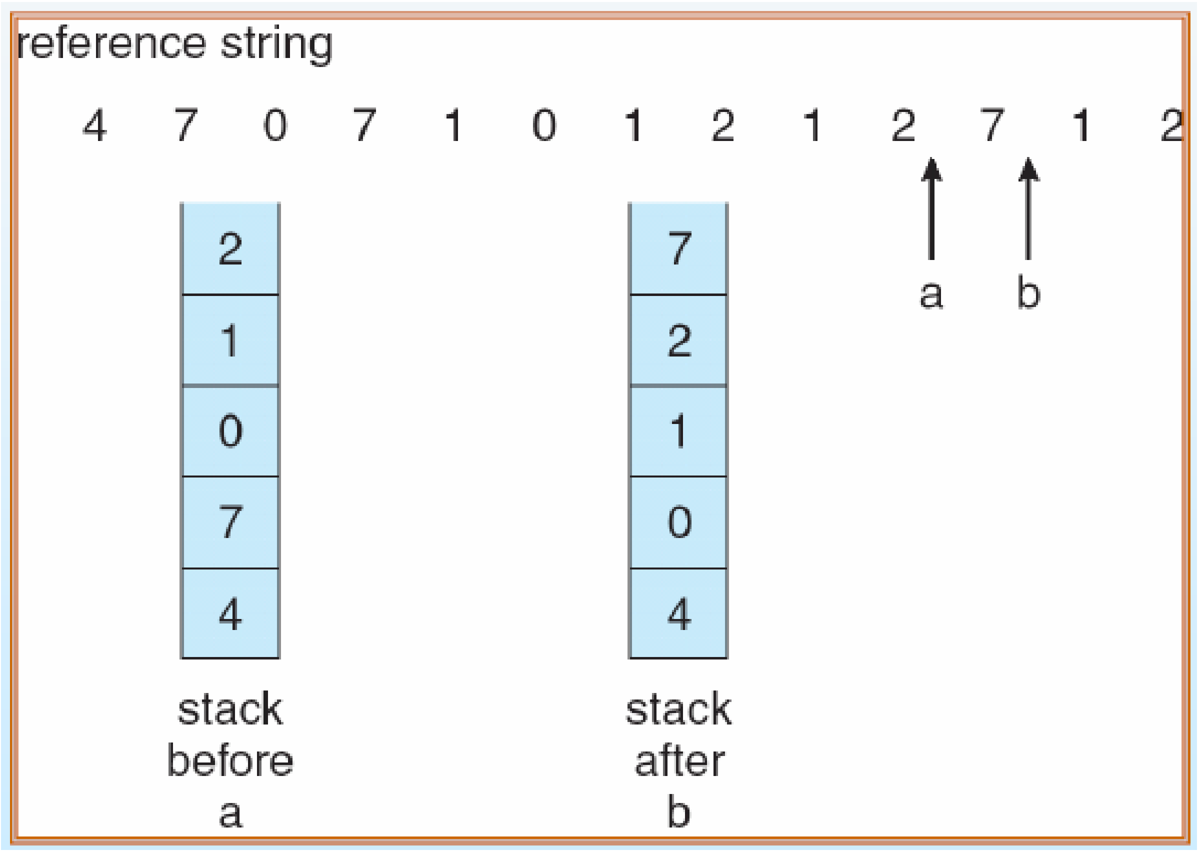


Least-recently-used (LRU) algorithm

- The major problem is how to implement LRU replacement:

1. Counter: whenever a reference to a page is made, the content of the clock register are copied to the time-of-use filed in the page table entry for the page. We replace the page with the **smallest** time value
2. Stack: Whenever a page is referenced, it is removed from the stack and put on the top. In this way, the most recently used page is always at the top of the stack

Stack implementation



Second-Chance Algorithm

- Basically, it’s a FIFO algorithm
- If the page is referenced, we set the bit into 1
- When a page has been selected, we inspect its reference bit.
- If the value is 0, we proceed to replace this page, otherwise, we give the page a second chance and move on to select the next FIFO page
- When a page get a second chance, it’s reference bit is cleared, and its arrival time is reset to the current time

Second-Chance Algorithm

- When a page get a second chance, it’s reference bit is cleared, and its arrival time is reset to the current time
- If a page is used often enough to keep its reference bit set, it will never be replaced

Counting Based Page Replacement

- Least Frequently used (LFU) page-replacement algorithm
- Most frequently used (MFU) page-replacement algorithm