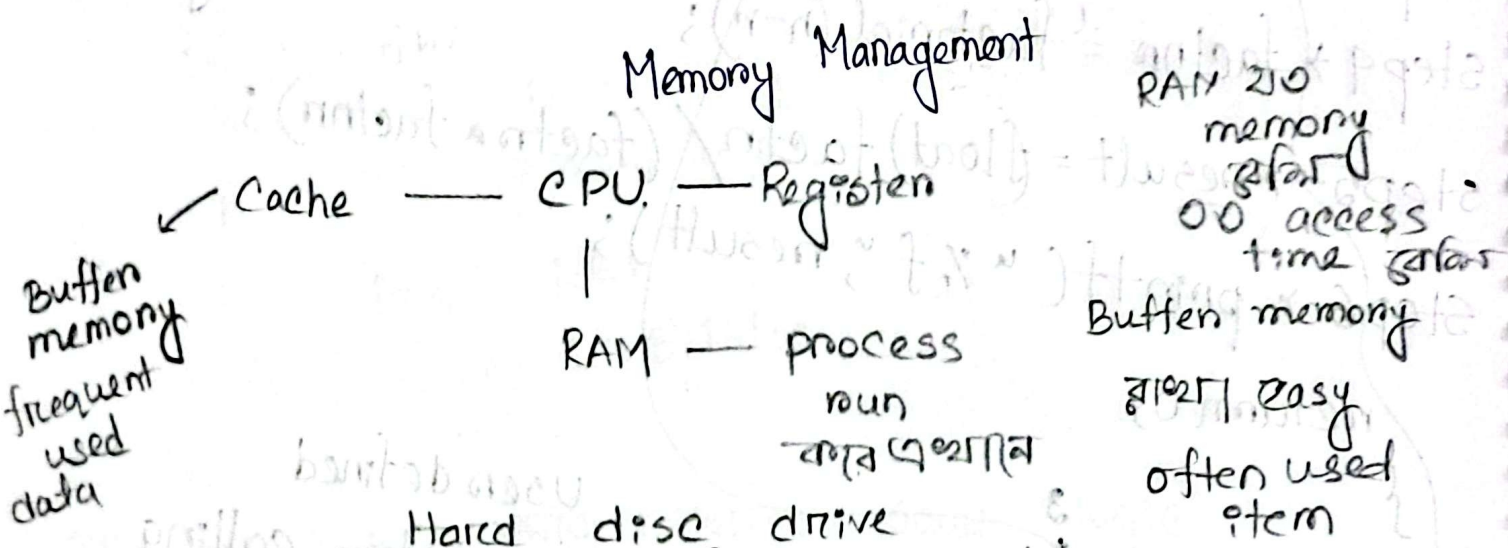
05 Syllabus

Memory Management – 15

Deadlocks

File system

Memory Management



Volatile memory

RAM → process memory

Cache → frequent used data of process

Register → The data that are currently in operation
CPU যে data নিয়ে operation করে

Volatile → electric power off করলে data remove হয়ে যায়।

HDD, SSD, Floppy, Disk, CD Rom → NON Volatile Memory (Used for permanent storage)

1(a) full

Physical Memory vs Virtual Memory

↓ ↓

program RAM এ load হয়ে run করে

Software program কোথায় memory allocate করবে সেটা বলে

Virtual memory or logical memory

↳ existed as software level / virtually existed

Physical memory or Main memory → RAM

প্রত্যেক process'র জন্য memory location 0 থেকে program'র memory allocate করে।

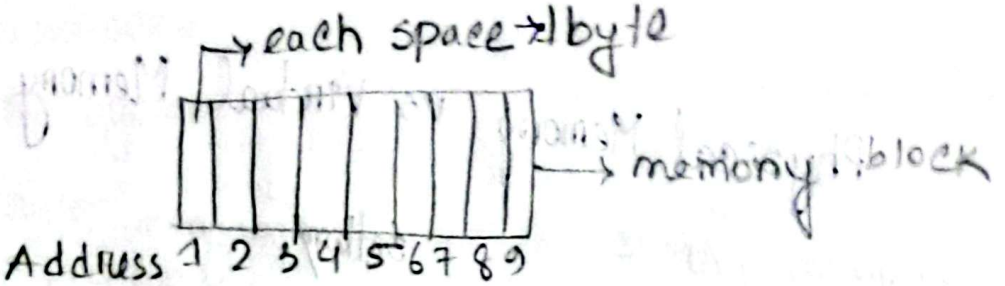
Code লিখলে → Virtual memory

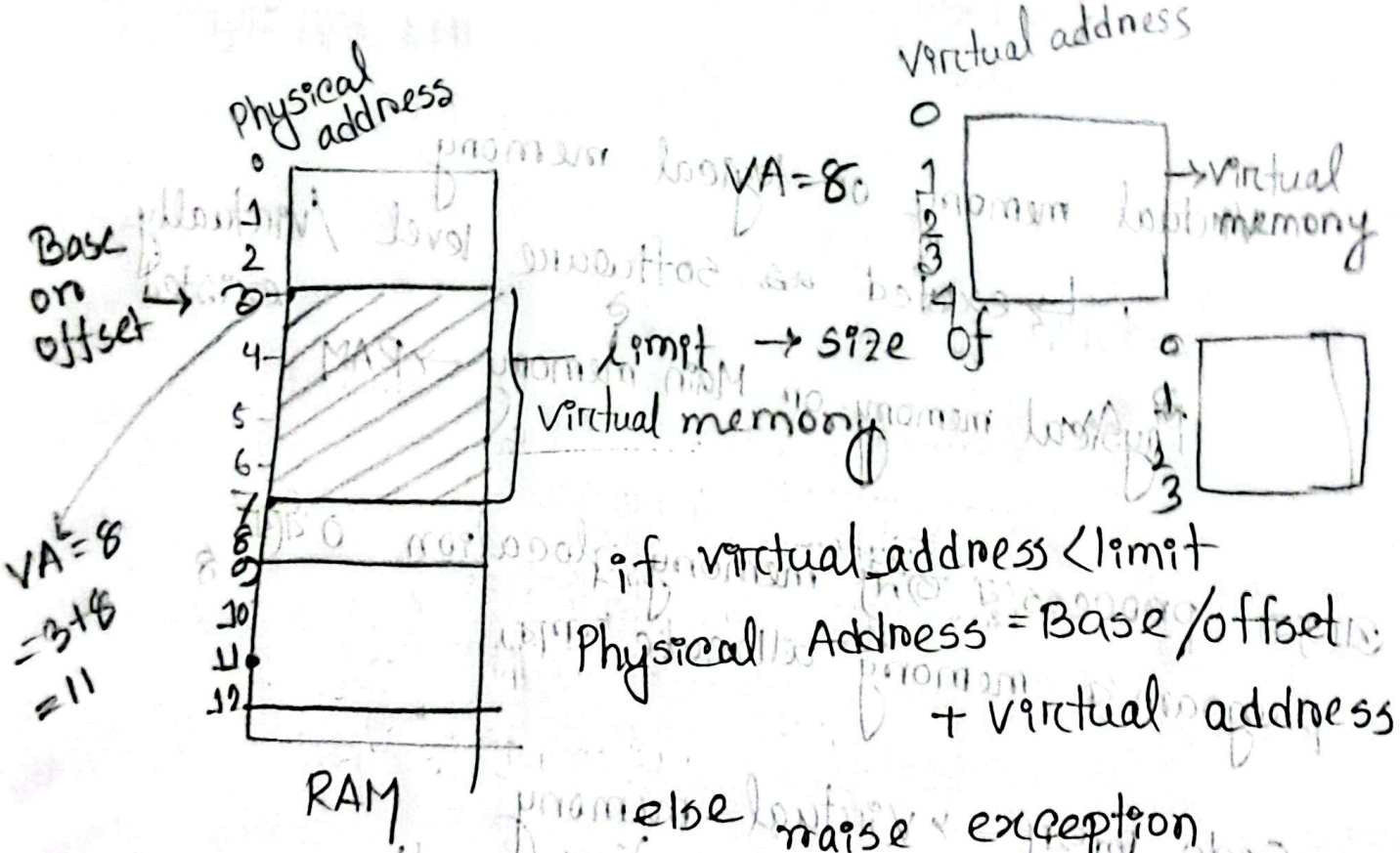
Code run → physical } translate

pointer value all virtual memory

Address Translation

Memory Address → unique location of each unit space (1 byte) of memory





if virtual_address < limit
Physical Address = Base/offset + virtual address
else raise exception

যদি virtual memory
তে বলা হয়েছে ৪ নাম্বার
RAM range'র বাইরে
access করেন তাহলে
আরেক process'B এর
memory তে হয়ে যাবে security breach

