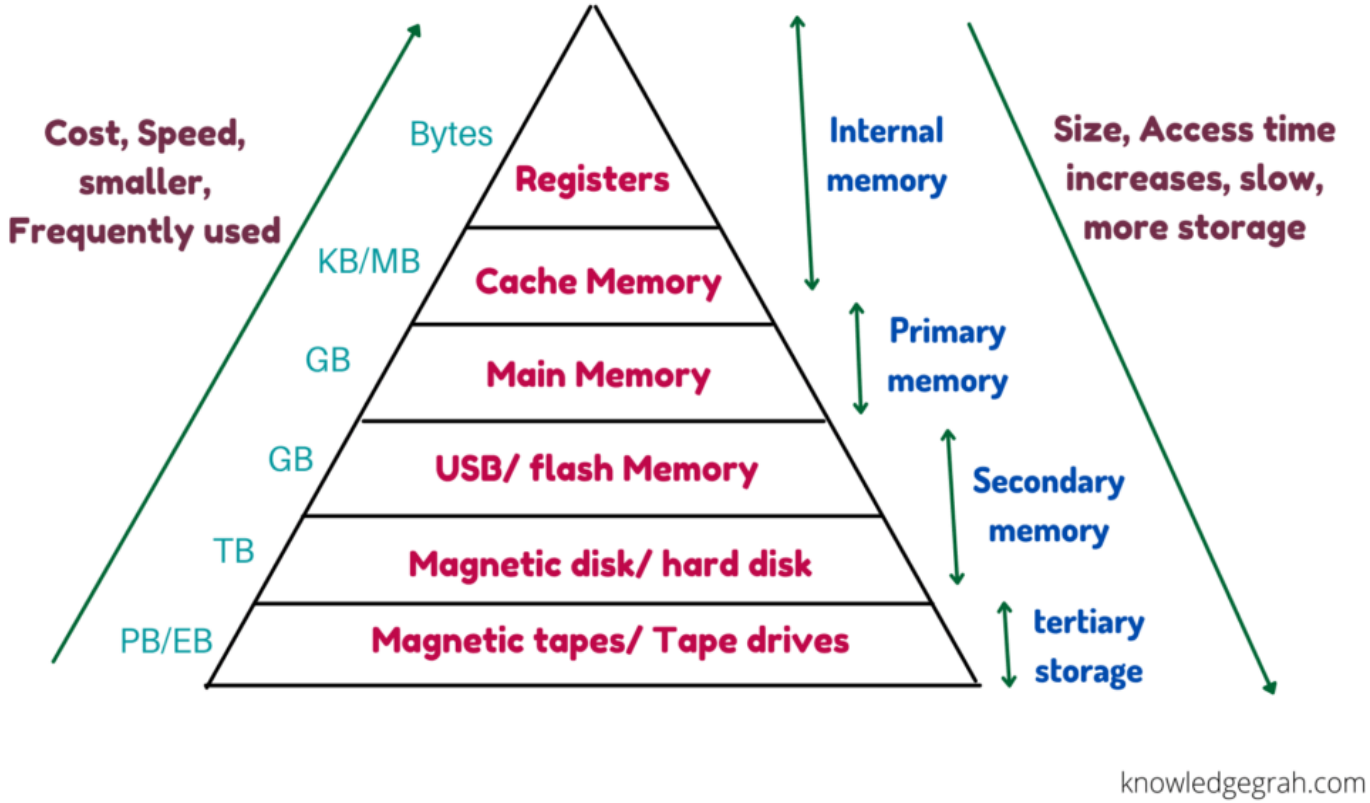
MEMORY MANAGEMENT

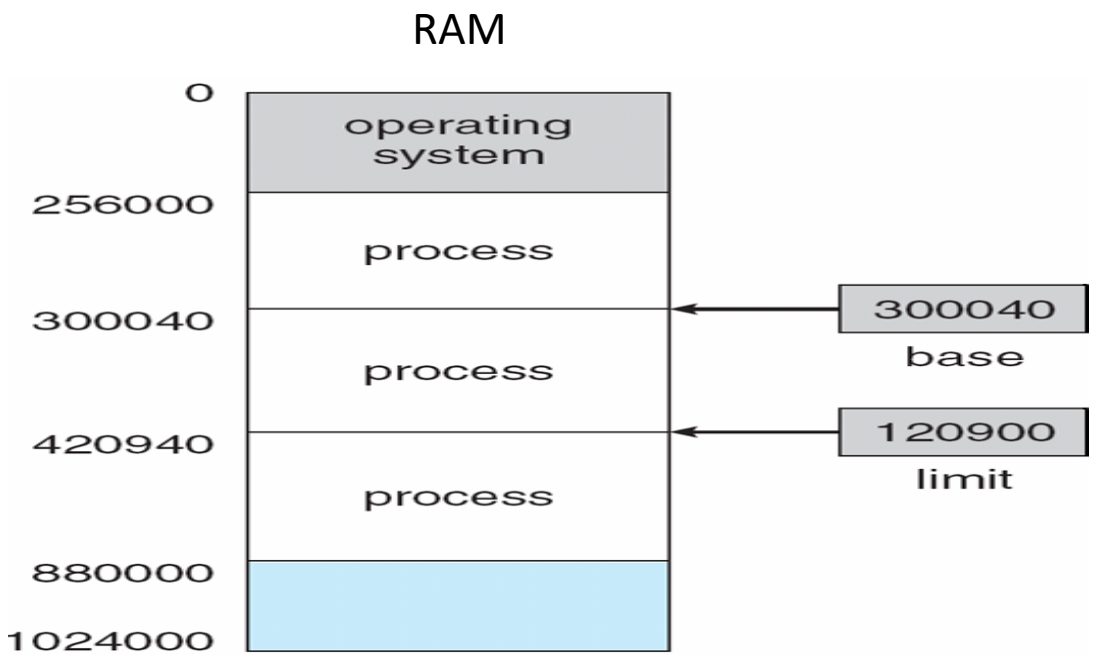
Memory Hierarchy



Memory management deals with the allocation and de allocation of space in main memory for programs or processes.

Performance of computer system is high when the CPU is kept busy at all times.

To keep the CPU busy at all times, number of processes or programs is loaded into main memory at a time.



When a process is blocked due to an input/output operation during its execution then the CPU is allocated to some other process.

Logical address

A program or process is a collection of statements or instructions.

When the user wants to execute his/her program then the operating system loads the program into RAM.

CPU don’t know the location of program inside the RAM.

To execute the program, CPU generates the addresses of statements in the program as 0, 1, 2, and so on.

When the CPU wants to execute 1st statement of the program then it generates the address 0.

Similarly, when the CPU wants to execute 2nd statement of the program then it generates the address 1.

This generation of addresses is continued till the last statement of the program.

The addresses (0, 1, 2, ….n-1) generated by the CPU for executing the statements of program are called logical addresses.

