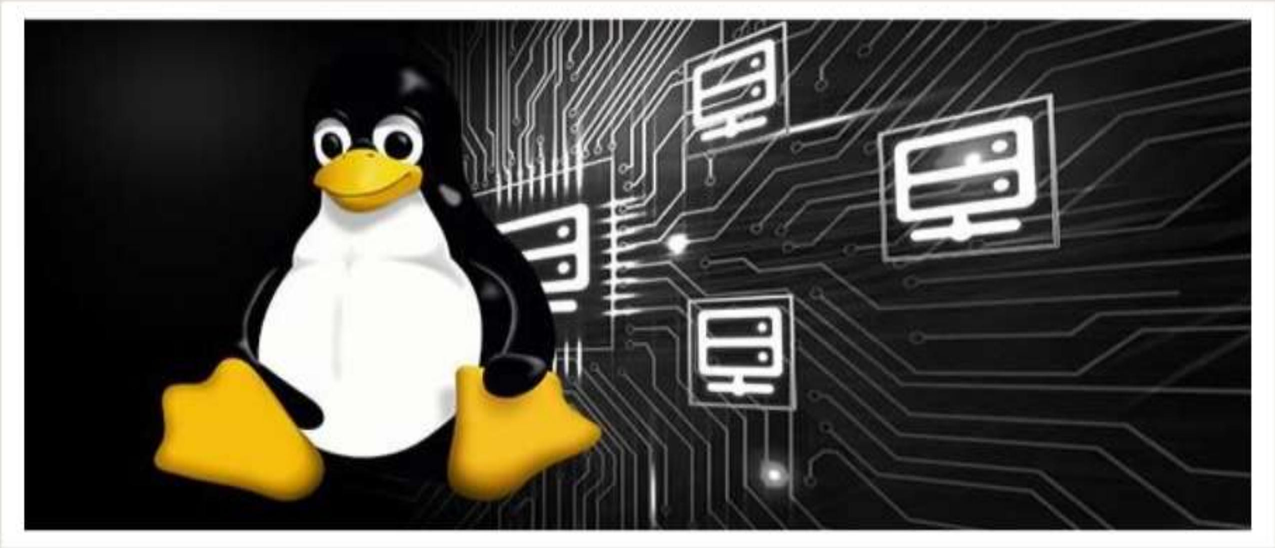


Operating systems

By
I Ravindra kumar, B.Tech, M.Tech,(Ph.D.)
Assistant professor,
Dept of CSE, VNR VJIET

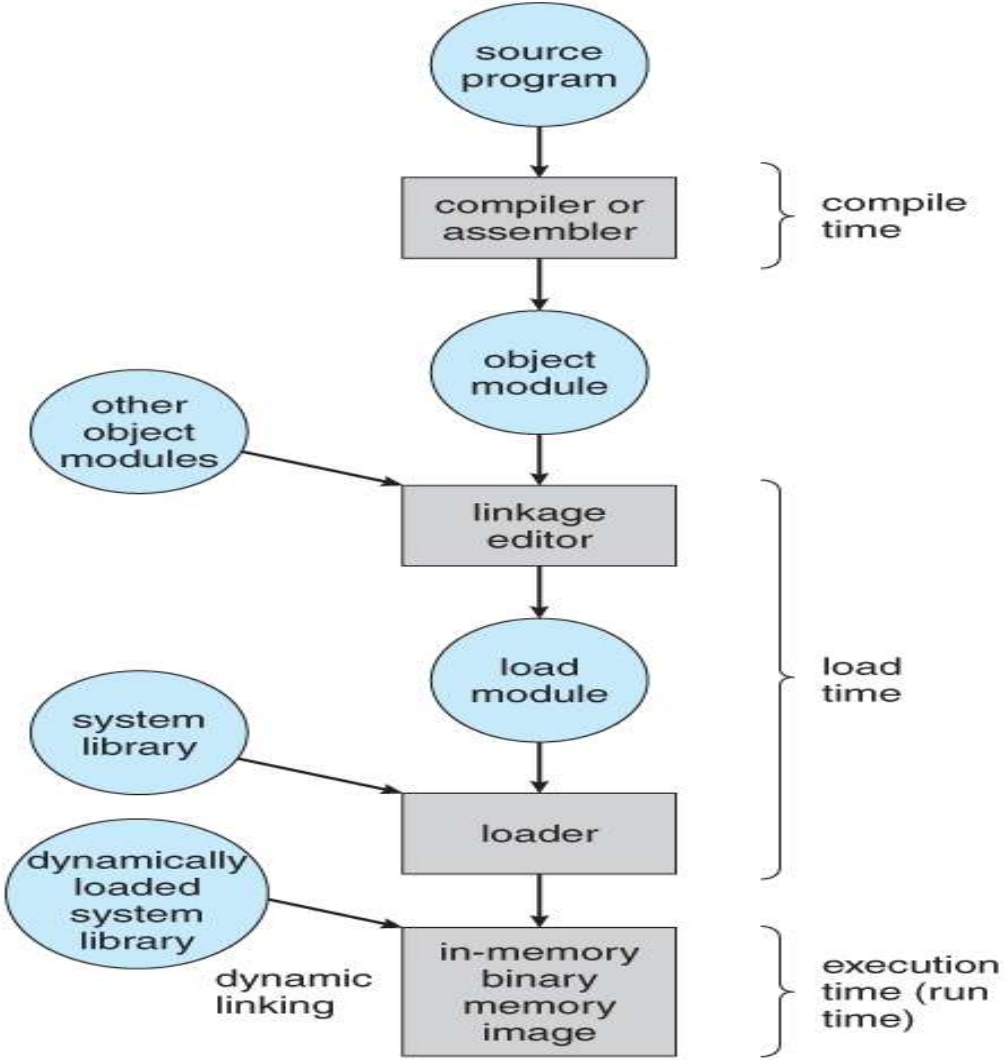
Memory Management

Memory Management :

- Memory consists of a large array of words or bytes, each with its own address.
- The CPU fetches instructions from memory according to the value of the program counter.
- A typical instruction-execution cycle,
- The memory unit sees only a stream of memory addresses; it does not know how they are generated
- we can ignore how a memory address is generated by a program.
    - We are interested in only the sequence of memory addresses generated by the running program.

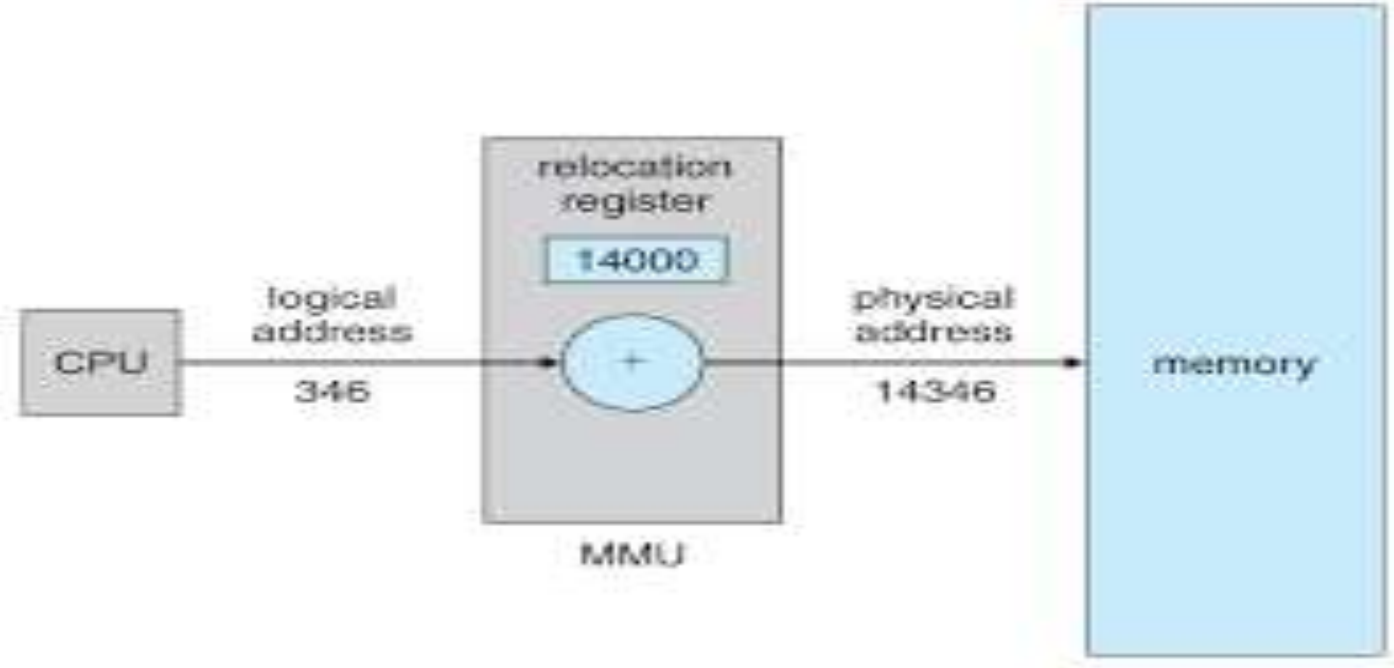
Address Binding:

- a program resides on a disk as a binary executable file.
- The program must be brought into memory and placed within a process for it to be executed.
- The collection of processes on the disk that is waiting to be brought into memory for execution forms the input queue.
- select one of the processes in the input queue and to load that process into memory.


source program
compiler or assembler
compile time
object module
other object modules
linkage editor
load module
system library
loader
load time
dynamically loaded system library
dynamic linking
in-memory binary memory image
execution time (run time)

Logical- Versus Physical-Address Space:

- Address generated by the CPU is commonly referred to as a logical address,
- an address seen by the memory unit-that is, the one loaded into the memory-address register of the memory-is commonly referred to as a physical address.
- usually refer to the logical address as a virtual address.
- set of all logical addresses generated by a program is a logical-address space;
- the set of all physical addresses corresponding to these logical addresses is a physical-address space.
- run-time mapping from virtual to physical addresses is done by a hardware device called the memory-management unit (MMU).
- The final location of a referenced memory address is not determined until the reference is made.



Dynamic Loading:

- The size of a process is limited to the size of physical memory.
- dynamic loading, a routine is not loaded until it is called.
- All routines are kept on disk in a relocatable load format.
- The main program is loaded into memory and is executed.

Dynamic Linking and Shared Libraries:

- **Static linking**,which system language libraries are treated like any other object module and are combined by the loader into the binary program image.
- **Dynamic linking**, a stub is included in the image for each library-routine reference.
- stub is a small piece of code that indicates
  - stub is executed, it checks to see whether the needed routine is already in memory.
  - If not, the program loads the routine into memory.
  - Either way, the stub replaces itself with the address of the routine, and executes the routine.
- More than one version of a library may be loaded into memory,
  - each program uses its version information to decide which copy of the library to use.
  - Thus, only programs that are compiled with the new library version are affected by the incompatible changes incorporated in it.
- Other programs linked before the new library was installed will continue using the older library. This system is also known as shared libraries.

