Lecture: 25

Question

Here we have a trace of **logical** address references in a segmented system. The system has two segments in each address space, and uses the first bit of the virtual address to differentiate which segment a reference is in. Segment 0 holds code and a heap (and therefore grows in the positive direction); Segment 1 holds a stack (and therefore grows in the negative direction). Please translate the following references, or mark a segmentation violation if the address reference is out of bounds.

The address space size is 16 bytes (tiny!).

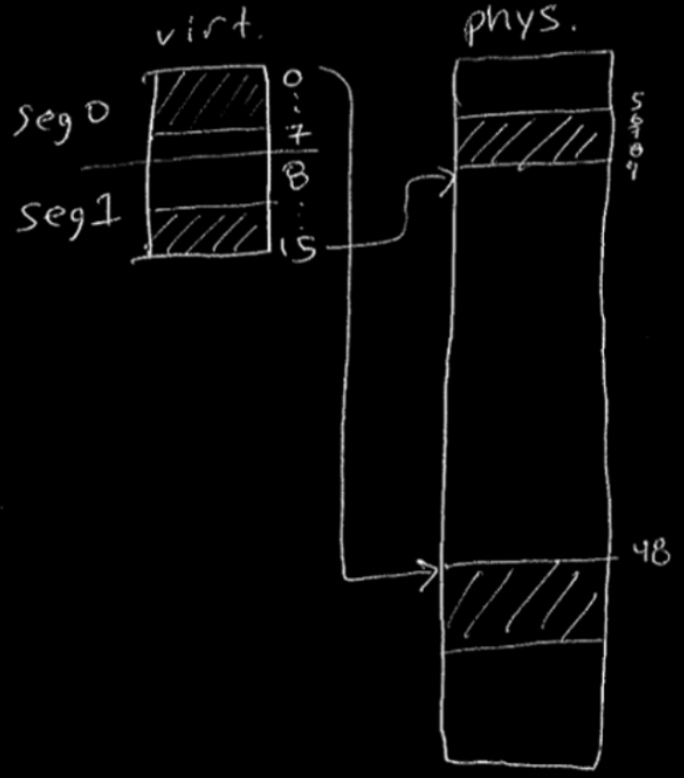
The physical memory is only 64 bytes (also tiny!).

Here is some segment register information:

```
  Segment 0 base  (grows positive) : 48
  Segment 0 limit                  : 7

  Segment 1 base  (grows negative) : 9
  Segment 1 limit                  : 4
```

And here is the **logical** Address Trace:

```
   14

    6

   10
```



GO CLASSES

```
Here is some segment register information:
  Segment 0 base  (grows positive) : 48
  Segment 0 limit                  : 7

  Segment 1 base  (grows negative) : 8
  Segment 1 limit                  : 4
```

initial sizes

And here is the logical Address Trace:

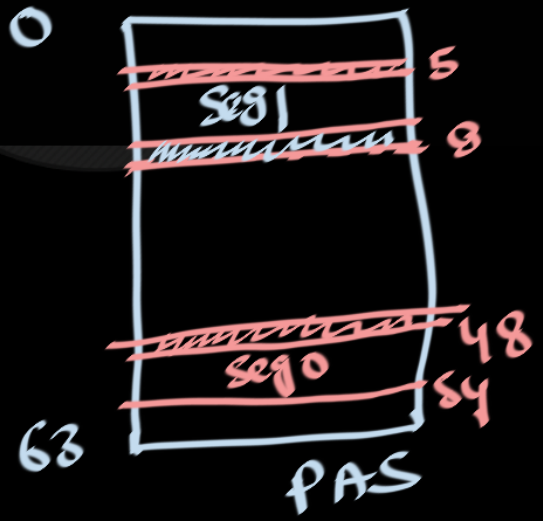
14 → 7

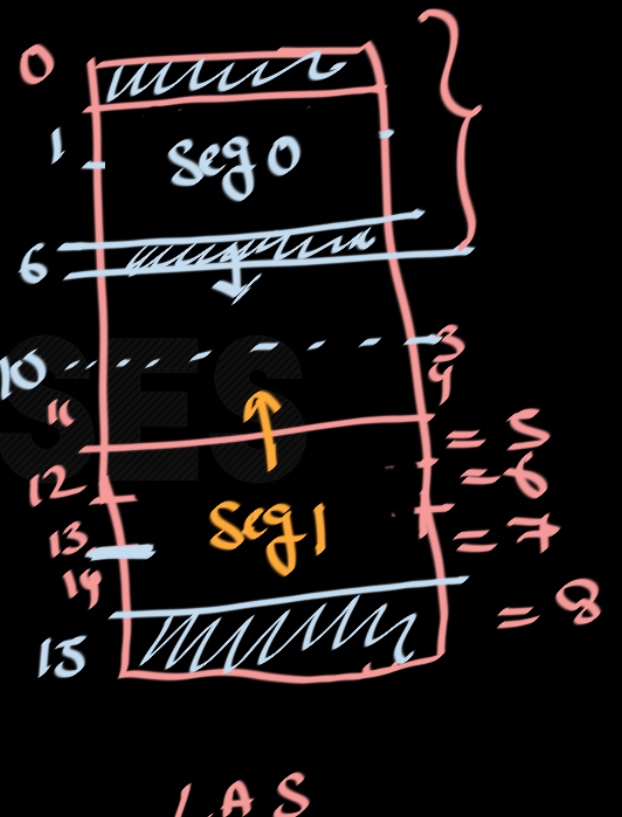
6 → 54

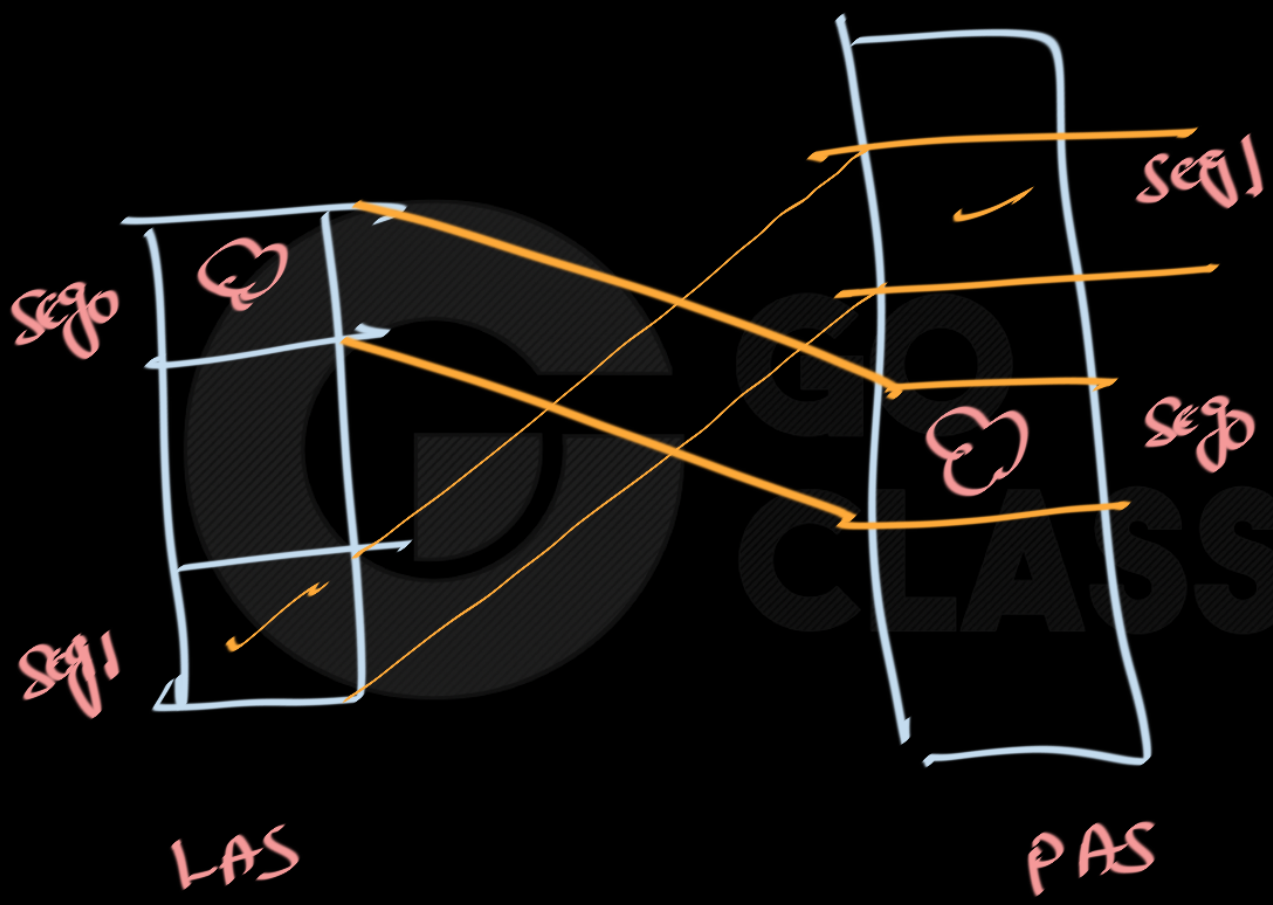
10 → 3

1010

assuming under seg1

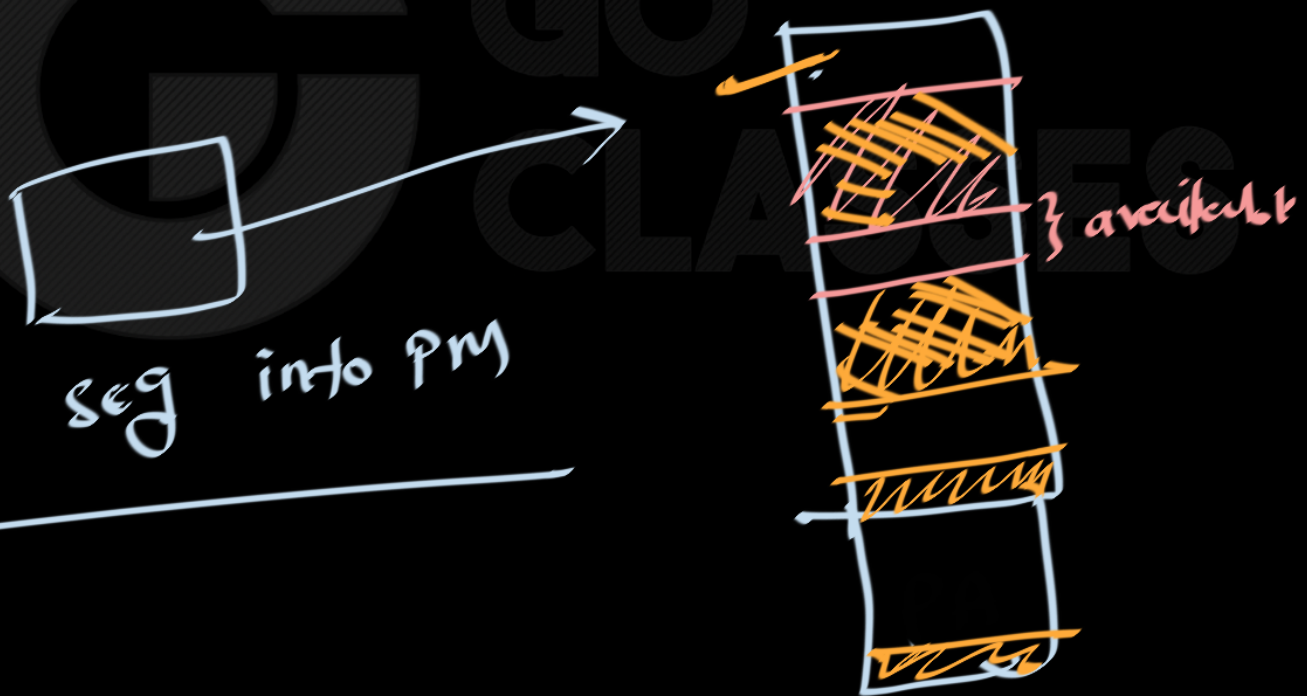





Seg0
Seg1
LAS
Seg1
Seg0
PAS

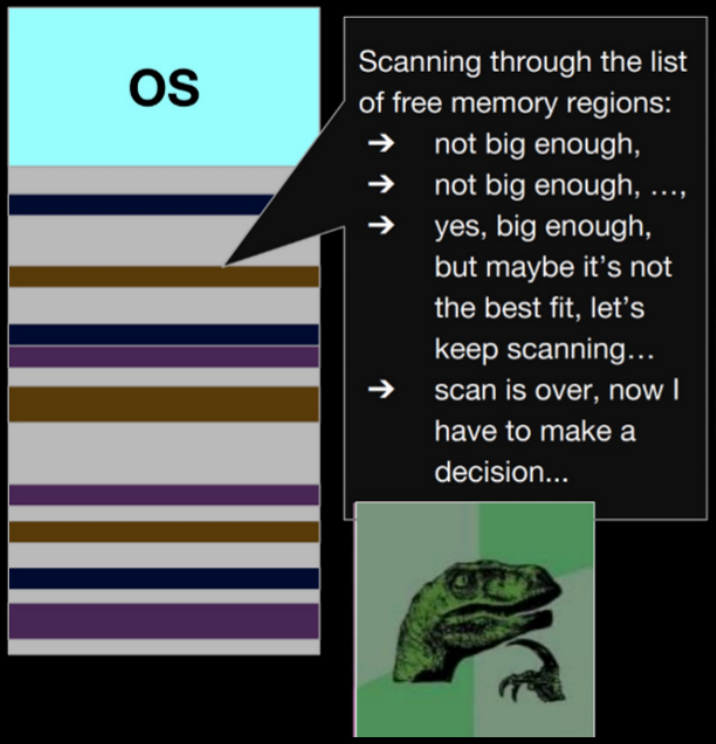
GO CLASSES

Problems in segmentation

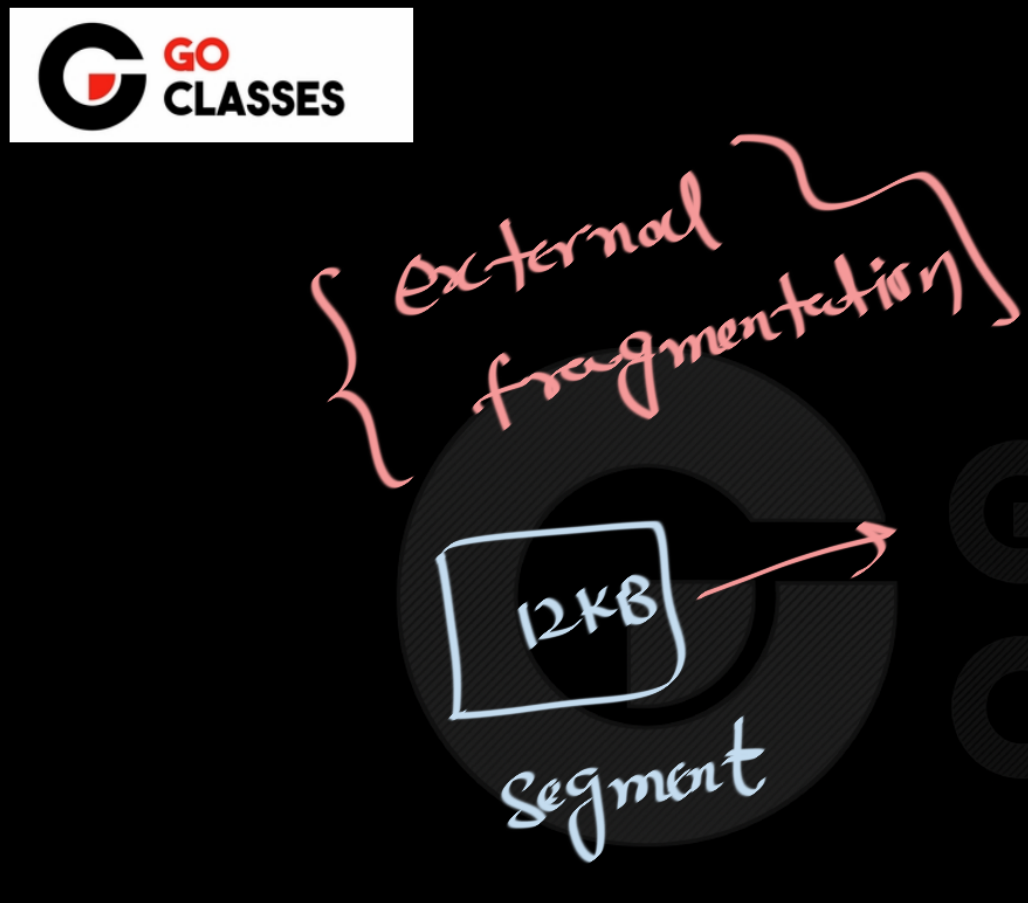


Free space management, when allocating a new chunk...

Segmentation



Problem 1



You can not put in one of the free space although total space satisfy the requirement

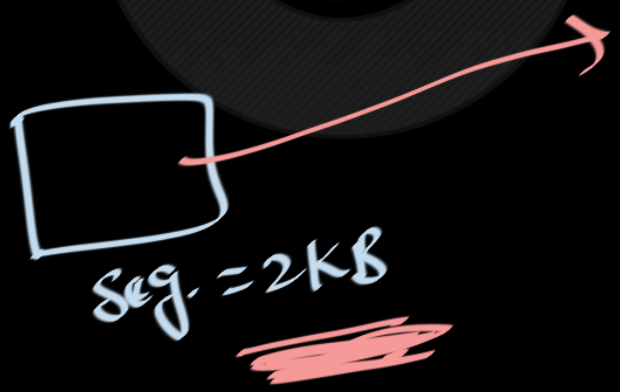
GO CLASSES

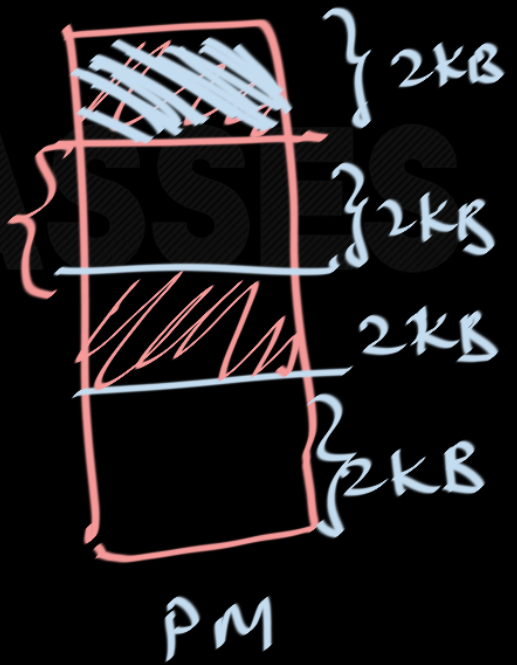
Problems in segmentation:

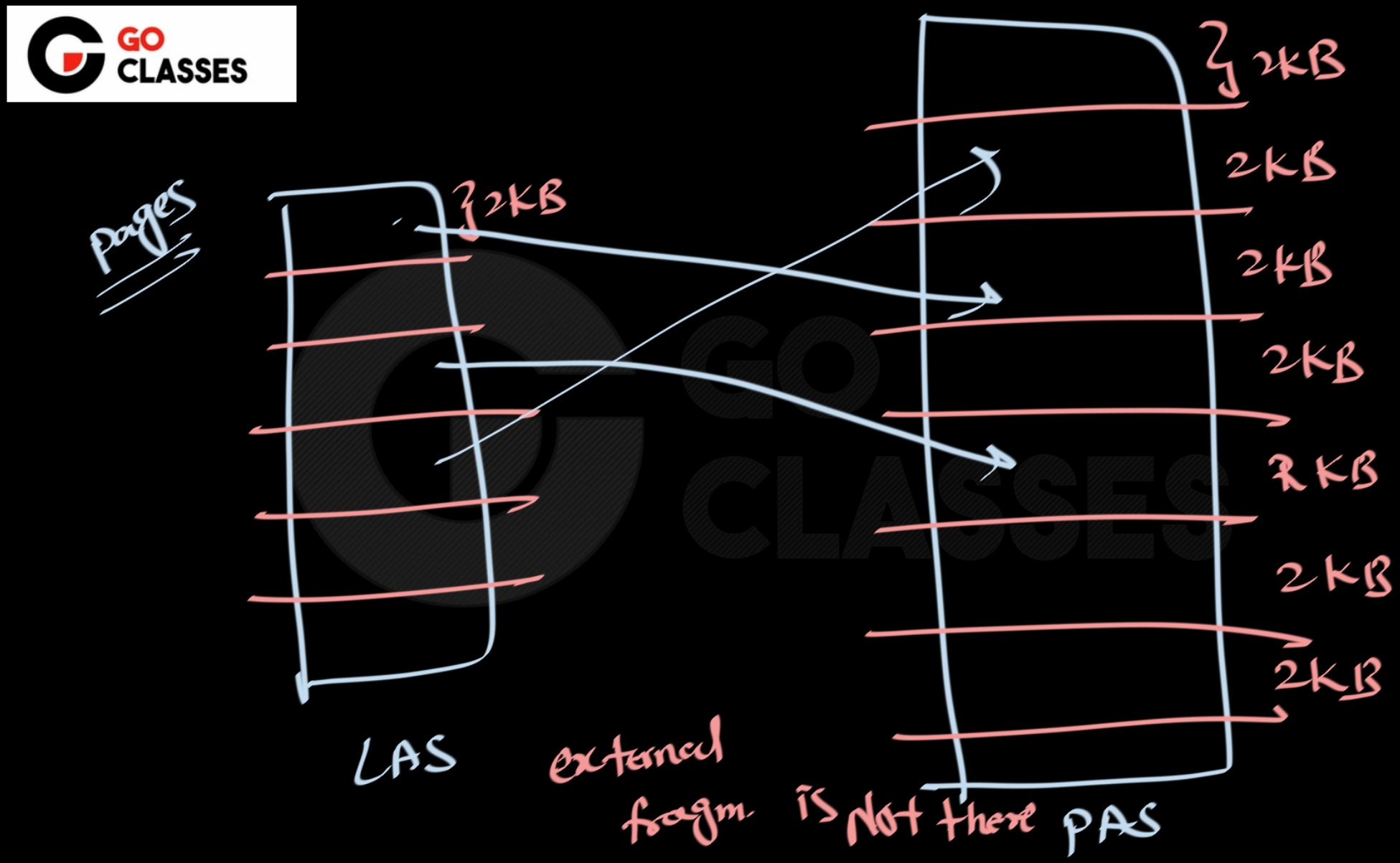
- external fragmentation (total space is enough but not contigious)
- Scanning over free chunks to see if we can put segment here

GO CLASSES

What if we keep all the segments to be equal size






GO CLASSES
Pages
2KB
2KB
2KB
2KB
2KB
2KB
2KB
2KB
LAS
external fragm. is Not there
PAS

GO CLASSES

Paging doesn't have external fragmentation

Why external fragmentation is so imp to us?

→ it is wastage of space

or time

↳ relocation of free partitions will save space but will waste time

GO CLASSES

external frag. is wasting Physical memory, and if we want to save memory by compaction then it is wasting time

Fragmentation

Fragmentation:

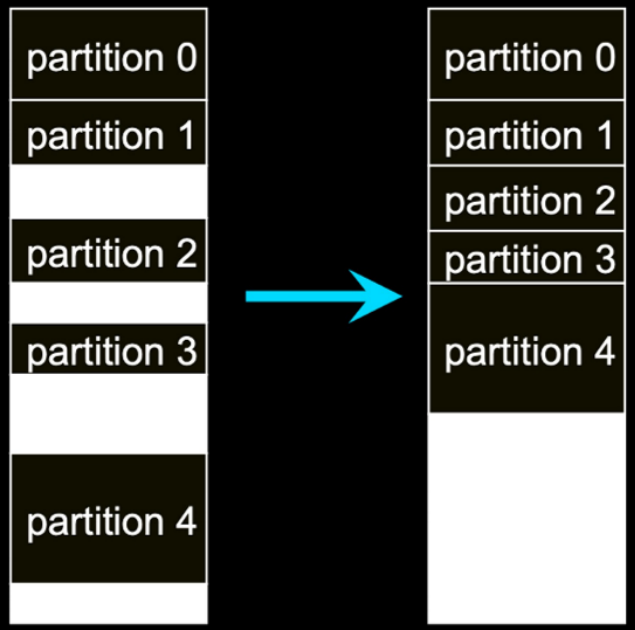
- As processes are loaded and removed from memory, the free memory space is broken into little pieces.
- It happens after sometimes that processes cannot be allocated to memory blocks considering their small size and memory blocks remains unused.
- This problem is known as Fragmentation.

