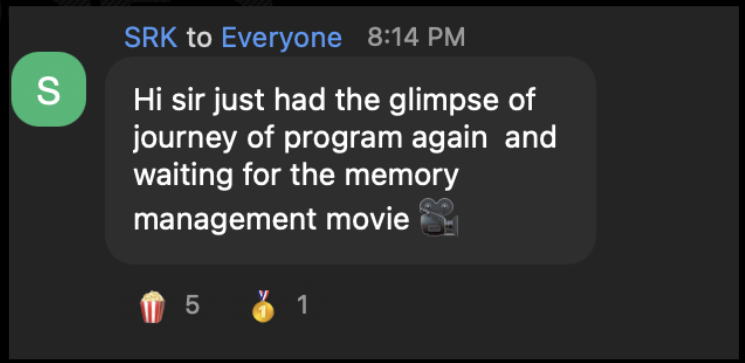
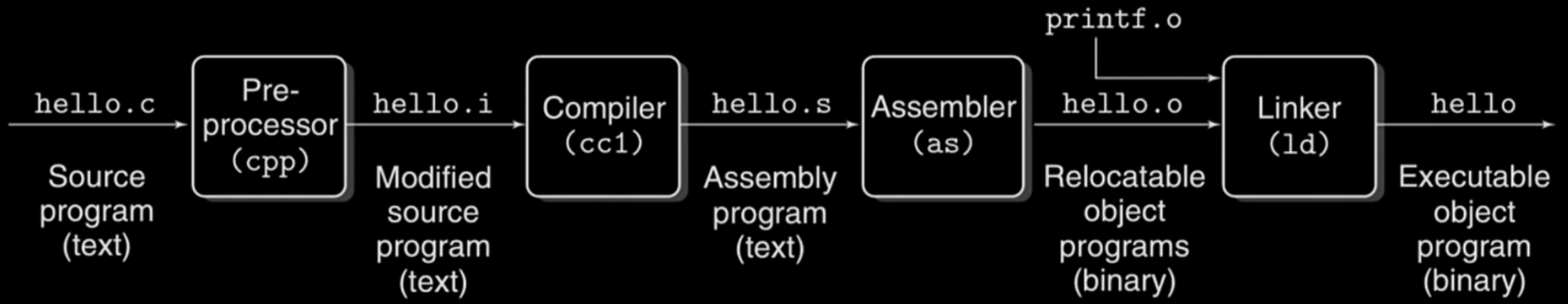
Lecture: 23


SRK to Everyone 8:14 PM
S
Hi sir just had the glimpse of journey of program again and waiting for the memory management movie
5
1



run after 5 years

may be on even different machine

The Journey of a Program | The Big Picture | Compiler, Linker, Assembler, and Loader.

1.2K views • 3 months ago

GO Classes for GATE CS

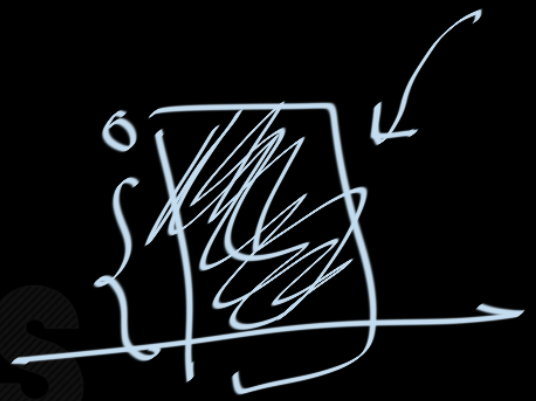
---------- Feel free to Contact Us for any query. ➤ **GO Classes** Contact : (+91)63025 36274 ...

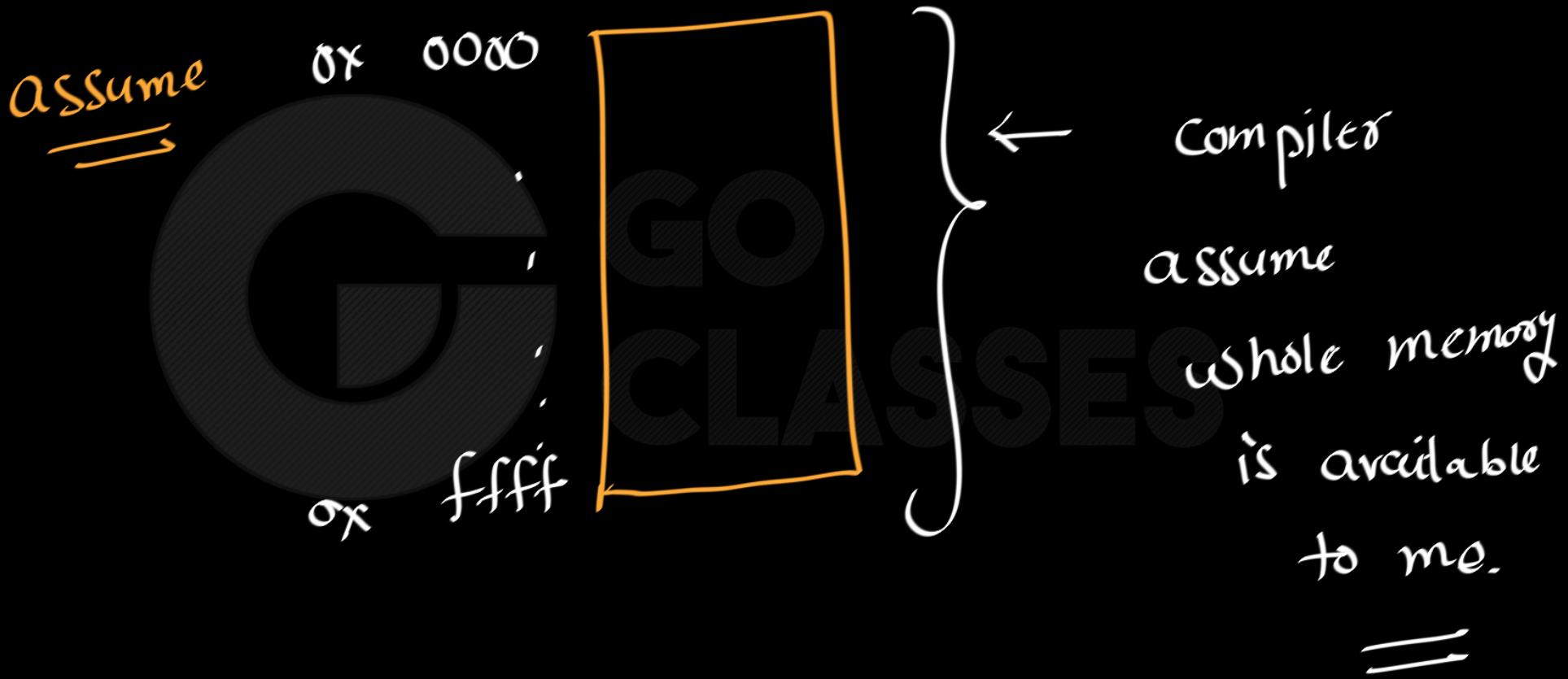
GO CLASSES

```
main ( )
{
    int x = 1;
}
```

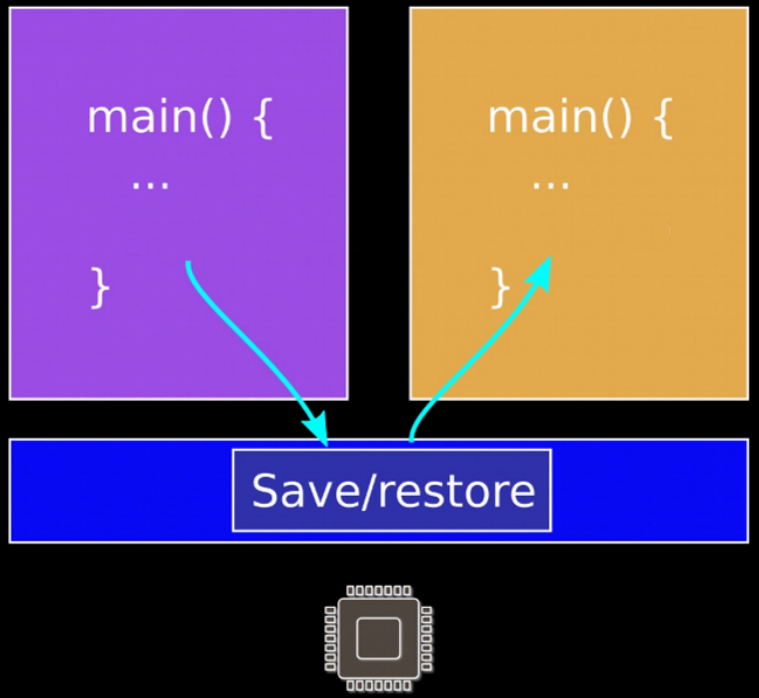
⇒ Load a, 1

x is at some address "a"




assume
0x 0000
0x ffff
Compiler
assume
whole memory
is available
to me.

Two programs one memory



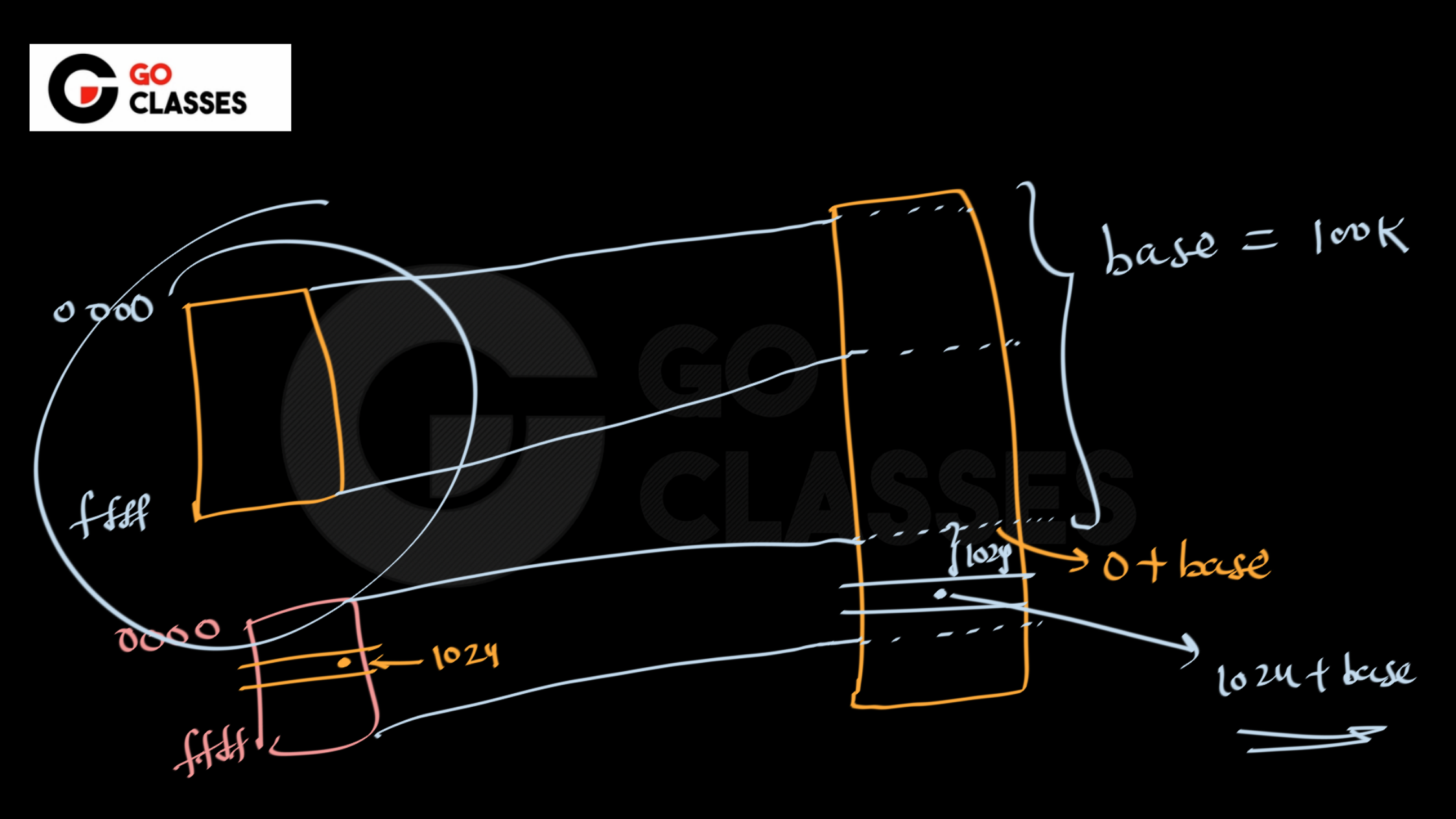
if compiler assume that any program start address is "0"

then we can not have 2 Programs at same time in MM.

GO CLASSES

we can not ask compiler to generate

Physical addresses.


GO CLASSES
base = 100K
0000
ffff
0000
1024
ffff
1024
0 + base
1024 + base

Relocation

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

Two processes, one memory?

→ Process is contigiously stored in main memory

→ Process start from "base" address

→ each Process has its base address

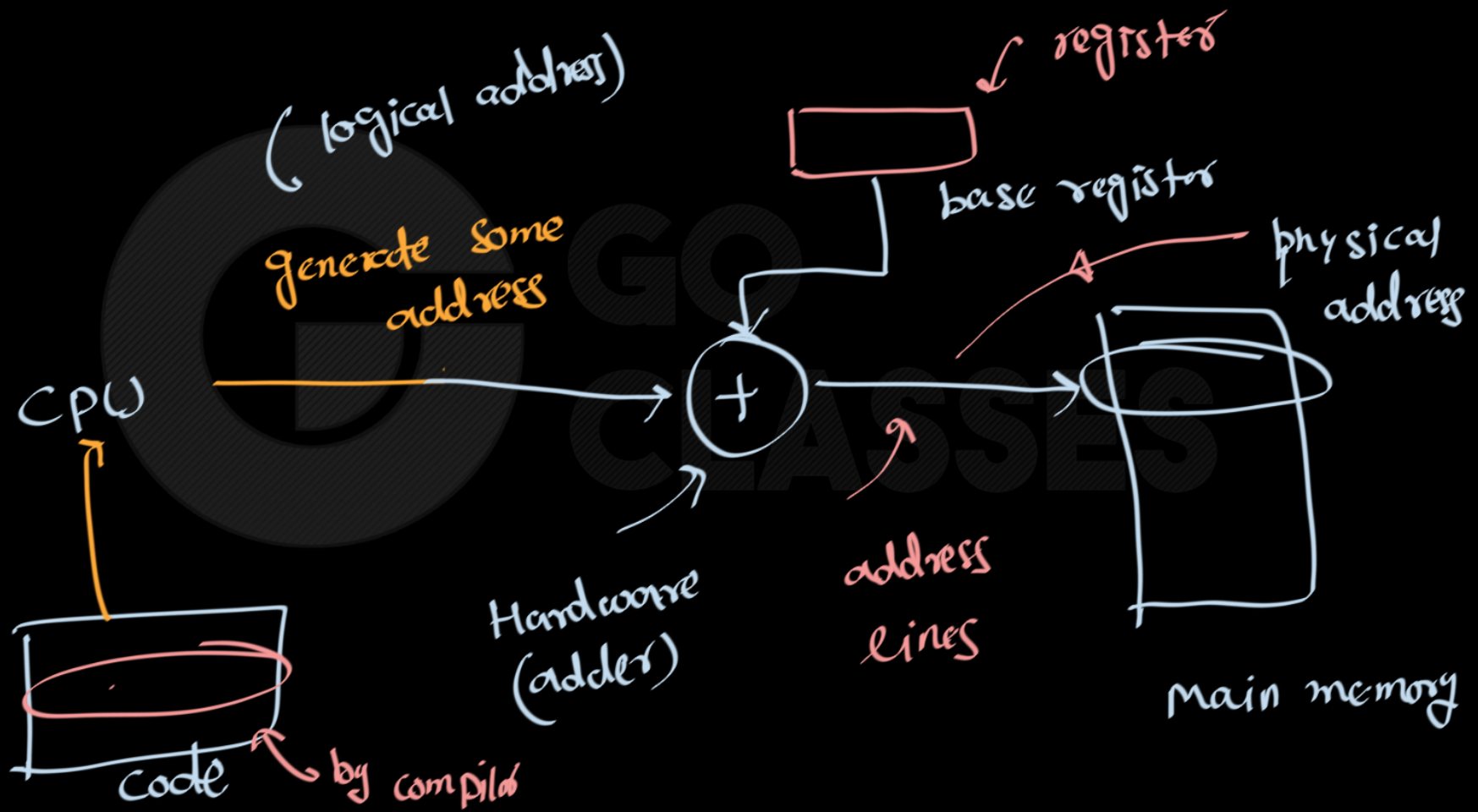
→ Base address is maintained in H/w register called base register

