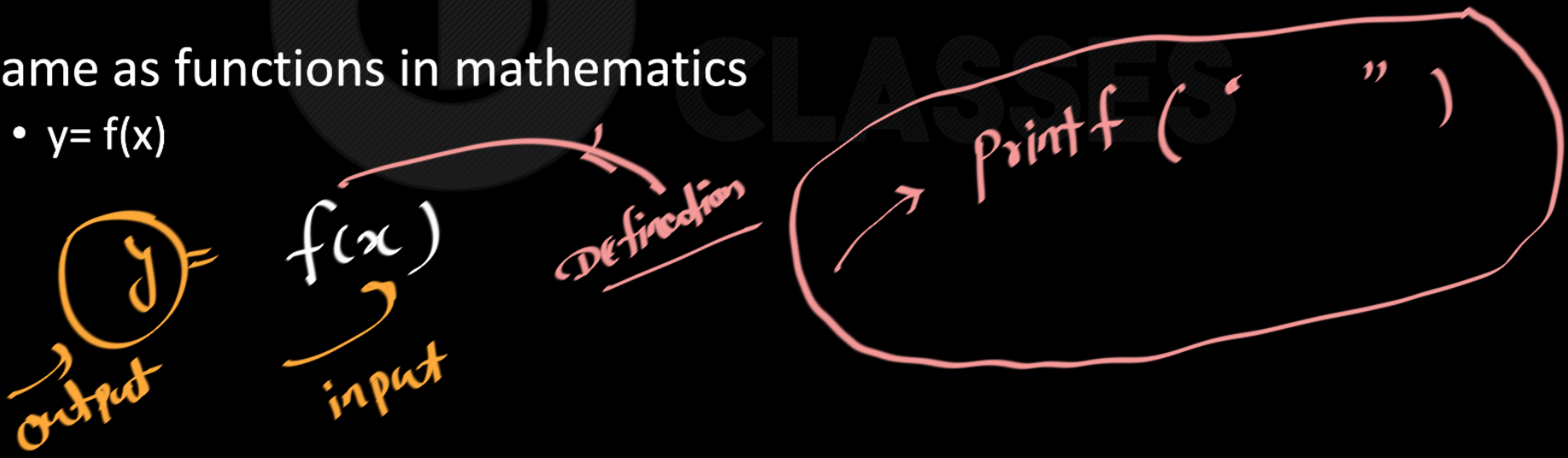
Functions in C

Basic Idea of a Function

- Real world programs often are large and complex
- Easier to manage in smaller pieces, like functions

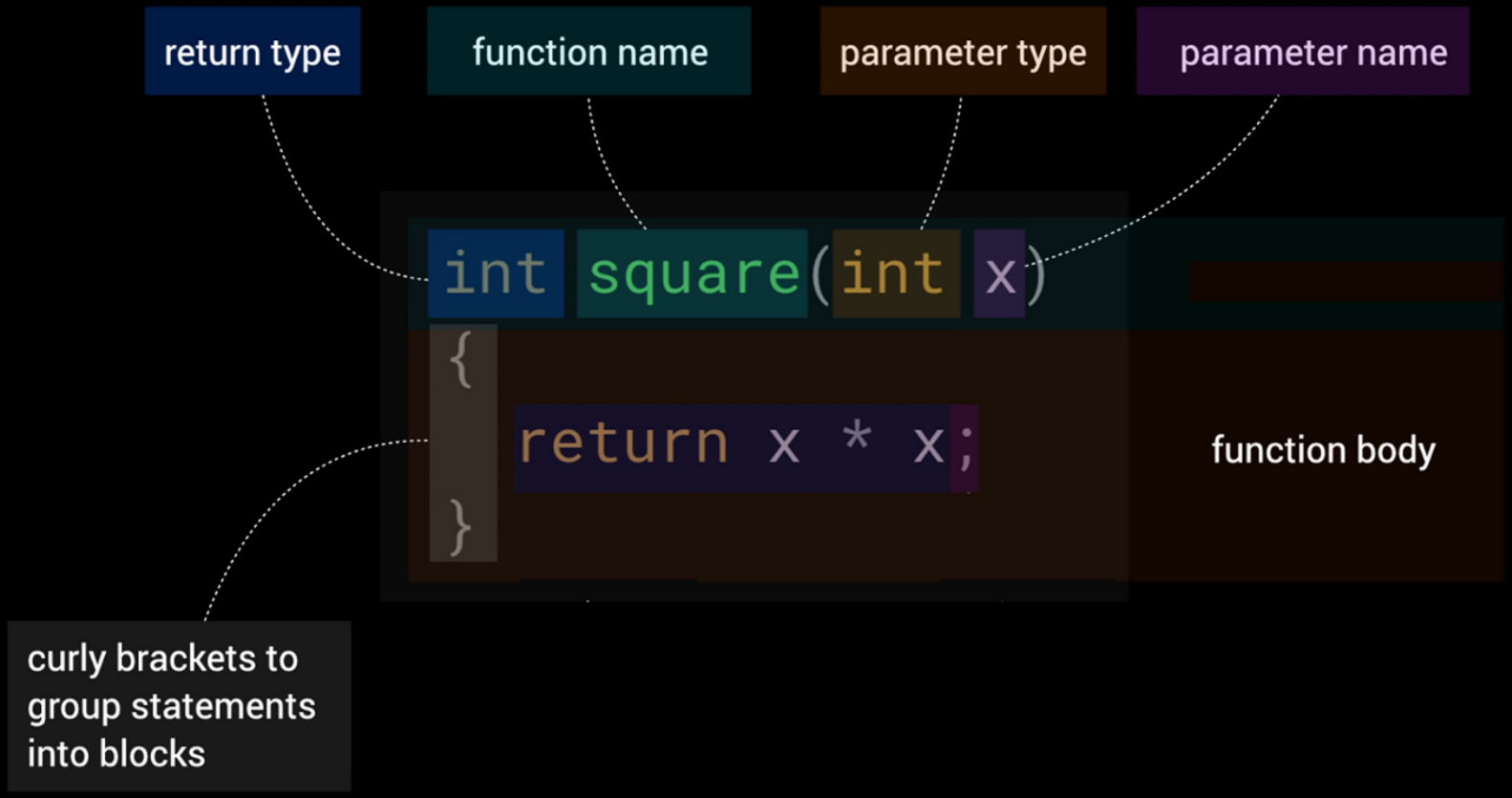
← We need to make it readable

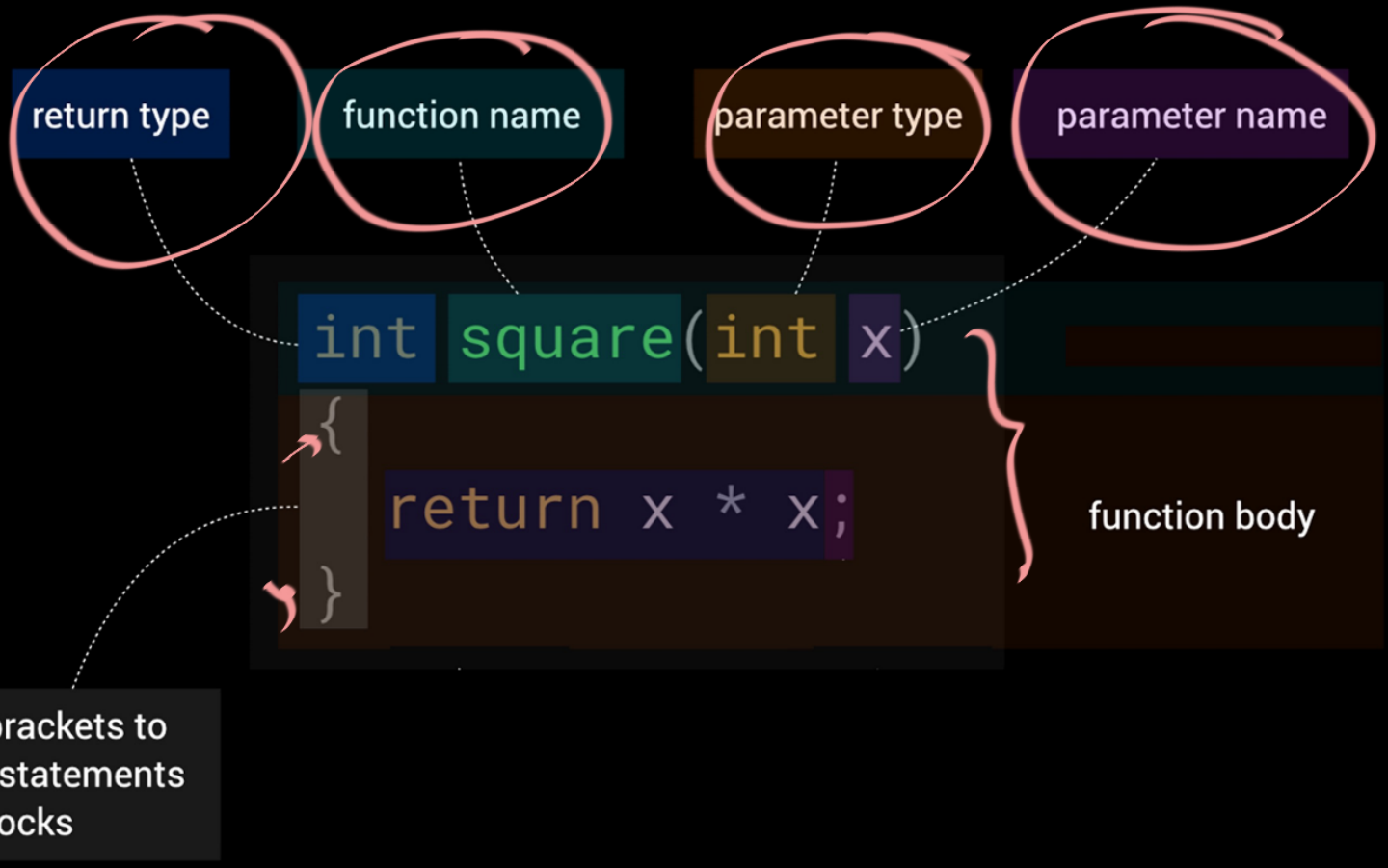
- Same as functions in mathematics
  - y= f(x)



Functions in C

```
function_type function_name(parameter list)
{
    . . . . .
    . . . . .

return statement;
}
```


return type
function name
parameter type
parameter name
int square(int x)
{
return x * x;
}
function body
curly brackets to group statements into blocks


return type
function name
parameter type
parameter name
int square(int x)
{
return x * x;
}
function body
curly brackets to group statements into blocks

Definition

```
type fname(type arg1, type arg2)
{
 /* Local variables and executable code */
}
```