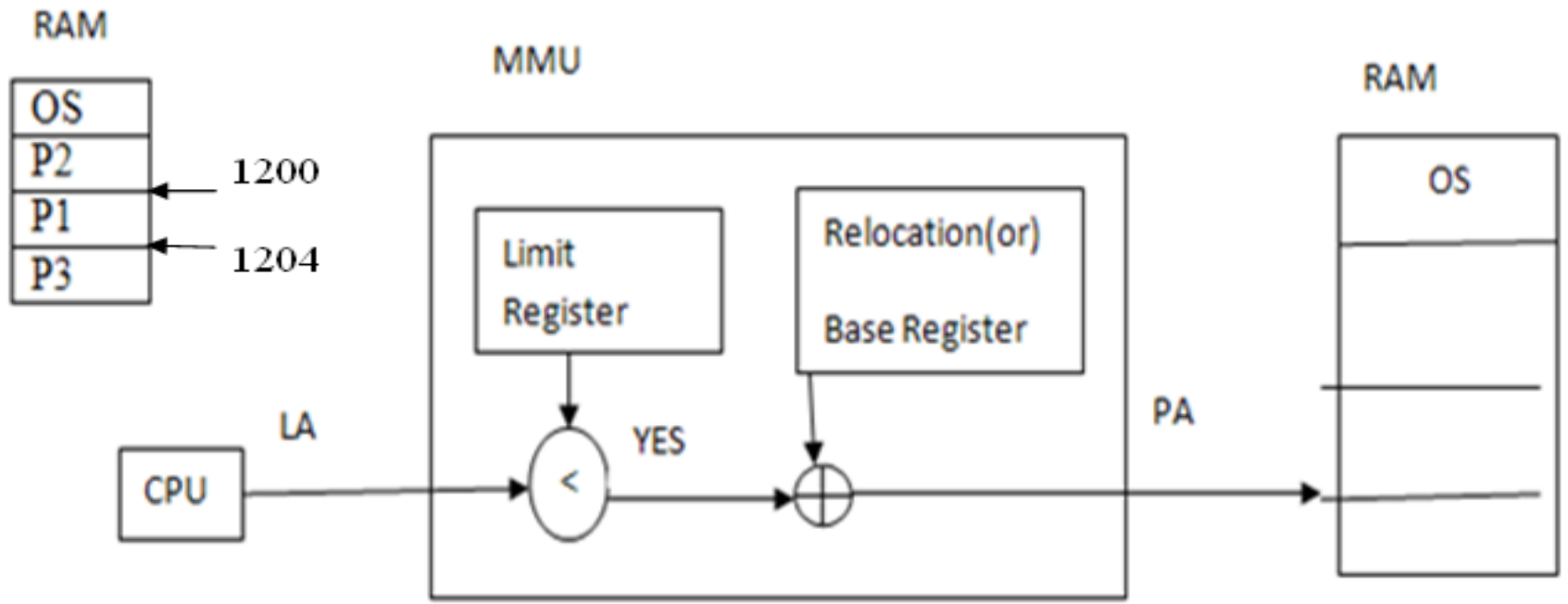
Physical address

Physical addresses are the addresses of locations inside RAM where the statements of program are loaded.

Memory Management Unit (MMU)

MMU maps the logical addresses to their corresponding physical addresses.

This mapping is also called as relocation.



Before starting the execution of a process, the address of starting location of process in the RAM is loaded into Relocation or Base Register and the number of statements in the process is loaded into the Limit Register.

For example, if a process P1 is loaded into RAM at the address 1200 and the number of statements in the process P1 is 5, when the execution of process P1 is started, the operating system loads the address 1200 into Relocation Register and value 5 into Limit Register.

While executing the process, the logical address generated by the CPU for executing a statement is compared with the value in Limit Register.

If the logical address value is less than the value in Limit Register then the logical address is added to the value in Relocation Register to generate the corresponding physical address.

Otherwise, the MMU reports an error to the operating system.

Memory Management Techniques

The operating system uses the following techniques to allocate space for programs in the main memory or RAM.

1) Contiguous Memory Allocation
2) Paging
3) Segmentation

Contiguous Memory Allocation

Operating system loads the entire program as a single unit into the RAM.

Different approaches used in contiguous memory allocation are:

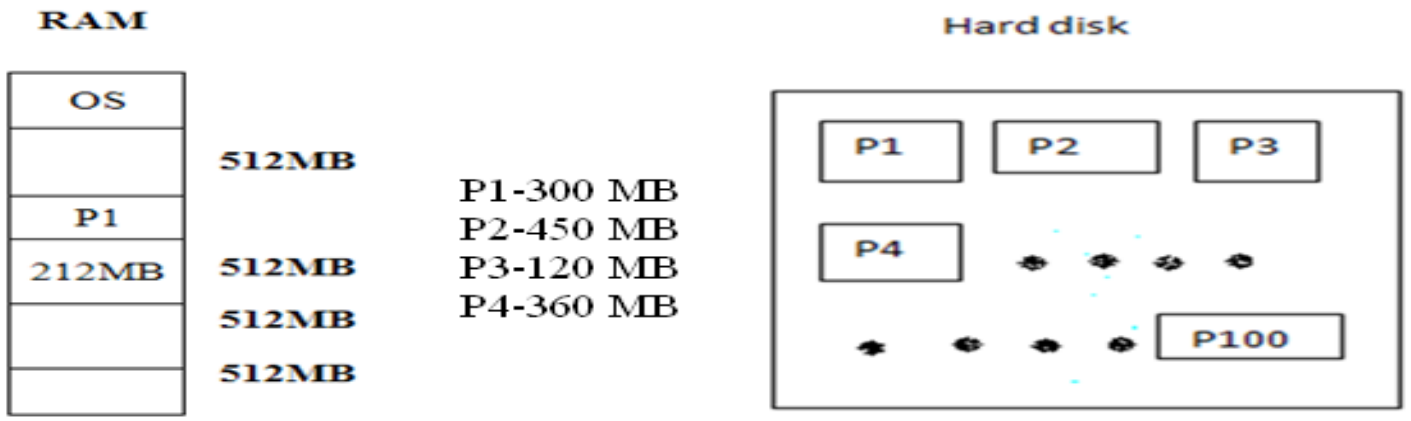
1) Equal size fixed partition
2) Unequal size fixed partition
3) Dynamic partition

Equal size fixed partition technique

Operating system divides the space in RAM into number of equal size parts before loading any program into RAM.

Only one program can be loaded into each partition of RAM.

Operating system can load a program into any free partition of RAM.



--->P5=600 MB

Advantage:

Operating system can select any free partition of RAM to load a program.

Disadvantages:

1) Some memory is wasted inside the partition when the size of program is less than the size of partition.

This wastage of memory is called "**internal fragmentation**".

For example, when program P1 is loaded into 2nd partition of RAM as shown in the above figure, 212 MB of memory in that partition is wasted.

2) When the size of program to be loaded is greater than the size of partition then it is not possible to load the program into RAM.

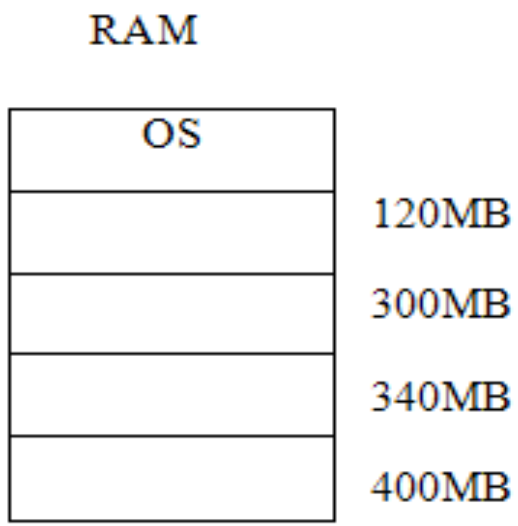
For example, if the size of program P5 is 600 MB then it is not possible to load P5 into RAM.

Un-equal size fixed partition

Operating system divides the space in RAM into number of parts with different sizes before loading any program into RAM.

Only one program can be loaded into each partition of RAM.

Operating system has to select a suitable partition of RAM for loading a program.



Advantage

Internal Fragmentation is less compared to equal size fixed partition.

Disadvantages

1) Operating System has to select a suitable partition for loading a program into RAM.

2) Some memory is wasted inside the partition when the size of process is less than the size of partition. This wastage of memory is called "**internal fragmentation**".

3) When the size of process is greater than the size of all partitions then it is not possible to load the process into RAM.

For example, if the size of process P5 is 600 MB then it is not possible to load P5 into RAM.

