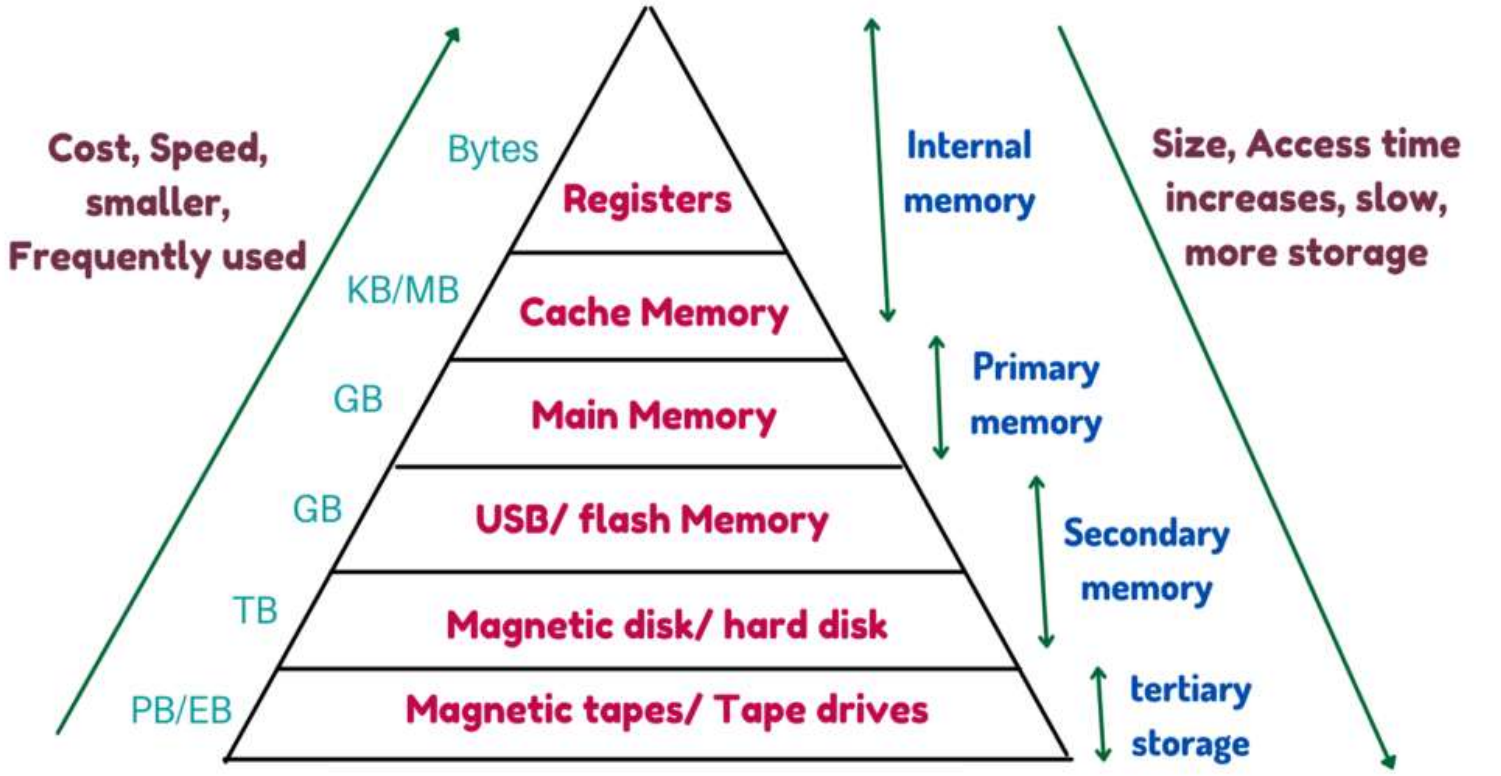
Module 4

MEMORY MANAGEMENT

Memory Hierarchy



knowledgegrah.com

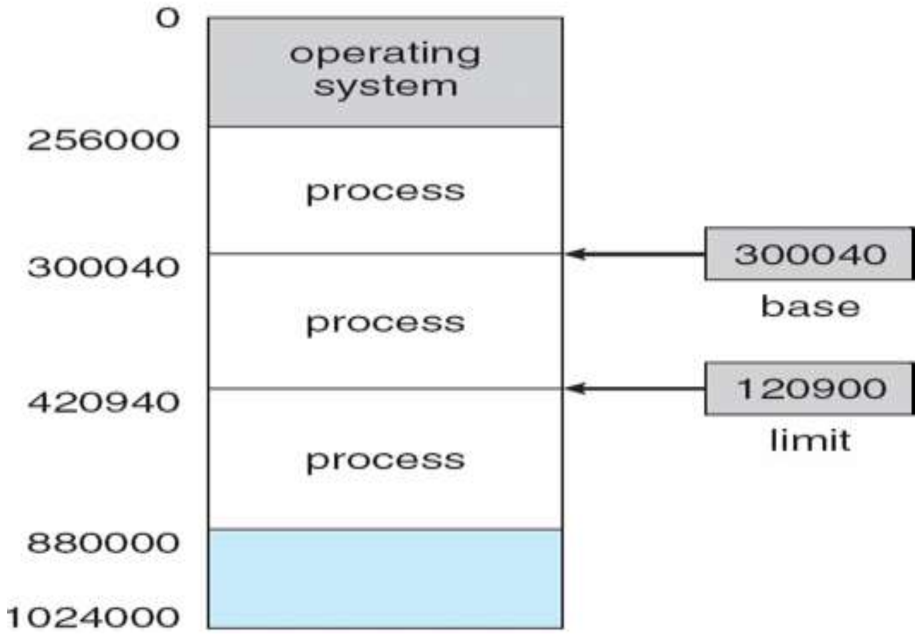
Memory management deals with the allocation and de allocation of space in main memory for programs or processes.