Fragmentation is of two types –

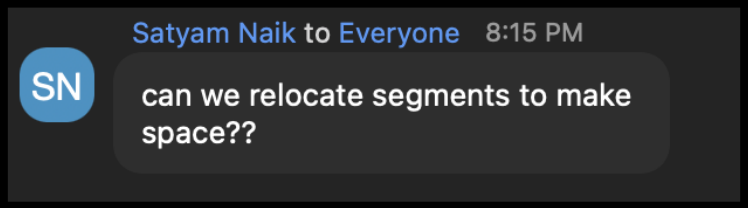
S.N.	Fragmentation & Description
1	**External fragmentation:** Total memory space is enough to satisfy a request or to reside a process in it, but it is not contiguous, so it cannot be used.
2	**Internal fragmentation:** Memory block assigned to process is bigger. Some portion of memory is left unused, as it cannot be used by another process.

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



too much time consuming

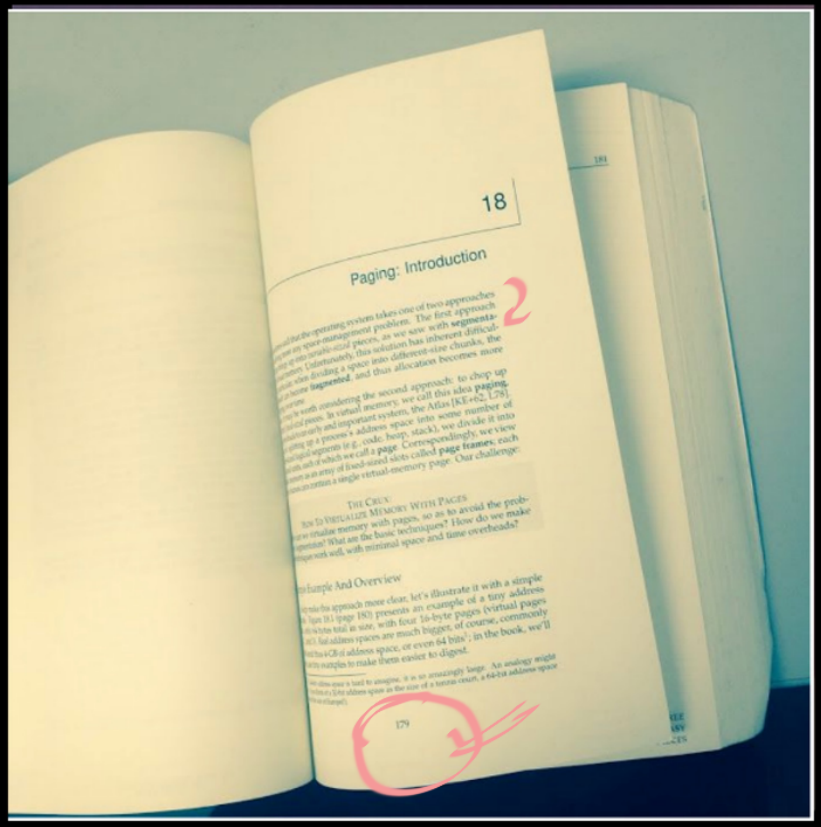


Base - bound segmentation questions on both

The most important topic

Paging

Not really a new idea...

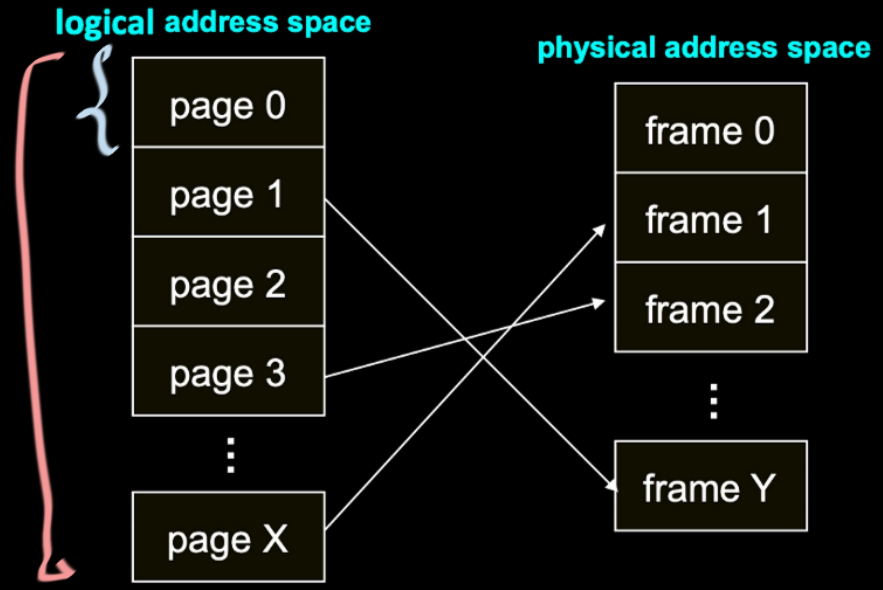


- Divide content into **fixed-sized** pages.
- Each page has a page number.
- Locate content in a page by an offset, e.g., "10th word of Page 8", …
- There is a "table" which tells you which content is in which page.

index

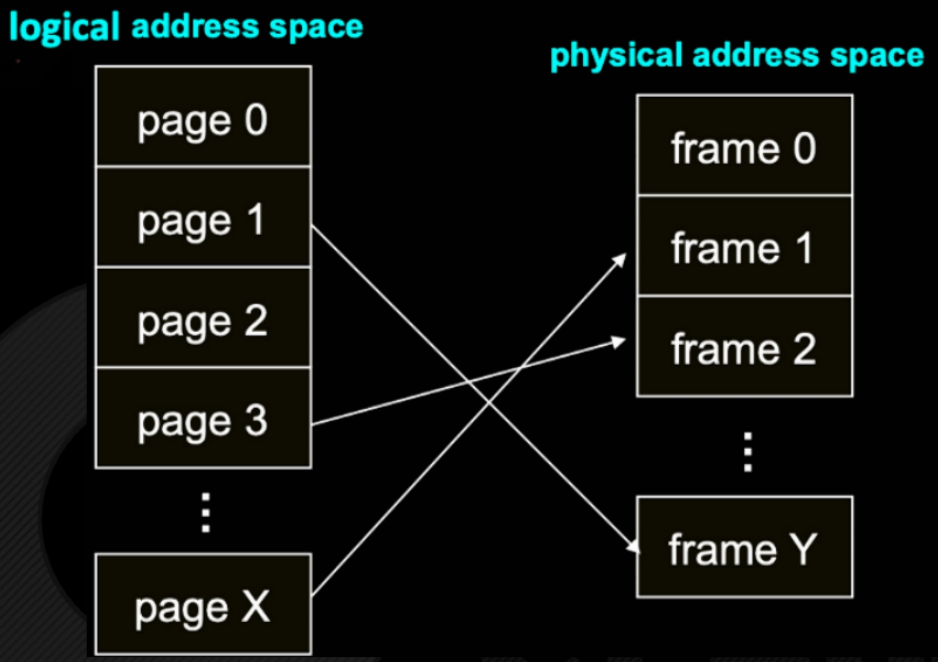
Modern technique: Paging

- 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

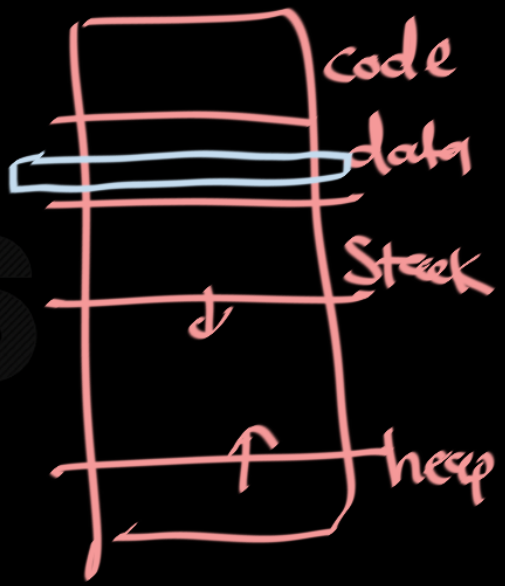


GO CLASSES

Page Size ≡ frame Size


logical address space
physical address space
page 0
page 1
page 2
page 3
page X
frame 0
frame 1
frame 2
frame Y

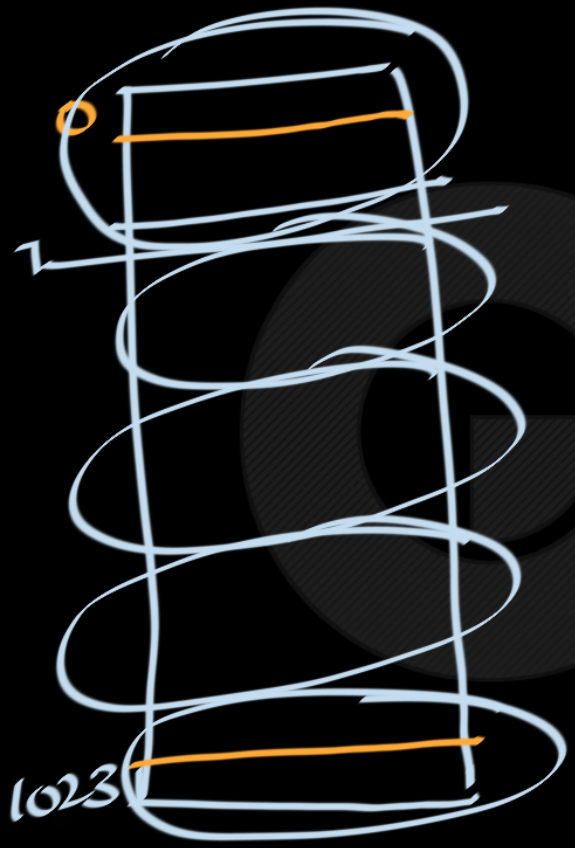
Process


code
data
Stack
heap

Page size = 8 Bytes

No. of pages ?

LAS = 1024 Bytes



Page size = 8 Bytes

No. of pages ?

$$\frac{1024}{8} = 128 \text{ pages}$$

LAS = 1024 Bytes

GO CLASSES

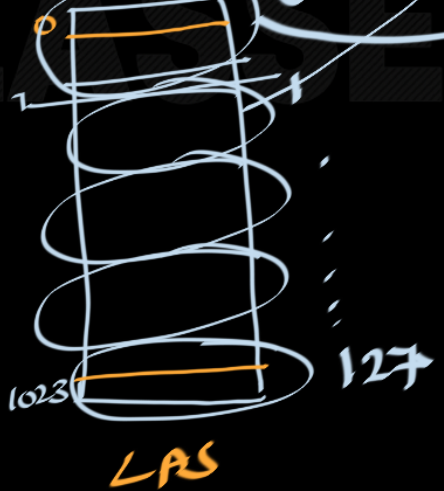
Page size = 8 Bytes

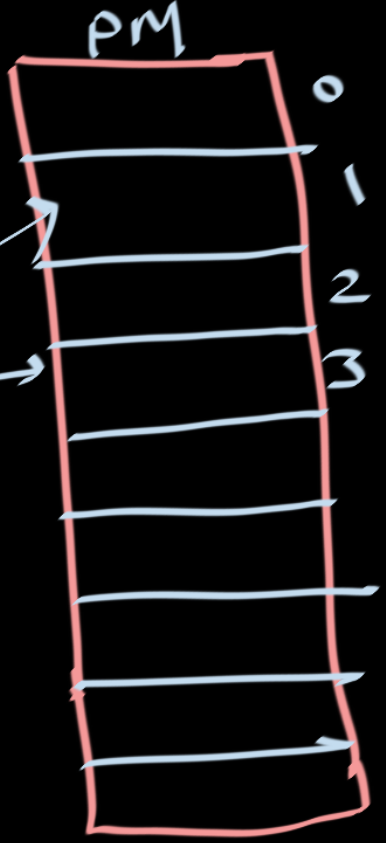
No. of pages ?

