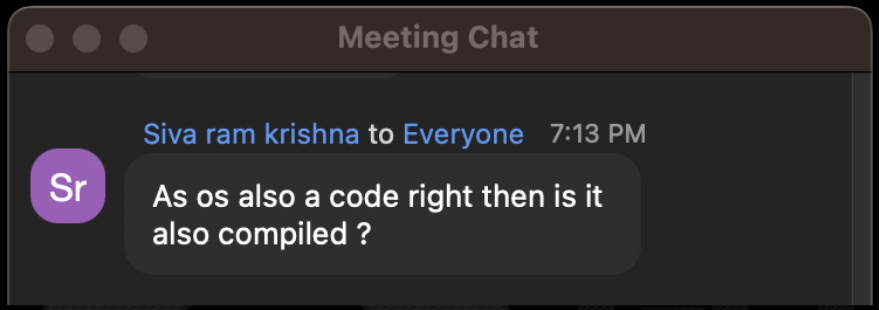
GO CLASSES

- What happens when you switch on the computer
- How does even OS code get loaded in the main memory?
- How to create a new process?
  - How OS load a process in the MM.

GO CLASSES

Q :



Answer : Yes, we already have compiled file (executable file) in the hard disk.

GO CLASSES

Q.

Meeting Chat

Jahnavi P to Everyone 7:13 PM

JP

Is it possible to tweak OS in our personal laptop?

...

Ans

Yes, possible.

High level steps

1) download some open source os code (Linux)

2) do all changes

3) compile it

4) store in the secondary memory

will create executable file

at very special place in secondary memory

5) ask bootloader to point to that address

GO CLASSES

Meeting Chat

Shubham to Everyone 7:14 PM

S

what comes first? os or compiler

Meeting Chat

Jeevan Somavarapu to Everyone 7:23 PM

JS

Will OS code compile everytime when we switch on?

we already have executable file (compile it once)

Meeting Chat

Nandini Tyagi to Everyone 7:15 PM

Meeting Chat

Satyavrat Yadav to Everyone 7:45 PM

SY

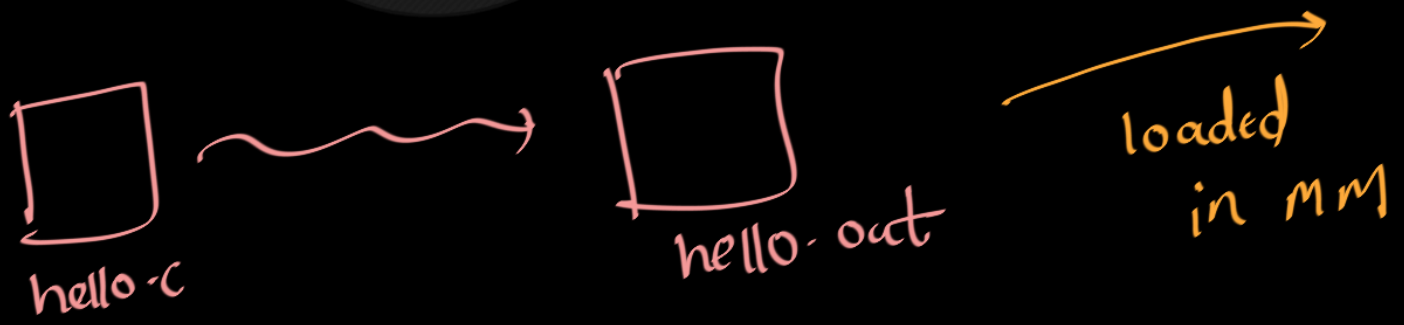
sir, in gui based os we always move cursor for any activities, so doesn't it mean the os is active all the time?

- there is one single CPU. Still it feels that the above two programs are running simultaneously
  - ↳ CPU virtualisation → (illusion)

Creating a New Process

How a program is run on machine?

- User writes a program
- Compiler converts the program into executable code (byte code)
- User requests OS to run the byte code
- OS "loads" the program into memory (loader)
- OS sets registers properly and starts running the code PC value



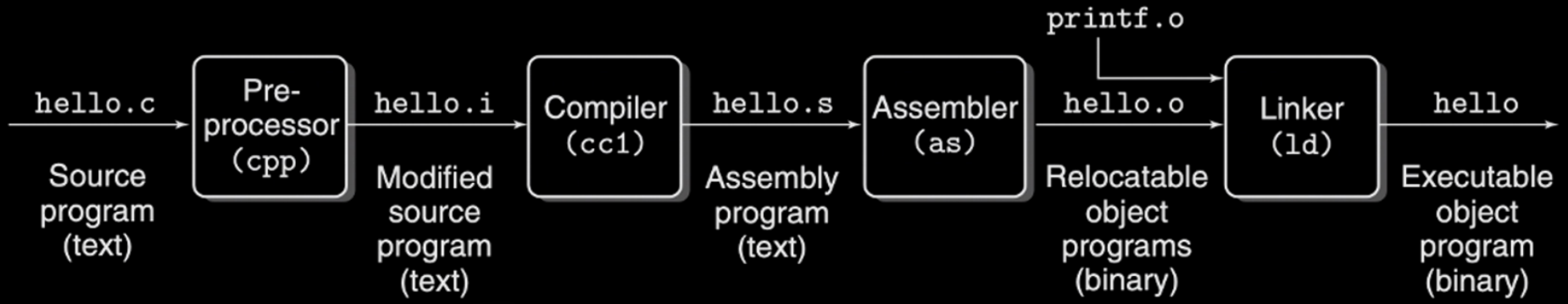


Figure The compilation system.

Please watch it

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

938 views • 2 months ago

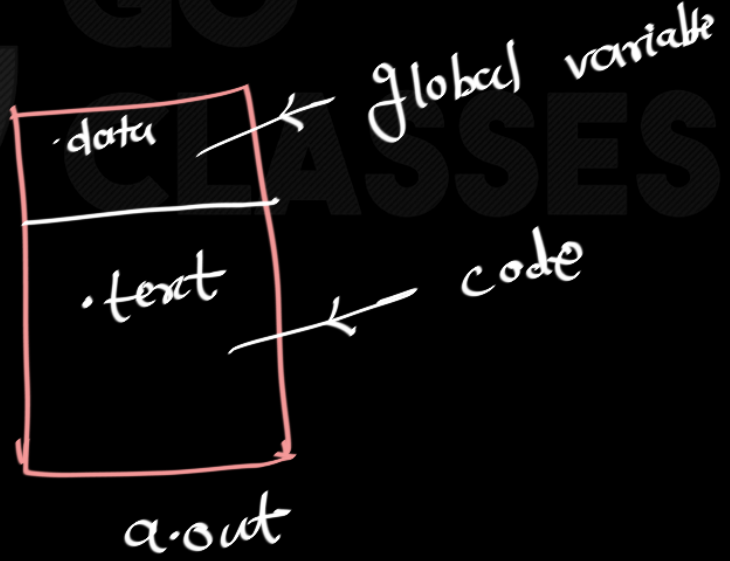
GO Classes for GATE CS

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

```c
int main() {
    int a = 1, b = 2, c;
    c = a + b;
    return 0;
}
```

we want to run this program.

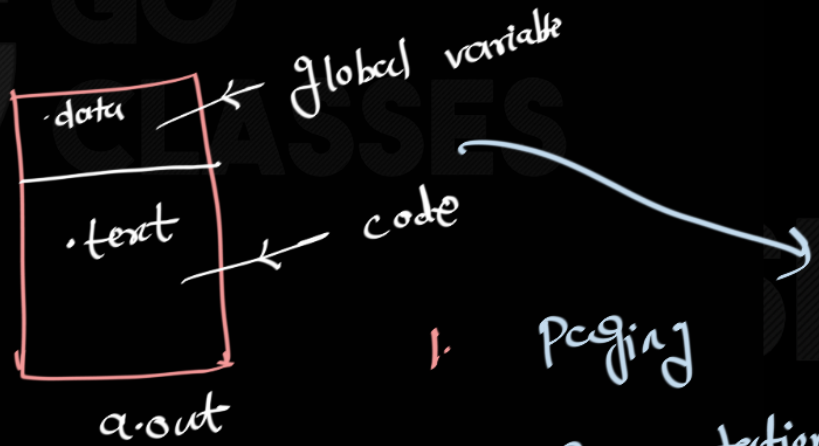
Compile it



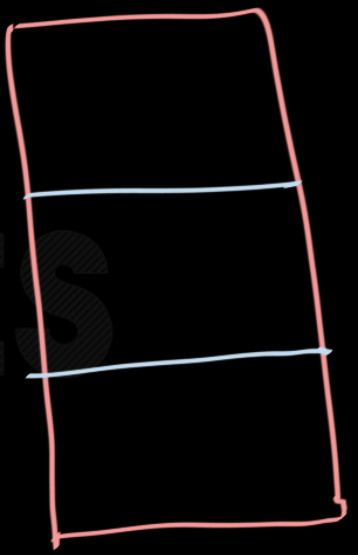
```c
int main() {
    int a = 1, b = 2, c;
    c = a + b;
    return 0;
}
```

we want to run this program.

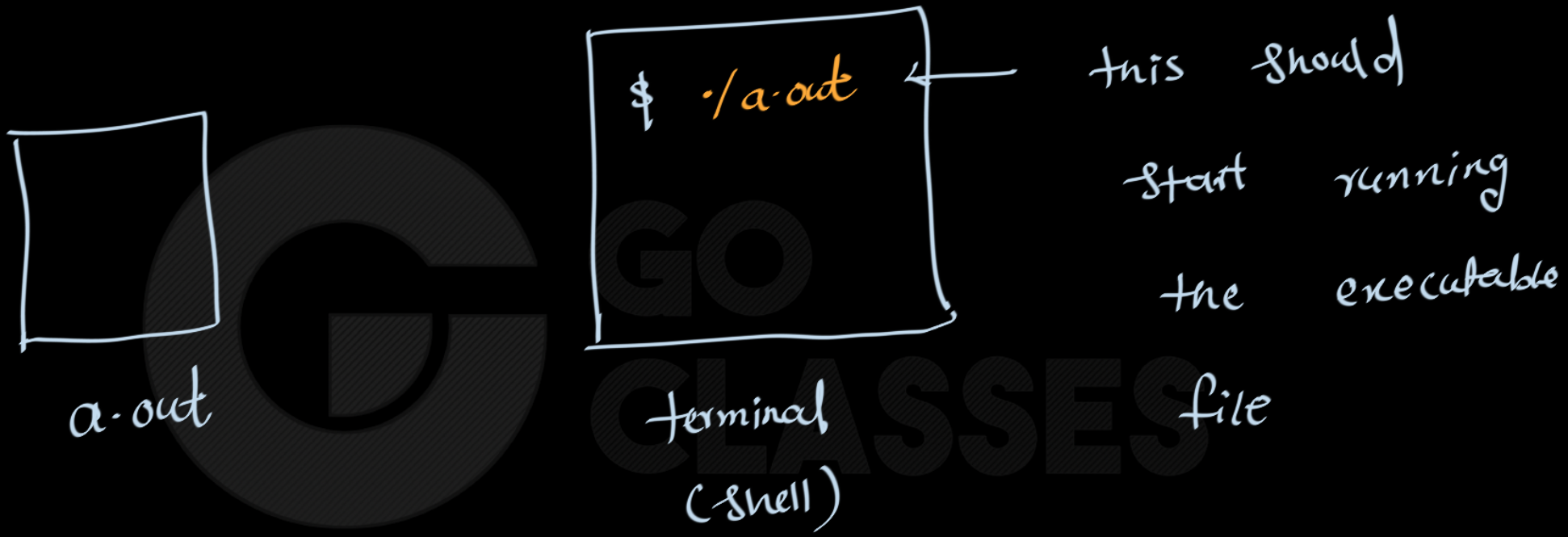
Compile it



M M.

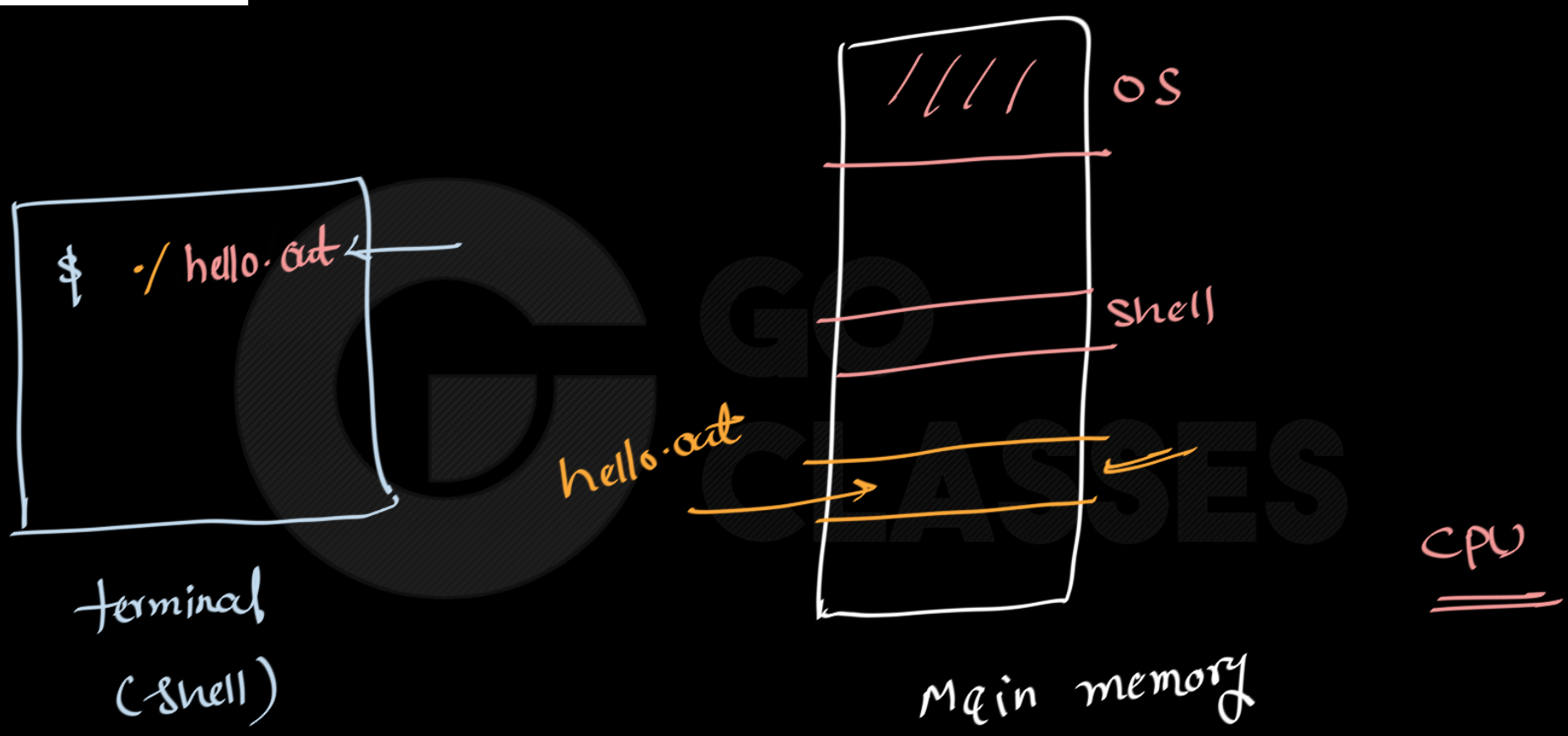


1. Paging
2. Segmentation
3. Base + Bound register



a.out : file name

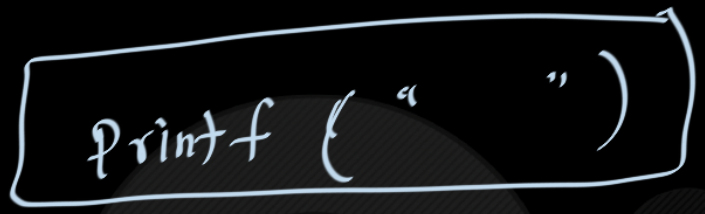
./a.out : Command



(it is not neccessary that we need to run OS code all time)

OS is always in MM (loaded by boot loader at start of computer)

but it does not mean that OS is running all the time.



bcoz we need to see if user program can access monitor

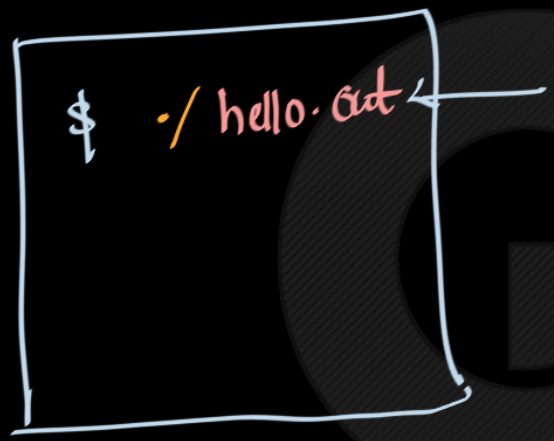
for example, this line can not be executed without interference of operating System.

