Lecture: 30

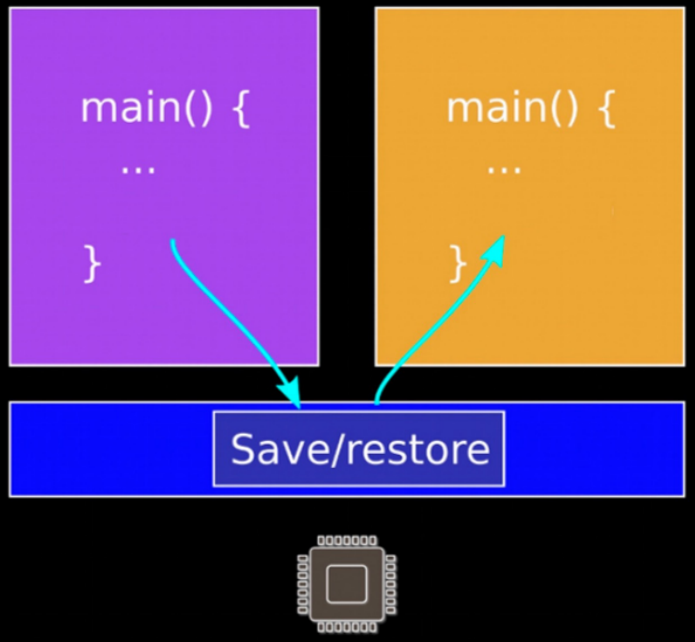
Summary So Far..

→ Base & Bound

→ Segmentation

→ Paging.

Two programs one memory



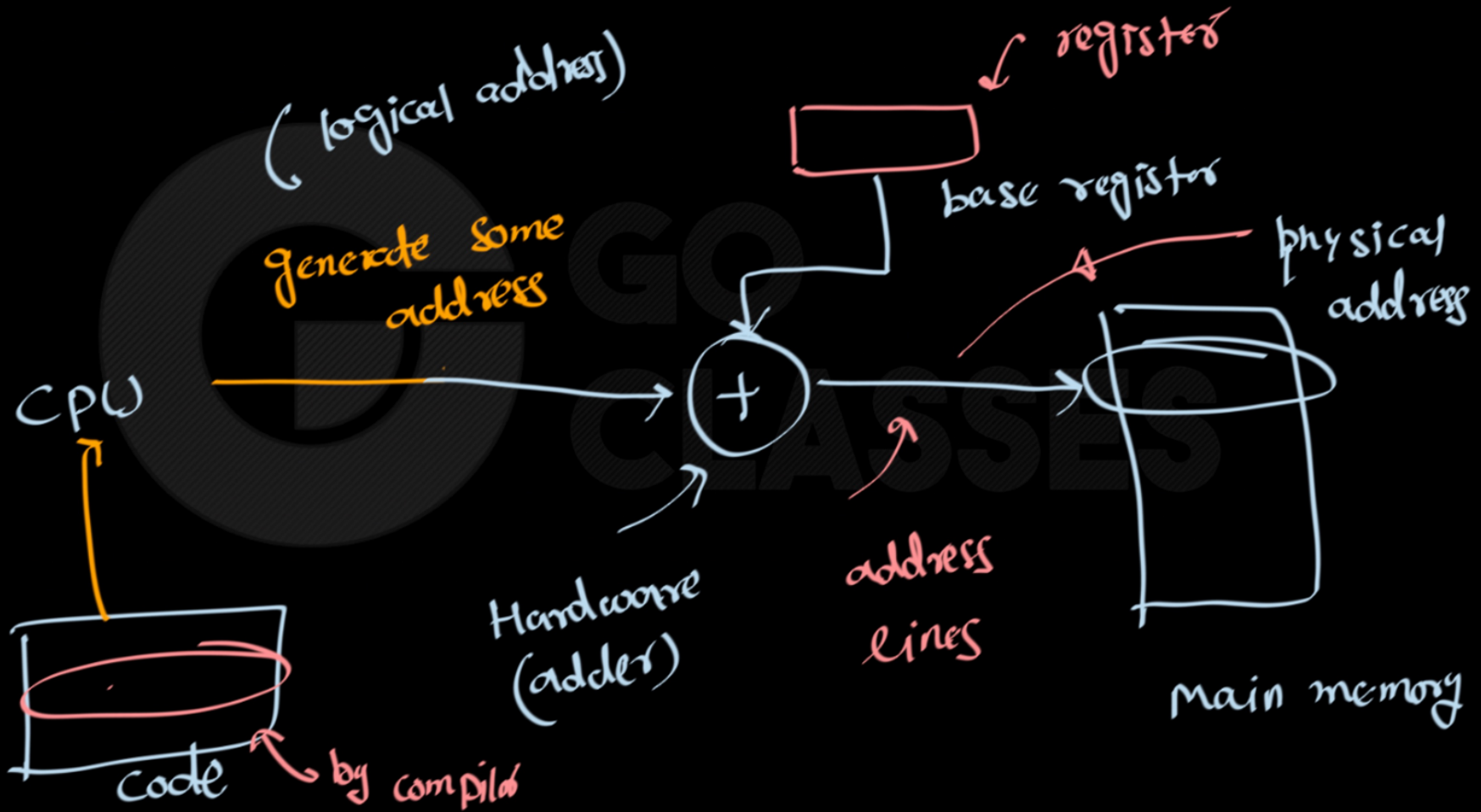
Relocation

- One way to achieve this is to relocate program at different addresses

→ Base register

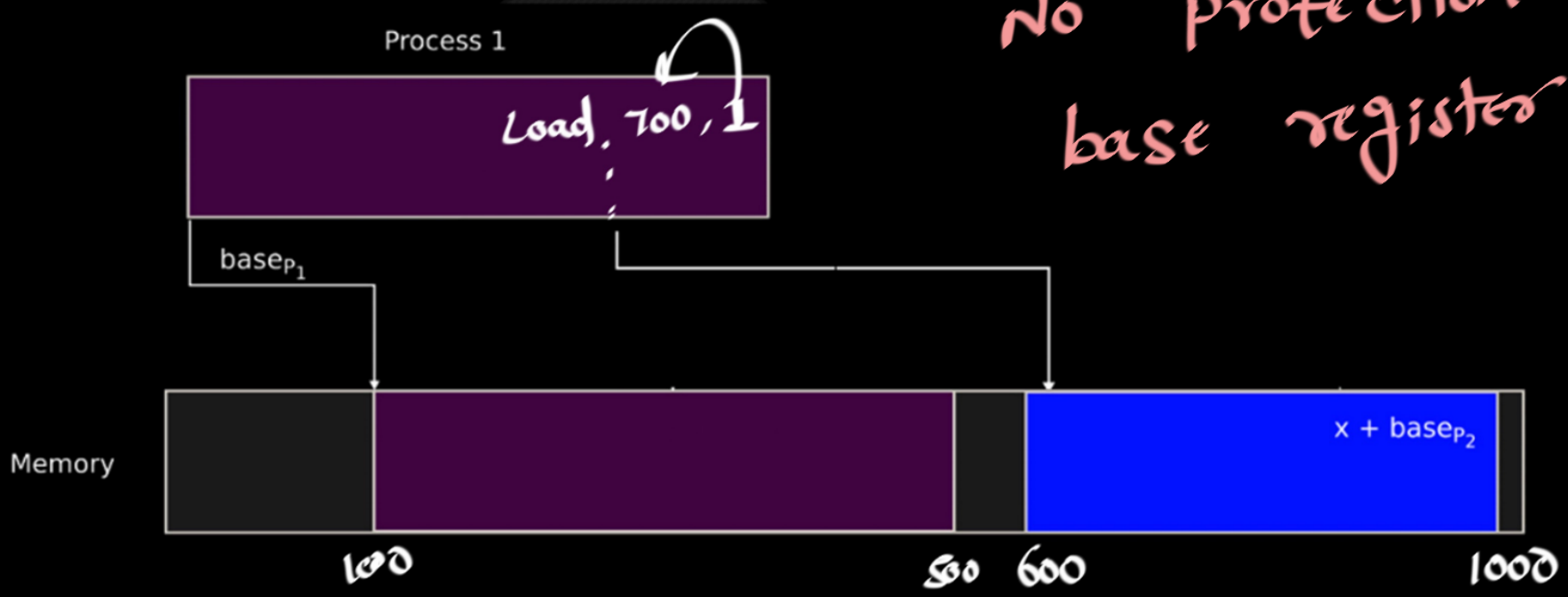
GO CLASSES

Address TRASLATION IN BASE Setup



Two processes, one memory?

What if Process1 generate some illegal address ?



No protection with base register

Base and Bound

(limit register)

- 2 hardware registers: base and bound
- A process can only access physical memory in [base, base+bound)

Process size

Process Size = bound (limit)

logical address ≡ [0, bound − 1] or [0, bound)

limit − 1 ; limit

Base and Bound

- 2 hardware registers: base and bound
- A process can only access physical memory in [base, base+bound)
- On a context switch:
  - Save/restore base and bound registers

← Providing Protection

Base+Bounds Advantages

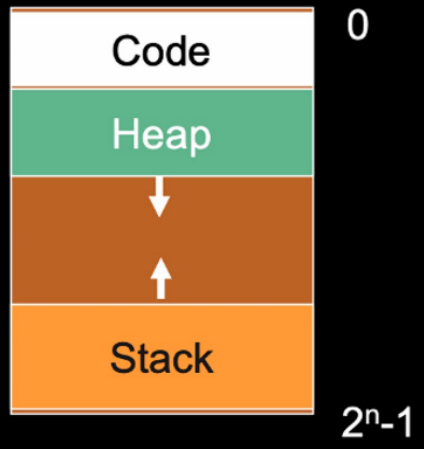
- Provides protection across address spaces
- Supports dynamic relocation
- Simple, inexpensive, fast HW implementation
  - Few registers, little logic in MMU
- Simple context switching logic
  - Save and restore a couple of registers

https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa17/cse306/slides/05-virtual_memory.pdf

Base+Bounds Disadvantages

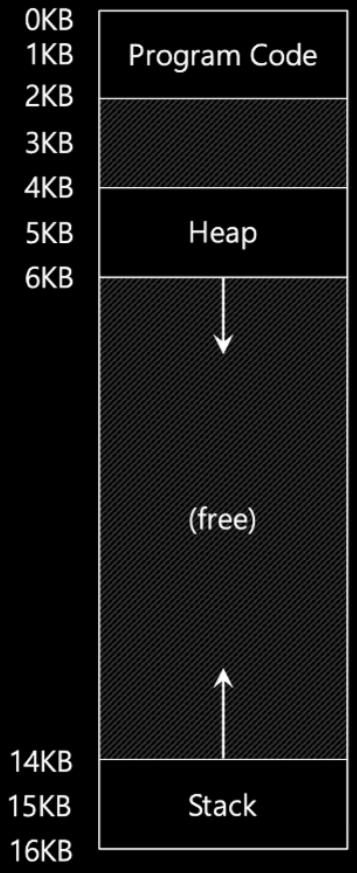
- Each process must be allocated contiguously in physical memory
  - Must allocate memory that may not be used by process

disadvantage



https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa17/cse306/slides/05-virtual_memory.pdf

Why not just Base and Bound?



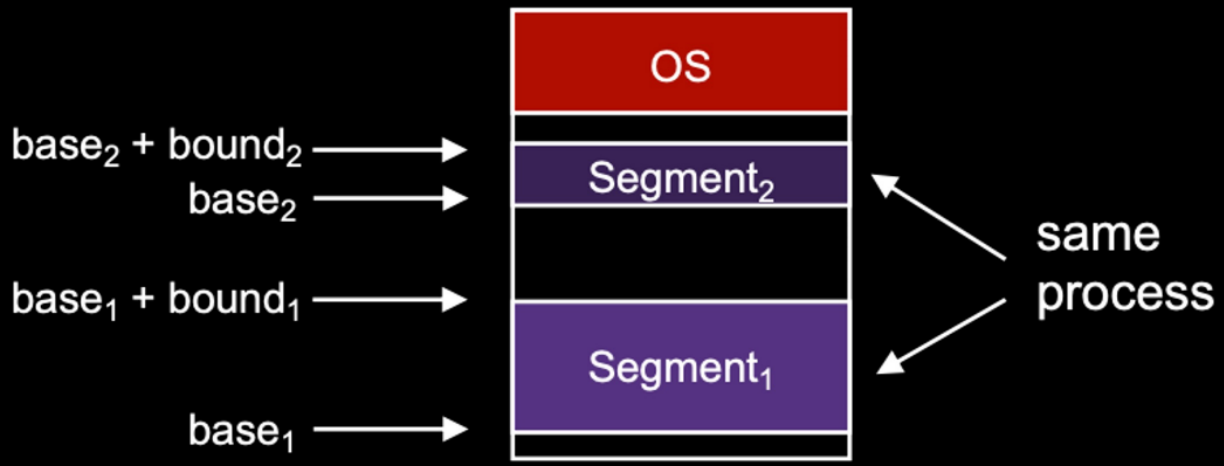
- Big chunk of “free” space
- “free” space takes up physical memory.

https://www.cs.unm.edu/~crandall/operatingsystems20/slides/14-Address-Space-Segmentation.pdf

Segmentation

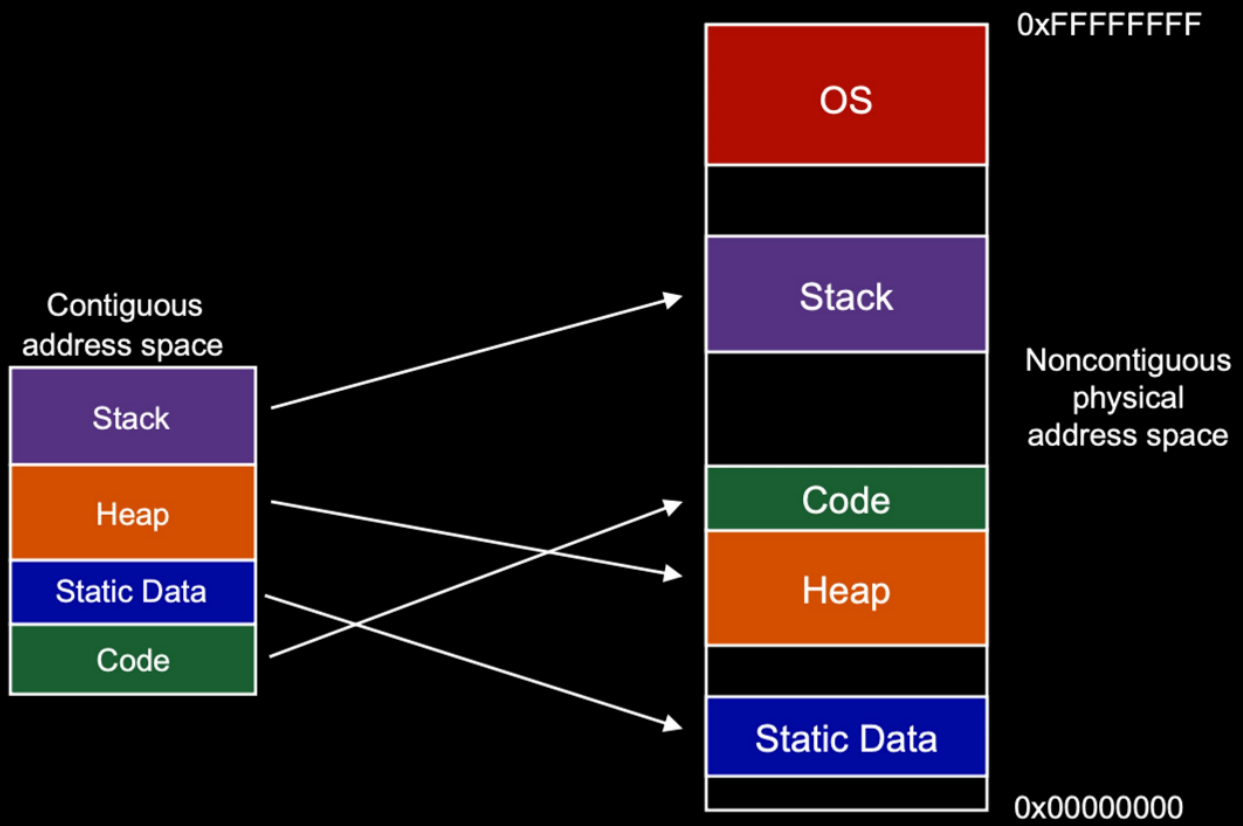
Segmentation

- Split each process' logical address space into multiple segments
- Segment: variable-sized area of memory
- Natural extension of base and bound
  - Base and bound: 1 segment per process
  - Segmentation: many segments per process

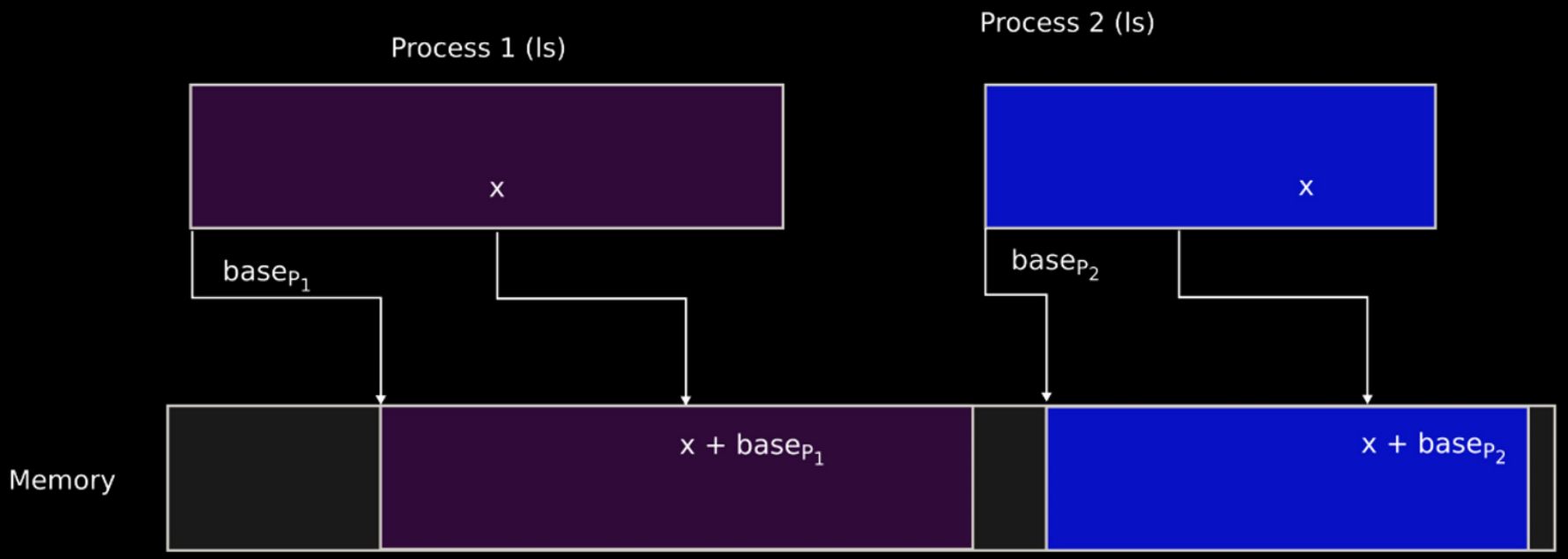


No longer

Contigious memory needed


Contiguous address space
Stack
Heap
Static Data
Code
OS
Stack
Code
Heap
Static Data
0xFFFFFFFF
Noncontiguous physical address space
0x00000000

Segmentation is ok... but

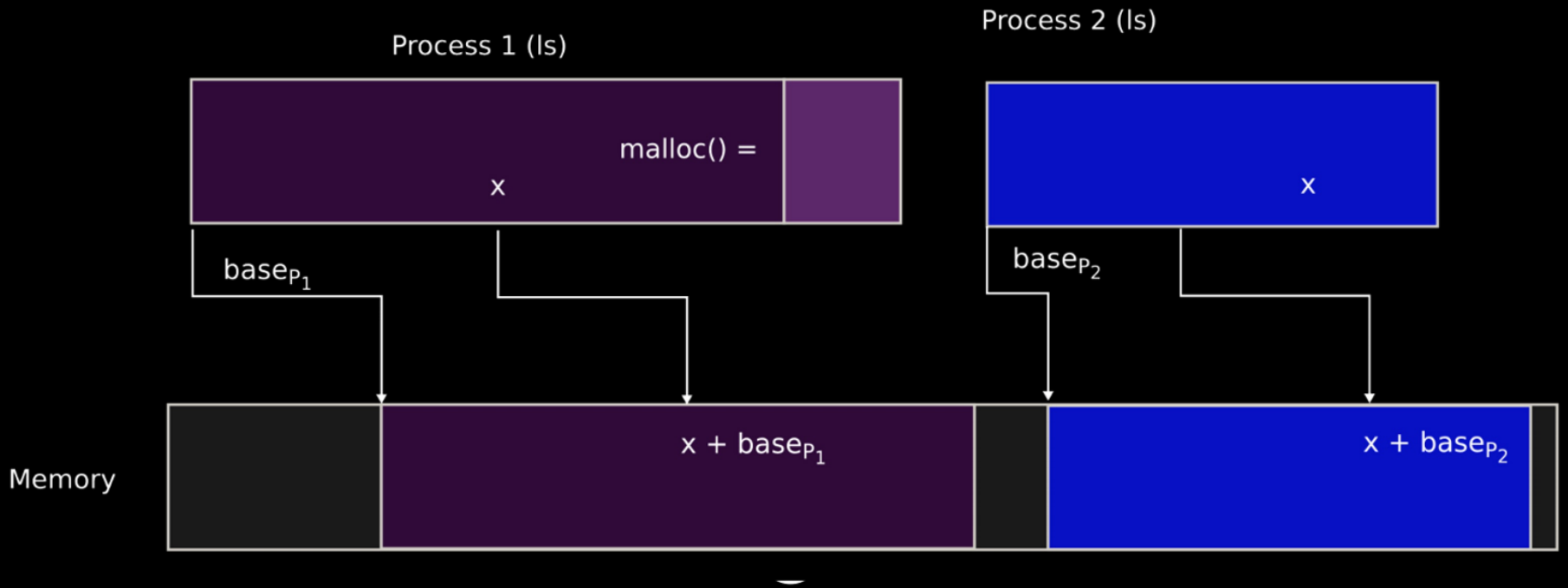


https://users.cs.utah.edu/~aburtsev/238P/lectures/lecture06-address-translation/lecture06-address-translation.pdf

