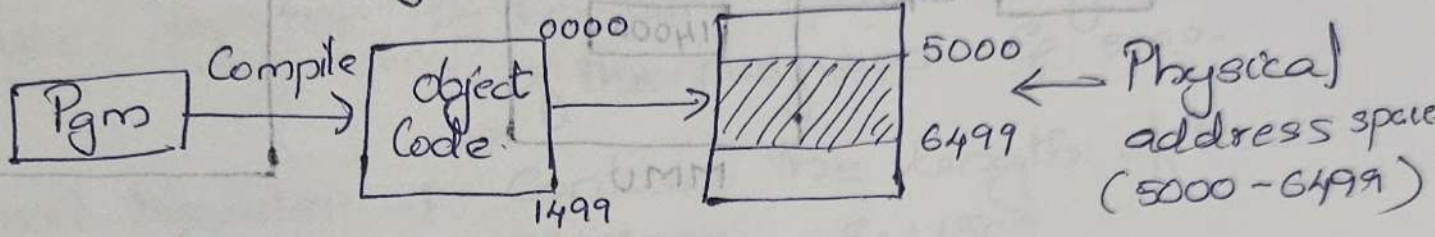
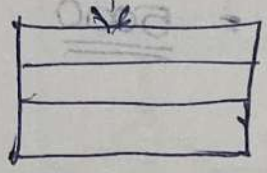
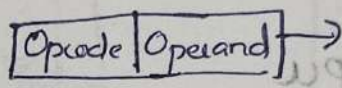
Memory Management



↳ logical address space (0000 - 1499)



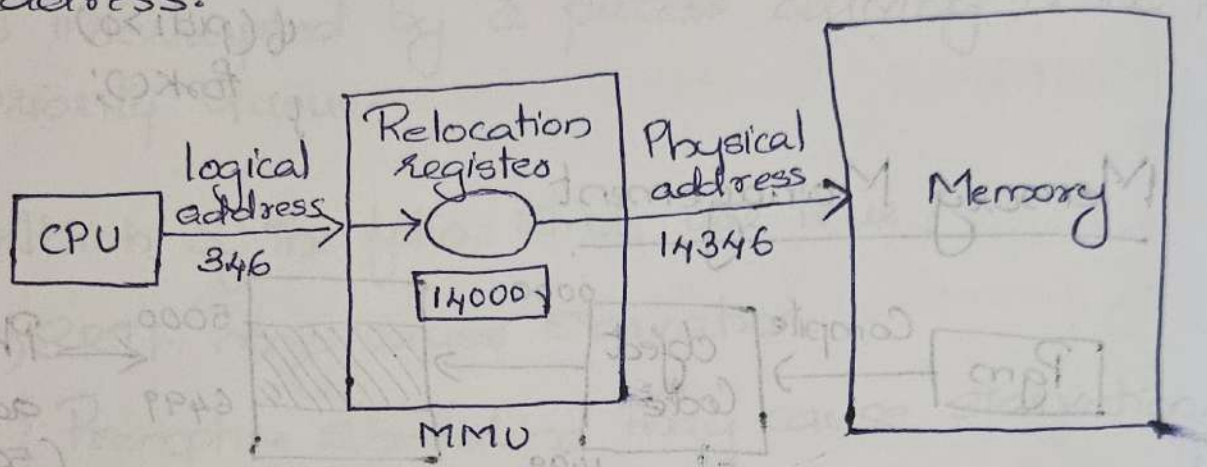


specify the logical address space. Only

∴ Memory Management Unit. Converts the logical address to Physical address. (MMU).

* **Logical Address** is generated by CPU while a pgm is running.
* The term Logical Address space is used for the set of all logical addresses generated by a pgm's perspective.
* **Physical Address** identifies a physical location of required data in a m/y.
* The user never directly deals with the physical address but can access by its corresponding logical address.

* The Hardware device called Memory Management Unit (MMU) is used for mapping logical address to its corresponding Physical address.



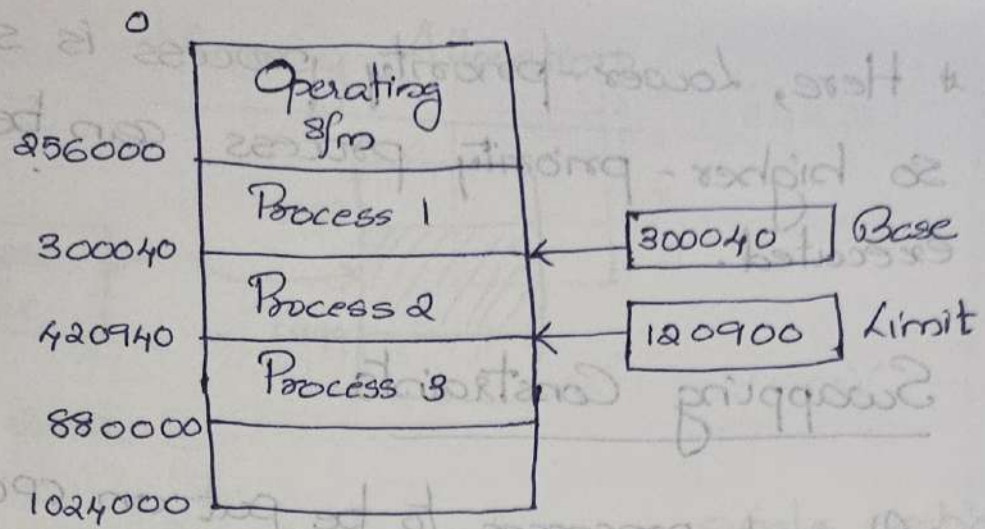
Eg: Relocation reg :→ |5000 − 0000|
= 5000

If CPU generates a logical address 200 then, →

5000 +
200
5200 ⟹ Physical address.

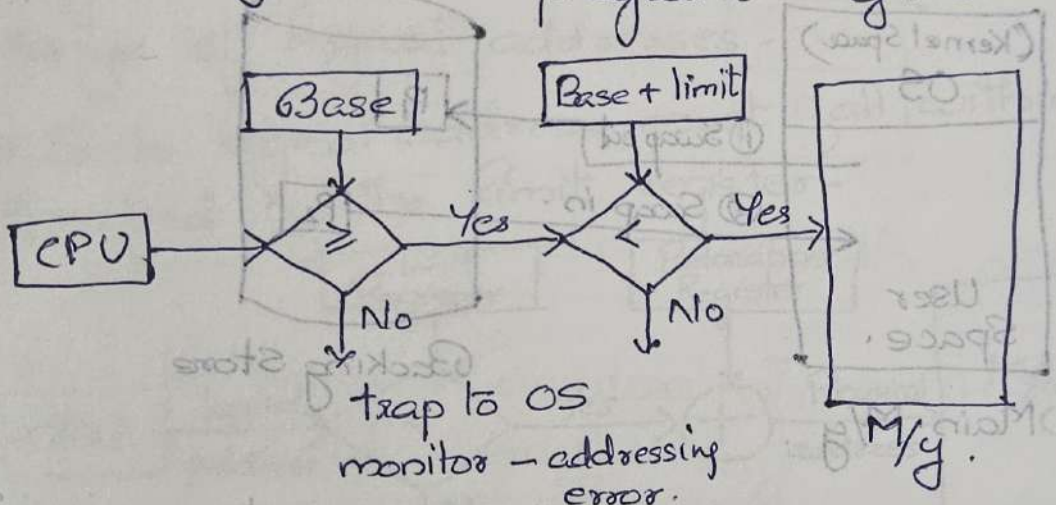
Memory Protection

* A pair of base & limit registers define the logical address space.
* CPU must check every memory access generated in user mode to be sure it is between base & limit for that user.
* Any attempt by a pgm executing in user mode to access OS m/y or other users m/y results in a trap to the OS, which treats the attempt as a fatal error.



Base Register :- Contains the starting address of the program. Eg: 5000.

Limit Register :- Contains the length of the program. Eg: 1500.



Swapping

* A process can be swapped temporarily out of m/y to a backing store, & then brought back into m/y for continued execution.

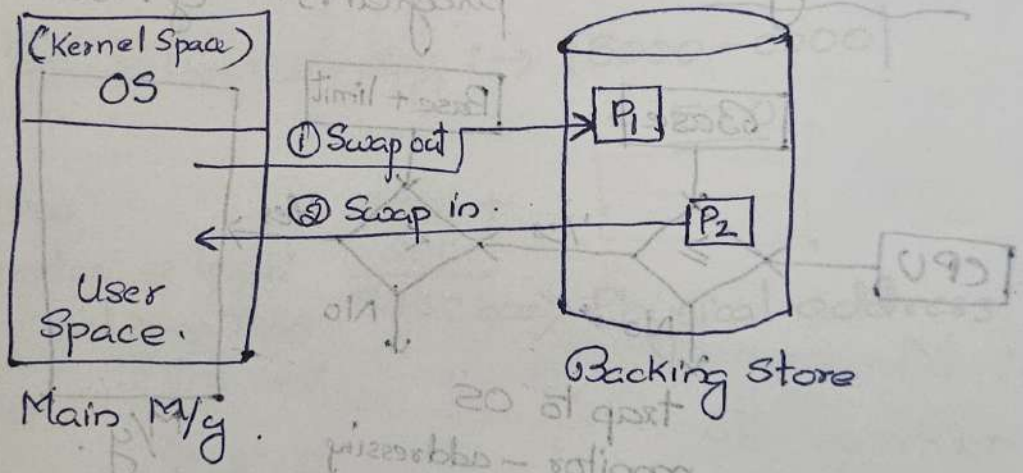
* Backing Store is a fast disk large enough to accomodate copies of all m/y images for all users.