↳ using the System call we will do it.

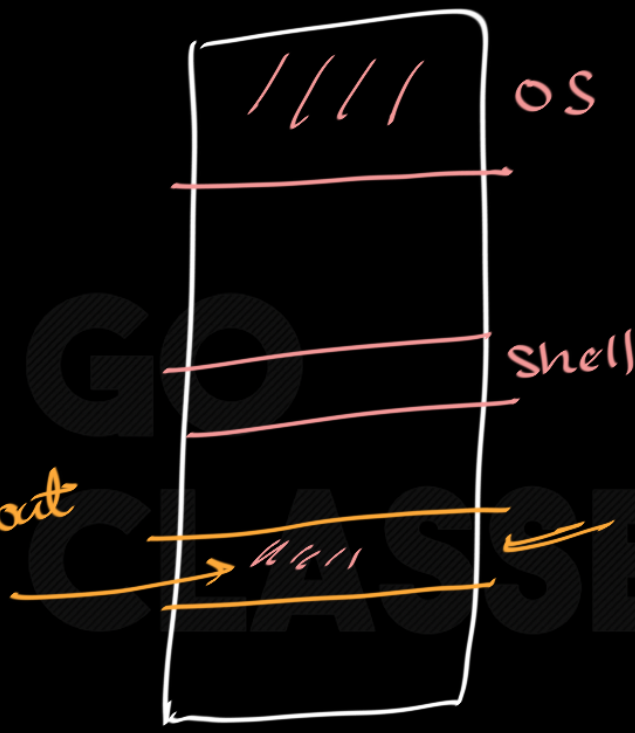
GO CLASSES

System calls will be taught in detail later

- → user mode - kernal mode
- → depth details of threads
- → system calls



terminal
(shell)



Main memory

1. Create space for hello.out

2. load hello.out in M

GO CLASSES

Objective :

understand how does the 2 steps get perform.

fork-exec

1. Create space for hello.out
2. load hello.out in that space

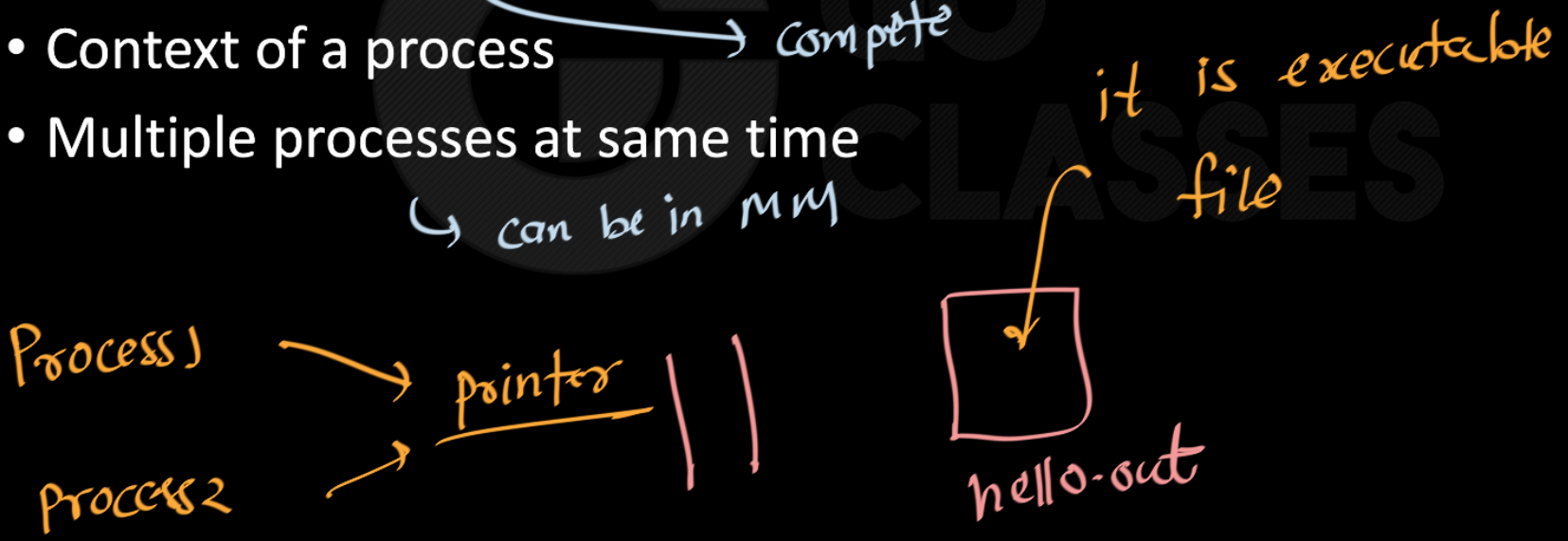
→ understand fork

→ understand exec

→ we will come back to process creation.

A Process

- Informally, program in execution
- A process contests for resources → compete
- Context of a process
- Multiple processes at same time
  ↳ can be in RM



GO CLASSES

- Context of a process

Operating System

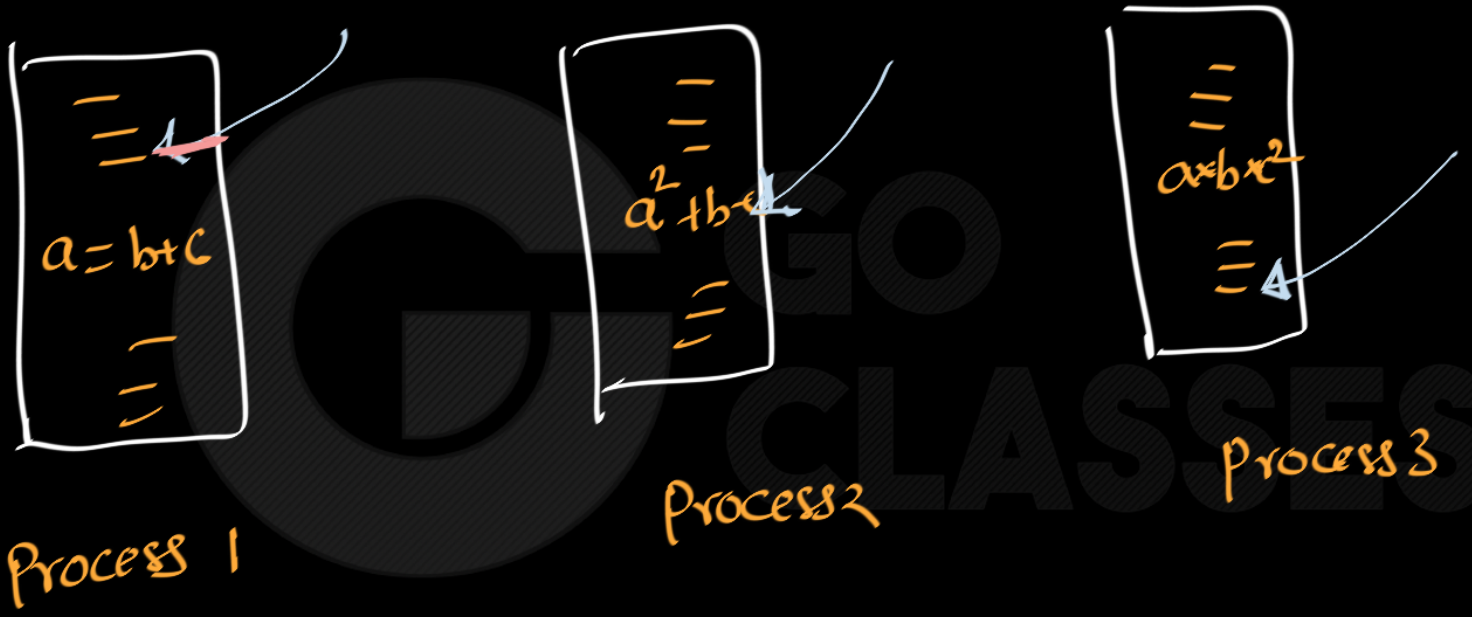
↳ it keep a close eye on every process.

School

↳ there are many kids

↳ every kid might be doing their own activity

↳ if we want to have a close eye on every kid then we need book keeping



" i am at this line " value of PC

$R_1 = R_2 + R_3$

Content

OS has to keep track of everything about the process

- value of PC
- general purpose registers (R_1, R_2, R_3)
- list of open files

```
fd = open( "a.txt" )
```

⇒ there is a special data structure that keep track of the Conent.

from C programing

struct node {

}

GO CLASSES

this Content of the process is getting stored in some location.

it is a part of OS code

```
struct pcb P1;

P1. listofopenfiles . insert ("files")
```

GO CLASSES

in C program:

fd = open("file5")

← user code

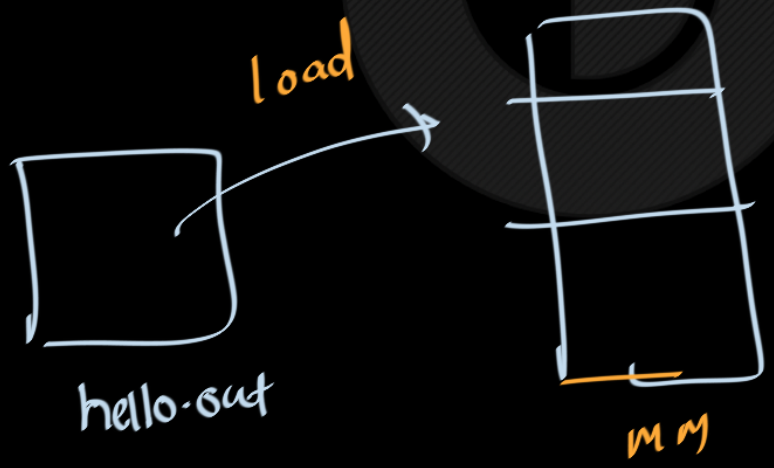
↳ one of the things that it will do is

this is a part of OS code → [P1. listofopenfiles.insert("file5")

"Ready queue"

A Process

- Informally, program in execution



→ after loading, it is not necessary that you will get chance to immediately run it

GO CLASSES

After putting in the main memory, should we call it a process or program?

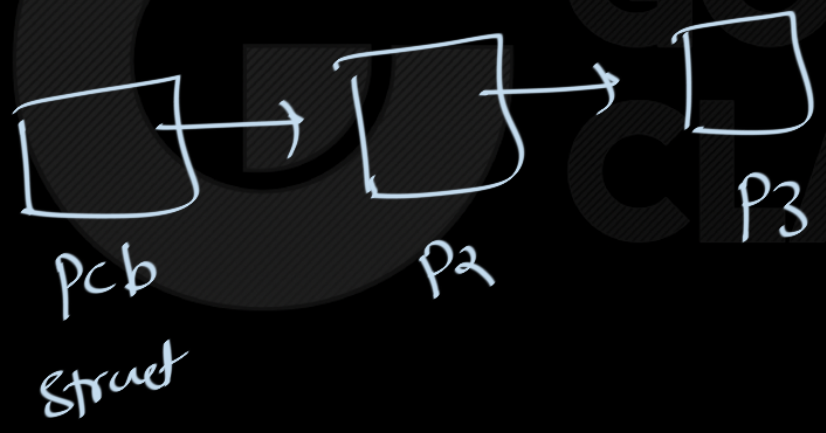
- Yes
- No

→ it is in a gray area.

(some author say that at least it run one time then we will call it process)

GO CLASSES

Round Robin ← we will talk about it later



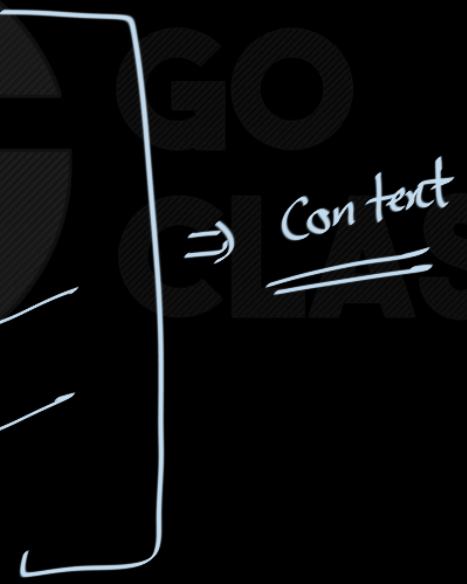
Process Control Block (PCB)

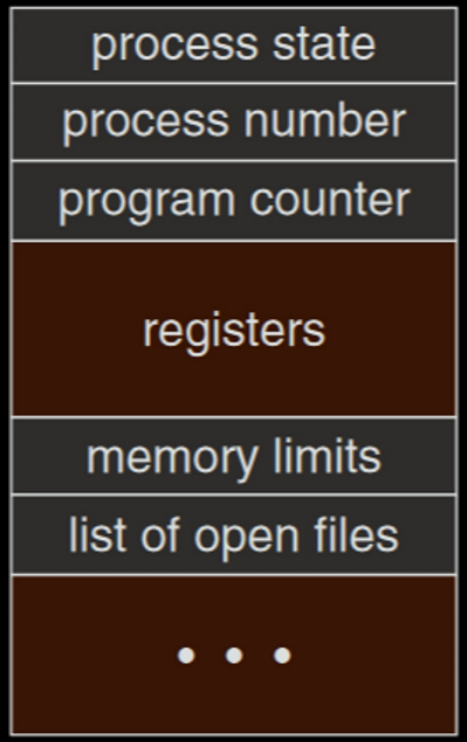
- it is a data structure that keep track of content.

Process Control Block

Contains information about

- Program state
- Program counter
- CPU registers
- Scheduling information
- Memory Management
- And so on.....





```
struct pcb{


pid_t pid; /* process identifier */
long state; /* state of the process */
unsigned int time_slice /* scheduling information */
struct task_struct *parent; /* this process's parent */
struct list_head children; /* this process's children */
struct files_struct *files; /* list of open files */
struct mm_struct *mm; /* address space of this process*/

...

};
```

(ignore this)

GO CLASSES

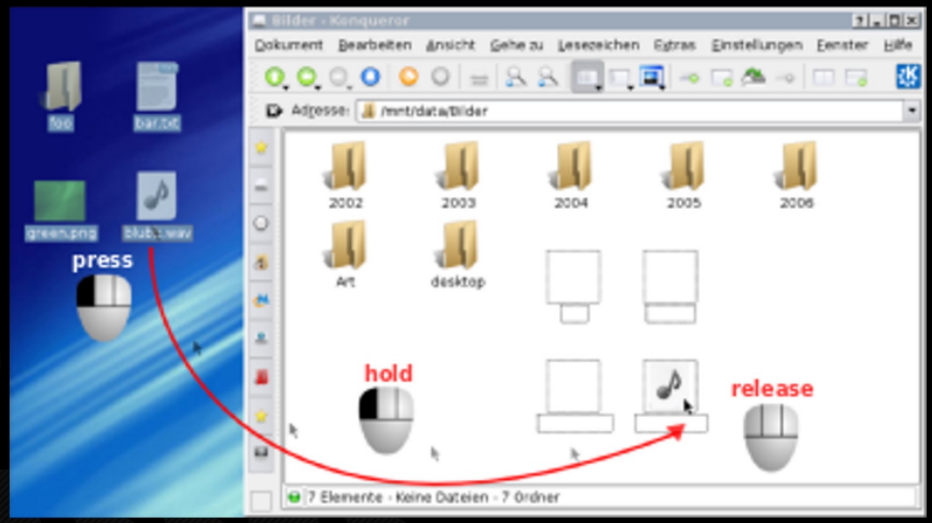
Operating System : it is just a Complex exercise on data structures.