$\frac{1024}{8} = 128$ pages

LAS = 1024 Bytes

page no.	frame no.
0	3
1	1





Assume OS maps as follows:

Page size = 8 Bytes

No. of pages ?

$\frac{1024}{8} = 128$ pages

LAS = 1024 Bytes

Assume OS maps as follows:

Pageno	frame No.
0	3
1	6
:	
113	10
:	
127	

} page table

GO CLASSES

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	

Page table

find Physical address for 905. ← logical address

GO CLASSES

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 → ??

GO CLASSES

Question

Suppose LAS = 1024 Bytes

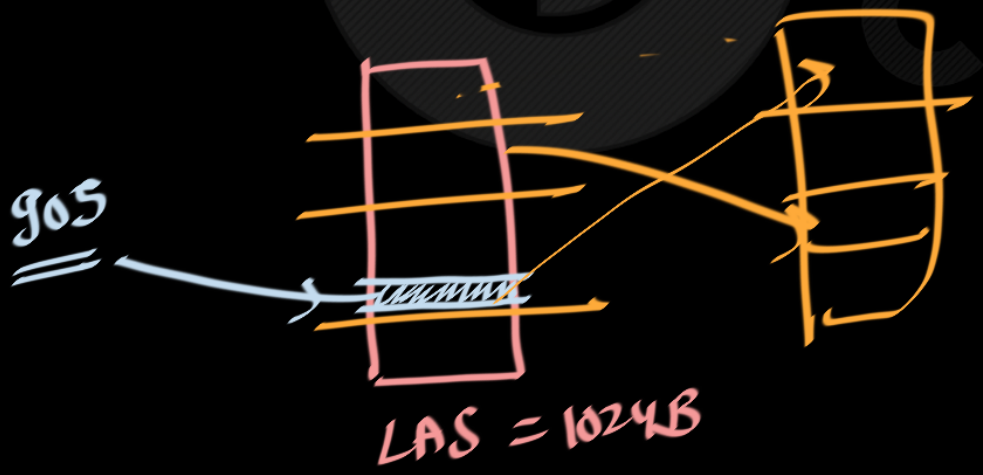
Page size = 8 bytes

and page table is given as follows

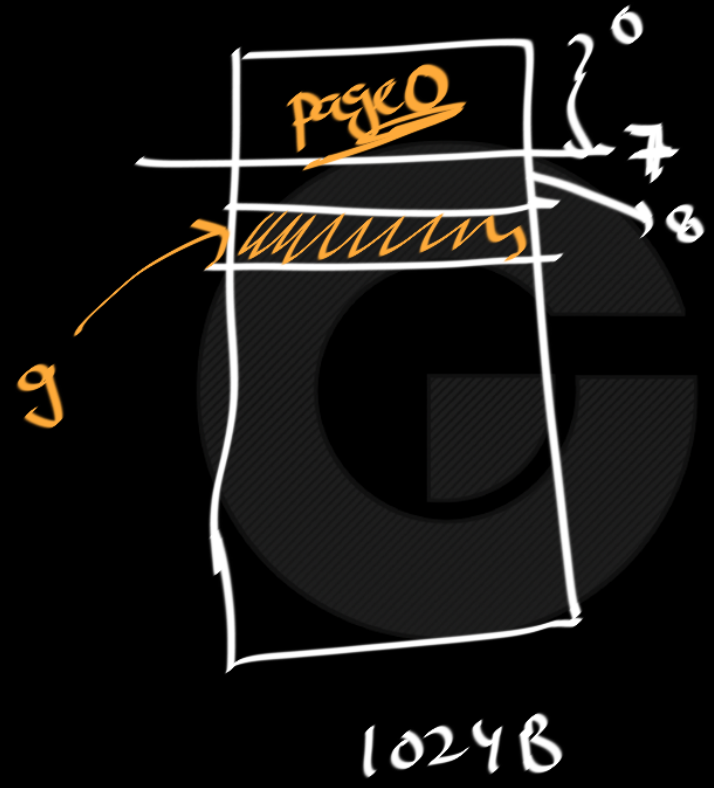
Logical → physical

905 → ??

PN	FN
0	3
1	6
:	
113	10
:	
127	



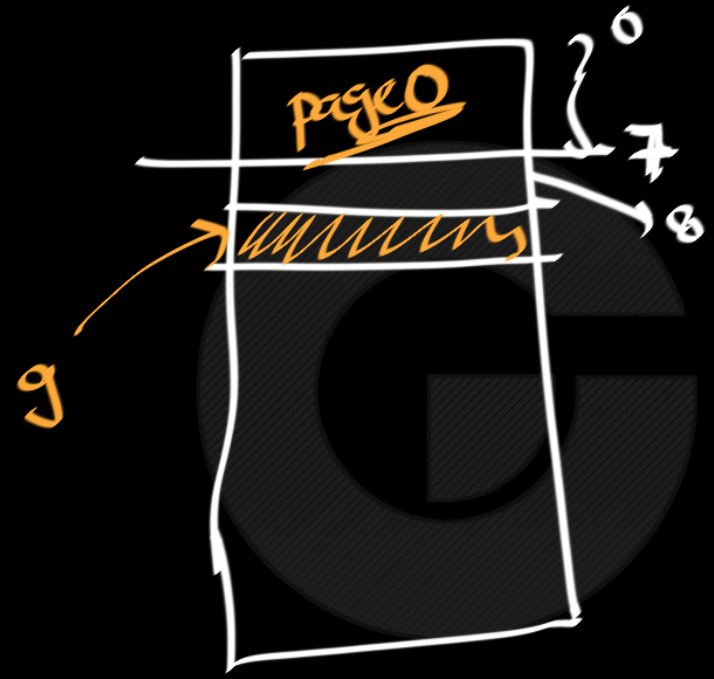
Page No. = $\frac{905}{8}$ = 113.xx



CPU says 9

$9/8 = 1.xx$

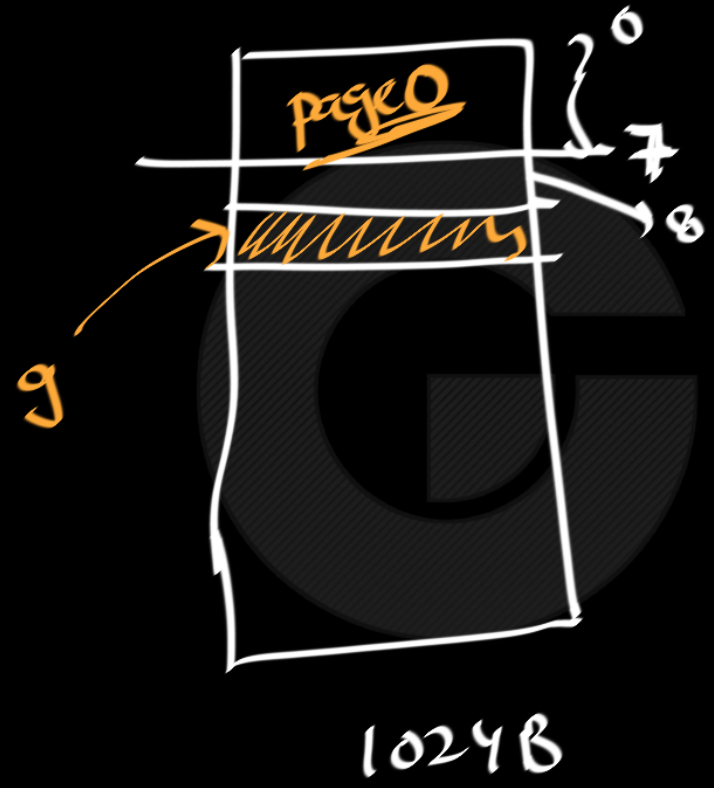
Page No. = 1



CPU says 8

8/8 = 1

Page No. = 1



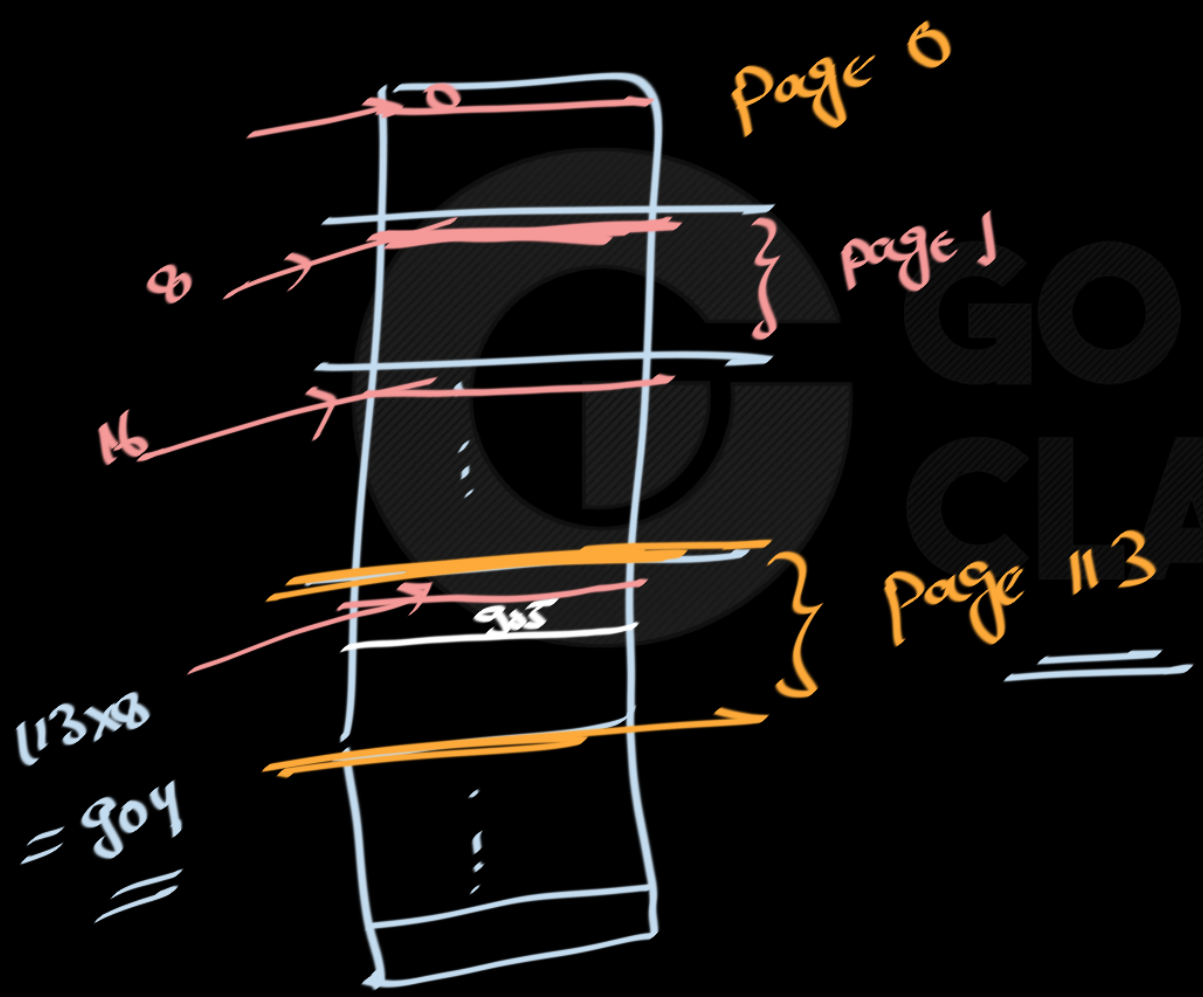
CPU says 905

logical

905 / 8 = 113.xx

Page size

Page No. = 113



Page No.

