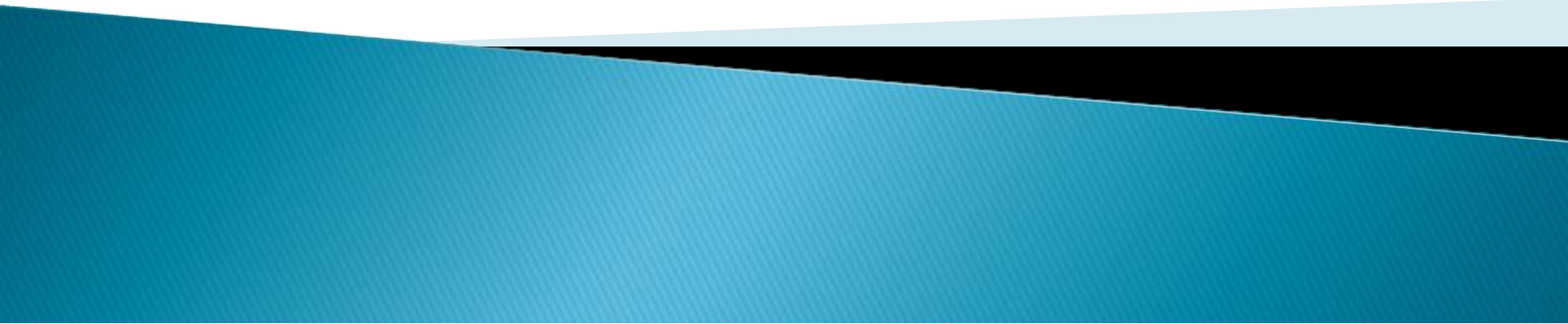


Memory Management



Introduction

- ▶ Main memory is divided into 2 parts :one for the operating system and the other part for user programs.
- ▶ In uniprogramming, there is OS in one part and the other part for the program currently being executed.
- ▶ In a multi programming system, the user part of the memory must be further subdivided to accommodate multiple processes.
- ▶ Degree of multiprogramming means the maximum number of processes in the main memory.
- ▶ The memory management function of the OS:
 1. Keeps track of which parts of memory are free and which are allocated.
 2. It determines how memory is allocated among competing processes, deciding who gets memory, and how much they are allowed.
 3. When memory is allocated it determines which memory locations will be assigned.
 4. It tracks when memory is freed or *unallocated* and updates the status.

Memory Management Techniques

The basic memory management function of the OS is to bring processes into main memory for execution by the processor.

1. **Memory Partitioning :**

- Obsolete technique

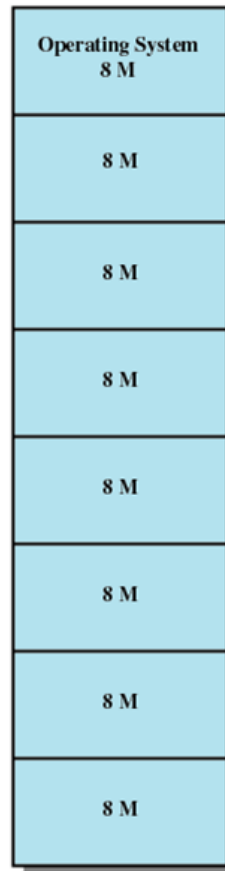
a) **Fixed Partitioning(Multiprogramming with fixed no. of tasks–MFT)**

- The user section of the MM is partitioned into regions of fixed sizes.
- Two variations in fixed partitioning:

Equal Size Partitions:

- The user memory is divided into partitions of fixed sizes.
- Any process whose size is less than or equal to the partition size can be loaded into any available partition.
- If all partitions are full , the OS can swap a process out of any of the partitions and load in any other process.

Memory Management Techniques



(a) Equal-size partitions

Fixed Partitioning of
a 64 MB memory

Memory Management Techniques

- ▶ Placement Algorithm:

As long as there are available partitions a process can be loaded in any available partition.

Because all partitions are of equal size it does not matter which partition is used.

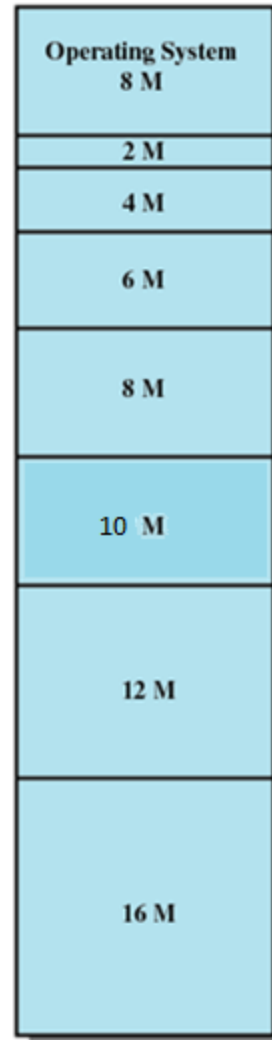
- ▶ Drawbacks of fixed partitioning :

1. A program may be too big to fit into a partition.
2. Inefficient utilization of MM. A program can be small but occupies an entire partition.
Eg: A program of 2 MB occupies an 8 MB partition.
Internal fragmentation – 6 MB
The wasted space internal to a partition due to the fact that the block of data loaded is smaller than the partition is called **internal fragmentation**.

Solution : Use unequal sized partitions



Memory Management Techniques



Unequal size partitions

Memory Management Techniques

► Placement algorithm with Unequal sized partitions :

Two possible ways :

1. Unequal size partitions, use of multiple queues:

- assign each process to the smallest partition within which it will fit.
- **a queue exists for each partition size.**
- tries to minimize internal fragmentation.
- problem: some queues might be empty while some might be loaded.

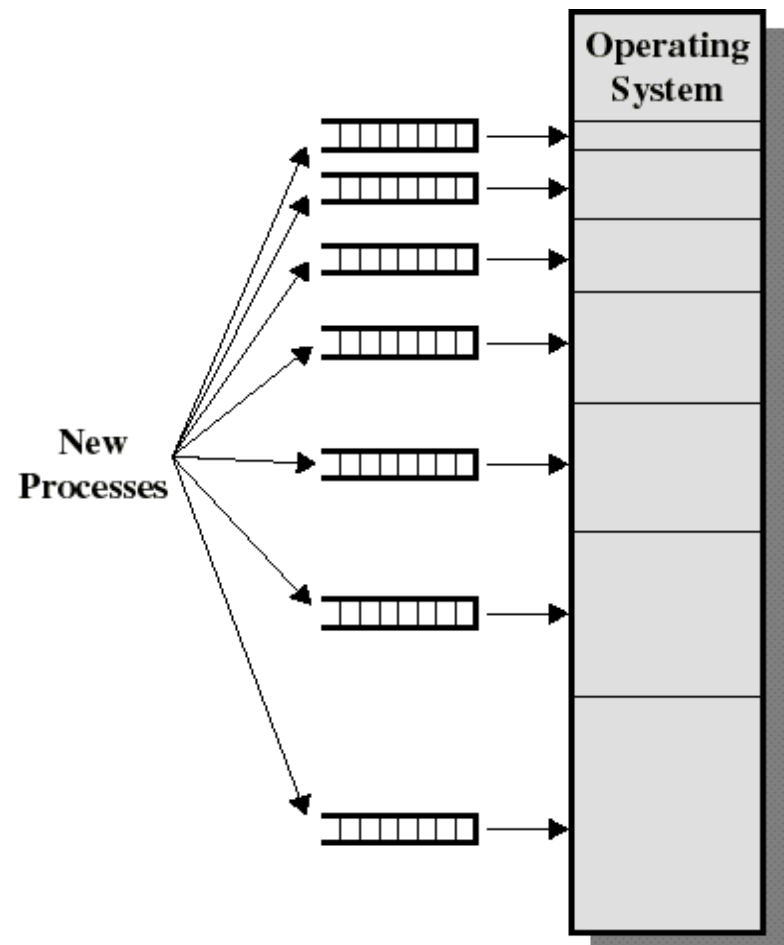
Eg: There might be no processes with a size between 12 and 16 M, hence that partition remains unused.

For eg: Processes of size 11M,1M,9M

11M –Queue of partition of 12 M

1 M– Queue of partition of 2 M

9 M– Queue of partition of 10 M



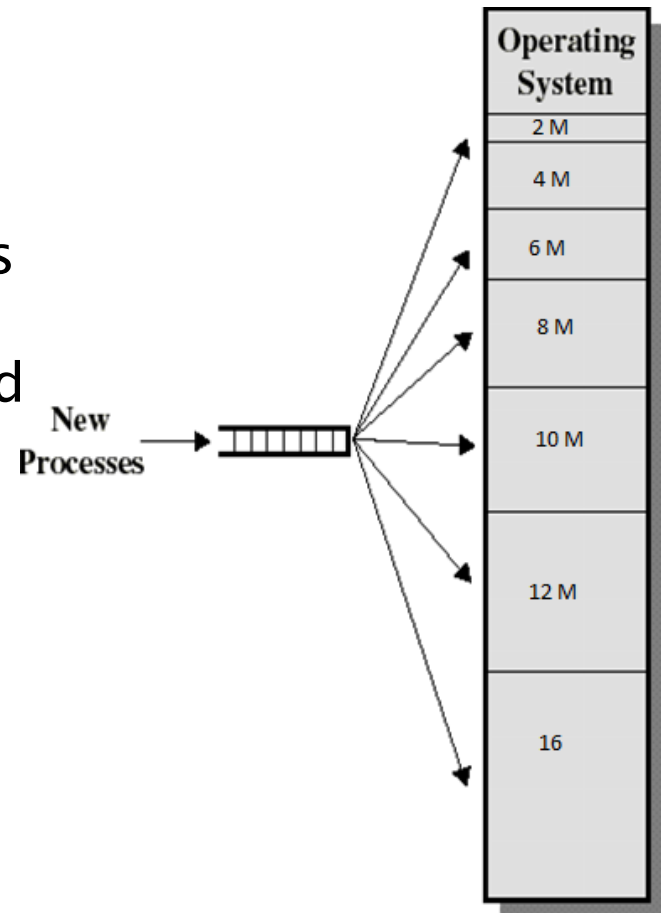
Memory Management Techniques

2.

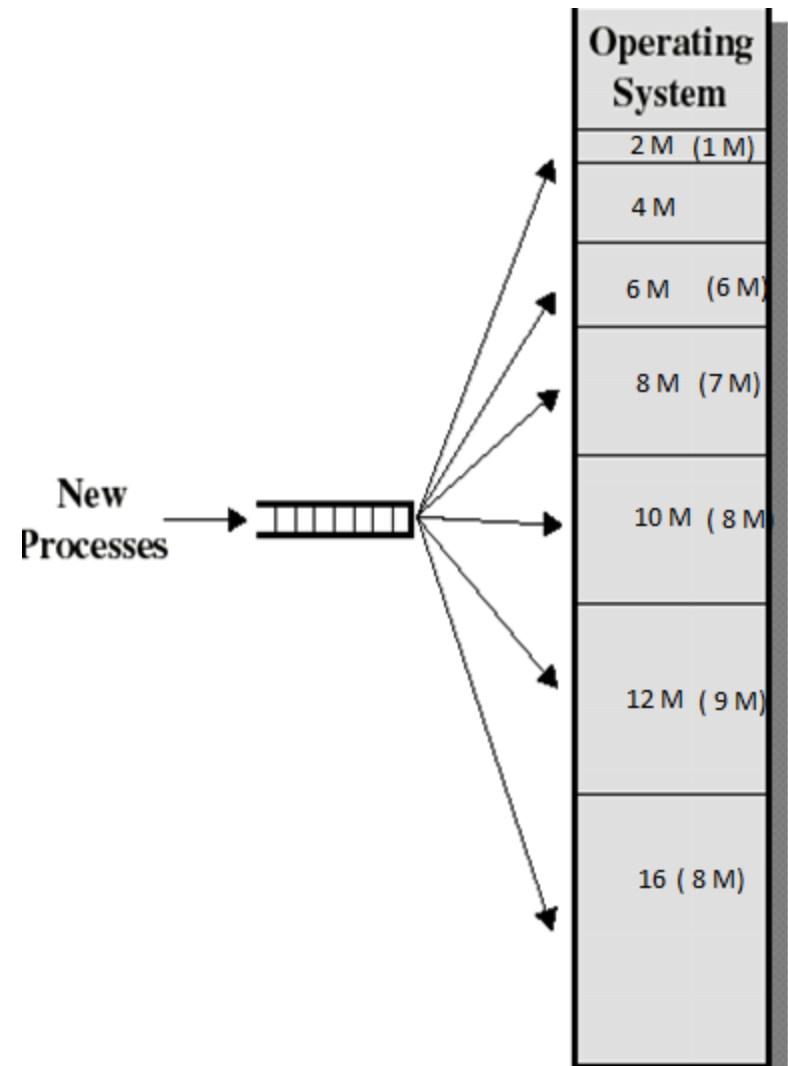
- ▶ Unequal-size partitions, use of a single queue:
 - when its time to load a process into memory, the smallest available partition that will hold the process is selected.
 - increases the level of multiprogramming at the expense of internal fragmentation.

