CS & IT ENGINEERING

Programming in C

Functions and Storage Classes

Lec- 03

By- Pankaj Sharma sir

TOPICS TO BE COVERED

Storage Classes and Recursion

①

```
int j = 10;
static int i = j ; ✗
```

Static variable must be initialized by literals.

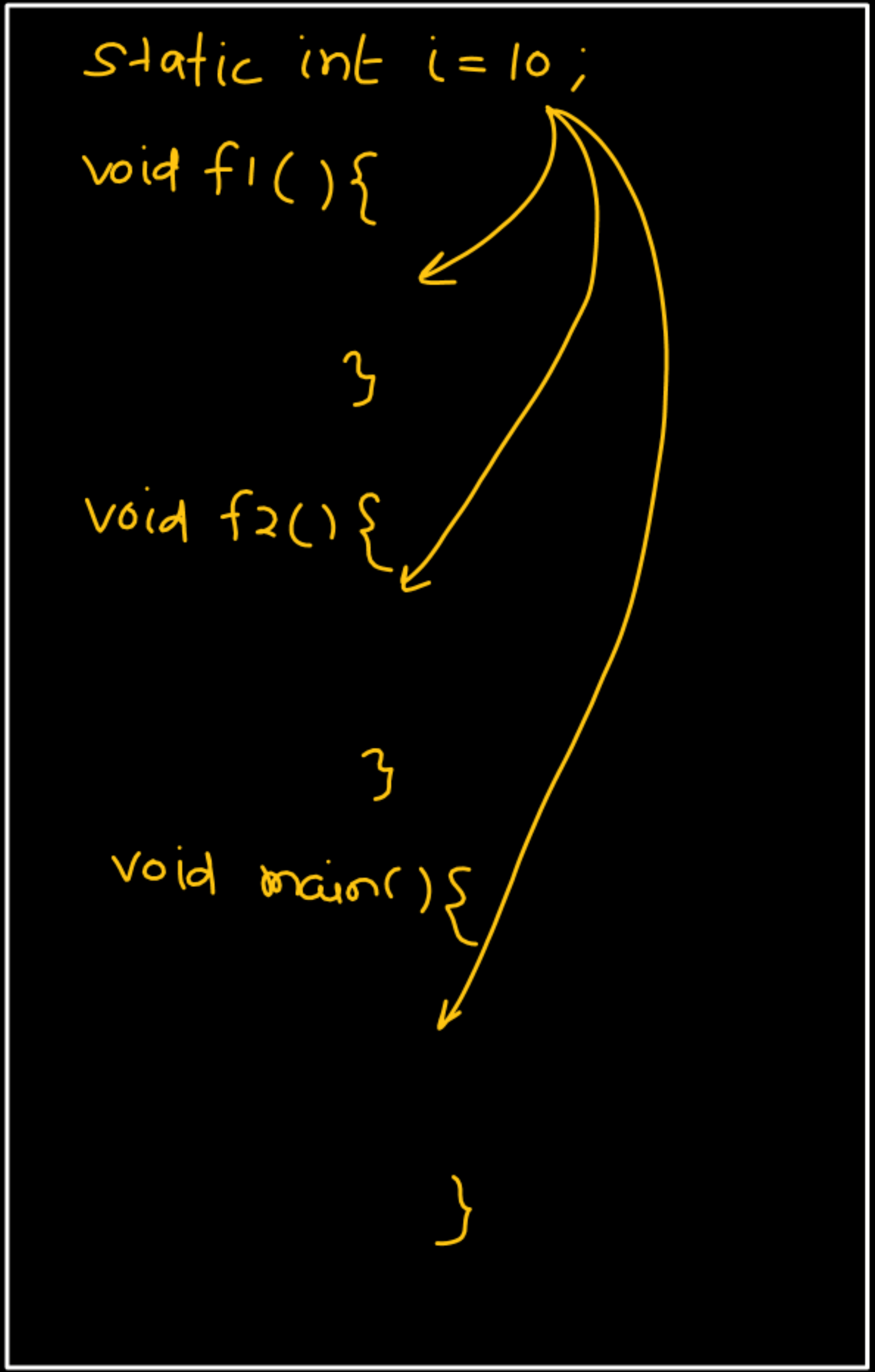
②

Valid

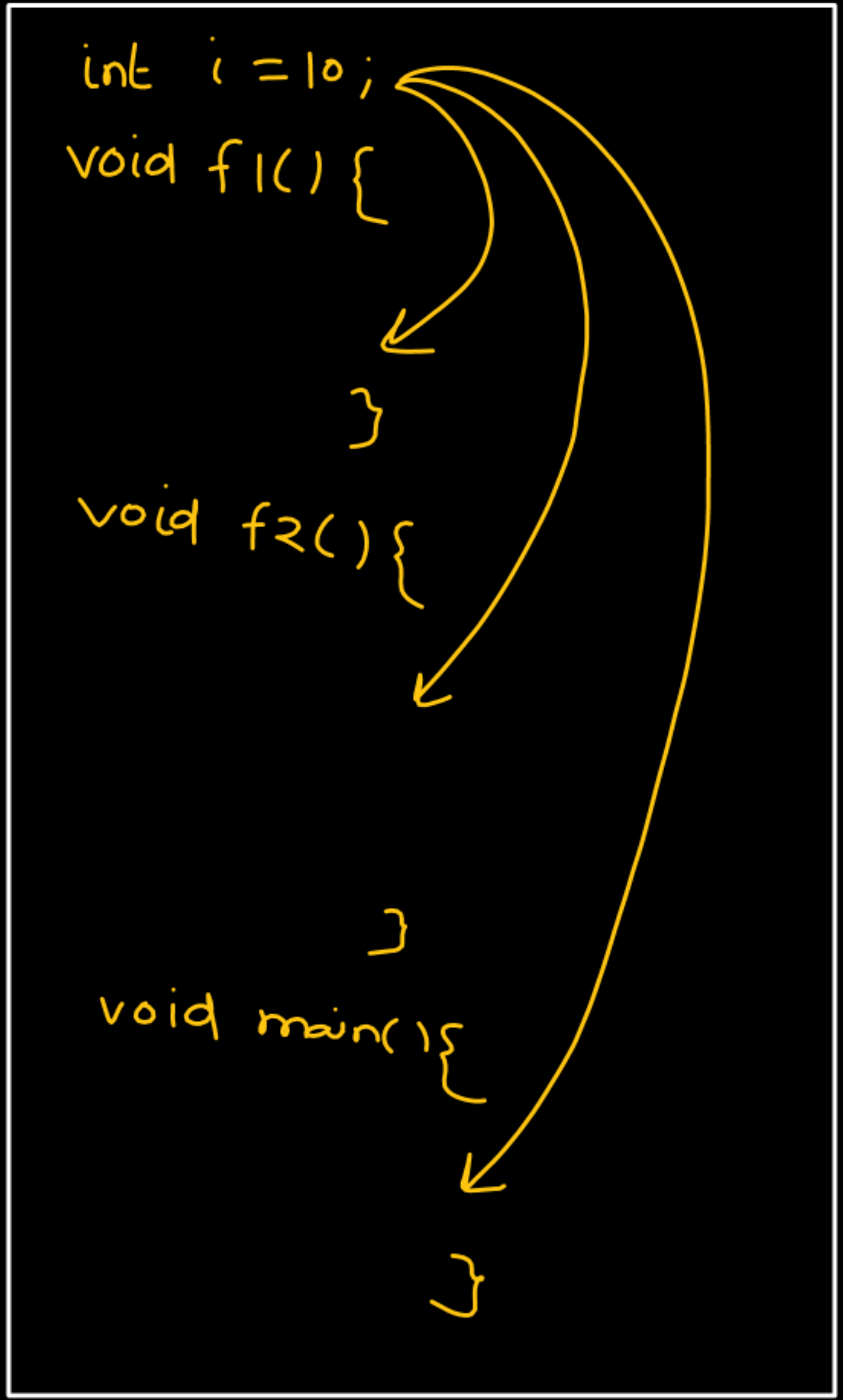
```
void main(){
    int i ;
    int j ;
}
```

