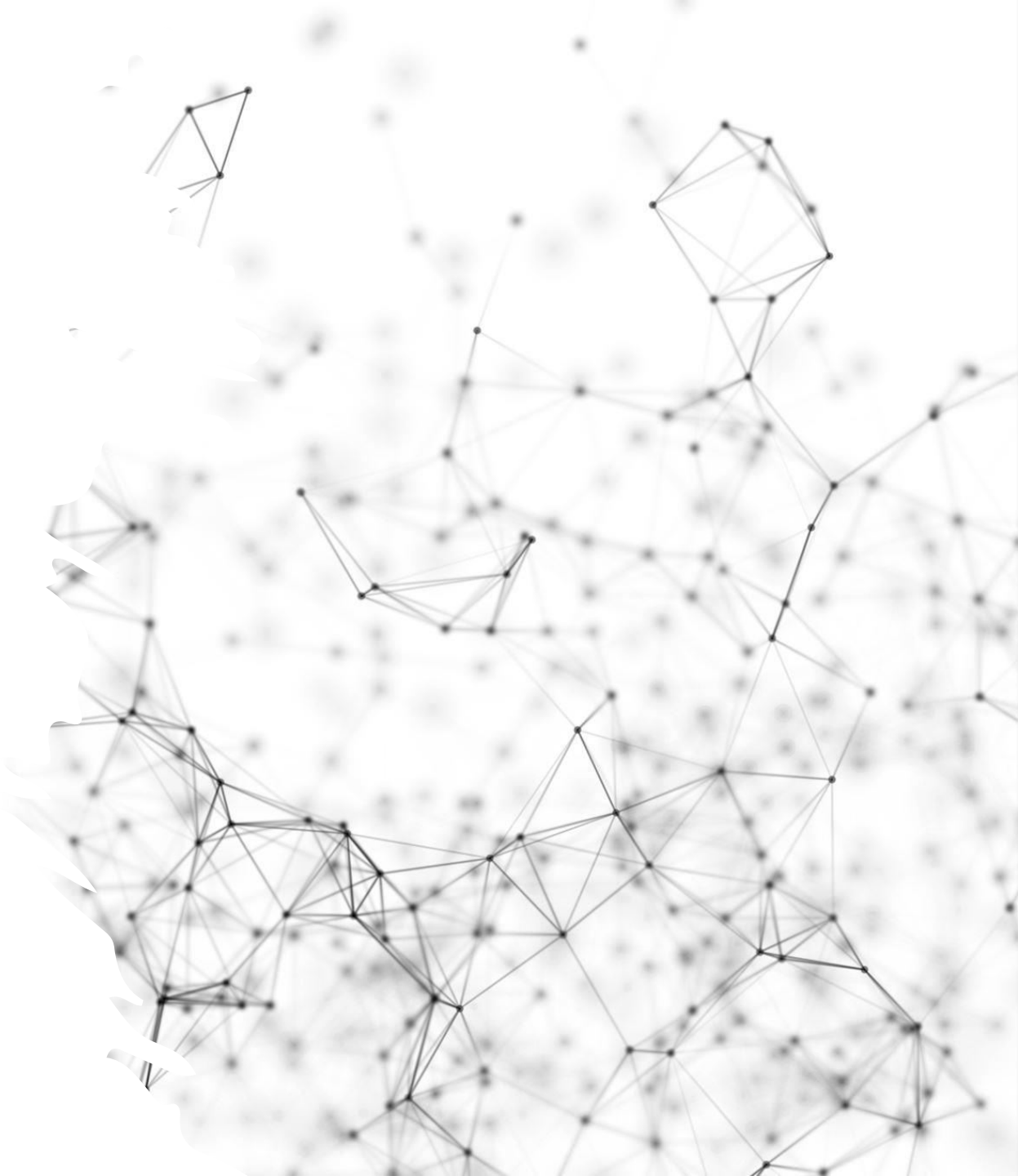
Virtual Memory



Virtual Memory

- A computer can address more memory than the amount physically installed on the system. This extra memory is actually called **virtual memory** and it is a section of a hard disk that's set up to emulate the computer's RAM.
- Virtual Memory is a space where large programs can store themselves in form of pages while their execution and only the required pages or portions of processes are loaded into the main memory. This technique is useful as large virtual memory is provided for user programs when a very small physical memory is there.

Virtual Memory

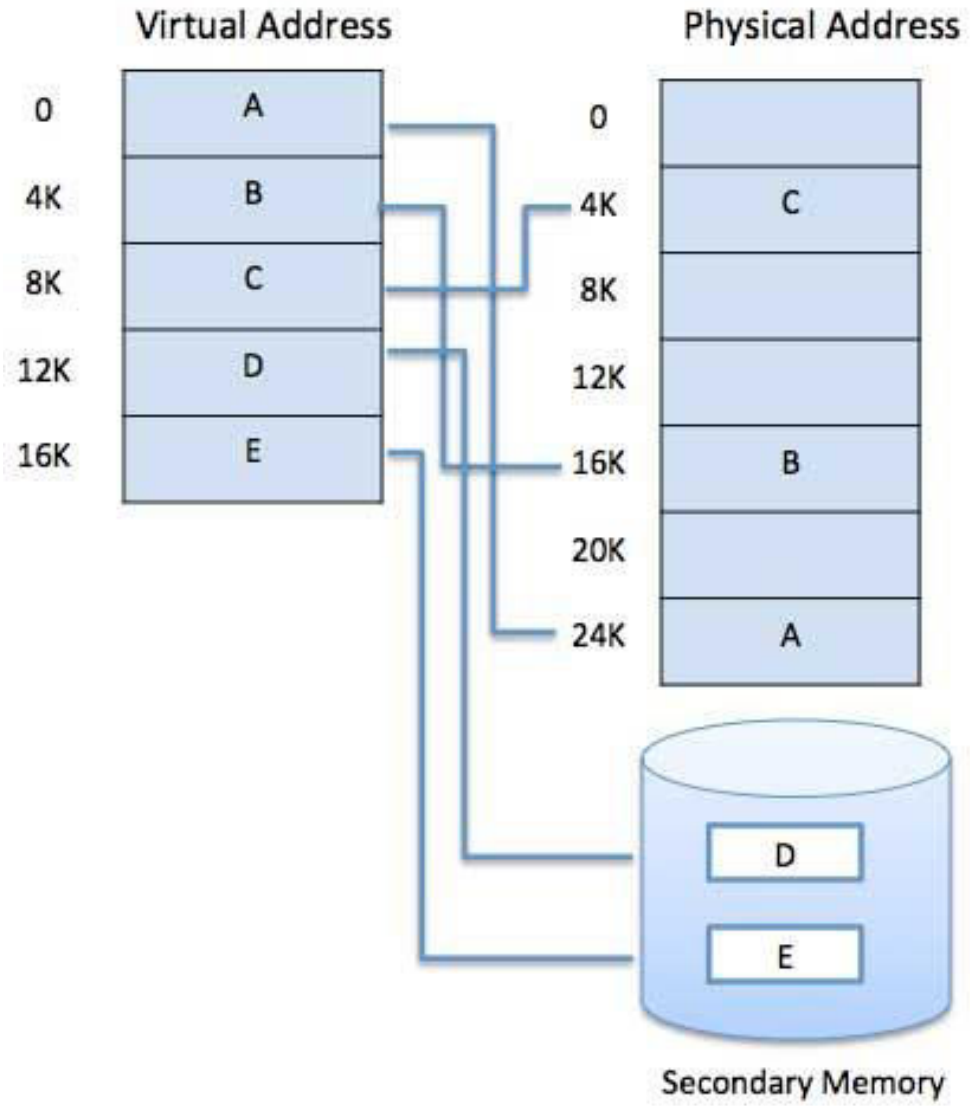
- The main visible advantage of this scheme is that programs can be larger than physical memory. Virtual memory serves two purposes. First, it allows us to extend the use of physical memory by using disk. Second, it allows us to have memory protection, because each virtual address is translated to a physical address.

Virtual Memory

- Following are the situations, when entire program is not required to be loaded fully in main memory.
  - User written error handling routines are used only when an error occurred in the data or computation.
  - Certain options and features of a program may be used rarely.
  - Many tables are assigned a fixed amount of address space even though only a small amount of the table is actually used.
  - The ability to execute a program that is only partially in memory would counter many benefits.
  - Less number of I/O would be needed to load or swap each user program into memory.

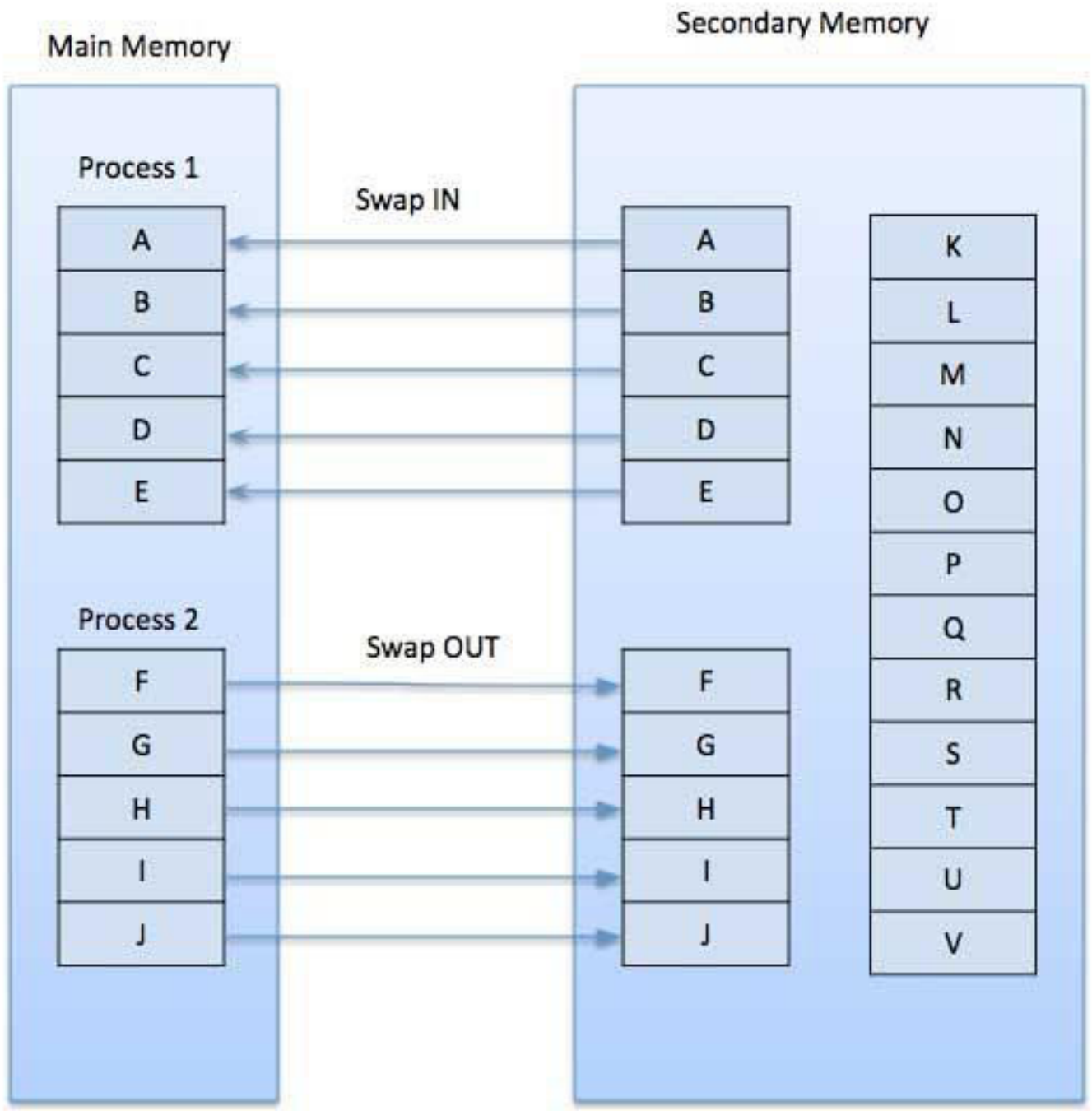
Virtual Memory

- Modern microprocessors intended for general-purpose use, a memory management unit, or MMU, is built into the hardware. The MMU's job is to translate virtual addresses into physical addresses.