Eg : Queue contains

7M, 6M , 8M, 1M , 9M, 8M, 5M



- ▶ Eg 7M, 6M , 8M,1M ,9M, 8M, 5M



Memory Management Techniques

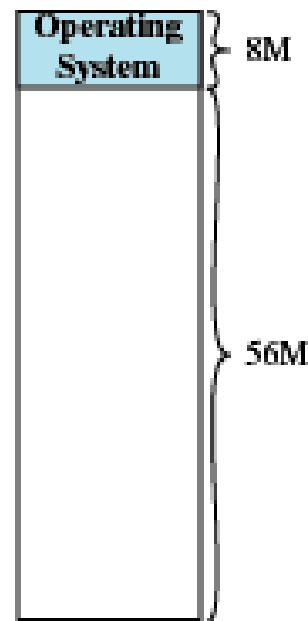
Disadvantage of MFT :

- ▶ The number of active processes is limited by the system
i.e. limited by the pre-determined number of partitions
- ▶ A large number of very small process will not use the space efficiently giving rise to internal fragmentation.
 - In either fixed or variable length partition methods

Memory Management Techniques

2. Dynamic Partitioning (Multiprogramming with variable no. of tasks–MVT)

- ▶ Developed to overcome the difficulties with fixed Partitioning.
- ▶ Partitions are of variable length and number dictated by the size of the process.
- ▶ A process is allocated exactly as much memory as required.
- ▶ Eg: 64 MB of MM. Initially MM is empty except for the OS

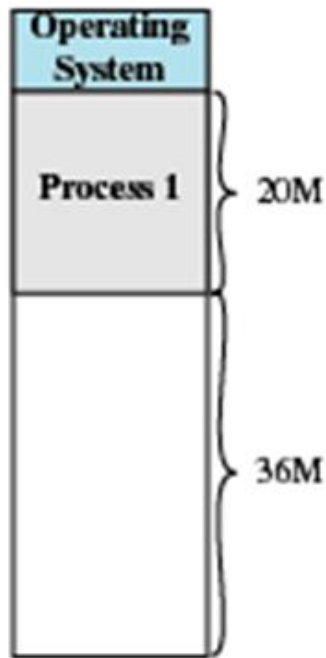


(a)

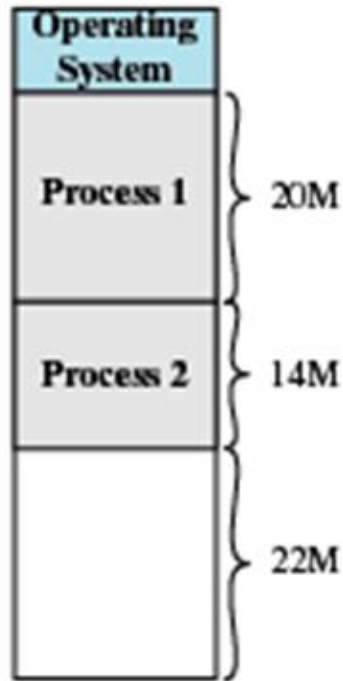
Memory Management Techniques

Eg Process 1 arrives of size 20 M.

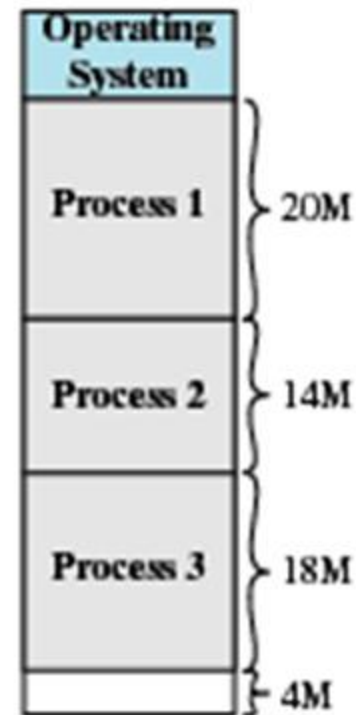
- Next process 2 arrives of size 14 M.
- Next process 3 arrives of size 18 M
- Next process 4 arrives of size 8 M



(b)



(c)

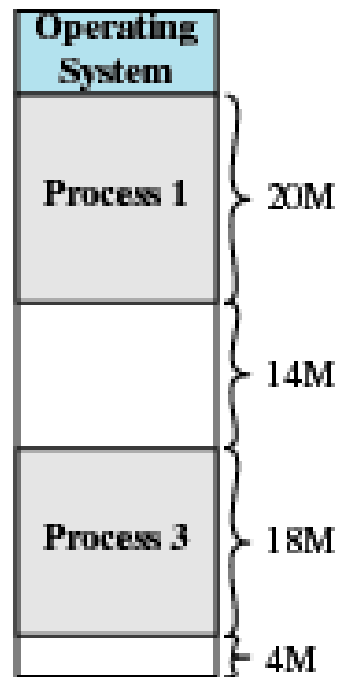


(d)

Unused memory at the end of 4 M that is too small for process 4 of 8 M

Memory Management Techniques

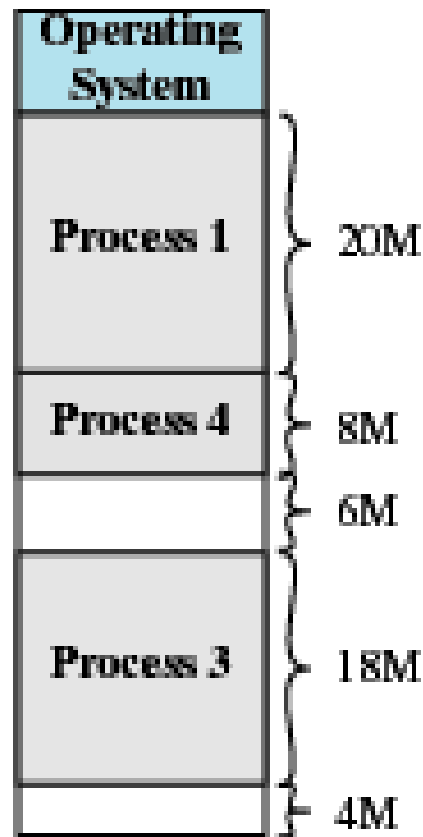
- At some point later assume process 2 gets blocked.
- The OS swaps out process2 which leaves sufficient room for process4



(e)

Memory Management Techniques

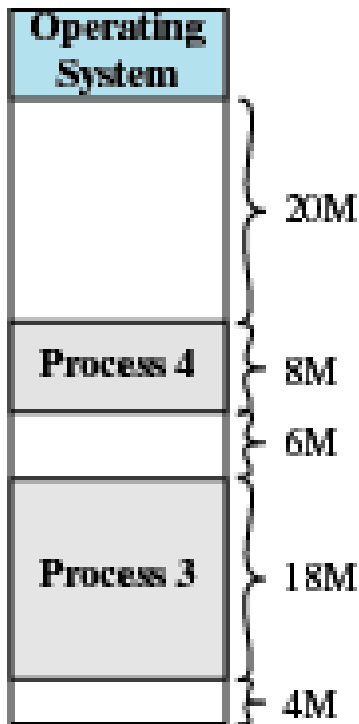
Process 4 is loaded creating another small hole of 6 M



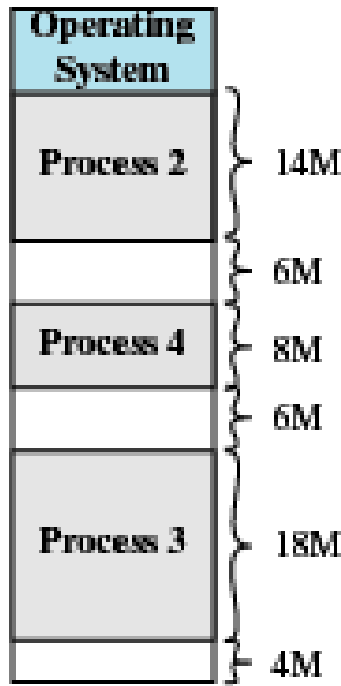
(f)

Memory Management Techniques

- Process 1 finishes execution on CPU.
- Hence OS swaps out Process 1
- Assume Process 2 gets unblocked.
- OS swaps in Process 2 creating another hole of 6 M



(g)



(h)

Memory Management Techniques

- ▶ MVT eventually leads to a situation in which there are a lot of small holes in memory.
- ▶ This is called external fragmentation.

There are really two types of fragmentation:

1. Internal Fragmentation –

Allocated memory may be slightly larger than requested memory; this size difference is memory internal to a partition, but not being used.

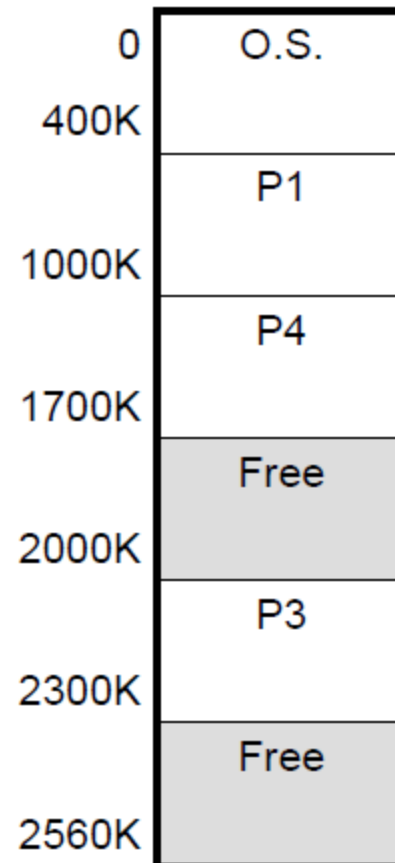
2. External fragmentation – total memory space exists to satisfy a request, but it is not contiguous; storage is fragmented into a large number of small holes.

Note : MVT suffers from external fragmentation and not internal fragmentation



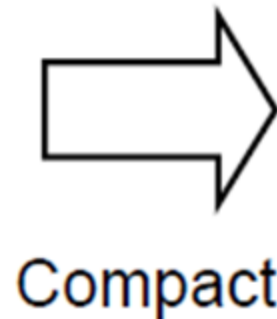
Example: We have a total external fragmentation of $(300+260)=560\text{KB}$.

- If P5 is 500KB, then this space would be large enough to run P5. But the space is not contiguous.



One solution to the problem of external fragmentation is compaction. The goal is to shuffle memory contents to place all free memory together in one large block.

OS
P1
<free> 20 KB
P2
<free> 7 KB
P3
<free> 10 KB



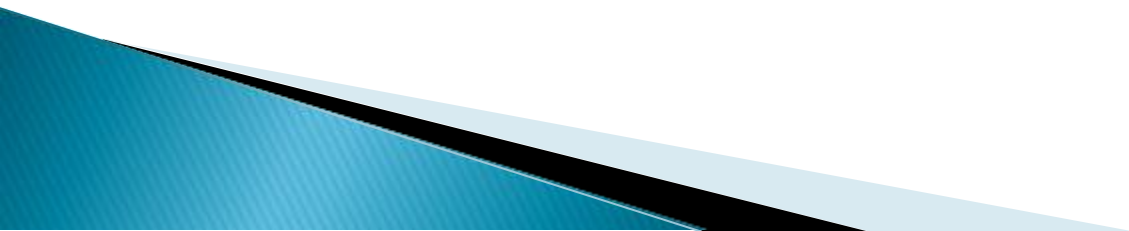
OS
P1
P2
P3
<free> 37 KB

Memory Management Techniques

From time to time the OS shifts the processes so that they are contiguous and so that all the free memory is together in 1 block which may be sufficient to load an additional process.


Drawbacks of compaction:

- ▶ Time consuming due to relocation.
- ▶ Wastes the CPU time.



Memory Management Techniques

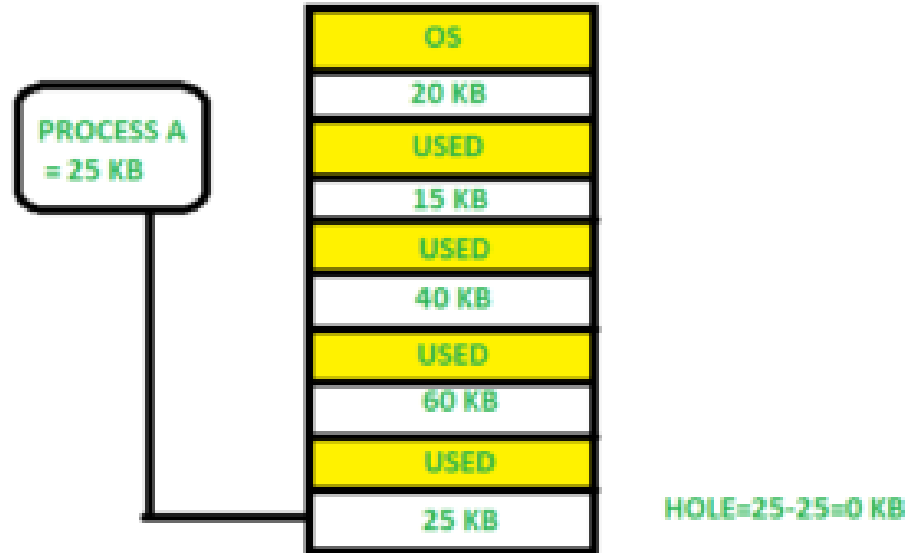
Dynamic Partitioning Placement Algorithm

- ▶ In **Partition Allocation**, when there is more than one partition freely available to accommodate a process's request, a partition must be selected. To choose a particular partition, a partition allocation method is needed.
 - ▶ i.e When a new process has to be brought in , the OS must decide free block to allocate.
 - ▶ Three placement algos for MVT :
 1. Best fit
 2. First fit
 3. Next fit
- 

Memory Management Techniques

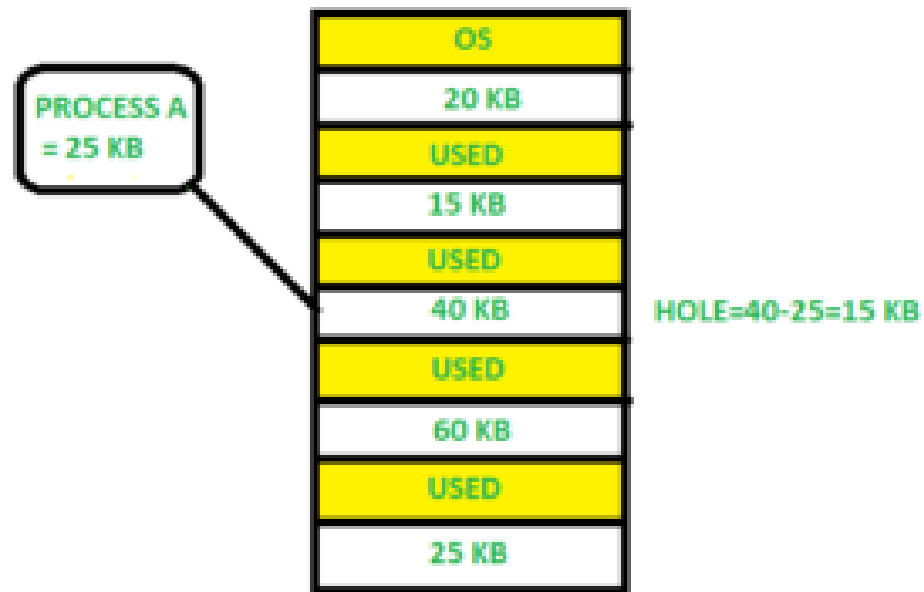
1. Best fit

Chooses the block that is closest in size to the request. It searches the entire list of holes to find the smallest hole whose size is greater than or equal to the size of the process.

