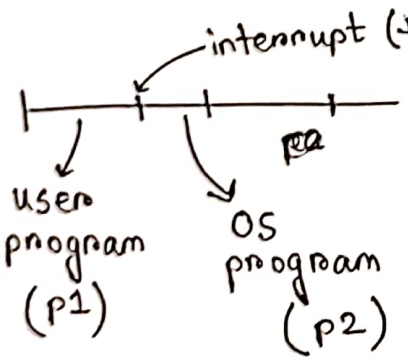
22.11.22

Virtualization of Memory

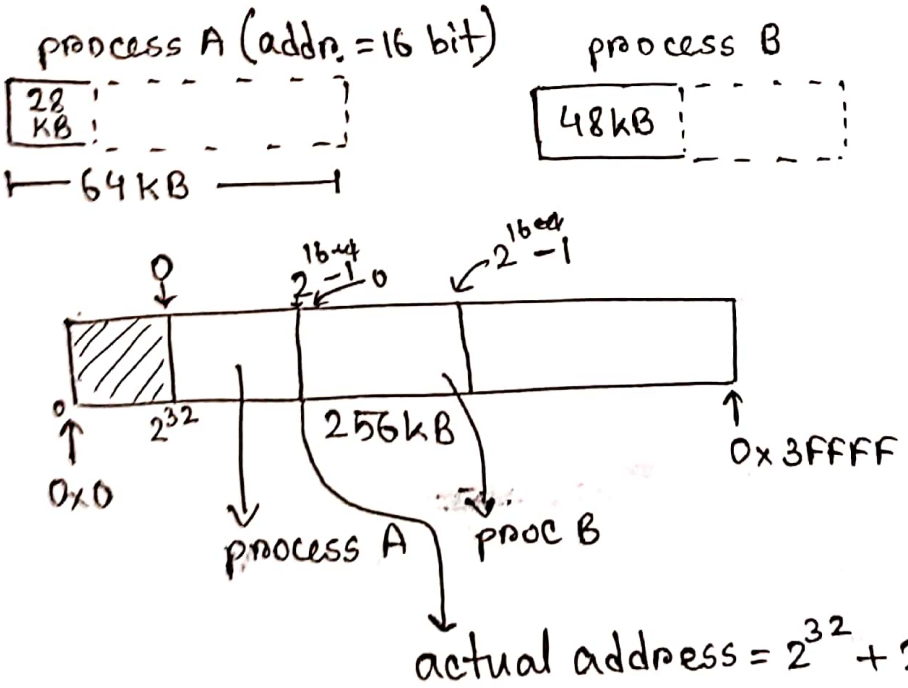


Physical Memory:

* physical address p bits ⇒ max. physical memory 2^p bytes.
* OS জানে কত্ত তার কাছে যতটুকু memory আছে।

Virtual Memory:

* প্রত্যেকটা program-এর নিজের virtual memory আছে।



actual address = $2^{32} + 2^{16} - 1$

Address Translation:

* base register + virtual addr. = physical addr.

এটা h/w কে বলে দিব, OS এটা manage করে না;

base register value ≡ offset

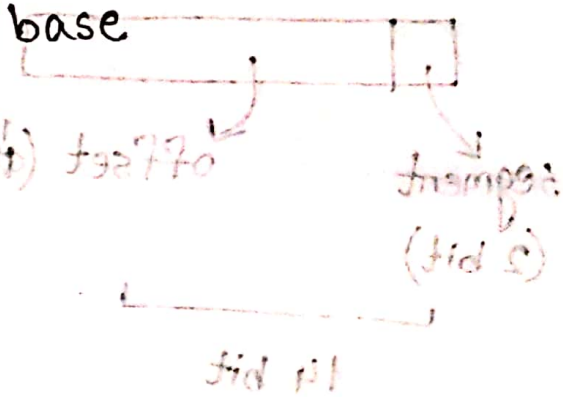
Dynamic Relocation:

* প্রতিটা process একবারবার run করার সময় একেক memory থেকে run করতে পারে।

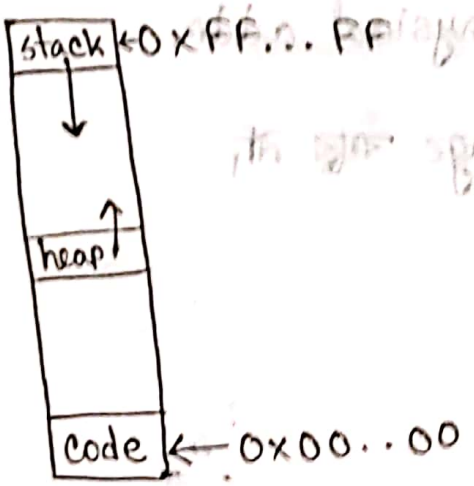
* Limit = amount of memory used by the process.