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

A number of processes or programs is loaded into main memory at a time in order to keep the CPU busy at all times.

During the execution of a process, if the process waits for an input/output then the CPU is switched to other process in the main memory.

Main Memory/Primary Memory/Random Access Memory



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 main memory.

CPU don’t know the location of program inside the main memory.

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

For example, CPU generates the logical address 0 to execute the 1st statement in the program.

CPU generates the logical address 1 to execute the 2nd statement in the program.

The logical address range is 0 to n-1, where n is the number of statements in the program.

Logical address space

The set of logical addresses generated by the CPU during the execution of the program is called as logical address space.

For example, if the program has 10 statements then the logical address space includes the logical addresses 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Physical address

Physical address refers to a location inside the main memory where a statement of the program is loaded.

Physical address space

The set of physical addresses of locations inside main memory where the statements of the program are loaded.

For example, if a program has 10 statements and are loaded into main memory locations whose address starts from 20340 then the physical address space includes the physical addresses 20340, 20341, 20342, 20343, 20344, 20345, 20346, 20347, 20348, 20349.

Memory Management Unit (MMU)

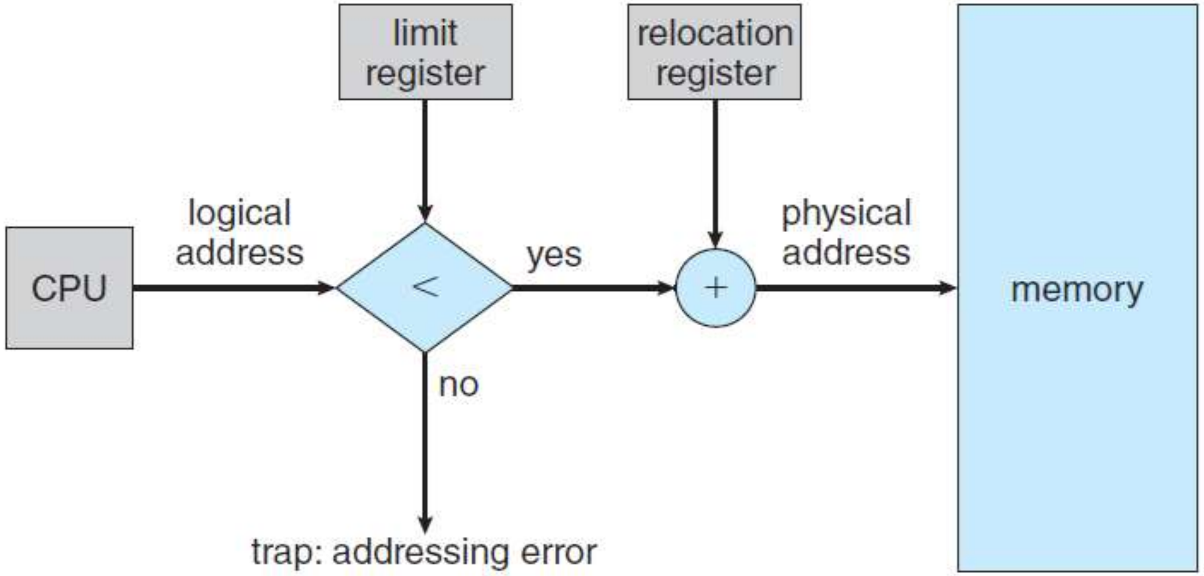
MMU maps the logical addresses to their corresponding physical addresses.

This mapping is also called as relocation.

The MMU maintains two registers: Base or relocation register and limit register.

The relocation register is used to store the starting physical address of the process.

The limit register is used to store the number of statements in the process.



Before starting the execution of a process, the starting physical address of the process 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 main memory 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.

1) Contiguous Memory Allocation
2) Paging
3) Segmentation

Contiguous Memory Allocation

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

Different approaches used in contiguous memory allocation are:

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

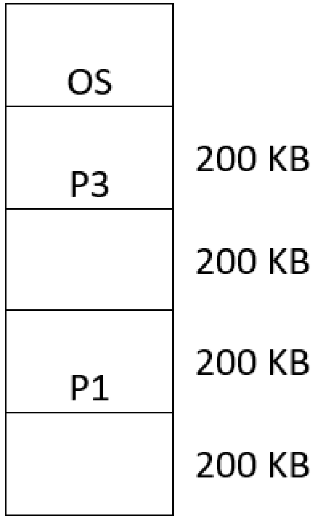
Equal size fixed partition technique

Before loading any program into main memory, the operating system divides the space in main memory into equal size parts.

Only one program can be loaded into each partition of main memory.

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

Main Memory



OS

P3 200 KB

200 KB

P1 200 KB

200 KB

Advantage

To load a program, the operating system can select any free partition of main memory.

Disadvantages

1) Wastage of space inside the partition when the size of program is less than the size of partition.

Wastage of space inside the partition is called “**internal fragmentation**”.

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 main memory.