Overlays:

- To enable a process to be larger than the amount of memory allocated to it
- Idea-
  - to keep in memory only those instructions and data that are needed at any given time.
  - When other instructions are needed, they are loaded into space occupied previously by instructions that are no longer needed.

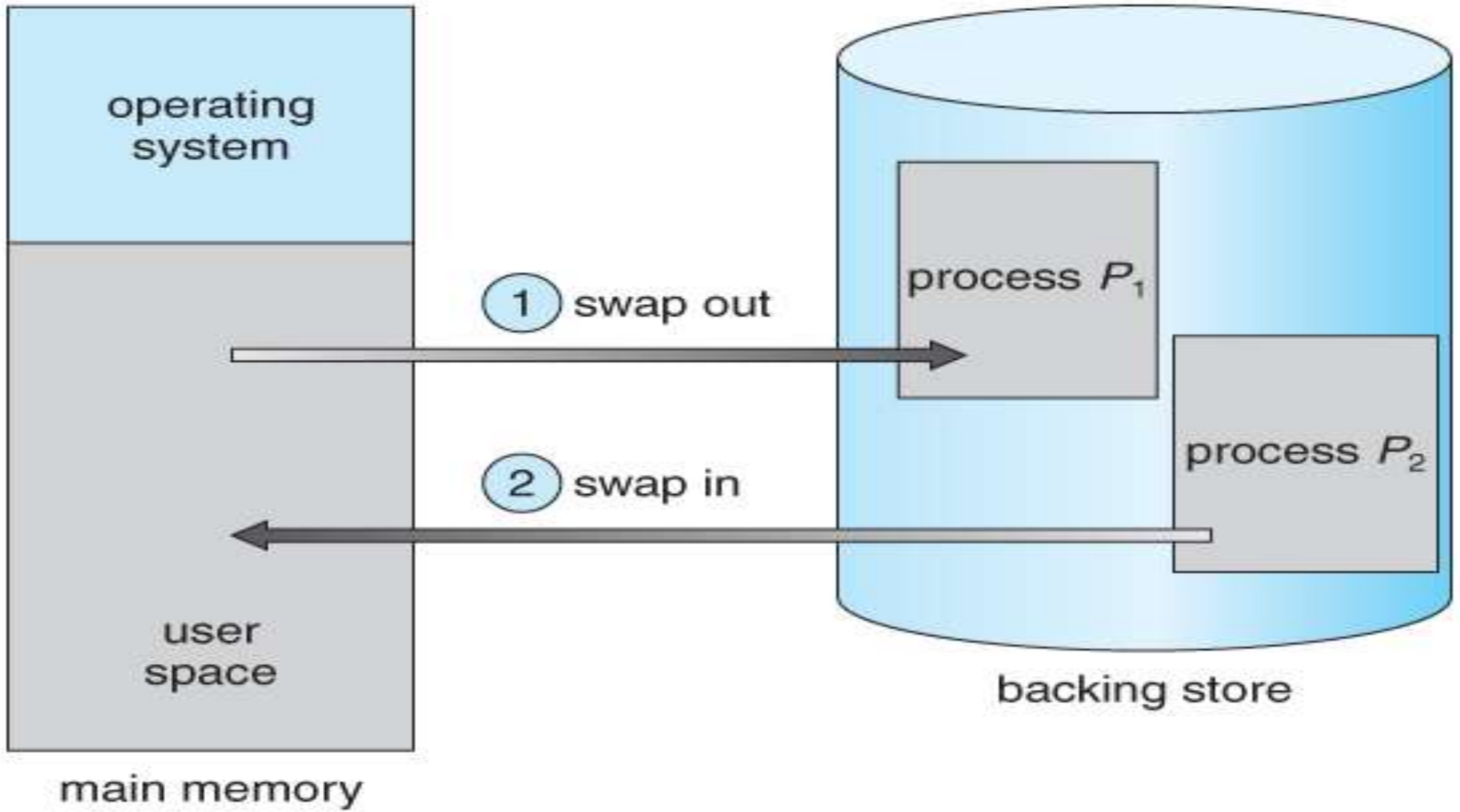
Swapping:

- A process needs to be in memory to be executed.
- A process, however, can be swapped temporarily out of memory to a backing store, and then brought back into memory for continued execution.
- A variant of this swapping policy is used for priority-based scheduling algorithms
  - When the higher-priority process finishes, the lower-priority process can be swapped back in and continued.
  - This variant of swapping is sometimes called roll out, roll in.

- The system maintains a ready queue consisting of all processes whose memory images are on the backing store or in memory and are ready to run.
- Whenever the CPU scheduler decides to execute a process, it calls the dispatcher.
- The dispatcher checks to see whether the next process in the queue is in memory.
- If not, and there is no free memory region, the dispatcher swaps out a process currently in memory and swaps in the desired process.
- It then reloads registers as normal and transfers control to the selected process.
- The context-switch time in such a swapping system is fairly high.
- To get an idea of the context-switch time,

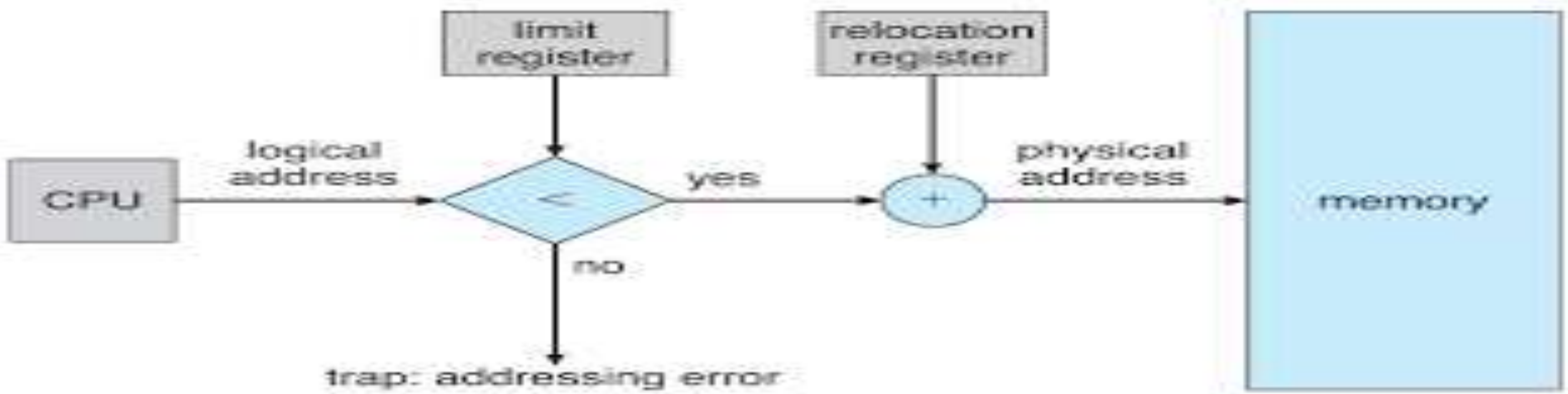
- let us assume that the user process is of size 1MB
- the backing store is a standard hard disk with a transfer rate of 5 MB per second.
- The actual transfer of the 1 MB process to or from memory takes

1000 KB/5000 KB per second = 1 /5 second = 200 milliseconds.


operating system
user space
main memory
1 swap out
2 swap in
process P_1
process P_2
backing store

Contiguous Memory Allocation:

- The memory is usually divided into two partitions:
  - one for the resident operating system, and one for the user processes.
  - We may place the operating system in either low memory or high memory.
- contiguous memory allocation, each process is contained in a single contiguous section of memory.
- Issue-protecting the operating system from user processes, and protecting user processes from one another.
- provide this protection by using a relocation register, with a limit register,



Memory Allocation:

- simplest method for memory allocation (called MFT) is
  - to divide memory into several fixed-sized partitions.
  - Each partition may contain exactly one process
  - the degree of multiprogramming is bound by the number of partitions.
  - it is no longer in use.
- a generalization of the fixed-partition scheme (called MVT);
  - it is used primarily in a batch environment.
  - The operating system keeps a table indicating which parts of memory are available and which are occupied.
  - initially, all memory is available for user processes, and is considered as one large block of available memory, a hole.
  - When a process arrives and needs memory, we search for a hole large enough for this process.
  - If we find one, we allocate only as much memory as is needed,
  - keeping the rest available to satisfy future requests.
- how to satisfy a request of size n from a list of free holes.
  - Firstfit: Allocate the first hole that is big enough.
  - Bestfit: Allocate the smallest hole that is big enough.
  - Worst fit: Allocate the largest hole.that is big enough.

Fragmentation:

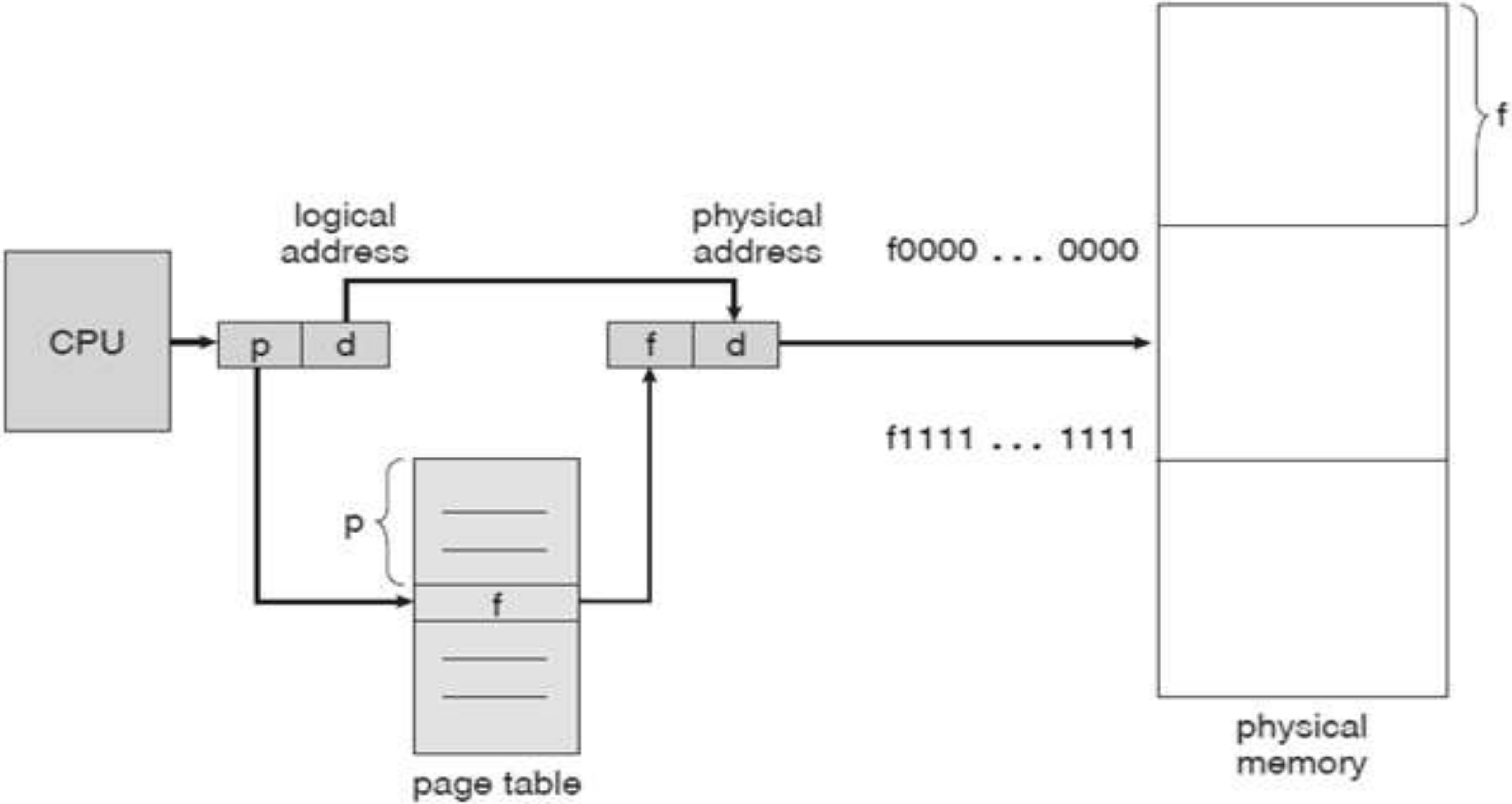
- The memory allocated to a process may be slightly larger than the requested memory.
- The difference between these two numbers is internal fragmentation-memory that is internal to a partition but is not being used.
- External fragmentation exists when enough total memory space exists to satisfy a request, but it is not contiguous;
    - storage is fragmented into a large number of small holes.
- One solution to the problem of external fragmentation is compaction.
- Compaction algorithm is simply to move all processes toward one end of memory; all holes move in the other direction, producing one large hole of available memory.
    - This scheme can be expensive.
- Another possible solution to the external-fragmentation problem is to permit the logical address space of a process to be noncontiguous,
- solution: paging and segmentation .These techniques can also be combined