Unit Conversion

$1 KB = 1024 \text{ Bytes} = 2^{10}$ bytes

$1 MB = 1024 KB = 2^{20}$ bytes

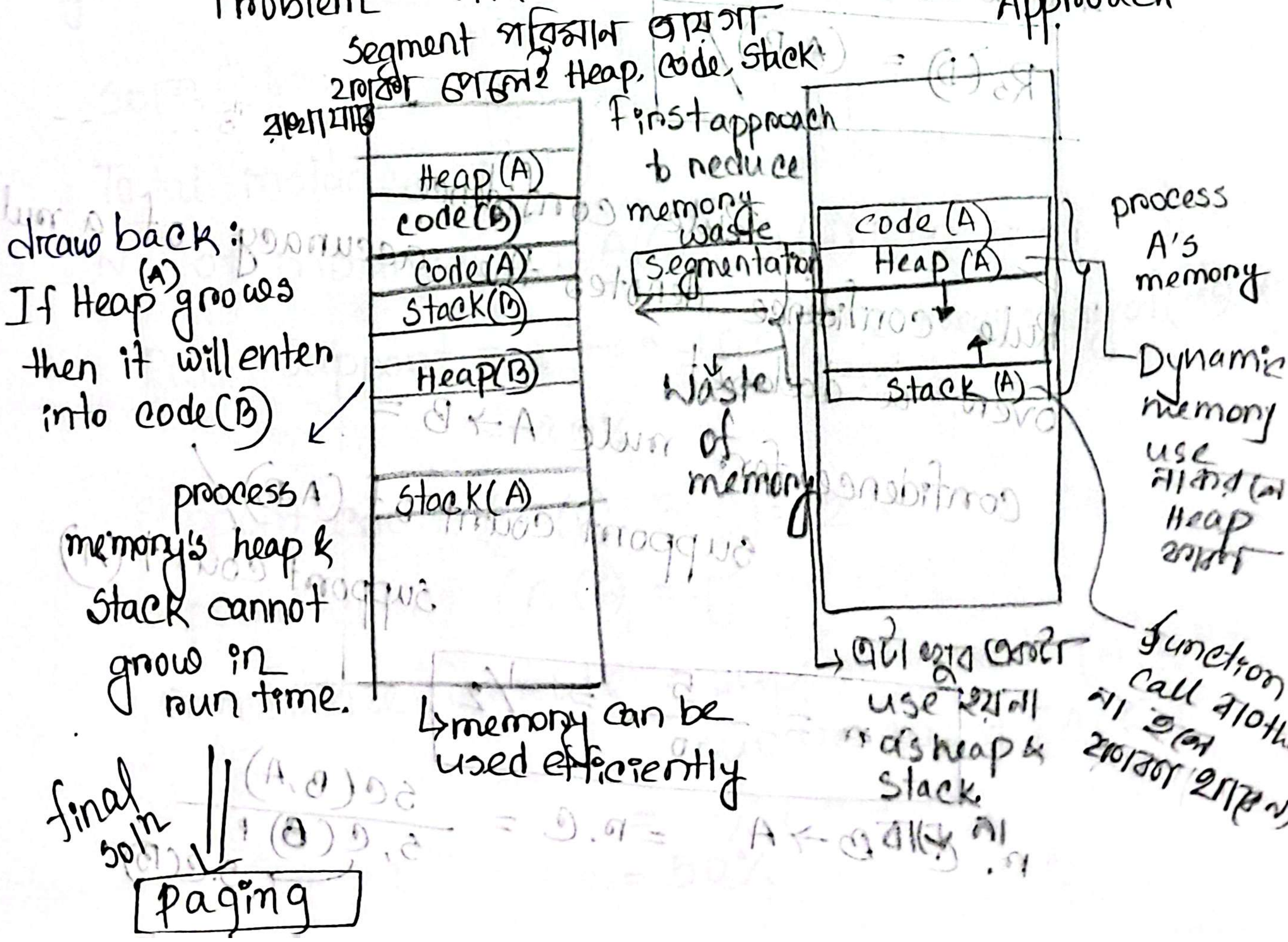
$1 GB = 2^{30}$ bytes

$1 TB = 2^{40}$ bytes

limit = 2048 B = 2KB

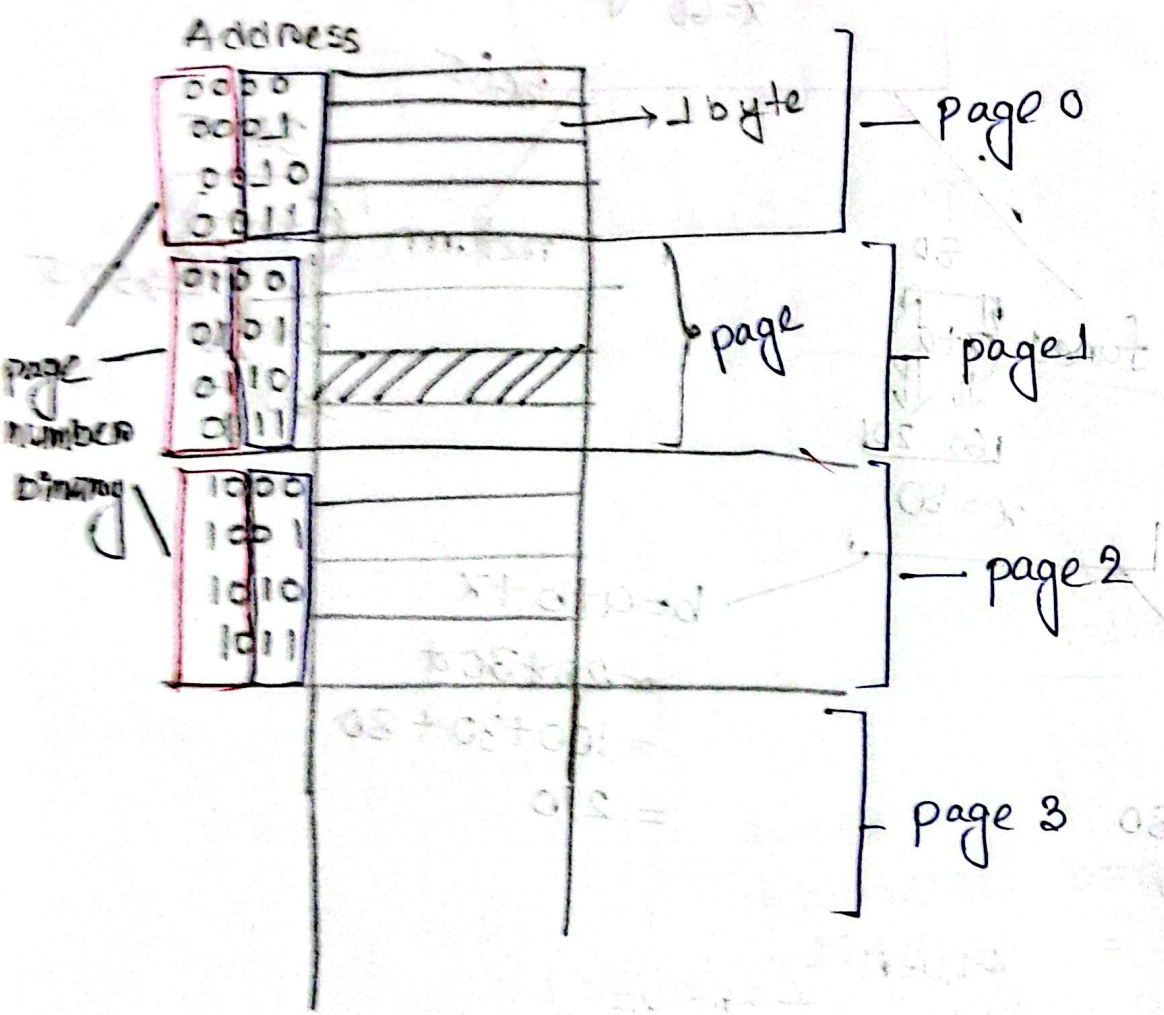
Vaild Virtual Address Range : 0 - 2047

Problem with current Memory Management Approach



OS

paging



Each page has same size

page number	offset

Address

offset

00
01
10
11

2 bit দিয়ে page এর কোন থেকে আছি জানা যায়।

offset = Address of specific position in a page

→ same byte size

both physical and vm
↳ same size of page

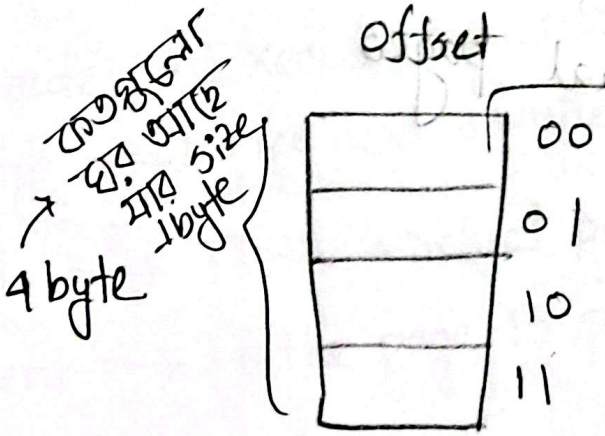
page number	offset
00	00
00	01
00	10
00	11

VPN → Virtual page number

Physical Frame Number → PFN

Page table → different virtual page
↕ mapped
different physical frame

page সম্পূর্ণ না লাগলে পুরোটাই page দিতে হবে
waste হলে OS কখনোই ভেঙে page দেই না।



total size = 1 × 4 byte = 4 byte

offset = $\log_2$ (page size in bytes)
↳ min. কতগুলো digit লাগবে

page size = 2^{offset} bytes

- Processes are given memory on demand and in a page size quantity [partial page is never allocated]

- page table is required to keep track of virtual page to physical frame mapping and page's permission, status etc.