Dynamic partitions

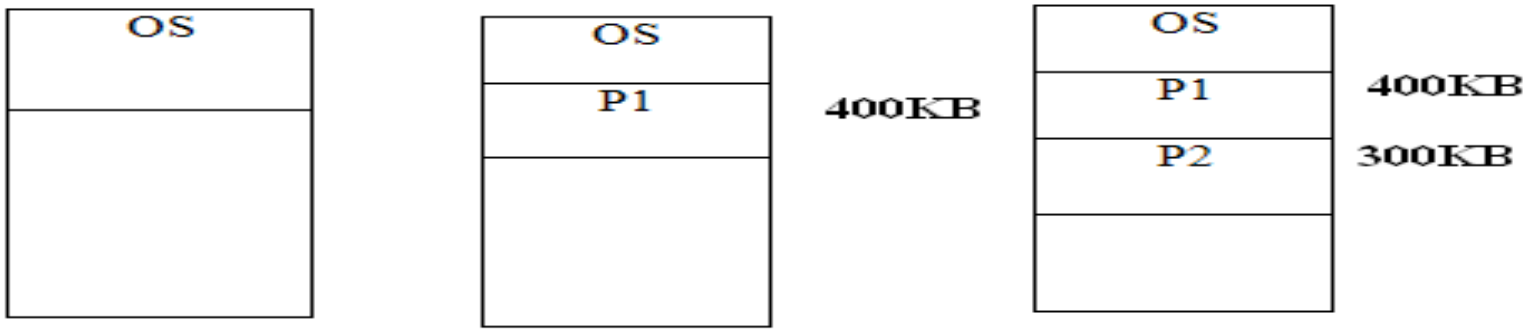
Operating system divides the space in RAM into parts at the time of loading programs into RAM.

Initially, RAM contains two parts.

Into one part, Operating System is loaded and the other part is used for loading user programs.

When a program say P1 is loaded into RAM then the RAM is divided into three parts.

When another program say P2 is loaded then the RAM is divided into four parts as shown in below figure.



Advantage

There is no wastage of memory inside partitions. So, no internal fragmentation.

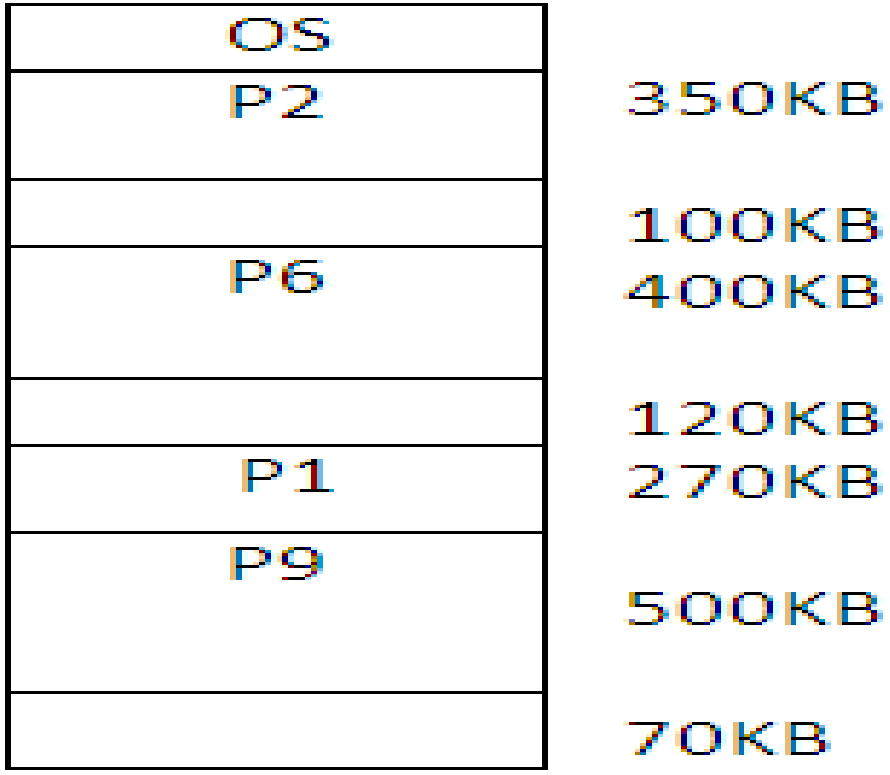
Disadvantages:

Some space is wasted outside of the partitions.

This wastage of space is called as ‘External Fragmentation’.

Consider the following position of RAM which is the result of loading some programs into RAM and removing completed programs from RAM.

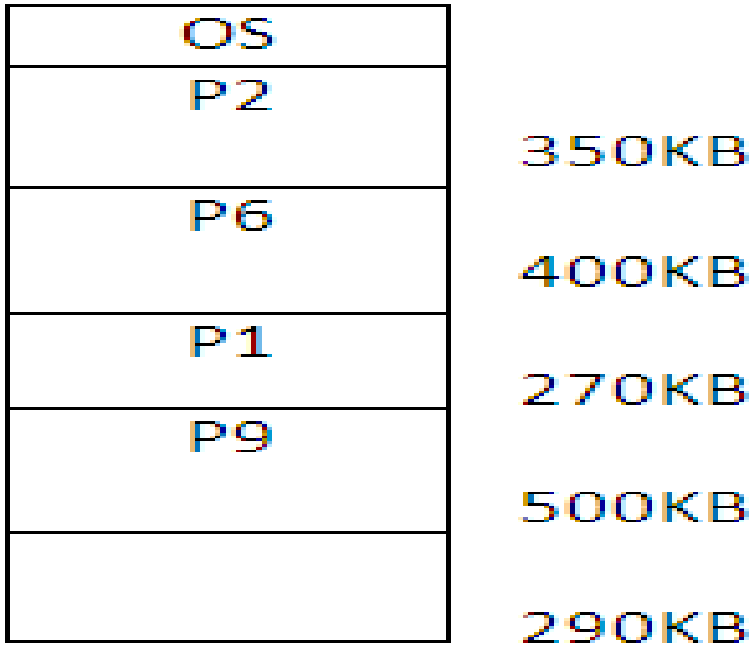
If the operating system wants to load a program say P20 with size 250 KB, then it is not possible to load that program into RAM as there is no free part with size greater than or equal to 250KB in the RAM.



To avoid external fragmentation, **compaction** technique can be used.

Compaction

Compaction technique moves all free spaces in the RAM together. After applying compaction technique to the above position of RAM, the position of RAM becomes



Now, the operating system can load program P20 into RAM.

With unequal size fixed partition and dynamic partition techniques, the operating system has to select suitable partition in the RAM for loading a program.

Operating System uses any of the following placement algorithms to select the suitable partition.

Placement Algorithms

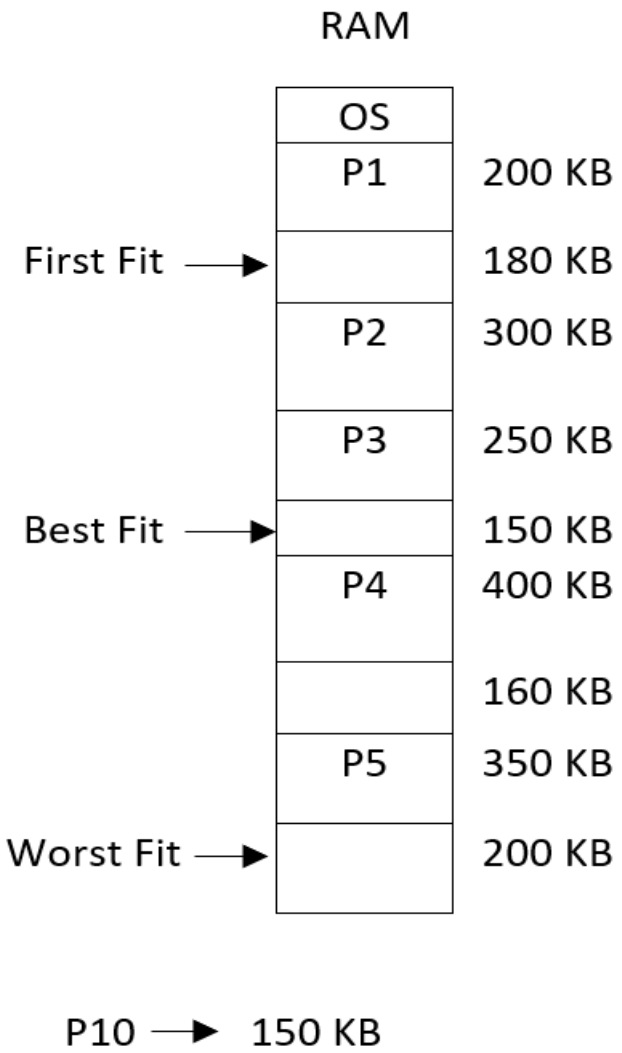
1) FIRST FIT
2) BEST FIT
3) WORST FIT

First fit algorithm scans the RAM from starting to ending and selects the first free partition whose size is greater than or equal to the size of program.