Invalid

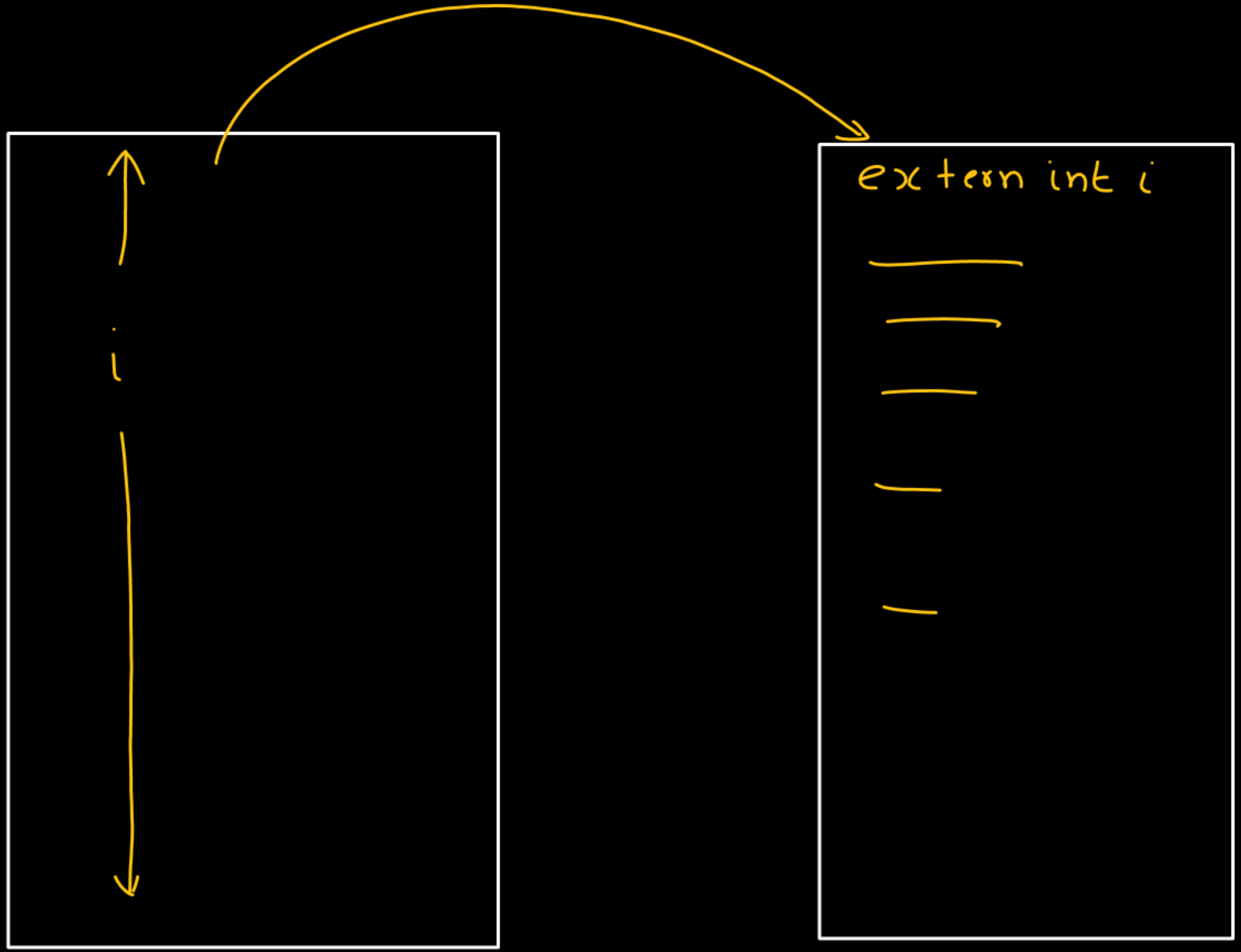
```
void main(){
    int i ;
    int i ;
}
```

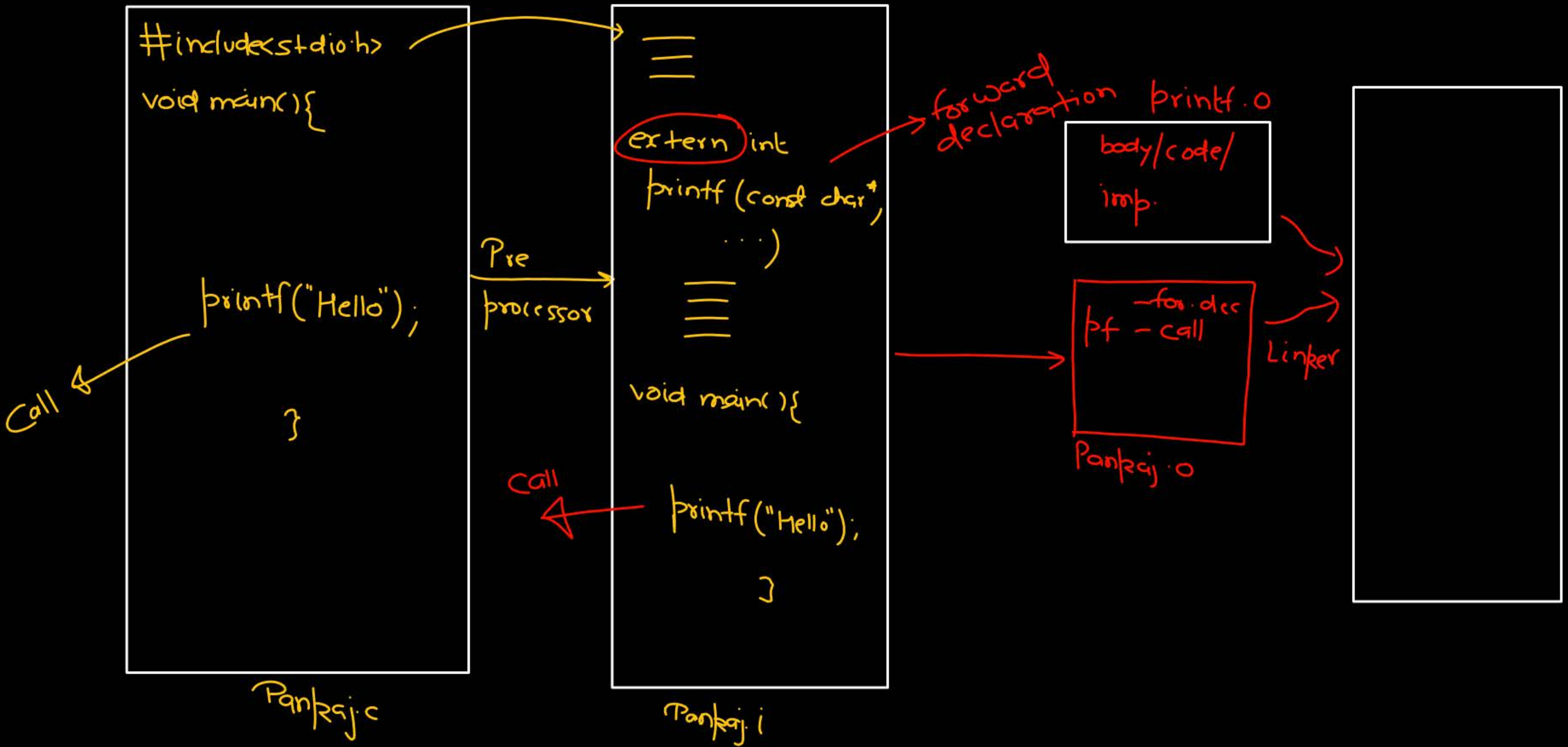


```
static int i=10;
void f1(){

}
void f2(){

}
void main(){

}
```



```
int i=10;
void f1(){

}
void f2(){

}
void main(){

}
```


i
extern int i


#include<stdio.h>
void main(){
printf("Hello");
}
Call
Pankaj.c
Pre
processor
extern int
printf (const char*,
...)
void main(){
printf("Hello");
}
call
Pankaj.i
forward
declaration
printf.o
body/code/
imp.
for. dec
pf - call
Pankaj.o
Linker

```
static int i ;
~~static int i ;~~
static int i = 10;
void main(){

    —
    —
    —

}
```

i=200; → Assignment ✗

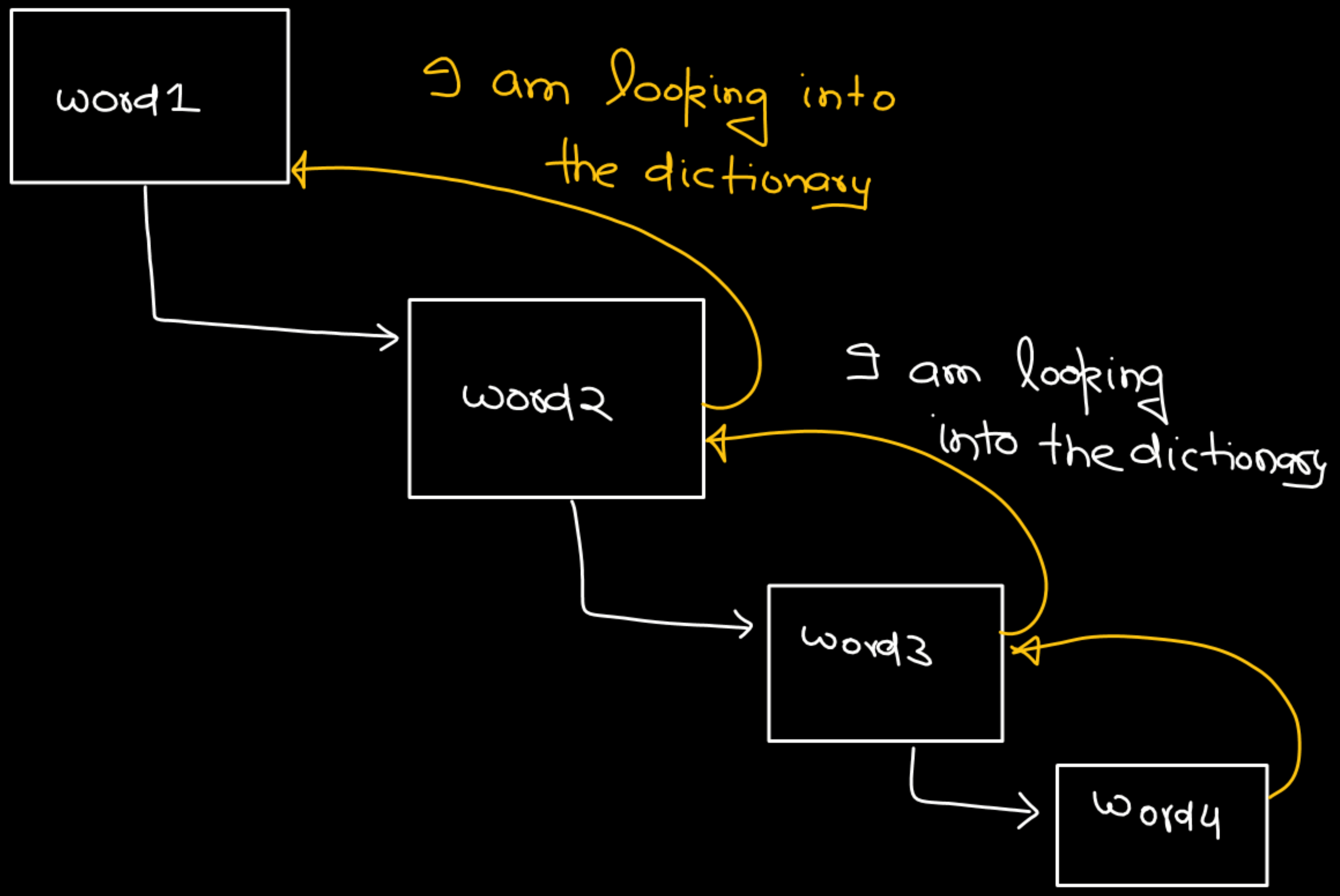
```
void f1(){

}

void f2(){

}
```

```
void main()
{

    Sampad

}
```