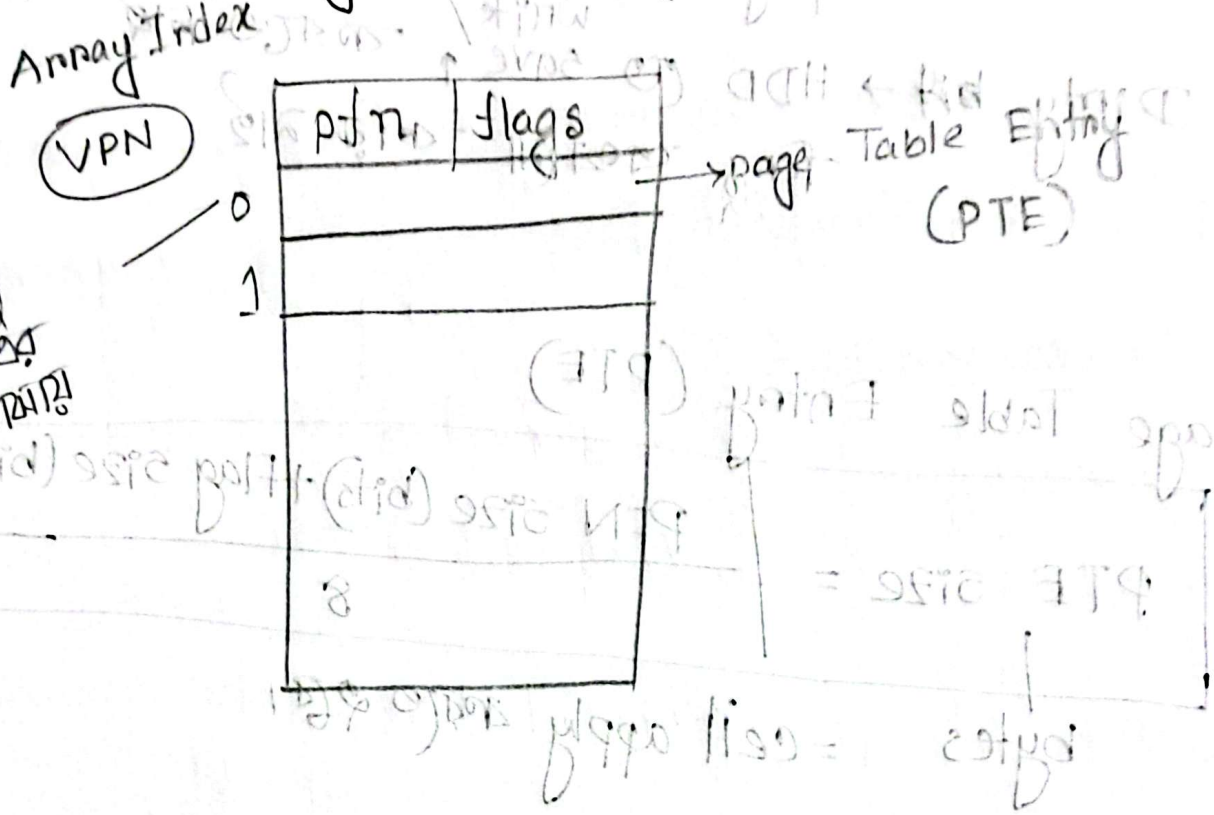
Page table

```
struct page_table_t {
        uint64 pfn;
        char flags;
};

page_table_t table[n];
//n = number of virtual pages
```

uint64
↳ unsigned integer 64bit
↳ 8byte

page table diagram



Flags

Read → Read permission → 1 bit

Write → Write permission → 1 bit

Execute → Execute permission → 1 bit

Dirty → Do we need to write the frame to HDD before using this frame for other page → 1 bit

Valid → If the virtual page has a valid pfn → 1 bit

Present → If the page is present in RAM or not → 1 bit

permission:
- code segment → Read, Execute
- Heap → Read, write

Present → process memory এর কতটুকু RAM আছে আর কতটুকু Hard disk এ আছে

Dirty bit → page এর content permentately write/save করতে চাই HDD তে
কিন্তু evict করি নাই

Page Table Entry (PTE)

$$\text{PTE size} = \frac{\text{PFN size (bits)} + \text{Flag size (bits)}}{8} \text{ Bytes}$$

bytes = ceil apply করতে হবে।

$\lceil 2.1 \rceil = 3$

$\lceil 4.5 \rceil = 5$

$$\text{Page Table size} = \text{number of virtual pages} * \text{PTE size}$$

int a[3] → 4×3 = 12 bytes

Clean confusion on bits & bytes

Address, offset, page number = bits

Memory size, page size, page table size = bytes

Address in bits → We can uniquely number 2^n byte
↳ Memory size = 2^n bytes

Memory size M bytes

Address size $= \log_2(M)$ bits

Address → number of ~~bits~~ memory numbering can be

Never divide address bits with 8 to convert into bytes

When size is given in bits, we can divide it with 8 to convert into bytes.

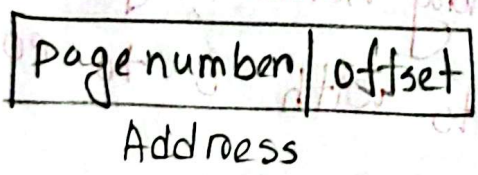
$$\text{PTE size} = \frac{\text{PFN}(=16\text{bits}) + \text{Flags}(=8\text{bits})}{8}$$

$$= \frac{16+8}{8} = 24/8 = 3 \text{ bytes}$$

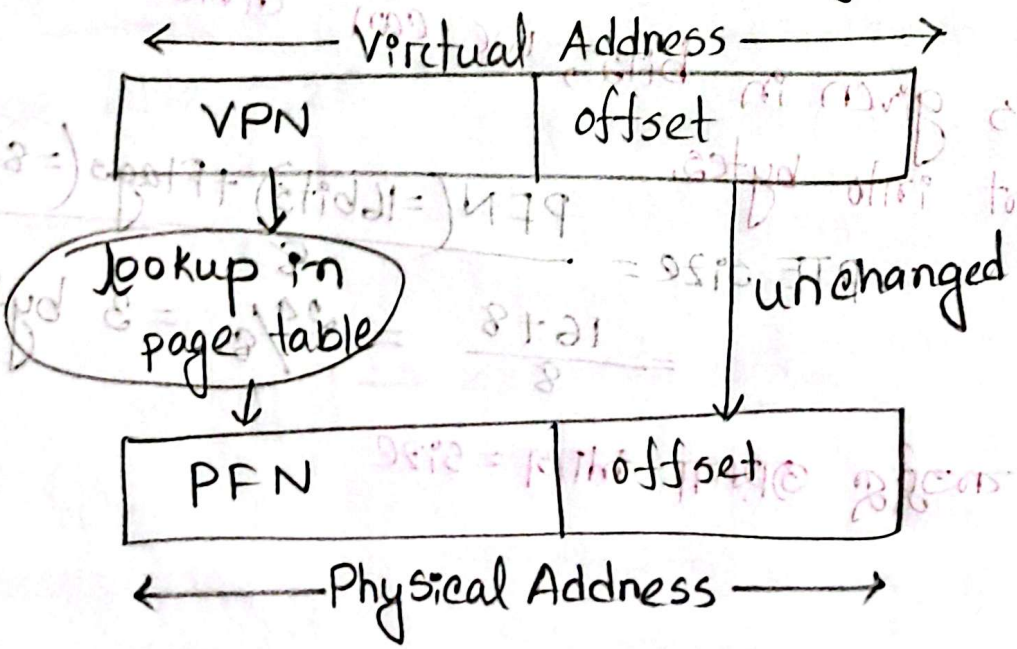
address কত Store করতে কতটুকু জায়গা লাগে = size

Paging Formula

1. offset = $\log_2$ (page size in bytes)
2. Address: m bits → memory size = 2^m bytes
3. Memory size M bytes → Address = $\log_2(M)$ bits
4. Num of bits for VPN + offset = Num. of bits for Virtual address
5. Num of bits for ~~VPN~~ PFN + offset = Num of bits for physical address



Address Translation in Paging:



- size of ~~of~~ PFN can be greater/ less than/ equal to VPN
- Offset is always same for virtual address and Physical address.

