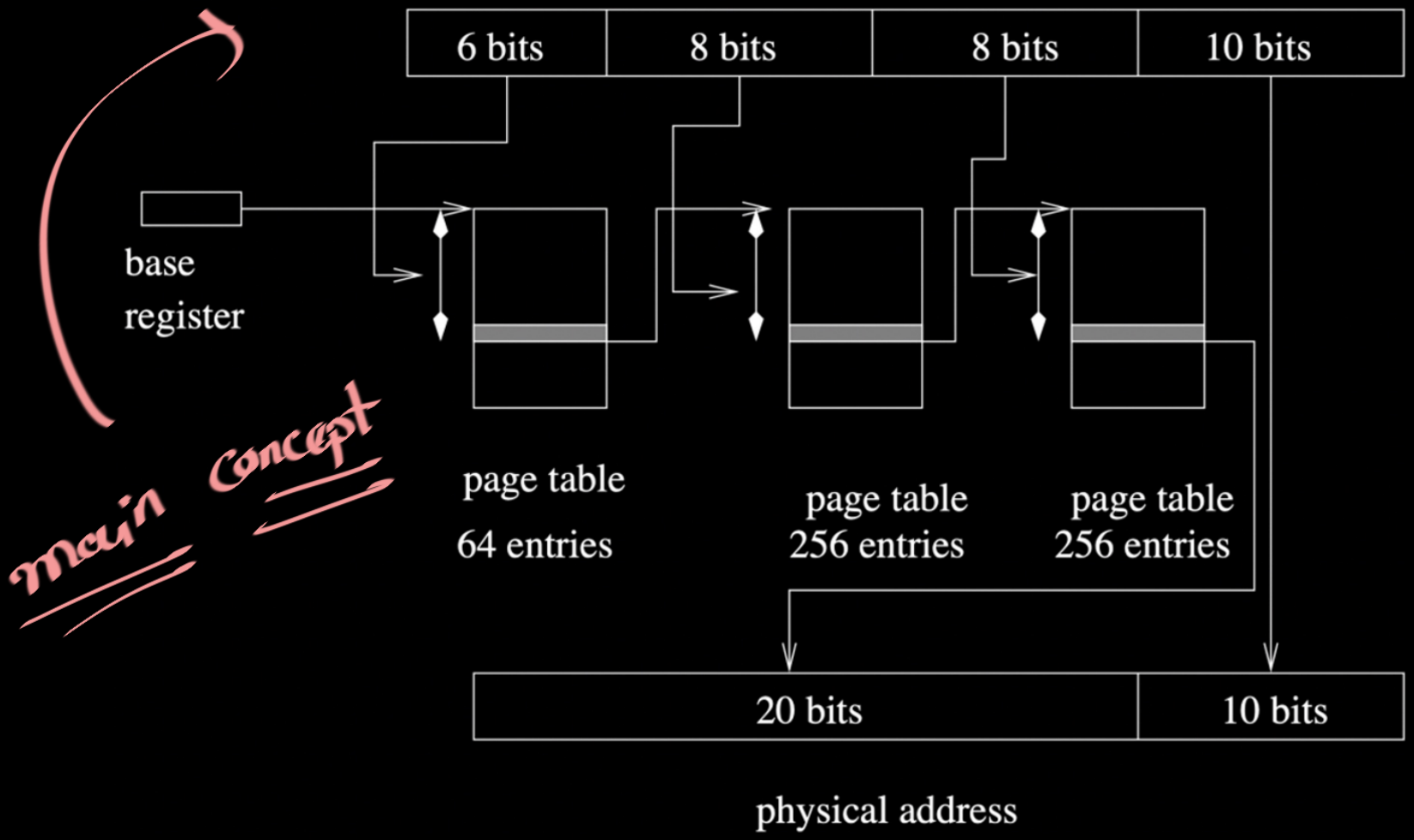
Lecture: 28

6 bits	8 bits	8 bits	10 bits

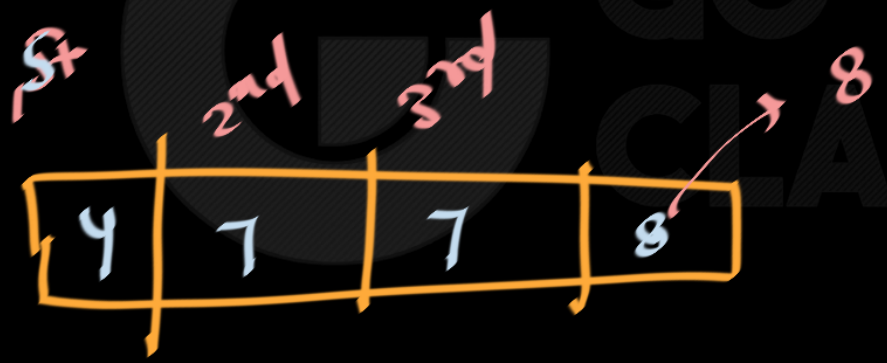

6 bits
8 bits
8 bits
10 bits
base
register
page table
64 entries
page table
256 entries
page table
256 entries
20 bits
10 bits
physical address
Main Concept

Q:

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 ?



$4 + 7 + 7 + 8 = 36$

Q:

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 Size = 2^6 B
- second level page tables Size = 2^7 B
- third level page tables Size = 2^8 B

What is size of VAS ?

Q:

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 Size = 2^6 B
- second level page tables Size = 2^7 B
- third level page tables Size = 2^8 B

What is size of VAS ?

$\frac{2^6 B}{2B} = 2^5$ entries

$\frac{2^7 B}{2B} = 2^6$ entries

$\frac{2^8 B}{2B} = 2^7$ entries

| 6 | 7 | 8 | 8 |

wrong

Correct

| VAS | 5 | 6 | 7 | 8 |

= 26 bits

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

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.

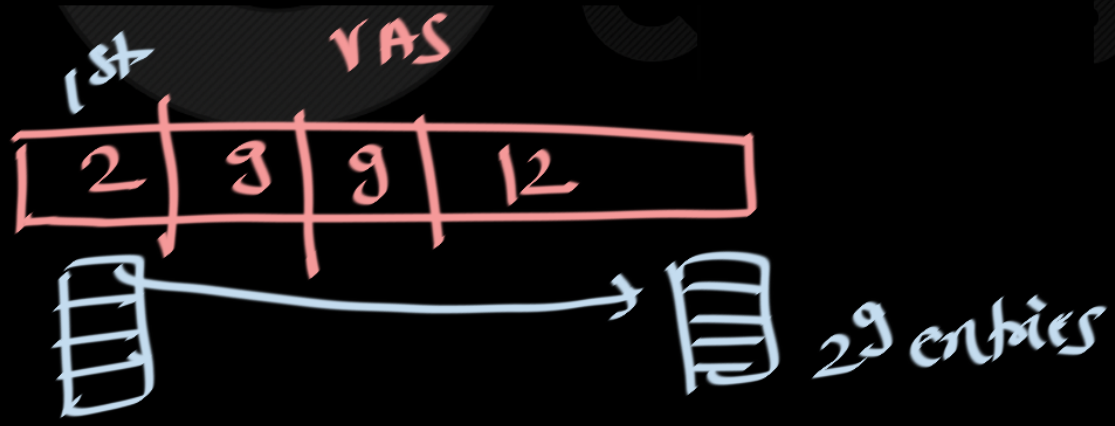
The number of bits required for addressing the next level page table(or page frame) in the page table entry of the first, second and third level page tables are respectively

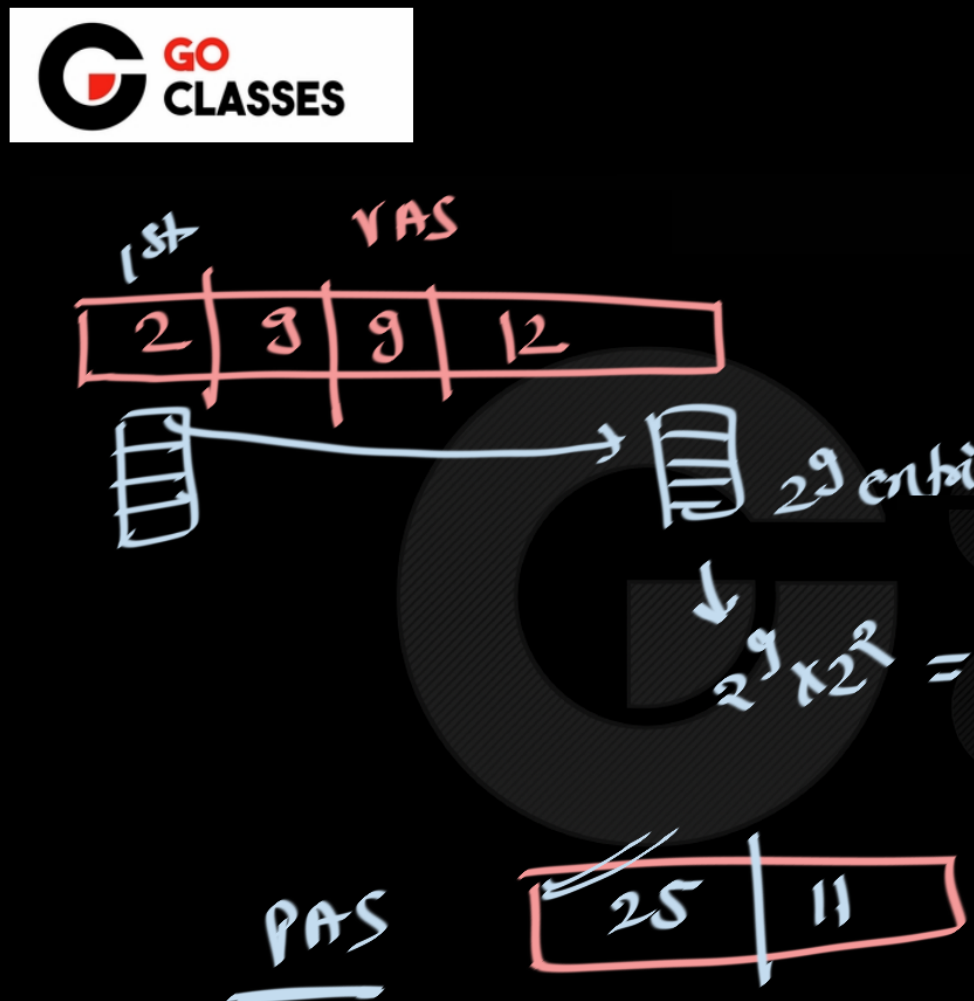
A. $20,20,20$
B. $24,24,24$
C. $24,24,20$
D. $25,25,24$