Paging:

- 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.
- Every address generated by the CPU is divided into two parts:

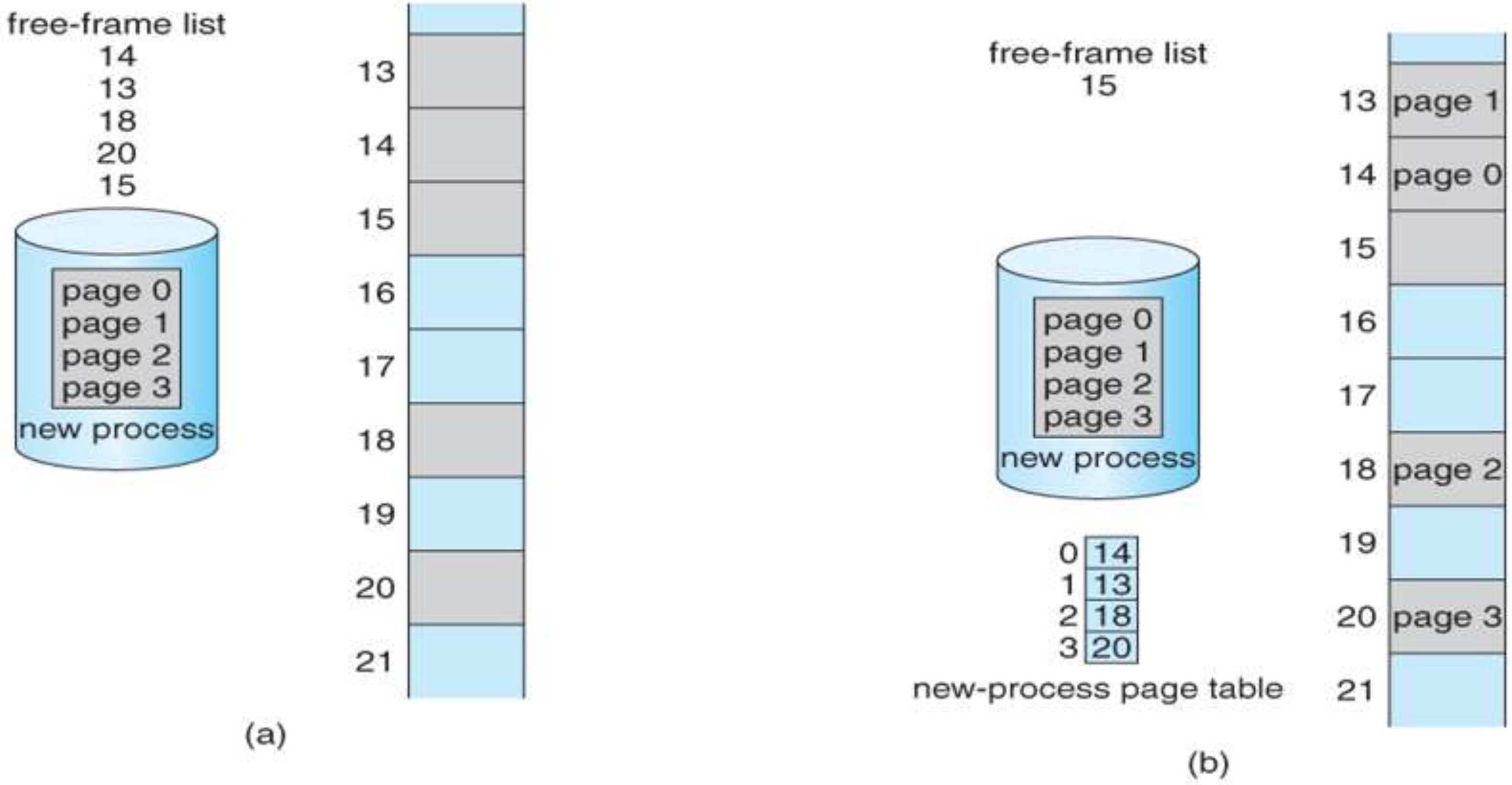
  a page number (p) and a page offset (d).

- The page number is used as an index into a page table.
- The page size (like the frame size) is defined by the hardware.
  - The size of a page is typically a power of 2, varying between 512 bytes and 16 MB per page, depending on the computer architecture.
- 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

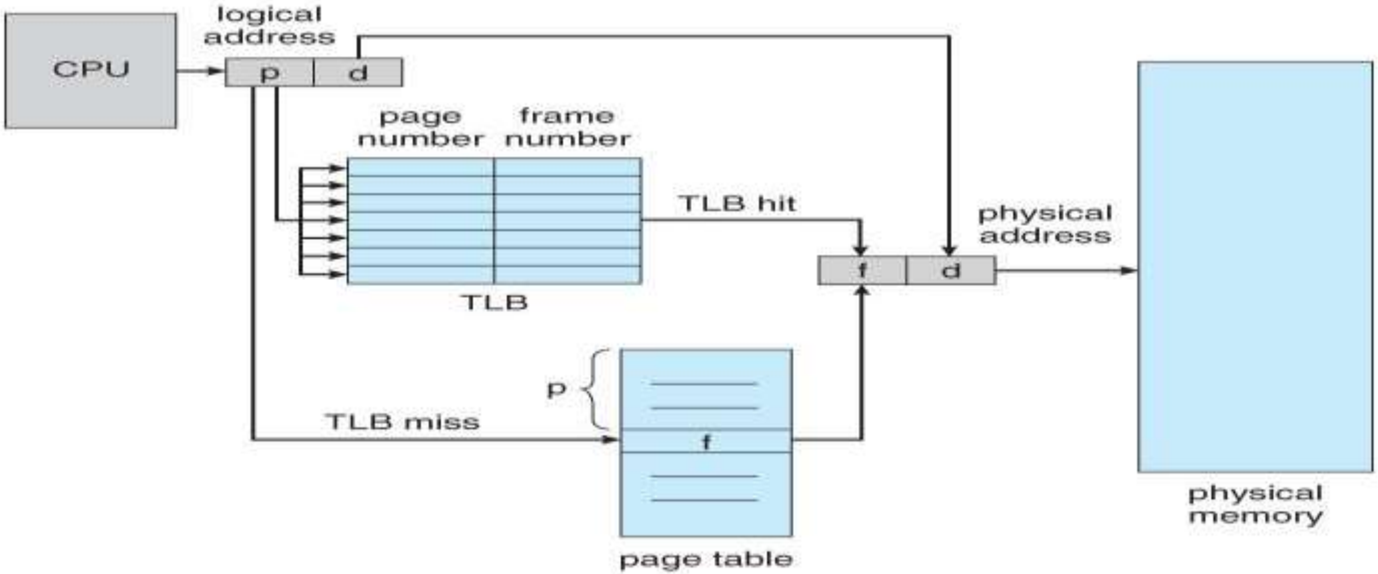
- An important aspect of paging is the clear separation between the user's view of memory and the actual physical memory



Hardware Support

to provide hardware support for this operation, in order to make it as fast as possible and to make process switches as fast as possible also

- ○to use a set of registers for the page table.
- ○ to store the page table in main memory, and to use a single register (called the ***page-table base register, PTBR***) to record where in memory the page table is locate
- ○to use a very special high-speed memory device called the ***translation look-aside buffer, TLB***



The percentage of times that a particular page number is found in the TLB is called the **hit ratio.**

- An 80-percent hit ratio means that we find the desired page number in the TLB 80 percent of the time.
- If it takes 20 nanoseconds to search the TLB, and 100 nanoseconds to access memory,
- then a mapped memory access takes 120 nanoseconds when the page number is in the TLB

Effective access Time=Hit time+miss time

=(0.80*120)+(0.20*220)

