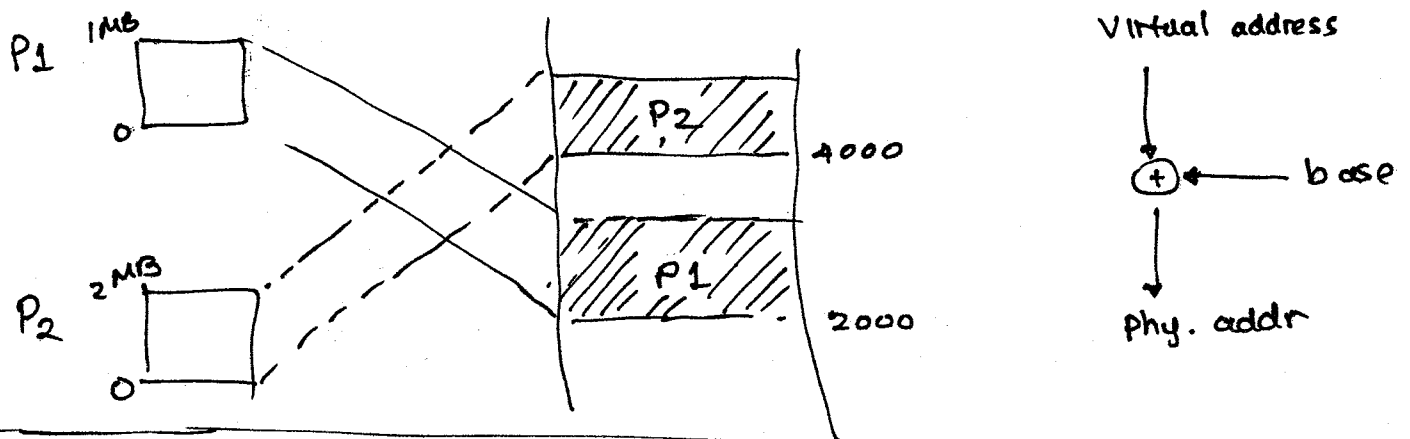


Dynamic Relocation (Base + Bounds)



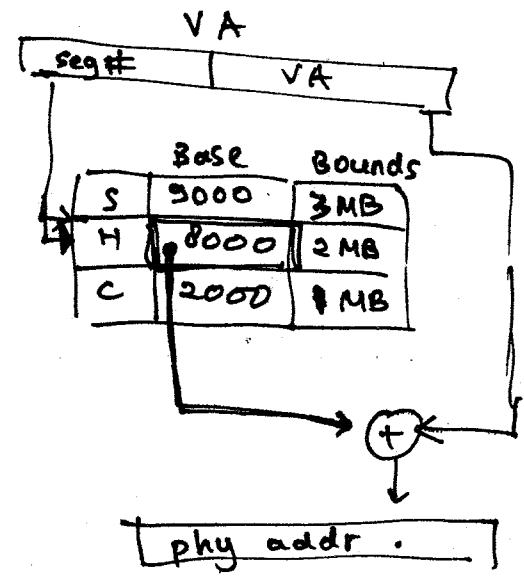
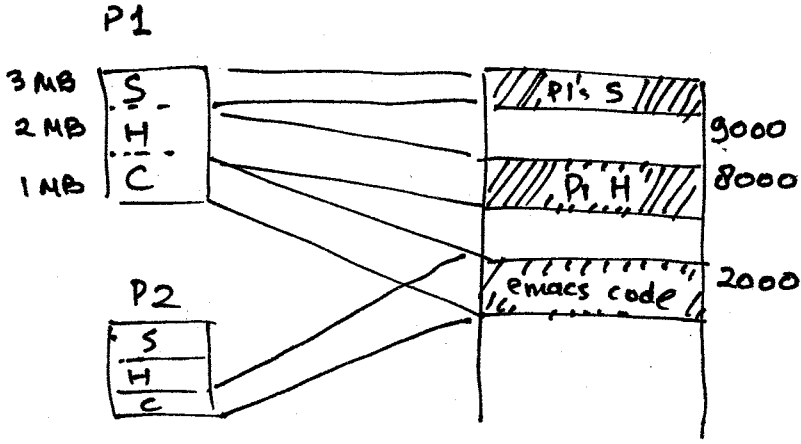
Case study:

- (A) P1 created? init B&B in MMU
- (B) Heap grows? Relocate all P1's memory
- (C) Context switch P1 → P2?

(+1) Easy relocation.