Calling a function

```
fname(arg1, arg2);
```

Prototype / declaration

```
type fname(type, type);
```

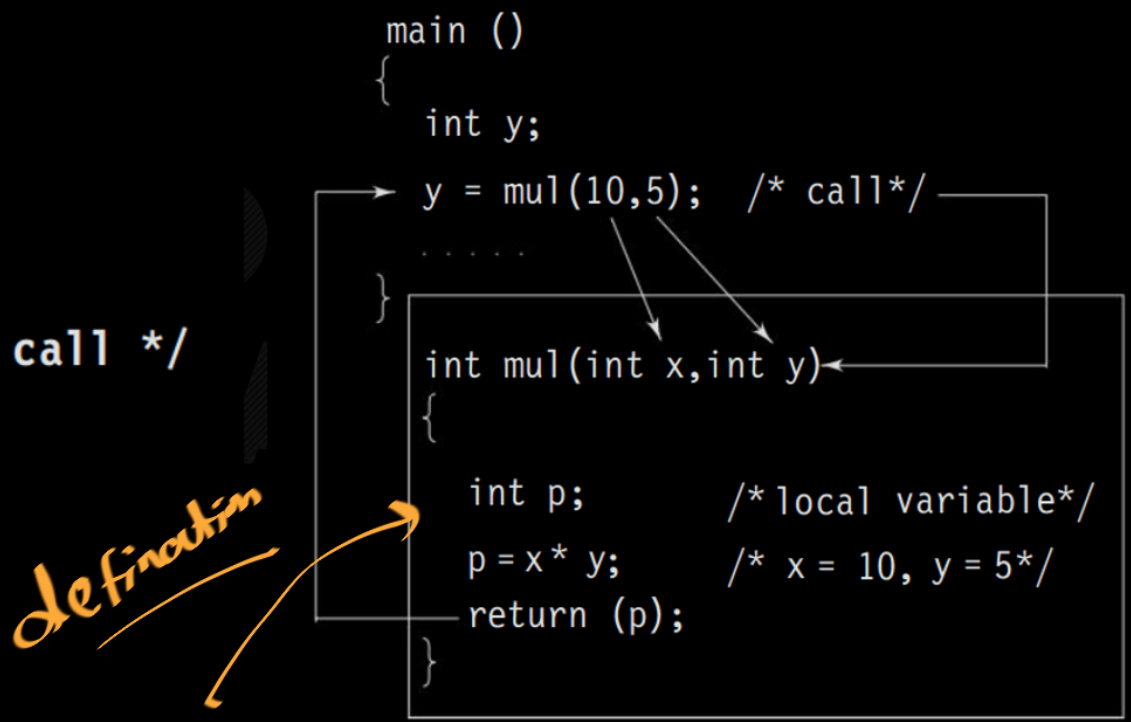
Example of function

```c
float mul (float x, float y)
{
    float result; /* local variable */
    result = x * y; /* computes the product */
    return result; /* returns the result */
}
```

```c
void display (void)
{ /* no local variables */
printf ("No type, no parameters");
/* no return statement */
}
```

```c
void sum (int a, int b)
{
    printf ("sum = %s", a + b); /* no local variables */
    return; /* optional */
}
```

```
main( )
{
   int y;
   y = mul(10,5);          /* Function call */
   printf("%d\n", y);
}
```



Function Declaration

- Function declaration also known as function prototype

```
int mul (int m, int n);  /* Function prototype */
```

return type

input type

Equally acceptable forms of declaration

```
int   mul   (int a, int b);
int   mul   (int, int);
      mul   (int a, int b);
      mul   (int, int);
```

By default return type is integer

Using a function

```c
#include <stdio.h>

int fun(int x, int y); //function declaration
main()
{
    int a;
    a = fun(2, 3);

    printf("%d", a);
}

int fun(int x, int y) //function defination
{
    return x+y;
}
```

Will this work ?

```c
#include <stdio.h>

main()
{
    int a;
    a = fun(2, 3);

    printf("%d", a);
}

int fun(int x, int y) //function defination
{
    return x+y;
}
```

Error

C88
C89
C90
C95

← if fn returns integer then declaration is optional

C99
C11
C17

LATEST C
You have to declare before using

GATE 2005

Consider the following C program:

```
double foo (double);            /* Line 1 */
int main() {
    double da, db;
    //input da
    db = foo(da);
}
double foo (double a) {
    return a;
}
```

The above code compiled without any error or warning. If Line 1 is deleted, the above code will show:

A. no compile warning or error

B. some compiler-warnings not leading to unintended results

C. some compiler-warnings due to type-mismatch eventually leading to unintended results

D. compiler errors

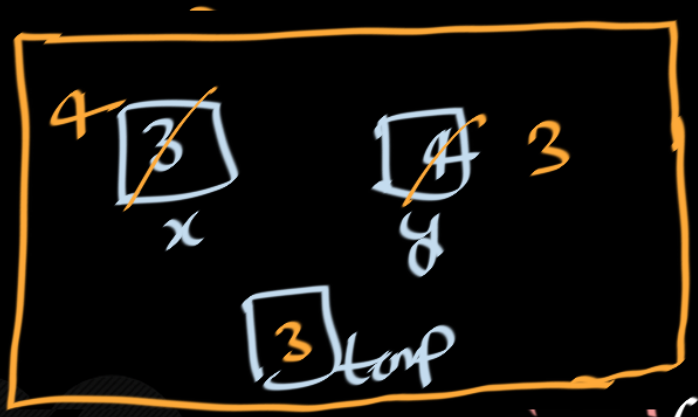
```c
void swap (int x, int y){

    int temp = x;
    x = y;
    y = temp;
}


int main(){
    int a =3;
    int b = 4;
    swap(a,b);
    printf("a = %d, b = %d", a,b)
}
```

Activation record

y in compiler

```
void swap (int x, int y){

    int temp = x;
    x = y;
    y = temp;
}

int main(){
    int a =3;
    int b = 4;
    swap(a,b);
    printf("a = %d, b = %d", a,b)
}
```

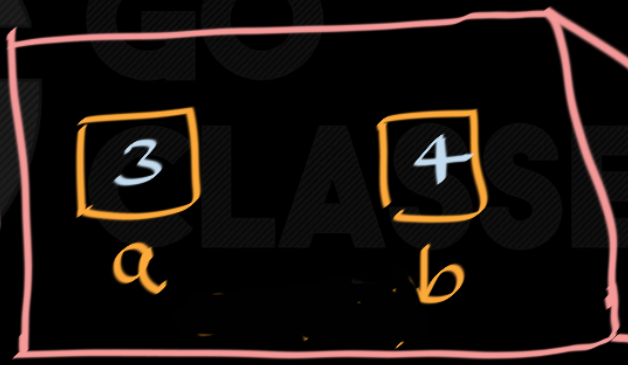
Pf(x), Pf(y)
4 3

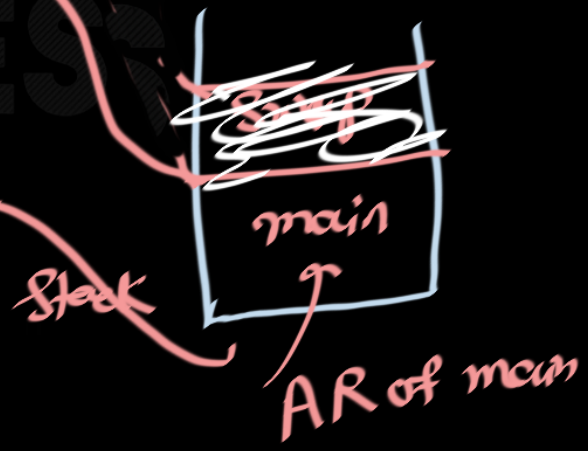


fake swap

```
void swap (int x, int y){

    int temp = x;
    x = y;
    y = temp;
}

int main(){
    int a =3;
    int b = 4;
    swap(a,b);
    printf("a = %d, b = %d", a,b)
}
```

Pf(x), Pf(y)
4 3



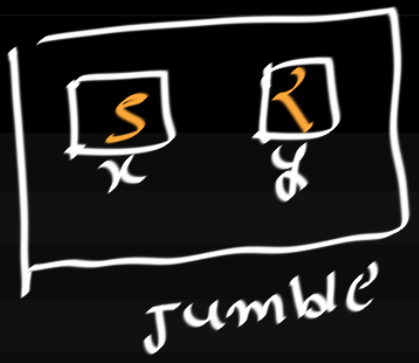


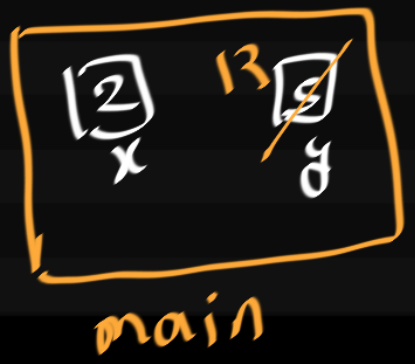
3 4

GATE CSE 2019

Consider the following C program :

```c
#include<stdio.h>
int jumble(int x, int y){
    x = 2*x+y;
    return x;
}
int main(){
    int x=2, y=5;
    y=jumble(y,x);
    x=jumble(y,x);
    printf("%d \n",x);
    return 0;
}
```



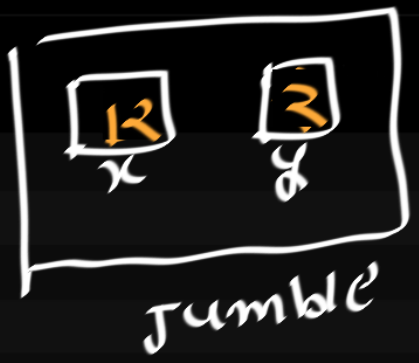


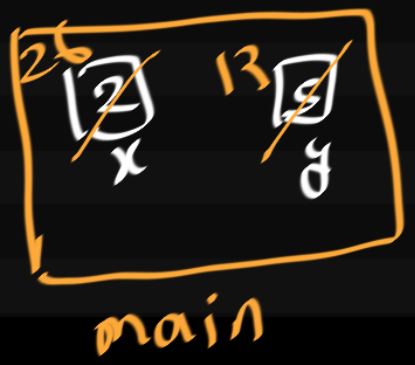
The value printed by the program is _____.

GATE CSE 2019

Consider the following C program :

```
#include<stdio.h>
int jumble(int x, int y){
    x = 2*x+y;
    return x;
}
int main(){
    int x=2, y=5;
    y=jumble(y,x);
    x=jumble(y,x);
    printf("%d \n",x);
    return 0;
}
```





The value printed by the program is _____.

From Writing to Running

CALL = Compiler Assembler Linker Loader


Boomelo
MAA KE PET SE MARGHAT TAK
HAI TERI KAHAANI PAG PAG PYARE
DANGAL DANGAL!
boomelomedia.com