* Roll out, Roll in is a process of swapping variant used for priority - based scheduling algorithms.

* Here, Lower-priority process is swapped out so higher-priority process can be loaded & executed.

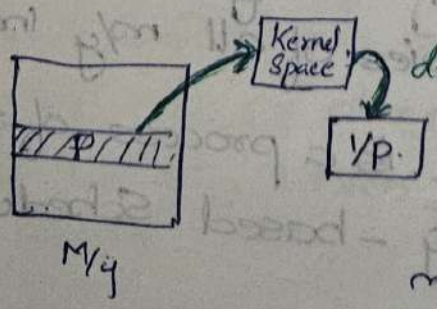
Swapping Constraints

* If next processes to be put on CPU is not in m/y, need to swap out a process & swap in target process Context Switch Time can then be very high.



* Pending I/O – We can't swap out process that is having a pending I/O as I/O could occur to wrong process.

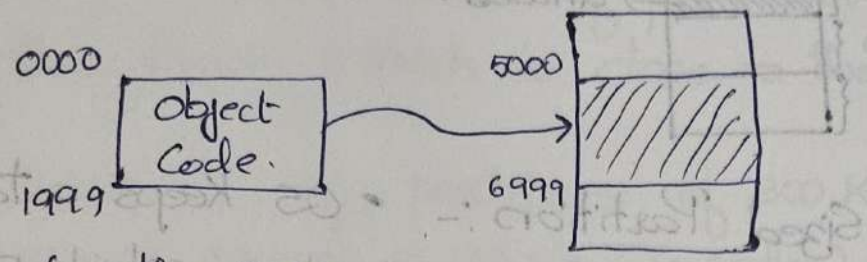
* In Sometimes, pending I/O will be transferred to Kernel space, then to I/O device & it is known as double buffering. It adds Overhead.



double buffering.

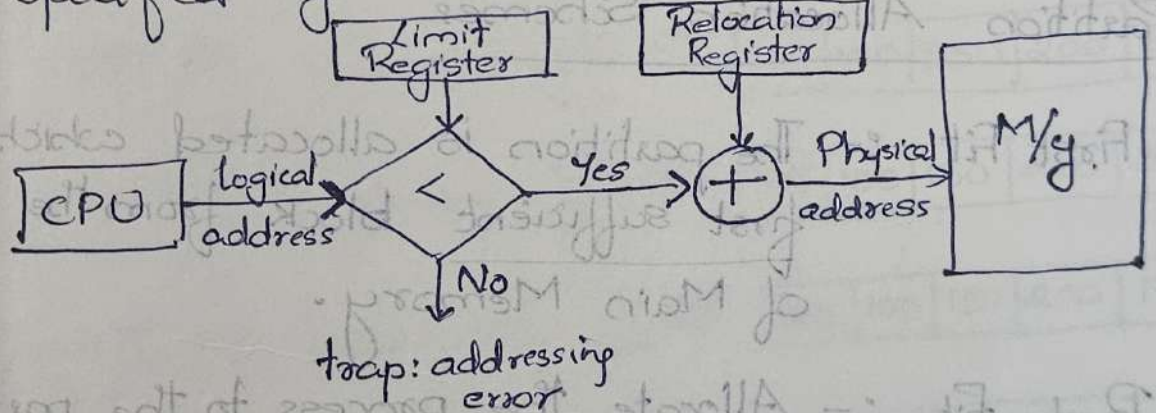
If P comes, if it to be processed, P should be moved to Kernel space as it has pending I/O ops. ie; if P is placed to backing store, then Q will given wrong I/P which is for P only

Contiguous Memory Allocation



Length = 2000

* Each process is contained in a single section of m/y that is contiguous to the section containing the next process.
* The relocation reg contains the value of the smallest physical address. & the limit reg contains the range of logical addresses.
* Each logical address must fall within the range specified by the limit register.



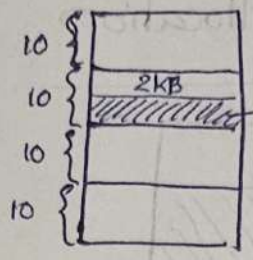
* Implemented in 2 ways :-

→ Fixed Sized Partitions :- • Divide m/y into several fixed sized partitions.
• Each partition may contain exactly one process.

Degree of Multiprogramming
↳ is bounded by the number of partitions of M/y.

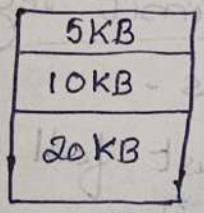
Each 10 KB Size.

If 2KB



→ Variable Sized Partition :- • OS keeps a table indicating which parts of m/y are available & which are occupied.

• Initially, all M/y is available for user processes & is considered one large block of available M/y
↳ Hole.



Partition Allocation Schemes

• First Fit :- The partition is allocated which is first sufficient block from the top of Main Memory.

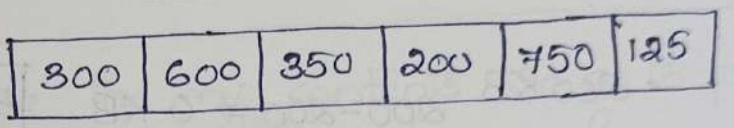
• Best fit :- Allocate the process to the partition which is the first smallest sufficient partition among the free available partition.

• Worst fit :- Allocate the process to the partition which is the largest sufficient among the freely available partitions available in main m/y.

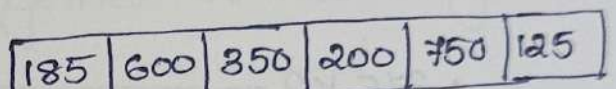
Here, the best fit minimizes the wastage space; but, it consumes a lot of processor time for searching the block which is close to the required size.

Q. Given 6 m/y partitions of 300 KB, 600 KB, 350 KB, 200 KB, 750 KB & 125 KB (in order). how would the first-fit, best-fit & Worst-fit algthms place processes of size 115 KB, 500 KB, 358 KB, 200 KB and 375 KB (in order)?

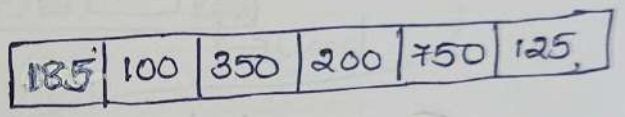
First Fit



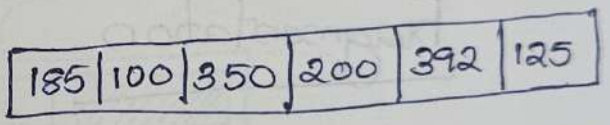
- 115 KB — 300-115 = 185 KB



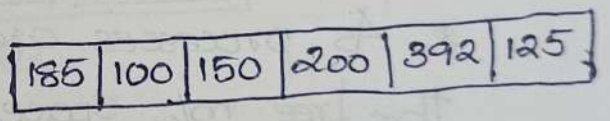
- 500 KB — 600-500 = 100 KB



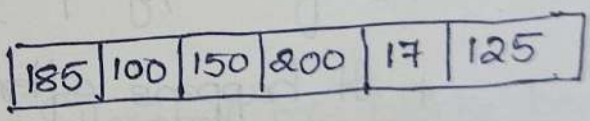
- 358 KB — 750-358 = 392 KB



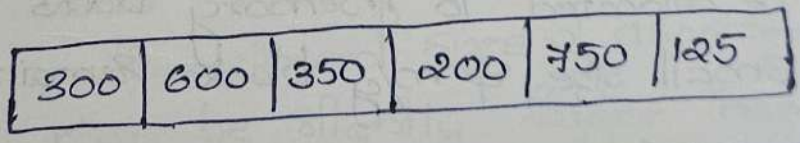
- 200 KB — 350-200 = 150 KB



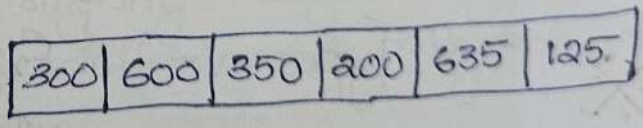
- 375 KB — 392-375 = 17 KB



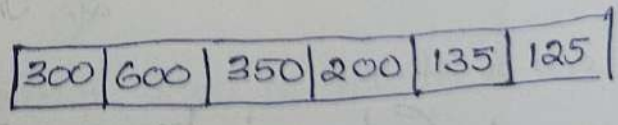
Worst fit



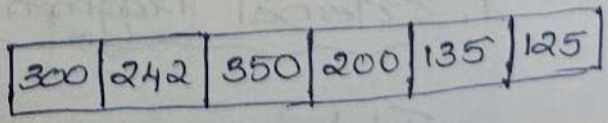
- 115 KB — 750-115 = 635 KB



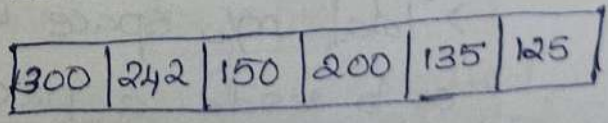
- 500 KB — 635-500 = 135 KB



- 358 KB — 600-358 = 242 KB

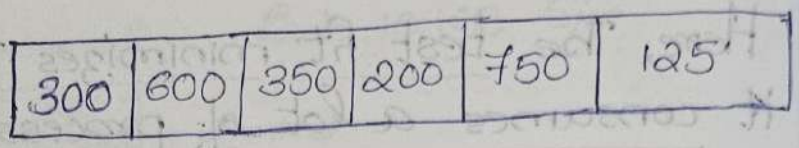


- 200 KB — 350-200 = 150 KB

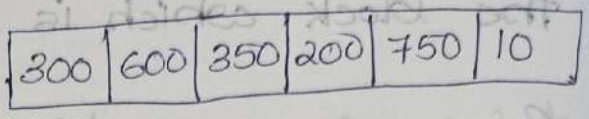


- 375 KB ⟹ Couldn't allocate.

