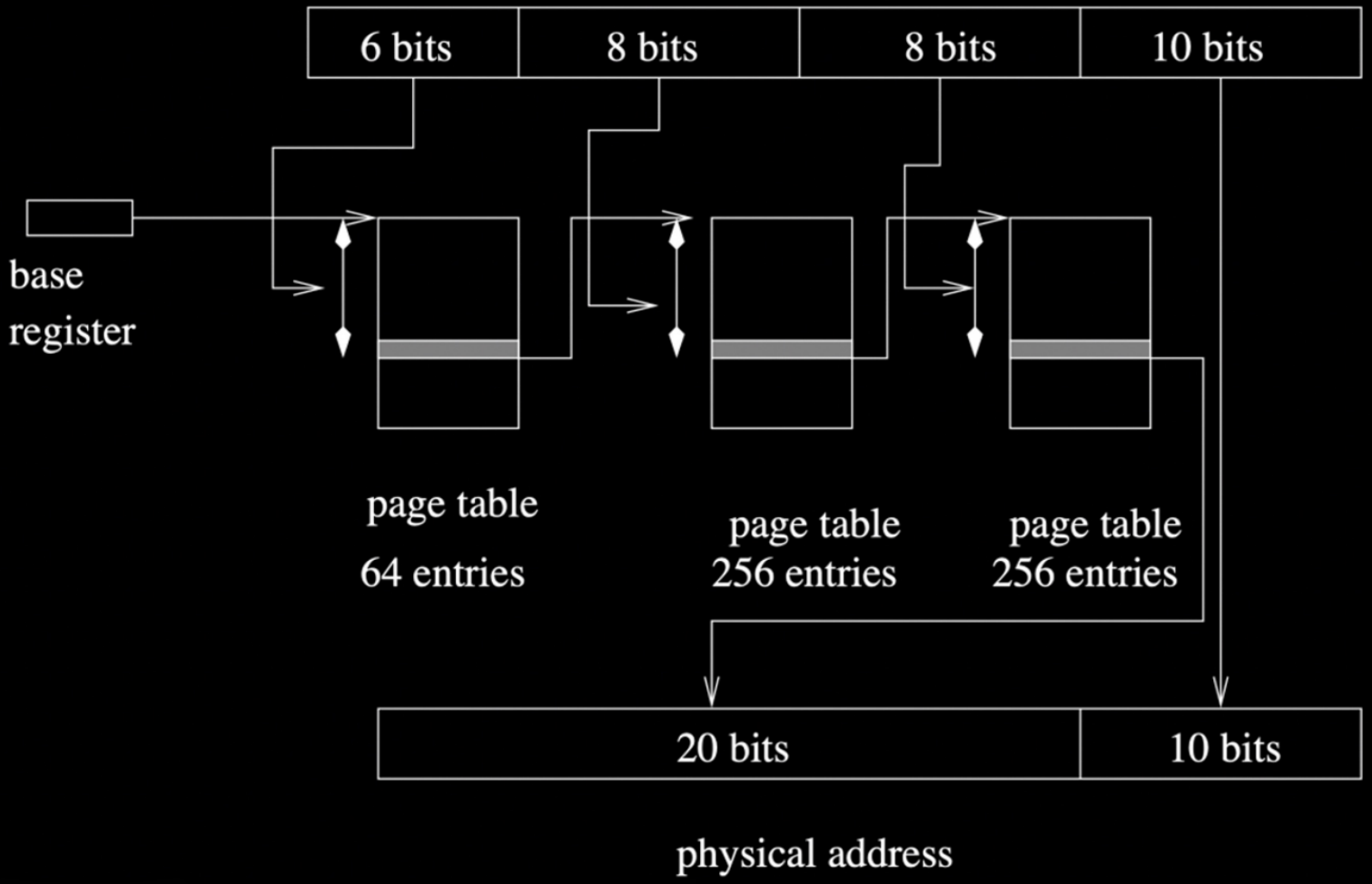
Lecture: 29

6 bits	8 bits	8 bits	10 bits

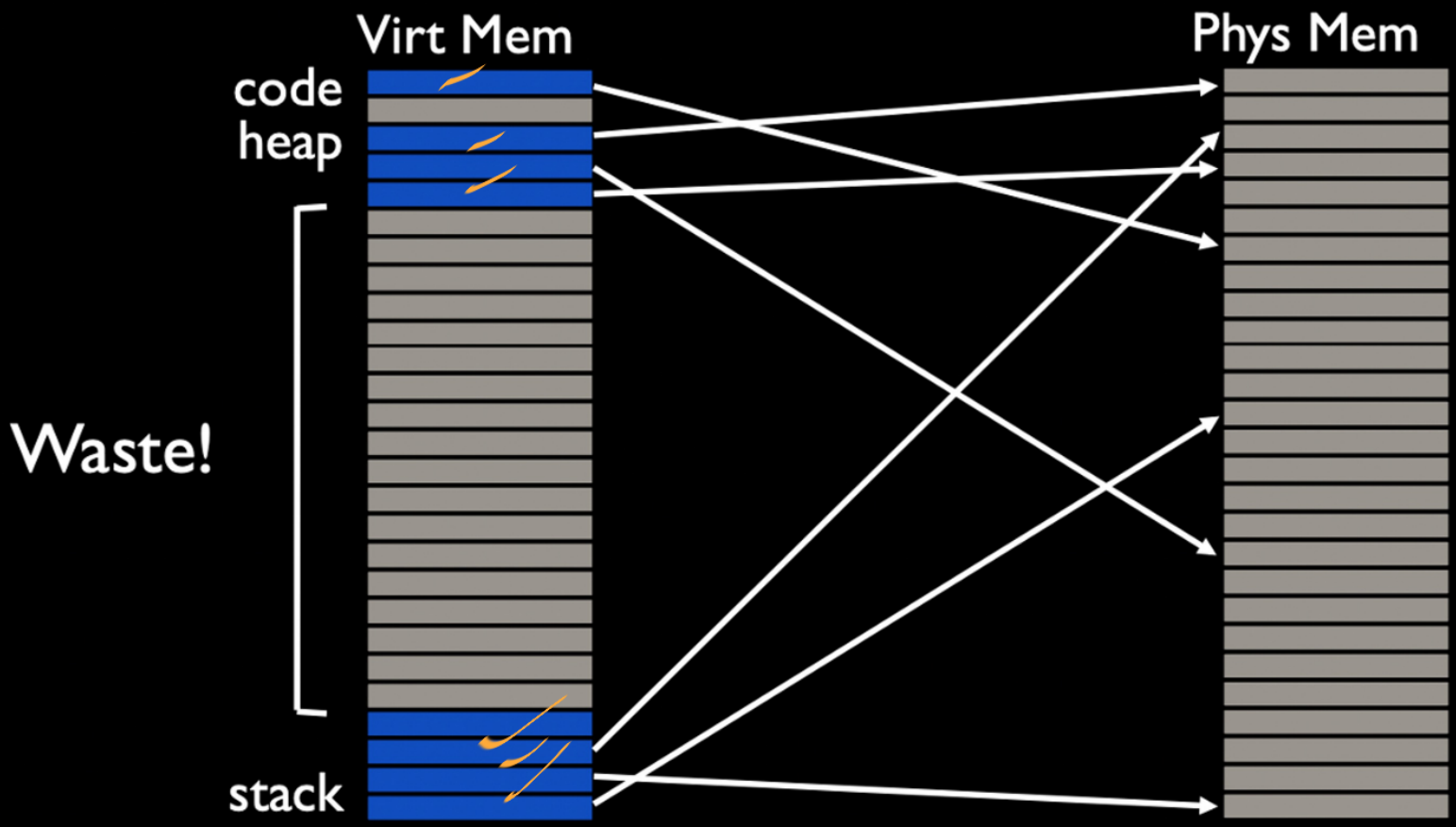


Crux of multi-level paging

But where are the savings ?

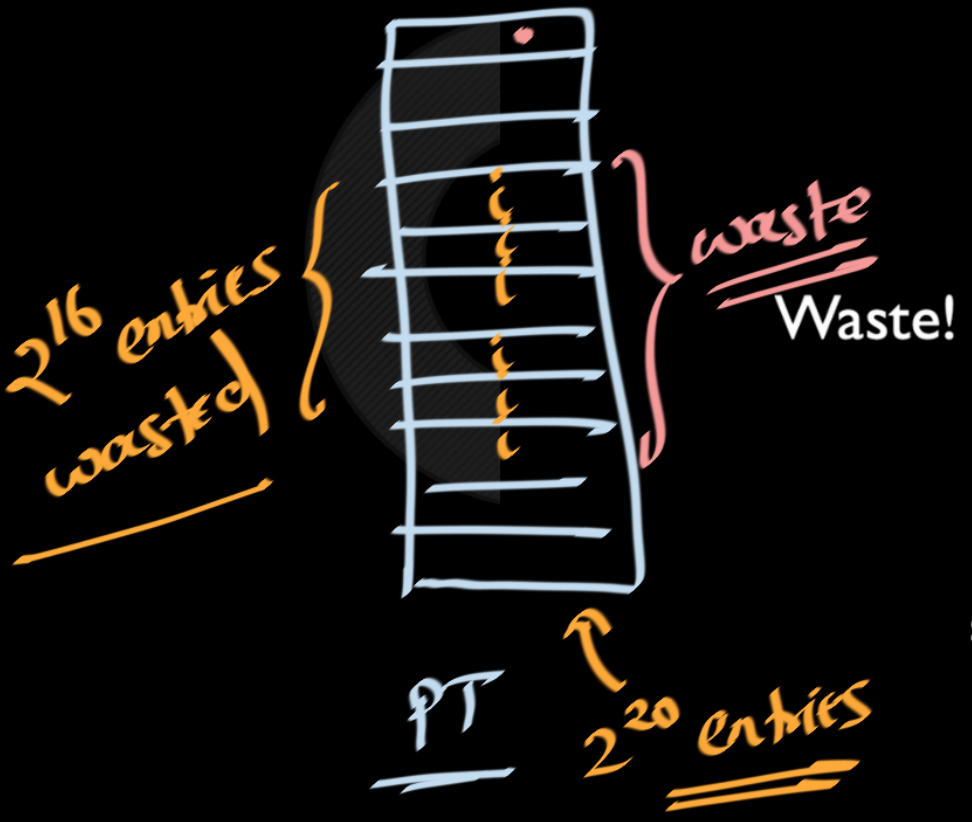
It seems we have added one more page table and not saving any space.

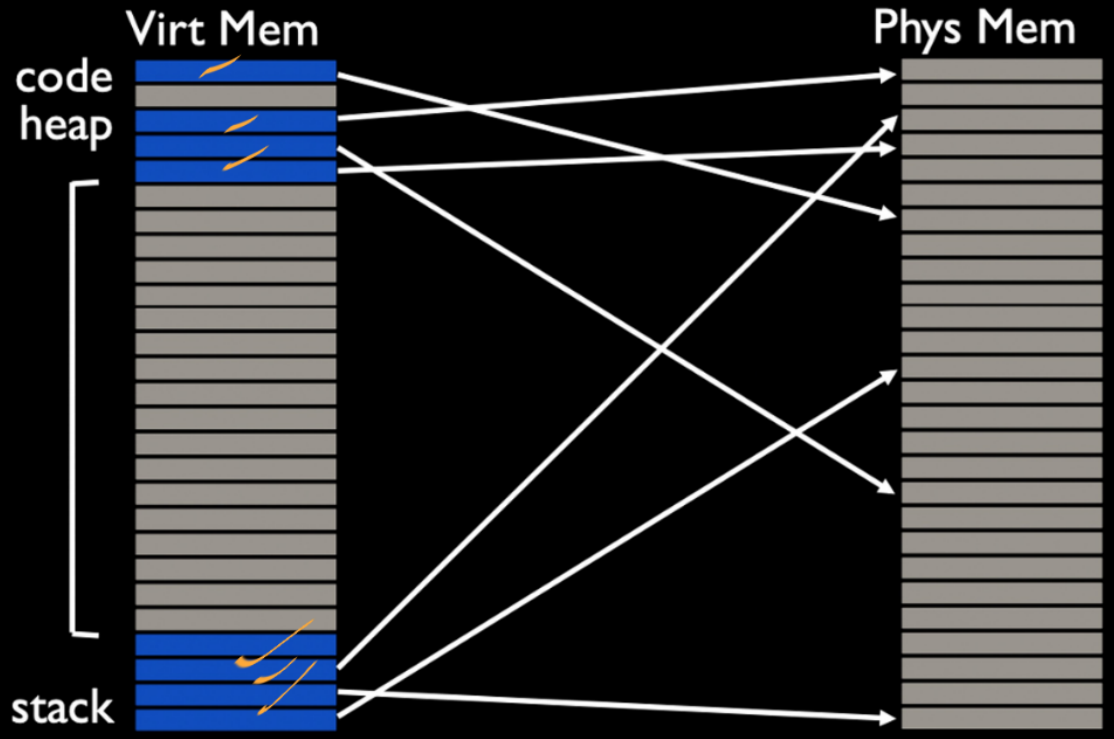
Where are savings that we promised earlier ?