05

Practice Question on Memory Management

(1) Given, virtual Address size 48 bits

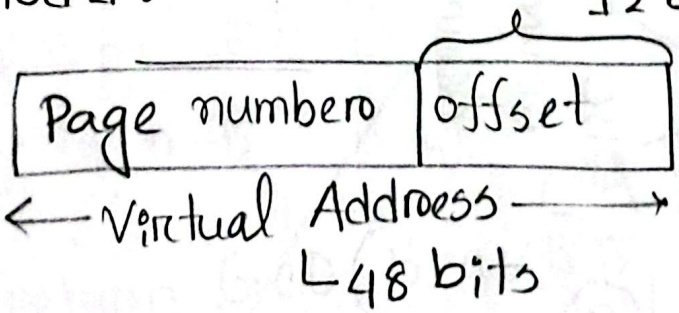
Physical memory size = 8 GiB $= 2^{30} \times 8$ bytes
$= 2^3 \times 2^{30}$ bytes
$= 2^{33}$ bytes

Page size $= 4$ KB $= 2^2 \times 2^{10}$ bytes
$= 2^{12}$ bytes

PTE size = 4 bytes

(a) offset = $\log_2$ (page size in bytes)
$= \log_2 (2^{12}) = 12$ bits

(b) Method 1:



(c) ∴ virtual page number VPN size
$= 48 - 12 = 36$ bits

Number of virtual pages $= 2^{\text{size of VPN}}$

$= 2^{36}$

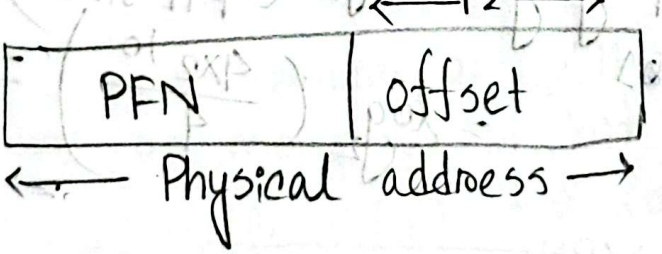
Method : 2

$$\text{Number of virtual pages} = \frac{\text{Virtual Memory size}}{\text{Page size}}$$

$$= \frac{248 \text{ bytes}}{4 \times 2^{10} \text{ bytes}}$$

$= 2^{36}$

(d)

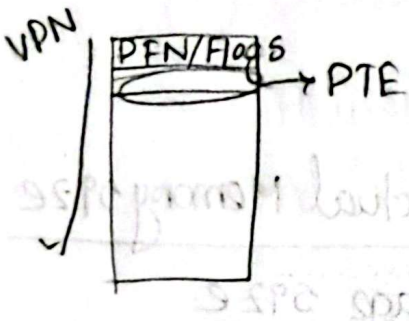


Physical Address size $= \log_2$ (Physical memory size)

$= \log_2 (2^{33}) = 33$ bits

$\therefore$ PFN $= 33 - 12 = 21$ bits

(e) size of single page table = Number of virtual pages * PTE size



$= 2^{36} * 4$ bytes

$= 2^{38}$ bytes

$= 2^{38} * 2^{30}$ bytes

$= 256 * 1$ GiB

$= 256$ GiB

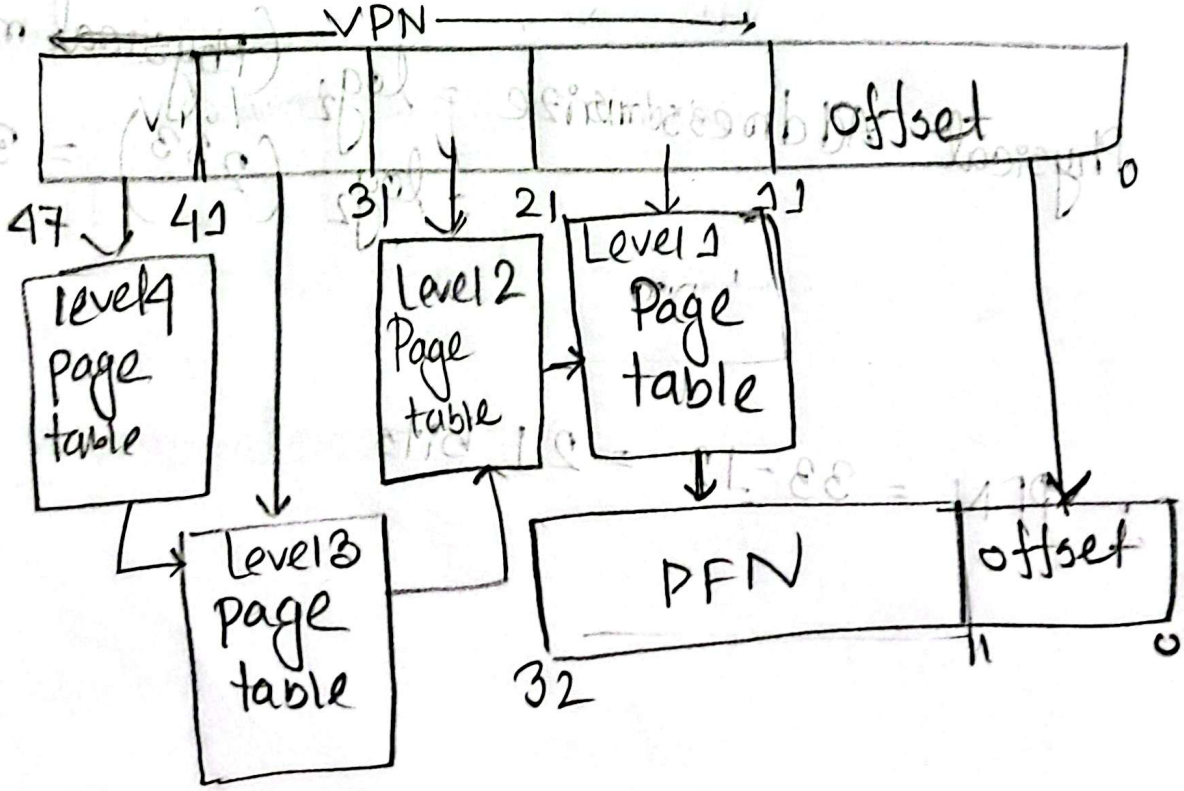
(f) Number of bits required at each level of Hierarchical paging $= \log_2 \left(\frac{\text{Page size}}{\text{PTE size}} \right)$

topmost level → level 4

inner level → level 3, 2, 1

$= \log_2 \left(\frac{4 \times 2^{10}}{4} \right) = \log(2^{10})$

$= 10$ bits.

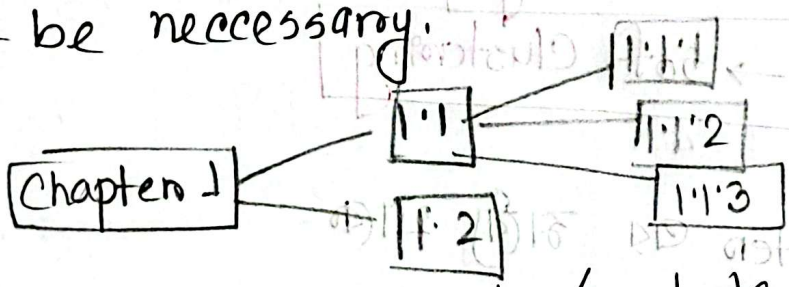


How to reduce the size of page table?

* Two things need to be considered

① We may not find large contignuous space in RAM to store the page table.

② Keeping track of all virtual pages may not be neccessary.