gatecse-2008 | operating-system | virtual-memory | normal

Page Size = 4KB

PTE = 4 B

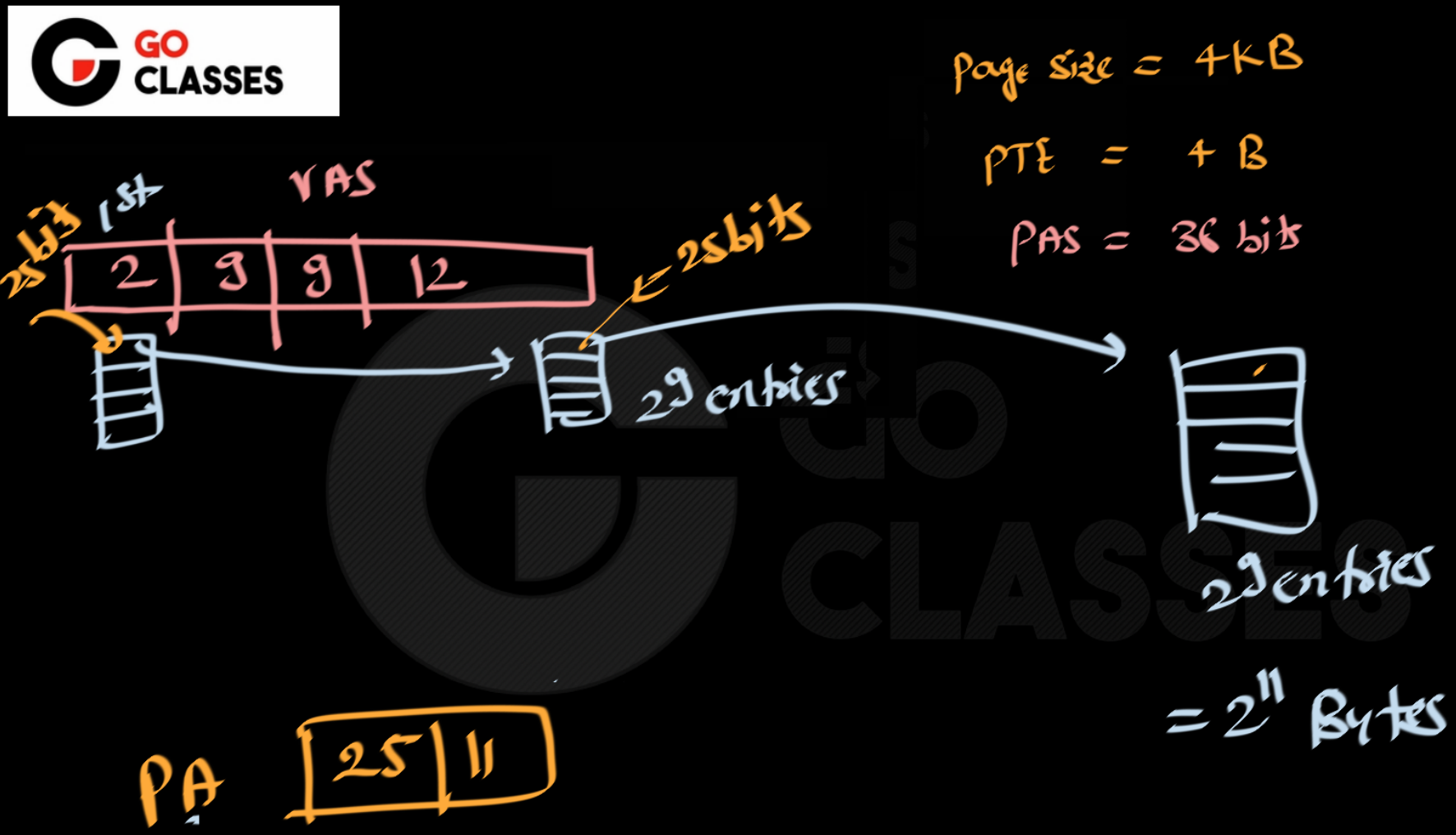


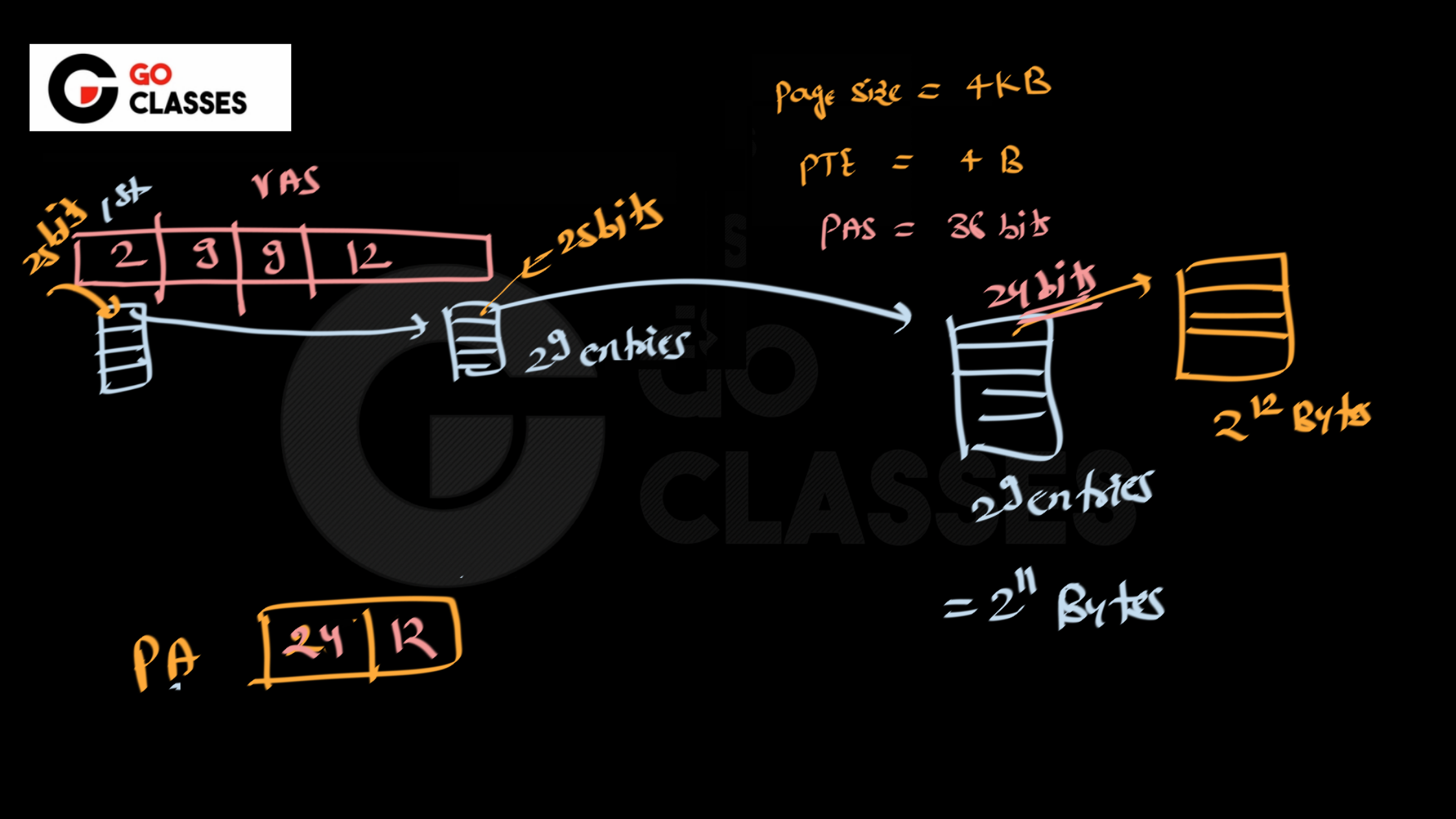


Page Size = 4KB

PTE = 4 B

PAS = 36 bits


GO CLASSES
Page size = 4KB
PTE = 4 B
PAS = 36 bits
VAS
25bits
1st
2
9
9
12
25bits
29 entries
29 entries
= 2^11 Bytes
PA
25
11


GO CLASSES
Page size = 4KB
PTE = 4B
PAS = 36 bits
VAS
25bits
1st
2
9
9
12
25bits
29 entries
24 bits
29 entries
2^{12} Bytes
$= 2^{11}$ Bytes
PA
24
12

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

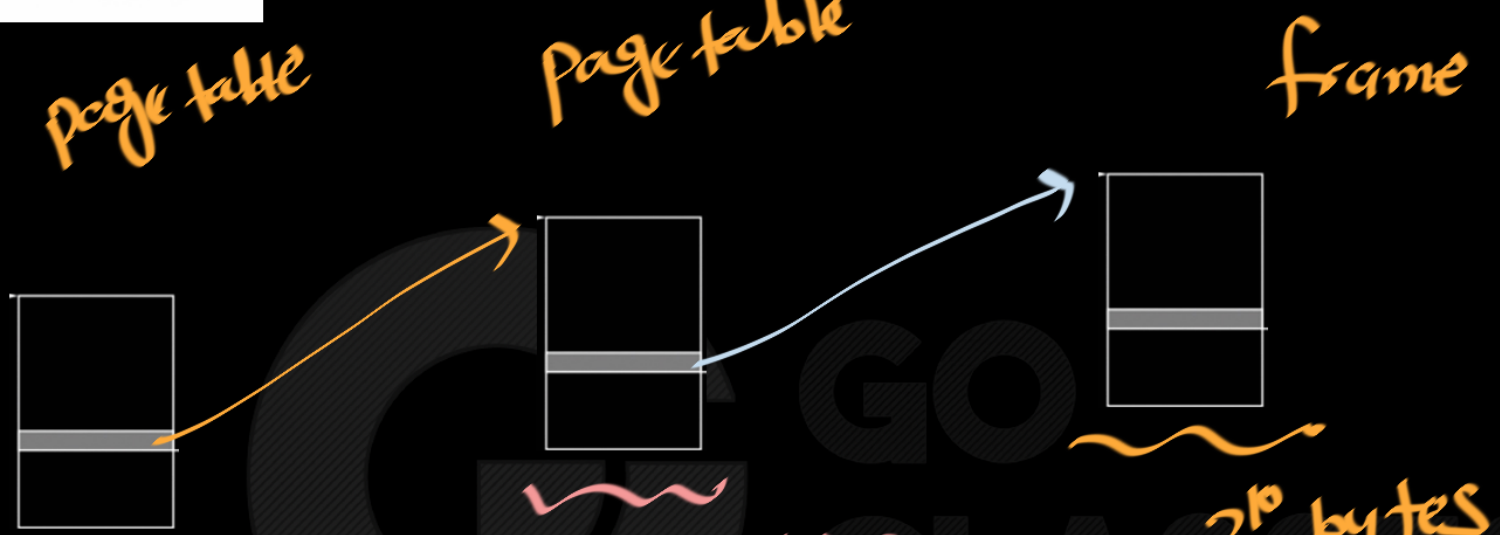
(Edit:-
There is nothing to edit for such awesome explanation but just adding one of my comment if it is useful - comment. However if anyone finds something to add (or correct) then feel free to do that in my comment.)