Best fit algorithm scans the RAM from starting to ending and selects the free partition whose size is very close to the size of program.

Worst fit algorithm scans the RAM from starting to ending and selects the largest free partition.

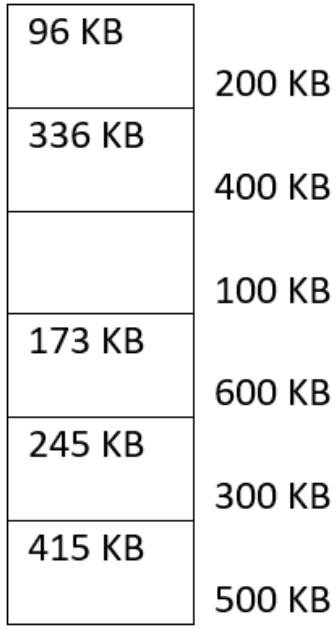
Consider the following position of RAM. To load the program P10 with size 150 KB, the partition selected by operating system with different algorithms is indicated in below diagram.



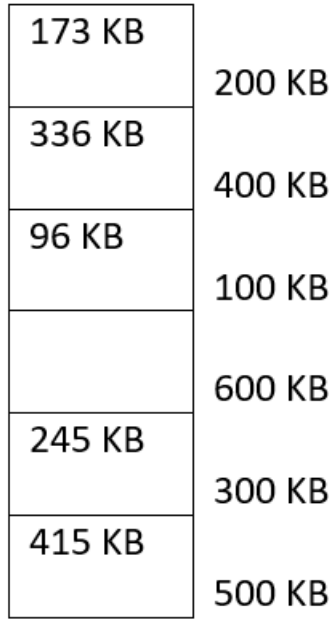
Assume the memory is divided into six partitions with size 200 KB, 400 KB, 100 KB, 600 KB, 300 KB, 500 KB (in order). Calculate the total internal fragmentation when the processes with size 336 KB, 96 KB, 173 KB, 415 KB, 245 KB (in order) are placed with

i) first-fit algorithm
ii) best-fit algorithm

First Fit



Best Fit



Total internal fragmentation with First Fit= (200-96)+(400-336)+(600-173)+(300-245)+(500-415)=735 KB

Total internal fragmentation with First Fit= (200-173)+(400-336)+(100-96)+(300-245)+(500-415)=235 KB

Given five memory partitions of 250 KB, 400 KB, 360 KB, 500 KB, and 180 KB (in order), how would the first-fit, best-fit algorithms place processes of 315 KB, 221 KB, 135 KB, and 284 KB (in order)? Which algorithm makes the most efficient use of memory?

221	250	221
315	400	284
135	360	315
284	500	-
-	180	135
First		Best

1)Internal Fragmentation of First Fit= 555KB
2)Internal Fragmentation of Best Fit= 235KB

Paging Technique

RAM is divided into a number of equal size frames before loading any program into RAM.

To load a program into RAM, the operating system divides the program into a number of equal size pages.

Size of page is equal to size of frame.

Generally, the page size or frame size is selected as a power of 2.

Then, the operating system loads pages of the program into RAM wherever free frames are available.

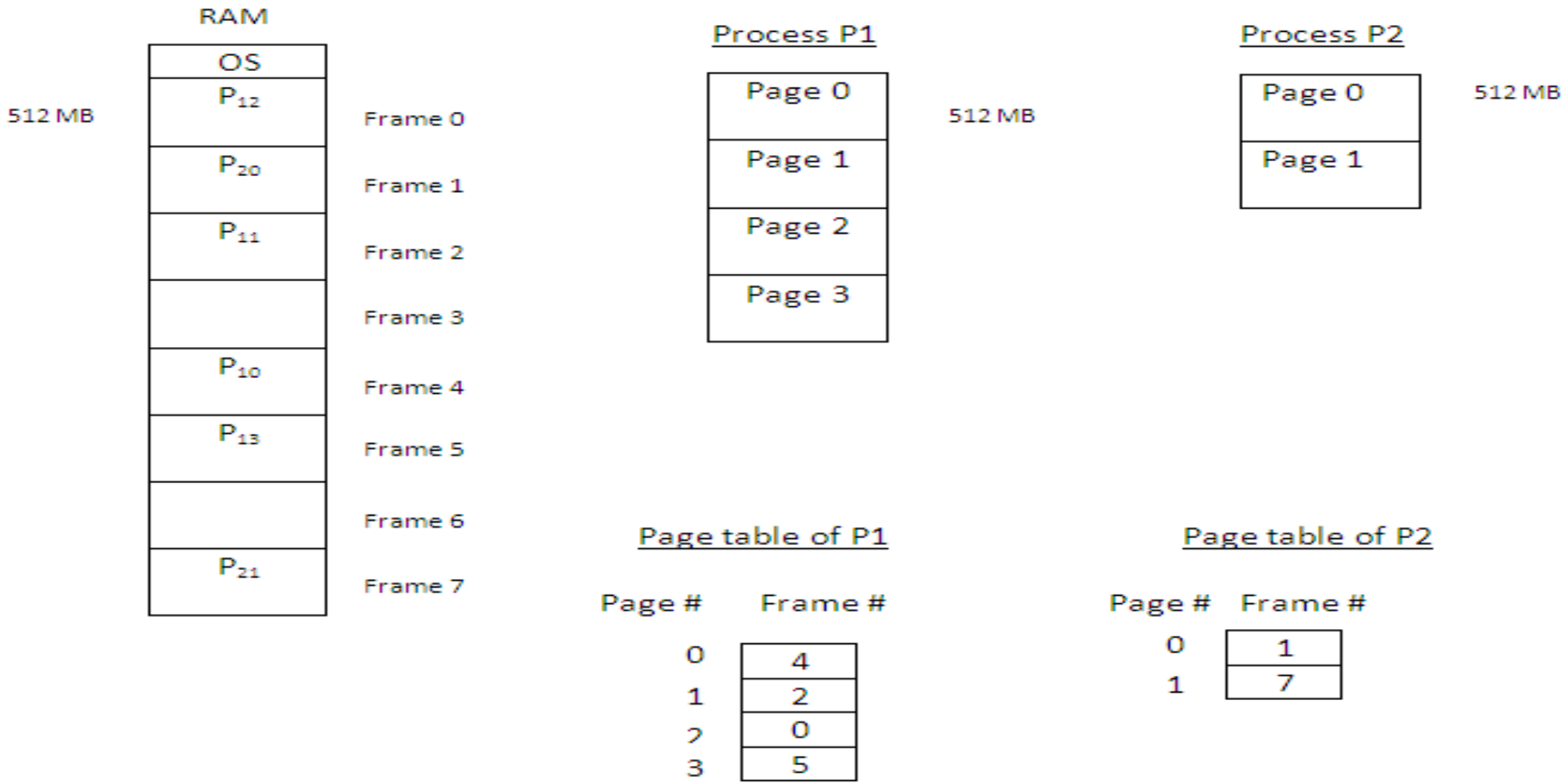
Later, the operating system creates a page table for the program.