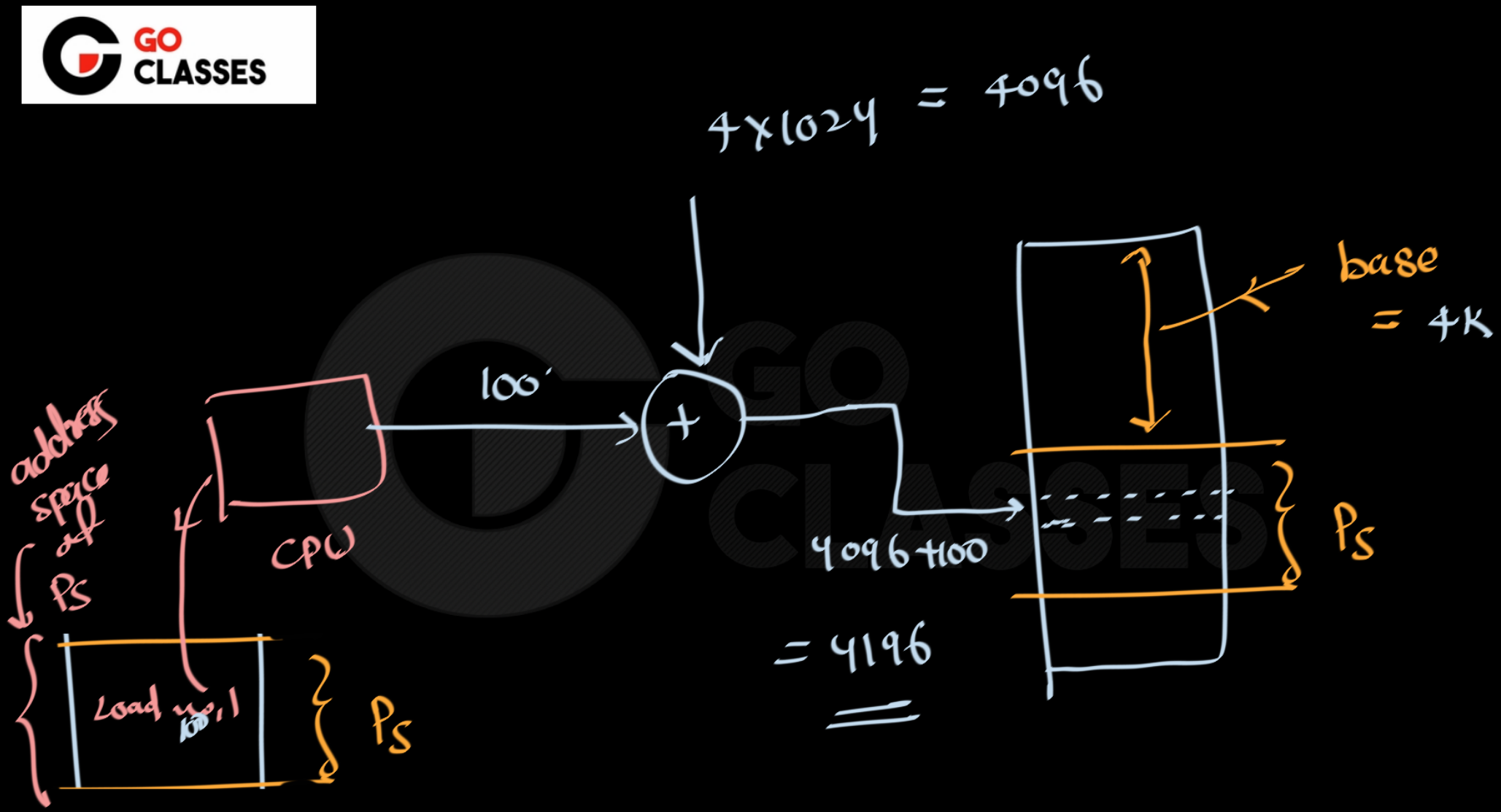
→ we can store process base address into PCB.

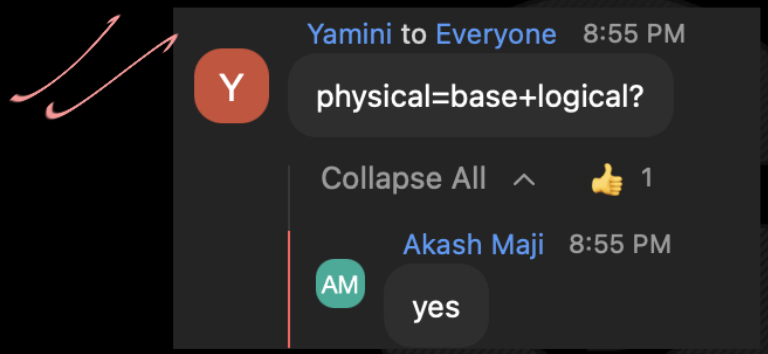
→ at time of context switch we will (just like other registers) load base register value corresponding to the particular process

GO CLASSES

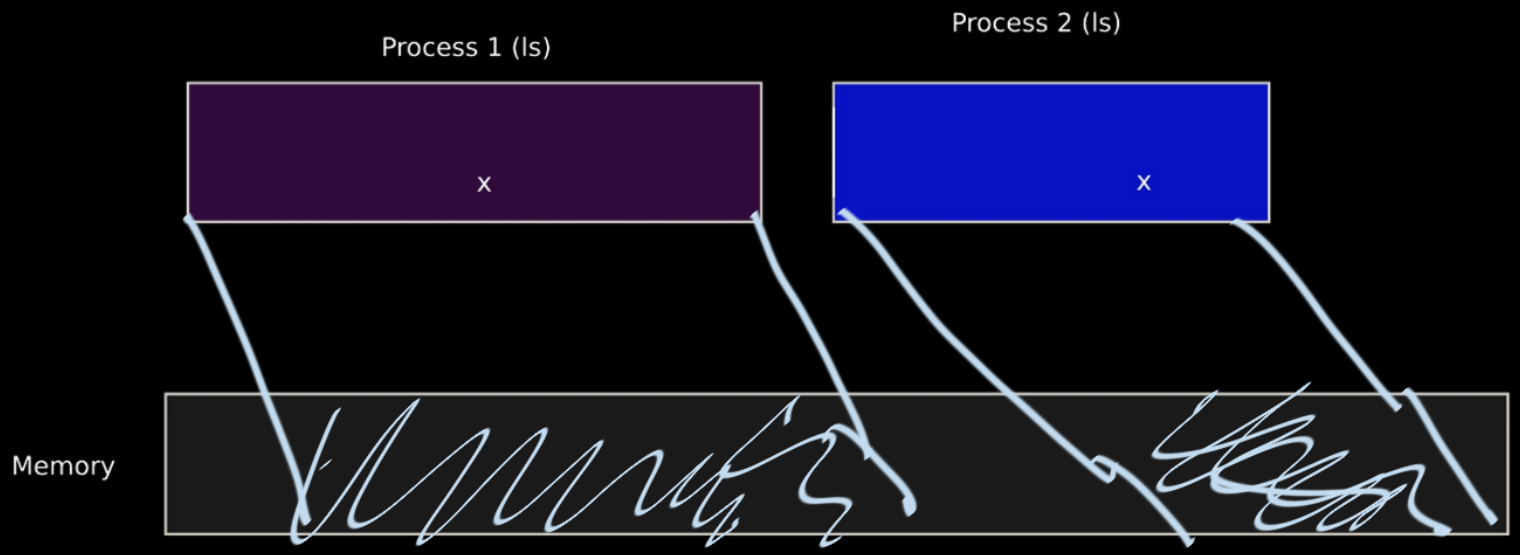
Address TRASLATION IN BASE Setup




4x1024 = 4096
base
= 4K
100
+
CPU
addressing space of Ps
Load 100,1
Ps
Ps
4096+100
= 4196


Yamini to Everyone 8:55 PM
Y
physical=base+logical?
Collapse All
1
Akash Maji 8:55 PM
AM
yes

Two processes, one memory?



Logical vs. Physical Address

- Logical address –

  Generated by the CPU.

  A programmers sees this.

  (`printf("%p", &a)` prints logical address)

- Physical address –

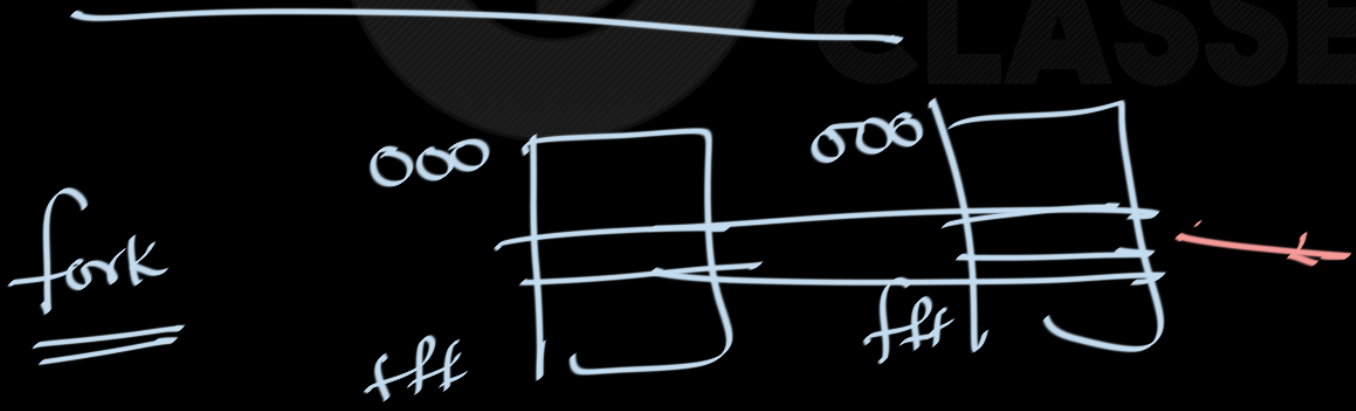
  Address sent to the RAM (or ROM, or IO) for a read or write operation

The user program deals with logical addresses; it never sees the real physical addresses

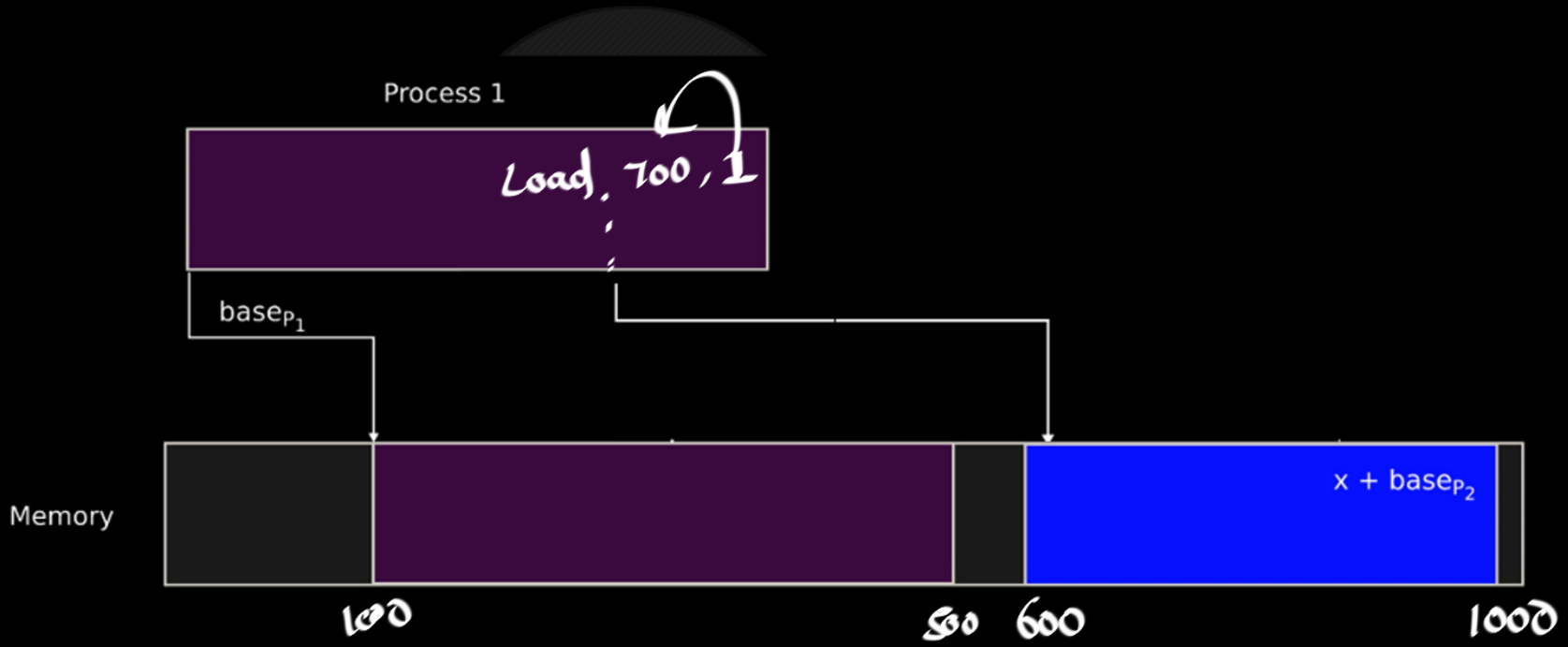
A note -

For two different programs in Main Memory at the same time, logically there addresses could be same.