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

edit flag hide comment Unfollow

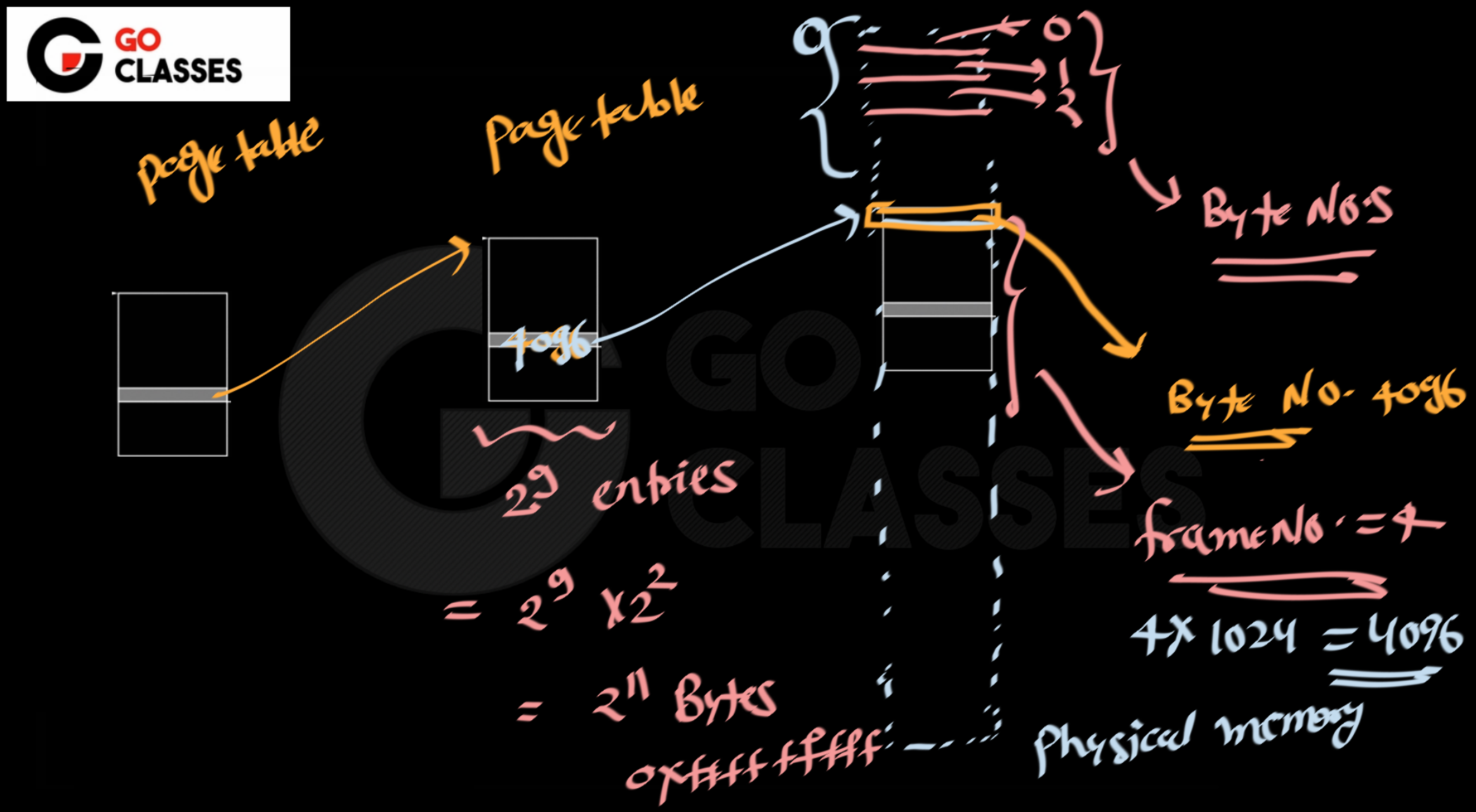
Pip Box Delete with Reason Wrong Useful

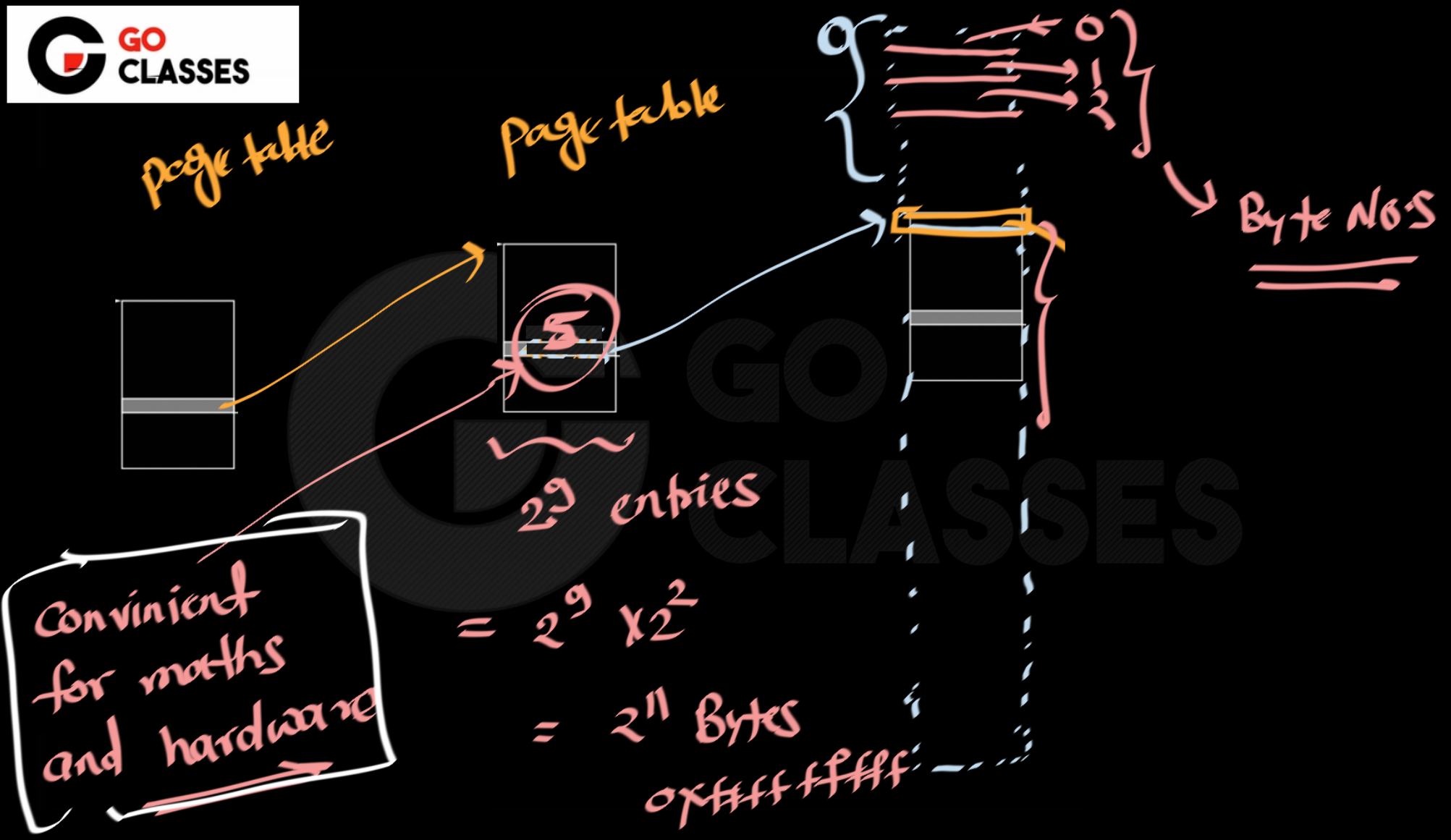
Arjun



$= 2^9 \times 2^2 = 2^{11}$ Bytes

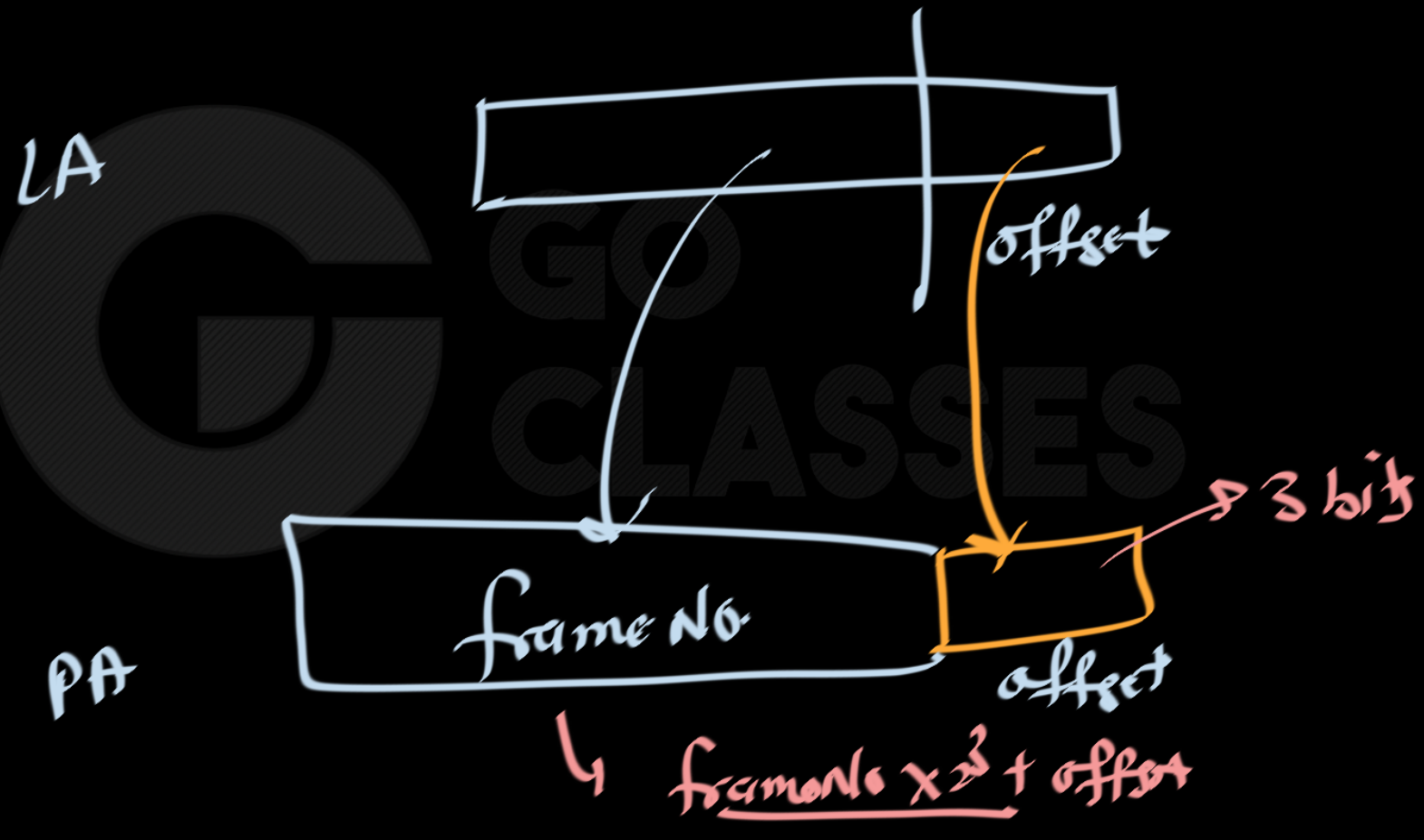
PAS = 36 bits


GO CLASSES
Page table
Page table
4096
Byte NOS
Byte No- 4096
frameNo := 4
2^9 entries
$= 2^9 \times 2^2$
$= 2^{11}$ Bytes
4 X 1024 = 4096
Physical memory


GO CLASSES
Page table
Page table
Byte Nos
2^9 entries
$= 2^9 \times 2^2$
$= 2^{11}$ Bytes
0xffffffff
Convinient for maths and hardware

GO CLASSES

Address translation in Paging



Question

Q2 (10 pts) The following table shows some parameters of a virtual memory system:

Virtual Address	Physical Address	Physical Page Size	Page Table Entry
42 bits	40 bits	64 KB	8 bytes

a) (3 pts) For a single-level page table, how many page table entries are needed? How much memory is needed for storing the page table?

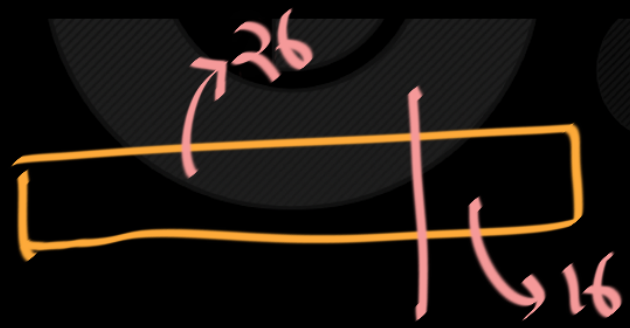
b) (4 pts) In general, the virtual address space of a process is mostly empty. To reduce the size of the page table, a multilevel page table is used for address translation. How many levels of page tables will be needed for address translation, given that the size of a page table at any level is the size of a physical page? Draw a virtual address showing the number of bits for the page table index at each level and the page offset.

Question

Q2 (10 pts) The following table shows some parameters of a virtual memory system:

Virtual Address	Physical Address	Physical Page Size	Page Table Entry
42 bits	40 bits	64 KB	8 bytes

a) (3 pts) For a single-level page table, how many page table entries are needed? How much memory is needed for storing the page table?



$2^6 \times 2^{10} = 2^{16}$ B

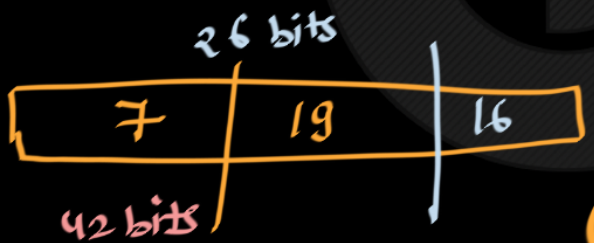
VA

42 bits

2^{26} pages $\equiv$ 2^6 page entries

Virtual Address	Physical Address	Physical Page Size	Page Table Entry
42 bits	40 bits	64 KB	8 bytes

b) (4 pts) In general, the virtual address space of a process is mostly empty. To reduce the size of the page table, a multilevel page table is used for address translation. How many levels of page tables will be needed for address translation, given that the size of a page table at any level is the size of a physical page? Draw a virtual address showing the number of bits for the page table index at each level and the page offset.