PCB is a data structure (struct) created for each process, and OS maintains a linked list of these struct elements as PCBs

Creating a new process: fork() system call

Creates a exact replica of the calling process

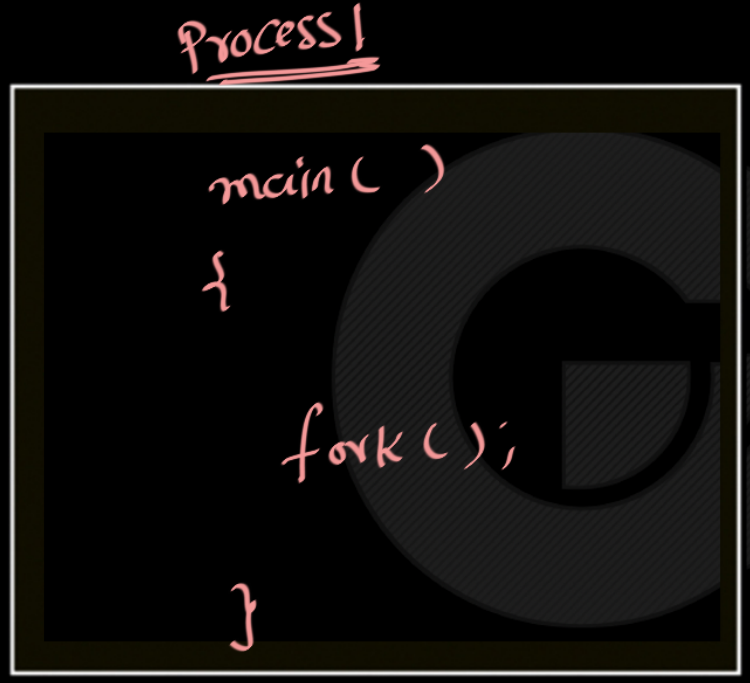
```
C:\Temp> dir
 Volume in drive C is C
 Volume Serial Number is 74F5-B93C

 Directory of C:\Temp

2009-08-25  11:59    <DIR>          .
2009-08-25  11:59    <DIR>          ..
2007-03-01  11:37         2,321,600 AdobeUpdater12345.exe
2009-04-03  10:01            27,988 dd_depcheckdotnetfx30.txt
2009-04-03  10:01               764 dd_dotnetfx3error.txt
2009-04-03  10:01            32,572 dd_dotnetfx3install.txt
2009-06-09  13:46            35,145 GenProfile.log
2009-08-05  12:11               155 KB969856.log
2009-04-20  08:37               402 MSI29e0b.LOG
2009-04-09  16:34            38,895 offcln11.log
2009-04-03  16:02    <DIR>          OfficePatches
2009-07-14  14:30    <DIR>          OHotfix
2009-08-25  10:52            16,384 Perflib_Perfdata_c30.dat
2009-04-03  10:01             1,744 uxeventlog.txt
2009-08-25  11:42        50,245,632 WFV2F.tmp
2009-04-20  10:07             1,397 {AC76BA86-7AD7-1033-7B44-A81200000003}.ini
2009-04-20  10:13               617 {AC76BA86-7AD7-1033-7B44-A81300000003}.ini
              13 File(s)     52,723,295 bytes
               4 Dir(s)  83,570,208,768 bytes free
```

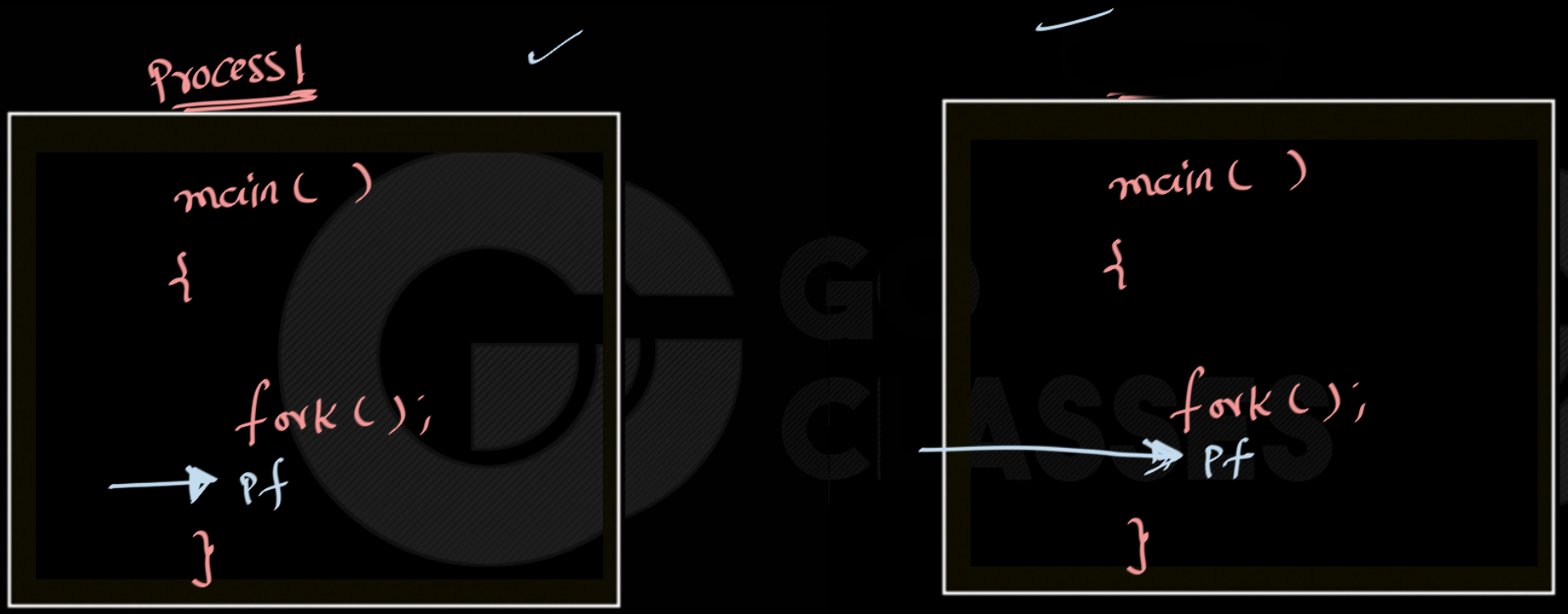


Command line interface

user interface

https://en.wikipedia.org/wiki/Shell_(computing)


Process1
main( )
{
fork( );
}


Process1
main ( )
{
fork ( );
Pf
}
main ( )
{
fork ( );
Pf
}

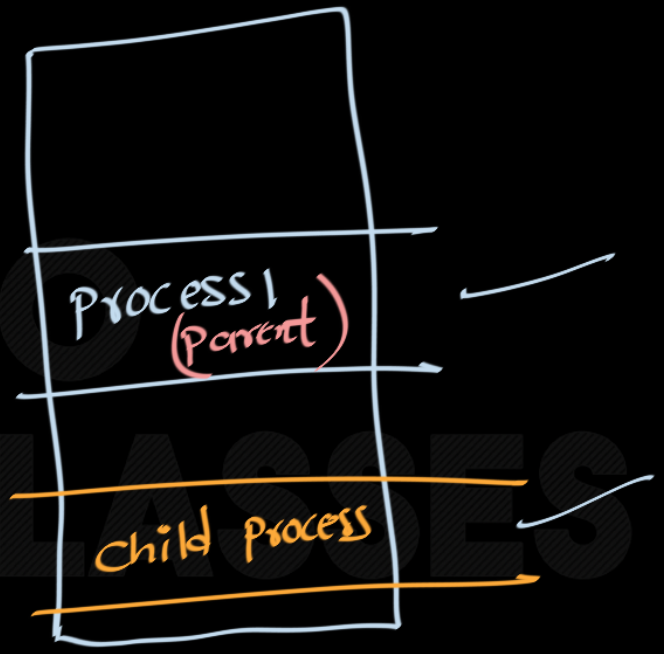
Process1

```
main ( )
{
   fork ( );
   Pf (a)
}
```

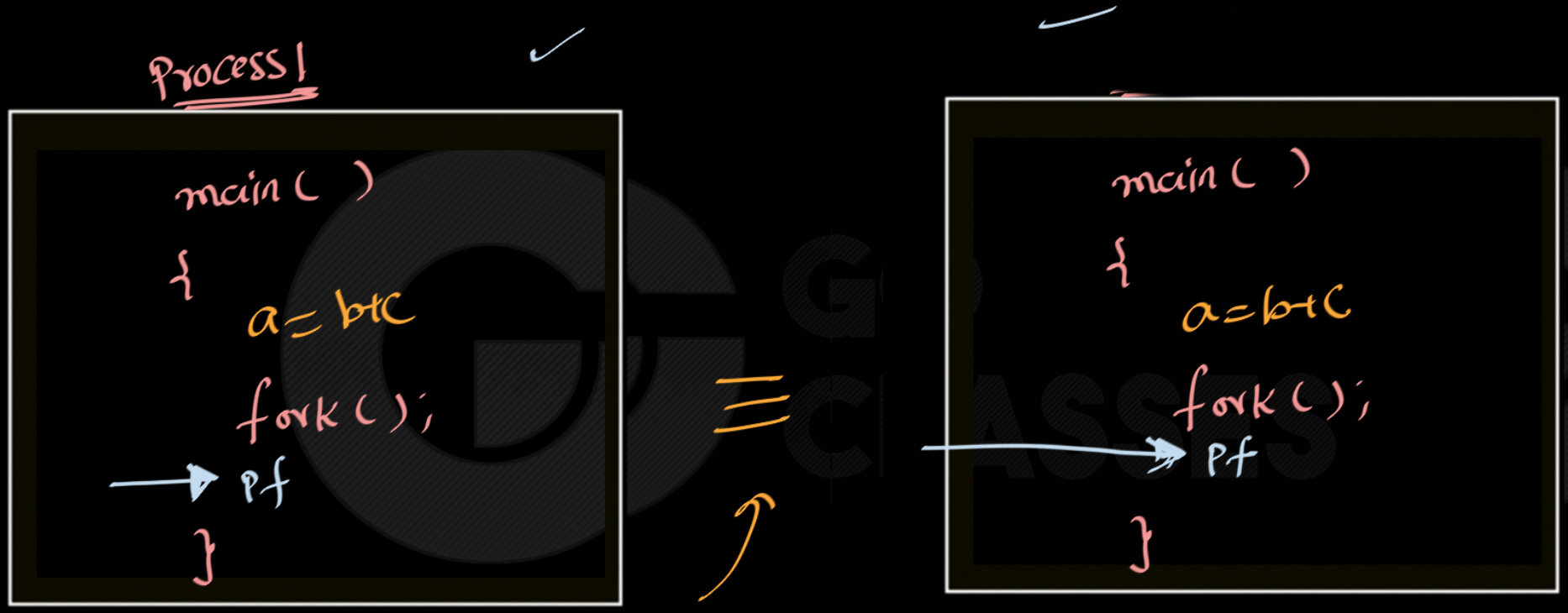
Parent

```
main ( )
{
   fork ( );
   Pf (a)
}
```

Child



Main memory



exact same copy

GO CLASSES

fork ( ) –

1) Creates EXACT same copy

2) Parent and child both will start executing after the line which had fork().

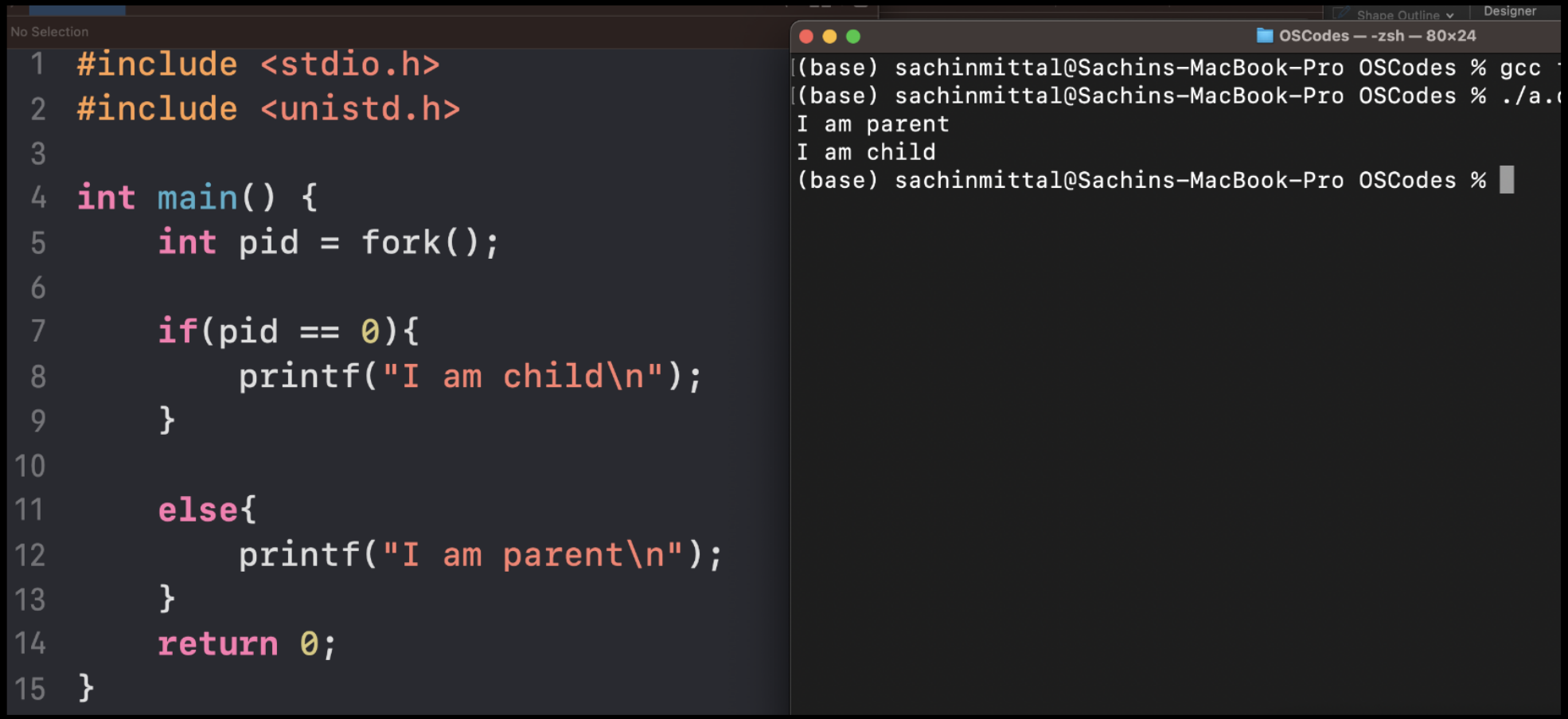
3) fork() will return some integer to both processes

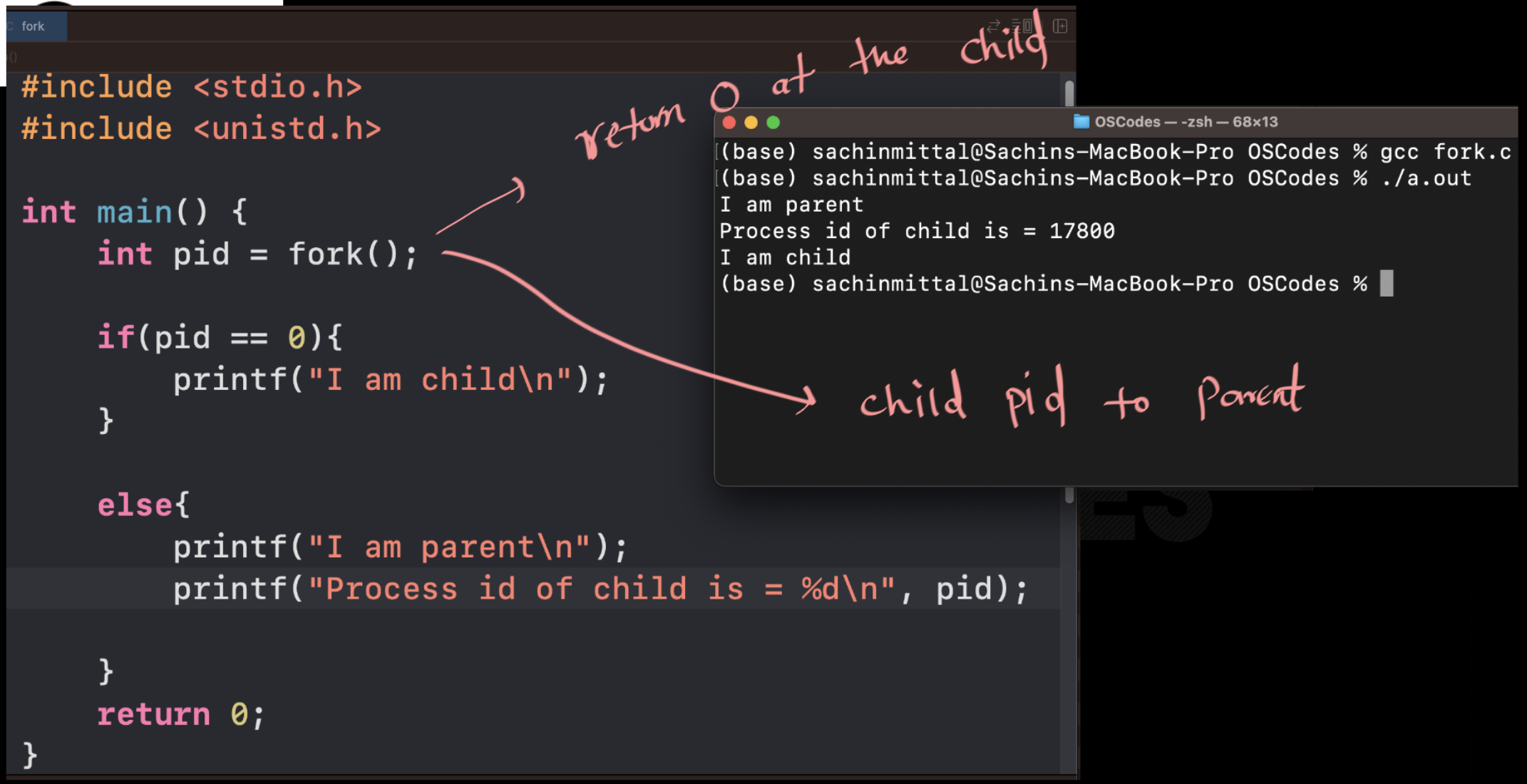
4) fork return "0" to child

5) fork return non zero to parent.