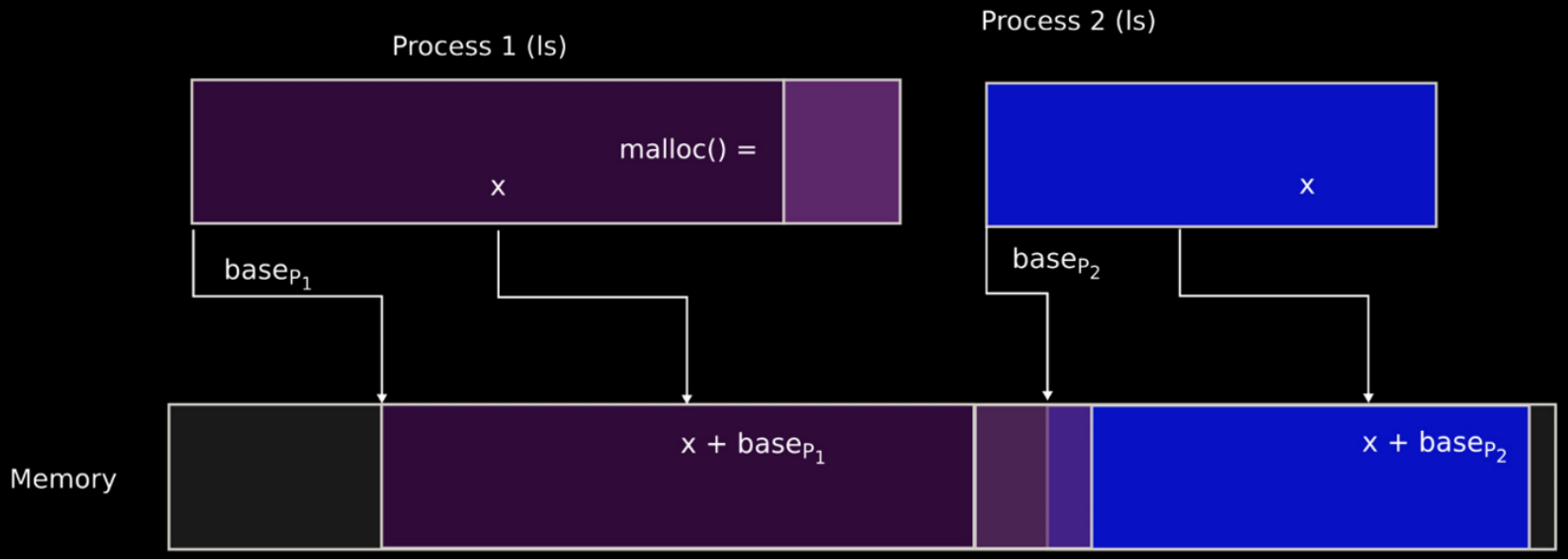
Segmentation is ok... but

initially we do not have idea about segment sizes (specifically for heap, and stack)
(can not)

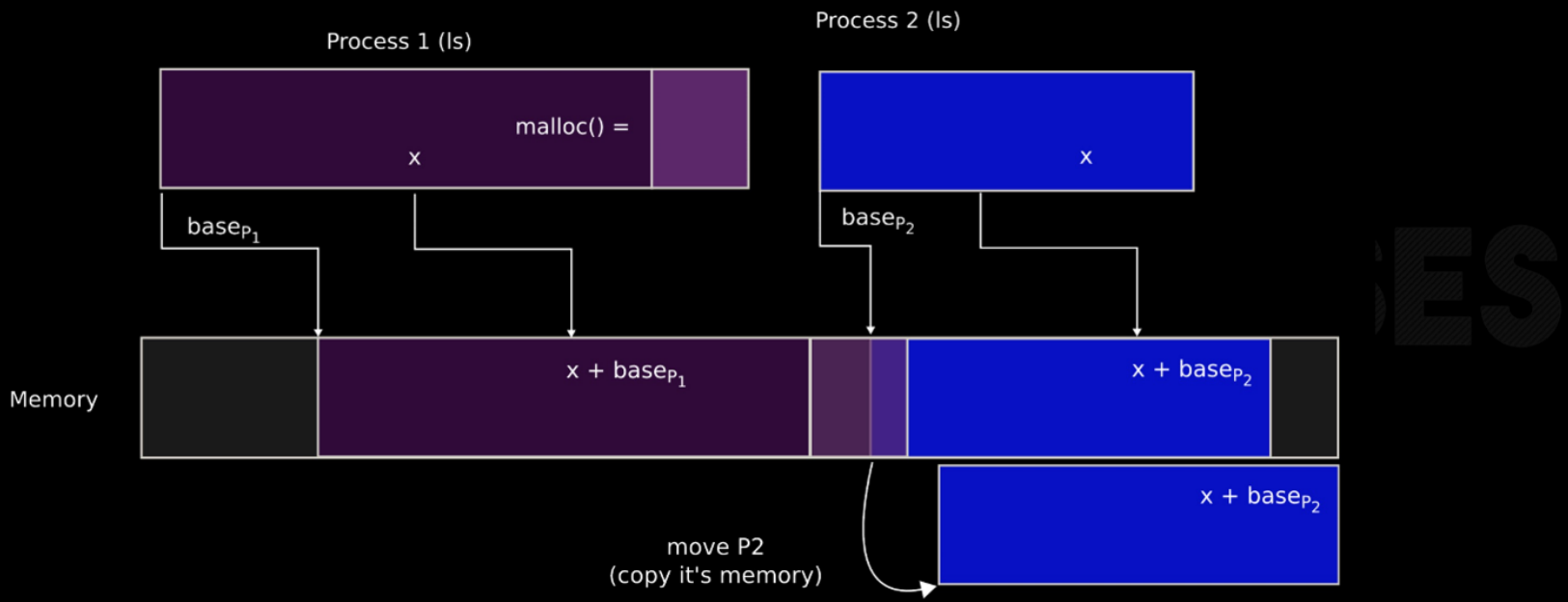
What if process needs more memory?



What if process needs more memory?



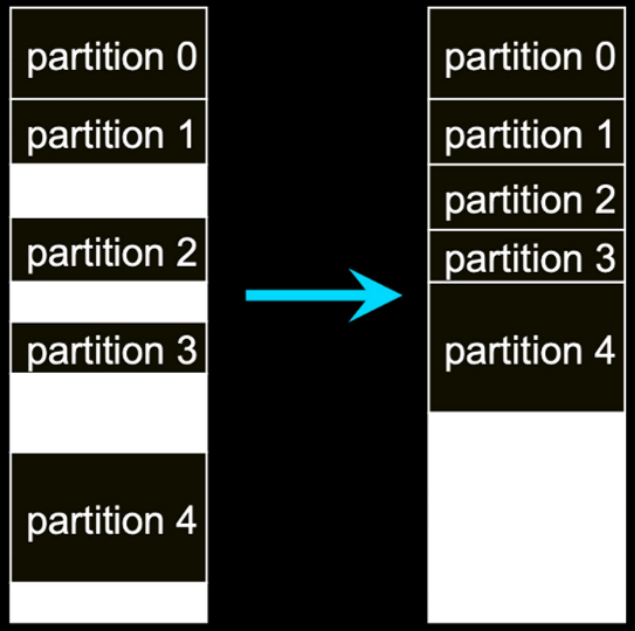
You can move P2 in memory



- Problems
  - External fragmentation
    - Segments of many different sizes have to be allocated contiguously
    - This problem also applies to base and bound schemes

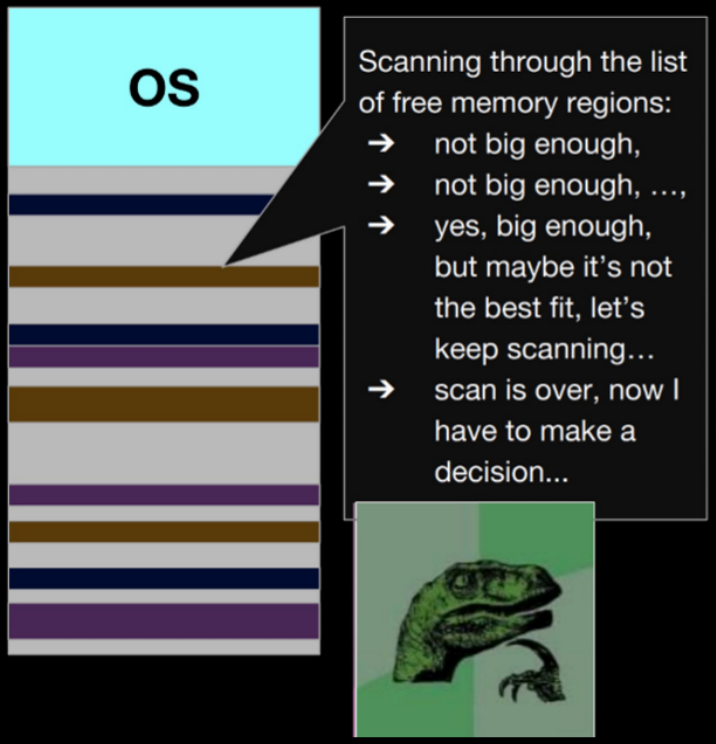
Dealing with fragmentation

- Compact memory by copying
  - Swap a program out
  - Re-load it, adjacent to another
  - Adjust its base register
  - “Lather, rinse, repeat”
  - Ugh



Free space management, when allocating a new chunk...

Segmentation



Questions on Base-Bound and Segmentation

Question

5. Assume you have the following process table (list of active processes):

```
[ Process A  base:100  bounds:10 ]
[ Process B  base:1000 bounds:20 ]
[ Process C  base:500  bounds:50 ]
```

Assume process A is running on a CPU. After the switch to process B, which of the following is a physical address that process B might legally refer to?

a. 0
b. 20
c. 500
d. 1015
e. None of the above

1000 — 1019

Question

5. Assume you have the following process table (list of active processes):

```
[ Process A  base:100  bounds:10 ]
[ Process B  base:1000 bounds:20 ]
[ Process C  base:500  bounds:50 ]
```

Assume process A is running on a CPU. After the switch to process B, which of the following is a physical address that process B might legally refer to?

a. 0
b. 20
c. 500
d. 1015
e. None of the above

1000 — 1019

5. Assume you have the following process table (list of active processes):

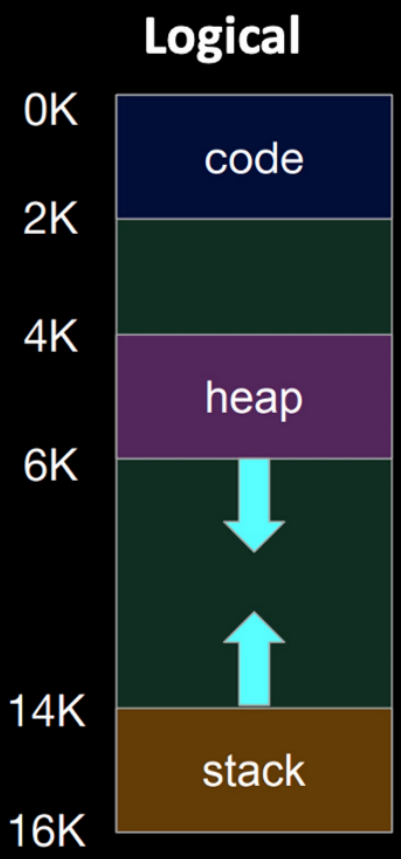
```
[ Process A  base:100  bounds:10 ]
[ Process B  base:1000 bounds:20 ]
[ Process C  base:500  bounds:50 ]
```

Assume process A is running on a CPU. After the switch to process B, which of the following is a physical address that process B might legally refer to?

a. 0
b. 20
c. 500
d. 1015
e. None of the above

B's base is 1000, bounds 20
=> 1000 ... 1019 are legal
=> 1015

Question



What will be physical address corresponding to
- logical address 100
- logical address 4200
- logical address 7000
- logical address 15K

seg	base	bound/size
code	32K	2K
heap	34K	2K
stack	28K	2K

Solution

- → check: 100 in code segment
- → physical = code base + offset
  - ◆ 32K + 100 = **32868**

address: 7000

- → check: not in any segment
- → **Segmentation fault**

- → check: 4200 in heap segment
- → physical = heap base + offset
  - ◆ 34K + (4200 - 4K) = **34920**

Stack

address: 15K ?

- → For home thinking / reading

Question

Consider the following Segment Table:

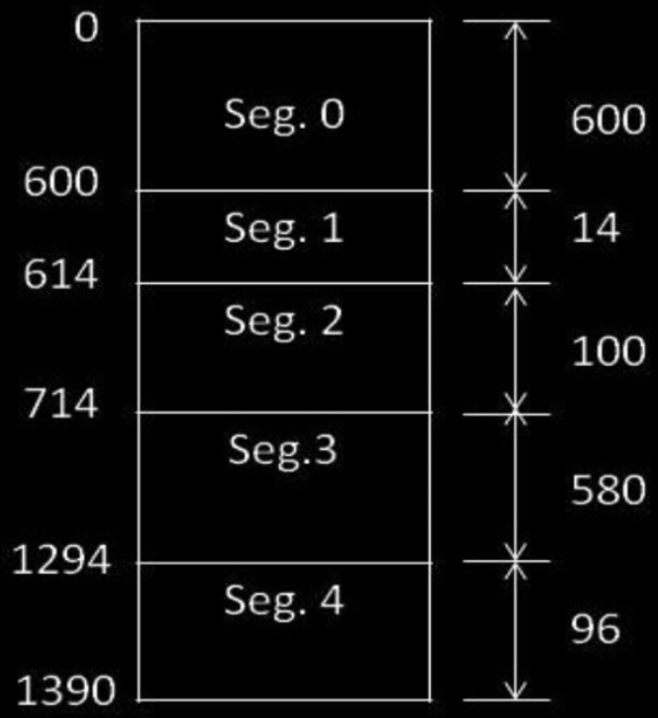
Base	Limit
219	600
2300	14
90	100
1327	580
1952	96

i. What are the physical addresses corresponding to the following two logical addresses:
1000, 2400

ii. What is the logical address for the physical address: 1375

Good

Solution : The logical address space of the process is drawn below according to the segment sizes of the 5 segments given in the segment table.



i.

So, logical address1000 is in Seg. No. 3
Offset of the address in its segment = 1000 – 714 (base logical addr. Of Seg. 3)
= 286
Hence, the corresponding physical address = 1327 (base physical addr. Of Seg. 3) + 286
= 1613

2400 is beyond the logical address space of the process. Hence, it will generate a TRAP: Addressing Error. (Ans.)

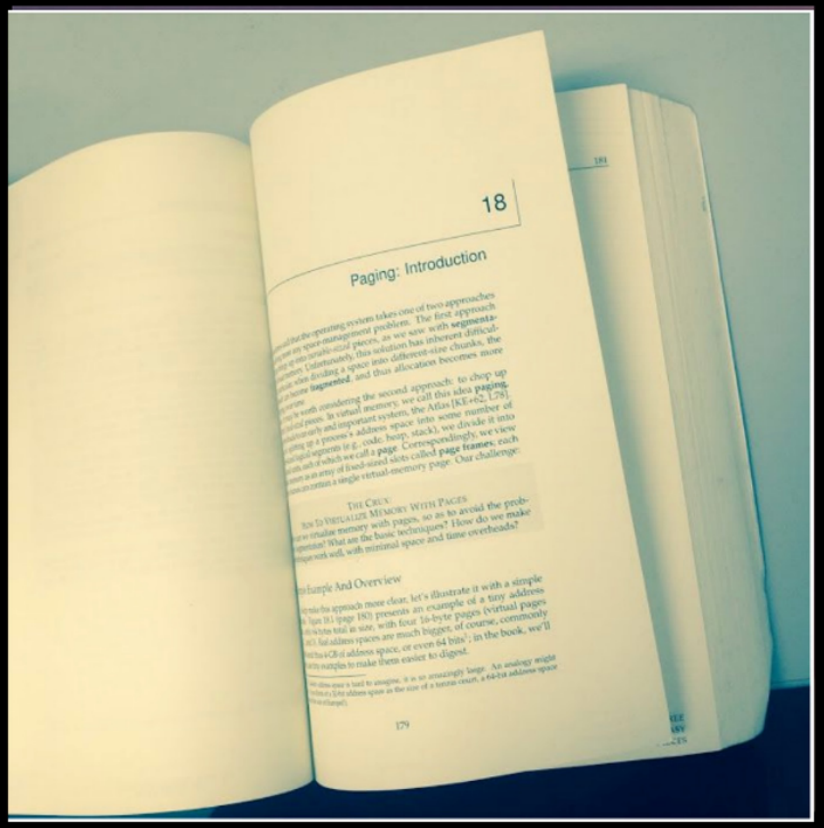
ii.

Physical address 1375 belongs to the space allocated to Seg. 3 in main memory. [Because physical address space for Seg. 3 is 1327 to (1327+580) = 1907. // See the Segment table give in the question paper.]

So, offset of this physical address = 1375 – 1327 (base physical addr. Of Seg. 3) = 48
Hence, the corresponding logical address = 714 (base logical addr. Of Seg. 3) + 48
= 762

Paging

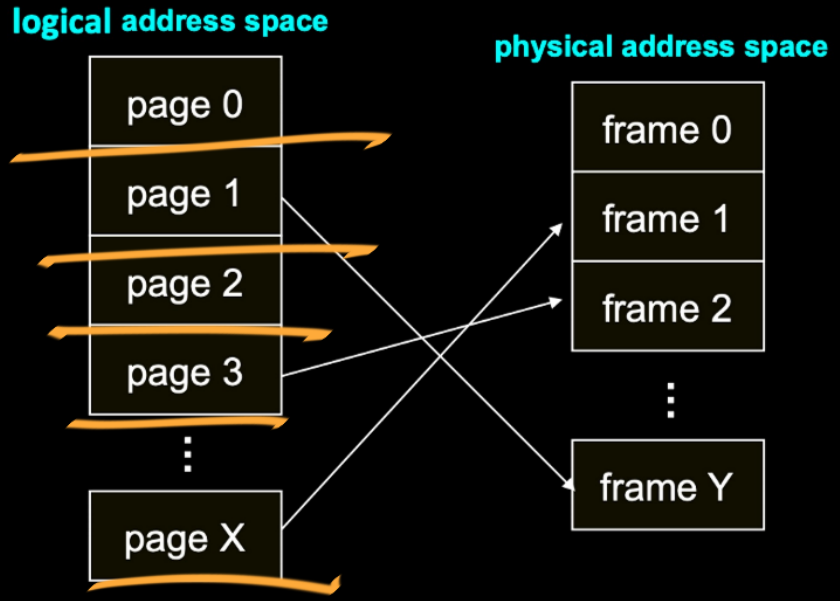
Not really a new idea...



- Divide content into **fixed-sized** pages.
- Each page has a page number.
- Locate content in a page by an offset, e.g., “10th word of Page 8”, …
- There is a “table” which tells you which content is in which page.

Modern technique: Paging

- Solve the external fragmentation problem by using fixed sized units in both physical and logical memory
- Solve the internal fragmentation problem by making the units small



Address Translation

Logical page number (LPN) to Physical page number (PPN) using page table.

$$\frac{Logical\ address}{page\_size}$$

$$Frame\ Number = page\_table\left[\frac{Logical\ address}{page\_size}\right]$$

Page_number

$$offset = logical\ address\ \%page\_size$$

Address Translation

Logical page number (LPN) to Physical page number (PPN) using page table.