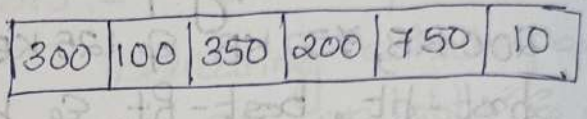
Best fit



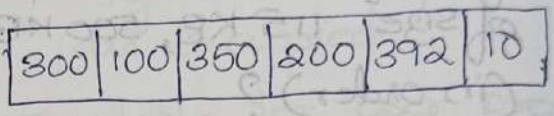
- 115 KB 125-115 = 10 KB



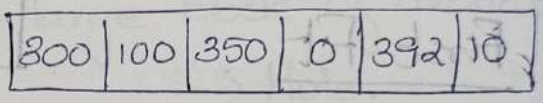
- 500 KB 600-500 = 100 KB



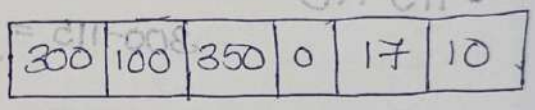
- 358 KB 750-358 = 392 KB



- 200 KB 200-200 = 0 KB



- 375 KB 392-375 = 17 KB



Fragmentation

* As processes are loaded & removed from m/y, the free m/y space is broken into little pieces.
* It happens after sometimes that processes cannot be allocated to memory blocks considering their small size & m/y blocks remains unused.
* 2 Types :- 1. External Fragmentation
  2. Internal Fragmentation.

1. External Fragmentation :- {Dangerous}

→ Total m/y space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.

Eg: In Worst Fit, 375 KB

$300 + 242 + 150 + 200 + 135 + 125$

$= 1152$ KB

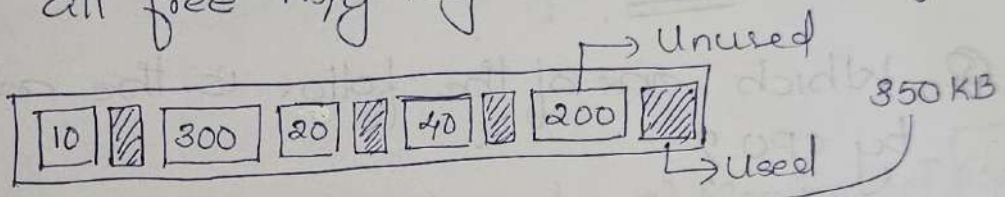
↓

Total M/y Space Enough to satisfy 875 KB, but it is not contiguous.

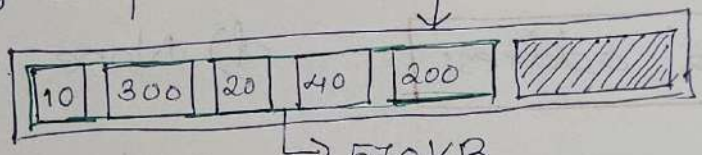
→ One solution to the Pblm of External Fragmentation is **Compaction**

↳ Shuffle the m/y contents so as to place all free m/y together in one large block.

Eg:



After Compaction :-



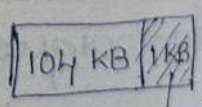
2. Internal Fragmentation :-

→ The space wasted inside of allocated m/y blocks because of restriction on the allowed sizes of allocated blocks. Allocated m/y may be slightly larger than the requested m/y; this size difference is m/y internal to a partition, but not being used.

Eg: A minimum fragment size is set to 2 KB.

free Space of 105 KB

Process of 104 KB



Here, 105 KB is fully allocated & extra space is kept unused.

(1 KB < 2 KB)

* 50-Percent Rule :- First fit Analysis reveals that given N blocks allocated, 0.5 N blocks lost to fragmentation 1/3 may be unusable

ie; Unusable = $\frac{0.5N}{1.5N} = \frac{1}{3}$
M/y

N-blocks + 0.5 N-blocks
↓ allocated. ↳ Lost to Fragment.

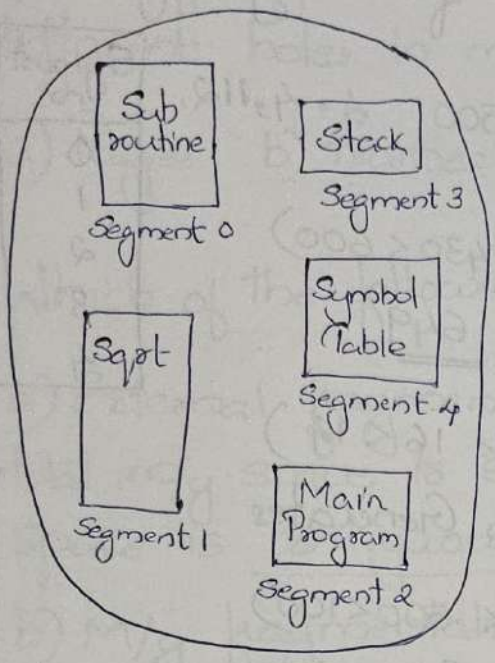
= 1 + 0.5

= 1.5 N-block → Total block Size.

2 Segmentation

* A Memory-management scheme that supports this programmer view of memory.
* A logical address space is a collection of segments.
  - Consist of 2 tuples
    <segment_number, offset>.
* Segment Table → Maps 2-D logical address into 1-D physical address.
  - Base Address : It contains the starting physical address where the segments reside in m/y.

• Limit : It specifies the length of the segment.



Logical Address Space

	Limit	Base
0	1000	1400
1	400	6300
2	400	4300
3	1100	3200
4	1000	4700

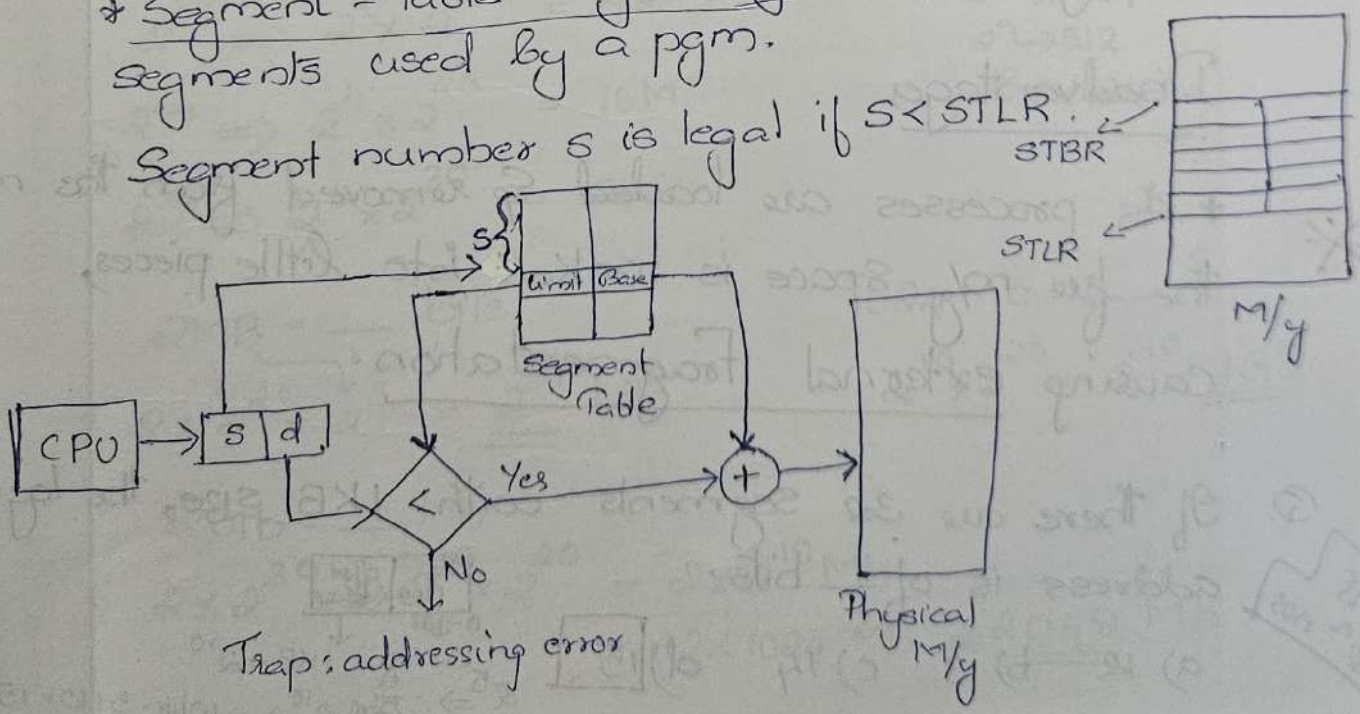
Segment Table.