Hierarchical / Multiple label paging

page 1 or more valid bit → keep

page no valid bit → lose

Page Table is divided into pages

If a page of the table has at least one valid entry keep that page in RAM, otherwise discard it.

logical page another name for virtual page

(a) Virtual memory = 4 GiB
Page size = 4 KB

(b) ∴ Number of virtual pages = $\frac{4GiB}{4KB}$

$= \frac{4X2^{30}}{4X2^{10}}$

$= 2^{20}$ pages

(c) offset = $\log_2(4X2^{10})$ = 12 bits

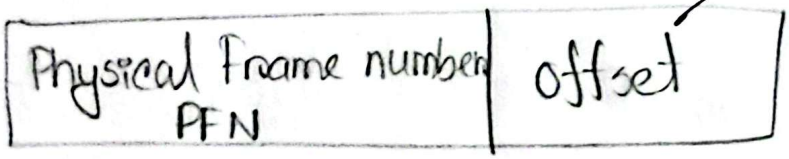


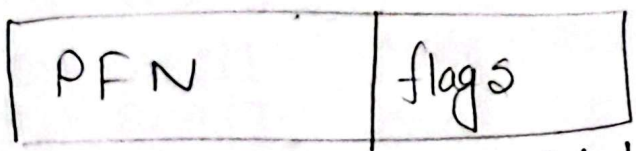
Table size = Number of VPs × PTE size

Physical address size = $\log_2(8GiB)$

$= \log_2(8X2^{30})$

$= 33$ bits

$4X2^{10}/4$



PFN = PAS - offset

PFN = 33 - 12 = 21 bits

∴ PTE size = PFN + flags

= 21 bits + 10 bits

= 31 bits

= ⌈31/8⌉ bytes

= 4 bytes

bits → byte ⌈/8⌉

আগের level এর page আরেক next level এর অবস্থান entry রাখবে।

Designing the multilevel page table → কতগুলো bit নিয়ে level এ রাখবে।

↳ to find out how many level ~~should~~

© Number of bits required for each level

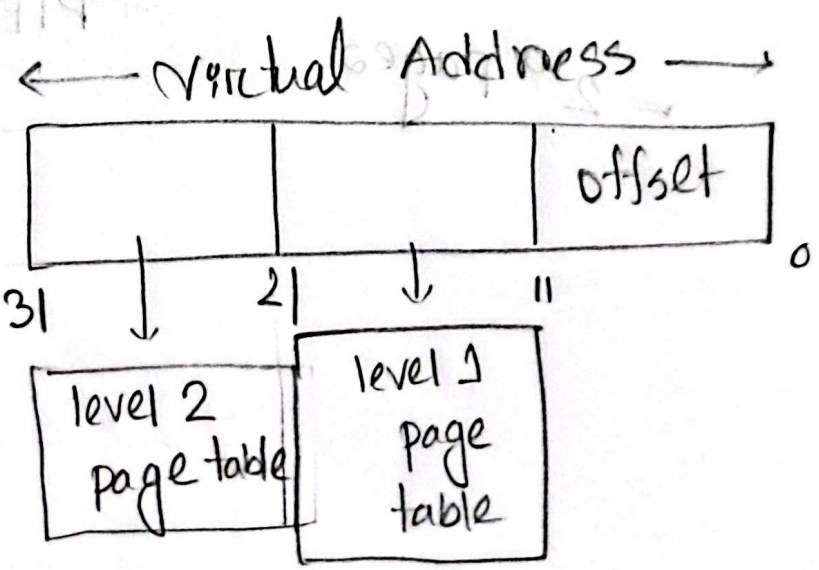
$$= \log_2\left(\frac{\text{page size}}{\text{PTE size}}\right)$$

$$= \log_2\left(\frac{4 \times 2^{10}}{4}\right) = 10 \text{ bits}$$

Table → virtual address নিয়ে কাজ করে।

set on RAM

virtual address = 4 GiB = 32 bi



level 1 এর page
level 2 এর entry

maximum memory → level 1 এর জন্য page allocate করতে হবে।

Assuming 2 level hierarchy

OS → to store the process tables of all processes → maximum memory required to store the page tables for one process → maximum page সাপোর্ট করে

Number of pages required to store level 1 page tables

for level 1 (Number of virtual pages)

$$= \frac{\text{Number of PTE}}{\text{Number of PTEs per page}}$$

$$= \frac{2^{20}}{2^{10}}$$

numbers of bits required for each level

$$\frac{\text{page size}}{\text{PTE size}}$$

$$= 2^{10} \text{ pages}$$

For level 2,

$$= \frac{\text{Number of PTEs}}{\text{Number of PTEs per page}}$$

→ আগের level এ যতগুলো page রাখা হয়েছে

→ $\frac{\text{page size}}{\text{PTE size}}$

$$= \frac{2^{10}}{2^{10}} = 1$$

∴ Total space required to store the page tables for all processes $= (2^{10} + 1) \times 1024$ pages

(e)

$(2^{10} + 1)$ → space required for one process

1024 → Maximum number of process

minimum memory to store process → প্রতি level থেকে 1 টা করে page allocate হয়

required to store the page tables for one process = Number of levels

= 2 pages (Ans:)

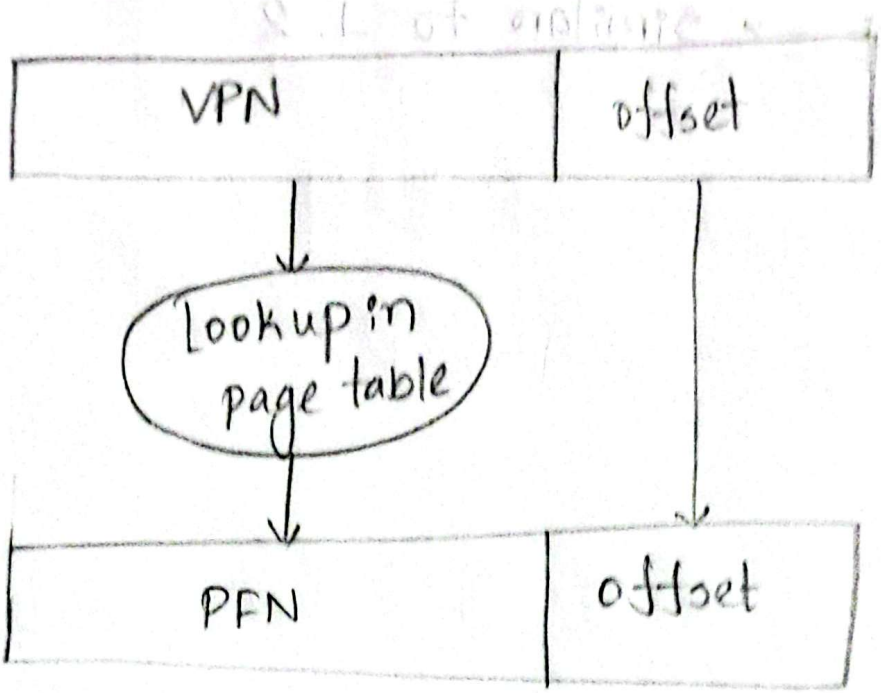
(১)

∴ For all processes = (2 × 1024) pages

maximum → all valid bit → 1

minimum → only one VPN → valid bit → 1

⑤ Address Translation in Paging:



$$\text{offset} = \log_2(4 \times 2^{10}) = 12 \text{ bits}$$

VPN = Virtual Address size − offset

= 24 − 12 = 12 bits

0x → Hexadecimal

VPN (first 12 bit)

binary to hexa

0000 0000 0011

= 0x 003

⑥ 0000 0000 0101

= 0x0005

⑦ 0000 0000 0111

= 0x007

PFN

0x032A0 → valid bit → 1

valid bit → 0 হলে

ex: 0x000AB → valid bit → 0

0x000AB → valid bit → 1

05

The Big picture of memory Access

TLB = Transition Lookahead Buffer

Cache → যে page গুলো often use হয় তা রাখা হয়।

Case1: TLB তে virtual address এর physical Address পাওয়া গেলে

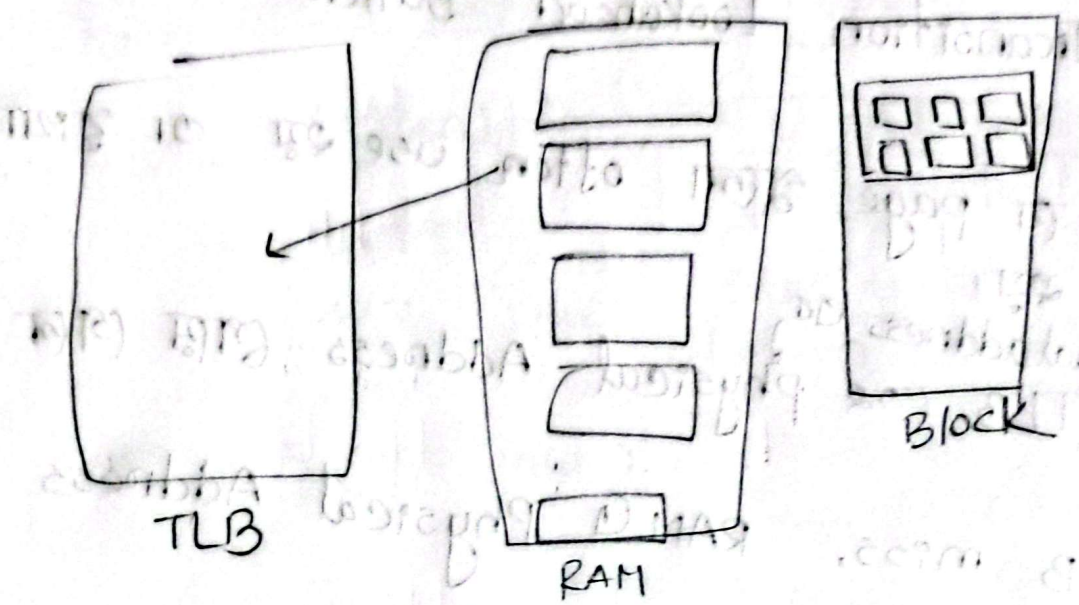
Case2: TLB miss, RAM এ Physical Address আছে.