physical = virtual + relocation (when virtual < limit)

offset / base

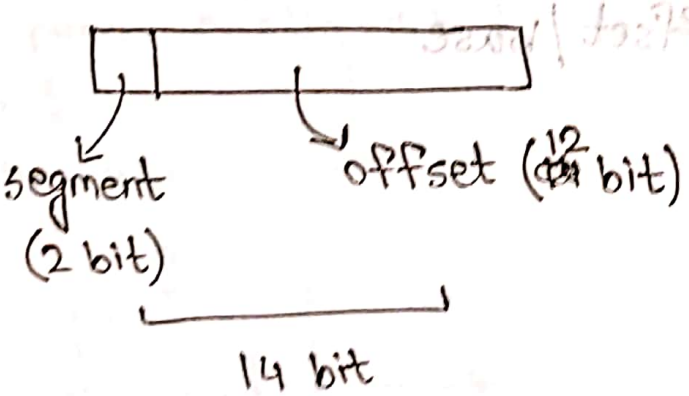


Realistic Virtual Memory:



* প্রতি program কে যতটুকু memory দিব তার একটা প্রক্রিয়া, segmentation হিসাবে দিব।

* segmentation fault occurs when program tries to access mem. addr. outside of its allocated memory space.



⇒ 1টা segment size 2^{12} byte

23.11.22

Translating Segmented Virtual Address:

Virtual address = V bit

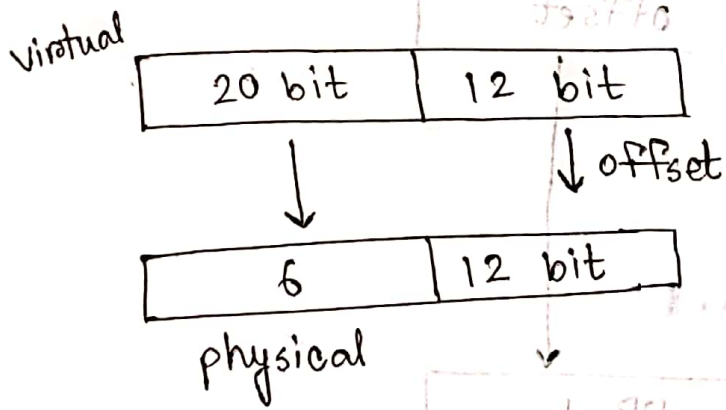
Physical address = P bit

4kB ⟶ page size = 2^{12} byte

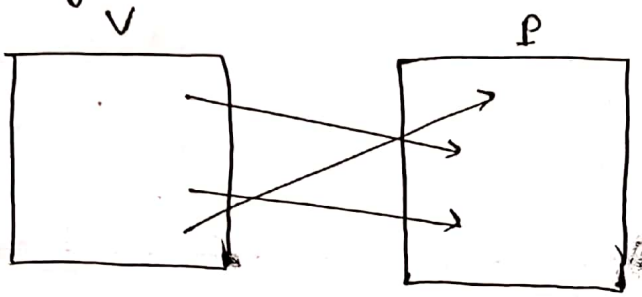
no. of page = $\frac{2^{32}}{2^{12}} = 2^{20}$

content size per page = 2^2 = 4 byte

total size = $2^{20} \times 2^2 = 2^{22}$ = 4 MB



Mapping:

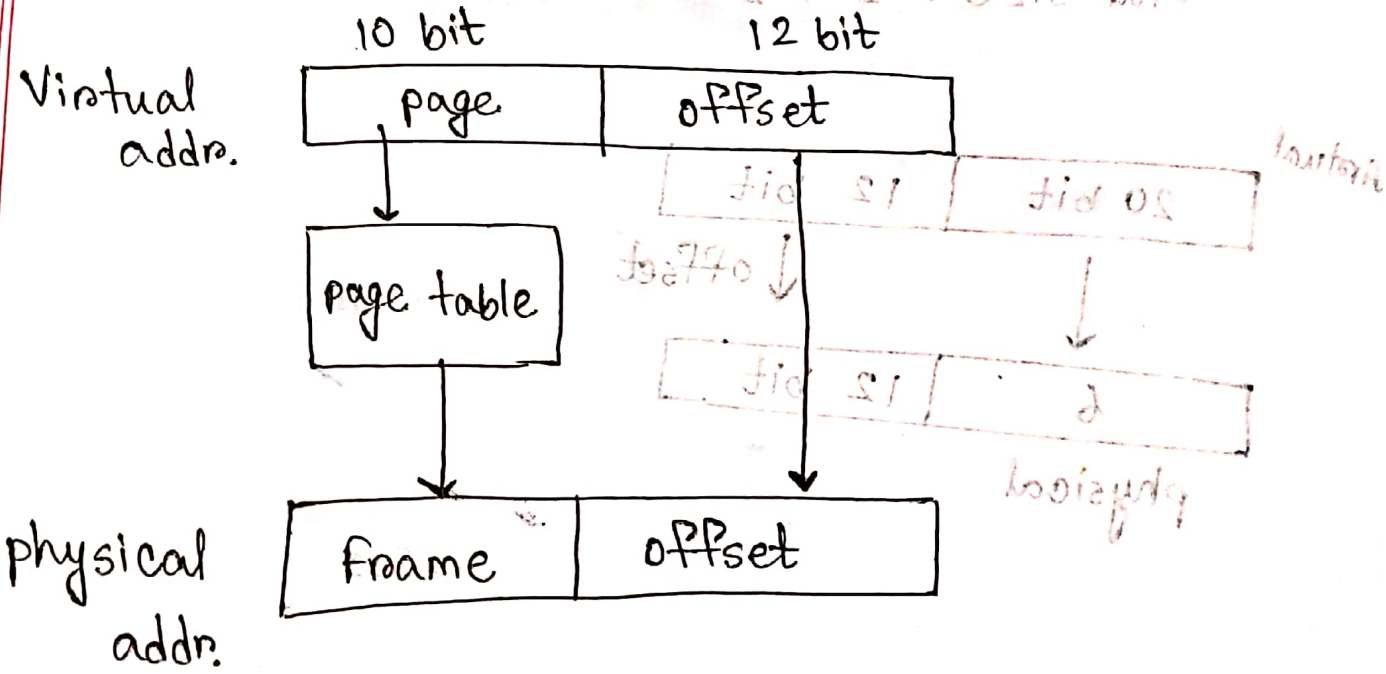


Page Table:

page table:

Frame	Valid
6 bit	1 bit

→ 1 byte enough



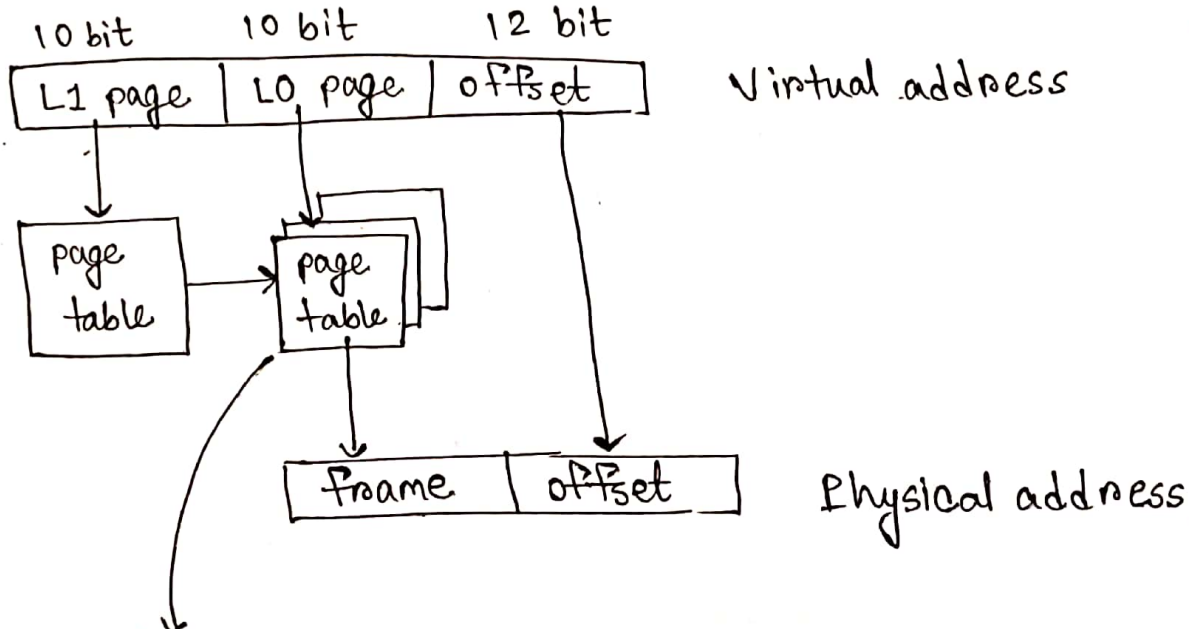
*page এর size যত, page table তার চেয়ে বড় হবে না,

page size = 4 kB ⟶ 12 bit for addressing

size of page table = size of page table entry (→ 4 byte (Assuming)) * no. of entries in page table