If the pgm is small, segment table size is also small.
& if large, segment table size is large.
∴ if the CPU register is enough to store the segment table then it is stored & if not it stored in memory.

* Segment-Table base register (STBR) points to the segment table's location in memory. (Starting address)

* Segment-Table length Register (STLR) indicates no: of segments used by a pgm.

Segment number s is legal if s < STLR.



Q. Consider the following segment table : What are the physical addresses for the following logical addresses?

1. 0,430 2. 1,16 3. 2,500 4. 4,112.

Segment no:	Base	length
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

1. 0,430 => 219 + (check 430 < 600)
219 + 430 = 649

2. 1,16 => 2300 + (check 16 < 14)
Trap Generates

3. 2,500 => 90 + (check 500 < 100)
Trap Generates

4. 4,112 => 1952 + (check 112 < 96)
Trap Generates.

Advantage

* No Internal fragmentation.
* Segment Table consumes less space in comparison to page table in paging.

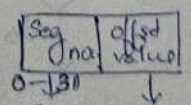
Disadvantages

* As processes are loaded & removed from the m/y the free m/y space is broken into little pieces, causing external fragmentation.

Q. If there are 32 segments with 1KB size, the logical address is of — bits.

a) 12 b) 13 c) 14 d) 15

bits 2^n nos



32 seg
$2^5 \Rightarrow \therefore 5$ bits $2^{10} \Rightarrow \therefore 10$ bits $\therefore 5+10 = 15$ bits

1 Byte = 8 bits

1 KB = 2^{10}

1 MB = 2^{20}

1 GB = 2^{30}

$2^{12} \Rightarrow 2^2 \times 2^{10} = 4KB$

$2^{24} \Rightarrow 2^4 \times 2^{20} = 16MB$

$2^{36} \Rightarrow 2^6 \times 2^{30} = 64GB$

$2^0 \rightarrow 1$
$2^1 \rightarrow 2$
$2^2 \rightarrow 4$
$2^3 \rightarrow 8$
$2^4 \rightarrow 16$
$2^5 \rightarrow 32$
$2^6 \rightarrow 64$
$2^7 \rightarrow 128$
$2^8 \rightarrow 256$
$2^9 \rightarrow 512$
$2^{10} \rightarrow 1024$

2MB = ___ bits

$2 \times 2^{20} \times 2^3 = \underline{\underline{2^{24}}}$ bits

1024 Mb = ___ ?

$2^{10} \times 2^{20} = 2^{30} = \underline{\underline{1Gb}}$

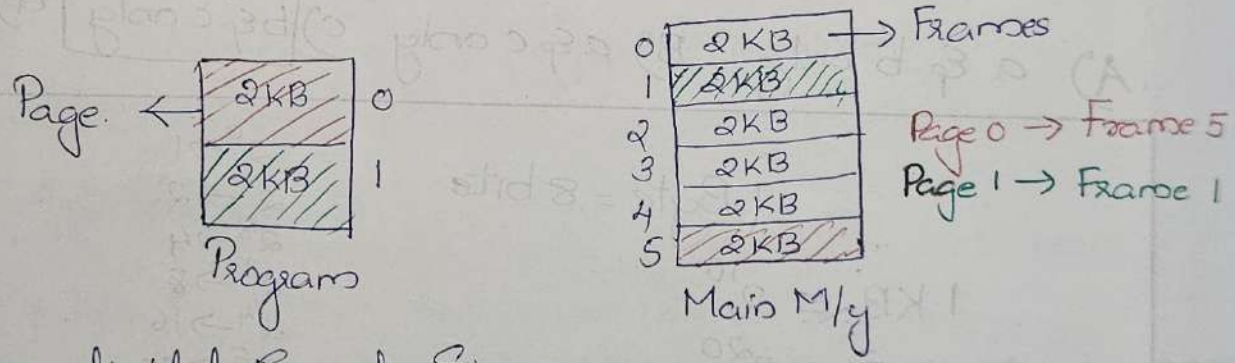
2Gb = ___ Mb

$2 \times 2^{30} = 2 \times 2^{10} \times 2^{20} = 2 \times 2^{10}$ Mb

$= 2 \times 1024$ Mb $= \underline{\underline{2048}}$ Mb

Paging

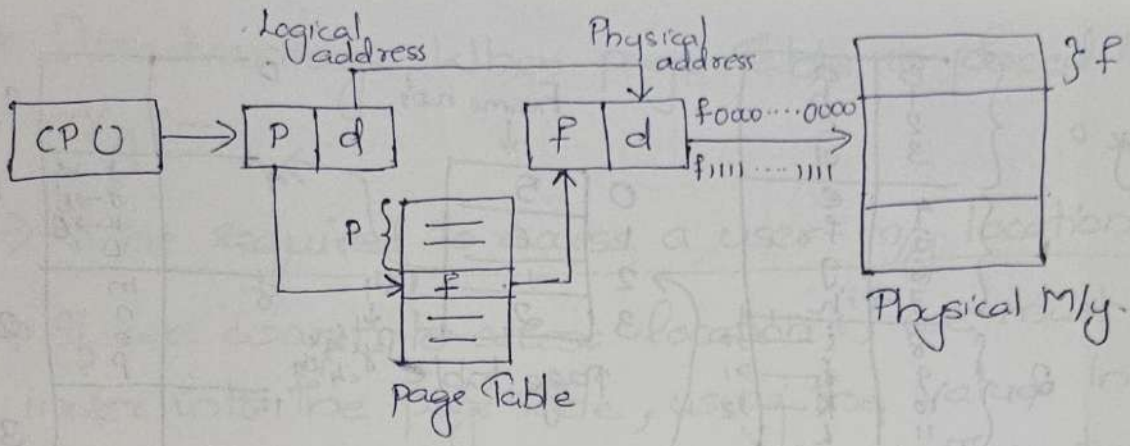
* Memory Management Scheme that eliminates the need for contiguous allocation of physical memory.
* Permits the physical address space of a process to be non-contiguous.
* The Basic method for implementing paging involves breaking physical m/y into fixed sized blocks called frames & breaking logical memory into blocks of the same size called pages.



Both are divided Equal Sizes

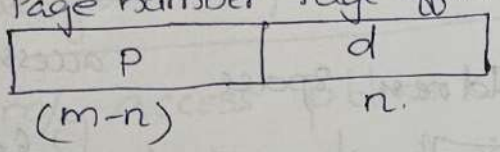
∴ no external fragmentation

* Every address generated by the CPU is divided into 2 parts : a page number (p) & a page offset (d)
* The page number is used as an index into a page table.
* The page table contains the base address of each page in physical m/y.
* This base address is combined with the page offset to define the physical m/y address that is sent to the m/y unit.



* The size of a page is a power of 2.
* The selection of a power of 2 as a page size makes the translation of a logical address into a page no: & page offset particularly easy.

* If the size of the logical address space is 2^m & a page size is 2^n bytes, then the high-order $(m-n)$ bits of a logical address designate the page number & the n low-order bits designate the page offset.