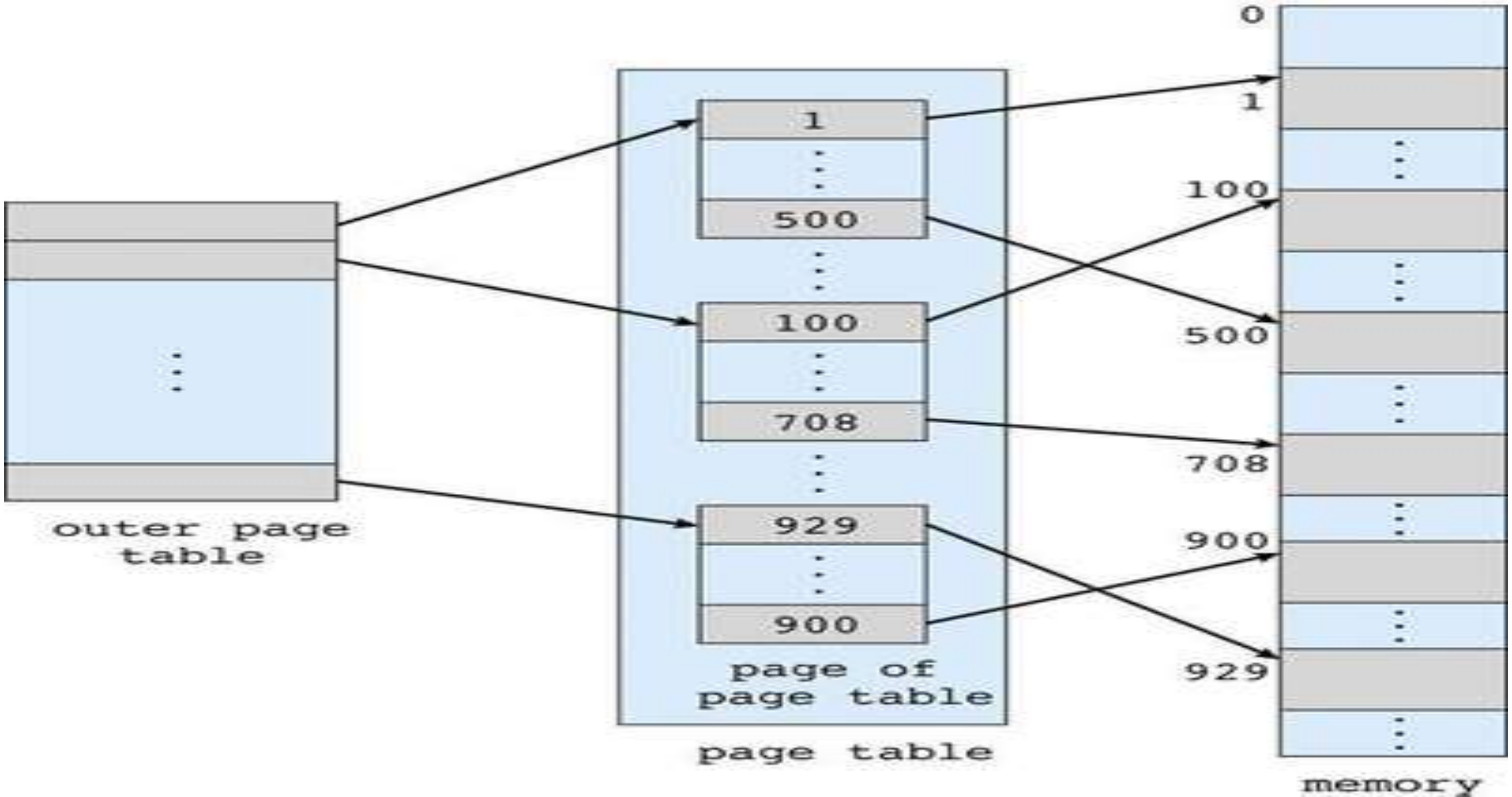
=140 nano seconds

Protection:

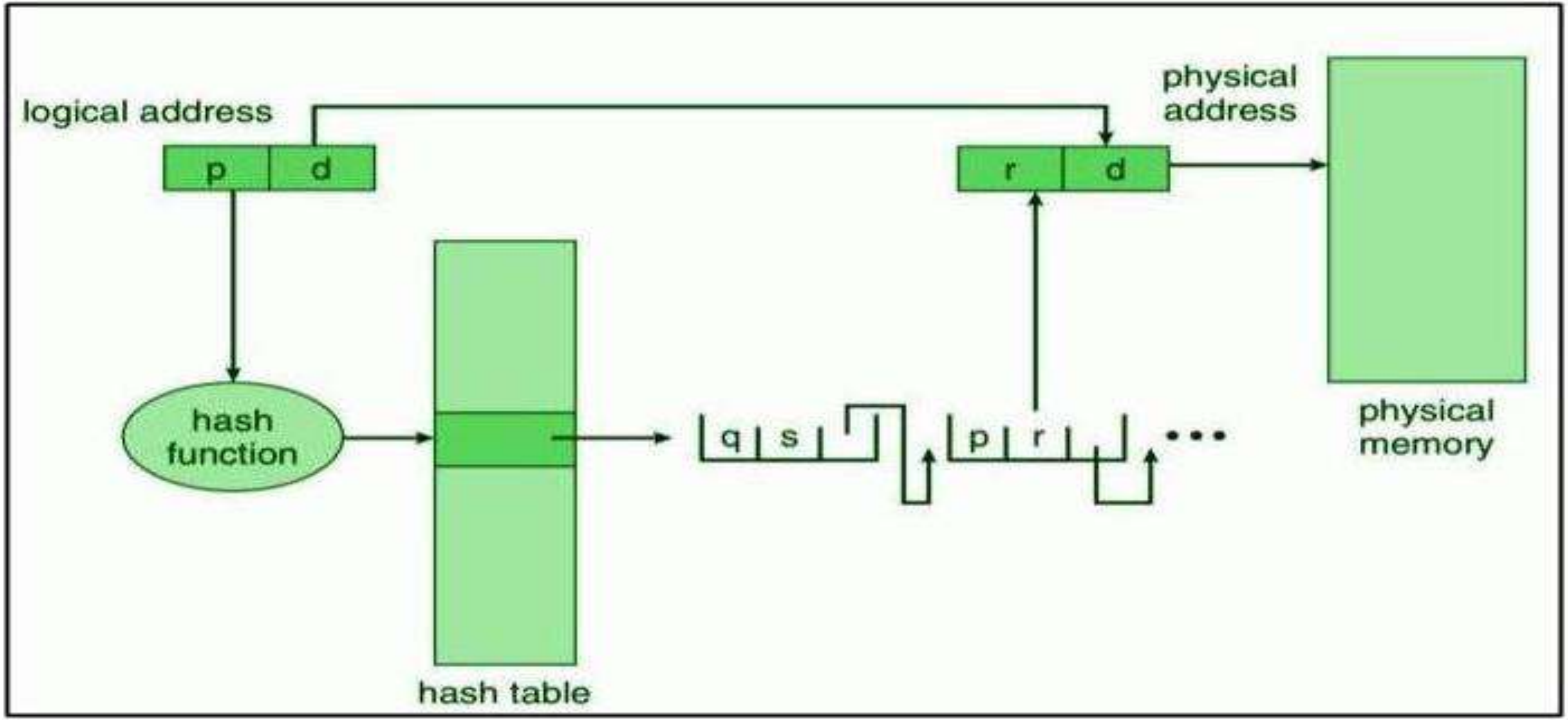
Memory protection in a paged environment is accomplished by protection bits that are associated with each frame. Normally, these bits are kept in the page table.

Structure of the Page Table:

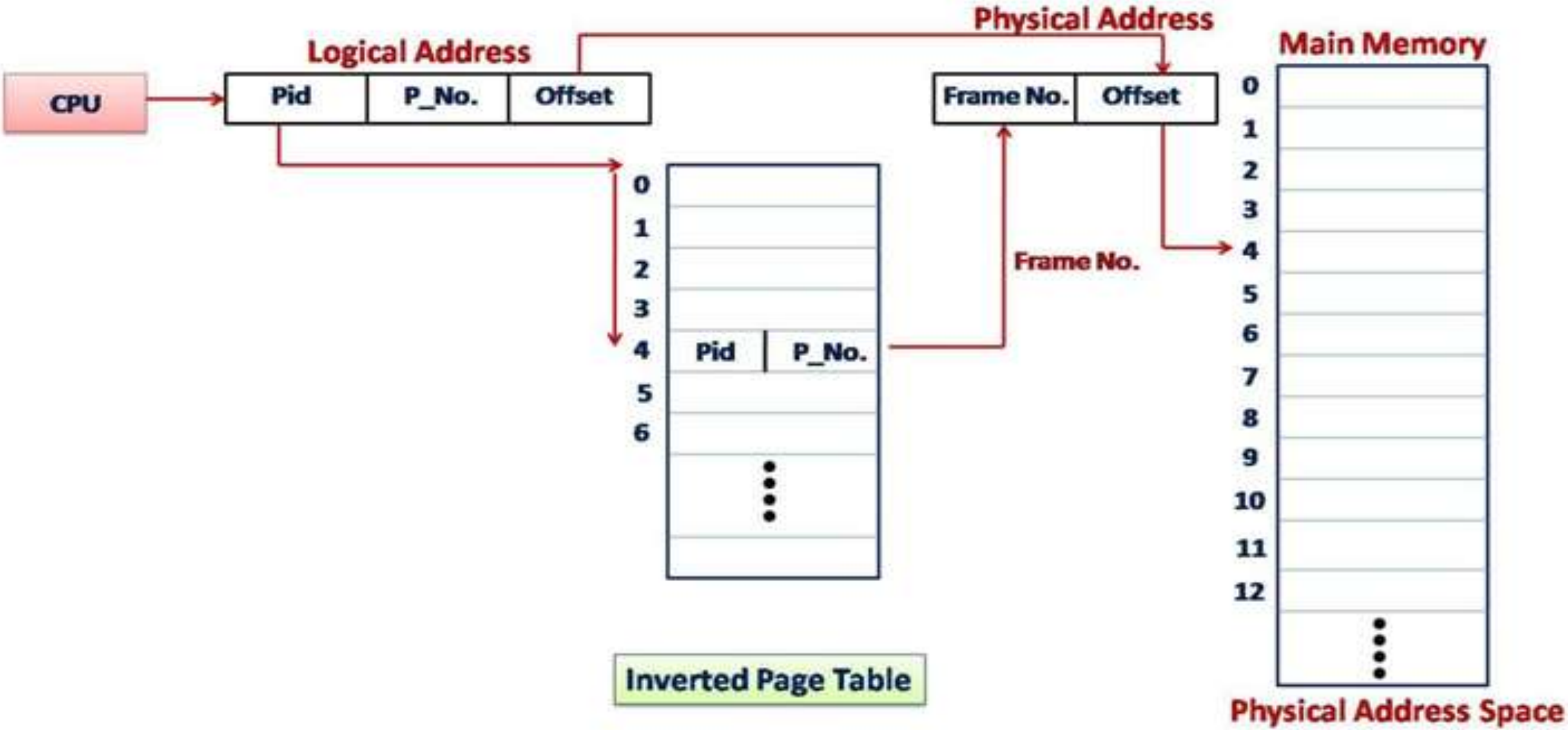
Hierarchical Paging:



Hashed Page Tables:

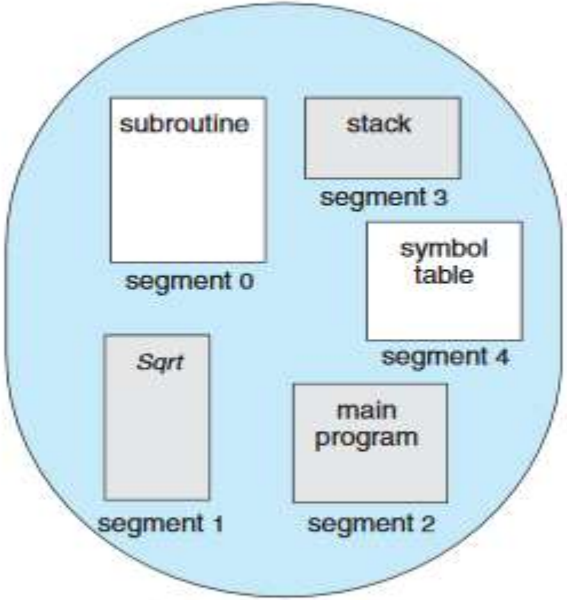


Inverted Page Table:



Segmentation

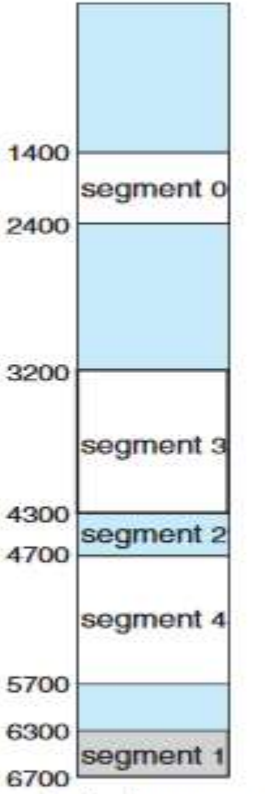
- paging is the separation of the user's view of memory and the actual physical memory.
- The user's view of memory is not the same as the actual physical memory.