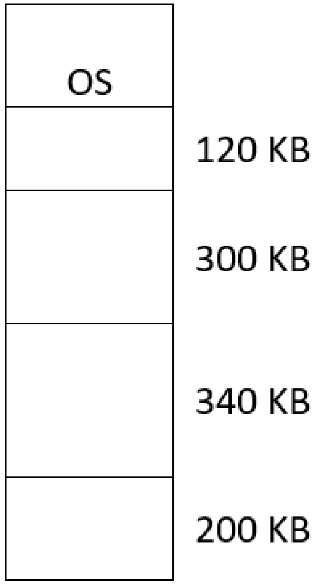
Un-equal size fixed partition technique

Before loading any program into main memory, the operating system divides the space in main memory into different size parts.

Only one program can be loaded into each partition of main memory.

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

Main Memory



OS

120 KB

300 KB

340 KB

200 KB

Advantage

Internal Fragmentation is less when compared with equal size fixed partition technique.

Disadvantages

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

2) Wastage of space inside the partition when the size of program is less than the size of partition.

3) 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 main memory.

Dynamic partition technique

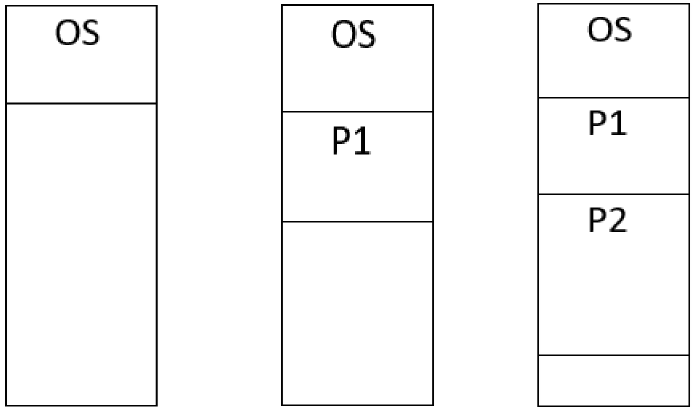
Operating system divides the space in main memory into parts at the time of loading programs into main memory.

Initially, main memory is divided into two parts.

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

After loading a program say P1, the space in main memory is divided into three parts.

After loading another program say P2, the space in main memory is divided into four parts as shown in below figure.



Advantage

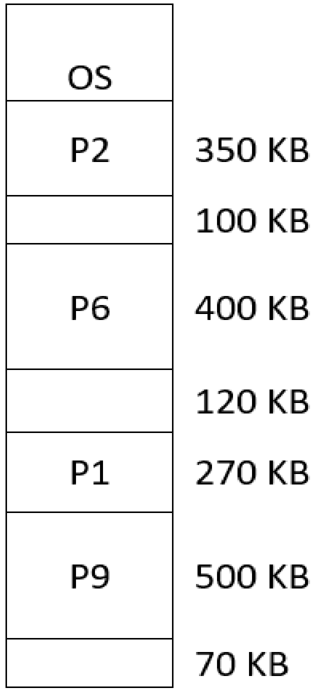
There is no wastage of memory inside partitions. So, no internal fragmentation.

Disadvantages

Some space is wasted outside of the partitions.

Wastage of space outside the partitions is called ‘External Fragmentation’.

Main Memory



Consider the above position of main memory which is the result of loading some programs into main memory and removing the completed programs from main memory.

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 main memory as there is no free part with size greater than or equal to 250KB in the main memory.

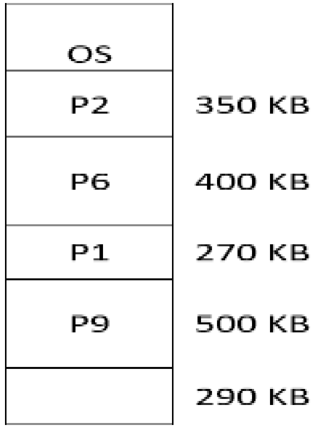
To avoid external fragmentation, **compaction** technique can be used.

Compaction

Compaction technique moves all free spaces in the main memory together.

After applying compaction technique to the above position of main memory, the position of main memory is as shown below

Main Memory



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

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

Placement Algorithms

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

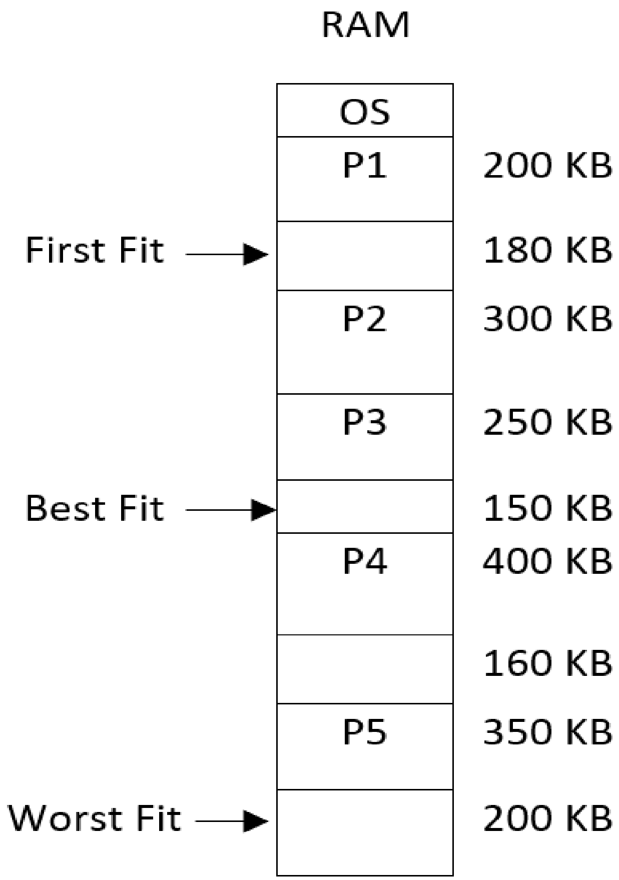
First fit algorithm scans the main memory either from the from starting or from the position where recent placement happened and selects the first free partition whose size is greater than or equal to the size of program to be loaded.