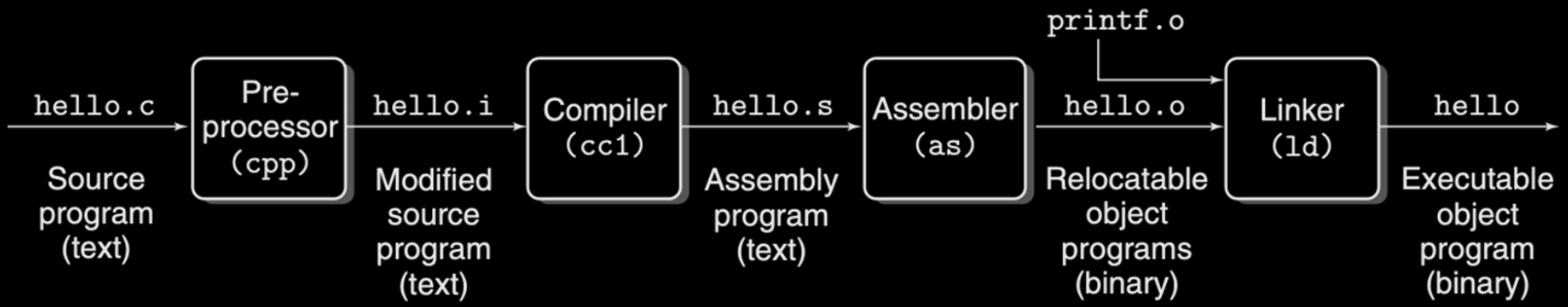
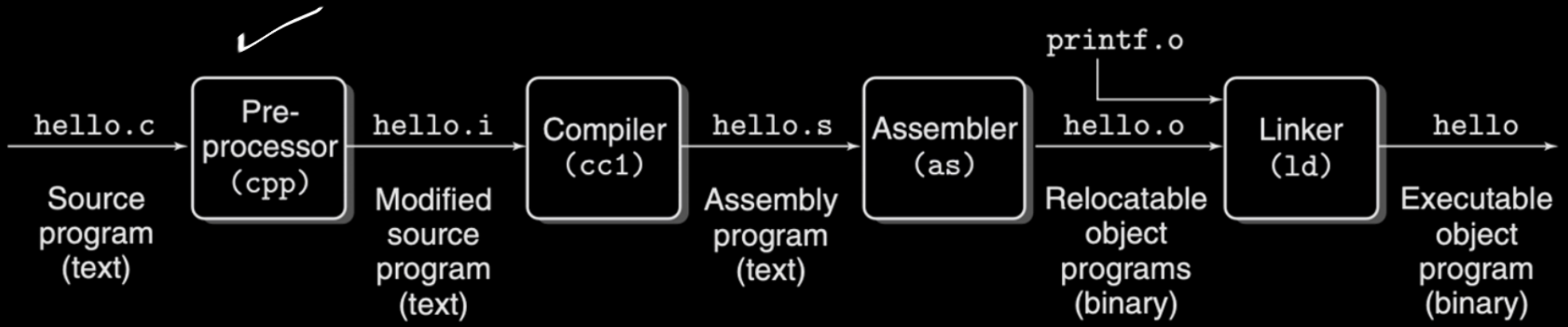
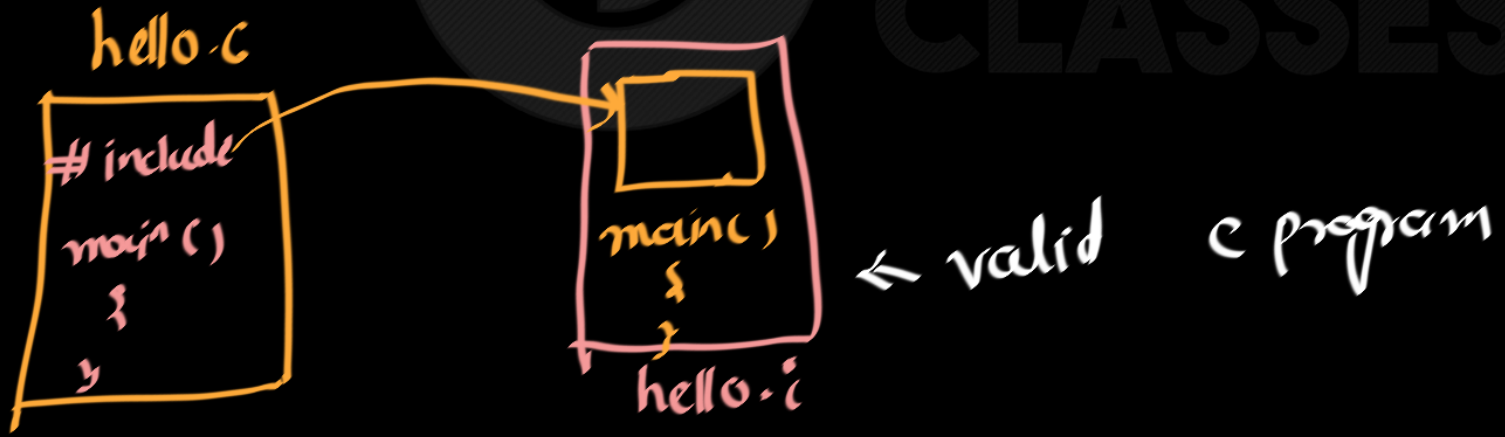
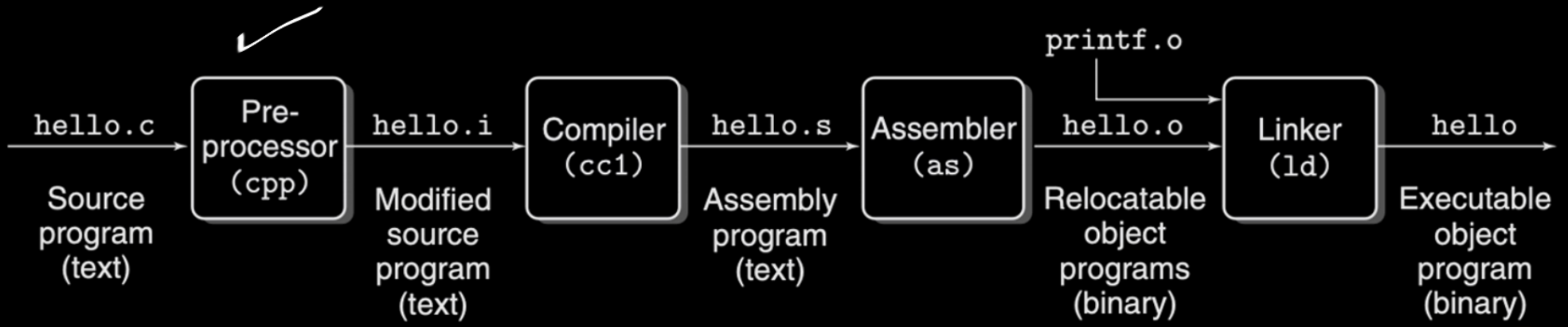


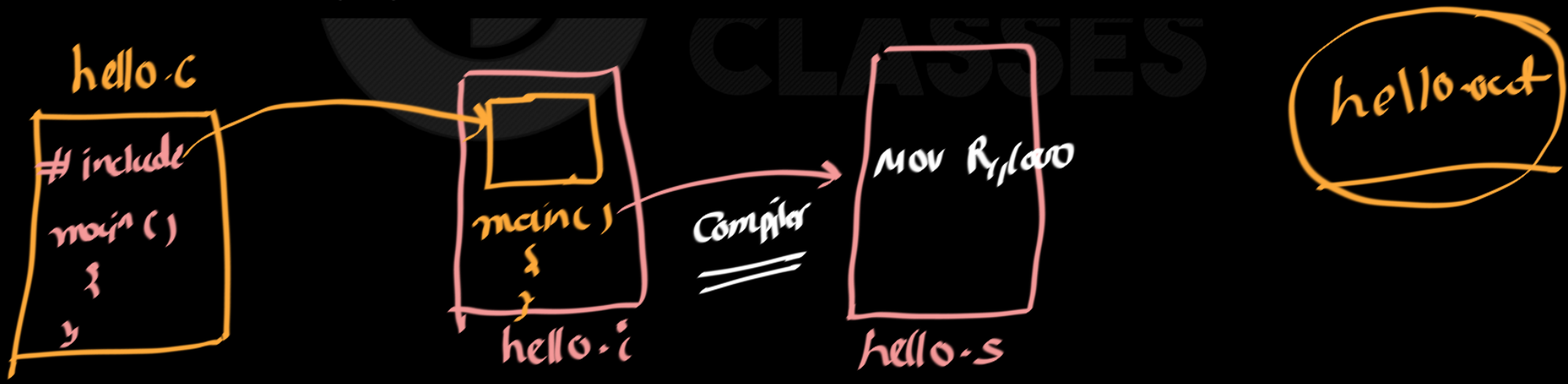
Figure 1.3 The compilation system.

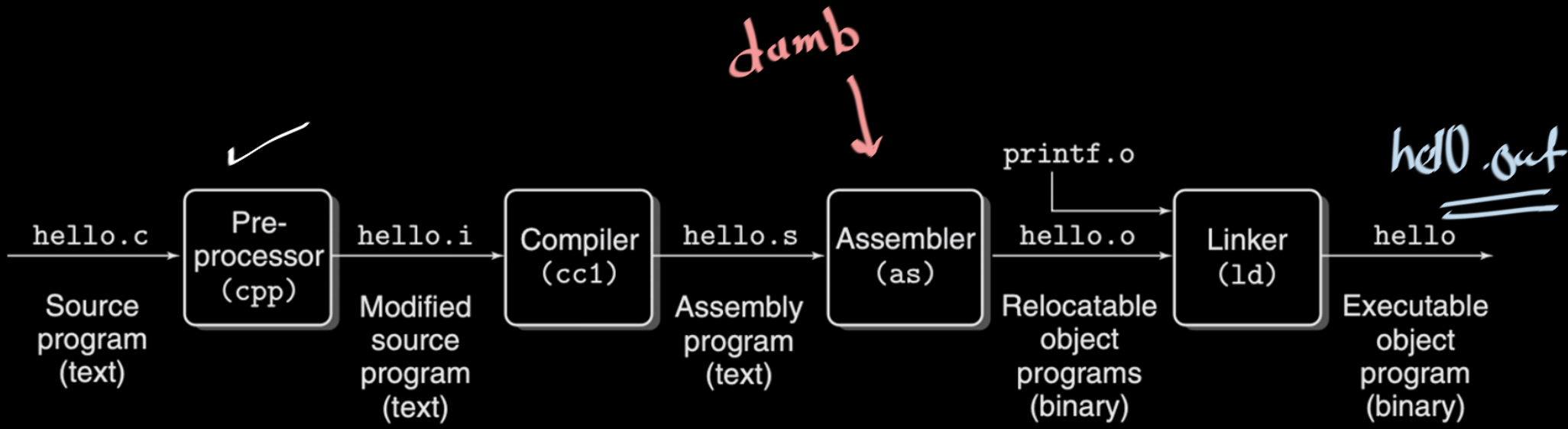
Image Source: - Computer Systems A Programmer's Perspective

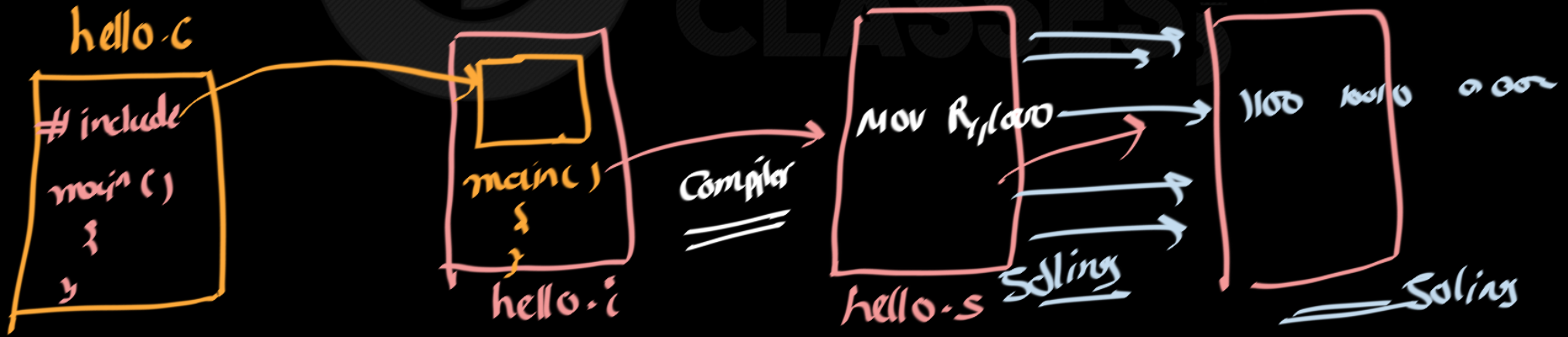

hello.c
Source program (text)
Pre-processor (cpp)
hello.i
Modified source program (text)
Compiler (cc1)
hello.s
Assembly program (text)
Assembler (as)
printf.o
hello.o
Relocatable object programs (binary)
Linker (ld)
hello
Executable object program (binary)


hello.c
#include
main()
{
}
main()
{
}
hello.i
← valid C program


hello.c
Source program (text)
Pre-processor (cpp)
hello.i
Modified source program (text)
Compiler (cc1)
hello.s
Assembly program (text)
Assembler (as)
printf.o
hello.o
Relocatable object programs (binary)
Linker (ld)
hello
Executable object program (binary)


hello.c
#include
main()
{
}
main()
{
}
hello.i
Compiler
mov R1,1000
hello.s
hello.out