Virt Mem
Phys Mem
code
heap
Waste!
stack

GO CLASSES

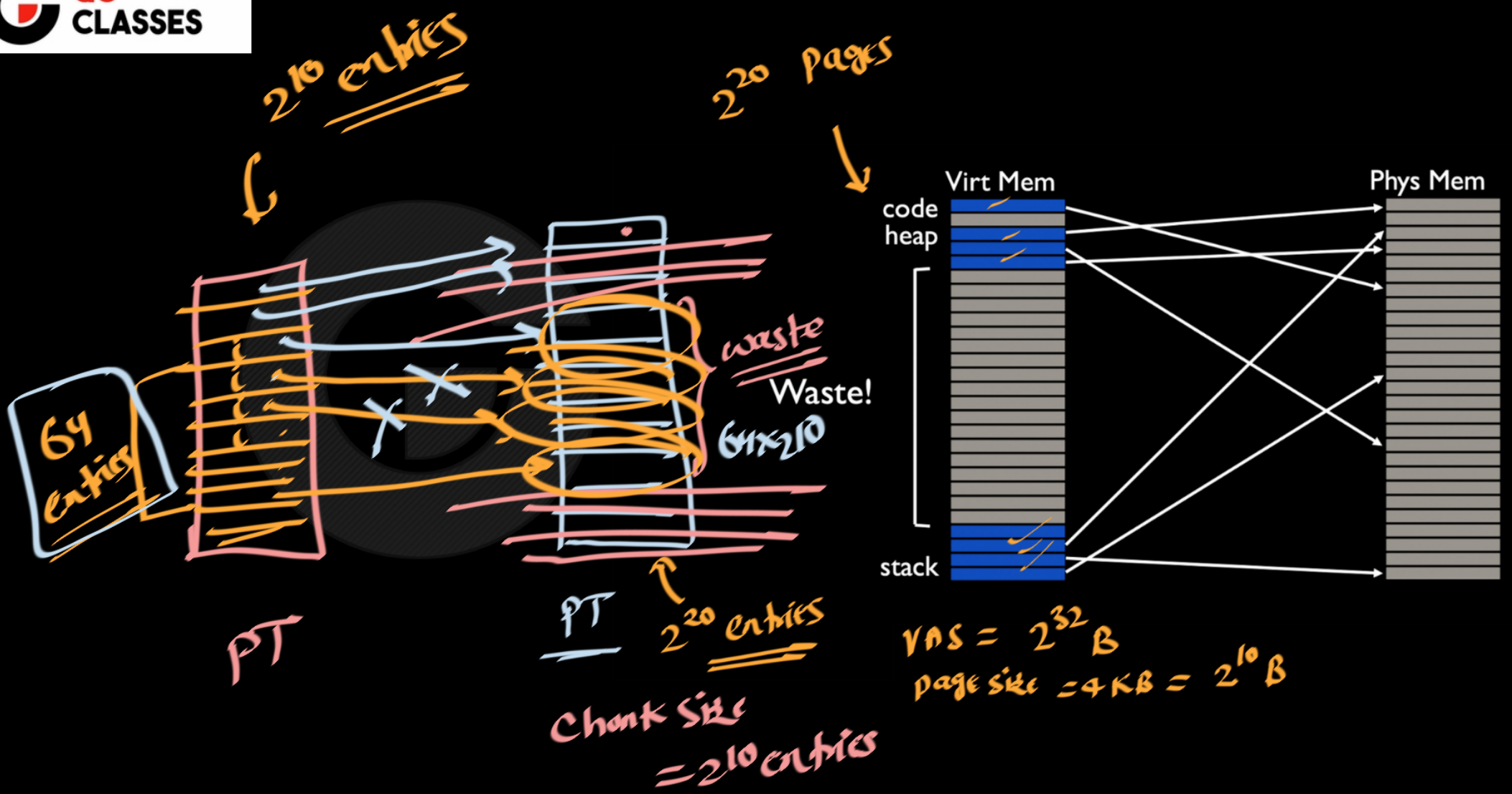
2^{20} pages

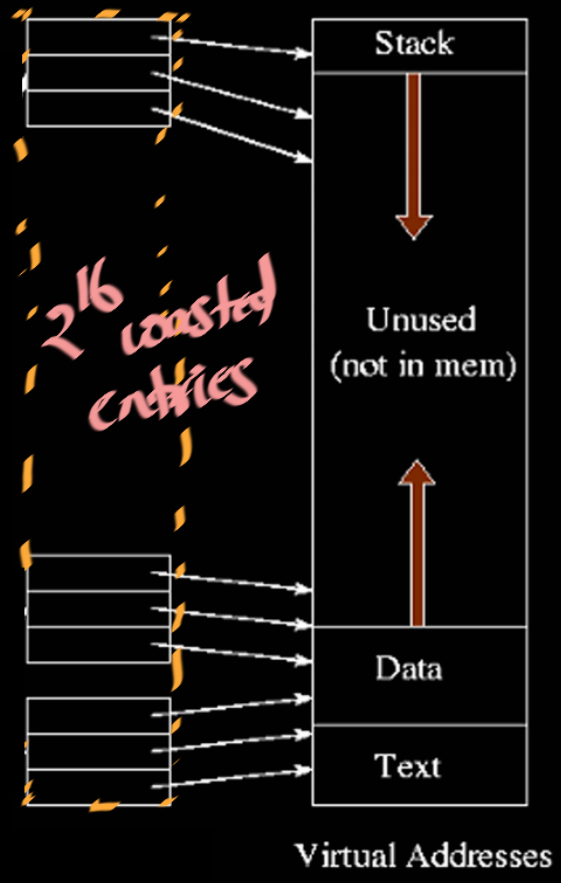


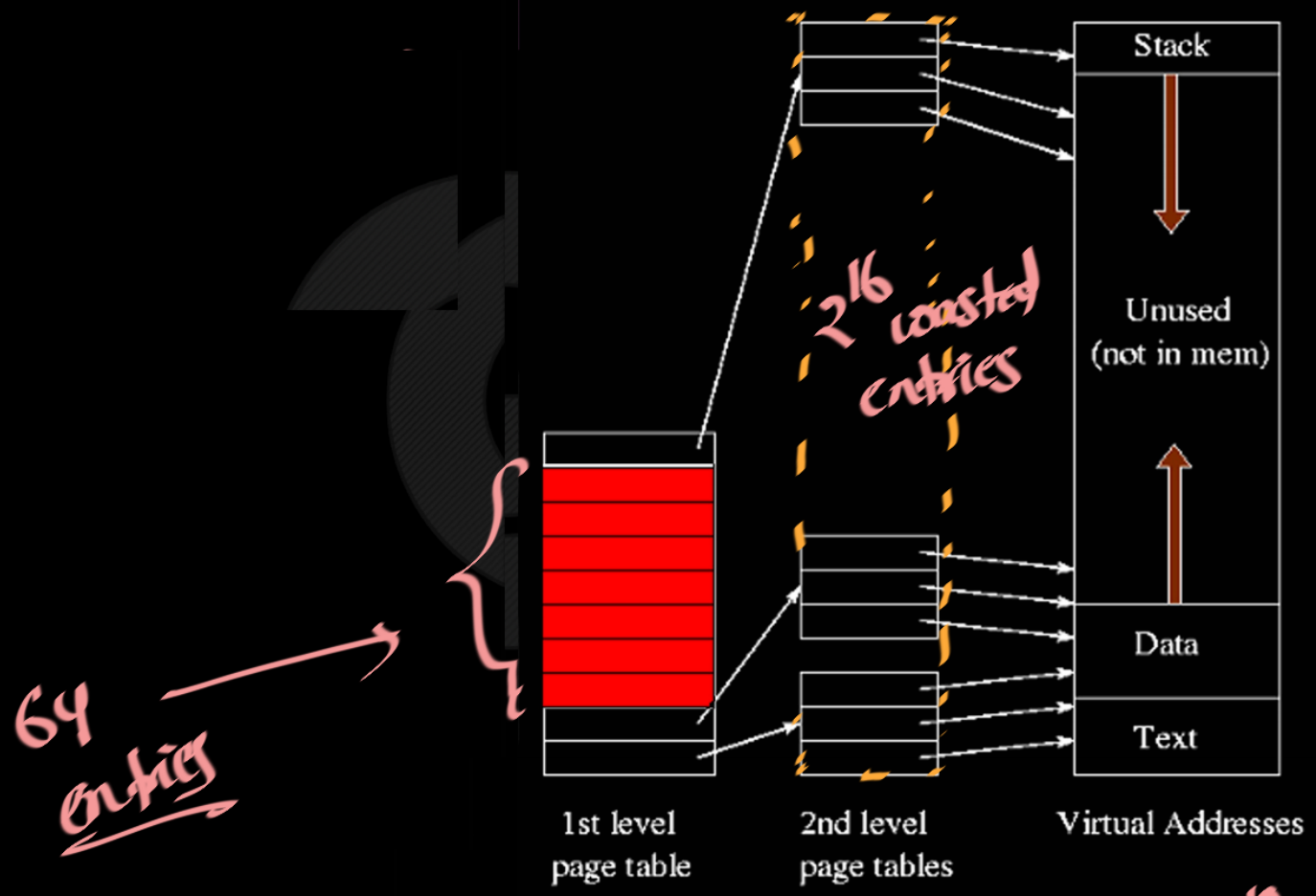


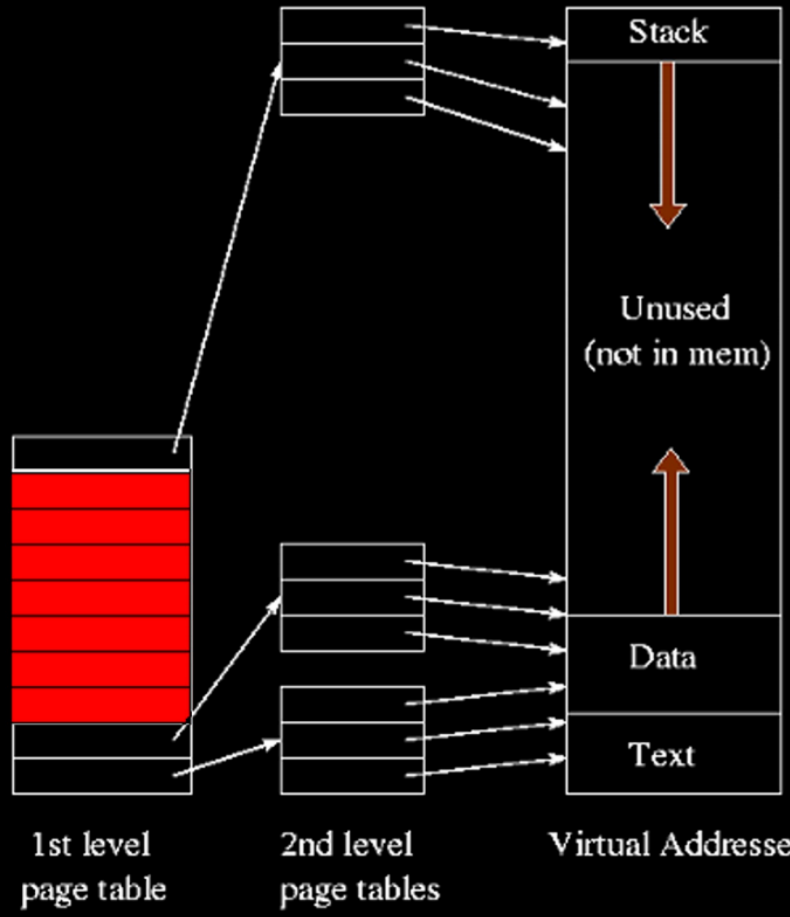
VAS = 2^{32} B

page size = 4 KB = 2^{12} B


GO CLASSES
2^{10} entries
2^{20} pages
64 entries
waste
Waste!
6bitx2b
PT
PT
2^{20} entries
Chunk Size
=210 entries
Virt Mem
Phys Mem
code
heap
stack
VAS = 2^{32} B
page size = 4 KB = 2^{12} B


Stack
Unused
(not in mem)
Data
Text
Virtual Addresses


Stack
Unused
(not in mem)
Data
Text
1st level
page table
2nd level
page tables
Virtual Addresses
2^{16} unused entries
64 entries
chunk size = 2^{10} entries



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).

Question

A Process having 32 bit VAS

Page size = 4KB ; PTE = 4B

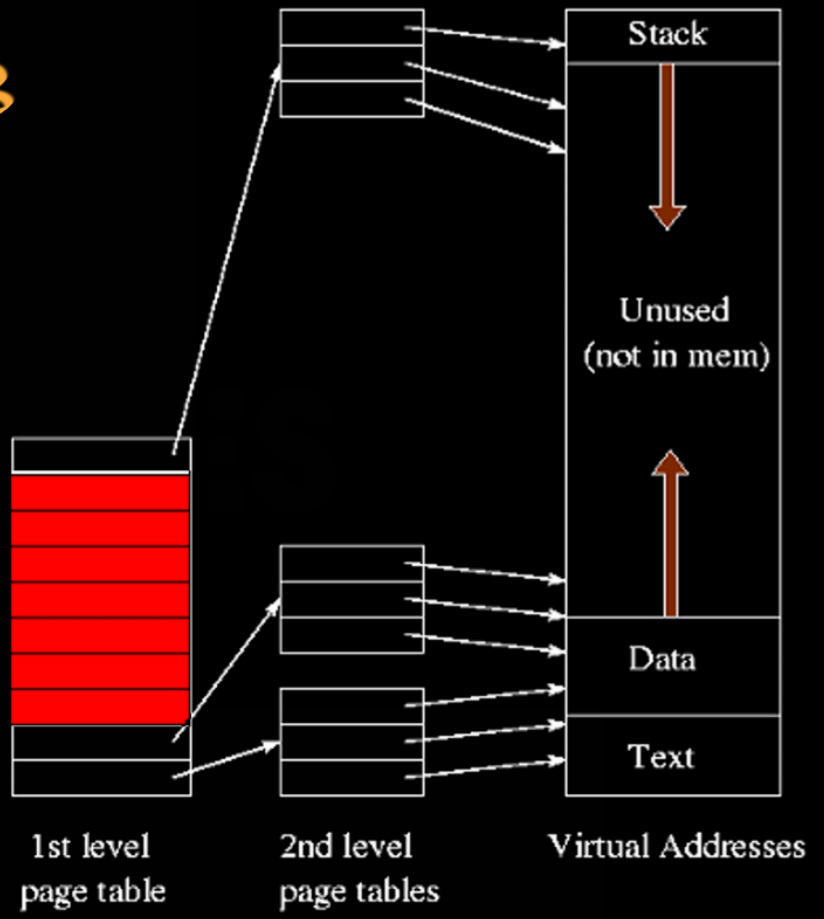
and using only 2^{16} Bytes

then calculate the size

of memory we need for

Page tables a) single level

b) 2 level



Question

A Process having 32 bit VAS

Page size = 4KB , PTE = 4B

and using only 2^{16} Bytes

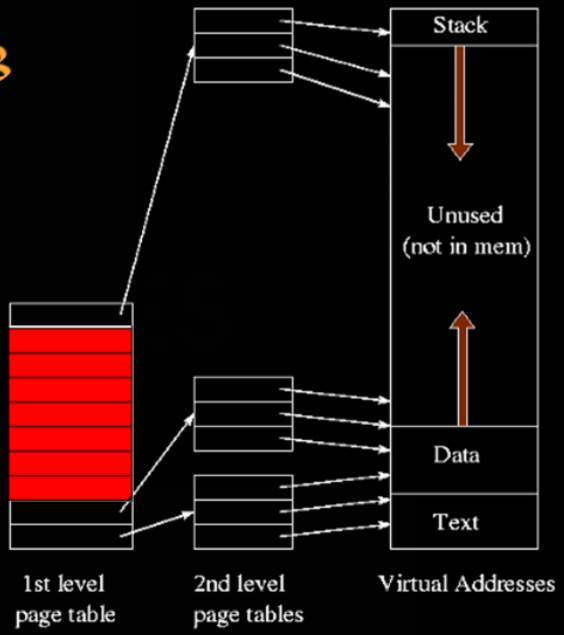
then calculate the size

of memory we need for

Page tables

a) single level

b) 2 level



a) No. of pages = 2^{20} = # of PTE

Size of Page table = $2^{20} \times 2^2$ B = 4MB

GO CLASSES

GO Classes

Question

A Process having 32 bit VAS

Page size = 4KB , PTE = 4B

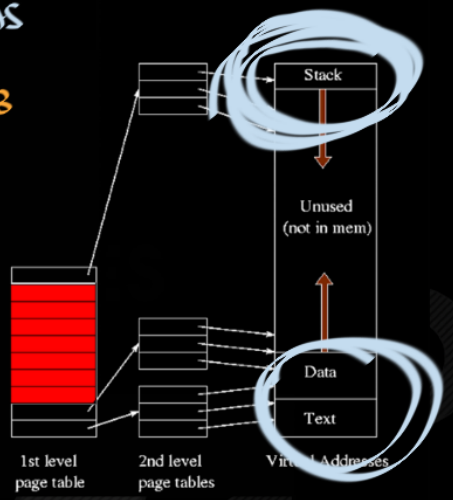
and using only 2^{16} Bytes

then calculate the size

of memory we need for

Page tables a) single level

b) 2 level



16 entries are useful at first level page table

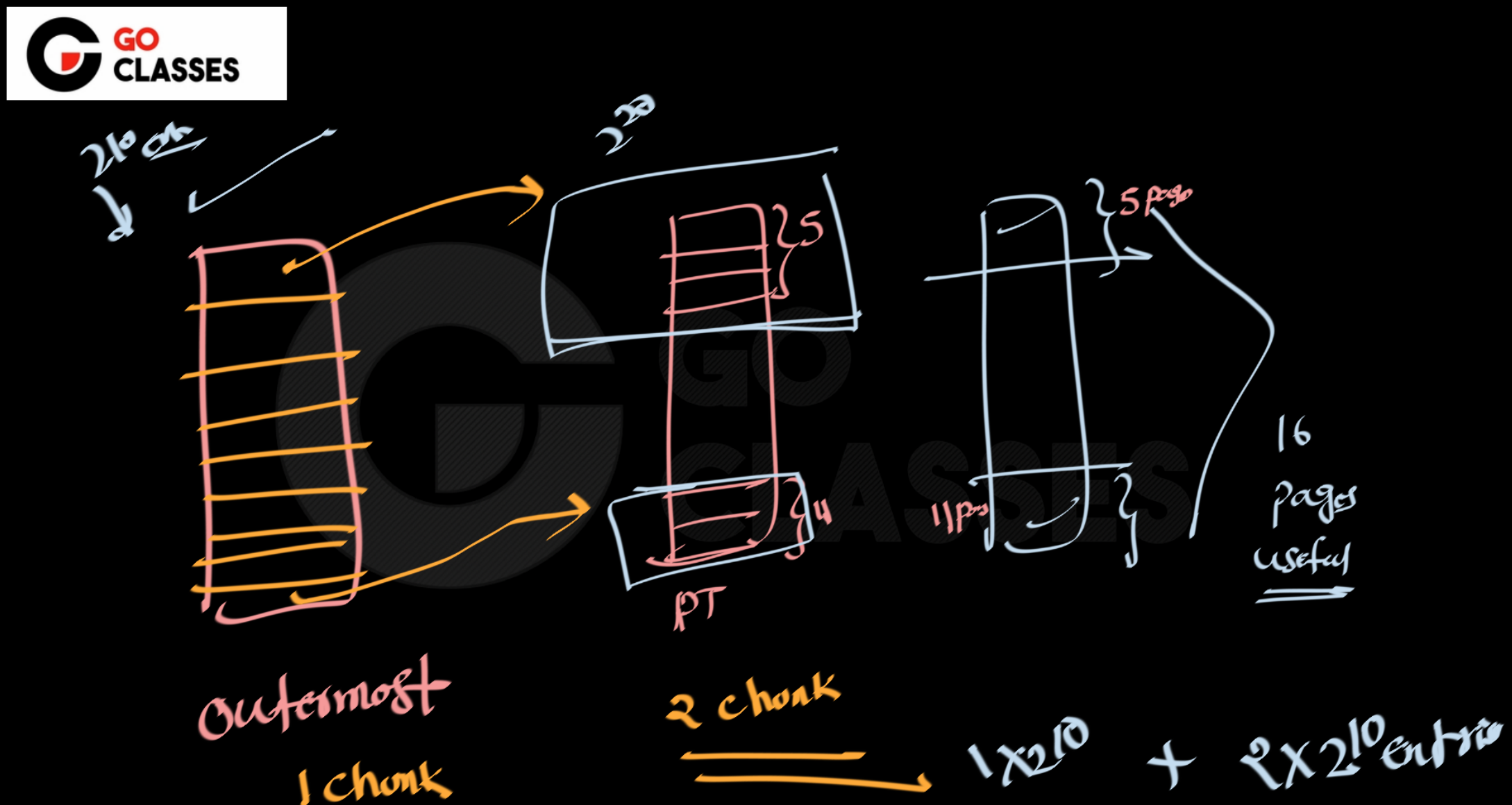
a) No. of pages = 2^{20} = # of PTE

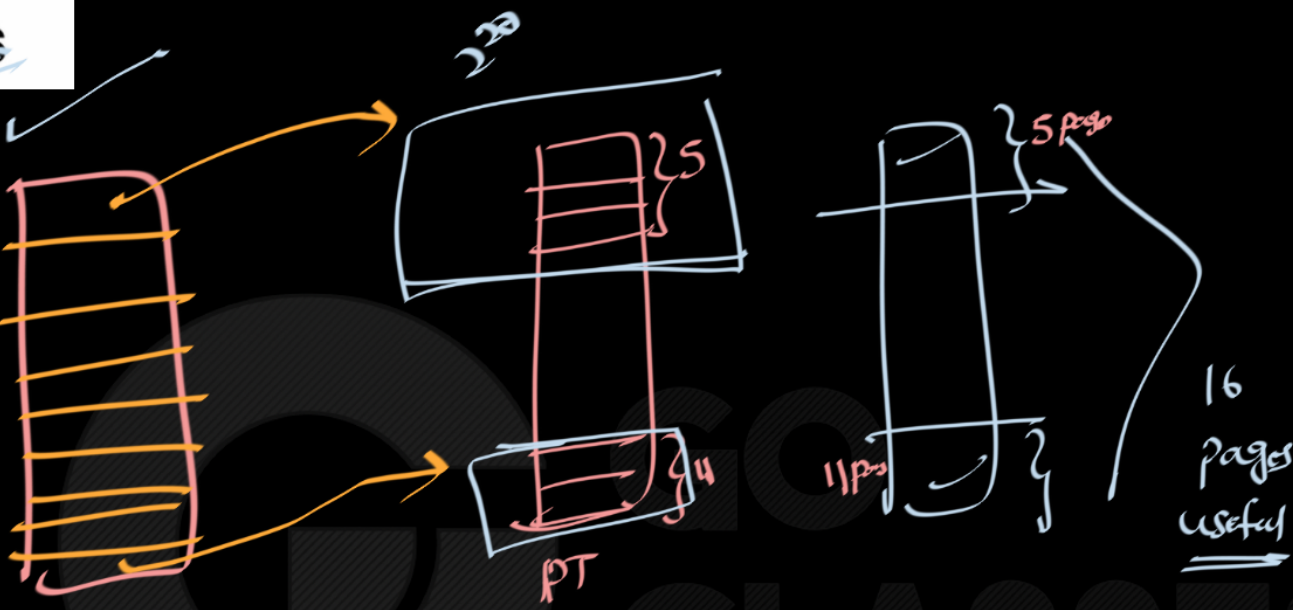
Size of page table = $2^{20} \times 2^2$ B = 4MB