↓

child Process id


No Selection
1 #include <stdio.h>
2 #include <unistd.h>
3
4 int main() {
5 int pid = fork();
6
7 if(pid == 0){
8 printf("I am child\n");
9 }
10
11 else{
12 printf("I am parent\n");
13 }
14 return 0;
15 }
Shape Outline
Designer
OSCodes — -zsh — 80×24
(base) sachinmittal@Sachins-MacBook-Pro OSCodes % gcc
(base) sachinmittal@Sachins-MacBook-Pro OSCodes % ./a.
I am parent
I am child
(base) sachinmittal@Sachins-MacBook-Pro OSCodes %


fork
#include <stdio.h>
#include <unistd.h>
int main() {
int pid = fork();
if(pid == 0){
printf("I am child\n");
}
else{
printf("I am parent\n");
printf("Process id of child is = %d\n", pid);
}
return 0;
}
return 0 at the child
OSCodes — -zsh — 68×13
(base) sachinmittal@Sachins-MacBook-Pro OSCodes % gcc fork.c
(base) sachinmittal@Sachins-MacBook-Pro OSCodes % ./a.out
I am parent
Process id of child is = 17800
I am child
(base) sachinmittal@Sachins-MacBook-Pro OSCodes %
child pid to Parent

GO CLASSES

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    int pid = fork();

    if(pid == 0){
        printf("I am child\n");
    }

    else{
        printf("I am parent\n");
        printf("Process id of child is = %d\n", pid);

    }
    return 0;
}
```

Parent

```c
#include <stdio.h>
#include <unistd.h>

int main() {
    int pid = fork();

    if(pid == 0){
        printf("I am child\n");
    }

    else{
        printf("I am parent\n");
        printf("Process id of child is = %d\n", pid);

    }
    return 0;
}
```

Child

```c
int main() {
    printf("GATE \n");

    int pid = fork();

    if(pid == 0){
        printf("GATEOverflow\n");
    }

    else{
        printf("GO Classes\n");

    }

    printf("IISc\n");

    return 0;
}
```

child

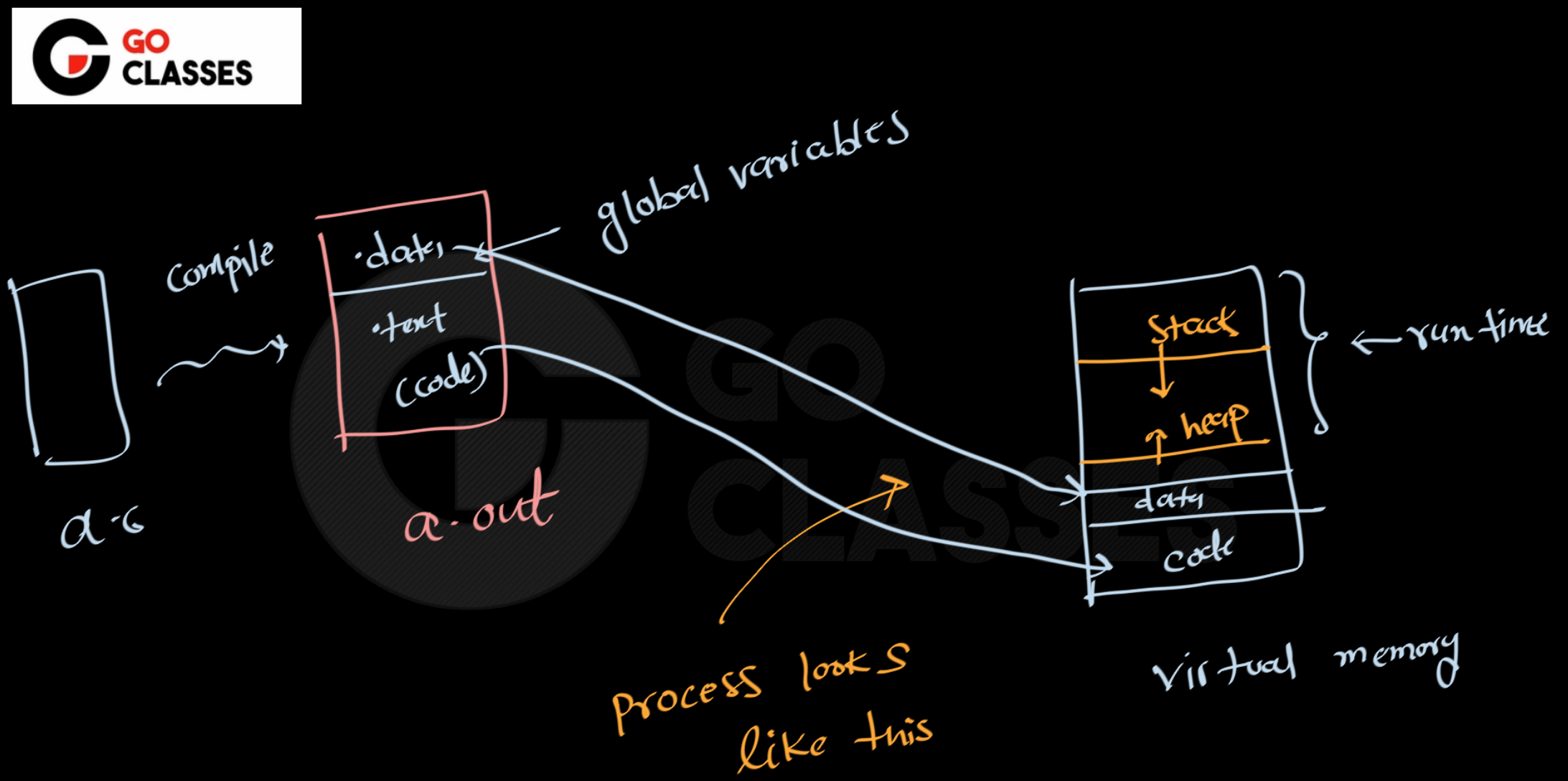
Parent

GATE will get printed
↳ 1 time

GATEOverflow — 1

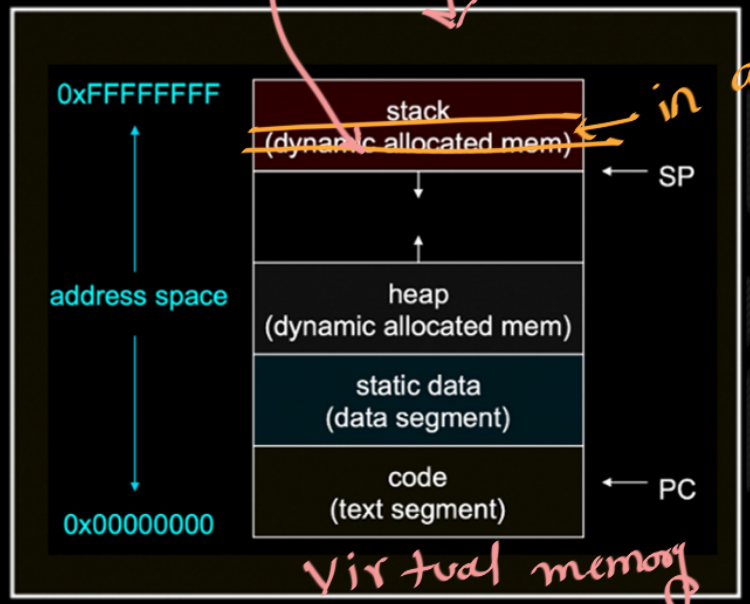
GOclasses — 1

IISc — 2


GO CLASSES
Compile
.data
.text
(code)
global variables
a.o
a.out
Stack
heap
data
code
runtime
Process looks like this
Virtual memory

0X FFFF001C

this is how Process looks like



Virtual memory

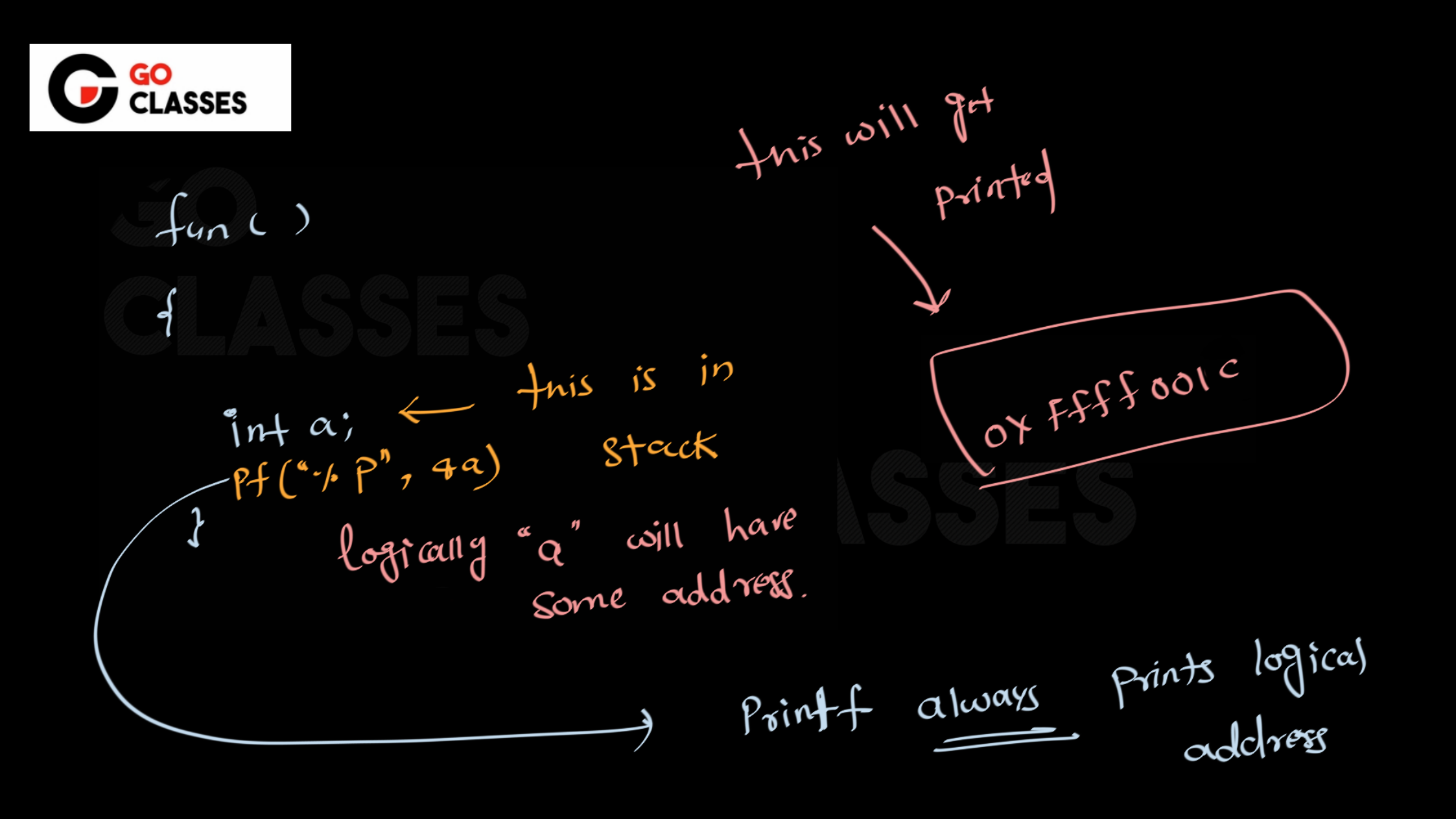
logical memory

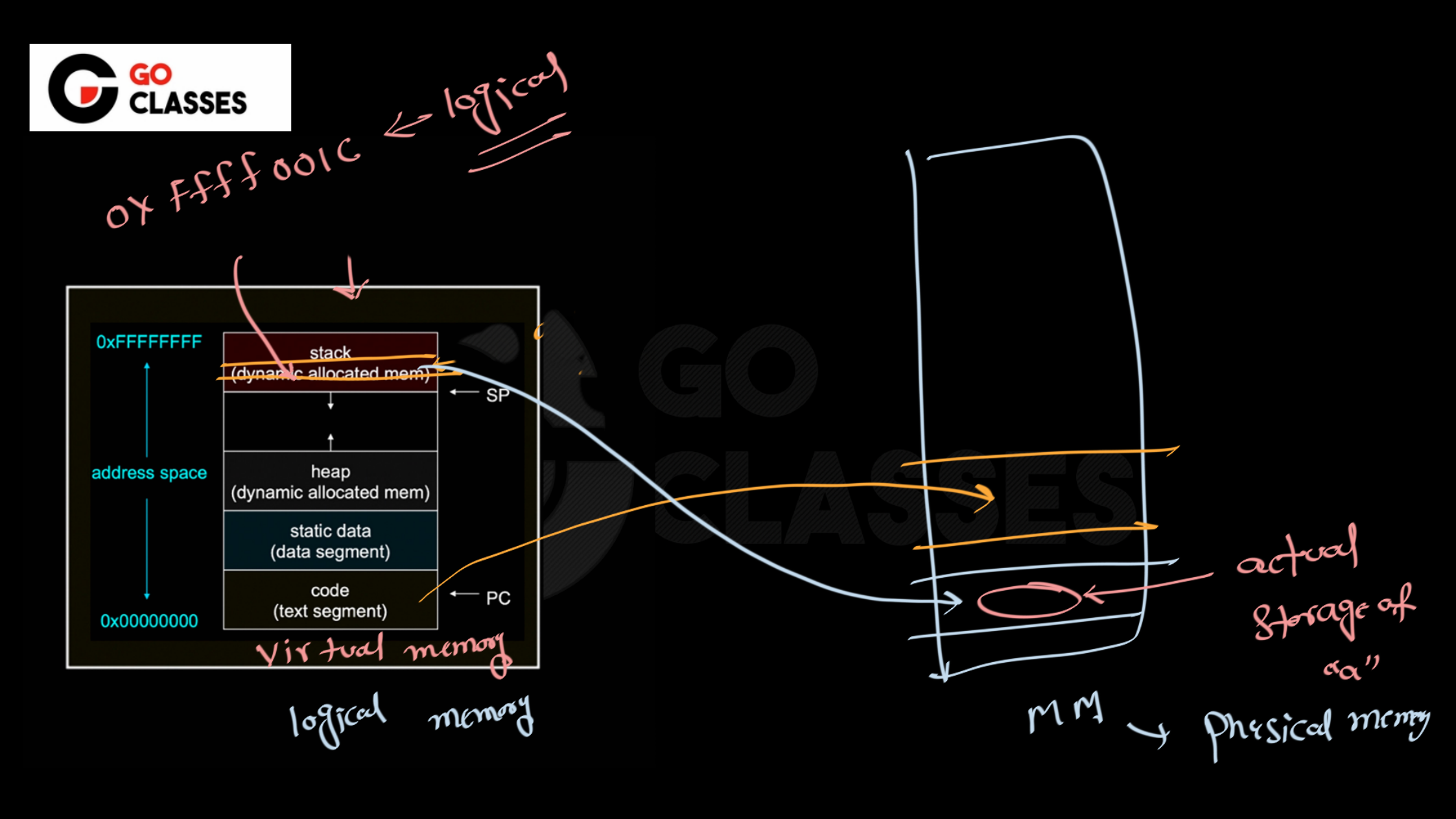
in activation record of fun we have "a"