Number of entries in the page table is equal to the number of pages in the program.

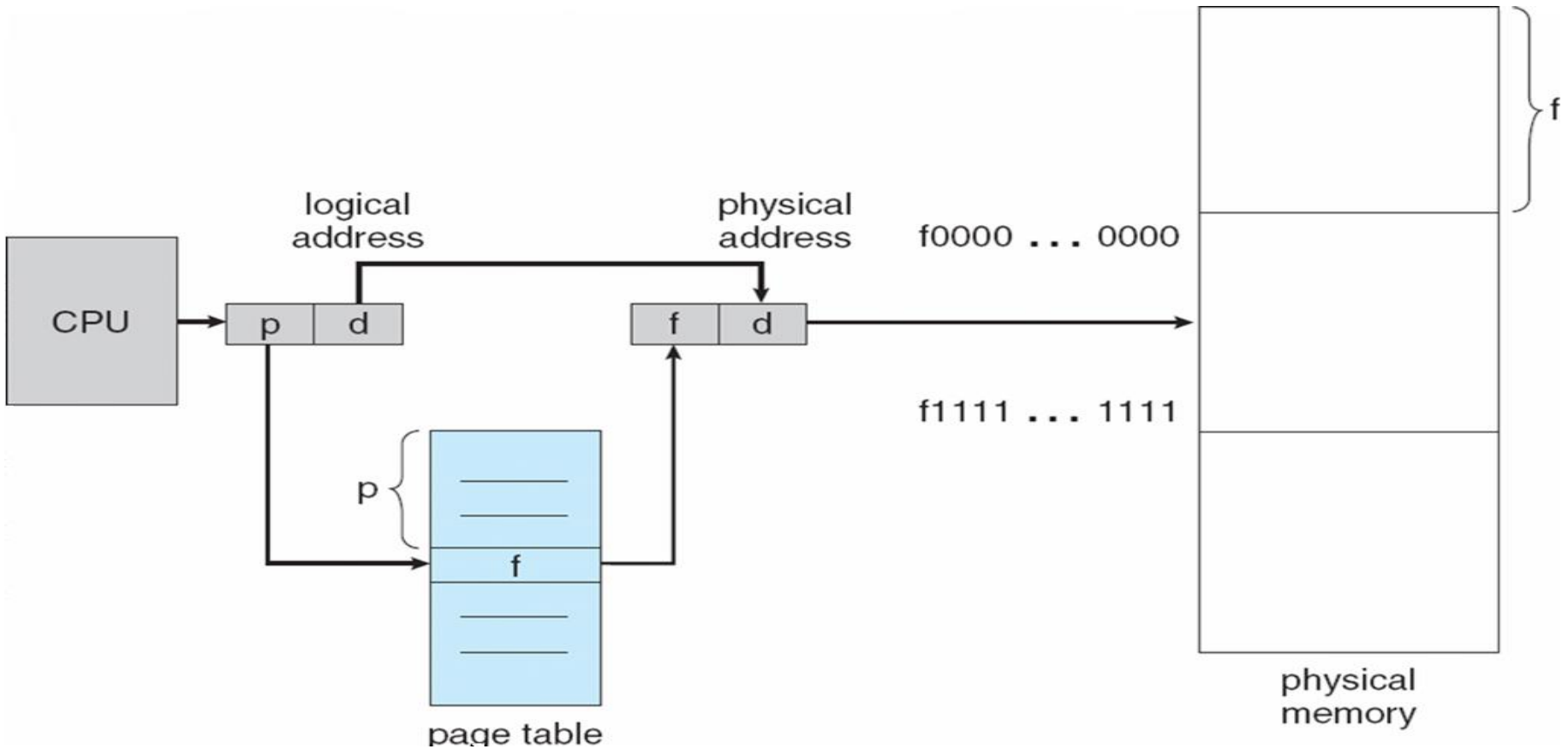
The entries of page table are indexed with page numbers.

The operating system stores frame numbers of RAM into the entries of the page table.

The following diagram shows how programs are loaded into RAM with the paging technique.



Mapping Logical Address to Physical Address in Paging (or) Hardware support for Paging



To start the execution of any process or program, the operating system divides the program into equal size pages, loads these pages into free frames of RAM, creates a page table for the program and stores frame numbers into the entries of the page table.

To execute a statement of the program, the CPU generates the logical address of that statement.

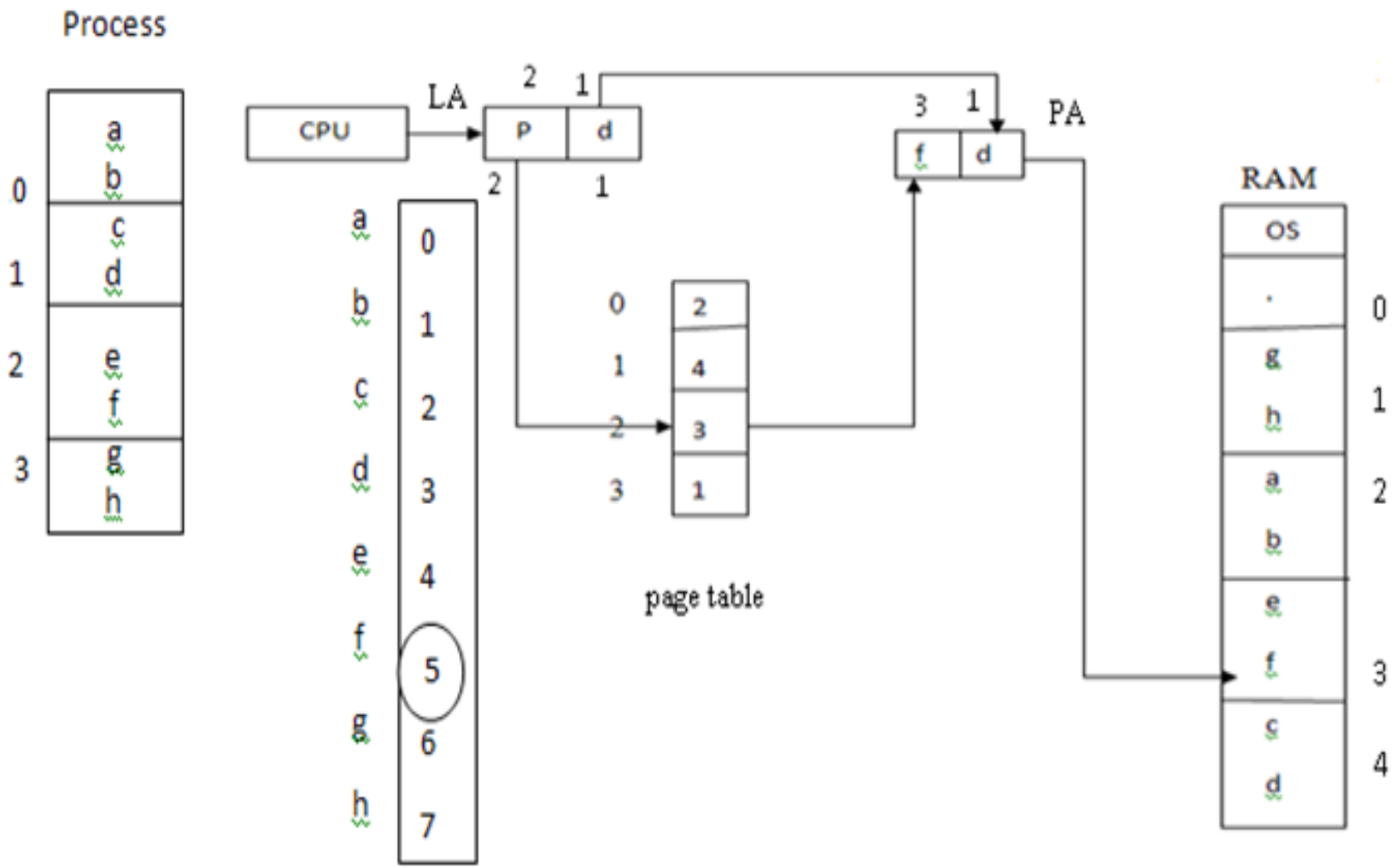
This logical address consists of two parts: page number (p) and offset (d).