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

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

Consider the following position of main memory.

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



RAM

	OS	
	P1	200 KB
First Fit →		180 KB
	P2	300 KB
	P3	250 KB
Best Fit →		150 KB
	P4	400 KB
		160 KB
	P5	350 KB
Worst Fit →		200 KB

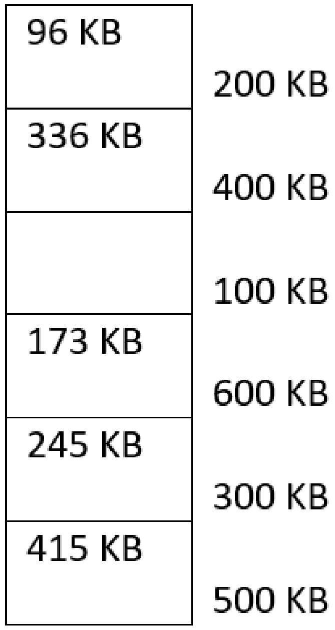
P10 → 150 KB

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

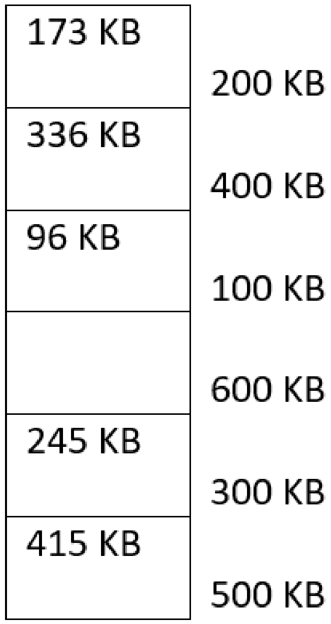
ii) Best Fit algorithm

First Fit



96 KB	200 KB
336 KB	400 KB
	100 KB
173 KB	600 KB
245 KB	300 KB
415 KB	500 KB

Best Fit



173 KB	200 KB
336 KB	400 KB
96 KB	100 KB
	600 KB
245 KB	300 KB
415 KB	500 KB

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

Total internal fragmentation with Best 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?

Paging Technique

Before loading any program into main memory, the space in main memory is divided into equal size frames.

To load a program into main memory, the program is divided into equal size pages.

Size of page is equal to size of frame.

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

The pages of program are loaded into free frames of main memory.

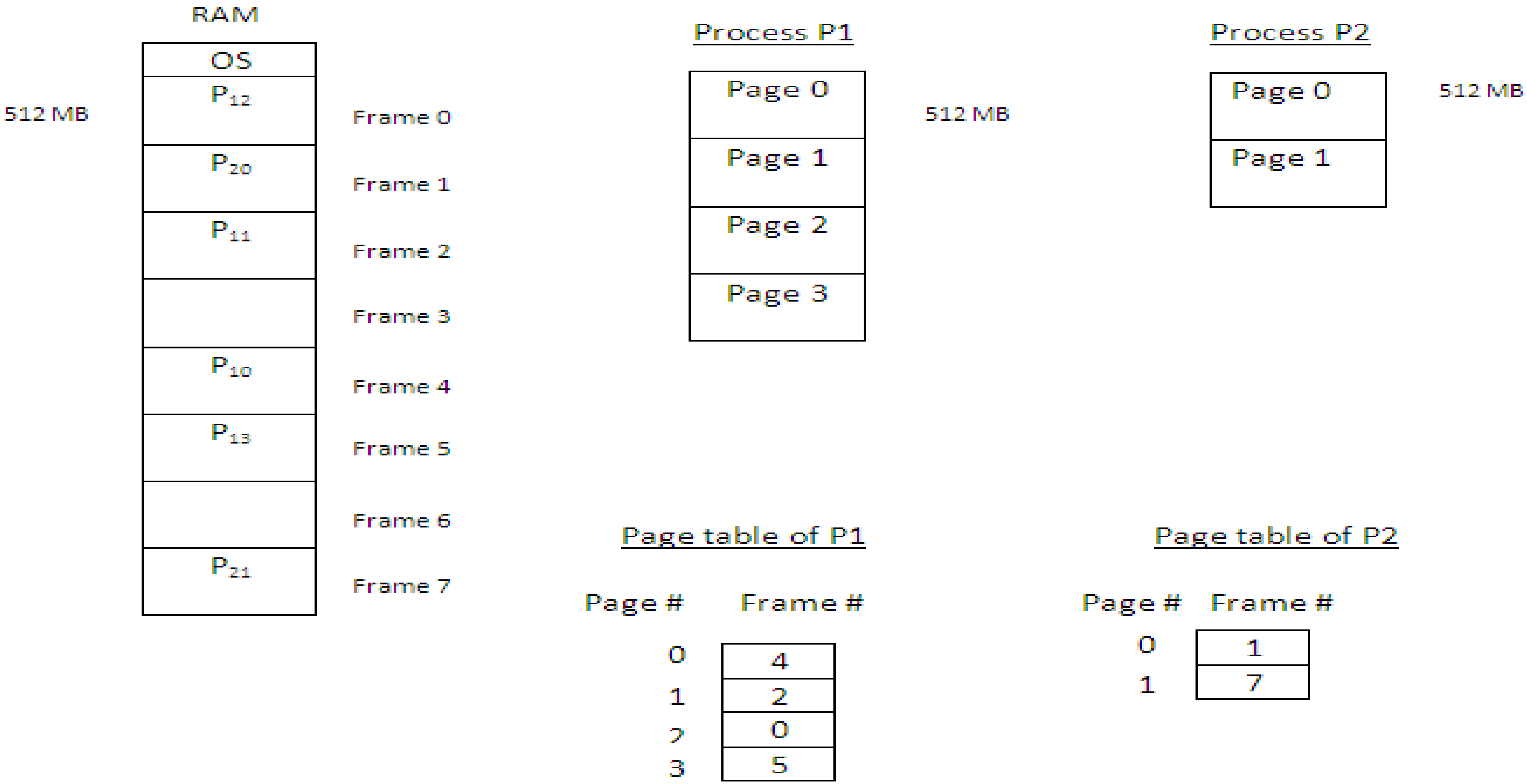
After loading all pages of program, a page table is created for the program.