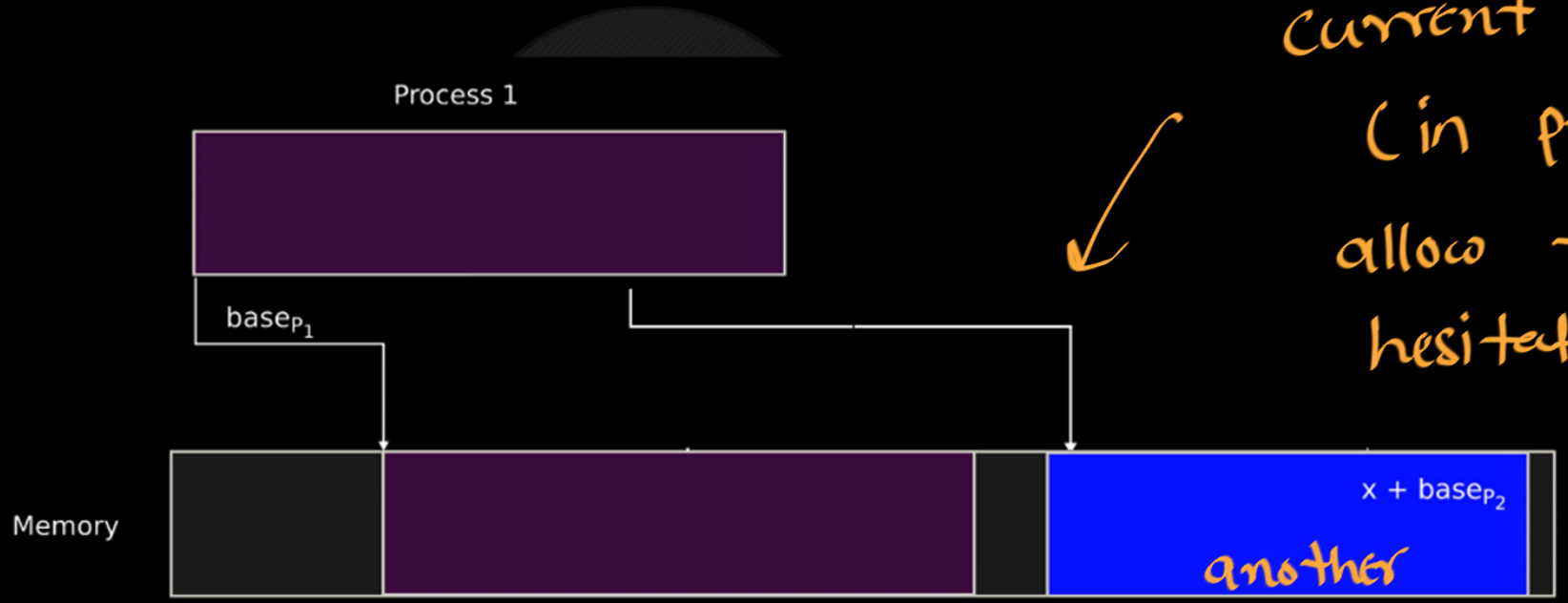
But there physical address will be different



What if Process1 generate some illegal address ?



What if Process1 generate some illegal address ?



current model (in prev slide) allow this without hesitation but we should not allow this

Memory Management Challenges

- Locating data in memory
  - Where is each processes' data located in memory?
- Protection
  - Processes should not be able to read or write each others' memory
  - Processes should not be able to corrupt OS memory
- Efficiency
  - Should support many processes at once

Base and Bound

(limit register)

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

Base and Bound

(limit register)

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

Process Size = bound (limit)

Process Size

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

bound − 1 = limit − 1

bound = limit

GO CLASSES

base = 1036

Process size = 100K = 102400

logical address range = [0, 102400)

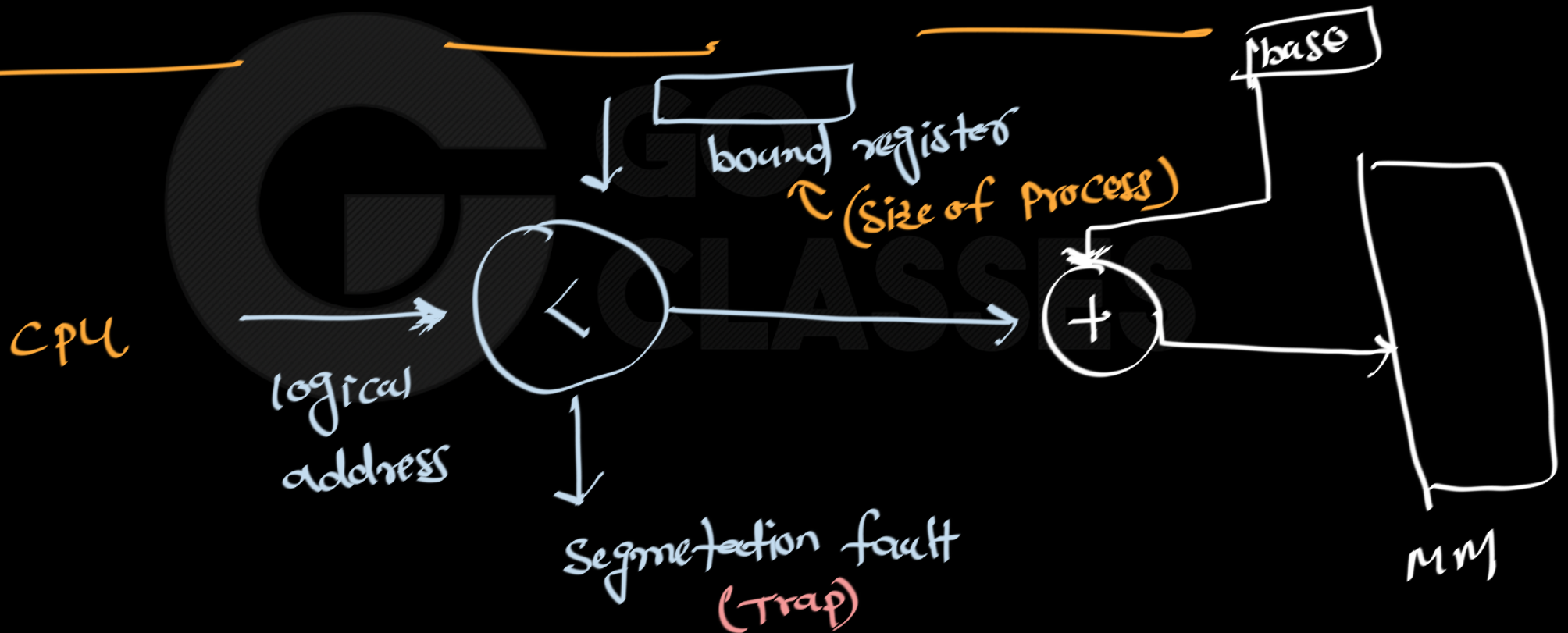
Physical address range = [1036, 1036 + 102400)

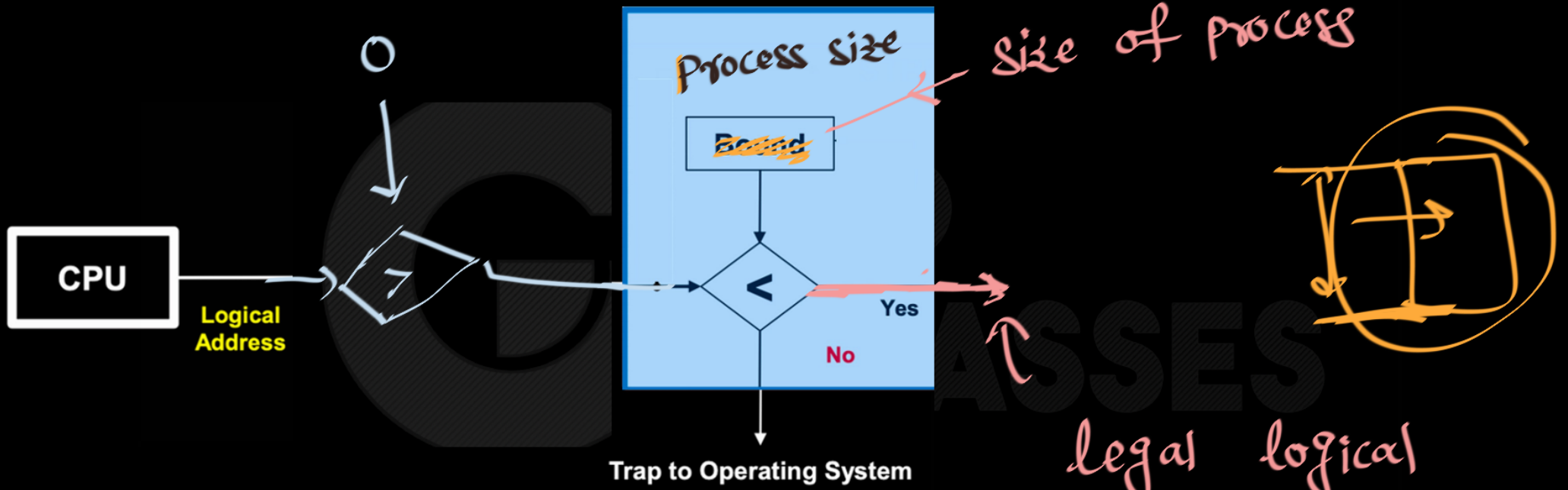
Base and Bound

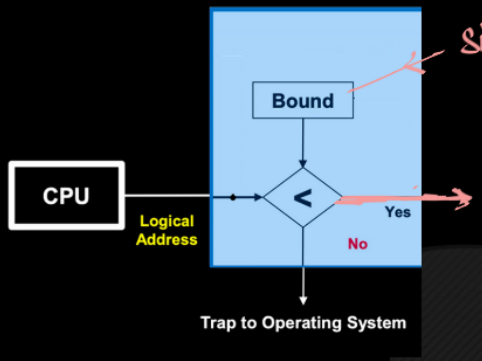
- 2 hardware registers: base and bound
- A process can only access physical memory in [base, base+bound)
- On a context switch:
  - Save/restore base and bound registers

GO CLASSES

Address Translation in Base Bound Setup




0
Process size
Size of process
CPU
Logical Address
<
Yes
No
Trap to Operating System
legal logical address
(protection is taken care)



size of process

base ≡ 1036

Process size = 100K = 102400

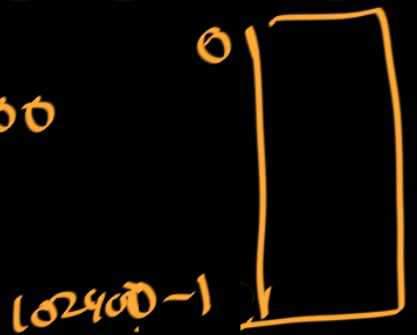
logical address range = (0, 102400)

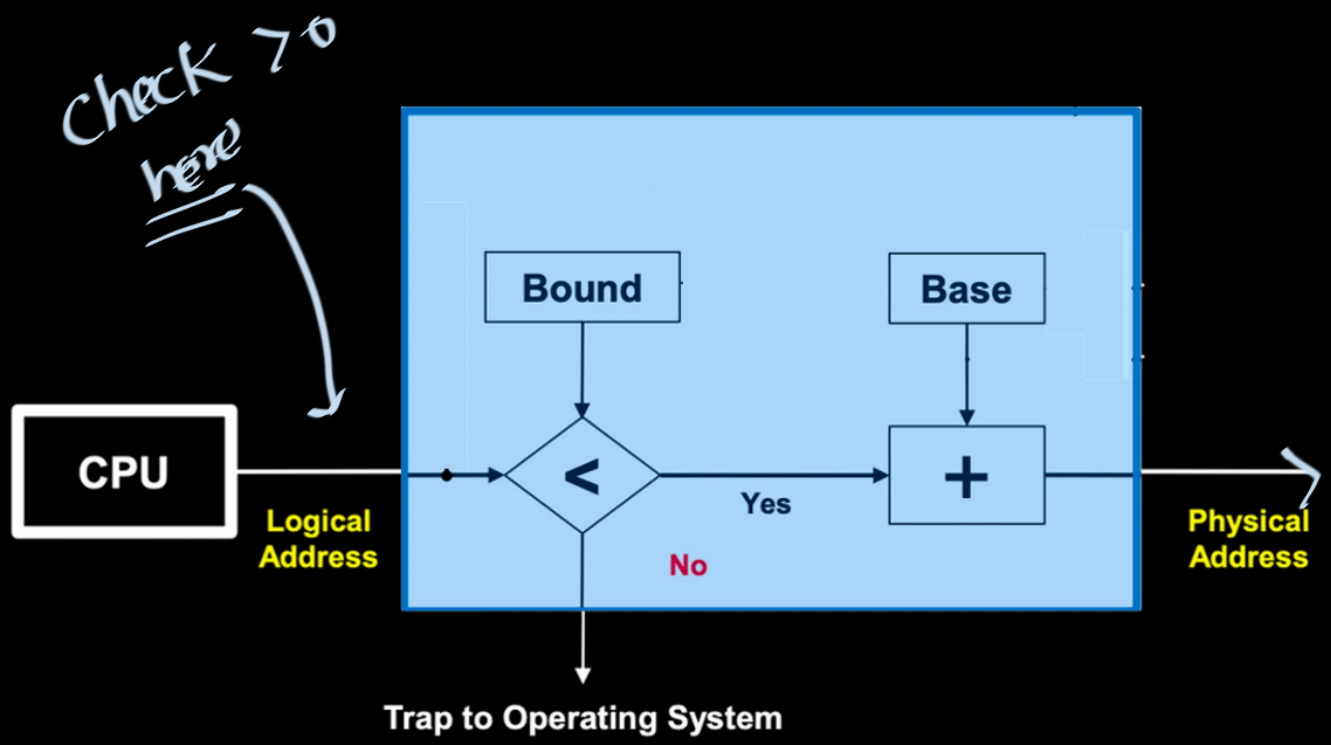
Physical address range = [1036, 1036 + 102400)

Bound reg value ≡ 100K = 102400

[0, 102400)

↳ 102399




Check here
Bound
Base
CPU
<
+
Logical Address
Yes
No
Physical Address
Trap to Operating System

Quiz: Who Controls Base Register?

- What entity should do the address translation with base register?
  1) User Process
  2) OS
  3) HW

↳ actually using base reg.

- What entity should modify the base register?
  1) User Process
  2) OS
  3) HW

Quiz: Who Controls Base Register?