Logical address space = m
page size = n

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

1. How many bits are required in the logical address?
2. How many bits are required in the Physical address?

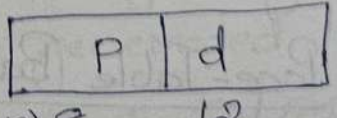
Logical Address Space

No: of Pages $= 256 = 2^8 =$ 8 bits

Page Size $= 4\,KB = 2^2 \times 2^{10} = 2^{12}$ Bytes $= 12$ Bits

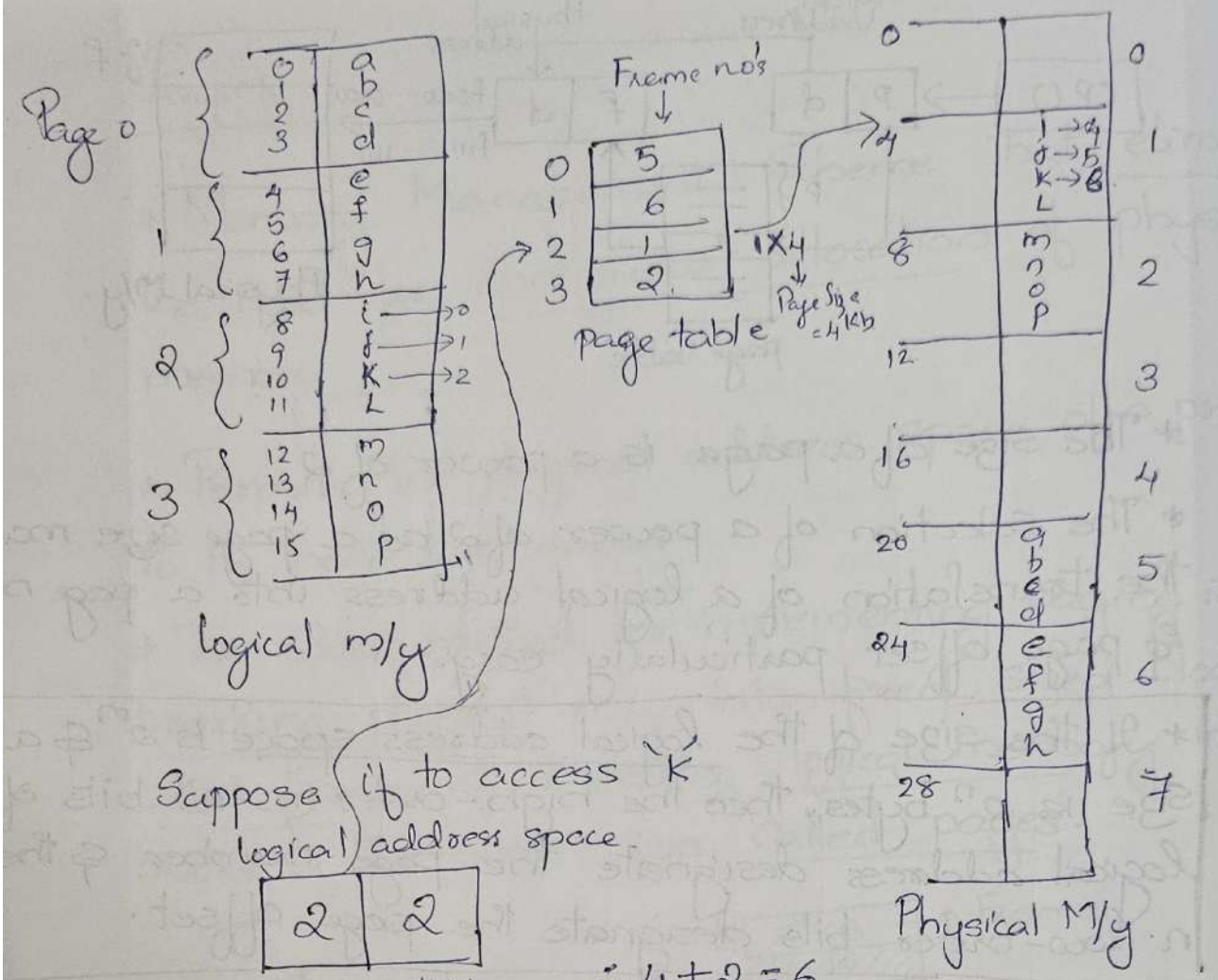
(Page Size = Frame Size)

Frames $= 64$ frames $= 2^6 =$ 6 Bits (Frame no)



$12 + 8 = 20$ bits

$\hookrightarrow 2^{20}$ logical address locations / space.



Suppose if to access 'K'

Logical address space.

2	2

Page no offset

∴ 4+2 = 6

Physical address space.

6	12

f d.

Frame no. Frame offset

6+12 = 18 Bits for Physical Address Space

access 'e'

1	0

6 ↳

6X4 = 24

24+0 = 24

access 'f'

1	1

6 ↳

6X4 = 24

24+1 = 25

* Smaller page tables are implemented as a set of regist
* The page table registers should be built with very high speed logic.
* Larger page tables are kept in main m/y.
* Registers associated with page table are :-
  - Page-Table Base register (PTBR) : Points to the Page Table
  - Page-Table length register (PTLR) : indicates size of the page table.

* Disadvantage when page Table is placed in main m/g :-

→ Time required to access a user m/g location is more.

→ If we want to access location i, we must first index into the page table, using the value in the PTBR offset by the page no. for i. This task requires a m/g access.

→ It provides us with the frame number, which is combined with the page offset to produce the actual address.

* Here; every data / instruction access requires 2 m/g accesses.

1. One for page table
2. One for data / Instruction.

* These 2 m/g access p blms can be solved using a special fast-lookup hardware cache called Associative Memory or Translation Look-Aside Buffers (TLBs)

(Locality of reference)

* Each entry in TLB consists of 2 parts : a key (or tag) & a value.

Key → Page no

Value → Frame no.

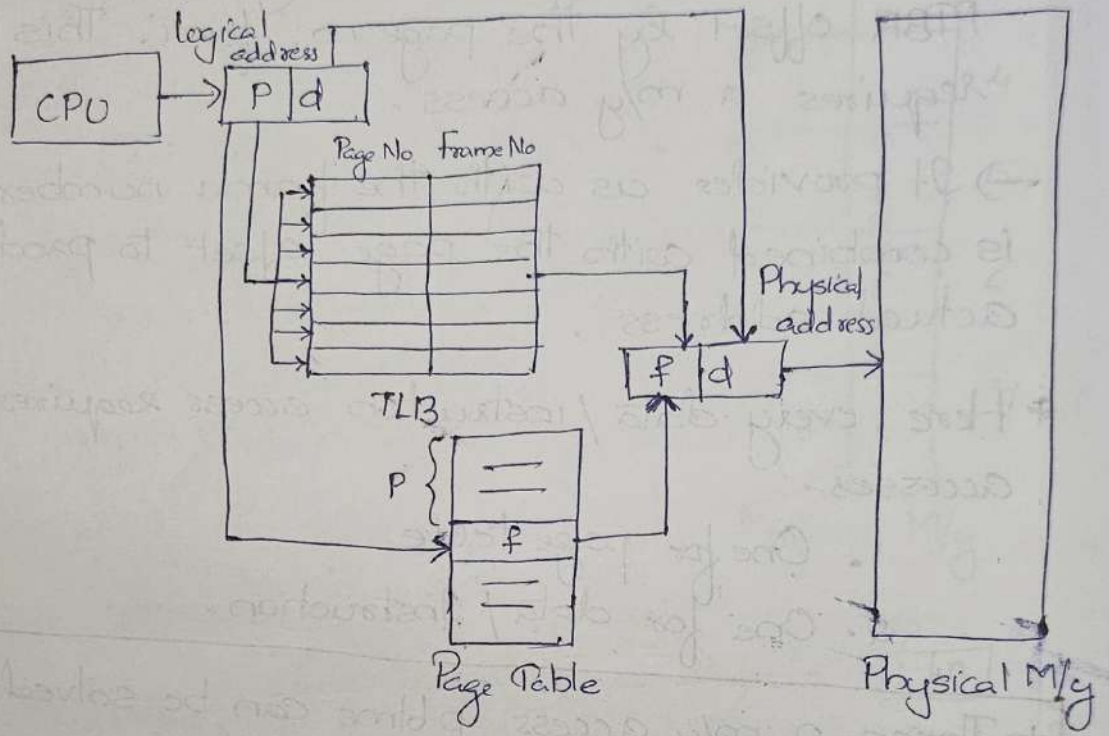
* TLB contains only a few of page-table entries.

* When a logical address is generated by the CPU, its page number is presented to the TLB.

(TLB hit)

* If the page no: is found, its frame number is immediately available and is used to access memory.

* If the page no: is not in the TLB (Known as TLB miss) a m/y reference to the page table must be made.



If in TLB → 1 m/y access (TLB hit)

Not in TLB → 2 m/y access (TLB Miss)

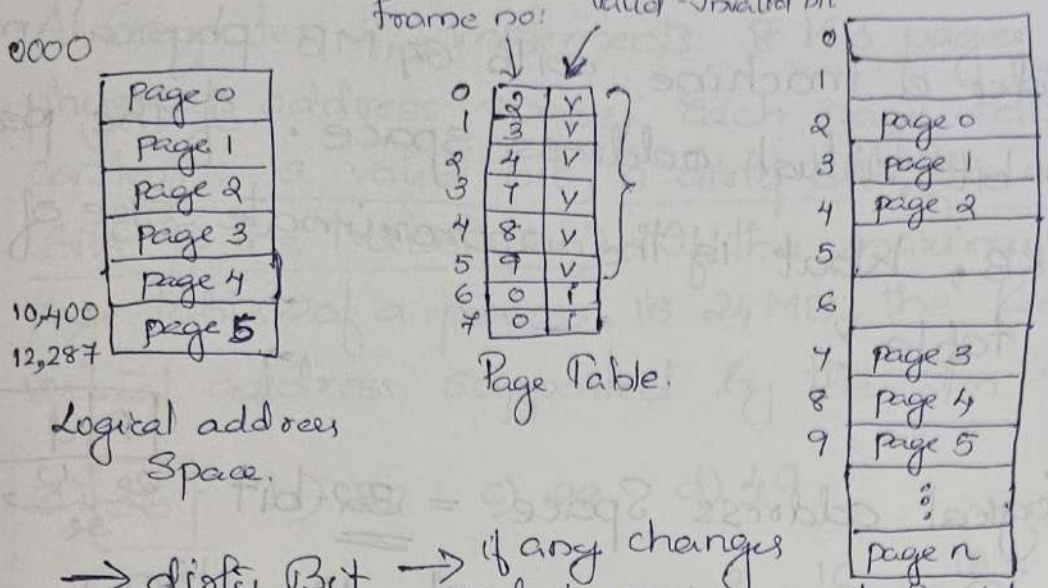
Paging - Memory Protection

* Valid - Invalid bit is attached to each entry in the page table :

Valid ⇒ the associated page is in the process logical address space & ∴ a legal page.

Invalid ⇒ the page is not in the process logical address space.

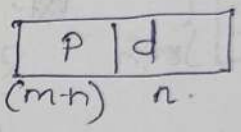
Any violation result in a trap to the Kernel.



→ dirty Bit → if any changes made in main pgm it should be reflected to physical m/y & will be set to 1. else 0.