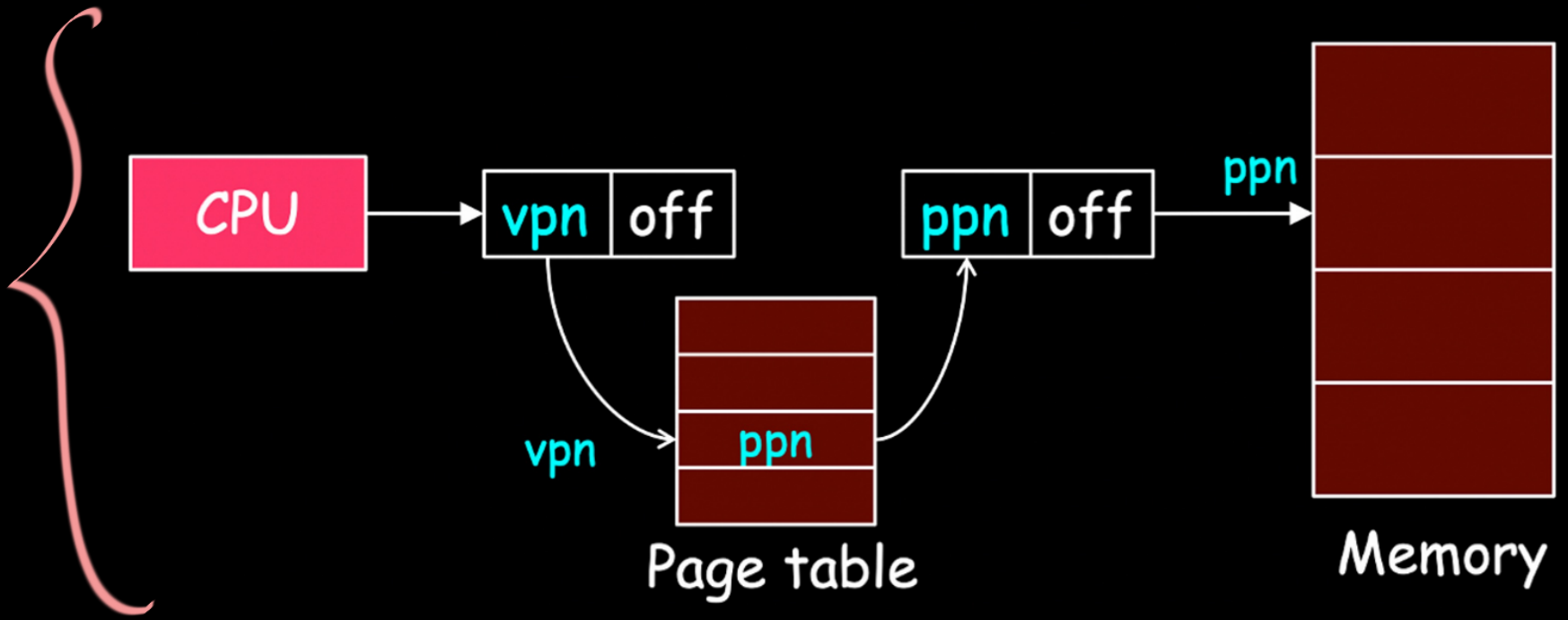
$$\frac{Logical\ address}{page\_size}$$

$$Frame\ Number = page\_table\left[\frac{Logical\ address}{page\_size}\right]$$

$$offset = logical\ address\ \%page\_size$$

Page_number

PA = frameNO x frame size + offset.



https://www.cs.columbia.edu/~junfeng/13fa-w4118/lectures/l05-mem.pdf

Question

Suppose LAS = 1024 Bytes

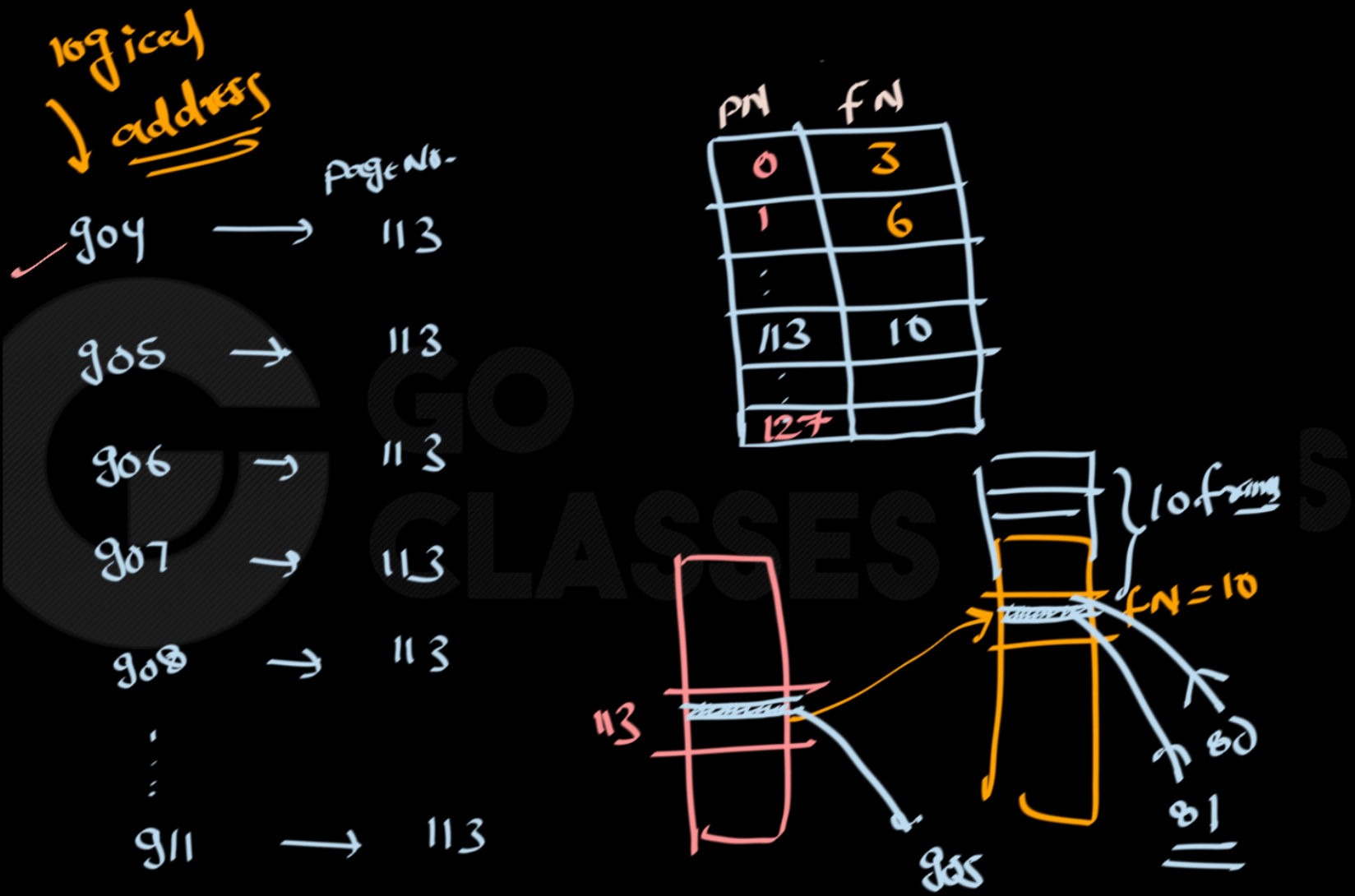
Page size = 8 bytes

and page table is given as follows

PN	FN
0	3
1	6
:	
113	10
:	
127	

Logical → Physical

905 → ??


logical address
Page No.
904 → 113
905 → 113
906 → 113
907 → 113
908 → 113
911 → 113
PN
FN
0
3
1
6
113
10
127
10 frame
FN = 10
113
905
80
81

Question

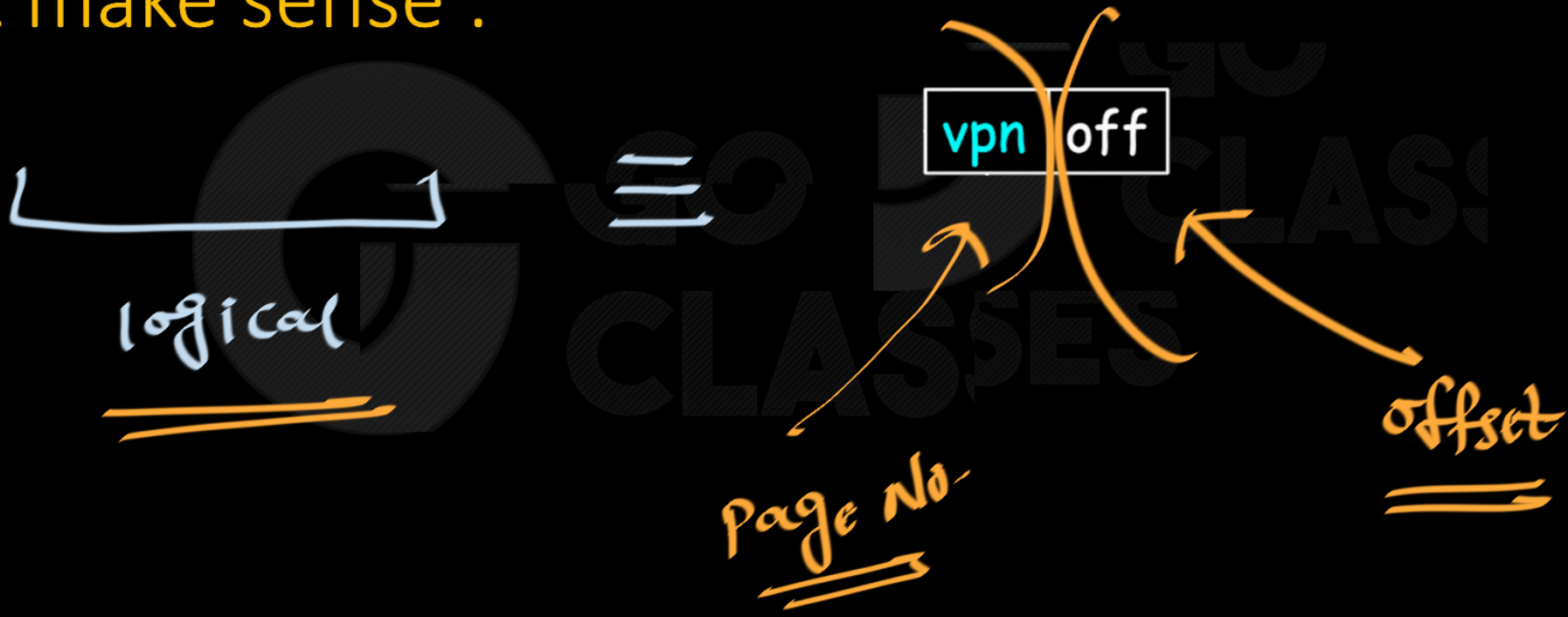
Each page contains = 8 Bytes
800 will be in 100^{th} page number

What will be physical address corresponding to 807 ?

Pg No	Frame No
0	
1	
100	xyz
$2^7 - 1$	

2^7 Entries

Two intuitive ways to understand why this split make sense :



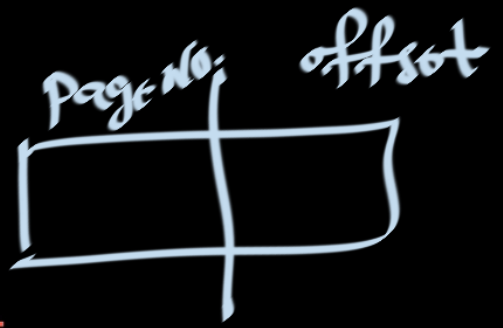
807 = (100 × 8) + 7

Page Size

remainder

logical address

"Power of 2"



Divide by 10

$1298376 \,\%\, 10 = 6$

$1298376 \,\%\, 100 = 76$

$1298376 \,\%\, 1000 = 376$

Decimal division is very good for power of 10.

Divide by 2

$1622 \,\%\, 2 = 0$

$1622 \,\%\, 2^2 = 2$

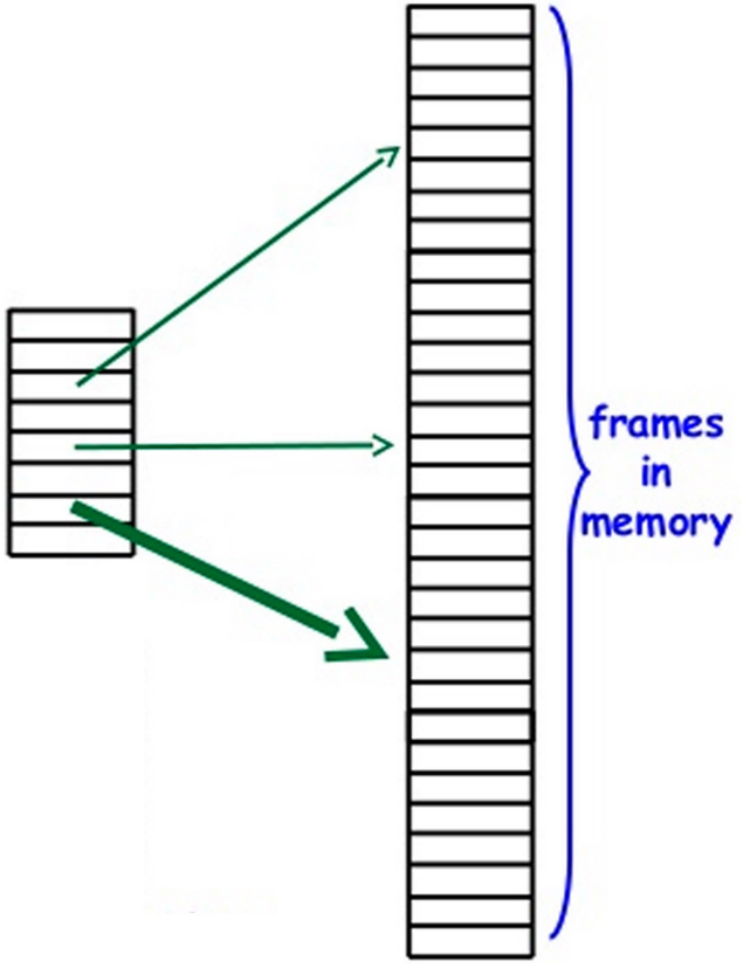
$1622 \,\%\, 2^3 = 6$

NO Shortcut

logical addr

page size

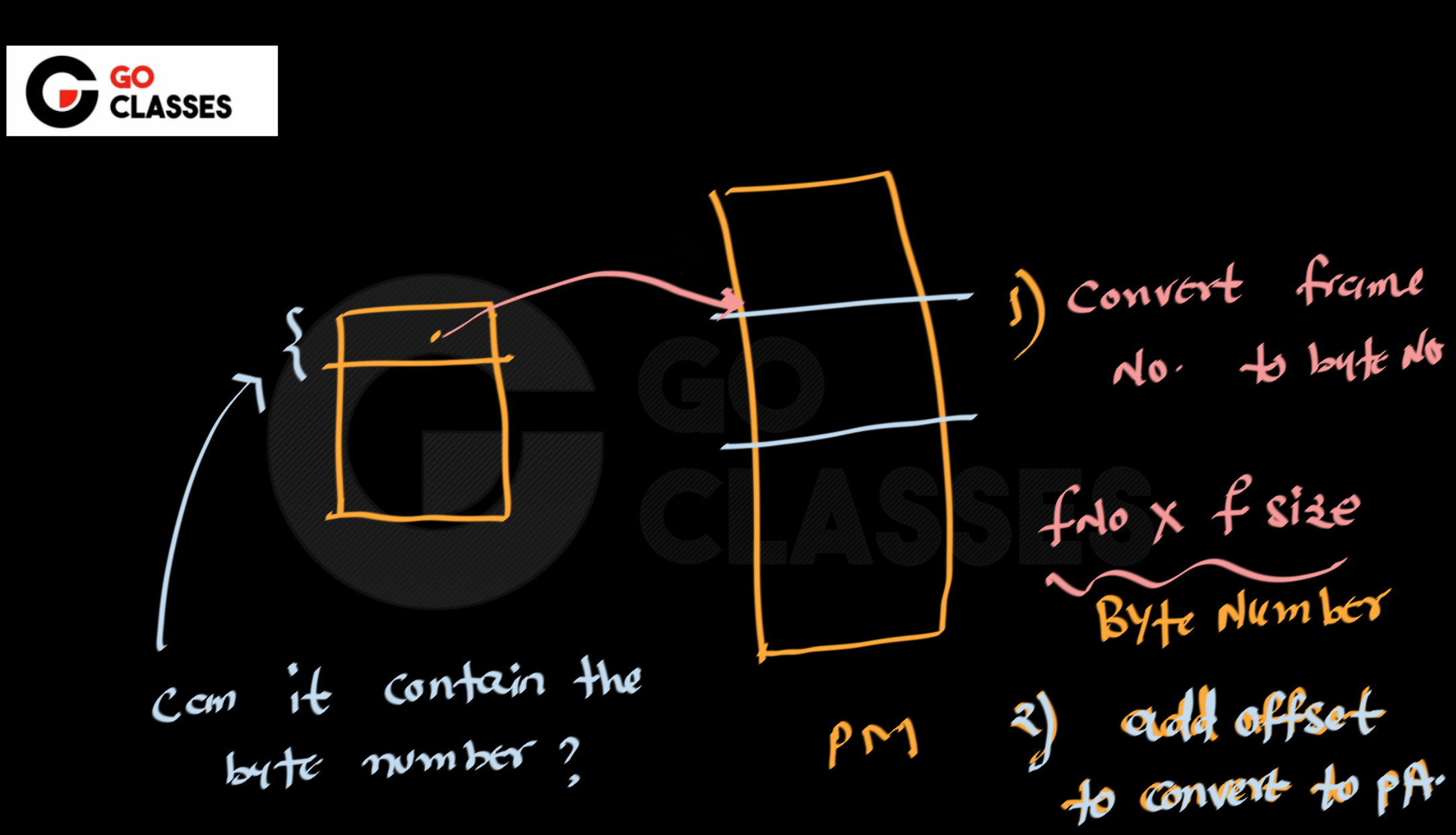
What does Page table entry contain ?



GO CLASSES

What does Page table entry contain ?

} ⇒ frame No. and few other bits


GO CLASSES
1) Convert frame no. to byte no
frNo x f size
Byte Number
Can it contain the byte number ?
PM
2) add offset to convert to PA.

GO CLASSES

Question

we have 8 pages (each size 32 B)

to store in PM of

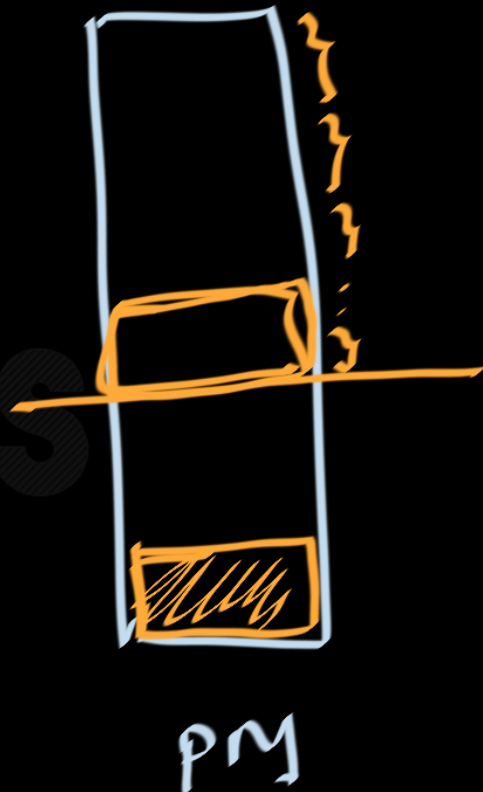
size 2^{32} B

bits required to identify page No. ?

GO CLASSES

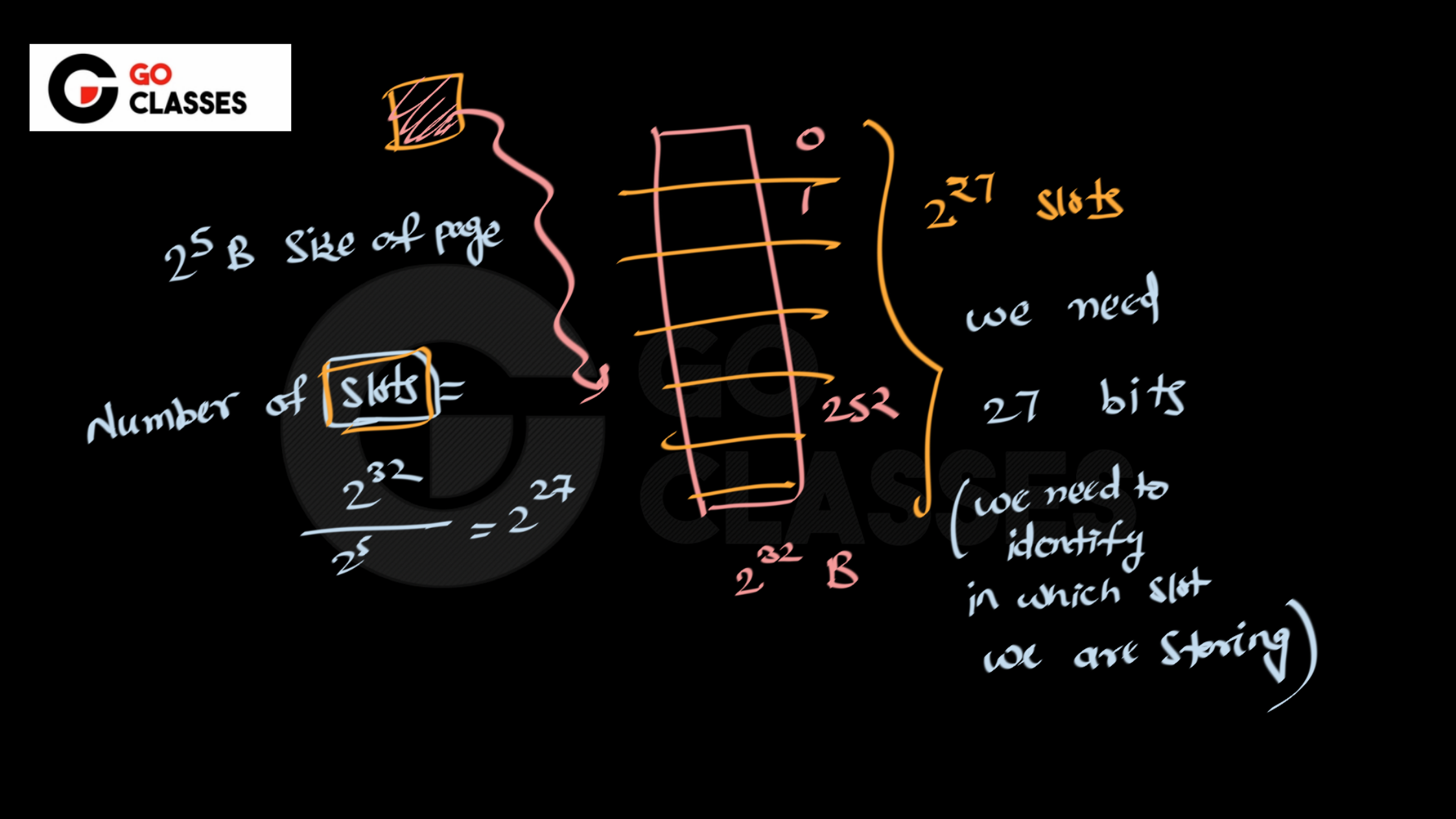
Why 3 is wrong answer

000 — 0
001 — 1
010 — 2
⋮
111 — 7



Number of Pages = 8 } ← doesn't matter to get the answer

Size of the page matters


GO CLASSES
2^5 B Size of page
Number of Slots =
$\frac{2^{32}}{2^5} = 2^{27}$
0
1
252
2^{32} B
2^{27} Slots
we need
27 bits
(we need to identify in which slot we are storing)

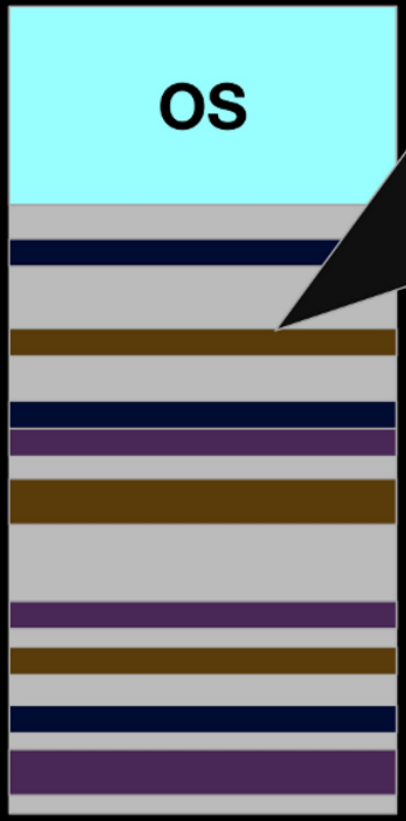
Other PTE Info

- What other info is in PTE besides PFN?
  - **Valid** bit
  - **Protection** bit
  - **Present** bit (needed later)
  - **Referenced** bit (needed later)
  - **Dirty** bit (needed later)

https://compas.cs.stonybrook.edu/~nhonarmand/courses/fa17/cse306/slides/06-paging.pdf

Free space management, when allocating a new chunk...