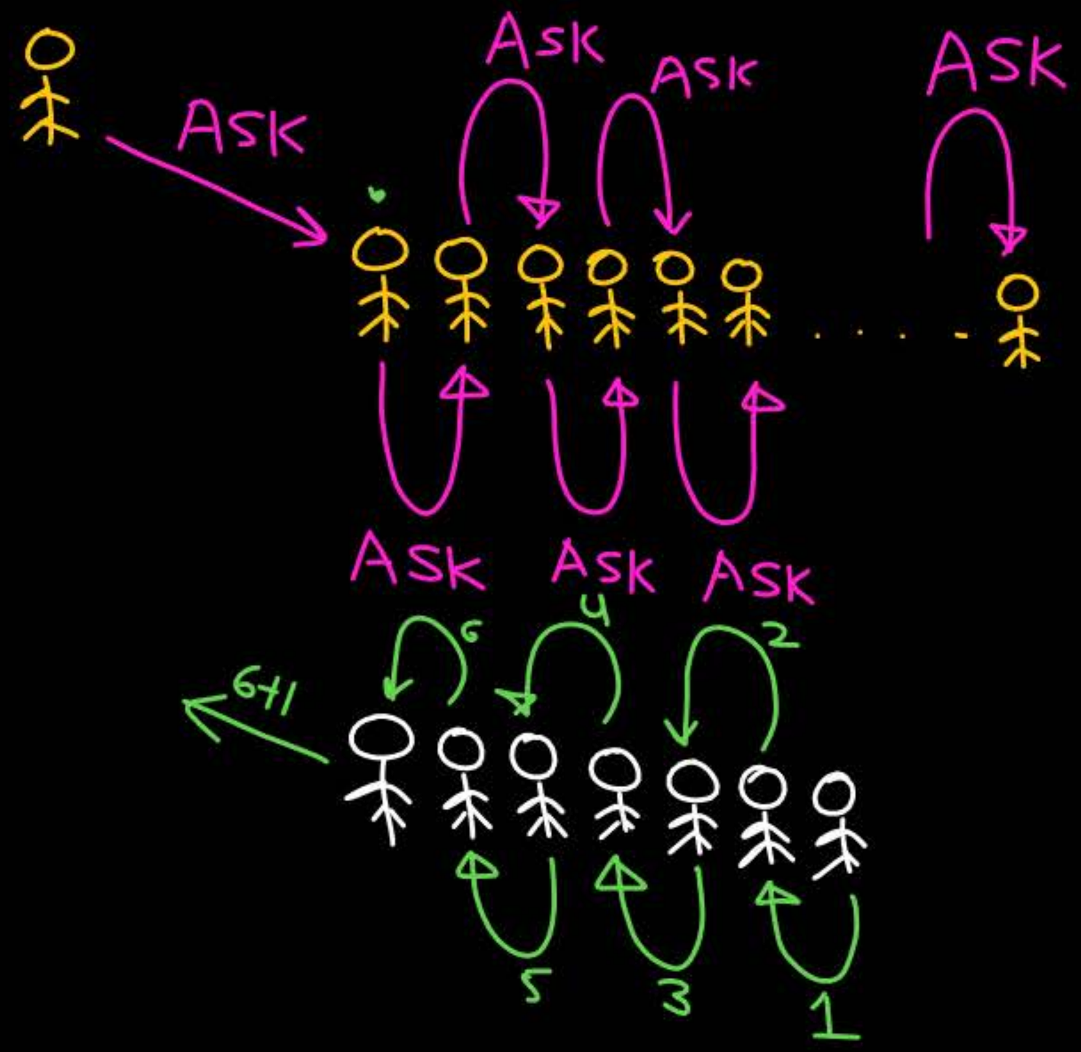
Recursion

Ex1:

Cyborg


word1
word2
word3
word4
I am looking into the dictionary
I am looking into the dictionary

Ex2:



What if only 1 student was there ?

input size is small

- → No recursion is needed
- → we can answer directy
- → Easy case

input is large

- → Hard to solve
- → Can not be onswer directly
- → Recursion is needed

Atleast 2 cases are there in recursion

```
if ( n is small)
    {
        → No recursion is needed
        → Easy case
        → can be answered directly
    }
else {
        // input is large
        * Hard case
        * Can not be answer directly
        * Rerursion is needed
    }
```

Printing Pankaj n times

Ex1 $n \geq 1$ ✓

i/p : 2

o/p : PankajPankaj

i/p : 4

o/p : PankajPankajPankajPankaj

i/p : 8

o/p : PankajPankajPankajPankaj PankajPankajPankaj Pankaj

```
void   Print(int n)
       {

       }
```

↑

printing
Pankaj
n
times

↓

↑

printing
Pankaj
n
times

↓

```
void    Print(int n)
        {
            if(n is small)
                {
               → can be answer directly
               → No rec. is needed
               → Easy case
                }
            else {
                       // large n
                       // Recursion is needed
                 }
        }
```

printing
Pankaj
n
times

```
void   Print(int n)
       {
           if(  n==1  )
                {
                   printf("Pankaj");
                   return ;

                }

           else {
                       // large n
                       // Recursion is needed


                 }
       }
```

↑

printing Pankaj n times

↓

```
void Print(int n)
{
        if( n==1 )
            {
                printf("Pankaj");
                return ;
            }

        else {
                                // large n
                                // Recursion is needed
        In every rec. call, some task/small unit of
        work is performed and rest is left for rec.
             }
}
```

printing
Pankaj
n
times

```
void    Print(int n)
        {
            if(  n==1  )
                 {
                     printf("Pankaj");

                 }

            else  {
               1.    printf("Pankaj");
               2.  Print(n-1);

                  }
        }
```

$n > 0$

i/p : $n = 125$

o/p : 8

i/p : $n = 32$

o/p : 5

i/p : 2397

o/p : 21

i/p : 3 ✓

o/p : 3

i/p : 9 ✓

o/p : 9

$d_1 + d_2 + d_3 + d_4$

```
int   sum_of_digit( int  n )
    {
          if(n is small)
              {

          → No  rec.  is  needed
          → Can  be  answer  directly

              }
         else  {

               Rec.  is needed


               }

    }
```

```
int   sum_of_digit( int n )
   {
        if( n>0 && n<=9 )

             return n ;

        else

     return n%10 + sum_of_digit(n/10);
   }
```

↑

recursion use

↓

$n = 1267$

$\text{sum\_of\_digit}(1267)$

$= \boxed{7} + \boxed{\text{sum\_of\_digit}(126)}$

$n\%10 +$

*** Recursion : n size

Always assume that we know the answer of small size problem

less than

n-1

n-2

a^b $b >= 0$

$a > 0$

a, b

3^{100}

$3 \times 3 \times 3 \times \ldots$

```
if( b is small )
{
```

```
int    Power(int a, int b)
       {
           if(b==0)
               return 1;
           else {

               return    a × Power(a,b-1);

           }
       }
```