- What entity should do the address translation with base register?
  1) User Process
  2) OS
  3) HW

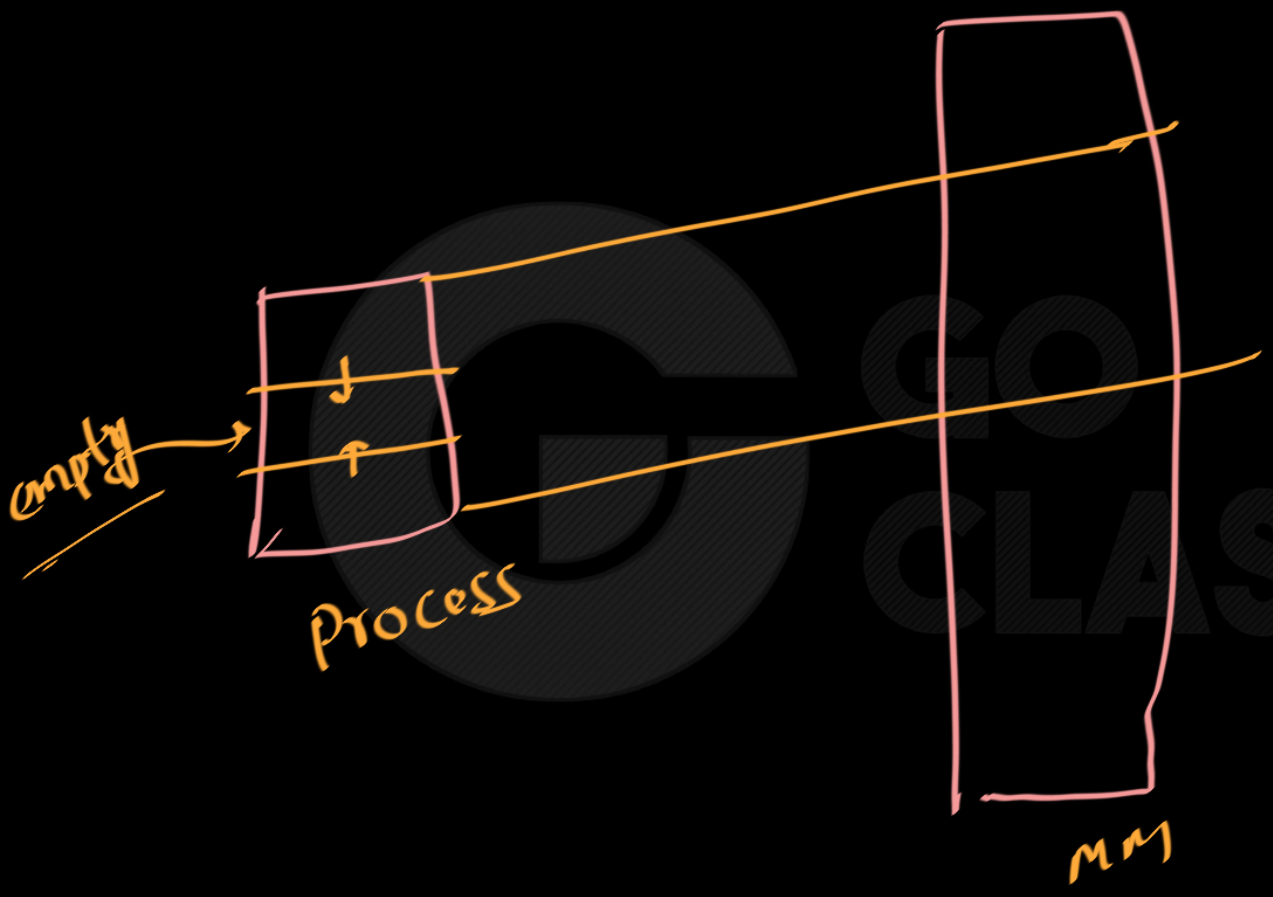
- What entity should modify the base register?
  1) User Process
  2) OS
  3) HW

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

Base+Bounds Advantages

- Provides protection across address spaces
- Supports dynamic relocation (was in Base reg alone too)
- Simple, inexpensive, fast HW implementation
  - Few registers, little logic in MMU → hardware
- Simple context switching logic
  - Save and restore a couple of registers

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


empty
J
f
Process
MM

Base+Bounds Disadvantages

Meeting Chat

Arturo Gangwar to Everyone 9:25 PM

AG

problem might be here is that we are forced to moved whole process to MM

963 2319 4199 to Everyone 9:26 PM

92

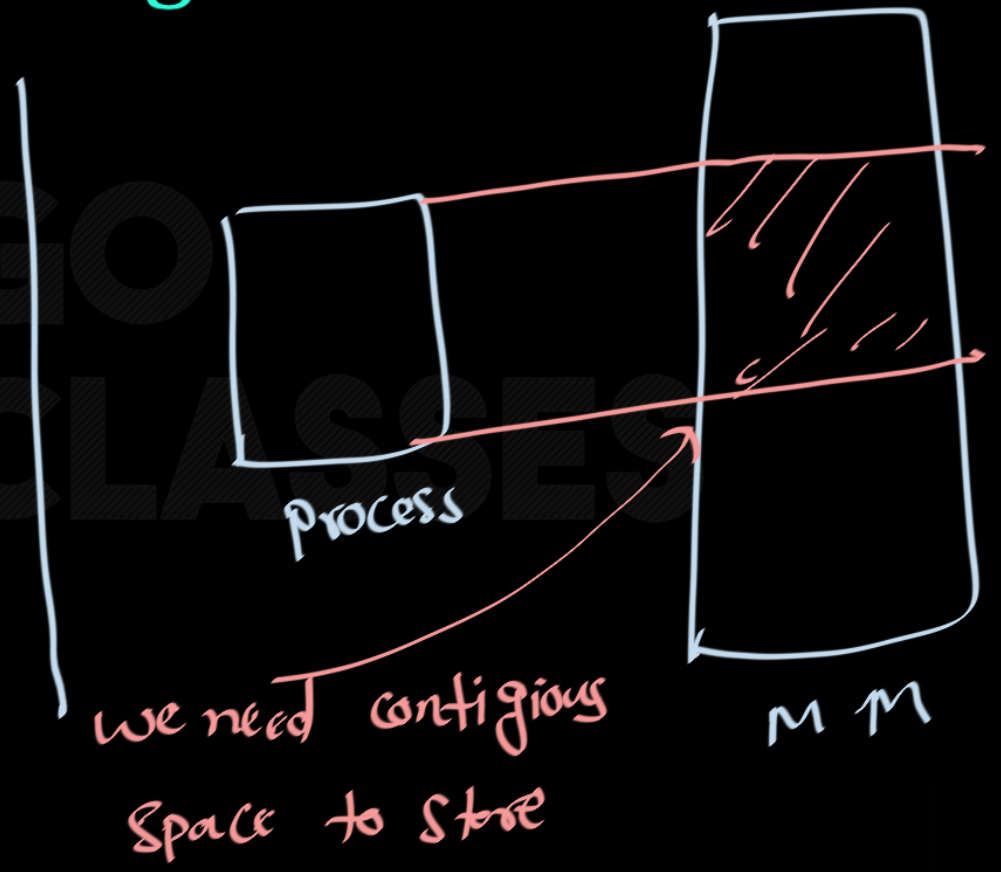
process size is fixed

Akash Maji to Everyone 9:26 PM

AM

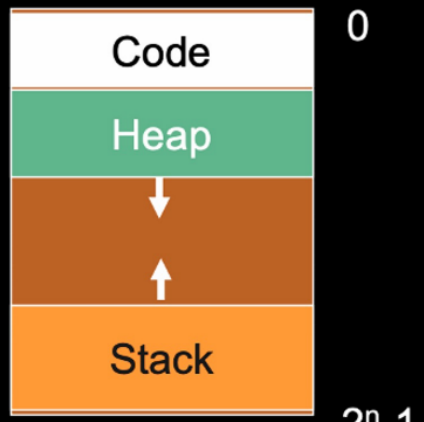
management of base and limit values by OS

must know process sizes



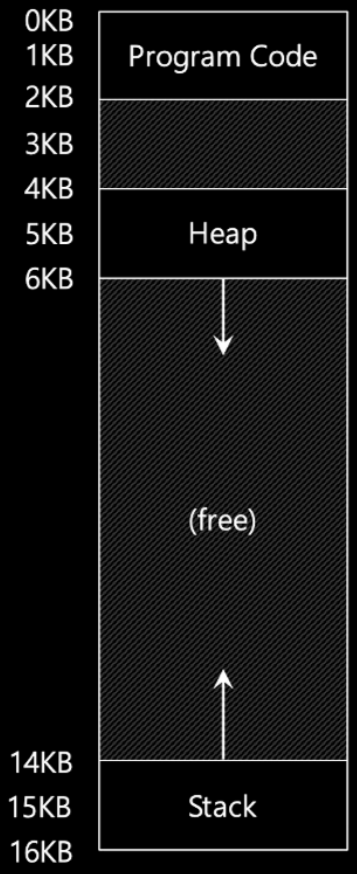
Base+Bounds Disadvantages

- Each process must be allocated contiguously in physical memory



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

Why not just Base and Bound?



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

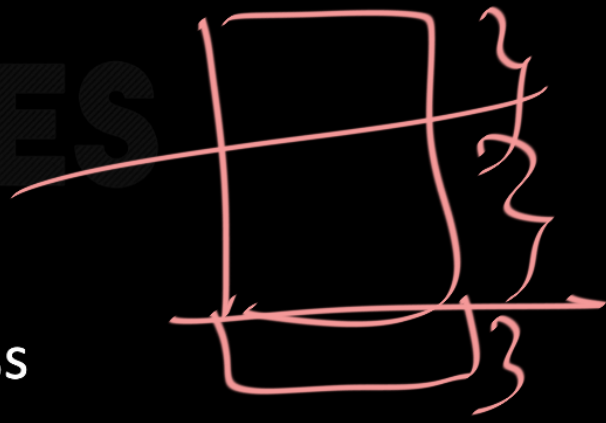
Segmentation

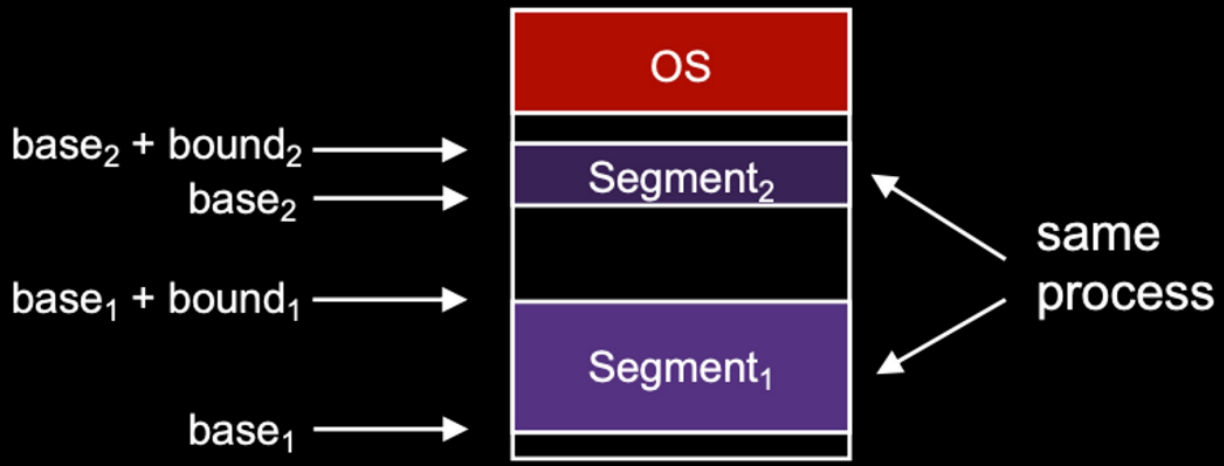
we are NOT (currently) saying hard disk

↳ we will divide into some chunks and store chunks independently in MM

Segmentation

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



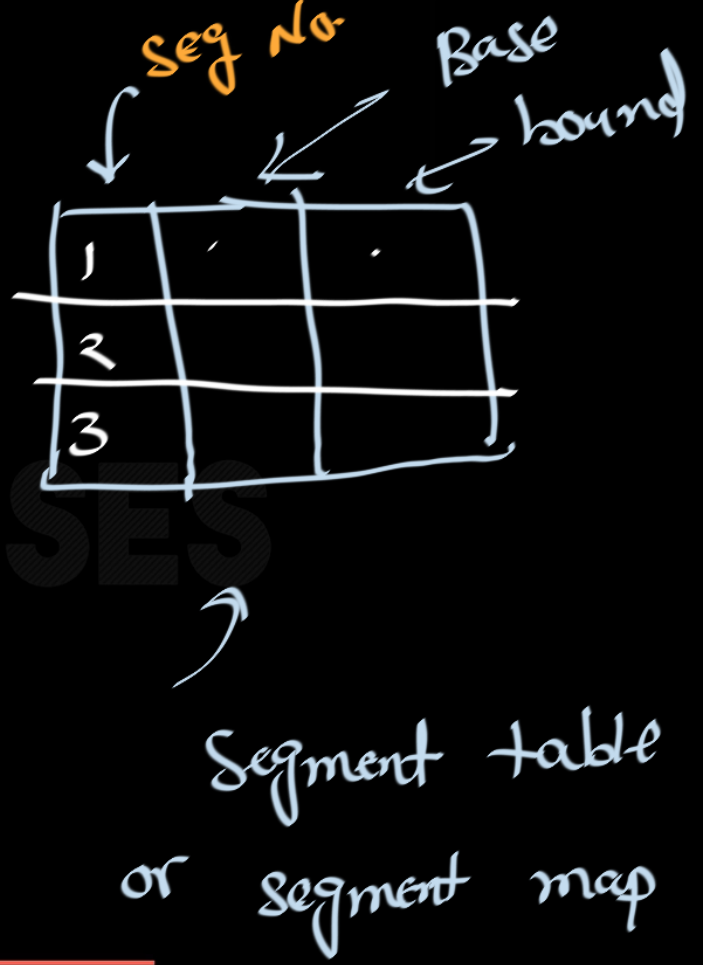


multiple Segments for each process, and one pair of base and bound for each Segment

Segmentation

- Segment map
  - One per process
  - Holds base and bounds registers, permissions for each segment
- Context switch
  - Save/restore table or pointer to table in kernel memory