Segmentation

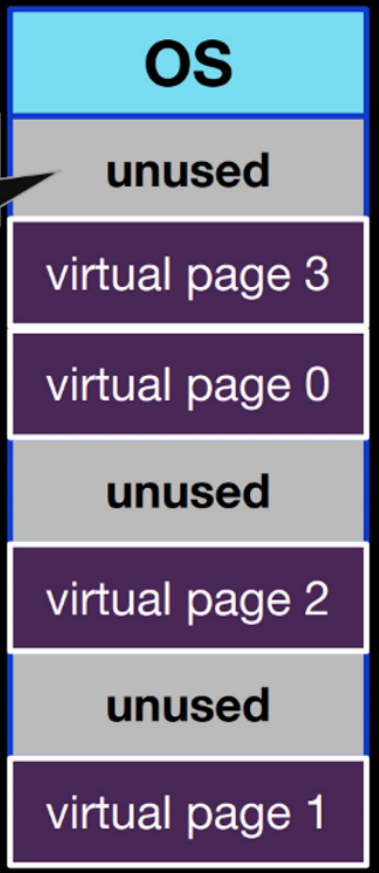


Scanning through the list of free memory regions:

- ➔ not big enough,
- ➔ not big enough, …,
- ➔ yes, big enough, but maybe it's not the best fit, let's keep scanning…
- ➔ scan is over, now I have to make a decision...

Paging

Yo, free page, use!



Paging is simple.

A Typical System has:

Page size = 4 KB, 32 bit LA

So, $\frac{2^{32}}{4\ KB} = \frac{2^{32}}{2^{12}} = 2^{20}\ pages$

There will be 2^{20} page table entries,
if each page table entry is 4B then page size = 4MB

- One such page table for each process.

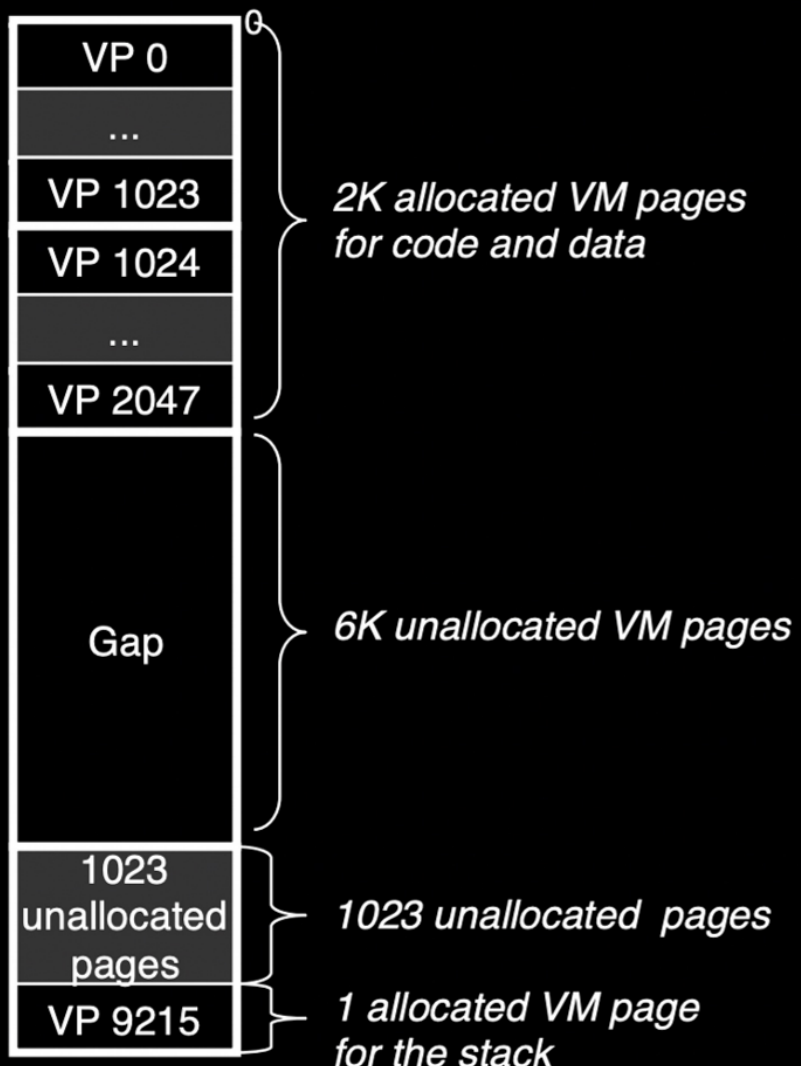
Observation

Usually there is lot of unallocated pages for a process in between heap and stack.

Process just usages few pages.

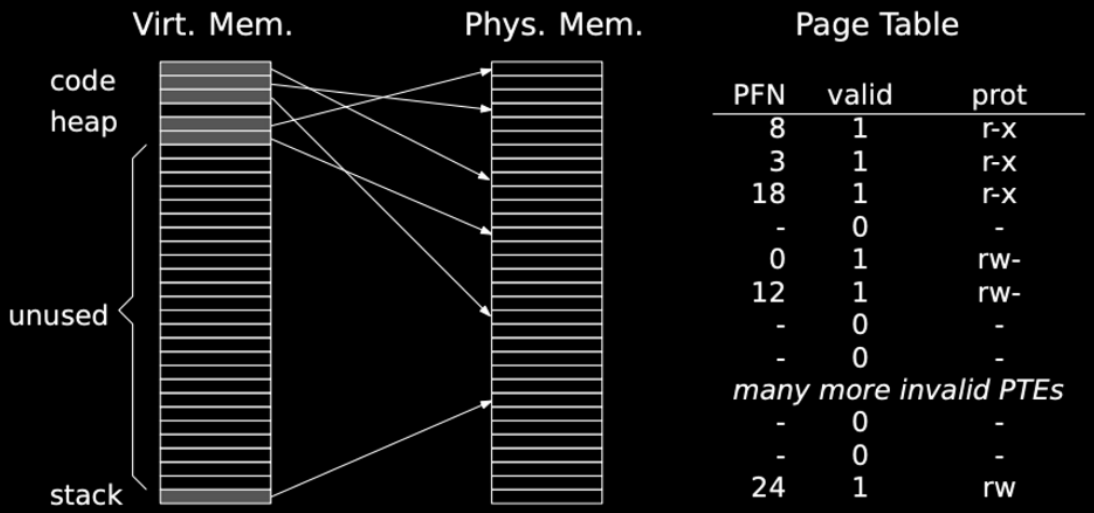
We can not afford this huge page table for every process

Process Size could be too small.


Virtual memory
0
VP 0
...
VP 1023
VP 1024
...
VP 2047
2K allocated VM pages for code and data
Gap
6K unallocated VM pages
1023 unallocated pages
1023 unallocated pages
VP 9215
1 allocated VM page for the stack

Why are page tables so large?

Answer Page tables are full of invalid PTEs



Page Table

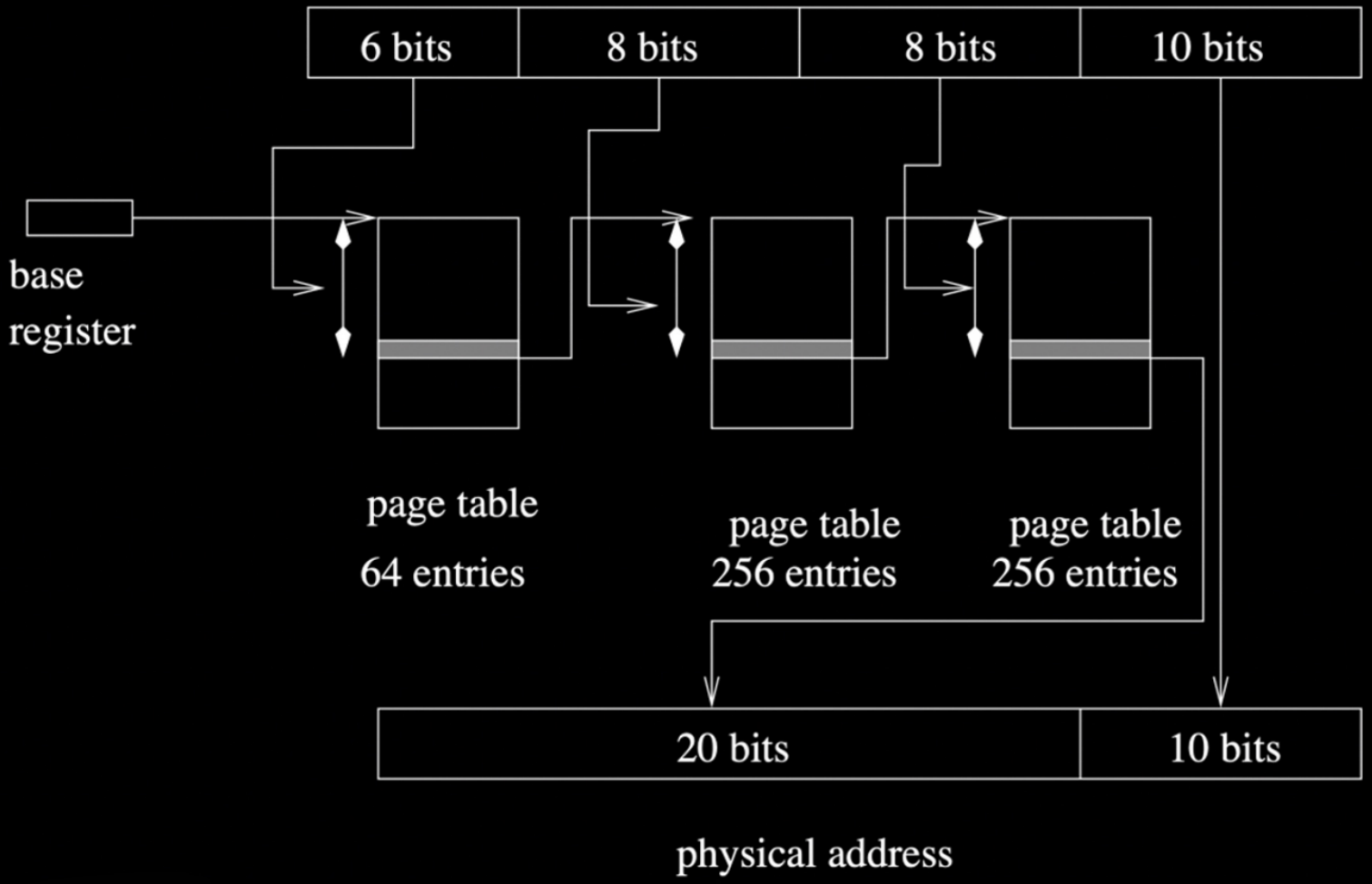
PFN	valid	prot
8	1	r-x
3	1	r-x
18	1	r-x
-	0	-
0	1	rw-
12	1	rw-
-	0	-
-	0	-
many more invalid PTEs		
-	0	-
-	0	-
24	1	rw

Question How to avoid storing invalid PTE?

https://web.fe.up.pt/~pfs/aulas/so2021/at/24vm_tables.pdf

Multi Level Paging

6 bits	8 bits	8 bits	10 bits


6 bits
8 bits
8 bits
10 bits
base
register
page table
64 entries
page table
256 entries
page table
256 entries
20 bits
10 bits
physical address

Consider a 3-level page table on a sytem where pages are 256 bytes, page table entries are 2 bytes, and

- first level page tables contain 16 entries
- second level page tables contains 128 entries
- third level page tables contain 128 entries

What is size of VAS ?

Question

Consider a system, Starting with a 48-bit virtual addresses, it uses 4 levels of page table to translate the address to a 52-bit physical address.

Each page table entry is 8 bytes long.

If the page size is increased, the number of levels of page table can be reduced. How large must pages be in order to translate 48-bit virtual addresses with only a 2 level page table?

Notice first that the page size is 2^12 in Figure 1. The page table size is also 2^12 (2^9 * 8 bytes per page table entry, for a 64 bit processor).

For a two level page table, say page size is 2^n. The page table size is also 2^n bytes in this multi-level page table. So the number of PTE in a page table is 2^(n-3).

So the virtual address is broken into (n-3), (n-3), and n bits.

(n-3) + (n-3) + n = 48

n = 18, so page size is 2^18 = 256KB.

The 48-bit virtual address is divided into 15, 15, 18 bits. The first 15 bits index into the first page table, the second 15 bits index in the second page table, and the rest of the 18 bits is the page offset.

p-2	p-2	p-2	p

logical address space is 46 bits as given. Hence. equation becomes,

$$(p-2)+(p-2)+(p-2)+p=46$$

$$\Rightarrow p=13.$$

Therefore, page size is 2^{13} Bytes = **8KB**.

GATE 2013

https://gateoverflow.in/379/gate-cse-2013-question-52

GATE CSE 2008 | Question: 67

237

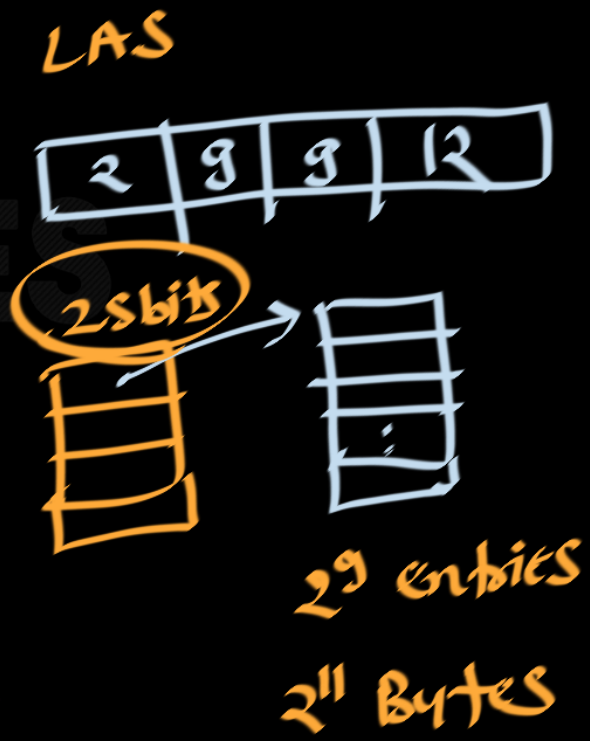
A processor uses 36 bit physical address and 32 bit virtual addresses, with a page frame size of 4 Kbytes. Each page table entry is of size 4 bytes. A three level page table is used for virtual to physical address translation, where the virtual address is used as follows:

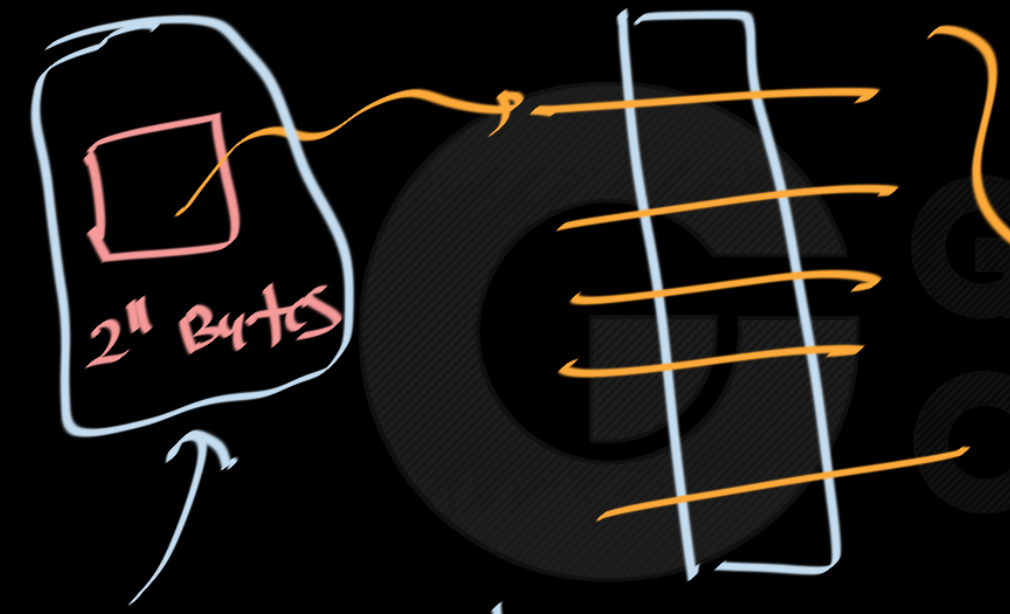
- Bits $30 - 31$ are used to index into the first level page table.
- Bits $21 - 29$ are used to index into the 2nd level page table.
- Bits $12 - 20$ are used to index into the 3rd level page table.
- Bits $0 - 11$ are used as offset within the page.

The number of bits required for addressing the next level page table(or page frame) in the page table entry of the first, second and third level page tables are respectively

A. $20,20,20$
B. $24,24,24$
C. $24,24,20$
D. $25,25,24$

gatecse-2008 operating-system virtual-memory normal





Should i must be having 25 Chunks? ⇒ No.

PM = 2^{36} B

285

Best answer

Physical address is 36 bits. So, number of bits to represent a page frame $= 36 - 12 = 24\ bits$ (12 offset bits as given in question to address 4 KB assuming byte addressing). So, each entry in a third level page table must have 24 bits for addressing the page frames.

A page in logical address space corresponds to a page frame in physical address space. So, in logical address space also we need 12 bits as offset bits. From the logical address which is of 32 bits, we are now left with $32 - 12 = 20\ bits$; these $20\ bits$ will be divided into three partitions (as given in the question) so that each partition represents 'which entry' in the i^{th} level page table we are referring to.

- An entry in level i page table determines 'which page table' at $(i+1)^{th}$ level is being referred.

Now, there is only 1 first level page table. But there can be many second level and third level page tables and "how many" of these exist depends on the physical memory capacity. (In actual the no. of such page tables depend on the memory usage of a given process, but for addressing we need to consider the worst case scenario). The simple formula for getting the number of page tables possible at a level is to divide the available physical memory size by the size of a given level page table.

$$\begin{aligned}\text{Number of third level page tables possible} &= \frac{\text{Physical memory size}}{\text{Size of a third level page table}} \\ &= \frac{2^{36}}{\text{Number of entries in a single third level page table} \times \text{ Size of an entry}} \\ &= \frac{2^{36}}{2^9 \times 4} \because \text{(bits 12-20 gives 9 bits)} \\ &= \frac{2^{36}}{2^{11}} \\ &= 2^{25}\end{aligned}$$

PS: No. of third level page tables possible means the no. of distinct addresses a page table can have. At any given time, no. of page tables at level j is equal to the no. of entries in the level $j-1$, but here we are considering the **possible** page table addresses.

http://www.cs.utexas.edu/~lorenzo/corsi/cs372/06F/hw/3sol.html See Problem 3, second part solution - It clearly says that we should not assume that page tables are page aligned (page table size need not be same as page size unless told so in the question and different level page tables can have different sizes).
So, we need 25 bits in second level page table for addressing the third level page tables.

Similarly we need to find the no. of possible second level page tables and we need to address each of them in first level page table.

Now,

$$\text{Number of second level page tables possible} = \frac{\text{Physical memory size}}{\text{Size of a second level page table}}$$
$$= \frac{2^{36}}{\text{Number of entries in a single second level page table} \times \text{ Size of an entry}}$$
$$= \frac{2^{36}}{2^9 \times 4} \because \text{(bits 21-29 gives 9 bits)}$$
$$= \frac{2^{36}}{2^{11}}$$
$$= 2^{25}$$

So, we need 25 bits for addressing the second level page tables as well.

So, answer is (**D**).

Video Explanation for Multi-level Paging: https://youtu.be/bArypfVmPb8

(Edit:-
There is nothing to edit for such awesome explanation but just adding one of my comment if it is useful - comment. However if anyone finds something to add (or correct) then feel free to do that in my comment.)

answered Nov 14, 2014 • edited Jul 13, 2018 by **kenzou**

edit flag hide comment Unfollow

Pip Box Delete with Reason Wrong Useful

Arjun

GO CLASSES

[Page Size = 4KB

Page table Size = 4MB
(Single)]

↳ we can store the page table into MM
but we need lot of Contiguous Space.

GO CLASSES

Even if we have Contigious 4MB of space, is it worth saving the page table?

↓ Always

↳ No, what if Page table contains too many invalid entries.

Why Multi-level paging ?

- We may not find continuous space to store page table in main memory.

- We have seen most of the pages are empty.