Page size = 2^{10} entries

b) No. of pages in the process = 2^{20}

No. of useful pages = $2^{16} / 2^{12} = 2^4 = 16$ pages


GO CLASSES
2^{10} entries
2^{20}
5
PT
64
5 pages
11 pages
16 pages useful
Outermost
1 chunk
2 chunk
1×2^{10} + 2×2^{10} entries



Outermost
1 chunk

2^{10} entries

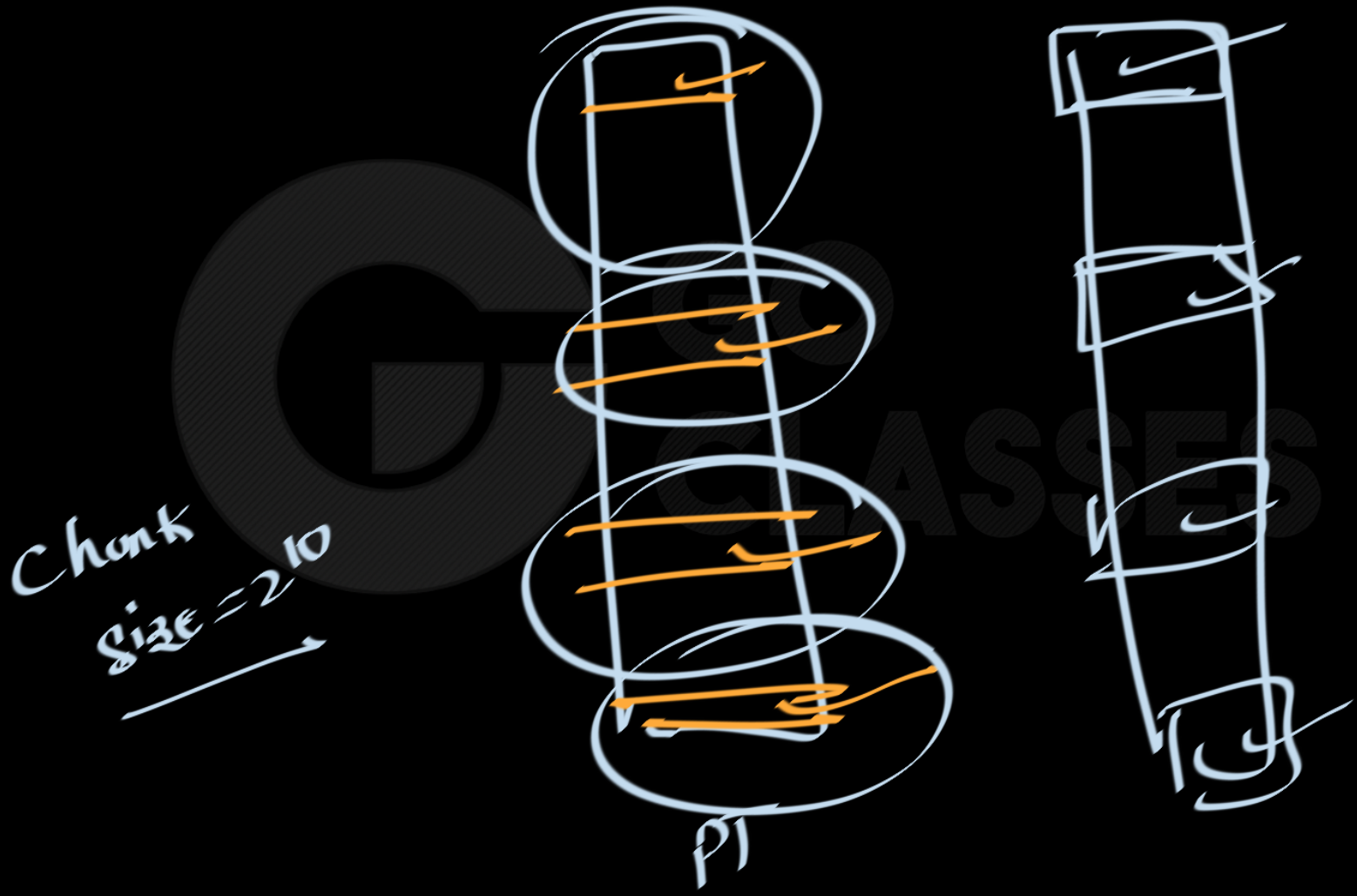
2 chunk

2^{10} entries

3×2^{10} entries

$3 \times 2^{10} \times 2^2$ B

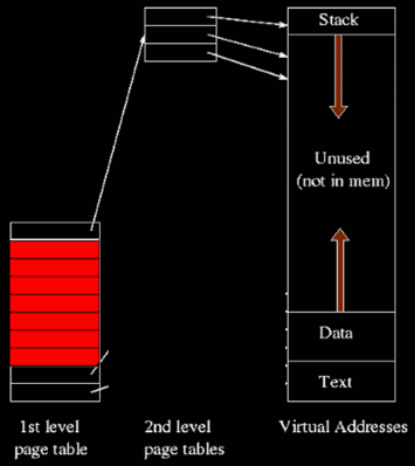
$= 12$KB


Chunk
Size = 2^10
P1

a) Single level

PTS = $2^{20} \times 2^2 B = 4\ MB$

b) Multi level page table



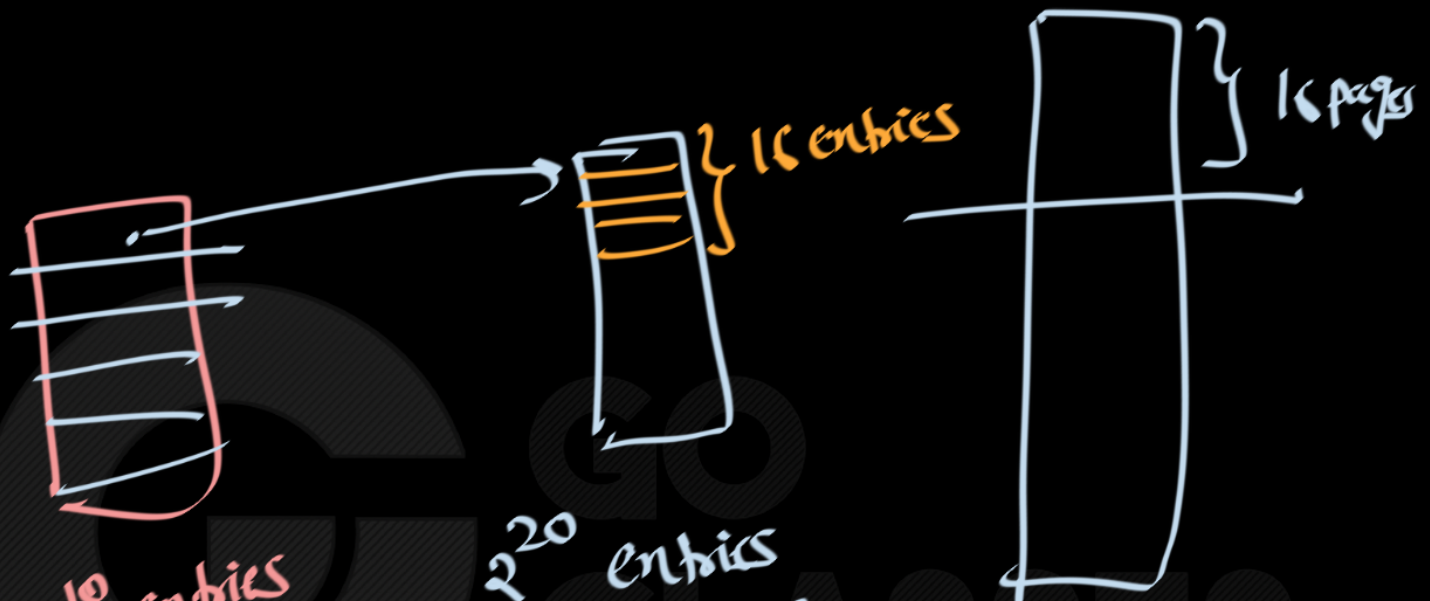
given: Process using only 16 pages, all pages are from address 0

2^{20} entries

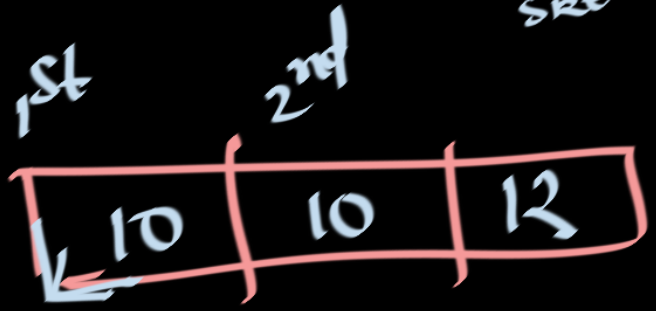
PTE = 4B

Calculate Size of PT in case of multilevel paging.

Page Size = 2^{10} entries

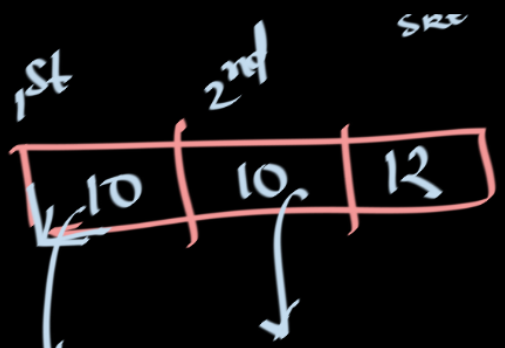


Address Split



GO CLASSES

Address split



useful chunks

1

Size of chunk

1

2^{10} entries

2^{10} entries

Size of all page tables = $2^{16} + 2^{10} = 2^{11}$ entries

$= 2^{11} \times 4 B = 8KB$

Question

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

Question

Problem 1

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

PTE = 2B

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?
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

GO CLASSES

10-bit	8-bit	6-bit	8 bit

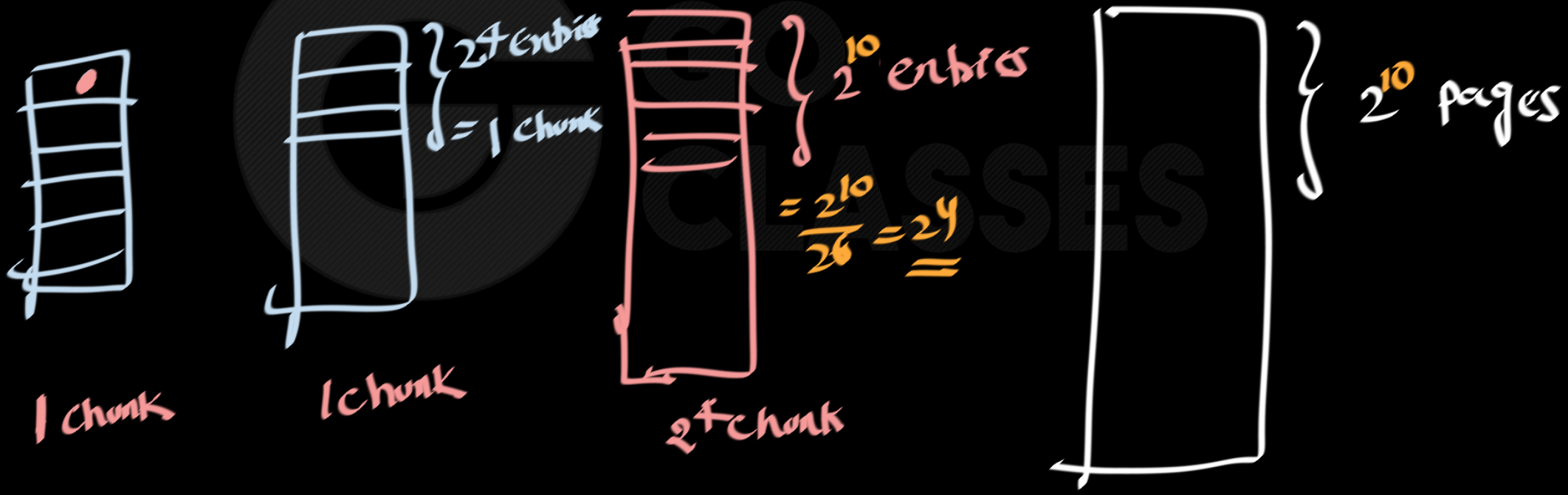
VAS

a) Page size = 2^8 = 256 B

b) 2. What is the size of a page table for a process that has 256K of memory starting at address 0?

# of useful pages = $\frac{2^8 \times 2^{10} \text{ B}}{2^{10}}$ = 2^{10} pages

10-bit	8-bit	6-bit	8 bit



1 chunk 1 chunk 16 chunks

2^{10} entries 2^{8} entries 16×2^{6} entries $= 2^{10}$ pages

$\Rightarrow \left(2^{10} + 2^{8} + 2^{10}\right)$ entries

$\Rightarrow \left(2^{10} + 2^{8} + 2^{10}\right) \times 2$ Bytes $= 2^{11} + 2^{9}$ Bytes

$2^{9}\left(2^{3}+1\right)$ Byte

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

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.

$$(2^{10} + 2^{8} + 16 \times 2^{6}) \times 2$$

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.

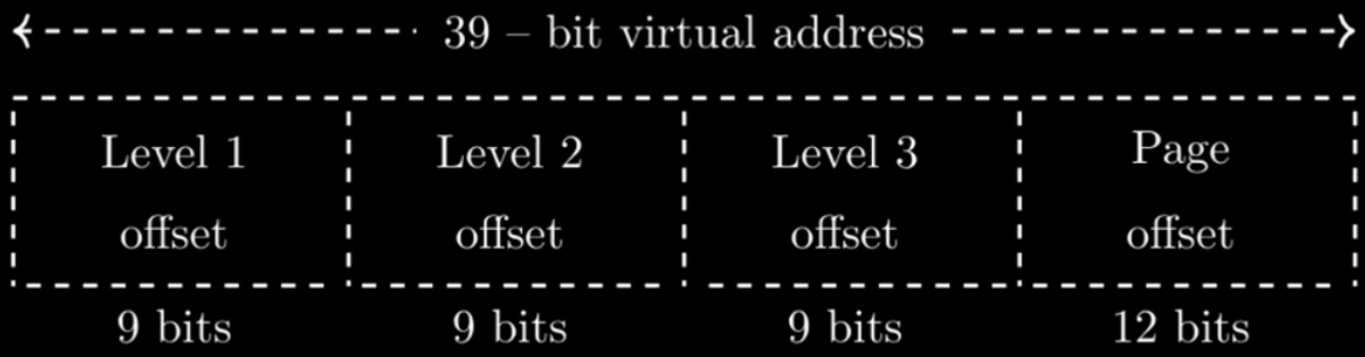
3. First, the stack, data and code segments are at addresses that require having 3 page tables entries active in the first level page table. For 64K, you need 256 pages, or 4 third-level page tables. For 600K, you need 2400 pages, or 38 third-level page tables and for 48K you need 192 pages or 3 third-level 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.