Recursion

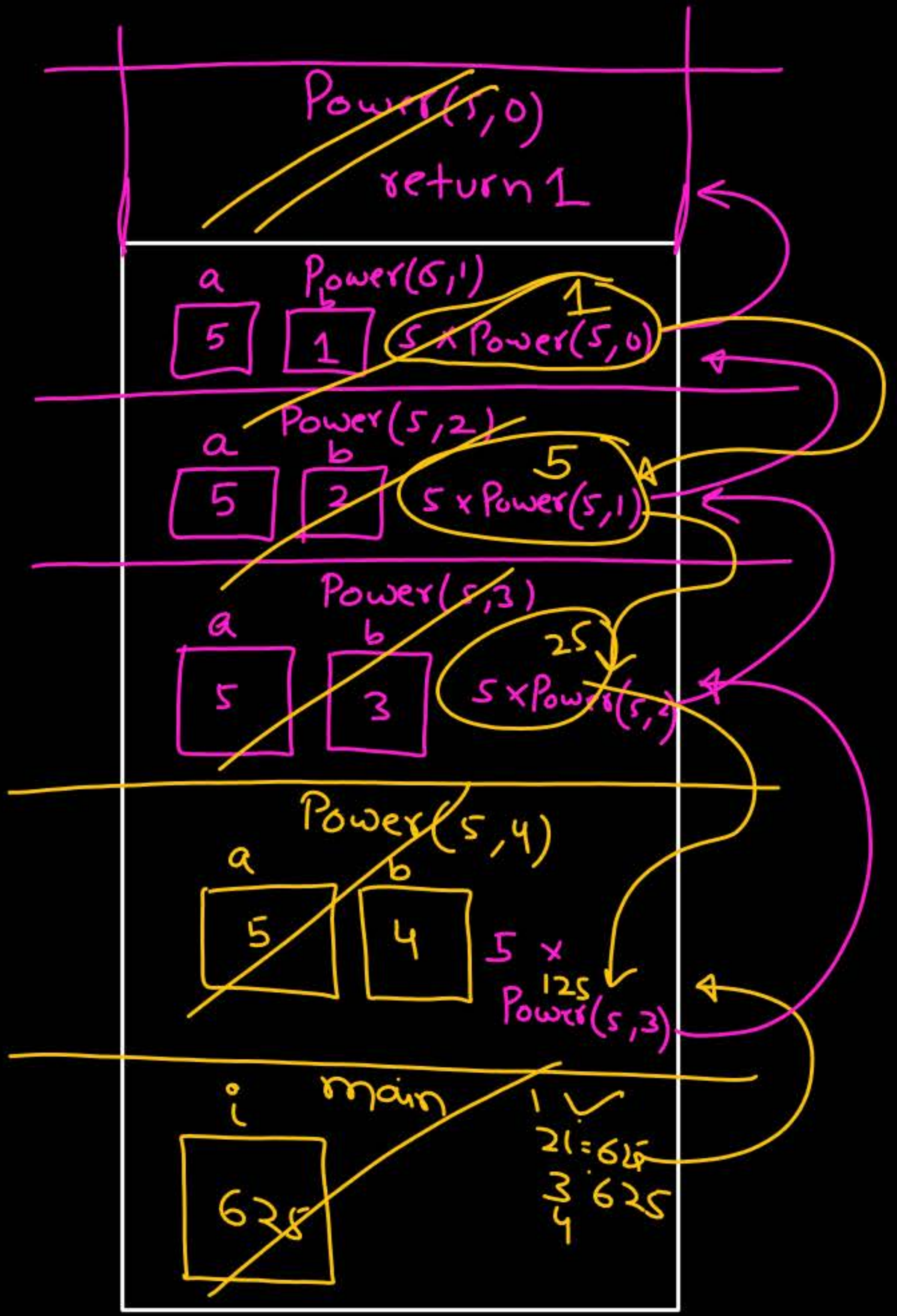
$$3^{100} = 3 \times 3 \times \cdot \;\_\_$$

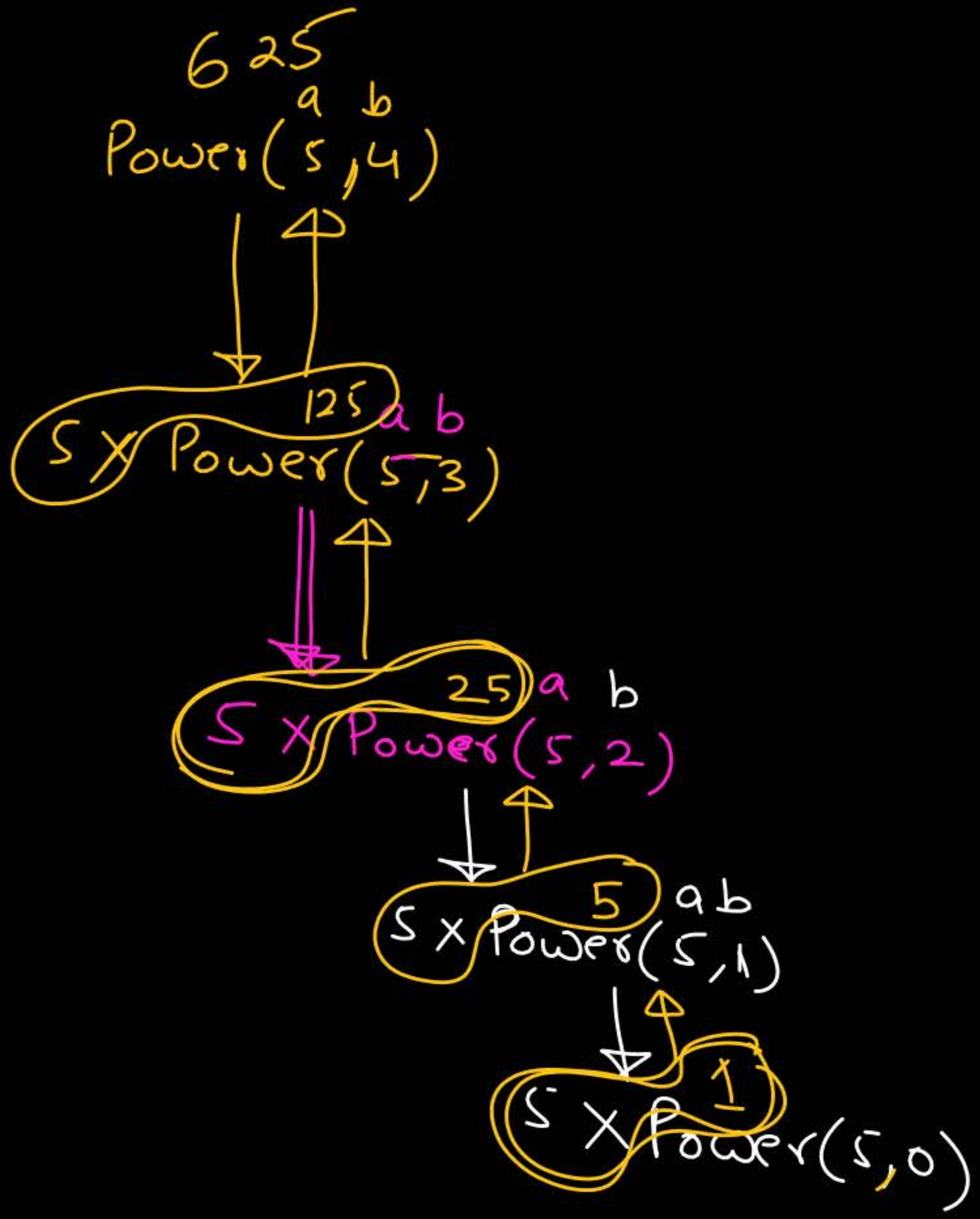
$$3^{100} = 3 \times \boxed{3^{99}}$$

$$\downarrow$$

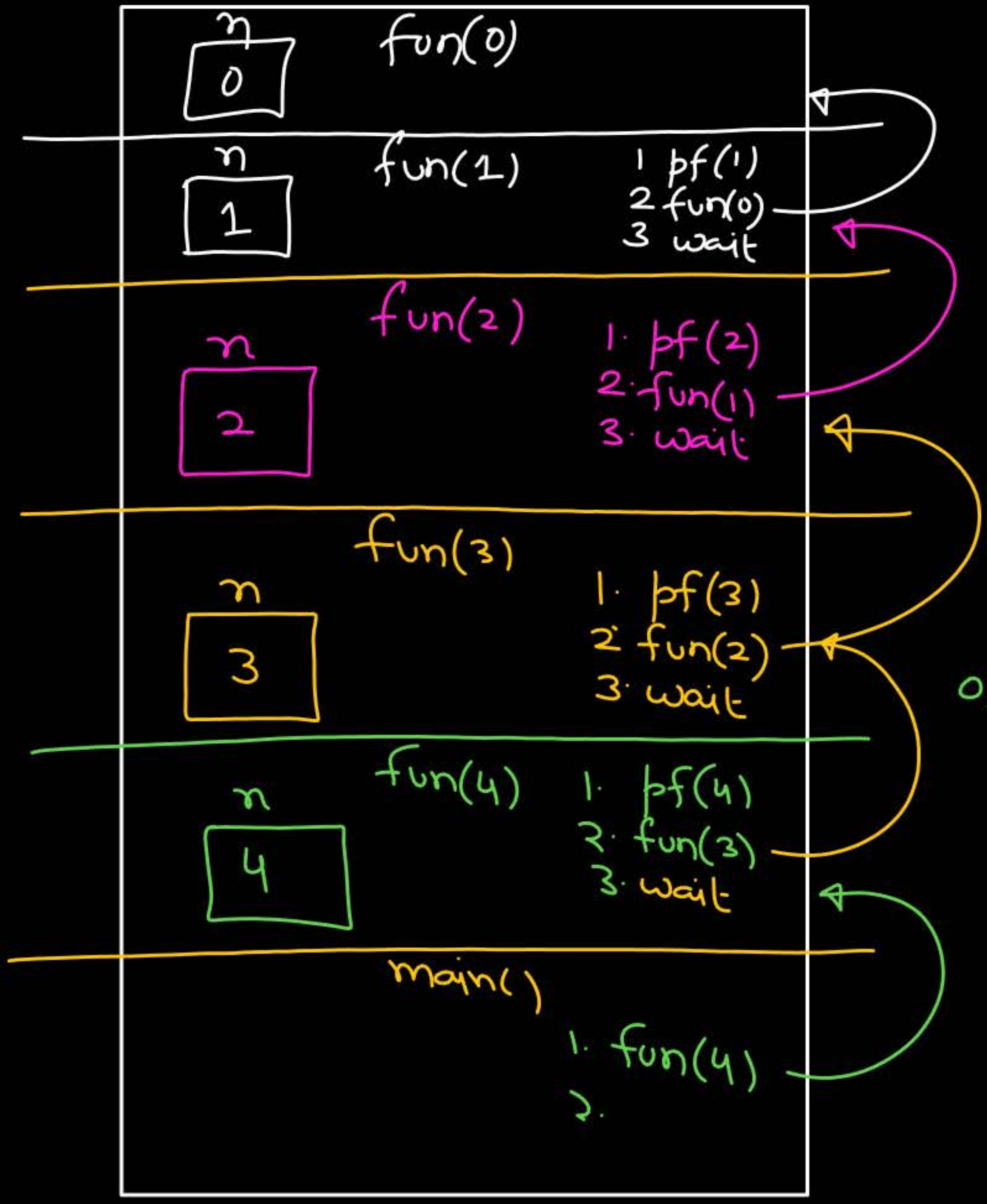
$$3 \times (\qquad)$$

```
int    Power(int a, int b)
       {
            if(b==0)
                 return 1;
            else {


       return    a × Power(a,b-1);

                 }
       }
void main(){
     1. int i;
     2. i = Power(5,4);
     3 printf("%d", i);
     4.   }
```