← this is NOT related to above question

Just one example

2^{19} entries should be chunk size at every level (except last)

Virtual Address	Physical Address	Physical Page Size	Page Table Entry
42 bits	40 bits	64 KB	8 bytes

b) (4 pts) In general, the virtual address space of a process is mostly empty. To reduce the size of the page table, a multilevel page table is used for address translation. How many levels of page tables will be needed for address translation, given that the size of a page table at any level is the size of a physical page? Draw a virtual address showing the number of bits for the page table index at each level and the page offset.

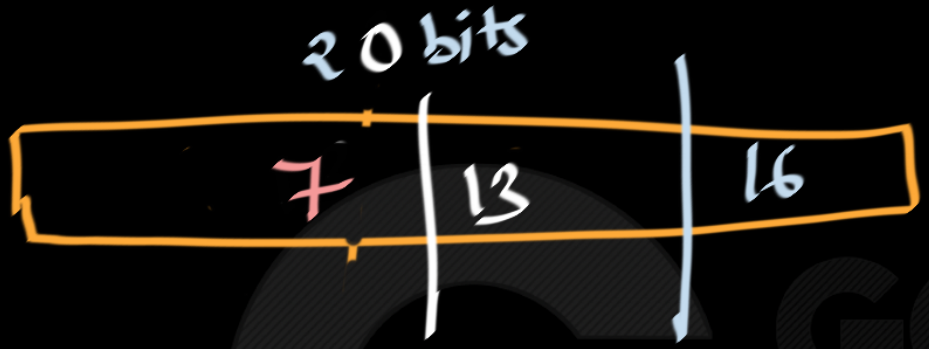
Page size = 2^{16} B

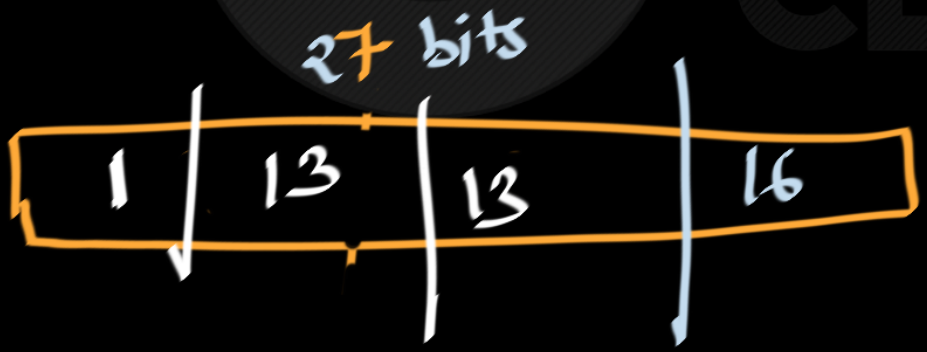
= 2^{13} entries

VAS

✓. we can partition anywhere in general.

✓ but if chunk size of some level is give then we need to follow that.

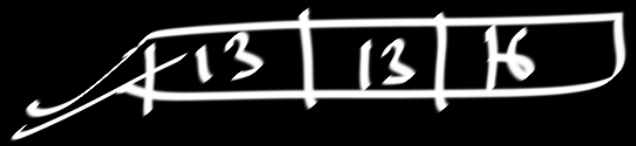

20 bits
7
13
16


27 bits
1
13
13
16

Q2 Solution:

a) Page size = 64 KB
Therefore, Page offset = 16 bits
Virtual page number = 42 bits – 16 bits = 26 bits
Number of page table entries = 2^{26}
Page Table Size = 2^{26} × 8 bytes = 2^{29} bytes = 512 MB

b) Size of a page table = Size of a physical page = 64 KB
Each page table entry is 8 bytes
Therefore, number of entries per page table = 2^{16} / 8 = 2^{13} entries
13 bits are needed to index a page table at any level
Virtual page number = 26 bits



Question

4. (27 points total) Two-Level Page-based virtual Addressing. Consider a 32-bit machine with a multi-level virtual memory system with 32-bit pointers and 4096-byte pages that supports two-levels of page tables. All Page Table Entries (PTEs) are 4 bytes.

   a. (6 points) Show the complete format of a virtual address.

   b. (6 points) Explain the steps the hardware takes in translating a virtual address to a physical address for this scheme (do not worry about supporting paging to disk).

   c. (3 points) How many memory operations are required to read or write a single 32-bit word?

   d. (4 points) List the fields of a Page Table Entry (PTE).

   e. (6 points) How much physical memory is needed for a process with one page of virtual memory?

   f. (2 points) What happens in the virtual memory subsystem on a context switch?

https://hkn.eecs.berkeley.edu/examfiles/cs162_sp03_mt1_sol.pdf

Answer

4. (27 points total) Two-Level Page-based Virtual Addressing. Consider a 32-bit machine with a multi-level virtual memory system with 32-bit pointers and 4096-byte pages that supports two-levels of page tables. All Page Table Entries (PTEs) are 4 bytes.

a. (6 points) Show the *complete* format of a virtual address.

Each address is broken up into three parts:

Page Level 1 – 10 bits	*Page Level 2 – 10 bits*	*Offset – 12 bits*

1 point for each field, 1 point for correct bit length of each field.

This is a multi-level scheme using two sets of page tables for ***each*** *process: an "outer" one and an "inner" one. PL 1 is an index into the outer page table, containing PTE's that contain pointers to the inner page table (which can be paged). In other words, the inner page table is broken up into pages that do not have to be stored contiguously.*

The PTE at outer [PL 1] contains a pointer to the PL 1^{th} page in the inner page table. PL 2 is an index into the PL 1^{th} page in the inner page table. PTE's are 4 bytes, so there are 1,024 PTE's per page. Thus, PL 2 must be exactly 10 bits. The index at PL 2 points to the page that holds virtual addresses starting with PL 1 PL 2. The particular byte is the d^{th} byte in that page. Since pages are 4K bytes, d must be exactly 12 bits. Since d is 12 bits and PL 2 is 10 bits, PL 1 must be 10 bits.

b. (6 points) Explain the steps the hardware takes in translating a virtual address to a physical address for this scheme (do not worry about supporting paging to disk).

See Problem 3a for the picture of the steps that are taken. The PTBR is used to locate the outer page table ***in physical memory****. PL 1 is used to index into the outer page table. As above, the PTE at this index points to a page in the inner page table. PL 2 is used to index in this inner page table, and the PTE in the inner page table points to the physical page. Offset is a location on the physical page. If you didn't mention the PTBR, we subtracted one point.*

Note that each PTE should have a valid bit and several protection bits – e.g., read, write, execute, valid. The memory operation (load, store, load for execution) must agree with these bits in both sets of PTE's (inner and outer). Otherwise the hardware will generate an interrupt. If you didn't mention the role of the valid bit or the protection bits, we subtracted one point for each missing role.

If you missed a level, we subtracted one point for each level missed. If you included an incorrect step, we subtracted one point.

c. (3 points) How many memory operations are required to read or write a single 32-bit word?

Without extra hardware, performing a memory operation takes 3 actual memory operations: two page table lookups in addition to the desired memory operation. We did not award partial credit for this problem.

d. (4 points) List the fields of a Page Table Entry (PTE).

Each PTE will have a pointer to the proper page (two points) plus several bits – read, write, execute, (1 point for protection bits), and valid (one point). This information can all fit into 4 bytes, since if physical memory is 2^{32} bytes, then 20 bits will be needed to point to the proper page, leaving ample space (12 bits) for the information bits.

If you didn't mention the valid and protection bits, then you lost 2 points. If you added incorrect fields, we subtracted one point for each incorrect field.

Question

You are designing a paged virtual memory system on a system with 32-bit virtual addresses and 48-bit physical addresses. Each page is 4KB (2^{12} bytes), and each page table entry is 64 bits (8 bytes, 2^3 bytes).

a. (2 marks)

In a single-level paged system, how many bits of a virtual memory address would refer to the page number and how many to the offset? How many bits of a physical address would refer to the frame number and how many to the offset?

b. (2 marks)

How many bytes would a single-level page table require?

c. (2 marks)

If a page table entry contains a frame number, a valid bit, a writeable bit, and a single bit for tracking page useage, how many bits per page table entry are unused?

d. (2 marks)

How many page table entries fit onto a single page?

e. (2 marks)

What is the minimum number of levels necessary to implement a multi-level paged system if each page table at each level must fit into a single page?

https://student.cs.uwaterloo.ca/~cs350/common/old-exams/F18-midterm-sol.pdf

Question

You are designing a paged virtual memory system on a system with 32-bit virtual addresses and 48-bit physical addresses. Each page is 4KB (2^{12} bytes), and each page table entry is 64 bits (8 bytes, 2^3 bytes).

a. (2 marks)

In a single-level paged system, how many bits of a virtual memory address would refer to the page number and how many to the offset? How many bits of a physical address would refer to the frame number and how many to the offset?

LAS = 32 bits

PAS = 48 bits

Page size = 2^{12} B

Question

You are designing a paged virtual memory system on a system with 32-bit virtual addresses and 48-bit physical addresses. Each page is 4KB (2^{12} bytes), and each page table entry is 64 bits (8 bytes, 2^3 bytes).

b. (2 marks)

How many bytes would a single-level page table require?

c. (2 marks)

If a page table entry contains a frame number, a valid bit, a writeable bit, and a single bit for tracking page useage, how many bits per page table entry are unused?

LAS = 32 bits

PAS = 48 bits

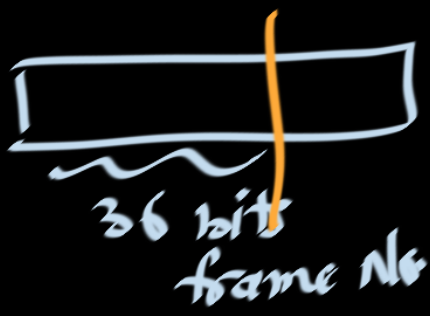
b) PTS $\Rightarrow 2^{20}$ entries $\times 2^3$ B

$= 2^{23}$ B

c)

PTE = 64 bits

64 − 39 = 25



Question

You are designing a paged virtual memory system on a system with 32-bit virtual addresses and 48-bit physical addresses. Each page is 4KB (2^{12} bytes), and each page table entry is 64 bits (8 bytes, 2^3 bytes).

LAS = 32 bits

PAS = 48 bits

d. (2 marks)

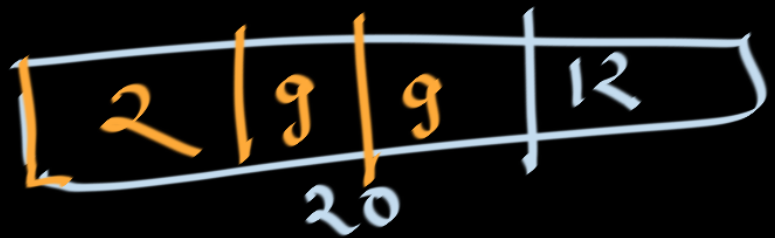
How many page table entries fit onto a single page?

e. (2 marks)

What is the minimum number of levels necessary to implement a multi-level paged system if each page table at each level must fit into a single page?

Page size = 2^{12} B = 2^9 entries

PTE = 2^3 B



Page size = 2^9 entries

we can stop here itself if chunk size is allowed to be anything

if we stop here, then no. of chunks = 1

chunk size = 2^{20} entries

$= 2^{23}$ Bytes

Page size = 2^9 entries

we can stop here itself if chunk size is allowed to be anything

if we stop here, then no. of chunks = 1

chunk size = 2^{20} entries
$= 2^{23}$ Bytes

Multilevel paging is our choice (Not forced), we can stop at any level.