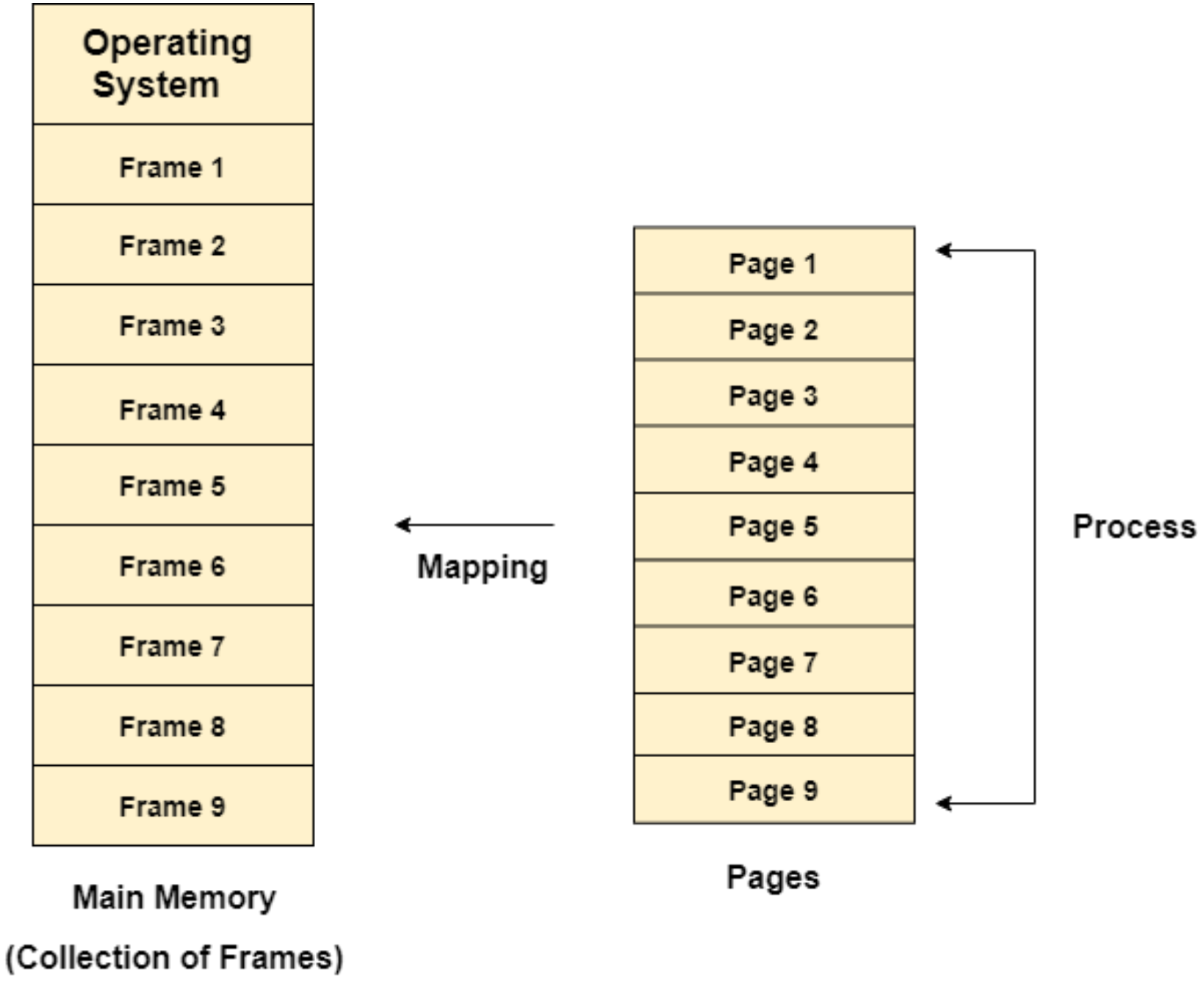
Demand Paging

- The basic idea behind demand paging is that 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 as lazy swapper, although a pager is a more accurate term.
- Initially only those pages are loaded which will be required the process immediately.



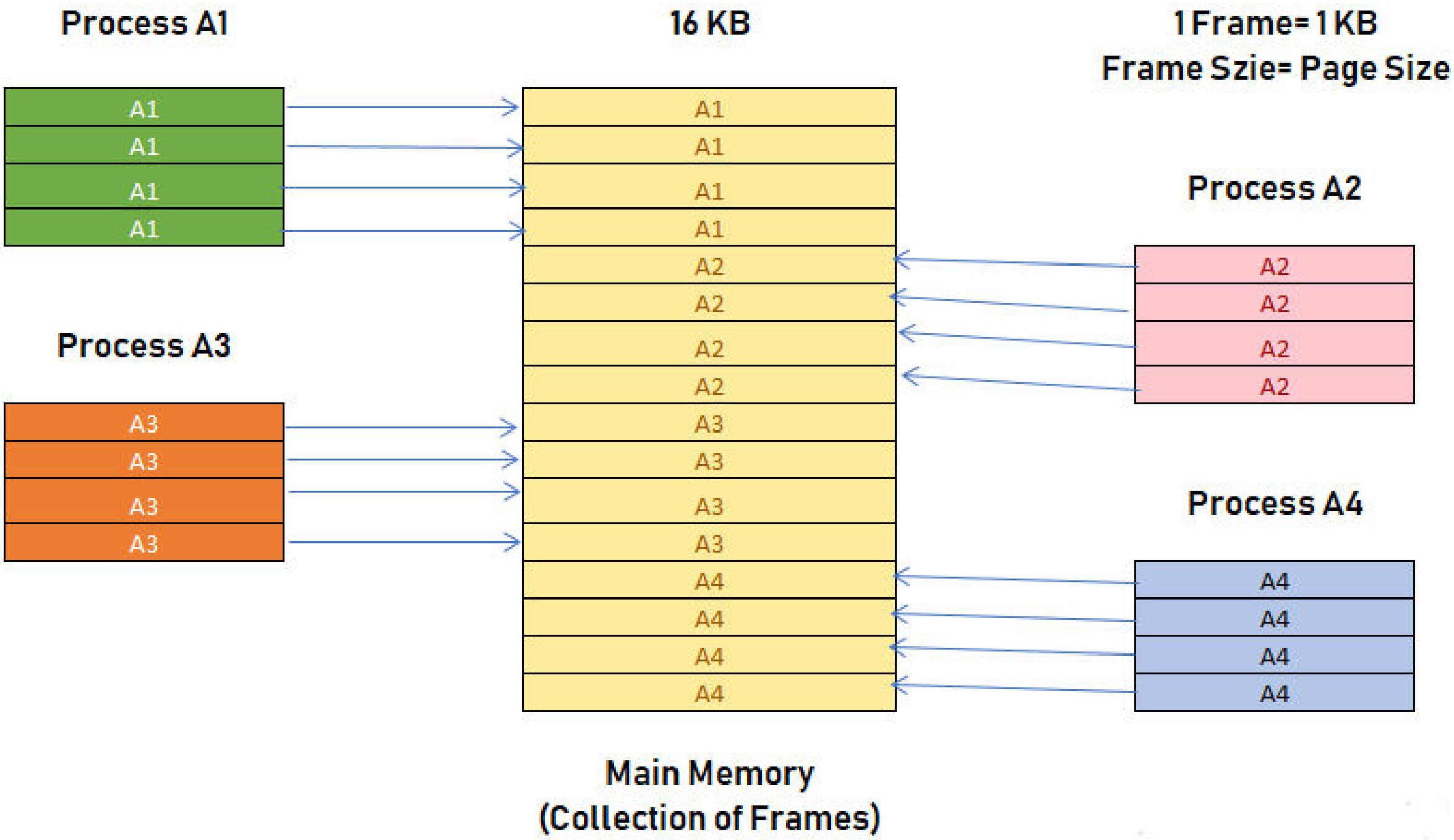
Paging

- **Paging** is a storage mechanism that allows OS to retrieve processes from the secondary storage into the main memory in the form of pages.
- The main idea behind the paging is to divide each process in the form of pages. The main memory will also be divided in the form of frames.
- One page of the process is to be stored in one of the frames of the memory. The pages can be stored at the different locations of the memory but the priority is always to find the contiguous frames or holes.
- Pages of the process are brought into the main memory only when they are required otherwise they reside in the secondary storage.
- Different operating system defines different frame sizes. The sizes of each frame must be equal. Considering the fact that the pages are mapped to the frames in Paging, page size needs to be as same as frame size.


Operating System
Frame 1
Frame 2
Frame 3
Frame 4
Frame 5
Frame 6
Frame 7
Frame 8
Frame 9
Main Memory
(Collection of Frames)
Mapping
Page 1
Page 2
Page 3
Page 4
Page 5
Page 6
Page 7
Page 8
Page 9
Pages
Process

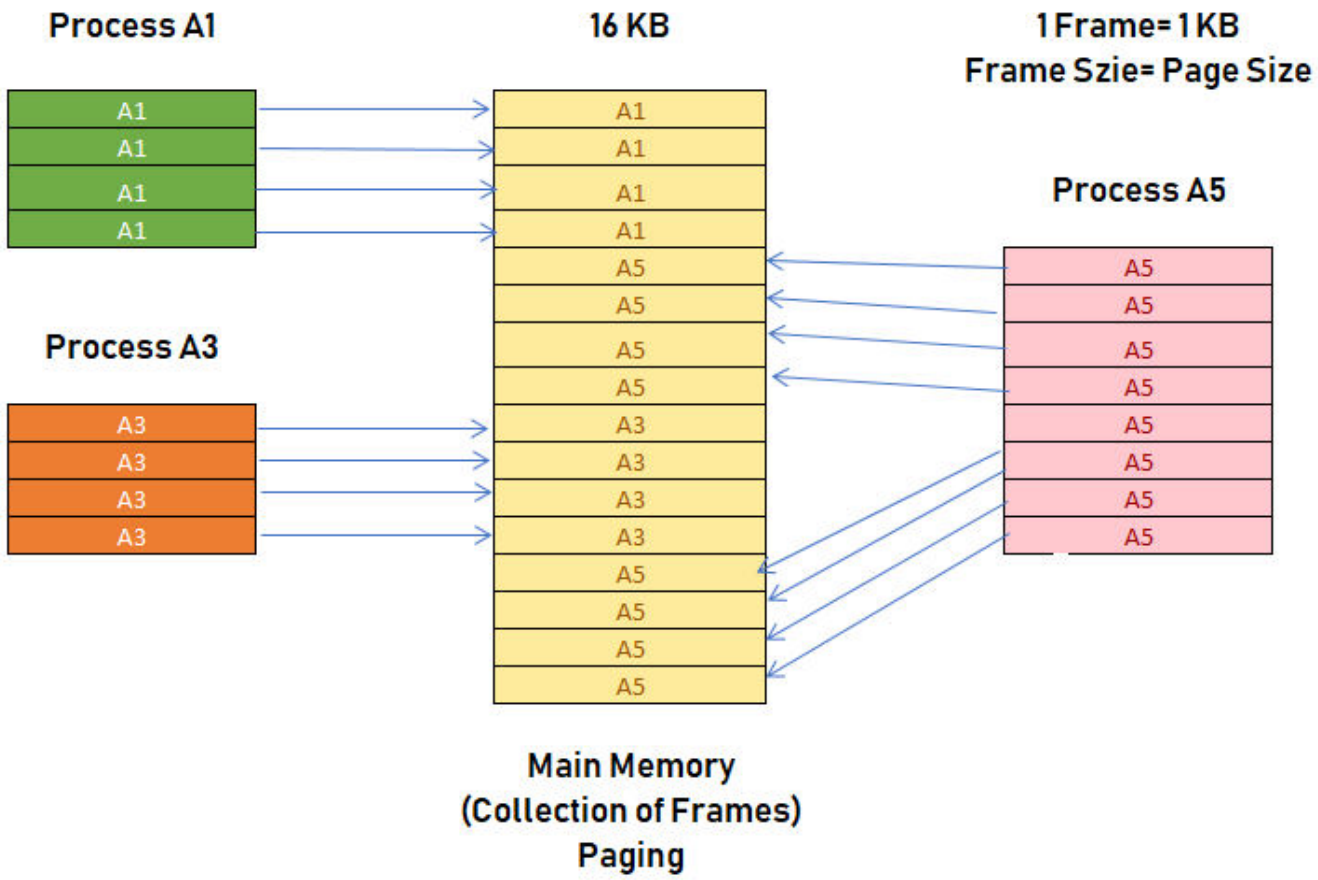
Example

- **For example, if the main memory size is 16 KB and Frame size is 1 KB. Here, the main memory will be divided into the collection of 16 frames of 1 KB each.**
- There are 4 separate processes in the system that is A1, A2, A3, and A4 of 4 KB each. Here, all the processes are divided into pages of 1 KB each so that operating system can store one page in one frame.
- At the beginning of the process, all the frames remain empty so that all the pages of the processes will get stored in a contiguous way.