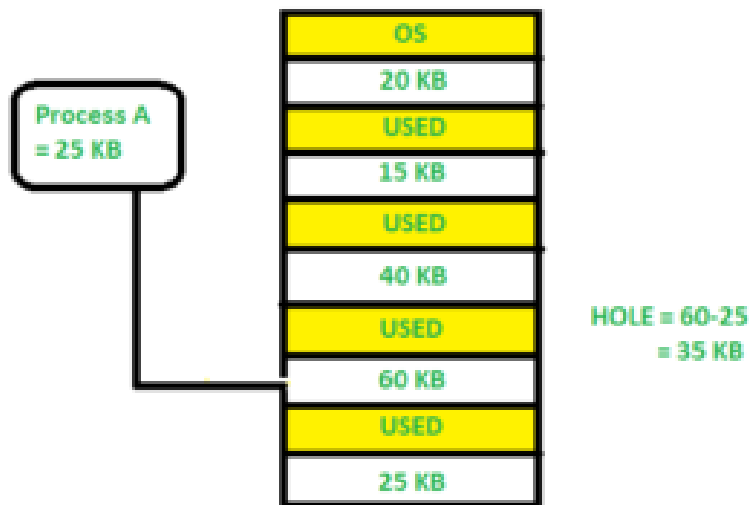


2.First fit

Scans memory from the beginning and chooses the first available block that is large enough



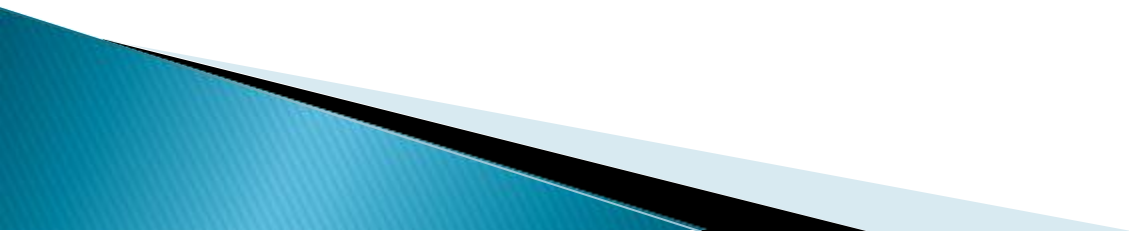
3. Worst Fit Allocate the process to the partition which is the largest sufficient among the freely available partitions available in the main memory. It is opposite to the best-fit algorithm. It searches the entire list of holes to find the largest hole and allocate it to process.



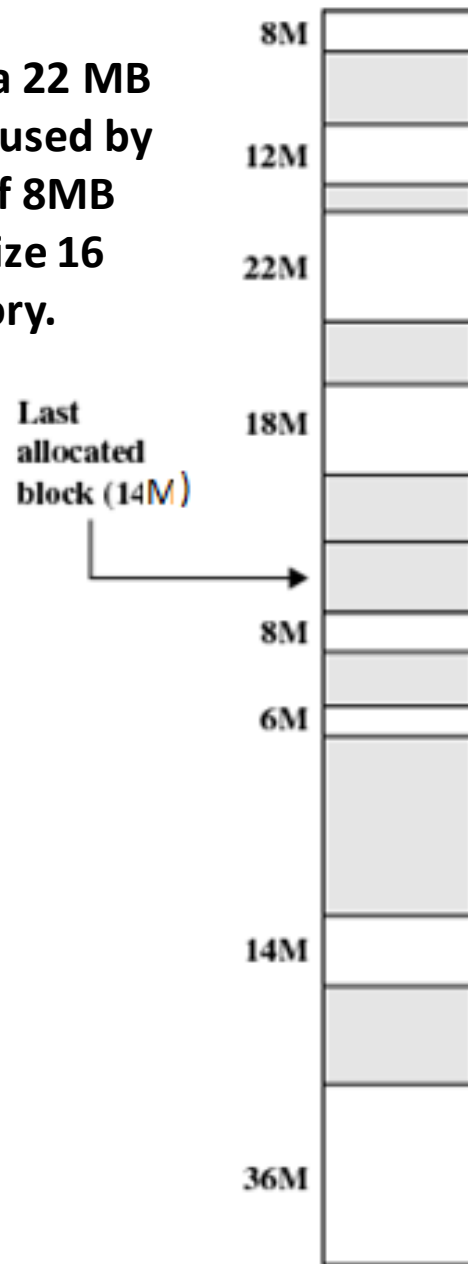
3. Next fit

Scans memory from the location of the last placement

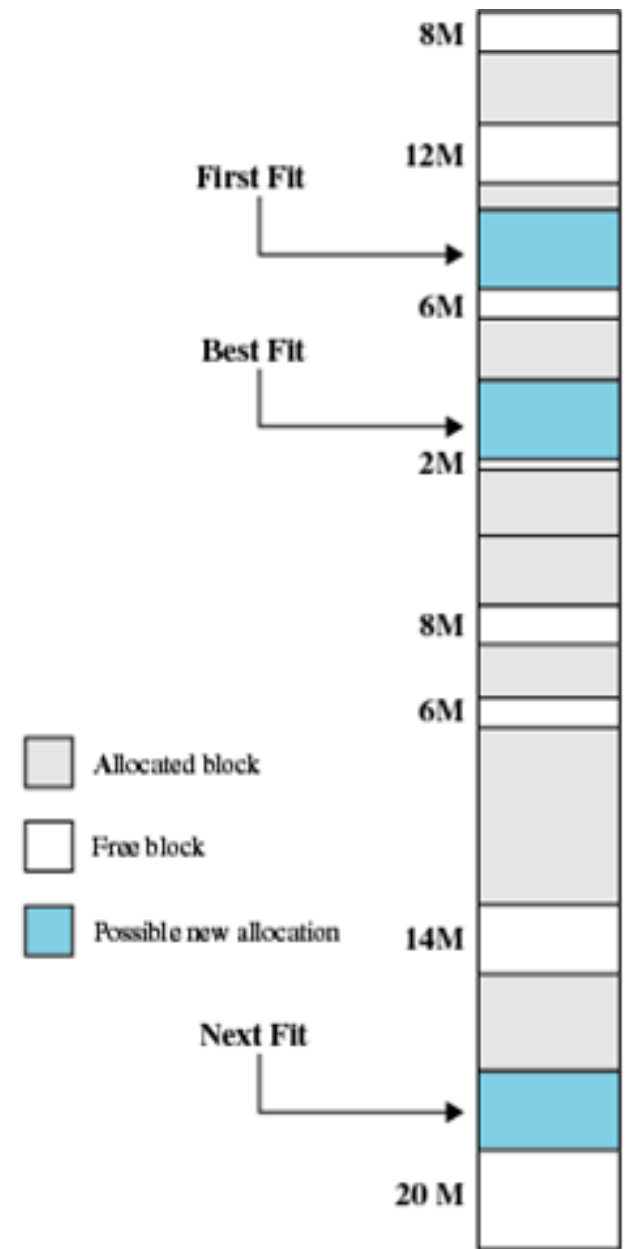
Next fit is similar to the first fit but it will search for the first sufficient partition from the last allocation point.



Last block that was used was a 22 MB block, from which 14 MB was used by the process returning a hole of 8MB
 Assuming now a program of size 16 MB had to be allocated memory.



(a) Before



(b) After

Memory Management Techniques

- ▶ Performance of the three approaches :

1. Best fit

- Worst performer overall
- Since smallest block is found for process, the smallest amount of fragmentation is left
- Memory compaction must be done more often

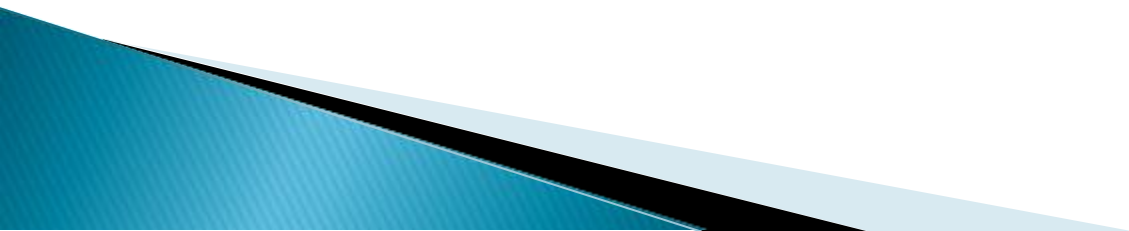
2. First fit

- Scans memory from the beginning and chooses the first available block that is large enough
- Fastest
- May have many process loaded in the front end of memory that must be searched over when trying to find a free block

Memory Management Techniques

3. Next fit

- Scans memory from the location of the last placement
- More often allocate a block of memory at the end of memory where the largest block is found

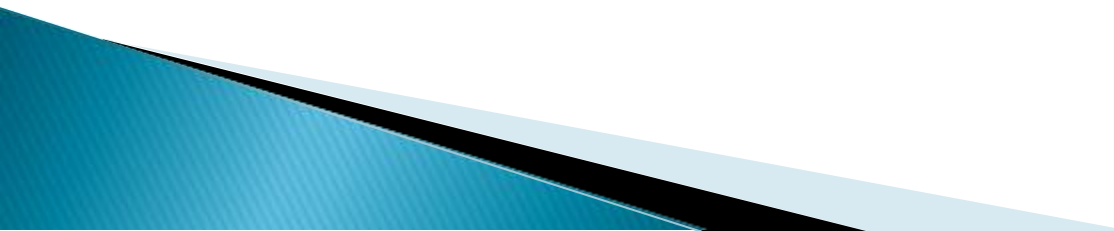


Virtual Memory

Issues in modern computer systems:

- ▶ Physical main memory is not as large as the size of the user programs.
- ▶ In a multiprogramming environment more than 1 programs are competing to be in MM.
- ▶ Size of each is larger than the size of MM.

Solution :


- ▶ Parts of the program currently required by the processor are brought into the MM and remaining stored on the secondary devices.
 - ▶ Movement of programs and data between secondary storage and MM is handled by the OS.
 - ▶ User does not need to be aware of limitations imposed by the available memory.
- 

Virtual Memory

Def: **Virtual memory**

- ▶ VM is a concept which enables a program to execute even when it's entire code and data are not present in memory.
- ▶ Permits execution of a program whose size exceeds the size of the MM.
- ▶ VM is a concept implemented through a techniques called **paging and segmentation**.

Paging

- ▶ Logical address : Location relative to the beginning of a program
 - ▶ Physical address : Actual locations in Main memory.
 - ▶ Cpu when executing a program is aware of only the logical addresses.
- 

Virtual Memory

- ▶ When a program needs to be accessed from the MM, MM should be presented with physical addresses.
- ▶ Memory Management Unit (MMU) converts the logical addresses into physical addresses.
- ▶ A memory management unit (MMU), is a computer hardware component responsible for converting the logical add. to physical add.

▶ In paging scheme --→

User programs are split into fixed sized blocks called pages.

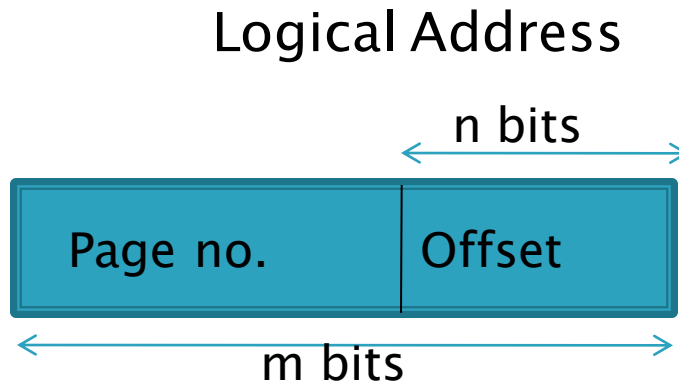
Physical memory is also divided into fixed size slots called page frames.

Page size ---→ 512 bytes, 1 K, 2K ,8K in length.

Allocation of memory to a program consists of finding a sufficient no. of unused frames for loading of pages of a program.

Each page requires one page frame.

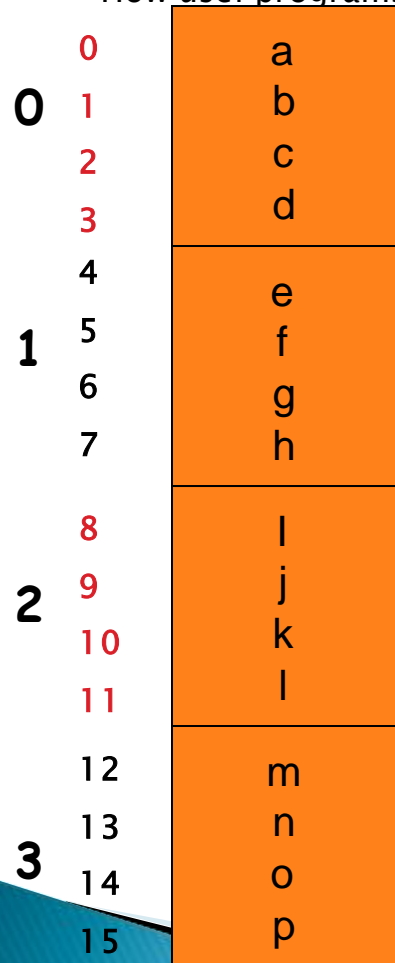
Paging



Lower order n bits designate the offset within the page.
Higher order $(m-n)$ bits designate the page no.

Paging

How user programs are mapped into physical memory?

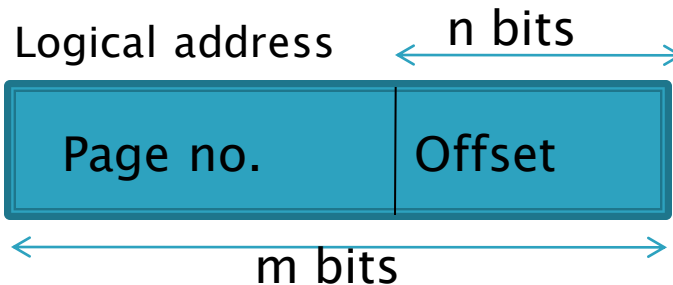


Let the logical memory of a program be 2^m bytes

Hence no. of bits in the logical address is m bits.

Here $m = 4$ bits

Size of a page be 2^n bytes.