Chunk size

$= 2^{20}$ entries

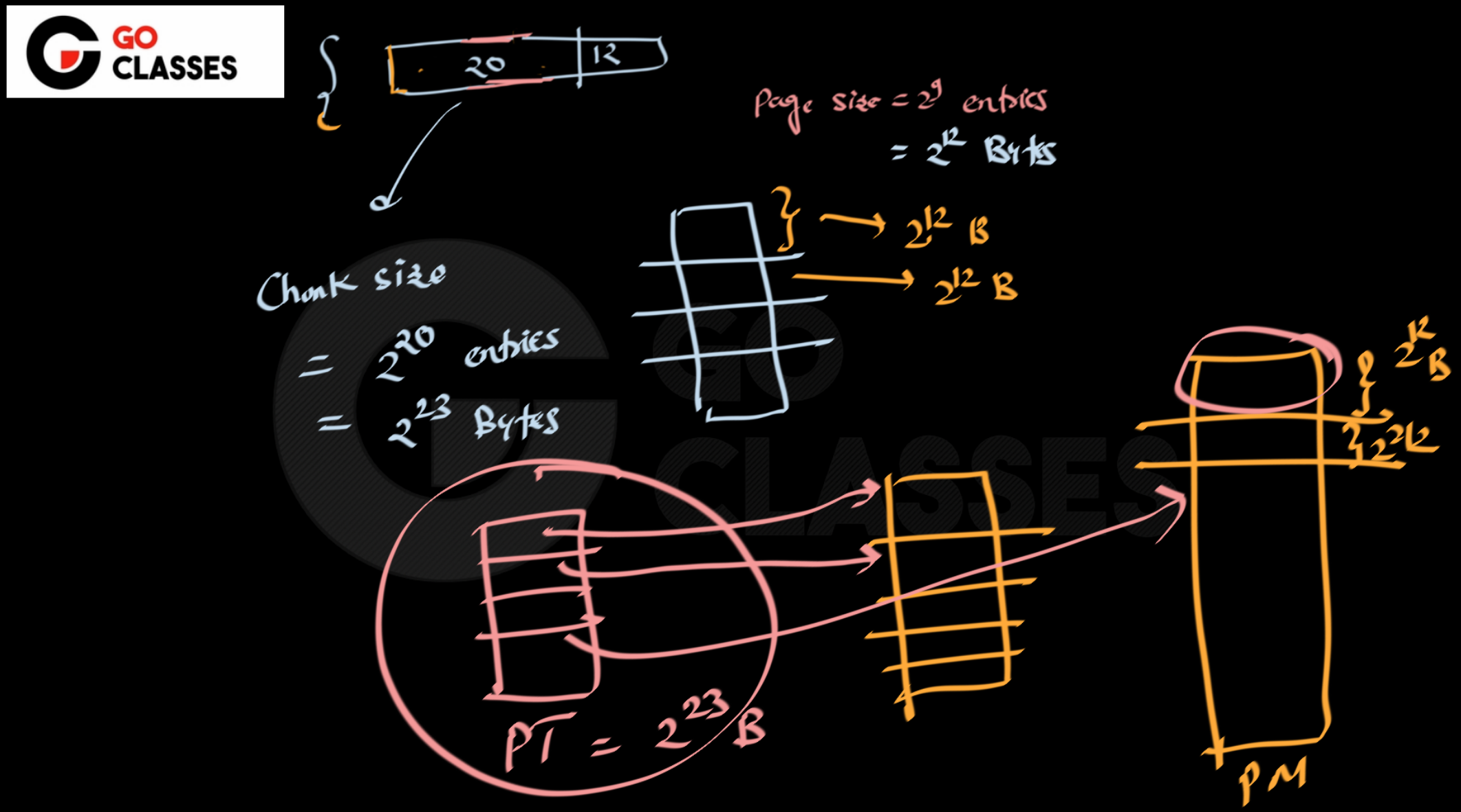
$= 2^{23}$ Bytes

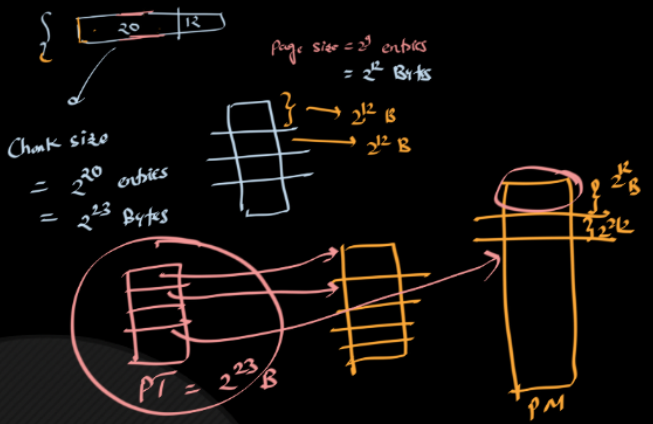
Page size $= 2^9$ entries

$= 2^{12}$ Bytes

2^{12} B

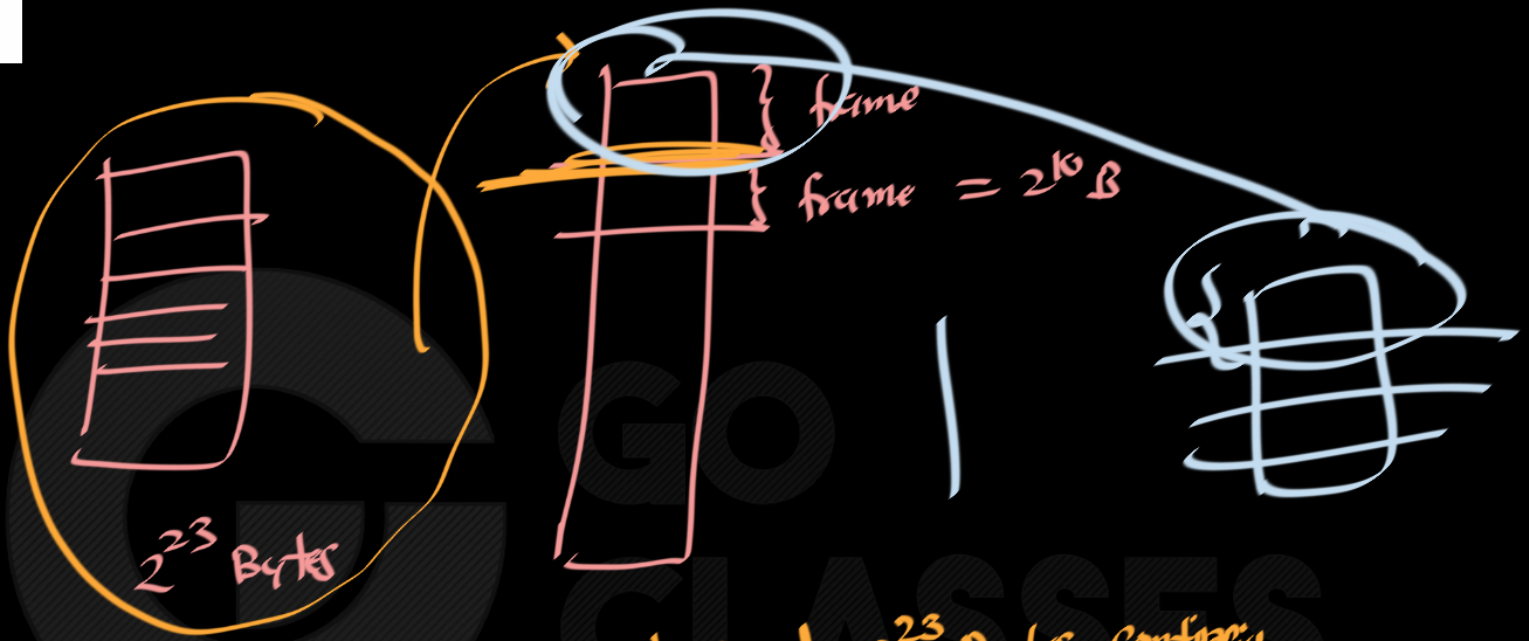
2^{12} B


GO CLASSES
20
12
Page size = 2^9 entries
= 2^{12} Bytes
2^{12} B
2^{12} B
Chunk size
= 2^{20} entries
= 2^{23} Bytes
2^{12} B
PT = 2^{23} B
PM



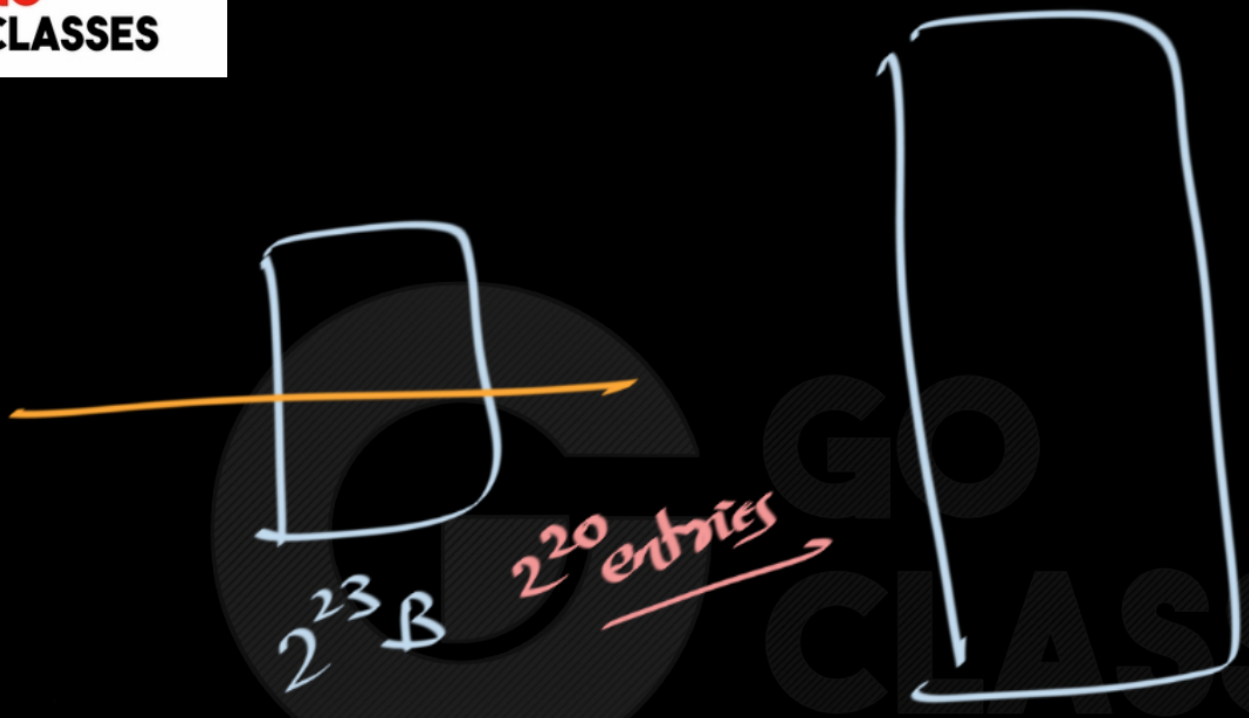
How to store PT = 2^{23} B into physical memory if memory itself is divided into frames of lesser size

in other words how we gonna fit large PT into one frame?