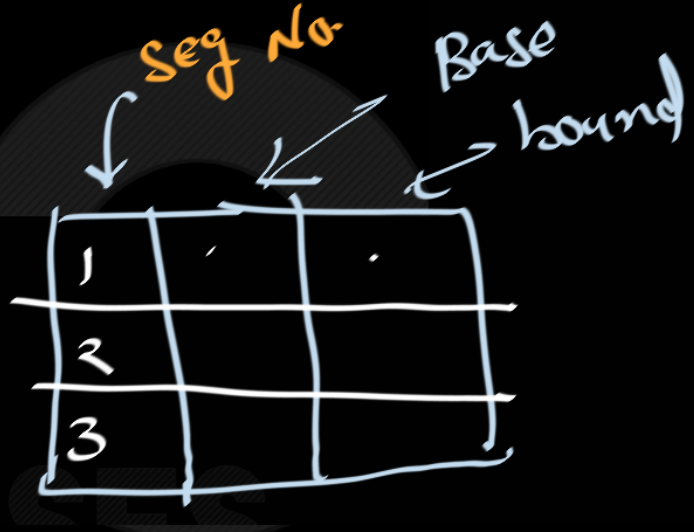
STBR



GO CLASSES

STBR ← for every process value will be different

Segment table base register



Store this in main memory

Operating Systems

Akash Maji to Everyone 9:42 PM

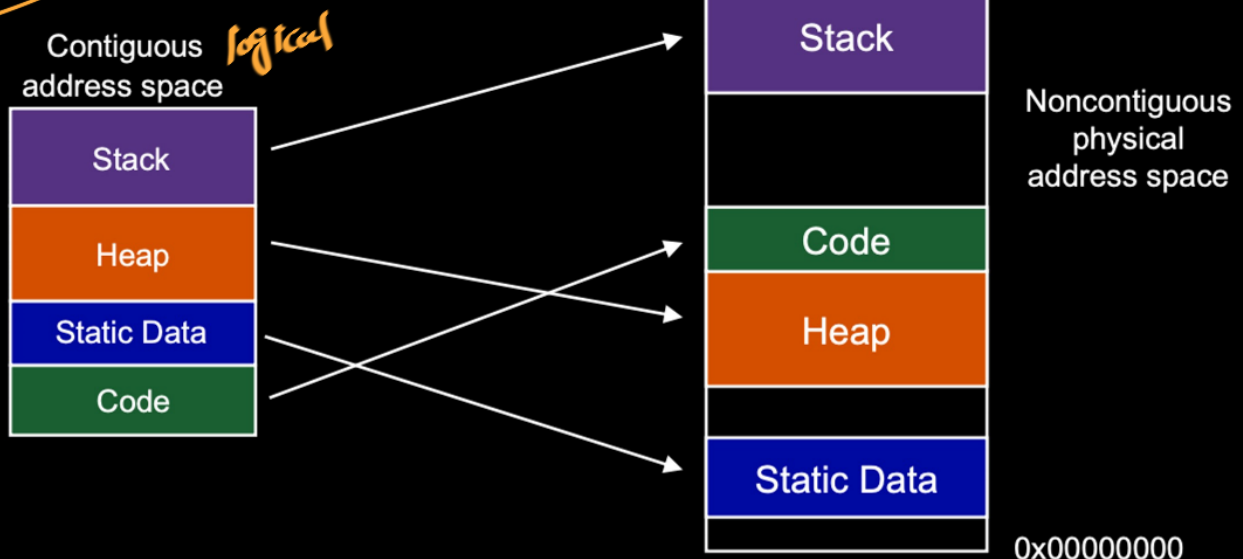
AM some fixed format of executable code

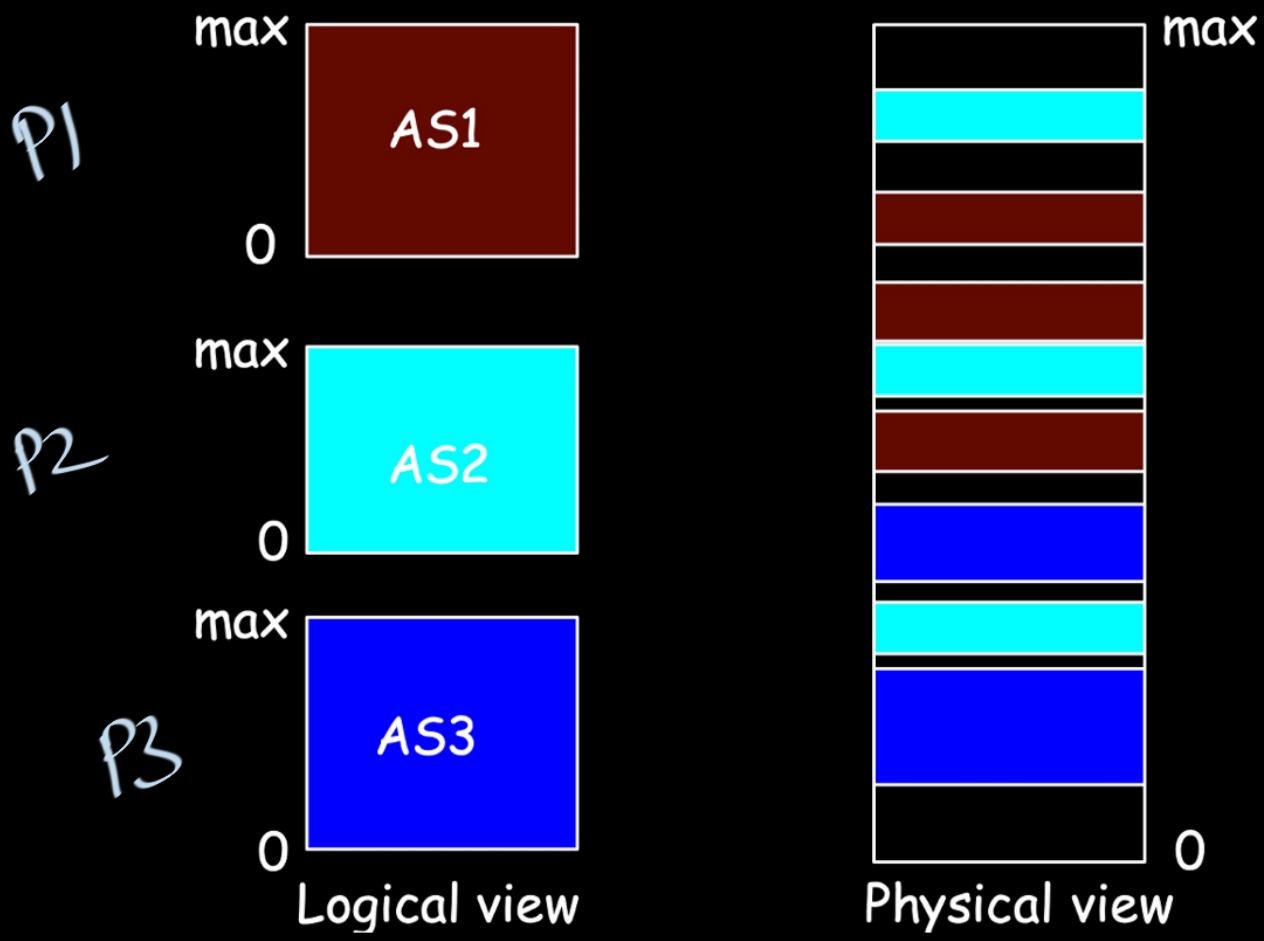
Akash Maji to Everyone 9:43 PM

AM .data
.bss
.text

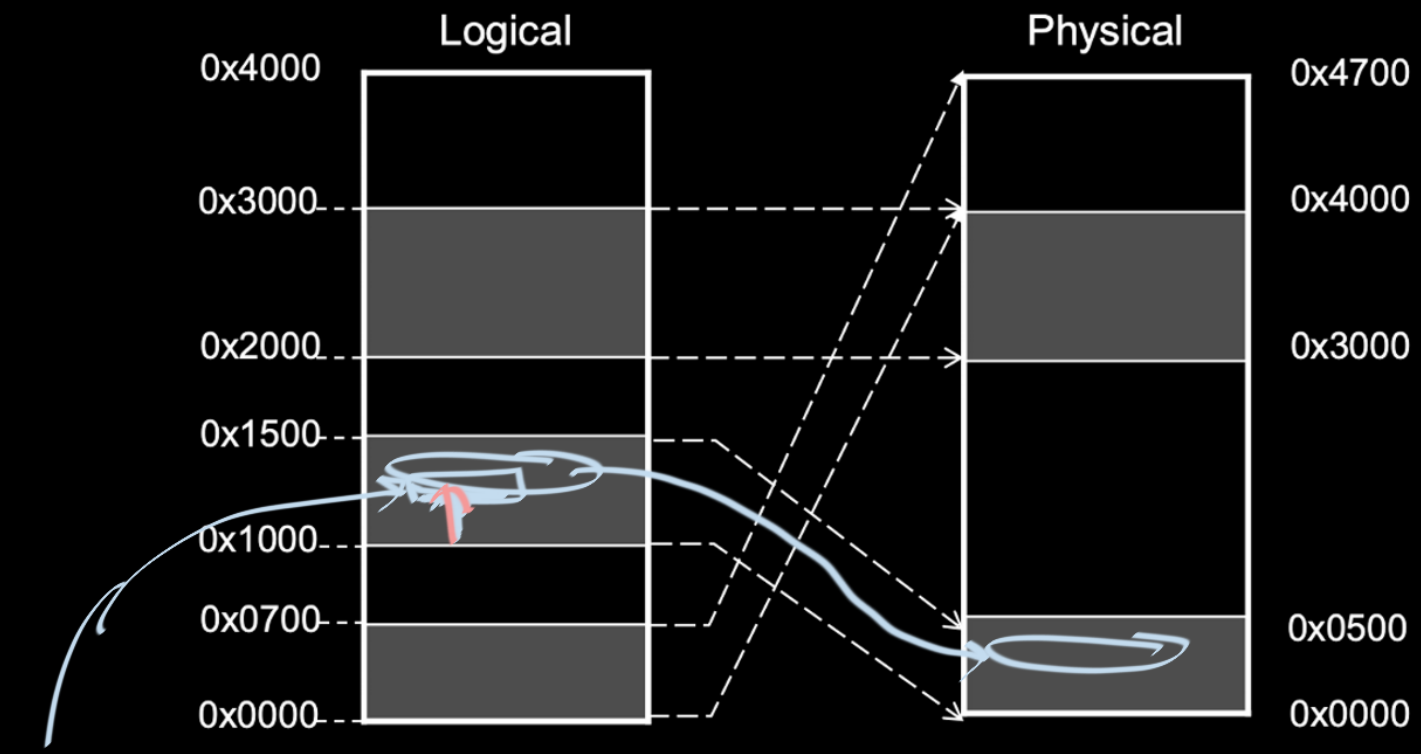
something like that

a.out has fixed format

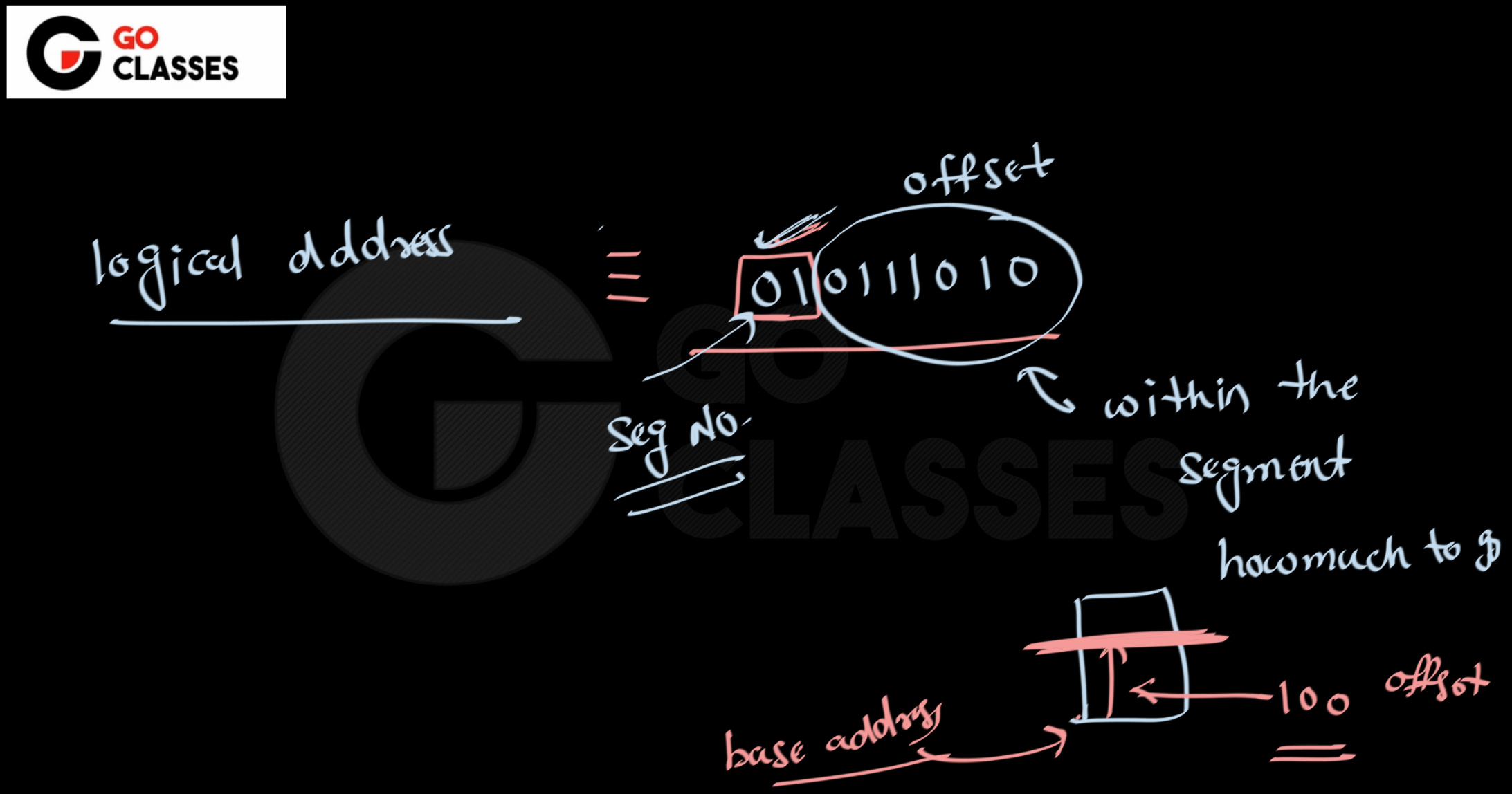



max
AS1
0
P1
max
AS2
0
P2
max
AS3
0
P3
Logical view
max
0
Physical view

Address space mapping

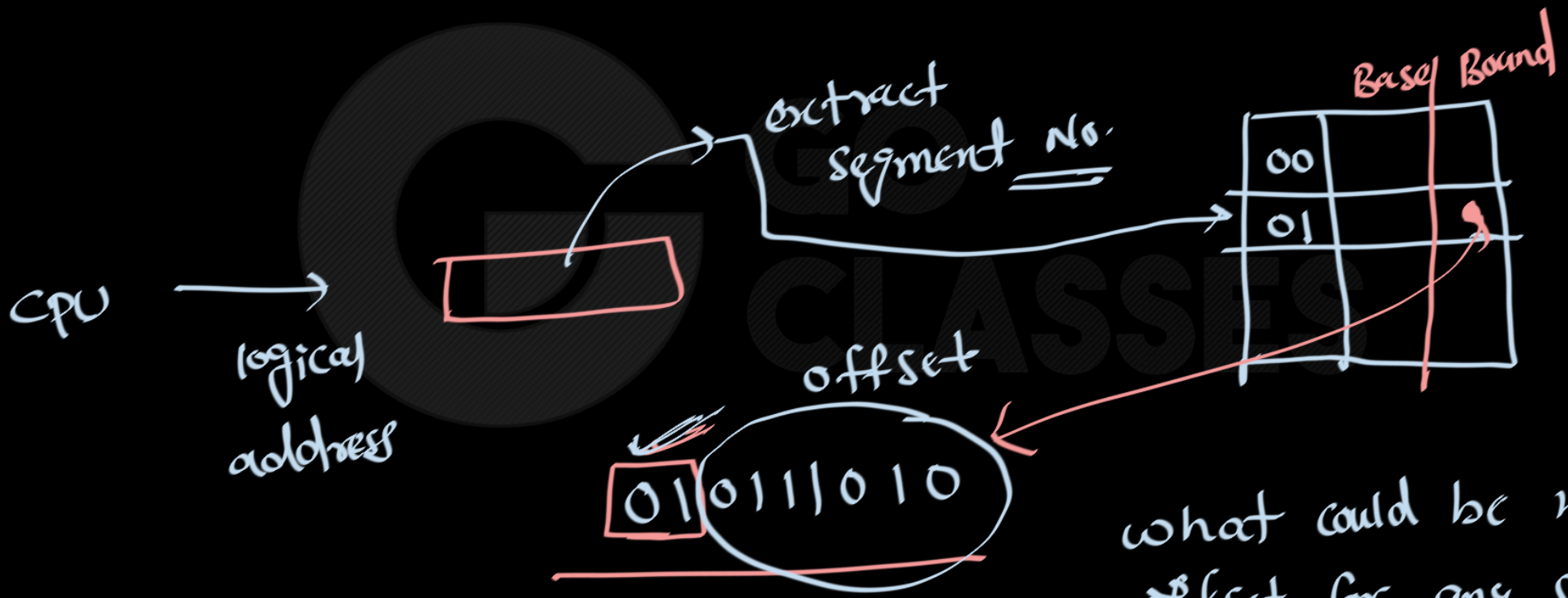


0x1200 = 0001 0010 0000 0000