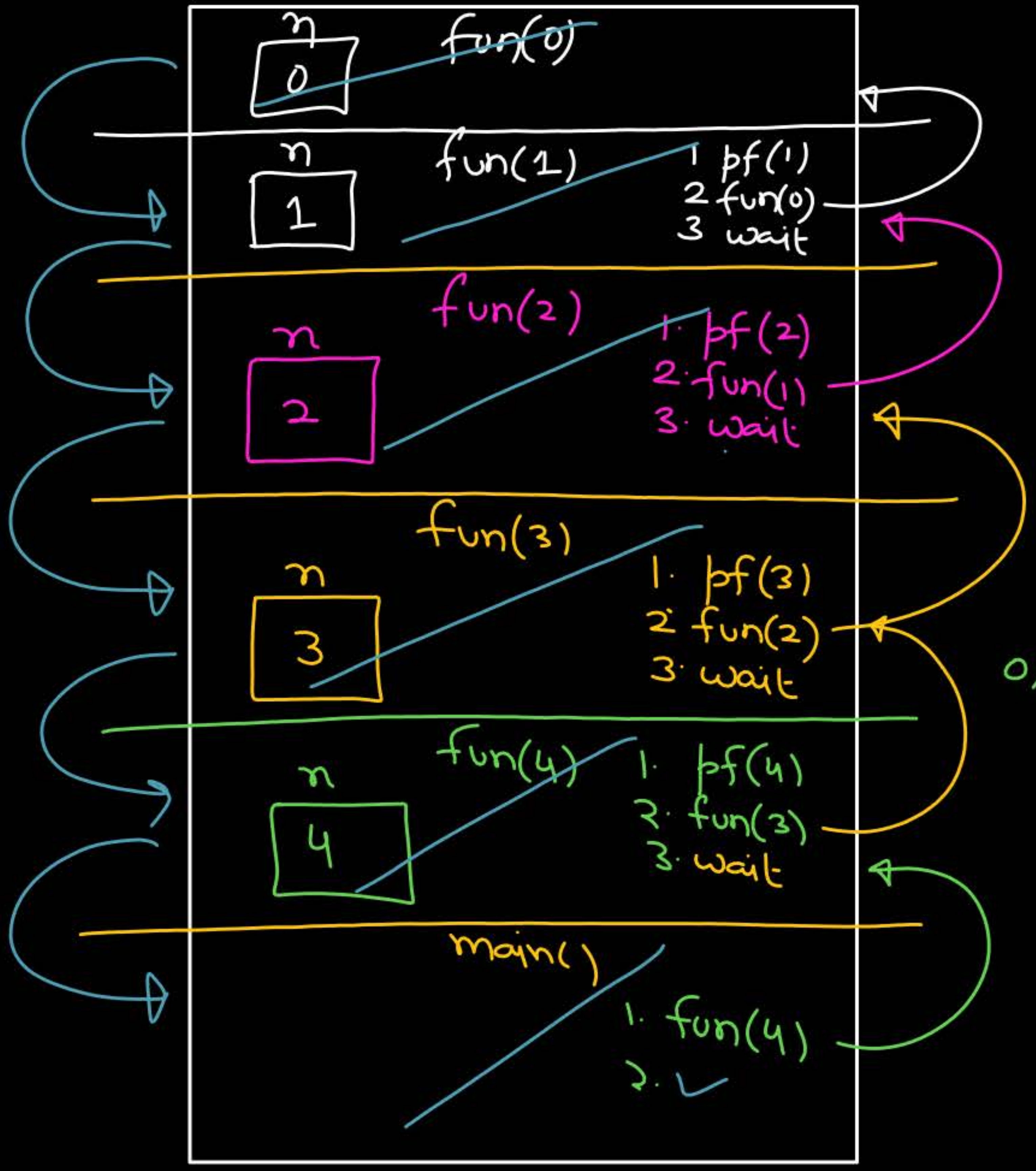

625
a b
Power(5,4)
125 a b
5 x Power(5,3)
25 a b
5 x Power(5,2)
5 a b
5 x Power(5,1)
1
5 x Power(5,0)

```
                     4
void   fun(int n)
       {
          if(n<=0)
               return ;

          else {

     1.      printf("%d",n);

     2.      fun(n-1);

     3.      }

       }
   void main(){

          1.   fun(4);

          2.   }
```



o/p: 4 3 2 1

```
                 4
void   fun(int n)
       {
          if(n<=0)
              return;

          else {

       1.     printf("%d",n);

       2.     fun(n-1);

       3.     }

       }
   void main(){
          1.  fun(4);
          2.  }
```



o/p: 4 3 2 1

```
                         4
void  fun(int n)
      {
         if(n<=0)
             return;

         else {
         1.      fun(n-1);
         2.  printf("%d",n);
         3.  }
      }
   void main(){
        1.  fun(4);
        2.   }
```