- (-1) If big process comes in? not fit in any hole, external fragmentation
- (-2) Sharing? e.g. code sharing → impossible
- (-3) Small process but given big memory? internal fragmentation
- (-4) A small process grows to a big process? relocate the whole process

SEGMENTATION

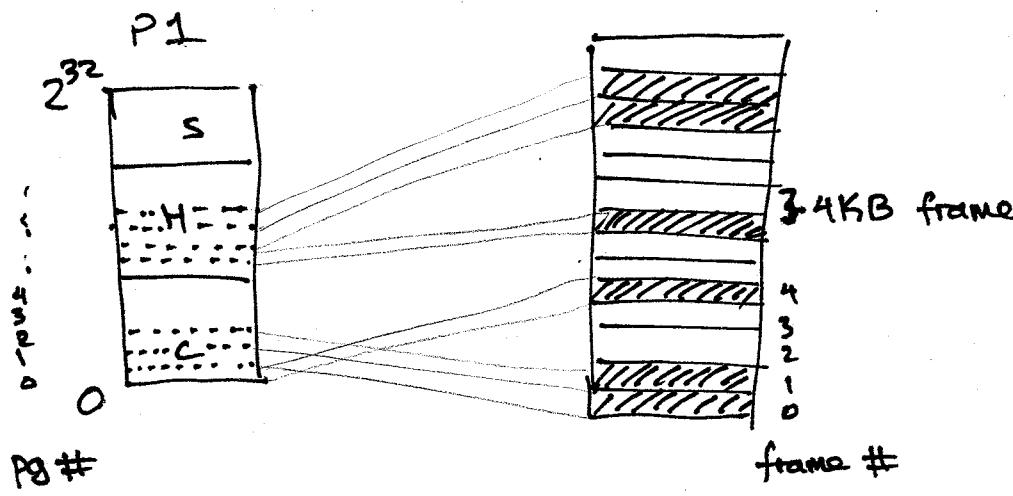


- P1's
- (A) P1 created? B&B for S, H, C
 - (B) P1 Heap grows > 2MB? relocate all P1's heap
 - (C) context switch? MMU uses P2's BBs for S, H, C

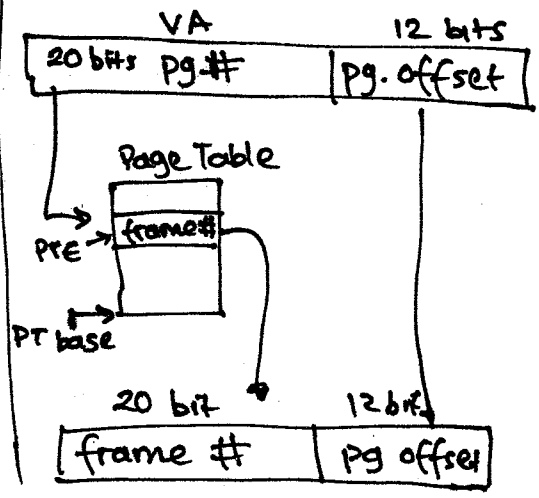
- (+1) Segment independently grows, and relocated.
- (+2) Sharing! yay!

(-1) Big segment is needed ?? no hole fits
 segment \rightarrow contiguous address.

PAGING



Addressing



- Q:
1. page size?
 2. # entries?
 3. pg tbl size?
 4. pg tbl in MMU?
 5. How to find the PT of a process

Case Study

- A System boots
- A process is created? code = 8 KB
- A process runs?? (see examples for paging)
- malloc (1B)?? (Note! different from system to system)
- malloc many times?!
- How many memory accesses??
- Context switch?

- (+1) no ext. frag
- (+2) simple free space mgmt
- (+3) fine-grained sharing (e.g. data sharing)
- (+4) easy to swap out more (next week)

- (-1) Best page size?
too big?
too small?
- (-2) Huge page table
- (-3) inefficient 2x slowdown