↙ 2^{12}

→ $\frac{2^{12}}{2^2} = 2^{10}$



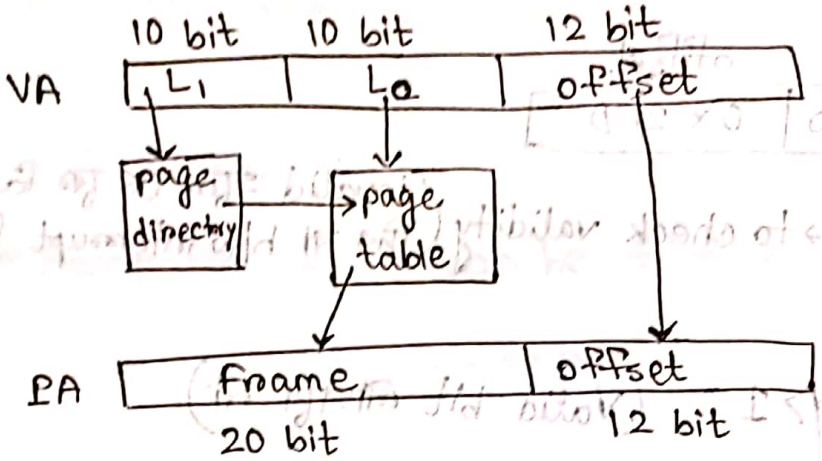
Virtual address

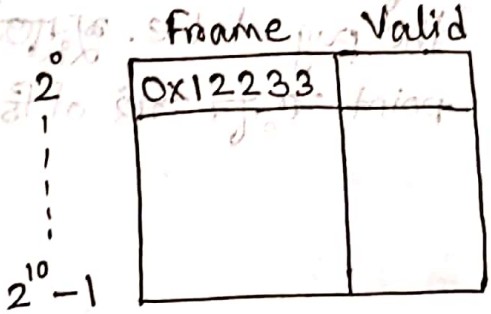
Physical address

no. of entry 2^{10}
no. of page tables 2^{10}
} total entries 2^{20}

26.11.22

Multilevel Paging:





* কারণ highest level pg. table (page directory) এর size pg এর চেয়ে ছোট হতে পারে। বাকি সব pg. table-এর size = pg. এর size

* pg. directory সরাসরি next level pg table-এ point করে না, ঐ pg. table-এর PA তে point করে।

Example:

	L_1	L_0	offset
VA:	0x3a5	0x200	0x2FD

(pd[0x3a5] & 1) → to check validity (invalid হলে, OS কে fault দিব বা h/w interrupt দিব)

then pd[0x3a5] >> 1 (Valid bit লাগবে না)

int*pt;

then pt=(pd[0x3a5] >>1) <<12 (higher 20 bit of PA of the page table. শুরুতেই point করবে তাই offset 0)

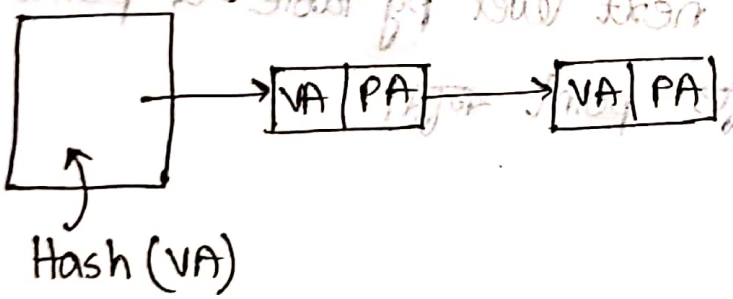
pt[200] & 1

frame=(pt[200] >>1) << 12

PA = frame | 0x2FD;

Inverted Page Table: ≡ Hash table

* Directly index map না করে hash value দিয়ে map করবে।

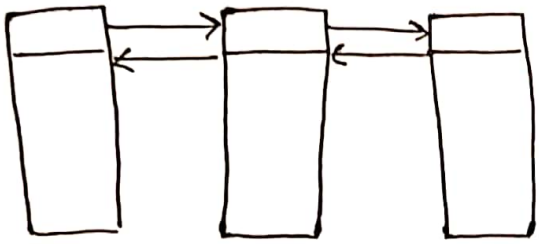


*H/W-এ implement করা হয় না কারণ variable num. of pointer operation করা লাগে, তবে software-এ use করা যায়।

Page Replacement:

* Swap Space → can be done by File system (Windows)

Replacement policy → FIFO



List of pages (Fix page structure)

```
Struct {
  size_t* next;
  size_t* prev;
  char body[4096-8];
} page;
```

body[4096-8]: ↓ pg size, → header

* size_t → virtual address এর size কত bit সেই type (int এর মত)

* Optimal page replacement ⟶ not realistic,

* Clock replacement algo ⟶ use bit 1→0 করে second chance দেয়।

* Library code গুলো OS নিজের virtual memory তে load করে রাখে এগুলো shared for all process.

Windows ⟶ .DLL (Dynamic linked libraries)