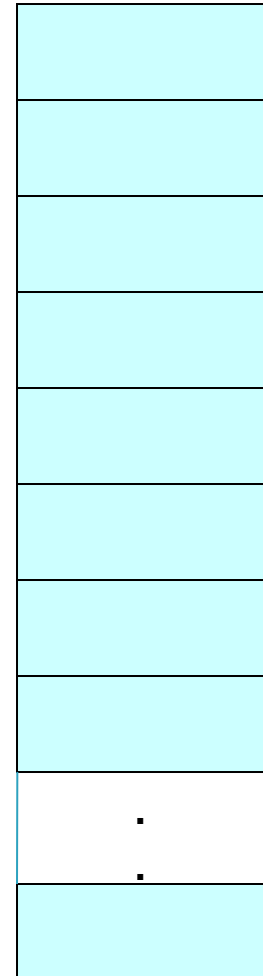


Frame 0

Frame 1

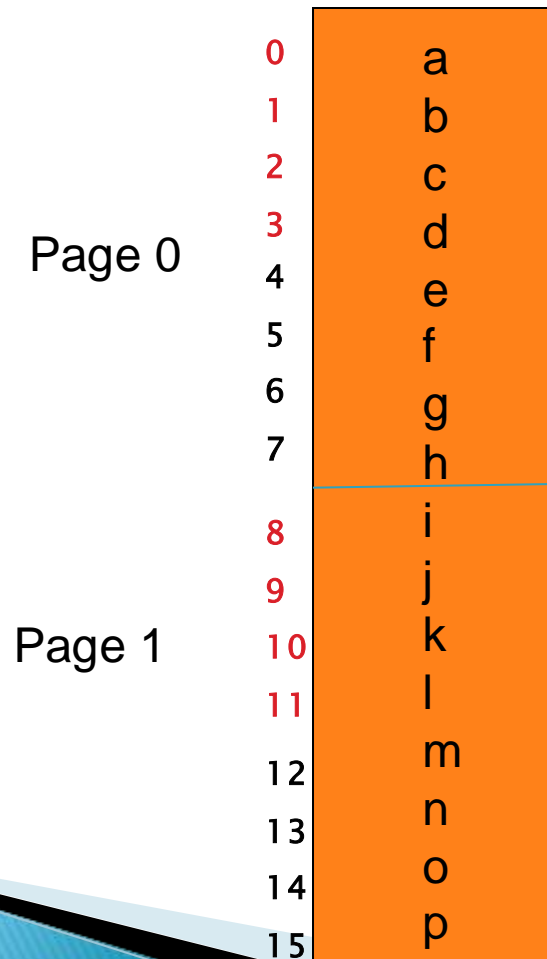
Frame 2

Frame 3

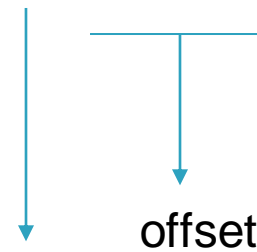


Logical Memory (user program)

- ▶ Eg: Page size 8 bytes



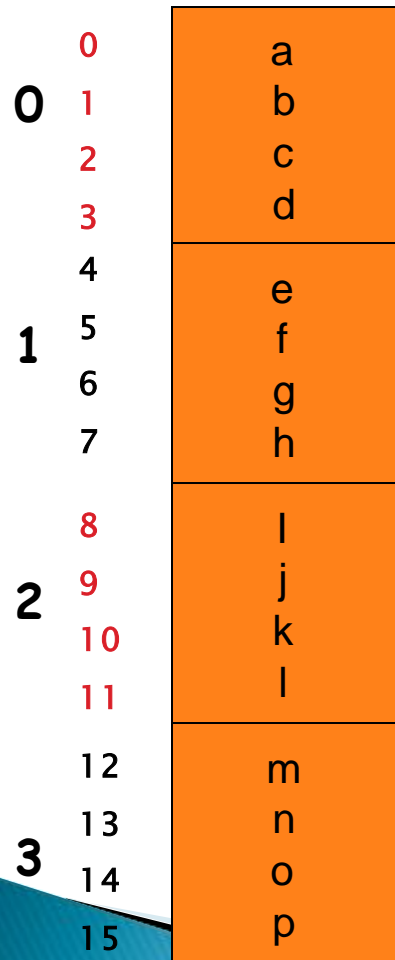
Logical address
0 1 1 1



Page no

Virtual Memory: Paging

Paging Example – 32-byte memory
with 4-byte pages



Logical Memory

0	5
1	6
2	1
3	2

Page Table

Gives information that a page
is in which page frame

Frame 0

1

2

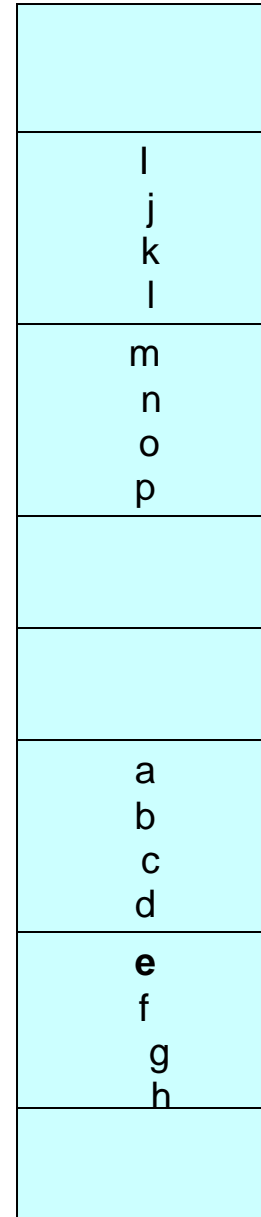
3

4

5

6

7



Physical Memory

Paging

Mapping from logical address to physical address→

Assume CPU wants to access data corresponding to logical

Address 14 :



Page no. Offset

1. Use page no. field (11) to index into the page table (PT)
2. Indexing the PT we find page 3 is in frame no. 2 of MM.
3. $2 \times 4(\text{page size}) = 8$
4. $8 \rightarrow$ base address of frame 2
5. Use the offset of the logical address to access the byte within the frame. Offset $\rightarrow 10(2)$
6. $8 + 2(\text{offset}) = 10$

Hence logical address 14 maps to physical address 10

Paging

- ▶ If a program is of size 2700 bytes and
- ▶ If 16-bit addresses are used, and the page size is 1K = 1024 bytes.
- ▶ Consider a relative address(logical address) 1502, which in binary form, is :

0 0 0 0 0 1 0 1 1 1 0 1 1 1 1 0

- ▶ Dividing the program into pages of size 1 K we have Logical address 1502 is located on logical page 1, at offset 478. $(p, d) = 1, 478$.

Logical address =
Page# = 1, Offset = 478

000001|0111011110

- Logical address consists of page no. and offset
Here no. of bits in address field is 16.

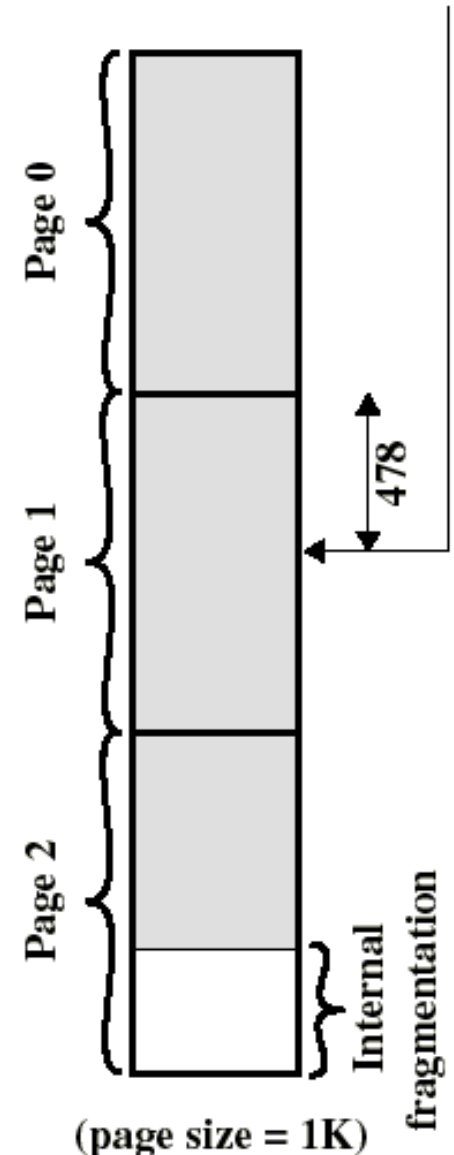
- No. of bits in the Offset depends on the page size.

- Since page size is 1024 bytes = 2^{10} bytes ..
10 bits are for offset and 6 bits for the page no.

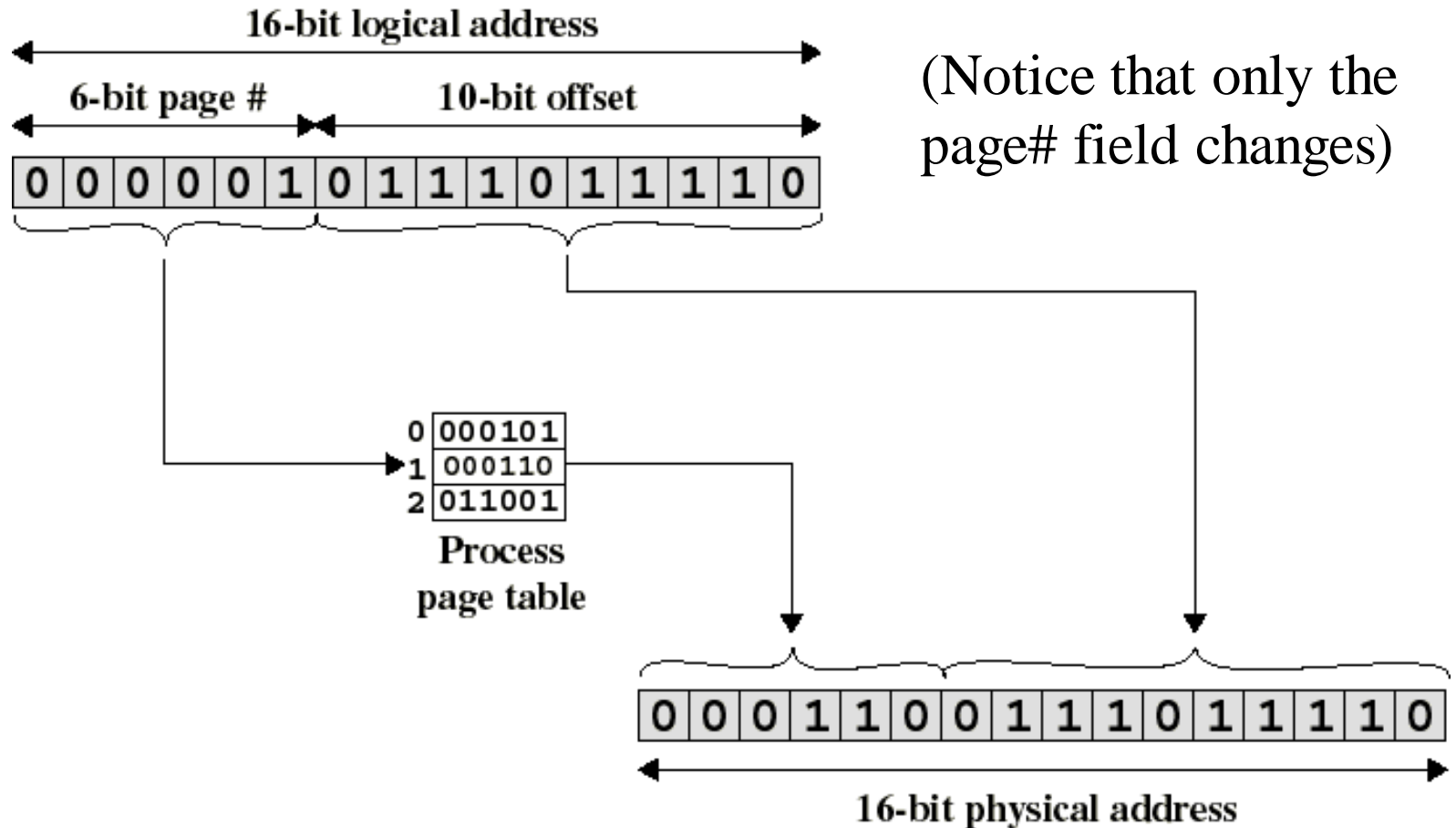
- Divide this into a six-bit page number field:

$000001_2 = 1$

and a 10-bit displacement field: $0111011110_2 = 478$



Paged Address Translation



- ▶ Eg 2
- ▶ Logical address 2100
- ▶ 000010 0000110100
page 2 offset=52

- ▶ Ex 3
- ▶ Let size of page=2048 bytes
- ▶ Program size= 72766 bytes
- ▶ 35 pages + 1086 bytes
- ▶ Internal fragmentation = $2048 - 1086 = 962$ bytes

Ex 4

- ▶ Consider a logical address space of 256 pages with a 4-KB page size, mapped onto a physical memory of 64 frames.

How many bits are required in the logical address?