→ Permission Bit

Q. Consider a Computer S/m with a 32-bit logical address & 4 KB page size. How many entries are there in Single level page table?



Logical address Space = 32-bit

Page Size = 4 KB = $2^2 \times 2^{10}$ B = 2^{12} B = 12 bits

$m = (m-n) + n$

$32 = (m-n) + 12$

$(m-n) = 32 - 12 = 20$ bits

$\therefore 2^{20}$ entries

Q. If it takes 100 ns to access memory & 80% of m/y Operation is a TLB hit. Then the effective m/y access?

TLB hit = 0.8 (80%)

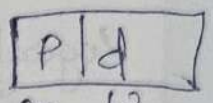
∴ TLB Miss = (1 − 0.8) = 0.2 (20%)

Average m/y Access Time = $(0.8 \times 100) + 0.2 \times (100 + 100)$

(TLB hit) 1 m/y access; (TLB Miss) 2 m/y access

$80 + 40 = 120$ ns

Q. Consider a machine with 64 MB physical m/y & a 32-bit virtual address space. If the page size is 4KB, what is the approximate size of the page table?



Logical address Space = 32-bit.

Page Size = 4KB = $2^2 \times 2^{10} = 2^{12}$ Bytes = 12 bits

Physical address Space = 64 MB = $2^6 \times 2^{20}$

$= 2^{26} = 26$ bits

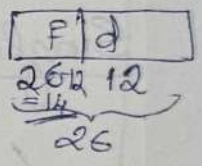
m = (m-n) + n

32 = (m-n) + 12

(m-n) = 32-12

(m-n) = 26-12

= 14



No: of Pages = 20 bits = 2^{20} entries

Size of page Table = No: of entries (Page no's) * Size of each entry

∴ Size of page Table = $2^{20} \times 2^{(14)}$ = 2 MB

fno	Pgno	v/i	dirty

1 Byte = 8 bits

2 Bytes = 16 bits. ∴ ≅ 2 Bytes = 14 bits

Assume more one.

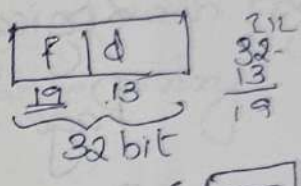
a) 16 MB b) 8 MB c) 2 MB d) 24 MB.

Q. A Computer S/m implements 8 KB pages & a 32-bit physical address space. Each page table entry contains a valid bit, a dirty bit, three permission bits & the translation. If the maximum size of the page table of a process is 24 MB, the length of the virtual address supported by the S/m is ___ bits.

a) 36 b) 32 c) 28 d) 40.

Frame Size = Page Size = 8 KB. $= 2^3 \times 2^{10} = 2^{13} = 13$ bits

Physical address Space = 32-bit



32 − 13 = 19

Size of Page Table = 24 MB $= 24 \times 2^{20} \times 2^3$

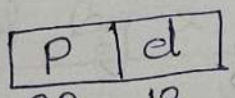
Virtual address Space. Length (d) = ?

Size of page Table = No: of entries × Size of each entry.

$24 \times 2^{20} \times 2^3$ = No: of entries × 24 bits {3 Bytes}

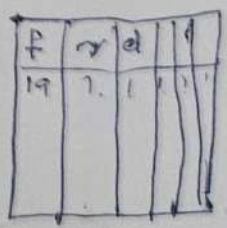
1 Byte - 8 bit
2 Byte = 16 bit
3 Byte = 24 bit

No: of entries $= \frac{24 \times 2^{20} \times 2^3}{24} = 2^{23} = 23$ bits



23 + 13 = 36 bits.

26 − 13 = 19



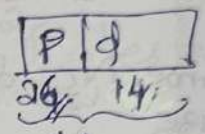
19 + 1 + 1 + 1 + 1 + 1

Each PT entry Size = 24

$\frac{24}{8} = 3$ //

Q. Consider a computer s/m with 40-bit virtual addressing & page size of 16 KB. If the computer s/m has a one-level page table per process & each page table entry requires 48 bits, then the size of the per-process page table is — MB.?

a) 384 b) 48 c) 192 d) 96.



Logical Address Space = 40 bits

Page Size = 16 KB = $2^4 \times 2^{10} = 2^{14}$ = 14 bits

Page Table Entry = 48 bits = $\frac{48}{8}$ = 6 Bytes

Page Table Size = ? MB

$m = (m-n) + n$

$40 = (m-n) + 14$

$(m-n) = 40-14$

No. of Pages = 26 bits = 2^{26} Bytes

Page Table

of Page Table. = No. of entries × Size of each entries

$= 2^{26} \times 6$

$= 2^{20} \times 2^6 \times 6$

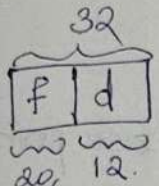
$= 64 \times 6$ MB

$= 384$ MB

64
6
384

Q If the page size in a 32-bit machine is 4KB then the size of page table is:

a) 1 MB b) 2MB c) 4MB d) 4KB.

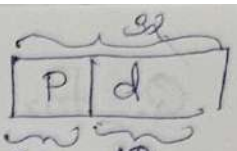


Physical Address Space = 32-bit.

Logical Address Space = 32-bit.

Page Size = 4 KB = $2^2 \times 2^{10} = 2^{12}$ = 12 bits

Size of Page Table = ?



$m = (m-n) + n$

$32 = (m-n) + 12$

$(m-n) = 32 - 12 = 20$ $\Rightarrow$ No: of Pages. $= 2^{20}$ entries

Size of Page Table = No: of entries × Size of each page Table entry.

$= 2^{20} \times 3$ {20 bits}.

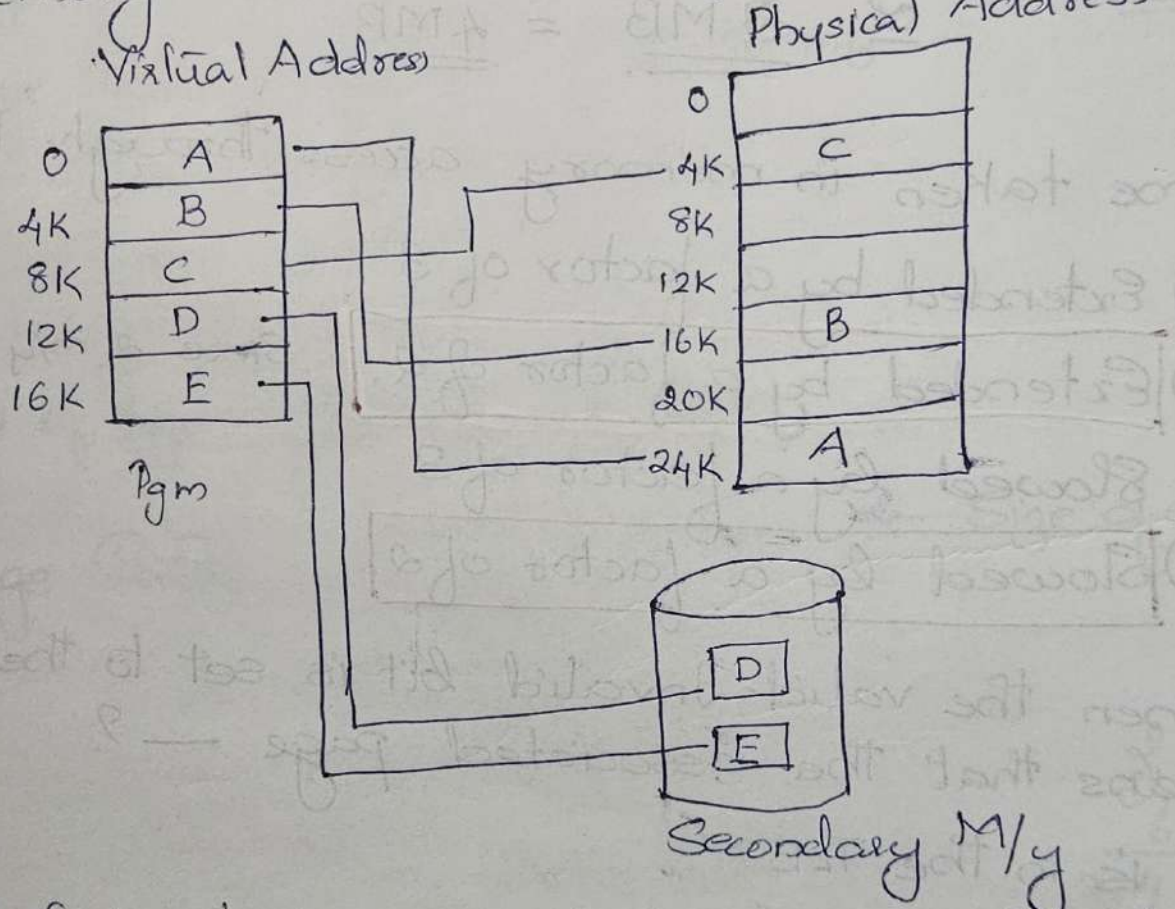
$\simeq$ 3 MB = 4 MB.

e) page table register

Virtual Memory

{ If the size of the pgm is > the size of our s/m m/y. }

* Storage Scheme that provides user an illusion of having a very big main memory.
* This is done by treating a part of secondary memory as the main m/y.



Initially, here pages A, B & C are placed in main m/y.