Process A1
16 KB
1 Frame= 1 KB
Frame Szie= Page Size
A1
A1
A1
A1
A1
A1
A1
A1
Process A2
A2
A2
A2
A2
A2
A2
A2
A2
Process A3
A3
A3
A3
A3
A3
A3
A3
A3
Process A4
A4
A4
A4
A4
A4
A4
A4
A4
Main Memory
(Collection of Frames)

example

- In above example if the process A2 and A4 are moved to the waiting state after some time. Therefore, eight frames become empty, and so other pages can be loaded in that empty blocks. The process A5 of size 8 pages (8 KB) are waiting in the ready queue.



- The paging process should be protected by using the concept of insertion of an additional bit called Valid/Invalid bit. Paging Memory protection in paging is achieved by associating protection bits with each page. These bits are associated with each page table entry and specify protection on the corresponding page.

Paging

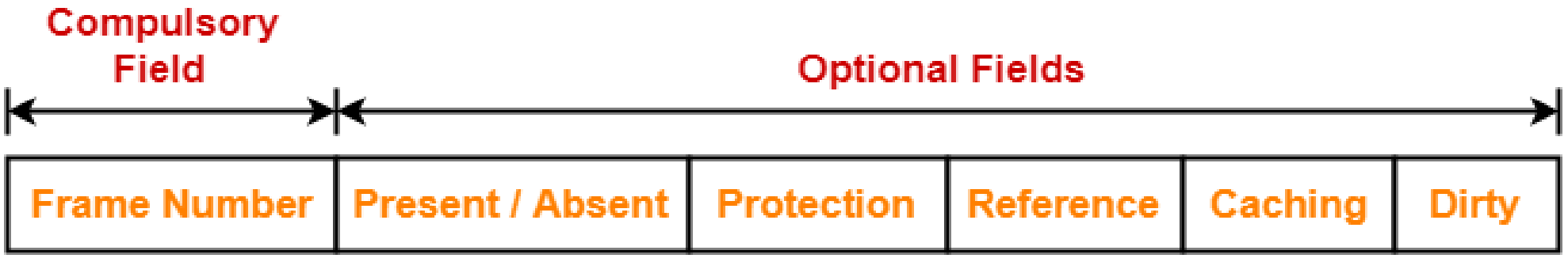
- **Advantages of Paging**
- Easy to use memory management algorithm
- No need for external Fragmentation
- Swapping is easy between equal-sized pages and page frames.
- **Disadvantages of Paging**
- May cause Internal fragmentation
- Complex memory management algorithm
- Page tables consume additional memory.
- Multi-level paging may lead to memory reference overhead.

Page Table

- Page Table is a data structure used by the virtual memory system to store the mapping between logical addresses and physical addresses.
- Page table is a data structure. It maps the page number referenced by the CPU to the frame number where that page is stored.
- Page table is stored in the main memory.
- Number of entries in a page table = Number of pages in which the process is divided.
- Each process has its own independent page table.

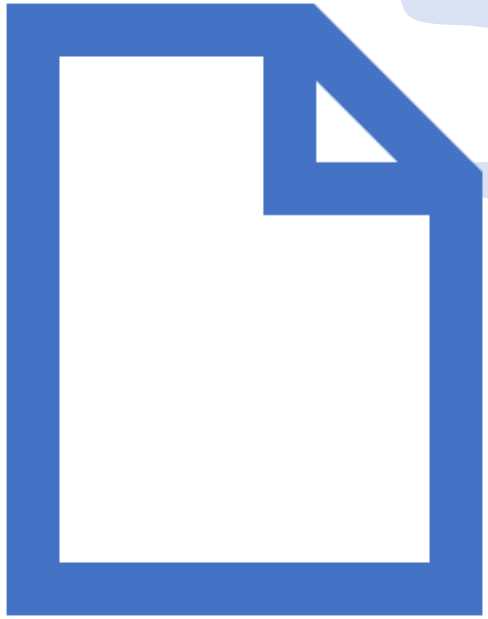
Page Table Entry

- A page table entry contains several information about the page.
- The information contained in the page table entry varies from operating system to operating system.
- The most important information in a page table entry is frame number.



Page Table Entry Format

Page Table Entry



- **Frame Number-**
- Frame number specifies the frame where the page is stored in the main memory.
- The number of bits in frame number depends on the number of frames in the main memory.
- **Present / Absent Bit-**
- This bit is also sometimes called as **valid / invalid bit**.
- This bit specifies whether that page is present in the main memory or not.
- If the page is not present in the main memory, then this bit is set to 0 otherwise set to 1.

Page Table Entry

- **Protection Bit**
- This bit is also sometimes called as "**Read / Write bit**".
- This bit is concerned with the page protection.
- It specifies the permission to perform read and write operation on the page.
- If only read operation is allowed to be performed and no writing is allowed, then this bit is set to 0.
- If both read and write operation are allowed to be performed, then this bit is set to 1.

Page Table Entry

- **Reference Bit**
- Reference bit specifies whether that page has been referenced in the last clock cycle or not.
- If the page has been referenced recently, then this bit is set to 1 otherwise set to 0.
- **Caching Enabled / Disabled-**
- This bit enables or disables the caching of page.
- Whenever freshness in the data is required, then caching is disabled using this bit.
- If caching of the page is disabled, then this bit is set to 1 otherwise set to 0.
- **Dirty Bit**
- This bit is also sometimes called as "**Modified bit**".i t specifies whether that page has been modified or not. If modified, then this bit is set to 1 otherwise set to 0.

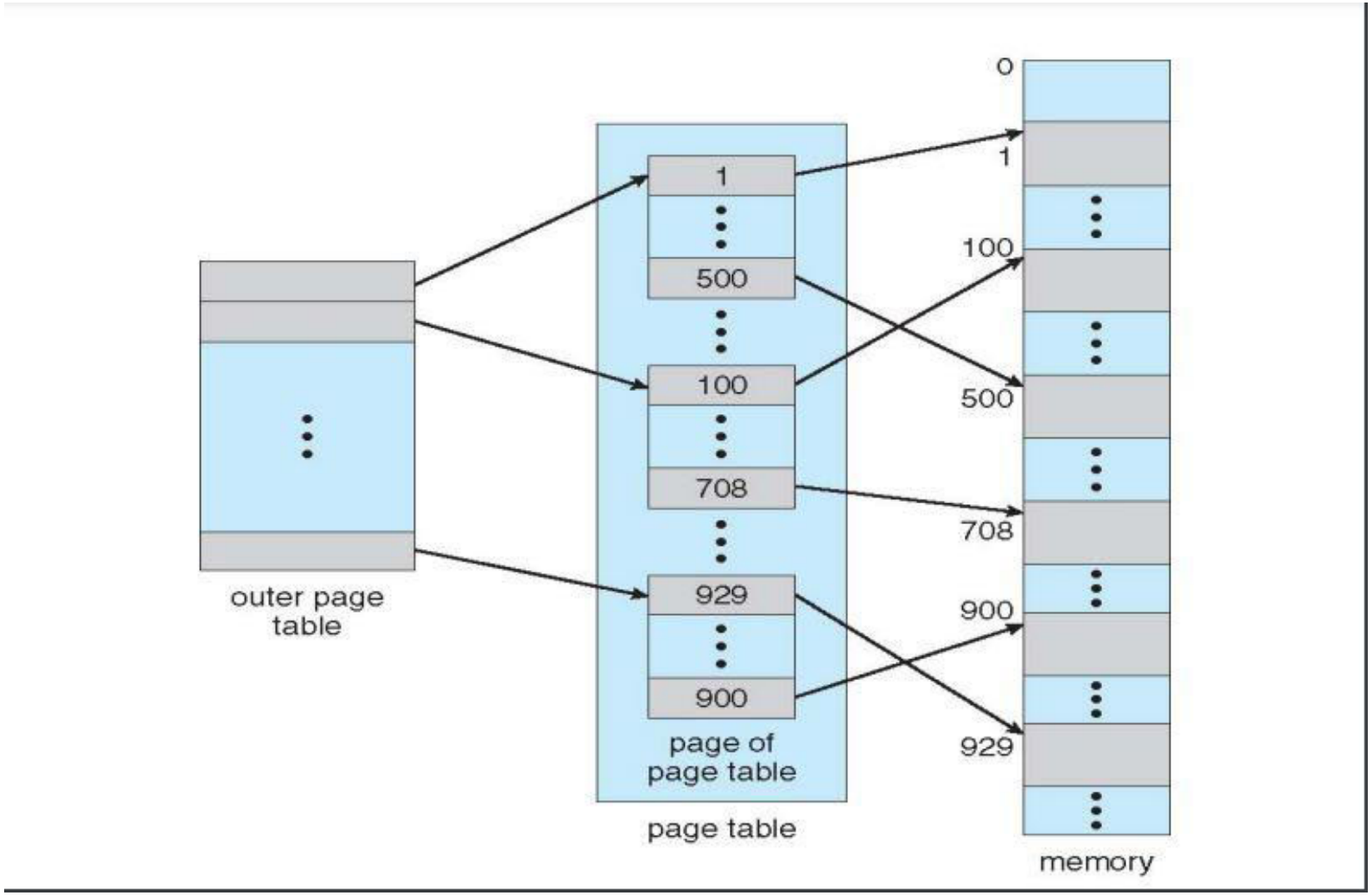
Types of Page Tables

- The major types of page tables are listed below :
  - Hierarchical page Table
  - Inverted Page Table
  - Shared Page Table

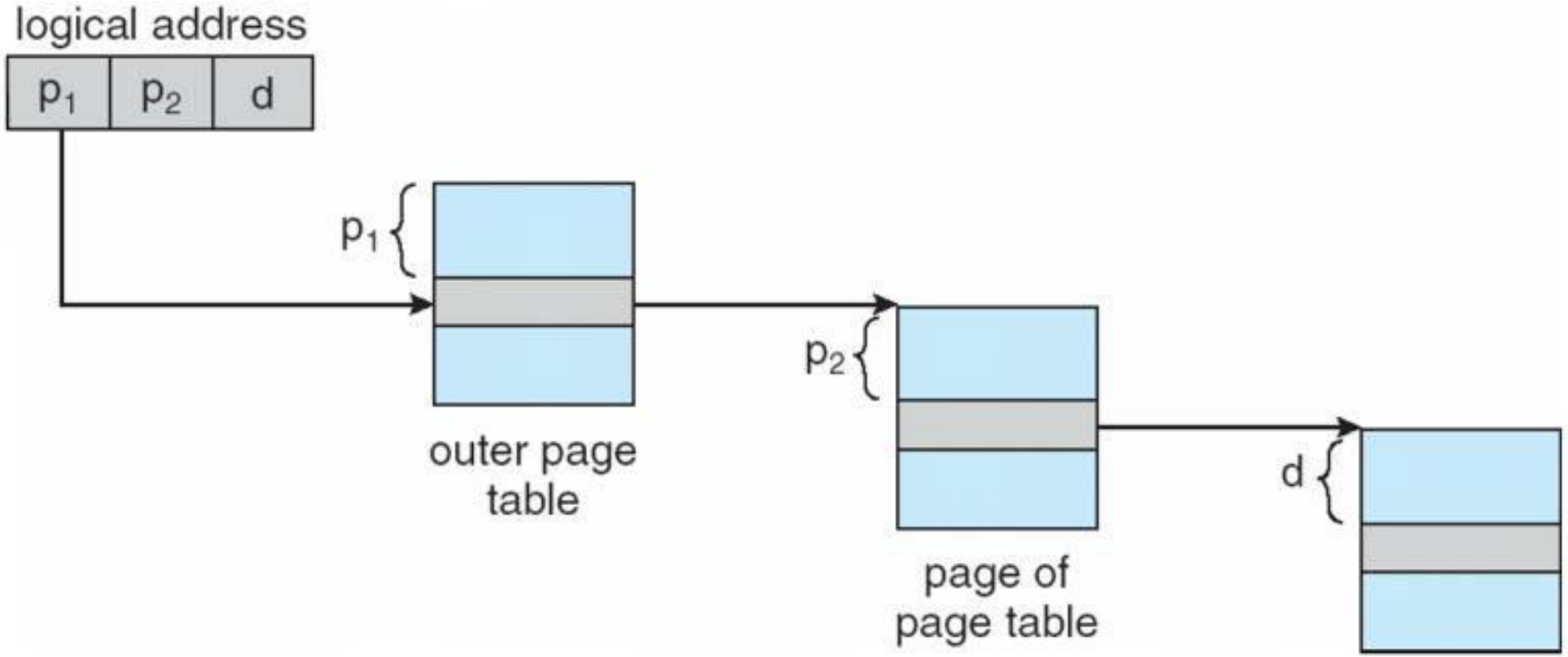
Hierarchical page Table

- hierarchical paging is a paging scheme which consist of two or more levels of page tables in a hierarchical manner. It is also known as **Multilevel Paging**.
- The entries of the level 1 page table are pointers to a level 2 page table and entries of the level 2 page tables are pointers to a level 3 page table and so on.
- The entries of the last level page table are stores actual frame information. Level 1 contain single page table and address of that table is stored in PTBR (Page Table Base Register).
- **Disadvantage:**
  Extra memory references to access address translation tables can slow programs down by a factor of two or more. Use translation look aside buffer (TLB) to speed up address translation by storing page table entries.