Question

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.

2^{31} Bytes

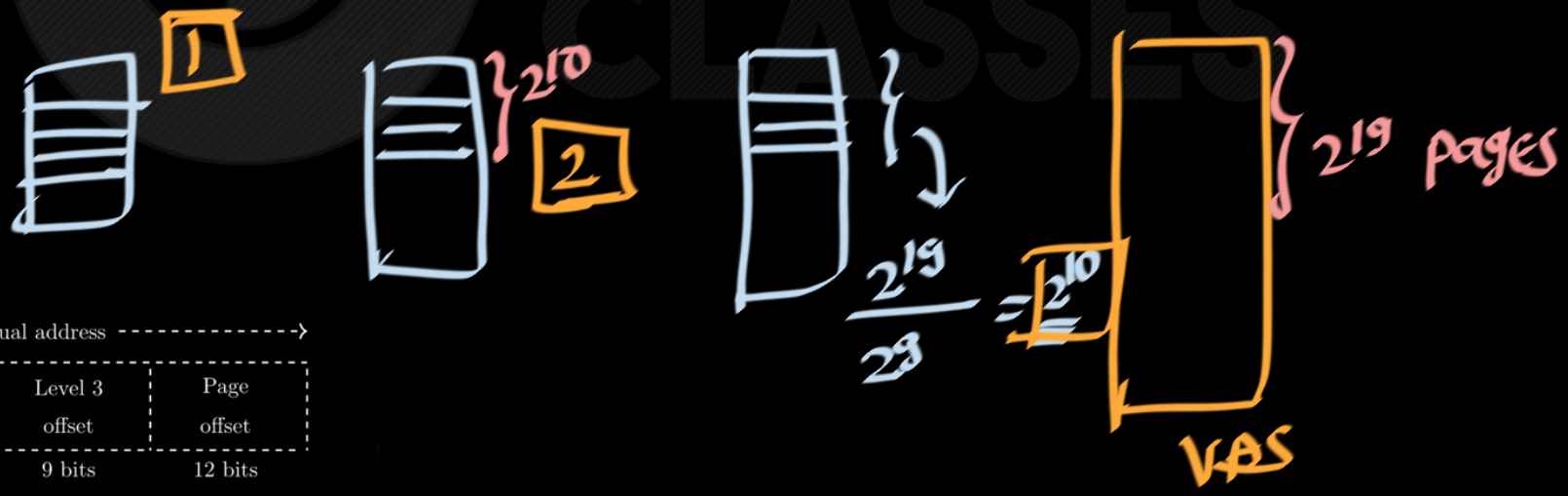
https://gateoverflow.in/357489/gate-cse-2021-set-2-question-48#

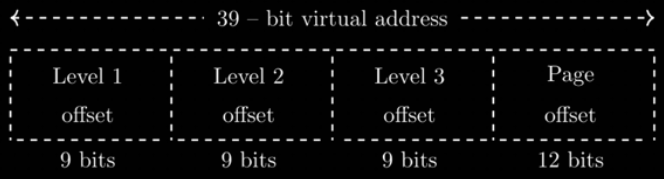
GO CLASSES

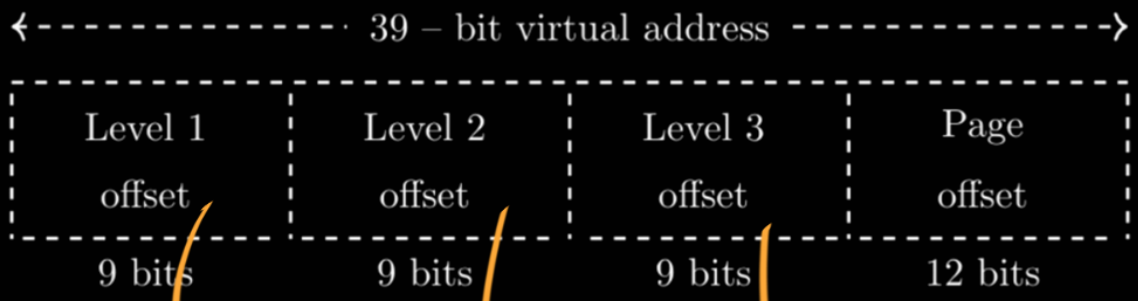
useful Bytes = 2^{31} B

Step1) useful pages = $\frac{2^{31}}{2^{12}} = 2^{19}$ pages

Step2 :







1 chunk → 2^9 entries

2 chunks → 2×2^9 entries

2^{10} chunks → $2^{10} \times 2^9$ entries

$$2^9(1+2+2^{10}) \text{ entries} = 2^3 \times 2^9 (3+2^{10}) \text{ B}$$

$$= 2^{12} (3+2^{10}) \text{ B}$$

$2^{12}\ (3+2^{10})\ B$

$\downarrow$

$2^{10} \times 2^2\ (3+2^{10})\ B$

$= \quad 4\ (3+2^{10})\ KB$

$1027 \times 4\ KB \quad = \quad 4108\ KB$

Answer: 4108

Question

Problem 1

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

PTE = 2B

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

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)?

Code Segment : 48 KB 0x1000000 → $\frac{48 \times 2^{10}}{2^8}$

Data Segment : 600 KB 0x80000000

Stack Segment : 64 KB 0xf0000000

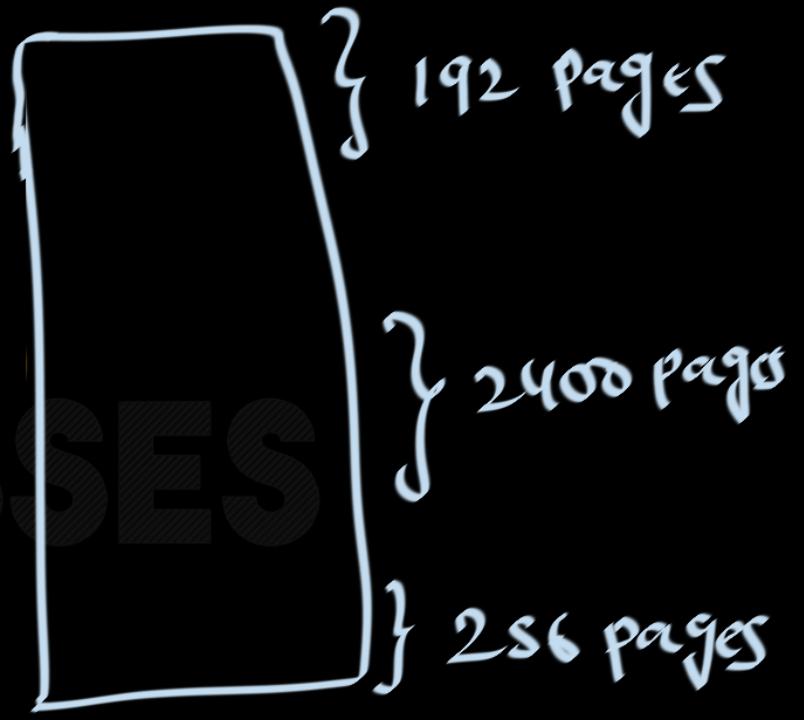
Code Segment : 48 KB 0x1000000 → $\frac{48 \times 2^{10}}{2^8}$ = 48 x 4 useful pages = 192

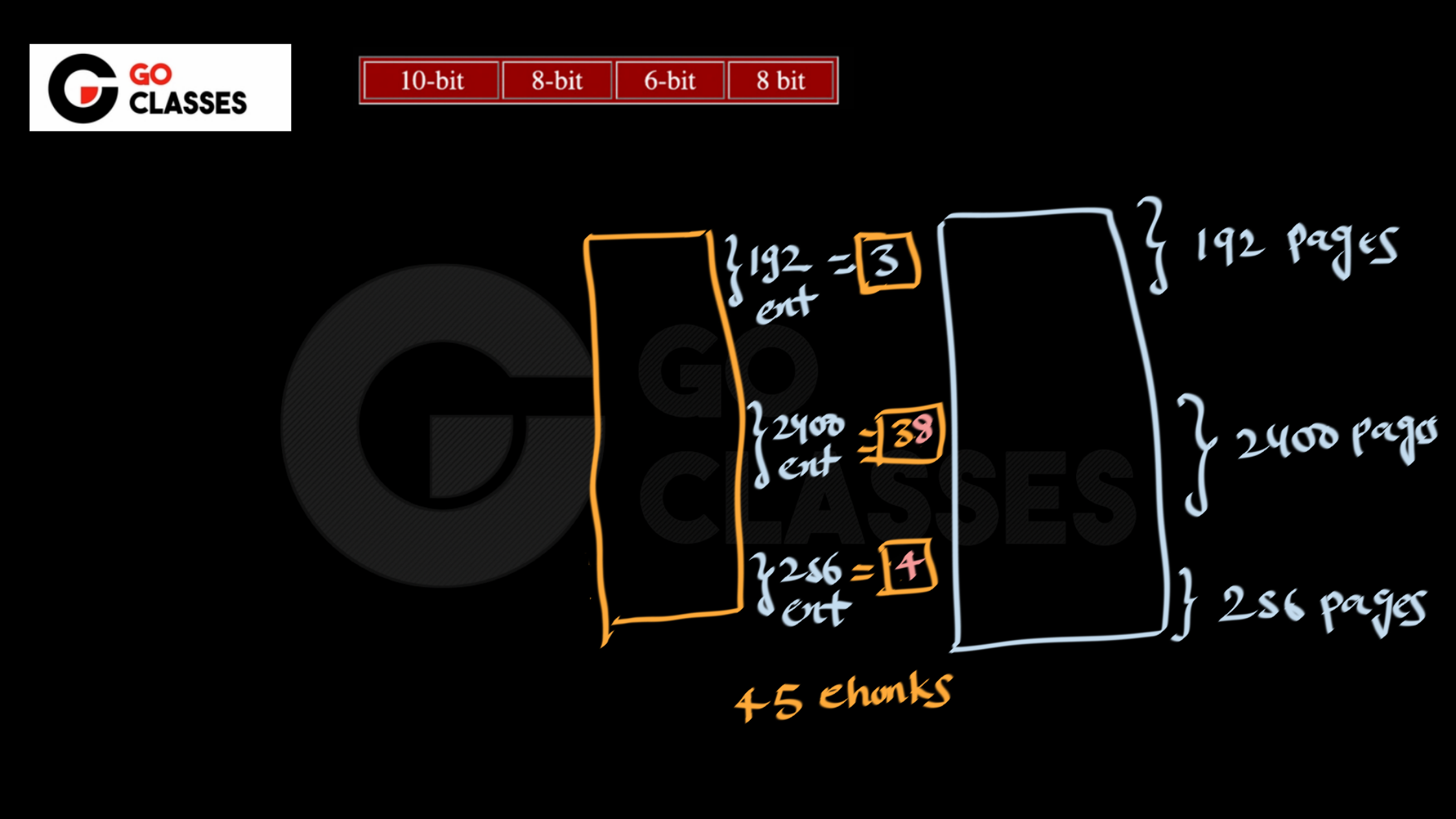
Data Segment : 600 KB 0x80000000 → 600 x 4 useful pages = 2400

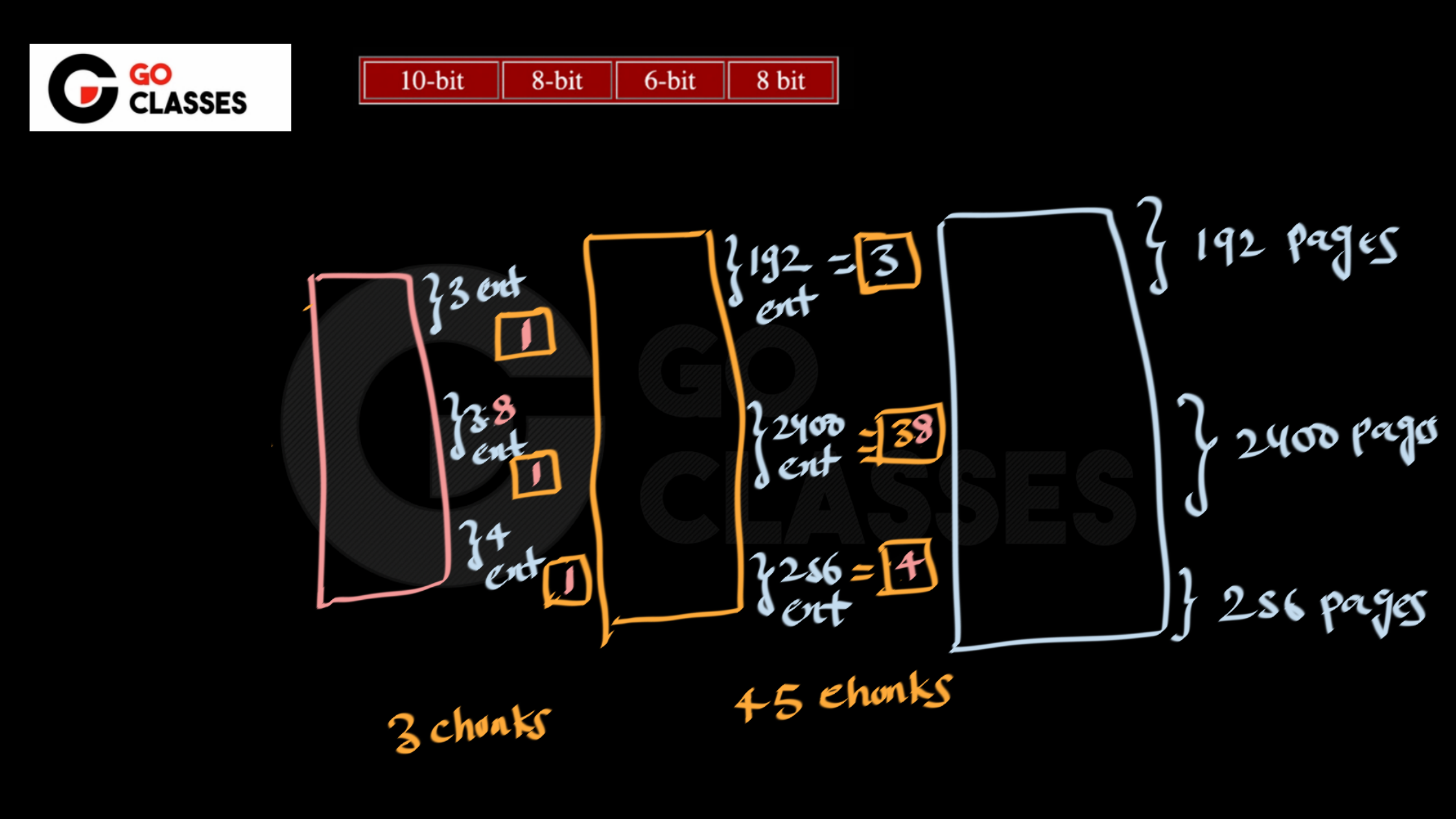
Stack Segment : 64 KB 0xf0000000 → 64 x 4 useful pages = 256

total useful pages = 2848 pages

10-bit	8-bit	6-bit	8 bit

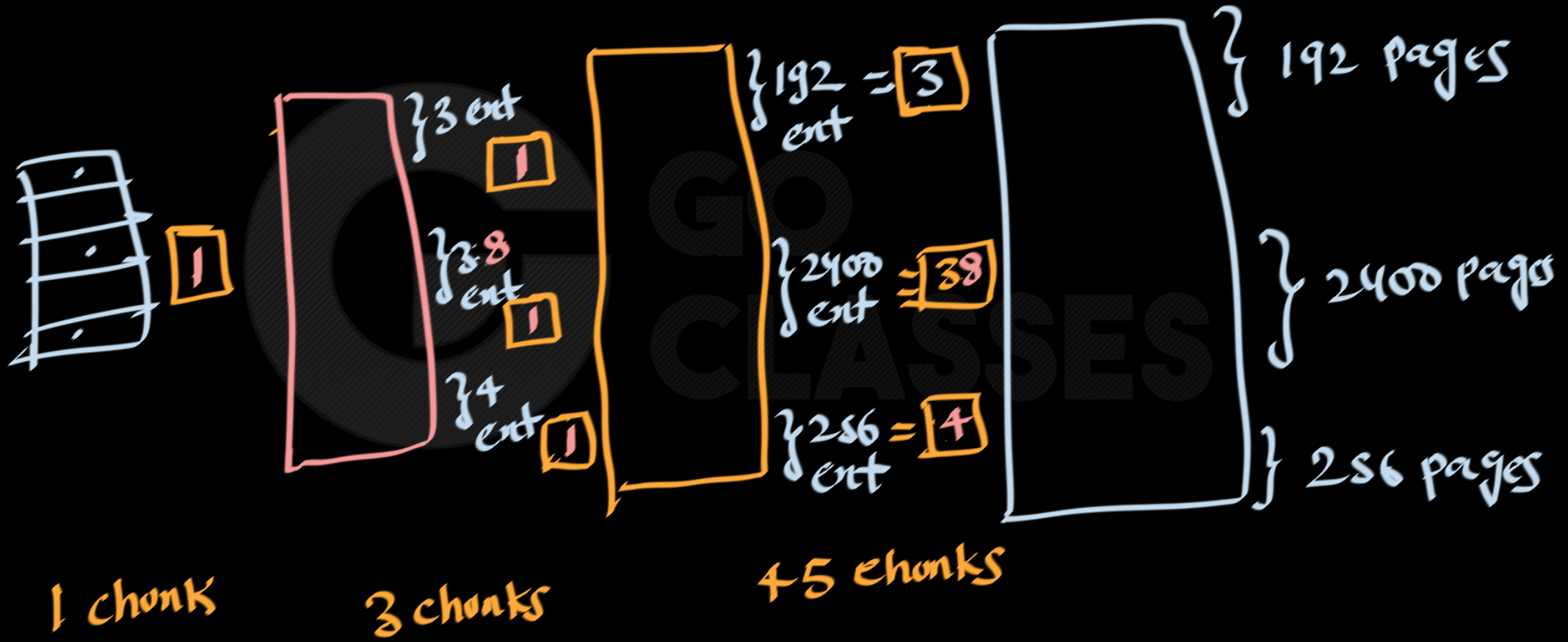



10-bit
8-bit
6-bit
8 bit
192 ent = 3
2400 ent = 38
256 ent = 4
45 chunks
192 pages
2400 pages
256 pages