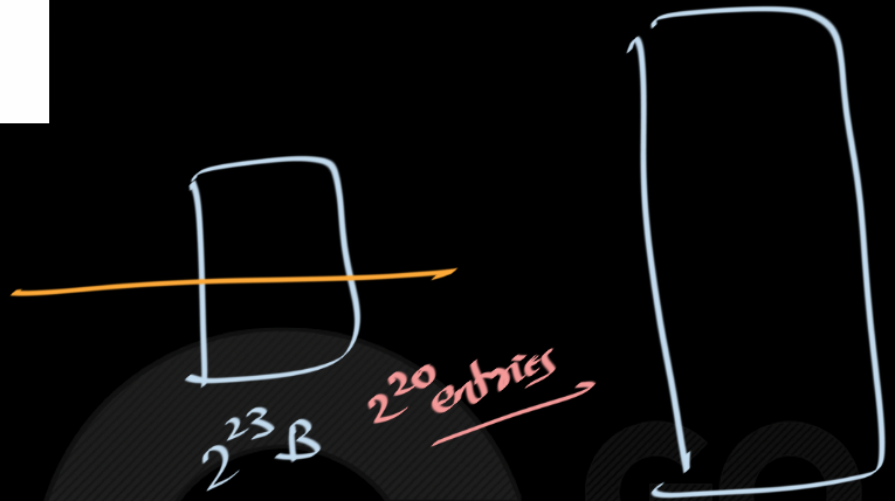


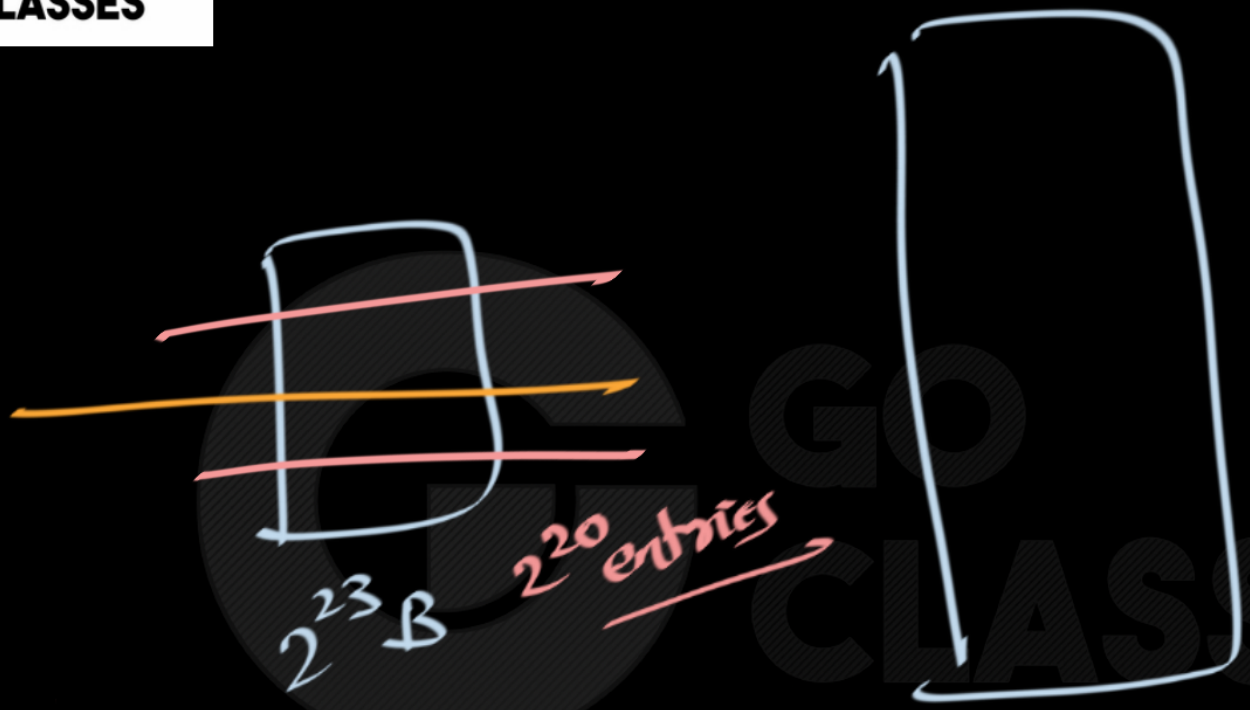
to save PT into MM, we just need 2^{23} Bytes contiguously

(why you even want to fit into one frame?)

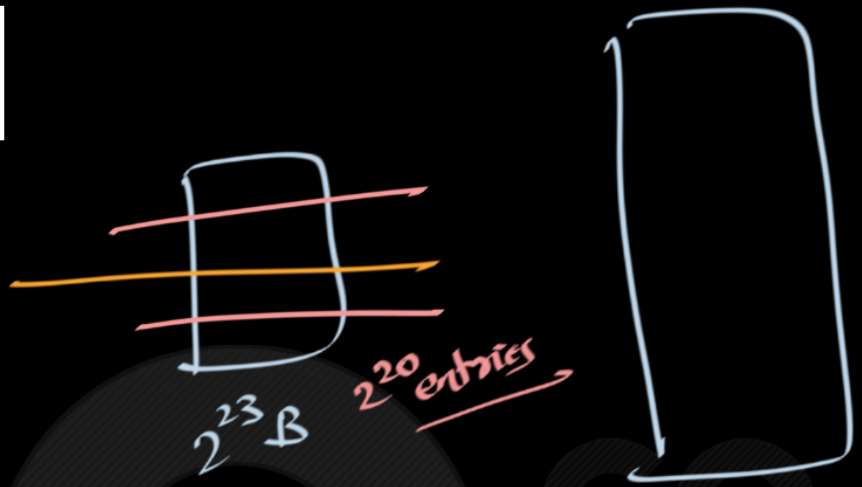


Can we divide into
two parts and store
mm


2^{23} B
2^{20} entries
Can we divide into two parts and store?
Yes, why not.
Chunk size = 2^{19} entries
1
19
12

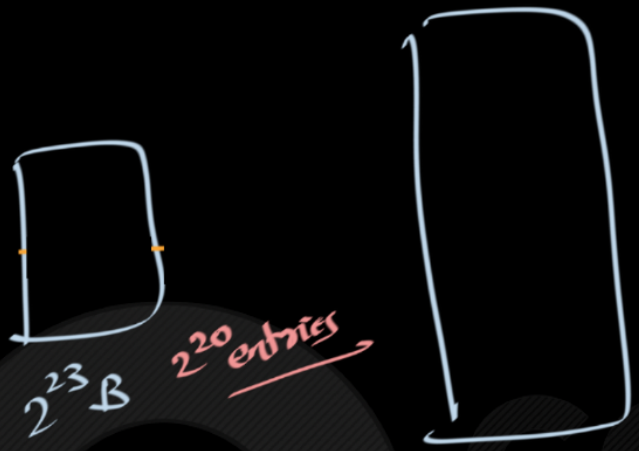


Can we divide into FOUR parts and store?



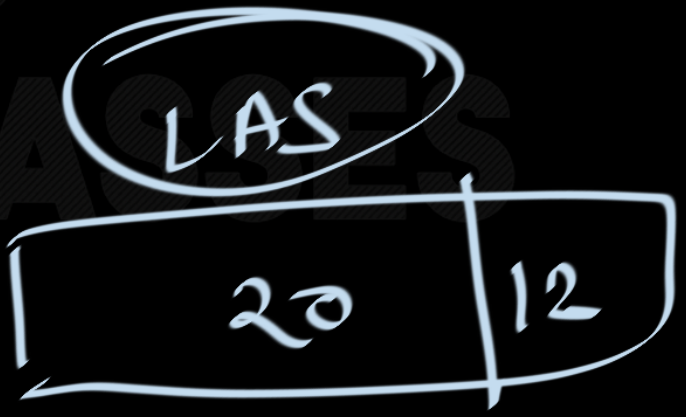
Can we divide into FOUR parts and store?

$$\frac{2^{20}\text{ entries}}{2^2} = 2^{18}\text{ entries}$$



Can we Choose Not to divide at all?

Yes, why not

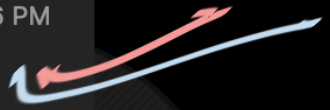


GO CLASSES

Sangeet Bhowmick to Everyone 8:46 PM

SB

just a wild guess cant PT span over multiple frames



Akash Maji to Everyone 8:44 PM

AM

2^23 / 2 ^ 12 = 2^11 pages needed,

store contiguously, anyway pages is no physical boundary

a. (2 marks)

$20, 12, 36, 12$

b. (2 marks)

2^{23}

c. (2 marks)

$64 - 36 - 3 = 25$

d. (2 marks)

$2^9 = 512$

e. (2 marks)

3

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?

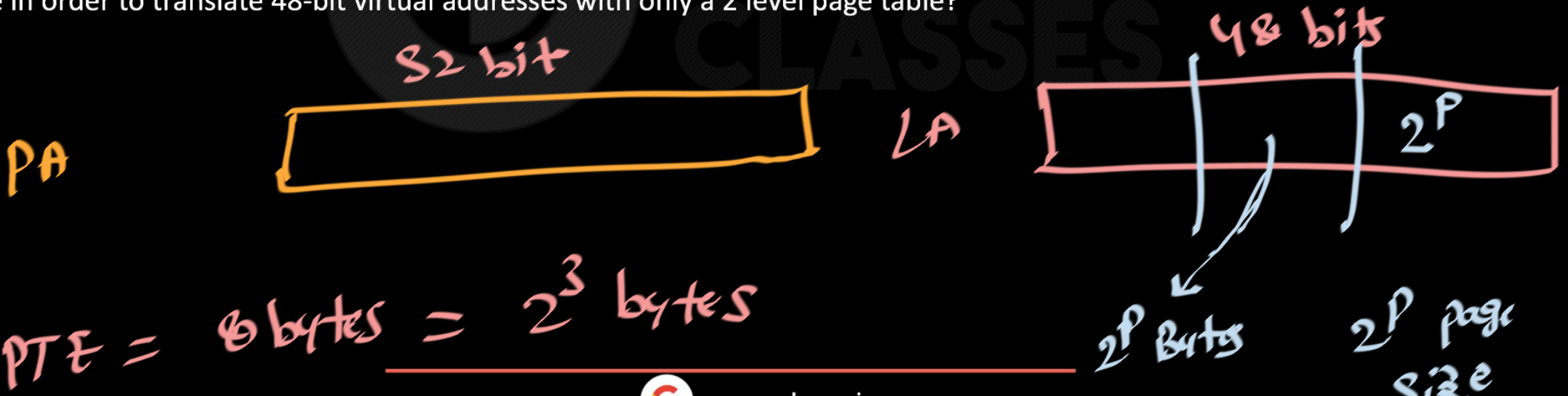
Question

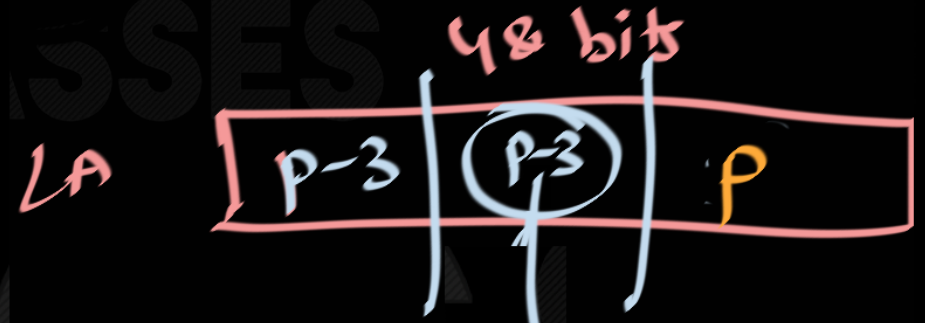
Consider a system, Starting with a 48-bit virtual addresses, it uses 4 levels of page table to translate the address to a 52-bit physical address.

Each page table entry is 8 bytes long.

given at every level Page tables must fit into a page.

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?





$p-3+p-3+p=48$

$\Rightarrow p=18$

Page Size $= 2^p$ B

PTE $= 2^3$ B

Chunk Size

2^p Bytes

$= 2^{p-3}$ entries

Page Size $= 2^{18}$ B

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).

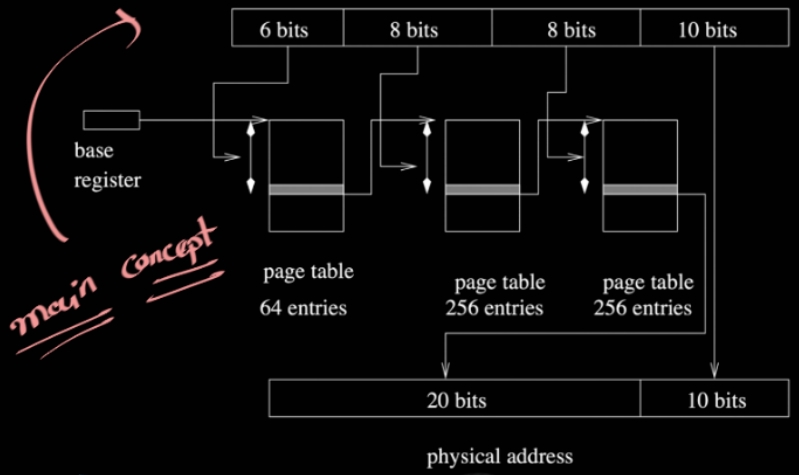
For a two level page table, say page size is 2^n. The page table size is also 2^n bytes in this multi-level page table. So the number of PTE in a page table is 2^(n-3).

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.