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

The entries of page table are indexed with page numbers.

The entries of page table are filled with the frame numbers in which the pages of the program are loaded.

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



RAM

OS

512 MB

P_{12}	Frame 0
P_{20}	Frame 1
P_{11}	Frame 2
	Frame 3
P_{10}	Frame 4
P_{13}	Frame 5
	Frame 6
P_{21}	Frame 7

Process P1

Page 0

Page 1

Page 2

Page 3

512 MB

Process P2

Page 0

Page 1

512 MB

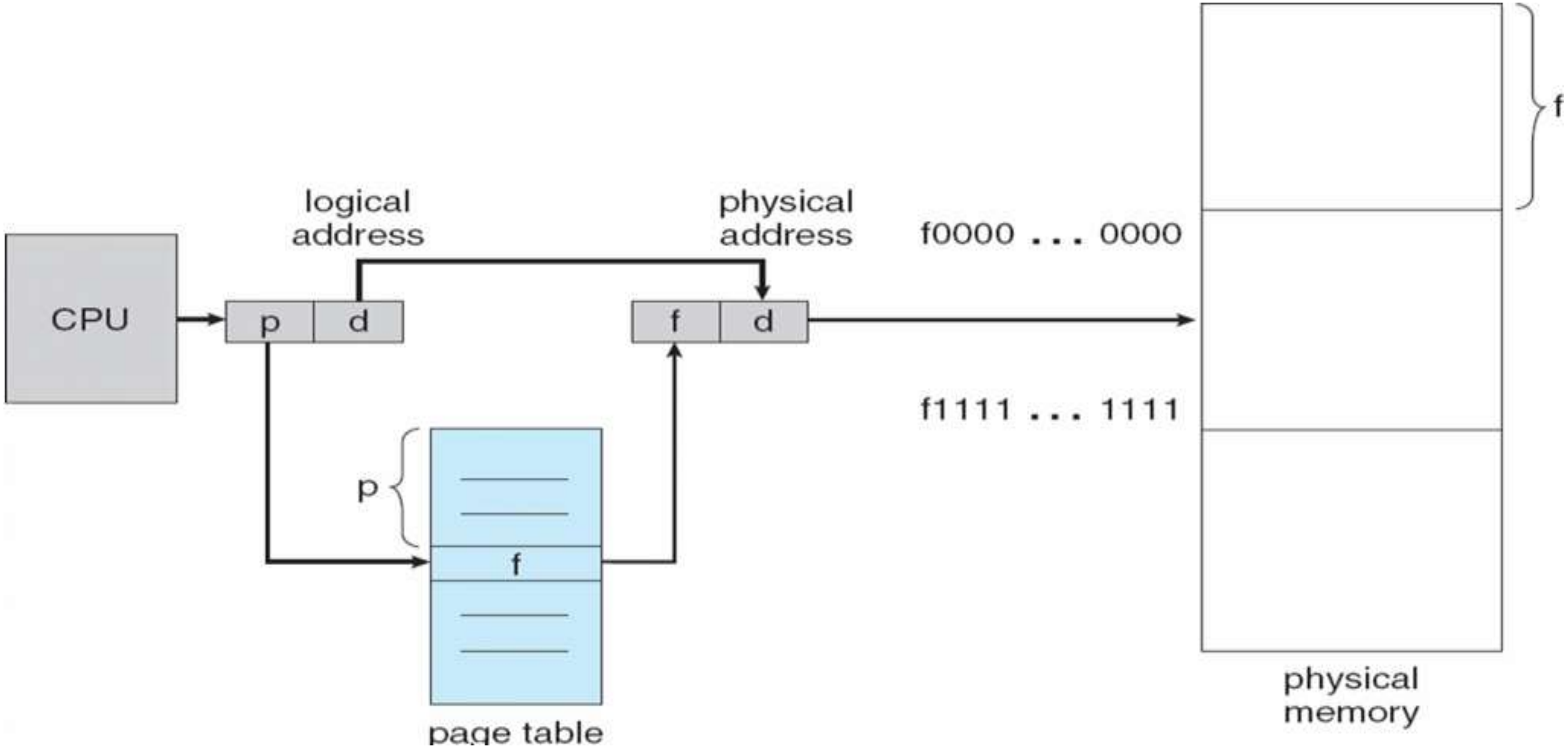
Page table of P1

Page #	Frame #
0	4
1	2
2	0
3	5

Page table of P2

Page #	Frame #
0	1
1	7

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



To execute a statement of the program loaded into main memory using the paging technique, the CPU generates the logical address of that statement.

