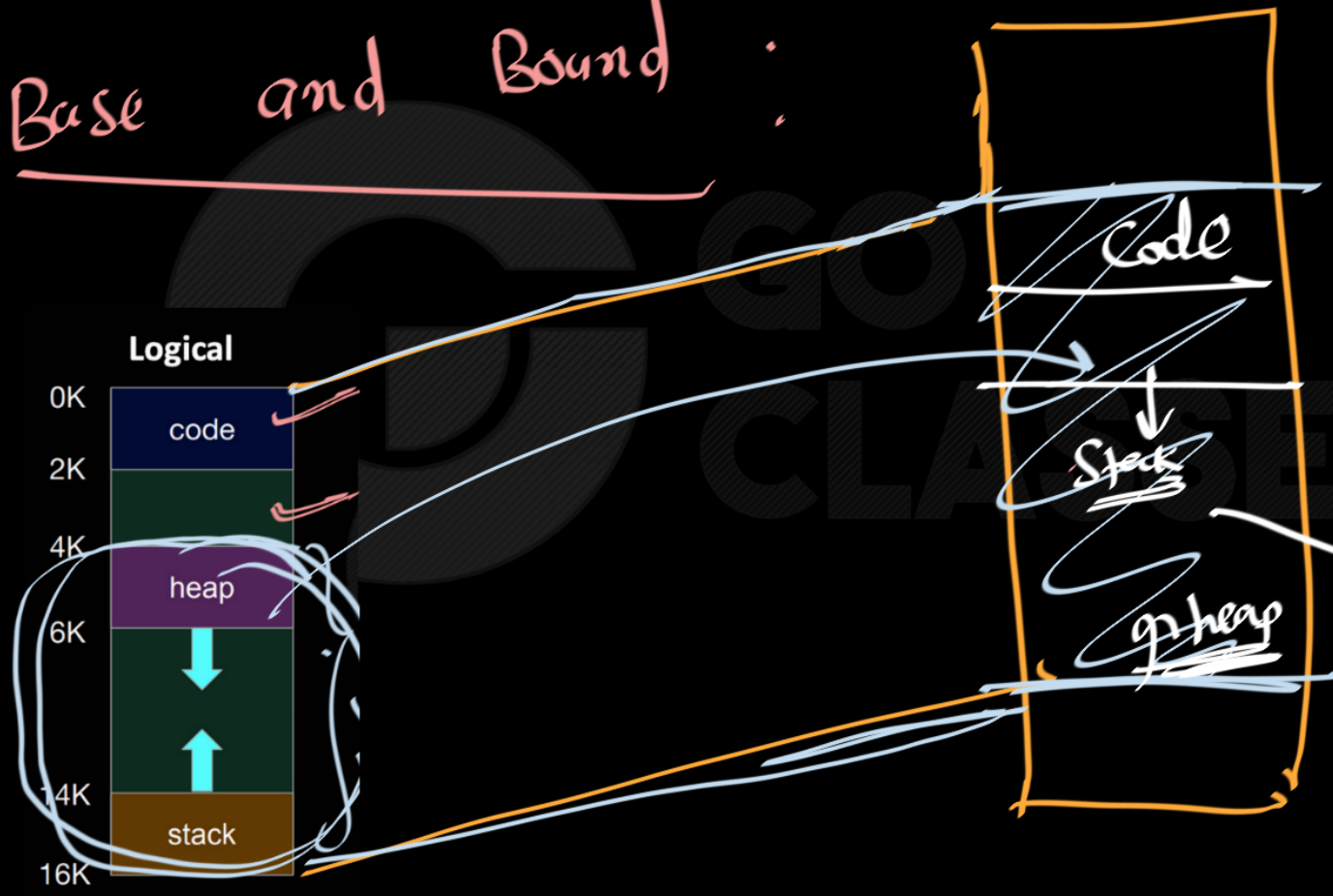
Lecture: 24

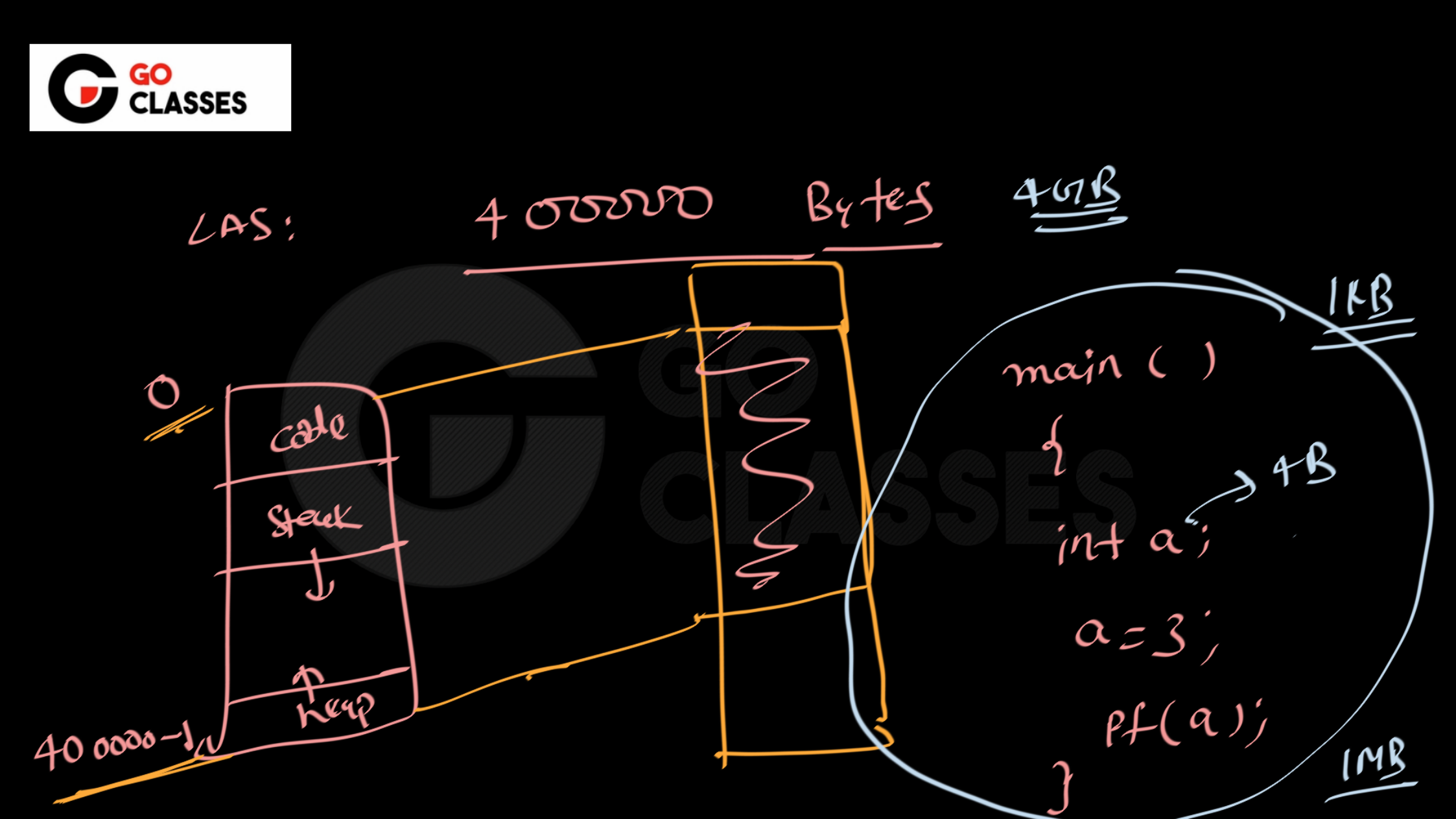
GO CLASSES

Base and Bound :