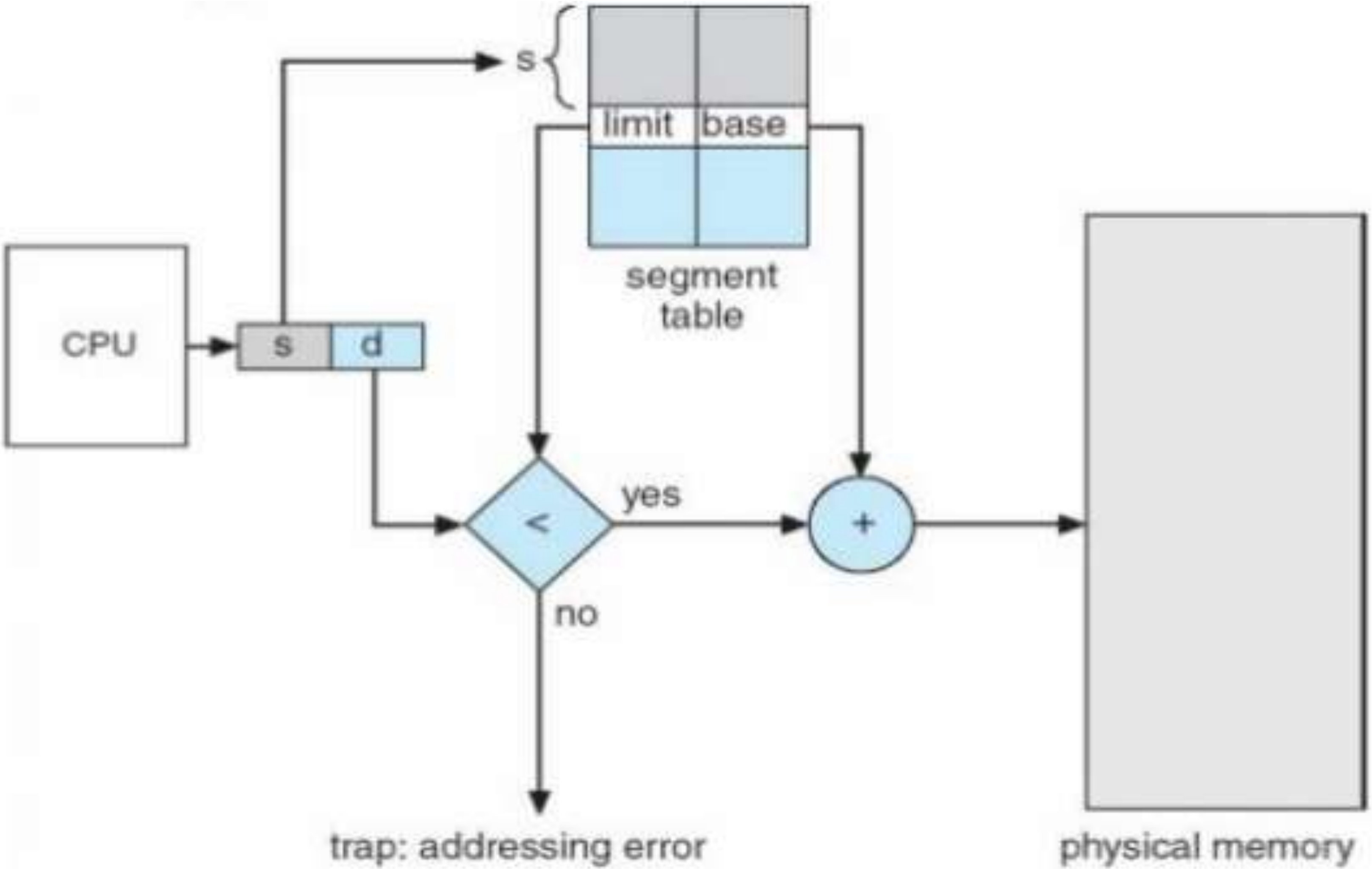
logical address space

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

segment table

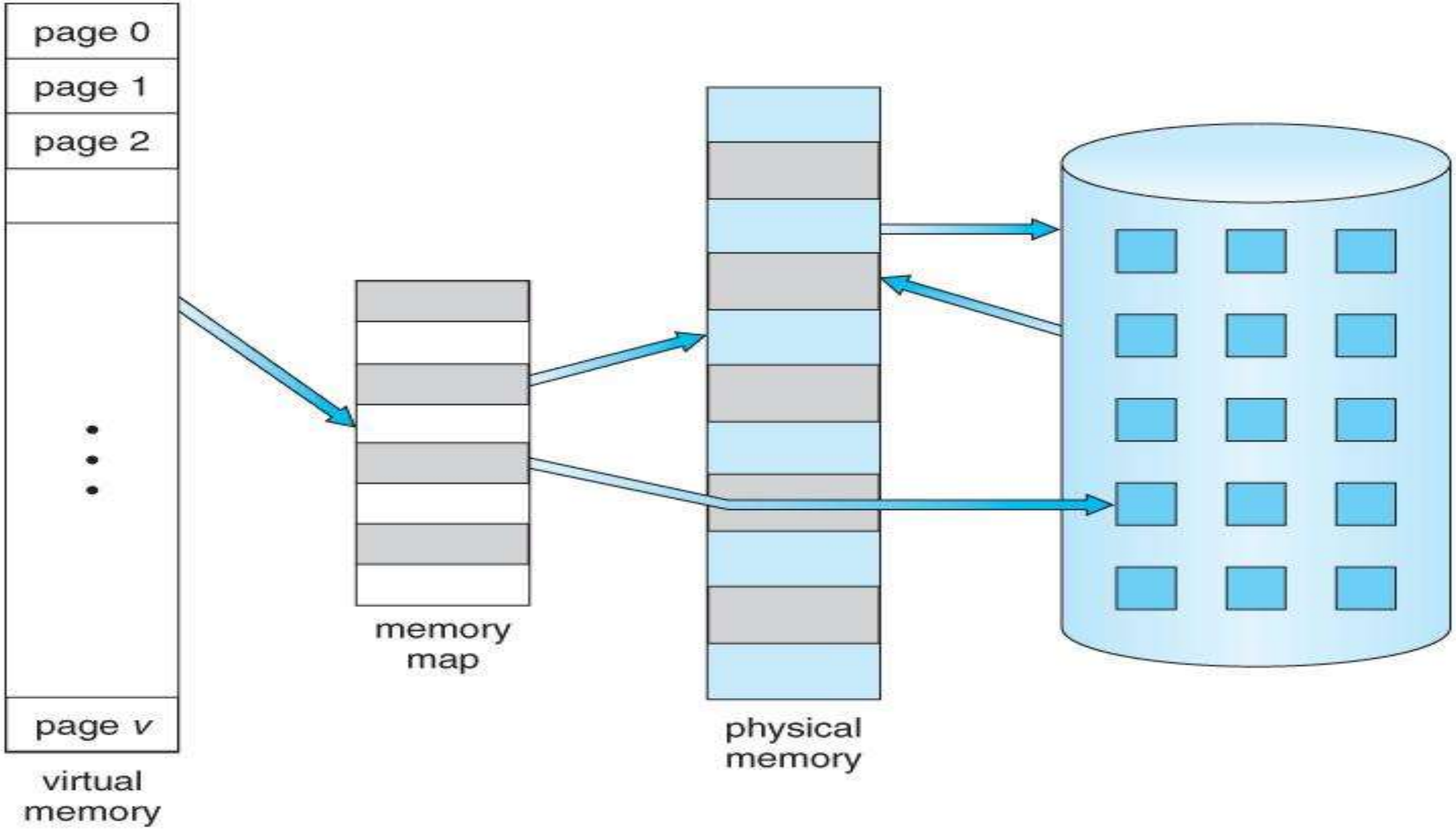


physical memory


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

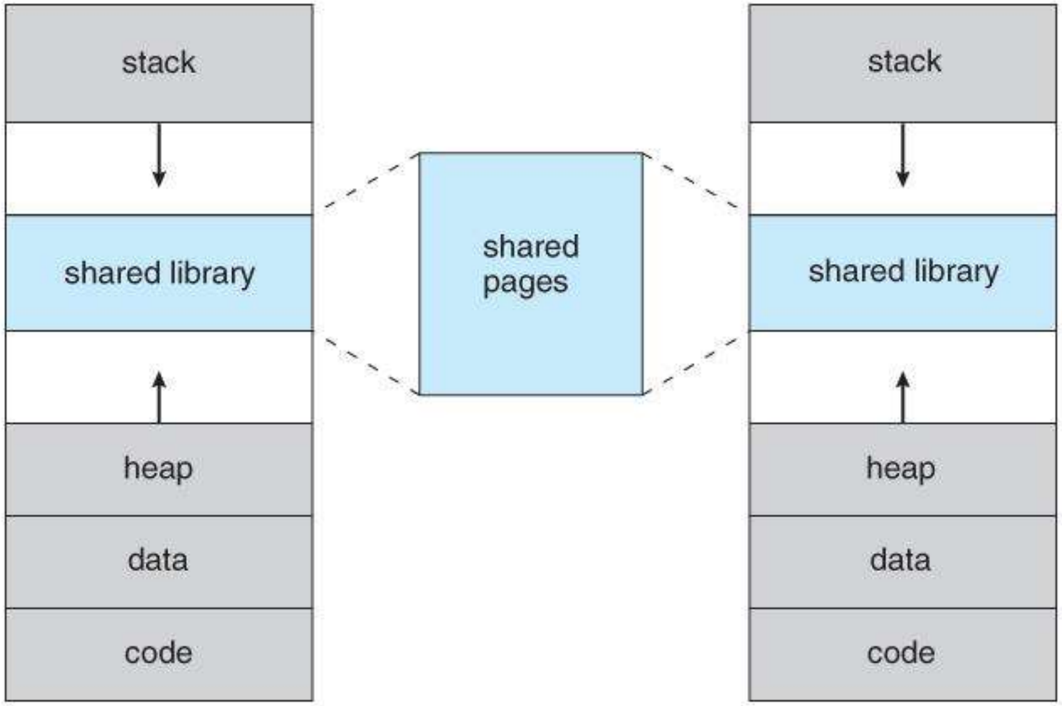
Virtual memory

- is a technique that allows the execution of processes that may not be completely in memory