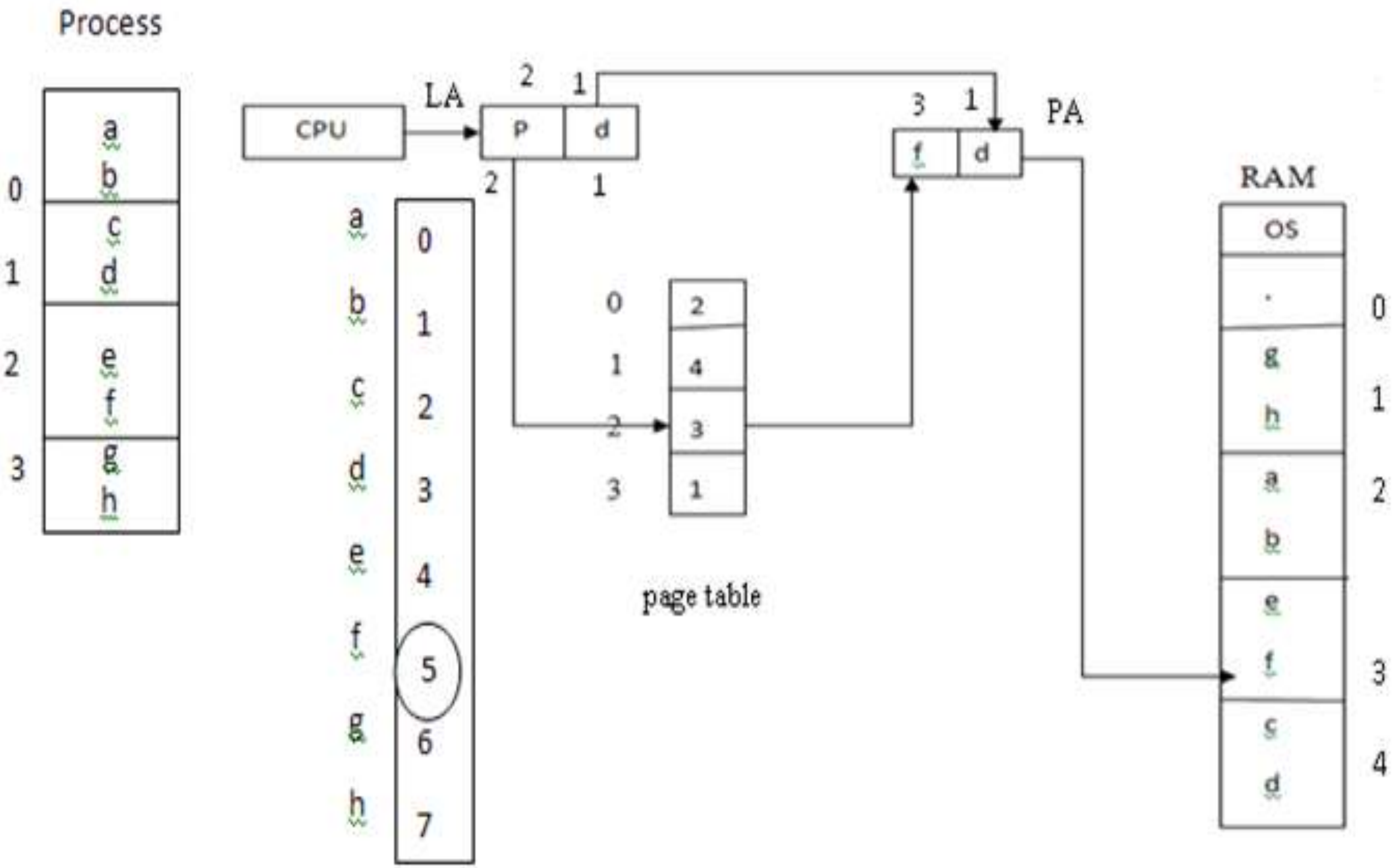
This logical address consists of two parts: page number (p) and offset (d).

The page number is used as an index into the page table of the program to find the corresponding frame number.

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

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

The offset identifies the statement inside the page or frame.


Process
0
a
b
1
c
d
2
e
f
3
g
h
CPU
LA
2
1
P
d
2
1
a 0
b 1
c 2
d 3
e 4
f 5
g 6
h 7
0 2
1 4
2 3
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 ($256=2^8$) + 6 bits ($64=2^6$) = 14 bits

Size of physical address = size of frame number field + size of offset field = 12 bits ($4096=2^{12}$) + 6 bits ($64=2^6$) = 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 2^{10} 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 2^{10} 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.

Ex: Calculate the size of memory if its address consists of 22 bits and the memory is 2-byte addressable.

Ex: Calculate the number of bits required in the address for memory having size of 16 GB. Assume the memory is 4-byte addressable.

Ex: Consider a system with byte-addressable memory, 32 bit logical addresses, 4 kilobyte page size and page table entries of 4 bytes each. Calculate the size of the page table in the system in megabytes.

Ex: Consider a machine with 64 MB physical memory and a 32 bit virtual address space. If the page size is 4 KB, what is the approximate size of the page table?