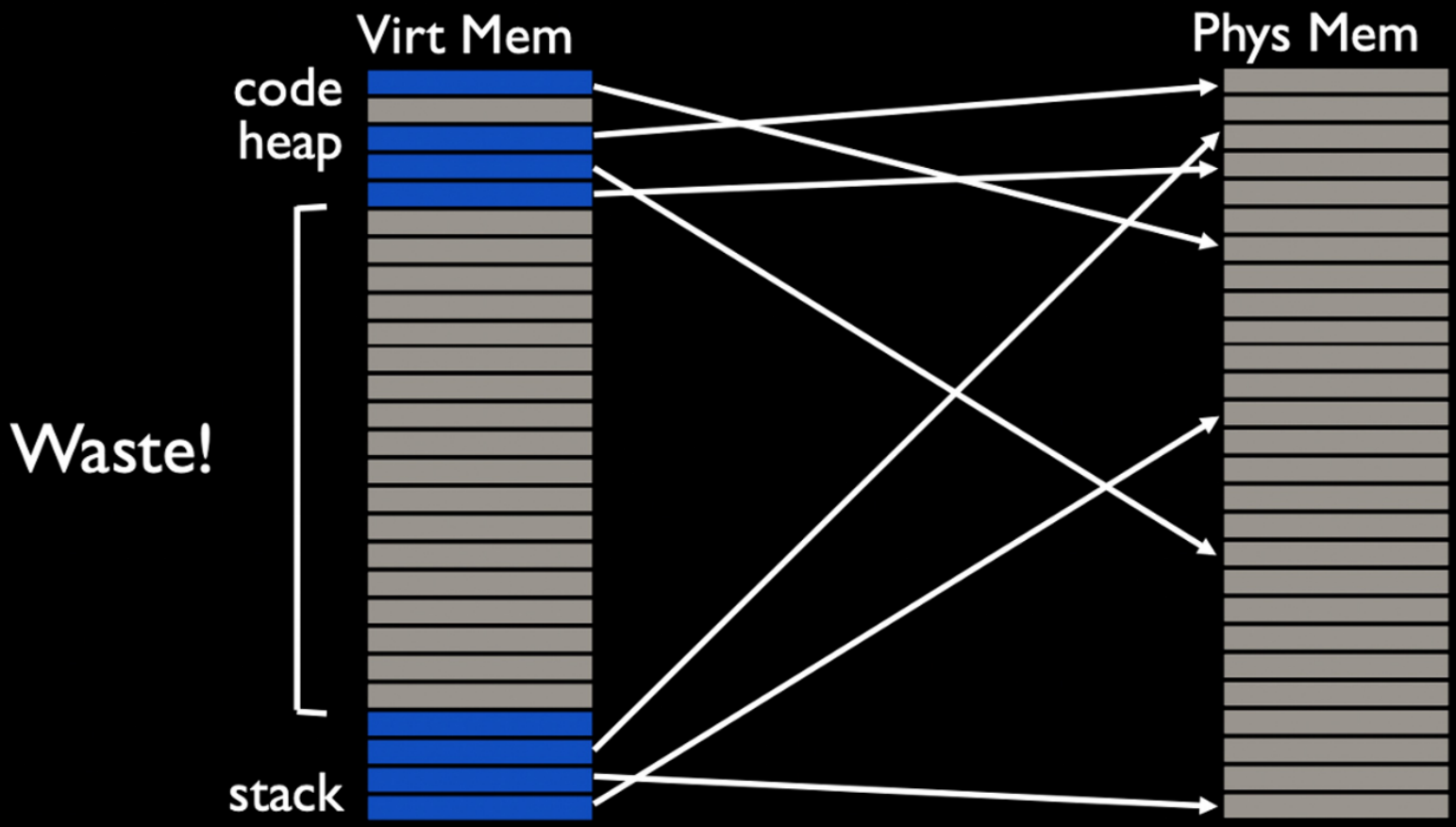
  Why to store entries corresponding to the pages which are empty ?

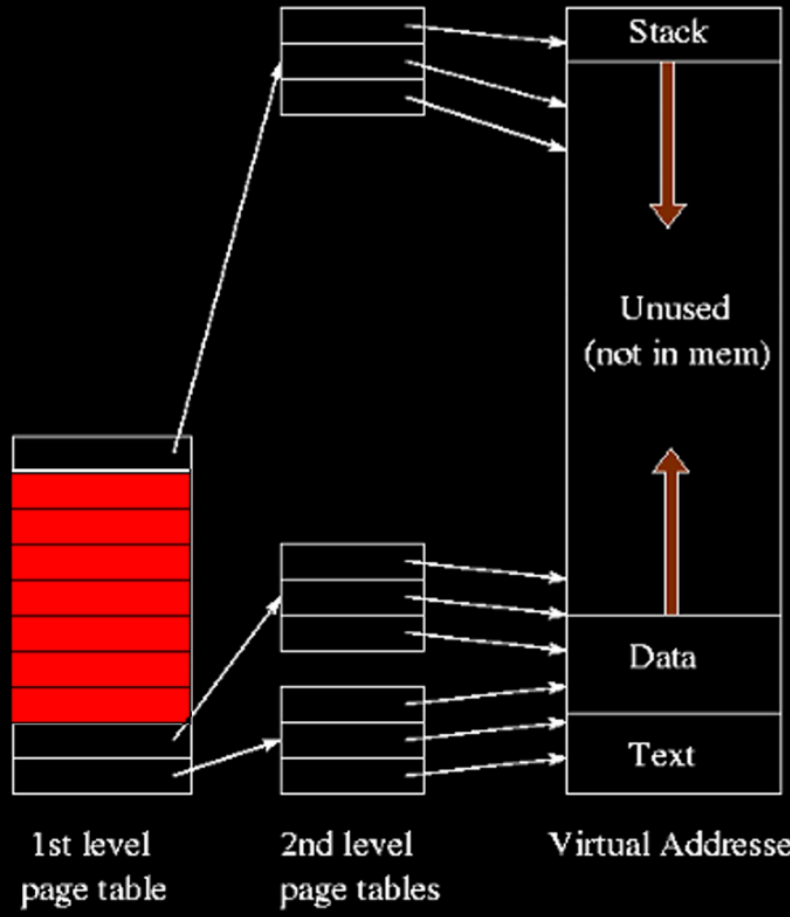
But where are the savings ?

It seems we have added one more page table and not saving any space.

Where are savings that we promised earlier ?

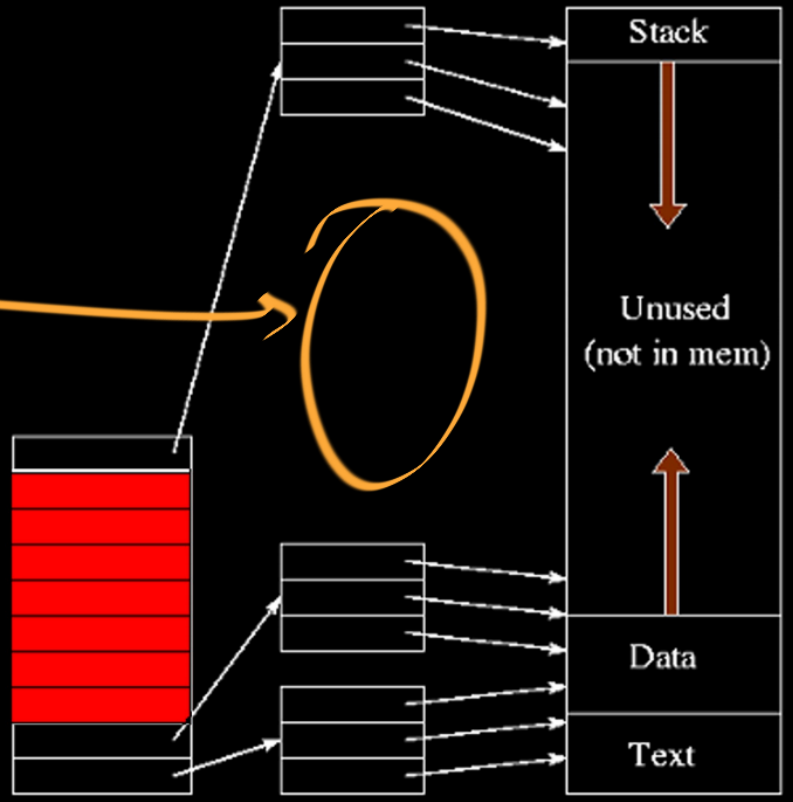
Operating Systems





The red area corresponds to (most of) the unused portion of the virtual address space. Red PTEs are marked as having no frame so cause a page fault if referenced. OS takes appropriate action (extend stack or data, create a new 2nd level page table).

this part of page table is not stored anywhere



The red area corresponds to (most of) the unused portion of the virtual address space. Red PTEs are marked as having no frame so cause a page fault if referenced. OS takes appropriate action (extend stack or data, create a new 2nd level page table).

Problem 1

In a 32-bit machine we subdivide the virtual address into 4 segments as follows:

We use a 3-level page table, such that the first 10-bit are for the first level and so on.

1. What is the page size in such a system?
2. What is the size of a page table for a process that has 256K of memory starting at address 0?
3. What is the size of a page table for a process that has a code segment of 48K starting at address 0x1000000, a data segment of 600K starting at address 0x80000000 and a stack segment of 64K starting at address 0xf0000000 and growing upward (like in the PA-RISC of HP)?

https://www.cs.utexas.edu/~lorenzo/corsi/cs372/06F/hw/3sol.html

Problem 1

In a 32-bit machine we subdivide the virtual address into 4 segments as follows:

10-bit	8-bit	6-bit	8 bit

We use a 3-level page table, such that the first 10-bit are for the first level and so on.

1. What is the page size in such a system?
2. What is the size of a page table for a process that has 256K of memory starting at address 0?

Solution:

1. The page field is 8-bit wide, then the page size is 256 bytes.

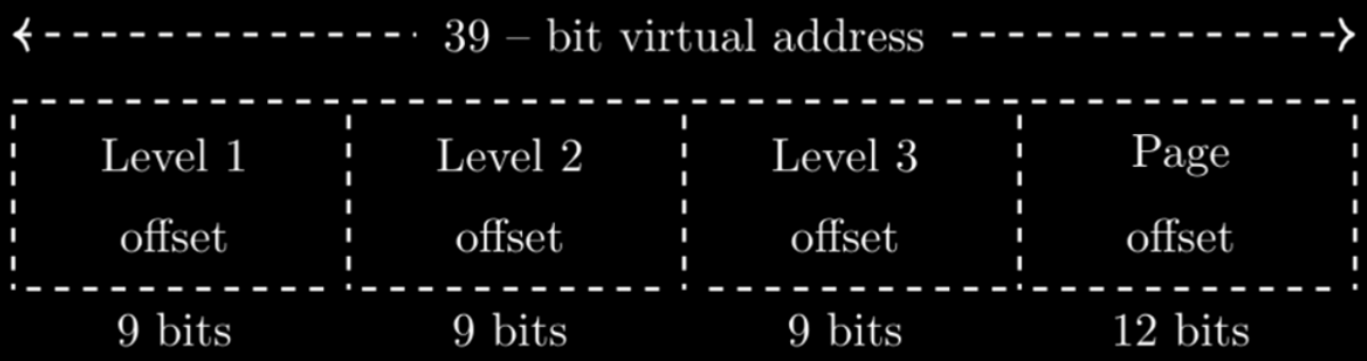
2. Using the subdivision above, the first level page table points to 1024 2nd level page tables, each pointing to 256 3rd page tables, each containing 64 pages. The program's address space consists of 1024 pages, thus we need we need 16 third-level page tables. Therefore we need 16 entries in a 2nd level page table, and one entry in the first level page table. Therefore the size is: 1024 entries for the first table, 256 entries for the 2nd level page table, and 16 3rd level page table containing 64 entries each. Assuming 2 bytes per entry, the space required is 1024 * 2 + 256 * 2 (one second-level paget table) + 16 * 64 * 2 (16 third-level page tables) = 4608 bytes.

3. First, the stack, data and code segments are at addresses that require having 3 page tables entries active in the first level page table. For 64K, you need 256 pages, or 4 third-level page tables. For 600K, you need 2400 pages, or 38 third-level page tables and for 48K you need 192 pages or 3 third-level page tables. Assuming 2 bytes per entry, the space required is 1024 * 2 + 256 * 3 * 2 (3 second-level page tables) + 64 * (38+4+3)* 2 (38 third-level page tables for data segment, 4 for stack and 3 for code segment) = 9344 bytes.

GATE CSE 2021 Set 2 | Question: 48

26

Consider a three-level page table to translate a $39-$bit virtual address to a physical address as shown below:



The page size is 4 KB $(1\text{KB} = 2^{10}$ bytes$)$ and page table entry size at every level is 8 bytes. A process P is currently using 2GB $(1\text{GB} = 2^{30}$ bytes$)$ virtual memory which is mapped to 2GB of physical memory. The minimum amount of memory required for the page table of P across all levels is _____ KB.

Answer: 4108

GATE CSE 2003 | Question: 79

asked in Operating System Apr 24, 2016 • edited Jun 23, 2018 by Pooja Khatri

17,893 views

69

A processor uses 2-level page tables for virtual to physical address translation. Page tables for both levels are stored in the main memory. Virtual and physical addresses are both 32 bits wide. The memory is byte addressable. For virtual to physical address translation, the 10 most significant bits of the virtual address are used as index into the first level page table while the next 10 bits are used as index into the second level page table. The 12 least significant bits of the virtual address are used as offset within the page. Assume that the page table entries in both levels of page tables are 4 bytes wide.

Suppose a process has only the following pages in its virtual address space: two contiguous code pages starting at virtual address $0x00000000$, two contiguous data pages starting at virtual address $0x00400000$, and a stack page starting at virtual address $0xFFFFF000$. The amount of memory required for storing the page tables of this process is

A. 8 KB
B. 12 KB
C. 16 KB
D. 20 KB

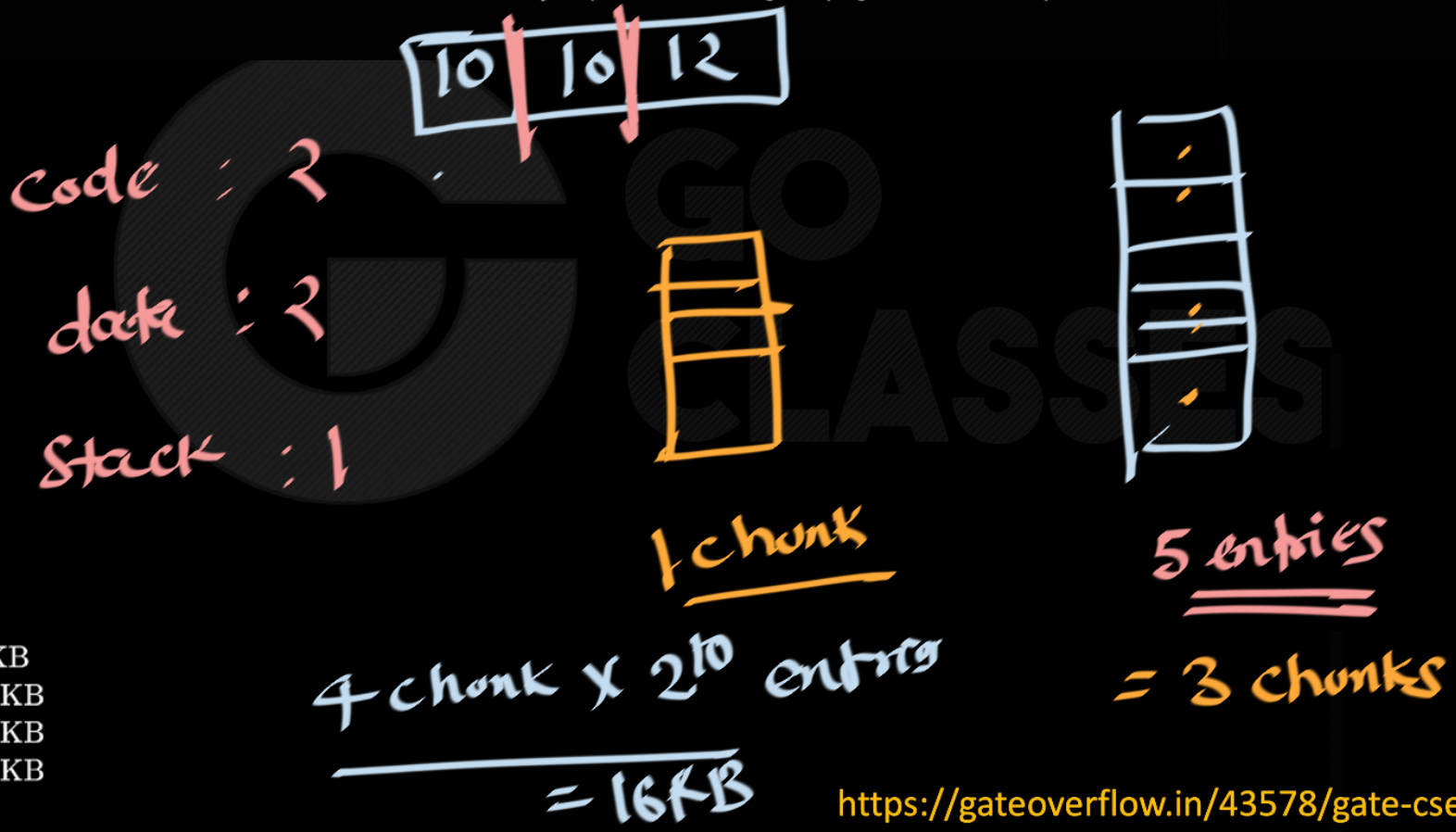
https://gateoverflow.in/43578/gate-cse-2003-question-79

Suppose a process has only the following pages in its virtual address space: two contiguous code pages starting at virtual address $0x00000000$, two contiguous data pages starting at virtual address $0x00400000$, and a stack page starting at virtual address $0xFFFFF000$. The amount of memory required for storing the page tables of this process is

$0x00000000$,

$0x00400000$,

$0xFFFFF000$.