Hierarchical page Table

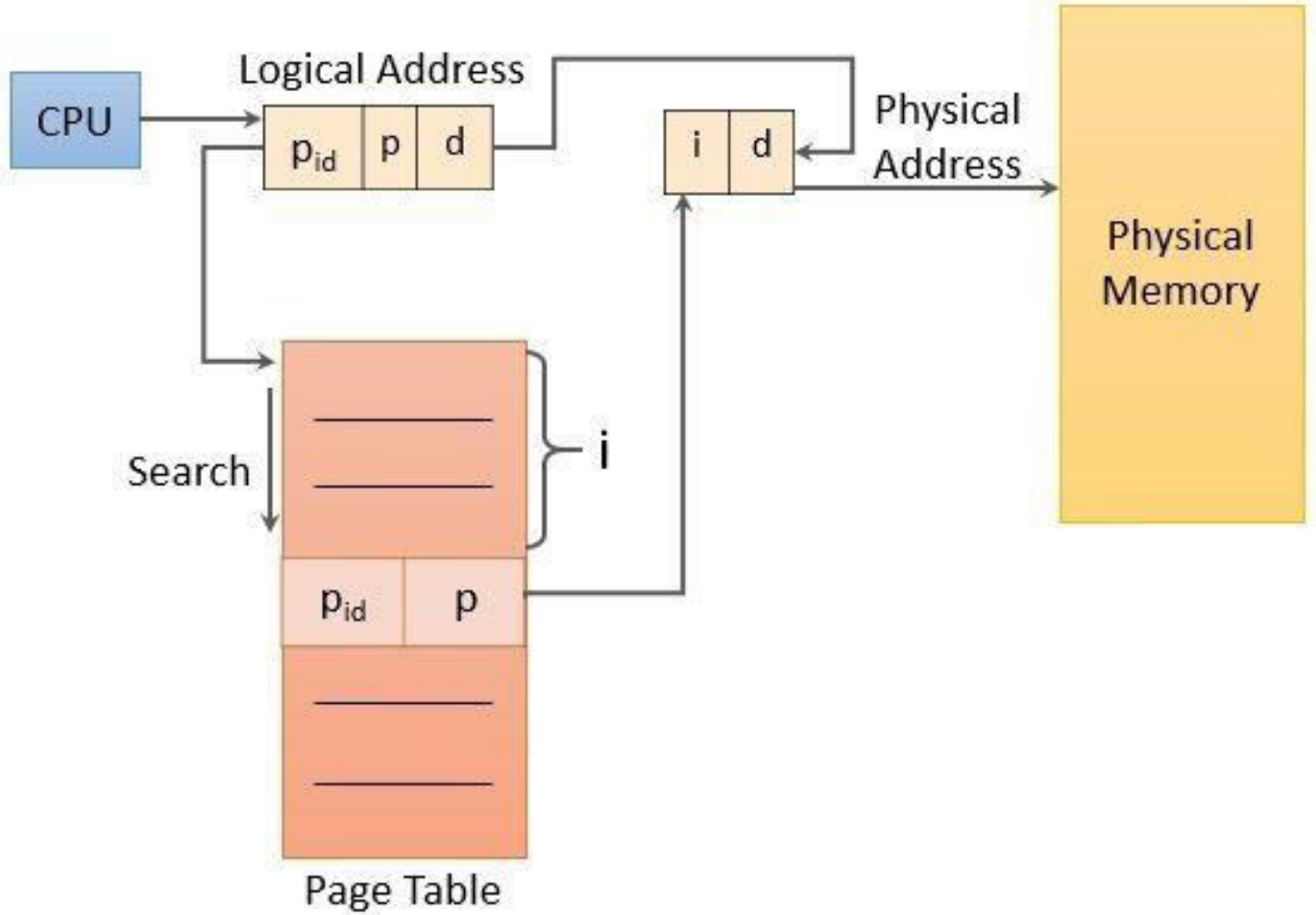


Hierarchical page Table



Inverted Page Table

- The concept of normal paging says that every process maintains its own page table which includes the entries of all the pages belonging to the process. The large process may have a page table with millions of entries. Such a page table consumes a large amount of memory.
- Consider we have six processes in execution. So, six processes will have some or the other of their page in the main memory which would compel their page tables also to be in the main memory consuming a lot of space. This is the drawback of the paging concept.
- The inverted page table is the solution to this wastage of memory.



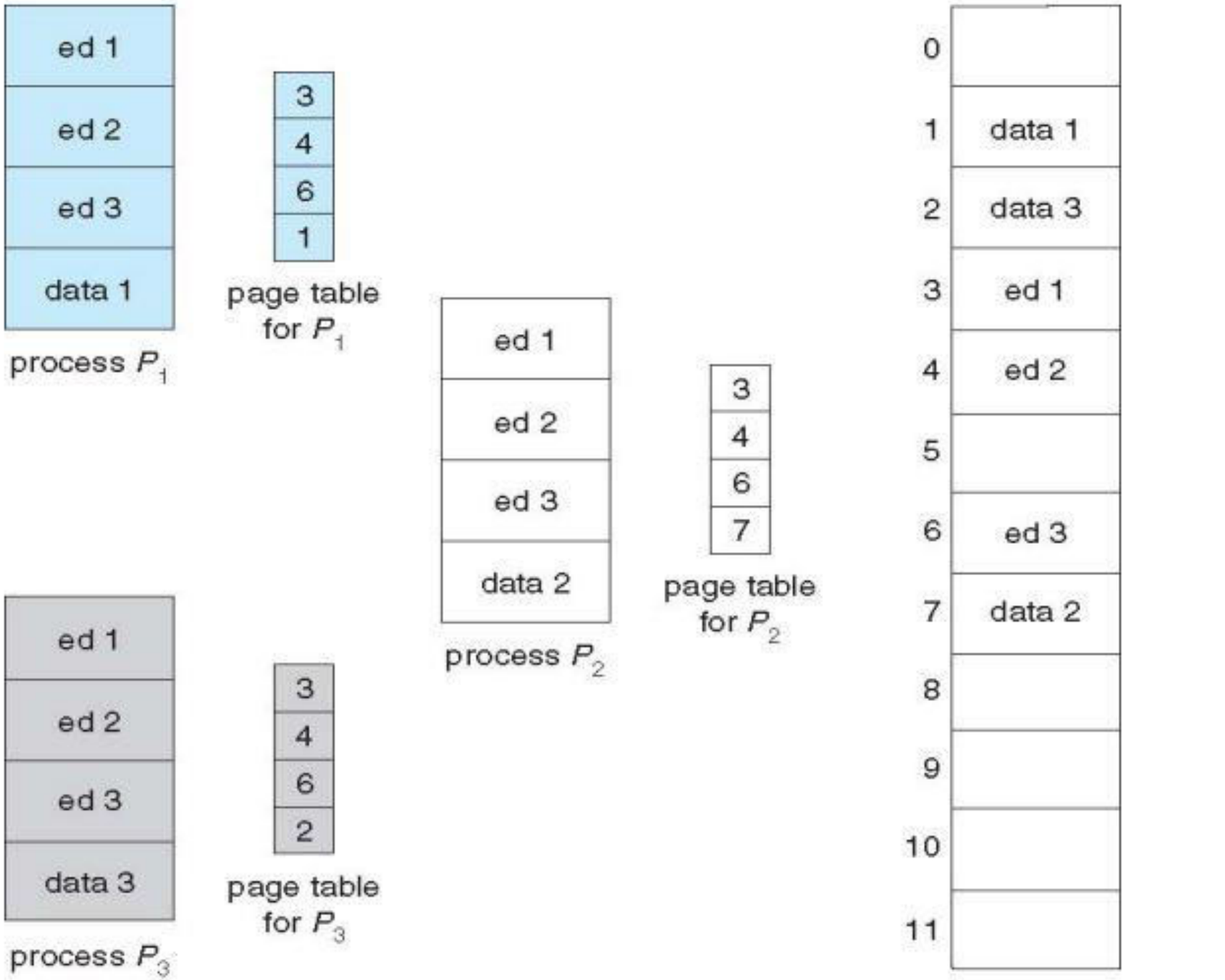
Structure of Inverted Page Table

Inverted Page Table

- inverted page table consist of single-page table which has entries of all the pages (may they belong to different processes) in main memory along with the information of the process to which they are associated.
- The CPU generates the logical address for the page it needs to access. This time the logical address consists of three entries process id, page number and the offset. The process id identifies the process, of which the page has been demanded, page number indicates which page of the process has been asked for and the offset value indicates the displacement required.

Shared Pages

- **Shared code**
  - One copy of read-only (**reentrant**) code shared among processes (i.e., text editors, compilers, window systems)
  - Similar to multiple threads sharing the same process space
  - Also useful for interprocess communication if sharing of read-write pages is allowed
- **Private code and data**
  - Each process keeps a separate copy of the code and data
  - The pages for the private code and data can appear anywhere in the logical address space


ed 1
ed 2
ed 3
data 1
process P_1
3
4
6
1
page table for P_1
ed 1
ed 2
ed 3
data 2
process P_2
3
4
6
7
page table for P_2
ed 1
ed 2
ed 3
data 3
process P_3
3
4
6
2
page table for P_3
0
1 data 1
2 data 3
3 ed 1
4 ed 2
5
6 ed 3
7 data 2
8
9
10
11

Shared Page Table

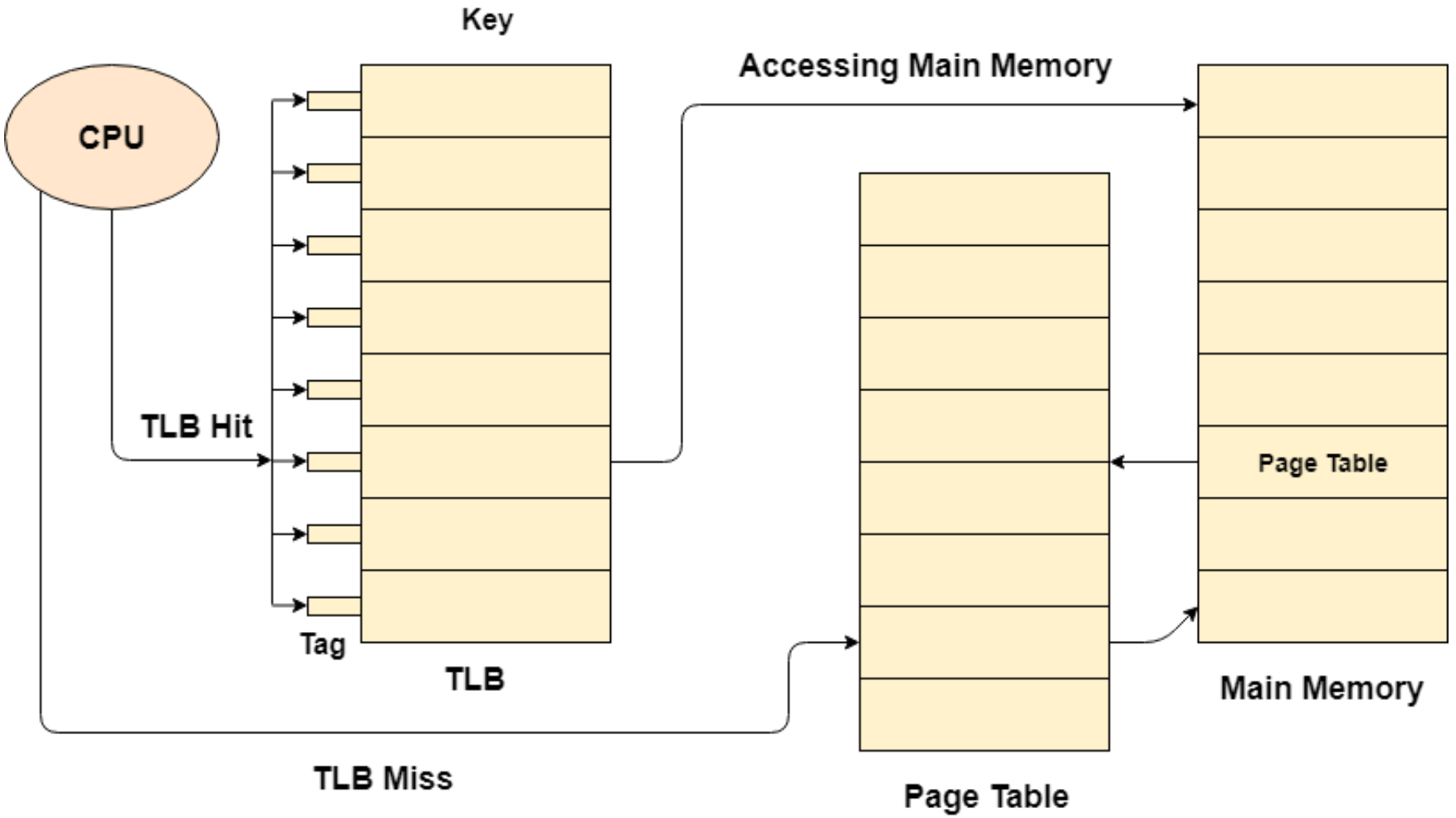
An advantage of paging is the possibility of sharing common code. This is important in a time sharing environment. If the code is reentrant code (or pure code); it can be shared.