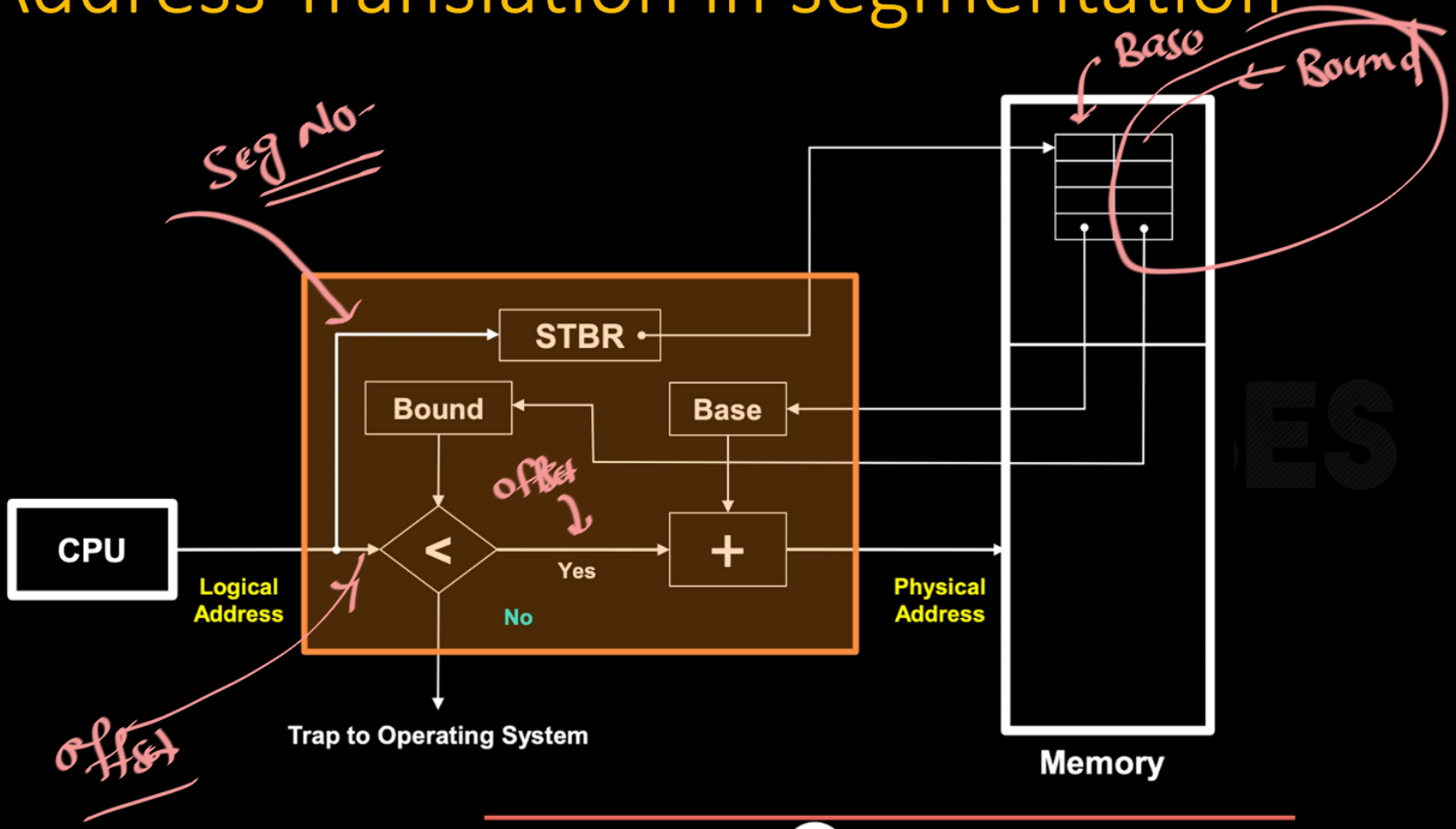

logical address
offset
01 0111010
Seg no.
within the segment
how much to go
base address
100
offset

GO CLASSES

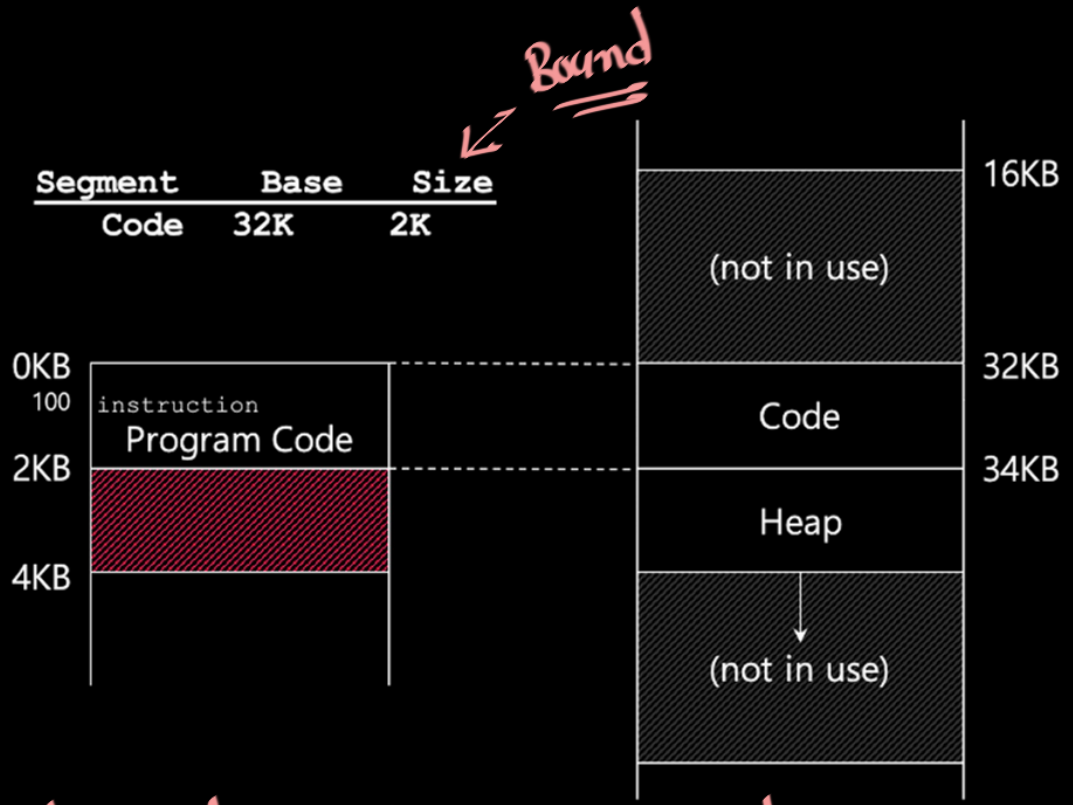
Address translation in Segmentation

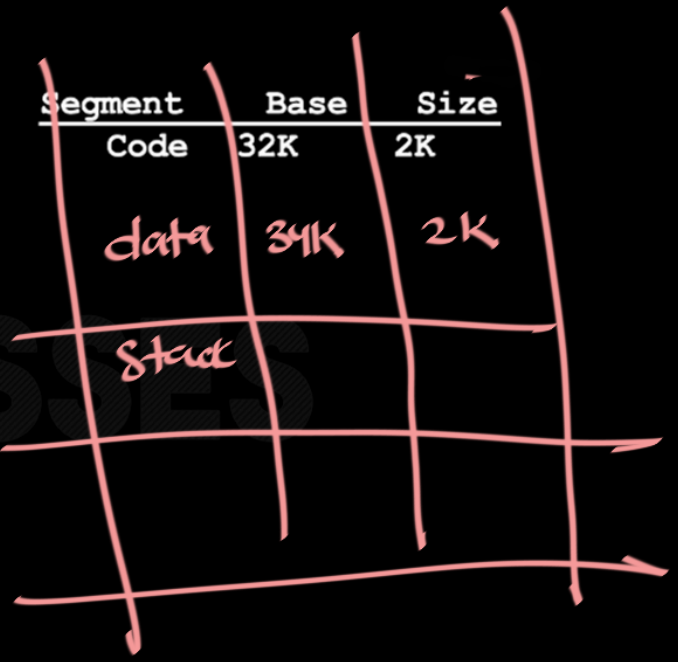


Address Translation in segmentation



Example 1 : Address Translation ..

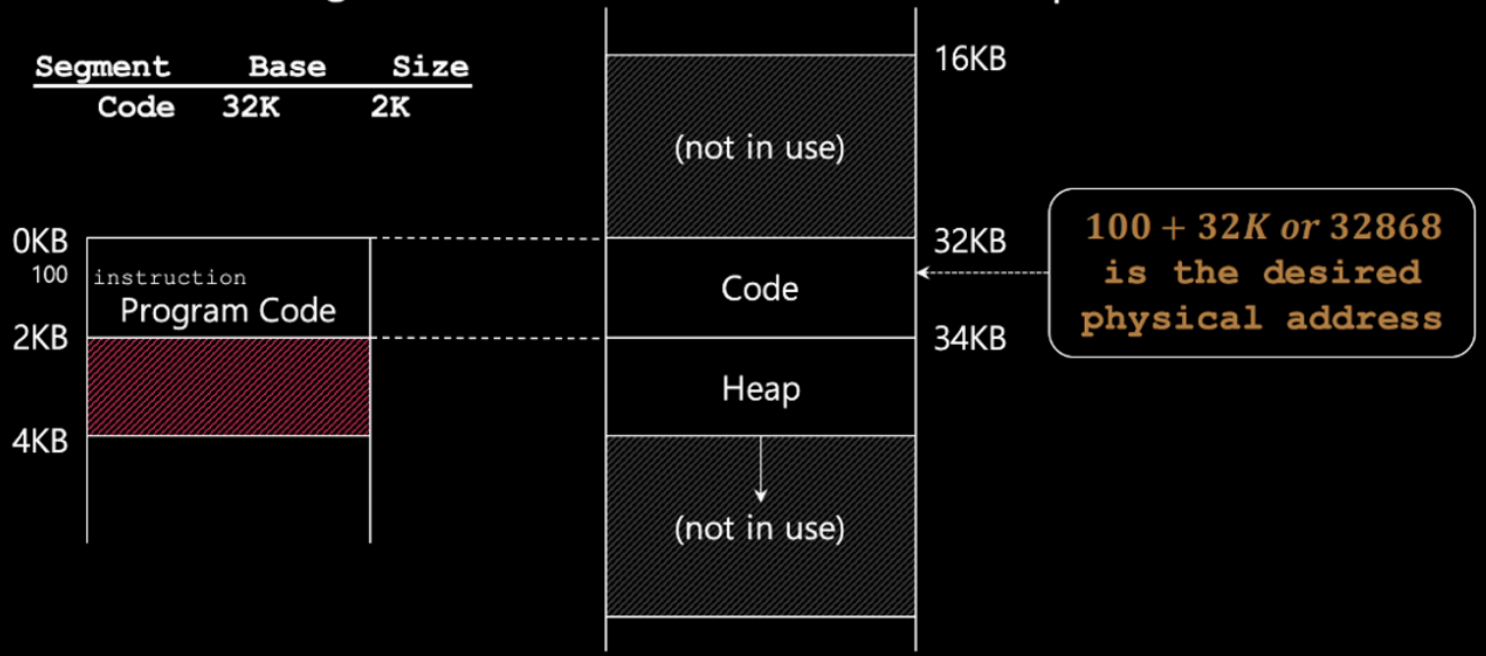




find out PA corresponding to 100?

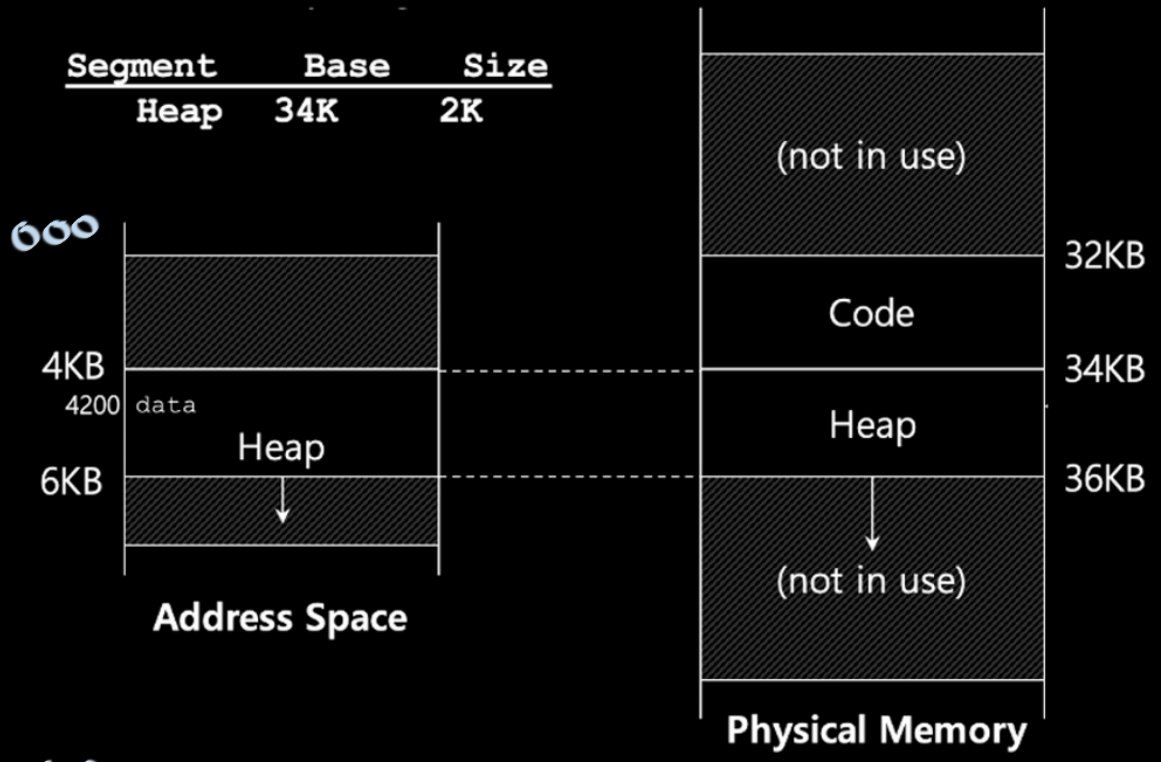
Seg table

- The `offset` of 100 is 100.
  - The code segment **starts at address 0** in address space.



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

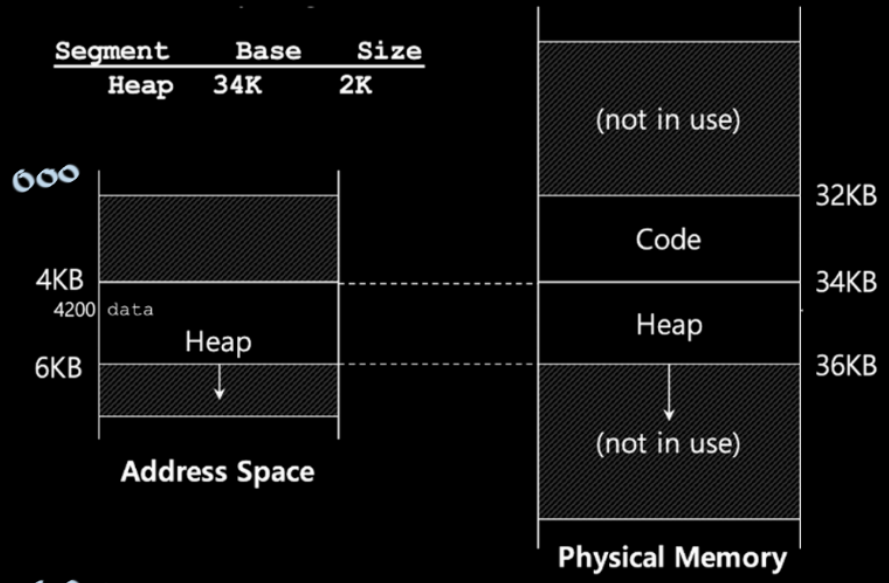
Example 2 : Address Translation ..