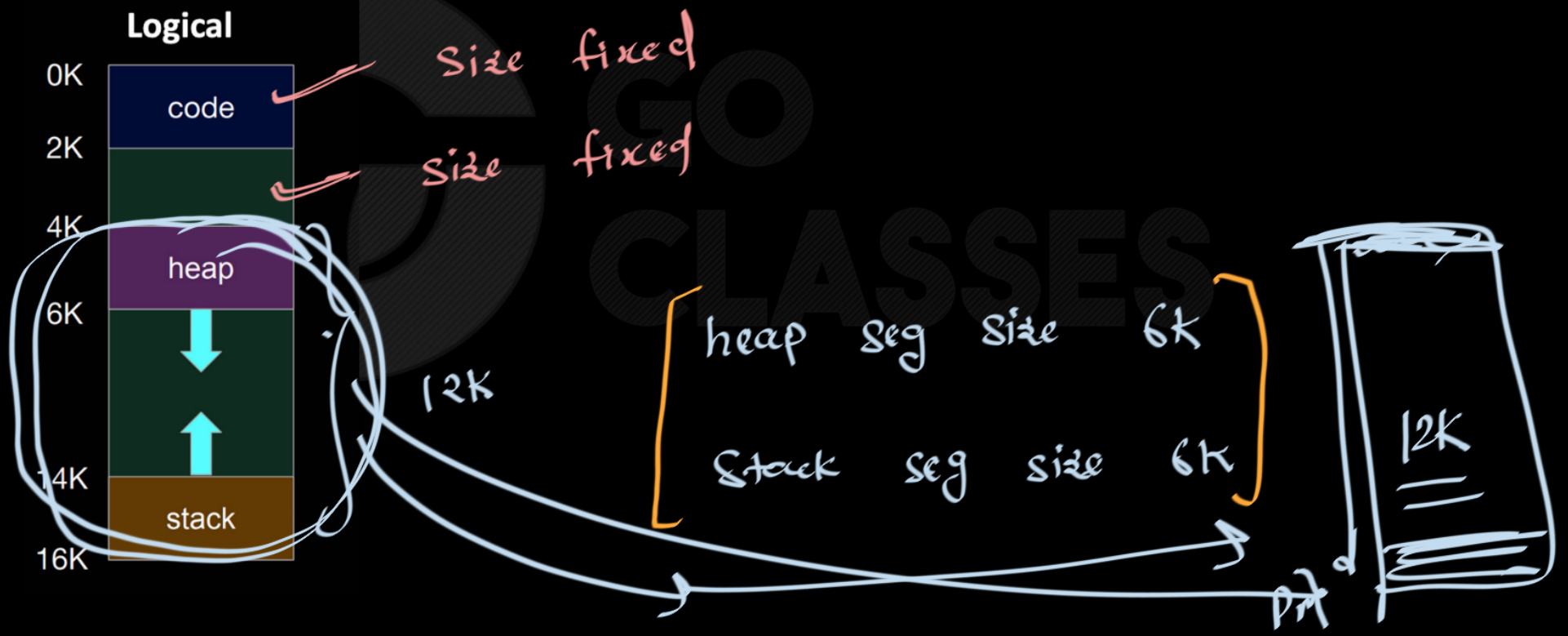
No negotiation

Space wasted


LAS: 4 00000 Bytes 4GB
0
code
Stack
Heap
40 0000 - 1
1KB
main ( )
{
int a;
4B
a = 3;
Pf(a);
}
1MB

GO CLASSES

Decide Size of segment



GO CLASSES

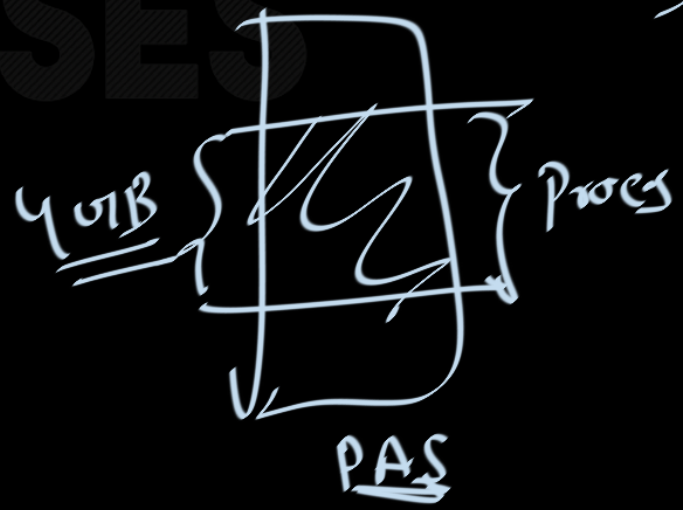
↓ Start sizes (later change using base bound)

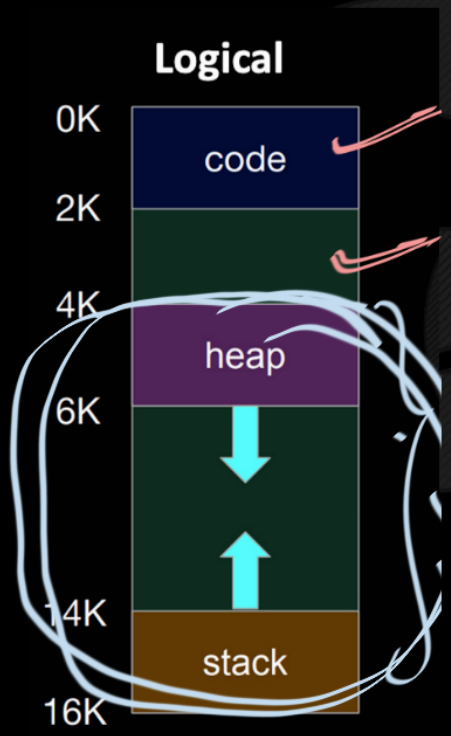
Case1:

{ heap : 6K
  stack : 6K

(Obvious problem: too much main memory needed)

[Suppose heap wants to grow more than 6K.





Case2:

heap: 2K

stack: 2K

it doesn't has obvious problem which was their earlier

Suppose heap coants to grow more than 2K?

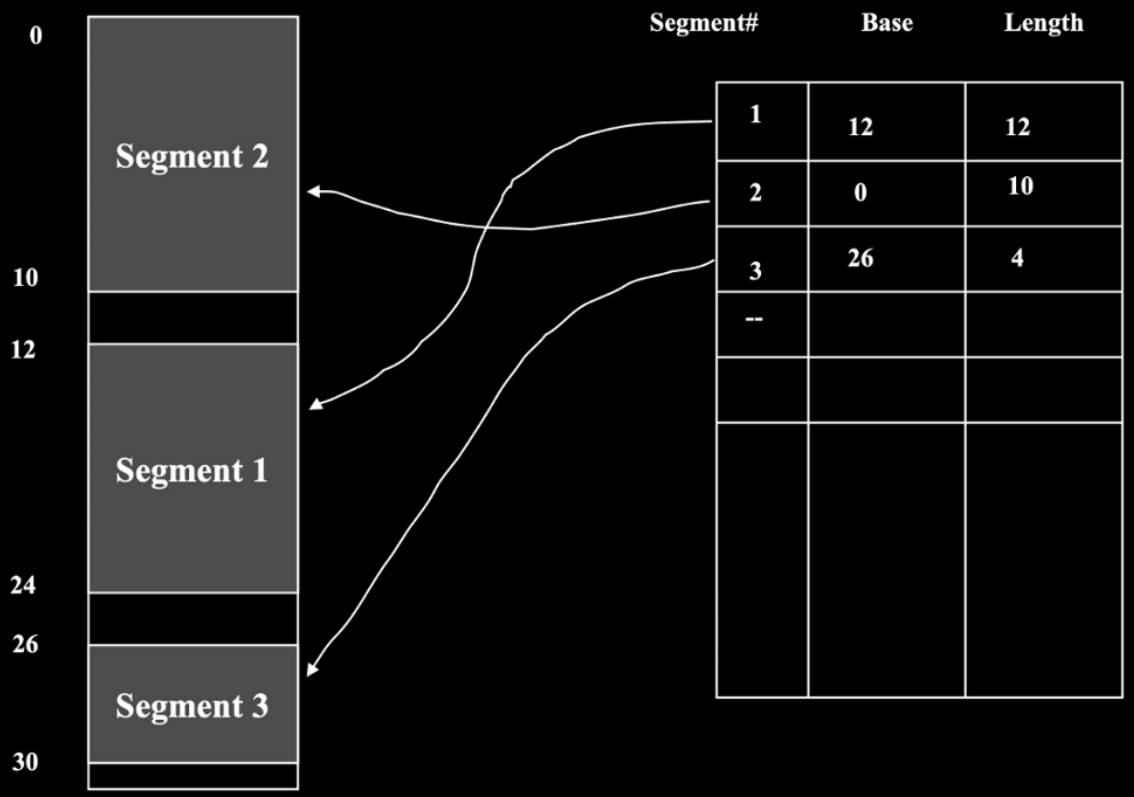
→ if you have Cont. memory then increase bound

GO CLASSES

we might need to incr size of seg. from whatever we start with.