904 → 113

905 → 113

906 → 113

907 → 113

908 → 113

⋮

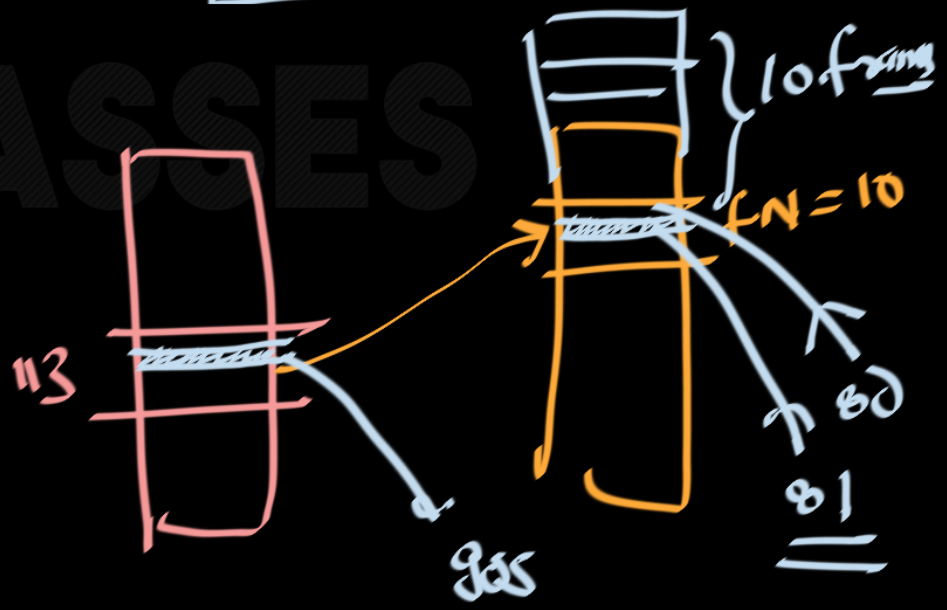
911 → 113

GO CLASSES

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	

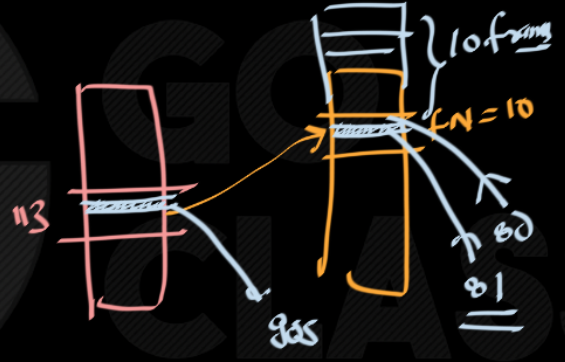


GO CLASSES

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	

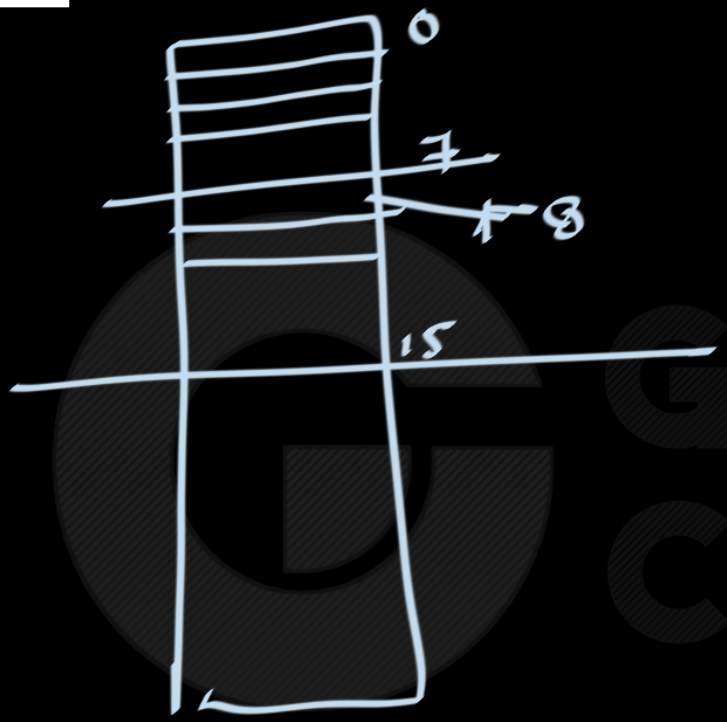


offset for 906

906 % 8

= 2

PA for 906 ⟹ 82



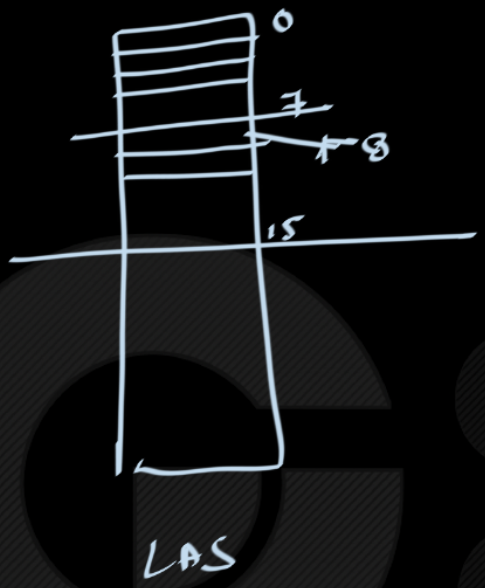
LAS

Page No. for LA = 9

$\frac{9}{8} = 1$ ← Page No.

offset for LA = 9

offset = 1



Page No. for LA = 9

$\frac{9}{8} = 1$ ⇒ Page No.

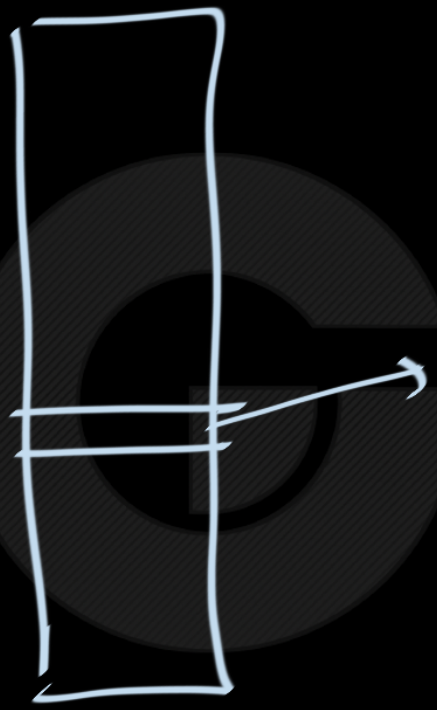
offset for LA = 9

offset = 1

$9 \% 8 = 1$ ⇒

LA = 9 → Page No. = $\frac{LA}{\text{Page size}}$

→ offset = LA % Page size



get page number → frame no.

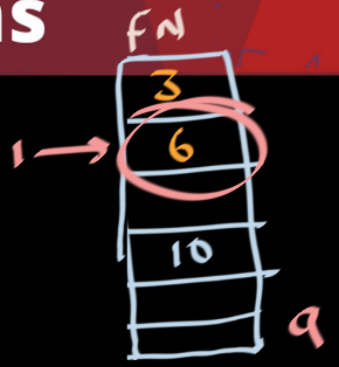
and offset

PA = frame no. × frameSize + offset

Address Translation

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

Address Translation



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

$$Page\ Number = \frac{Logical\ address}{page\_size}$$

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

array

Page table is just array

Address Translation

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

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

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

Address Translation

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

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

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

PA = frame No. × framesize + offset

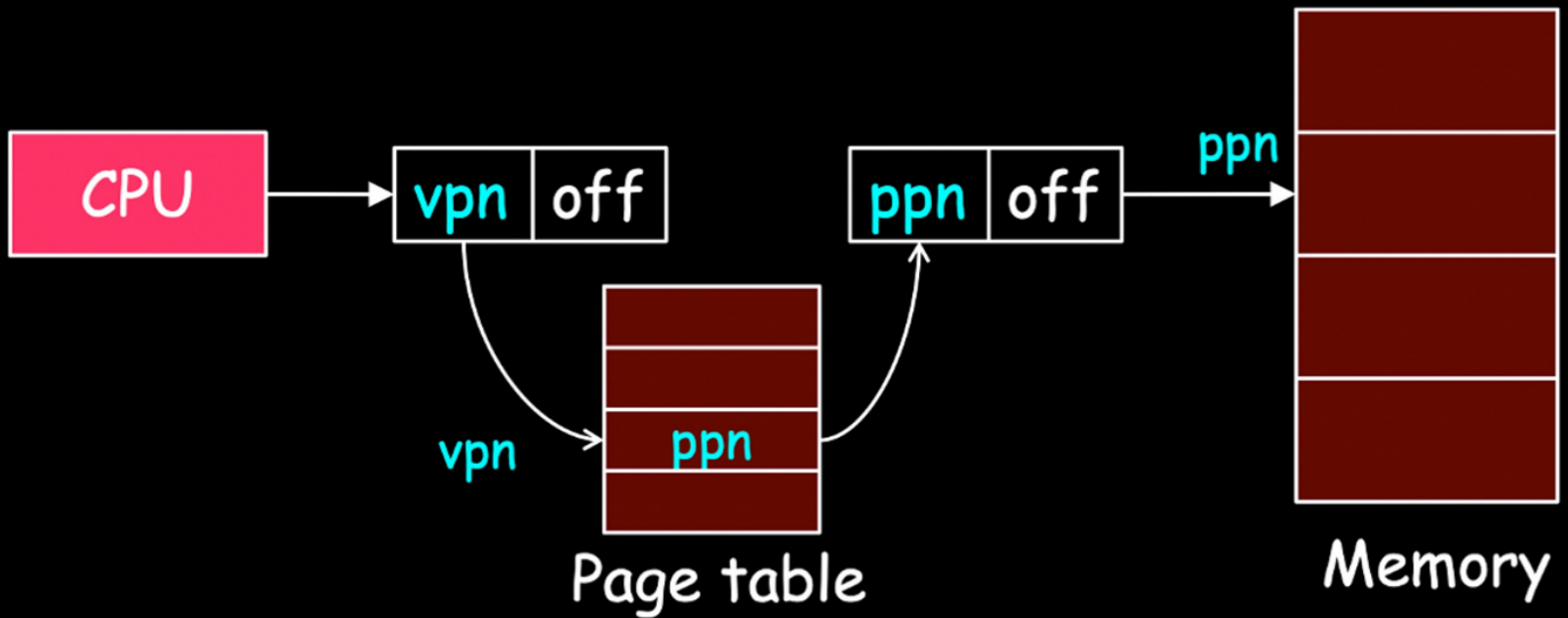
Address Translation

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

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

$Page\_number$

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



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

Size to powers of 2

- $1\ KB\ = 2^{10}\ Bytes$
- $1\ MB\ = 2^{20}\ Bytes$
- $1\ GB\ = 2^{30}\ Bytes$
- $1\ TB\ = 2^{40}\ Bytes$

10^3

10^6

10^9

When we measure data transfer speed (in CN course)

GO CLASSES

1KB = 1024 B ← our PC has this

Buy a hard disk

1KB = 10^3

1 KB
$= 1 \times 10^3$ B
$= 10^3$ B = 1000 B

lesser than 1KB as per your computer =

GO CLASSES

internet speed : 8 Mbps

when you start downloading

it shows : 1 MBps

Questions

(Bytes)

Convert Following sizes into power of 2 –

$128\ KB\ =$

$64\ GB\ =$

$1024\ MB\ =$

$2^{10} \times 2^{20}\ B$

$128\ KB = 2^7 \times 2^{10}\ B = 2^{17}\ B$

$64\ GB = 64 \times 2^{30}\ B$

Address translation Exercise

- Address in the process, A = 10,020
- page size = 4 KB

Calculate Frame number and Frame offset.

Calculate PA also.