Ex: In a virtual memory system, size of virtual address is 32-bit, size of physical address is 30-bit, page size is 4 Kbyte and size of each page table entry is 32-bit. The main memory is byte addressable. Which one of the following is the maximum number of bits that can be used for storing protection and other information in each page table entry?

Translation Look-aside Buffer (TLB)

The operating system stores the page table of a process in the registers of computer system when the number of entries in the page table is less. Otherwise, stores the page table in the main memory.

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

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

To get a statement from the main memory, 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 main memory 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 main memory.

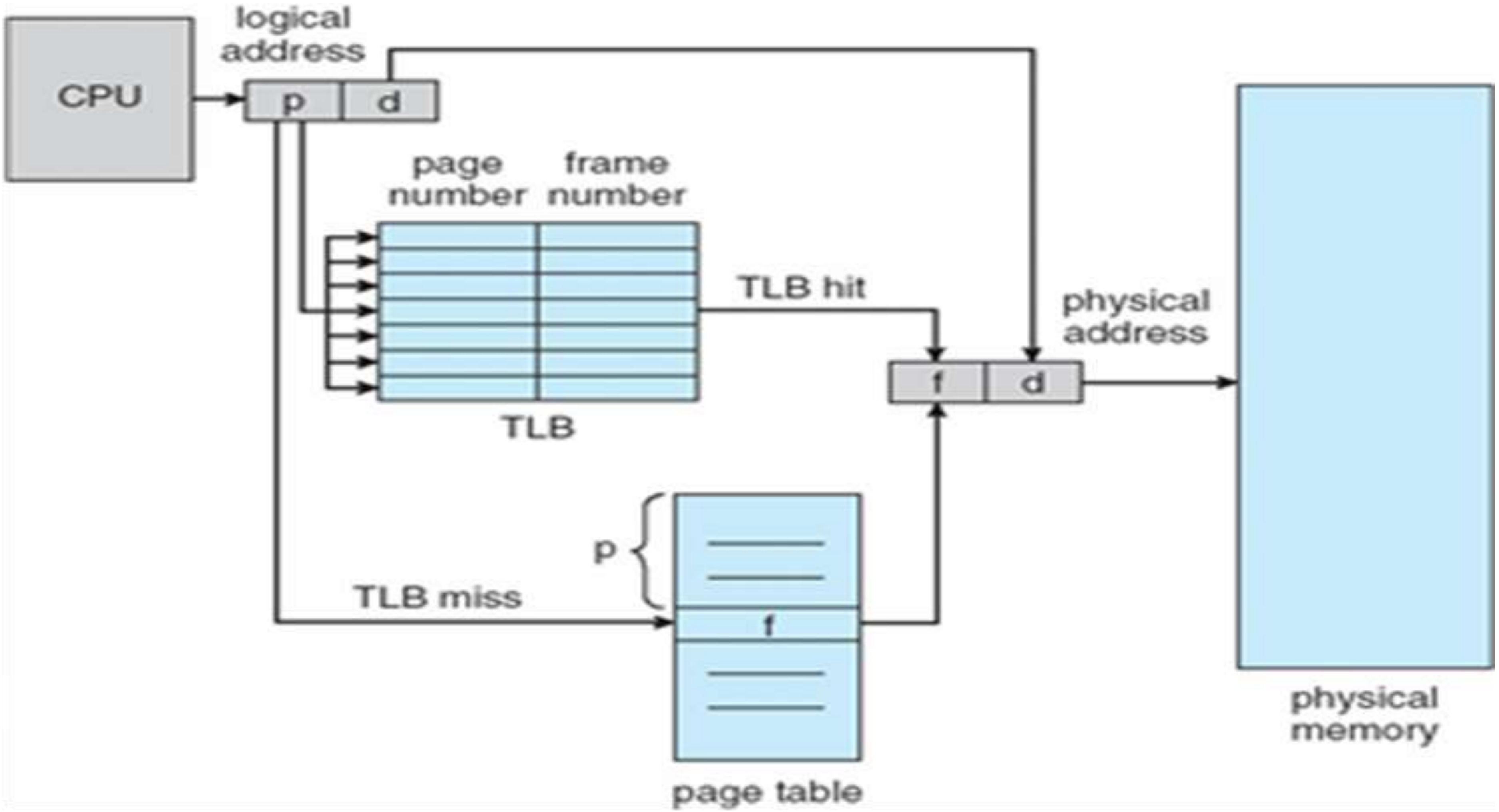
To get a statement of the process from main memory with one memory access or to reduce the time for access, information about frequently executed pages is stored in the translation look-aside buffer (TLB).

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 logical address of a statement, 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 memory access time = probability of finding the page in the TLB x time to access the statement from the main memory + probability of not finding the page in the TLB x time to access the statement from the main 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

Segmentation is one of techniques used to load programs into main memory.

Operating system divides the space in main memory at the time of loading programs into main memory.

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

The operating system then loads the segments of program into main memory wherever free space is available.

The operating system then creates a segment table for the program.

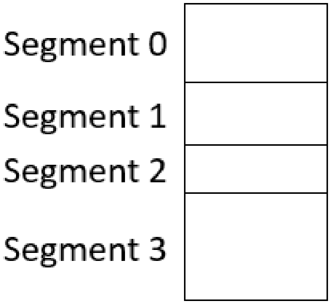
The number of entries in the segment table is equal to the number of segments in the program.

The entries of segment table are indexed with segment numbers.