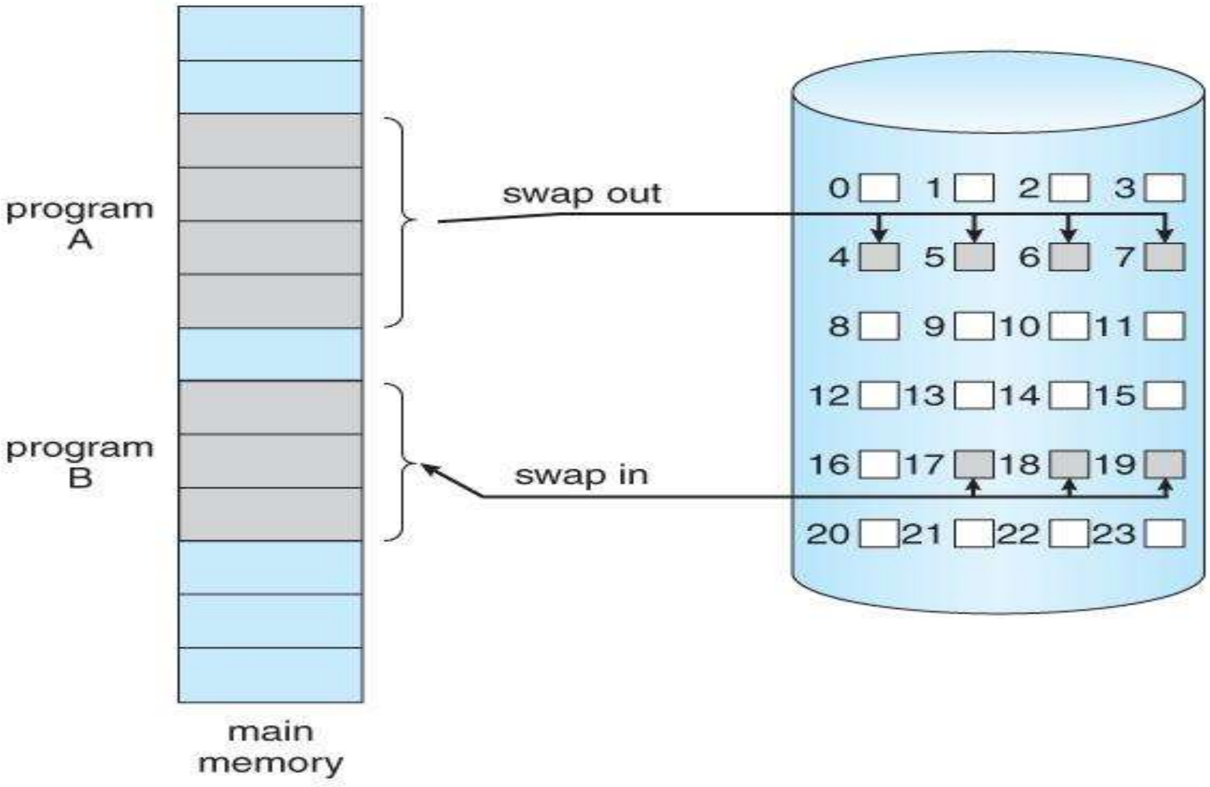
Virtual memory also allows the sharing of files and memory by multiple processes, with several benefits:

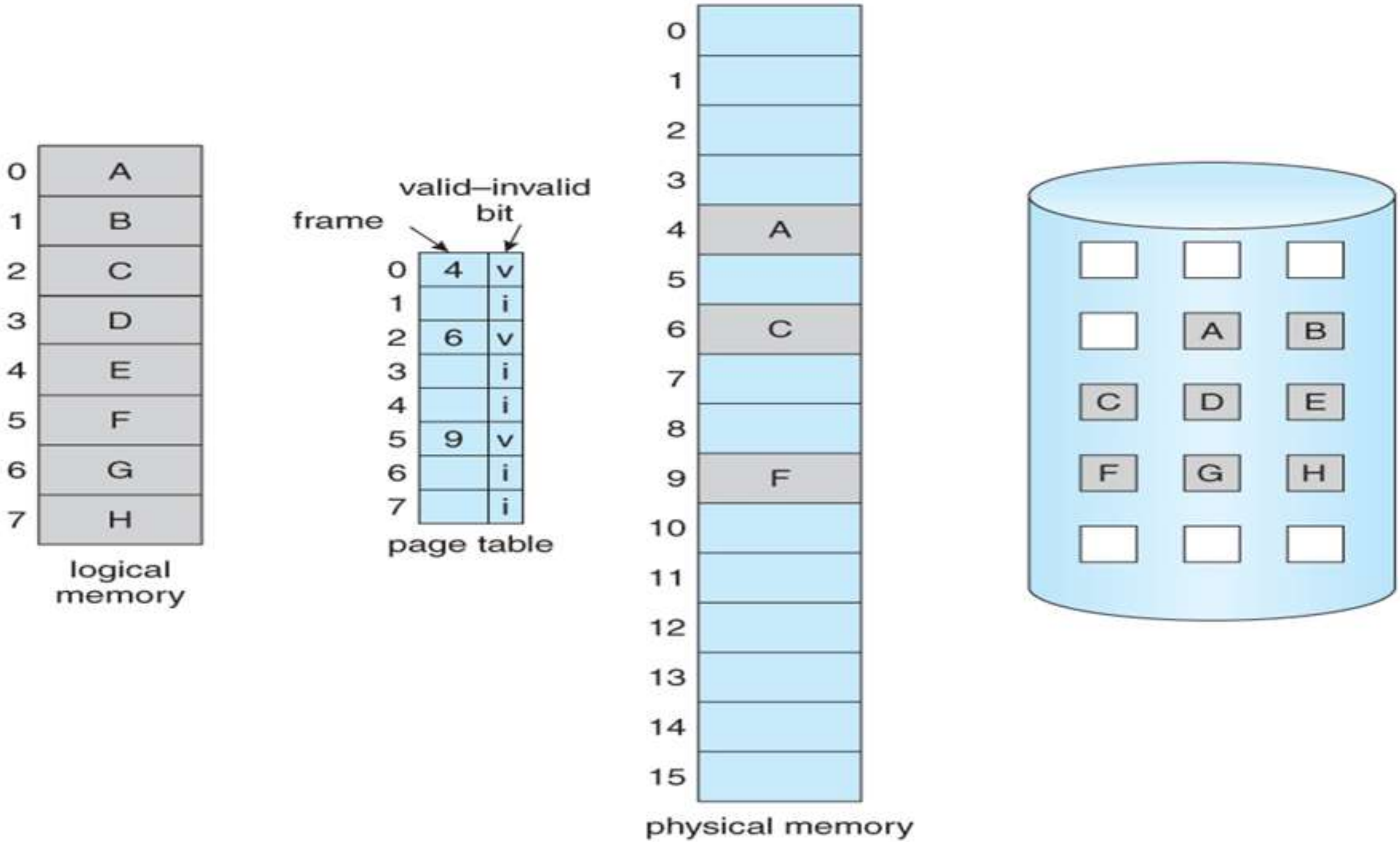
- System libraries
- Processes can also share virtual memory by mapping the same block of memory to more than one process.
- Process pages can be shared during a fork( ) system call,



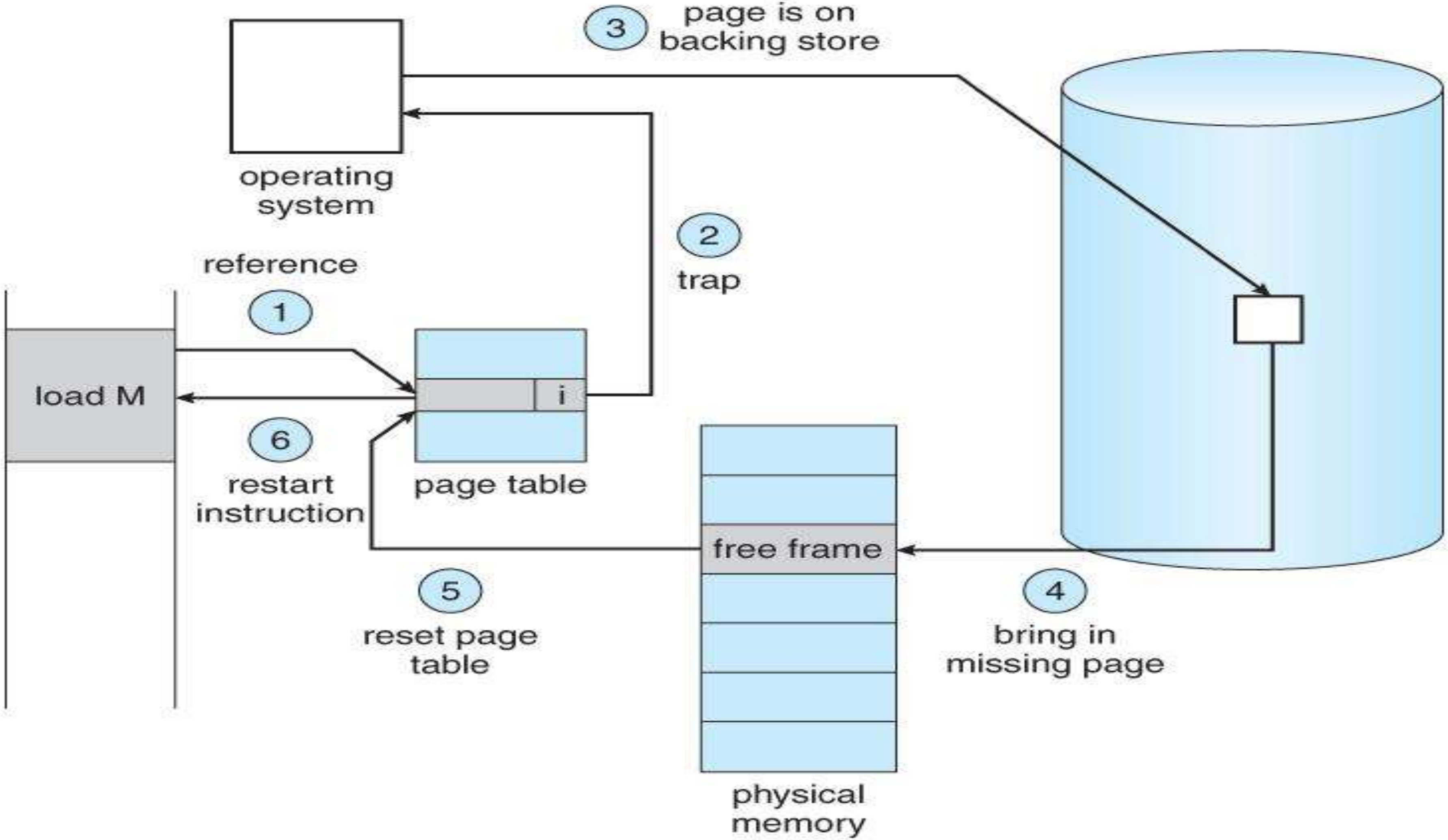
Demand Paging

- when a process is swapped in, its pages are not swapped in all at once.
- Rather they are swapped in only when the process needs them. (on demand.)
- This is termed a ***lazy swapper***, although a ***pager*** is a more accurate term.