Reentrant code is non-self modifying code; it never changes during execution. Thus, two or more processes can execute the same code at the same time. Each process has its own copy of registers and data storage to hold the data for the process's execution. The data for different processes will be different.

Translation look aside buffer (TLB)

- A Translation look aside buffer can be defined as a memory cache which can be used to reduce the time taken to access the page table again and again.
- It is a memory cache which is closer to the CPU and the time taken by CPU to access TLB is lesser then that taken to access main memory.
- In other words, we can say that TLB is faster and smaller than the main memory but cheaper and bigger than the register.
- TLB follows the concept of locality of reference which means that it contains only the entries of those many pages that are frequently accessed by the CPU.

Translation look aside buffer (TLB)

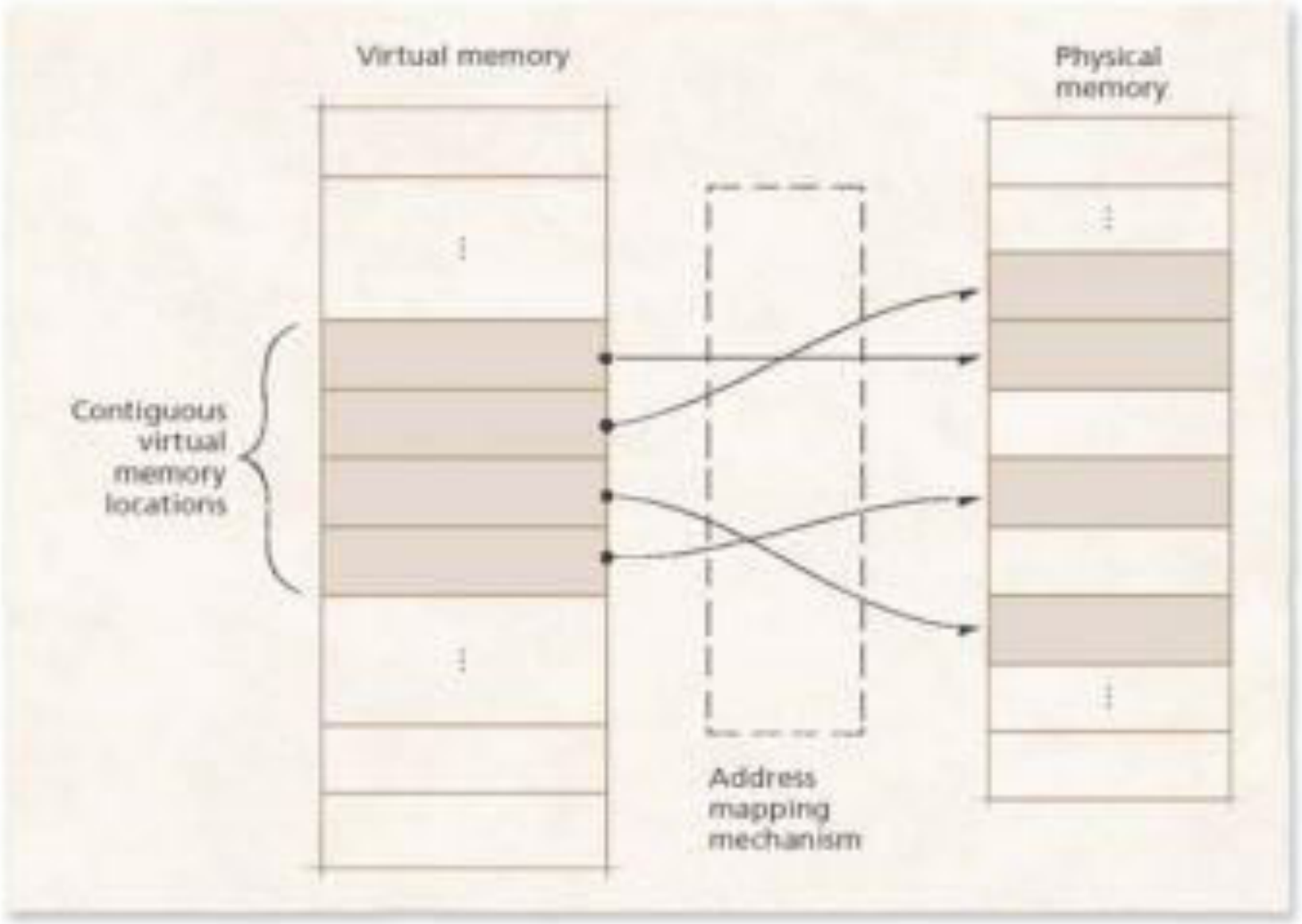


Translation look aside buffer (TLB)

- In translation look aside buffers, there are tags and keys with the help of which, the mapping is done.
- TLB hit is a condition where the desired entry is found in translation look aside buffer. If this happens then the CPU simply access the actual location in the main memory.
- However, if the entry is not found in TLB (TLB miss) then CPU has to access page table in the main memory and then access the actual frame in the main memory.
- Therefore, in the case of TLB hit, the effective access time will be lesser as compare to the case of TLB miss.

10.3 Block Mapping

Figure 10.5 Artificial contiguity.



- Pages
  - Blocks are fixed size
  - Technique is called paging
- Segments
  - Blocks maybe of different size
  - Technique is called segmentation
- Block mapping
  - System represents addresses as ordered pairs

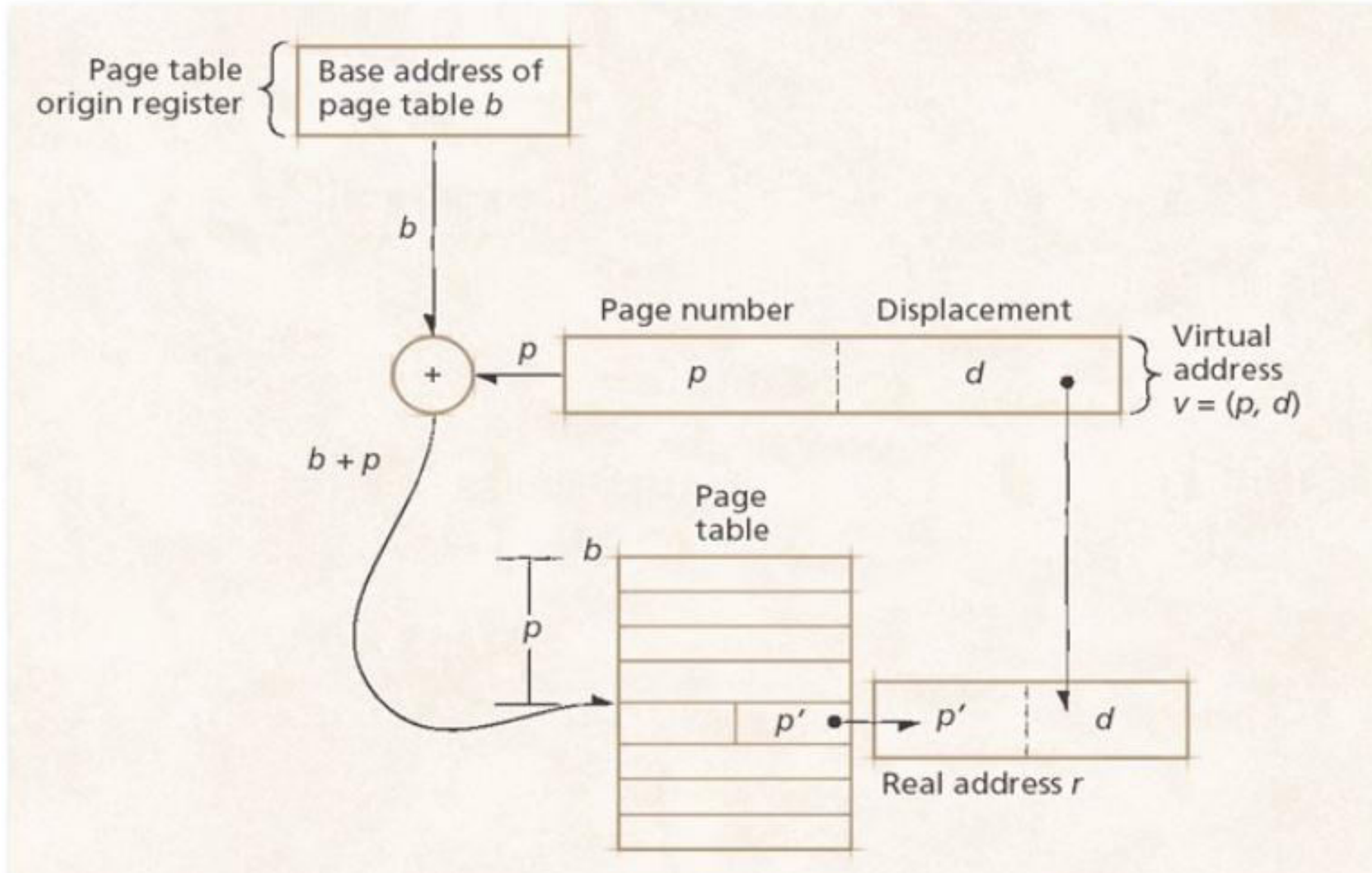
Figure 10.6 Virtual address format in a block mapping system.

- **Page offset(d):**
- Number of bits required to represent particular word in a page or page size of Logical Address Space or word number of a page or page offset.

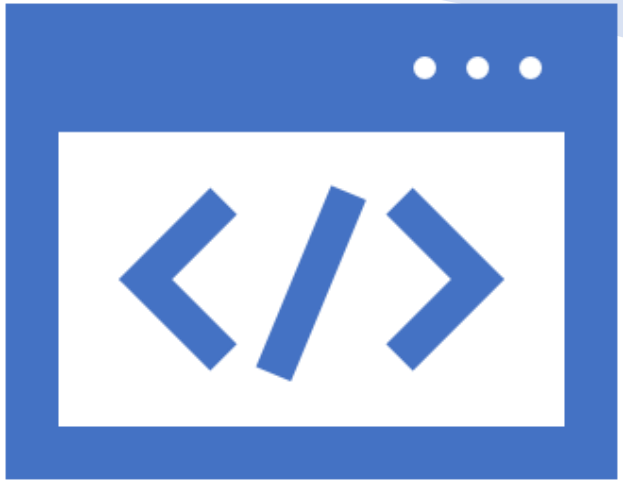
- Direct mapping
  - Dynamic address translation under paging is similar to block address translation
  - Process references virtual address $v = (p, d)$
    - DAT adds the process's page table base address, b, to referenced page number, p
    - $b + p$ forms the main memory address of the PTE for page p
    - System concatenates p' with displacement, d, to form real address, r

10.4.1 Paging Address Translation by Direct Mapping

Figure 10.12 Paging address translation by direct mapping.



page fault



- While executing a program, if the program references a page which is not available in the main memory because it was swapped out a little ago, the processor treats this invalid memory reference as a **page fault** and transfers control from the program to the operating system to demand the page back into the memory.

