* For execution : code – in memory

But entire code in memory – error code
↓
loaded

- limit of physical memory
- more memory
- large DS

Solution:

execute partially loaded program

- less memory
- runs faster (less I/O need)
- more programs at same time
- increase CPU utilization
- increase throughput
- no increase → response time.

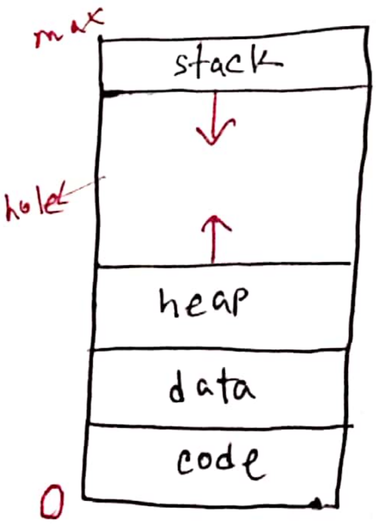
* VM: separation of user logical memory from physical memory

- execute partially loaded program

* Virtual Address Space: logical view of how process is stored in memory

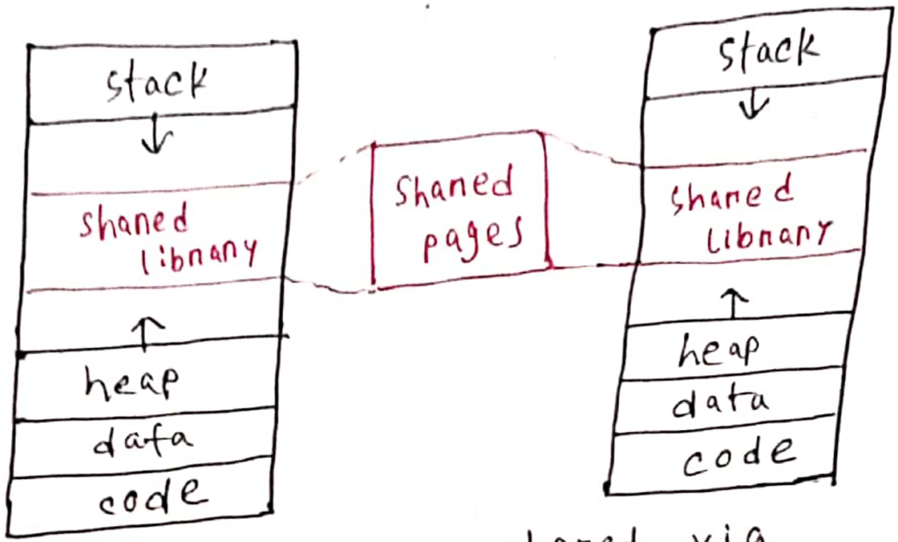
implemented in 2 ways:

1) demand paging
2) demand segmentation



vm maps to PM ; VM > PM
↓
physical

* Shared library using VM:



(system library shared via mapping into virtual address space)

* Demand paging:

need to execute an instruction from page
→ demand for that page
→ load a page into main memory
(instead of entire process)

Less memory
Less I/O
Fast response
more users

Step: when page needed → reference to the page

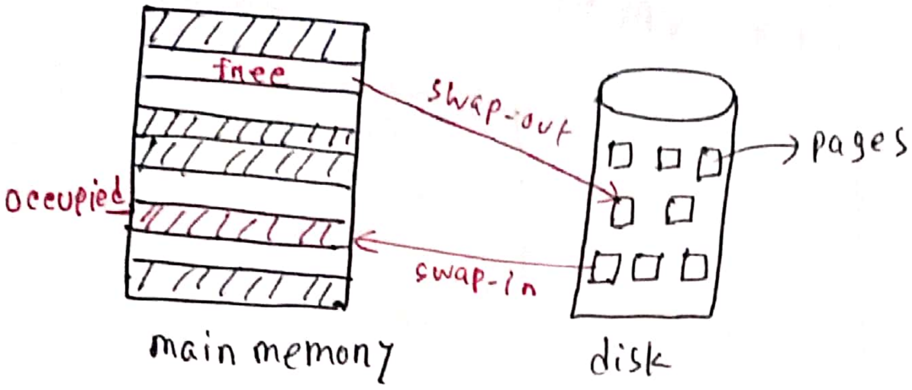
3 cases:
1) invalid refer — abort (ক্ষান্ত) - terminate
2) page not in memory — bring it to mem
3) page in memory — use it