10-bit
8-bit
6-bit
8 bit
3 ent
1
38 ent
1
4 ent
1
192 ent = 3
2400 ent = 38
256 ent = 4
192 pages
2400 pages
256 pages
3 chunks
45 chunks

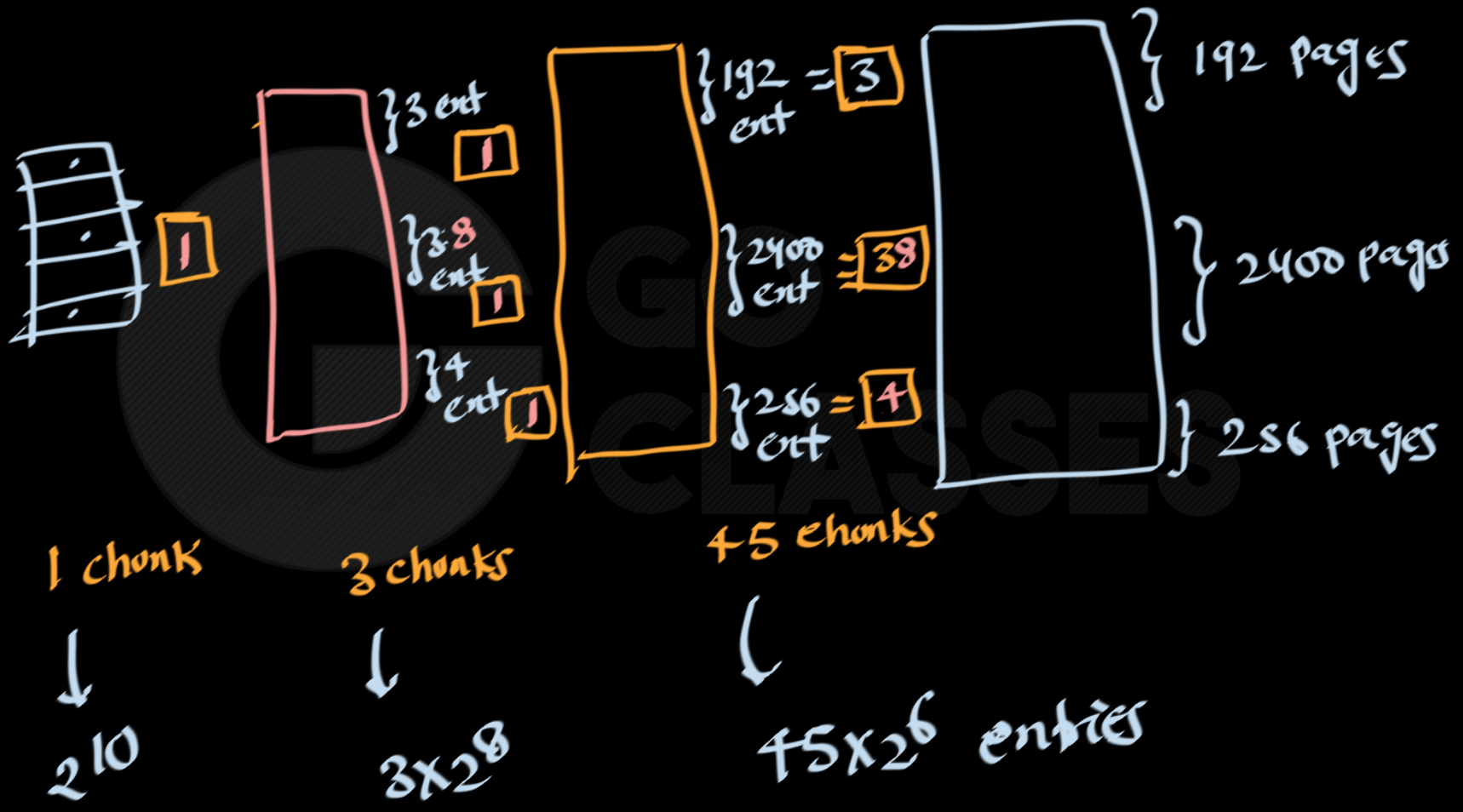
GO CLASSES

10-bit	8-bit	6-bit	8 bit



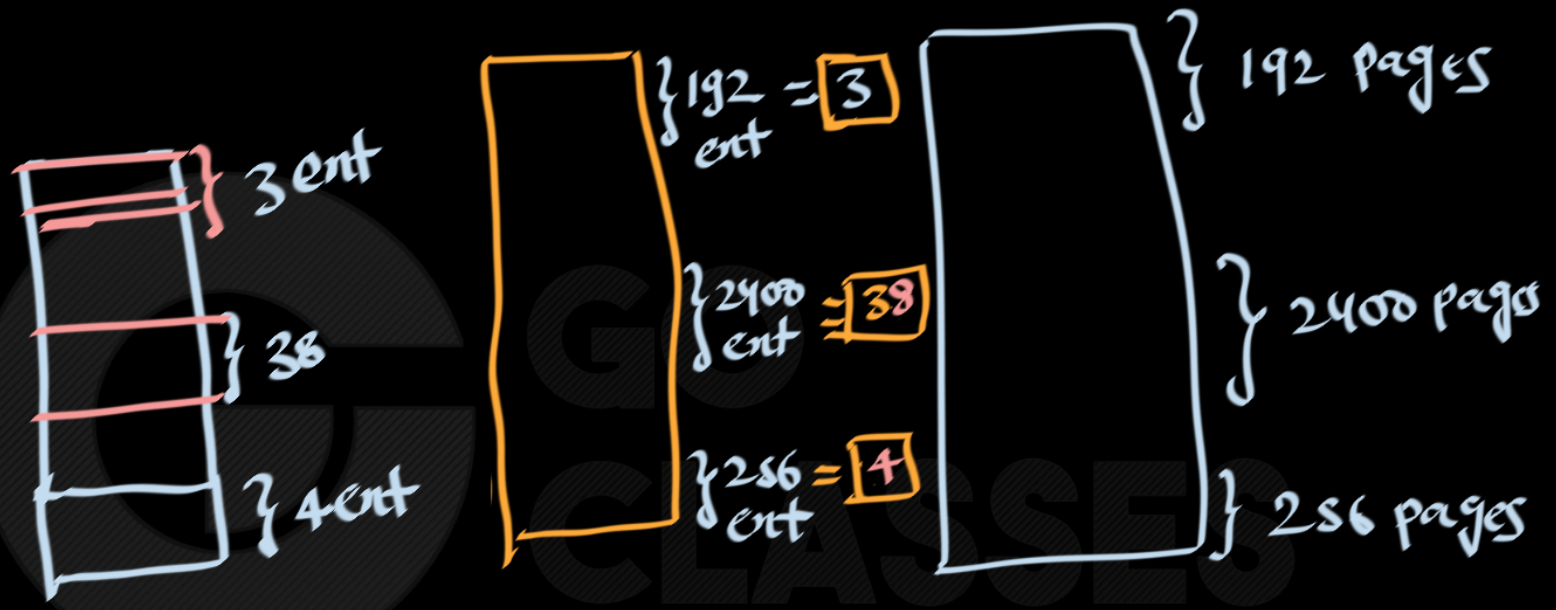
GO CLASSES

10-bit	8-bit	6-bit	8 bit



GO CLASSES

10-bit	8-bit	6-bit	8 bit



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.