main

↓

fun(4), pf wait

↓

fun(3), pf wait

↓

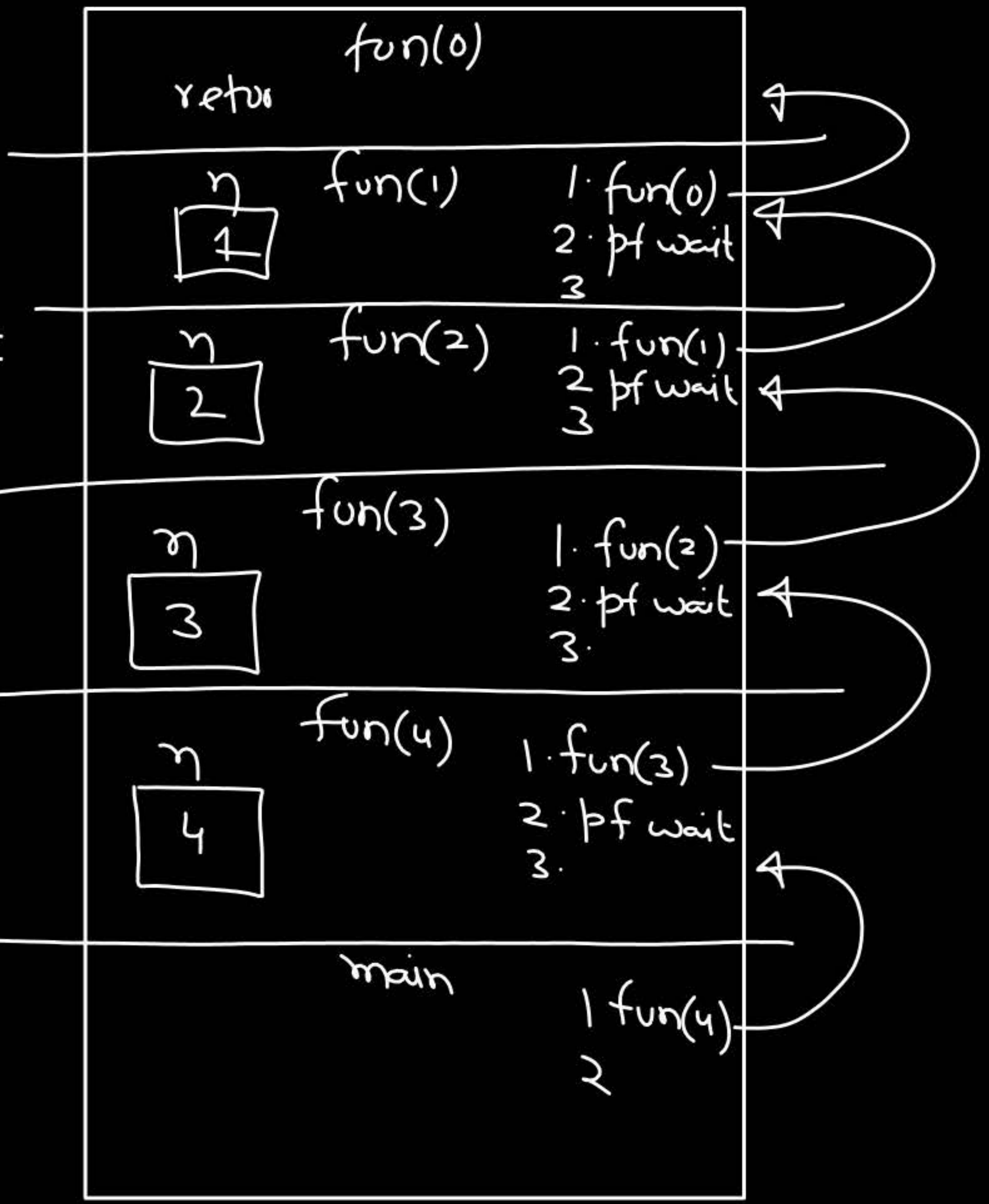
fun(2), pf wait

↓

fun(1), pf wait

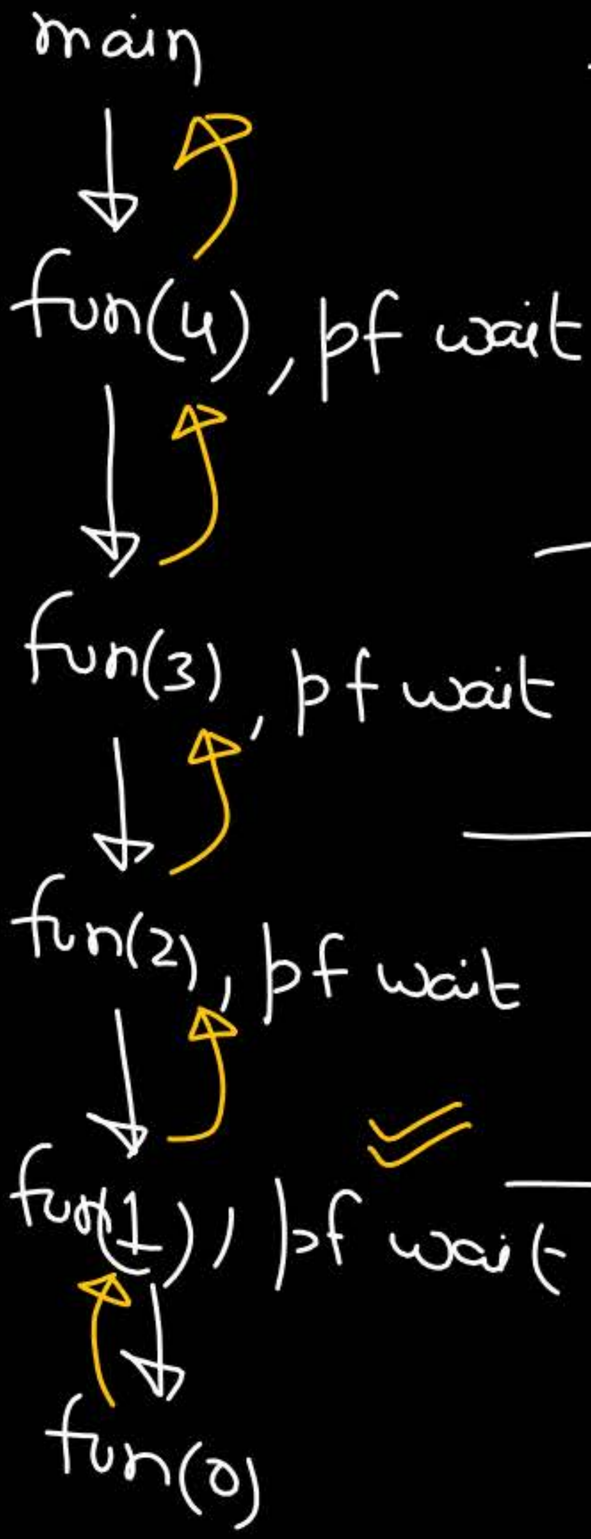
↓

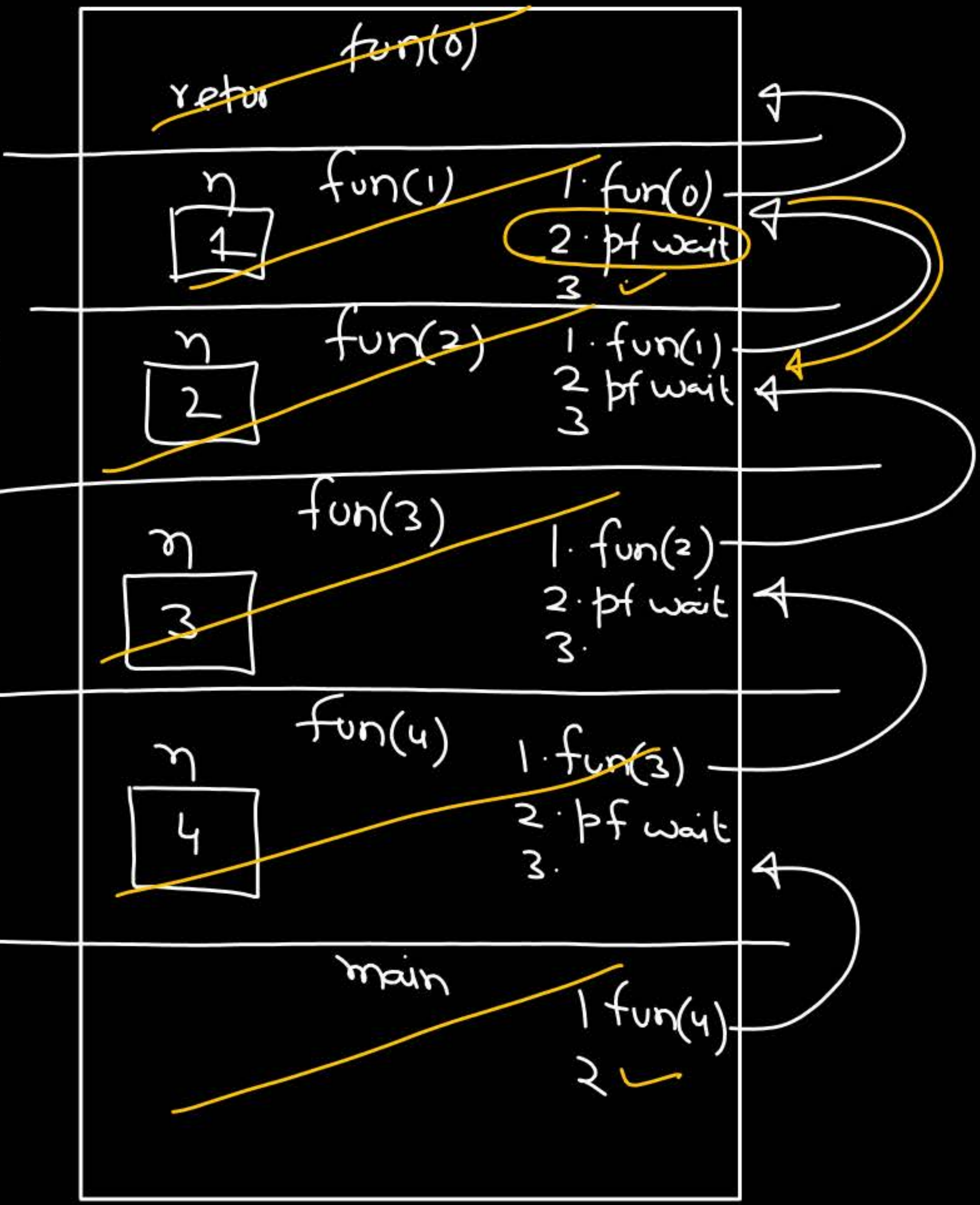
fun(0)



```
                   4
void   fun(int n)
       {
         if(n<=0)
             return ;

         else {
          1.       fun(n-1);
          2    printf("%d",n);
          3.   }
       }
  void  main(){
          1.  fun(4);
          2.    }
```

1 2 3 4





main → fun(4) → fun(3) → fun(2) → fun(1)