Demand Paging

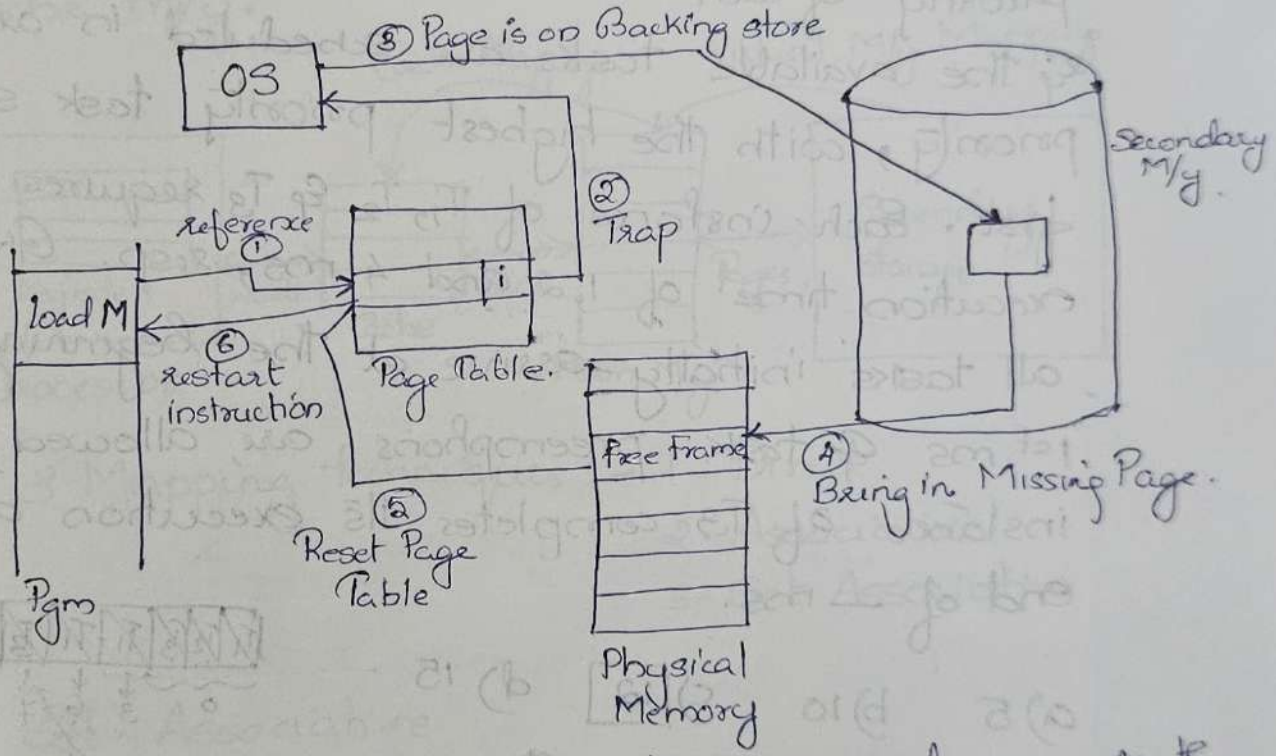
↳ According to needs only pages are placed in main m/y for execution.

If the main m/y is full & so space for new page to be placed, then existing page is replaced by the demanded one using page replacement algorithm.

Demand Paging

* The process of loading the page into m/y on demand (whenever a page fault occurs).
* The process includes the follo: steps :-

1. If CPU try to refer a page that is currently not available in the main memory, it generates an interrupt indicating m/y access fault.
2. The OS puts the interrupted process in a blocking state, for the execution to proceed the os must bring the required page into the m/y.



3. The OS will search for the required page in the logical address space.
4. The required page will be brought from the logical address space to physical address space. The page replacement algthms are used for the decision making of replacing the page in physical address space.

Advantage :- 1. More Efficient obligation of the processor.
2. A process larger than the main m/y can be executed.
3. Greater Multiprogramming by using less of available m/y for each process. Primary

Demand Paging

* The time taken to service the page fault is called as page fault service time.
* The page fault service time includes the time taken to perform all the above 6 steps.
* Let the main m/y access time is : m
  Page fault service time is : s
  Page fault rate is : p

∴ Effective M/y Access Time = (p * s) + (1-p) * m

Q. In a paged m/y, the page hit ratio is 0.35. The Time required to access a page in secondary m/y is equal to 100 ns. The time required to access a page in primary memory is 10 ns. The average time required to access a page is ?

a) 3.0 ns b) 68.0 ns c) **68.5 ns** d) 78.5 ns.

Page hit ratio = 0.35 (1-P)

∴ P = 1 - 0.35 = 0.65

Secondary m/y access (s) = 100 ns

Primary m/y access (m) = 10 ns

Average Time = ?

= (0.65 x 100) + (0.35 x 10)

= 65 + 3.5

= 68.5 ns

Demand Paging

* The time taken to service the page fault is called as page fault service time.
* The page fault service time includes the time taken to perform all the above 6 steps.
* Let the main m/y access time is : m
  Page fault service time is : s
  Page fault rate is : P

∴ Effective M/y Access Time = (P * s) + (1-P) * m

Q. In a paged m/y, the page hit ratio is 0.35. The Time required to access a page in secondary m/y is equal to 100 ns. The time required to access a page in primary memory is 10 ns. The average time required to access a page is ?

a) 3.0 ns b) 68.0 ns c) **68.5 ns** d) 78.5 ns.

Page hit ratio = 0.35
(1-P)

∴ P = 1 - 0.35 = 0.65

Secondary m/y access (s) = 100 ns

Primary m/y access (m) = 10 ns

Average Time = ?

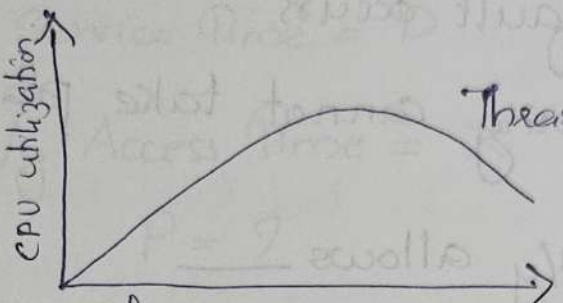
= (0.65 x 100) + (0.35 x 10)

= 65 + 3.5

= 68.5 ns

Thrashing

* Condition or a situation when the s/m is spending a major portion of its time in servicing the page faults, but the actual processing done is very negligible.



* The basic concept involved is that if a process is allocated too few frames, then there will be too many & too frequent page faults.
* As a result, no useful work would be done by the CPU & the CPU utilization would fall drastically.
* The long-term scheduler would then try to improve the CPU utilization by loading some more processes into the m/y thereby increasing the degree of multiprogramming.
* This would result in a further decrease in the CPU utilization triggering a chained reaction of higher page faults followed by an increase in the degree of multiprogramming called Thrashing.

Q. Consider a process executing on an os that uses demand paging. The average time for a m/y access in the s/m is M units if the corresponding memory page is available in m/y & D units if the m/y access causes a page fault. It has been experimental measured that the average time taken for a m/y access in the process is X units. Which one of the follo: is the correct expression for the page fault rate experienced by the process?

a) (D-M) / (X-M) b) (X-M) / (D-M)

c) (D-X) / (D-M) d) (X-M) / (D-X)

Memory Access Time = M.

Page fault Service Time = D

Effective M/y Access Time = X.

P = ?

$X = (P * D) + (1-P) * M$.

$X = PD + M - PM$

$X = P(D-M) + M$

$(X-M) = P(D-M)$

$P = \frac{(X-M)}{(D-M)}$ //.

Q. Let the page fault service time be 10ms in a computer with average m/y access time being 20ns. If one page fault is generated for every 10^6 m/y accesses, what is the effective access time for the memory?

a) 21 ns b) 30 ns c) 23ns d) 35 ns.

Page fault Service time (S) = 10 ms $= 10 \times 10^{-3} \times \frac{10^{-6}}{10^{-6}}$

Average m/y access time (m) = 20 ns

page fault rate = 1 for 10^6 m/y access

(P) = $\frac{1}{10^6}$ ∴ (1-P) = $(1 - \frac{1}{10^6})$

Effective Access Time = ?

$X = \left(\frac{1}{10^6} \times 10 \times 10^{-3} \times \frac{10^{-6}}{10^{-6}}\right) + \left(1 - \frac{1}{10^6}\right) \times 20$

$= 29.999 \text{ ns} \cong 30 \text{ ns}$ //

4/2022

Reference String

* The string of m/y references is called reference string.
* Reference strings are generated artificially or by tracing a given s/m and recording the address of each m/y reference.
* For eg: Consider the follo: sequence of addresses:-

  123, 215, 600, 1234, 76, 96.