↑ in segmentation

Segment Table



Questions on Base-Bound and Segmentation

multiple Base - Bound

Question

1. With one CPU, how many **base registers** are in the system (total)?
   a. 0
   b. 1
   c. 4
   d. 8
   e. None of the above

2. Assume the base register is set to 1000, and the bounds to 100. How many accesses to memory will the following instruction sequence generate, when fetched and then executed upon that CPU?

```
load 10, r1
```

   a. 0
   b. 1
   c. 2
   d. 3
   e. None of the above

3. With base=1000, and bounds=100, what **physical address** will virtual address 10 be translated to when it is accessed?
   a. 100
   b. 1000
   c. 1010
   d. 110
   e. None of the above

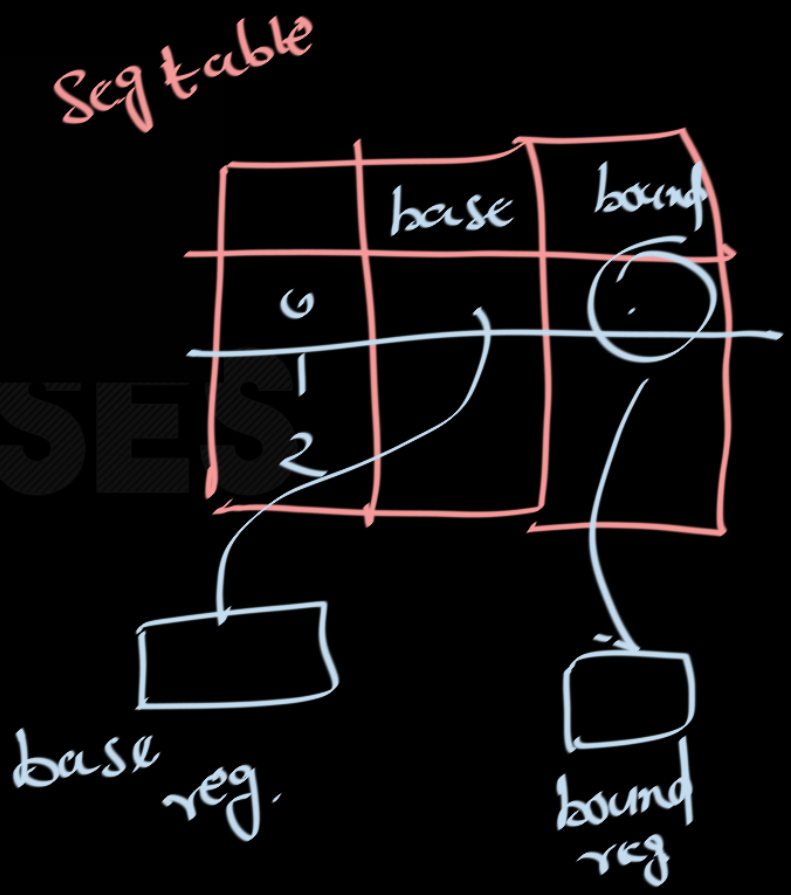
4. With the bounds register set to 100, how many **different legal addresses** can a process access?
   a. 0
   b. 1
   c. 100
   d. 1000
   e. None of the above

https://pages.cs.wisc.edu/~remzi/Classes/537/Spring2009/OldExams/22-fall-mid-solutions.pdf

GO CLASSES

1. With one CPU, how many **base registers** are in the system (total)?
   a. 0
   b. 1
   c. 4
   d. 8
   e. None of the above

in hardware



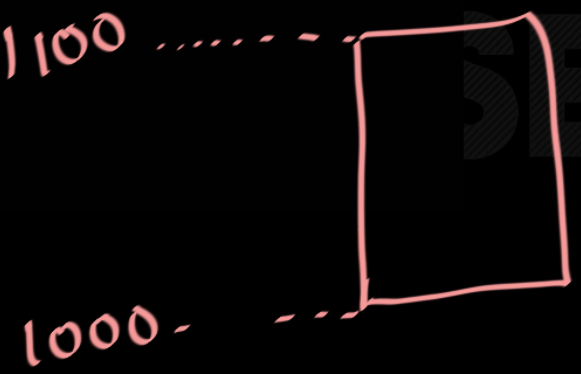
2. Assume the base register is set to 1000, and the bounds to 100. How many accesses to memory will the following instruction sequence generate, when fetched and then executed upon that CPU?

```
load 10, r1
```

a. 0
b. 1
c. 2
d. 3
e. None of the above

1 m/m access to load from 10 to r1

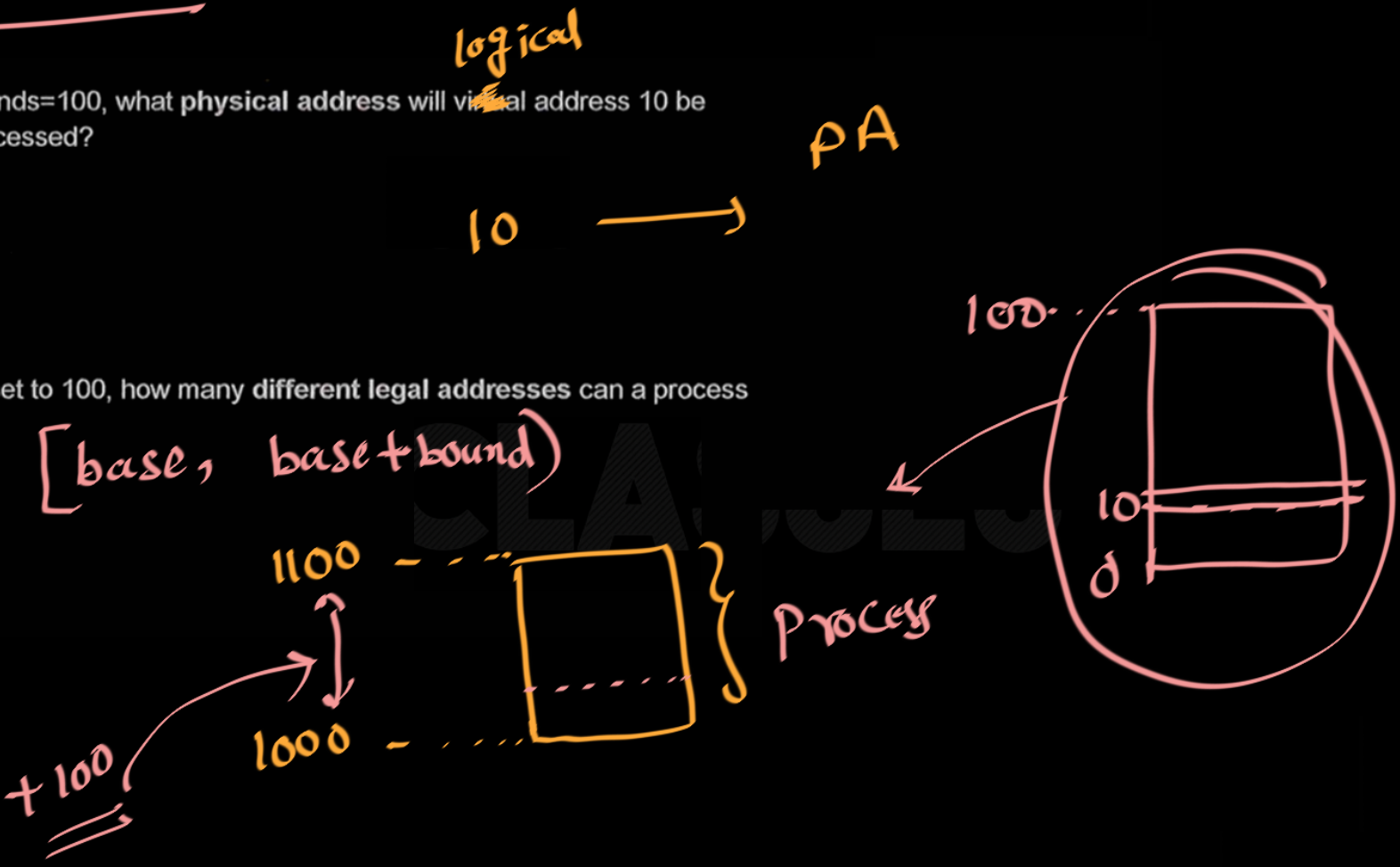
↳ memory



Base and Bound

3. With base=1000, and bounds=100, what **physical address** will virtual address 10 be translated to when it is accessed?
   a. 100
   b. 1000
   c. 1010
   d. 110
   e. None of the above

4. With the bounds register set to 100, how many **different legal addresses** can a process access?
   a. 0
   b. 1
   c. 100
   d. 1000
   e. None of the above



1. With one CPU, how many **base registers** are in the system (total)?
   a. 0
   b. 1
   c. 4
   d. 8
   e. None of the above

there is one base register per CPU

2. Assume the base register is set to 1000, and the bounds to 100. How many accesses to memory will the following instruction sequence generate, when fetched and then executed upon that CPU?

```
load 10, r1
```

   a. 0
   b. 1
   c. 2
   d. 3
   e. None of the above

fetch instruction: one access
load data into r_1: one access
thus, two total

3. With base=1000, and bounds=100, what **physical address** will virtual address 10 be translated to when it is accessed?
   a. 100
   b. 1000
   c. 1010
   d. 110
   e. None of the above

PA = VA + base
= 10 + 1000 = 1010
(within bounds)

4. With the bounds register set to 100, how many **different legal addresses** can a process access?
   a. 0
   b. 1
   c. 100
   d. 1000
   e. None of the above

bounds set to 100 means 0 ... 99 are legal
=> 100 legal addresses

Legal PAs
[base, base+100)

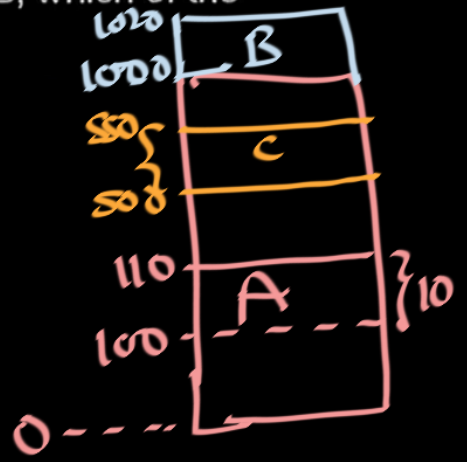
Question

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

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

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

a. 0
b. 20
c. 500
d. 1015
e. 1020



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

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

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

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

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

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


Question

7. When switching between process A and process B, what would happen if OS updated the base register correctly (changing it from 100 to 1000) but **forgot** to update the bounds register (thus leaving it at 10).
   a. Everything would work as expected
   b. Process B might fault unexpectedly
   c. Process B might be able to access memory it shouldn't be able to access
   d. Process A might be mad
   e. None of the above

1015 → trap (seg fault)

↳ false because hardware check

8. Finally, assume you configure a system to have **four CPUs** (not just one). How many **total base registers** are in the system now?
   a. 1
   b. 2
   c. 4
   d. 8
   e. None of the above

every CPU has its own registers

Base 1000

Bound 10

20

7. When switching between process A and process B, what would happen if xOS updated the base register correctly (changing it from 100 to 1000) but **forgot** to update the bounds register (thus leaving it at 10).
   a. Everything would work as expected
   b. Process B might fault unexpectedly
   c. Process B might be able to access memory it shouldn't be able to access
   d. Process A might be mad
   e. None of the above

Bounds should be 20, but is 10

B might generate legal address (like 15), but fault!