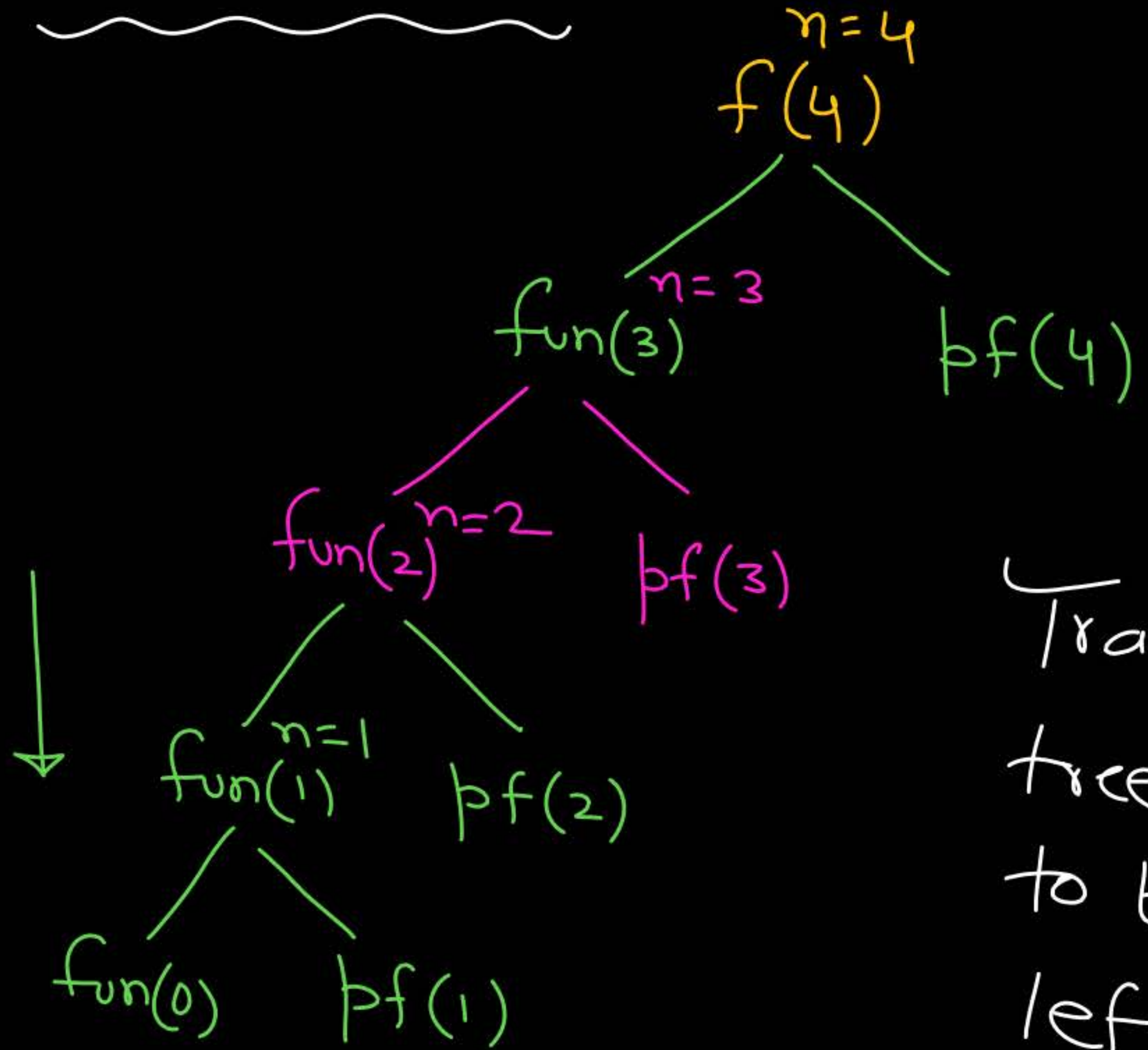
4 ← 3 ← 2 ← 1

pf is written after rec. call

Statements writter after recursive call executes in opposite order of call.

Recursion tree

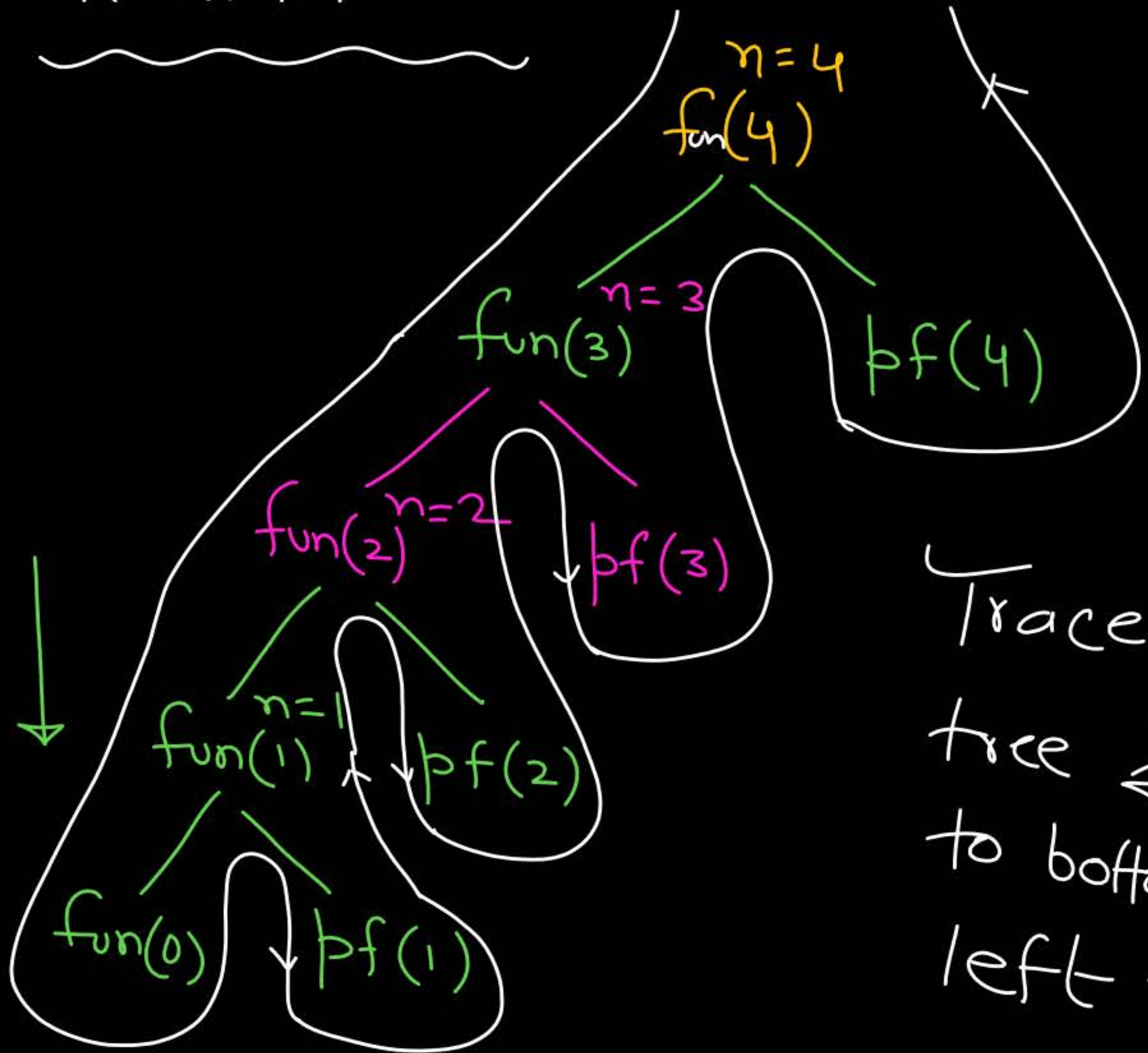
```
void fun(int n)
  {
   if(n<=0)
        return ;

   else {
   1. fun(n-1);
   2. printf("%d",n);
      }
  }
void main(){
        f(4);
       }
```



Trace this tree from top to bottom and left to right

Recursion tree

```
void fun(int n)
   {
    if(n<=0)
        return ;

    else {
    1. fun(n-1);
    2. printf("%d",n);
       }
   }
void main(){
        f(4);
       }
```

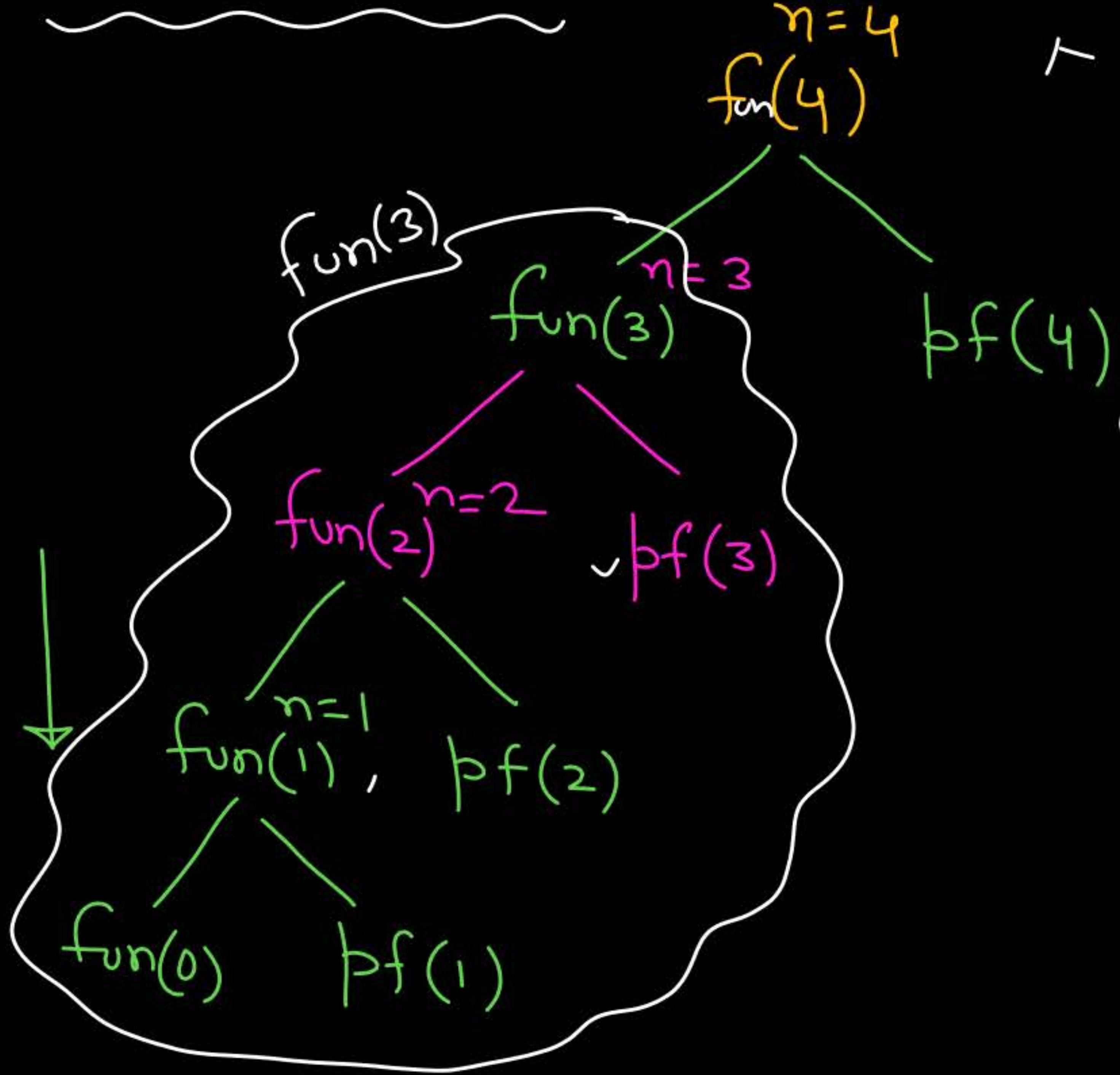


1 2 3 4

Trace this tree from top to bottom and left to right

Recursion tree

```
void fun(int n)
   {
    if(n<=0)
        return ;

    else {
    1. fun(n-1);
    2. printf("%d",n);
       }
   }
void main(){
       f(4);
      }
```