* If page size is 100, then the reference string is

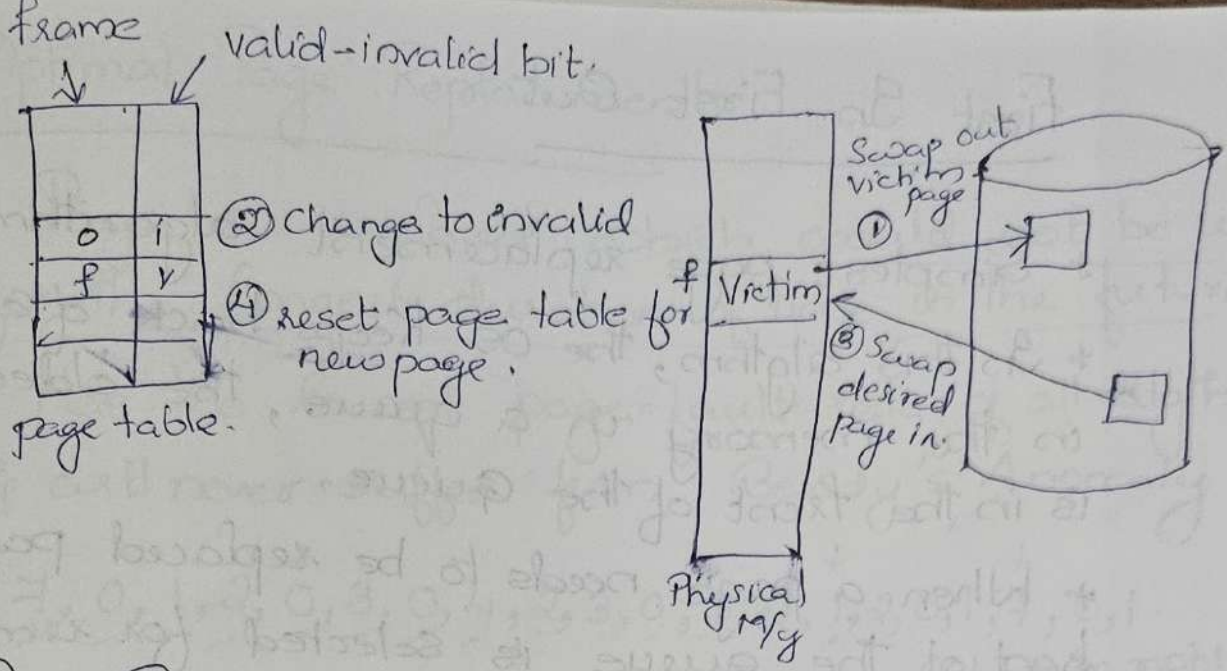
  1, 2, 6, 12, 0, 0.

ie; 1-199, 200-299, 600-699 .... etc.

↓ Page 1 ↓ 2 ↓ 6

Page Replacement

* If no frame is free, we find one that is not currently being used & free it.
* Find the location of the desired page on the disk.
* Find a free frame:
  → If there is a free frame, use it.
  → If there is no free frame, use a page replacement algorithm to select a victim frame.
  → Write the victim frame to the disk; change the page & frame tables accordingly.
  → Read the desired page into the newly freed frame; change the page and the frame tables.
  → Continue the user process from where the page fault occured.



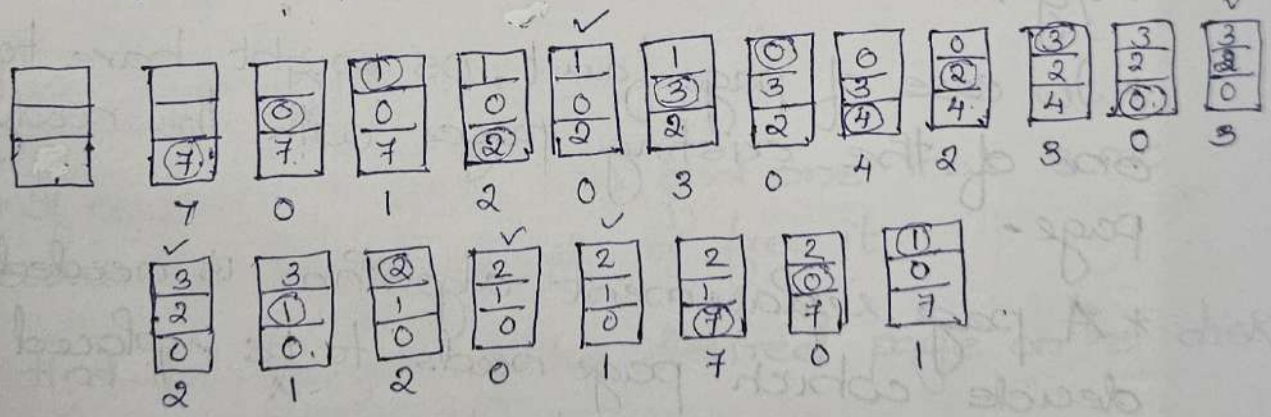
Page Replacement Algorithms

* A page fault happens when a running pgm accesses a m/y page that is mapped into the virtual address space, but not loaded in physical m/y.
* In case of page fault, OS might have to replace one of the existing pages with the newly needed page.
* A page replacement algorithm is needed to decide which page needs to be replaced when the new page comes in.
* Diff: page replacement algthms suggest diff ways to decide which page to replace.
* Used to reduce the no: of page faults.
* Page Replacement Algthms are :-
  1. First In first out (FIFO)
  2. Optimal Page Replacement
  3. Least Recently used (LRU)

First In First Out

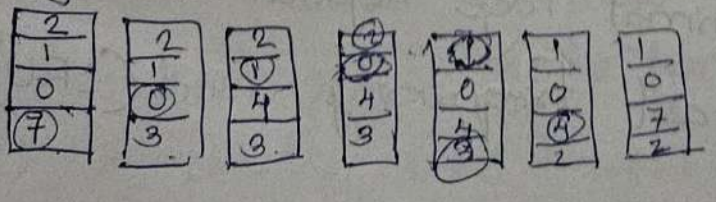
* Simplest page replacement algorithm.
* In this algthm, the OS keeps track of all pages in the memory in a queue, the oldest page is in the front of the Queue.
* When a page needs to be replaced page in the front of the queue is selected for removal.

Q. How many page fault occur if FIFO page replacement is used when following pages are requested. Consider no: of frames available is 3 & initially all are empty: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.



15

* Belady's Anomaly proves that it is possible to have more page faults when increasing the no: of page frames while using the FIFO page replacement algthm.

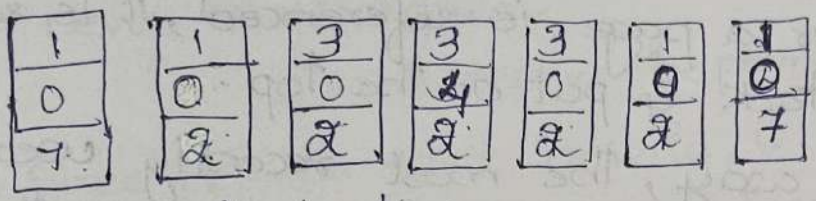


Optimal Page Replacement

* Pages are replaced which could not be used for the longest duration of time in the future.
* Has the lowest page-fault rate of all algorithms & will never suffer from Belady's Anomaly.

Q: 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.

(Check to right side)



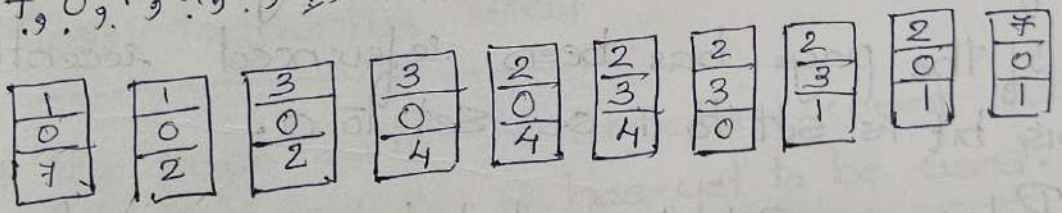
7 used after long time

9// Last pages are 7, 0, 1

Least Recently Used

* Check to left side.

Q. 7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2, 1, 2, 0, 1, 7, 0, 1.



12//

* Two implementations are feasible :-

1. Counters :-

→ In the simplest case, we associate with each page table entry a time-of-use field & add to the CPU a logical clock or counter.

→ The clock is incremented for every m/y reference.

→ Whenever a reference to a page is made, the contents of the clock register are copied to the time-of-use field in the page-table entry for that page.

→ ie; the counter having Loca value is used.

2. Stack :-

→ To keep a stack of page numbers.

→ Whenever a page is referenced, it is removed from the stack & put on the top.

→ In this way, the most recently used page is always at the top of the stack & the least recently used page is always at the bottom.

Reference Bit { Used in LFU & MFU }.

* Reference bit specifies whether that page has been referenced in the last clock cycle or not.

* If the page has been referenced recently, then this bit is set to 1 ow set to 0.

* Reference Bit is useful for page replacement policy.

* A page that has not been referenced recently is considered a good candidate for page replacement in LRU page replacement policy.

1. Least Frequently Used (LFU)
2. Most Frequently Used (MFU)

Least Frequently Used.

* Requires that the page with the smallest count be replaced. (i.e; counter value low is used.)
* The reason for this selection is that an actively used page should have a large reference count.
* A pblm arises when a page is used heavily during the initial phase of a process but then is never used again.

↓ so to resolve the pblm

* Since it was used heavily, it has a large count and remains in m/y even though it is no longer needed.
* One solution is to shift the counts right by 1 bit at regular intervals, forming an exponentially decaying average usage count.

Eg. 1011 → 1102 decaying
right shift ⇒ 0101 → 5

Most Frequently Used

* The page with the smallest count was probably just brought in & has yet to be used.