H.O. and comment

Page Number	Frame Number
1	2
2	5
3	7
4	1
5	6
6	3

Page Table

Solution:

page number = 10,020 / 4k = 10,020 / 4096 = 2.xxx = truncate to 2
page offset = 10,020 % 4096 = 1828

$Frame\ number = page\_table[2] = 5$
$Frame\ offset = page\_offset = 1828$

$Physical\ address = ?$

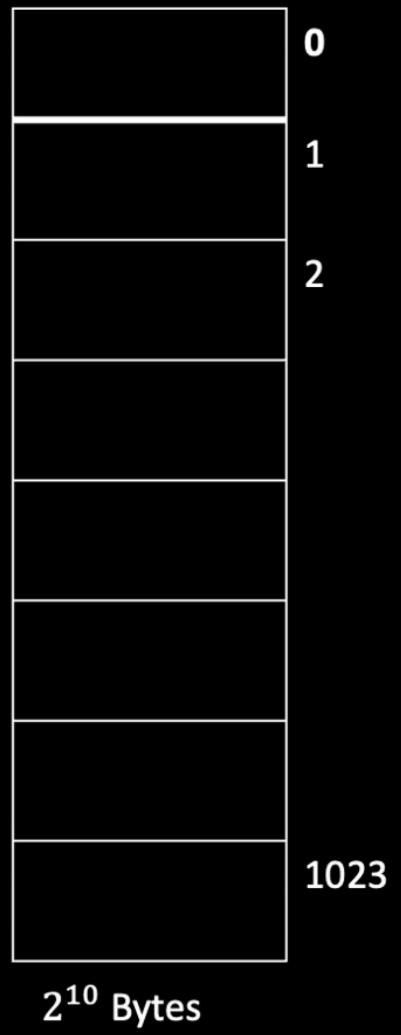
Page Number	Frame Number
1	2
2	5
3	7
4	1
5	6
6	3

Page Table

Suppose CPU generates logical address of 800

Page size = 8 Bytes

Logical Address Space = 1024 Bytes



Suppose CPU generates logical address of 800

Page size = 8 Bytes

$$\text{Page no.} = \frac{800}{8} = 100$$

$$\text{offset} = 0$$

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

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

2^7 Entries

Question

Each page contains = 8 Bytes

What will be physical address corresponding to 807 ?

Pg No	Frame No
0	
1	
100	3
$2^7 - 1$	

2^7 Entries

frame size

$PA = 3 \times 8 + offset$

$= 24 + 7$

$= 31$

Page No. $= \frac{807}{8} = 100.xx$

offset $= 7$

GO CLASSES

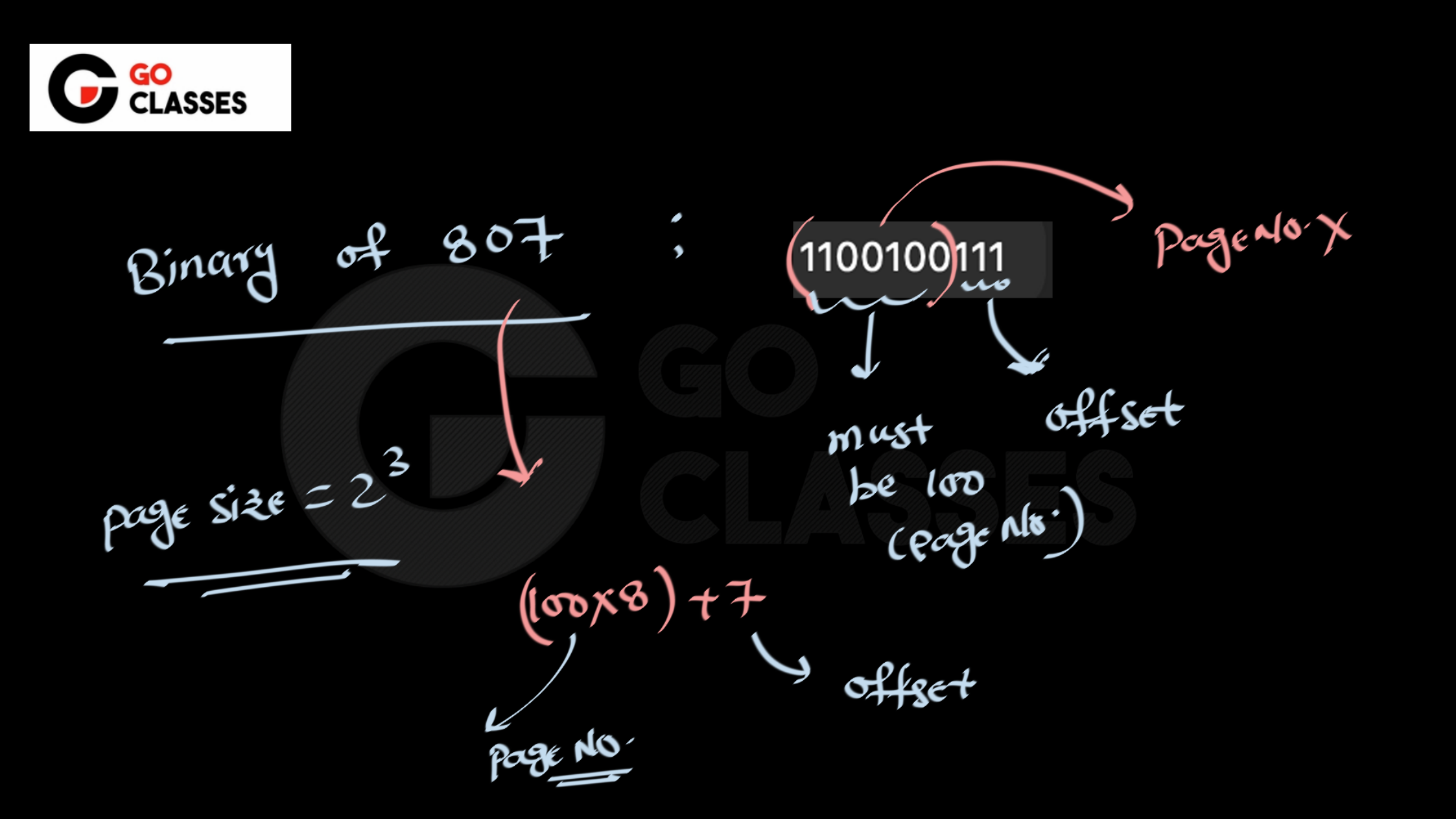
$807 = (100 \times 8) + 7$

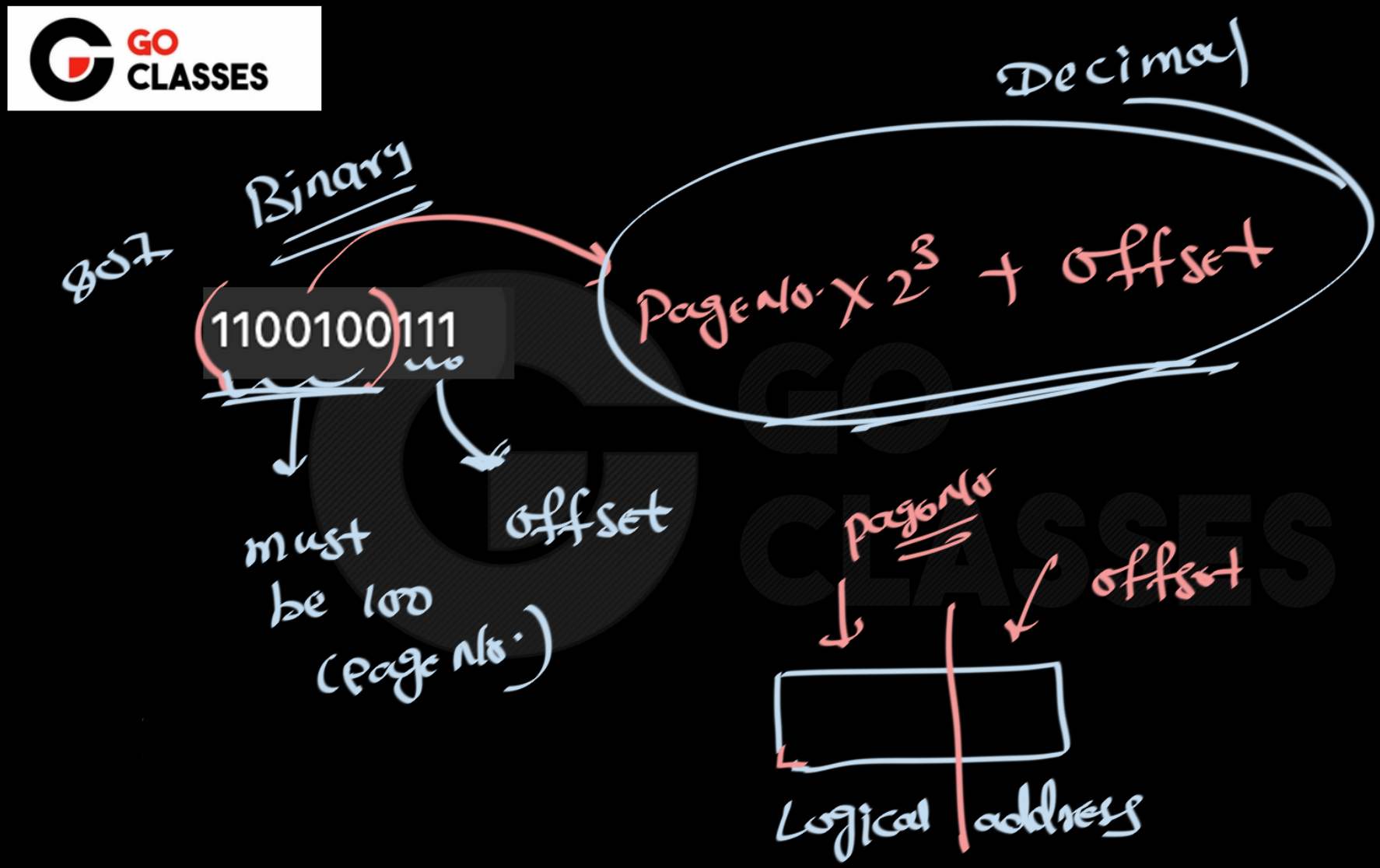
Page Size

$523 \times 2^3 =$ 523 000

Binary of 523 + 7

Binary of 523 | Binary of 7


GO CLASSES
Binary of 807 :
1100100111
Page no. X
must be 100 (page no.)
offset
Page size = 2^3
(100x8) + 7
offset
Page NO.


GO CLASSES
Decimal
Binary
807
1100100111
Page no. × 2^3 + offset
must be 100 (page no.)
offset
page no
offset
Logical address

Divide by 10

Divide by 2

$1298376 \% 10 =$

$1298376 \% 100 =$

$1298376 \% 1000 =$

Divide by 10

Divide by 2

$1298376 \% 10 = 6$

$1298376 \% 100 = 76$

$1298376 \% 1000 = 376$

Divide by 10

Last digit

Divide by 2

$1298376 \% 10 = 6$

Last two digits

$1298376 \% 100 = 76$

$1298376 \% 1000 = 376$

Last three digits

Divide by 10

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

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

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

Divide by 2

$1622 \,\%\, 2 =$

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

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

Divide by 10

$1298376 \% 10 = 6$

$1298376 \% 100 = 76$

$1298376 \% 1000 = 376$

Divide by 2

$1622 \% 2 = 0$

$1622 \% 2^2 = 2$

$1622 \% 2^3 = 6$

logical addrs

page size

Things are better in binary system

Why ? – See next slide

Decimal

$1298376 \% 10 = 6$

$1298376 \% 100 = 76$

$1298376 \% 1000 = 376$

Binary

1622 in binary = 0110 0101 0110

$1622 \% 2 = 0$

$1622 \% 2^2 = 2$

$1622 \% 2^3 = 6$

Decimal

$1298376 \% 10 = 6$

$1298376 \% 100 = 76$

$1298376 \% 1000 = 376$

Binary

1622 in binary = 0110 0101 0110

$1622 \% 2 = 0$

$1622 \% 2^2 = 2$

$1622 \% 2^3 = 6$

Decimal

Binary

$1298376 \% 10 = 6$

$1298376 \% 100 = 76$

$1298376 \% 1000 = 376$

1622 in binary = 0110 0101 0110

5×2^3

101 000

$1622 \% 2 = 0$

$1622 \% 2^2 = 2$

$1622 \% 2^3 = 6$

Same example in binary system

$Logical\ address = 10{,}020 = 0010\ 0111\ 0010\ 0100$

(in binary 16 bits)