Main Concept

→ the address translation in multilevel paging

→ what each entry of page table contains

→ Given that page table must fit into one page
Calculate the min. no. of levels

→ minimum no. of levels are given ⇒ Calculate page size

GO CLASSES

Solve ALL PYQs purely related to multilevel paging

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 ?

A Typical System has:

Page size = 4 KB, 32 bit LA

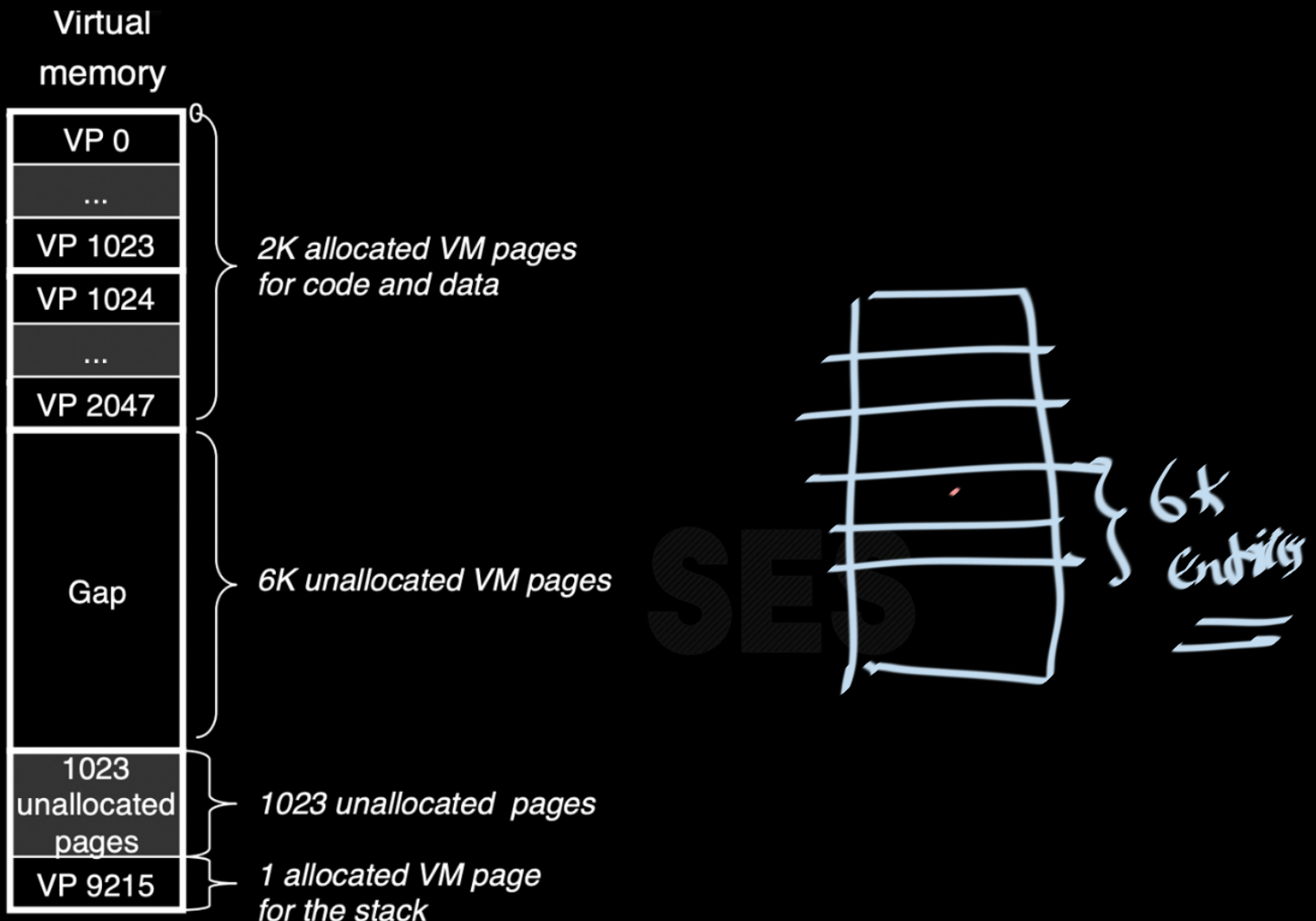
So, $\frac{2^{32}}{4\ KB} = \frac{2^{32}}{2^{12}} = 2^{20}\ pages$

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.

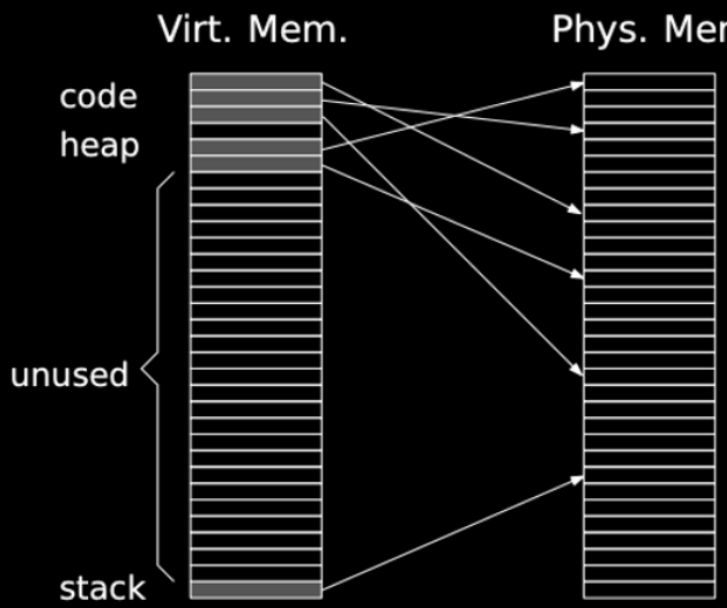
```
main ( )
{
    int a, *p;
    p = malloc(4);
}
```


Code
Stack
J
empty
P
heap
page table


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

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

A Typical System has:

Page size = 4 KB, 32 bit LA

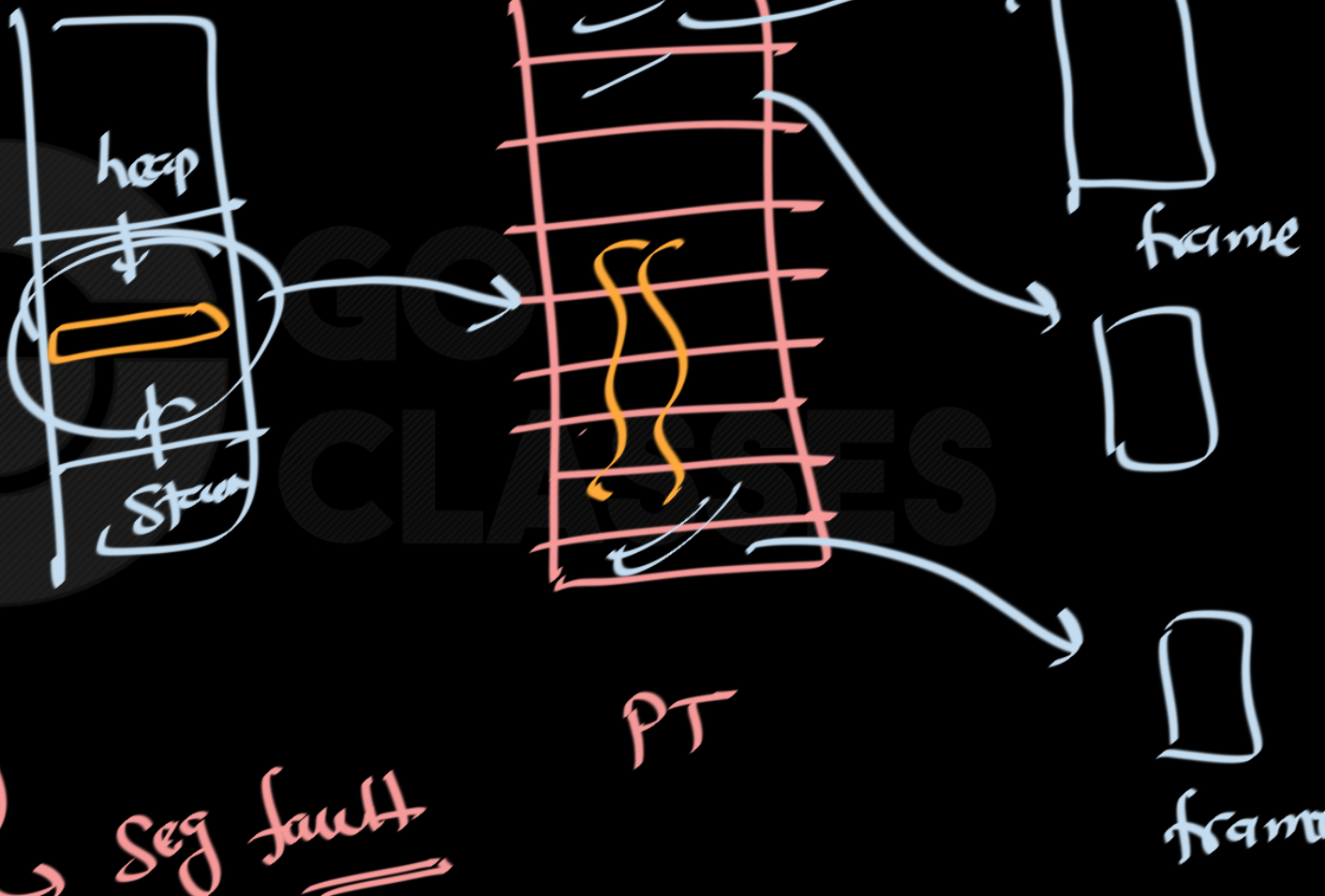
So, $\frac{2^{32}}{4\ KB} = \frac{2^{32}}{2^{12}} = 2^{20}\ pages$

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

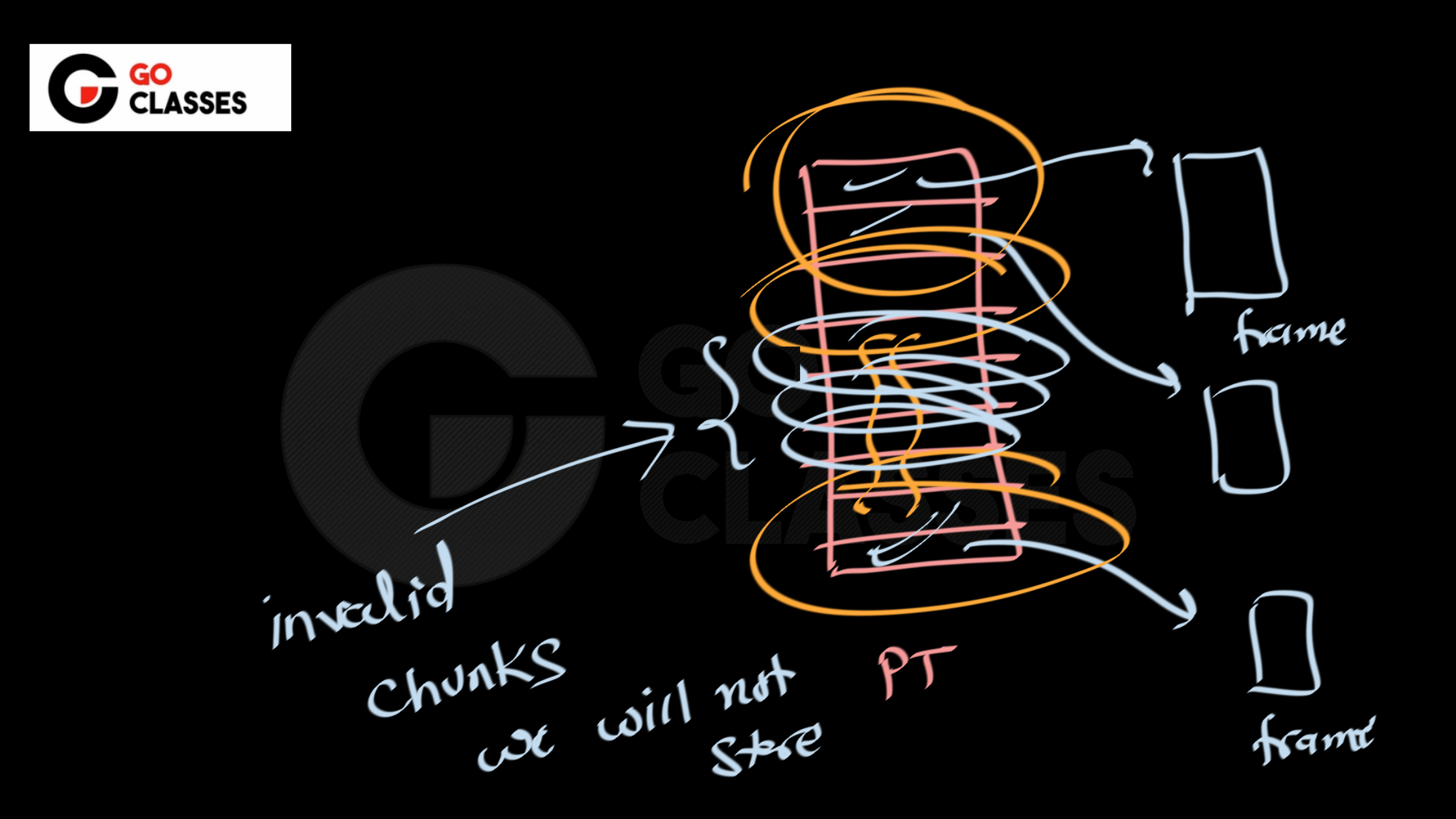
- One such page table for each process.