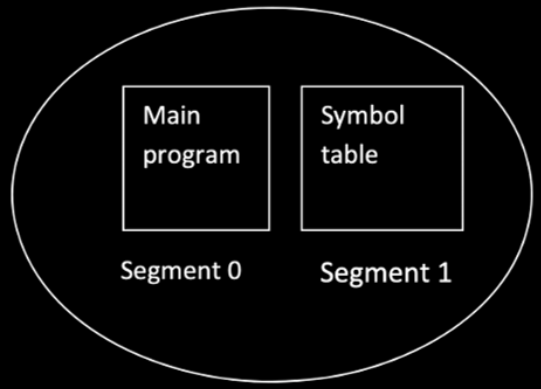
8. Finally, assume you configure a system to have **four CPUs** (not just one). How many **total base registers** are in the system now?
   a. 1
   b. 2
   c. 4
   d. 8
   e. None of the above

one per CPU, as each is running a different process and each is virtualized

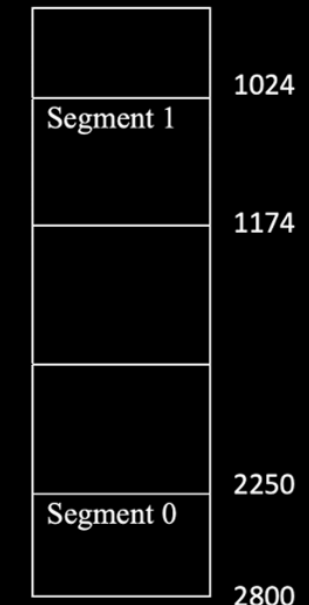
https://pages.cs.wisc.edu/~remzi/Classes/537/Spring2009/OldExams/22-fall-mid-solutions.pdf

Question

Given the following segmentation:



Logical address space



Physical memory

a) **Draw the segment table.**

b) **For each of the following logical addresses, determine the physical address or indicate if a segment fault occurs:**

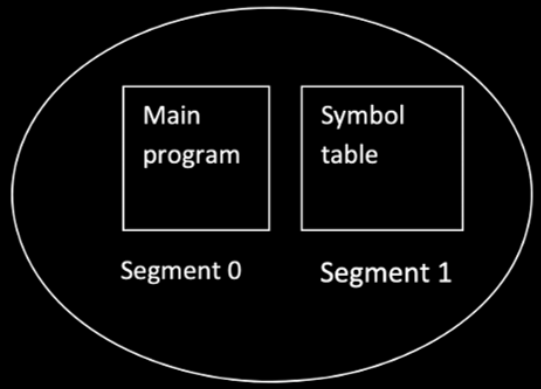
a) (0, 240)

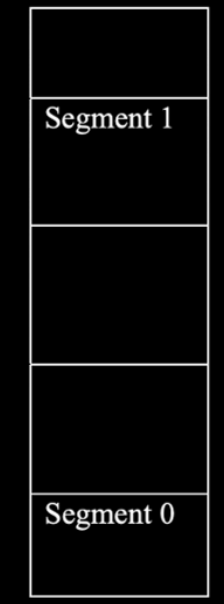
b) (1, 350)

c) (0, 76)

Question

Given the following segmentation:





a) **Draw the segment table.**

b) **For each of the following logical addresses, determine the physical address or indicate if a segment fault occurs:**

a) (0, 240)

b) (1, 350)

c) (0, 76)

a) Draw the segment table. (1.5 marks)

	limit	base
0	551	2250
1	151	1024

b) For each of the following logical addresses, determine the physical address or indicate if a segment fault occurs: (1.5 marks: 0.5 for each part)

a) (0, 240) 240<551? Yes. Physical address = 2250+240=2490

b) (1, 350) 350<151? No. Segment fault

c) (0, 76) 76<551? Yes. Physical address=2250+76=2326

Question

Consider the following Segment Table:

	Base	Limit
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

Size of segment

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

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

1000 →

2400 →

GO CLASSES

	Base	Limit
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

LAS

LA ⟶ PA

Base + offset

1000 ⟶ offset 1000 − 714 = 286

Base : 1327

PA : offset + Base = 1613

1000 address is in seg3

GO CLASSES

	Base	Limit
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

LA :

PA

Base + offset

+ offset

belong to seg3 (we saw base of every seg

PA: 1375

offset 48

Base + offset

PA 1375

1327 - - - -

1327+579- - - - -

seg3 in main memory

	Base	Limit
0	219	600
1	2300	14
2	90	100
3	1327	580
4	1952	96

PA: 1375

LA

Step1: find seg No. : 3

1327 ---- seg 3

offset: 48

1327+579 ----

PAS

714

714+48

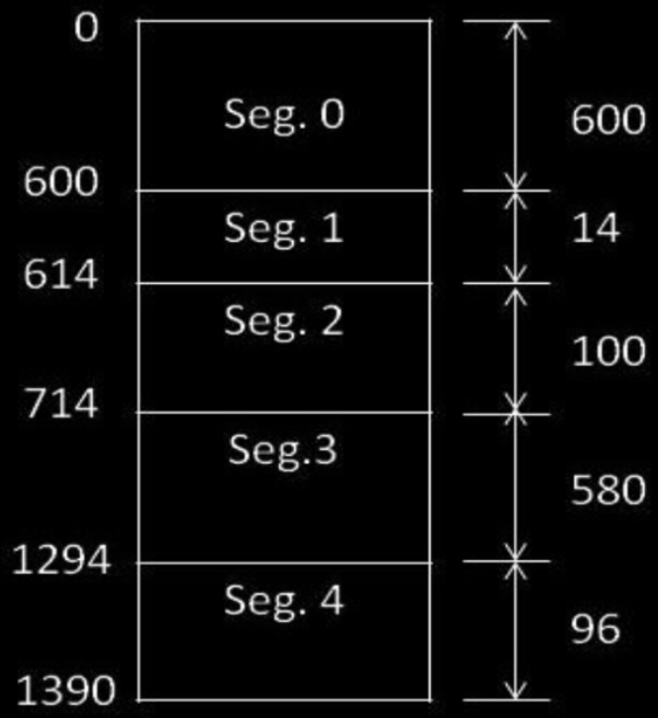
LAS

LA: 700

Step1: find

seg No. : 2

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



i.

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

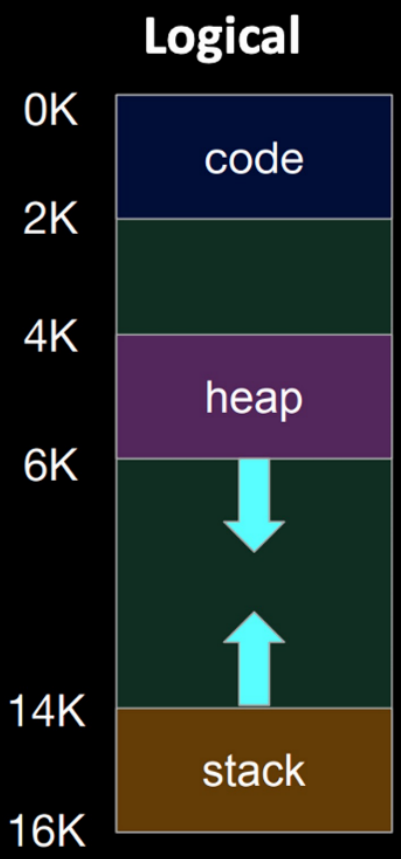
2400 is beyond the logical address space of the process. Hence, it will generate a TRAP: Addressing Error. (Ans.)

ii.

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

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

Question



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

H.W

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

they have not allocated 6K~6K.

Solution

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

address: 7000

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

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

address: 15K ?

- For home thinking / reading

Question

Base & bound

Consider the following translation of logical address to physical address.

```
0: 776   --> 14611  VALID
1: 597   --> 14432  VALID
2: 929   --> SEGMENTATION VIOLATION
```

What can we say about the value of the base register? What abount the bounds register?