$Page\ size = 4KB = 4096\ Bytes = 2^{12}\ Bytes$

offset = 12 bits

page no = 16 - 12 = 4 bits

Page Number	Frame Number
1	2
2	5
3	7
4	1
5	6
6	3

Page Table

Same example in binary system

$$\text{Page offset} = \frac{Logical\ address}{page\ size} = \frac{10020}{2^{12}}$$

$$Logical\ address = 0010\ 0111\ 0010\ 0100$$

12 bits

Page offset = $0111\ 0010\ 0100$ = 1828

(In binary) (In decimal)

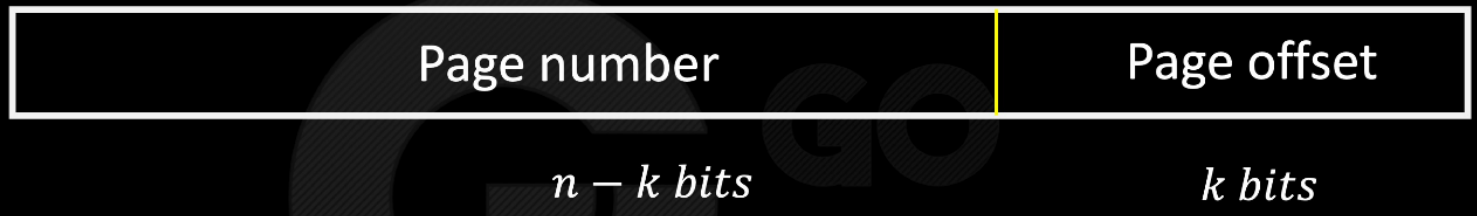
Same example in binary system

$$Logical\ address = 0010\ 0111\ 0010\ 0100$$

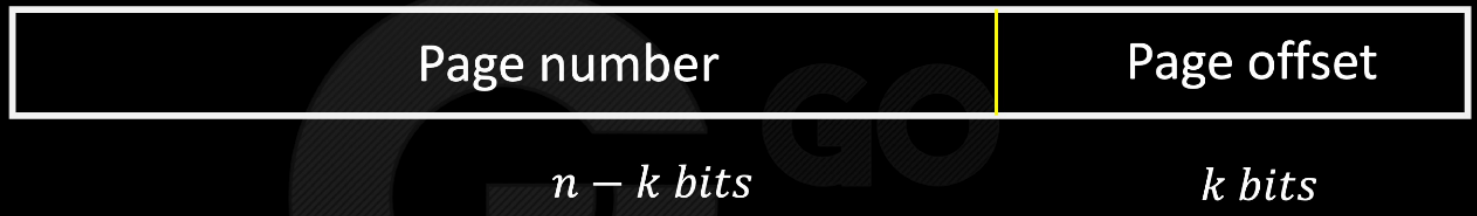
Page number = 0010 = 2

Computers never need decimal number system

If logical address is n bits and page size is 2^k then



If logical address is n bits and page size is 2^k then



This is very important slide

Logical address space = n $bits$

Physical address space = m $bits$

2^k

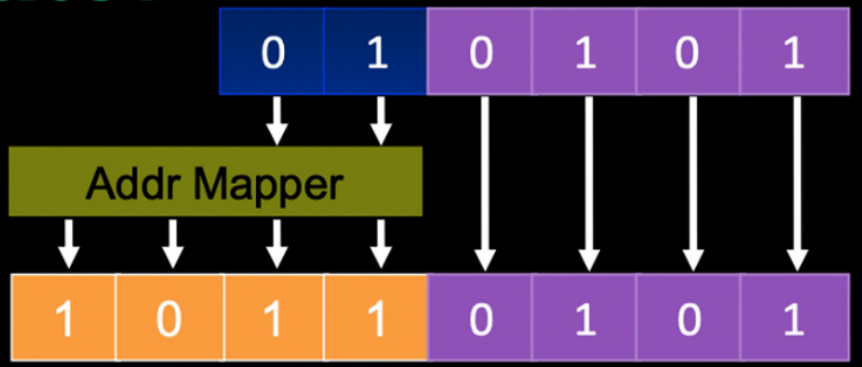
Page translation exercise

- 8-bit virtual address, 10-bit physical address, and each page is 64 bytes
  - How many virtual pages?
  - How many physical pages?
  - How many entries in page table?
  - Given page table = [2, 5, 1, 8], what's the physical address for virtual address 241?

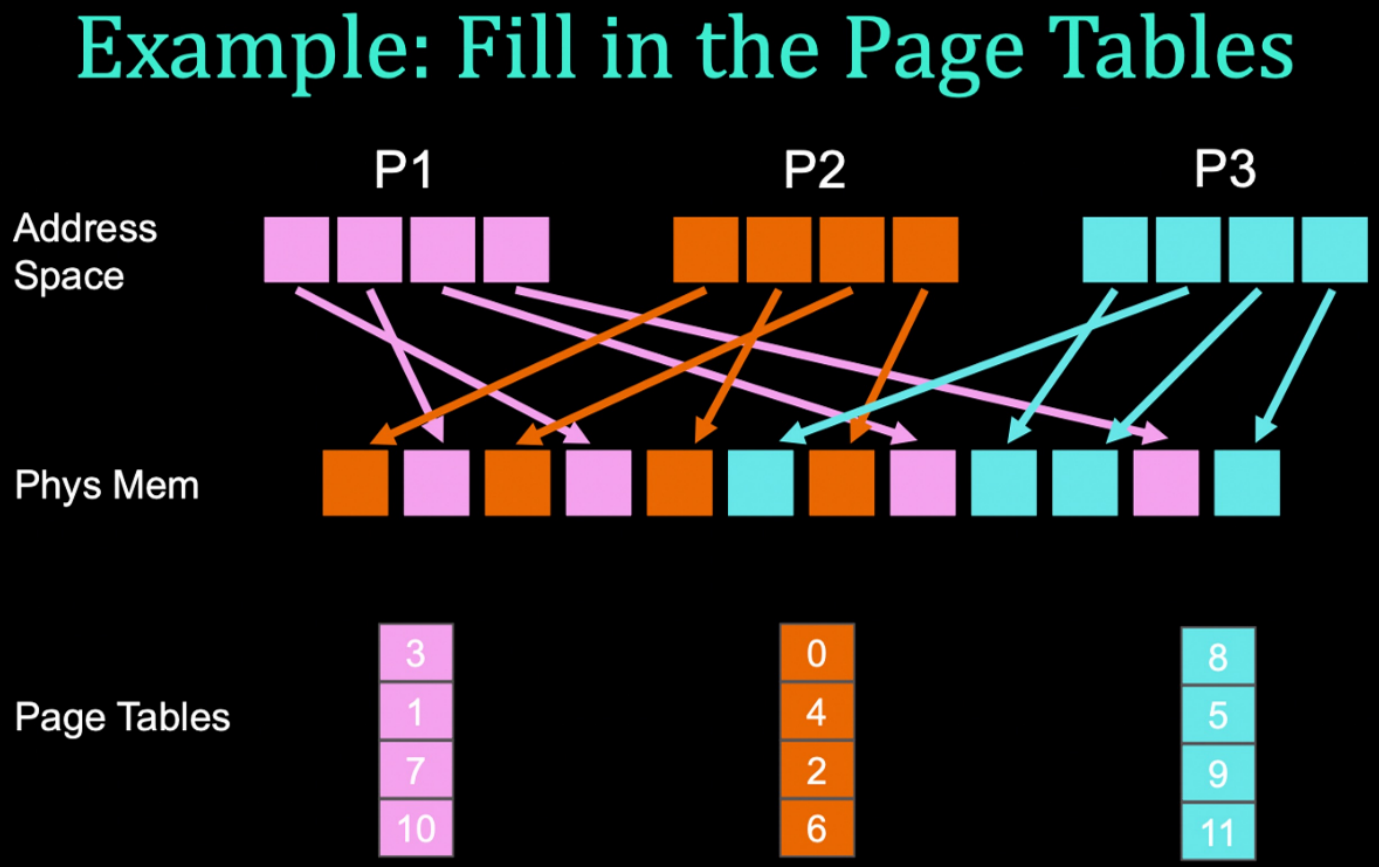
Optional Image

How to Translate?

Note: number of bits in virtual address does not need to equal number of bits in physical address



Optional Image



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

Common Flags Of Page Table Entry

- Valid Bit: Indicating whether the particular translation is valid.
- Protection Bit: Indicating whether the page could be read from, written to, or executed from
- Present Bit: Indicating whether this page is in physical memory or on disk(swapped out)
- Dirty Bit: Indicating whether the page has been modified since it was brought into memory
- Reference Bit(Accessed Bit): Indicating that a page has been accessed

Complete Page Table Entry (PTE)

Valid	Protection R/W/X	Ref	Dirty	Frame number

Synonyms:

- Valid bit == Present bit
- Dirty bit == Modified bit
- Referenced bit == Accessed bit

Question

- Logical address = 256KB
- Physical address = 64KB
- Page size = 4 KB

Calculate:

- Number of pages
- Number of bits in page offset
- Number of frames
- If each page table entry size is 2^p Bytes then size of page table

Question

- Physical address = 128MB
- Page size = 4 KB
- Number of pages = 2^6

Calculate:

- Number of frames
- Page offset bits
- Logical address space size
- If each page table entry size is 2^p Bytes then size of page table

Question

- Logical address = 1GB
- Page size = 4 MB
- Number of frames = 2^{10}

Calculate:

- Physical address space size
- Page offset bits
- Logical address space size
- If each page table entry size is 2^p Bytes then size of page table

Question

- Logical address = 1TB
- Number of pages = 2^{12}
- Number of frames = 2^{10}

Calculate:

- Physical address space size
- Page offset bits
- If each page table entry size is 2^{p} Bytes then size of page table

Question

Problem 5
Consider the following page table used for address translations:
(for this portion of the question, all numbers are expressed using base 10).

Virtual page	Frame
0	3
1	10
2	9
3	2
4	0

← page table

Assume the page size is 1024 bytes, convert the following virtual addresses into physical addresses (show your calculations):

a. **(2 marks)** 697

b. **(2 marks)** 1054

c. **(2 marks)** 1024

d. **(4 marks)** If possible, convert the **physical address** 2075 into a **virtual address**. If not explain why it can not be done.

a. **(2 marks)** 697

————————begin solution————————

697 = 0 * 1024 + 697 and so vpn = 3.
The real Address is 3 * 1024 + 697 = 3769

b. **(2 marks)** 1054

————————begin solution————————

1054 = 1024 + 30 and so is on page 1.
The real Address is 10*1024 + 30 = 10270;

————————end solution————————

c. **(2 marks)** 1024

————————begin solution————————

```
page 0 is virtual address 0 .. 1023.
So 1024 =  is vpn = 1  offset = 0
The real Address is 10 * 1024 = 10240
```

————————end solution————————

d. **(4 marks)** If possible, convert the **physical address** 2075 into a **virtual address**. If not explain why it can not be done.

————————begin solution————————

```
2075 is on physical frame 2 with offset 27.
From the page table we see that frame 2 is virtual page 3.
Virtual page 3 is 3 * 1024 = 3072 + the offset (27)
So the virtual address = 3099.
```

————————end solution————————

Question

Consider a single-level paging system with 12-bit virtual addresses, 24-bit physical addresses, and a 256 (2^8) byte page size.

What is the maximum number of entries in a page table in this system?

Solution

Consider a single-level paging system with 12-bit virtual addresses, 24-bit physical addresses, and a 256 (2^8) byte page size.

What is the maximum number of entries in a page table in this system?

$2^4 = 16$

Question

Consider a virtual memory system that uses paging. Virtual and physical addresses are both 32 bits long, and the page size is 4KB = 2^{12} bytes. A process P_1 has the following page table. Frame numbers are given in notation (recall that each hexadecimal digit represents 4 bits).

	Frame Number
0	0x00788
1	0x00249
2	0x0023f
3	0x00ace
4	0x00bcd

For each of the following virtual addresses, indicate the physical address to which it maps. If the virtual address is not part of the address space of P_1, write `NO TRANSLATION` instead. Use hexadecimal notation for the physical addresses.