A. 8 KB
B. 12 KB
C. 16 KB
D. 20 KB

https://gateoverflow.in/43578/gate-cse-2003-question-79

code : 2 $0x00000000$,

$0x00400000$,

data : 2 $0xFFFFF000$

stack : 1

1 chunk

5 entries

A. 8 KB
B. 12 KB
C. 16 KB
D. 20 KB

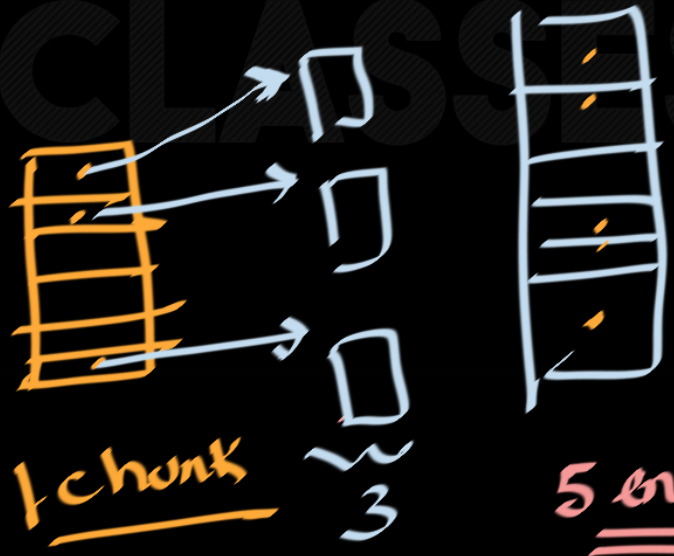
GO CLASSES

code : 2 $0x00000000$, ⇒ 0

data : 2 $0x00400000$, ⇒ 1

stack : 1 $0xFFFFF000$ ⇒ $2^{10}-1$ = 1023

ENTRY numbers of outermost page table



A. 8 KB
B. 12 KB
C. 16 KB
D. 20 KB

GO CLASSES

code : 2 { $0x00000000$, ⇒ 0

data : 2

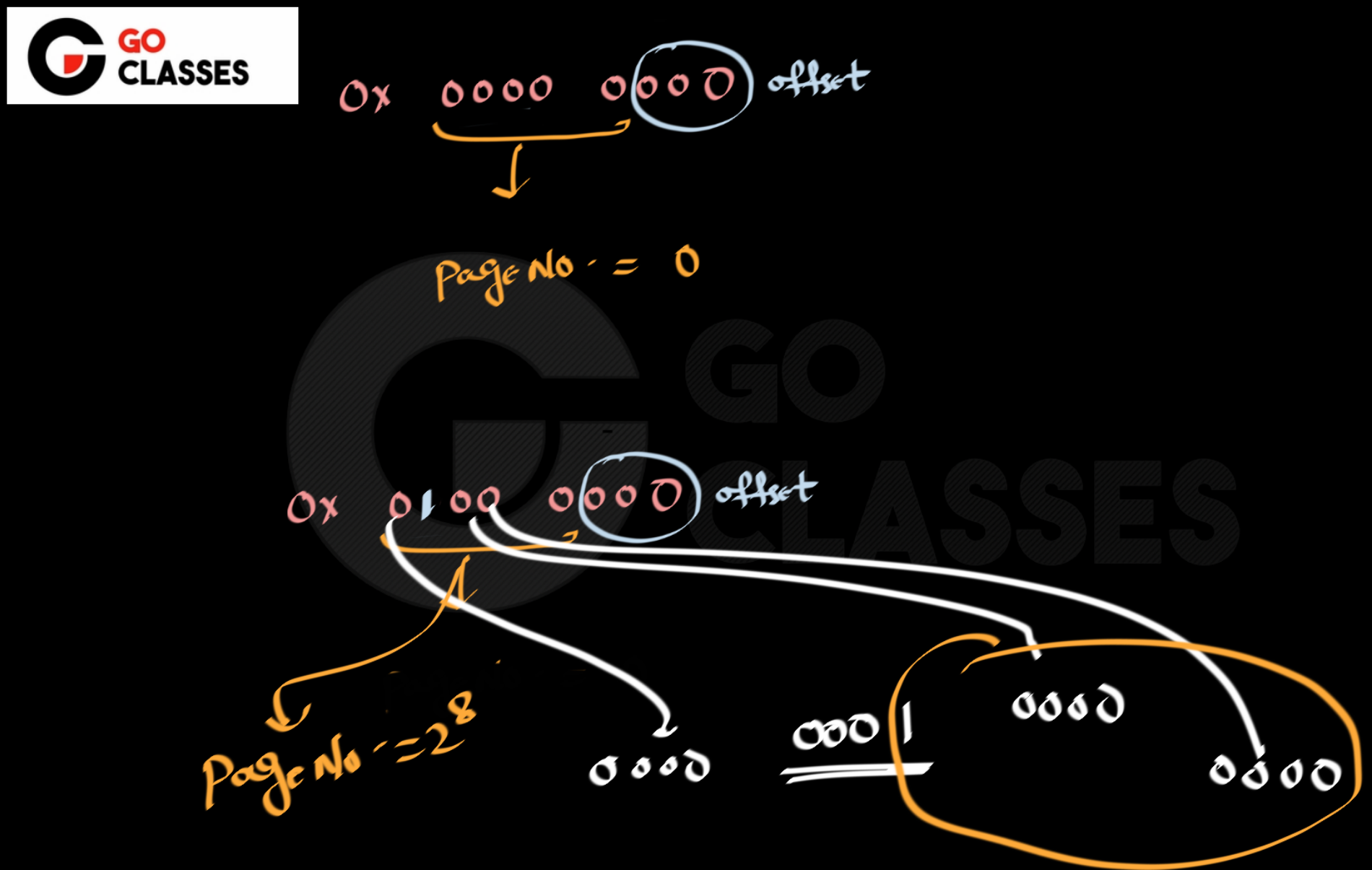
stack : 1 $0x00400000$, ⇒ 1

$0xFFFFF000$ ⇒ $2^{10}-1$

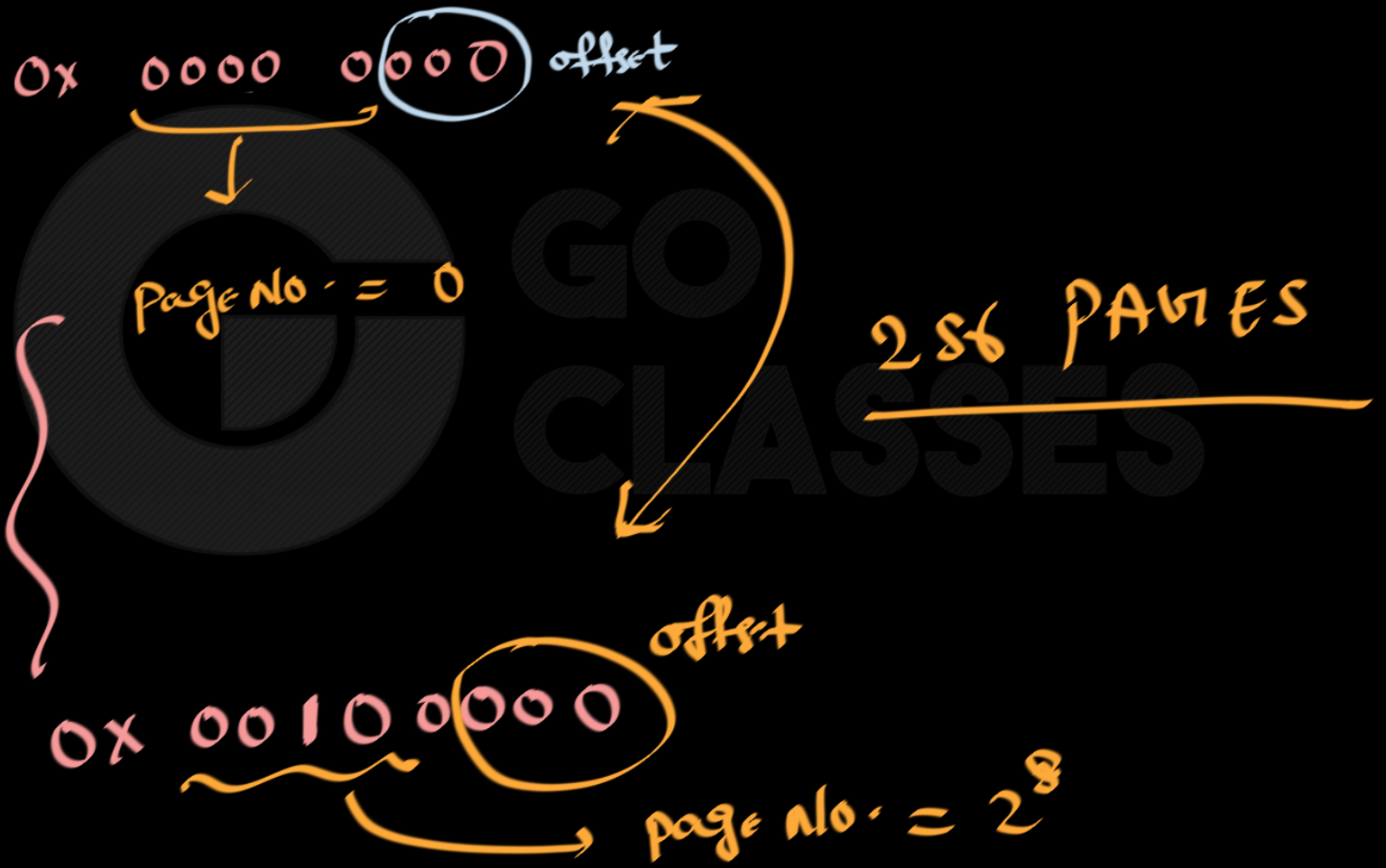
ENTRY numbers of outermost page table

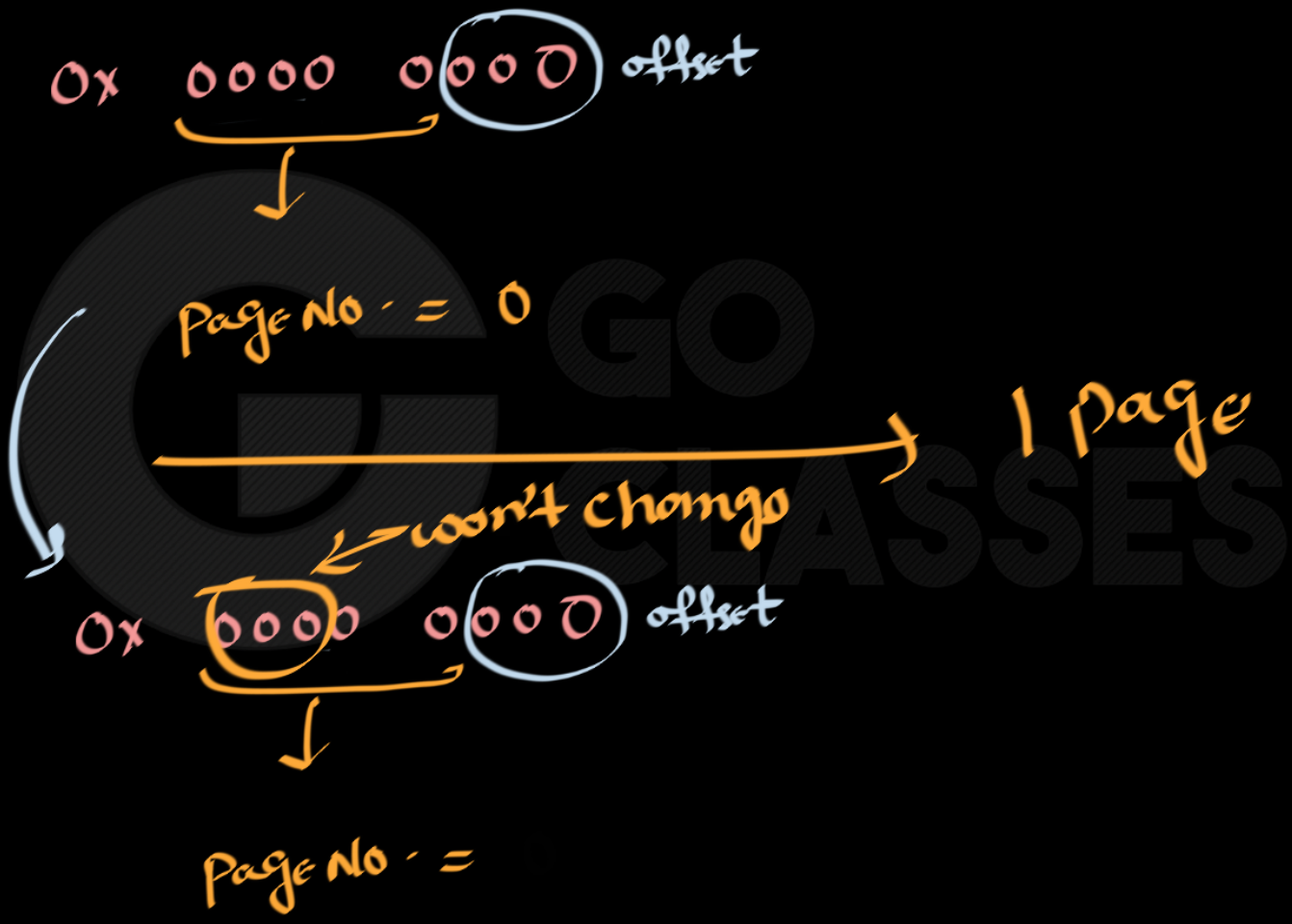
0x 0000 0000 offset

Address of 2nd page (or page 1) : - 0x 0000 1000

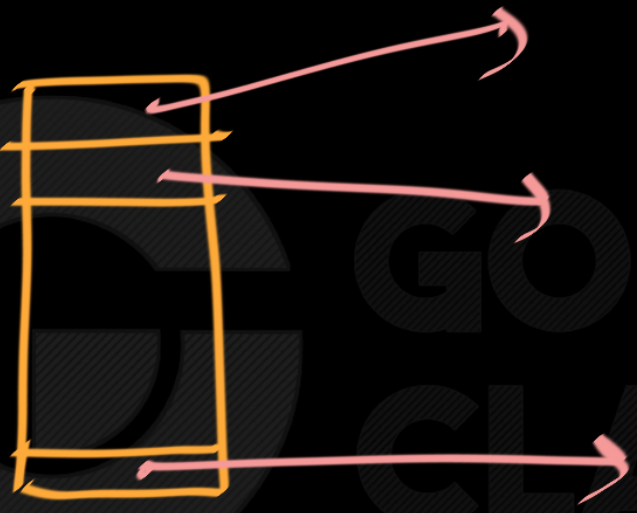

GO CLASSES
0x 0000 0000 offset
Page No. = 0
0x 0100 0000 offset
Page No. = 2^8
0000
0001
0000
0000


0x 0001 0000 offset
Page No. = 1
0x 0000 0000 offset
Page No. = 0


0x 0000 0000
offset
Page No. = 0
256 PAGES
0x 0010 0000
offset
page no. = 2^8

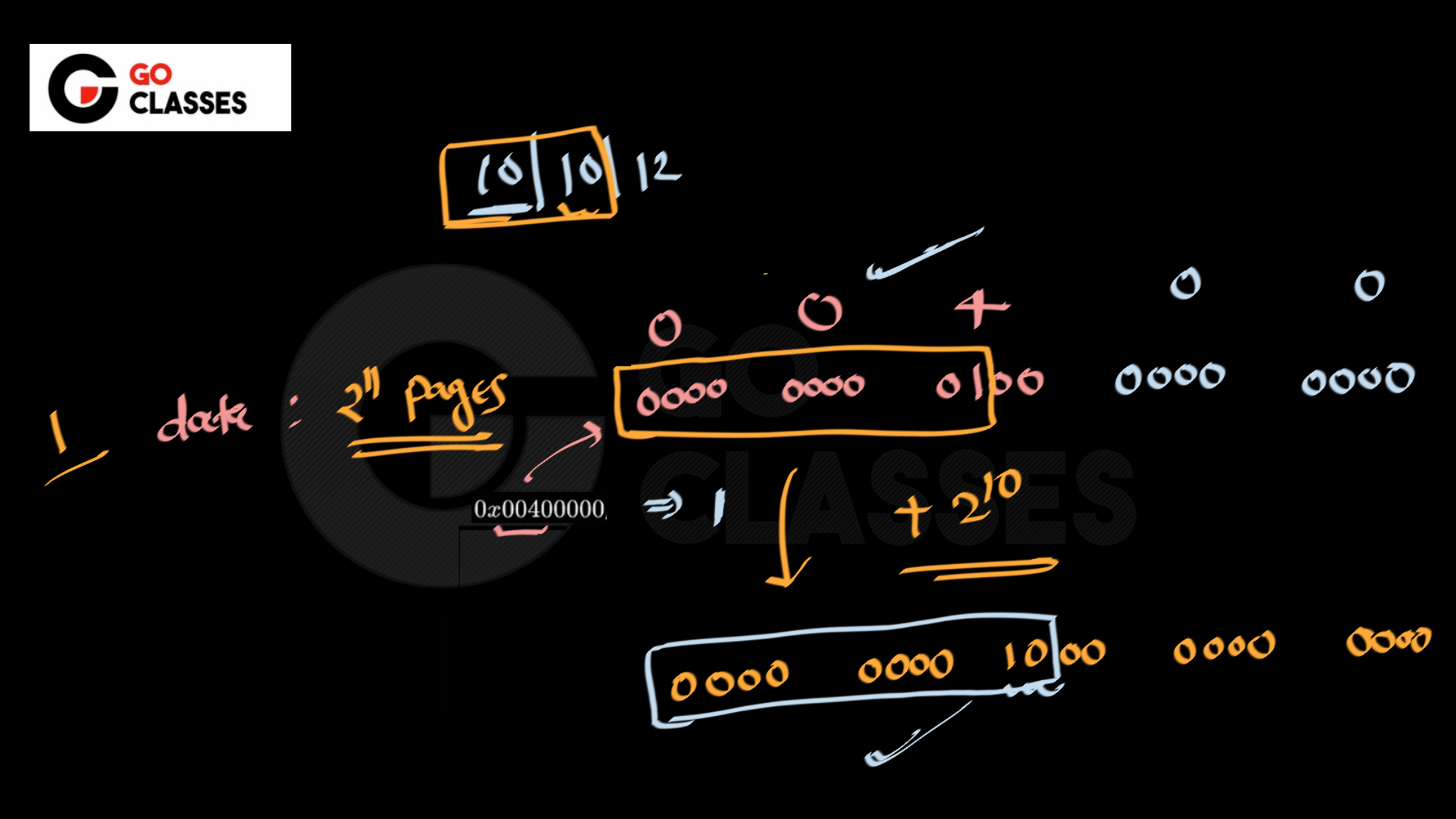

0x 0000 0000 offset
Page No. = 0
1 Page
won't change
0x 0000 0000 offset
Page No. =


10/10/12
0 code : 2 {0x00000000, ⇒ 0
1 data : 2
0000 0000 0100
1023 stack : 1 0x00400000, ⇒ 1
entry number :)
0xFFFFF000 ⇒ 2^16 - 1



outermost

we only need to store 3 chunks here.


GO CLASSES
10 10 12
1 data : 2^11 pages
0x00400000
0 0 4 0 0
0000 0000 0100 0000 0000
⇒ 1 + 2^10
0000 0000 1000 0000 0000

89

Best answer

First level page table is addressed using 10 bits and hence contains 2^{10} entries. Each entry is 4 bytes and hence this table requires 4 KB. Now, the process uses only 3 unique entries from this 1024 possible entries (two code pages starting from $0x00000000$ and two data pages starting from $0x00400000$ have same first 10 bits). Hence, there are only 3 second level page tables. Each of these second level page tables are also addressed using 10 bits and hence of size 4 KB. So,

```
total page table size of the process
= 4 KB + 3 * 4 KB
= 16 KB
```

Correct Answer: C

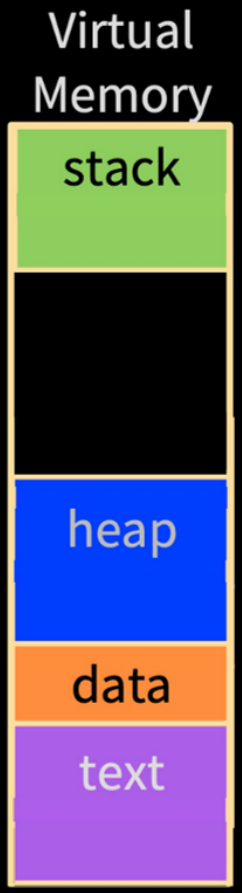
answered Apr 29, 2016 • edited May 22, 2019 by **Naveen Kumar 3**