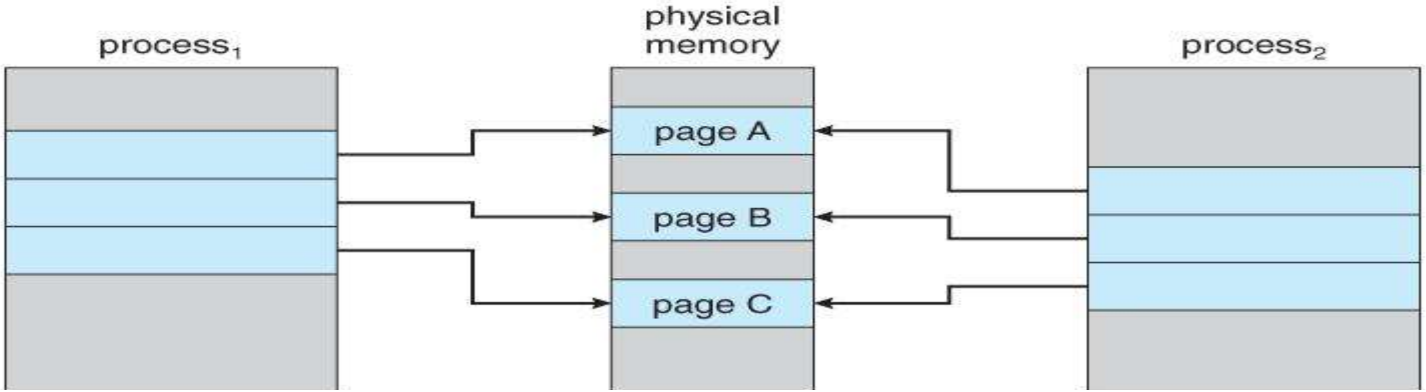


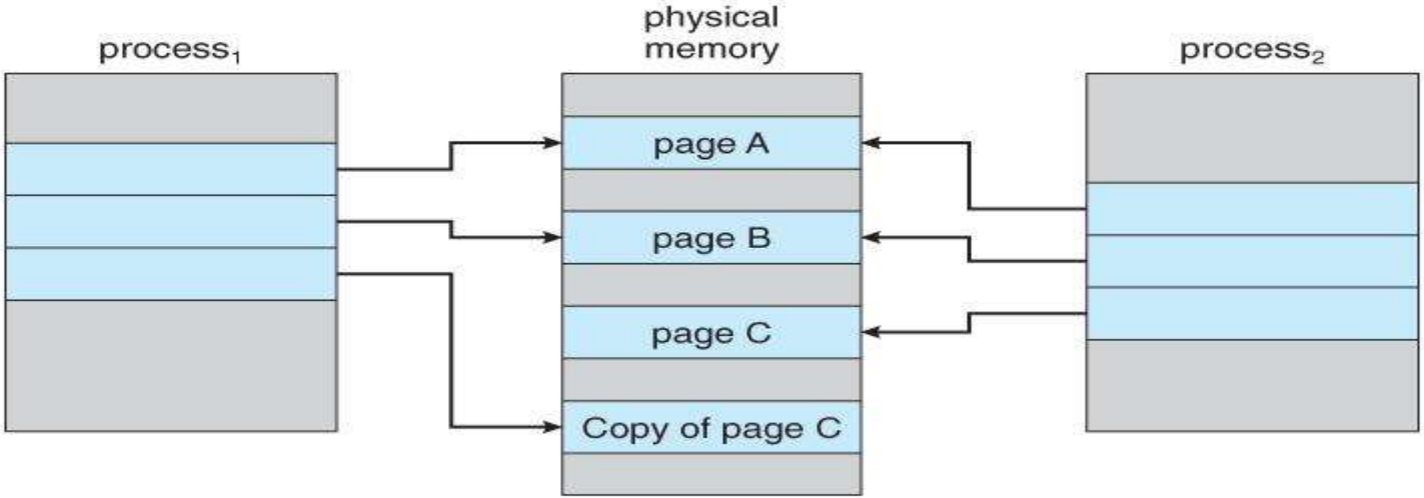
0
1
2
3
4
5
6
7
A
B
C
D
E
F
G
H
logical memory
valid–invalid bit
frame
0 4 v
1 i
2 6 v
3 i
4 i
5 9 v
6 i
7 i
page table
0
1
2
3
4 A
5
6 C
7
8
9 F
10
11
12
13
14
15
physical memory
A
B
C
D
E
F
G
H



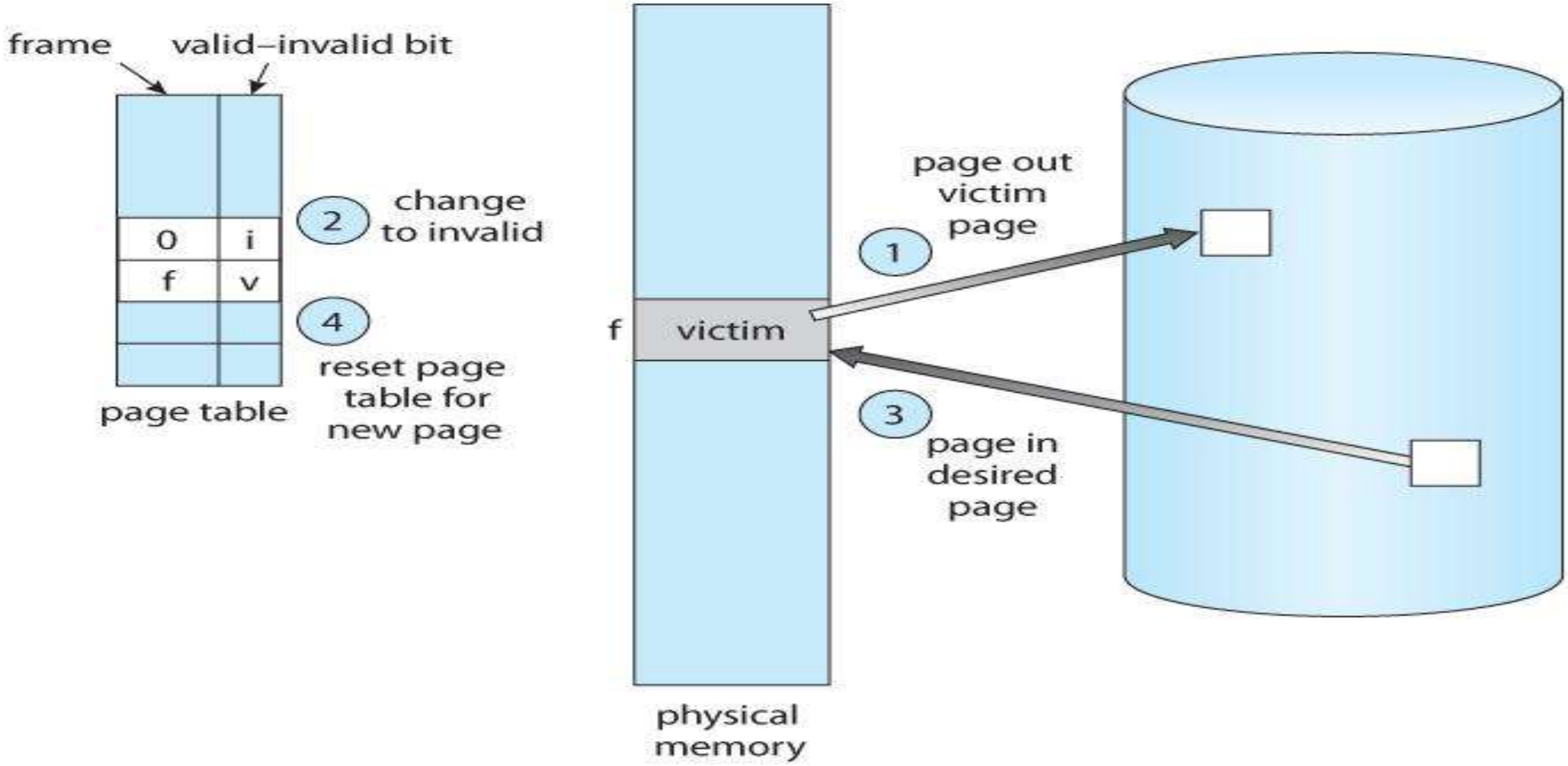
3
page is on
backing store
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory

Copy-on-Write



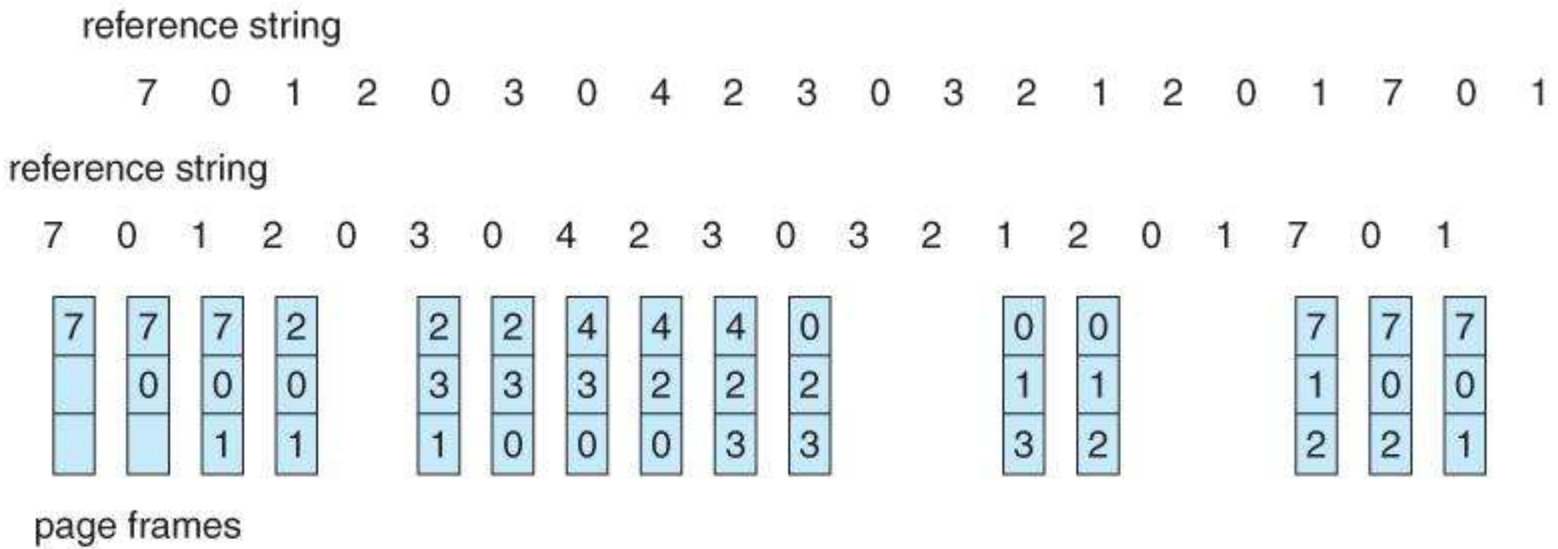


Page Replacement



Page replacement Algorithms

- **FIFO Page Replacement**



- ***Belady's anomaly***

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

- **Optimal Page Replacement**

  Replace the page that will not be used for the longest time in the future.