Question

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

PTE = 4B

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

code : 2 pages

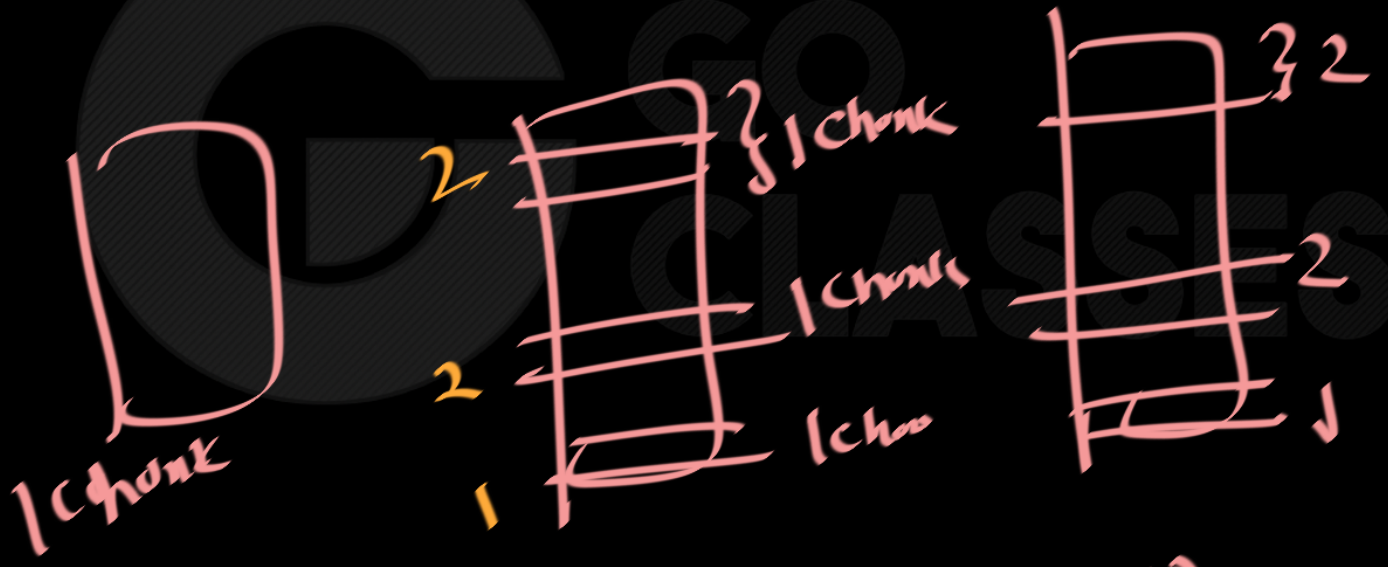
data : 2 pages

stack : 1 page

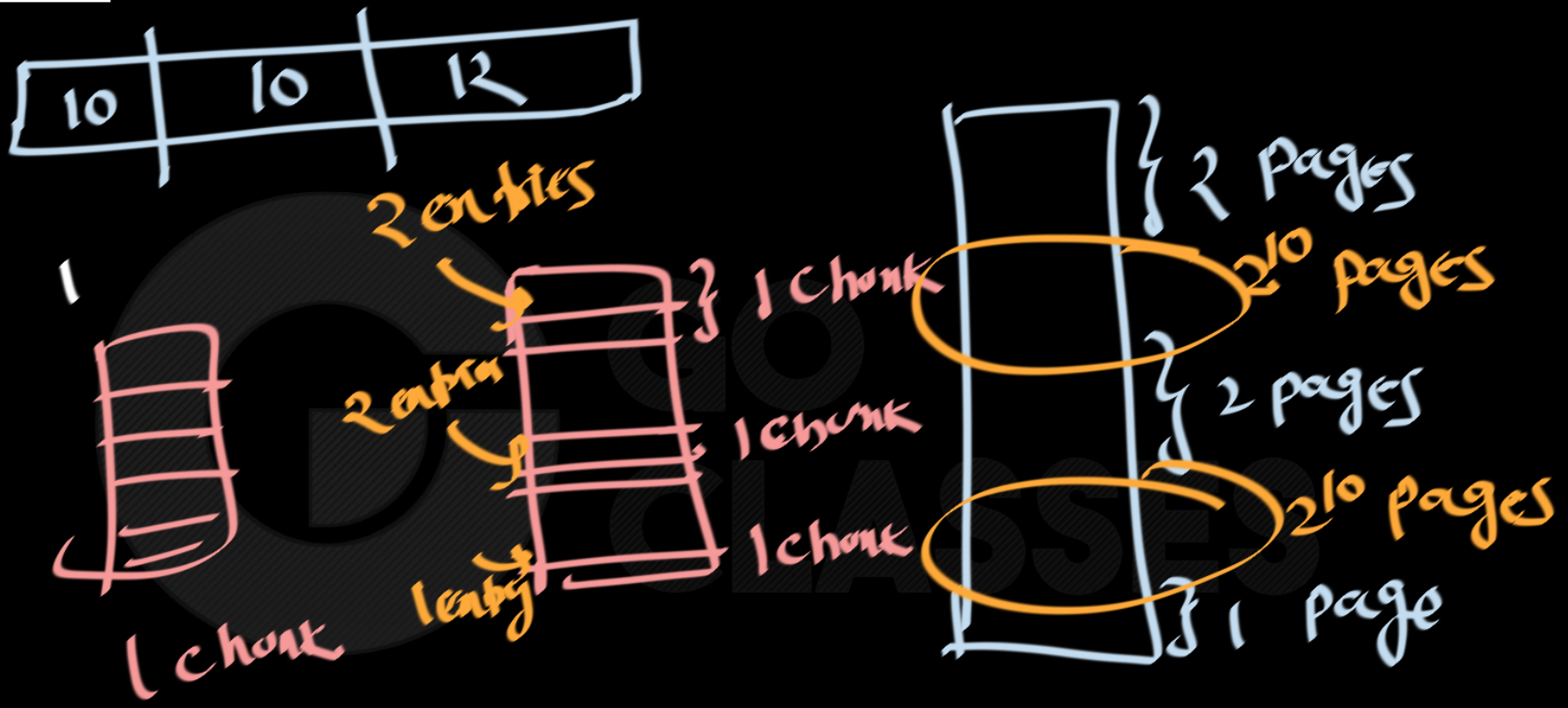
GO CLASSES

quickly

10	10	12

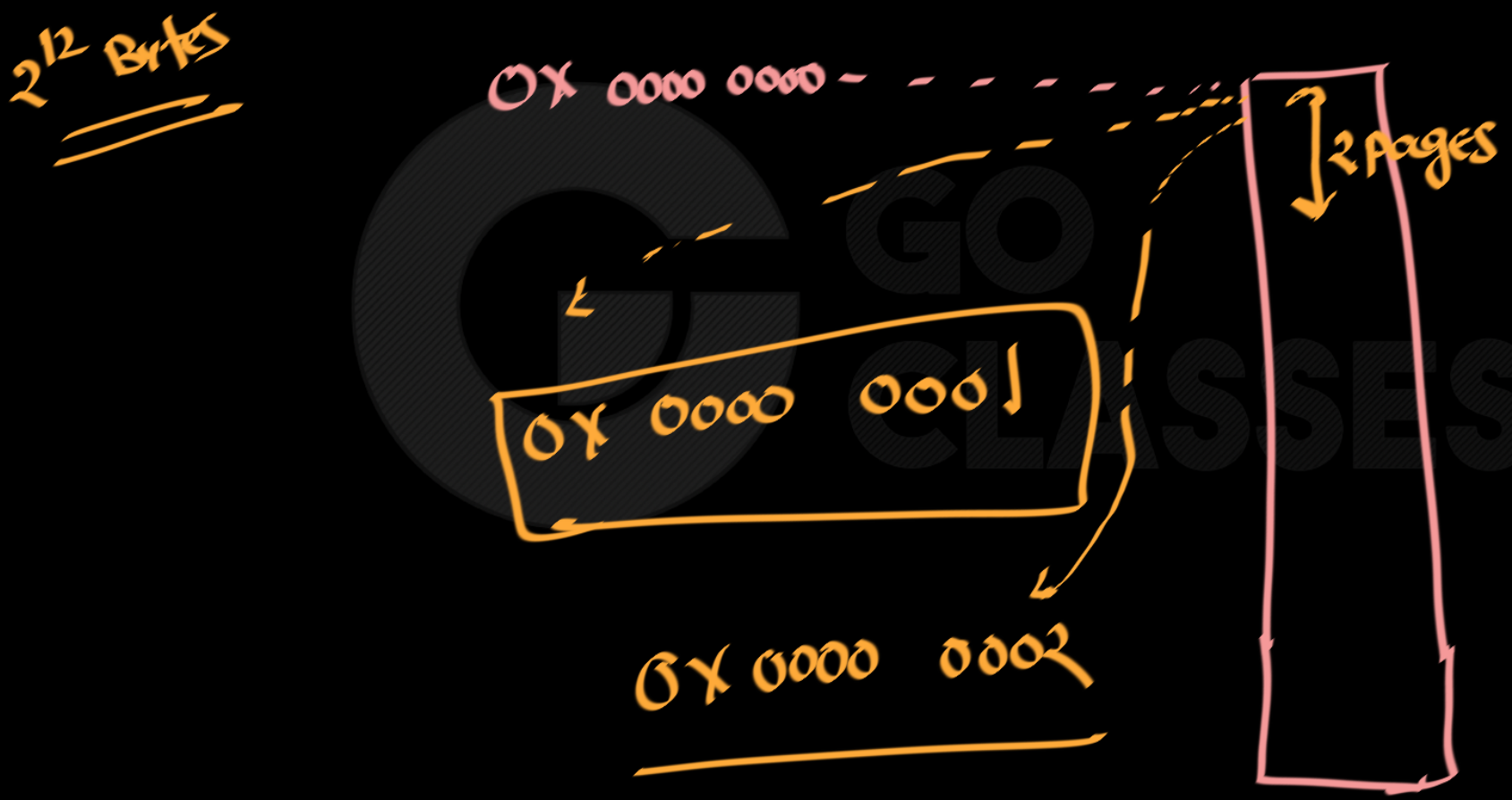


4 chunks × 210

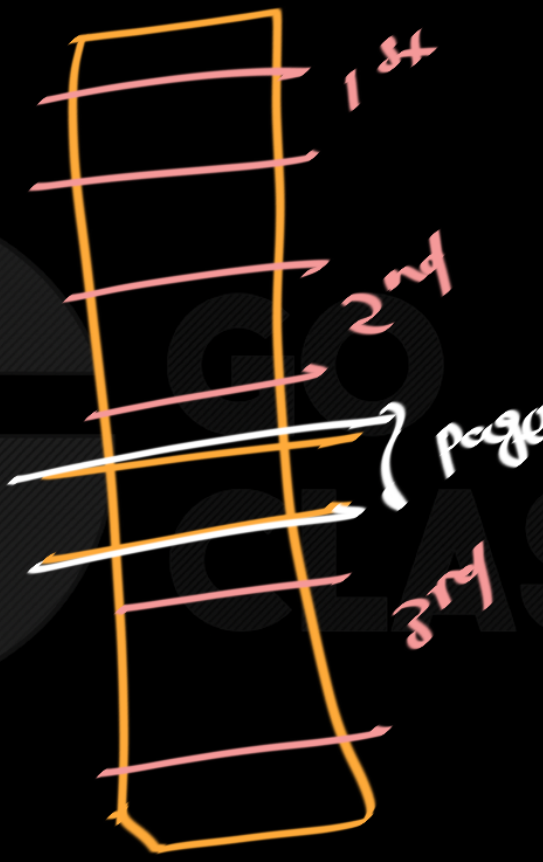


4 chunks $\Rightarrow$ 4×2^{10} entries $\Rightarrow$ $4 \times 2^{10} \times 2^2$ B
$= 16$ KB

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

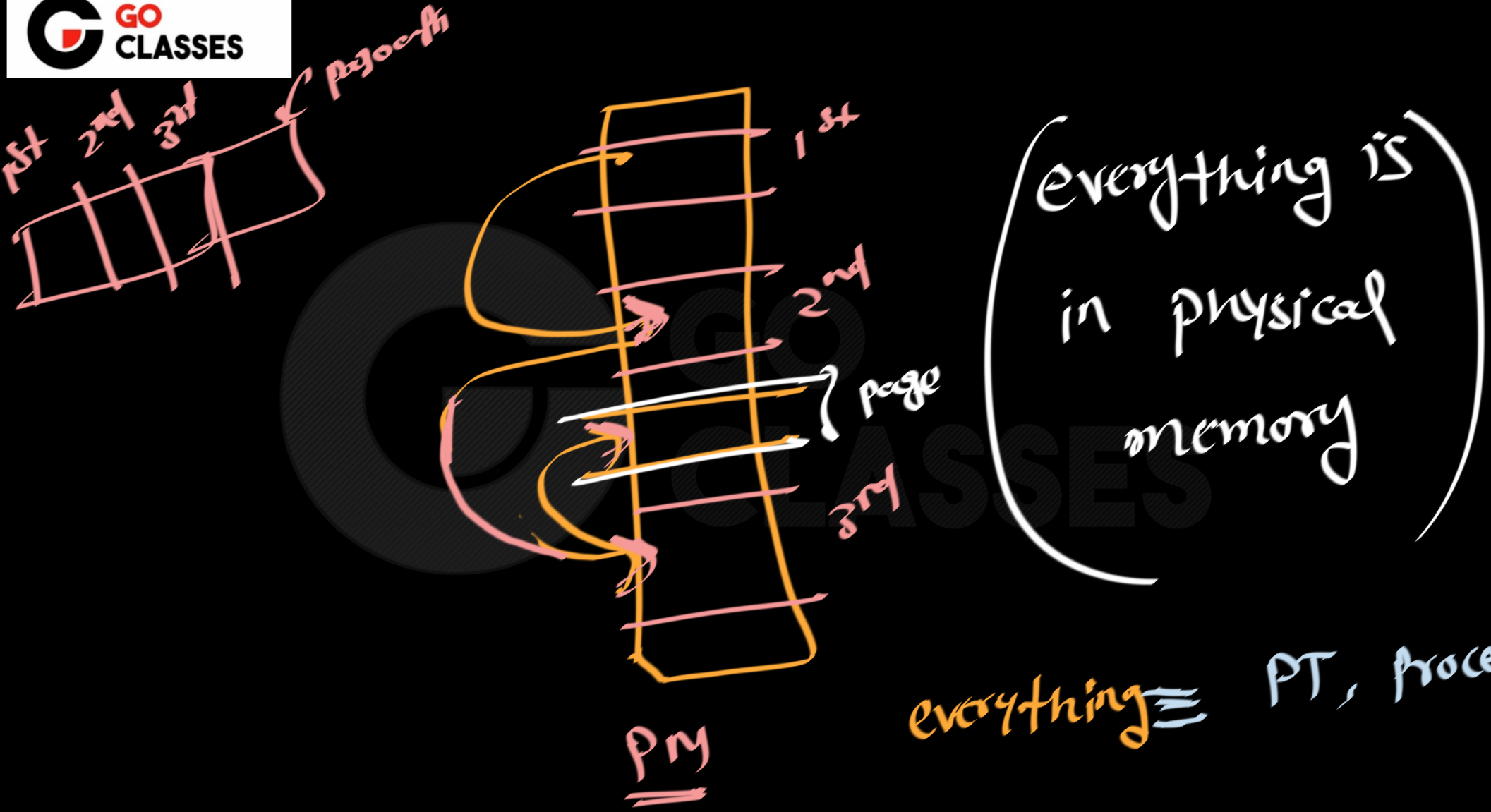


Suppose a process has only the following pages in its virtual address space: two contiguous code pages starting at virtual address $0x00000000$, two contiguous data pages starting at virtual address $0x00400000$, and a stack page starting at virtual address $0xFFFFF000$. The amount of memory required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



everything is in physical memory

everything ≡ PT, Process


1st
2nd
3rd
page
PM
everything is in physical memory
everything ≡ PT, Process

GO CLASSES

Optionally (Not required for GATE)

Memory alignment

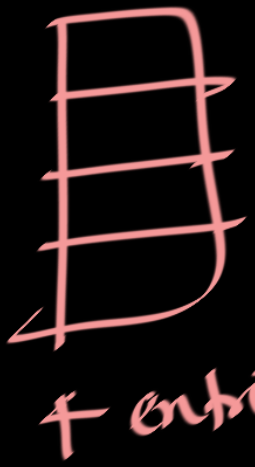
PageNo :- 5

ByteNo :- 5 X PageSize

GO CLASSES

22 bits

No. of Chonks = 4



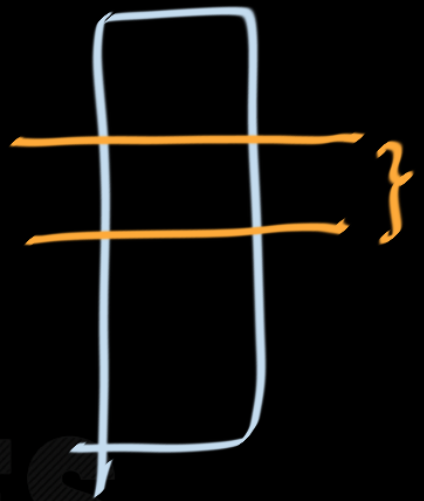
4 entries

2^{10} entries

Chonk size
$= 2^8$ entries
$= 2^{10}$ B

2^{20} entries

Chonk No. = 3



PM $= 2^{32}$ B

chonk size $= 2^{10}$ entries

GO CLASSES

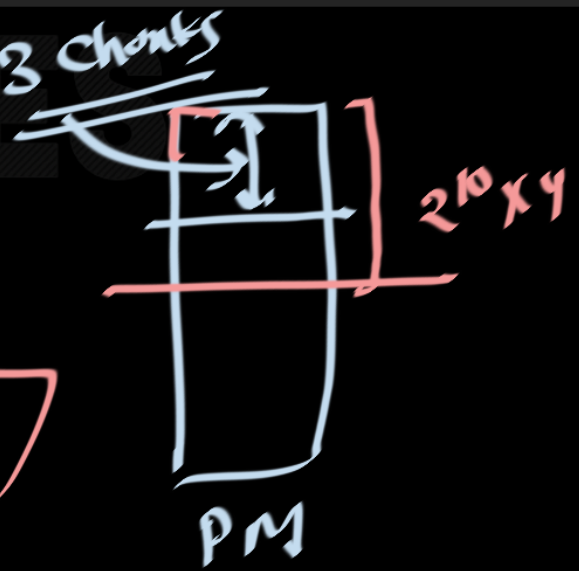
we want to store 4 chunks each of size 2^{10} B

PM = 2^{32} B

Min No. of bits required to address this chunk ?

00
01
10
11

0, 1, 2, 3



Sangeet Bhowmick to Everyone 10:02 PM

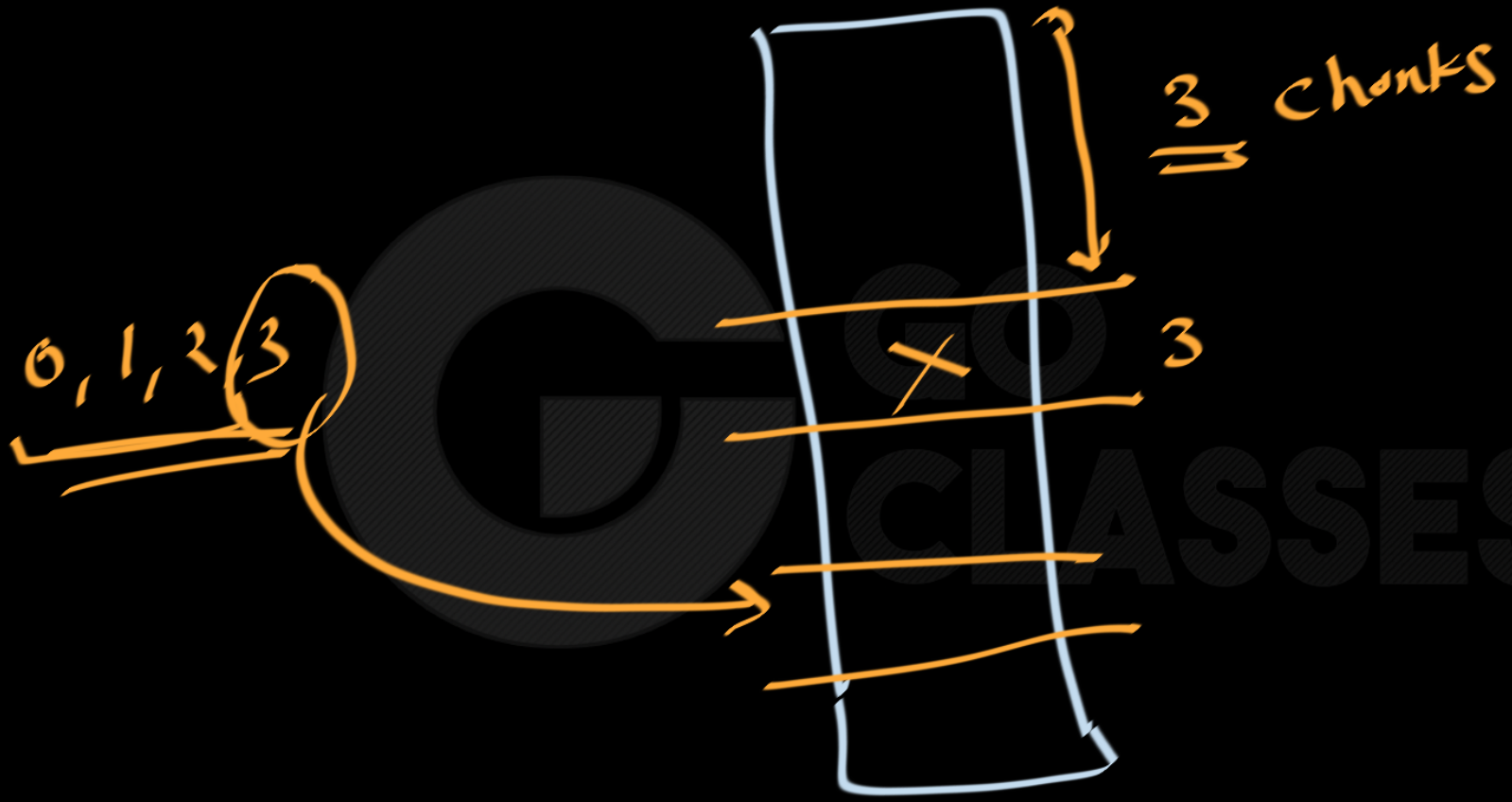
SB

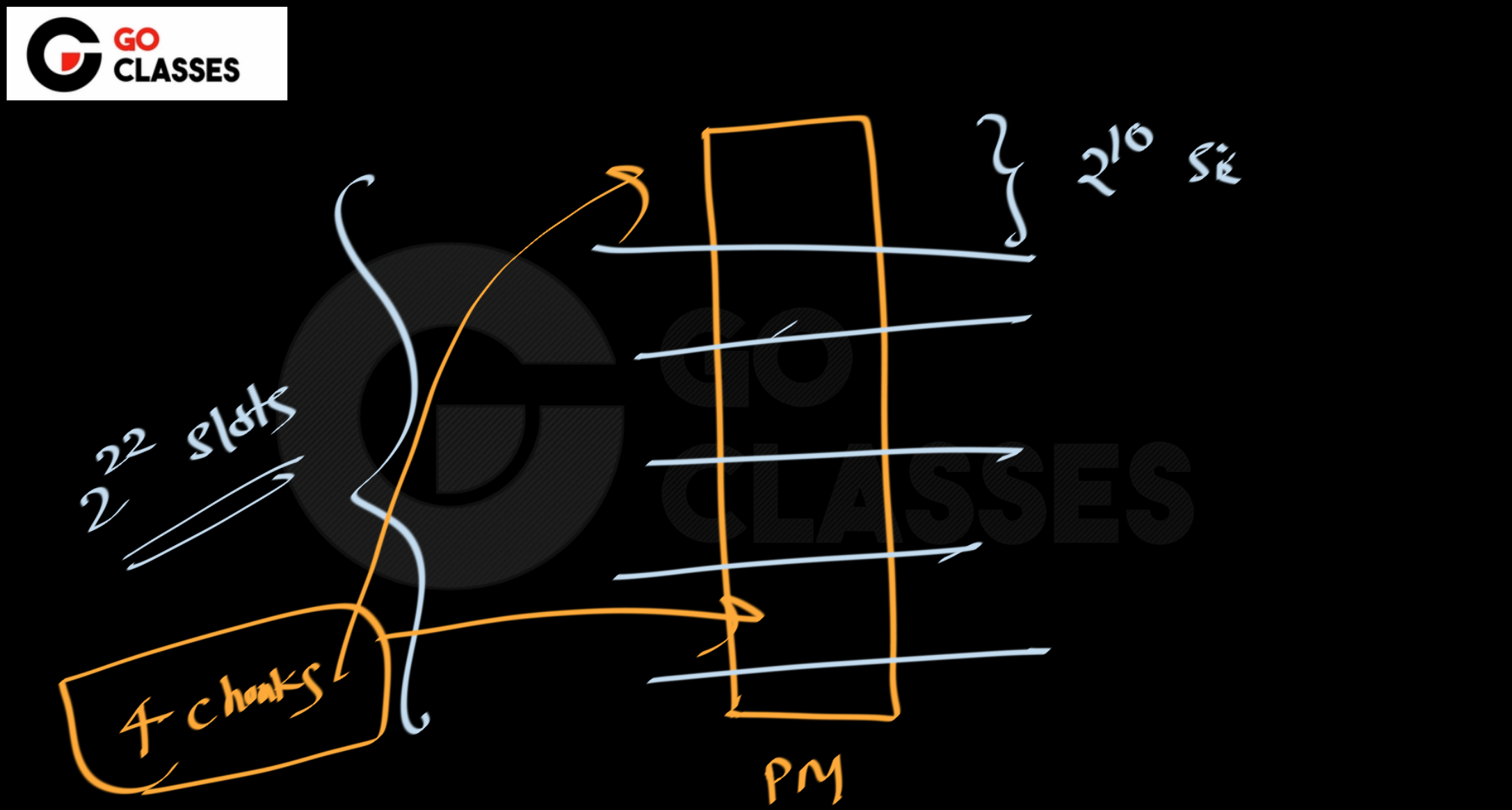
no

here we are addresing frame numbers

total size of PM

here we are directly addresing frame numbers which are of in quatities 2^22


3 chunks
0, 1, 2, 3
x
3


2^{10} Sz
2^{22} slots
4 chunks
PM

Chunk Size $= 2^{10}$

PA $=$ 32 bits

No. of bits required to address 1 chunk?