Page Replacement Algorithms

- The page replacement algorithm decides which memory page is to be replaced. The process of replacement is sometimes called swap out or write to disk. Page replacement is done when the requested page is not found in the main memory (page fault).
- When the page that was selected for replacement and was paged out, is referenced again, it has to read in from disk, and this requires for I/O completion. This process determines the quality of the page replacement algorithm: the lesser the time waiting for page-ins, the better is the algorithm.

Reference String

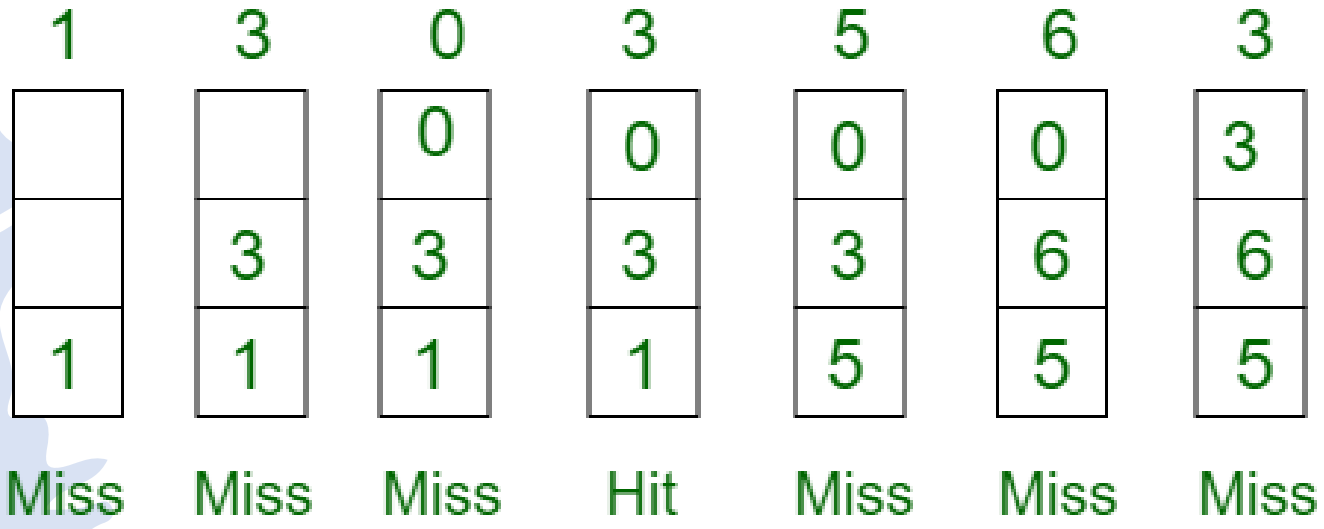
- The string of memory references is called reference string. Reference strings are generated artificially or by tracing a given system and recording the address of each memory reference.
  - For a given page size, we need to consider only the page number, not the entire address.
  - If we have a reference to a page **p**, then any immediately following references to page **p** will never cause a page fault. Page p will be in memory after the first reference; the immediately following references will not fault.
  - For example, consider the following sequence of addresses − 123,215,600,1234,76,96
  - If page size is 100, then the reference string is 1,2,6,12,0,0

First In First Out (FIFO)

- This is the simplest page replacement algorithm. In this algorithm, the operating system keeps track of all pages in the memory in a queue, the oldest page is in the front of the queue. When a page needs to be replaced page in the front of the queue is selected for removal.
- **Example-1** Consider page reference string 1, 3, 0, 3, 5, 6 with 3 page frames. Find number of page faults.

First In First Out (FIFO)

Page reference 1, 3, 0, 3, 5, 6, 3



Total Page Fault = 6

First In First Out (FIFO)

- Initially all slots are empty, so when 1, 3, 0 came they are allocated to the empty slots —> **3 Page Faults.**
  when 3 comes, it is already in memory so **—> 0 Page Faults.**
  Then 5 comes, it is not available in memory so it replaces the oldest page slot **i.e** 1. **—>1 Page Fault.**
  6 comes, it is also not available in memory so it replaces the oldest page slot **i.e** 3 **—>1 Page Fault.**
  Finally when 3 come it is not available so it replaces 0 **1 page fault**

Belady's anomaly

- Belady's anomaly proves that it is possible to have more page faults when increasing the number of page frames while using the First in First Out (FIFO) page replacement algorithm. For example, if we consider reference string 3, 2, 1, 0, 3, 2, 4, 3, 2, 1, 0, 4 and 3 slots, we get 9 total page faults, but if we increase slots to 4, we get 10 page faults.

First In First Out (FIFO)

- ***Advantages***
  - Simple and easy to implement.
  - Low overhead.
- ***Disadvantages***
  - Poor performance.
  - Doesn't consider the frequency of use or last used time, simply replaces the oldest page.
  - Suffers from Belady's Anomaly(i.e. more page faults when we increase the number of page frames).

Optimal Page replacement

In this algorithm, pages are replaced which would not be used for the longest duration of time in the future.

Example-2:*Consider the page references 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, with 4 page frame. Find number of page fault.*

Optimal Page replacement

Page reference 7,0,1,2,0,3,0,4,2,3,0,3,2,3

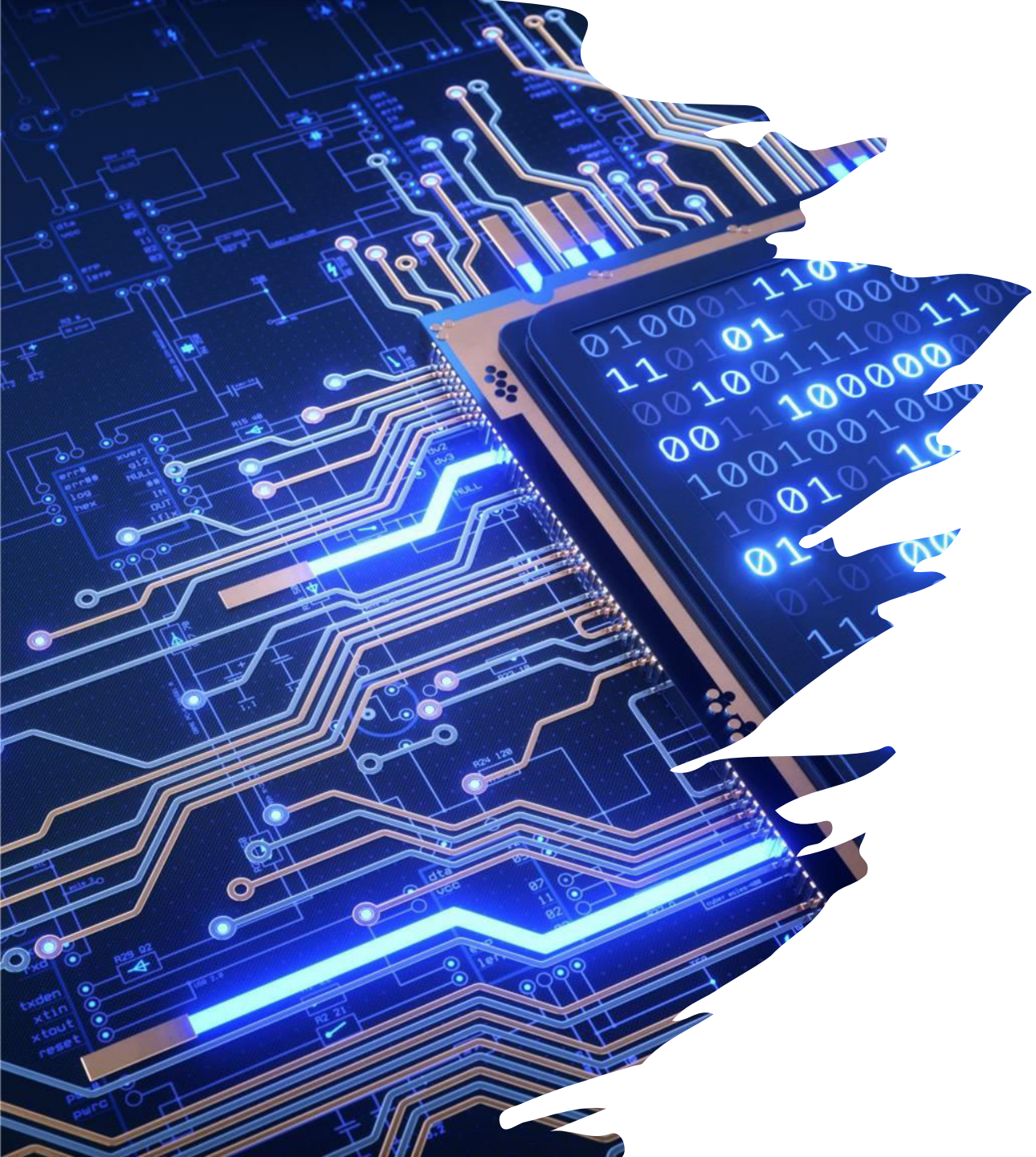
No. of Page frame - 4

Total Page Fault = 6

Optimal Page replacement

- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
  0 is already there so —> 0 Page fault.
  when 3 came it will take the place of 7 because it is not used for the longest duration of time in the future.—>1 Page fault.
  0 is already there so —> 0 Page fault..
  4 will takes place of 1 —> **1 Page Fault.**

  Now for the further page reference string —> 0 Page fault because they are already available in the memory.
- Optimal page replacement is perfect, but not possible in practice as the operating system cannot know future requests. The use of Optimal Page replacement is to set up a benchmark so that other replacement algorithms can be analyzed against it.

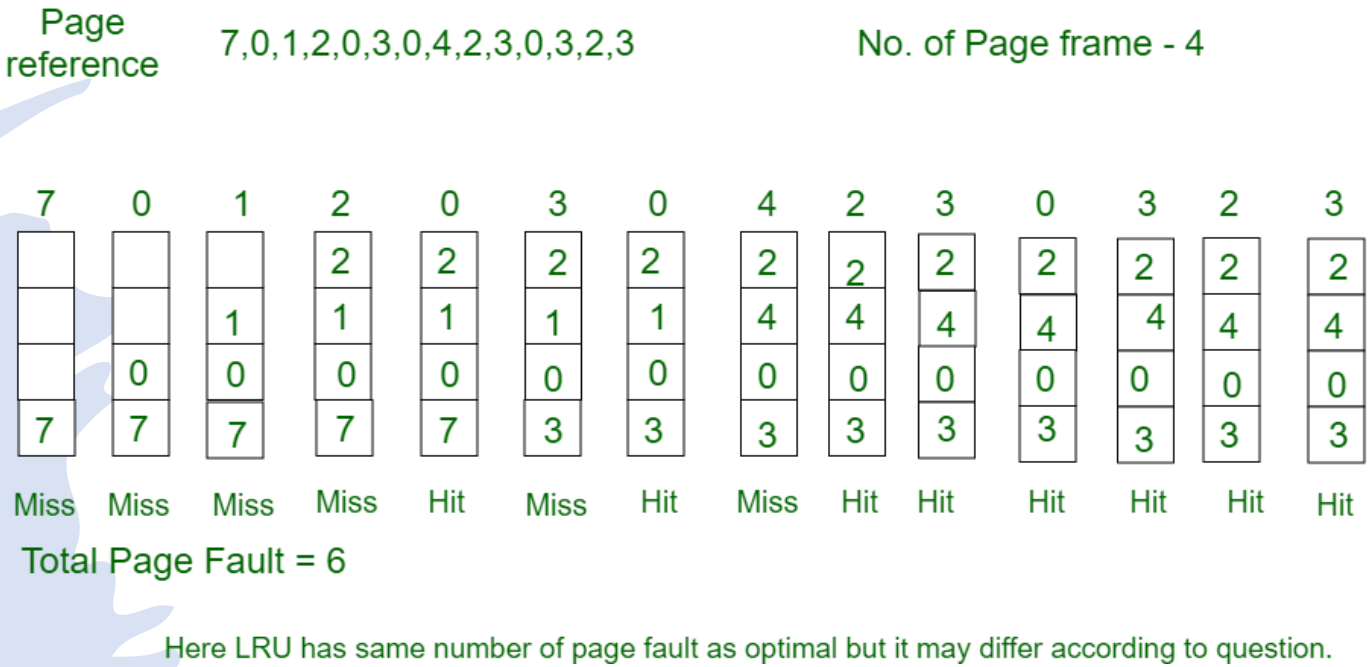


Optimal Page replacement

- ***Advantages***
  - Easy to Implement.
  - Simple data structures are used.
  - Highly efficient.
- ***Disadvantages***
  - Requires future knowledge of the program.
  - Time-consuming.

Least Recently Used

- In this algorithm page will be replaced which is least recently used.
- **Example-3**Consider the page reference string 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 3 with 4 page frames.Find number of page faults.