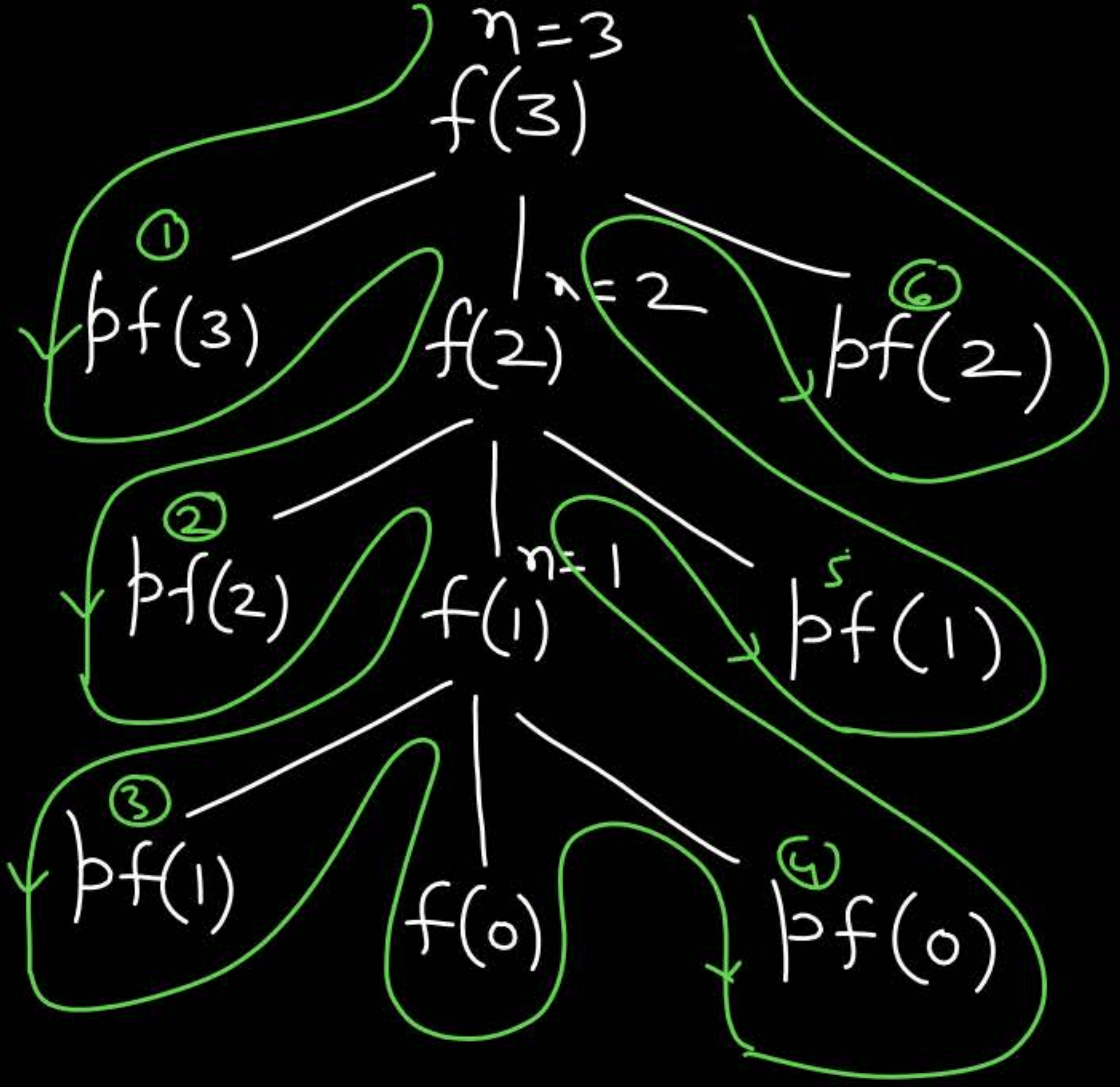


this will execute only when fun(3) finish its execution

1 2 3 4

1.

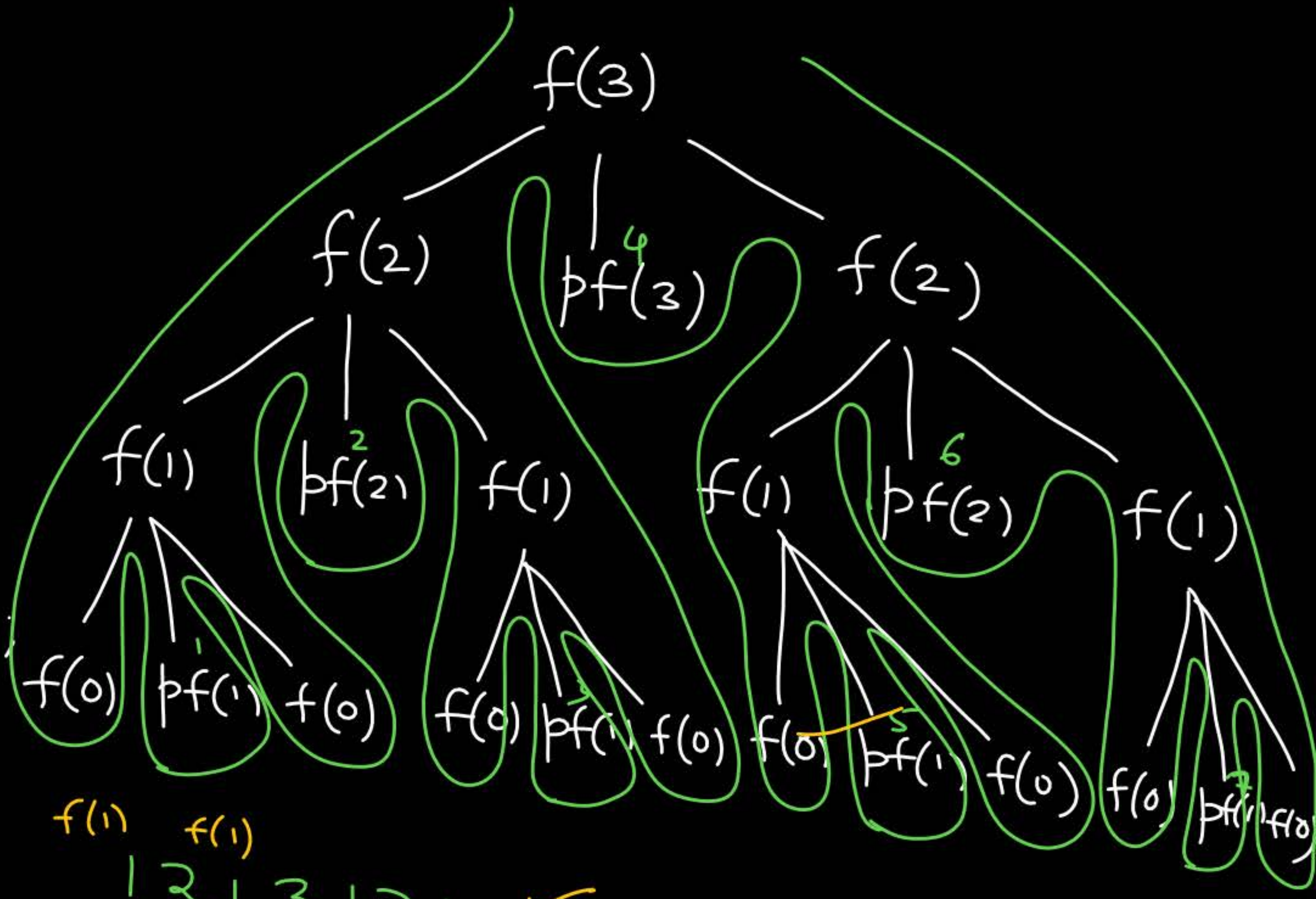
```
void f(int n){
        if(n<=0)
            return
        else{
1.      pf("%d",n);
2.      f(n-1);
3.      pf("%d",n-1);
        }
    }
void main(){
        f(3);
    }
```



321012

1.

```
void f(int n){
        if(n<=0)
            return
        else{
1   ✓  f(n-1);
2   printf("%d",n);
3   ✓  f(n-1);
        }
    }
void main(){
        f(3);
    }
```



$\underbrace{\overset{f(1)}{1}\,2\,\overset{f(1)}{1}}_{f(2)}\;3\;\underbrace{1\,2\,1}_{f(2)}$ ✓

THANK YOU GW SOLDIERS !