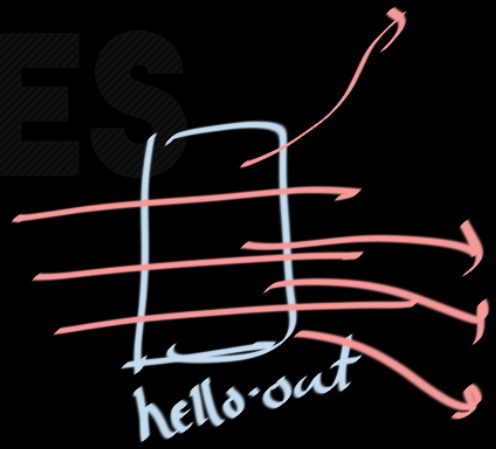

damb
printf.o
hello.out
hello.c
Pre-processor (cpp)
hello.i
Compiler (cc1)
hello.s
Assembler (as)
hello.o
Linker (ld)
hello
Source program (text)
Modified source program (text)
Assembly program (text)
Relocatable object programs (binary)
Executable object program (binary)


hello.c
#include
main()
{
}
main()
{
}
hello.i
Compiler
MOV R1,1000
hello.s
Solving
Solving

GO CLASSES

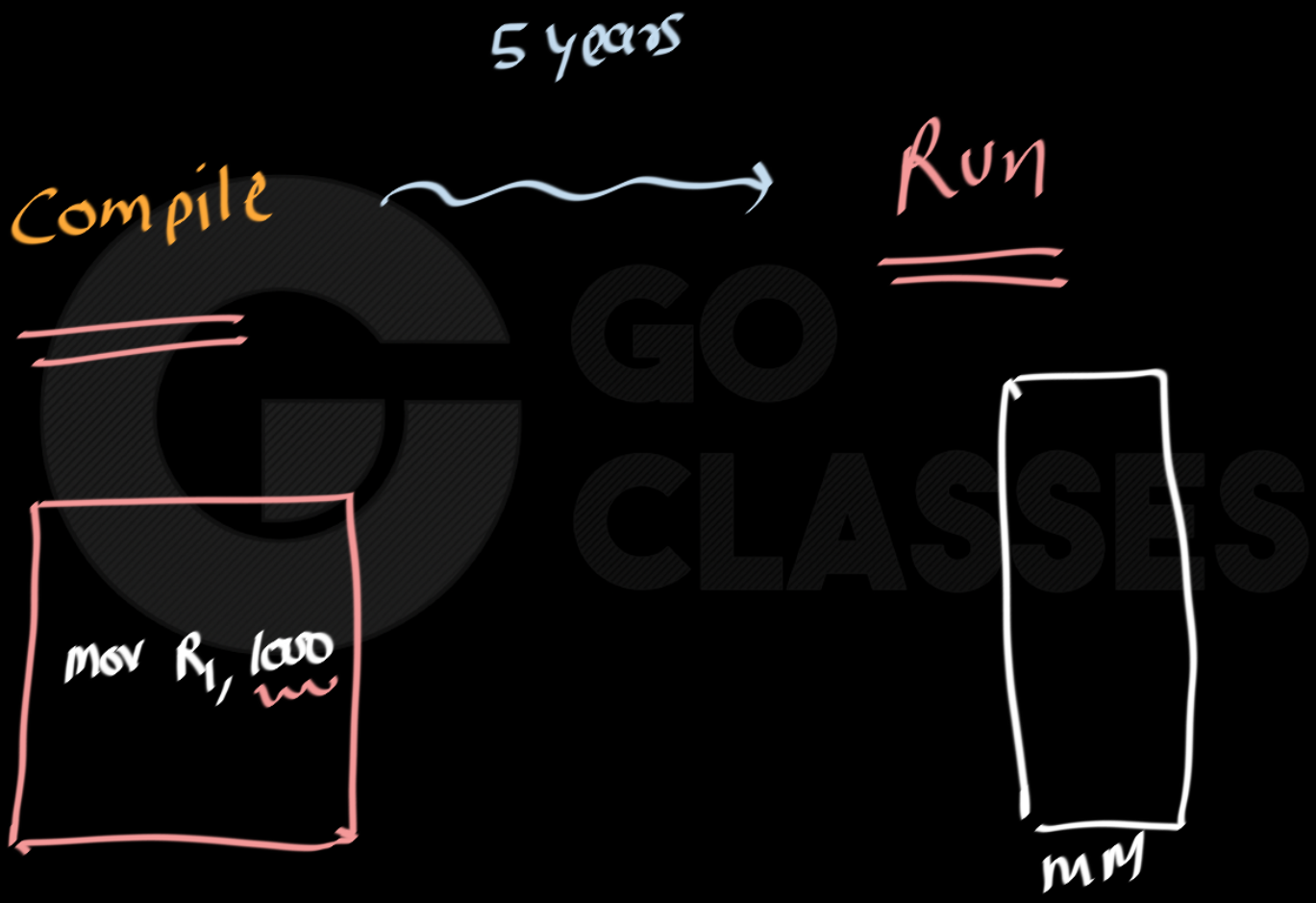
logical addresses

Mov R_1, 1000



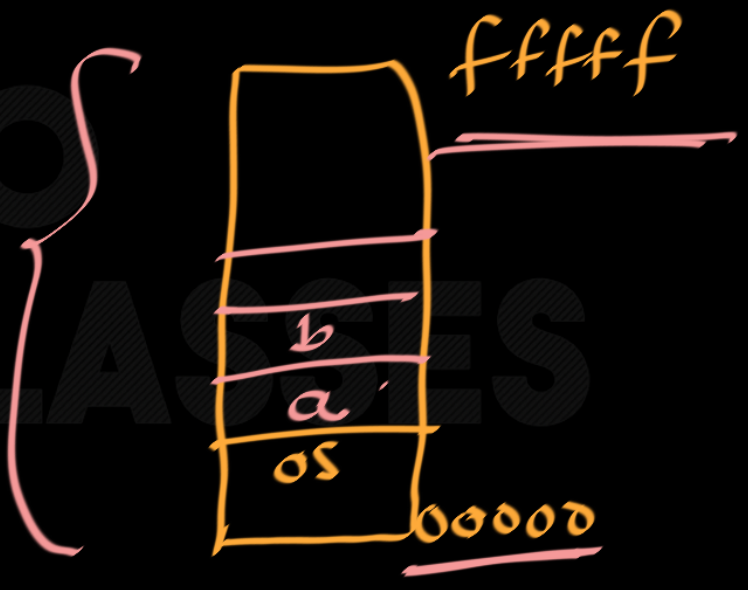
int a = 5;

Load M[5] s


5 years
Compile
Run
mov R1, 1000
MM

Logical memory

int a=5

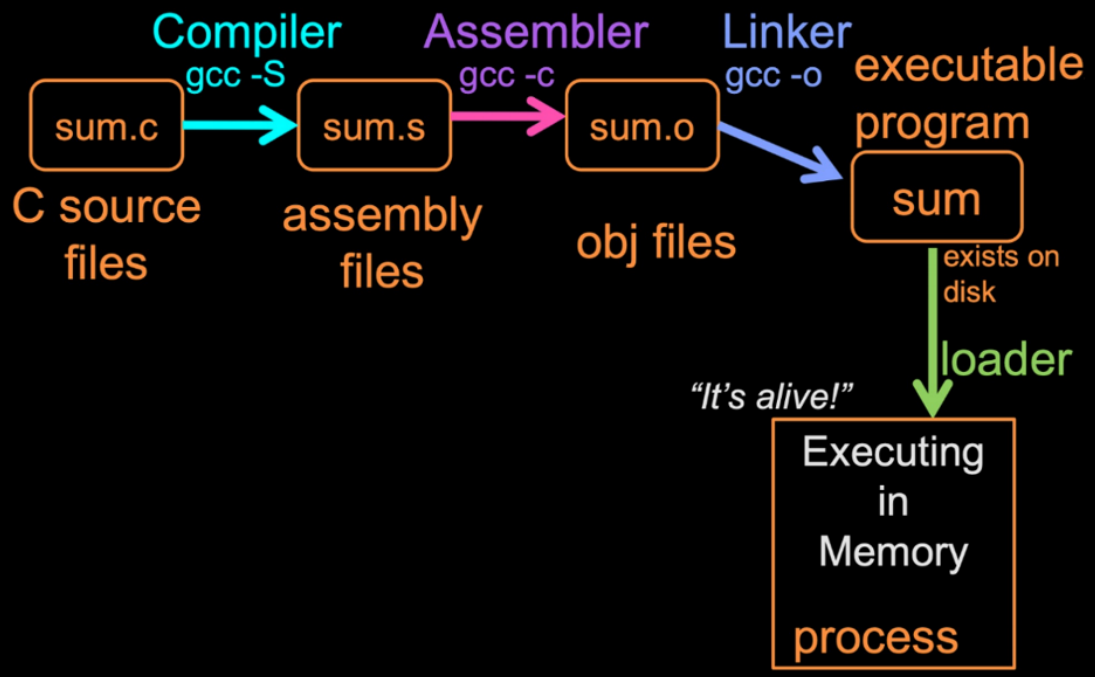


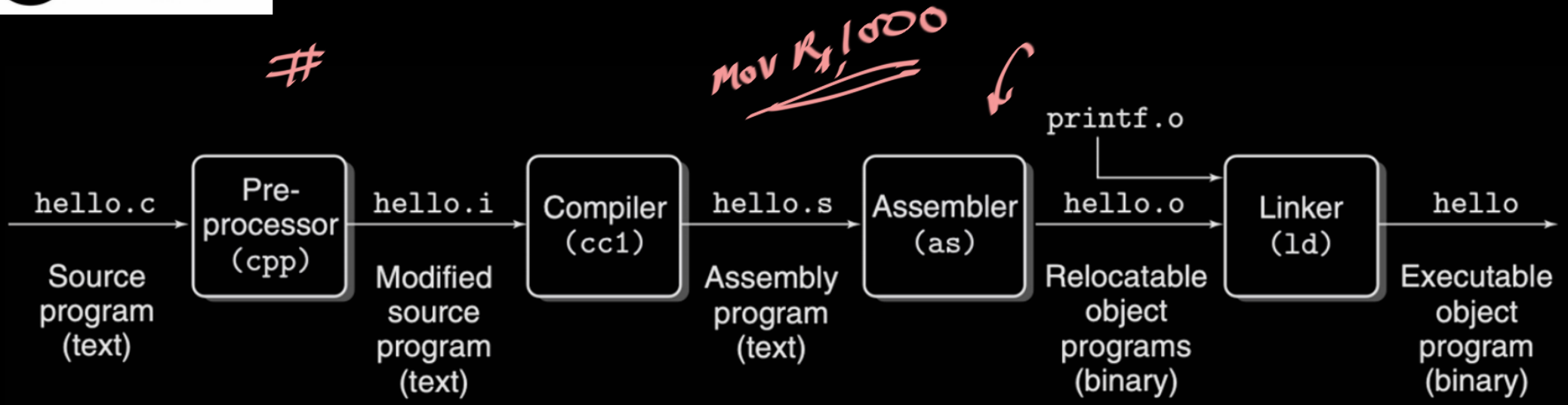
GO CLASSES

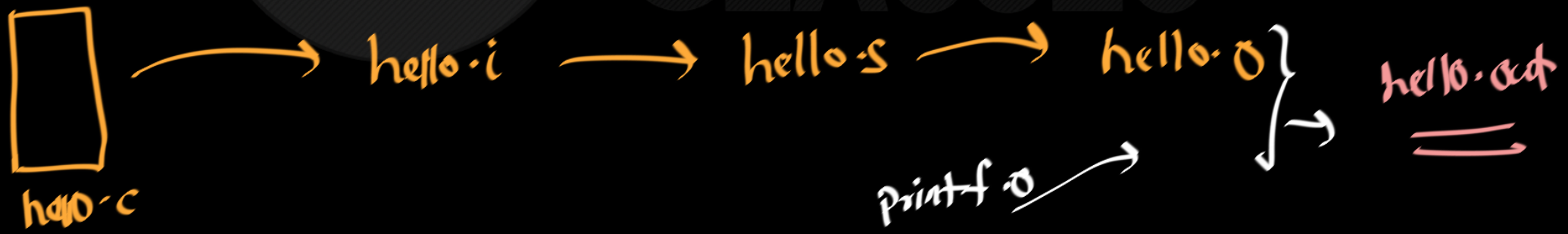
Problem : How does compiler get to know the (physical) address where we will load the prog. (after 5 years)