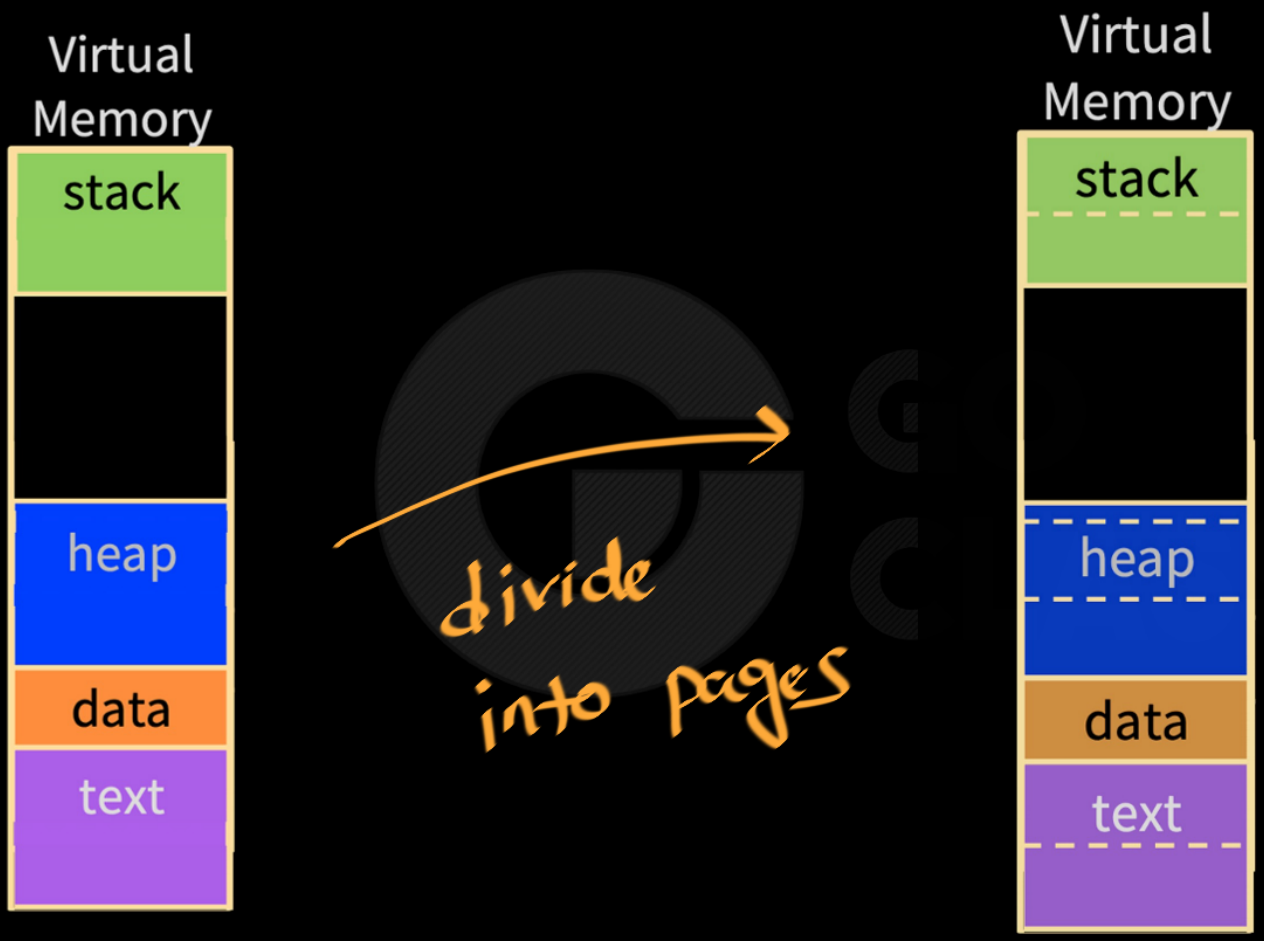
edit flag hide comment Unfollow Pip Box

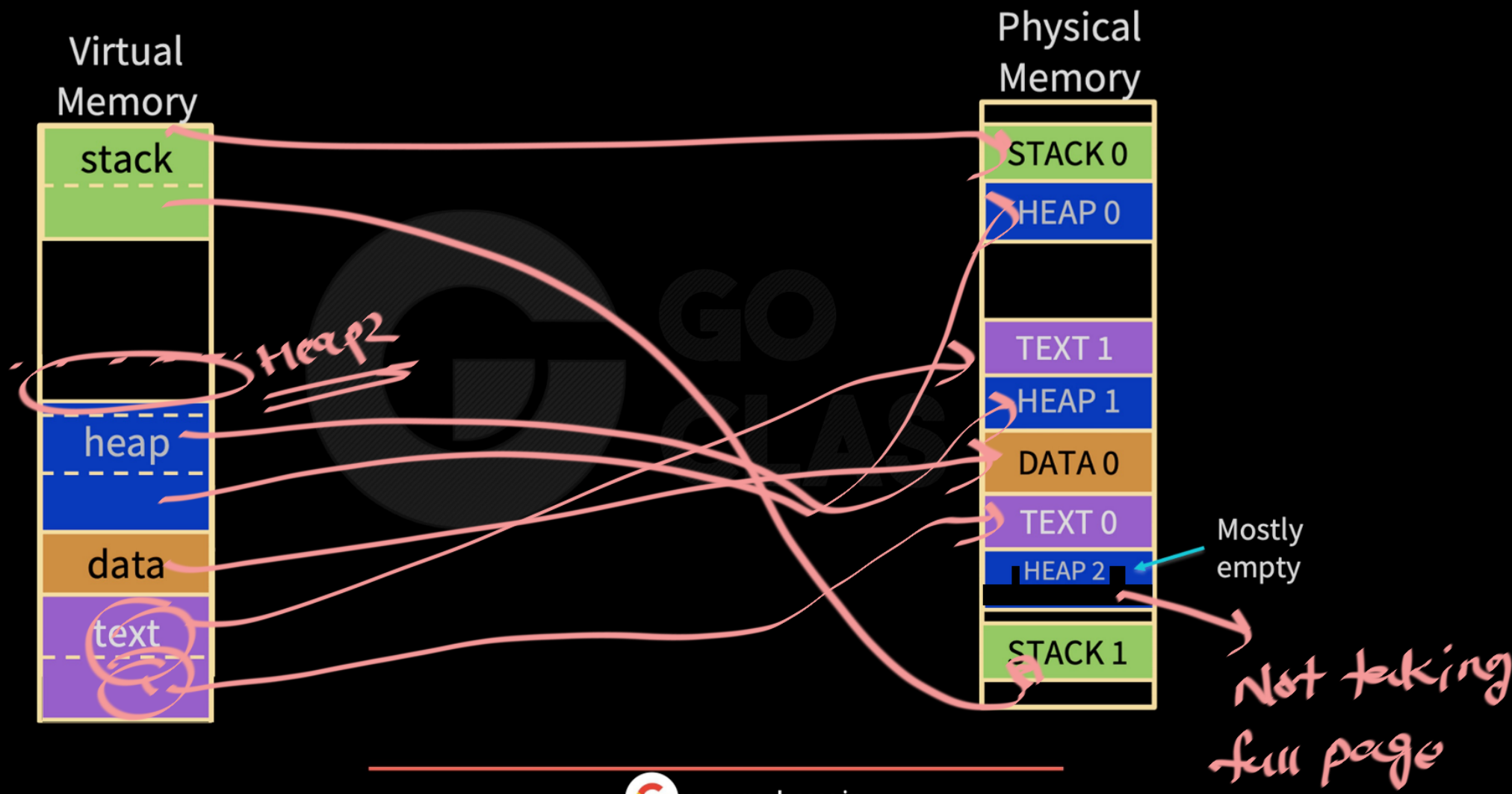
Delete with Reason Wrong Useful

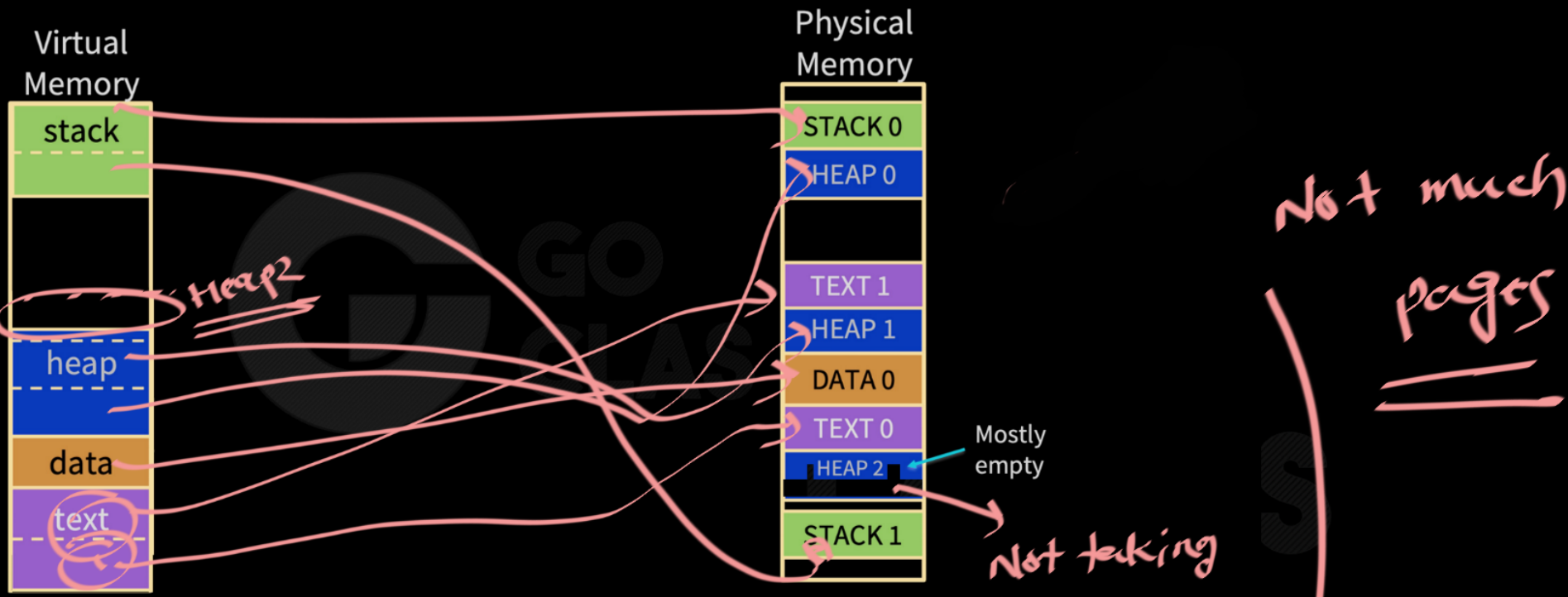
share this

Arjun


Virtual
Memory
stack
heap
data
text


Virtual Memory
stack
heap
data
text
divide into pages
Virtual Memory
stack
heap
data
text


Virtual Memory
stack
heap
data
text
Heap2
Physical Memory
STACK 0
HEAP 0
TEXT 1
HEAP 1
DATA 0
TEXT 0
HEAP 2
STACK 1
Mostly empty
Not taking full page



this is called "internal fragementation" (within the page there is nothing)

- Advantages
  - No external fragmentation
  - Fast to allocate (no searching for space) and free (no coalescing) → compacting
- Disadvantages
  - Additional memory reference to page table (hint: use a cache)
  - Internal fragmentation (tension regarding page size)
  - Required space for page table may be substantial (use multilevel paging)

https://hexhive.epfl.ch/OSTEP-slides/14-paging.pdf

Hashed Page Table:

↳ used as an alternative to multi level paging (reducing the size of page table)

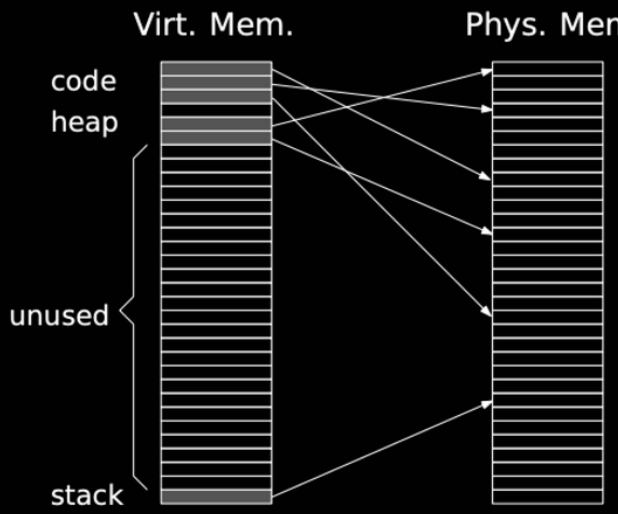
GO CLASSES

idea is to reduce the size of page table but without using multilevel paging.

GO CLASSES

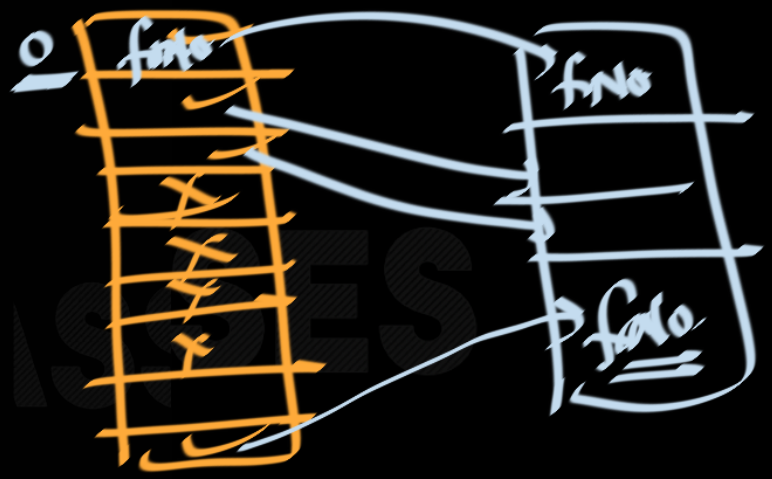
Why are page tables so large?

Answer Page tables are full of invalid PTEs



Page Table

PFN	valid	prot
8	1	r-x
3	1	r-x
18	1	r-x
-	0	-
0	1	rw-
12	1	rw-
-	0	-
-	0	-
many more invalid PTEs		
-	0	-
-	0	-
24	1	rw

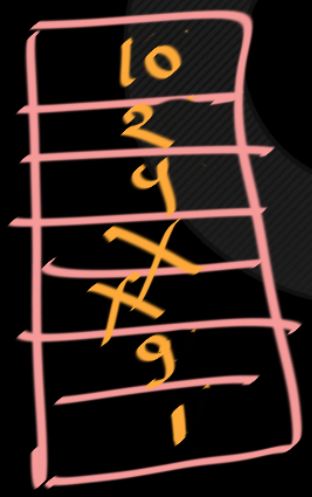


Question How to avoid storing invalid PTE?

https://web.fe.up.pt/~pfs/aulas/so2021/at/24vm_tables.pdf

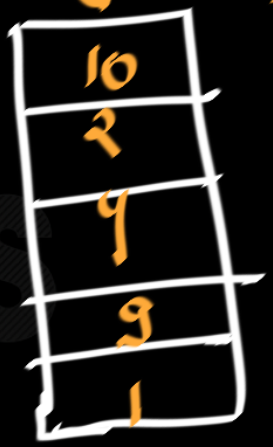
GO CLASSES

Store only valid PTEs



Page table (suppose only 5 valid)

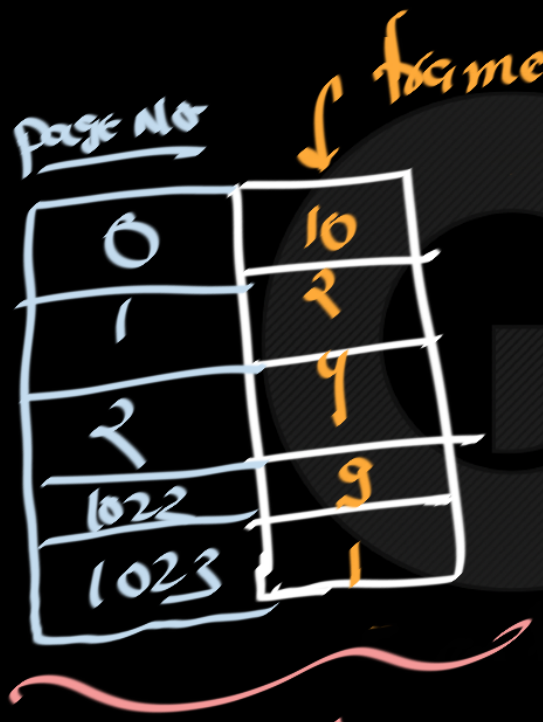
frame no's



5 entries only

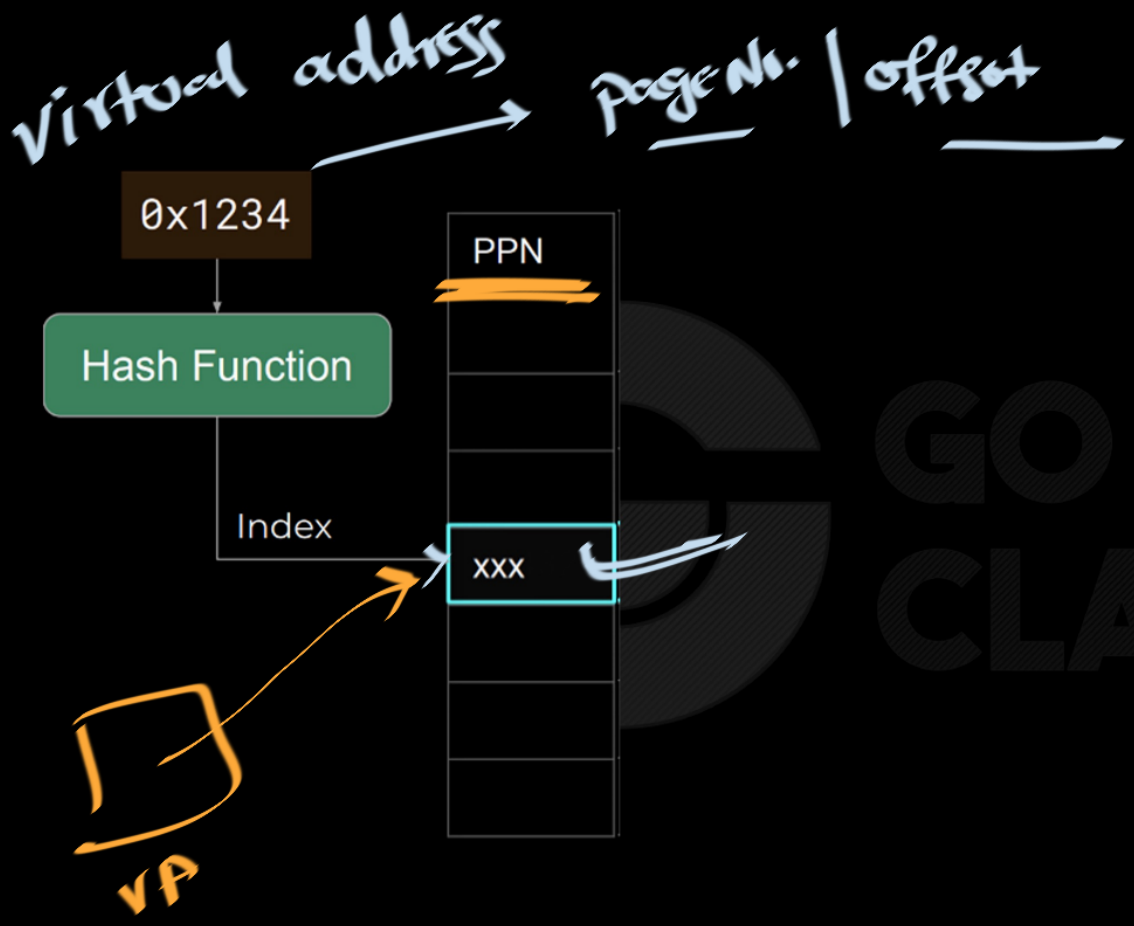
Store only valid PTEs

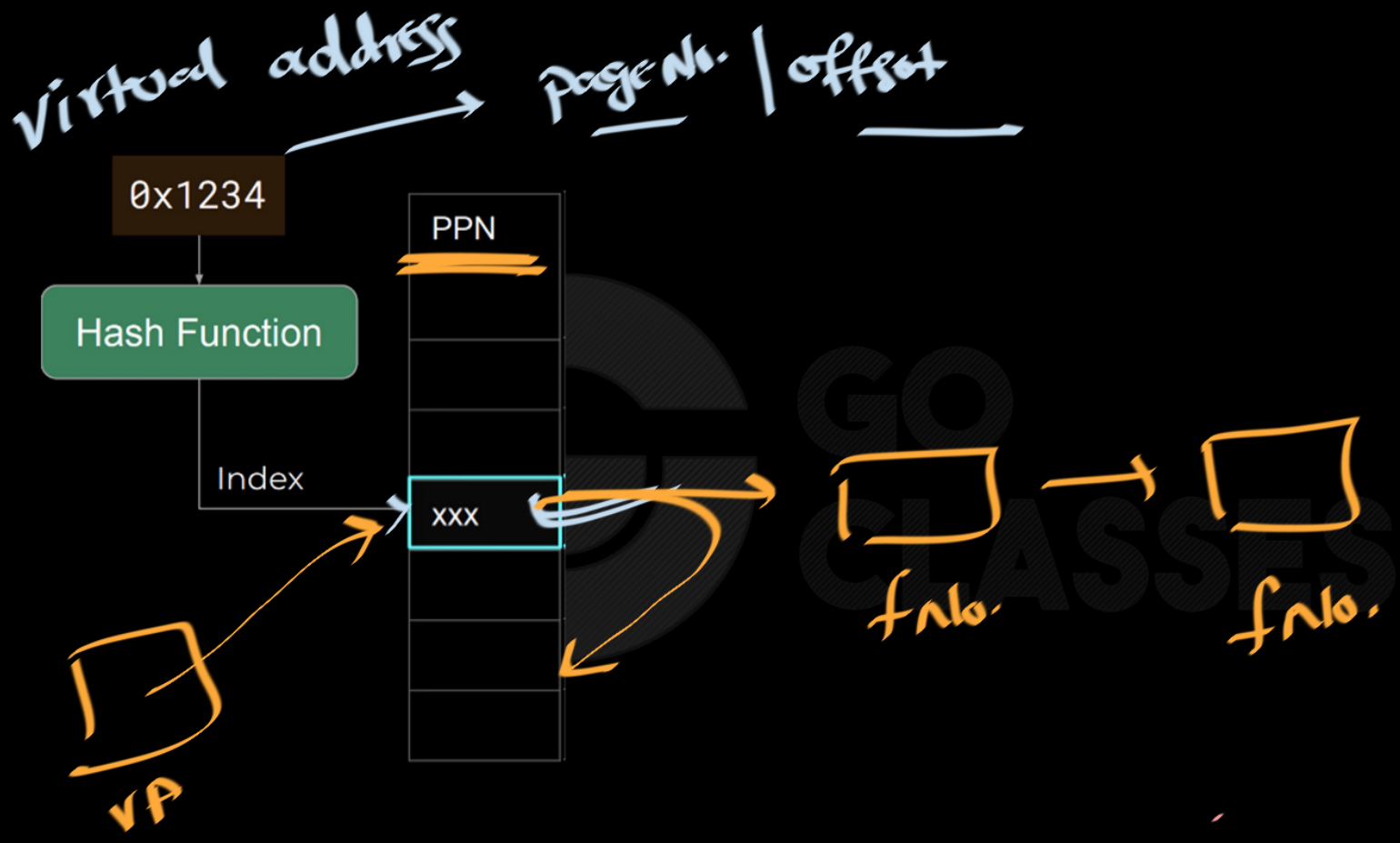
Page table (suppose only 5 valid) 5 entries only



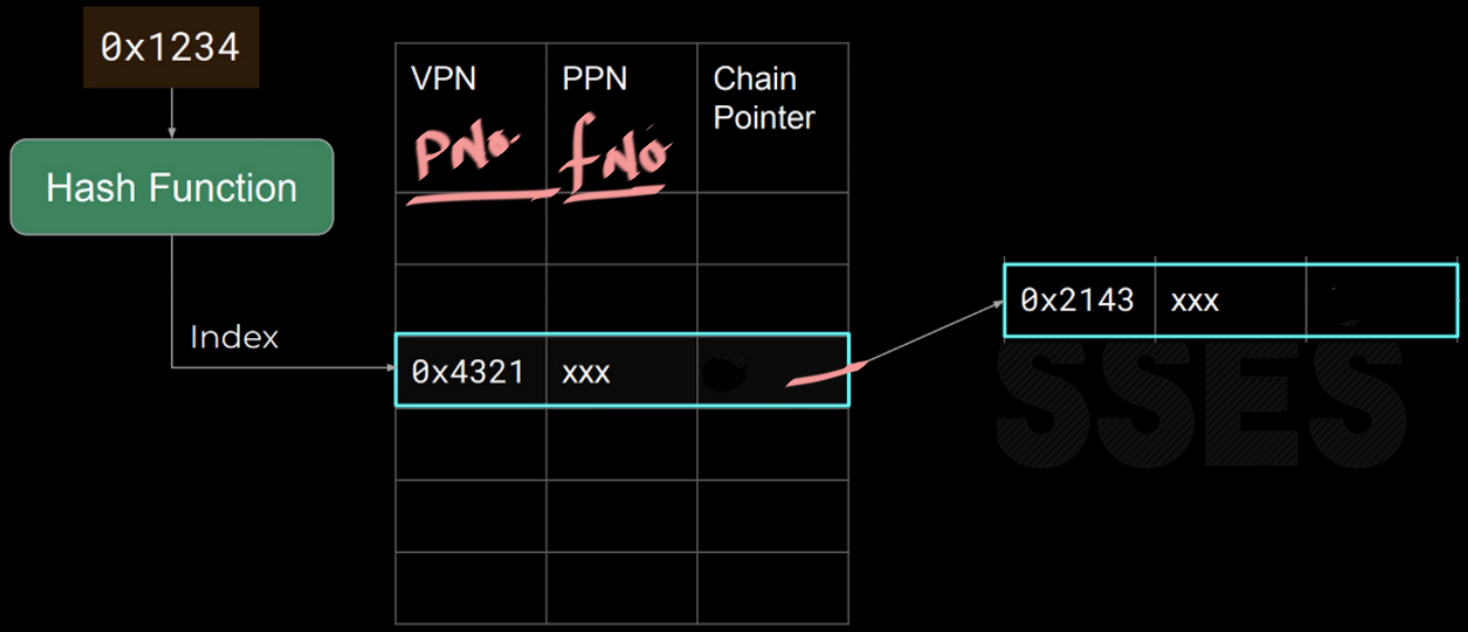
it will take lot of time to search

Soln is hashing


Virtual address
Page No. | offset
0x1234
Hash Function
Index
PPN
xxx
VA


Virtual address
Page No. | offset
0x1234
Hash Function
Index
PPN
xxx
frame
frame
VA

- Need collision resolution
  - One possible solution is to use a chain table



Many collisions means we have to traverse a long linked list

hash value being the virtual page number.