```
fun ( )
{
    int a;
    Pf("%p", &a)
}
```

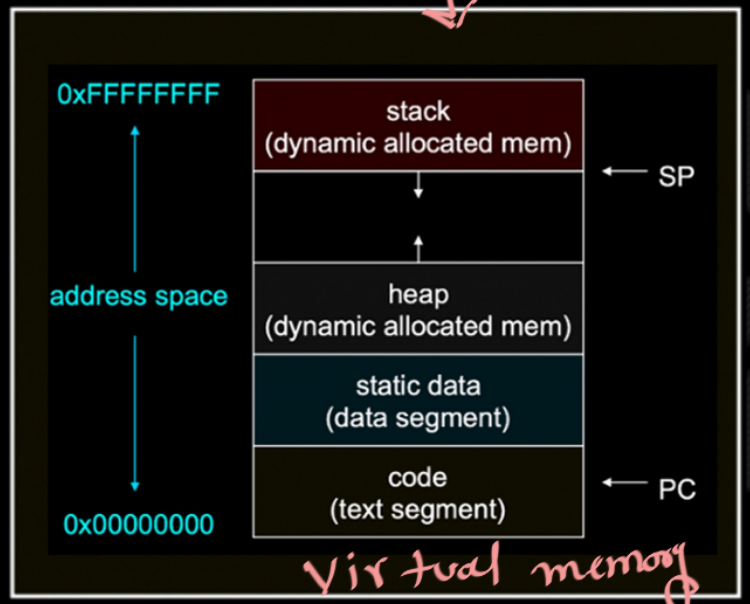
this is in stack

logically "a" will have some address.


GO CLASSES
fun ( )
{
int a;
Pf ("-% P", &a)
}
this is in stack
logically "a" will have some address.
this will get printed
0X ffff001c
Printf always prints logical address

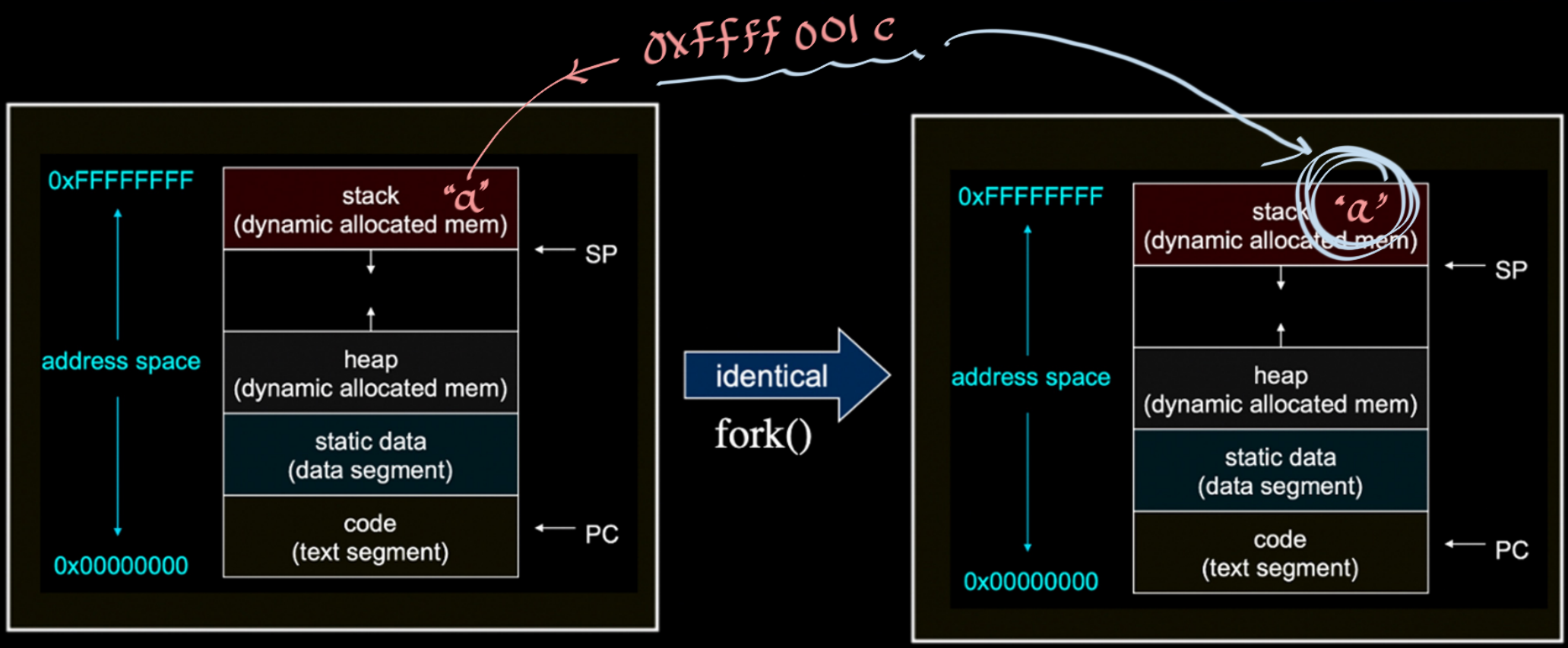

GO CLASSES
0x ffff001c ← logical
0xFFFFFFFF
stack
(dynamic allocated mem)
SP
address space
heap
(dynamic allocated mem)
static data
(data segment)
code
(text segment)
PC
0x00000000
virtual memory
logical memory
actual storage of "a"
MM → physical memory

this is how Process looks like



Virtual memory

logical memory




```
main ( )
{ int a ;
  fork
```

```
main ( )
{
    int a;
    fork ( )
    pf (&a)
}
```

0xffff001c

```
main ( )
{
    int a;
    fork ( )
    pf (&a)
}
```

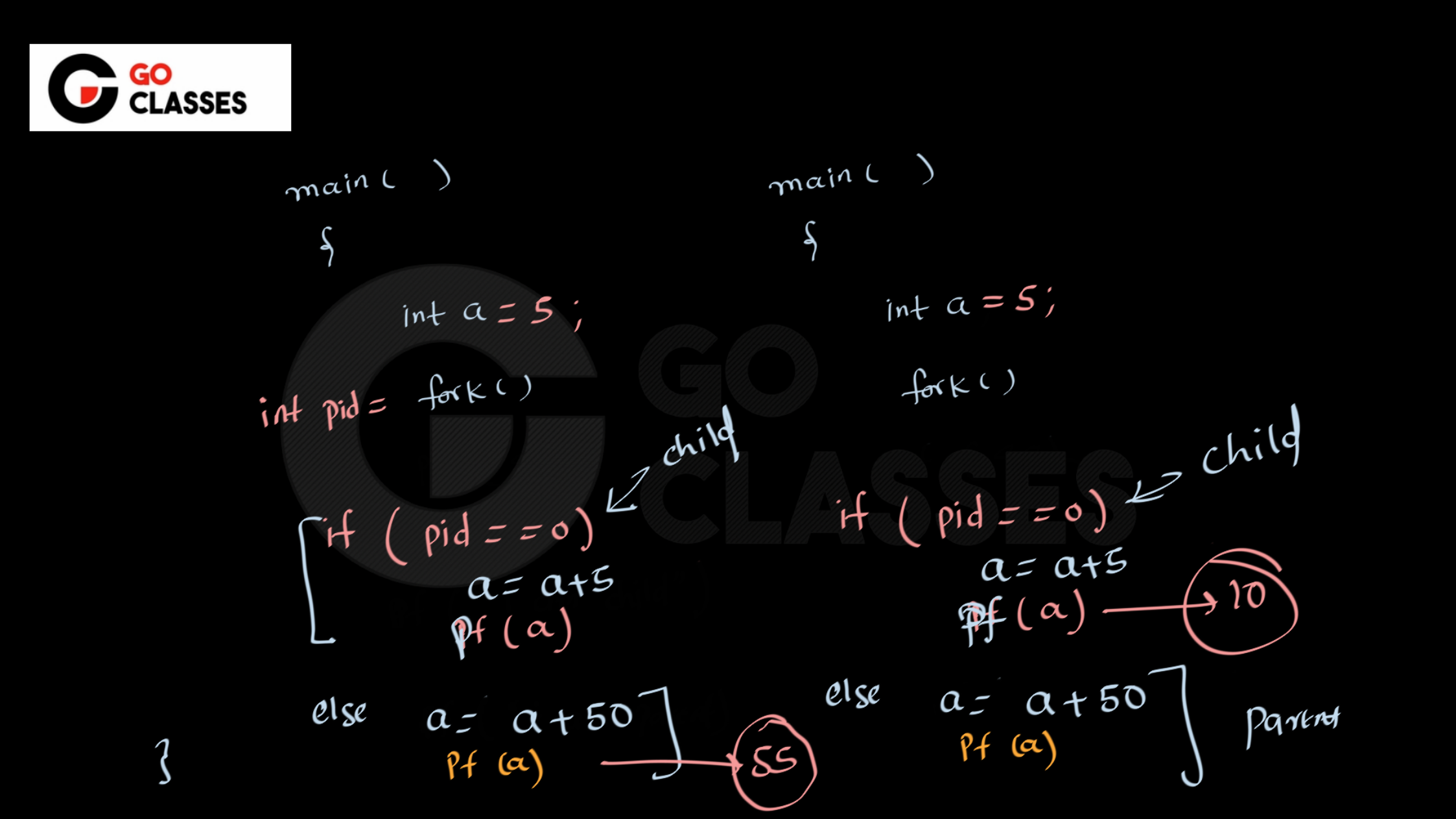
0xffff001c

```
main ( )
{
    int a = 5 ;
int pid = fork ( )
    Pf (&a)
    if ( pid == 0 )
        Pf ("i am child")
    else  Pf ("i am parent")
}
```

0x ffff 001c

```
main ( )
{
    int a = 5 ;
    fork ( )
    Pf (&a)
}
    if ( pid == 0 )
        Pf ("i am child")
    else  Pf ("i am parent")
}
```

0x ffff 001c


main ( )
{
int a = 5 ;
int pid = fork ( )
if (pid == 0)
child
a = a + 5
Pf (a)
else a = a + 50
Pf (a)
55
}
main ( )
{
int a = 5 ;
fork ( )
if (pid == 0)
child
a = a + 5
Pf (a)
10
else a = a + 50
Pf (a)
parent

GO CLASSES

a = 50

```
if (fork() == 0)
{
   a = a + 5;
   printf("%d, %p n", a, &a);
}
else
{
   a = a - 5;
   printf ("%d, %p n", a,& a);
}
```

↳ child

55 ↳ 200

a = 50

```
if (fork() == 0)
{
   a = a + 5;
   printf("%d, %p n", a, &a);
}
else
{
   a = a - 5;
   printf ("%d, %p n", a,& a);
}
```

Parent

45 ↳ 200

GO CLASSES

a = 50

```
if (fork() == 0)
{
    a = a + 5;
    printf("%d, %p n", a, &a);
}
```

↳ child

55 ↳ 200