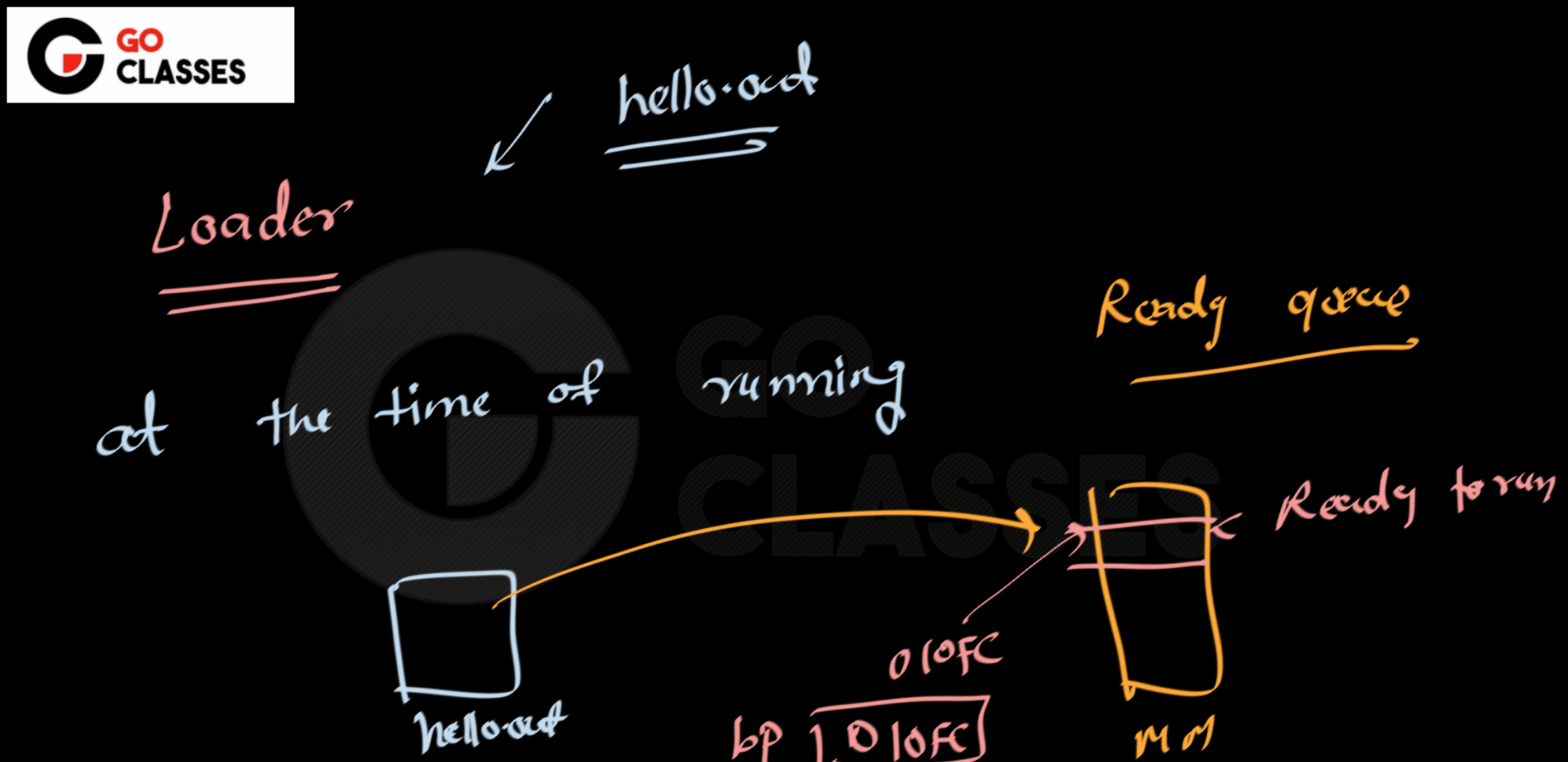
Soln : Compiler don't even care. Compiler assumes all the (logical) memory is available to just one process. and generates logical addresses. (Relocatable)


Most crucial
(compiler)
hello.c
hello.i
hello.s
Assembler
(printf.o)
hello.o
hello.out


Compiler
gcc -S
Assembler
gcc -c
Linker
gcc -o
executable program
sum.c
sum.s
sum.o
sum
C source files
assembly files
obj files
exists on disk
loader
"It's alive!"
Executing in Memory
process


hello.c
Pre-processor (cpp)
hello.i
Compiler (cc1)
hello.s
Assembler (as)
hello.o
printf.o
Linker (ld)
hello
Source program (text)
Modified source program (text)
Assembly program (text)
Relocatable object programs (binary)
Executable object program (binary)


hello.c
hello.i
hello.s
hello.o
printf.o
hello.exe


hello.out
Loader
at the time of running
Ready queue
Ready to run
hello.out
0 10FC
bp 1 0 10FC
M 01

Q1.2 (1.25 points) True or False: The assembler is the step with the highest computational complexity among CALL.

○ True ○ False

https://inst.eecs.berkeley.edu/~cs61c/fa21/pdfs/exams/fa21-midterm-sols.pdf

Q1.2 (1.25 points) True or False: The assembler is the step with the highest computational complexity among CALL.

○ True ● False

Solution: False. The compiler is more complex than the assembler.

Grading: 1.25 points for False.

https://inst.eecs.berkeley.edu/~cs61c/fa21/pdfs/exams/fa21-midterm-sols.pdf

Q1.3 (1.25 points) True or False: The assembler produces an executable.

○ True ○ False

a.out

Q1.3 (1.25 points) True or False: The assembler produces an executable.

○ True ● False

Solution: False. The assembler creates an object file. The linker creates an executable.

Grading: 1.25 points for False.

Q1.4 (1.25 points) True or False: In the loader, the program is placed in memory in preparation of running the code.

○ True ○ False

Ready

Q1.4 (1.25 points) True or False: In the loader, the program is placed in memory in preparation of running the code.

True ○ False

Solution: True. Executable files (the program instructions after passing through the compiler, assembler, and linker) are stored on the disk, and the loader will place the executable in memory before running it.

Grading: 1.25 points for True.

For each of the following questions select which stage of CALL (Compiler, Assembler, Linker, Loader) the action occurs in:

5. Static data is placed in memory

   Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

6. External labels are resolved

   Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

7. Operator precedence is resolved

   Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

https://inst.eecs.berkeley.edu/~cs61c/sp21/pdfs/exams/Su19_MT2_Solutions.pdf

For each of the following questions select which stage of CALL (Compiler, Assembler, Linker, Loader) the action occurs in:

5. Static data is placed in memory

   Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

6. External labels are resolved

   Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

7. Operator precedence is resolved

   Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

https://inst.eecs.berkeley.edu/~cs61c/sp21/pdfs/exams/Su19_MT2_Solutions.pdf

For each of the following questions select which stage of CALL (Compiler, Assembler, Linker, Loader) the action occurs in:

5. Static data is placed in memory

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

6. External labels are resolved

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

7. Operator precedence is resolved

Ⓐ Compiler Ⓑ Assembler Ⓒ Linker Ⓓ Loader

https://inst.eecs.berkeley.edu/~cs61c/sp21/pdfs/exams/Su19_MT2_Solutions.pdf

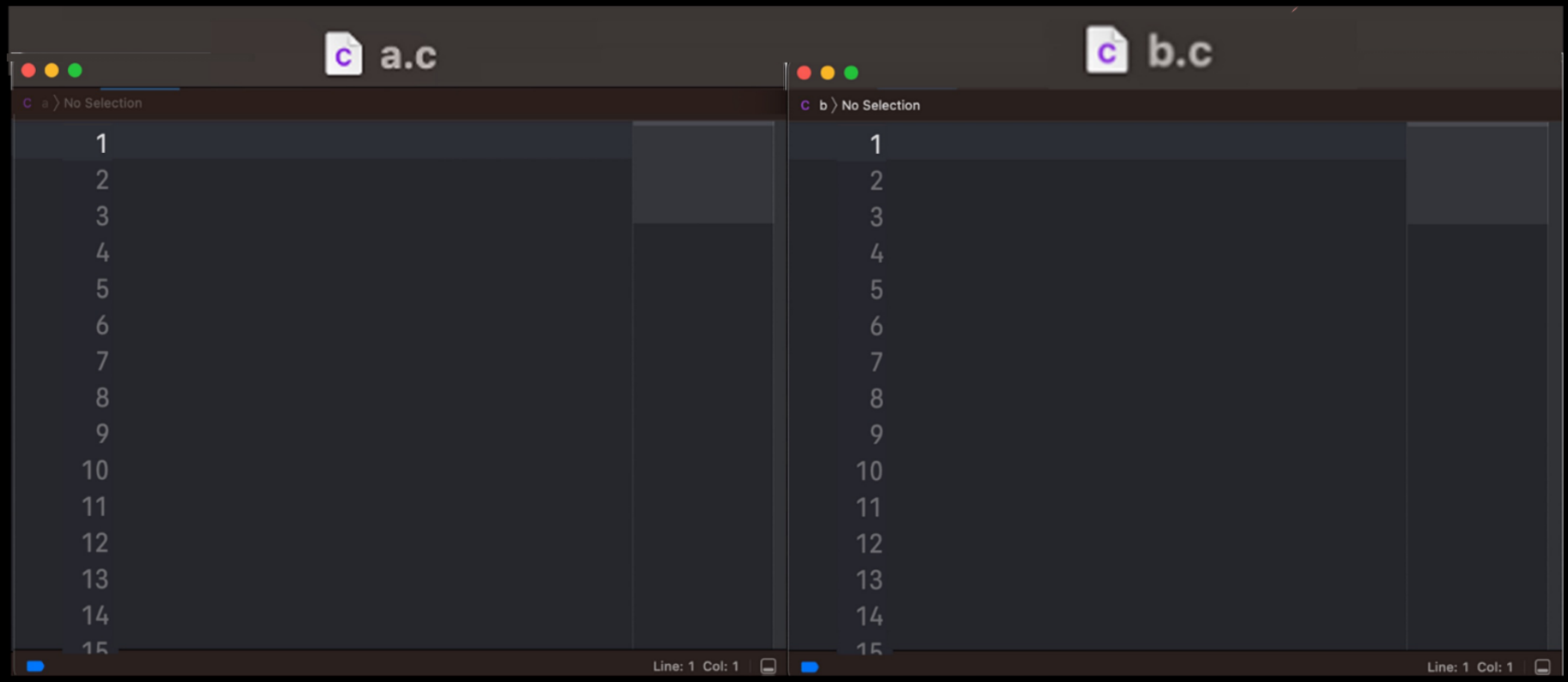
1. (5 points) Which of the following creates a binary from multiple object files?