Here LRU has same number of page fault as optimal but it may differ according to question.

Least Recently Used

- Initially all slots are empty, so when 7 0 1 2 are allocated to the empty slots —> **4 Page faults**
- 0 is already their so —> **0 Page fault.**
- when 3 came it will take the place of 7 because it is least recently used —>**1 Page fault**
- 0 is already in memory so —> **0 Page fault**.
- 4 will takes place of 1 —> **1 Page Fault**
  Now for the further page reference string —> **0 Page fault**
- because they are already available in the memory.

Least Recently Used

- ***Advantages***
- Efficient.
- Doesn't suffer from Belady's Anomaly.
- ***Disadvantages***
- Complex Implementation.
- Expensive.
- Requires hardware support

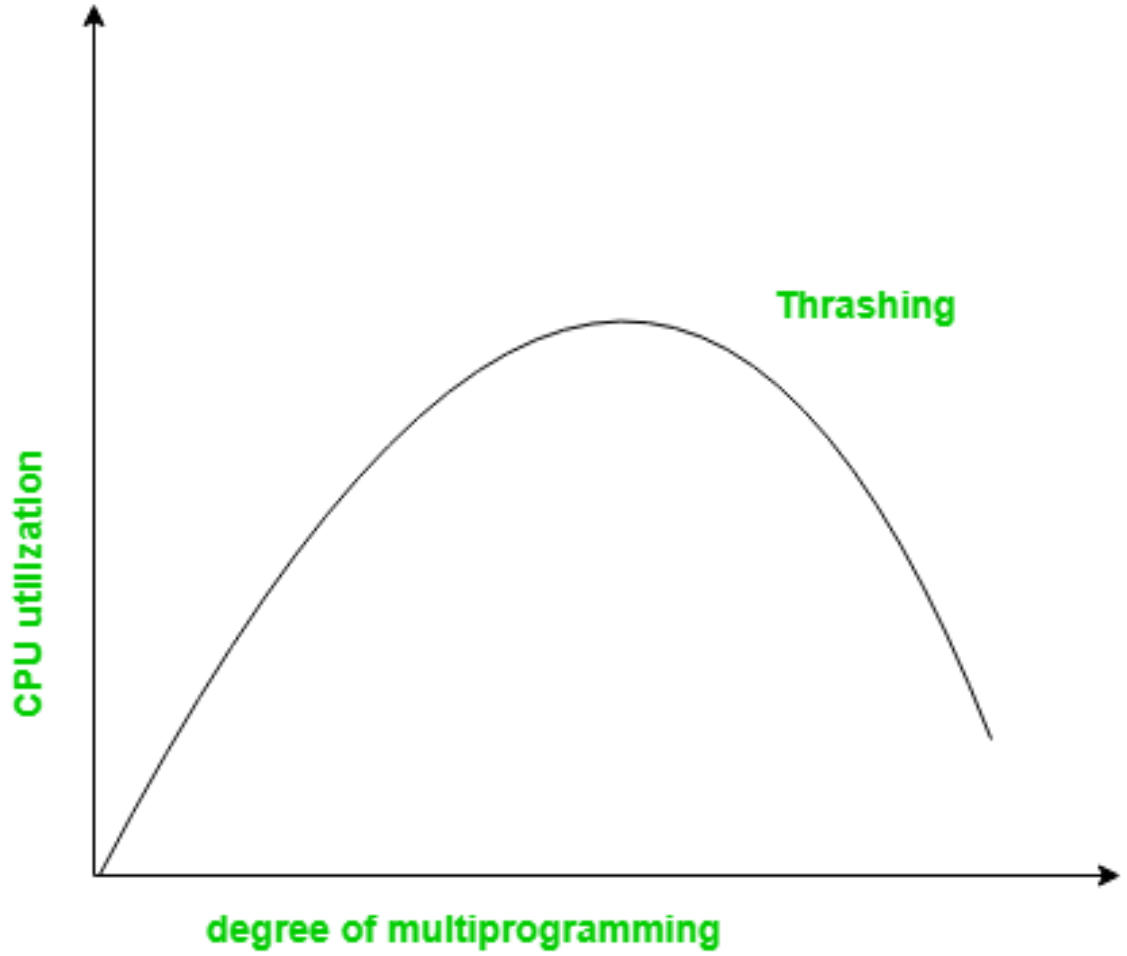
The Second Chance Page Replacement Algorithm

- A simple modification to FIFO that avoids the problem of throwing out a heavily used page is to inspect the R bit of the oldest page.
- If it is 0, the page is both old and unused, so it is replaced immediately. If the R bit is 1, the bit is cleared, the page is put onto the end of the list of pages, and its load
- time is updated as though it had just arrived in memory. Then the search continues.
- The operation of this algorithm is called second chance.

What second chance is doing is looking for an old page that has not been referenced in the previous clock interval. If all the pages have been referenced, second chance degenerates into pure FIFO.

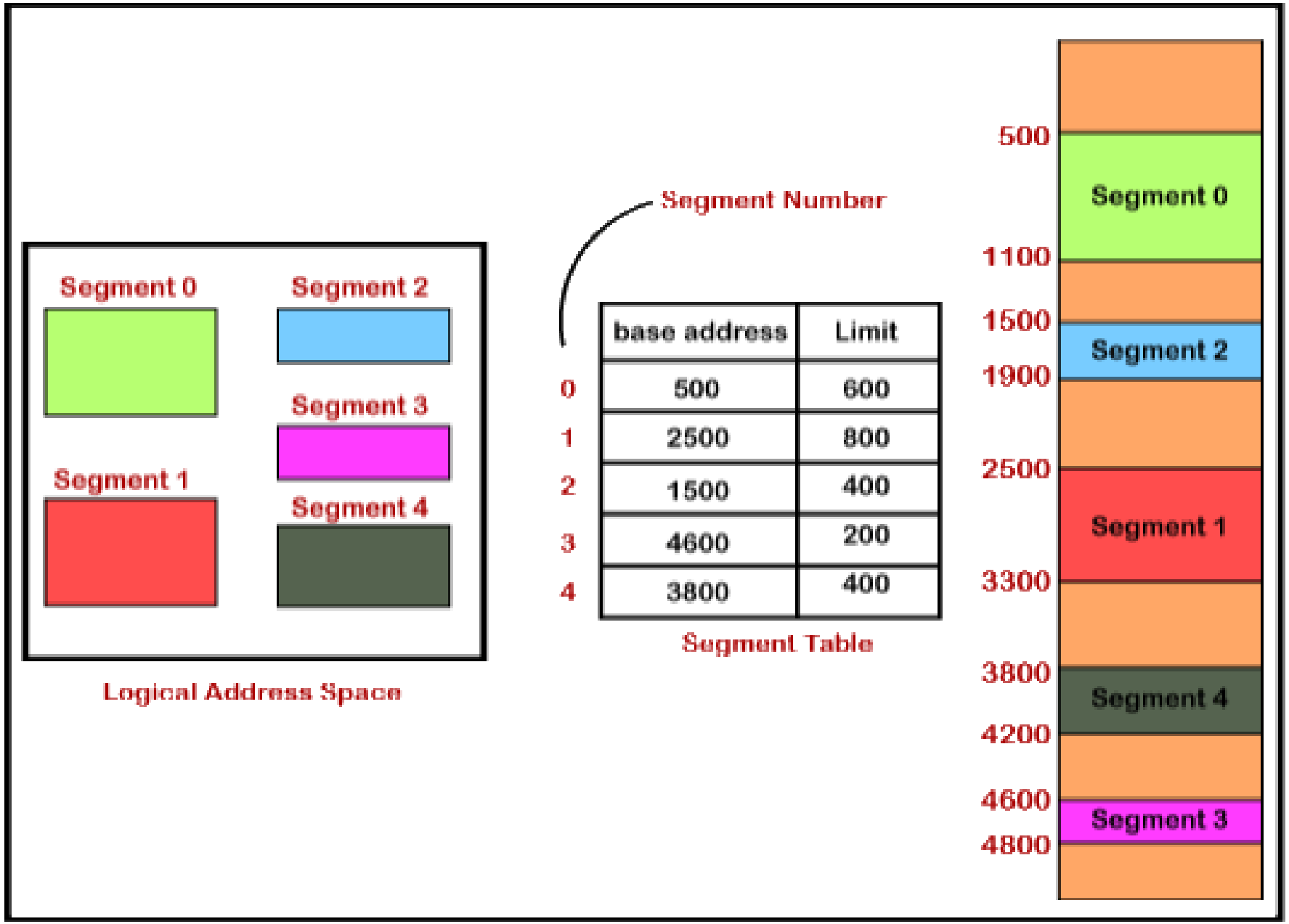
What is Thrashing in OS?

- In case, if the page fault and swapping happens very frequently at a higher rate, then the operating system has to spend more time swapping these pages. This state in the operating system is termed thrashing. Because of thrashing the CPU utilization is going to be reduced.



Segmentation

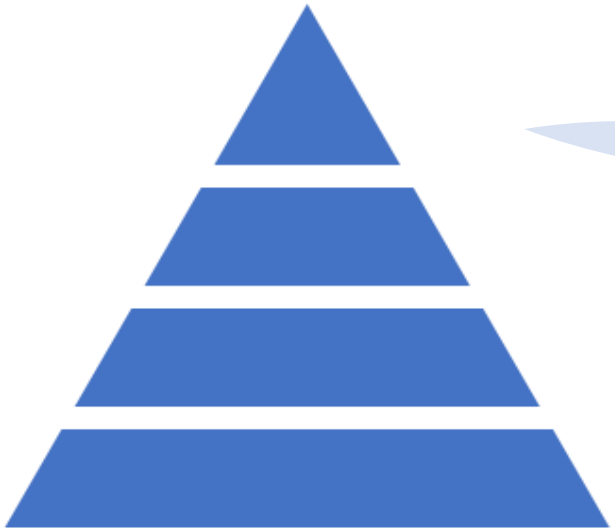
- Segmentation is a technique of memory management.in the form of segments. They are of variable length but, in Paging, the pages are of fixed size.
- In segmentation,memory is split into variable-length parts. Each part is known as segments. The information which is related to the segment is stored in a table which is called a segment table.
- There are two types of information stored in the segment table:
  - Limit and Base
- **Limit:** – The limit is the length or size of the segment
- **Base:** – The base is the base address of the segment.


Segment 0
Segment 2
Segment 3
Segment 1
Segment 4
Logical Address Space
Segment Number
base address
Limit
0
500
600
1
2500
800
2
1500
400
3
4600
200
4
3800
400
Segment Table
500
Segment 0
1100
1500
Segment 2
1900
2500
Segment 1
3300
3800
Segment 4
4200
4600
Segment 3
4800

Types of Segmentation:

- There are two types of Segmentation:
  - Simple Memory Segmentation
  - Virtual Memory Segmentation
- **Simple Memory Segmentation: –** In simple memory segmentation, each process is split into different segments, and at the run time, all the processes are loaded. Also, not all the processes need to be loaded into a contiguous way.
- **Virtual Memory Segmentation: –** As simple memory segmentation, in virtual memory segmentation, each process is split into different segments, but not all of them are residents at any point of time.