"using Base and Bound"

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

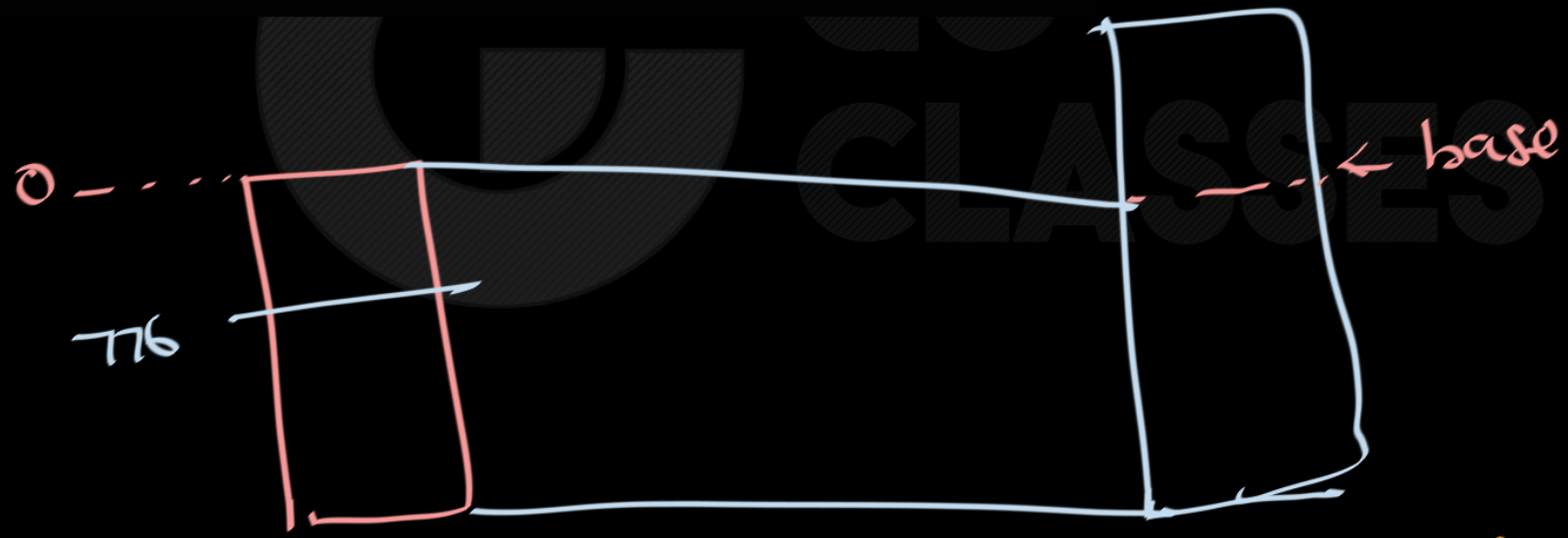
GO CLASSES

LA PA → LA + base

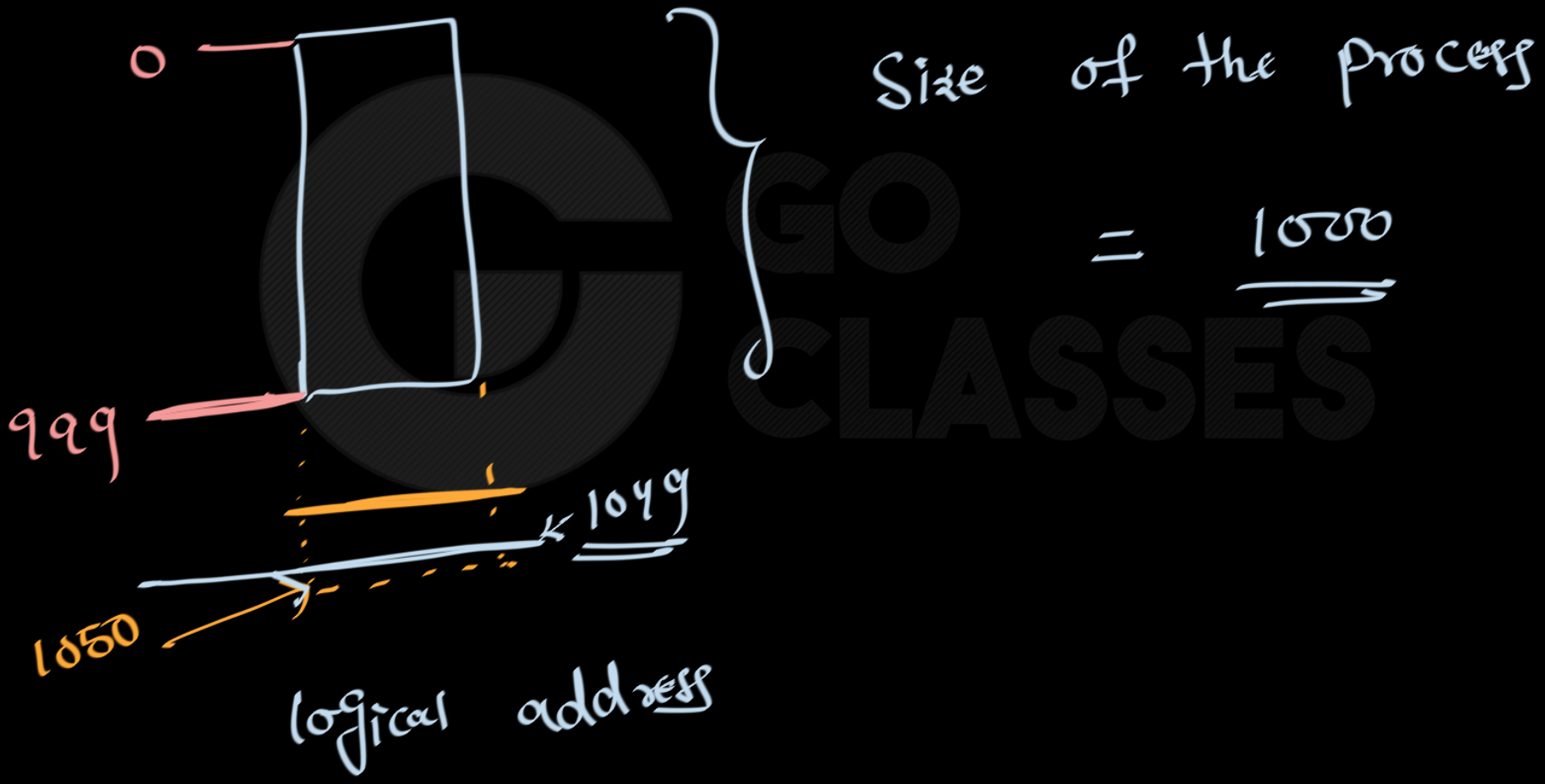
```
0: 776   --> 14611  VALID
1: 597   --> 14432  VALID
2: 929   --> SEGMENTATION VIOLATION
```