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

Reference string:

1 2 3 4 5 3 4 1 6 7 8 7 8 9 7 8 9 5 4 5 4 2

- **LRU Page Replacement**

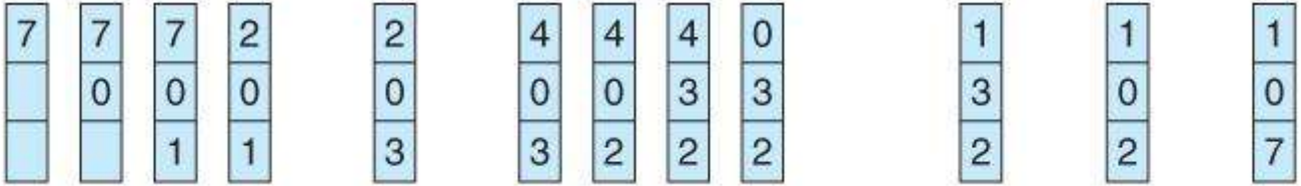
    The page that has not been used in the longest time

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

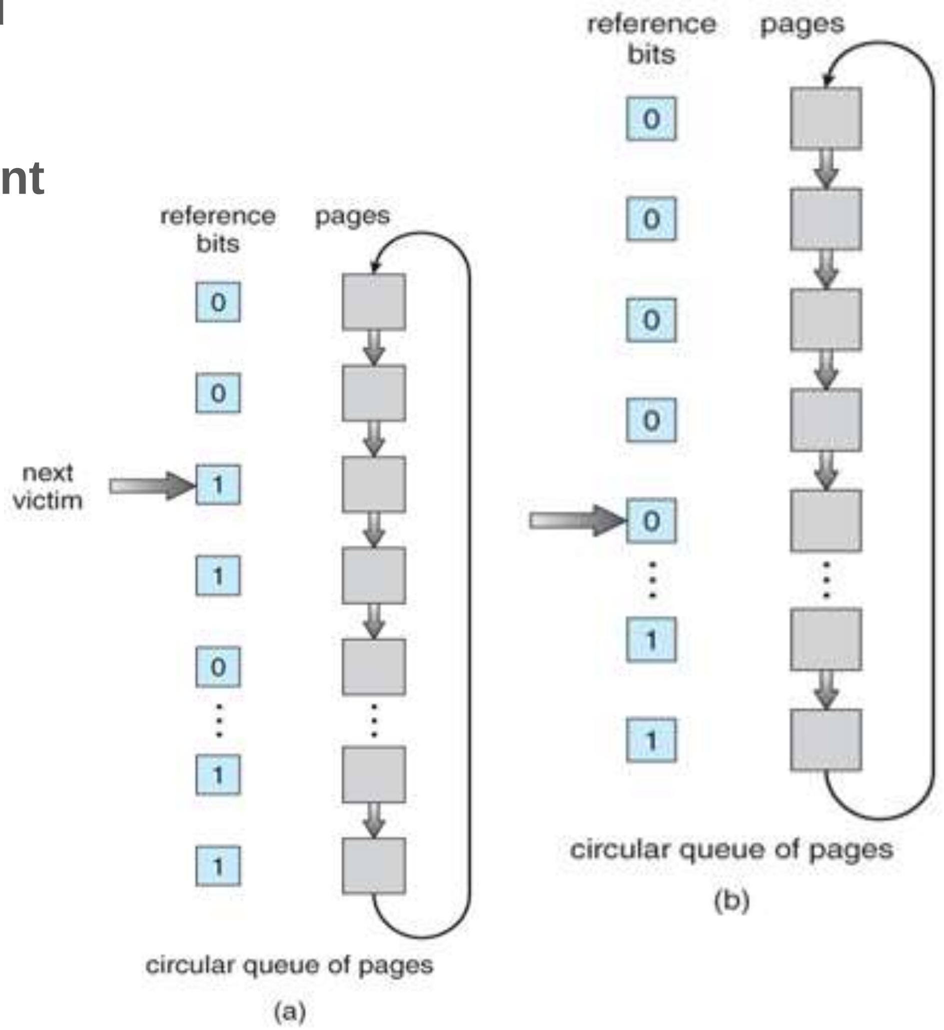


page frames

reference string

4 7 0 7 1 0 1 2 1 2 7 1 2

- Two simple approaches commonly used
  - **Counters**
  - **Stack**
- **LRU-Approximation Page Replacement** provide a ***reference bit*** for every entry in a page table
- **Additional-Reference-Bits Algorithm** most recent 8 reference bits for each page
- **Second-Chance Algorithm**



- **Enhanced Second-Chance Algorithm**

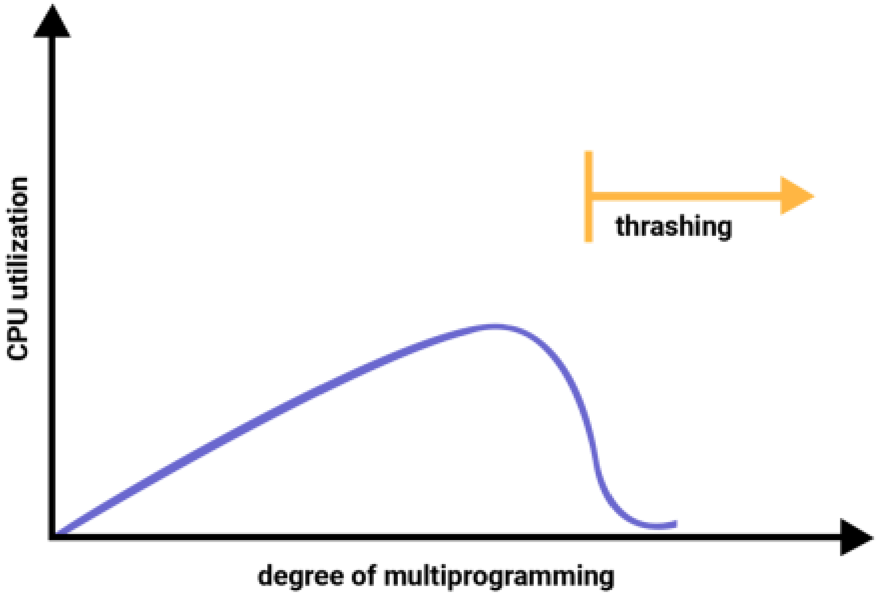
  the reference bit and the modify bit (dirty bit)
  - (0, 0) - Neither recently used nor modified.
  - (0, 1) - Not recently used, but modified.
  - (1, 0) - Recently used, but clean.
  - (1, 1) - Recently used and modified

- **Counting-Based Page Replacement :**
  - ***Least Frequently Used, LFU:***
  - ***Most Frequently Used, MFU:***
- **Page-Buffering Algorithms**
- Frames Allocation:
  - minimum number of free frames at all times
  - Equal Allocation
  - Proportional Allocation

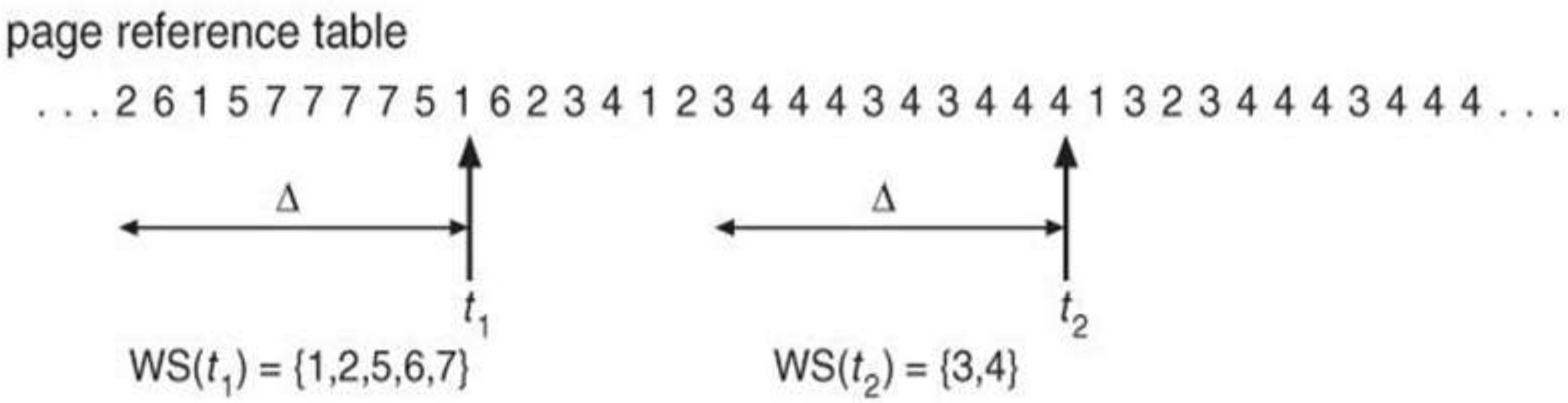
$$a_i = m * S_i / S$$

Thrashing

- A process is thrashing if it is spending more time paging than executing.
- limit the effects of thrashing by using a **local replacement algorithm** (or **priority replacement algorithm**).
- the **locality model of process execution.**



- **Working-Set Model:**
  - uses a parameter, Δ , to define the **working-set window.**
  - **The idea is to examine** the most recent Δ **page references.**
  - **The set of pages in the most recent Δ page** references is the **working set**



- **Page-Fault Frequency:**
  - upper and lower bounds on the desired page-fault rate
  - If the actual page-fault rate exceeds the upper limit, we allocate that process another frame;
  - if the page-fault rate falls below the lower limit, we remove a frame from that process.