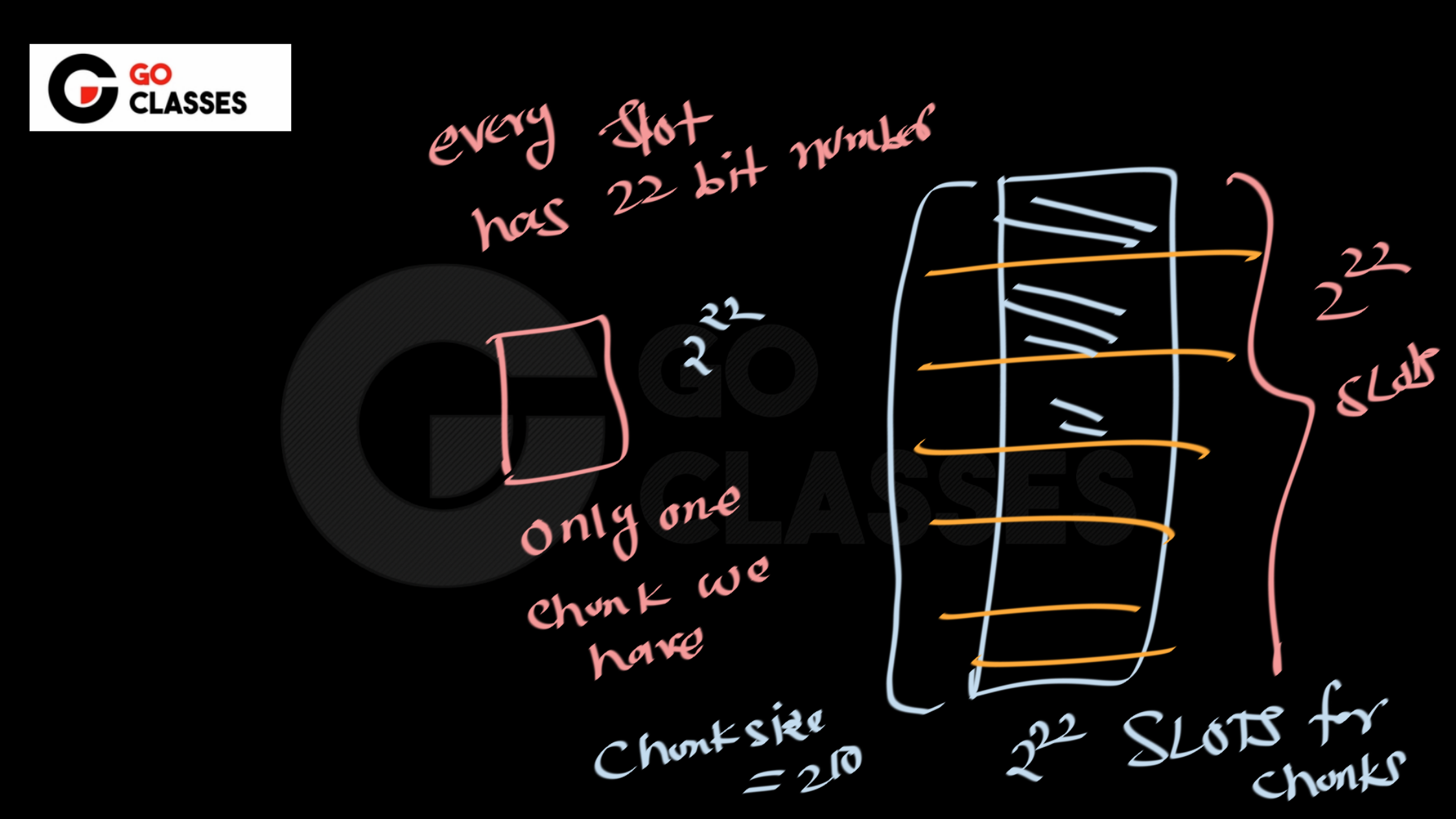
GO CLASSES

Chunk Size $= 2^{10}$

PA $=$ 32 bits

No. of bits required to address 1 chunk?

$$\#\text{SLOTS} = \frac{2^{32}}{2^{10}} = 2^{22} \text{ SLOTS}$$


every slot has 22 bit number
2^{22}
only one chunk we have
2^{22} slots
Chunk size = 210
2^{22} SLOTS for chunks

integer in memory

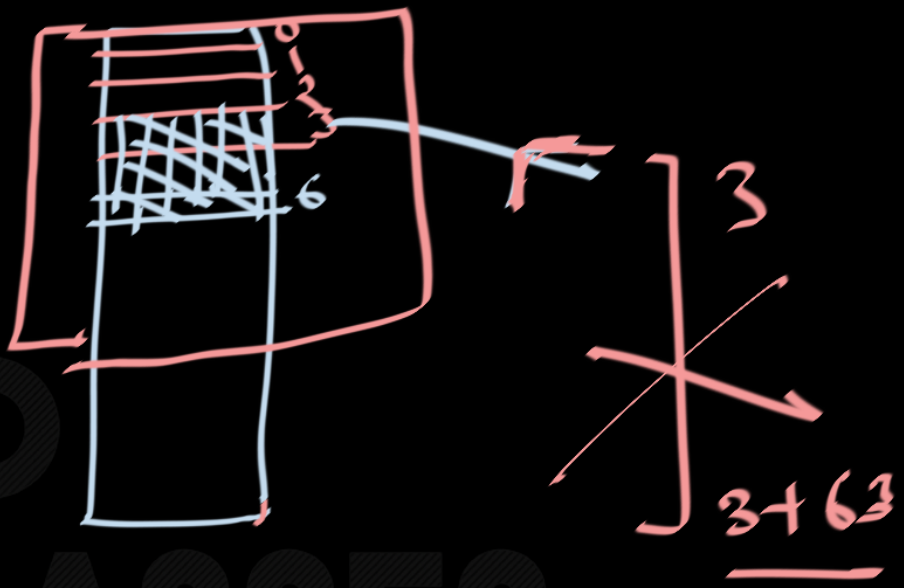
= 4 Bytes

Store int possible

Memory

integer in memory

= 4 Bytes



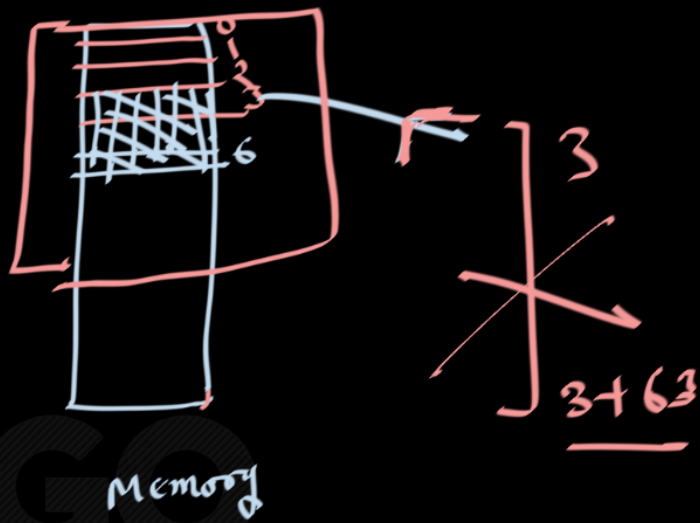
Hardware

data bus = 64 bits

if data bus = 64 bits then CPU feteches 64 bits at a time.

integer in memory

= 4 Bytes



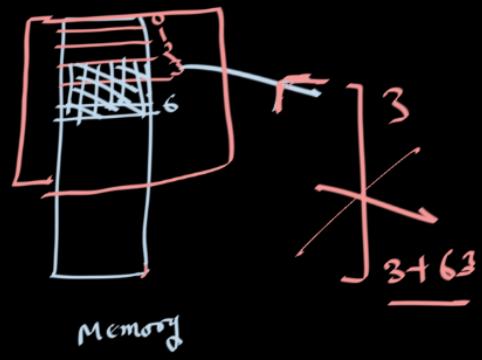
Hardware

data bus = 64 bits

if data bus = 64 bits then CPU feteches 64 bits at a time.

↳ if int is stored at 3 then CPU still give bytes from 0 to 63

integer in memory = 4 Bytes



Hardware

data bus = 64 bits

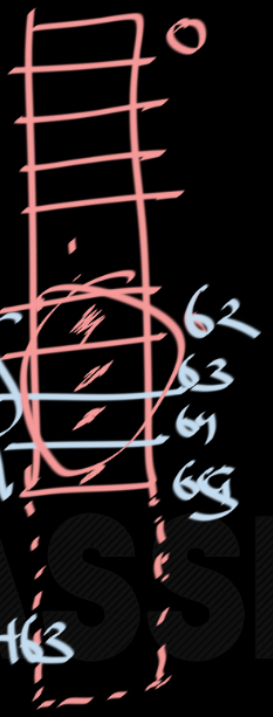
if data bus = 64 bits then CPU fetches 64 bits at a time.

if int is stored at 3 then CPU still fetch bytes from 0 to 63 → wrong its not recommended

1) extra work of left shift

GO CLASSES

give me bytes 62 ... 65



1) CPU first fetch 0-63 integer

left shift, keep 62, 63

2) Extra cycle

2) next cycle fetch 64 - - - bit 63

keep 64, 65

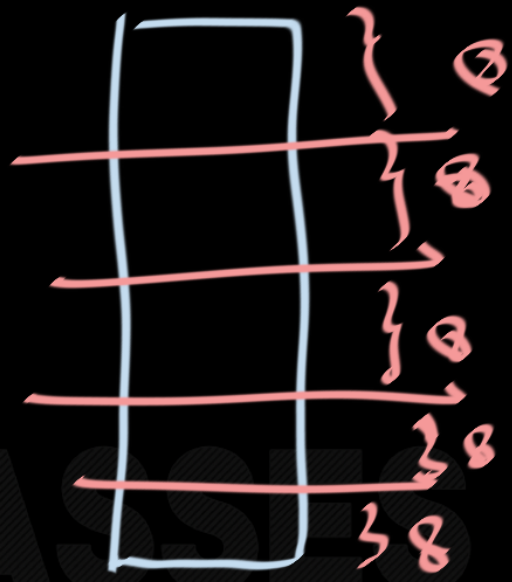
GO CLASSES

Suppose we want to store Character

= 1 Byte

multiple cycles are Not required in case of char

Data bus = 8 bytes



Anything of size 2^K should be stored at the start address multiple of 2^K

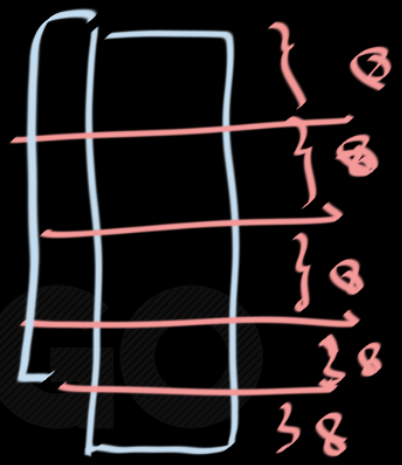
4 Bytes

0, 4, 8, 16

$2^4 = 16$

GO CLASSES

Data bus = 8 bytes



4

0

16

Anything of size 2K should be stored at the start address multiple of 2K

4 Bytes

0, 4, 8, 16

32

64

integer in memory

= 4 Bytes

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

Arjun

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

Question

GATE CSE 2009 | Question: 34

33

A multilevel page table is preferred in comparison to a single level page table for translating virtual address to physical address because

A. It reduces the memory access time to read or write a memory location.

B. It helps to reduce the size of page table needed to implement the virtual address space of a process

C. It is required by the translation lookaside buffer.

D. It helps to reduce the number of page faults in page replacement algorithms.

gatecse-2009 operating-system virtual-memory easy

https://gateoverflow.in/1320/gate-cse-2009-question-34

47

Best answer

Option - > (**B**)

A. It reduces the memory access time to read or write a memory location. -> No This is false. Actually because of multi level paging we increase no of memory accesses.

B. It helps to reduce the size of page table needed to implement the virtual address space of a process -> This is **true**, In case of big virtual memory page, size of Page table can also be too huge to fit in single Page. So we do multi level paging.

C. It is required by the translation lookaside buffer.-> Examiner was not being enough creative here, This is false & There is no relation. This option is just given for no reason !

D. It helps to reduce the number of page faults in page replacement algorithms.-> This is false, we might increase no of page faults. (Due to second / third level page not in memory here !) So this is false.

answered Nov 25, 2015 • edited Jan 23 by **JAINchiNMay**

edit flag hide comment Follow

Pip Box Delete with Reason Wrong Useful

share this

Akash Kanase

Question

Problem 7 (20 points): *(Note: You are allowed to use a calculator to solve parts of the question.)* A byte-addressable machine implements virtual memory with a 4 MB virtual address space using a three-level page table tree system similar to the x86 model. Each page table/page directory occupies exactly one frame of physical memory. All page tables live in physical address space.

The breakdown of the virtual address bits is as follows:

L1	L2	L3	Offset
5	5	5	7

Part a (2 points): How many PTEs can fit in a frame?

Part b (2 points): What is the PTE size in bytes?

Parts c, d and e

A user process runs on this machine that requires exactly 343,240 contiguous bytes of virtual memory and does not perform any dynamic memory allocation. The OS allocates at least this amount of memory to the process at the time it starts.

https://users.ece.utexas.edu/~patt/22f.460n/handouts/f20_final_sol.pdf

Problem 7 (20 points): *(Note: You are allowed to use a calculator to solve parts of the question.)* A byte-addressable machine implements virtual memory with a 4 MB virtual address space using a three-level page table tree system similar to the x86 model. Each page table/page directory occupies exactly one frame of physical memory. All page tables live in physical address space.

The breakdown of the virtual address bits is as follows:

L1	L2	L3	Offset
5	5	5	7

Part a (2 points): How many PTEs can fit in a frame?

2^5 = 32

Part b (2 points): What is the PTE size in bytes?

2^7 / 2^5 = 4

Parts c, d and e

A user process runs on this machine that requires exactly 343,240 contiguous bytes of virtual memory and does not perform any dynamic memory allocation. The OS allocates at least this amount of memory to the process at the time it starts.

https://users.ece.utexas.edu/~patt/22f.460n/handouts/f20_final_sol.pdf

GO CLASSES

Question

7 (continued).

(e) (1 mark) A process uses a contiguous 2^{20} bytes of memory for its address space. How many valid entries will the page table have?

(f) (10 marks) What is the page number for each of the following virtual addresses? If the process described in (e) uses virtual addresses $[0, 2^{20})$, which of these addresses will be valid?

(i) 0xF00D 5555

(ii) 0xEA5E 0ACE

(iii) 0xC0DE C0DE

(iv) 0x0000 1234

(v) 0xEEEE EEEE

https://student.cs.uwaterloo.ca/~cs350/common/old-exams/W20-midterm-sol.pdf

7 (continued).

(e) (1 mark) A process uses a contiguous 2^{20} bytes of memory for its address space. How many valid entries will the page table have?

$2^{20}/2^{16} = 2^4$

(f) (10 marks) What is the page number for each of the following virtual addresses? If the process described in (e) uses virtual addresses $[0, 2^{20})$, which of these addresses will be valid?

(i) 0xF00D 5555

(ii) 0xEA5E 0ACE

(iii) 0xC0DE C0DE

(iv) 0x0000 1234

(v) 0xEEEE EEEE

---

i 0xF00D, exception

ii 0xEA5E, exception

iii 0xCODE, exception

iv 0x0000, valid

v 0xEEEE, exception

Question

Consider a virtual memory system that uses 2-level paging. The page size in this system is 256 (2^8) bytes. Each individual page table fits exactly into one memory frame, and the size of each page table entry (PTE) is 8 bytes.

a. (3 marks)
What is the maximum size (in bytes) of a virtual address space in this system?

b. (3 marks)
Suppose that there is a process with a virtual address space of the maximum size. How many bytes of memory are occupied by the page tables for this process?

c. (3 marks)
Suppose that there is a process with a virtual address space of size 10240 ($10 \cdot 2^{10}$) bytes. Also suppose that the entire address space is in memory. How many *valid* PTEs will exist in the page tables for this process?

https://student.cs.uwaterloo.ca/~cs350/common/old-exams/S07mid2sol.pdf

Consider a virtual memory system that uses 2-level paging. The page size in this system is 256 (2^8) bytes. Each individual page table fits exactly into one memory frame, and the size of each page table entry (PTE) is 8 bytes.

a. (3 marks)

What is the maximum size (in bytes) of a virtual address space in this system?