* Lazy swapper: never swaps a pages into memory unless page will be needed

* Pager: swapper that deal with pages

* Pure demand paging
* Locality of reference

Hardware needed for demand paging:
1) PTE with valid/invalid bit
2) secondary memory
3) instruction restart



Pros:
- less memory
- less swap time
- increase degree of multiprogramming

Cons: page fault
- page not in memory

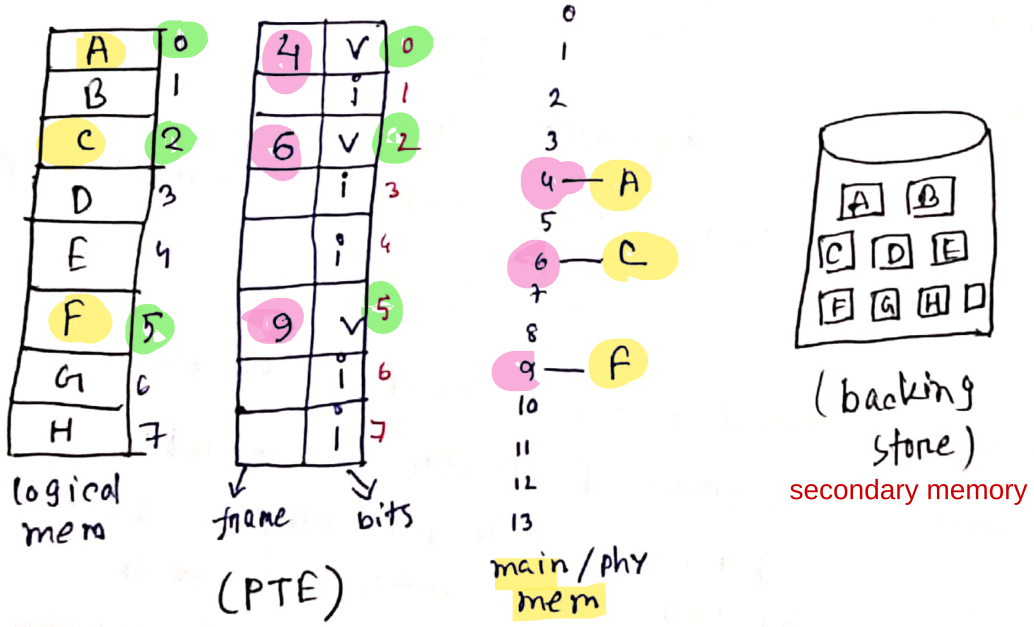
fig: Lazy swapper (swap pages when needed)

* Valid - invalid bit:

v - in memory

i - not in memory
causes page fault

Initially, all bits - invalid (i)
in page table entry (PTE)



* Steps in handling page fault:

1) reference to a page
2) page not in memory → set i bits in PTE
3) look for missing page in the backing store (disk)
4) bring it to free frame of main/physical memory.

5) reset PTE → set bits 'v'

6) restart

* No free frame in main memory
  - use page replacement algorithm
    - decrease page faults

Page replacement: refers to a scenario in which a page from the main memory replaced by a page from secondary memory

modify page fault ← - prevents over-allocation of memory
↓
uses more memory than is physically available

modify (dirty) bit ← - reduce overhead of page transfers

determines which page is to be replaced.

Frame allocation algorithm

- determines → 1) how many frame allocate
  2) which frame → replace

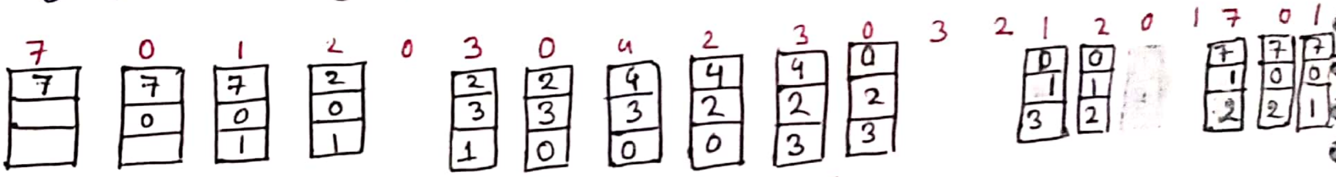
visit

* **FIFO Algo :** Replace page that comes in first

→ reference string of referenced page numbers :