Benefits and Drawbacks of Segmentation



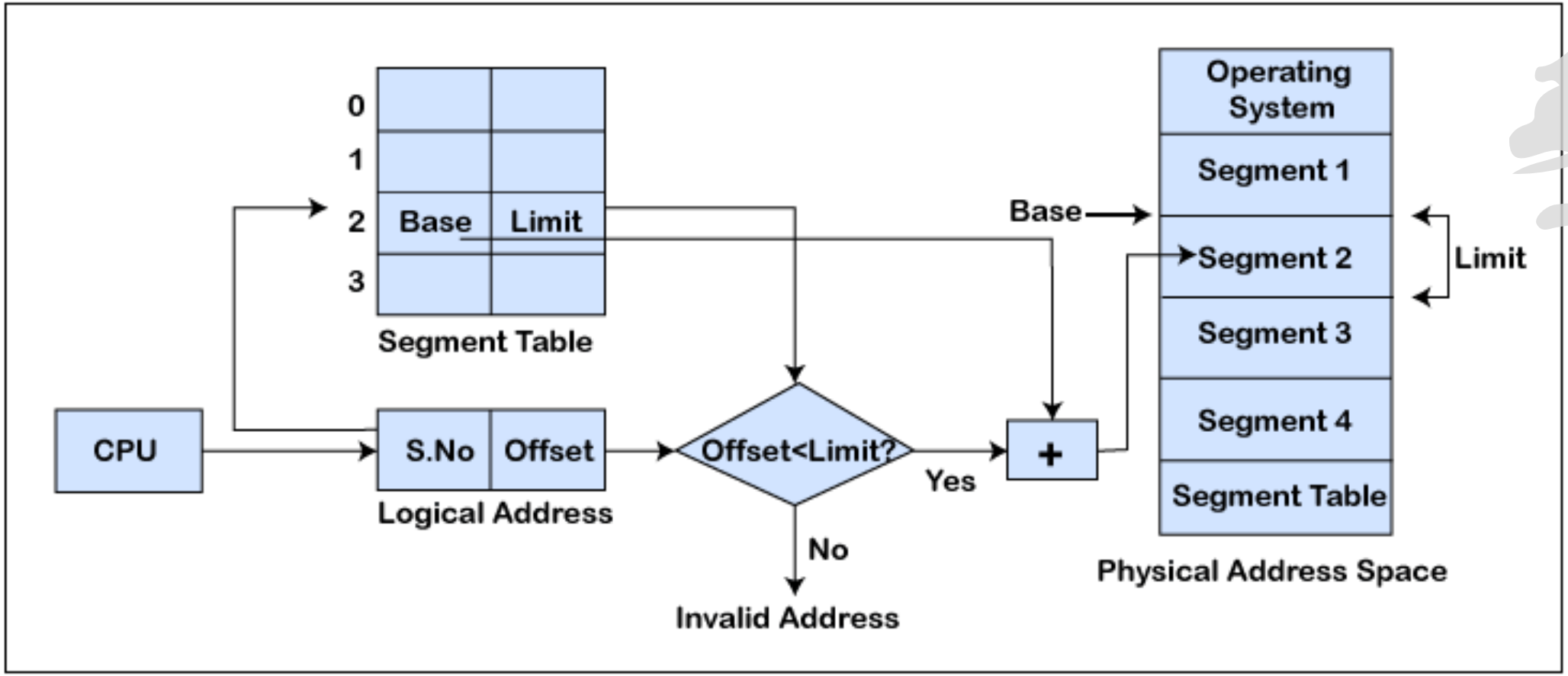
- **The benefits of Segmentation are:**
- Internal fragmentation is not present in the segmentation.
- Less overhead.
- The segment table size is less than the page table size.
- The relocation of the segment is easier than the whole address space.
- **The drawbacks of segmentation are:**
- There may be a chance of external fragmentation.
- This technique is expensive.
- It is tough to allocate memory in a contiguous manner to a variable-sized partition.

Why Segmentation is needed?

- Segmentation is closer to OS.
- In paging, all the processes are split into the pages. Sometimes there may be a situation where a process having some relevant sections of the function requires being loaded on the same page to increase the efficiency in processing.
- So, it is better to use the segmentation technique for memory management. In segmentation, the process is split into the segments. In this, every segment consists of a similar kind of function as a library function contained in one segment, and the main function contained in the other segment.

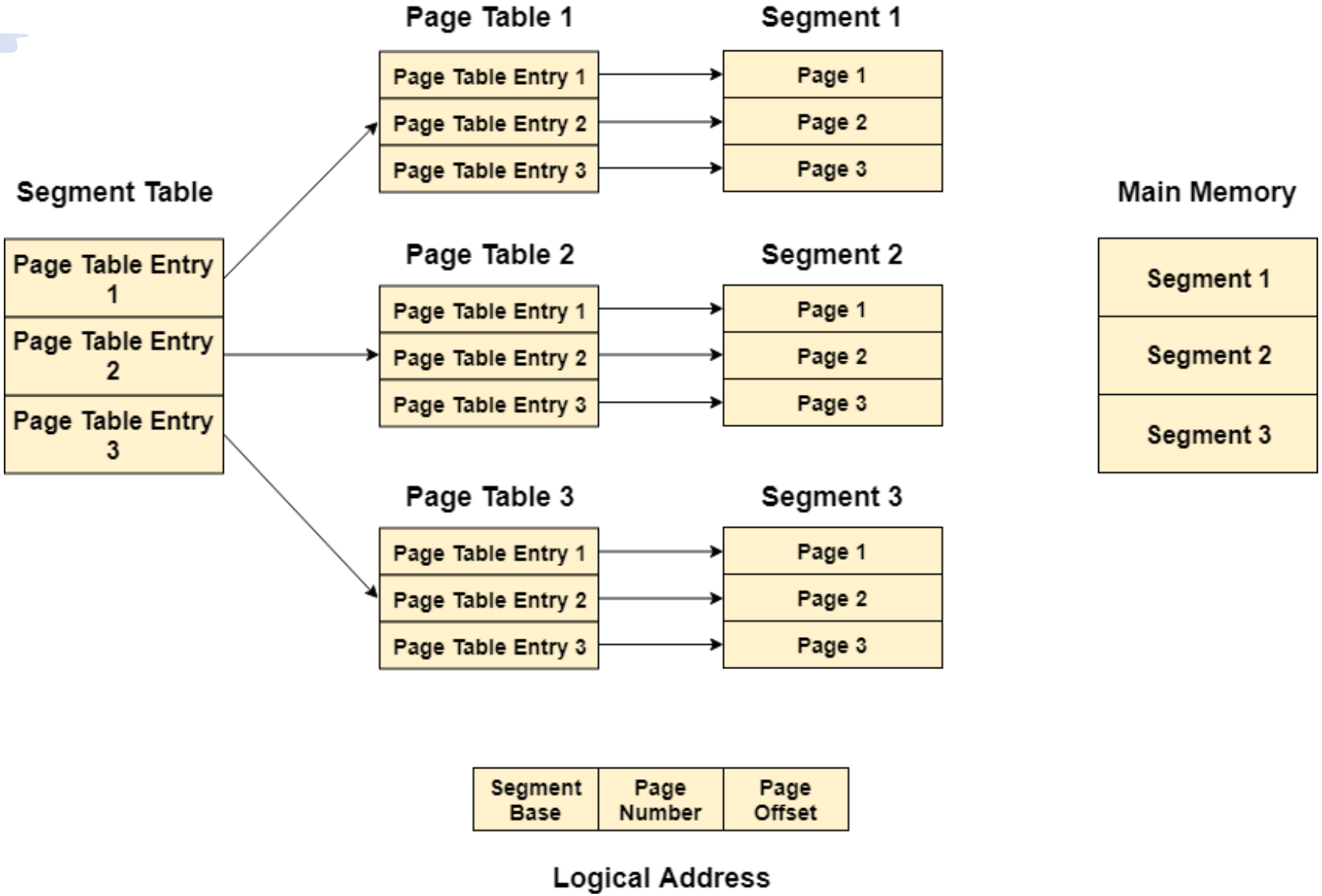
Translation of Logical Address into Physical Address by Segment Table

- The CPU generates a logical address that comprises of two parts:
  - Segment Number
  - Segment Offset
- **Segment Number: –** Segment Number is defined as the number of bits that are needed to represent the segment.
- **Segment Offset:** – Segment Offset is defined as the number of bits which are needed to represent the size of the segment.


0
1
2
3
Base
Limit
Segment Table
CPU
S.No
Offset
Logical Address
Offset<Limit?
Yes
No
Invalid Address
+
Base
Operating System
Segment 1
Segment 2
Segment 3
Segment 4
Segment Table
Limit
Physical Address Space

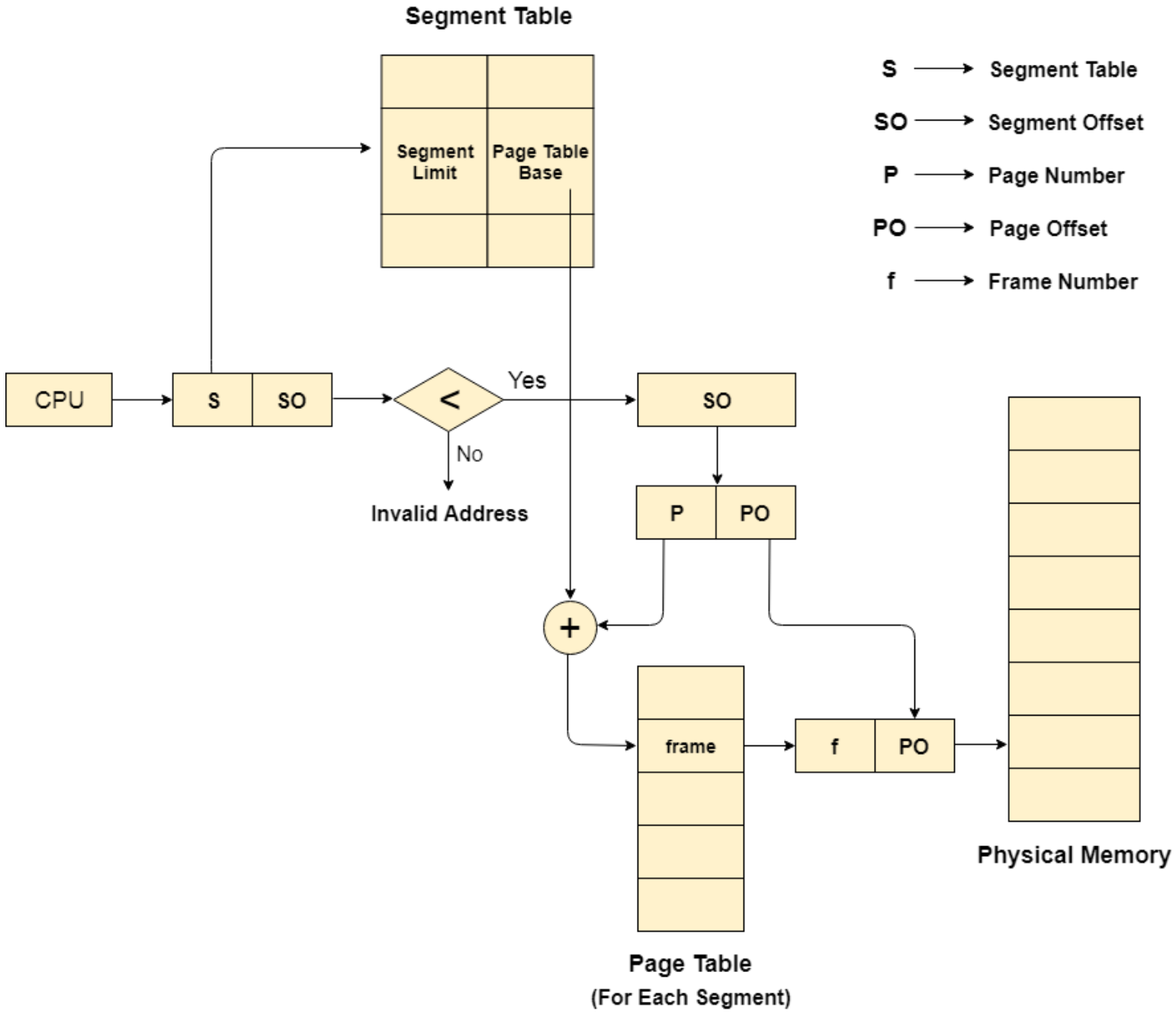
Segmented Paging

- Pure segmentation is not very popular and not being used in many of the operating systems. However, Segmentation can be combined with Paging to get the best features out of both the techniques.
- In Segmented Paging, the main memory is divided into variable size segments which are further divided into fixed size pages.
- **Segment Number** → It points to the appropriate Segment Number.
- **Page Number** → It Points to the exact page within the segment
- **Page Offset** → Used as an offset within the page frame


Page Table 1
Segment 1
Page Table Entry 1
Page Table Entry 2
Page Table Entry 3
Page 1
Page 2
Page 3
Segment Table
Page Table Entry 1
Page Table Entry 2
Page Table Entry 3
Main Memory
Segment 1
Segment 2
Segment 3
Page Table 2
Segment 2
Page Table 3
Segment 3
Segment Base
Page Number
Page Offset
Logical Address

Translation of logical address to physical address

- The CPU generates a logical address which is divided into two parts: Segment Number and Segment Offset. The Segment Offset must be less than the segment limit. Offset is further divided into Page number and Page Offset. To map the exact page number in the page table, the page number is added into the page table base.
- The actual frame number with the page offset is mapped to the main memory to get the desired word in the page of the certain segment of the process.


Segment Table
Segment Limit
Page Table Base
S → Segment Table
SO → Segment Offset
P → Page Number
PO → Page Offset
f → Frame Number
CPU
S
SO
<
Yes
No
Invalid Address
SO
P
PO
+
frame
f
PO
Physical Memory
Page Table
(For Each Segment)

Thank You