```
else
{
    a = a - 5;
    printf ("%d, %p n", a,& a);
}
```

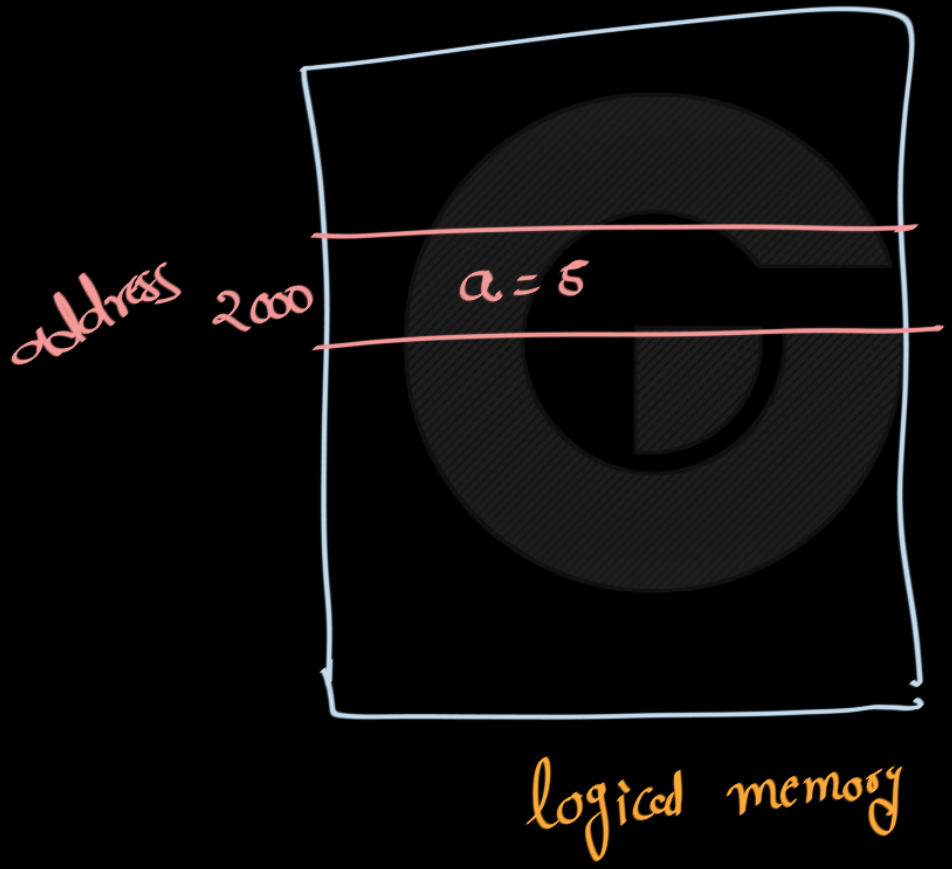
Parent

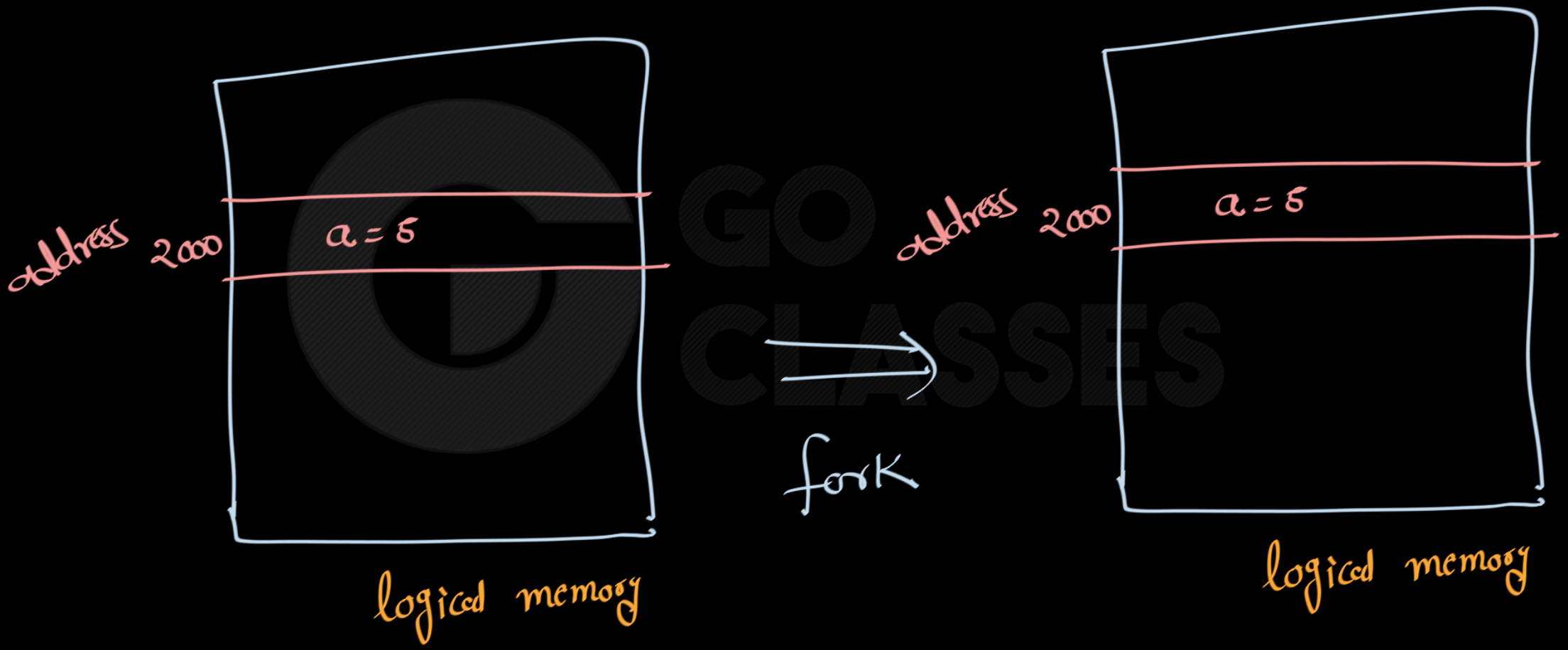
a = 50

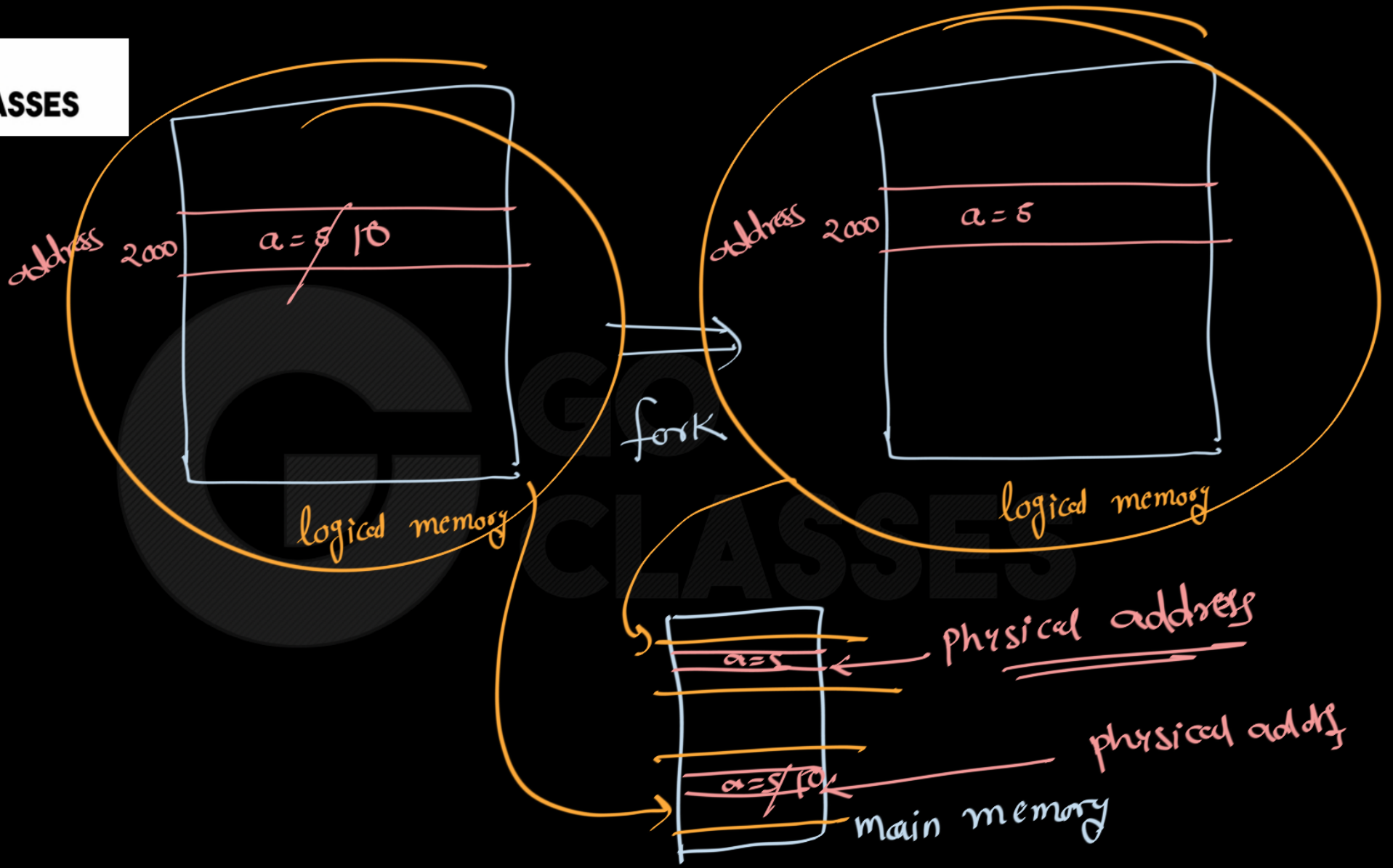
45 ↳ 200

always

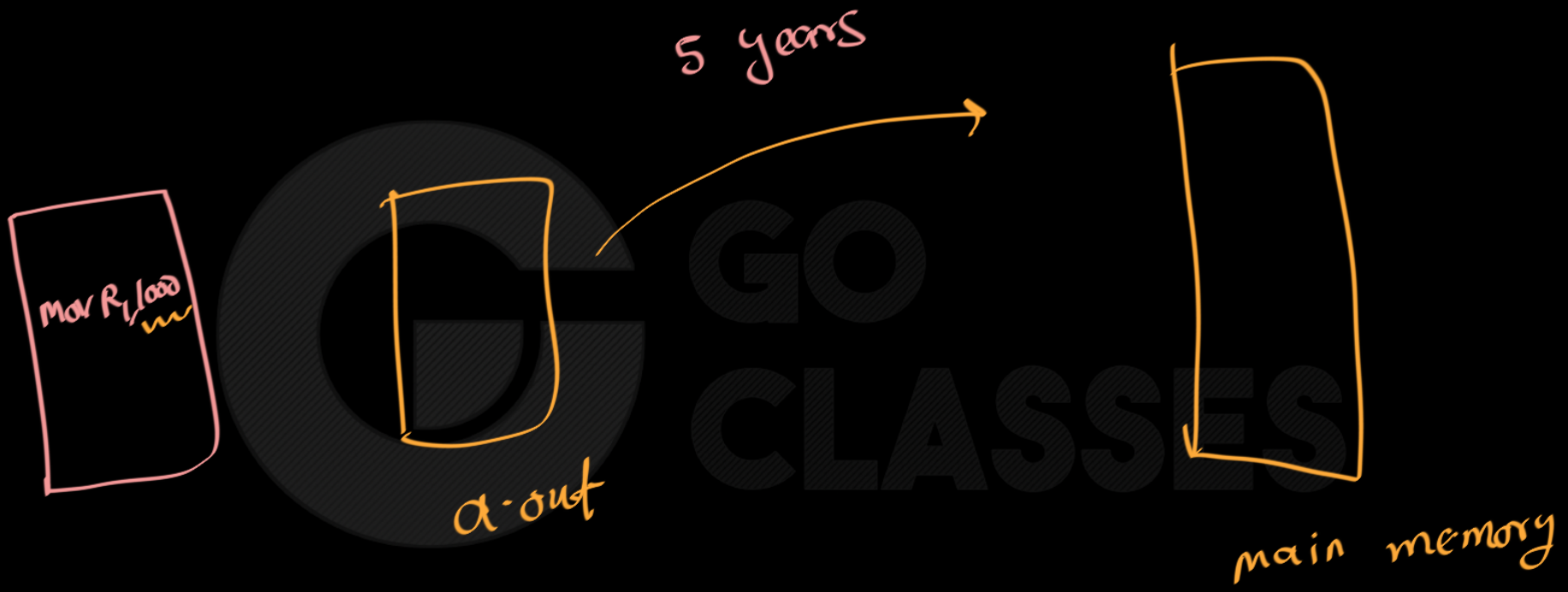
Child ⇒ Parent

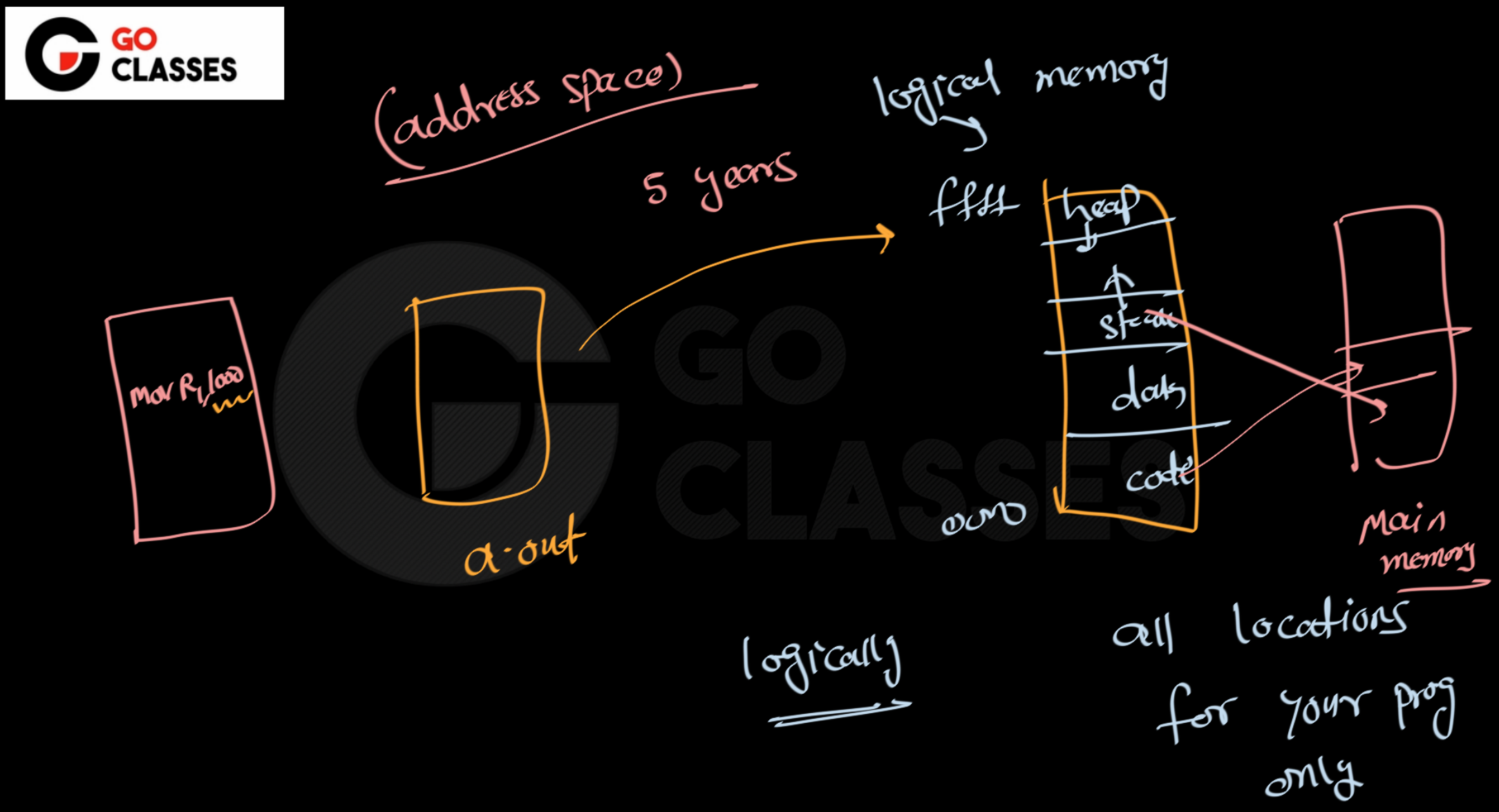

address
2000
a = 5
logical memory


address
2000
a = 5
logical memory
fork
address
2000
a = 5
logical memory


address
2000
a = 5 / 10
logical memory
fork
address
2000
a = 5
logical memory
a=5
Physical address
a=5/10
physical addrf
main memory


Mov R1 1000
logical address


5 years
mov R1,1000
a.out
main memory


(address space)
logical memory
5 years
mov R1, 1000
a.out
heap
stack
data
code
Main memory
logically
all locations for your prog only

fork Creates EXACT same copies

hence logical addresses (printed by printf)

are same but physically variables are

stored at different locations.

GO CLASSES

does not matter even if global variable

```c
int a = 50;

int main() {

    int pid = fork();

    if(pid == 0){
        a = a+5;
        printf("I am child and a = %d \n", a);
        printf("I am child and &a = %p \n", &a);

    }

    else{
        a = a-5;
        printf("I am parent and a = %d \n", a);
        printf("I am parent and &a = %p \n", &a);
    }

    return 0;
}
```

```
OSCodes — -zsh — 68×13
(base) sachinmittal@Sachins-MacBook-Pro OSCodes % gcc f
(base) sachinmittal@Sachins-MacBook-Pro OSCodes % ./a.o
I am parent and a = 45
I am parent and &a = 0x1007d4000
I am child and a = 55
I am child and &a = 0x1007d4000
(base) sachinmittal@Sachins-MacBook-Pro OSCodes %
```

GATE CSE 2005 | Question: 72

27,646 views

98

```
Consider the following code fragment:
if (fork() == 0)
{
   a = a + 5;
   printf("%d, %p n", a, &a);
}
else
{
   a = a - 5;
   printf ("%d, %p n", a,& a);
}
```

Let u, v be the values printed by the parent process and x, y be the values printed by the child process. Which one of the following is **TRUE**?

A. $u = x + 10$ and $v = y$
B. $u = x + 10$ and $v! = y$
C. $u + 10 = x$ and $v = y$
D. $u + 10 = x$ and $v! = y$

GATE CSE 2005 | Question: 72

27,646 views

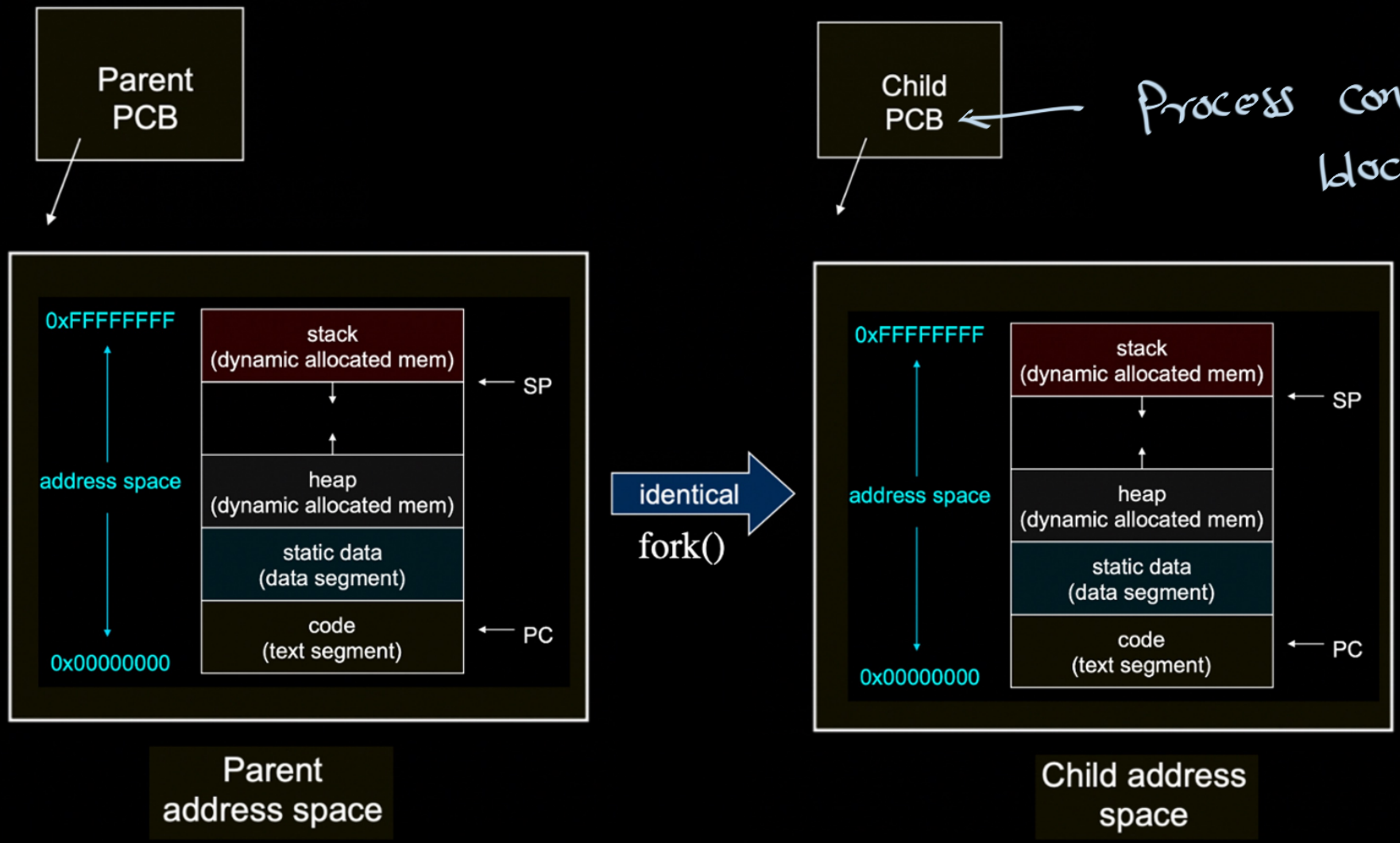
98

```
Consider the following code fragment:
if (fork() == 0)
{
   a = a + 5;
   printf("%d, %p n", a, &a);
}
else
{
   a = a - 5;
   printf ("%d, %p n", a,& a);
}
```

Let u, v be the values printed by the parent process and x, y be the values printed by the child process. Which one of the following is **TRUE**?

A. $u = x + 10$ and $v = y$
B. $u = x + 10$ and $v! = y$
C. $u + 10 = x$ and $v = y$
D. $u + 10 = x$ and $v! = y$

Operating Systems



What will be the output of following program ?