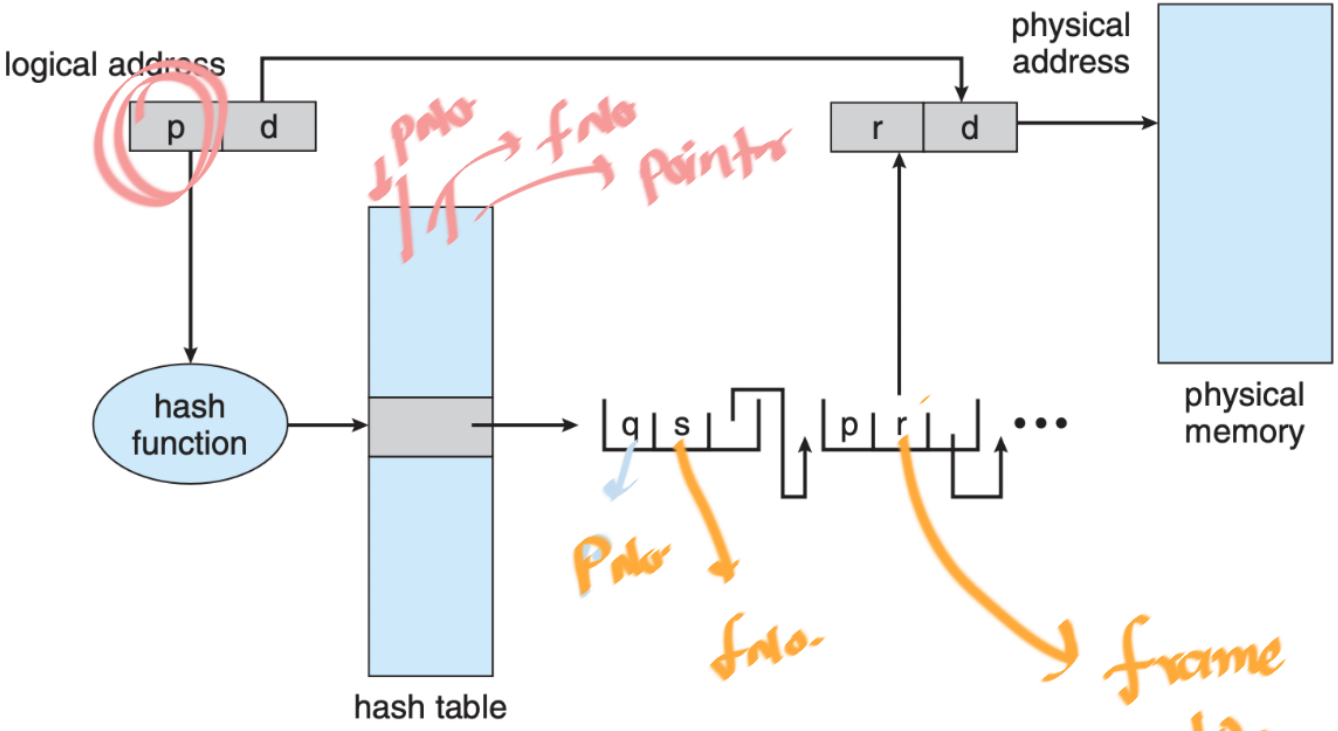


Figure 8.19 Hashed page table.

8.6.2 Hashed Page Tables

A common approach for handling address spaces larger than 32 bits is to use a **hashed page table**, with the hash value being the virtual page number. Each entry in the hash table contains a linked list of elements that hash to the same location (to handle collisions). Each element consists of three fields: (1) the virtual page number, (2) the value of the mapped page frame, and (3) a pointer to the next element in the linked list.

The algorithm works as follows: The virtual page number in the virtual address is hashed into the hash table. The virtual page number is compared with field 1 in the first element in the linked list. If there is a match, the corresponding page frame (field 2) is used to form the desired physical address. If there is no match, subsequent entries in the linked list are searched for a matching virtual page number. This scheme is shown in Figure 8.19.

GO CLASSES

Page table size is too big

used by almost all systems → multi level paging

→ Hashed page table

Now → inverted page table

Inverted Page Table:

Inverted Page Table:

frame No.

0	4
1	0
2	1
3	
4	
5	
6	
7	

Page Table for P1

0	2
1	3
2	5
3	
4	
5	
6	
7	

Page Table for P2

	pid	Page no.
0	1	1
1	1	2
2	2	0
3	2	1
4	1	0
5	2	2

Inverted Page Table

One GLOBAL Page table for ALL Processes

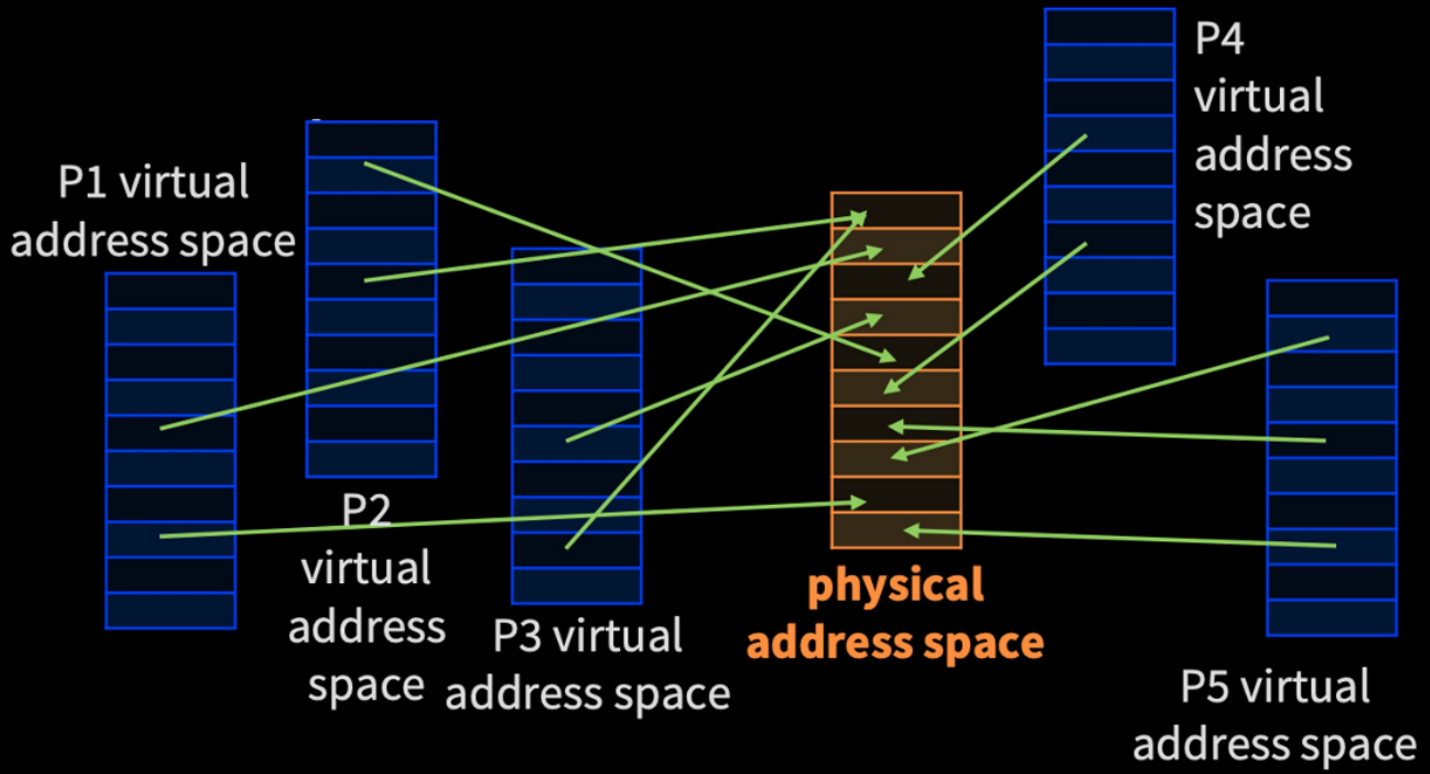
GO CLASSES

for every frame there is a row.

hence size of inveted page table =

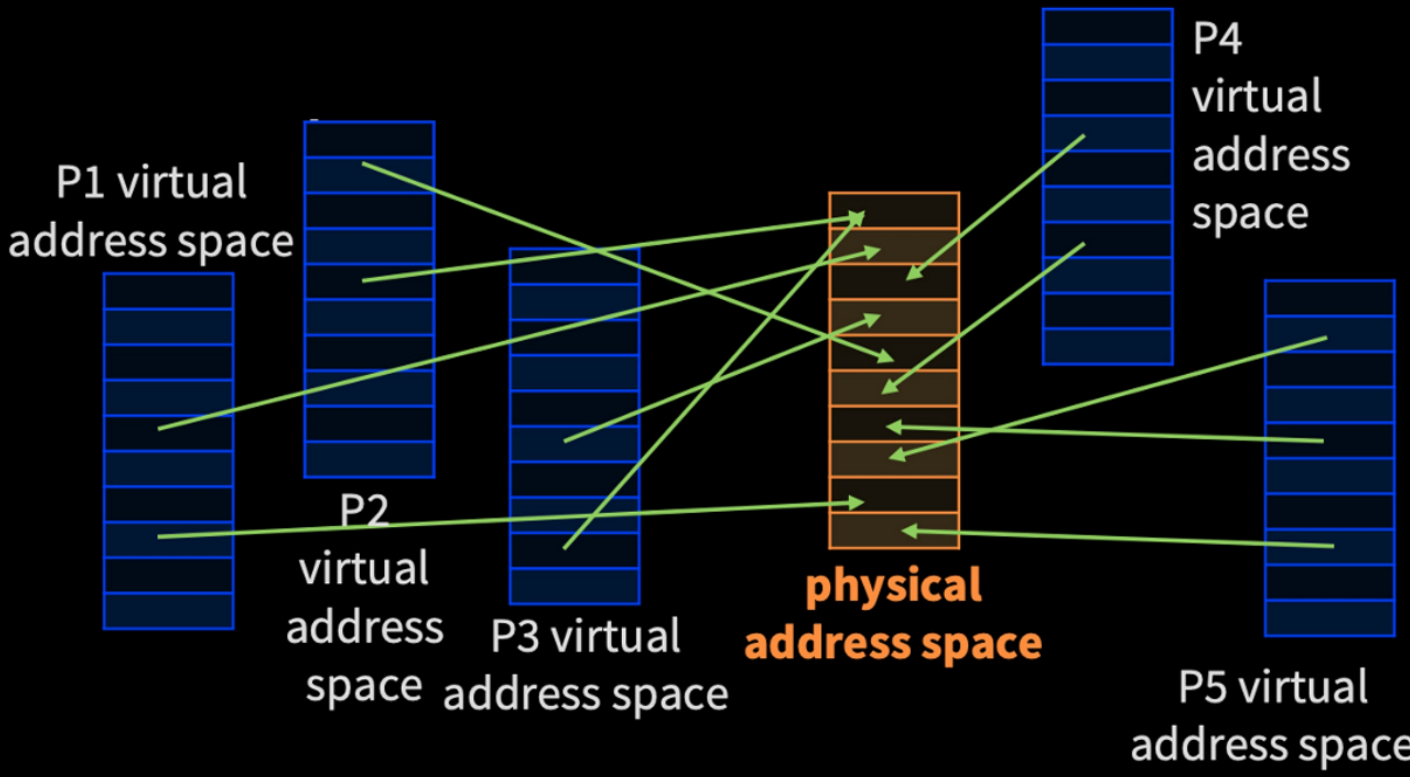
No. of frames X Intarted page table entry

So many virtual pages...



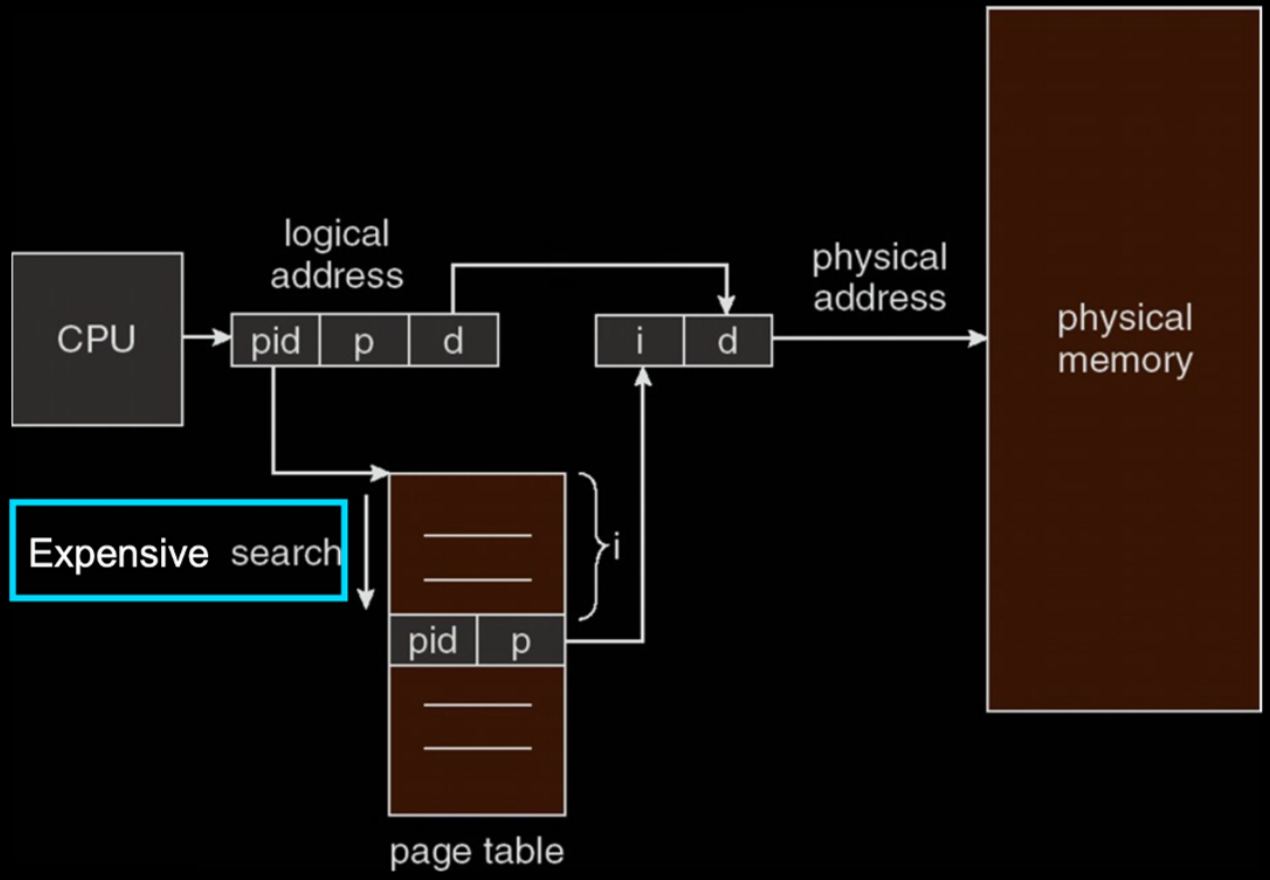
https://www.cs.cornell.edu/courses/cs4410/2019su/lectures/10-page-tables.pdf

So many virtual pages…



… comparatively few physical frames

Inverted Page Tables



Question: True/False

Problem 1a[2pts]: The size of an inverted page table grows with the size of the virtual address space that it is supporting.

https://hkn.eecs.berkeley.edu/examfiles/cs162_fa10_mt1_sol.pdf

Problem 1a[2pts]: The size of an inverted page table grows with the size of the virtual address space that it is supporting.

True / (False)

Explain: *Since an inverted page table holds a* hash *on entries, it only needs to grow with the size of the physical address space. Entries not found in the inverted page table can be assumed to be invalid.*

Question:

1. **[20 points] Memory Management/Design:**
A. [**10 points**] Assume a machine has a **16-bit virtual address** space and is byte-addressable. The **physical memory is 8K bytes (2^13)** and the **page/frame size is 256 bytes (2^8)**. Give the best answer to each of the following with a short justification. If you need to make additional assumptions to give an answer, state the assumptions that you are making *[Note: if the answer is a power of 2, leave it in exponential form.]*

   a. [0.5pt] How many bits are used for the page offset? _____

   b. [0.5pt] How many bits of virtual address are used for page number? _____

B. [10 points] For the previous problem (**1.A**), suppose you want to use an **inverted** page table:
a. [5pt] determine the size of the **inverted page table** (show/explain the number/meaning of bits)

b. [5pt] Explain why (or why not) you would use the **inverted page table** in this system instead of the **conventional page table** given in 1.A.

http://www.cs.utsa.edu/~korkmaz/teaching/resources-os-ug/tk-samples/solutions/2017a-Spring-cs3733-Midterm-II-solution-v2.pdf

1. **[20 points] Memory Management/Design:**
   A. [**10 points**] Assume a machine has a **16-bit virtual address** space and is byte-addressable. The **physical memory is 8K bytes (2^13)** and the **page/frame size is 256 bytes (2^8).** Give the best answer to each of the following with a short justification. If you need to make additional assumptions to give an answer, state the assumptions that you are making *[Note: if the answer is a power of 2, leave it in exponential form.]*

   a. [0.5pt] How many bits are used for the page offset? _____8 bits_____

   b. [0.5pt] How many bits of virtual address are used for page number? _____8 bits_____

   B. [10 points] For the previous problem **(1.A)**, suppose you want to use an **inverted** page table:
   a. [5pt] determine the size of the **inverted page table** (show/explain the number/meaning of bits)

a. Inverted page table will have 2^5 = 32 entries * 4 bytes(2bytes for pid + 8 bits for p + 8 bits for control e.g., i/v)

= 2^7 = 128 Bytes while conventional table size was 2^8= 256 Bytes for each process

b. [5pt] Explain why (or why not) you would use the **inverted page table** in this system instead of the **conventional page table** given in 1.A.

b. both are OK as long as you explain why! For example, I would prefer inverted as it significantly uses less memory! Unfortunately it requires searching, through the array but since it is a small size array, this search should be done quickly.
Conventional (1 page table per process) is OK as it is faster than inverted table but requires more memory when the number of process increases ...

Question:

17. Consider a computer system with a 32-bit logical address and 4 KB page size. The system supports up to 512 MB of physical memory. How many entries are there in:
    a. A conventional single-level page table?
    b. An inverted page table?

H.W

Solution

17. Consider a computer system with a 32-bit logical address and 4 KB page size. The system supports up to 512 MB of physical memory. How many entries are there in:
    a. A conventional single-level page table?
    b. An inverted page table?

    Answer: a. 4kb page size – 12 bits. Therefore, 20 bits for page table. Page table contains 2^{20} entries.

    b. 512 MB of physical memory requires 29 bits. Frame size = page size = 4kb. So, page offset is still 12 bits. So, number of physical frames = $2^{29-12} = 2^{17}$.

http://ittc.ku.edu/~kulkarni/teaching/EECS678/F19-prep-final-solution.pdf

Question:

Assume we make a single memory access on a processor which has no caches and no TLB. All else being equal, which of the following are true?

- ☐ Base-and-bound will on average be faster than a single-level page table.
- ☐ An inverted page table will on average be faster than a single-level page table.
- ☐ A single-level page table will on average be faster than a multi-level page table.
- ☐ A multi-level page table will on average be faster than an inverted page table.
- ☐ None of the above.

https://cs162.org/static/exams/sp21-mt2-solutions.pdf

Question:

Assume we make a single memory access on a processor which has no caches and no TLB. All else being equal, which of the following are true?

- ☐ Base-and-bound will on average be faster than a single-level page table.
- ☐ An inverted page table will on average be faster than a single-level page table.
- ☐ A single-level page table will on average be faster than a multi-level page table.
- ☐ A multi-level page table will on average be faster than an inverted page table.
- ☐ None of the above.

one extra memory access

https://cs162.org/static/exams/sp21-mt2-solutions.pdf

Assume we make a single memory access on a processor which has no caches and no TLB. All else being equal, which of the following are true?

- [x] Base-and-bound will on average be faster than a single-level page table.
- [ ] An inverted page table will on average be faster than a single-level page table.
- [x] A single-level page table will on average be faster than a multi-level page table.
- [ ] A multi-level page table will on average be faster than an inverted page table.
- [ ] None of the above.

Searching is also required

To translate the virtual address to a physical address, base-and-bound would require 0 memory accesses (only arithmetic operations), a single-level page table and an inverted page table would both require 1 memory access (to access the page table), and a multi-level page table would require multiple memory accesses, depending on the number of page tables to traverse.

Inverted Page Table

- One global page table
  - One page table entry per physical page.
  - Entries keyed by PID and virtual page number.
  - Physical frame number = index in page table.
- Advantages
  - Bounded amount of memory for page table(s).
    - 32-bit address space, 4K pages, 4GB RAM, 4B per PTE
    - 1-level page table: _____ x # processes.
    - Inverted Table: _____ MB
- Disadvantages
  - Costly translation
    - Using hashing can help

→ Hashed Inverted Page table

What's done in practice

- Some systems have used inverted page tables (e.g., IBM RS/6000, PowerPC), but hierarchical page tables seem to dominate at this point

http://courses.ics.hawaii.edu/ReviewICS332/morea/090_VirtualMemory/ics332_virtualmemory1.pdf

Inverted Page Table: Discussion

Tradeoffs:

↓ memory to store page tables

↑ time to search page tables

Solution: hashing → hashed inverted page table

- hash(page,pid) → PT entry (or chain of entries)

GATE CSE 2022 | Question: 28

14

Which one of the following statements is FALSE?

A. The TLB performs an associative search in parallel on all its valid entries using page number of incoming virtual address.

B. If the virtual address of a word given by CPU has a TLB hit, but the subsequent search for the word results in a cache miss, then the word will always be present in the main memory.

C. The memory access time using a given inverted page table is always same for all incoming virtual addresses.

D. In a system that uses hashed page tables, if two distinct virtual addresses V1 and V2 map to the same value while hashing, then the memory access time of these addresses will not be the same.

gatecse-2022 | operating-system | memory-management | translation-lookaside-buffer | 2-marks

https://gateoverflow.in/371908/gate-cse-2022-question-28

GATE CSE 2022 | Question: 28

14

Which one of the following statements is FALSE?

C. The memory access time using a given inverted page table is always same for all incoming virtual addresses.

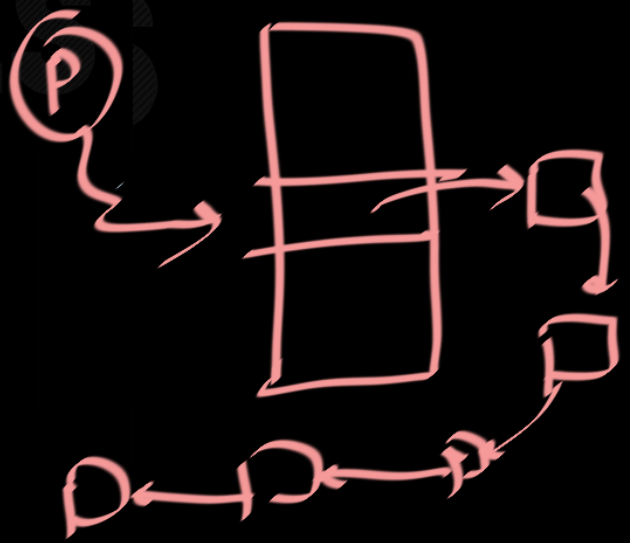
D. In a system that uses hashed page tables, if two distinct virtual addresses V1 and V2 map to the same value while hashing, then the memory access time of these addresses will not be the same.

gatecse-2022 | operating-system | memory-management | translation-lookaside-buffer | 2-marks

false

false

T



https://gateoverflow.in/371908/gate-cse-2022-question-28