The page number is used as an index into the page table of the process and the corresponding frame number is identified.

The identified frame number is appended with offset of logical address to get the physical address of the statement.

The statement in the RAM at the generated physical address is transferred to the CPU for execution.

The offset identifies the statement inside the page or frame.


Process
a
b
c
d
e
f
g
h
0
1
2
3
CPU
LA
2
1
P
d
2
1
a
b
c
d
e
f
g
h
0
1
2
3
4
5
6
7
0
1
2
3
2
4
3
1
page table
3
1
f
d
PA
RAM
OS
0
g
h
1
a
b
2
e
f
3
c
d
4

Consider a logical address space of 256 pages with 64 words per page mapped onto a physical memory of 4096 frames.

i. How many bits are required in the logical address?
ii. How many bits are required in the physical address?

Size of logical address = size of page number field + size of offset field = 8 bits (2^8 =256) + 6 bits (2^6 =64) = 14 bits

Size of physical address = size of frame number field + size of offset field = 12 bits (2^{12} =4096) + 6 bits (2^6 =64) = 18 bits

Ex: Consider a machine with 128 MB physical memory and a 32 bit virtual address space. If the page size is 8 KB.

i) Find the number of bits in physical address

ii) Find the number of frames in the Main memory

iii) Find the number of bits allocated for offset

iv) Find the number of entries in page table

Size of physical memory/main memory/primary memory/RAM = 128 MB = 128 X 2^{20} B = 2^{7} X 2^{20} B = 2^{27} B

Size of virtual address/logical address = 32 bits

Size of program = 2^{32} B

Size of page = 8 KB = 8 X 1024 B = 2^{3} X 2^{10} B = 2^{13} B

Size of frame = 2^{13} B

i) Number of bits in physical address or size of physical address = 27 bits

ii) Number of frames in main memory = size of main memory/size of frame = 2^{27} B / 2^{13} B = 2^{14} frames

iii) Number of bits allocated for offset = 13 bits

iv) Number of entries in page table=Number of pages in the program=size of program/size of page=2^{32} B / 2^{13} B = 2^{19} entries

Ex: In a virtual memory system, size of virtual address is 64-bit, size of physical address is 60-bit, page size is 8 Kbyte and size of each page table entry is 32 bytes. The main memory is byte addressable.

i) Calculate the Main memory size.

ii) Calculate the number of frames in the main memory.

iii) Calculate the page table size.

iv) Find the number of bits used for offset.

Size of virtual address/logical address = 64 bits

Size of physical address = 60 bits

Size of page = 8 KB = 8 X 1024 B = 2^3 X 2^{10} B = 2^{13} B

Size of each page table entry = 32 B = 2^5 B

i) Size of physical memory/main memory/primary memory/RAM = 2^{60} B

ii) Number of frames in the main memory = size of main memory/size of frame = 2^{60} B / 2^{13} B = 2^{47} frames

iii) Size of page table = Number of entries in page table X Size of each entry in page table = Number of pages in the program X 2^5 B = (size of program/size of page) X 2^5 B = (2^{64} B / 2^{13} B) X 2^5 B = 2^{51} X 2^5 B = 2^{56} B

iv) Number of bits used for offset = 13 bits

Ex: Size of logical address is 48-bit, page size is 4 Kbyte, size of main memory is 256 MB and size of each page table entry is 32 bytes. The main memory is byte addressable.

i) Find the number of bits in physical address

ii) Calculate the number of frames in the main memory.

iii) Calculate the page table size.

iv) Find the number of bits used for offset.

LA=48 bit; LAS=2^48B
PAS=256 MB=2^28B
Page Size=4KB=2^12B

1. 28 Bits
2. 2^16 Frames
3. 2^36*2^5B
4. 12 Bits

Translation Look-aside Buffer (TLB)

The operating system can store the page table of a process in either registers or RAM.

The operating system stores the page table in the registers of computer system when the number of entries in the page table is less.

Otherwise, stores the page table in the RAM.

When the page table is stored in the registers, then one memory access is enough for getting a statement of the process from RAM.

When the page table is stored in the RAM, then two memory accesses are required for getting a statement of the process from RAM.

To get any statement from the RAM, frame number of that statement needs to be obtained first.

To get frame number of the statement, page table of the process stored in the RAM has to be accessed and it requires one memory access.

One more memory access is to get the statement from the identified frame of the RAM.

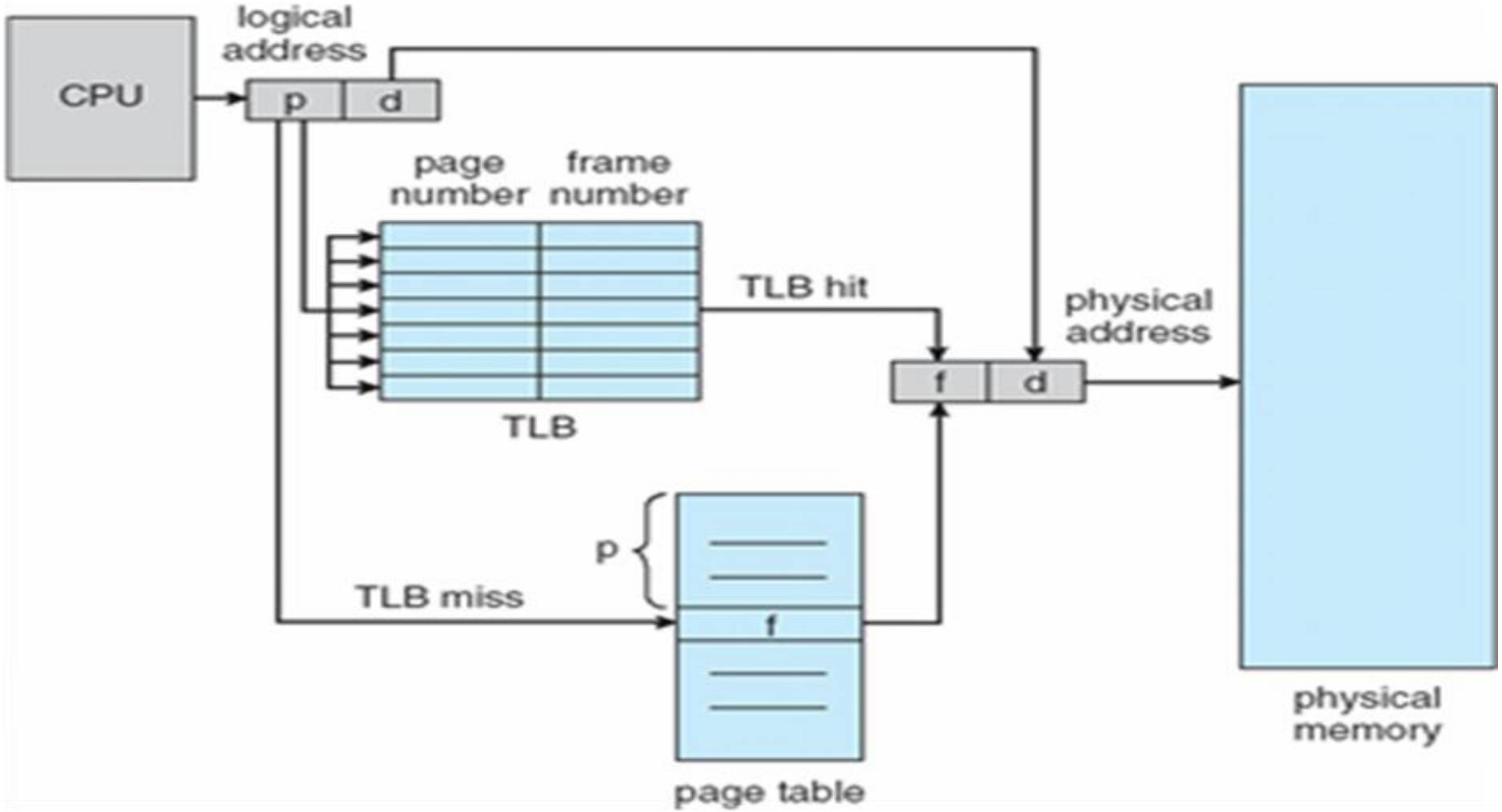
To get a statement of the process from RAM with one memory access or to reduce the time for access, information about frequently used pages is stored in the translation look-aside buffer (TLB).