32 bits VA, 4 KB Frames

0x 00004010 → ? 7010

0x 00003010 → ? D010

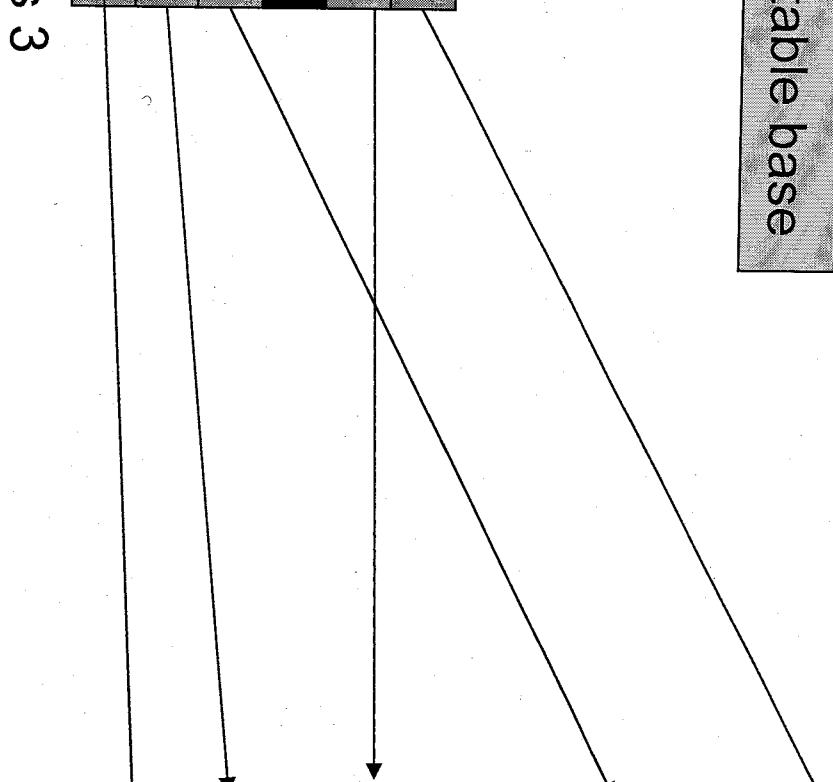
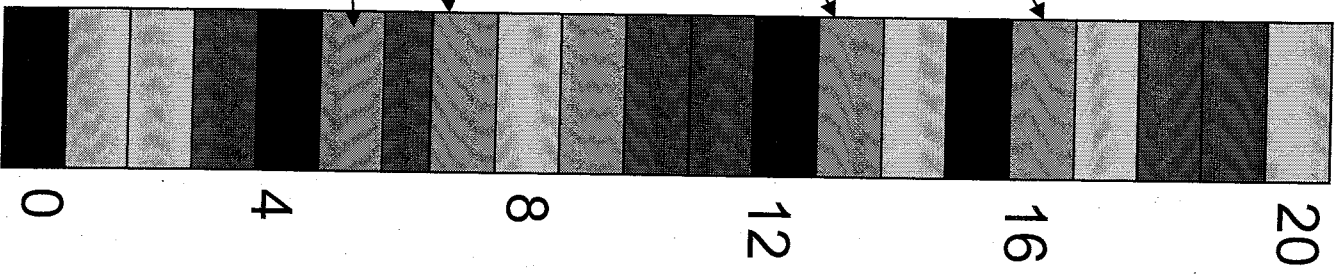
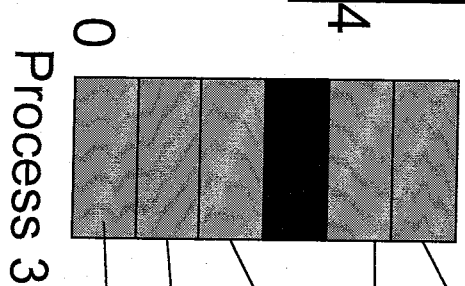
0x 00002010 → ? X

0x 00000010 → ? 10010

Pg. Example

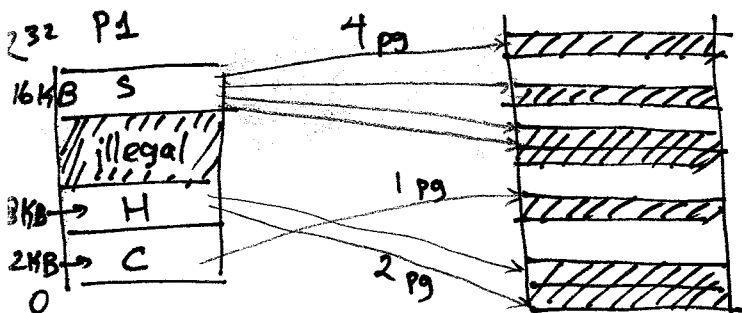
frame	R	W
5	1	1
7	1	1
13 (0xD)	1	1
2	0	0
9	1	1
0	1	1

page table base



SEGMENTATION + PAGING

Goal: solve (-2) of Paging.



Case Study

(A) System boots

(B) A process is created?

(code: 2KB) \rightarrow frame # \rightarrow PT \rightarrow PT base \rightarrow segment

Code:

(C) A process runs (go to examples)

(D) malloc (1B):

frame # \rightarrow update PT entry \rightarrow return VA

(E) malloc (> 8KB)

- current heap bounds 8KB
- PT must grow, but PT must be in contiguous memory

(F) How many memory accesses?