   ◯ compiler ◯ assembler ◯ linker ◯ interpreter

https://courses.cs.duke.edu/fall21/compsci310/exams/midterm.pdf

Optional

Linking 2 files

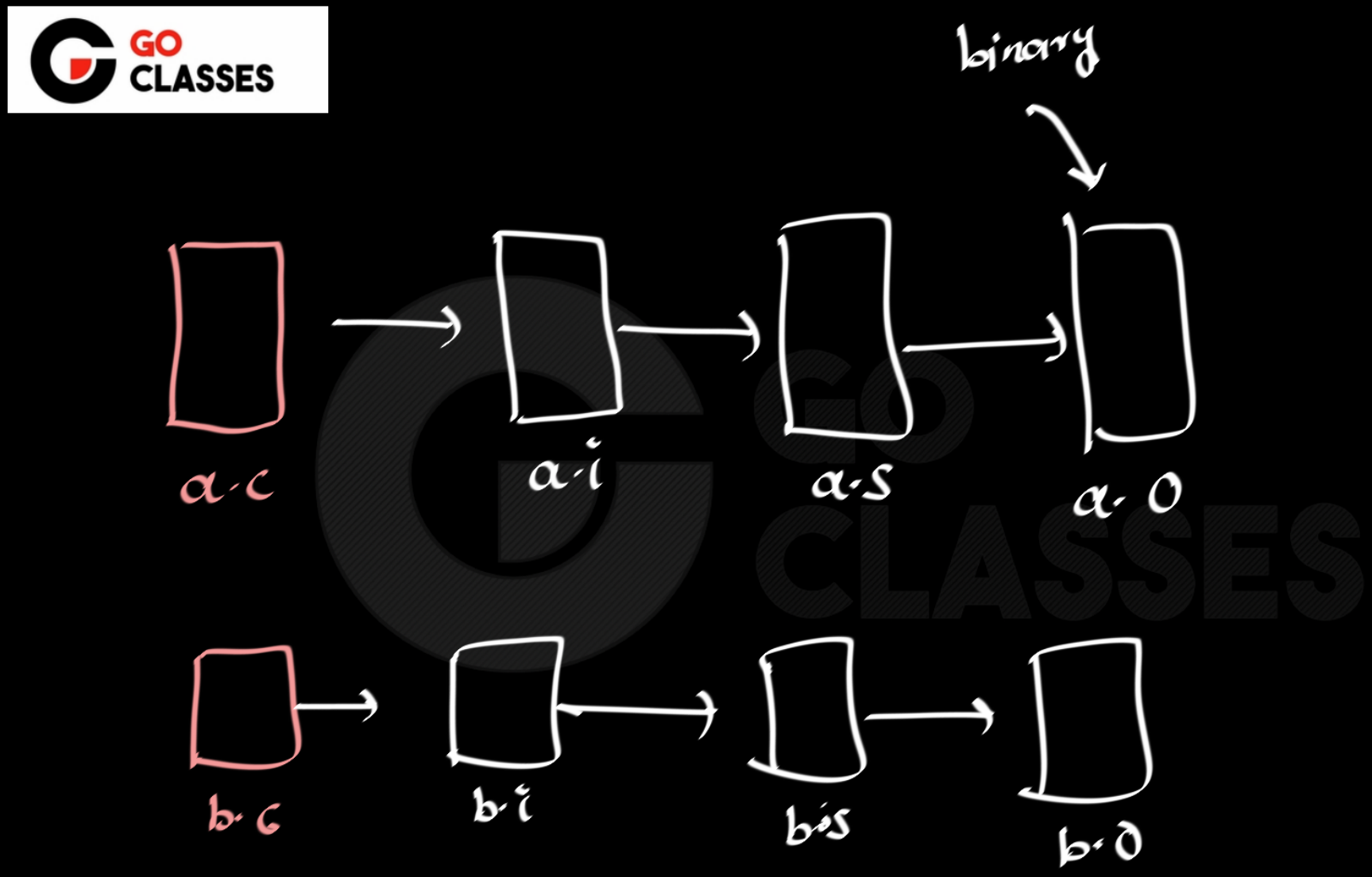

a.c
No Selection
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Line: 1 Col: 1
b.c
No Selection
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Line: 1 Col: 1

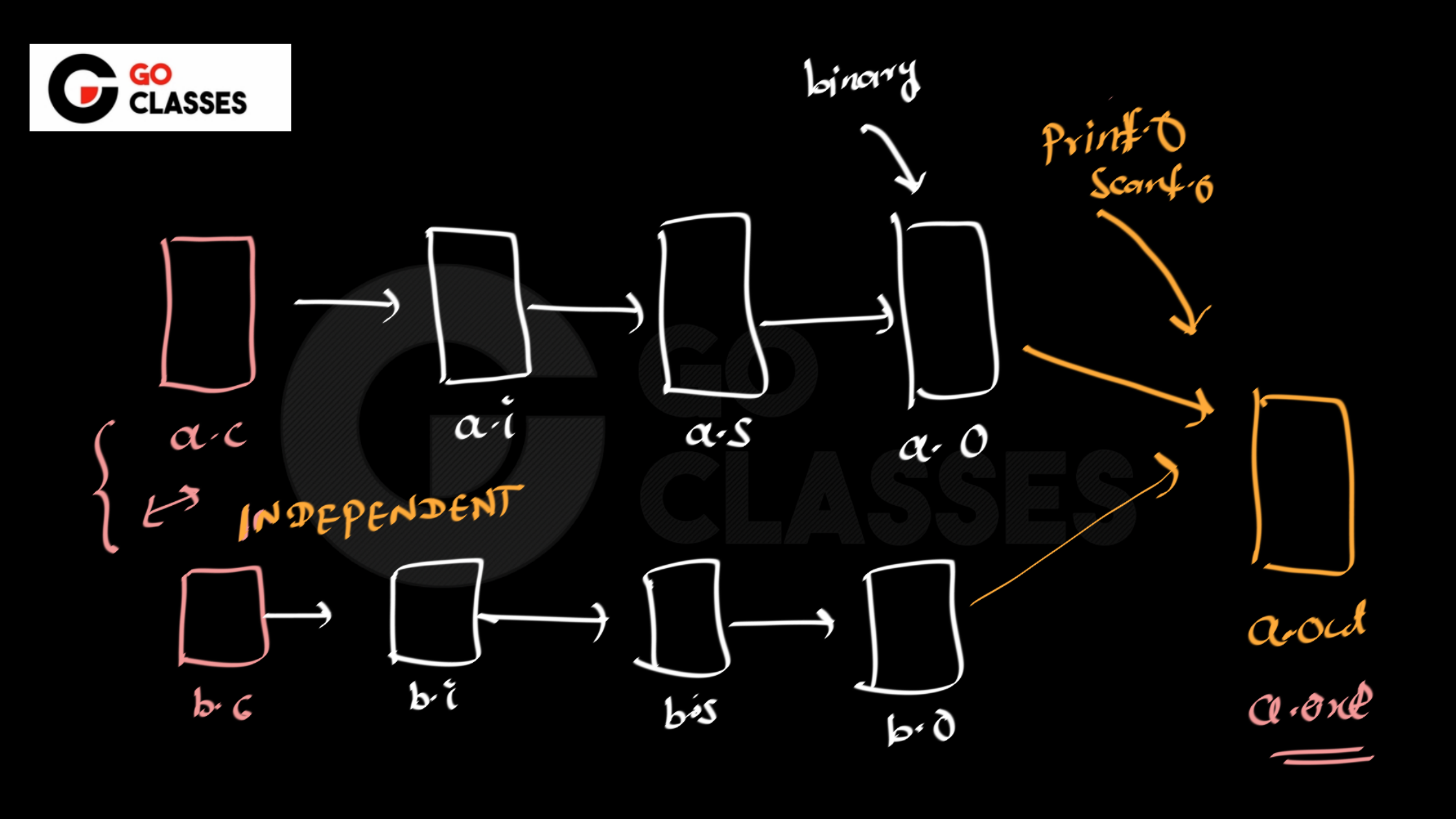
a.c

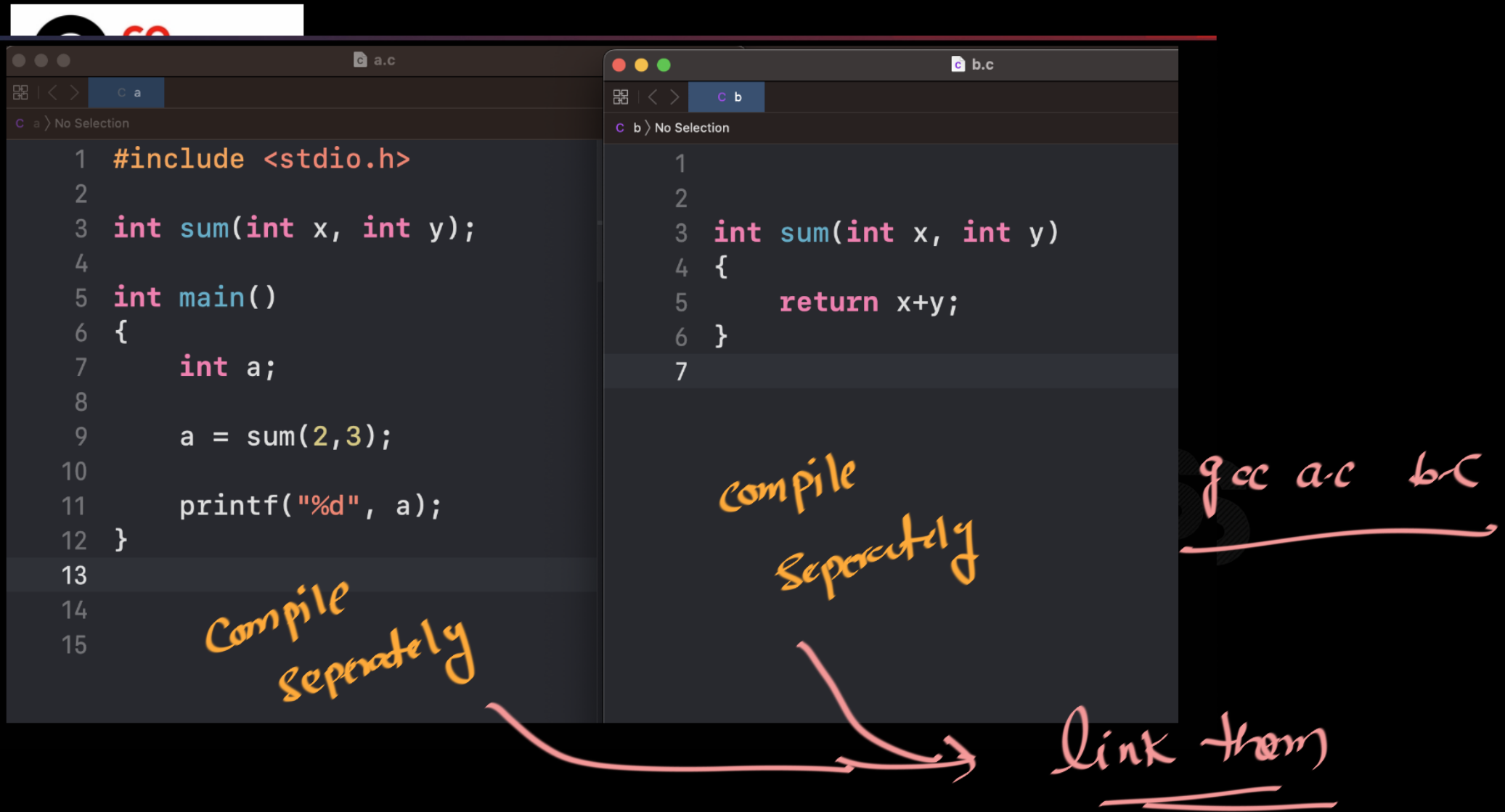
```c
#include <stdio.h>

int sum(int x, int y);

int main()
{
    int a;

    a = sum(2,3);

    printf("%d", a);
}
```

b.c

```c
int sum(int x, int y)
{
    return x+y;
}
```


binary
a.c
a.i
a.s
a.o
b.c
b.i
b.s
b.o


GO CLASSES
binary
Printf()
Scanf()
a.c
a.i
a.s
a.o
INDEPENDENT
b.c
b.i
b.s
b.o
a.out
a.exe


a.c
a
No Selection
#include <stdio.h>
int sum(int x, int y);
int main()
{
int a;
a = sum(2,3);
printf("%d", a);
}
b.c
b
No Selection
int sum(int x, int y)
{
return x+y;
}
Compile Seperately
Compile Seperately
gcc a.c b.c
link them

GO CLASSES

```c
int printf(char *s);

int main()
{

    printf("hello \n");
}
```

explicit declaration of printf

include <stdio.h>

↳ it has only prototype of printf.