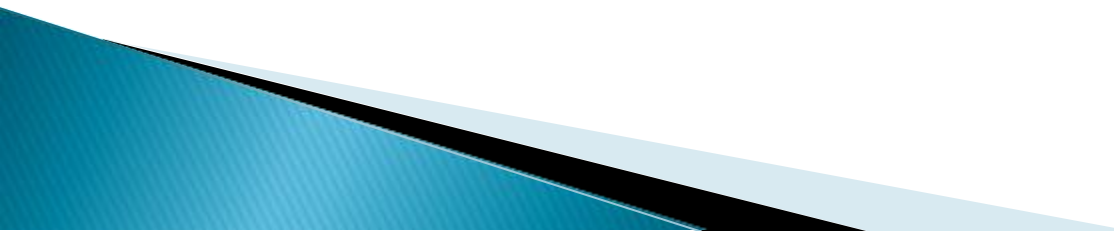
256 pages – 8 bit page no

4 KB page size – $2^{12} \times 1024 = 2^{12}$ bytes

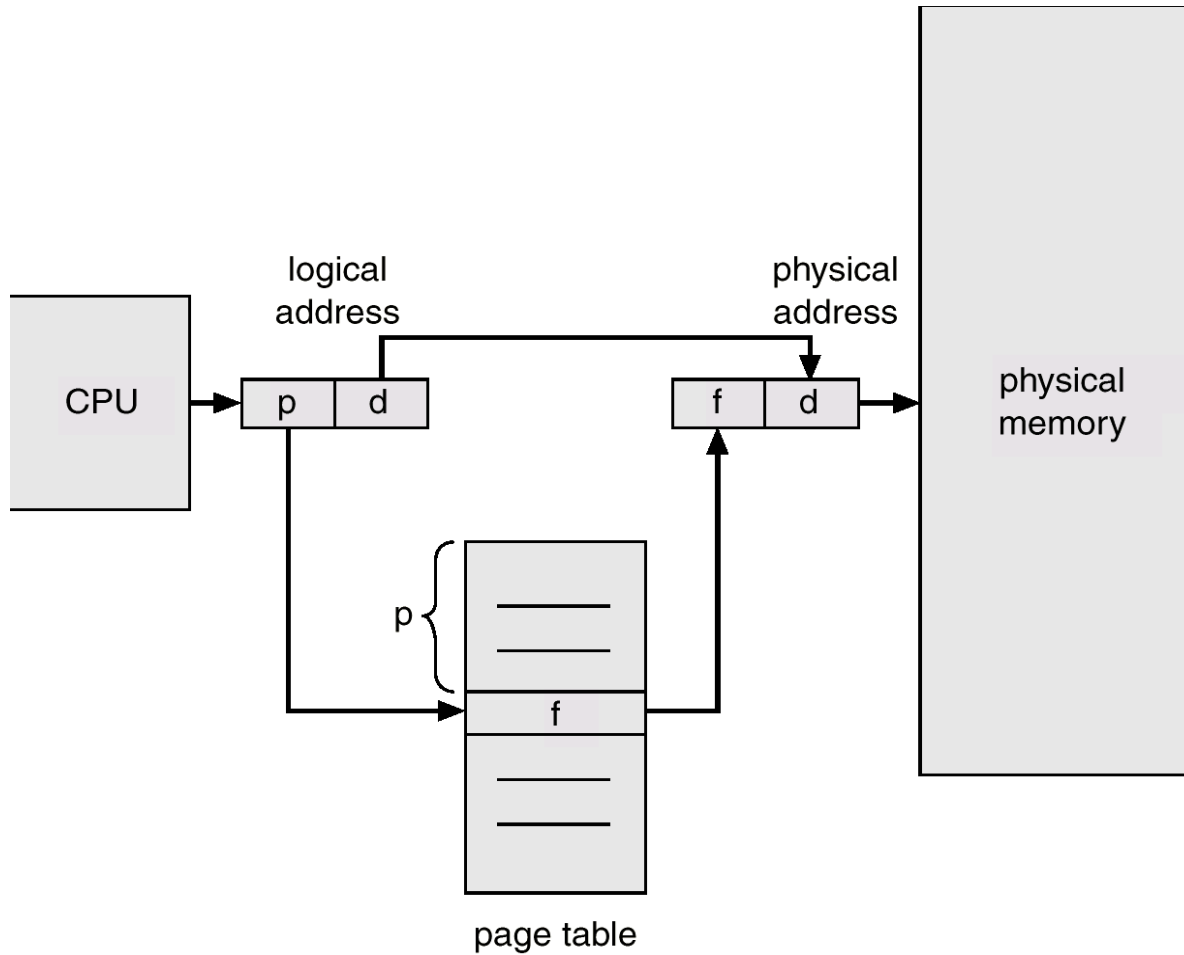
Offset – 12 bits

Logical Address – 20 bits

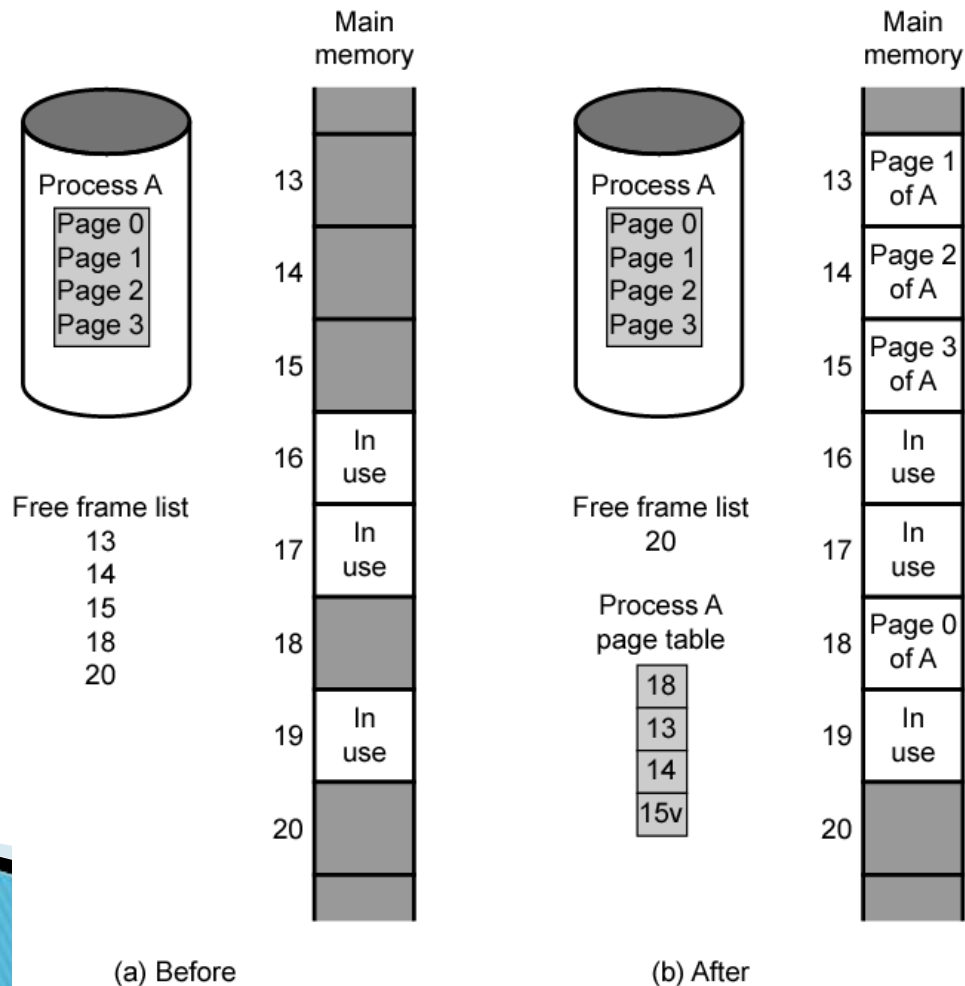
Ex 5

- ▶ Consider a computer system with a 32-bit logical address and 4-KB page size. The system supports up to 512 MB of physical memory. How many entries are there in the page table?
 - ▶ Offset – 12 bits(out of 32 bits)
 - ▶ Remaining 20 bits are for page nos.
 - ▶ No of pages in program– 2^{20}
 - ▶ 2^{20} entries in the page table
- 

Paging



- At a given point in time, some of the frames in memory are in use and some are free



Variation of paging(Demand paging)

- ▶ Do not load all pages of a program in memory, put some pages in memory which will be required and other pages are kept on the backing store (secondary memory)
- ▶ **Structure of a page table**


Page No	Frame No.	Valid	Dirty bit
0	5	1	1
1	2	1	0
2	3	1	1
3	–	0	0
4	–	0	

Valid bit – Indicates whether a page is actually loaded in main memory.

Dirty bit – Indicates whether the page has been modified during its residency in the main memory

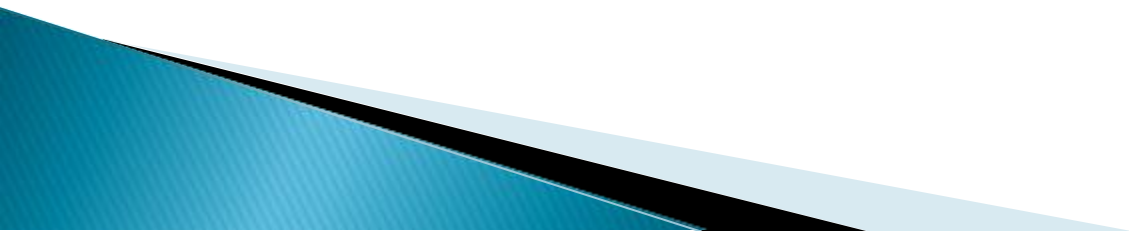
Demand paging

What happens when the CPU tries to use a page that was not brought into the memory ?

1. The MMU tries to translate the logical address into physical address by using the page table.
 2. During the translation procedure, the page no. of the logical address is used to index the PT. The valid bit for that page table entry is checked to indicate whether the page is in memory or not.
 3. If bit =0 then page is not in memory, and it is a page fault which causes an interrupt to the OS. i.e a request for a page that is not in MM.
 4. OS searches MM to find a free frame to bring in the required page.
 5. Disk operation is scheduled to bring in the desired page into the newly allocated frame.
 6. Page table is modified to indicate that the page is in MM.
 7. MMU uses the modified PT to translate the logical address to physical address.
- 

Implementation of paging

- ▶ There is one page table per process.
- ▶ When a process is in memory its page table is also brought into the memory(the area of MM Where the OS is loaded)
- ▶ The no. of entries in the page table depends on the size of the process.



Translation lookaside buffer(TLB)

- ▶ Page tables are stored in MM
- ▶ Every reference to memory causes **2 physical memory accesses.**
 1. One to fetch the appropriate PT entry from the PT.
 2. Second to fetch the desired data from MM once the frame no. is obtained from the PT.
- ▶ Hence implementing a virtual memory scheme has the effect of doubling the memory access time.

Solution:

Use a special cache for Page Table Entries called the Translation lookaside buffer(TLB)

TLB contains the most recently used Page Table Entries



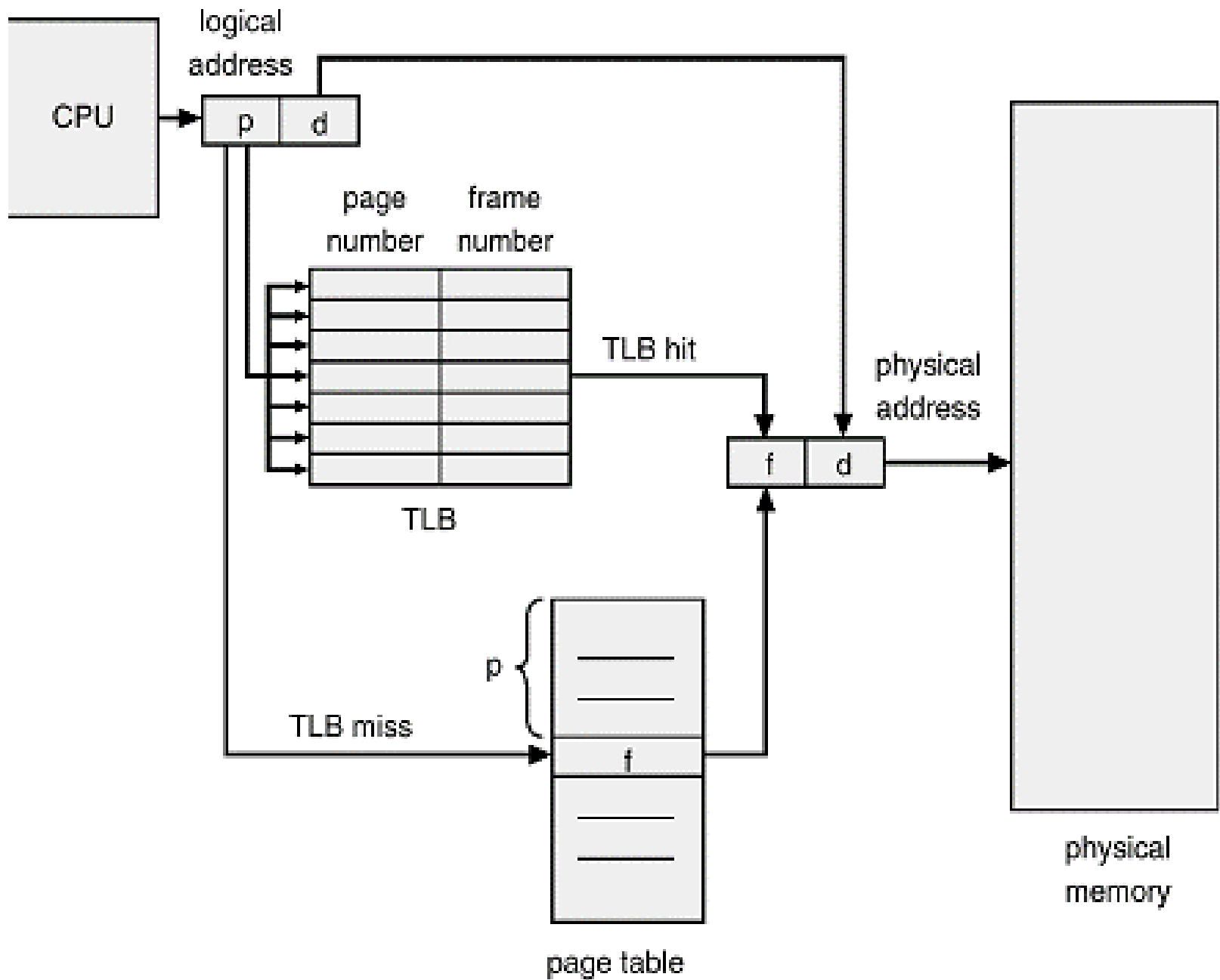
Translation lookaside buffer(TLB)

Operation of paging using a TLB:

1. Logical address is in the form



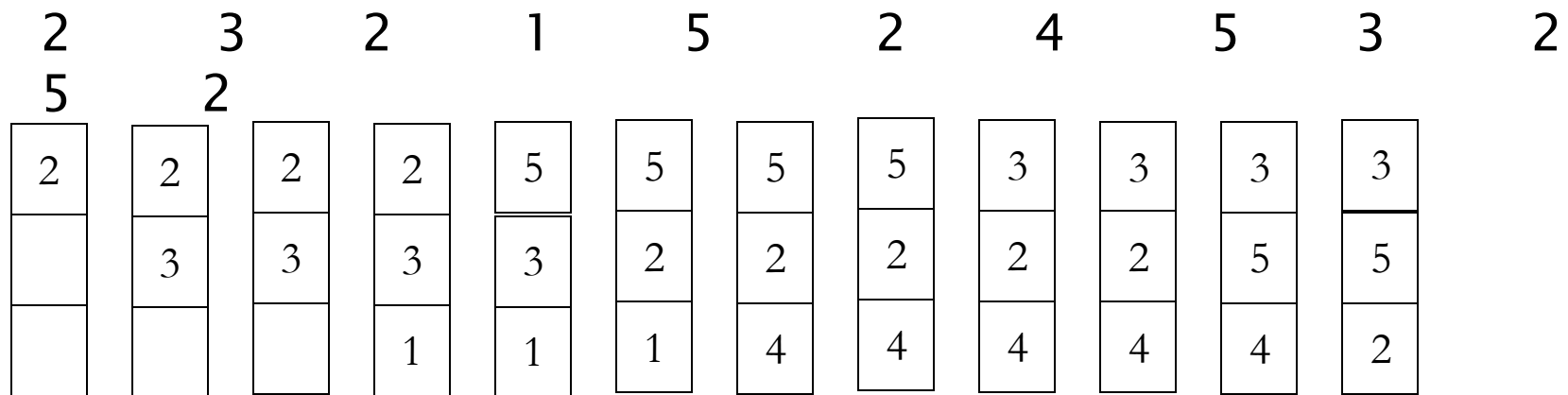
2. During the translation Process MMU consults the TLB to see if the matching Page table entry is present.
3. If match found, physical address is generated by combining the frame no. present at that entry at that entry in the TLB with the offset.
4. If match not found, entry is accessed from the page table in the memory.