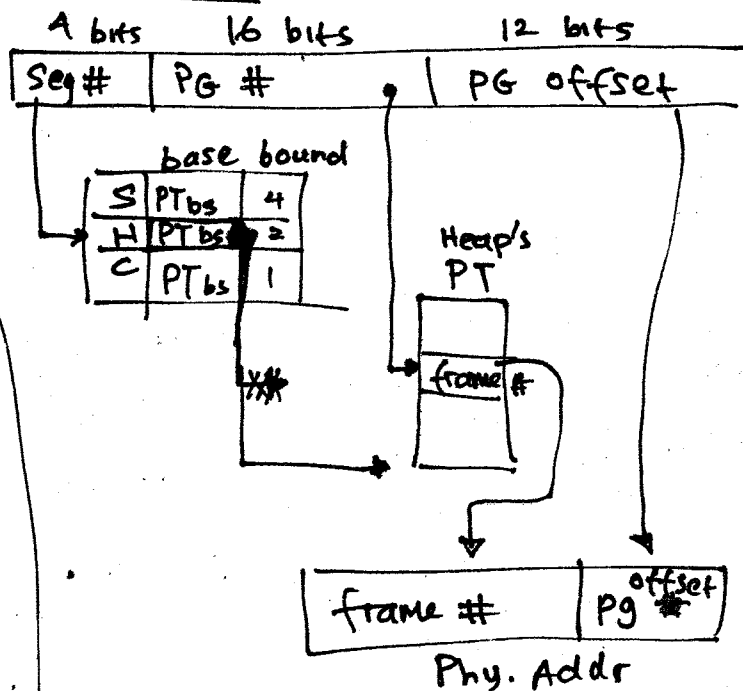
- (+1) PT size \downarrow
- (+2) sparse address space
- (+3) No ext. frag
- (+4) Segment grow \rightarrow no reshuffle

Recap segmentation:

base + bounds for each segment

$S_g + P_g \rightarrow$ A page table for each segment

ADDRESSING



Questions

- (1) Page size
- (2) # entries in PT? bounded
- (3) PT size $\approx 2^{16} \times 4 \text{ bytes} = 1 \text{ MB}$
- (4) Where is page table?
- (5) Where is segment table?

(+5) flexible sharing
code \rightarrow share ??
data \rightarrow share ??

(-1) Inefficient

(-2) Large PT must be contiguous.

Example of Paging and Segmentation

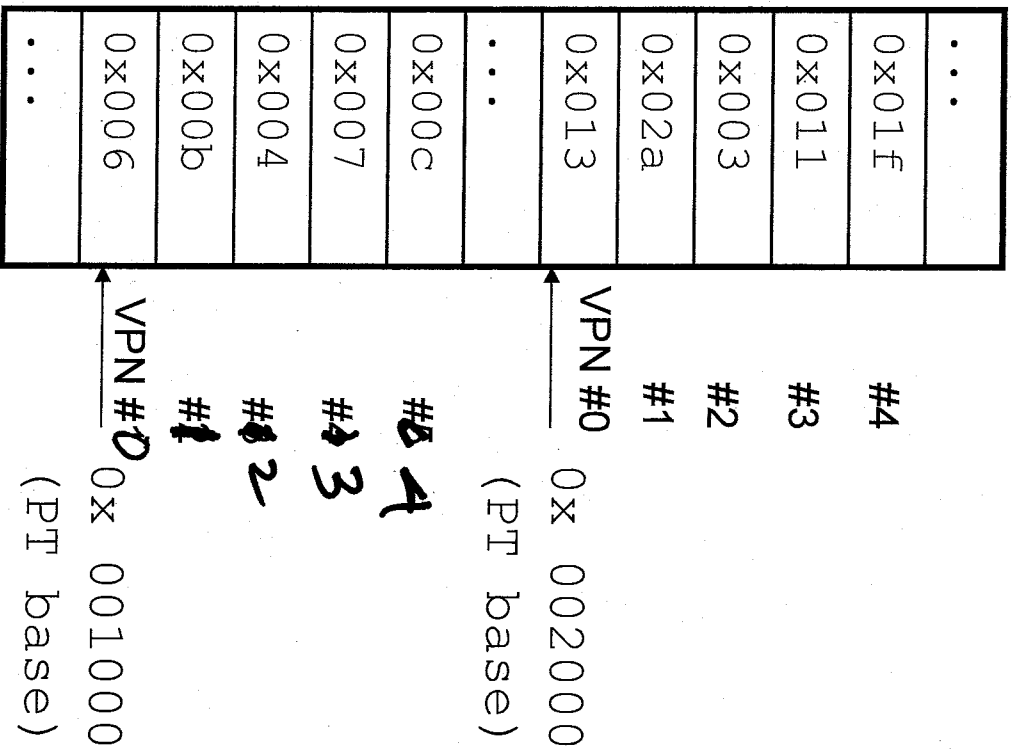
Translate 24-bit logical to physical addresses