Case 3: page fork
- Invalid virtual address
- Disk এ আছে RAM upload নাই

RAM is not free to get the address
↳ RAM swap some pages to disk which is not needed in near time.

lazy loading
↳ যতটুকু program দরকার সেটুকু RAM এ upload করা

multiple page
↳ block



optimal (Practically cannot be implemented because it replaces the page that are never used but we never know which page is never used

Page Replacement policy

FIFO → যে page টা RAM এ আগে আসছে সেটা RAM থেকে আগে সরাবো।

Ref: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 → Virtual address

find out PFN

Page ref	Hit/fault	Evicted (কোন page Replace হচ্ছে)	Memory stack (Max^m 3page) (initially blank)
1	Fault	X → No replacer as RAM এ জায়গা আছে	1
2	fault	X	1, 2
3	fault	X	1, 2, 3
4	fault	1 → Max 3 page so RAM full FIFO এ 1 out	4, 2, 3 → 1 এর PFN এ 4 বসছে
1	fault	2	4, 1, 3
2	fault	3	4, 1, 2

Page ref	fault/Hit	Evicted	Memory state (Max 3 pa)
5	fault	4	5, 1, 2
1	Hit	X	5, 1, 2
2	Hit	X	5, 1, 2

page hit → No change in memory state

3	fault	1	5, 3, 2
4	fault	2	5, 3, 4
5	Hit	X	5, 3, 4

#page Hit = 3

page fault = 9

Algorithm accuracy
↳ যত বেশি page hit তত ভাল

Target page hit number

Algorithm optimize

Two ways

RAM ও বেশি Table ঢুকানো entry করা যাবে না anamoly হয়, number of Hit কম
↳ Belady's

Belady's Anamoly → only for FIFO

Hit decreases with increase # pages

Page ref:

page ref	Hit/fault	Evicted	Memory state (Max^m 4 page)
1	fault	x	1
2	u	x	1, 2
3	u	x	1, 2, 3
4	u	x	1, 2, 3, 4
1	Hit	x	1, 2, 3, 4
2	Hit	x	1, 2, 3, 4
5	fault	1	5, 2, 3, 4
1	fault	2	5, 1, 3, 4
2	fault	3	5, 1, 2, 4
3	fault	4	5, 1, 2, 3
4	u	5	4, 1, 2, 3
5	u	1	4, 5, 2, 3

To optimized Belady's → Second chance Algorithm

Second Chance Algorithm

keep a reference bit. If we get a page hit, then ref. bit for that page is set to 1. otherwise ref bit is 0.

When a page needs to be replaced, we follow FIFO with a modification: If ref bit 1, don't replace the page, Reset the bit to 0 and move on to next page. If ref bit 0 replace that page

Page ref	Hit/fault	Evicted	Memory state (Max 4 page)
1	fault	X	1/0 → reference bit, → memory of page
2	fault	X	1/0, 2/0
3	fault	X	1/0, 2/0, 3/0
4	fault	X	1/0, 2/0, 3/0, 4/0
1	Hit	X	1/1, 2/0, 3/0, 4/0
2	hit	X	1/1, 2/1, 3/0, 4/0
5	fault	3	1/0, 2/0, 5/0, 4/0
1	hit	X	1/1, 2/0, 5/0, 4/0
2	hit	X	1/1, 2/1, 5/0, 4/0
3	fault	5	1/0, 2/0, 3/0, 4/0
4	hit	X	1/0, 2/0, 3/0, 4/[illegible]
5	fault	1	5/0, 2/0, 3/0, 4/1

ref bit - 1, hit → "1" রেখে দিব

যদি প্রথমবার ref → 1 পেলে আবার iterate হবে

ref bit 1 থাকলে change করতে পারবেন

1/1 2/1 3/1 4/1

5/0 2/1 3/1 4/1

Hit = 5

fault = 7

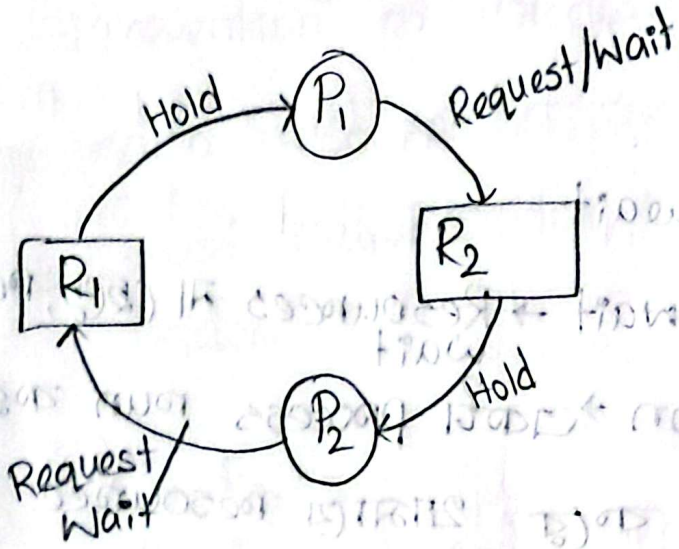
যত ref বেশি তত fault

Least Recently Used

যত আগে use হইছে সেটা তত আগে replace হবে।

one page'র জন্য recently usage এর বিরে ~~decision~~ decision নিবা।

page fault থেকে উপরের row তে যাব ~~max. page~~ memory state এর যে page আগে পাব তা রাখার আর যেটা last এ পাব সেটা বাদ দিবা।



4 characteristics → These 4 characteristics must occur simultaneously for Deadlock

1. Mutual Exclusion
2. Hold and Wait
3. No preemption
4. Circular wait

Resource Allocation Graph

□ → Resource

○ → Process

⋮ → instance of a resource

(P) → [R] : P requests for R

[R] → (P) : P is using R

একাধিক process'র মধ্যে Resource গত বাধ্যবাধিকতা

circular wait

Hold and wait → Resources না ছেড়ে, resource জন্য wait

preemption → একটা process run করতেসে তাকে জোর করে থামায়ে resource নিয়ে অন্য process run করানো

Live Lock → process running but infinite loop এ পড়ে no output

Dead lock → process stuck no instruction is not runned

 → instance

Resource – printer

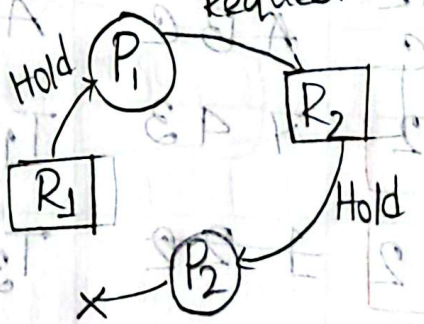
CPU তে 3টা printer attached প্রতিটা printer একেকটা instance.

Deadlock prevention :

Prevention of circular wait

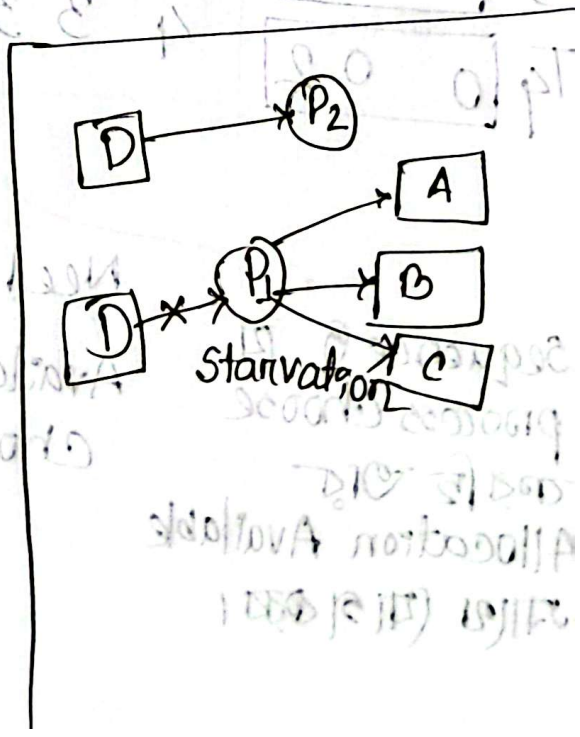
* Give each resource a serial number.
* Process must request for resources in a specific order

Resources : $R_1, R_2, R_3 \ldots$



P_2 cannot request in disorder.