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

→ 3 frames (3 pages in memory at a time per process)



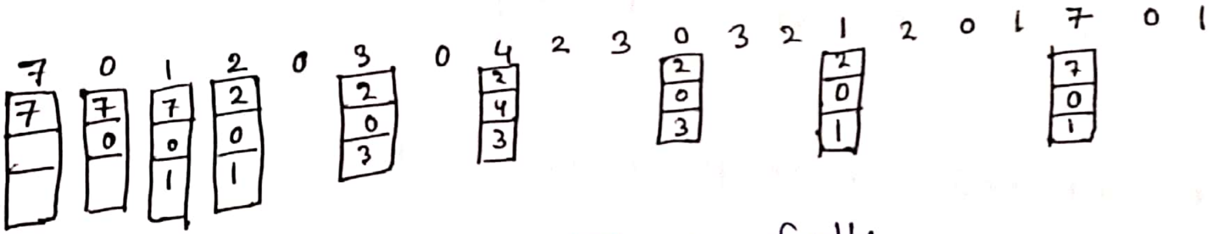
15 page faults

→ more frame → more page faults

known as : Belady's Anomaly

→ যেটা সবচেয়ে পরে use হবে – replace

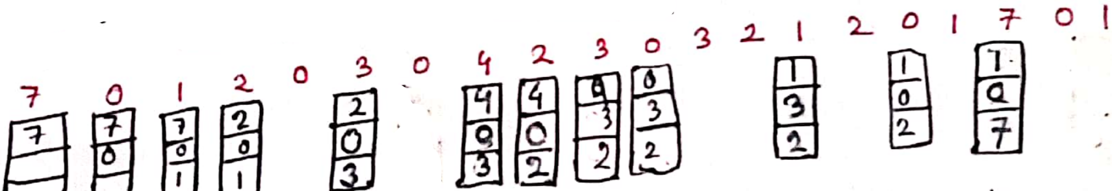
* **Optimal Algo :** Replace page → will not used for long period of time (in future)



→ optimal ans : 9 page faults

→ ~~যেটা সবচেয়ে আগে~~ use – replace

* **LRU (Least Recently Used) Algo :** Replace page – used earlier in past



→ 12 page faults – worse than optimal, better than FIFO

* Counting Algo:

keep a counter of number of references made to each page

1) LFU → Frequently used

L → Lease

Replace page with smallest count

2) MFU

M → most

Replace page with smallest count that

- Just আনা হয়েছে (brought in)
- not used yet

* 3 major activities of demand paging:
1) Service the interrupt
2) Read the page
3) Restart the process

* EAT = $(1-p) \times$ memory access + p (page fault overhead + swap page out + swap page in)

Here, EAT – Effective access time
1st part – hit , 2nd part – miss
p – page fault rate
$\boxed{0 \le p \le 1}$

$p = 0$; no page fault
$p = 1$; every reference is a fault

$\therefore 1-p =$ hit rate

Q: Memory access time = 200 ns
Avg. page-fault service time = 8 ms = 8×10^6 ns

$\therefore$ EAT = $(1-p) \times 200 + p \times 8 \times 10^6$
$= 7{,}999{,}800p + 200$

now, 1 access out of 1000 → page fault
$\therefore p = \frac{1}{1000} = 10^{-3}$

$\therefore$ EAT $= 7{,}999{,}800 \times 10^{-3} + 200$
$= 8{,}199.8$ ns
$= 8.2\ \mu s$

slow down by factor of $= \frac{\text{miss}}{\text{hit}} = \frac{10^{-3} \times 8 \times 10^6}{(1-10^{-3}) \times 200} = 40$

p p

* if performance degradation $< 10\% = \frac{10}{100}$

$\therefore \frac{\text{miss}}{\text{hit}} < \frac{10}{100}$

$\rightarrow \frac{8 \times 10^6 \times p}{(1-p) \times 200} < \frac{1}{10}$

$\rightarrow 8 \times 10^6 \times p < 20 - 20p$

$\rightarrow p < \frac{20}{8 \times 10^6 + 20}$

$\rightarrow p < 2.5 \times 10^{-6}$

$\rightarrow p < \frac{1}{0.4 \times 10^6}$

$\rightarrow p < \frac{1}{4 \times 10^5}$

less than 1 page fault in 4,00,000 mem. access