Page Replacement Algorithms

- ▶ When MM is full page replacement algorithm determines which page in memory will be replaced to bring in the new page.
- 1. First-In-First-Out (FIFO)
 - ▶ Replaces the oldest page in memory i.e the page that has spent longest time in memory.
 - ▶ Implemented using a FIFO queue. Replace the page at the head of the queue.

- ▶ Assume MM has 3 page frames. Execution of a program requires 5 distinct pages P_i where $i = 1, 2, 3, 4, 5$
- ▶ Let the reference string be



H

H

H

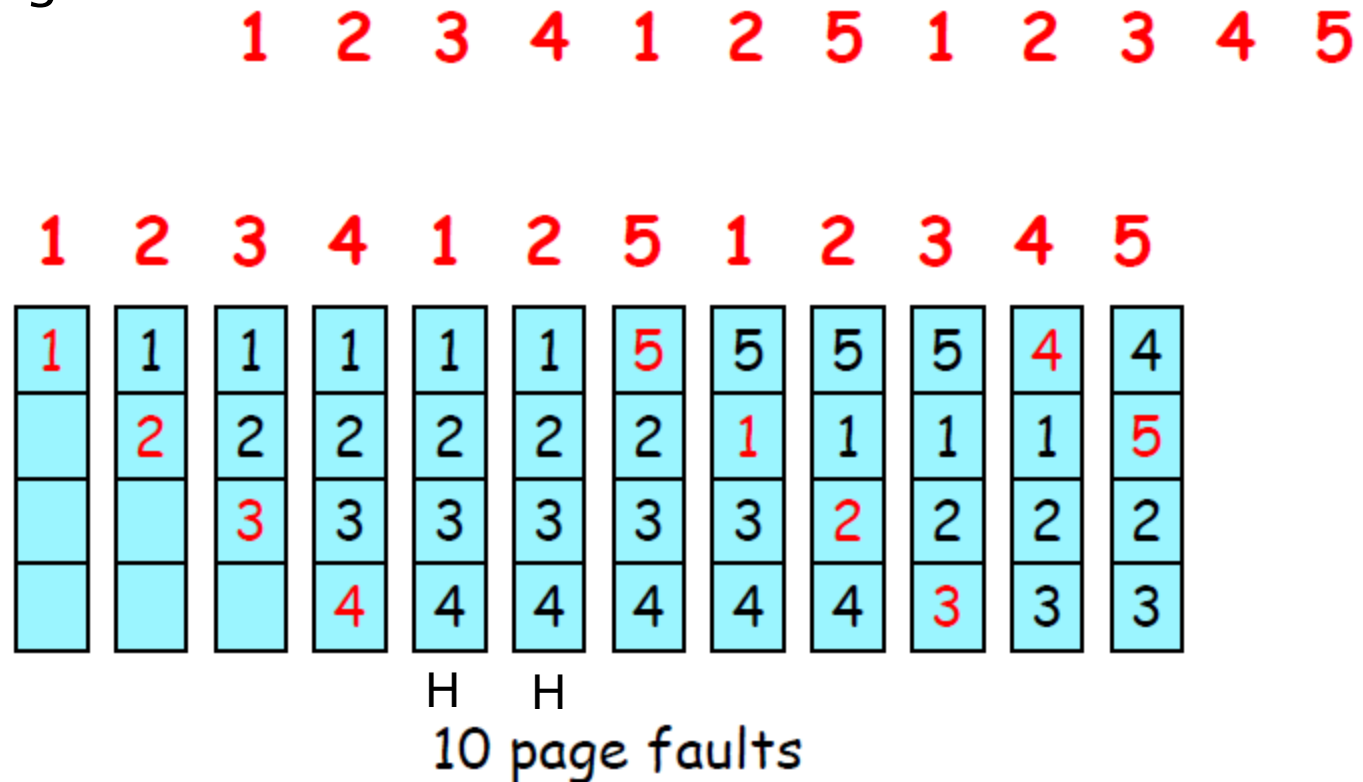
Hit ratio = $3/12$

Drawback: Tends to throw away frequently used pages because it does not take into account the pattern of usage of a given page

- ▶ Adv : Simple to implement
- ▶ Disadv: Suffers from belady's anomaly

Belady's anomaly

With 4 page frames



With 3 page frame

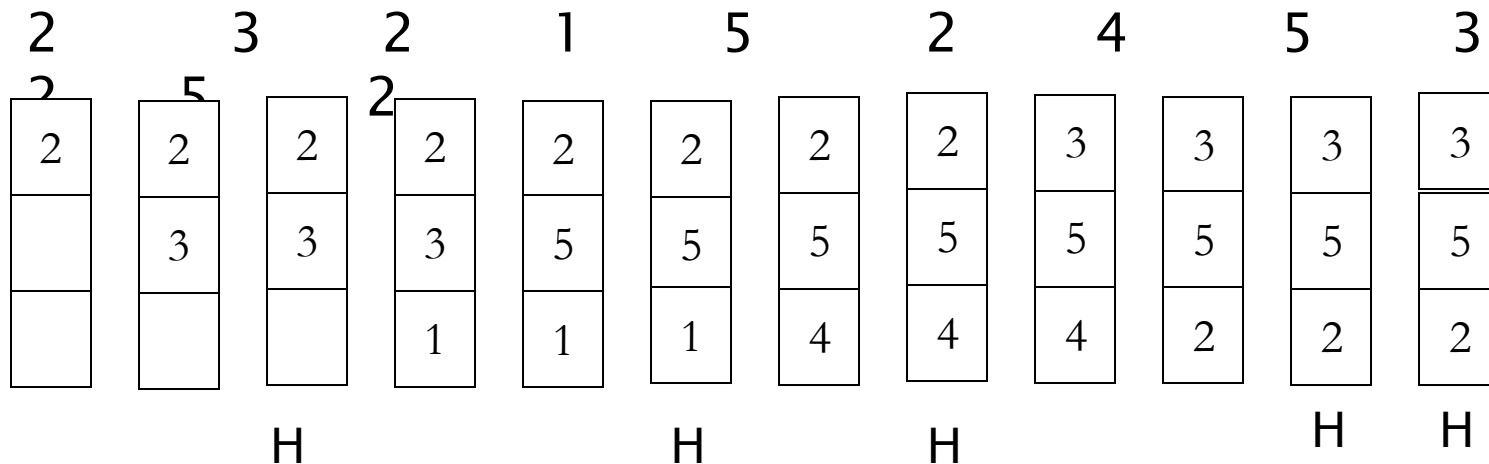
1 2 3 4 1 2 5 1 2 3 4 5

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

Bela , 9 page faults , the
number of page frames may also increase the
number of page faults.

2. Least Recently used Algorithm :

- Replaces that page which as not been **used** for the longest period of time.



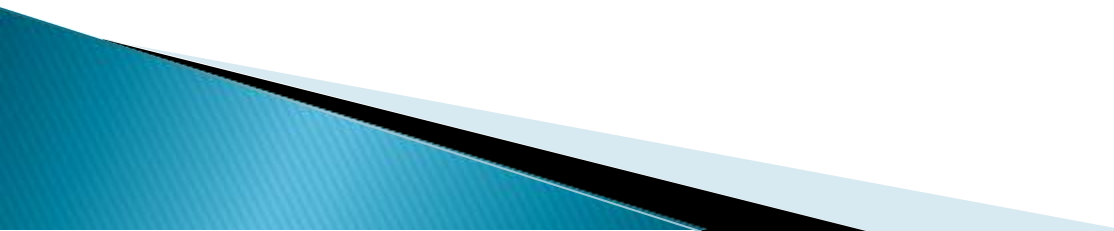
No. of page faults = 7

Hit ratio = 5/12

Generates less no. of page faults than FIFO

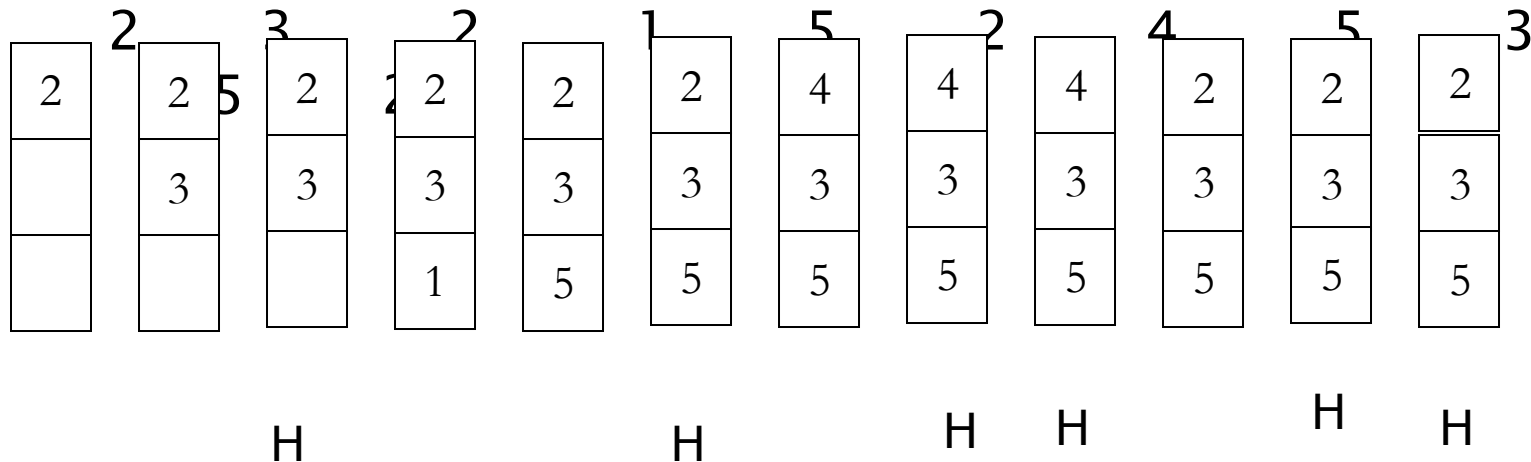
Commonly used algorithms

Implementation of LRU

- ▶ A counter for each page
 - ▶ Every time page is referenced, save system clock into the counter of the page
 - ▶ Page replacement: scan through pages to find the one with the oldest clock
 - ▶ Problem: have to search all pages/counters!
- 

3. Optimal page Replacement algorithm

- ▶ Replace that page which will not be used for the longest period of time.



$$\text{Hit Ratio} = 6/12 = 1/2$$

Drawback: Difficult to implement since it requires a future knowledge of the reference string.

Used mainly for comparison studies.

Reference String : 7 0 1 2 0 3 0 4 2 3 0

Assume no. of page frames = 3

FIFO

reference string

7 0 1 2 0 3 0 4 2 3 0

H

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

page frames

Hit ratio = 1/11

► LRU

reference string

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

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

page frames

Hit ratio = 2/11

► Optimal

reference string

7 0 1 2 0 3 0 4 2 3 0

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

page frames

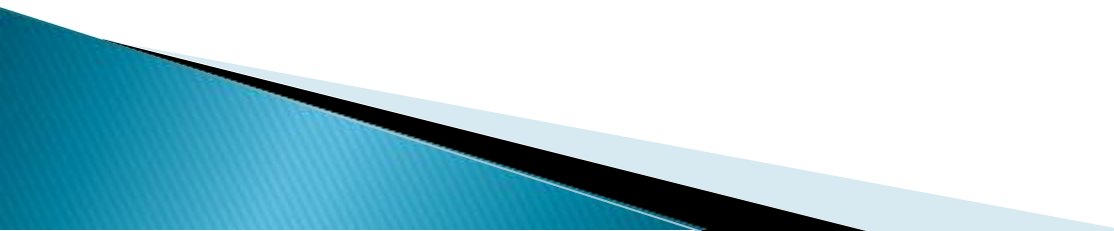
Hit ratio = 4/11

Issues associated with paging

1. Resident size management :

- ▶ The OS must decide how many page frames to allocate to a process:
 - large page fault rate if too few frames are allocated.
 - low multiprogramming level if too many frames are allocated.

Resident Set Size

- ▶ **Fixed-allocation policy:**
 - allocates a fixed number of frames that remains constant over time
 - the number is determined at load time and depends on the type of the application.
- 

Issues associated with paging

- ▶ **Variable-allocation policy:**

- the number of frames allocated to a process may vary over time:
 - may increase if page fault rate is high.
 - may decrease if page fault rate is very low.
- requires more OS overhead to assess behavior of active processes.

Issues associated with paging

2. Replacement Scope

- ▶ Replacement scope is the set of frames to be considered for replacement when a page fault occurs.

Local replacement policy:

- each process selects from only its own set of allocated frames.

Global replacement policy:

- process selects a replacement frame from the set of all frames; one process can take a frame from another.

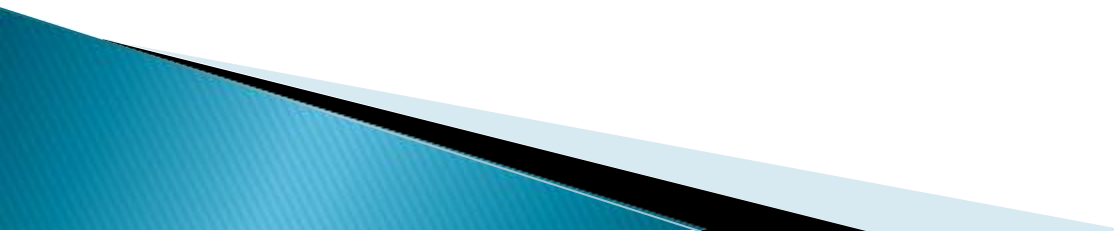


Table 8.5 Resident Set Management

	Local Replacement	Global Replacement
Fixed Allocation	<ul style="list-style-type: none">• Number of frames allocated to process is fixed.• Page to be replaced is chosen from among the frames allocated to that process.	<ul style="list-style-type: none">• Not possible.
Variable Allocation	<ul style="list-style-type: none">• The number of frames allocated to a process may be changed from time to time, to maintain the working set of the process.• Page to be replaced is chosen from among the frames allocated to that process.	<ul style="list-style-type: none">• Page to be replaced is chosen from all available frames in main memory; this causes the size of the resident set of processes to vary.