14611 − 776 = 13835

14432 − 597 = 13835

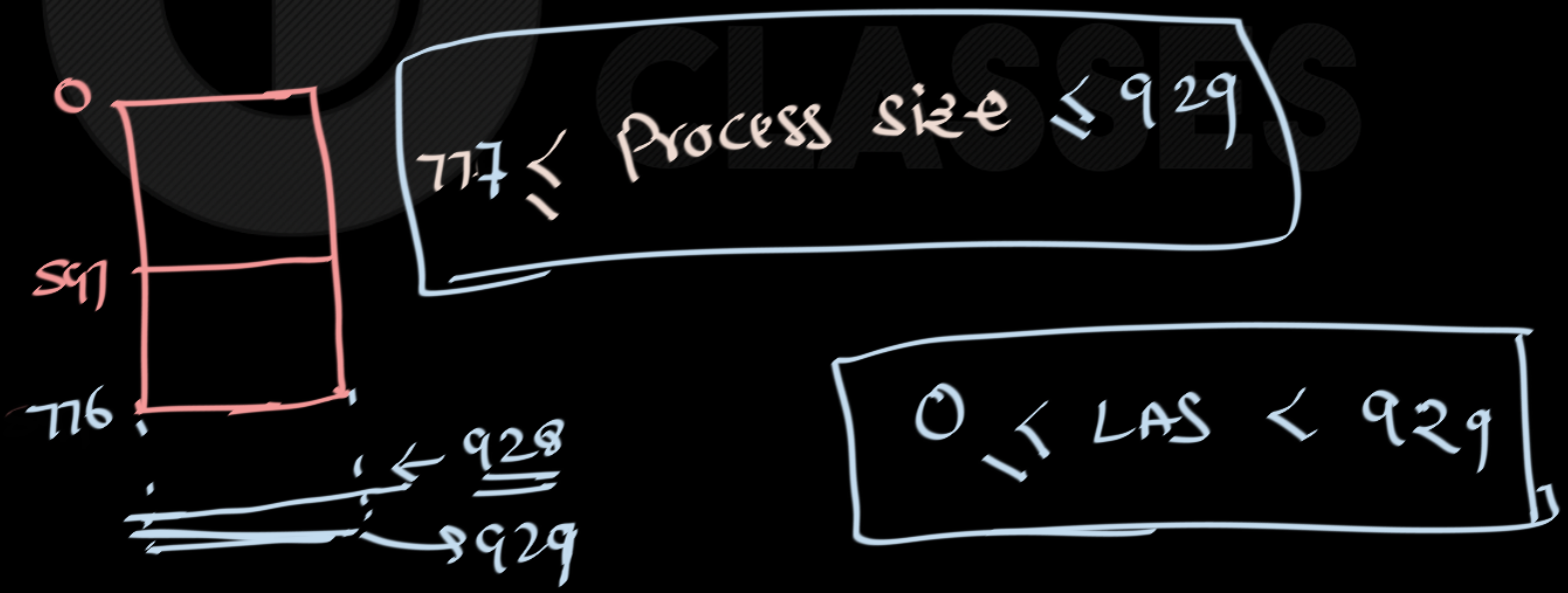


776 < limit of process ≤ 929


0
999
1049
1050
logical address
Size of the process
= 1000

GO CLASSES

```
0: 776   --> 14611  VALID
1: 597   --> 14432  VALID
2: 929   --> SEGMENTATION VIOLATION
```



Solution

Compute base by subtracting : 14611 − 776 = 13835 ✓

Compute base again : 14432 − 579 = 13853 (base changed?) ✗

597 ≠ 13835

Bounds : ~~can't~~ we know 776 is valid, 929 isn't ; thus

recall: bounds is size of valid region

$$776 < \text{bounds} \leq 929$$

Question

Consider the logical address which is having 2 segments
The logical to physical address translation is shown below.

```
108  --> VALID in SEG1:1004
 29  --> VALID in SEG0:541
 80  --> SEGMENTATION VIOLATION (SEG1)
 30  --> SEGMENTATION VIOLATION (SEG0)
 88  --> VALID in SEG1:984
 97  --> VALID in SEG1:993
 53  --> SEGMENTATION VIOLATION (SEG0)
 33  --> SEGMENTATION VIOLATION (SEG0)
100  --> VALID in SEG1:996
 61  --> SEGMENTATION VIOLATION (SEG0)
 12  --> VALID in SEG0:524
  5  --> VALID in SEG0:517
 47  --> SEGMENTATION VIOLATION (SEG0)
```

(a) From this trace, what was the **base register of segment 0** set to?

(b) From this trace, what was the **bounds register of segment 0** set to?

(c) From this trace, what was the **base register of segment 1** set to?

(d) From this trace, what was the **bounds register of segment 1** set to?

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

```
108  --> VALID in SEG1:1004
 29  --> VALID in SEG0:541
 80  --> SEGMENTATION VIOLATION (SEG1)
 30  --> SEGMENTATION VIOLATION (SEG0)
 88  --> VALID in SEG1:984
 97  --> VALID in SEG1:993
 53  --> SEGMENTATION VIOLATION (SEG0)
 33  --> SEGMENTATION VIOLATION (SEG0)
100  --> VALID in SEG1:996
 61  --> SEGMENTATION VIOLATION (SEG0)
 12  --> VALID in SEG0:524
  5  --> VALID in SEG0:517
 47  --> SEGMENTATION VIOLATION (SEG0)
```

(a) From this trace, what was the **base register of segment 0** set to?

(a) From this trace, what was the **base register of segment 0** set to?
To calculate this, find a valid reference to segment 0. For example:

```
5  --> VALID in SEG0:517
```

Here you can see that address 5 translates to physical address 517. Subtracting the offset into segment 0 (5) from 517 gets us 512, which must be the base address (0x400 for those who love hex).

(b) From this trace, what was the **bounds register of segment 0** set to? **For this, you need to find (in the best case) two references, one that is at address N and is valid, and one that is at address N+1 but is not valid; that will tell us the bound precisely. And hence:**

```
29  --> VALID in SEG0:541
30  --> SEGMENTATION VIOLATION (SEG0)
```

This tells us that any address from 0 through 29 is valid, whereas 30 is not. Hence, the size of segment zero is 30, which we thus conclude is the bound. If you want the physical address instead, it is base plus size, or 542.

(c) From this trace, what was the **base register of segment 1** set to? **Similar to the question above, but remember that segment 1 goes backwards:**

```
108  --> VALID in SEG1: 1004
```

Thus, if 108 maps to 1004, we know that 127 (the last valid byte of the address space) would map to 1023 (add 19 to 108 gets us 127; thus add 19 to 1004 to get 1023). Hence, the base (as described in the notes) would be 1024, just beyond the end of where the backward-growing segment lies.

(d) From this trace, what was the **bounds register of segment 1** set to?
The bound here is harder, because it is imprecise.

```
80  --> SEGMENTATION VIOLATION (SEG1)
88  --> VALID in SEG1:984
```

As you can see, 80 goes too far backwards, but 88 is fine. With 88, the size must at least allow 88 through 127 to be valid; thus 40 bytes is the minimum segment size. It clearly could also be that 81 is valid, which adds 7 more bytes into the segment size, or 47. Thus, the size of this segment is between 40 and 47, inclusive.

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!).

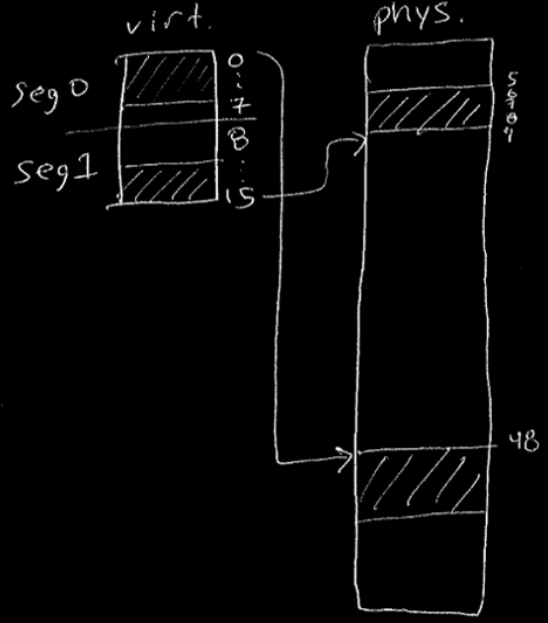
The physical memory is only 64 bytes (also tiny!).