GO CLASSES

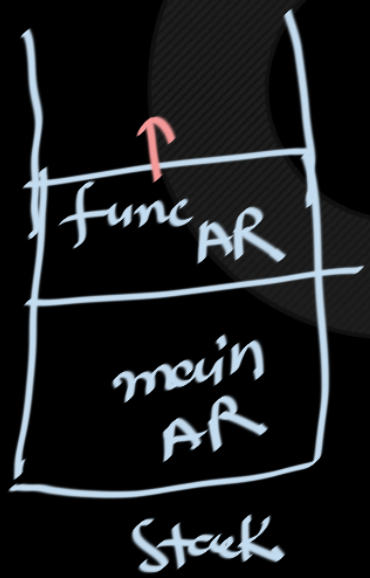
MOV R1, 1000

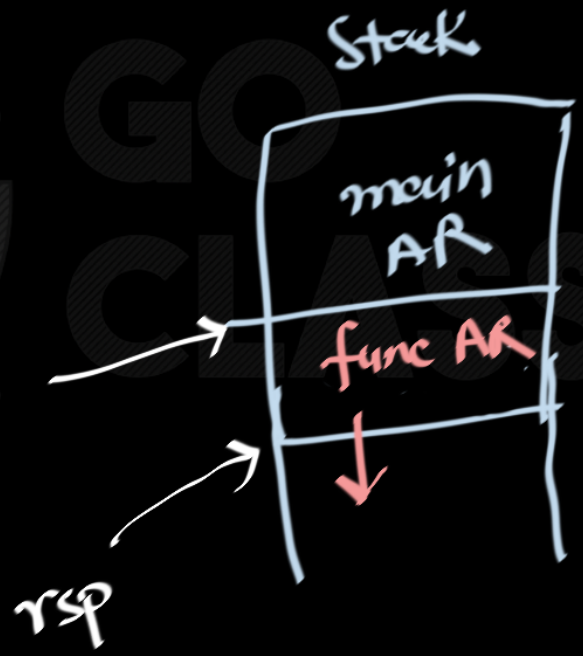
Optional

Example of C code to Assembly

GO CLASSES

Activation record





BP rbp

Base pointer

SP rsp

Stack Pointer

```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```

```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

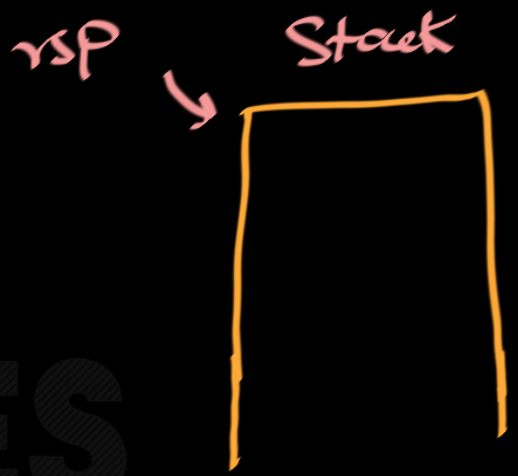
```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```

entry

exit

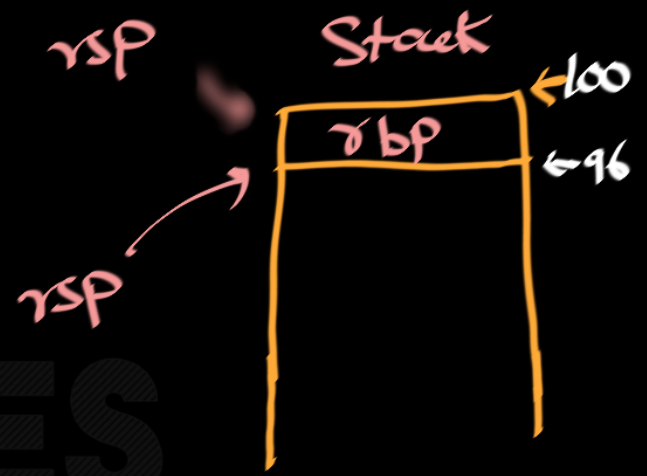
entry

exit



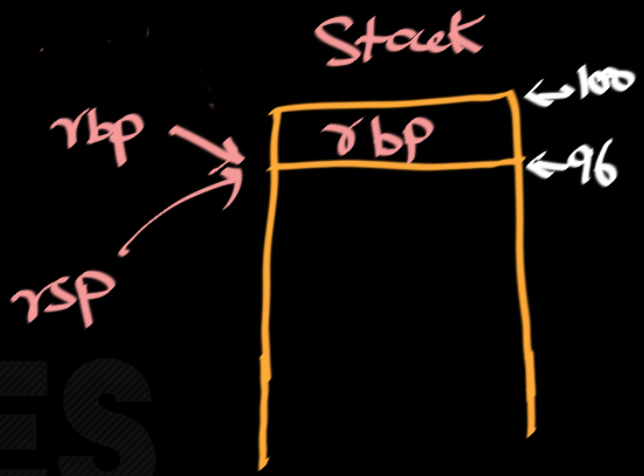
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



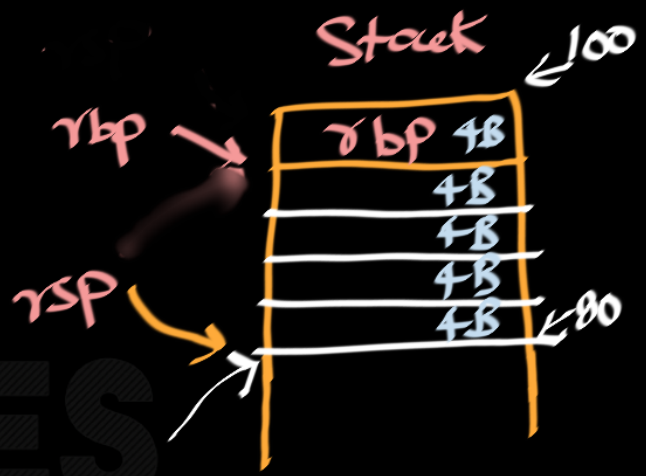
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

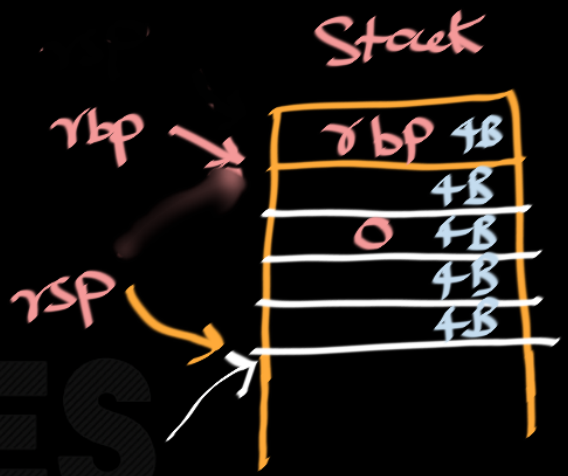
```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



rsp = 80 , rbp = 96

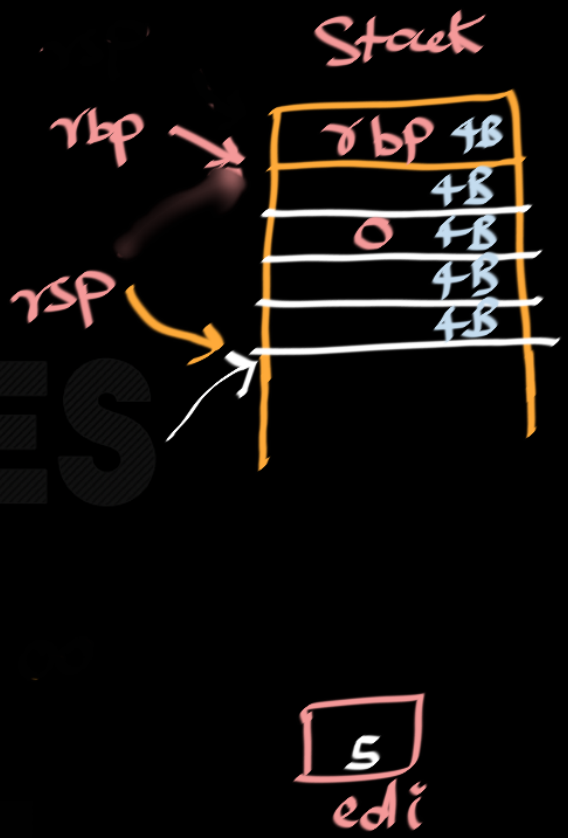
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



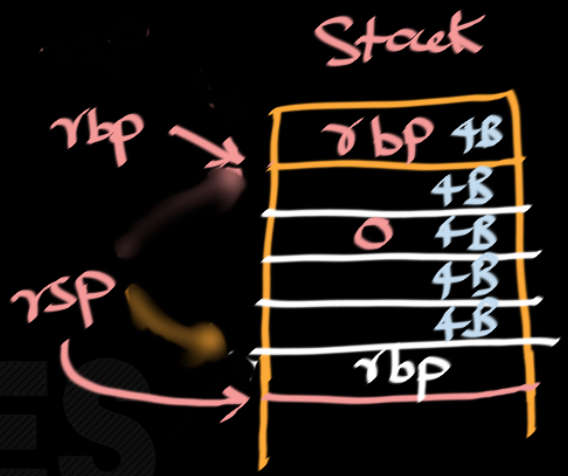
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