- Page size: 4KB, 12 bits offset..
- 4 bits segment
- 8 bits page number

seg	PT Base	bounds	R	W
0	0x 002000	0x 14	1	0
1	0x 000000	0x 00	0	0
2	0x 001000	0x 0d	1	1

0x 002070 read: **3070**
 0x 202016 read: **4016**
 0x 104c84 read: **X**
 0x 010424 write: **X**
 0x 210014 write: ~~eaty~~ **X**
 0x 203568 read: **7568**

~~2K568 read:~~

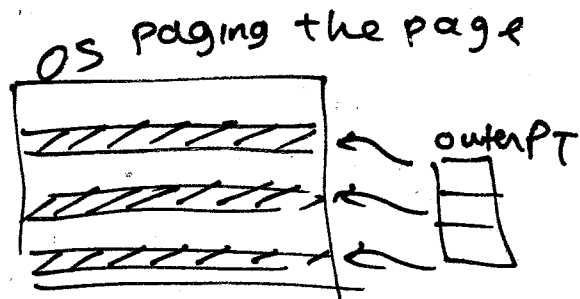
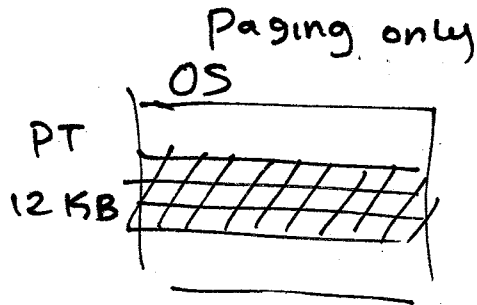


PAGING THE PAGE TABLES

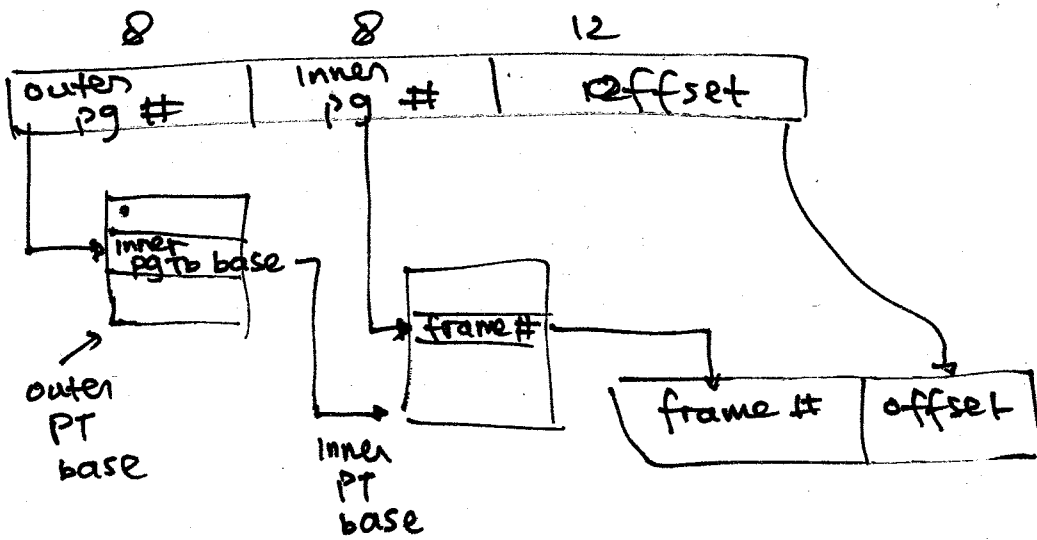
Paging → process' memory not contiguous

Hence, Page the PT → PT memory not contiguous

Mem View
the same



Addressing (simpler = 28 bits)



- Q1: page size
- Q2: entries?
- Q3: PT size - 1 KB

Case study

- (A) System boots
- (B) Process is created
code (1KB) how many inner table?
code (8MB) how many inner table?
how many frames for 8 inner table? 2 frames
(bad!, you want 1 frame to be 1 inner table)
- (D) How many accesses ??
- (E) Context switch ?

$$\frac{2^{23}}{2^{12}} = 2^{11} \text{ entries} \rightarrow \frac{2^{11}}{2^8} = 8$$