TLB contains a number of entries.

Each entry of the TLB contains a page number and its corresponding frame number.

TLB is associative in nature. All entries of TLB are searched at a time in parallel.

The following diagram shows the usage of TLB in paging.



When the CPU generates a logical address, the page number of logical address is first searched in the TLB.

If the page number is found in the TLB (TLB hit) then the corresponding frame number is appended with the offset of logical address to generate the corresponding physical address.

Otherwise (TLB miss), the page number of logical address is used as an index into the page table, the corresponding frame number is identified and then appended with the offset of logical address to generate the physical address.

Hit ratio is the percentage of times that a particular page number is found in the TLB.

An 80 percent hit ratio means that we find the desired page number in the TLB 80 percent of the time.

The formula to calculate the effective memory access time is

Effective access time = probability of finding the page in the TLB x time to access the statement from the memory + probability of not finding the page in the TLB x time to access the statement from the memory

If the hit ratio is 80 percent and it takes 20 nanoseconds to search the TLB, 100 nanoseconds to access the memory then the effective memory access time is

Effective access time = 0.80 x (20+100) + 0.20 x (20+100+100) = 0.8 x 120 + 0.2 x 220 = 140 nanoseconds

If the hit ratio is 98 percent and it takes 20 nanoseconds to search the TLB, 100 nanoseconds to access the memory then the effective memory access time is

Effective access time = 0.98 x 120 + 0.02 x 220 = 122 nanoseconds

Segmentation Technique

To load a program into RAM, the operating system divides the program into a number of segments of either equal or unequal size.

Divides the RAM into a number of parts at the time of loading the segments of the program.

The operating system loads the segments of program into RAM wherever free space is available.

Then, create a segment table for the program.

The number of entries in the segment table is equal to the number of segments in the program.

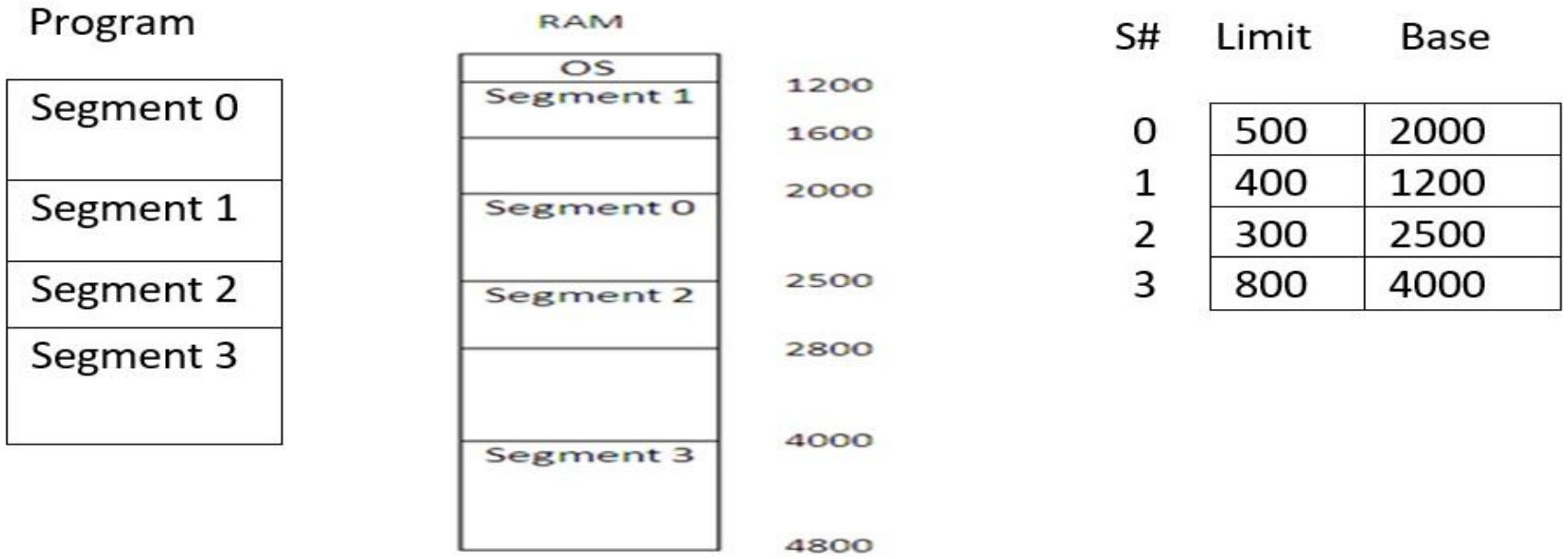
Each entry of the segment table contains a base which indicates the starting address of the segment in the RAM, and a limit which indicates the number of statements in that segment.

In the following figure, a program containing 2000 statements is divided into four segments.

The number of statements in segment 0, segment 1, segment 2 and segment 3 are 500, 400, 300 and 800 respectively.

The segments are loaded into RAM and then a segment table is created for the program.

The segment table contains four entries.



Mapping Logical Address to Physical Address

To execute a statement of the process, the CPU generates the logical address of the statement.

The logical address consists of two parts: segment number and offset.