Issues associated with paging

Thrashing

- ▶ Consider the scenario when the resident set size of a process is inadequate:
- ▶ If a process does not have enough frames it will quickly page fault.
- ▶ At this point it must replace some page.
- ▶ However since all its pages are in active use, it must replace a page that will be needed right way.
- ▶ Hence it will quickly fault again and again.
- ▶ The process continues to fault replacing the pages for which it then faults and brings back right away.

This high paging activity is called thrashing


A process is thrashing if it is spending more time paging than executing.



Issues associated with paging

Cause and effects of thrashing

Consider the following scenario:

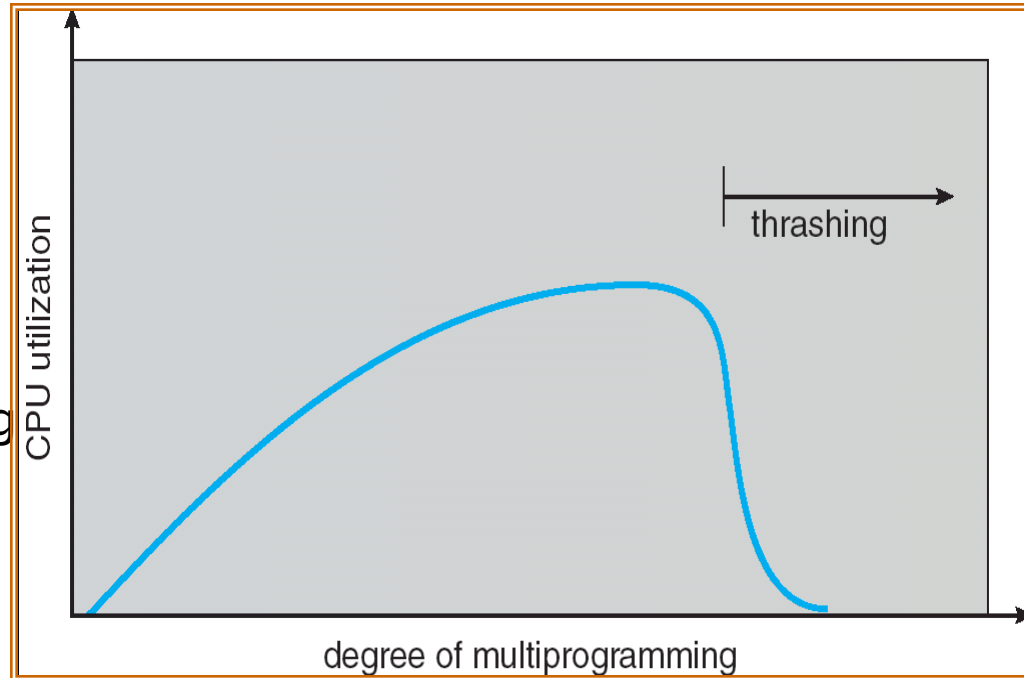
- ▶ OS monitors CPU utilization.
 - ▶ If the CPU utilization is too low , the degree of multiprogramming is increased by introducing a new process to the system.
 - ▶ Assume a global page replacement is used.
 - ▶ Suppose a process enters a new phase in its execution and needs more frames.
 - ▶ It starts faulting and taking frames away from other processes.
 - ▶ These processes need those pages , hence these processes also start faulting taking frames from other processes.
- 

Issues associated with paging

- ▶ All these faulting processes must use the paging device to swap pages in and out.
- ▶ As a result they queue for the paging device and as a result the ready queue empties.
- ▶ Hence the cpu utilization decreases.
- ▶ OS sees decreasing cpu utilization and increases the degree of multiprogramming.
- ▶ The new processes try to get started by taking frames from running processes, causing more page faults and longer queue for the paging device.
- ▶ As a result the CPU utilization even drops further and the OS further tries to increase the degree of multiprogramming even more.
- ▶ Thrashing has occurred and the system throughput plunges.
- ▶ No work is getting done because processes are spending more time paging.

Issues associated with paging

- ▶ CPU utilization is plotted against the degree of multiprogramming
- ▶ As degree of multiprogramming increases, CPU utilization increases until a maximum is reached
- ▶ If degree of multiprogramming is increased even further, thrashing sets in and CPU utilization drops sharply.
- ▶ At this point to increase CPU utilization and stop thrashing, the degree of multiprogramming should be decreased.



Issues associated with paging

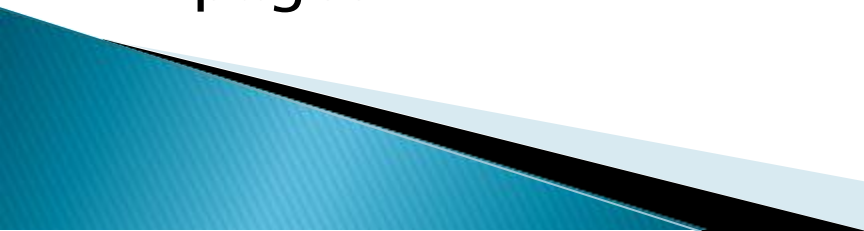
How to limit effects of thrashing ?

- ▶ Use local replacement algorithm.
- ▶ Hence one process cannot steal frames from other processes and cause latter to thrash.
- ▶ But processes which are thrashing will be in the queue for the paging device most of the time.
- ▶ Hence the average service time for a page fault will increase for a process that's not thrashing.


Issues associated with paging

How to prevent thrashing?


1. Working set Model

- ▶ Provide a process as many frames as it needs..
 - ▶ How do we know that?
 - ▶ Use the **working set strategy**.
 - ▶ Uses the concept of locality model of process execution.
 - ▶ They exhibit a **locality of reference**, meaning that **during any phase of execution**, a process references only a relatively small fraction of its pages.
- 

Issues associated with paging

- ▶ Locality model states that as a process executes, it moves from locality to locality.
 - ▶ A locality are a set of pages actively used together.
 - ▶ When a function is called all pages that span the function are used . This is a locality for that function.
 - ▶ When a function is exited , the process leaves that locality end enters another locality.
- 

Issues associated with paging

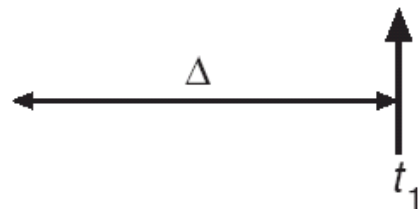
- ▶ Hence if enough frames are allocated to a process to accommodate its current locality then it will fault for the pages in its locality until all these pages are in memory, then it will not fault again until the process changes locality.
 - ▶ If fewer frames are allocated than the size of the current locality, the process will thrash, since it cannot keep in memory all the pages that it is actively using.
 - ▶ WORKING SET MODEL is based on the assumption of locality.
- 

Issues associated with paging

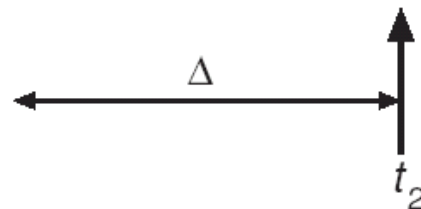
- ▶ The working-set model is a way of estimating the size of the current locality for a process.
- ▶ $\Delta \equiv$ working-set window \equiv a fixed number of page references
- ▶ The working set is the set of unique pages in the most recent Δ page references. Example: $\Delta = 10,000$

page reference table

... 2 6 1 5 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 ...



$WS(t_1) = \{1, 2, 5, 6, 7\}$



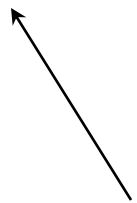
$WS(t_2) = \{3, 4\}$

Here $\Delta = 10$

Issues associated with paging

▶ $T = 5$

▶ 1 2 3 2 3 1 2 3 1 2 2 4 3 4 7 4 3 3 4 3 4 7 3 4 3 1 2 2 2 1



$W = \{1, 2, 3\}$

$W = \{3, 4, 7\}$

$W = \{1, 2\}$

The accuracy of the working set depends on the selection of T .

- if T too small, will not encompass locality
 - if T too large, will encompass several localities
 - if $T \rightarrow$ infinity, will encompass entire program
- ▶ If the entire working set is in memory, the process will run without causing many faults until it moves into another execution phase

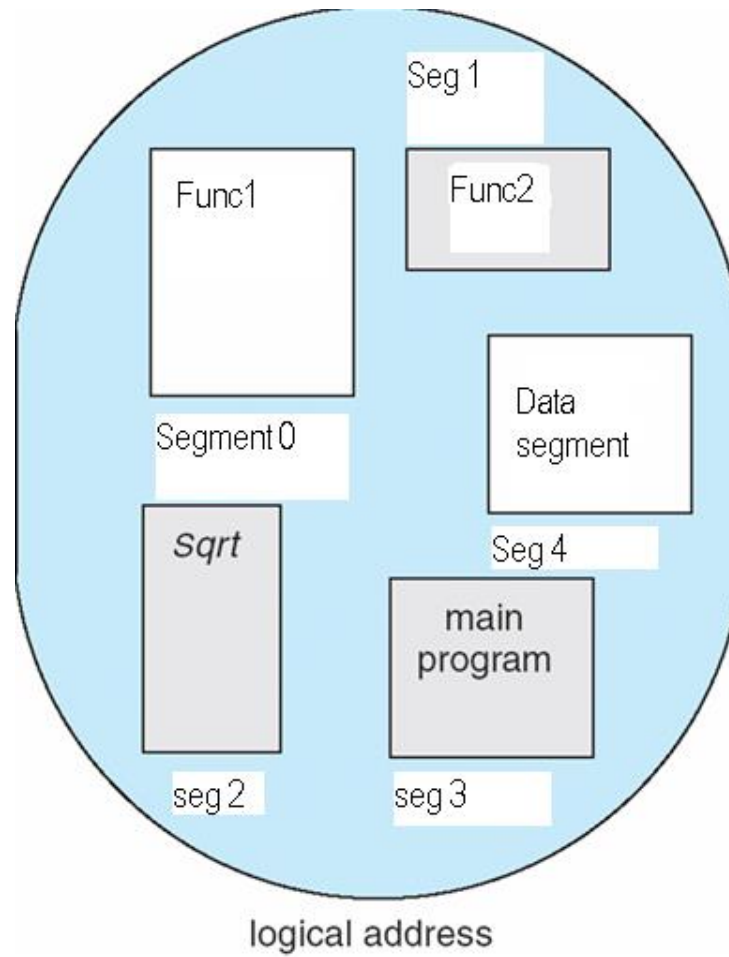
Issues associated with paging

- ▶ m = number of frames in memory
- ▶ $D = \sum WSS_i \equiv$ total demand frames
- ▶ if $D > m \Rightarrow$ Thrashing
- ▶ Policy if $D > m$, then suspend one of the processes

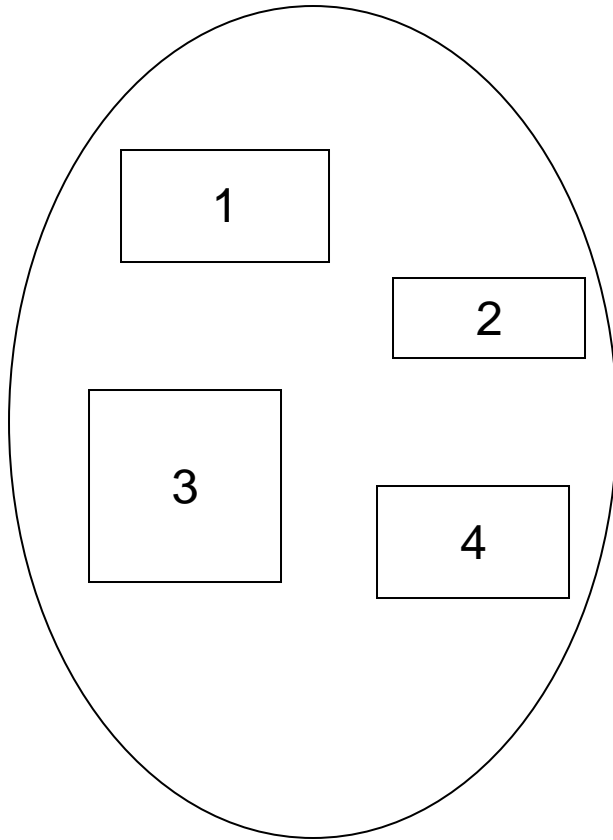
Segmentation

- ▶ Paging is not (usually) visible to the programmer
- ▶ Segmentation is visible to the programmer.
- ▶ Segmentation is a memory management scheme that supports user's view of memory.
- ▶ Programmers never think of their programs as a linear array of words. Rather, they think of their programs as a collection of logically related entities, such as subroutines or procedures, functions, global or local data areas, stack etc.
- ▶ i.e they view programs as a collection of segments.
- ▶ A program consists of functions/subroutines , main program and data which are required for the execution of the program.
- ▶ Each logical entity is a segment.
- ▶ Hence if we have a program consisting of 2 functions and an inbuilt function sqrt.
- ▶ Then we have a segment for the main program, each of the functions, data segment and a segment for sqrt.