There are $2^8/2^3 = 2^5$ PTEs per page table. There can be up to 2^5 page tables in the second level, each with 2^5 PTEs, for at total of 2^{10} PTEs. Each refers to a page of size 2^8 bytes, so the maximum total address space size is $2^{10}2^8 = 2^{18}$ bytes (256 Kbytes).

b. (3 marks)

Suppose that there is a process with a virtual address space of the maximum size. How many bytes of memory are occupied by the page tables for this process?

There are 2^5 page tables in the second level plus one at the first level, each of size 2^8 bytes. Total size is $2^8(2^5 + 1) = 2^{13} + 2^8 = 8448$ bytes.

c. (3 marks)

Suppose that there is a process with a virtual address space of size 10240 ($10 \cdot 2^{10}$) bytes. Also suppose that the entire address space is in memory. How many *valid* PTEs will exist in the page tables for this process?

This process will occupy $10(2^{10})/2^8 = 40$ pages. This will require 40 PTEs in two page tables at the second level, and 2 PTEs at the first level, for a total of 42 PTEs.

Question

10. [9 points] Now consider the x86-64 architecture. Below we are asking about the *physical pages consumed by a process, including the page tables themselves*. As you answer the question, assume that any allocated memory consumes physical pages in RAM; that is, there is no swapping or demand paging. Note that it may be helpful for you to draw pictures (but you don't have to).

As a reminder, the x86-64 imposes a multi-level page table structure: pages are 4KB, each page table entry is 8 bytes, and each individual page table (a node in the "tree") occupies one page. Thus, each page table holds $\frac{4KB}{8B} = 512 = 2^9$ entries. Recall that the structure is four levels; each level is indexed by 9 bits of the virtual address.

What is the minimum number of physical pages consumed by a process that allocates 12KB (for example, 1 page each for code, stack, and data)?

What is the minimum number of physical pages consumed by a process that makes $2^9 + 1$ allocations of size 4KB each? You can leave your answer in terms of powers of 2, and sums thereof.

What is the minimum number of physical pages consumed by a process that makes $2^{18} + 1$ allocations of size 4KB each? You can leave your answer in terms of powers of 2, and sums thereof.

10. [9 points] Now consider the x86-64 architecture. Below we are asking about the *physical pages consumed by a process, including the page tables themselves*. As you answer the question, assume that any allocated memory consumes physical pages in RAM; that is, there is no swapping or demand paging. Note that it may be helpful for you to draw pictures (but you don't have to).

As a reminder, the x86-64 imposes a multi-level page table structure: pages are 4KB, each page table entry is 8 bytes, and each individual page table (a node in the "tree") occupies one page. Thus, each page table holds $\frac{4KB}{8B} = 512 = 2^9$ entries. Recall that the structure is four levels; each level is indexed by 9 bits of the virtual address.

What is the minimum number of physical pages consumed by a process that allocates 12KB (for example, 1 page each for code, stack, and data)?

7 pages: 3 for the memory it's allocated, and 4 to implement the paging structure.

What is the minimum number of physical pages consumed by a process that makes $2^9 + 1$ allocations of size 4KB each? You can leave your answer in terms of powers of 2, and sums thereof.

$2^9 + 6$, or 518 pages. Each L4 page table holds 2^9 mappings. So we need two L4 page tables. The total is $1 + 1 + 1 + 2$ for the page structures plus the $2^9 + 1$ pages themselves.

What is the minimum number of physical pages consumed by a process that makes $2^{18} + 1$ allocations of size 4KB each? You can leave your answer in terms of powers of 2, and sums thereof.

$2^{18} + 2^9 + 6$. Each L3 page table points indirectly to 2^{18} last-level page entries (each L3 page table has 2^9 entries, each of which points to an L4 page table with 2^9 entries). Thus, the question requires two L3 page tables. The first L3 page table points to 2^9 L4 page tables; the second points to one L4 page table, for a total of $2^9 + 1$ L4 page tables. Thus, the total is: $1 + 1 + 2 + 2^9 + 1$ for the page structures plus $2^{18} + 1$ for the pages themselves.

https://cs.nyu.edu/~mwalfish/classes/20sp/exams/midterm-soln-2019.pdf

Question

Consider a virtual memory system that uses multi-level paging for address translation. Virtual addresses and physical addresses are 64 bits long. The page size is 1 MB (2^{20} bytes). The size of a page table entry is 16 (2^4) bytes. Each individual page table, at each level, must fit in a single frame.

a. How many bits of each virtual address are needed to represent the page offset?

b. What is the maximum number of entries in an individual page table?

c. What is the minimum number of levels of page tables that will be required for virtual-to-physical translation in this system? Show your work for possible partial credit.

d. Suppose that a particular process uses only 128 MB (2^{27} bytes) of virtual memory, with a virtual address range from 0 to $2^{27} - 1$. How many individual page tables, *at each level*, will be required to translate this process' address space? Your answer should be of this form:

```
Level 1:  X page tables
Level 2:  Y page tables
...
Level N:  Z page tables
```

Show your work for possible partial credit.

https://student.cs.uwaterloo.ca/~cs350/common/old-exams/S09mid2sol.pdf

a. How many bits of each virtual address are needed to represent the page offset?

20 bits (because the page size is 2^{20} bytes).

b. What is the maximum number of entries in an individual page table?

$$\frac{2^{20}}{2^4} = 2^{16} \text{entries}$$

c. What is the minimum number of levels of page tables that will be required for virtual-to-physical translation in this system? Show your work for possible partial credit.

20 of 64 bits are used for the offset, leaving 44 bits for page numbers. Each level of page tables will require a 16 bit page number (because 2^{16} is the maximum size of each page table). Thus, a total of 3 levels of page tables will be required.

d. Suppose that a particular process uses only 128 MB (2^{27} bytes) of virtual memory, with a virtual address range from 0 to $2^{27} - 1$. How many individual page tables, *at each level*, will be required to translate this process' address space? Your answer should be of this form:

```
Level 1:  X page tables
Level 2:  Y page tables
...
Level N:  Z page tables
```

Show your work for possible partial credit.

The address space for this process has only $2^{27}/2^{20} = 2^7$ pages. A single level 3 page table can have up to 2^{16} entries, so only a single page table will be needed at level 3. Only a single page table entry will be needed at levels 1 and 2 to index the level 3 page table.

```
Level 1: 1 page table
Level 2: 1 page table
Level 3: 1 page table
```

Question

Assume the following: a 32-bit virtual address space, with a 1KB page size.

Finally, given the following memory allocations, write down both

(a) how much memory our multi-level page table consumes and

(b) how much memory a linear page table consumes:

(a) Code is located at address 0 and there are 100 4-byte instructions. The heap starts at page 1 and uses 3 total pages. The stack starts at the other end of the address space, grows backward, and uses 3 total pages.

- Multi-level page table size?
- Linear page table size?

(b) Code is located at address 0 and there are 100 4-byte instructions. The heap starts at page 1 and uses 1000 total pages. The stack starts at the other end, grows backwards, and uses 1000 total pages.

- Multi-level page table size?
- Linear page table size?

(c) The entire address space (every page) is used by the process.

- Multi-level page table size?
- Linear page table size?

https://pages.cs.wisc.edu/~remzi/Classes/537/Spring2009/OldExams/09-spring-mid-answers.pdf

(a)
- Multi-level page table size?
  Each chunk of the page table covers 256 consecutive pages; hence the code pages and heap in this example fit onto the same chunk of the page table. The stack, at the other end of the address space, needs one too. Hence 2 pages (1KB each) plus the page directory (64KB), or 66KB.
- Linear page table size?
  The linear page table is always the same regardless of the usage of pages in the address space. Hence, 16 MB.

(b)
- Multi-level page table size?
  Here we need four pages of the page table to translate the first 1001 pages of the address space (four pages of the page table covers 1024 pages) and another four pages to translate the last 1000 pages of the stack. Hence, 8 pages (1KB each) plus the page directory (64KB) gets us to 72KB.
- Linear page table size?
  16 MB.

(c)
- Multi-level page table size?
  If all pages are used, you get the full linear page table (16 MB) plus the page directory (64KB).
- Linear page table size?
  16 MB.

Question

VII. (6 + 1 bonus) Consider a system with a page size of 2^{14}, and a two level page table with an outer page table size of 2^{12} entries. Assume that each page table entry is 8 bytes. Assume the size of a page is 1Kbyte. The system has a physical address space of 1 Gbytes (2^{30} bytes).

(a) (1 point) How many bits are there in the physical address?

(b) (1 point) What is the maximum total size of the page table?

(c) (1 point) What is the minimum size of the page table that must be in memory for a (small) program to run?

(d) (1.5 point) Given your answer in (b), how many bits are there in the virtual address for each process?

(e) (1.5 + 1 bonus) Consider a situation where each process on average needs 1/10 of its pages to be resident (i.e., part of its working set). How many processes can you run before thrashing occurs (Hint: don't' forget the size of the page tables)?

https://www.cs.ucr.edu/~amazl001/teaching/cs153/Summ21/slides/key-final-w15.pdf

(a) (1 point) How many bits are there in the physical address?

30 bits

(b) (1 point) What is the maximum total size of the page table?

We have 2^12 PTEs in the top level page table. That means we have 2^12 pages of the lower level of the page table (one pointed to by each entry in the top level page table).

Each page is 2^14 bytes, so it can hold 2^14/8 PTEs since each PTE is 8 bytes. So, each page of the lower level of the page table has 2^11 PTEs. Since we have 2^12 lower level pages (we know this from the size of the top level page table), the lower level page table has 2^12*2^11 PTEs or 2^23 PTEs.

The total size of the page table = size of top level + size of bottom level

Top level size= Number of PTEs * size of PTE = 2^12*8 = 2^15 byts.

Bottom level size = number of pages * size of page = 2^12*2^14 = 2^26 bytes

Total size = 2^26 +2^15.

(c) (1 point) What is the minimum size of the page table that must be in memory for a (small) program to run?

Recall that a multi-level page table can be smaller if we are not using the whole address space since pages of the lower level of the page table do not have to be allocated if they are empty. The minimum size it can be is one page in the top level page table, and one page in the bottom level page table. This would allow us to address 2^14/8 pages (the PTEs held in the one page of the lower level page table). If you answered that the whole top level page table must be preallocated (2^12*8 or 2 pages) and one page of the lower level page table, that is fine too.

(d) (1.5 point) Given your answer in (b), how many bits are there in the virtual address for each process?

Each one of PTEs in the bottom level holds the translation for a data page. Since the total size of the lower level page table is 2^23 PTEs, the VPN is 23 bits. Another way to see this is to see that the first 12 bits indicate which entry in the top level we use to find the right page of the page table, and the next 11 bits to find the right entry inside that page (every page of the bottom level holds 2^11 PTEs). The VPO is 14 bits (since the size of the page is 2^14). So, the virtual address space is 2^37 in size.

(e) (1.5 + 1 bonus) Consider a situation where each process on average needs 1/10 of its pages to be resident (i.e., part of its working set). How many processes can you run before thrashing occurs (Hint: don't' forget the size of the page tables)?

1/10 of 2^37 is bigger than 2^33. Since the physical memory size is 2^30, that means we thrash even with one process. Not a very good system!

Question

Information for questions 22–24
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

Question 22 [2 pt]: (see above) What is the maximum possible space (in bytes) that the page tables for a single process can occupy? You may leave your answer as unsimplified arithmetic expression.

Question 23 [2 pt]: (see above) If the page containing address `0x100` and containing the address `0x10000` are valid for a process, and no other pages are valid, how much spcae (in bytes) do the page tables for this process occupy? You may leave your answer as unsimplified arithmetic expression.

https://www.cs.virginia.edu/~cr4bd/3330/F2020/files/f2018key3E.pdf

Information for questions 22–24

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

Question 22 [2 pt]: (see above) What is the maximum possible space (in bytes) that the page tables for a single process can occupy? You may leave your answer as unsimplified arithmetic expression.

Question 23 [2 pt]: (see above) If the page containing address `0x100` and containing the address `0x10000` are valid for a process, and no other pages are valid, how much spcae (in bytes) do the page tables for this process occupy? You may leave your answer as unsimplified arithmetic expression.

Information for questions 22–24

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

Question 22 [2 pt]: (see above) What is the maximum possible space (in bytes) that the page tables for a single process can occupy? You may leave your answer as unsimplified arithmetic expression.

Answer: 32 + 16 * 256 + 16 * 128 * 256 (half for forgetting to multiply by page table entry size)

Question 23 [2 pt]: (see above) If the page containing address `0x100` and containing the address `0x10000` are valid for a process, and no other pages are valid, how much spcae (in bytes) do the page tables for this process occupy? You may leave your answer as unsimplified arithmetic expression.

Answer: 32 + 256 + 256 * 2

Question

10. (3 points) Consider a system with:

- 6-level page tables
- 1KB page tables at each level
- 4 byte page table entries

How much page table space would be required to allow a process to access two distinct pages, where the virtual addresses only differ in their most significant bit?

https://www.cs.virginia.edu/~cr4bd/3330/F2020/files/f2019key3.pdf

Solution: 11KB (1 first-level table, 2 tables at each other level)

Question

b. (18 points total) Address Translation:

i) (5 points) Consider a machine with a physical memory of 8 GB, a page size of 8 KB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page? Be explicit in your explanation.

ii) (6 points) How much physical memory is needed for a process with three pages of virtual memory (for example, one code, one data, and one stack page)?

https://inst.eecs.berkeley.edu/~cs162/sp20/static/exams/fa13-mt1-solutions.pdf

b. (18 points total) Address Translation:

i) (5 points) Consider a machine with a physical memory of 8 GB, a page size of 8 KB, and a page table entry size of 4 bytes. How many levels of page tables would be required to map a 46-bit virtual address space if every page table fits into a single page? Be explicit in your explanation.

Since each PTE is 4 bytes and each page contains 8KB, then a one-page page table would point to 2048 or 2^{11} pages, addressing a total of $2^{11} * 2^{13} = 2^{24}$ bytes. Continuing this process:

Depth	*Address Space*
1	2^{24} *bytes*
2	2^{35} *bytes*
3	2^{46} *bytes*

We deducted 3 pts if your calculation uses the physical memory size to determine the bits required for virtual page number, 3pts for the right idea but wrong interpretation (leading to the wrong number of levels), and 3 to 4 pts for a major errors depending on whether you answer was on the right track or not.

https://inst.eecs.berkeley.edu/~cs162/sp20/static/exams/fa13-mt1-solutions.pdf

ii) (6 points) How much physical memory is needed for a process with three pages of virtual memory (for example, one code, one data, and one stack page)?

Six physical pages totaling 48KB are needed: one for the first-level page table, one for one page of the second-level page table, one for one page of the third-level page table, and three for the process' three pages.

*Note that the second- and third-level page tables do not need 16MB (2^{11} * 8KB) each, because the top-level (and second-level) page tables enable you to only have the next level page table pages for those pages that are part of the process's virtual address space.*

For partially correct answers, we subtracted three points for not including the page table memory, 2 pts for having more than one top level page table, 1 pt for not including the data pages, 1 pt for minor errors and 1 pt if your answer was not consistent with prior parts.

Question

Consider a memory management system that translates 16 bit virtual addresses to 24 bit physical addresses. These addresses are byte addresses; the memory consists of 16 bit words. The system uses a two-level page table, with a 4-bit first level page number, a 4-bit second level page number, and an 8-bit offset within the page.

a. What is the maximum number of physical memory frames that can be addressed by this memory management system?

b. Explain the circumstances under which the pages tables in this system will take up less memory than a single-level page table with an 8-bit page number and an 8-bit offset. It may be useful to give an example, but this is not required for full credit.

http://web.eecs.utk.edu/~mbeck/classes/cs560/oldtests/t2/2007/final-answers.pdf

GO CLASSES

Consider a memory management system that translates 16 bit virtual addresses to 24 bit physical addresses. These addresses are byte addresses; the memory consists of 16 bit words. The system uses a two-level page table, with a 4-bit first level page number, a 4-bit second level page number, and an 8-bit offset within the page.

a. What is the maximum number of physical memory frames that can be addressed by this memory management system?

2^4 first level page table entries x 2^4 second level page table entries = 2^8 frames

b. Explain the circumstances under which the pages tables in this system will take up less memory than a single-level page table with an 8-bit page number and an 8-bit offset. It may be useful to give an example, but this is not required for full credit.

If most second page tables are empty, they need not be represented by this system, and thus take up no memory.

The space taken is $16(1 + n)$ where n is the number of non-empty second level page tables. This system takes less memory when $16(1 + n) < 64$, ie when $n < 3$.

http://web.eecs.utk.edu/~mbeck/classes/cs560/oldtests/t2/2007/final-answers.pdf