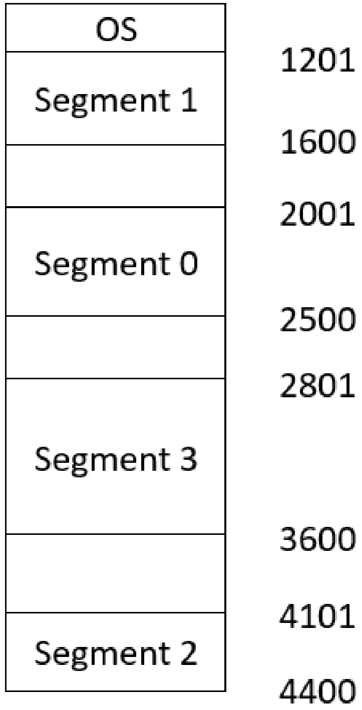
Each entry of the segment table contains a base which indicates the starting address of the segment in the main memory, and a limit/length which indicates the number of statements in that segment.

The following figure depicts how a program say P1 is loaded into main memory using segmentation technique.

Program-P1



Main Memory



Segment Table

S#	Limit	Base
0	500	2001
1	400	1600
2	300	4101
3	800	2801

The program P1 has 2000 statements and is divided into four segments.

The number of statements in the four segments are 500, 400, 300 and 800 respectively.

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

The segment table contains four entries.

Logical address to physical address mapping in Segmentation technique

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

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

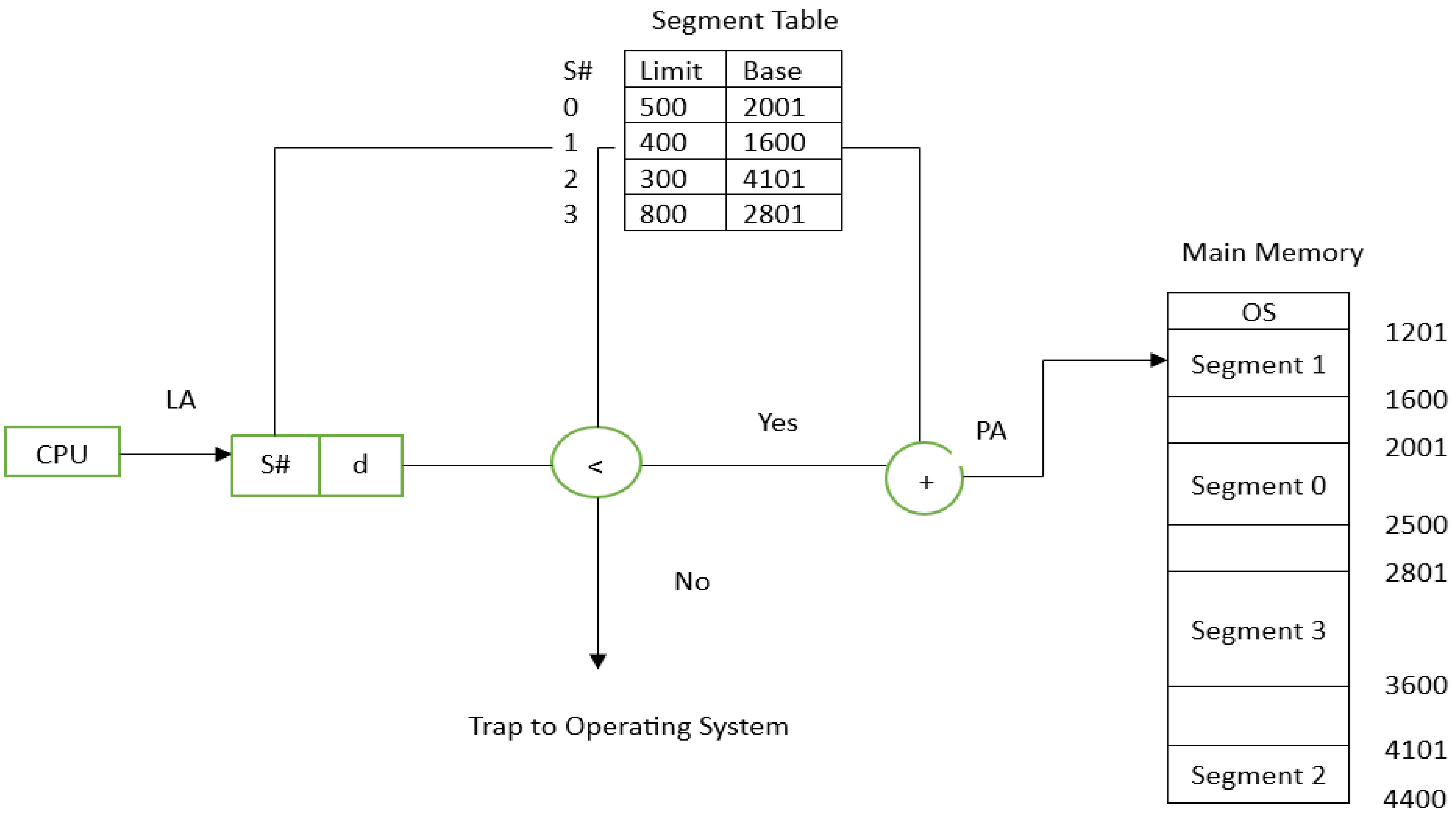
The segment number is used as an index into the segment table to identify the entry in the segment table.

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.

If the offset value is greater then the limit value, then an error is reported.

The following diagram depicts this mapping procedure.



Segment Table

S#	Limit	Base
0	500	2001
1	400	1600
2	300	4101
3	800	2801

Ex: 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)