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
```



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:

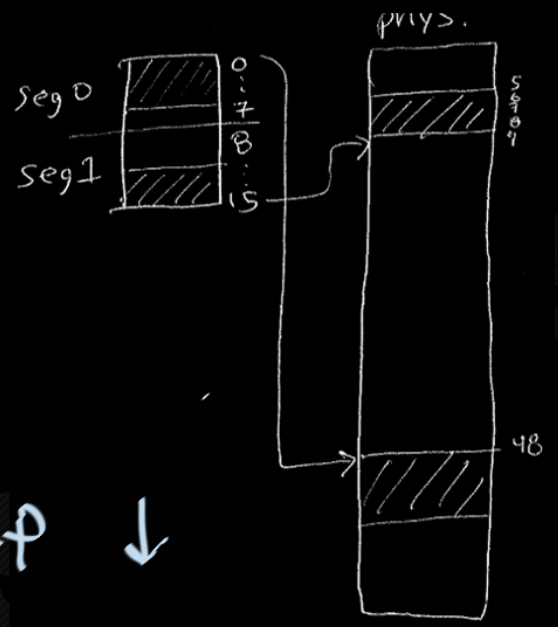
Segment 0 base	(grows positive)	: 48
Segment 0 limit		: 7
Segment 1 base	(grows negative)	: 9
Segment 1 limit		: 4

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

14

6

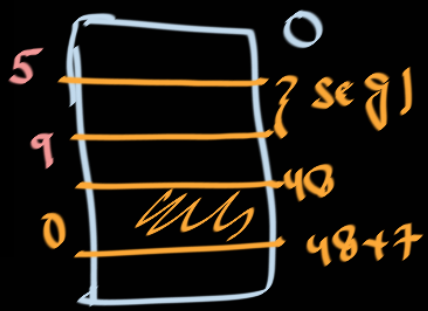
10

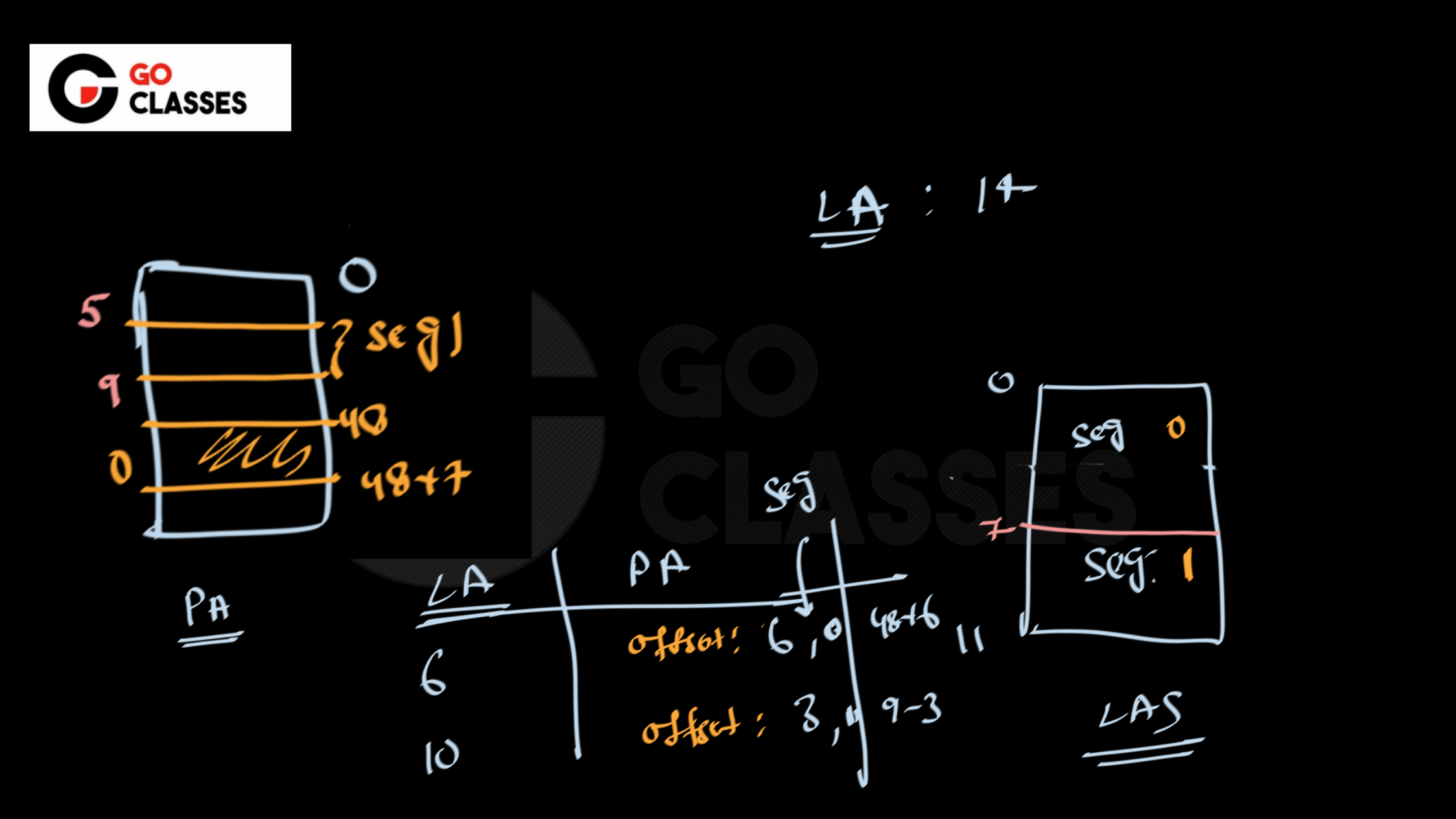


Seg0 : code + heap ↓

Seg1 : Stack ↑

LAS : 16 byte PAS : 64




LA : 14
5
9
0
0
7 seg 1
40
4847
PA
LA
PA
seg
offset: 6, 0
6
offset: 3, 4
10
4846
9-3
11
0
seg 0
7
seg. 1
LAS

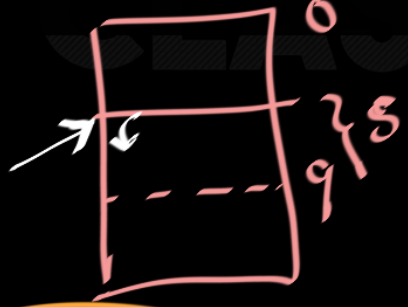
Solution

initial sizes

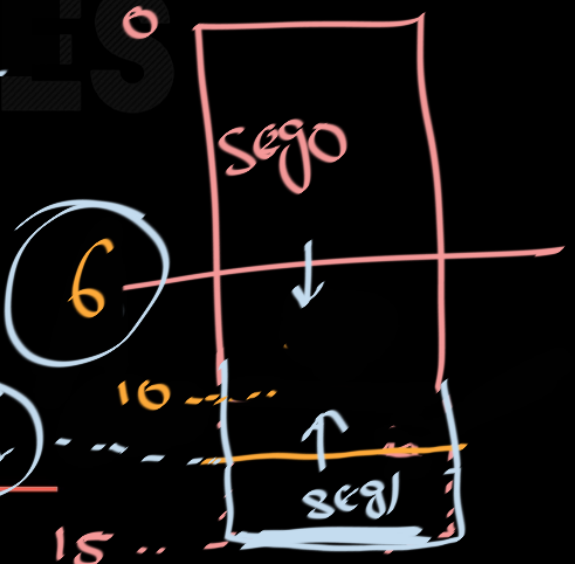
```
Here is some segment register information:
  Segment 0 base  (grows positive) : 48
  Segment 0 limit                  : 7

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

And here is the logical Address Trace:
  14
   6
  10
```

14	-->	7
6	-->	54
10	-->	3

14 → LA offset : 2 5+2 = 7



6 → offset : 6 48+6 = 54



Solution

initial sizes

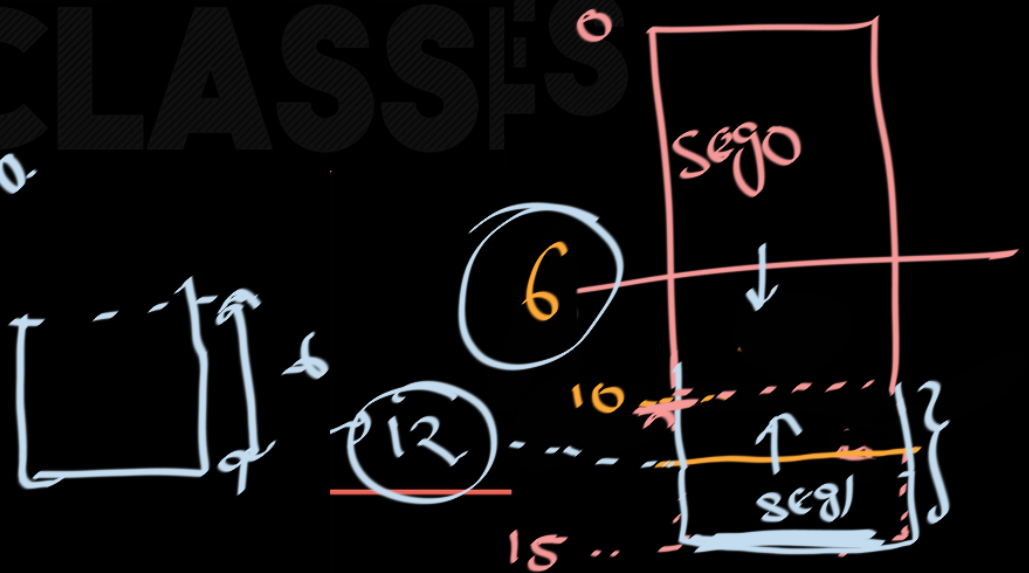
```
Here is some segment register information:
  Segment 0 base  (grows positive) : 48
  Segment 0 limit                  : 7

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

And here is the logical Address Trace:
  14
  6
  10
```

14	-->	7
6	-->	54
10	-->	3

Case 1: Seg1 is growing till 10.

offset for 10: 0

10 → 9 − 6 = 3



GO CLASSES

int *p = malloc(4)

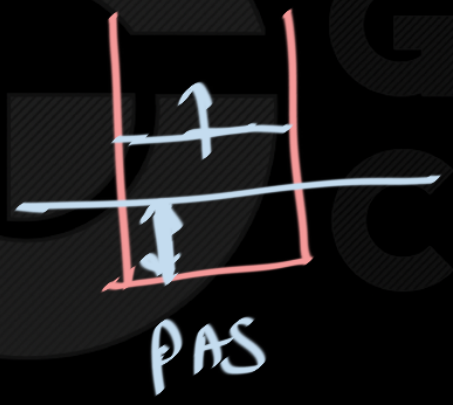
*p = 3;

Pf (*p)

3

Pf (*(p + 3))

↳ may be run time error


PAS


LAS

Question

Assume the following in a simple segmentation system that supports **two** segments: one (positive growing) for code and a heap, and one (negative growing) for a stack:

- Logical address space size 128 bytes (small!)
- Physical memory size 512 (small!)