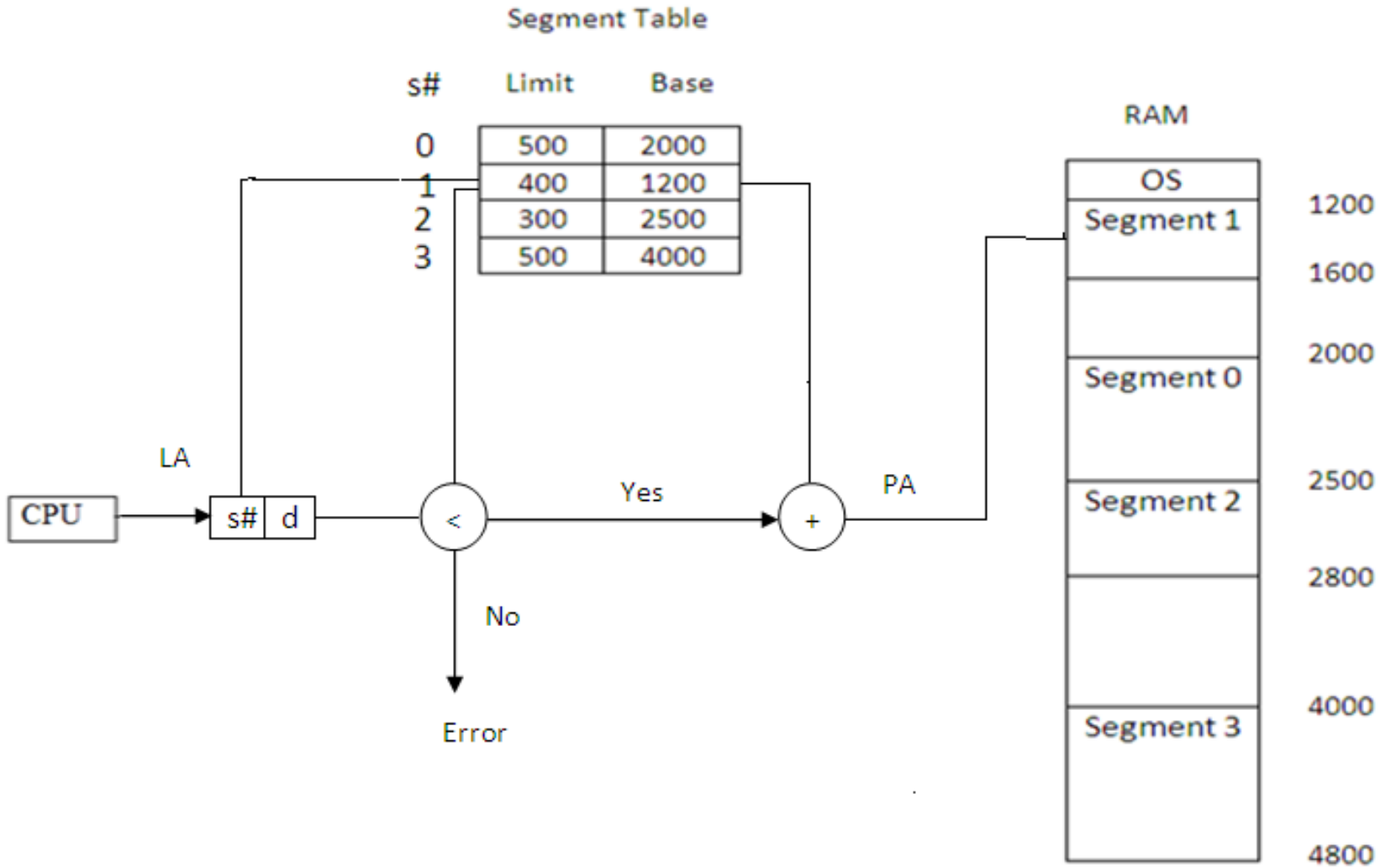
The segment number is used as an index into the segment table to identify an entry of the segment table.

Then, the offset value of logical address is compared with the limit value in the identified entry.

If the offset value is less than the limit value, then the offset value is added to the base value to generate the physical address.

Otherwise, an error is reported.

The following diagram depicts this mapping procedure.


Segment Table
s#
Limit
Base
0
1
2
3
500
2000
400
1200
300
2500
500
4000
RAM
OS
Segment 1
Segment 0
Segment 2
Segment 3
1200
1600
2000
2500
2800
4000
4800
LA
CPU
s#
d
<
Yes
No
Error
+
PA

Consider the following segment table:

Segment	Base	Length
0	620	450
1	1800	60
2	170	74
3	2145	321
4	1450	113

What are the physical addresses for the following logical addresses?

a. 0, 512

b. 1, 47

c. 2, 13

d. 3, 185

Physical addresses are:

a) Trap to the operating system as 512>450

b) 1800+47=1847 (47<60)

c) 170+13=183 (13<74)

d) 2145+185=2330 (185<321)

Paging with Segmentation

The operating system divides the RAM into equal size frames before loading any program into the RAM.

To load a program into RAM, the operating system divides the program into a number of segments.

Later, divides each segment into a number of pages.

Then, loads the pages of the program into free frames of RAM.

Creates a segment table for the program.

Each entry of the segment table points to the page table of the corresponding segment.

Each entry of the page table contains the corresponding frame number.

The logical address generated by the CPU contains 3 parts: segment number, page number, offset

Address space

The collection of addresses is called the address space.

The collection of logical addresses of a program is called the logical address space of the program.

The collection of physical addresses of a program is called the physical address space of the program.

Swapping

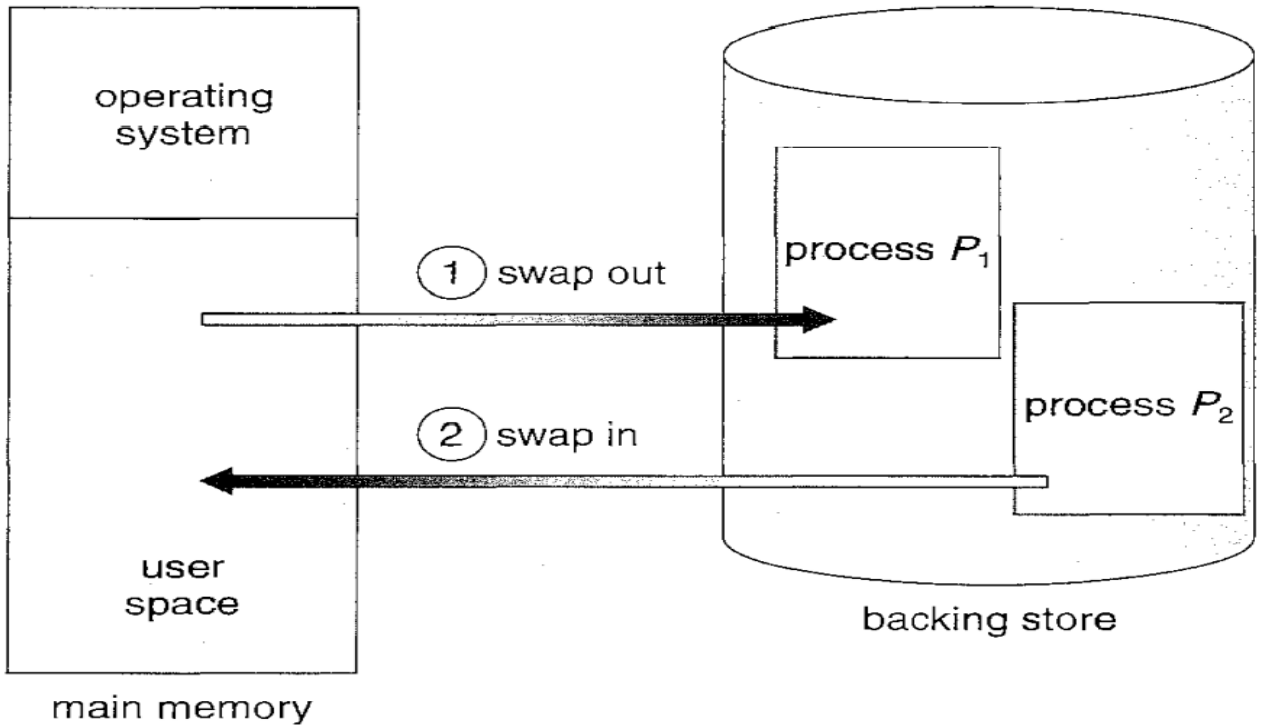
A process must be in memory (RAM) to be executed.

A process can be swapped temporarily out of RAM to a backing store (HD) and then brought into RAM for continued execution.

Swapping allows more processes to be run than can be fit into memory at one time.

For example, assume a multiprogramming environment with a round-robin CPU-scheduling algorithm.

When a quantum expires, the memory manager will start to swap out the process that just finished and to swap another process into the RAM space that has been freed.



When each process finishes its quantum, it will be swapped with another process.

A variant of this swapping policy is used for priority-based scheduling algorithms.

If a higher-priority process arrives and wants service, the memory manager can swap out the lower-priority process and then load and execute the higher-priority process.

When the higher-priority process finishes, the lower-priority process can be swapped back in and continued.

The system maintains a ready queue consisting of all processes that 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 RAM.

If it is not, and if there is no free memory region, the dispatcher swaps out a process currently in RAM and swaps in the desired process.

A process that is swapped out will be swapped back into either the same memory space it occupied previously or to a different location.

A process to be swapped out should be completely idle.

A process waiting for an i/o operation is generally not swapped out.

Relocation

Relocation is moving a program in the RAM from one location to another.