- `0x00001a60`
- `0x000051ff`
- `0x00000fb5`

https://student.cs.uwaterloo.ca/~cs350/common/old-exams/W11-midterm-sol.pdf

- 0x00001a60

  0x00249a60

- 0x000051ff

  NO TRANSLATION

- 0x00000fb5

  0x00788fb5

Question

Consider a virtual memory system that uses paging. Virtual and physical addresses are both 32 bits long, and the page size is $4\text{KB} = 2^{12}$ bytes. A process P_1 has the following page table. Frame numbers are given in notation (recall that each hexadecimal digit represents 4 bits).

	Frame Number
0	0x00788
1	0x00249
2	0x0023f
3	0x00ace
4	0x00bcd

For each of the following physical addresses, indicate the virtual address that maps to it. If the physical address is not part of the physical memory assigned to P_1, write `NO TRANSLATION` instead. Use hexadecimal notation for the virtual addresses.

- `0x00aceff6`
- `0x00249000`
- `0x00000887`

- 0x00aceff6

  0x00003ff6

- 0x00249000

  0x00001000

- 0x00000887

  NO TRANSLATION

Question

4.

Virtual Address	Physical Address
0x0008 0AF0	0x0010 1AF0
0x0000 2224	0x0008 3224
0x0007 3234	0x0018 3234

Suppose that while a process P is running, it uses the virtual addresses shown in the left column of the table above. For each of these virtual addresses, the corresponding physical address (after address translation) is also shown in the table. On the machine on which P is running, both virtual addresses and physical addresses are 32 bits long.

a. (3 marks)

Is it possible that the MMU used *paging*, with a page size of 64 KB (2^{16} bytes), to translate P's virtual addresses to physical addresses? If so, write "YES". If not, write "NO" and explain, briefly and clearly, how you know that paging with this page size was not used.

b. (2 marks)

Is it possible that the MMU used paging, with a page size of 4 KB (2^{12} bytes), to translate P's virtual addresses to physical addresses? If so, write "YES". If not, write "NO" and explain, briefly and clearly, how you know that paging with this page size was not used.

a. (3 marks)

Is it possible that the MMU used *paging*, with a page size of 64 KB (2^{16} bytes), to translate P's virtual addresses to physical addresses? If so, write "YES". If not, write "NO" and explain, briefly and clearly, how you know that paging with this page size was not used.

NO. If page sizes of 64 KB are used, virtual and physical addresses should have the same 16-bit offset. However, `0x0000 2224` and `0x0008 3224` have different offsets.

b. (2 marks)

Is it possible that the MMU used paging, with a page size of 4 KB (2^{12} bytes), to translate P's virtual addresses to physical addresses? If so, write "YES". If not, write "NO" and explain, briefly and clearly, how you know that paging with this page size was not used.

YES.

Question

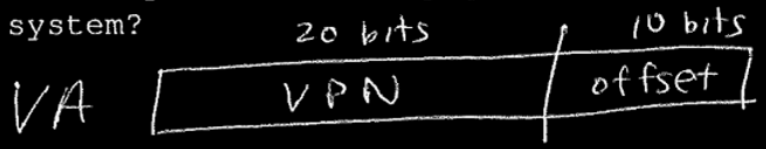
14. A system uses paging to implement virtual memory. Specifically, it uses a simple linear page table. The virtual address space is of size 1 GB (30 bits); the page size is 1 KB; each page table entry holds only a valid bit and the resulting page-frame number (PFN); the system has a maximum of 2^{15} physical pages (32 MB of physical memory can be addressed at most).

How much memory is used for page tables, when there are 100 processes running in the system?

https://pages.cs.wisc.edu/~remzi/Classes/537/Fall2013/OldExams/11-fall-mid.pdf

14. A system uses paging to implement virtual memory. Specifically, it uses a simple linear page table. The virtual address space is of size 1 GB (30 bits); the page size is 1 KB; each page table entry holds only a valid bit and the resulting page-frame number (PFN); the system has a maximum of 2^{15} physical pages (32 MB of physical memory can be addressed at most).

How much memory is used for page tables, when there are 100 processes running in the system?



⇒ size of page table is thus 2^{20} or

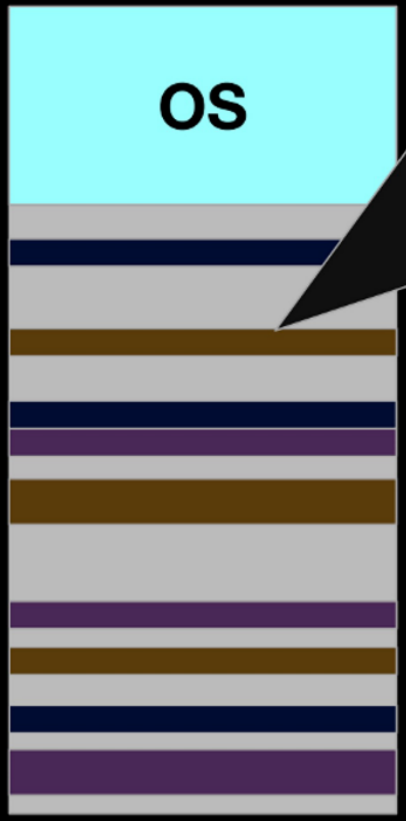
1 MB

⇒ if 100 processes,

100 MB

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

Segmentation

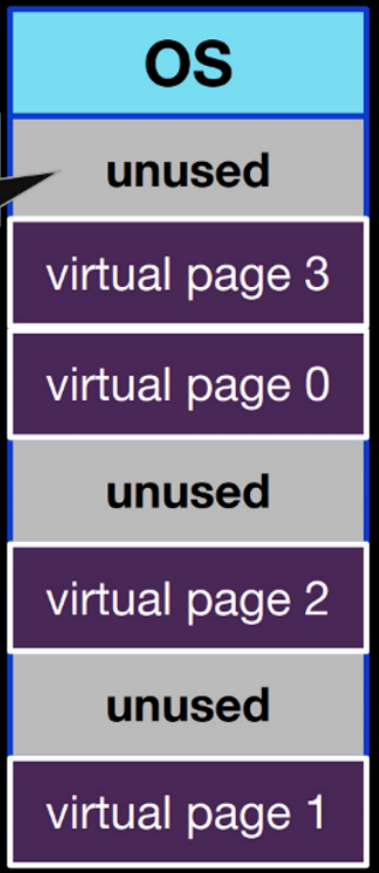


Scanning through the list of free memory regions:

- ➔ 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.

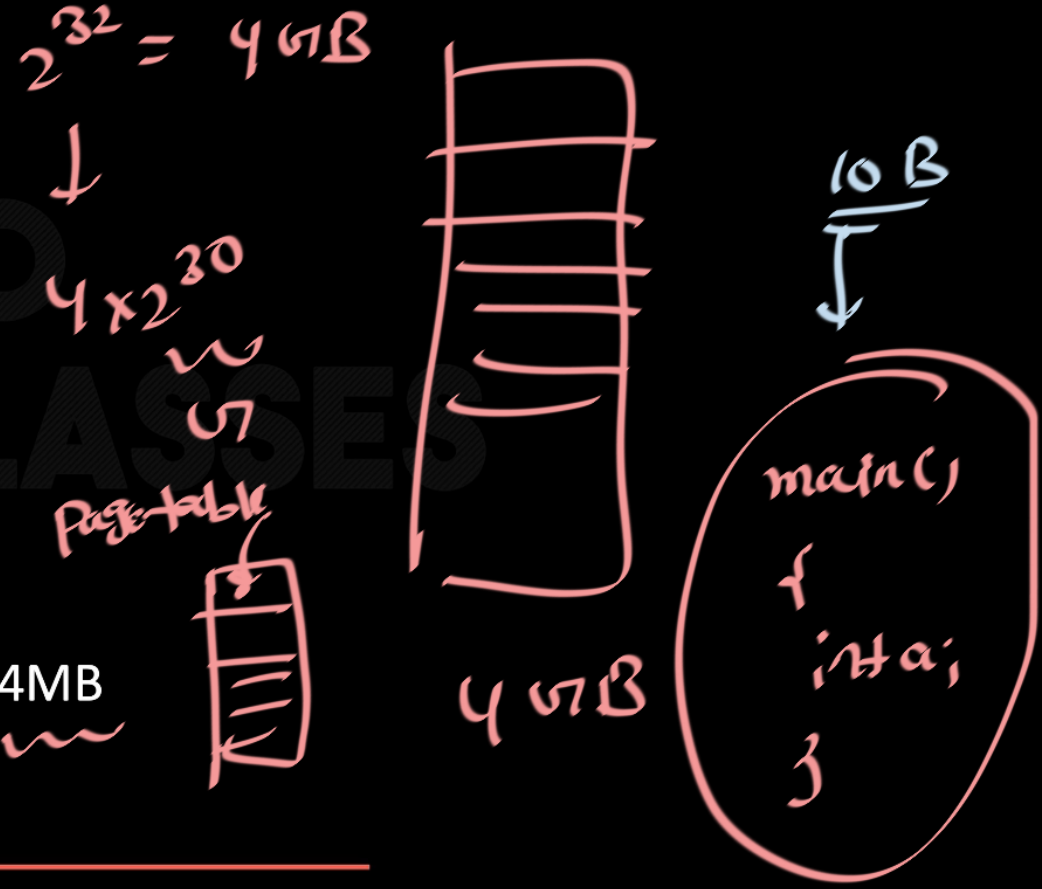
A Typical System has:

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.



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

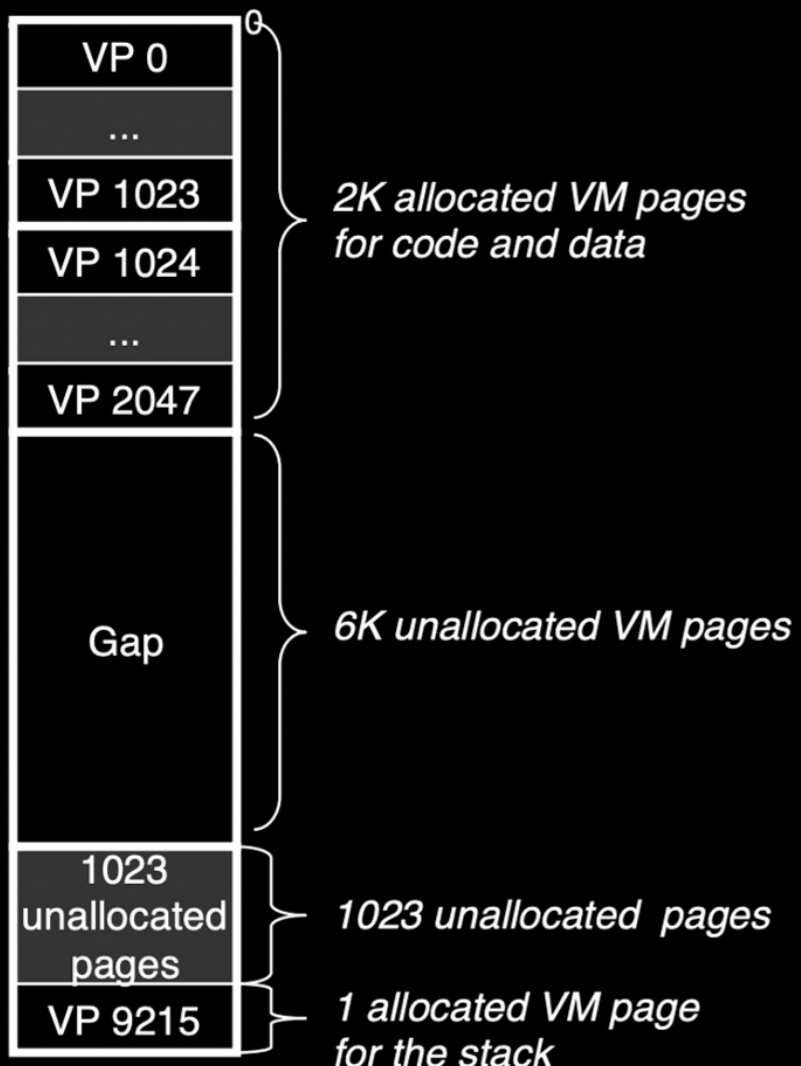
- One such page table for each process.

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


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

now we have a challenge:

how to reduce the size of page tables?

the smart solution:

Multi-Level Page Tables

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 ?

Multi Level Paging