একসাথে সব characteristic prevent করা লাগে না যেকোন একটা করলেই হয়।



Deadlock Avoidance: Banker's Algorithm

Deadlock Avoidance:

Deadlock prevention এ কোন measure নিতে পারছি না, এমন situation এড়ানো approach কাজে করলাম যেন deadlock না হয়।

Allocation → Max → Max-Allocation = Need → Safe sequence

Allocation A B C	Max A B C	Available A B C	Need A B C	Safe sequence A B C
T_0 0 1 0	7 5 3	3 3 2	7 4 3	T_1
T_1 2 0 0	3 2 2	5 3 2	1 2 2	T_3
T_2 3 0 2	9 0 2	7 4 3	6 0 0	T_5
T_3 2 1 1	2 2 2	7 4 5	0 1 1	T_0
T_4 0 0 2	4 3 3	7 5 5	4 3 1	T_2
		10 5 7		

Sequence এ যে process choose করছি তার Allocation Available সাথে যোগ করব।

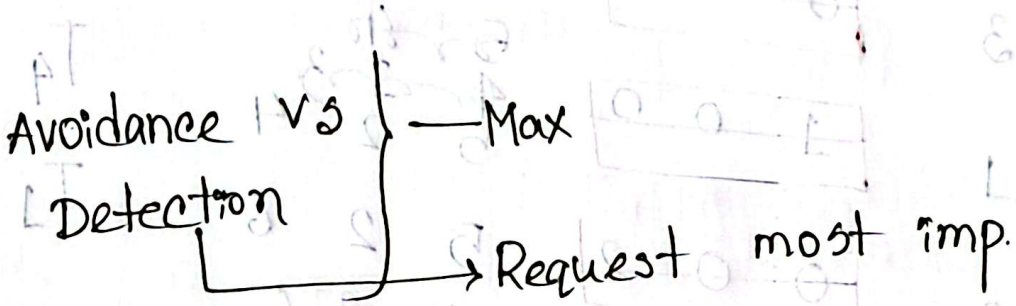
Need & Available মিলাবো

Available থেকে ছোট resource choose করব।

যদি কোনটা match না করে তাহলে Deadlock prevention হবে না

As all processes are in safe sequence, deadlock can be avoided

If not all process don't come in safe sequence then safe sequence process's can prevent deadlock others cann't



Deadlock Detection:

9/10/25

	Allocation			Request			Available			Safe Sequence
	A	B	C	A	B	C	A	B	C	
							0	0	0	T_0
T_0	0	1	0	~~0~~	~~0~~	~~0~~				
							0	1	0	T_2
T_1	2	0	0	~~2~~	~~0~~	~~2~~				
							3	1	3	T_3
T_2	3	0	3	~~0~~	~~0~~	~~0~~				
							5	1	3	T_4
T_3	2	1	1	~~1~~	~~0~~	~~0~~				
							5	2	4	T_1
T_4	0	0	2	~~0~~	~~0~~	~~2~~				
							7	2	6	

Safe Sequence ও আনা process এর available + allocation বসাব

A, B & C available & request ~~করা~~ compare করে safe sequence select করব।

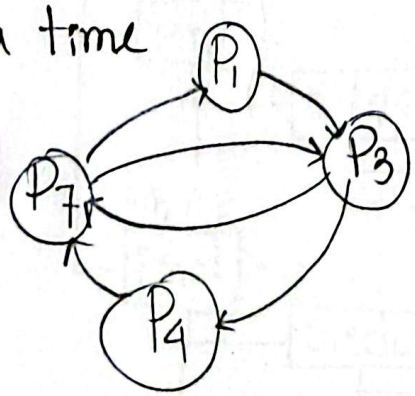
	Allocation			Request			Available			Safe Sequence
	A	B	C	A	B	C	A	B	C	
T_0	0	1	0	0	0	0	0	0	0	T_0
T_1	2	0	0	2	0	2	0	1	0	
T_2	3	0	3	0	0	1				
T_3	2	1	1	1	0	0				
T_4	0	0	2	0	0	2				

Deadlocked

তখন request থেকে কোনটাই select করতে পারব না যেকোনো একটা resource তাই available থেকে একটা বাড়াবো।

Deadlock Recovery — short Question

Process Termination
instead of Abort all deadlock threads, abort one threads/process at a time