```
int main() {

    printf("one\n");
    fork();
    printf("two\n");
    return 0;
}
```

What will be the output of following program ?

Parent

```
int main() {

    printf("one\n");
    fork();
    printf("two\n");
    return 0;
}
```

child

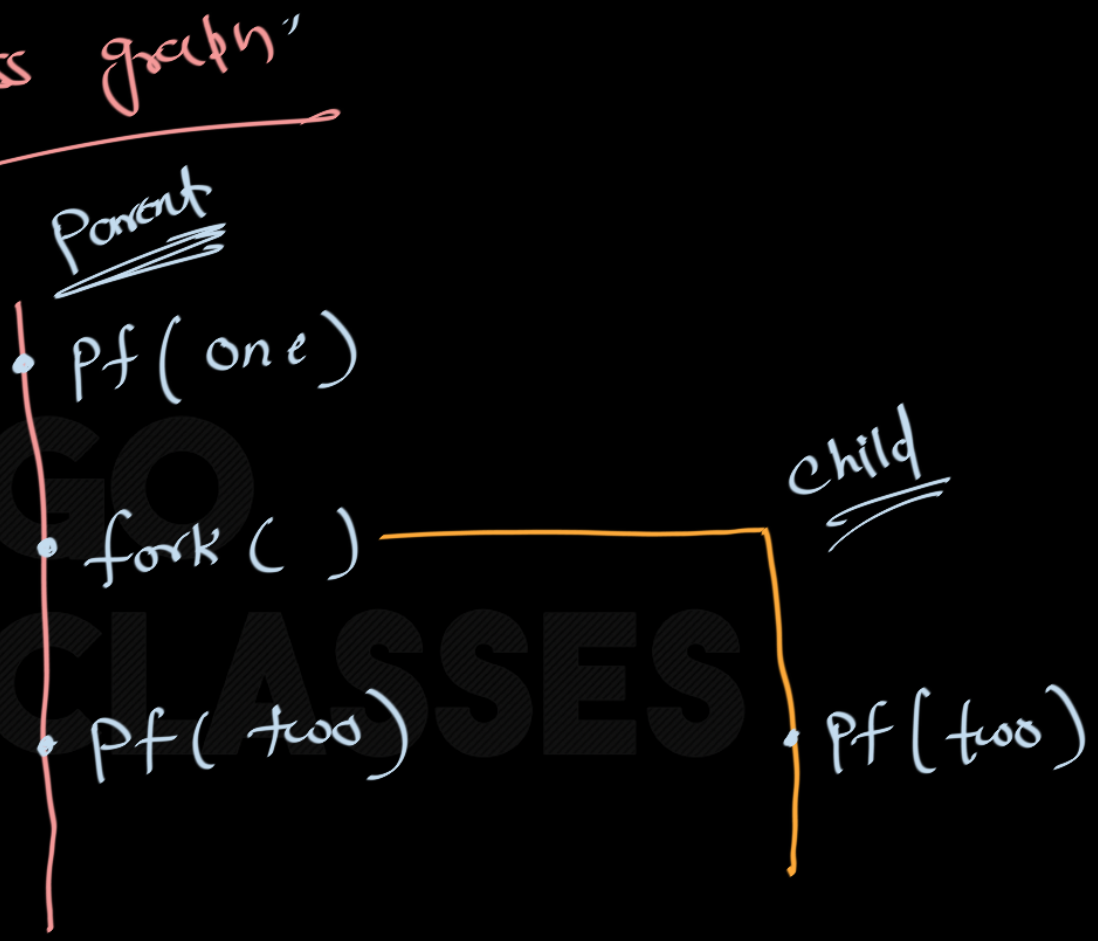
```
int main() {

    printf("one\n");
    fork();
    printf("two\n");
    return 0;
}
```

output

one
two
two

GO CLASSES

```c
int main() {

    printf("one\n");
    fork();
    printf("two\n");
    return 0;
}
```

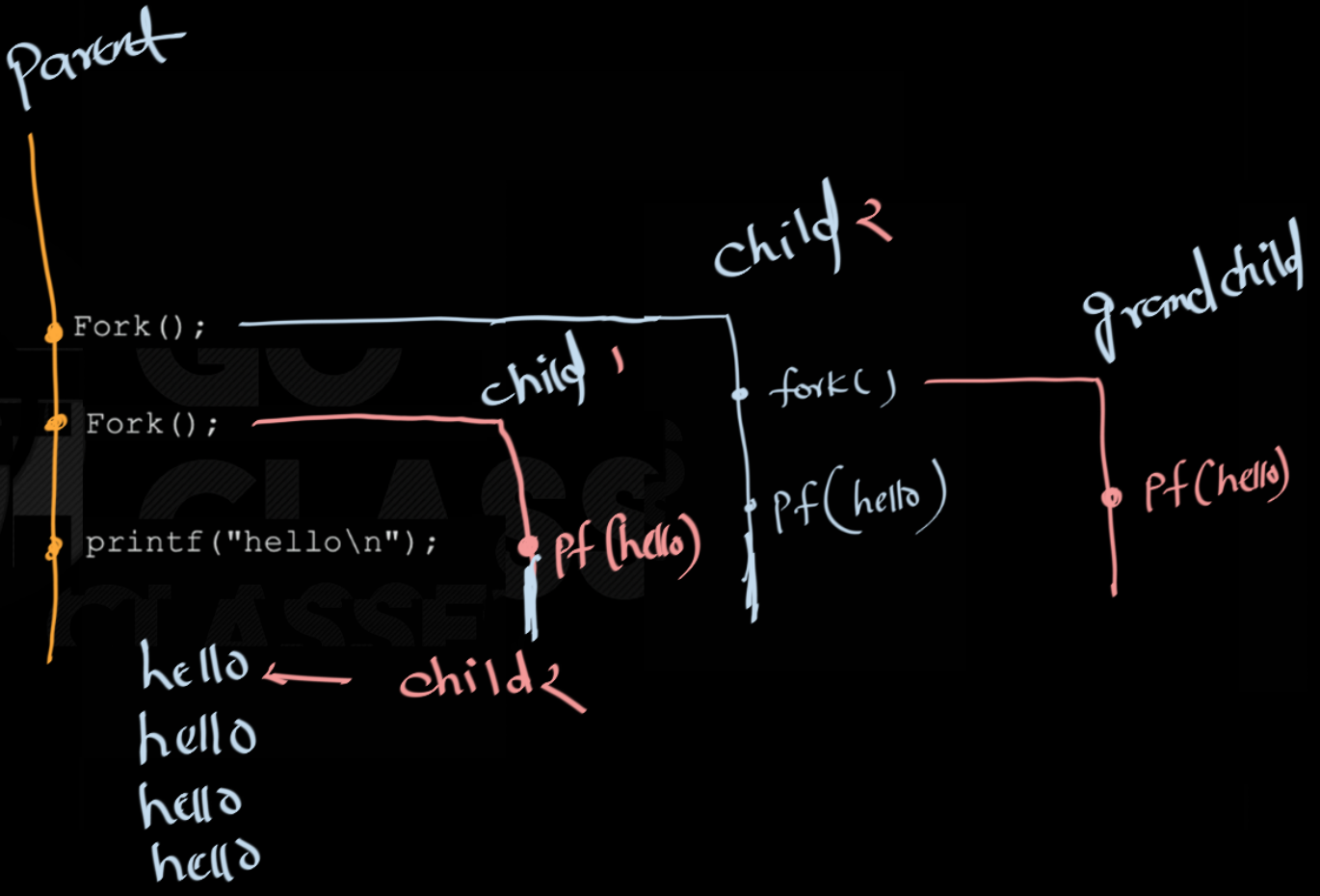


Fork questions

Process graph

Question 1

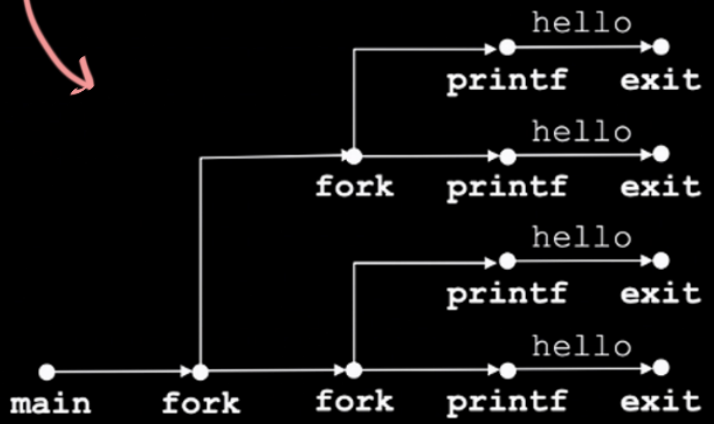
```
1 int main()
2 {
3     Fork();
4     Fork();
5     printf("hello\n");
6     exit(0);
7 }
```



No. of processes = 4

```
1 int main()
2 {
3     Fork();
4     Fork();
5     printf("hello\n");
6     exit(0);
7 }
```

Process graph



Question 2

1. How many Hello's are printed. (Assume fork does not fail.)

```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
    printf("Hello\n");
  }
  return 0;
}
```

1. 2
2. 4
3. 6
4. 8

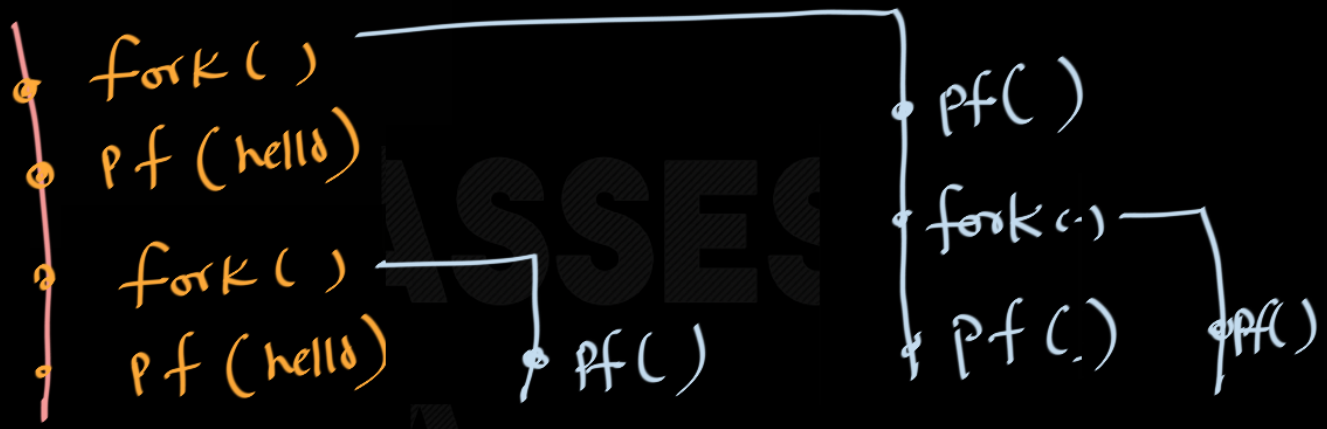
Question 2

1. How many Hello's are printed. (Assume fork does not fail.)

```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
    printf("Hello\n");
  }
  return 0;
}
```

1. 2
2. 4
3. 6
4. 8

6 times pf hello.


fork ( )
Pf (hello)
fork ( )
Pf (hello)
Pf( )
Pf( )
fork (-)
Pf (-)
Pf( )

1. How many Hello's are printed. (Assume fork does not fail.)

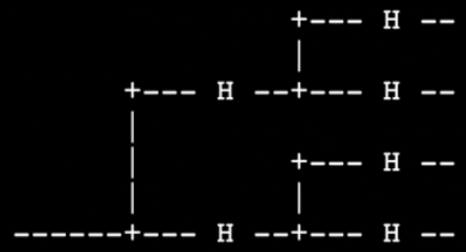
```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
    printf("Hello\n");
  }
  return 0;
}
```

1. 2
2. 4
3. 6 ✓
4. 8

Solution Remarks: This code is equivalent to:

```
fork();
printf("Hello\n");
fork();
printf("Hello\n");
```

Now draw the diagram (where H is for the print statement):



Question 3

How many Hello's are printed. (Assume fork does not fail.)

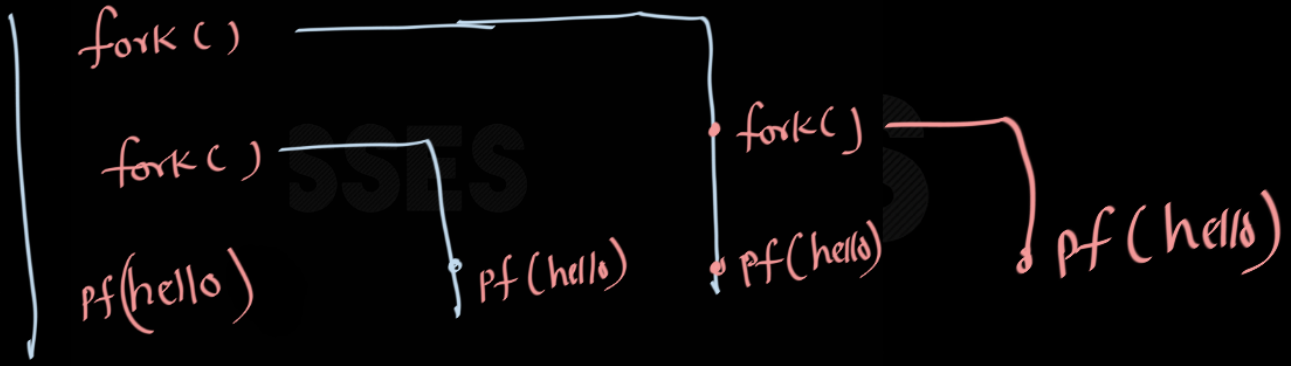
```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
  }
  printf("Hello\n");
  return 0;
}
```

1. 2
2. 4
3. 6
4. 8

Question 3

How many Hello's are printed. (Assume fork does not fail.)