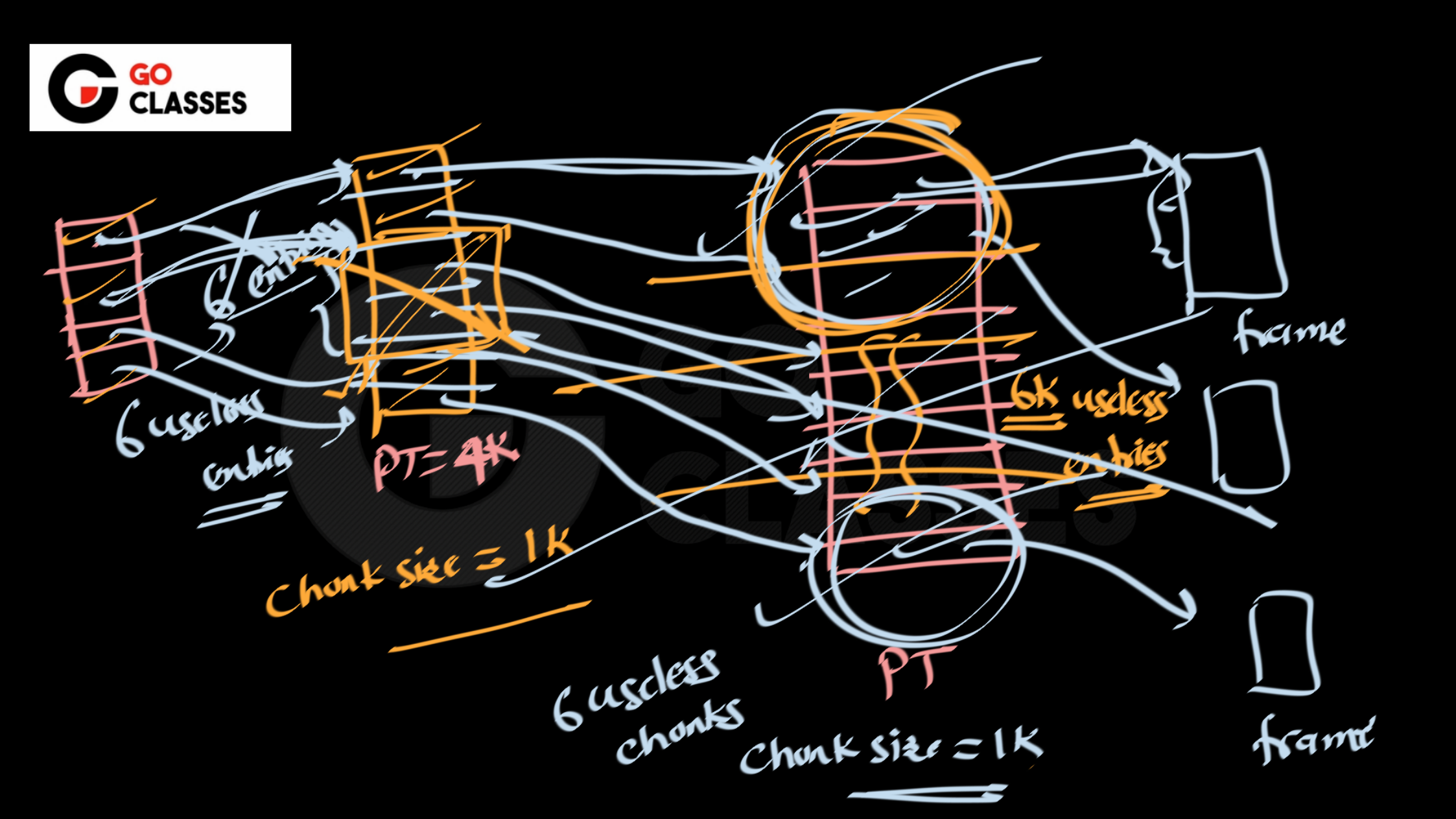
GO CLASSES

```
main()
{
int *p;
p = malloc(4)
*p=5
Pf("%d", *(p+50));
}
```



Seg fault


GO CLASSES
invalid
chunks
we will not
store
PT
frame
frame


GO CLASSES
6 useless entries
PT = 4K
Chunk size = 1K
6K useless entries
frame
6 useless chunks
PT
Chunk size = 1K
frame

GO CLASSES

Intuition :

- if we do not divide page table (single level page table) then we do not have any choice but to store ALL entries (including invalid one's)

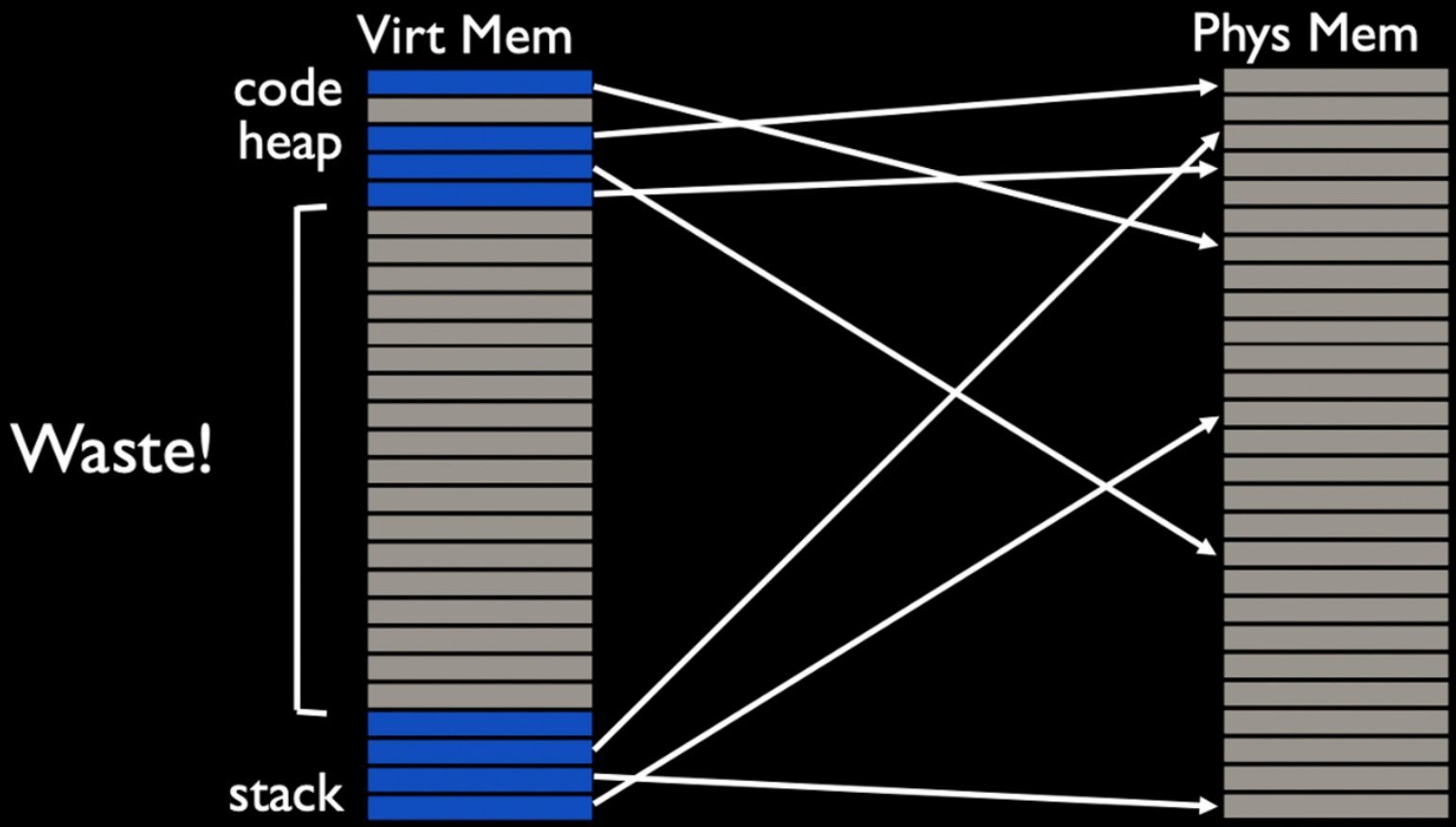
GO CLASSES

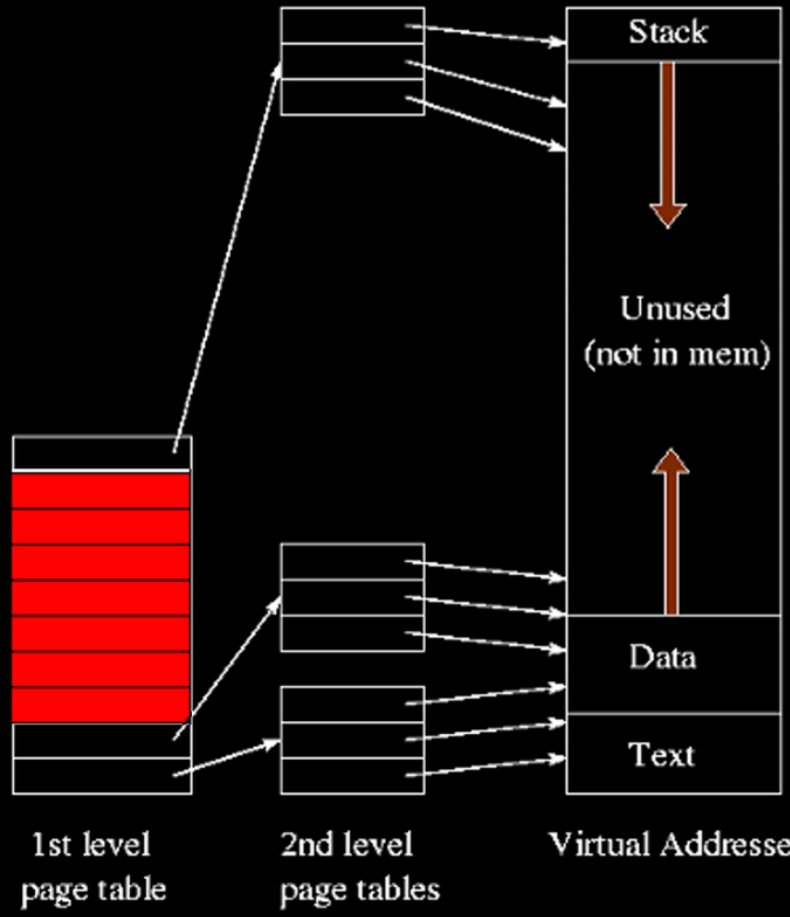
Intuition:

if we do not divide page table (single level page table) then we do not have any choice but to store ALL entries (including invalid one's)

* if we divide the page table into multiple pages (chunks) then we have choice to save chunks independently hence we can avoid saving invalid entry pages into physical mem.

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).