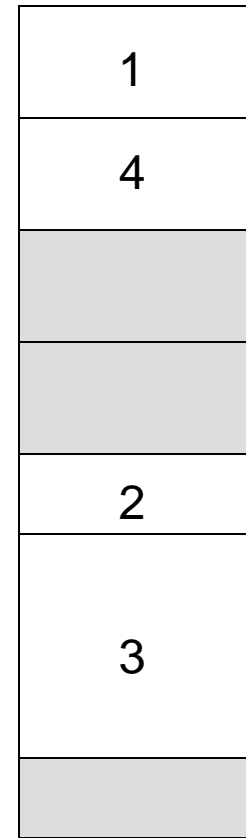
Segmentation



Segmentation

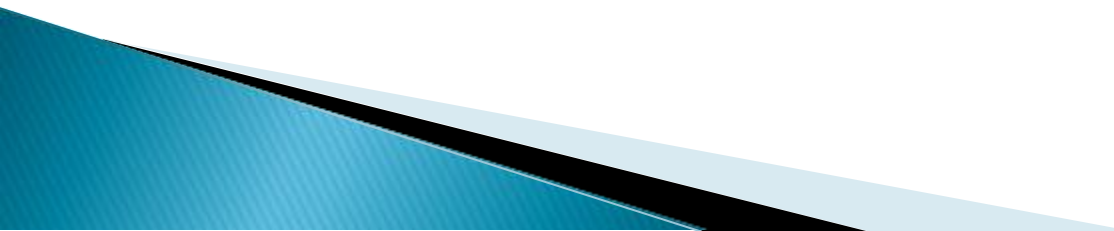


user space



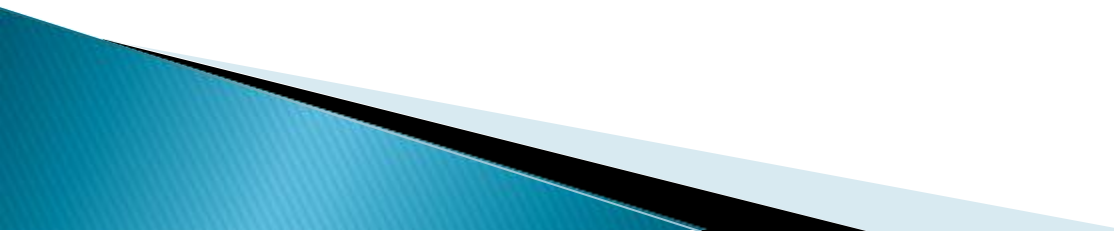
physical memory space

Segmentation

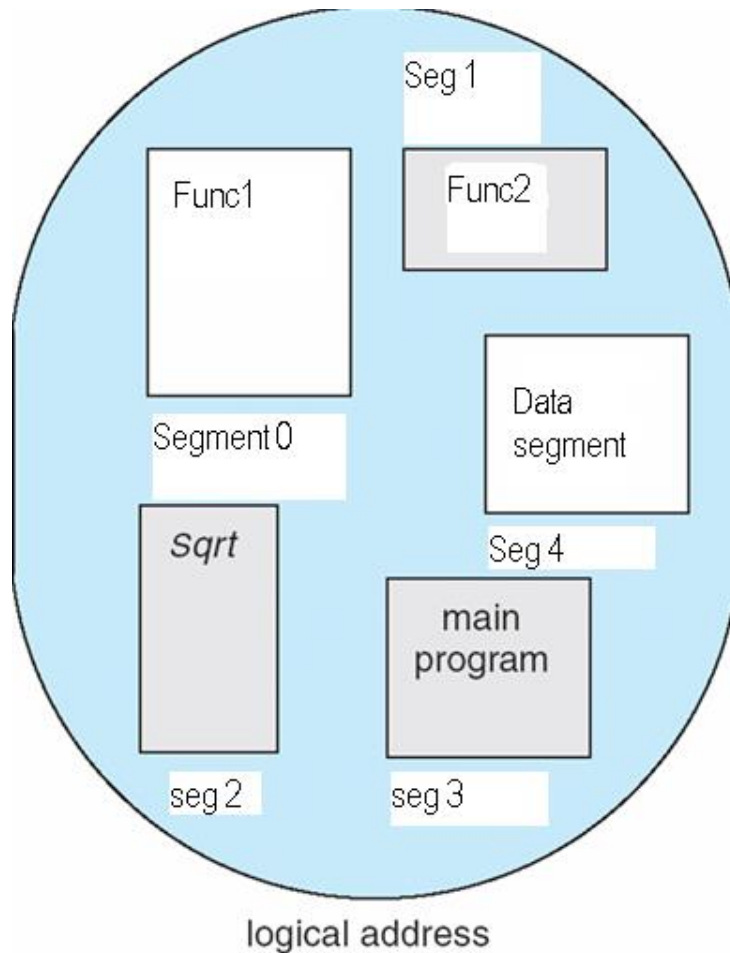
- ▶ Each segment has a name and a length.
 - ▶ A logical address in segmentation is specified
 1. A segment name (segment number)
 2. An offset within the segment
 - ▶ The user therefore specifies each address by 2 quantities :
segment no. and the offset
 - ▶ When a user program is compiled, the compiler automatically constructs different segments reflecting the input program.
- 

Segmentation

Conversion of logical address to physical address :

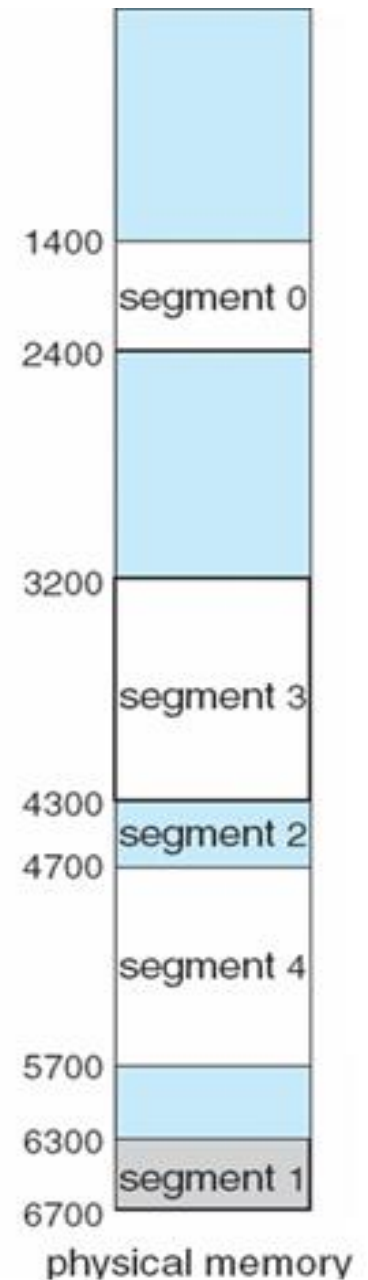
- ▶ This mapping is done with the help of a segment table.
 - ▶ Every process has its own segment table.
 - ▶ Each entry in the segment table has a segment base and a segment limit.
 - ▶ The **segment base** contains the starting physical address where the segment resides in memory.
 - ▶ **Segment limit** specifies the length of the segment.
- 

Segmentation



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

segment table



Segmentation

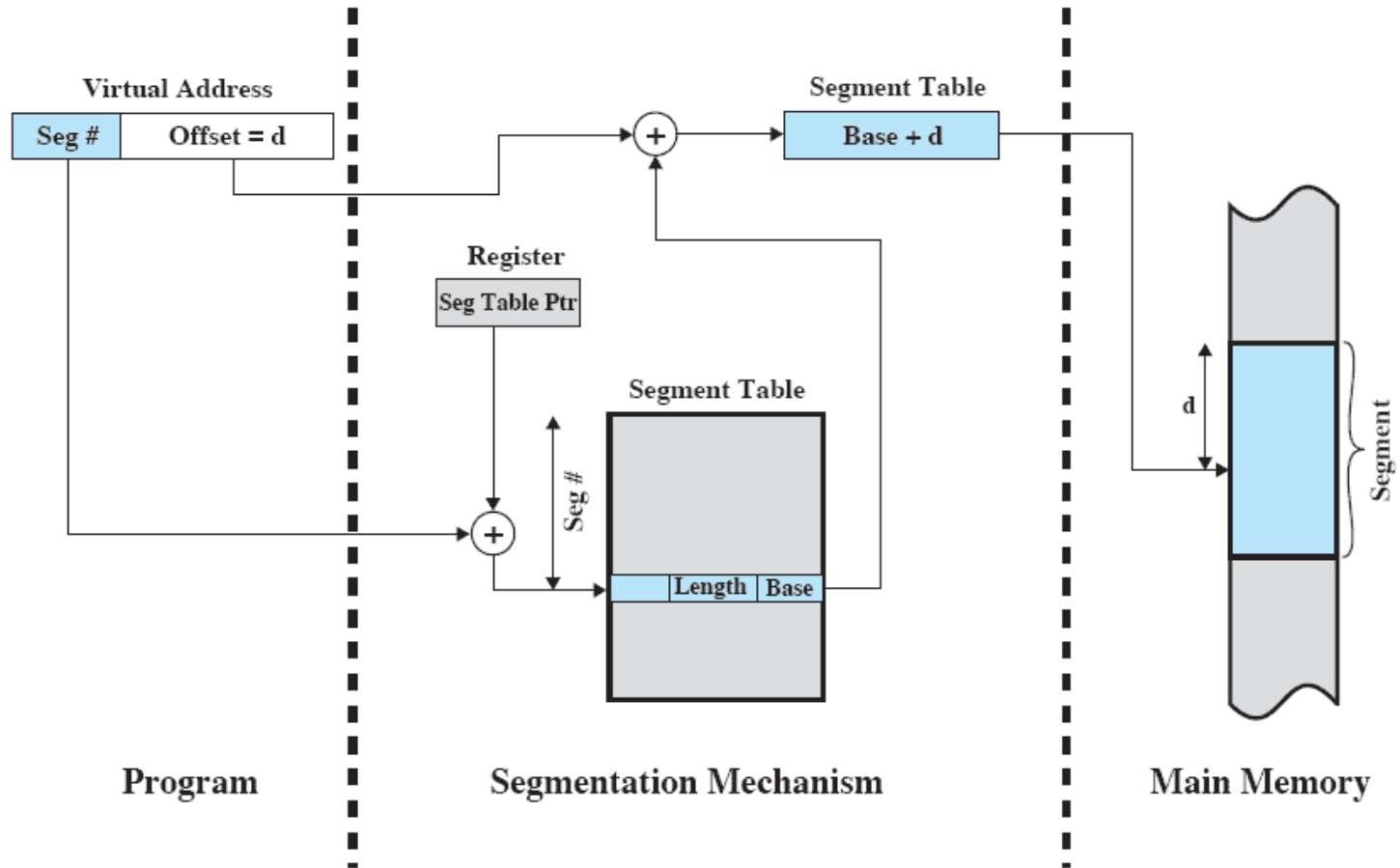
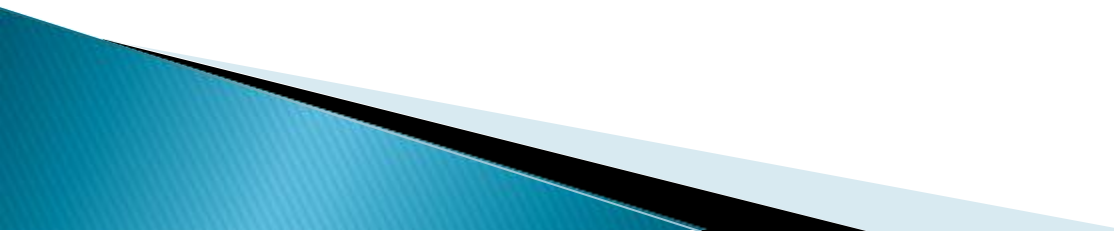


Figure 8.12 Address Translation in a Segmentation System

Segmentation

- ▶ Eg: Segment 2 is 400 bytes long and begins at location(address) 4300.
 - ▶ A reference to byte no. 53 of segment 2 is mapped onto location $4300 + 53 = 4353$.
 - ▶ A reference to segment 3, byte 852, is mapped onto $3200 + 852 = 4052$.
 - ▶ A reference to byte 1222 of segment 0 would result in a trap(interrupt) to the OS as this segment is only 1000 bytes.
- 

Segmentation

- ▶ **External Fragmentation**– total memory space exists to satisfy a request, but it is not contiguous.
- ▶ Each segment is allocated a contiguous piece of physical memory. As segments are of different sizes swapping in/out of segments leads to holes in memory hence causes external fragmentation.
- ▶ Memory allocation to a segment is done using first fit, worst fit and best fit strategy
- ▶ Compaction is needed to resolve external fragmentation

Problems

Assuming a 15-bit address space with 8 logical pages. How large are the pages?

It takes 3 bits to reference 8 logical pages ($2^3 = 8$). This leaves 12 bits for the page size thus pages are 2^{12} bytes long (4096 bytes)

Assume a page size of 1K and a 15-bit logical address space. How many pages are in the system?

$2^5 = 32$.

Consider a paging system with the page table stored in memory. If a memory reference takes 200 nanoseconds, how long does a paged memory reference take?

- $200 \text{ ns} \times 2 = 400 \text{ ns}$

- Consider logical address 1025 and the following page table for some process P0. Assume a 15-bit address space with a page size of 1K. What is the physical address to which logical address 1025 will be mapped?

8
0
2

Step 1. Convert 1025 to binary:
000010000000001

Step2. Determine the logical page number:

Since there are 5-bits allocated to the logical page, the address is broken up as follows:

00001

0000000001

Step 3. Use logical page number as an index into the page table.

00001 0000000001

Take the physical page number from the page table and add the offset.

The physical address is byte 1.

0000000000000001

Numericals

- Consider a 15-bit logical address space and a page size of 1k. How many pages are there in the system?

1. 16

2. 12

3. 32

4. 8

Ans : 3

Assume a 18- bit logical address with 8 pages.
What is the size of the page ?

1. 4096 bytes(4 KB)
2. 8192 bytes(8 KB)
3. 16384 bytes(16 KB)
4. 32768 bytes(32 KB)

Ans 4

Consider a 16 bit virtual address and let the size of the page 1 KB with each entry of page table consisting of 4 bytes. What is the size of the page table ?

1. 256 bytes
2. 512 bytes
3. 64 bytes
4. 1 KB

Ans: 1)256 bytes

Assume a program of 1024 pages and page frame size of 512 bytes. The number of bits in the logical address is

1. 12

2. 16

3. 15

4. 19

Ans : 19 bits

- Assuming a 1-KB page size, what are the page numbers and offsets for the following address references :
- 3085 b. 42095
- Answer:
- a. page = 3; offset = 13
- b. page = 41; offset = 111

Assuming a 15-bit address space with 8 logical pages. How large are the pages?

It takes 3 bits to reference 8 logical pages ($2^3 = 8$). This leaves 12 bits for the page size thus pages are 2^{12} bytes long (4096 bytes)

Assume a page size of 1K and a 15-bit logical address space. How many pages are in the system?

$2^5 = 32$.

- Consider logical address 1025 and the following page table for some process P0. Assume a 15-bit logical and physical address space with a page size of 1K. What is the physical address to which logical address 1025 will be mapped?

8
0
2

Step 1. Convert 1025 to binary:
000010000000001

Step2. Determine the logical page number:

Since there are 5-bits allocated to the logical page, the address is broken up as follows:

00001

0000000001

Step 3. Use logical page number as an index into the page table.

00001 0000000001

Take the physical page number from the page table and add the offset.

The physical address is byte 1.

0000000000000001

- Consider a logical address space of 8 pages of 1024 bytes each mapped into memory of 32 frames.

a. How many bits are there in the logical address ?

b. How many bits are there in physical address ?

Ans:

a. Logical address will have

3 bits to specify the page number (for 8 pages) .

10 bits to specify the offset into each page ($2^{10} = 1024$ words) = **13 bits.**

b. For 32 frames of 1024 words each (Page size = Frame size)

We have $5 + 10 = \mathbf{15 \text{ bits.}}$