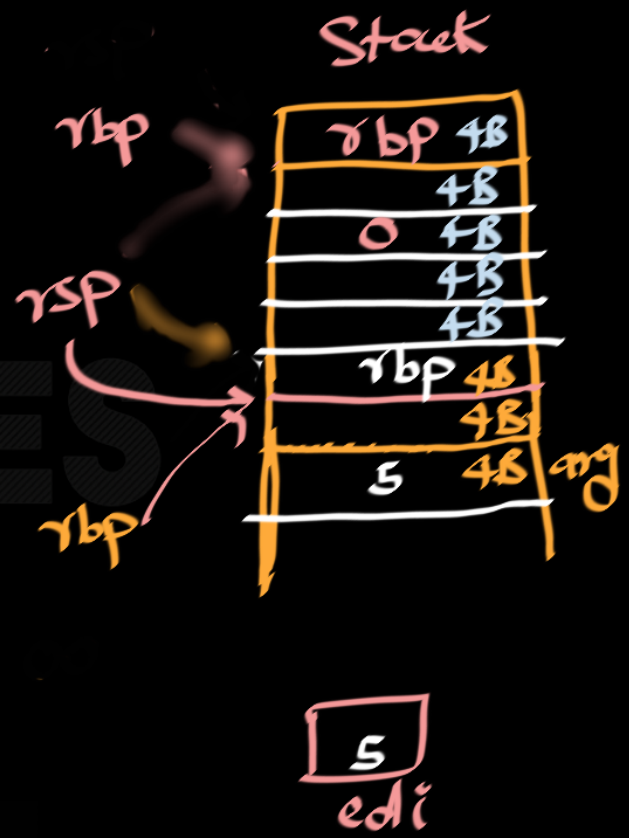
```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



rsp = 80 , rbp = 96

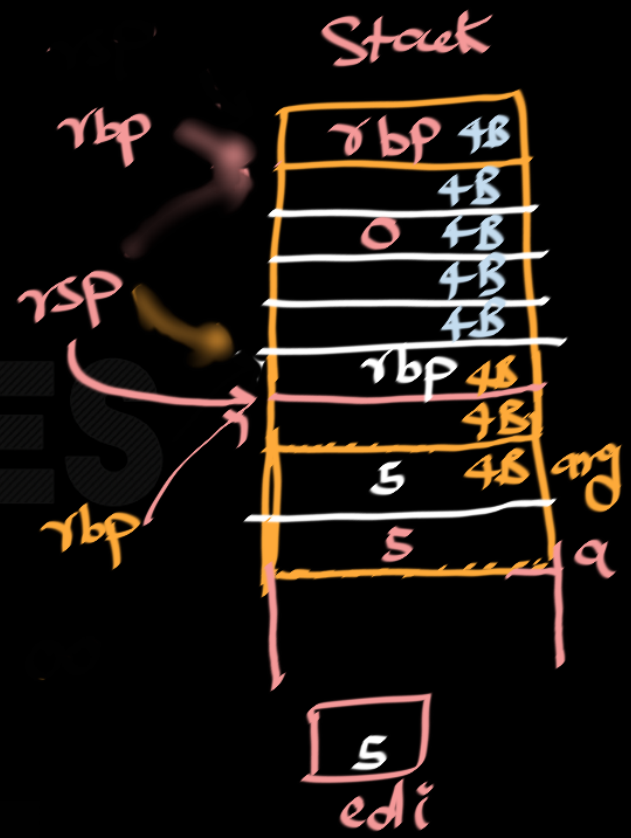
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



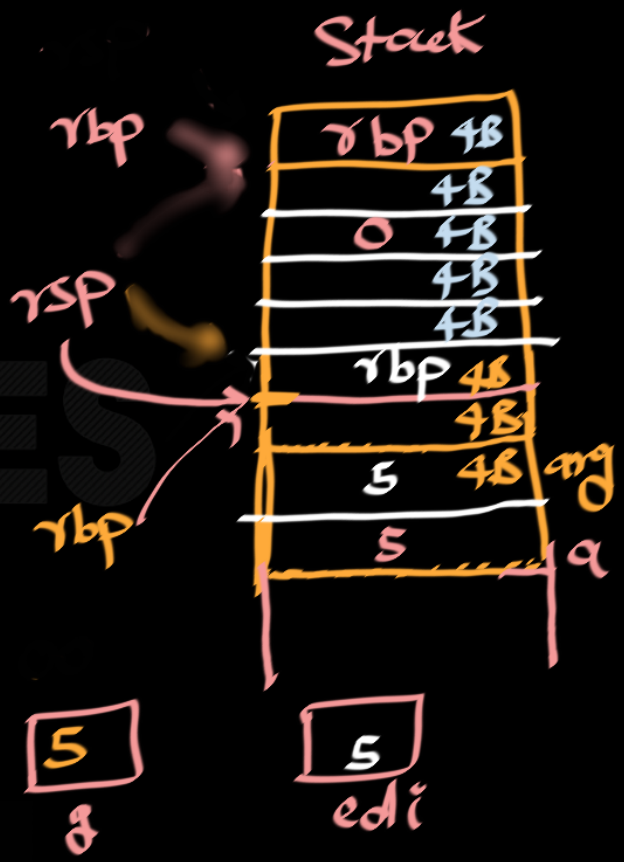
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



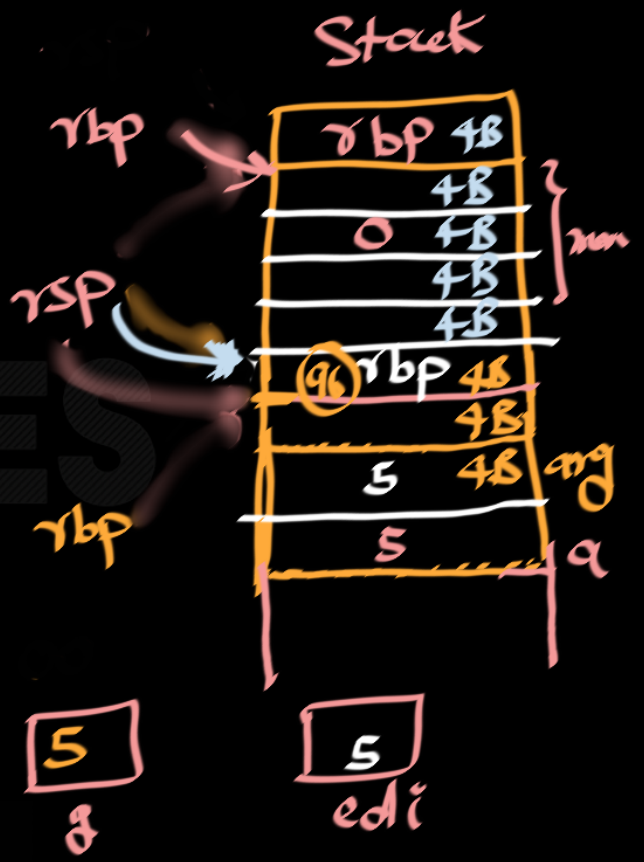
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



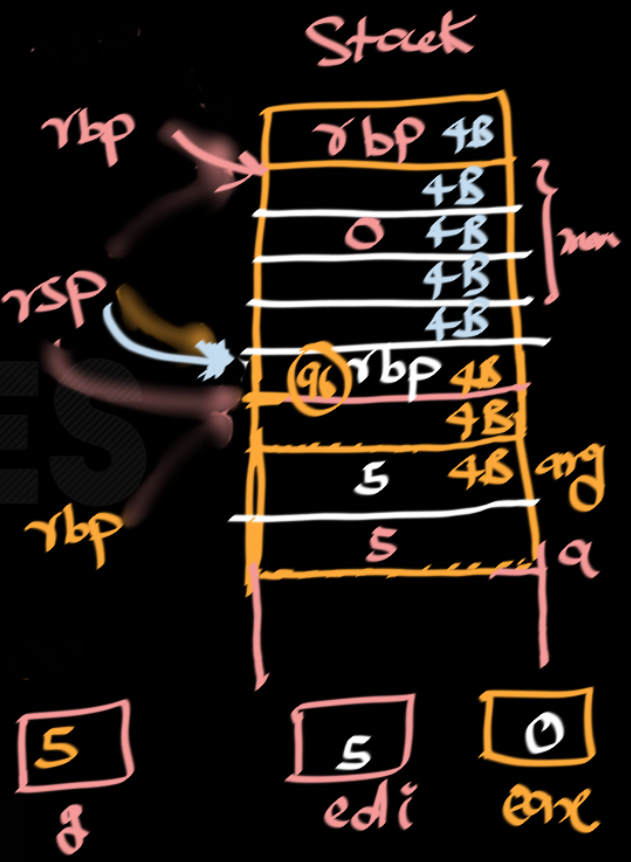
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



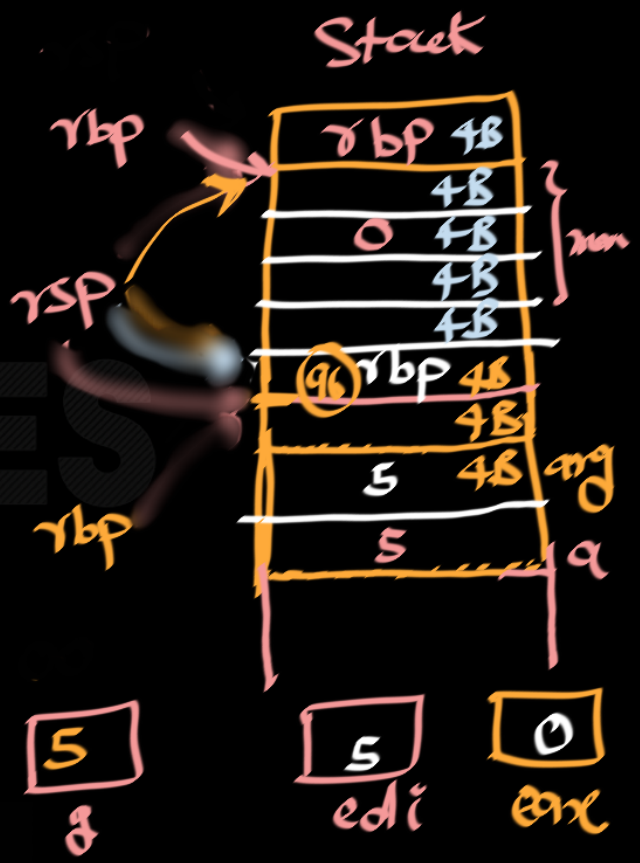
```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



```c
int g;

void func(int arg)
{
    int a = 5;
    g = 5;
}

int main()
{
    func(5);
    return 0;
}
```

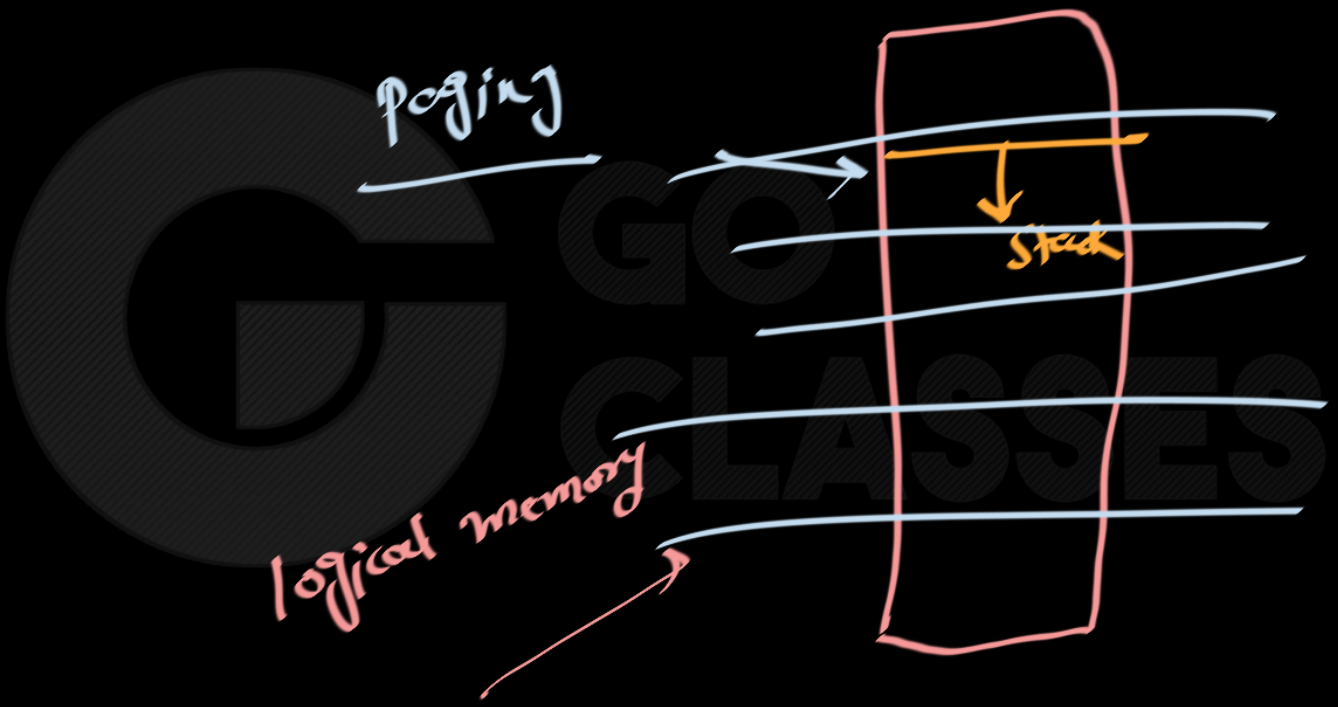
```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 5
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 5
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```



```
int g;

void func(int arg)
{
    int a = 4;
    g = 5;
}

int main()
{
    func(3);
    return 0;
}
```

```
1   func:                                   # @func
2           push    rbp
3           mov     rbp, rsp
4           mov     dword ptr [rbp - 4], edi
5           mov     dword ptr [rbp - 8], 4
6           mov     dword ptr [g], 5
7           pop     rbp
8           ret
9   main:                                   # @main
10          push    rbp
11          mov     rbp, rsp
12          sub     rsp, 16
13          mov     dword ptr [rbp - 4], 0
14          mov     edi, 3
15          call    func
16          xor     eax, eax
17          add     rsp, 16
18          pop     rbp
19          ret
```

relocatable code


Paging
Stack
logical memory

GO CLASSES

- Compiler generates logical addresses

and Based on the place wherever we are loading prog, OS translate logical address ⟶ physical address.

OS ⇒ (Base + Bound, paging, Segmentation) ← in OS