```
Segment register information:
  Segment 0 base  (grows positive) : 0
  Segment 0 limit                  : 20 (decimal)
  Segment 1 base  (grows negative) : 0x200 (decimal 512)
  Segment 1 limit                  : 20 (decimal)
```

Which of the following are **valid** logical memory accesses?

To answer: Fill in **A** for valid virtual accesses, **B** for non-valid accesses.

71. 0x1d (decimal: 29)
72. 0x7b (decimal: 123)
73. 0x10 (decimal: 16)
74. 0x5a (decimal: 90)
75. 0x0a (decimal: 10)

https://pages.cs.wisc.edu/~remzi/Classes/537/Spring2009/OldExams/18-spring-mid-answers.pdf

Solution

With the above address space, logical addresses 0 ... 19 are valid (the first 20 bytes of the AS, as described by the segment 0 base/limit pair), and logical addresses 127 ... 108 are valid too (the last 20 bytes of the AS. It doesn't matter where they map to in physical space for this question. Thus:

71. 0x1d (decimal: 29) **B. Not Possible**
72. 0x7b (decimal: 123) **A. Possible**
73. 0x10 (decimal: 16) **A. Possible**
74. 0x5a (decimal: 90) **B. Not Possible**
75. 0x0a (decimal: 10) **A. Possible**

Question

Logical Address	Physical Address	
0	1000	Base? _____
100	1100	
1999	2999	Bounds? _____
2000	[fault]	

Logical Address	Physical Address	
0	1000	Base? _____
100	1100	
1999	2999	Bounds? _____
2000	3000	

Logical Address	Physical Address	
100	3400	Base? _____
2000	5300	
2001	_____ ?	Bounds? _____
3000	6300	

Logical Address	Physical Address	
0	_____ ?	Base? 6050
100	_____ ?	
2000	_____ ?	Bounds? _____
2001	[fault]	

Logical Address	Physical Address	
_____ ?	0900	Base? 500
_____ ?	1100	
_____ ?	3000	Bounds? 3000
_____ ?	[fault]	

Logical Address	Physical Address	
9000	10001	Base? _____
100	1101	
2000	3001	Bounds? _____
2001	3002	

Logical Address	-> Physical Address
0	1000
100	1100
1999	2999
2000	[fault]

Base? 1000

Bounds? 2000

Logical Address	-> Physical Address
0	1000
100	1100
1999	2999
2000	3000

Base? 1000

Bounds? > 2000

Logical Address	-> Physical Address
100	3400
2000	5300
2001	5301 ?
3000	6300

Base? 3300

Bounds? > 3000

Logical Address	-> Physical Address
0	6050 ?
100	6150 ?
2000	8050 ?
2001	[fault]

Base? 6050

Bounds? 2001

OR [fault] " " bounds 0

Logical Address	-> Physical Address
400 ?	0900
600 ?	1100
2500 ?	3000
>= 3000 ?	[fault]

Base? 500

Bounds? 3000

Logical Address	-> Physical Address
9000	10001
100	1101
2000	3001
2001	3002

Base? 1001

Bounds? > 9000

Question

With base-and-bounds based virtual memory, two registers (base and bounds) are used to implement a primitive form of virtualization. The subtle change we explore here is to the bounds register. Specifically, in this subtly-different base-and-bounds, the bounds register is checked only on writes to memory, and not on reads from memory. What is the impact of this change?

8. With base-and-bounds based virtual memory, two registers (base and bounds) are used to implement a primitive form of virtualization. The subtle change we explore here is to the bounds register. Specifically, in this subtly-different base-and-bounds, the bounds register is checked only on writes to memory, and not on reads from memory. What is the impact of this change?

Good: can't overwrite memory that isn't yours

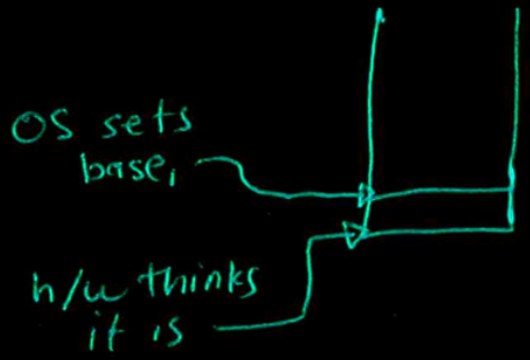
Bad: might leak sensitive info from process → process or OS → process

Question

Assume we have a system that uses segmentation to provide a **logical** memory to processes. Assume the segmentation chops the address space into two parts (segment 0 and 1); segment 0 grows in the increasing direction, while segment 1 grows backwards. Unfortunately, there is confusion over the interpretation of the base register of segment 1; while the hardware thinks it should point to the physical address one beyond the bottom of the backwards-growing segment, the OS has been subtly changed to assume it points to the last byte of the backwards-growing segment. Describe what would happen when running processes on this subtly-changed **logical** memory system.

https://pages.cs.wisc.edu/~remzi/Classes/537/Spring2009/OldExams/12-spring-mid-answers.pdf

Assume we have a system that uses segmentation to provide a **logical** memory to processes. Assume the segmentation chops the address space into two parts (segment 0 and 1); segment 0 grows in the increasing direction, while segment 1 grows backwards. Unfortunately, there is confusion over the interpretation of the base register of segment 1; while the hardware thinks it should point to the physical address one beyond the bottom of the backwards-growing segment, the OS has been subtly changed to assume it points to the last byte of the backwards-growing segment. Describe what would happen when running processes on this subtly-changed **logical** memory system.



result: chaos!
OS thinks it is OK to access last byte on stack → h/w doesn't
⇒ likely fault

Question

Assume logical memory hardware that uses segmentation, and divides the address space in two by using the top bit of the logical address. Each segment is thus relocated independently.

What we'll be drawing in this question is what physical memory looks given some different parameters. We'll also label where a particular memory reference ends up.

For all questions, assume a logical address space of size 16 bytes (yes tiny!) and a physical memory of size 64 bytes. Thus, if we had a logical address space placed in physical memory, it might look like this (with spaces between every 8 physical bytes):

0000FFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFFFFFF FFFF1111

In this example, the segment θ base register is θ, segment 1 base is 64 (it grows backwards), and both length registers are 4.0 's are used to record where segment 0 is in memory; 1 's are for segment 1 ; F means free.

(a) What would physical memory look like if we had the following values instead? (draw a picture below)

$$\begin{array}{llll} \text{seg0 (base)} & :12 & \text{seg0 (limit)} & : 6 \\ \text{seg1 (base)} & : 10 & \text{seg1 (limit)} & \end{array}$$

(b) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of logical address 4 (or DRAW AN X on the physical-memory address if the access is illegal)

(c) What would physical memory look like if we had the following values instead? (draw a picture below)

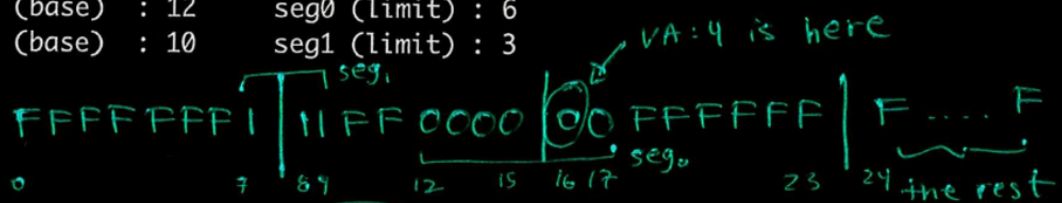
$$\begin{array}{llll} \text{seg0 (base)} & : 40 & \text{seg0 (limit)} & : 4 \\ \text{seg1 (base)} & : 50 & \text{seg1 (limit)} & : 4 \end{array}$$

(d) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of logical address 14 (or DRAW AN X on the physical-memory address if the access is illegal)

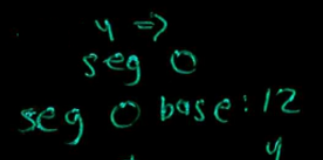
(e) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of logical address 4 (or DRAW AN X on the physical-memory address if the access is illegal)

(a) What would physical memory look like if we had the following values instead? (draw a picture below)

seg0 (base) : 12 seg0 (limit) : 6
seg1 (base) : 10 seg1 (limit) : 3

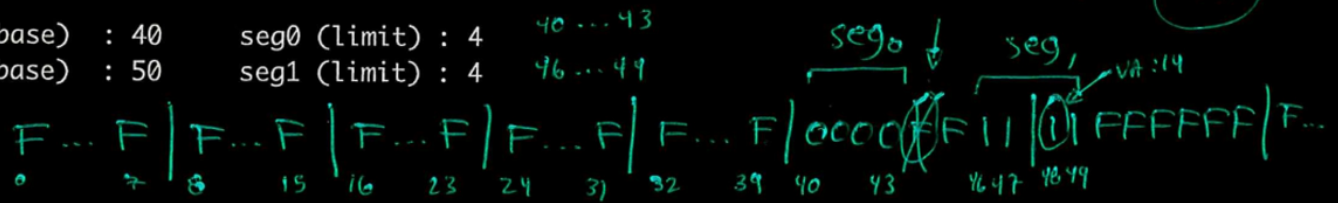


(b) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of address 4
(or DRAW AN X on the physical-memory address if the access is illegal)

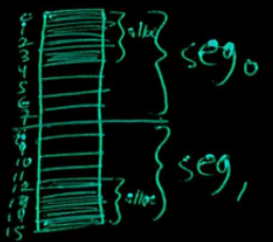


(c) What would physical memory look like if we had the following values instead? (draw a picture below)

seg0 (base) : 40 seg0 (limit) : 4
seg1 (base) : 50 seg1 (limit) : 4



(d) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of virtual address 14
(or DRAW AN X on the physical-memory address if the access is illegal)



(e) In your picture above, CIRCLE which byte of memory is accessed when the process generates a byte load of address 4
(or DRAW AN X on the physical-memory address if the access is illegal)