PA for 4200

34K +

Example 2 : Address Translation ..



base

offset

34K + 104

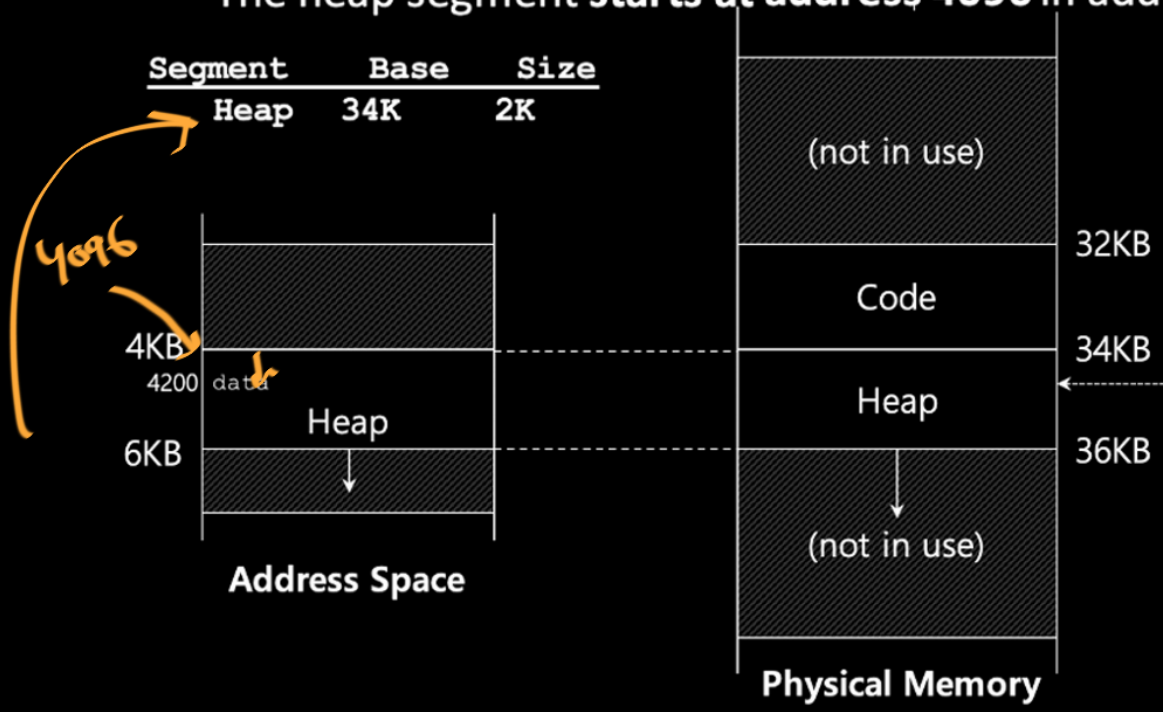
PA for 4200

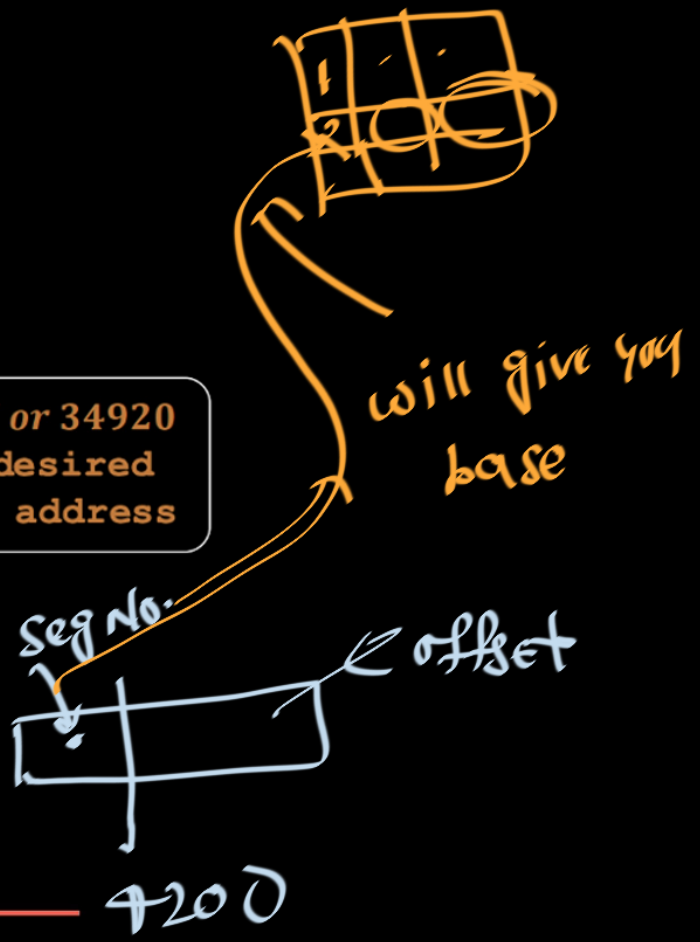
offset : 4X1024 = 4096

offset

4200 - 4096 = 104

- The `offset` of address 4200 is `104`.
  - The heap segment **starts at address 4096** in address space.



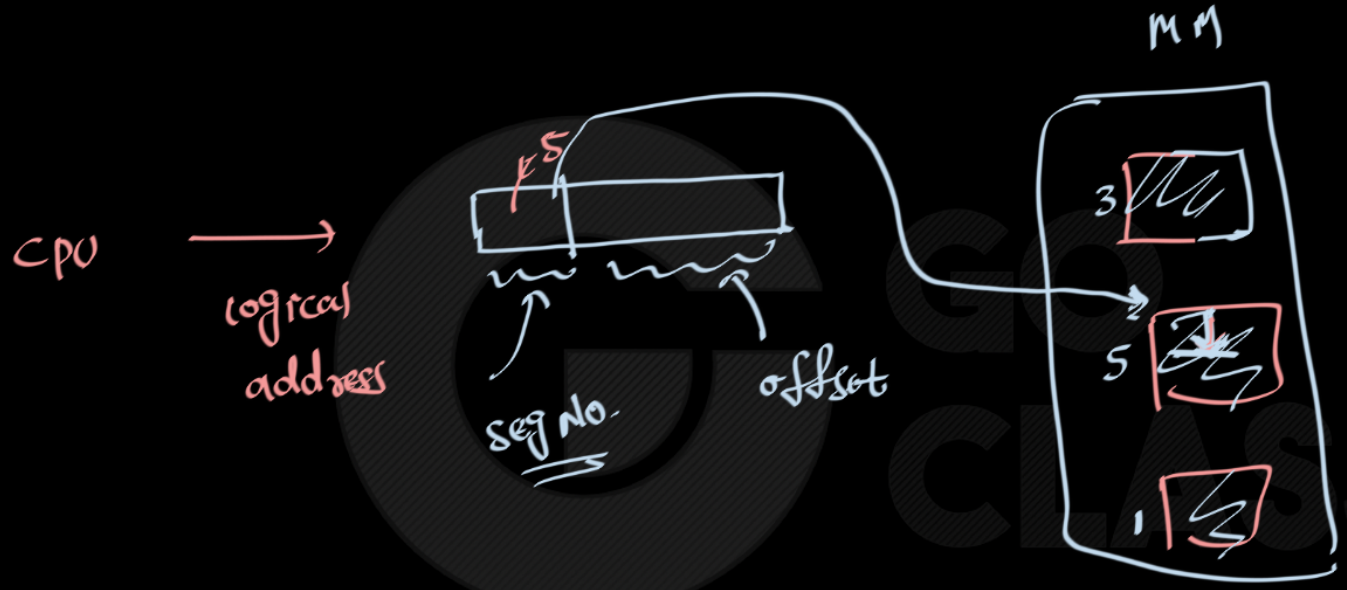


4200 − 4096 = 104 ← offset

Address Translation on Segmentation

$$physical\ address = offset + base$$

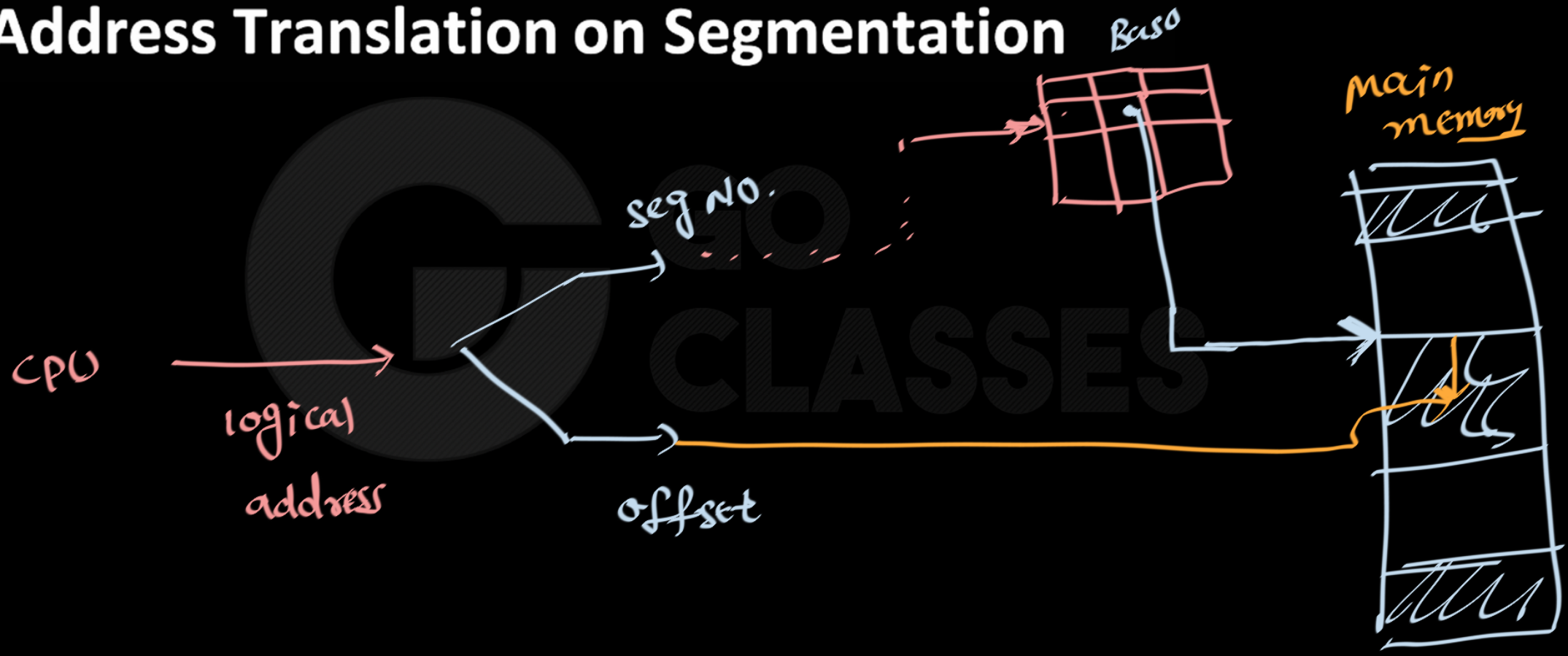
Address Translation on Segmentation



get base address of the particular seg. using seg. table

GO CLASSES

Address Translation on Segmentation



GO CLASSES

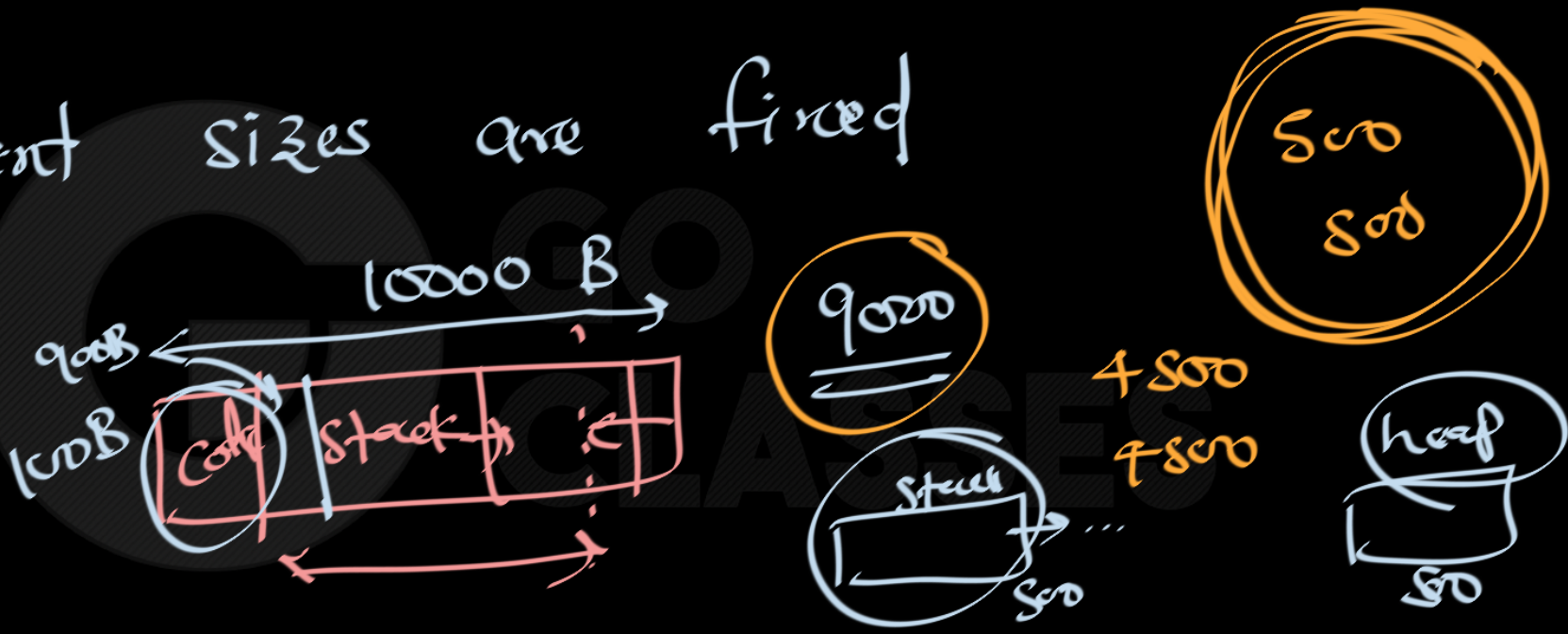
Basic idea of Segmentation

→ we can divide process and store in MM at non contigious location.