Progress করা যায়

Resource Preemption
যতক্ষণ progress হচ্ছে তা save করে রাখা এবং delete করা।

4 characteristics & their prevention

Banker's Algorithm → Avoidance
Banker's Algorithm → Detection
} simulation

Deadlock Recovery → Process Termination
Deadlock Recovery → Resource Preemption

File System

~~File Attributes~~

* Folders/Directories, .txt, .pdf, .mp3, .mp4 etc → Needs persistance storing

* keywords, mouse, monitor etc → peripheral devices

Reposition within file → 1st page's content 2nd page ও নিয়ে যাওয়া

Disk Block ও data store হয়

SLd

student user

OS Lab

concurrency.c

faculty user

OS Lab

demo.c

Admin user

কোন folder টা কোন user এর কাছে আসে বা OS জানে না → naming problem

↳ which folder belong which user

OSLab | OS Lab

concurrency.c demo.c

Group system is lost

Final

slides

1.pptx 2.pptx

Mid

slides

1.pptx 2.pptx

Summer'25 OS

slides

1.pptx 2.pptx

Grouping problem

Group into is preserved

summer 25 OS

Mid

slides

1.pptx

2.pptx

Final

slides

1.pptx

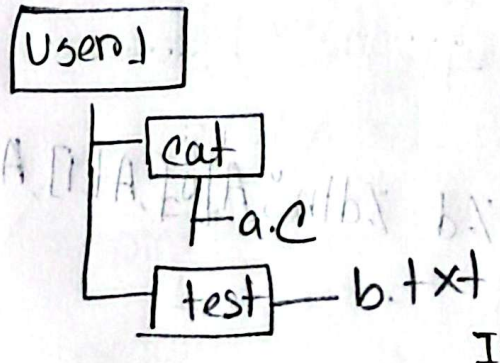
2.pptx

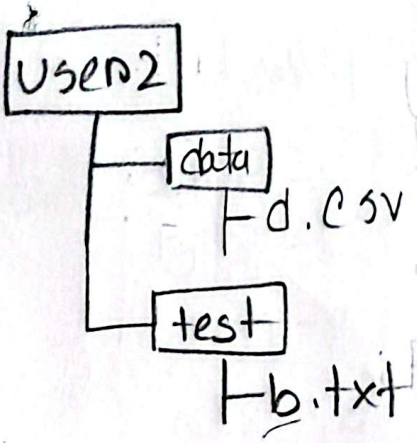
using single level directory

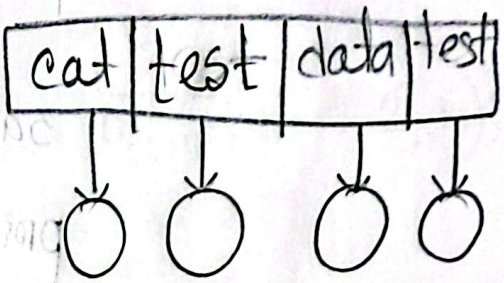
12/10

File Directory structure

Single level Directory: List of directories from all users



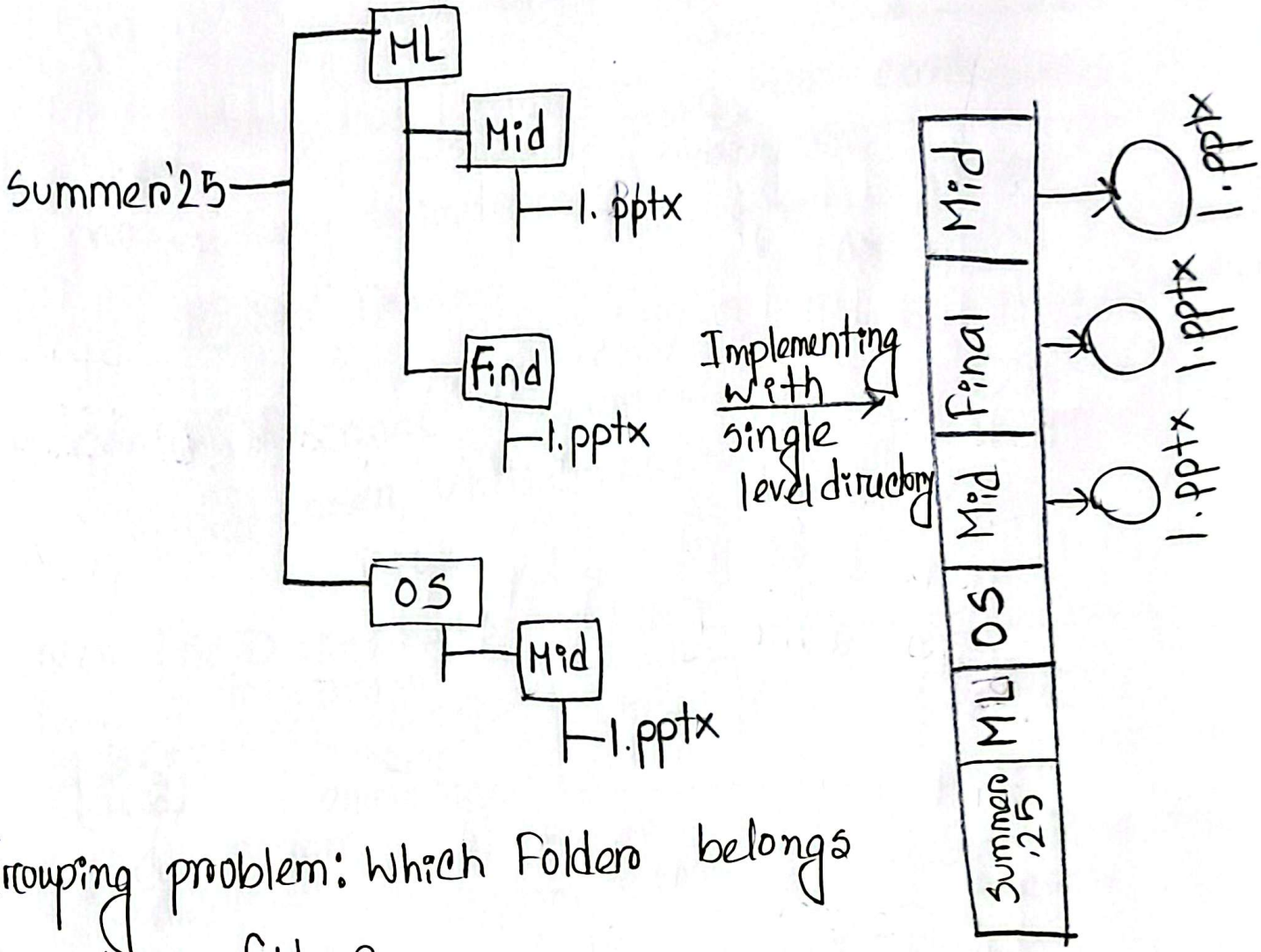




Implementing with single level directory:

Naming problem: which "test" belongs to which user?
↳ different user has to be unique.

*imp. data lost হবে।



Grouping problem: Which Folder belongs to which folder?

First Improvement → Two level directory

Naming problem will be solved

But grouping problem won't be solved.

→ Tree structured directories

both naming & grouping problem will be solved

File Allocation Methods

(ছড়ায়ে ছিটিয়ে Data রাখা যায়)

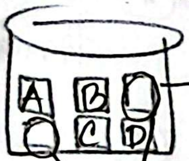
Contiguous
(গুছানো বা পাশাপাশি রাখা)

disk এ
1) Starting block
2) # of blocks
to store

pros: ① Easy to implement.
② Fast Random Access:
a[5] → constant time
a[0] → constant time

Cons: ① External Fragmentation
② Lower utilization of Disk

New File D size 2 blocks



→ block
কিন্তু রাখতে পারব না
↑ জায়গা থাকার পরও

কতগুলো page/ফাঁকা block
external fragmentation
block

Disk এর ভিতরের ক্ষুদ্রতম space → block

Random Access: O(1)
↳ constant time এ access করা যায়

Non-contiguous

pros:
① No External Fragmentation
② Better utilization of Disk

Linked List

Cons: Slower Random Access

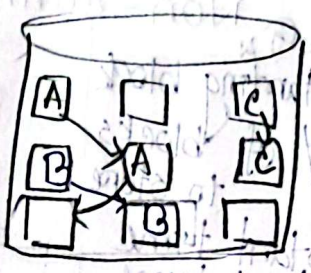
Indexing

pros:
Fast Random Access

Cons:
① Additional spaces for indexing blocks
② Multi level Indexing

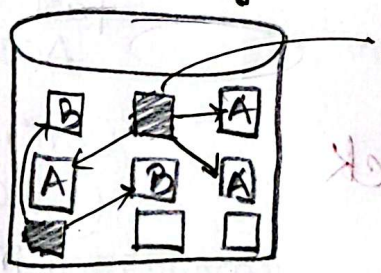
Array তে যেকোন value constant time এ access করা যায়

Linked List:



node - 5 যেতে হলে Head node থেকে start করে Traversen করে node 5 এ যেতে হয়।
তাতে time complexity বাড়ে।

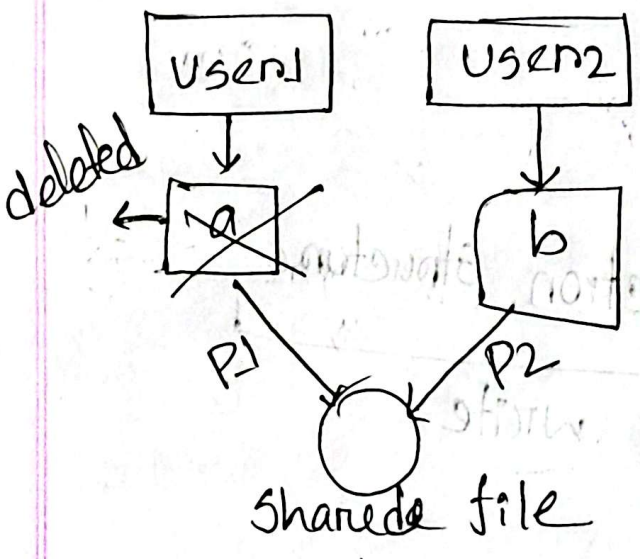
Indexing:



251

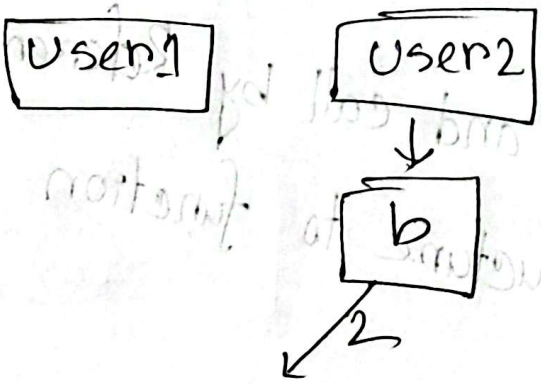
(a) Indexing

Dangling pointer



↓

Delete Folder A



P2 pointer is pointing to NULL

This is known as Dangling pointer problem.

→ Back pointer → limit সংরক্ষণ folder এর সাথে share করা যায় due to array size of BP

BP = [P1, P2, P3]

→ Daisy Chain: using linked list instead of array

Problem: To delete a pointer you need to traverse a lot. [BP can do it easily]

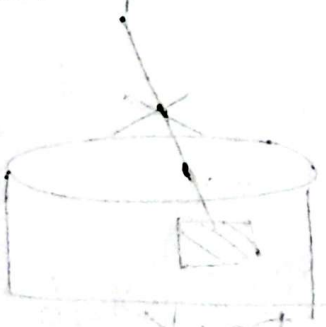
inefficient in detection $P_1 \rightarrow P_2 \rightarrow P_3$

→ EHC (Best Approach) Entry Hold Count

introducing count variable

when count = 0, delete the file

count = 3



Free space Management

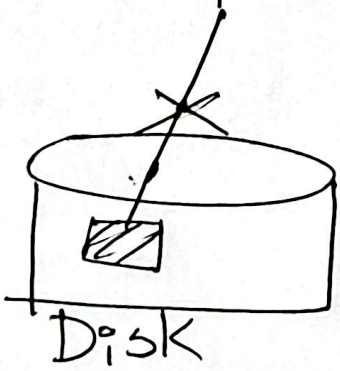
Motivation:
Computer ত কোন file delete করি

2 Approach

① Disk address of file
↳ link Delete (faster)

Disk ত file থাকে কিন্তু shows delete হয়ে গেছে

1.pdf for normal computer



② Over write with 0s (corrupt)
for more secure computer

Now find how to store files in disk

Two ways of ~~free~~ file management

Bit Map

0 → occupied

1 → unoccupied

কোন কোন block free আছে আর কোন block free নাই?

2byte = 16 bits

এ প্রত্যেক block এ bit map রাখা হয়।

per block → 1 byte

per bit → block info store করা।

keep the record of whole disk space, whether block is free or occupied using bit

linked list

free space এক অন্যকে point করে রাখে

Grouping (Traverse time improved)

counting (contiguous free block ↳ সুবিধাজনক)

↳ n-1 linked list

(non contiguous, contiguous)

guidelines

~~Code (2 ta) Deadlock~~ → A বাদ

Deadlock
- Deadlock Avoidance → simulation
- Prevention → must
- Recovery → short Question

File System → 25.1 → 6, 7 ; class Note

Deadlock
- Resource Allocation Graph → short Question
  - Single Instance resource allocation graph has cycle then deadlock exists
  - Multi Instance resource allocation graph has cycle → Deadlock may or may not exist

Memory Management

Paging & Page Table → Maths (90%, 10% complex)

Page replacement policy → simulation + short questions

Block Diagram of memory access → ① what happens when a page fault / TLB miss occur

FIFO → Belady's Anamoly

second Chance

LRU