```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
  }
  printf("Hello\n");
  return 0;
}
```

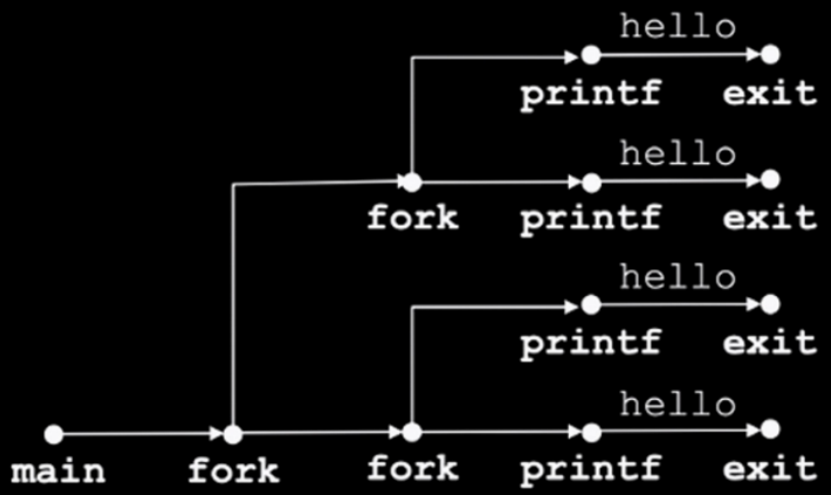


1. 2
2. 4
3. 6
4. 8

```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
  }
  printf("Hello\n");
  return 0;
}
```

This code is equivalent to:

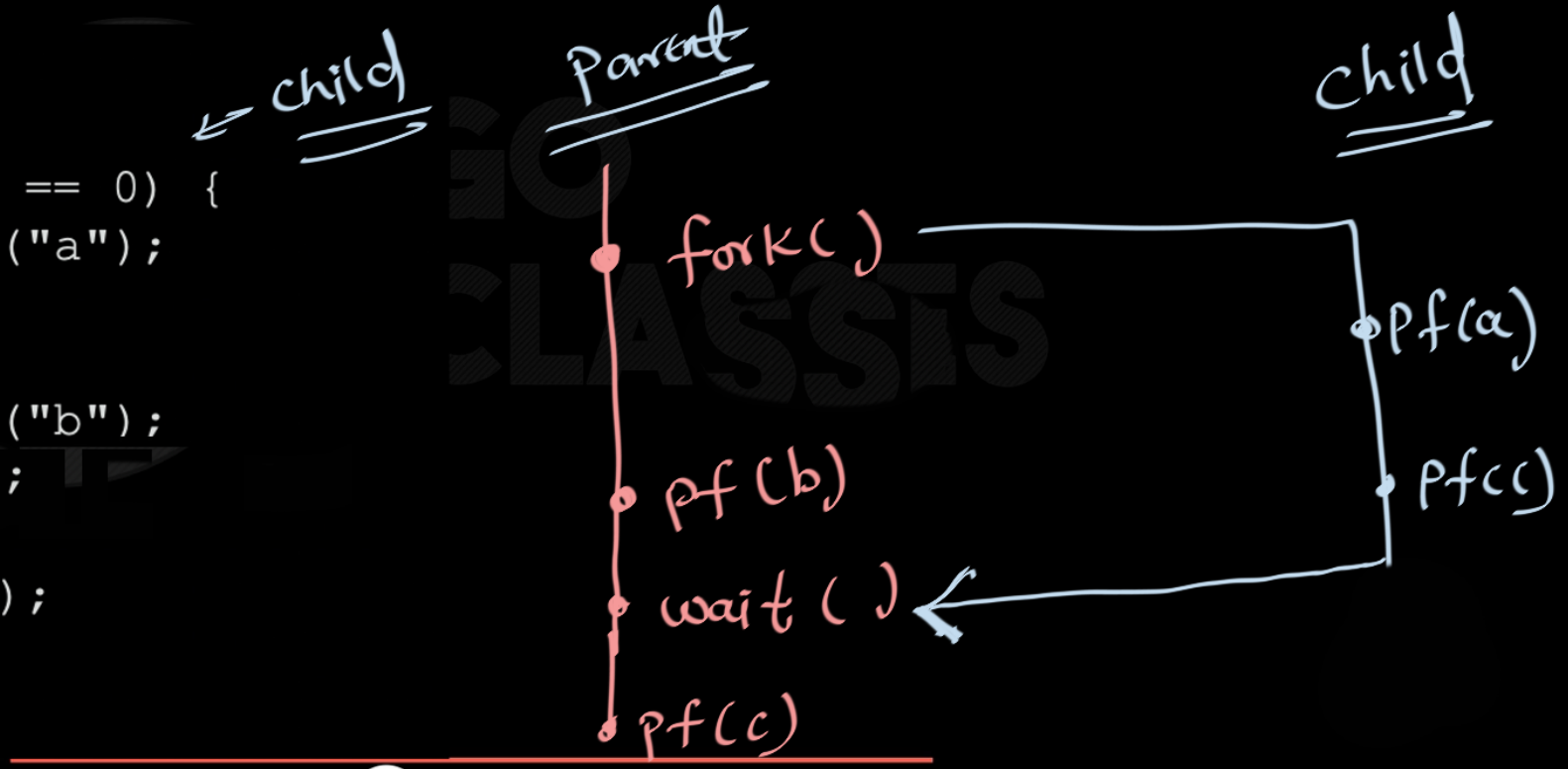
```
Fork();
Fork();
printf("hello\n");
exit(0);
```

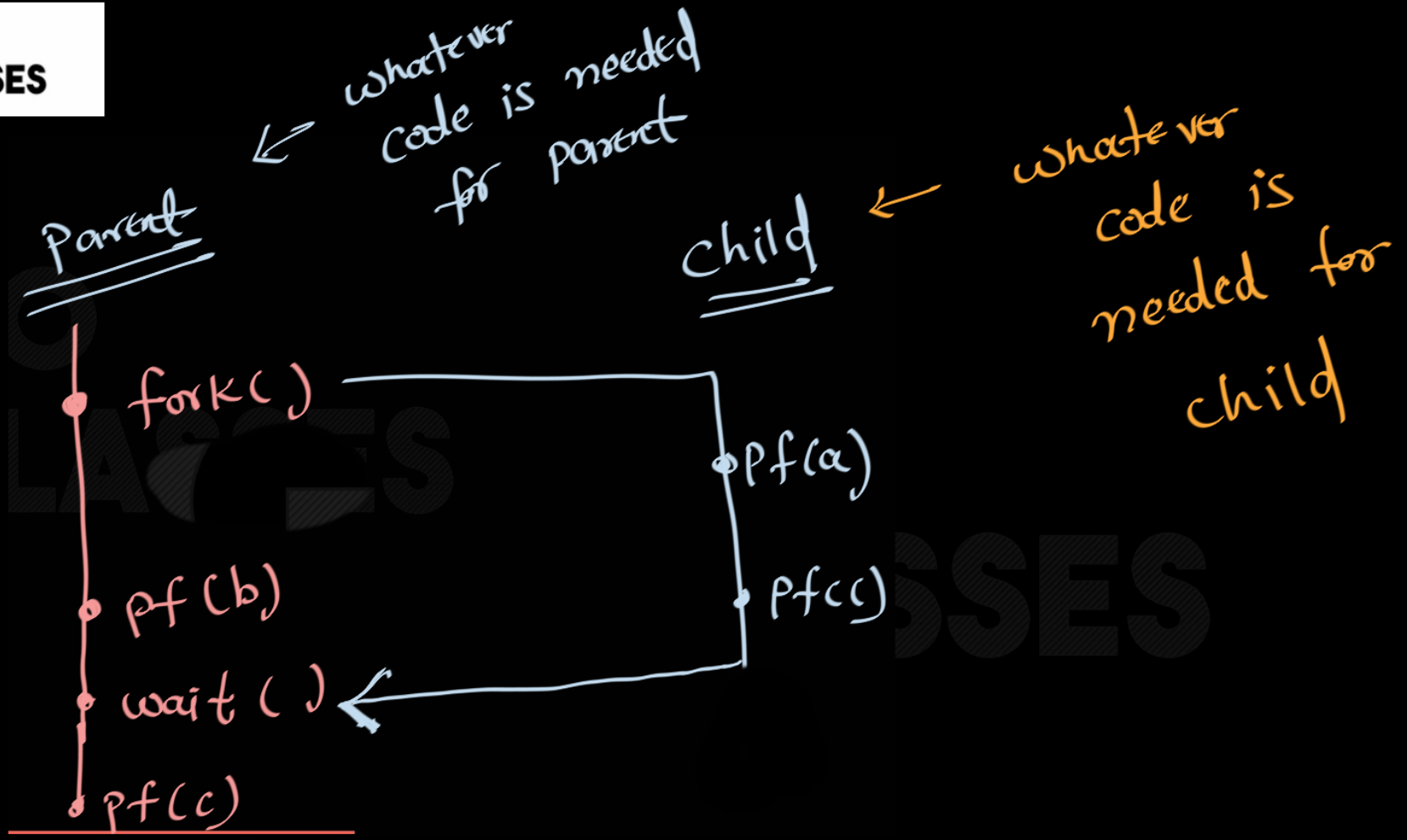


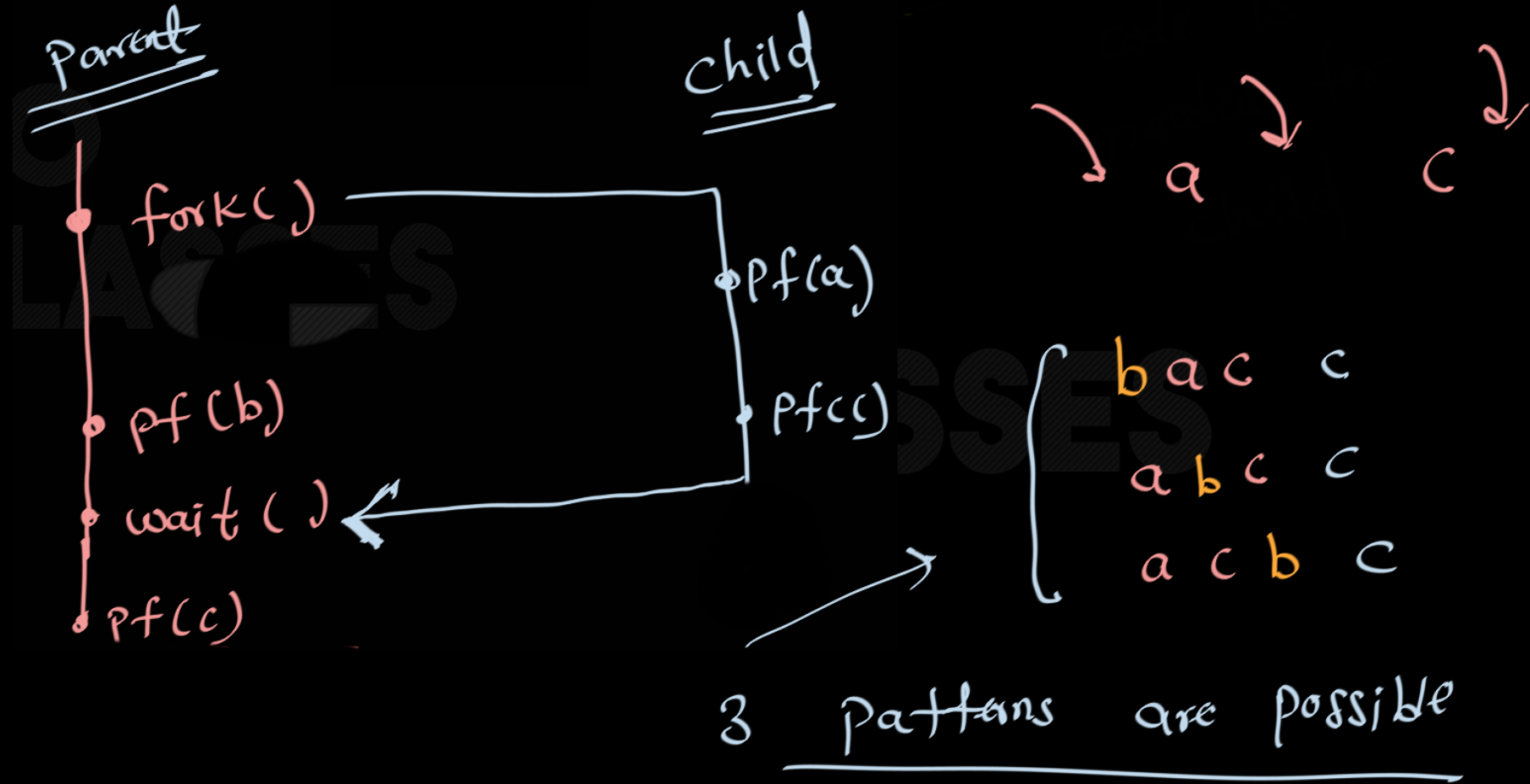
Question 4

Write down all possible patterns printed by following code.

```
1  int main()
2  {
3      if (Fork() == 0) {
4          printf("a");
5      }
6      else {
7          printf("b");
8          wait();
9      }
10     printf("c");
11     exit(0);
12 }
```

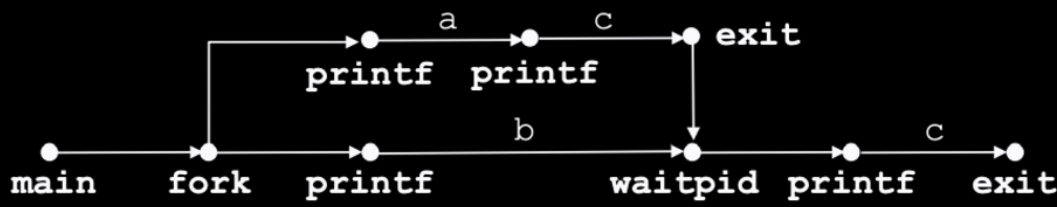



Parent
whatever code is needed for parent
Child
whatever code is needed for child
fork()
pf(b)
wait()
pf(c)
pf(a)
pf(c)


Parent
Child
fork()
pf(a)
pf(b)
pf(c)
wait()
pf(c)
a
c
b a c c
a b c c
a c b c
3 Patterns are possible

Write down all possible patterns printed by following code.

```
1  int main()
2  {
3      if (Fork() == 0) {
4          printf("a");
5      }
6      else {
7          printf("b");
8          waitpid(-1, NULL, 0);
9      }
10     printf("c");
11     exit(0);
12 }
```



GO CLASSES

SUMMARY

→ we talked about Process
↳ a program in execution

→ PCB (context)

→ How to create a process
↳ create space fork
↳ load in that space exec

→ fork creates identical copy of address spaces (logical address)

we talked little about what is logical memory. (we say to the compiler that whole memory is for you)

→ copies by fork are different physically (of course!)

→ we understood fork() copies by

Printing addresses and changing values

↓ ✓

same change to one thing does not reflect in another process.

→ order of execution of child and parent depend on scheduling

GO CLASSES

Biswanath to Everyone 10:57 PM

B

```
if(pid==0)
{
child will execute
}
else {
parent  will execute
}
```

🔥 3 💯 2

Process graph.

Biswanath 10:57 PM

B

- order of execution depends on scheduling(parent ,child)

What if global variables or statice variable used in the code?(no difference ,have same logical address but different physical addres)

Question 5

How many child process will create ?

```
for ( i = 1; i<=3; i++ )
{
   fork();
}
```

Question 6

How many child process will create ?

```
for ( i = 1; i<=3; i++ )
{
   fork();
   fork();

}
```

Question

GATE CSE 2008 | Question: 66

14,032 views

24

A process executes the following code

```
for(i=0; i<n; i++) fork();
```

The total number of child processes created is

A. n
B. $2^n - 1$
C. 2^n
D. $2^{n+1} - 1$

gatecse-2008 operating-system fork-system-call normal

Question 8

How many processes are created with these fork() statements?

15

11

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  pid_t pid = fork();
  pid = fork();
  pid = fork();
  if (pid == 0)
  {
    fork();
  }
  fork();
  return 0;
}
```

https://stackoverflow.com/questions/19106576/how-many-processes-are-created-with-these-fork-statements

It's fairly easy to reason through this. The `fork` call creates an additional process every time that it's executed. The call returns `0` in the new (child) process and the process id of the child (not zero) in the original (parent) process.

```
pid_t pid = fork();  // fork #1
pid = fork();        // fork #2
pid = fork();        // fork #3
if (pid == 0)
{
  fork();            // fork #4
}
fork();              // fork #5
```

1. Fork #1 creates an additional processes. You now have two processes.
2. Fork #2 is executed by two processes, creating two processes, for a total of four.
3. Fork #3 is executed by four processes, creating four processes, for a total of eight. Half of those have `pid==0` and half have `pid != 0`
4. Fork #4 is executed by half of the processes created by fork #3 (so, four of them). This creates four additional processes. You now have twelve processes.
5. Fork #5 is executed by all twelve of the remaining processes, creating twelve more processes; you now have twenty-four.