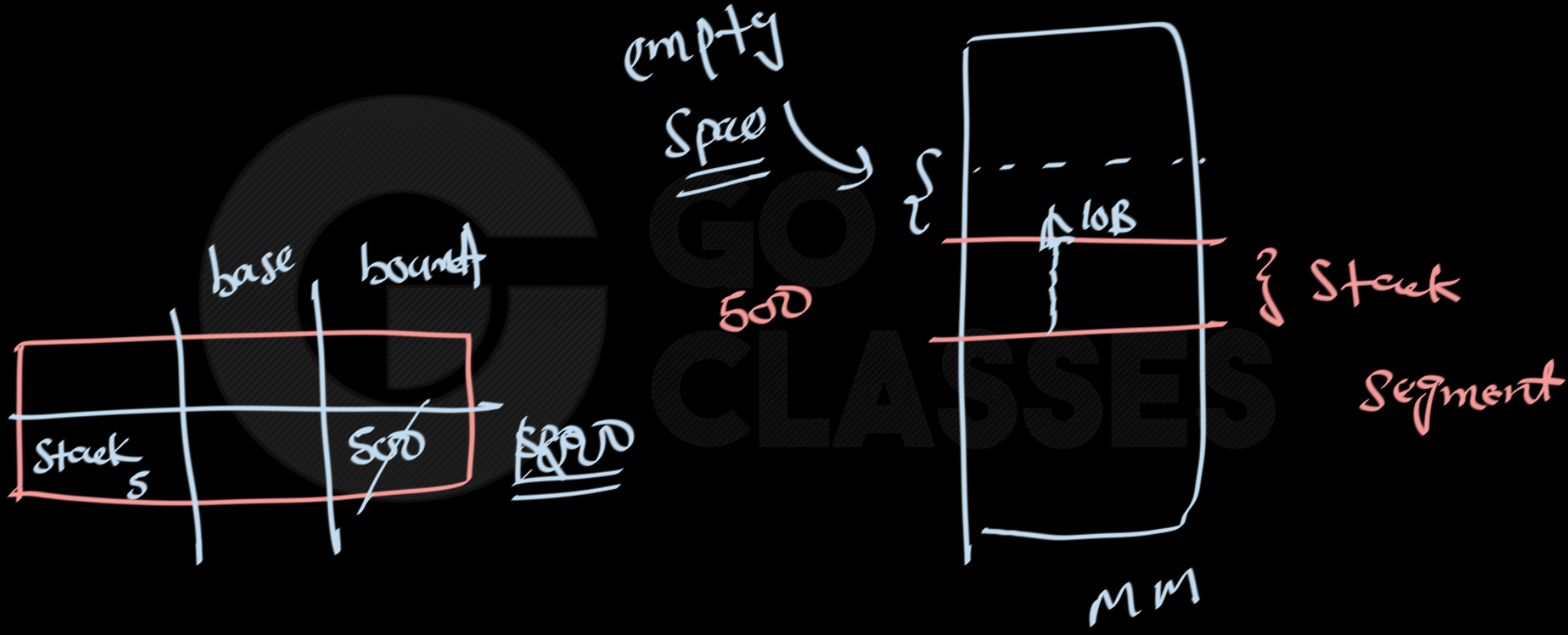
(forget about address translation for now)

GO CLASSES

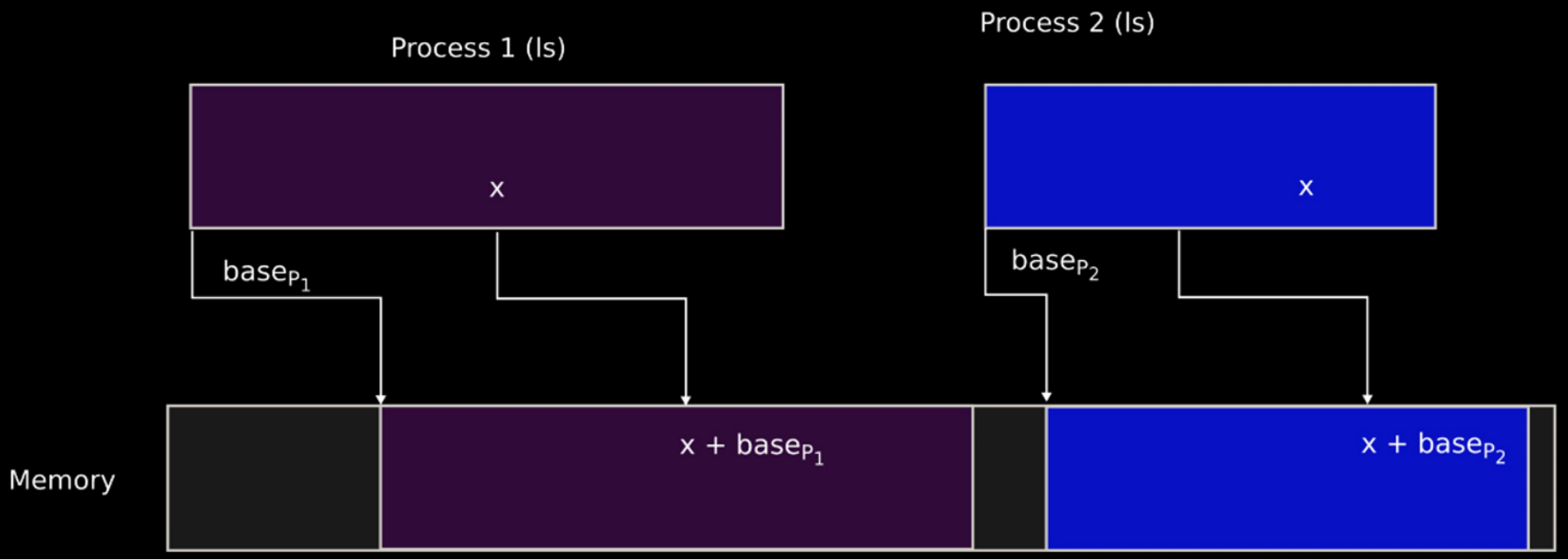
Segment sizes are fixed



at runtime the sizes of segments may change than what we initially thought

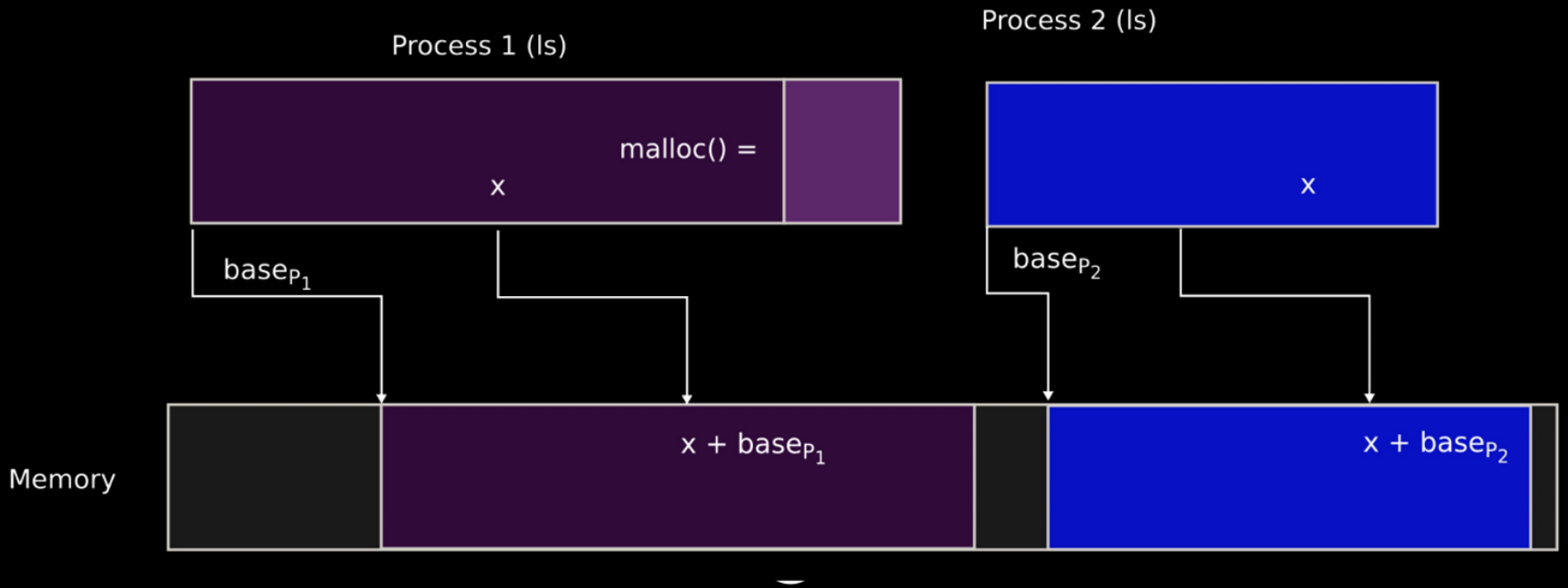

base
bound
Stack
S
500
500
empty
space
500
1OB
Stack
Segment
MM

Segmentation is ok... but

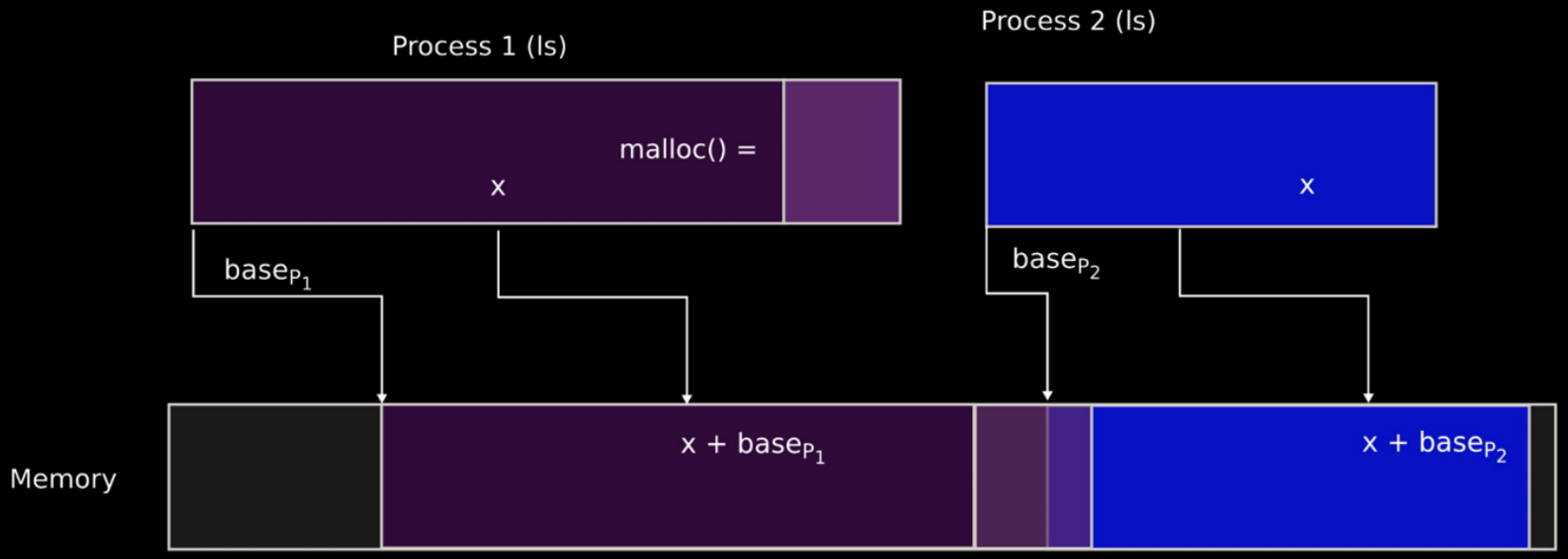


https://users.cs.utah.edu/~aburtsev/238P/lectures/lecture06-address-translation/lecture06-address-translation.pdf

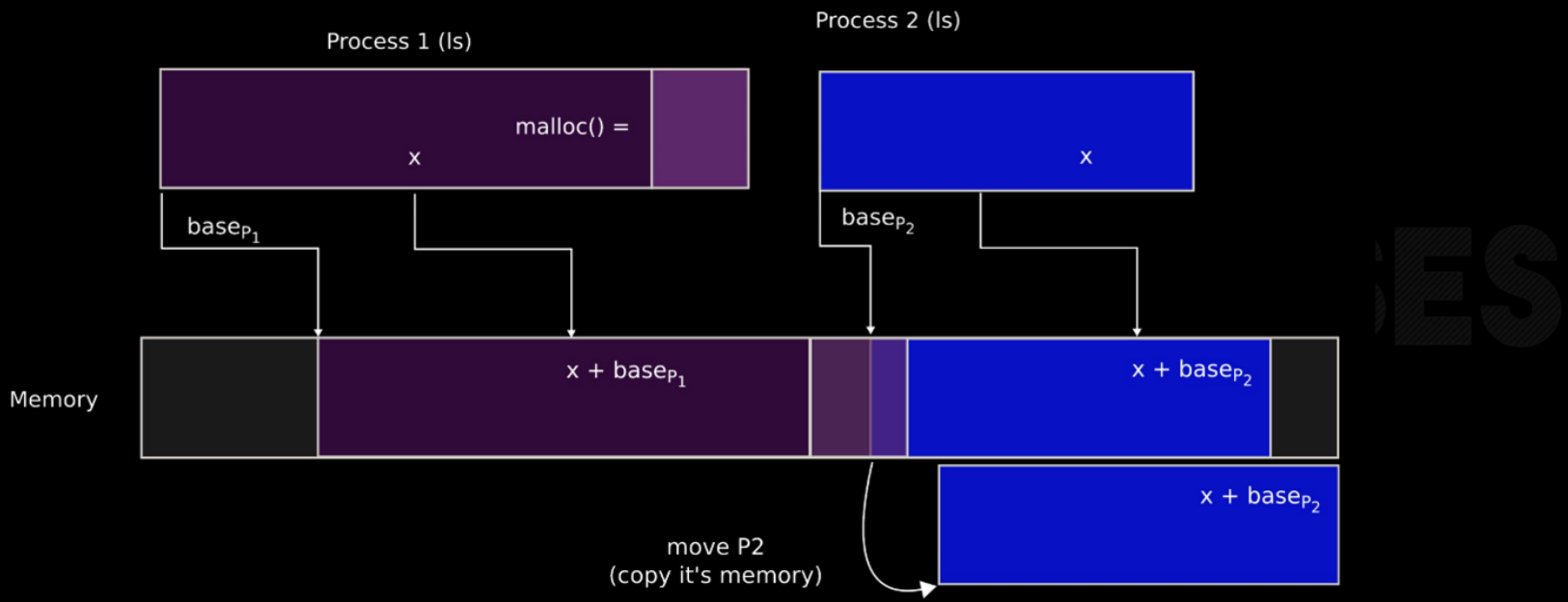
What if process needs more memory?



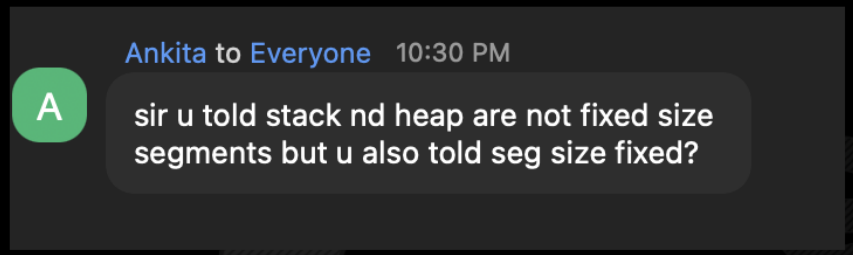
What if process needs more memory?

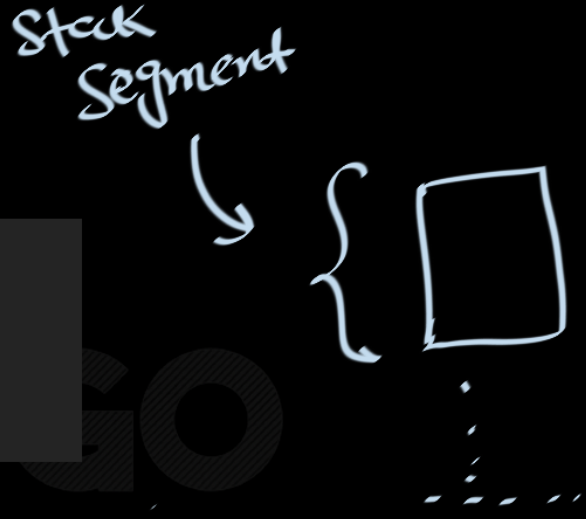


You can move P2 in memory



GO CLASSES





Segments are of fixed sizes

but Stack, heaps are not

→ if there is empty space in mm the just change bound

→ otherwise allocate some prog

Problems with segments

- Segments are somewhat inconvenient
  - Relocating or swapping the entire process takes time

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

Space is available but not contiguously

→ only base reg.

- No protection
- Cont. memory needed
- external frag.

→ Base + Bound

- Protection ✓
- Cont. memory needed
- external frag.

Segmentation

(multiple base and bound)

- Protection ✓
- Cont. mem needed ✓
- external frag.

→ seg. size is fix but heap/stack are not fixed